

# OPERATING SYSTEM

## UNIT-3

### Deadlock

Every process needs some resources to complete its execution. However, the resource is granted in a sequential order.

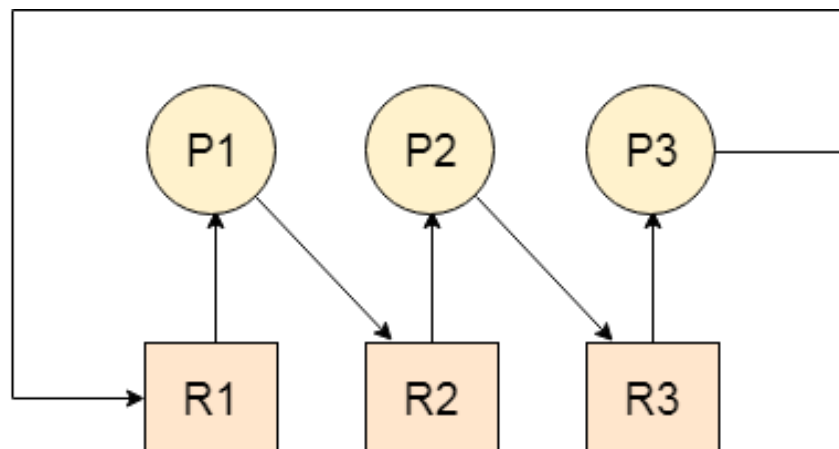
1. The process requests for some resource.
2. OS grant the resource if it is available otherwise let the process waits.
3. The process uses it and release on the completion.

A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the process gets executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released.

Let us assume that there are three processes P1, P2 and P3. There are three different resources R1, R2 and R3. R1 is assigned to P1, R2 is assigned to P2 and R3 is assigned to P3.

After some time, P1 demands for R2 which is being used by P2. P1 halts its execution since it can't complete without R2. P2 also demands for R3 which is being used by P3. P2 also stops its execution because it can't continue without R3. P3 also demands for R1 which is being used by P1 therefore P3 also stops its execution.

In this scenario, a cycle is being formed among the three processes. None of the process is progressing and they are all waiting. The computer becomes unresponsive since all the processes got blocked.



## System Model:-

- For the purposes of deadlock discussion, a system can be modeled as a collection of limited resources, which can be partitioned into different categories, to be allocated to a number of processes, each having different needs.
- Resource categories may include memory, printers, CPUs, open files, tape drives, CD-ROMS, etc.
- By definition, all the resources within a category are equivalent, and a request of this category can be equally satisfied by any one of the resources in that category. If this is not the case ( i.e. if there is some difference between the resources within a category ), then that category needs to be further divided into separate categories. For example, "printers" may need to be separated into "laser printers" and "color inkjet printers".
- Some categories may have a single resource.
- In normal operation a process must request a resource before using it, and release it when it is done, in the following sequence:
  1. **Request** - If the request cannot be immediately granted, then the process must wait until the resource(s) it needs become available. For example the system calls `open( )`, `malloc( )`, `new( )`, and `request( )`.
  2. **Use** - The process uses the resource, e.g. prints to the printer or reads from the file.
  3. **Release** - The process relinquishes the resource. so that it becomes available for other processes. For example, `close( )`, `free( )`, `delete( )`, and `release( )`.
- For all kernel-managed resources, the kernel keeps track of what resources are free and which are allocated, to which process they are allocated, and a queue of processes waiting for this resource to become available. Application-managed resources can be controlled using mutexes or `wait( )` and `signal( )` calls, ( i.e. binary or counting semaphores. )
- A set of processes is deadlocked when every process in the set is waiting for a resource that is currently allocated to another process in the set ( and which can only be released when that other waiting process makes progress.)

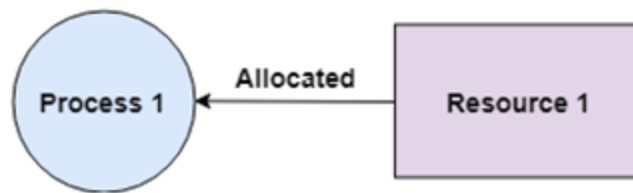
## Deadlock Characterization:-

### Necessary conditions for Deadlocks

#### 1. Mutual Exclusion

A resource can only be shared in mutually exclusive manner. It implies, if two process cannot use the same resource at the same time.

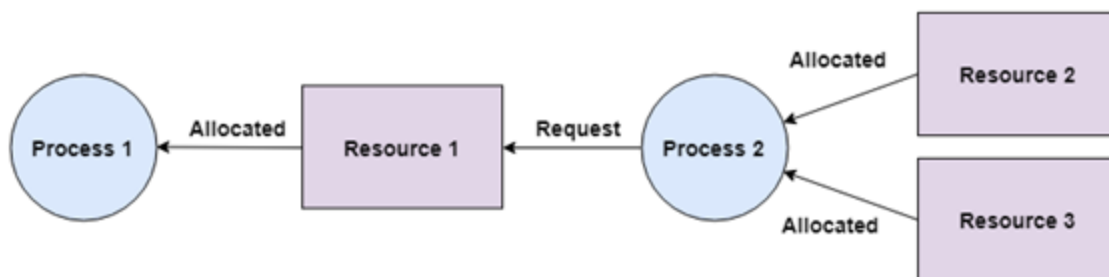
There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



## 2. Hold and Wait

A process waits for some resources while holding another resource at the same time.

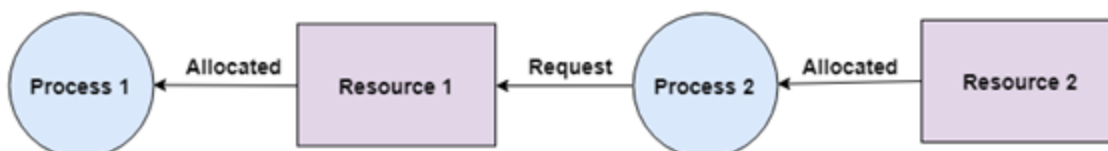
A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



## 3. No preemption

The process which once scheduled will be executed till the completion. No other process can be scheduled by the scheduler meanwhile.

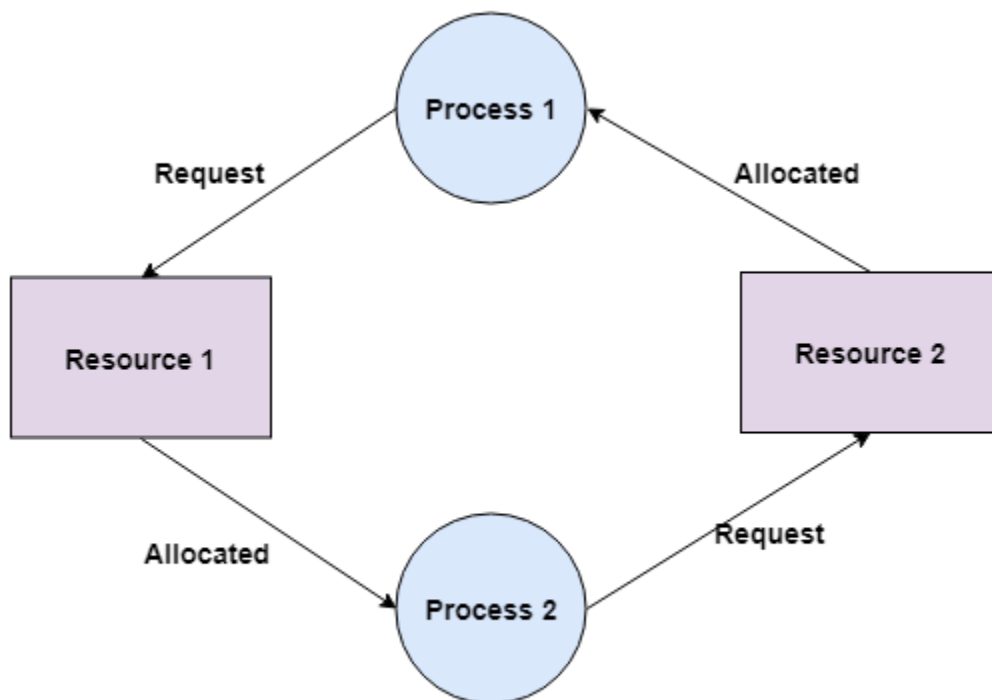
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



#### 4. Circular Wait

All the processes must be waiting for the resources in a cyclic manner so that the last process is waiting for the resource which is being held by the first process.

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop.



#### Methods for Handling Deadlocks

- Generally speaking there are three ways of handling deadlocks:
  1. **Deadlock prevention or avoidance** - Do not allow the system to get into a deadlocked state. In order to avoid deadlocks, the system must have additional information about all processes. In particular, the system must know what resources a process will or may request in the future. (Ranging from a simple worst-case maximum to a complete resource request and release plan for each process, depending on the particular algorithm.)

2. **Deadlock detection and recovery** - Abort a process or preempt some resources when deadlocks are detected. Deadlock detection is fairly straightforward, but deadlock recovery requires either aborting processes or preempting resources, neither of which is an attractive alternative.
3. **Ignore the problem all together** - If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take. If deadlocks are neither prevented nor detected, then when a deadlock occurs the system will gradually slow down, as more and more processes become stuck waiting for resources currently held by the deadlock and by other waiting processes. Unfortunately this slowdown can be indistinguishable from a general system slowdown when a real-time process has heavy computing needs.

## 1. Deadlock Prevention

If we simulate deadlock with a table which is standing on its four legs then we can also simulate four legs with the four conditions which when occurs simultaneously, cause the deadlock.

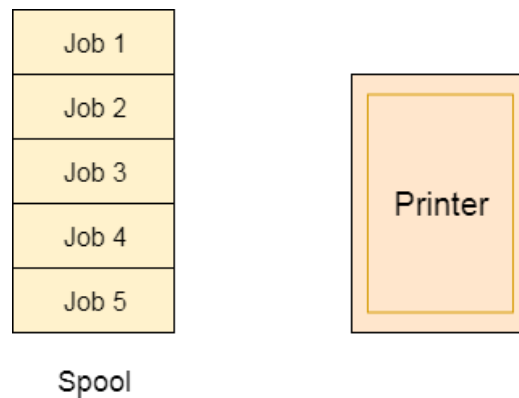
However, if we break one of the legs of the table then the table will fall definitely. The same happens with deadlock, if we can be able to violate one of the four necessary conditions and don't let them occur together then we can prevent the deadlock.

Let's see how we can prevent each of the conditions.

## Mutual Exclusion

### Spooling

For a device like printer, spooling can work. There is a memory associated with the printer which stores jobs from each of the process into it. Later, Printer collects all the jobs and print each one of them according to FCFS. By using this mechanism, the process doesn't have to wait for the printer and it can continue whatever it was doing. Later, it collects the output when it is produced.



Although, Spooling can be an effective approach to violate mutual exclusion but it suffers from two kinds of problems.

1. This cannot be applied to every resource.
2. After some point of time, there may arise a race condition between the processes to get space in that spool.

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. Therefore, we cannot violate mutual exclusion for a process practically.

## 2. Hold and Wait

Hold and wait condition lies when a process holds a resource and waiting for some other resource to complete its task. Deadlock occurs because there can be more than one process which are holding one resource and waiting for other in the cyclic order.

However, we have to find out some mechanism by which a process either doesn't hold any resource or doesn't wait. That means, a process must be assigned all the necessary resources before the execution starts. A process must not wait for any resource once the execution has been started.

**!(Hold and wait) = !hold or !wait (negation of hold and wait is, either you don't hold or you don't wait)**

This can be implemented practically if a process declares all the resources initially. However, this sounds very practical but can't be done in the computer system because a process can't determine necessary resources initially.

Process is the set of instructions which are executed by the CPU. Each of the instruction may demand multiple resources at the multiple times. The need cannot be fixed by the OS.

The problem with the approach is:

1. Practically not possible.
2. Possibility of getting starved will be increases due to the fact that some process may hold a resource for a very long time.

### **3. No Preemption**

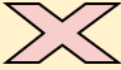
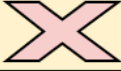


Deadlock arises due to the fact that a process can't be stopped once it starts. However, if we take the resource away from the process which is causing deadlock then we can prevent deadlock.

This is not a good approach at all since if we take a resource away which is being used by the process then all the work which it has done till now can become inconsistent.

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.

### **4. Circular Wait**

To violate circular wait, we can assign a priority number to each of the resource. A process can't request for a lesser priority resource. This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

Condition	Approach	Is Practically Possible?
Mutual Exclusion	Spooling	
Hold and Wait	Request for all the resources initially	
No Preemption	Snatch all the resources	
Circular Wait	Assign priority to each resources and order resources numerically	

Among all the methods, violating Circular wait is the only approach that can be implemented practically.

### **Deadlock Avoidance:-**

In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.

In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.

The simplest and most useful approach states that the process should declare the maximum number of resources of each type it may ever need. The Deadlock avoidance algorithm examines the resource allocations so that there can never be a circular wait condition.

### **Safe and Unsafe States**

The resource allocation state of a system can be defined by the instances of available and allocated resources, and the maximum instance of the resources demanded by the processes.

A state of a system recorded at some random time is shown below.



## Resources Assigned

Process	Type 1	Type 2	Type 3	Type 4
A	3	0	2	2
B	0	0	1	1
C	1	1	1	0
D	2	1	4	0

## Resources still needed

Process	Type 1	Type 2	Type 3	Type 4
A	1	1	0	0
B	0	1	1	2
C	1	2	1	0
D	2	1	1	2

$$E = (7 \ 6 \ 8 \ 4)$$

$$P = (6 \ 2 \ 8 \ 3)$$

$$A = (1 \ 4 \ 0 \ 1)$$

Above tables and vector E, P and A describes the resource allocation state of a system. There are 4 processes and 4 types of the resources in a system. Table 1 shows the instances of each resource assigned to each process.

Table 2 shows the instances of the resources, each process still needs. Vector E is the representation of total instances of each resource in the system.

Vector P represents the instances of resources that have been assigned to processes. Vector A represents the number of resources that are not in use.

A state of the system is called safe if the system can allocate all the resources requested by all the processes without entering into deadlock.

If the system cannot fulfill the request of all processes then the state of the system is called unsafe.

The key of Deadlock avoidance approach is when the request is made for resources then the request must only be approved in the case if the resulting state is also a safe state.

### **Banker's Algorithm:-**

Banker's algorithm is a **deadlock avoidance algorithm**. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are  $n$  account holders in a bank and the sum of the money in all of their accounts is  $S$ . Every time a loan has to be granted by the bank, it subtracts the **loan amount** from the **total money** the bank has. Then it checks if that difference is greater than  $S$ . It is done because, only then, the bank would have enough money even if all the  $n$  account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly.

Let us assume that there are  $n$  processes and  $m$  resource types. Some data structures that are used to implement the banker's algorithm are:

### 1. Available

It is an **array** of length  $m$ . It represents the number of available resources of each type. If  $Available[j] = k$ , then there are  $k$  instances available, of resource type  $R(j)$ .

### 2. Max

It is an  $n \times m$  matrix which represents the maximum number of instances of each resource that a process can request. If  $Max[i][j] = k$ , then the process  $P(i)$  can request atmost  $k$  instances of resource type  $R(j)$ .

### 3. Allocation

It is an  $n \times m$  matrix which represents the number of resources of each type currently allocated to each process. If  $Allocation[i][j] = k$ , then process  $P(i)$  is currently allocated  $k$  instances of resource type  $R(j)$ .

### 4. Need

It is an  $n \times m$  matrix which indicates the remaining resource needs of each process. If  $Need[i][j] = k$ , then process  $P(i)$  may need  $k$  more instances of resource type  $R(j)$  to complete its task.

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

### Safety Algorithm:-

**This algo used for finding out whether a system in a safe state or not.**

1) Let *Work* and *Finish* be vectors of length ' $m$ ' and ' $n$ ' respectively.

Initialize:  $Work = Available$

$Finish[i] = false$ ; for  $i=1, 2, 3, 4, \dots, n$

2) Find an  $i$  such that both

a)  $Finish[i] = false$

b)  $Need_i \leq Work$

if no such  $i$  exists goto 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

## Resource-Request Algorithm

Let  $Request_i$  be the request array for process  $P_i$ .  $Request_i[j] = k$  means process  $P_i$  wants  $k$  instances of resource type  $R_j$ . When a request for resources is made by process  $P_i$ , the following actions are taken:

1) If  $Request_i \leq Need_i$

Goto step (2) ; otherwise, raise an error condition, since the process has exceeded its maximum claim.

2) If  $Request_i \leq Available$

Goto step (3); otherwise,  $P_i$  must wait, since the resources are not available.

3) Have the system pretend to have allocated the requested resources to process  $P_i$  by modifying the state as follows:

$Available = Available - Request_i$

$Allocation_i = Allocation_i + Request_i$

$Need_i = Need_i - Request_i$

### Example:

Considering a system with five processes  $P_0$  through  $P_4$  and three resources of type A, B, C. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time  $t_0$  following snapshot of the system has been taken:

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

### Question1. What will be the content of the Need matrix?

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

For  $P_0 = (7, 5, 3) - (0, 1, 0)$

Need for  $P_0 = (7, 4, 3)$

For  $P_1 = (3, 2, 2) - (2, 0, 0)$

Need for  $P_1 = (1, 2, 2)$

For  $P_2 = (9,0,2) - (3,0,2)$

Need for  $P_2 = (6,0,0)$

For  $P_3 = (2,2,2) - (2,1,1)$

Need for  $P_3 = (0,1,1)$

For  $P_4 = (4,3,3) - (0,0,2)$

Need for  $P_4 = (4,3,1)$

So, the content of Need Matrix is:

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

## Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Applying the Safety algorithm on the given system,

$m=3, n=5$  Step 1 of Safety Algo

Work = Available

Work = 

3	3	2
---	---	---

0      1      2      3      4

Finish = 

false	false	false	false	false
-------	-------	-------	-------	-------

For  $i=0$  Step 2:

Need<sub>0</sub> = 7, 4, 3 ✗

Finish [0] is false and Need<sub>0</sub> > Work ✗

So P<sub>0</sub> must wait But Need ≤ Work

For  $i=1$  Step 2:

Need<sub>1</sub> = 1, 2, 2 ✓

Finish [1] is false and Need<sub>1</sub> < Work ✓

So P<sub>1</sub> must be kept in safe sequence

Step 3

Work = Work + Allocation<sub>1</sub>

Work = 

5	3	2
---	---	---

0      1      2      3      4

Finish = 

false	true	false	false	false
-------	------	-------	-------	-------

For  $i=2$  Step 2:

Need<sub>2</sub> = 6, 0, 0 ✗

Finish [2] is false and Need<sub>2</sub> > Work ✗

So P<sub>2</sub> must wait

For  $i=3$  Step 2:

Need<sub>3</sub> = 0, 1, 1 ✓

Finish [3] = false and Need<sub>3</sub> < Work ✓

So P<sub>3</sub> must be kept in safe sequence

Step 3

Work = Work + Allocation<sub>3</sub>

Work = 

7	4	3
---	---	---

0      1      2      3      4

Finish = 

false	true	false	true	false
-------	------	-------	------	-------

For  $i=4$  Step 2:

Need<sub>4</sub> = 4, 3, 1 ✓

Finish [4] = false and Need<sub>4</sub> < Work ✓

So P<sub>4</sub> must be kept in safe sequence

Step 3

Work = Work + Allocation<sub>4</sub>

Work = 

7	4	5
---	---	---

0      1      2      3      4

Finish = 

false	true	false	true	true
-------	------	-------	------	------

For  $i=0$  Step 2:

Need<sub>0</sub> = 7, 4, 3 ✓

Finish [0] is false and Need < Work ✓

So P<sub>0</sub> must be kept in safe sequence

Step 3

Work = Work + Allocation<sub>0</sub>

Work = 

7	5	5
---	---	---

0      1      2      3      4

Finish = 

true	true	false	true	true
------	------	-------	------	------

For  $i=2$  Step 2:

Need<sub>2</sub> = 6, 0, 0 ✓

Finish [2] is false and Need<sub>2</sub> < Work ✓

So P<sub>2</sub> must be kept in safe sequence

Step 3

Work = Work + Allocation<sub>2</sub>

Work = 

10	5	7
----	---	---

0      1      2      3      4

Finish = 

true	true	true	true	true
------	------	------	------	------

Step 4

Finish [i] = true for  $0 \leq i \leq n$

Hence the system is in Safe state

The safe sequence is P<sub>1</sub>, P<sub>3</sub>, P<sub>4</sub>, P<sub>0</sub>, P<sub>2</sub>

**Question3.** What will happen if process  $P_1$  requests one additional instance of resource type A and two instances of resource type C?

A B C  
Request<sub>1</sub> = 1, 0, 2

To decide whether the request is granted we use Resource Request algorithm

1, 0, 2      1, 2, 2 ✓ Step 1  
Request<sub>1</sub> < Need<sub>1</sub>

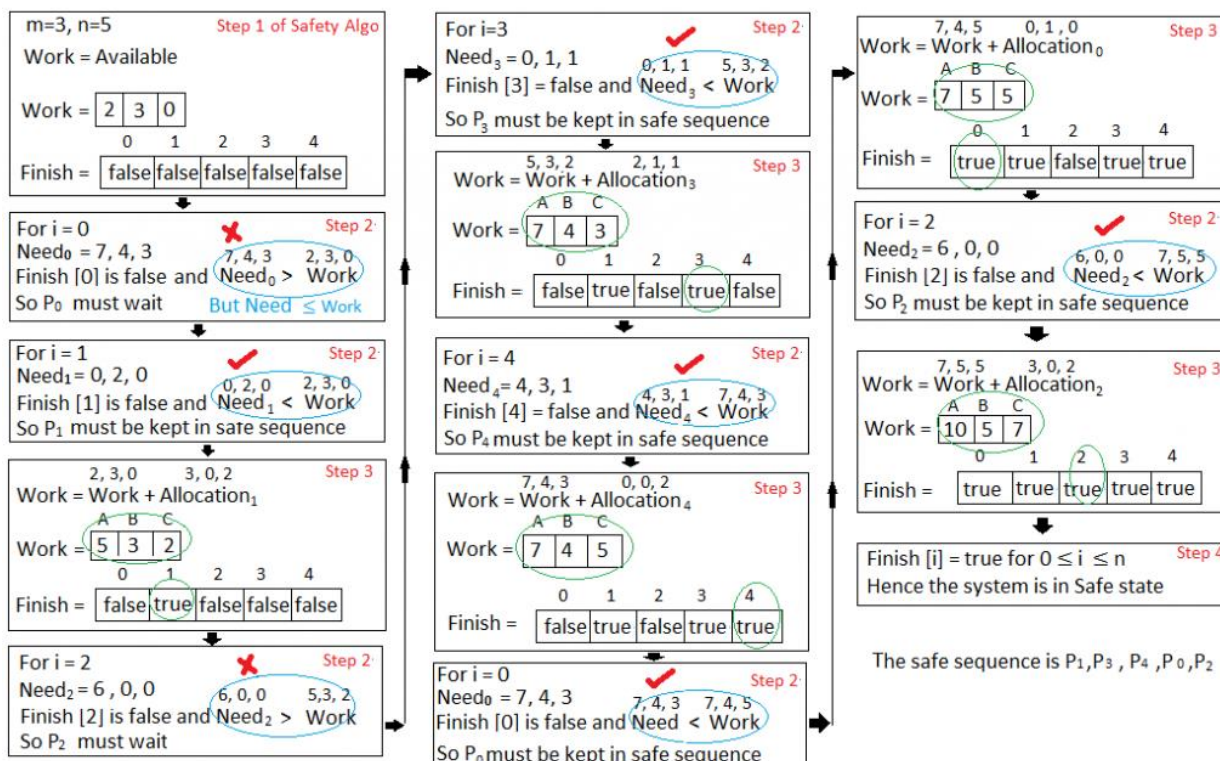
1, 0, 2      3, 3, 2 ✓ Step 2  
Request<sub>1</sub> < Available

Step 3

Available = Available – Request<sub>1</sub>  
Allocation<sub>1</sub> = Allocation<sub>1</sub> + Request<sub>1</sub>  
Need<sub>1</sub> = Need<sub>1</sub> - Request<sub>1</sub>

Process	Allocation	Need	Available
	A B C	A B C	A B C
P <sub>0</sub>	0 1 0	7 4 3	2 3 0
P <sub>1</sub>	3 0 2	0 2 0	
P <sub>2</sub>	3 0 2	6 0 0	
P <sub>3</sub>	2 1 1	0 1 1	
P <sub>4</sub>	0 0 2	4 3 1	

We must determine whether this new system state is safe. To do so, we again execute Safety algorithm on the above data structures.



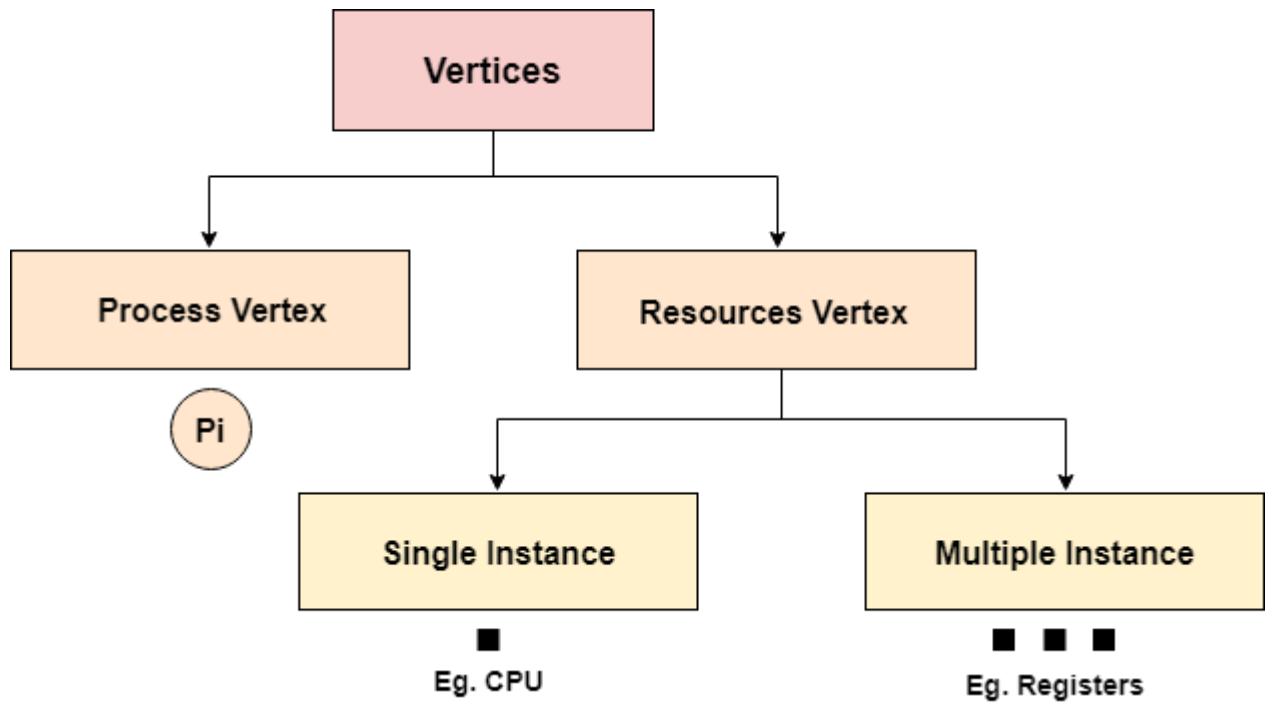
Hence the new system state is safe, so we can immediately grant the request for process  $P_1$ .

## Resource Allocation Graph:-

The resource allocation graph is the pictorial representation of the state of a system. As its name suggests, the resource allocation graph is the complete information about all the processes which are holding some resources or waiting for some resources.

It also contains the information about all the instances of all the resources whether they are available or being used by the processes.

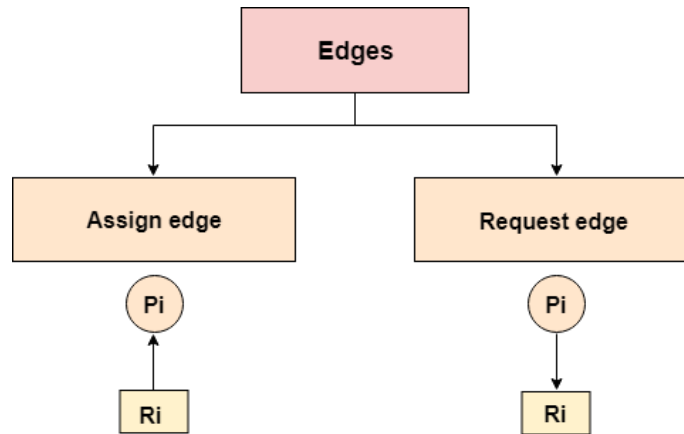
In Resource allocation graph, the process is represented by a Circle while the Resource is represented by a rectangle. Let's see the types of vertices and edges in detail.



Vertices are mainly of two types, Resource and process. Each of them will be represented by a different shape. Circle represents process while rectangle represents resource.

A resource can have more than one instance. Each instance will be represented by a dot inside the rectangle.

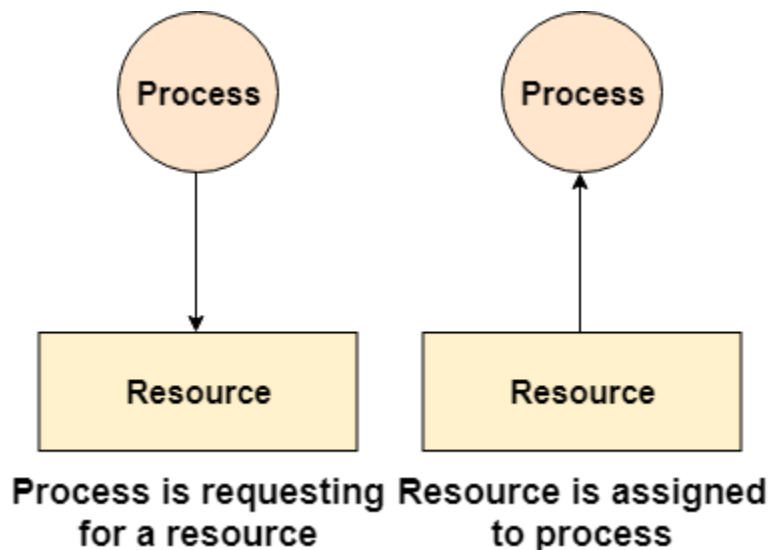




Edges in RAG are also of two types, one represents assignment and other represents the wait of a process for a resource. The above image shows each of them.

A resource is shown as assigned to a process if the tail of the arrow is attached to an instance to the resource and the head is attached to a process.

A process is shown as waiting for a resource if the tail of an arrow is attached to the process while the head is pointing towards the resource.

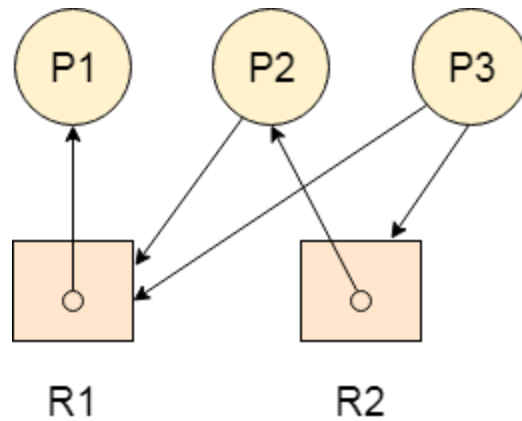


## Example

Let's consider 3 processes P1, P2 and P3, and two types of resources R1 and R2. The resources are having 1 instance each.

According to the graph, R1 is being used by P1, P2 is holding R2 and waiting for R1, P3 is waiting for R1 as well as R2.

The graph is deadlock free since no cycle is being formed in the graph.

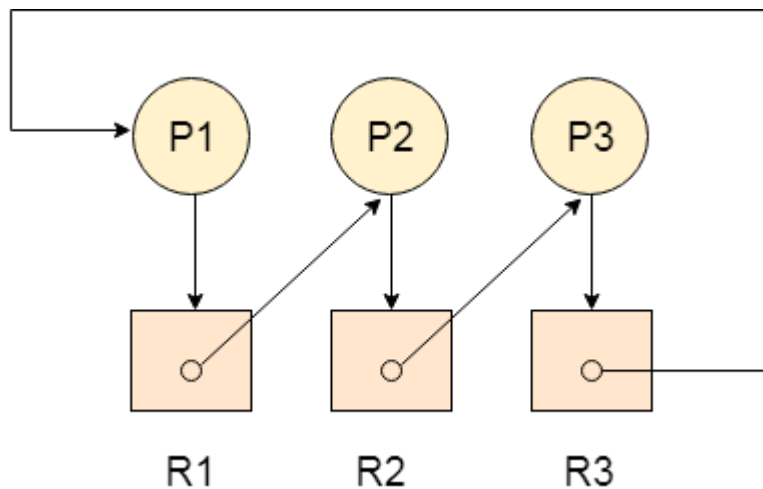


### Deadlock Detection using RAG

If a cycle is being formed in a Resource allocation graph where all the resources have the single instance then the system is deadlocked.

In Case of Resource allocation graph with multi-instanced resource types, Cycle is a necessary condition of deadlock but not the sufficient condition.

The following example contains three processes P1, P2, P3 and three resources R1, R2, R3. All the resources are having single instances each.



If we analyze the graph then we can find out that there is a cycle formed in the graph since the system is satisfying all the four conditions of deadlock.

### Allocation Matrix

Allocation matrix can be formed by using the Resource allocation graph of a system. In Allocation matrix, an entry will be made for each of the resource assigned. For Example, in the following matrix, an entry is being made in front of P1 and below R3 since R3 is assigned to P1.

Process	R1	R2	R3
P1	0	0	1
P2	1	0	0
P3	0	1	0

### Request Matrix

In request matrix, an entry will be made for each of the resource requested. As in the following example, P1 needs R1 therefore an entry is being made in front of P1 and below R1.

Process	R1	R2	R3
P1	1	0	0
P2	0	1	0
P3	0	0	1

**Avial = (0,0,0)**

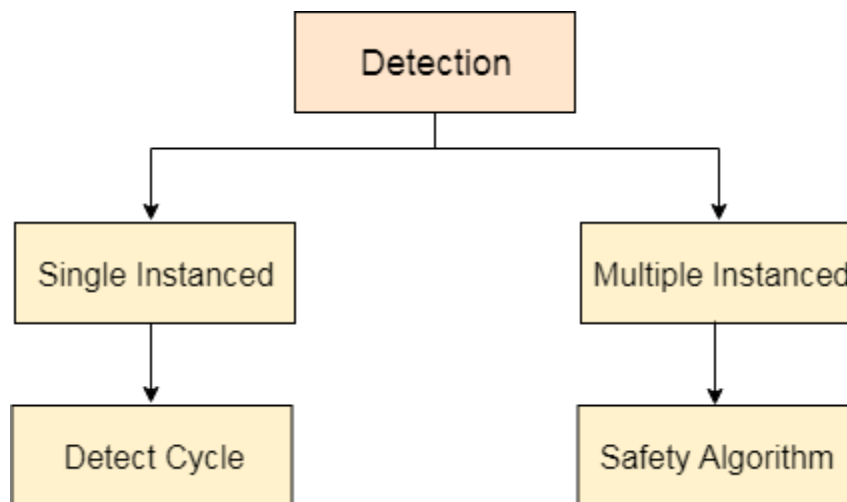
Neither we are having any resource available in the system nor a process going to release. Each of the process needs at least single resource to complete therefore they will continuously be holding each one of them.

We cannot fulfill the demand of at least one process using the available resources therefore the system is deadlocked as determined earlier when we detected a cycle in the graph.

## Deadlock Detection and Recovery

In this approach, The OS doesn't apply any mechanism to avoid or prevent the deadlocks. Therefore the system considers that the deadlock will definitely occur. In order to get rid of deadlocks, The OS periodically checks the system for any deadlock. In case, it finds any of the deadlock then the OS will recover the system using some recovery techniques.

The main task of the OS is detecting the deadlocks. The OS can detect the deadlocks with the help of Resource allocation graph.



In single instanced resource types, if a cycle is being formed in the system then there will definitely be a deadlock. On the other hand, in multiple instanced resource type graph, detecting a cycle is not just enough. We have to apply the safety algorithm on the system by converting the resource allocation graph into the allocation matrix and request matrix.

In order to recover the system from deadlocks, either OS considers resources or processes

## For Resource

### Preempt the resource

We can snatch one of the resources from the owner of the resource (process) and give it to the other process with the expectation that it will complete the execution and will release this resource sooner. Well, choosing a resource which will be snatched is going to be a bit difficult.

### Rollback to a safe state

System passes through various states to get into the deadlock state. The operating system can rollback the system to the previous safe state. For this purpose, OS needs to implement check pointing at every state.

The moment, we get into deadlock, we will rollback all the allocations to get into the previous safe state.

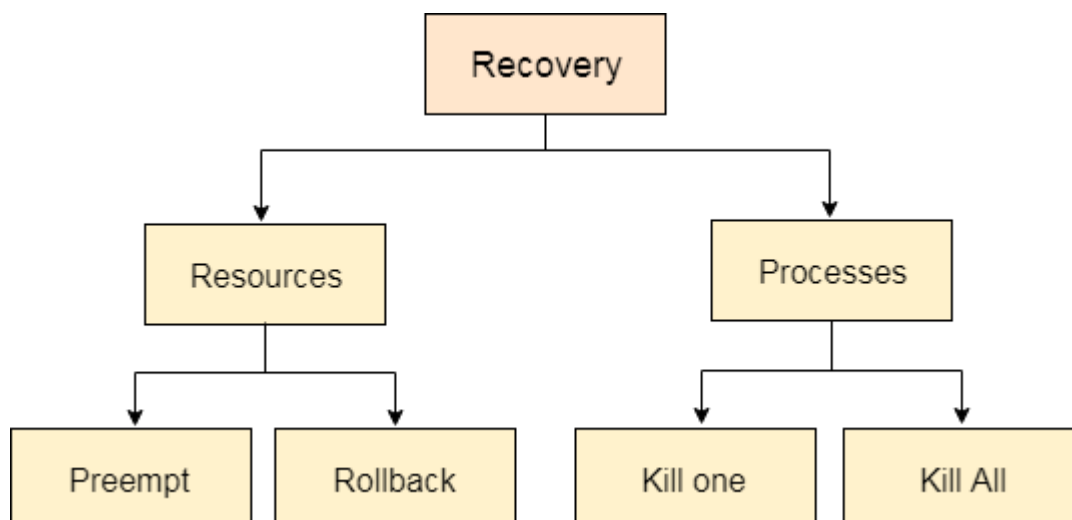
## For Process

### Kill a process

Killing a process can solve our problem but the bigger concern is to decide which process to kill. Generally, Operating system kills a process which has done least amount of work until now.

### Kill all process

This is not a suggestible approach but can be implemented if the problem becomes very serious. Killing all process will lead to inefficiency in the system because all the processes will execute again from starting.



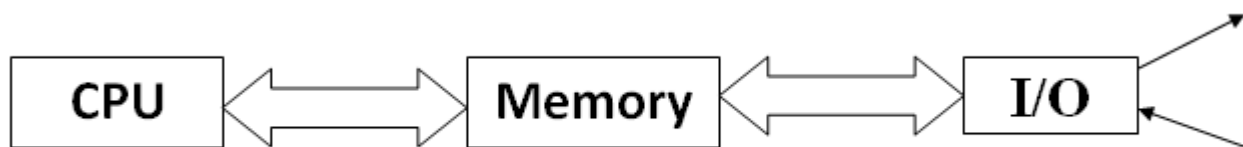
## **Computer Memory:-**

Computer memory can be defined as a collection of some data represented in the binary format. A computer device that is capable to store any information or data temporally or permanently is called storage device.

## **Memory Management:-**

Memory management is the functionality of an operating system which handles or manages primary memory and moves processes back and forth between main memory and disk during execution. Memory management keeps track of each and every memory location, regardless of either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

The memory management method deals with allocation of finite amount of memory to the requesting process. In ready state it is necessary that the process should have access to the certain amount of memory. Memory is a long array of bytes. Each with its own address using read and write statement the CPU and input/output device interact with the memory.



---

One of the main tasks of an operating system is to manage the computer's memory. This includes many responsibilities, including

- Being aware of what parts of the memory are in use and which parts are not.
- Allocating memory to processes when they request it and de-allocating memory when a process releases its memory.
- Moving data from memory to disc, when the physical capacity becomes full, and vice versa.

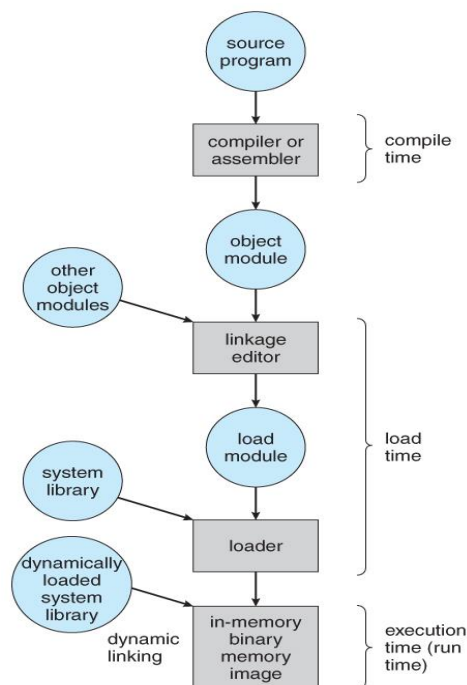
**Address:-** It is two types

1. **Logical address:-** Logical address are generated by the CPU. These address are defined by CPU while writing any program generated by CPU.
2. **Physical address:-** The address of memory area where the user program actually resides is called Physical address.

**Address Binding Mechanism:-** Memory consist of large array of words or bytes. Each with its own address. The processor is to select one of the process from the queue and load into memory as the process is executed. It access data and program from the memory.

User programs typically refer to memory addresses with symbolic names such as "i", "count", and "average Temperature". These symbolic names must be mapped or bound to physical memory addresses, which typically occurs in several stages:

Diagram shows the various stages of the binding processes and the units involved in each stage:



### Multistep processing of a user program

**Compile Time** - If it is known at compile time where a program will reside in physical memory, then absolute code can be generated by the compiler, containing actual physical addresses. However if the load address changes at some later time, then the program will have to be recompiled. DOS .COM programs use compile time binding.

**Load Time** - If the location at which a program will be loaded is not known at compile time, then the compiler must generate relocatable code, which references addresses relative to the start of the program. If that starting address changes, then the program must be reloaded but not recompiled.

**Execution Time** - If a program can be moved around in memory during the course of its execution, then binding must be delayed until execution time. This requires special hardware, and is the method implemented by most modern Operating System .

**Dynamic loading:-** Size of the process depend on the size of physical memory. Hence to obtain the better utilization of the memory space dynamic loading is performed. The major advantage of dynamic loading is that it never load any unused process. This method is useful while handling the large amount of code.

## **Swapping:-**

Swapping is a technique for making memory compact. It is a mechanism that is used to temporarily swap processes out of the main memory to secondary memory, and this makes more memory available for some other processes. At some later time, the system can swap back the process from the secondary memory to the main memory.

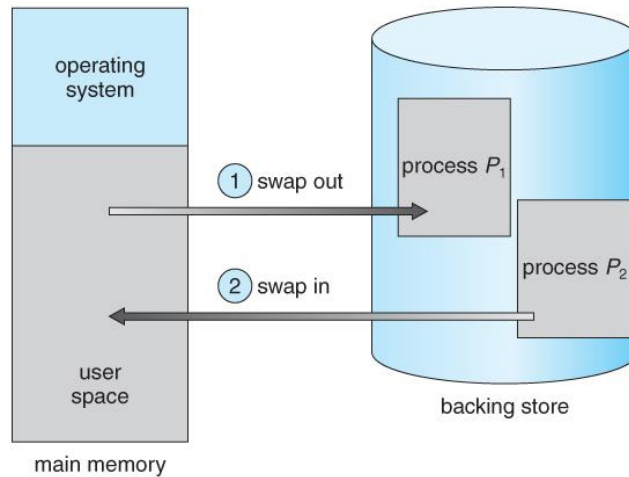
Swapping does affect the performance of the system, but it helps in running multiple processes parallelly. The total time taken by the swapping of a process includes the time it takes to move the entire process to the secondary memory and then again to the main memory.

It is a method of taking out the current content of memory to backstore(disk) and bring the content of backstore to main memory.

There are two operations in swapping method:-

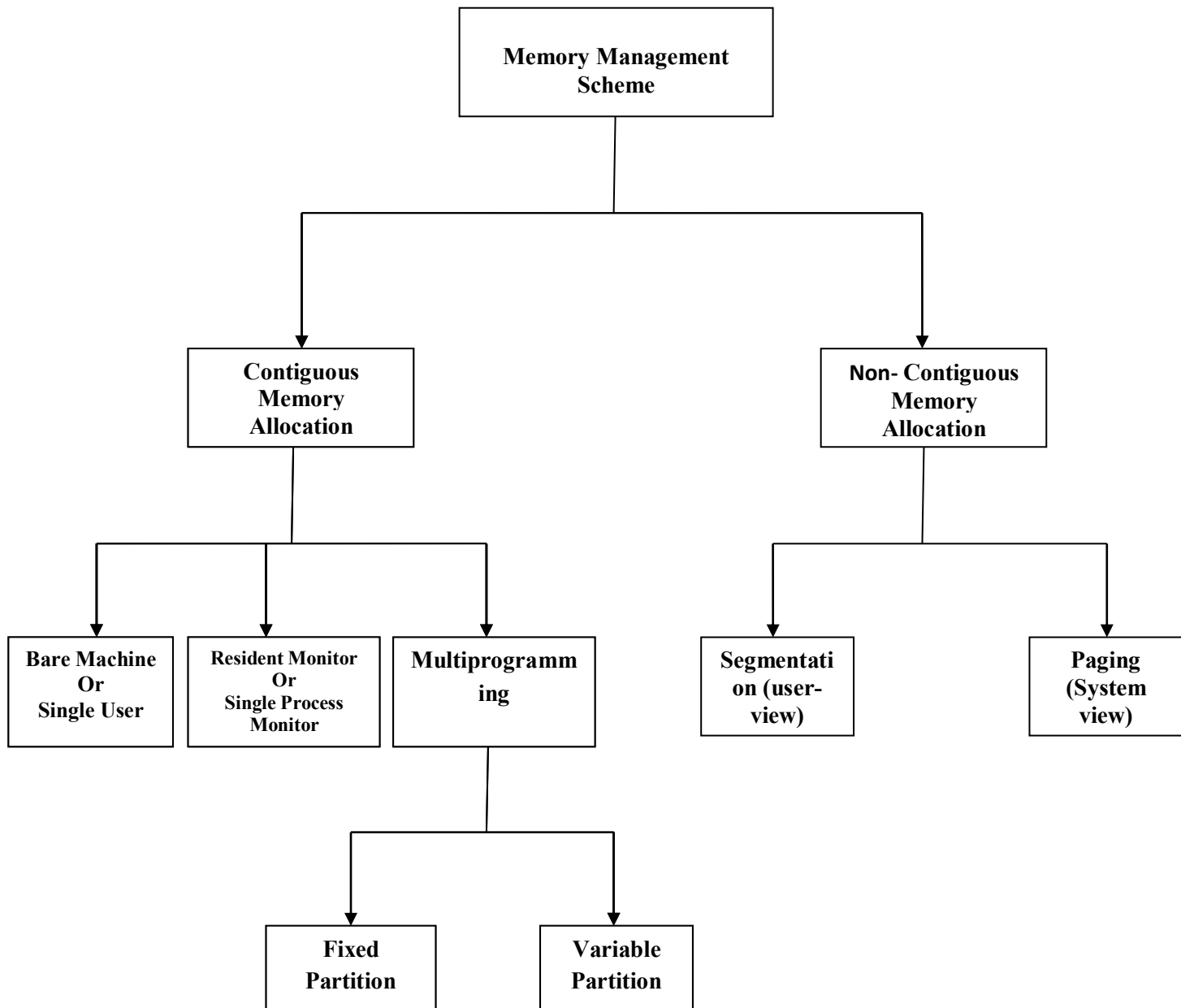
1. Swap out(Read out):- take out to the current data from the main memory.
2. Swap in(Read in):- bring the data of new user into main memory.





Swapping of two processes using a disk as a backing store

### Memory Management Scheme:-



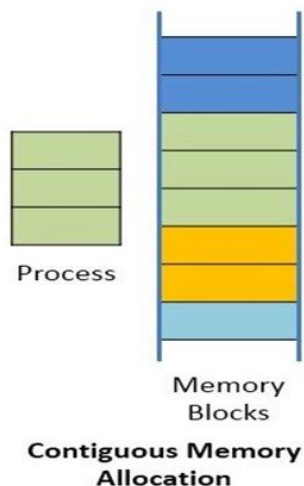
## Contiguous Memory Allocation:-

In **contiguous memory allocation**, all the available memory space remain together in one place. It means freely available memory partitions are not scattered here and there across the whole memory space.

In the **contiguous memory allocation**, both the operating system and the user must reside in the main memory. The main memory is divided into two portions one portion is for the operating and other is for the user program.

In the **contiguous memory allocation** when any user process request for the memory a single section of the contiguous memory block is given to that process according to its need. We can achieve contiguous memory allocation by dividing memory into the fixed-sized partition.

A single process is allocated in that fixed sized single partition. But this will increase the degree of multiprogramming means more than one process in the main memory that bounds the number of fixed partition done in memory. Internal fragmentation increases because of the contiguous memory allocation.

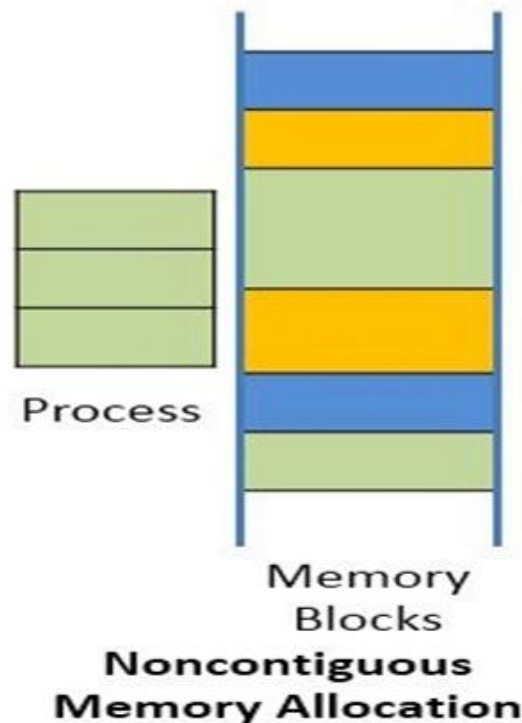


## Non-contiguous memory allocation:-

In the **non-contiguous memory allocation** the available free memory space are scattered here and there and all the free memory space is not at one place. So this is time-consuming.

In the **non-contiguous memory allocation**, a process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement.

This technique of **non-contiguous memory allocation** reduces the wastage of memory which leads to internal and external fragmentation. This utilizes all the free memory space which is created by a different process.



### **Difference between Contiguous and Non-contiguous Memory Allocation :**

S.NO.	CONTIGUOUS MEMORY ALLOCATION	NON-CONTIGUOUS MEMORY ALLOCATION
1.	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2.	Faster in Execution.	Slower in Execution.
3.	It is easier for the OS to control.	It is difficult for the OS to control.
4.	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5.	Internal fragmentation occurs in Contiguous memory allocation method.	External fragmentation occurs in Non-Contiguous memory allocation method.

S.NO.	CONTIGUOUS MEMORY ALLOCATION	NON-CONTIGUOUS MEMORY ALLOCATION
6.	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7.	Wastage of memory is there.	No memory wastage is there.
8.	In contiguous memory allocation, swapped-in processes are arranged in the originally allocated space.	In non-contiguous memory allocation, swapped-in processes can be arranged in any place in the memory.

### **Bare Machine:-**

Bare Machine is logic hardware in the computer system which can execute the programs in the processor without using the Operating System. Till now we have studied that we cannot execute any process inside the processor without the Operating System. But, with the Bare Machine, it is possible.

In the early days, before the Operating systems were developed, the instructions were executed directly on the hardware without any interfering software. But the only drawback was that the Bare Machine accepts the program and instructions in Machine Language. Due to this, only the trained people who were qualified in the computer field and were able to understand and instruct the computer in Machine language were able to operate on a computer. Due to this reason, the Bare Machine was termed as inefficient and cumbersome after the development of different Operating Systems.

It is one of the simplest memory method in which user has complete control over the memory.

The main advantage of bare machine is –

1. Flexible
2. No hardware and software support needed

### **Resident Monitor:-**

The Resident Monitor is a code which runs on Bare Machine. Its acts like an operating system which controls everything inside a processor and performs all the functions. The Resident Monitor is thus also known as the Job Sequencer because like the Operating system, it also sequences the jobs and sends it to

the processor for execution. After the jobs are scheduled, the Resident Monitor loads the Programs one by one into the main memory according to their sequence. The advantage of using a Resident Monitor over an Operating System is that there is no gap or lag between the program executions. So, the processing is faster in the Resident Monitors.

The Resident Monitors are divided into 3 parts:

1. **Control Language Interpreter**

The job of the Control Language Interpreter is to read and carry out the instructions line by line to the next level.

2. **Loader**

The Loader is the main part of the Resident Monitor. As the name suggests, it Loads all the required system and application programs into the main memory.

3. **Device Driver**

The Device Driver Takes care of all the Input-Output devices connected with the system. So, all the communication that takes place between the user and the system is handled by the Device Driver. It simply acts as an intermediate between the requests and the response, requests that are made by the user to the system, and they respond that the system produces to fulfill these requests.

## **Multiprogramming:-**

In the multiprogramming, the multiple users can share the memory simultaneously. By multiprogramming we mean there will be more than one process in the main memory and if the running process wants to wait for an event like I/O then instead of sitting idle CPU will make a context switch and will pick another process.

Multiprogramming are two types:-

1. Fixed (Static)
2. Variable (Dynamic)

**Fixed sized partition (Static) :-** In the fixed sized partition the system divides memory into fixed size partition (may or may not be of the same size) here entire partition is allowed to a process and if there is

some wastage inside the partition is allocated to a process and if there is some wastage inside the partition then it is called internal fragmentation.

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. The memory is assigned to the processes in contiguous way.

In fixed partitioning,

1. The partitions cannot overlap.
2. A process must be contiguously present in a partition for the execution.

There are various cons of using this technique.

### **1. Internal Fragmentation**

If the size of the process is lesser than 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.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

### **2. 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.

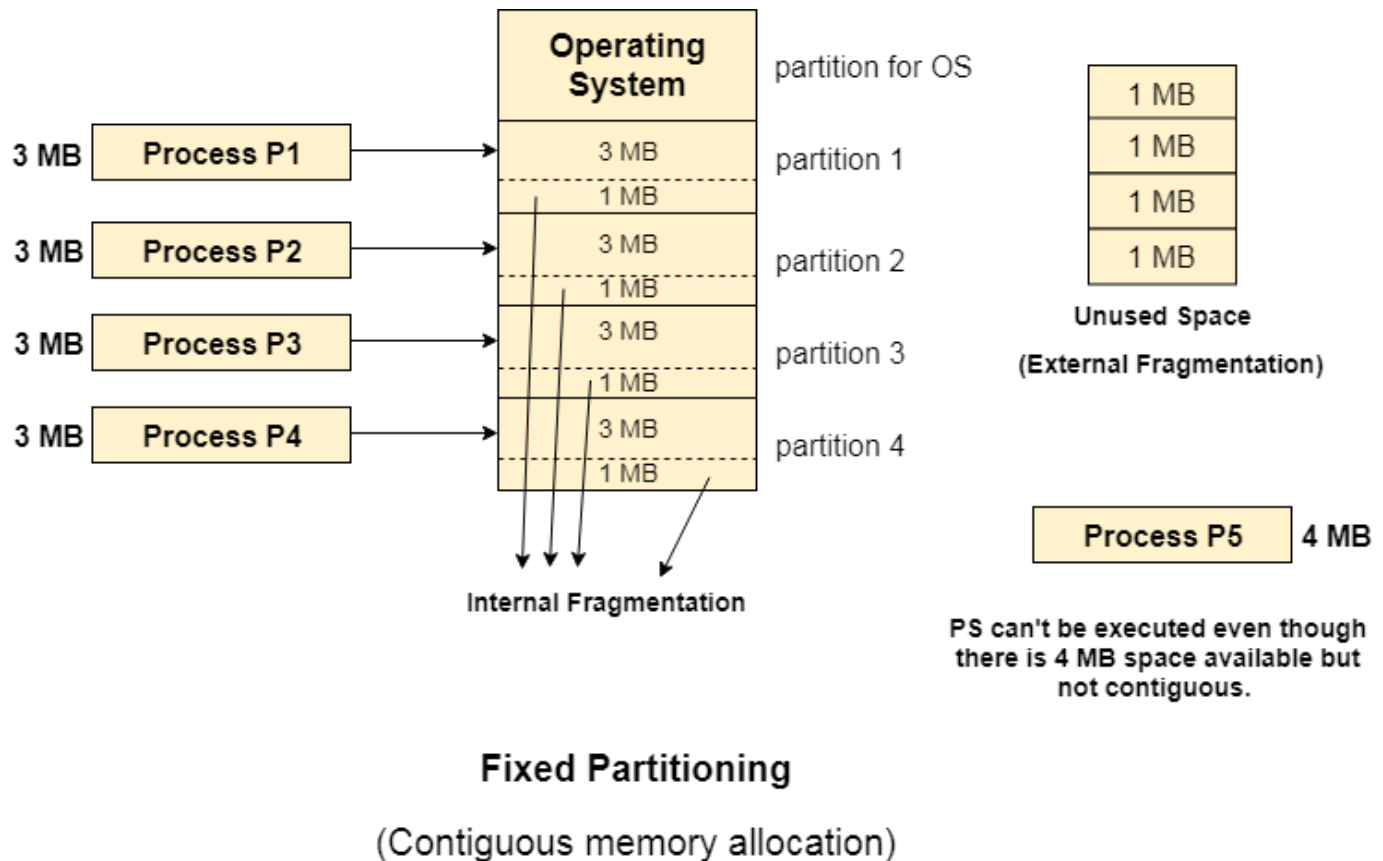
As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

### **3. Limitation on the size of the process**

If the process size is larger than the size of maximum sized partition then that process cannot be loaded into the memory. Therefore, a limitation can be imposed on the process size that is it cannot be larger than the size of the largest partition.

#### 4. Degree of multiprogramming is less

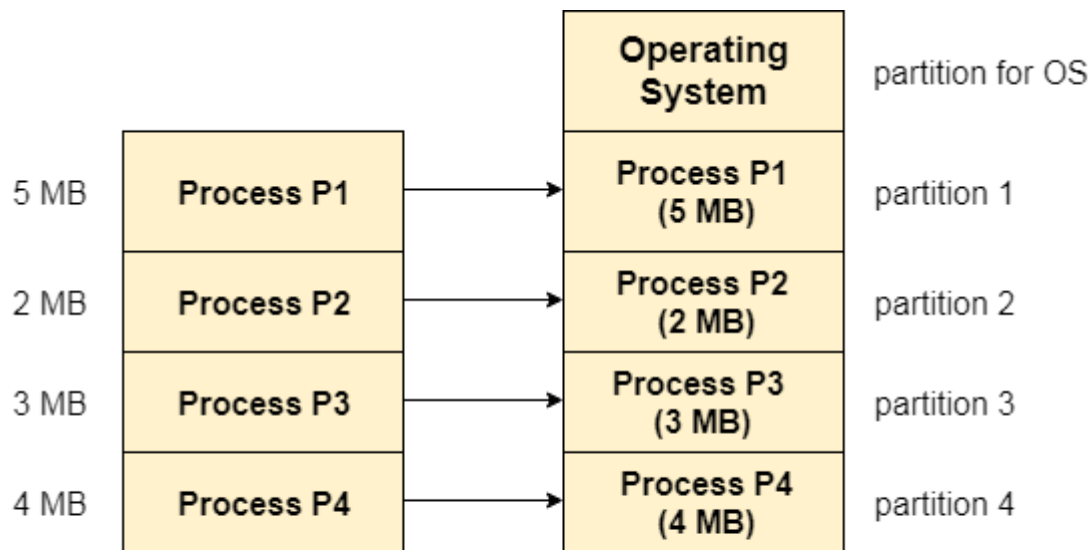
By Degree of multi programming, we simply mean the maximum number of processes that can be loaded into the memory at the same time. In fixed partitioning, the degree of multiprogramming is fixed and very less due to the fact that the size of the partition cannot be varied according to the size of processes.



#### Variable (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.



## Dynamic Partitioning

(Process Size = Partition Size)

### Advantages of Dynamic Partitioning over fixed partitioning

#### 1. No Internal Fragmentation

Given the fact that the partitions in dynamic partitioning are created according to the need of the process, It is clear that there will not be any internal fragmentation because there will not be any unused remaining space in the partition.

#### 2. No Limitation on the size of the process

In Fixed partitioning, the process with the size greater than the size of the largest partition could not be executed due to the lack of sufficient contiguous memory. Here, In Dynamic partitioning, the process size can't be restricted since the partition size is decided according to the process size.

#### 3. 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.

### Disadvantages of dynamic partitioning



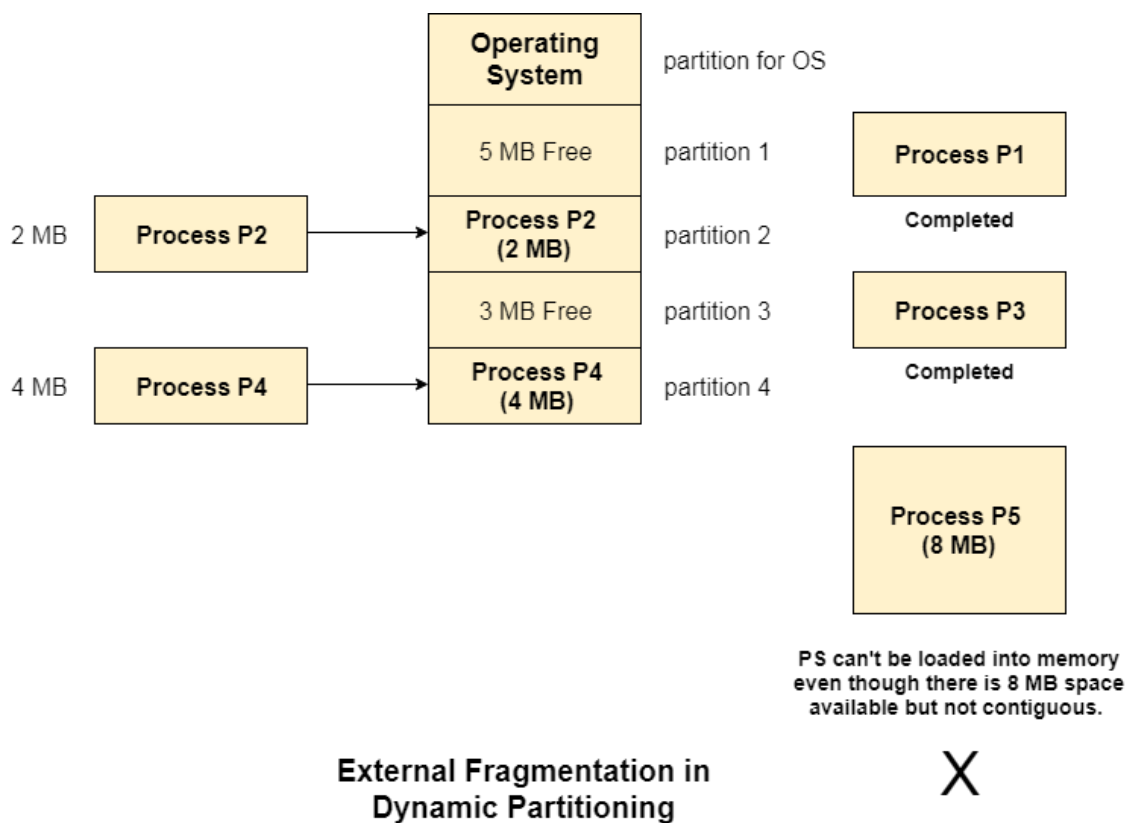
## External Fragmentation

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

Let's consider three processes P1 (1 MB) and P2 (3 MB) and P3 (1 MB) are being loaded in the respective partitions of the main memory.

After some time P1 and P3 got completed and their assigned space is freed. Now there are two unused partitions (1 MB and 1 MB) available in the main memory but they cannot be used to load a 2 MB process in the memory since they are not contiguously located.

The rule says that the process must be contiguously present in the main memory to get executed. We need to change this rule to avoid external fragmentation.



## Complex Memory Allocation

In Fixed partitioning, the list of partitions is made once and will never change but in dynamic partitioning, the allocation and deallocation is very complex since the partition size will be varied every time when it is assigned to a new process. OS has to keep track of all the partitions.

Due to the fact that the allocation and deallocation are done very frequently in dynamic memory allocation and the partition size will be changed at each time, it is going to be very difficult for OS to manage everything.

### Algorithm use for selection of a Free area of a memory:-

1. **First Fit:**

The first hole that is big enough is allocated to program.

2. **Best Fit:**

The smallest hole that is big enough is allocated to program.

3. **Worst Fit:**

The largest hole that is big enough is allocated to program.

**Q1. For the Partition of 200k, 500k, 300k, 600k (In order) place the process of 212k, 417k, 112k, 426k (In order) now fit the value according to the Best Fit algorithm.**

**Solution:-**

200k	500k	300k	600k
------	------	------	------

P1= 212k,

P2= 417k,

P3= 112k,

P4= 426k

	OS
200k	P3
500k	P2
300k	P1
600k	P4

		Fragment
		Size
P1	→ 212k → 300k	88k
P2	→ 417k → 500k	83k
P3	→ 112k → 200k	88k
P4	→ 426k → 600k	174k
Unused Space =		433k

**Q2. For the Partition of 100k, 500k, 200k, 300k, 600k (In order) place the process of 212k, 417k, 112k, 426k (In order) now fit the value according to the First Fit algorithm.**

**Solution:-**

212k	417k	112k	426k
------	------	------	------

P1      P2      P3      P4

	OS	
100k		
500k	P1 = 212k	P4=
200k	P3 = 112k	426k
300k		Wait
600k	P2 = 417k	

	Allocation Partitions	Fragment Size
P1 = 212k	500k	288k
P2 = 417k	600k	183k
P3 = 112K	200k	88k
P4 = 426k	Wait	Wait
	Unused Space	559k

**Q3. For the Partition of 100k, 500k, 200k, 300k, 600k (In order) place the process of 212k, 417k, 112k, 426k (In order) now fit the value according to the Worst Fit algorithm.**

**Solution:-**

OS	100k	500k	200k	300k	600k
----	------	------	------	------	------

**Worst Fit:-**

	100k	
<b>83k</b>	500k	<b>P2</b>
	200k	<b>P4 Wait</b>
<b>288k</b>	300k	<b>P3</b>
<b>388k</b>	600k	<b>P4</b>

$$\text{Unused Space} = 388 + 288 + 83$$

$$= 759k$$

**Memory Fragmentation:-** It is a drawback of memory management method.

### 1. Internal Fragmentation

If the size of the process is lesser than 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.

As shown in the image below, the 4 MB partition is used to load only 3 MB process and the remaining 1 MB got wasted.

### 2. 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.

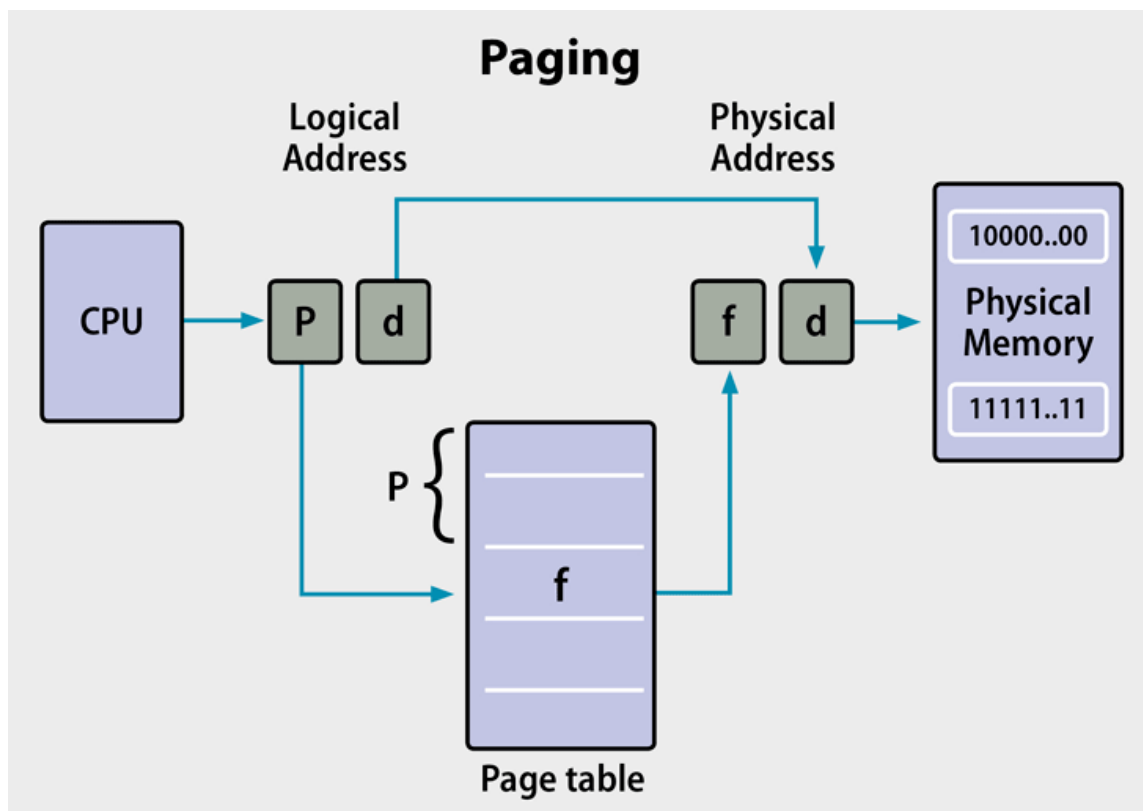
As shown in the image below, the remaining 1 MB space of each partition cannot be used as a unit to store a 4 MB process. Despite of the fact that the sufficient space is available to load the process, process will not be loaded.

**Paging** :-As mentioned above, the memory management function called *paging* specifies storage locations to the CPU as additional memory, called virtual memory. The CPU cannot directly access storage disk, so the MMU emulates memory by mapping pages to frames that are in RAM.

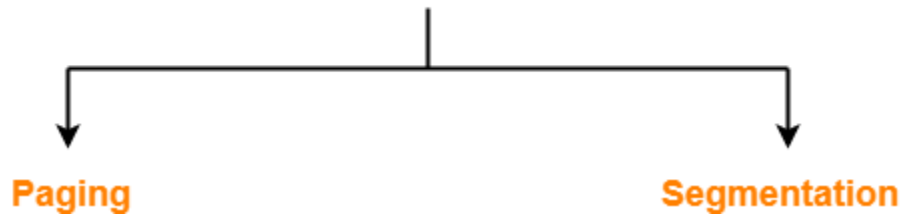
Before we launch into a more detailed explanation of pages and frames, let's define some technical terms.

- **Page:** A fixed-length contiguous block of virtual memory residing on disk.

- **Frame:** A fixed-length contiguous block located in RAM; whose sizing is identical to pages.
- **Physical memory:** The computer's random access memory (RAM), typically contained in DIMM cards attached to the computer's motherboard.
- **Virtual memory:** Virtual memory is a portion of an HDD or SSD that is reserved to emulate RAM. The MMU serves up virtual memory from disk to the CPU to reduce the workload on physical memory.
- **Virtual address:** The CPU generates a virtual address for each active process. The MMU maps the virtual address to a physical location in RAM and passes the address to the bus. A virtual address space is the range of virtual addresses under CPU control.
- **Physical address:** The physical address is a location in RAM. The physical address space is the set of all physical addresses corresponding to the CPU's virtual addresses. A physical address space is the range of physical addresses under MMU control.



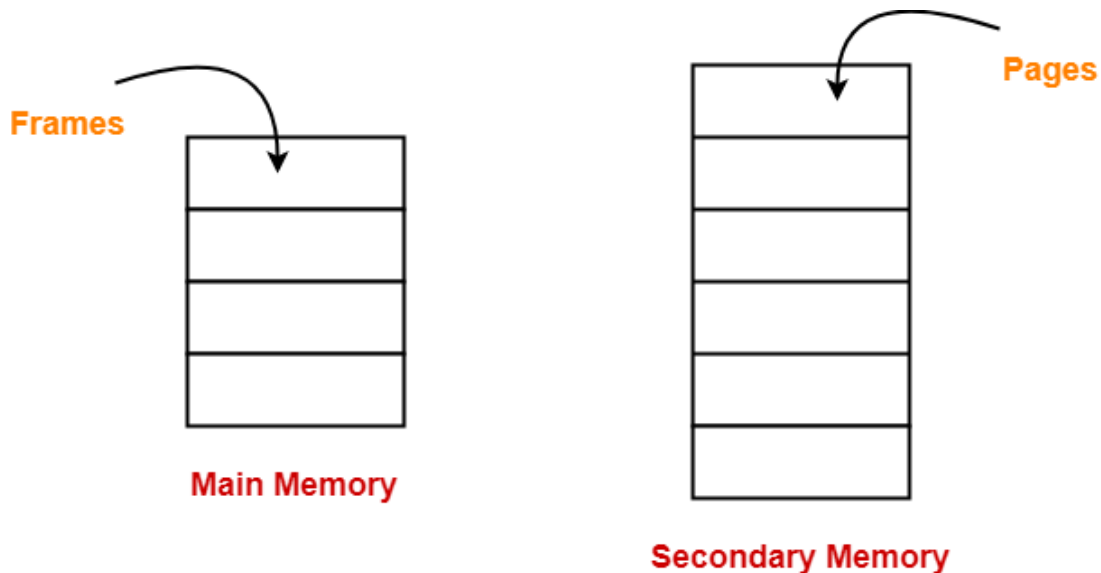
## Non-Contiguous Memory Allocation Techniques



### Paging:-

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contiguous. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

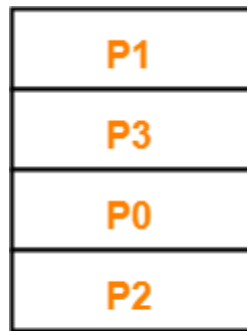
- Paging is a fixed size partitioning scheme.
- In paging, secondary memory and main memory are divided into equal fixed size partitions.
- The partitions of secondary memory are called as **pages**.
- The partitions of main memory are called as **frames**.



- Each process is divided into parts where size of each part is same as page size.
- The size of the last part may be less than the page size.
- The pages of process are stored in the frames of main memory depending upon their availability.

### **Example-**

- Consider a process is divided into 4 pages  $P_0$ ,  $P_1$ ,  $P_2$  and  $P_3$ .
- Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-



**Main Memory**

### **Translating Logical Address into Physical Address-**

- CPU always generates a logical address.
- A physical address is needed to access the main memory.

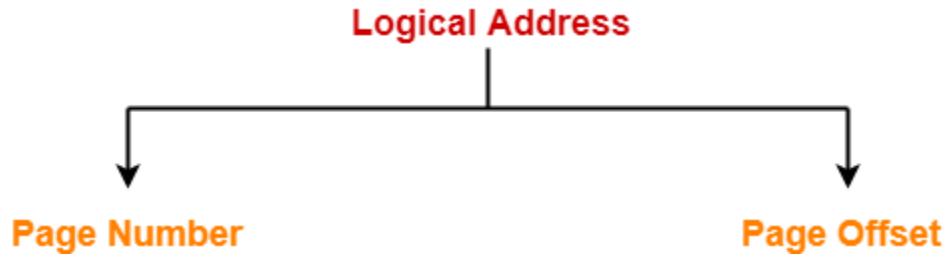
Following steps are followed to translate logical address into physical address-

#### **Step-01:**

CPU generates a logical address consisting of two parts-

1. Page Number
2. Page Offset





- Page Number specifies the specific page of the process from which CPU wants to read the data.
- Page Offset specifies the specific word on the page that CPU wants to read.

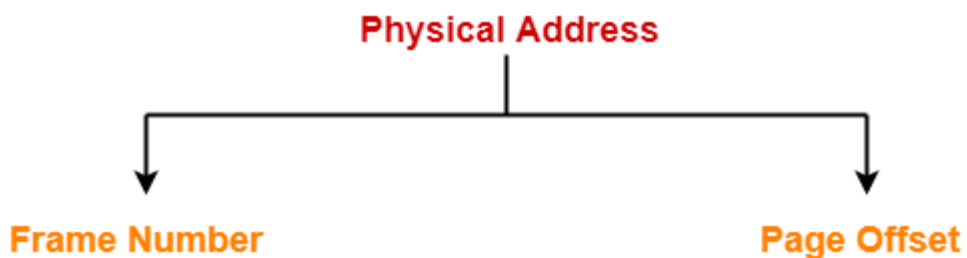
### Step-02:

For the page number generated by the CPU,

- **Page Table** provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.

### Step-03:

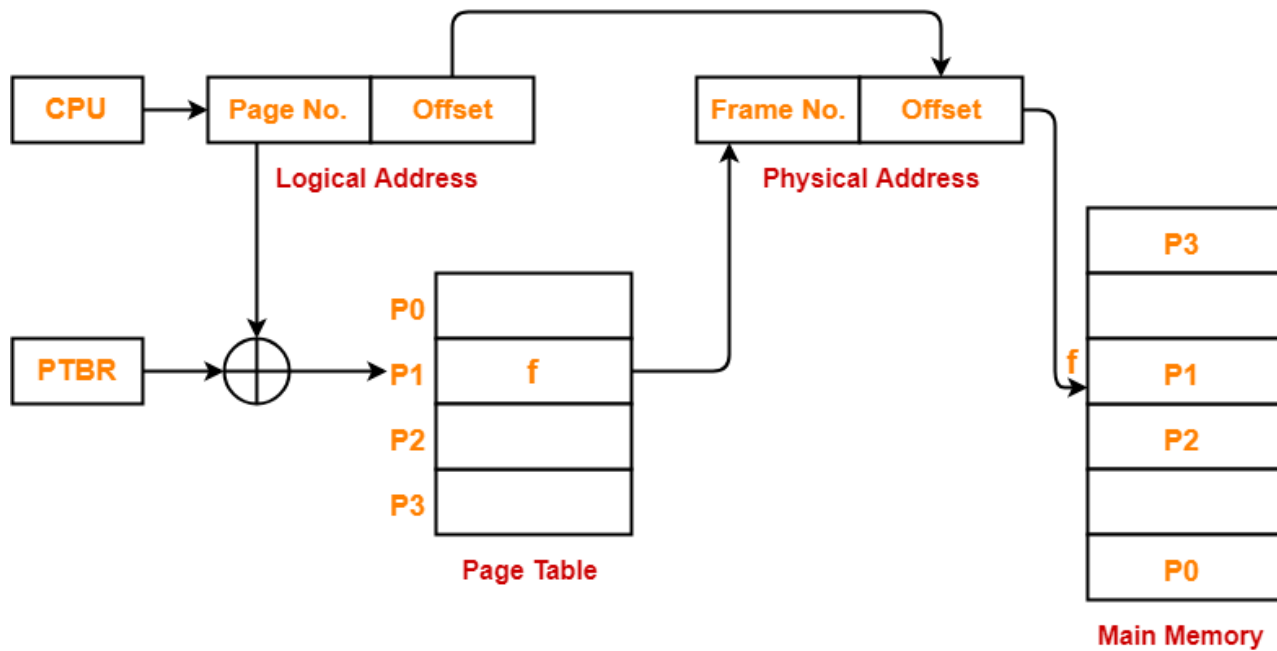
- The frame number combined with the page offset forms the required physical address.



- Frame number specifies the specific frame where the required page is stored.
- Page Offset specifies the specific word that has to be read from that page.

### Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



**Translating Logical Address into Physical Address**

### Advantages-

The advantages of paging are-

- It allows to store parts of a single process in a non-contiguous fashion.
- It solves the problem of external fragmentation.

### Disadvantages-

The disadvantages of paging are-

- It suffers from internal fragmentation.
- There is an overhead of maintaining a page table for each process.
- The times taken to fetch the instruction increases since now two memory accesses are required.

### **Q- Why is page size always power of 2?**

**Solution:-** Recall that paging is implemented by breaking up an address into a page and offset number. It is most efficient to break the address into X page bits and Y offset bits, rather than perform arithmetic on the address to calculate the page number and offset. Because each bit position represents a power of 2, splitting an address between bits results in a page size that is a power of 2.

### **Q- Consider a logical address of eight pages of 1024 words each mapped on to physical memory of 32 frames then find out**

**1. How many bits in the logical address?**

**2. How many bits in the physical address?**

#### **Solution 1:-**

Let No of bits in Physical memory is m

Then size of physical memory is  $2^m$

No of pages = 8 =  $2^3$

No of frames = 32 =  $2^5$

Size of each frame = Size of each page =  $2^{10}$

No of Frame =  $\frac{\text{Physical memory}}{\text{Size of page}}$

Frame(f) =  $\frac{m}{p}$

$$2^5 = \frac{2^m}{2^{10}}$$

$$2^m = 2^5 \times 2^{10}$$

$$m = 15$$

m is Physical address

#### **Solution 2:-**

Let No of bits in logical memory bits is n

Then size of logical memory is  $2^n$

No of Pages =  $\frac{\text{Size of logical memory}}{\text{Size of each page}}$

$$2^3 = \frac{2^n}{2^{10}}$$

n = 13

n = logical address

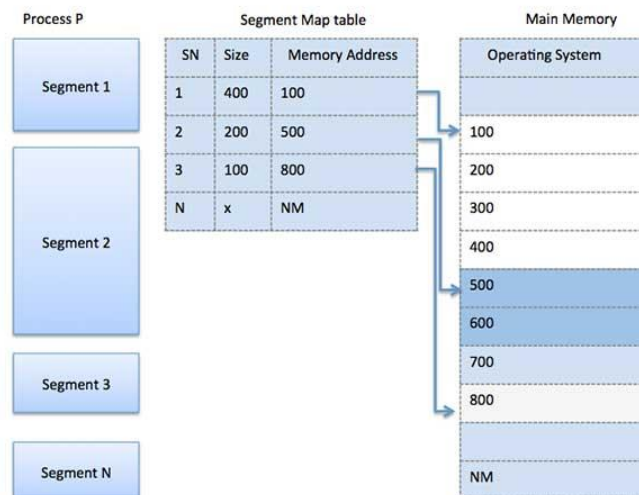
## Segmentation:-

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



- Like Paging, Segmentation is another non-contiguous memory allocation technique.

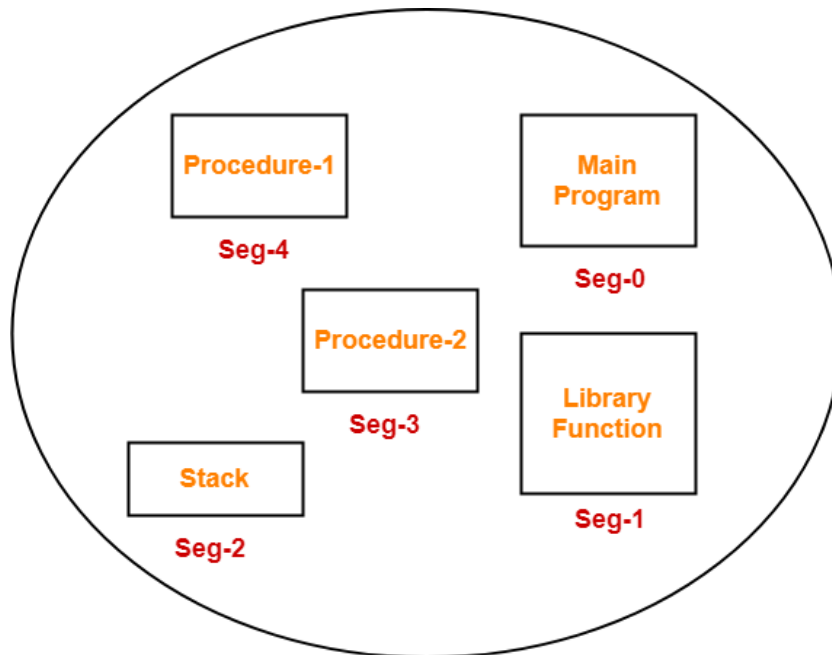
- In segmentation, process is not divided blindly into fixed size pages.
- Rather, the process is divided into modules for better visualization.

### Characteristics-

- Segmentation is a variable size partitioning scheme.
- In segmentation, secondary memory and main memory are divided into partitions of unequal size.
- The size of partitions depend on the length of modules.
- The partitions of secondary memory are called as **segments**.

### Example-

Consider a program is divided into 5 segments as-



### Segment Table-

- Segment table is a table that stores the information about each segment of the process.
- It has two columns.
- First column stores the size or length of the segment.
- Second column stores the base address or starting address of the segment in the main memory.
- Segment table is stored as a separate segment in the main memory.
- Segment table base register (STBR) stores the base address of the segment table.

For the above illustration, consider the segment table is-

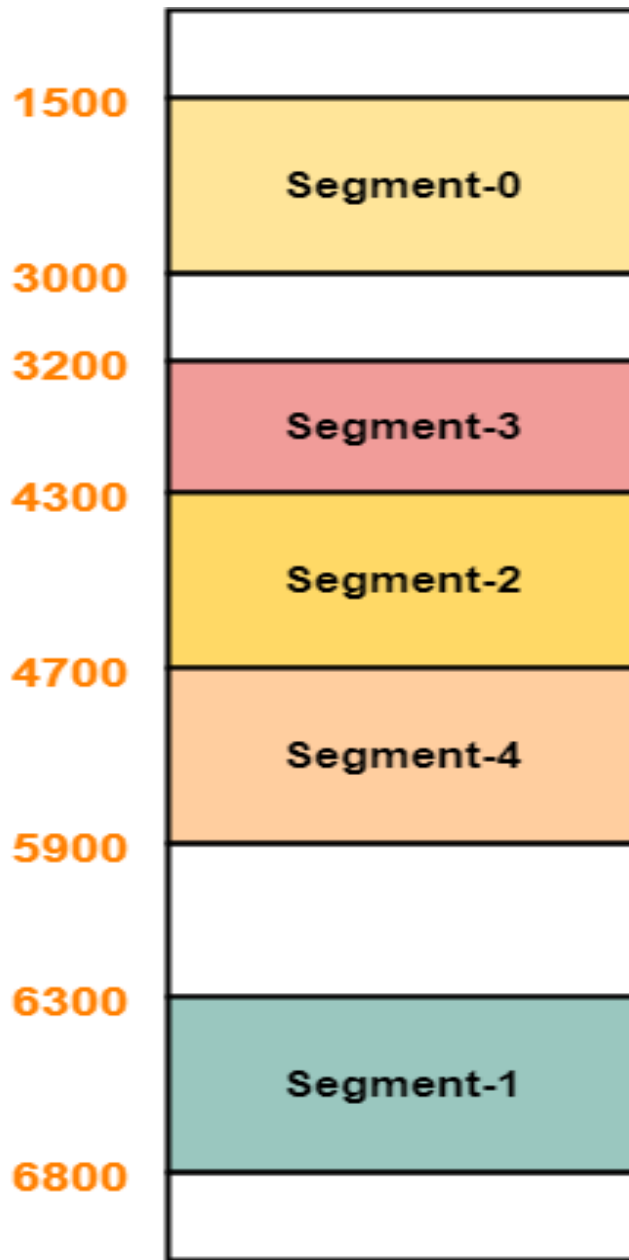
	<b>Limit</b>	<b>Base</b>
<b>Seg-0</b>	<b>1500</b>	<b>1500</b>
<b>Seg-1</b>	<b>500</b>	<b>6300</b>
<b>Seg-2</b>	<b>400</b>	<b>4300</b>
<b>Seg-3</b>	<b>1100</b>	<b>3200</b>
<b>Seg-4</b>	<b>1200</b>	<b>4700</b>

**Segment Table**

Here,

- Limit indicates the length or size of the segment.
- Base indicates the base address or starting address of the segment in the main memory.

In accordance to the above segment table, the segments are stored in the main memory as-



## **Main Memory**

### **Translating Logical Address into Physical Address-**

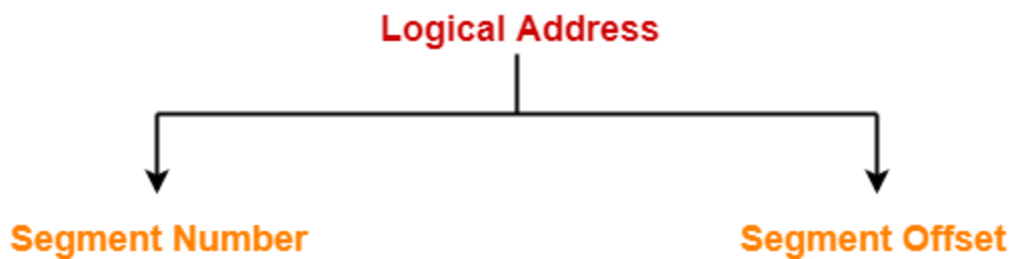
- CPU always generates a logical address.
- A physical address is needed to access the main memory.

Following steps are followed to translate logical address into physical address-

Step-01:

CPU generates a logical address consisting of two parts-

1. Segment Number
2. Segment Offset



- Segment Number specifies the specific segment of the process from which CPU wants to read the data.
- Segment Offset specifies the specific word in the segment that CPU wants to read.

Step-02:

- For the generated segment number, corresponding entry is located in the segment table.
- Then, segment offset is compared with the limit (size) of the segment.

Now, two cases are possible-

Case-01: Segment Offset  $\geq$  Limit

- If segment offset is found to be greater than or equal to the limit, a trap is generated.

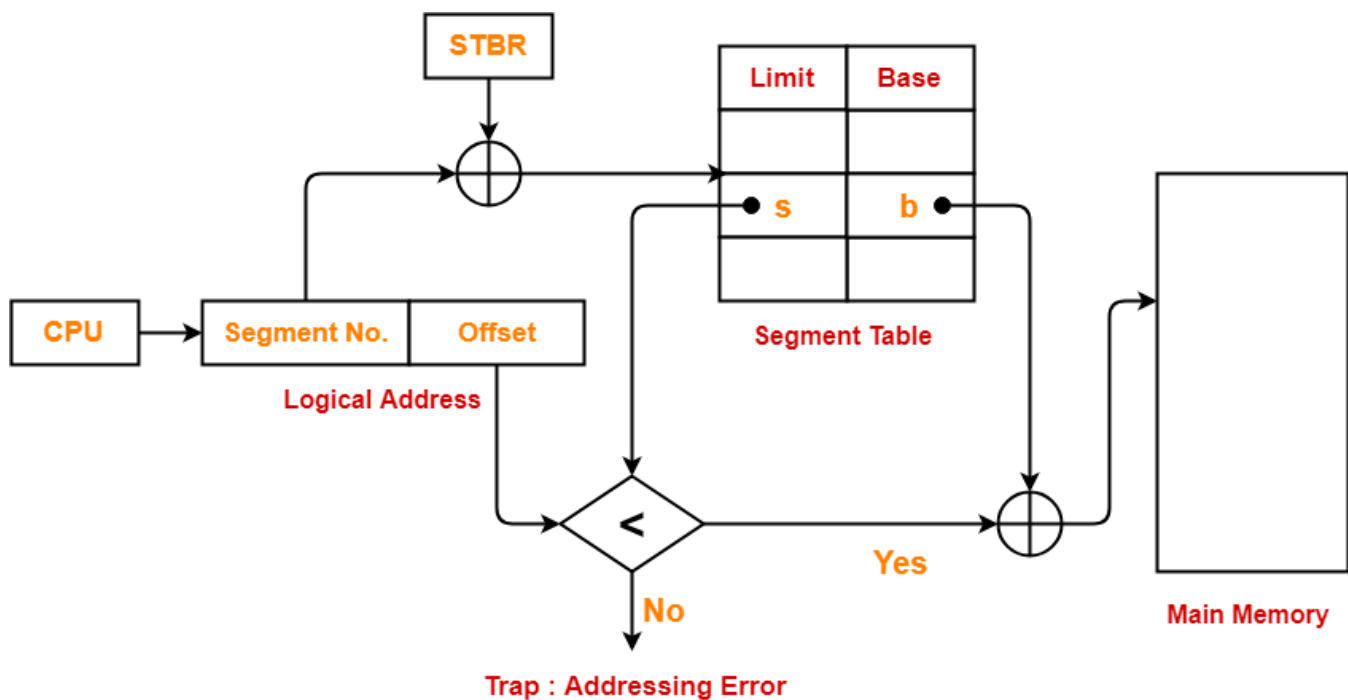


### Case-02: Segment Offset < Limit

- If segment offset is found to be smaller than the limit, then request is treated as a valid request.
- The segment offset must always lie in the range  $[0, \text{limit}-1]$ ,
- Then, segment offset is added with the base address of the segment.
- The result obtained after addition is the address of the memory location storing the required word.

### Diagram-

The following diagram illustrates the above steps of translating logical address into physical address-



**Translating Logical Address into Physical Address**

## **Advantages-**

The advantages of segmentation are-

- It allows to divide the program into modules which provides better visualization.
- Segment table consumes less space as compared to **Page Table** in paging.
- It solves the problem of internal fragmentation.

## **Disadvantages-**

The Dis-advantages of segmentation are-

- There is an overhead of maintaining a segment table for each process.
- The time taken to fetch the instruction increases since now two memory accesses are required.
- Segments of unequal size are not suited for swapping.
- It suffers from external fragmentation as the free space gets broken down into smaller pieces with the processes being loaded and removed from the main memory.

Paging VS Segmentation

<b>S.r. No.</b>	<b>Paging</b>	<b>Segmentation</b>
1	Non-Contiguous memory allocation	Non-contiguous memory allocation
2	Paging divides program into fixed size pages.	Segmentation divides program into variable size segments.
3	OS is responsible	Compiler is responsible.
4	Paging is faster than segmentation	Segmentation is slower than paging
5	Paging is closer to Operating System	Segmentation is closer to User

6	It suffers from internal fragmentation	It suffers from external fragmentation
7	There is no external fragmentation	There is no external fragmentation
8	Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
9	Page table is used to maintain the page information.	Segment Table maintains the segment information
10	Page table entry has the frame number and some flag bits to represent details about pages.	Segment table entry has the base address of the segment and some protection bits for the segments.

**Q1. 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 address of following logical address**

1. 0430,
2. 110
3. 2500
4. 3400
5. 4112

**Solution 1 :-** In 0430 first digit 0 refer the segment value and 430 refer the offset value

So, Physical address= Base + Offset

$$= 219 + 430$$

$$= 649$$

**Solution 2 :-** In 110 first digit 1 refer the segment value and 10 refer the offset value

96 < 112

So, Physical address = Base + Offset

$$= 2300 + 10$$

$$= 2310$$

**Solution 3 :-** In 2500 first digit 2 refer the segment value and 500 refer the offset value

So, Physical address = Base + Offset

$$= 90 + 500$$

$$= 590$$

**Solution 4 :-** In 3400 first digit 3 refer the segment value and 400 refer the offset value

So, Physical address = Base + Offset

$$= 1327 + 400$$

$$= 1727$$

**Solution 5 :-** In 4112 first digit 4 refer the segment value and 112 refer the offset value

So, Physical address = Base + Offset

$$= 1952 + 112$$

$$= 2064$$

### **Paged Segmentation:-**

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

- Segmented Paging is a scheme that implements the combination of **Segmentation** and **Paging**.
- First, segmentation divides the process into segments.
- Then, paging divides each segment into pages.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

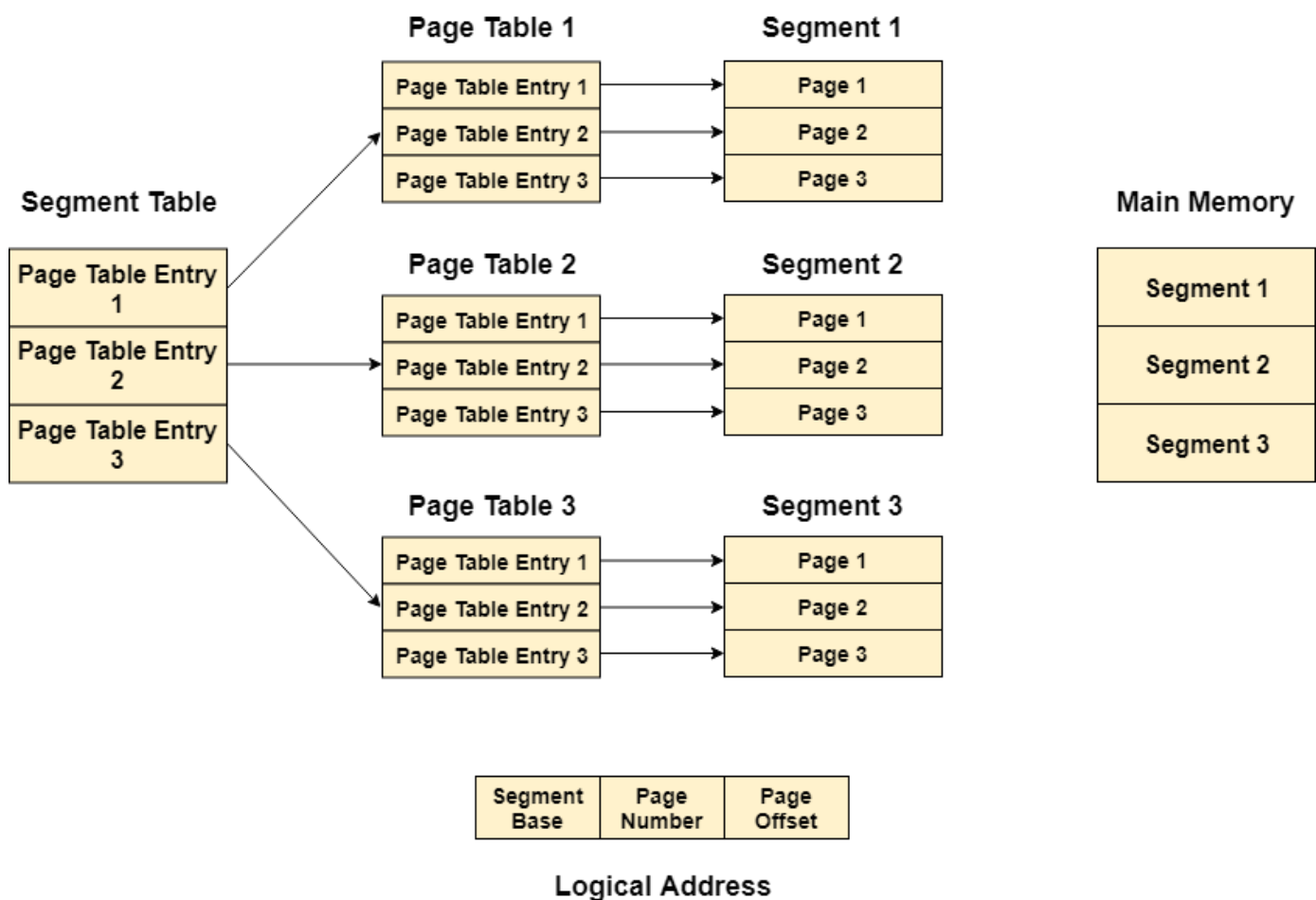
1. Pages are smaller than segments.
2. Each Segment has a page table which means every program has multiple page tables.
3. The logical address is represented as Segment Number (base address), Page number and page offset.

**Segment Number** → It points to the appropriate Segment Number.

**Page Number** → It Points to the exact page within the segment

**Page Offset** → Used as an offset within the page frame

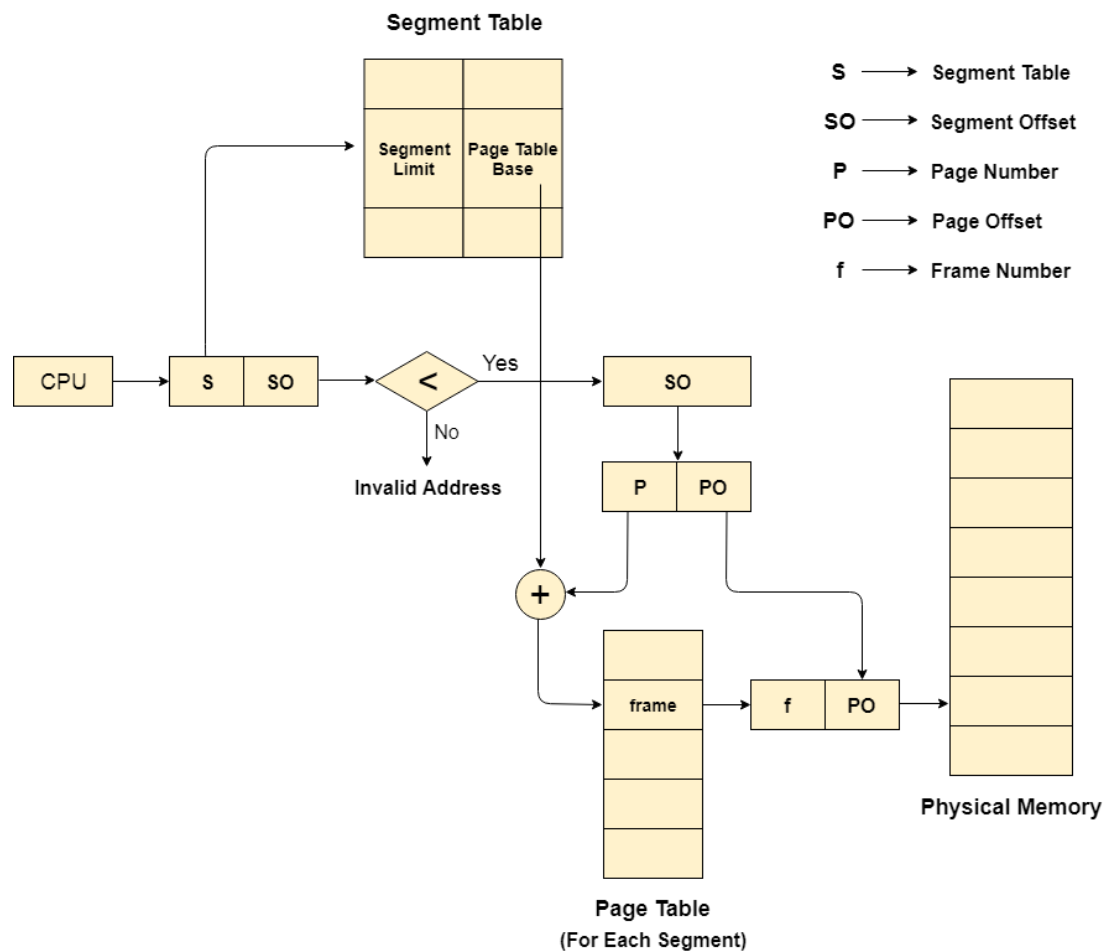
Each Page table contains the various information about every page of the segment. The Segment Table contains the information about every segment. Each segment table entry points to a page table entry and every page table entry is mapped to one of the page within a segment.



### Translation of logical address to physical address:-

The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.

The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.



### Advantages of Segmented Paging

1. It reduces memory usage.
2. Page table size is limited by the segment size.
3. Segment table has only one entry corresponding to one actual segment.
4. External Fragmentation is not there.
5. It simplifies memory allocation.

## **Disadvantages of Segmented Paging**

1. Internal Fragmentation will be there.
2. The complexity level will be much higher as compare to paging.
3. Page Tables need to be contiguously stored in the memory.

### **Problem-01:**

A certain computer system has the segmented paging architecture for virtual memory. The memory is byte addressable. Both virtual and physical address spaces contain  $2^{16}$  bytes each. The virtual address space is divided into 8 non-overlapping equal size segments. The memory management unit (MMU) has a hardware segment table, each entry of which contains the physical address of the page table for the segment. Page tables are stored in the main memory and consists of 2 byte page table entries. What is the minimum page size in bytes so that the page table for a segment requires at most one page to store it?

### **Solution-**

Given-

- Virtual Address Space = Process size =  $2^{16}$  bytes
- Physical Address Space = Main Memory size =  $2^{16}$  bytes
- Process is divided into 8 equal size segments
- Page table entry size = 2 bytes

Let page size = n bytes.

Now, since page table has to be stored into a single page, so we must have-

$$\text{Size of page table} \leq \text{Page size}$$

### **Size of Each Segment-**

Size of each segment

$$= \text{Process size} / \text{Number of segments}$$

$$\begin{aligned}
&= 2^{16} \text{ bytes} / 8 \\
&= 2^{16} \text{ bytes} / 2^3 \\
&= 2^{13} \text{ bytes} \\
&= 8 \text{ KB}
\end{aligned}$$

### **Number of Pages Of Each Segment-**

Number of pages each segment is divided

$$\begin{aligned}
&= \text{Size of segment} / \text{Page size} \\
&= 8 \text{ KB} / n \text{ bytes} \\
&= (8K / n) \text{ pages}
\end{aligned}$$

### **Size of Each Page Table-**

Size of each page table

$$\begin{aligned}
&= \text{Number of entries in page table} \times \text{Page table entry size} \\
&= \text{Number of pages the segment is divided} \times 2 \text{ bytes} \\
&= (8K / n) \times 2 \text{ bytes} \\
&= (16K / n) \text{ bytes}
\end{aligned}$$

### **Page Size-**

Substituting values in the above condition, we get-

$$\begin{aligned}
(16K / n) \text{ bytes} &\leq n \text{ bytes} \\
(16K / n) &\leq n \\
n^2 &\geq 16K \\
n^2 &\geq 2^{14} \\
n &\geq 2^7
\end{aligned}$$

Thus, minimum page size possible =  $2^7$  bytes = 128 bytes.



## Virtual Memory:-

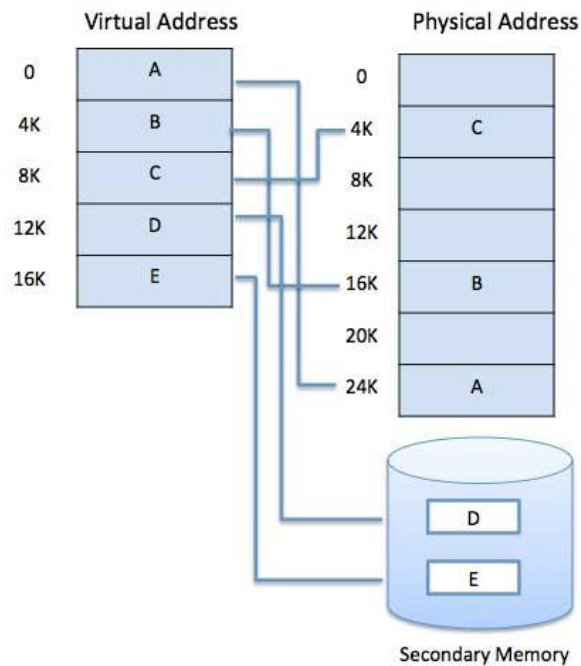
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

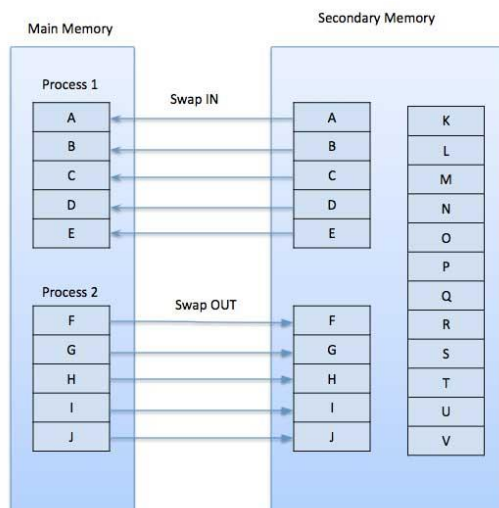
Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below –



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

### 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.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the 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

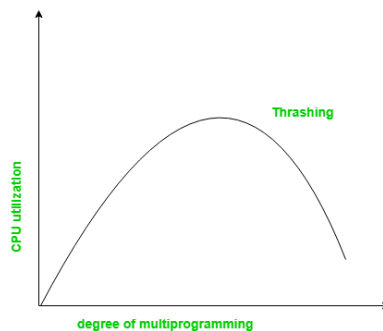
Following are the advantages of Demand Paging –

- 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.

**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 basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no useful work would be done by the CPU and the CPU utilization would fall drastically. The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the memory thereby increasing the degree of multiprogramming. This would result in a further decrease in the CPU utilization triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called Thrashing.