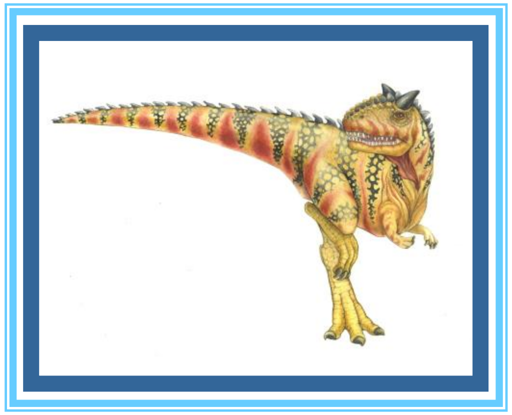
Week 7
Operating System Concepts

Muhammad Daniyal Liaquat



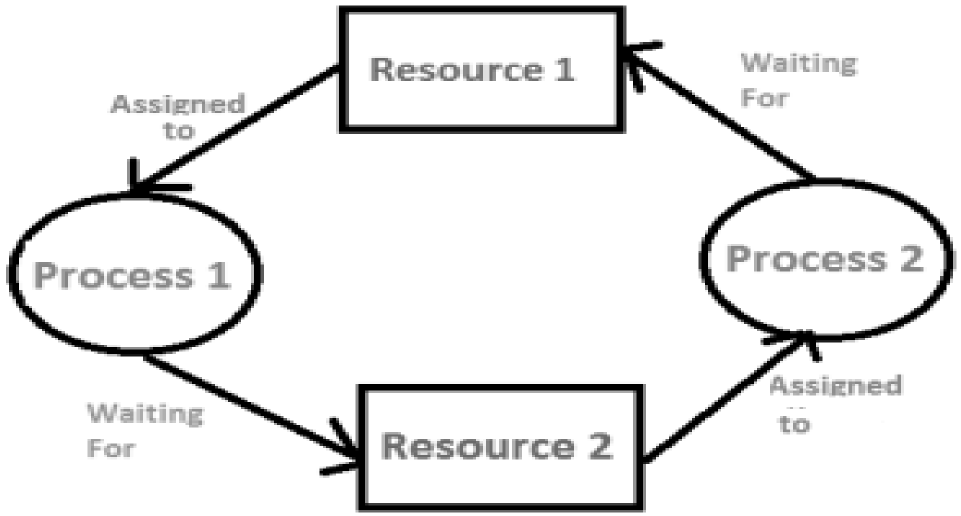
Topics

1. Deadlocks
2. Deadlock Conditions
3. Deadlock Prevention
4. Deadlock Avoidance

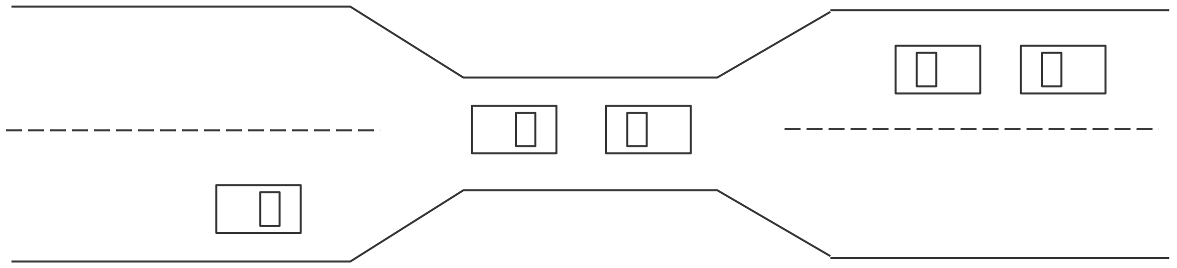
1. Deadlocks

Deadlocks

- Deadlock is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
- For example, in the below diagram, Process 1 is holding Resource 1 and waiting for resource 2 which is acquired by process 2, and process 2 is waiting for resource 1.



Bridge Crossing Example



- Traffic only in one direction.
- Each section of a bridge can be viewed as a resource.
- If a deadlock occurs, it can be resolved if one car backs up (preempt resources and rollback).
- Several cars may have to be backed up if a deadlock occurs.
- Starvation is possible.

System Model

- System consists of resources
- Resource types $R_1, R_2, \ldots, R_m$

  CPU cycles, memory space, I/O devices
- Each resource type R_i has W_i instances.
- Each process utilizes a resource as follows:
  - **request**
  - **use**
  - **release**

2. Deadlock Conditions

Conditions of Deadlocks

- Deadlock can arise if following four necessary conditions hold simultaneously.

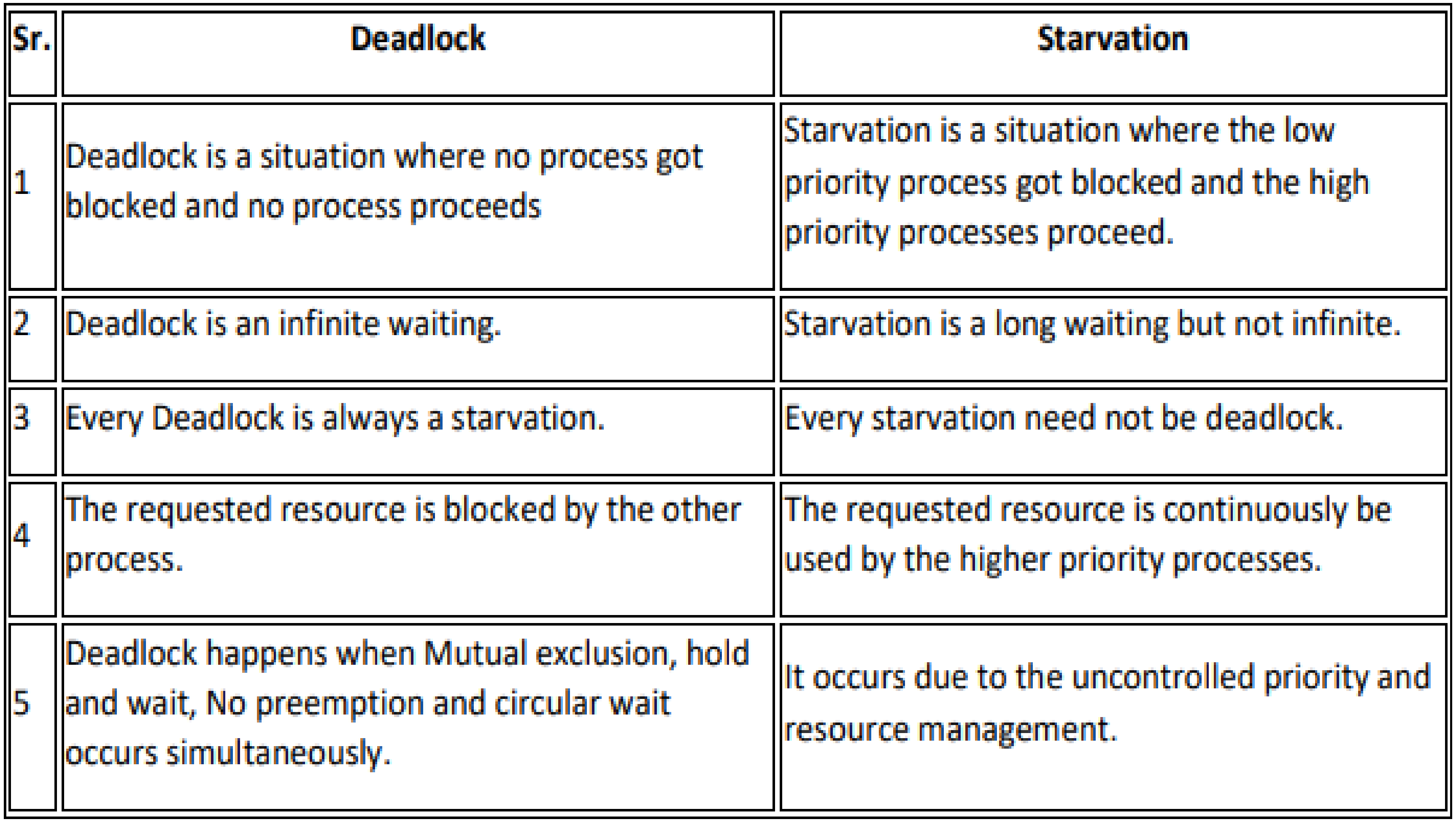
1. Mutual Exclusion: One or more than one resource are non-sharable means Only one process can use at a time.

2. Hold and Wait: A process is holding at least one resource and waiting for another resources.

3. No Pre-emption: A resource cannot be taken from a process unless the process releases the resource means the process which once scheduled will be executed till the completion and no other process can be scheduled by the scheduler meanwhile.

4. Circular Wait: A set of processes are waiting for each other in circular form means All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process

Difference between Starvation and Deadlock



Sr.	Deadlock	Starvation
1	Deadlock is a situation where no process got blocked and no process proceeds	Starvation is a situation where the low priority process got blocked and the high priority processes proceed.
2	Deadlock is an infinite waiting.	Starvation is a long waiting but not infinite.
3	Every Deadlock is always a starvation.	Every starvation need not be deadlock.
4	The requested resource is blocked by the other process.	The requested resource is continuously be used by the higher priority processes.
5	Deadlock happens when Mutual exclusion, hold and wait, No preemption and circular wait occurs simultaneously.	It occurs due to the uncontrolled priority and resource management.

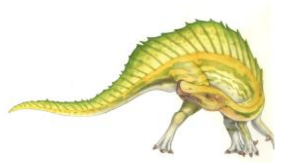
Deadlock with Mutex Locks

- Deadlocks can occur via system calls, locking, etc.
- See example box in text page 318 for mutex deadlock

Resource-Allocation Graph

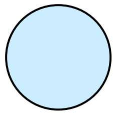
A set of vertices *V* and a set of edges *E*.

- V is partitioned into two types:
  - $P = \{P_1, P_2, \ldots, P_n\}$, the set consisting of all the processes in the system
  - $R = \{R_1, R_2, \ldots, R_m\}$, the set consisting of all resource types in the system
- **request edge** – directed edge $P_i \rightarrow R_j$
- **assignment edge** – directed edge $R_j \rightarrow P_i$

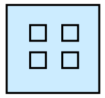


Resource-Allocation Graph (Cont.)

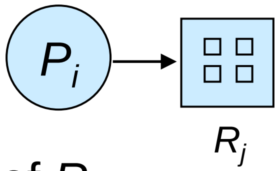
- Process



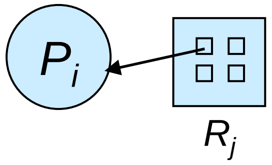
- Resource Type with 4 instances



- P_i requests instance of R_j



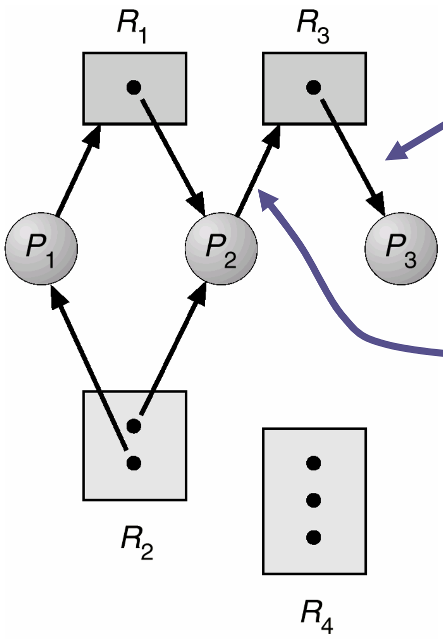
- P_i is holding an instance of R_j



RESOURCE ALLOCATION GRAPH

- If the graph contains no cycles, then no process is deadlocked.
- If there is a cycle, then:
  - a) If resource types have multiple instances, then deadlock MAY exist.
  - b) If each resource type has 1 instance, then deadlock has occurred.

Resource allocation graph

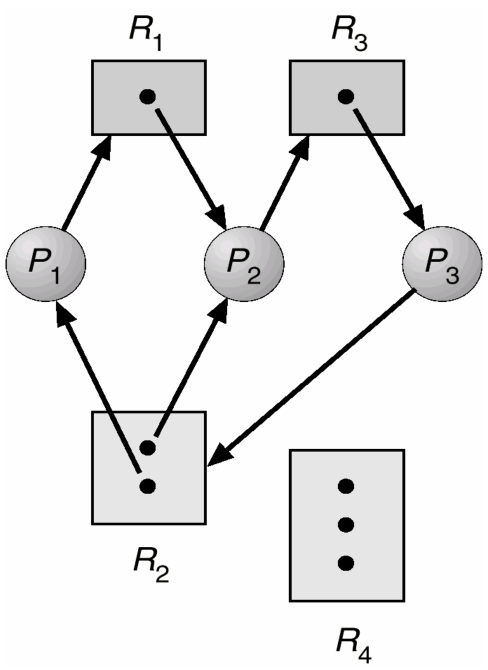


R3 Assigned to P3

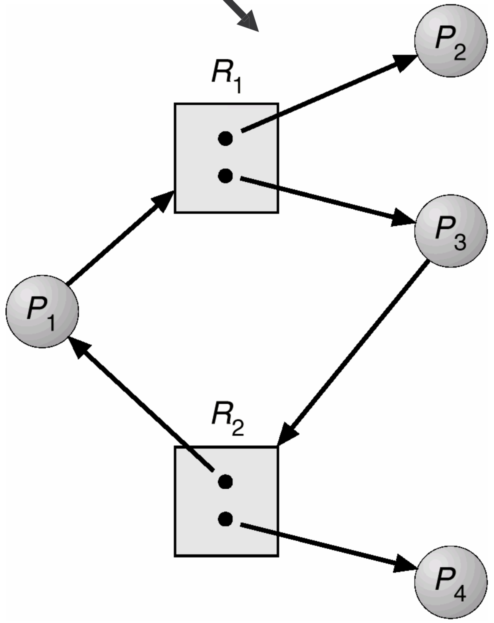
P2 Requests P3

RESOURCE ALLOCATION GRAPH

Resource allocation graph with a deadlock.



Resource allocation graph with a cycle but no deadlock.



Basic Facts

- If graph contains no cycles $\Rightarrow$ no deadlock
- If graph contains a cycle $\Rightarrow$
  - if only one instance per resource type, then deadlock
  - if several instances per resource type, possibility of deadlock

Deadlock Handling

- The various strategies for handling deadlock are

1. Deadlock Prevention
2. Deadlock Avoidance
3. Deadlock Detection and Recovery
4. Deadlock Ignorance

3. Deadlock Prevention

Deadlock Prevention

Deadlocks can be prevented by preventing at least one of the four required conditions:

- **Mutual Exclusion** – not required for sharable resources (e.g., read-only files); must hold for non-sharable resources
- Shared resources such as read-only files do not lead to deadlocks.
- Unfortunately, some resources, such as printers and tape drives, require exclusive access by a single process.

- **Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources
  - Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none allocated to it.
  - Low resource utilization; starvation possible.
  - To prevent this condition processes must be prevented from holding one or more resources while simultaneously

Deadlock Prevention (Cont.)

- **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
  - Preemption of process resource allocations can prevent this condition of deadlocks, when it is possible.
- **Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration
  - One way to avoid circular wait is to number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order. waiting for one or more others.

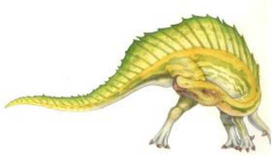
Deadlock Example

```
/* thread one runs in this function */
void *do_work_one(void *param)
{
   pthread_mutex_lock(&first_mutex);
   pthread_mutex_lock(&second_mutex);
   /** * Do some work */
   pthread_mutex_unlock(&second_mutex);
   pthread_mutex_unlock(&first_mutex);
   pthread_exit(0);
}
/* thread two runs in this function */
void *do_work_two(void *param)
{
   pthread_mutex_lock(&second_mutex);
   pthread_mutex_lock(&first_mutex);
   /** * Do some work */
   pthread_mutex_unlock(&first_mutex);
   pthread_mutex_unlock(&second_mutex);
   pthread_exit(0);
}
```

Deadlock Example with Lock Ordering

```
void transaction(Account from, Account to, double amount)
{
   mutex lock1, lock2;
   lock1 = get_lock(from);
   lock2 = get_lock(to);
   acquire(lock1);
      acquire(lock2);
         withdraw(from, amount);
         deposit(to, amount);
      release(lock2);
   release(lock1);
}
```

Transactions 1 and 2 execute concurrently. Transaction 1 transfers $25 from account A to account B, and Transaction 2 transfers $50 from account B to account A



4. Deadlock Avoidance

Deadlock Avoidance

Requires that the system has some additional ***a priori*** information available

- Simplest and most useful model requires that each process declare the ***maximum number*** of resources of each type that it may need
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition
- Resource-allocation *state* is defined by the number of available and allocated resources, and the maximum demands of the processes
- In deadlock avoidance, the operating system checks whether the system is in safe state or in unsafe state at every step which the operating system performs.
- The process continues until the system is in safe state.
- Once the system moves to unsafe state, the OS has to backtrack one step.
- In simple words, The OS reviews each allocation so that the allocation doesn't cause the deadlock in the system.

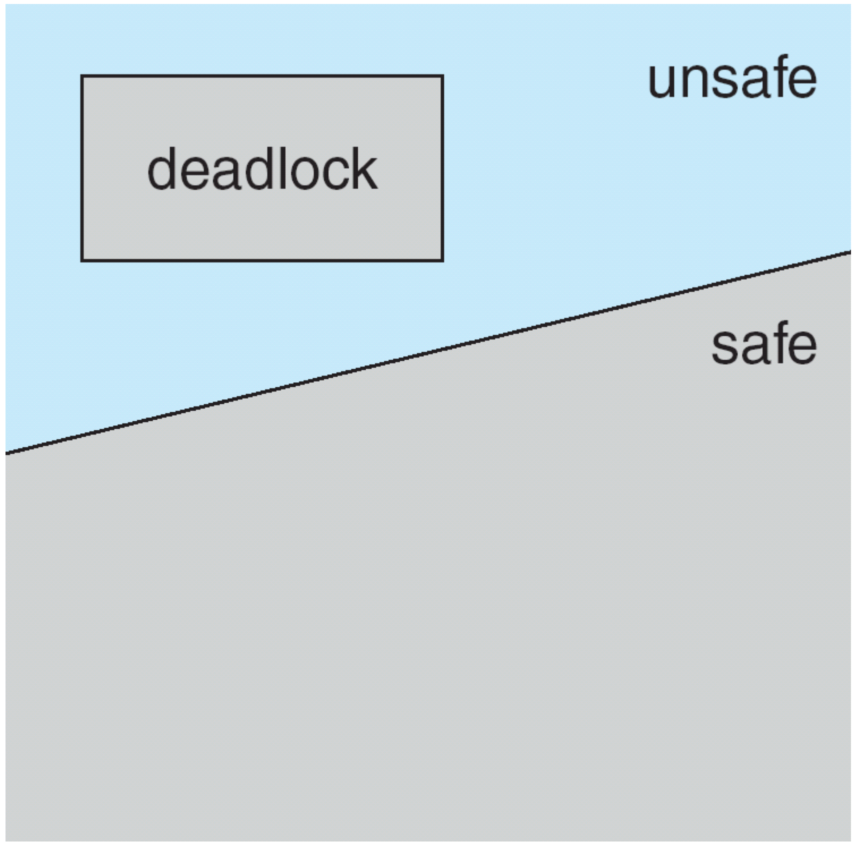
Safe State

- 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 $<P_1, P_2, \ldots, P_n>$ of ALL the processes in the systems such that for each P_i, the resources that P_i can still request can be satisfied by currently available resources + resources held by all the P_j, with $j < I$
- That is:
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j have finished
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on

Basic Facts

- If a system is in safe state $\Rightarrow$ no deadlocks
- If a system is in unsafe state $\Rightarrow$ possibility of deadlock
- Avoidance $\Rightarrow$ ensure that a system will never enter an unsafe state.

Safe, Unsafe, Deadlock State

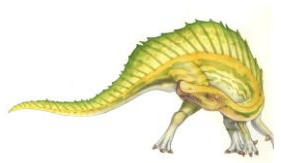


Avoidance Algorithms

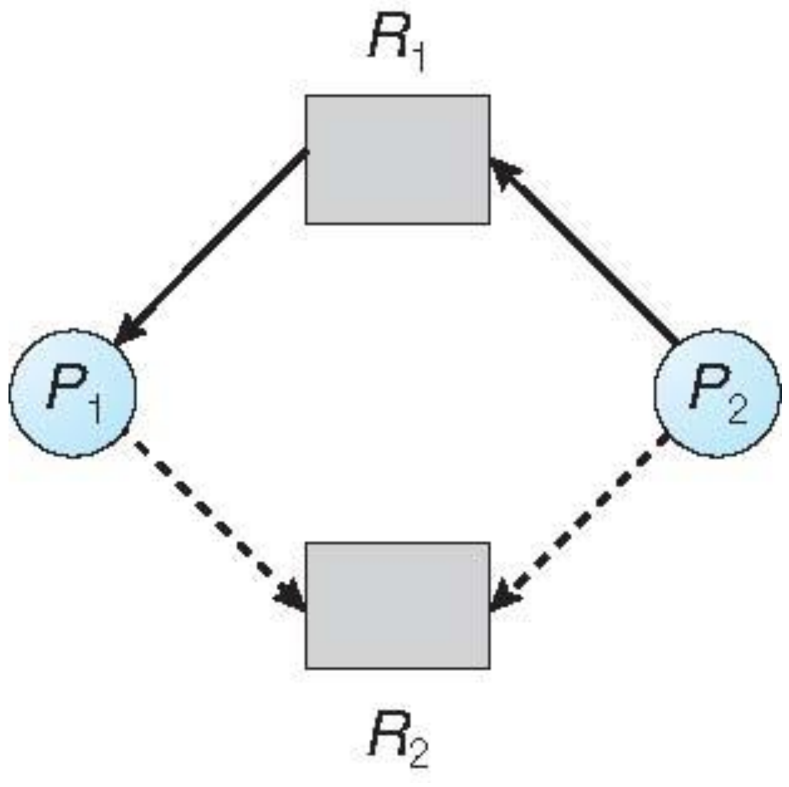
- Single instance of a resource type
  - Use a resource-allocation graph
- Multiple instances of a resource type
  - Use the banker’s algorithm

Resource-Allocation Graph Scheme

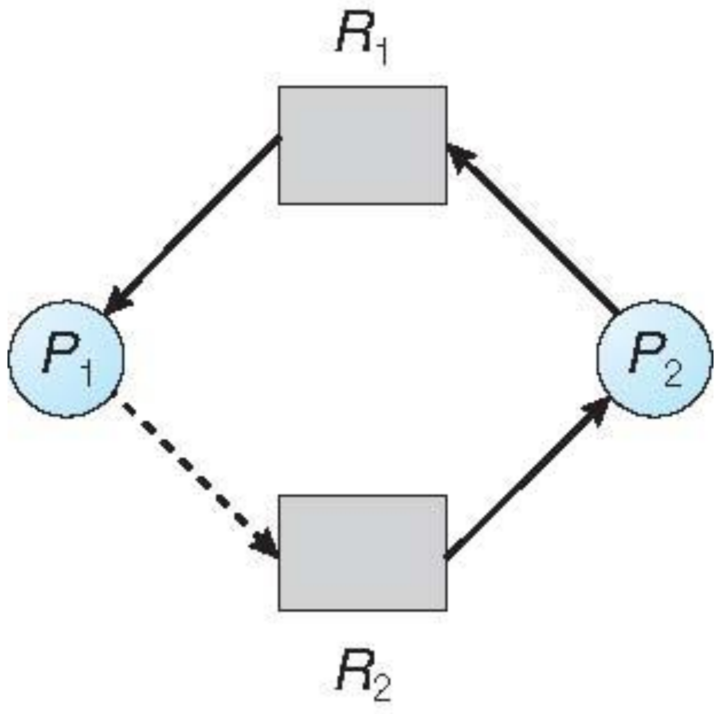
- **Claim edge** $P_i \rightarrow R_j$ indicated that process P_i may request resource R_j; represented by a dashed line
- Claim edge converts to request edge when a process requests a resource
- Request edge converted to an assignment edge when the resource is allocated to the process
- When a resource is released by a process, assignment edge reconverts to a claim edge
- Resources must be claimed *a priori* in the system



Resource-Allocation Graph



Unsafe State In Resource-Allocation Graph



Resource-Allocation Graph Algorithm

- Suppose that process *P_i* requests a resource *R_j*
- The request can be granted only if converting the request edge to an assignment edge does not result in the formation of a cycle in the resource allocation graph

Banker’s Algorithm

- 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

Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- **Available***:* Vector of length m. If available $[j] = k$, there are k instances of resource type R_j available
- **Max***:* n *x* m matrix. If *Max* $[i,j] = k$, then process P_i may request at most k instances of resource type R_j
- **Allocation***:* n x m matrix. If Allocation$[i,j] = k$ then P_i is currently allocated k instances of R_j
- **Need***:* n x m matrix. If *Need*$[i,j] = k$, then P_i may need k more instances of R_j to complete its task

$$Need\ [i,j] = Max[i,j] - Allocation\ [i,j]$$

Safety Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length *m* and *n*, respectively. Initialize:

   Work = Available

   ***Finish* [*i*] = *false* for *i* = 0, 1, …, *n*- 1**

2. Find an ***i*** such that both:

   (a) ***Finish* [*i*] = *false***

   (b) $\boldsymbol{Need_i \leq Work}$

   If no such ***i*** exists, go to step 4

3. $\boldsymbol{Work = Work + Allocation_i}$
   Finish*[*i*] = *true
   go to step 2

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

Resource-Request Algorithm for Process P_i

$Request_i$ = request vector for process P_i. If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j

1. If $Request_i \leq Need_i$ go to step 2. Otherwise, raise error condition, since process has exceeded its maximum claim
2. If $Request_i \leq Available$, go to step 3. Otherwise P_i must wait, since resources are not available
3. Pretend to allocate requested resources to P_i by modifying the state as follows:

$$Available = Available - Request_i;$$

$$Allocation_i = Allocation_i + Request_i;$$

$$Need_i = Need_i - Request_i;$$

- If safe $\Rightarrow$ the resources are allocated to P_i
- If unsafe $\Rightarrow$ P_i must wait, and the old resource-allocation state is restored

Example of Banker’s Algorithm

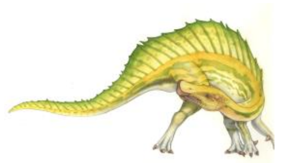
- 5 processes P_0 through P_4;

  3 resource types:

  A (10 instances), *B* (5instances), and *C* (7 instances)

- Snapshot at time T_0:

	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	



Example (Cont.)

- The content of the matrix ***Need*** is defined to be ***Max – Allocation***

	Need
	A B C
P_0	7 4 3
P_1	1 2 2
P_2	6 0 0
P_3	0 1 1
P_4	4 3 1

- The system is in a safe state since the sequence < P_1, P_3, P_4, P_2, P_0> satisfies safety criteria

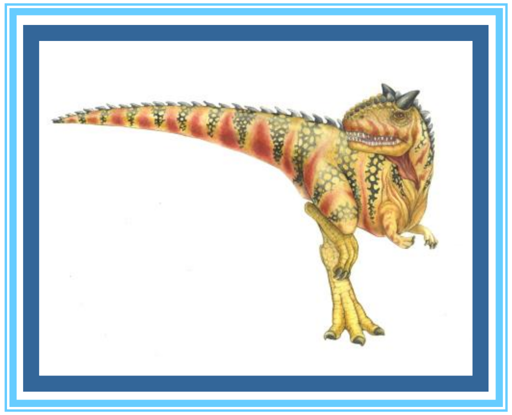
Example: P_1 Request (1,0,2)

- Check that Request $\leq$ Available (that is, $(1,0,2) \leq (3,3,2) \Rightarrow$ true

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

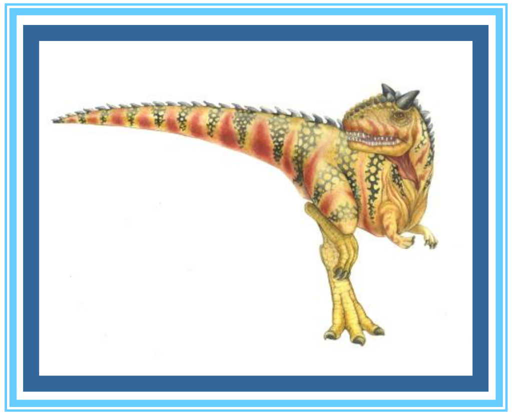
- Executing safety algorithm shows that sequence < $\boldsymbol{P_1, P_3, P_4, P_0, P_2}$> satisfies safety requirement
- Can request for (3,3,0) by $\boldsymbol{P_4}$ be granted?
- Can request for (0,2,0) by $\boldsymbol{P_0}$ be granted?

Any Questions ?



Week 8
Operating System Concepts

Muhammad Daniyal Liaquat



Topics

1. **Deadlock Detection & Recovery**
2. **Deadlock Ignorance**

1. Deadlock Detection & Recovery

3. Deadlock Detection

- Allow system to enter deadlock state
- Detection algorithm
- Recovery scheme
- This strategy involves waiting until a deadlock occurs.
- After deadlock occurs, the system state is recovered.
- The main challenge with this approach is detecting the deadlock

Single Instance of Each Resource Type

- Maintain **wait-for** graph
  - Nodes are processes
  - $\boldsymbol{P_i} \rightarrow \boldsymbol{P_j}$ if $\boldsymbol{P_i}$ is waiting for $\boldsymbol{P_j}$
- Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock
- An algorithm to detect a cycle in a graph requires an order of $\boldsymbol{n^2}$ operations, where ***n*** is the number of vertices in the graph

Deadlock Detection

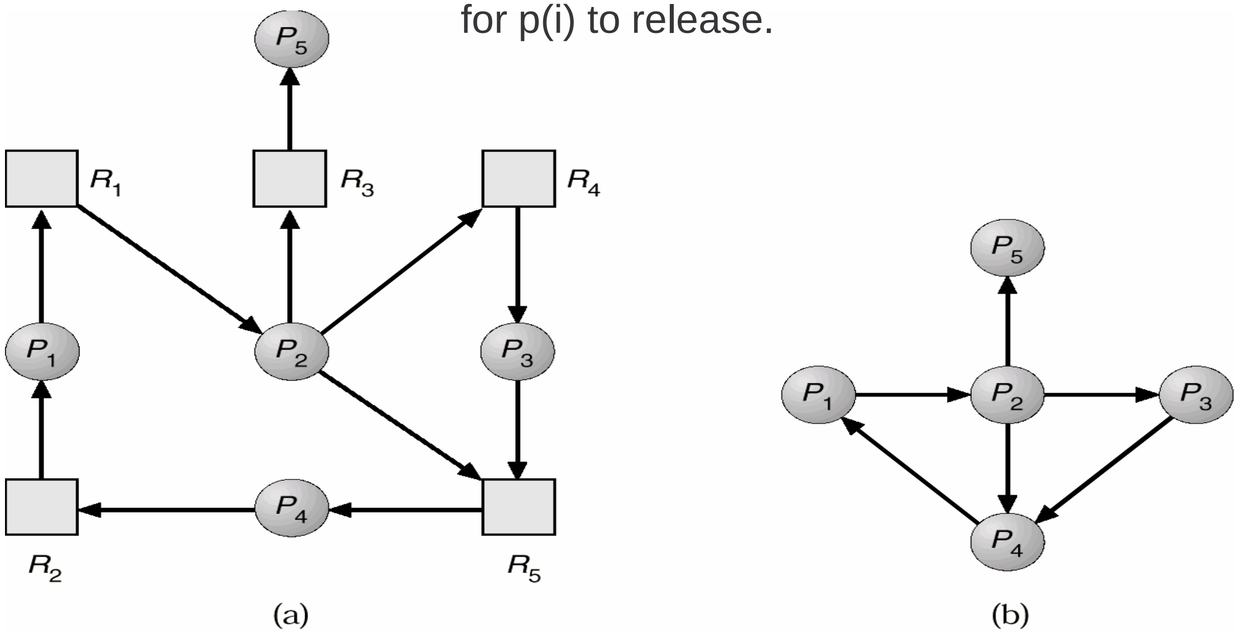
Need an algorithm that determines if deadlock occurred.

Also need a means of recovering from that deadlock.

SINGLE INSTANCE OF A RESOURCE TYPE

•Wait-for graph == remove the resources from the usual graph and collapse edges.

•An edge from p(j) to p(i) implies that p(j) is waiting for p(i) to release.



Several Instances of a Resource Type

- **Available***:* A vector of length ***m*** indicates the number of available resources of each type
- **Allocation***:* An ***n* x *m*** matrix defines the number of resources of each type currently allocated to each process
- **Request***:* An ***n* x *m*** matrix indicates the current request of each process. If ***Request* [*i*][*j*] = *k***, then process P_i is requesting ***k*** more instances of resource type R_j.

Detection Algorithm

1. Let ***Work*** and ***Finish*** be vectors of length ***m*** and ***n***, respectively Initialize:

   (a) ***Work = Available***

   (b) For ***i* = 1,2, …, *n***, if $\boldsymbol{Allocation_i} \neq \mathbf{0}$, then
   Finish*[i] = *false; otherwise, ***Finish*[i] = *true***

2. Find an index ***i*** such that both:

   (a) ***Finish*[*i*] == *false***

   (b) $\boldsymbol{Request_i} \leq \boldsymbol{Work}$

   If no such ***i*** exists, go to step 4…and if i doesn’t exist then go to step 3.

Detection Algorithm (Cont.)

3. ***Work* = *Work* + *Allocation*$_i$**
   Finish*[*i*] = *true
   go to step 2

4. If ***Finish[i] == false***, for some ***i***, 1 ≤ ***i*** ≤ ***n***, then the system is in deadlock state. Moreover, if ***Finish*[*i*] == *false***, then ***P$_i$*** is deadlocked

Algorithm requires an order of O(m x n^2) operations to detect whether the system is in deadlocked state

Example of Detection Algorithm

- Five processes ***P_0*** through ***P_4***; three resource types A (7 instances), *B* (2 instances), and *C* (6 instances)

- Snapshot at time ***T_0***:

	Allocation	*Request*	*Available*
	A B C	*A B C*	*A B C*
P_0	0 1 0	0 0 0	0 0 0
P_1	2 0 0	2 0 2	
P_2	3 0 3	0 0 0	
P_3	2 1 1	1 0 0	
P_4	0 0 2	0 0 2	

- Sequence <***P_0, P_2, P_3, P_1, P_4***> will result in ***Finish[i] = true*** for all ***i***

Example (Cont.)

- ***P*$_2$** requests an additional instance of type ***C***

	Request
	A B C
P$_0$	0 0 0
P$_1$	2 0 2
P$_2$	0 0 1
P$_3$	1 0 0
P$_4$	0 0 2

- State of system?
  - Can reclaim resources held by process ***P*$_0$**, but insufficient resources to fulfill other processes; requests
  - Deadlock exists, consisting of processes ***P*$_1$**, ***P*$_2$**, ***P*$_3$**, and ***P*$_4$**

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolled back?
    - one for each disjoint cycle

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes "caused" the deadlock.

Recovery from Deadlock: Process Termination

- Abort all deadlocked processes
- Abort one process at a time until the deadlock cycle is eliminated
- In which order should we choose to abort?
  1. Priority of the process
  2. How long process has computed, and how much longer to completion
  3. Resources the process has used
  4. Resources process needs to complete
  5. How many processes will need to be terminated
  6. Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- **Selecting a victim** – minimize cost
- **Rollback** – return to some safe state, restart process for that state
- **Starvation** – same process may always be picked as victim, include number of rollback in cost factor

Deadlock Recovery

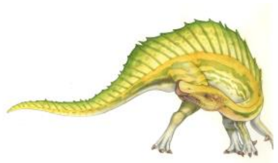
So, the deadlock has occurred. Now, how do we get the resources back and gain forward progress?

PROCESS TERMINATION:

- Could delete all the processes in the deadlock -- this is expensive.
- Delete one at a time until deadlock is broken (time consuming).
- Select who to terminate based on priority, time executed, time to completion, needs for completion, or depth of rollback
- In general, it's easier to preempt the resource, than to terminate the process.

RESOURCE PREEMPTION:

- Select a victim - which process and which resource to preempt.
- Rollback to previously defined "safe" state.
- Prevent one process from always being the one preempted (starvation).



Deadlock Recovery

- COMBINED APPROACH TO DEADLOCK HANDLING:

- Type of resource may dictate best deadlock handling. Look at ease of implementation, and effect on performance.

- In other words, there is no one best technique.

- Cases include:

- Preemption for memory, Preallocation for swap space,

- Avoidance for devices (can extract Needs from process.)

2. Deadlock Ignorance

4. Deadlock Ignorance

- This strategy involves ignoring the concept of deadlock and assuming as if it does not exist.
- This strategy helps to avoid the extra overhead of handling deadlock.
- Windows and Linux use this strategy and it is the most widely used method.

- "Deadlock ignorance" in the context of operating systems typically refers to a situation where system administrators, developers, or users are unaware of the possibility of deadlocks occurring in a system or how to deal with them. Deadlocks are a common issue in multi-threaded or multi-process systems, and they can lead to system unresponsiveness and inefficiency. Here are some key points related to "deadlock ignorance" in operating systems:

4. Deadlock Ignorance

1. Lack of Awareness: Deadlock ignorance can occur when those responsible for managing or developing an operating system are not aware of the conditions that can lead to deadlocks. This lack of awareness may result in systems that are more prone to experiencing deadlock situations.

2. No Detection or Handling: An operating system that is "deadlock ignorant" may lack mechanisms for detecting or handling deadlocks. Without proper deadlock detection and resolution techniques, a system can remain in a deadlock state indefinitely.

3. Failure to Implement Prevention Strategies: Operating system designers and administrators should be aware of various strategies for preventing deadlocks, such as resource allocation graphs, deadlock avoidance, or deadlock detection with recovery. Ignoring these prevention strategies can lead to an increased risk of deadlocks occurring.

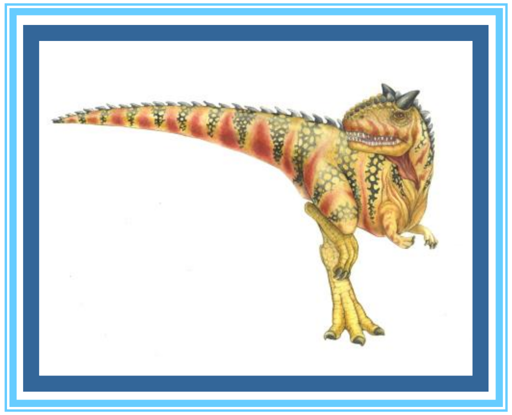
4. Deadlock Ignorance

4. Inadequate Resource Allocation Policies: Ignorance of proper resource allocation policies can lead to deadlocks. For example, if an operating system allows resources to be allocated without considering potential resource conflicts, it can increase the likelihood of deadlocks.

5. Insufficient User Education: Users of the operating system may also contribute to deadlock ignorance if they do not understand how to use the system's resources in a way that minimizes the risk of deadlocks. Educating users on proper system usage is essential to reduce the likelihood of deadlock scenarios.

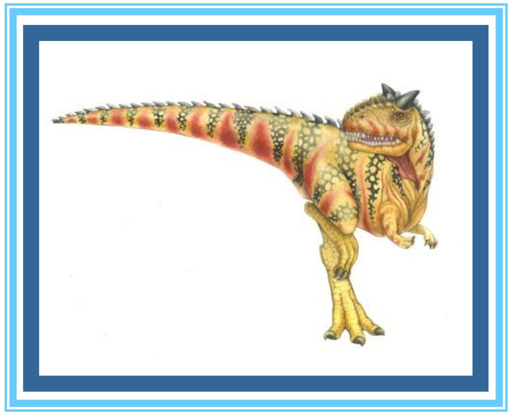
- To address deadlock ignorance, it's important to promote education and awareness of deadlock-related issues, both among system administrators and developers, and among end-users. Additionally, implementing proper deadlock detection and resolution mechanisms within the operating system can help mitigate the impact of deadlock situations when they do occur.

Any Questions ?



Week 9
Operating System Concepts

Muhammad Daniyal Liaquat



Topics

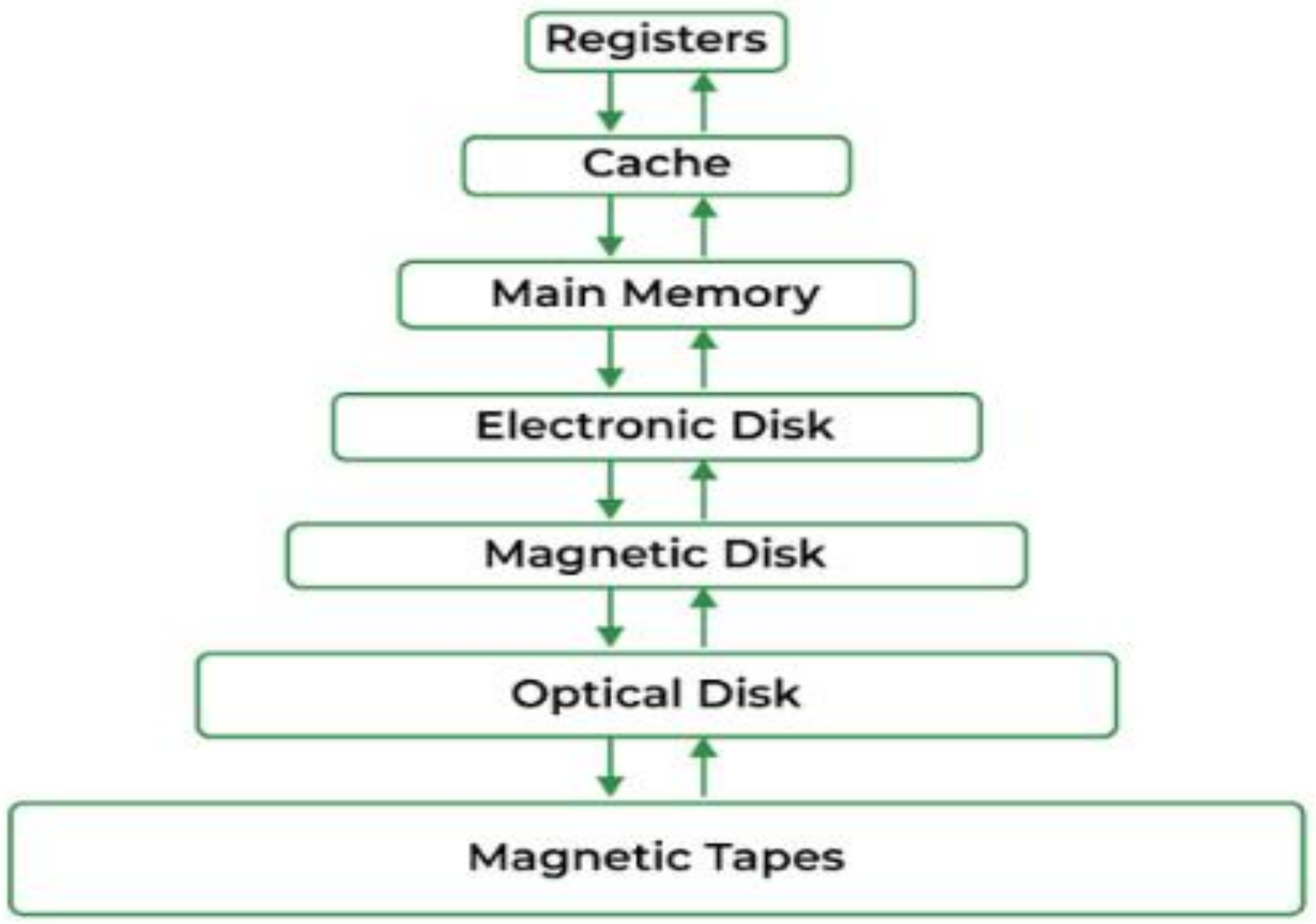
1. **Main Memory**
2. **Memory Management**

1. Main Memory

What is Main Memory?

- The main memory is central to the operation of a Modern Computer. Main Memory is a large array of words or bytes, ranging in size from hundreds of thousands to billions. Main memory is a repository of rapidly available information shared by the CPU and I/O devices. Main memory is the place where programs and information are kept when the processor is effectively utilizing them. Main memory is associated with the processor, so moving instructions and information into and out of the processor is extremely fast. Main memory is also known as RAM (Random Access Memory). This memory is volatile. RAM loses its data when a power interruption occurs.

What is Main Memory?



Main Memory

2. Memory Management

What is Memory Management?

- In a multiprogramming computer, the Operating System resides in a part of memory, and the rest is used by multiple processes. The task of subdividing the memory among different processes is called Memory Management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is Required?

- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

Logical and Physical Address Space

- **Logical Address Space:** An address generated by the CPU is known as a “Logical Address”. It is also known as a <u>Virtual address</u>. Logical address space can be defined as the size of the process. A logical address can be changed.
- **Physical Address Space:** An address seen by the memory unit (i.e. the one loaded into the memory address register of the memory) is commonly known as a “Physical Address”. A <u>Physical address</u> is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A <u>physical address</u> is computed by Memory Management Unit(MMU). The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

Static and Dynamic Loading

- Loading a process into the main memory is done by a loader. There are two different types of loading :
- **Static Loading:** Static Loading is basically loading the entire program into a fixed address. It requires more memory space.
- **Dynamic Loading:** The entire program and all data of a process must be in physical memory for the process to execute. So, the size of a process is limited to the size of physical memory. To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that the unused routine is never loaded. This loading is useful when a large amount of code is needed to handle it efficiently.

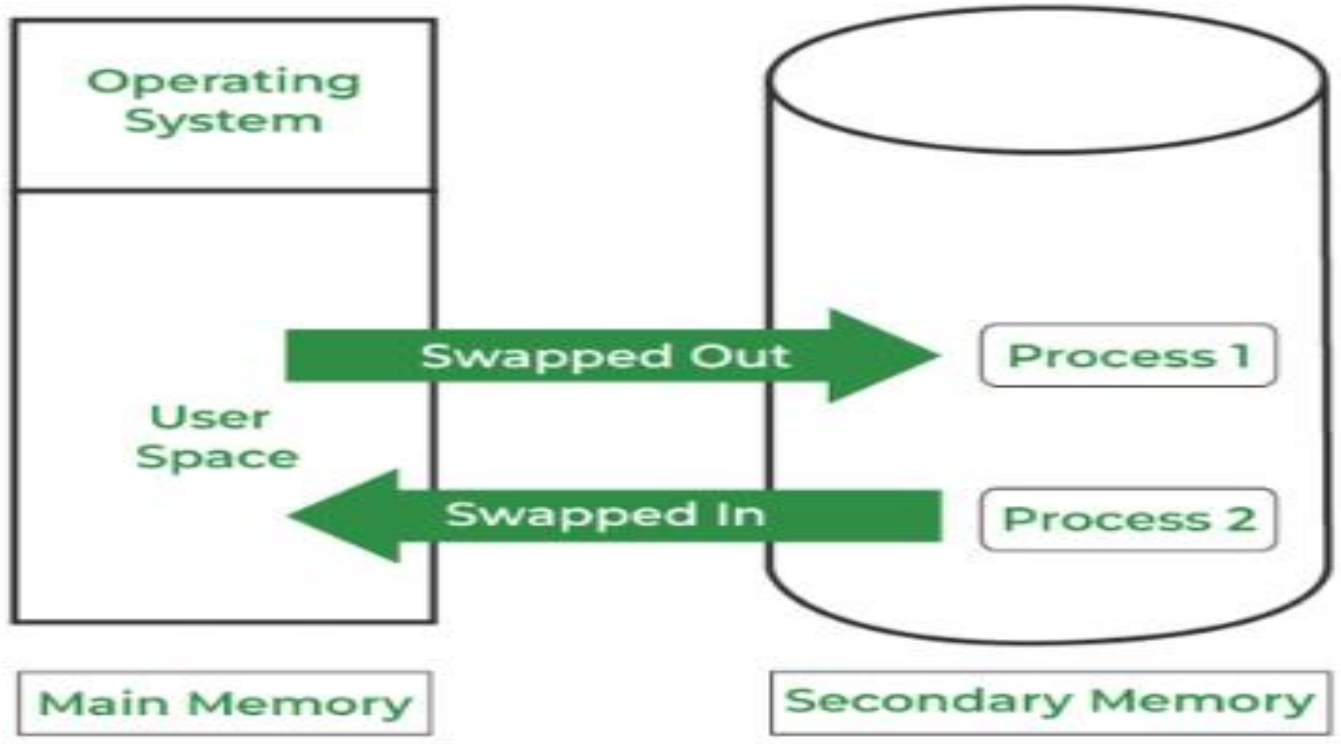
Static and Dynamic Linking

- To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.
- **Static Linking:** In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.
- **Dynamic Linking:** The basic concept of dynamic linking is similar to dynamic loading. In dynamic linking, “Stub” is included for each appropriate library routine reference. A stub is a small piece of code. When the stub is executed, it checks whether the needed routine is already in memory or not. If not available then the program loads the routine into memory.

Swapping

- When a process is executed it must have resided in memory. Swapping is a process of swapping a process temporarily into a secondary memory from the main memory, which is fast compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time. The main part of swapping is transferred time and the total time is directly proportional to the amount of memory swapped. Swapping is also known as roll-out, or roll because if a higher priority process arrives and wants service, the memory manager can swap out the lower priority process and then load and execute the higher priority process. After finishing higher priority work, the lower priority process swapped back in memory and continued to the execution process.

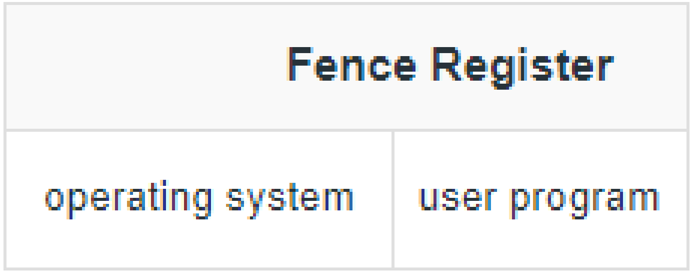
Swapping


Operating System
User Space
Swapped Out
Swapped In
Process 1
Process 2
Main Memory
Secondary Memory

Memory Management with Monoprogramming (Without Swapping)

This is the simplest memory management approach the memory is divided into two sections:

- One part of the operating system
- The second part of the user program



Fence Register	
operating system	user program

- In this approach, the operating system keeps track of the first and last location available for the allocation of the user program
- The operating system is loaded either at the bottom or at top
- Interrupt vectors are often loaded in low memory therefore, it makes sense to load the operating system in low memory
- Sharing of data and code does not make much sense in a single process environment
- The Operating system can be protected from user programs with the help of a fence register.

Advantages of Memory Management

- It is a simple management approach

Disadvantages of Memory Management

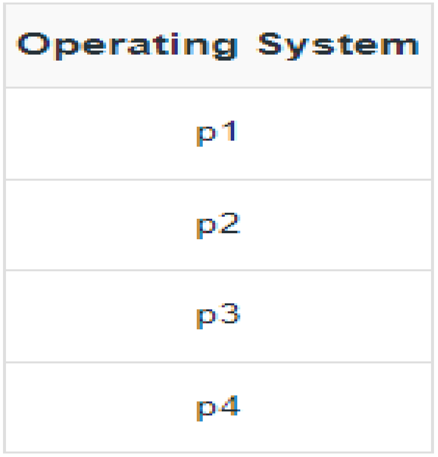
- It does not support multiprogramming
- Memory is wasted

Multiprogramming with Fixed Partitions (Without Swapping)

- A memory partition scheme with a fixed number of partitions was introduced to support multiprogramming. this scheme is based on contiguous allocation
- Each partition is a block of contiguous memory
- Memory is partitioned into a fixed number of partitions.
- Each partition is of fixed size

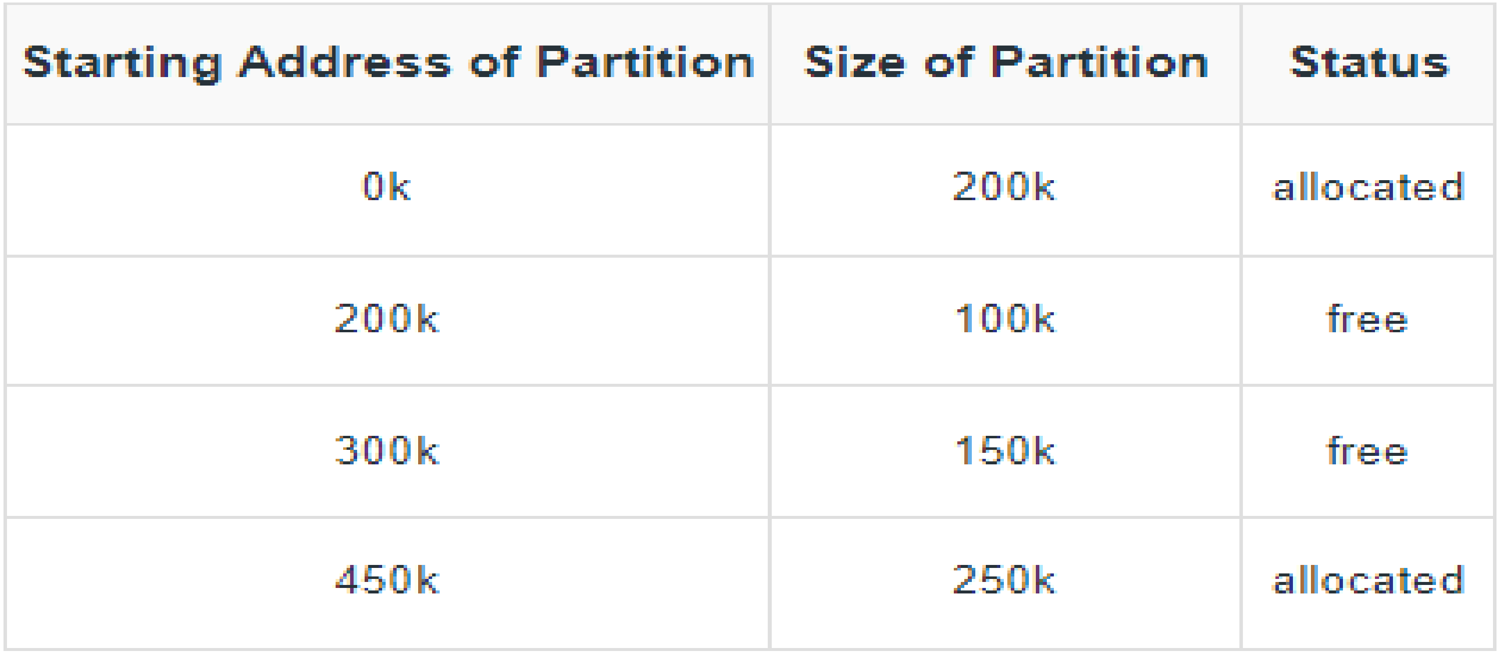
Example: As shown in fig. memory is partitioned into 5 regions the region is reserved for updating the system the remaining four partitions are for the user program.

Fixed Size Partitioning



Partition Table

- Once partitions are defined operating system keeps track of the status of memory partitions it is done through a data structure called a partition table.
- **Sample Partition Table**



Starting Address of Partition	Size of Partition	Status
0k	200k	allocated
200k	100k	free
300k	150k	free
450k	250k	allocated

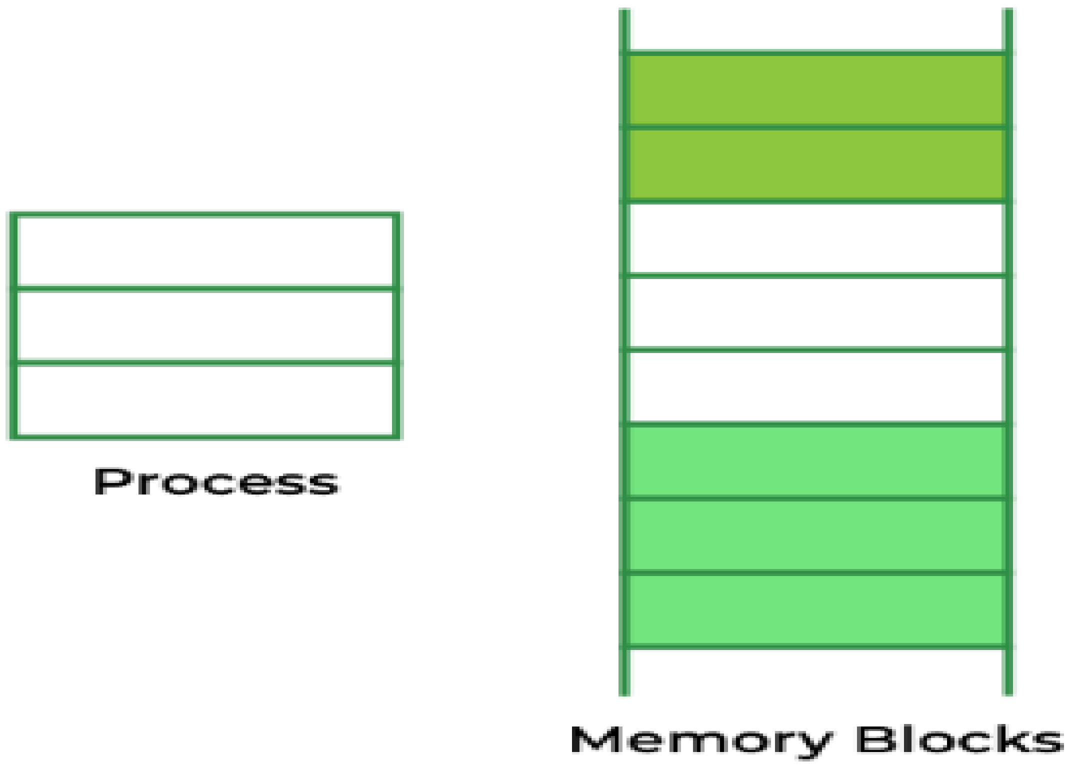
Logical vs Physical Address

- An address generated by the CPU is commonly referred to as a logical address. the address seen by the memory unit is known as the physical address. The logical address can be mapped to a physical address by hardware with the help of a base register this is known as dynamic relocation of memory references.

- **Contiguous Memory Allocation**

  The main memory should accommodate both the operating system and the different client processes. Therefore, the allocation of memory becomes an important task in the operating system. The memory is usually divided into two partitions: one for the resident operating system and one for the user processes. We normally need several user processes to reside in memory simultaneously. Therefore, we need to consider how to allocate available memory to the processes that are in the input queue waiting to be brought into memory. In adjacent memory allotment, each process is contained in a single contiguous segment of memory.

Contiguous Memory Allocation



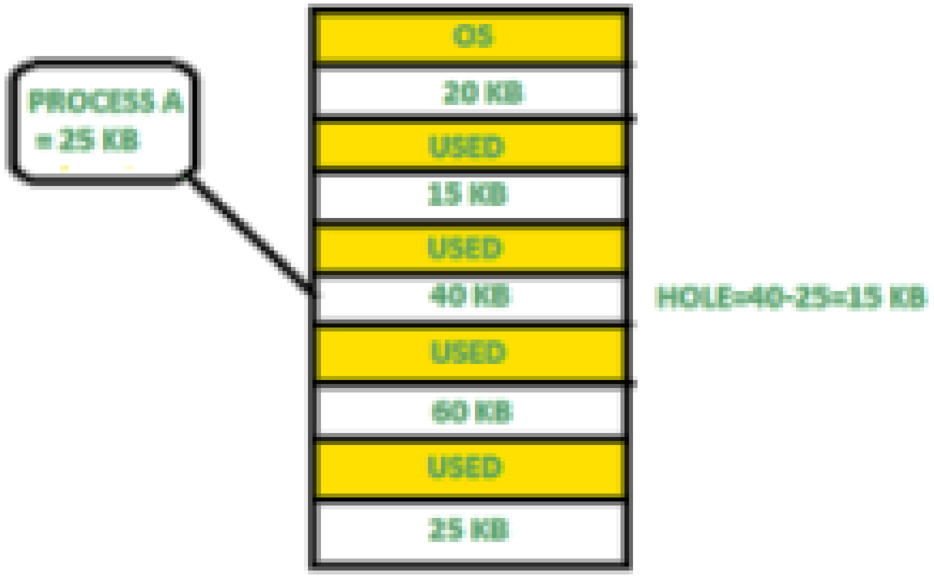
Contiguous Memory Allocation

Memory Allocation

- To gain proper memory utilization, memory allocation must be allocated efficient manner. One of the simplest methods for allocating memory is to divide memory into several fixed-sized partitions and each partition contains exactly one process. Thus, the degree of multiprogramming is obtained by the number of partitions.
- **Multiple partition allocation:** In this method, a process is selected from the input queue and loaded into the free partition. When the process terminates, the partition becomes available for other processes.
- **Fixed partition allocation:** In this method, the operating system maintains a table that indicates which parts of memory are available and which are occupied by processes. Initially, all memory is available for user processes and is considered one large block of available memory. This available memory is known as a "Hole". When the process arrives and needs memory, we search for a hole that is large enough to store this process. If the requirement is fulfilled then we allocate memory to process, otherwise keeping the rest available to satisfy future requests. While allocating a memory sometimes dynamic storage allocation problems occur, which concerns how to satisfy a request of size n from a list of free holes. There are some solutions to this problem:

First Fit

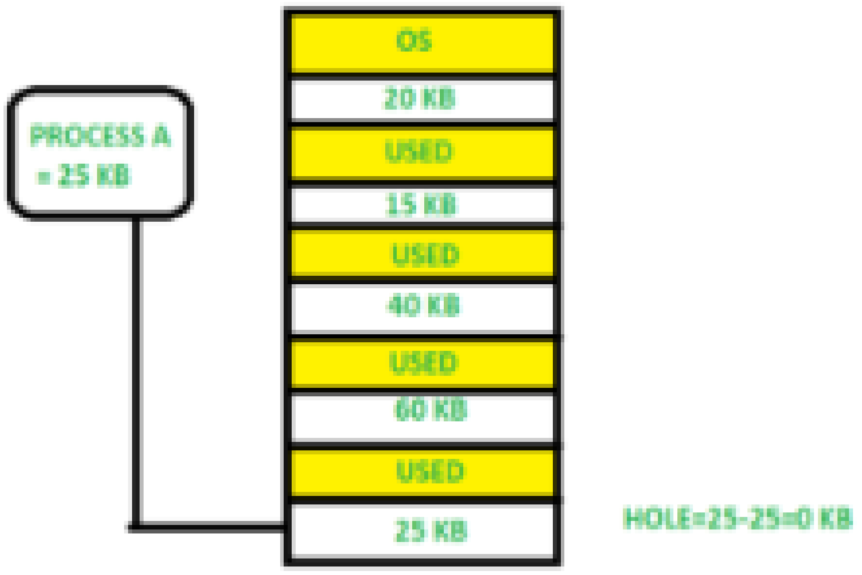
- In the First Fit, the first available free hole fulfil the requirement of the process allocated.



- Here, in this diagram, a 40 KB memory block is the first available free hole that can store process A (size of 25 KB), because the first two blocks did not have sufficient memory space.

Best Fit

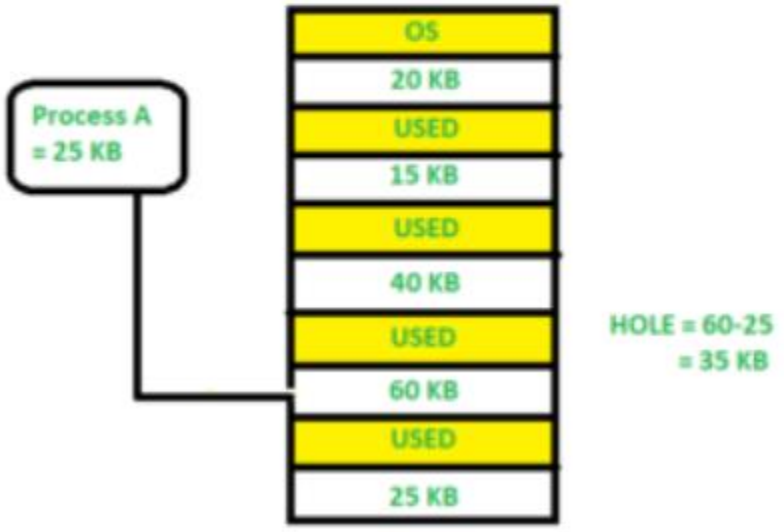
- In the Best Fit, allocate the smallest hole that is big enough to process requirements. For this, we search the entire list, unless the list is ordered by size.



- Here in this example, first, we traverse the complete list and find the last hole 25KB is the best suitable hole for Process A(size 25KB). In this method, memory utilization is maximum as compared to other memory allocation techniques.

Worst Fit

- In the Worst Fit, allocate the largest available hole to process. This method produces the largest leftover hole.



- Here in this example, Process A (Size 25 KB) is allocated to the largest available memory block which is 60KB. Inefficient memory utilization is a major issue in the worst fit.

Fragmentation

- Fragmentation is defined as when the process is loaded and removed after execution from memory, it creates a small free hole. These holes can not be assigned to new processes because holes are not combined or do not fulfill the memory requirement of the process. To achieve a degree of multiprogramming, we must reduce the waste of memory or fragmentation problems. In the operating systems two types of fragmentation:

1. **Internal fragmentation:** Internal fragmentation occurs when memory blocks are allocated to the process more than their requested size. Due to this some unused space is left over and creating an internal fragmentation problem.

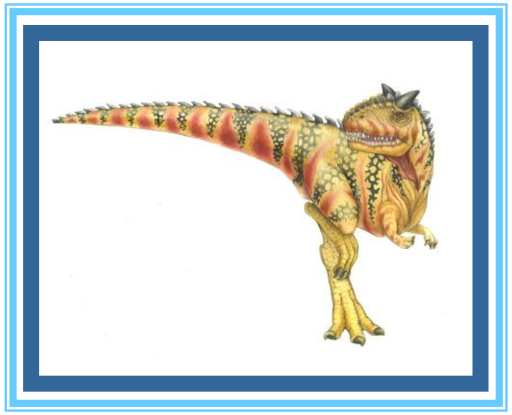
   Example: Suppose there is a fixed partitioning used for memory allocation and the different sizes of blocks 3MB, 6MB, and 7MB space in memory. Now a new process p4 of size 2MB comes and demands a block of memory. It gets a memory block of 3MB but 1MB block of memory is a waste, and it can not be allocated to other processes too. This is called internal fragmentation.

Fragmentation

2. **External fragmentation:** In External Fragmentation, we have a free memory block, but we can not assign it to a process because blocks are not contiguous. **Example:** Suppose (consider the above example) three processes p1, p2, and p3 come with sizes 2MB, 4MB, and 7MB respectively. Now they get memory blocks of size 3MB, 6MB, and 7MB allocated respectively. After allocating the process p1 process and the p2 process left 1MB and 2MB. Suppose a new process p4 comes and demands a 3MB block of memory, which is available, but we can not assign it because free memory space is not contiguous. This is called external fragmentation.

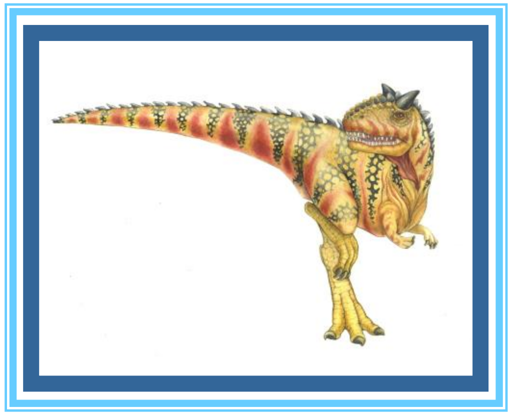
- Both the first-fit and best-fit systems for memory allocation are affected by external fragmentation. To overcome the external fragmentation problem Compaction is used. In the compaction technique, all free memory space combines and makes one large block. So, this space can be used by other processes effectively.
- Another possible solution to the external fragmentation is to allow the logical address space of the processes to be noncontiguous, thus permitting a process to be allocated physical memory wherever the latter is available.

Any Questions ?



Week 10
Operating System Concepts

Muhammad Daniyal Liaquat



Topics

1. **Paging**
2. **Segmentation**

1. Paging

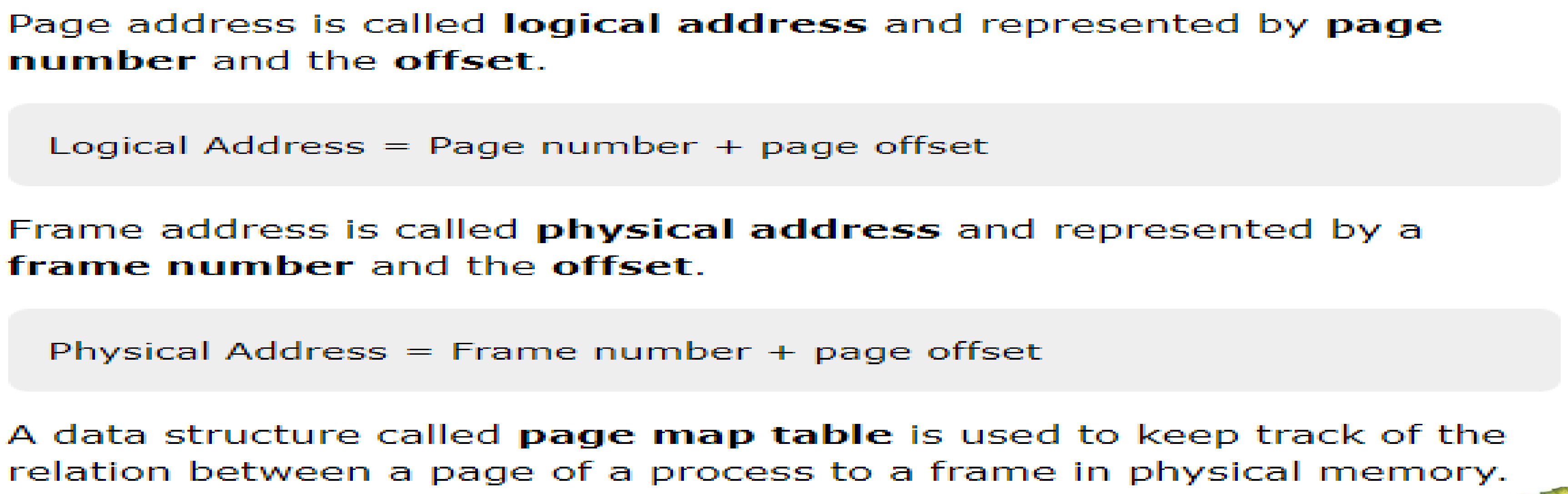
What is Paging?

- Paging is a technique that eliminates the requirements of contiguous allocation of main memory. In this, the main memory is divided into fixed-size blocks of physical memory called frames. The size of a frame should be kept the same as that of a page to maximize the main memory and avoid external fragmentation.
- A computer system can address and utilize more memory than the size of the memory present in the computer hardware. This extra memory is referred to as virtual memory. Virtual memory is a part of secondary memory that the computer system uses as primary memory(RAM). Paging has a vital role in the implementation of virtual memory.
- Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Page table

- A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between the virtual address and physical addresses.
- The virtual address is also known as the Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

Address Translation



Page address is called **logical address** and represented by **page number** and the **offset**.

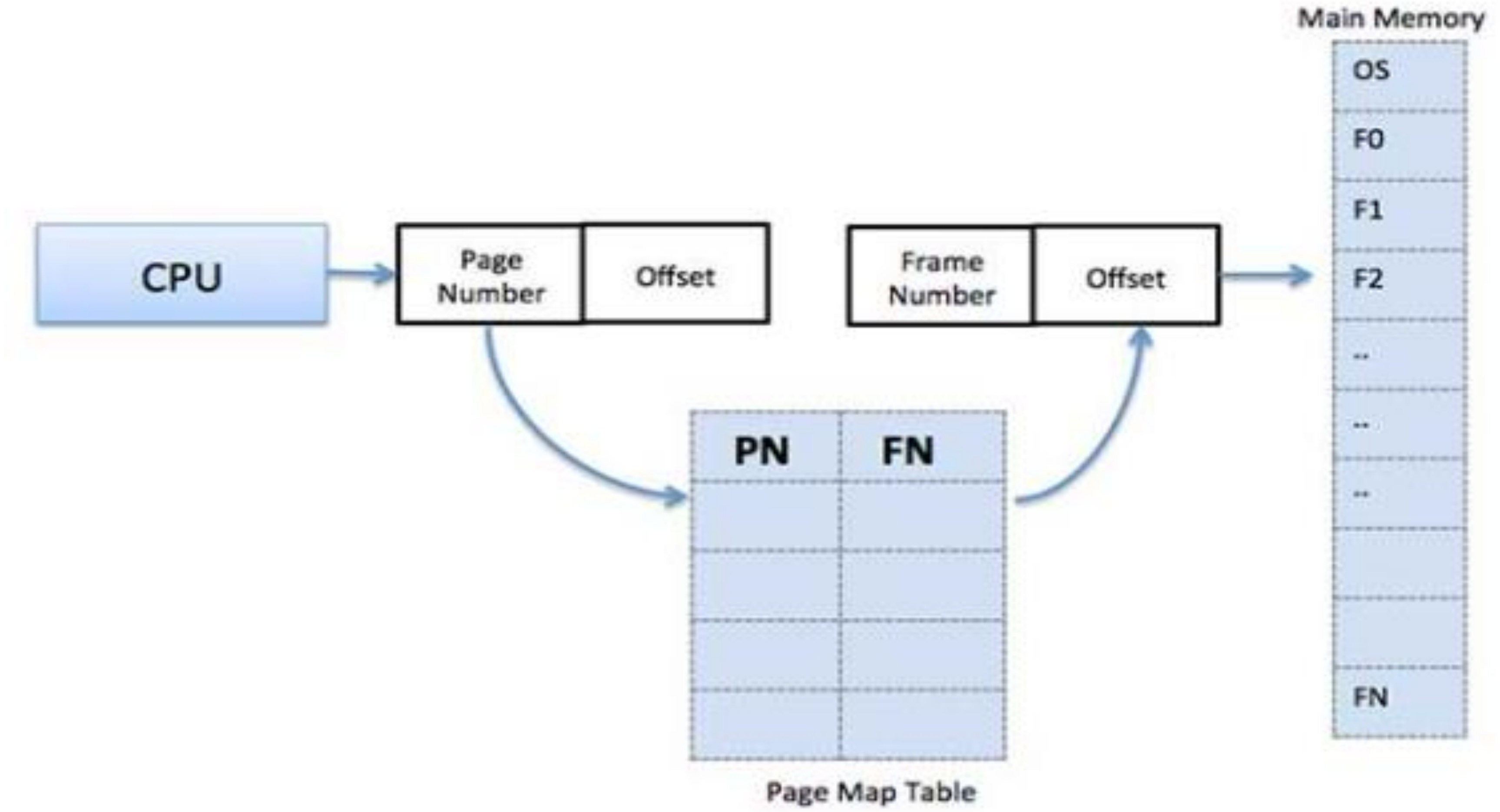
```
Logical Address = Page number + page offset
```

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

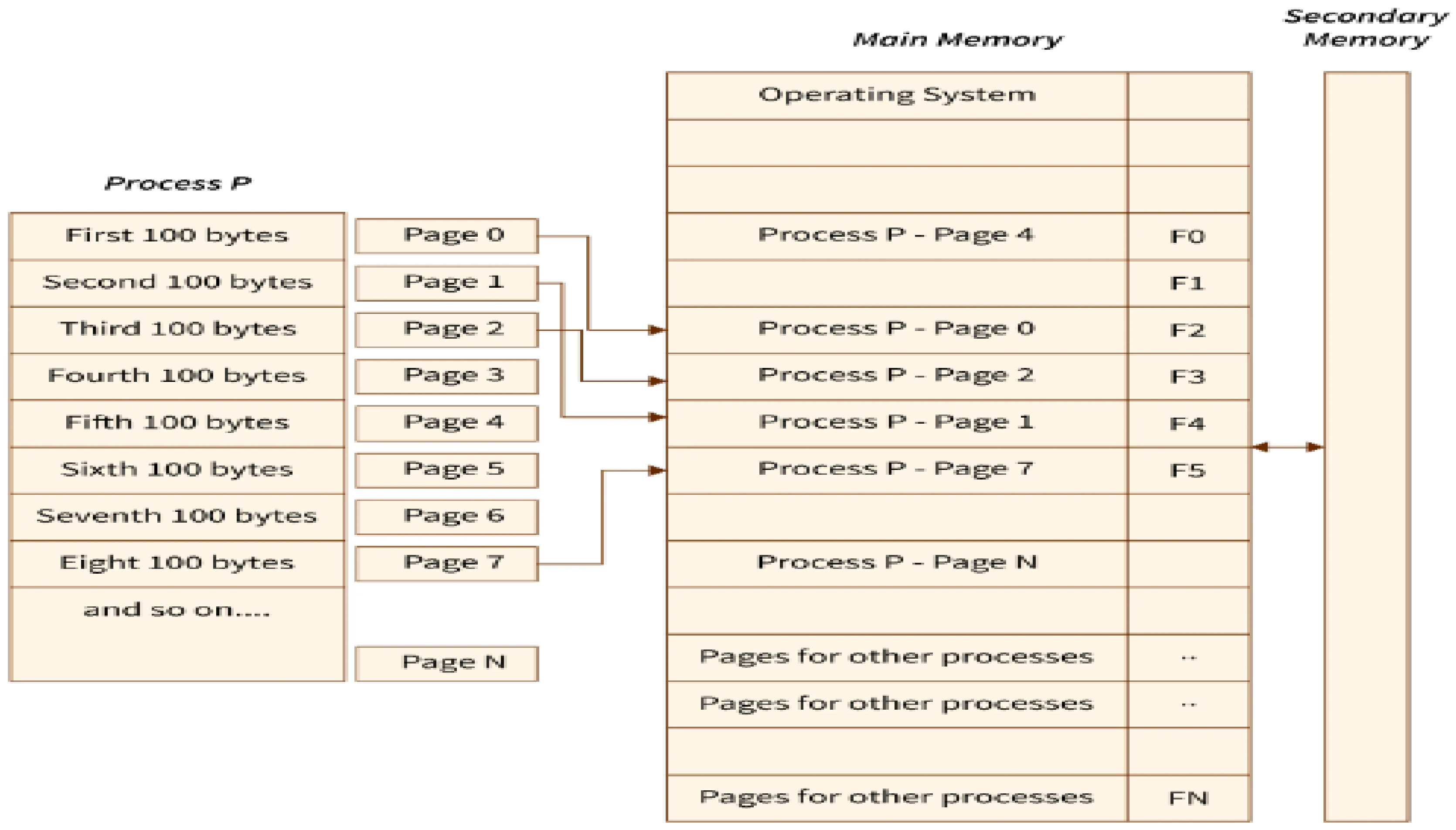
```
Physical Address = Frame number + page offset
```

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.

Address Translation



What is Paging?



PAGING

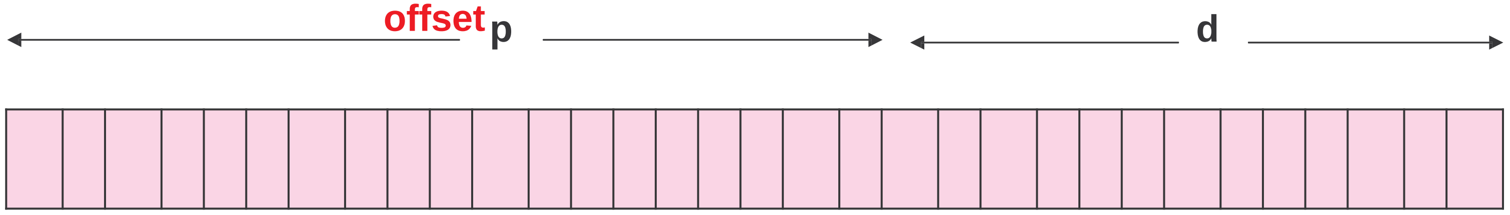
Address Translation Scheme

Address generated by the CPU is divided into:

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

•*Page offset (d)* – combined with base address to define the physical memory address that is sent to the memory unit.

4096 bytes = 2^12 – it requires 12 bits to contain the Page offset



PAGING

- A 32 bit machine can address 4 gigabytes which is 4 million pages (at 1024 bytes/page). WHO says how big a page is, anyway?
- Could use dedicated registers (OK only with small tables.)

•Could use a register pointing to table in memory (slow access.)

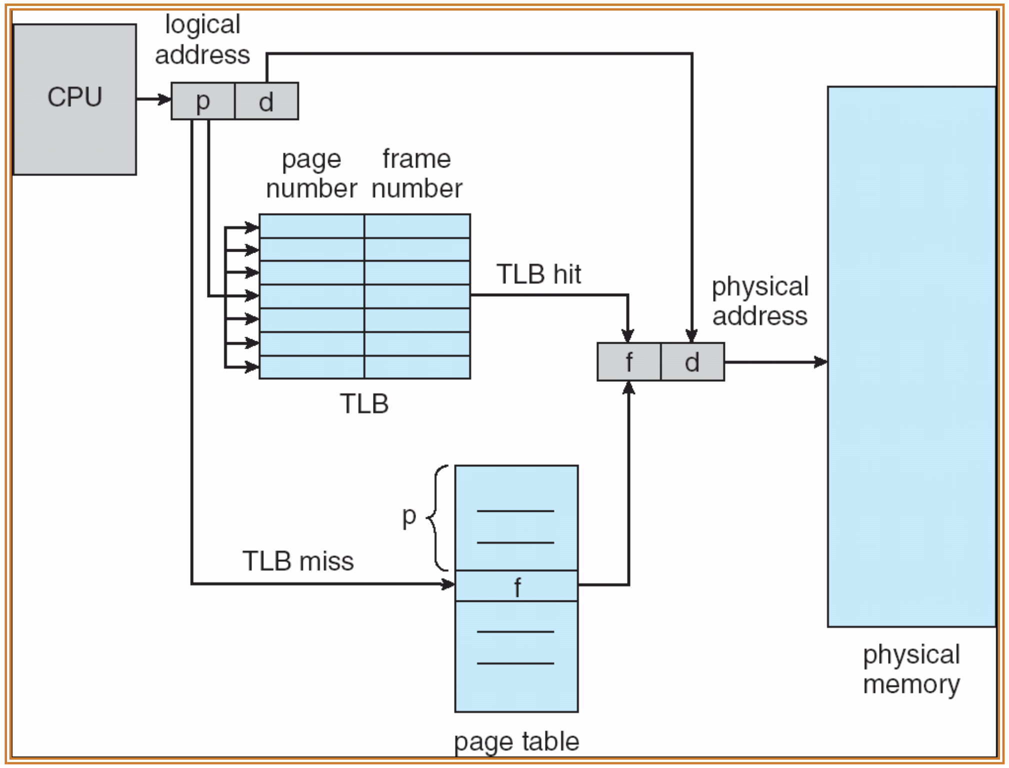
•Cache or associative memory

•(TLB = Translation Lookaside Buffer):

• simultaneous search is fast and uses only a few registers.

IMPLEMENTATION OF THE PAGE TABLE

TLB = Translation Lookaside Buffer



PAGING

IMPLEMENTATION OF THE PAGE TABLE

Issues include:

key and value
hit rate 90 - 98% with 100 registers
add entry if not found

Relevant times:

2 nanoseconds to search associative memory – the TLB.
20 nanoseconds to access processor cache and bring it into TLB for next time.

Calculate time of access:

hit = 1 search + 1 memory reference
miss = 1 search + 1 mem reference(of page table) + 1 mem reference.

PAGING

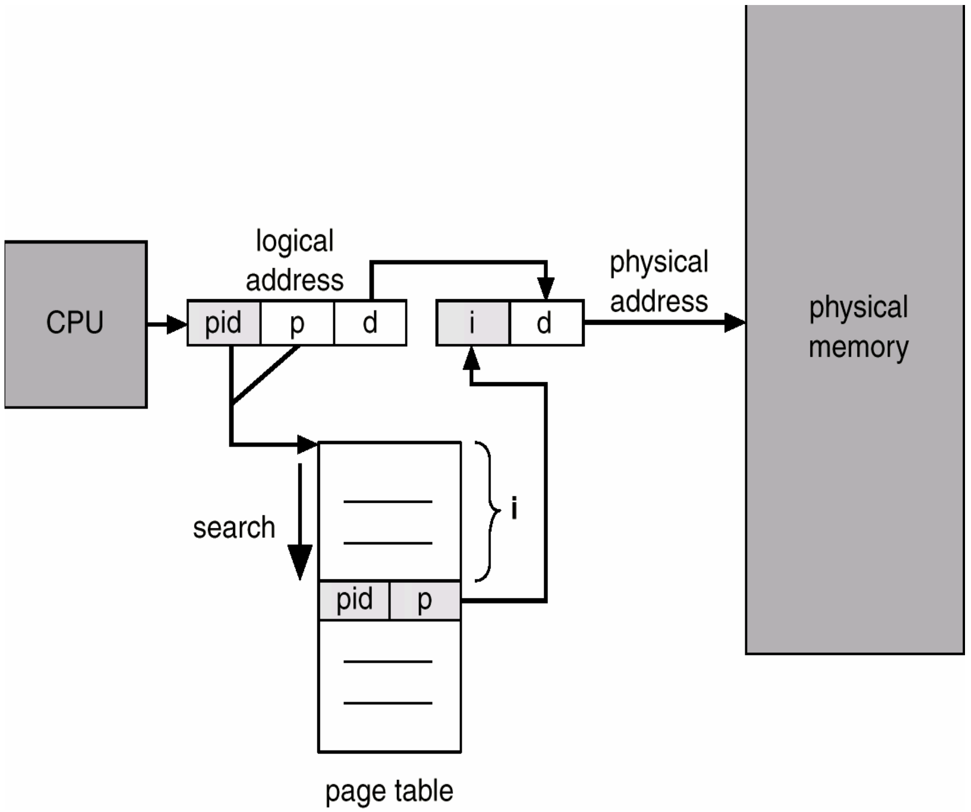
INVERTED PAGE TABLE:

One entry for each real page of memory.

Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.

Essential when you need to do work on the page and must find out what process owns it.

Use hash table to limit the search to one - or at most a few - page table entries.



PAGING

PROTECTION:

•Bits associated with page tables.
•Can have read, write, execute, valid bits.
•Valid bit says page isn't in address space.
•Write to a write-protected page causes a fault. Touching an invalid page causes a fault.

ADDRESS MAPPING:

•Allows physical memory larger than logical memory.
•Useful on 32 bit machines with more than 32-bit addressable words of memory.
•The operating system keeps a frame containing descriptions of physical pages; if allocated, then to which logical page in which process.

Advantages and Disadvantages of Paging

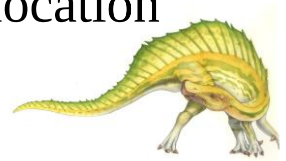
Here is a list of advantages and disadvantages of paging.

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

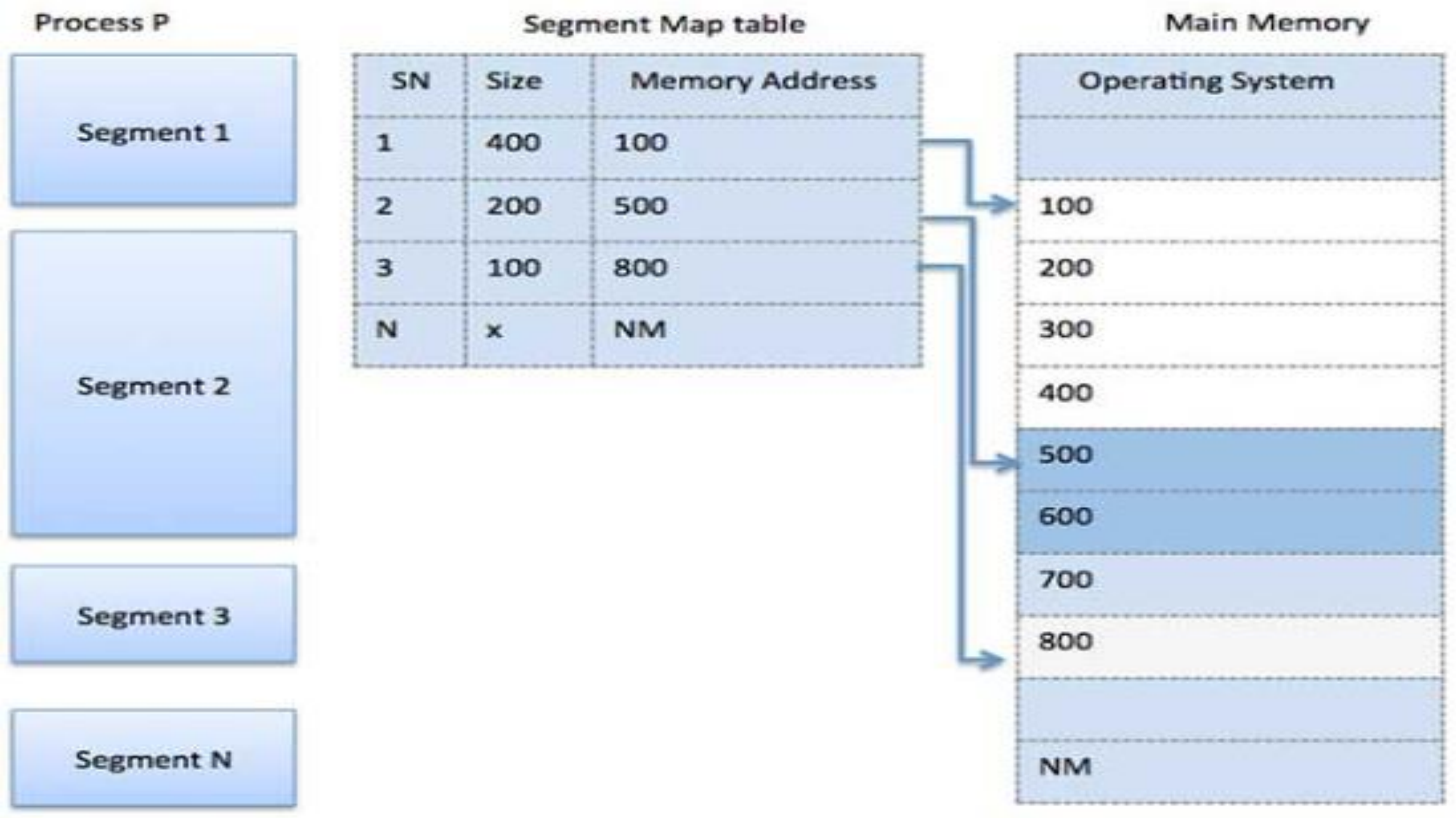
2. Segmentation

What is Segmentation?

- Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.
- When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.
- Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.
- A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



What is Segmentation?



SN	Size	Memory Address
1	400	100
2	200	500
3	100	800
N	x	NM

SEGMENTATION

USER'S VIEW OF MEMORY

A programmer views a process consisting of unordered segments with various purposes. This view is more useful than thinking of a linear array of words. We really don't care at what address a segment is located.

Typical segments include

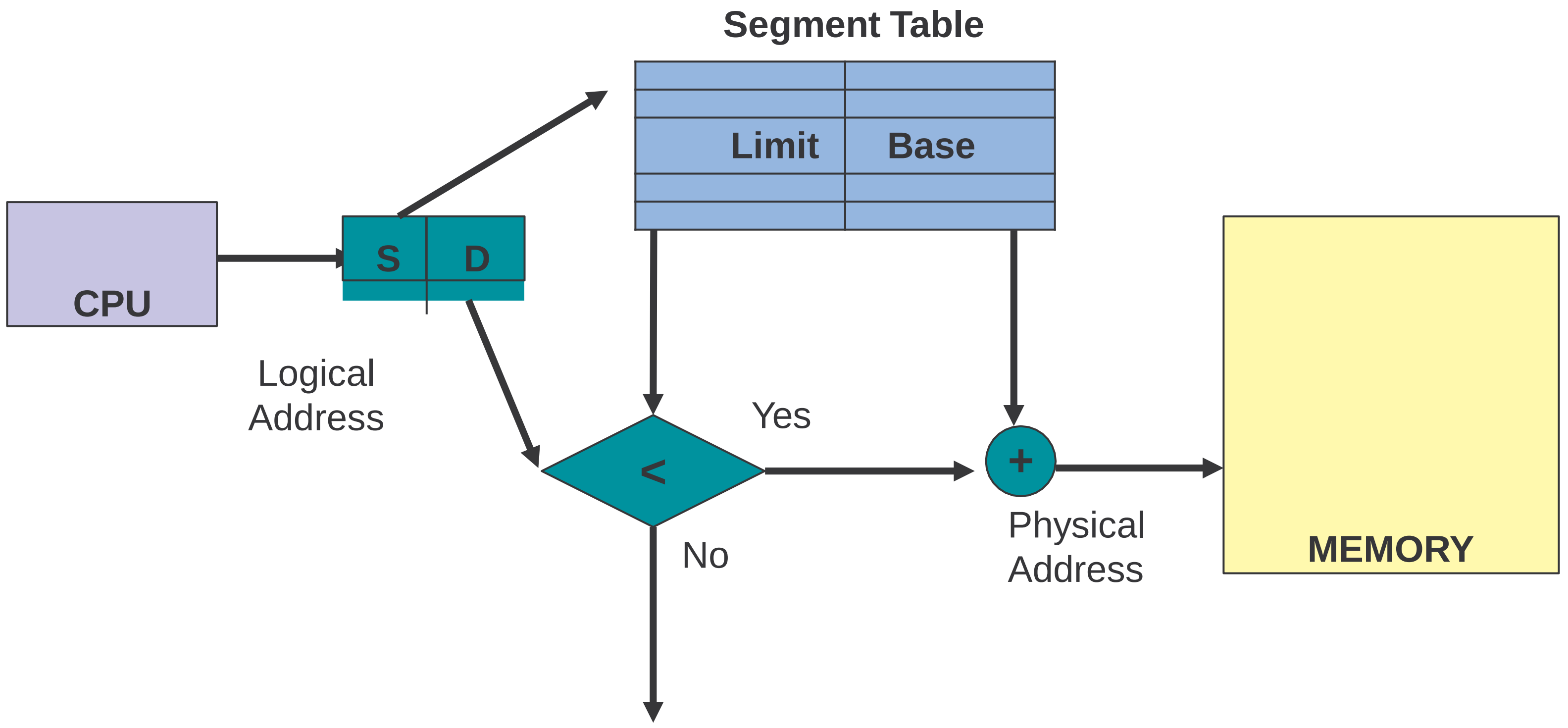
- global variables procedure call stack code for each function
- local variables for each large data structures

Logical address = segment name (number) + offset Memory is addressed by both

segment and offset.

SEGMENTATION

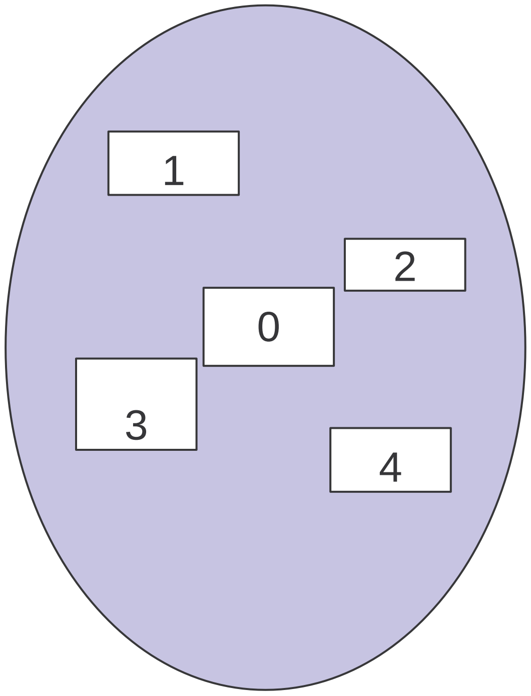
HARDWARE -- Must map a dyad (segment / offset) into one-dimensional address.



SEGMENTATION

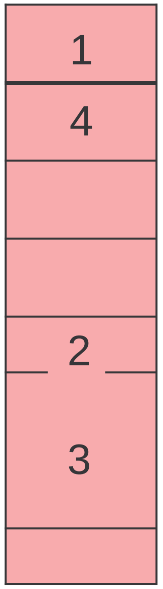
HARDWARE

base / limit pairs in a segment table.



Logical Address Space

	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700



Physical Memory

SEGMENTATION

PROTECTION AND SHARING

Addresses are associated with a logical unit (like data, code, etc.) so protection is easy.

Can do bounds checking on arrays

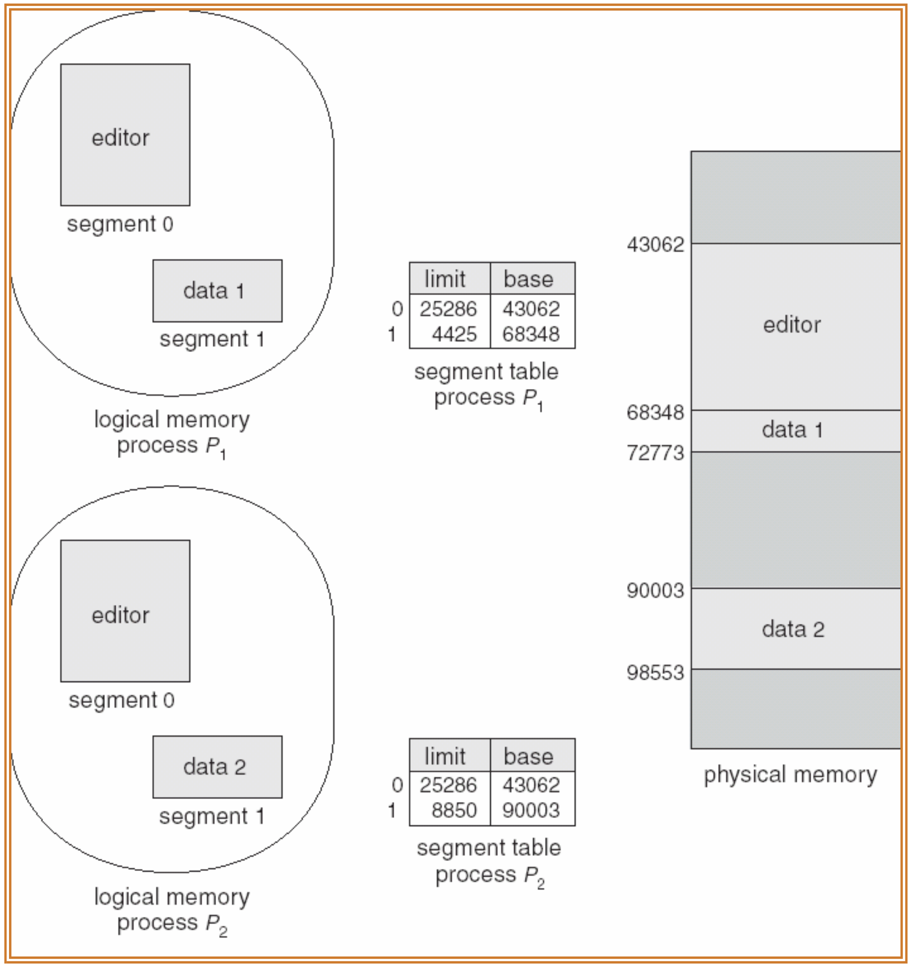
Sharing specified at a logical level, a segment has an attribute called "shareable".

Can share some code but not all - for instance a common library of subroutines.

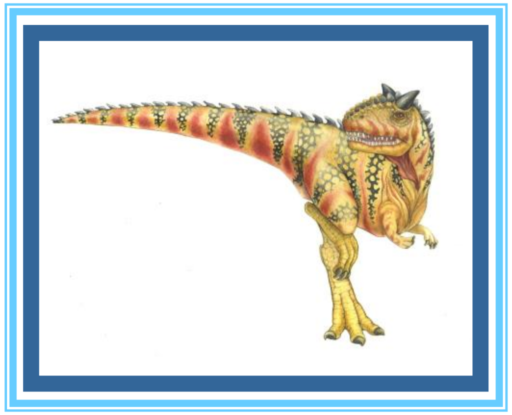
FRAGMENTATION

Use variable allocation since segment lengths vary.

Again have issue of fragmentation; Smaller segments means less fragmentation. Can use compaction since segments are relocatable.

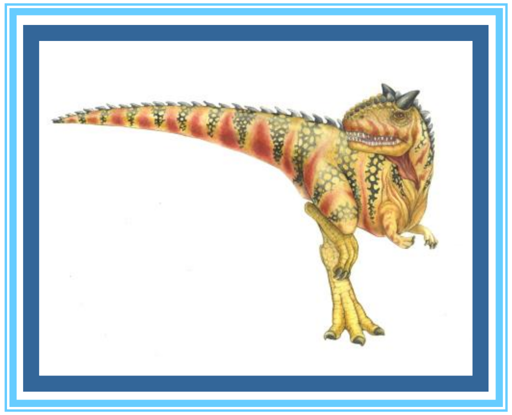


Any Questions ?



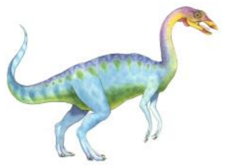
Week 11
Operating System Concepts

Muhammad Daniyal Liaquat



Topics

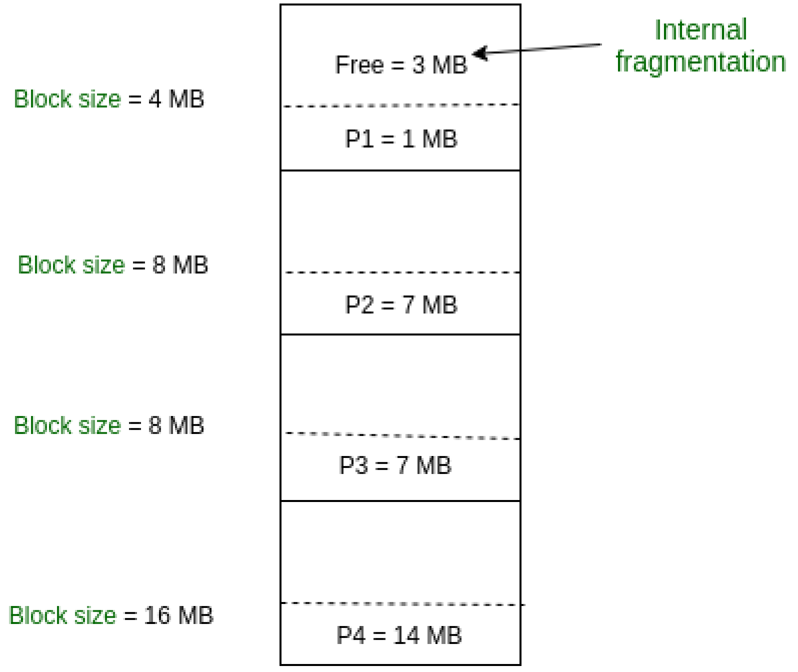
1. **Fixed and Dynamic Partitioning**
2. **Virtual Memory**



1. Fixed (Static) & Dynamic (Variable) Partitioning

Fixed Partitioning

- This is the oldest and simplest technique used to put more than one process in the main memory. In this partitioning, the number of partitions (non-overlapping) in RAM is **fixed but the size** of each partition may or **may not be the same**. As it is a **contiguous** allocation, hence no spanning is allowed. Here partitions are made before execution or during system configure.



Fixed size partition

Fixed Partitioning

- As illustrated in above figure, first process is only consuming 1MB out of 4MB in the main memory.
  Hence, Internal Fragmentation in first block is (4-1) = 3MB.
  Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.
- Suppose process P5 of size 7MB comes. But this process cannot be accommodated in spite of available free space because of contiguous allocation (as spanning is not allowed). Hence, 7MB becomes part of External Fragmentation.

Advantages of Fixed Partitioning

- **Easy to implement:** The algorithms needed to implement Fixed Partitioning are straightforward and easy to implement.
- **Low overhead:** Fixed Partitioning requires minimal overhead, which makes it ideal for systems with limited resources.
- **Predictable:** Fixed Partitioning ensures a predictable amount of memory for each process.
- **No external fragmentation:** Fixed Partitioning eliminates the problem of external fragmentation.
- **Suitable for systems with a fixed number of processes:** Fixed Partitioning is well-suited for systems with a fixed number of processes and known memory requirements.
- **Prevents processes from interfering with each other:** Fixed Partitioning ensures that processes do not interfere with each other's memory space.

Advantages of Fixed Partitioning

- **Efficient use of memory:** Fixed Partitioning ensures that memory is used efficiently by allocating it to fixed-sized partitions.
- **Good for batch processing:** Fixed Partitioning is ideal for batch processing environments where the number of processes is fixed.
- **Better control over memory allocation:** Fixed Partitioning gives the operating system better control over the allocation of memory.
- **Easy to debug:** Fixed Partitioning is easy to debug since the size and location of each process are predetermined.

Disadvantages of Fixed Partitioning

1. **Internal Fragmentation:**
   Main memory use is inefficient. Any program, no matter how small, occupies an entire partition. This can cause internal fragmentation.

2. **External Fragmentation:**
   The total unused space (as stated above) of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form (as spanning is not allowed)

3. **Limit process size:**
   Process of size greater than the size of the partition in Main Memory cannot be accommodated. The partition size cannot be varied according to the size of the incoming process size. Hence, the process size of 32MB in the above-stated example is invalid.

Disadvantages of Fixed Partitioning

Limitation on Degree of Multiprogramming: Partitions in Main Memory are made before execution or during system configure. Main Memory is divided into a fixed number of partitions. Suppose if there are

$n1$

partitions in RAM and

$n2$

are the number of processes, then

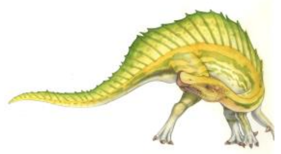
$n2 <= n1$

condition must be fulfilled. Number of processes greater than the number of partitions in RAM is invalid in Fixed Partitioning.

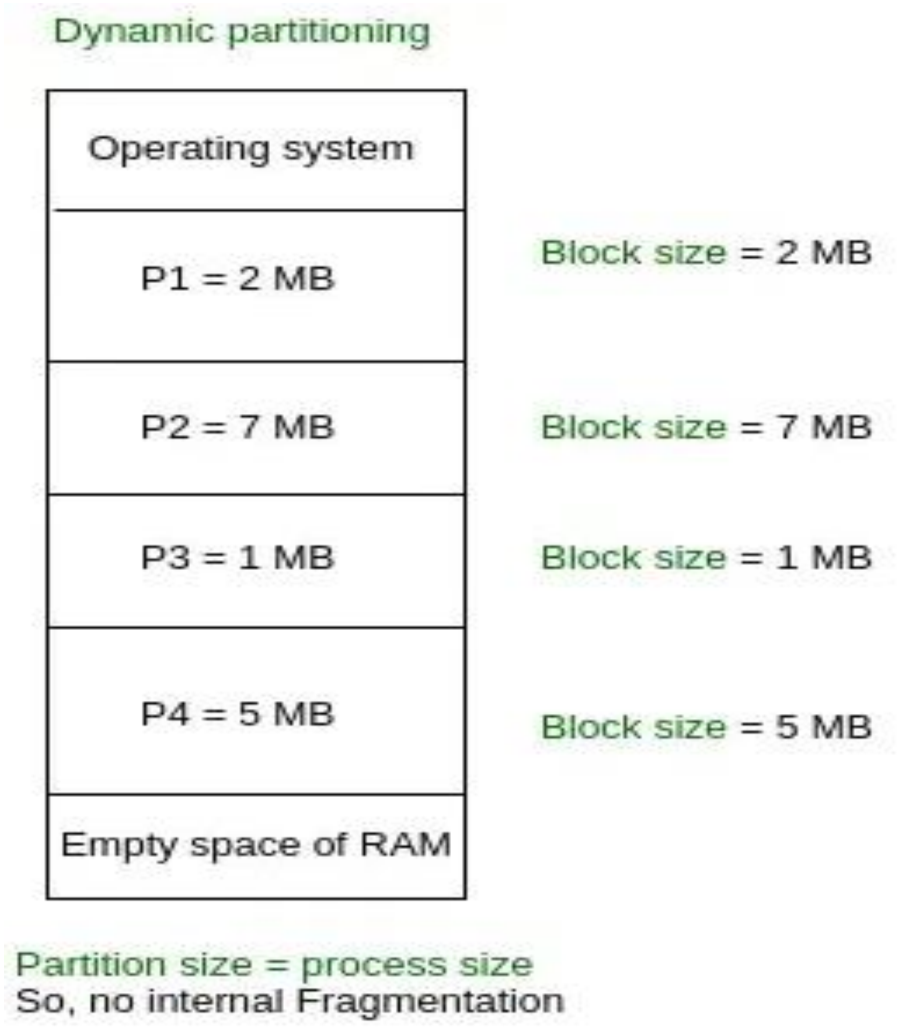
Dynamic Partitioning

It is a part of the Contiguous allocation technique. It is used to alleviate the problem faced by Fixed Partitioning. In contrast with fixed partitioning, partitions are not made before the execution or during system configuration. Various **features** associated with variable Partitioning-

- Initially, RAM is empty and partitions are made during the run-time according to the process's need instead of partitioning during system configuration.
- The size of the partition will be equal to the incoming process.
- The partition size varies according to the need of the process so that internal fragmentation can be avoided to ensure efficient utilization of RAM.
- The number of partitions in RAM is not fixed and depends on the number of incoming processes and the Main Memory's size.



Dynamic Partitioning



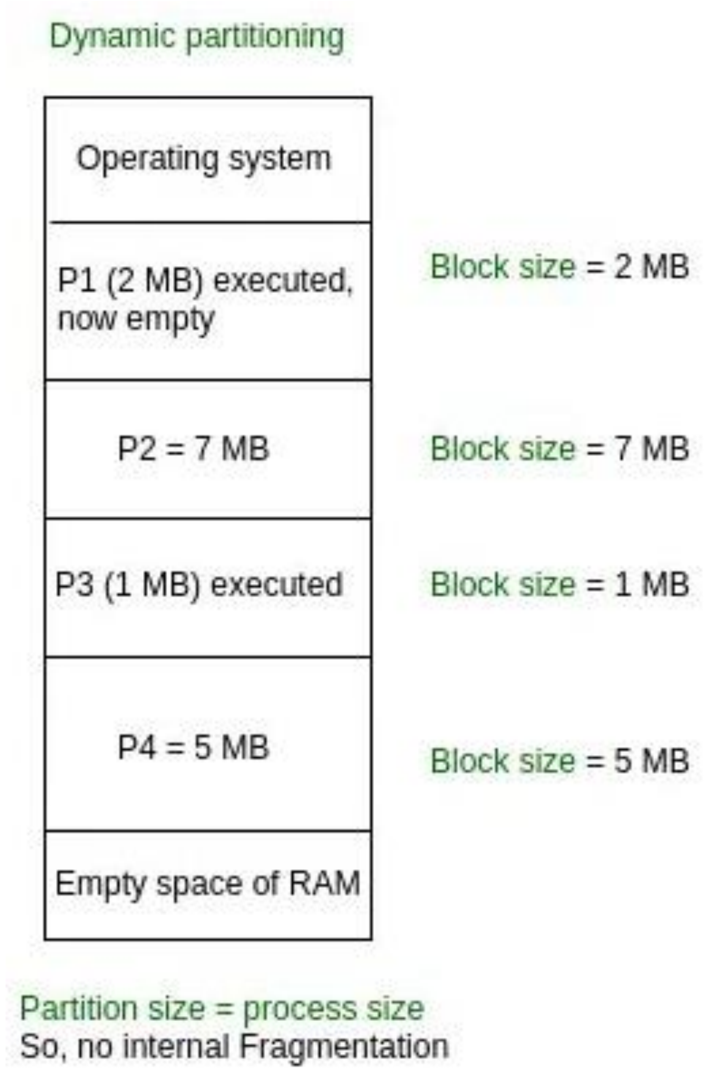
Advantages of Dynamic Partitioning

- **No Internal Fragmentation:** In variable Partitioning, space in the main memory is allocated strictly according to the need of the process, hence there is no case of internal fragmentation. There will be no unused space left in the partition.
- **No restriction on the Degree of Multiprogramming:** More processes can be accommodated due to the absence of internal fragmentation. A process can be loaded until the memory is empty.
- **No Limitation on the Size of the Process:** In Fixed partitioning, the process with a size greater than the size of the largest partition could not be loaded and the process can not be divided as it is invalid in the contiguous allocation technique. Here, In variable partitioning, the process size can't be restricted since the partition size is decided according to the process size.

Disadvantages of Dynamic Partitioning

- **Difficult Implementation:** Implementing variable Partitioning is difficult as compared to Fixed Partitioning as it involves the allocation of memory during run-time rather than during system configuration.
- **External Fragmentation:** There will be external fragmentation despite the absence of internal fragmentation. For example, suppose in the above example- process P1(2MB) and process P3(1MB) completed their execution. Hence two spaces are left i.e. 2MB and 1MB. Let’s suppose process P5 of size 3MB comes. The space in memory cannot be allocated as no spanning is allowed in contiguous allocation. The rule says that the process must be continuously present in the main memory to get executed. Hence it results in External Fragmentation.

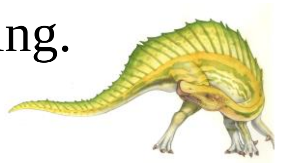
Disadvantages of Dynamic Partitioning



Now P5 of size 3 MB cannot be accommodated despite the required available space because in contiguous no spanning is allowed.

Key Points On Variable (or Dynamic) Partitioning in Operating Systems

- Variable (or dynamic) partitioning is a memory allocation technique that allows memory partitions to be created and resized dynamically as needed.
- The operating system maintains a table of free memory blocks or holes, each of which represents a potential partition. When a process requests memory, the operating system searches the table for a suitable hole that can accommodate the requested amount of memory.
- Dynamic partitioning reduces internal fragmentation by allocating memory more efficiently, allows multiple processes to share the same memory space, and is flexible in accommodating processes with varying memory requirements.
- However, dynamic partitioning can also lead to external fragmentation and requires more complex memory management algorithms, which can make it slower than fixed partitioning.
- Understanding dynamic partitioning is essential for operating system design and implementation, as well as for system-level programming.



2. Virtual Memory

Virtual Memory

Virtual Memory is a storage allocation scheme in which secondary memory can be addressed as though it were part of the main memory. The addresses a program may use to reference memory are distinguished from the addresses the memory system uses to identify physical storage sites and program-generated addresses are translated automatically to the corresponding machine addresses.

- The size of virtual storage is limited by the addressing scheme of the computer system and the amount of secondary memory available not by the actual number of main storage locations.
- It is a technique that is implemented using both hardware and software. It maps memory addresses used by a program, called virtual addresses, into physical addresses in computer memory.

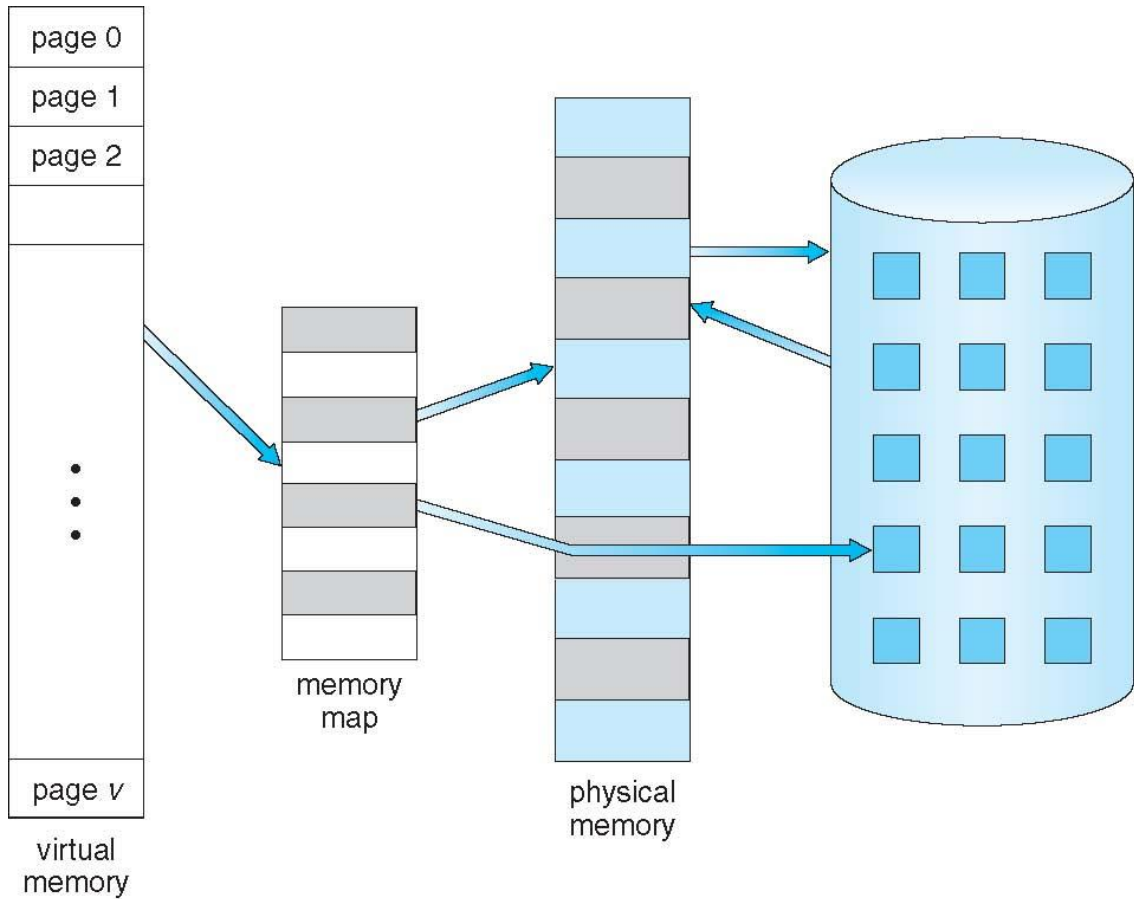
Virtual Memory

- All memory references within a process are logical addresses that are dynamically translated into physical addresses at run time. This means that a process can be swapped in and out of the main memory such that it occupies different places in the main memory at different times during the course of execution.
- A process may be broken into a number of pieces and these pieces need not be continuously located in the main memory during execution. The combination of dynamic run-time address translation and the use of a page or segment table permits this.
- If these characteristics are present then, it is not necessary that all the pages or segments are present in the main memory during execution. This means that the required pages need to be loaded into memory whenever required. Virtual memory is implemented using Demand Paging or Demand Segmentation.

Virtual Memory

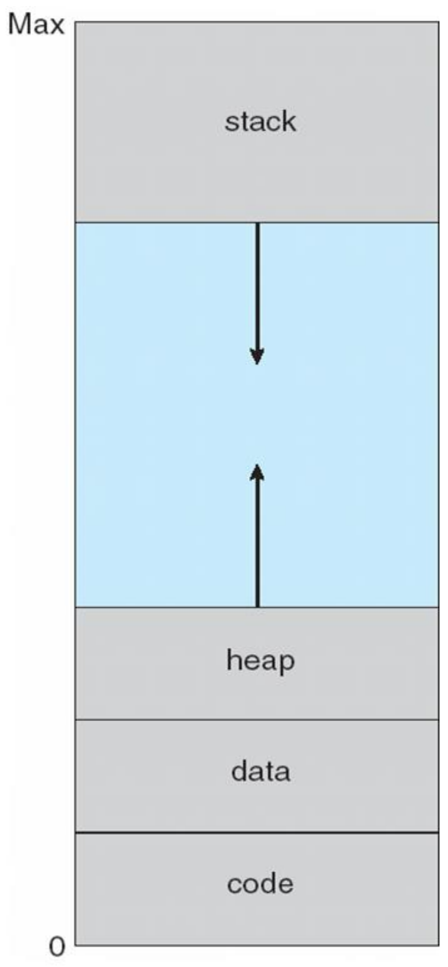
- **Virtual memory** – separation of user logical memory from physical memory
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes
  - Allows for more efficient process creation
  - More programs running concurrently
  - Less I/O needed to load or swap processes
- **Virtual address space** – logical view of how process is stored in memory
  - Usually start at address 0, contiguous addresses until end of space
  - Meanwhile, physical memory organized in page frames
  - MMU must map logical to physical
- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

Virtual Memory That is Larger Than Physical Memory

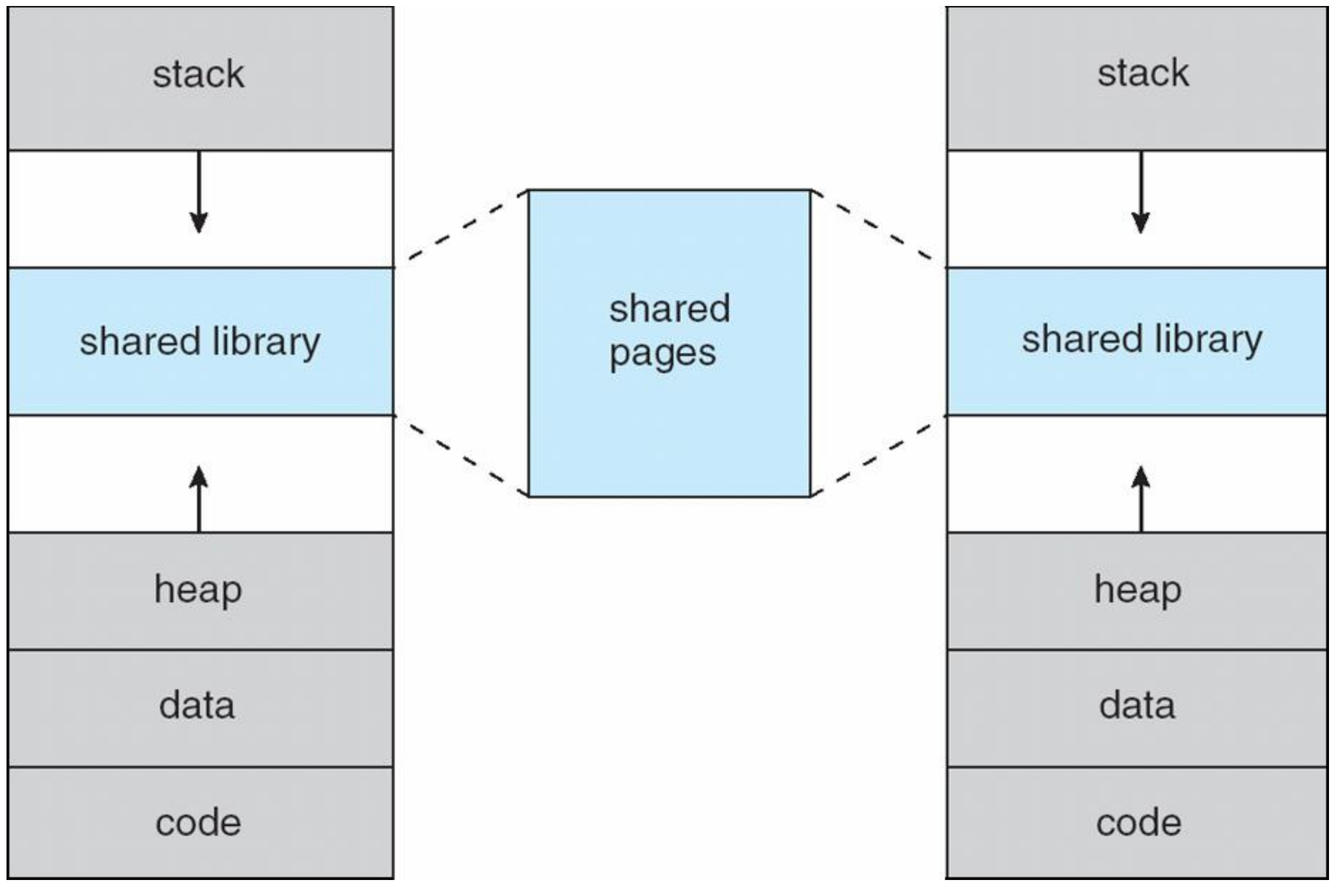


Virtual-address Space

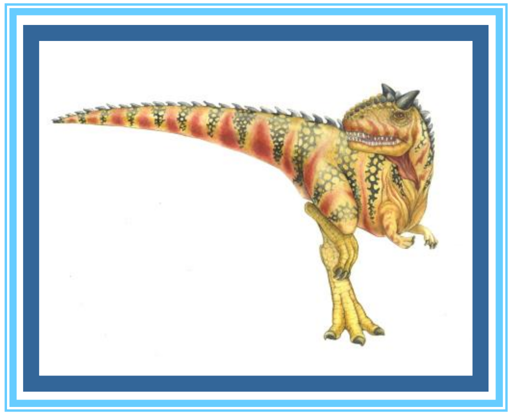
- Usually design logical address space for stack to start at Max logical address and grow “down” while heap grows “up”
  - Maximizes address space use
  - Unused address space between the two is hole
    - No physical memory needed until heap or stack grows to a given new page
- Enables **sparse** address spaces with holes left for growth, dynamically linked libraries, etc
- System libraries shared via mapping into virtual address space
- Shared memory by mapping pages read-write into virtual address space
- Pages can be shared during `fork()`, speeding process creation



Shared Library Using Virtual Memory



Any Questions ?



Week 12
Operating System Concepts

Page Replacement Algorithms

FIFO, NRU, LRU, NFU…

Muhammad Daniyal Liaquat

What is page replacement?

- When memory located in secondary memory is needed, it can be retrieved back to main memory.
- Process of storing data from main memory to secondary memory ->**swapping out**
- Retrieving data back to main memory ->**swapping in**

Basic Page Replacement

1. Find the location of the desired page on disk
2. Find a free frame:
   - If there is a free frame, use it
   - If there is no free frame, use a page replacement algorithm to select a **victim frame**
     - Write victim frame to disk if dirty
3. Bring the desired page into the (newly) free frame; update the page and frame tables
4. Continue the process by restarting the instruction that caused the trap

Note now potentially 2 page transfers for page fault – increasing EAT

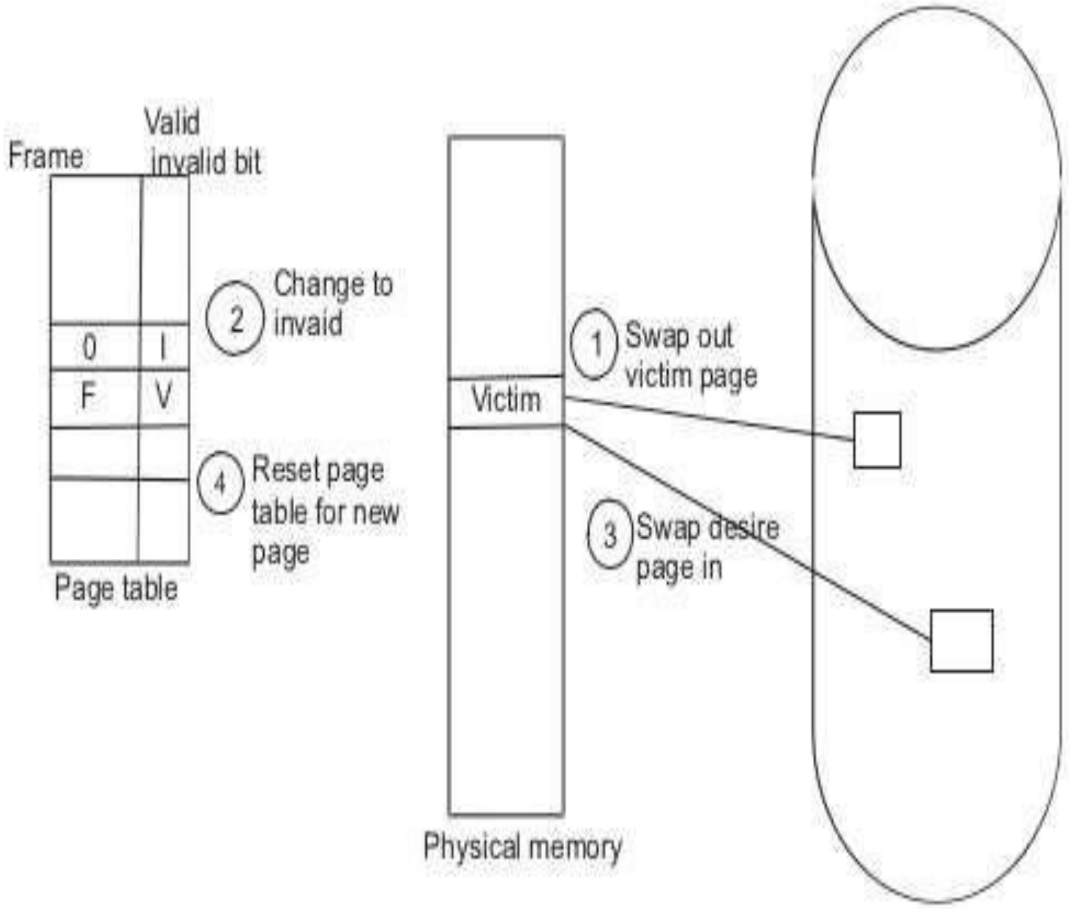


Fig: Page Replacement

What are Page Replacement Algorithms?

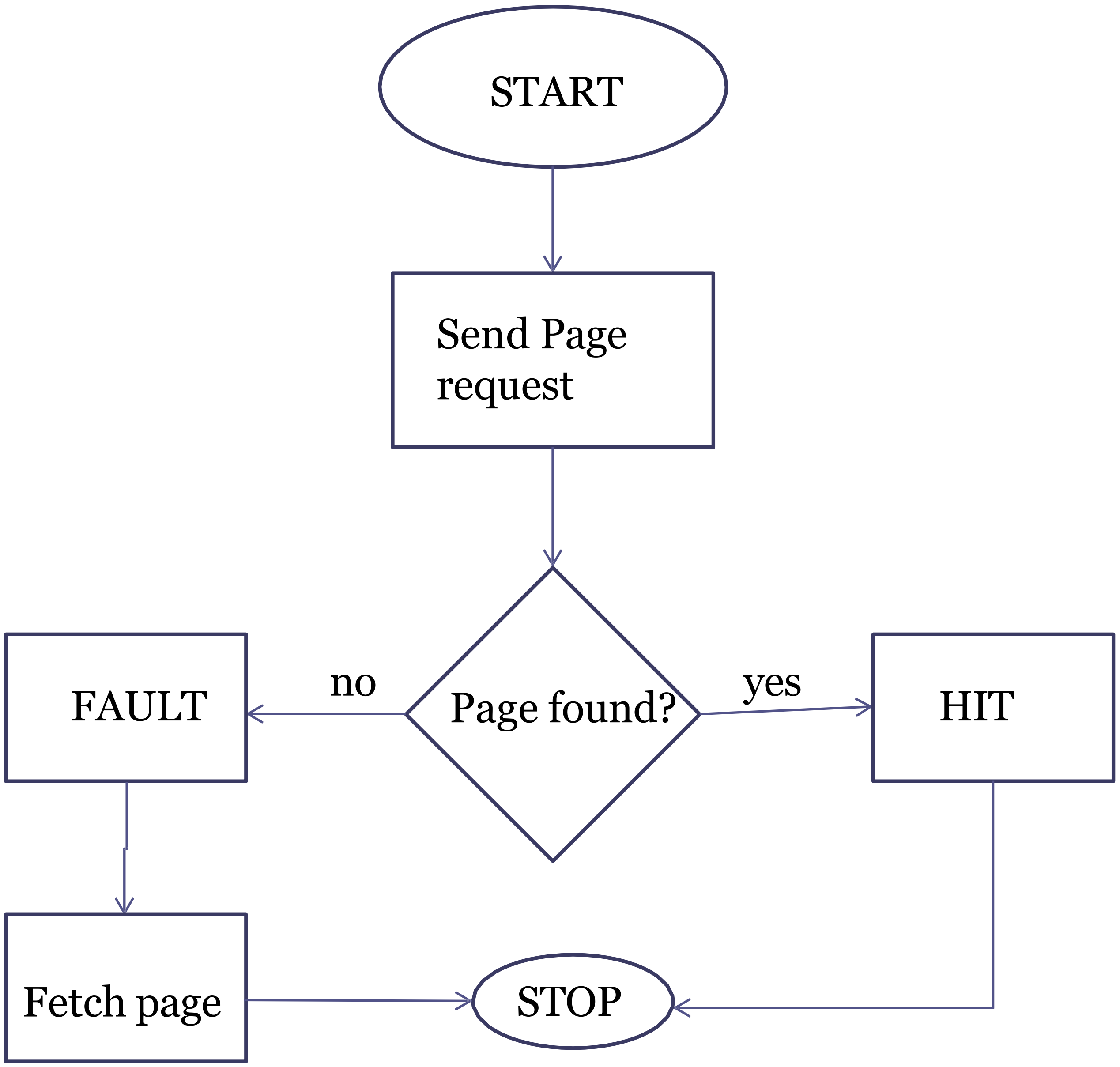
- Deals with which pages need to be swapped out and which are the ones that need to be swapped in
- The efficiency lies in the least time that is wasted for a page to be paged in

Types

- Local Page Replacement Strategy
- Global Page Replacement Strategy

Why we need a page replacement algorithm?

- The main goal of page replacement algorithms is to provide lowest page fault rate.


START
Send Page request
Page found?
no
yes
FAULT
HIT
Fetch page
STOP

Page and Frame Replacement Algorithms

Frame-allocation algorithm determines

- How many frames to give each process
- Which frames to replace

Page-replacement algorithm

- Want lowest page-fault rate on both first access and re-access

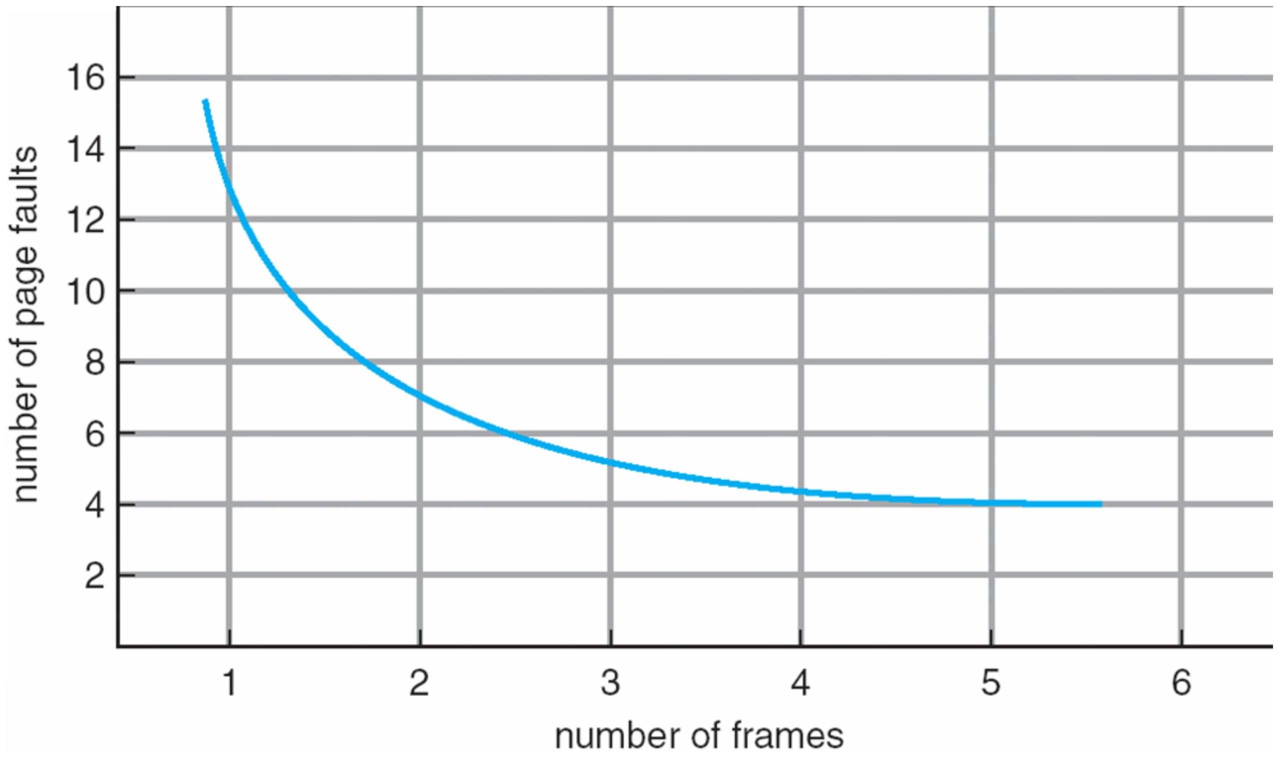
Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string

- String is just page numbers, not full addresses
- Repeated access to the same page does not cause a page fault
- Results depend on number of frames available

In all our examples, the **reference string** of referenced page numbers is

7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

No. of Page Faults Vs No. of Frames



Algorithms

- First In First Out
- Optimal Replacement
- Not Recently Used
- Second Chance
- Least Recently Used
- Not Frequently Used

First-In First-Out (FIFO)

- Pages in main memory are kept in a list
- Newest page is in head and the oldest in tail
- It does not take advantage of page access patterns or frequency

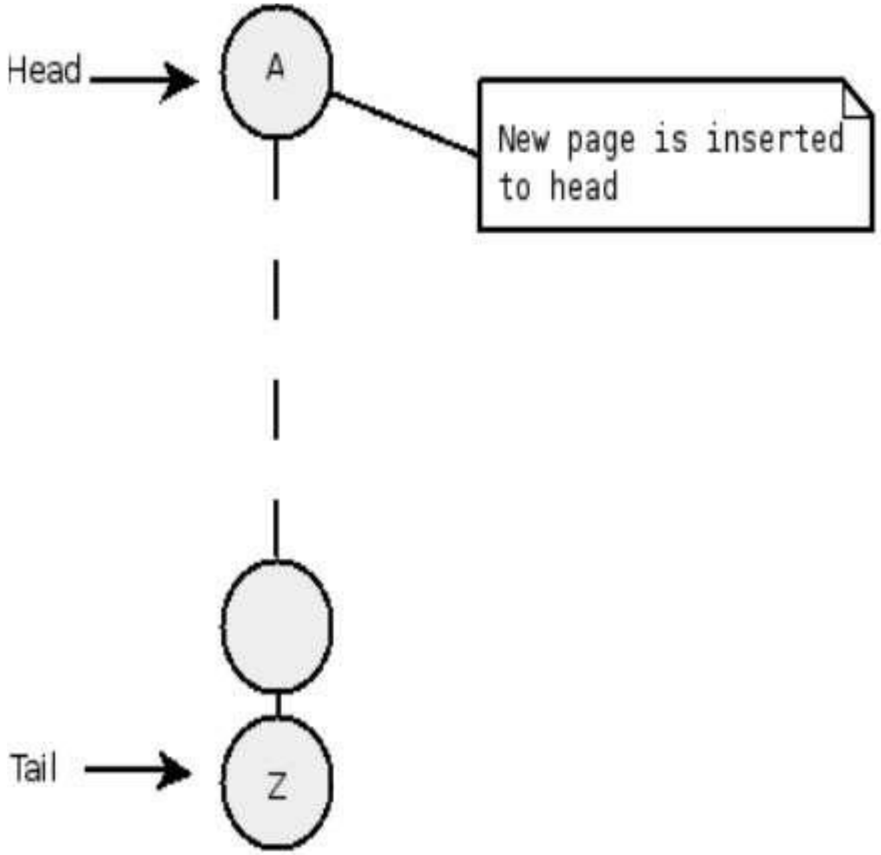
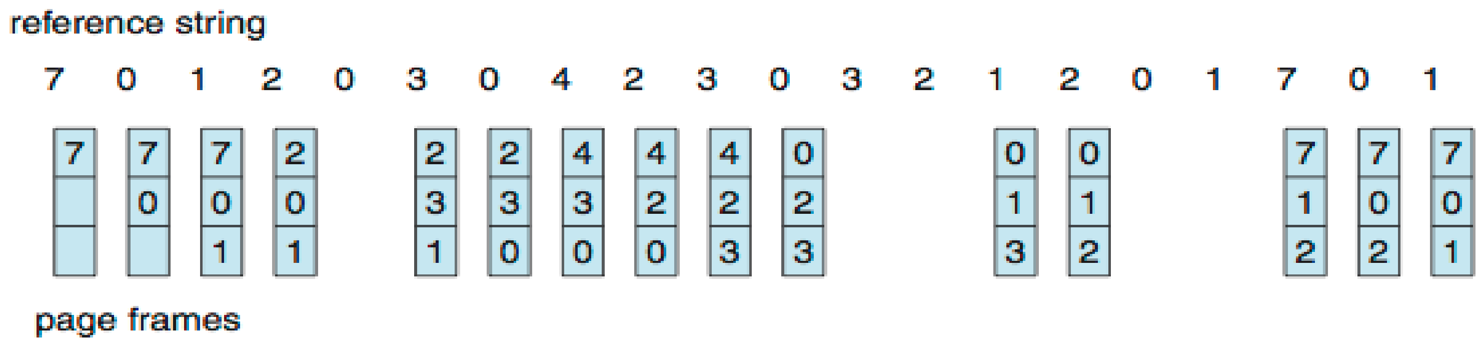


Fig: FIFO

First-In-First-Out (FIFO) Example

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



15 page faults

- Can vary by reference string: consider 1,2,3,4,1,2,5,1,2,3,4,5
  - Adding more frames can cause more page faults!
- How to track ages of pages?
  - Just use a FIFO queue
  - Page hit: no of hits/total no of refrence x 100
  - Page fault: no of faults/total no of refrence x 100

Optimal Replacement (OPT)

- When the memory is full, evict a page that will be unreferenced for the longest time
- The OS keeps track of all pages referenced by the program
- Only if the program's memory reference pattern is relatively consistent

OPTIMAL Example

- Replace page that will not be used for longest period of time
  - 9 is optimal for the example
- How do you know this?
  - Can't read the future
- Used for measuring how well your algorithm performs

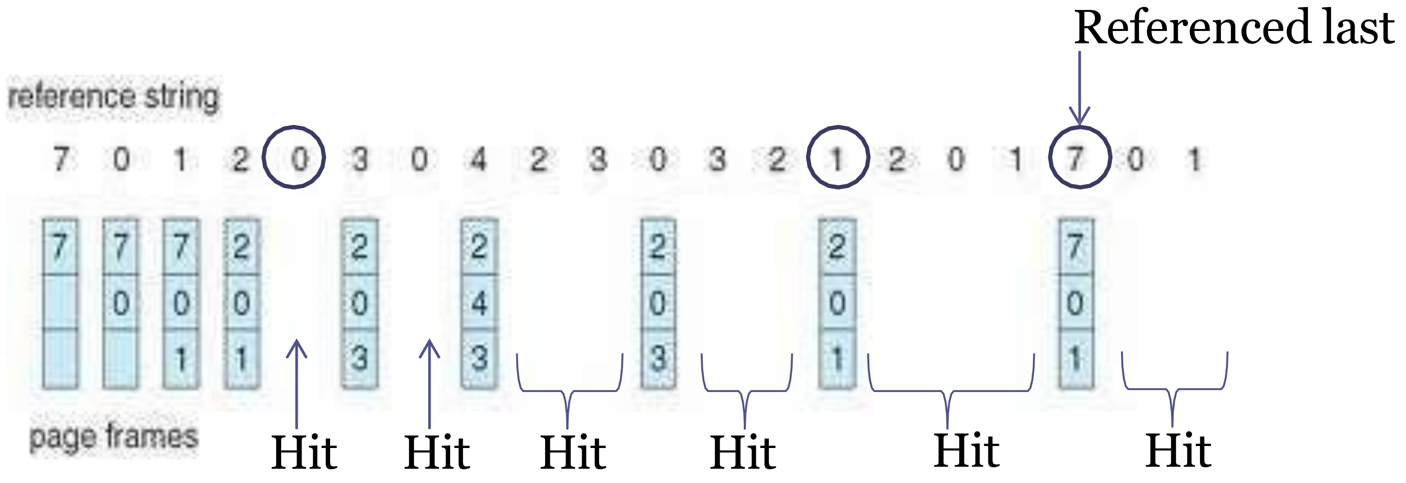


Fig: OPTIMAL example

Not Recently Used (NRU)

- It favours keeping pages in memory that have been recently used.
- The OS divides the pages into four classes based on usage during the last clock tick:

    3. Referenced, modified

    2. Referenced, not modified

    1. Not referenced, modified

    0. Not referenced, not modified

- Pick a random page from the lowest category for removal
- i.e. the not referenced, not modified page

NRU Example

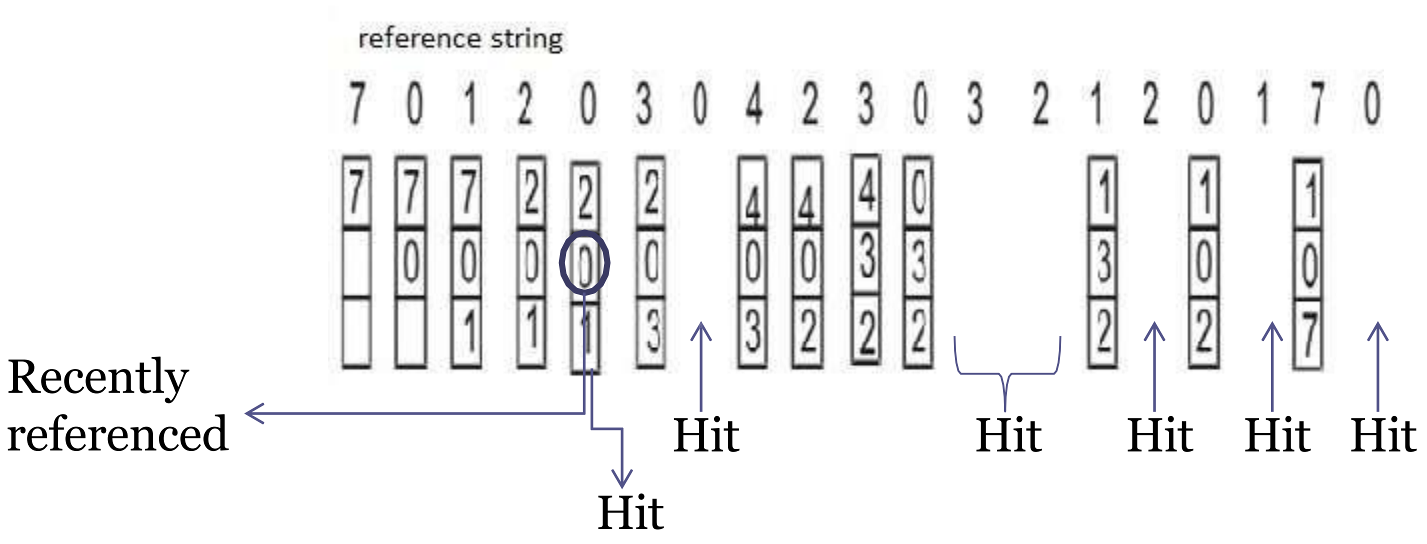


Fig: NRU example

Second Chance

- Modified version of FIFO
- Instead of swapping out the last page, the referenced bit is checked
- Gives every page a "second-chance"

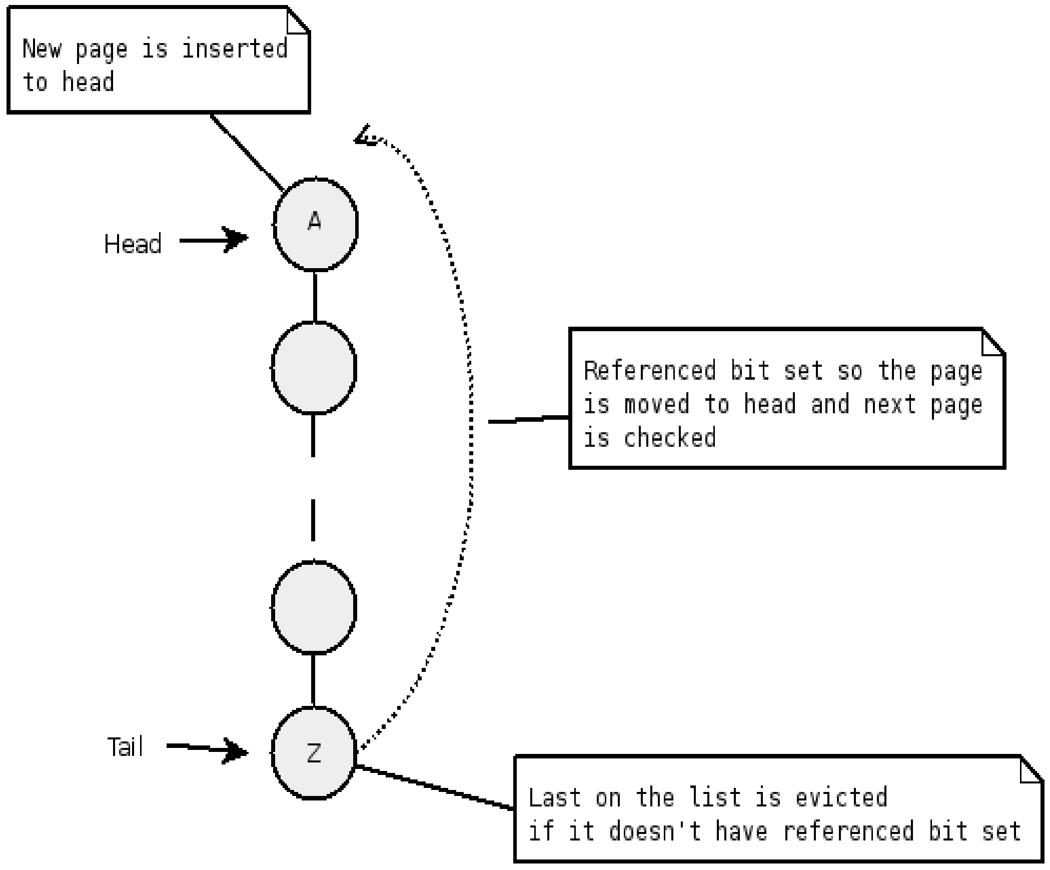
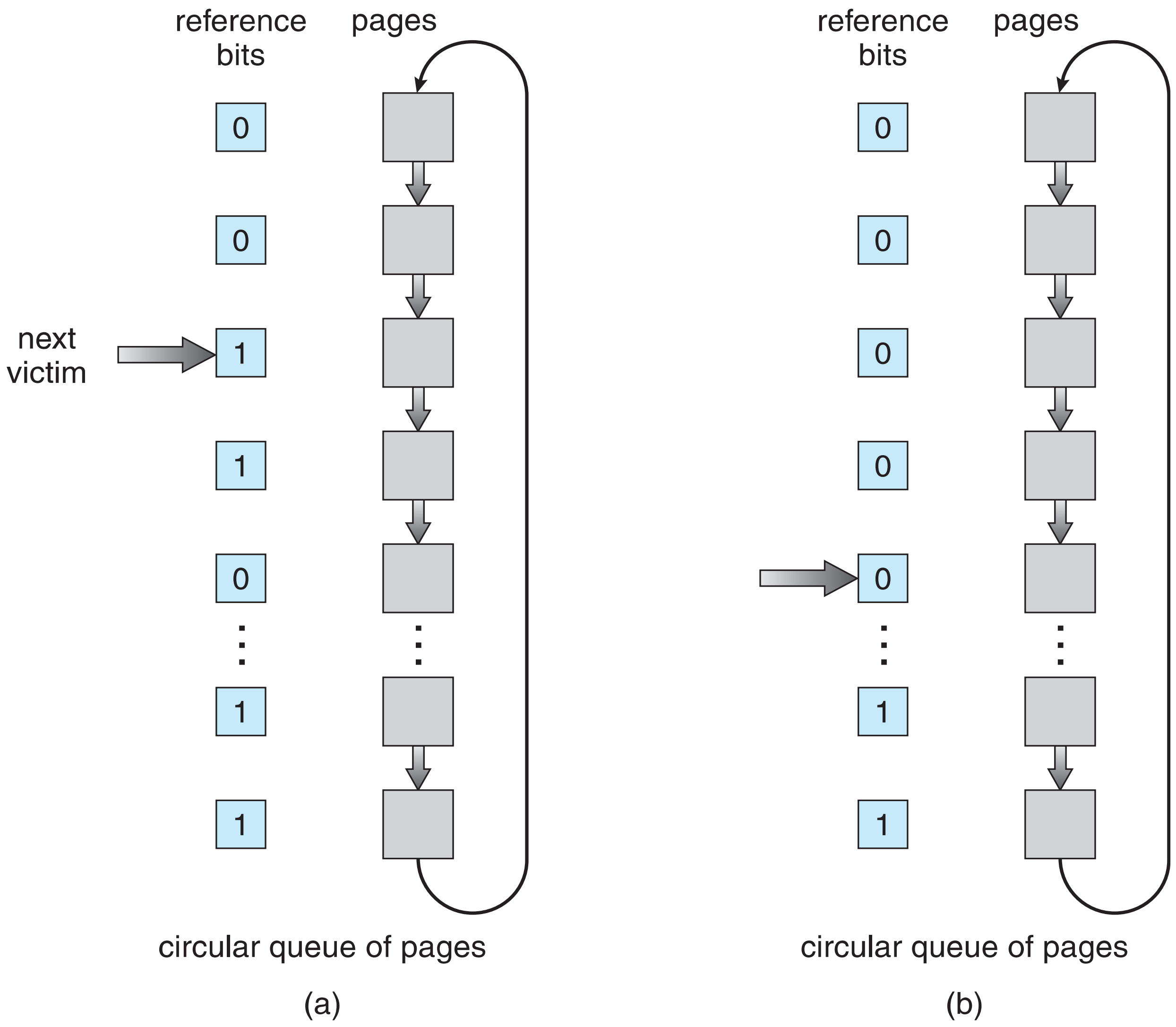

New page is inserted to head
Head
A
Referenced bit set so the page is moved to head and next page is checked
Tail
Z
Last on the list is evicted if it doesn't have referenced bit set

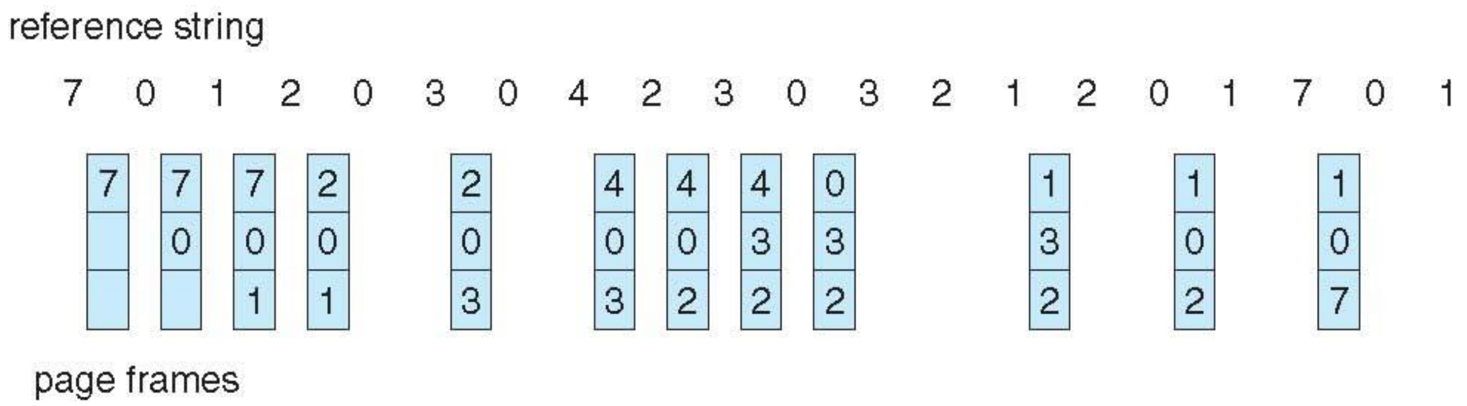
Fig: Second Chance

Second-Chance (clock) Page-Replacement Algorithm



Least Recently Used (LRU) Algorithm

Use past knowledge rather than future
Replace page that has not been used in the most amount of time
Associate time of last use with each page



12 faults – better than FIFO but worse than OPT
Generally good algorithm and frequently used
But how to implement?

- It swaps the pages that have been used the least over a period of time.
- It is free from Belady's anomaly.

LRU Algorithm (Cont.)

Counter implementation

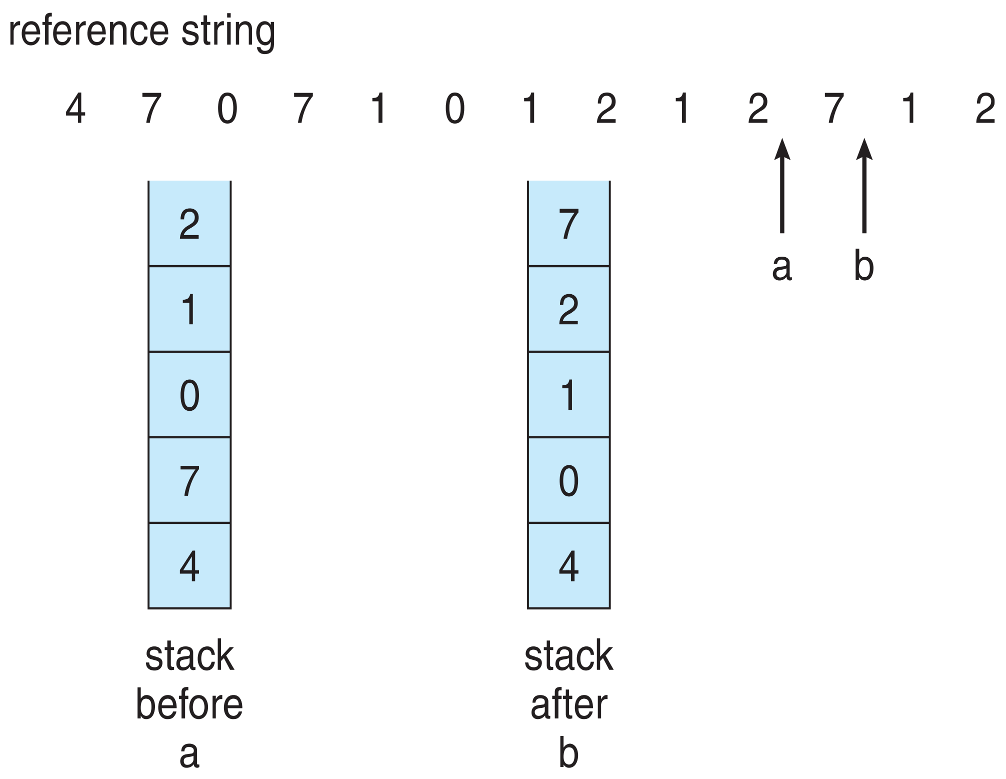
- Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
- When a page needs to be changed, look at the counters to find smallest value
  - Search through table needed

Stack implementation

- Keep a stack of page numbers in a double link form:
- Page referenced:
  - move it to the top
  - requires 6 pointers to be changed
- But each update more expensive
- No search for replacement

LRU and OPT are cases of **stack algorithms** that don' t have Belady' s Anomaly

Use Of A Stack to Record Most Recent Page References



LRU Approximation Algorithms

- LRU needs special hardware and still slow
- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - We do not know the order, however
- **Second-chance algorithm**
  - Generally FIFO, plus hardware-provided reference bit
  - Clock replacement
  - If page to be replaced has
    - Reference bit = 0 -> replace it
    - reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - replace next page, subject to same rules

Not frequently used (NFU)

- This page replacement algorithm requires a counter
- The counters keep track of how frequently a page has been used
- The page with the lowest counter can be swapped out

reference sequence : 3230842509832

```
 P   U   3     P   U   2    P   U   3    P   U   0    P   U   8    P   U  4
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
|   | 0 |*    | 3 | 1 |    | 3 | 1 |    | 3 | 1 |    | 3 | 1 |    | 3 | 1 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
|   | 0 |     |   | 0 |*   | 2 | 1 |    | 2 | 1 |    | 2 | 1 |    | 2 | 1 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
|   | 0 |     |   | 0 |    |   | 0 |*   |   | 0 |*   | 0 | 1 |    | 0 | 1 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
|   | 0 |     |   | 0 |    |   | 0 |    |   | 0 |    |   | 0 |*   | 8 | 1 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
|   | 0 |     |   | 0 |    |   | 0 |    |   | 0 |    |   | 0 |    |   | 0 |*
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+------

  P   U   2    P   U   5    P   U   0    P   U   9    P   U   8    P   U  3
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
| 3 | 1 |*    | 3 | 1 |*   | 5 | 1 |    | 5 | 1 |    | 5 | 1 |    | 5 | 1 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
| 2 | 1 |     | 2 | 1 |    | 2 | 0 |*   | 2 | 0 |*   | 9 | 1 |    | 9 | 1 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
| 0 | 1 |     | 0 | 1 |    | 0 | 0 |    | 0 | 1 |    | 0 | 1 |*   | 0 | 1 |*
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
| 8 | 1 |     | 8 | 1 |    | 8 | 0 |    | 8 | 0 |    | 8 | 0 |    | 8 | 1 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+---+
| 4 | 1 |     | 4 | 1 |    | 4 | 0 |    | 4 | 0 |    | 4 | 0 |    | 4 | 0 |
+---+---+     +---+---+    +---+---+    +---+---+    +---+---+    +---+------
```

```
 P  U  2   P  U
+---+---+   +---+---+
|5|1|*   |5|0|
|9|1|    |9|0|
+---+---+   +---+---+
|0|0|    |2|1|
+---+---+   +---+---+
|8|0|    |8|0|*
+---+---+   +---+---+
|3|1|    |3|1|
+---+---+   +---+---+
```

* = indicates the pointer which identifies the next location
to scan P = page# stored in that frame U = used flag
0 = not used recently 1 = referenced recently

Fig: NFU example

Conclusion

Algorithm	Comment
FIFO	Might throw out important pages
OPTIMAL	Not implementable
LRU	Excellent but difficult to implement
NRU	Crude approximation of LRU
NFU	Crude approximation of LRU
Second Chance	Big improvement over FIFO
CLOCK	Realistic

References

- Web Links

  www.wikipedia.com

  www.youtube.com

  www.vbForum.com

- Papers

  Operating System Page Replacement Algorithms by A. Frank C. Wersberg

- Books

  Computer Organization & Architecture by William Stallings

Reference video Web Links

FIFO

https://youtu.be/nkV11C8G998

https://youtu.be/16kaPQtYo28

OPTIMAL

https://youtu.be/LvBpXzqKWDE

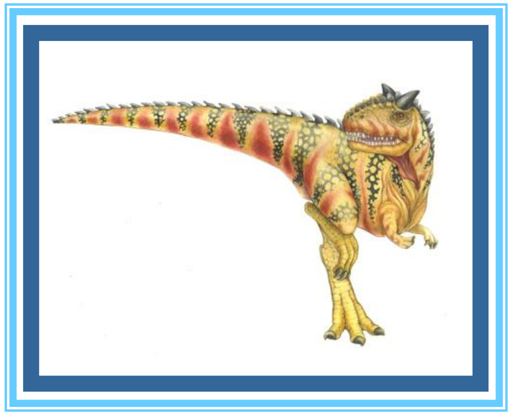
https://youtu.be/jeJIKKQcqpU

For more details subscribe the computer science page on YouTube.

Thank You

Week 13
Operating System Concepts

Muhammad Daniyal Liaquat



Topic

Storage Management

Storage Management

- **Storage Management** is defined as it refers to the management of the data storage equipment's that are used to store the user/computer generated data. Hence it is a tool or set of processes used by an administrator to keep your data and storage equipment's safe. Storage management is a process for users to optimize the use of storage devices and to protect the integrity of data for any media on which it resides and the category of storage management generally contain the different type of subcategories covering aspects such as security, virtualization and more, as well as different types of provisioning or automation, which is generally made up the entire storage management software market.

- **Storage management key attributes:** Storage management has some key attribute which is generally used to manage the storage capacity of the system. These are given below:

1. Performance
2. Reliability
3. Recoverability
4. Capacity

Feature of Storage management

- There is some feature of storage management which is provided for storage capacity. These are given below:
- Storage management is a process that is used to optimize the use of storage devices.
- Storage management must be allocated and managed as a resource in order to truly benefit a corporation.
- Storage management is generally a basic system component of information systems.
- It is used to improve the performance of their data storage resources.

Advantage of storage management:

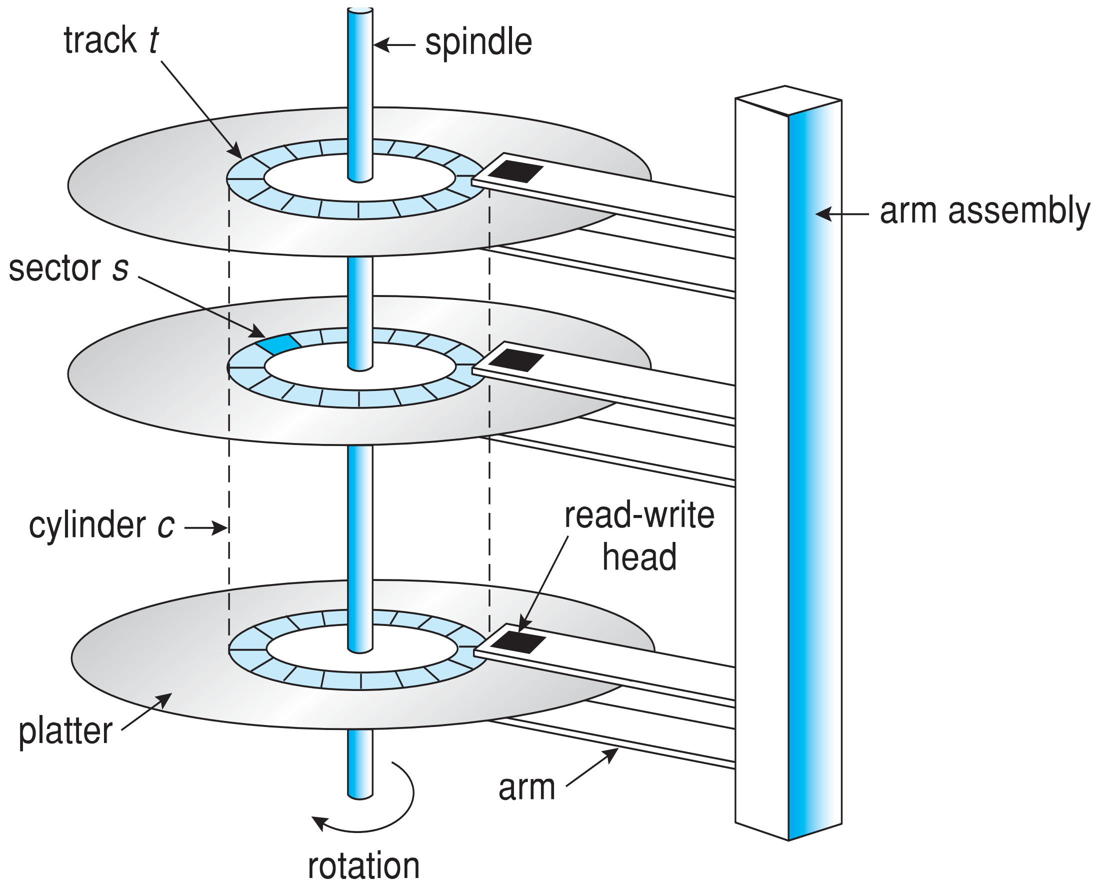
There are some advantage of storage management which are given below:

- It becomes very simple to manage a storage capacity.
- It generally reduces the time consumption.
- It improves the performance of system.
- In virtualization and automation technologies, it can help an organization improve its agility.

Overview of Mass Storage Structure

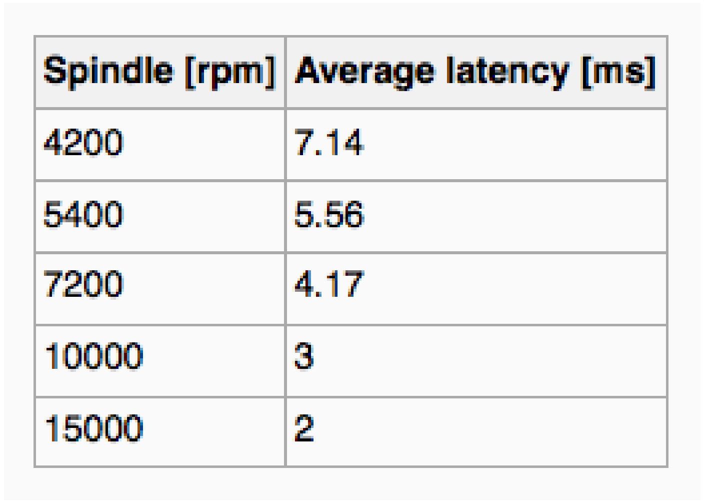
- **Magnetic disks** provide bulk of secondary storage of modern computers
  - Drives rotate at 60 to 250 times per second
  - **Transfer rate** is rate at which data flow between drive and computer
  - **Positioning time** (**random-access time**) is time to move disk arm to desired cylinder (**seek time**) and time for desired sector to rotate under the disk head (**rotational latency**)
  - **Head crash** results from disk head making contact with the disk surface -- That's bad
- Disks can be removable
- Drive attached to computer via **I/O bus**
  - Busses vary, including **EIDE**, **ATA**, **SATA**, **USB**, **Fibre Channel**, **SCSI, SAS, Firewire**
  - **Host controller** in computer uses bus to talk to **disk controller** built into drive or storage array

Moving-head Disk Mechanism



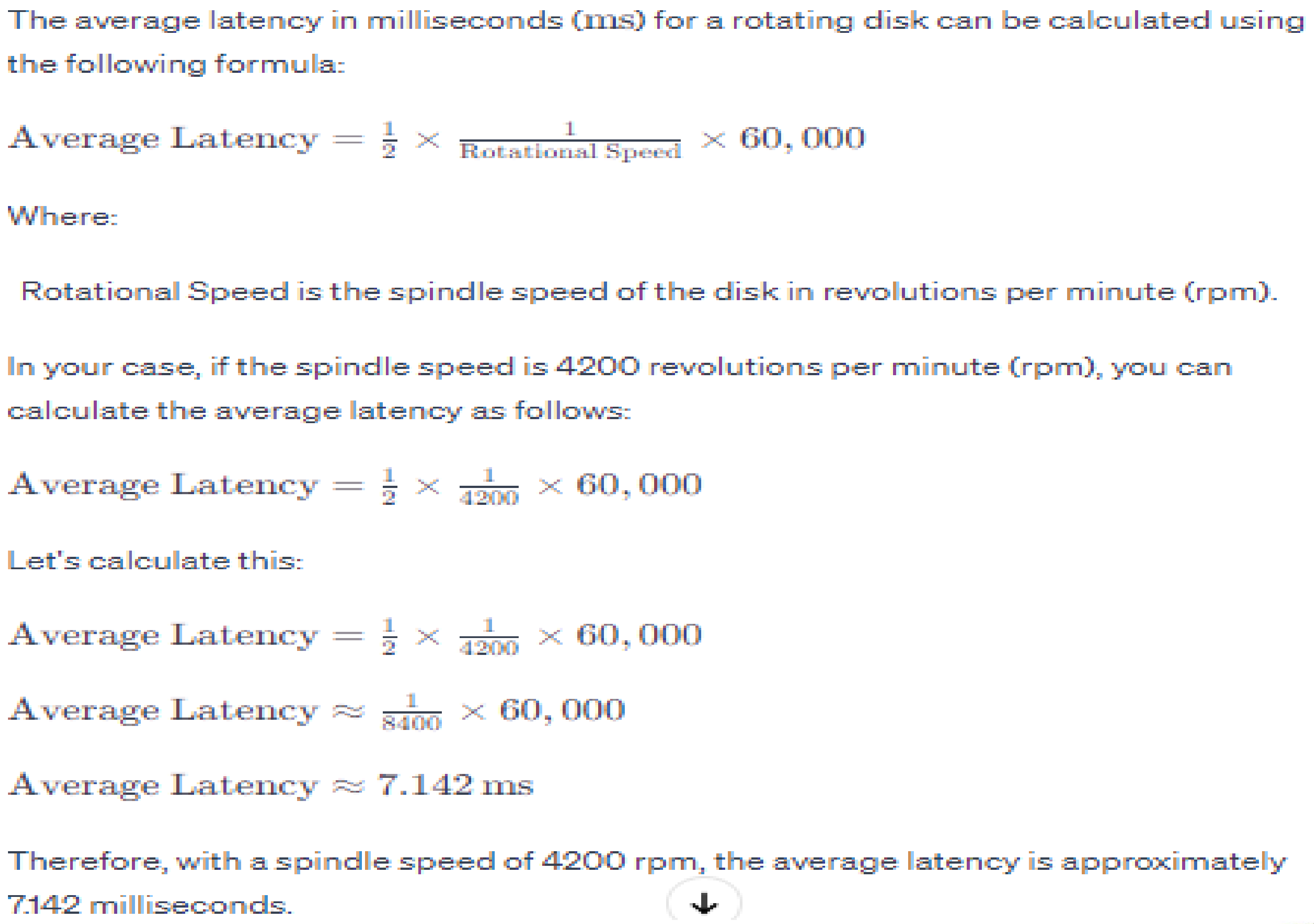
Hard Disks

- Platters range from .85" to 14" (historically)
  - Commonly 3.5", 2.5", and 1.8"
- Range from 30GB to 3TB per drive
- Performance
  - Transfer Rate – theoretical – 6 Gb/sec
  - Effective Transfer Rate – real – 1Gb/sec
  - Seek time from 3ms to 12ms – 9ms common for desktop drives
  - Average seek time measured or calculated based on 1/3 of tracks
  - Latency based on spindle speed
    - 1 / (RPM / 60) = 60 / RPM
  - Average latency = ½ latency



Spindle [rpm]	Average latency [ms]
4200	7.14
5400	5.56
7200	4.17
10000	3
15000	2

(From Wikipedia)



The average latency in milliseconds (ms) for a rotating disk can be calculated using the following formula:

$$\text{Average Latency} = \frac{1}{2} \times \frac{1}{\text{Rotational Speed}} \times 60,000$$

Where:

Rotational Speed is the spindle speed of the disk in revolutions per minute (rpm).

In your case, if the spindle speed is 4200 revolutions per minute (rpm), you can calculate the average latency as follows:

$$\text{Average Latency} = \frac{1}{2} \times \frac{1}{4200} \times 60,000$$

Let's calculate this:

$$\text{Average Latency} = \frac{1}{2} \times \frac{1}{4200} \times 60,000$$

$$\text{Average Latency} \approx \frac{1}{8400} \times 60,000$$

$$\text{Average Latency} \approx 7.142 \text{ ms}$$

Therefore, with a spindle speed of 4200 rpm, the average latency is approximately 7.142 milliseconds.

The First Commercial Disk Drive

1956
IBM RAMDAC computer included the IBM Model 350 disk storage system

5M (7 bit) characters
50 x 24” platters
Access time = < 1 second

Solid-State Disks

- Nonvolatile memory used like a hard drive
  - Many technology variations
- Can be more reliable than HDDs
- More expensive per MB
- Maybe have shorter life span
- Less capacity
- But much faster
- Busses can be too slow -> connect directly to PCI for example
- No moving parts, so no seek time or rotational latency

Magnetic Tape

- Was early secondary-storage medium
  - Evolved from open spools to cartridges
- Relatively permanent and holds large quantities of data
- Access time slow
- Random access ~1000 times slower than disk
- Mainly used for backup, storage of infrequently-used data, transfer medium between systems
- Kept in spool and wound or rewound past read-write head
- Once data under head, transfer rates comparable to disk
  - 140MB/sec and greater
- 200GB to 1.5TB typical storage
- Common technologies are LTO-{3,4,5} and T10000

Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of **logical blocks**, where the logical block is the smallest unit of transfer
  - Low-level formatting creates **logical blocks** on physical media
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
  - Sector 0 is the first sector of the first track on the outermost cylinder
  - 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
  - Logical to physical address should be easy
    - Except for bad sectors
    - Non-constant # of sectors per track via constant angular velocity

Disk Attachment

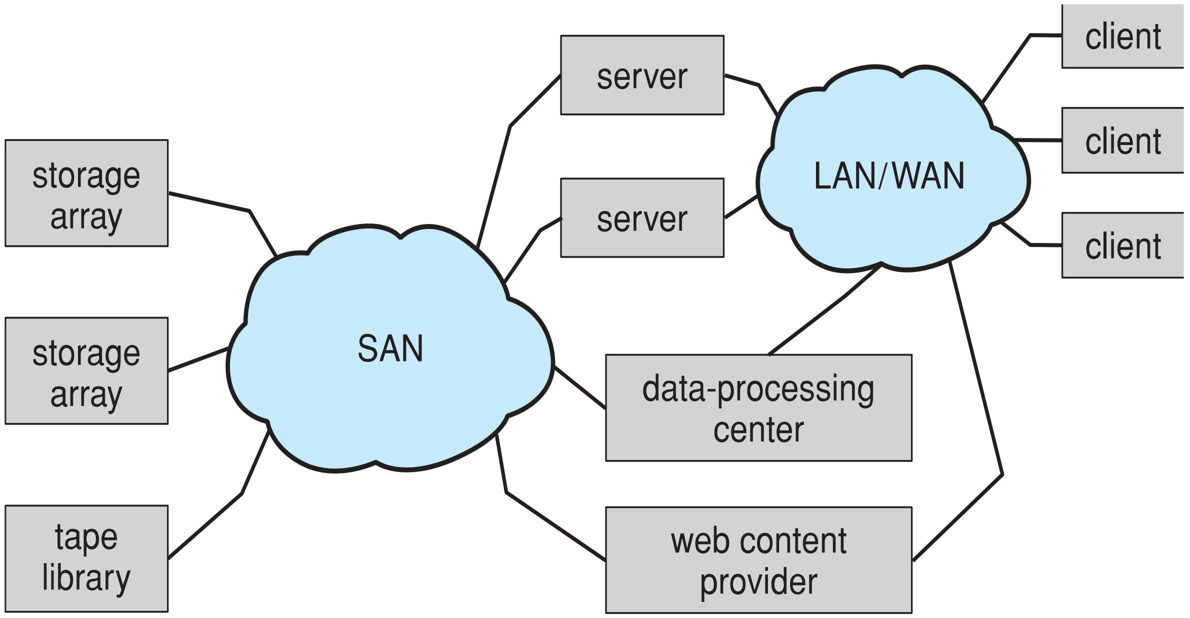
- Host-attached storage accessed through I/O ports talking to I/O busses
- SCSI itself is a bus, up to 16 devices on one cable, **SCSI initiator** requests operation and **SCSI targets** perform tasks
  - Each target can have up to 8 **logical units** (disks attached to device controller)
- FC is high-speed serial architecture
  - Can be switched fabric with 24-bit address space – the basis of **storage area networks (SAN**s**)** in which many hosts attach to many storage units
- I/O directed to bus ID, device ID, logical unit (LUN)

Storage Array

- Can just attach disks, or arrays of disks
- Storage Array has controller(s), provides features to attached host(s)
  - Ports to connect hosts to array
  - Memory, controlling software (sometimes NVRAM, etc)
  - A few to thousands of disks
  - RAID, hot spares, hot swap (discussed later)
  - Shared storage -> more efficiency
  - Features found in some file systems
    - Snaphots, clones, thin provisioning, replication, deduplication, etc

Storage Area Network

- Common in large storage environments
- Multiple hosts attached to multiple storage arrays - flexible

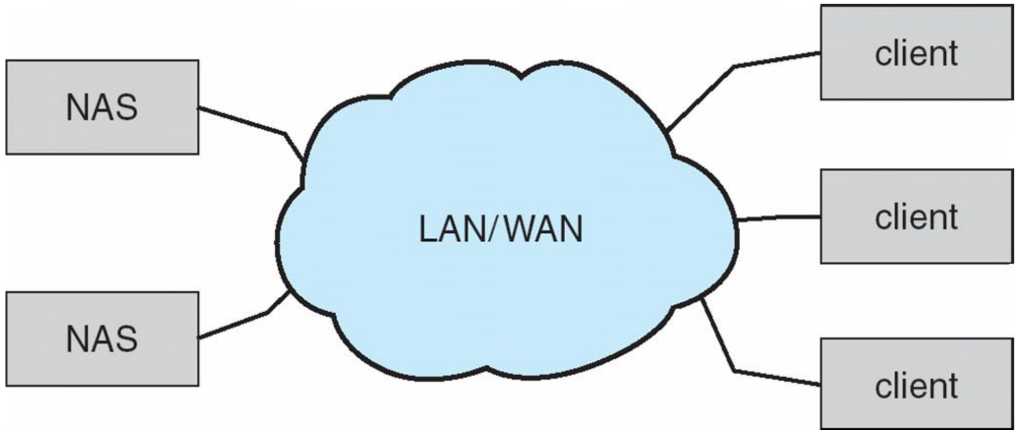


Storage Area Network (Cont.)

- SAN is one or more storage arrays
  - Connected to one or more Fibre Channel switches
- Hosts also attach to the switches
- Storage made available via **LUN Masking** from specific arrays to specific servers
- Easy to add or remove storage, add new host and allocate it storage
  - Over low-latency Fibre Channel fabric
- Why have separate storage networks and communications networks?
  - Consider iSCSI, FCOE

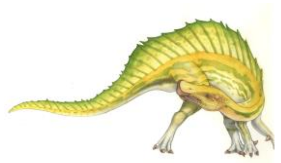
Network-Attached Storage

- Network-attached storage (**NAS**) is storage made available over a network rather than over a local connection (such as a bus)
  - Remotely attaching to file systems
- NFS and CIFS are common protocols
- Implemented via remote procedure calls (RPCs) between host and storage over typically TCP or UDP on IP network
- **iSCSI** protocol uses IP network to carry the SCSI protocol
  - Remotely attaching to devices (blocks)



Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time ≈ seek distance
- 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
- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists

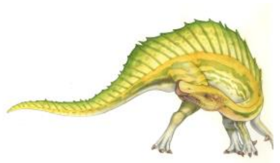


Disk Scheduling (Cont.)

- Note that drive controllers have small buffers and can manage a queue of I/O requests (of varying “depth”)
- Several algorithms exist to schedule the servicing of disk I/O requests
- The analysis is true for one or many platters
- We illustrate scheduling algorithms with a request queue (0-199)

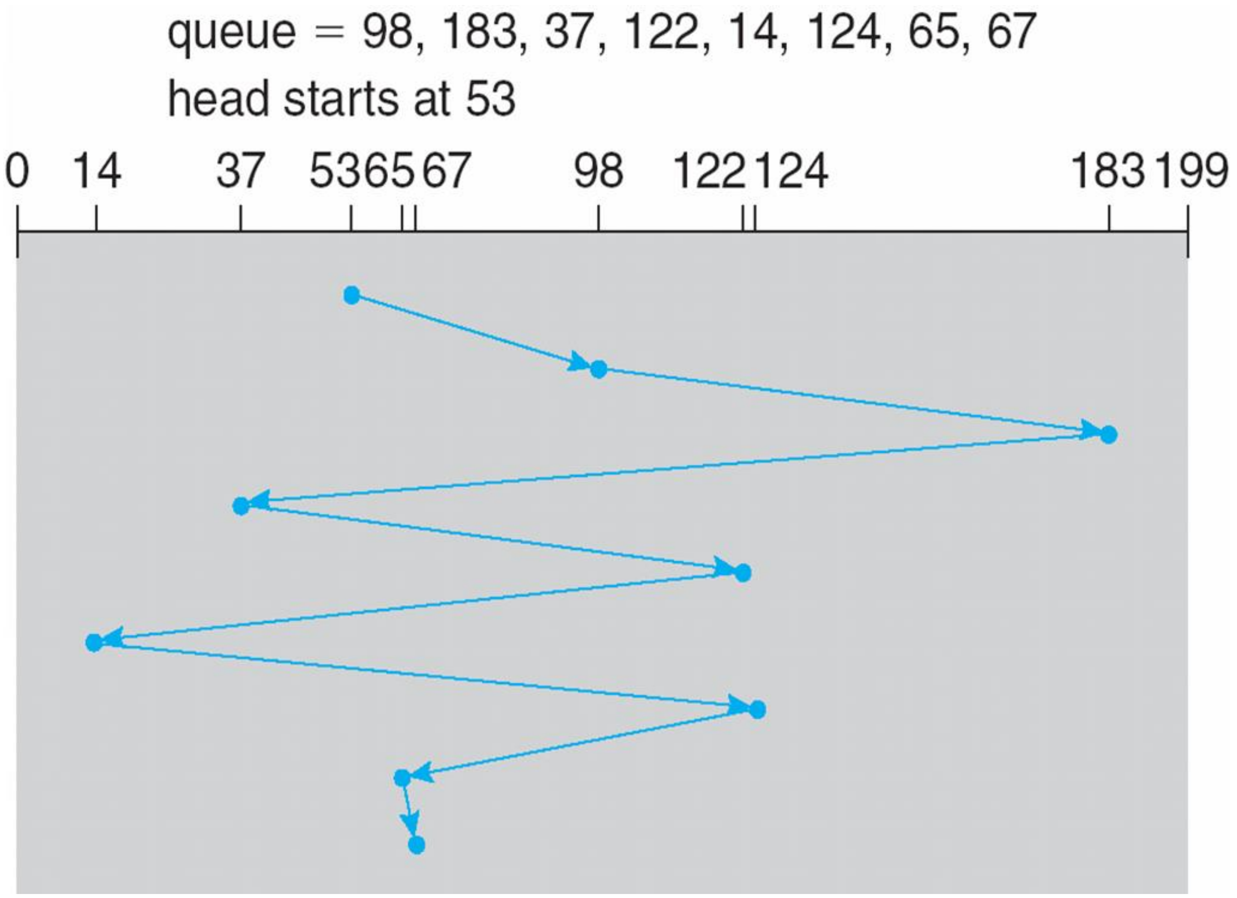
98, 183, 37, 122, 14, 124, 65, 67

Head pointer 53



FCFS

Illustration shows total head movement of 640 cylinders

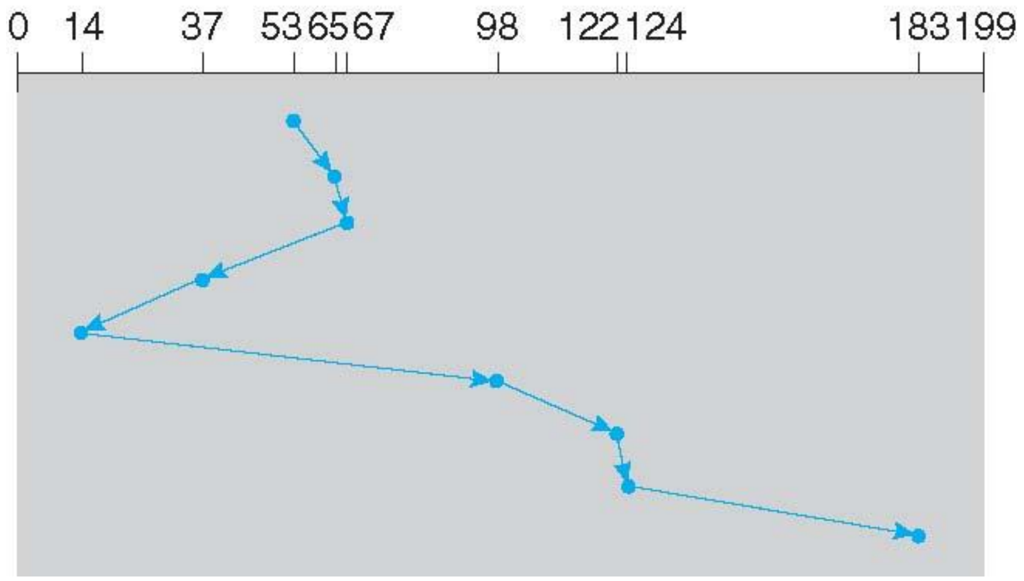


SSTF

- 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

queue = 98, 183, 37, 122, 14, 124, 65, 67

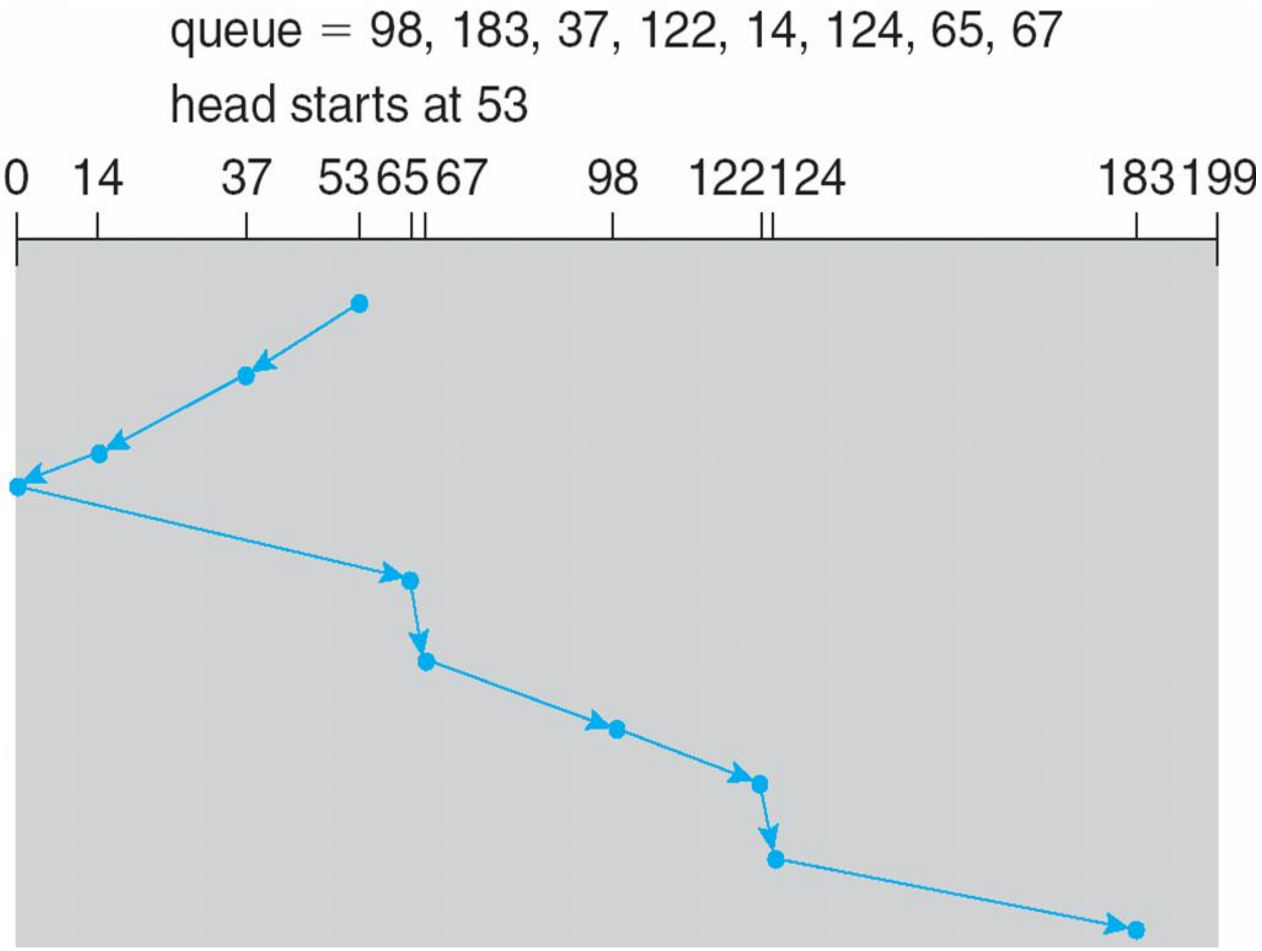
head starts at 53



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 236 cylinders
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest

SCAN (Cont.)



queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method
  - And metadata layout
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary.

Disk Management

- **Low-level formatting**, or **physical formatting** — Dividing a disk into sectors that the disk controller can read and write
  - Each sector can hold header information, plus data, plus error correction code (**ECC**)
  - Usually 512 bytes of data but can be selectable
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk
  - **Partition** the disk into one or more groups of cylinders, each treated as a logical disk
  - **Logical formatting** or "making a file system"
  - To increase efficiency most file systems group blocks into **clusters**
    - Disk I/O done in blocks
    - File I/O done in clusters
- Raw disk access for apps that want to do their own block management, keep OS out of the way (databases for example)
- Boot block initializes system
  - The bootstrap is stored in ROM
  - **Bootstrap loader** program stored in boot blocks of boot partition
- Methods such as **sector sparing** used to handle bad blocks.

Stable-Storage Implementation

- Write-ahead log scheme requires stable storage
- Stable storage means data is never lost (due to failure, etc)
- To implement stable storage:
  - Replicate information on more than one nonvolatile storage media with independent failure modes
  - Update information in a controlled manner to ensure that we can recover the stable data after any failure during data transfer or recovery
- Disk write has 1 of 3 outcomes
  1. **Successful completion -** The data were written correctly on disk
  2. **Partial failure -** A failure occurred in the midst of transfer, so only some of the sectors were written with the new data, and the sector being written during the failure may have been corrupted
  3. **Total failure -** The failure occurred before the disk write started, so the previous data values on the disk remain intact

Stable-Storage Implementation (Cont.)

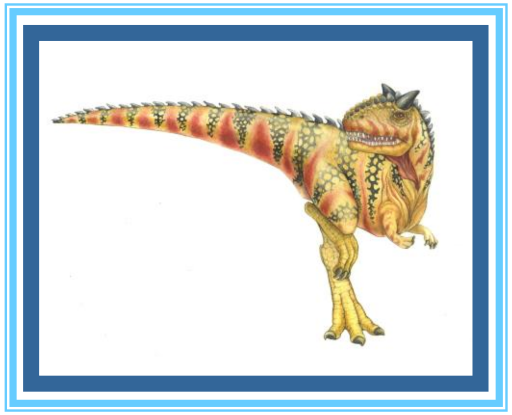
- If failure occurs during block write, recovery procedure restores block to consistent state
  - System maintains 2 physical blocks per logical block and does the following:
    1. Write to 1st physical
    2. When successful, write to 2nd physical
    3. Declare complete only after second write completes successfully

  Systems frequently use NVRAM as one physical to accelerate

Limitations of storage management:

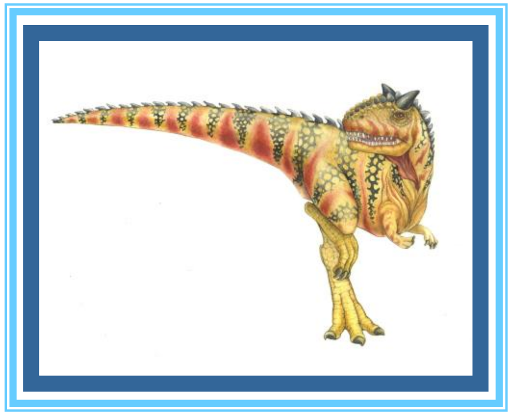
- Limited physical storage capacity: Operating systems can only manage the physical storage space that is available, and as such, there is a limit to how much data can be stored.
- Performance degradation with increased storage utilization: As more data is stored, the system’s performance can decrease due to increased disk access time, fragmentation, and other factors.
- Complexity of storage management: Storage management can be complex, especially as the size of the storage environment grows.
- Cost: Storing large amounts of data can be expensive, and the cost of additional storage capacity can add up quickly.
- Security issues: Storing sensitive data can also present security risks, and the operating system must have robust security features in place to prevent unauthorized access to this data.
- Backup and Recovery: Backup and recovery of data can also be challenging, especially if the data is stored on multiple systems or devices.

Any Questions ?



Week 14
Operating System Concepts

Muhammad Daniyal Liaquat



Topic

1. **File System Concepts**
2. **Access Methods, Directory Structures**

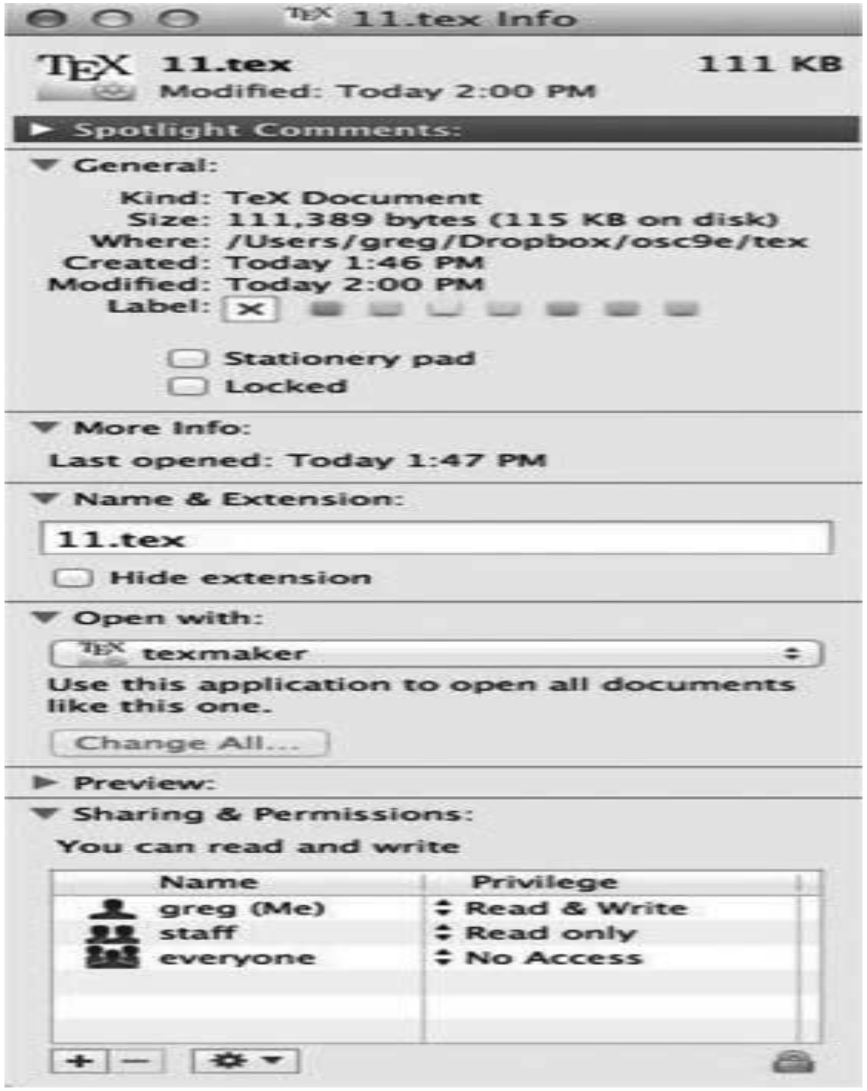
File Concept

- Contiguous logical address space
- Types:
  - Data
    - numeric
    - character
    - binary
  - Program
- Contents defined by file's creator
  - Many types
    - Consider **text file, source file, executable file**

File Attributes

- **Name** – only information kept in human-readable form
- **Identifier** – unique tag (number) identifies file within file system
- **Type** – needed for systems that support different types
- **Location** – pointer to file location on device
- **Size** – current file size
- **Protection** – controls who can do reading, writing, executing
- **Time, date, and user identification** – data for protection, security, and usage monitoring
- Information about files are kept in the directory structure, which is maintained on the disk
- Many variations, including extended file attributes such as file checksum
- Information kept in the directory structure

File info Window on Mac OS X



11.tex Info

11.tex 111 KB
Modified: Today 2:00 PM

▶ Spotlight Comments:

▼ General:

Kind:	TeX Document
Size:	111,389 bytes (115 KB on disk)
Where:	/Users/greg/Dropbox/osc9e/tex
Created:	Today 1:46 PM
Modified:	Today 2:00 PM
Label:	✕

☐ Stationery pad
☐ Locked

▼ More Info:

Last opened: Today 1:47 PM

▼ Name & Extension:

11.tex

☐ Hide extension

▼ Open with:

texmaker

Use this application to open all documents like this one.

Change All...

▶ Preview:

▼ Sharing & Permissions:

You can read and write

Name	Privilege
greg (Me)	Read & Write
staff	Read only
everyone	No Access

File Operations

- File is an **abstract data type**
- **Create**
- **Write –** at **write pointer** location
- **Read –** at **read pointer** location
- **Reposition within file - seek**
- **Delete**
- **Truncate**
- ***Open(F_i)*** – search the directory structure on disk for entry ***F_i***, and move the content of entry to memory
- ***Close (F_i)*** – move the content of entry ***F_i*** in memory to directory structure on disk

Open Files

- Several pieces of data are needed to manage open files:
  - **Open-file table**: tracks open files
  - File pointer: pointer to last read/write location, per process that has the file open
  - **File-open count**: counter of number of times a file is open – to allow removal of data from open-file table when last processes closes it
  - Disk location of the file: cache of data access information
  - Access rights: per-process access mode information

Open File Locking

- Provided by some operating systems and file systems
  - Similar to reader-writer locks
  - **Shared lock** similar to reader lock – several processes can acquire concurrently
  - **Exclusive lock** similar to writer lock
- Mediates access to a file
- Mandatory or advisory:
  - **Mandatory** – access is denied depending on locks held and requested
  - **Advisory** – processes can find status of locks and decide what to do

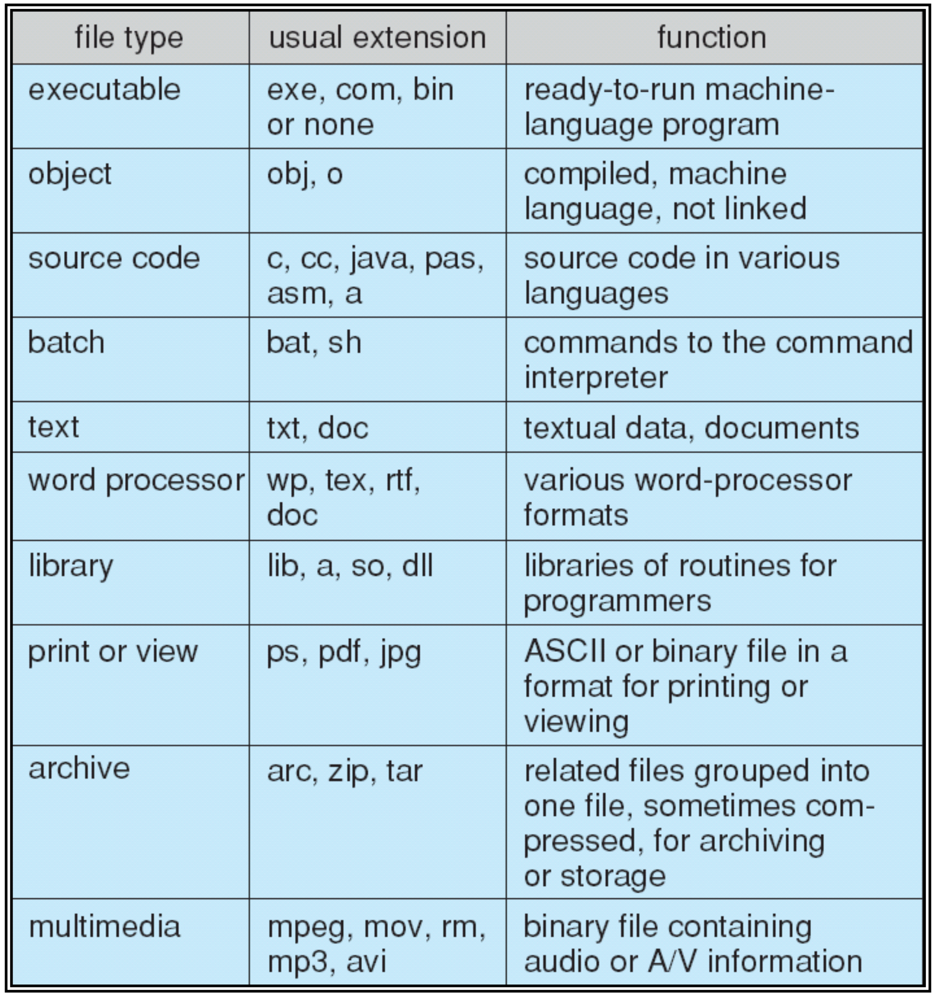
File Locking Example – Java API

```
import java.io.*;
import java.nio.channels.*;
public class LockingExample {
    public static final boolean EXCLUSIVE = false;
    public static final boolean SHARED = true;
    public static void main(String arsg[]) throws IOException {
        FileLock sharedLock = null;
        FileLock exclusiveLock = null;
        try {
            RandomAccessFile raf = new RandomAccessFile("file.txt", "rw");
            // get the channel for the file
            FileChannel ch = raf.getChannel();
            // this locks the first half of the file - exclusive
            exclusiveLock = ch.lock(0, raf.length()/2, EXCLUSIVE);
            /** Now modify the data . . . */
            // release the lock
            exclusiveLock.release();
```

File Locking Example – Java API (Cont.)

```
                // this locks the second half of the file - shared
                sharedLock = ch.lock(raf.length()/2+1, raf.length(),
                                     SHARED);
                /** Now read the data . . . */
                // release the lock
                sharedLock.release();
          } catch (java.io.IOException ioe) {
                System.err.println(ioe);
          }finally {
                if (exclusiveLock != null)
                exclusiveLock.release();
                if (sharedLock != null)
                sharedLock.release();
          }
     }
  }
```

File Types – Name, Extension

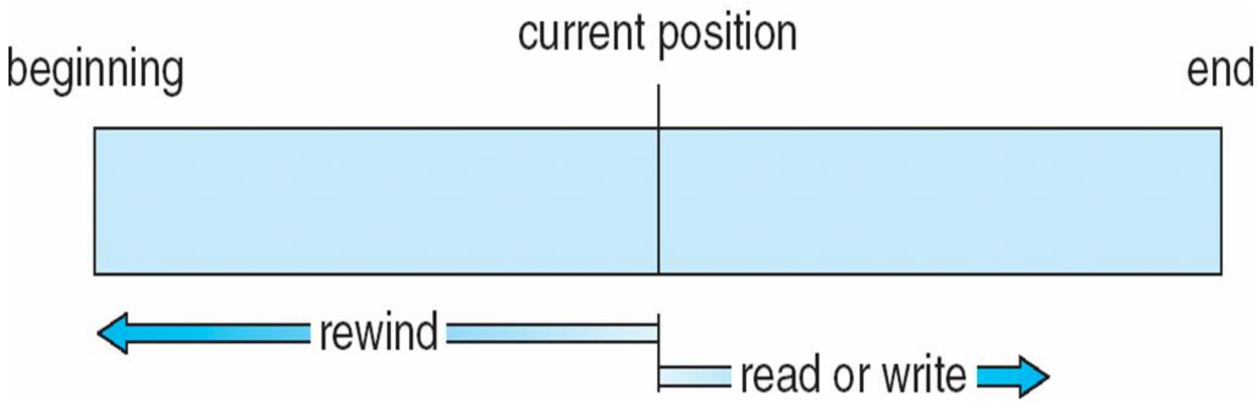


file type	usual extension	function
executable	exe, com, bin or none	ready-to-run machine-language program
object	obj, o	compiled, machine language, not linked
source code	c, cc, java, pas, asm, a	source code in various languages
batch	bat, sh	commands to the command interpreter
text	txt, doc	textual data, documents
word processor	wp, tex, rtf, doc	various word-processor formats
library	lib, a, so, dll	libraries of routines for programmers
print or view	ps, pdf, jpg	ASCII or binary file in a format for printing or viewing
archive	arc, zip, tar	related files grouped into one file, sometimes com-pressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

File Structure

- None - sequence of words, bytes
- Simple record structure
  - Lines
  - Fixed length
  - Variable length
- Complex Structures
  - Formatted document
  - Relocatable load file
- Can simulate last two with first method by inserting appropriate control characters
- Who decides:
  - Operating system
  - Program

Sequential-access File



Access Methods

- **Sequential Access**

```
read next
write next
reset
```

  no read after last write
  (rewrite)

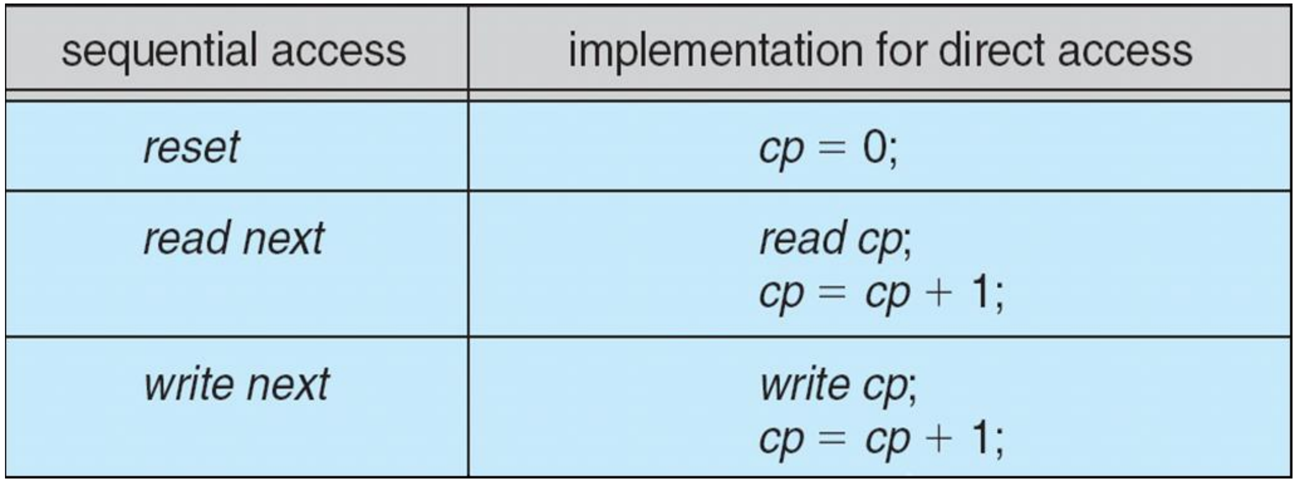
- **Direct Access –** file is fixed length logical records

```
read n
write n
position to n
        read next
        write next
rewrite n
```

  n = relative block number

- Relative block numbers allow OS to decide where file should be placed
  - See allocation problem in Ch 12

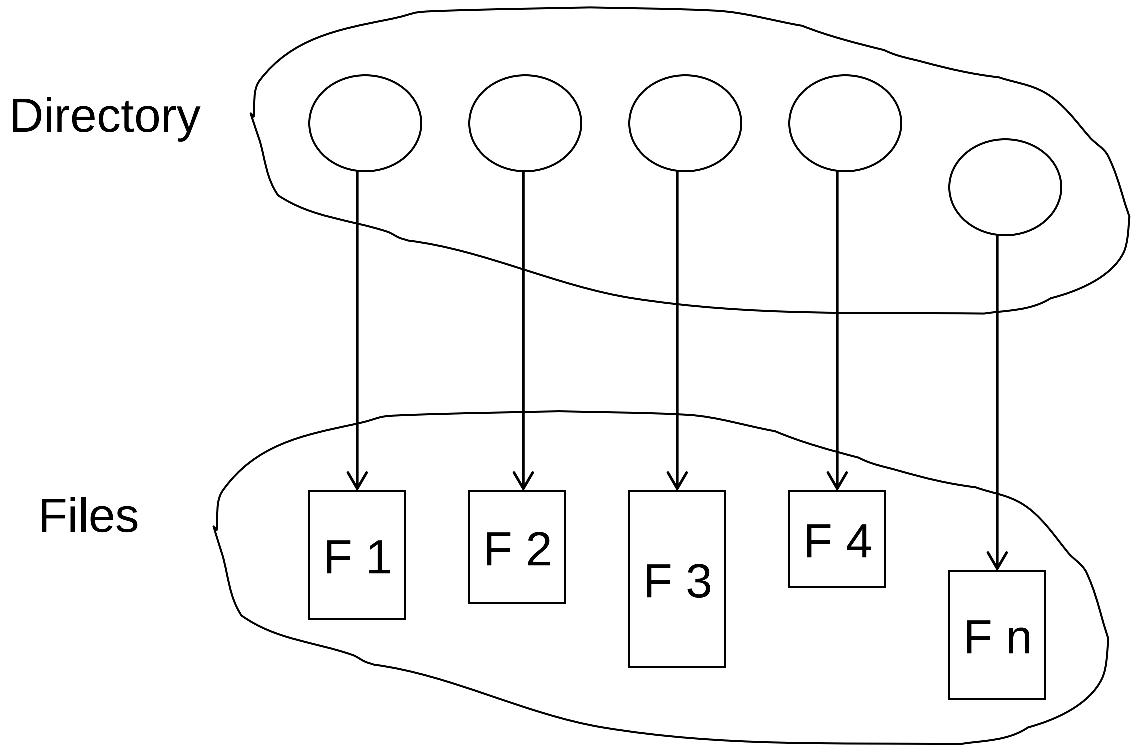
Simulation of Sequential Access on Direct-access File



sequential access	implementation for direct access
reset	$cp = 0;$
read next	*read cp*; $cp = cp + 1;$
write next	*write cp*; $cp = cp + 1;$

Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk

Disk Structure

- Disk can be subdivided into **partitions**
- Disks or partitions can be **RAID** protected against failure
- Disk or partition can be used **raw** – without a file system, or **formatted** with a file system
- Partitions also known as minidisks, slices
- Entity containing file system known as a **volume**
- Each volume containing file system also tracks that file system's info in **device directory** or **volume table of contents**
- As well as **general-purpose file systems** there are many **special-purpose file systems**, frequently all within the same operating system or computer

Operations Performed on Directory

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

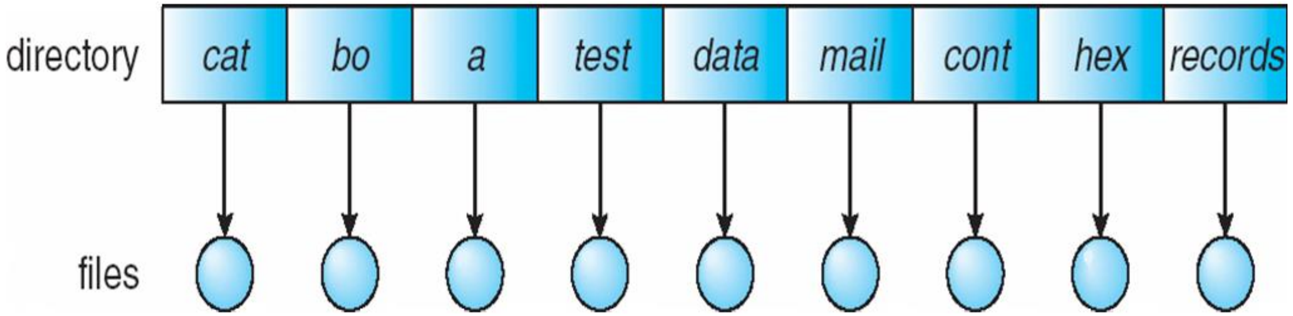
Directory Organization

The directory is organized logically to obtain

- Efficiency – locating a file quickly
- Naming – convenient to users
  - Two users can have same name for different files
  - The same file can have several different names
- Grouping – logical grouping of files by properties, (e.g., all Java programs, all games, …)

Single-Level Directory

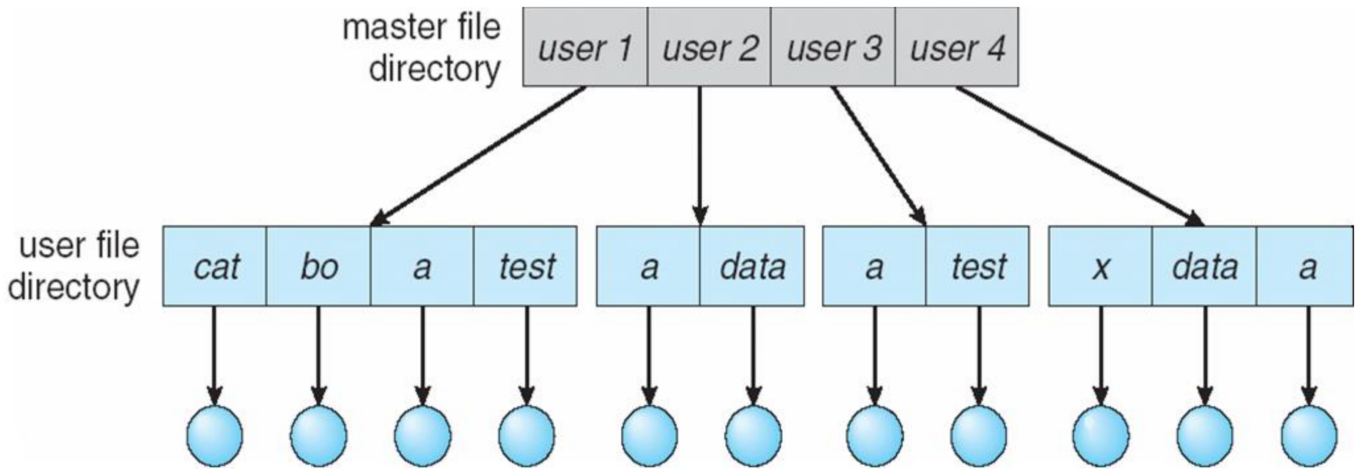
- A single directory for all users



- Naming problem
- Grouping problem

Two-Level Directory

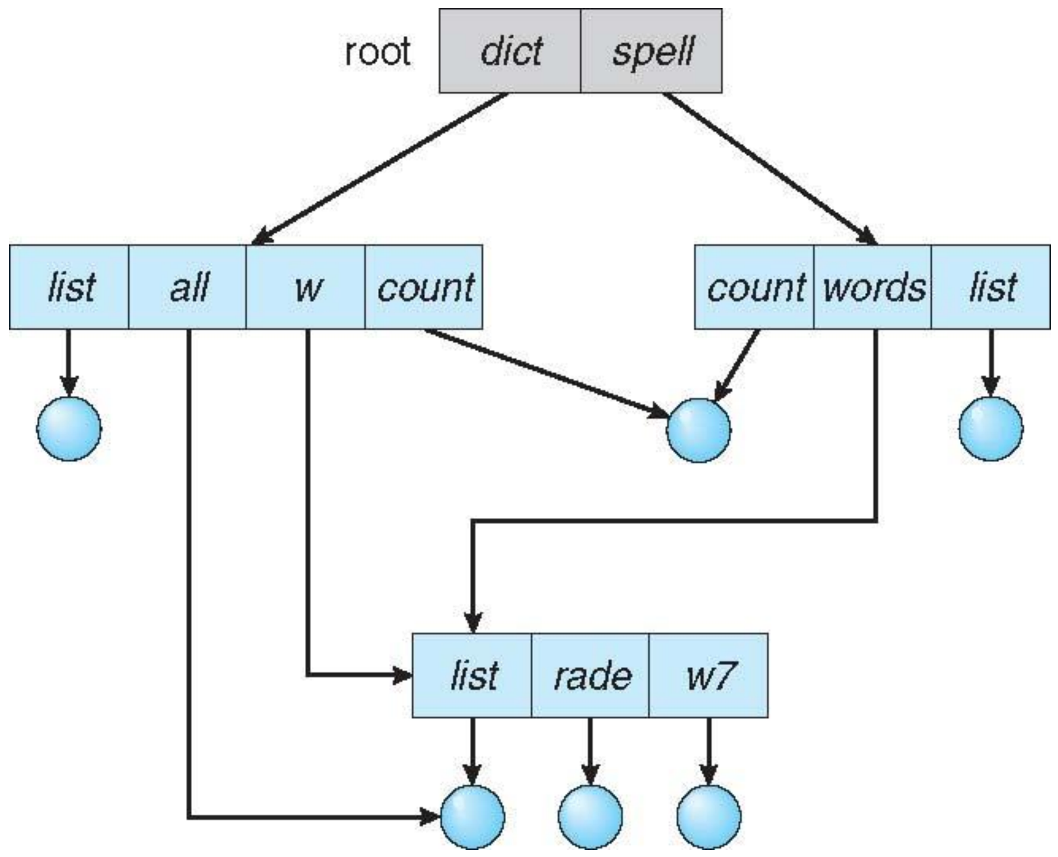
- Separate directory for each user



- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Acyclic-Graph Directories

- Have shared subdirectories and files



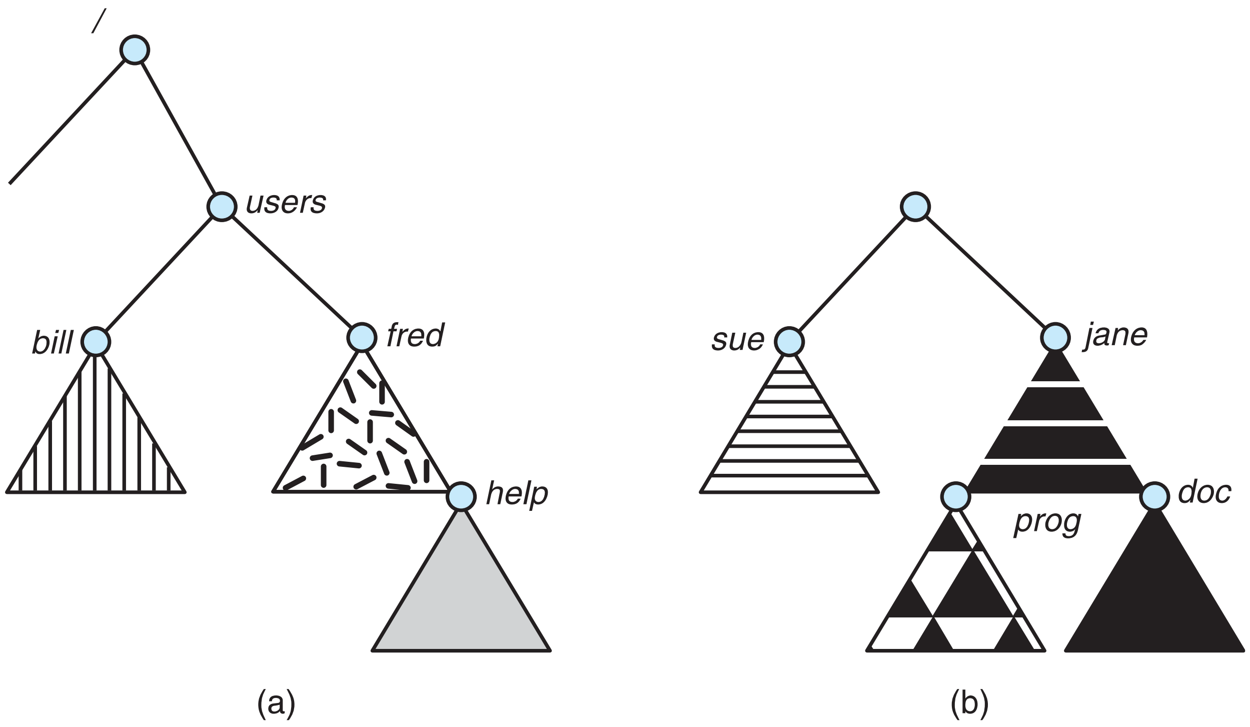
Acyclic-Graph Directories (Cont.)

- Two different names (aliasing)
- If ***dict*** deletes ***list*** $\Rightarrow$ dangling pointer

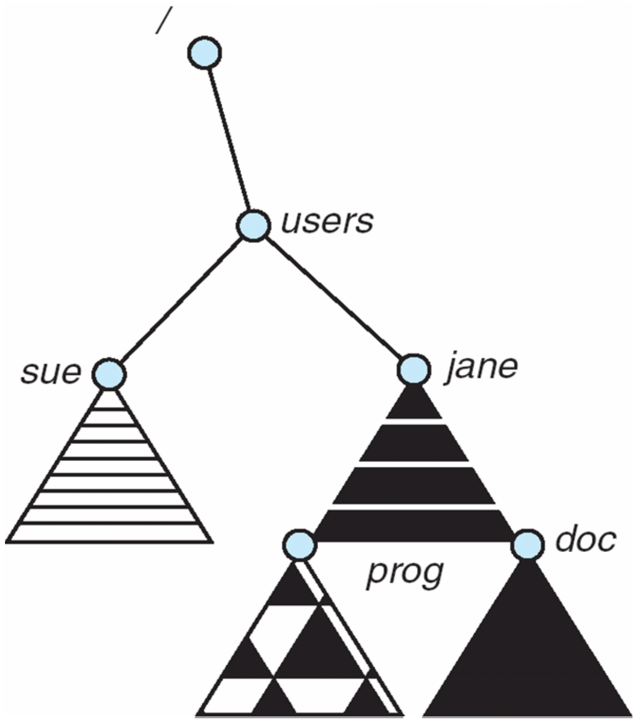
  Solutions:
  - Backpointers, so we can delete all pointers
    Variable size records a problem
  - Backpointers using a daisy chain organization
  - Entry-hold-count solution
- New directory entry type
  - **Link** – another name (pointer) to an existing file
  - **Resolve the link** – follow pointer to locate the file

File System Mounting

- A file system must be **mounted** before it can be accessed
- A unmounted file system (i.e., Fig. 11-11(b)) is mounted at a **mount point**



Mount Point



File Sharing

- Sharing of files on multi-user systems is desirable
- Sharing may be done through a **protection** scheme
- On distributed systems, files may be shared across a network
- Network File System (NFS) is a common distributed file-sharing method
- If multi-user system
  - **User IDs** identify users, allowing permissions and protections to be per-user
    Group IDs allow users to be in groups, permitting group access rights
  - Owner of a file / directory
  - Group of a file / directory

File Sharing – Remote File Systems

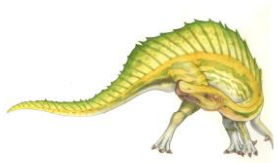
- Uses networking to allow file system access between systems
  - Manually via programs like FTP
  - Automatically, seamlessly using **distributed file systems**
  - Semi automatically via the **world wide web**
- **Client-server** model allows clients to mount remote file systems from servers
  - Server can serve multiple clients
  - Client and user-on-client identification is insecure or complicated
  - **NFS** is standard UNIX client-server file sharing protocol
  - **CIFS** is standard Windows protocol
  - Standard operating system file calls are translated into remote calls
- Distributed Information Systems (**distributed naming services**) such as LDAP, DNS, NIS, Active Directory implement unified access to information needed for remote computing

File Sharing – Failure Modes

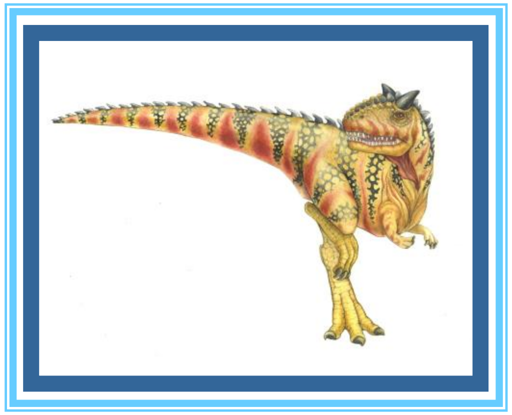
- All file systems have failure modes
  - For example corruption of directory structures or other non-user data, called **metadata**
- Remote file systems add new failure modes, due to network failure, server failure
- Recovery from failure can involve **state information** about status of each remote request
- **Stateless** protocols such as NFS v3 include all information in each request, allowing easy recovery but less security

Protection

- File owner/creator should be able to control:
  - what can be done
  - by whom
- Types of access
  - **Read**
  - **Write**
  - **Execute**
  - **Append**
  - **Delete**
  - **List**

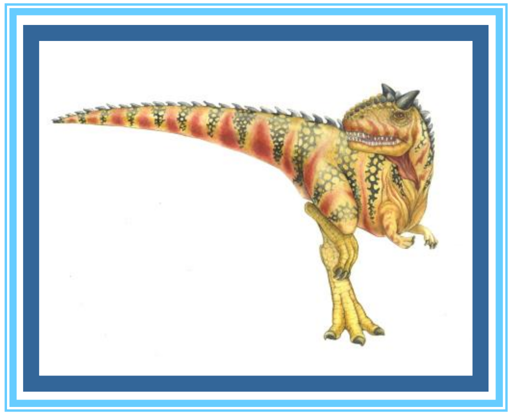


Any Questions ?



Week 15
Operating System Concepts

Muhammad Daniyal Liaquat



Topic

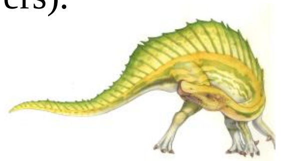
- Networking
- Security

Networking

- Windows 7 supports both peer-to-peer and client/server networking; it also has facilities for network management.
- To describe networking in Windows 7, we refer to two of the internal networking interfaces:
  - NDIS (Network Device Interface Specification) — Separates network adapters from the transport protocols so that either can be changed without affecting the other.
  - TDI (Transport Driver Interface) — Enables any session layer component to use any available transport mechanism.
- Windows 7 implements transport protocols as drivers that can be loaded and unloaded from the system dynamically.

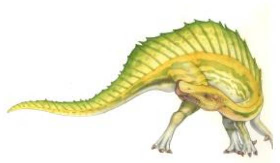
Networking — Protocols

- The server message block (SMB) protocol is used to send I/O requests over the network. It has four message types:
  1. Session control
  2. File
  3. Printer
  4. Message
- The network basic Input/Output system (NetBIOS) is a hardware abstraction interface for networks
  - Used to:
    - Establish logical names on the network
    - Establish logical connections of sessions between two logical names on the network
    - Support reliable data transfer for a session via NetBIOS requests or *SMBs.*
- Windows 7 uses the TCP/IP Internet protocol version 4 and version 6 to connect to a wide variety of operating systems and hardware platforms.
- PPTP (Point-to-Point Tunneling Protocol) is used to communicate between Remote Access Server modules running on Windows 7 machines that are connected over the Internet.
- The Data Link Control protocol (DLC) is used to access IBM mainframes and HP printers that are directly connected to the network (possible on 32-bit only versions using unsigned drivers).



Networking — Distributed Processing Mechanisms

- Windows 7 supports distributed applications via named NetBIOS, named pipes and mailslots, Windows Sockets, Remote Procedure Calls (RPC), and Network Dynamic Data Exchange (NetDDE).
- NetBIOS applications can communicate over the network using TCP/IP.
- Named pipes are connection-oriented messaging mechanism that are named via the uniform naming convention (UNC).
- Mailslots are a connectionless messaging mechanism that are used for broadcast applications, such as for finding components on the network.
- Winsock, the windows sockets API, is a session-layer interface that provides a standardized interface to many transport protocols that may have different addressing schemes.
- The Windows 7 RPC mechanism follows the widely-used Distributed Computing Environment standard for RPC messages, so programs written to use Windows 7 RPCs are very portable.
    - RPC messages are sent using NetBIOS, or Winsock on TCP/IP networks, or named pipes on LAN Manager networks.
    - Windows 7 provides the Microsoft Interface Definition Language(MIDL) to describe the remote procedure names, arguments, and results.

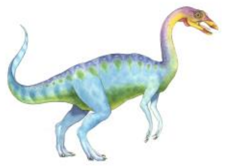


Networking — Redirectors and Servers

- In Windows 7, an application can use the Windows 7 I/O API to access files from a remote computer as if they were local, provided that the remote computer is running an MS-NET server.
- A *redirector* is the client-side object that forwards I/O requests to remote files, where they are satisfied by a server.
- For performance and security, the redirectors and servers run in kernel mode.

Networking — Domains

- NT uses the concept of a domain to manage global access rights within groups.
- A domain is a group of machines running NT server that share a common security policy and user database.
- Windows 7 provides three models of setting up trust relationships
  - *One way, A trusts B*
  - *Two way, transitive, A trusts B, B trusts C so A, B, C trust each other*
  - *Crosslink – allows authentication to bypass hierarchy to cut down on authentication traffic.*



Name Resolution in TCP/IP Networks

- On an IP network, name resolution is the process of converting a computer name to an IP address
  - e.g., `www.bell-labs.com` resolves to `135.104.1.14`
- Windows 7 provides several methods of name resolution:
  - Windows Internet Name Service (WINS)
  - broadcast name resolution (BNR)
  - domain name system (DNS)
  - a host file
  - an LMHOSTS file
- WINS consists of two or more WINS servers that maintain a dynamic database of name to IP address bindings, and client software to query the servers.
- WINS uses the Dynamic Host Configuration Protocol (DHCP), which automatically updates address configurations in the WINS database, without user or administrator intervention.

Security and Problem

- System **secure** if resources used and accessed as intended under all circumstances
  - Unachievable
- **Intruders** (**crackers**) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
- Attack can be accidental or malicious
- Easier to protect against accidental than malicious misuse

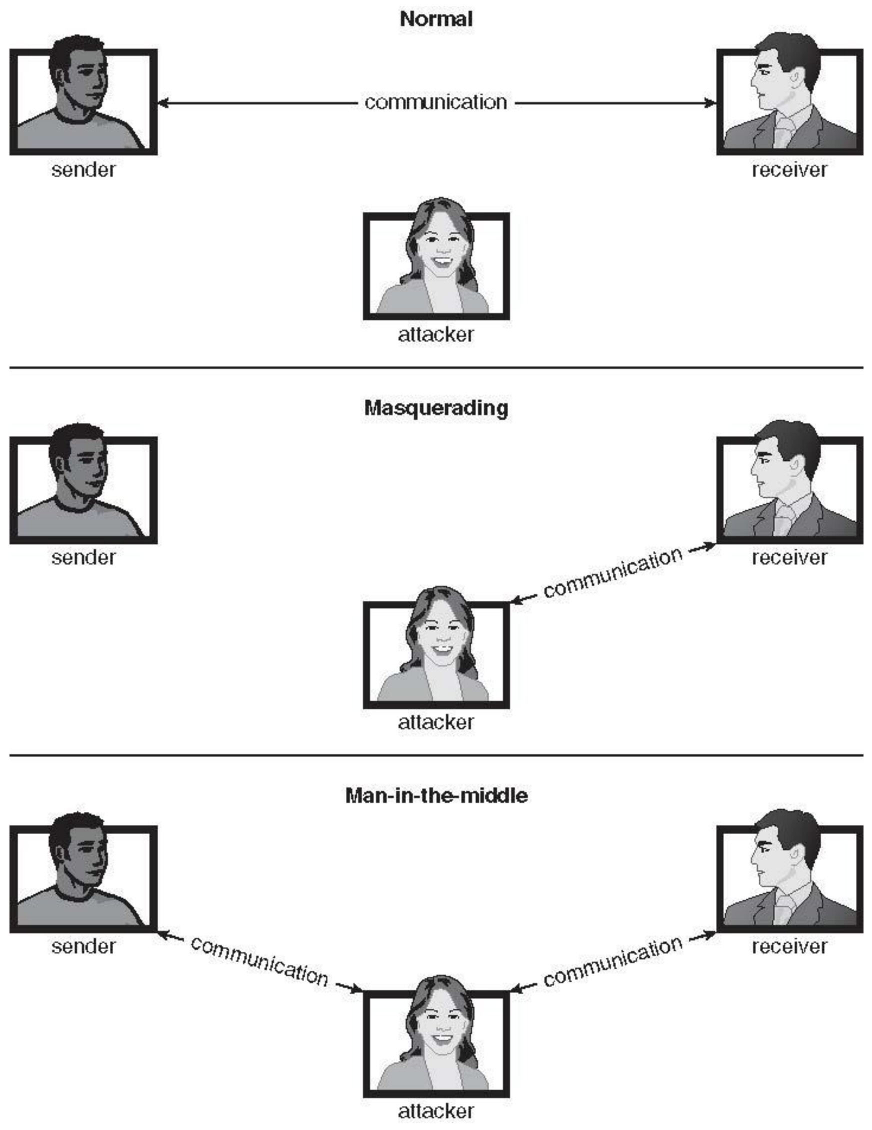
Security Violation Categories

- **Breach of confidentiality**
  - Unauthorized reading of data
- **Breach of integrity**
  - Unauthorized modification of data
- **Breach of availability**
  - Unauthorized destruction of data
- **Theft of service**
  - Unauthorized use of resources
- **Denial of service (DOS)**
  - Prevention of legitimate use

Security Violation Methods

- **Masquerading** (breach **authentication**)
  - Pretending to be an authorized user to escalate privileges
- **Replay attack**
  - As is or with **message modification**
- **Man-in-the-middle attack**
  - Intruder sits in data flow, masquerading as sender to receiver and vice versa
- **Session hijacking**
  - Intercept an already-established session to bypass authentication

Standard Security Attacks



Security Measure Levels

- Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders
- Security must occur at four levels to be effective:
  - **Physical**
    - Data centers, servers, connected terminals
  - **Human**
    - Avoid **social engineering**, **phishing**, **dumpster diving**
  - **Operating System**
    - Protection mechanisms, debugging
  - **Network**
    - Intercepted communications, interruption, DOS
- Security is as weak as the weakest link in the chain
- But can too much security be a problem?

Program Threats

- Many variations, many names
- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - **Spyware**, **pop-up browser windows**, **covert channels**
  - Up to 80% of spam delivered by spyware-infected systems
- **Trap Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler
  - How to detect them?

Program Threats (Cont.)

- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
- **Stack** and **Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

Program Threats (Cont.)

- **Viruses**
  - Code fragment embedded in legitimate program
  - Self-replicating, designed to infect other computers
  - Very specific to CPU architecture, operating system, applications
  - Usually borne via email or as a macro
  - Visual Basic Macro to reformat hard drive

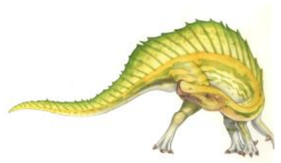
```
Sub AutoOpen()
Dim oFS
  Set oFS = CreateObject(''Scripting.FileSystemObject'')
  vs = Shell(''c:command.com /k format c:'',vbHide)
End Sub
```

Program Threats (Cont.)

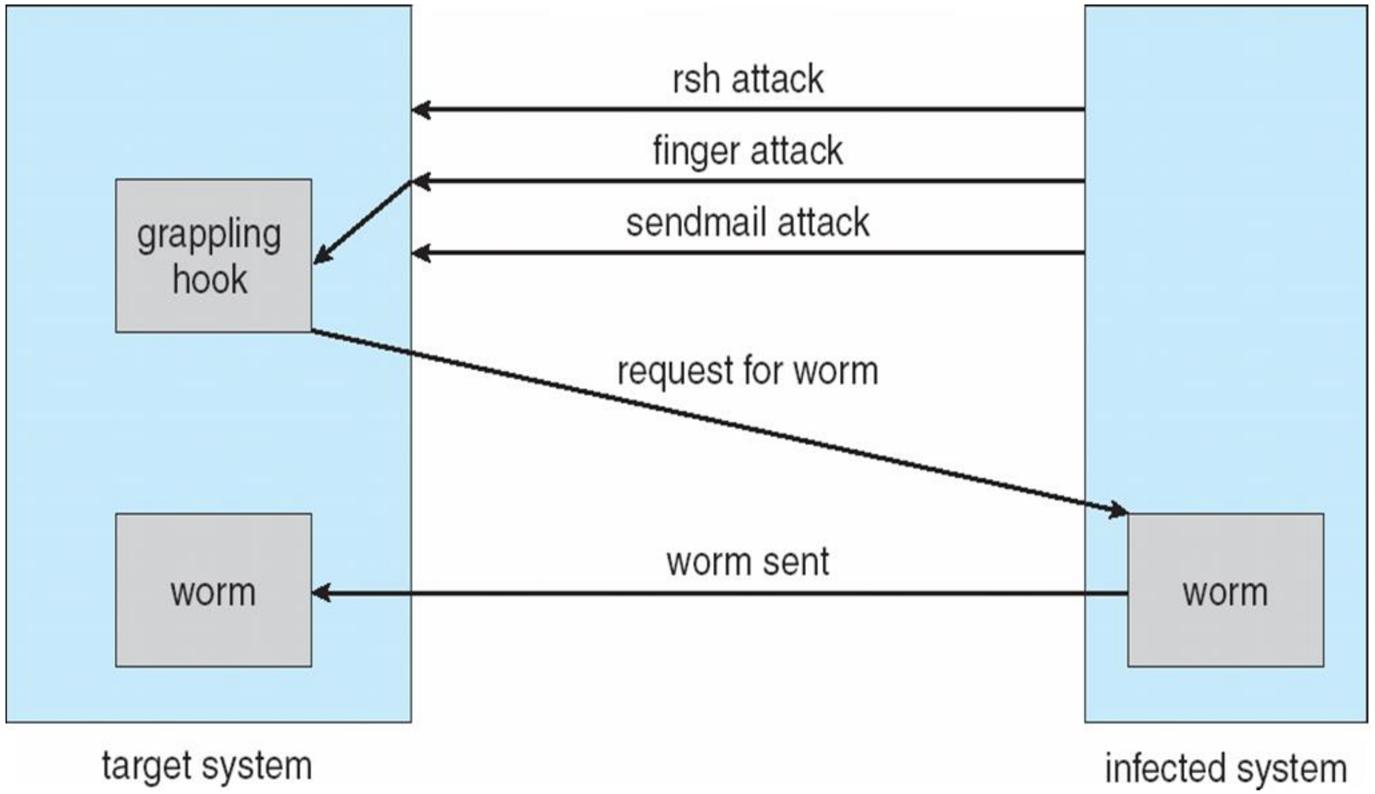
- **Virus dropper** inserts virus onto the system
- Many categories of viruses, literally many thousands of viruses
  - File / parasitic
  - Boot / memory
  - Macro
  - Source code
  - Polymorphic to avoid having a **virus signature**
  - Encrypted
  - Tunneling

System and Network Threats

- **Worms** – use **spawn** mechanism; standalone program
- Internet worm
  - Exploited UNIX networking features (remote access) and bugs in *finger* and *sendmail* programs
  - Exploited trust-relationship mechanism used by *rsh* to access friendly systems without use of password
  - **Grappling hook** program uploaded main worm program
    - 99 lines of C code
  - Hooked system then uploaded main code, tried to attack connected systems
  - Also tried to break into other users accounts on local system via password guessing
  - If target system already infected, abort, except for every 7th time



The Morris Internet Worm



System and Network Threats (Cont.)

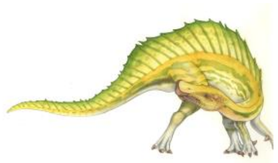
- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
  - Detection of answering service protocol
  - Detection of OS and version running on system
  - `nmap` scans all ports in a given IP range for a response
  - `nessus` has a database of protocols and bugs (and exploits) to apply against a system
  - Frequently launched from **zombie systems**
    - To decrease trace-ability

System and Network Threats (Cont.)

- **Denial of Service**
  - Overload the targeted computer preventing it from doing any useful work
  - **Distributed denial-of-service** (**DDOS**) come from multiple sites at once
  - Consider the start of the IP-connection handshake (SYN)
    - How many started-connections can the OS handle?
  - Consider traffic to a web site
    - How can you tell the difference between being a target and being really popular?
  - Accidental – CS students writing bad `fork()` code
  - Purposeful – extortion, punishment

Cryptography

- Means to constrain potential senders (*sources*) and / or receivers (*destinations*) of *messages*
  - Based on secrets (**keys**)
  - Enables
    - Confirmation of source
    - Receipt only by certain destination
    - Trust relationship between sender and receiver



Encryption

- Constrains the set of possible receivers of a message
- **Encryption** algorithm consists of
  - Set *K* of keys
  - Set *M* of Messages
  - Set *C* of ciphertexts (encrypted messages)

Symmetric Encryption

- Same key used to encrypt and decrypt
  - Therefore *k* must be kept secret
- DES was most commonly used symmetric block-encryption algorithm (created by US Govt)
  - Encrypts a block of data at a time
  - Keys too short so now considered insecure
- Triple-DES considered more secure
  - Algorithm used 3 times using 2 or 3 keys
- 2001 NIST adopted new block cipher - Advanced Encryption Standard (**AES**)
  - Keys of 128, 192, or 256 bits, works on 128 bit blocks
- RC4 is most common symmetric stream cipher, but known to have vulnerabilities
  - Encrypts/decrypts a stream of bytes (i.e., wireless transmission)
  - Key is a input to pseudo-random-bit generator
    - Generates an infinite **keystream**

Asymmetric Encryption

- **Public-key encryption** based on each user having two keys:
  - **public key** – published key used to encrypt data
  - **private key** – key known only to individual user used to decrypt data
- Must be an encryption scheme that can be made public without making it easy to figure out the decryption scheme
  - Most common is **RSA** block cipher
  - Efficient algorithm for testing whether or not a number is prime
  - No efficient algorithm is know for finding the prime factors of a number.
- Formally, it is computationally infeasible to derive $k_{d,N}$ from $k_{e,N}$, and so k_e need not be kept secret and can be widely disseminated
  - k_e is the **public key**
  - k_d is the **private key**
  - N is the product of two large, randomly chosen prime numbers p and q (for example, p and q are 512 bits each)

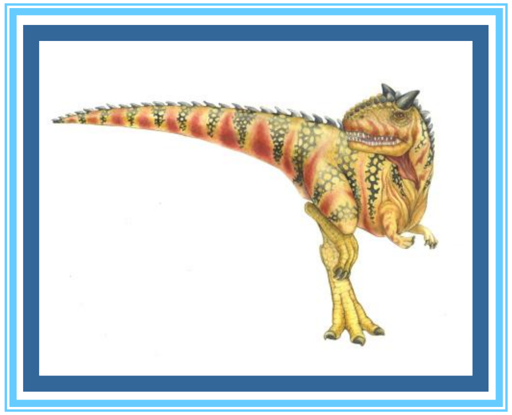
Firewalling to Protect Systems and Networks

- A network **firewall** is placed between trusted and untrusted hosts
  - The firewall limits network access between these two **security domains**
- Can be tunneled or spoofed
  - Tunneling allows disallowed protocol to travel within allowed protocol (i.e., telnet inside of HTTP)
  - Firewall rules typically based on host name or IP address which can be spoofed
- **Personal firewall** is software layer on given host
  - Can monitor / limit traffic to and from the host
- **Application proxy firewall** understands application protocol and can control them (i.e., SMTP)
- **System-call firewall** monitors all important system calls and apply rules to them (i.e., this program can execute that system call)

Computer Security Classifications

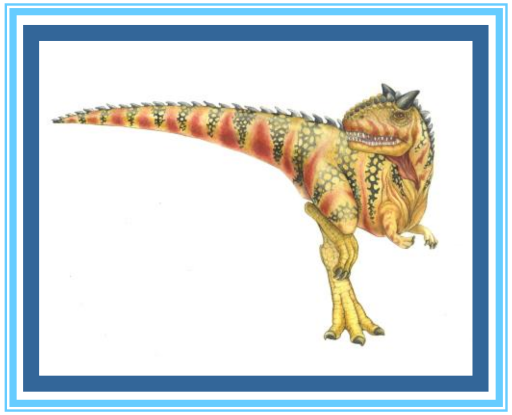
- U.S. Department of Defense outlines four divisions of computer security: **A**, **B**, **C**, and **D**
- **D** – Minimal security
- **C** – Provides discretionary protection through auditing
  - Divided into **C1** and **C2**
    - **C1** identifies cooperating users with the same level of protection
    - **C2** allows user-level access control
- **B** – All the properties of **C**, however each object may have unique sensitivity labels
  - Divided into **B1**, **B2**, and **B3**
- **A** – Uses formal design and verification techniques to ensure security

Any Questions ?



Week 16
Operating System Concepts

Muhammad Daniyal Liaquat

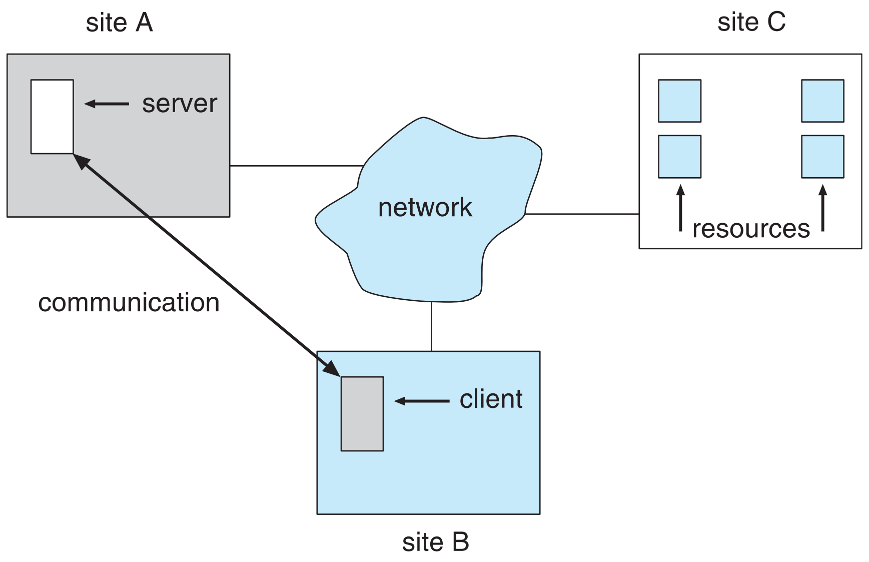


Topics

- Distributed Systems
- Distributed System Structure
- Distributed File Structures

Distributed system

- **Distributed system** is collection of loosely coupled processors interconnected by a communications network
- Processors variously called ***nodes, computers, machines, hosts***
  - ***Site*** is location of the processor
  - Generally a ***server*** has a resource a ***client*** node at a different site wants to use



Reasons for Distributed Systems

- Reasons for distributed systems
  - **Resource sharing**
    - Sharing and printing files at remote sites
    - Processing information in a distributed database
    - Using remote specialized hardware devices
  - **Computation speedup** – **load sharing** or **job migration**
  - Reliability – detect and recover from site failure, function transfer, reintegrate failed site
  - Communication – **message** passing
    - All higher-level functions of a standalone system can be expanded to encompass a distributed system
  - Computers can be downsized, more flexibility, better user interfaces and easier maintenance by moving from large system to multiple smaller systems performing distributed computing

Types of Distributed Operating Systems

- Network Operating Systems

- Distributed Operating Systems

Network-Operating Systems

- Users are aware of multiplicity of machines
- Access to resources of various machines is done explicitly by:
  - Remote logging into the appropriate remote machine (telnet, ssh)
  - Remote Desktop (Microsoft Windows)
  - Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism
- Users must change paradigms – establish a **session**, give network-based commands
  - More difficult for users

Distributed-Operating Systems

- Users not aware of multiplicity of machines
  - Access to remote resources similar to access to local resources
- **Data Migration** – transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
- **Computation Migration** – transfer the computation, rather than the data, across the system
  - Via remote procedure calls (RPCs)
  - or via messaging system

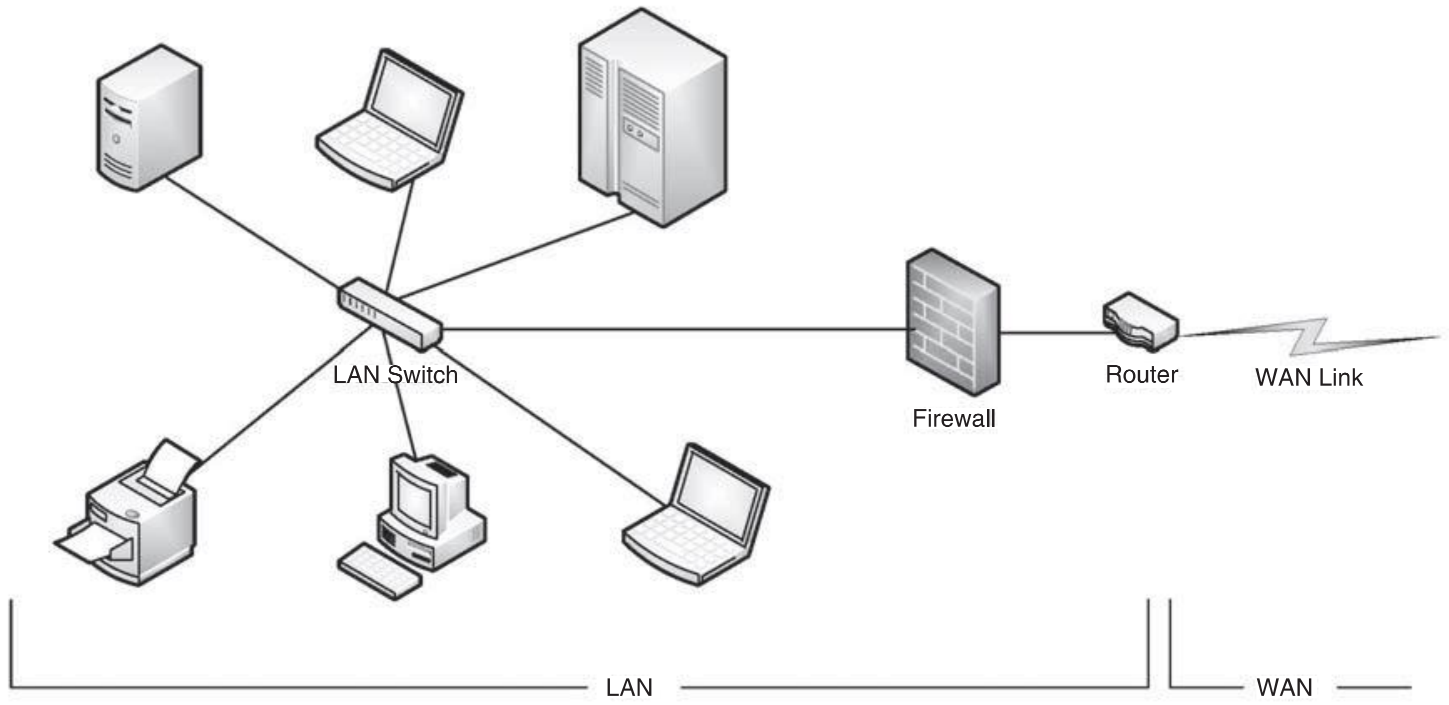
Distributed-Operating Systems (Cont.)

- **Process Migration** – execute an entire process, or parts of it, at different sites
  - **Load balancing** – distribute processes across network to even the workload
  - **Computation speedup** – subprocesses can run concurrently on different sites
  - **Hardware preference** – process execution may require specialized processor
  - **Software preference** – required software may be available at only a particular site
  - **Data access** – run process remotely, rather than transfer all data locally
- Consider the World Wide Web

Distributed System Structure of Network

- **Local-Area Network** (**LAN**) – designed to cover small geographical area
  - Multiple topologies like star or ring
  - Speeds from 1Mb per second (Appletalk, bluetooth) to 40 Gbps for fastest Ethernet over twisted pair copper or optical fibre
  - Consists of multiple computers (mainframes through mobile devices), peripherals (printers, storage arrays), routers (specialized network communication processors) providing access to other networks
  - Ethernet most common way to construct LANs
    - Multiaccess bus-based
    - Defined by standard IEEE 802.3
  - Wireless spectrum (**WiFi**) increasingly used for networking
    - I.e. IEEE 802.11g standard implemented at 54 Mbps

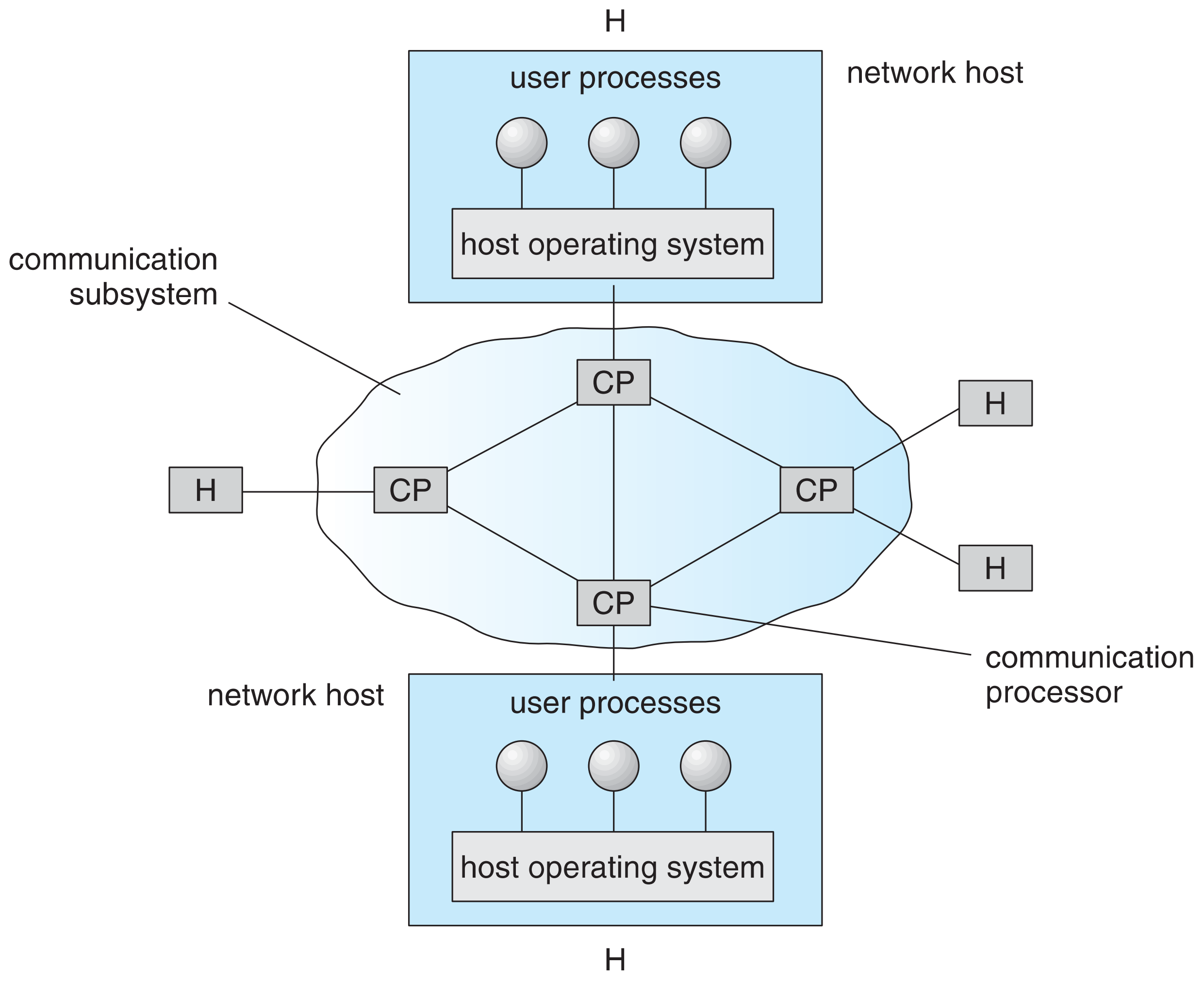
Local-area Network



Network Types (Cont.)

- **Wide-Area Network** (**WAN**) – links geographically separated sites
  - Point-to-point connections over long-haul lines (often leased from a phone company)
    - Implemented via **connection processors** known as **routers**
  - Internet WAN enables hosts world wide to communicate
    - Hosts differ in all dimensions but WAN allows communications
  - Speeds
    - T1 link is 1.544 Megabits per second
    - T3 is 28 x T1s = 45 Mbps
    - OC-12 is 622 Mbps
  - WANs and LANs interconnect, similar to cell phone network:
    - Cell phones use radio waves to cell towers
    - Towers connect to other towers and hubs

Communication Processors in a Wide-Area Network



Routing Strategies

- **Fixed routing** - A path from *A* to *B* is specified in advance; path changes only if a hardware failure disables it
  - Since the shortest path is usually chosen, communication costs are minimized
  - Fixed routing cannot adapt to load changes
  - Ensures that messages will be delivered in the order in which they were sent
- **Virtual routing**- A path from *A* to *B* is fixed for the duration of one session. Different sessions involving messages from *A* to *B* may have different paths
  - Partial remedy to adapting to load changes
  - Ensures that messages will be delivered in the order in which they were sent

Routing Strategies (Cont.)

- **Dynamic routing** - The path used to send a message form site *A* to site *B* is chosen only when a message is sent
  - Usually a site sends a message to another site on the link least used at that particular time
  - Adapts to load changes by avoiding routing messages on heavily used path
  - Messages may arrive out of order
    - This problem can be remedied by appending a sequence number to each message
  - Most complex to set up
- Tradeoffs mean all methods are used
  - UNIX provides ability to mix fixed and dynamic
  - Hosts may have fixed routes and **gateways** connecting networks together may have dynamic routes

Routing Strategies (Cont.)

- **Router** is communications processor responsible for routing messages
- Must have at least 2 network connections
- Maybe special purpose or just function running on host
- Checks its tables to determine where destination host is, where to send messages
  - Static routing – table only changed manually
  - Dynamic routing – table changed via **routing protocol**
- More recently, routing managed by intelligent software more intelligently than routing protocols
  - **OpenFlow** is device-independent, allowing developers to introduce network efficiencies by decoupling data-routing decisions from underlying network devices
- Messages vary in length – simplified design breaks them into **packets** (or **frames**, or **datagrams**)
- **Connectionless message** is just one packet
  - Otherwise need a connection to get a multi-packet message from source to destination.

Connection Strategies

- **Circuit switching** - A permanent physical link is established for the duration of the communication (i.e., telephone system)
- **Message switching** - A temporary link is established for the duration of one message transfer (i.e., post-office mailing system)
- **Packet switching** - Messages of variable length are divided into fixed-length packets which are sent to the destination
  - Each packet may take a different path through the network
  - The packets must be reassembled into messages as they arrive
- Circuit switching requires setup time, but incurs less overhead for shipping each message, and may waste network bandwidth
  - Message and packet switching require less setup time, but incur more overhead per message

Distributed File System

- **Distributed file system** (**DFS**) – a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources
- A DFS manages set of dispersed storage devices
- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces
- There is usually a correspondence between constituent storage spaces and sets of files
- Challenges include:
  - Naming and Transparency
  - Remote File Access

DFS Structure

- **Service** – software entity running on one or more machines and providing a particular type of function to a priori unknown clients
- **Server** – service software running on a single machine
- **Client** – process that can invoke a service using a set of operations that forms its client interface
- A client interface for a file service is formed by a set of primitive file operations (create, delete, read, write)
- Client interface of a DFS should be transparent, i.e., not distinguish between local and remote files
- Sometimes lower level **intermachine** interface need for cross-machine interaction

Naming and Transparency

- **Naming** – mapping between logical and physical objects
- **Multilevel mapping** – abstraction of a file that hides the details of how and where on the disk the file is actually stored
- A **transparent** DFS hides the location where in the network the file is stored
- For a file being **replicated** in several sites, the mapping returns a set of the locations of this file's replicas; both the existence of multiple copies and their location are hidden

Naming Structures

- **Location transparency** – file name does not reveal the file's physical storage location
- **Location independence** – file name does not need to be changed when the file's physical storage location changes

Naming Schemes — Three Main Approaches

- Files named by combination of their host name and local name; guarantees a unique system-wide name
- Attach remote directories to local directories, giving the appearance of a coherent directory tree; only previously mounted remote directories can be accessed transparently
- Total integration of the component file systems
  - A single global name structure spans all the files in the system
  - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable
- In practice most DFSs use static, location-transparent mapping for user-level names
  - Some support file migration
  - Hadoop supports file migration but without following POSIX standards

Remote File Access

- **Remote-service mechanism** is one transfer approach
- Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally
  - If needed data not already cached, a copy of data is brought from the server to the user
  - Accesses are performed on the cached copy
  - Files identified with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches
  - **Cache-consistency problem** – keeping the cached copies consistent with the master file
    - Could be called **network virtual memory**

The End of the Course..
Any Questions ?