Memory Management

Module 2.1

Memory Management : Introduction

- The task of subdividing the memory among different processes is called Memory Management.
- Memory management is a method in the operating system to manage operations between main memory and disk during process execution.
- The main aim of memory management is to achieve efficient utilization of memory.

Logical and Physical Address Space

Logical Address Space

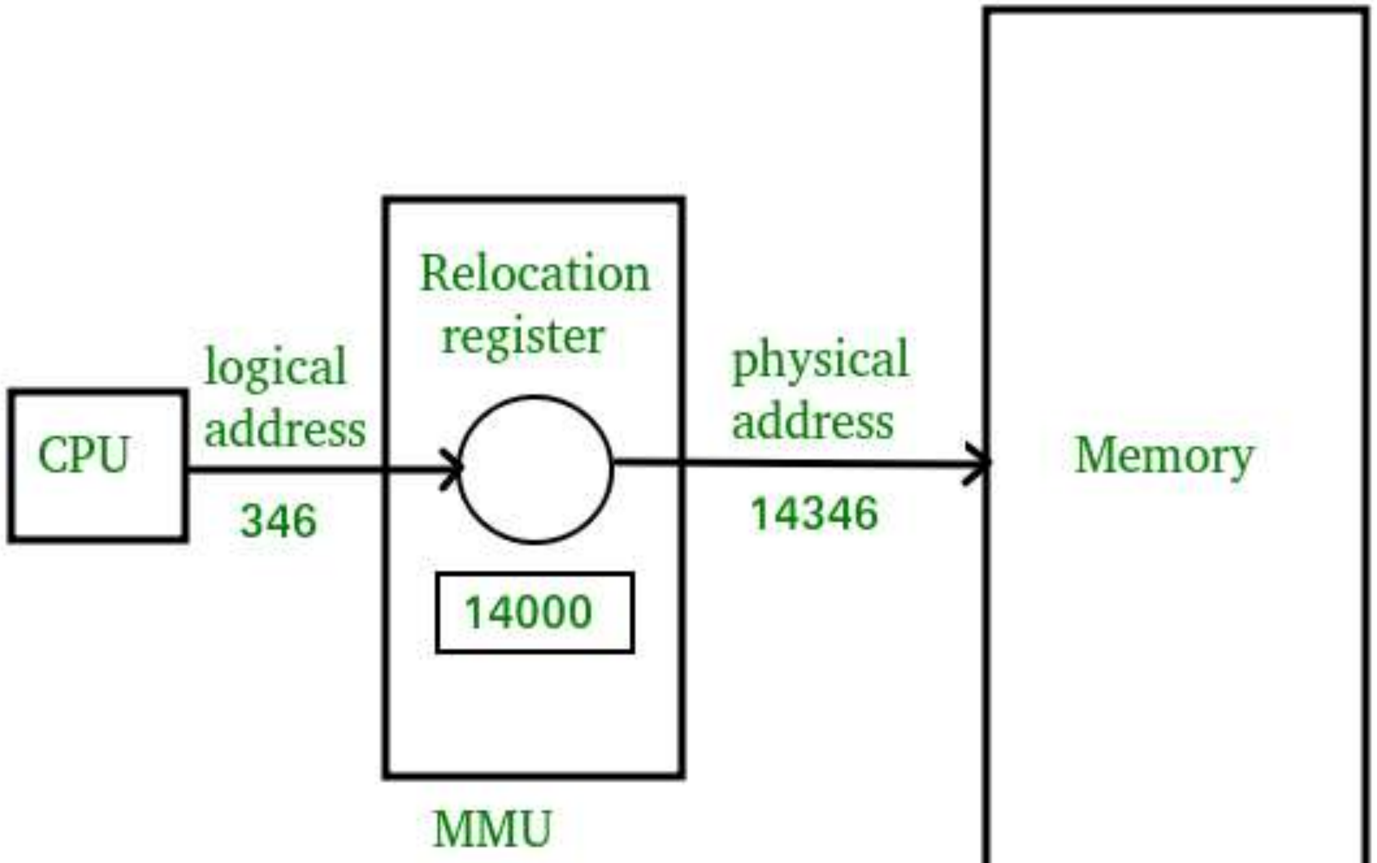
- A logical address is an address generated by the CPU during program execution, and is relative to the program’s address space.
- It is also known as a Virtual address.
- It is the address seen by the process and is used to access memory.
- The logical address may be different from the physical address due to the operation of an address translator or mapping function.
- The logical address is mapped to the physical address by the Memory Management Unit (MMU).
- The logical address provides a sort of memory protection.

Logical and Physical Address Space

Physical Address Space

- A physical address is the actual address in main memory where data is stored.
- A Physical address is also known as a Real address.
- The set of all physical addresses corresponding to these logical addresses is known as Physical address space.

Logical and Physical Address Space



Static and Dynamic Loading

Loading a process into the main memory is done by a loader. There are two different types of loading :

- **Static Loading:** Static Loading is basically loading the entire program into a fixed address. It requires more memory space.
- **Dynamic Loading:** To gain proper memory utilization, dynamic loading is used. In dynamic loading, a routine(sequence of code) is not loaded until it is called. All routines are residing on disk in a relocatable load format. One of the advantages of dynamic loading is that the unused routine is never loaded.

Static and Dynamic Linking

To perform a linking task a linker is used. A linker is a program that takes one or more object files generated by a compiler and combines them into a single executable file.

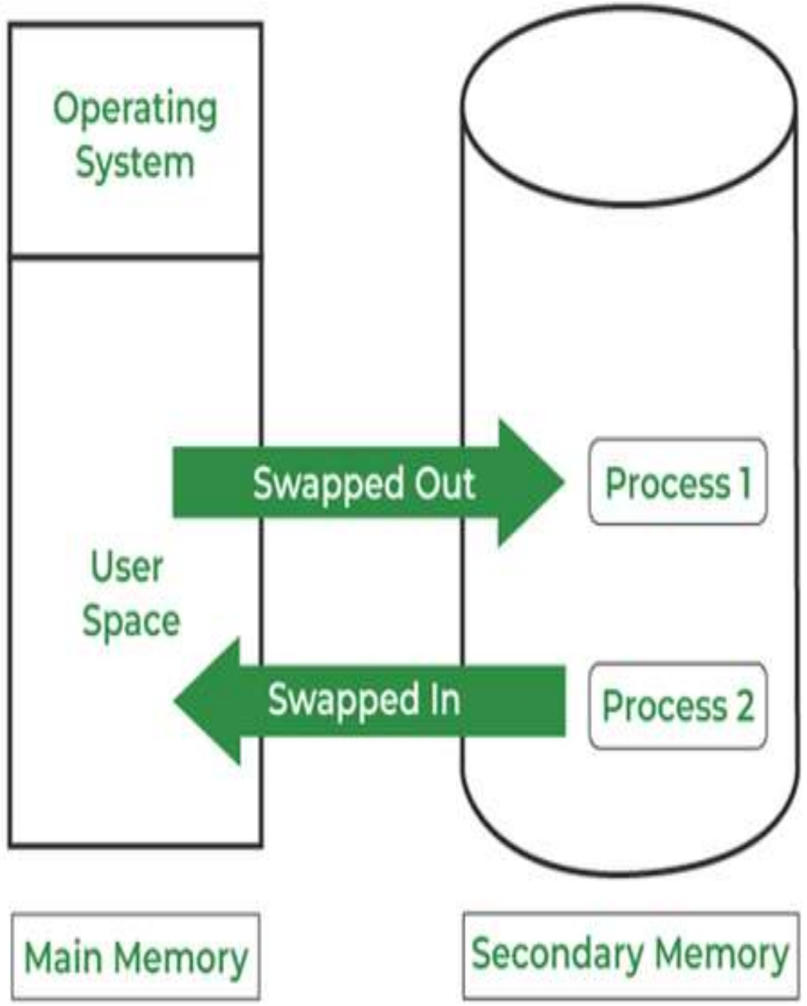
- **Static Linking:** In static linking, the linker combines all necessary program modules into a single executable program. So there is no runtime dependency. Some operating systems support only static linking, in which system language libraries are treated like any other object module.
- **Dynamic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

Swapping

- To increase CPU utilization in multiprogramming, a memory management scheme known as swapping can be used.
- Swapping is the process of bringing a process into memory and then temporarily copying it to the disc after it has run for a while.
- The purpose of swapping in an operating system is to access data on a hard disc and move it to RAM so that application programs can use it.
- It’s important to remember that swapping is only used when data isn’t available in RAM. Although the swapping process degrades system performance, it allows larger and multiple processes to run concurrently.
- Because of this, swapping is also known as memory compaction. The CPU scheduler determines which processes are swapped in and which are swapped out.

Swapping

- Consider a multiprogramming environment that employs a priority-based scheduling algorithm, When a high-priority process enters the input queue, a low-priority process is swapped out so the high-priority process can be loaded and executed.
- When this process terminates, the low priority process is swapped back into memory to continue its execution.
- Below figure shows the swapping process in operating system:



Memory Allocation

Introduction

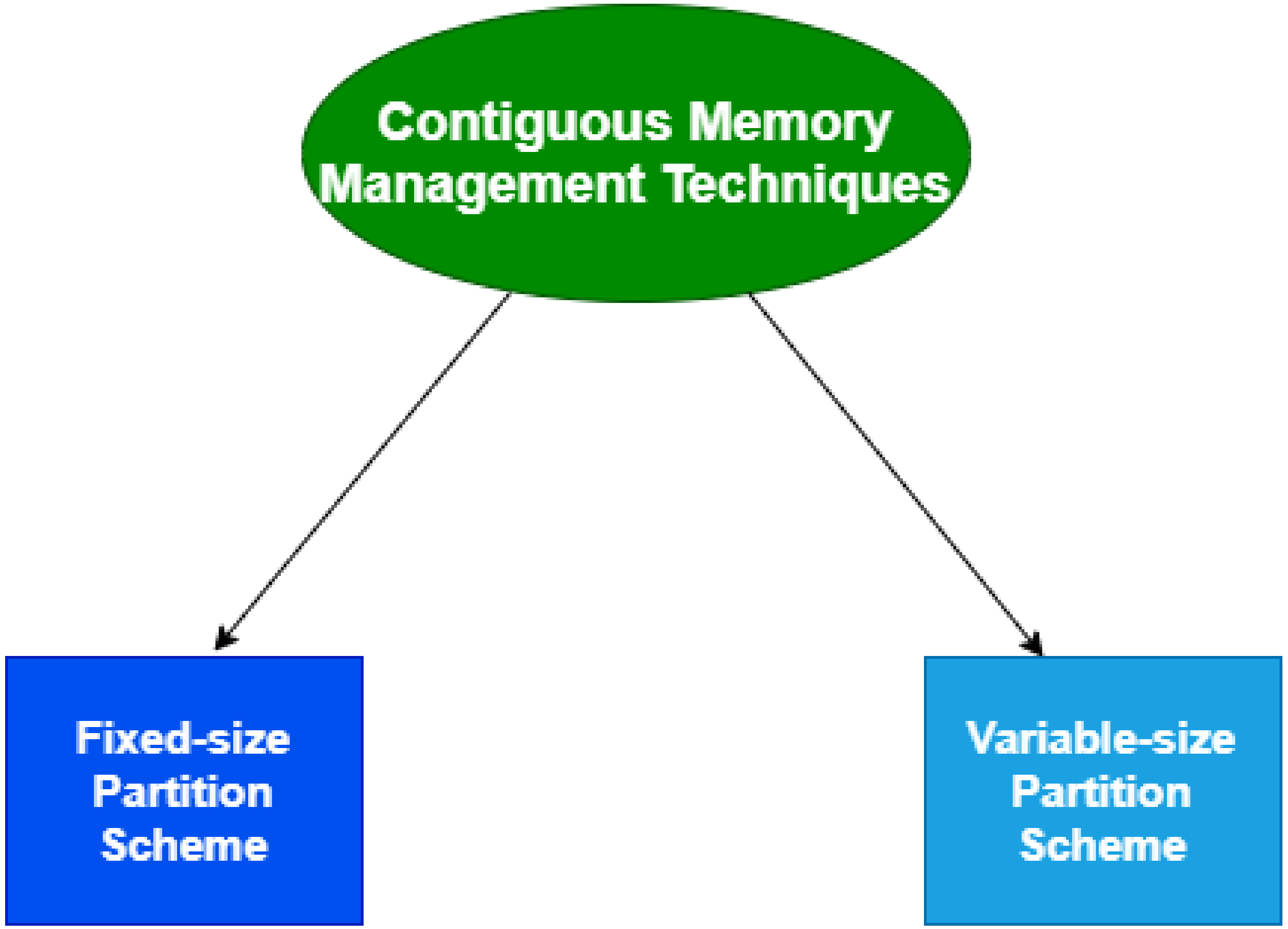
- To store the data and to manage the processes, we need a large-sized memory and, at the same time, we need to access the data as fast as possible.
- But if we increase the size of memory, the access time will also increase and, as we know, the CPU always generates addresses for secondary memory, i.e. logical addresses.
- But we want to access the main memory, so we need Address translation of logical address into physical address.
  The main memory interacts with both the user processes and the operating system.
- So we need to efficiently use the main memory.Main memory is divided into non-overlapping memory regions called partitions.

The main memory allocation can be broadly allocated in two ways –

- Contiguous memory allocation
- Non-Contiguous memory allocation

Contiguous Memory Allocation

- The main memory should accommodate both the operating system and the different client processes.
- **Contiguous memory allocation is a memory management technique used by operating systems to allocate and manage a computer's memory in a continuous or contiguous block.**
- In this allocation scheme, the main memory is divided into fixed-size or variable-size partitions, and processes are loaded into these partitions. Each process is given a contiguous block of memory, meaning that all the memory addresses occupied by the process are consecutive and do not have any gaps or interruptions.


Contiguous Memory Management Techniques
Fixed-size Partition Scheme
Variable-size Partition Scheme

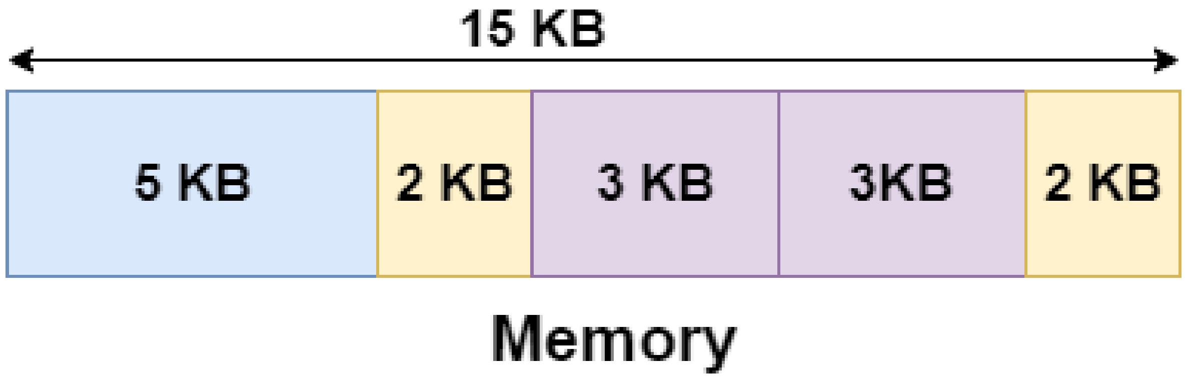
Fixed-size/Static partition scheme

- This technique is also known as **Static partitioning.**
- In this scheme, the system divides the memory into fixed-size partitions.
- The partitions may or may not be the same size.
- The size of each partition is fixed as indicated by the name of the technique and it cannot be changed.
- In this partition scheme, each partition may contain exactly one process. There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.
- Whenever any process terminates then the partition becomes available for another process.

Fixed-size partition scheme

Example

Let's take an example of fixed size partitioning scheme, we will divide a memory size of 15 KB into fixed-size partitions:



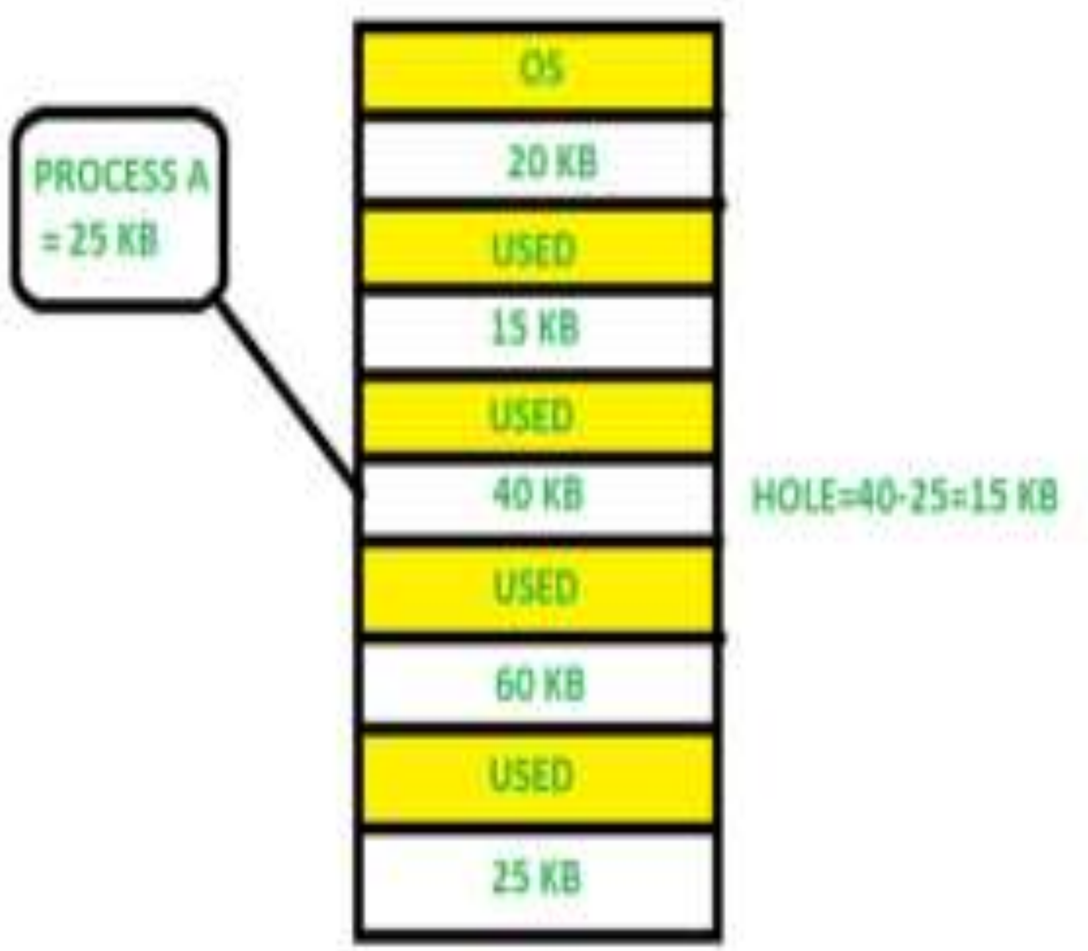
It is important to note that these partitions are allocated to the processes as they arrive and the partition that is allocated to the arrived process basically depends on the algorithm followed.

Partition Allocation methods

- First Fit
- Best Fit
- Worst Fit
- Next Fit

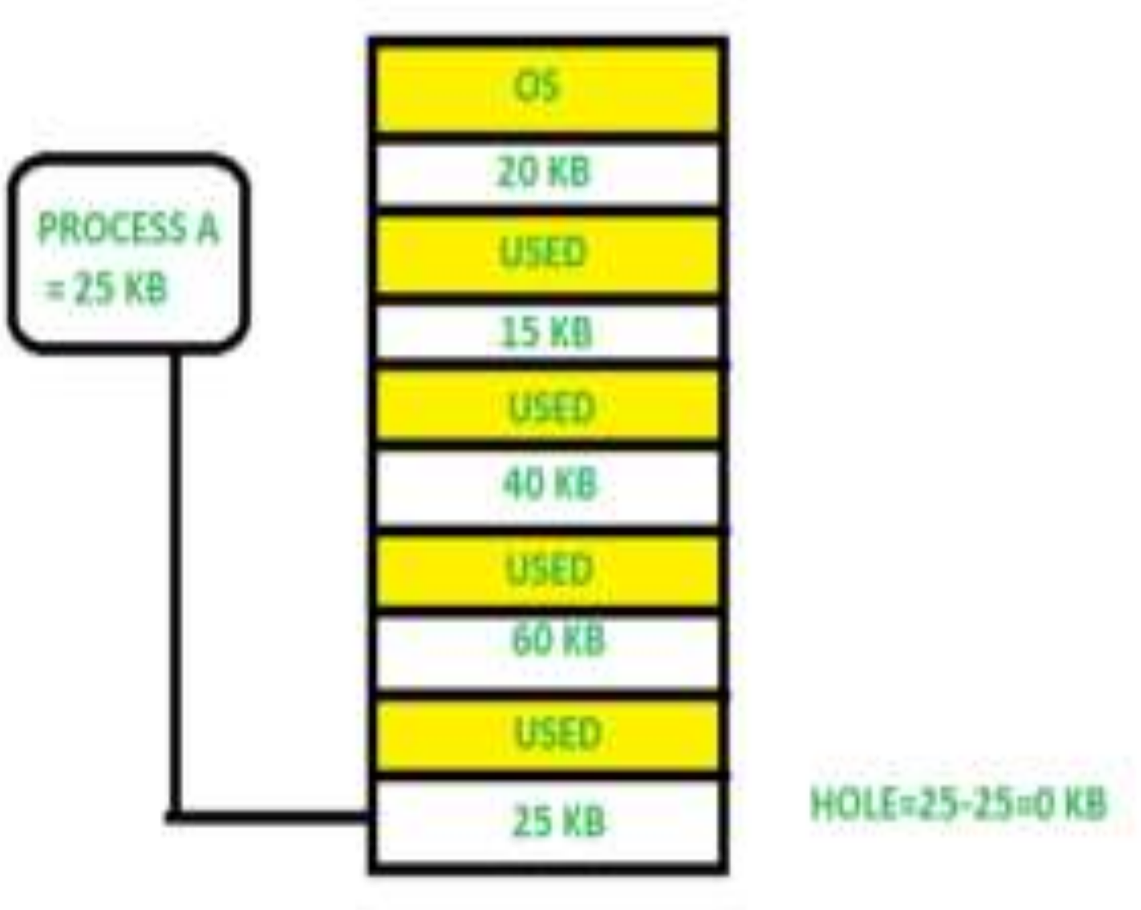
First Fit

- In the first fit, the partition is allocated which is the first sufficient block from the top of main memory.
- It scans memory from the beginning and chooses the first available block that is large enough.
- Thus it allocates the first hole that is large enough.



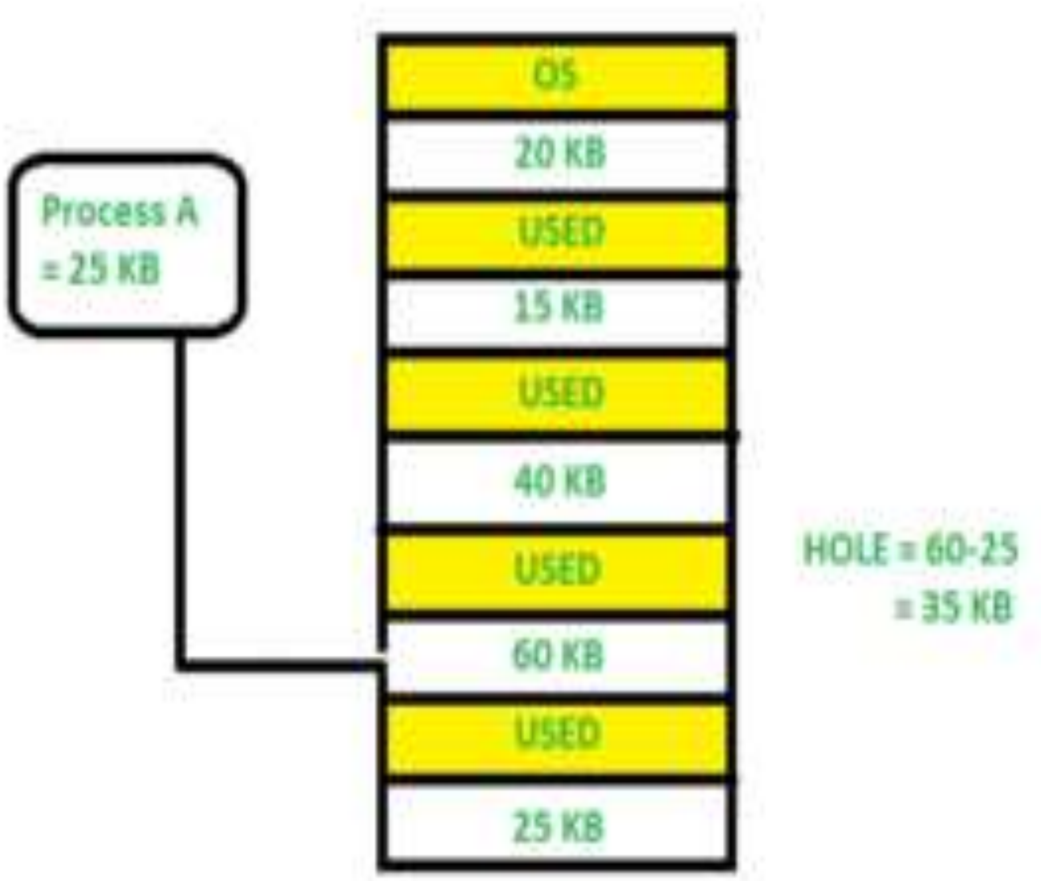
Best Fit

- Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.
- It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.



Worst Fit

- Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory.
- It is opposite to the best-fit algorithm.
- It searches the entire list of holes to find the largest hole and allocate it to process.



Next Fit

- Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.

Advantages of Fixed-size Partition Scheme

- This scheme is simple and is easy to implement
- It supports multiprogramming as multiple processes can be stored inside the main memory.
- Management is easy using this scheme

Disadvantages of Fixed-size Partition Scheme

Some disadvantages of using this scheme are as follows:

1. Internal Fragmentation

- Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as Internal fragmentation
- As we have shown in the above diagram the 70 KB partition is used to load a process of 50 KB so the remaining 20 KB got wasted.

2. Limitation on the size of the process

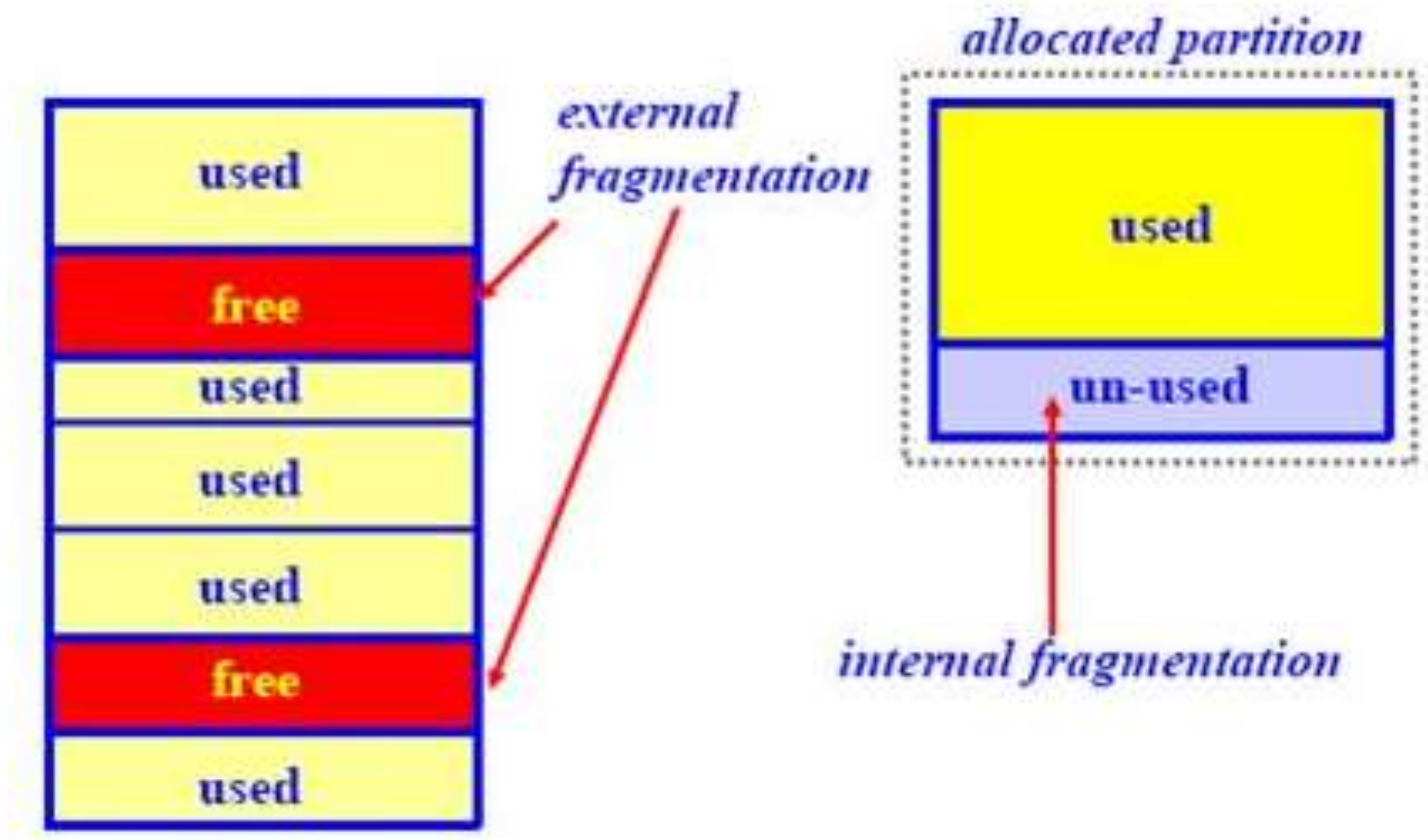
- If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory.
- Due to this, a condition is imposed on the size of the process and it is: the size of the process cannot be larger than the size of the largest partition.

3. External Fragmentation

- It is another drawback of the fixed-size partition scheme as total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.

4. Degree of multiprogramming is less

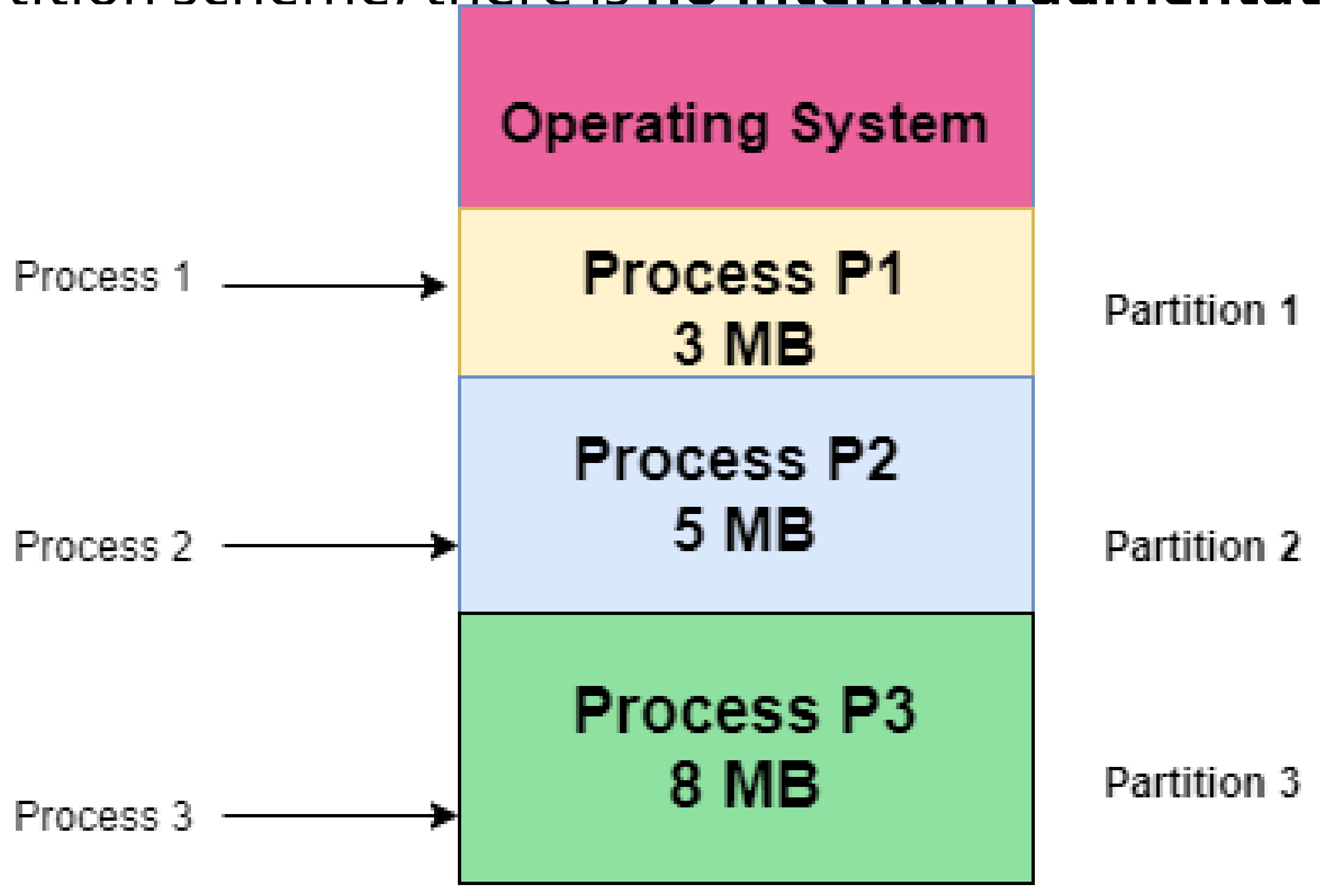
- In this partition scheme, as the size of the partition cannot change according to the size of the process.
- Thus the degree of multiprogramming is very less and is fixed.



Variable-size/DynamicPartition Scheme

- This scheme is also known as Dynamic partitioning and is came into existence to overcome the drawback i.e internal fragmentation that is caused by Static partitioning.
- In this partitioning, size allocation is done dynamically.
- The size of the partition is not declared initially.
- Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process.
- Thus the size of each partition is equal to the size of the process.

- As partition size varies according to the need of the process, so in this partition scheme, there is **no internal fragmentation**.



Size of Partition =Size of Process

Advantages of Variable-size Partition Scheme

1. **No Internal Fragmentation**
   As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
2. **Degree of Multiprogramming is Dynamic**
   As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
3. **No Limitation on the Size of Process**
   In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

Disadvantages of Variable-size Partition Scheme

1. **External Fragmentation**
   As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will no external fragmentation. Let us understand this with the help of an example: In the above diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let's there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.

2. Difficult Implementation

The implementation of this partition scheme is difficult as compared to the Fixed Partitioning scheme as it involves the allocation of memory at run-time rather than during the system configuration. As we know that OS keeps the track of all the partitions but here allocation and deallocation are done very frequently and partition size will be changed at each time so it will be difficult for the operating system to manage everything.

Problem-01:

Consider six memory partitions of size 200 KB, 400 KB, 600 KB, 500 KB, 300 KB and 250 KB. These partitions need to be allocated to four processes of sizes 357 KB, 210 KB, 468 KB and 491 KB in that order.

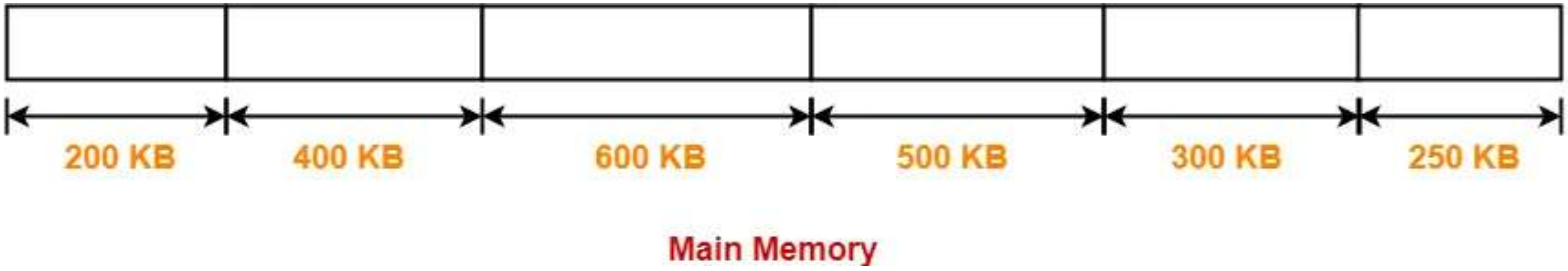
Perform the allocation of processes using-

1. First Fit Algorithm
2. Best Fit Algorithm
3. Worst Fit Algorithm

Solution-01:

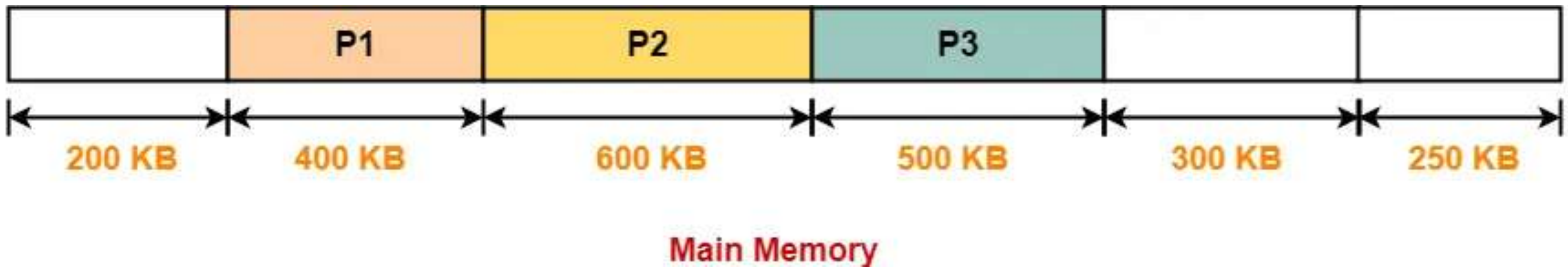
Let us say the given processes are-

- Process P1 = 357 KB
- Process P2 = 210 KB
- Process P3 = 468 KB
- Process P4 = 491 KB



Solution-01:

First Fit

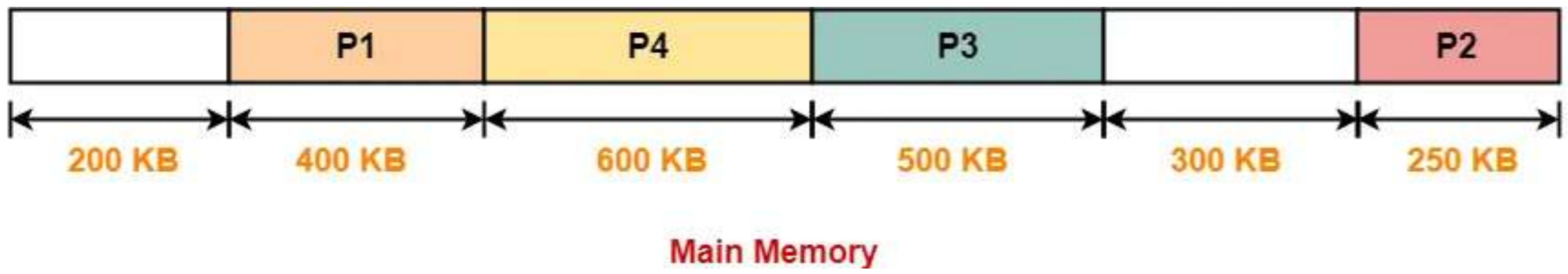


Process P4 can not be allocated the memory.

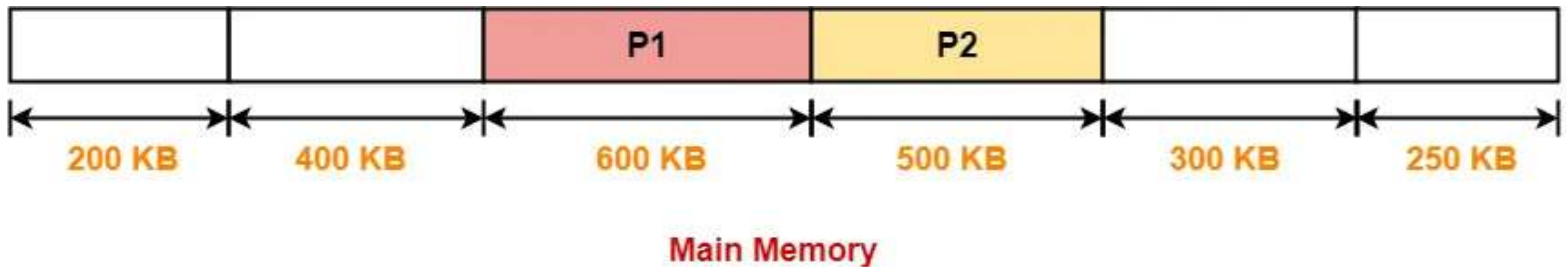
This is because no partition of size greater than or equal to the size of process P4 is available.

Solution-01:

Best Fit



Worst Fit



Non-Contiguous Memory Allocation

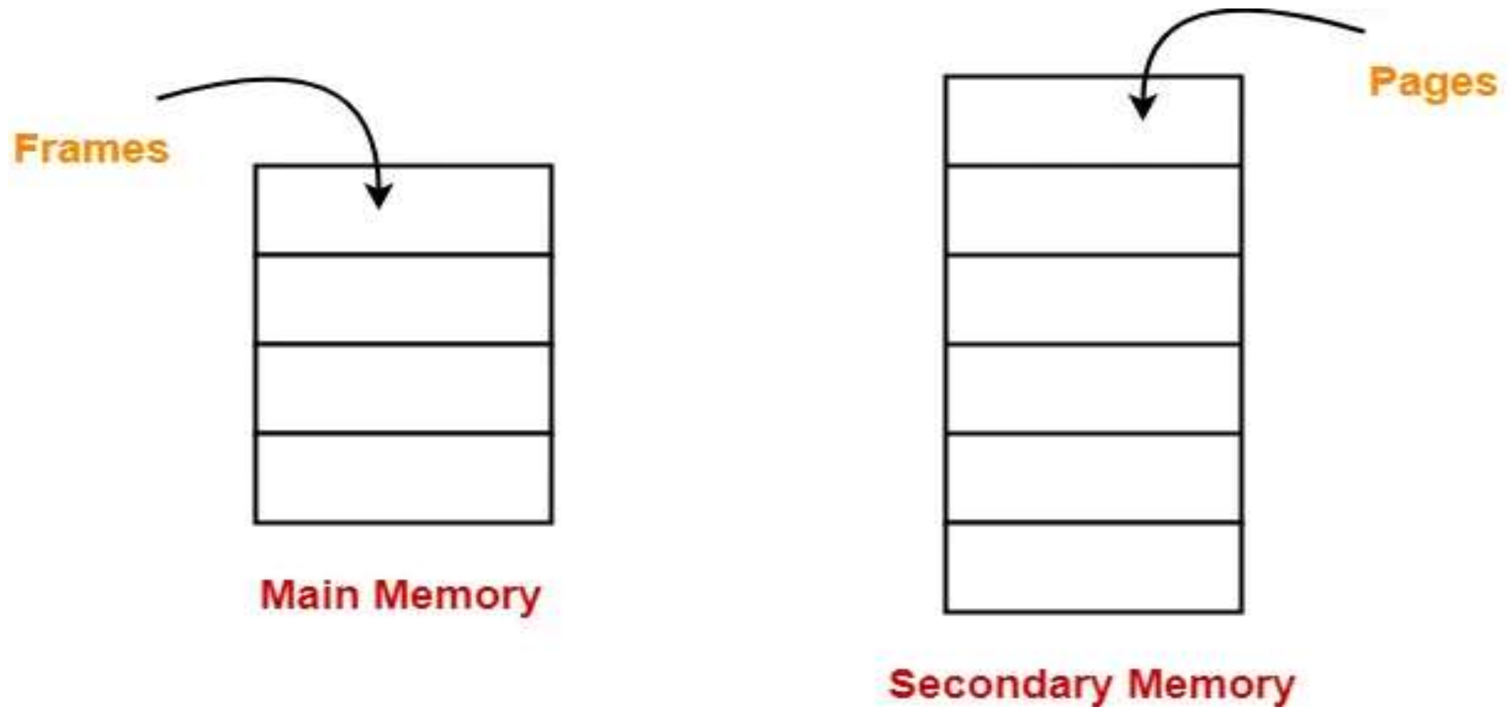
- Non-contiguous memory allocation is a memory allocation technique.

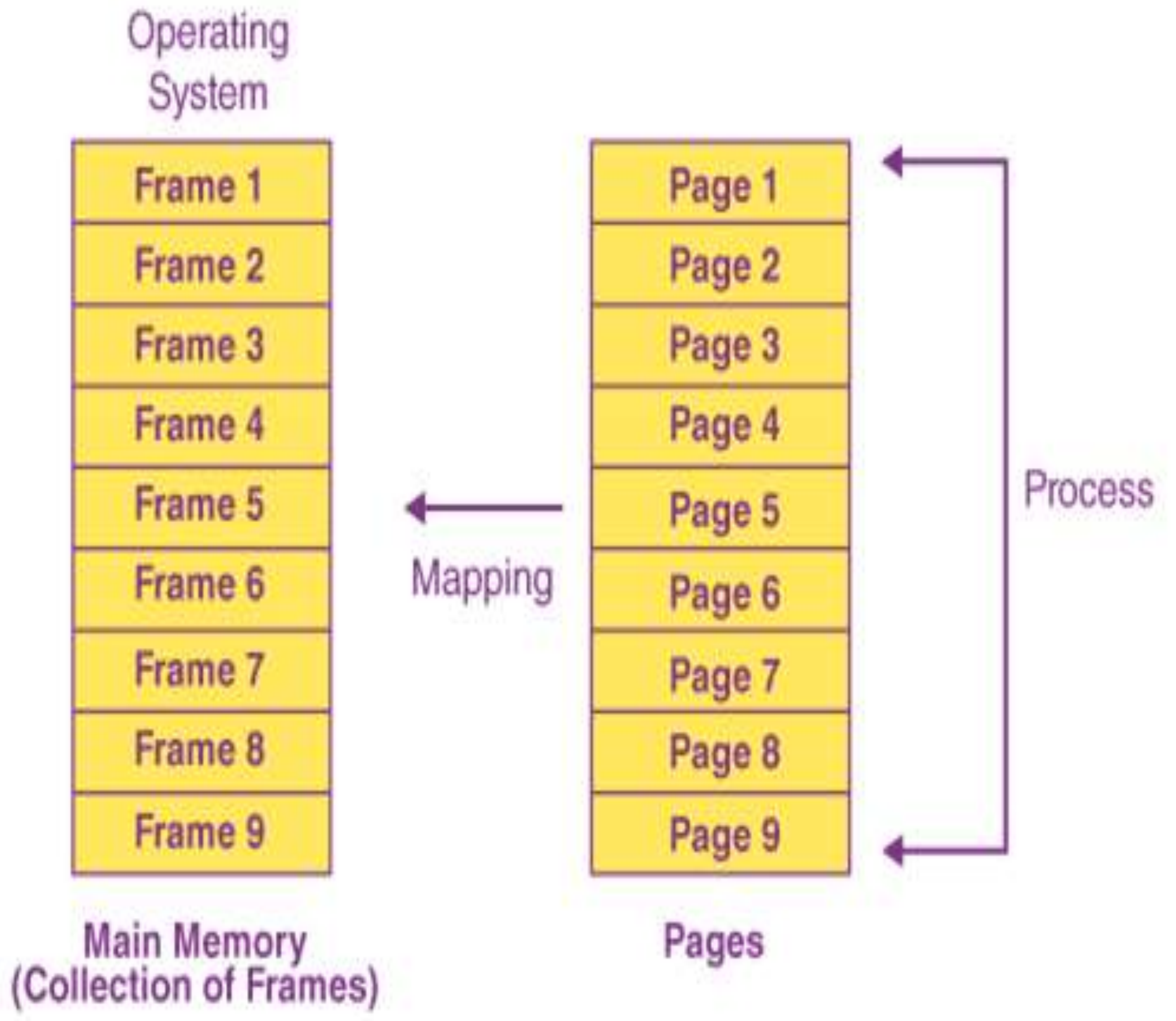
- It allows to store parts of a single process in a non-contiguous fashion.
- Thus, different parts of the same process can be stored at different places i the main memory.

- **Non-Contiguous memory allocation can be categorized into:**
  - **Paging**
  - **Segmentation**

Paging

- 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.



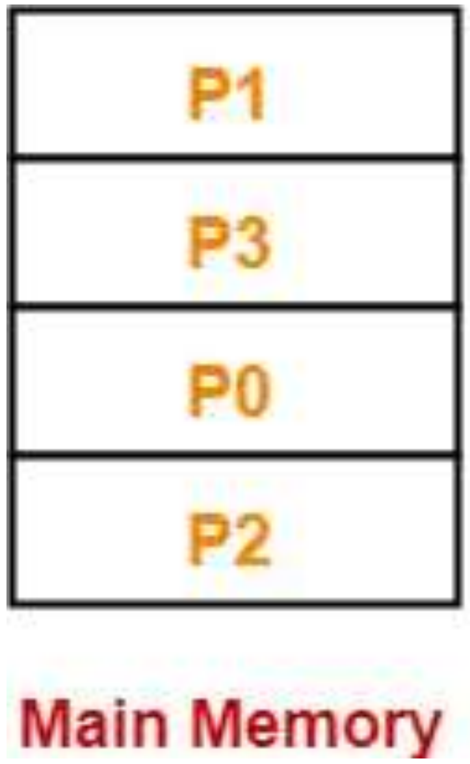


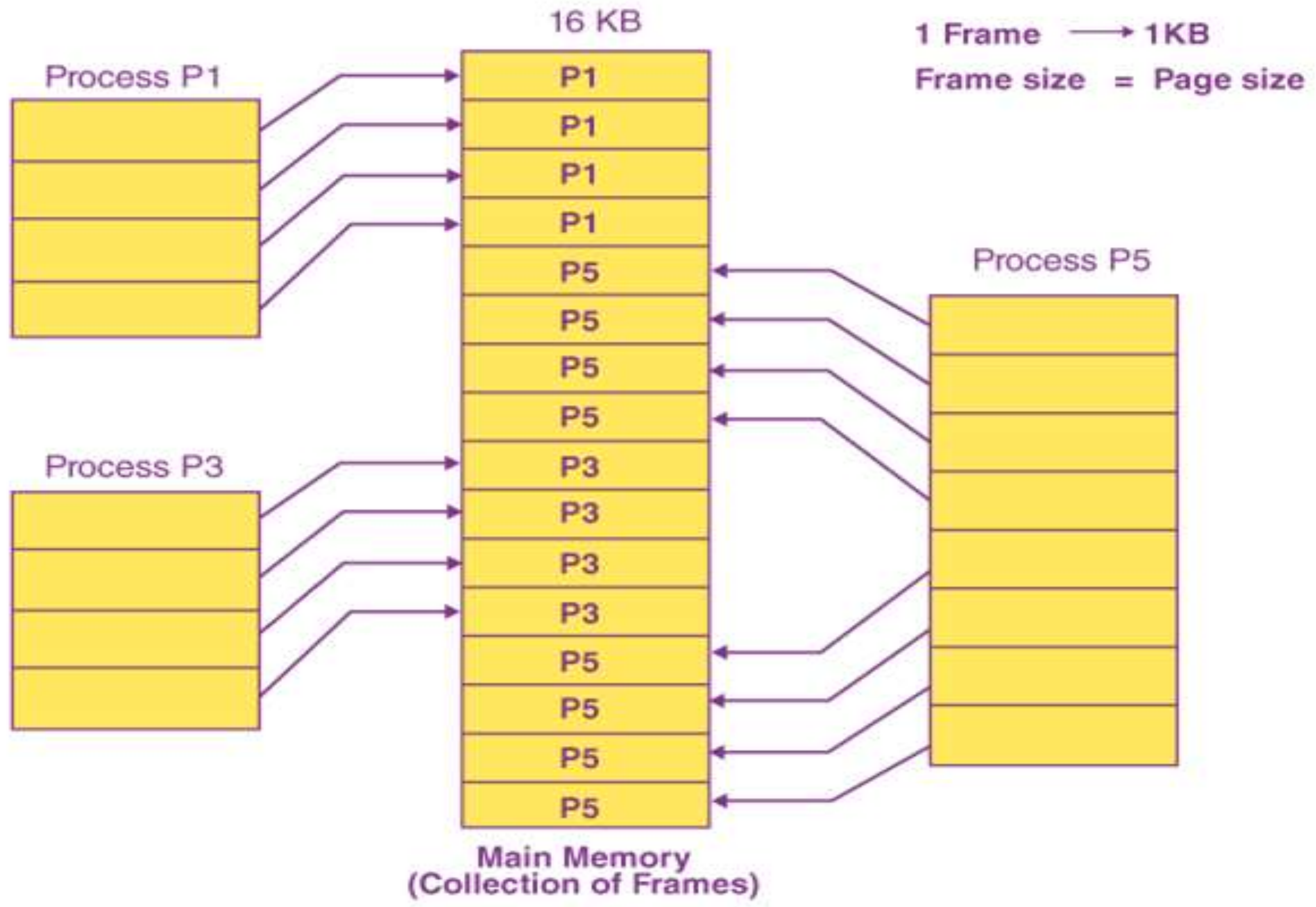
- The frame sizes may vary depending on the OS.
- Each frame must be of the same size.
- **Since the pages present in paging are mapped on to the frames, the page size should be similar to the frame size.**

- 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

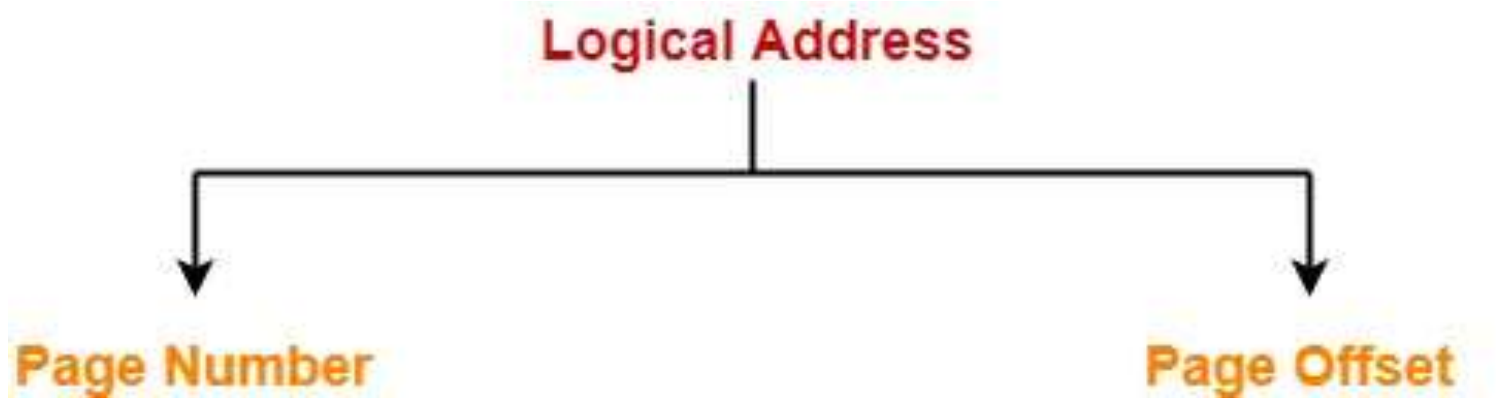
- Consider a process is divided into 4 pages P0, P1, P2 and P3.
- Depending upon the availability, these pages may be stored in the main memory frames in a non-contiguous fashion as shown-




16 KB
1 Frame → 1KB
Frame size = Page size
Process P1
Process P3
Process P5
P1
P1
P1
P1
P5
P5
P5
P5
P3
P3
P3
P3
P5
P5
P5
P5
Main Memory
(Collection of Frames)

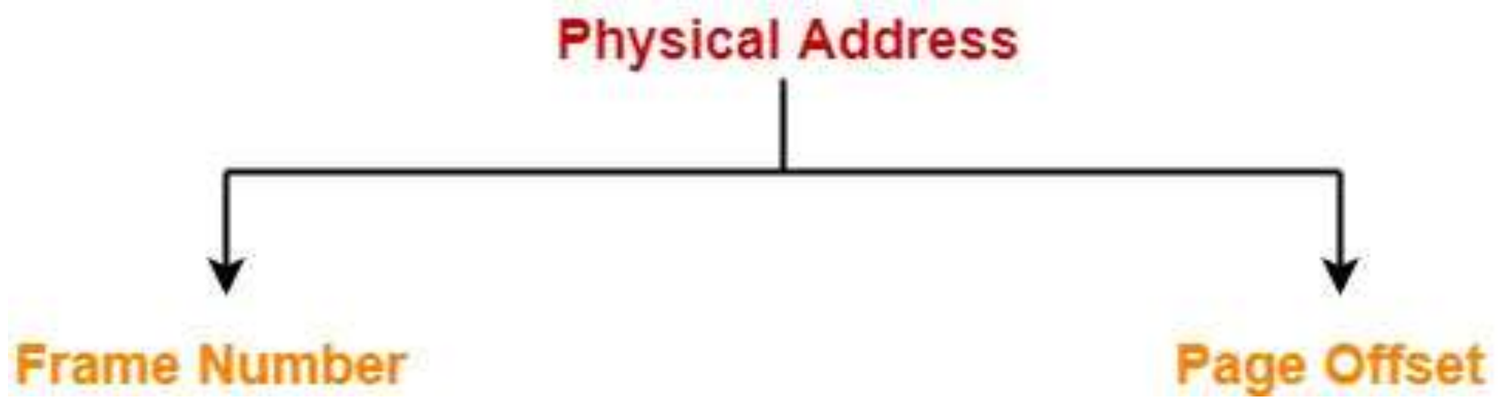
Paging

Translating Logical Address into Physical Address (Address Translation)



- 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.

Translating Logical Address into Physical Address (Address Translation)



- 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.

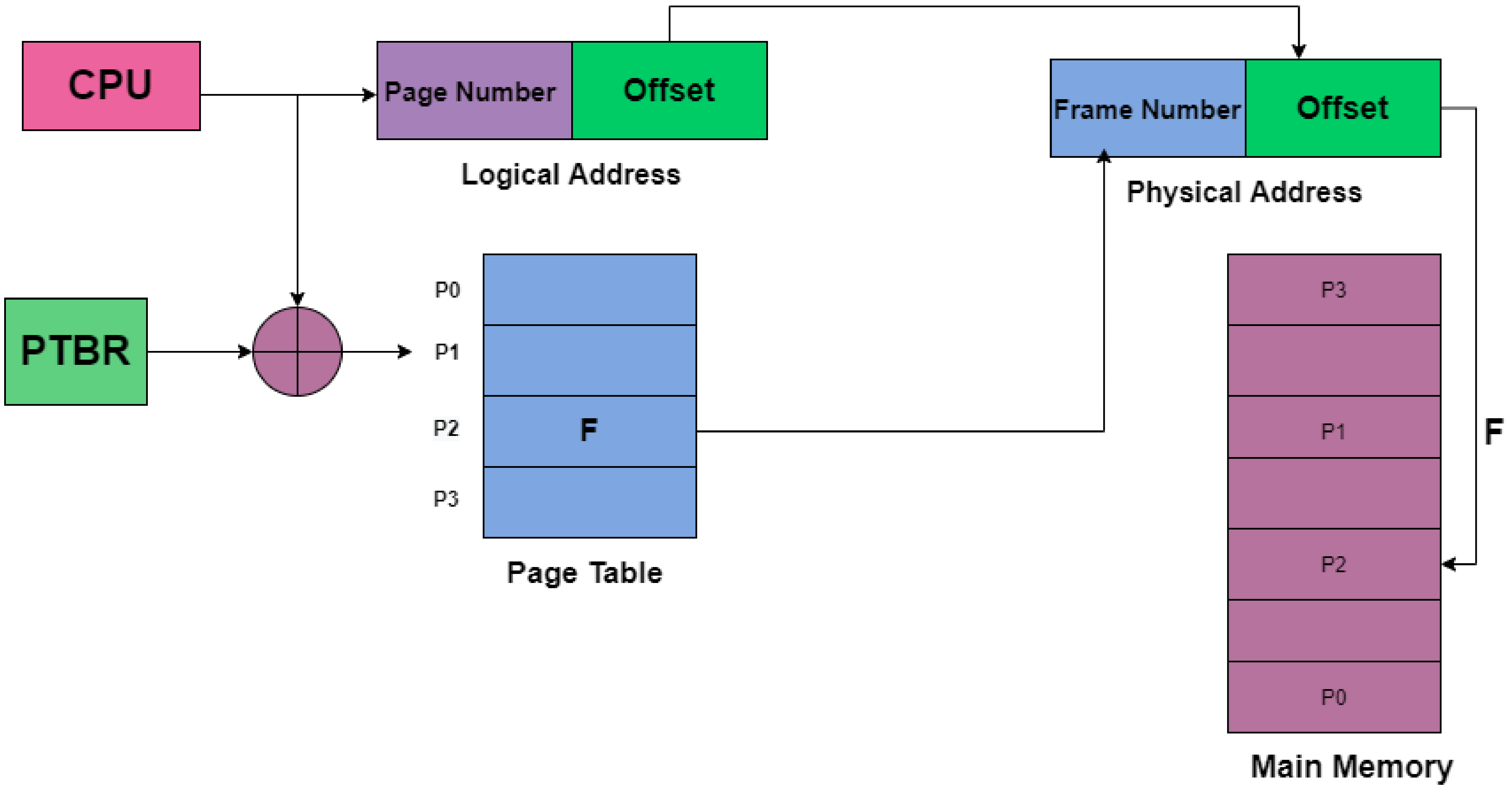
Page Table

- The data structure that is used by the virtual memory system in the operating system of a computer in order to store the mapping between physical and logical addresses is commonly known as Page Table.
- The Page table mainly contains the base address of each page in the Physical memory. The base address is then combined with the page offset in order to define the physical memory address which is then sent to the memory unit.
- **Thus page table mainly provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.**

Page Table

- Page table is stored in the main memory.
- **Each process has its own independent page table.**
- Number of entries in a page table = Number of pages in which the process is divided.
- **Page Table Base Register (PTBR) contains the base address of page table of current process.**

Mapping logical address to physical address

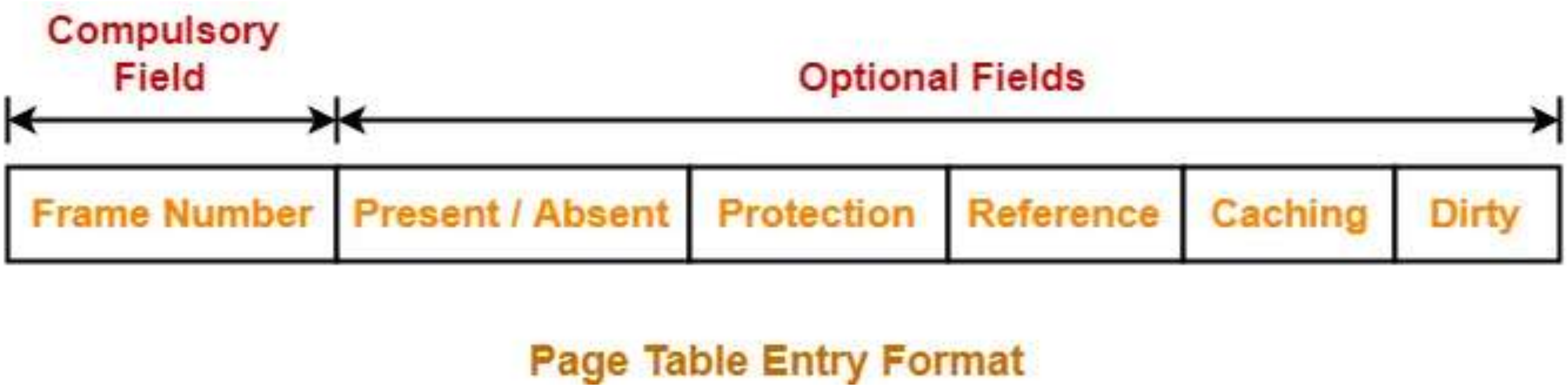


- The base address of the page table is added with the page number referenced by the CPU.
- It gives the entry of the page table containing the frame number where the referenced page is stored.

Page Table Entry

- A page table entry contains several information about the page.
- The information contained in the page table entry varies from operating system to operating system.
- **The most important information in a page table entry is frame number.**

- In general, each entry of a page table contains the following information-



Page Table Entry Format

Frame Number

- Frame number specifies the frame where the page is stored in the main memory.
- The number of bits in frame number depends on the number of frames in the main memory

Present / Absent Bit-

- This bit is also sometimes called as valid / invalid bit.
- This bit specifies whether that page is present in the main memory or not.
- If the page is not present in the main memory, then this bit is set to 0 otherwise set to 1.

- If the required page is not present in the main memory, then it is called as **Page Fault.**
- A page fault requires page initialization.
- The required page has to be initialized (fetched) from the secondary memory and brought into the main memory.

Protection Bit

- This bit is also sometimes called as "Read / Write bit".
- This bit is concerned with the page protection.
- It specifies the permission to perform read and write operation on the page and represented as 0 or 1.

Reference bit

- Reference bit specifies whether that page has been referenced in the last clock cycle or not.
- If the page has been referenced recently, then this bit is set to 1 otherwise set to 0.

Caching Enabled / Disabled

- This bit enables or disables the caching of page.
- Whenever freshness in the data is required, then caching is disabled using this bit.
- If caching of the page is disabled, then this bit is set to 1 otherwise set to 0.

Dirty Bit

- This bit is also sometimes called as “Modified bit“.
- This bit specifies whether that page has been modified or not.
- If the page has been modified, then this bit is set to 1 otherwise set to 0.

- Physical Address space = Size of Main Memory.
- Page size= frame size, should be represented as power of 2.
- **Number of frames= Physical Address space / frame size**

Example 1:

If the physical address space is 64 bytes and the frame size is 8 bytes, What is the number of frames required?

Answer

Number of frames = $2^6 / 2^3$

$= 2^3$

Example 2:

If the physical address space is 64 MB and the frame size is 16 KB, What is the number of frames required?

Answer

1 MB = 2^{20} Bytes

1 KB = 2^{10} Bytes

Physical address space is 64 MB = 64 X 2^{20} Bytes

= 2^{6} X

2^{20} Bytes

= 2^{26}

Bytes

Frame size 16 KB = 16 X 2^{10} Bytes

= 2^{4} X 2^{10} Bytes

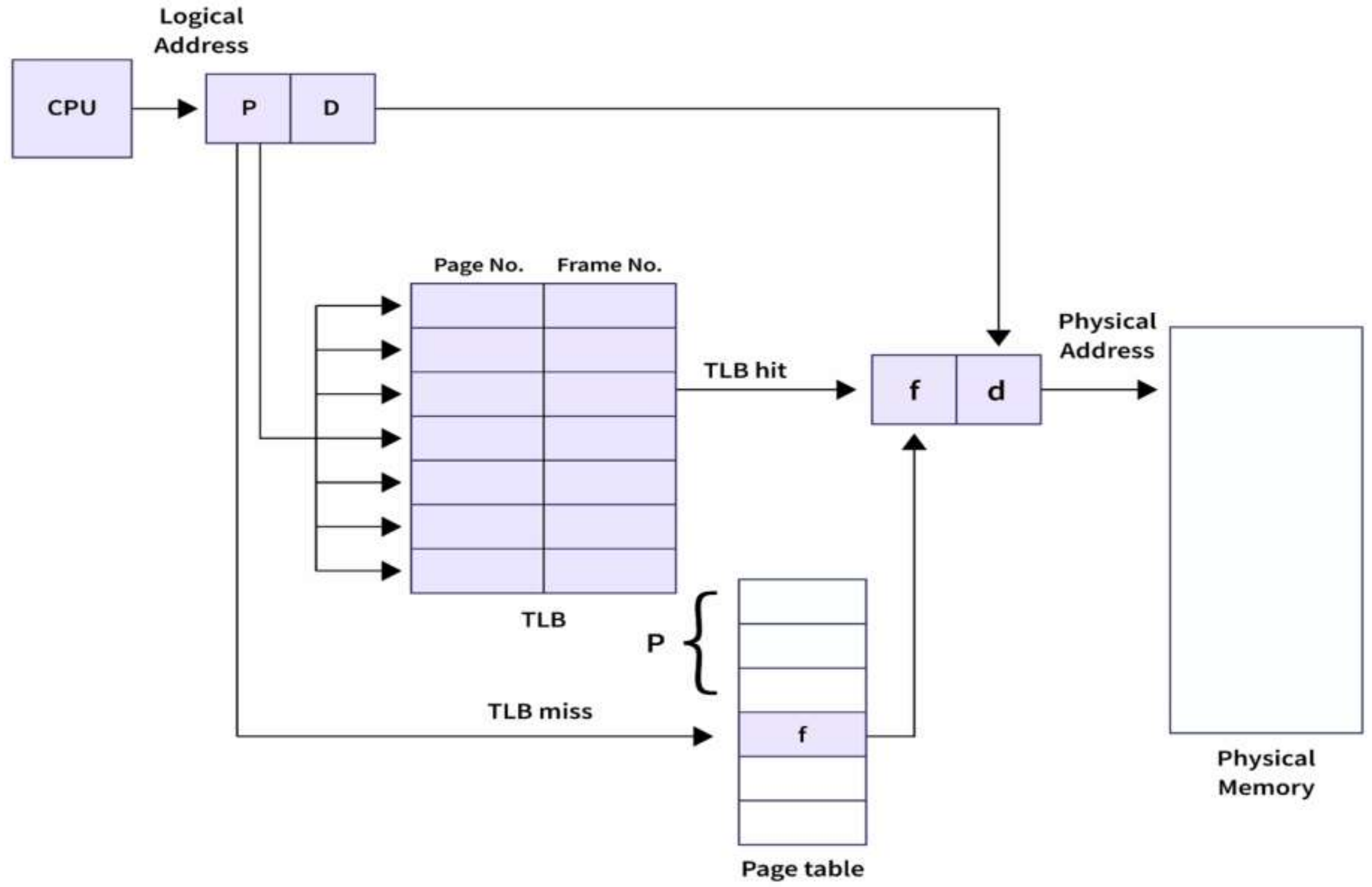
= 2^{14} Bytes

Translation Lookaside Buffer (TLB)

- A translation lookaside buffer (TLB) is a type of memory cache that stores recent translations of virtual memory to physical addresses to enable faster retrieval. This high-speed cache is set up to keep track of recently used page table entries (PTEs).
- Also known as an address-translation cache, a TLB is a part of the processor's memory management unit (MMU).
- The TLB checks if the page is already in the main memory. The processor examines the TLB for a PTE, retrieves the frame number and forms the real address.
- If the page is not in the main memory, a page fault is issued, and the TLB gets updated with the new PTE.

- This high-speed cache can keep track of recently used transactions through PTEs.
- This enables the processor to examine the TLB to confirm if a PTE is present, known as a **TLB hit**, and then retrieve the frame number and real address.
- If the PTE is not found in the TLB, the page number is used as an index, and the TLB gets updated with a new PTE.

Paging with TLB



TLB hit and TLB miss

A **TLB hit** means a PTE is present in the TLB and the processor has found it, given a virtual address. When this happens, the CPU accesses the actual location in the main memory. It consists of these steps:

- The CPU generates a virtual address. This is a logical address.
- The address is checked in the TLB and is present.
- If the address is present, its frame number is retrieved.
- The CPU now knows where the page lies in the main memory.

TLB hit and TLB miss

A **TLB miss** means a PTE was not found in the TLB. In this case, the page number is used as the index while processing the page table. In other words, the processor accesses the page table in the main memory and then accesses the actual frame in the main memory. The steps in a TLB miss are as follows:

- The CPU generates a virtual address. Again, this is a logical address.
- The address is checked in the TLB and is not present.
- The page number is matched to a page table residing in the main memory.
- The corresponding frame number is retrieved.
- The CPU now knows where the page lies in the main memory.
- The TLB is updated with the new PTE

Paging - Advantages

- Easy to use memory management algorithm
- No need for external Fragmentation
- Swapping is easy between equal-sized pages and page frames.

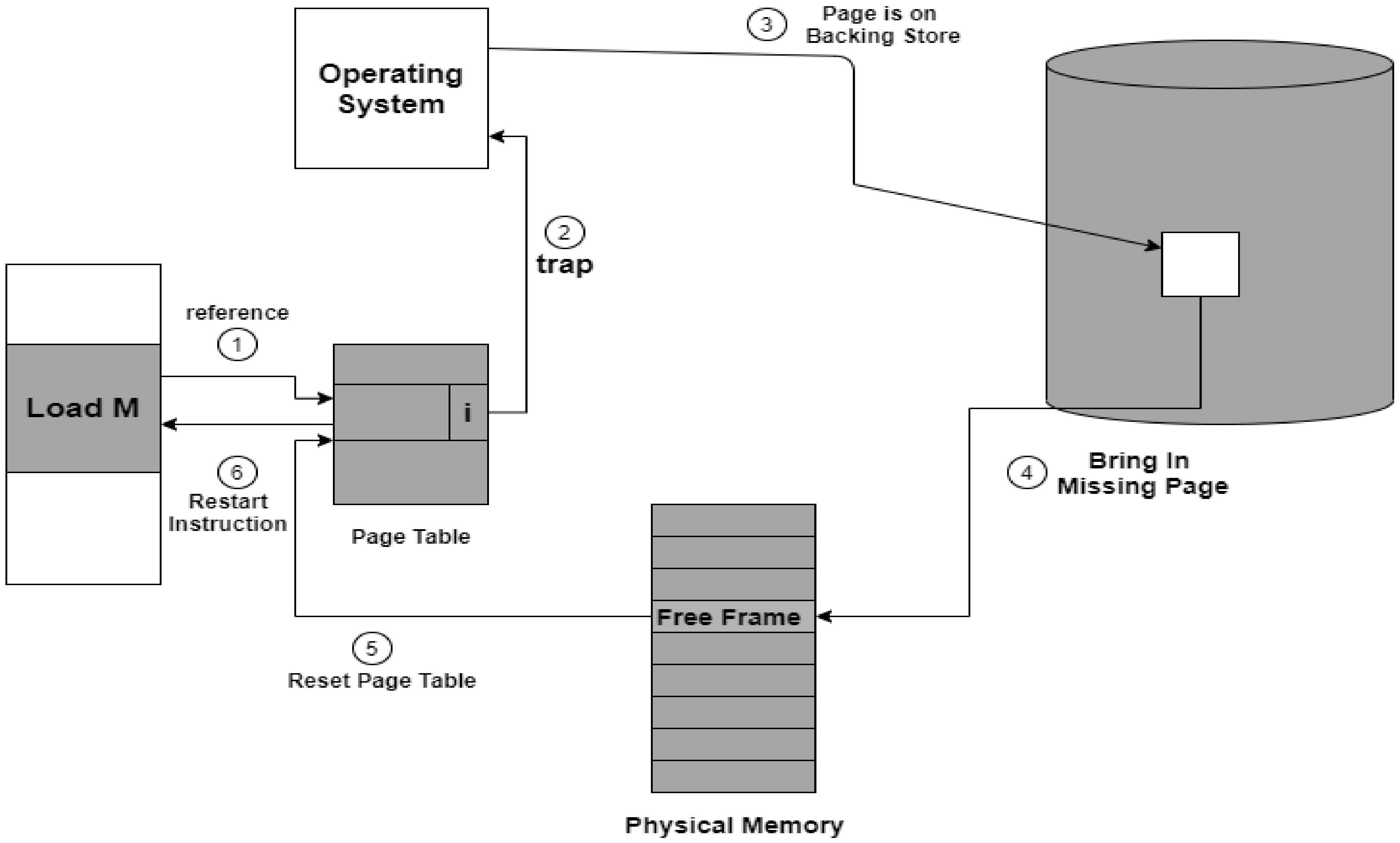
Paging - Disadvantages

- May cause **Internal fragmentation**
- Page tables consume additonal memory.
- Multi-level paging may lead to memory reference overhead.

Page Fault

- A page fault occurs when the referenced page is not found in the main memory.
- Page fault handling routine is executed on the occurrence of page fault.
- The time taken to service the page fault is called as **page fault service time.**
- Page faults dominate more like an error. A page fault will happen if a program tries to access a piece of memory that does not exist in physical memory (main memory).
- Whenever any page fault occurs, then the required page has to be fetched from the secondary memory into the main memory.

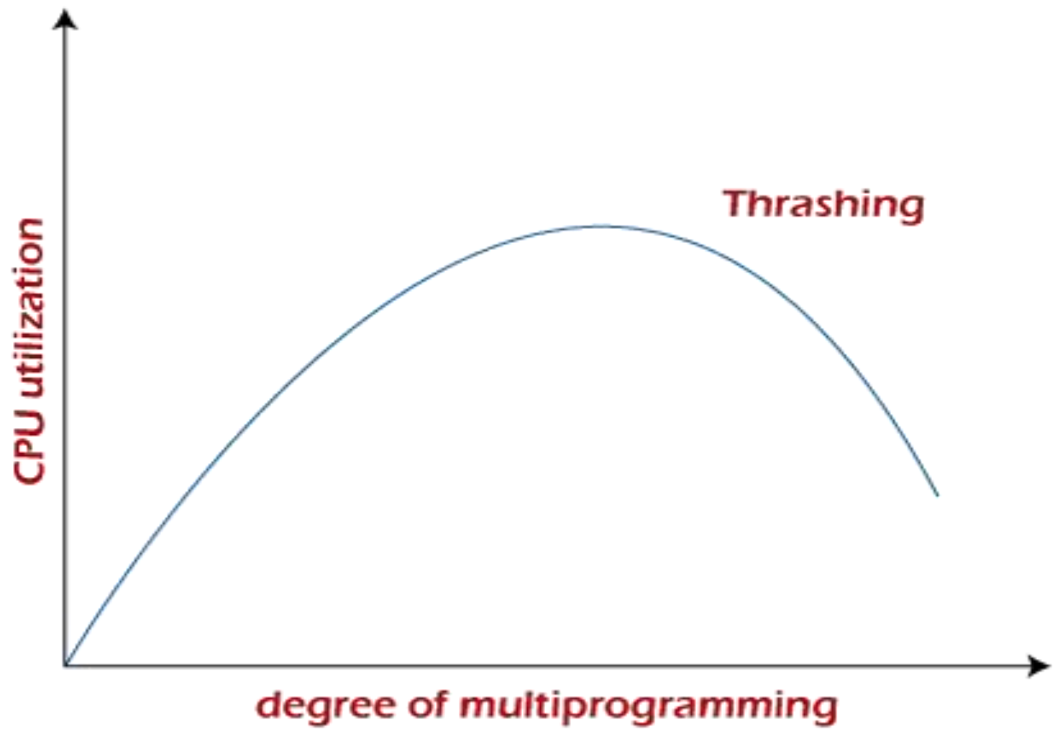
Handling Page Fault



- First of all, internal table(that is usually the process control block) for this process in order to determine whether the reference was valid or invalid memory access.
- If the reference is invalid, then we will terminate the process. If the reference is valid, but we have not bought in that page so now we just page it in.
- Then we locate the free frame list in order to find the free frame.
- Now a disk operation is scheduled in order to read the desired page into the newly allocated frame.
- When the disk is completely read, then the internal table is modified that is kept with the process, and the page table that mainly indicates the page is now in memory.
- Now we will restart the instruction that was interrupted due to the trap. Now the process can access the page as though it had always been in memory.

Thrashing

- Thrashing is when the page fault and swapping happens very frequently at a higher rate, and then the operating system has to spend more time swapping these pages.
- This state in the operating system is known as thrashing.
- Because of thrashing, the CPU utilization is going to be reduced or negligible.



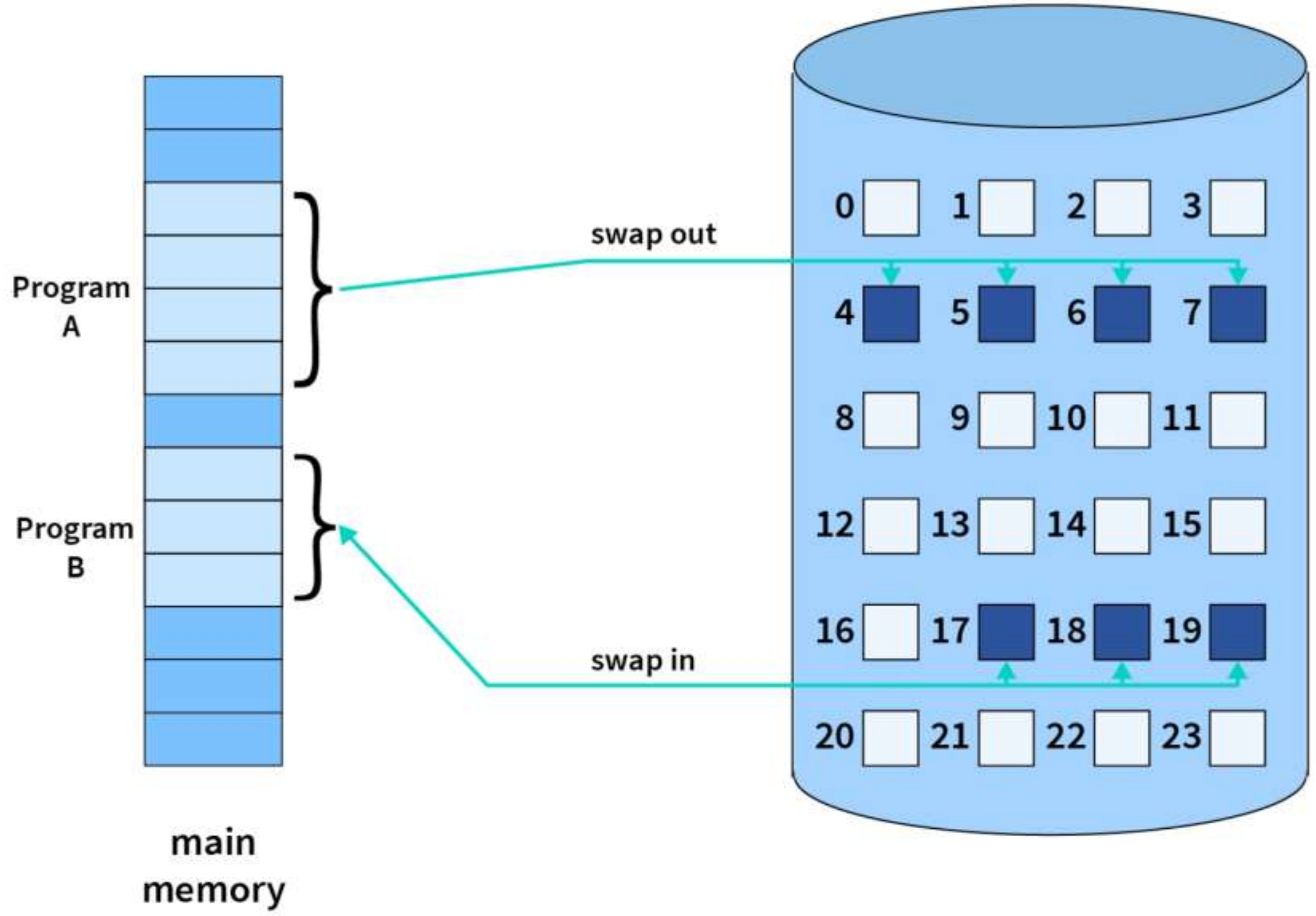
Virtual Memory

- Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory.
- This is done by treating a part of secondary memory as the main memory.
- In this scheme, User can load the bigger size processes than the available main memory by having the illusion that the memory is available to load the process.
- Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.
- By doing this, the degree of multiprogramming will be increased and therefore, the CPU utilization will also be increased.

Demand Paging

- Demand Paging is a popular method of virtual memory management. In demand paging, the pages of a process which are least used, get stored in the secondary memory.
- A page is copied to the main memory when its demand is made or page fault occurs.
- **There are various page replacement algorithms which are used to determine the pages which will be replaced.**

Demand Paging



Demand Paging

Advantages

- With the help of Demand Paging, memory is utilized efficiently.
- With this technique, portions of the process that are never called are never loaded.
- If, any program is larger to physical memory then It helps to run this program.

Disadvantages

- It has more probability of internal fragmentation.
- Its memory access time is longer.
- Page map table is needed additional memory and registers.

Page Replacement Algorithm

Introduction

- When Page Fault occurs, operating System might have to replace one of the existing pages with the newly needed page.
- Different page replacement algorithms suggest different ways to decide which page to replace.
- The target for all algorithms is to reduce the number of page faults.

Cache hit Ratio

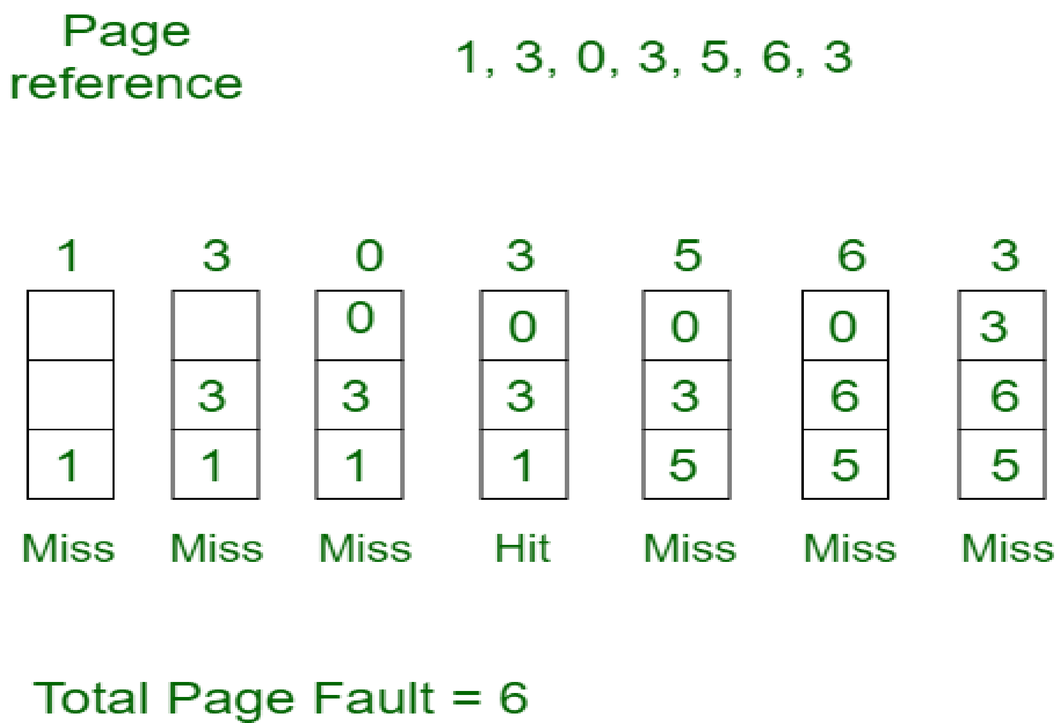
$$\text{Cache hit ratio} = \frac{\text{Number of cache hits}}{\text{Number of cache hits} + \text{Number of cache misses}}$$

Cache miss Ratio

$$\text{Cache miss ratio} = \frac{\text{Number of cache misses}}{\text{Number of cache hits} + \text{Number of cache misses}} = 1 - \text{Cache hit ratio}$$

First In First Out (FIFO)

- This is the simplest page replacement algorithm.
- In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue.
- When a page needs to be replaced page in the front of the queue is selected for removal.
- **Example:** Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3 page frames.Find the number of page faults.



Hit ratio= 14
Miss ratio = 86

First In First Out (FIFO)

- Belady’s anomaly may occur in FIFO algorithm.

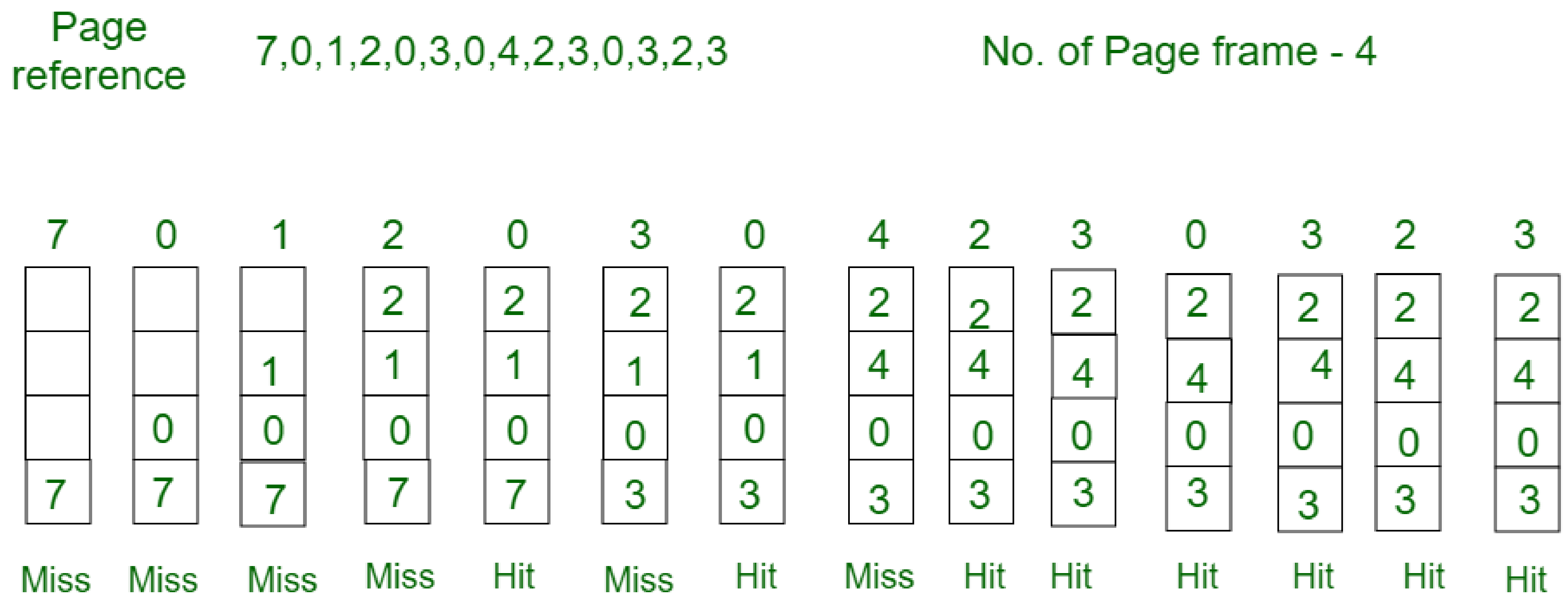
- Calculating the hit and miss ratio for the previous example :

Hit ratio= (1/7) * 100 = 14

Miss ratio=(6/7)* 100= 86

Optimal Page Replacement

- In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
- **Example:** Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frame. Find number of page fault.

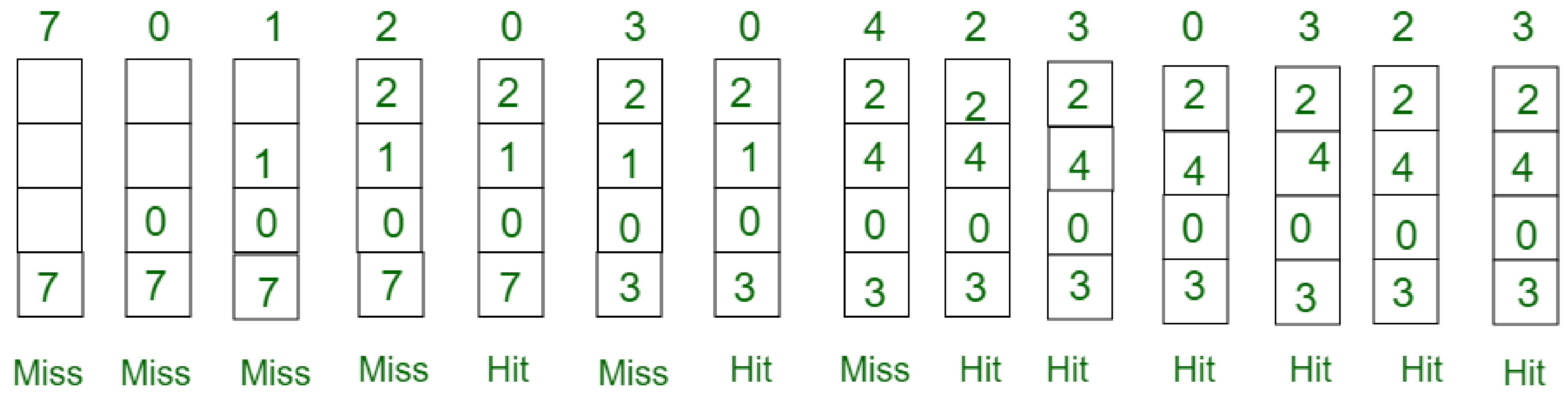


Total Page Fault = 6

Least Recently Used(LRU)

- In this algorithm, page will be replaced which is least recently used.
- **Example:** Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames. Find number of page faults.

Page reference   7,0,1,2,0,3,0,4,2,3,0,3,2,3   No. of Page frame - 4

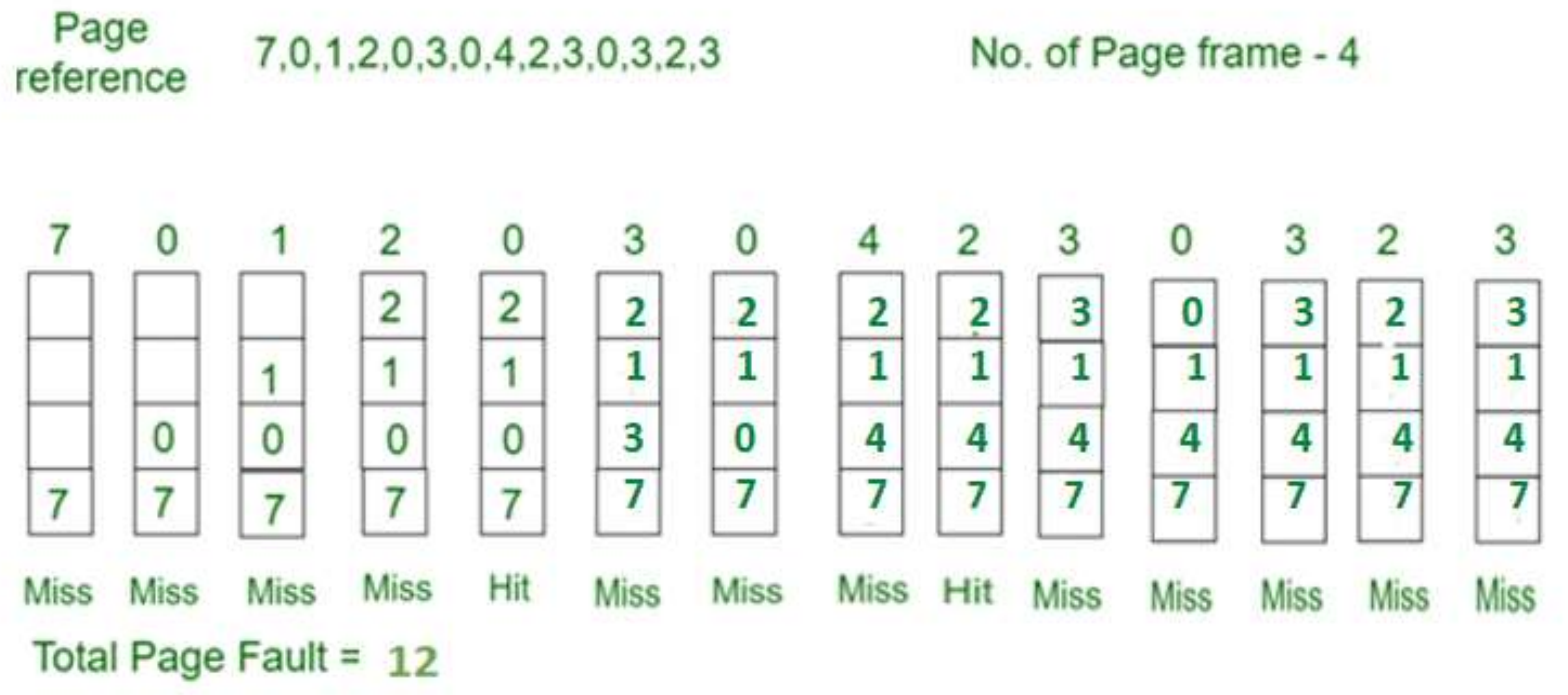


Total Page Fault = 6

Here LRU has same number of page fault as optimal but it may differ according to question.

Most Recently Used (MRU)

- In this algorithm, page will be replaced which has been used recently.



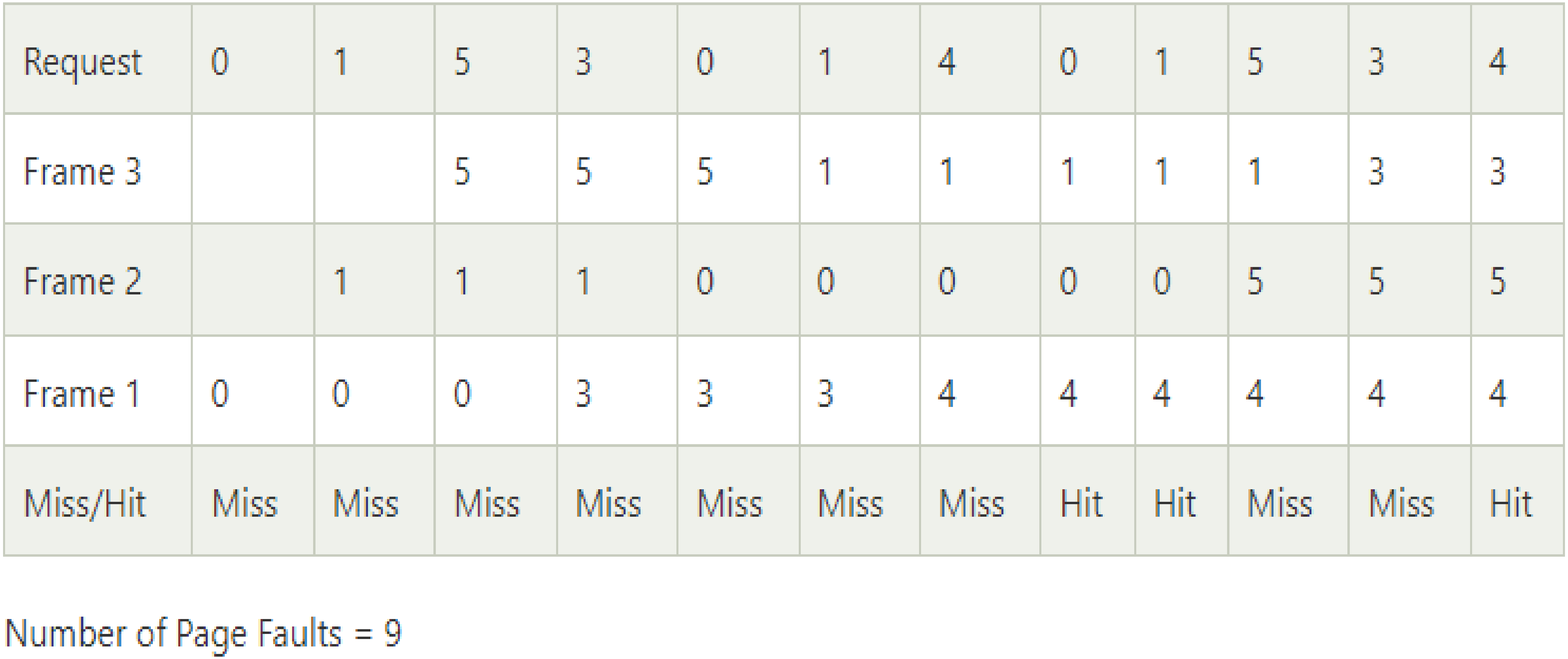
Belady's Anomaly

- In the case of LRU and optimal page replacement algorithms, it is seen that the number of page faults will be reduced if we increase the number of frames.
- However, Balady found that, In FIFO page replacement algorithm, the number of page faults will get increased with the increment in number of frames.
- **This is the strange behavior shown by FIFO algorithm in some of the cases.**
- **This is an Anomaly called as Belady's Anomaly.**

Example:

- The reference String is given as 0 1 5 3 0 1 4 0 1 5 3 4.
- Let's analyze the behavior of FIFO algorithm in two cases.

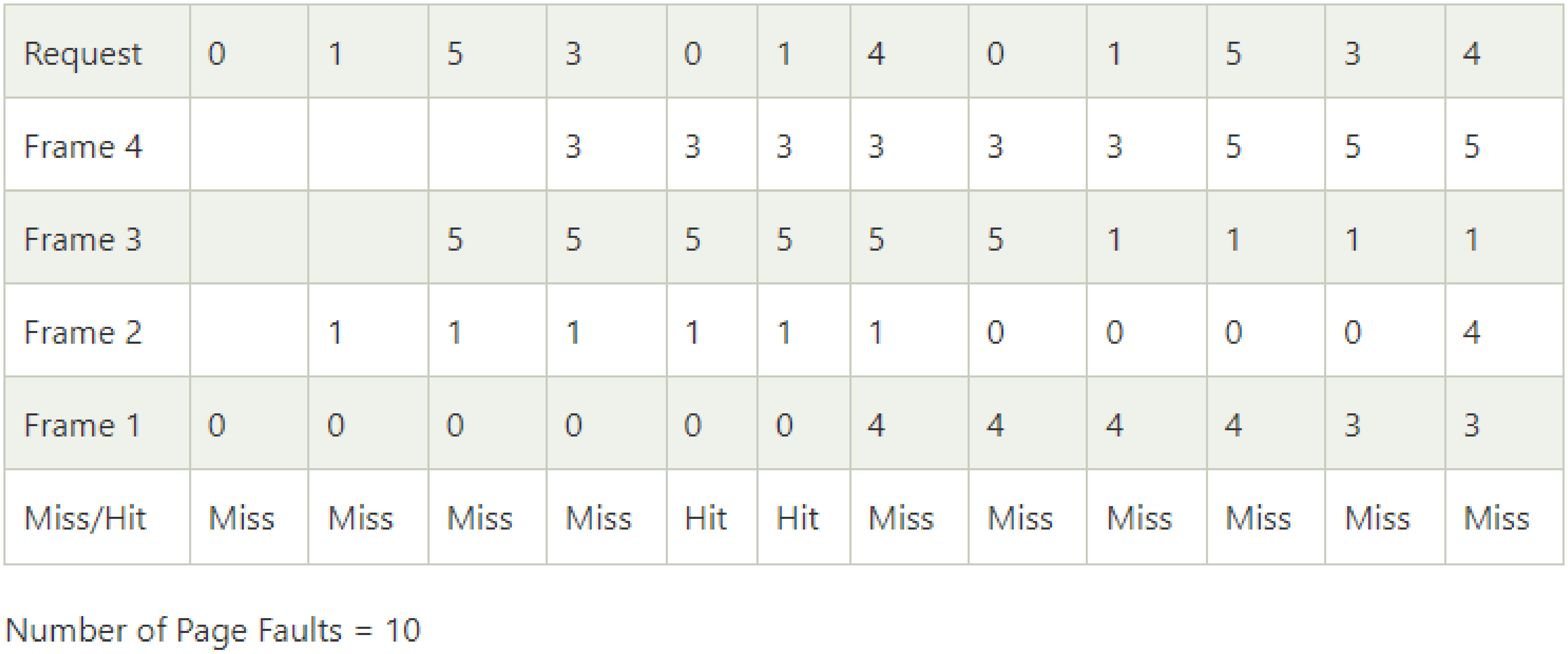
Case 1: Number of frames = 3



Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 3			5	5	5	1	1	1	1	1	3	3
Frame 2		1	1	1	0	0	0	0	0	5	5	5
Frame 1	0	0	0	3	3	3	4	4	4	4	4	4
Miss/Hit	Miss	Miss	Miss	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Hit

Number of Page Faults = 9

Case 2: Number of frames = 4



Request	0	1	5	3	0	1	4	0	1	5	3	4
Frame 4				3	3	3	3	3	3	5	5	5
Frame 3			5	5	5	5	5	5	1	1	1	1
Frame 2		1	1	1	1	1	1	0	0	0	0	4
Frame 1	0	0	0	0	0	0	4	4	4	4	3	3
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Miss	Miss	Miss	Miss	Miss	Miss

Number of Page Faults = 10

- Therefore, in this example, the number of page faults is increasing by increasing the number of frames hence this suffers from Belady'sAnomaly.

Frame Allocation Algorithms

There are mainly five ways of frame allocation algorithms in the OS. These are as follows:

- **Equal Frame Allocation**
- **Proportional Frame Allocation**
- **Priority Frame Allocation**
- **Global Replacement Allocation**
- **Local Replacement Allocation**

Equal Frame Allocation

- In equal frame allocation, the processes are assigned equally among the processes in the OS.
- For example, if the system has 30 frames and 7 processes, each process will get 4 frames.
- The 2 frames that are not assigned to any system process may be used as a free-frame buffer pool in the system.
- The main disadvantages are the following,
- In a system with processes of varying sizes, assigning equal frames to each process makes little sense.
- Many allotted empty frames will be wasted if many frames are assigned to a small task.

Proportional Frame Allocation

- The proportional frame allocation technique assigns frames based on the size needed for execution and the total number of frames in memory.
- The allocated frames for a process pi of size si are **ai = (si/S)*m**, in which S represents the total of all process sizes, and m represents the number of frames in the system.

- The only drawback of this algorithm is that it doesn't allocate frames based on priority. Priority frame allocation solves this problem.

Priority Frame Allocation

- Priority frame allocation assigns frames based on the number of frame allocations and the processes.
- Suppose a process has a high priority and requires more frames that many frames will be allocated to it.
- Following that, lesser priority processes are allocated.

Global Replacement Allocation

- When a process requires a page that isn't currently in memory, it may put it in and select a frame from the all frames sets, even if another process is already utilizing that frame. In other words, one process may take a frame from another.
- The advantage is that, the process performance is not hampered, resulting in higher system throughput.
- The disadvantage is that the process itself may not solely control the page fault ratio of a process. The paging behavior of other processes also influences the number of pages in memory for a process.

Local Replacement Allocation

- When a process requires a page that isn't already in memory, it can bring it in and assign it a frame from its set of allocated frames.
- The advantage is that, the paging behavior of a specific process has an effect on the pages in memory and the page fault ratio.
- Disadvantage is that, a low priority process may obstruct a high priority process by refusing to share its frames.

Paging - Advantages and Disadvantages

Advantages

- Allocating memory is easy and cheap
- Any free page is ok, OS can take first one out of list it keeps
- **Eliminates external fragmentation.**
- Data (page frames) can be scattered all over PM
- Pages are mapped appropriately anyway
- Allows demand paging and prepaging

Paging - Advantages and Disadvantages

Disadvantages

- **Internal fragmentation is caused on older systems.**
- The memory lookup time is more in paging compared to segmentation.
- Additional memory is consumed by the page tables.
- Multi-level paging can cause memory reference overhead.

Segmentation

Introduction

- Segmentation is another memory management technique that gives the user's view of a process. The user's view is mapped into the Physical Memory.
- In Segmentation, a process is divided into multiple segments.
- The size of each segment is not necessarily the same which is different from paging. In Paging, a process was divided into equal partitions called pages.
- The module contained in a segment decides the size of the segment

Why Segmentation is needed?

- We used Paging as one of the memory management techniques.
- In Paging, the process was divided into equal-sized pages irrespective of the fact that what is inside the pages.
- It also divides some relative parts of a process into different pages which should be loaded on the same page.
- It decreases the efficiency of the system and doesn't give the user's view of a process.
- **In Segmentation, similar modules are loaded in the same segments.** It gives the user's view of a process and also increases the efficiency of the system as compared to Paging.

Types of Segmentation

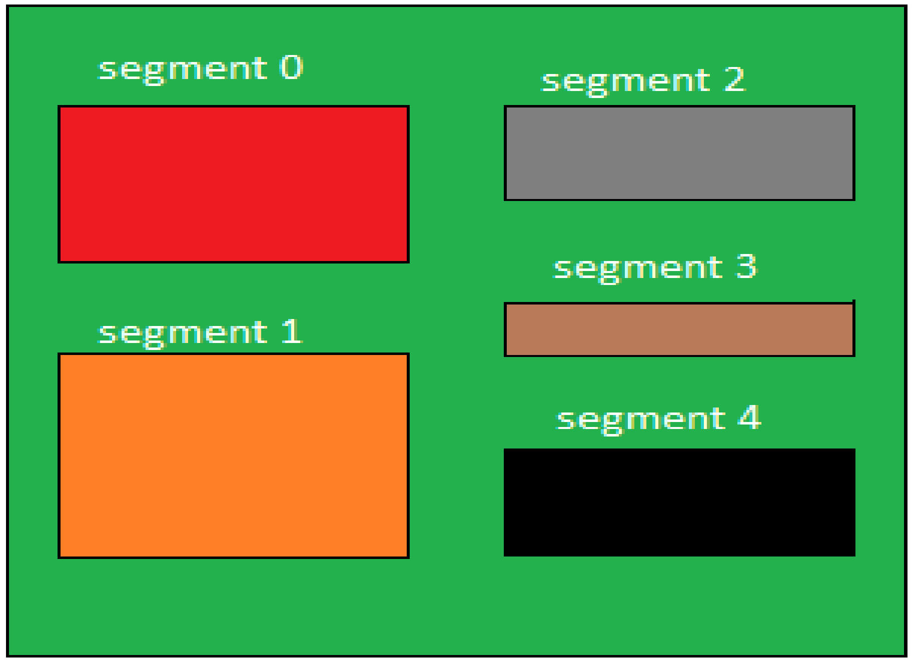
There are two types of Segmentation. These are:

- **Simple Segmentation** - In Simple Segmentation, each process is divided into multiple segments and all of them are loaded into the memory at run time. They can be at non-contagious locations.
- **Virtual Memory Segmentation** - In Virtual Memory Segmentation, each process is divided into multiple segments and all of them do not reside at one point at a time.

Segment Table

- Segment Table stores the information about all the segments of a process.
- It helps in the mapping of the two-dimensional logical addresses to the physical addresses.
- It is stored in the main memory.
- **There are two entries in the Segment Table.**
  - **Base Address** - It is the starting physical address of the particular segment inside the main memory.
  - **Limit** - It denotes the size of a particular segment.

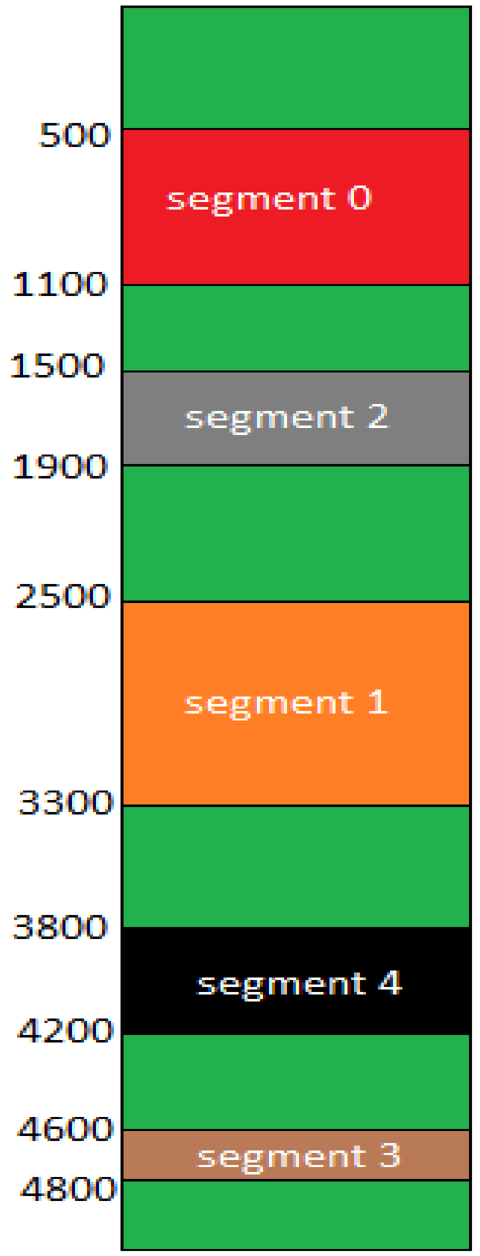
Logical View of Segmentation



Logical Address Space

Segment Number	base address	Limit
0	500	600
1	2500	800
2	1500	400
3	4600	200
4	3800	400

Segment Table

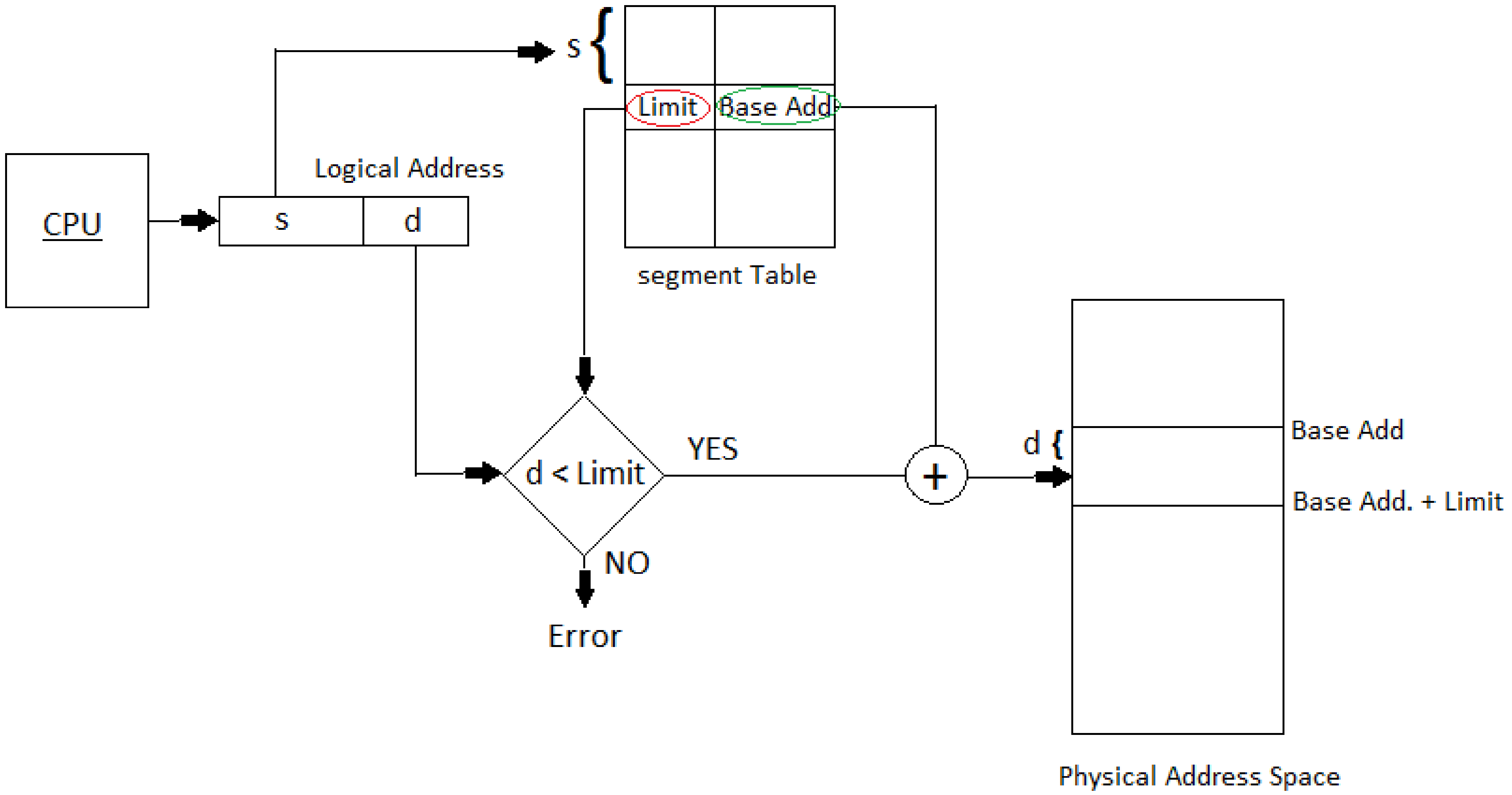


Physical Address Space

Translation of Logical Address into Physical Address

- The CPU generates the logical address which consists of two parts:
  - **Segment Number(s)** - It is the number of bits required to represent a Segment. It is used as an index in the Segment Table.
  - **Segment Offset(d)** - It is the number of bits required to represent the size of a Segment.

Translation of Logical Address into Physical Address



Advantages and Disadvantages of Segmentation

Advantages

- No internal fragmentation is there in segmentation.
- Segment Table is used to store the records of the segments. The segment table itself consumes small memory as compared to a page table in paging.
- Segmentation provides better CPU utilization as an entire module is loaded at once.
- Segmentation is near to the user's view of physical memory. Segmentation allows users to partition the user programs into modules. These modules are nothing but the independent codes of

Advantages and Disadvantages of Segmentation

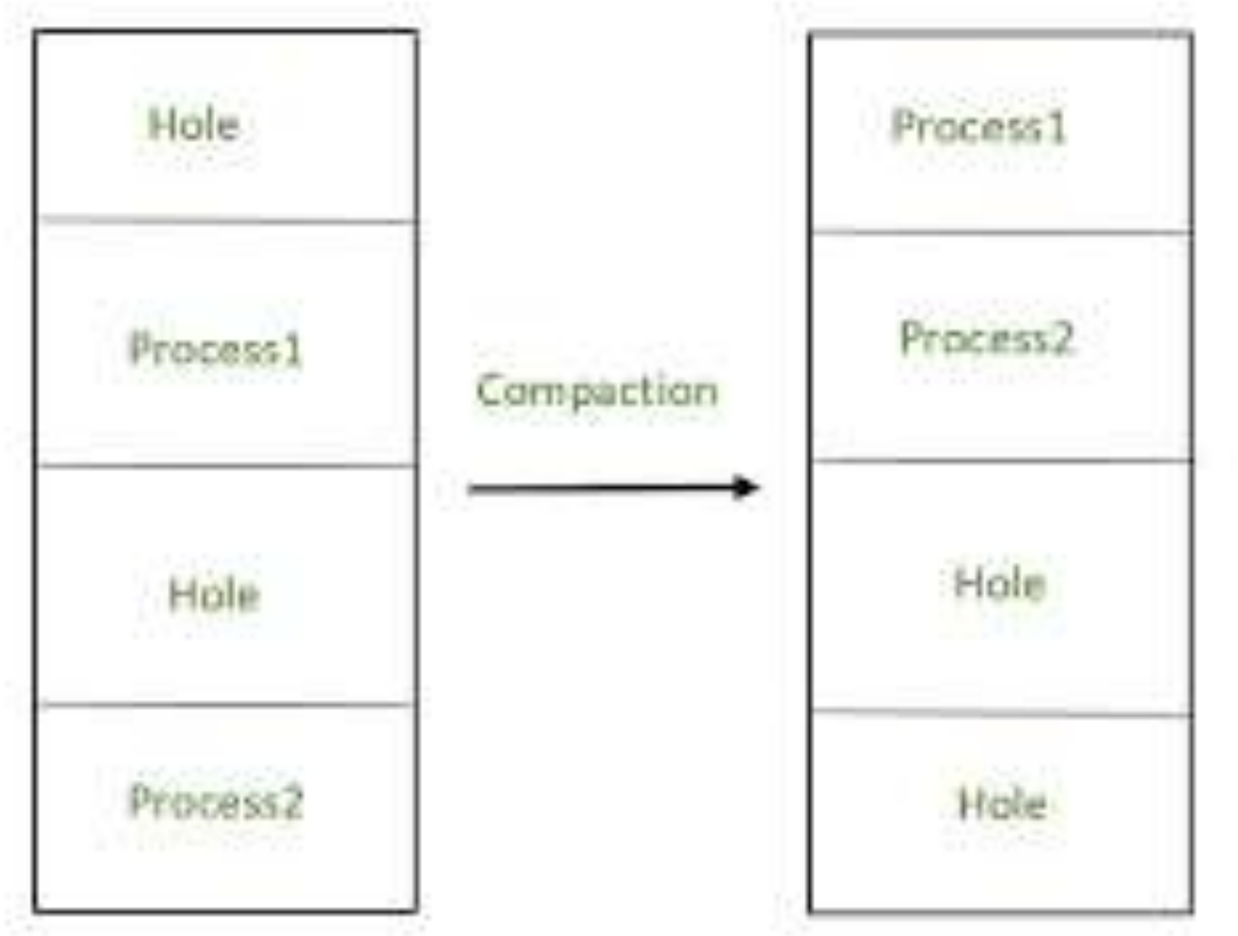
Disadvantages

- External Fragmentation.
- During the swapping of processes the free memory space is broken into small pieces, which is a major problem in the segmentation technique.
- Time is required to fetch instructions or segments.
- The swapping of segments of unequal sizes is not easy.

Compaction

- Compaction is a technique to collect all the free memory present in form of fragments into one large chunk of free memory, which can be used to run other processes.
- It does that by moving all the processes towards one end of the memory and all the available free space towards the other end of the memory so that it becomes contiguous.
- It is not always easy to do compaction. Compaction can be done only when the relocation is dynamic and done at execution time.

Compaction


Hole
Process1
Hole
Process2
Compaction
Process1
Process2
Hole
Hole