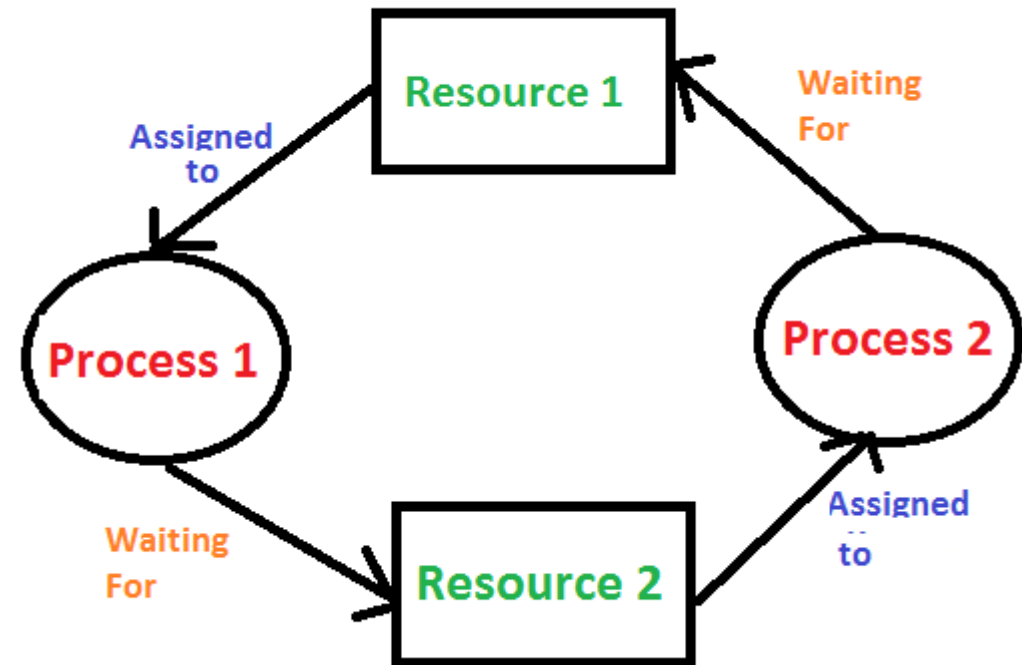
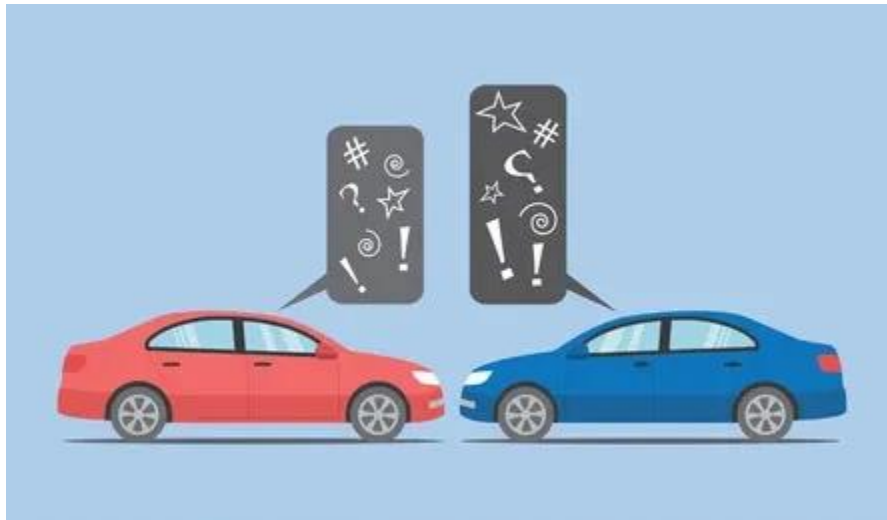


Unit 2 :Deadlocks, Memory Management

- Deadlocks: Definition
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Physical Memory and Virtual Memory
- Memory Allocation
- Internal and External fragmentation
- Contiguous Memory Allocation
- Non contiguous Memory Allocation
- Virtual Memory Using Paging
- Virtual Memory Using Segmentation

Deadlocks: Definition

- Deadlock is a situation where no process got blocked and no process proceeds.
- A deadlock is a situation where a set of processes is blocked because each process is holding a resource and waiting for another resource acquired by some other process.



Deadlock Prevention

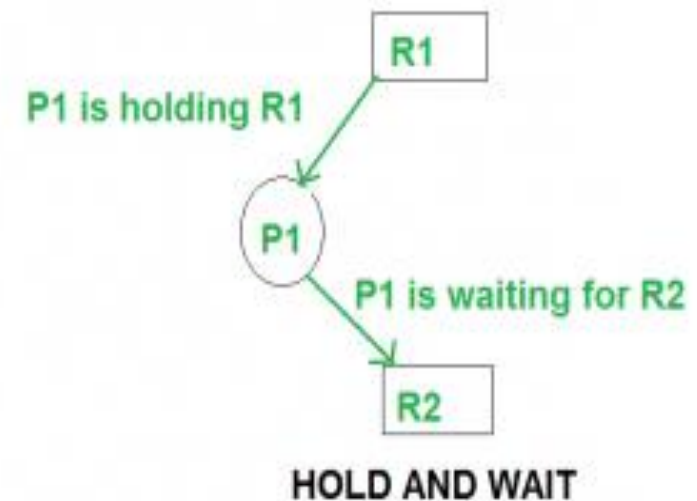
- In deadlock prevention the aim is to not let full-fill one of the required condition of the deadlock. This can be done by this method:
 - **(i) Mutual Exclusion**
 - **(ii) Hold and Wait**
 - **(iii) No Preemption**
 - **(iv) Circular Wait:**

1. Mutual Exclusion

- Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

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.

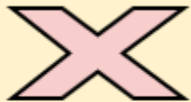
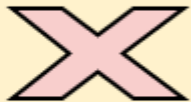
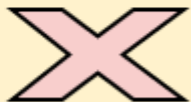



3. No Preemption

- 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.

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	

Deadlock Avoidance

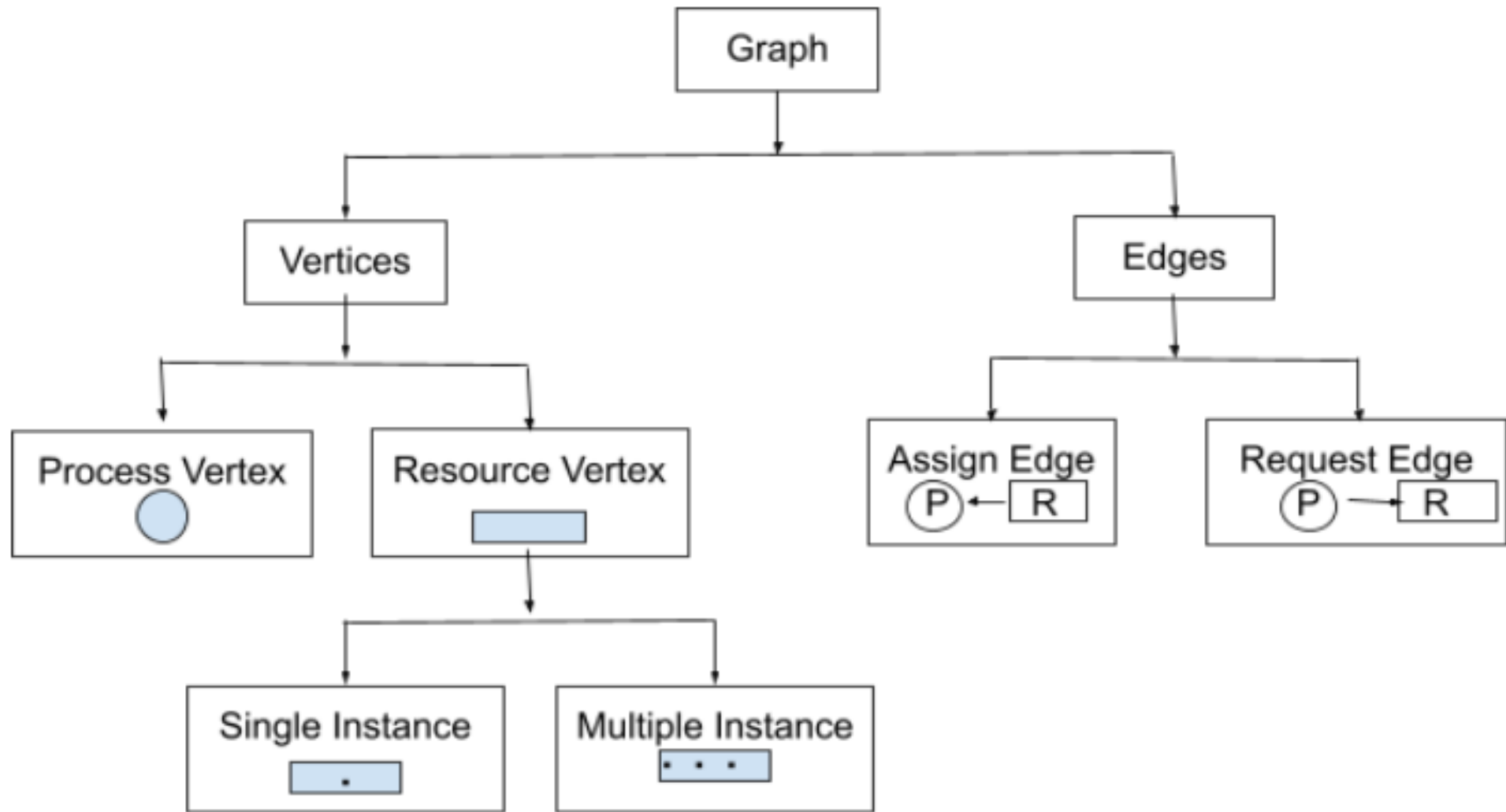
- In deadlock avoidance, the request for any resource will be granted if the resulting state of the system doesn't cause deadlock in the system. The state of the system will continuously be checked for safe and unsafe states.
- In order to avoid deadlocks, the process must tell OS, the maximum number of resources a process can request to complete its execution.
- Deadlock Avoidance can be solved by two different algorithms:
 - **Resource allocation Graph**
 - **Banker's Algorithm**

- ***Safe State:***

A state is safe if the system can allocate resources to each process in some order (up to its maximum requirement) while avoiding a deadlock.

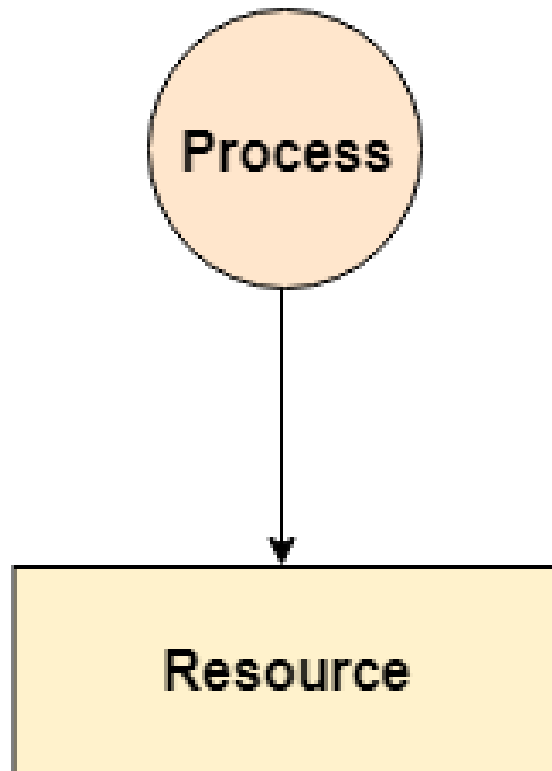
- ***Unsafe State:***

If the operating system is unable to prevent processes from requesting resources, resulting in Deadlock, then the system is said to be in an Unsafe State. Unsafe State may lead to a deadlock.

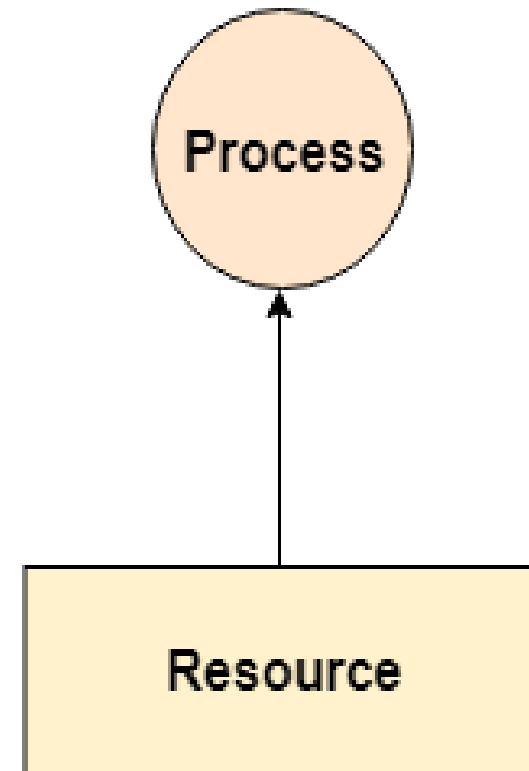


Resource Allocation Graph (RAG)

- The Resource Allocation Graph, also known as RAG is a graphical representation of the state of a system. It has all the information about the resource allocation to each process and the request of each process.
- A **Resource Allocation Graph** shows which **resources** are held by which **processes** and which processes are waiting for specific types of resources.
- A **Resource Allocation Graph (RAG)** is a visual way to understand how **resources** are assigned in an **operating system**. Instead of using only tables to show which resources are allocated, requested, or available, the RAG uses nodes and edges to clearly illustrate relationships between **processes** and their required resources.



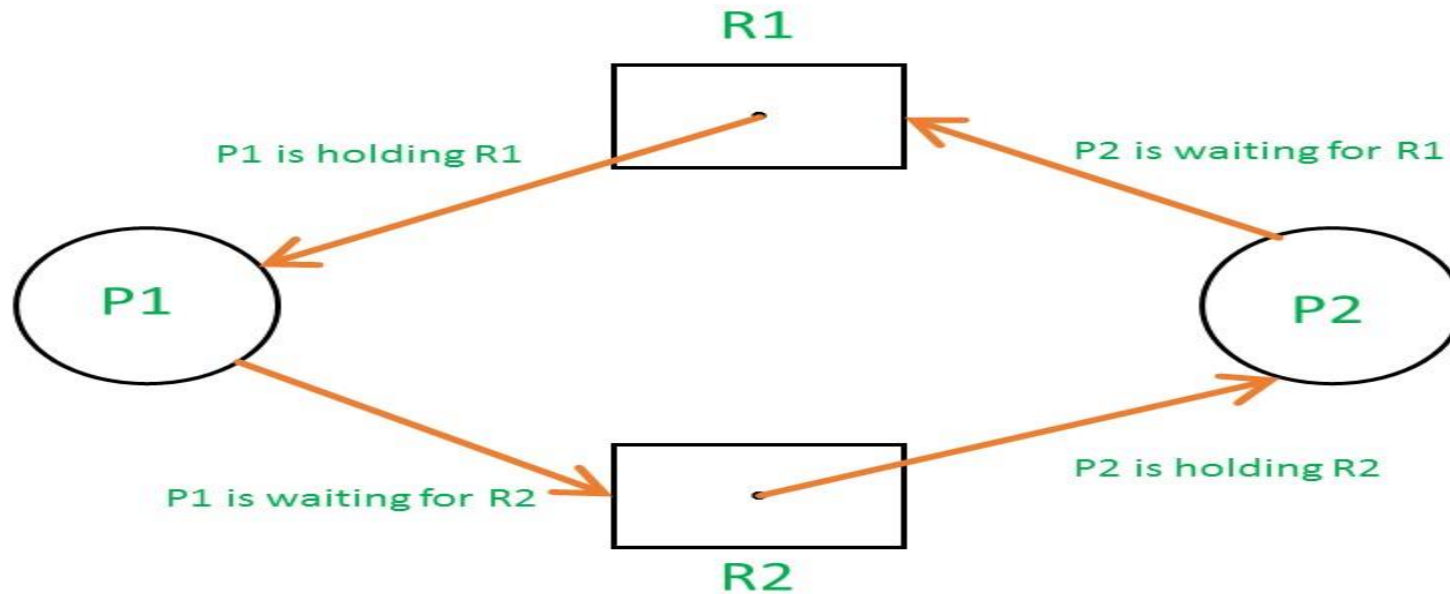
**Process is requesting
for a resource**



**Resource is assigned
to process**

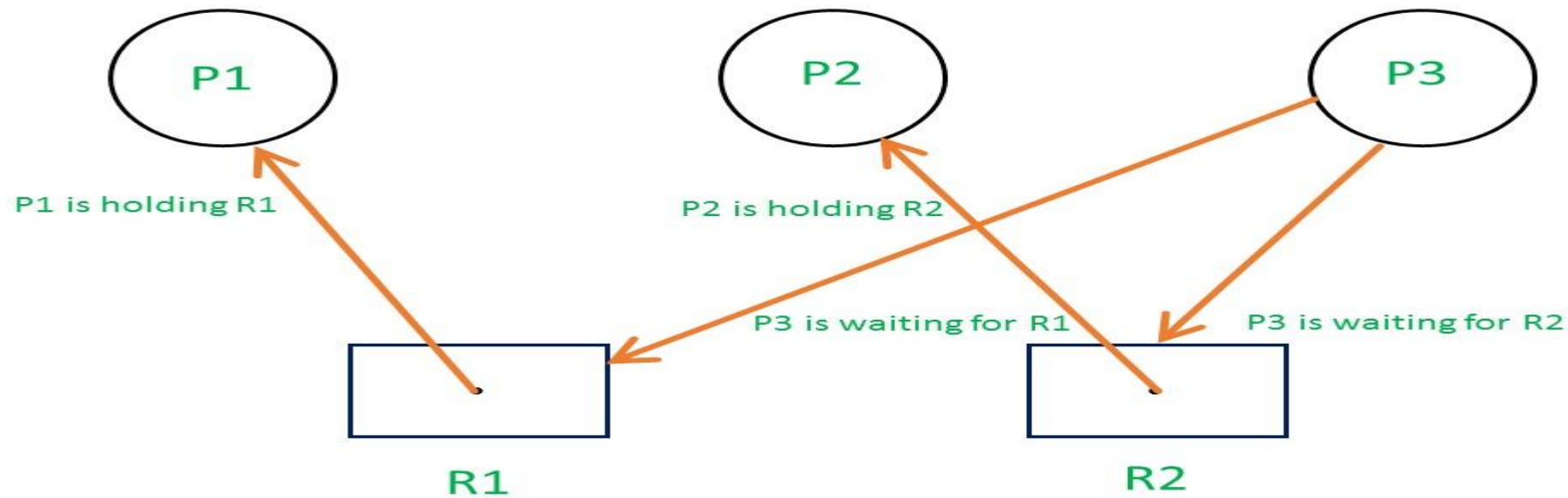
Example 1 (Single Instances RAG)

- If there is a cycle in the Resource Allocation Graph and each resource in the cycle provides only one instance, then the processes will be in deadlock. For example, if process P1 holds resource R1, process P2 holds resource R2 and process P1 is waiting for R2 and process P2 is waiting for R1, then process P1 and process P2 will be in deadlock.



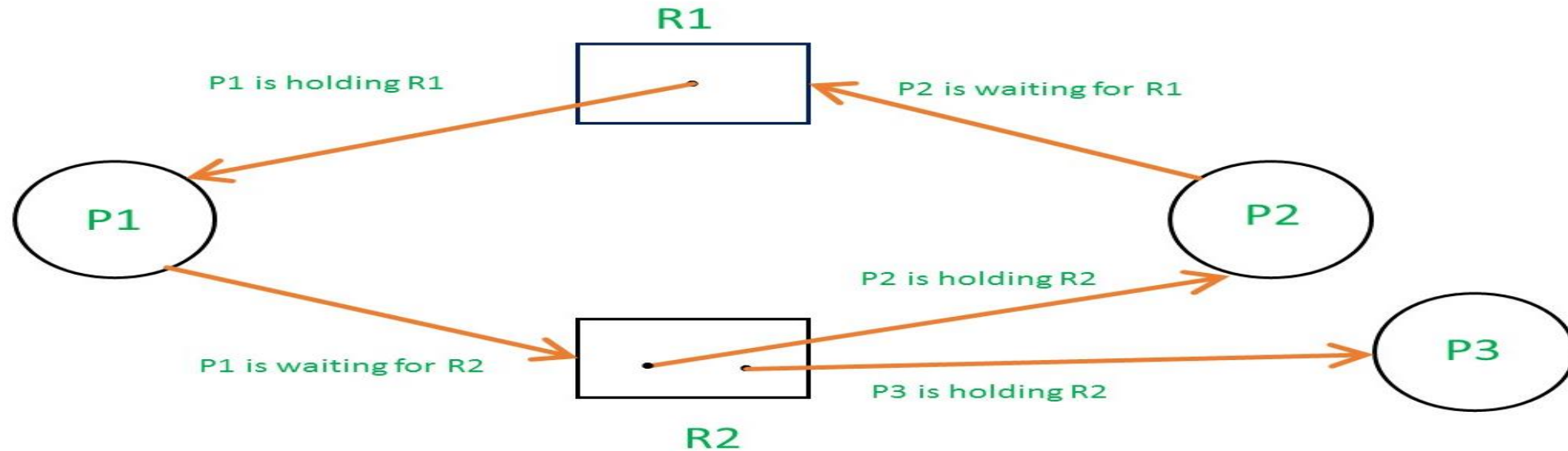
SINGLE INSTANCE RESOURCE TYPE WITH DEADLOCK

- Here's another example, that shows Processes P1 and P2 acquiring resources R1 and R2 while process P3 is waiting to acquire both resources. In this **example**, there is no deadlock because there is no circular dependency. So cycle in single-instance resource type is the sufficient condition for deadlock.



SINGLE INSTANCE RESOURCE TYPE WITHOUT DEADLOCK

Example 2 (Multi-instances RAG)



MULTI INSTANCES WITHOUT DEADLOCK

- From the above example, it is not possible to say the RAG is in a safe state or in an unsafe state. So to see the state of this RAG, let's construct the allocation matrix and request matrix.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	0	0

The total number of processes are three: P1, P2 & P3 and the total number of resources are two: R1 & R2.

- **Allocation matrix**

- For constructing the allocation matrix, just go to the resources and see to which process it is allocated.
- R1 is allocated to P1, therefore write 1 in allocation matrix and similarly, R2 is allocated to P2 as well as P3 and for the remaining element just write 0.

- **Request matrix**

- In order to find out the request matrix, you have to go to the process and see the outgoing edges.
- P1 is requesting resource R2, so write 1 in the matrix and similarly, P2 requesting R1 and for the remaining element write 0.
- So now available resource is = (0, 0).

- **Checking deadlock (safe or not)**

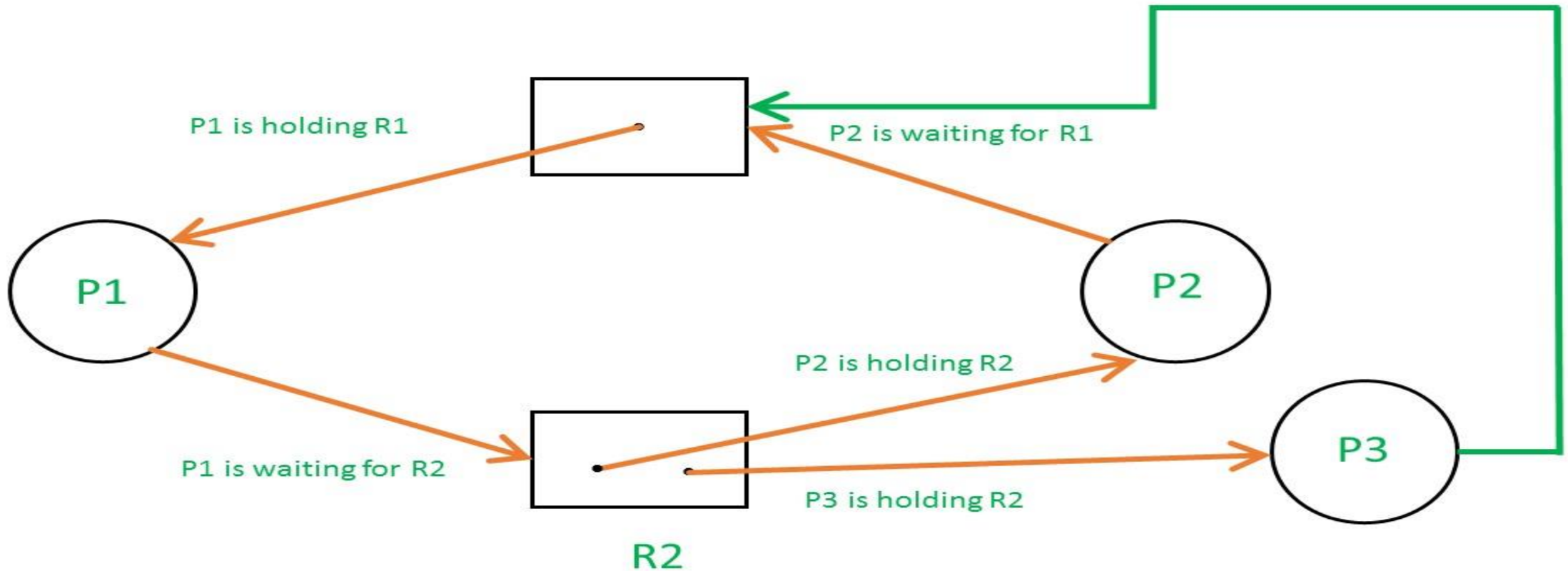
Available = 0 0 (As P3 does not require any extra resource to complete the execution and after
P3 0 1 completion P3 release its own resource)

New Available = 0 1 (As using new available resource we can satisfy the requirement of process P1
P1 1 0 and P1 also release its previous resource)

New Available = 1 1 (Now easily we can satisfy the requirement of process P2)
P2 0 1

New Available = 1 2

- So there is no deadlock in this RAG. Even though there is a cycle, still there is no deadlock. Therefore in multi-instance resource cycle is not sufficient condition for deadlock.



MULTI INSTANCES WITH DEADLOCK

- Above example is the same as the previous example except that, the process P3 requesting for resource R1. So the table becomes as shown in below.

Process	Allocation		Request	
	Resource		Resource	
	R1	R2	R1	R2
P1	1	0	0	1
P2	0	1	1	0
P3	0	1	1	0

- So, the Available resource is = $(0, 0)$, but requirement are $(0, 1)$, $(1, 0)$ and $(1, 0)$. So you can't fulfill any one requirement. Therefore, it is in deadlock. Therefore every cycle in a multi-instance resource type graph is not a deadlock. If there has to be a deadlock, there has to be a cycle. So in case of RAG with multi-instance resource type, the cycle is a necessary condition for deadlock but not sufficient.

Banker's Algorithm

- There is an algorithm called Banker's Algorithm used in removing deadlocks while dealing with the safe allocation of resources to processes in a computer system. It gives all the resources to each process and avoids resource allocation conflicts.

Real Life Example

- Imagine having a bank with T amount of money and n account holders. At some point, whenever one of the account holders requests a loan:
- The bank withdraws the requested amount of cash from the total amount available for any further withdrawals.
- The bank checks if the cash that is available for withdrawal will be enough to cater to all future requests/withdrawals.
- If there is enough money available (that is to say, the available cash is greater than T), he lends the loan.
- This ensures that the bank will not suffer operational problems when it receives subsequent applications.

Example

- Consider a system that contain five process P1,P2,P3,P4,P5 and three resource types R1,R2,R3. R1 has 10, R2 has 5 and R3 has 7 instance.

Process	Allocation			Max			Available		
	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3	3	3	2
P2	2	0	0	3	2	2			
P3	3	0	2	9	0	2			
P4	2	1	1	2	1	1			
P5	0	0	2	4	3	3			

- Step 1 work available work = 3 3 2
- Step 2 Find needs (Needs=Max-Allocation)

Process	Allocation			Max			Available			Needs		
	R1	R2	R3	R1	R2	R3	R1	R2	R3	R1	R2	R3
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2				1	2	2
P3	3	0	2	9	0	2				6	0	0
P4	2	1	1	2	1	1				0	0	0
P5	0	0	2	4	3	3				4	3	1

Step 3 :

(i) $\text{Needs}(P1) \leq \text{work}$ $7\ 4\ 3 \leq 3\ 3\ 2$ Condition False

(ii) $\text{Needs}(P2) \leq \text{work}$ $1\ 2\ 2 \leq 3\ 3\ 2$ Condition True so that is Executed

Work = Work + Allocation of P2 [Work = $3\ 3\ 2 + 2\ 0\ 0 = 5\ 3\ 2$]

(iii) $\text{Needs}(P3) \leq \text{work}$ $6\ 0\ 0 \leq 5\ 3\ 2$ Condition False

(iv) $\text{Needs}(P4) \leq \text{work}$ $0\ 0\ 0 \leq 5\ 3\ 2$ Condition True so that is Executed

Work = Work + allocation of P4 [Work = $5\ 3\ 2 + 2\ 1\ 1 = 7\ 4\ 3$]

New work = work + allocation of P4 [$5\ 3\ 2 + 2\ 1\ 1 = 7\ 3\ 4$]

(V) $\text{Needs}(5\ \text{process}) \leq \text{work}$ $4\ 3\ 1 \leq 7\ 4\ 3$ Condition True so that is executed

Work = work + allocation of p4 [Work = $7\ 4\ 3 + 0\ 0\ 2 = 7\ 4\ 5$]

New work = work + allocation of p5 [$7\ 4\ 3 + 0\ 0\ 2 = 7\ 4\ 5$]

Needs		Condition			work
P1	743	\leq	332	FALSE	
P2	122	\leq	332	TRUE	$332+200=532$
P3	600	\leq	532	FALSE	
P4	000	\leq	532	TRUE	$532+211=743$
P5	431	\leq	743	TRUE	$743+002=745$

Step 4:

(i) Needs(1 process) \leq Work 7 4 3 \leq 7 4 5 Condition True so that is executed

Work = work + Allocation of P1 [work = 7 4 5 + 0 1 0 = 7 5 5]

New work = Work + Allocation of P1 [7 4 5 + 0 1 0 = 7 5 5]

(iii) Needs(3 process) \leq Work 7 4 3 \leq 7 5 5 Condition True so that is executed

Work = Work + Allocation of P3 [work = 7 5 5 + 3 0 2 = 10 5 7]

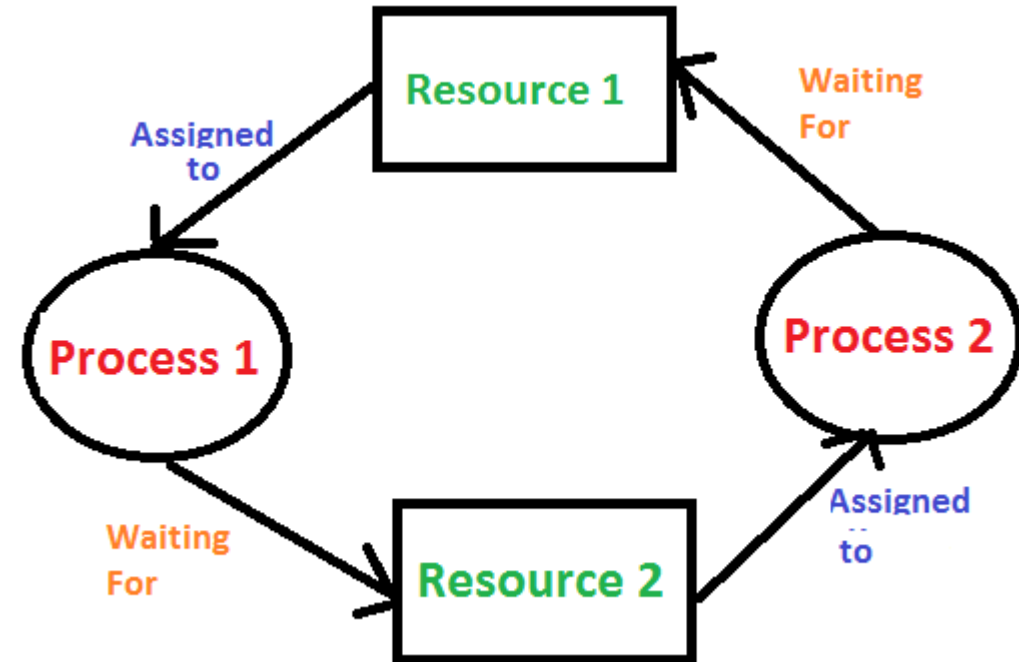
New work = Work + Allocation of P3 [7 5 5 + 3 0 2 = **10 5 7**] Safe state : [P2, P4, P5, P1, P3]

Needs	Condition				work
P1	743	\leq	745	TRUE	745+010=755
P2	122	\leq	332	TRUE	332+200=532
P3	600	\leq	755	TRUE	755+302=1057
P4	000	\leq	532	TRUE	532+211=743
P5	431	\leq	743	TRUE	743+002=745

Deadlock Detection

1. If Resources Have a Single Instance (Wait-For Graph Algorithm)

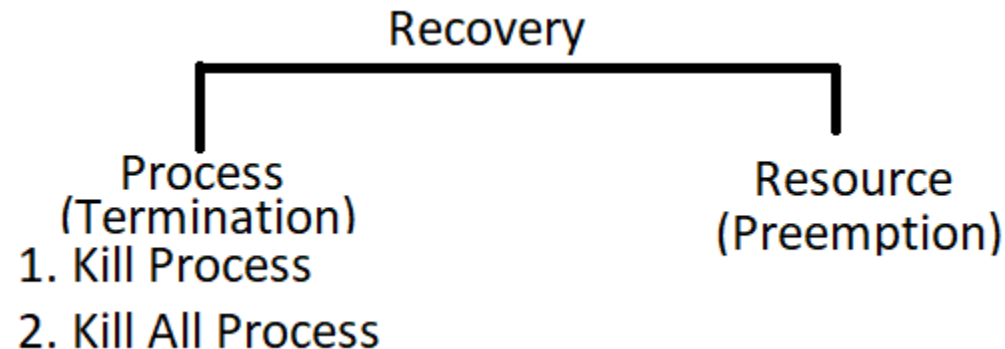
- In this case for Deadlock detection, we can run an algorithm to check for the cycle in the Resource Allocation Graph. The presence of a cycle in the graph is a sufficient condition for deadlock.
- In the above diagram, resource 1 and resource 2 have single instances. There is a cycle $R1 \rightarrow P1 \rightarrow R2 \rightarrow P2$. So, Deadlock is Confirmed.



- **2. If There are Multiple Instances of Resources**
- Detection of the cycle is necessary but not a sufficient condition for deadlock detection, in this case, the system may or may not be in deadlock varies according to different situations.
- **EX : Banker's Algorithm**

Deadlock Recovery

- **Killing The Process:** Killing all the processes involved in the deadlock. Killing process one by one. After killing each process check for deadlock again and keep repeating the process till the system recovers from deadlock. Killing all the processes one by one helps a system to break circular wait conditions.
- **Resource Preemption:** Resources are preempted from the processes involved in the deadlock, and preempted resources are allocated to other processes so that there is a possibility of recovering the system from the deadlock. In this case, the system goes into starvation



Physical Memory and Virtual Memory

Physical Memory

- Physical memory refers to the RAM or the primary memory in the computer. Physical memory is a volatile memory. Therefore, it requires a continuous flow of power to retain data. However, power failures and interruptions can erase the data in the physical memory.
- Physical memory, also known as RAM (random access memory), is the hardware component that stores data and instructions that can be accessed directly by the processor. It is faster than secondary storage devices, but also more expensive and limited in capacity.
- Random Access Memory (RAM) is a type of computer memory that stores data temporarily while a computer is running. It's called "random access" because the computer can access any part of the memory directly and quickly. RAM (Random Access Memory) is very similar to memory in the Human Brain. The human brain's memory is the most essential part played by the brain.

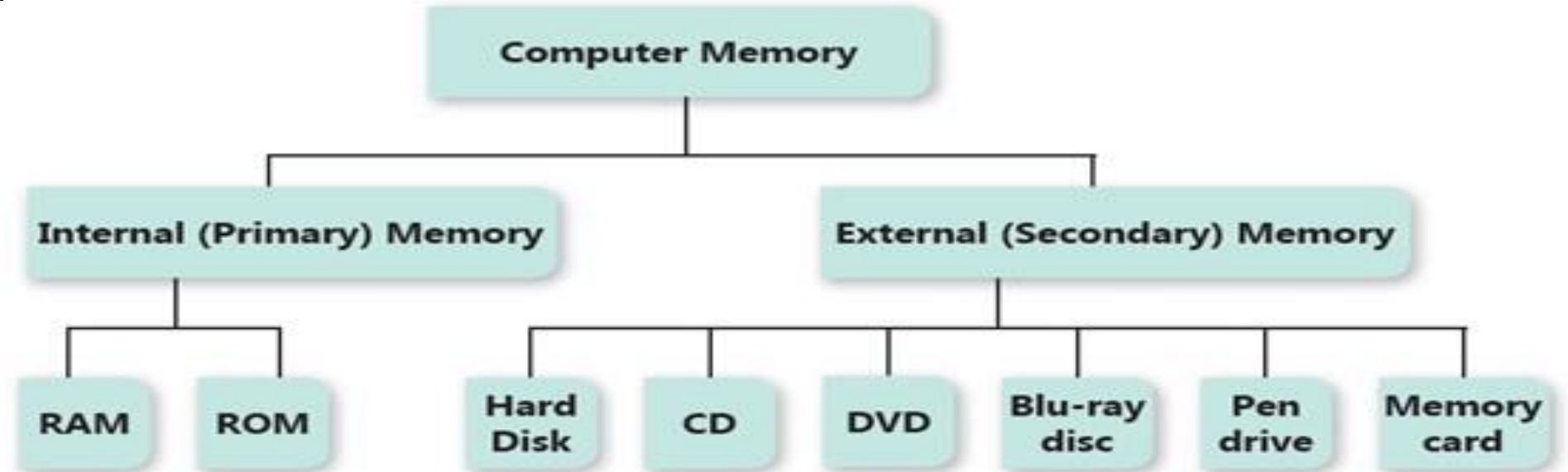
RAM



ROM



- It is one of the parts of the Main memory, also famously known as Read Write Memory. Random Access memory is present on the [motherboard](#) and the computer's data is temporarily stored in [RAM](#).
- Random Access Memory (RAM) is a type of computer memory that stores data temporarily while a computer is running. It's called "random access" because the computer can access any part of the memory directly and quickly. RAM (Random Access Memory) is very similar to memory in the Human Brain. The human brain's memory is the most essential part played by the brain.



Virtual Memory

- Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited. It allows larger applications to run on systems with less RAM.
- Virtual memory uses both hardware and software to operate. When an application is in use, data from that program is stored in a physical address using RAM. A memory management unit (MMU) maps the address to RAM and automatically translates addresses. The MMU can, for example, map a logical address space to a corresponding physical address.
- There are two main types of virtual memory:
 - Paging
 - Segmentation

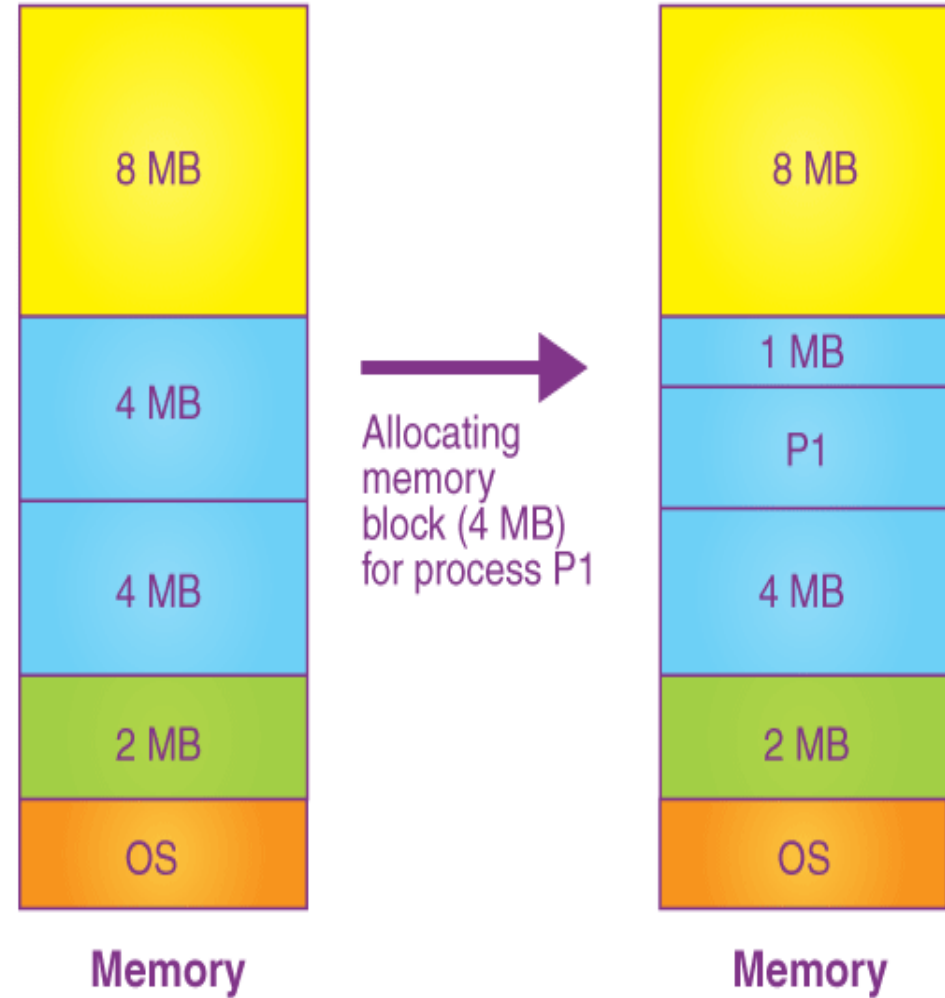
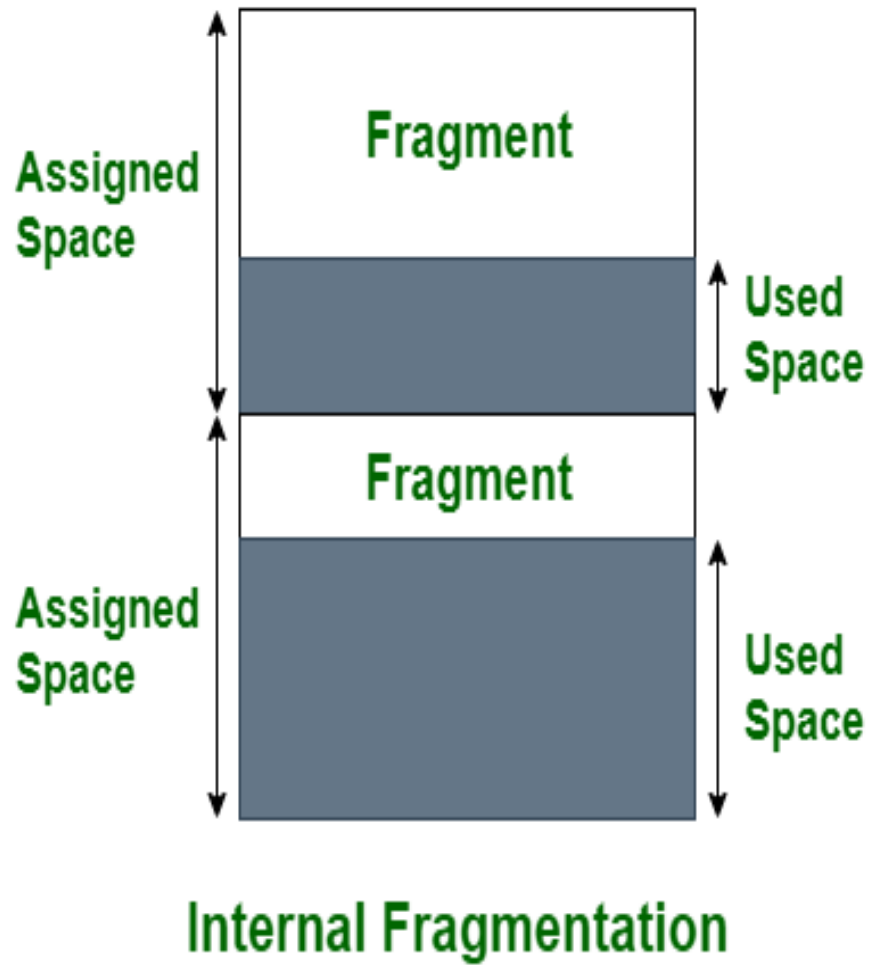
Feature	Virtual Memory	Physical Memory (RAM)
Definition	An abstraction that extends the available memory by using disk storage	The actual hardware (RAM) that stores data and instructions currently being used by the CPU
Location	On the hard drive or SSD	On the computer's motherboard
Speed	Slower (due to disk I/O operations)	Faster (accessed directly by the CPU)
Capacity	Larger, limited by disk space	Smaller, limited by the amount of RAM installed
Cost	Lower (cost of additional disk storage)	Higher (cost of RAM modules)
Data Access	Indirect (via paging and swapping)	Direct (CPU can access data directly)
Volatility	Non-volatile (data persists on disk)	Volatile (data is lost when power is off)

Fragmentation

- The process of dividing a computer file, such as a data file or an executable program file, into fragments that are stored in different parts of a computer's storage medium, such as its hard disc or RAM, is known as fragmentation in computing.
- There are two main types of fragmentation:
 - Internal Fragmentation
 - External Fragmentation

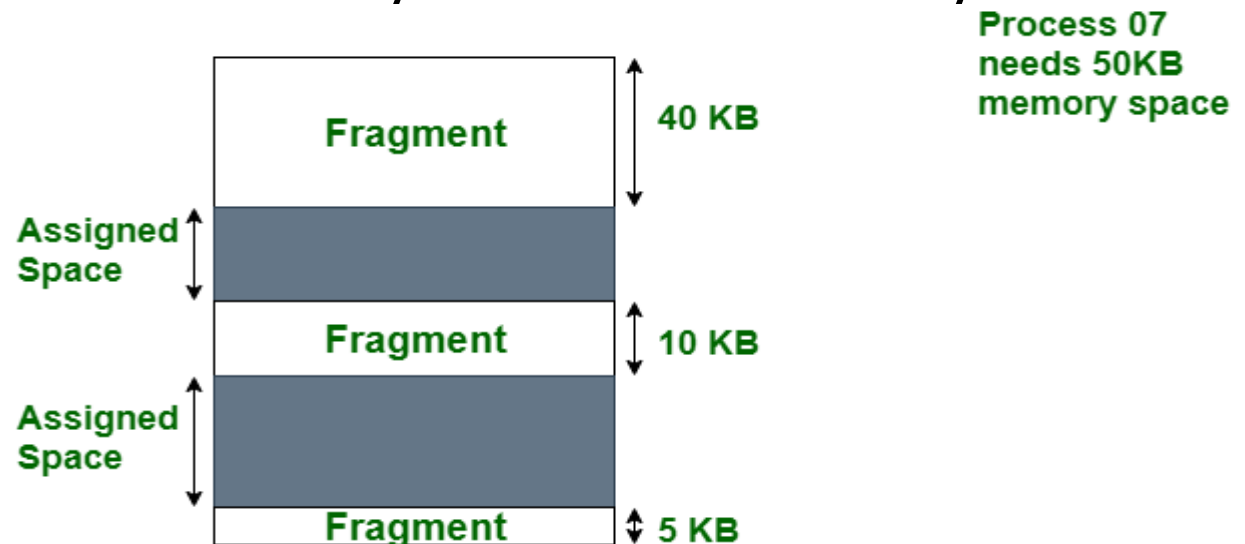
Internal Fragmentation

- Whenever a memory block gets allocated with a process, and in case the process happens to be smaller than the total amount of requested memory, a free space is ultimately created in this memory block. And due to this, the memory block's free space is unused. This is what causes internal fragmentation.
- For example, in the case of dynamic storage allocation, the storage reservoirs reduce internal fragmentation greatly by the extension of the space overhead over a significant amount of elements.
- In the figure mentioned below, you can see internal fragmentation. Internal fragmentation is considered to be the distinction between the needed space or memory and the assigned storage space.

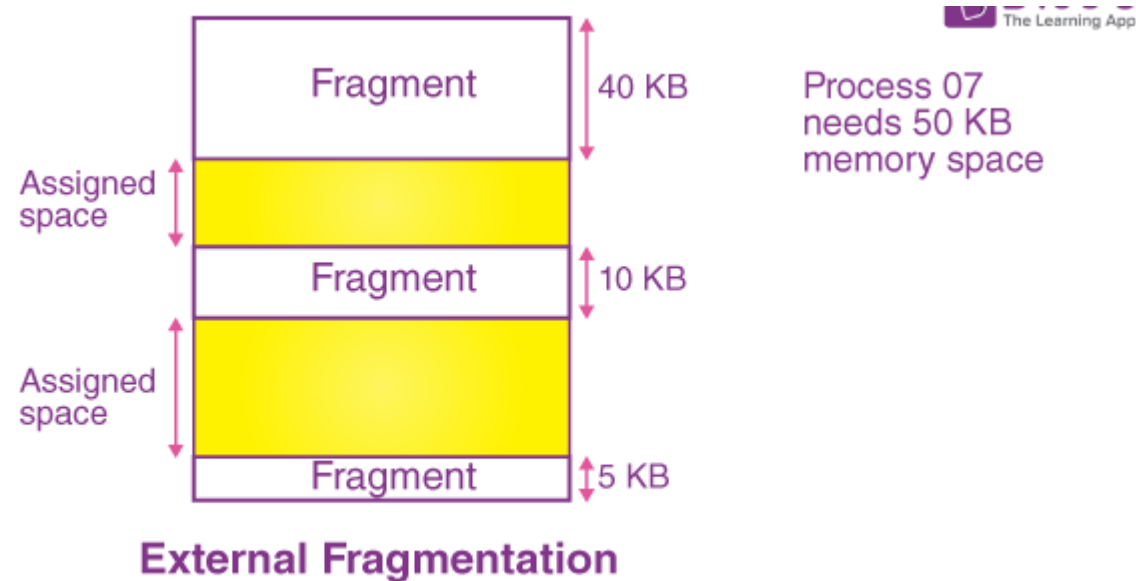


External Fragmentation

- External fragmentation occurs whenever a method of dynamic memory allocation happens to allocate some memory and leave a small amount of unusable memory. The total quantity of the memory available is reduced substantially in case there's too much external fragmentation.
- External fragmentation occurs whenever the used storage gets differentiated into certain smaller lots and punctuated using assigned memory space. It's actually a weak point of various storage allocation methodologies when they can't actually schedule memory utilised by systems.



- **Example :**
- As you can see in the illustration mentioned below, there is sufficient memory space (55 KB) in order to execute process 07 (mandated 50 KB). But here, the storage (fragment) isn't adjacent. Thus, to use the empty room to run a procedure, one can use paging, compression, or segmentation strategies.

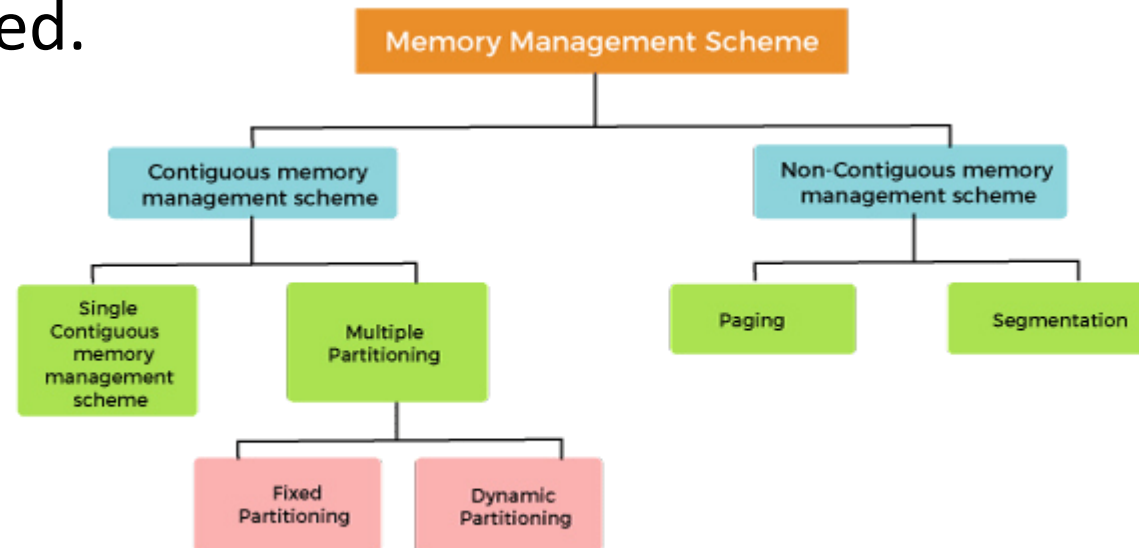


Internal fragmentation vs External fragmentation

Internal fragmentation	External fragmentation
In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	In external fragmentation, variable-sized memory blocks square measure appointed to the method.
Internal fragmentation happens when the method or process is smaller than the memory.	External fragmentation happens when the method or process is removed.
The solution of internal fragmentation is the best-fit block .	The solution to external fragmentation is compaction and paging .
Internal fragmentation occurs when memory is divided into fixed-sized partitions .	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
The difference between memory allocated and required space or memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation.
Internal fragmentation occurs with paging and fixed partitioning.	External fragmentation occurs with segmentation and dynamic partitioning .
It occurs on the allocation of a process to a partition greater than the process's requirement. The leftover space causes degradation system performance.	It occurs on the allocation of a process to a partition greater which is exactly the same memory space as it is required.
It occurs in worst fit memory allocation method .	It occurs in best fit and first fit memory allocation method.

Memory allocation

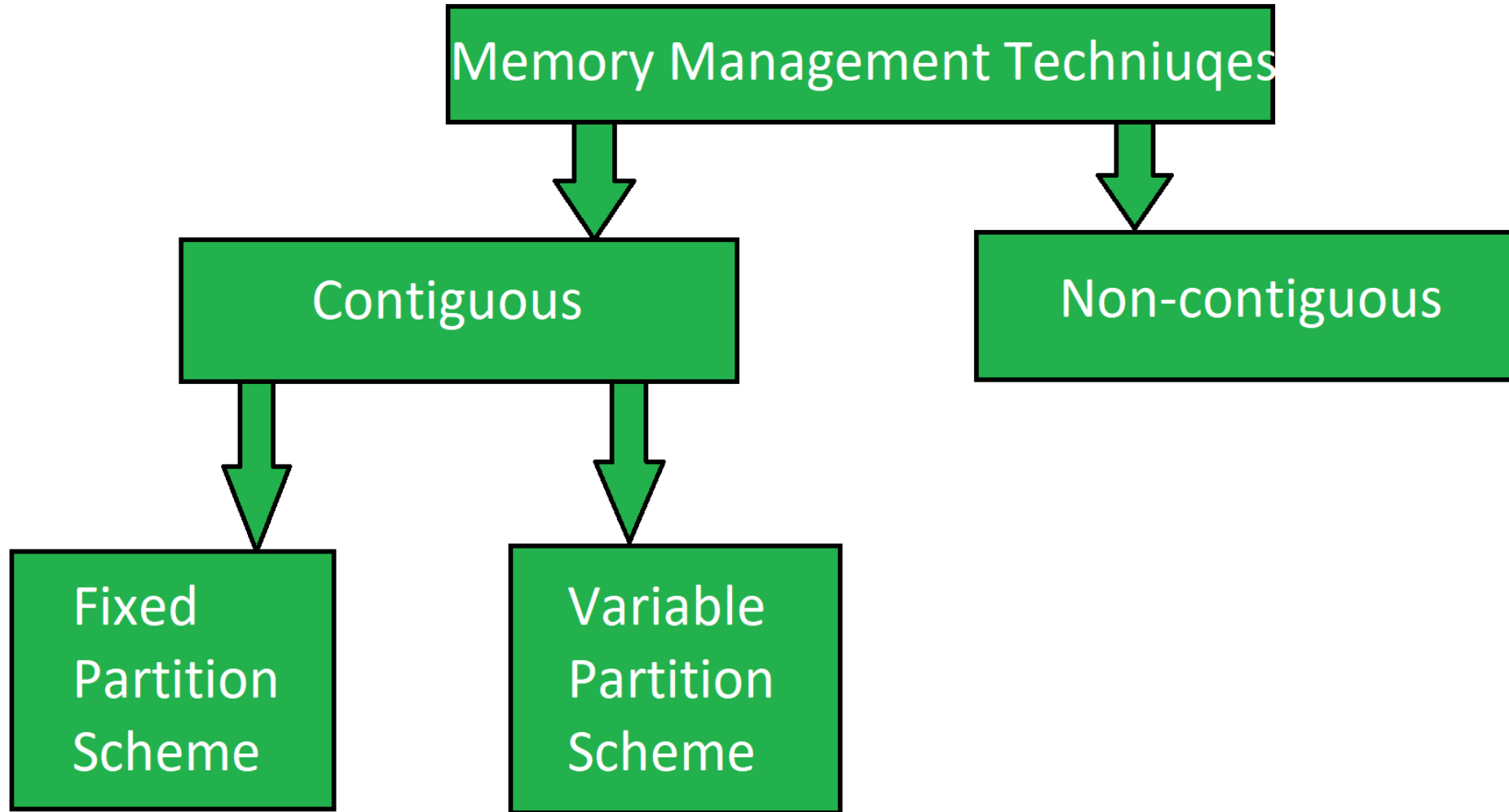
- Memory allocation is the process of reserving virtual or physical computer space for a specific purpose (e.g., for computer programs and services to run). Memory allocation is part of the management of computer memory resources, known as memory management. Through memory allocation, computer programs and services are assigned a specific memory portion, depending on how much memory they need.



Classification of memory management schemes

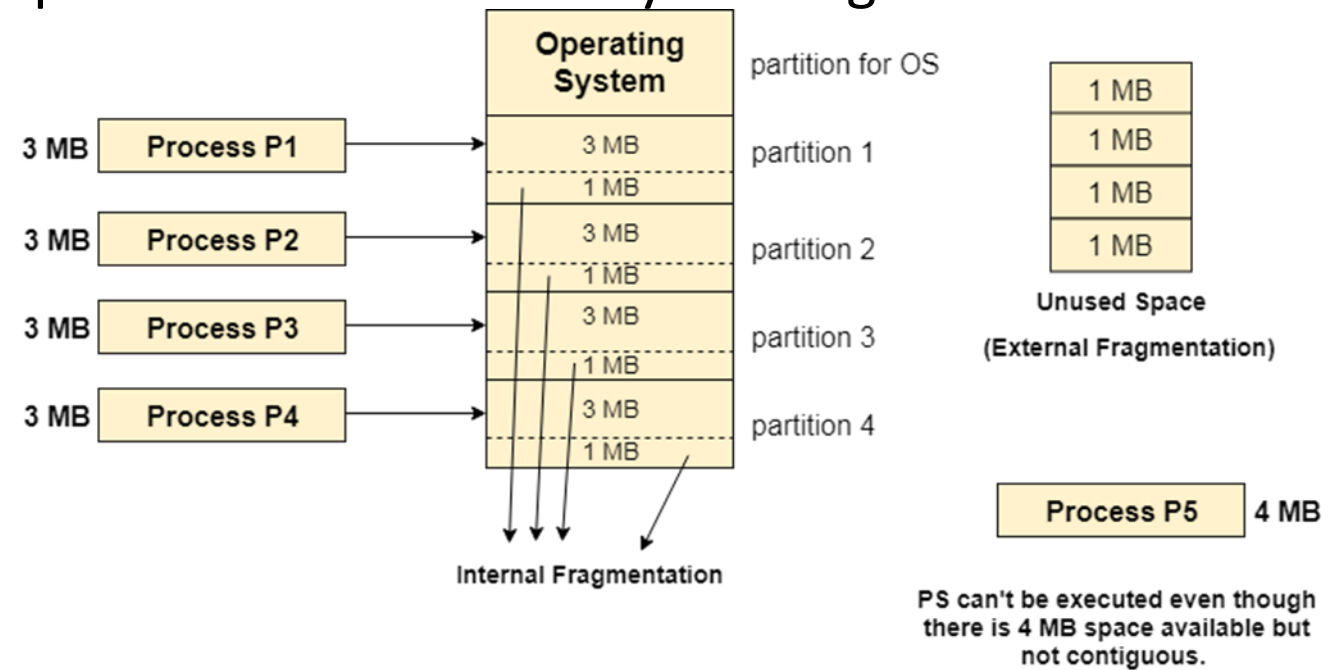
Contiguous memory allocation

- Contiguous memory allocation is a memory allocation strategy. As the name implies, we utilize this technique to assign contiguous blocks of memory to each task. Thus, whenever a process asks to access the main memory, we allocate a continuous segment from the empty region to the process based on its size. In this technique, memory is allotted in a continuous way to the processes. Contiguous Memory Management has two types:
 - Fixed(or Static) Partition
 - 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. The memory is assigned to the processes in contiguous way.

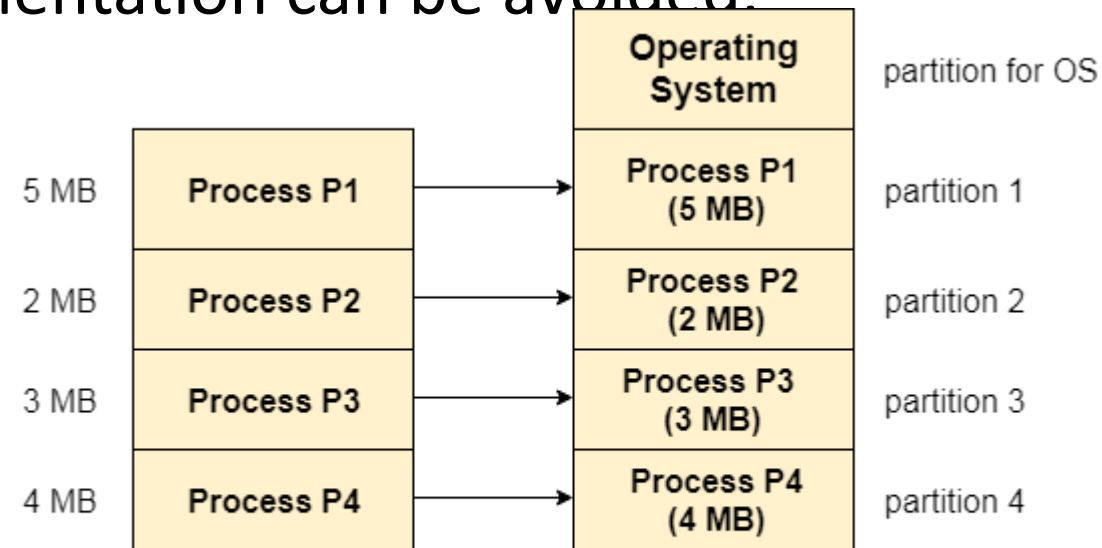


Fixed Partitioning

(Contiguous memory allocation)

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.

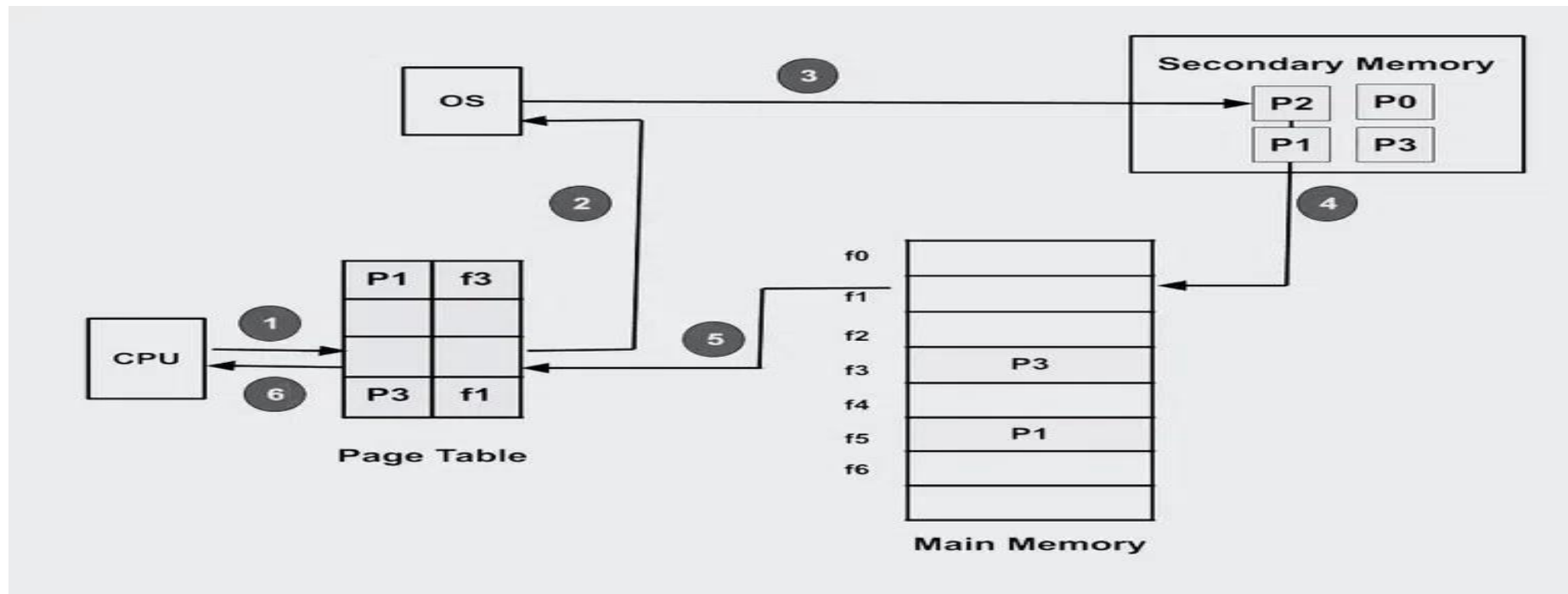


Non-Contiguous Allocation

- Non-contiguous allocation, also known as dynamic or linked allocation, is a memory allocation technique used in operating systems to allocate memory to processes that do not require a contiguous block of memory. In this technique, each process is allocated a series of non-contiguous blocks of memory that can be located anywhere in the physical memory.
 - Paging
 - Segmentation

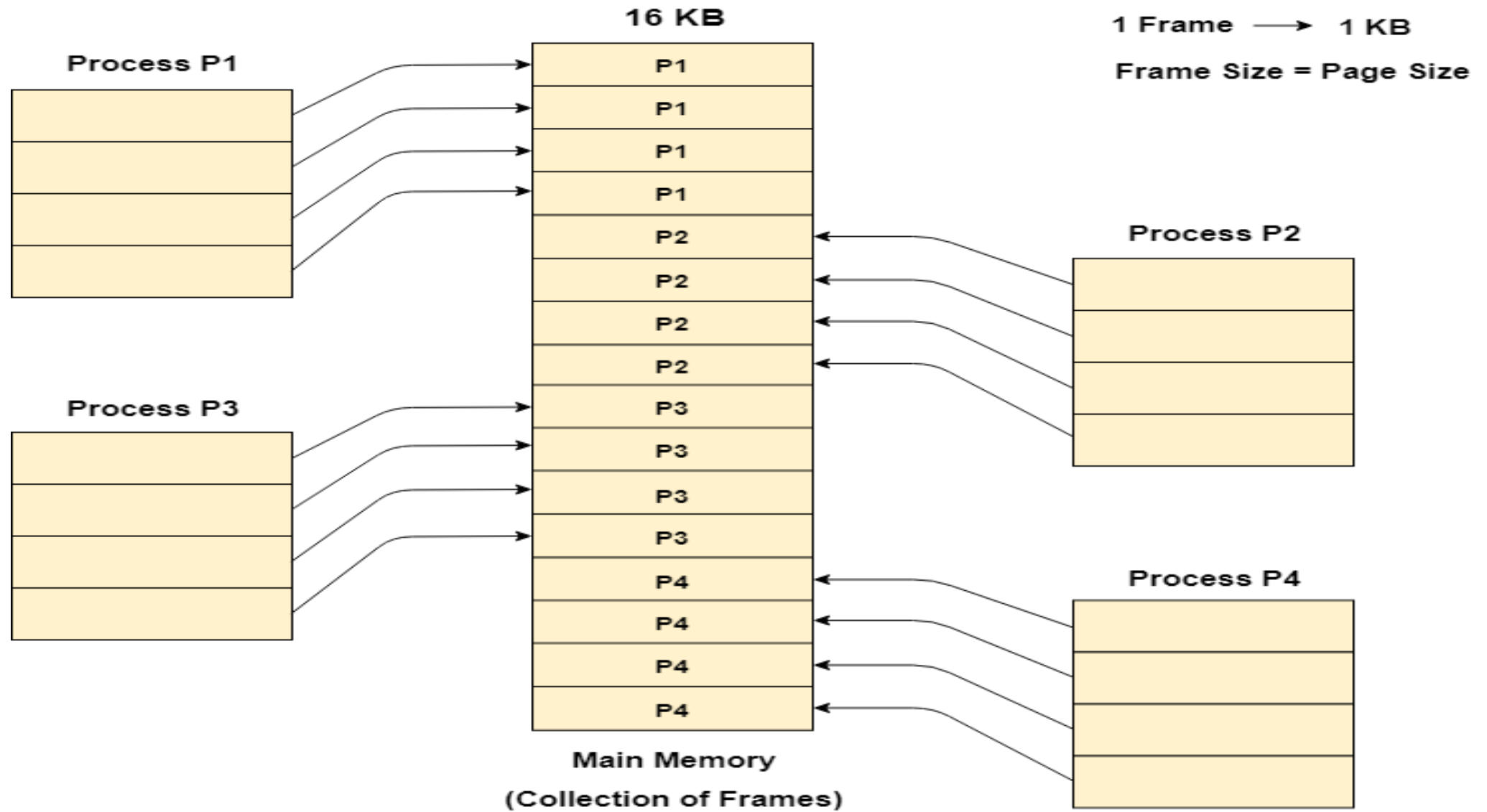
Paging

- Paging divides memory into small fixed-size blocks called pages. When the computer runs out of RAM, pages that aren't currently in use are moved to the hard drive, into an area called a swap file. The swap file acts as an extension of RAM. When a page is needed again, it is swapped back into RAM, a process known as page swapping. This ensures that the operating system (OS) and applications have enough memory to run.
- Paging is a technique that divides memory into fixed-sized blocks. The main memory is divided into blocks known as Frames and the logical memory is divided into blocks known as Pages.



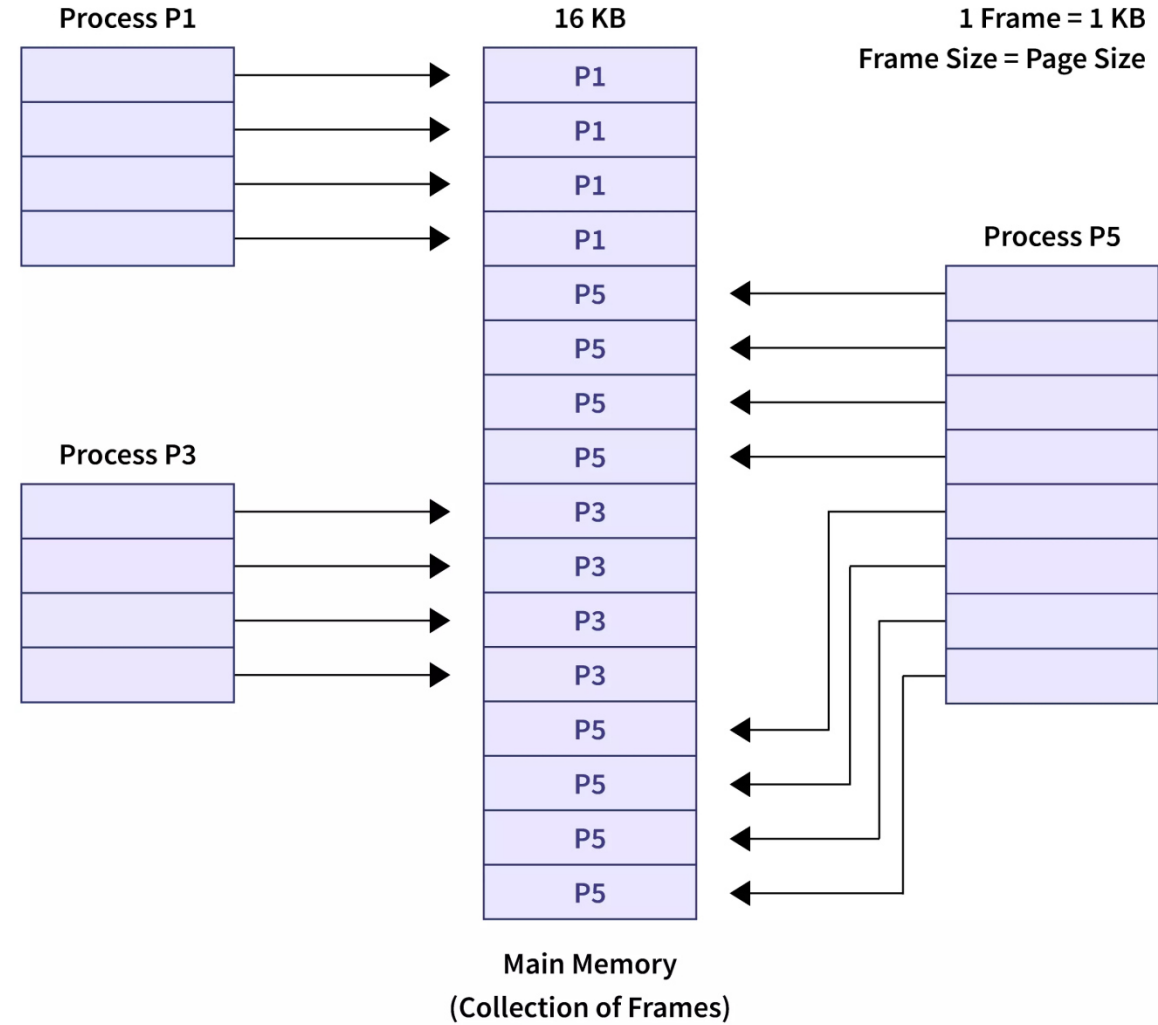
Example (Contiguous Allocation of Pages)

- Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.
- There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.
- Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.
- Frames, pages and the mapping between the two is shown in the image below.



Paging

- Example (**Non - Contiguous Allocation of Pages**)



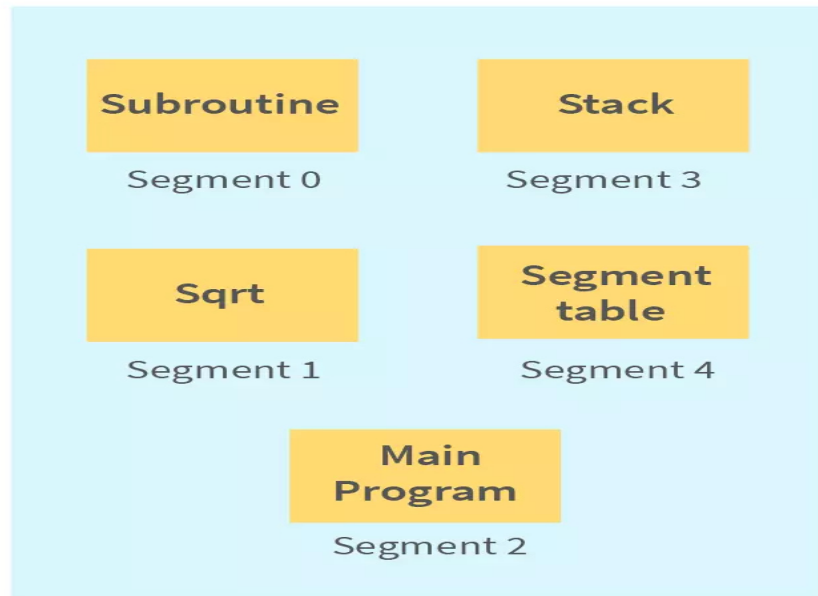
Segmentation

Segmentation is a memory administration approach used in operating systems that divides memory into multiple-sized segments.

Segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process.

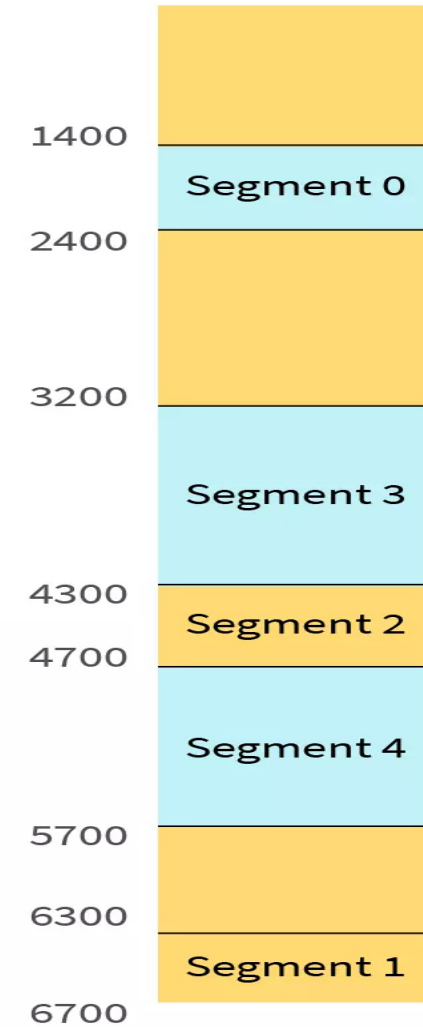
The details about each segment are stored in a table called a segment table. Segment table is stored in one (or many) of the segments. A process can be assigned to each component, referred to as a segment.

An offset generally denotes the number add address location that need to be added to the base address to get the specified absolute physical address

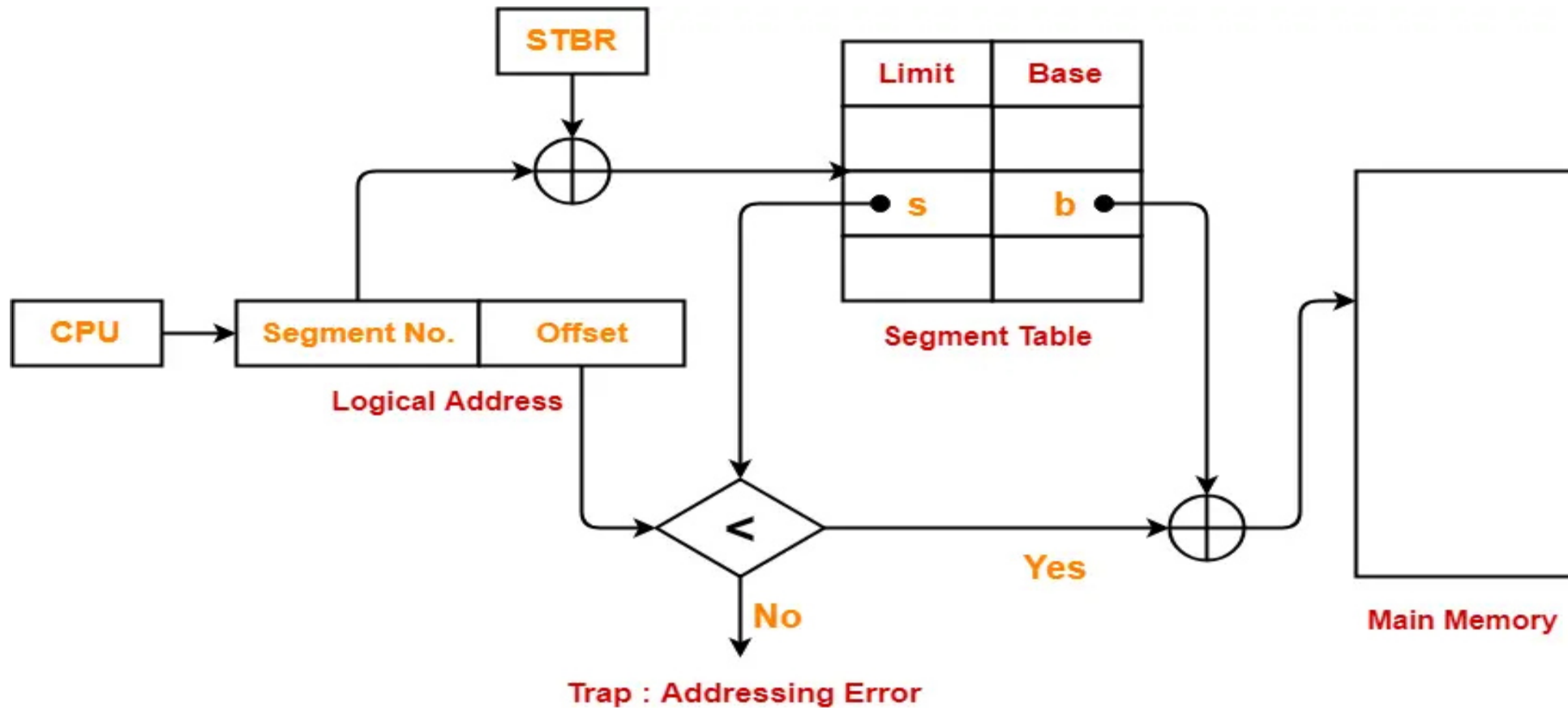


Logical Address Space

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



Physical Memory



Translating Logical Address into Physical Address

Assignment 2

- Explain deadlocks in details
- Explain Deadlocks Avoidance, Resource allocation Graph And Banker's Algorithm
- Explain deadlock detection and recovery
- Explain Virtual Memory & Physical Memory
- Explain Virtual Memory with segmentation
- Explain Virtual Memory using paging
- Explain types of Memory Allocation