ESTD. 2001

PRATHYUSHA ENGINEERING COLLEGE

DEPARTMENT OF INFORMATION TECHNOLOGY
E CONTENT DEVELOPED BY THE FACULTY

S.NO	COURSE CODE	COURSE NAME	Name of the Faculty
1	CS6401	Operating Systems	M.D.Boomija
2	CS6703	Grid and Cloud Computing	M.D.Boomija
3	CS8591	COMPUTER NETWORKS	A.Subbarayudu/A.P
4	CS8591	COMPUTER NETWORKS	Ms B.S.Liya/A.P
5	CS8792	Crypography and Network Security	Ms B.S.Liya/A.P
6	CS6601	Distributed Systems	M.D.Boomija

CS6401-OPERATING SYSTEMS
UNIT-I COMPUTER SYSTEM OVERVIEW

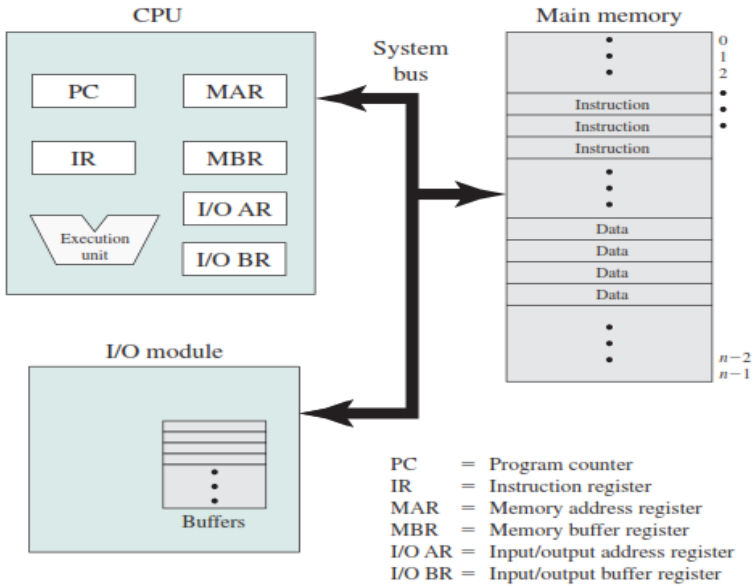
Computer System Overview-Basic Elements, Instruction Execution, Interrupts, Memory Hierarchy, Cache Memory, Direct Memory Access, Multiprocessor and Multicore Organization. Operating system overview-objectives and functions, Evolution of Operating System.- Computer System Organization-Operating System Structure and Operations- System Calls, System Programs, OS Generation and System Boot.

BASIC ELEMENTS OF A COMPUTER.

➔At a top level, a computer consists of processor, memory, and I/O components, with one or more modules of each type. These components are interconnected in some fashion to achieve the main function of the computer, which is to execute programs.

Thus, there are four main structural elements:

- **Processor:** Controls the operation of the computer and performs its data processing functions. When there is only one processor, it is often referred to as the central processing unit (CPU).
- **Main memory:** Stores data and programs. This memory is typically volatile; that is, when the computer is shut down, the contents of the memory are lost. In contrast, the contents of disk memory are retained even when the computer system is shut down. Main memory is also referred to as real memory or primary memory.
- **I/O modules:** Move data between the computer and its external environment. The external environment consists of a variety of devices, including secondary memory devices (e.g., disks), communications equipment, and terminals.
- **System bus:** Provides for communication among processors, main memory, and I/O modules.



Computer Components: Top-Level View

➔The figure depicts these top - level components. One of the processor's functions is to exchange data with memory. For this purpose, it typically makes use of two internal (to the processor) registers: a memory address register (MAR), which specifies the address in memory for the next read or write; and a memory buffer register (MBR), which contains the data to be written into memory or which receives the data read from memory.

➔Similarly, an I/O address register (I/OAR) specifies a particular I/O device. An I/O buffer register (I/OBR) is used for the exchange of data between an I/O module and the processor.

➔A memory module consists of a set of locations, defined by sequentially numbered addresses. Each location contains a bit pattern that can be interpreted as either an instruction or data. An I/O module transfers data from external devices to processor and memory, and vice versa. It contains internal buffers for temporarily holding data until they can be sent on.

INSTRUCTION EXECUTION WITH INSTRUCTION EXECUTION CYCLE.

➔A program to be executed by a processor consists of a set of instructions stored in memory. In its simplest form, instruction processing consists of two steps:

➔The processor reads (fetches) instructions from memory one at a time and executes each instruction. Program execution consists of repeating the process of instruction fetch and instruction execution.

➔The processing required for a single instruction is called an instruction cycle. Using a simplified two-step description, the instruction cycle is depicted in Figure.

The two steps are referred to as the

(i)Fetch stage (ii)Execution stage.

➔Program execution halts only if the processor is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the processor is encountered.

➔The program counter (PC) holds the address of the next instruction to be fetched. Unless instructed otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.

➔For example, consider a simplified computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to location 300. The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on. This sequence may be altered, as explained subsequently.

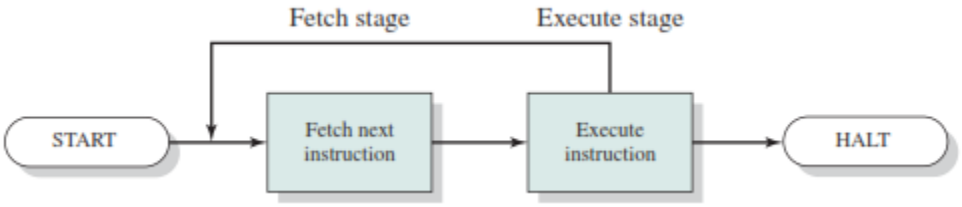
➔The fetched instruction is loaded into the instruction register (IR). The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required action.

➔In general, these actions fall into four categories:

- **Processor-memory**: Data may be transferred from processor to memory or from memory to processor.
- **Processor-I/O**: Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- **Data processing**: The processor may perform some arithmetic or logic operation on data.
- **Control:** An instruction may specify that the sequence of execution be altered. For example, the processor may fetch an instruction from location 149, which specifies that the next instruction will be from location 182. The processor sets the program counter to

182. Thus, on the next fetch stage, the instruction will be fetched from location 182 rather than 150.

Basic Instruction Cycle



(a) Instruction format

(b) Integer format

Program counter (PC) = Address of instruction
Instruction register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

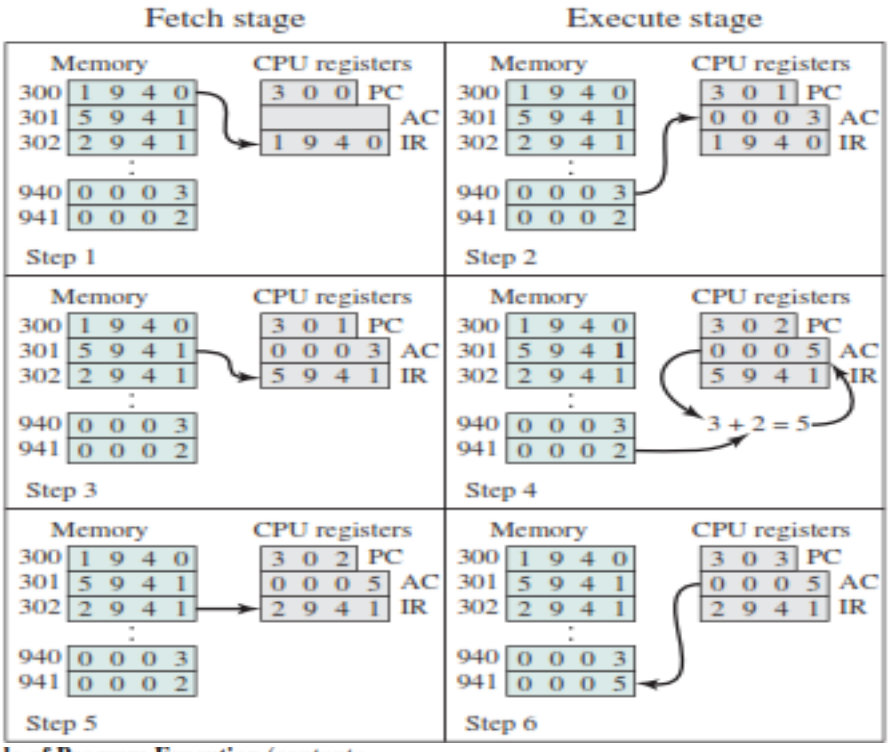
0001 = Load AC from memory
0010 = Store AC to memory
0101 = Add to AC from memory

(d) Partial list of opcodes

➔Figure shows a partial program execution, showing the relevant portions of memory and processor registers. The program fragment shown adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location.

Three instructions, which can be described as three fetch and three execute stages, are required:

1. The PC contains 300, the address of the first instruction. This instruction (the value 1940 in hexadecimal) is loaded into the IR and the PC is incremented. Note that this process involves the use of a memory address register (MAR) and a memory buffer register (MBR). For simplicity, these intermediate registers are not shown.
2. The first 4 bits (first hexadecimal digit) in the IR indicate that the AC is to be loaded from memory. The remaining 12 bits (three hexadecimal digits) specify the address, which is 940.



Example of Program Execution (contents of memory and registers in hexadecimal)

3. The next instruction (5941) is fetched from location 301 and the PC is incremented.
4. The old contents of the AC and the contents of location 941 are added and the result is stored in the AC.
5. The next instruction (2941) is fetched from location 302 and the PC is incremented.
6. The contents of the AC are stored in location 941.

In this example, three instruction cycles, each consisting of a fetch stage and an execute stage, are needed to add the contents of location 940 to the contents of 941. With a more complex set of instructions, fewer instruction cycles would be needed.

INTERRUPT PROCESSING.

➔Virtually all computers provide a mechanism by which other modules (I/O , memory) may interrupt the normal sequence of the processor. Interrupts are provided primarily as a way to improve processor utilization.

Four Classes of Interrupts are

1. **Program** Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
2. **Timer** Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
3. **I/O** Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
4. **Hardware failure** Generated by a failure, such as power failure or memory parity error.

➔The user program performs a series of WRITE calls interleaved with processing. The solid vertical lines represent segments of code in a program. Code segments 1, 2, and 3 refer to sequences of instructions that do not involve I/O. The WRITE calls are to an I/O routine that is a system utility and that will perform the actual I/O operation.

The I/O program consists of three sections:

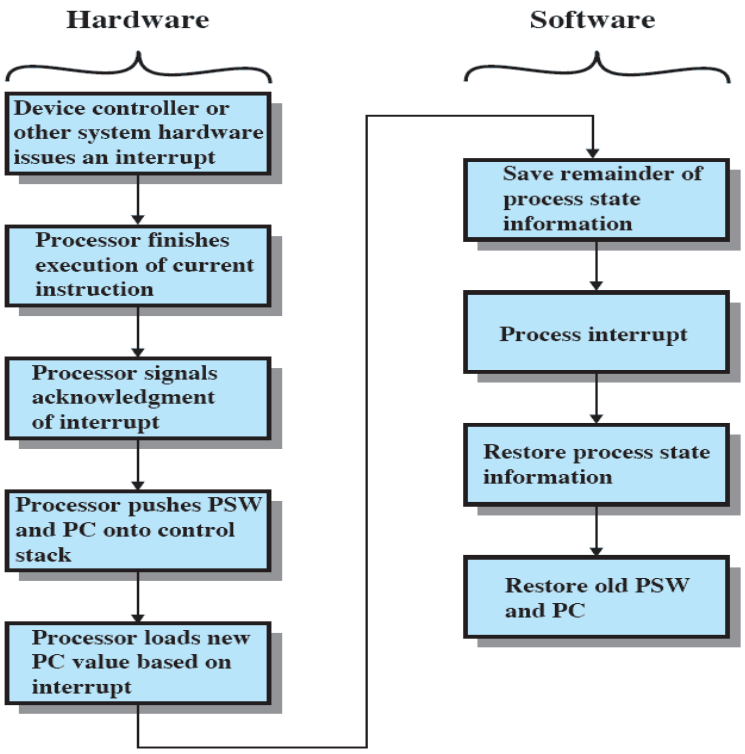
• A sequence of instructions, labeled 4 in the figure, to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command.

• The actual I/O command. Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function (or periodically check the status, or poll, the I/O device). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.

• A sequence of instructions, labeled 5 in the figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.

INTERRUPT PROCESSING

The following gives the detailed interrupt processing procedure:



➔An interrupt triggers a number of events, both in the processor hardware and in software.

This figure shows a typical sequence. When an I/O device completes an I/O operation, the following sequence of hardware events occurs:

1. The device issues an interrupt signal to the processor.
2. The processor finishes execution of the current instruction before responding to the interrupt.
3. The processor tests for a pending interrupt request, determines that there is one, and sends an acknowledgment signal to the device that issued the interrupt. The acknowledgment allows the device to remove its interrupt signal.
4. The processor next needs to prepare to transfer control to the interrupt routine.
5. The processor then loads the program counter with the entry location of the interrupt- handling routine that will respond to this interrupt.
6. At this point, the program counter and PSW relating to the interrupted program have been saved on the control stack.
7. The interrupt handler may now proceed to process the interrupt.
8. The saved register values are retrieved from the stack and restored to the registers
9. The final act is to restore the PSW and program counter values from the stack.
It is important to save all of the state information about the interrupted program for later resumption.

Because the interrupt is not a routine called from the program.
Rather, the interrupt can occur at any time and therefore at any point in the execution of a user program.
Its occurrence is unpredictable.

MULTIPLE INTERRUPTS

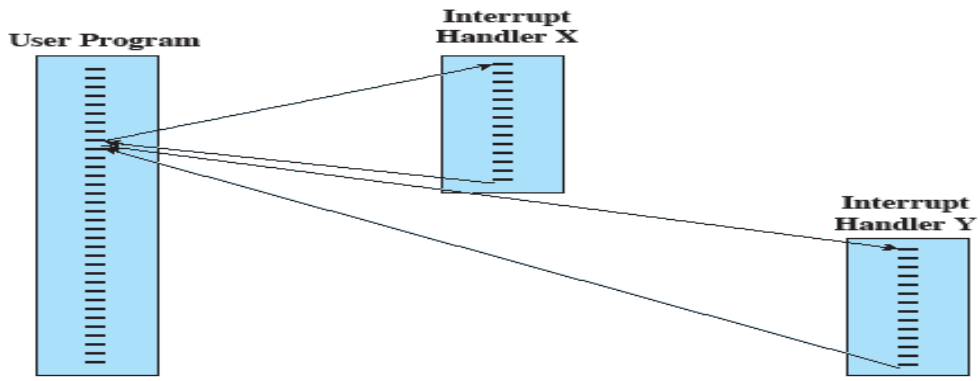
➔The above only discussed the case in which a single interrupt happens. Actually, in a computer system, there are multiple interrupt signal sources, so more than one interrupt requests may happen at the same time or during a same period.

➔The typical two approaches are: sequential interrupt processing - by disabling interrupt request while an interrupt is being processed, all interrupts will be processed sequentially (usually PSW contains a bit for this purpose); nested interrupt processing - all the interrupts may be assigned different priorities, so that whenever an interrupt occurs while an interrupt handler is running, their priorities will be compared first, and the further action will be determined according to the result. These two approaches are illustrated by the following figures:

a) Sequential Interrupt Processing

➔Two approaches can be taken to dealing with multiple interrupts. The first is to disable interrupts while an interrupt is being processed. A *disabled interrupt* simply means that the processor ignores any new interrupt request signal. If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has re-enabled interrupts.

Thus if an interrupt occurs when a user program is executing, then interrupts are disabled immediately. After the interrupt-handler routine completes, interrupts are re-enabled before resuming the user program and the processor checks to see if additional interrupts have occurred. This approach is simple, as interrupts are handled in strict sequential order

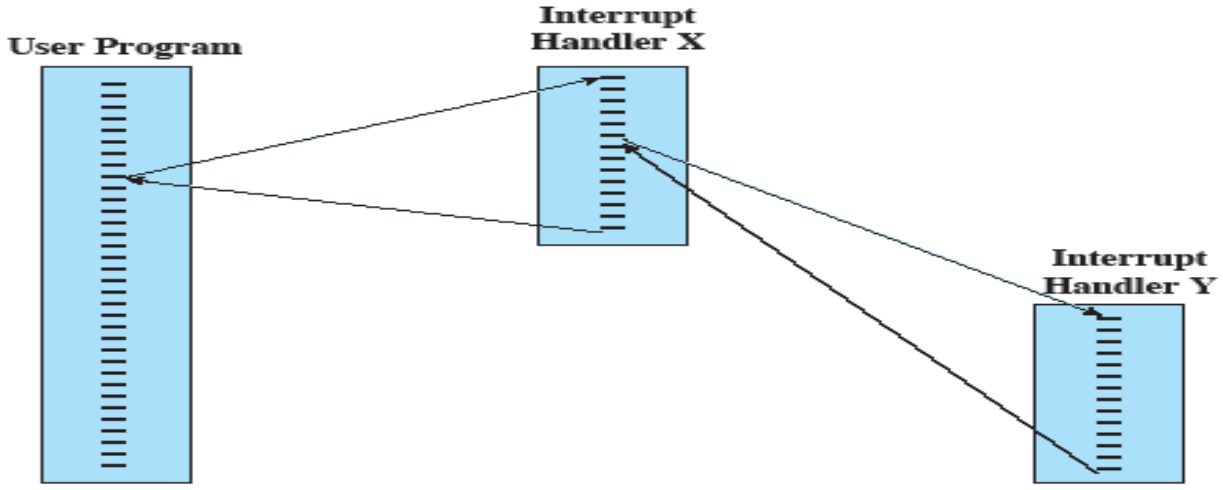


(a) Sequential interrupt processing

The drawback of sequential approach is that it does not take into account relative priority or time-critical needs.

b) Nested Interrupt Processing

A second approach is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to be interrupted.



(b) Nested interrupt processing

As an example of this second approach, consider a system with three I/O devices:

- a printer (priority 2),
- a disk (priority 4), and
- a communications line (priority 5).

This figure illustrates a possible sequence.

1. A user program begins at *t*= 0.
2. At *t*=10, a printer interrupt occurs;

   user information is placed on the control stack and execution continues at the printer interrupt service routine (ISR).

3. While this routine is still executing, at t =15 a communications interrupt occurs.

   Because the communications line has higher priority than the printer, the interrupt request is honored.

4. The printer ISR is interrupted, its state is pushed onto the stack, and execution continues at the communications ISR.

5. While this routine is executing, a disk interrupt occurs (t=20).

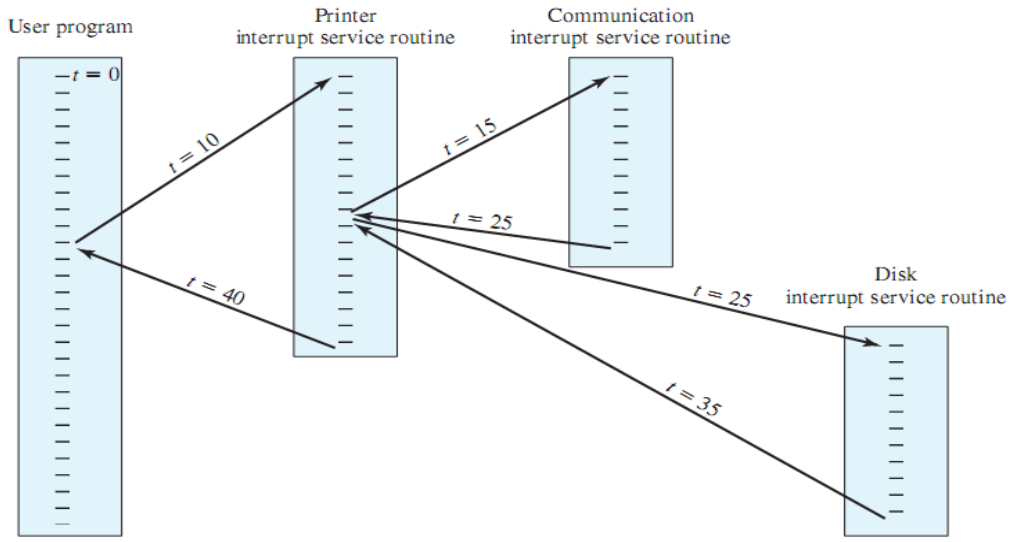
   Because this interrupt is of lower priority, it is simply held, and the communications ISR runs to completion.

6. When the communications ISR is complete (t=25), the previous processor state is restored, which is the execution of the printer ISR.

7. However, before even a single instruction in that routine can be executed, the processor honors the higher-priority disk interrupt and transfers control to the disk ISR.

8. Only when that routine is complete (t=35) is the printer ISR resumed.

9. When that routine completes (t=40), control finally returns to the user program.



MEMORY HIERARCHY.

Memory Hierarchy

➔The memory unit is an essential component in any digital computer since it is needed for storing programs and data.

➔Not all accumulated information is needed by the CPU at the same time. Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU.

➔Computer Memory Hierarchy is a pyramid structure that is commonly used to illustrate the significant differences among memory types.

➔The memory unit that directly communicates with CPU is called the main memory. Devices that provide backup storage is called auxiliary memory.

➔The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory

Performance

Access time —Time between presenting the address and getting the valid data

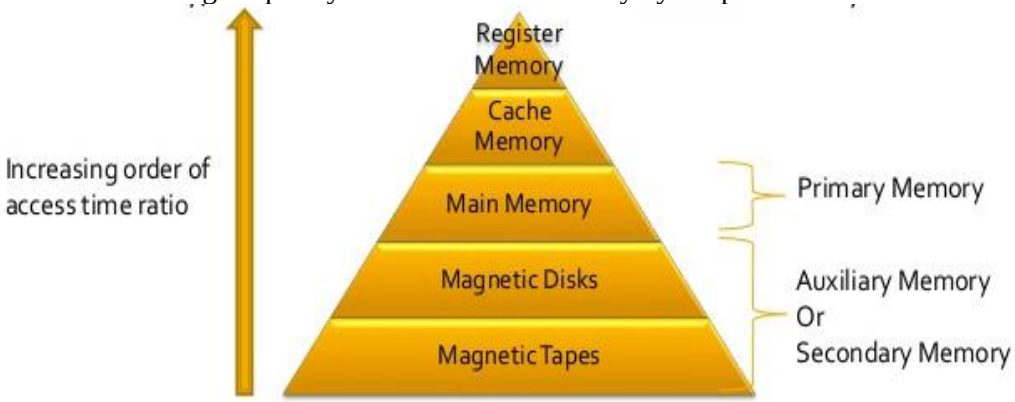
Memory Cycle time —Time may be required for the memory to "recover" before next access

—Cycle time is access + recovery

Transfer Rate —Rate at which data can be moved

Going down the hierarchy

– Decreasing cost per bit
– Increasing capacity
– Increasing access time
– Decreasing frequency of access to the memory by the processor



Main Memory

➔Most of the main memory in a general purpose computer is made up of RAM integrated circuits chips, but a portion of the memory may be constructed with ROM chips

1. RAM– Random Access memory
2. ROM– Read Only memory

RAM

A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip when needed.

Key features

RAM is packaged as a chip.
Basic storage unit is a cell (one bit per cell).
Multiple RAM chips form a memory.

Static RAM (SRAM)

Each cell stores bit with a six-transistor circuit.
Retains value indefinitely, as long as it is kept powered.
Relatively insensitive to disturbances such as electrical noise.
Faster and more expensive than DRAM.

Dynamic RAM (DRAM)

Each cell stores bit with a capacitor and transistor.
Value must be refreshed every 10-100 ms.

Sensitive to disturbances.
Slower and cheaper than SRAM.

ROM

➔ROM is used for storing programs that are **PERMENTLY** resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.

➔The ROM portion of main memory is needed for storing an initial program called *bootstrap loader*, which is to start the computer software operating when power is turned off.

➔Data is programmed into the chip using an external ROM programmer
The programmed chip is used as a component into the circuit
The circuit doesn't change the content of the ROM

Auxiliary Memory

➔Auxiliary memory, also known as auxiliary storage, secondary storage, secondary memory or external memory, is a non-volatile memory (does not lose stored data when the device is powered down) that is not directly accessible by the CPU, because it is not accessed via the input/output channels (it is an external device).

➔Some examples of auxiliary memory would be disks, external hard drives, USB drives, etc.

Cache Memory

➔Cache memory, also called CPU memory, is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. This memory is typically integrated directly with the CPU chip or placed on a separate chip that has a separate bus interconnect with the CPU.

➔The basic purpose of cache memory is to store program instructions that are frequently re- referenced by software during operation. Fast access to these instructions increases the overall speed of the software program.

➔As the microprocessor processes data, it looks first in the cache memory; if it finds the instructions there (from a previous reading of data), it does not have to do a more time-consuming reading of data from larger memory or other data storage devices.

Tertiary Storage

➔Tertiary Storage, also known as tertiary memory, consists of anywhere from one to several storage drives. It is a comprehensive computer storage system that is usually very slow, so it is usually used to archive data that is not accessed frequently. A computer can access tertiary storage without being told to do so, which is unlike off-line storage.

➔This type of computer storage device is not as popular as the other two storage device types. Its main use is for storing data at a very large-scale. This includes optical jukeboxes and tape libraries. Tertiary storage devices require a database to organize the data that are stored in them, and the computer needs to go through the database to access those data.

Memory hierarchy is just like the real world situation where -

1. A train fare is cheaper and it can carry a lot people at a time but it takes long time
2. The air fare of professional flights is more than the train, it can carry lesser number of people but it is much faster than the train
3. The air fare for personal jet is further high, it can carry further lesser number of people but it is fastest of the three.

So, depending upon the price and the urgency to reach destination, you will use combination of these in different situations.

The memory hierarchy is exactly the same. Here, the situation is-

1. We need a lot of memory which is cheap and could be slow (secondary memory, Hard Disk)
2. We also need some memory which could be smaller than secondary memory but should be faster than it (primary memory, RAM)
3. We also need another kind of memory which could be smaller than the primary memory but it should be much faster than it (cache memory).

That's why we need memory hierarchy.

CACHE MEMORY

Concept

Small amount of fastest memory.

Sits between normal main memory and CPU.

May be located on CPU chip or module.

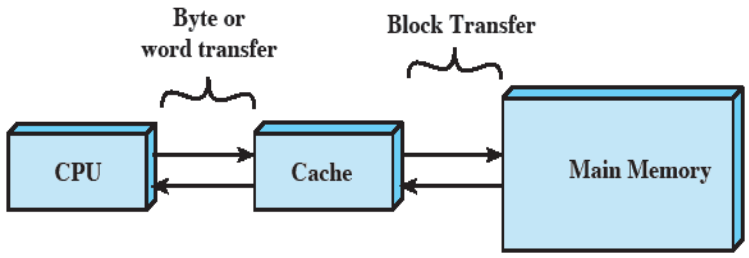
Cache Principles

Contains copy of a portion of main memory

Processor first checks cache

If desired data item not found, relevant block of memory read into cache

Because of locality of reference, it is likely that future memory references are in that block.



Cache Operation

CPU requests contents of memory location.

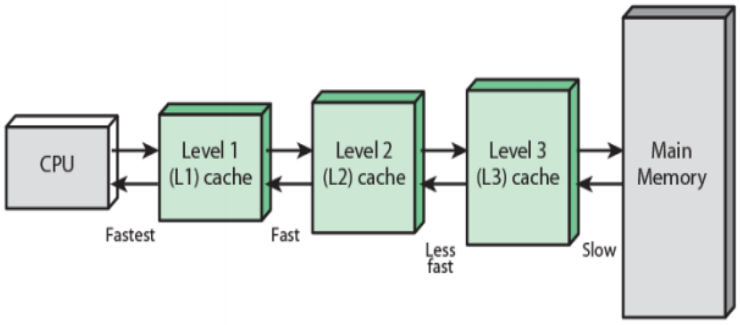
Check cache for this data.

If present, get from cache (fast).

If not present, read required block from main memory to cache.

Then deliver from cache to CPU.

Cache includes tags to identify which block of main memory is in each cache slot.



Three Level Cache Memory Hierarchy

The L3 cache is usually built onto the motherboard between the main memory (RAM) and the L1 and L2 caches of the processor module.

This serves as another bridge to park information like processor commands and frequently used data in order to prevent bottlenecks resulting from the fetching of these data from the main memory.

In short, the L3 cache of today is what the L2 cache was before it got built-in within the processor module itself.

The CPU checks for information it needs from L1 to the L3 cache. If it does not find this info in L1 it looks to L2 then to L3, the biggest yet slowest in the group.

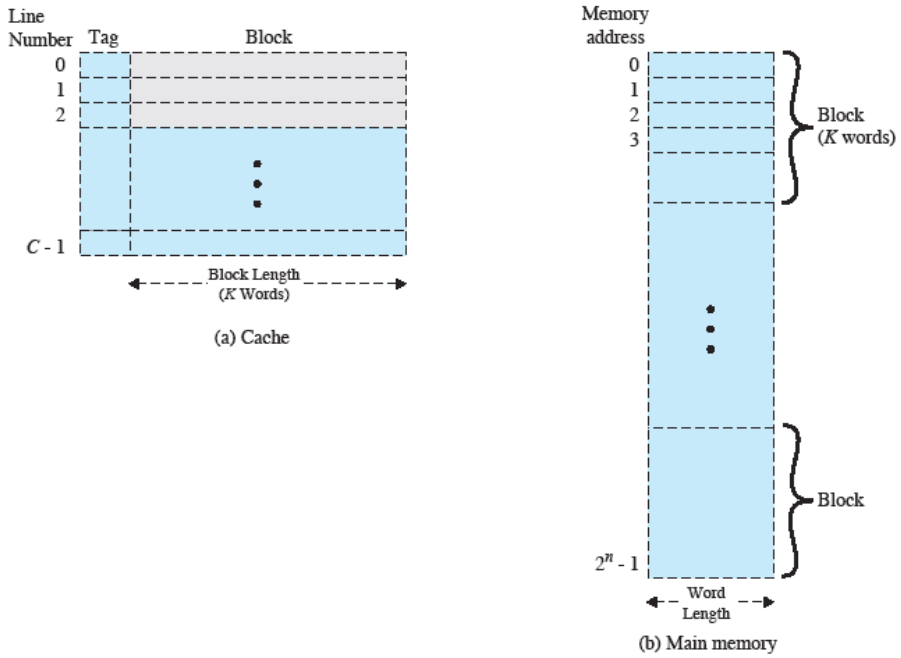
The purpose of the L3 differs depending on the design of the CPU. In some cases the L3 holds copies of instructions frequently used by multiple cores that share it.

Most modern CPUs have built-in L1 and L2 caches per core and share a single L3 cache on the motherboard, while other designs have the L3 on the CPU die itself.

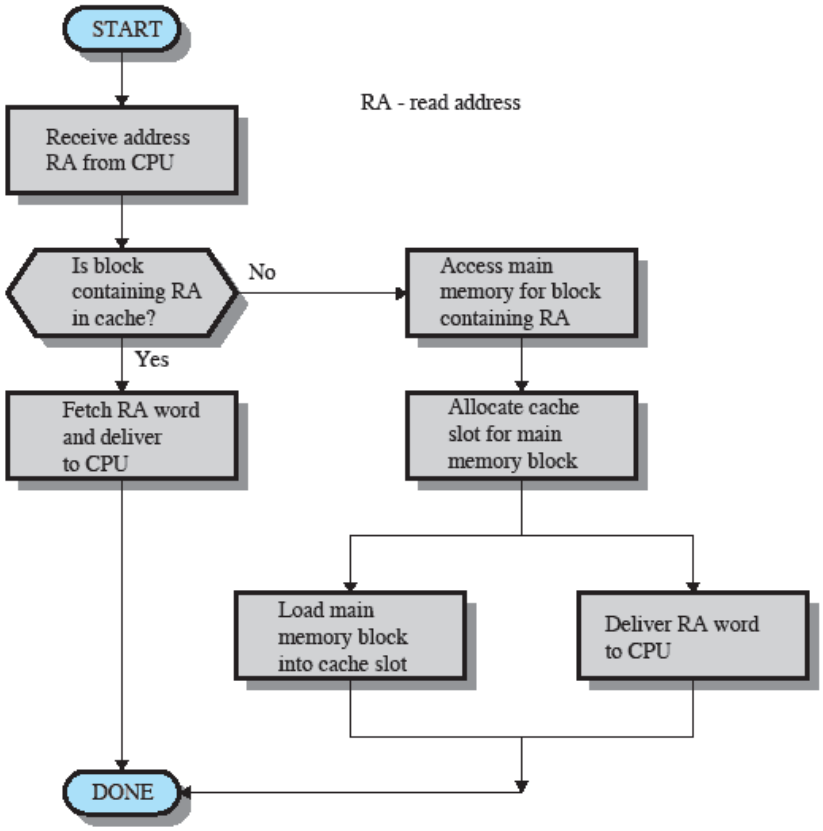
Cache Memory Structure

➔ N address lines => 2n words of memory
➔ Cache stores fixed length blocks of K words
➔ Cache views memory as an array of M blocks where M = 2n/K
➔ A block of memory in cache is referred to as a line. K is the line size
➔ Cache size of C blocks where C < M
(considerably)
➔ Each line includes a tag that identifies the block being stored
➔ Tag is usually upper portion of memory address

As a simple example, suppose that we have a 6-bit address and a 2-bit tag.
The tag 01 refers to the block of locations with the following addresses: 010000, 010001, 010010, 010011, 010100, 010101, 010110, 010111, 011000, 011001, 011010, 011011, 011100, 011101, 011110, and 011111.



Cache Read Operation



The processor generates the address, RA, of a word to be read. If the word is contained in the cache, it is delivered to the processor. Otherwise, the block containing that word is loaded into the cache and the word is delivered to the processor.

Cache Design

Elements of Cache Design

- Addresses (logical or physical)
- Size
- Mapping Function (direct, associative, set associative)
- Replacement Algorithm (LRU, LFU, FIFO, random)
- Write Policy (write through, write back, write once)
- Line Size
- Number of Caches (how many levels, unified or split)

Cache size

Even small caches have significant impact on performance

Block size

The unit of data exchanged between cache and main memory

Larger block size yields more hits until probability of using newly fetched data becomes less than the probability of reusing data that have to be moved out of cache.

Mapping function

Determines which cache location the block will occupy

Replacement algorithm

Chooses which block to replace

Least-recently-used (LRU) algorithm

Write policy

- Dictates when the memory write operation takes place
- Can occur every time the block is updated
- Can occur when the block is replaced

Minimize write operations

Leave main memory in an obsolete state

DIRECT MEMORY ACCESS (DMA)

➔Three techniques are possible for I/O operations: programmed I/O, interrupt-driven I/O, and direct memory access (DMA). Before discussing DMA, we briefly define the other two techniques; see Appendix C for more detail. When the processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to the appropriate I/O module.

➔In the case of **programmed I/O** , the I/O module performs the requested action and then sets the appropriate bits in the I/O status register but takes no further action to alert the processor. In particular, it does not interrupt the processor. Thus, after the I/O instruction is invoked, the processor must take some active role in determining when the I/O instruction is completed. For this purpose, the processor periodically checks the status of the I/O module until it finds that the
Operation is complete.

➔With programmed I/O, the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of more data. The processor, while waiting, must repeatedly interrogate the status of the I/O module.

➔As a result, the performance level of the entire system is severely degraded. An alternative, known as **interrupt-driven I/O** , is for the processor to issue an I/O command to a module and then go on to do some other useful work.

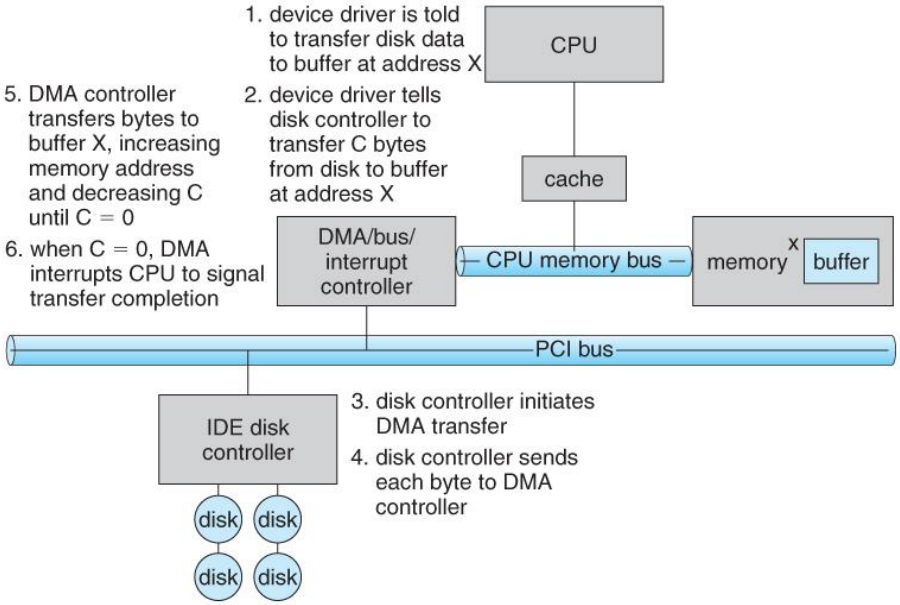
➔The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor. The processor then executes the data transfer, as before, and then resumes its former processing.

➔When large volumes of data are to be moved, a more efficient technique is required: **direct memory access (DMA).** The DMA function can be performed by a separate module on the system bus or it can be incorporated into an I/O module. In either case, the technique works as follows. When the processor wishes to read or write a block of data, it issues a command to the DMA module, by sending to the DMA module the following information:

- Whether a read or write is requested
- The address of the I/O device involved
- The starting location in memory to read data from or write data to
- The number of words to be read or written

➔The processor then continues with other work. It has delegated this I/O operation to the DMA module, and that module will take care of it. The DMA module transfers the entire block of data, one word at a time, directly to or from memory without going through the processor. When the transfer is complete, the DMA module sends an interrupt signal to the processor. Thus, the processor is involved only at the beginning and end of the transfer.

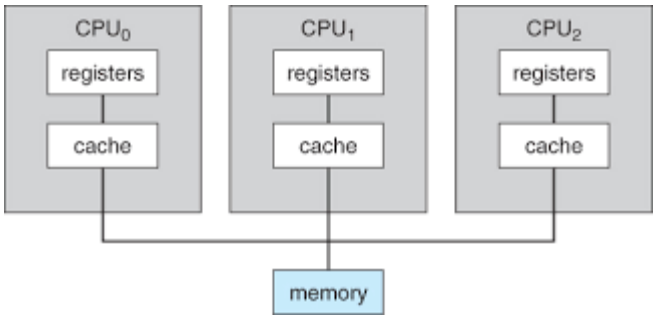
The Figure below illustrates the DMA process.



MULTIPROCESSOR AND MULTICORE ORGANIZATION

MULTIPROCESSING

➔Multiprocessing is the use of two or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them.



➔There are multiple processors, each of which contains its own control unit, arithmetic logic unit, and registers. Each processor has access to a shared main memory and the I/O devices through some form of interconnection mechanism; a shared bus is a common facility. The processors can communicate with each other through memory (messages and status information left in shared address spaces). It may also be possible for processors to exchange signals directly. The memory is often organized so that multiple simultaneous accesses to separate blocks of memory are possible.

➔Multiprocessor systems have three main advantages.

1. Increased throughput.
2. Economy of scale.
3. Increased reliability.

The most common multiple-processor systems now use **symmetric multiprocessing** (SMP), in which each processor runs an identical copy of the operating system, and these copies communicate with one another as needed.

Some systems use **asymmetric multiprocessing**, in which each processor is assigned a specific task. A master processor controls the system; the other processors either look to the master for instruction or have predefined tasks. This scheme defines a master-slave relationship. The master processor schedules and allocates work to the slave processors.

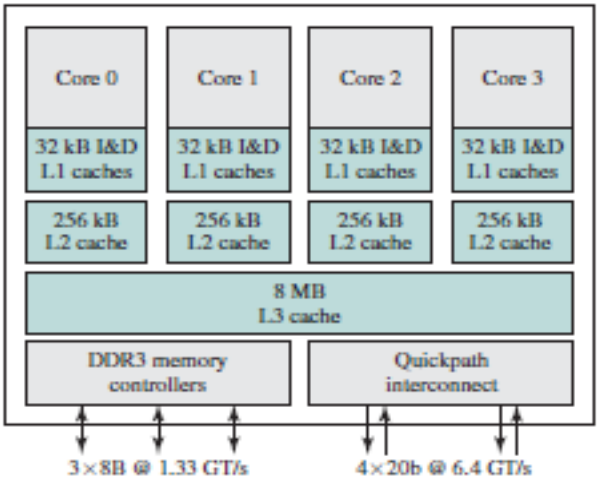
An SMP organization has a number of potential advantages over a uni-processor organization, including the following:

- **Performance:** If the work to be done by a computer can be organized so that some portions of the work can be done in parallel, then a system with multiple processors will yield greater performance than one with a single processor of the same type.
- **Availability:** In a symmetric multiprocessor, because all processors can perform the same functions, the failure of a single processor does not halt the machine. Instead, the system can continue to function at reduced performance.
- **Incremental growth:** A user can enhance the performance of a system by adding an additional processor.
- **Scaling:** Vendors can offer a range of products with different price and performance characteristics based on the number of processors configured in the system.

➔**MULTICORE COMPUTERS**

➔A **multicore** computer, also known as a **chip multiprocessor,** combines two or more processors (called cores) on a single piece of silicon (called a die). Typically, each core consists of all of the components of an independent processor, such as registers, ALU, pipeline hardware, and control unit, plus L1 instruction and data caches. In addition to the multiple cores, contemporary multicore chips also include L2 cache and, in some cases, L3 cache. The motivation for the development of multicore computers can be summed up as follows.

➔For decades, microprocessor systems have experienced a steady, usually exponential, increase in performance. This is partly due to hardware trends, such as an increase in clock frequency and the ability to put cache memory closer to the processor because of the increasing miniaturization of microcomputer components. Performance has also been improved by the increased complexity of processor design to exploit parallelism in instruction execution and memory access.



➔In brief, designers have come up against practical limits in the ability to achieve greater performance by means of more complex processors. Designers have found that the best way to improve performance to take advantage of advances in hardware is to put multiple processors and a substantial amount of cache memory on a single chip

➔An example of a multicore system is the Intel Core i7, which includes four x86 processors, each with a dedicated L2 cache, and with a shared L3 cache. One mechanism Intel uses to make its caches more effective is prefetching, in which the hardware examines memory access patterns and attempts to fill the caches speculatively with data that's likely to be requested soon.

COMPONENTS OF OPERATING SYSTEM.

There are eight major operating system components.

They are :
- Process management
- Main-memory management
- File management
- I/O-system management
- Secondary-storage management
- Networking
- Protection system
- Command-interpreter system

(i) Process Management

- A process can be thought of as a program in execution. A batch job is a process. A time shared user program is a process.
- A process needs certain resources-including CPU time, memory, files, and I/O devices-to accomplish its task.
- A program by itself is not a process; a program is a passive entity, such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute.
- A process is the unit of work in a system.
- The operating system is responsible for the following activities in connection with process management:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

(ii) Main – Memory Management

- Main memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Each word or byte has its own address.
- Main memory is a repository of quickly accessible data shared by the CPUand I/O devices.
- To improve both the utilization of the CPU and the speed of the computer's response to its users, we must keep several programs in memory.
- The operating system is responsible for the following activities in connection with memory management:
  - Keeping track of which parts of memory are currently being used and by whom.
  - Deciding which processes are to be loaded into memory when memory space becomes available
  - Allocating and deallocating memory space as needed.

(iii) File Management

- File management is one of the most visible components of an operating system.
- The operating system is responsible for the following activities in connection with file management:
  - Creating and deleting files
  - Creating and deleting directories
  - Supporting primitives for manipulating files and directories
  - Mapping files onto secondary storage
  - Backing up files on stable (nonvolatile) storage media

(iv) I/O System management

- One of the purposes of an operating system is to hide the peculiarities of specific hardware devices from the user. This is done using the I/O subsystem.
- The I/O subsystem consists of
  - A memory-management component that includes buffering, caching, and spooling
  - A general device-driver interface
  - Drivers for specific hardware devices

(v) Secondary storage management

- Because main memory is too small to accommodate all data and programs, and because the data that it holds are lost when power is lst the computer system must provide secondary storage to back up main memory.
- The operating system is responsible for the following activities in connection with disk management:
  - Free-space management
  - Storage allocation
  - Disk scheduling

(vi) Networking

- A distributed system is a collection of processors that do not share memory, peripheral devices, or a clock.
- Instead, each processor has its own local memory and clock, and the processors communicate with one another through various communication lines, such as high-speed buses or networks.
- The processors in the system are connected through a communication network, which can be configured in a number of different ways.

(vii) Protection System

- Various processes must be protected from one another's activities. For that purpose, mechanisms ensure that the files, memory segments, CPU, and other resources can be operated on by only those processes that have gained proper authorization from the operating system.
- Protection is any mechanism for controlling the access of programs, processes, or users to the resources defined by a computer system.
- Protection can improve reliability by detecting latent errors at the interfaces between component subsystems.

(viii) Command-Interpreter System

- One of the most important systems programs for an operating system is the command interpreter.
- It is the interface between the user and the operating system.
- Some operating systems include the command interpreter in the kernel. Other operating systems, such as MS-DOS and UNIX, treat the

command interpreter as a special program that is running when a job is initiated, or when a user first logs on (on time-sharing systems).

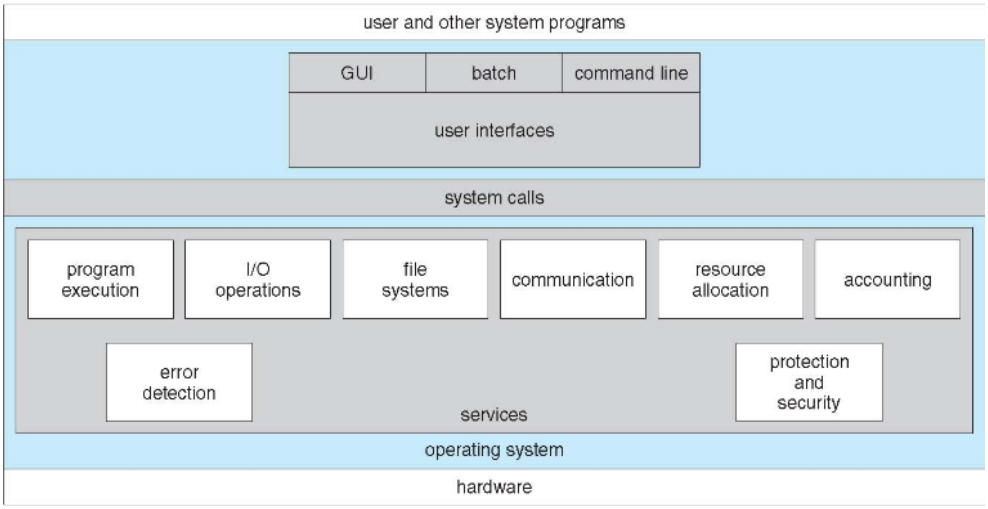
- Many commands are given to the operating system by control statements.
- When a new job is started in a batch system, or when a user logs on to a time-shared system, a program that reads and interprets control statements is executed automatically.
- This program is sometimes called the control-card interpreter or the Command-line interpreter, and is often known as the shell.

SERVICES OF OPERATING SYSTEM

➔An operating system provides services to programs and to the users of those programs. It provided by one environment for the execution of programs.

➔The services provided by one operating system is difficult than other operating system. Operating system makes the programming task easier. The common service provided by the operating system is listed below.

1.Program execution
2.I/O operation
3.File system manipulation
4.Communications
5.Error detection



The OS provides certain services to programs and to the users of those programs.

1. **Program execution:** The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).
2. **I/O operations:** A running program may require I/O. This I/O may involve a file or an I/O device.
3. **File-system manipulation:** The program needs to read, write, create, delete files.
4. **Communications:** In many circumstances, one process needs to exchange Information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network.
5. **Error detection:** The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure),

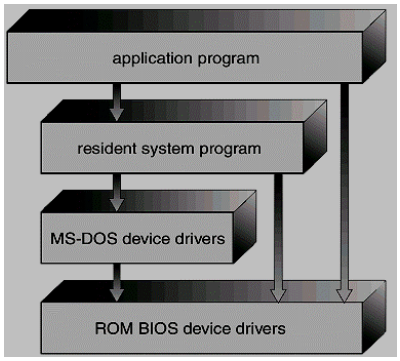
in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer), and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing.

6. **Resource allocation:** Different types of resources are managed by the Os.When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them.
7. **Accounting:** We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting or simply for accumulating usage statistics.
8. **Protection:** The owners of information stored in a multiuser computer system may want to control use of that information. Security of the system is also important.

OPERATING SYSTEM STRUCTURES

SIMPLE STRUCTURE:

➔In MS-DOS, application programs are able to access the basic I/O routines to write directly to the display and disk drives. Such freedom leaves MS-DOS vulnerable to errant (or malicious) programs, causing entire system to crash when user programs fail.



MS-DOS LAYER STRUCTURE:

➔ UNIX operating system. It consists of two separable parts, the kernel and the system programs. The kernel is further separated into a series of interfaces and device drivers. We can view the traditional UNIX operating system as being layered. Everything below the system call interface and above the physical hardware is the kernel.

(the users)		
shells and commands compilers and interpreters system libraries		
system-call interface to the kernel		
signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory
kernel interface to the hardware		
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory

The kernel provides the file system, CPU scheduling, memory management, and other operating system functions through system calls. There is number of functionality to be combined into one level. This monolithic structure was difficult to implement and maintain.

LAYERED APPROACH:

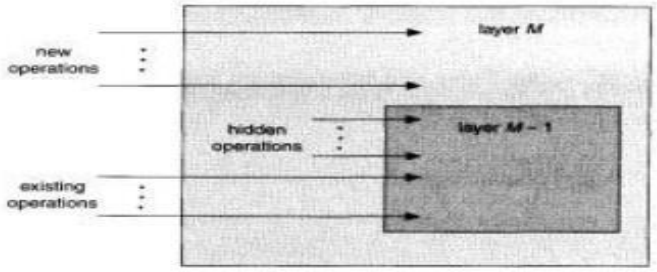
➔The operating system is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware; the highest (layer M) is the user interface.

➔The main advantage of the layered approach is simplicity of construction and debugging. The layers are selected so that each uses functions (operations) and services of only lower-level layers. This approach simplifies debugging and system verification. The first layer can be debugged without any concern for the rest of the system, because, by definition, it uses only the basic hardware to implement its functions.

➔Once the first layer is debugged, its correct functioning can be assumed while the second layer is debugged, and so on. If an error is found during the debugging of a particular layer, the error must be on that layer, because the layers below it are already debugged. Each layer hides the existence of certain data structures, operations, and hardware from higher-level layers.

➔The major difficulty with the layered approach involves appropriately defining the various layers as a layer can use only lower-level layers. Another problem with layered implementations is they tend to be less efficient than other types. Each layer adds overhead to the system call; the net result is a system call that takes longer than a non- layered system.

Example of Layered Approach

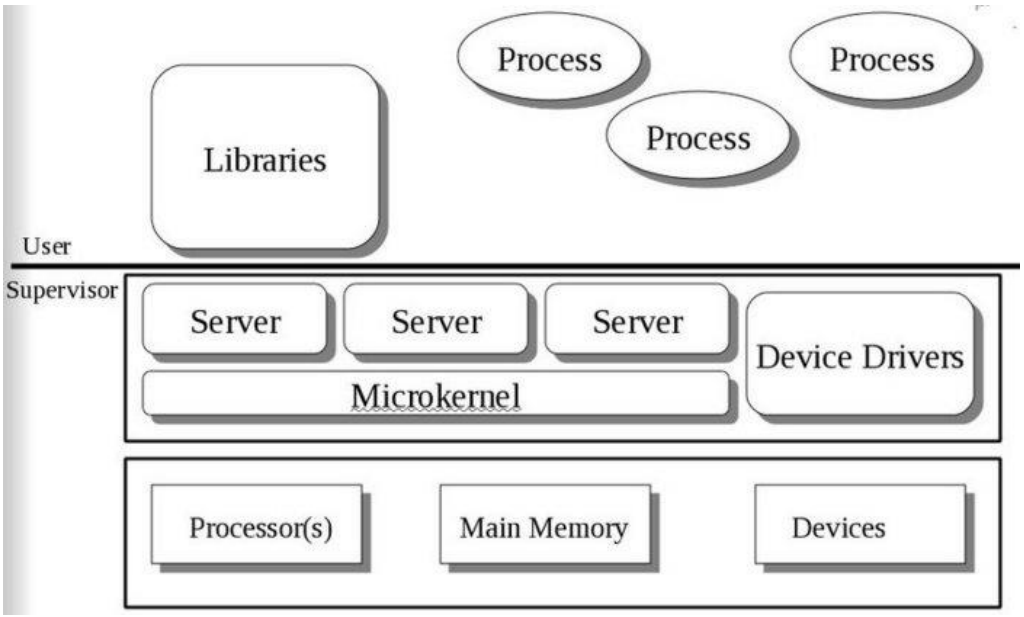


MICROKERNEL APPROACH:

➔In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called Mach that modularized the kernel using the microkernel approach. Microkernel's provide minimal process and memory management, in addition to a communication facility.

➔The main function of the micro kernel is to provide a communication facility between the client program and the various services running in user space. One benefit of the microkernel approach is ease of extending the operating system. All new services are added to user space and consequently do not require modification of the kernel. The microkernel also provides more security and reliability, since most services are running as user, rather than kernel-processes.

➔Microkernel's can suffer from decreased performance due to increased system function overhead.



MODULES:

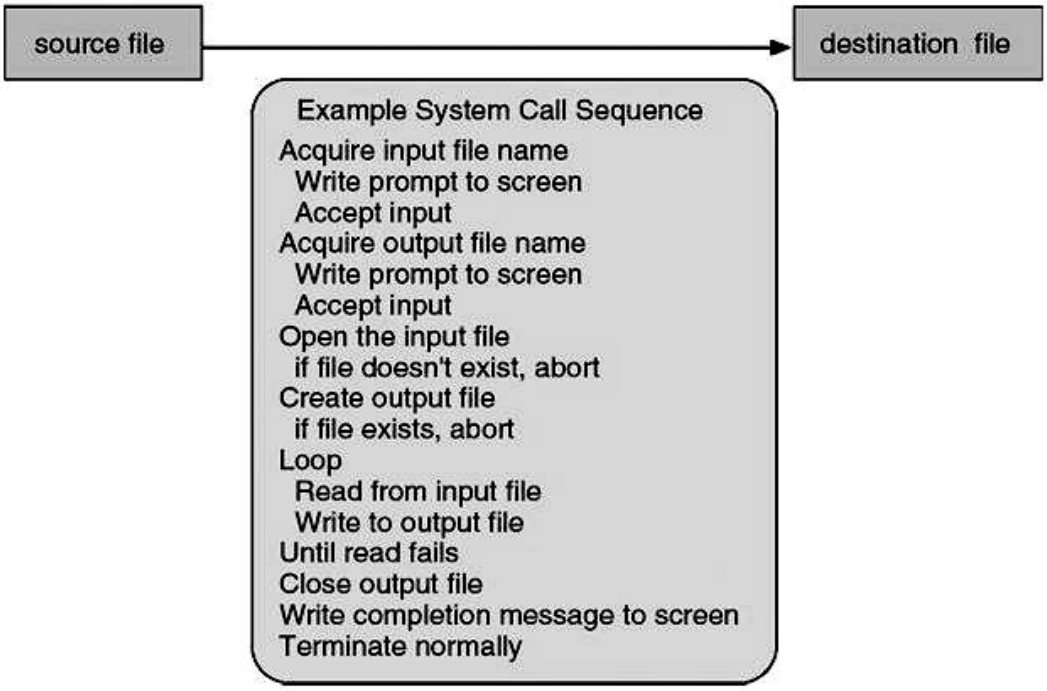
➔The current methodology for operating-system design involves using object-oriented programming techniques to create a modular kernel. Here, the kernel has a set of core components and links in additional services either during boot time or during run time. Such a strategy uses dynamically loadable modules.

➔Such a design allows the kernel to provide core services yet also allows certain features to be implemented dynamically.

SYSTEM CALLS

➔A system call is a request that a program makes to the kernel through a software interrupt. System calls provide the interface between a process and the operating system.

➔These calls are generally available as assembly-language instructions. Certain systems allow system calls to be made directly from a high-level language program, in which case the calls normally resemble predefined function or subroutine calls.



TYPES OF SYSTEM CALLS:

Traditionally, System Calls can be categorized in six groups, which are: Process Control, File Management, Device Management, Information Maintenance, Communications and Protection.

PROCESS CONTROL

- A running program needs to be able to stop execution either normally or abnormally.
- When execution is stopped abnormally, often a dump of memory is taken and can be examined with a debugger.

Following are functions of process control:

End, abort
Load, execute
Create process, terminate process
Get process attributes, set process attributes
Wait for time
Wait event, signal event
Allocate and free memory

FILE MANAGEMENT

- We first need to be able to create and delete files. Either system call requires the name of the file and perhaps some of the file's attributes.
- Once the file is created, we need to open it and to use it. We may also read, write, or reposition. Finally, we need to close the file, indicating that we are no longer using it.
- We may need these same sets of operations for directories if we have a directory structure for organizing files in the file system.
- In addition, for either files or directories, we need to be able to determine the values of various attributes and perhaps to reset them if necessary. File attributes include the file name, a file type, protection codes, accounting information, and so on

Functions:

Create, delete file
Open, close
Read, write, reposition
Get file attributes, set file attributes

DEVICE MANAGEMENT

- A process may need several resources to execute - main memory, disk drives, access to files, and so on. If the resources are available, they can be granted, and control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.
- The various resources controlled by the OS can be thought of as devices. Some of these devices are physical devices (for example, tapes), while others can be thought of as abstract or virtual devices (for example, files).
- Once the device has been requested (and allocated to us), we can read, write, and (possibly) reposition the device, just as we can with files.
- In fact, the similarity between I/O devices and files is so great that many OSs, including UNIX, merge the two into a combined file-device structure.
- A set of system calls is used on files and devices. Sometimes, 1/0 devices are identified by special file names, directory placement, or file attributes.

Functions:

Request device, release device
Read, write, reposition
Get device attributes, set device attributes
Logically attach or detach devices

INFORMATION MAINTENANCE

- Many system calls exist simply for the purpose of transferring information between the user program and the OS. For example, most systems have a system call to return the current time and date.
- Other system calls may return information about the system, such as the number of current users, the version number of the OS, the amount of free memory or disk space, and so on.
- In addition, the OS keeps information about all its processes, and system calls are used to access this information. Generally, calls are also used to reset the process information.

Functions:

Get time or date, set time or date
Get system data, set system data
Get process, file, or device attributes
Set process, file, or device attributes

COMMUNICATIONS

- There are two common models of interprocess communication: the message-passing model and the shared-memory model. In the message-passing model, the communicating processes exchange messages with one another to transfer information.
- In the shared-memory model, processes use shared memory creates and shared memory attaches system calls to create and gain access to regions of memory owned by other processes.
- Recall that, normally, the OS tries to prevent one process from accessing another process's memory. Shared memory requires that two or more processes agree to remove this restriction. They can then exchange information by reading and writing data in the shared areas.
- Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. It is also easier to implement than is shared memory for intercomputer communication.
- Shared memory allows maximum speed and convenience of communication, since it can be done at memory speeds when it takes place within a computer. Problems exist, however, in the areas of protection and synchronization between the processes sharing memory.

Functions:

Create, delete communication connection
Send, receive messages
Transfer status information
Attach or detach remote devices

PROTECTION

Get File Security, Set File Security

Get Security Group, Set Security Group

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

SYSTEM PROGRAMS

➔System programs provide a convenient environment for program development and execution. They can be divided into several categories:

1. **File management:** These programs create, delete, copy, rename, print, dump, list, and generally manipulate files and directories.
2. **Status information:** The status such as date, time, amount of available memory or diskspace, number of users or similar status information.
3. **File modification:** Several text editors may be available to create and modify the content of files stored on disk or tape.
4. **Programming-language support:** Compilers, assemblers, and interpreters for common programming languages are often provided to the user with the operating system.
5. **Program loading and execution:** The system may provide absolute loaders, relocatable loaders,

linkage editors, and overlay loaders.

6. **Communications:** These programs provide the mechanism for creating virtual connections among processes, users, and different computer systems. (email, FTP, Remote log in)
7. **Application programs:** Programs that are useful to solve common problems, or to perform common operations.

Eg. Web browsers, database systems.

GENERATION AND SYSTEM BOOT.

Operating-System Generation

➔ It is possible to design, code, and implement an operating system specifically for one machine at one site. More commonly, however, operating systems are designed to run on any of a class of machines at a variety of sites with a variety of peripheral configurations. The system must then be configured or generated for each specific computer site, a process sometimes known as system generation (SYSGEN).

➔The operating system is normally distributed on disk or CD-ROM. To generate a system, we use a special program. The SYSGEN program reads from a given file, or asks the operator of the system for information concerning the specific configuration of the hardware system, or probes the hardware directly to determine what components are there. The following kinds of information must be determined.

➔What CPU is to be used?

What options (extended instruction sets, floating-point arithmetic, and so on) are installed? For multiple CPU systems, each CPU must be described.

➔ How much memory is available?

Some systems will determine this value themselves by referencing memory location after memory location until an "illegal address" fault is generated. This procedure defines the final legal address and hence the amount of available memory.

➔ What devices are available?

The system will need to know how to address each device (the device number), the device interrupt number, the device's type and model, and any special device characteristics.

➔What operating-system options are desired, or what parameter values are to be used? These options or values might include how many buffers of which sizes should be used, what type of CPU-scheduling algorithm is desired, what the maximum number of processes to be supported is, and so on. Once this information is determined, it can be used in several ways.

System Boot

➔After an operating system is generated, it must be made available for use by the hardware. But how does the hardware know where the kernel is or how to load that kernel? The procedure of starting a computer by loading the kernel is known as booting the system.

➔On most computer systems, a small piece of code known as the **bootstrap program** or **bootstrap loader** locates the kernel, loads it into main memory, and starts its execution. Some computer systems, such as PCs, use a two-step process in which a simple bootstrap loader fetches a more complex boot program from disk, which in turn loads the kernel.

➔When a CPU receives a reset event—for instance, when it is powered up or rebooted—the instruction register is loaded with a predefined memory location, and execution starts there. At that location is the initial bootstrap program.

➔This program is in the form of read-only memory (ROM), because the RAM is in an unknown state at system startup. ROM is convenient because it needs no initialization and cannot be infected by a computer virus.

➔The bootstrap program can perform a variety of tasks. Usually, one task is to run diagnostics to determine the state of the machine. If the diagnostics pass, the program can continue with the booting steps. It can also initialize all aspects of the system, from CPU registers to device controllers and the contents of main memory.

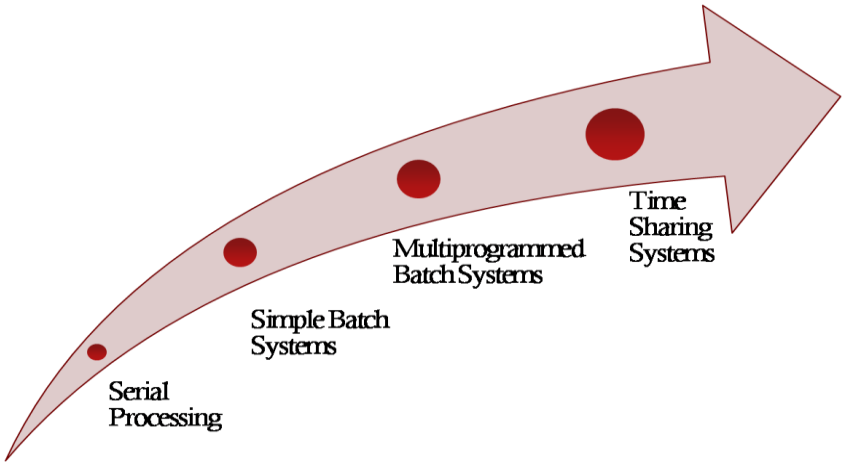
➔Sooner or later, it starts the operating system. Some systems—such as cellular phones, PDAs, and game consoles—store the entire operating system in ROM. Storing the operating system in ROM is suitable for small operating systems, simple supporting hardware, and rugged operation.

➔A problem with this approach is that changing the bootstrap code requires changing the ROM hardware chips. Some systems resolve this problem by using erasable programmable read-only memory (EPROM), which is read-only except when explicitly given a command to become writable.

➔All forms of ROM are also known as firmware, since their characteristics fall somewhere between those of hardware and those of software. A problem with firmware in general is that executing code there is slower than executing code in RAM. Some systems store the operating system in firmware and copy it to RAM for fast execution. A final issue with firmware is that it is relatively expensive, so usually only small amounts are available.

EVALUATION OF OPERATING SYSTEMS.

Stages of Evaluation



Serial Processing

➔Users access the computer in series. From the late 1940's to mid 1950's, the programmer interacted directly with computer hardware i.e., no operating system.

➔These machines were run with a console consisting of display lights, toggle switches, some form of input device and a printer. Programs in machine code are loaded with the input device like card reader.

➔If an error occur the program was halted and the error condition was indicated by lights. Programmers examine the registers and main memory to determine error. If the program is success, then output will appear on the printer.

➔Main problem here is the setup time. That is single program needs to load source program into memory, saving the compiled (object) program and then loading and linking together.

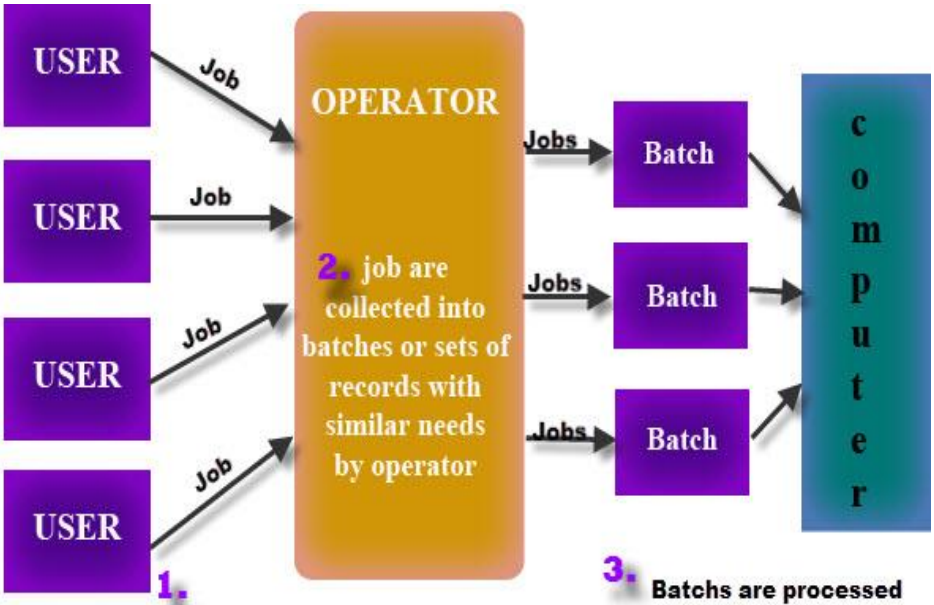
Simple Batch Systems

➔To speed up processing, jobs with similar needs are batched together and run as a group. Thus, the programmers will leave their programs with the operator. The operator will sort programs into batches with similar requirements.

The problems with Batch Systems are:

➔Lack of interaction between the user and job.CPU is often idle, because the speeds of the mechanical I/O devices are slower than CPU. For overcoming this problem use the Spooling

➔Technique. Spool is a buffer that holds output for a device, such as printer, that can not accept interleaved data streams. That is when the job requests the printer to output a line. That line is copied into a system buffer and is written to the disk. When the job is completed, the output is printed. Spooling technique can keep both the CPU and the I/O devices working at much higher rates.

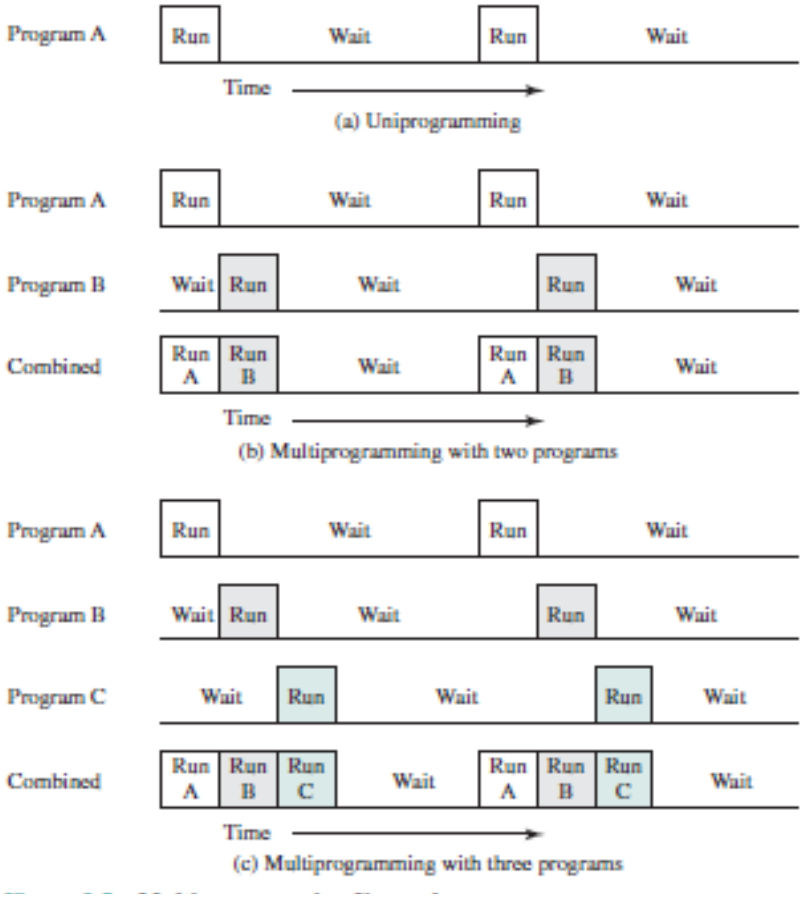


Multiprogrammed Batch Systems

➔Jobs must be run sequentially, on a first-come, first-served basis. However when several jobs are on a direct-access device like disk, job scheduling is possible. The main aspect of job scheduling is multiprogramming. Single user cannot keep the CPU or I/O devices busy at all times.

Thus multiprogramming increases CPU utilization.

➔In when one job needs to wait, the CPU is switched to another job, and so on. Eventually, the first job finishes waiting and gets the CPU back.

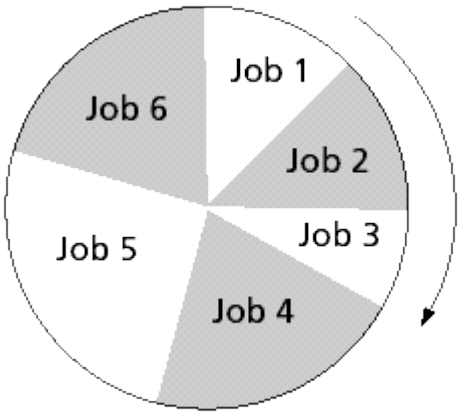


Time-Sharing Systems

➔Time-sharing systems are not available in 1960s. Time-sharing or multitasking is a logical extension of multiprogramming. That is processors time is shared among multiple users simultaneously is called time-sharing. The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is in multiprogrammed batch systems its objective is maximize processor use, whereas in Time-Sharing Systems its objective is minimize response time.

➔Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receives an immediate response. For example, in a transaction processing, processor execute each user program in a short burst or quantum of computation. That is if n users are present, each user can get time quantum. When the user submits the command, the response time is seconds at most.

➔Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.
For example IBM's OS/360.
Time-sharing operating systems are even more complex than multi-programmed operating systems. As in multiprogramming, several jobs must be kept simultaneously in memory.



OBJECTIVES AND FUNCTIONS OF AN OPERATING SYSTEMS

An OS is a program that controls the execution of application programs and acts as an interface between applications and the computer hardware. It can be thought of as having three objectives:

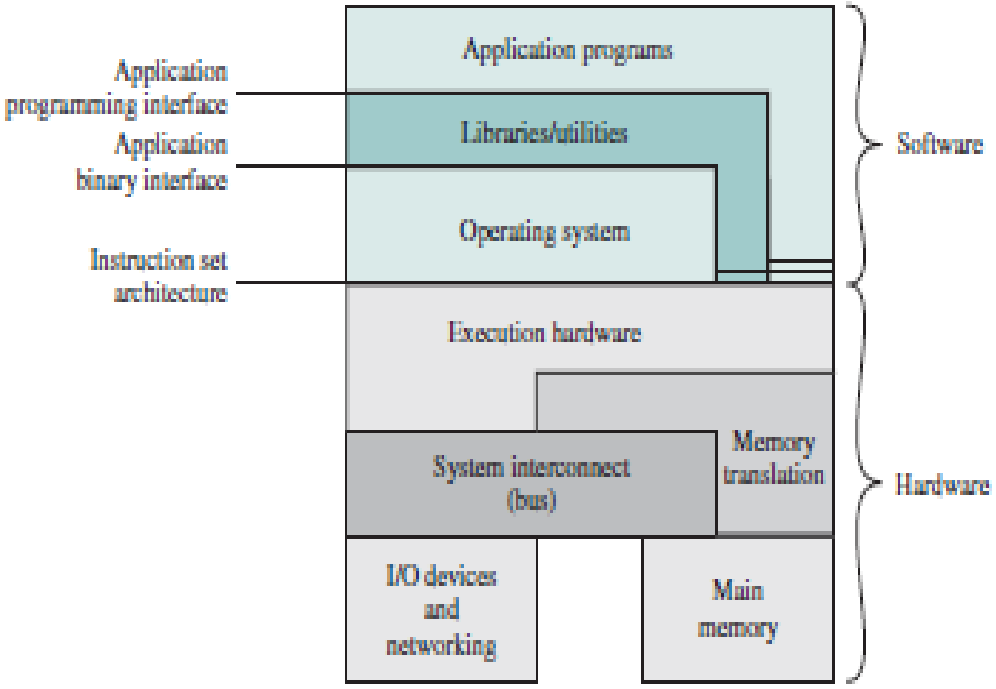
- **Convenience**: An OS makes a computer more convenient to use.
- **Efficiency**: An OS allows the computer system resources to be used in an efficient manner.
- **Ability to evolve**: An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

The Operating System as a User/Computer Interface

➔The hardware and software used in providing applications to a user can be viewed in a layered or hierarchical fashion. The user of those applications, the end user, generally is not concerned with the details of computer hardware. Thus, the end user views a computer system in terms of a set of applications.

➔An application can be expressed in a programming language and is developed by an application programmer. If one were to develop an application program as a set of machine instructions that is completely responsible for controlling the computer hardware, one would be faced with an overwhelmingly complex undertaking.

➔To ease this chore, a set of system programs is provided. Some of these programs are referred to as utilities, or library programs. These implement frequently used functions that assist in program creation, the management of files, and the control of I/O devices. A programmer will make use of these facilities in developing an application, and the application, while it is running, will invoke the utilities to perform certain functions.



Three key interfaces in a typical computer system:

· **Instruction set architecture (ISA)** : The ISA defines the repertoire of machine language instructions that a computer can follow. This interface is the boundary between hardware and software. Note that both application programs and utilities may access the ISA directly. For these programs, a subset of the instruction repertoire is available (user ISA). The OS has access to additional machine language instructions that deal with managing system resources (system ISA).

· **Application binary interface (ABI)** : The ABI defines a standard for binary portability across programs. The ABI defines the system call interface to the operating system and the hardware resources and services available in a system through the user ISA.

· **Application programming interface (API)** : The API gives a program access to the hardware resources and services available in a system through the user ISA supplemented with high-level language (HLL) library calls. Any system calls are usually performed through libraries. Using an API enables application software to be ported easily, through recompilation, to other systems that support the same API.

The Operating System as Resource Manager

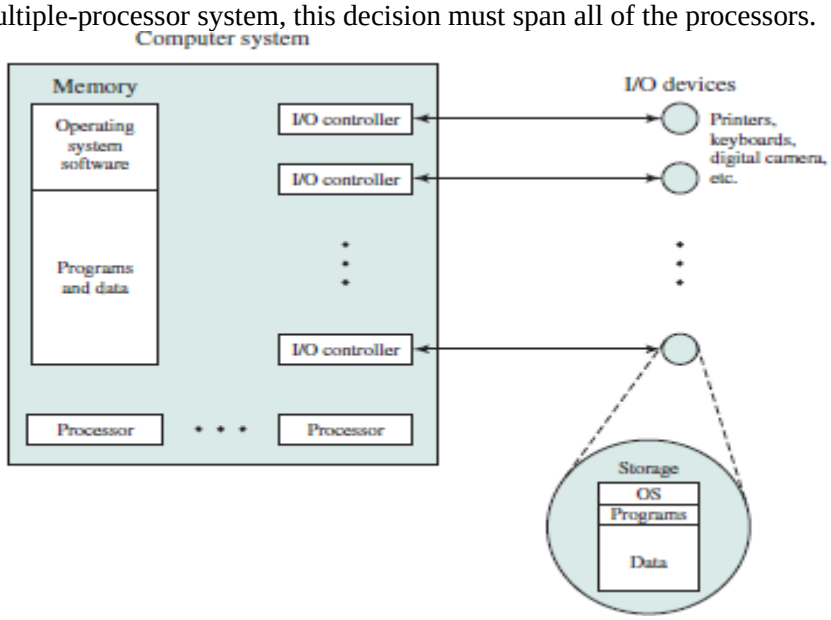
A computer is a set of resources for the movement, storage, and processing of data and for the control of these functions. The OS is responsible for managing these resources.

By managing the computer's resources, the OS is in control of the computer' s basic functions

The main resources that are managed by the OS. A portion of the OS is in main memory. This includes the **kernel** , or **nucleus** , which contains the most frequently used functions in the OS and, at a given time, other portions of the OS currently in use. The remainder of main memory contains user programs and data.

The memory management hardware in the processor and the OS jointly control the allocation of main memory, as we shall see. The OS decides when an I/O device can be used by a program in execution and

controls access to and use of files. The processor itself is a resource, and the OS must determine how much processor time is to be devoted to the execution of a particular user program. In the case of a multiple-processor system, this decision must span all of the processors.



Ease of Evolution of an Operating System

A major OS will evolve over time for a number of reasons:

- **Hardware upgrades plus new types of hardware**
- **New services**
- **Fixes**

OPERATING SYSTEM OPERATIONS

Modern operating systems are interrupt driven. If there are no processes to execute, OS will sit idle and wait for some event to happen. Interrupts could be hardware interrupts or software interrupts. The OS is designed to handle both. A trap (or an exception) is a software generated interrupt caused either by an error (e.g. divide by zero) or by a specific request from a user program. A separate code segment is written in the OS to handle different types of interrupts. These codes are known as interrupt handlers/ interrupt service routine. A properly designed OS ensures that an illegal program should not harm the execution of other programs. To ensure this, the OS operates in dual mode.

Dual mode of operation

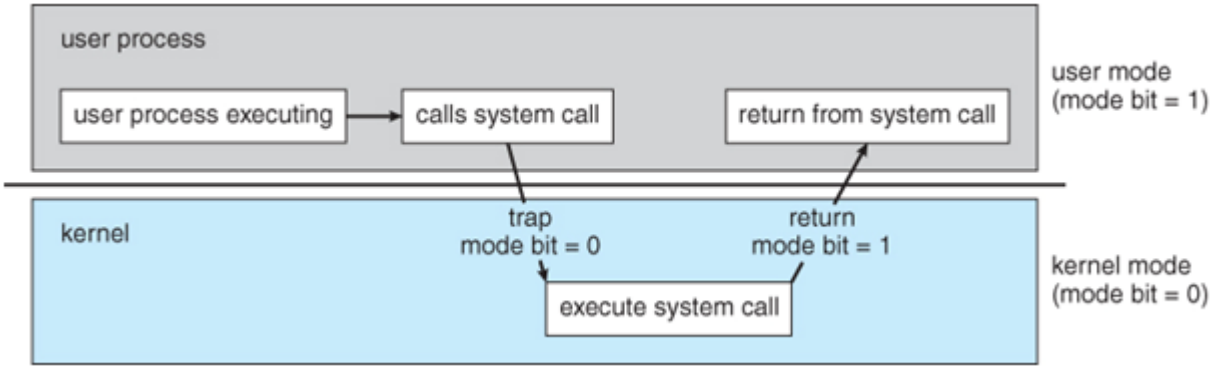
The OS is design in such a way that it is capable of differentiating between the execution of OS code and user defined code. To achieve this OS need two different modes of operations this is thereby controlled by mode bit added to hardware of computer system as shown in Table 4.

Mode Type	**Definition**	**Mode Bit**	**Examples**
User Mode	User Defined codes are executed	Mode Bit=1	Creation of word document or in general user using any application program
Kernel Mode	OS system codes are executed (also known as supervisor, system, or privileged mode)	Mode Bit=0	Handling interrupts-Transferring control of a process from CPU to I/O on request

User and Kernel Mode of Operating System

Transition from User to Kernel mode

When a user application is executing on the computer system OS is working in user mode. On signal of system call via user application, the OS transits from user mode to kernel mode to service that request as shown in Fig. 11.



Transition from user to kernel mode

When the user starts the system the hardware starts in monitor/ kernel mode and loads the operating system. OS has the initial control over the entire system, when instructions are executed in kernel mode. OS then starts the user processes in user mode and on occurrence of trap, interrupt or system call again switch to kernel mode and gains control of the system. System calls are designed for the user programs through which user can ask OS to perform tasks reserved for operating system. System calls usually take the form of the trap. Once the OS service the interrupt it transfers control back to user program hence user mode by setting mode bit=1.

Benefits of Dual Mode

The dual mode of operation protects the operating system from errant users, and errant users from one another by designating some of the machine instructions that may cause harm as privileged instructions. These instructions can execute only in kernel mode. If an attempt is made to execute a privileged instruction in user mode, the hardware does not execute the instruction, but rather treats the instruction as illegal and traps to the operating system. Examples of privileged instructions:

1. Switching to kernel mode
2. Managing I/O control
3. Timer Management
4. Interrupt Management

Timer

Since OS operates in dual mode it should maintain control over CPU. The system should not allow a user application:

1. To be stuck in an infinite loop
2. To fail to call system services
3. Never return control to the OS

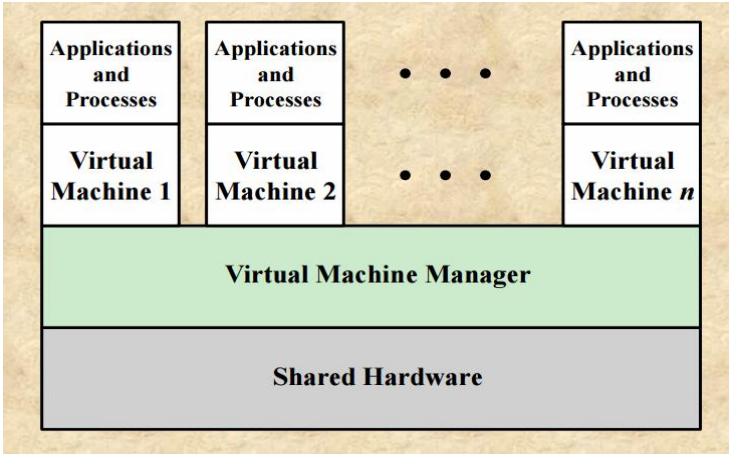
To achieve this goal, we can use timer. This timer control mechanism will interrupt the system at a specified period; thereby preventing user program from running too long. This can be implemented either as fixed timer or variable timer

Additional Topics

Virtual Machines (VM)

Virtualization technology enables a single PC or server to simultaneously run multiple operating systems or multiple sessions of a single OS

 A machine with virtualization software can host numerous applications, including those that run on different operating systems, on a single platform
 The host operating system can support a number of virtual machines, each of which has the characteristics of a particular OS
 The solution that enables virtualization is a virtual machine monitor (VMM), or hypervisor



A virtual machine takes the layered approach to its logical conclusion. It treats hardware and the operat

with its own (virtual) memory.

The resources of the physical computer are shared to create the virtual machines.

1. CPU scheduling can create the appearance that users have their own processor.
2. Spooling and a file system can provide virtual card readers and virtual line printers.
3. A normal user time-sharing terminal serves as the virtual machine operator's console.

Advantages/Disadvantages of Virtual Machines

The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines.

This isolation, however, permits no direct sharing of resources.

A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.

The virtual machine concept is difficult to implement due to the effort required to provide an exact duplicate to the underlying machine.

CS6401-OPERATING SYSTEMS

UNIT II PROCESS MANAGEMENT

Processes-Process Concept, Process Scheduling, Operations on Processes, Interprocess Communication; Threads- Overview, Multicore Programming, Multithreading Models; Windows 7 -Thread and SMP Management. Process Synchronization - Critical Section Problem, Mutex Locks, Semophores, Monitors; CPU Scheduling and Deadlocks.

PROCESS CONCEPTS

Process Concept

➔A process can be thought of as a program in execution.

➔A process is the unit of the unit of work in a modern time-sharing system.

A process is more than the program code, which is sometimes known as the **text section**. It also includes the current activity, as represented by the value of the **program counter** and the contents of the processor's registers.

A process generally also includes the process **stack**, which contains temporary data (such as function parameters, return addresses, and local variables), and a **data section**, which contains global variables. A process may also include a **heap**, which is memory that is dynamically allocated during process run time.

Difference between program and process

➔A program is a passive entity, such as the contents of a file stored on disk, whereas a process is an active entity, with a program counter specifying the next instruction to execute and a set of associated resources.

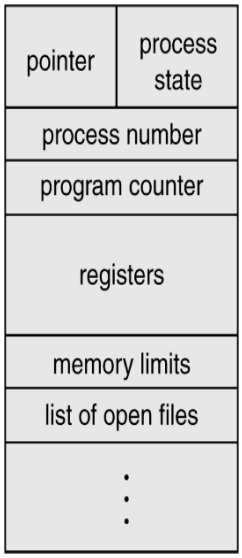
Process Control Block (PCB)

➔Each process is represented in the operating system by a process control block (PCB)-also called a task control block.

➔A PCB defines a process to the operating system.

➔It contains the entire information about a process.

Some of the information a PCB.



Process state: The state may be new, ready, running, and waiting, halted, and SO on.

Program counter: The counter indicates the address of the next instruction to be executed for this process.

CPU registers: The registers vary in number and type, depending on the computer architecture.

CPU-scheduling information: This information includes a process priority, pointers to scheduling queues, and any other scheduling parameters.

Memory-management information: This information may include such information as the value of the base and limit registers, the page tables, or the segment tables, depending on the memory system used by the operating system.

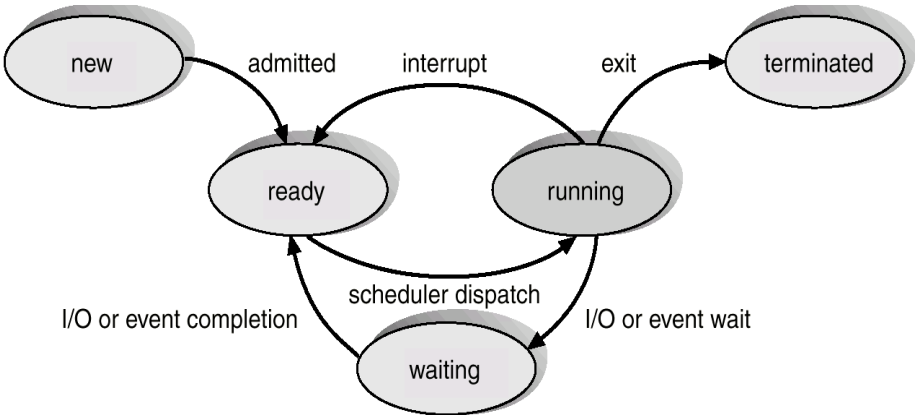
Accounting information: This information includes the amount

of CPU and real time used, time limits, account numbers, job or process numbers, and so on.

Status information: The information includes the list of I/Odevices allocated to this process, a list of open files, and so on.

Process States:

- As a process executes, it changes state.
- The state of a process is defined in part by the current activity of that process.
- Each process may be in one of the following states:
  - **New**: The process is being created.
  - **Running**: Instructions are being executed.
  - **Waiting**: The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - **Ready**: The process is waiting to be assigned to a processor.
  - **Terminated**: The process has finished execution.



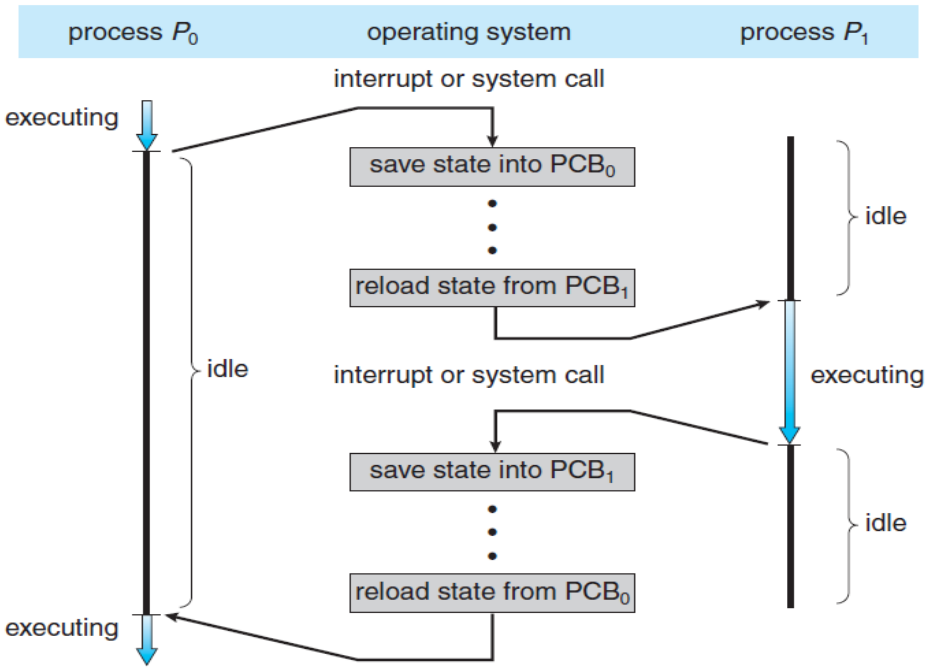


Diagram shows CPU switch from process to process.

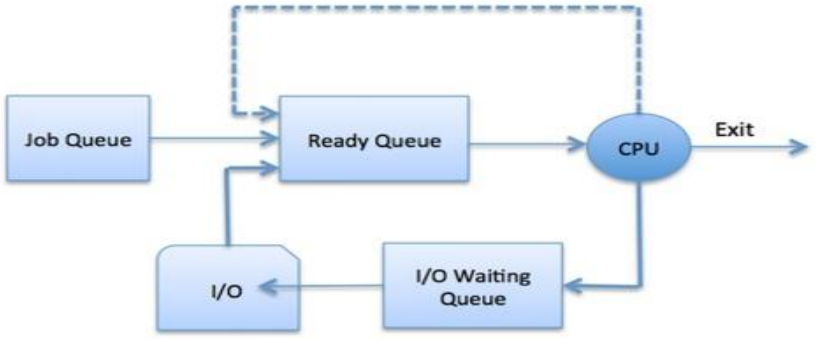
PROCESS SCHEDULING

- The objective of multiprogramming is to have some process running at all times, so as to maximize CPU utilization.

Scheduling Queues

There are 3 types of scheduling queues .They are :

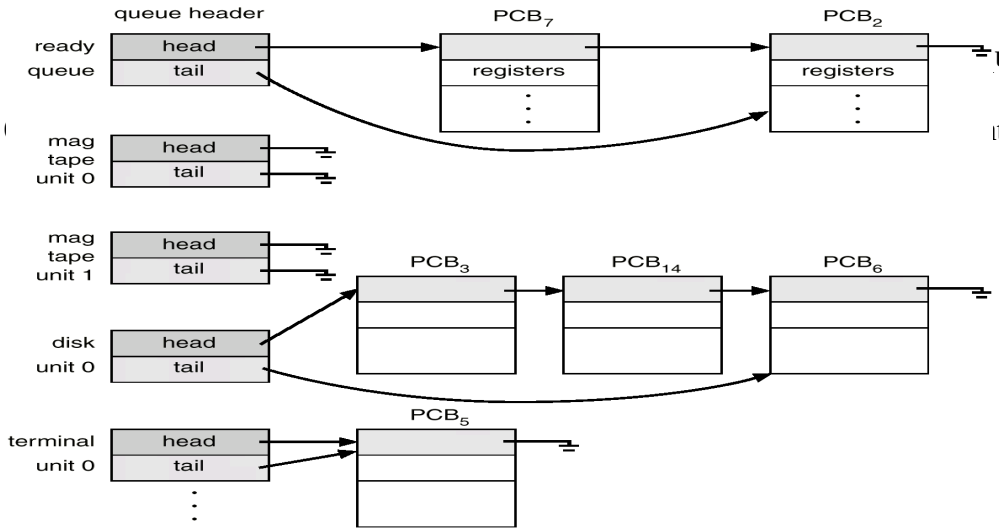
1. Job Queue
2. Ready Queue
3. Device Queue



As processes enter the system, they are put into a **job queue.**

The processes that are residing in main memory and are ready and waiting to execute are kept on a list called the **ready queue**.

The list of processes waiting for an I/O device is kept in a **device queue** for that particular device.



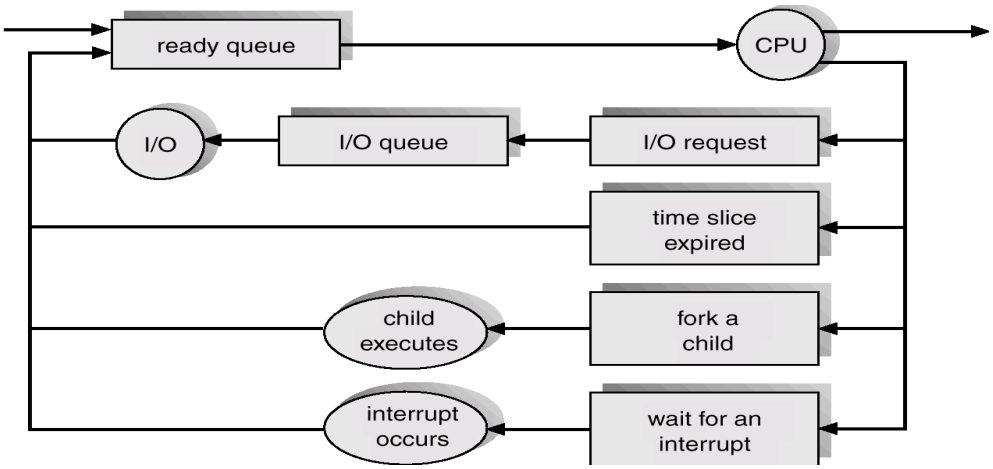
ueue until it is select d for

ts could occur:

- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new subprocess and wait for its

termination.

- The process could be removed forcibly from the CPU, as a result of aninterrupt, and be put back in the ready Queue.
- A common representation of process scheduling is a queueing diagram.



Schedulers

- The operating system must select, for scheduling purposes, processes from these queues in some order
- The selection process is carried out by the appropriate scheduler.

They are:

1. Long-term Scheduler or Job Scheduler
2. Short-term Scheduler or CPU Scheduler
3. Medium term Scheduler

Long-Term Scheduler

- The **long-term scheduler**, or **job scheduler**, selects processes from this pool and loads them into memory for execution. It is invoked very infrequently. It controls the degree of multiprogramming.

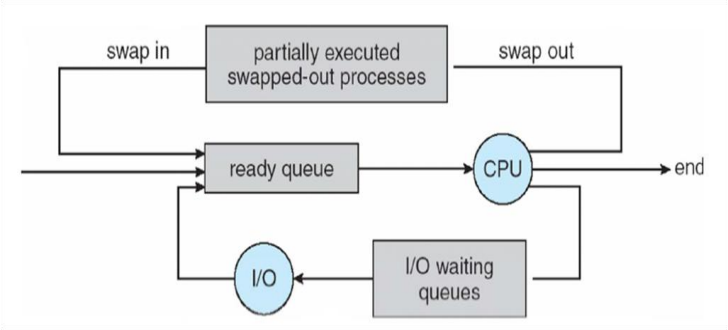
Short-Term Scheduler

- The **short-term scheduler**, or **CPU scheduler**, selects from among the processes that are ready to execute, and allocates the CPU to one of them. It is invoked very frequently.
- Processes can be described as either **I/O bound** or **CPU bound**.
- An **I\O-bound process** spends more of its time doing I/O than it spends doing computations.
- A **CPU-bound process**, on the other hand, generates I/O requests infrequently, using more of its time doing computation than an I/O-bound process uses.
- The system with the best performance will have a combination of CPU- bound and I/O-bound processes.

Medium Term Scheduler

- Some operating systems, such as time-sharing systems, may introduce an additional, intermediate level of scheduling.
- The key idea is medium-term scheduler, removes processes from memory and thus reduces the degree of multiprogramming.

- At some later time, the process can be reintroduced into memory and its execution can be continued where it left off. This scheme is called swapping.



Context Switching

- Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process.This task is known as a context switch.
- Context-switch time is pure overhead, because the system does no useful work while switching.
- Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions.

OPERATIONS ON PROCESS

1. Process Creation

A process may create several new processes, during execution.

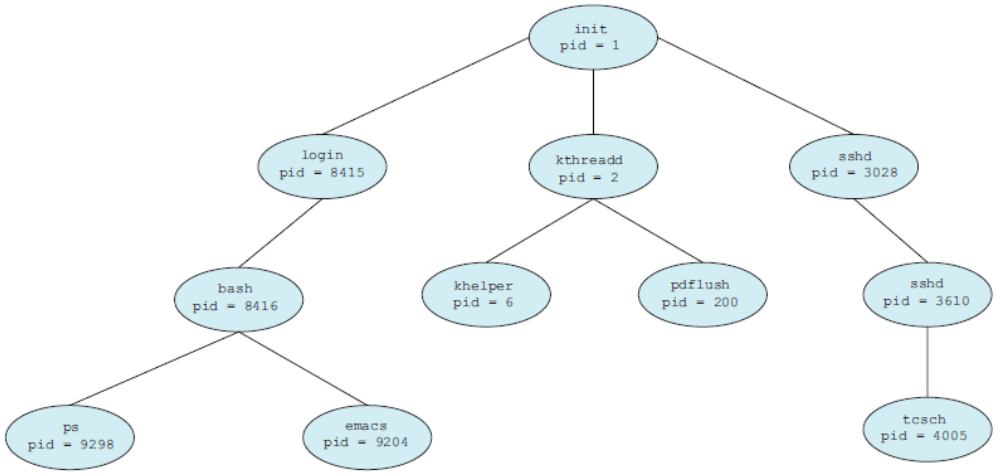
The creating process is called a **parent** process, whereas the new processes are called the **children** of that process.

When a process creates a new process, two possibilities exist **in terms of execution:**

1. The parent continues to execute concurrently with its children.
2. The parent waits until some or all of its children have terminated.

There are also two possibilities **in terms of the address space** of the new process:

1. The child process is a duplicate of the parent process.



2. The child process has a program loaded into it.

In UNIX, each process is identified by its process identifier, which is a unique integer. A new process is created by the **fork** system call.

A tree of processes on a typical Linux system.

we see two children of **init**—kthreadd and **sshd**.

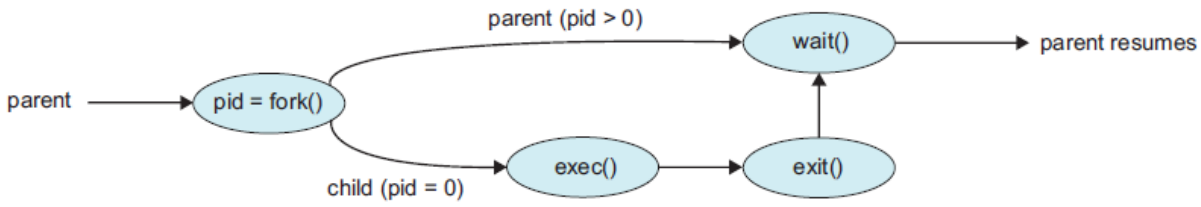
The kthreadd process is responsible for creating additional processes that perform tasks on behalf of the kernel (in this situation, khelper and pdflush).

The sshd process is responsible for managing clients that connect to the system by using ssh (which is short for *secure shell*). The login process is responsible for managing clients that directly log onto the system.

In general, when a process creates a child process, that child process will need certain resources (CPU time, memory, files, I/O devices) to accomplish its task.

A child process may be able to obtain its resources directly from the operating system, or it may be constrained to a subset of the resources of the parent process.

The parent may have to partition its resources among its children, or it may be able to share some resources (such as memory or files) among several of its children. Restricting a child process to a subset of the parent's resources prevents any process from overloading the system by creating too many child processes.



2. Process Termination

A process terminates when it finishes executing its final statement and asks the operating system to delete it by using the **exit** system call.

At that point, the process may return data (output) to its parent process (via the **wait** system call).

A process can cause the termination of another process via an appropriate system call.

A parent may terminate the execution of one of its children for a variety of reasons, such as these:

1. The child has exceeded its usage of some of the resources that it has Been allocated.

2. The task assigned to the child is no longer required.
3. The parent is exiting, and the operating system does not allow a child to continue if its parent terminates. On such systems, if a process terminates (either normally or abnormally), then all its children must also be terminated. This phenomenon, referred to as cascading **termination**, is normally initiated by the operating system.

When a process terminates, its resources are de-allocated by the operating system.

A process that has terminated, but whose parent has not yet called wait(), is known as a zombie process.

Now consider what would happen if a parent did not invoke wait() and instead terminated, thereby leaving its child processes as orphans.

CO-OPERATING PROCESS

Processes executing concurrently in the operating system may be either **independent processes** or **cooperating processes**.

A process is independent if it cannot affect or be affected by the other processes executing in the system. Any process that does not share data with any other process is independent.

A process is cooperating if it can affect or be affected by the other processes executing in the system. Clearly, any process that shares data with other processes is a cooperating process.

There are several reasons for providing an environment that allows process cooperation:

• **Information sharing.** Since several users may be interested in the same piece of information (for instance, a shared file), we must provide an environment to allow concurrent access to such information.

• **Computation speedup**. If we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Notice that such a speedup can be achieved only if the computer has multiple processing cores.

• **Modularity.** We may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.

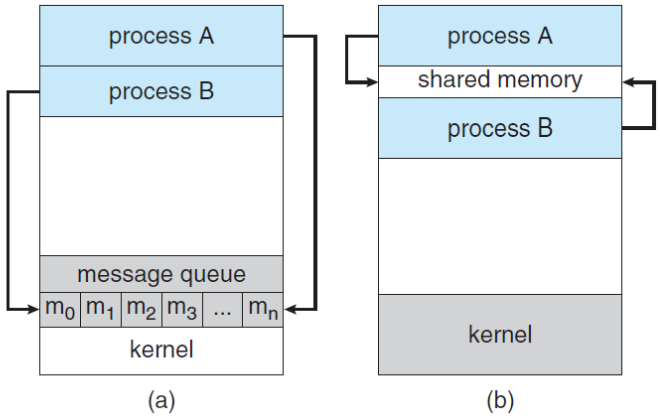
• **Convenience.** Even an individual user may work on many tasks at the same time. For instance, a user may be editing, listening to music, and compiling in parallel.

INTERPROCESS COMMUNICATION

Cooperating processes require an **Inter Process Communication (IPC)** mechanism that will allow them to exchange data and information. There are two fundamental models of interprocess communication: **shared memory** and **message passing**.

In the shared-memory model, a region of memory that is shared by cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region.

In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes.



(a) Message passing. (b) Shared memory.

Shared-Memory Systems

Interprocess communication using shared memory requires communicating processes to establish a region of shared memory.

Other processes that wish to communicate using this shared-memory segment must attach it to their address space.

Message passing

Message passing provides a mechanism to allow processes to communicate and to synchronize their actions without sharing the same address space.

1. Basic Structure:

If processes P and Q want to communicate, they must send messages to and receive messages from each other; a communication link must exist between them.

Physical implementation of the link is done through a hardware bus , network etc,

There are several methods for logically implementing a link and the operations:

1. **Direct or indirect communication**
2. **Symmetric or asymmetric communication**
3. **Automatic or explicit buffering**
4. **Send by copy or send by reference**
5. **Fixed-sized or variable-sized messages**

2. Naming

Processes that want to communicate must have a way to refer to each other.

They can use either direct or indirect communication.

1. **Direct Communication**

Each process that wants to communicate must explicitly name the recipient or sender of the communication.

A communication link in this scheme has the following properties:

i. A link is established automatically between every pair of processes that want to communicate. The processes need to know only each other's identity to communicate.

ii. A link is associated with exactly two processes.

iii. Exactly one link exists between each pair of processes.

There are two ways of addressing namely

Symmetry in addressing

Asymmetry in addressing

In symmetry in addressing, the send and receive primitives are defined as:

send(P, message) Send a message to process P

receive(Q, message) Receive a message from Q

In asymmetry in addressing , the send & receive primitives are defined as:

send (p, message) send a message to process p

receive(id, message) receive message from any process

2. Indirect Communication

With indirect communication, the messages are sent to and received from mailboxes, or ports.

The send and receive primitives are defined as follows:

send (A, message) Send a message to mailbox A.

receive (A, message) Receive a message from mailbox A.

A communication link has the following properties:

i. A link is established between a pair of processes only if both members of the pair have a shared mailbox.

ii. A link may be associated with more than two processes.

iii. A number of different links may exist between each pair of communicating processes, with each link corresponding to one mailbox.

3. Buffering

A link has some capacity that determines the number of message that can reside in it temporarily. This property can be viewed as a queue of messages attached to the link.

There are three ways that such a queue can be implemented.

Zero capacity : Queue length of maximum is 0. No message is waiting in a queue. The sender must wait until the recipient receives the message.

Bounded capacity: The queue has finite length n. Thus at most n messages can reside in it.

Unbounded capacity: The queue has potentially infinite length. Thus any number of messages can wait in it. The sender is never delayed

4. Synchronization

Message passing may be either blocking or non-blocking.

1. **Blocking Send** - The sender blocks itself till the message sent by it is received by the receiver.
2. **Non-blocking Send** - The sender does not block itself after sending the message but continues with its normal operation.
3. **Blocking Receive** - The receiver blocks itself until it receives the message.
4. **Non-blocking Receive** – The receiver does not block itself.

THREADS

Thread

A thread is a basic unit of CPU utilization; it comprises a thread ID, a program counter, a register set, and a stack.

It shares with other threads belonging to the same process its code section, data section, and other operating-system resources, such as open files and signals. Traditional (or heavyweight) process has a single thread of control.

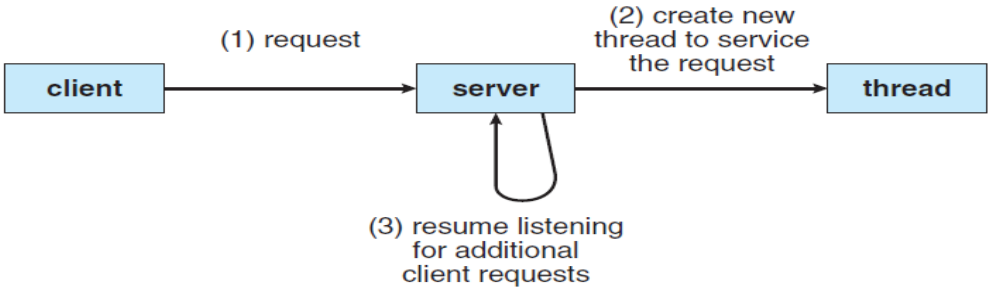
If a process has multiple threads of control, it can perform more than one task at a time.

Motivation

Most software applications that run on modern computers are multithreaded. An application typically is implemented as a separate process with several threads of control.

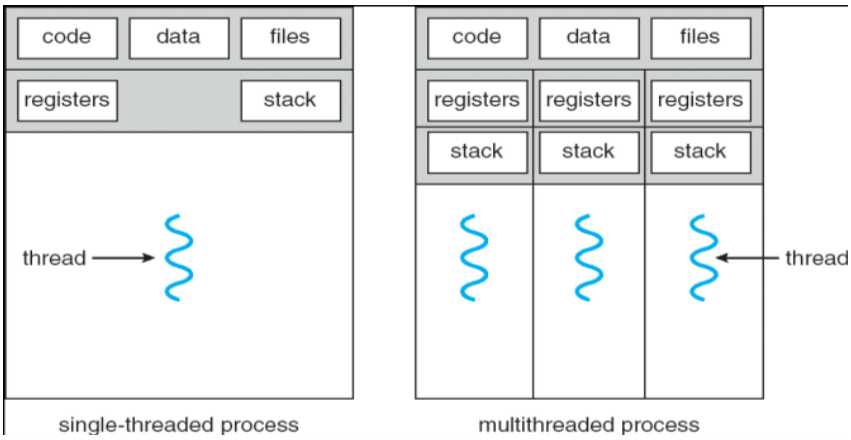
A web browser might have one thread display images or text while another thread retrieves data from the network.

A word processor may have a thread for displaying graphics, another thread for responding to keystrokes from the user, and a third thread for performing spelling and grammar checking in the background.



MULTITHREADING

Multithreading is the ability of a program or an operating system process to manage its use by more than one user at a time and to even manage multiple requests by the same user without having to have multiple copies of the programming running in the computer.



Benefits

There are four major categories of benefits to multi-threading:

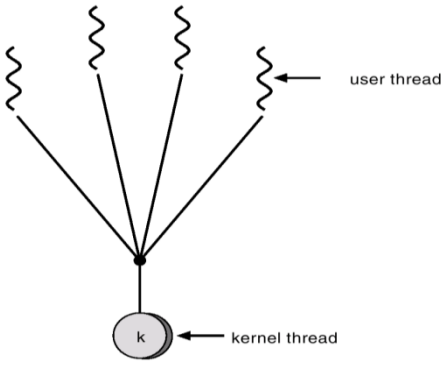
1. **Responsiveness** - One thread may provide rapid response while other threads are blocked or slowed down doing intensive calculations.
2. **Resource sharing** - By default threads share common code, data, and other resources, which allows multiple tasks to be performed simultaneously in a single address space.
3. **Economy -** Creating and managing threads (and context switches between them) is much faster than performing the same tasks for processes.
4. **Scalability**, i.e. Utilization of multiprocessor architectures - A single threaded process can only run on one CPU, no matter how many may be available, whereas the execution of a multi-threaded application may be split amongst available processors

Multithreading Models

1. Many-to-One
2. One-to-One
3. Many-to-Many

1. Many-to-One:

Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.



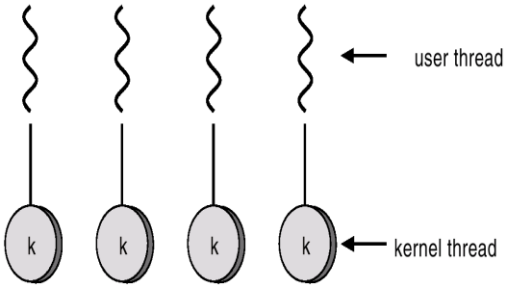
If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.

2.One-to-One:

There is one to one relationship of user level thread to the kernel level thread.

This model provides more concurrency than the many to one model.

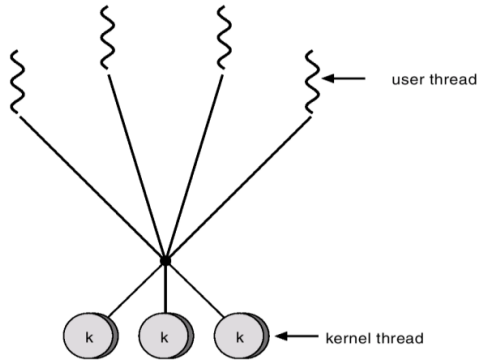
It also another thread to run when a thread



makes a blocking system call. It supports multiple thread to execute in parallel on microprocessors.

3.Many-to-Many Model:

In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers.



The number of Kernel threads may be specific to either a particular application or a particular machine.

In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.

THREADING ISSUES:

1. **fork() and exec() system calls.**

   A fork () system call may duplicate all threads or duplicate only the thread that invoked fork().

   If a thread invoke exec() system call ,the program specified in the parameter to exec will replace the entire process.

2. **Thread cancellation.**

   It is the task of terminating a thread before it has completed . A thread that is to be cancelled is called a target thread.

   There are two types of cancellation namely

   1. **Asynchronous Cancellation** – One thread immediately terminates the target thread.
   2. **Deferred Cancellation** – The target thread can periodically check if it should terminate , and does so in an orderly fashion.

3. **Signal handling**

   1. A signal is a used to notify a process that a particular event has occurred.
   2. A generated signal is delivered to the process.
      a. Deliver the signal to the thread to which the signal applies. b. Deliver the signal to every thread in the process.
      c. Deliver the signal to certain threads in the process.
      d. Assign a specific thread to receive all signals for the process.
   3. Once delivered the signal must be handled. a. Signal is handled by
      i. A default signal handler
      ii. A user defined signal handler

4. **Thread pools**

   - Creation of unlimited threads exhaust system resources such as CPU time or memory. Hence we use a thread pool.
   - In a thread pool , a number of threads are created at process startup and placed in the pool.
   - When there is a need for a thread the process will pick a thread from the pool and assign it a task.

- After completion of the task,the thread is returned to the pool.

5. **Thread specific data**

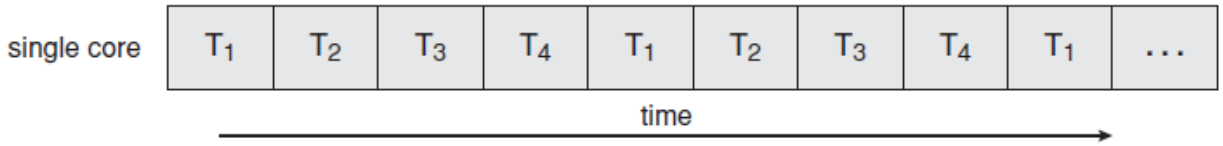
Threads belonging to a process share the data of the process. However each thread might need its own copy of certain data known as thread-specific data.

MULTICORE PRORGAMMING

Single-CPU systems evolved into multi-CPU systems. A more recent, similar trend in system design is to place multiple computing cores on a single chip.

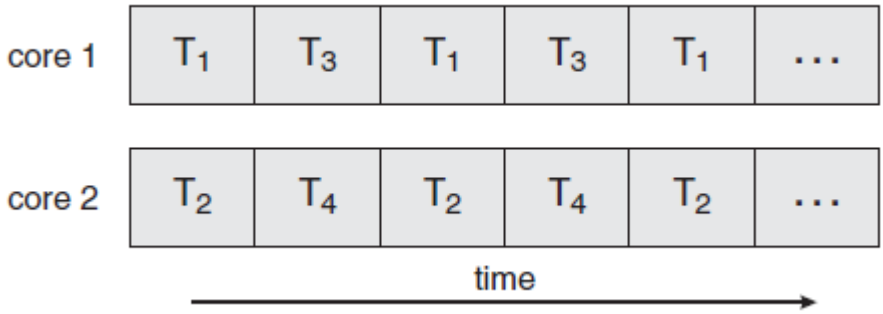
Each core appears as a separate processor to the operating system. Whether the cores appear across CPU chips or within CPU chips, we call these systems multicore or multiprocessor systems.

Multithreaded programming provides a mechanism for more efficient use of these multiple computing cores and improved concurrency.



A system is parallel if it can perform more than one task simultaneously.

A concurrent system supports more than one task by allowing all the tasks to make progress. Thus, it is possible to have concurrency without parallelism.



In general, five areas present challenges in programming for multicore systems:

1. **Identifying tasks**. This involves examining applications to find areas that can be divided into separate, concurrent tasks.

2. **Balance.** While identifying tasks that can run in parallel, programmers must also ensure that the tasks perform equal work of equal value.

3. **Data splitting**. Just as applications are divided into separate tasks, the data accessed and manipulated by the tasks must be divided to run on separate cores.

4. **Data dependency**. The data accessed by the tasks must be examined for dependencies between two or more tasks. When one task depends on data from another, programmers must ensure that the execution of the tasks is synchronized to accommodate the data dependency.

5. **Testing and debugging**. When a program is running in parallel on multiple cores, many different execution paths are possible. Testing and debugging such concurrent programs is inherently more difficult than testing and debugging single-threaded applications.

Types of Parallelism

In general, there are two types of parallelism: **data parallelism and task parallelism**.

Data parallelism focuses on distributing subsets of the same data across multiple computing cores and performing the same operation on each core.

Task parallelism involves distributing not data but tasks (threads) across multiple computing cores.

PROCESS SYNCHRONIZATION

 Concurrent access to shared data may result in data inconsistency.

 Maintaining data consistency requires mechanisms to ensure the orderly execution of cooperating processes.

 Shared-memory solution to bounded-butter problem allows at most $n - 1$ items in buffer at the same time. A solution, where all N buffers are used is not simple.

 Suppose that we modify the producer-consumer code by adding a variable *counter*, initialized to 0 and increment it each time a new item is added to the buffer

 Race condition: The situation where several processes access – and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.

 To prevent race conditions, concurrent processes must be synchronized.

THE CRITICAL-SECTION PROBLEM

Definition: Each process has a segment of code, called a critical section (CS), in which the process may be changing common variables, updating a table, writing a file, and so on.

 The important feature of the system is that, when one process is executing in its CS, no other process is to be allowed to execute in its CS.

 That is, no two processes are executing in their CSs at the same time.

 Each process must request permission to enter its CS. The section of code implementing this request is the entry section.

 The CS may be followed by an exit section.

 The remaining code is the remainder section.

Requirements to be satisfied for a Solution to the Critical-Section Problem:

1. **Mutual Exclusion -** If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress -** If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting -** A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

General structure of process P_i

```
{
        entry section
        critical section

        exit section

        remainder section

} while (1);
```

Two general approaches are used to handle critical sections in operating systems: preemptive kernels and nonpreemptive kernels.

 A preemptive kernel allows a process to be preempted while it is running in kernel mode.

 A non-preemptive kernel does not allow a process running in kernel mode to be preempted; a kernelmode process will run until it exits kernel mode, blocks, or voluntarily yields control of the CPU.

MUTEX LOCKS

 Operating-systems designers build software tools to solve the critical-section problem. The simplest of these tools is the mutex lock.

 We use the mutex lock to protect critical regions and thus prevent race conditions.

 That is, a process must acquire the lock before entering a critical section; it releases the lock when it exits the critical section.

 The acquire()function acquires the lock, and the release() function releases the lock.
Solution to the critical-section problem using mutex locks.

```
do {
        acquire lock
        critical section
        release lock
        remainder section
} while (true);
```

 A mutex lock has a boolean variable available whose value indicates if the lock is available or not.
 If the lock is available, a call to acquire() succeeds, and the lock is then considered unavailable.
 A process that attempts to acquire an unavailable lock is blocked until the lock is released.

 The definition of acquire() is as follows:

```
        acquire()
        {
                while (!available); /* busy wait */
                available = false;;
        }
```

 The definition of release() is as follows:

```
                release()
                {
                        available = true;
                }
```

 Calls to either acquire() or release() must be performed atomically. Thus, mutex locks are often implemented using one of the hardware mechanisms.

Disadvantage of the implementation given here is that it requires busy waiting.

 While a process is in its critical section, any other process that tries to enter its critical section must loop continuously in the call to acquire().

 In fact, this type of mutex lock is also called a spinlock because the process "spins" while waiting for the lock to become available.

 This continual looping is clearly a problem in a real multiprogramming system, where a single CPU is shared among many processes. Busy waiting wastes CPU cycles that some other process might be able to use productively.

SEMAPHORES

 A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations:
wait() and
signal().
 The wait() operation was originally termed P (from the Dutch proberen, "to test"); signal() was originally called V (from verhogen, "to increment").

 The definition of wait() is as follows:

```
wait(S)
{
while (S <= 0); // busy wait
S--;
}
```

 The definition of signal() is as follows:

```
signal(S)
{
S++;
}
```

Semaphore Usage

 The value of a counting semaphore can range over an unrestricted domain. The value of a binary semaphore can range only between 0 and 1.

 Binary semaphores behave similarly to mutex locks.

 On systems that do not provide mutex locks, binary semaphores can be used instead for providing mutual exclusion.

 Counting semaphores can be used to control access to a given resource consisting of a finite number of instances.

 The semaphore is initialized to the number of resources available.

 Each process that wishes to use a resource performs a **wait()** operation on the semaphore (thereby decrementing the count).

 When a process releases a resource, it performs a **signal()** operation (incrementing the count).

When the count for the semaphore goes to 0, all resources are being used. After that, processes that wish to use a resource will block until the count becomes greater than 0

 We can also use semaphores to solve various synchronization problems.

 For example, consider two concurrently running processes: P1 with a statement S1 and P2 with a statement S2. Suppose we require that S2 be executed only after S1 has completed. We can implement this scheme readily by letting P1 and P2 share a common semaphore synch, initialized to 0. In process P1, we insert the statements

```
S1;
signal(synch);
```

In process P2, we insert the statements

```
wait(synch);
S2;
```

 Because synch is initialized to 0, P2 will execute S2 only after P1 has invoked signal(synch), which is after statement S1 has been executed.

Semaphore Implementation

 To overcome the need for busy waiting, we can modify the definition of the wait() and

signal() operations as follows: When a process executes the wait() operation and finds that the semaphore value is not positive, it must wait.

 Rather than engaging in busy waiting, the process can block itself.

 The block operation places a process into a waiting queue associated with the semaphore, and the state of the process is switched to the waiting state.

 Then control is transferred to the CPU scheduler, which selects another process to execute.
 A process that is blocked, waiting on a semaphore S, should be restarted when some other process executes a signal() operation.

 The process is restarted by a wakeup() operation, which changes the process from the waiting state to the ready state.
 The process is then placed in the ready queue. (The CPU may or may not be switched from the running process to the newly ready process, depending on the CPU-scheduling algorithm.)

 To implement semaphores under this definition, we define a semaphore as follows:

```
typedef struct
{
int value;
struct process *list;
} semaphore;
```

 Each semaphore has an integer value and a list of processes list.

 When a process must wait on a semaphore, it is added to the list of processes.
A signal() operation removes one process from the list of waiting processes and awakens that process.

 The wait() semaphore operation can be defined as

```
wait(semaphore *S)
{
S->value--;
if (S->value < 0)
{
add this process to S->list;
block();
}
}
```

 The signal() semaphore operation can be defined as

```
signal(semaphore *S)
{
S->value++;
if (S->value <= 0)
{
remove a process P from S->list;
wakeup(P);
}
```

}

 The block() operation suspends the process that invokes it.

 The wakeup(P) operation resumes the execution of a blocked process P.

Deadlocks and Starvation

 The implementation of a semaphore with a waiting queue may result in a situation where two or more processes are waiting indefinitely for an event that can be caused only by one of the waiting processes

.

 When such a state is reached, these processes are said to be deadlocked

 To illustrate this, consider a system consisting of two processes, P0 and P1, each accessing two semaphores, S and Q, set to the value 1:

P0	P1
wait(S);	wait(Q);
wait(Q);	wait(S);
. .	..
. .	..
. .	..
signal(S);	signal(Q);
signal(Q);	signal(S);

 Suppose that P0 executes wait(S) and then P1 executes wait(Q).When P0 executes wait(Q), it must wait until P1 executes signal(Q).

 Similarly, when P1 executes wait(S), it must wait until P0 executes signal(S).

 Since these signal() operations cannot be executed, P0 and P1 are deadlocked.

 We say that a set of processes is in a deadlocked state when every process in the set is waiting for an event that can be caused only by another process in the set.

 Another problem related to deadlocks is indefinite blocking or starvation, a situation in which processes wait indefinitely within the semaphore.

 Indefinite blocking may occur if we remove processes from the list associated with a semaphore in LIFO (last-in, first-out) order.

Priority Inversion

 A scheduling challenge arises when a higher-priority process needs to read or modify kernel data that are currently being accessed by a lower-priority process—or a chain of lower-priority processes.

 The kernel data are typically protected with a lock, the higher-priority process will have to wait for a lower-priority one to finish with the resource.

 The situation becomes more complicated if the lower-priority process is preempted in favor

of another process with a higher priority.

 This problem is known as priority inversion. It occurs only in systems with more than two priorities, so one solution is to have only two priorities.

 Typically these systems solve the problem by implementing a priority-inheritance protocol. According to this protocol, all processes that are accessing resources needed by a higher-priority process inherit the higher priority until they are finished with the resources in question.

 When they are finished, their priorities revert to their original values. In the example above, a priority-inheritance protocol would allow process L to temporarily inherit the priority of process.

CLASSIC PROBLEMS OF SYNCHRONIZATION

1. Bounded Buffer Problem
2. Reader Writer Problem
3. Dining Philosopher's Problem

The Bounded-Buffer Problem

 We assume that the pool consists of n buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for accesses to the buffer pool and is initialized to the value 1.

 The empty and full semaphores count the number of empty and full buffers.

 The semaphore empty is initialized to the value n.

 The semaphore full is initialized to the value 0.

The producer and consumer processes share the following data structures:

```
int n;
semaphore mutex = 1;
semaphore empty = n;
semaphore full = 0
```

The structure of the producer process.

```
do {
. . .
/* produce an item in next produced */
. . .
wait(empty);
wait(mutex);
. . .
/* add next produced to the buffer */
. . .
signal(mutex);
signal(full);
```

```
} while (true);
```

The structure of the consumer process.

```
do {
wait(full);
wait(mutex);
. . .
/* remove an item from buffer to next consumed */
. . .
signal(mutex);
signal(empty);
. . .
/* consume the item in next consumed */
. . .
} while (true);
```

 We can interpret this code as the producer producing full buffers for the consumer or as the consumer producing empty buffers for the producer.

Reader Writer Problem

The R-W problem is another classic problem for which design of synchronization and concurrency mechanisms can be tested. The producer/consumer is another such problem; the dining philosophers is another.

Definition

- There is a data area that is shared among a number of processes.
- Any number of readers may simultaneously write to the data area.
- Only one writer at a time may write to the data area.
- If a writer is writing to the data area, no reader may read it.
- If there is at least one reader reading the data area, no writer may write to it.
- Readers only read and writers only write
- A process that reads and writes to a data area must be considered a writer (consider producer or consumer)

In the solution to the first readers–writers problem, the reader processes share the following data structures:

```
semaphore rw mutex = 1;
semaphore mutex = 1;
int read count = 0;
```

 The semaphores mutex and rw mutex are initialized to 1; read count is initialized to 0.
 The semaphore rw mutex is common to both reader and writer processes.
 The mutex semaphore is used to ensure mutual exclusion when the variable read count is updated.
 The read count variable keeps track of how many processes are currently reading the object.
 The semaphore rw mutex functions as a mutual exclusion semaphore for the writers.

The structure of a writer process.

```
do {
```

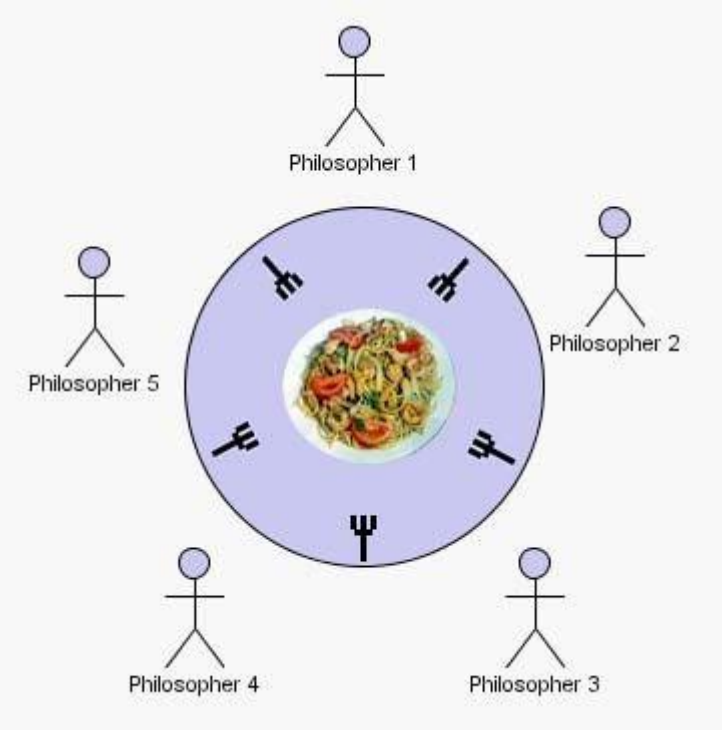
```
        wait(rw mutex);
        . . .
        /* writing is performed */
        . . .
        signal(rw mutex);
        } while (true);
```

The structure of a reader process.

```
        do {
        wait(mutex);
        readcount++;
        if (read count == 1)
        wait(rw mutex);
        signal(mutex);
        . . .
        /* reading is performed */
wait(mutex);
read count--;
if (read count == 0)
signal(rw mutex);
signal(mutex);
} while (true);
```

Dining Philosophers Problem

Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and a bowl of rice in the middle.



At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right.

When a philosopher wants to think, he keeps down both chopsticks at their original place.

- When a philosopher thinks, he does not interact with his others.
- From time to time, a philosopher gets hungry and tries to pick up the two forks that are closest to him (the forks that are between him and his left and right neighbors).
- A philosopher may pick up only one fork at a time. Obviously, he cannot pick up a fork that is already in the hand of a neighbor.
- When a hungry philosopher has both his forks at the same time, he eats without releasing his forks.
- When he is finished eating, he puts down both of his forks and starts thinking again.

Solution:

From the problem statement, it is clear that a philosopher can think for an indefinite amount of time. But when a philosopher starts eating, he has to stop at some point of time. The philosopher is in an endless cycle of thinking and eating.

An array of five semaphores, **stick[5]**, for each of the five chopsticks.

The code for each philosopher looks like:

```
while(TRUE) {
wait(stick[i]);
wait(stick[(i+1) % 5]);  // mod is used because if i=5, next
                         // chopstick is 1 (dining table is circular)
/* eat */
signal(stick[i]);
signal(stick[(i+1) % 5]);
/* think */
}
```

When a philosopher wants to eat the rice, he will wait for the chopstick at his left and picks up that chopstick. Then he waits for the right chopstick to be available, and then picks it too. After eating, he puts both the chopsticks down.

But if all five philosophers are hungry simultaneously, and each of them pickup one chopstick, then a deadlock situation occurs because they will be waiting for another chopstick forever.

The possible solutions for this are:

1) A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.

2) Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating and eventually, two chopsticks will be available. In this way, deadlocks can be avoided.

MONITORS

Definition: Monitor is a high-level language construct with a collection of procedures, variables, and data structures that are all grouped together in a special kind of module or package.

 Processes may call the procedures in a monitor whenever they want to, but they cannot directly access the monitor's internal data structures from procedures declared outside the monitor.

 Monitors have an important property that makes them useful for achieving mutual exclusion: only one process can be active in a monitor at any instant.

Monitor Usage

 A monitor type presents a set of programmer-defined operations that are provided mutual exclusion within the monitor.

 The monitor type also contains the declaration of variables whose values define the state of an instance of that type, along with the bodies of procedures or functions that operate on those variables.

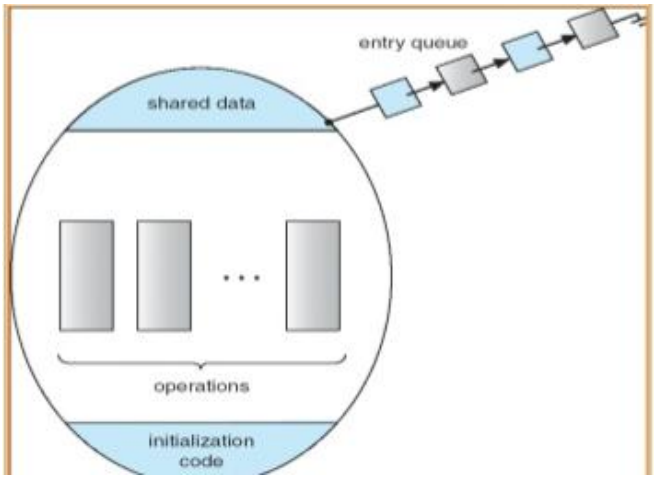
```
monitor monitor name
{
/* shared variable declarations */
function P1 ( . . . ) {
. . .
}
function P2 ( . . . ) {
. . .
}
.
.
function Pn ( . . . ) {
. . .
}
initialization code ( . . . ) {
. . .
}
}
```

 The representation of a monitor type cannot be used directly by the various processes. Thus, a procedure defined within a monitor can access only those variables declared locally within the monitor and its formal parameters.

 Similarly, the local variables of a monitor can be accessed by only the local procedures.

 The monitor construct ensures that only one process at a time can be active within the monitor.

Schematic view of a Monitor



The monitor construct is not sufficiently powerful for modeling some synchronization schemes.
 For this purpose, we need to define additional synchronization mechanisms. These mechanisms are provided by the condition construct condition x, y;

The only operations that can be invoked on a condition variable are wait() and signal(). The operation

x.wait();

means that the process invoking this operation is suspended until another process invokes

x.signal();

The x.signal() operation resumes exactly one suspended process.

A monitor solution to the dining-philosopher problem.

```
monitor DiningPhilosophers
{
enum {THINKING, HUNGRY, EATING} state[5];
condition self[5];
void pickup(int i)
{
state[i] = HUNGRY;
test(i);
if (state[i] != EATING)
self[i].wait();
}
void putdown(int i)
{
state[i] = THINKING;
test((i + 4) % 5);
```

```
test((i + 1) % 5);
}
void test(int i)
{
if ((state[(i + 4) % 5] != EATING) && (state[i] == HUNGRY) && (state[(i + 1) % 5] != EATING))
{
state[i] = EATING;
self[i].signal();
}
}
initialization code()
{
for (int i = 0; i < 5; i++)
state[i] = THINKING;
}
```

CPU SCHEDULING

CPU scheduling is the basis of multi-programmed operating systems.

By switching the CPU among processes, the operating system can make the computer more productive.

Basic Concepts

- The objective of multi-programming is to have some process running at all times, to maximize CPU utilization.
- For a Uni-processor system, there will never be more than one running process.
- Scheduling is a fundamental operating system function.
- The idea of multi-programming is to execute a process until it must wait, typically for the completion of some I/O request.
- The CPU is one of the primary computer resources.
- The CPU scheduling is central to operating system design.

Cpu Scheduler

- When the CPU becomes idle, the operating system must select on of the processes in the ready queue to be executed.
- The selection process is carried out by the short-term scheduler (CPU scheduler)
- The scheduler selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- A ready queue may be implemented as a FIFO queue, a priority queue, a tree or simply an unordered link list.
- All the processes in the ready queue are lined up waiting for a chance to run on the CPU.

CPU scheduling decisions may take place when a process.

1. Switches from running to waiting state
2. Switches from running to ready state
3. Switches from waiting to ready

4. Terminates

Scheduling under 1 and 4 is **nonpreemptive.**
All other scheduling is **preemptive.**

Nonpreemptive Scheduling➔A scheduling discipline is non preemptive if, once a process has been given the CPU, the CPU cannot be taken away from that process.

Preemptive Scheduling ➔ A scheduling discipline is preemptive if, once a process has been given the CPU can taken away.

Dispatcher

Dispatcher is a module that gives control of the CPU to the process selected by the short-term scheduler. This function involves the following:

- switching context.
- switching to user mode.
- jumping to the proper location in the user program to restart that program.

Dispatch latency – The time taken for the dispatcher to stop one process and start another running.

Scheduling criteria

1. **CPU utilization** – keep the CPU as busy as possible Throughput – # of processes that complete their execution per time unit .
2. **Turnaround time** – amount of time to execute a particular process
3. **Waiting time** – amount of time a process has been waiting in the ready queue
4. **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)
5. **Throughput** – The number of processes that complete their execution per time unit.

Best Algorithm consider following:
- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Formulas to calculate Turn-around time & waiting time is:
Waiting time = Finishing Time – (CPU Burst time + Arrival Time)
Turnaround time = Waiting Time + Burst Time

Scheduling Algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms.

1. First-Come, First-Served (FCFS) Scheduling
2. Shortest-Job-First (SJF) Scheduling
3. Priority Scheduling
4. Round Robin(RR) Scheduling

First-Come, First-Served (FCFS) Scheduling algorithm.

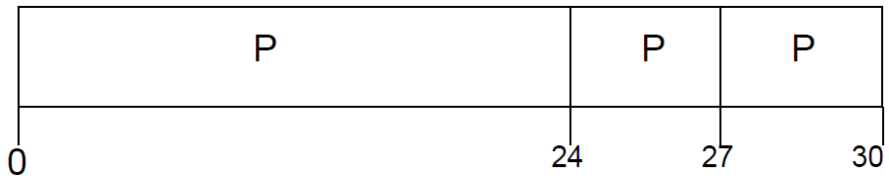
- This is the simplest CPU-scheduling algorithm.
- According to this algorithm, the process that requests the CPU first is allocated the CPU first.
- The implementation of FCFS is easily managed with a FIFO queue.
- When a process enters the ready queue, its PCB is linked onto the tail of the queue.
- When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue.

Example Problem

Consider the following set of processes that arrive at time 0, with the length of the CPU burst time given in milliseconds:

Process	Burst Time(ms)
P1	24
P2	3
P3	3

Suppose that the processes arrive in the order: P1 , P2 , P3 The Gantt Chart:



Waiting time

- Waiting time for P1 = 0; P2 = 24; P3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17 ms.

Turnaround Time = Waiting Time + Burst Time

- Turnaround Time for P1 = (0+24)=24; P2 = (24+3)=27; P3 = (27+3)=30
- Average Turnaround Time = (24+27+30)/3 = 27 ms

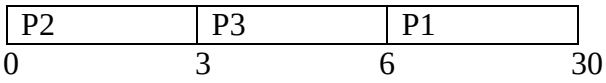
Shortest-Job-First (SJF) Scheduling

- This algorithm associates with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- When the CPU is available, it is assigned to the process that has the smallest next CPU burst. It is also called as shortest next CPU burst.

 If two processes have the same length next CPU burst, FCFS scheduling is used to break the tie.

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart



Waiting time

For P1=6,P2=0,P3=3

Average Waiting Time=(6+0+3)/3=3 ms.

Turnaround Time = Waiting Time + Burst Time

Turnaround Time for P1=(6+24)=30,P2=(0+3)=3,P3=(3+3)=6

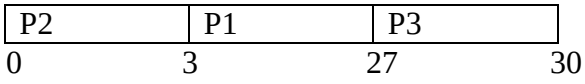
Average Turnaround Time=(30+3+6)/3=13ms

Priority Scheduling

- The SJF algorithm is a special case of the general priority-scheduling algorithm.
- A priority number (integer) is associated with each process and the CPU is allocated to the process with the highest priority.
- Equal-priority processes are scheduled in FCFS order.
- The CPU is allocated to the process with the highest priority (smallest integer º highest priority) .

Process	Burst Time	Priority
P1	24	2
P2	3	1
P3	3	3

Gantt Chart



Waiting time

For P1=3,P2=0,P3=27

Average Waiting Time=(3+0+27)/3=10ms

Turnaround Time = Waiting Time + Burst Time

Turnaround Time for P1=(3+24)=27,P2=(0+3)=3,P3=(27+3)=30

Average Turn Around Time=(27+3+30)/3=20ms.

Round robin scheduling

- Round robin scheduling is designed especially for time-sharing systems.
- It is similar to FCFS, but preemption is added to switch between processes.
- Each process gets a small unit of CPU time called a time quantum or

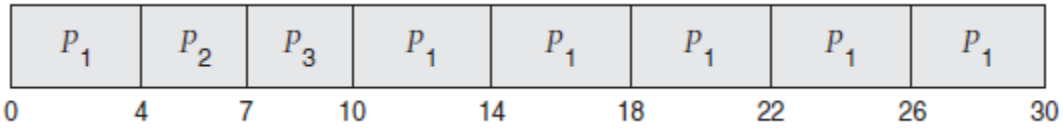
time slice.

- To implement RR scheduling, the ready queue is kept as a FIFO queue of processes. New processes are added to the tail of the ready queue. The CPU scheduler picks the first process from the ready queue, sets a timer to interrupt after 1 time quantum and dispatches the process.

- If the CPU burst time is less than the time quantum, the process itself will release the CPU voluntarily. Otherwise, if the CPU burst of the currently running process is longer than the time quantum a context switch will be executed and the process will be put at the tail of the ready queue.

Process	**Burst Time**
P_1	24
P_2	3
P_3	3

Gantt Chart



Waiting time

Average waiting time = [6+4+7]/3 = 17/3 =5.66

Turnaround Time = Waiting Time + Burst Time

Turnaround Time for P1=(6+24)=30,P2=(4+3)=7,P3=(7+3)=10

Average Turnaround Time=(30+7+10)/3=15.6ms.

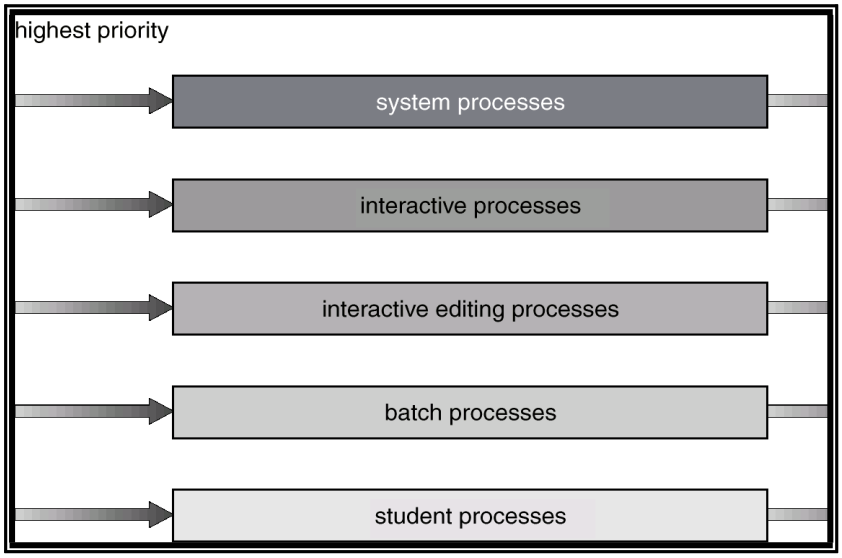
Multilevel Queue Scheduling

- It partitions the ready queue into several separate queues .
- The processes are permanently assigned to one queue, generally based on some property of the process, such as memory size, process priority, or process type.
- There must be scheduling between the queues, which is commonly implemented as a fixed-priority preemptive scheduling.
- For example the foreground queue may have absolute priority over the background queue.

Example : of a multilevel queue scheduling algorithm with five queues

1. System processes
2. Interactive processes
3. Interactive editing processes
4. Batch processes
5. Student processes

Each queue has absolute priority over lower-priority queue.

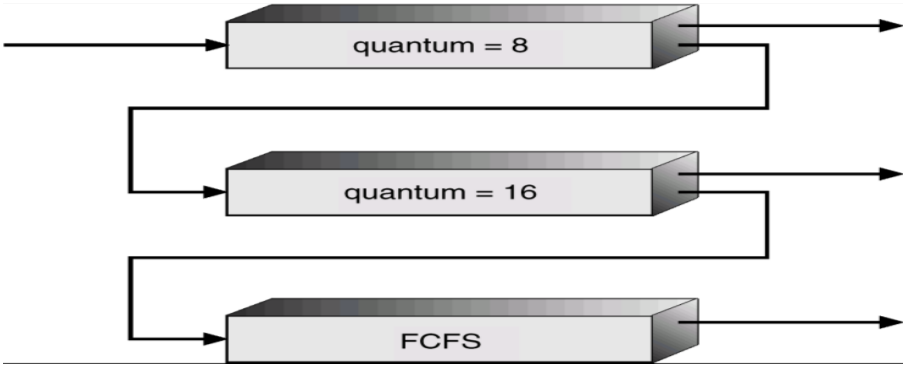


Multilevel Feedback Queue Scheduling

- It allows a process to move between queues.
- The idea is to separate processes with different CPU-burst characteristics.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- This scheme leaves I/O-bound and interactive processes in the higher- priority queues.
- Similarly, a process that waits too long in a lower priority queue may be moved to a higher-priority queue.
- This form of aging prevents starvation.

Example:

- Consider a multilevel feedback queue scheduler with three queues, numbered from 0 to 2 .



- The scheduler first executes all processes in queue 0.
- Only when queue 0 is empty will it execute processes in queue 1.
- Similarly, processes in queue 2 will be executed only if queues 0 and 1 are empty.
- A process that arrives for queue 1 will preempt a process in queue 2.
- A process that arrives for queue 0 will, in turn, preempt a process in queue 1.

- A multilevel feedback queue scheduler is defined by the following parameters:

  1. The number of queues
  2. The scheduling algorithm for each queue
  3. The method used to determine when to upgrade a process to a higher priority queue
  4. The method used to determine when to demote a process to a lower-priority queue
  5. The method used to determine which queue a process will enter when that process needs service

Multiple Processor Scheduling

- If multiple CPUs are available, the scheduling problem is correspondingly more complex.
- If several identical processors are available, then load-sharing can occur.
- It is possible to provide a separate queue for each processor.
- In this case however, one processor could be idle, with an empty queue, while another processor was very busy.
- To prevent this situation, we use a common ready queue.
- All processes go into one queue and are scheduled onto any available processor.
- In such a scheme, one of two scheduling approaches may be used.

  1. **Self Scheduling** - Each processor is self-scheduling. Each processor examines the common ready queue and selects a process to execute. We must ensure that two processors do not choose the same process, and that processes are not lost from the queue.
  2. **Master – Slave Structure** - This avoids the problem by appointing one processor as scheduler for the other processors, thus creating a master-slave structure.

Real-Time Scheduling

- Real-time computing is divided into two types.

  1. Hard real-time systems
  2. Soft real-time systems

Hard real-time systems

- Hard RTS are required to complete a critical task within a guaranteed amount of time.
- Generally, a process is submitted along with a statement of the amount of time in which it needs to complete or perform I/O.
- The scheduler then either admits the process, guaranteeing that the process will complete on time, or rejects the request as impossible. This is known as **resource reservation**.

Soft real-time systems

- Soft real-time computing is less restrictive. It requires that critical processes recieve

priority over less fortunate ones.

- The system must have priority scheduling, and real-time processes must have the highest priority.
- The priority of real-time processes must not degrade over time, even though the priority of non-real-time processes may.
- Dispatch latency must be small. The smaller the latency, the faster a real- time process can start executing.
- The high-priority process would be waiting for a lower-priority one to finish. This situation is known as **priority inversion**.

DEAD LOCK

Definition:
A process request resources, if the resources are not available at that time, the process enters in to a wait state. It may happen that waiting processes will never again change the state, because the resources they have requested are held by other waiting processes. *This situation is called as **dead lock.***

System Model

- A system consists of a finite number of resources to be distributed among a number of competing processes.

- The resources may be partitioned into several types (or classes), each consisting of some number of identical instances.

- CPUcycles, files,and I/O devices (such as printers and DVD drives) are examples of resource types.

A process must request a resource before using it and must release the resource after using it.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:
1.Request. The process requests the resource. If the request cannot be granted immediately then the requesting process must wait until it can acquire the resource.
2. Use. The process can operate on the resource
3. Release. The process releases the resource.

Deadlock Characterizations:-
In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

Necessary Conditions for Deadlock:-

A dead lock situation can arise if the following four conditions hold simultaneously in a system.
1) MUTUAL EXCLUSION:- At least one resource must be held in a on-sharable mode. i.e only

one process can hold this resource at a time . other requesting processes should wait till it is released.

2) HOLD & WAIT:- there must exist a process that is holding at least one resource and is waiting to acquire additional resources that are currently being held by other processes.

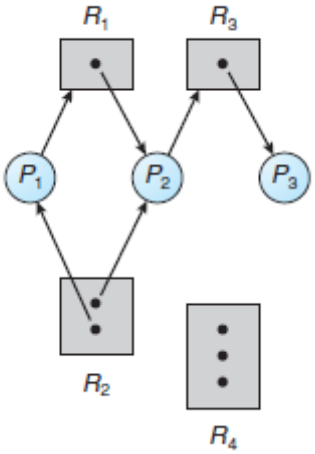
3) NO PREEMPTION:- Resources cannot be preempted, that is a resource can be released voluntarily by the process holding it, after that process has completed its task.

4) CIRCULAR WAIT:- There must exist a set {p0,p1,p1….pn} of waiting processes such that p0 is waiting for a resource that is held by the p1, p1 is waiting for the resource that is held by the p2…. And so on. pn is waiting for a resource that is held by the p0.

Resource-Allocation Graph

A deadlock can be described in terms of a directed graph called system resource-allocation graph.

- A set of vertices V and a set of edges E.
  - V is partitioned into two types:
  - P = {P1, P2, …, Pn}, the set consisting of all the processes in the system.
  - R = {R1, R2, …, Rm}, the set consisting of all resource types in the system.
    - request edge – directed edge Pi → Rj
    - assignment edge – directed edge Rj → P



The resource-allocation graph depicts the following situation.

The sets P, R, and E:

- P = {P1, P2, P3}
- R = {R1, R2, R3, R4}
- E = {P1 → R1, P2 → R3, R1 → P2, R2 → P2, R2 → P1, R3 → P3}

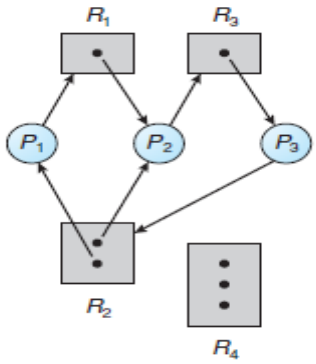
Resource instances:

- One instance of resource type R1
- Two instances of resource type R2
- One instance of resource type R3
- Three instances of resource type R4

Process states:

- Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.
- Process P2 is holding an instance of R1 and an instance of R2 and is waiting for an instance of R3.
- Process P3 is holding an instance of R3.

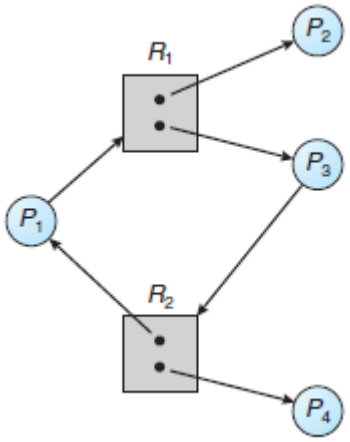
Resource-allocation graph with a deadlock.



- Processes P1, P2, and P3 are deadlocked. Process P2 is waiting for the resource R3, which is held by process P3. Process P3 is waiting for either process P1 or process P2 to release resource R2. In addition, process P1 is waiting for process P2 to release resource R1.
- We also have a cycle: P1 → R1 → P3 → R2 → P1
- If the graph contains no cycles, then no process in the system is deadlocked.

If the graph does contain a cycle, then a deadlock may exist.

Resource-allocation graph with a cycle but no deadlock.



Methods for Handling Deadlocks

We can deal with the deadlock problem in one of three ways:

1. We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state
2. We can allow the system to enter a deadlocked state, detect it, and recover.
3. We can ignore the problem altogether and pretend that deadlocks never occur in the system.

The third solution is the one used by most operating systems, including Linux and Windows.

- **Deadlock prevention** provides a set of methods to ensure that at least one of the necessary conditions cannot hold.
- **Deadlock avoidance** requires that the operating system be given additional information in advance concerning which resources a process will request and use during its lifetime.

DEADLOCK PREVENTION

- For a deadlock to occur, each of the four necessary conditions must hold.
- By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

1. Mutual Exclusion

– not required for sharable resources; must hold for non-sharable resources.
– For example, a printer cannot be simultaneously shared by several processes.
– A process never needs to wait for a sharable resource.

2. Hold and Wait

– must guarantee that whenever a process requests a resource, it does not hold any other resources.

– One protocol requires each process to request and be allocated all its resources before it begins execution,

– Or another protocol allows a process to request resources only when the process has none. So, before it can request any additional resources, it must release all the resources that it is currently allocated.

3. Denying No preemption

– If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released.

– Preempted resources are added to the list of resources for which the process is waiting.

– Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

4. Denying Circular wait

- Impose a total ordering of all resource types and allow each process to request for resources in an increasing order of enumeration.
- Let R = {R1,R2,...Rm} be the set of resource types.
- Assign to each resource type a unique integer number.
- If the set of resource types R includes tapedrives, disk drives and printers.

  F(tapedrive)=1,
  F(diskdrive)=5,
  F(Printer)=12.
- Each process can request resources only in an increasing order of enumeration.

DEADLOCK AVOIDANCE

An alternative method for avoiding deadlocks is to require additional information about how resources are to be requested.

Each request requires that in making this decision the system consider

- the resources currently available,
- the resources currently allocated to each process,
- the future requests and releases of each process.

A deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that a circular-wait condition can never exist.

The resource-allocation state is defined by the number of available and allocated resources and the maximum demands of the processes.

Safe State

• When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state.

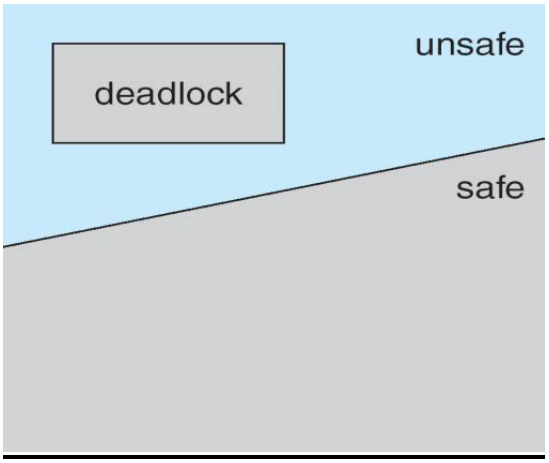
• System is in safe state if there exists a sequence <P1, P2, …, Pn> of ALL the processes is the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j < i.

• That is:

– If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.

– When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.

– When Pi terminates, Pi +1 can obtain its needed resources, and so on.



Banker's Algorithm

 The resource-allocation-graph algorithm is not applicable to a resource allocation system with multiple instances of each resource type.

 The name was chosen because the algorithm could be used in a banking system to ensure that the bank never allocated its available cash in such a way that it could no longer satisfy the needs of all its customers.

Multiple instances.

Each process must a priori claim maximum use.

- When a process requests a resource it may have to wait.
- When a process gets all its resources it must return them in a finite amount of time.
- Let n = number of processes, and m = number of resources types.
    1. **Available:** indicates the number of available resources of each type.
    2. **Max:** Max[i, j]=k then process Pi may request at most k instances of resource type Rj
    3. **Allocation :** Allocation[i. j]=k, then process Pi is currently allocated K instances of resource type Rj
    4. **Need :** if Need[i, j]=k then process Pi may need K more instances of resource type Rj ,Need [i, j]=Max[i, j]-Allocation[i, j]

Need [i,j] = Max[i,j] – Allocation [i,j].

Safety algorithm

1. Initialize work := available and Finish [i]:=false for i=1,2,3 .. n

2.Find an i such that both

a. Finish[i]=false

b. Needi<= Work if no such i exists, goto step 4

3. work :=work+ allocationi; Finish[i]:=true goto step 2

4. If finish[i]=true for all i, then the system is in a safe state

Example:

Given the following state for the Banker's Algorithm.

5 processes P0 through P4

3 resource types A (6 instances), B (9 instances) and C (5 instances).

Snapshot at time T0:

	Max	*Allocation*
	A B C	*A B C*
P_0	6 7 3	1 1 1
P_1	2 2 2	1 1 2
P_2	2 6 3	0 3 0
P_3	2 2 2	2 1 1
P_4	4 6 3	1 1 1

a) Calculate the available vector.

b) Calculate the Need matrix.

c) Is the system in a safe state? If so, show one sequence of processes which allows the system to complete. If not, explain why.

d) Given the request (1, 2, 0) from Process P2. Should this request be granted? Why or why not?

a) Calculate the available vector.

Available
A B C
1 2 0

b) Calculate the Need matrix.

	Need
	A B C
P_0	5 6 2
P_1	1 1 0
P_2	2 3 3
P_3	0 1 1
P_4	3 5 2

c) Is the system in a safe state? If so, show one sequence of processes which allows the system to complete. If not, explain why.

1. Initialize the *Work* and *Finish* vectors.

Work = *Available* = (1, 2, 0)

Finish = (false, false, false, false, false)

2. Find index *i* such that *Finish*[*i*] = false and *Needi* <= *Work*

i	$Work = Work + Allocation_i$	*Finish*
1	(1, 2, 0) + (1, 1, 2) = (2, 3, 2)	(false, true, false, false, false)
3	(2, 3, 2) + (2, 1, 1) = (4, 4, 3)	(false, true, false, true, false)
2	(4, 4, 3) + (0, 3, 0) = (4, 7, 3)	(false, true, true, true, false)
4	(4, 7, 3) + (1, 1, 1) = (5, 8, 4)	(false, true, true, true, true)
0	(5, 8, 4) + (1, 1, 1) = (6, 9, 5)	(true, true, true, true, true)

3. Since *Finish*[*i*] = true for all *i*, hence the system is in a safe state. The sequence of processes which allows the system to complete is P1, P3, P2, P4, P0**.**

d) Given the request (1, 2, 0) from Process P2. Should this request be granted? Why or why not?

1. Check that *Request2* <= *Need2*.

   Since (1, 2, 0) <= (2, 3, 3), hence, this condition is satisfied.

2. Check that *Request2* <= *Available*.

   Since (1, 2, 0) <= (1, 2, 0), hence, this condition is satisfied.

3. Modify the system's state as follows:

   Available = *Available* – *Request2* = (1, 2, 0) – (1, 2, 0) = (0, 0, 0)

   Allocation2 = *Allocation2* + *Request2* = (0, 3, 0) + (1, 2, 0) = (1, 5, 0)

   Need2 = *Need2* – *Request2* = (2, 3, 3) – (1, 2, 0) = (1, 1, 3)

4. Apply the safety algorithm to check if granting this request leaves the system in a safe state.

   1. Initialize the *Work* and *Finish* vectors.

      Work = *Available* = (0, 0, 0)

      Finish = (false, false, false, false, false)

   2. At this point, there does not exist an index *i* such that *Finish*[*i*] = false and *Needi* <= *Work*.

   Since *Finish*[*i*] ≠ true for all *i*, hence the system is not in a safe state.

   Therefore, this request from process *P2* should not be granted.

Resource-Request Algorithm

Let Requesti be the request vector for process Pi . If Requesti [j] == k, then process Pi wants k instances of resource type Rj . When a request for resources is made by process Pi , the following actions are taken:

1. If Requesti ≤Needi , go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2. If Requesti ≤ Available, go to step 3. Otherwise, Pi must wait, since the resources are not available.

3. Have the system pretend to have allocated the requested resources to process Pi by modifying the state as follows:

   Available = Available–Requesti ;

   Allocationi = Allocationi + Requesti ;

Needi = Needi –Requesti ;

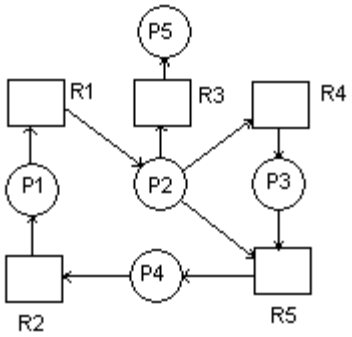
DEADLOCK DETECTION

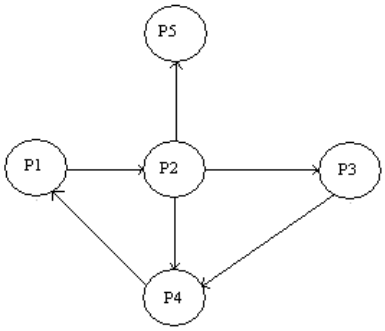
Deadlock Detection

(i) Single instance of each resource type

If all resources have only a single instance, then we can define a deadlock detection algorithm that use a variant of resource-allocation graph called a wait for graph.

Resource Allocation Graph **Wait for Graph**





(ii) Several Instance of a resource type

Available : Number of available resources of each type

Allocation : number of resources of each type currently allocated to each process

Request : Current request of each process

If Request [i,j]=k, then process P_i is requesting K more instances of resource typeR_j.

1. Initialize work := available
   Finish[i]=false, otherwise finish [i]:=true
2. Find an index i such that both
   a. Finish[i]=false
   b. Request$_i$<=work
   if no such i exists go to step4.
3. Work:=work+allocation$_i$
   Finish[i]:=true goto step2
4. If finish[i]=false then process P_i is deadlocked

DEADLOCK RECOVERY.

- There are three basic approaches to recovery from deadlock:
  1. Inform the system operator, and allow him/her to take manual intervention.
  2. Terminate one or more processes involved in the deadlock
  3. Preempt resources.

1. Process Termination

Two basic approaches, both of which recover resources allocated to terminated processes:

➔Terminate all processes involved in the deadlock. This definitely solves the

deadlock, but at the expense of terminating more processes than would be absolutely necessary.

➔Terminate processes one by one until the deadlock is broken. This is more conservative, but requires doing deadlock detection after each step.

➔In the latter case there are many factors that can go into deciding which processes to terminate next:

- Process priorities.
- How long the process has been running, and how close it is to finishing.
- How many and what type of resources is the process holding. (Are they easy to preempt and restore?)
    1. How many more resources does the process need to complete.
    2. How many processes will need to be terminated
    3. Whether the process is interactive or batch.
    4. (Whether or not the process has made non-restorable changes to any resource.)

2. Resource Preemption

➔When preempting resources to relieve deadlock, there are three important issues to be addressed:

1. Selecting a victim - Deciding which resources to preempt from which processes involves many of the same decision criteria outlined above.
2. Rollback - Ideally one would like to roll back a preempted process to a safe state prior to the point at which that resource was originally allocated to the process. Unfortunately it can be difficult or impossible to determine what such a safe state is, and so the only safe rollback is to roll back all the way back to the beginning. (I.e. abort the process and make it start over.)
3. Starvation - How do you guarantee that a process won't starve because its resources are constantly being preempted? One option would be to use a priority system, and increase the priority of a process every time its resources get preempted. Eventually it should get a high enough priority that it won't get preempted any more.

WINDOWS 7 – THREAD AND SMP MANAGEMENT

The native process structures and services provided by the Windows Kernel are relatively simple and general purpose, allowing each OS subsystem to emulate a particular process structure and functionality.

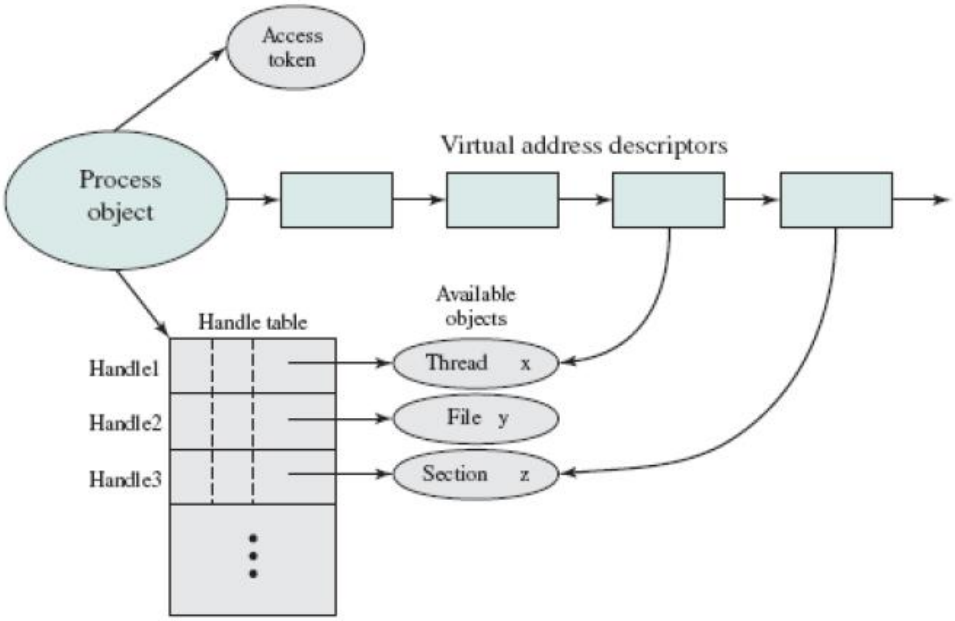
Characteristics of Windows processes:

- Windows processes are implemented as objects.
- A process can be created as new process, or as a copy of an existing process.
- An executable process may contain one or more threads.
- Both process and thread objects have built-in synchronization capabilities.

A Windows Process and Its Resources

- Each process is assigned a security access token, called the primary token of the process. When a user first logs on, Windows creates an access token that includes the security ID for the user.

- Every process that is created by or runs on behalf of this user has a copy of this access token.
- Windows uses the token to validate the user's ability to access secured objects or to perform restricted functions on the system and on secured objects. The access token controls whether the process can change its own attributes.



- Related to the process is a series of blocks that define the virtual address space currently assigned to this process.

- The process cannot directly modify these structures but must rely on the virtual memory manager, which provides a memory allocation service for the process.

- The process includes an object table, with handles to other objects known to this process. The process has access to a file object and to a section object that defines a section of shared memory.

Process and Thread Objects

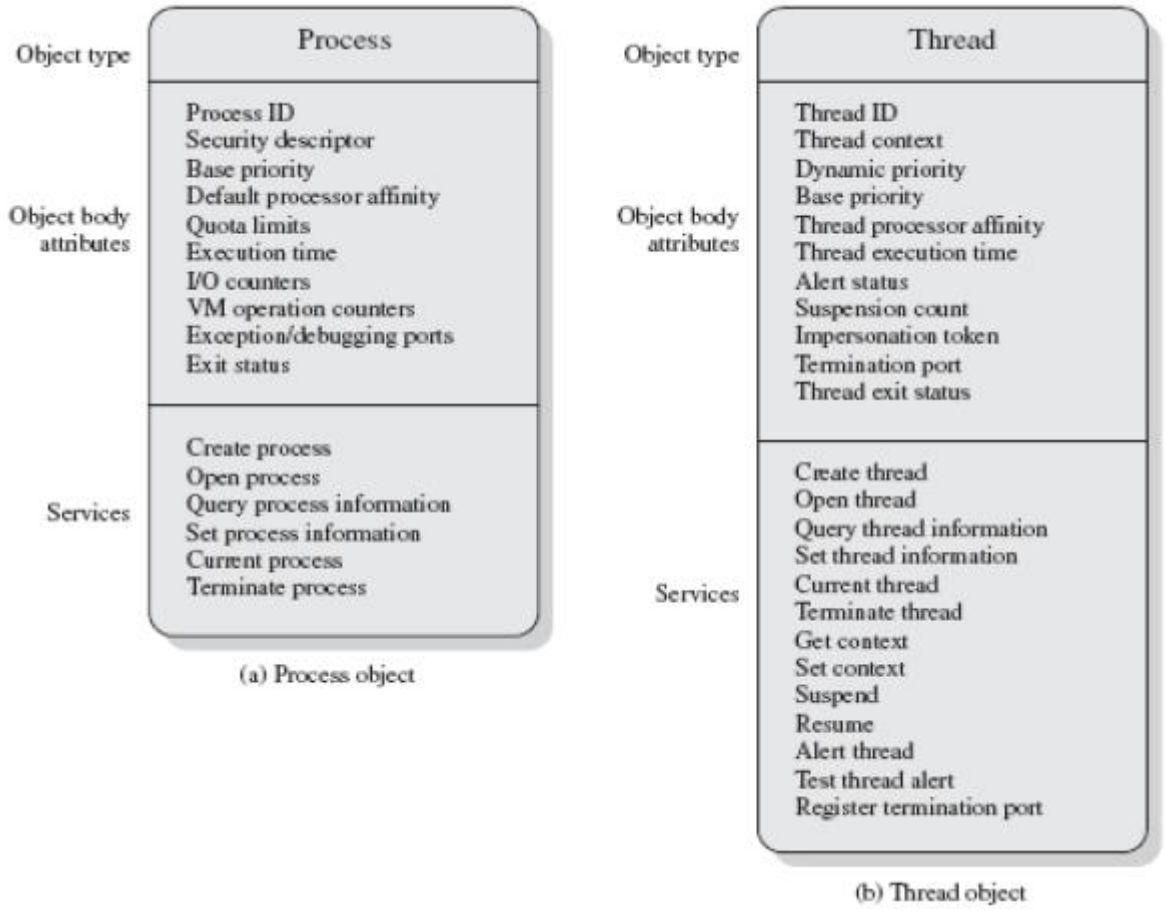
 The object-oriented structure of Windows facilitates the development of a general-purpose process facility.

 Windows makes use of two types of process-related objects: processes and threads.

 A process is an entity corresponding to a user job or application that owns resources, such as memory and open files.

 A thread is a dispatchable unit of work that executes sequentially and is interruptible, so that the processor can turn to another thread.

Windows Process and Thread Objects



(a) Process object

(b) Thread object

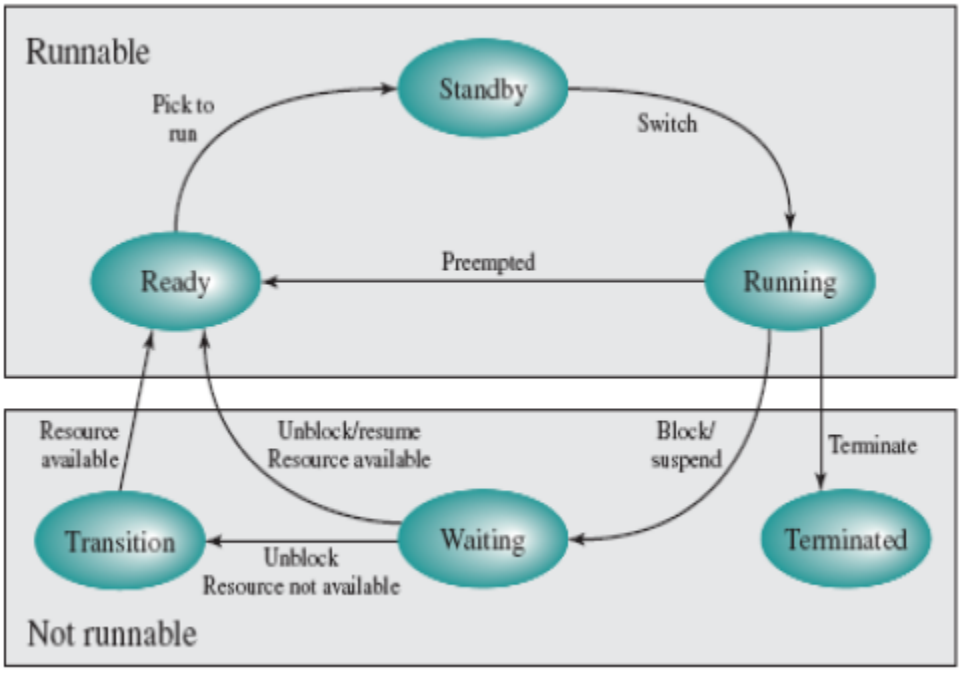
Windows Process Object Attributes

Process ID	A unique value that identifies the process to the operating system.
Security descriptor	Describes who created an object, who can gain access to or use the object, and who is denied access to the object.
Base priority	A baseline execution priority for the process's threads.
Default processor affinity	The default set of processors on which the process's threads can run.
Quota limits	The maximum amount of paged and nonpaged system memory, paging file space, and processor time a user's processes can use.
Execution time	The total amount of time all threads in the process have executed.
I/O counters	Variables that record the number and type of I/O operations that the process's threads have performed.
VM operation counters	Variables that record the number and types of virtual memory operations that the process's threads have performed.
Exception/debugging ports	Interprocess communication channels to which the process manager sends a message when one of the process's threads causes an exception. Normally, these are connected to environment subsystem and debugger processes, respectively.
Exit status	The reason for a process's termination.

Windows Thread Object Attributes

Thread ID	A unique value that identifies a thread when it calls a server.
Thread context	The set of register values and other volatile data that defines the execution state of a thread.
Dynamic priority	The thread's execution priority at any given moment.
Base priority	The lower limit of the thread's dynamic priority.
Thread processor affinity	The set of processors on which the thread can run, which is a subset or all of the processor affinity of the thread's process.
Thread execution time	The cumulative amount of time a thread has executed in user mode and in kernel mode.
Alert status	A flag that indicates whether a waiting thread may execute an asynchronous procedure call.
Suspension count	The number of times the thread's execution has been suspended without being resumed.
Impersonation token	A temporary access token allowing a thread to perform operations on behalf of another process (used by subsystems).
Termination port	An interprocess communication channel to which the process manager sends a message when the thread terminates (used by subsystems).
Thread exit status	The reason for a thread's termination.

Thread States



Problem

1.Consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time	Arrival Time	Priority
P1	23	0	2
P2	3	1	1
P3	6	2	4
P4	2	3	3

a. Draw four *Gantt charts* illustrating the execution of these processes using FCFS, SJF(Preemptive), a non-preemptive priority (a smaller priority number implies a higher priority), and RR (quantum = 1) scheduling.

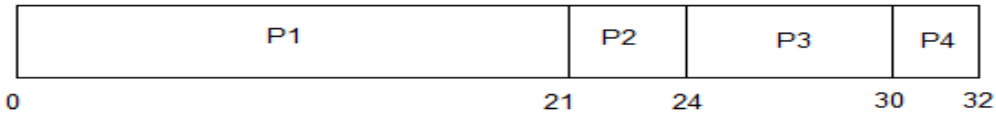
c. What is the *waiting time* of each process for each of the scheduling algorithms in part a?

d. Which of the schedules in part a results in the *minimal average waiting time* (over all processes)?

FCFS SCHEDULING

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The average waiting time will be = (0 + 21 + 24 + 30)/4 = 18.75 ms



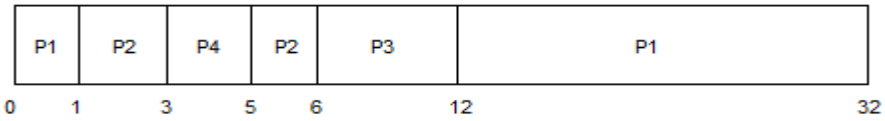
This is the GANTT chart for the above processes

SJF(SHORTEST JOB FIRST)

In Pre-emptive Shortest Job First Scheduling, jobs are put into ready queue as they arrive, but as a process with short burst time arrives, the existing process is pre-empted.

PROCESS	BURST TIME	ARRIVAL TIME
P1	21	0
P2	3	1
P3	6	2
P4	2	3

The GANTT chart for Preemptive Shortest Job First Scheduling will be,



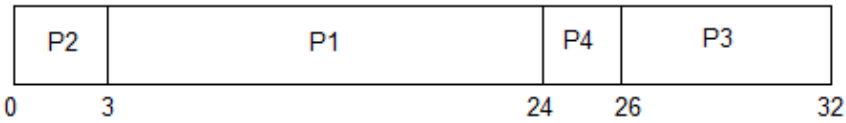
The average waiting time will be, ((5-3) + (6-2) + (12-1))/4 = 4.25 ms

The average waiting time for preemptive shortest job first scheduling is less than both, non-preemptive SJF scheduling and FCFS scheduling.

PRIORITY

PROCESS	BURST TIME	PRIORITY
P1	21	2
P2	3	1
P3	6	4
P4	2	3

The GANTT chart for following processes based on Priority scheduling will be,

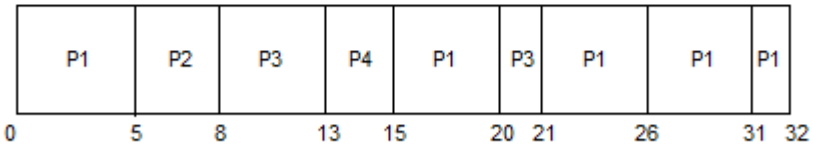


The average waiting time will be, (0 + 3 + 24 + 26)/4 = 13.25 ms

ROUND ROBIN

PROCESS	BURST TIME
P1	21
P2	3
P3	6
P4	2

The GANTT chart for round robin scheduling will be,

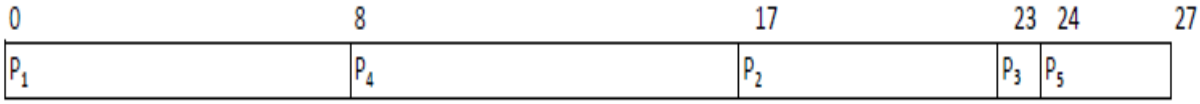


The average waiting time will be, 11 ms.

2.

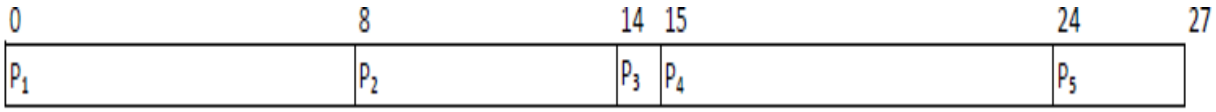
Process	Burst	Priority	Arrival Time
P_1	8	4	0
P_2	6	1	2
P_3	1	2	2
P_4	9	2	1
P_5	3	3	3

First Come First Served



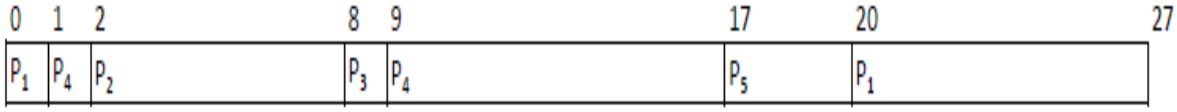
Avg. Wait = 0+8-1+17-2+23-2+24-3 = 0+7+15+21+21=64/5 = 12.8 AVG TAT = 8+17-1+23-2+24-2+27-3 = 8+16+21+22+24=91/5=18.2

SJF



Avg. Wait = 8-2+14-2+15-1+24-3 = 6+12+14+21 = 53/5=10.6ms AVG TAT = 8+14-2+15-2+24-1+27-3 = 8+12+13+23+24=80/5=16ms

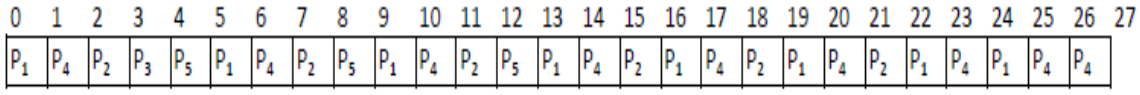
Priority



Avg. Wait Time = 0+20-1+2-2+8-2+9-2+17-3 = 0+19+0+6+7+14 = 46/5=9.2ms AVG TAT = 27+8-2+9-2+16+20-3 = 73/5 = 14.6ms

Round Robin

Round Robin (1ms Quantum)



Wait Time P_1 = 0+4+3+3+2+2+1+1 = 16

Wait Time P2 = 0+4+3+3+2+2+2+1 = 17

Wait Time P3 = 1

Wait Time P4 = 4+4+3+2+3+2+1 = 19

Wait Time P5 = 1+3+3 = 7

Avg Wait Time = 60/5 = 12ms

Avg TAT =25+21+2+26+10 = 84/5 = 16.8

UNIT III STORAGE MANAGEMENT

Main Memory-Contiguous Memory Allocation, Paging, Segmentation, Segmentation with paging, 32 and 64 bit architecture Examples; Virtual Memory- Background, Demand Paging, Page Replacement, Allocation, Thrashing; Allocating Kernel Memory, OS Examples.

1. MEMORY MANAGEMENT: BACKGROUND

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution.

Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes.

It decides which process will get memory at what time.

It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

1.1 Basic Hardware

Program must be brought (from disk) into memory and placed within a process for it to be run

- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a **stall**
- Cache sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

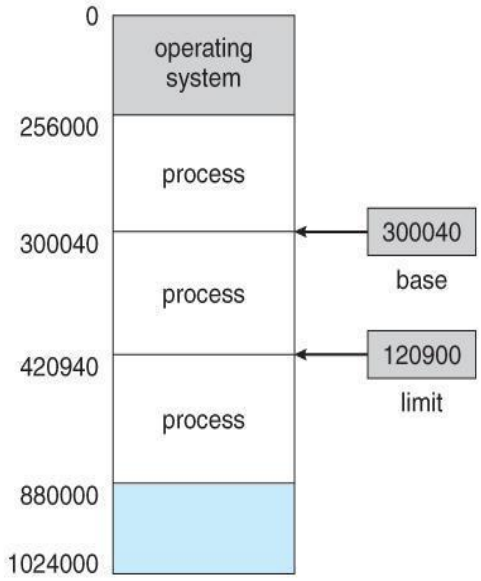
➔ We can provide this protection by using two registers, usually a **base** and a **limit**

The base register holds the smallest legal physical memory

address; The limit register specifies the size of the range.

For example, if the base register holds 300040 and limit register is 120900, then the

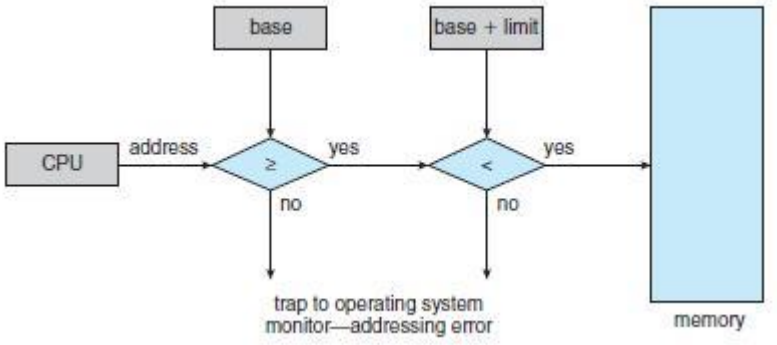
program can legally access all addresses from 300040 through 420940 (inclusive).



Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with the registers.

Any attempt by a program executing in user mode to access operating-system memory or other users' memory results in a trap to the operating system, which treats the attempt as a fatal error.

This scheme prevents a user program from (accidentally or deliberately) modifying the code or data structures of either the operating system or other users.



1.2 Address Binding

Definition

Converting the address used in a program to an actual physical address.

Address binding is the process of mapping the program's logical or virtual addresses to corresponding physical or main memory addresses.

In other words, a given logical address is mapped by the MMU (Memory Management Unit) to a physical address.

User programs typically refer to memory addresses with symbolic names such as "i", "count", and "average Temperature".

These symbolic names must be mapped or bound to physical memory addresses, which typically occurs in several stages:

Three different stages of binding:

1. **Compile time**. The compiler translates symbolic addresses to absolute addresses. If you know at compile time where the process will reside in memory, then absolute code can be generated (Static).

2. **Load time**. The compiler translates symbolic addresses to relative (relocatable) addresses. The loader translates these to absolute addresses. If it is not known at compile time where the process will reside in memory, then the compiler must generate relocatable code (Static).

3. **Execution time**. If the process can be moved during its execution from one memory segment to another, then binding must be delayed until run time. The absolute addresses are generated by hardware. Most general-purpose OS use this method (Dynamic).

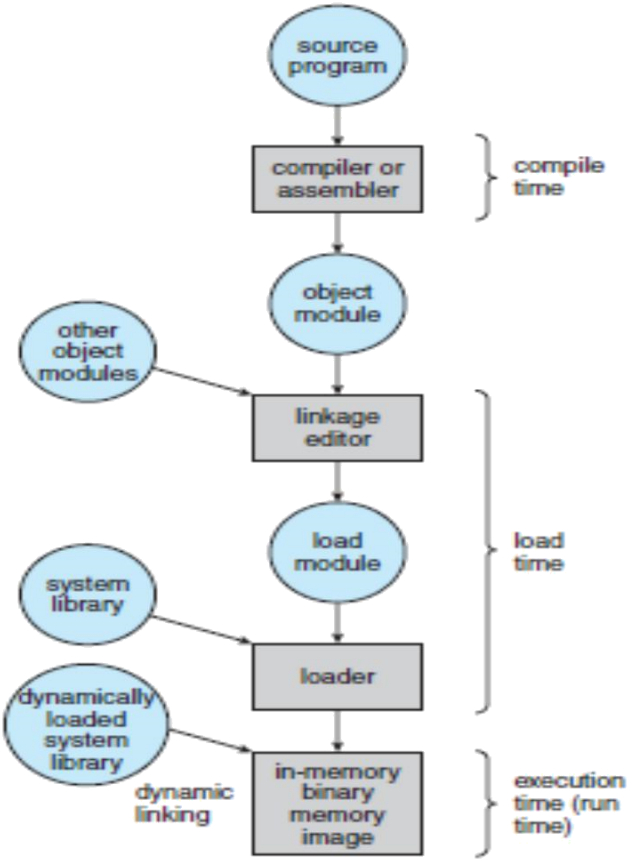


Figure 8.3 Multistep processing of a user program.

1.3 Logical vs. Physical Address Space

Logical address – generated by the CPU; also referred to as **"virtual address"**

Physical address – address seen by the memory unit.

Logical and physical addresses are the **same** in compile-time and load-time address-binding schemes

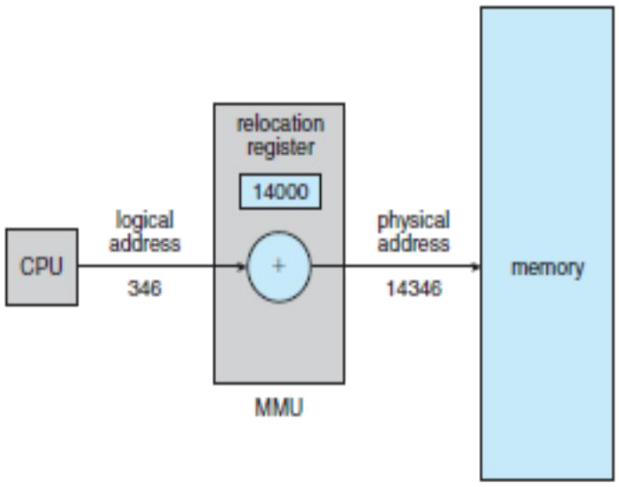
Logical (virtual) and physical addresses **differ** in execution-time address- binding scheme

Memory-Management Unit (MMU)

It is a hardware device that maps virtual / Logical address to physical address.
In this scheme, the relocation register's value is added to Logical address generated by a user process.

The Base register is called a relocation register.

- The value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- For example, if the base is at 14000, then an attempt by the user to address location 0 is dynamically relocated to location 14000; an access to location 346 is mapped to location 14346.
- The user program never sees the real physical addresses. The program can create a pointer to location 346, store it in memory, manipulate it, and compare it with other addresses -all as the number 346.
- The user program deals with logical addresses.



1.4 Dynamic Loading

Dynamic loading is a mechanism by which a computer program can, at run time, load a library (or other binary) into memory, retrieve the addresses of functions and variables contained in the library, execute those functions or access those variables, and unload the library from memory.

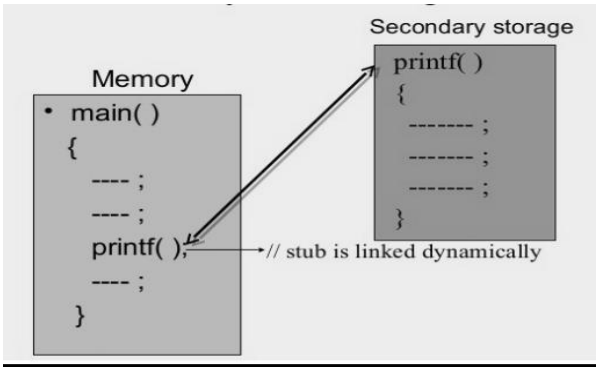
Dynamic loading means loading the library (or any other binary for that matter) into the memory during load or run-time.

Dynamic loading can be imagined to be similar to plugins, that is an exe can actually execute before the dynamic loading happens(The dynamic loading for example can be created using Load Library call in C or C++)

1.5 Dynamic Linking and shared libraries

Dynamic linking refers to the linking that is done during load or run-time and not when the exe is created.
In case of dynamic linking the linker while creating the exe does minimal work. For the dynamic linker to work it actually has to load the libraries too. Hence it's also called linking loader.
Small piece of code, *stub*, used to indicate how to load library routine.
Stub replaces itself with the address of the routine, and executes the routine.
Operating system needed to check if routine is in processes memory address.
Dynamic linking is particularly useful for libraries.

- Shared libraries: Programs linked before the new library was installed will continue using the older library.



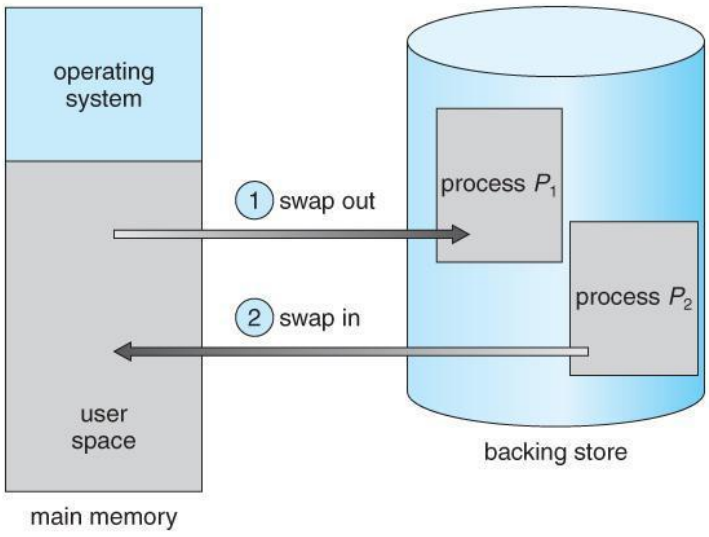
2. SWAPPING

2.1 Basic

- A process can be swapped temporarily out of memory to a backing store (SWAP OUT)and then brought back into memory for continued execution (SWAP IN).
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users & it must provide direct access to these memory images
- **Roll out, roll in –** swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- **Transfer time:** Major part of swap time is transfer time. Total transfer time is directly proportional to the amount of memory swapped.

□ **Example**: Let us assume the user process is of size 1MB & the backing store is a standard hard disk with a transfer rate of 5MBPS.

Transfer time = 1000KB/5000KB per second
= 1/5 sec = 200ms



A process with dynamic memory requirements will need to issue system calls (request memory() and release memory()) to inform the operating system o f its changing memory needs.

2.2 Swapping on Mobile Systems

Swapping is typically not supported on mobile platforms, for several reasons:

Mobile devices typically use flash memory in place of more spacious hard drives for persistent storage, so there is not as much space available.

Flash memory can only be written to a limited number of times before it becomes unreliable.

The bandwidth to flash memory is also lower.

Apple's IOS asks applications to voluntarily free up memory

Read-only data, e.g. code, is simply removed, and reloaded later if needed.

Modified data, e.g. the stack, is never removed.

Apps that fail to free up sufficient memory can be removed by the

OS Android follows a similar strategy.

Prior to terminating a process, Android writes its application state to flash memory for quick restarting.

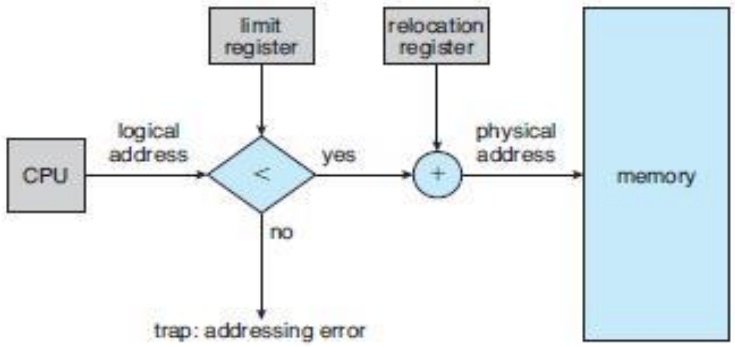
3. CONTIGUOUS MEMORY ALLOCATION

One approach to memory management is to load each process into a contiguous space.

The operating system is allocated space first, usually at either low or high memory locations, and then the remaining available memory is allocated to processes as needed.

3.1 Memory Protection

Protection against user programs accessing areas that they should not, allows programs to be relocated to different memory starting addresses as needed, and allows the memory space devoted to the OS to grow or shrink dynamically as needs change.



3.2 Memory Allocation

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process.

When a partition is free, a process is selected from the input queue and loaded into it.

There are two methods namely:

- Fixed – P**a**rtition Method
- Variable – Partition Method

- **Fixed – Partition Method:**

Divide memory into fixed size partitions, where each partition has exactly one process. The drawback is Memory space unused within a partition is wasted.(eg. When process size < partition size)

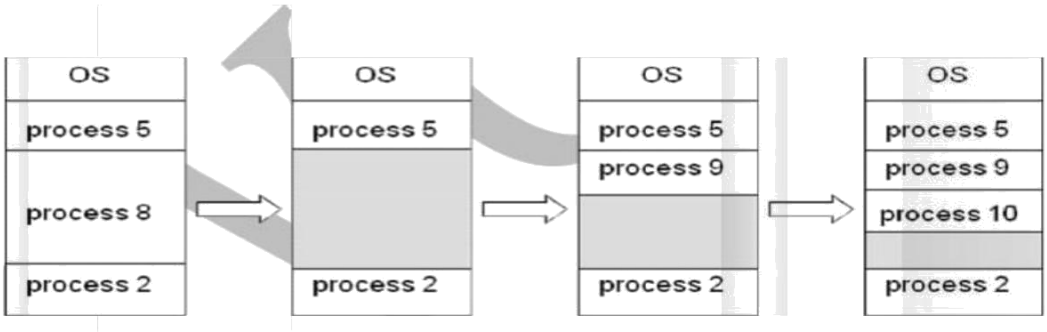
- **Variable-partition method:**

o Divide memory into variable size partitions, depending upon the size of the incoming process.

o When a process terminates, the partition becomes available for another process.

o As processes complete and leave they create holes in the main memory.

o ***Hole*** – block of available memory; holes of various size are scattered throughout memory.



Dynamic Storage- Allocation Problem:

How to satisfy a request of size n‘from a list of free holes?

➔The free blocks of memory are known as holes. The set of holes is searched to determine which hole is best to allocate.

Solution:

- First-fit: Allocate the first hole that is big enough.
- Best-fit: Allocate the smallest hole that is big enough; must search entire list, unless ordered by size. Produces the smallest leftover hole.
- Worst-fit: Allocate the largest hole; must also search entire list. Produces the largest leftover hole.

Example :

Given five memory partitions of 100 KB, 500 KB, 200 KB, 300 KB, and 600 KB (in order), how would each of the first-fit, best-fit, and worst-fit algorithms place processes of 212 KB, 417 KB, 112 KB, and 426 KB (in order)?Which algorithm makes the most efficient use of memory?

a. First-fit:
1. 212K is put in 500K partition
2. 417K is put in 600K partition
3. 112K is put in 288K partition (new partition 288K = 500K − 212K)
4. 426K must wait

b. Best-fit:
1. 212K is put in 300K partition
2. 417K is put in 500K partition
3. 112K is put in 200K partition
4. 426K is put in 600K partition

c. Worst-fit:
1. 212K is put in 600K partition
2. 417K is put in 500K partition
3. 112K is put in 388K partition
4. 426K must wait

In this example, best-fit turns out to be the best.

NOTE: First-fit and best-fit are better than worst-fit in terms of speed and storage utilization

3.3Fragmentation:

Fragmentation is a phenomenon in which storage space is used inefficiently, reducing capacity or performance and often both.

1. **External Fragmentation** – This takes place when enough total memory space exists to satisfy a request, but it is not contiguous i.e, storage is fragmented into a large number of small holes scattered throughout the main memory.

2. Internal Fragmentation – Allocated memory may be slightly larger than requested memory.

Example: hole = 184 bytes Process size = 182 bytes.

We are left with a hole of 2 bytes.

➔ **Solutions**

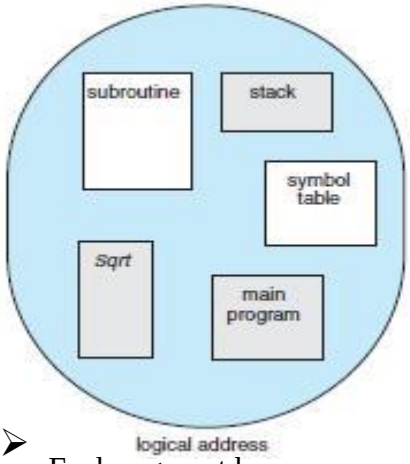
Compaction: Move all processes towards one end of memory, hole towards other end of memory, producing one large hole of available memory. This scheme is expensive as it can be done if relocation is dynamic and done at execution time.

4. SEGMENTATION

4.1 Basic Method

o Memory-management scheme that supports user view of memory

o A program is a collection of segments. A segment is a logical unit such as:Main program, Procedure, Function, Method, Object, Local variables, global variables, Common block, Stack, Symbol table, arrays



- Each segment has a name and a length.
- The addresses specify both the segment name and the offset within the segment.
- The programmer therefore specifies each address by two quantities:

a segment name and an offset.

A logical address consists of a two tuple:

<segment-number, offset>.

4.2Segmentation Hardware

Each entry in the segment table has a segment base and a segment limit.

The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment

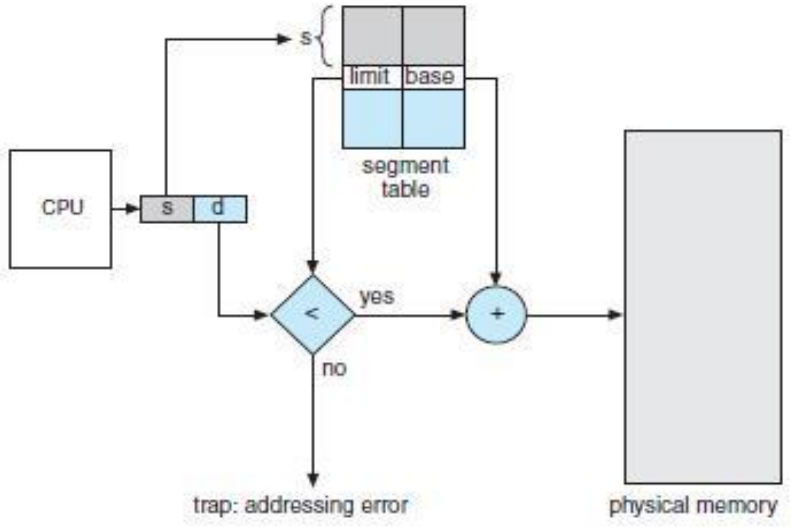
A logical address consists of two parts:

a **segment number** ➔ s, and an **offset** into that segment ➔ d. The **segment number** is used as an index to the segment table.

The **offset** d of the logical address must be between 0 and the segment limit.

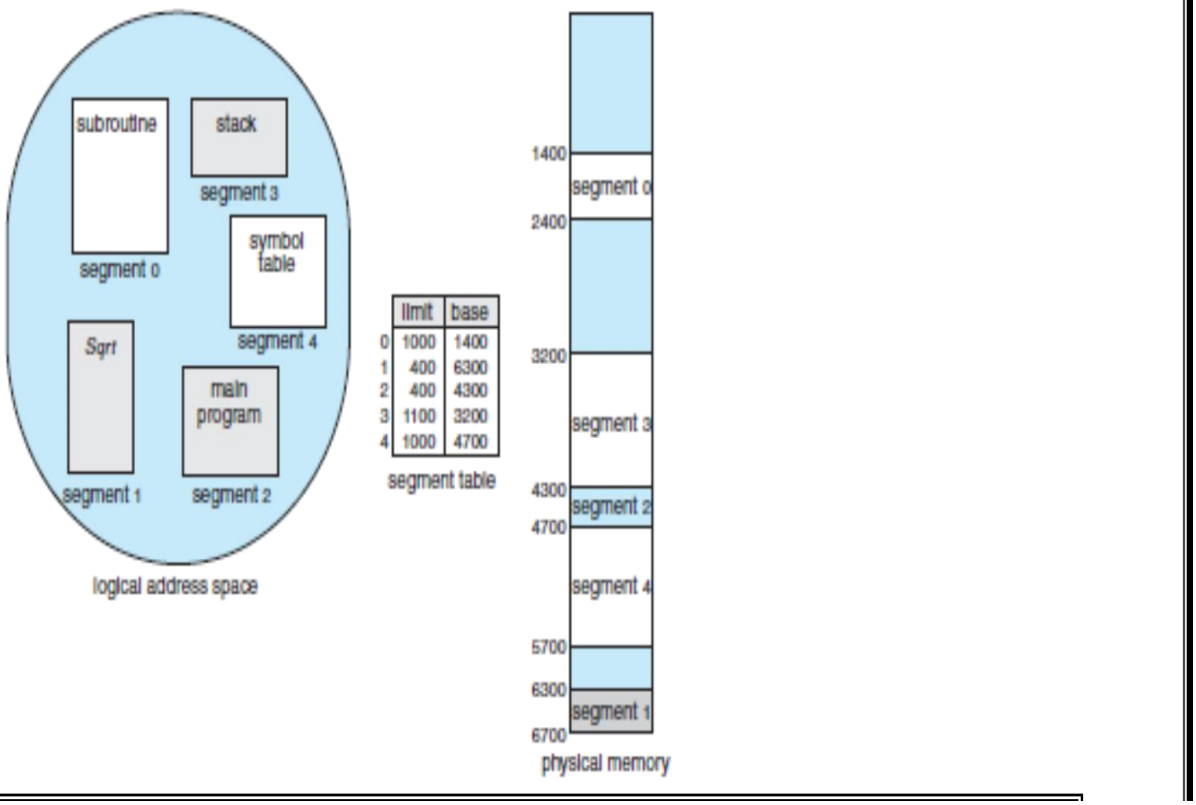
If it is not, we trap to the operating system (logical addressing attempt beyond end of segment).

When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte.



For example,

segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 + 53 = 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.



5. PAGING

- It is a memory management scheme that permits the physical address space of a process to be noncontiguous.
- It avoids the considerable problem of fitting the varying size memory chunks on to the backing store.

5.1Basic Method

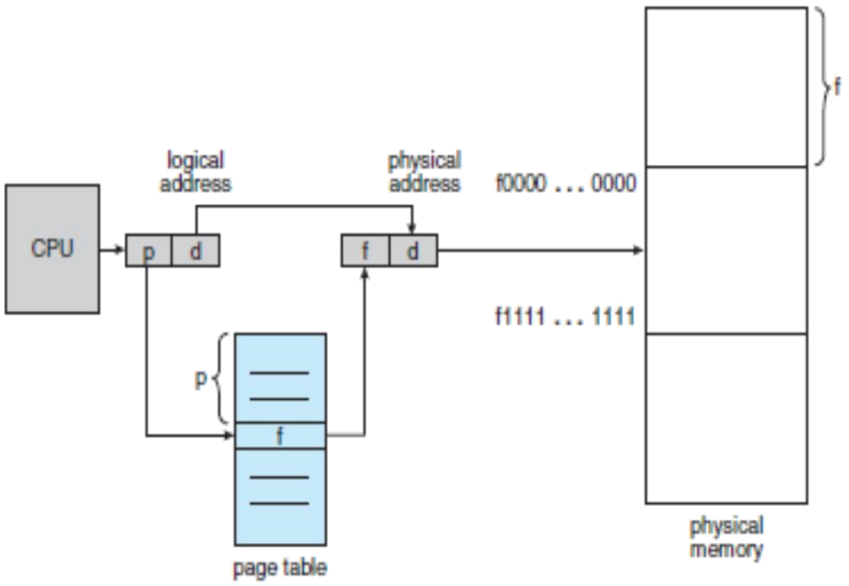
- Divide logical memory into blocks of same size called **"pages"**.
- Divide physical memory into fixed-sized blocks called **"frames"**
- Page size is a power of 2, between 512 bytes and 16MB.

Address Translation Scheme

each page Address generated by CPU (logical address) is divided into:

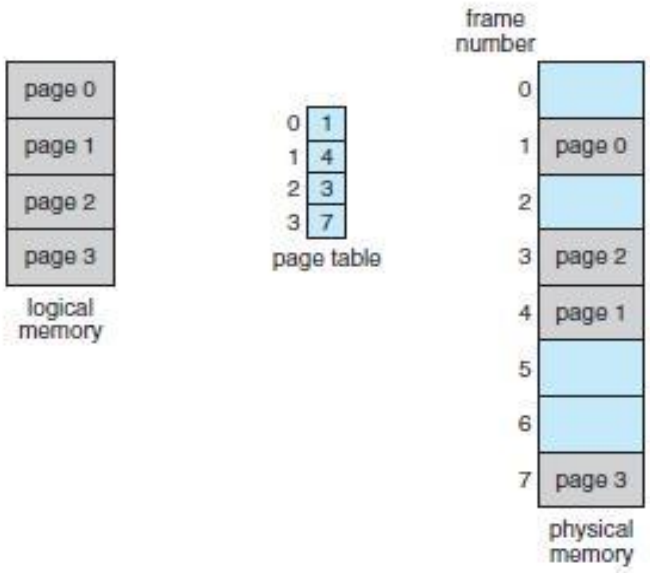
Page number *(p)* – used as an index into a page table which contains base address of in physical memory

Page offset *(d)* – combined with base address to define the physical address
i.e., Physical address = base address + offset



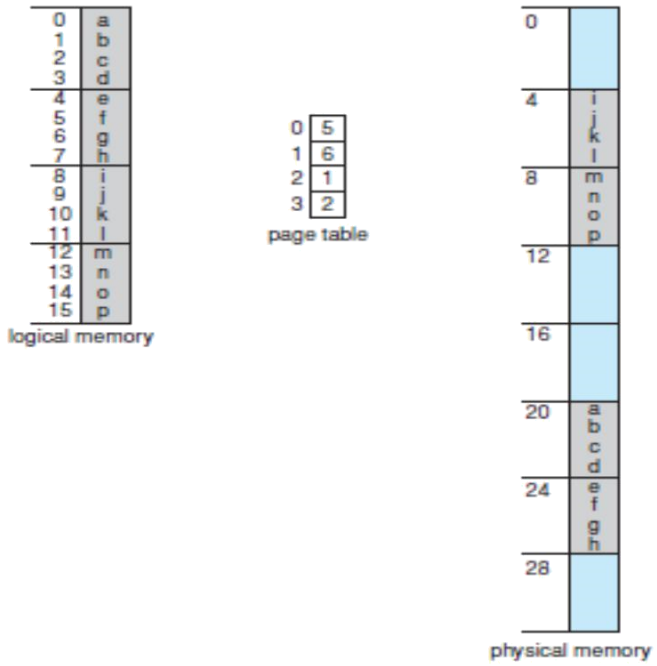
The page number is used as an index into a page table. The page table contains the base address of each page in physical memory.

This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.



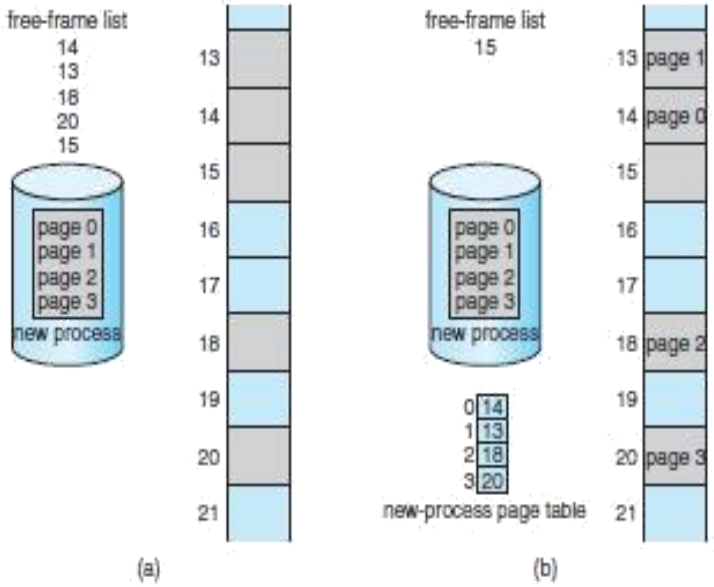
Consider the memory in the logical address, n= 2 and m = 4. Using a page size of 4 bytes and a physical memory of 32 bytes (8 pages), we show how the programmer's view of memory can be mapped into physical memory. Logical address 0 is page 0, offset 0. Indexing into the page table, we find that page 0 is in frame 5.

Thus, logical address 0 maps to physical address 20 [= (5 × 4) + 0]. Logical address 3 (page 0, offset 3) maps to physical address 23 [= (5 × 4) + 3]. Logical address 4 is page 1, offset 0; according to the page table, page 1 is mapped to frame 6. Thus, logical address 4 maps to physical address 24 [= (6 × 4) + 0]. Logical address 13 maps to physical address 9.



Since the operating system is managing physical memory, it must be aware of the allocation details of physical memory, which frames are allocated, which frames are available, how many total frames there are, and so on.

This information is generally kept in a data structure called a frame table.
The frame table has one entry for each physical page frame, indicating whether the latter is free or allocated and, if it is allocated, to which page of which process or processes.



5.2 Hardware Support

- The TLB is associative, high-speed memory.
- Each entry in the TLB consists of two parts:
  - a key (or tag) and a value.
- When the associative memory is presented with an item, the item is compared with all keys simultaneously.
- If the item is found, the corresponding value field is returned.
- The TLB contains only a few of the page-table entries.
- When a logical address is generated by the CPU, its page number is presented to the TLB.
- If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made.
- Depending on the CPU, this may be done automatically in hardware or via an interrupt to the operating system.
- If the page number is found, its frame number is immediately available and is used to access Memory.

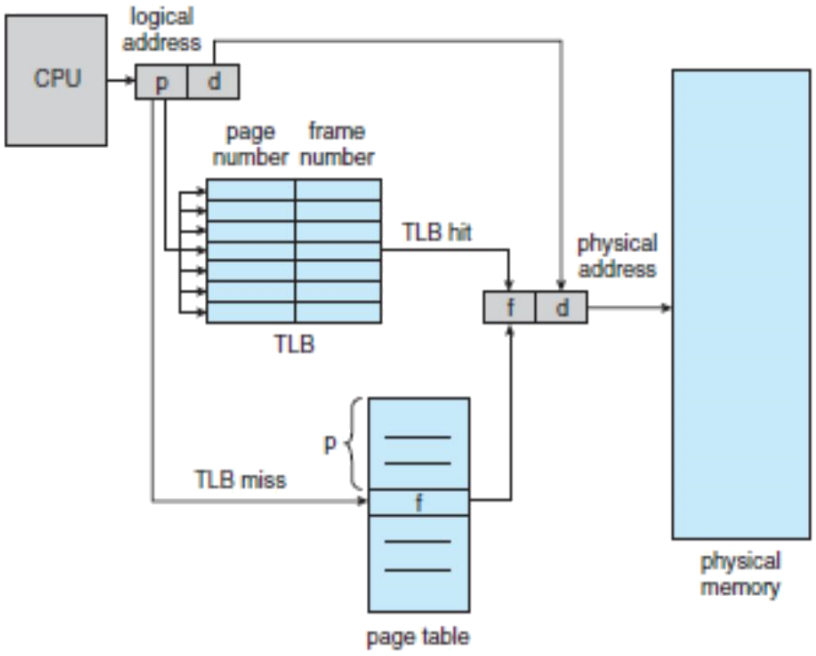
  Hit Ratio - The percentage of times that the page number of interest is found in the TLB is called the hit ratio.

- An 80-percent hit ratio, for example, means that we find the desired page number in the TLB 80 percent of the time. If it takes 100 nanoseconds to access memory, then a mapped-memory access takes 100 nanoseconds when the page number is in the TLB.

- If we fail to find the page number in the TLB then we must first access memory for the page table and frame number (100 nanoseconds) and then access the desired byte in memory (100 nanoseconds), for a total of 200 nanoseconds.

  effective access time = $0.80 \times 100 + 0.20 \times 200$
  = 120 nanoseconds

  For a 99-percent hit ratio, which is much more realistic, we have effective access time = $0.99 \times 100 + 0.01 \times 200 = 101$ nanoseconds

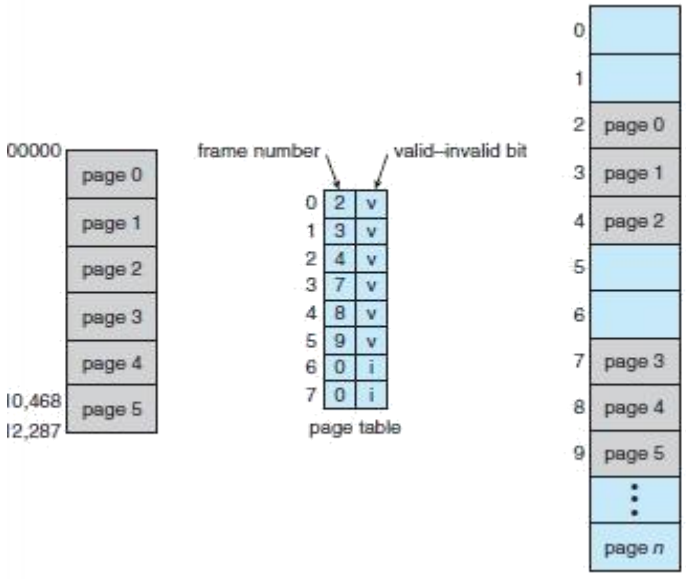


5.3 Protection

Memory protection in a paged environment is accomplished by protection bits associated with each frame.
One additional bit is generally attached to each entry in the page table: a valid–invalid bit.

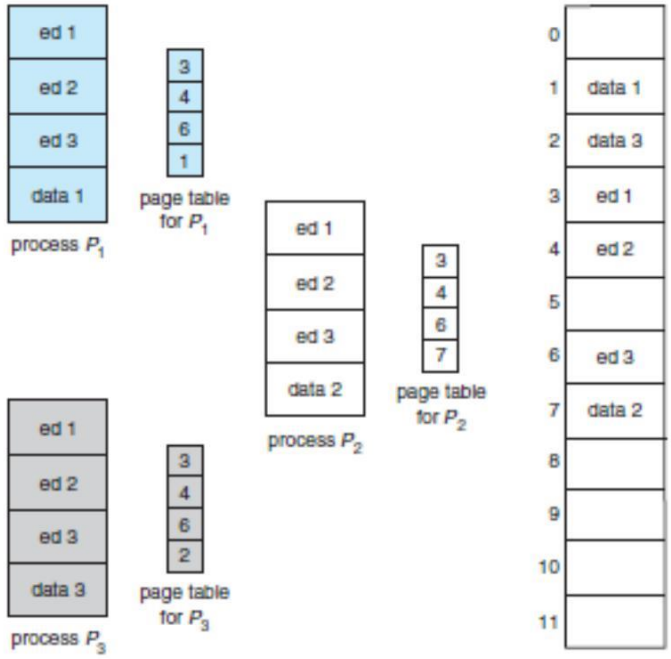
When this bit is set to valid, the associated page is in the process's logical address space and is thus a legal (or valid) page.

When the bit is set to invalid, the page is not in the process's logical address space. Illegal addresses are trapped by use of the valid–invalid bit.



5.4 Shared Pages

An advantage of paging is the possibility of sharing common code.



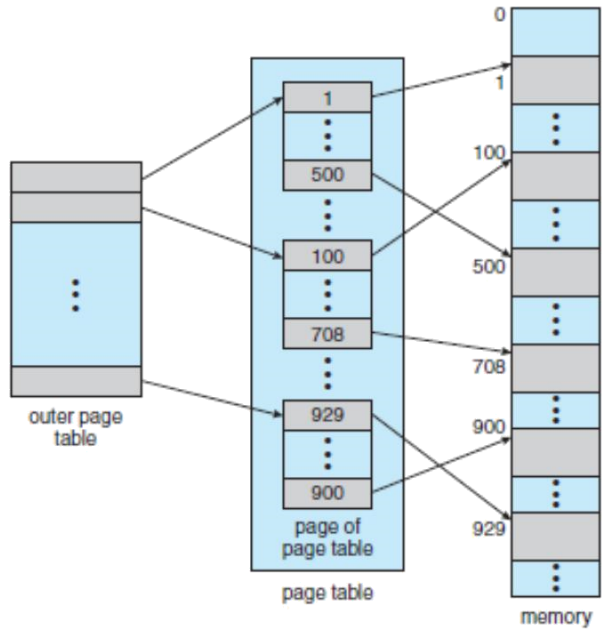
6.STRUCTURE OF PAGE TABLE

The most common techniques for structuring the page table, including hierarchical paging, hashed page tables, and inverted page tables.

1.Hierarchical Paging

The page table itself becomes large for computers with large logical address space (232 to 264).
Example:

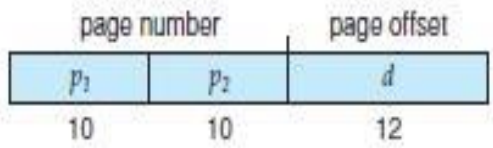
- Consider a system with a 32-bit logical address space. If the page size in such a system is 4 KB(212), then a page table may consist of up to 1 million entries (232/212).
- Assuming that each entry consists of 4 bytes, each process may need up to 4 MB of physical address space for the page table alone.
- The page table should be allocated contiguously in main memory.
- The solution to this problem is to divide the page table into smaller pieces.
- One way of dividing the page table is to use a two-level paging algorithm,



For example, consider again the system with a 32-bit logical address space and a page size of 4 KB. A logical address is divided into a page number consisting of 20 bits and a page offset consisting of 12
bits.

Because we page the page table, the page number is further divided into a 10-bit page number and a 10-bit page offset.

Thus, a logical address is as follows:

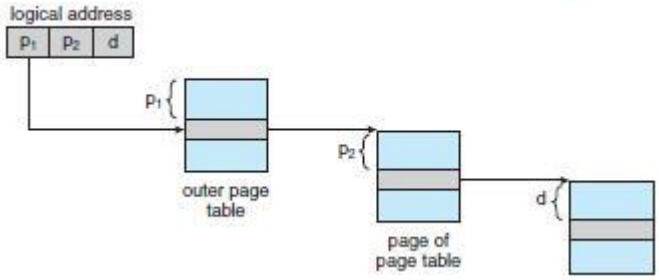


where

p1 - an index into the outer page table

p2 - the displacement within the page of the inner page table.

The address-translation method for this architecture is shown in the figure. Because address translation
works from the outer page table inward, this scheme is also known as a forward-mapped page table.



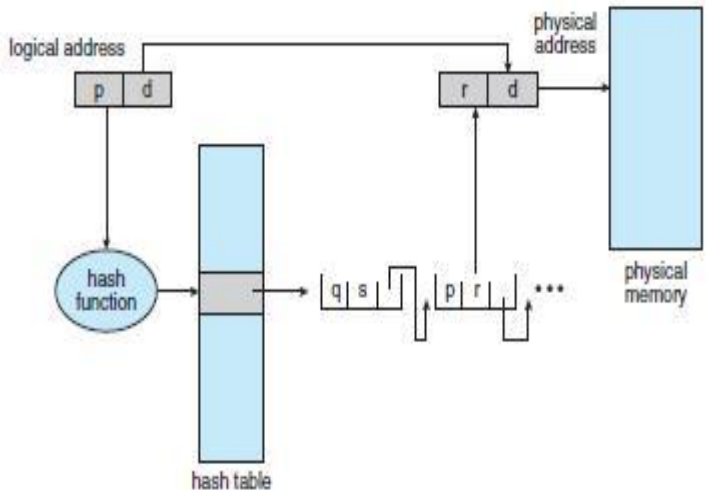
2.Hashed Page Tables

- A common approach for handling address spaces larger than 32 bits is to use a hashed page table, with the hash value being the virtual page number.
- Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions).

- Each element consists of three fields:
  - The virtual page number
  - The value of the mapped page frame
  - A pointer to the next element in the linked list.

Algorithm:

- The virtual page number in the virtual address is hashed into the hash table.
- The virtual page number is compared with field 1 in the first element in the linked list.
- If there is a match, the corresponding page frame (field 2) is used to form the desired physical address.
- If there is no match, subsequent entries in the linked list are searched for a matching virtual page number.



3. Inverted Page Table

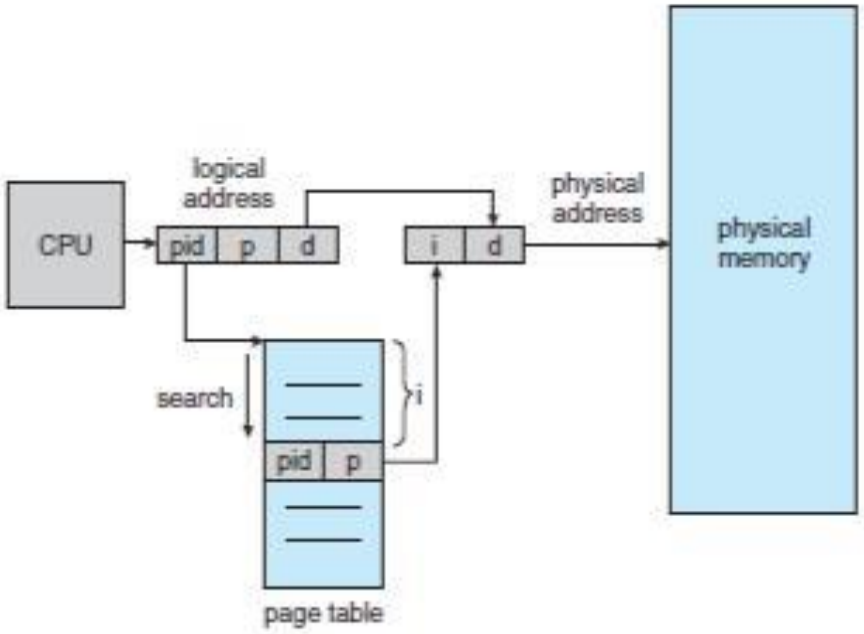
With each process having its own page table, and with each page table consuming considerable amount of memory

We use a lot of memory to keep track of memory.

Inverted page table has one entry for each real page of memory.

Lookup time is increased because it requires a search on the inverted table.

Hash table can be used to reduce this problem.

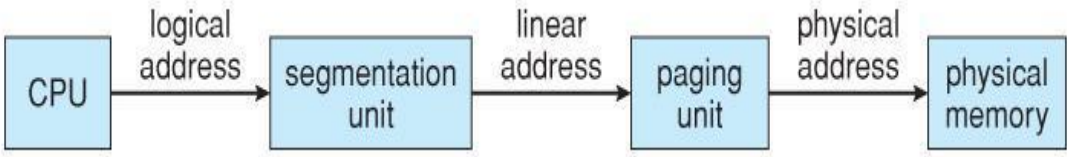


Each virtual address in the system consists of a triple:
<process-id, page-number, offset>.

7.INTEL 32 AND 64-BIT ARCHITECTURES

IA-32 Segmentation

The Pentium CPU provides both pure segmentation and segmentation with paging. In the latter case, the CPU generates a logical address (segment-offset pair), which the segmentation unit converts into a logical linear address, which in turn is mapped to a physical frame by the paging unit



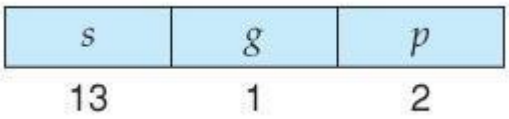
IA-32 Segmentation

The Pentium architecture allows segments to be as large as 4 GB, (24 bits of offset).
Processes can have as many as 16K segments, divided into two 8K groups:

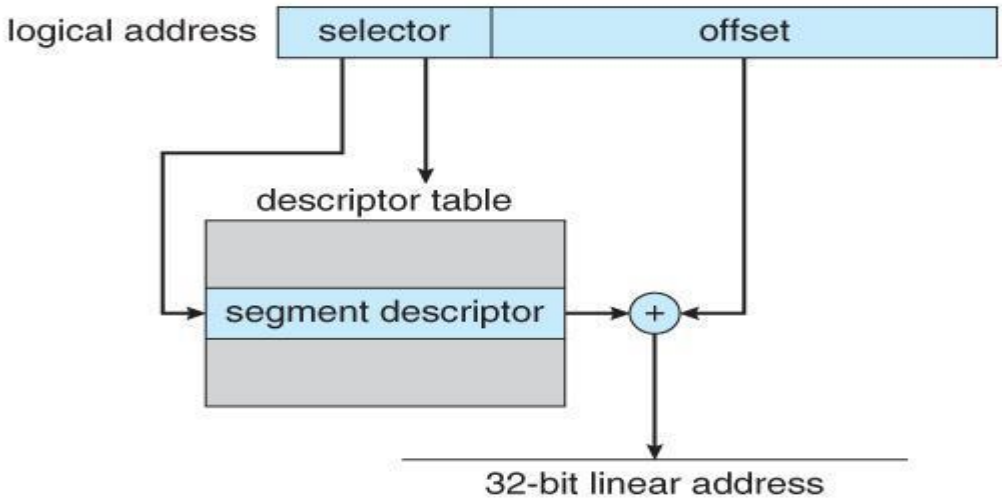
- 8K private to that particular process, stored in the Local Descriptor Table, LDT.
- 8K shared among all processes, stored in the Global Descriptor Table, GDT.

Logical addresses are (selector, offset) pairs, where the selector is made up of 16 bits:

- A 13 bit segment number (up to 8K)
- A 1 bit flag for LDT vs. GDT.
- 2 bits for protection codes.

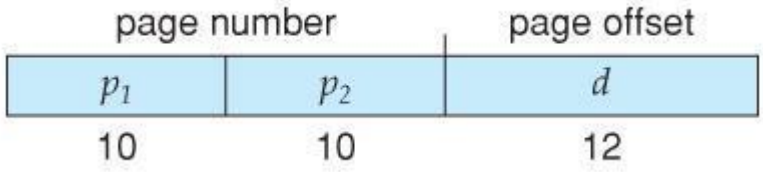


The descriptor tables contain 8-byte descriptions of each segment, including base and limit registers. Logical linear addresses are generated by looking the selector up in the descriptor table and adding the appropriate base address to the offset.



IA-32 Paging

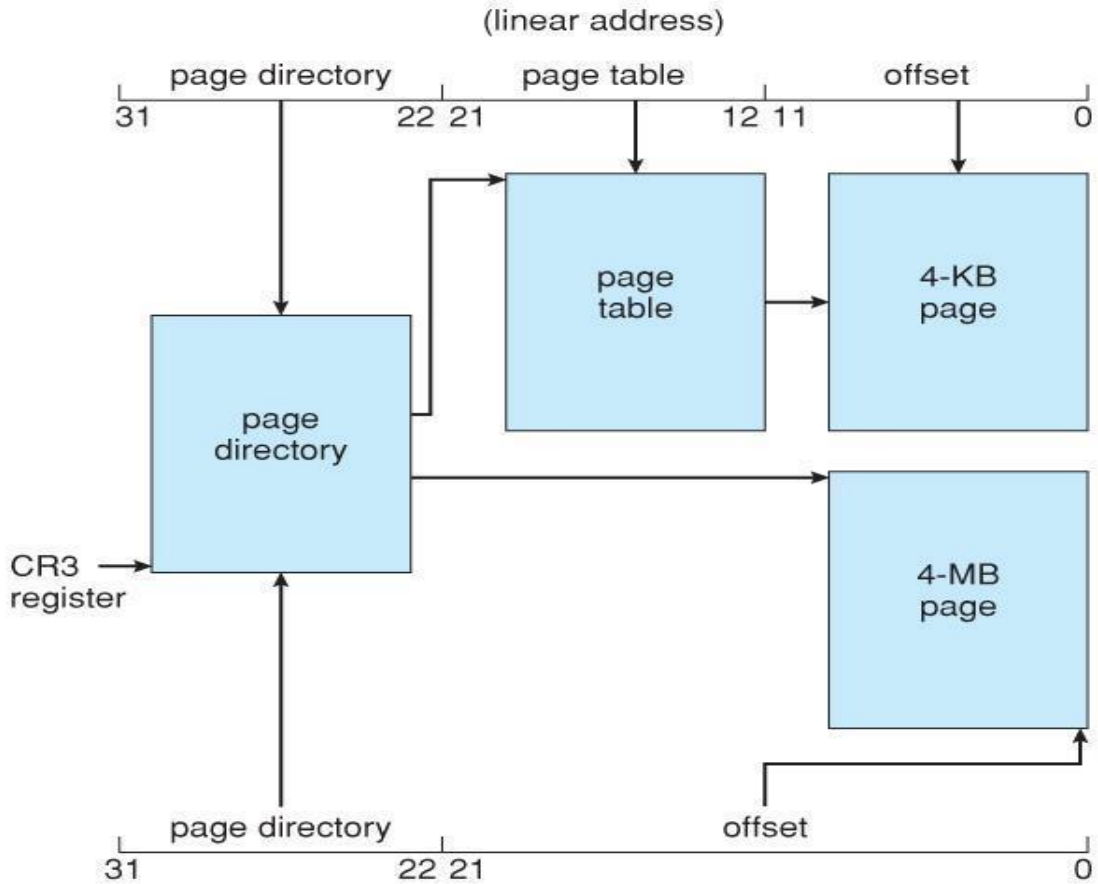
Pentium paging normally uses a two-tier paging scheme, with the first 10 bits being a page number for an outer page table (a.k.a. page directory), and the next 10 bits being a page number within one of the 1024 inner page tables, leaving the remaining 12 bits as an offset into a 4K page.



A special bit in the page directory can indicate that this page is a 4MB page, in which case the remaining 22 bits are all used as offset and the inner tier of page tables is not used.

The CR3 register points to the page directory for the current process.

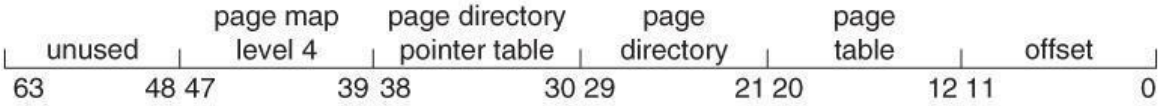
If the inner page table is currently swapped out to disk, then the page directory will have an "invalid bit" set, and the remaining 31 bits provide information on where to find the swapped out page table on the disk.



x86-64

The initial entry of Intel developing 64-bit architectures was the IA-64 (later named Itanium) architecture, but was not widely adopted.

- Meanwhile, AMD —began developing a 64-bit architecture known as x86-64 that was based on extending the existing IA-32 instruction set.
- The x86-64 supported much larger logical and physical address spaces, as well as several other architectural advances.
- Support for a 64-bit address space yields an astonishing 264 bytes of addressable memory— a number greater than 16 quintillion (or 16 exabytes).



8.VIRTUAL MEMORY

- It is a technique that allows the execution of processes that may not be completely in main memory.

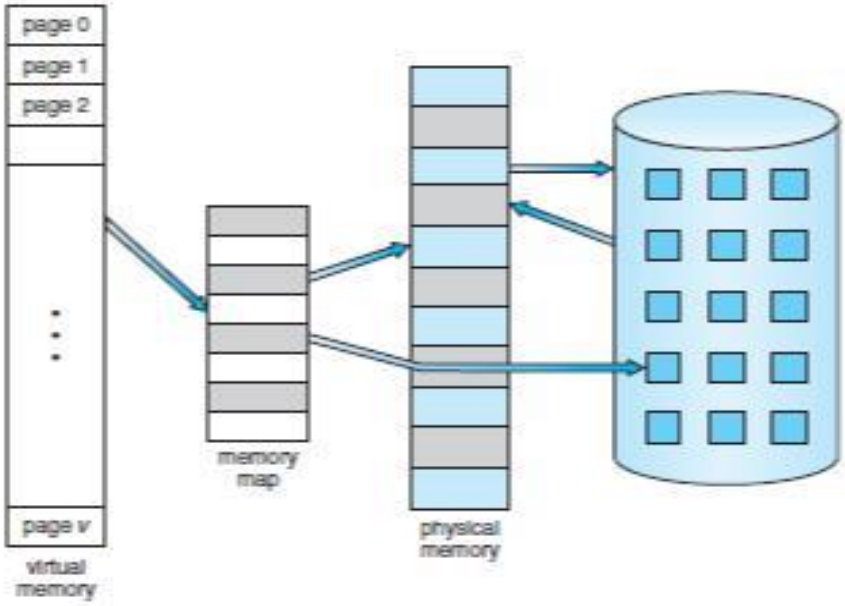
  Virtual memory is the separation of user logical memory from physical memory. This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.

  – Only part of the program needs to be in memory for execution.

  – Logical address space can therefore be much larger than physical address space.

  – Need to allow pages to be swapped in and out.

- **Advantages:**
  - Allows the program that can be larger than the physical memory.
  - Separation of user logical memory from physical memory
  - Allows processes to easily share files & address space.
  - Allows for more efficient process creation.



- Virtual memory can be implemented using
  - Demand paging
  - Demand segmentation

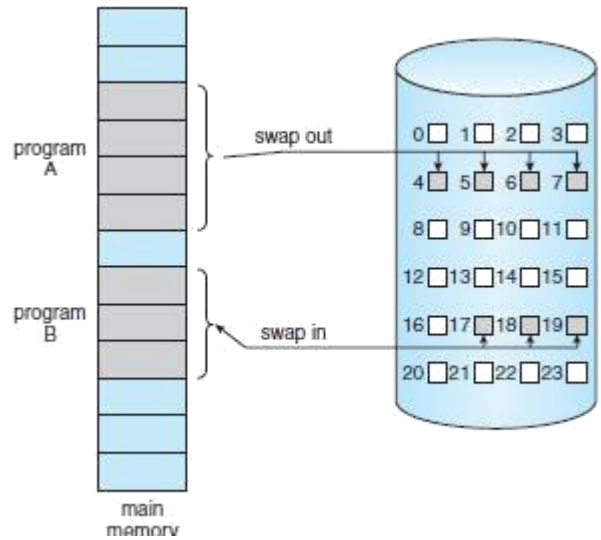
9. DEMAND PAGING

9.1 Concept

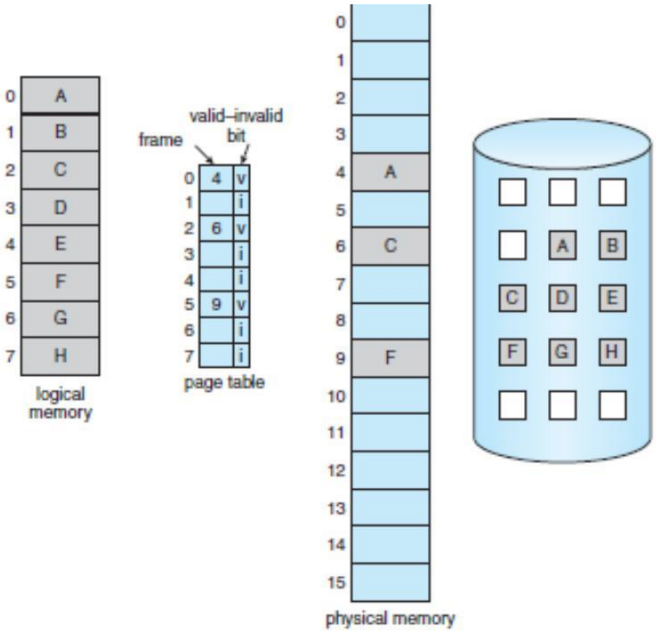
The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them (On demand). This is termed as lazy swapper.

Advantages

- Less I/O needed
- Less memory needed
- Faster response
- More users



Page table when some pages are not in main memory.



The procedure for handling this page fault

1. We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

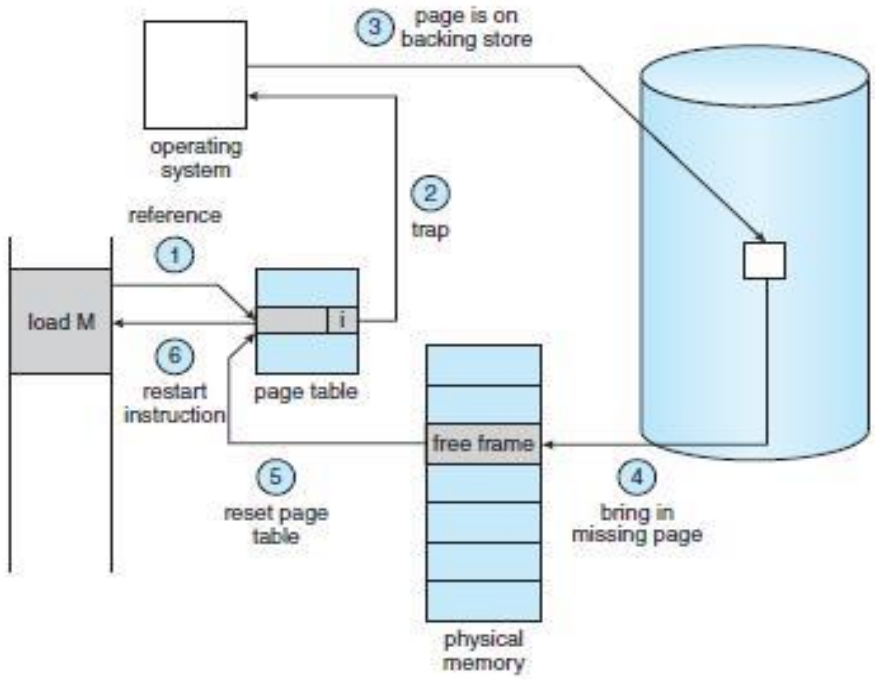
2. If the reference was invalid, we terminate the process. If it was valid but we have not yet brought in that page, we now page it in.

3. We find a free frame (by taking one fromthe free-frame list, for example).

4. We schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.



9.2 Performance of Demand Paging

Effective Access Time (EAT) for a demand-paged memory.
Memory Access Time (ma) for most computers now ranges from 10 to 200 nanoseconds.
If there is no page fault, then EAT = ma.
If there is page fault, then

EAT = (1 – p) x (ma) + p x (page-fault time).
p: the probability of a page fault ($0 \leq p \leq 1$),
we expect p to be close to zero (a few page faults).
If p=0 then no page faults, but if p=1 then every reference is a fault

If a page fault occurs, we must first read the relevant page from disk, and then access the desired word.

Example

Assume an average page-fault service time of 25 milliseconds (10-3), and a Memory Access Time of 100 nanoseconds (10-9). Find the Effective Access Time?

Solution: Effective Access Time (EAT)

= (1 – p) x (ma) + p x (page fault time)
= (1 – p) x 100 + p x 25,000,000
= 100 – 100 x p + 25,000,000 x p
= 100 + 24,999,900 x p.

•Note: The Effective Access Time is directly proportional to the page-fault rate.

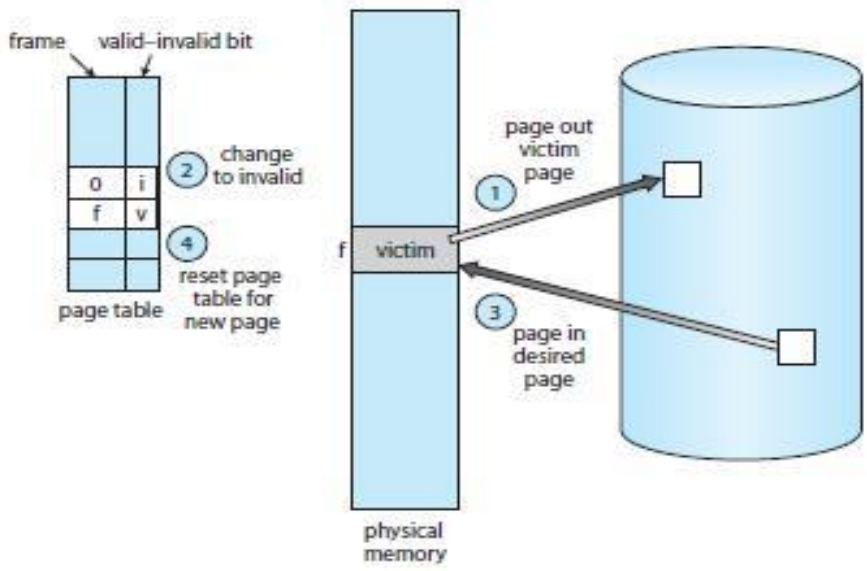
10. PAGE REPLACEMENT

10.1 Page fault

A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Need for page replacement

Page replacement is needed to decide which page needed to be replaced when new page comes in.



1. Find the location of the desired page on the disk.
2. Find a free frame:
   a. If there is a free frame, use it.
   b. If there is no free frame, use a page-replacement algorithm to select a victim frame.
   c. Write the victim frame to the disk; change the page and frame tables accordingly.
3. Read the desired page into the newly freed frame; change the page and frame tables.
4. Continue the user process from where the page fault occurred.

10.2 Page replacement algorithms

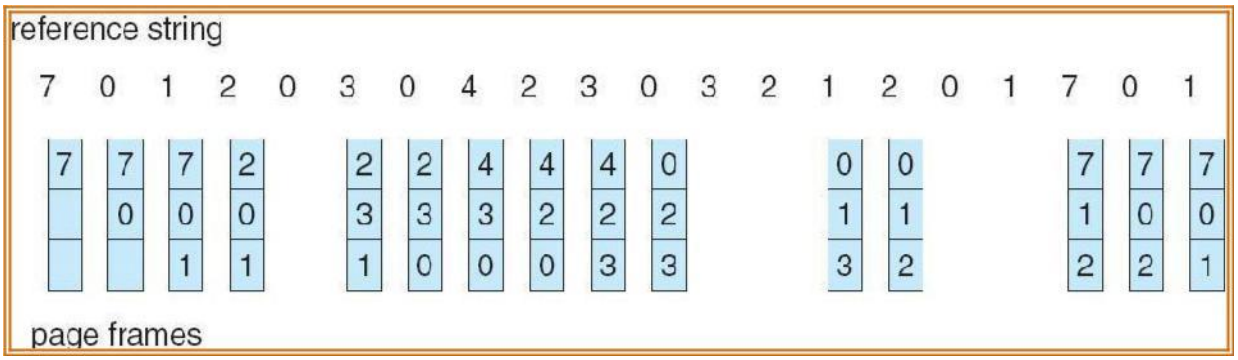
(a) FIFO page replacement algorithm

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No.of available frames = 3 (3 pages can be in memory at a time per process)

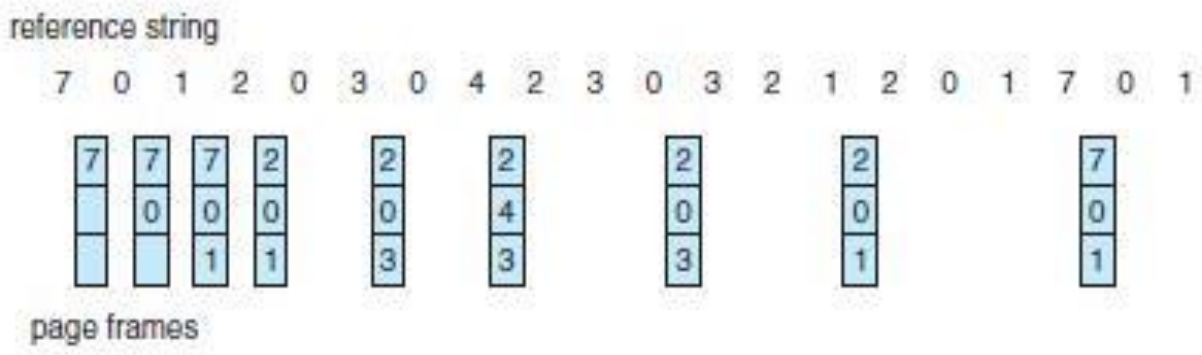


No. of page faults = 15

(b) Optimal page replacement algorithm

In this algorithm, pages are replaced which are not used for the longest duration of time in the future.

Example:



No. of page faults = 9

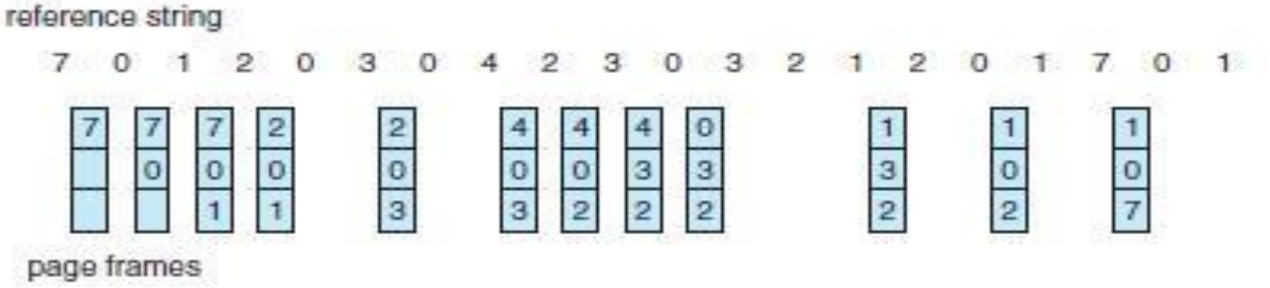
(c) LRU(Least Recently Used) page replacement algorithm

In this algorithm page will be replaced which is least recently used.

Example:

Reference string: 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

No.of available frames = 3

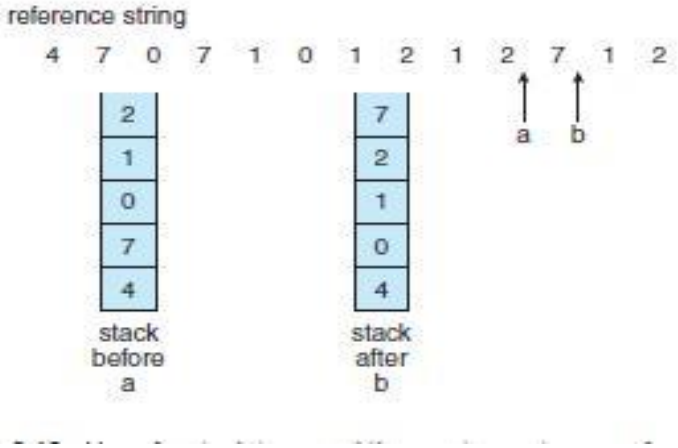


Page Fault =12

Implementation of LRU

1. **Counter**
   - The counter or clock is incremented for every memory reference.
   - Each time a page is referenced , copy the counter into the time- of-use field.
   - When a page needs to be replaced, replace the page with the smallest counter value.
2. **Stack**
   - Keep a stack of page numbers
   - Whenever a page is referenced, remove the page from the stack and put it on top of the stack.
   - When a page needs to be replaced, replace the page that is at the bottom of the stack.(LRU page)

Use of A Stack to Record The Most Recent Page References



(d) LRU Approximation Page Replacement

- Reference bit
  - With each page associate a reference bit, initially set to 0
  - When page is referenced, the bit is set to 1
- When a page needs to be replaced, replace the page whose reference bit is 0
- The order of use is not known , but we know which pages were used and which were not used.

(i) Additional Reference Bits Algorithm

- Keep an 8-bit byte for each page in a table in memory.
- At regular intervals , a timer interrupt transfers control to OS.
- The OS shifts reference bit for each page into higher- order bit shifting the other bits right 1 bit and discarding the lower-order bit.

Example:

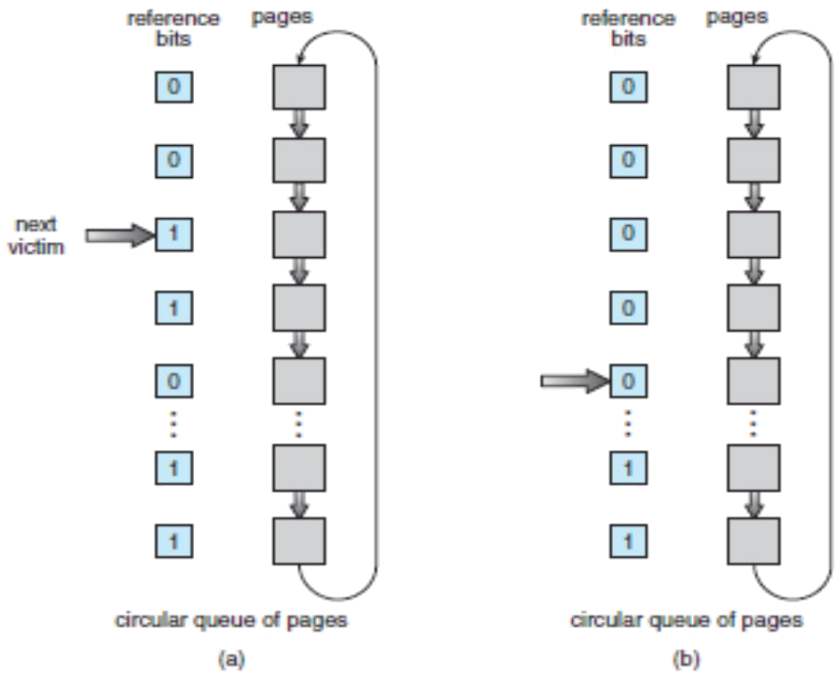
oIf reference bit is 00000000 then the page has not been used for 8 time periods.
oIf reference bit is 11111111 then the page has been used atleast once each time period. oIf the reference bit of page 1 is 11000100 and page 2 is 01110111 then page 2 is the LRU page.

(ii) Second Chance Algorithm

oBasic algorithm is FIFO

oWhen a page has been selected , check its reference bit.

- If 0 proceed to replace the page
- If 1 give the page a second chance and move on to the next FIFO page.
- When a page gets a second chance, its reference bit is cleared and arrival time is reset to current time.
- Hence a second chance page will not be replaced until all other pages are replaced.



(iii) Enhanced Second Chance Algorithm

o Consider both reference bit and modify bit o

There are four possible classes

1. (0,0) – neither recently used nor modified est page to replace
2. (0,1) – not recently used but modifiedpage has to be written out before replacement.
3. (1,0) - recently used but not modified page may be used again
4. (1,1) – recently used and modifiedpage may be used again and page has to be written to disk

(iv) Counting-Based Page Replacement

o Keep a counter of the number of references that have been made to each page

1. **Least Frequently Used (LFU)Algorithm**: replaces page with smallest count
2. **Most Frequently Used (MFU)Algorithm**: replaces page with largest count
   - □ is based on the argument that the page with the smallest count was probably just brought in and has yet to be used

11. ALLOCATION OF FRAMES

11.1 Allocation of Frames

o There are two major allocation schemes

- □ Equal Allocation
- □ Proportional Allocation

Equal allocation

- If there are n processes and m frames then allocate m/n frames to each process.
- **Example:** If there are 5 processes and 100 frames, give each process 20 frames.
- Allocate according to the size of process

Let si be the size of process i.

Let m be the total no. of

frames Then $S = \sum si$

$ai = si / S * m$

where ai is the no.of frames allocated to process i.

11.2 Global vs. Local Replacement

- **Global replacement** – each process selects a replacement frame from the set of all frames; one process can take a frame from another.
- **Local replacement** – each process selects from only its own set of allocated frames.

With proportional allocation, we would split 62 frames between two processes, one of 10 pages and one of 127 pages, by allocating 4 frames and 57 frames

$10/137 \times 62 \approx 4$, and

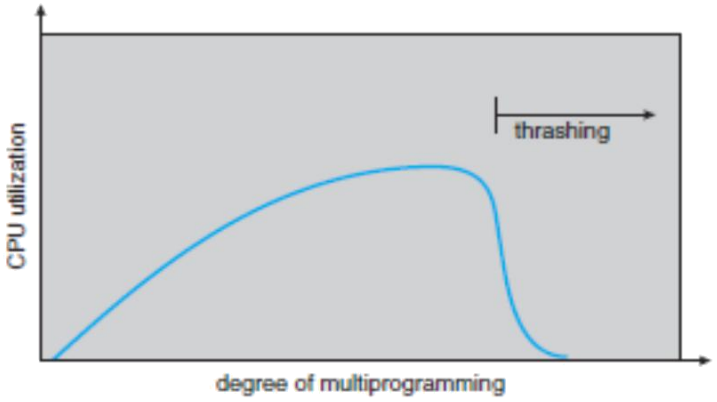
$127/137 \times 62 \approx 57$.

12. THRASHING

Thrashing

- High paging activity is called **thrashing**.
- If a process does not have enough pages, the page-fault rate is very high. This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process is added to the system
- When the CPU utilization is low, the OS increases the degree of multiprogramming.
- If global replacement is used then as processes enter the main memory they tend to steal frames belonging to other processes.
- Eventually all processes will not have enough frames and hence the page fault rate becomes very high.

o Thus swapping in and swapping out of pages only takes place.
o This is the cause of thrashing.

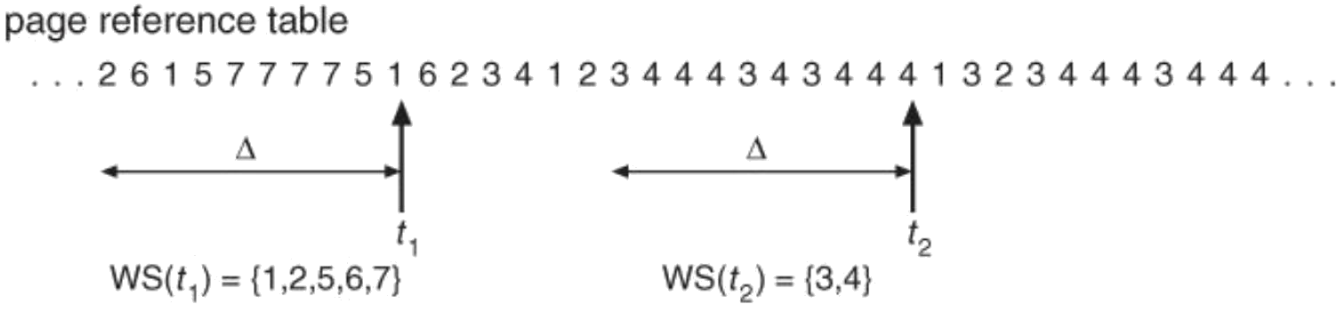


o To **limit thrashing**, we can use a **local replacement** algorithm. o To prevent thrashing, there are two methods namely ,

- Working Set Strategy
- Page Fault Frequency

1. Working-Set Strategy

o It is based on the assumption of the model of locality.
o Locality is defined as the set of pages actively used together.
o Whatever pages are included in the most recent page references are said to be in the processes working set window, and comprise its current working set .



If a page is in active use, it will be in the working set. If it is no longer being used, it will drop from the working set time units after its last reference.

- Thus, the working set is an approximation of the program's locality.
- if Δ = 10 memory references, then the working set at time t1 is {1, 2, 5,6, 7}.
- By time t2, the working set has changed to {3, 4}.
- The accuracy of the working set depends on the selection of .
- If Δ is too small, it will not encompass the entire locality; if is too large, it may overlap several localities.
- In the extreme, if Δ is infinite, the working set is the set of pages touched during the process execution.

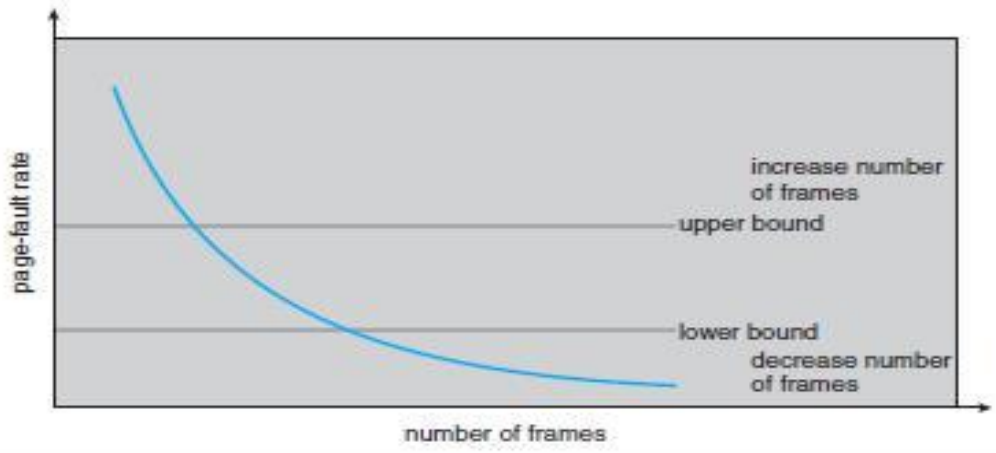
 The most important property of the working set, then, is its size.
 If we compute the working-set size, WSSi , for each process in the system, we can then consider that

$$D = \sum WSS_i,$$

 where D is the total demand for frames. Each process is actively using the pages in its working set.
 Thus, process i needs WSSi frames. If the total demand is greater than the total number of available frames (D> m), thrashing will occur, because some processes will not have enough frames.

2. Page-Fault Frequency Scheme

- Thrashing has a high page-fault rate. Thus, we want to control the page-fault rate.
- When it is too high, we know that the process needs more frames. Conversely, if the page-fault rate is too low, then the process may have too many frames.
- We can establish upper and lower bounds on the desired page-fault rate.
- If the actual page-fault rate exceeds the upper limit, we allocate the process another frame.
- If the page-fault rate falls below the lower limit, we remove a frame from the process.
- Thus, we can directly measure and control the page-fault rate to prevent thrashing.



13. ALLOCATING KERNEL MEMORY

Allocating Kernel Memory

When a process running in user mode requests additional memory, pages are allocated from the list of free page frames maintained by the kernel. This list is typically populated using a page-replacement algorithm such as those discussed in Section 9.4 and most likely contains free pages scattered throughout physical memory, as explained earlier. Remember, too, that if a user process requests a single byte of memory, internal fragmentation will result, as the process will be granted an entire page frame.

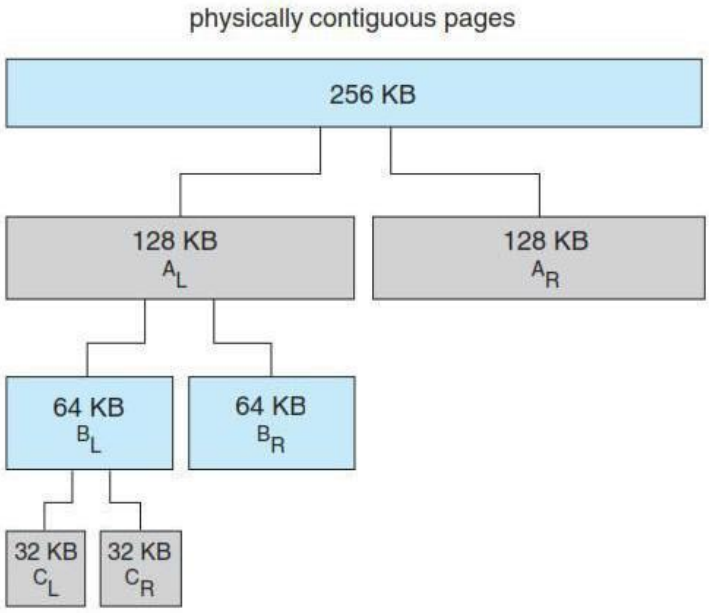
Kernel memory is often allocated from a free-memory pool different from the list used to satisfy ordinary user-mode processes. There are two primary reasons for this:

1. The kernel requests memory for data structures of varying sizes, some of which are less than a page in size. As a result, the kernel must use memory conservatively and attempt to minimize waste due to fragmentation. This is especially important because many operating systems do not subject kernel code or data to the paging system.

2. Pages allocated to user-mode processes do not necessarily have to be in contiguous physical memory. However, certain hardware devices interact directly with physical memory—without the benefit of a virtual memory interface—and consequently may require memory residing in physically contiguous pages.

Buddy System

The buddy system allocates memory from a fixed -size segment consisting of physically contiguous pages. Memory is allocated from this segment using a **power-of-2 allocator**, which satisfies requests in units sized as a power of 2 (4KB, 8KB, 16KB, and so forth). A request in units not appropriately sized is rounded up to the next highest power of 2. For example, a request for 11 KB is satisfied with a 16K segment



Let's consider a simple example. Assume the size of a memory segment is initially 256 KB and the kernel requests 21 KB of memory.

The segment is initially divided into two buddies—which we will call AL and AR—each 128 KB in size. One of these buddies is further divided into two 64-KB buddies— BL and BR.

However, the next-highest power of 2 from 21 KB is 32 KB so either BL or BR is again divided into two 32-KB buddies, CL and CR. One of these buddies is used to satisfy the 21-KB request.

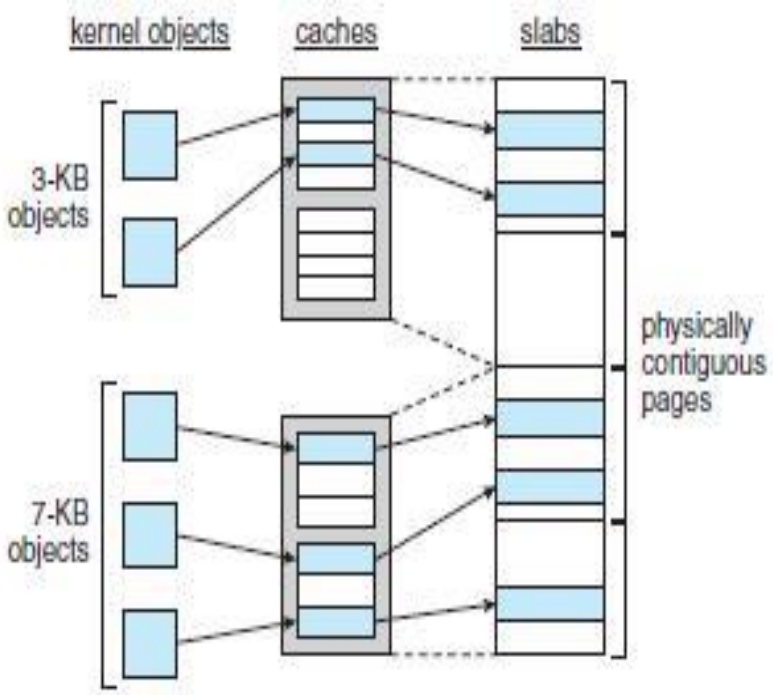
Slab Allocation

A second strategy for allocating kernel memory is known as slab allocation. A slab is made up of one or more physically contiguous pages. A cache consists of one or more slabs.

The slab-allocation algorithm uses caches to store kernel objects.

When a cache is created, a number of objects which are initially marked as free are allocated to the cache. The number of objects in the cache depends on the size of the associated slab.

For example, a 12-KB slab (made up of three contiguous 4-KB pages) could store six 2-KB objects.



In Linux, a slab may be in one of three possible states:

1. Full. All objects in the slab are marked as used.
2. Empty. All objects in the slab are marked as free.
3. Partial. The slab consists of both used and free objects.

The slab allocator first attempts to satisfy the request with a free object in a partial slab.

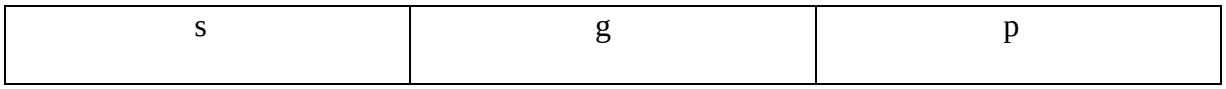
If none exists, a free object is assigned from an empty slab.

If no empty slabs are available, a new slab is allocated from contiguous physical pages and assigned to a cache; memory for the object is allocated from this slab.

14. SEGMENTATION WITH PAGING

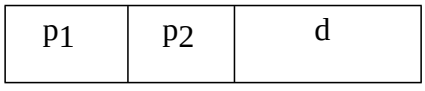
- The IBM OS/ 2.32 bit version is an operating system running on top of the Intel 386 architecture. The 386 uses segmentation with paging for memory management. The maximum number of segments per process is 16 KB, and each segment can be as large as 4 gigabytes.
- The local-address space of a process is divided into two partitions.
  The first partition consists of up to 8 KB segments that are private to that process.
  The second partition consists of up to 8KB segments that are shared among all the processes.
- Information about the first partition is kept in the **local descriptor table (LDT)**, information about the second partition is kept in the **global descriptor table (GDT)**.
- Each entry in the LDT and GDT consist of 8 bytes, with detailed information about a particular segment including the base location and length of the segment.

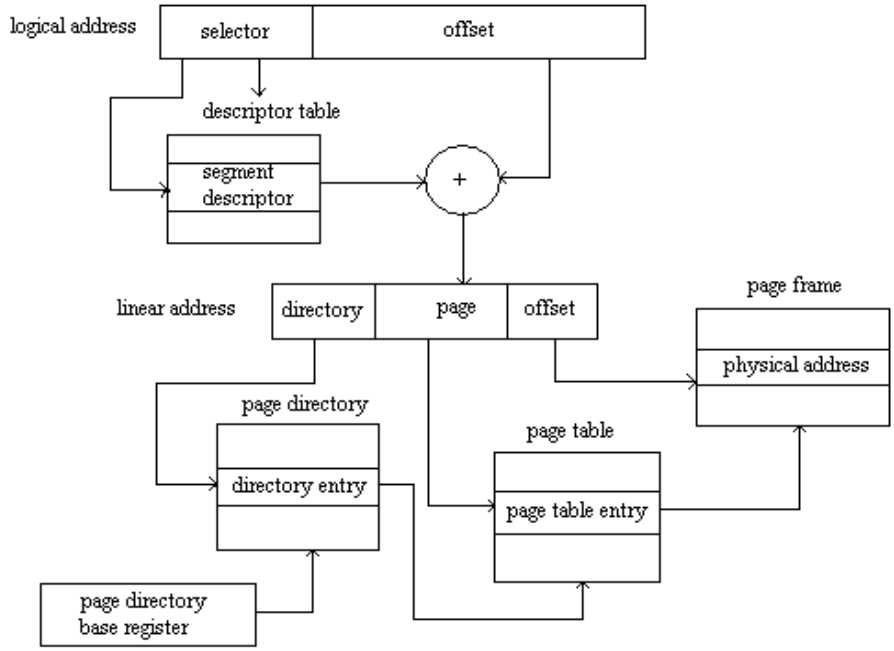
The logical address is a pair (selector, offset) where the selector is a16-bit number:



Where s designates the segment number, g indicates whether the segment is in the GDT or LDT, and p deals with protection. The offset is a 32-bit number specifying the location of the byte within the segment in question.

- The base and limit information about the segment in question are used to generate a linear-address.
- First, the limit is used to check for address validity. If the address is not valid, a memory fault is generated, resulting in a trap to the operating system. If it is valid, then the value of the offset is added to the value of the base, resulting in a 32-bit linear address. This address is then translated into a physical address.
- The linear address is divided into a page number consisting of 20 bits, and a page offset consisting of 12 bits. Since we page the page table, the page number is further divided into a 10-bit page directory pointer and a 10-bit page table pointer. The logical address is as follows.





oTo improve the efficiency of physical memory use. Intel 386 page tables can be swapped to disk. In this case, an invalid bit is used in the page directory entry to indicate whether the table to which the entry is pointing is in memory or on disk.

oIf the table is on disk, the operating system can use the other 31 bits to specify the disk location of the table; the table then can be brought into memory on demand.

UNIT IV FILE SYSTEMS AND I/O SYSTEMS

Mass Storage system – Overview of Mass Storage Structure, Disk Structure, Disk Scheduling and Management, swap space management; File-System Interface – File concept, Access methods, Directory Structure, Directory organization, File system mounting, File Sharing and Protection; File System Implementation- File System Structure, Directory implementation, Allocation Methods, Free Space Management, Efficiency and Performance, Recovery; I/O Systems – I/O Hardware, Application I/O interface, Kernel I/O subsystem, Streams, Performance.

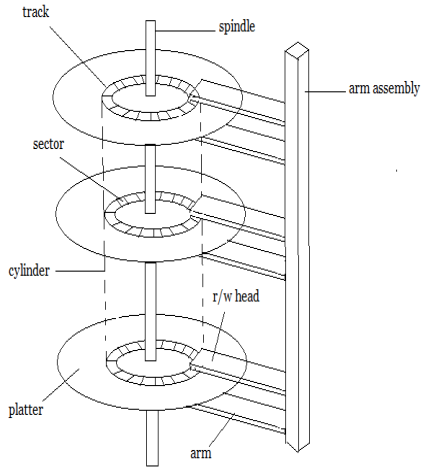
MASS STORAGE STRUCTURE

1. Overview of Mass Storage Structure

Magnetic Disks

- In modern computers, most of the secondary storage is in the form of magnetic disks.
- A magnetic disk contains several platters. Each platter is divided into circular shaped tracks.
- The length of the tracks near the centre is less than the length of the tracks farther from the centre.
- Each track is further divided into sectors.
- Tracks of the same distance from centre form a cylinder.
- A read-write head is used to read data from a sector of the magnetic disk.
- The speed of the disk is measured as two parts:

Transfer rate: This is the rate at which the data moves from disk to the computer.
Random access time: It is the sum of the seek time and rotational latency.
Seek time is the time taken by the arm to move to the required track.
Rotational latency is defined as the time taken by the arm to reach the required sector in the track.



Solid-State Disks

SD is non-volatile memory that is used like a hard drive. SSDs have the same characteristics as traditional hard disks but can be more reliable because they have no

moving parts and faster because they have no seek time or latency. In addition, they consume less power.

SSDs have less capacity than the larger hard disks, and may have shorter life spans. use for SSDs is in storage arrays, where they hold file- system metadata that require high performance. Some SSDs are designed to connect directly to the system bus.

Magnetic Tapes

Magnetic tape was used as an early secondary-storage medium. Although it is relatively permanent and can hold large quantities of data, its access time is slow compared with that of main memory and magnetic disk.

In addition, random access to magnetic tape is about a thousand times slower than random access to magnetic disk, so tapes are not very useful for secondary storage.

2. Disk Structure

In Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer. n The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially.

➔In Sector 0 is the first sector of the first track on the outermost cylinder.

➔In Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost.

3. Disk Scheduling

The operating system is responsible for using hardware efficiently.

For the disk drives, this means having a fast access time & disk bandwidth.

➔Access time has two major components:

➔Seek time is the time for the disk to move the heads to the cylinder containing the desired sector

➔Rotational latency time waiting for the disk to rotate the desired sector to the disk head

➔We like to minimize seek time.

➔ Disk bandwidth is the total number of bytes transferred divided by the total time between the first request for service and the completion of the last transfer.

➔Several algorithms exist to schedule the servicing of disk I/O requests.

We illustrate them with a Request Queue (cylinder range 0-199):

98, 183, 37, 122, 14, 124, 65, 67

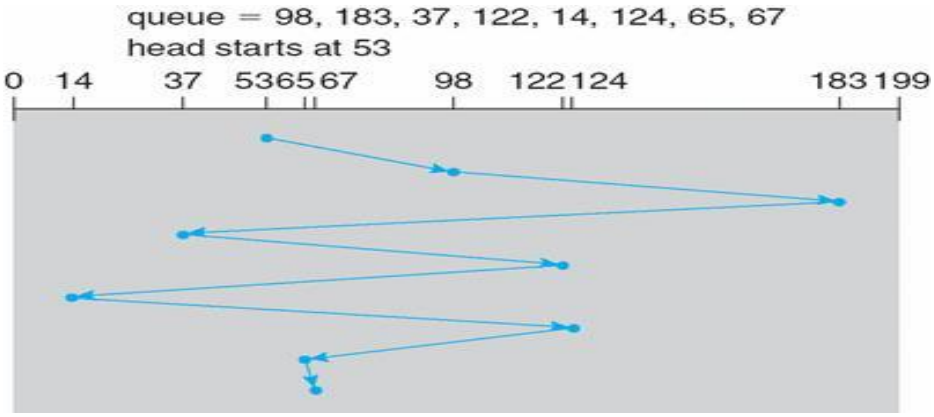
Head pointer: cylinder 53

1. First Come First Serve

This algorithm performs requests in the same order asked by the system. Let's take an example where the queue has the following requests with cylinder numbers as follows:

98, 183, 37, 122, 14, 124, 65, 67

Illustration shows total head movement of 640 cylinders

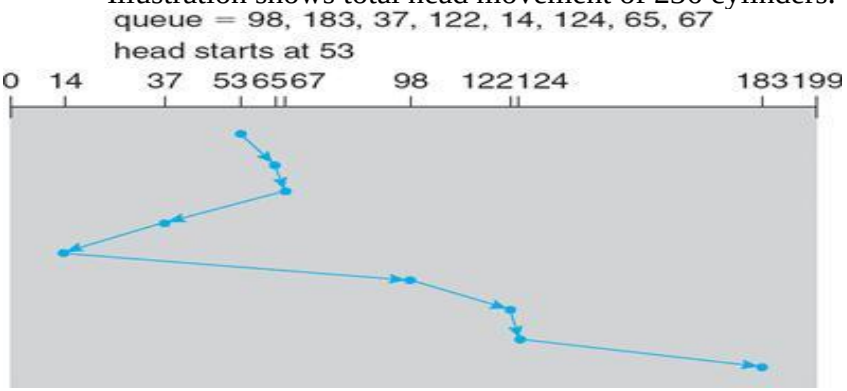


2. SSTF (Shortest Seek Time First)

Selects the request with the minimum seek time from the current head position.

SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

Illustration shows total head movement of 236 cylinders.

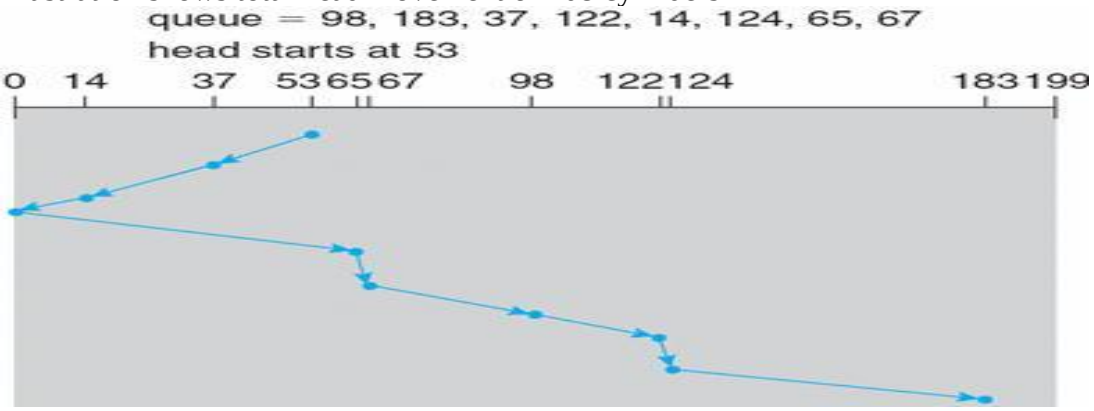


3. **SCAN**

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

SCAN algorithm sometimes called the elevator algorithm.

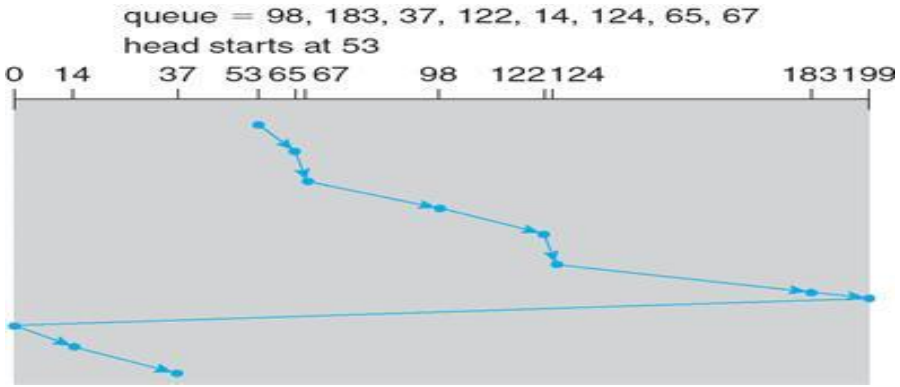
Illustration shows total head movement of 208 cylinders



4. C-SCAN

Provides a more uniform wait time than SCAN. The head moves from one end of the disk to the other, servicing requests as it goes When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.

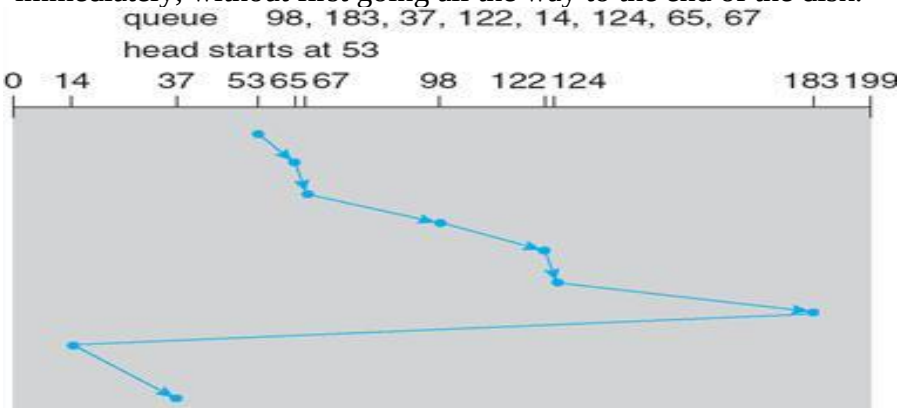
Treats the cylinders as a Circular list that wraps around from the last cylinder to the first one .



5.LOOK

➔ Version of C-SCAN

➔ Arm only goes as far as the **last request** in each direction,then reverses direction immediately, without first going all the way to the end of the disk.



4. Disk Management

The operating system is responsible for disk initialization, booting from disk, and bad-block recovery.

Disk Formatting

A new magnetic disk must be divided into sectors that the disk controller can read and write. This process is called **low-level formatting, or physical formatting**. Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector typically consists of a header, a data area (usually 512 bytes in size), and a trailer.

The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code (ECC).**

This formatting enables the manufacturer to 1. Test the disk and 2. To initialize the mapping from logical block numbers

To use a disk to hold files, the operating system still needs to record its own data structures on the disk.

It does so in two steps.

(a) The first step is **Partition** the disk into one or more groups of cylinders. Among the partitions,one partition can hold a copy of the OS‘s executable code, while another holds user files.

(b) The second step is **logical formatting**. The operating system stores the initial file-system data structures onto the disk. These data structures may include maps of free and allocated space and an initial empty directory.

Boot Block

For a computer to start running-for instance, when it is powered up or rebooted-it needs to have an initial program to run. This initial program is called bootstrap program & it should be simple.

It initializes all aspects of the system, from CPU registers to device controllers and the contents of main memory, and then starts the operating system.

The bootstrap is stored in read-only memory (ROM). This location is convenient, because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. And, since ROM is read only, it cannot be infected by a computer virus.

The full bootstrap program is stored in the “**boot blocks**” at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk. The work of boot block as follows

1. Finds the operating system kernel on disk,
2. Loads that kernel into memory, and
3. Jumps to an initial address to begin the operating-system execution.

The full **bootstrap program** is stored in a partition called the boot blocks, at a fixed location on the disk. A disk that has a boot partition is called a boot disk or system disk.

The code in the boot ROM instructs the disk controller to read the boot blocks into memory and then starts executing that code.

Bootstrap loader - load the entire operating system from a non-fixed location on disk, and to start the operating system running.

Bad Blocks

The disk with defected sector is called as bad block. Depending on the disk and controller in use, these blocks are handled in a variety of ways;

Method 1: "Handled manually‖

If blocks go bad during normal operation, a **special program** must be run manually to search for the bad blocks and to lock them away as before. Data that resided on the bad blocks usually are lost.

Method 2: "sector sparing or forwarding"

The controller maintains a list of bad blocks on the disk. Then the controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

A typical bad-sector transaction might be as follows:

- The operating system tries to read logical block 87.
- The controller calculates the ECC and finds that the sector is bad.
- It reports this finding to the operating system.
- The next time that the system is rebooted, a special command is run to tell the controller to replace the bad sector with a spare.
- After that, whenever the system requests logical block 87, the request is translated into the replacement sector's address by the controller.

Method 3: "sector slipping"

For an example, suppose that logical block 17 becomes defective, and the first available spare follows sector 202. Then, sector slipping would remap all the sectors from 17 to 202, moving them all down one spot. That is, sector 202 would be copied into the spare, then sector 201 into 202, and then 200 into 201, and so on, until sector 18 is copied into sector 19. Slipping the sectors in this way frees up the space of sector 18, so sector 17 can be mapped to it.

5. Swap-Space Management

➔Swap-space — virtual memory uses disk space as an extension of main memory.

➔Main goal for the design and implementation of swap space is to provide the best throughput for VM system

1.Swap-space use

Swapping –use swap space to hold entire process image

Paging –store pages that have been pushed out of memory

➔Some OS may support multiple swap-space

–Put on separate disks to balance the load

➔Better to overestimate than underestimate

–If out of swap-space, some processes must be aborted or system crashed

2**. Swap-Space Location**

➔Swap-space can be carved out of the normal file system, or in a separate disk partition

➔A large file within the file system: simple but inefficient

–Navigating the directory structure and the disk-allocation data structure takes time and potentially extra disk accesses

–External fragmentation can greatly increase swapping times by forcing multiple seeks during reading or writing of a process image

–Improvement

Caching block location information in main memory

•Contiguous allocation for the swap file

But, the cost of traversing FS data structure still remains

➔In a separate partition: raw partition

–Create a swap space during disk partitioning

–A separate swap-space storage manager is used to allocate and de-allocate blocks

–Use algorithms optimized for speed, rather than storage efficiency

–Internal fragment may increase

Linux supports both approaches

➔Swap-space Management: Example

Solaris 1

–Text-segment pages are brought in from the file system and are thrown away if selected for paged out

More efficient to re-read from FS than write it to the swap space

-Swap space: only used as a backing store for pages of anonymous memory

Stack, heap, and uninitialized data

Solaris 2

–Allocates swap space only when a page is forced out of physical memory

Not when the virtual memory page is first created.

FILE SYSTEM INTERFACE

1. File Concepts

A file is a named collection of related information that is recorded on secondary storage. From user's perspective a , a file is the smallest allotment of that logical secondary storage; unless they are within a file. Commonly, files represent programs (both source and object forms) and data. Data files may be numeric, alphabetic, alphanumeric, or binary.

In general, a file is a sequence of bits, bytes, lines, or records, the meaning of which is defined by the file's creator and user.

A **text file** is a sequence of characters organized into lines (and possibly pages).

An **executable** file is a series of code sections that the loader can bring into memory and execute.

2. File Attributes

The information about all files is kept in the directory structure, a directory entry consists of the file's name and its unique identifier. The identifier in turn locates the other file attributes.

- **Name:** The symbolic file name is the only information kept in human readable form.

- **Identifier:** This unique tag, usually a number identifies the file within the file system. It is the non-human readable name for the file.
- **Type:** This information is needed for those systems that support different types.
- **Location:** This information is a pointer to a device and to the location of the file on that device.
- **Size:** The current size of the file (in bytes, words or blocks) and possibly the maximum allowed size are included in this attribute.
- **Protection:** Access-control information determines who can do reading, writing, executing and so on.
- **Time, date and user identification:** This information may be kept for creation, last modification and last use. These data can be useful for protection, security and usage monitoring.

3.File Operations

The operating system can provide system calls to create, write, read, reposition, delete, and truncate files.

Creating a file - First, space in the file system must be found for the file, Second, an entry for the new file must be made in the directory.

Writing a file - System call specifying both the name of the file and the information to be written to the file.

Reading a file - we use a system call that specifies the name of the file and where (in memory) the next block of the file should be put.

Repositioning within a file - The directory is searched for the appropriate entry, and the current-file-position pointer is repositioned to a given value.

Deleting a file - search the directory for the named file. Having found the associated directory entry, we release all file space

Truncating a file - this function allows all attributes to remain unchanged—except for file length—but lets the file be reset to length zero and its file space released.

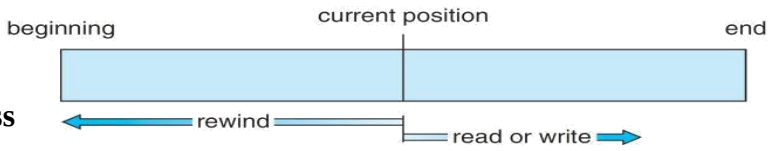
4. **File Types**

file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, perl, asm	source code in various languages
batch	bat, sh	commands to the command interpreter
markup	xml, html, tex	textual data, documents
word processor	xml, rtf, docx	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	gif, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	rar, zip, tar	related files grouped into one file, sometimes compressed, for archiving or storage
multimedia	mpeg, mov, mp3, mp4, avi	binary file containing audio or A/V information

5. **Access Methods**

1.Sequential Access

- Data is accessed one record right after another is an order.
- Read command cause a pointer to be moved ahead by one.
- Write command allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape.



2. **Direct Access**

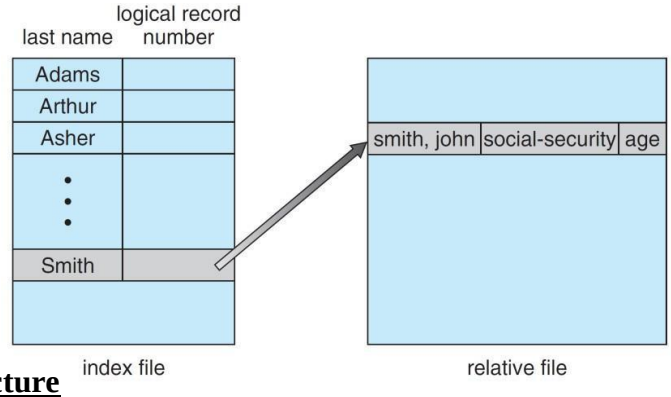
- This method is useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read/written, it can be dobe in any order.
- User now says "read n" rather than "read next".
- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

As a simple example, on an **airline – reservation system**, we might store all the information about a particular flight (for example, flight 713) in the block identified by the flight number.

Thus, the number of available seats for flight 713 is stored in block 713 of the reservation file. To store information about a larger set, such as people, we might compute a hash function on the people's names, or search a small in-memory index to determine a block to read and search.

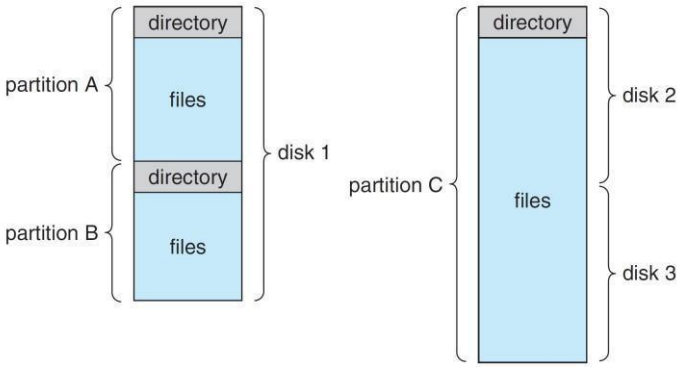
3. Indexed Access

- If a file can be sorted on any of the filed then an index can be assigned to a group of certain records.
- However, A particular record can be accessed by its index.
- The index is nothing but the address of a record in the file.
- In index accessing, searching in a large database became very quick and easy but we need to have some extra space in the memory to store the index value.



6.Directory Structure

- Directory can be defined as the listing of the related files on the disk.
- The directory may store some or the entire file attributes.
- Each partition must have at least one directory in which, all the files of the partition can be listed.
- A directory entry is maintained for each file in the directory which stores all the information related to that file.



Operations that are to be performed on a directory

Search for a file. We need to be able to search a directory structure to find the entry for a particular file. Since files have symbolic names, and similar names

may indicate a relationship among files, we may want to be able to find all files whose names match a particular pattern.

Create a file. New files need to be created and added to the directory.
Delete a file. When a file is no longer needed, we want to be able to remove it from the directory.
List a directory. We need to be able to list the files in a directory and the contents of the directory entry for each file in the list.
Rename a file. Because the name of a file represents its contents to its users, we must be able to change the name when the contents or use of the file changes. Renaming a file may also allow its position within the directory structure to be changed.
Traverse the file system. We may wish to access every directory and every file within a directory structure. For reliability, it is a good idea to save the contents and structure of the entire file system at regular intervals.

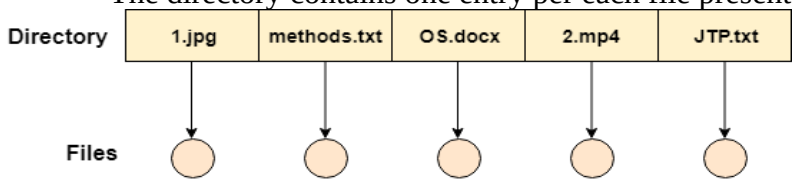
➔Often, we do this by copying all files to magnetic tape. This technique provides a backup copy in case of system failure.
➔In addition, if a file is no longer in use, the file can be copied to tape and the disk space of that file released for reuse by another file.

Logical Structure (or) Level of Directory

- Single-level directory
- Two-level directory
- Tree-Structured directory
- Acyclic Graph directory
- General Graph directory

Single – Level Directory

- The simplest method is to have one big list of all the files on the disk.
- The entire system will contain only one directory which is supposed to mention all the files present in the file system.
- The directory contains one entry per each file present on the file system.



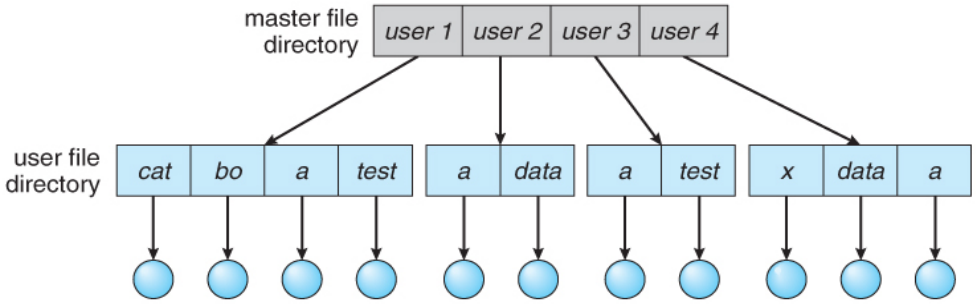
Single Level Directory

Disadvantages

1. We cannot have two files with the same name.
2. The directory may be very big therefore searching for a file may take so much time.
3. Protection cannot be implemented for multiple users.
4. There are no ways to group same kind of files.

Two Level Directory

- In two level directory systems, we can create a separate directory for each user.
- There is one master directory which contains separate directories dedicated to each user. For each user, there is a different directory present at the second level, containing group of user's file.
- The system doesn't let a user to enter in the other user's directory without permission.



Characteristics of two level directory system

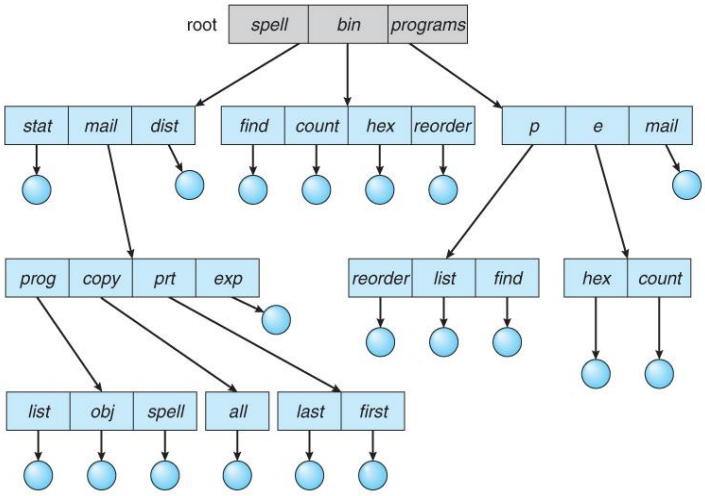
1.Each files has a path name as /User-name/directory-name/

2. Different users can have the same file name.

3. Searching becomes more efficient as only one user's list needs to be traversed.

Tree Structured Directory

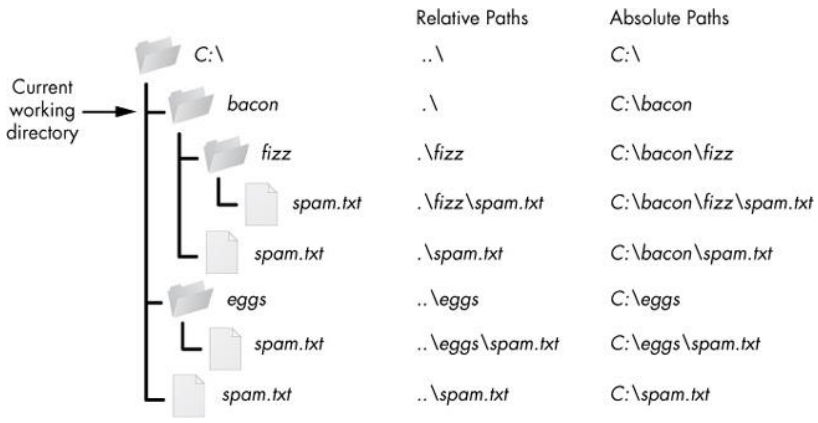
- Tree structured directory system overcomes the drawbacks of two level directory system.
- The similar kind of files can now be grouped in one directory.
- Each user has its own directory and it cannot enter in the other user's directory.
- Searching is more efficient in this directory structure



A file can be accessed by two types of path, either ➔ 1.Relative or 2. Absolute.

1. **Absolute path** is the path of the file with respect to the root directory of the system.

2. **Relative path** is the path with respect to the current working directory of the system

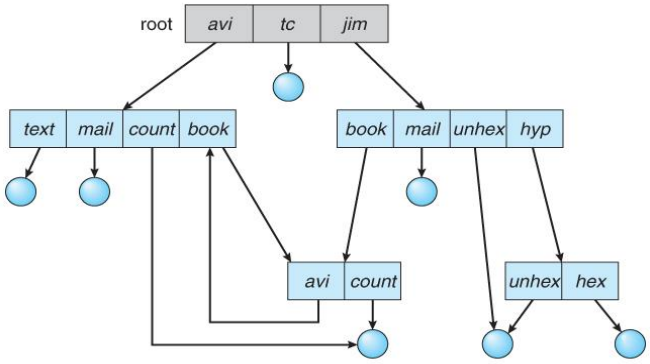


Acyclic-Graph Structured Directories

- ❖ When the **same files need to be accessed** in **more than one place** in the directory structure it can be useful to provide an acyclic-graph structure.
- ❖ In this system two or more directory entry can point to the same file or sub directory. That file or sub directory is shared between the two directory entries.
  It provides two types of ***links*** for implementing the acyclic-graph structure
  Soft link, the file just gets deleted and we are left with a dangling pointer.
  Hard link, the actual file will be deleted only if all the references to it gets deleted.

General Graph Directory

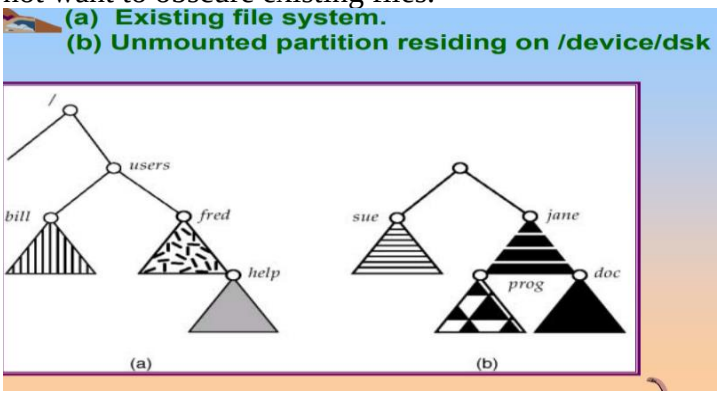
- ❖ In general graph directory structure, cycles are allowed within a directory structure where multiple directories can be derived from more than one parent directory
- ❖ The main problem with this kind of directory structure is to calculate total size or space that have been taken by the files and directories.

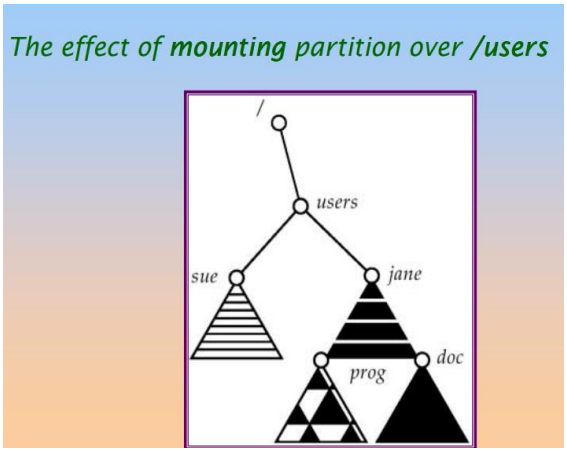


<u>7.File System Mounting</u>

- ❖ Before you can access the files on a file system, you need to mount the file system.

- Mounting a file system attaches that file system to a directory (mount point) and makes it available to the system.
- The root (/) file system is always mounted. Any other file system can be connected or disconnected from the root (/) file system.
- When you mount a file system, any files or directories in the underlying mount point directory are unavailable as long as the file system is mounted.
- These files are not permanently affected by the mounting process, and they become available again when the file system is unmounted.
- However, mount directories are typically empty, because you usually do not want to obscure existing files.





8. File Sharing

- File sharing is the accessing or sharing of files by one or more users.
- File sharing is performed on computer networks as an easy and quick way to transmit data.

For example, a user may share an instruction document on his computer that is connected to a corporate network allowing all other employees to access and read that document.

1.Multiple Users

- On a multi-user system, more information needs to be stored for each file:The owner (user) who owns the file, and who can control its access.

- ❖ The group of other user IDs that may have some special access to the file.
- ❖ What access rights are afforded to the owner (User), the Group, and to the rest of the world.

2. Remote File Systems

The advent of the Internet introduces issues for accessing files stored on remote computers

- ❖ The original method was ftp, allowing individual files to be transported across systems as needed.
- ❖ The Client-Server Model(the system which physically owns the files acts as a *server*, and the system which mounts them is the *client.)*
- ❖ Distributed Information Systems➔ service that runs on a single central location.
- ❖ Failure Modes ➔ When a local disk file is unavailable, the result is generally known immediately, and is generally non-recoverable. The only reasonable response is for the response to fail. Remote access systems allow for blocking or delayed response.

3. Consistency Semantics

Consistency Semantics deals with the consistency between the views of shared files on a networked system. When one user changes the file, when do other users see the changes?

1. UNIX Semantics

➔ Writes to an open file are immediately visible to any other user who has the file open.

2. Session Semantics

AFS uses the following semantics:

➔Writes to an open file are not immediately visible to other users.

➔When a file is closed, any changes made become available only to users who open the file at a later time.

3. Immutable-Shared-Files Semantics

➔ when a file is declared as *shared* by its creator, it becomes immutable and the name cannot be re-used for any other resource. Hence it becomes read-only, and shared access is simple.

9. File Protection

- ❖ Files must be kept safe for reliability (against accidental damage), and protection (against deliberate malicious access.) The former is usually managed with backup copies. This section discusses the latter.
- ❖ One simple protection scheme is to remove all access to a file. However this makes the file unusable, so some sort of controlled access must be arranged.

Types of Access

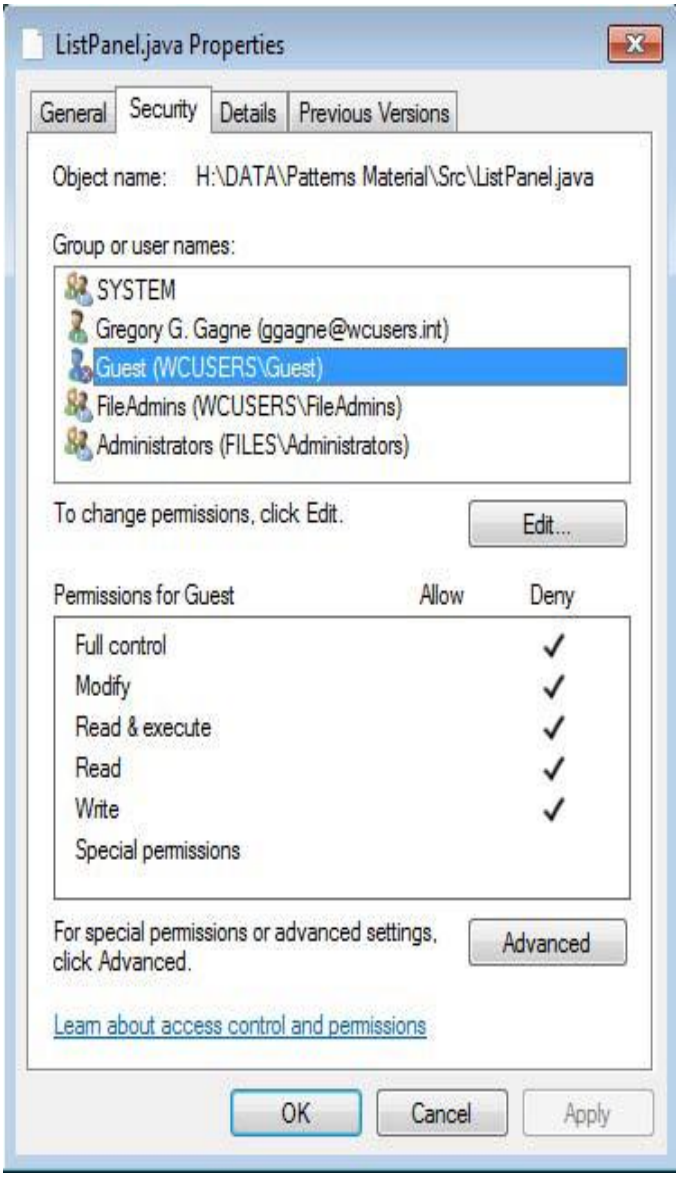
- The following low-level operations are often controlled:

    - Read - View the contents of the file
    - Write - Change the contents of the file.
    - Execute - Load the file onto the CPU and follow the instructions contained therein.
    - Append - Add to the end of an existing file.
    - Delete - Remove a file from the system.
    - List -View the name and other attributes of files on the system.
- Higher-level operations, such as copy, can generally be performed through combinations of the above.

Access Control

- One approach is to have complicated ***Access Control Lists, ACL,*** which specify exactly what access is allowed or denied for specific users or groups.
    - The AFS uses this system for distributed access.
    - Control is very finely adjustable, but may be complicated, particularly when the specific users involved are unknown. (AFS allows some wild cards, so for example all users on a certain remote system may be trusted, or a given username may be trusted when accessing from any remote system.)
- UNIX uses a set of 9 access control bits, in three groups of three. These correspond to R, W, and X permissions for each of the Owner, Group, and Others. (See "man chmod" for full details.) The RWX bits control the following privileges for ordinary files and directories:

bit	Files	Directories
R	**Read (view) file contents.**	**Read directory contents. Required to get a listing of the directory.**
W	**Write (change) file contents.**	**Change directory contents. Required to create or delete files.**
X	**Execute file contents as a program.**	**Access detailed directory information. Required to get a long listing, or to access any specific file in the directory. Note that if a user has X but not R permissions on a directory, they can still access specific files, but only if they already know the name of the file they are trying to access.**



FILE SYSTEM IMPLEMENTATION

1. File System Structure

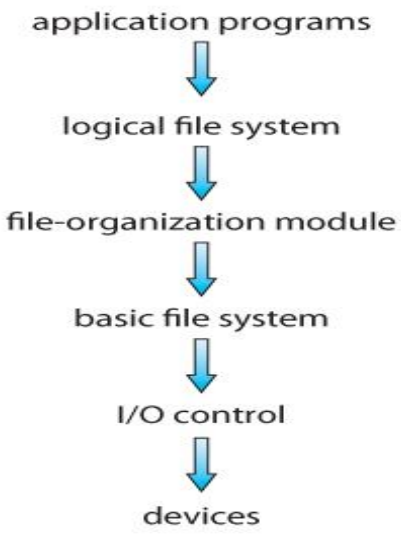
- File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way.
- A file System must be able to store the file, locate the file and retrieve the file.
- Most of the Operating Systems use layering approach for every task including file systems.
- Every layer of the file system is responsible for some activities.

Logical file system

✂ **Provides** users the view of a contiguous sequence of words, bytes stored somewhere.

✂ Uses a directory structure, symbolic name
✂Provides protection and security
✂OS/user interface

☎E.g., to create a new file the API provides a call that calls the logical file system



The file organization module

✂Knows about files and their logical blocks (say 1,..N)
✂Files are organized in blocks of 32 bytes to 4K bytes
✂Translates logical blocks into physical
✂Knows location of file, file allocation type
✂Includes a free space manages that tracks unallocated blocks

Basic file system

✂Issues commands to the device driver (layer of software that directly controls disk hardware) to read and write physical blocks on the disk,
✂Each physical block identified by a disk address (e.g., drive 2, cylinder 34, track 2, sector 11)

IO control

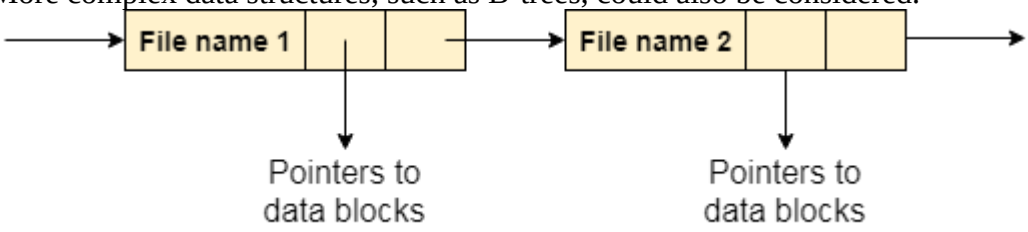
✂The lowest level in the file system
✂Consists of device drivers and interrupt handlers to transfer information between the memory and the disk
✂A device driver translates commands such as "get me block 111" into hardware specific ISA used by hardware controller. This is accomplished by writing specific bits into IO registers

2.Directory Implementation

- Directories need to be fast to search, insert, and delete, with a minimum of wasted disk space.

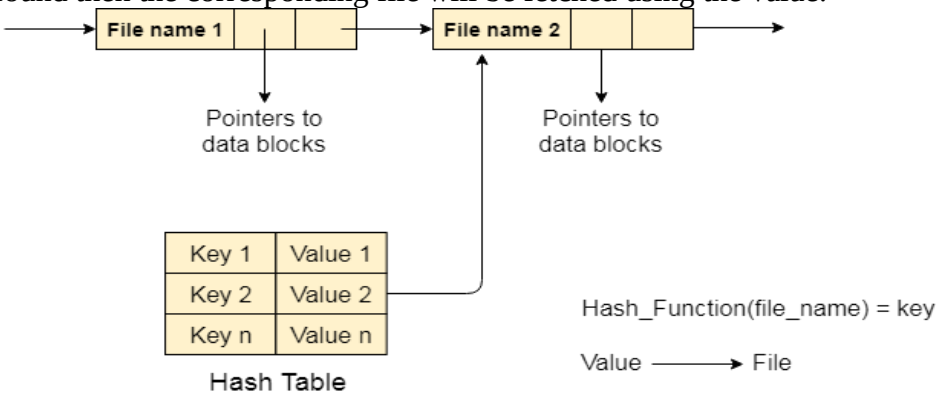
Linear List

- ❖ A linear list is the simplest and easiest directory structure to set up, but it does have some drawbacks.
- ❖ Finding a file (or verifying one does not already exist upon creation) requires a linear search.
- ❖ Deletions can be done by moving all entries, flagging an entry as deleted, or by moving the last entry into the newly vacant position.
- ❖ Sorting the list makes searches faster, at the expense of more complex insertions and deletions.
- ❖ A linked list makes insertions and deletions into a sorted list easier, with overhead for the links.
- ❖ More complex data structures, such as B-trees, could also be considered.



Hash Table

- ❖ A hash table can also be used to speed up searches.
- ❖ Hash tables are generally implemented in addition to a linear or other structure.
- ❖ A key-value pair for each file in the directory gets generated and stored in the hash table.
- ❖ The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.
- ❖ **Searching➔** Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.

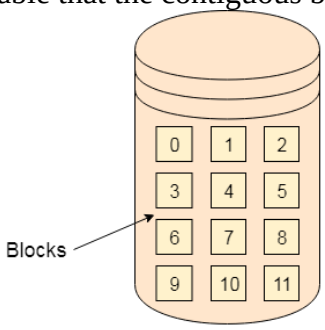


3. Allocation Methods

- There are various methods which can be used to allocate disk space to the files. Selection of an appropriate allocation method will significantly affect the performance and efficiency of the system.
- Allocation method provides a way in which the disk will be utilized and the files will be accessed.

Contiguous Allocation

- If the blocks are allocated to the file in such a way that all the logical blocks of the file get the contiguous physical block in the hard disk then such allocation scheme is known as contiguous allocation.
- In the image shown below, there are three files in the directory.
- The starting block and the length of each file are mentioned in the table. We can check in the table that the contiguous blocks are assigned to each file as per its need.



Hard Disk

File Name	Start	Length	Allocated Blocks
abc.text	0	3	0,1,2
video.mp4	4	2	4,5
jtp.docx	9	3	9,10,11

Directory

- All these algorithms suffer from the problem of external fragmentation.
- As files are allocated and deleted, the free disk space is broken into little pieces. External fragmentation exists whenever free space is broken into chunks.
- It becomes a problem when the largest contiguous chunk is insufficient for a request; storage is fragmented into a number of holes, none of which is large enough to store the data.
- This scheme effectively **compacts** all free space into one contiguous space, solving the fragmentation problem.

Advantages

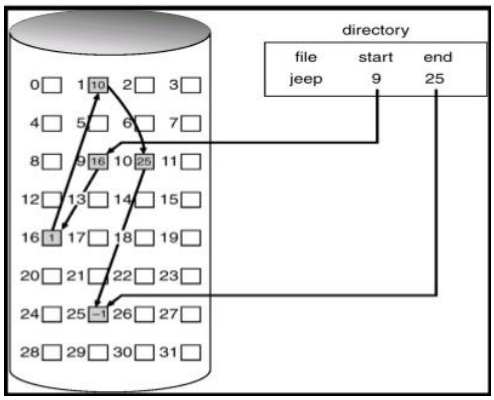
- It is simple to implement.
- We will get Excellent read performance.
- Supports Random Access into files.

Disadvantages

- The disk will become fragmented.
- It may be difficult to have a file grow.

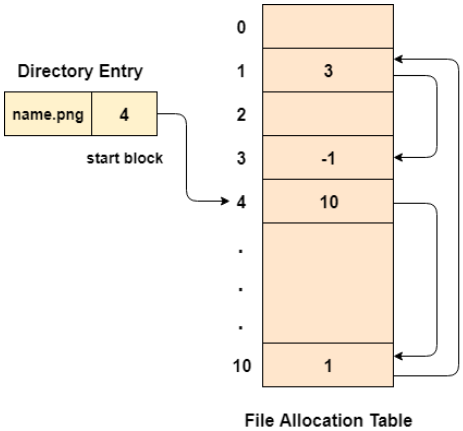
Linked List Allocation

- Linked List allocation solves all problems of contiguous allocation.
- In linked list allocation, each file is considered as the linked list of disk blocks.
- However, the disks blocks allocated to a particular file need not to be contiguous on the disk.
- Each disk block allocated to a file contains a pointer which points to the next disk block allocated to the same file.
- For example, a file of five blocks might start at block 9 and continue at block 16, then block 1, then block 10, and finally block 25 (See Figure). Each block contains a pointer to the next block. These pointers are not made available to the user. Thus, if each block is 512 bytes in size, and a disk address (the pointer) requires 4 bytes, then the user sees blocks of 508 bytes.



File Allocation Table

- The main disadvantage of linked list allocation is that the Random access to a particular block is not provided. In order to access a block, we need to access all its previous blocks.
- File Allocation Table overcomes this drawback of linked list allocation. In this scheme, a file allocation table is maintained, which gathers all the disk block links. The table has one entry for each disk block and is indexed by block number.
- File allocation table needs to be cached in order to reduce the number of head seeks. Now the head doesn't need to traverse all the disk blocks in order to access one successive block.



File Allocation Table

Advantages

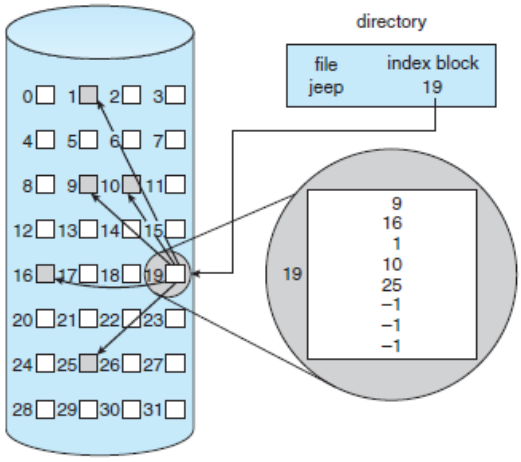
- There is no external fragmentation with linked allocation.
- Any free block can be utilized in order to satisfy the file block requests.
- File can continue to grow as long as the free blocks are available.
- Directory entry will only contain the starting block address.

Disadvantages

- Random Access is not provided.
- Pointers require some space in the disk blocks.
- Any of the pointers in the linked list must not be broken otherwise the file will get corrupted.
- Need to traverse each block.

Indexed Allocation

- Indexed allocation solves this problem by bringing all the pointers together into one location: **the index block.**
- Each file has its own index block, which is an array of disk-block addresses.
- The ith entry in the index block points to the ith block of the file. The directory contains the address of the index block.
- To find and read the *ith* block, we use the pointer in the *ith* index-block entry. This scheme is similar to the paging scheme.
- When the file is created, all pointers in the index block are set to null.
- When the *ith* block is first written, a block is obtained from the free-space manager, and its address is put in the *i*th index-block entry.
- Indexed allocation supports direct access, without suffering from external fragmentation, because any free block on the disk can satisfy a request for more space.



Advantages

1. Supports direct access
2. A bad data block causes the lost of only that block.

Disadvantages

1. A bad index block could cause the lost of entire file.

2. Size of a file depends upon the number of pointers, a index block can hold.
3. Having an index block for a small file is totally wastage.
4. More pointer overhead

4.Free Space Management

Since disk space is limited, we need to reuse the space from deleted files for new files, if possible. To keep track of free disk space, the system maintains a free-space list. The free-space list records all free disk blocks – those not allocated to some file or directory.

To create a file, we search the free-space list for the required amount of space, and allocate that space to the new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

1. Bit Vector

The free-space list is implemented as a bit map or bit vector. Each block is represented by 1 bit.

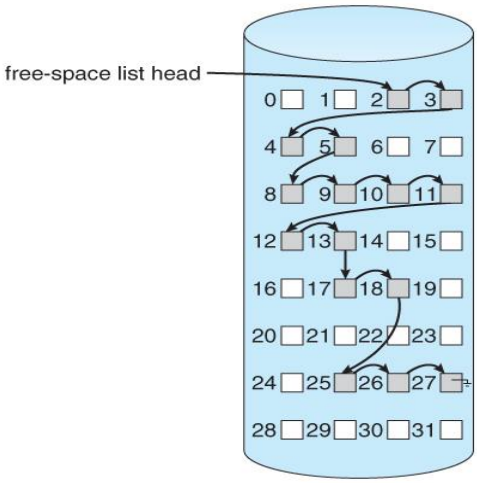
If the block is free, the bit is 1; if the block is allocated, the bit is 0. For example, Consider a disk where block 2,3,4,5,8,9,10,11,12,13,17,18,25,26 and 27 are free, and the rest of the block are allocated. The free space bit map would be 00111100111111000110000001110000 0 …

The main advantage **of** this approach is its relatively simplicity and efficiency in finding the first free block, or n consecutive free blocks on the disk.

2. Linked List

Another approach to free-space management is to link together all the free disk blocks, keeping a pointer to the first free block in a special location on the disk and caching it in memory. This first block contains a pointer to the next free disk block, and so on.

In our example, we would keep a pointer to block 2, as the first free block. Block 2 would contain a pointer to block 3, which would point to block 4, which would point to block 5, which would point to block 8, and so on. However, this scheme is not efficient; to traverse the list, we must read each block, which requires substantial I/O time. The FAT method incorporates free-block accounting data structure. No separate method is needed.

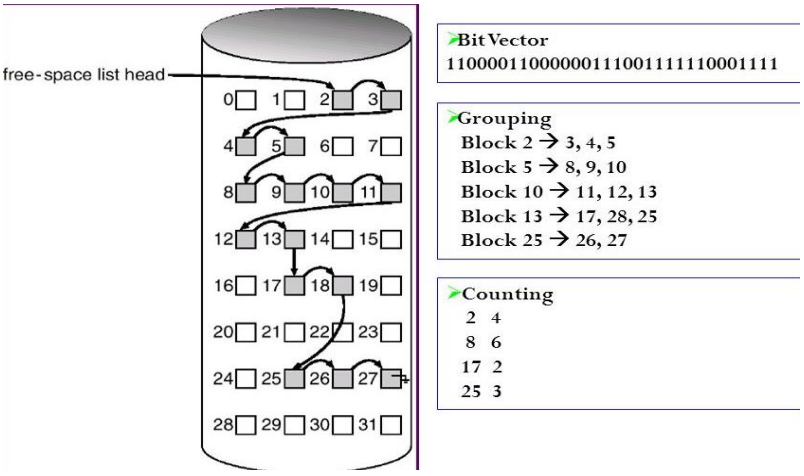


3. Grouping

A modification of the free-list approach is to store the addresses of n free blocks in the first free block. The first n-1 of these blocks are actually free. The last block contains the addresses of another n free blocks, and so on. The importance of this implementation is that the addresses of a large number of free blocks can be found quickly.

4. Counting

We can keep the address of the first free block and the number n of free contiguous blocks that follow the first block. Each entry in the free-space list then consists of a disk address and a count. Although each entry requires more space than would a simple disk address, the overall list will be shorter, as long as the count is generally greater than1.



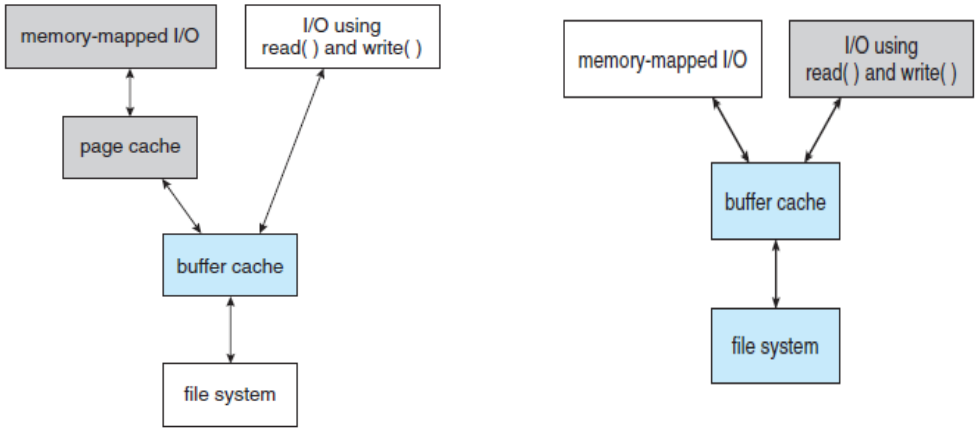
5. Efficiency and Performance

Efficiency

- The efficient use of disk space depends heavily on the disk-allocation and directory algorithms in use.
- Let's reconsider the clustering scheme, which improves file-seek and file-transfer performance at the cost of internal fragmentation. To reduce this fragmentation, BSD UNIX varies the cluster size as a file grows. Large clusters are used where they can be filled, and small clusters are used for small files and the last cluster of a file. This
- The types of data normally kept in a file's directory (or inode) entry also require consideration. Commonly, a "last write date" is recorded to supply information to the user and to determine whether the file needs to be backed up. Some systems also keep a "last access date," so that a user can determine when the file was last read.
- The result of keeping this information is that, whenever the file is read, a field in the directory structure must be written to. That means the block must be read into memory, a section changed, and the block written back out to disk, because operations on disks occur only in block (or cluster) chunks. So any time a file is opened for reading, its directory entry must be read and written as well.
- Generally, every data item associated with a file needs to be considered for its effect on efficiency and performance.

Performance

- Some systems maintain a separate section of main memory for a **buffer cache**, where blocks are kept under the assumption that they will be used again shortly. Other systems cache file data using a **page cache**.
- The **page cache** uses virtual memory techniques to cache file data as pages rather than as file-system-oriented blocks.
- Caching file data using virtual addresses is far more efficient than caching through physical disk blocks, as accesses interface with virtual memory rather than the file system.
- Several systems—including Solaris, Linux, and Windows —use page caching to cache both process pages and file data. This is known as **unified virtual memory**.



- The two alternatives for opening and accessing a file. One approach is to use memory mapping the second is to use the standard system calls **read()** and **write()**.
- Here, the read() and write() system calls go through the buffer cache.
- The memory-mapping call, however, requires using two caches—the **page cache** and the **buffer cache.**
- A memory mapping proceeds by reading in disk blocks from the file system and storing them in the buffer cache. Because the virtual memory system does not interface with the buffer cache, the contents of the file in the buffer cache must be copied into the page cache. This situation, known as **double caching**, requires caching file-system data twice.

6. Recovery

Consistency Checking

- The storing of certain data structures (e.g. directories and inodes) in memory and the caching of disk operations can speed up performance, but what happens in the result of a system crash? All volatile memory structures are lost, and the information stored on the hard drive may be left in an inconsistent state.
- A Consistency Checker (fsck in UNIX, chkdsk or scandisk in Windows) is often run at boot time or mount time, particularly if a filesystem was not closed down properly. Some of the problems that these tools look for include:
  - Disk blocks allocated to files and also listed on the free list.
  - Disk blocks neither allocated to files nor on the free list.
  - Disk blocks allocated to more than one file.
  - The number of disk blocks allocated to a file inconsistent with the file's stated size.
  - Properly allocated files / inodes which do not appear in any directory entry.
  - Link counts for an inode not matching the number of references to that inode in the directory structure.
  - Two or more identical file names in the same directory.
  - Illegally linked directories, e.g. cyclical relationships where those are not allowed, or files/directories that are not accessible from the root of the directory tree.
  - Consistency checkers will often collect questionable disk blocks into new files with names such as chk00001.dat. These files may contain valuable information that would otherwise be lost, but in most cases they can be safely deleted, (returning those disk blocks to the free list.)

UNIX caches directory information for reads, but any changes that affect space allocation or metadata changes are written synchronously, before any of the corresponding data blocks are written to.

Log-Structured File Systems

- Log-based transaction-oriented (a.k.a. journaling) filesystems borrow techniques developed for databases, guaranteeing that any given transaction either completes successfully or can be rolled back to a safe state before the transaction commenced:

- All metadata changes are written sequentially to a log.
- A set of changes for performing a specific task (e.g. moving a file) is a transaction.
- As changes are written to the log they are said to be committed, allowing the system to return to its work.
- In the meantime, the changes from the log are carried out on the actual filesystem, and a pointer keeps track of which changes in the log have been completed and which have not yet been completed.
- When all changes corresponding to a particular transaction have been completed, that transaction can be safely removed from the log.
- At any given time, the log will contain information pertaining to uncompleted transactions only, e.g. actions that were committed but for which the entire transaction has not yet been completed.
- From the log, the remaining transactions can be completed, or if the transaction was aborted, then the partially completed changes can be undone.

Backup and Restore

- In order to recover lost data in the event of a disk crash, it is important to conduct backups regularly.
- Files should be copied to some removable medium, such as magnetic tapes, CDs, DVDs, or external removable hard drives.
- A full backup copies every file on a file system.Incremental backups copy only files which have changed since some previous time.
- A combination of full and incremental backups can offer a compromise between full recoverability, the number and size of backup tapes needed, and the number of tapes that need to be used to do a full restore.
- A typical backup schedule may then be as follows:

Day 1. Copy to a backup medium all files from the disk. This is called a full backup.

Day 2. Copy to another medium all files changed since day 1. This is an incremental backup.

Day 3. Copy to another medium all files changed since day 2.

Day N. Copy to another medium all files changed since day N−1. Then go back to day 1.

I/O SYSTEMS

1. I/O Hardware

The role of the operating system in computer I/O is to manage and control I/O operations and I/O devices. A device communicates with a computer system by sending signals over a cable or even through the air.

Port: The device communicates with the machine via a connection point (or port), for example, a serial port.

Bus: If one or more devices use a common set of wires, the connection is called a bus.

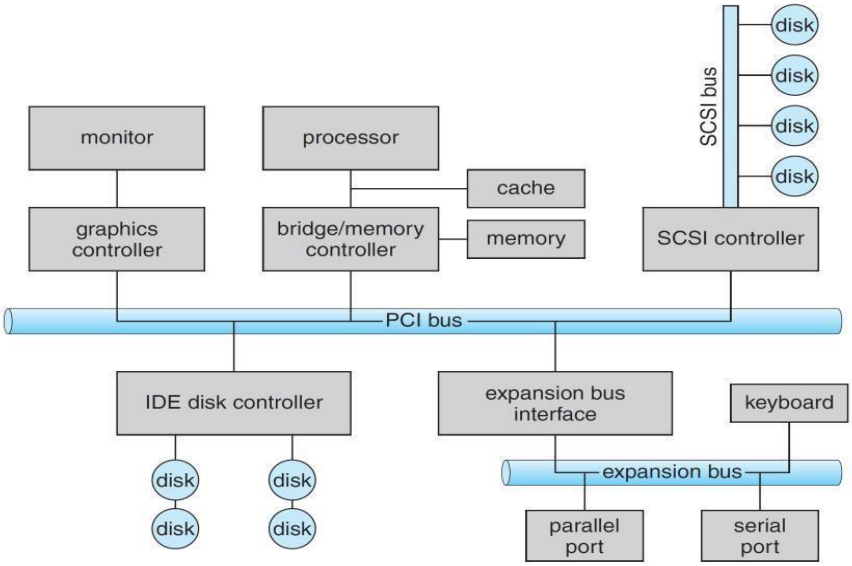
Daisy chain: Device _A 'has a cable that plugs into device _B ', and device _B 'has a cable that plugs into device _C ', and device _C 'plugs into a port on the computer, this arrangement is called a daisy chain. A daisy chain usually operates as a bus.

PC bus structure

A PCI bus that connects the processor-memory subsystem to the fast devices, and an expansion bus that connects relatively slow devices such as the keyboard and serial and parallel ports. In the upper- right portion of the figure, four disks are connected together on a SCSI bus plugged into a SCSI controller.

A **controller or host adapter** is a collection of electronics that can operate a port, a bus, or a device. A serial-port controller is a simple device controller. It is a single chip in the computer that controls the signals on the wires of a serial port. By contrast, a SCSI bus controller is not simple.

Because the SCSI protocol is complex, the SCSI bus controller is often implemented as a separate circuit board. It typically contains a processor, microcode, and some private memory. Some devices have their own built- in controllers.



How can the processor give commands and data to a controller to accomplish an I/O transfer?

- Direct I/O instructions
- Memory-mapped I/O

Direct I/O instructions

Use special I/O instructions that specify the transfer of a byte or word to an I/O port address. The I/O instruction triggers bus lines to select the proper device and to move bits into or out of a device register

Memory-mapped I/O

The device-control registers are mapped into the address space of the processor. The CPU executes I/O requests using the standard data-transfer instructions to read and write the device- control registers.

Status register	Read by the host to indicate states such as whether the current command has completed, whether a byte is available to be read from the data-in register, and whether there has been a device error.
Control register	Written by the host to start a command or to change the mode of a device.
data-in register	Read by the host to get input
data-out register	Written by the host to send output

- An I/O port typically consists of four registers: status, control, data-in, and data-out registers.

1. Polling

Interaction between the host and a controller

- The controller sets the busy bit when it is busy working, and clears the busy bit when it is ready to accept the next command.
- The host sets the command ready bit when a command is available for the controller to execute.

Coordination between the host & the controller is done by handshaking as follows:

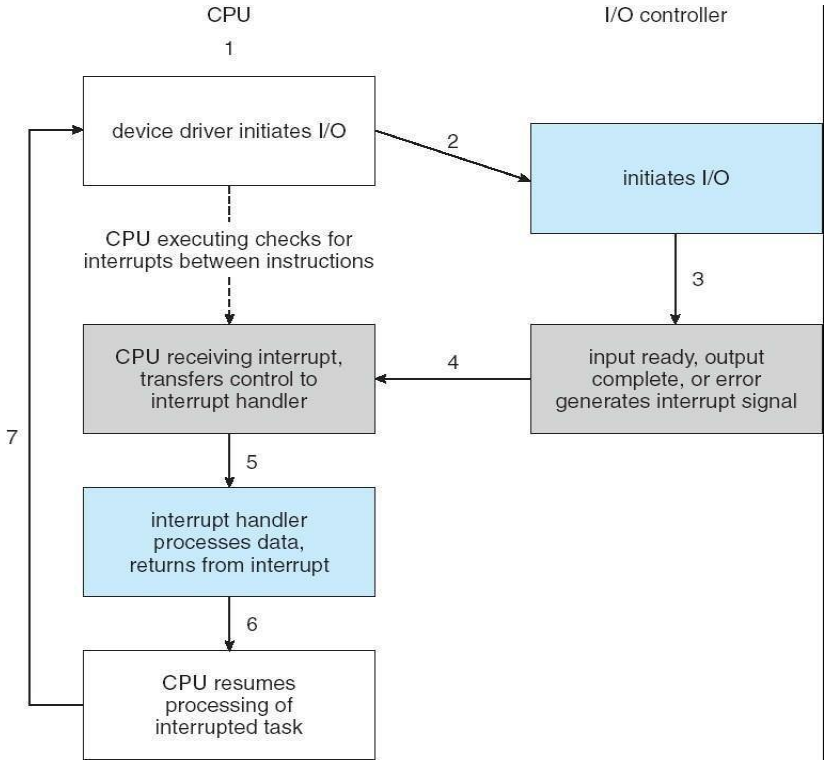
1. The host repeatedly reads the busy bit until that bit becomes clear.
2. The host sets the write bit in the command register and writes a byte into the data-out register.
3. The host sets the command-ready bit.
4. When the controller notices that the command-ready bit is set, it sets the busy bit.
5. The controller reads the command register and sees the write command. It reads the data-out register to get the byte, and does the I/O to the device.
6. The controller clears the command-ready bit, clears the error bit in the status register to indicate that the device I/O succeeded, and clears the busy bit to indicate that it is finished.
7. In step 1, the host is ―**busy-waiting or polling**‖: It is in a loop, reading the status register over and over until the busy bit becomes clear.

2. Interrupts

The CPU hardware has a wire called the ―interrupt-request line‖.
The basic interrupt mechanism works as follows;

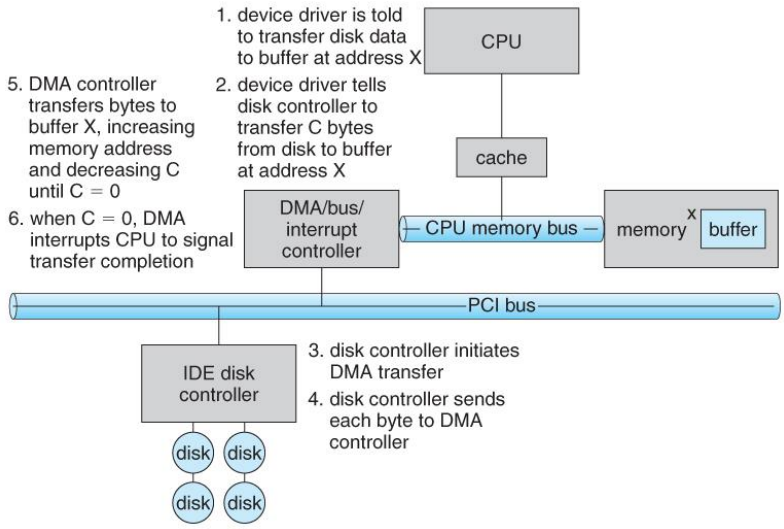
1. Device controller raises an interrupt by asserting a signal on the interrupt request line.
2. The CPU catches the interrupt and dispatches to the interrupt handler and
3. The handler clears the interrupt by servicing the device.

- **Nonmaskable interrupt**: which is reserved for events such as unrecoverable memory errors?
- **Maskable interrupt**: Used by device controllers to request service



3. Direct Memory Access (DMA)

In general it is tough for the CPU to do the large transfers between the memory buffer & disk; because it is already equipped with some other tasks ,then this will create overhead. So a special-purpose processor called a direct memory-access **(DMA)** controller is used.



2. Application I/O Interface

I/O system calls encapsulate device behaviours in generic classes. Device-driver layer hides differences among I/O controllers from kernel

Devices vary on many dimensions, as illustrated in

• **Character-stream or block**. A character-stream device transfers bytes one by one, whereas a block device transfers a block of bytes as a unit.

• **Sequential or random access**. A sequential device transfers data in a fixed order determined by the device, whereas the user of a random-access device can instruct the device to seek to any of the available data storage locations.

• **Synchronous or asynchronous**. A synchronous device performs data transfers with predictable response times, in coordination with other aspects of the system. An asynchronous device exhibits irregular or unpredictable response times not coordinated with other computer events.

• **Sharable or dedicated**. A sharable device can be used concurrently by several processes or threads; a dedicated device cannot.

Speed of operation. Device speeds range from a few bytes per second to a few gigabytes per second.

• **Read–write, read only, or write only**. Some devices perform both input and output, but others support only one data transfer direction.

1. Block and Character Devices

Block-device: The block-device interface captures all the aspects necessary for accessing disk drives and other block-oriented devices. The device should understand the commands such as read () & write (), and if it is a random access device, it has a seek() command to specify which block to transfer next.

Character Devices: A keyboard is an example of a device that is accessed through a character stream interface. The basic system calls in this interface enable an application to get() or put() one character.

2. Network Devices

Because the performance and addressing characteristics of network I/O differ

significantly from those of disk I/O, most operating systems provide a network I/O interface that is different from the read0
-write() -seek() interface used for disks.

- Windows NT provides one interface to the network interface card, and a second interface to the network protocols.

- In UNIX, we find half-duplex pipes, full-duplex FIFOs, full-duplex STREAMS, message queues and sockets.

3. Clocks and Timers

Most computers have hardware clocks and timers that provide three basic functions:

- Give the current time
- Give the elapsed time
- Set a timer to trigger operation X at time T

Programmable interval timer: The hardware to measure elapsed time and to trigger operations is called a programmable interval timer. It can be set to wait a certain amount of time and then to generate an interrupt. To generate periodic interrupts, it can be set to do this operation once or to repeat.

Uses of Programmable interval timer:

Scheduler	To generate an interrupt that will pre-empt a process at the end of its time slice.
Disk I/O subsystem	To invoke the flushing of dirty cache buffers to disk periodically
Network subsystem	To cancel operations those are proceeding too slowly because of network congestion or failures.

When the timer interrupts, the kernel signals the requester, and reloads the timer with the next earliest time.

Counter: The hardware clock is constructed from a high frequency counter.

In some computers, the value of this counter can be read from a device register, in which the counter can be considered to be a high-resolution clock.

4. Blocking and Non-blocking I/O (or) synchronous & asynchronous: Blocking I/O: When an application issues a blocking system call;

- The execution of the application is suspended.
- The application is moved from the operating system's run queue to a wait queue.
- After the system call completes, the application is moved back to the run queue, where it is eligible to resume execution, at which time it will receive the values returned by the system call.

Non-blocking, I/O: Some user-level processes need non-blocking

I/O***Examples:***

User interface that receives keyboard and mouse input while processing and displaying data on the screen.

Video application that reads frames from a file on disk while simultaneously decompressing and displaying the output on the display.

3.Kernel I/O Subsystem

Kernels provide many services related to I/O.

- One way that the I/O subsystem improves the efficiency of the computer is by scheduling I/O operations.
- Another way is by using storage space in main memory or on disk, via techniques called buffering, caching, and spooling.

1. I/O Scheduling:

To determine a good order in which to execute the set of I/O requests. Uses:

- It can improve overall system performance,
- It can share device access fairly among processes, and
- It can reduce the average waiting time for 1/0 to complete.

Implementation: OS developers implement scheduling by maintaining a ―queue of requests for each device.

- When an application issues a blocking I/O system call,
- The request is placed on the queue for that device.
- The I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by applications.

2. Buffering:

Buffer: A memory area that stores data while they are transferred between two devices or between a device and an application.

Reasons for buffering:

- To cope with a speed mismatch between the producer and consumer of a data stream.
- To adapt between devices that have different data-transfer sizes.
- To support copy semantics for application I/O.

Copy semantics Suppose that an application has a buffer of data that it wishes to write to disk. It calls the write () system call, providing a pointer to the buffer and an integer specifying the number of bytes to write.

3. Caching

A cache is a region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original

Cache vs buffer: A buffer may hold the only existing copy of a data item, whereas a cache just holds a copy on faster storage of an item that resides elsewhere.

When the kernel receives a file I/O request,

1. The kernel first accesses the buffer cache to see whether that region of the file is already available in main memory.

2. If so, a physical disk I/O can be avoided or deferred. Also, disk writes are accumulated in the buffer cache for several seconds, so that large transfers are gathered to allow efficient write schedules.

4. Spooling and Device Reservation:

Spool: A buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams.

A printer can serve only one job at a time, several applications may wish to print their output concurrently, without having their output mixed together

The OS provides a control interface that enables users and system administrators ;

- To display the queue,
- To remove unwanted jobs before those jobs print,
- To suspend printing while the printer is serviced, and so on.

Device reservation - provides exclusive access to a device

- System calls for allocation and de-allocation
- Watch out for deadlock

5. Error Handling

An operating system that uses protected memory can guard against many kinds of hardware and application errors. OS can recover from disk read, device unavailable, transient write failures Most return an error number or code when I/O request fails System error logs hold problem reports

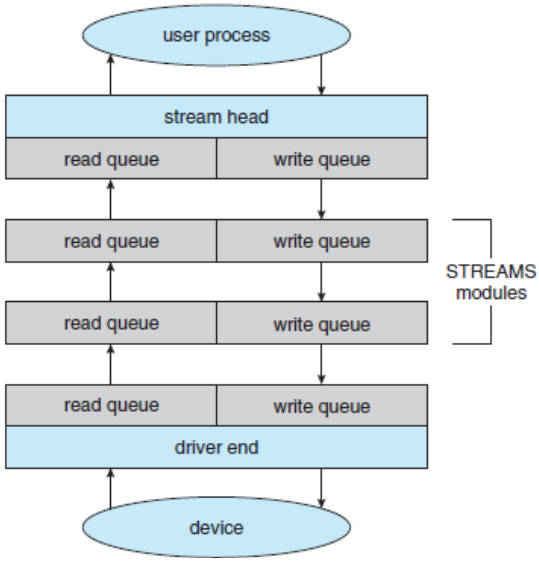
STREAMS

Stream is a full-duplex communication channel between a user-level process and a device in Unix System V and beyond A STREAM consists of:

- STREAM head interfaces with the user process
- Driver end interfaces with the device
- Zero or more STREAM modules between them.

Each module contains a read queue and a write queue. Message passing is used to communicate between queues. Modules provide the functionality of STREAMS processing and they are pushed onto a stream using the ioct () system call.

Flow control: Because messages are exchanged between queues in adjacent modules, a queue in one module may overflow an adjacent queue. To prevent this from occurring, a queue may support flow control.



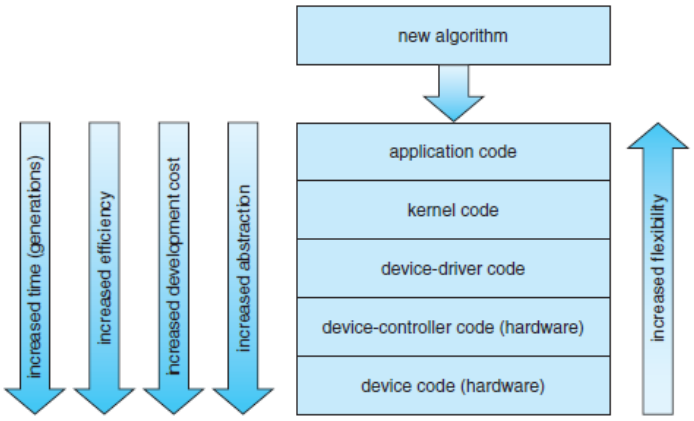
PERFORMANCE

I/O a major factor in system performance:

- Heavy demands on CPU to execute device driver, kernel I/O code. So context switches occur due to interrupts.
- Interrupt handling is a relatively expensive task: Each interrupt causes the system to perform a state change, to execute the interrupt handler & then to restore state
- Network traffic especially stressful.
- Systems use separate ―front-end processors" for terminal I/O, to reduce the interrupt burden on the main CPU.

We can employ several principles to improve the efficiency of I/O:

- Reduce the number of context switches.
- Reduce the number of times that data must be copied in memory while passing between device and application.
- Reduce the frequency of interrupts by using large transfers, smart controllers & polling.
- Increase concurrency by using DMA-knowledgeable controllers or channels to offload simple data copying from the CPU.
- Move processing primitives into hardware, to allow their operation in device controllers concurrent with the CPU and bus operation.
- Balance CPU, memory subsystem, bus, and I/O performance, because an overload in any one area will cause idleness in others.



Device functionality progression.

a) **An application-level implementation**: Implement experimental I/O algorithms at the application level, because application code is flexible, and application bugs are unlikely to cause system crashes.

It can be inefficient;

- Because of the overhead of context switches and
- Because the application cannot take advantage of internal kernel data structures and kernel functionality

b) **In-kernel implementation**: Re-implement application-level algorithm in the kernel. This can improve the performance, but the development effort is more challenging, because an operating-system kernel is a large, complex software system. Moreover, an in-kernel implementation must be thoroughly debugged to avoid data corruption and system crashes.

c) **A hardware implementation**: The highest performance may be obtained by a specialized implementation in hardware, either in the device or in the controller.

- ❖ Difficult and expense of making further improvements or of fixing bugs, (-) Increased development time
- ❖ Decreased flexibility.

UNIT V CASE STUDY

Linux System - Design Principles, Kernel Modules, Process Management, Scheduling, Memory Management, Input-Output Management, File System, Inter-process Communication; Mobile OS - iOS and Android - Architecture and SDK Framework, Media Layer, Services Layer, Core OS Layer, File System.

1. LINUX SYSTEM

1.1 Linux History

- Its development began in 1991, when a Finnish university student, Linus Torvalds, began developing a small but self-contained kernel for the 80386 processor, the first true 32-bit processor in Intel's range of PC-compatible CPUs.
- Early in its development, the Linux source code was made available free— both at no cost and with minimal distributional restrictions—on the Internet.
- The **Linux kernel** is an original piece of software developed from scratch by the Linux community.
- The **Linux system**, includes a multitude of components, some written from scratch, others borrowed from other development projects, and still others created in collaboration with other teams.
- A **Linux distribution** includes all the standard components of the Linux system, plus a set of administrative tools to simplify the initial installation and subsequent upgrading of Linux and to manage installation and removal of other packages on the system.

1.2The Linux Kernel

- The first Linux kernel released to the public was version 0.01, dated May 14, 1991. It had no networking, ran only on 80386-compatible Intel processors and PC hardware, and had extremely limited device-driver support.
- The next milestone, **Linux 1.0**, was released on March 14, 1994.
- This release culminated three years of rapid development of the Linux kernel. Perhaps the single biggest new feature was networking: 1.0 included support for UNIX's standard TCP/IP networking protocols such as socket interface for networking programming.
- In **March 1995**, the **1.2 kernel** was released. This release did not offer nearly the same improvement in functionality as the 1.0 release, but it did support a much wider variety of hardware, including the new PCI hardware bus architecture.
- In **June 1996** as Linux **version 2.0**was released. This release was given a major version-number increment because of two major new capabilities: support for multiple architectures, including a 64-bit native Alpha port, and symmetric multiprocessing (SMP) support
- Improvements continued with the release of **Linux 2.2** in **1999**. A port to UltraSPARC systems was added. Networking was enhanced with more

flexible firewalling, improved routing and traffic management, and support for TCP large window and selective acknowledgement.

- Linux kernel version 3.0 was released in July 2011.

2. DESIGN PRINCIPLES

- Linux runs on a wide variety of platforms, it was originally developed exclusively on PC architecture.
- Linux can run happily on a multiprocessor machine with many gigabytes of main memory and many terabytes of disk space, but it is still capable of operating usefully in under 16 MB of RAM.

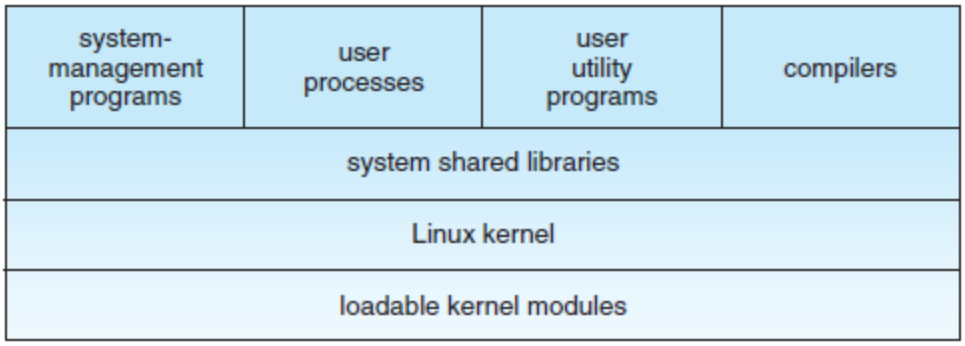
➔Components of a Linux System

The Linux system is composed of three main bodies of code

1. **Kernel**. The kernel is responsible for maintaining all the important abstractions of the operating system, including such things as virtual memory and processes.

2. **System libraries**. The system libraries define a standard set of functions through which applications can interact with the kernel. These functions implement much of the operating-system functionality that does not need the full privileges of kernel code. The most important system library is the C library, known as libc. In addition to providing the standard C library, libc implements the user mode side of the Linux system call interface, as well as other critical system-level interfaces.

3. **System utilities**. The system utilities are programs that perform individual, specialized management tasks. Some system utilities are invoked just once to initialize and configure some aspect of the system. Others —known as daemons in UNIX terminology—run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals, and updating log files.



- All the kernel code executes in the processor's privileged mode with full access to all the physical resources of the computer.
- Linux refers to this **privileged mode** as kernel mode.
- Under Linux, no user code is built into the kernel.
- Any operating-system-support code that does not need to run in **kernel mode** is placed into the system libraries and runs in user **mode**.
- Unlike **kernel mode**, **user mode** has access only to a controlled subset of the system's resources.

3. KERNEL MODULES

- The Linux kernel has the ability to load and unload arbitrary sections of kernel code on demand.
- These loadable kernel modules run in privileged kernel mode and as a consequence have full access to all the hardware capabilities of the machine on which they run.
- Kernel modules are convenient for several reasons.
  1. **Linux's source code is free**, so anybody wanting to write kernel code is able to compile a modified kernel and to reboot into that new functionality.
  2. However, **recompiling, relinking, and reloading** the entire kernel is a cumbersome cycle to undertake when you are developing a new driver.
  3. If you use kernel modules, you **do not have to make a new kernel** to test a new driver—the driver can be compiled on its own and loaded into the already running kernel.
- Kernel modules allow a Linux system to be set up with a standard minimal kernel, without any extra device drivers built in.
- Any device drivers that the user needs can be either loaded explicitly by the system at startup or loaded automatically by the system on demand and unloaded when not in use.
- For example, a mouse driver can be loaded when a USB mouse is plugged into the system and unloaded when the mouse is unplugged.

1. **The module-management** system allows modules to be loaded into memory and to communicate with the rest of the kernel.
2. **The module loader and unloader**, which are user-mode utilities, work with the module-management system to load a module into memory.
3. **The driver-registration** system allows modules to tell the rest of the kernel that a new driver has become available.
4. **A conflict-resolution mechanism** allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

Module Management

- Loading a module requires more than just loading its binary contents into kernel memory.
- Linux maintains an internal symbol table in the kernel.
- The loading of the module is performed in **two stages**.

  ➔First, **the module loader utility** asks the kernel to reserve a continuous area of virtual kernel memory for the module. The kernel returns the address of the memory allocated, and the loader utility can use this address to relocate the module's machine code to the correct loading address.

  ➔A **second system call then passes** the module, plus any symbol table that the new module wants to export, to the kernel.

Driver Registration

- provides a set of routines to allow drivers to be added to or removed.

- A module may register many types of functionality
- For example, a device driver might want to register two separate mechanisms for accessing the device. Registration tables include, among others, the following items:

  Device drivers. These drivers include character devices (such as printers, terminals, and mice), block devices (including all disk drives), and network interface devices.

  File systems. The file system may be anything that implements Linux's virtual file system calling routines. It might implement a format for storing files on a disk, but it might equally well be a network file system, such as NFS, or a virtual file system whose contents are generated on demand, such as Linux's /proc file system.

  Network protocols. A module may implement an entire networking protocol, such as TCP or simply a new set of packet-filtering rules for a network firewall.

  Binary format. This format specifies a way of recognizing, loading, and executing a new type of executable file.

Conflict Resolution

Linux provides a central conflict-resolution mechanism to help arbitrate access to certain hardware resources. Its aims are as follows:

➔ To prevent modules from clashing over access to hardware resources

➔ To prevent autoprobes—device-driver probes that auto-detect device configuration—from interfering with existing device drivers

➔ To resolve conflicts among multiple drivers trying to access the same hardware—as, for example, when both the parallel printer driver and the parallel line IP (PLIP) network driver try to talk to the parallel port

4. PROCESS MANAGEMENT

A process is the basic context in which all user-requested activity is serviced within the operating system.

➔The fork() and exec() Process Model

- The basic principle of UNIX process management is to separate into two steps two operations that are usually combined into one: The **creation of a new process** and the **running of a new program**.
- A new process is created by the fork() system call, and a new program is run after a call to exec().
- These are two distinctly separate functions.
- We can create a new process with fork() without running a new program—the new subprocess simply continues to execute exactly the same program, at exactly the same point, that the first (parent) process was running.

1.Process Identity

➔ A process identity consists mainly of the following items:

Process ID (PID). Each process has a unique identifier.

Credentials. Each process must have an associated user ID and one or more group IDs that determine the rights of a process to access system resources and files.

Personality: Personalities are primarily used by emulation libraries to request the system calls be compatible with certain varieties of UNIX.

Namespace: Each process is associated with a specific view of the file system hierarchy, called its namespace. Most processes share a common namespace and thus operate on a shared file-system hierarchy.

➔ Process Environment

A process's environment is inherited from its parent and is composed of **two null-terminated vectors**: **the argument vector** and the **environment vector**.

The argument vector simply lists the command-line arguments used to invoke the running program; it conventionally starts with the name of the program itself.

The environment vector is a list of "NAME=VALUE" pairs that associates named environment variables with arbitrary textual values. The environment is not held in kernel memory but is stored in the process's own user-mode address space as the first datum at the top of the process's stack.

➔ Process Context

- Process context is the state of the running program at any one time; it changes constantly. Process context includes the following parts:

  Scheduling context: The most important part of the process context is its scheduling context—the information that the scheduler needs to suspend and restart the process. This information includes saved copies of all the process's registers.

  Accounting: The kernel maintains accounting information about the resources currently being consumed by each process and the total resources consumed by the process in its entire lifetime so far.

  File table. The file table is an array of pointers to kernel file structures representing open files.

  File-system context : Whereas the file table lists the existing open files, the file-system context applies to requests to open new files. The file-system context includes the process's root directory, current working directory, and namespace.

  Signal-handler table: The signal-handler table defines the action to take in response to a specific signal.

  Virtual memory context : The virtual memory context describes the full contents of a process's private address space.

2. Processes and Threads

- Linux provides the fork() system call, which duplicates a process without loading a new executable image. Linux also provides the ability to create threads via the clone() system call.
- The clone() system call behaves identically to fork(), except that it accepts as arguments a set of flags that dictate what resources are shared between the parent and child.
- The flags include

flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

5. SCHEDULING

- Scheduling is the job of allocating CPU time to different tasks within an operating system.
- Linux, like all UNIX systems, supports preemptive multitasking.
- In such a system, the process scheduler decides which process runs and when.

➔Process Scheduling

Linux has two separate process-scheduling algorithms.

1. One is a time-sharing algorithm for fair, preemptive scheduling among multiple processes.
2. The other is designed for real-time tasks, where absolute priorities are more important than fairness.

Completely Fair Scheduler (CFS).

- In CFS each core of the CPU has its own run queue.
- Each task has a so called nice value and weight assigned to it. The nice value represents how "kind" the specific task is to other tasks.
- In other words, a task with a high nice value has a lower priority and is thus less likely to take more of the CPUs bandwidth than a task with a low nice value.
- CFS introduced a new scheduling algorithm called **fair scheduling** that eliminates **time slices** in the traditional sense. **Instead of time slices**, all processes are **allotted a proportion of the processor's time**. CFS calculates how long a process should run as a function of the total number of runnable processes.
- To calculate the actual length of time a process runs, CFS relies on a configurable variable called **target latency**, which is the interval of time during which every runnable task should run at least once.

➔Real-Time Scheduling

- Linux implements the two real-time scheduling classes: first-come, first served (FCFS) and round-robin.
- In both cases, each process has a priority in addition to its scheduling class.
- The scheduler always runs the process with the highest priority. Among processes of equal priority, it runs the process that has been waiting longest.
- The only difference between FCFS and round-robin scheduling is that FCFS processes continue to run until they either exit or block, whereas a round-robin process will be preempted after a while and will be moved to the end of the scheduling queue, so round-robin processes of equal priority will automatically time-share among themselves.
- Linux's real-time scheduling is **soft— and hard—real time**.

  ➔ A hard real time system guarantees that critical tasks complete on time, whereas in soft real time system, a critical real time task gets priority over other tasks and retains that priority until it completes.

➔ Kernel Synchronization

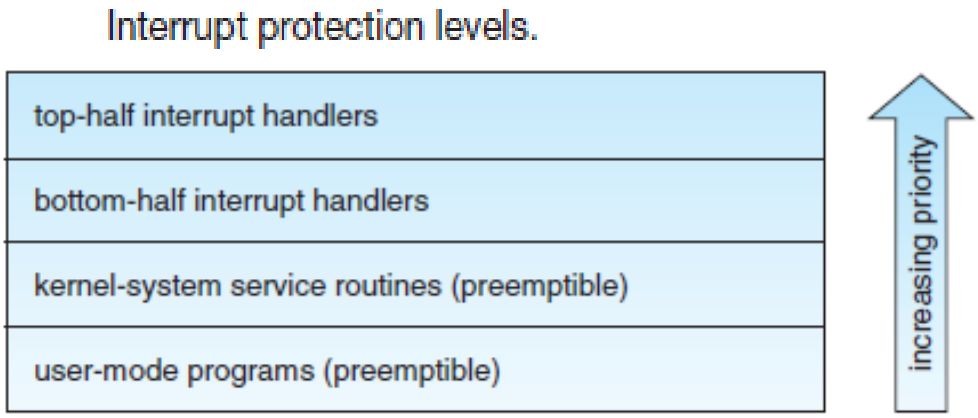
- The way the kernel schedules its own operations is fundamentally different from the way it schedules processes.
- A request for kernel-mode execution can occur in two ways.
- A running program may request an operating-system service, either explicitly via a system call or implicitly—for example, when a page fault occurs.
- Alternatively, a device controller may deliver a hardware interrupt that causes the CPU to start executing a kernel-defined handler for that interrupt.
- The problem for the kernel is that all these tasks may try to access the same internal data structures.
- If one kernel task is in the middle of accessing some data structure when an interrupt service routine executes, then that service routine cannot access or modify the same data without risking data corruption.
- The Linux kernel provides spinlocks and semaphores (as well as reader–writer versions of these two locks) for locking in the kernel.
- Linux uses an interesting approach to disable and enable kernel preemption. It provides two simple kernel interfaces—**preempt disable()** and **preempt enable().**

single processor	multiple processors
Disable kernel preemption.	Acquire spin lock.
Enable kernel preemption.	Release spin lock.

- The counter is incremented when a lock is acquired and decremented when a lock is released.
- Linux implements this architecture by separating interrupt service routines into two sections: **the top half** and the **bottom half**.

  The **top half** is the standard interrupt service routine that runs with recursive interrupts disabled.

Interrupts of the same number (or line) are disabled, but other interrupts may run.
The **bottom half** of a service routine is run, with all interrupts enabled, by a miniature scheduler that ensures that bottom halves never interrupt themselves.



➔ Symmetric Multiprocessing

Linux kernel to support symmetric multiprocessor (SMP) hardware, allowing separate processes to execute in parallel on separate processors. The original implementation of SMP imposed the restriction that only one processor at a time could be executing kernel code.

6.MEMORY MANAGEMENT

Memory management under Linux has two components. The first deals with allocating and freeing physical memory—pages, groups of pages, and small blocks of RAM. The second handles virtual memory, which is memory-mapped into the address space of running processes.

➔Management of Physical Memory

- Due to specific hardware constraints, Linux separates physical memory into four different zones, or regions:

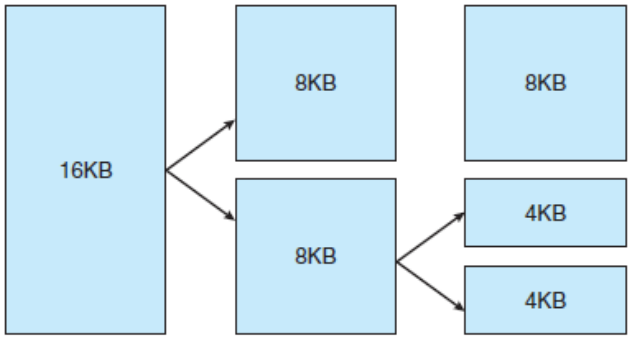
  • ZONE DMA
  • ZONE DMA32
  • ZONE NORMAL
  • ZONE HIGHMEM

- ZONE_DMA. This zone contains pages that can undergo DMA.
- ZONE_DMA32. Like ZOME_DMA, this zone contains pages that can undergo DMA. Unlike ZONE_DMA, these pages are accessible only by 32-bit devices. On some architectures, this zone is a larger subset of memory.
- ZONE_NORMAL. This zone contains normal, regularly mapped, pages.

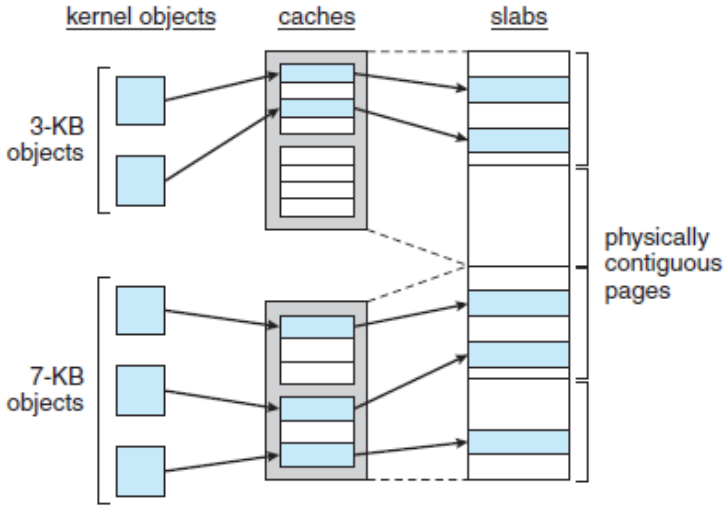
- ZONE_HIGHMEM. This zone contains "high memory", which are pages not permanently mapped into the kernel's address space. The relationship of zones and physical addresses on the Intel x86-32 architecture is shown Below

zone	physical memory
ZONE_DMA	< 16 MB
ZONE_NORMAL	16 .. 896 MB
ZONE_HIGHMEM	> 896 MB

- The primary physical-memory manager in the Linux kernel is the **page allocator**.
- Each zone has its own allocator, which is responsible for allocating and freeing all physical pages for the zone and is capable of allocating ranges of physically contiguous pages on request.
- The allocator uses a **buddy system** to keep track of available physical pages.
- Each allocatable memory region has an adjacent partner (or buddy). Whenever two allocated partner regions are freed up, they are combined to form a larger region—a buddy heap.



- Another strategy adopted by Linux for allocating kernel memory is known as slab allocation. A slab is used for allocating memory for kernel data structures and is made up of one or more physically contiguous pages. A cache consists of one or more slabs.



In Linux, a slab may be in one of three possible states:

1. Full. All objects in the slab are marked as used.
2. Empty. All objects in the slab are marked as free.
3. Partial. The slab consists of both used and free objects.

➔Virtual Memory

- The Linux virtual memory system is responsible for maintaining the address space accessible to each process.
- It creates pages of virtual memory on demand and manages loading those pages from disk and swapping them back out to disk as required.
- Under Linux, the virtual memory manager maintains two separate views of a process's address space: as a set of **separate regions** and as a **set of pages**.

1. Virtual Memory Regions

- Linux implements several types of virtual memory regions.
- One property that characterizes virtual memory is the backing store for the region, which describes where the pages for the region come from.
- Most memory regions are backed either by a file or by nothing.
- A region backed by nothing is the simplest type of virtual memory region.
- Such a region represents demand-zero memory: when a process tries to read a page in such a region, it is simply given back a page of memory filled with zeros.
- A virtual memory region is also defined by its reaction to writes. The mapping of a region into the process's address space can be either private or shared.

2. Lifetime of a Virtual Address Space

- The kernel creates a new virtual address space in two situations:
- when a process runs a new program with the exec() system call and when a new process is created by the fork() system call.

3. Swapping and Paging

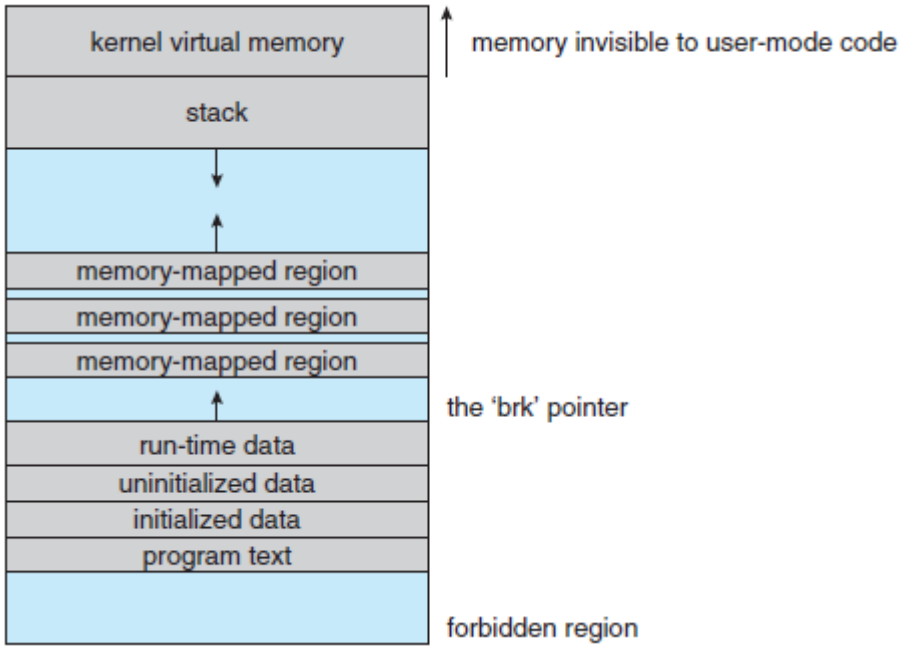
- An important task for a virtual memory system is to relocate pages of memory from physical memory out to disk when that memory is needed.
- The paging system can be divided into two sections.
- First, the policy algorithm decides which pages to write out to disk and when to write them.
- Second, the paging mechanism carries out the transfer and pages data back into physical memory when they are needed again.

4. Kernel Virtual Memory

- kernel virtual memory area contains two regions.
- The first is a static area that contains page-table references to every available physical page of memory in the system, so that a simple translation from physical to virtual addresses occurs when kernel code is run.
- The remainder of the kernel's reserved section of address space is not reserved for any specific purpose.

➔Execution and Loading of User Programs

- The Linux kernel's execution of user programs is triggered by a call to the exec() system call.
- This exec() call commands the kernel to run a new program within the current process, completely overwriting the current execution context with the initial context of the new program.
- The first job of this system service is to verify that the calling process has permission rights to the file being executed.
- Newer Linux systems use the more modern ELF format, now supported by most current UNIX implementations.



Memory layout for ELF programs.

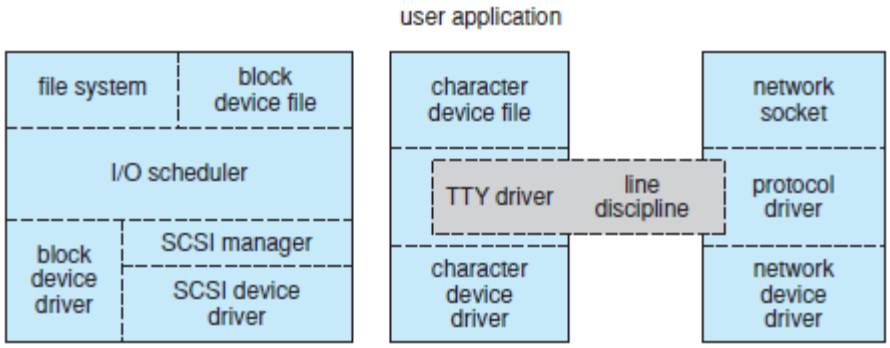
7. INPUT AND OUTPUT MANAGEMENT

- Linux splits all devices into three classes: block devices, character devices, and network devices.

➔Block Devices

- Block devices provide the main interface to all disk devices in a system.
- Performance is particularly important for disks, and the block-device system must provide functionality to ensure that disk access is as fast as possible.
- This functionality is achieved through the scheduling of I/O operations.

- In the context of block devices, a block represents the unit with which the kernel performs I/O.
- When a block is read into memory, it is stored in a buffer.
- The request manager is the layer of software that manages the reading and writing of buffer contents to and from a block-device driver.
- A separate list of requests is kept for each block-device driver.
- These requests have been scheduled according to a unidirectional-elevator (C-SCAN) algorithm that exploits the order in which requests are inserted in and removed from the lists.
- When a request is accepted for processing by a block-device driver, it is not removed from the list.
- It is removed only after the I/O is complete, at which point the driver continues with the next request in the list, even if new requests have been inserted in the list before the active request.



Device-driver block structure.

➔**Character Devices**

- A character-device driver can be almost any device driver that does not offer random access to fixed blocks of data.
- Any character-device drivers registered to the Linux kernel must also register a set of functions that implement the file I/O operations that the driver can handle.
- The kernel performs almost no preprocessing of a file read or write request to a character device. It simply passes the request to the device in question and lets the device deal with the request.
- A line discipline is an interpreter for the information from the terminal device.
- The most common line discipline is the **tty discipline**, which glues the terminal's data stream onto the standard input and output streams of a user's running processes, allowing those processes to communicate directly with the user's terminal

➔**Network devices**

- Network devices are dealt with differently from block and character devices.
- Users cannot directly transfer data to network devices. Instead, they must communicate indirectly by opening a connection to the kernel's networking subsystem.

8. INTERPROCESS COMMUNICATION

Linux provides a rich environment for processes to communicate with each other.

➔Synchronization and Signals

- The standard Linux mechanism for informing a process that an event has occurred is the signal.
- Signals can be sent from any process to any other process, with restrictions on signals sent to processes owned by another user.
- The kernel also generates signals internally. For example, it can send a signal to a server process when data arrive on a network channel, to a parent process when a child terminates, or to a waiting process when a timer expires.
- Internally, the Linux kernel does not use signals to communicate with processes running in kernel mode. If a kernel-mode process is expecting an event to occur, it will not use signals to receive notification of that event.
- Rather, communication about incoming asynchronous events within the kernel takes place through the use of scheduling states and **wait queue structures**
- Whenever a process wants to wait for some event to complete, it places itself on a wait queue associated with that event and tells the scheduler that it is no longer eligible for execution.
- Once the event has completed, every process on the wait queue will be awoken.

➔Passing of Data among Processes

- Linux offers several mechanisms for passing data among processes.
- The standard UNIX pipe mechanism allows a child process to inherit a communication channel from its parent; data written to one end of the pipe can be read at the other.
- Under Linux, pipes appear as just another type of inode to virtual file system software, and each pipe has a pair of wait queues to synchronize the reader and writer.
- Another process communications method, shared memory, offers an extremely fast way to communicate large or small amounts of data.
- Any data written by one process to a shared memory region can be read immediately by any other process that has mapped that region into its address space.

9. FILE SYSTEMS

- The Linux kernel handles all types of files by hiding the implementation details of any single file type behind a layer of software, the virtual file system (VFS).

➔The Virtual File System

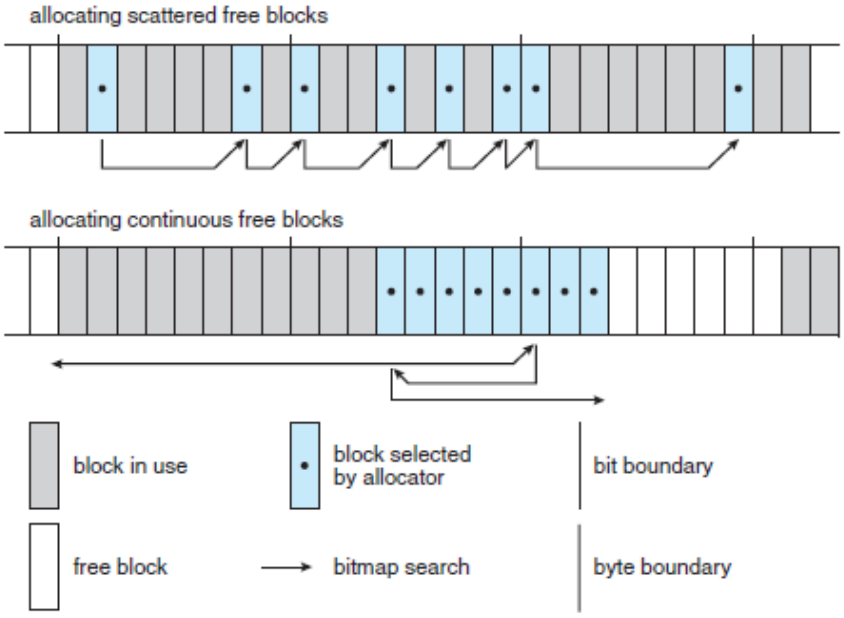
- The Linux VFS is designed around object-oriented principles. It has two components: a set of definitions that specify what file-system objects are allowed to look like and a layer of software to manipulate the objects.
- The VFS defines four main object types:
  - ➔ An inode object represents an individual file.
  - ➔ A file object represents an open file.
  - ➔ A superblock object represents an entire file system.
  - ➔ A dentry object represents an individual directory entry.
- For each of these four object types, the VFS defines a set of operations.

- ❖ Every object of one of these types contains a pointer to a function table.
- ❖ The function table lists the addresses of the actual functions that implement the defined operations for that object.
- ❖ For example, an abbreviated API for some of the file object's operations includes:

```
int open(. . .) — Open a file.
ssize t read(. . .) — Read from a file.
ssize t write(. . .) — Write to a file.
int mmap(. . .) — Memory-map a file.
```

➔The Linux ext3 File System

- ❖ The standard on-disk file system used by Linux is called ext3, for historical reasons.
- ❖ Linux was originally programmed with a Minix-compatible file system, to ease exchanging data with the Minix development system, but that file system was severely restricted by 14-character file-name limits and a maximum file-system size of 64 MB.
- ❖ The ext3 allocation policy works as follows: As in FFS, an ext3 file system is partitioned into multiple segments. In ext3, these are called block groups.
- ❖ FFS uses the similar concept of cylinder groups, where each group corresponds to a single cylinder of a physical disk.
- ❖ When allocating a file, ext3 must first select the block group for that file.
- ❖ For data blocks, it attempts to allocate the file to the block group to which the file's inode has been allocated. For inode allocations, it selects the lock group in which the file's parent directory resides for nondirectory files.



ext3 block-allocation policies.

➔Journaling

- The ext3 file system supports a popular feature called journaling, whereby modifications to the file system are written sequentially to a journal.
- A set of operations that performs a specific task is a transaction.
- Once a transaction is written to the journal, it is considered to be committed. Meanwhile, the journal entries relating to the transaction are replayed across the actual file system structures.
- As the changes are made, a pointer is updated to indicate which actions have completed and which are still incomplete. When an entire committed transaction is completed, it is removed from the journal.
- If the system crashes, some transactions may remain in the journal.
- Those transactions were never completed to the file system even though they were committed by the operating system, so they must be completed once the system recovers.
- The transactions can be executed from the pointer until the work is complete, and the file-system structures remain consistent.

➔The Linux Process File System

- The Linux process file system, known as the /proc file system, is an example of a file system whose contents are not actually stored anywhere but are computed on demand according to user file I/O requests.
- The /proc file system contains a illusionary file system.
- It does not exist on a disk. Instead, the kernel creates it in memory. It is used to provide information about the system (originally about processes, hence the name).
- Some of the more important files and directories are explained below. The /proc file system is described in more detail in the proc manual page.
- The /proc file system must implement **two things**: a **directory structure** and the **file contents within**.
- To allow efficient access to these variables from within applications, the /proc/sys subtree is made available through a special system call, sysctl(), that reads and writes the same variables in binary, rather than in text, without the overhead of the file system. sysctl() is not an extra facility; it simply reads the /proc dynamic entry tree to identify the variables to which the application is referring.

MOBILE OPERATING SYSTEMS

- A mobile operating system (OS) is software that allows smartphones, tablet PCs and other devices to run applications and programs.
- A mobile OS typically starts up when a device powers on, presenting a screen with icons or tiles that present information and provide application access. Mobile operating systems also manage cellular and wireless network connectivity, as well as phone access.
- Examples of mobile device operating systems include **Apple iOS**, **Google Android**, Research in Motion's **BlackBerry OS**, Nokia's Symbian, Hewlett-Packard's webOS (formerly Palm OS) and Microsoft's **Windows Phone OS**. Some, such as Microsoft's Windows 8, function as both a traditional desktop OS and a mobile operating system.

- ❖ Most mobile operating systems are tied to specific hardware, with little flexibility. Users can jailbreak or root some devices, however, which allows them to install another mobile OS or unlock restricted applications.

10.ANDRIOD VS IOS

IOS

It is Apple's mobile operating system used to run the popular iPhone, iPad, and iPod Touch devices. Formerly known as the iPhone OS, the name was changed with the introduction of the iPad. It interprets the commands of software applications ("apps") and it gives those apps access to features of the device, such as the multi-touch screen or the storage.

Features of IOS

- System Fonts
- Folders
- Notification center
- Accessibility
- Multitasking
- Switching Applications(application does not execute any code and may be removed from memory at any time)
- Task Completion (helps to ask extra time for completion of task)
- Background audio (helps to run in background)
- Voice over IP (in case phone call is not in progress)
- Background location (notified when location changes)
- Push notifications

ANDROID

Android is a software package and Linux based operating system for mobile devices such as tablet computers and smartphones. It is developed by Google in 2007 and later the OHA (Open Handset Alliance). Because Google developed Android, it comes with a lot of Google app services installed right out of the box. Gmail, Google Calendar, Google Maps, and Google Now are all pre-installed on most Android phones

Android OS has many features, among which are the following:

- Enhanced interface with the array of icons on the menu. Android adapts to high quality 2D and 3D graphics, with multi-touch support.
- Android supports multitasking, i.e. many applications will run at the same time, like in a computer. This is not possible with simple mobile phones and many other smartphones.
- All new means of connectivity are support: GSM, 3G, 4G, Wi-Fi, Bluetooth, GPS etc.
- Android supports many languages, including those with right-to-left text.
- Multimedia messaging system (MMS) is supported.

- Java runs great on Android. Applications for Android are developed in Java, but instead of a Java Runtime Environment, Android uses the Dalvik Executer, which is lighter on resources.
- Android supports most voice and video media formats, including streaming media.
- Additional hardware like sensors, gaming devices, other touchscreens can be integrated in Android.
- Voice and Video over IP. VoIP has many benefits, and Android manages cameras and has embedded support for seamless use of VoIP for free and cheap calls.
- On versions 2.2 and up, tethering is possible, which is the ability to use the Android device as a mobile WiFi hot spot.

Comparison chart

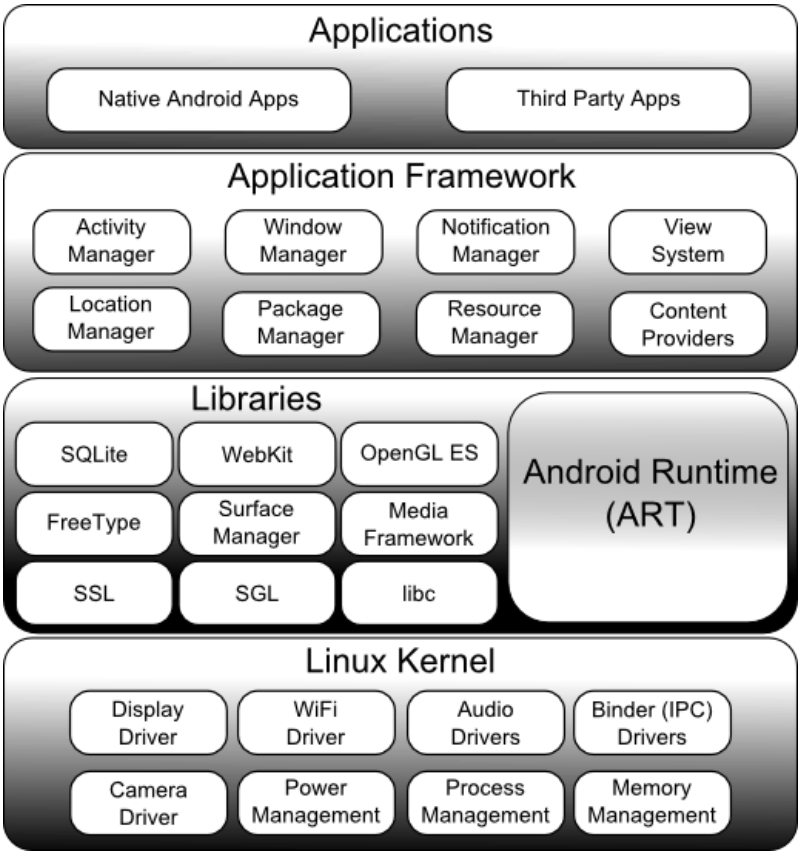
	Andriod	**iOS**
Source model	Open source	Closed, with open source components.
OS family	Linux	OS X, UNIX
Initial release	September 23, 2008	July 29, 2007
Customizability	A lot. Can change almost anything.	Limited unless jailbroken
Developer	Google, Open Handset Alliance	Apple Inc.
Widgets	Yes	No, except in NotificationCenter
Available language(s)	100+ Languages	34 Languages
File transfer	Easier than iOS. Using USB port and Android File Transfer desktop app. Photos can be transferred via USB without apps.	More difficult. Media files can be transferred using iTunes desktop app. Photos can be transferred out via USB without apps.
Available on	Many phones and tablets. Major manufacturers are Samsung, Motorola, LG, HTC and Sony.. Nexus and Pixel line of devices is pure Android, others bundle manufacturer software.	iPod Touch, iPhone, iPad, Apple TV (2nd and 3rd generation)
Calls and messaging	Google Hangouts. 3rd party apps like Facebook Messenger, WhatsApp, Google Duo and Skype all work on	iMessage, FaceTime (with other Apple devices only). 3rd party apps like Google Hangouts, Facebook

	Android and iOS both.	Messenger, WhatsApp, Google Duo and Skype all work on Android and iOS both.
Internet browsing	Google Chrome (or Android Browser on older versions; other browsers are available)	Mobile Safari (Other browsers are available)
App store , Affordability and interface	Google Play – 1,000,000+ apps. Other app stores like Amazon and Getjar also distribute Android apps. (unconfirmed ".APKs")	Apple app store – 1,000,000+ apps
Video chat	Google Duo and other 3rd party apps	FaceTime (Apple devices only) and other 3rd party apps
Voice commands	Google Now, Google Assistant	Siri
Working state	Current	Current
Maps	Google Maps	Apple Maps (Google Maps also available via a separate app download)
Latest stable release and Updates	Android 8.0.0, Oreo (Aug 21, 2017)	11 (Sep 19, 2017)
Alternative app stores and side loading	Several alternative app stores other than the official Google Play Store. (e.g. Aptoide, Galaxy Apps)	Apple blocks 3rd party app stores. The phone needs to be jailbroken if you want to download apps from other stores.
Battery life and management	Many Android phone manufacturers equip their devices with large batteries with a longer life.	Apple batteries are generally not as big as the largest Android batteries. However, Apple is able to squeeze decent battery life via hardware/software optimizations.
Open source	Kernel, UI, and some standard apps	The iOS kernel is not open source but is based on the open-source Darwin OS.
File manager	Yes. (Stock Android File Manager included on devices running Android	Not available

	7.1.1)	
Photos & Videos backup	Apps available for automatic backup of photos and videos. Google Photos allows unlimited backup of photos. OneDrive, Amazon Photos and Dropbox are other alternatives.	Up to 5 GB of photos and videos can be automatically back up with iCloud. All other vendors like Google, Amazon, Dropbox, Flickr and Microsoft have auto-backup apps for both iOS and Android.
Security	Android software patches are available soonest to Nexus device users. Manufacturers tend to lag behind in pushing out these updates. So at any given time a vast majority of Android devices are not running updated fully patched software.	Most people will never encounter a problem with malware because they don't go outside the Play Store for apps. Apple's software updates support older iOS devices also.
Rooting, bootloaders, and jailbreaking	Access and complete control over your device is available and you can unlock the bootloader.	Complete control over your device is not available.
Cloud services	Native integration with Google cloud storage. 15GB free, $2/mo for 100GB, 1TB for $10. Apps available for Amazon Photos, OneDrive and Dropbox.	Native integration with iCloud. 5GB free, 50GB for $1/mo, 200GB for $3/mo, 1TB for $10/mo. Apps available for Google Drive and Google Photos, Amazon Photos, OneDrive and Dropbox.
Interface	Touch Screen	Touch Screen
Supported versions	Android 5.0 & later (Android 4.4 is also supported but with patches)	iOS 8 & later
First version	Android 1.0, Alpha	iOS 1.0

11. IOS AND ANDROID ARCHITECTURE AND SDK FRAMEWORK

1. Android Architecture



Android System Architecture

The Android software stack generally consists of a Linux kernel and a collection of C/C++ libraries that is exposed through an application framework that provides services, and management of the applications and run time.

Linux Kernel

Android was created on the open source kernel of Linux. One main reason for choosing this kernel was that it provided proven core features on which to develop the Android operating system. The features of Linux kernel are:

1. Security:

The Linux kernel handles the security between the application and the system.

2. Memory Management:

It efficiently handles the memory management thereby providing the freedom to develop our apps.

3.Process Management:

It manages the process well, allocates resources to processes whenever they need them.

4. Network Stack:

It effectively handles the network communication.

5. Driver Model:

It ensures that the application works. Hardware manufacturers can build their drivers into the Linux build.

Libraries:

Running on the top of the kernel, the Android framework was developed with various features. It consists of various C/C++ core libraries with numerous of open source tools. Some of these are:

1. **The Android runtime:**
   The Android runtime consist of core libraries of Java and ART(the Android RunTime). Older versions of Android (4.x and earlier) had Dalvik runtime.
2. **Open GL(graphics library):**
   This cross-language, cross-platform application program interface (API) is used to produce 2D and 3D computer graphics.
3. **WebKit:**
   This open source web browser engine provides all the functionality to display web content and to simplify page loading.
4. **Media frameworks:**
   These libraries allow you to play and record audio and video.
5. **Secure Socket Layer (SSL):**
   These libraries are there for Internet security.

Android Runtime:

It is the third section of the architecture. It provides one of the key components which is called Dalvik Virtual Machine. It acts like Java Virtual Machine which is designed specially for Android. Android uses it's own custom VM designed to ensure that multiple instances run efficiently on a single device.

The Delvik VM uses the device's underlying Linux kernel to handle low-level functionality,including security,threading and memory management.

Application Framework

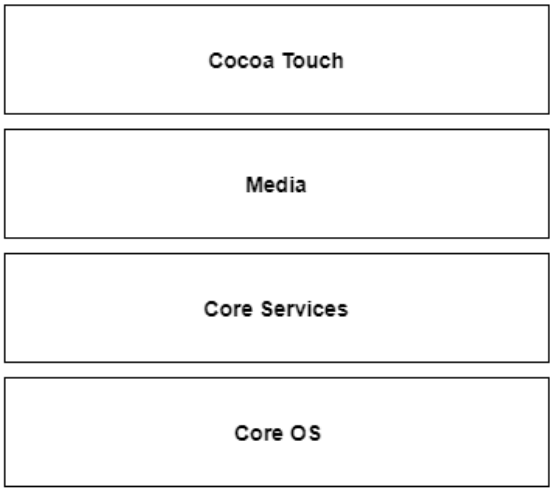
The Android team has built on a known set proven libraries, built in the background, and all of it these is exposed through Android interfaces. These interfaces warp up all the various libraries and make them useful for the Developer. They don't have to build any of the functionality provided by the android. Some of these interfaces include:

1. **Activity Manager:**
   It manages the activity lifecycle and the activity stack.
2. **Telephony Manager:**
   It provides access to telephony services as related subscriber information, such as phone numbers.
3. **View System:**
   It builds the user interface by handling the views and layouts.
4. **Location manager:**
   It finds the device's geographic location.

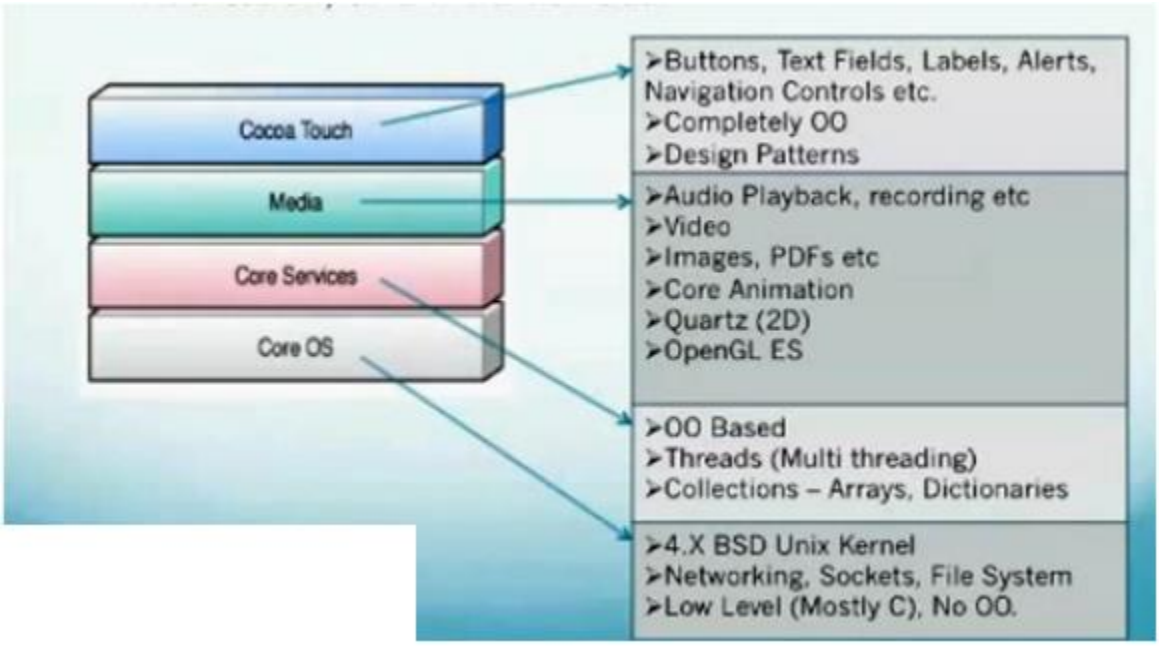
Applications:

Android applications can be found at the topmost layer. At application layer we write our application to be installed on this layer only. Examples of applications are Games, Messages, Contacts etc.

2. iOS Architecture



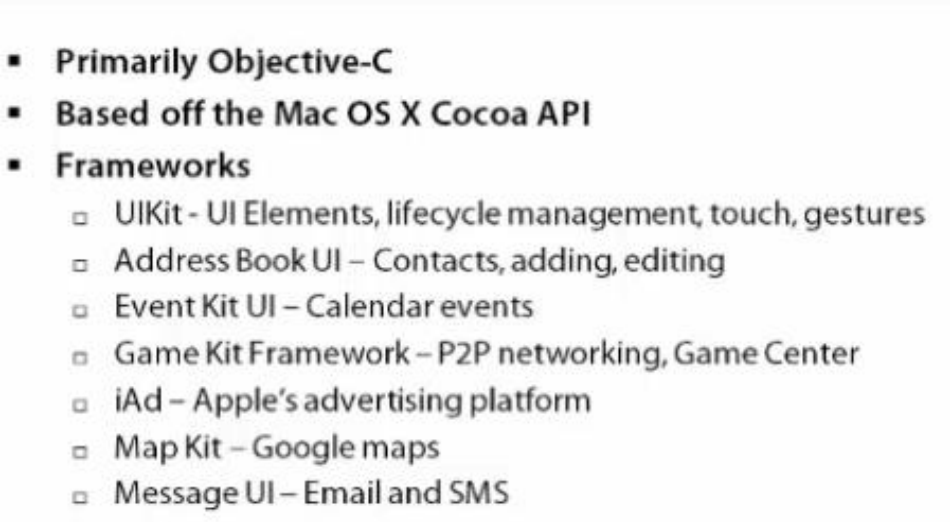
Apple iOS Architecture



Cocoa Touch Layer

The Cocoa Touch layer sits at the top of the iOS stack and contains the frameworks that are most commonly used by iPhone application developers. Cocoa Touch is primarily written in Objective-C, is based on the standard Mac OS X Cocoa API (as found on Apple

desktop and laptop computers) and has been extended and modified to meet the needs of the iPhone.



The iOS Media Layer

The role of the Media layer is to provide iOS with audio, video, animation and graphics capabilities. As with the other layers comprising the iOS stack, the Media layer comprises a number of frameworks that may be utilized when developing iPhone apps.

Core OS Layer:

All the iOS technologies are build on the low level features provided by the Core OS layer. These technologies include Core Bluetooth Framework, External Accessory Framework, Accelerate Framework, Security Services Framework, Local Authorisation Framework etc.

iOS Core Services

The iOS Core Services layer provides much of the foundation on which the previously referenced layers are built

********** **End of Andriod and iOS Architecture Framework**********

12. THE iOS MEDIA LAYER

- The role of the Media layer is to provide iOS with audio, video, animation and graphics capabilities.
- As with the other layers comprising the iOS stack, the Media layer comprises a number of frameworks that may be utilized when developing iPhone apps.

Core Video Framework (CoreVideo.framework)

A new framework introduced with iOS 4 to provide buffering support for the Core Media framework. This may be utilized by application developers it is typically not necessary to use this framework.

Core Text Framework (CoreText.framework)

- The iOS Core Text framework is a C-based API designed to ease the handling of advanced text layout and font rendering requirements.

Image I/O Framework (ImageIO.framework)

- The Image IO framework, the purpose of which is to facilitate the importing and exporting of image data and image metadata, was introduced in iOS 4.
- The framework supports a wide range of image formats including PNG, JPEG, TIFF and GIF.

Assets Library Framework (AssetsLibrary.framework)

- The Assets Library provides a mechanism for locating and retrieving video and photo files located on the iPhone device.
- In addition to accessing existing images and videos, this framework also allows new photos and videos to be saved to the standard device photo album.

Core Graphics Framework (CoreGraphics.framework)

- The iOS Core Graphics Framework (otherwise known as the Quartz 2D API) provides a lightweight two dimensional rendering engine.
- Features of this framework include PDF document creation and presentation, vector based drawing, transparent layers, path based drawing, anti-aliased rendering, color manipulation and management, image rendering and gradients.
- Those familiar with the Quartz 2D API running on MacOS X will be pleased to learn that the implementation of this API is the same on iOS.

Quartz Core Framework (QuartzCore.framework)

- The purpose of the Quartz Core framework is to provide animation capabilities on the iPhone.
- It provides the foundation for the majority of the visual effects and animation used by the UIKit framework and provides an Objective-C based programming interface for creation of specialized animation within iPhone apps.

OpenGL ES framework (OpenGLES.framework)

- For many years the industry standard for high performance 2D and 3D graphics drawing has been OpenGL.
- Originally developed by the now defunct Silicon Graphics, Inc (SGI) during the 1990s in the form of GL, the open version of this technology (OpenGL) is now under the care of a non-profit consortium comprising a number of major companies including Apple, Inc., Intel, Motorola and ARM Holdings.

- OpenGL for Embedded Systems (ES) is a lightweight version of the full OpenGL specification designed specifically for smaller devices such as the iPhone,iOS 3 or later supports both OpenGL ES 1.1 and 2.0 on certain iPhone models (such as the iPhone 3GS and iPhone 4). Earlier versions of iOS and older device models support only OpenGL ES version 1.1.

iOS Audio Support

- iOS is capable of supporting audio in AAC, Apple Lossless (ALAC), A-law, IMA/ADPCM, Linear PCM, µ-law, DVI/Intel IMA ADPCM, Microsoft GSM 6.10 and AES3-2003 formats through the support provided by the following frameworks.

AV Foundation framework (AVFoundation.framework)

- An Objective-C based framework designed to allow the playback, recording and management of audio content.

Core Audio Frameworks (CoreAudio.framework, AudioToolbox.framework and AudioUnit.framework)

- The frameworks that comprise Core Audio for iOS define supported audio types, playback and recording of audio files and streams and also provide access to the device's built-in audio processing units.

Open Audio Library (OpenAL)

- OpenAL is a cross platform technology used to provide high-quality, 3D audio effects (also referred to as positional audio).
- Positional audio can be used in a variety of applications though is typically using to provide sound effects in games.

Media Player framework (MediaPlayer.framework)

- The iOS Media Player framework is able to play video in .mov, .mp4, .m4v, and .3gp formats at a variety of compression standards, resolutions and frame rates.

Core Midi Framework (CoreMIDI.framework)

- Introduced in iOS 4, the Core MIDI framework provides an API for applications to interact with MIDI compliant devices such as synthesizers and keyboards via the iPhone's dock connector.

*****************************End of iOS Media Layer*****************************

13. THE iOS CORE SERVICES LAYER

- The iOS Core Services layer provides much of the foundation on which the previously referenced layers are built and consists of the following frameworks.

Address Book framework (AddressBook.framework)

- The Address Book framework provides programmatic access to the iPhone Address Book contact database allowing applications to retrieve and modify contact entries.

CFNetwork Framework (CFNetwork.framework)

- The CFNetwork framework provides a C-based interface to the TCP/IP networking protocol stack and low level access to BSD sockets.
- This enables application code to be written that works with HTTP, FTP and Domain Name servers and to establish secure and encrypted connections using Secure Sockets Layer (SSL) or Transport Layer Security (TLS).

Core Data Framework (CoreData.framework)

- This framework is provided to ease the creation of data modeling and storage in Model-View-Controller (MVC) based applications.
- Use of the Core Data framework significantly reduces the amount of code that needs to be written to perform common tasks when working with structured data in an application.

Core Foundation Framework (CoreFoundation.framework)

- The Core Foundation is a C-based Framework that provides basic functionality such as data types, string manipulation, raw block data management, URL manipulation, threads and run loops, date and times, basic XML manipulation and port and socket communication. Additional XML capabilities beyond those included with this framework are provided via the libXML2 library.
- Though this is a C-based interface, most of the capabilities of the Core Foundation framework are also available with Objective-C wrappers via the Foundation Framework.

Core Media Framework (CoreMedia.framework)

- The Core Media framework is the lower level foundation upon which the AV Foundation layer is built.
- While most audio and video tasks can, and indeed should, be performed using the higher level AV Foundation framework, access is also provided for situations where lower level control is required by the iOS application developer.

Core Telephony Framework (CoreTelephony.framework)

- The iOS Core Telephony framework is provided to allow applications to interrogate the device for information about the current cell phone service provider and to receive notification of telephony related events.

EventKit Framework (EventKit.framework)

- An API designed to provide applications with access to the calendar and alarms on the device.

Foundation Framework (Foundation.framework)

- The Foundation framework is the standard Objective-C framework that will be familiar to those that have programmed in Objective-C on other platforms (most likely Mac OS X).

- Essentially, this consists of Objective-C wrappers around much of the C-based Core Foundation Framework.

Core Location Framework (CoreLocation.framework)

- The Core Location framework allows you to obtain the current geographical location of the device (latitude and longitude) and compass readings from with your own applications.
- The method used by the device to provide coordinates will depend on the data available at the time the information is requested and the hardware support provided by the particular iPhone model on which the app is running (GPS and compass are only featured on recent models).
- This will either be based on GPS readings, Wi-Fi network data or cell tower triangulation (or some combination of the three).

Mobile Core Services Framework (MobileCoreServices.framework)

- The iOS Mobile Core Services framework provides the foundation for Apple's Uniform Type Identifiers (UTI) mechanism, a system for specifying and identifying data types.
- A vast range of predefined identifiers have been defined by Apple including such diverse data types as text, RTF, HTML, JavaScript, PowerPoint .ppt files, PhotoShop images and MP3 files.

Store Kit Framework (StoreKit.framework)

- The purpose of the Store Kit framework is to facilitate commerce transactions between your application and the
- Apple App Store. Prior to version 3.0 of iOS, it was only possible to charge a customer for an app at the point that they purchased it from the App Store. iOS 3.0 introduced the concept of the "in app purchase" whereby the user can be given the option make additional payments from within the application.
- This might, for example, involve implementing a subscription model for an application, purchasing additional functionality or even buying a faster car for you to drive in a racing game.

SQLite library

- Allows for a lightweight, SQL based database to be created and manipulated from within your iPhone application.

System Configuration Framework (SystemConfiguration.framework)

- The System Configuration framework allows applications to access the network configuration settings of the device to establish information about the "reachability" of the device (for example whether Wi-Fi or cell connectivity is active and whether and how traffic can be routed to a server).

Quick Look Framework (QuickLook.framework)

- One of the many new additions included in iOS 4, the Quick Look framework provides a useful mechanism for displaying previews of the contents of files types loaded onto the

device (typically via an internet or network connection) for which the application does not already provide support.

- File format types supported by this framework include iWork, Microsoft Office document, Rich Text Format, Adobe PDF, Image files, public.text files and comma separated (CSV).

********************************End of Core Services Layer************************

14.The iOS Core OS Layer

- The Core OS Layer occupies the bottom position of the iOS stack and, as such, sits directly on top of the device hardware.
- The layer provides a variety of services including low level networking, access to external accessories and the usual fundamental operating system services such as memory management, file system handling and threads.

Accelerate Framework (Accelerate.framework)

- Introduced with iOS 4, the Accelerate Framework provides a hardware optimized C-based API for performing complex and large number math, vector, digital signal processing (DSP) and image processing tasks and calculations.

External Accessory framework (ExternalAccessory.framework)

- Provides the ability to interrogate and communicate with external accessories connected physically to the iPhone via the 30-pin dock connector or wirelessly via Bluetooth.

Security Framework (Security.framework)

- The iOS Security framework provides all the security interfaces you would expect to find on a device that can connect to external networks including certificates, public and private keys, trust policies, keychains, encryption, digests and Hash-based Message Authentication Code (HMAC).

System (LibSystem)

- As we have previously mentioned, the iOS is built upon a UNIX-like foundation.
- The System component of the Core OS Layer provides much the same functionality as any other UNIX like operating system. This layer includes the operating system kernel (based on the Mach kernel developed by Carnegie Mellon University) and device drivers.
- The kernel is the foundation on which the entire iOS is built and provides the low level interface to the underlying hardware.
- Amongst other things the kernel is responsible for memory allocation, process lifecycle management, input/output, inter-process communication, thread management, low level networking, file system access and thread management.
- As an app developer your access to the System interfaces is restricted for security and stability reasons. Those interfaces that are available to you are contained in a C-based library called LibSystem.

**********************End od Core OS Layer****************************

15. FILE SYSTEM BASICS

- A *file system* handles the persistent storage of data files, apps, and the files associated with the operating system itself. Therefore, the file system is one of the fundamental resources used by all processes.
- APFS is the default file system in macOS, iOS, watchOS, and tvOS. APFS replaces HFS+ as the default file system for iOS 10.3 and later, and macOS High Sierra and later macOS additionally supports a variety of other formats, as described in Supported File Systems.
- The file system uses directories to create a hierarchical organization
- Before you begin writing code that interacts with the file system, you should first understand a little about the organization of file system and the rules that apply to your code.
- Aside from the basic tenet that you cannot write files to directories for which you do not have appropriate security privileges, apps are also expected to be good citizens and put files in appropriate places.
- Precisely where you put files depends on the platform, but the overarching goal is to make sure that the user's files remain easily discoverable and that the files your code uses internally are kept out of the user's way.

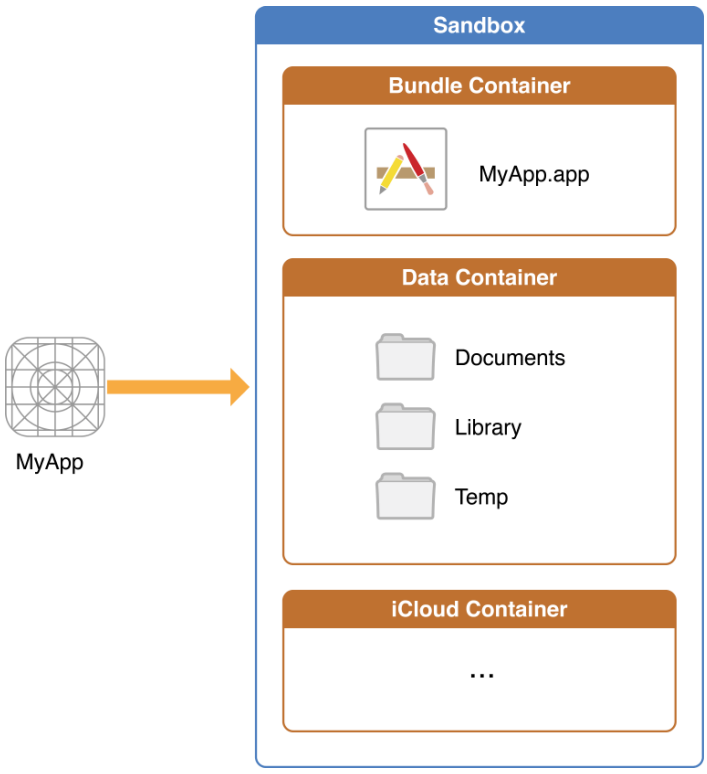
About the iOS File System

- The iOS file system is geared toward apps running on their own. To keep the system simple, users of iOS devices do not have direct access to the file system and apps are expected to follow this convention.

iOS Standard Directories: Where Files Reside ?

- For security purposes, an iOS app's interactions with the file system are limited to the directories inside the app's sandbox directory.
- During installation of a new app, the installer creates a number of container directories for the app inside the sandbox directory.
- Each container directory has a specific role.
- The bundle container directory holds the app's bundle, whereas the data container directory holds data for both the app and the user.
- The data container directory is further divided into a number of subdirectories that the app can use to sort and organize its data.

- The app may also request access to additional container directories—for example, the iCloud container—at runtime.
- These container directories constitute the app's primary view of the file system. The Figure shows a representation of the sandbox directory for an app.
- An iOS app operating within its own sandbox directory



- An app is generally prohibited from accessing or creating files outside its container directories.
- One exception to this rule is when an app uses public system interfaces to access things such as the user's contacts or music.
- In those cases, the system frameworks use helper apps to handle any file-related operations needed to read from or modify the appropriate data stores.
- Table lists some of the more important subdirectories inside the sandbox directory and describes their intended usage.
- This table also describes any additional access restrictions for each subdirectory and points out whether the directory's contents are backed up by iTunes and iCloud.

Table Commonly used directories of an iOS app

Directory	Description

AppName`.app`	This is the app's bundle. This directory contains the app and all of its resources. You cannot write to this directory. To prevent tampering, the bundle directory is signed at installation time. Writing to this directory changes the signature and prevents your app from launching. You can, however, gain read-only access to any resources stored in the apps bundle. For more information, see the *Resource Programming Guide* The contents of this directory are not backed up by iTunes or iCloud. However, iTunes does perform an initial sync of any apps purchased from the App Store.
`Documents/`	Use this directory to store user-generated content. The contents of this directory can be made available to the user through file sharing; therefore, his directory should only contain files that you may wish to expose to the user. The contents of this directory are backed up by iTunes and iCloud.
`Documents/Inbox`	Use this directory to access files that your app was asked to open by outside entities. Specifically, the Mail program places email attachments associated with your app in this directory. Document interaction controllers may also place files in it. Your app can read and delete files in this directory but cannot create new files or write to existing files. If the user tries to edit a file in this directory, your app must silently move it out of the directory before making any changes. The contents of this directory are backed up by iTunes and iCloud.
`Library/`	This is the top-level directory for any files that are not user data files. You typically put files in one of several standard subdirectories. iOS apps commonly use the `Application Support` and `Caches` subdirectories; however, you can create custom subdirectories. Use the `Library` subdirectories for any files you don't want exposed to the user. Your app should not use these directories for user data files. The contents of the `Library` directory (with the exception of the `Caches` subdirectory) are backed up by iTunes and iCloud. For additional information about the Library directory and its commonly used subdirectories, see The Library Directory Stores App-Specific Files.
`tmp/`	Use this directory to write temporary files that do not need to persist between launches of your app. Your app should remove files from this directory when they are no longer needed; however, the system may purge this directory when your app is not running.

	The contents of this directory are not backed up by iTunes or iCloud.

An iOS app may create additional directories in the `Documents`, `Library`, and `tmp` directories. You might do this to better organize the files in those locations.

Where You Should Put Your App's Files

- To prevent the syncing and backup processes on iOS devices from taking a long time, be selective about where you place files. Apps that store large files can slow down the process of backing up to iTunes or iCloud.
- These apps can also consume a large amount of a user's available storage, which may encourage the user to delete the app or disable backup of that app's data to iCloud. With this in mind, you should store app data according to the following guidelines:
- **Put user data in `Documents/`.** User data generally includes any files you might want to expose to the user—anything you might want the user to create, import, delete or edit. For a drawing app, user data includes any graphic files the user might create. For a text editor, it includes the text files. Video and audio apps may even include files that the user has downloaded to watch or listen to later.
- **Put app-created support files in the `Library/Application support/`** directory. In general, this directory includes files that the app uses to run but that should remain hidden from the user. This directory can also include data files, configuration files, templates and modified versions of resources loaded from the app bundle.
- **Remember that files in `Documents/` and `Application Support/`** are backed up by default. You can exclude files from the backup by calling using the **`NSURLIsExcludedFromBackupKey` key**. Any file that can be re-created or downloaded must be excluded from the backup. This is particularly important for large media files. If your application downloads video or audio files, make sure they are not included in the backup.
- **Put temporary data in the `tmp/`** directory. Temporary data comprises any data that you do not need to persist for an extended period of time. Remember to delete those files when you are done with them so that they do not continue to consume space on the user's device. The system will periodically purge these files when your app is not running; therefore, you cannot rely on these files persisting after your app terminates.
- **Put data cache files in the Library/Caches/** directory. Cache data can be used for any data that needs to persist longer than temporary data, but not as long as a support file. Generally speaking, the application does not require cache data to operate properly, but it can use cache data to improve performance.
- Examples of cache data include (but are not limited to) database cache files and transient, downloadable content. Note that the system may delete the **Caches/** directory to free up disk space, so your app must be able to re-create or download these files as needed.

PART A

1. **What are the components of a Linux System?**
   Linux system composed of three main modules . They are:
   i) Kernel
   ii) System libraries
   iii) System utilities
2. **What are the main supports for the Linux modules?**
   The Module support under Linux has three components. They are:
   i) Module management
   ii) Driver registration
   iii) Conflict resolution mechanism.
3. **Define shell.**
   A shell is a program that provides the traditional, text-only user interface for Linux and other Unix-like operating systems. Its primary function is to read commands that are typed into the console.
4. **What is meant by kernel in Linux system?**
   Kernel is responsible for maintaining all the important abstractions of the operating system including such things as virtual memory and processes.
5. **What is meant by system Libraries?**
   System libraries define a standard set of functions through which applications can interact with the kernel and that implement much of the operating-system functionality that doesn't need the full privileges of kernel code.
6. **What is meant by system Utilities?**
   System Utilities are system programs that perform individual, specialized management tasks. Some of the system utilities may be invoked just to initialize and configure some aspect of the system and others may run permanently, handling such tasks as responding to incoming network connections, accepting logon requests from terminals or updating log files.
7. **What do you meant by process?**
   Process is the basic context within in which all user-requested activity is serviced with in the OS.
8. **What is meant by Process-ID**
   A PID is an acronym for process identification number on a Linux or Unix-like operating system. A PID is automatically assigned to each process when it is created. A process is nothing but running instance of a program and each process has a unique PID on a Unix-like system.
9. **What is meant by personality?**
   Process personality are primarily used by emulation libraries to request that system call be compatible with certain version of UNIX
10. **What is meant by buffer cache?**
    It is the kernel's main cache for blocked-oriented devices such as disk drivers and is the main mechanism through which I/O to these devices is performed.
11. **What is the disadvantage of static linking?**
    The main disadvantage of static linking is that every program generated must contain copies of exactly the same common system library functions.

12. What is the function of module management?

The module management allows modules to be loaded into memory and to talk to the rest of the kernel.

13. What is the function of driver registration?

Driver registration allows modules to tell the rest of the kernel that a new driver has become available.

14. What is the function of conflict resolution mechanism?

This mechanism allows different device drivers to reserve hardware resources and to protect those resources from accidental use by another driver.

15. What is meant by device drivers?

Device drivers include i) character devices such as printers, terminals ii) Block devices (including all disk drives) and network interface devices.

16. What is a character device?

A device driver which does not offer random access to fixed blocks of data. A character device driver must register a set of functions which implement the driver's various file I/O operations.

17. What is Mobile OS?

A mobile operating system (mobile OS) is an OS built exclusively for a mobile device, such as a smartphone, personal digital assistant (PDA), tablet or other embedded mobile OS.

18. What is iOS?

iOS is a mobile operating system created and developed by Apple Inc. exclusively for its hardware. It is the operating system that presently powers many of the company's mobile devices, including the iPhone, iPad, and iPod Touch. It is the second most popular mobile operating system globally after Android.

19. List the services available in iOS.

i) Cocoa Touch
ii) Media layer
iii) Service layer
iv) Core OS layer

20. List the features of iOS.

i) System fonts
ii) Folders
iii) Notification center
iv) Accessibility
v) Multitasking
vi) Switching Applications
vii) Task completion
viii) Background audio
ix) Voice over IP
x) Background Location
xi) Push notification

21. List the advantages of iOS

- Best gaming experience.
- A vast number of applications.

- Suits for business and gaming.
- Excellent UI and fluid responsive.
- The latest version has two notification menus.
- Excellent security.
- Multitasking.
- Jailbreaking for customization.
- Wearables are getting launched.
- Feel is awesome.
- Excellent for media entertainment.
- Multi-language support.
- Apple Pay Support.
- Quick settings in the notification bar.

22. List the disadvantages of iOS

- It has a review process, when developers want to publish an app they need to send it to Apple for review that takes around 7 days and it takes even more in some cases.
- Applications are very large when compared to other mobile platforms
- Using iOS are costly Apps and no widget support
- Battery performance is very poor on 3G
- Repair costs are very piracy
- Not flexible only supports iOS devices

23. List the advantages of Android

- Android Is More Customizable Can change almost anything.
- In Android, any new publication can be done easily and without any review process
- Use a Different Messaging App for SMS
- Android Offers an Open Platform
- Easy access to the Android App Market
  - Cost Effective
- Upcoming versions have a support to save RAW images
- Built in Beta Testing and staged rollout

24. List the disadvantages of Android

- Need internet connection
- Advertising
- Wasteful memory
- Many application contain viruses

25. How are iOS and Android similar? How are they different?

Similarities: Both are based on existing kernel. Both have architecture that uses software stacks. Both provide framework for developers

Difference: iOS is closed-source and Android is open source. iOS applications are developed in objective C, Android in java. Android uses a virtual machine, and iOS executes code natively.

26. Describe some challenges of designing OS for mobile device compared with designing OS for traditional PC's

- Less storage capacity means the OS must be manage memory
- Less processing power plus fewer processors mean the operating system carefully apportion
- Processors to applications

Part- B

1. Draw a neat sketch of overview of iOS architecture and explain in detail. (13)
2. Discuss process management and scheduling in LINUX. (13)
3. Illustrate some existing SDK architecture implementation frameworks.(13)
4. Describe about the network structure of LINUX system.(13)
5. Explain in detail the design principles, kernel modules in LINUX system. (7+6)
6. Demonstrate the functions of the kernel, service and command layers ofOS.(13)
7. Generalize the importance of memory management in Operating system.(13)
8. Explain in detail about file system management done in LINUX.(13)
9. Summarize Inter Process Communication with suitable example.(13)
10. Analyze:
    a. mobile OS (5) ii) desktop OS(4) iii) multi-user OS (4)
11. Compare and contrast Andriod OS and IOS. (13)
12. Explain in detail about Linux architecture. (13)
13. Compare the functions of media layer, service layer and core OS layer.
14. Explain the basic concepts of the Linux system
15. 2. Explain about kernel modules
16. 3. Explain in detail about the process management in Linux
17. 4. Explain in detail about the scheduling in Linux
18. 5. Explain the iOS architecture and various layers available in iOS
19. Discuss about various services in the media layer
20. Discuss about various services in the iOS core OS layer
21. Discuss about various services in the iOS service OS layer

CS6703 GRID AND CLOUD COMPUTING

OBJECTIVES:

The student should be made to:

- Understand how Grid computing helps in solving large scale scientific problems.
- Gain knowledge on the concept of virtualization that is fundamental to cloud computing.
- Learn how to program the grid and the cloud.
- Understand the security issues in the grid and the cloud environment.

UNIT I INTRODUCTION 9

Evolution of Distributed computing: Scalable computing over the Internet – Technologies for network based systems – clusters of cooperative computers – Grid computing Infrastructures – cloud computing – service oriented architecture – Introduction to Grid Architecture and standards – Elements of Grid – Overview of Grid Architecture.

UNIT II GRID SERVICES 9

Introduction to Open Grid Services Architecture (OGSA) – Motivation – Functionality Requirements – Practical & Detailed view of OGSA/OGSI – Data intensive grid service models – OGSA services.

UNIT III VIRTUALIZATION 9

Cloud deployment models: public, private, hybrid, community – Categories of cloud computing: Everything as a service: Infrastructure, platform, software – Pros and Cons of cloud computing – Implementation levels of virtualization – virtualization structure – virtualization of CPU, Memory and I/O devices – virtual clusters and Resource Management – Virtualization for data center automation.

UNIT IV PROGRAMMING MODEL 9

Open source grid middleware packages – Globus Toolkit (GT4) Architecture , Configuration – Usage of Globus – Main components and Programming model – Introduction to Hadoop Framework – Mapreduce, Input splitting, map and reduce functions, specifying input and output parameters, configuring and running a job – Design of Hadoop file system, HDFS concepts, command line and java interface, dataflow of File read & File write.

UNIT V SECURITY 9

Trust models for Grid security environment – Authentication and Authorization methods – Grid security infrastructure – Cloud Infrastructure security: network, host and application level – aspects of data security, provider data and its security, Identity and access management architecture, IAM practices in the cloud, SaaS, PaaS, IaaS availability in the cloud, Key privacy issues in the cloud.

TOTAL: 45 PERIODS

OUTCOMES:

At the end of the course, the student should be able to:

Apply grid computing techniques to solve large scale scientific problems.

Apply the concept of virtualization.

Use the grid and cloud tool kits.

Apply the security models in the grid and the cloud environment.

TEXT BOOK:

1. Kai Hwang, Geoffery C. Fox and Jack J. Dongarra, "Distributed and Cloud Computing: Clusters, Grids, Clouds and the Future of Internet", First Edition, Morgan Kaufman Publisher, an Imprint of Elsevier, 2012.

REFERENCES: 1. Jason Venner, "Pro Hadoop- Build Scalable, Distributed Applications in the Cloud", A Press, 2009

2. Tom White, "Hadoop The Definitive Guide", First Edition. O"Reilly, 2009.

3. Bart Jacob (Editor), "Introduction to Grid Computing", IBM Red Books, Vervante, 2005

4. Ian Foster, Carl Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", 2nd Edition, Morgan Kaufmann.

5. Frederic Magoules and Jie Pan, "Introduction to Grid Computing" CRC Press, 2009.

6. Daniel Minoli, "A Networking Approach to Grid Computing", John Wiley Publication, 2005.

7. Barry Wilkinson, "Grid Computing: Techniques and Applications", Chapman and Hall, CRC, Taylor and Francis Group, 2010.

CS6703 / GRID AND CLOUD COMPUTING

Unit I

INTRODUCTION

1. Define cloud computing

Cloud computing is the delivery of computing as a service rather than a product, hereby shared resources, software, and information are provided to computers and other devices as a utility.

2. What is Distributed computing

This is a field of computer science/engineering that studies distributed systems. A distributed system consists of multiple autonomous computers, each having its own private memory, communicating through a computer network. Information exchange in a distributed system is accomplished through message passing. A computer program that runs in a distributed system is known as a distributed program. The process of writing distributed programs is referred to as distributed programming.

3. Difference between distributed and parallel computing.

Distributed computing	Parallel computing
• Each processor has its own private memory (distributed memory). Information is exchanged by passing messages between the processors. • It is loosely coupled. • An important goal and challenge of distributed systems is location transparency.	• All processors may have access to a shared memory to exchange information between processors. • It is tightly coupled. • Large problems can often be divided into smaller ones, which are then solved concurrently ("in parallel").

4. What is mean by service oriented architecture?

In grids/web services, Java, and CORBA, an entity is, respectively, a service, a Java object, and a CORBA distributed object in a variety of languages. These architectures build on the traditional seven Open Systems Interconnection (OSI) layers that provide the base networking abstractions. On top of this we have a base software environment, which would be .NET or Apache Axis for web services, the Java Virtual Machine for Java, and a broker network for CORBA.

5. State High Performance Computing (HPC)?

HPC: use of parallel processing to execute large programs quickly; often equated to supercomputers, typically applied to systems generating teraflops (10^{12}) or more. Emphasis: raw speed performance and accuracy

State High Throughput Computing (HTC)?

HTC: running a job that takes days to complete, or an application that must produce a high number of completed operations per unit of time. Performance measured in flops per month or year, as opposed to per second.eg high-flux computing is in Internet searches and web services, Emphasis: batch processing speed, cost, energy savings, security, and reliability.

6. Define peer-to-peer network.

The P2P architecture offers a distributed model of networked systems. Every node acts as both a client and a server, providing part of the system resources. Peer machines are simply client computers connected to the Internet. All client machines act autonomously to join or leave the system freely. This implies that no master-slave relationship exists among the peers. No central coordination or central database is needed.

7. What are the Three New Computing Paradigms Radio-frequency identification (RFID),
Global Positioning System (GPS),
Internet of Things (IoT).

8. What is degree of parallelism and types

The degree of parallelism (DOP) is a metric which indicates how many operations can be or are being simultaneously executed by a computer. It is especially useful for describing the performance of parallel programs and multi-processor systems.

- Bit-level parallelism (BLP)
- Instruction-level parallelism (ILP)
- VLIW (very long instruction word)
- Data-level parallelism (DLP)
- Multicore processors and chip multiprocessors (CMPs)
- Job-level parallelism (JLP)

9. What is Cyber-Physical Systems

A cyber-physical system (CPS) is the result of interaction between computational processes and the physical world. A CPS integrates ―cyber‖ (heterogeneous, asynchronous) with ―physical‖ (concurrent and information-dense) objects.

10. Define multi core CPU.

Advanced CPUs or microprocessor chips assume a multi-core architecture with dual, quad, six, or more processing cores. These processors exploit parallelism at ILP and TLP levels. CPU has reached its limit in terms of exploiting massive DLP due to the aforementioned memory wall problem

11. Define GPU.

A GPU is a graphics coprocessor or accelerator on a computer's graphics card or video card. A GPU offloads the CPU from tedious graphics tasks in video editing applications. The GPU chips can process a minimum of 10 million polygons per second. GPU's have a throughput architecture that exploits massive parallelism by executing many concurrent threads.

12. Clusters of Cooperative Computers

A computing cluster consists of interconnected stand-alone computers which work cooperatively as a single integrated computing resource.

13. W**hat is single-system image (SSI)**

An ideal cluster should merge multiple system images into a single-system image (SSI). Cluster designers desire a cluster operating system or some middleware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes.

14. What is Grid Computing

Grid computing is the collection of computer resources from multiple locations to reach a common goal. The grid can be thought of as a distributed system with non-interactive workloads that involve a large number of files. Grid computing is distinguished from conventional high performance computing systems such as cluster computing in that grid computers have each node set to perform a different task/application.

15. What is Computational Grids

A computing grid offers an infrastructure that couples computers, software, middleware, special instruments, and people and sensors together. The grid is often constructed across LAN, WAN, or Internet backbone networks at a regional, national, or global scale. Enterprises or organizations present grids as integrated computing resources.

16. What is Overlay Networks and its types

Overlay is a virtual network formed by mapping each physical machine with its ID, logically, through a virtual mapping. When a new peer joins the system, its peer ID is added as a node in the overlay network.

Two types of overlay networks:

1.Unstructured 2. Structured.

17. Write the any three Grid Applications.
• Schedulers
• Resource Broker
• Load Balancing

18. Difference between grid and cloud computing

Grid computing	**cloud computing**
• Grids enable access to shared computing power and storage capacity from your desktop	• Clouds enable access to leased computing power and storage capacity from your desktop.
• In computing centers distributed across different sites, countries and continents	• The cloud providers private data centers which are often centralized in a few locations with excellent network connections and cheap electrical power.
• Grids were designed to handle large sets of limited duration jobs that produce or use large quantities of data (e.g. the LHC)	• Clouds best support long term services and longer running jobs (E.g. facebook.com)

19. What are the derivatives of grid computing?
There are 8 derivatives of grid computing. They are as follows:
a)Compute grid
b) Data grid
c) Science grid
d) Access grid
e) Knowledge grid
f) Cluster grid
g) Terra grid
h) Commodity grid.
20. What is grid infrastructure?

Grid infrastructure forms the core foundation for successful grid applications. This infrastructure is a complex combination of number of capabilities and resources identified for the specific problem and environment being addressed.

21. What are the Applications of High-Performance and High-Throughput Systems

1.**Science and engineering**- Scientific simulations, genomic analysis, etc.Earthquake prediction, global warming, weather forecasting, etc.

2. **Business, education, services industry, and health care**- Telecommunication, content delivery, e-commerce, etc.Banking, stock exchanges, transaction processing, etc.Air traffic control, electric power grids, distance education, etc.Health care, hospital automation, telemedicine, etc

3. **Internet and web services, and government applications**- Internet search, data centers, decision-making systems, etc.Traffic monitoring, worm containment, cyber security, etc.Digital government, online tax return processing, social networking, etc.

22. What is Utility computing?

It is a service provisioning model in which a service provider makes computing resources and infrastructure management available to the customer as needed, and charges them for specific usage rather than a flat rate

23. What is SLA?

A service-level agreement (SLA) is a part of a standardized service contract where a service is formally defined. Particular aspects of the service – scope, quality, responsibilities – are agreed between the service provider and the service user. A common feature of an SLA is a contracted delivery time (of the service or performance)

24. Define Hype Cycle

This cycle shows the expectations for the technology at five different stages. The expectations rise sharply from the trigger period to a high peak of inflated expectations. Through a short period of disillusionment, the expectation may drop to a valley and then increase steadily over a long enlightenment period to a plateau of productivity. The number of years for an emerging technology to reach a certain stage is marked by special symbols.

25. Difference between CPU and GPU ?

A Graphics Processing Unit (GPU) is a special purpose processor, optimized or calculations commonly (and repeatedly) required for Computer Graphics, particularly SIMD operations.

A Central Processing Unit (CPU) is a general purpose processor - it can in principle do any computation, but not necessarily in an optimal fashion for any given computation. One can

do graphics processing on a CPU - but it likely will not produce the result anywhere nearly as fast as a properly programmed GPU.

26. What are the two most important technologies for building semantic webs?

XML

Resource Description Framework(RDF)

27. What is meant by Virtual Machines?

Virtual machines (VMs) offer novel solutions to underutilized resources, application inflexibility, and software manageability, and security concerns in existing physical machines.

28. What are the business benefits in Grid Computing?

· Acceleration of implementation time frames in order to intersect with the anticipated business end results.

· Improved productivity and collaboration of virtual organizations and respective computing and data resources.

· Allowing widely dispersed departments and business to create virtual organizations to share data and resources.

29. What are the areas are difficult to implement in Grid Computing Infrastructure?

A Grid computing infrastructure component must address several potentially complicated areas in many stages of the implementation. These areas are

- Security
- Resource management
- Information services
- Data management

30. Give the different layers of grid architecture.

- Fabric Layer: Interface to local resources
- Connectivity Layer: Manages Communications
- Collective Layer: Coordinating Multiple Resources
- Application Layer: User-Defined Application.

31. Define Moore's law?

Moore's law indicates that processor speed doubles every 18 months.

PART B

1.1 SCALABLE COMPUTING OVER THE INTERNET

Over the past 60 years, computing technology has undergone a series of platform and environment changes. Evolutionary changes in machine include architecture, operating system platform, network connectivity, and application workload. Instead of using a centralized computer to solve computational problems, a parallel and **distributed computing** system uses multiple computers to solve large-scale problems over the Internet. Thus, distributed computing becomes **data-intensive and network-centric.**

- The Age of Internet computing

        - The platform evolution
        - High performance computing
        - High throughput computing
        - Three new computing paradigm
        - Computing paradigm distinction
        - Distributed system families
    - Scalable computing trends and new paradigms
        - Degrees of parallelism
        - Innovative applications
        - The trend towards utility computing
        - The hype cycle of new technologies
    - The internet of things and cyber physical system
        - The internet of things
        - Cyber physical systems

1.1.1 The Age of Internet Computing

- Billions of people use the Internet every day.
- Supercomputer sites and large data centers must provide high-performance computing services to huge numbers of Internet users concurrently.
- The Linpack Benchmark for high-performance computing (HPC) applications is no longer optimal for measuring system performance.
- The emergence of computing clouds instead demands high-throughput computing (HTC) systems built with parallel and distributed computing technologies.
- The purpose is to advance network-based computing and web services with the emerging new technologies.

The Platform Evolution

- Computer technology has gone through five generations of development, with each generation lasting from 10 to 20 years.
- 1950 to 1970 – mainframes – eg: IBM 360 and CDC 6400.
- 1960 to 1980 - lower-cost mini-computers – eg: DEC PDP 11 and VAX Series .
- 1970 to 1990 - personal computers built with VLSI microprocessors.
- 1980 to 2000 - portable computers
- Since 1990, the use of both HPC and HTC systems hidden in clusters, grids, or Internet clouds has proliferated.
- These systems are employed by both consumers and high-end web-scale computing and information services.
- The general computing trend is to leverage shared web resources and massive amounts of data over the Internet.
- On the HPC side, supercomputers (massively parallel processorsor MPPs) are gradually replaced by clusters of cooperative computers out of a desire to share computing resources.
- The cluster is often a collection of homogeneous compute nodes that are physically connected in close range to one another.

- On the HTC side, peer-to-peer (P2P) networks are formed for distributed file sharing and content delivery applications.
- A P2P system is built over many client machines.
- Peer machines are globally distributed in nature.

High-Performance Computing

- For many years, HPC systems emphasize the raw speed performance.
- The speed of HPC systems has increased from Gflops in the early 1990s to now Pflops in 2010.
- This improvement was driven mainly by the demands from scientific, engineering, and manufacturing communities.

High-Throughput Computing

- The development of market-oriented high-end computing systems is undergoing a strategic change from an HPC paradigm to an HTC paradigm.
- The main application for high-flux computing is in Internet searches and web services by millions or more users simultaneously.
- The performance goal thus shifts to measure high throughput or the number of tasks completed per unit of time.
- HTC technology needs to not only improve in terms of batch processing speed, but also address the acute problems of cost, energy savings, security, and reliability at many data and enterprise computing centers.

Three New Computing Paradigms

With the introduction of SOA, Web 2.0 services become available.

Advances in virtualization make it possible to see the growth of Internet clouds as a new computing paradigm.

Three new computing paradigms:

- The maturity of radio-frequency identification (RFID), Global Positioning System (GPS) , and sensor technologies has triggered the development of the Internet of Things (IoT).
- In 1984, John Gage of Sun Microsystems created the slogan, “The net-work is the computer.” In 2008, David Patterson of UC Berkeley said, “The data center is the computer. There are dramatic differences between developing software for millions to use as a service versus distributing software to run on their PCs.”
- Recently, Rajkumar Buyya of Melbourne University simply said: “The cloud is the computer.”

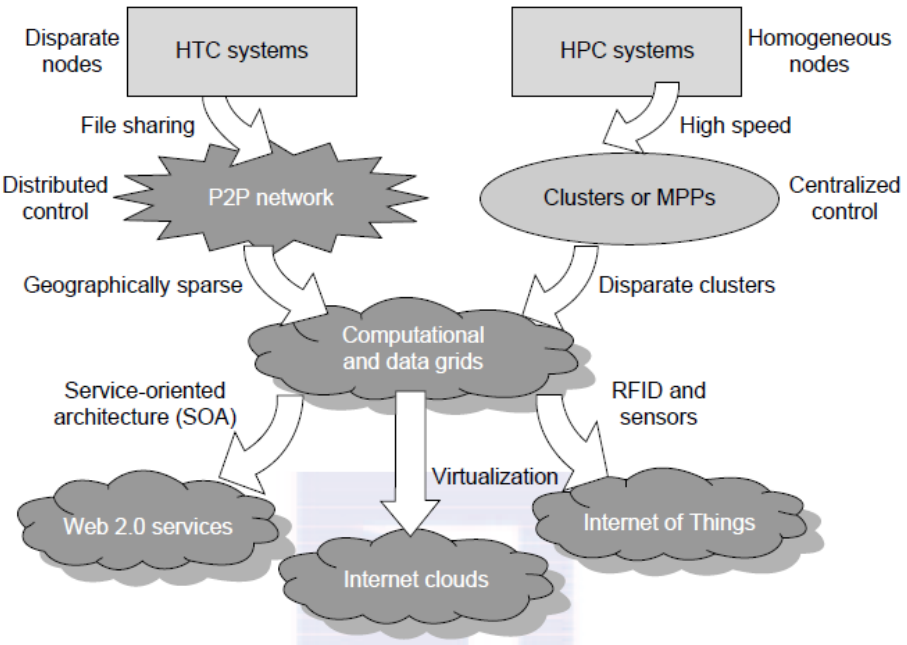


FIGURE 1.1

Evolutionary trend toward parallel, distributed, and cloud computing with clusters, MPPs, P2P networks, grids, clouds, web services, and the Internet of Things.

Computing Paradigm Distinction

•**Centralized computing** - The computer resources are centralized in one physical system. All resources (processors, memory, and storage) are fully shared and tightly coupled within one integrated OS.

• **Parallel computing** - In parallel computing, all processors are either tightly coupled with centralized shared memory or loosely coupled with distributed memory. Interprocessor communication is accomplished through shared memory or via message passing. A computer system capable of parallel computing is commonly known as a parallel computer Programs running in a parallel computer are called parallel programs. The process of writing parallel programs is often referred to as parallel programming.

• **Distributed computing** - It consists of multiple autonomous computers, each having its own private memory, communicating through a computer network. Information exchange in a distributed system is accomplished through message passing. A computer program that runs in a distributed system is known as a distributed program. The process of writing distributed programs is referred to as distributed programming.

• **Cloud computing** - An Internet cloud of resources can be either a centralized or a distributed computing system. The cloud applies parallel or distributed computing, or both. Clouds can be built with physical or virtualized resources over large data centers that are centralized or distributed. Ubiquitous computing refers to computing with pervasive devices at any place and time using wired or wireless communication. The Internet of Things (IoT) is a networked connection of everyday objects including computers, sensors,

humans, etc. The IoT is supported by Internet clouds to achieve ubiquitous computing with any object at any place and time.

Distributed System Families

- Since the mid-1990s, technologies for building P2P networks and networks of clusters have been consolidated into many national projects designed to establish wide area computing infrastructures, known as computational grids or data grids.
- Massively distributed systems are intended to exploit a high degree of parallelism or concurrency among many machines.
- In October 2010, the highest performing cluster machine was built in China with 86016 CPU processor cores and 3,211,264 GPU cores in a Tianhe-1A system.
- The largest computational grid connects up to hundreds of server clusters.
- A typical P2P network may involve millions of client machines working simultaneously. Experimental cloud computing clusters have been built with thousands of processing nodes.
- In the future, both HPC and HTC systems will demand multicore or many-core processors that can handle large numbers of computing threads per core.
- Both HPC and HTC systems emphasize parallelism and distributed computing.
- The system efficiency is decided by speed, programming, and energy factors (i.e., throughput per watt of energy consumed).
- Meeting these goals requires to yield the following design objectives:
- Efficiency
- Dependability.
- Adaptation in the programming model
- Flexibility in application deployment

1.1.2 Scalable Computing Trends and New Paradigm

Degrees of Parallelism

- Bit-level parallelism (BLP) converts bit-serial processing to word-level processing gradually.
- Over the years, users graduated from 4-bit microprocessors to 8-, 16-, 32-, and 64-bit CPUs. The instruction-level parallelism (ILP), in which the processor executes multiple instructions simultaneously rather than only one instruction at a time.
- For the past 30 years, we have practiced ILP through pipelining, super-scalar computing, VLIW (very long instruction word) architectures, and multithreading.
- ILP requires branch prediction, dynamic scheduling, speculation, and compiler support to work efficiently.
- Data-level parallelism (DLP) was made popular through SIMD (single instruction, multiple data) and vector machines using vector or array types of instructions.
- Ever since the introduction of multicore processors and chip multiprocessors (CMPs), we have been exploring task-level parallelism (TLP) .
- A modern processor explores all of the aforementioned parallelism types.

Innovative Applications

- Both HPC and HTC systems desire transparency in many application aspects.
- For example, data access, resource allocation, process location, concurrency in execution, job replication, and failure recovery should be made transparent to both users and system management.

- Users must deal with multiple database servers in distributed transactions. Maintaining the consistency of replicated transaction records is crucial in real-time banking services.
- Other complications include lack of software support, network saturation, and security threats in these applications.

Table 1.1 Applications of High-Performance and High-Throughput Systems

Domain	Specific Applications
Science and engineering	Scientific simulations, genomic analysis, etc.
	Earthquake prediction, global warming, weather forecasting, etc.
Business, education, services industry, and health care	Telecommunication, content delivery, e-commerce, etc.
	Banking, stock exchanges, transaction processing, etc.
	Air traffic control, electric power grids, distance education, etc.
	Health care, hospital automation, telemedicine, etc.
Internet and web services, and government applications	Internet search, data centers, decision-making systems, etc.
	Traffic monitoring, worm containment, cyber security, etc.
	Digital government, online tax return processing, social networking, etc.
Mission-critical applications	Military command and control, intelligent systems, crisis management, etc.

The Trend toward Utility Computing

Characteristics:

- Ubiquitous
- Reliability
- Scalability
- Autonomic
- Self-organized to support dynamic discovery.
- Composable with QoS and SLAs (service-level agreements).
- Utility vision - Utility computing focuses on a business model in which customers receive computing resources from a paid service provider. All grid/cloud platforms are regarded as utility service providers.

The **Hype Cycle** of New Technologies

1. Hype cycle copyrighted 2010 by Garther.
2. Hype cycle is the graphical representation of the relative maturity of technologies, IT methodologies and management disciplines.

- Any new and emerging computing and information technology may go through a hype cycle, as this cycle shows the expectations for the technology at five different stages.The expectations rise sharply from the trigger period to a high peak of inflated expectations.

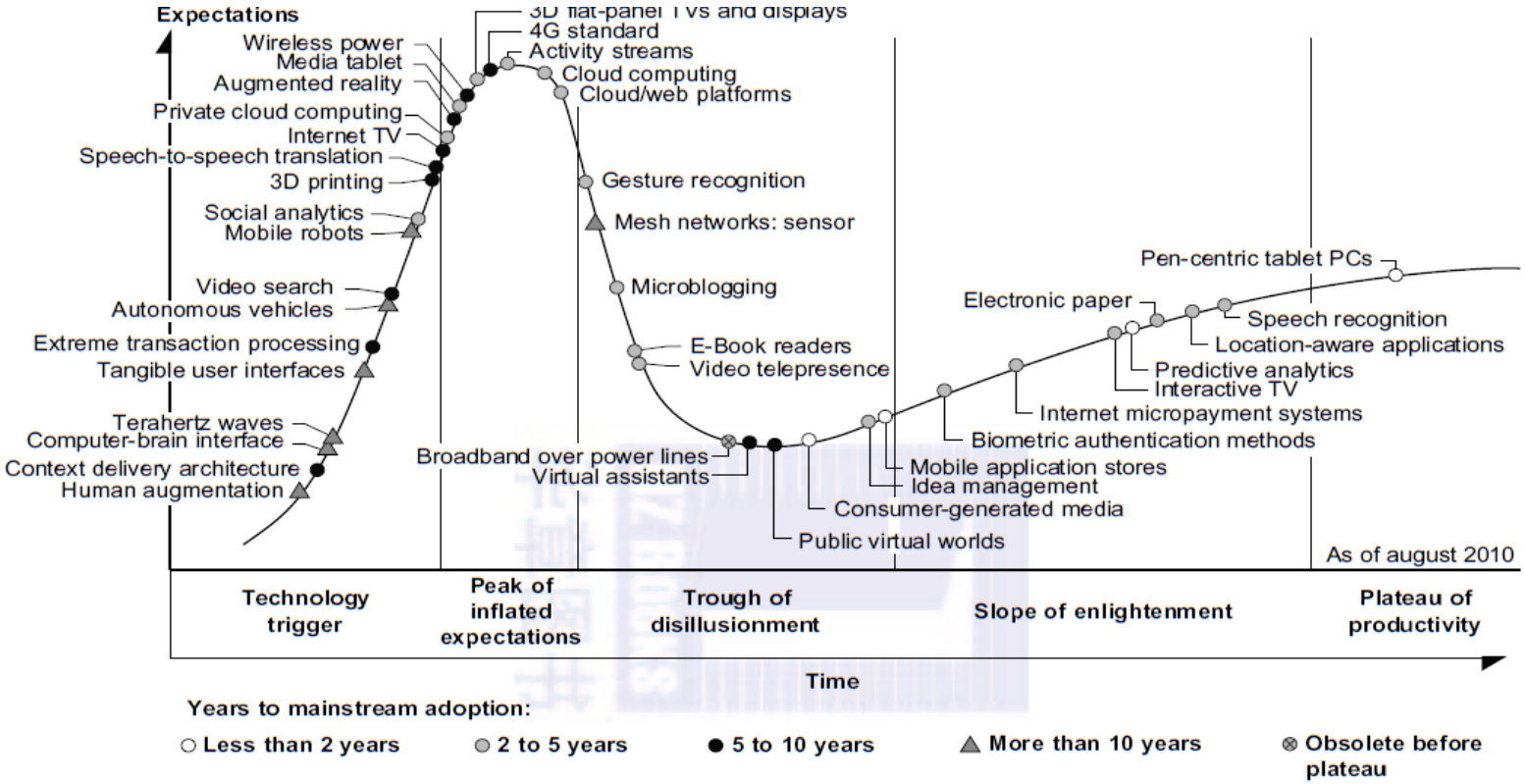


FIGURE 1.3

Hype cycle for Emerging Technologies, 2010.

(*Source: Gartner Press Release "Gartner's 2010 Hype Cycle Special Report Evaluates Maturity of 1,800 Technologies" 7 October 2010.*)

- Through a short period of disillusionment, the expectation may drop to a valley and then increase steadily over a long enlightenment period to a plateau of productivity.
- The number of years for an emerging technology to reach a certain stage is marked by special symbols.
- The hollow circles - indicate technologies that will reach mainstream adoption in two years.
- The gray circles - represent technologies that will reach mainstream adoption in two to five years.
- The solid circles represent those that require five to 10 years to reach mainstream adoption, and the triangles denote those that require more than 10 years.
- The crossed circles represent technologies that will become obsolete before they reach the plateau.

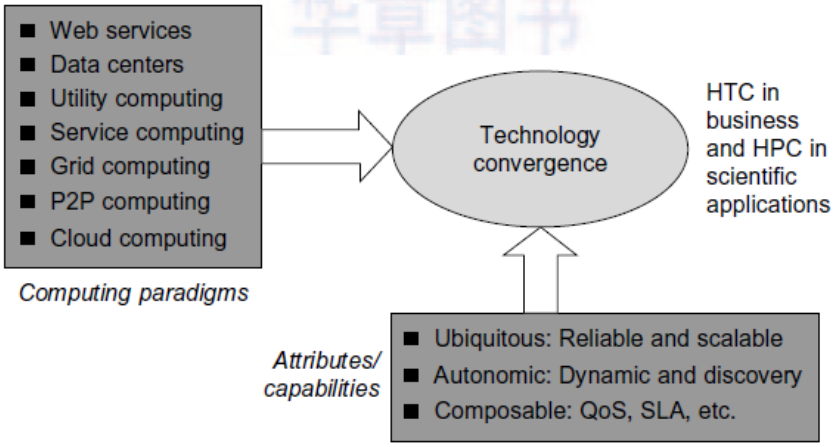


FIGURE 1.2

The vision of computer utilities in modern distributed computing systems.

1.1.3 **The Internet of Things and Cyber-Physical Systems**

These evolutionary trends emphasize the extension of the Internet to every-day objects.

The Internet of Things

- The traditional Internet connects machines to machines or web pages to web pages The idea is to tag every object using RFID or a related sensor or electronic technology such as GPS.
- With the introduction of the IPv6 protocol, 2^{128} IP addresses are available to distinguish all the objects on Earth, including all computers and pervasive devices.
- The IoT researchers have estimated that every human being will be surrounded by 1,000 to 5,000 objects.
- The IoT demands universal addressability of all of the objects or things.
- The IoT obviously extends the Internet and is more heavily developed in Asia and European countries.
- In the IoT era, all objects and devices are instrumented, interconnected, and interacted with each other intelligently.

- This communication can be made between people and things or among the things themselves. Three communication patterns co-exist: namely H2H (human-to-human), H2T (human-to-thing), and T2T (thing-to-thing). Here things include machines such as PCs and mobile phones.
- The idea here is to connect things (including human and machine objects) at any time and any place intelligently with low cost. Any place connections include at the PC, indoor (away from PC), outdoors, and on themove. Any time connections include daytime, night, outdoors and indoors, and on the move as well.
- The dynamic connections will grow exponentially into a new dynamic network of networks, called the Internet of Things (IoT). The IoT is still in its infancy stage of development. Many prototype IoTs with restricted areas of coverage are under experimentation at the time of this writing.
- A smart Earth should have intelligent cities, clean water, efficient power, convenient transportation, good food supplies, responsible banks, fast telecommunications, green IT, better schools, good health care, abundant resources, and so on. This dream living environment may take some time to reach fruition at different parts of the world.

Cyber-Physical Systems

- A cyber-physical system (CPS) is the result of interaction between computational processes and the physical world. A CPS integrates "cyber" (heterogeneous, asynchronous) with "physical" (concurrent and information-dense) objects.
- A CPS merges the **"3C" technologies** of computation, communication, and control into an intelligent closed feedback system between the physical world and the information world, a concept which is actively explored in the United States.

1.2 TECHNOLOGIES FOR NETWORK-BASED SYSTEMS

- Multicore CPUs and multithreading technologies
  - Advances in CPU processors
  - Multicore CPU and manycore GPU architectures
  - Multithreading technology
- GPU computing to exascale and beyond
  - How GPUs work
  - GPU programming model
  - Power efficiency of GPU
- Memory, storage and Wide area networking
  - Memory technology
  - Disk and storage technology
  - System area interconnect
  - Wide area networking
- Virtual machines and virtualization middleware
  - Virtual machines
  - VM primitive operations
  - Virtual infrastructure
- Data center virtualization for cloud computing
  - Data center growth and cost breakdown

- Low cost design philosophy
- Convergence of technologies

1.2.1 Multicore CPUs and Multithreading Technologies

Advances in CPU Processors

- A multicore architecture - with dual, quad, six, or more processing cores.
- These processors exploit parallelism at ILP and TLP levels.
- Processor speed growth is plotted in the upper curve in across generations of microprocessors or CMPs.
- 1 MIPS - for the VAX 780 in 1978
- 1,800 MIPS - for the Intel Pentium 4 in 2002,
- 22,000 MIPS - the Sun Niagara 2 in 2008.
- As the figure shows, Moore law has proven to be pretty accurate in this case.
- The clock rate for these processors:
- 10 MHz for the Intel 286
- 4 GHz for the Pentium 4
- However, the clock rate reached its limit on CMO S-based chips due to power limitations.
- The clock rate will not continue to improve unless chip technology matures.
- This limitation is attributed primarily to excessive heat generation with high frequency or high voltages.
- Both multi-core CPU and many-core GPU processors can handle multiple instruction threads at different magnitudes today.
- Each core is essentially a processor with its own private cache (L1 cache).
- Multiple cores are housed in the same chip with an L2 cache that is shared by all cores. In the future, multiple CMPs could be built on the same CPU chip with even the L3 cache on the chip.
- Multicore and multi- threaded CPUs are equipped with many high-end processors, including the Intel i7, Xeon, AMD Opteron, Sun Niagara, IBM Power 6, and X cell processors. Each core could be also multithreaded.
- For example, the Niagara II is built with eight cores with eight threads handled by each core.

Multicore CPU and Many-Core GPU Architectures

- Multicore CPUs may increase from the tens of cores to hundreds or more in the future.
- But the CPU has reached its limit in terms of exploiting massive DLP due to the memory wall problem.
- A many-core GPUs have with hundreds or more thin cores.

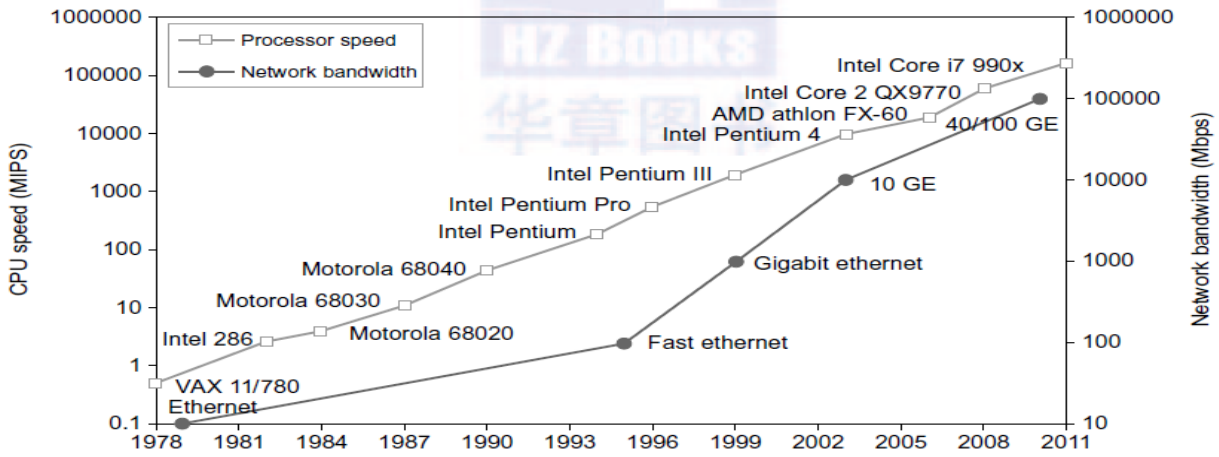


FIGURE 1.4

Improvement in processor and network technologies over 33 years.

- Now, x-86 processors have been extended to serve HPC and HTC systems in some high-end server processors.
- Many RISC processors have been replaced with multicore x-86 processors and many-core GPUs in the Top 500 systems.
- This trend indicates that x-86 upgrades will dominate in data centers and supercomputers. The GPU also has been applied in large clusters to build supercomputers in MPPs.

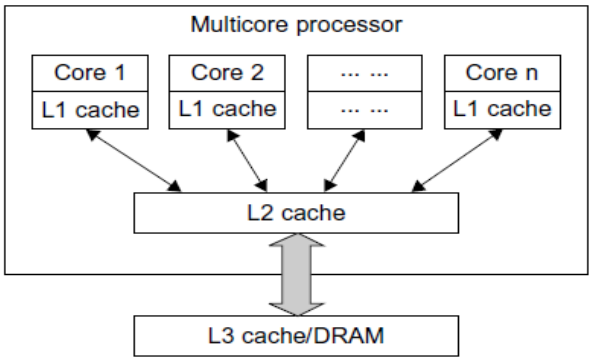


FIGURE 1.5

Schematic of a modern multicore CPU chip using a hierarchy of caches, where L1 cache is private to each core, on-chip L2 cache is shared and L3 cache or DRAM Is off the chip.

Multithreading Technology

- Types of processor : four-issue superscalar processor, a fine-grain multithreaded processor, a coarse-grain multi- threaded processor, a two-core CMP, and a simultaneous multithreaded (SMT) processor.
- The superscalar processor is single-threaded with four functional units. Each of the three multithreaded processors is four-way multithreaded over four functional data paths.

- In the dual-core processor, assume two processing cores, each a single-threaded two-way superscalar processor. Instructions from different threads are distinguished by specific shading patterns for instructions from five independent threads.
- Fine-grain multithreading switches the execution of instructions from different threads per cycle.
- Course-grain multi-threading executes many instructions from the same thread for quite a few cycles before switching to another thread.
- The multicore CMP executes instructions from different threads completely. These execution patterns closely mimic an ordinary program.
- The blank squares correspond to no available instructions for an instruction data path at a particular processor cycle.

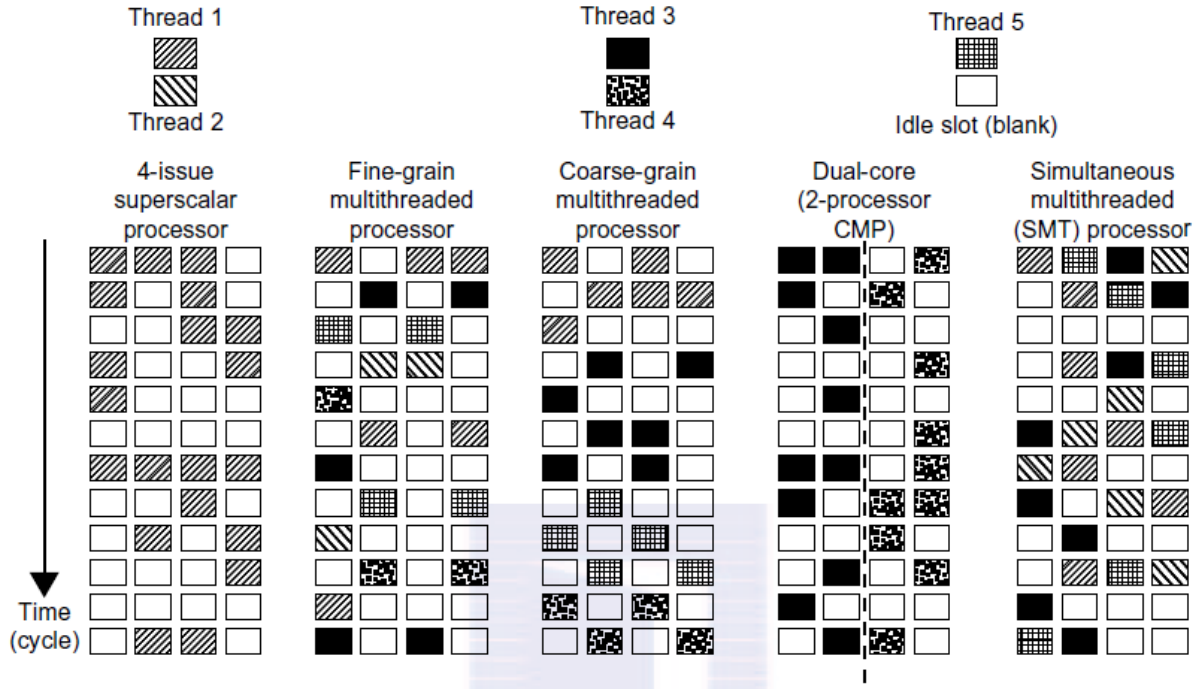


FIGURE 1.6

Five micro-architectures in modern CPU processors, that exploit ILP and TLP supported by multicore and multithreading technologies.

1.2.2 GPU Computing to Exascale and Beyond

- A GPU is a graphics coprocessor or accelerator mounted on a computer's graphics card or video card.
- A GPU offloads the CPU from tedious graphics tasks in video editing applications.
- The world's first GPU, the GeForce 256, was marketed by NVIDIA in 1999.
- These GPU chips can process a minimum of 10 million polygons per second.
- Traditional CPUs are structured with only a few cores.
- For example, the Xeon X5670 CPU has six cores., a modern GPU chip can be built with hundreds of processing cores.

- GPUs have a throughput architecture that exploits massive parallelism by executing many concurrent threads slowly, instead of executing a single long thread in a conventional microprocessor very quickly.
- General-purpose computing on GPUs , known as GPGPUs, have appeared in the HPC field. NVIDIA' s CUDA model was for HPC using GPGPU.

How GPUs Work

- Early GPUs functioned as coprocessors attached to the CPU.
- Today, the NVIDIA GPU has been upgraded to 128 cores on a single chip.
- Furthermore, each core on a GPU can handle eight threads of instructions.
- This translates to having up to 1 ,024 threads executed concurrently on a single GPU.
- Modern GPUs are not restricted to accelerated graphics or video coding.
- They are used in HPC systems to power supercomputers with massive parallelism at multicore and multithreading levels.
- GPUs are designed to handle large numbers of floating-point operations in parallel.
- In a way, the GPU offloads the CPU from all data-intensive calculations.
- GPU are widely used in mobile phones, game consoles, embedded systems, PCs, and servers.
- The NVIDIA CUDA Tesla or Fermi is used in GPU clusters or in HPC systems for parallel processing of massive floating-pointing data.

GPU Programming Model

- The interaction between a CPU and GPU for performing parallel execution offloating-point operations concurrently.
- The CPU is the conventional multicore processor with limited parallelism to exploit.
- The GPU has a many-core architecture that has hundreds of simple processing cores organized as multiprocessors.

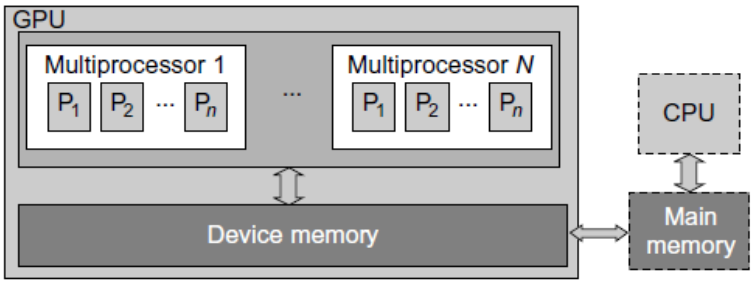


FIGURE 1.7

The use of a GPU along with a CPU for massively parallel execution in hundreds or thousands of processing cores.

- Each core can have one or more threads.

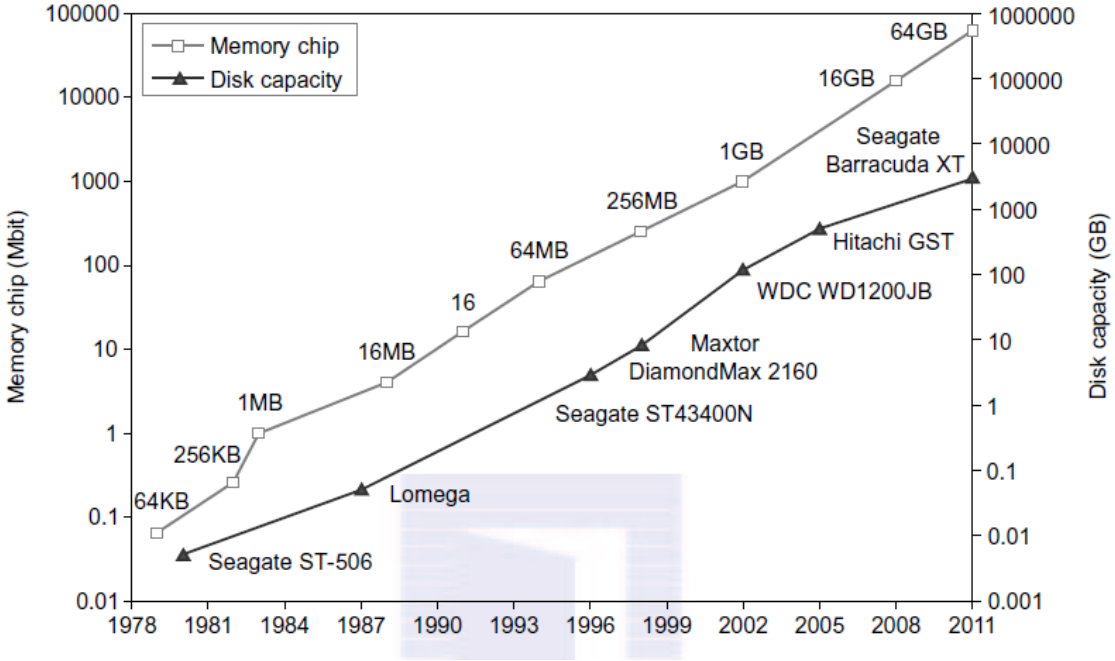
- Essentially, the CPU's floating-point kernel computation role is largely offloaded to the many-core GPU.
- The CPU instructs the GPU to perform massive data processing.
- The bandwidth must be matched between the on-board main memory and the on-chip GPU memory.
- This process is carried out in NVIDIA CUDA programming using the GeForce 8800 or Tesla and Fermi GPUs.
- In a DARPA report published in September 2008, four challenges are identified for exascale computing: (1) energy and power,
  (2) memory and storage,
  (3) concurrency and locality, and
  (4) system resiliency.

Power Efficiency of the GPU

- Bill Dally of Stanford University considers power and massive parallelism as the major benefits of GPUs over CPUs for the future.
- By extrapolating current technology and computer architecture, it was estimated that 60 Gflops/watt per core is needed to run an exaflops system (see Figure 1.10).
- Power constrains what we can put in a CPU or GPU chip.
- Dally has estimated that the CPU chip consumes about 2 nJ/instruction, while the GPU chip requires 200 pJ/instruction, which is 1/10 less than that of the CPU. 1.2.3 Memory, Storage, and Wide-Area Networking

Memory Technology

- The upper curve in plots the growth of DRAM chip capacity from 16 KB in 1976 to 64 GB in 2011.
- This shows that memory chips have experienced a 4x increase in capacity every three years.
- Memory access time did not improve much in the past.
- For hard drives,
- capacity increased from 260 MB in 1981 to 250 GB in 2004.
- The Seagate Barracuda XT hard drive reached 3 TB in 2011.
- This represents an approximately 10x increase in capacity every eight years.
- Faster processor speed and larger memory capacity result in a wider gap between
- processors and memory. The memory wall may become even worse a problem limiting the CPU performance in the future.



Disks and Storage Technology

- Beyond 2011, disks or disk arrays have exceeded 3 TB in capacity.
- The lower curve in shows the disk storage growth in 7 orders of magnitude in 33 years.
- The rapid growth of flash memory and solid-state drives (SSDs) also impacts the future of HPC and HTC systems.
- The mortality rate of SSD is not bad at all.
- A typical SSD can handle 300,000 to 1 million write cycles per block.
- Flash and SSD will demonstrate impressive speedups in many applications.
- Eventually, power consumption, cooling, and packaging will limit large system development.
- Power increases linearly with respect to clock frequency and quadratic ally with respect to voltage applied on chips.
- Clock rate cannot be increased indefinitely.

System-Area Interconnects

- The nodes in small clusters are mostly interconnected by an Ethernet switch or a local area network (LAN)., a LAN typically is used to connect client hosts to big servers.
- A storage area network (SAN) connects servers to network storage such as disk arrays. Network attached storage (NAS) connects client hosts directly to the disk arrays.
- All three types of networks often appear in a large cluster built with commercial network components.
- If no large distributed storage is shared, a small cluster could be built with a multiport Gigabit Ethernet switch plus copper cables to link the end machines.

Wide-Area Networking

- The lower curve plots the rapid growth of Ethernet bandwidth from 10 Mbps in 1979 to 1 Gbps in 1999, and 40 ~ 100 GE in 2011.
- It has been speculated that 1 Tbps network links will become available by 2013.
- According to Berman, Fox, and Hey, network links with 1,000, 1,000, 100, 10, and 1 Gbps bandwidths were reported, respectively, for international, national, organization, optical desktop, and copper desktop connections in 2006. An increase factor of two per year on network performance was reported, which is faster than Moore's law on CPU speed doubling every 18 months.
- The IDC 2010 report predicted that both InfiniBand and Ethernet will be the two major interconnect choices in the HPC arena.

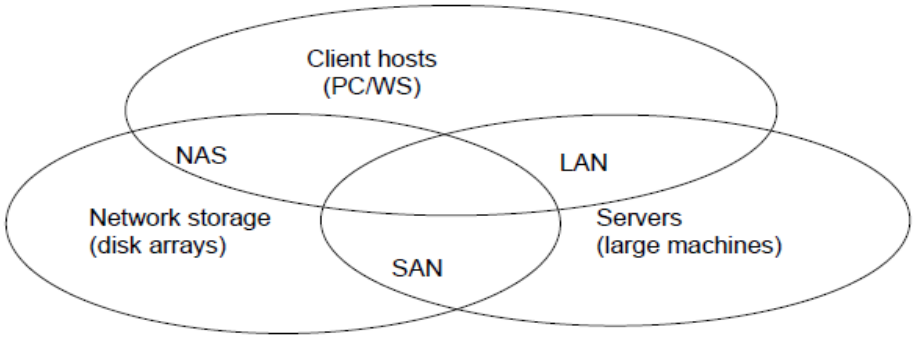


FIGURE 1.11

Three interconnection networks for connecting servers, client hosts, and storage devices; the LAN connects client hosts and servers, the SAN connects servers with disk arrays, and the NAS connects clients with large storage systems in the network environment.

1.2.4 Virtual Machines and Virtualization Middleware

- A conventional computer has a single OS image.
- This offers a rigid architecture that tightly couples application software to a specific hardware platform.
- Some software running well on one machine may not be executable on another platform with a different instruction set under a fixed OS.
- Virtual machines (VMs) offer novel solutions to underutilized resources, application inflexibility, software manageability, and security concerns in existing physical machines. Today, to build large clusters, grids, and clouds, we need to access large amounts of computing, storage, and networking resources in a virtualized manner.
- In particular, a cloud of provisioned resources must rely on virtualization of processors, memory, and I/O facilities dynamically.

Virtual Machines

- In the host machine is equipped with the physical hardware, as shown at the bottom of the figure.
- An example is an x-86 architecture desktop running its installed Windows OS, as shown in part (a) of the figure.
- The VM can be provisioned for any hardware system.
- The VM is built with virtual resources managed by a guest OS to run a specific application. Between the VMs and the host platform, one needs to deploy a middleware layer called a virtual machine monitor (VMM) .
- Figure 1.12(b) shows a native VM installed with the use of a VMM called a hypervisor in privileged mode.
- This hypervisor approach is also called bare-metal VM , because the hypervisor handles the bare hardware (CPU, memory, and I/O) directly.
- Another architecture is the host VM shown in Figure 1.12(c). Here the VMM runs in non privileged mode.
- The host OS need not be modified.
- The VM can also be implemented with a dual mode, as shown in Figure 1.12(d).
- Part of the VMM runs at the user level and another part runs at the supervisor level. In this case, the host OS may have to be modified to some extent.
- Multiple VMs can be ported to a given hardware system to support the virtualization process.

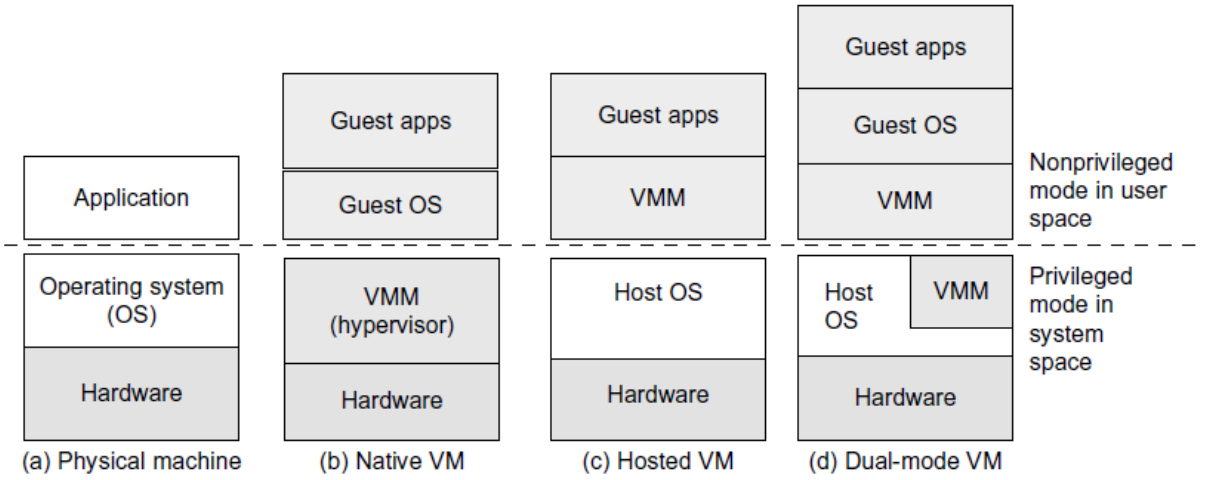


FIGURE 1.12

Three VM architectures in (b), (c), and (d), compared with the traditional physical machine shown in (a).

(Courtesy of M. Abde-Majeed and S. Kulkarni, 2009 USC)

VM Primitive Operations

- The VMM provides the VM abstraction to the guest OS. With full virtualization, the VMM exports a VM abstraction identical to the physical machine so that a standard OS such as Windows 2000 or Linux can run just as it would on the physical hardware.

- First, the VMs can be multiplexed between hardware machines, as shown in Figure 1.13(a).
- Second, a VM can be suspended and stored in stable storage, as shown in Figure 1.13(b).
- Third, a suspended VM can be resumed or provisioned to a new hardware platform, as shown in Figure 1.13(c).
- Finally, a VM can be migrated from one hardware platform to another, as shown in Figure 1.13(d).

Virtual Infrastructures

- Physical resources for compute, storage, and networking at the bottom of Figure 1.14 are mapped to the needy applications embedded in various VMs at the top.
- Hardware and software are then separated. Virtual infrastructure is what connects resources to distributed applications.
- It is a dynamic mapping of system resources to specific applications.
- The result is decreased costs and increased efficiency and responsiveness.

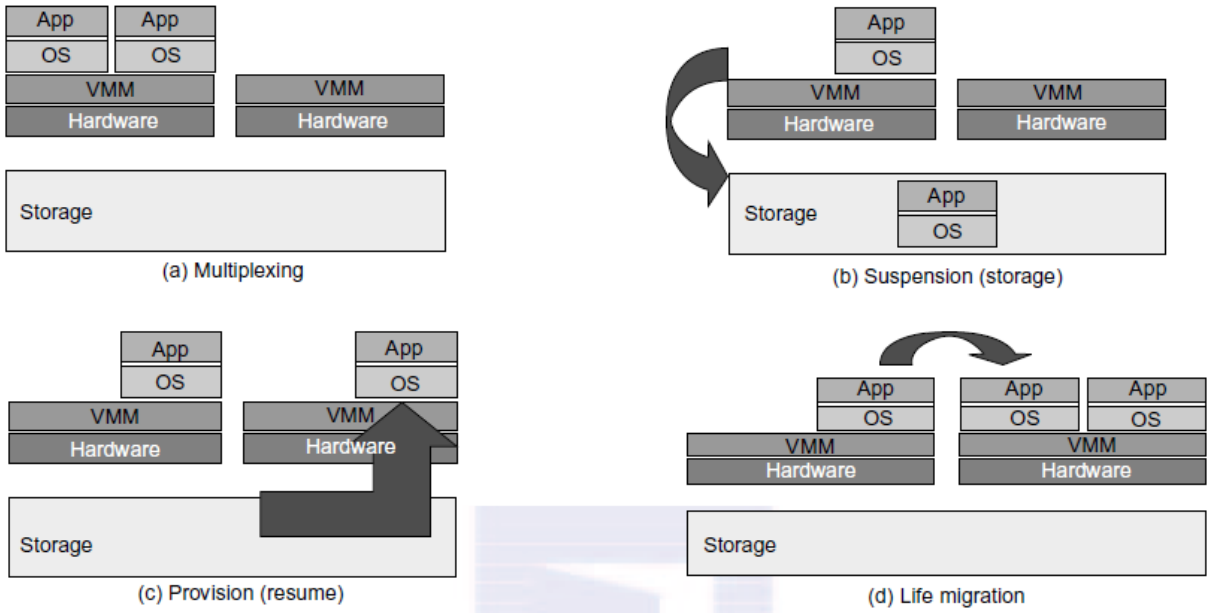


FIGURE 1.13

VM multiplexing, suspension, provision, and migration in a distributed computing environment.

1.2.5 Data Center Virtualization for Cloud Computing

- Cloud architecture is built with commodity hardware and network devices.
- Low-cost terabyte disks and Gigabit Ethernet are used to build data centers.
- Data center design emphasizes the performance/price ratio over speed performance alone. In other words, storage and energy efficiency are more important than shear speed performance.

- Figure 1.13 shows the server growth and cost breakdown of data centers over the past 15 years.
- Worldwide, about 43 million servers are in use as of 2010. The cost of utilities exceeds the cost of hardware after three years.

Data Center Growth and Cost Breakdown

- A large data center may be built with thousands of servers. Smaller data centers are typically built with hundreds of servers. The cost to build and maintain data center servers has increased over the years. According to a 2009 IDC report, typically only 30 percent of data center costs goes toward purchasing IT equipment, 33 percent is attributed to the chiller, 18 percent to the uninterruptible power supply (UPS) , 9 percent to computer room air conditioning (CRAC) , and the remaining 7 percent to power distribution, lighting, and transformer costs. Thus, about 60 percent of the cost to run a data center is allocated to management and maintenance. The server purchase cost did not increase much with time. The cost of electricity and cooling did increase from 5 percent to 14 percent in 15 years.

Convergence of Technologies

Essentially, cloud computing is enabled by the convergence of technologies in four areas:
(1) hard- ware virtualization and multi-core chips,
(2) utility and grid computing,
(3) SOA, Web 2.0, and WS mashups,
(4) atonomic computing and data center automation.

- With science becoming data-centric, a new paradigm of scientific discovery is becoming based on data-intensive technologies.
- On January 11, 2007, the Computer Science and Telecommunication Board (CSTB) recommended fostering tools for data capture, data creation, and data analysis. A cycle of interaction exists among four technical areas.

1.3 CLUSTERS OF COOPERATIVE COMPUTERS

  - Cluster architecture
  - SSI
  - Hardware, software and middleware support
  - Major cluster design issues

- Distributed and cloud computing systems are built over a large number of autonomous computer nodes.
- These node machines are interconnected by SANs, LANs, or WANs in a hierarchical manner.
- A WAN can connect many local clusters to form a very large cluster of clusters.
- Massive systems are considered highly scalable, and can reach web-scale connectivity, either physically or logically.
- A massive system is classified into four groups:
- Clusters
- P2Pnetworks
- Computing grids

- Internet clouds
- In terms of node number, these four system classes may involve hundreds, thousands, or even millions of computers as participating nodes.
- These machines work collectively, cooperatively, or collaboratively at various levels.
- From the application perspective, clusters are most popular in supercomputing applications.
- P2P networks appeal most to business applications. However, the content industry was reluctant to accept P2P technology for lack of copyright protection in ad hoc networks. Potential advantages of cloud computing include its low cost and simplicity for both providers and users.

Clusters of Cooperative Computers

- A computing cluster consists of interconnected stand-alone computers
- Work cooperatively as a single integrated computing resource.

1.3.1Cluster Architecture

- The architecture of a typical server cluster built around a low-latency, high-bandwidth interconnection network.
- This network can be as simple as a SAN (e.g., Myrinet) or a LAN (e.g., Ethernet).
- To build a larger cluster with more nodes, the interconnection network can be built with multiple levels of Gigabit Ethernet, Myrinet, or InfiniBand switches.
- Through hierarchical construction using a SAN, LAN, or WAN, one can build scalable clusters with an increasing number of nodes.
- The cluster is connected to the Internet via a virtual private network (VPN) gateway.
- The gateway IP address locates the cluster.
- The system image of a computer is decided by the way the OS manages the shared cluster resources.
- Most clusters have loosely coupled node computers.
- All resources of a server node are managed by their own OS.
- Thus, most clusters have multiple system images as a result of having many autonomous nodes under different OS control.

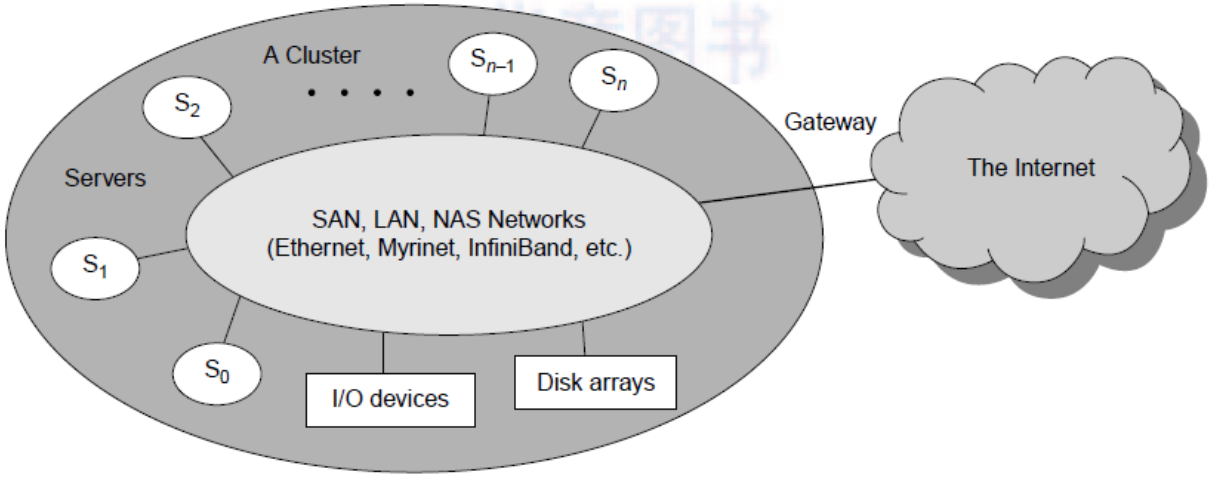


FIGURE 1.15

A cluster of servers interconnected by a high-bandwidth SAN or LAN with shared I/O devices and disk arrays; the cluster acts as a single computer attached to the Internet.

1.3.2Single-System Image

Cluster designers desire a cluster operating system or some middle-ware to support SSI at various levels, including the sharing of CPUs, memory, and I/O across all cluster nodes.

An SSI is an illusion created by software or hardware that presents a collection of resources as one integrated, powerful resource.

SSI makes the cluster appear like a single machine to the user. A cluster with multiple system images is nothing but a collection of independent computers.

1.3.3 Hardware, Software, and Middleware Support

- Almost all HPC clusters in the Top 500 list are also MPPs. The building blocks are computer nodes (PCs, workstations, servers, orSMP), special communication software such as PVM or MPI, and a network interface card in each computer node.
- Most clusters run under the Linux OS.
- The computer nodes are interconnected by a high-bandwidth network (such as Gigabit Ethernet, Myrinet, InfiniBand, etc.).
- Special cluster middleware supports are needed to create SSI or high availability (HA).
- Both sequential and parallel applications can run on the cluster, and special parallel environments are needed to facilitate use of the cluster resources.
- For example, distributed memory has multiple images.
- Users may want all distributed memory to be shared by all servers by forming distributed shared memory (DSM).

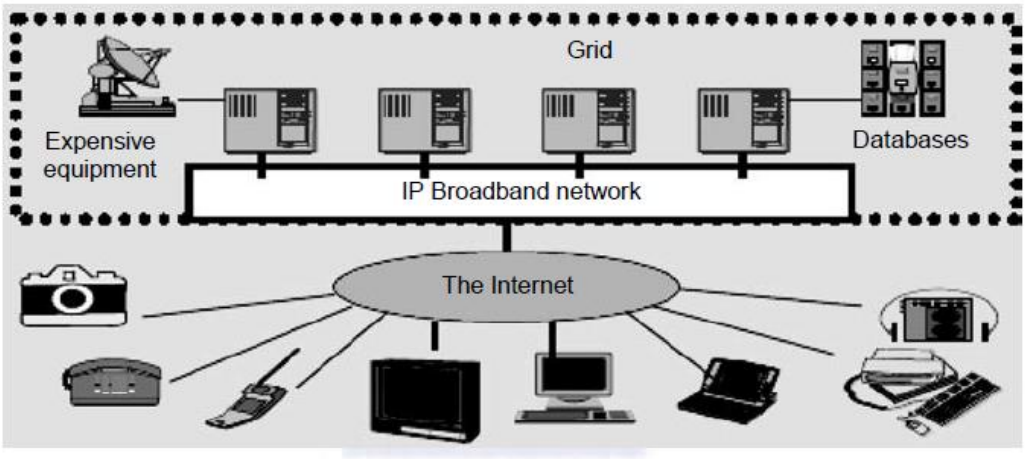
1.3.4 Major Cluster Design Issues

- Unfortunately, a cluster-wide OS for complete resource sharing is not available yet. Middleware or OS extensions were developed at the user space to achieve SSI at selected functional levels. Without this middleware, cluster nodes cannot work together effectively to achieve cooperative computing.

1.4 Grid Computing Infrastructures

- ***Computational grid***
- ***Grid families***

In the past 30 years, users have experienced a natural growth path from Internet to web and grid computing services.
Internet services such as the Telnet command enables a local computer to connect to a remote computer.
A web service such as HTTP enables remote access of remote web pages.
Grid computing is envisioned to allow close interaction among applications running on distant computers simultaneously.



1.4.1 Computational Grids

- Like an electric utility power grid, a computing grid offers an infrastructure that couples computers, software/middleware, special instruments, and people and sensors together. The grid is often constructed across LAN, WAN, or Internet backbone networks at a regional, national, or global scale.
- Enterprises or organizations present grids as integrated computing resources.
- The computers used in a grid are primarily workstations, servers, clusters, and supercomputers.
- Personal computers, laptops, and PDAs can be used as access devices to a grid system. The resource sites offer complementary computing resources, including workstations, large servers, a mesh of processors, and Linux clusters to satisfy a chain of computational needs. The grid is built across various IP broadband

networks including LANs and WANs already used by enterprises or organizations over the Internet.

- Special instruments may be involved such as using the radio telescope in SETI@Home
- search of life in the galaxy and the austrophysics@Swineburne for pulsars.
- At the server end, the grid is a network.
- At the client end, we see wired or wireless terminal devices. The grid integrates the computing, communication, contents, and transactions as rented services.

1.4.2 Grid Families

- Grid technology demands new distributed computing models, software/ middleware support, network protocols, and hardware infrastructures.
- National grid projects are followed by industrial grid platform development by IBM, Microsoft, Sun, HP, Dell, Cisco, EMC, Platform Computing, and others.
- Categories:
    - computational or data grids
    - P2P grids.

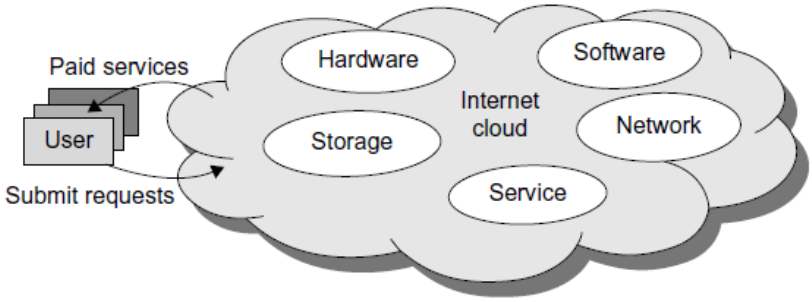
1.5 Cloud Computing over the Internet

- Internet clouds
- The cloud landscape

- "A cloud is a pool of virtualized computer resources. A cloud can host a variety of different workloads, including batch-style backend jobs and interactive and user-facing applications"
- The cloud allows workloads to be deployed and scaled out quickly through rapid provisioning of virtual or physical machines.
- The cloud supports redundant, self-recovering, highly scalable programming models that allow workloads to recover from many unavoidable hardware/software failures.
- Finally, the cloud system should be able to monitor resource use in real time to enable rebalancing of allocations when needed.

1.5.1 Internet Clouds

- Cloud computing applies a virtualized platform with elastic resources on demand by provisioning hardware, software, and data sets dynamically.
- The idea is to move desktop computing to a service-oriented platform using server clusters and huge databases at data centers.
- Cloud computing leverages its low cost and simplicity to benefit both users and providers.



1.5.2 The Cloud Landscape

Traditionally, a distributed computing system tends to be owned and operated by an autonomous administrative domain (e.g., a research laboratory or company) for on-premises computing needs.
However, these traditional systems have encountered several performance bottlenecks: constant system maintenance, poor utilization, and increasing costs associated with hardware/software upgrades.
Cloud computing as an on-demand computing paradigm resolves or relieves us from these problems.
Cloud service models:

•Infrastructure as a Service (IaaS)

- This model puts together infrastructures demanded by users — namely servers, storage, networks, and the data center fabric.
- The user can deploy and run on multiple VMs running guest OSes on specific applications. The user does not manage or control the underlying cloud infrastructure, but can specify when to request and release the needed resources.

•Platform as a Service (PaaS)

- This model enables the user to deploy user-built applications onto a virtualized cloud platform. PaaS includes middleware, databases, development tools, and some runtime support such as Web 2.0 and Java.
- The platform includes both hardware and software integrated with specific programming interfaces. The provider supplies the API and software tools (e.g., Java, Python, Web 2.0, .NET).
- The user is freed from managing the cloud infrastructure.

• Software as a Service (SaaS)

- This refers to browser-initiate d application software over thousands of paid cloud customers.
- The SaaS model applies to business processes, industry applications, consumer relationship management (CRM), enterprise resources planning (ERP), human resources (HR) , and collaborative applications.
- On the customer side, there is no upfront investment in servers or software licensing.

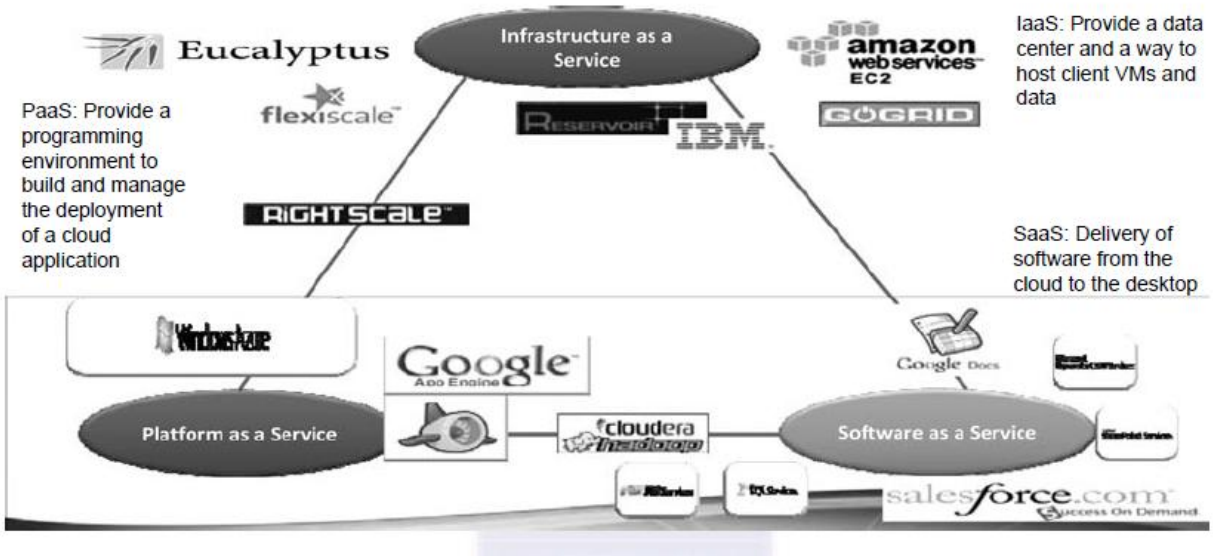
- On the provider side, costs are rather low, compared with conventional hosting of user applications.

Internet clouds offer four deployment modes:

private, public, managed , and hybrid

The following list highlights eight reasons to adapt the cloud for upgraded Internet applications and web services:

1. Desired location in areas with protected space and higher energy efficiency
2. Sharing of peak-load capacity among a large pool of users, improving overall utilization
3. Separation of infrastructure maintenance duties from domain-specific application development
4. Significant reduction in cloud computing cost, compared with traditional computing paradigms
5. Cloud computing programming and application development
6. Service and data discovery and content/service distribution
7. Privacy, security, copyright, and reliability issues



1.6 Service-Oriented Architecture (SOA)

- In grids/web services, Java, and CORBA, an entity is, respectively, a service, a Java object, and a CORBA distributed object in a variety of languages.
- These architectures build on the traditional seven Open Systems Interconnection (OSI) layers that provide the base networking abstractions.
- On top of this we have a base software environment.
- Eg: .NET or Apache Axis for web services
- the Java Virtual Machine for Java
- broker network for CORBA.

    - Layered architecture for web services and grids
    - Web services and tools
    - Evolution of SOA

- Grids vs clouds

1.6.1 Layered Architecture for Web Services and Grids

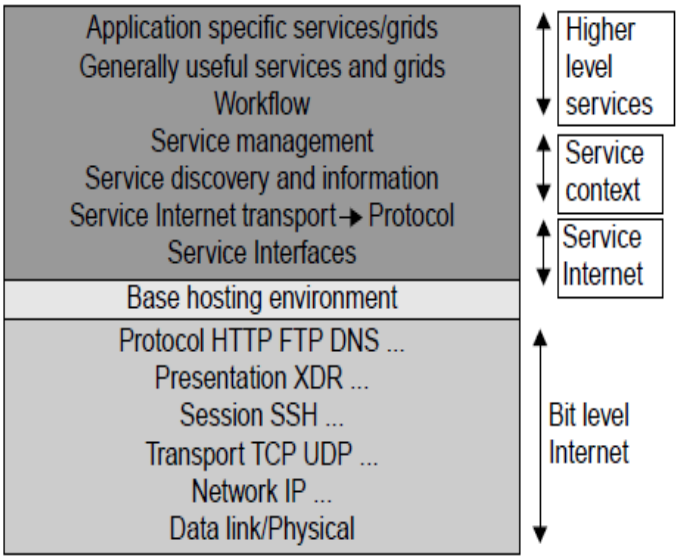
- The entity interfaces correspond to the Web Services Description Language (WSDL), Java method, andCORBA interface definition language (IDL) specifications in these example distributed systems.
- These interfaces are linked with customized, high-level communication systems: SOAP, RMI, and IIOP in the three examples.
- These communication systems support features including particular message patterns
- (such as Remote Procedure Call or RPC), fault recovery, and specialized routing.
- These communication systems are built on message-oriented middleware (enterprise bus) infrastructure such as Web-Sphere MQ or Java Message Service (JMS) which provide rich functionality and support virtualization of routing, senders, and recipients.
- Models: for example,
  - JNDI (Jini and Java Naming and DirectoryInterface) illustrating different approaches within the Java distributed object model.
  - The CORBA Trading Service, UDDI (Universal Description, Discovery, and Integration),
  - LDAP (Lightweight DirectoryAccess Protocol), and ebXML (Electronic Business using eXtensible Markup Language) are other examples of discovery and information services described.
- However, the distributed model has two critical advantages:
- higher performance (from multiple CPUs when communication is unimportant) and a cleaner separation of software functions with clear software reuse and maintenance advantages.

1.6.2 Web Services and Tools

- Loose coupling and support of heterogeneous implementations make services more attractive than distributed objects.
- In web services, one aims to fully specify all aspects of the service and its environment.
- This specification is carried with communicated messages using Simple Object Access Protocol (SOAP).
- The hosting environment then becomes a universal distributed operating system with fully distributed capability carried by SOAP messages.
- This approach has mixed success as it has been hard to agree on key parts of the protocol and even harder to efficiently implement the protocol by software such as Apache Axis.
- In the REST approach, one adopts simplicity as the universal principle and delegates most of the difficult problems to application (implementation-specific) software.
- In a web services language, REST has minimal information in the header, and the message body (that is opaque to generic message processing) carries all the needed

information. REST architectures are clearly more appropriate for rapid technology environments.

- In CORBA and Java, the distributed entities are linked with RPCs, and the simplest way to build composite applications is to view the entities as objects and use the traditional ways of linking themtogether.
- For Java, this could be as simple as writing a Java program with method calls replaced by Remote Method Invocation (RMI), while CORBA supports a similar model with a syntax reflecting the C++ style of its entity (object) interfaces.
- The "grid" to refer to a single service or to represent a collection of services, here sensors represent entities that output data (as messages), and grids and clouds represent collections of services that have multiple message-based inputs and outputs.



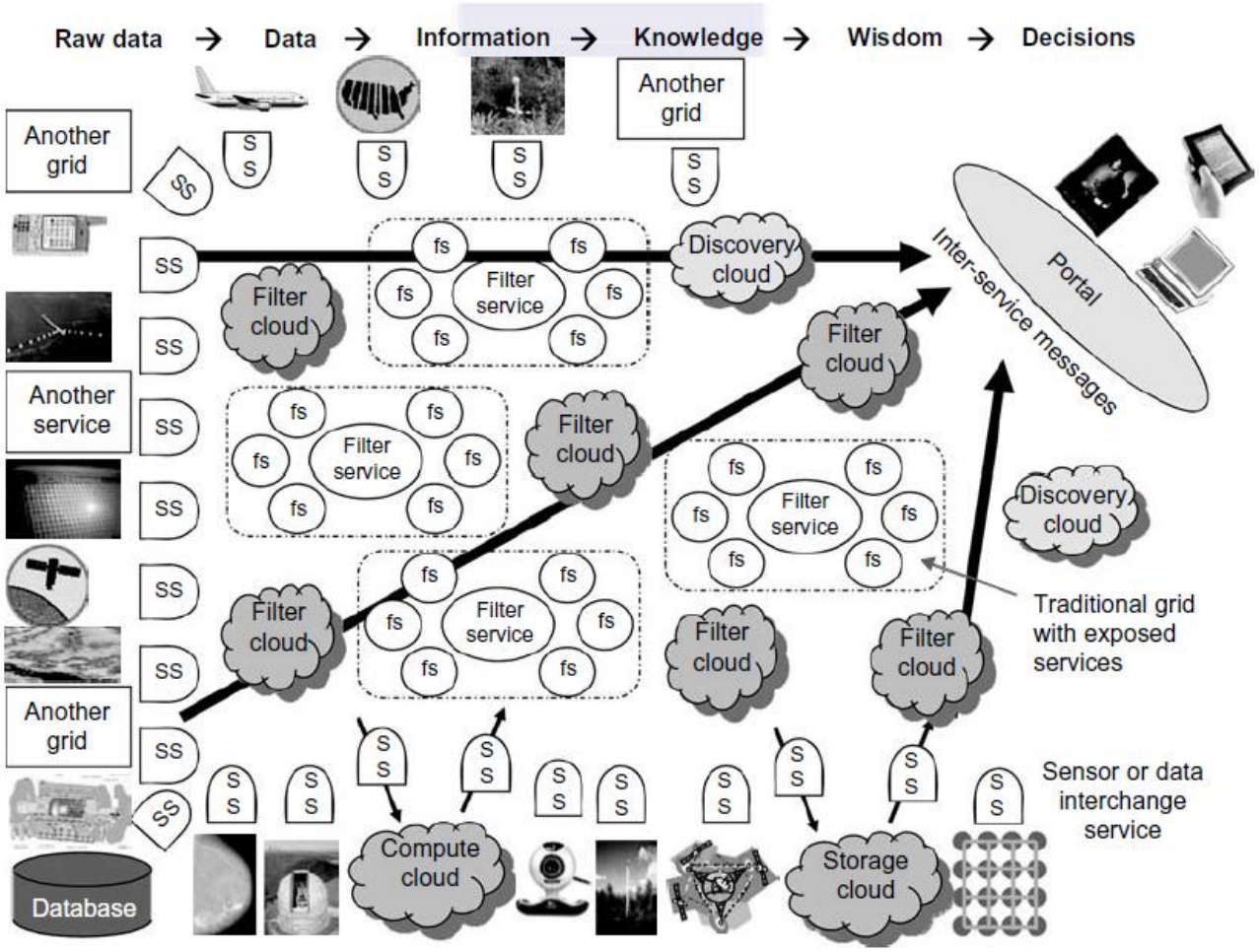
1.6.3 Evolution of SOA

- As shown in Figure 1.21, service-oriented architecture (SOA) has evolved over the years.
- SOA applies to building grids, clouds, grids of clouds, clouds of grids, clouds of clouds (also known as interclouds), and systems of systems in general.
- A large number of sensors provide data-collection services, denoted in the figure as SS (sensor service). A sensor can be a ZigBee device, a Bluetooth device, a WiFi access point, a personal computer, a GPA, or a wireless phone, among other things.
- Raw data is collected by sensor services.
- All the SS devices interact with large or small computers, many forms of grids, databases, the compute cloud, the storage cloud, the filter cloud, the discovery cloud, and so on. Filter services (fs in the figure) are used to eliminate unwanted raw data, in order to respond to specific requests from the web, the grid, or web services.
- A collection of filter services forms a filter cloud.
- Most distributed systems require a web interface or portal.

- For raw data collected by a large number of sensors to be transformed into useful information or knowledge, the data stream may go through a sequence of compute, storage, filter, and discovery clouds.
- Finally, the inter-service messages converge at the portal, which is accessed by all users. Two example portals, OGFCE and HUBzero, using both web service (portlet) and Web 2.0 (gadget) technologies.
- Many distributed programming models are also built on top of these basic constructs.

1.6.4 Grids versus Clouds

- The boundary between grids and clouds are getting blurred in recent years. For web services, workflow
- Technologies are used to coordinate or orchestrate services with certain specifications used to define critical business process models such as two-phase transactions.
- In all approaches, one is building a collection of services which together tackle all or part of a distributed computing problem.
- In general, a grid system applies static resources, while a cloud emphasizes elastic resources.
- One can build a grid out of multiple clouds.
- This type of grid can do a better job than a pure cloud, because it can explicitly support negotiated resource allocation. Thus one may end up building with a system of systems: such as a cloud of clouds, a grid of clouds, or a cloud of grids, or inter-clouds as a basic SOA architecture



1.7 Grid Architecture: Basic Concepts

Grid architecture refers to those aspects of a grid system that are taken into consideration when a grid is designed and implemented. Grid architecture can be visualized as a layered architecture.

The topmost layer consists of the grid applications and the APIs from a user's perspec-tive. Then we have the middleware, which includes the software and packages used for grid implementation, for example Globus Toolkit, gLite.

The third layer covers the resources available to the grid such as storage, processing capabilities and other application-specific hardware.

Finally the fourth layer is the network, layer which deals with the network components like routers, switches, and the protocols used for communication between any two systems in the grid.

The basic functionalities of middleware layer provide the basic functionality needed for grid computing.

1.7.1 Security:

We look at the three most desirable security features a grid should provide.

These are single sign-on, authentication and authorization.

Single sign-on means that the user is able to login once using his security credentials and can then access the service of the grid for a certain duration.

Authentication refers to providing the necessary proof to establish one's identity.

So, when you login to your email account, you authenticate to the server by providing your username and password.

Authorization is the process that checks the privileges assigned to a user.

For example, a website may have two kinds of user, a guest user and a registered user. A guest user may be allowed to perform basic tasks while the registered user may he allowed to perform a range of tasks based on his preferences.

Authorization is performed after the identity of a user has been established through authentication.

1.7.2 Resource Management

A grid must optimize the resources under its disposal to achieve maximum possible throughput.

Resource management includes submission of a job re-motel checking its status while it is in progress and obtaining the result.

When a job is submitted, the available resources are discovered through a directory service.

Then, the resources are selected to run the individual job. This decision is made by another resource management component of the grid, namely, the grid scheduler. The scheduling decision can be based on a number of factors.

1.7.3 Data Management:

Data management in grids covers a wide variety of aspects needed for man-aging large amounts of data. This includes secure data access, replication and migration of data, management of metadata, indexing, data-aware scheduling, caching etc.

We described replication of data in our discussion on fault tolerance.

Data aware-scheduling means that scheduling decisions should take into account the location of data.

For example, the grid scheduler can assign a job to a resource located close to data instead of transferring large amounts of data over the network, which can have significant performance overheads.

Suppose the job has been scheduled to run on a system that does not have the data needed for the job.

1.7.4 Information Discovery and Monitoring

A grid scheduler needs to be aware of the available resources to allocate resources for carrying out a job.

This information is obtained from an information discovery service running in the grid.

The information discovery service contains a list of resources available for the disposal of the grid and their current status.

When a grid scheduler queries the service for the available resources, it can put constraints such as finding those resources that are relevant and best suited for a job.

The computing capacity needed for a job and the job requires fast CPUs for its execution, we select only those machines fast enough for the timely completion of the job.
The information discovery services can function in two ways.
It can publish the status of available resources through a defined interface (web services) or it can be queried for the list of available re-sources.
The hierarchical structure brings about the flexibility needed for grids, which contains a vast amount of resources, because it can become practically impossible to store the information about all the available resources in one place.

1.8 Grid Services:

1.8.1 Web Service:

1. Xtensible Markup Language (AML) – A XML is a markup language whose purpose is to facilitate sharing of data across different interfaces using a common format. It forms the basis of web services. All the messages exchanged in web services adhere to the XML document format.

2. Simple Object Access Protocol (SOAP) - a message-based communication protocol, which can be used by two parties communicating over the Internet. SOAP messages are based on XML and are hence platform independent. It forms the foundation of the web services protocol stack. SOAP messages are transmitted over HTTP. So unlike other technologies like RPC or CORBA, SOAP messages can traverse a firewall. SOAP messages are suitable when small messages are sent.

3. Web Service Definition Language (WSDL) - WSDL is an XML document used to describe the web service interface. A WSDL document describes a web service using the following major elements:
(a) portType - The set of operations performed by the web service. Each operation is defined by a set of input and output messages.
(b) message - It represents the messages used by the web service. It is an abstraction of the data being transmitted.
(c) types - It refers to the data types defined to describe the message exchange.
(d) binding - It specifies the communication protocol used by the web service.
(e) port - It defines the binding address for the web service.
(f) service - It is used for aggregating a set of related ports

4. Universal Description, Discovery and Integration (UDDI) - UDDI is an XML-based registry used for finding a web service on the Inter-net. It is a specification that allows a business to publish information about it and its web services allowing other web services to locate this information. A UDDI registry is an XML-based service listing. Each listing contains the necessary information required to find and bind to a particular web service.

1.8.2 Open Grid Services Architecture (OGSA)

Open Grid Services Architecture (OGSA) defines a web services based framework for the implementation of a grid.
It seeks to standardize service provided by a grid such as resource discovery, resource management, security, etc, through a standard web service interface. It also defines those

features that are not necessarily needed for the implementation of a grid, OGSA is based on existing web services specifications and adds features to web services to make it suitable for the grid environment.
OGSA literature talks of grid services, an extension to the web services suitable for grid requirements.

1.8.3 Open Grid Services Infrastructure (OGSI)

OGSA describes the features that are needed for the implementation of services provided by the grid, as web services.
It, however, does not provide the details of the implementation.
Open Grid Services Infrastructure (OGSI) provides a formal and technical specification needed for the implementation of grid services. It provides a description of Web Service Description Language (WSDL), which defines a grid service. OGSI also provides the mechanisms for creation, management and interaction among grid services.

1.8.4 Web Services Resource Framework (WSRF)

The motivation behind development of WS-Resource framework is to define a "generic and open framework for modeling and accessing stateful resources using web services" [10]. It defines conventions for state management enabling applications to discover and interact with stateful web services in a standard way. Standard web services do not have a notion of state. Grid-based applications need the notion of state because they often perform a series of requests where output from one operation may depend on the result of previous operations. WSResource Framework can be used to develop such stateful grid services.

1.8.5 OGSA-DAI Open Grid Services Architecture

Data Access and Integration (OGSA-DAI) is a project conceived by the UK Database Task Force. This project's aim is to develop middleware to provide access and integration to distributed data sources using a grid. This middleware provides support for various data sources such as relational and WI. databases. These data sources can be queried, updated and transformed via OGSA-DAI web service. These web services can be deployed within a grid, thus making the data sources grid-enabled.

1.9 Grid Elements

In this section, we identify, at a high level, the major components of a grid computing system from a functional perspective.
A resource is an entity that is to be shared; this includes computers, storage, data, and software. A resource does *not* have to be a physical entity. A resource is defined in terms of interfaces, not devices; for example, schedulers such as Platform's LSF and PBS define a compute resource. Open/close/read/write define access to a distributed file system, for example, NFS, AFS, DFS.
In this section, we look at following grid components:

- ***Grid portal***
- ***Security (grid security infrastructure)***
- ***Broker (along with directory)***

- ***Scheduler***
- ***Data management***
- ***Job and resource management***
- ***Resources***

1.9.1 Portal/User Interface Function/Functional Block

A portal/user interface functional block usually exists in the grid environment.
The user interaction mechanism can take a number of forms.
The interaction mechanism typically is application specific.
In the simplest grid environment, the user access may be via a portal (see Figure top).
Such a portal provides the user with an interface to launch applications.
The applications make transparent the use of resources and/or services provided by the grid.
With this arrangement, the user perceives the grid as a virtual computing resource.

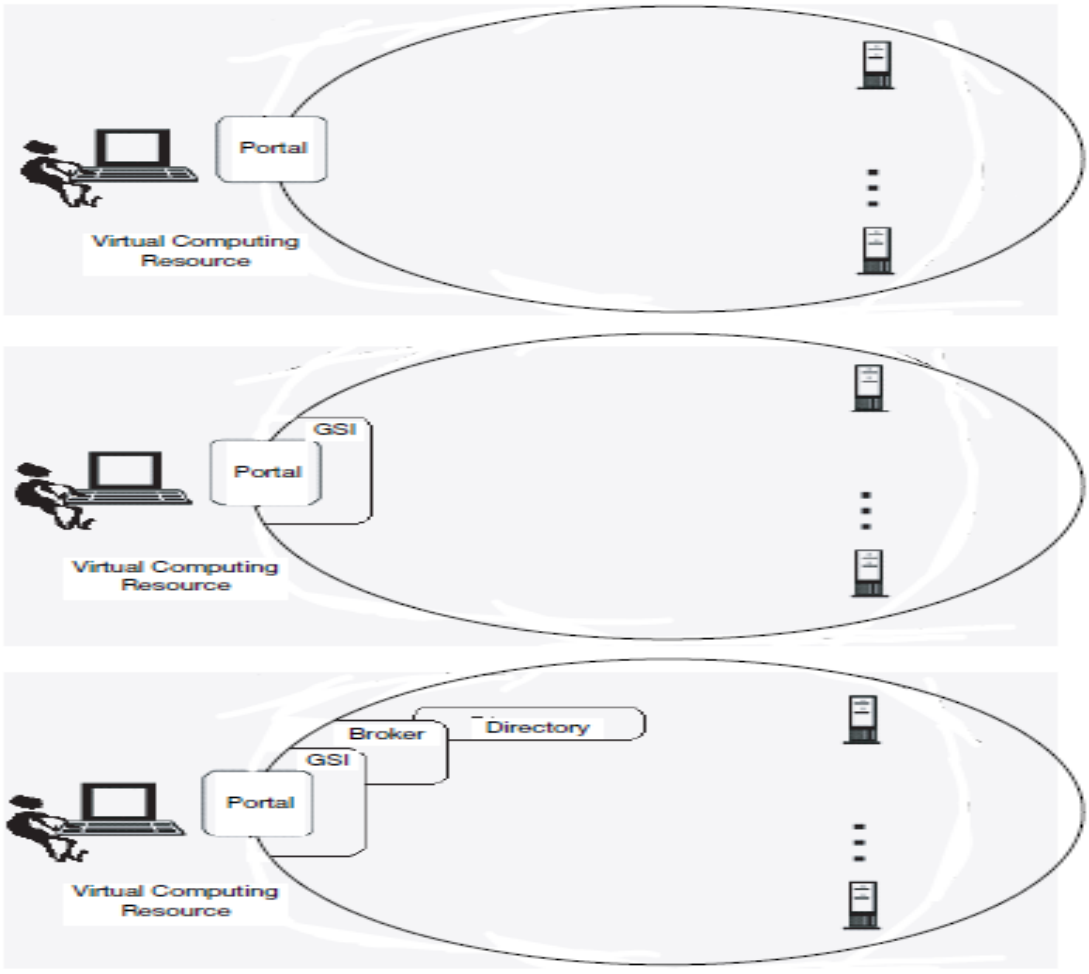


Figure 3.5 Basic grid elements—a user's view.

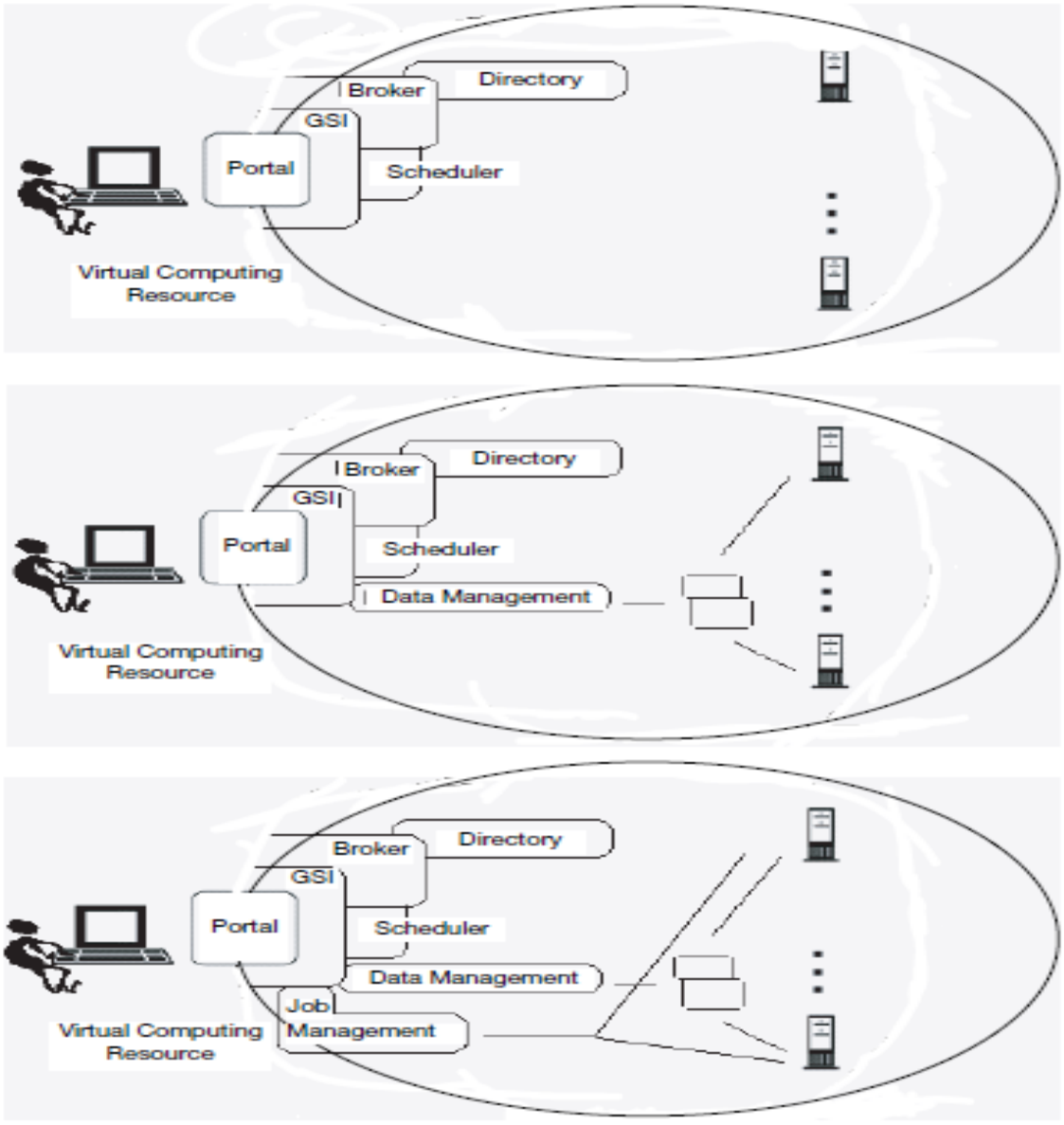


Figure 3.6 Additional grid elements—a user's view.

The Grid Security Infrastructure: User Security

1.9.1Function/Functional Block

- A user security functional block usually exists in the grid environment and, as noted above, a key requirement for grid computing is security.
- In a grid environment, there is a need for mechanisms to provide authentication, authorization, data confidentiality, data integrity, and availability, particularly from a user's point of view; see Figure, center.
- When a user's job executes, typically it requires confidential message-passing services. There may be on-the-fly relationships.
- But also, the user of the grid infrastructure software (such as a specialized scheduler) may need to set up a long-lived service; administrators may require that only certain users are allowed to access the service.

- In each of these cases, the application must anticipate and be designed to provide this required security functionality.
- In grids (particularly intergrids), there is a requirement to support security across organizational boundaries.
- This makes a centrally managed security system impractical; administrators want to support "single sign-on" for users of the grid, including delegation of credentials for computations that involve multiple resources and/or sites.
- The grid security infrastructure provides a single-sign-on, run-anywhere authentication service, with support for local control over access rights and mapping from global to local user identities.
- The grid security infrastructure supports uniform authentication, authorization, and message-protection mechanisms in multiinstitutional settings.

1.9.2 Node Security Function/Functional Block

- A node security functional block usually exists in the grid environment.
- Authentication and authorization is a "two-way street"; not only does the user need to be authenticated, but also the computing resource.
- There is the need for secure (authenticated and, in most instances, also confidential) communication between internal elements of a computational grid.
- This is because a grid is comprised of a collection of hardware and software resources whose origins may not be obvious to a grid user.
- When a user wants to run on a particular processor, the user needs assurances that the processor has not been compromised, making his or her proprietary application, or data, subject to undesired exposure.
- A certificate authority (CA) can be utilized to establish the identity of the "donor" processor, as well as the users and the grid itself. Some grid systems provide their own log-in to the grid, whereas other grid systems depend on the native operating systems for user authentication.

1.9.3Broker Function/Functional Block and Directory

- A broker functional block usually exists in the grid environment.
- After the user is authenticated by the user security functional block, the user is allowed to launch an application.
- At this juncture, the grid system needs to identify appropriate and available resources that can/should be used within the grid, based on the application and application-related parameters provided by the user of the application.
- This task is carried out by a broker function.
- The broker functionality provides information about the available resources on the grid and the working status of these resources.

1.9.4 Scheduler Function/Functional Block

- A scheduler functional block usually exists in the grid environment. If a set of stand-alone jobs without any interdependencies needs to execute, then a scheduler is not necessarily required.

- In the situation where the user wishes to reserve a specific resource or to ensure that different jobs within the application run concurrently, then a scheduler is needed to coordinate the execution of the jobs.
- In a "trivial" environment, the user may select a processor suitable for running the job and then execute a grid instruction that routes the job to the selected processor.
- In "nontrivial" environments, a grid-based system is responsible for routing a job to a properly selected processor so that the job can execute.
- Here, the scheduling software identifies a processor on which to run a specific grid job that has been submitted by a user; see Figure, top.
- After available resources have been identified, the follow-on step is to schedule the individual jobs to run on these resources.
- Schedulers are designed to dynamically react to grid load.
- They accomplish this by utilizing measurement information relating to the current utilization of processors to determine which ones are available before submitting a job.
- In an entry-level case, the scheduler could assign jobs in a round-robin fashion to the "next" processor matching the resource requirements.
- More commonly, the scheduler automatically finds the most appropriate processor on which to run a given job.
- Some schedulers implement a job priority (queue) mechanism.
- In this environment, as grid processors become available to execute jobs, the jobs are selected
- from the highest-priority queues first. Policies of various kinds can be implemented via the scheduler; for example, there could be a policy that restricts grid jobs from executing at certain time windows.
- In some more complex environments, there could be different levels of schedulers organized in a hierarchy.
- For example, a meta scheduler may submit a job to a cluster scheduler or other lower-level scheduler rather than to an individual target processor.
- As another example, a cluster could be represented as a single resource; here, the cluster could have its own scheduler to manage the internal cluster nodes, while a higher-level scheduler could be employed to schedule work to be supported by the cluster in question as an ensemble.
- Advanced schedulers monitor the progress of active jobs, managing the overall workflow.
- If the jobs were to become lost due to system or network outages, a highend scheduler would automatically resubmit the job elsewhere.
- However, if a job appears to be in an infinite loop and reaches a maximum timeout, then such jobs will not be rescheduled.
- Typically, jobs have different kinds of completion codes. This code determines if the job is suitable for resubmission or not.
- Some grids also have a "reservation system."
- These systems allow one to reserve resources on the grid.
- These are a calendar-based mechanism for reserving resources for specific time periods, and preventing others from reserving the same resource at the same time.

1.9.5 Data Management Function/Functional Block

A data management functional block usually exists in a grid environment.
There typically needs to be a reliable (and secure) method for moving files and data to various nodes within the grid.
This functionality is supported by the data management functional block. Figure, middle, depicts a data management function needed to support this data management function.

1.9.6 Job Management and Resource Management Function/Functional Block

- A job management and resource management functional block usually exists in a grid environment. This functionality is also known as the grid resource allocation manager (GRAM).
- The job management and resource management function (see Figure 3.6, bottom) provides the services to actually launch a job on a particular resource, to check the job's status, and to retrieve the results when the job is complete.
- Typically, the management component keeps track of the resources available to the grid and which users are members of the grid.
- This information is used by the scheduler to decide where grid jobs should be assigned.
- Also, typically, there are measurement mechanisms that determine both the capacities of the nodes on the grid and their current utilization levels at any given point in time; this information is used to schedule jobs in the grid, to monitor the health of the grid.
- Furthermore, advanced grid management software can automatically manage recovery from a number of grid failures and/or outages.
- With grid computing, administrators can "virtualize," or pool, IT resources (computers, storage, and applications) into a single virtual system whose resources can be managed from a single administration console and can be allocated dynamically, based on demand.
- The job management and resource management functional block supports this simplified view of the enterprise-wide resources.
- The work involved in managing the grid may be distributed hierarchically, in order to increase the scalability of the grid.
- For example, a central job scheduler may not schedule a submitted job directly, but instead, the job request is sent to a secondary scheduler that handles a specified set of processors (e.g., a cluster); the secondary scheduler handles the assignment to the specific processor.
- Hence, in this instance, the grid operation, the resource data, and the job scheduling are distributed to match the topology of the grid.
- On the resource management side of this function, mechanisms usually exist to handle observation, management, measurement, and correlation.
- It was noted above that schedulers need to react to instantaneous loads on the grid.

- The donor software typically includes "load sensors" that measure the instantaneous load and activity on resources or processors.
- Such measurement information is useful not only for instantaneous scheduling of tasks and work, but also for assessing (administratively) overall grid usage patterns.
- Observation, management, and measurement data can be used, in aggregate, to support capacity planning and initiate deployment of additional hardware.
- Furthermore, measurement information about specific jobs can be collected and used to forecast the resource requirements of that job the next time it executes. Some grid systems provide the means for implementing custom load sensors for more than just processor or storage resources.

1.9.7 User/Application Submission Function/Functional Block

- A user/application submission functional block usually exists.
- Typically, any member of a grid can submit jobs to the grid and perform grid queries, but in some grid systems, this function is implemented as a separate component installed on "submission nodes or clients".

1.9.8 Resources

- A grid would be of no value if it did not contribute resources to the ultimate user and/or application.
- As noted, resources include processors, data storage, scientific equipment, etc. Besides "physical presence" on the grid (by way of an interconnecting network), there has to be "logical presence." "Logical presence" is achieved by installing grid-support software on the participating processors.
- After loading and activating the software that manages the grid's use of its affiliated resources, each processor contributing itself or contributing ancillary resources to the grid needs to properly enroll as a member of the grid.
- User accessing the grid to accomplish a task submits a job for execution on the grid.
- The grid management software communicates with the grid donor software of the resource(s) logically present to forward the job to an appropriate processor.
- The grid-support software on the processor accepts an executable job from the grid management system and executes it.
- The grid software on the "donor" processor must be able to receive the executable file (in some cases the executable copies preinstalled on the processor.)
- The software is run and the output is sent back to the requester.
- More advanced implementations can dynamically adjust the priority of a running job, suspend a job and resume it later, or checkpoint a job with the possibility of resuming its execution on a different processor .
- The grid system sends information about any available resources on that processor to the resource management functional block described in the previous subsection.
- The participating donor processor typically has a self monitoring capability that determines or measures how busy the processor is.
- This information is "distributed" to the management software of the grid and it is utilized to schedule the appropriate use of the resources.

- For example, In a scavenging system, this utilization information informs the grid management software when the processor is idle and available to accept work.

1.9.9 Protocols

- After identifying the functional blocks, a generic architecture description proceeds by defining the protocols to be employed between (specifically, on the active interfaces of the) functional blocks. To interconnect these functional blocks, we need protocols, especially standardized protocols. Protocols are formal descriptions of message formats and a set of rules for message exchange.
- The rules may define sequence of message exchanges. Protocols are generally layered.
- Standards allows the grid to establish resource-sharing arrangements dynamically with *any* interested party and thus to create something more than a plethora of balkanized, incompatible, non interoperable distributed systems; standards are also important as a means of enabling general-purpose services and tools.
- Both open source and commercial products can, then, interoperate effectively in this heterogeneous, multivendor grid world, thus providing the pervasive infrastructure that will enable successful grid applications.
- At this juncture, the Global Grid Forum is in the process of developing consensus standards for grid environments. On the commercial side, nearly a decade of experience and refinement have resulted in a widely used de-facto standard in the form of the open source Globus Toolkit.
- The Global Grid Forum has a major effort underway to define the Open Grid Services Architecture (OGSA), which modernizes and extends Globus Toolkit protocols to address new requirements, while also embracing Web services.

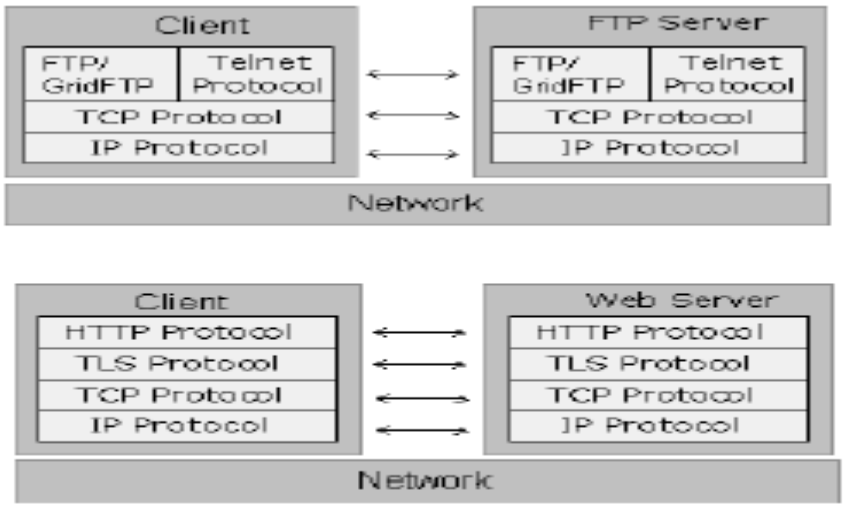


Figure 3.7 Example of protocol stacks and network-enabled services.

1.10 Overview of the Grid Architecture

- The “hour-glass” model of thev Grid architecture
- Thin center: few standards
- Wide top: many high-level behaviors can be mapped
- Wide bottom: many underlying technologies and systems
- Role of layers:
- Fabric: interfaces local control
- Connectivity: secure communications
- Resource: sharing a single resource
- Collective: coordinated sharing of multiple resources
- Application

1.10.1 Fabric Layer

- Need to manage a resource
  - Reserve usage in advance
  - Allocate space (e.g. storage)
- Grid fabric layer provides standardized access to localv resource-specific operations
- Software is provided to discover
- Computers (OS version, hardware config, usage load)
- Storage systems
- Networks
- Globus General-purpose Architecture for Reservation and Allocation (GARA)

1.10.2 Connectivity Layer

- Need secure connectivity to resources
- Assumes trust is based on user, not service providers
- Use public key infrastructure (PKI)
  - User is recognized by a Certificate Authority (CA) (within Grid)
  - Single sign-on: allow users to authenticate only once
  - Delegation: create proxy credentials to allow services/agents to act on a user’s behalf
- Integrate and obey local security policies in global view
  - Unix file permissions, ACLs, …
- Globus Security Infrastructure (GSI)
  - Standardized mechanism for proxy credential creation and mapping to local access authentication scheme (logins)
  - Based on generic services security (GSS) API, which allows applications to perform these security operations autonomously

1.10.3 Resource Layer

- Need access to resources
  - Invoke remote computation
  - Monitor computation
  - Discover resources
  - Perform data transport (e.g. to and from computations)

- Globus Grid Resource Allocation and Management (GRAM)
    - Job manager and reporter
- Globus Monitoring and Discovery Service (MDS-2)
    - Registry of resources and monitoring of resource usage stats
- Globus GridFTP
    - FTP but faster (multiple streams), integrated security
    - Partial file access

1.10.4 Collective Layer

Need to coordinate sharing of resources

- Directory services
- Coallocation,
- scheduling,
- brokering services
- Monitoring and diagnostic services
- Data replication services

- Tools
    - Workflow systems
    - Collaboratory services
- Globus DUROC resource coallocation library

Anna University Questions

Nov/dec 2016

1. Bring out the differences between private cloud and public cloud. (2m)
2. Highlight the importance of the term "cloud computing" (2m)
3. Illustrate the architecture of virtual machine and brief about the operations. (16 m)
4. Write short notes on i)Cluster of cooperative computers (8m), ii)Service oriented architecture (8m)

Apr / May 2017

1. Tabulate the difference between high performance computing and high throughput computing
2. Give the basic operations of a VM.
3. Brief the interaction between the GPU and CPU in performing parallel execution of operations. (16 m)
4. Illustrate with the neat sketch, the grid computing infrastructure. (16m)

Nov / Dec 2017

1. "Grid inherits features of P2P and cluster computing systems". Is the statement true? Valjdate your answer. (2m)
2. Differentiate behnreen grid and cloud computing (2m)
3. 3. i) Describe the infrastructure requirements for grid computing. (8m)
   ii) What are the issues in cluster design ? How can they be resolved ? (8m)
4. i) Describe layered grid architecture. How does it map onto internet protocol architecture? (8m)
   ii) Describe the architecture of a cluster with suitable illustrations. (8m)

UNIT II : GRID SERVICES

Introduction to Open Grid Services Architecture (OGSA) – Motivation – Functionality Requirements – Practical & Detailed view of OGSA/OGSI – Data intensive grid service models – OGSA services.

PART A (2 marks)

1. What are the major goals of OGSA?
   - Identify the use cases that can drive the OGSA platform components.
   - Identify and define the core OGSA platform components.
   - Define hosting and platform specific bindings.
   - Define resource models and resource profiles with interoperable solutions.

2. What are the more specific goals of OGSA?
   - Facilitating distributed resource management across heterogeneous platforms
   - Providing seamless quantity of service delivery.
   - Providing common infrastructure building blocks to avoid "Stove pipe solutions towers".
   - Open and published interfaces and messages.

3. What are the main purposes of use case defined by OGSA?
   - To identify and define core OGSA platform functionalities.
   - To define core platform components based on the functionality requirements.
   - To define the high level requirements on those core components and identify their interrelationship.

4. List out the categories of OGSA services?
   - Infrastructure Services
   - Execution Management Services
   - Data Management Services
   - Resource Management Services
   - Security Services
   - Information Services
   - Self-Management Services

5.what are the benefits of OGSI standard ?
   - Increased effective computing capacity.
   - Interoperability of resources.
   - Speed of application development.

6.What are The objectives of OGSA ?
   - Manage resources across distributed heterogeneous platforms.
   - Support QoS-oriented Service Level Agreements (SLAs).
   - It is critical that the grid provide robust services such as authorization, access control, and delegation.
   - Provide a common base for autonomic management.
   - Define open, published interfaces and protocols for the interoperability of diverse resources. OGSA is an open standard managed by a standards body.

7. what are two fundamental requirements for describing Web services based on the OGSI?

The ability to describe interface inheritance—a basic concept with most of the distributed object systems.The ability to describe additional information elements with the interface definitions.

8.what is a Grid Service Instance

A grid service instance is a (potentially transient) service that conforms to a set of conventions, expressed as WSDL interfaces, extensions, and behaviors, for such purposes as lifetime management, discovery of characteristics,and notification.

9.what is grid service description?

A grid service description describes how a client interacts with service instances.This description is independent of any particular instance. Within a WSDL document, the grid service description is embodied in the most derived portType of the
instance, along with its associated portTypes,bindings, messages, and types definitions.

10.name the few use cases of OGSA?

- National fusion collaboration
- IT infrastructure and management
- Commercial data centers
- Service-based distributed query processing
- Severe storm prediction
- Online media and entertainment

11. what are the Basic Functionality Requirements of OGSA/Grids?

- Discovery and brokering.
- Metering and accounting.
- Data sharing.
- Deployment
- Virtual organizations (VOs).
- Monitoring.
- Policy.

12.what are the Grids security requirements ?

- Multiple security infrastructures.
- Perimeter security solutions
- Authentication, Authorization, and Accounting.
- Encryption.
- Application and Network-Level Firewalls..
- Certification.

13.name the few Resource Management Requirements?

- Provisioning.
- Resource virtualization.

- Transport management
- Access
- Management and monitoring
- Load balancing

14.what are the System Properties Requirements?

- Fault tolerance
- Disaster recovery.
- Strong monitoring
- Legacy application management.

15. what are the Four Grid Families Identified in the Great Global Grid (GGG)?

- Computational Grids or Data Grids
- Information Grids or Knowledge Grids
- Business Grids
- P2P/Volunteer Grids

16.mention the few Grid Data Access Models?

- Monadic model.
- Hierarchical model.
- Federation model.
- Hybrid model.

17. Name some representational use cases from OGSA architecture working group?

- Commercial Data Center (Commercial grid)
- National Fusion Collaboratory (Science grid)
- Online Media and Entertainment (Commercial grid)

18. What are the layers available in OGSA architectural organizations?

- Native platform services and transport mechanisms.
- OGSA hosting environment.
- OGSA transport and security.
- OGSA infrastructure (OGSI).
- OGSA basic services (meta-OS and domain services)

19. What are the OGSA basic services?

- Common Management Model (CMM)
- Service domains
- Distributed data access and replication.
- Policy, security
- Provisioning and resource management.

20.what is meant by CPU scavenging.

The concept of creating a “grid” from the unused resources in a network of computers is known as CPU scavenging.

PART B (16 marks)

1. Explain in detail about Open Grid Services Architecture

The OGSA is an open source grid service standard jointly developed by academia and the IT industry under coordination of a working group in the Global Grid Forum (GGF). The standard was specifically developed for the emerging grid and cloud service communities. The OGSA is extended from web service concepts and technologies. The standard defines a common framework that allows businesses to build grid platforms across enterprises and business partners. The intent is to define the standards required for both open source and commercial software to support a global grid infrastructure

OGSA Framework

The OGSA was built on two basic software technologies: the Globus Toolkit widely adopted as a grid technology solution for scientific and technical computing, and web services (WS 2.0) as a popular standards-based framework for business and network applications. The OGSA is intended to support the creation, termination, management, and invocation of stateful, transient grid services via standard interfaces and conventions

OGSA Interfaces

The OGSA is centered on grid services. These services demand special well-defined application interfaces.

These interfaces provide resource discovery, dynamic service creation, lifetime management, notification, and manageability. These properties have significant implications regarding how a grid service is named, discovered, and managed

Port Type	Operation	Brief Description
Grid service	Find service data	Query a grid service instance, including the handle, reference, primary key, home handle map, interface information, and service-specific information. Extensible support for various query languages.
	Termination time	Set (and get) termination time for grid service instance.
	Destroy	Terminate grid service instance.
Notification source	Subscribe to notification topic	Subscribe to notifications of service events. Allow delivery via third-party messaging services.
Notification sink	Deliver notification	Carry out asynchronous delivery of notification messages.
Registry	Register service	Conduct soft-state registration of Grid Service Handles (GSHs).
	Unregister service	Unregister a GSH.
Factory	Create service	Create a new grid service instance.
Handle map	Find by handle	Return the Grid Service Reference (GSR) associated with the GSH.

Grid Service Handle

A GSH is a globally unique name that distinguishes a specific grid service instance from all others. The status of a grid service instance could be that it exists now or that it will exist in the future.

These instances carry no protocol or instance-specific addresses or supported protocol bindings. Instead, these information items are encapsulated along with all other instance-specific information. In order to interact with a specific service instance, a single abstraction is defined as a GSR.

Grid Service Migration

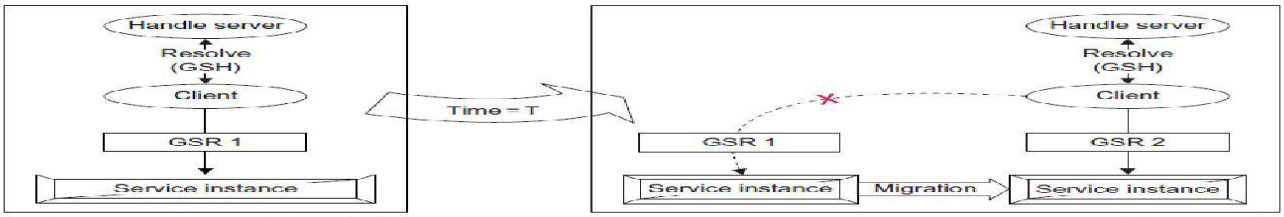
This is a mechanism for creating new services and specifying assertions regarding the lifetime of a service. The OGSA model defines a standard interface, known as a factor, to implement this reference. This creates a requested grid service with a specified interface and returns the GSH and initial GSR for the new service instance.

If the time period expires without having received a reaffirmed interest from a client, the service instance can be terminated on its own and release the associated resources accordingly

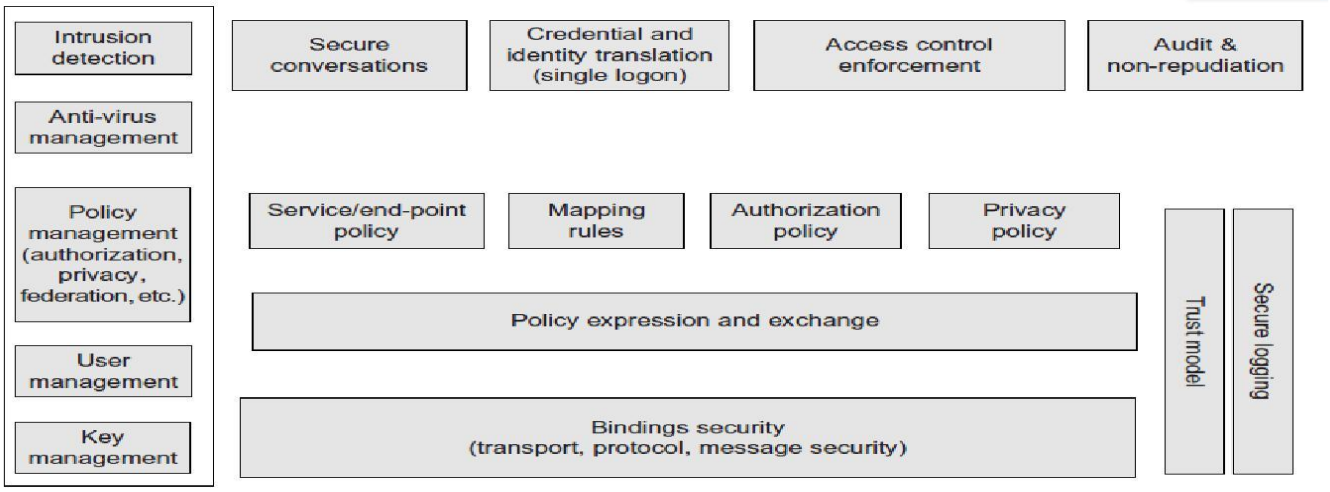
OGSA Security Models

The grid works in a heterogeneous distributed environment, which is essentially open to the general public. We must be able to detect intrusions or stop viruses from spreading by implementing secure conversations, single logon, access control, and auditing for nonrepudiation.

At the security policy and user levels, we want to apply a service or endpoint policy, resource mapping rules, authorized access of critical resources, and privacy protection. At the Public Key Infrastructure (PKI) service level, the OGSA demands security binding with the security protocol stack and bridging of certificate authorities (CAs), use of multiple trusted intermediaries, and so on.



A GSH resolving to a different GSR for a migrated service instance before (shown on the left) and after (on the right) the migration at time T.



The OGSA security model implemented at various protection levels.

2. Explain the Motivations for standardization in OGSA.

The lack of standards has meant that companies, developers, and organizations have had to develop and support grid technology using proprietary techniques and solutions, hereby limiting its deployment potential.

Some specific areas where a lack of grid standards limit deployment are

Data management

For a grid to work effectively, there is a need to store information and distribute it. Without a standardized method for describing the work and how it should be exchanged, one quickly encounters limits related to the flexibility and interoperability of the grid.

Dispatch management

There are a number of approaches that can be used to handle brokering of work units and to distribute these work units to client resources.

Again, not having a standard method for this restricts the service providers that can connect to the grid and accept units of work from the grid;this also restricts the ability of grid services users to submit work.

Information services

Metadata about the grid service helps the system to distribute information. The metadata is used to identify requesters (grid users), providers, and their respective requirements and resource availability.

Again, without a standard, one can only use specific software and solutions to support the grid applications.

Scheduling Work must be scheduled across the service providers to ensure they are kept busy. To accomplish this, information about remote loads must be collected and administered.

A standardized method of describing the grid service enables grid implementations to specify how work is to be scheduled.

Security

Without a standard for the security of a grid service and for the secure distribution of work units, one runs the risk of distributing information to the ―wrong‖ clients.

Although proprietary methods can provide a level of security, they limit accessibility.

Work unit management

Grid services require management of the distribution of work units to ensure that the work is uniformly distributed over the service providers. Without a standard way of advertising and managing this process, efficiencies are degraded.

Increased effective computing capacity

When the resources utilize the same conventions, interfaces, and mechanisms, one can transparently switch jobs among grid systems, both from the perspective of the server as well as from the perspective of the client. allows grid users to use more capacity and allows clients a more extensive choice of projects that can be supported on the grid. Hence, with a gamut of platforms and environments supported, along with the ability to more easily publish the services available, there will be an increase in the effective computing capacity.

Interoperability of resources
Grid systems can be more easily and efficiently developed and deployed when utilizing a variety of languages and a variety of platforms. For example, it is desirable to mix service-provider components, work-dispatch tracking systems, and systems management; this makes it easier to dispatch work to service providers and for service providers to support grid services.

Speed of application development
Using middleware (and/or toolkits) based on a standard expedites the development of grid-oriented applications supporting a business environment. Rather than spending time developing communication and management systems to help support the grid system, the planner can, instead, spend time optimizing the business/algorithmic logic related to the processing the data.

For useful applications to be developed, a rich set of grid services (the OGSA architected services) need to be implemented and delivered by both open source efforts (such as the Globus project) and by middleware software companies.

In a way, OGSI and the extensions it provides for Web services are necessary but insufficient for the maturation of the service-oriented architecture; the next required step is that these standards be fully implemented and truly observed (in order to provide portability and interoperability) a simple environment to put the network-based services in context.

3. Describe in detail about functionality requirements

The development of the OGSA document is based on a variety of use case scenarios. The functional requirements for the use cases provide useful input for the development of OGSA functions.

1. Basic functionality requirements
2. Security requirements
3. Resource Management Requirements
4. System Properties Requirements
5. Other functional requirements

1. Basic functionality requirements

Discovery and brokering. Mechanisms are required for discovering and/or allocating services, data, and resources with desired properties. For example, clients need to discover network services before they are used, service brokers need to discover hardware and software availability, and service brokers must identify codes and platforms suitable for execution requested by the client

Metering and accounting. Applications and schemas for metering, auditing, and billing for IT infrastructure and management use cases. The metering function records the usage and duration, especially metering the usage of licenses. The auditing function audits usage and application profiles on machines, and the billing function bills the user based on metering.

Data sharing. Data sharing and data management are common as well as important grid applications. chanisms are required for accessing and managing data archives, for caching data and managing its consistency, and for indexing and discovering data and metadata.

Deployment. Data is deployed to the hosting environment that will execute the job (or made available in or via a high-performance infrastructure). Also, applications (executable) are migrated to the computer that will execute them

Virtual organizations (VOs). The need to support collaborative VOs introduces a need for mechanisms to support VO creation and management, including group membership services [58]. For the commercial data center use case [55], the grid creates a VO in a data center that provides IT resources to the job upon the customer's job request.

Monitoring. A global, cross-organizational view of resources and assets for project and fiscal planning, troubleshooting, and other purposes. The users want to monitor their applications running on the grid. Also, the resource or service owners need to surface certain states so that the user of those resources or services may manage the usage using the state information

Policy. An error and event policy guides self-controlling management, including failover and provisioning. It is important to be able to represent policy at multiple stages in hierarchical systems, with the goal of automating the enforcement of policies that might otherwise be implemented as organizational processes or managed manually

2.Security requirements

Grids also introduce a rich set of security requirements; some of these requirements are: ***Multiple security infrastructures***. Distributed operation implies a need to interoperate with and manage multiple security infrastructures. For example,for a commercial data center application, isolation of customers in the same commercial data center is a crucial requirement; the grid should provide not only access control but also performance isolation.

Perimeter security solutions. Many use cases require applications to be deployed on the other side of firewalls from the intended user clients. Intergrid collaboration often requires crossing institutional firewalls.

Authentication, Authorization, and Accounting. Obtaining application programs and deploying them into a grid system may require authentication/authorization. In the commercial data center use case, the commercial data center authenticates the customer and authorizes the submitted request when the customer submits a job request.

Encryption. The IT infrastructure and management use case requires encrypting of the communications, at least of the payload

Application and Network-Level Firewalls. This is a long-standing problem; it is made particularly difficult by the many different policies one is dealing with and the particularly harsh restrictions at international sites.

Certification. A trusted party certifies that a particular service has certain semantic behavior. For example, a company could establish a policy of only using e-commerce services certified by Yahoo

3.Resource Management Requirements

Resource management is another multilevel requirement, encompassing SLA negotiation, provisioning, and scheduling for a variety of resource types and activities

Provisioning. Computer processors, applications, licenses, storage, networks, and instruments are all grid resources that require provisioning. OGSA needs a framework that allows resource provisioning to be done in a uniform, consistent manner.

Resource virtualization. Dynamic provisioning implies a need for resource virtualization mechanisms that allow resources to be transitioned flexibly to different tasks as required; for example, when bringing more Web servers on line as demand exceeds a threshold..

Optimization of resource usage while meeting cost targets (i.e., dealing with finite resources). Mechanisms to manage conflicting demands from various organizations, groups, projects, and users and implement a fair sharing of resources and access to the grid

Transport management. For applications that require some form of real-time scheduling, it can be important to be able to schedule or provision bandwidth dynamically for data transfers or in support of the other data sharing applications. In many (if not all) commercial applications, reliable transport management is essential to obtain the end-to-end QoS required by the application

Management and monitoring. Support for the management and monitoring of resource usage and the detection of SLA or contract violations by all relevant parties. Also, conflict management is necessary;

Processor scavenging is an important tool that allows an enterprise or VO to use to aggregate computing power that would otherwise go to waste

Scheduling of service tasks. Long recognized as an important capability for any information processing system, scheduling becomes extremely important and difficult for distributed grid systems.

Load balancing. In many applications, it is necessary to make sure make sure deadlines are met or resources are used uniformly. These are both forms of load balancing that must be made possible by the underlying infrastructure

Advanced reservation. This functionality may be required in order to execute the application on reserved resources.

Notification and messaging. Notification and messaging are critical in most dynamic scientific problems.

Logging. It may be desirable to log processes such as obtaining/deploying application programs because, for example, the information might be used for accounting. This functionality is represented as ―metering and accounting.‖

Workflow management. Many applications can be wrapped in scripts or processes that require licenses and other resources from multiple sources. Applications coordinate using the file system based on events

Pricing. Mechanisms for determining how to render appropriate bills to users of a grid.

4. System Properties Requirements

Fault tolerance. Support is required for failover, load redistribution, and other techniques used to achieve fault tolerance. Fault tolerance is particularly important for long running queries that can potentially return large amounts of data, for dynamic scientific applications, and for commercial data center applications.

Disaster recovery. Disaster recovery is a critical capability for complex distributed grid infrastructures. For distributed systems, failure must be considered one of the natural behaviors and disaster recovery mechanisms must be considered an essential component of the design.

Self-healing capabilities of resources, services and systems are required. Significant manual effort should not be required to monitor, diagnose, and repair faults.

Legacy application management. Legacy applications are those that cannot be changed, but they are too aluable to give up or to complex to rewrite. Grid infrastructure has to be built around them so that they can continue to be used

Administration. Be able to ―codify‖ and ―automate‖ the normal practices used to administer the environment. The goal is that systems should be able to selforganize and self-describe to manage low-level configuration details based on higher-level configurations and management policies specified by administrators.

Agreement-based interaction. Some initiatives require agreement-based interactions capable of specifying and enacting agreements between clients and servers (not necessarily human) and then composing those agreements into higher-level end-user structures

Grouping/aggregation of services. The ability to instantiate (compose) services using some set of existing services is a key requirement. There are two main types of composition techniques: selection and aggregation. Selection involves choosing to use a particular service among many services with the same operational interface.

5. **Other Functional Requirements**

Grid environments tend to be heterogeneous and distributed

Platforms:

The platforms themselves are heterogeneous, including a variety of operation systems (Unixes,Linux, windows, embedded systems), hosting environments (J2EE,.NET) and devices (computers, instruments, sensors, storage systems, databases, networks)

Mechanisms:

Grid software can need to interoperate with a variety of distinct implementation mechanisms for core functions such as security.

Administrative environments:

Geographically distributed environment often feature varied usage, management and administration policies that need to be honoured and managed.

4. Describe in detail about Practical view of OGSA/OGSI

OGSA aims at addressing standardization (for interoperability) by defining the basic framework of a grid application structure. Some of the mechanisms employed in the standards formulation of grid computing

The objectives of OGSA are

- Manage resources across distributed heterogeneous platforms
- Support QoS-oriented Service Level Agreements (SLAs). The topology of grids is often complex; the interactions between/among grid resources are almost invariably dynamic.
- Provide a common base for autonomic management. A grid can contain a plethora of resources, along with an abundance of combinations of resource MPICH-G2: Grid-enabled message passing (Message Passing Interface)

_ CoG Kits, GridPort: Portal construction, based on N-tier architectures

_ Condor-G: workflow management

_ Legion: object models for grid computing

_ Cactus: Grid-aware numerical solver framework

Portals

N-tier architectures enabling thin clients, with middle tiers using grid functions . Thin clients = web browsers

Middle tier = e.g., Java Server Pages, with Java CoG Kit, GPDK, GridPort utilities _ Bottom tier = various grid resources

Numerous applications and projects, e.g.,

Unicore, Gateway, Discover, Mississippi Computational Web Portal, NPACI Grid Port, Lattice Portal, Nimrod-G, Cactus, NASA IPG Launchpad, Grid Resource

Broker

High-Throughput Computing and Condor

High-throughput computing

- Processor cycles/day (week, month, year?) under nonideal circumstances
- How many times can I run simulation X in a month using all available machines?
- Condor converts collections of distributively owned workstations and dedicated clusters into a distributed high-throughput computing facility _ Emphasis on policy management and reliability

Object-Based Approaches

- Grid-enabled CORBA
- NASA Lewis, Rutgers, ANL, others
- CORBA wrappers for grid protocols
- Some initial successes
- Legion
- University of Virginia

Object models for grid components (e.g.,"vault" = storage, "host" = computer)
Cactus: Modular, portable framework for parallel, multidimensional simulations Construct codes by linking

Small core: management services
Selected modules: Numerical methods, grids and domain decomps, visualization and steering, etc.
Custom linking/configuration tools
Developed for astrophysics, but not astrophysics specific

Table 4.6 Proposed OGSA grid service interfaces*

Port type	Operation	Description
GridService	FindServiceData	Query a variety of information about the grid service instance, including basic introspection information (handle, reference, primary key, home handle map: terms to be defined), richer per-interface information, and service-specific information (e.g., service instances known to a registry). Extensible support for various query languages.
	SetTermination Time	Set (and get) termination time for grid service instance
	Destroy	Terminate grid service instance.
Notification-Source	SubscribeTo-NotificationTopic	Subscribe to notifications of service-related events, based on message type and interest statement. Allows for delivery via third-party messaging services.
Notification-Sink	Deliver Notification	Carry out asynchronous delivery of notification messages.
Registry	RegisterService UnregisterService	Conduct soft-state registration of grid service handles. Deregister a grid service handle.
Factory	CreateService	Create new grid service instance.
Handle Map	FindByHandle	Return grid service reference currently associated

There are two fundamental requirements for describing Web services based on the OGSI

1.The ability to describe interface inheritance—a basic concept with most of the distributed object systems.

2. The ability to describe additional information elements with the interface definitions.

5.What is OGSA/OGSI? A More Detailed View

Introduction

The OGSA integrates key grid technologies with Web services mechanisms to create a distributed system framework based on the OGSI. A *grid service instance* is a service that conforms to a set of conventions, expressed as WSDL interfaces, extensions, and behaviours, for such purposes as lifetime management, discovery of characteristics, and notification. Grid services provide for the controlled management of the distributed and often long-lived state that is commonly required in sophisticated distributed applications. OGSI also introduces standard factory and registration interfaces for creating and discovering grid services.

OGSI defines a component model that extends WSDL and XML schema definition to incorporate the concepts of

- Stateful Web services
- Extension of Web services interfaces
- Asynchronous notification of state change
- References to instances of services
- Collections of service instances

Service state data that augment the constraint capabilities of XML schema definition

The OGSI specifies (1) how grid service instances are named and referenced; (2) the base, common interfaces that all grid services implement; and (3) the additional interfaces and behaviours associated with factories and service groups.

Setting the Context

GGF calls OGSI the "base for OGSA." Specifically, there is a relationship between OGSI and distributed object systems and also a relationship between OGSI and the existing Web services framework.

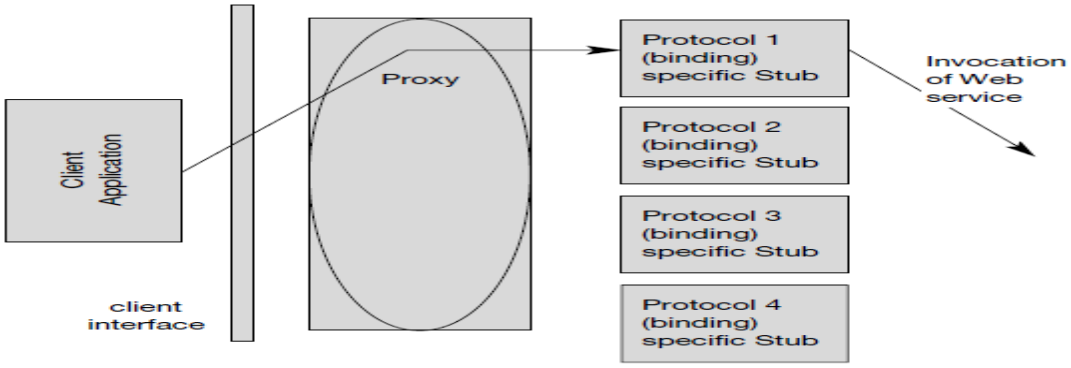
Relationship to Distributed Object Systems.

A given grid service implementation is an addressable and potentially stateful instance that implements one or more interfaces described by WSDL portTypes. Grid service factories can be used to create instances implementing a given set of portType(s). Each grid service instance has a notion of identity with respect to the other instances in the distributed grid. Each instance can be characterized as state coupled with behaviour published through type-specific operations.

Grid service instances are made accessible to client applications through the use of a grid service handle and a grid service reference (GSR).A client application can use a grid service reference to send requests, represented by the operations defined in the portType(s) of the target service description directly to the specific instance at the specified network-attached service endpoint identified by the grid service reference.

Client-Side Programming Patterns.

OGSI exploits an important component of the Web services framework: the use of WSDL to describe multiple protocol bindings, encoding styles, messaging styles, and so on, for a given Web service. The Web Services Invocation Framework (WSIF) and Java API for XML RPC (JAX-RPC) are among the many examples of infrastructure software that provide this capability.

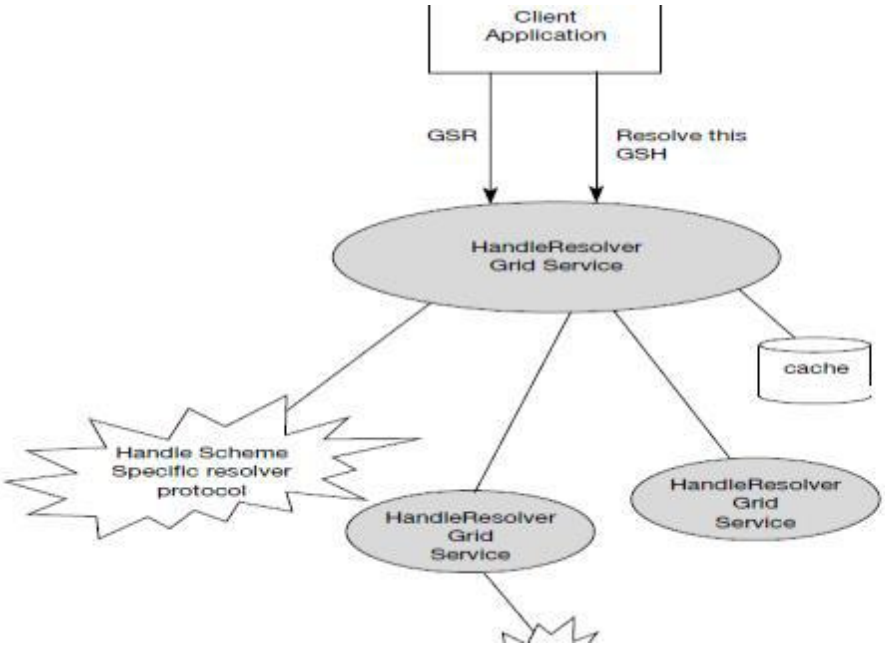


Various tools can take the WSDL description of the Web service and generate interface definitions in a wide range of programming-language-specific constructs.

A *proxy* provides a client-side representation of remote service instance's interface. Proxy behaviors specific to a particular encoding and network protocol are encapsulated in a *protocol-specific (binding-specific) stub*. This includes both application-specific services and common infrastructure services that are defined by OGSA.

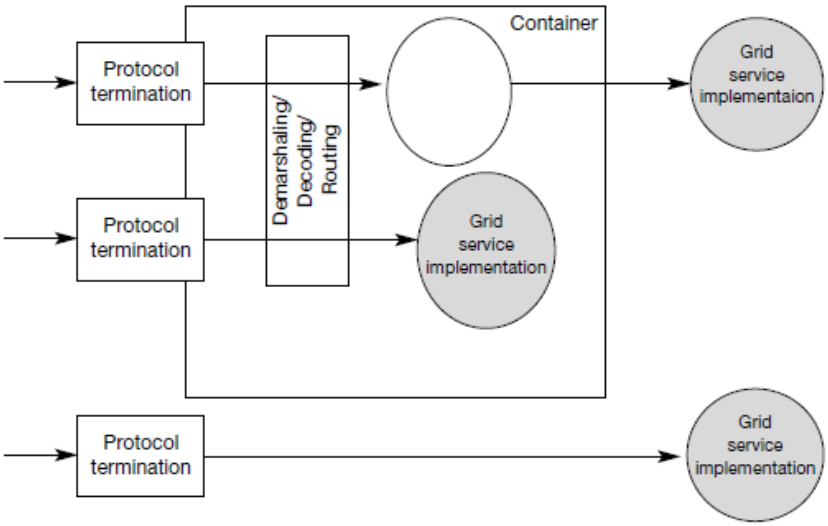
Client Use of Grid Service Handles and References.

A grid service handle (GSH) can be thought of as a permanent network pointer to a particular grid service instance. The client resolves a GSH into a GSR by invoking a HandleResolver grid service instance identified by some out-of-band mechanism. The HandleResolver may have the GSR stored in a local cache. The HandleResolver may need to invoke another HandleResolver to resolve the GSH.



Relationship to Hosting Environment.

OGSI does not dictate a particular service-provider-side implementation architecture. A container implementation may provide a range of functionality beyond simple argument demarshaling.



The Grid Service

The purpose of the OGSI document is to specify the (standardized) interfaces and behaviours that define a *grid service.*

WSDL Extensions and Conventions

It uses WSDL as the mechanism to describe the public interfaces of grid services. er, WSDL 1.1 is deficient in two critical areas: lack of interface (portType) extension and the inability to describe additional information elements on a portType.

OGSI is based on Web services; in particular, it uses WSDL as the mechanism to describe the public interfaces of grid services.

Service Data

The approach to stateful Web services introduced in OGSI identified the need for a common mechanism to expose a service instance's state data to service requestors for query, update, and change notification.

Motivation and Comparison to JavaBean Properties

OGSI specification introduces the serviceData concept to provide a flexible, properties- style approach to accessing state data of a Web service. The serviceData concept is similar to the notion of a public instance variable or field in object-oriented programming languages such as Java, Smalltalk, and C++.

Extending portType with serviceData.

ServiceData defines a new portType child element named serviceData, used to define serviceData elements, or SDEs, associated with that portType. These serviceData element definitions are referred to as serviceData declarations, or SDDs.

ServiceDataValues.

Each service instance is associated with a collection of serviceData elements: those serviceData elements defined within the various portTypes that form the service's interface, and also,

potentially, additional service-Data elements added at runtime. OGSI calls the set of serviceData elements associated with a service instance its "serviceData set."
Each service instance must have a "logical" XML document, with a root element of serviceDataValues that contains the serviceData element values. An example of a serviceDataValues element was given above.

SDE Aggregation within a portType Interface Hierarchy.
WSDL 1.2 has introduced the notion of multiple portType extension, and one can model that construct within the GWSDL namespace. A portType can extend zero or more other portTypes. There is no direct relationship between a wsdl: service and the portTypes supported by the service modeled in the WSDL syntax."

Dynamic serviceData Elements.
The grid service portType illustrates the use of dynamic SDEs. This contains a serviceData element named "serviceDataName" that lists the serviceData elements currently defined.

Short notes on Core Grid Service Properties.
This subsection discusses a number of properties and concepts common to all grid services.

Service Description and Service Instance.
One can distinguish in OGSI between the *description*of a grid service and an *instance* of a grid service:
A *grid service description* describes how a client interacts with service instances. This description is independent of any particular instance. Within a WSDL document, the grid service description is embodied in the most derived portType of the instance, along with its associated portTypes bindings, messages, and types definitions.
A grid service description may be simultaneously used by any number of*grid service instances*, each of which

- Embodies some state with which the service description describes how to interact
- Has one or more grid service handles
- Has one or more grid service references to it

A service description is used primarily for **two purposes.**
First, as a description of a service interface, it can be used by tooling to automatically generate client interface proxies, server skeletons, and so forth.
Second, it can be used for discovery, for example, to find a service instance that implements a particular service description, or to find a factory that can create instances with a particular service description.
The service description is meant to capture both interface syntax and semantics. *Interface syntax* is described by WSDL portTypes. *Semantics* may be inferred through the name assigned to the portType.

Modeling Time in OGSI. The need arises at various points throughout this specification to represent time that is meaningful to multiple parties in the distributed Grid. Clients need to negotiate service instance lifetimes with services, and multiple services may need a common

understanding of time in order for clients to be able to manage their simultaneous use and interaction.

XML Element Lifetime Declaration Properties

Service Data elements may represent instantaneous observations of the dynamic state of a service instance, it is critical that consumers of serviceData be able to understand the valid lifetimes of these observations.

The three lifetime declaration properties are:

1.ogsi:goodFrom. Declares the time from which the content of the element is said to be valid. This is typically the time at which the value was created.

2. ogsi:goodUntil. Declares the time until which the content of the element is said to be valid. This property must be greater than or equal to the goodFrom time

3.ogsi:availableUntil. Declares the time until which this element itself is expected to be available, perhaps with updated values. Prior to this time, a client should be able to obtain an updated copy of this element

6. Explain Data intensive grid service models?

Data intensive grid service models

Applications in the grid are normally grouped into two categories

(i) Computation-intensive and

(ii) Data intensive

- Data intensive applications deals with massive amounts of data. The grid system must specially designed to discover, transfer and manipulate the massive data sets.
- Transferring the massive data sets is a time consuming task.
- Data access method is also known as caching, which is often applied to enhance data efficiency in a grid environment.
- By replicating the same data block and scattering them in multiple regions in a grid, users can access the same data.
- Replication strategies determine when and where to create a replica of the data.
- The strategies of replications can be classified into ***dynamic and static***

Static method

- The locations and number of replicas are determined in advance and will not be modified.
- Replication operation require little overhead
- Static strategic cannot adapt to changes in demand, bandwidth and storage variability
- Optimization is required to determine the location and number of data replicas.

Dynamic strategies

- Dynamic strategies can adjust locations and number of data replicas according to change in conditions
- Frequent data moving operations can result in much more overhead the static strategies
- Optimization may be determined based on whether the data replica is being created, deleted or moved.
- The most common replication include preserving locality, minimizing update

Data Replication and Unified Namespace

This data access method is also known as caching, which is often applied to enhance data efficiency in a grid environment. By replicating the same data blocks and scattering them in multiple regions of a grid, users can access the same data with locality of references. Furthermore, the replicas of the same data set can be a backup for one another. Some key data will not be lost in case of failures. The increase in storage requirements and network bandwidth may cause additional problems.

Replication strategies determine when and where to create a replica of the data. The factors to consider include data demand, network conditions, and transfer cost. The strategies of replication can be classified into method types: dynamic and static. Dynamic strategies can adjust locations and number of data replicas according to changes in conditions.

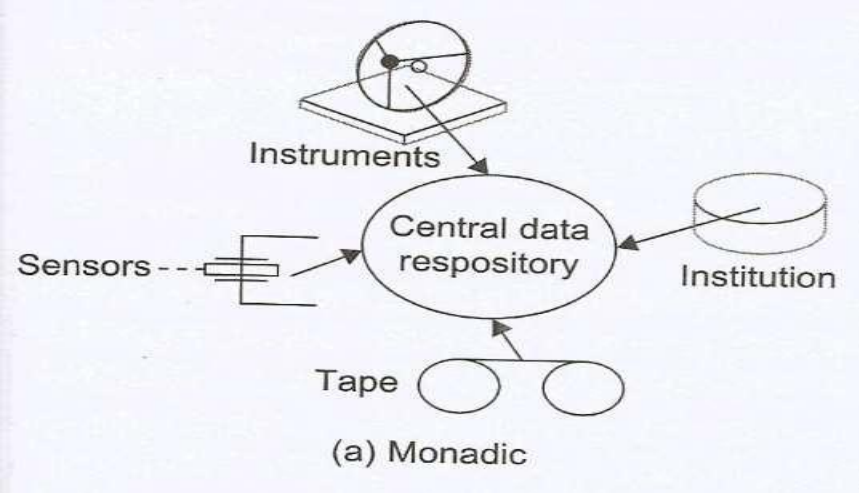
The most common replication strategies include preserving locality, minimizing update costs, and maximizing profits.

In general there are four access models for organizing a data grid as listed here

1. **Monadic method**
2. **Hierarchical model**
3. **Federation model**
4. **Hybrid model**

1. Monadic method

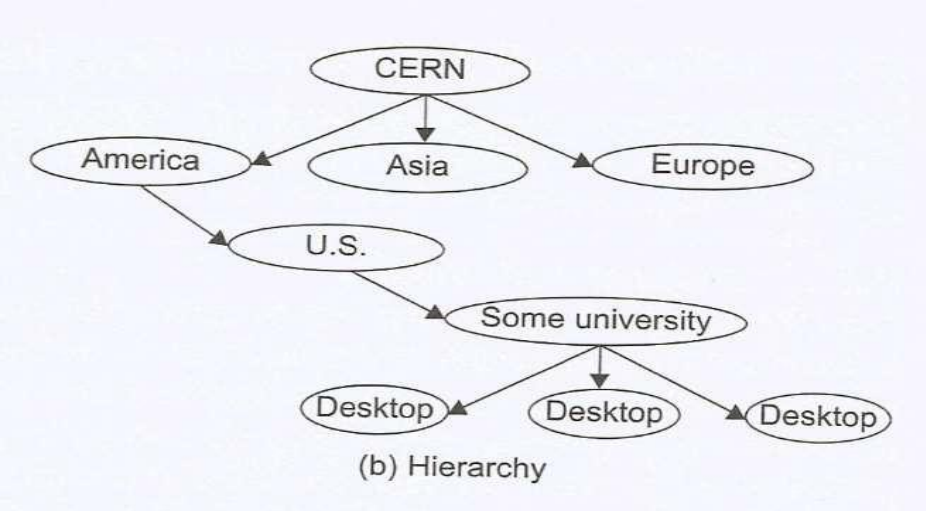
- This is a centralized data repository model. All data is saved in central data repository.
- When users want to access some data they have no submit request directly to the central repository.
- No data is replicated for preserving data locality.



- For a larger grid this model is not efficient in terms of performance and reliability.
- Data replication is permitted in this model only when fault tolerance is demanded.

Hierarchical model

- It is suitable for building a large data grid which has only one large data access directory
- Data may be transferred from the source to a second level center. Then some data in the regional center is transferred to the third level centre.

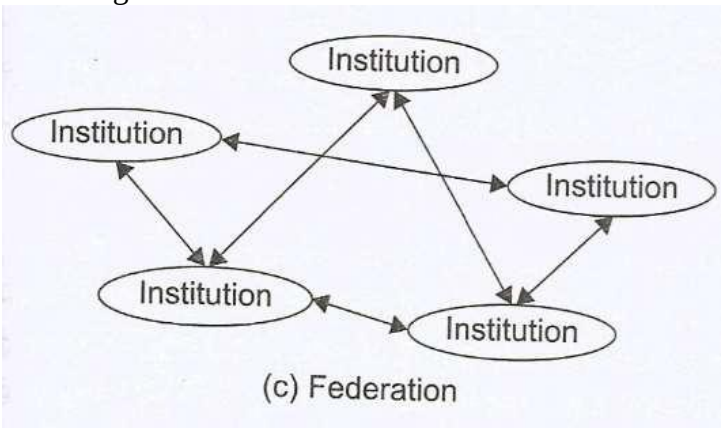


(b) Hierarchy

- After being forwarded several times specific data objects are accessed directly by users.
- Higher level data center has a wider coverage area.
- PKI security services are easier to implement in this hierarchical data access model

Federation model

- It is suited for designing a data grid with multiple source of data supplies.
- It is also known as a mesh model. The data is shared the data and items are owned and controlled by their original owners.

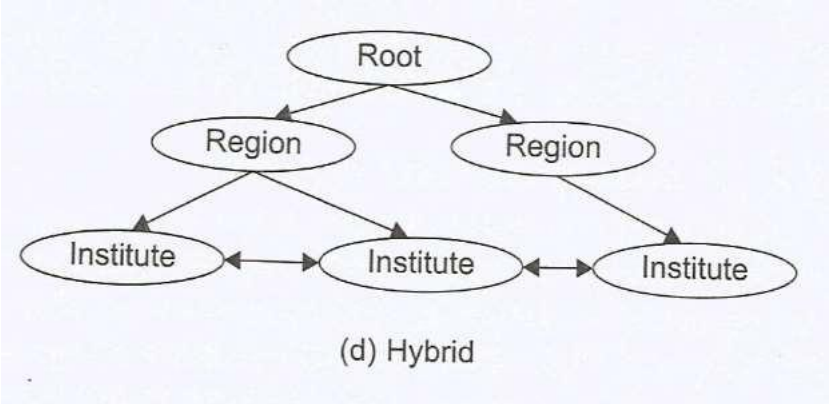


(c) Federation

- Only authenticated users are authorized to request data from any data source.
- This mesh model cost the most when the number of grid intuitions becomes very
- large

Hybrid model

- This model combines the best features of the hierarchical and mesh models.
- Traditional data transfer technology such as FTP applies for networks with lower bandwidth.
- High bandwidth are exploited by high speed data transfer tools such as GridFTP developed with Globus library.
- The cost of hybrid model can be traded off between the two extreme models of hierarchical and mesh connected grids.



Parallel versus Striped Data Transfers

- **Parallel data transfer** opens multiple data streams for passing subdivided segments of a file simultaneously. Although the speed of each stream is same as in sequential streaming, the total time to move data in all streams can be significantly reduced compared to FTP transfer.

- **Striped data transfer** a data objects is partitioned into a number of sections and each section is placed in an individual site in a data grid. When a user requests this piece of data, a data stream is created for each site in a data gird. When user requests this piece of data, data stream is created for each site, and all the sections of data objects ate transected simultaneously.

7. What are the various OGSA Services?

OGSA SERVICES

1. Handle Resolution
2. Virtual Organization Creation and Management
3. Service Groups and Discovery Services
4. Choreography, Orchestrations and Workflow
5. Transactions
6. Metering Service
7. Rating Service
8. Accounting Service
9. Billing and Payment Service
10. Installation, Deployment, and Provisioning
11. Distributed Logging
12. Messaging and Queuing
13. Event
14. Policy and Agreements
15. Base Data Services
16. Other Data Services
17. Discovery Services
18. Job Agreement Service

19. Reservation Agreement Service
20. Data Access Agreement Service
21. Queuing Service
22. Open Grid Services Infrastructure
23. Common Management Model

Handle Resolution

OGSI defines a two-level naming scheme for grid service instances based on abstract, long-lived grid service handles(GSHs) that can be mapped by HandleMapper services to concrete, but potentially less long lived, grid service references (GSRs). These constructs are basically network-wide pointers to specific grid service instances hosted in (potentially remote) execution environments.

The format of the GSH is a URL, where the schema directive indicates the naming scheme used to express the handle value. Based on the GSH naming scheme, the application should find an associated naming-scheme-specific HandleMapper service that knows how to resolve that name to the associated GSR.

OGSI defines the basic GSH format and portType for the HandleMapper service that resolve a GSH to a GSR.

The expectation is that different implementations of the two-level naming scheme with the naming directive will enable features such as transparent service instance migration, fault tolerance through transparent failover, high availability through mirroring, and advanced security through fine-grained access control on the name resolution.

Virtual Organization Creation and Management

VOs are a concept that supplies a "context" for operation of the grid that can be used to associate users, their requests, and resources. VO contexts permit the grid resource providers to associate appropriate policy and agreements with their resources.

Users associated with a VO can then exploit those resources consistent with those policies and agreements. VO creation and management functions include mechanisms for associating users/groups with a VO, manipulation of user roles (administration, configuration, use, etc.) within the VO, association of services (encapsulated resources) with the VO, and attachment of agreements and policies to the VO as a whole or to individual services within the VO.

Finally, creation of a VO requires a mechanism by which the "VO context" is referenced and associated with user requests (this is most likely via a GSH, since a "service" is the likely embodiment of the VO).

Service Groups and Discovery Services

GSHs and GSRs together realize a two-level naming scheme, with HandleResolver services mapping from handles to references; however, GSHs are not intended to contain semantic information and indeed may be viewed for most purposes as opaque.

Thus, other entities (both humans and applications) need other means for discovering services with particular properties, whether relating to interface, function, availability, location, policy, or other criteria.

It is important that OGSA defines standard functions for managing such name spaces, otherwise services and clients developed by different groups cannot easily discover each other's existence and properties. These functions must address the creation, maintenance, and querying of name mappings.

Two types of such semantic name spaces are common—naming by attribute, and naming by path. **Attribute naming** schemes associate various metadata with services and support retrieval via queries on attribute values.

A registry implementing such a scheme allows service providers to publish the existence and properties of the services that they provide, so that service consumers can discover them.

One can envision special-purpose registries being built on the base service group mechanisms provided by the OGSI definition. In other words, an OGSA-compliant registry is a concrete specialization of the OGSI service group.

A ServiceGroup is a collection of entries, where each entry is a grid service implementing the ServiceGroupEntry interface. The ServiceGroup interface also extends the GridService interface. There is a ServiceGroupEntry for each service in the group (i.e., for each group member).

Choreography, Orchestration, and Workflow

Over these interfaces OGSA provides a rich set of behaviors and associated operations and attributes for business process management (additional work remains to be done in this area):

- Definition of a job flow, including associated policies
- Assignment of resources to a grid flow instance
- Scheduling of grid flows (and associated grid services)
- Execution of grid flows (and associated grid services)
- Common context and metadata for grid flows (and associated services)
- Management and monitoring for grid flows (and associated grid services)
- Failure handling for grid flows (and associated grid services); more generally, managing the potential transiency of grid services
- Business transaction and coordination services

Transactions

Transaction services are important in many grid applications, particularly in industries such as financial services and in application domains such as supply chain management.

However, transaction management in a widely distributed, high-latency, heterogeneous RDBMS environment is more complicated than in a single data center with a single vendor's software.

Traditional distributed transaction algorithms, such as two-phase distributed commit, may be too expensive in a wide-area grid, and other techniques such as optimistic protocols may be more appropriate.

Metering Service

A metering interface provides access to a standard description of such aggregated data (metering serviceData).

A key parameter is the time window over which measurements are aggregated.

In commercial Unix systems, measurements are aggregated at administrator-defined intervals (chronological entry), usually daily, primarily for the purpose of accounting.

On the other hand, metering systems that drive active workload management systems might aggregate measurements using time windows measured in seconds.

Dynamic provisioning systems use time windows somewhere between these two examples. Several use cases require metering systems that support multitier, end-to-end flows involving multiple services

An OGSA metering service must be able to meter the resource consumption of configurable classes of these types of flows executing on widely distributed, loosely coupled server, storage, and network resources.

The metering service must be able to support the measurement of this class of service (resource) consumption.

Rating Service

A rating interface needs to address two types of behaviors.

Once the metered information is available, it has to be translated into financial terms. That is, for each unit of usage, a price has to be associated with it.

This step is accomplished by the rating interfaces, which provide operations that take the metered information and a rating package as input and output the usage in terms of chargeable amounts.

Accounting Service

Once the rated financial information is available, an accounting service can manage subscription users and accounts information, calculate the relevant monthly charges and maintain the invoice information.

This service can also generate and present invoices to the user. Account-specific information is also applied at this time.

Billing and Payment Service

Billing and payment service refers to the financial service that actually carries out the transfer of money; for example, a credit card authorization service.

Installation, Deployment, and Provisioning

Computer processors, applications, licenses, storage, networks, and instruments are all grid resources that require installation, deployment, and provisioning (other new resource types will be invented and added to this list.)

OGSA affords a framework that allows resource provisioning to be done in a uniform, consistent manner.

Distributed Logging

Distributed logging can be viewed as a typical messaging application in which message producers generate log artifacts, (atomic expressions of diagnostic information) that may or may not be used at a later time by other independent message consumers.

OGSA-based logging can leverage the notification mechanism available in OGSI as the transport for messages.

Logging services provide the extensions needed to deal with the following issues:

Decoupling.

The logical separation of logging artifact creation from logging artifact consumption. The ultimate usage of the data (e.g., logging, tracing, management) is determined by the message consumer; the message producer should not be concerned with this.

Transformation and common representation.

Logging packages commonly annotate the data that they generate with useful common information such as category, priority, time stamp, and location.

An OGSA logging service should not only provide the capability of annotating data, but also the

Also, a general mechanism for transformation may be required. Filtering and aggregation. The amount of logging data generated can be large, whereas the amount of data actually consumed can be small. Therefore, it can be desirable to have a mechanism for controlling the amount of data generated and for filtering out what is actually kept and where. Through the use of different filters, data coming from a single source can be easily separated into different repositories, and/or “similar” data coming from different sources can be aggregated into a single repository.

Configurable persistency.

Depending on consumer needs, data may have different durability characteristics. For example, in a real-time monitoring application, data may become irrelevant quickly, but be needed as soon as it is generated; data for an auditing program may be needed months or even years after it was generated. Hence, there is a need for a mechanism to create different data repositories, each with its own persistency characteristics. In addition, the artifact retention policy (e.g., determining which log artifacts to drop when a buffer reaches its size limit) should be configurable. Consumption patterns. Consumption patterns differ according to the needs of the consumer application. For example, a real-time monitoring application needs to be notified whenever a particular event occurs, whereas a postmortem problem determination program queries historical data, trying to find known patterns. Thus, the logging repository should support both synchronous query- (pull-) based consumption and asynchronous push-based (eventdriven) notification. The system should be flexible enough that consumers can easily customize the event mechanism—for example, by sending digests of messages instead of each one—and maybe even provide some predicate logic on log artifacts to drive the notifications.

These considerations lead to an architecture for OGSA logging services in which producers talk to filtering and transformation services either directly, or indirectly through adapters. Consumers also use this service to create custom message repositories (baskets) or look for existing producers and baskets; that is, this service should also function as a factory (of baskets) and a registry (of producers and baskets). There is also a need for a configurable storage and delivery service, with which data from different filtering services is collected, stored, and, if required, delivered to interested consumers.

UNIT III VIRTUALIZATION

Cloud deployment models: public, private, hybrid, community – Categories of cloud computing: Everything as a service: Infrastructure, platform, software - Pros and Cons of cloud computing – Implementation levels of virtualization – virtualization structure – virtualization of CPU, Memory and I/O devices – virtual clusters and Resource Management – Virtualization for data center automation.

PART A

1. Define cloud computing?

The NIST definition"Cloud computing" a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

The NIST definition also identifies

- 5 essential characteristics
- 3 service models
- 4 deployment models

2. What is public cloud?

A public cloud is built over the Internet and can be accessed by any user who has paid for the service. Public clouds are owned by service providers and are accessible through a subscription. Many public clouds are available, including Google App Engine (GAE),Amazon Web Services (AWS), Microsoft Azure, IBM Blue Cloud, and Salesforce.com's Force.com.

3. What is private cloud?

A private cloud is built within the domain of an intranet owned by a single organization.Therefore, it is client owned and managed, and its access is limited to the owning clients and their partners.

4. Highlights six design objectives for cloud computing

- Shifting computing from desktops to data centers
- Service provisioning and cloud economics
- Scalability in performance
- Data privacy protection
- High quality of cloud services
- New standards and interfaces

5. What are the essential characteristics of cloud computing?

On-demand self services, Broad network access, Resource pooling, Rapid elasticity, Measured service.

6. What are the advantages of cloud computing?

Cost efficient, almost unlimited storage, backup and recovery, automatic software integration, easy access to information, quick development.

7. What are the disadvantages of cloud computing?

Technical issues, security in the cloud, prone to attack.

8. What is meant by virtualization ?

Virtualization is a computer architecture technology by which multiple virtual machines (VMs) are multiplexed in the same hardware machine. The purpose of a VM is to enhance resource sharing by many users and improve computer performance in terms of resource utilization and application flexibility.

9. What are the levels of virtualization ?

- Instruction set architecture (ISA) level,
- hardware level,
- operating system level,
- library support level, and
- application level

10. What are the classes of VM architecture

The hypervisor architecture are para-virtualization, and host-based virtualization.The hypervisor is also known as the VMM (Virtual Machine Monitor).

11. What is meant by XEN hypervisor

Xen is an open source hypervisor program developed by CambridgeUniversity. Xen is a micro-kernel hypervisor, which separates the policy from the mechanism. The Xen hypervisor implements all the mechanisms, leaving the policy to be handled by Domain 0.

12. What is meant by Full Hypervisor ?

- Not able to modify guest OS
- Noncritical instructions run on the hardware directly while critical instructions are discovered and replaced with traps into the VMM to be emulated by software. Both the hypervisor and VMM approaches are considered full virtualization.

Example : Xen

13. What is meant by Para-virtualization ?

Para-virtualization needs to modify the guest operating systems. A para-virtualized VM provides special APIs requiring substantial OS modifications in user applications.Performance degradation is a critical issue of a virtualized system.

Example: KVM, VMWare ESX

14. What are the categories of instructions?

privileged instructions, control-sensitive instructions, and behavior-sensitive instructions.

15. What **are** the properties of Cloud Computing?

There are six key properties of cloud computing:

Cloud computing is

• user-centric

• task-centric
• powerful
• accessible
• intelligent
• programmable

16. Differentiate Physical versus Virtual Clusters ?

Virtual clusters are built with VMs installed at distributed servers from one or more physical clusters. The VMs in a virtual cluster are interconnected logically by a virtual network across several physical networks. Each virtual cluster is formed with physical machines or a VM hosted by multiple physical clusters.

17. What are are four steps to deploy a group of VMs onto a target cluster?

Preparing the disk image, configuring the VMs, choosing the destination nodes, and executing the VM deployment command on every host.

18. State the states of VM ?

- An inactive state is defined by the virtualization platform, under which the VM is not enabled.
- An active state refers to a VM that has been instantiated at the virtualization platform to perform a real task.
- A paused state corresponds to a VM that has been instantiated but disabled to process a task or paused in a waiting state.
- A VM enters the suspended state if its machine file and virtual resources are stored back to the disk.

19. What are the side effects of server virtualization ?

• Consolidation enhances hardware utilization
• This approach enables more agile provisioning and deployment of resources.
• The total cost of ownership is reduced.
• This approach improves availability and business continuity.

- It becomes easier to transfer a VM from one server to another, because virtual servers are unaware of the underlying hardware.

PART – B

1.Explain the NIST reference architecture of cloud computing in detail

The Conceptual Reference Model Figure 1 presents an overview of the NIST cloud computing reference architecture, which identifies the major actors, their activities and functions in cloud computing. The diagram depicts a generic high-level architecture and is intended to facilitate the understanding of the requirements, uses, characteristics and standards of cloud computing.

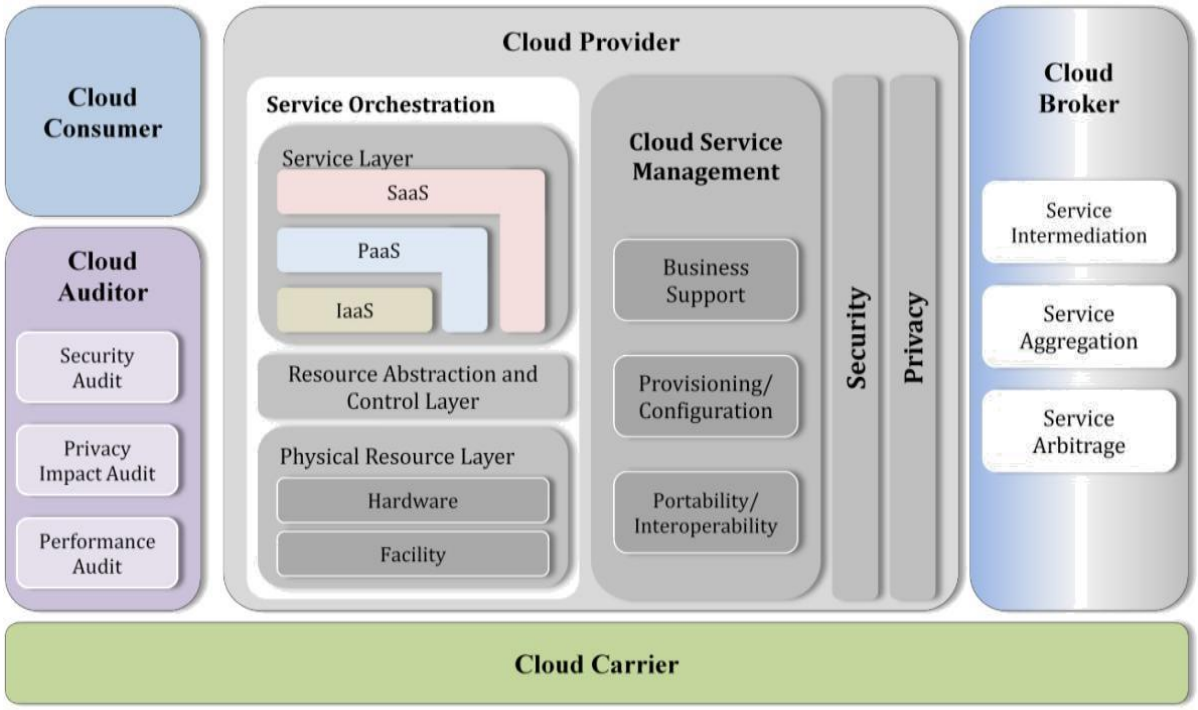


Figure 1: The Conceptual Reference Model

As shown in Figure 1, the NIST cloud computing reference architecture defines five major actors: cloud consumer, cloud provider, cloud carrier, cloud auditor and cloud broker. Each actor is an entity (a person or an organization) that participates in a transaction or process and/or performs tasks in cloud computing.

Actor	Definition
Cloud Consumer	A person or organization that maintains a business relationship with, and uses service from, *Cloud Providers*.
Cloud Provider	A person, organization, or entity responsible for making a service available to interested parties.
Cloud Auditor	A party that can conduct independent assessment of cloud services, information system operations, performance and security of the cloud implementation.
Cloud Broker	An entity that manages the use, performance and delivery of cloud services, and negotiates relationships between *Cloud Providers* and *Cloud Consumers.*
Cloud Carrier	An intermediary that provides connectivity and transport of cloud services from *Cloud Providers* to *Cloud Consumers*.

2.Explain about cloud deployment models and cloud service models.

The NIST definition "Cloud computing"

It is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction."

- The NIST definition also identifies
    - 5 essential characteristics
    - 3 service models
    - 4 deployment models

4 deployment models / Types of cloud

- **Public**: Accessible, via the Internet, to anyone who pays

– Owned by service providers; e.g., Google App Engine, Amazon Web Services, Force.com.

A *public cloud* is a publicly accessible cloud environment owned by a third-party cloud provider. The IT resources on public clouds are usually provisioned via the previously described cloud delivery models and are generally offered to cloud consumers at a cost or are commercialized via other avenues (such as advertisement).

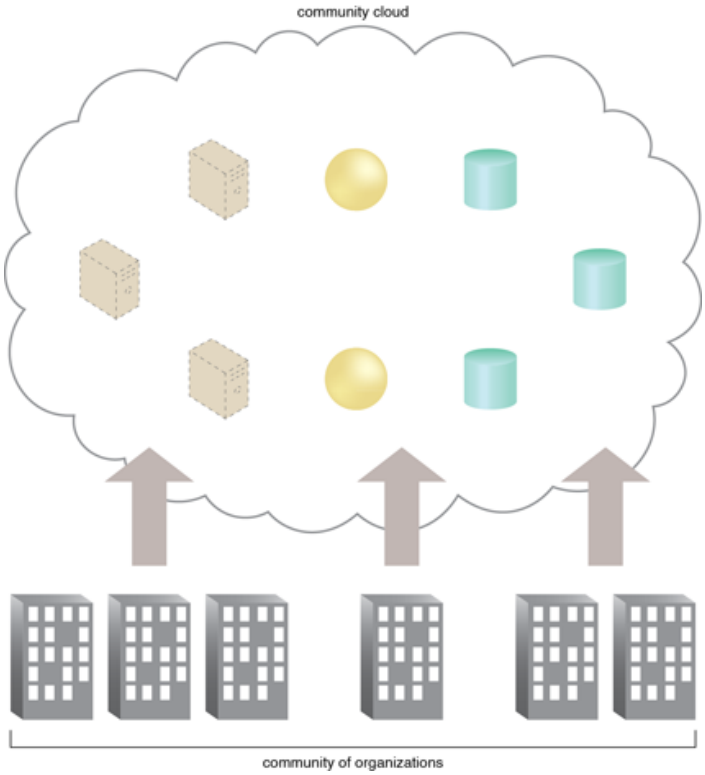
The cloud provider is responsible for the creation and on-going maintenance of the public cloud and its IT resources. Many of the scenarios and architectures explored in upcoming chapters involve public clouds and the relationship between the providers and consumers of IT resources via public clouds.

Figure 1 shows a partial view of the public cloud landscape, highlighting some of the primary vendors in the marketplace.

- **Community**: Shared by two or more organizations with joint interests, such as colleges within a university

  A community cloud is similar to a public cloud except that its access is limited to a specific community of cloud consumers. The community cloud may be jointly owned by the community members or by a third-party cloud provider that provisions a public cloud with limited access. The member cloud consumers of the community typically share the responsibility for defining and evolving the community cloud (Figure 1).

  Membership in the community does not necessarily guarantee access to or control of all the cloud's IT resources. Parties outside the community are generally not granted access unless allowed by the community.

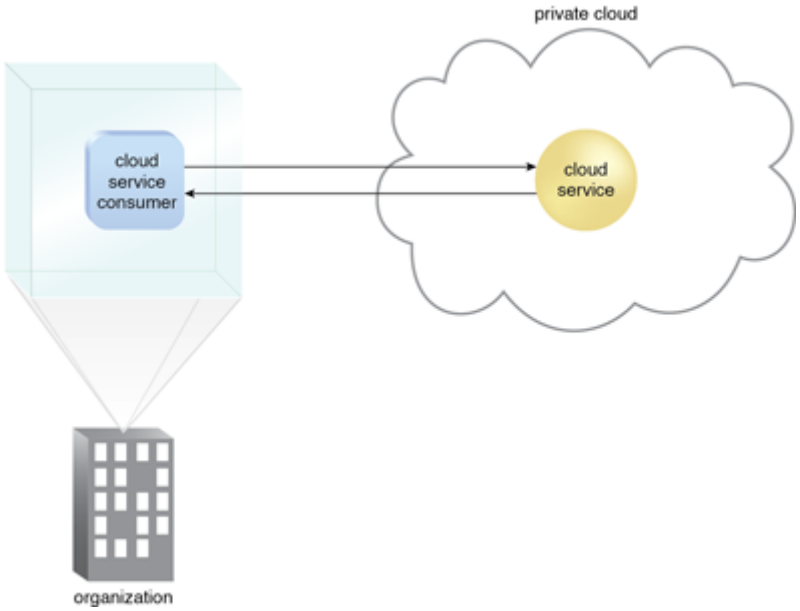


An example of a "community" of organizations accessing IT resources from a community cloud.

- **Private**: Accessible via an intranet to the members of the owning organization
  - Can be built using open source software such as CloudStack or OpenStack
  - Example of private cloud: NASA's cloud for climate modeling

A private cloud is owned by a single organization. Private clouds enable an organization to use cloud computing technology as a means of centralizing access to IT resources by different parts, locations, or departments of the organization. When a private cloud exists as a controlled environment, the problems described in the Risks and Challenges section do not tend to apply.

The use of a private cloud can change how organizational and trust boundaries are defined and applied. The actual administration of a private cloud environment may be carried out by internal or outsourced staff.



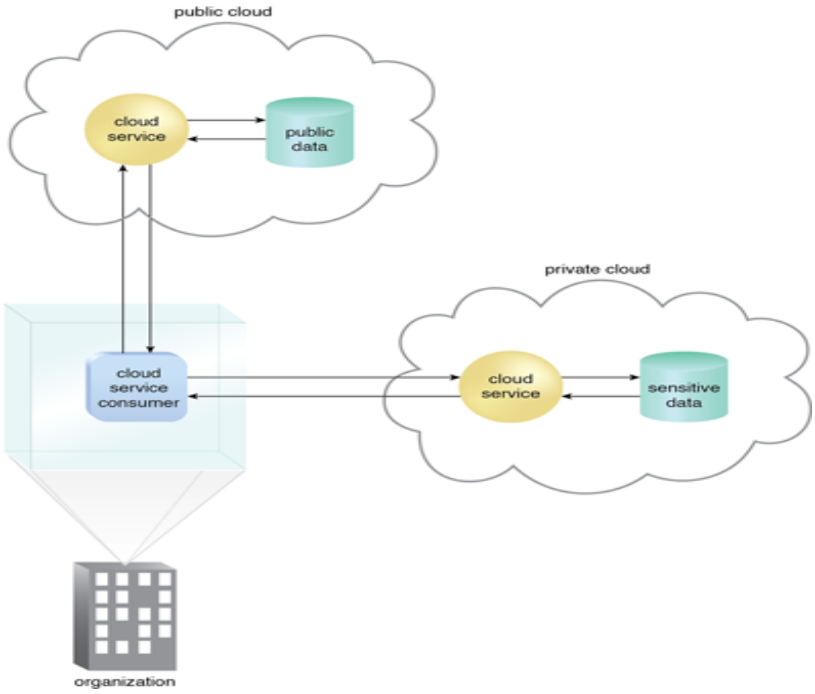
A cloud service consumer in the organization's on-premise environment accesses a cloud service hosted on the same organization's private cloud via a virtual private network.

With a private cloud, the same organization is technically both the cloud consumer and cloud provider . In order to differentiate these roles:

- a separate organizational department typically assumes the responsibility for provisioning the cloud (and therefore assumes the cloud provider role)
- departments requiring access to the private cloud assume the cloud consumer role

- **Hybrid**

A hybrid cloud is a cloud environment comprised of two or more different cloud deployment models. For example, a cloud consumer may choose to deploy cloud services processing sensitive data to a private cloud and other, less sensitive cloud services to a public cloud. The result of this combination is a hybrid deployment model.



An organization using a hybrid cloud architecture that utilizes both a private and public cloud.

Hybrid deployment architectures can be complex and challenging to create and maintain due to the potential disparity in cloud environments and the fact that management responsibilities are typically split between the private cloud provider organization and the public cloud provider.

Three service models / Categories of cloud computing.

- Software as a Service (SaaS)
  – Use provider's applications over a network
- Platform as a Service (PaaS)
  – Deploy customer-created applications to a cloud
- Infrastructure as a Service (IaaS)

- *Software as a Service (SaaS).* The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through a thin client interface such as a web browser (e.g., web-based email). The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of provider-defined user-specific application configuration settings.

- *Platform as a Service (PaaS).* The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly application hosting environment configurations.
- *Infrastructure as a Service (IaaS).* The capability provided to the consumer is to provision processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications. The consumer does not manage or control the underlying cloud physical infrastructure but has control over operating systems, storage, deployed applications, and possibly limited control of select networking components.

Comparison of cloud service models

Type	Consumer	Service Provided By Cloud	Service Level Coverage	Customization
SaaS	End user	• Finished application	• Application uptime • Application Performance	• Minimal to no customization • Capabilities dictated by market or provider
PaaS	Application owner	• Runtime environment for application code • Cloud storage • Other Cloud services such as integration	• Environment availability • Environment performance • No application coverage	• High degree of application level customization available within constraints of the service offered • Many applications will need to be rewritten
IaaS	Application owner or IT provides OS, middleware and application support	• Virtual server • Cloud storage	• Virtual server availability • Time to provision • No platform or application coverage	• Minimal constraints on applications installed on standardized virtual OS builds

3. Listout the essential characteristics and Design Objectives of cloud?

5 Essential characteristics

- On-demand self-service: consumers can acquire the necessary computational resources without having to interact with human service providers.
- Ubiquitous network access: cloud features don't require special devices – laptops, mobile phones, etc. are generally supported.
- Resource pooling: cloud resources are pooled to serve many customers "… using a multi-tenant model, with different physical and virtual resources…"
- Rapid elasticity: resources can be allocated and de-allocated quickly as needed.
- Measured service: resource use is measured and monitored; charges are made based on usage and service type (e.g., storage, CPU cycles, etc.)

Cloud Design Objectives

The following list highlights six design objectives for cloud computing:

• **Shifting computing from desktops to data centers** Computer processing, storage, and software delivery is shifted away from desktops and local servers and toward data centers over the Internet.

• **Service provisioning and cloud economics** Providers supply cloud services by signing SLAs with consumers and end users. The services must be efficient in terms of computing, storage, and power consumption. Pricing is based on a pay-as-you-go policy.

• **Scalability in performance** the cloud platforms and software and infrastructure services must be able to scale in performance as the number of users increases.

• **Data privacy protection** Can you trust data centers to handle your private data and Records? This concern must be addressed to make clouds successful as trusted services.

• **High quality of cloud services** The QoS of cloud computing must be standardized to make clouds interoperable among multiple providers.

• **New standards and interfaces** this refers to solving the data lock-in problem associated with data centers or cloud providers. Universally accepted APIs and access Protocols are needed to provide high portability and flexibility of virtualized applications.

4. Listout the advantages and disadvantages of cloud / Pros and cons of cloud.

Advantages of Cloud Computing

- Cost Savings : Perhaps, the most significant cloud computing benefit is in terms of IT cost savings. Businesses, no matter what their type or size, exist to earn money while keeping capital and operational expenses to a minimum. With cloud computing, you can save substantial capital costs with zero in-house server storage and application requirements. The lack of on-premises infrastructure also removes their associated operational costs in the form of power, air conditioning and administration costs. You pay for what is used and disengage whenever you like - there is no invested IT capital to worry about. It's a common misconception that only large businesses can afford to use the cloud, when in fact, cloud services are extremely affordable for smaller businesses.
- Reliability: With a managed service platform, cloud computing is much more reliable and consistent than in-house IT infrastructure. Most providers offer a Service Level Agreement which guarantees 24/7/365 and 99.99% availability. Your organization can benefit from a massive pool of redundant IT resources, as well as quick failover mechanism - if a server fails, hosted applications and services can easily be transited to any of the available servers.
- Manageability :Cloud computing provides enhanced and simplified IT management and maintenance capabilities through central administration of resources, vendor managed infrastructure and SLA backed agreements. IT infrastructure updates and maintenance are eliminated, as all resources are maintained by the service provider. You enjoy a simple web-based user interface for accessing software, applications and services – without the need for installation - and an SLA ensures the timely and guaranteed delivery, management and maintenance of your IT services.
- Lower computer costs:
    - You do not need a high-powered and high-priced computer to run cloud computing's web-based applications.
    - Since applications run in the cloud, not on the desktop PC, your desktop PC does not need the processing power or hard disk space demanded by traditional desktop software.

    - When you are using web-based applications, your PC can be less expensive, with a smaller hard disk, less memory, more efficient processor...
    - In fact, your PC in this scenario does not even need a CD or DVD drive, as no software programs have to be loaded and no document files need to be saved.
- Improved performance:
    - With few large programs hogging your computer's memory, you will see better performance from your PC.
    - Computers in a cloud computing system boot and run faster because they have fewer programs and processes loaded into memory…
- Reduced software costs:
    - Instead of purchasing expensive software applications, you can get most of what you need for free-ish!
        - most cloud computing applications today, such as the Google Docs suite.
    - better than paying for similar commercial software
        - which alone may be justification for switching to cloud applications.
- Instant software updates:
    - Another advantage to cloud computing is that you are no longer faced with choosing between obsolete software and high upgrade costs.
    - When the application is web-based, updates happen automatically
        - available the next time you log into the cloud.
    - When you access a web-based application, you get the latest version
        - without needing to pay for or download an upgrade.
- Unlimited storage capacity:
    - Cloud computing offers virtually limitless storage.
    - Your computer's current 1 Tbyte hard drive is small compared to the hundreds of Pbytes available in the cloud.
- Increased data reliability:
    - Unlike desktop computing, in which if a hard disk crashes and destroy all your valuable data, a computer crashing in the cloud should not affect the storage of your data.

        - if your personal computer crashes, all your data is still out there in the cloud, still accessible
    - In a world where few individual desktop PC users back up their data on a regular basis, cloud computing is a

data-safe computing platform!

- Universal document access:
    - That is not a problem with cloud computing, because you do not take your documents with you.
    - Instead, they stay in the cloud, and you can access them whenever you have a computer and an Internet connection
    - Documents are instantly available from wherever you are
- Latest version availability:
    - When you edit a document at home, that edited version is what you see when you access the document at work.
    - The cloud always hosts the latest version of your documents
        - as long as you are connected, you are not in danger of having an outdated version
- Easier group collaboration:
    - Sharing documents leads directly to better collaboration.
    - Many users do this as it is an important advantages of cloud computing
        - multiple users can collaborate easily on documents and projects
- Device independence.
    - You are no longer tethered to a single computer or network.
    - Changes to computers, applications and documents follow you through the cloud.
    - Move to a portable device, and your applications and documents are still available.

Disadvantage of cloud computing

- Downtime : As cloud service providers take care of a number of clients each day, they can become overwhelmed and may even come up against technical outages. This can lead to your business processes being temporarily suspended. Additionally, if your

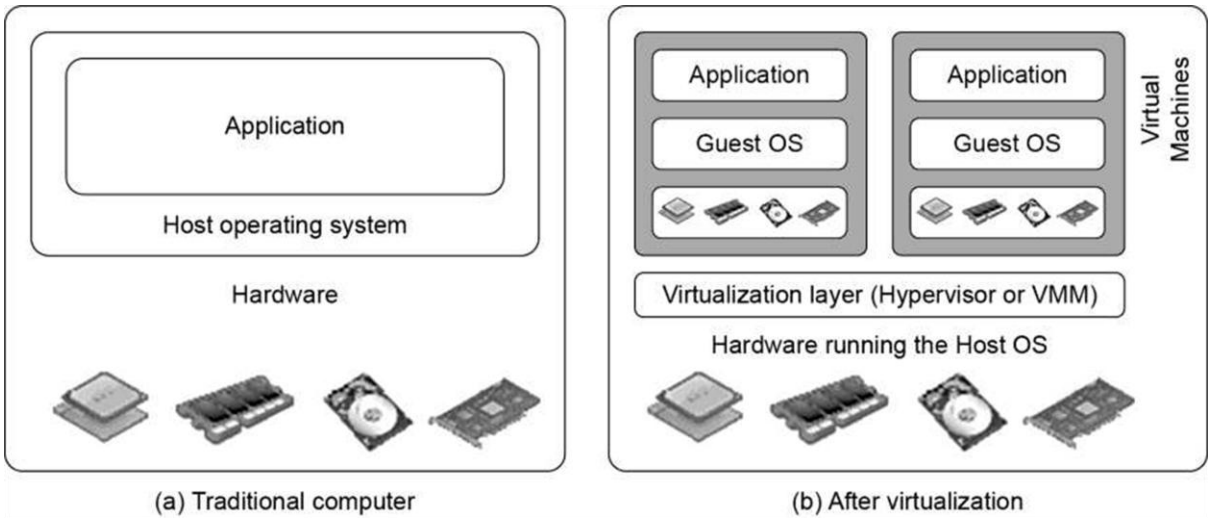
internet connection is offline, you will not be able to access any of your applications, server or data from the cloud.

- Security :Although cloud service providers implement the best security standards and industry certifications, storing data and important files on external service providers always opens up risks. Using cloud-powered technologies means you need to provide your service provider with access to important business data. Meanwhile, being a public service opens up cloud service providers to security challenges on a routine basis. The ease in procuring and accessing cloud services can also give nefarious users the ability to scan, identify and exploit loopholes and vulnerabilities within a system. For instance, in a multi- tenant cloud architecture where multiple users are hosted on the same server, a hacker might try to break into the data of other users hosted and stored on the same server. However, such exploits and loopholes are not likely to surface, and the likelihood of a compromise is not great.
- Vendor Lock-In: Although cloud service providers promise that the cloud will be flexible to use and integrate, switching cloud services is something that hasn't yet completely evolved. Organizations may find it difficult to migrate their services from one vendor to another. Hosting and integrating current cloud applications on another platform may throw up interoperability and support issues. For instance, applications developed on Microsoft Development Framework (.Net) might not work properly on the Linux platform.
- Requires a constant Internet connection:
    - Cloud computing is impossible if you cannot connect to the Internet.
    - Since you use the Internet to connect to both your applications and documents, if you do not have an Internet connection you cannot access anything, even your own documents.
    - A dead Internet connection means no work and in areas where Internet connections are few or inherently unreliable, this could be a deal-breaker.
- Can be slow:
    - Even with a fast connection, web-based applications can sometimes be slower than accessing a similar software program on your desktop PC.

    - Everything about the program, from the interface to the current document, has to be sent back and forth from your computer to the computers in the cloud.
    - If the cloud servers happen to be backed up at that moment, or if the Internet is having a slow day, you would not get the instantaneous access you might expect from desktop applications.
- Does not work well with low-speed connections:
    - Similarly, a low-speed Internet connection, such as that found with dial-up services, makes cloud computing painful at best and often impossible.
    - Web-based applications require a lot of bandwidth to download, as do large documents.
- Features might be limited:
    - This situation is bound to change, but today many web-based applications simply are not as full-featured as their desktop-based applications.
        - For example, you can do a lot more with Microsoft PowerPoint than with Google Presentation's web-based
- Stored data might not be secure:
    - With cloud computing, all your data is stored on the cloud.
        - The questions is How secure is the cloud?
    - Can unauthorized users gain access to your confidential data?
- Stored data can be lost:
    - Theoretically, data stored in the cloud is safe, replicated across multiple machines.
    - But on the off chance that your data goes missing, you have no physical or local backup.
        - Put simply, relying on the cloud puts you at risk if the cloud lets you down.
- HPC Systems:
    - Not clear that you can run compute-intensive HPC applications that use MPI/OpenMP!
    - Scheduling is important with this type of application
        - as you want all the VM to be co-located to minimize communication latency!
- General Concerns:
    - Each cloud systems uses different protocols and different APIs
        - may not be possible to run applications between cloud based systems

5. Explain Implementation levels of virtualization in detail

- A traditional computer runs with a host operating system specially tailored for its hardware architecture.
- After virtualization, different user applications managed by their own operating systems (guest OS) can run on the same hardware, independent of the host OS. This is often done by adding additional software, called a virtualization layer. This virtualization layer is known as hypervisor or virtual machine monitor (VMM).
- The VMs are shown in the upper boxes, where applications run with their own guest OS over the virtualized CPU, memory, and I/O resources.



(a) Traditional computer (b) After virtualization

The main function of the software layer for virtualization is to virtualize the physical hardware of a host machine into virtual resources to be used by the VMs, exclusively. This can be implemented at various operational levels, as we will discuss shortly.

The Virtualization software creates the abstraction of VMs by interposing a virtualization layer at various levels of a computer system.

Common virtualization layers include

- Instruction set architecture (ISA) level,
- hardware level,
- operating system level,
- library support level, and
- application level

<u>Five abstraction levels</u>

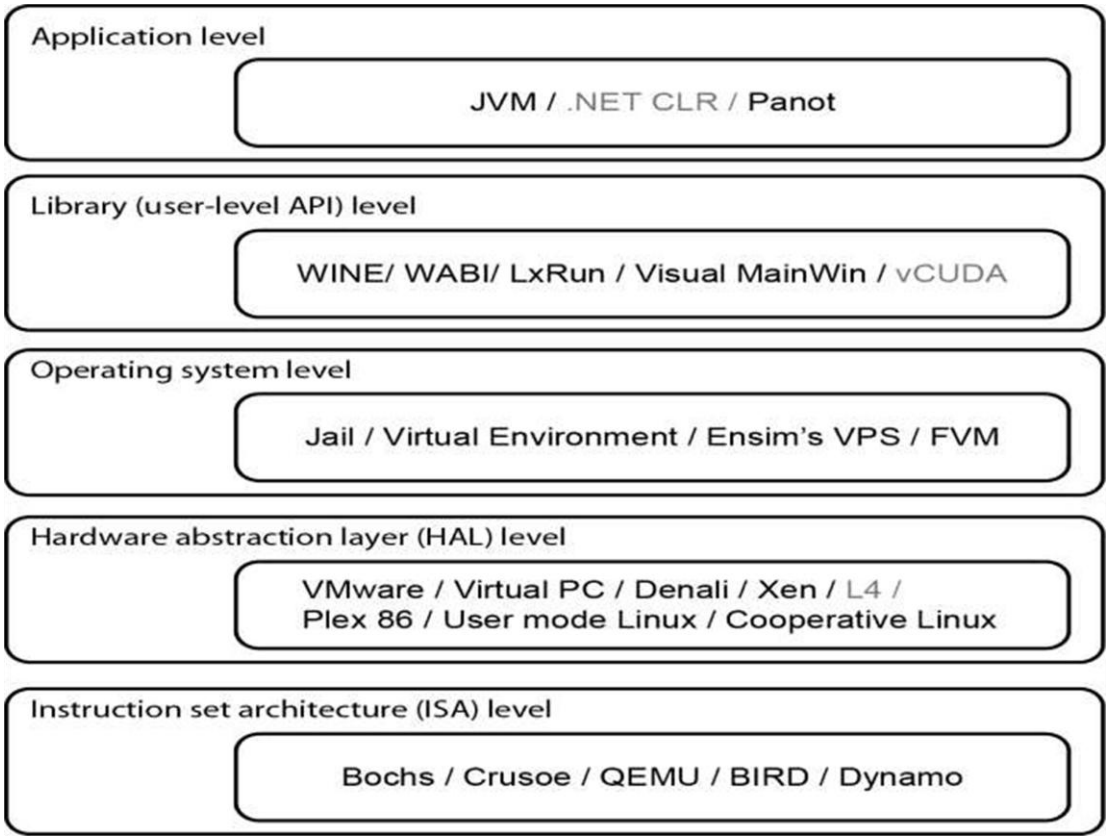


Figure: Virtualization ranging from hardware to applications in five abstraction levels

Instruction Set Architecture Level

- At the ISA level, virtualization is performed by emulating a given ISA by the ISA of the host machine. For example, MIPS binary code can run on an x86-based host machine with the help of ISA emulation.
- The basic emulation method is through code interpretation. An interpreter program interprets the source instructions to target instructions one by one. One source instruction may require tens or hundreds of native target instructions to perform its function.
- This approach translates basic blocks of dynamic source instructions to target instructions. The basic blocks can also be extended to program traces or super blocks to increase translation efficiency. Instruction set emulation requires binary translation and optimization. A virtual instruction set architecture (V-ISA) thus requires adding a processor-specific software translation layer to the compiler.

Hardware Abstraction Level

- Hardware-level virtualization is performed right on top of the bare hardware. On the one hand, this approach generates a virtual hardware environment for a VM. On the other hand, the process manages the underlying hardware through virtualization.
- The idea is to virtualize a computer's resources, such as its processors, memory, and I/O devices. The intention is to upgrade the hardware utilization rate by multiple users concurrently. The idea was implemented in the IBM VM/370 in the 1960s. More recently, the Xen hypervisor has been applied to virtualize x86-based machines to run Linux or other guest OS.

Operating System Level

This refers to an abstraction layer between traditional OS and user applications. OS-level virtualization creates isolated containers on a single physical server and the OS instances to utilize the hardware and software in data centers. The containers behave like real servers. OS-level virtualization is commonly used in creating virtual hosting environments to allocate hardware resources among a large number of mutually distrusting users.

Advantages of OS Extension for Virtualization

1. VMs at OS level has minimum startup/shutdown costs
2. OS-level VM can easily synchronize with its environment

Disadvantage of OS Extension for Virtualization

- All VMs in the same OS container must have the same or similar guest OS, which restrict application flexibility of different VMs on the same physical machine.

Library Support Level

- Most applications use APIs exported by user-level libraries rather than using lengthy system calls by the OS. Since most systems provide well-documented APIs, such an interface becomes another candidate for virtualization.
- Virtualization with library interfaces is possible by controlling the communication link between applications and the rest of a system through API hooks. The software tool WINE has implemented this approach to support Windows applications on top of UNIX hosts.

User-Application Level

Virtualization at the application level virtualizes an application as a VM. On a traditional OS, an application often runs as a process. Therefore, application-level virtualization is also known as process-level virtualization.

- The most popular approach is to deploy high level language (HLL) VMs. In this scenario, the virtualization layer sits as an application program on top of the operating system, and the layer exports an abstraction of a VM that can run programs written and compiled to a particular abstract machine definition.
- Any program written in the HLL and compiled for this VM will be able to run on it. The Microsoft .NET CLR and Java Virtual Machine (JVM) are two good examples of this class of VM.

6. Explain Virtualization structures/tools and mechanisms in detail.

Figure showed the architectures of a machine before and after virtualization.

- Before virtualization, the operating system manages the hardware.
- After virtualization, a virtualization layer is inserted between the hardware and the operating system. In such a case, the virtualization layer is responsible for converting portions of the real hardware into virtual hardware.
- Therefore, different operating systems such as Linux and Windows can run on the

same physical machine, simultaneously.

- Depending on the position of the virtualization layer, there are several classes of VM architectures, namely the hypervisor architecture, para-virtualization, and host- based virtualization.
- The hypervisor is also known as the VMM (Virtual Machine Monitor). They both perform the same virtualization operations.

Hypervisor

- A hypervisor is a hardware virtualization technique allowing multiple operating systems, called guests to run on a host machine. This is also called the Virtual Machine Monitor (VMM).

Type 1: bare metal hypervisor

- sits on the bare metal computer hardware like the CPU, memory, etc.
- All guest operating systems are a layer above the hypervisor.

- The original CP/CMS hypervisor developed by IBM was of this kind.

Type 2: hosted hypervisor

- Run over a host operating system.
- Hypervisor is the second layer over the hardware.
- Guest operating systems run a layer over the hypervisor.
- The os is usually unware of the virtualization

The XEN Architecture

- Xen is an open source hypervisor program developed by Cambridge University. Xen is a micro- kernel hypervisor, which separates the policy from the mechanism.
- Xen does not include any device drivers natively . I t just provides a mechanism by which a guest OS can have direct access to the physical devices.
- As a result, the size of the Xen hypervisor is kept rather small. Xen provides a

Virtual environment located between the hardware and OS.

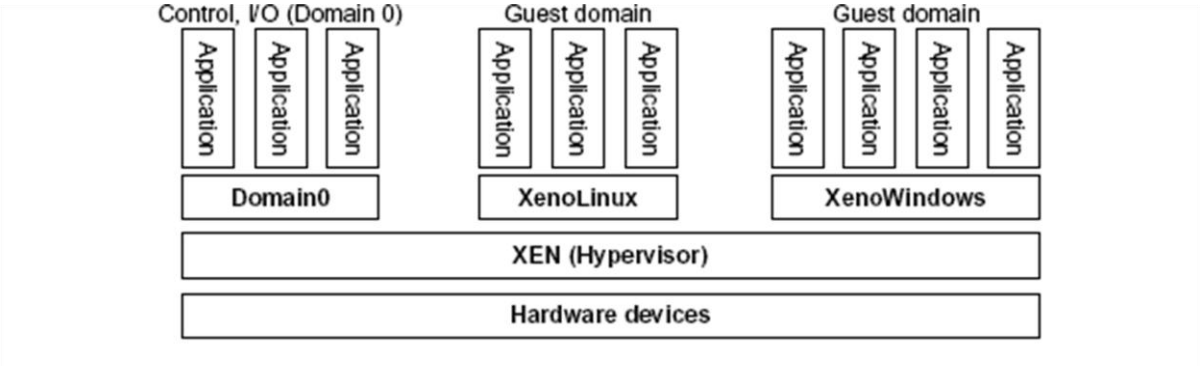


FIGURE 3.5

The Xen architecture's special domain 0 for control and I/O, and several guest domains for user applications.

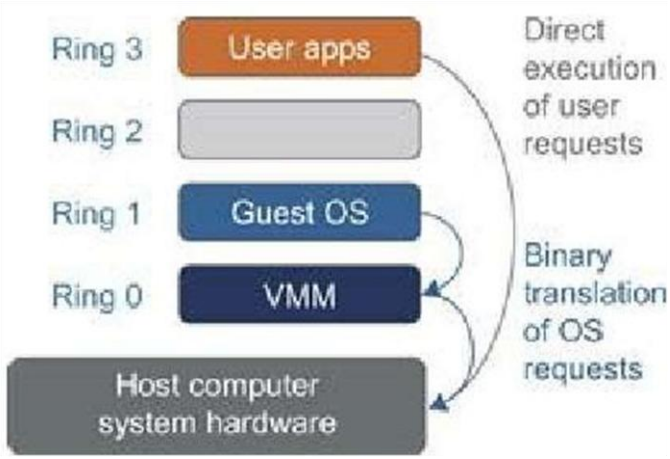
Binary Translation with Full Virtualization

- Depending on implementation technologies, hardware virtualization can be classified into two categories: full virtualization and host-based virtualization.
- Full virtualization does not need to modify the host OS. I t relies on binary translation to trap and to virtualizes the execution of certain sensitive, non virtualizable instructions. The guest OSes and their applications consist of noncritical and critical instructions.
- I n a host-based system, both a host OS and a guest OS are used. A virtualization software layer is built between the host OS and guest OS.

- These two classes of VM architecture are introduced next.

Binary Translation of Guest OS Requests Using a VMM

- This approach was implemented by VMware and many other software companies.
- VMware puts the VMM at Ring 0 and the guest OS at Ring 1. The VMM scans the instruction stream and identified the privileged, control- and behavior sensitive instructions.
- When these instructions are identified, they are trapped into the VMM, which emulates the behavior of these instructions.
- The method used in this emulation is called binary translation. Therefore, full virtulation combines binary translation and direct execution.



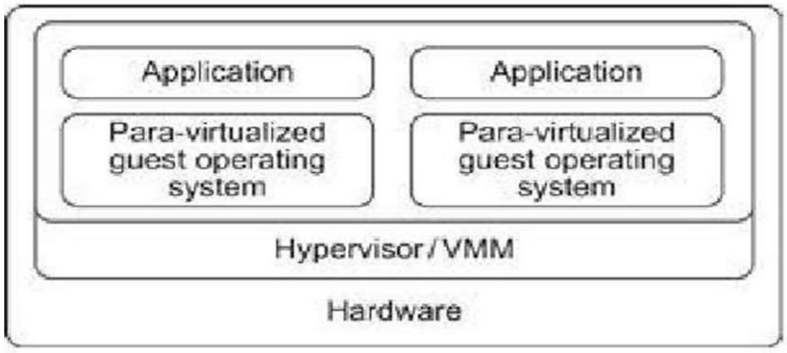
Host-Based Virtualization

- An alternative VM architecture is to install a virtualization layer on top of the host OS. This host OS is still responsible for managing the hardware.
- This host-based architecture has some distinct advantages. First, the user can install this VM architecture without modifying the host OS. The virtualizing software can rely on the host OS to provide device drivers and other low-level services. This will simplify the VM design and ease its deployment.
- Second, the host-based approach appeals to many host machine configurations. Compared to the hypervisor/VMM architecture, the performance of the host-based architecture may also be low.

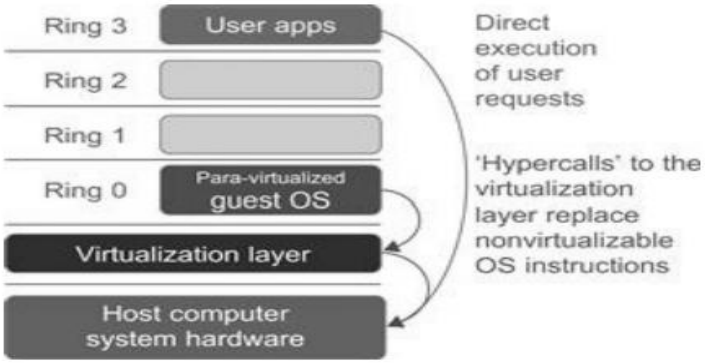
Para –virtualization

- Para -virtualization needs to modify the guest operating systems. A para-virtualized VM
- provides special API s requiring substantial OS modifications in user applications.

- Performance degradation is a critical issue of a virtualized system.



The use of para-virtualized guest OS assisted by an intelligent compiler to replace nonvirtualizable OS instructions by hypercalls



Full virtualization vs. Para virtualization

Full virtualization

- Does not need to modify guest OS, and critical instructions are emulated by software through the use of binary translation.
- VMware Workstation applies full virtualization, which uses binary translation to automatically modify x86 software on-the-fly to replace critical instructions.

Advantage: no need to modify OS.

Disadvantage: binary translation slows down the performance.

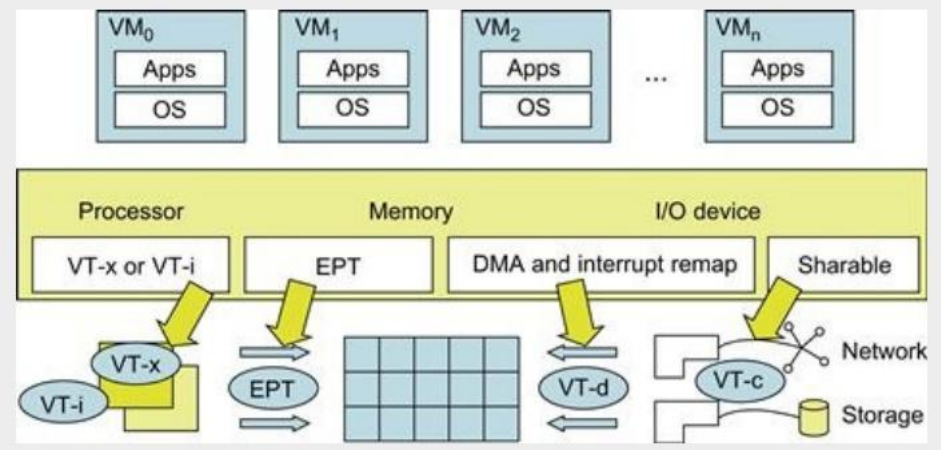
Para virtualization

- Reduces the overhead, but cost of maintaining a paravirtualized OS is high.
- The improvement depends on the workload.
- Para virtualization must modify guest OS, non-virtualizable instructions are replaced by hyper calls that communicate directly with the hypervisor or VMM.

7. Explain Virtualization of CPU, Memory, and I/O Devices?

Hardware Support for Virtualization

Modern operating systems and processors permit multiple processes to run simultaneously. If there is no protection mechanism in a processor, all instructions from different processes will access the hardware directly and cause a system crash. Therefore, all processors have at least two modes, user mode and supervisor mode, to ensure controlled access of critical hardware. Instructions running in supervisor mode are called privileged instructions. Other instructions are unprivileged instructions. The VMware Workstation is a VM software suite for x86 and x86-64 computers. This software suite allows users to set up multiple x86 and x86-64 virtual computers and to use one or more of these VMs simultaneously with the host operating system. The VMware Workstation assumes the host-based virtualization. Xen is a hypervisor for use in IA-32, x86-64, Itanium, and PowerPC 970 hosts.



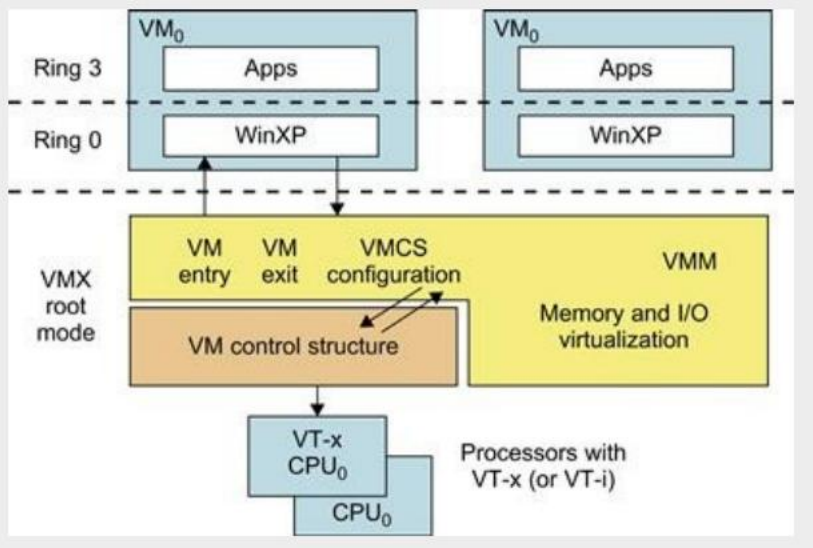
CPU Virtualization

A VM is a duplicate of an existing computer system in which a majority of the VM instructions are executed on the host processor in native mode. Thus, unprivileged instructions of VMs run directly on the host machine for higher efficiency. Other critical instructions should be handled carefully for correctness and stability. The critical instructions are divided into three categories: privileged instructions, control-sensitive instructions, and behavior-sensitive instructions. Privileged instructions execute in a privileged mode and will be trapped if executed outside this mode. Control-sensitive instructions attempt to change the configuration of resources used. Behavior-sensitive instructions have different behaviors depending on the configuration of resources, including the load and store operations over the virtual memory.

The VMM acts as a unified mediator for hardware access from different VMs to guarantee the correctness and stability of the whole system.

Hardware-Assisted CPU Virtualization

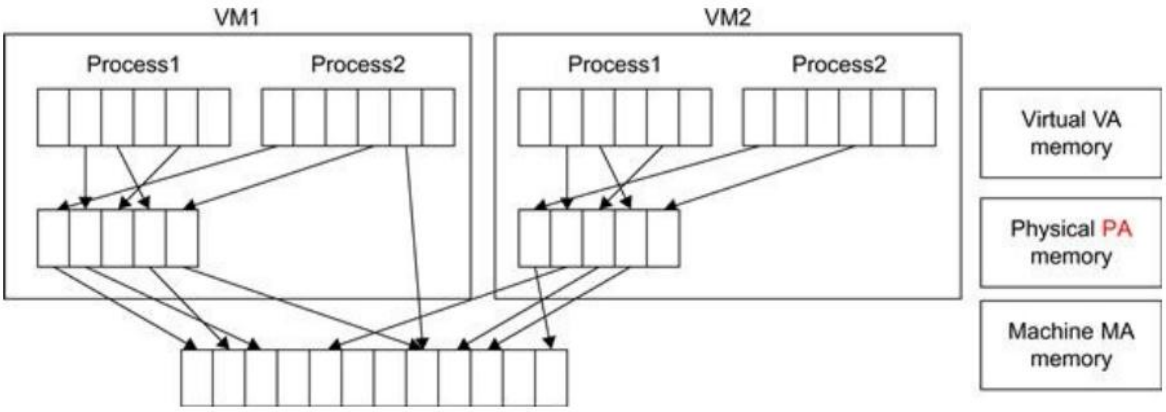
This technique attempts to simplify virtualization because full or paravirtualization is complicated. Intel and AMD add an additional mode called privilege mode level to x86 processors.



Memory Virtualization

Virtual memory virtualization is similar to the virtual memory support provided by modern operating systems. All modern x86 CPUs include a memory management unit (MMU) and a translation lookaside buffer (TLB) to optimize virtual memory performance. A two-stage mapping process should be maintained by the guest OS and the VMM, respectively: virtual memory to physical memory and physical memory to machine memory.

Two level memory mapping process



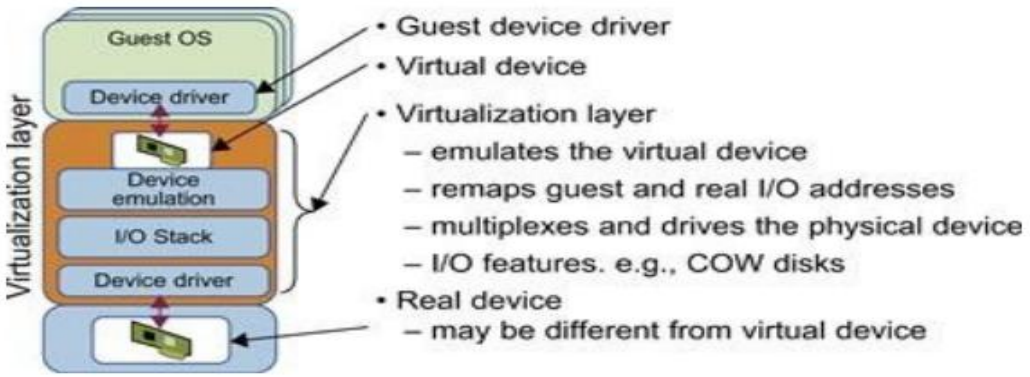
The guest OS continues to control the mapping of virtual addresses to the physical memory addresses of VMs. The MMU already handles virtual-to-physical translations as defined by the OS. Then the physical memory addresses are translated to machine addresses using another set of page tables defined by the hypervisor. Processors use TLB hardware to map the virtual memory directly to the machine memory to avoid the two levels of translation on every access. When the guest OS changes the virtual memory to a physical memory mapping, the VMM updates the shadow page tables to enable a direct lookup.

I/O Virtualization

I/O virtualization involves managing the routing of I/O requests between virtual devices and the shared physical hardware. At the time of this writing, there are **three ways to implement I/O virtualization: full device emulation, para-virtualization, and direct I/O.**

Full device emulation is the first approach for I/O virtualization. Generally, this approach emulates well-known, real-world devices. All the functions of a device or bus infrastructure, such as device enumeration, identification, interrupts, and DMA, are replicated in software. This software is located in the VMM and acts as a virtual device. The I/O access requests of the guest OS are trapped in the VMM which interacts with the I/O devices.

Full device emulation



The para-virtualization method of I/O virtualization is typically used in Xen. It is also known as the split driver model consisting of a frontend driver and a backend driver. The frontend driver is running in **Domain U** and the backend driver is running in **Domain 0**.They interact with each other via a block of shared memory. The frontend driver manages the I/O requests of the guest OSes and the backend driver is responsible for managing the real I/O devices and multiplexing the I/O data of different VMs.

Direct I/O virtualization lets the VM access devices directly. It can achieve close-to-native performance without high CPU cost.

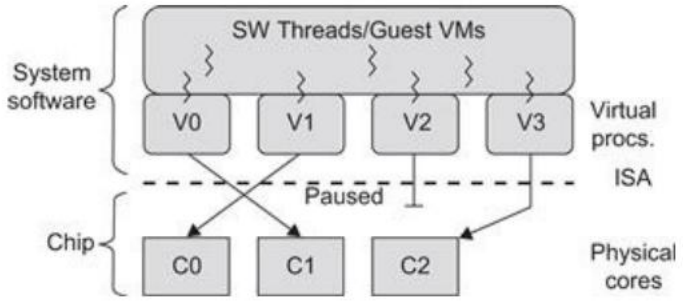
Virtualization in Multi-Core Processors

Virtualizing a multi-core processor is relatively more complicated than virtualizing a unicore processor. Though multicore processors are claimed to have higher performance by integrating multiple processor cores in a single chip, muti-core virtualization has raised some new challenges to computer architects, compiler constructors, system designers, and application programmers. There are mainly two difficulties: Application programs must be parallelized to use all cores fully, and software must explicitly assign tasks to the cores, which is a very complex problem.

Physical versus Virtual Processor Cores

This technique alleviates the burden and inefficiency of managing hardware resources by software. It is located under the ISA and remains unmodified by the operating system or VMM (hypervisor).

Multicore virtualization method that exposes four VCPUs to the software, when only three cores are actually present.



Virtual Hierarchy

The emerging many-core chip multiprocessors (CMPs) provides a new computing landscape. To optimize for space-shared workloads, they propose using virtual hierarchies to overlay a coherence and caching hierarchy onto a physical processor. Unlike a fixed physical hierarchy, a virtual hierarchy can adapt to fit how the work is space shared for improved performance and performance isolation.

8. Explain Live VM Migration Steps and Performance Effects?

In a cluster built with mixed nodes of host and guest systems, the normal method of operation is to run everything on the physical machine. When a VM fails, its role could be replaced by another VM on a different node, as long as they both run with the same guest OS. The advantage is enhanced failover flexibility.

The migration copies the VM state file from the storage area to the host machine.

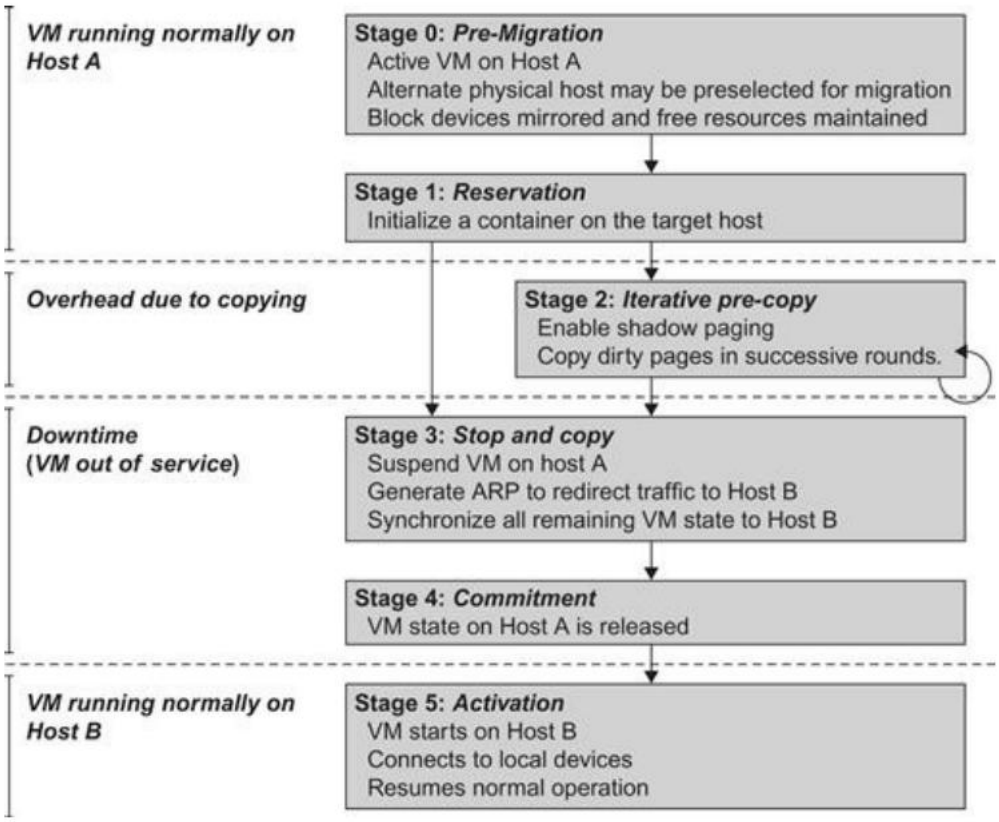
There are four ways to manage a virtual cluster.

1. First, you can use a guest-based manager, by which the cluster manager resides on a guest system. The host-based manager supervises the guest systems and can restart the guest system on another physical machine.
2. These two cluster management systems are either guest-only or host-only, but they do not mix.
3. A third way to manage a virtual cluster is to use an independent cluster manager on both the host and guest systems. This will make infrastructure management more complex.
4. Finally, you can use an integrated cluster on the guest and host systems.

VM can be in one of the following four states.

- An **inactive** state is defined by the virtualization platform, under which the VM is not enabled.
- An **active** state refers to a VM that has been instantiated at the virtualization platform to perform a real task.
- A **paused** state corresponds to a VM that has been instantiated but disabled to process a task or paused in a **waiting** state.

Live migration of a VM consists of the following six steps:



Live migration process of a VM from one host to another. Courtesy of C. Clark, et al. [14]

Steps 0 and 1: **Start migration**. This step makes preparations for the migration, including determining the migrating VM and the destination host. Although users could manually make a VM migrate to an appointed host, in most circumstances, the migration is automatically started by strategies such as load balancing and server consolidation.

Steps 2: **Transfer memory**. Since the whole execution state of the VM is stored in memory, sending the VM's memory to the destination node ensures continuity of the service provided by

the VM. All of the memory data is transferred in the first round, and then the migration controller recopies the memory data which is changed in the last round. These steps keep iterating until the dirty portion of the memory is small enough to handle the final copy. Although precopying memory is performed iteratively, the execution of programs is not obviously interrupted.

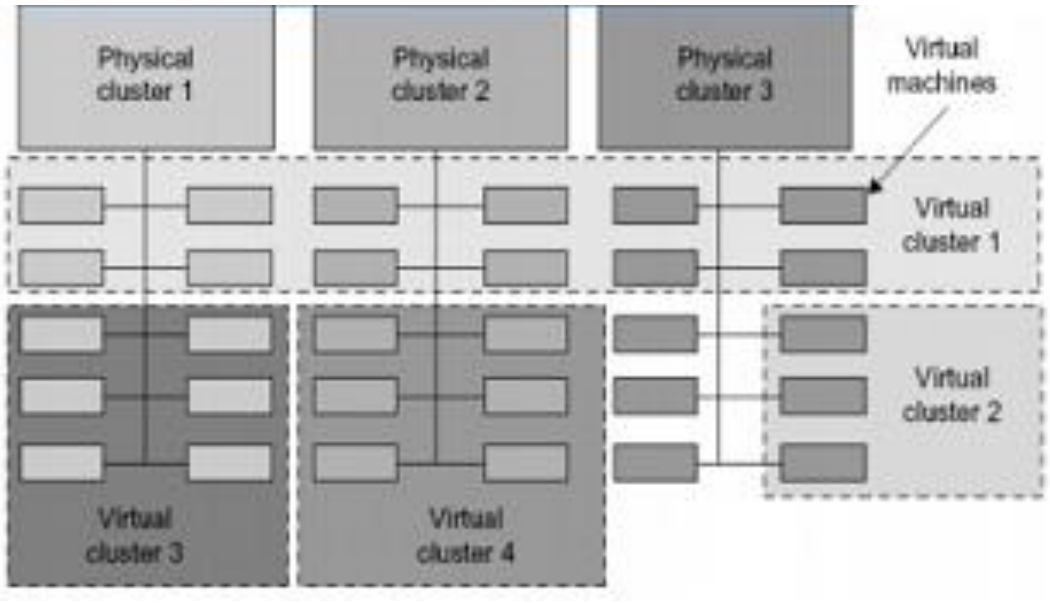
Step 3: **Suspend** the VM and copy the last portion of the data. The migrating VM's execution is suspended when the last round's memory data is transferred. Other nonmemory data such as CPU and network states should be sent as well. During this step, the VM is stopped and its applications will no longer run. This "service unavailable" time is called the "downtime" of migration, which should be as short as possible so that it can be negligible to users.

Steps 4 and 5: **Commit and activate** the new host. After all the needed data is copied, on the destination host, the VM reloads the states and recovers the execution of programs in it, and the service provided by this VM continues. Then the network connection is redirected to the new VM and the dependency to the source host is cleared. The whole migration process finishes by removing the original VM from the source host.

9. Explain Virtual clusters and resource management?

A physical cluster is a collection of servers (physical machines) interconnected by a physical network such as a LAN.

Virtual clusters are built with VMs installed at distributed servers from one or more physical clusters. The VMs in a virtual cluster are interconnected logically by a virtual network across several physical networks. Each virtual cluster is formed with physical machines or a VM hosted by multiple physical clusters. The virtual cluster boundaries are shown as distinct boundaries.



The provisioning of VMs to a virtual cluster is done dynamically to have the following Interesting properties:

• The virtual cluster nodes can be either physical or virtual machines. Multiple VMs running with different OSes can be deployed on the same physical node.

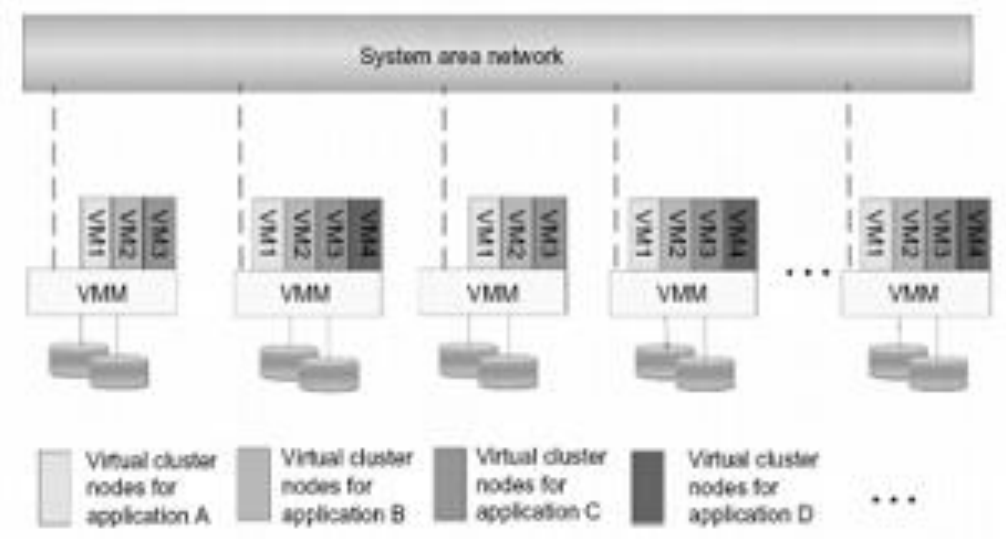
• A VM runs with a guest OS, which is often different from the host OS, that manages the resources in the physical machine, where the VM is implemented.

• The purpose of using VMs is to consolidate multiple functionalities on the same server. This will greatly enhance server utilization and application flexibility.

• VMs can be colonized (replicated) in multiple servers for the purpose of promoting distributed parallelism, fault tolerance, and disaster recovery.

• The size (number of nodes) of a virtual cluster can grow or shrink dynamically, similar to the way an overlay network varies in size in a peer-to-peer (P2P) network.

• The failure of any physical nodes may disable some VMs installed on the failing nodes. But the failure of VMs will not pull down the host system.



A Virtual Clusters based on Application Partitioning:

Parallax Providing Virtual Disks to Clients VMs from a Large Common Shared Physical Disk.

10. Discuss about Virtualization for Data-Center Automation

- Data-center automation means that huge volumes of hardware, software, and database resources in these data centers can be allocated dynamically to millions of Internet users simultaneously, with guaranteed QoS and cost-effectiveness.
- Google, Yahoo!, Amazon, Microso, HP, Apple, and I BM companies have invested billions of dollars in data-center construction and automation.

Server Consolidation in Data Centers

- In data centers, a large number of heterogeneous workloads can run on servers at various times. These heterogeneous workloads can be roughly divided into two categories:

1. Chatty workloads and
2. Noninteractive workloads.

- Chay workloads may burst at some point and return to a silent state at some other point. A web video service is an example of this, whereby a lot of people use it at night and few people use it during the day.
- Noninteractive workloads do not require people's efforts to make progress after they are submitted. High-performance computing is a typical example of this. At various stages, the requirements for resources of these workloads are dramatically different.

- It is common that most servers in data centers are underutilized. A large amount of hardware, space, power, and management cost of these servers is wasted.
- Server consolidation is an approach to improve the low utility ratio of hardware resources by reducing the number of physical servers.
- Among several server consolidation techniques such as centralized and physical Consolidation, *virtualization-based server consolidation is the most powerful*.
- Consolidation enhances hardware utilization. Many underutilized servers are consolidated into fewer servers to enhance resource utilization. Consolidation also facilitates backup services and disaster recovery.
- This approach enables more agile provisioning and deployment of resources. I n a virtual environment, the images of the guest OSes and their applications are readily cloned and reused.
- The total cost of ownership is reduced. I n this sense, server virtualization causes deferred purchases of new servers, a smaller data-center footprint, lower maintenance costs, and lower power, cooling, and cabling requirements.
- This approach improves availability and business continuity. The crash of a guest OS has no effect on the host OS or any other guest OS.

Virtual Storage Management

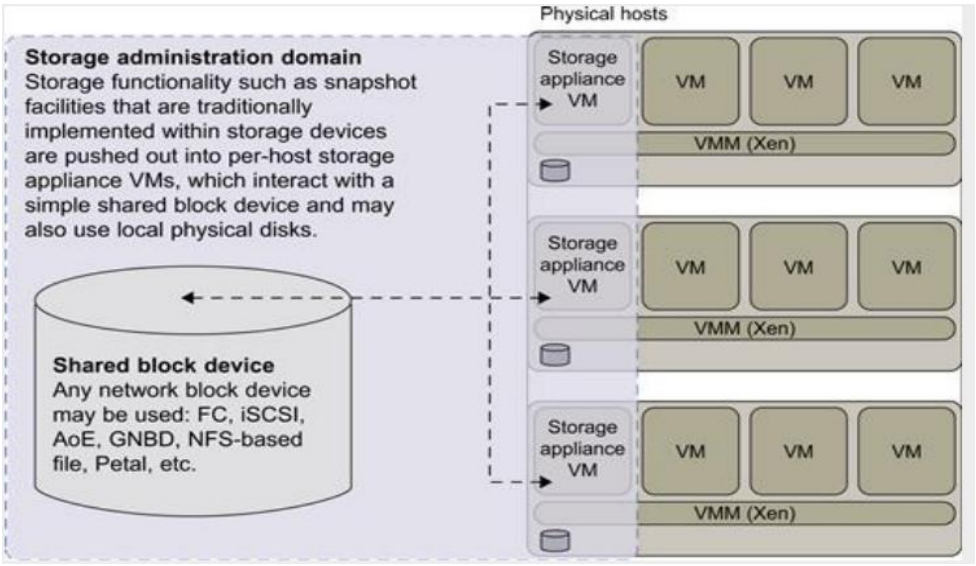
- I n system virtualization, virtual storage includes the storage managed by *VMMs* and *guest OSes*. Generally, the data stored in this environment can be classified into two categories:

1. VM images and
2. Application data.

- The *VM images* are special to the virtual environment,
- The *application data* includes all other data which is the same as the data in traditional OS environments.
- The most important aspects of system virtualization are encapsulation and isolation.
- Traditional operating systems and applications running on them can be encapsulated in VMs. Only one operating system runs in a virtualization while many applications run in the operating system. System virtualization allows multiple VMs to run on a physical machine and the VMs are completely isolated.

- To achieve encapsulation and isolation both the system software and the hardware platform, such as CPUs and chipsets, are rapidly updated. However, storage is lagging. The storage systems become the main bottleneck of VM deployment.
- Parallax is a distributed storage system customized for virtualization environments. Content Addressable Storage (CAS) is a solution to reduce the total size of VM images and therefore supports a large set of VM based systems in data centers.

Parallax architecture



Cloud OS for Virtualized Data Centers

- Data centers must be virtualized to serve as cloud providers.
- The table summarizes four virtual l infra structure (VI) managers and OSes.
- These VI managers and OSes are specially tailored for virtualizing data centers which own a large number of servers in clusters.
- Nimbus, Eucalyptus, and Open Nebula are all open source software available to the general public.

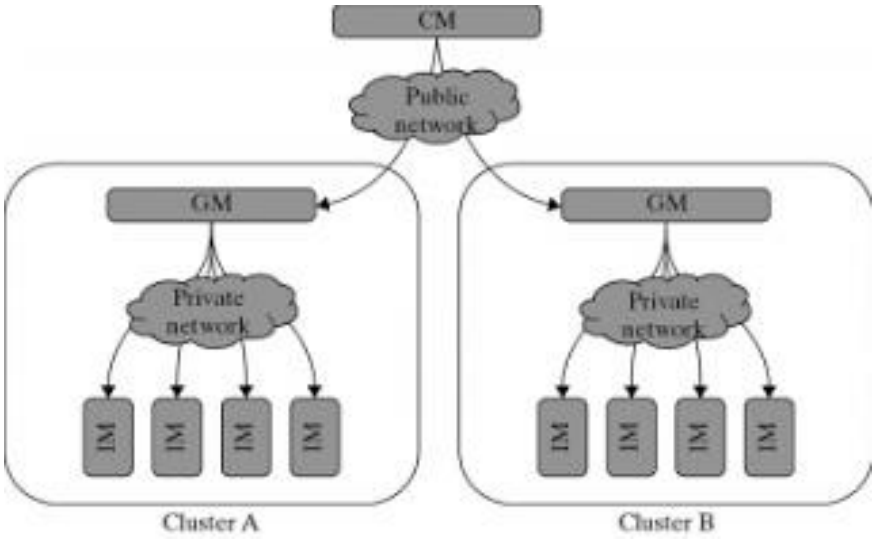
Table 3.6 VI Managers and Operating Systems for Virtualizing Data Centers [9]

Manager/ OS, Platforms, License	Resources Being Virtualized, Web Link	Client API, Language	Hypervisors Used	Public Cloud Interface	Special Features
Nimbus Linux, Apache v2	VM creation, virtual cluster, www .nimbusproject.org/	EC2 WS, WSRF, CLI	Xen, KVM	EC2	Virtual networks
Eucalyptus Linux, BSD	Virtual networking (Example 3.12 and [41]), www .eucalyptus.com/	EC2 WS, CLI	Xen, KVM	EC2	Virtual networks
OpenNebula Linux, Apache v2	Management of VM, host, virtual network, and scheduling tools, www.opennebula.org/	XML-RPC, CLI, Java	Xen, KVM	EC2, Elastic Host	Virtual networks, dynamic provisioning
vSphere 4 Linux, Windows, proprietary	Virtualizing OS for data centers (Example 3.13), www .vmware.com/ products/vsphere/ [66]	CLI, GUI, Portal, WS	VMware ESX, ESXi	VMware vCloud partners	Data protection, vStorage, VMFS, DRM, HA

Cloud OS for Building Private Clouds (VI: Virtual Infrastructure, EC2: Elastic Compute Cloud).

Eucalyptus: An Open-Source OS for Setting Up and Managing Private Clouds (IaaS)

Three Resource Managers: CM (Cloud Manager), GM (Group Manager), and IM (Instance Manager)Works like AWS APIs



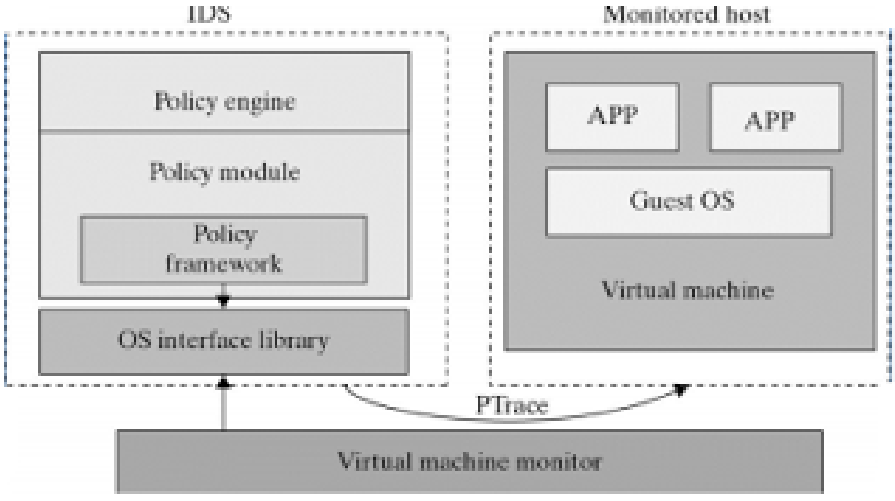
Trust Management in Virtualized Data Centers

- A VMM changes the computer architecture. I t provides a layer of software between the operating systems and system hardware to create one or more VMs on a single physical platform.
- VMM can provide secure isolation and a VM accesses hardware resources through the control of the VMM, so the VMM is the base of the security of a virtual system. Normally, one VM is taken as a management VM to have some privileges such as creating, suspending, resuming, or deleting a VM.
- Once a hacker successfully enters the VMM or management VM, the whole system is in danger.

VM-Based Intrusion Detection

- Intrusion detection is used to recognize the unauthorized access.
- An intrusion detection system (I DS) is built on operating systems, and is based on the characteristics of intrusion actions.
- A typical I DS can be classified as a host-based IDS (HIDS) or a network based IDS (NIDS), depending on the data source.
- A HI DS can be implemented on the monitored system. When the monitored system is haacked by hackers, the HI DS also faces the risk of being haacked. A NI DS is based on the flow of network traffic which can't detect fake actions.
- Virtualization-based intrusion detection can isolate guest VMs on the same hardware platform. Even some VMs can be invaded successfully; they never influence other VMs

VM-based Intrusion Detection.



Techniques for establishing trusted zones for virtual cluster insulation and VM isolation

CS6703 / GCC UNIT IV - PROGRAMMING MODEL

Open source grid middleware packages – Globus Toolkit (GT4) Architecture, Configuration – Usage of Globus – Main components and Programming model - Introduction to Hadoop Framework - Mapreduce, Input splitting, map and reduce functions, specifying input and output parameters, configuring and running a job – Design of Hadoop file system, HDFS concepts, command line and java interface, dataflow of File read & File write.

1. Explain in detail about Open Source Grid Middleware Packages.

- The Open Grid Forum and Object Management are two well- formed organizations behind the standards
- Middleware is the software layer that connects software components. It lies between operating system and the applications.
- Grid middleware is specially designed a layer between hardware and software, enable the sharing of heterogeneous resources and managing virtual organizations created around the grid.

The popular grid middleware are

Package	Brief Description
BOINC	Berkeley Open Infrastructure for Network Computing.
UNICORE	Middleware developed by the German grid computing community.
Globus (GT4)	A middleware library
CGSP in ChinaGrid	a middleware library developed by 20 top universities in China
Condor-G	Originally developed at the Univ. of Wisconsin for general distributed computing, and later extended to Condor-G for grid job management
Sun Grid Engine (SGE)	Developed by Sun Microsystems for business grid applications

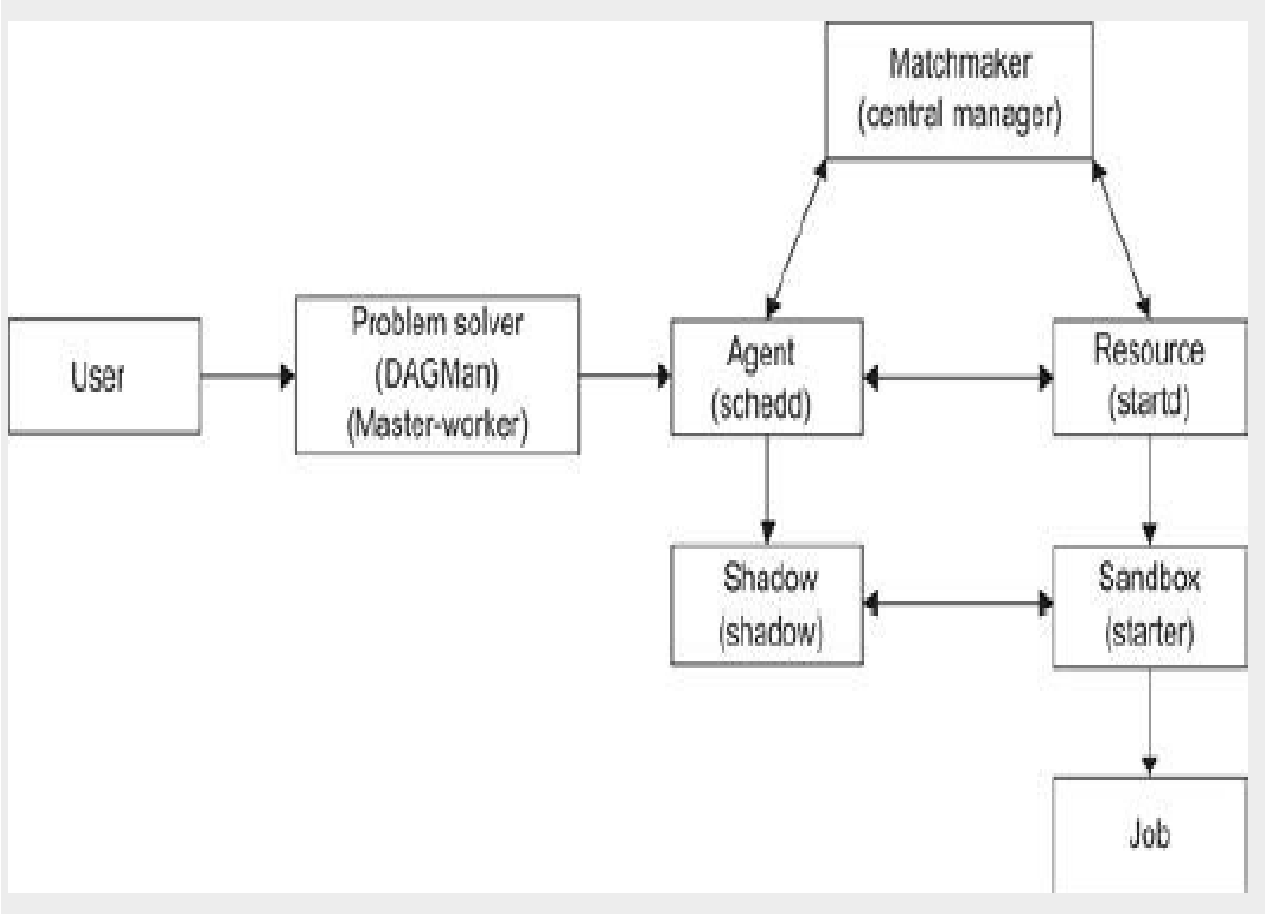
Grid Standards and APIs

- The Open Grid Forum and Object Management Group are two well-formed organizations behind those standards.
- OGSA - Open Grid Services Architecture
- GLUE - Grid Laboratory Uniform Environment
- SAGA - Simple API for Grid Applications
- GSI - Grid Security Infrastructure
- OGSI - Open Grid Service Infrastructure

- WSRE - Web Service Resource Framework
- The grid standards have guided the development of several middleware libraries and API tools for grid computing.
- They are applied in both research grids and production grids today.
- Research grids tested include the
    - EGEE,
    - France Grilles,
    - D-Grid (German),
    - CNGrid (China),
    - TeraGrid (USA), etc.
- Production grids built with the standards include the
    - EGEE,
    - INFN grid (Italian),
    - NorduGrid,
    - Sun Grid,
    - Techila, and
    - Xgrid
- Software Support and Middleware
- The middleware products enable the sharing of heterogeneous resources and managing virtual organizations created around the grid.
- Middleware glues the allocated resources with specific user applications.
- Popular grid middleware tools include the Globus Toolkits (USA), gLight, UNICORE (German), BOINC (Berkeley), CGSP (China), CondorG, and Sun Grid Engine.

Example:

- Features in Condor Kernel and Condor-G for Grid Computing
- Condor is a software tool for high-throughput distributed batch computing.
- It was designed to explore the idle cycles of a network of distributed computers.
- The major components of Condor are the user agent, resources, and matchmaker.
- The ClassAds (classified advertisements) language was exploited in Condor to express user requests against available resources in a distributed system.
- Agents and resources advertise their status and requirements in ClassAds to a central matchmaker.
- The matchmaker scans the ClassAds and creates pairs of (resources, agents) that match each other's requirements.
- A matched agent negotiates with the available resource to execute the job.
- Two problem solvers are provided in Condor: the masterworker and the DAG manager

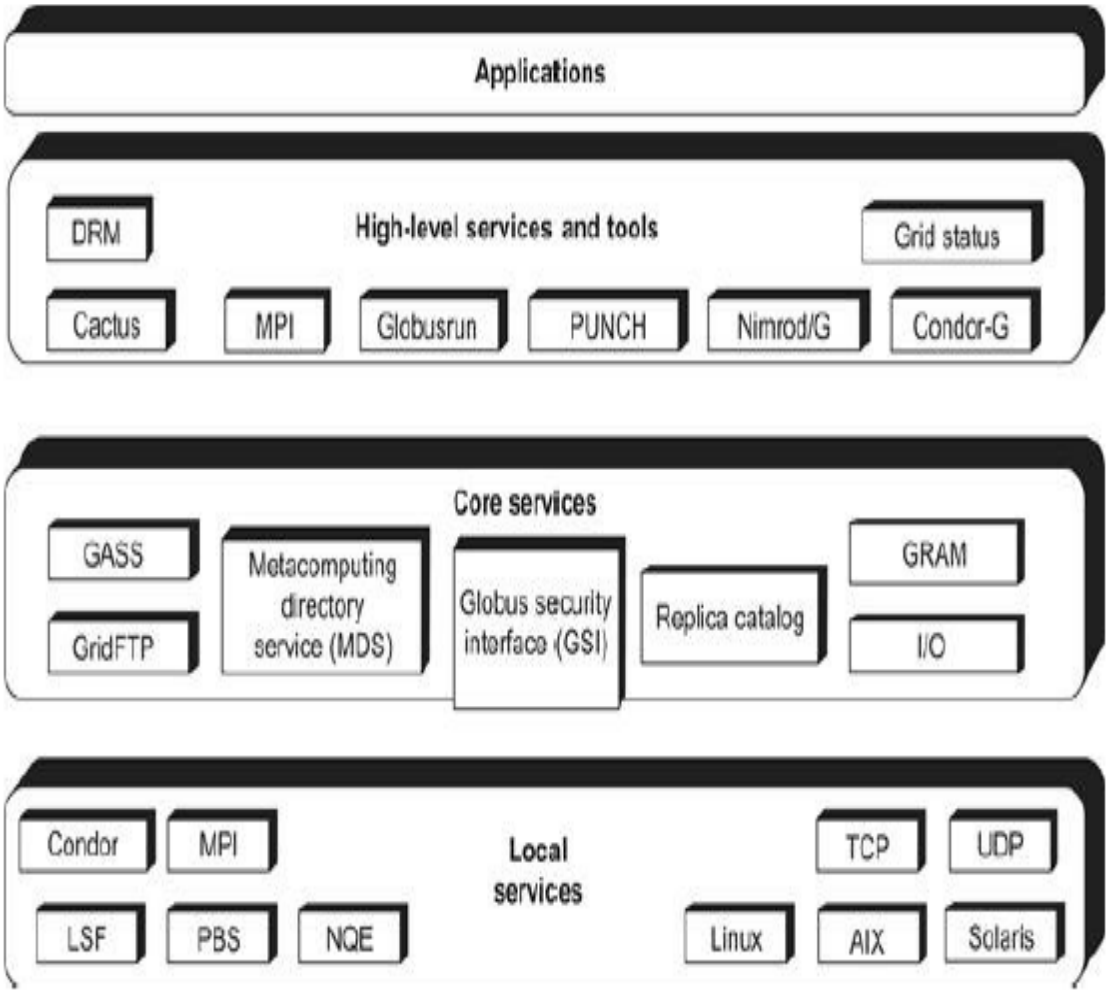


Major functional components in a Condor system

2. Explain on the Globus Toolkit Architecture (GT4).

- The Globus Toolkit, started in 1995 with funding from DARPA, is an open middleware
- library for the grid computing communities.
- The toolkit addresses common problems and issues related to
  - grid resource discovery,
  - management,
  - communication,
  - security,
  - fault detection, and
  - portability.
- The software itself provides a variety of components and capabilities.
- The library includes a rich set of service implementations.
- The implemented software
  - Supports grid infrastructure management,
  - provides tools for building new web services in Java, C, and Python,
  - builds a powerful standard-based security infrastructure and client APIs
  - offers comprehensive command-line programs for accessing various grid services.
- The toolkit is for sharing of resources and services, in scientific and engineering

- applications.
- The shared resources can be computers, storage, data, services, networks, science instruments.



Globus Tookit GT4 supports distributed and cluster computing services

The GT4 Library

- GT4 offers the middle-level core services in grid applications.
- The high-level services and tools, such as MPI, Condor-G, and Nirod/G, are developed by third parties.
- It is been for generalpurpose distributed computing applications.
- The local services, such as LSF, TCP, Linux, and Condor, are at the bottom level.
- The functional modules help users to discover available resources, move data betweensites, manage user credentials.
- GT4 is based on industry-standard web service technologies.

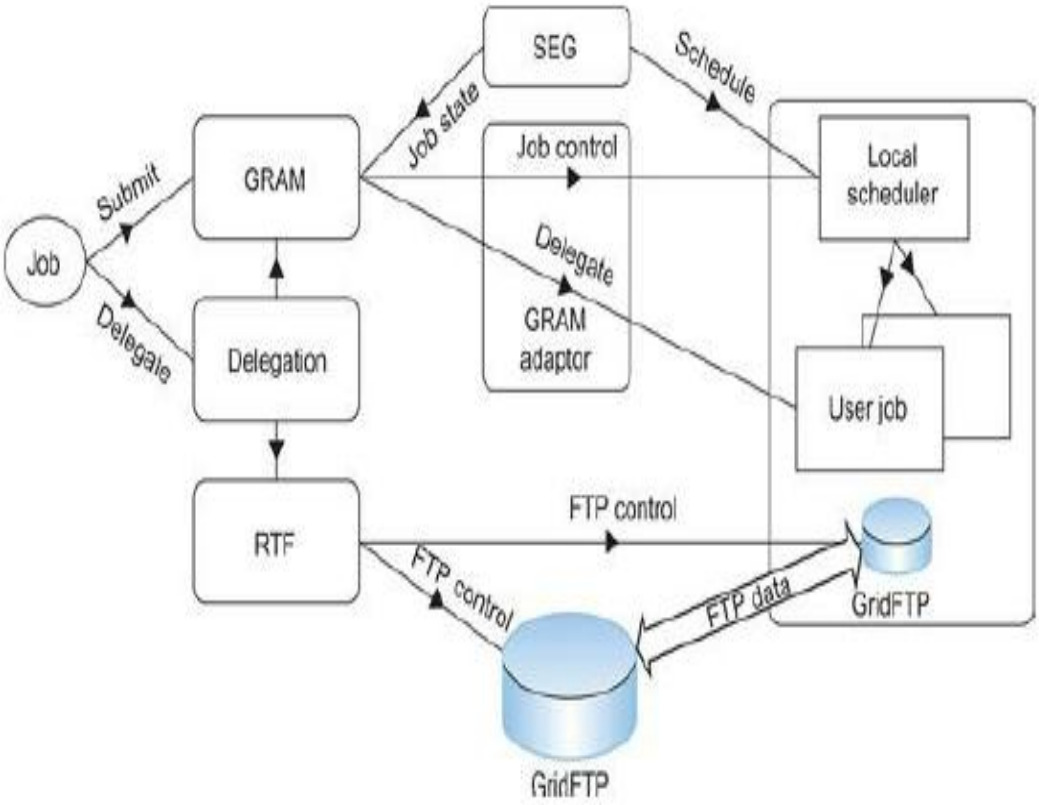
Functional Modules in Globus GT4 Library

Service Functionality	Module Name	Functional Description
Global Resource Allocation Manager	GRAM	Grid Resource Access and Management (HTTP-based)
Communication	Nexus	Unicast and multicast communication
Grid Security Infrastructure	GSI	Authentication and related security services
Monitory and Discovery Service	MDS	Distributed access to structure and state information
Health and Status	HBM	Heartbeat monitoring of system components
Global Access of Secondary Storage	GASS	Grid access of data in remote secondary storage
Grid File Transfer	GridFTP	Inter-node fast file transfer

- Nexus is used for collective communications and HBM for heartbeat monitoring of resource
- nodes.
- GridFTP is for speeding up internode file transfers.
- The module GASS is used for global access of secondary storage.

Globus Job Workflow

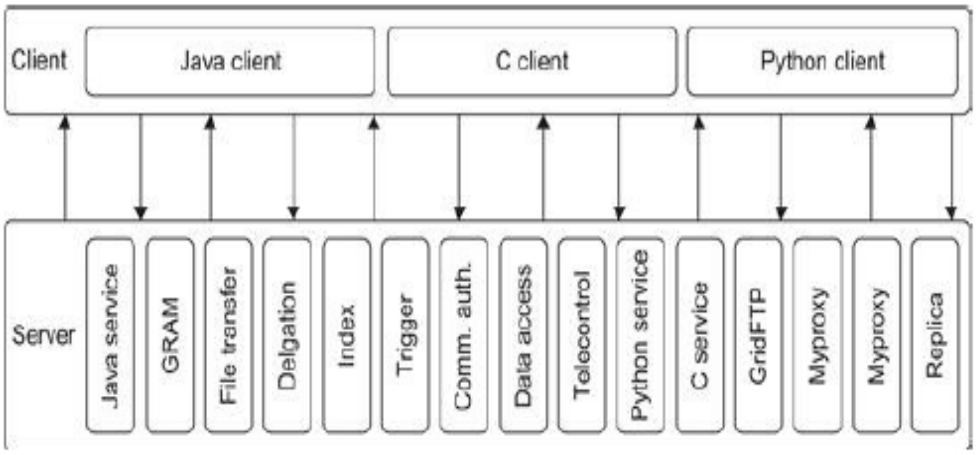
- Typical job execution sequence proceeds as follows:
- The user delegates his credentials to a *delegation service*.
- The user submits a job request to GRAM with the delegation identifier as a parameter. GRAM parses the request, retrieves the user proxy certificate from the delegation service, and then acts on behalf of the user.
- GRAM sends a transfer request to the RFT (Reliable File Transfer),
- It applies GridFTP to bring in the necessary files.
- GRAM invokes a *local scheduler* via a GRAM adapter and the SEG (Scheduler Event
- Generator) initiates a set of user jobs.
- The local scheduler reports the job state to the SEG.
- Once the job is complete, GRAM uses RFT and GridFTP to stage out the resultant files.
- The grid monitors the progress of these operations and sends the user a notification
- when they succeed, fail, or are delayed.



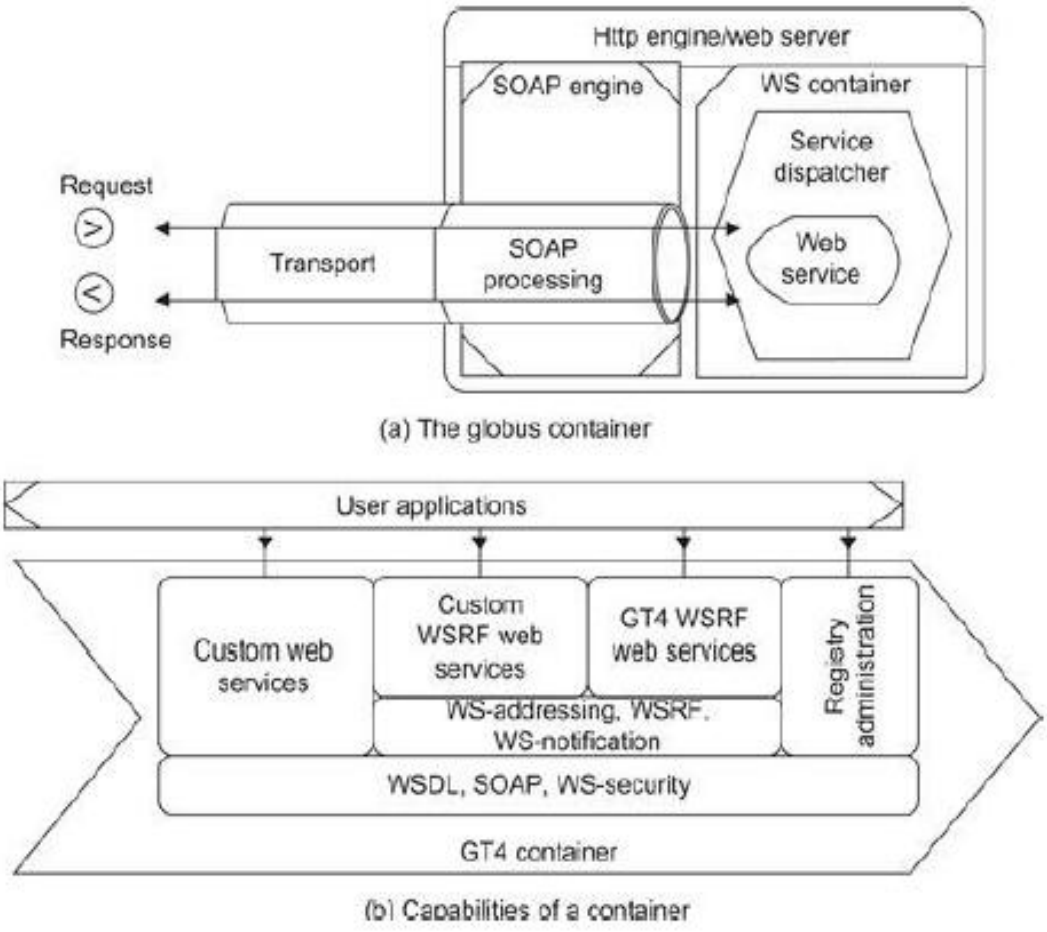
Globus job workflow among interactive functional modules

Client-Globus Interactions

- GT4 service programs are designed to support user applications.
- There are strong interactions between provider programs and user code.
- GT4 makes heavy use of industry-standard web service protocols and mechanisms in service description, discovery, access, authentication, authorization, and the like.
- GT4 makes extensive use of Java, C, and Python to write user code.
- Web service mechanisms define specific interfaces for grid computing.
- Web services provide flexible, extensible, and widely adopted XML-based interfaces.



- Client and GT4 server interactions; vertical boxes correspond to service programs and horizontal boxes represent the user codes.
- GT4 components do not, in general, address end-user needs directly. Instead, GT4
- provides a set of infrastructure services for accessing, monitoring, managing, and
- controlling access to infrastructure elements. The server code in the vertical boxes in corresponds to 15 grid services that are in heavy use in the GT4 library.
- These demand computational, communication, data, and storage resources. We must enable a range of end-user tools that provide the higher-level capabilities needed in specific user applications.
- Wherever possible, GT4 implements standards to facilitate construction of operable and reusable user code.
- Developers can use these services and libraries to build simple and complex systems quickly.
- A high-security subsystem addresses message protection, authentication, delegation, and authorization. The horizontal boxes in the client domain denote custom applications and/or third-party tools that access GT4 services. The toolkit programs provide a set of useful infrastructure services.



Globus container serving as a run time environment for implementing web services in grid

- Three containers are used to host user-developed services written in Java, Python, and C, respectively.
- These containers provide implementations of security, management, discovery, state management, and other mechanisms frequently required when building services.
- They extend open source service hosting environments with support for a range of useful web service specifications, including WSRF, WS-Notification, and WS-Security.
- A set of client libraries allow client programs in Java, C, and Python to invoke operations on both GT4 and user-developed services.
- In many cases, multiple interfaces provide different levels of control:

3. Explain in detail about Hadoop.

- Hadoop is an open source implementation of MapReduce coded and released in Java
- (rather than C) by Apache.
- The Hadoop implementation of MapReduce uses the *Hadoop Distributed File System (HDFS)*.

- The Hadoop core is divided into two fundamental layers: the MapReduce engine and HDFS.
- The MapReduce engine is the computation engine running on top of HDFS as its data storage manager.
- The following two sections cover the details of these two fundamental layers.

1. **HDFS:** HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.
2. **HDFS Architecture:** HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves).

- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).
- The mapping of blocks to DataNodes is determined by the NameNode.
- The NameNode (master) also manages the file system's metadata and namespace.

HDFS Features:

1. **HDFS Fault Tolerance:** One of the main aspects of HDFS is its fault tolerance characteristic. Since Hadoop is designed to be deployed on low-cost hardware by default, a hardware failure in this system is considered to be common rather than an exception.
2. **Block replication** To reliably store data in HDFS, file blocks are replicated in this system. In other words, HDFS stores a file as a set of blocks and each block is replicated and distributed across the whole cluster. The replication factor is set by the user and is three by default.
3. **Replica placement** The placement of replicas is another factor to fulfill the desired fault tolerance in HDFS.
   For the default replication factor of three, HDFS stores one replica in the same node the original data is stored, one replica on a different node but in the same rack, and one replica on a different node in a different rack to provide three copies of the data.
4. **Heartbeat and Blockreport messages** Heartbeats and Blockreports are periodic messages sent to the NameNode by each DataNode in a cluster.
5. **HDFS High-Throughput Access to Large Data Sets (Files):** Also, because applications run on HDFS typically have large data sets, individual files are broken into large blocks (e.g., 64 MB) to allow HDFS to decrease the amount of metadata storage required per file.
   This provides two advantages: The list of blocks per file will shrink as the size of individual blocks increases, and by keeping large amounts of data sequentially within a block, HDFS provides fast streaming reads of data.

HDFS Operation: The control flow of HDFS operations such as write and read can properly highlight roles of the NameNode and DataNodes in the managing operations.

1. **Reading a file** To read a file in HDFS, a user sends an "open" request to the NameNode to get the location of file blocks.

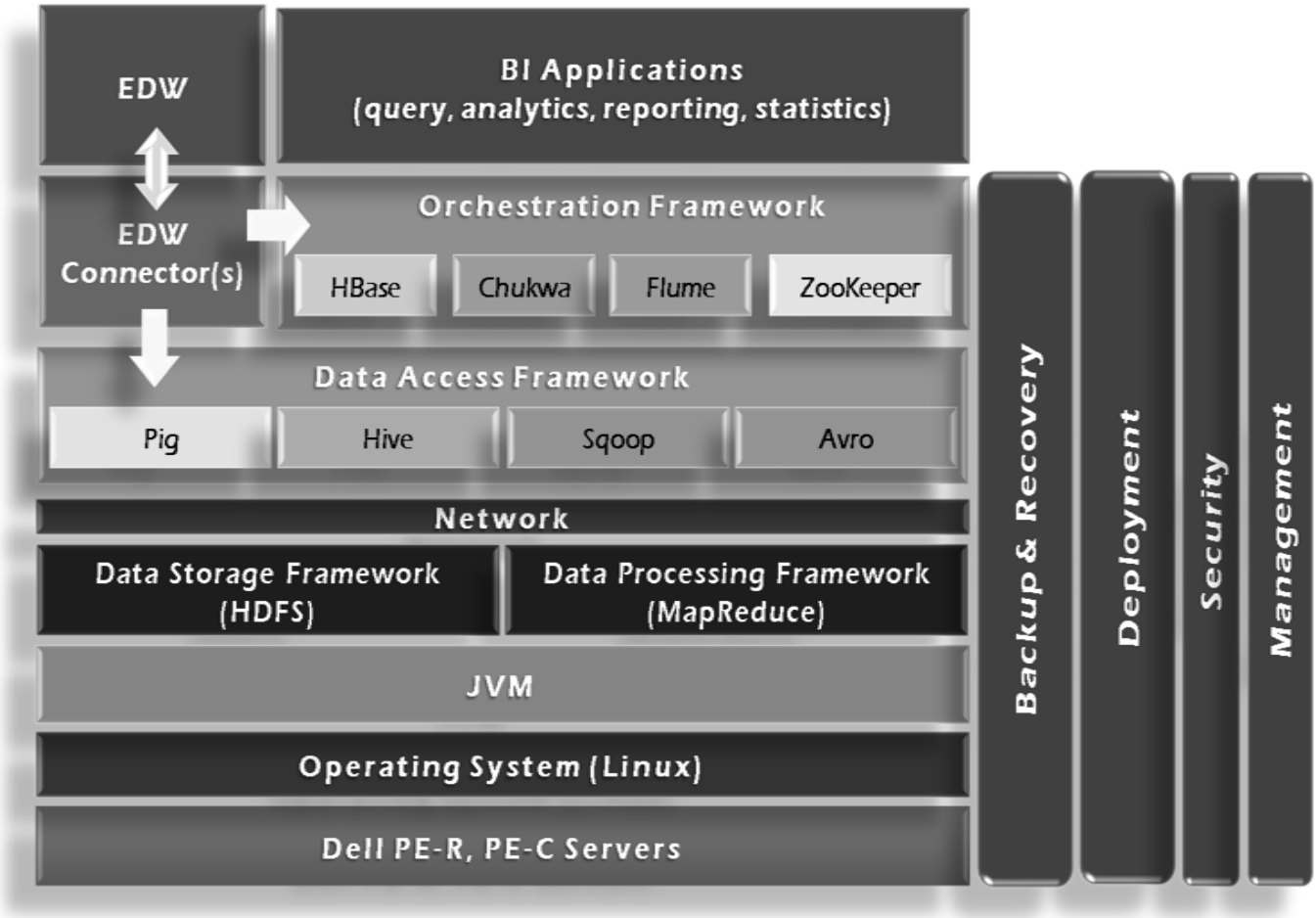
For each file block, the NameNode returns the address of a set of DataNodes containing replica information for the requested file. The number of addresses depends on the number of block replicas.

Upon receiving such information, the user calls the *read* function to connect to the closest DataNode containing the first block of the file.

2. **Writing to a file** To write a file in HDFS, a user sends a "create" request to the NameNode to create a new file in the file system namespace.

The first block of the file is written to an internal queue termed the data queue while a data streamer monitors its writing into a DataNode.
Since each file block needs to be replicated by a predefined factor, the data streamer first sends a request to the NameNode to get a list of suitable DataNodes to store replicas of the first block.
The steamer then stores the block in the first allocated DataNode.

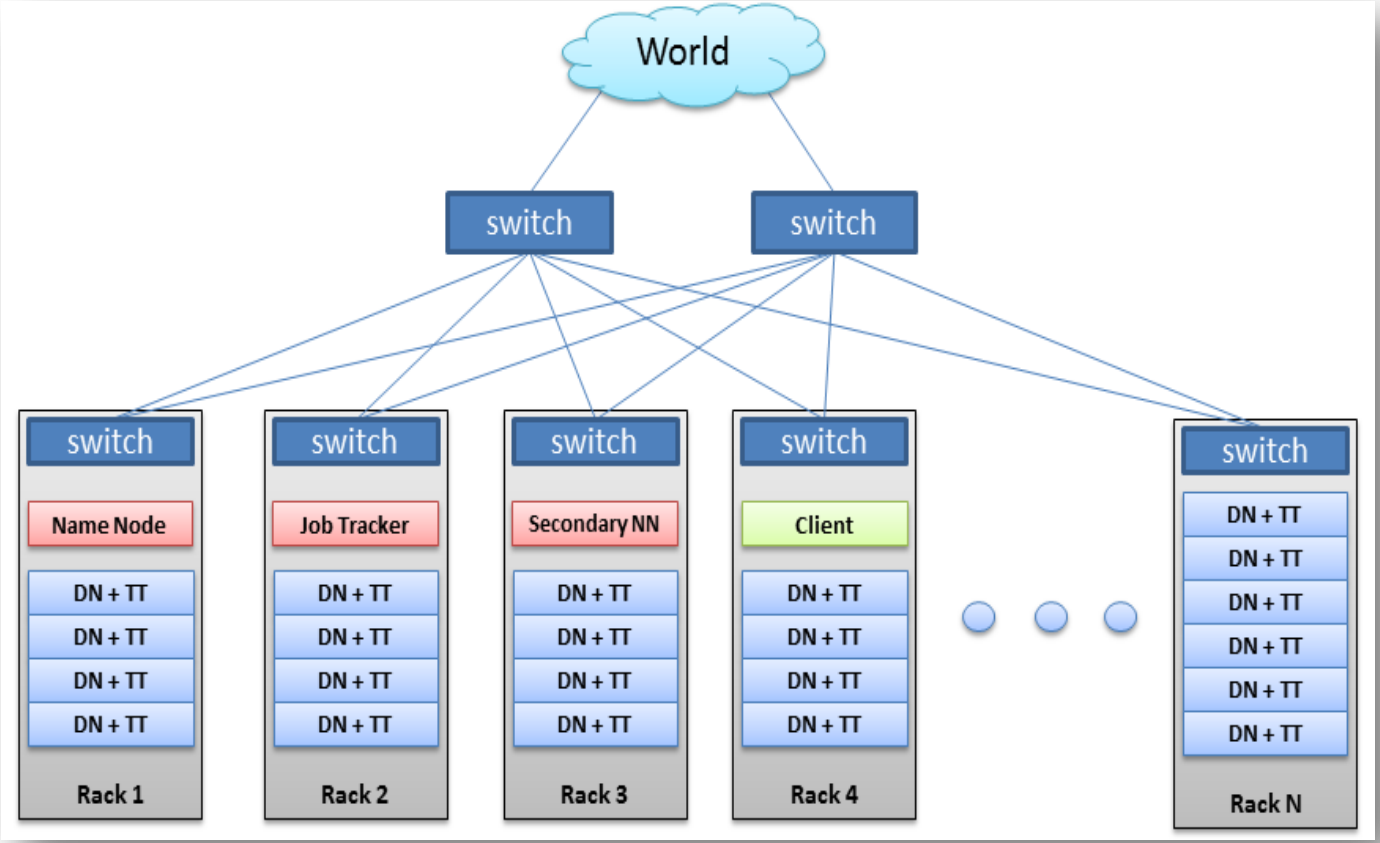
Hadoop Framework Tools



4. Hadoop's Architecture

- Distributed, with some centralization
- Main nodes of cluster are where most of the computational power and storage of the system lies

- Main nodes run TaskTracker to accept and reply to MapReduce tasks, and also DataNode to store needed blocks closely as possible
- Central control node runs NameNode to keep track of HDFS directories & files, and JobTracker to dispatch compute tasks to TaskTracker
- Written in Java, also supports Python and Ruby



- Hadoop Distributed Filesystem
- Tailored to needs of MapReduce
- Targeted towards many reads of filestreams
- Writes are more costly
- High degree of data replication (3x by default)
- No need for RAID on normal nodes
- Large blocksize (64MB)
- Location awareness of DataNodes in network

NameNode:

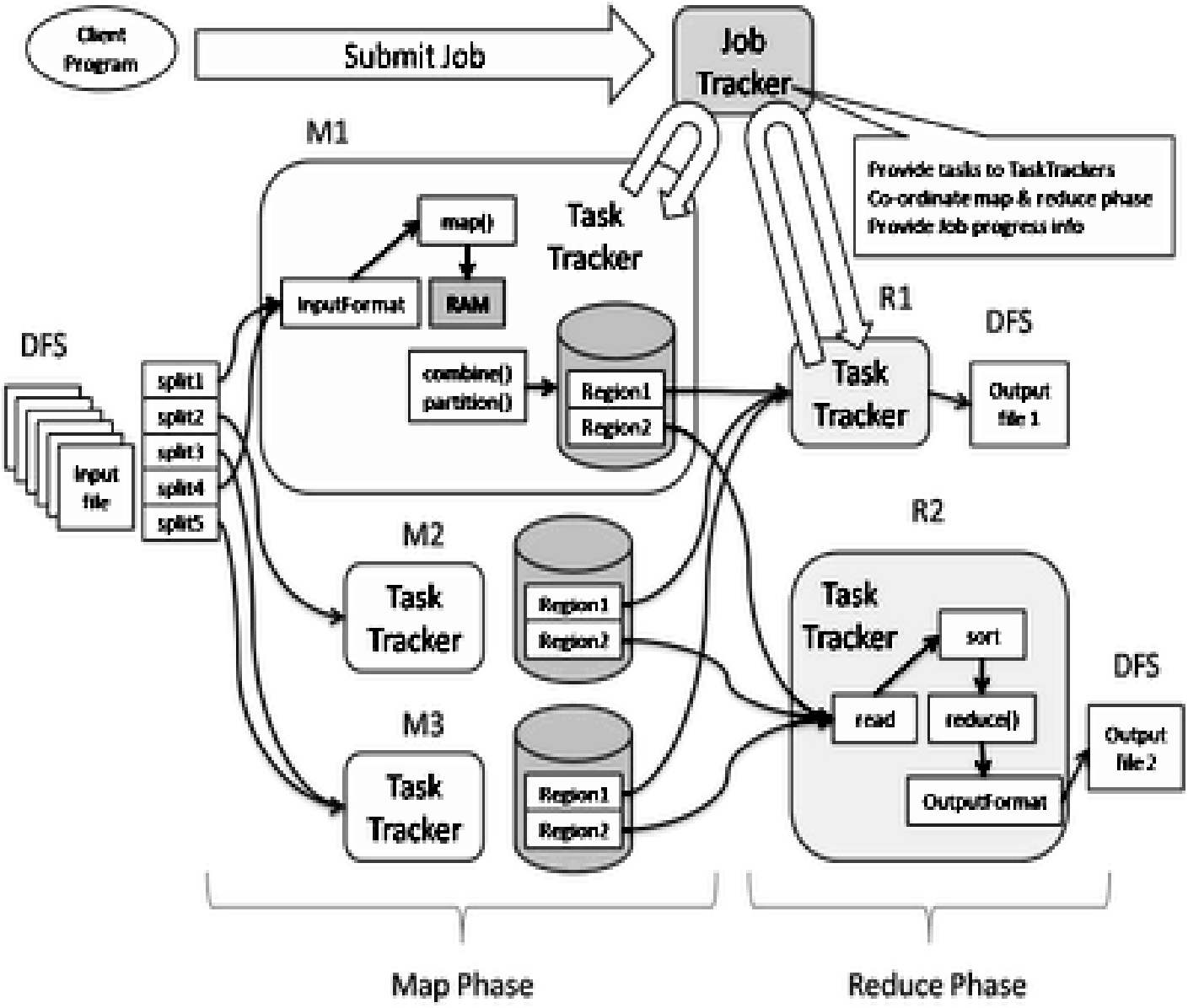
- Stores metadata for the files, like the directory structure of a typical FS.
- The server holding the NameNode instance is quite crucial, as there is only one.
- Transaction log for file deletes/adds, etc. Does not use transactions for whole blocks or file-streams, only metadata.

- Handles creation of more replica blocks when necessary after a DataNode failure

DataNode:

- Stores the actual data in HDFS
- Can run on any underlying filesystem (ext3/4, NTFS, etc)
- Notifies NameNode of what blocks it has

NameNode replicates blocks 2x in local rack, 1x elsewhere



MapReduce Engine:

- JobTracker & TaskTracker
- JobTracker splits up data into smaller tasks("Map") and sends it to the TaskTracker process in each node

- TaskTracker reports back to the JobTracker node and reports on job progress, sends data ("Reduce") or requests new jobs
- None of these components are necessarily limited to using HDFS
- Many other distributed file-systems with quite different architectures work
- Many other software packages besides Hadoop's MapReduce platform make use of HDFS

Hadoop is in use at most organizations that handle big data:

- Yahoo!
- Facebook
- Amazon
- Netflix
- Etc…

Three main applications of Hadoop

- Advertisement (Mining user behavior to generate recommendations)
- Searches (group related documents)
- Security (search for uncommon patterns)

Hadoop Highlights

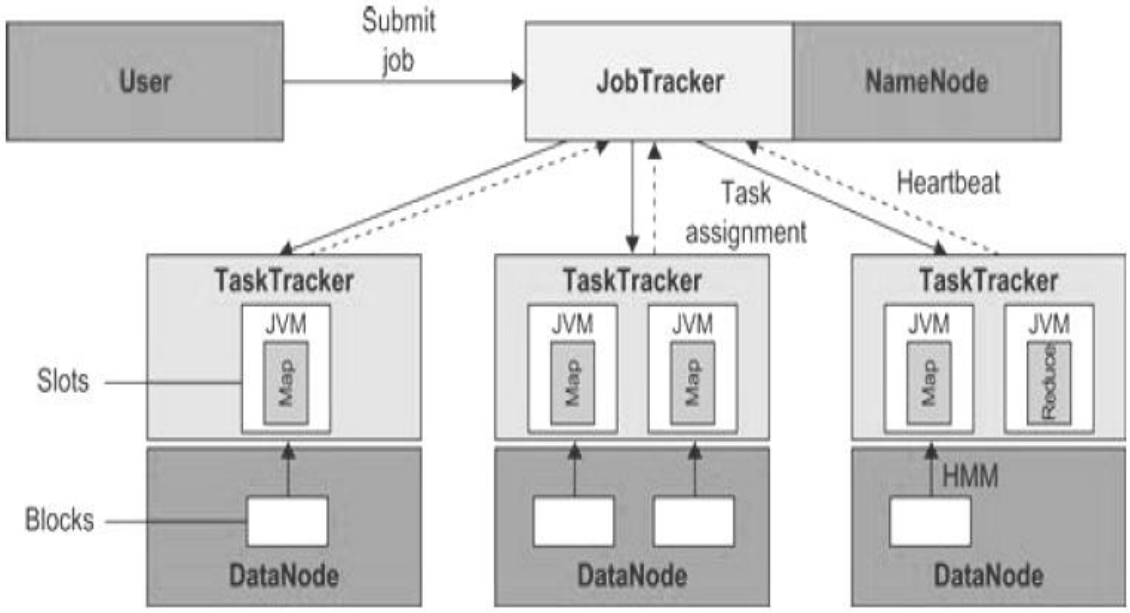
1. Distributed file system
2. Fault tolerance
3. Open data format
4. Flexible schema
5. Queryable Database

Why use Hadoop?

1. Need to process Multi petabyte datasets
2. Data may not have strict schema
3. Expensive to build reliability in each application
4. Nodes fails everyday
5. Need common infrastructure
6. Very Large Distributed file sytem
7. Assumes commodity hardware
8. Optimized for Batch processing
9. Runs on heterogeneous OS.

5. **Explain in detail about the architecture of MapReduce in Hadoop.**

- The topmost layer of Hadoop is the MapReduce engine that manages the data flow and control flow of MapReduce jobs over distributed computing systems.
- Similar to HDFS, theMapReduce engine also has a master/slave architecture consisting of a single JobTrackeras the master and a number of TaskTrackers as the slaves (workers).
- The JobTrackermanages the MapReduce job over a cluster and is responsible for monitoring jobs andassigning tasks to TaskTrackers.
- The TaskTracker manages the execution of the mapand/or reduce tasks on a single computation node in the cluster.
- Each TaskTracker node has a number of simultaneous execution slots, each executing either a map or a reduce task.
- Slots are defined as the number of simultaneous threads supported by CPUs of the TaskTracker node.



Data flow in running a MapReduce

- Three components contribute in running a job in this system: a user node, a JobTracker, and several TaskTrackers.
- The data flow starts by calling the *runJob(conf)* function inside a user program running on the user node, in which *conf*is an object containing some tuning parameters for the MapReduce framework and HDFS.
- The *runJob(conf)* function and *conf*are comparable to the *MapReduce(Spec, &Results)* function and *Spec* in the first implementation of MapReduce by Google

1. **Job Submission**
   - Each job is submitted from a user node to the JobTracker node that might be situated in a different node within the cluster through the following procedure:
   - A user node asks for a new job ID from the JobTracker and computes input file splits.
   - The user node copies some resources, such as the job's JAR file, configuration file, and computed input splits, to the JobTracker's file system.
   - The user node submits the job to the JobTracker by calling the *submitJob()* function.

2. **Task assignment**
   - The JobTracker creates one map task for each computed input split by the user node and assigns the map tasks to the execution slots of the TaskTrackers.
   - The JobTracker considers the localization of the data when assigning the map tasks to the TaskTrackers.
   - The JobTracker also creates reduce tasks and assigns them to the TaskTrackers.
   - The number of reduce tasks is predetermined by the user, and there is no locality consideration in assigning them.

3. **Task execution** The control flow to execute a task (either map or reduce) starts inside the TaskTracker by copying the job JAR file to its file system. Instructions inside the job JAR file are executed after launching a Java Virtual Machine (JVM) to run its map or reduce task.

4. **Task running check** A task running check is performed by receiving periodic heartbeat messages to the JobTracker from the TaskTrackers. Each heartbeat notifies the JobTracker that the sending TaskTracker is alive, and whether the sending TaskTracker is ready to run a new task.

MapReduce:

MapReduce, is a software framework which supportsparallel and distributed computing on large data sets.

This softwareframework abstracts the data flow of running a parallel program on a distributedcomputing system by providing users with two interfaces in the form of two functions: **Map and Reduce.**

Users can override these two functions to interact with and manipulate the data flow of running their programs.

Figure illustrates the logical data flow from the Map to the Reduce function in MapReduce frameworks.

The Map Reduce software framework provides an abstraction layer with the data flow and flow of control to users.

It hides the implementation of all data flow steps such as data partitioning, mapping, synchronization, communication, and scheduling.

The abstraction layer provides two well defined interfaces in the form of two functions: Map and Reduce.

Therefore, the user overrides the Map and Reduce functions first and then invokes the provided Map Reduce (Spec, & Results) function from the library to start the flow of data.

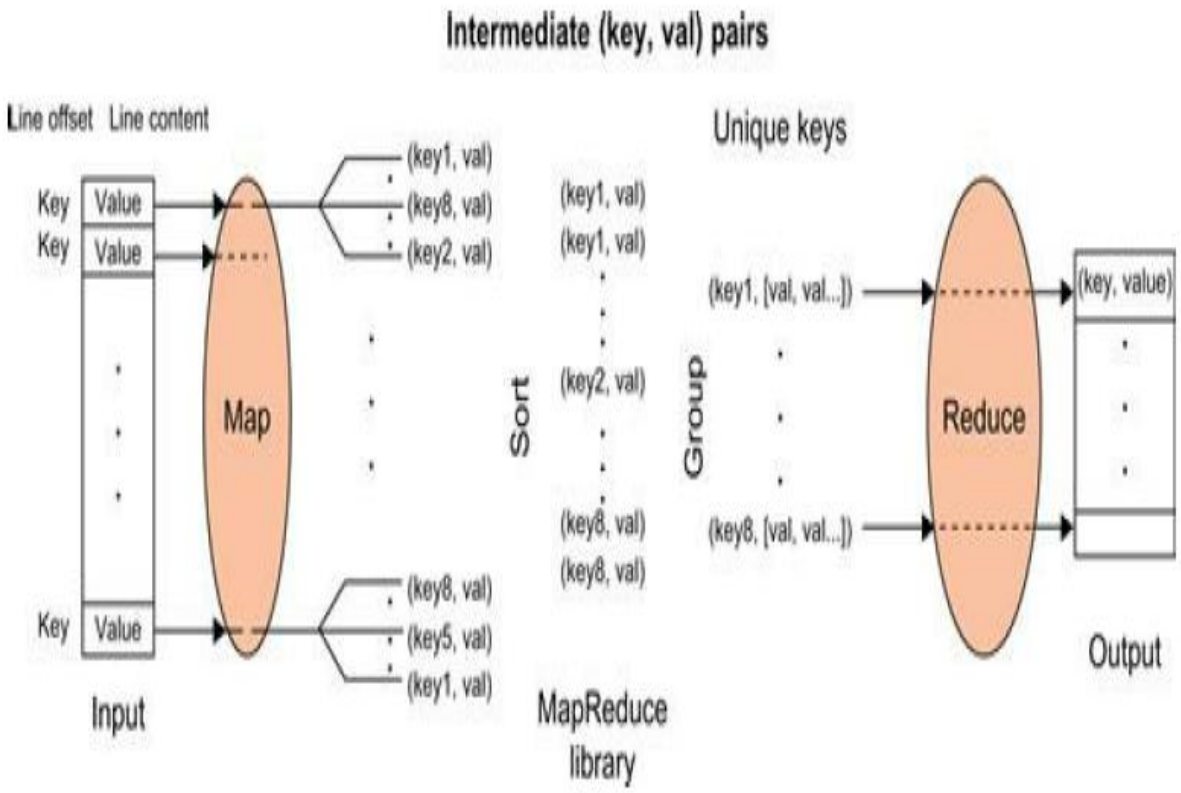
The Map Reduce function, Map Reduce (Spec, & Results), takes an important parameter which is a specification object, the Spec.
This object is first initialized inside the users program, and then the user writes code to fill it with the names of input and output files, as well as other optional tuning parameters. This object is also filled with the name of the Map and Reduce functions to identify these user-defined functions to the MapReduce library.
The overall structure of a user's program containing the Map, Reduce, and the Main functions is given below.
The Map and Reduce are two major subroutines.
They will be called to implement the desired function performed in the main program.

Map Function (… .)
{
… …
}
Reduce Function (… .)
{
… …
}
Main Function (… .)
{
Initialize **Spec** object
… …
MapReduce(Spec, & Results)
}

Map Reduce data flow:

- The input data to both the *Map* and the *Reduce* functions has a particular structure. This
- also pertains for the output data.
- The input data to the *Map* function is in the form of a (key, value) pair.
- For example, the key is the line offset within the input file and the value is the content of the line.
- The output data from the *Map* function is structured as (key, value) pairs called intermediate (key, value) pairs.

Map Reduce logical data flow in 5 processing stages over successive (key, value) pairs

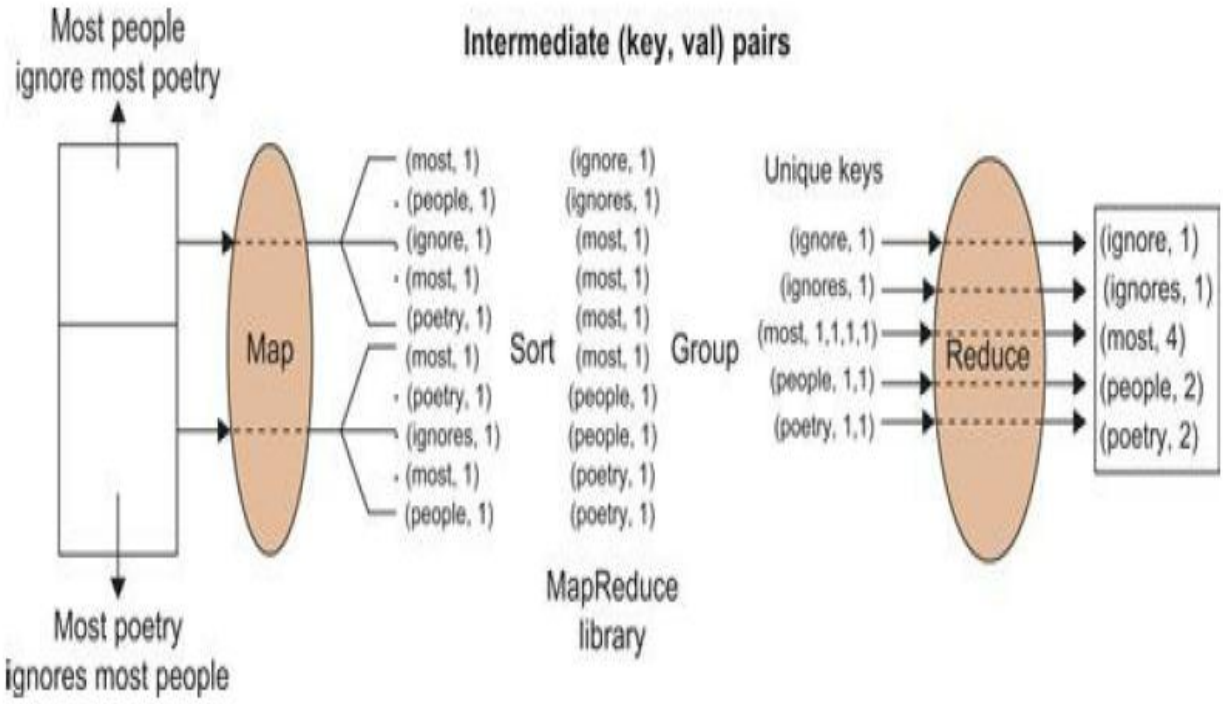


- The *Reduce* function receives the intermediate (key, value) pairs in the form of a
- group of intermediate values associated with one intermediate key, *(key, [set of values])*.
- In fact, the MapReduce framework forms these groups by first sorting the intermediate (key, value) pairs and then grouping values with the same key.
- The *Reduce* function processes each (key, [set of values]) group and produces a set of (key, value) pairs as output.

Eg: word count

- MapReduce problems, namely word count, to count the number of occurrences of each
- word in a collection of documents is presented here.
- For a simple input file containing only two lines asfollows:
    - "most people ignore most poetry" and (2) "most poetry ignores most people."
- In this case, the *Map* function simultaneously produces a number ofintermediate (key, value) pairs for each line of content so that each word is theintermediate key with *1* as its intermediate value; for example, *(ignore, 1)*.
- Then theMapReduce library collects all the generated intermediate (key, value) pairs and sortsthem to group the *1*'s for identical words; for example, *(people, [1,1])*.
- Groups are thensent to the *Reduce* function in parallel so that it can sum up the *1* values for each wordand generate the actual number of occurrence for each word in the file; for example,*(people, 2)*.

The data flow of the word count problem



To solve map reduce:
Problem 1: Counting the number of occurrences of each word in a collection of documents
Solution: unique "key": each word, intermediate "value": number of occurrences
Problem 2: Counting the number of occurrences of words having the same size, or the same number of letters, in a collection of documents
Solution: unique "key": each word, intermediate "value": size of the word
Problem 3: Counting the number of occurrences of anagrams in a collection of documents. Anagrams are words with the same set of letters but in a different order
(e.g., the words "listen" and "silent").
Solution: unique "key": alphabetically sorted sequence of letters for each word (e.g., "eilnst"), intermediate "value": number of occurrences

5. Give a detailed view of Data and control flow and running a job in mapreduce.

1. **Data partitioning :** The MapReduce library splits the input data (files), already stored in GFS, into *M* pieces that also correspond to the number of map tasks.
2. **Computation partitioning:** This is implicitly handled (in the MapReduce framework) by obliging users to write their programs in the form of the *Map* and *Reduce* functions.
3. **Determining the master and workers** The MapReduce architecture is based on a master-worker model. Therefore, one of the copies of the user program becomes the

master and the rest become workers. The master picks idle workers, and assigns the map and reduce tasks to them.
A map/reduce *worker* is typically a computation engine such as a cluster node to run map/reduce *tasks* by executing *Map*/*Reduce functions*. Steps 4–7 describe the map workers.

4. **Reading the input data (data distribution)** Each map worker reads its corresponding portion of the input data, namely the input data split, and sends it to its *Map* function.
Although a map worker may run more than one *Map* function, which means it has been assigned more than one input data split, each worker is usually assigned one input split only.

5. ***Map* function** Each *Map* function receives the input data split as a set of (key, value) pairs to process and produce the intermediated (key, value) pairs.

6. ***Combiner* function** This is an optional local function within the map worker which applies to intermediate (key, value) pairs. The user can invoke the *Combiner* function inside the user program.

7. ***Partitioning* function T**he MapReduce data flow, theintermediate (key, value) pairs with identical keys are grouped together because allvalues inside each group should be processed by only one *Reduce* function to generatethe final result.
The intermediate (key, value) pairs produced by each map worker arepartitioned into *R* regions, equal to the number of reduce tasks, by the *Partitioning*function to guarantee that all (key, value) pairs with identical keys are stored in thesame region.
As a result, since reduce worker *i* reads the data of region *i* of all mapworkers, all (key, value) pairs with the same key will be gathered by reduce worker *i* accordingly.

8. **Synchronization** MapReduce applies a simple synchronization policy to coordinate map workers with reduce workers, in which the communication between them starts when all map tasks finish.

9. **Communication** Reduce worker *i*, already notified of the location of region *i* of all map workers, uses a remote procedure call to read the data from the respective region of all map workers. Since all reduce workers read the data from all map workers, all-to-all communication among all map and reduce workers, which incurs network congestion, occurs in the network.

IdentityMapper.java

```
package org.apache.hadoop.mapred.lib;
import java.io.IOException;
import org.apache.hadoop.mapred.Mapper;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.MapReduceBase;
/** Implements the identity function, mapping inputs directly to outputs. */
```

public class IdentityMapper<K, V> extends MapReduceBase implements Mapper<K, V, K, V>
{
/** The identify function. Input key/value pair is written directly to output.*/
public void map(K key, V val,
OutputCollector<K, V> output, Reporter reporter)
throws IOException {
output.collect(key, val);
}
}
A Simple Reduce Function: IdentityReducer
The Hadoop framework calls the reduce function one time for each unique key. The framework provides the key and the set of values that share that key.

IdentityReducer.java

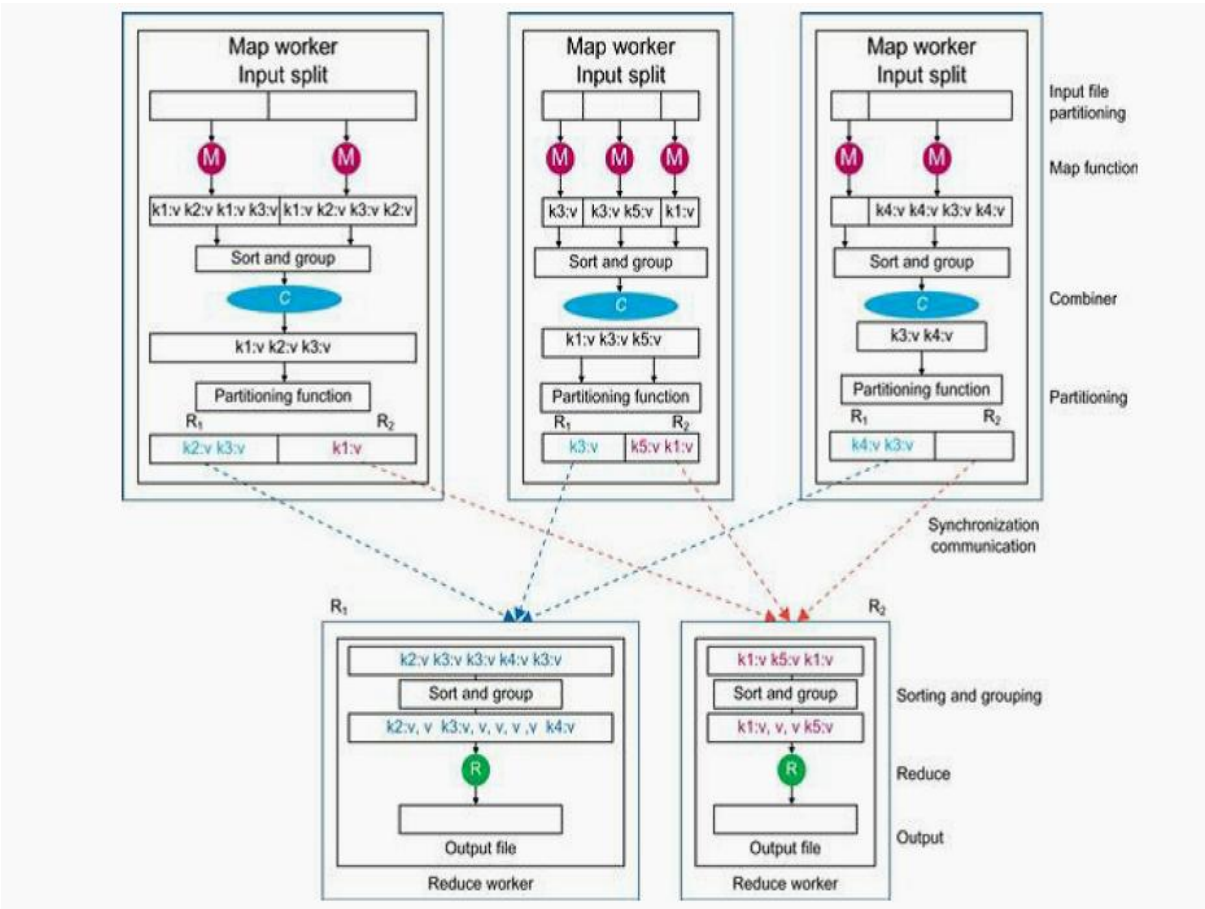
package org.apache.hadoop.mapred.lib;
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.mapred.Reducer;
import org.apache.hadoop.mapred.OutputCollector;
import org.apache.hadoop.mapred.Reporter;
import org.apache.hadoop.mapred.MapReduceBase;
/** Performs no reduction, writing all input values directly to the output. */
public class IdentityReducer<K, V>
extends MapReduceBase implements Reducer<K, V, K, V> {

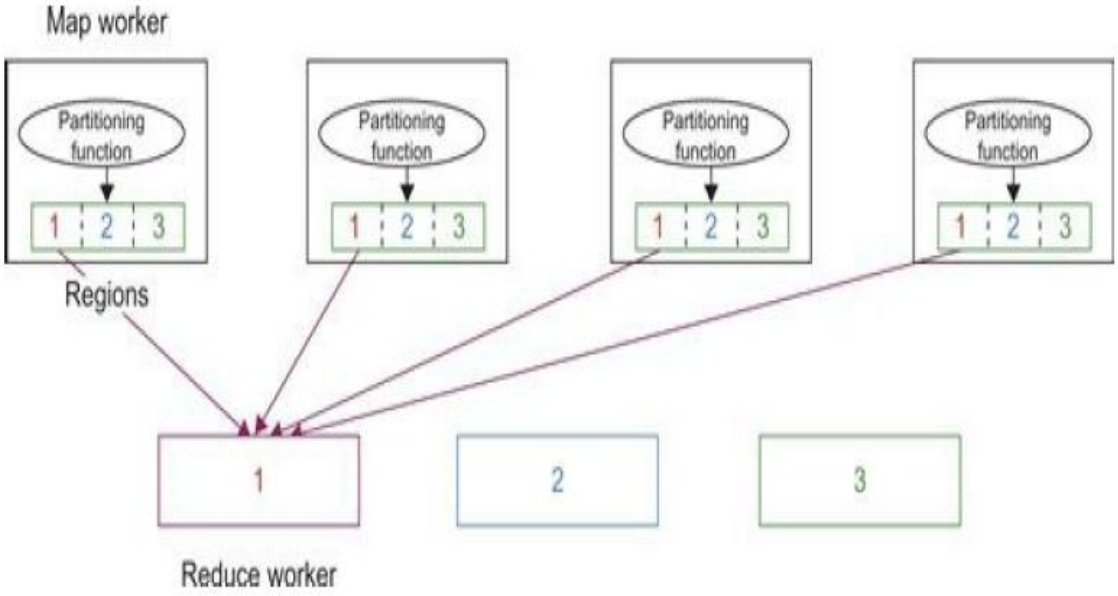
/ Writes all keys and values directly to output. */**
public void reduce(K key, Iterator<V> values,
OutputCollector<K, V> output, Reporter reporter)
throws IOException {
while (values.hasNext()) {
output.collect(key, values.next());
}
}
Steps 10 and 11 correspond to the reduce worker domain:

10. **Sorting and Grouping** When the process of reading the input data is finalized by a reduce worker, the data is initially buffered in the local disk of the reduce worker. Then the reduce worker groups intermediate (key, value) pairs by sorting the data based on their keys, followed by grouping all occurrences of identical keys.

11. ***Reduce* function** The reduce worker iterates over the grouped (key, value) pairs, and foreach unique key, it sends the key and corresponding values to the *Reduce* function. Then this function processes its input data and stores the output results inpredetermined files in the user's program.

Use of Mapreduce partitioning function





6. Write about HDFS.

HDFS: HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.

HDFS Architecture: HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves).

- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).
- The mapping of blocks to DataNodes is determined by the NameNode.
- The NameNode (master) also manages the file system's metadata and namespace.

Blocks

- A disk has a block size, which is the minimum amount of data that it can read or write. Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes.
- HDFS has the concept of a block, but it is a much larger unit—64 MB by default.File0sin HDFS are broken into block-sized chunks, which are stored as independent units.
- Unlike a filesystem for a single disk, a file in HDFS that is smaller than a single block does not occupy a full block's worth of underlying storage.
- The storage subsystem deals with blocks, simplifying storage management and eliminating metadata concerns

Namenodes and Datanodes

- An HDFS cluster has two types of node operating in a master-worker pattern: a *namenode*(the master) and a number of *datanodes*(workers).
- The namenode manages the filesystem namespace.
- It maintains the filesystem tree and the metadata for all the files and directories in the tree.
- The namenode also knows the datanodes on which all the blocks for a given file are located, however, it does not store block locations persistently, since this information is reconstructed from datanodes when the system starts.
- A *client* accesses the filesystem on behalf of the user by communicating with the namenode and datanodes.
- Datanodes are the workhorses of the filesystem.
- Hadoop can be configured so that the namenode writes its persistent state to multiple filesystems.
- The usual configuration choice is to write to local disk as well as a remote NFS mount.
- It is also possible to run a *secondary namenode,* which despite its name does not act as a namenode.
- Its main role is to periodically merge the namespace image with the edit log to prevent the edit log from becoming too large.
- The secondary namenode usually runs on a separate physical machine, since it requires plenty of CPU and as much memory as the namenode to perform the merge.
- It keeps a copy of the merged namespace image, which can be used in the event of the namenode failing.

HDFS Federation

- The namenode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling.
- HDFS Federation, introduced in the 0.23 release series, allows a cluster to scale by adding namenodes, each of which manages a portion of the filesystem namespace. For example, one namenode might manage all the files rooted under */user*, say, and a second Namenode might handle files under */share*.
- Under federation, each namenode manages a *namespace volume*, which is made up of the metadata for the namespace, and a *block pool* containing all the blocks for the files in the namespace.

HDFS High-Availability

- The combination of replicating namenode metadata on multiple filesystems, and using
- the secondary namenode to create checkpoints protects against data loss, but does not provide high-availability of the filesystem.
- The namenode is still a *single point of failure* (SPOF), since if it did fail, all clients—including MapReduce jobs—would be unable to read, write, or list files, because the namenode is the sole repository of the metadata and the file-to-block mapping.
- In such an event the whole Hadoop system would effectively be out of service until a new namenode could be brought online.
- In the event of the failure of the active namenode, the standby takes over its duties to continue servicing client requests without a significant interruption.

A few architectural changes are needed to allow this to happen:

• The namenodes must use highly-available shared storage to share the edit log.

When a standby namenode comes up it reads up to the end of the shared edit log to synchronize its state with the active namenode, and then continues to read new entries as they are written by the active namenode.

• Datanodes must send block reports to both namenodes since the block mappings are stored in a namenode's memory, and not on disk.

• Clients must be configured to handle namenode failover, which uses a mechanism that is transparent to users.

Failover and fencing

- The transition from the active namenode to the standby is managed by a new entity in the system called the *failover controller*.
- Failover controllers are pluggable, but the first implementation uses ZooKeeper to ensure that only one namenode is active.
- Each namenode runs a lightweight failover controller process whose job it is to monitor its namenode for failures and trigger a failover should a namenode fail.
- Failover may also be initiated manually by an administrator, in the case of routine maintenance, for example.

- The HA implementation goes to great lengths to ensure that the previously active namenode is prevented from doing any damage and causing corruption—a method known as *fencing*.
- The system employs a range of fencing mechanisms, including killing the namenode's process, revoking its access to the shared storage directory, and disabling its network port via a remote management command.

7. Expalin in detail on anatomy of File Read and file write in HDFS.

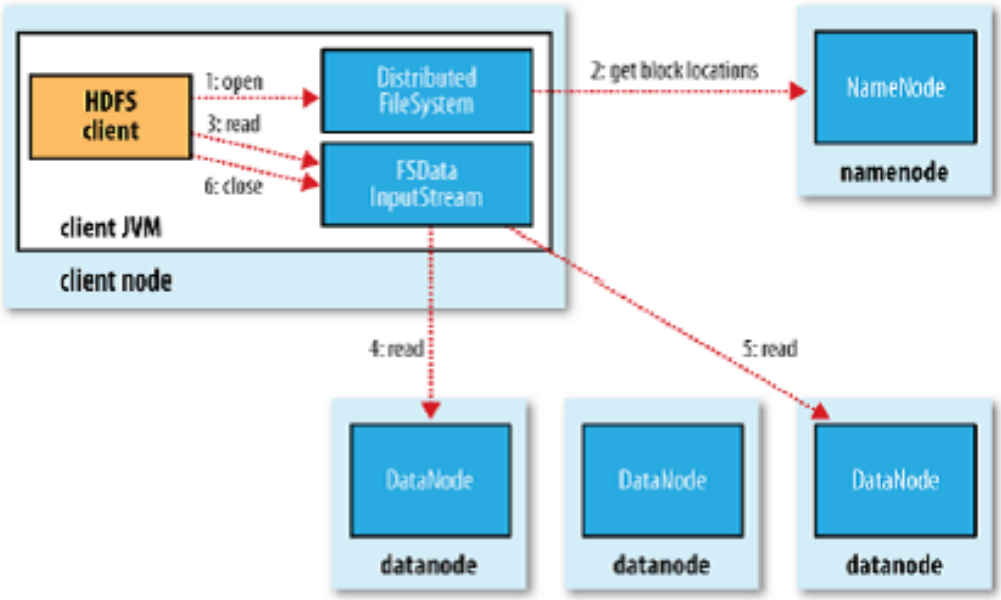
HDFS: HDFS is a distributed file system inspired by GFS that organizes files and stores their data on a distributed computing system.

HDFS Architecture: HDFS has a master/slave architecture containing a single NameNode as the master and a number of DataNodes as workers (slaves).

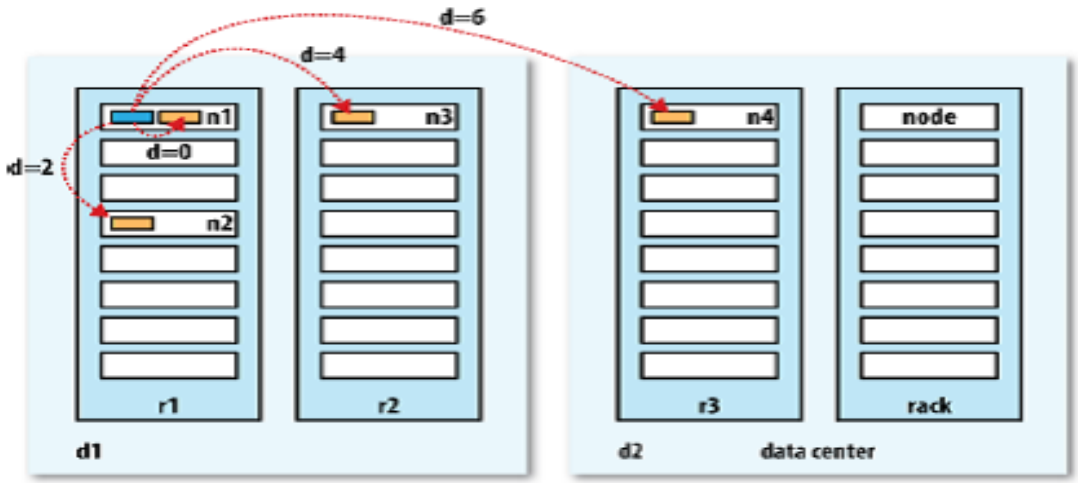
- To store a file in this architecture, HDFS splits the file into fixed-size blocks (e.g., 64 MB) and stores them on workers (DataNodes).
- The mapping of blocks to DataNodes is determined by the NameNode.
- The NameNode (master) also manages the file system's metadata and namespace.

File Read in HDFS

- The client opens the file it wishes to read by calling open () on the FileSystem object, which for HDFS is an instance of DistributedFileSystem.
- DistributedFileSystem calls the namenode, using RPC, to determine the locations of the blocks for the first few blocks in the file.
- The namenode returns the addresses of the datanodes that have a copy of that block.
- If the client is itself a datanode ,then it will read from the local datanode, if it hosts a copy of the block .
- The DistributedFileSystem returns an FSDataInputStream to the client for it to read data from. FSDataInputStream in turn wraps a DFSInputStream, which manages the datanode and namenode I/O.
- The client then calls read () on the stream. DFSInputStream, which has stored the datanode addresses for the first few blocks in the file, then connects to the first (closest) datanode for the first block in the file.
- Data is streamed from the datanode back to the client, which calls read () repeatedly on the stream.
- When the end of the block is reached, DFSInputStream will close the connection to the datanode, then find the best datanode for the next block.
- This happens transparently to the client, which from its point of view is just reading a continuous stream.
- Blocks are read in order with the DFSInputStream opening new connections to datanodes as the client reads through the stream.

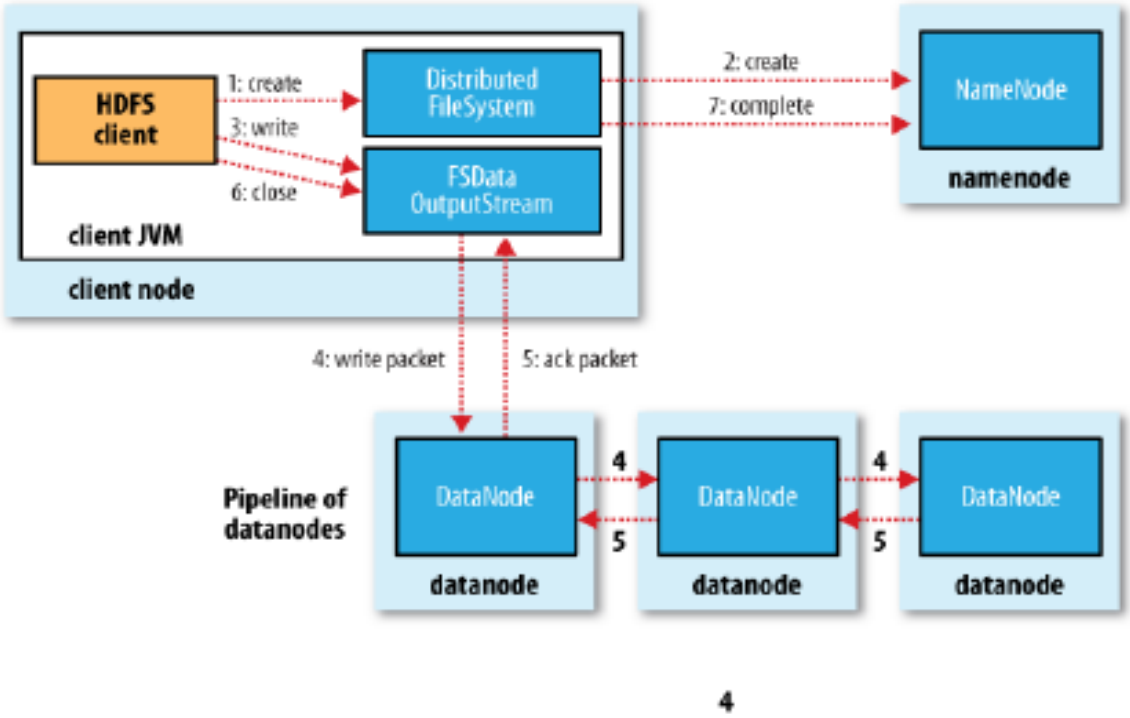


- It will also call the namenode to retrieve the datanode locations for the next batch of blocks as needed. When the client has finished reading, it calls close () on the FSDataInputStream .
- During reading, if the DFSInputStream encounters an error while communicating with a datanode, then it will try the next closest one for that block.
- It will also remember datanodes that have failed so that it doesn't needlessly retry them for later blocks.
- The DFSInputStream also verifies checksums for the data transferred to it from the datanode.
- If a corrupted block is found, it is reported to the namenode before the DFSInput Stream attempts to read a replica of the block from another datanode.



Network distance in Hadoop

FileWrite:

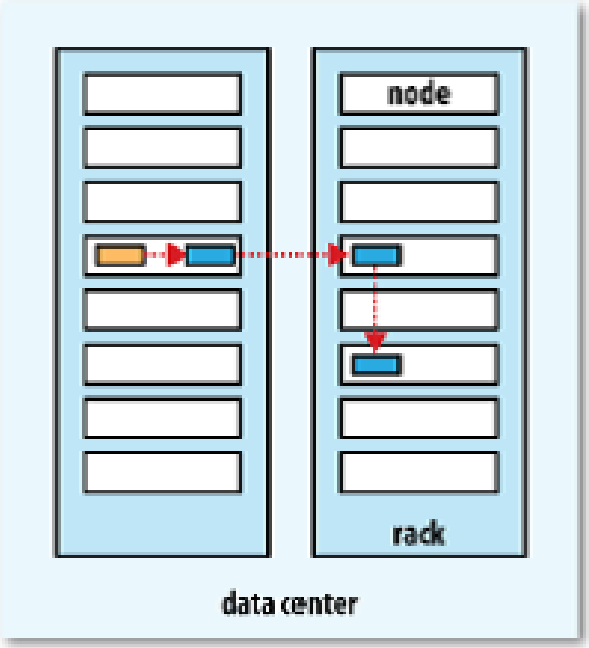


4

- The client creates the file by calling create () on DistributedFileSystem. DistributedFileSystem makes an RPC call to the namenode to create a new file in the filesystem's namespace, with no blocks associated with it (step 2).
- The namenode performs various checks to make sure the file doesn't already exist, and that the client has the right permissions to create the file. If these checks pass, the namenode makes a record of the new file.
- The DistributedFileSystem returns an FSDataOutputStream for the client to start writing data to. Just as in the read case, FSDataOutputStream wraps a DFSOutput Stream, which handles communication with the datanodes and namenode.
- As the client writes data (step 3), DFSOutputStream splits it into packets, which it writes to an internal queue, called the *data queue*.
- The data queue is consumed by the Data Streamer, whose responsibility it is to ask the namenode to allocate new blocks by picking a list of suitable datanodes to store the replicas.
- The DataStreamer streams the packets to the first datanode in the pipeline, which stores the packet and forwards it to the second datanode in the pipeline. Similarly, the second datanode stores the packet and forwards it to the third (and last) datanode in the pipeline (step 4).
- DFSOutputStream also maintains an internal queue of packets that are waiting to be acknowledged by datanodes, called the *ack queue*.
- A packet is removed from the ack queue only when it has been acknowledged by all the datanodes in the pipeline (step 5).

- If a datanode fails while data is being written to it, then the following actions are taken, which are transparent to the client writing the data

A replica pipleline



- First the pipeline is closed, and any packets in the ack queue are added to the front of the data queue so that datanodes that are downstream from the failed node will not miss any packets.
- The failed datanode is removed from the pipeline and the remainder of the block's data is written to the two good datanodes in the pipeline.
- The namenode notices that the block is under-replicated, and it arranges for a further replica to be created on another node.
- Subsequent blocks are then treated as normal. It's possible, but unlikely, that multiple datanodes fail while a block is being written.
- As long as dfs.replication.min replicas (default one) are written, the write will succeed, and the block will be asynchronously replicated across the cluster until its target replication factor is reached.
- When the client has finished writing data, it calls close () on the stream (step 6). This action flushes all the remaining packets to the datanode pipeline and waits for acknowledgments before contacting the namenode to signal that the file is complete (step 7).
- The namenode already knows which blocks the file is made up.

8. Explain about the Command-Line Interface.

- There are many other interfaces to HDFS, but the command line is one of the simplest and, to many developers, the most familiar.
- The first is fs.default.name, set to *hdfs://localhost/*, which is used to set a default filesystem for Hadoop. Filesystems are specified by a URI, and here we have used an hdfsURI to configure Hadoop to use HDFS by default

Basic Filesystem Operations

- The filesystem is ready to be used, and we can do all of the usual filesystem operations such as reading files, creating directories, moving files, deleting data, and listing directories.
- You can type hadoopfs -help to get detailed help on every command.
- Start by copying a file from the local filesystem to HDFS:

```
% hadoopfs -copyFromLocal input/docs/quangle.txt hdfs://localhost/user/tom/
quangle.txt
```

by copying a file from the local filesystem to HDFS:

```
% hadoopfs -copyFromLocal input/docs/quangle.txt hdfs://localhost/user/tom/
quangle.txt
```

- This command invokes Hadoop'sfilesystem shell command fs, which supports a
- number of subcommands—in this case, we are running -copyFromLocal. The local file *quangle.txt* is copied to the file */user/tom/quangle.txt* on the HDFS instance running on localhost.

```
% hadoopfs -copyFromLocal input/docs/quangle.txt /user/tom/quangle.txt
```

We could also have used a relative path and copied the file to our home directory in HDFS, which in this case is */user/tom*:

```
% hadoopfs -copyFromLocal input/docs/quangle.txt quangle.txt
```

Let's copy the file back to the local filesystem and check whether it's the same:

```
% hadoopfs -copyToLocal quangle.txt quangle.copy.txt
% md5 input/docs/quangle.txt quangle.copy.txt
```

MD5 (input/docs/quangle.txt) = a16f231da6b05e2ba7a339320e7dacd9
MD5 (quangle.copy.txt) = a16f231da6b05e2ba7a339320e7dacd9
The MD5 digests are the same, showing that the file survived its trip to HDFS and is back intact.

HadoopFilesystems

- Hadoop has an abstract notion of filesystem, of which HDFS is just one implementation.
- The Java abstract class org.apache.hadoop.fs.FileSystem represents a filesystem
- in Hadoop, and there are several concrete implementations

Filesystem	URI scheme	Java implementation (all under org.apache.hadoop)	Description
Local	*file*	`fs.LocalFileSystem`	A filesystem for a locally connected disk with client-side checksums. Use `RawLocalFileSystem` for a local filesystem with no checksums. See "LocalFileSystem" on page 84.
HDFS	*hdfs*	`hdfs.DistributedFileSystem`	Hadoop's distributed filesystem. HDFS is designed to work efficiently in conjunction with MapReduce.
HFTP	*hftp*	`hdfs.HftpFileSystem`	A filesystem providing read-only access to HDFS over HTTP. (Despite its name, HFTP has no connection with FTP.) Often used with *distcp* (see "Parallel Copying with distcp" on page 76) to copy data between HDFS clusters running different versions.
HSFTP	*hsftp*	`hdfs.HsftpFileSystem`	A filesystem providing read-only access to HDFS over HTTPS. (Again, this has no connection with FTP.)
WebHDFS	*webhdfs*	`hdfs.web.WebHdfsFileSystem`	A filesystem providing secure read-write access to HDFS over HTTP. WebHDFS is intended as a replacement for HFTP and HSFTP.
HAR	*har*	`fs.HarFileSystem`	A filesystem layered on another filesystem for archiving files. Hadoop Archives are typically used for archiving files in HDFS to reduce the namenode's memory usage. See "Hadoop Archives" on page 78.
KFS (CloudStore)	*kfs*	`fs.kfs.KosmosFileSystem`	CloudStore (formerly Kosmos filesystem) is a distributed filesystem like HDFS or Google's GFS, written in C++. Find more information about it at *http://kosmosfs.sourceforge.net/*.
FTP	*ftp*	`fs.ftp.FTPFileSystem`	A filesystem backed by an FTP server.
S3 (native)	*s3n*	`fs.s3native.NativeS3FileSystem`	A filesystem backed by Amazon S3. See *http://wiki.apache.org/hadoop/AmazonS3*.
S3 (block-based)	*s3*	`fs.s3.S3FileSystem`	A filesystem backed by Amazon S3, which stores files in blocks (much like HDFS) to overcome S3's 5 GB file size limit.

9. Explain about Java Interface.

Reading Data from a Hadoop URL

One of the simplest ways to read a file from a Hadoopfilesystem is by using a java.net.URL object to open a stream to read the data from. The general idiom is:

```
InputStream in = null;
Try
 {
in = new URL("hdfs://host/path").openStream();
// process in
}
 finally {
IOUtils.closeStream(in);
}
```

IOUtils class that comes with Hadoop for closing the streamin the finally clause, and also for copying bytes between the input stream and theoutput stream

Reading Data Using the FileSystem API

Sometimes it is impossible to set a URLStreamHandlerFactory for your application. In this case, we will need to use the FileSystem API to open an input stream for a file

A file in a Hadoopfilesystem is represented by a Hadoop Path object. FileSystem is a general filesystem API, so the first step is to retrieve an instance for the filesystem we want to use—HDFS in this case. There are several static factory methods for getting a FileSystem instance:

public static FileSystem get(Configuration conf) throws IOException
public static FileSystem get(URI uri, Configuration conf) throws IOException
public static FileSystem get(URI uri, Configuration conf, String user) throws IOException

Displaying files from a Hadoopfilesystem on standard output by using the FileSystem directly

```
public class FileSystemCat {
public static void main(String[] args) throws Exception {
String uri = args[0];
Configuration conf = new Configuration();
FileSystemfs = FileSystem.get(URI.create(uri), conf);
InputStream in = null;
try {
in = fs.open(new Path(uri));
IOUtils.copyBytes(in, System.out, 4096, false);
} finally {
IOUtils.closeStream(in);
}
}
}
```

FSDataInputStream
The open() method on FileSystem actually returns a FSDataInputStream rather than a standard java.io class. This class is a specialization of java.io.DataInputStream with support for random access

The Seekable interface permits seeking to a position in the file and a query method for the current offset from the start of the file (getPos()):

```
public interface Seekable
{
void seek(long pos) throws IOException;
long getPos() throws IOException;
}
```

The read() method reads up to length bytes from the given position in the file into the buffer at the given offset in the buffer. The return value is the number of bytes actually read: callers should check this value as it may be less than length. The readFully() methods will read length bytes into the buffer.

Writing Data
The FileSystem class has a number of methods for creating a file. The simplest is the method that takes a Path object for the file to be created and returns an output stream to write to:
public FSDataOutputStream create(Path f) throws IOException

As an alternative to creating a new file, you can append to an existing file using the **append() method (there are also some other overloaded versions):**
public FSDataOutputStream append(Path f) throws IOException

Copying a local file to a Hadoopfilesystem

```
public class FileCopyWithProgress {
public static void main(String[] args) throws Exception {
String localSrc = args[0];
String dst = args[1];
InputStream in = new BufferedInputStream(new FileInputStream(localSrc));
Configuration conf = new Configuration();
FileSystemfs = FileSystem.get(URI.create(dst), conf);
OutputStream out = fs.create(new Path(dst), new Progressable() {
public void progress() {
System.out.print(".");
}
});
IOUtils.copyBytes(in, out, 4096, true); }}
```

Typical usage:
% hadoopFileCopyWithProgress input/docs/1400-8.txt hdfs://localhost/user/tom/
1400-8.txt
.....

FSDataOutputStream

The create() method on FileSystem returns an FSDataOutputStream, which, like FSDataInputStream, has a method for querying the current position in the file:
package org.apache.hadoop.fs;

Directories

FileSystem provides a method to create a directory:
public booleanmkdirs(Path f) throws IOException
This method creates all of the necessary parent directories if they don't already exist,
just like the java.io.File'smkdirs() method. It returns true if the directory (and all parent directories) was (were) successfully created.

Querying the Filesystem

File metadata: FileStatus
An important feature of any filesystem is the ability to navigate its directory structure and retrieve information about the files and directories that it stores. The FileStatusclass encapsulates filesystem metadata for files and directories, including file length,block size, replication, modification time, ownership, and permission information.The method getFileStatus() on FileSystem provides a way of getting a FileStatusobject for a single file or directory.

```
public void fileStatusForDirectory() throws IOException {
Path dir = new Path("/dir");
FileStatus stat = fs.getFileStatus(dir);
assertThat(stat.getPath().toUri().getPath(), is("/dir"));
assertThat(stat.isDir(), is(true));
assertThat(stat.getLen(), is(0L));
assertThat(stat.getModificationTime(),
is(lessThanOrEqualTo(System.currentTimeMillis())));
assertThat(stat.getReplication(), is((short) 0));
assertThat(stat.getBlockSize(), is(0L));
assertThat(stat.getOwner(), is("tom"));
assertThat(stat.getGroup(), is("supergroup"));
assertThat(stat.getPermission().toString(), is("rwxr-xr-x"));
}
}
```

Listing files

public FileStatus[] listStatus(Path f) throws IOException
public FileStatus[] listStatus(Path f, PathFilter filter) throws IOException
public FileStatus[] listStatus(Path[] files) throws IOException
public FileStatus[] listStatus(Path[] files, PathFilter filter) throws IOException

File patterns

We can use wildcard characters to match multiple files.
with a single expression, an operation that is known as globbing. Hadoop provides two

FileSystem method for processing globs:
public FileStatus[] globStatus(Path pathPattern) throws IOException
public FileStatus[] globStatus(Path pathPattern, PathFilter filter) throws IOException
The globStatus() method returns an array of FileStatus objects whose paths match the supplied pattern, sorted by path.

Deleting Data

Use the delete() method on FileSystem to permanently remove files or directories:
public boolean delete(Path f, boolean recursive) throws IOException
If f is a file or an empty directory, then the value of recursive is ignored

Glob characters and their meanings

Glob	Name	Matches
*	*asterisk*	Matches zero or more characters
?	*question mark*	Matches a single character
[ab]	*character class*	Matches a single character in the set {a, b}
[^ab]	*negated character class*	Matches a single character that is not in the set {a, b}
[a-b]	*character range*	Matches a single character in the (closed) range [a, b], where a is lexicographically less than or equal to b
[^a-b]	*negated character range*	Matches a single character that is not in the (closed) range [a, b], where a is lexicographically less than or equal to b
{a,b}	*alternation*	Matches either expression a or b
\c	*escaped character*	Matches character c when it is a metacharacter

Glob	Expansion
/*	*/2007 /2008*
/*/*	*/2007/12 /2008/01*
/*/12/*	*/2007/12/30 /2007/12/31*
/200?	*/2007 /2008*
/200[78]	*/2007 /2008*
/200[7-8]	*/2007 /2008*
/200[^01234569]	*/2007 /2008*
/*/*/{31,01}	*/2007/12/31 /2008/01/01*
/*/*/3{0,1}	*/2007/12/30 /2007/12/31*
/*/{12/31,01/01}	*/2007/12/31 /2008/01/01*

PathFilter

Glob patterns are not always powerful enough to describe a set of files you want to access. For example, it is not generally possible to exclude a particular file using a glob pattern. The listStatus() and globStatus() methods of FileSystem take an optional PathFilter, which allows programmatic control over matching:

package org.apache.hadoop.fs;

```
public interface PathFilter
 {
boolean accept(Path path);
}
```

PathFilter is the equivalent of java.io.FileFilter for Path objects rather than File objects.

Unit IV

Two marks

1. What are the organizations where grid standards developed?

OGSA – Open Grid Services Architecture
Global Grid Forum

2. Give some middleware packages of grid.

BONIC – Berkeley Open Infrastructure for Network Computing
UNICORE
Globus GT4
CGSP in ChinaGrid
Condor – G
Sun Grid Engine

3. Define Condor.

Condor is a software tool for high throughput distributed batch computing. It was designed to explore the idle cycles of a network of distributed computers.
The major components are matchmaker, user agent and resources.

4. Define Sun Grid Engine middleware package.

SGE was developed by Sun Microsystems in response to increasing requirement of business grid application. SGE features are applied to private grids and local clusters within an enterprise or campus for intranet based cluster or grid applications.

5. What is Globus Toolkit Architecture?

The Globus Toolkit is an open middleware library for the grid computing communities. It was funded by DARPA and it address common problems and issues related to:
Grid resource management
Grid resource discovery
Communication
Security
Fault detection

Portability

6. What is the working model of map reduce?

MapReduce works by breaking the processing into two phases: the map phase and the reduce phase. Each phase has key-value pairs as input and output, the types of which may be chosen by the programmer. The programmer also specifies two functions: the map function and the reduce function.

7. How mapreduce is implemented in java?

A java program has map function, a reduce function, and some code to run the job. The map function is represented by an implementation of the Mapper interface, which declares a map() method. The Mapper interface is a generic type, with four formal type parameters that specify theinput key, input value, output key, and output value types. The map() method also provides an instance of OutputCollectorto write the output to. A JobConfobject forms the specification of the job. It gives you control over how the job is run. The setOutputKeyClass() and setOutputValueClass() methods control the output typesfor the map and the reduce functions. The output path (of which there is only one) is specified by the static setOutputPath() method on FileOutputFormat. To specify the map and reduce types to use via the setMapperClass() and setReducerClass() methods.The input types are controlled via the input format, which we have not explicitly setsince we are using the default TextInputFormat.

8. How does hadoop work?

A MapReduce job is a unit of work that the client wants to be performed: it consists of the input data, the MapReduce program, and configuration information. Hadoop runs the job by dividing it into tasks, of which there are two types: map tasks and reduce tasks.

There are two types of nodes that control the job execution process: a jobtracker and a number of tasktrackers. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers. Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If atask fails, the jobtracker can reschedule it on a different tasktrackerHadoop divides the input to a MapReduce job into fixed-size pieces called input splits, or just splits. Hadoop creates one map task for each split, which runs the userdefined map function for each record in the split.Hadoop allows the user to specify a combiner function to be run on the map output—the combiner function'soutput forms the input to the reduce function.

9. What is mean by data locatlity?

Hadoop does its best to run the map task on a node where the input data resides in HDFS. This is called the data locality optimization.

10. What are hadoop pipes?

Hadoop Pipes is the name of the C++ interface to HadoopMapReduce. Unlike Streaming, which uses standard input and output to communicate with the map and reduce code, Pipes uses sockets as the channel over which the tasktracker communicates with the process running the C++ map or reduce function

11. Define hadoop streaming and working.

Hadoop provides an API to MapReduce that allows you to write your map and reduce functions in languages other than Java. Hadoop Streaming uses Unix standard streams as the interface

between Hadoop and your program, so you can use any language that can read standard input and write to standard output to write your MapReduce
program.

12. Define hadoop.

Hadoop is an open-source framework that allows to store and process big data in a distributed environment across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage.

13. Mention the disadvantages of HDFS.

Lot of small files
Low latency data access
Multiple writes

14. Define name node and data node.

An HDFS cluster has two types of node operating in a master-worker pattern: a namenode(the master) and a number of datanodes(workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. Datanodes are the workhorses of the filesystem. They store and retrieve blocks when they are told to, and they report back to the namenode periodically with lists of blocks that they are storing.

15. What is HDFS?

The Hadoop Distributed File System (HDFS) is designed to store very large data sets reliably, and to stream those data sets at high bandwidth to user applications. In a large cluster, thousands of servers both host directly attached storage and execute user application tasks. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size.

16. Mention the pros of HDFS.

Handle very large file
Streaming data access
Execute through the Commodity hardware

17. Define blocks.

A disk has a block size, which is the minimum amount of data that it can read or write. Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size. Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes.

18. What are the file permissions in HDFS?

Each file and directory has an owner, a group, and a mode. The mode is made up of the permissions for the user who is the owner, the permissions for the users who are members of the group, and the permissions for users who are neither the owners nor members of the group.

19. What is fuse?

Filesystem in Userspace(FUSE) allows filesystems that are implemented in user space to be integrated as a Unixfilesystem.

20. Mention the APIs for hadooop.

C
Thrift
Fuse
WebDAV
Http
Java

21. Write about seekable interface.

The Seekable interface permits seeking to a position in the file and a query method for the current offset from the start of the file (getPos()):

```
public interface Seekable
 {
void seek(long pos) throws IOException;
longgetPos() throws IOException;
}
```

Calling seek() with a position that is greater than the length of the file will result in an IOException.

22. What is FSData Output Stream?

FSDataOutputStream

The create() method on FileSystem returns an FSDataOutputStream, which, like FSDataInputStream, has a method for querying the current position in the file:

```
packageorg.apache.hadoop.fs;
public class FSDataOutputStream extends DataOutputStream implements Syncable {
public long getPos() throws IOException {
// implementation elided
}
// implementation elided
}
```

SET II

1. What is The Globus Toolkit Architecture (GT4)

The Globus Toolkit, started in 1995 with funding from DARPA, is an open middleware library for the grid computing communities. The toolkit addresses common problems and issues related to grid resource discovery,management, communication, security, fault detection, and portability. The library includes a rich set of service implementations.

2. What is GT4 library?

The high-level services and tools, such as MPI, Condor-G, and Nirod/G, are developed by third parties for generalpurpose distributed computing applications. The local

services, such as LSF, TCP, Linux, and Condor, are at the bottom level and are fundamental tools supplied by other developers.

3. What is meant by Globus Container ?

The Globus Container provides a basic runtime environment for hosting the web services needed to execute grid jobs.

4. What are the Functional Modules in Globus GT4 Library ?

· Global Resource Allocation Manager
· Communication
· Grid Security Infrastructure
· Monitory and Discovery Service
· Health and Status
· Global Access of Secondary Storage
· Grid File Transfer

5. What is meant by input splitting ?

For the framework to be able to distribute pieces of the job to multiple machines, it needs to fragment the input into individual pieces, which can in turn be provided as input to the individual distributed tasks. Each fragment of input is called an input split.

6. What are the five categories of Globus Toolkit 4 ?

- Common runtime components
- Security
- Data management
- Information services
- Execution management

7. What are the are the available input formats?

- KeyValueTextInputFormat
- TextInputFormant
- NLineInputFormat
- MultiFileInputFormat
- SequenceFIleInputFormat

8. What is meant by HDFS ?

Hadoop comes with a distributed filesystem called HDFS, which stands for Hadoop Distributed Filesystem. HDFS is a filesystem designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware.

9. What is meant by Block

A disk has a block size, which is the minimum amount of data that it can read or write.Filesystems for a single disk build on this by dealing with data in blocks, which are an integral multiple of the disk block size. Filesystem blocks are typically a few kilobytes in size, while disk blocks are normally 512 bytes. HDFS, too, has the concept of a block, but it is a much larger unit—64 MB by default.

10. Differentiate Namenodes and Datanodes

An HDFS cluster has two types of node operating in a master-worker pattern: a namenode (the master) and a number of datanodes(workers). The namenode manages the filesystem namespace. It maintains the filesystem tree and the metadata for all the files and directories in the tree. This information is stored persistently on the local disk in the form of two

files: the namespace image and the edit log. The namenode also knows the datanodes on which all the blocks for a given file are located.

11. List the various Hadoopfilesystems ?

Local,HDFS, HFTP, HSFTP, WebHDFS.

12. What is meant by FUSE?

Filesystem in Userspace (FUSE) allows filesystems that are implemented in user space to be integrated as a Unixfilesystem. Hadoop's Fuse-DFS contrib module allows any Hadoopfilesystem (but typically HDFS) to be mounted as a standard filesystem.

13. What is Hadoop File system ?

Hadoop is written in Java, and all Hadoopfilesystem interactions are mediated through the Java API. The filesystem shell, for example, is a Java application that uses the Java FileSystem class to provide filesystem operations.

14. How to Reading Data from a Hadoop URL

One of the simplest ways to read a file from a Hadoopfilesystem is by using a java.net.URL object to open a stream to read the data from. The general idiom is:

```
InputStream in = null;
try {
in = new URL("hdfs://host/path").openStream();
// process in} finally {
IOUtils.closeStream(in);
}
```

15. How to write data in Hadoop?

The FileSystem class has a number of methods for creating a file. The simplest is the method that takes a Path object for the file to be created and returns an output stream to write to:

```
publicFSDataOutputStream create(Path f) throws IOException
```

16. How are Deleting Datas are Deleted in Hadoop ?

Use the delete() method on FileSystem to permanently remove files or directories:

```
publicboolean delete(Path f, boolean recursive) throws IOException
```

If f is a file or an empty directory, then the value of recursive is ignored.

17. Illustrate MapReduce logical data flow

18. What are two types of nodes that control the job execution process?

ajobtracker and a number of tasktrackers controls the job execution process. The jobtracker coordinates all the jobs run on the system by scheduling tasks to run on tasktrackers. Tasktrackers run tasks and send progress reports to the jobtracker, which keeps a record of the overall progress of each job. If a task fails, the jobtracker can reschedule it on a different tasktracker.

19. Illustrate MapReduce data flow with a single reduce task

20. Illustrate MapReduce data flow with multiple reduce tasks

CS6703 / GCC UNIT V- SECURITY

Trust models for Grid security environment – Authentication and Authorization methods – Grid security infrastructure – Cloud Infrastructure security: network, host and application level – aspects of data security, provider data and its security, Identity and access management architecture, IAM practices in the cloud, SaaS, PaaS, IaaS availability in the cloud, Key privacy issues in the cloud.

5.1 Trust models for Grid security environment

Definition of Trust

- Trust is the firm belief in the competence of an entity to behave as expected such that this firm belief is a dynamic value associated with the entity and is subject to the entity's behavior and applies only within a specific context at a given time.

Trust

- Trust value is a continuous and dynamic value in the range of [0,1]

  1 means very trustworthy

  0 means very untrustworthy

- It is built on past experience
- It is context based (under different context may have different trust value)

Reputation

- When making trust-based decisions, entities can rely on others for information regarding to a specific entity.
- The information regarding to a specific entity x is defined as the reputation of entity x.

Definition of Reputation

- The reputation of an entity is an expectation of its behavior based on other entities' observations or information about the entity's past behavior within a specific context at a given time.

Evaluating Trust and Reputation

- Trusts decays with time
- Entities may form alliances and they may trust their allies and business partners more than others

Trust value is based on the combination of direct trust and reputation

- Trust models for Grid security environment
- Many potential security issues may occur in a grid environment if qualified security mechanisms are not in place. These issues include

    1. Network sniffers,
    2. Out-of-control access,
    3. Faulty operation,
    4. Malicious operation,
    5. Integration of local security mechanisms,
    6. Delegation,
    7. Dynamic resources and services,
    8. Attack provenance, and so on.
    9. Security demand (SD) and Trust Index (TI)

    On the one hand, a user job demands the resource site to provide security assurance by issuing a security demand (SD).

    On the other hand, the site needs to reveal its trustworthiness, called its trust index (TI).

    These two parameters must satisfy a security assurance condition: $TI \geq SD$ during the j ob mapping process.

    When determining its security demand, users usually care about some typical attributes. These attributes and their values are dynamically changing and depend heavily on the trust model, security policy, accumulated reputation, self-defense capability, attack history, and site vulnerability.

    Three challenges are outlined below to establish the trust among grid sites

    The first challenge is integration with existing systems and technologies.

- The resources sites in a grid are usually heterogeneous and autonomous.
- It is unrealistic to expect that a single type of security can be compatible with and adopted by every hosting environment.

- At the same time, existing security infrastructure on the sites cannot be replaced overnight. Thus, to be successful, grid security architecture needs to step up to the challenge of integrating with existing security architecture and models across platforms and hosting environments.

The second challenge is interoperability with different "hosting environments."

Services are often invoked across multiple domains, and need to be able to interact with one another.

The interoperation is demanded at the protocol, policy, and identity levels. For all these levels, interoperation must be protected securely.

The third challenge is to construct trust relationships among interacting hosting environments.

- Grid service requests can be handled by combining resources on multiple security domains.
- Trust relationships are required by these domains during the end-to-end traversals.
- A service needs to be open to friendly and interested entities so that they can submit requests and access securely.

Trust Model

- Resource sharing among entities is one of the major goals of grid computing. A trust relationship must be established before the entities in the grid interoperate with one another.
- To create the proper trust relationship between grid entities, two kinds of trust models are often used.

1. One is the PKI -based model, which mainly exploits the PKI to authenticate and authorize entities.
2. The other is the reputation-based model.

A Generalized Trust Model

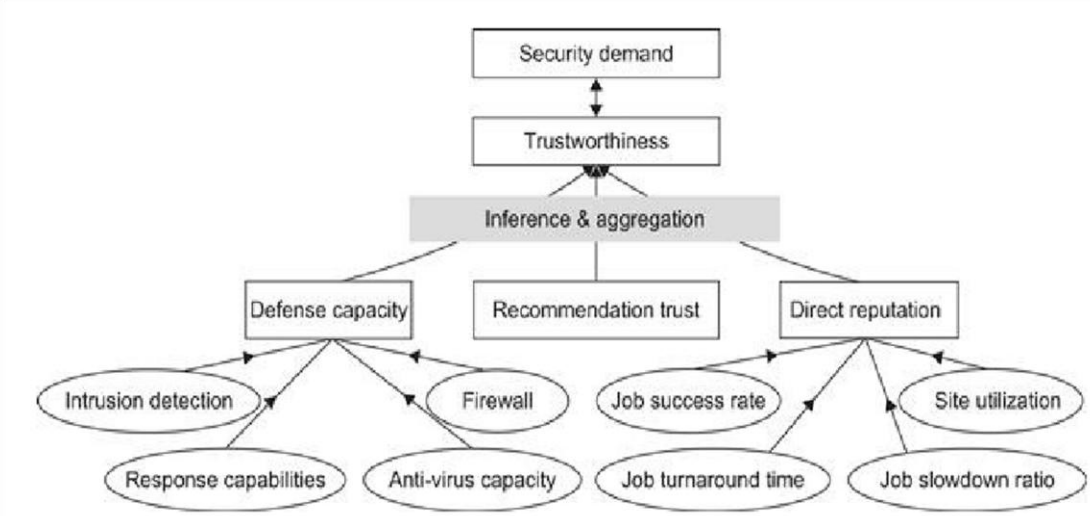


Figure shows a general trust model.

- The three major factors which influence the trustworthiness of a resource site.
- An inference module is required to aggregate these factors. Followings are some existing inference or aggregation methods. An intra-site fuzzy inference procedure is called to assess defense capability and direct reputation.
- Defense capability is decided by the firewall, *intrusion detection system (I DS), intrusion response capability, and anti-virus capacity of the individual* resource site.
- Direct reputation is decided based on the j ob success rate, site utilization, job turnaround time, and j ob slowdown ratio measured.
- Recommended trust is also known as secondary trust and is obtained indirectly over the grid network.

Reputation-Based Trust Model

- In a reputation-based model, jobs are sent to a resource site only when the site is trustworthy to meet users' demands.
- The site trustworthiness is usually calculated from the following information: the defense capability, direct reputation, and recommendation trust. The defense capability refers to the site's ability to protect itself from danger.
- It is assessed according to such factors as intrusion detection, firewall, response capabilities, anti-virus capacity, and so on.
- Direct reputation is based on experiences of prior jobs previously submitted to the site.
- The reputation is measured by many factors such as prior j ob execution success rate, cumulative site utilization, job turnaround time, job slowdown ratio, and so on.
- A positive experience associated with a site will improve its reputation. On the contrary, a negative experience with a site will decrease its reputation.

A Fuzzy-Trust Model

- In this model, the job ***security demand (SD) is supplied by the user programs. The trust index(TI) of a resource site is aggregated through the fuzzy-logic inference process*** over all related parameters. Specifically, one can use a two-level fuzzy logic to estimate the aggregation of numerous trust parameters and security attributes into scalar quantities that are easy to use in the j ob scheduling and resource mapping process.
- The TI is normalized as a single real number with 0 representing the condition with the highest risk at a site and 1 representing the condition which is totally risk-free or fully trusted.
- The fuzzy inference is accomplished through four steps: *fuzzification, inference, aggregation, and defuzzification.*
- *The second salient feature of the trust model is that if a* site's trust index cannot match the job security demand (i.e., SD > TI), the trust model could deduce detailed security features to guide the site security upgrade as a result of tuning the fuzzy system.

5.2 Authentication and Authorization Methods

- The major authentication methods in the grid include passwords, PKI, and Kerberos. The password is the simplest method to identify users, but the most vulnerable one to use.
- The PKI is the most popular method supported by GSI. To implement PKI , we use a trusted third party, called the ***certificate authority (CA).*** *Each user applies a unique pair of* public and private keys. The *public keys are issued by the CA by issuing a certificate, after* recognizing a legitimate user.
- The *private key is exclusive for each user to use, and is* unknown to any other users. A digital certificate in IEEE X.509 format consists of the user name, user public key, CA name, and a secrete signature of the user.

Authorization for Access Control

- The authorization is a process to exercise access control of shared resources.
- Decisions can be made either at the access point of service or at a centralized place.
- Typically, the resource is a host that provides processors and storage for services deployed on it. Based on a set predefined policies or rules, the resource may enforce access for local services.
- The central authority is a special entity which is capable of issuing and revoking polices of access rights granted to remote accesses.
- The authority can be classified into three categories:

1. Attribute authorities - issue attribute assertions
2. Policy authorities - authorization policies , and
3. Identity authorities - issue certificates

The authorization server makes the final authorization decision.

Three Authorization Models

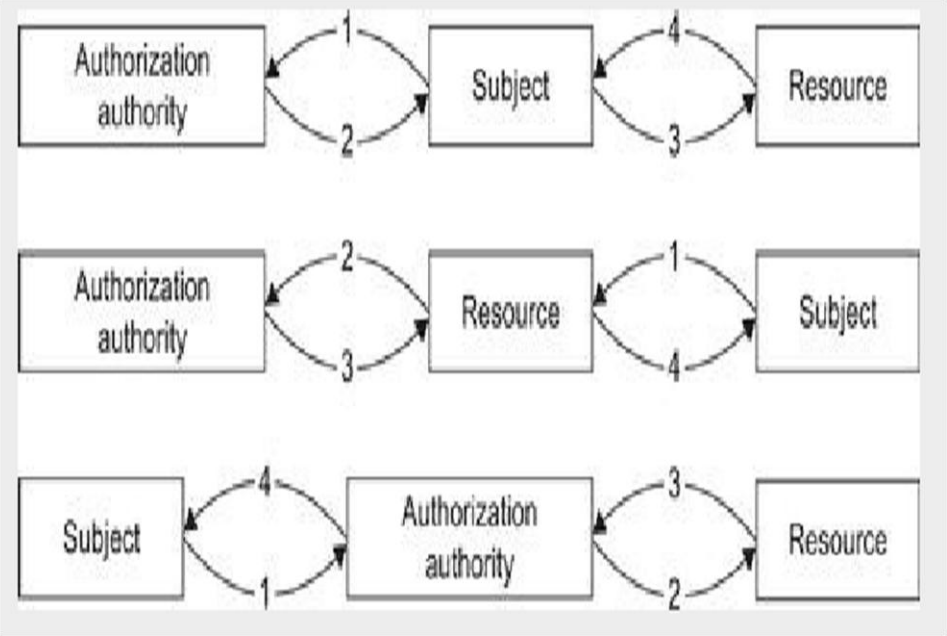


Figure shows three authorization models.

1. The subject-push model
2. The resource-pulling model
3. The authorization agent model

Three authorization model

- The *subject-push model is shown at the top diagram. The user* conducts handshake with the authority first and then with the resource site in a sequence.
- The *resource-pulling model puts the resource in the middle. The user checks the* resource first. Then the resource contacts its authority to verify the request, and the authority authorizes at step 3. Finally the resource accepts or rejects the request from the subject at step 4.
- The *authorization agent model puts the authority in the middle. The* subject check with the authority at step 1 and the authority makes decisions on the access of the requested resources. The authorization process is complete at steps 3 and 4 in the reverse direction.

5.3 Grid Security Infrastructure (GSI)

- The grid is increasingly deployed as a common approach to constructing dynamic, inter domain, distributed computing and data collaborations, still there is "lack of security/trust between different services".
- It is still an important challenge of the grid.
- The **grid requires** a security infrastructure with the following **properties**:

1. ***easy to use;***
2. ***conforms with the virtual organization (VO's) security needs while working well with site policies of each resource provider site; and***
3. ***provides appropriate authentication and encryption of all interactions.***

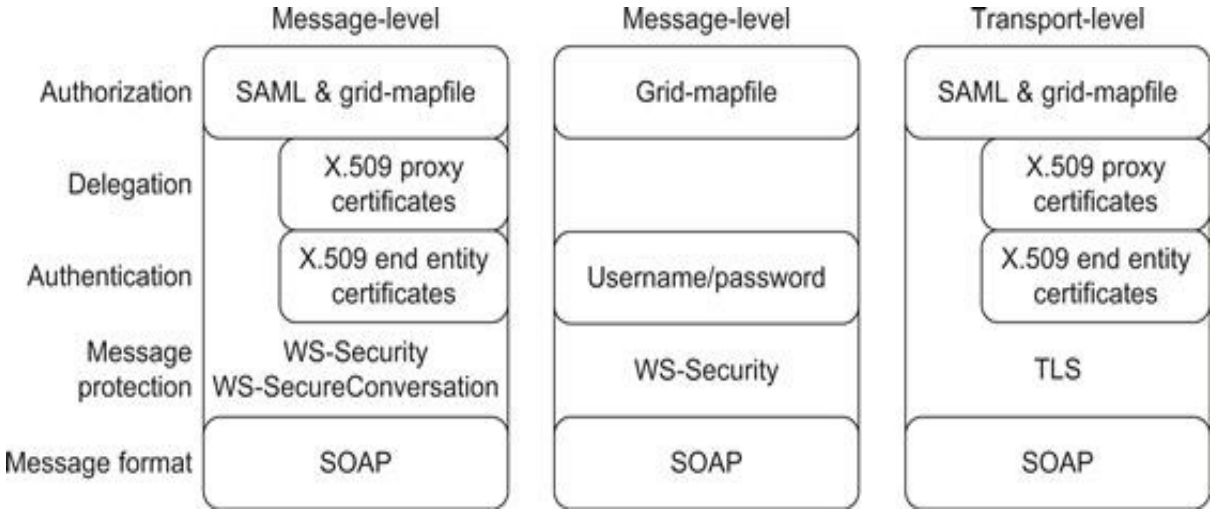
The GSI is an important step toward satisfying these requirements.

- GSI is well-known security solution in the grid environment,
- GSI is a portion of the Globus Toolkit and provides fundamental security services needed to support grids, including supporting for message protection, authentication and delegation, and authorization.
- GSI enables secure authentication and communication over an open network, and permits mutual authentication across and among distributed sites with single sign-on capability.
- No centrally managed security system is required, and the grid maintains the integrity of its members' local policies.

4. GSI supports both message-level security, which supports the WS-Security standard and the WS-Secure Conversation specification to provide message protection for SOAP messages, and transport-level security, which means authentication via *transport-level security (*TLS)with support for X.509 proxy certificates.

GSI Functional Layers

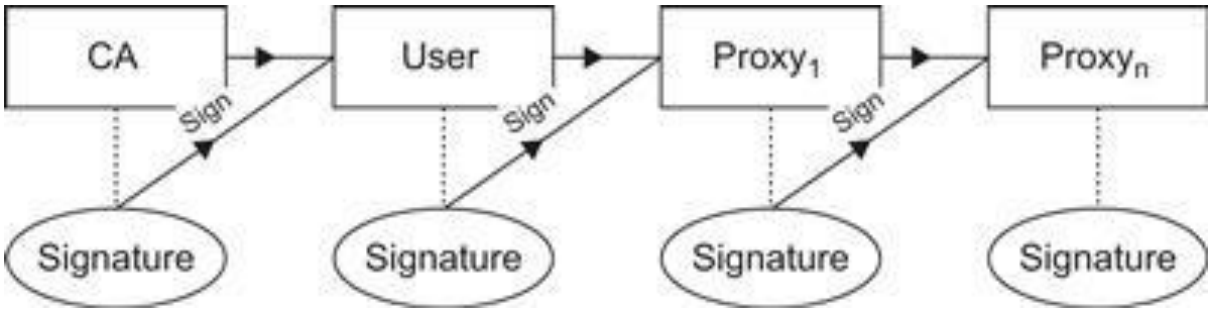
- GT4 provides distinct WS and pre-WS authentication and authorization capabilities.
- Both build on the same base, namely the X.509 standard and entity certificates and proxy certificates, which are used to identify persistent entities such as users and servers and to support the temporary delegation of privileges to other entities, respectively. As shown in Figure.
- GSI may be thought of as being composed of four distinct functions: message protection, authentication, delegation, and authorization.



- TLS (transport-level security) or WS-Security and WS-Secure Conversation (message level) are used as message protection mechanisms in combination with SOAP.
- X.509 End Entity Certificates or Username and Password are used as authentication credentials. X.509 Proxy Certificates and WS-Trust are used for delegation.
- An Authorization Framework allows for a variety of authorization schemes, including a "grid-mapfile" ACL, an ACL defined by a service, a custom authorization handler, and access to an authorization service via the SAML protocol.
- In addition, associated security tools provide for the storage of X.509 credentials, the mapping between GSI and other authentication mechanisms and maintenance of information used for authorization.
- The web services portions of GT4 use SOAP as their message protocol for communication.
- Message protection can be provided either by
    1. transport-level security, which transports SOAP messages over TLS, or by
    2. message-level security, which is signing and/or encrypting Portions of the SOAP message using the WS-Security standard.
- The X.509 certificates used by GSI are conformant to the relevant standards and conventions.
- Grid deployments around the world have established their own CAs based on third-party software to issue the X.509 certificate for use with GSI and the Globus Toolkit.
- GSI also supports delegation and single sign-on through the use of standard .X.509 proxy certificate. Proxy certificate allow bearers of X.509 to delegate their privileges temporarily to another entity.

- For the purposes of authentication and authorization, GSI treats certificate and proxy certificate equivalently. Authentication with X.509 credentials can be accomplished either via TLS, in the case of transport-level security, or via signature as specified by WS-Security, in the case of message-level security.

Authentication and Delegation



- To reduce or even avoid the number of times the user must enter his passphrase when several grids are used or have agents (local or remote) requesting services on behalf of a user, GSI provides a delegation capability and a delegation service that provides an interface to allow clients to delegate (and renew) X.509 proxy certificates to a service.

- The interface to this service is based on the WS-Trust specification. A proxy consists of a new certificate and a private key. The key pair that is used for the proxy, that is, the public key embedded in the certificate and the private key, may either be regenerated for each proxy or be obtained by other means.

- The new certificate contains the owner's identity, modified slightly to indicate that it is a proxy. The new certificate is signed by the owner, rather than a CA

Trust Delegation

- GSI has traditionally supported authentication and delegation through the use of X.509 certificate and public keys. As a new feature in GT4, GSI also supports authentication through plain usernames and passwords as a deployment option.

- As a central concept in GSI authentication, a certificate includes four primary pieces of information:

  (1) a subject name, which identifies the person or object that the certificate represents;

  (2) the public key belonging to the subject;

  (3) the identity of a CA that has signed the certificate to certify that the public key and the identity both belong to the subject; and

(4) the digital signature of the named CA. X.509 provides each entity with a unique identifier (i.e., a distinguished name) and a method to assert that identifier to another party through the use of an asymmetric key pair bound to the identifier by the certificate.

Risks and Security Concerns With Cloud Computing

- Many of the cloud computing associated risks are not new and can be found in the computing environments. There are many companies and organizations that outsource significant parts of their business due to the globalization.
- It means not only using the services and technology of the cloud provider, but many questions dealing with the way the provider runs his security policy.
- After performing an analysis the top threats to cloud computing can be summarized as follows

1. Abuse and Unallowed Use of Cloud Computing;
2. Insecure Application Programming Interfaces;
3. Malicious Insiders;
4. Shared Technology Vulnerabilities;
5. Data Loss and Leakage
6. Account, Service and Traffic Hijacking;
7. Unknown Risk Profile.

5.4 Cloud Security Principles

- Public cloud computing requires a security model that coordinates scalability and multi-tenancy with the requirement for trust. As enterprises move their computing environments with their identities, information and infrastructure to the cloud, they must be willing to give up some level of control.
- In order to do so they must be able to trust cloud systems and providers, as well as to verify cloud processes and events.
- Important building blocks of trust and verification relationships include access control, data security, compliance and event management - all security elements well understood by IT departments today, implemented with existing products and technologies, and extendable into the cloud. The cloud security principles comprise three categories:
- identity,

- information and
- infrastructure.

Identity security

- End-to-end identity management, third-party authentication services and identity must become a key element of cloud security. Identity security keeps the integrity and confidentiality of data and applications while making access readily available to appropriate users. Support for these identity management capabilities for both users and infrastructure components will be a major requirement for cloud computing and identity will have to be managed in ways that build trust. It will require:
- *Stronger authentication:* Cloud computing must move beyond authentication of username and password, which means adopting methods and technologies that are IT standard IT such as strong authentication, coordination within and between enterprises, and risk-based authentication, measuring behavior history, current context and other factors to assess the risk level of a user request.
- *Stronger authorization:* Authorization can be stronger within an enterprise or a private cloud, but in order to handle sensitive data and compliance requirements, public clouds will need stronger authorization capabilities that can be constant throughout the lifecycle of the cloud infrastructure and the data.

Information security

- In the traditional data center, controls on physical access, access to hardware and software and identity controls all combine to protect the data. In the cloud, that protective barrier that secures infrastructure is diffused. The data needs its own security and will require
- *Data isolation:* In multi-tenancy environment data must be held securely in order to protect it when multiple customers use shared resources. Virtualization, encryption and access control will be workhorses for enabling varying degrees of separation between corporations, communities of interest and users.
- *Stronger data security:* In existing data center environments the role-based access control at the level of user groups is acceptable in most cases since the information remains within the control of the enterprise. However, sensitive data will require security at the file, field or block level to meet the demands of assurance and compliance for information in the cloud.
- *Effective data classification:* Enterprises will need to know what type of data is important and where it is located as prerequisites to making performance cost-benefit decisions, as well as ensuring focus on the most critical areas for data loss prevention procedures.

- *Information rights management:* it is often treated as a component of identity on which users have access to. The stronger data-centric security requires policies and control mechanisms on the storage and use of information to be associated directly with the information itself.

- *Governance and compliance:* A major requirement of corporate information governance and compliance is the creation of management and validation information - monitoring and auditing the security state of the information with logging capabilities. The cloud computing infrastructures must be able to verify that data is being managed per the applicable local and international regulations with appropriate controls, log collection and reporting.

5.5 Cloud Infrastructure Security

- IaaS application providers treat the applications within the customer virtual instance as a black box and therefore are completely indifferent to the operations and management of a applications of the customer. The entire pack (customer application and run time application) is run on the customers' server on provider infrastructure and is managed by customers themselves. For this reason it is important to note that the customer must take full responsibility for securing their cloud deployed applications.

- Cloud deployed applications must be designed for the internet threat model.

- They must be designed with standard security counter measures to guard against the common web vulnerabilities.

- Customers are responsible for keeping their applications up to date - and must therefore ensure they have a patch strategy to ensure their applications are screened from malware and hackers scanning for vulnerabilities to gain unauthorized access to their data within the cloud.

- Customers should not be tempted to use custom implementations of Authentication, Authorization and Accounting as these can become weak if not properly implemented.

- The foundational infrastructure for a cloud must be inherently secure whether it is a private or public cloud or whether the service is SAAS, PAAS or IAAS.

- *Inherent component-level security:* The cloud needs to be architected to be secure, built with inherently secure components, deployed and provisioned securely with strong interfaces to other components and supported securely, with vulnerability-assessment and change-management processes that pro- duce management information and service-level assurances that build trust.

- *Stronger interface security:* The points in the system where interaction takes place (user-to-network, server-to application) require stronger security policies and controls that ensure consistency and accountability.

- *Resource lifecycle management:* The economics of cloud computing are based on multi-tenancy and the sharing of resources. As the needs of the customers and requirements will change, a service provider must provision and decommission correspondingly those resources - bandwidth, servers, storage and security. This lifecycle process must be managed in order to build trust.

- The infrastructure security can be viewed, assessed and implemented according its building levels - the network, host and application levels

5.6 Infrastructure Security - The Network Level

- When looking at the network level of infrastructure security, it is important to distinguish between public clouds and private clouds. important to distinguish between public clouds and private clouds. With private clouds, there are no new attacks, vulnerabilities, or changes in risk specific to this topology that information security personnel need to consider.

- If public cloud services are chosen, changing security requirements will require changes to the network topology and the manner in which the existing network topology interacts with the cloud provider's network topology should be taken into account.

There are four significant risk factors in this use case:

1. ***Ensuring the confidentiality and integrity of organization's data-in-transit to and from a public cloud provider;***
2. ***Ensuring proper access control (authentication, authorization, and auditing) to whatever resources are used at the public cloud provider;***
3. ***Ensuring the availability of the Internet-facing resources in a public cloud that are being used by an organization, or have been assigned to an organization by public cloud providers;***
4. ***Replacing the established model of network zones and tiers with domains.***

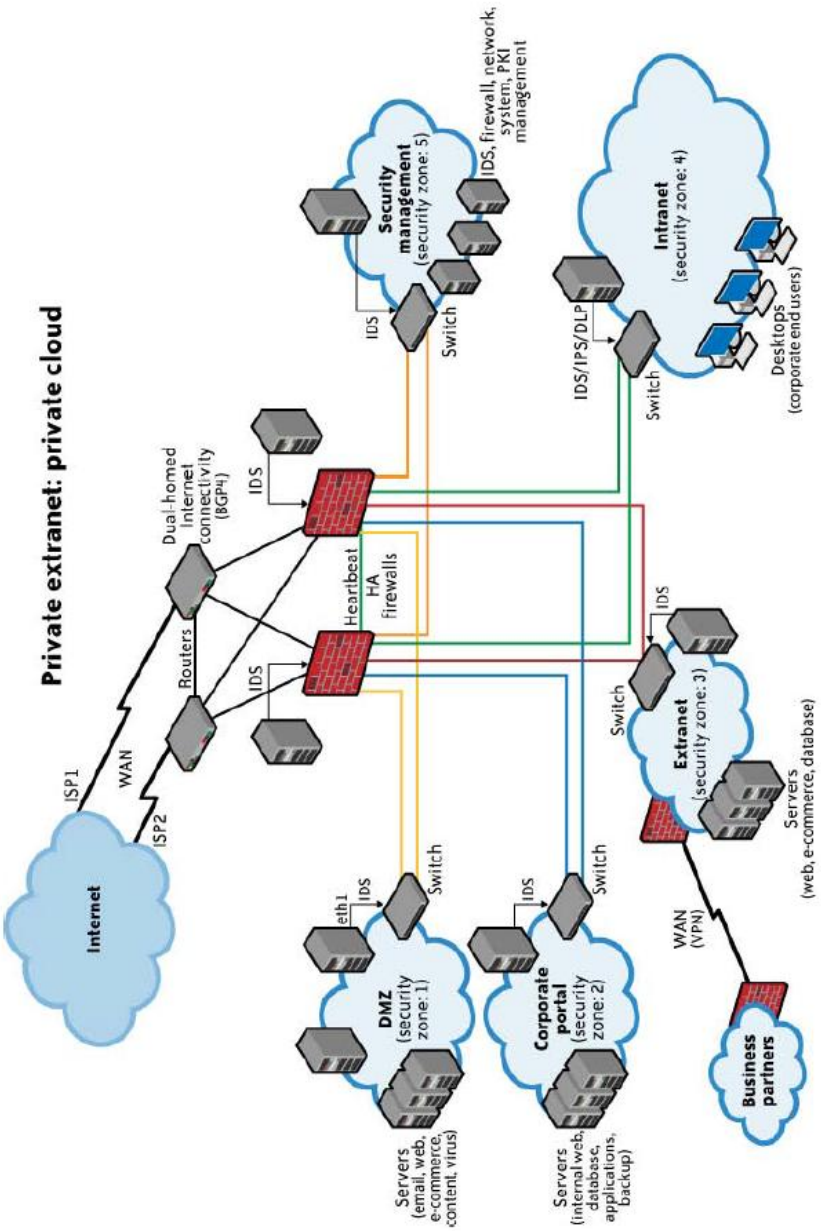
Ensuring Data Confidentiality and Integrity

- Some resources and data previously confined to a private network are now exposed to the Internet, and to a shared public network belonging to a third-party cloud provider.
  Eg:
- An example of problems associated with this first risk factor is an Amazon Web Services (AWS) security vulnerability reported in December 2008

Ensuring Proper Access Control

- Since some subset of these resources (or maybe even all of them) is now exposed to the Internet, an organization using a public cloud faces a significant increase in risk to its data.
- A decreased access to relevant network-level logs and data, and a limited ability to thoroughly conduct investigations and gather forensic data.
- An example of the problems associated with this second risk factor is the issue of reused (reassigned) IP addresses.

- Addresses are usually reassigned and reused by other customers as they become available.
- From a cloud provider's perspective this makes sense.
- IP addresses are a finite quantity and a billable asset.
- There is necessarily a lag time between the change of an IP address in DNS and the clearing of that address in DNS caches.
- There is a similar lag time between when physical (i.e., MAC) addresses are changed in ARP tables and when old ARP addresses are cleared from cache; an old address persists in ARP caches until they are cleared.
- This means that even though addresses might have been changed, the (now) old addresses are still available in cache, and therefore they still allow users to reach these supposedly non-existent resources.
- Ex: AWS announcement of the Amazon Elastic IP capabilities in March 2008



Ensuring the Availability of Internet-Facing Resources

- BGP prefix hijacking i.e., the falsification of Network Layer Reachability Information
- It involves announce of autonomous system‡ address space that belongs to someone else without her permission.
- Such announcements often occur because of a configuration mistake, but that misconfiguration may still affect the availability of your cloud-based resources.
- Eg: misconfiguration mistake occurred in February 2008 when Pakistan Telecom made an error by announcing a dummy route for YouTube to its own telecommunications partner, PCCW, based in Hong Kong. The intent was to block YouTube within Pakistan

because of some supposedly blasphemous videos hosted on the site. The result was that YouTube was globally unavailable for two hours.

- Although prefix hijackings are not new, that attack figure will certainly rise, and probably significantly, along with a rise in cloud computing.
- As the use of cloud computing increases, the availability of cloud-based resources increases in value to customers.
- That increased value to customers translates to an increased risk of malicious activity to threaten that availability.

DNS attack:

- Not only are there vulnerabilities in the DNS protocol and in implementations of DNS.
- It is there are fairly widespread DNS cache poisoning attacks whereby a DNS server is tricked into accepting incorrect information.

Types:

- Variants of this basic cache poisoning attack include
- redirecting the target domain's name server (NS),
- redirecting the NS record to another target domain, and
- responding before the real NS (called DNS forgery).

Denial of service (DoS) and distributed denial of service (DDoS) attacks

- Increase the risk of network.
- There continue to be rumors of continued DDoS attacks on AWS, making the services unavailable for hours at a time to AWS users.
- When using IaaS, the risk of a DDoS attack is not only external (i.e., Internet-facing).
- There is also the risk of an internal DDoS attack through the portion of the IaaS provider's network used by customers.

Replacing the Established Model of Network Zones and Tiers with Domains

- The established isolation model of network zones and tiers no longer exists in the public IaaS andPaaS clouds.
- The traditional model of network zones and tiers has been replaced in public cloud computing with "security groups," "security domains," or "virtual data centers" that have logical separation between tiers but are less precise and afford less protection than the formerly established model.
- For example, the security groups feature in AWS allows virtual machines (VMs) to access each other using a virtual firewall that has the ability to filter traffic based on IP address (a specific address or a subnet), packet types (TCP, UDP, or ICMP), and ports (or a range of ports). Domain names are used in various networking contexts and application-specific naming and addressing purposes, based on DNS.
- For example, Google's App Engine provides a logical grouping of applications based on domain names such as mytestapp.test.mydomain.com and myprodapp.prod.mydomain.com.

Network-Level Mitigation

- To reduce your confidentiality risks by using encryption; specifically by using validated implementations of cryptography for data-in-transit.

- Secure digital signatures make it much more difficult, if not impossible, for someone to tamper with your data, and this ensures data integrity.

Threat outlook	Low (with the exception of DoS attacks)
Preventive controls	Network access control supplied by provider (e.g., firewall), encryption of data in transit (e.g., SSL, IPSec)
Detective controls	Provider-managed aggregation of security event logs (security incident and event management, or SIEM), network-based intrusion detection system/intrusion prevention system (IDS/IPS)

5.7 Infrastructure Security - The Host Level

- When reviewing host security and assessing risks, the context of cloud services delivery models (SaaS, PaaS, and IaaS) and deployment models public, private, and hybrid) should be considered.
- The host security responsibilities in SaaS and PaaS services are transferred to the provider of cloud services.
- IaaS customers are primarily responsible for securing the hosts provisioned in the cloud (virtualization software security, customer guest OS or virtual server security).

SaaS and PaaS Host Security

- In general, CSPs do not publicly share information related to their host platforms, host operating systems, and the processes that are in place to secure the hosts, since hackers can exploit that information when they are trying to intrude into the cloud service.
- NDA:
- To get assurance from the CSP on the security hygiene of its hosts, you should ask the vendor to share information under a nondisclosure agreement (NDA) or simply demand that the CSP share the information via a controls assessment framework such as SysTrust or ISO 27002.
- From a controls assurance perspective, the CSP has to ensure that appropriate preventive and detective controls are in place and will have to ensure the same via a third-party assessment or ISO 27002 type assessment framework.
- Since virtualization is a key enabling technology that improves host hardware utilization, among other benefits, it is common for CSPs to employ virtualization platforms, including Xen and VMware hypervisors, in their host computing platform architecture.

Abstraction:

- Both the PaaS and SaaS platforms abstract and hide the host operating system from end users with a host abstraction layer.
- One key difference between PaaS and SaaS is the accessibility of the abstraction layer that hides the operating system services the applications consume.

- In the case of SaaS, the abstraction layer is not visible to users and is available only to the developers and the CSP's operations staff, where PaaS users are given indirect access to the host abstraction layer in the form of a PaaS application programming interface (API) that in turn interacts with the host abstraction layer.

IaaS Host Security

- Unlike PaaS and SaaS, IaaS customers are primarily responsible for securing the hosts
- provisioned in the cloud.

Virtualization software security

- The software layer that sits on top of bare metal and provides customers the ability to create and destroy virtual instances.
- Virtualization at the host level can be accomplished using any of the virtualization models, including OS-level virtualization, para virtualization, or hardware-based virtualization.
- It is important to secure this layer of software that sits between the hardware and the virtual servers. In a public IaaS service, customers do not have access to this software layer; it is managed by the CSP only.
  Eg
- A recent incident at a tiny UK-based company called Vaserv.comexemplifies the threat to hypervisor security. By exploiting a zero-day vulnerability inHyperVM, a virtualization application made by a company called Lxlabs, hackers destroyed100,000 websites hosted by Vaserv.com. The zero-day vulnerability gave the attackers theability to execute sensitive Unix commands on the system, including rm -rf, which forces arecursive delete of all files. Evidently, just days before the intrusion, an anonymous user postedon a hacker website called milw0rm a long list of yet-unpatched vulnerabilities in Kloxo, ahosting control panel that integrates into HyperVM. The situation was worse for approximately50% of Vaserv's customers who signed up for unmanaged service, which doesn't include databackup.

Customer guest OS or virtual server security

- The virtual instance of an operating system that is provisioned on top of the virtualization layer and is visible to customers from the Internet; e.g., various flavors of Linux, Microsoft, and Solaris. Customers have full access to virtual servers.

Threats to the hypervisor

- A vulnerable hypervisor could expose all user domains to malicious insiders.
- Furthermore, hypervisors are potentially susceptible to subversion attacks.
- To illustrate the vulnerability of the virtualization layer, some members of the security research community demonstrated a "Blue Pill" attack on a hypervisor.

Virtual Server Security

- Customers of IaaS have full access to the virtualized guest VMs that are hosted and isolated from each other by hypervisor technology.
- Hence customers are responsible for securing andongoing security management of the guest VM.

- A public IaaS, such as Amazon's Elastic Compute Cloud (EC2), offers a web services API to perform management functions such as provisioning, decommissioning, and replication of virtual servers on the IaaS platform.
- The CSP blocks all port access to virtual servers and recommends that customers use port 22 (Secure Shell or SSH) to administer virtual server instances.
- Thecloud management API adds another layer of attack surface and must be included in the scopeof securing virtual servers in the public cloud.
- Some of the new host security threats in the public IaaS include:

• Stealing keys used to access and manage hosts
• Attacking unpatched, vulnerable services listening on standard ports
• Hijacking accounts that are not properly secured
• Attacking systems that are not properly secured by host firewalls
• Deploying Trojans embedded in the software component in the VM or within the VM

Securing virtual servers

- Use a secure-by-default configuration
- Track the inventory of VM images and OS versions that are prepared for cloud hosting
- Protect the integrity of the hardened image from unauthorized access.
- Safeguard the private keys required to access hosts in the public cloud.
- isolate the decryption keys from the cloud where the data is hosted
- Include no authentication credentials in your virtualized images
- Do not allow password-based authentication for shell access
- Require passwords for sudo* or role-based access
- Run a host firewall and open only the minimum ports necessary to support the services on an instance.
- Run only the required services and turn off the unused services
- Install a host-based IDS such as OSSEC or Samhain
- Enable system auditing and event logging, and log the security events to a dedicated log server
- snapshot your block volumes, and back up the root filesystem
- Institute a process for patching the images in the cloud
- Periodically review logs for suspicious activities

5.8 Infrastructure Security - The Application Level

- Application or software security should be a critical element of a security pro- gram. Most enterprises with information security programs have yet to institute an application security program to address this realm.

- Designing and implementing applications aims at deployment on a cloud platform will require existing application security programs to reevaluate current practices and standards.

- The application security spectrum ranges from standalone single-user applications to sophisticated multiuser e-commerce applications used by many users. The level is responsible for managing.

1. Application-level security threats;
2. End user security;
3. SaaS application security;
4. PaaS application security;
5. PaaS application security;
6. Customer-deployed application security
7. IaaS application security
8. Public cloud security limitations

It can be summarized that the issues of infrastructure security and cloud computing lie in the area of definition and provision of security specified aspects each party delivers.

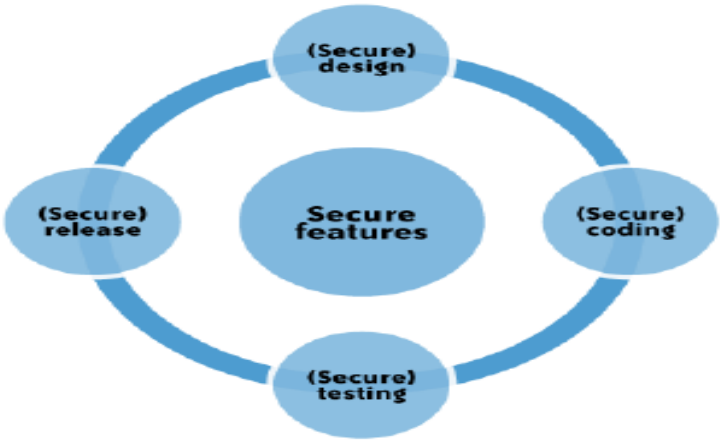
Application levels:

- Single user applications
- Multi user applications (e-commerce applications)
- Web applications such as content management systems (CMSs), wikis, portals, bulletin boards, and discussion forums are used by small and large organizations.
- According to SANS, until 2007 few criminals attacked vulnerable websites because other attack vectors were more likely to lead to an advantage in unauthorized economic or information access.

Application-Level Security Threats

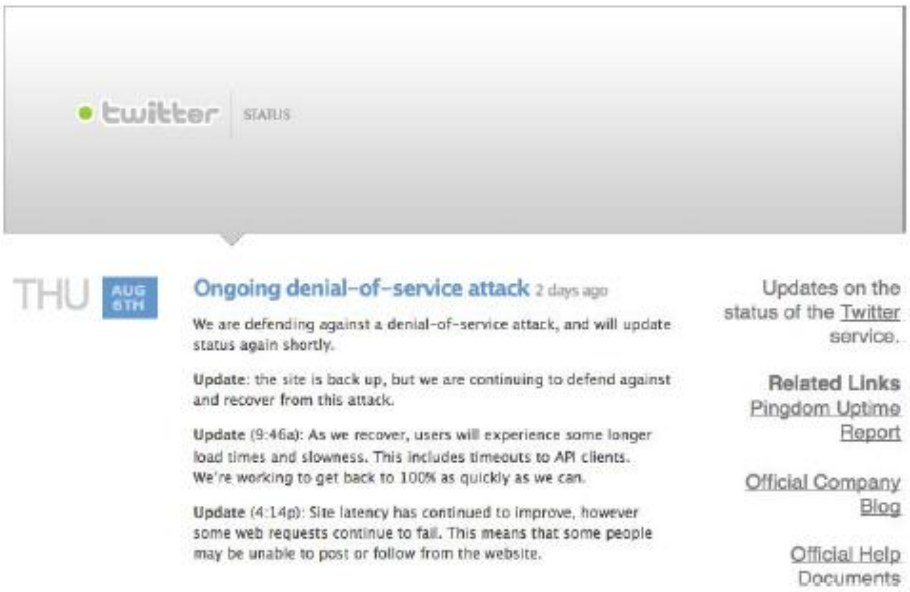
- Application vulnerabilities are cross-site scripting (XSS),SQL injection, malicious file execution, and other vulnerabilities resulting from programming errors and design flaws.
- hackers are constantly scanning web applications - exploiting the vulnerabilities they discover for various illegal activities including financial fraud, intellectual property theft, converting trusted websites into malicious servers serving client-side exploits, and phishing scams.
- All web frameworks and all types of web applications are at risk of web application security defects, ranging from insufficient validation to application logic errors.
- Web applications built and deployed in a public cloud platform will be subjected to a high threat level, attacked, and potentially exploited by hackers to support fraudulent and illegal activities.
- security must be embedded into the Software Development Life Cycle (SDLC).

SDLC



DoS and EDoS

- Additionally, you should be cognizant of application-level DoS and DDoS attacks that can potentially disrupt cloud services.
- Application-level DoS attacks could manifest themselves as high-volume web page reloads, XML* web services requests (over HTTP or HTTPS), or protocol-specific requests supported by a cloud service
- For example, a DDoS attack on Twitter on August 6, 2009, brought the service down for several hours
- Dos effect - poor user experience and service-level Impacts
- DoS attacks can quickly drain your company's cloud services budget.
- DoS attacks on pay-as-you-go cloud applications will result in a dramatic increase in your cloud utility bill increased use of network bandwidth, CPU, and storage consumption. This type of attack is also being characterized as economic denial of sustainability.



DDOS attack on twitter

End User Security

- A customer of a cloud service, are responsible for end user security tasks.
- Protection measures include
- use of security software, such as anti-malware,
- antivirus,
- personal firewalls,
- security patches, and
- IPS-type software on your Internet-connected computer.
- browsers have become the ubiquitous "operating systems" for consuming cloud services.
- All Internet browsers routinely suffer from software vulnerabilities that make them vulnerable to end user security attacks.
- Hence, our recommendation is that cloud customers take appropriate steps to protect browsers from attacks.
- To achieve end-to-end security in a cloud, it is essential for customers to maintain good browser hygiene
- The means keeping the browser (e.g., Internet Explorer, Firefox, Safari) patched and updated to mitigate threats related to browser vulnerabilities.

Who Is Responsible for Web Application Security in the Cloud?

- Depending on the cloud services delivery model (SPI) and service-level agreement (SLA), the scope of security responsibilities will fall on the shoulders of both the customer and the cloud provider.
- cloud customers do not have the transparency required in the area of software vulnerabilities in cloud services.
- Due to this lack of transparency, customers are left with no choice but to trust their CSPs to disclose any new vulnerability that may affect the confidentiality, integrity, or availability of their application.
- For example, as of March 2009, no prominent IaaS, PaaS, or SaaS vendors are participating in the Common Vulnerability and Exposures (CVE) project.
- Case in point: AWS took 7.5 months to fix a vulnerability that Colin Percival reported in May 2007.
- This vulnerability was a cryptographic weakness in Amazon's request signing code that affected its database API (SimpleDB) and EC2 API services, and it was not made public until after it was fixed in December 2008.
- Enterprise customers should understand the vulnerability disclosure policy of cloud services and factor that into the CSP risk assessment.

SaaS Application Security

- The SaaS model dictates that the provider manages the entire suite of applications delivered to users.
- Therefore, SaaS providers are largely responsible for securing the applications and components they offer to customers.

- Customers are usually responsible for operational security functions, including user and access management as supported by the provider.
- It is a common practice for prospective customers, usually under an NDA, to request information related to the provider's security practices.
- Extra attention needs to be paid to the authentication and access control features offered by SaaS CSPs.
- Usually that is the only security control available to manage risk to information.
- Most services, including those from Salesforce.com and Google, offer a web-based administration user interface tool to manage authentication and access control of the application.
- Some SaaS applications, such as Google Apps, have built-in features that end users can invoke to assign read and write privileges to other users.
- One example that captures this issue is the mechanism that Google Docs employs in handling images embedded
- in documents, as well as access privileges to older versions of a document
- Cloud customers should try to understand cloud-specific access control mechanisms—including support for strong authentication and privilege management based on user roles and functions—and take the steps necessary to protect information hosted in the cloud.
- Additional controls should be implemented to manage privileged access to the SaaS administration tool, and enforce segregation of duties to protect the application from insider threats.

Security controls at the application level

Threat outlook	Medium
Preventive controls	Identity management, access control assessment, browser hardened with latest patches, multifactor authentication via delegated authentication, endpoint security measures including antivirus and IPS
Detective controls	Login history and available reports from SaaS vendors

PaaS Application Security

- PaaS vendors broadly fall into the following two major categories:
    - Software vendors (e.g., Bungee, Etelos, GigaSpaces, Eucalyptus)
    - CSPs (e.g., Google App Engine, Salesforce.com's Force.com, Microsoft Azure, Intuit QuickBase)
- PaaS software to build a solution for internal consumption.
- Currently, no major public clouds are known to be using commercial off-the-shelf or open source PaaS software such as Eucalyptus.
- By definition, a PaaS cloud (public or private) offers an integrated environment to design,develop, test, deploy, and support custom applications developed in the language the platform supports.
- PaaS application security encompasses two software layers:

- Security of the PaaS platform itself (i.e., runtime engine)
- Security of customer applications deployed on a PaaS platform

PaaS application container

- CSPs are responsible for monitoring new bugs and vulnerabilities that may be used to exploit the PaaS platform and break out of the sandbox architecture. This type of situation is the worst case scenario for a PaaS service; the privacy implications for customer-sensitive information are undesirable and could be very damaging to your business. Hence, enterprise customers should seek information from the CSP on the containment and isolation architecture of the PaaS service

Customer-Deployed Application Security

- PaaS developers need to get familiar with specific APIs to deploy and manage software modules that enforce security controls.
- API is unique to a PaaS cloud service, developers are required to become familiar with platform-specific security features.
- Currently, the Google App Engine supports only Python and Java, and Salesforce.com's Force.com supports only a proprietary language called Apex.
- Developers should expect CSPs to offer a set of security features, including user authentication, single sign-on (SSO) using federation, authorization (privilege management), and SSL or TLS support, made available via the API.
- Currently, there is no PaaS security management standard: CSPs have unique security models, and security features will vary from provider to provider.
- In the case of the Google App Engine, a developer using Python or Java objects can configure the user profile and select HTTPS as a transport protocol.

Security controls

Threat outlook	**Medium**
Preventive controls	User authentication, account management, browser hardened with latest patches, endpoint security measures including antivirus and IPS
Detective controls	Application vulnerability scanning

IaaS Application Security

- Web applications deployed in a public cloud must be designed for an Internet threat model, embedded with standard security countermeasures against common web vulnerabilities (e.g., the OWASP Top 10).
- In adherence with common security development practices, they should also be periodically tested for vulnerabilities, and most importantly, security should be embedded into the SDLC.

- Customers are solely responsible for keeping their applications and runtime platform patched to protect the system from malware and hackers scanning for vulnerabilities to gain unauthorized access to their data in the cloud.
- In summary, the architecture for IaaS hosted applications closely resembles enterprise web applications with an n-tier distributed architecture. In an enterprise, distributed applications run with many controls in place to secure the host and the network connecting the distributed hosts.
- Comparable controls do not exist by default in an IaaS platform and must be added through a network, user access, or as application-level controls.

Security controls to – IaaS

Threat outlook	**High**
Preventive controls	Application developed using security-embedded SDLC process, least-privileged configuration, timely patching of application, user authentication, access control, account management, browser hardened with latest patches, endpoint security measures including antivirus, IPS, host-based IDS, host firewall, and virtual private network (VPN) for administration
Detective controls	Logging, event correlation, application vulnerability scanning and monitoring

5.9 Aspects of Data Security

Security for

1. **Data in transit**
2. **Data at rest**
3. **Processing of data including multitenancy**
4. **Data Lineage**
5. **Data Provenance**
6. **Data remanance**

Data-in-transit

- Primary risk involved is in not using a vetted encryption algorithm
- Encryption should be employed regardless of whether it is an IaaS, SaaS or PaaS
- FTPS over SSL, HTTPS, SCP- preferred

- Protocol should provide confidentiality and integrity
- Merely by using plain FTP, HTTP confidentiality can be provided by not integrity

Data-at –rest

- For IaaS service, encrypting data-at-rest is possible
- For PaaS and SaaS, encrypting data-at-rest as a compensating control is not always feasible
- Data-at-rest used by a cloud application is generally not encrypted
- Indexing or searching of data is not possible when encrypted
- PaaS and SaaS based architectures use multi tenancy
- Commingling of data with other user's data happens in cloud
- Encryption during processing of data in the cloud is not feasible
- No known methods are available to process encrypted data
- For any application to process data, data must be unencrypted
- Fully homomorphic encryption scheme allows data to be processed without being decrypted
- Predicate encryption- limit the amount of data that would need to be decrypted before processing in the cloud

Data Lineage

- Following the path of data
- Mapping of application data path or data flow visualization
- Important for auditors assurance
- External, internal and regulatory

Data Lineage

Data lineage is defined as a data life cycle that includes the data's origins and where it moves over time. It describes what happens to data as it goes through diverse processes.

Following the path of data (mapping application data flows or data path visualization) is known as *data lineage,* and it is important for an auditor's assurance (internal, external, and regulatory).

Data Provenance

- *Integrity of data* refers to data that has not been changed in
- an unauthorized manner or by an unauthorized person. *Provenance* means not only that the
- data has integrity, but also that it is computationally accurate;

Data Remanance

- A final aspect of data security is *data remanence*. "Data remanence is the residual representation
- of data that has been in some way nominally erased or removed. This residue may be due to
- data being left intact by a nominal delete operation, or through physical properties of the
- storage medium. Data remanence may make inadvertent disclosure of sensitive information
- possible, should the storage media be released into an uncontrolled environment (e.g., thrown
- in the trash, or given to a third party)."

Clearing and Sanitization"

Instructions on clearing, sanitization, and release of information systems (IS) media shall be issued by the accrediting Cognizant Security Agency (CSA).

"a. Clearing"

Clearing is the process of eradicating the data on media before reusing the media in an environment that provides an acceptable level of protection for the data that was on the media before clearing. All internal memory, buffer, or other reusable memory shall be cleared to effectively deny access to previously stored information.

"b. Sanitization"

Sanitization is the process of removing the data from media before reusing the media in an environment that does not provide an acceptable level of protection for the data that was on the media before sanitizing. IS resources shall be sanitized before they are released from classified information controls or released for use at a lower classification level.

- Solutions include encryption, identity management, sanitation

1. **Write about provider data and its security in cloud.**

- In addition to the security of your own customer data, customers should also be concerned about what data the provider collects and how the CSP protects that data.

- Customer data : metadata does the provider have about your data, how is it secured, and what access do you, the customer, have to that metadata.

- A Provider collects and must protect a huge amount of security-related data.
- For example, at the network level, your provider should be collecting, monitoring, and protecting firewall, intrusion prevention system (IPS), security incident and event management (SIEM), and router flow data.

Storage

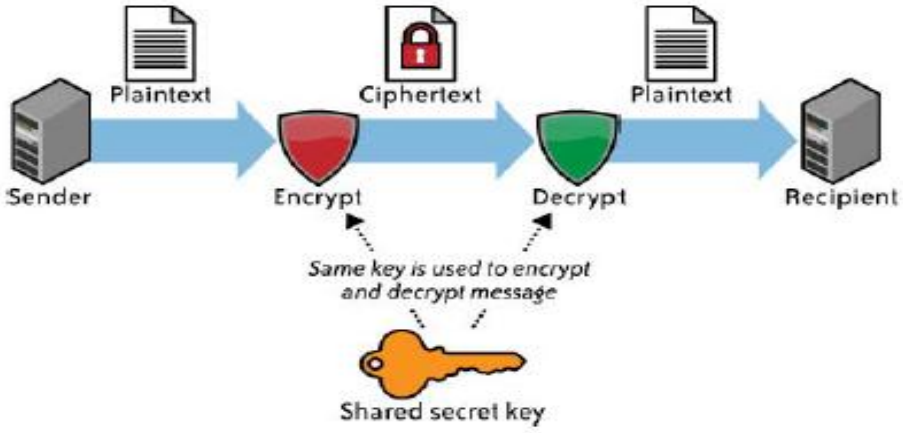
- Three information security concerns are associated with this data stored in the cloud (e.g., Amazon's S3) as with data stored elsewhere:
- confidentiality,

- integrity, and
- availability.

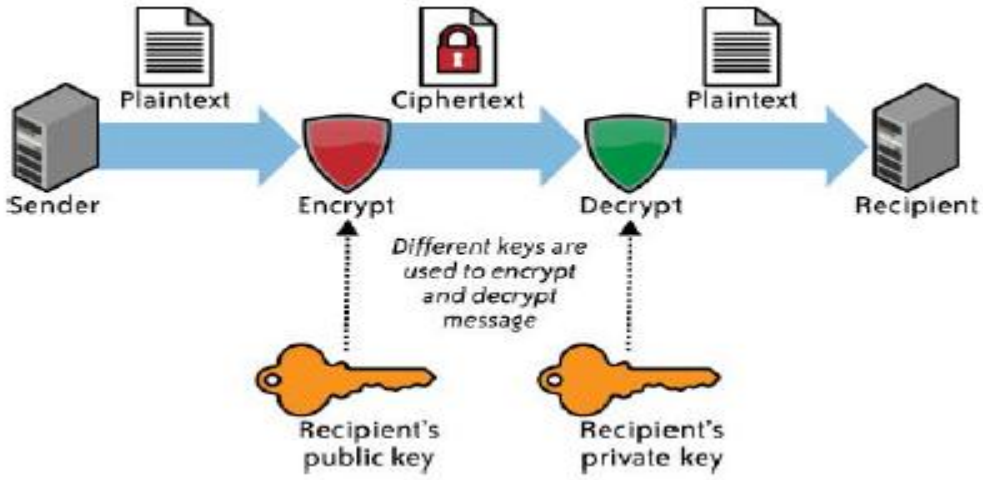
Confidentiality

- CSPs generally use weak authentication mechanisms (e.g., username + password), and the authorization.
- The only authorization levels cloud vendors provide are administrator authorization (i.e., the owner of the account itself) and user authorization (i.e., all other authorized users)—with no levels in between.
- The data and encryption depends in the service provider.
- Customers are able to encrypt their own data themselves prior to uploading, but S3 does not provide encryption.
- If a CSP does encrypt a customer's data, the next consideration concerns what encryption algorithm it uses. Not all encryption algorithms are created equal.
- Symmetric encryption involves the use of a single secret key for both the encryption and decryption of data.

Symmetric encryption



Asymmetric encryption



Key length:

- With symmetric encryption, the longer the key length (i.e., the greater number of bits in the key), the stronger the encryption.
- Although long key lengths provide more protection, they are also more computationally intensive, and may strain the capabilities of computer processors.
- Key lengths should be a minimum of 112 bits for Triple DES,
- 128-bits for AES.
- At a minimum, a customer should consult all three parts of NIST's 800-57,
- "Recommendation for Key Management":

• "Part 1: General"
• "Part 2: Best Practices for Key Management Organization"
• "Part 3: Application-Specific Key Management Guidance (Draft)"

Integrity

- In addition to the confidentiality of the data, it is also need to worry about the integrity of the data.
- Confidentiality does not imply integrity;
- Encryption alone is sufficient for confidentiality, but integrity also requires the use of message authentication codes (MACs).
- The simplest way to use MACs on encrypted data is to use a block symmetric algorithm (as opposed to a streaming symmetric algorithm) in cipher block chaining (CBC) mode, and to include a one-way hash function
- Another aspect of data integrity is important, especially with bulk storage using IaaS.
- Once a customer has several gigabytes (or more) of its data up in the cloud for storage, how does the customer check on the integrity of the data stored there.

Availability

- Assuming that a customer's data has maintained its confidentiality and integrity, you must also be concerned about the availability of your data.
- There are currently three major threats in this regard.
- The first threat to availability is network-based attacks
- The second threat to availability is the CSP's own availability
- A number of high-profile cloud provider outages have occurred.
- For example, Amazon's S3 suffered a 2.5-hour outage in February 2008 and an eight-hour outage in July 2008. AWS is one of the more mature cloud providers, so imagine the difficulties that other, smaller or less mature cloud providers are having.
- Finally, prospective cloud storage customers must be certain to ascertain just what services their provider is actually offering.
- Cloud storage does not mean the stored data is actually backed up. Some cloud storage providers do back up customer data, in addition to providing storage.
- However, many cloud storage providers do not back up customer data, or do so only as an additional service for an additional cost.

- For example, “data stored in Amazon S3, Amazon SimpleDB, or Amazon Elastic Block Store is redundantly stored in multiple physical locations as a normal part of those services and at no additional charge.”
- However, “data that is maintained within running instances on Amazon EC2, or within Amazon S3 and AmazonSimpleDB, is all customer data and therefore AWS does not perform backups.”
- For availability, this is a seemingly simple yet critical question that customers should be asking of cloud storage providers.

Identity and Access Management (IAM) in the Cloud

Trust Boundaries and IAM

- In a traditional environment, trust boundary is within the control of the organization
- This includes the governance of the networks, servers, services, and applications
- In a cloud environment, the trust boundary is dynamic and moves within the control of the service provider as well ass organizations
- Identity federation is an emerging industry best practice for dealing with dynamic and loosely coupled trust relationships in the collaboration model of an organization
- Core of the architecture is the directory service which is the repository for the identity, credentials and user attributes

Why IAM?

- Improves operational efficiency and regulatory compliance management
- IAM enables organizations to achieve access cont6rol and operational security
- Cloud use cases that need IAM
- Organization employees accessing SaaS se4rvidce using identity federation
- IT admin access CSP management console to provision resources and access foe users using a corporate identity
- Developers creating accounts for partner users in PaaS
- End uses access storage service in a cloud
- Applications residing in a cloud serviced provider access storage from another cloud service

IAM Challenges

Provisioning resources to users rapidly to accommodate their changing roles

Handle turnover in an organization

Disparate dictionaries, identities, access rights

Need standards and protocols that address the IAM challenges

IAM Definitions

- **Authentication - Verifying the identity of a user, system or service**
- **Authorization - Privileges that a user or system or service has after being authenticated (e.g., access control)**
- **Auditing - Exam what the user, system or service has carried out**

Check for compliance

Explain in detail about IAM and its architecture.

- The organizations invest in IAM practices to improve operational efficiency and to comply with regulatory, privacy, and data protection requirements.

- Improve operational efficiency
- Properly architected IAM technology and processes can improve efficiency by automating user on-boarding and other repetitive tasks

- Regulatory compliance management
- IAM processes and practices can help organizations meet objectives in the area of access control and operational security

IAM Definitions
Authentication
Authentication is the process of verifying the identity of a user or system (e.g., Lightweight Directory Access Protocol [LDAP] verifying the credentials presented by the user, where the identifier is the corporate user ID that is unique and assigned to an employee

Authorization
Authorization is the process of determining the privileges the user or system is entitled to once the identity is established. In the context of digital services, authorization usually follows the authentication step and is used to determine whether the user or service has the necessary privileges to perform certain operations—in other words, authorization is the process of enforcing policies.

Auditing
In the context of IAM, auditing entails the process of review and examination of authentication, authorization records, and activities to determine the adequacy of IAM system controls, to verify compliance with established security policies and procedures

(e.g., separation of duties), to detect breaches in security services (e.g., privilege escalation), and to recommend any changes that are indicated for countermeasures.

IAM Architecture

- IAM is not a monolithic solution that can be easily deployed to gain capabilities immediately.
- It is as much an aspect of architecture (see Figure) as it is a collection of technology components, processes, and standard practices. Standard enterprise IAM architecture encompasses several layers of technology, services, and processes.
- The directory interacts with IAM technology components such as authentication, user management, provisioning, and federation services that support the standard IAM practice and processes within the organization.
- The IAM processes to support the business can be broadly categorized as follows:
- User management
- Activities for the effective governance and management of identity life cycles

Authentication management

- *Activities for the effective governance and management of the process for determining that an entity is who or what it claims to be*

Authorization management

- *Activities for the effective governance and management of the process for determining entitlement rights that decide what resources an entity is permitted to access in accordance with the organization's policies*

Access management

- *Enforcement of policies for access control in response to a request from an entity (user, services) wanting to access an IT resource within the organization*
- *Data management and provisioning*
- *Propagation of identity and data for authorization to IT resources via automated or manual processes*

Enterprise IAM functional architecture

Monitoring and auditing

Monitoring, auditing, and reporting compliance by users regarding access to resources within the organization based on the defined policies

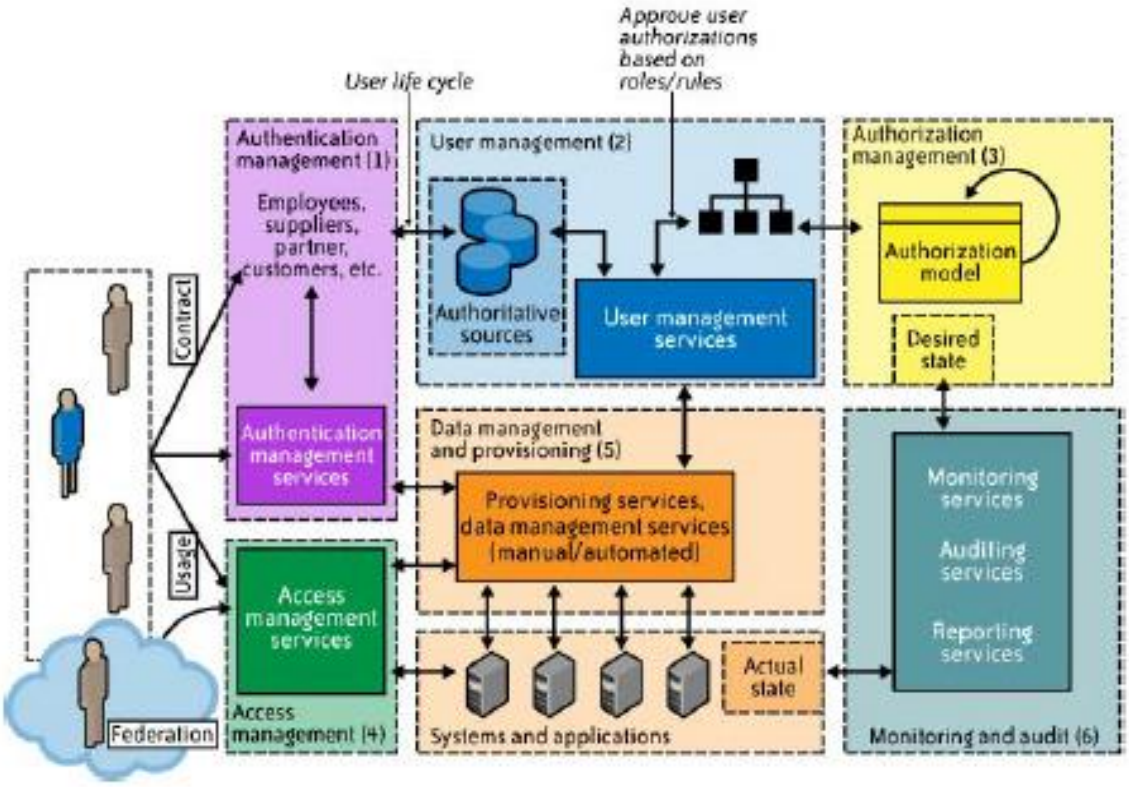
IAM processes support the following operational activities:

Provisioning

This is the process of on-boarding users to systems and applications.

These processes provide users with necessary access to data and technology resources.

Provisioning can be thought of as a combination of the duties of the human resources and IT departments



Credential and attribute management

These processes are designed to manage the life cycle of credentials and user attributes— create, issue, manage, revoke—to minimize the business risk associated with identity impersonation and inappropriate account use. Credentials are usually bound to an individual and are verified during the authentication process

Entitlement management

Entitlements are also referred to as authorization policies.

The processes in this domain address the provisioning and deprovisioning of privileges needed for the user to access resources including systems, applications, and databases.

Entitlement management can be used to strengthen the security of web services, web applications, legacy applications, documents and files, and physical security systems.

Compliance management

This process implies that access rights and privileges are monitored and tracked to ensure the security of an enterprise's resources.

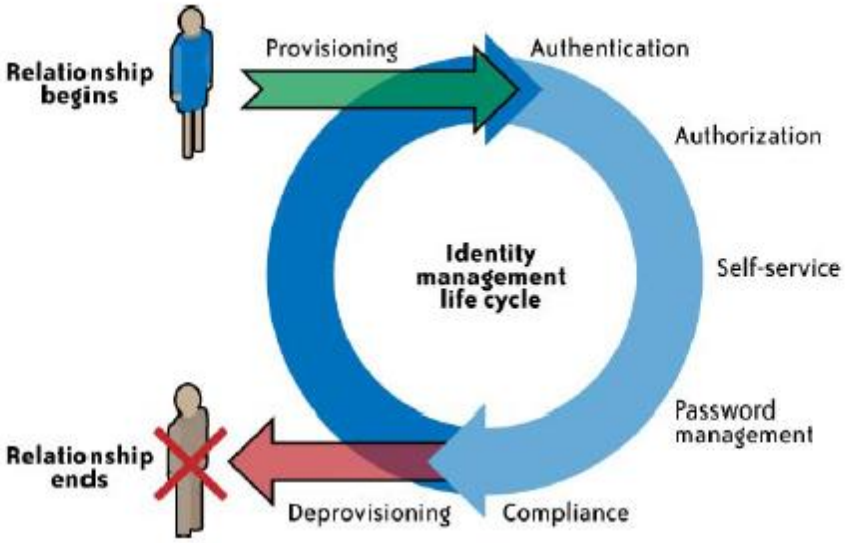
The process also helps auditors verify compliance to various internal access control policies, and standards that include practices such as segregation of duties, access monitoring, periodic auditing, and reporting

Identity federation management

Federation is the process of managing the trust relationships established beyond the internal network boundaries or administrative domain boundaries among distinct organizations.
A federation is an association of organizations that come together to exchange information about their users and resources to enable collaborations and transactions

Centralization of authentication (authN) and authorization (authZ)
A central authentication and authorization infrastructure alleviates the need for application developers to build custom authentication and authorization features into their applications

Identity life cycle



Explain about IAM practices in cloud.

Comparison of SPPM maturity models

Level	SaaS	PaaS	IaaS
User Management, New Users	Capable	Immature	Aware
User Management, User Modifications	Capable	Immature	Immature
Authentication Management	Capable	Aware	Capable
Authorization Management	Aware	Immature	Immature

The maturity model takes into account the dynamic nature of IAM users, systems, and applications in the cloud and addresses the four key components of the IAM automation process:
• User Management, New Users
• User Management, User Modifications
• Authentication Management

• Authorization Management

Although the principles and purported benefits of established enterprise IAM practices and processes are applicable to cloud services, they need to be adjusted to the cloud environment. Broadly speaking, user management functions in the cloud can be categorized as follows:
• Cloud identity administration
• Federation or SSO
• Authorization management
• Compliance management

Cloud Identity Administration

- Cloud identity administrative functions should focus on life cycle management of user
- identities in the cloud—provisioning, deprovisioning, identity federation, SSO, password or credentials management, profile management, and administrative management.
- By federating identities using either an internal Internet-facing IdP or a cloud identity
- management service provider, organizations can avoid duplicating identities and attributes and storing them with the CSP.
- Provisioning users when federation is not supported can be complex and laborious.
- It is not unusual for organizations to employ manual processes, web-based administration, outsourced (delegated) administration that involves uploading of spreadsheets, and execution of custom scripts at both the customer and CSP locations.

Federated Identity (SSO)

Organizations planning to implement identity federation that enables SSO for users can take one of the following two paths (architectures):
• Implement an enterprise IdP within an organization perimeter.
• Integrate with a trusted cloud-based identity management service provider.
Both architectures have pros and cons.

Enterprise identity provider

- In this architecture, cloud services will delegate authentication to an organization's IdP. In this delegated authentication architecture, the organization federates identities within a trusted circle of CSP domains. A circle of trust can be created with all the domains that are authorized to delegate authentication to the IdP. In this deployment architecture, where the organization will provide and support an IdP, greater control can be exercised over user identities, attributes, credentials, and policies for authenticating and authorizing users to a cloud service.

Pros

- Organizations can leverage the existing investment in their IAM infrastructure and extend the practices to the cloud
- They are consistent with internal policies, processes, and access management
- frameworks.
- They have direct oversight of the service-level agreement (SLA) and security of the IdP.
- They have an incremental investment in enhancing the existing identity architecture to support federation.

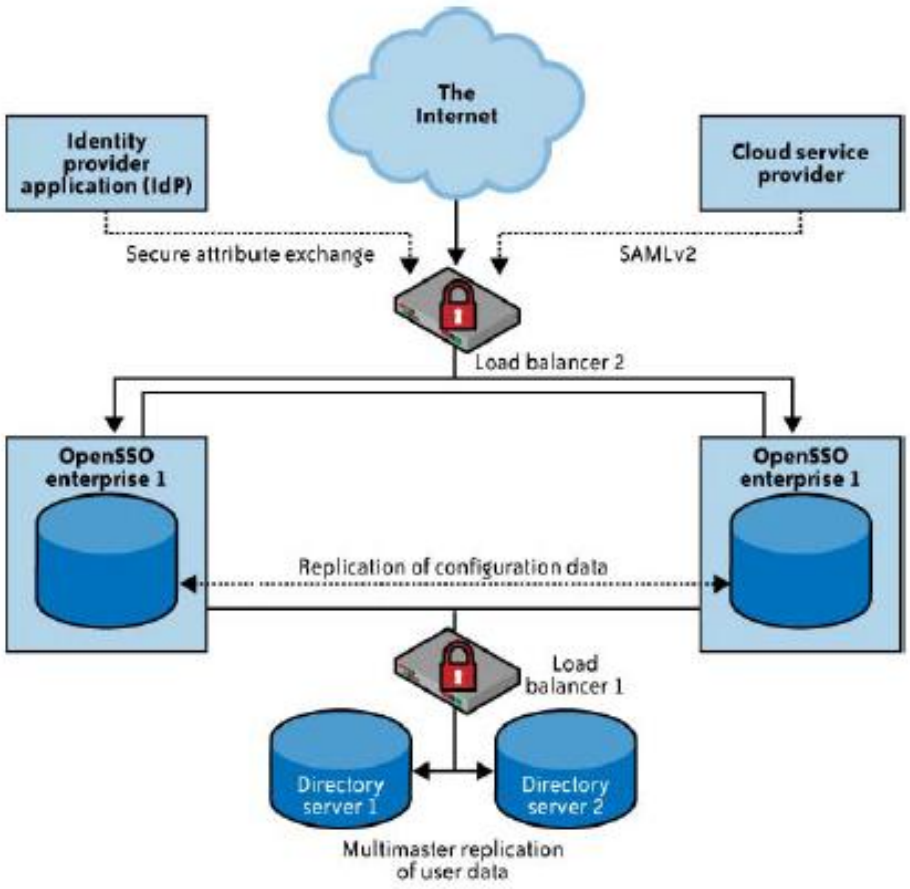
Cons

- By not changing the infrastructure to support federation, new inefficiencies can result due to the addition of life cycle management for non-employees such as customers.

Identity management-as-a-service

- In this architecture, cloud services can delegate authentication to an identity management-as a- service (IDaaS) provider. In this model, organizations outsource the federated identity
- management technology and user management processes to a third-party service provider, such as Ping Identity

Identity provider deployment architecture



- When federating identities to the cloud, organizations may need to manage the identity life cycle using their IAM system and processes.
- In cases where credentialing is difficult and costly, an enterprise might also outsource credential issuance (and background investigations) to a service provider, such as the GSA Managed Service Organization (MSO) that issues personal identity verification (PIV) cards and, optionally, the certificates on the cards.

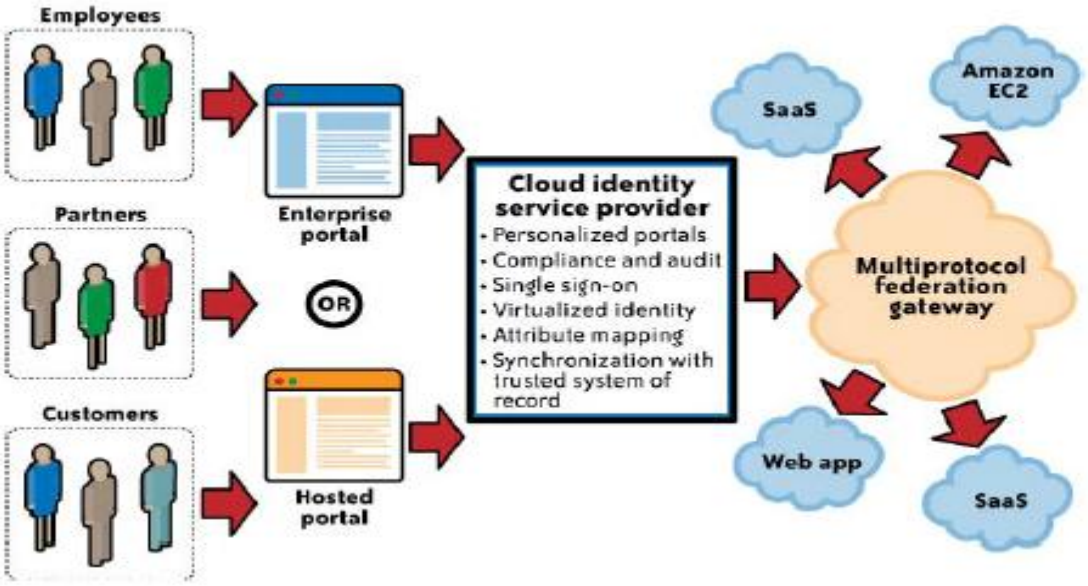
- In essence, this is a SaaS model for identity management, where the SaaS IdP stores identities in a "trusted identity store" and acts as a proxy for the organization's users accessing cloud services.

Pros

- Delegating certain authentication use cases to the cloud identity management service hides the complexity of integrating with various CSPs supporting different federation standards.

Cons

- When you rely on a third party for an identity management service, you may have less
- visibility into the service, including implementation and architecture details. Hence, the availability and authentication performance of cloud applications hinges on the identity management service provider's SLA, performance management, and availability. It is important to understand the provider's service level, architecture, service redundancy, and performance guarantees of the identity management service provider.



Identity management as a service

<u>Write about SaaS, PaaS, Iaas availability in cloud.</u>

SaaS

- One of the primary concerns of IT and business decision makers regarding software-as-a service applications is security management.
- Organizations considering integrating into SaaS services should consider two major challenges for identity management:
    - Is the organization ready to provision and manage the user life cycle by extending its
- established IAM practice to the SaaS service?

- Are the SaaS provider capabilities sufficient to automate user provisioning and life cycle management without implementing a custom solution for the SaaS service?

Customer responsibilities

- In SaaS services, customers have limited responsibility and available controls to secure information.

User provisioning

- User provisioning methods are typically unique to the SaaS provider.
- Customers need to understand the preferred method, lag time to activate users, and user attributes that are supported by the SaaS service.
- Almost all SaaS providers support bulk upload of user identities, as that's the most common use case for provisioning users. Some SaaS providers may support just-in-time provisioning where user identities are created on the fly using a provisioning request.

Profile management

- As part of the provisioning process, customers may have the ability to create user profiles that play a role in user authorization. User profiles such as user and manager are an approach to assigning entitlements to users within the SaaS application

SaaS IAM capability evaluation

- Customers are responsible for evaluating the support for IAM features such as SSO (using identity federation) by CSPs.
- SAML is the de facto standard for federating identities and is now supported by large SaaS providers

Investigation support

- Logs and audit trails are also often needed to investigate incidents. For example, PCI DSS requires the provider to "provide for timely forensic investigation" if the service provider suffers a breach.

Compliance management

CSP responsibilities

- With regard to IAM, some responsibilities belong to the CSP and some belong to the customer.
- Here are CSP responsibilities:

Authentication services

- Since users can be accessing the service from anywhere on the Internet, it is up to the SaaS provider to authenticate users based on the network trust level.

Account management policies

- CSPs should communicate the account management policies including account lock-outs, account provisioning methods, and privilege account management roles.

Federation

- CSPs supporting identity federation using standards such as SAML should publish the information necessary for customers to take advantage of this feature and enable SSO for their users. Such information includes the version SAML 1.1, SAML 2.0

PaaS

- Organizations considering extending their established IAM practices to PaaS cloud providers have few options at their disposal.
- PaaS CSPs typically delegate authentication functions using federation to the PaaS provider's IdP.

IaaS

- Some of the responsibilities and challenges in managing users in IaaS services are:

User provisioning

- Provisioning of users (developers, administrators) on IaaS systems that are dynamic in nature. Given that hundreds of systems are provisioned for workload management, user provisioning will have to be automated at the time of image creation and should be policy based.

Privileged user management

- Managing private keys of system administrators and protecting the keys when system administrators leave the company (e.g., SSH host keys).

Customer key assignment

- Assigning IDs and keys required to access the service. These keys are used for managing access to customer accounts for billing reasons, as well as for authenticating customers to their services.

Developer user management

- Provisioning of developers and testers to IaaS instances, and de provisioning the same when access is no longer required.

End user management

- Provisioning users who need access to applications hosted on IaaS.

List out the key privacy issues in cloud.

- Privacy rights or obligations are related to the collection, use, disclosure, storage, and destruction of personal data.
- The rights and obligations of individuals and organizations with respect to the collection, use, retention, and disclosure of personal information.

Access

- Data subjects have a right to know what personal information is held and, in some cases, can make a request to stop processing it.

- In the cloud, the main concern is the organization's ability to provide the individual with access to all personal information, and to comply with stated requests.
- If a data subject exercises this right to ask the organization to delete his data, will it be possible to ensure that all of his information has been deleted in the cloud?

Compliance

- What are the privacy compliance requirements in the cloud?
- What are the applicable laws, regulations, standards, and contractual commitments that govern this information, and who is responsible for maintaining the compliance?
- How are existing privacy compliance requirements impacted by the move to the cloud?

Storage

- Where is the data in the cloud stored?
- Was it transferred to another data center in another country?
- Is it commingled with information from other organizations that use the same
- CSP?
- Privacy laws in various countries place limitations on the ability of organizations to
- transfer some types of personal information to other countries.

Retention

- How long is personal information (that is transferred to the cloud) retained? Which
- retention policy governs the data? Does the organization own the data, or the CSP?

Destruction

- How does the cloud provider destroy PII at the end of the retention period?

Audit and monitoring

- How can organizations monitor their CSP and provide assurance to relevant stakeholders that privacy requirements are met when their PII is in the cloud?

Privacy breaches

- How do you know that a breach has occurred, how do you ensure that the CSP notifies you when a breach occurs, and who is responsible for managing the breach notification process.

Unit V

TWO MARKS

SET I

1. State the use of Security demand and Trust Index.

A user job demands the resource site to provide security assurance by issuing a security demand. The site needs to reveal its trustworthiness called trust index.

2. Mention the trust models.

Generalized trust model
Reputation based trust model

Fuzzy trust model
Authentication and Authorization models

3. State the challenges to establish trust among grids.

The first is integration with existing system technology.
Interoperability with hosting environments.
Constructing trust relationships among interacting hosting environment.

4. Mention the security issues of grid.

Out of control access
Network sniffers
Integration of local security mechanism
Faulty operation
Malicious operation
Dynamic service

5. What is implementing PKI and CA?

To implement PKI we use trusted third party called the certificate authority.
Each user applies a unique pair of public and private keys.
The public keys are issued by certificate authority after recognizing.

6. What are the risk factors in interacting with cloud topology with our existing network topology?

Ensuring the confidentiality and integrity of your organization's data-in-transit to and from your public cloud provider
Ensuring proper access control (authentication, authorization, and auditing) to whatever resources you are using at your public cloud provider
Ensuring the availability of the Internet-facing resources in a public cloud that are being used by your organization, or have been assigned to your organization by your public cloud providers
Replacing the established model of network zones and tiers with domains

7. State the security controls at network level.

Threat outlook Low
Preventive controls :-Network access control supplied by provider (e.g., firewall), encryption of data in transit (e.g., SSL,IPSec)
Detective controls:- Provider:-managed aggregation of security event logs (security incident and event management, or SIEM), network-based intrusion detection system/intrusion prevention system (IDS/IPS)

8. List the types of host level security.

SaaS and PaaS Host Security
IaaS Host Security
Virtualisation software security
Virtual server security

9. What are the threats in hypervisor?

A vulnerable hypervisor could expose all user domains to malicious insiders. Furthermore, hypervisors are potentially susceptible to subversion attacks.

10. What are the threats in public IaaS cloud?

Stealing keys used to access and manage hosts (e.g., SSH private keys)
Attacking unpatched, vulnerable services listening on standard ports (e.g., FTP, NetBIOS, SSH)
Hijacking accounts that are not properly secured (i.e., weak or no passwords for standard accounts)
Attacking systems that are not properly secured by host firewalls
Deploying Trojans embedded in the software component in the VM or within the VM image (the OS) itself

11. State the security controls at host level.

Preventive controls:- Host firewall, access control, patching, hardening of system, strong authentication
Detective controls:- Security event logs, host-based IDS/IP

12. List the risks in DoS attacks.

Application-level DoS and DDoS attacks that can potentially disrupt cloud services for an extended time. These attacks typically originate from compromised computer systems attached to the Internet. Apart from disrupting cloud services, resulting in poor user experience and service-level impacts, DoS attacks can quickly drain company's cloud services budget. DoS attacks on pay-as-you-go cloud applications will result in a dramatic increase in the cloud utility bill. This type of attack is also being characterized as economic denial of sustainability (EDoS).

13. State the security controls at application level.

Preventive controls:- Identity management, access control assessment, browser hardened with latest patches,multifactor authentication via delegated authentication, endpoint security measures including antivirus and IPS
Detective controls: - Login history and available reports from SaaS vendors

14. State the types of application security.

SaaS application security
PaaS application security
    Security of the PaaS platform itself (i.e., runtime engine)
    Security of customer applications deployed on a PaaS platform
Customer deployed application security
Iaas application security

15. Mention are the aspects of data security.

Data-in-transit
Data-at-rest
Processing of data, including multitenancy
Data lineage
Data provenance
Data remanence

16. Define integrity of data and provenance.

Integrity of data refers to data that has not been changed in
an unauthorized manner or by an unauthorized person. Provenance means not only that the data has integrity, but also that it is computationally accurate; that is, the data was accurately calculated

17. Define data remanence.

Data remanence is the residual representation of data that has been in some way nominally erased or removed. This residue may be due to data being left intact by a nominal delete operation, or through physical properties of the storage medium. Data remanence may make inadvertent disclosure of sensitive information possible, should the storage media be released into an uncontrolled environment.

18. Define data lineage.

Following the path of data - mapping application data flows or data path visualization is known as data lineage, and it is important for an auditor's assurance (internal, external, and regulatory).

19. What are the provider operations for security?

Provider collects and must protect a huge amount of security-related data.
For example, at the network level, your provider should be collecting, monitoring, andprotecting firewall, intrusion prevention system (IPS), security incident and event
management (SIEM), and router flow data. At the host level your provider should be collecting system logfiles, and at the application level SaaS providers should be collecting application log data, including authentication and authorization information.
Storage
Confidentiality
Integrity
Availability

20. State the need for IAM.

The organization invest in IAM practices to improve operational efficiency and to comply with regulatory, privacy, and data protection requirements:
Improve operational efficiency
Regulatory compliance management

21. What is IAM?

An identity access management (IAM) system is a framework for business processes that facilitates the management of electronic identities.

22. State some cases that require IAM support from CSP.

Some of the cloud use cases that require IAM support from the CSP include:
Employees and on-site contractors of an organization accessing a SaaS service using
identity federation
IT administrators accessing the CSP management console to provision resources and access for users using a corporate identity

Developers creating accounts for partner users in a PaaS platform
End users accessing storage service in the cloud
An application residing in a cloud service provider

23. What are the challenges of IAM?

One critical challenge of IAM concerns managing access for diverse user populations (employees, contractors, partners, etc.) accessing internal and externally hosted services. Another issue is the turnover of users within the organization.

24. What is the definition of IAM?

Authentication

Authentication is the process of verifying the identity of a user or system.

Authorization

Authorization is the process of determining the privileges the user or system is entitled to once the identity is established.

Auditing

In the context of IAM, auditing entails the process of review and examination of authentication, authorization records, and activities to determine the adequacy of IAM system controls, to verify compliance with established security policies and procedures , to detect breaches in security services, and to recommend any changes that are indicated for countermeasures.

25. What is the category of IAM processes to support the business?

User management
Authentication management
Authorization management
Access management
Data management and provisioning
Monitoring and auditing

26. Mention the operational activities supported by IAM process.

Provisioning
Credential and attribute management
Entitlement management
Compliance management
Identity federation management
Centralization of authentication (authN) and authorization (authZ)

27. Mention the key privacy issues in cloud.

Access
Compliance
Storage
Retention
Destruction
Audit and monitoring
Privacy breaches

Additional question: (Not in syllabus but may be asked)

IAM Standards and Specifications for Organizations

The following IAM standards and specifications will help organizations implement effective and efficient user access management practices and processes in the cloud.

These sections are ordered by four major challenges in user and access management faced by cloud users:

1. How can I avoid duplication of identity, attributes, and credentials and provide a single sign-on user experience for my users? SAML.
2. How can I automatically provision user accounts with cloud services and automate the process of provisoning and deprovisioning? SPML.
3. How can I provision user accounts with appropriate privileges and manage entitlements for my users? XACML.
4. How can I authorize cloud service X to access my data in cloud service Y without disclosing credentials? OAuth.

Organizations should start with an IAM strategy and architecture and invest in foundational technology elements that support user management and federation. In addition to providing a consistent user experience, federation can help to mitigate risks to organizations since it supports the SSO user experience: users will not be required to sign in multiple times, nor will they have to remember cloud-service-specific user authentication information.

Architecting an identity federation model will help organizations gain capabilities to support an identity provider (IdP), also known as an *SSO provider*

In that architecture, enterprise can share identities with trusted CSPs without sharing user credentials or private user attributes.

Management of identity attributes also plays a role in federation; the definition, descriptions, and management of mandatory, non-mandatory, and key attributes are necessary steps to prepare for federation.

This approach can help organizations extend IAM processes and practices, and implement a standardized federation model to *federate identities* and support single or reduced sign-on to cloud services.

Federation technology is typically built on a centralized identity management architecture leveraging industry-standard identity management protocols, such as Security Assertion Markup Language (SAML), WS Federation (WS-*), or Liberty Alliance.

These federation standards combined their work in enhancing SAML 1.0 to create SAML 2.0, which is the culmination of work stemming from the Organization for the Advancement of Structured Information Standards (OASIS), the Liberty Alliance, and the Shibboleth Project.

- ☐ avoid duplication of identity, attributes, and credentials and provide a single sign-on user experience
    - ☐ SAML(Security Assertion Markup Lang).
- ☐ automatically provision user accounts with cloud services and automate the process of provisioning and deprovisioning
    - ☐ SPML (service provisioning markup lang).

- provision user accounts with appropriate privileges and manage entitlements
  - XACML (extensible access control markup lang).
- authorize cloud service X to access my data in cloud service Y without disclosing credentials
  - Oauth (open authentication).

eXensible Access Control Markup Language (XACML)

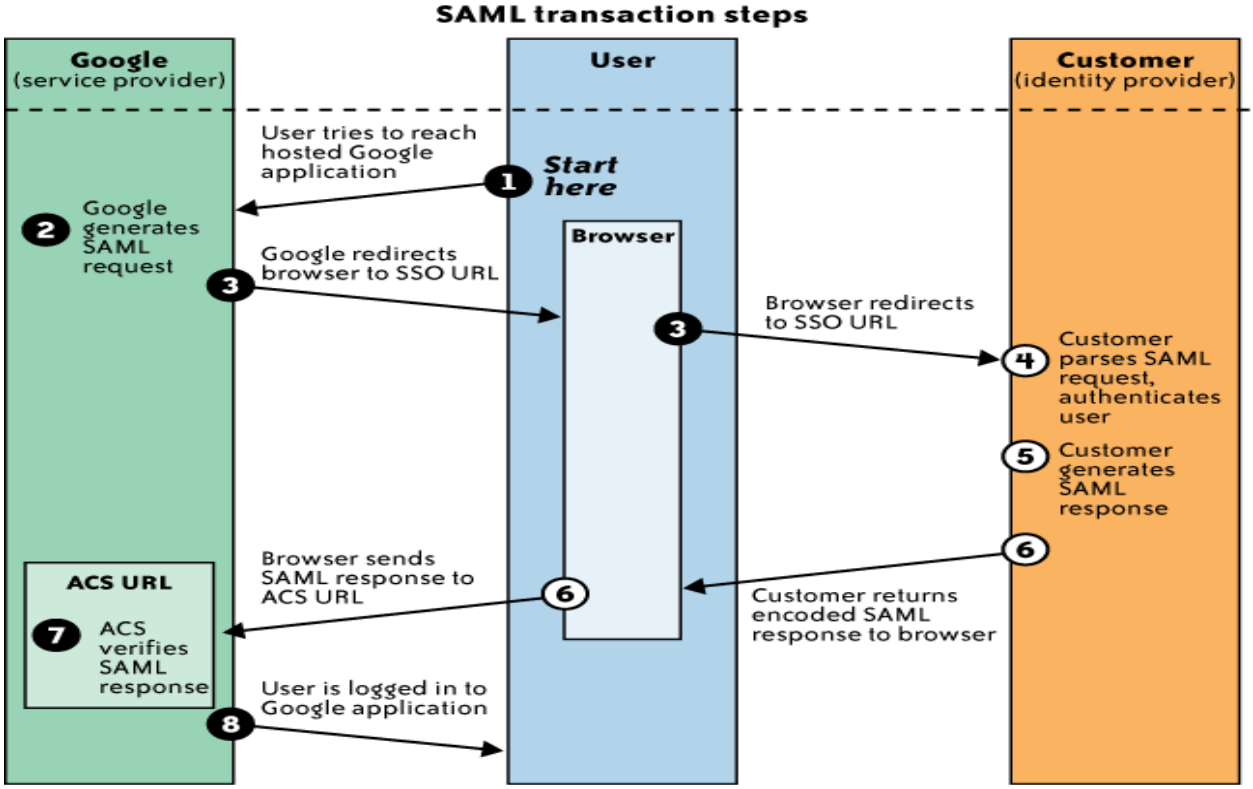
XACML is an OASIS-ratified, general-purpose, XML-based access control language for policy management and access decisions.

It provides an XML schema for a general policy language which is used to protect any kind of resource and make access decisions over these resources.

The XACML standard not only gives the model of the policy language, but also proposes a processing environment model to manage the policies and to conclude the access decisions.

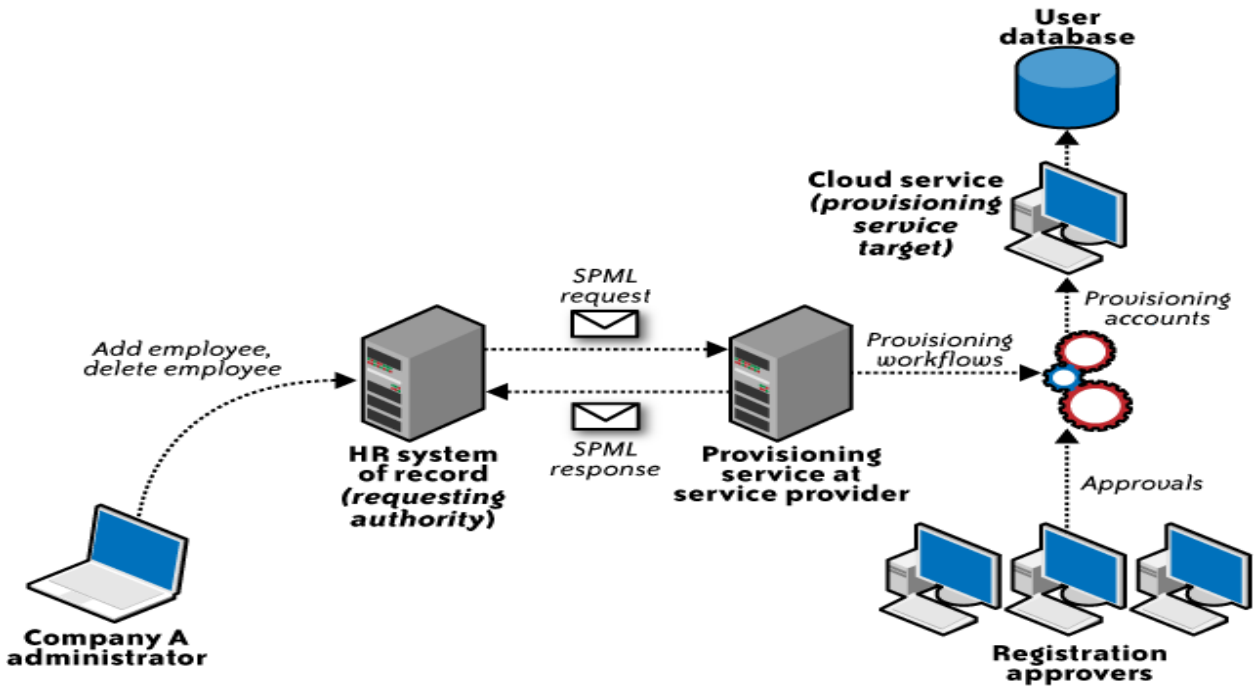
The XACML context also specifies the request/response protocol that the application environment can use to communicate with the decision point. The response to an access request is also specified using XML.

Google Account ex



SPML example:

What happens when an account is created?



Open Authentication (OAuth)

OAuth is an emerging authentication standard that allows consumers to share their private resources (e.g., photos, videos, contact lists, bank accounts) stored on one CSP with another CSP without having to disclose the authentication information (e.g., username and password). OAuth is an open protocol and it was created with the goal of enabling authorization via a secure application programming interface (API)—a simple and standard method for desktop, mobile, and web applications.

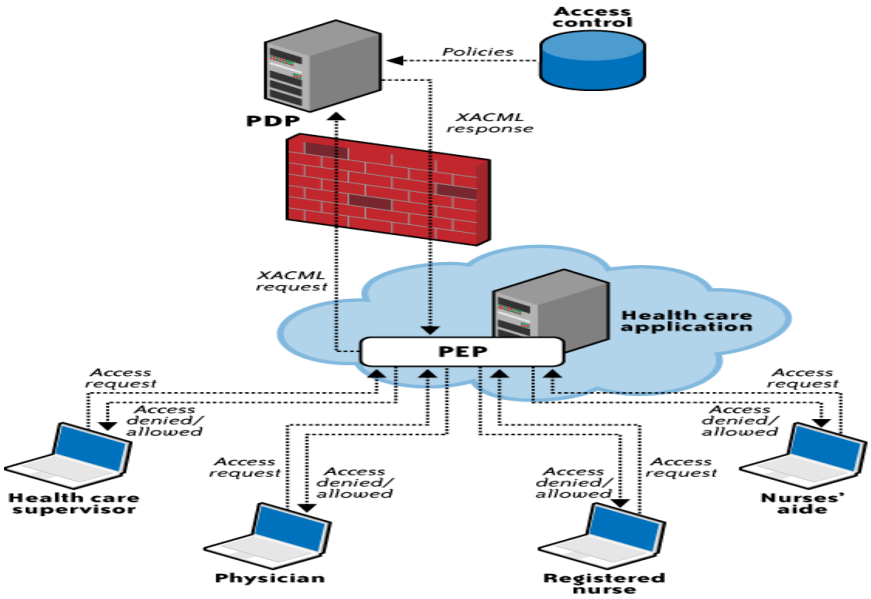
XACM Examples:

How does your access is verified?

PEP: policy enforcement point

(app interface)

PDP: policy decision point



The figure illustrates the following steps involved in the XACML process:

1. The health care application manages various hospital associates (the physician, registered nurse, nurses' aide, and health care supervisor) accessing various elements of the patient record. This application relies on the policy enforcement point (PEP) and forwards the request to the PEP.
2. The PEP is actually the interface of the application environment. It receives the access requests and evaluates them with the help of the policy decision point (PDP). It then permits or denies access to the resource (the health care record).
3. The PEP then sends the request to the PDP. The PDP is the main decision point for access requests. It collects all the necessary information from available information sources and concludes with a decision on what access to grant. The PDP should be located in a trusted network with strong access control policies, e.g., in a corporate trusted network protected by a corporate firewall.
4. After evaluation, the PDP sends the XACML response to the PEP.
5. The PEP fulfills the obligations by enforcing the PDP's authorization decision.

1. Customer web application contacts the Google Authorization service, asking for a request token for one or more Google service.
2. Google verifies that the web application is registered and responds with an unauthorized request token.
3. The web application directs the end user to a Google authorization page, referencing the request token.
4. On the Google authorization page, the user is prompted to log into his account (for verification) and then either grant or deny limited access to his Google service data by the web application.
5. The user decides whether to grant or deny access to the web application. If the user denies access, he is directed to a Google page and not back to the web application.
6. If the user grants access, the Authorization service redirects him back to a page designated with the web application that was registered with Google. The redirect includes the nowauthorized

request token.
7. The web application sends a request to the Google Authorization service to exchange the authorized request token for an access token.
8. Google verifies the request and returns a valid access token.
9. The web application sends a request to the Google service in question. The request is signed and includes the access token.
10. If the Google service recognizes the token, it supplies the requested data.

OAuth example:
Authorize the third party to Access your data/credential

ANNA UNIVERSITY QUESTIONS (UNIT 4 & 5)
Nov/Dec 2016
PART A

1. Name any four services offered in GT4.
2. What are the advantages of using Hadoop?
3. Mention the importance of transport level security.
4. Discuss on the application and use of identity and access management.

Part B

1. Draw and explain the global toolkit architecture
2. Give a detailed note on Hadoop framework.
3. Explain trust models for grid security environment.
4. Write in detail about cloud security infrastructure.

Apr / May 2017
PART A

Write the significant use f GRAM.
Name the different modules in Hadoop framework.
What are the various challenges in building the trust environment?
Write a brief note on the security requirements of a grid.

PART B

1. Discuss MAPREDUCE with suitable diagrams.
2. Elaborate HDFS concepts with suitable illustrations.
3. Write detailed note on identity and access management architecture.
4. Explain grid security infrastructure.

Nov/dec 2017
PART A

1. "HDFS is fault tolerant. Is it true? Justify your answer.

2. What is the purpose of heart beat in hadoop.
3. List any four host security threats in public IaaS.
4. Identify the trust model based on a site's trust worthiness.

PART B

1. Illustrate dataflow in HDFS during the components of GT4 with a suitable diagram.
2. What is GT4? Describe in detail the components of GT4 with a suitable diagram.
3. What is the purpose of GSI? Describe the functionality of various layers in GSI.
4. What is the purpose of IAM? Describe its functional architecture with an illustration.

Apr/May 2018
PART A

1. How does divide-and-conquer strategy relates to MapReduce praradigm?
2. Brief out the main components of Globus Toolkit.
3. On what basis trust models are set for grid environment?
4. State how CIA Triad plays a vital role in managing cloud security.

Part B

1. List the characteristics of globus tool kit. With a neat sketch describe sketch describe the architecture of globus GT4 and the services offered.
2. With an illustration, emphasize the significance of MapReduce paradigm in Hadoop framework. Listout the assumptions and goals set in HDFS architetucre for processing the data based on divide-and-conquer strategy.
3. " In today's world, infrastructure security and data security is highly challenging at network, host and application levels". Justify and explain the several ways of protecting the data at transit and at rest
4. Explain the baseline identity and access management (IAM) factors to be practiced by the stakeholders of cloud services and the common key privacy issues likely to happen in the environment.