GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A+ Grade by NAAC
3.46 Score
12-B Status from UGC

OPERATING SYSTEMS

DEADLOCKS

Class Presentations on Operating System

by Subhash Chand Agrawal

Resources

- A resource used by process can be a piece of hardware such as
  - Memory
  - CPU cycle
  - Printer
  - CD ROM

or a piece of information such as

  - a file
  - a record within a file
  - a shared variable
  - a critical section
  - etc.

Types of Resources

Resources come in two flavors: preemptable and nonpreemptable.

A **preemptable resource** is one which can be allocated to a given process for a period of time, then be allocated to another process and then be reallocated to the first process without any ill effects.

Examples of preemptable resources include:

- memory
- buffers
- CPU
- etc.

- A **nonpreemptable resource** cannot be taken from one process and given to another without side effects.
- One obvious example is a printer: certainly we would not want to take the printer away from one process and give it to another in the middle of a print job

DEADLOCKS

EXAMPLES:

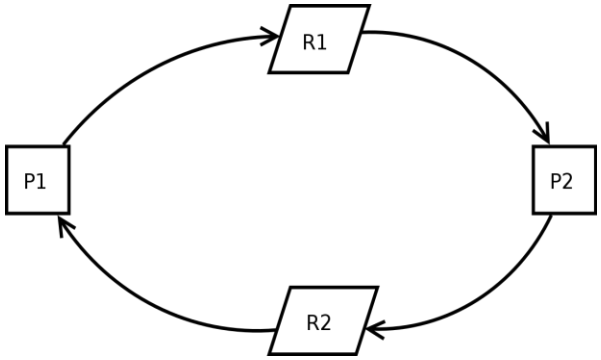
- You can't get a job without experience; you can't get experience without a job.

BACKGROUND:

Each resource type R_i has W_i instances.

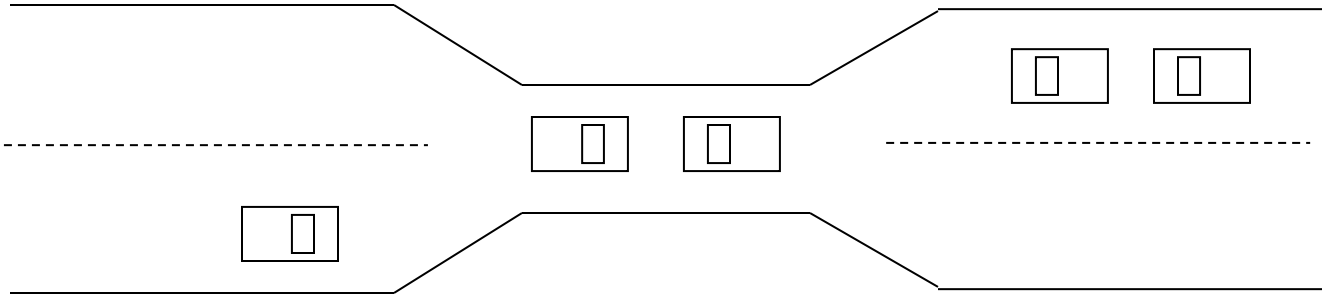
Under normal operation, a resource allocations proceed like this:

1. Request a resource (suspend until available if necessary).
2. Use the resource.
3. Release the resource.



Bridge Crossing Example

DEADLOCKS



- 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.

Q1.

- A single-lane bridge connects the two villages X and Y. Farmers in the two villages use this bridge to deliver their produce to the neighboring town. The bridge can become deadlocked if both a X and a Y farmer get on the bridge at the same time. Using exactly one semaphore, design an algorithm that prevents deadlock. Do not be concerned about starvation and inefficiency.

Solution

```
semaphore ok_to_cross = 1;
void enter_bridge()
 {
P(ok_to_cross);
 }
void exit_bridge()
{
V(ok_ to_cross);
 }
```

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A+ Grade by NAAC 3.46 Score
12-B Status from UGC

DEADLOCK CHARACTERISATION

NECESSARY CONDITIONS

ALL of these four **must** happen simultaneously for a deadlock to occur:

Mutual exclusion

At least one resource must be held in a non sharable mode.

only one process at a time can use a resource.

Hold and Wait

A process holds a resource while waiting for another resource, currently held by another process.

GLA UNIVERSITY MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A+ Grade by NAAC 3.46 Score
12-B Status from UGC

DEADLOCK CHARACTERISATION

No Preemption

a resource can be released only voluntarily by the process holding it, after that process has completed its task.

Circular Wait

There exists a set $\{P_0, P_1, \ldots, P_n\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by P_2, …, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

RESOURCE ALLOCATION GRAPH

A visual (mathematical) way to determine if a deadlock has, or may occur.

G = (V, E) The graph contains nodes and edges.

- 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

RESOURCE ALLOCATION GRAPH

- **E** **request edge** – directed edge $P_i \rightarrow R_j$

  assignment edge – directed edge $R_j \rightarrow P_i$

➢An arrow from the **process** to **resource** indicates the process is **requesting** the resource.

➢An arrow from **resource** to **process** shows an instance of the resource has been **allocated** to the process.

➢ Process is a circle,

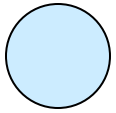
➢resource type is square;

➢dots represent number of instances of resource in type.

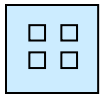
➢ Request points to square, assignment comes from dot.

Resource-Allocation Graph (Cont.)

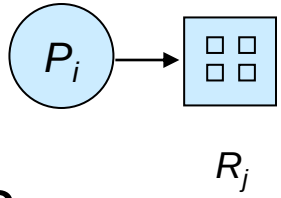
- Process



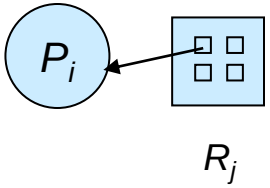
- Resource Type with 4 instances



- P_i requests instance of R_j

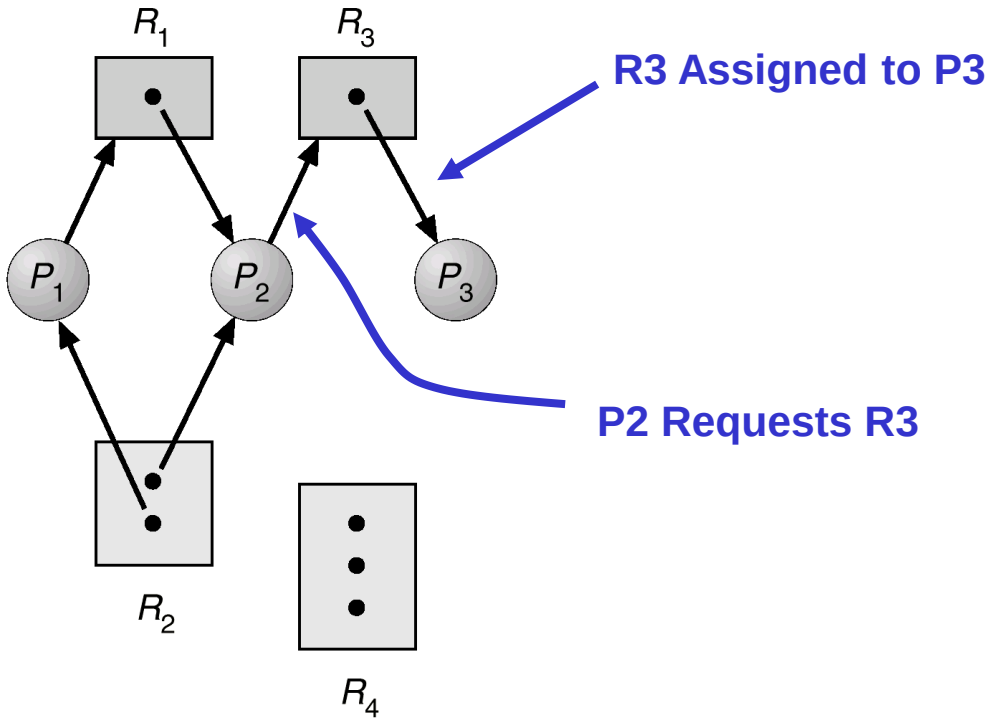


- P_i is holding an instance of R_j



RESOURCE ALLOCATION GRAPH

Resource allocation graph

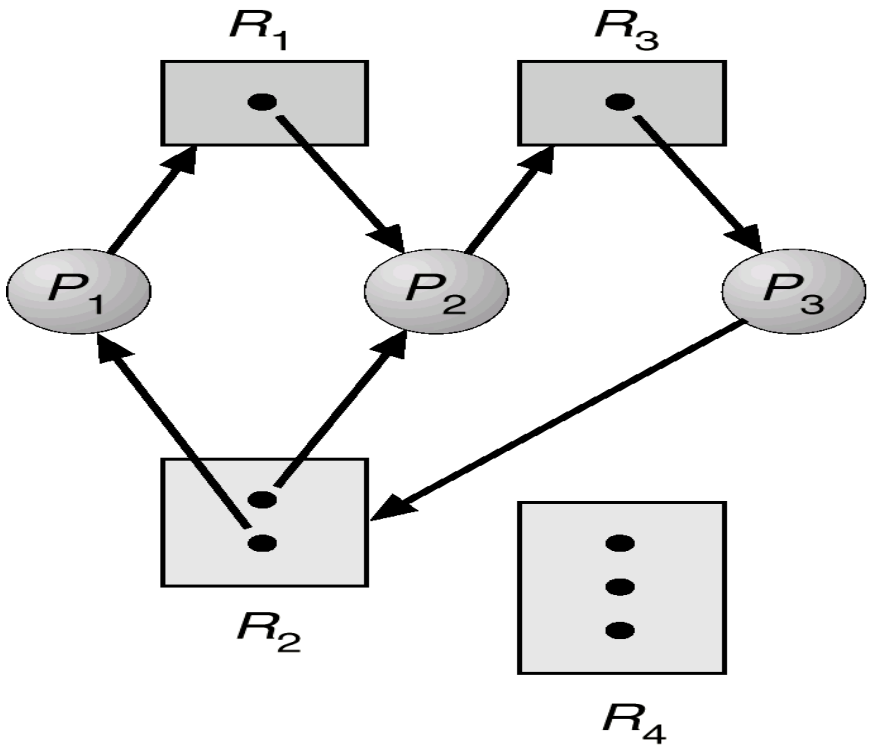


RESOURCE ALLOCATION GRAPH

- 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.

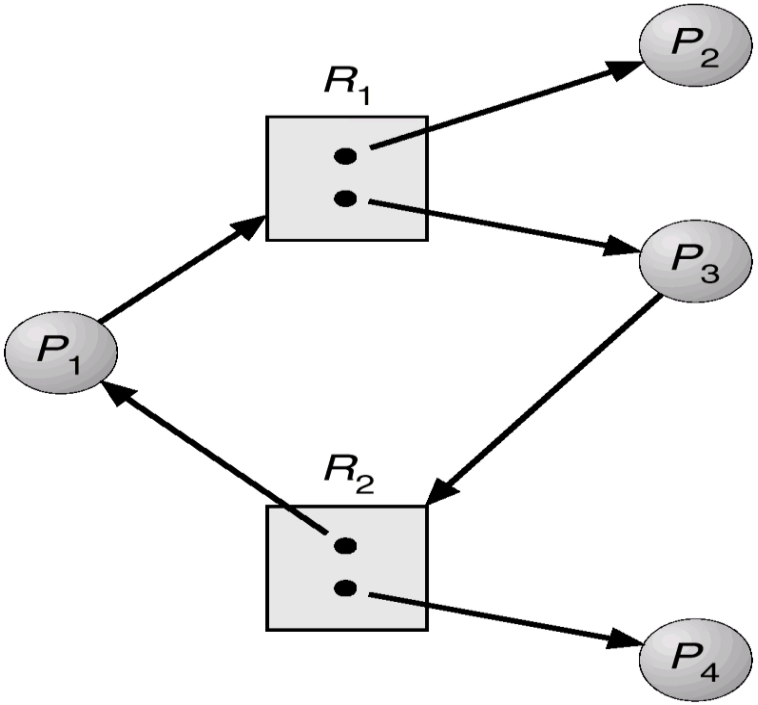
RESOURCE ALLOCATION GRAPH

Resource allocation graph with a deadlock.



RESOURCE ALLOCATION GRAPH

Resource allocation graph with a cycle but no deadlock.



HOW TO HANDLE DEADLOCKS – GENERAL STRATEGIES

There are three methods:

Ignore Deadlocks pretend that deadlocks never occur in the system :

Most Operating systems do this!!

Ensure deadlock **never** occurs using either

Prevention Prevent any one of the 4 conditions from happening.

Avoidance In this method, the request for any resource will be granted only if the resulting state of the system doesn't cause any deadlock in the system. Any process continues its execution until the system is in a safe state. Once the system enters into an unsafe state, the operating system has to take a step back.

Example: Banker's Algorithm

HOW TO HANDLE DEADLOCKS – GENERAL STRATEGIES

Allow deadlock to happen. This requires using both:

If deadlocks do occur, the operating system must detect and resolve them.

Deadlock Detection

Deadlock detection algorithms, such as the Wait-For Graph, are used to identify deadlocks, and

Deadlock Recovery

Recovery algorithms, such as the Rollback and Abort algorithm, are used to resolve them. The recovery algorithm releases the resources held by one or more processes, allowing the system to continue to make progress.

Deadlock Prevention

Do not allow one of the four conditions to occur.

Mutual Exclusion

a) Shared entities (read only files) don't need mutual exclusion (and aren’t susceptible to deadlock.)

(b) Prevention not possible, since some devices are naturally non-sharable.

(c) We cannot force a resource to be used by more than one process at the same time since it will not be fair enough and some serious problems may arise in the performance.

Deadlock Prevention

Do not allow one of the four conditions to occur.

Hold and wait:

!(Hold and wait) = !hold or !wait

a) Collect all resources before execution. (**By eliminating wait)**

b) A particular resource can only be requested when no others are being held. (**By eliminating hold)**

c) Utilization is low, starvation possible.

(Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.)

- **Challenges:**
- As a process executes instructions one by one, it cannot know about all required resources before execution.
- Releasing all the resources a process is currently holding is also problematic as they may not be usable by other processes and are released unnecessarily.
- **For example:** When Process1 releases both Resource2 and Resource3, Resource3 is released unnecessarily as it is not required by Process2.

Deadlock Prevention

No preemption:

- Deadlock arises due to the fact that a process can't be stopped once it starts.
- However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

(Consider a printer is being used by any process. If we take the printer away from that process and assign it to some other process then all the data which has been printed can become inconsistent and ineffective and also the fact that the process can't start printing again from where it has left which causes performance inefficiency.)

Deadlock Prevention

Circular wait:

a) impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration

$F: R \longrightarrow N$

Tape drive, disk drive, printers

F(Tape drive)=1

F(disk drive)=5

F(printer)=12

$F(R_j) > F(R_i)$

Deadlock Avoidance

If we have prior knowledge of how resources will be requested, it's possible to determine if we are entering an "unsafe" state.

Possible states are:

Deadlock No forward progress can be made.

Unsafe state A state that **may** allow deadlock.

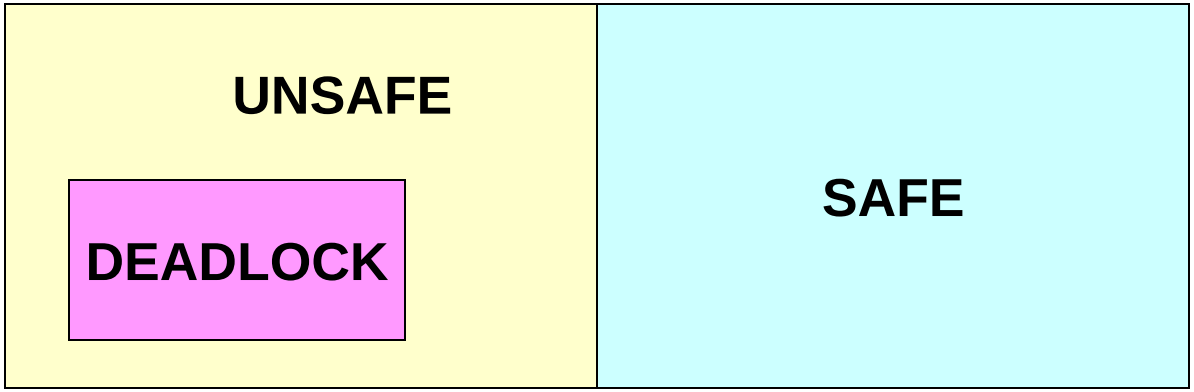
Deadlock Avoidance

Safe state A state is safe if a sequence of processes exist such that there are enough resources for the first to finish, and as each finishes and releases its resources there are enough for the next to finish.

The rule is simple: If a request allocation would cause an unsafe state, do not honor that request.

Deadlock Avoidance

NOTE: All deadlocks are unsafe, but all unsafes are NOT deadlocks.



Only with luck will processes avoid deadlock.

O.S. can avoid deadlock.

Deadlock Avoidance

Let's assume a very simple model: each process declares its maximum needs. In this case, algorithms exist that will ensure that no unsafe state is reached.

There are multiple instances of the resource in these examples.

EXAMPLE:

There exists a total of 12 tape drives. The current state looks like this:

<Available=3>

In this example, < p1, p0, p2 > is a workable sequence.

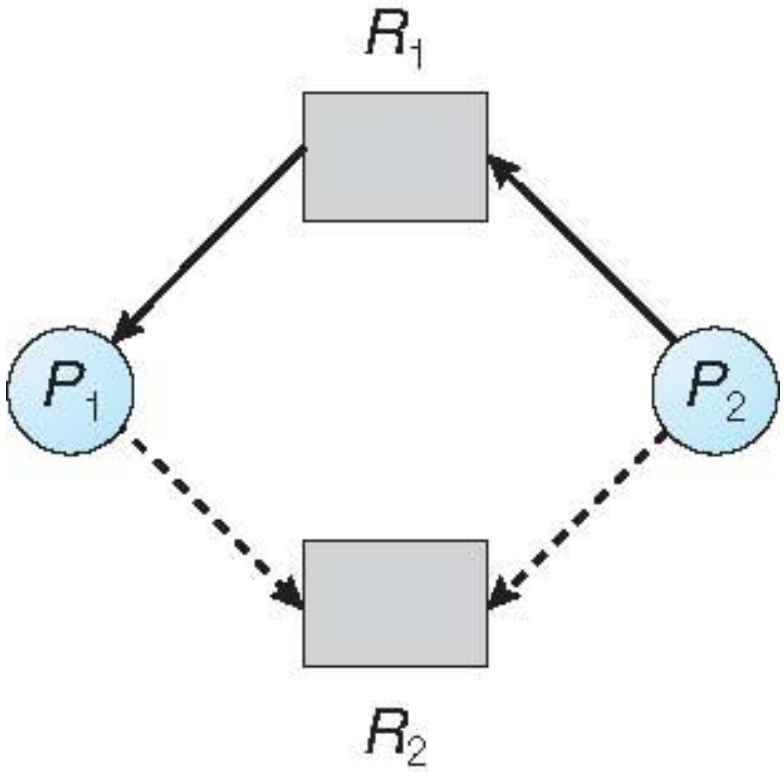
if process P2 requests and is granted one more tape drive?What happens then?

Process	Max Needs	Allocated	Current Needs
P0	10	5	5
P1	4	2	2
P2	9	2	7

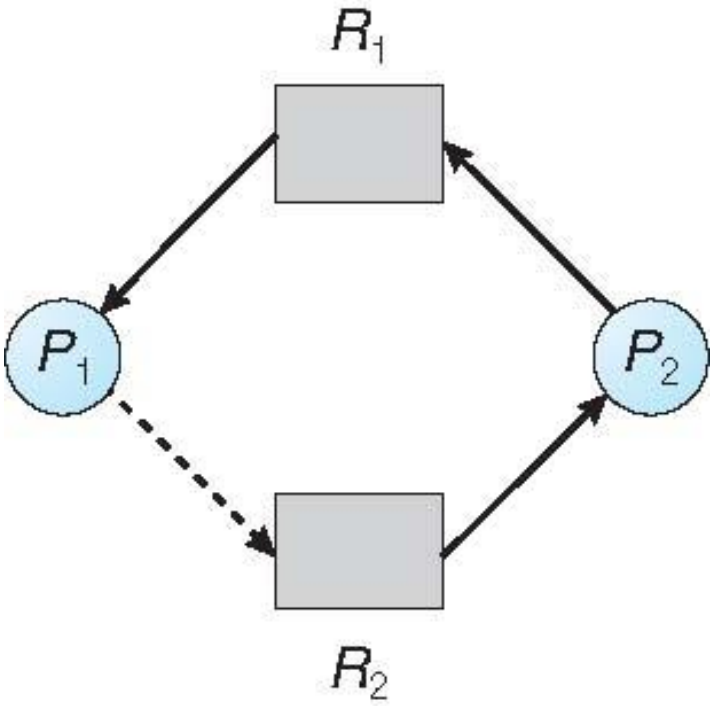
Resource-Allocation Graph Scheme

- **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

Deadlock Avoidance

Safety Algorithm

A method used to determine if a particular state is safe.

It's safe if there exists a sequence of processes such that for all the processes, there's a way to avoid deadlock:

The algorithm uses these variables:

Need[I] – the remaining resource needs of each process.

Work - Temporary variable – how many of the resource are currently available.

Deadlock Avoidance

Safety Algorithm

Finish[I] – flag for each process showing we've analyzed that process or not.

Let **work** and **finish** be vectors of length **m** and **n** respectively.

Now tell, What is the size of array or matrix?

Available?

Max need?

Allocation?

Need?

Deadlock Avoidance

Safety Algorithm

1. Initialize work = available

   Initialize finish[i] = false, for i = 0,1,2,3,..n-1

2. Find an i such that:

   finish[i] == false and need[i] <= work

   If no such i exists, go to step 4.

3. work = work + allocation[i]

   finish[i] = true

   goto step 2

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

Deadlock Avoidance

Safety Algorithm

Consider a system with: five processes, P0 → P4, three resource types, A, B, C. Type A has 10 instances, B has 5 instances, C has 7 instances. At time T0 the following snapshot of the system is taken.

Max Needs = allocated + can-be-requested

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3	3	3	2
P1	3	2	2	2	0	0	1	2	2			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

Is the system in a safe state?

<P1, P3, P4, P2, P0>

Deadlock Avoidance

Safety Algorithm

P1 requests one additional resource of type A, and two more of type C.

Request1 = (1,0,2).

Is Request1 < available?

	Max			Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	7	5	3	0	1	0	7	4	3	2	3	0
P1	3	2	2	3	0	2	0	2	0			
P2	9	0	2	3	0	2	6	0	0			
P3	2	2	2	2	1	1	0	1	1			
P4	4	3	3	0	0	2	4	3	1			

<P1, P3, P4,P0,P2>

Produce the state chart as if the request is Granted and see if it's safe.

Deadlock Avoidance

Do these examples:

P4 request additional (3,3,0)

Can the request be granted?

Request can not be granted since the resources are not available

	←	Alloc	→		←	Need	→		←	Avail	→
	A	B	C		A	B	C		A	B	C
P0	0	1	0		7	4	3		2	3	0
P1	3	0	2		0	2	0				
P2	3	0	2		6	0	0				
P3	2	1	1		0	1	1				
P4	0	0	2		4	3	1				

Deadlock Avoidance

P0 request additional (0,2,0)
Can the request be granted?

	←	Alloc	→		←	Need	→		←	Avail	→
	A	B	C		A	B	C		A	B	C
P0	0	3	0		7	2	3		2	1	0
P1	3	0	2		0	2	0				
P2	3	0	2		6	0	0				
P3	2	1	1		0	1	1				
P4	0	0	2		4	3	1				

Even resources are available, since the resulting state is unsafe

Deadlock Detection

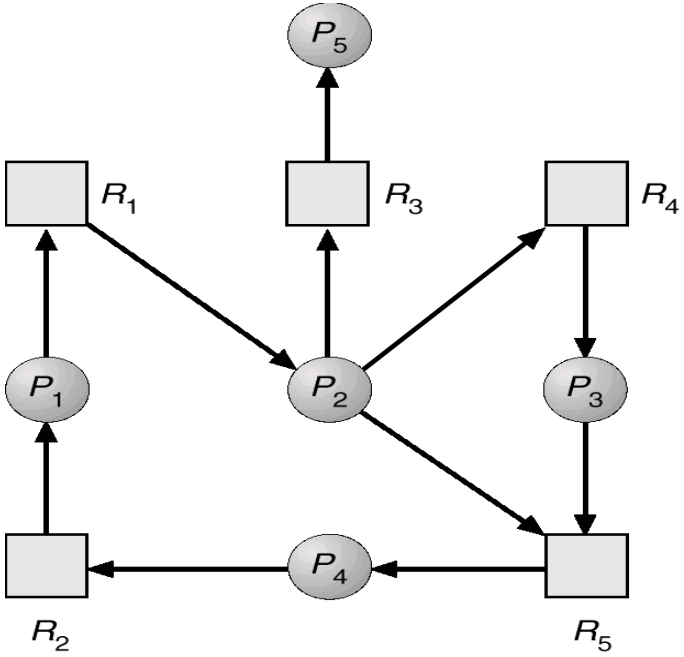
Need an algorithm that determines if deadlock occurred.

Also need a means of recovering from that deadlock.

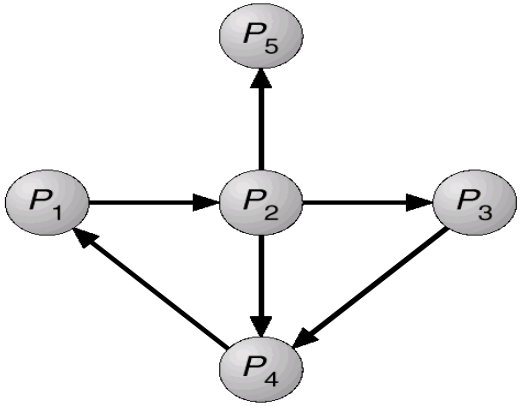
SINGLE INSTANCE OF A RESOURCE TYPE

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

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



(a)



(b)

Deadlock Detection

SEVERAL INSTANCES OF A RESOURCE TYPE

Complexity is of order m * n * n.

We need to keep track of:

available - records how many resources of each type are available.

allocation - number of resources of type m allocated to process n.

request - number of resources of type m requested by process n.

Let **work** and **finish** be vectors of length **m** and **n** respectively.

Deadlock Detection

1. Initialize work[] = available[]
   For i = 1,2,...n, if allocation[i] != 0 then
   finish[i] = false; otherwise, finish[i] = true;

2. Find an i such that:
   finish[i] == false and request[i] <= work

   If no such i exists, go to step 4.

3. work = work + allocation[i]
   finish[i] = true
   goto step 2

4. if finish[i] == false for some i, then the system is in deadlock state.
   IF finish[i] == false, then process p[i] is deadlocked.

Deadlock Detection

We have three resources, A, B, and C. A has 7 instances, B has 2 instances, and C has 6 instances. At this time, the allocation, etc. looks like this:

Is there a sequence that will allow deadlock to be avoided?

Is there more than one sequence that will work?

		←	Alloc	→		←	Req	→		←	Avail	→
		A	B	C		A	B	C		A	B	C
P0		0	1	0		0	0	0		0	0	0
P1		2	0	0		2	0	2				
P2		3	0	3		0	0	0				
P3		2	1	1		1	0	0				
P4		0	0	2		0	0	2				

Deadlock Detection

EXAMPLE

Suppose the Request matrix is changed like this. In other words, the maximum amounts to be allocated are initially declared so that this request matrix results.

Is there now a sequence that will allow deadlock to be avoided?

USAGE OF THIS DETECTION ALGORITHM

Frequency of check depends on how often a deadlock occurs and how many processes will be affected.

		←	Alloc	→		←	Req	→		←	Avail	→
		A	B	C		A	B	C		A	B	C
P0		0	1	0		0	0	0		0	0	0
P1		2	0	0		2	0	2				
P2		3	0	3		0	0	1#				
P3		2	1	1		1	0	0				
P4		0	0	2		0	0	2				

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.

Deadlock Recovery

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).

Q2.

e following snapshot of a system:

	cation				*Max*				*Available*			
	A	B	C	D	A	B	C	D	A	B	C	D
P_0	0	0	1	2	0	0	1	2	1	5	2	0
P_1	1	0	0	0	1	7	5	0				
P_2	1	3	5	4	2	3	5	6				
P_3	0	6	3	2	0	6	5	2				
P_4	0	0	1	4	0	6	5	6				

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from process *P1 arrives for (0,4,2,0), can the request be granted immediately?*

Solution

(a) The values of *Need for processes P0 through P4 respectively are (0, 0, 0, 0), (0, 7, 5, 0),* (1,0, 0, 2), (0, 0, 2, 0), and (0, 6, 4, 2).

b. Yes. With *Available being equal to (1,5, 2, 0), either process P0 or P3 could run. Once*

process P3 runs, it releases its resources, which allow all other existing processes to run.

c. Yes, it can. This results in the value of *Available being (1, 1, 0, 0). One ordering of* processes that can finish is *P0, P2, P3, P1, and P4.*

Q3.

Consider a system consisting of four resources of the same type that are shared by three processes, each of which needs at most two resources. Is this system deadlock-free? Why or why not?

Solution

Yes, this system is deadlock-free.
Proof by contradiction. Suppose the system is deadlocked. This implies that each process is holding one resource and is waiting for one more. Since there are three processes and four resources, one process must be able to obtain two resources. This process requires no more resources and, therefore it will return its resources when done.

Q4.

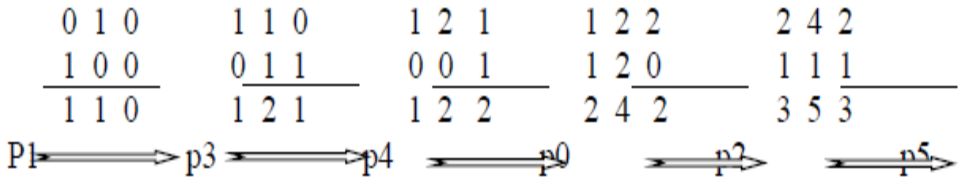
Is it possible to have a deadlock involving only a single process? Explain your answer.

- Consider we have five processes P0, P1, . . . P5 and three resources A, B, and C. Is the executing the following processes in the safe state?

Process	Allocation			Maximum need		
	A	B	C	A	B	C
P0	1	2	0	2	2	2
P1	1	0	0	1	1	0
P2	1	1	1	1	4	3
P3	0	1	1	1	1	1
P4	0	0	1	1	2	2
P5	1	0	0	1	5	1

Available		
A	B	C
0	1	0

Process	Need		
P0	1	0	2
P1	0	1	0
P2	0	3	2
P3	1	0	0
P4	1	2	1
P5	0	5	1



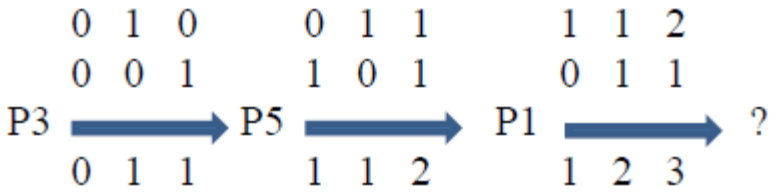
The process in safe state if they are executed in the sequence<P1, P3, P4, P0, P2, P5>

Suppose we have five processes and three resources, A, B, and C. A has 2 instances, B has 5 instances and C has 4 instances. Can the system execute the following processes without deadlock occurring, where we have the following?

Process	Maximum need			Allocation		
	A	B	C	A	B	C
P1	1	2	3	0	1	1
P2	2	2	0	0	1	0
P3	0	1	1	0	0	1
P4	3	5	3	1	2	1
P5	1	1	2	1	0	1

The available is A=0, B=1, C=0.
The current need is

Process	Current need		
	A	B	C
P1	1	1	2
P2	2	1	0
P3	0	1	0
P4	2	3	2
P5	0	1	1



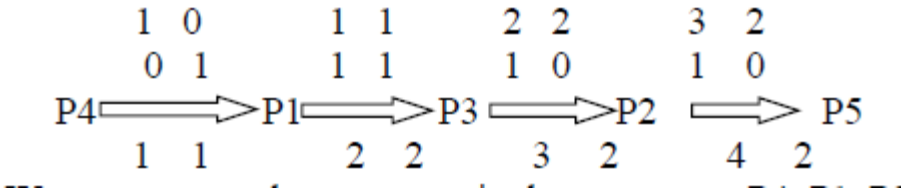
The deadlock is occurred since the available resources is less than the needs of P2 and P4

P-1

- Suppose we have two resources, A, and B. A has 6 instances and B has 3 instances. Can the system execute the following processes without deadlock occurring?

Process	Allocate		Maximum need	
	A	B	A	B
P1	1	1	2	2
P2	1	0	4	2
P3	1	0	3	2
P4	0	1	1	1
P5	2	1	6	3

Process	Need	
	A	B
P1	1	1
P2	3	2
P3	2	2
P4	1	0
P5	4	2



P-2

... means only one process at a time can use a resource

(a) Hold and wait

(b) Mutual Exclusion

(c) No preemption

(d) Circular wait

P-3

... means process acquiring at least one resource is waiting to acquire additional resources held by other processes

(a) Hold and wait

(b) Mutual Exclusion

(c) No preemption

(d) Circular wait

P-4

- ... means a resource can be released only voluntarily by the process holding it, after that process has completed its task
- (a) Hold and wait
- (b) Mutual Exclusion
- (c) No preemption
- (d) Circular wait

P-5

- What is a reusable resource?
  a) that can be used by one process at a time and is not depleted by that use
  b) that can be used by more than one process at a time
  c) that can be shared between various threads
  d) none of the mentioned
- Ans: a

```
P1                                  P2
...                                 ...
receivefrom(P2, &M2);               receivefrom(P1, &M1);
...                                 ...
sendto(P2, M1);                     sendto(P1, M2);
```

P-6

- The number of resources requested by a process _____

  a) must always be less than the total number of resources available in the system
  b) must always be equal to the total number of resources available in the system
  c) must not exceed the total number of resources available in the system
  d) must exceed the total number of resources available in the system
- Ans: C

P-7

- Deadlock prevention is a set of methods

  _____

  a) to ensure that at least one of the necessary conditions cannot hold
  b) to ensure that all of the necessary conditions do not hold
  c) to decide if the requested resources for a process have to be given or not
  d) to recover from a deadlock
- Ans: a

P-8

- For non sharable resources like a printer, mutual exclusion _____
  a) must exist
  b) must not exist
  c) may exist
  d) none of the mentioned

- Answer: a
  Explanation: A printer cannot be simultaneously shared by several processes.

P-9

- Given a priori information about the _____ number of resources of each type that maybe requested for each process, it is possible to construct an algorithm that ensures that the system will never enter a deadlock state.
  a) minimum
  b) average
  c) maximum
  d) approximate

- Ans: c

P-10

- All unsafe states are _____
  a) deadlocks
  b) not deadlocks
  c) fatal
  d) none of the mentioned

- Ans: b

P-11

- The resource allocation graph is not applicable to a resource allocation system _____
  a) with multiple instances of each resource type
  b) with a single instance of each resource type
  c) single & multiple instances of each resource type
  d) none of the mentioned
- Ans: a

P-12

- The content of the matrix Need is _____
  a) Allocation – Available
  b) Max – Available
  c) Max – Allocation
  d) Allocation – Max

- Ans: c

OPERATING SYSTEMS

Memory Management

Class Presentations on Operating System

by Subhash Chand Agrawal

Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile
- Memory hierarchy
  - small amount of fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - Gigabytes slow, cheap disk storage

Background

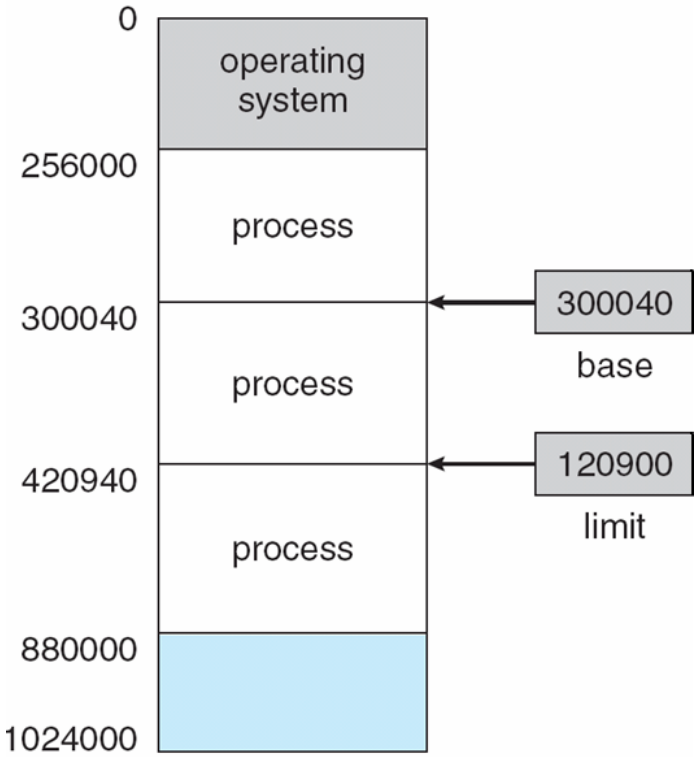
- Program must be brought (from disk) into memory and placed within a process for it to be run
- Main memory and registers are only storage CPU can access directly
- Memory unit only sees a stream of addresses + read requests, or address + data and write requests
- Register access in one CPU clock (or less)
- Main memory can take many cycles, causing a processor **stall**
- **Cache** sits between main memory and CPU registers
- Protection of memory required to ensure correct operation

Basic Hardware

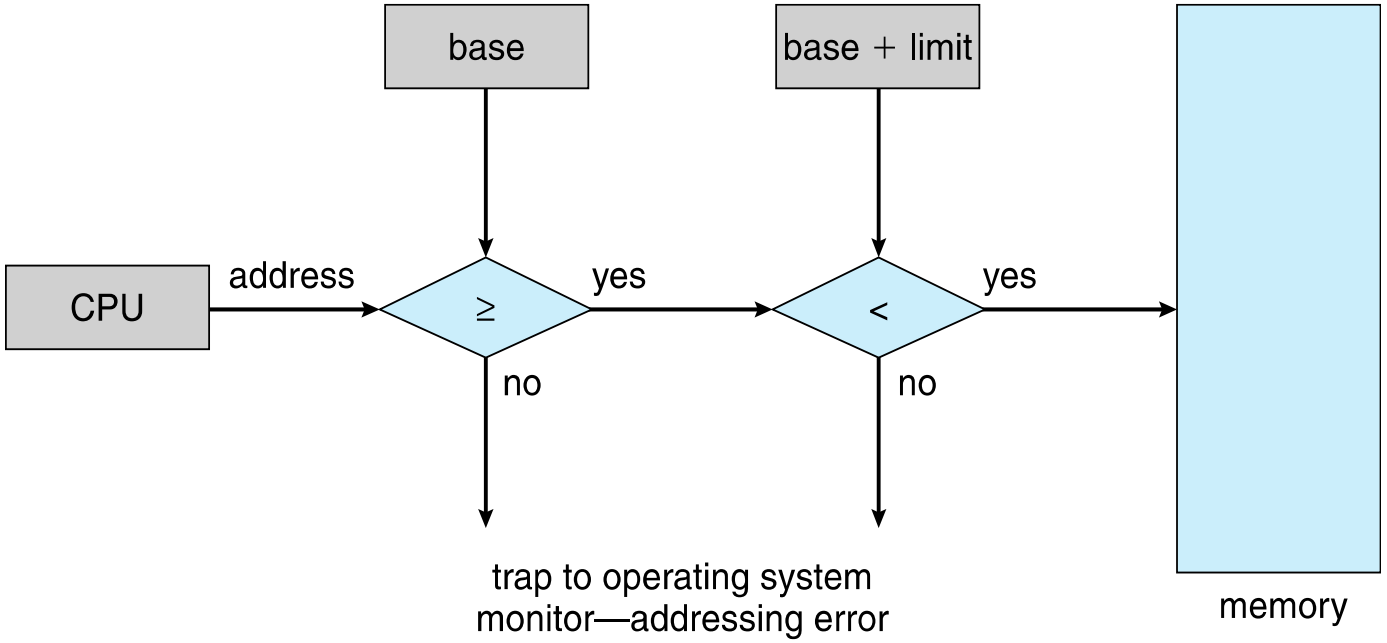
- Each process has a separate memory space
- We need to defined the range of legal addresses that a process may access.
- This is ensured by two registers:
- Base Register: Holds the smallest legal physical memory address.
- Limit Register: specifies the size of the range.
- **For e.g. if the base register holds 1250 and limit register is 356, then the program can access all addresses from 1250 to 1605(inclusive)**

Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



Hardware Address Protection



Logical vs. Physical Address Space

- **Logical address** – generated by the CPU; also referred to as **virtual address**
- **Physical address** – address seen by the memory unit

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme
- **Logical address space** is the set of all logical addresses generated by a program
- **Physical address space** is the set of all physical addresses generated by a program

If, Physical Address Space(in words) = N Words then, Physical Address (in bits) = $\log_2 N$

If, logical address space = L words Then, Logical Address = $\text{Log}_2 L$ bits

Review

- Physical Address Space = Size of the Main Memory
- If, physical address space = 4 K words then what will be the physical address?

12 bits

- Logical Address = 23 bits,then what will be the logical address space?

8 MB or 8 M words

Review

- If, physical address space = 64 KB and Let us consider word size = 8 Bytes
- Physical address space (in words)

- Physical Address =

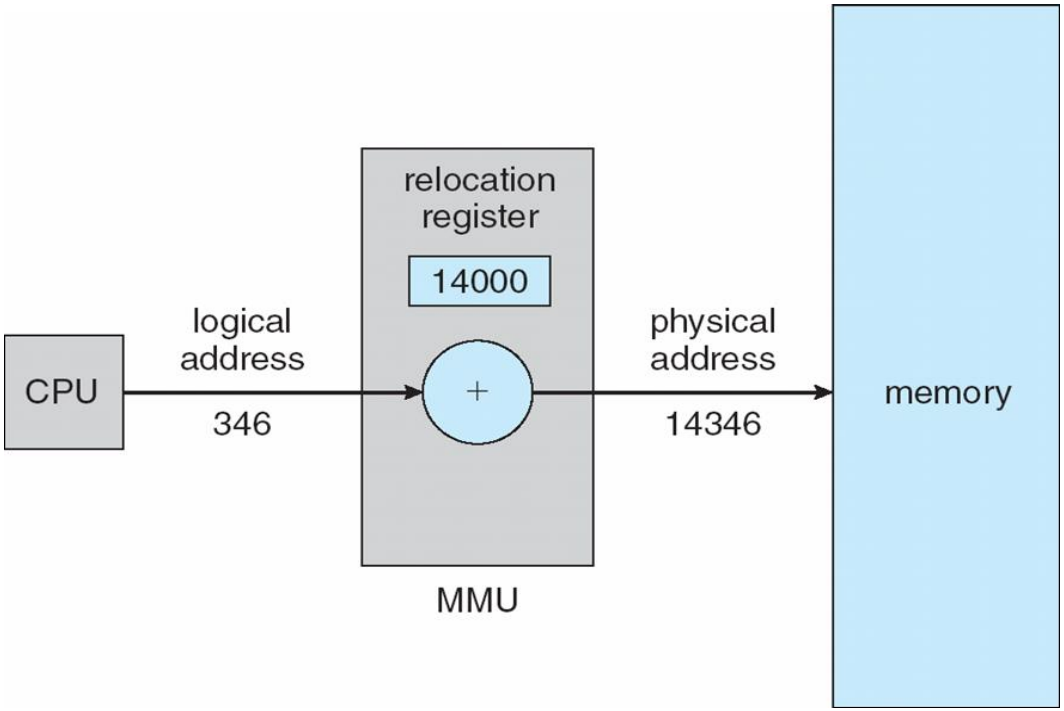
2 ^ 13 Words

13 bits

Memory-Management Unit (MMU)

- Hardware device that at run time maps virtual to physical address
- To start, consider simple scheme where the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
  - Base register now called **relocation register**
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

Dynamic relocation using a relocation register



Memory Management Techniques

- There are two Memory Management Techniques: **Contiguous**, and **Non-Contiguous**.
- In Contiguous Technique, executing process must be loaded entirely in main-memory.
- Contiguous Technique can be divided into:
  - Fixed (or static) partitioning
  - Variable (or dynamic) partitioning

Fixed Partitioning:

- The earliest and one of the simplest technique which can be used to load more than one processes into the main memory is Fixed partitioning or Contiguous memory allocation.
- In this technique, the main memory is divided into partitions of equal or different sizes.
- The operating system always resides in the first partition while the other partitions can be used to store user processes.
- In fixed partitioning,
  - The partitions cannot overlap.
  - A process must be contiguously present in a partition for the execution.

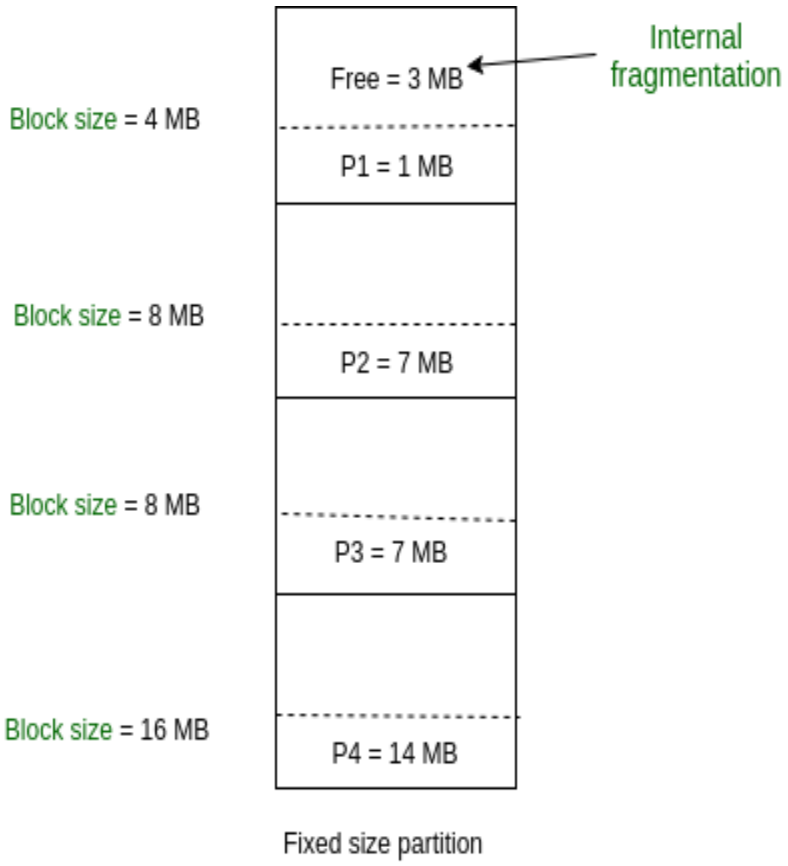
Disadvantages

Internal Fragmentation

If the size of the process is lesser then the total size of the partition then some size of the partition get wasted and remain unused. This is wastage of the memory and called internal fragmentation.

External Fragmentation

The total unused space of various partitions cannot be used to load the processes even though there is space available but not in the contiguous form.



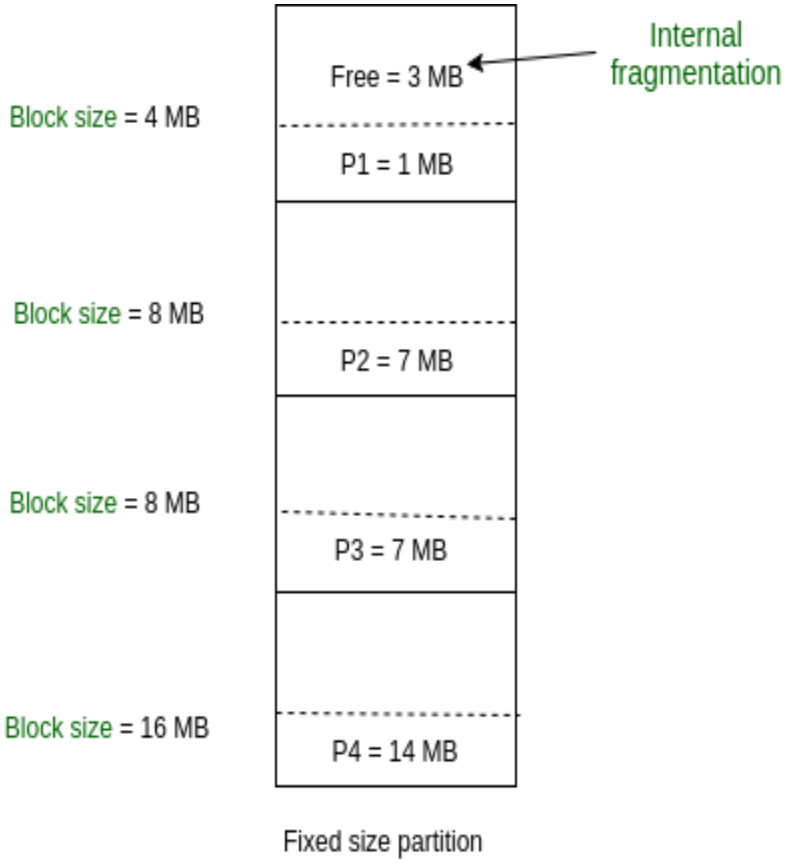
Disadvantages

Limitation on the size of the process

Process of size greater than size of partition in Main Memory cannot be accommodated.

Limitation on Degree of Multiprogramming:

Suppose if there are n1 partitions in RAM and n2 are the number of processes, then condition n2<=n1 must be fulfilled.



Dynamic Partitioning

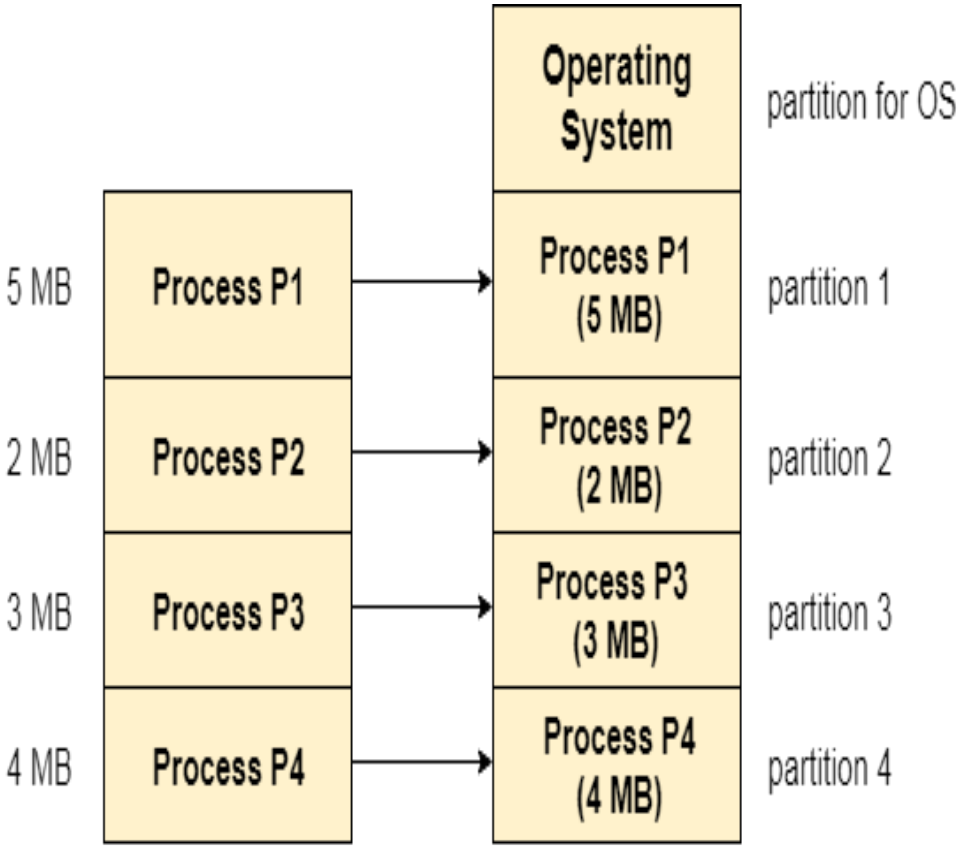
- Dynamic partitioning tries to overcome the problems caused by fixed partitioning.
- In this technique, the partition size is not declared initially.
- It is declared at the time of process loading.
- The first partition is reserved for the operating system.
- The remaining space is divided into parts.
- The size of each partition will be equal to the size of the process.
- The partition size varies according to the need of the process so that the internal fragmentation can be avoided.

Advantages of Dynamic Partitioning over fixed partitioning

No Internal Fragmentation
there will not be any unused remaining space in the partition.

No Limitation on the size of the process

Degree of multiprogramming is dynamic
Due to the absence of internal fragmentation, there will not be any unused space in the partition hence more processes can be loaded in the memory at the same time.

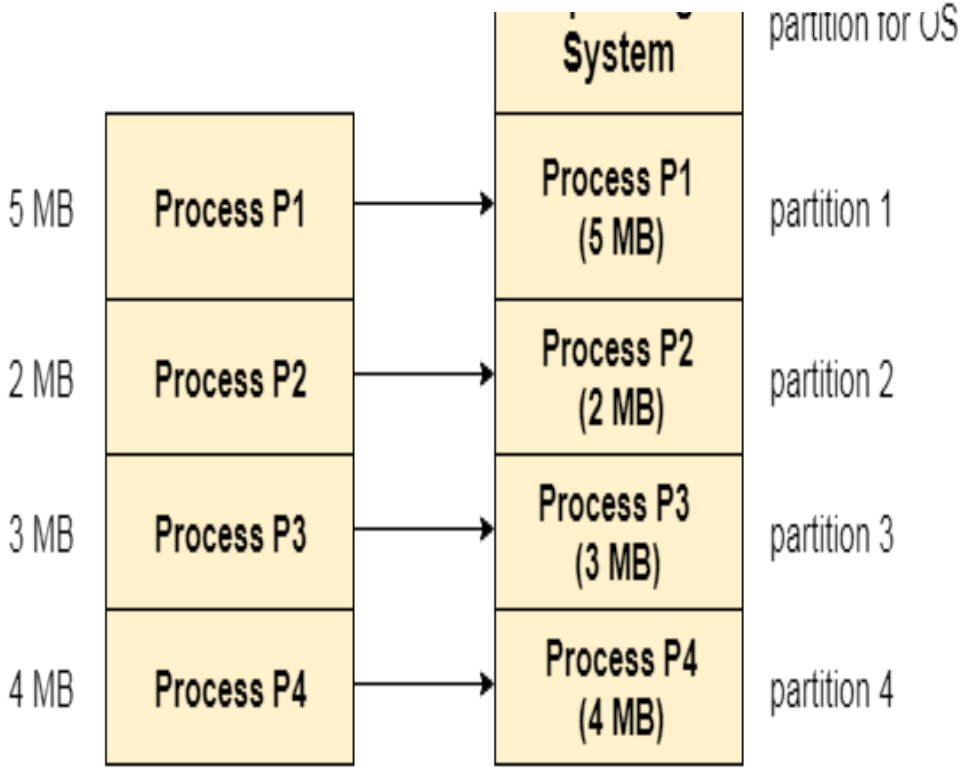


Dynamic Partitioning

(Process Size = Partition Size)

Disadvantages

External Fragmentation
Absence of internal fragmentation doesn't mean that there will not be external fragmentation.



Dynamic Partitioning

(Process Size = Partition Size)

Partitioning Algorithms

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

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

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

Example

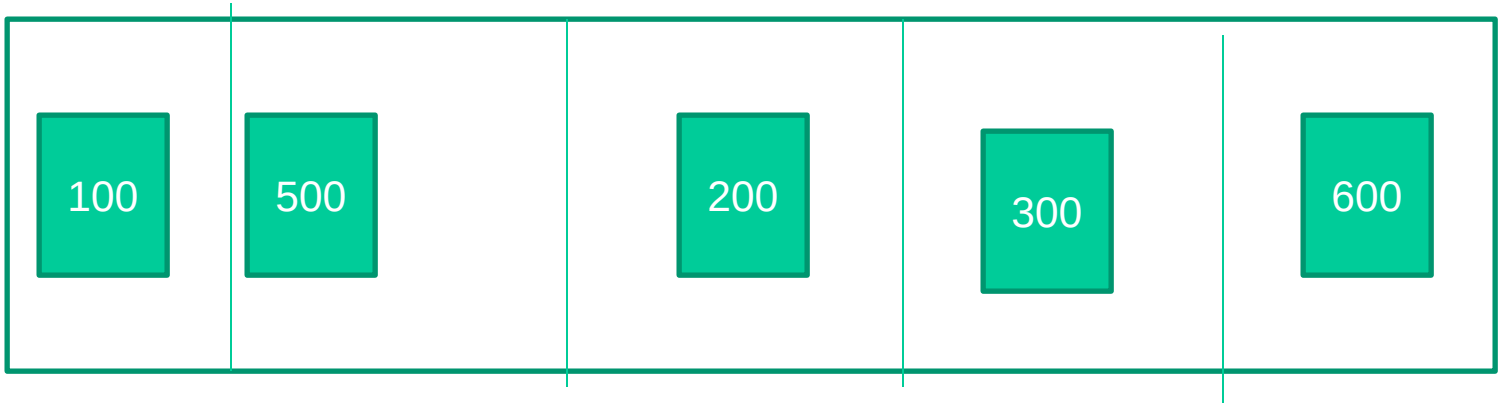
- Consider the following memory map and assume a new process P4 comes with a memory requirement of 3 KB. Locate this process.

OS
P1
<free> 10 KB
P2
<free> 16 KB
P3
<free> 4 KB

a. First fit algorithm allocates from the 10 KB block.
b. Best fit algorithm allocates from the 4 KB block.
c. Worst fit algorithm allocates from the 16 KB block

Que

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



a. First-fit:

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

b. Best-fit:

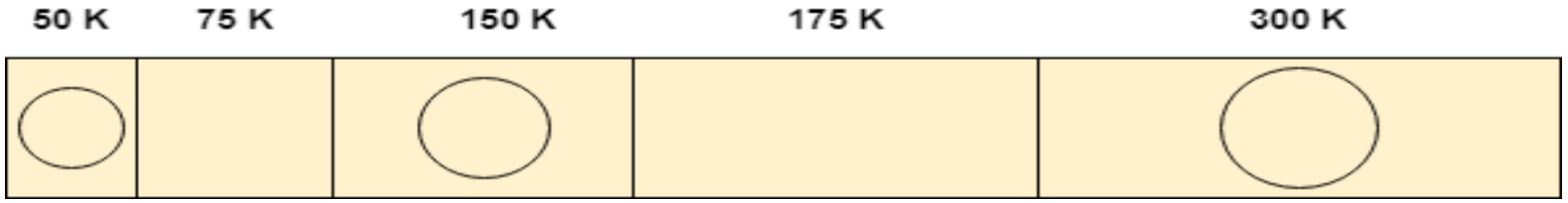
1. 212K is put in 300K partition
2. 417K is put in 500K partition
3. 112K is put in 200K partition
4. 426K is put in 600K partition

c. Worst-fit:

l. 212K is put in 600K partition
2. 417K is put in 500K partition
3. 112K is put in 388K partition
4. 426K must wait

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

Q. Process requests are given as;
25 K , 50 K , 100 K , 75 K



Determine the algorithm which can optimally satisfy this requirement.

a. First Fit algorithm
b. Best Fit Algorithm
c. Neither of the two
d. Both of them

In the question, there are five partitions in the memory. 3 partitions are having processes inside them and two partitions are holes.

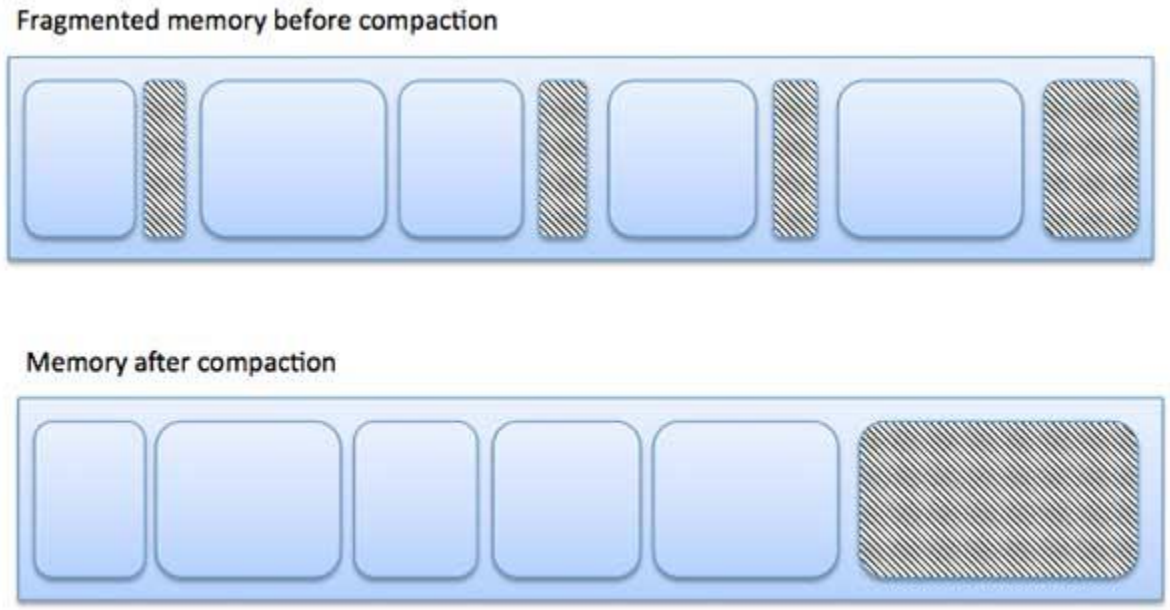
Our task is to check the algorithm which can satisfy the request optimally.

Fragmentation

- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used

Fragmentation (Cont.)

- Use compaction to minimize the probability of external fragmentation.
- In compaction, all the free partitions are made contiguous and all the loaded partitions are brought together.

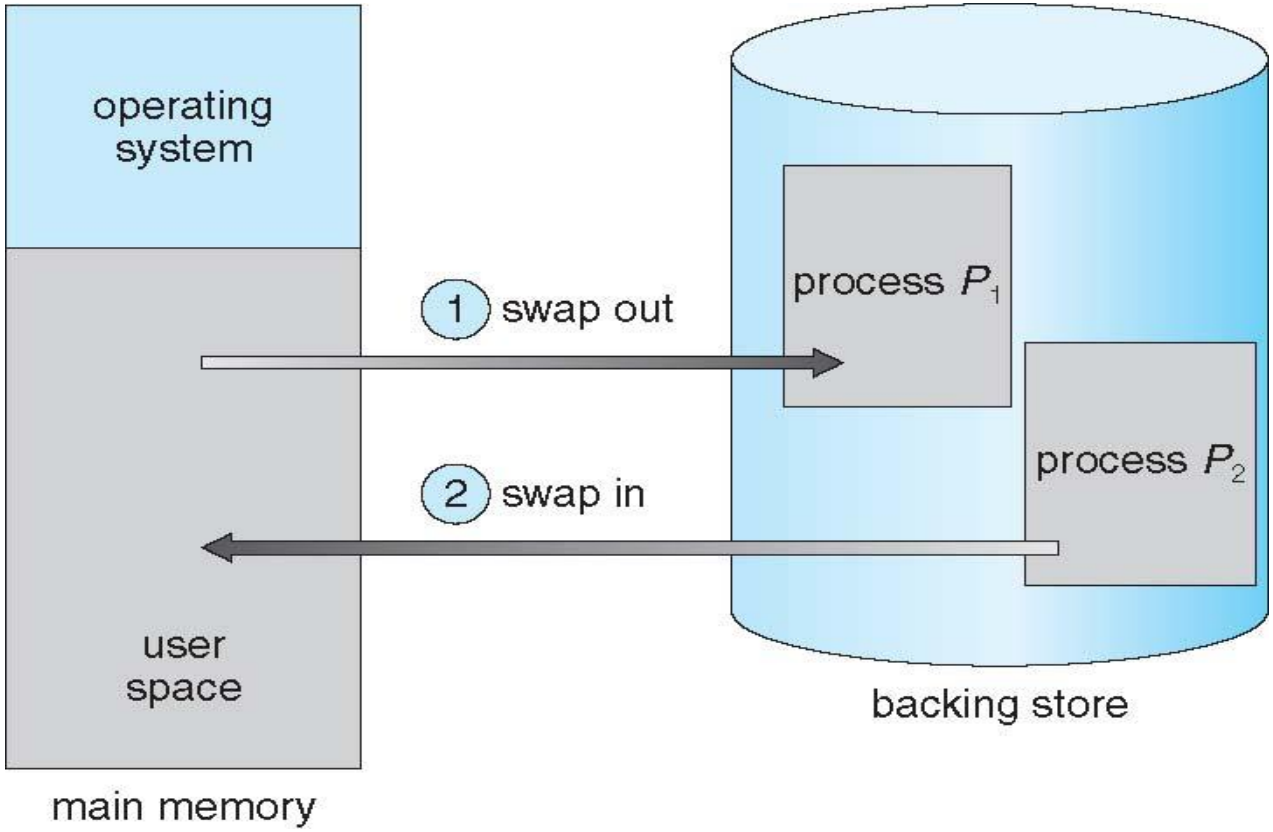


Reference:
https://www.tutorialspoint.com/operating_system/os_memory_management.htm

Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
- **Backing store** – fast disk large enough to accommodate copies of all memory images for all users;
- **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped
- System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



Paging

- In Operating Systems, Paging is a storage mechanism used to retrieve processes from the secondary storage into the main memory in the form of pages.
- Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.

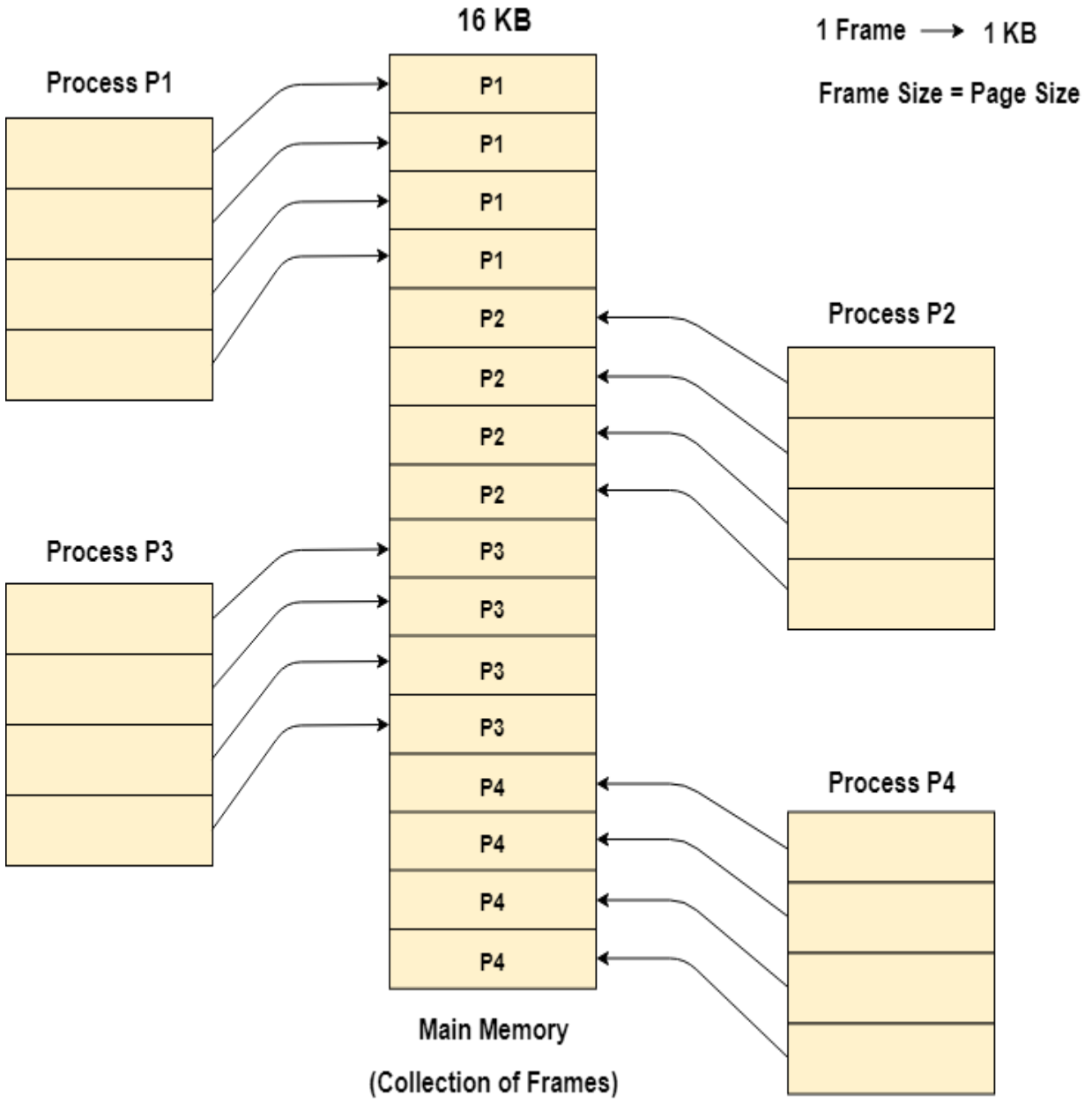
Paging

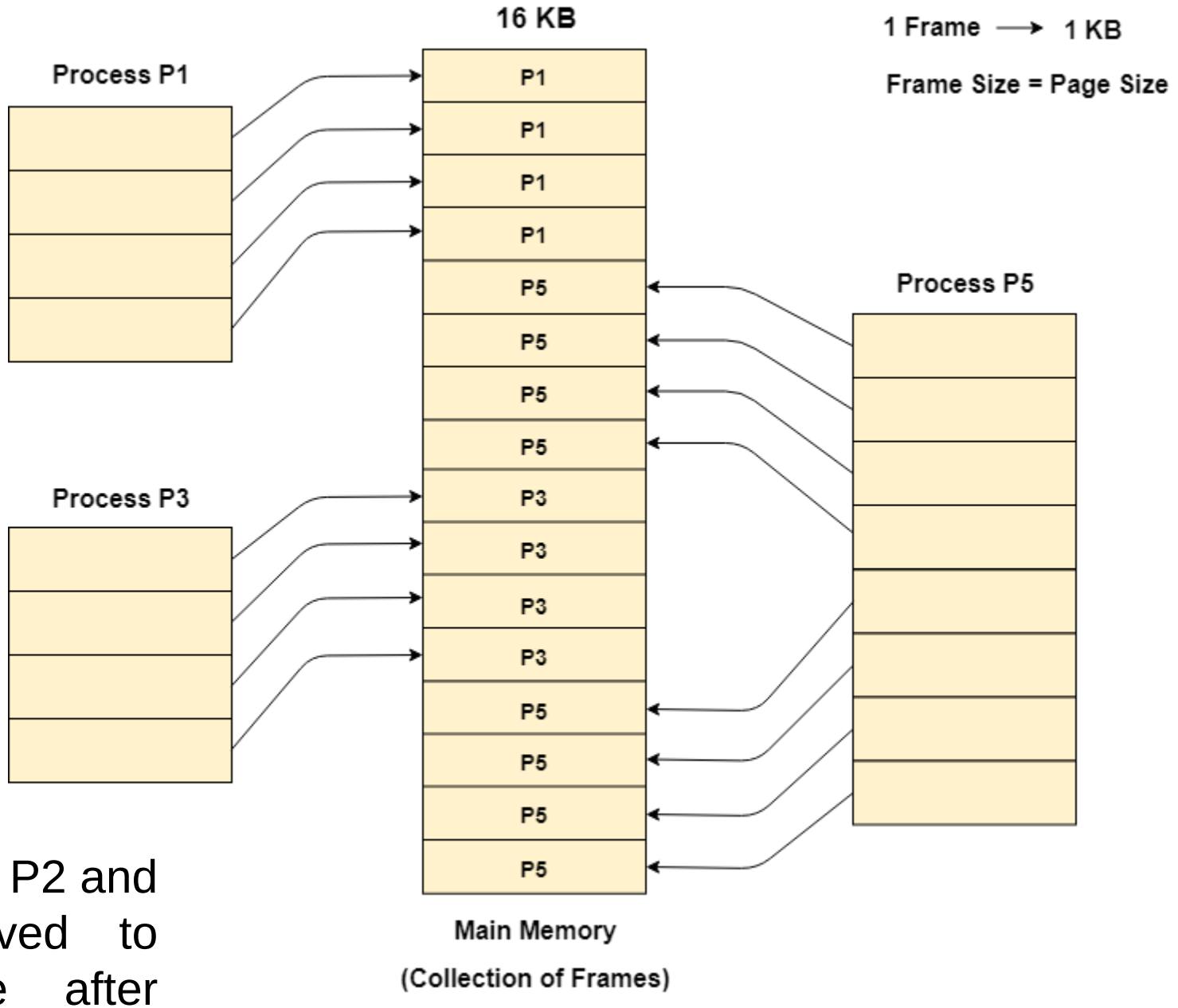
- Physical address space of a process can be noncontiguous;
  - Avoids external fragmentation
  - Avoids problem of varying sized memory chunks
- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- Keep track of all free frames

- To run a program of size ***N*** pages, need to find ***N*** free frames and load program
- Set up a **page table** to translate logical to physical addresses
- Backing store likewise split into pages
- Still have Internal fragmentation

Assume main memory size 16 Kb and Frame size is 1 KB

P1, P2, P3 and P4 of 4 KB each

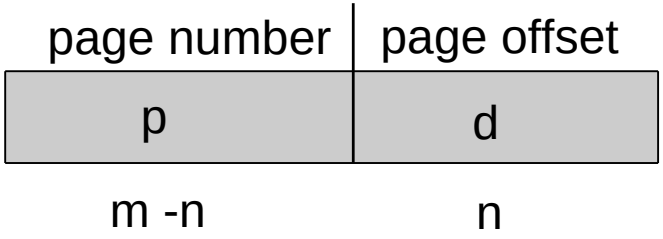




Consider that, P2 and P4 are moved to waiting state after some time.

Address Translation Scheme

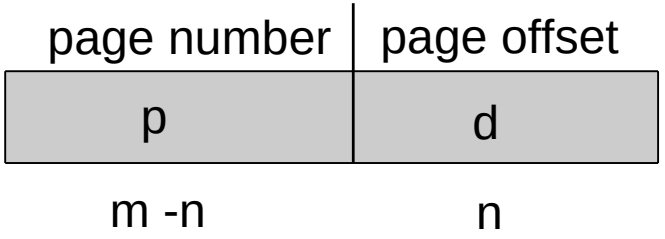
- Address generated by CPU is divided into:
  - **Page number** (***p***) – used as an index into a **page table** which contains base address of each page in physical memory
  - **Page offset** (***d***) – combined with base address to define the physical memory address that is sent to the memory unit



  - For given logical address space 2^m and page size 2^n

Terminology

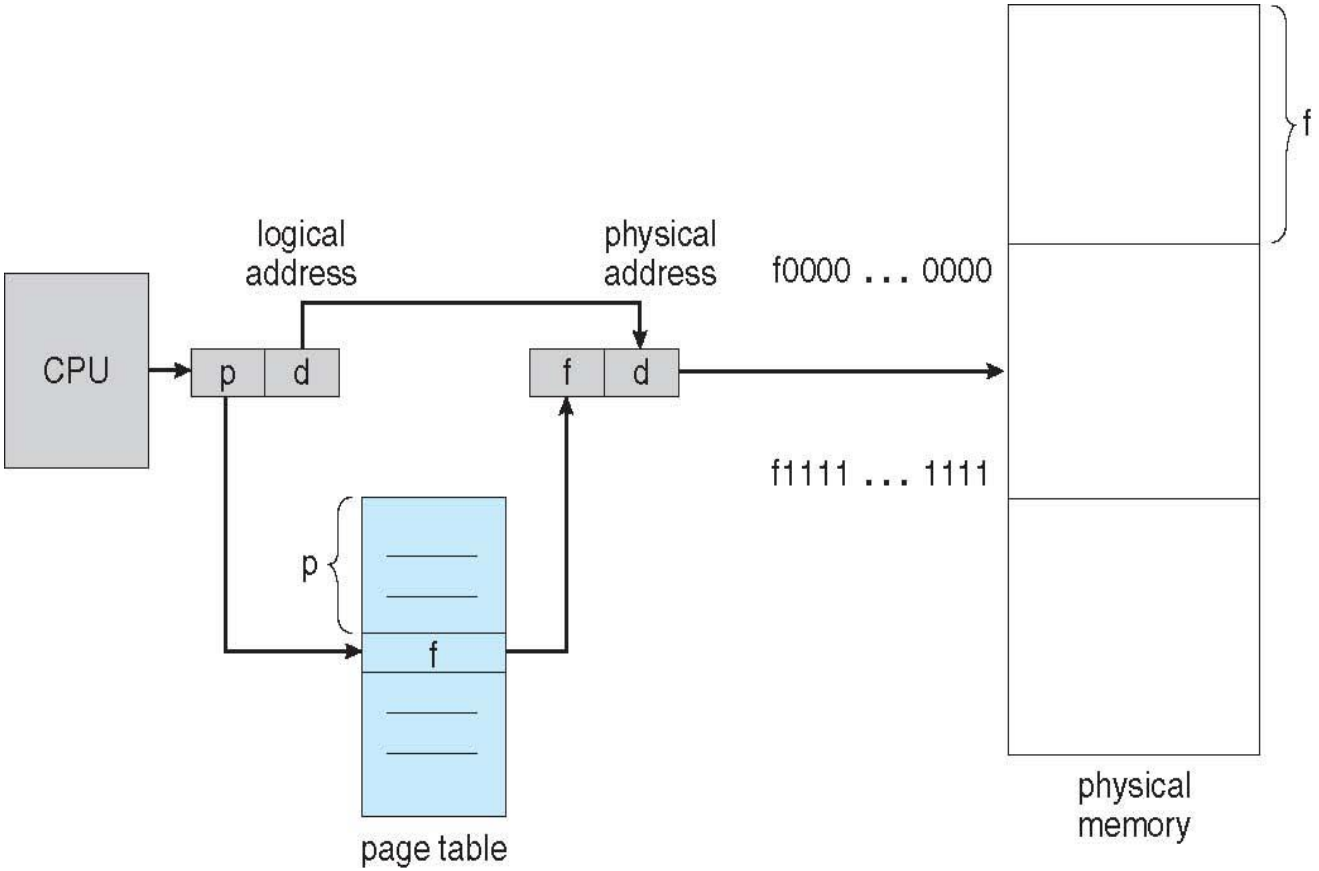
- Logical Address or Virtual Address (represented in bits): An address generated by the CPU
- Logical Address Space or Virtual Address Space(represented in words or bytes): The set of all logical addresses generated by a program
- Physical Address (represented in bits): An address actually available on memory unit
- Physical Address Space (represented in words or bytes): The set of all physical addresses corresponding to the logical addresses



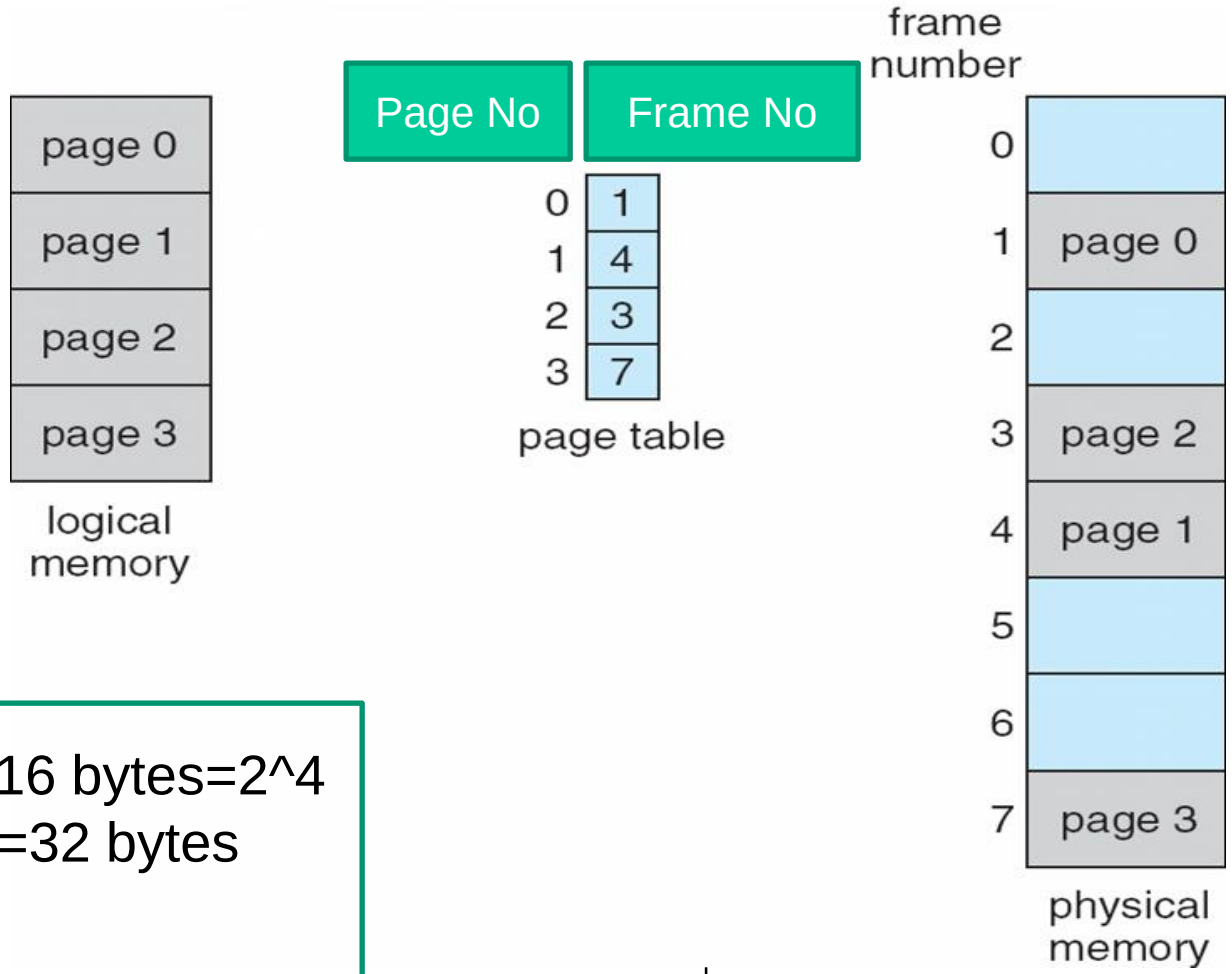
- Logical Address = 13 bits
- Physical Address = 12 bits
- Page size = frame size = 1 K words
- Logical Address Space?
- Physical Address Space
- How many pages?
- How many frames?
- What is the value of p and d?

2^13 bytes or words
2^12 bytes or words
8
4
3, 10

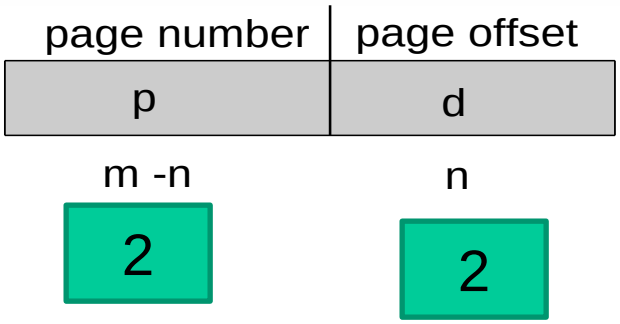
Paging Hardware



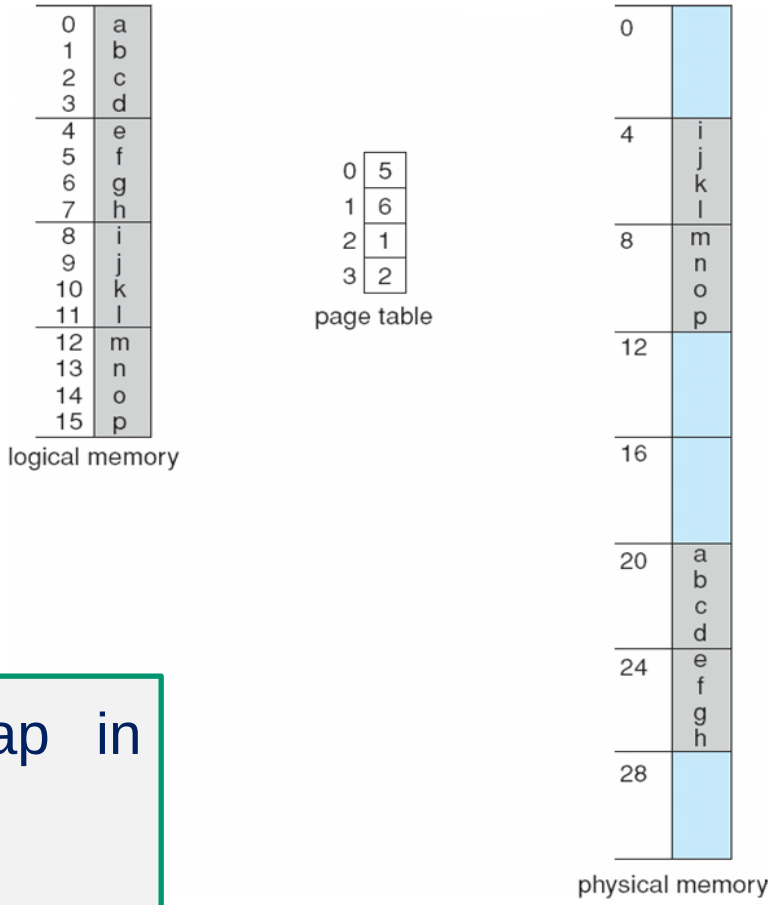
Paging Model of Logical and Physical Memory



Logical address space=16 bytes=2^4
Physical Address space=32 bytes
Page size=4 bytes=2^2
Logical Address? 4 bits
Physical address? 5 bits
No of pages=16/4=4 pages
No. of frames=32/4=8 frames



Paging Example



Logical address 0-> 0000 will map in physical address 20
Logical address 3->physical 23
Logical address 10->physical address 6
Logical address 13-> physical address 9
Logical address 4-> physical address 24

n=2 and *m*=4 32-byte memory and 4-byte pages

Paging (Cont.)

- Calculating internal fragmentation
  - Page size = 2,048 bytes
  - Process size = 72,766 bytes
  - 35 pages + 1,086 bytes
  - Internal fragmentation of 2,048 - 1,086 = 962 bytes
  - So small frame sizes desirable?

Q1

- Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.

•Number of locations possible with 22 bits = 2^{22} locations
•It is given that the size of one location = 2 bytes

Thus, Size of memory
= 2^{22} x 2 bytes
= 2^{23} bytes
= 8 MB

Q2

- Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

  Let ‘n’ number of bits are required. Then, Size of memory = 2^n x 4 bytes.
  Since, the given memory has size of 16 GB, so we have-
  2^n x 4 bytes = 16 GB
  2^n x 4 = 16 G
  2^n x 2^2 = 2^{34}
  2^n = 2^{32}
  ∴ n = 32 bits

Q3

- Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. The size of the page table in the system in megabytes is _____.

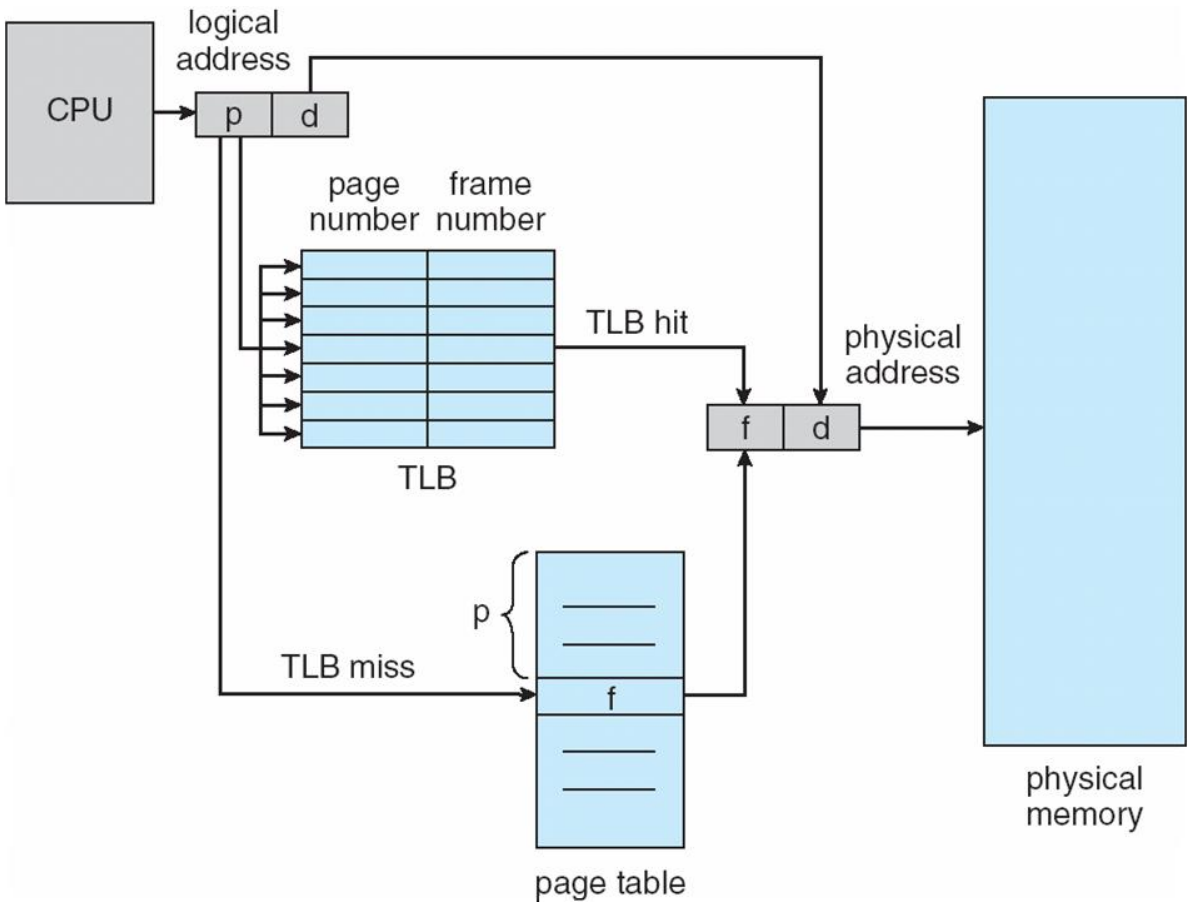
- Number of pages=$2^{32}/2^{12}=2^{20}$ pages
- Page table size
- = Number of entries in page table x Page table entry size
- $=2^{20}*4=4$ MB

Q4

- Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?
- Number of pages=$2^{32}/2^{12}=2^{20}$
- Number of frames in main memory
- = Size of main memory / Frame size
- = 64 MB / 4 KB
- = 2^{26} B / 2^{12} B
- = 2^{14}
- Thus, Number of bits in frame number = 14 bits
- Page table size
- = Number of entries in page table x Page table entry size
- = Number of entries in page table x Number of bits in frame number
- = 2^{20} x 14 bits
- = 2^{20} x 16 bits (Approximating 14 bits ≈ 16 bits)
- = 2^{20} x 2 bytes
- = 2 MB

- In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is the maximum number of bits that can be used for storing protection and other information in each page table entry?
- Number of bits in physical address = 30 bits
- Thus, Size of main memory = 2^{30} B = 1 GB
- Number of frames in main memory
- = Size of main memory / Frame size = 1 GB / 4 KB
- = 2^{30} B / 2^{12} B = 2^{18}
- Thus, Number of bits in frame number = 18 bits
- Maximum number of bits that can be used for storing protection and other information
- = Page table entry size – Number of bits in frame number
- = 32 bits – 18 bits
- = 14 bits

Paging Hardware With TLB(translation lookaside buffer)



Effective Access Time =

Hit ratio of TLB x { Access time of TLB + Access time of main memory }

+

Miss ratio of TLB x { Access time of TLB + 2 x Access time of main memory }

Q1

A paging scheme uses a Translation Lookaside buffer (TLB). A TLB access takes 10 ns and a main memory access takes 50 ns. What is the effective access time (in ns) if the TLB hit ratio is 90% and there is no page fault?

A.54
B.60
C.65
D. 75

Option C is correct

Q2

A paging scheme uses a Translation Lookaside buffer (TLB). The effective memory access takes 160 ns and a main memory access takes 100 ns. What is the TLB access time (in ns) if the TLB hit ratio is 60% and there is no page fault?

54
60
20
75

Option C is correct

Segmentation

- Memory-management scheme that supports user view of memory
- A program is a collection of segments
- A segment is a logical unit such as:

  main program

  procedure

  function

  method

  object

  local variables, global variables

  common block

  stack

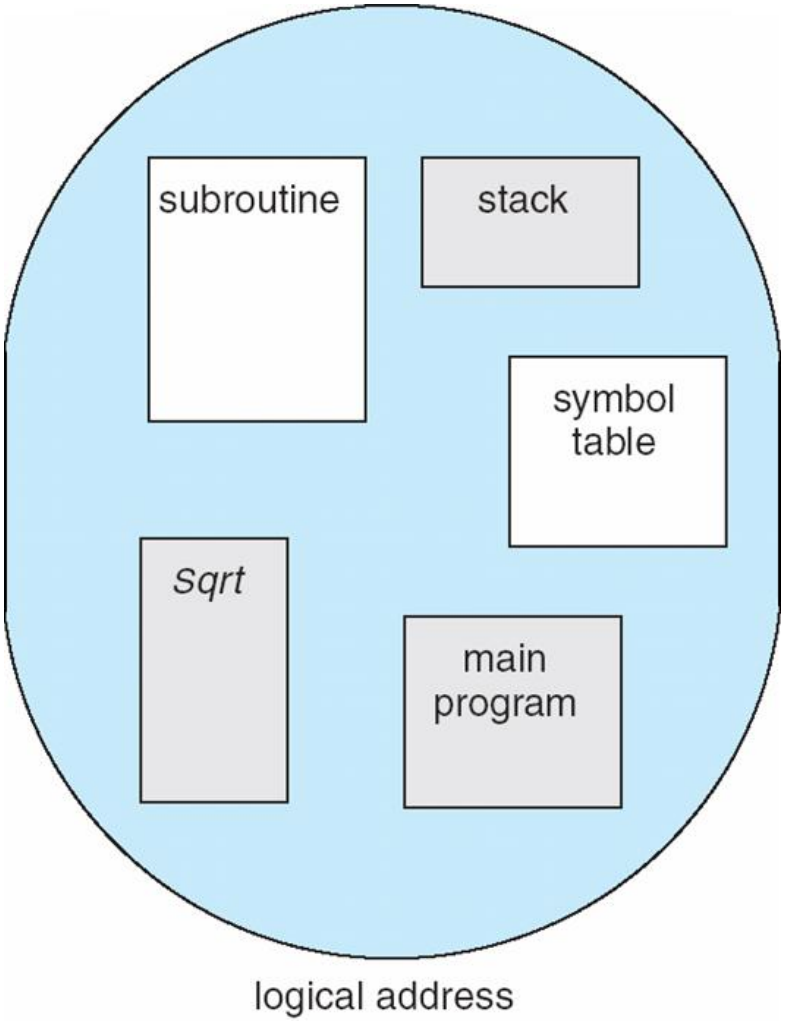
  symbol table

  arrays

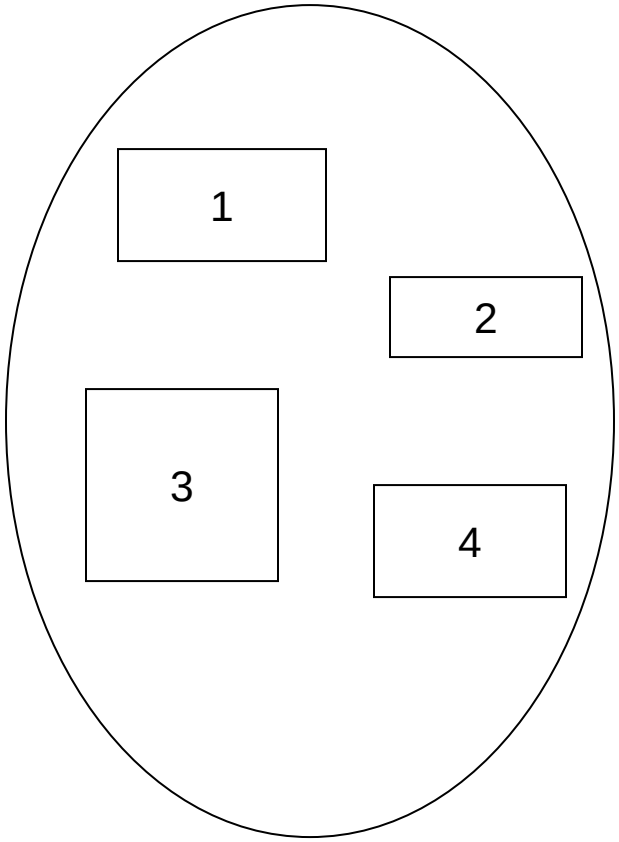
Why Segmentation is required?

- Operating system doesn't care about the User's view of the process.
- It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory.
- It decreases the efficiency of the system.

User's View of a Program



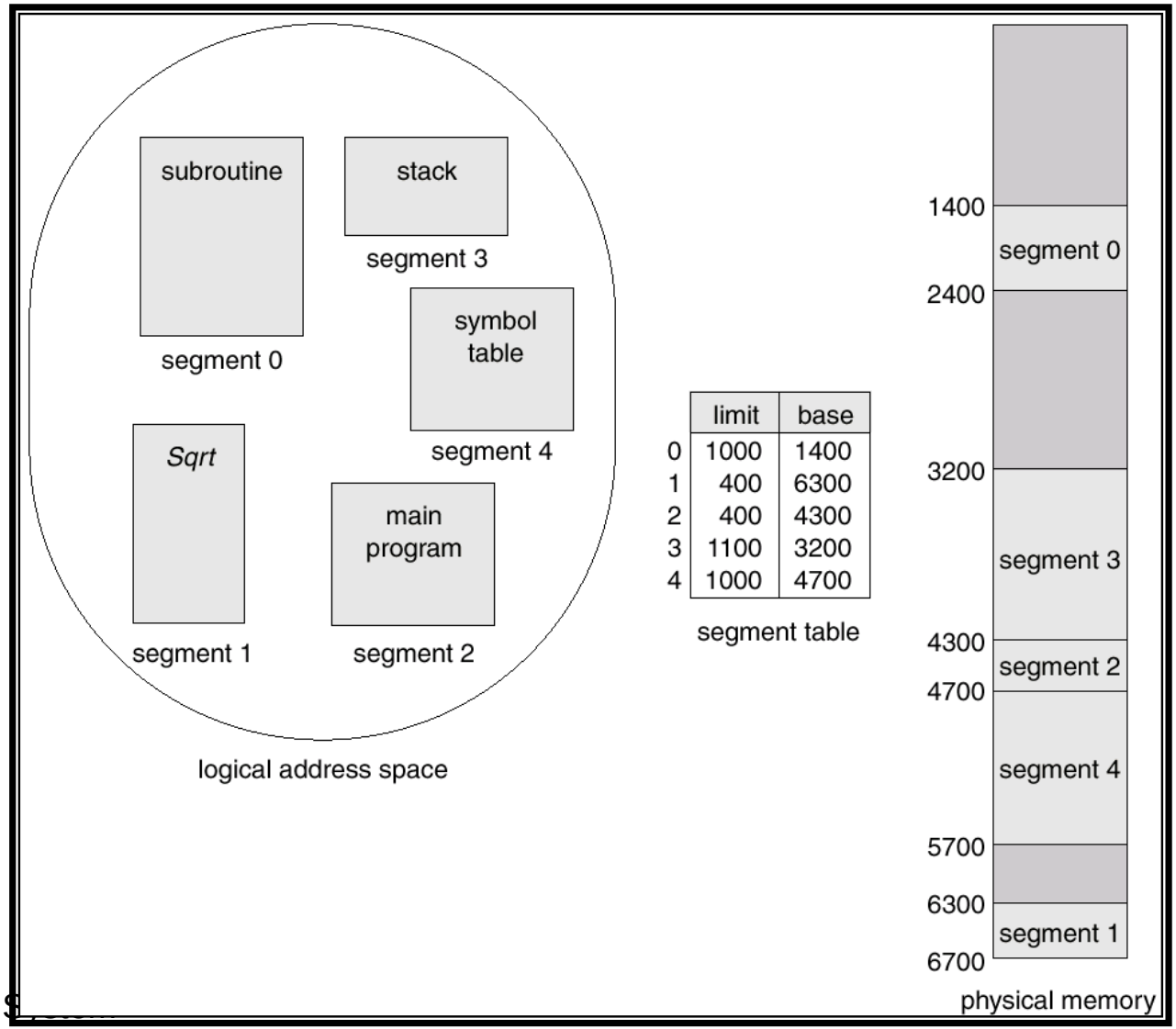
Logical View of Segmentation



user space

physical memory space

Segment Table Example



Segmentation Architecture

- Logical address consists of a two tuple:

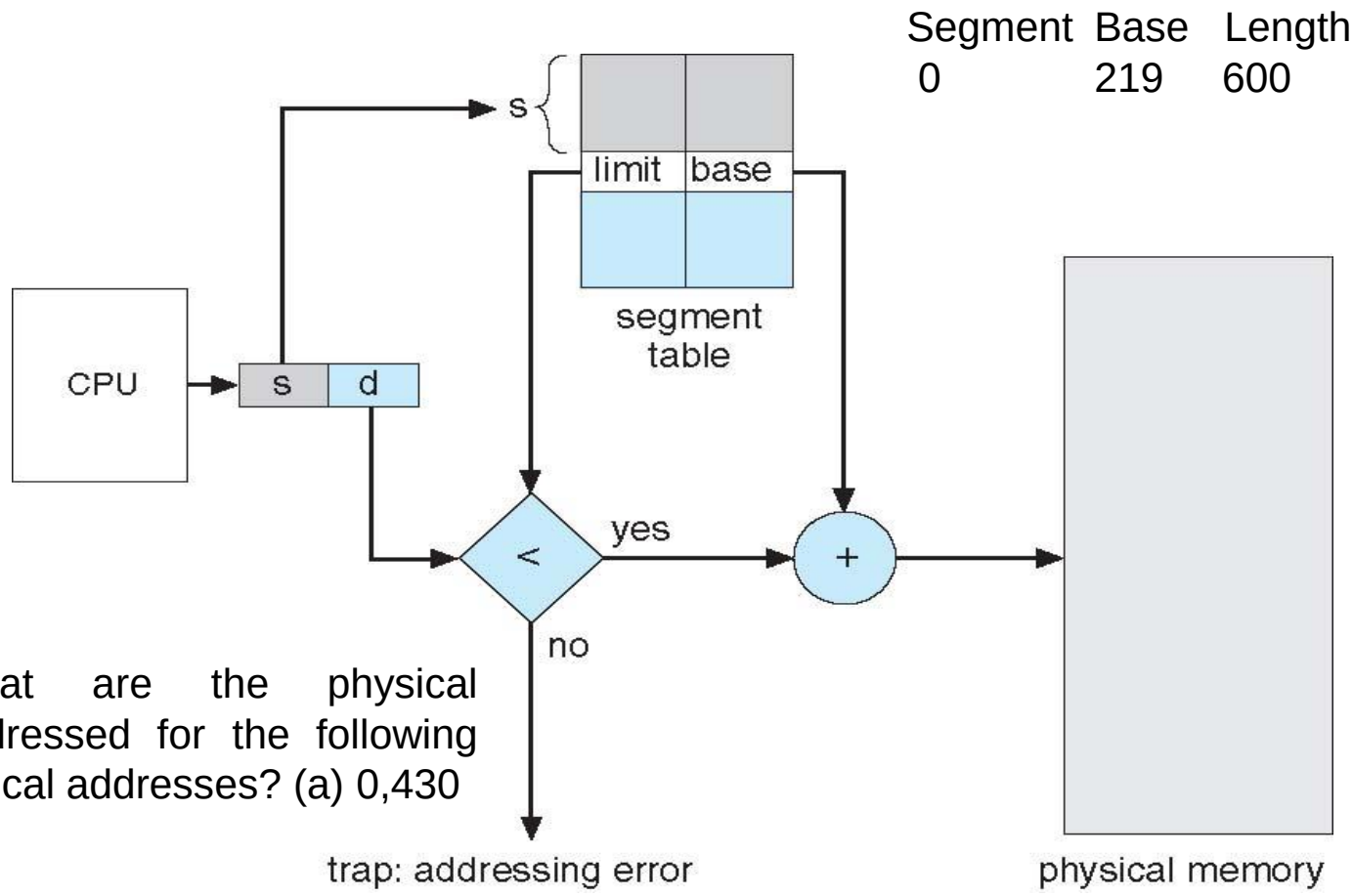
  <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:
  - **base** – contains the starting physical address where the segments reside in memory
  - **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program;

  segment number ***s*** is legal if ***s*** < **STLR**

Segmentation Architecture (Cont.)

- Since segments vary in length, memory allocation is a dynamic storage-allocation problem



What are the physical addressed for the following logical addresses? (a) 0,430

Question

- Consider the following segment table:
- Segment Base Length
- 0 219 600
- 1 2300 14
- 2 90 100
- 3 1327 580
- 4 1952 96
- What are the physical addressed for the following logical addresses? (a) 0,430 (b) 1,10 (c) 2,500 (d) 3,400 (e) 4,112

- (a) 219 + 430 = 649
- (b) 2300 + 10 = 2310
- (c) illegal reference; traps to operating system
- (d) 1327 + 400 = 1727
- (e) illegal reference; traps to operating system

Advantages of Segmentation

- No internal fragmentation
- Average Segment Size is larger than the actual page size.
- Less overhead
- It is easier to relocate segments than entire address space.
- The segment table is of lesser size as compare to the page table in paging.

Disadvantages

- It can have external fragmentation.
- it is difficult to allocate contiguous memory to variable sized partition.
- Costly memory management algorithms.

Refrences

- https://www.gatevidyalay.com/paging-formulas-practice-problems/

OPERATING SYSTEMS

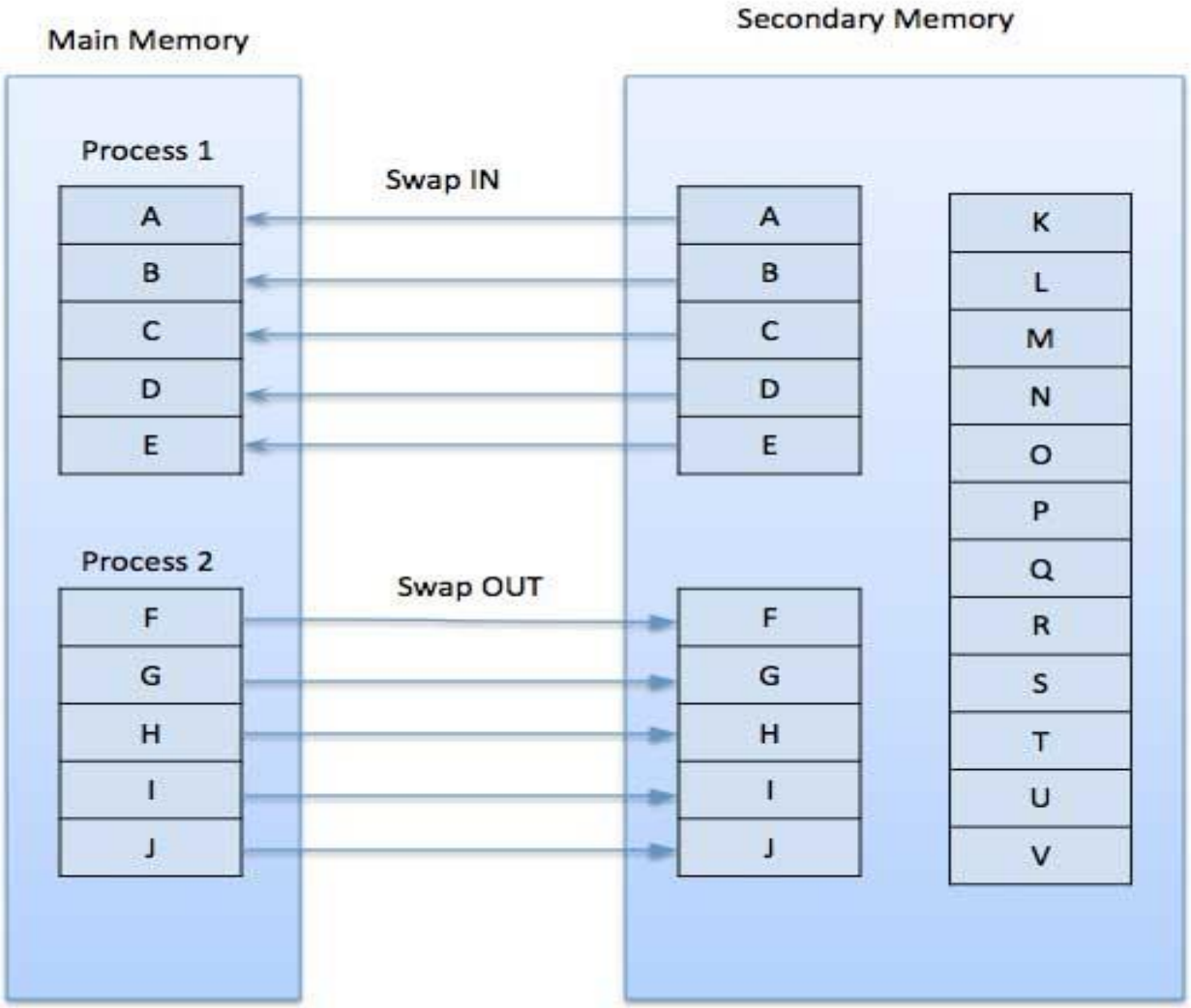
Memory Management

Class Presentations on Operating System

by Subhash Chand Agrawal

Demand Paging

- A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance.
- When a context switch occurs, the operating system does not copy any of the old program’s pages out to the disk or any of the new program’s pages into the main memory.
- Instead, it just begins executing the new program after loading the first page and fetches that program’s pages as they are referenced.



https://www.tutorialspoint.com/operating_system/os_virtual_memory.htm

Page Fault

- While executing a program, if the program references a page which is not available in the main memory.
- Processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Paging Revisitted

- If a page is not in physical memory
  - find the page on disk
  - find a free frame
  - bring the page into memory
- What if there is no free frame in memory?

Page Replacement

- Basic idea
  - if there is a free page in memory, use it
  - if not, select a *victim* frame
  - write the victim out to disk
  - read the desired page into the now free frame
  - update page tables
  - restart the process

Page Replacement

- Main objective of a good replacement algorithm is to achieve a low *page fault rate*
  - insure that heavily used pages stay in memory
  - the replaced page should not be needed for some time
- Secondary objective is to reduce latency of a page fault
  - efficient code
  - replace pages that do not need to be written out

Reference String

- Reference string is the sequence of pages being referenced
- If user has the following sequence of addresses
  - 123, 215, 600, 1234, 76, 96
- If the page size is 100, then the reference string is
  - 1, 2, 6, 12, 0, 0

First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Very simple to implement
  - keep a list
    - victims are chosen from the tail
    - new pages in are placed at the head

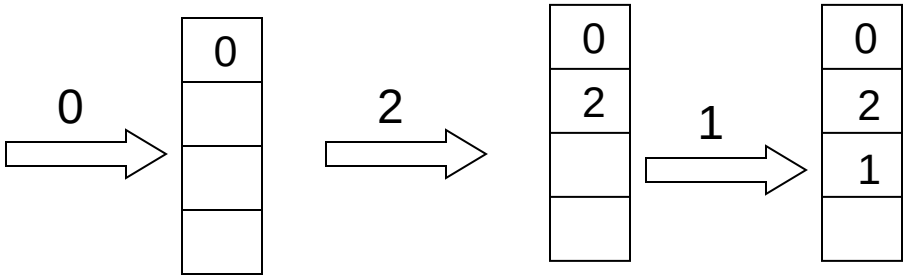
FIFO

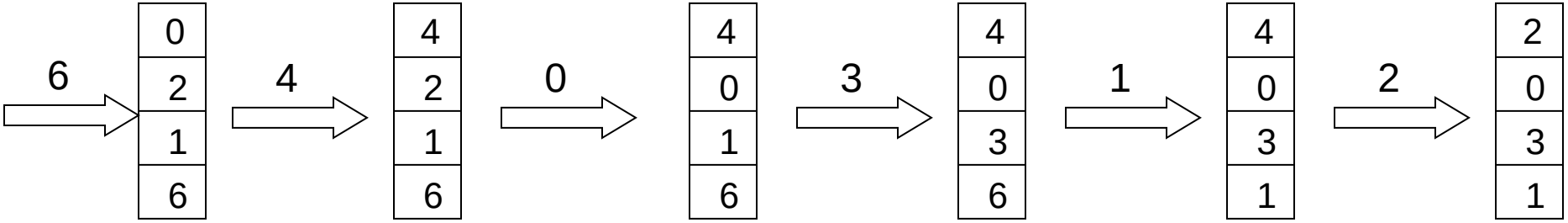
- Page Fault Rate = 9 / 12 = 0.75

- Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

x x x x x x x x x

Compulsory Misses





FIFO Issues

- Poor replacement policy
- Evicts the oldest page in the system
  - usually a heavily used variable should be around for a long time
  - FIFO replaces the oldest page - perhaps the one with the heavily used variable
- FIFO does not consider page usage

- Giving more memory to process would improve its performance?

- Consider the following reference string:

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- Calculate the page fault rate for 3 frames and 4 frames with FIFO page replacement algorithm.

Belady's Anomaly:

- For some page replacement algorithms, the page fault rate may increase as the number of allocated frames increases.

Optimal Page Replacement

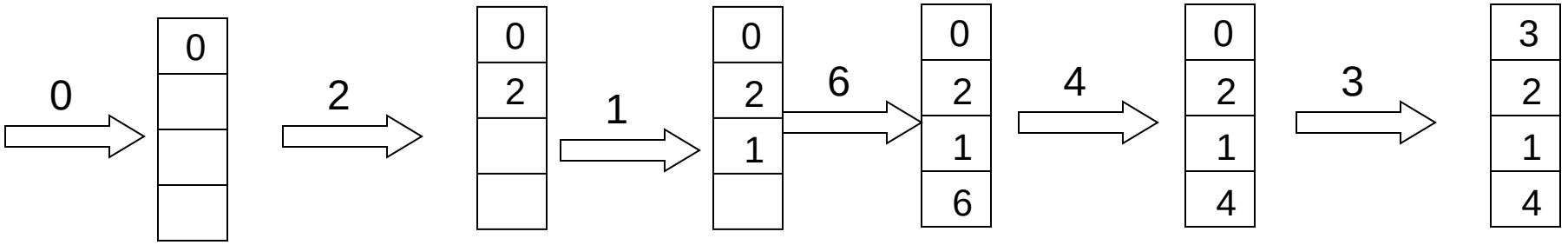
- Basic idea
  - replace the page that will not be referenced for the longest time
- This gives the lowest possible fault rate

Optimal Page Replacement(Future)

•Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

x x x x x x

Compulsory Misses



•Fault Rate = 6 / 12 = 0.50

•With the above reference string, this is the best we can hope to do

Least Recently Used (LRU)(Past)

- Basic idea
  - replace the page in memory that has not been accessed for the longest time
- Optimal policy looking back in time
  - as opposed to forward in time
  - fortunately, programs tend to follow similar behavior

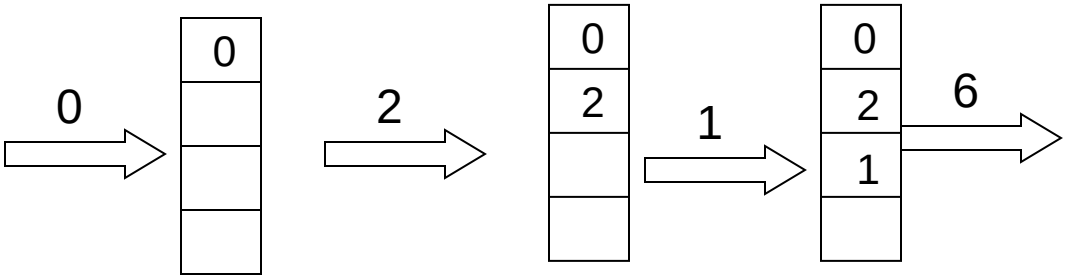
LRU

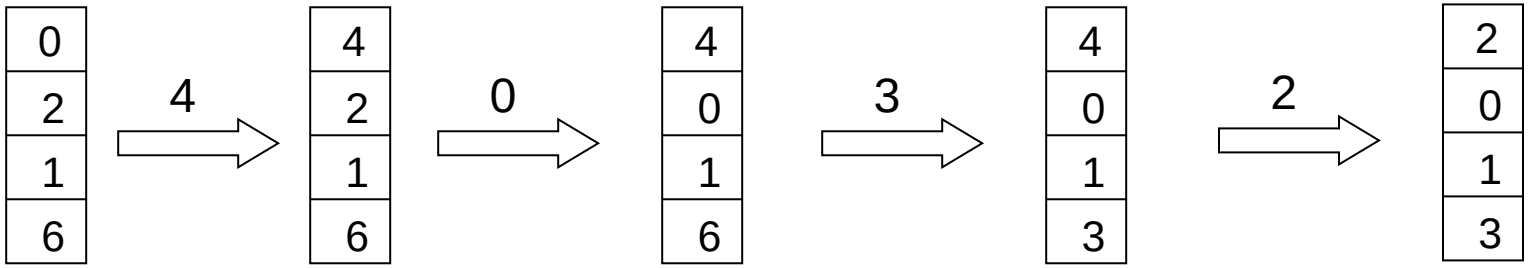
•Consider the following reference string: 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

x x x x x x x x

Compulsory Misses

•Fault Rate = 8 / 12 = 0.67





LRU Issues

- How to keep track of last page access?
  - requires special hardware support
- 2 major solutions
  - Counters
    - the page with the smallest “time” value is replaced
  - stack
    - keep a stack of references
    - on every reference to a page, move it to top of stack
    - page at bottom of stack is next one to be replaced

Q1

- Find the page fault rate by considering the following reference string:
- 7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 using three frames with
- (i) FIFO
- (ii) Optimal
- (iii) LRU

(i)15
(ii) 9
(iii) 12

Q2

- Consider the following page reference string: 1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6

  How many page faults would occur for replacement by LRU, FIFO, optimal, for one, two, three, four, five, six and seven frames? All frames are initially empty and first unique page reference causes a page fault.

# frames	1	2	3	4	5	6	7
LRU	20	18	15	10	8	7	7
FIFO	20	18	16	14	10	10	7
Optimal	20	15	11	8	7	7	7

- In a demand-paged system, the degree of multiprogramming is fixed at 4. Measurements were made to determine the utilization of cpu and the paging disk. The result is one of the following. For each of case a, b, c, state in one line what is happening?
- a. cpu utilization 13%, disk 97%
- b. cpu utilization 92%, disk 11%
- c. cpu utilization 21%, disk 3%

- Answer:
- a. Thrashing
- b. okay
- c. can increase multiprogramming

- In which one of the following page replacement algorithms it is possible for the page fault rate to increase even when the number of allocated frames increases?

- (A) LRU (Least Recently Used)
- (B) OPT (Optimal Page Replacement)
- (C) MRU (Most Recently Used)
- (D) FIFO (First In First Out)
- Ans: D

OPERATING SYSTEMS

Disk Scheduling

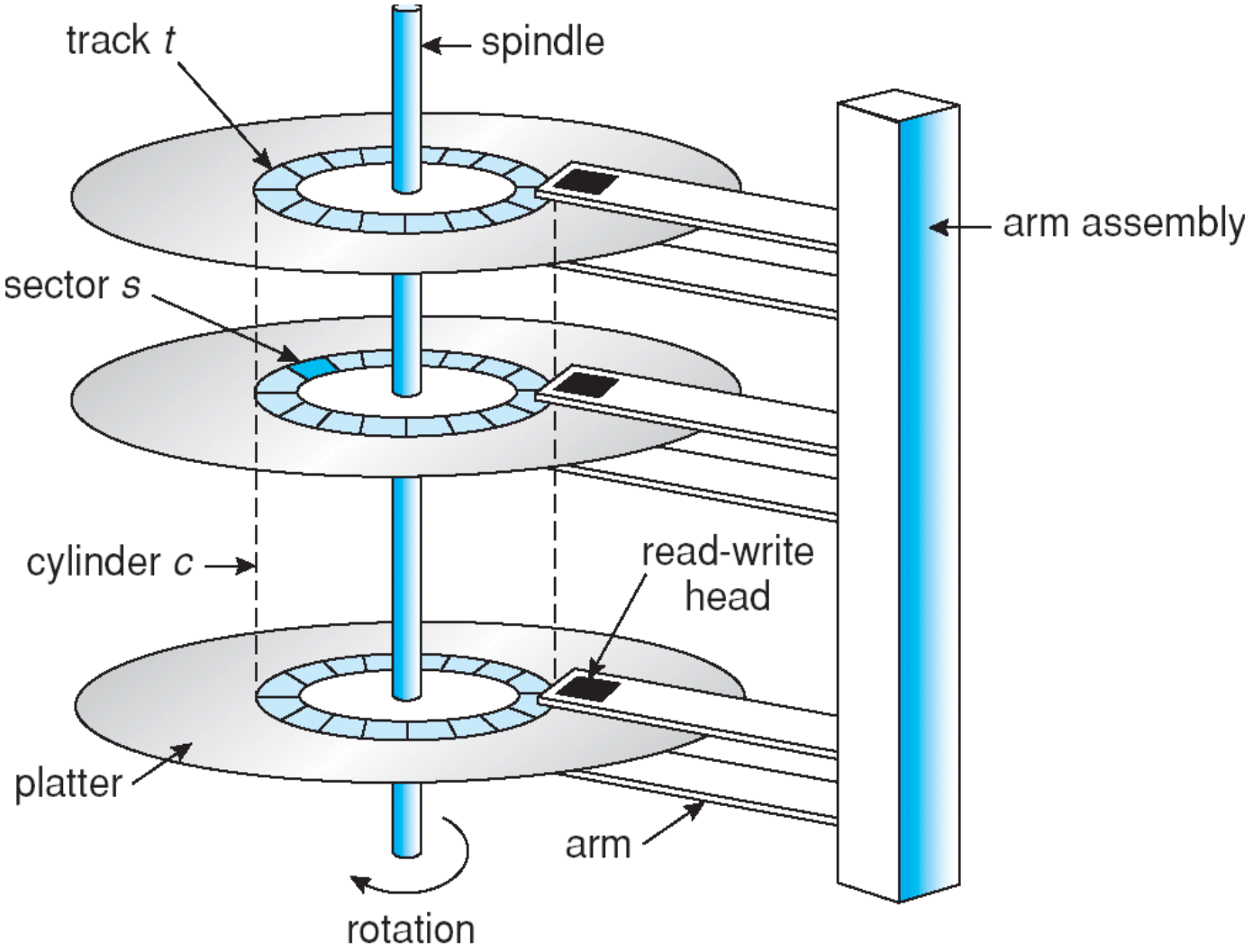
Class Presentations on Operating System

by Subhash Chand Agrawal

Disk Scheduling

- As we know, a process needs two type of time, CPU time and IO time. For I/O, it requests the Operating system to access the disk.
- However, the operating system must be fare enough to satisfy each request and at the same time, operating system must maintain the efficiency and speed of process execution.
- The technique that operating system uses to determine the request which is to be satisfied next is called disk scheduling.

Moving-head Disk Mechanism



- **Seek Time**
- Seek time is the time taken in locating the disk arm to a specified track where the read/write request will be satisfied.
- **Rotational Latency**
- It is the time taken by the desired sector to rotate itself to the position from where it can access the R/W heads.
- **Transfer Time**
- It is the time taken to transfer the data.

- **Disk Access Time**
- Disk access time is given as,
- Disk Access Time = Rotational Latency + Seek Time + Transfer Time
- **Disk Response Time**
- It is the average of time spent by each request waiting for the IO operation.

Purpose of Disk Scheduling

- The main purpose of disk scheduling algorithm is to select a disk request from the queue of IO requests and decide the schedule when this request will be processed.

Goal of Disk Scheduling Algorithm

- Fairness
- High throughout
- Minimal traveling head time

Disk Scheduling Algorithms

- FCFS scheduling algorithm
- SSTF (shortest seek time first) algorithm
- SCAN scheduling
- C-SCAN scheduling
- LOOK Scheduling
- C-LOOK scheduling

Disk Scheduling Algorithms

- We illustrate them with a I/O request queue (cylinders are between 0-199):

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

  head starts at 53

First Come First Serve (FCFS) Example

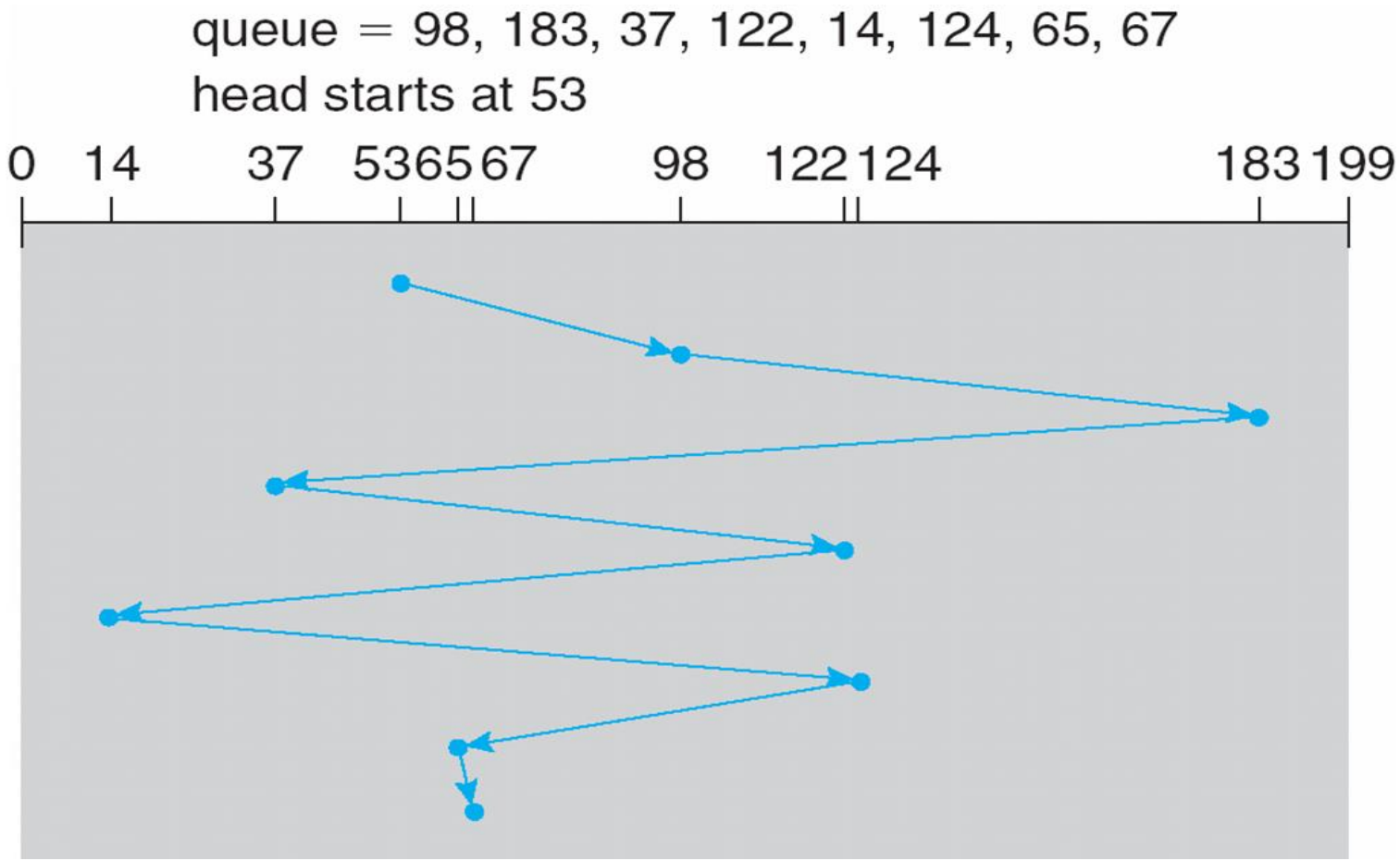


Illustration shows total head movement of 640 cylinders.

First Come First Serve (FCFS)

- Handle I/O requests sequentially.
- Fair to all processes.
- Suffers from global zigzag effect.

Shortest Seek Time First (SSTF) Example

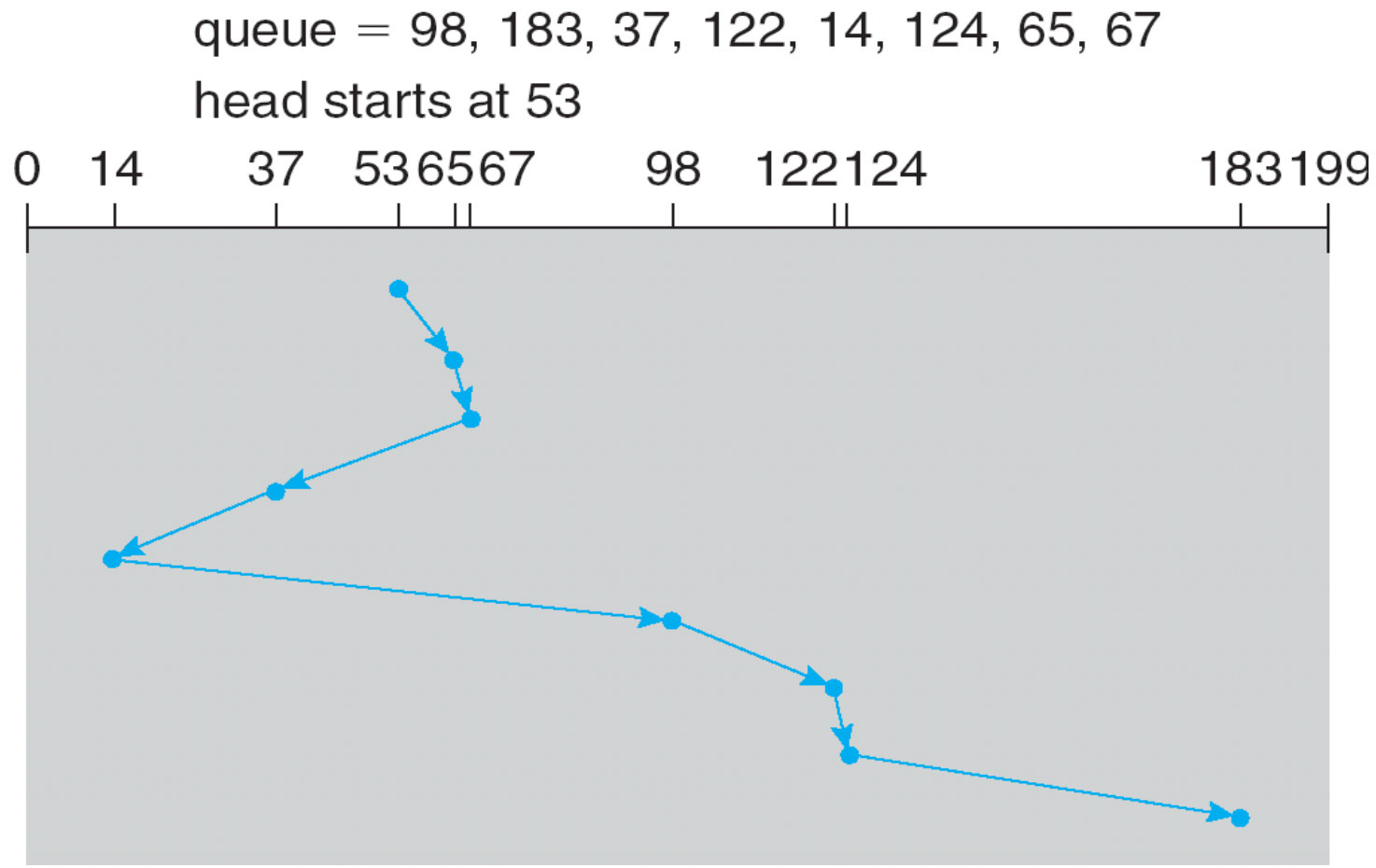


Illustration shows total head movement of 236 cylinders.

Shortest Seek Time First (SSTF)

- Selects the request with the minimum seek time from the current head position.
- Also called Shortest Seek Distance First (SSDF) – It's easier to compute distances.
- It's biased in favor of the middle cylinders requests.
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests.

FCFS

- Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
- Initial head position = 50
- Total number of seek operations or head movement?

Therefore, the total seek count is calculated as:
=(176-50)+(176-79)+(79-34)+(60-34)+(92-60)+(92-11)+(41-11)+(114-41) = 510

SSTF

- Consider the following disk request sequence for a disk with 100 tracks
- 45, 21, 67, 90, 4, 89, 52, 61, 87, 25
- Head pointer starting at 50. Find the number of head movements in cylinders using SSTF scheduling.

4 21 25 45 50 52 61 67 87 89 90

Number of cylinders = 5 + 7 + 9 + 6 + 20 + 2 + 1 + 65 + 4 + 17 = 136

Elevator Algorithms

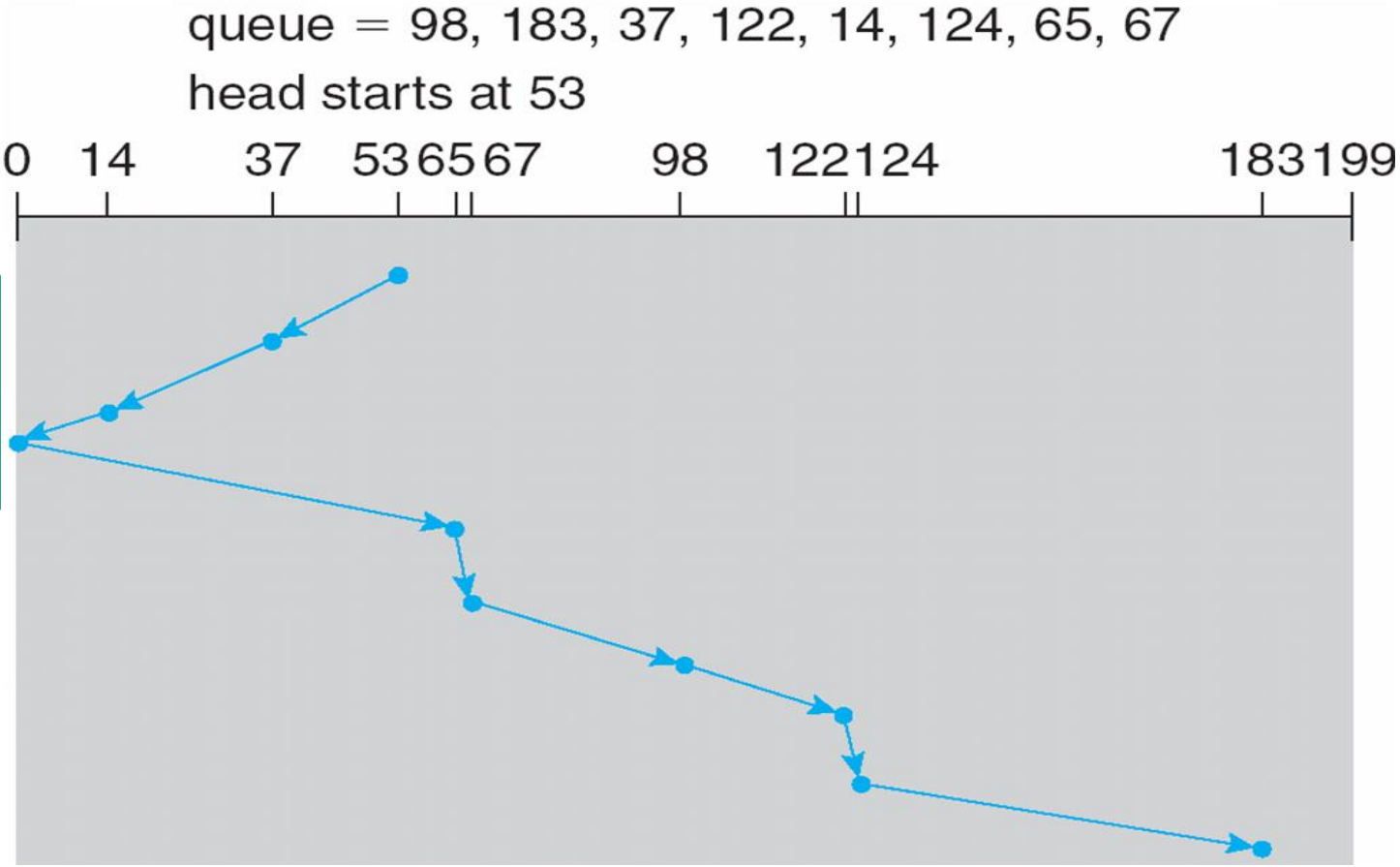
- Algorithms based on the common elevator principle.
- Four combinations of Elevator algorithms:
  - Service in both directions or in only one direction.
  - Go until last cylinder or until last I/O request.

Direction \ Go until	Go until the last cylinder	Go until the last request
Service both directions	**Scan**	**Look**
Service in only one direction	**C-Scan**	**C-Look**

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.
- It moves in both directions until both ends.
- Tends to stay more at the ends so more fair to the extreme cylinder requests.

Scan Example



Assuming disk arm is moving toward 0

Illustration shows total head movement of 236 cylinders.

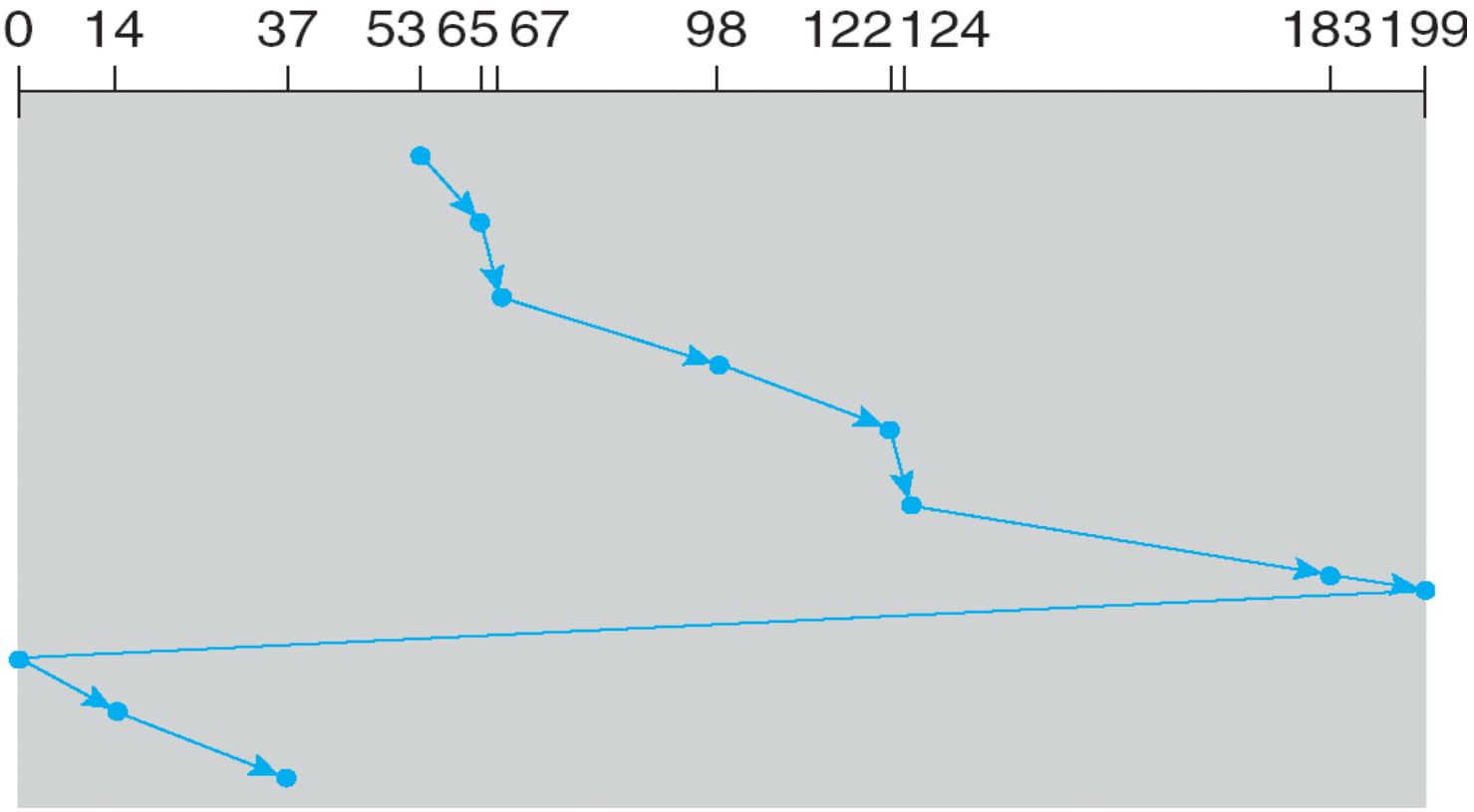
C-Scan

- The head moves from one end of the disk to the other, servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first one.
- Provides a more uniform wait time than SCAN; it treats all cylinders in the same manner.

C-Scan Example

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

head starts at 53



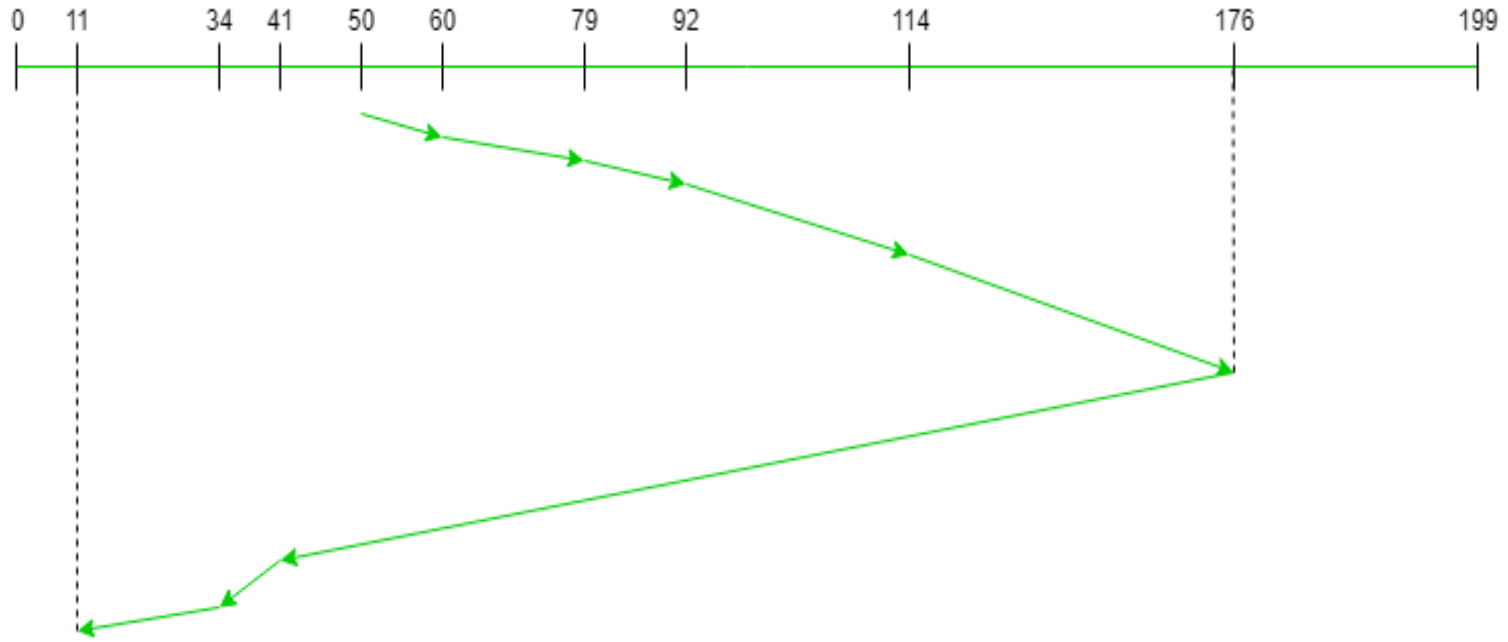
Look

- The disk arm starts at the first I/O request on the disk, and moves toward the last I/O request on the other end, servicing requests until it gets to the other extreme I/O request on the disk, where the head movement is reversed and servicing continues.
- It moves in both directions until both last I/O requests;

291

Example

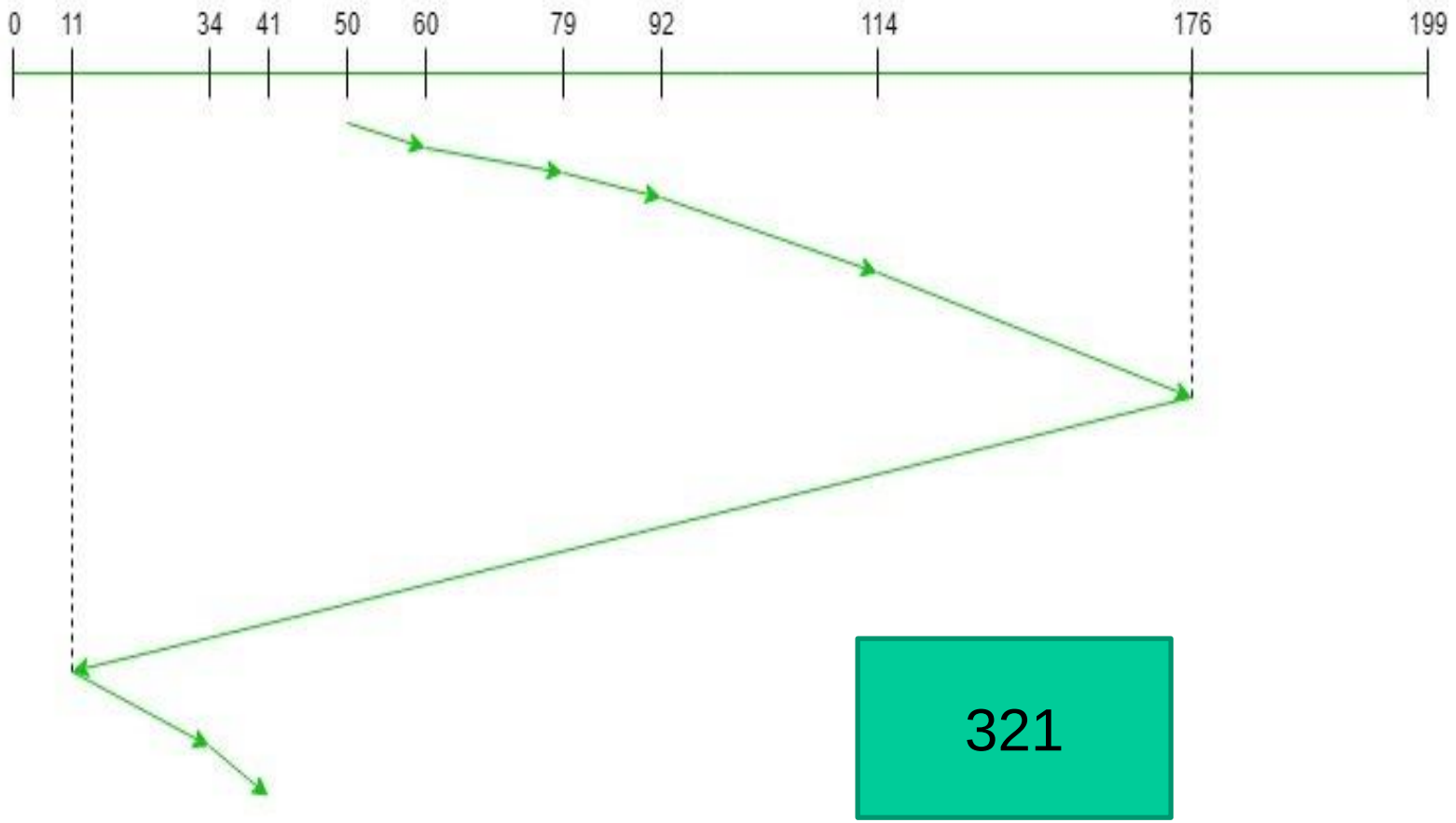
- Request sequence = {176, 79, 34, 60, 92, 11, 41, 114} Initial head position = 50 Direction = right (We are moving from left to right)



C-Look

- Look version of C-Scan.
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.
- In general, Circular versions are more fair but pay with a larger total seek time.
- Scan versions have a larger total seek time than the corresponding Look versions.

Request sequence = {176, 79, 34, 60, 92, 11, 41, 114}
Initial head position = 50
Direction = right (Moving from left to right)

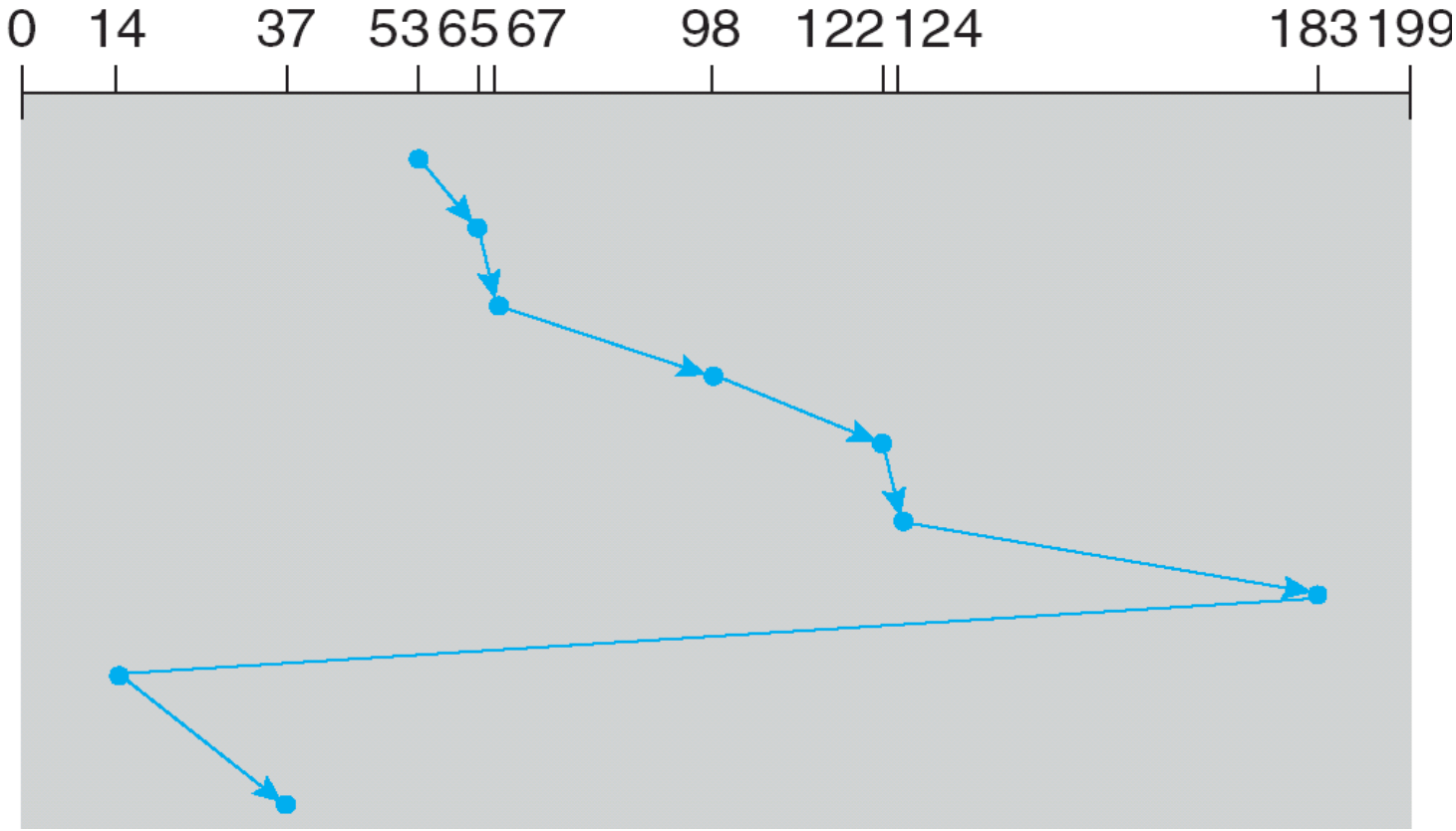


321

C-Look Example

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

head starts at 53

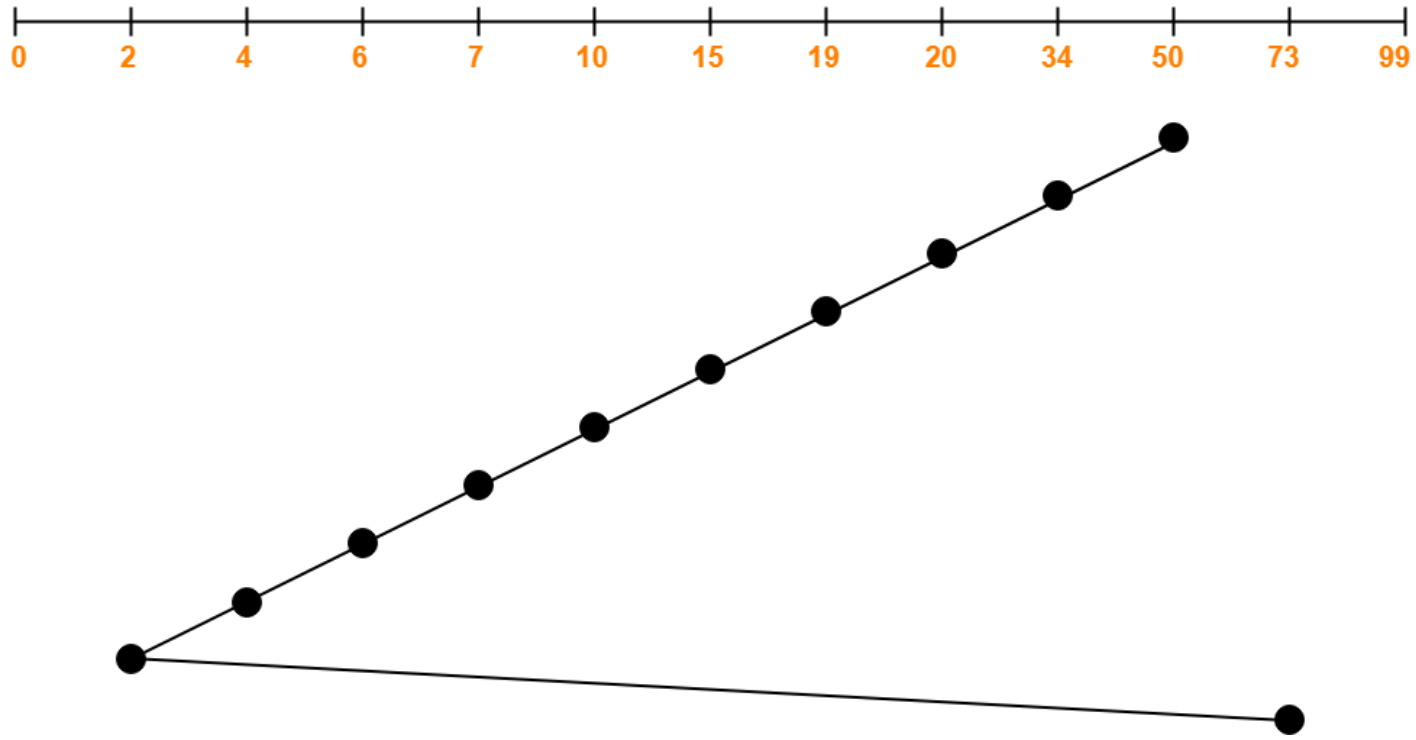


Q1

- Consider a disk system with 100 cylinders. The requests to access the cylinders occur in following sequence-
- 4, 34, 10, 7, 19, 73, 2, 15, 6, 20
- Assuming that the head is currently at cylinder 50, what is the time taken to satisfy all requests if it takes 1 ms to move from one cylinder to adjacent one and shortest seek time first policy is used?
- 95 ms
- 119 ms
- 233 ms
- 276 ms

Solution

- 119

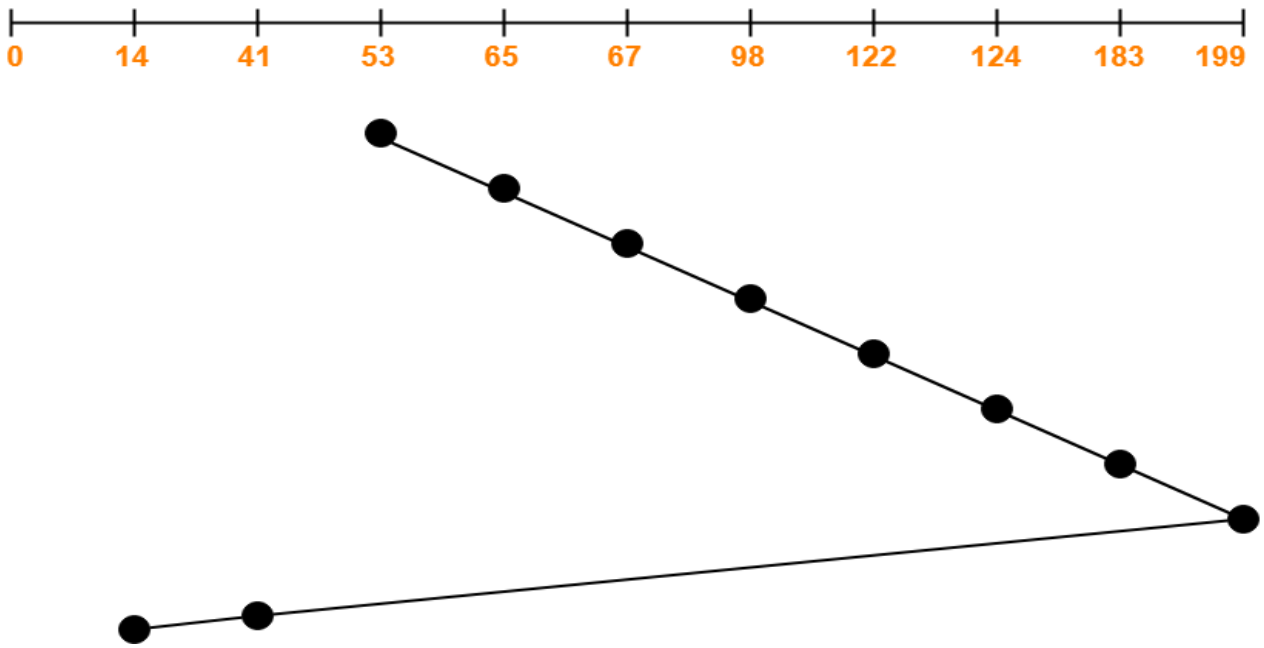


Q2

- Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution

- 331

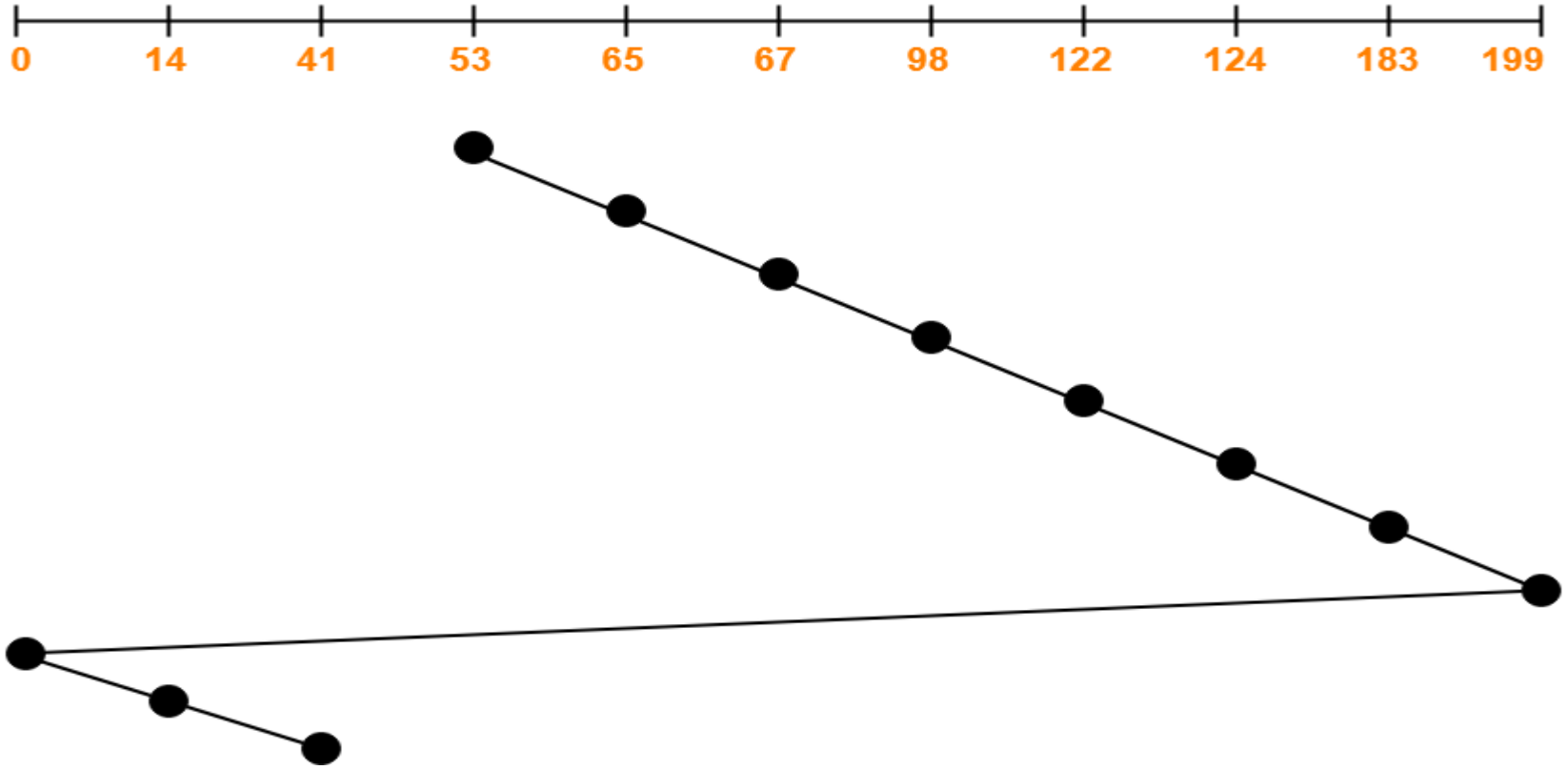


Q3

- Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-SCAN scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution

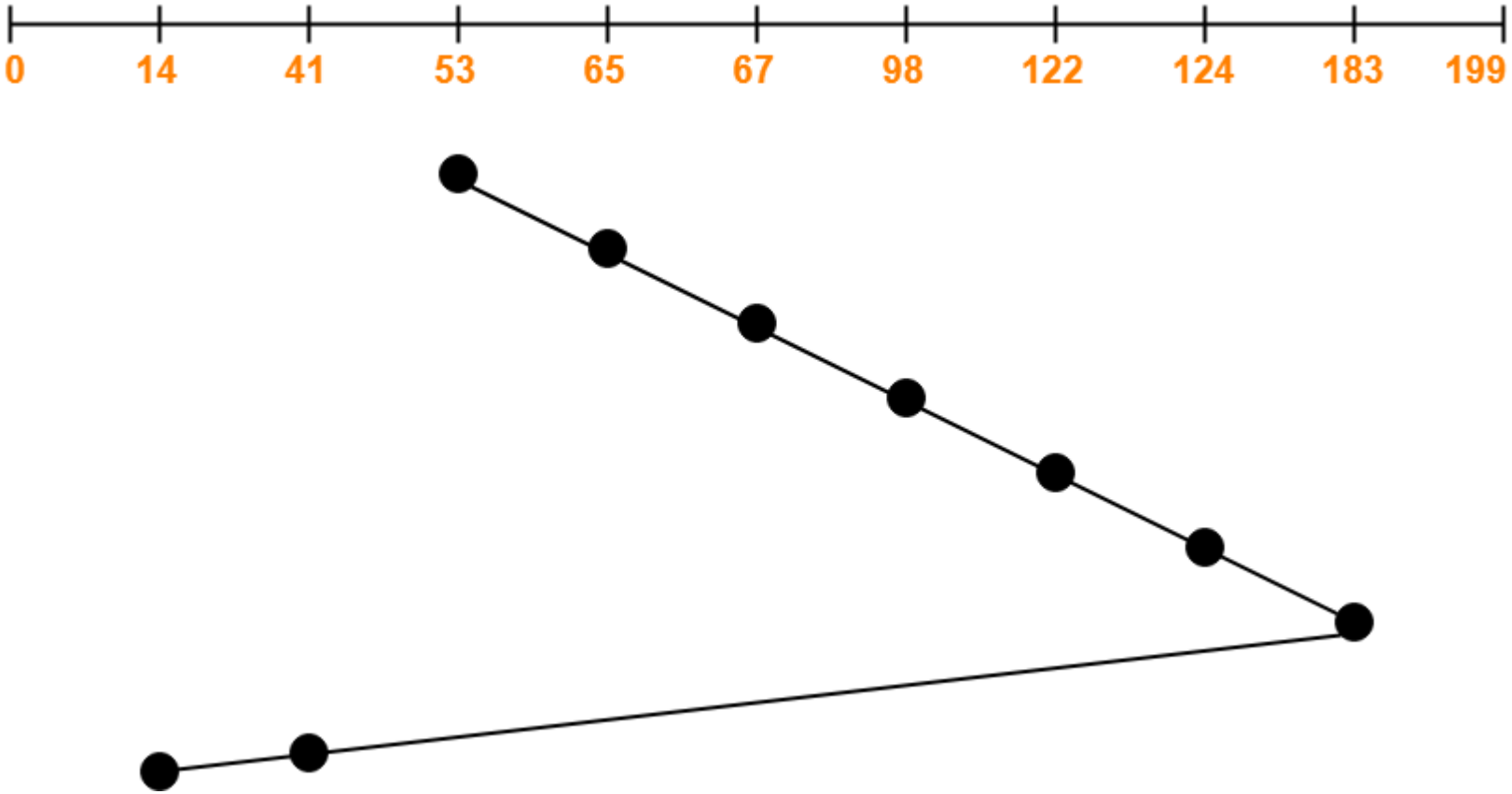
- 386



Q4

- Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

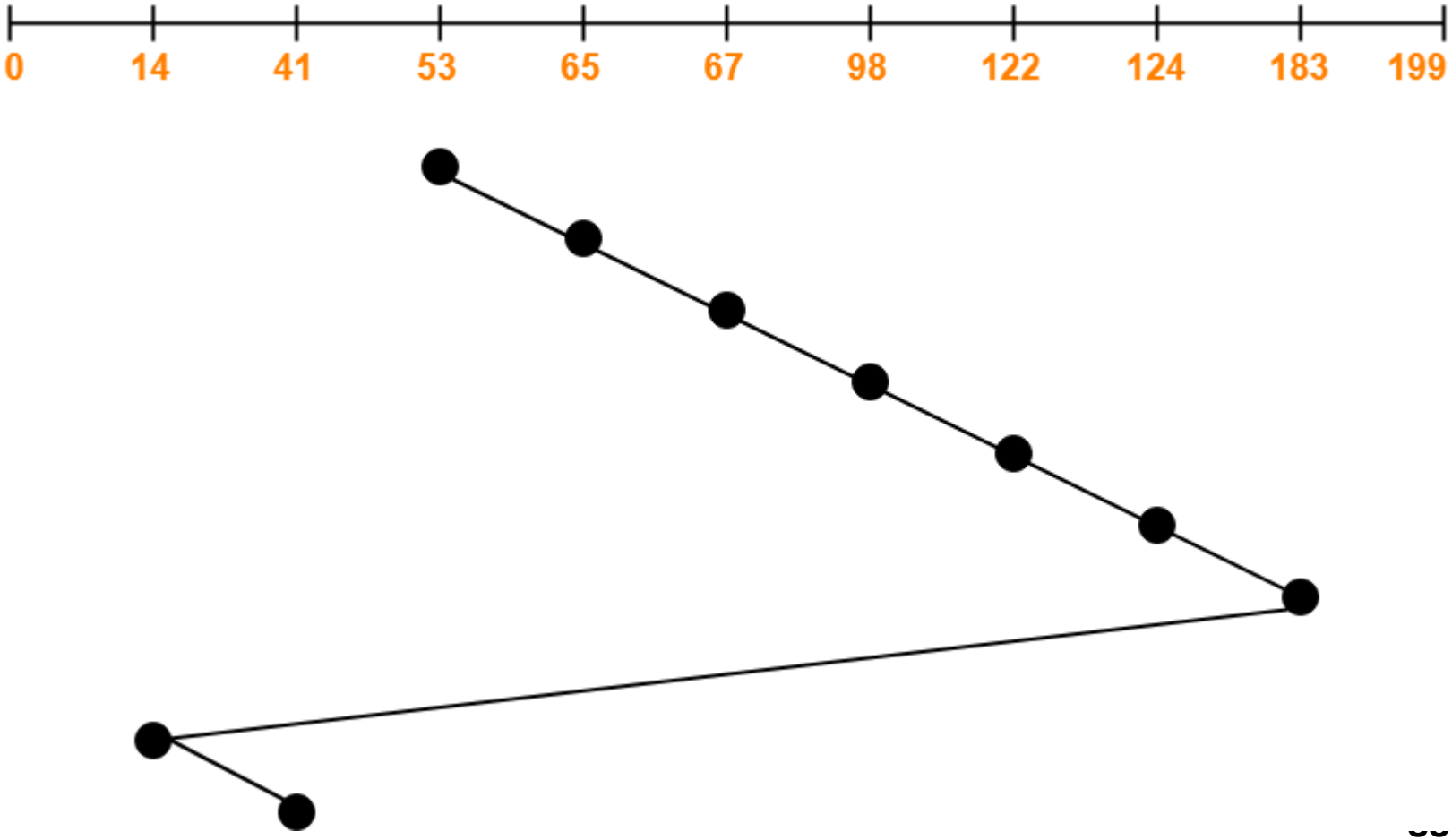
Solution: 299



Q5

- Consider a disk queue with requests for I/O to blocks on cylinders 98, 183, 41, 122, 14, 124, 65, 67. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 53 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

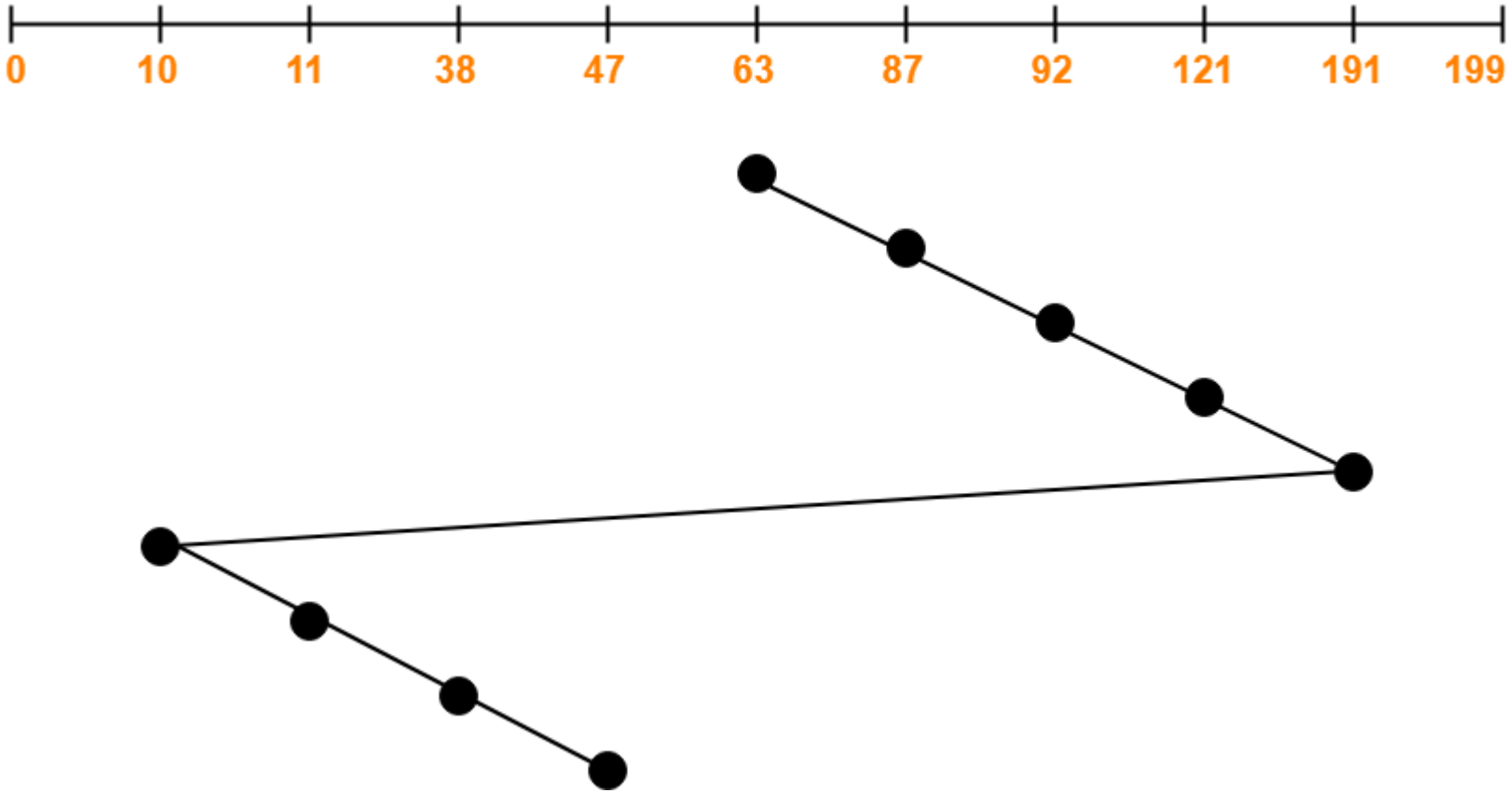
Solution:326



Q6:

- Consider a disk queue with requests for I/O to blocks on cylinders 47, 38, 121, 191, 87, 11, 92, 10. The C-LOOK scheduling algorithm is used. The head is initially at cylinder number 63 moving towards larger cylinder numbers on its servicing pass. The cylinders are numbered from 0 to 199. The total head movement (in number of cylinders) incurred while servicing these requests is _____.

Solution:346



- https://www.geeksforgeeks.org/disk-scheduling-algorithms/

GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

OPERATING SYSTEMS

File System

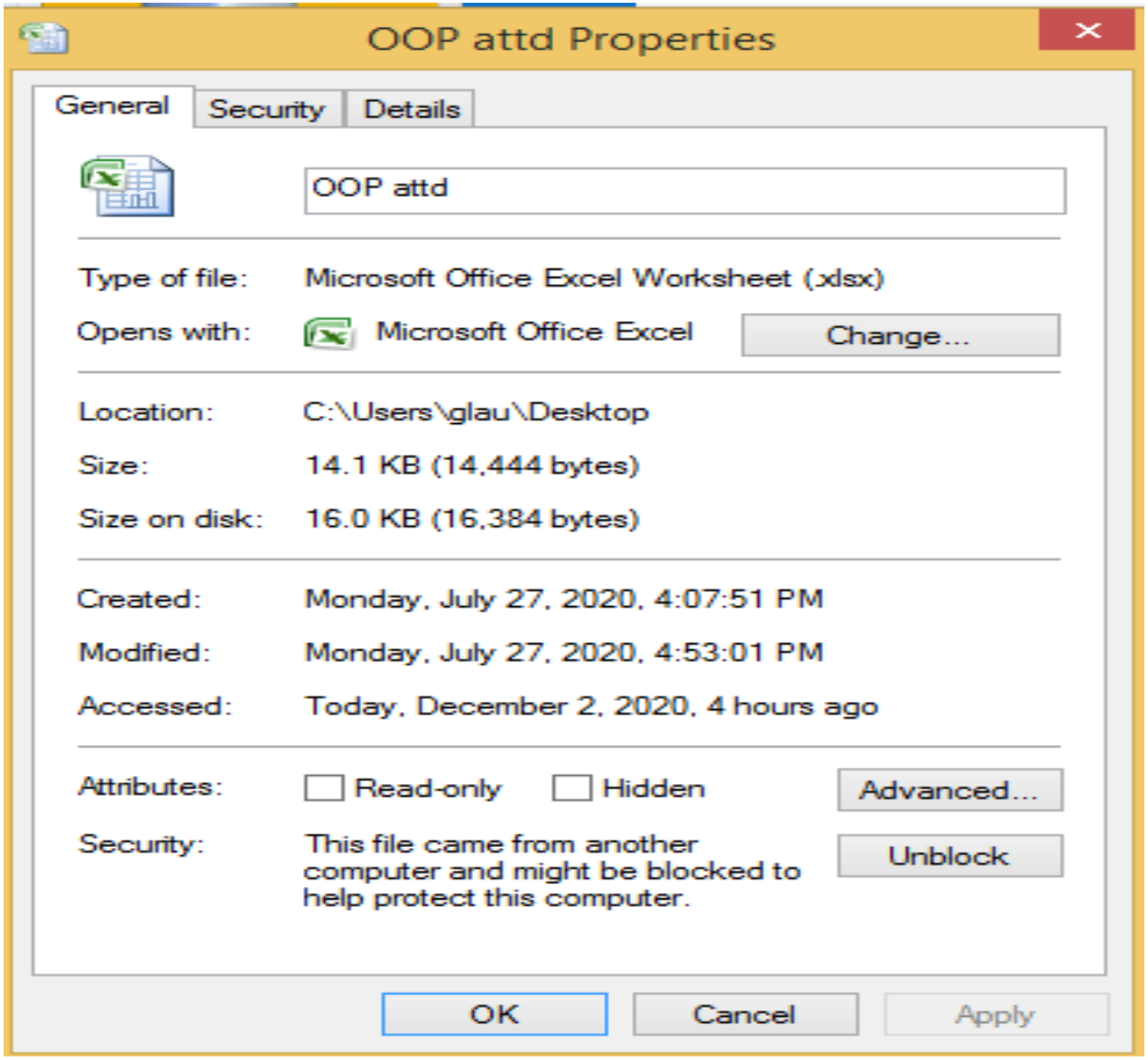
File Concept

- A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.
- In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.
- 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** – Along with the name, Each File has its own extension which identifies the type of the file. For example, a text file has the extension **.txt,** A video file can have the extension **.mp4.**
- **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

e info Window on Windows



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

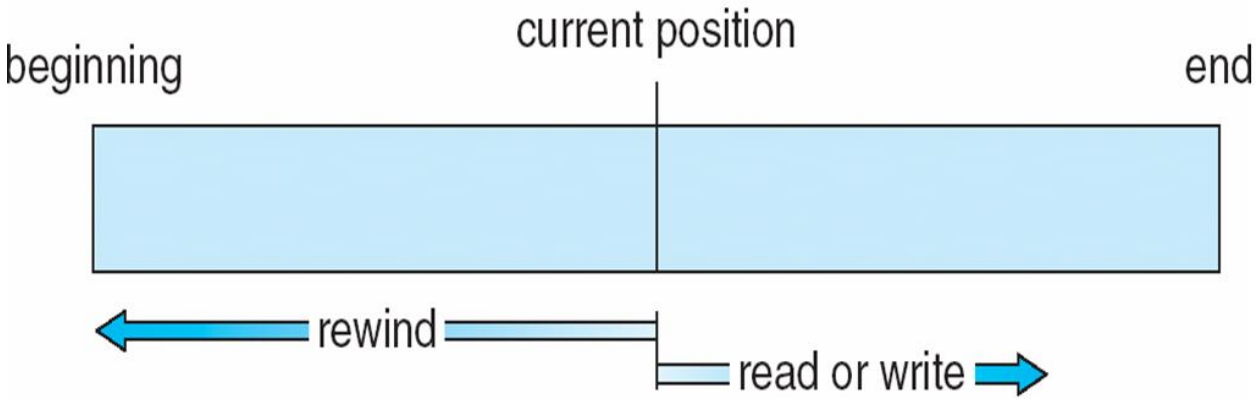
File Types – Name, Extension

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 compressed, for archiving or storage
multimedia	mpeg, mov, rm, mp3, avi	binary file containing audio or A/V information

Access Methods

- Sequential access
- Direct/Random access
- Indexed sequential access

Sequential-access File



A sequential access is that in which the records are accessed in some sequence, i.e., the information in the file is processed in order, one record after the other.

This access method is the most primitive one.

Example: Compilers usually access files in this fashion.

Direct/Random access

Random access file organization provides, accessing the records directly.

Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.

The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.

The Direct Access is mostly required in the case of database systems

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;

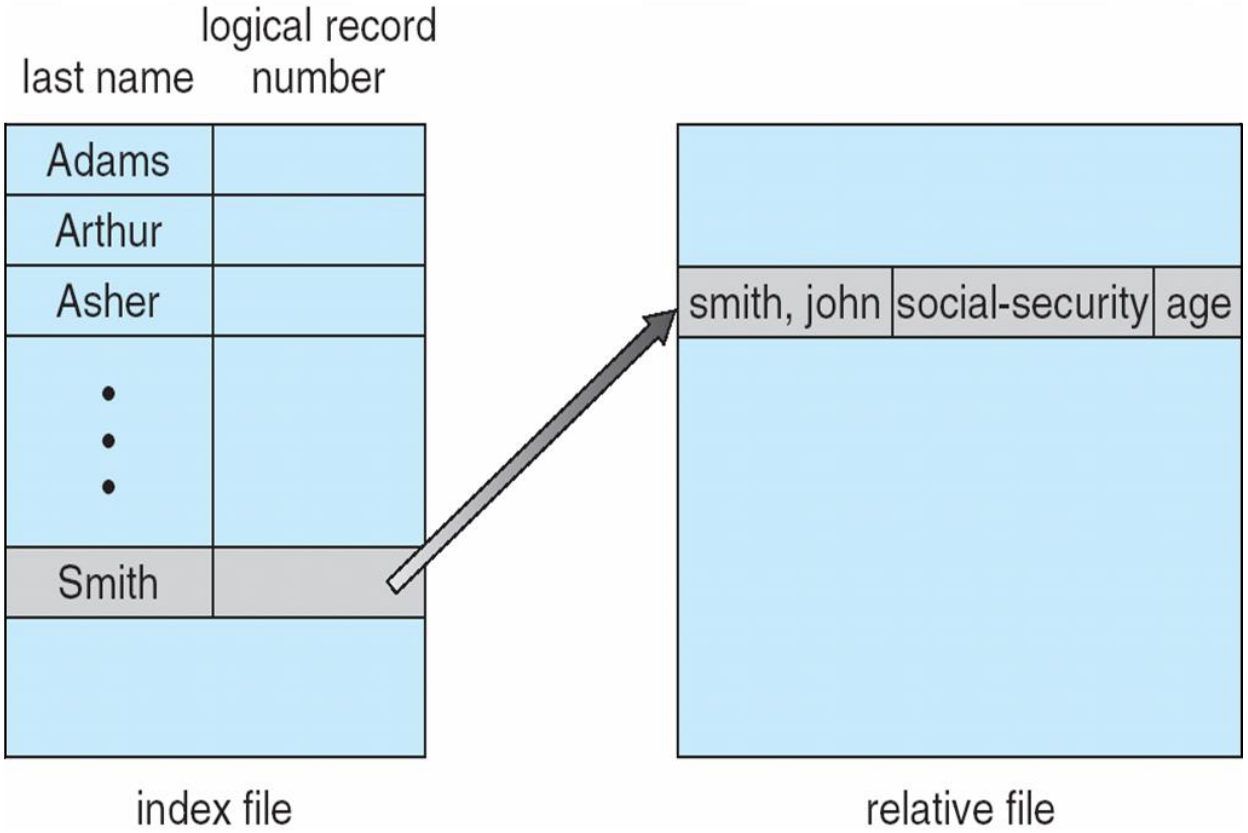
Other Access Methods

- General involve creation of an index for the file
- Keep index in memory for fast determination of location of data to be operated on
- A particular record can be accessed by its index.
- The index is nothing but the address of a record in the file.

Indexed sequential access

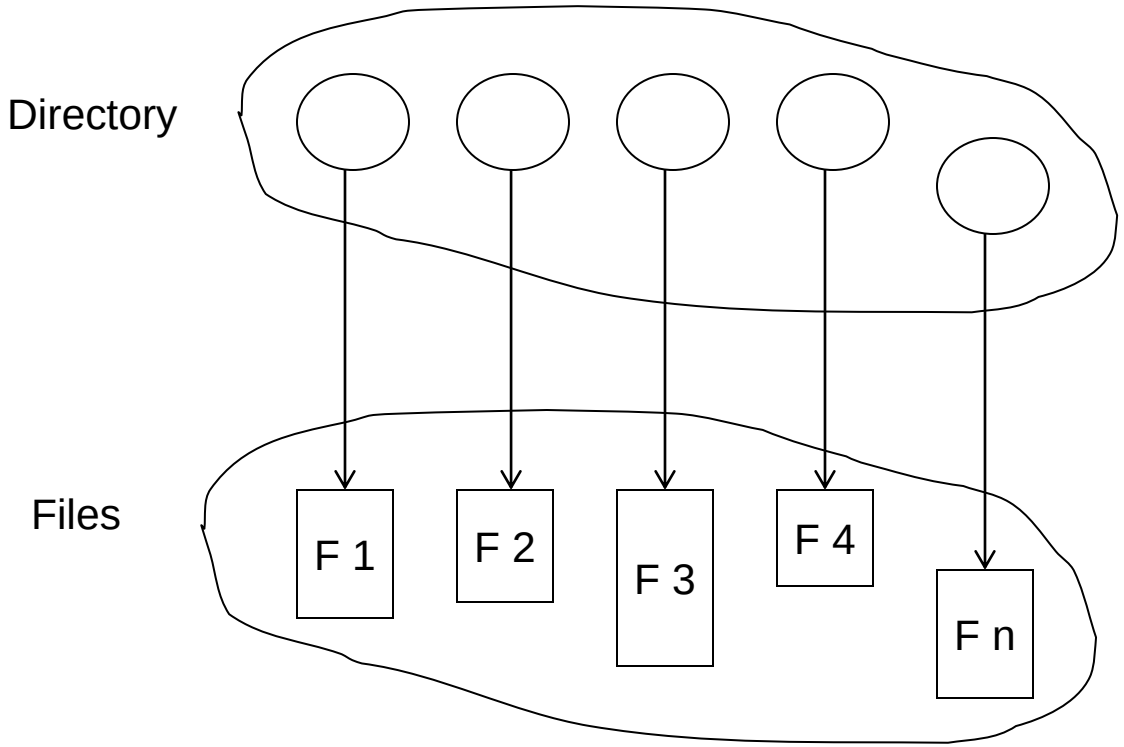
This mechanism is built up on base of sequential access.
An index is created for each file which contains pointers to various blocks.
Index is searched sequentially and its pointer is used to access the file directly.

Example of Index and Relative Files

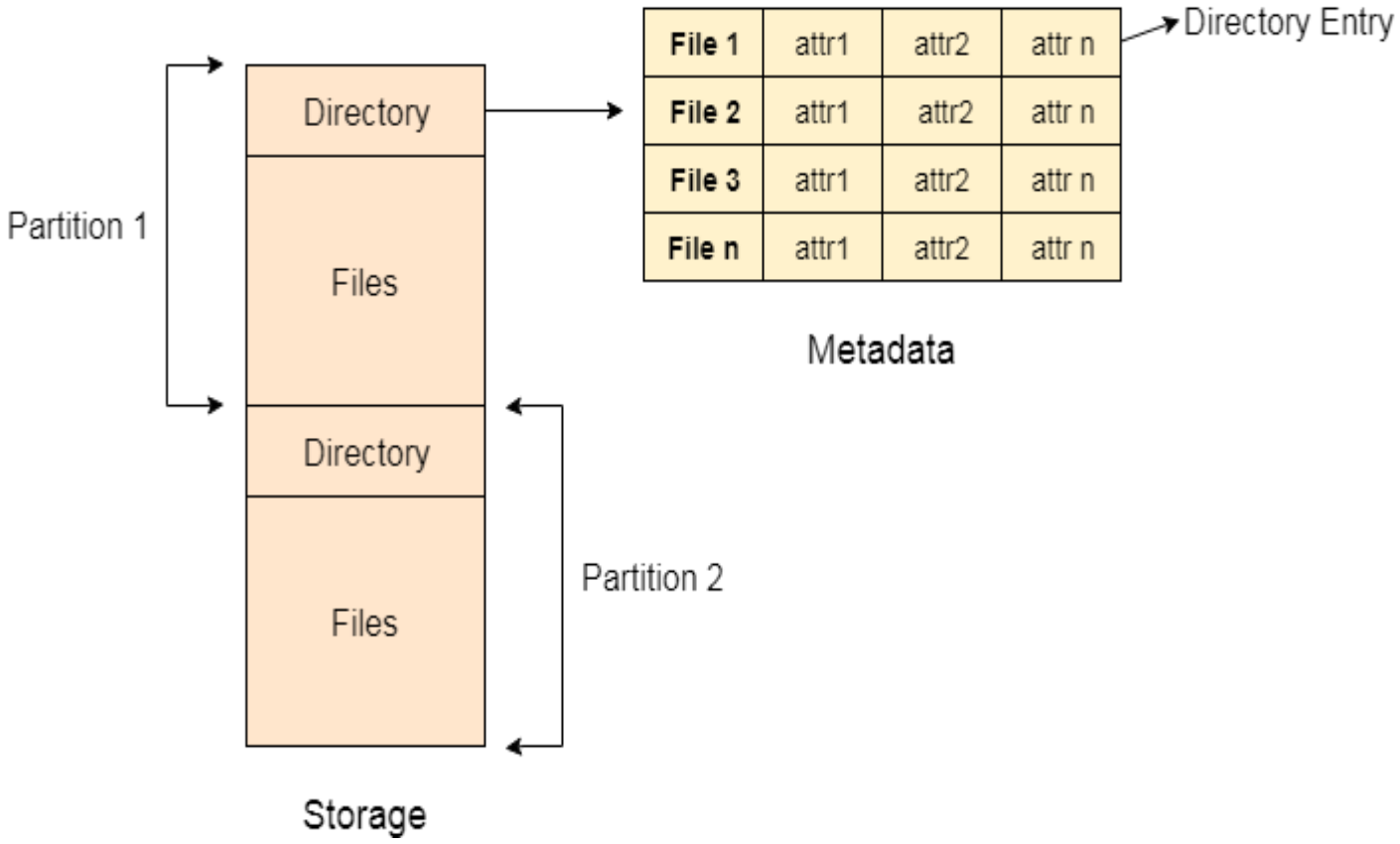


Directory Structure

- A collection of nodes containing information about all files



Both the directory structure and the files reside on disk


Directory Entry
File 1
attr1
attr2
attr n
File 2
attr1
attr2
attr n
File 3
attr1
attr2
attr n
File n
attr1
attr2
attr n
Metadata
Partition 1
Directory
Files
Directory
Files
Partition 2
Storage

Operations Performed on Directory

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

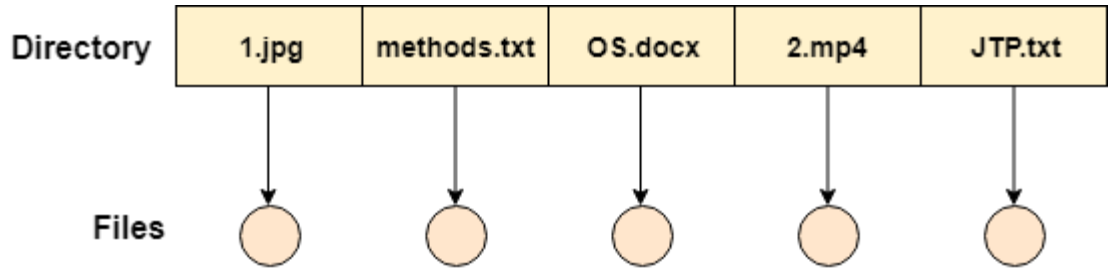
Directory Organization

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
- implementation is very simple.
- If the sizes of the files are very small then the searching becomes faster.
- File creation, searching, deletion is very simple since we have only one directory.



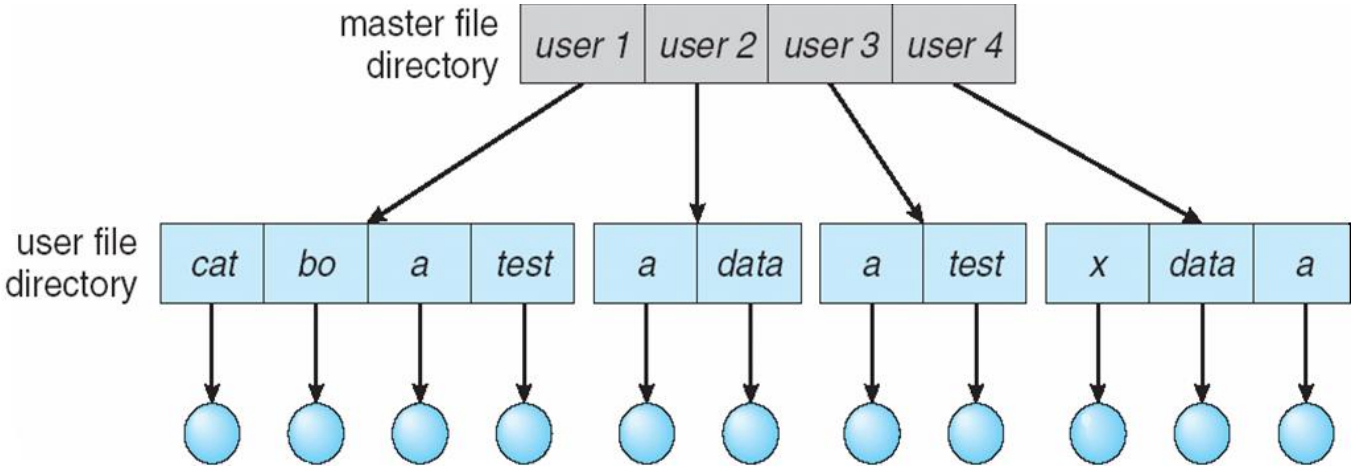
Single Level Directory

Disadvantages

- We cannot have two files with the same name.
- The directory may be very big therefore searching for a file may take so much time.
- Protection cannot be implemented for multiple users.
- There are no ways to group same kind of files.
- Choosing the unique name for every file is a bit complex and limits the number of files in the system because most of the Operating System limits the number of characters used to construct the file name.

Two-Level Directory

- Separate directory for each user



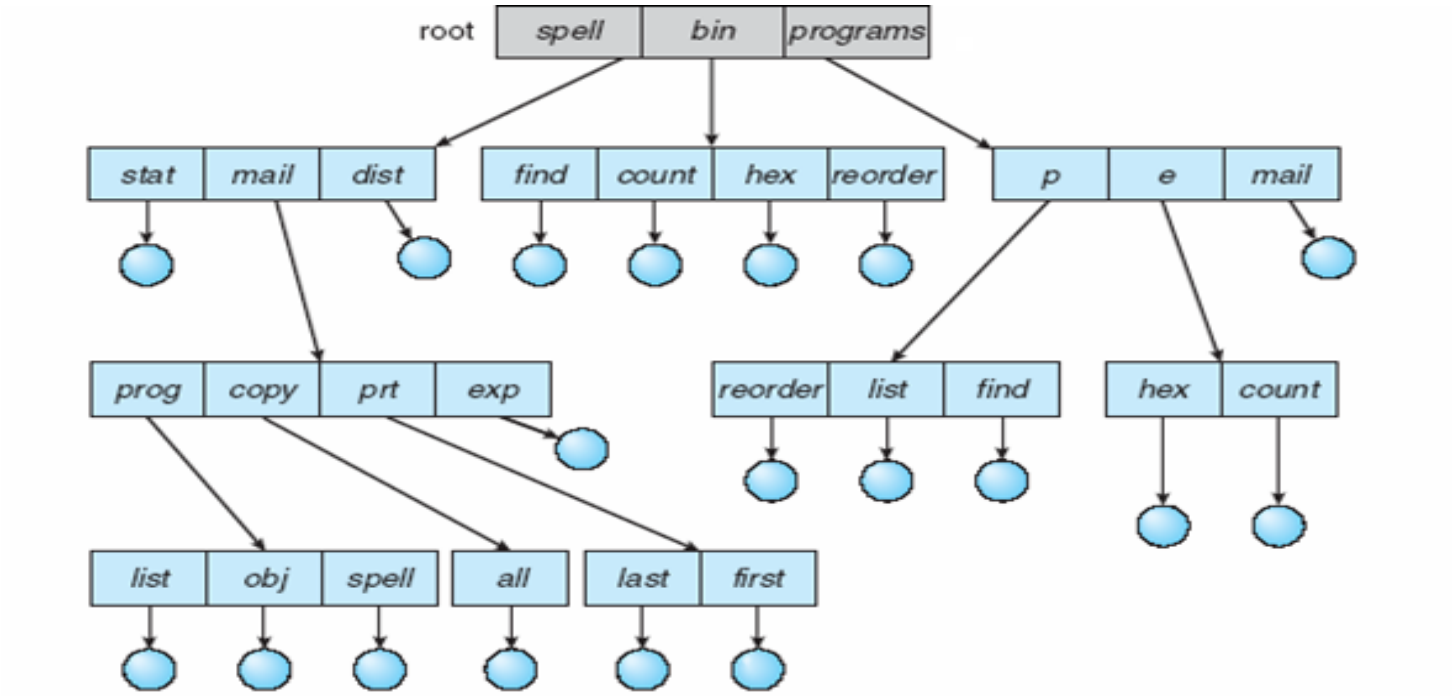
- Path name
- Can have the same file name for different user
- Efficient searching
- No grouping capability

Tree-Structured Directories

In Tree structured directory system, any directory entry can either be a file or sub directory.

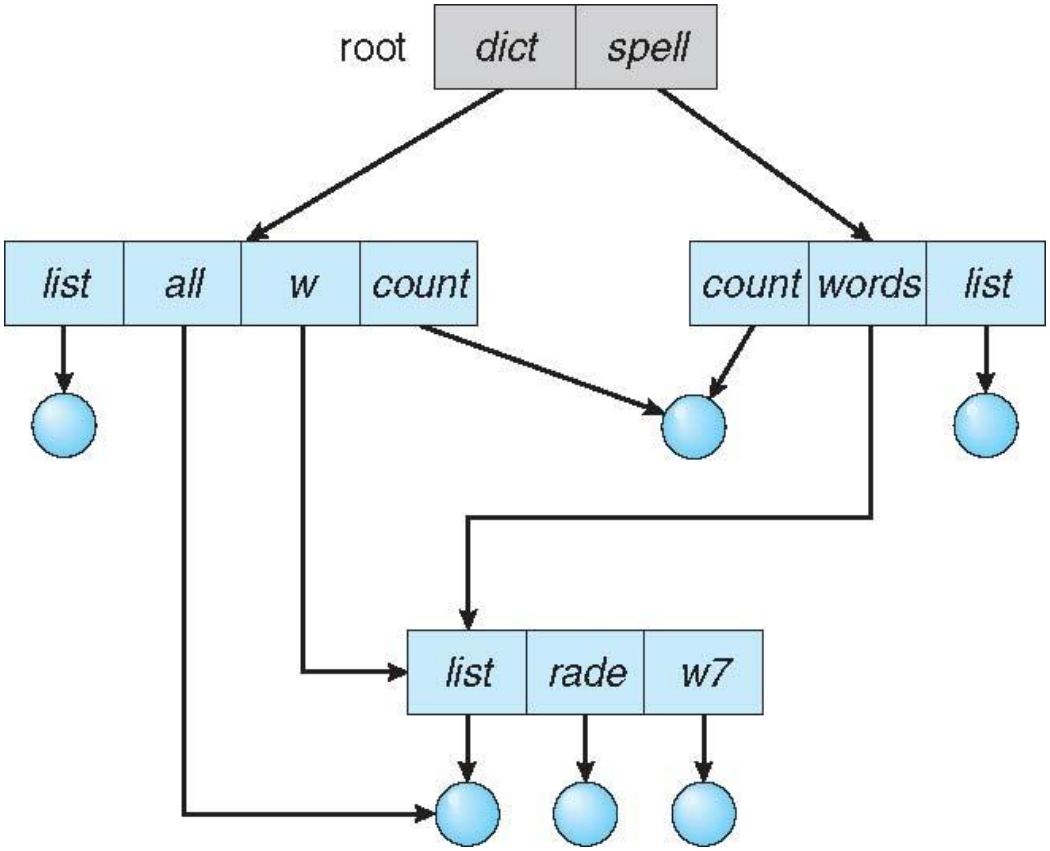
Tree structured directory system overcomes the drawbacks of two level directory system.

The similar kind of files can now be grouped in one directory.



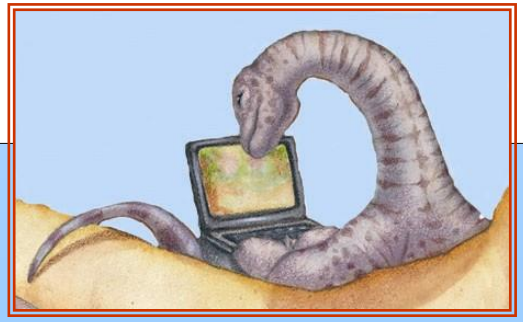
Acyclic-Graph Directories

- In this system, two or more directory entry can point to the same file or sub directory.



- Thank You.

File System Implementation



Allocation Methods

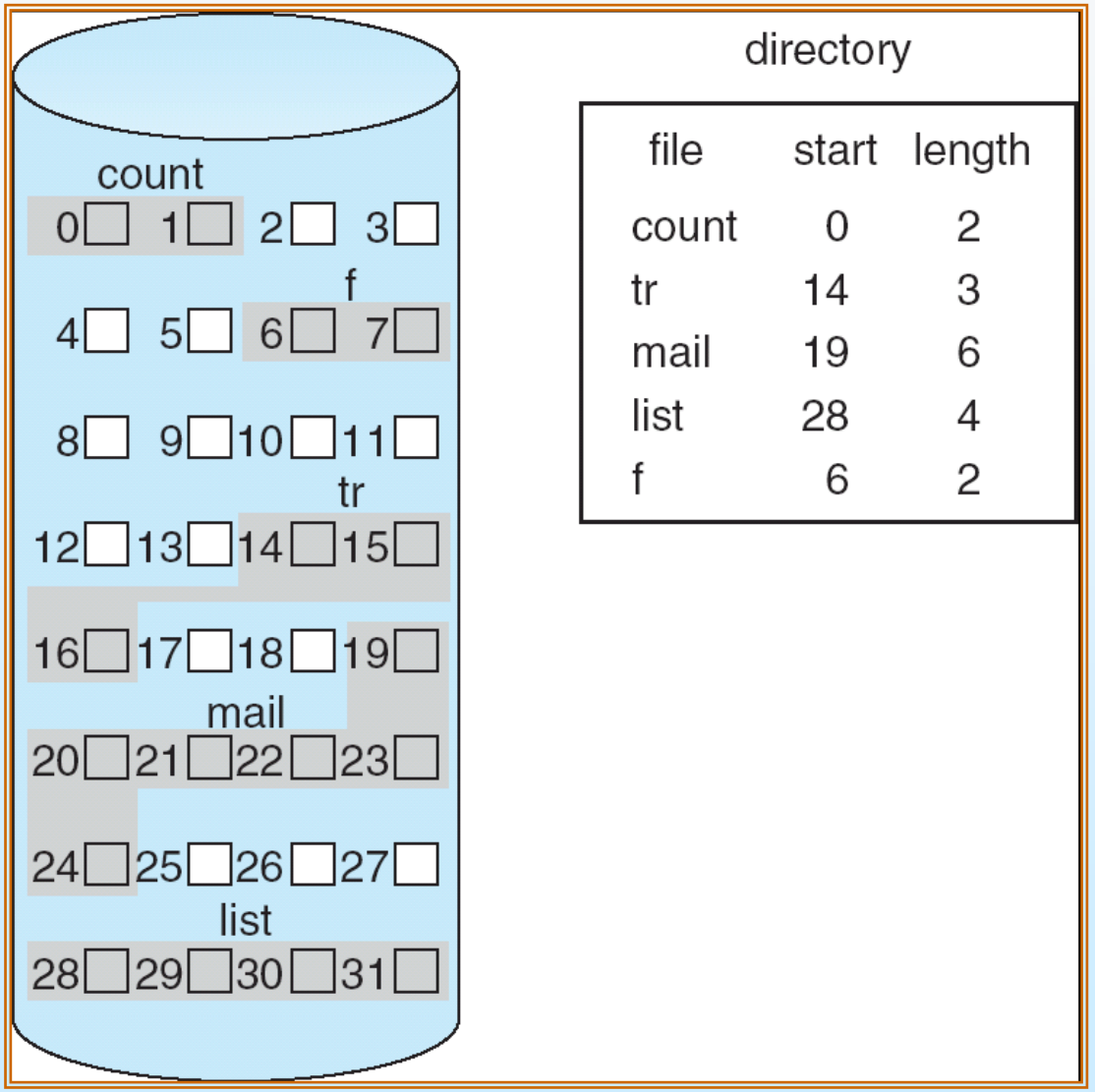
- An allocation method refers to how disk blocks are allocated for files:
- **Contiguous allocation**
- **Linked allocation**
- **Indexed allocation**

Contiguous Allocation

- Each file occupies a set of contiguous blocks on the disk
- Simple – only starting location (block #) and length (number of blocks) are required
- Sequential and Random access

If a file is n block long, and starts at location b, then it occipies b,b+1,b+2,… b+n-1

Contiguous Allocation of Disk Space



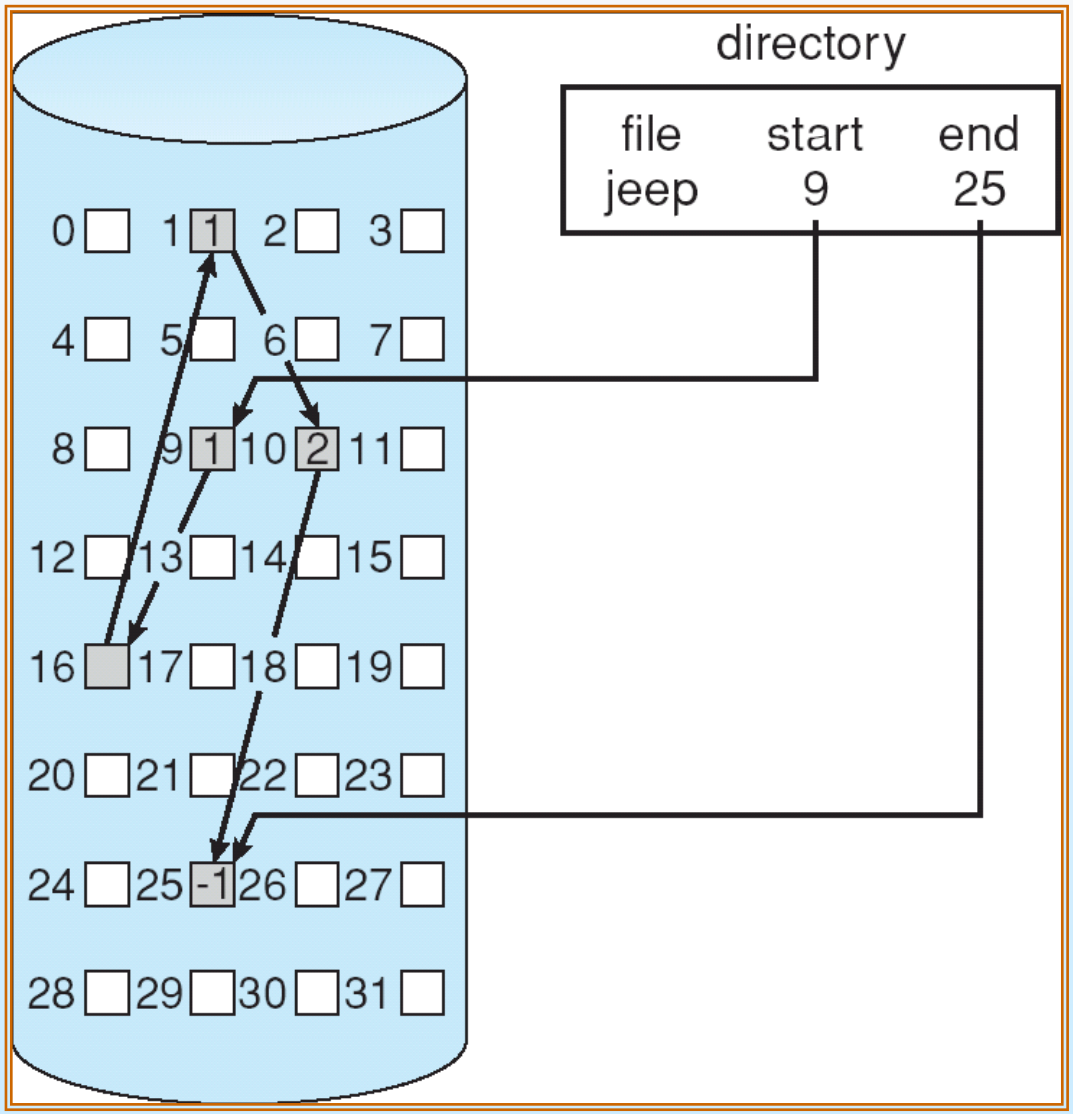
Problems

- Difficulty is finding space for a new file
- Determining how much space is needed for a file. When a file is created, the total amount of space it will need must be found and allocated.
- Wasteful of space
- Files cannot grow

Linked Allocation (Cont.)

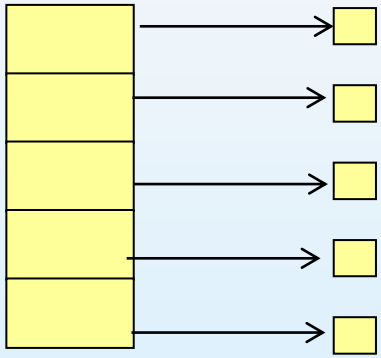
- Simple – need only starting address
- Free-space management system – no waste of space
- No random access

Linked Allocation



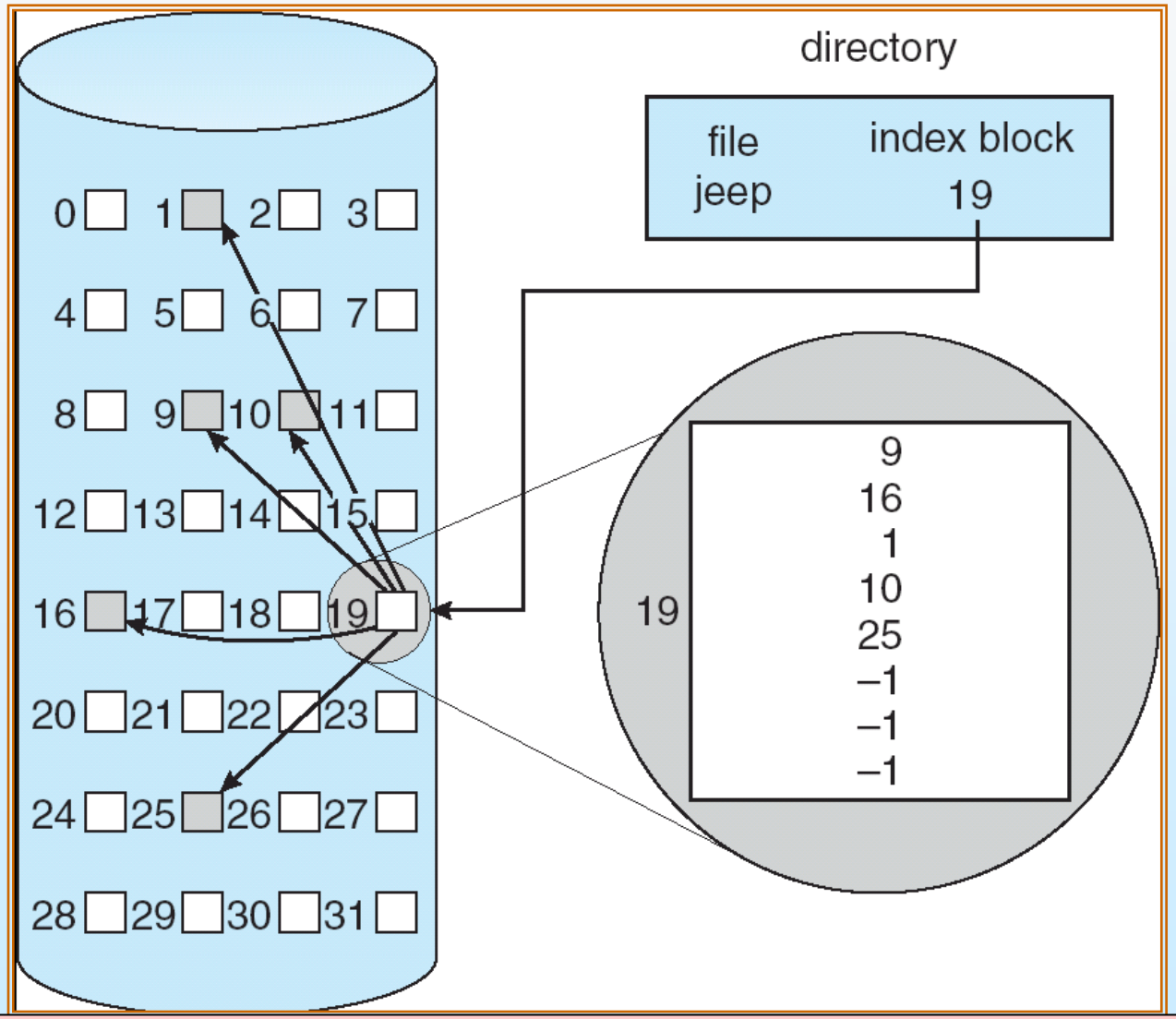
Indexed Allocation

- Linked allocation solves external fragmentation and size-declaration problem of contiguous allocation.
- Brings all pointers together into the *index block.*
- Logical view.



index table

Example of Indexed Allocation



Disadvantages

A bad index block could cause the lost of entire file.
Size of a file depends upon the number of pointers, a index block can hold.
Having an index block for a small file is totally wastage.
More pointer overhead

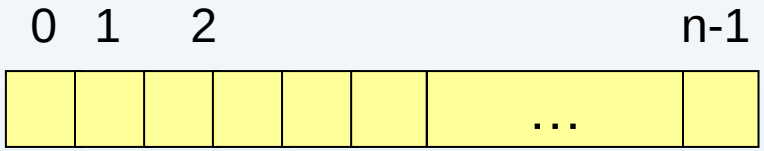
Free-Space Management

A file system is responsible to allocate the free blocks to the file therefore it has to keep track of all the free blocks present in the disk.

Free-Space Management

000011100000110.

- Bit vector (*n* blocks)



$$bit[i] = \begin{cases} 1 \Rightarrow block[i] \text{ free} \\ 0 \Rightarrow block[i] \text{ occupied} \end{cases}$$

Block number calculation

(number of bits per word) *
(number of 0-value words) +
offset of first 1 bit

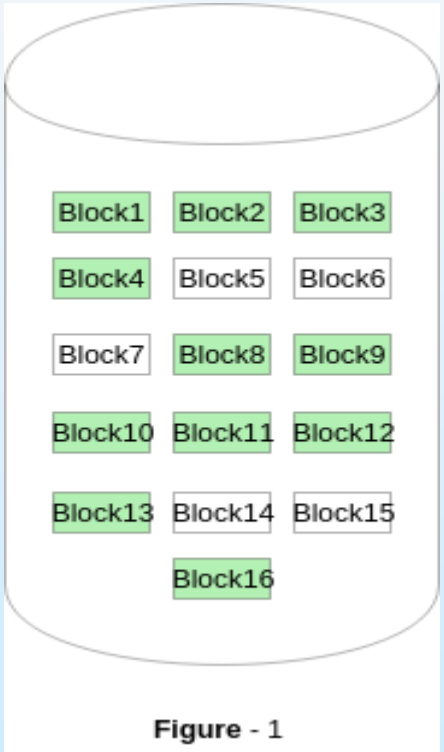
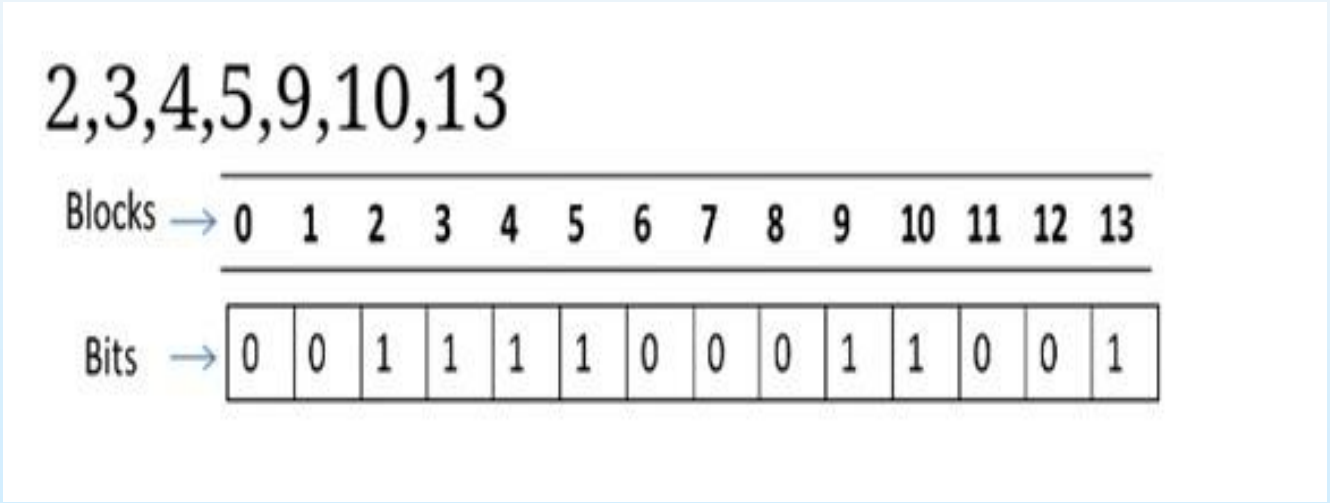


Figure - 1

- Assume the following are free. Rest are allocated:
- 2, 3,4,5,9,10,13



Free-Space Management (Cont.)

- Bit map requires extra space
  - Example:

    block size = 2^{12} bytes

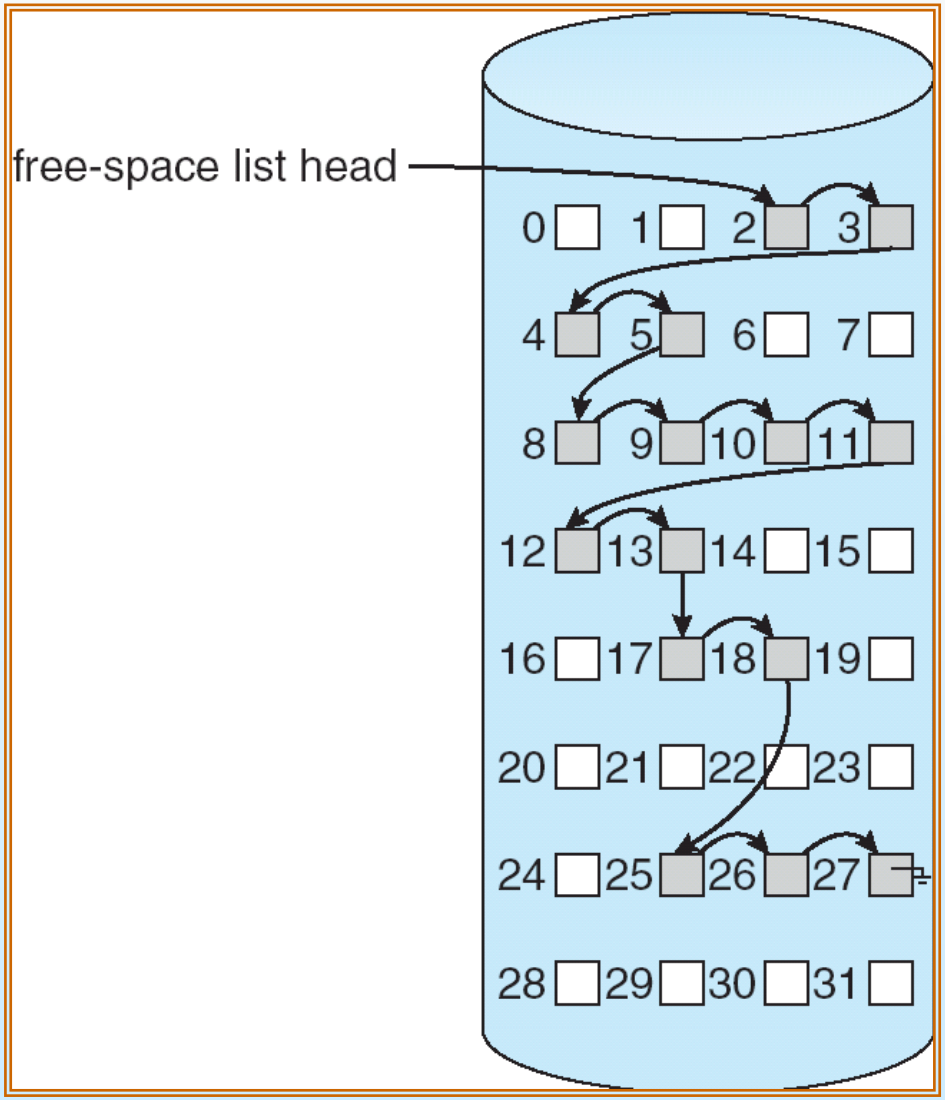
    disk size = 2^{30} bytes (1 gigabyte)

    $n = 2^{30}/2^{12} = 2^{18}$ bits (or 32K bytes)
- Easy to get contiguous files

Linked Free Space List on Disk

In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

Linked Free Space List on Disk



Directory Implementation

- Linear list of file names with pointer to the data blocks
  - simple to program
  - time-consuming to execute
- Hash Table – linear list with hash data structure
  - decreases directory search time
  - **collisions** – situations where two file names hash to the same location
  - fixed size

Grouping

- A free block contains *n pointers to free blocks*
- The last block among *n pointers contains another free blocks.*

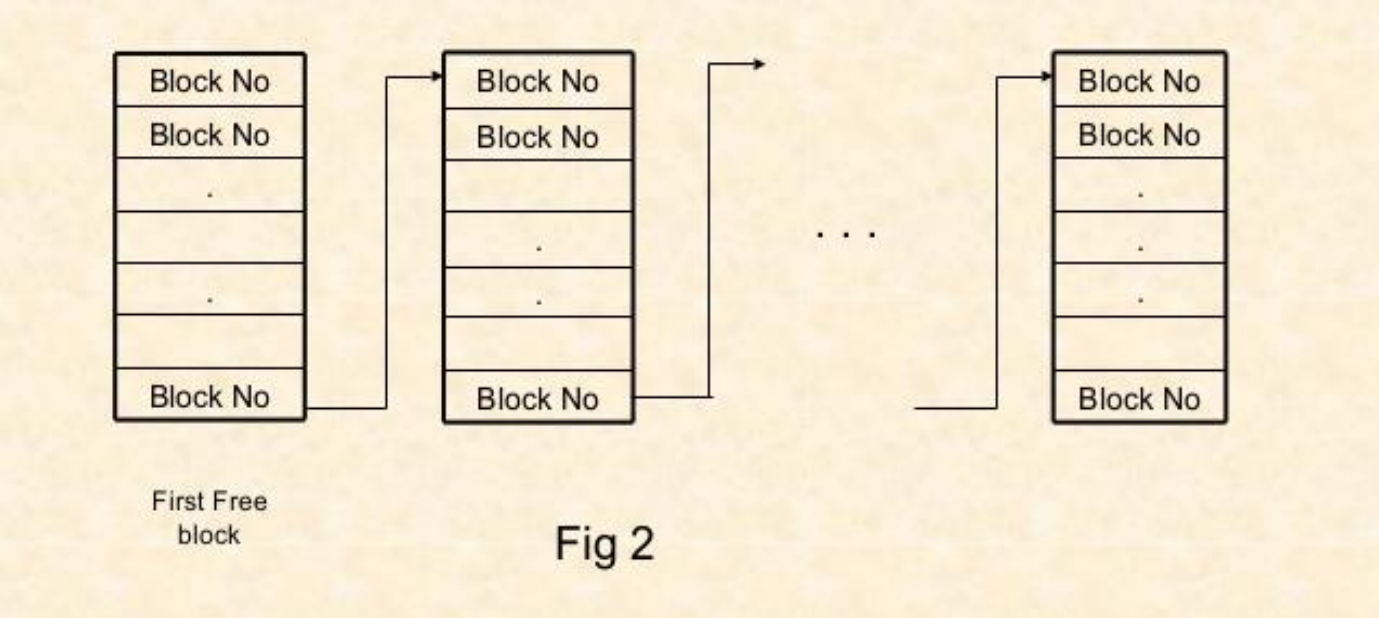


Fig 2

Example :

- Suppose we have a disk with some free blocks and some occupied blocks.
- The free block numbers are 3,4,5,6,9,10,11,12, and 14. And occupied block numbers are 1,2,7,8,15, and 16 i.e., they are allocated to some files.

Block 3 -> 4, 5, 6
Block 6 -> 9, 10, 11
Block 11 -> 12, 13, 14

Counting

- This approach stores the address of the first free disk block and a number n of free contiguous disk blocks that follow the first block.
- Every entry in the list would contain:
- Address of first free disk block
- A number n
- For example, *in Figure-1*, the first entry of the free space list would be: ([Address of Block 5], 2), because 2 contiguous free blocks follow block 5.

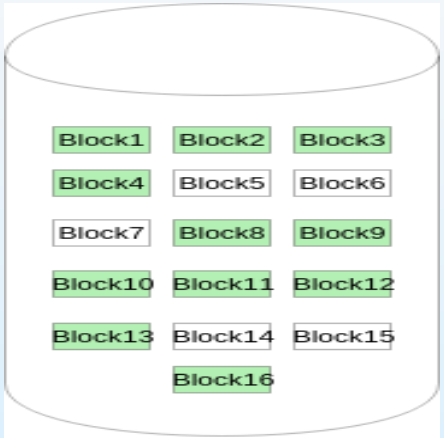
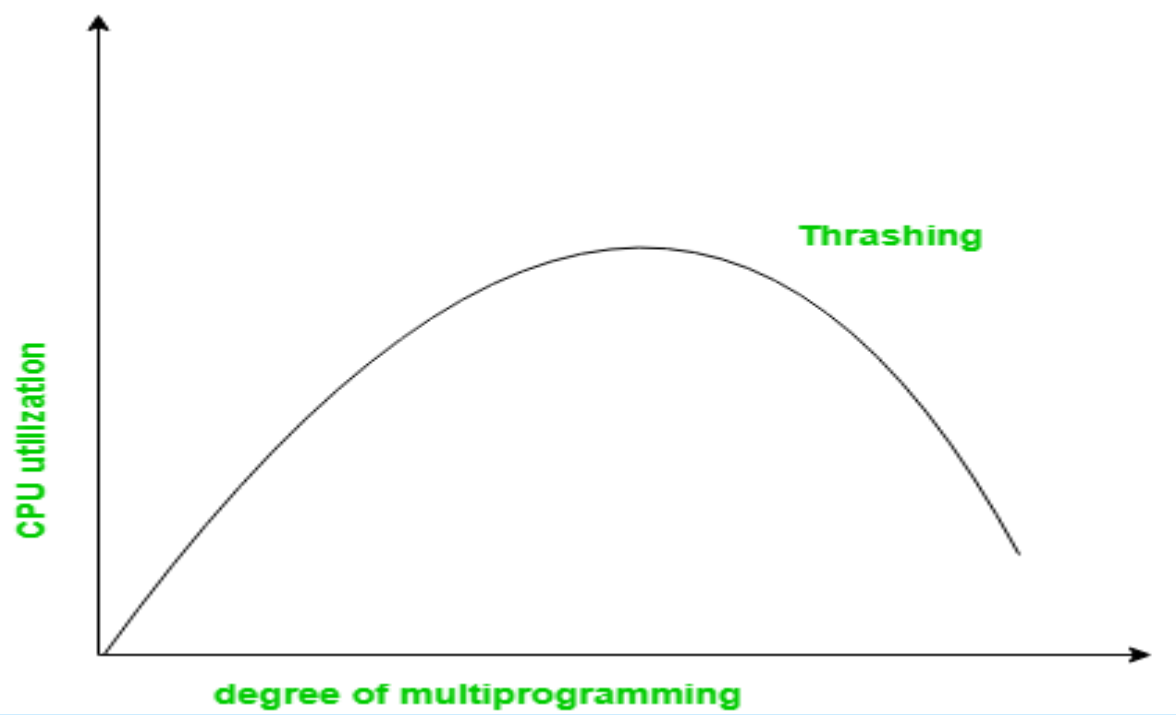


Figure - 1

Thrashing

- **Thrashing** is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



- The effect of Thrashing is

1) CPU becomes idle.
2) Decreasing the utilization increases the degree of multiprogramming and hence bringing more processes at a time which in fact increases the thrashing exponentially.

Techniques to Handle Thrashing

- **Working Set Model**
- This model is based on locality.
- What locality is saying, the page used recently can be used again and also the pages which are nearby this page will also be used.
- Working set means set of pages in the most recent D time.

Working Set Model

- Page reference 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 Maximum working set window: 5

Requested page	Time	Working set windows	Page fault (Hit/Miss)	Current windows
2	t_1	{2}	Page fault (Miss)	1
6	t_2	{2, 6}	Page fault (Miss)	2
1	t_3	{2, 6, 1}	Page fault (Miss)	3
5	t_4	{2, 6, 1, 5}	Page fault (Miss)	4
7	t_5	{2, 6, 1, 5, 7}	Page fault (Miss)	5
7	t_6	{6, 1, 5, 7}	Page fault (Hit)	4
7	t_7	{1, 5, 7}	Page fault (Hit)	3
5	t_8	{5, 7}	Page fault (Hit)	2
1	t_9	{1, 5, 7}	Page fault (Miss)	3
6	t_{10}	{1, 5, 7, 6}	Page fault (Miss)	4
2	t_{11}	{1, 5, 7, 6, 2}	Page fault (Miss)	5
3	t_{12}	{1, 5, 6, 2, 3}	Page fault (Miss)	5

4	t_{13}	{1, 6, 2, 3, 4}	Page fault (Miss)	5
1	t_{14}	{1, 6, 2, 3, 4}	Page fault (Miss)	5
2	t_{15}	{1, 2, 3, 4}	Page fault (Hit)	4
3	t_{16}	{1, 2, 3, 4}	Page fault (Hit)	4
4	t_{17}	{1, 2, 3, 4}	Page fault (Hit)	4
4	t_{18}	{1, 2, 3, 4}	Page fault (Hit)	4
4	t_{19}	{2, 3, 4}	Page fault (Hit)	3

Average Frame requirement=71/19=3.73

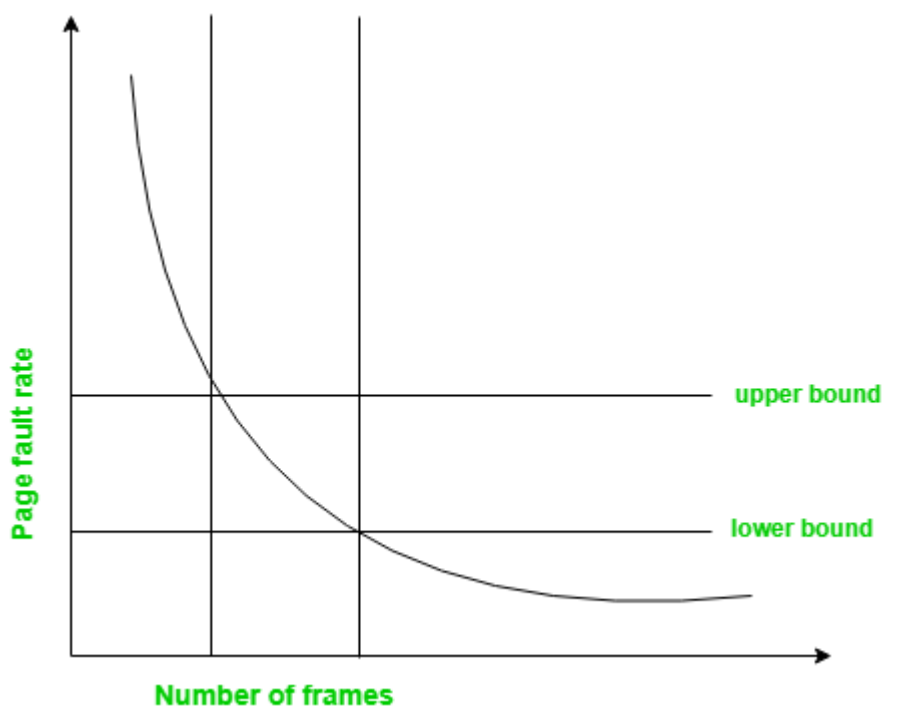
- If D is the total demand for frames and is the working set size for a process i,

$$D = \sum_{n} WSS_i$$

- Now, if ‘m’ is the number of frames available in the memory, there are 2 possibilities:

- (i) D>m i.e. total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- (ii) D<=m, then there would be no thrashing.
- If there are enough extra frames, then some more processes can be loaded in the memory. On the other hand, if the summation of working set sizes exceeds the availability of frames, then some of the processes have to be suspended(swapped out of memory).

Page Fault Frequency



- It is some direct approach than working set model.
- When thrashing occurring we know that it has few number of frames.
- And if it is not thrashing that means it has too many frames.
- Based on this property we assign an upper and lower bound for the desired page fault rate.
- According to page fault rate we allocate or remove pages.
- If the page fault rate become less than the lower limit, frames can be removed from the process.

- Similarly, if the page fault rate become more than the upper limit, more number of frames can be allocated to the process.
- And if no frames available due to high page fault rate, we will just suspend the processes and will restart them again when frames available.

Q1

Q) Disk requests are received by a disk drive for cylinders 5, 25, 18, 3, 39, 8, and 35 in that order. A seek takes 5 ms per cylinder moved. How much seek time is needed to serve these requests if serviced in the order that they are received (FCFS)? Assume that the arm is at cylinder 20 when the last of these requests is made with none of these requests yet served.

FCFS=151

SSTF=59

SCAN=57

LOOK=55

Q2

- **Consider a reference string: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2. the number of frames in the memory is 3. Find out the number of page faults respective to:**
- Optimal Page Replacement Algorithm — 5
- FIFO Page Replacement Algorithm — 6
- LRU Page Replacement Algorithm — 6

Q3

- Consider a paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB and 100 ns to access the physical memory. If TLB hit ratio is 80%, the effective memory access time is _____ msec.

- hit ratio * (TLB access time + Main memory access time) +
(1 – hit ratio) * (TLB access time + 2 * main memory time)

1)

- In _____ information is recorded magnetically on platters.
  a) magnetic disks
  b) electrical disks
  c) assemblies
  d) cylinders

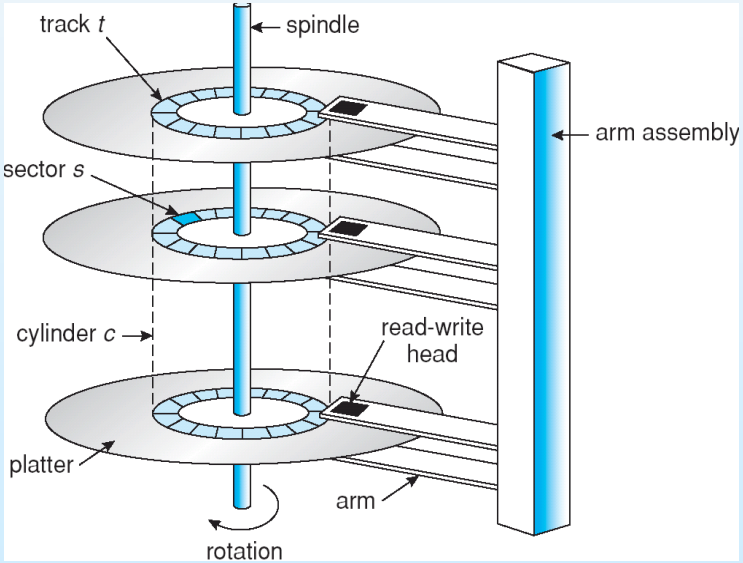
2)

- The heads of the magnetic disk are attached to a _____ that moves all the heads as a unit.

  a) spindle
  b) disk arm
  c) track
  d) none of the mentioned

3)

- The set of tracks that are at one arm position make up a _____
  a) magnetic disks
  b) electrical disks
  c) assemblies
  d) cylinders



4)

- The time taken to move the disk arm to the desired cylinder is called the _____
  a) positioning time
  b) random access time
  c) seek time
  d) rotational latency

5)

- The time taken for the desired sector to rotate to the disk head is called _____
  a) positioning time
  b) random access time
  c) seek time
  d) rotational latency

6)

- . _____ is a unique tag, usually a number identifies the file within the file system.
  a) File identifier
  b) File name
  c) File type
  d) None of the mentioned

7)

- File type can be represented by _____
  a) file name
  b) file extension
  c) file identifier
  d) none of the mentioned

8)

- Which one of the following is the address generated by CPU?
  a) physical address
  b) absolute address
  c) logical address
  d) none of the mentioned

9)

- Run time mapping from virtual to physical address is done by _____
  a) Memory management unit
  b) CPU
  c) PCI
  d) None of the mentioned

10)

- What is compaction?
  a) a technique for overcoming internal fragmentation
  b) a paging technique
  c) a technique for overcoming external fragmentation
  d) a technique for overcoming fatal error

11)

- DMA is used for _____
  a) High speed devices(disks and communications network)
  b) Low speed devices
  c) Utilizing CPU cycles
  d) All of the mentioned

12)

- In an interrupt driven input/output _____
  a) the CPU uses polling to watch the control bit constantly, looping to see if a device is ready
  b) the CPU writes one data byte to the data register and sets a bit in control register to show that a byte is available
  c) the CPU receives an interrupt when the device is ready for the next byte
  d) the CPU runs a user written code and does accordingly

13)

- At least one resource must be held in a non sharable mode. Only one process at a time can use a resource. These statements are true for which deadlock necessary condition?
- A. Mutual Exclusion
- B. Hold and wait
- C. Circular wait
- D. No preemption

- What is convoy effect?
- What is Seek time and rotational latency time?
- What is Belady's Anomalies?
- What is thrashing? What are the different techniques to handle it?

Q14.

Let m[0]…m[4] be mutexes (binary semaphores) and P[0]…P[4] be processes. Suppose each process P[i] executes the following:

wait(m[i]);

wait(m[(i+1) mod 4]);

…….

release(m[i]);

release(m[(i+1) mod 4]);

This could cause-

(a) Thrashing (b) Deadlock (c) Starvation but not deadlock

(d) None of the above

MCQ

- A solution to the critical-section problem must satisfy the following requirements :
- **Mutual Exclusion :**
- **Progress:**
- **Bounded Waiting**
- **Circular wait**
- **Hold and wait**

Consider the following C code for process P1 and P2. a=4, b=0, c=0 (initialization)

```
   P1                    P2
if (a < 0)            b = 10;
  c = b-a;            a = -3;
else
  c = b+a;
```

If the processes P1 and P2 executes concurrently (shared variables a, b and c), which of the following can be the value of ‘c’ after both processes complete?

(A) 4 **(B)** 7 **(C)** 10 **(D)** 13

Answer: (C)
Explanation:
P1 : 1, 3, 4 -> c = 0+4 =4 {hence option a}
P2 : i, ii and P1 : 1, 2 -> c = 10-(-3) = 13 {hence option d}
P1 : 1 , P2 : i, ii and P1 : 3, 4 -> c= 10+(-3) = 7 { hence option b}

- Thank You. All the Best