CHAPTER 4

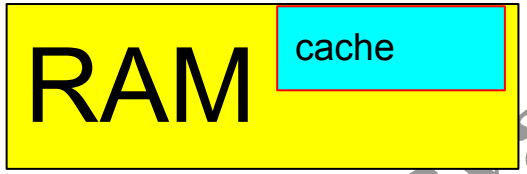
Memory Management

Shubhangi Chavan

Memory Management	Memory Management Requirements, Memory Partitioning: Fixed, Partitioning, Dynamic Partitioning, Memory Allocation Strategies: Best-Fit, First Fit, Worst Fit, Paging and Segmentation, TLB	**9**
	4.2 Virtual Memory: Demand Paging, Page Replacement Strategies:FIFO, Optimal, LRU, Thrashing	

Memory Management

- Ideally programmers want memory that is
  - large
  - fast
  - non volatile



- Memory hierarchy
  - small amount fast, expensive memory – cache
  - some medium-speed, medium price main memory
  - gigabytes - slow, cheap disk storage

- Memory manager handles the memory hierarchy

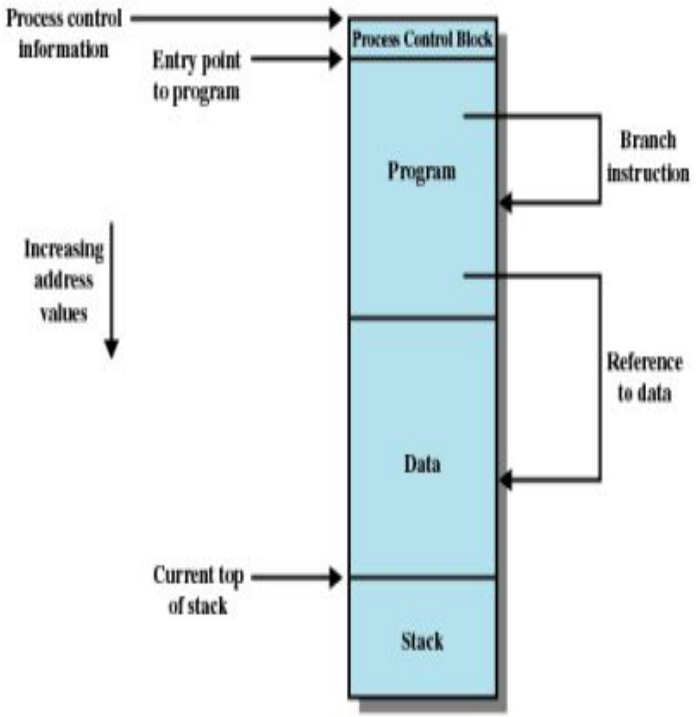
Requirements of Memory Management System

- 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.

Relocation

- Programmer does not know **where the program will be placed** in memory **when it is executed**
- While the program is executing, it may be **swapped to disk and returned to main memory at a different location (relocated)**
- Memory references must be translated in the code to actual physical memory address

- The process image is **occupying a continuous region of main memory**.
- The OS will need to know many things including the **location of PCB, the execution stack, and the code entry.**
- Within a program, there are **memory references in various instructions and these are called logical addresses.**
- After loading of the program into main memory, the processor and the **operating system must be able to translate logical addresses into physical addresses.**



Addressing Requirements for a Process

Protection

- Processes **should not be able to reference memory locations** in another process without permission
- Impossible to check absolute addresses(specifies a unique location within the **address space**) at compile time
- Must be checked at runtime
- Memory protection requirement must be satisfied by the processor (hardware) rather than the operating system (software)

Sharing

- Allow several processes to **access the same portion of memory**
- Better to allow each process access to the same copy of the program rather than have their own separate copy

Logical Organization

- Programs are written in modules
- Modules can be written and compiled independently
- Different degrees of protection given to modules (read-only, execute-only)
- Share modules among processes

Physical Organization

- Memory available for a program plus its data may be insufficient
- Overlaying allows various modules to be assigned the same region of memory
- Programmer does not know how much space will be available

There are three types of addresses used in a program before and after memory is allocated −

Symbolic addresses

The addresses used in a source code. The variable names, constants, and instruction labels are the basic elements of the symbolic address space.

Relative addresses

At the time of compilation, a compiler converts symbolic addresses into relative addresses

Physical addresses

The loader generates these addresses at the time when a program is loaded into main memory.

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

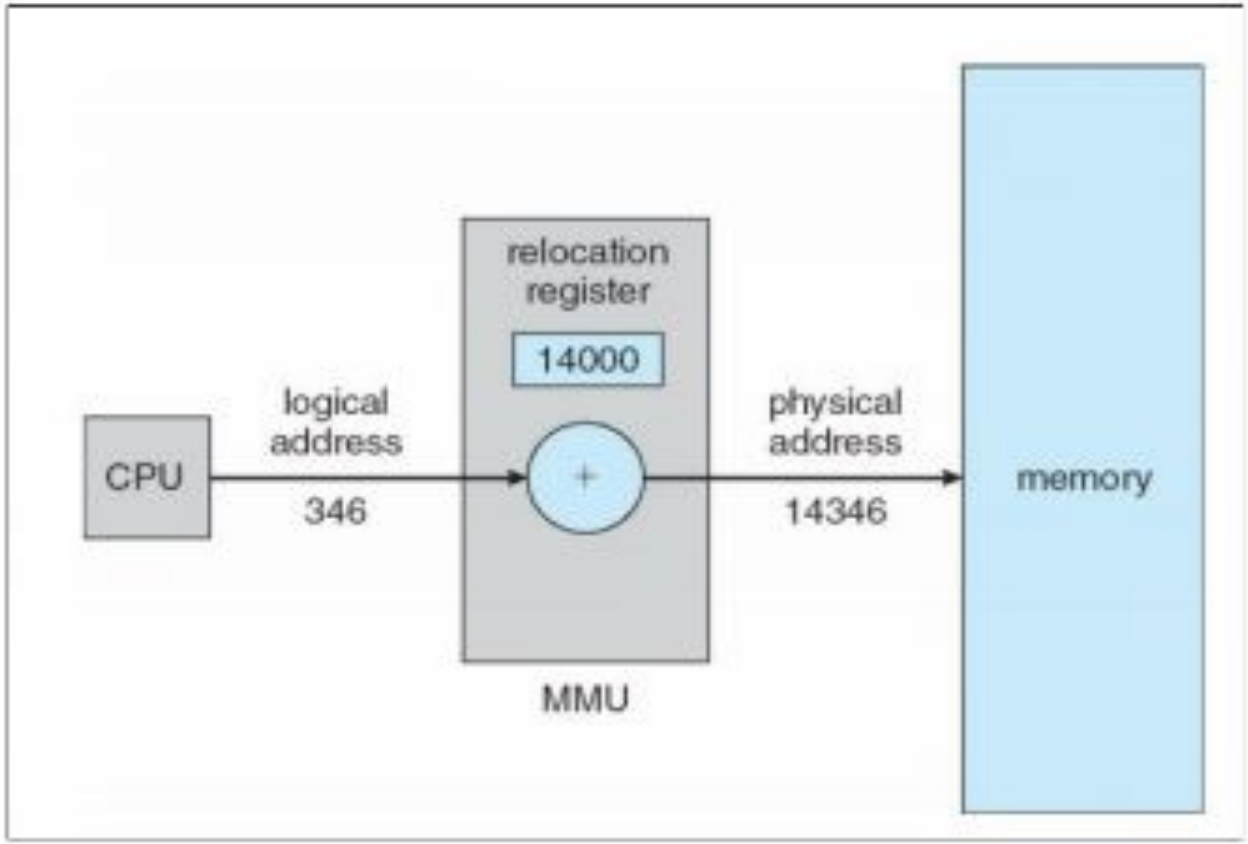
- Logical and physical addresses are the **same in compile-time** and load-time address-binding schemes
- **logical (virtual)** and physical addresses **differ in execution-time** address-binding scheme.
- Memory-Management Unit (MMU) is a hardware device that maps virtual to physical address.

Relocation Register

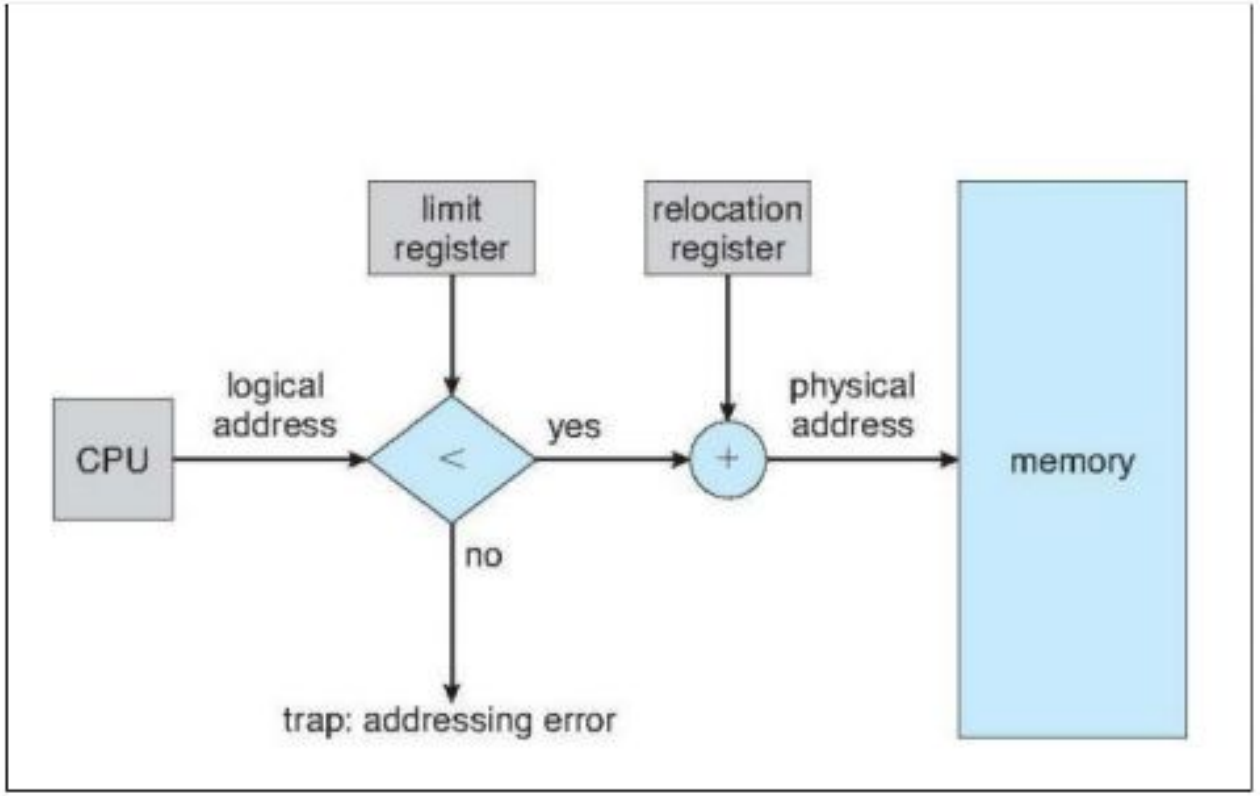
Single partition allocation

1. Relocation register scheme used to protect user processes from each other, and from changing OS code and data.

2. Relocation register contains value of smallest physical address; limit register contains range of logical addresses - each logical address must be less than the limit register.


relocation
register
14000
logical
address
346
CPU
+
physical
address
14346
memory
MMU

Relocation and Limit Registers



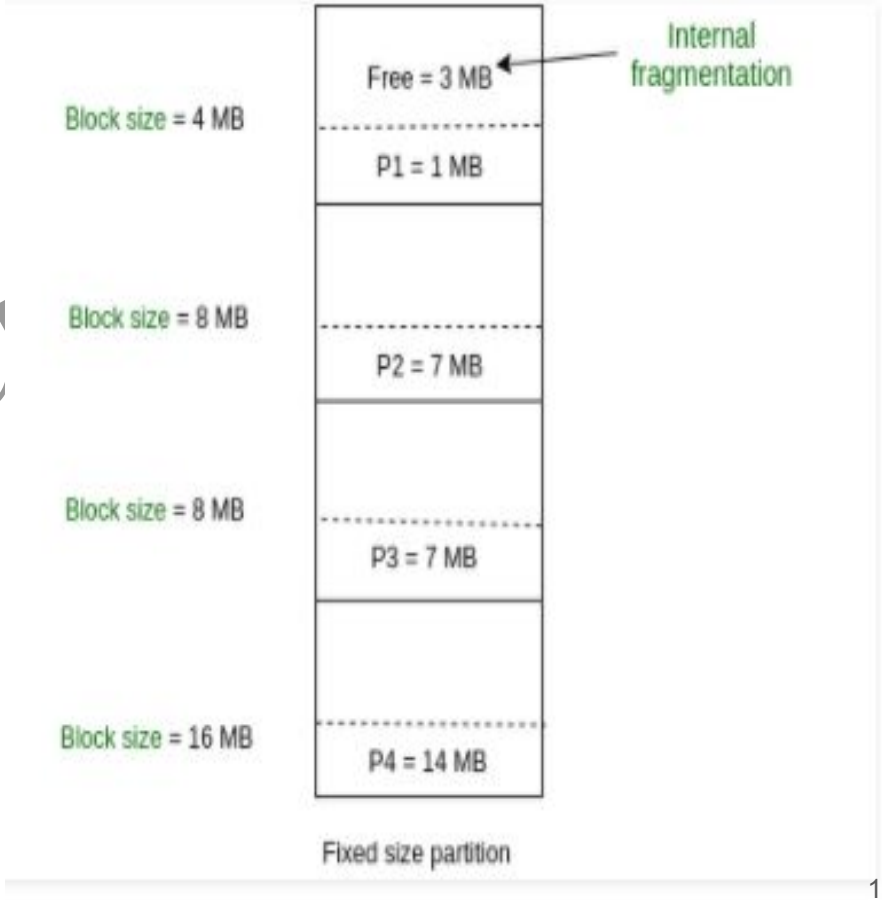
Contiguous Memory Allocation

Memory Partitioning

- In operating systems, Memory Management is the function **responsible for allocating and managing computer's main memory.**
- **What Is Memory Partitioning?**
  - Memory partitioning is the system by which the **memory of a computer system is divided into sections for use by the resident programs.** These memory divisions are known as partitions.
- There are two Memory Management Techniques:
  - Contiguous,
  - Non-Contiguous.
    - In Contiguous Technique, executing process must be loaded entirely in main-memory. **Contiguous Technique can be divided into:**
      - Fixed (or static) partitioning
      - Variable (or dynamic) partitioning

Fixed Partitioning:

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



first process is only consuming 1MB out of 4MB in the main memory.

Hence, Internal Fragmentation in first block is (4-1) = 3MB.

Sum of Internal Fragmentation in every block = (4-1)+(8-7)+(8-7)+(16-14)= 3+1+1+2 = 7MB.

As the process gets loaded and removed from the memory these spaces get broken into small pieces of memory that it can't be allocated to the coming processes. This problem is called **fragmentation**

Basically, there are two types of fragmentation:

- Internal Fragmentation
- External Fragmentation

External fragmentation

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

Internal fragmentation

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

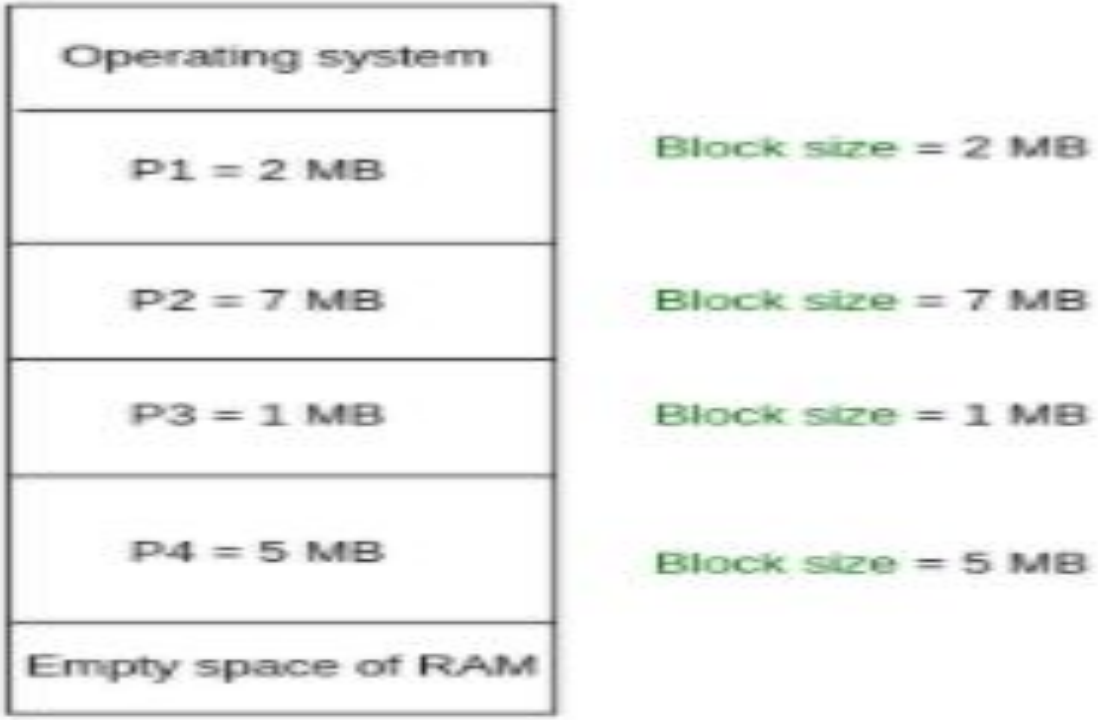
To remove external fragmentation is **compaction**.

Variable Partitioning

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

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

Dynamic partitioning



Partition size = process size
So, no internal Fragmentation

Memory Allocation Strategies: Best-Fit, First Fit, Worst Fit,

Memory allocation for Fixed sized Partition

First Fit

In the first fit approach is to allocate the first free partition or hole large enough which can accommodate the process. It finishes after finding the first suitable free partition.

Advantage

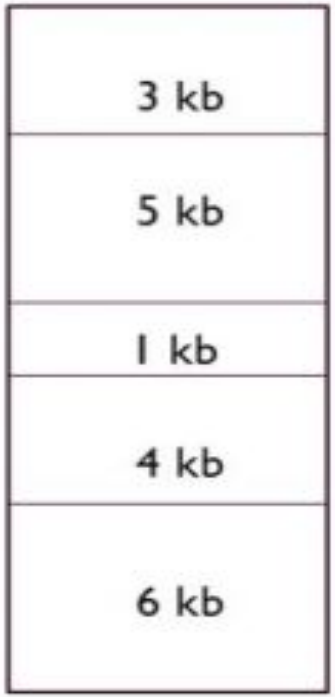
Fastest algorithm because it searches as little as possible.

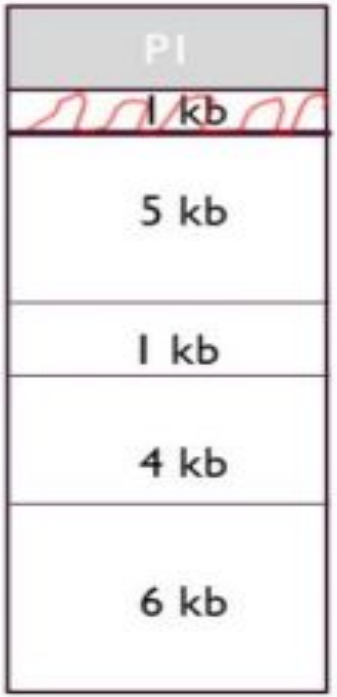
Disadvantage

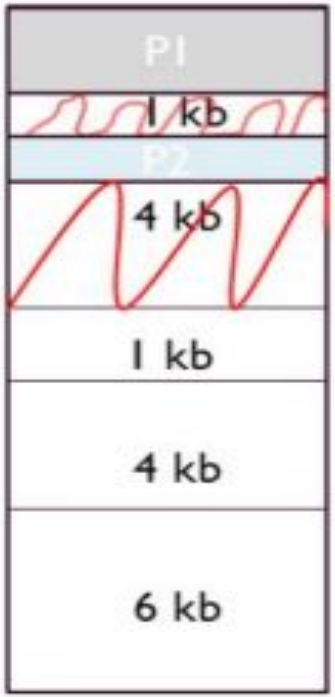
The remaining unused memory areas left after allocation become waste if it is too smaller. Thus request for larger memory requirement cannot be accomplished.

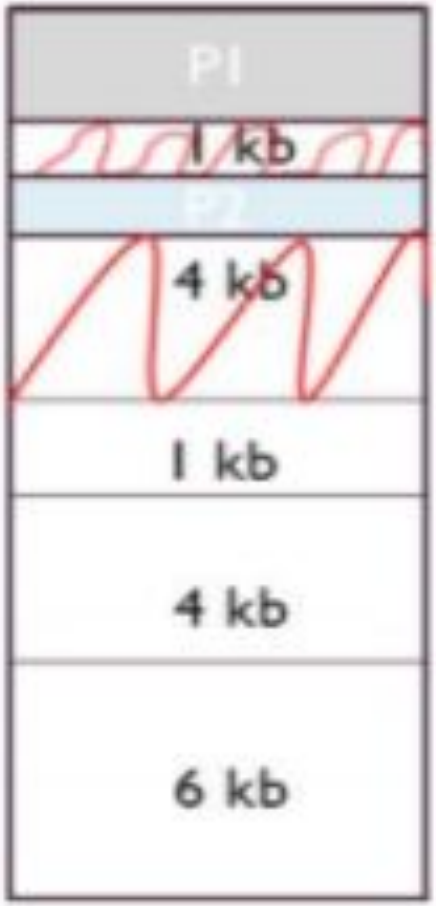
FIRST FIT

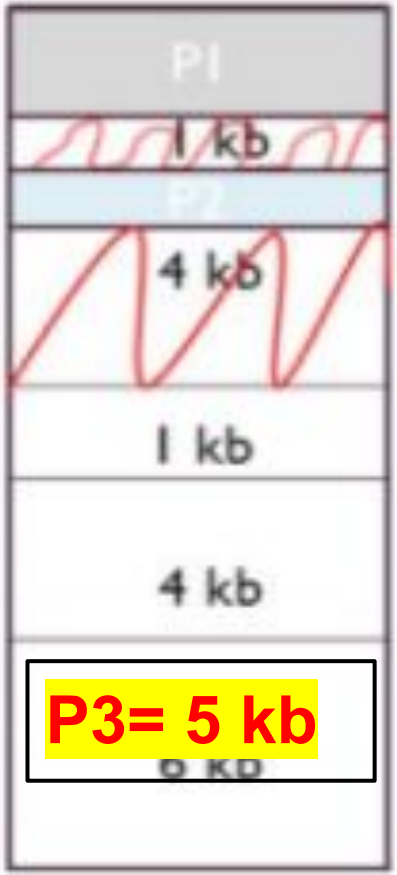
- Two processes P1 of 2kb & P2 of 1kb comes.











If **P3** with 5kb where it will allocate?

Best Fit

The best fit deals with allocating the smallest free partition which meets the requirement of the requesting process. This algorithm first searches the entire list of free partitions and considers the smallest hole that is adequate. It then tries to find a hole which is close to actual process size needed.

Advantage

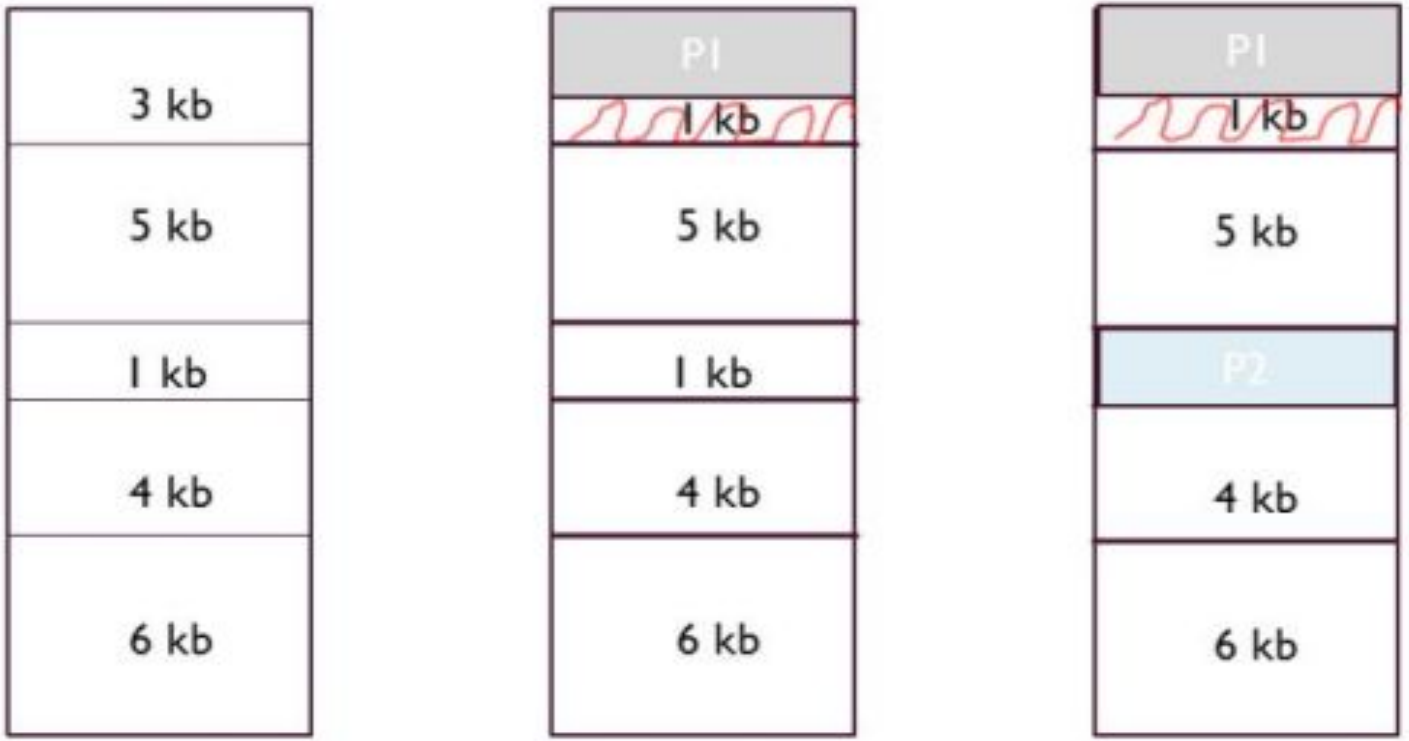
Memory utilization is much better than first fit as it searches the smallest free partition first available.

Disadvantage

It is slower and may even tend to fill up memory with tiny useless holes.

BEST FIT

- Two processes P1 of 2kb & P2 of 1kb comes.



Worst fit

In worst fit approach is to locate largest available free portion so that the portion left will be big enough to be useful. It is the reverse of best fit.

Advantage

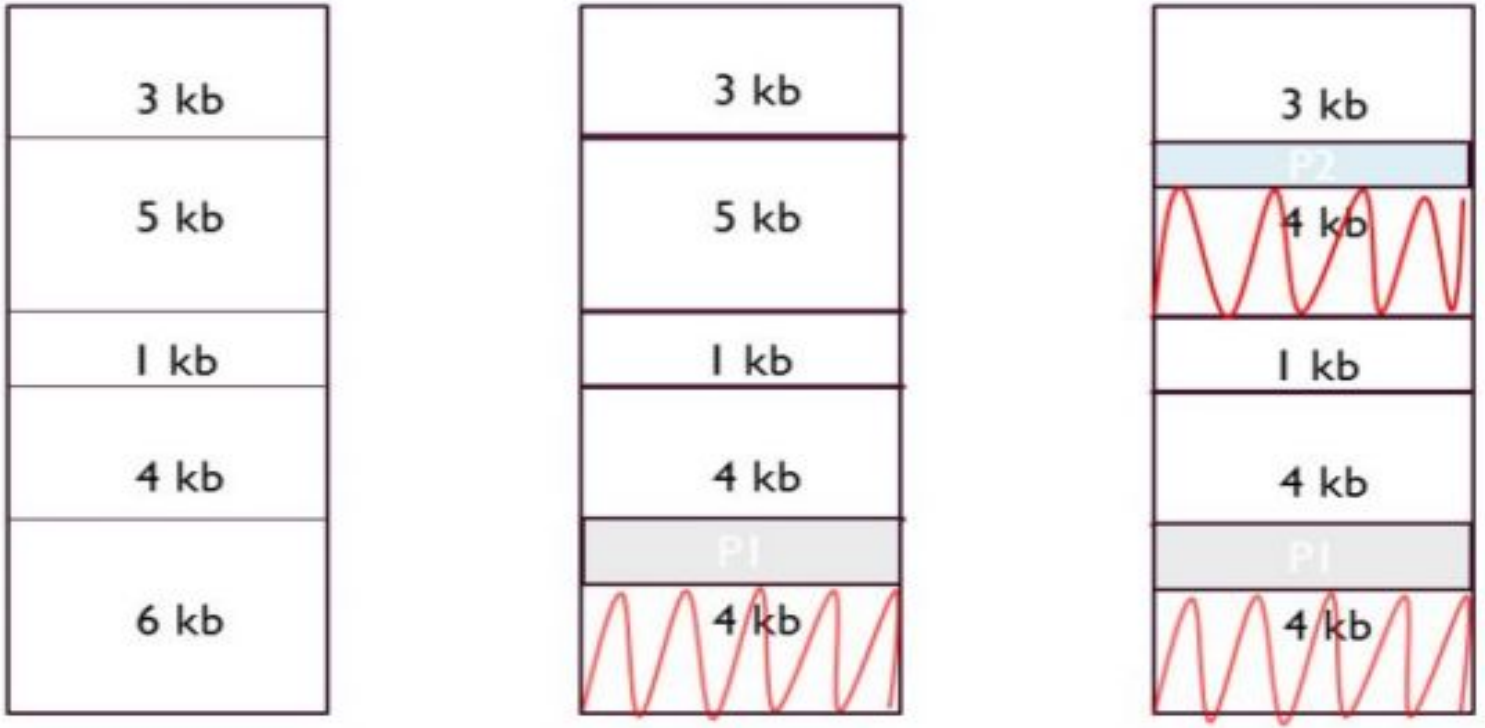
Reduces the rate of production of small gaps.

Disadvantage

If a process requiring larger memory arrives at a later stage then it cannot be accommodated as the largest hole is already split and occupied.

WORST FIT

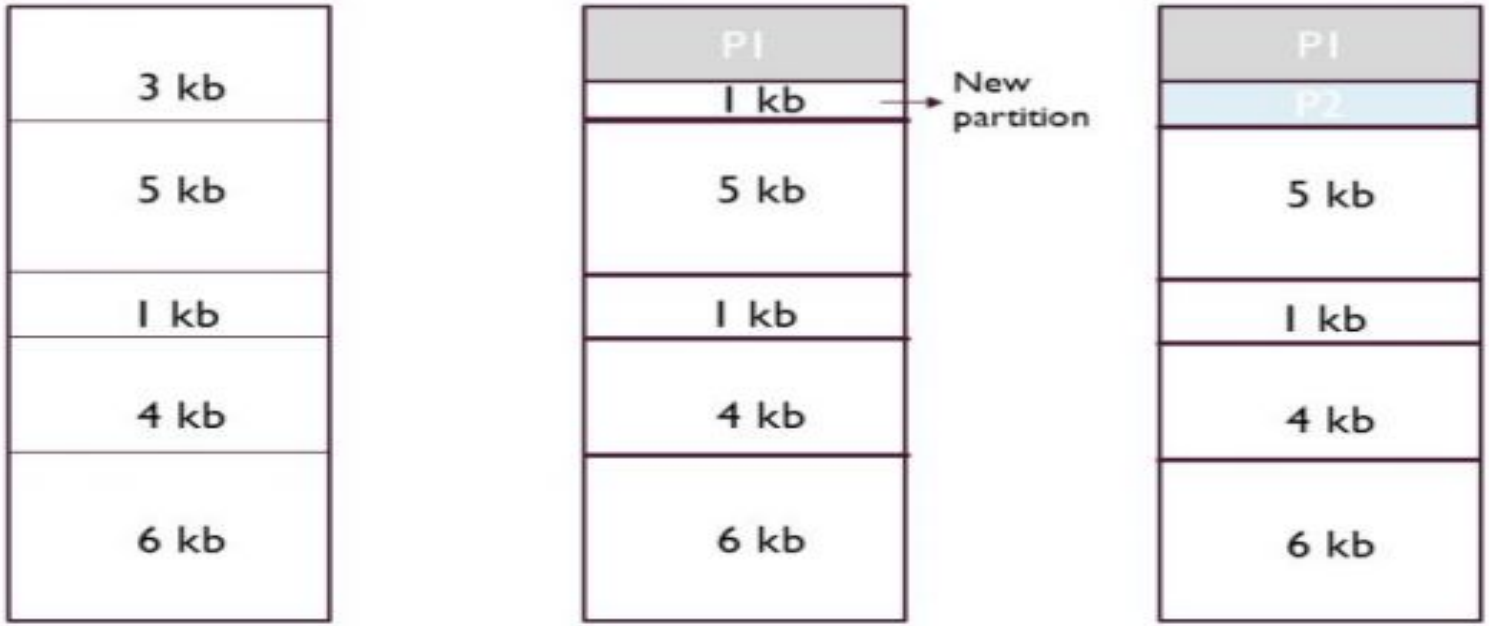
- Two processes P1 of 2kb & P2 of 1kb comes.



Memory allocation for Variable sized Partition

FIRST FIT

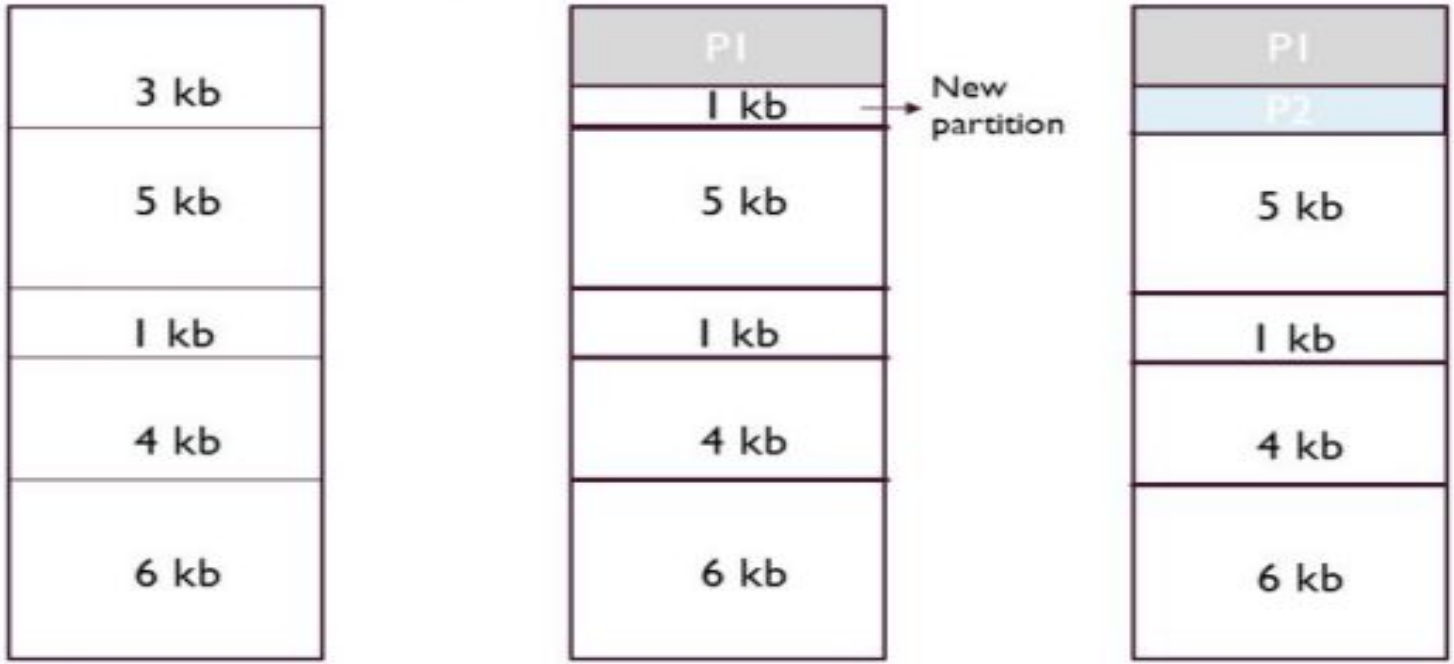
- Two processes P1 of 2kb & P2 of 1kb comes.



Memory allocation for Variable sized Partition

BEST FIT

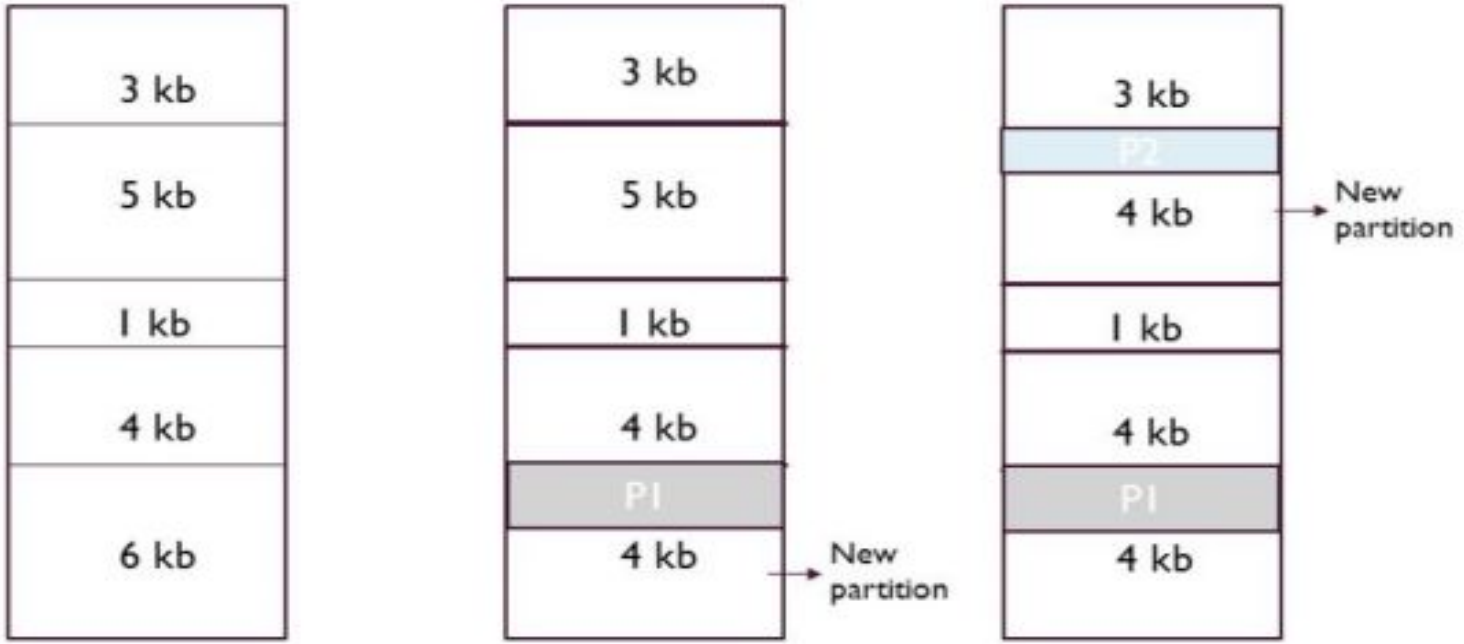
- Two processes P1 of 2kb & P2 of 1kb comes.



Memory allocation for Variable sized Partition

WORST FIT

- Two processes P1 of 2kb & P2 of 1kb comes.

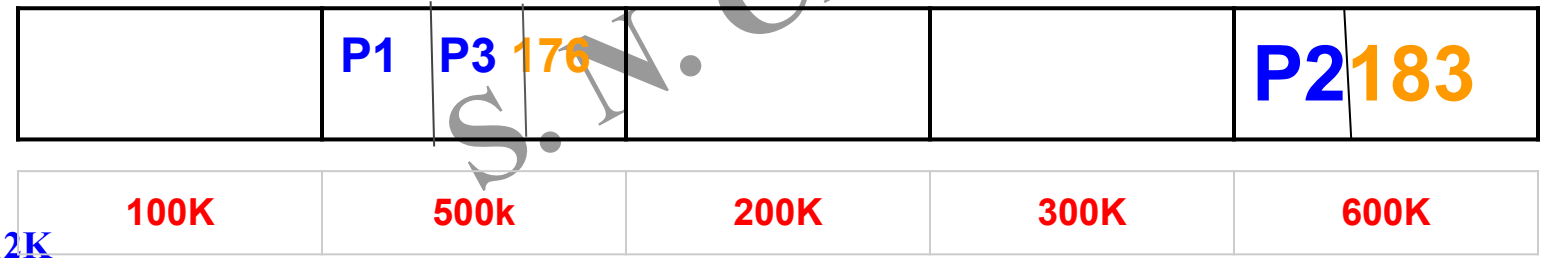


EXAMPLE:

Given memory partition of 100K, 500K, 200K, 300K and 600Kin order. How would each of the first fit,best fit and worst fit algorithm to place process of 212K ,417K, 112K and 426K in order.which algorithm make the efficient use of memory?

Given memory partition of 100K, 500K, 200K, 300K and 600Kin order. How would each of the first fit,best fit and worst fit algorithm to place process of 212K, 417K, 112K and 426K in order.which algorithm make the efficient use of memory?

FIRST FIT



P1=212K

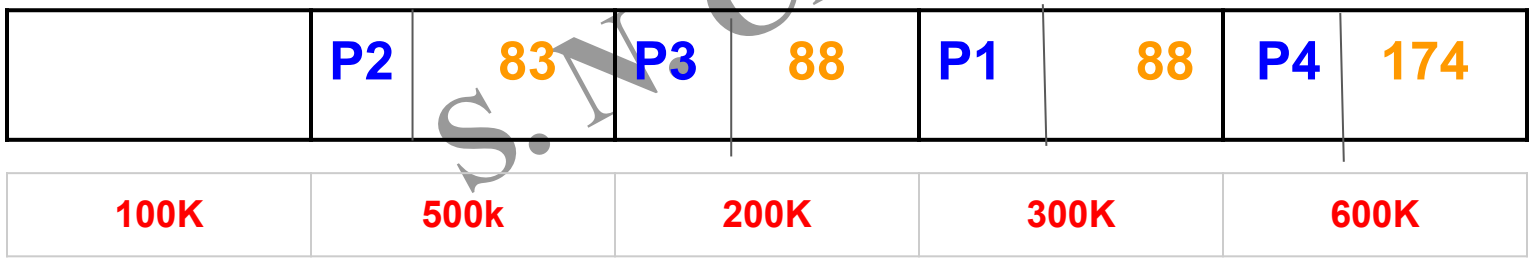
P2 = 417 K

P3 = 112 K

P4 = 426K it will meet with external fragmentation

Given memory partition of 100K, 500K, 200K, 300K and 600Kin order. How would each of the first fit,best fit and worst fit algorithm to place process of 212K, 417K, 112K and 426K in order.which algorithm make the efficient use of memory?

BEST FIT



P1=212K

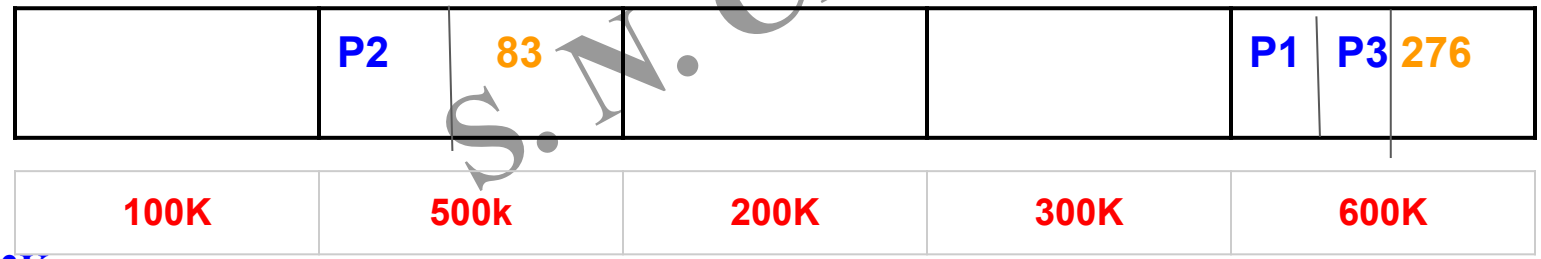
P2 = 417 K

P3 = 112 K

P4 = 426K

Given memory partition of 100K, 500K, 200K, 300K and 600Kin order. How would each of the first fit,best fit and worst fit algorithm to place process of 212K, 417K, 112K and 426K in order.which algorithm make the efficient use of memory?

Worst FIT



P1=212K

P2 = 417 K

P3 = 112 K

P4 = 426K it can't allocate and meet with external fragmentation

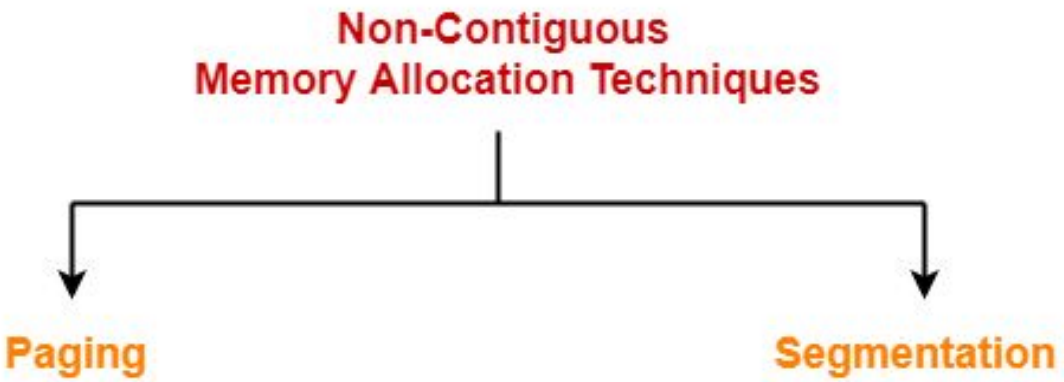
Placement Algorithms

Best-fit	First-fit	Next-fit
• chooses the block that is closest in size to the request	• begins to scan memory from the beginning and chooses the first available block that is large enough	• begins to scan memory from the location of the last placement and chooses the next available block that is large enough

Non-Contiguous Memory Allocation

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 in the main memory.

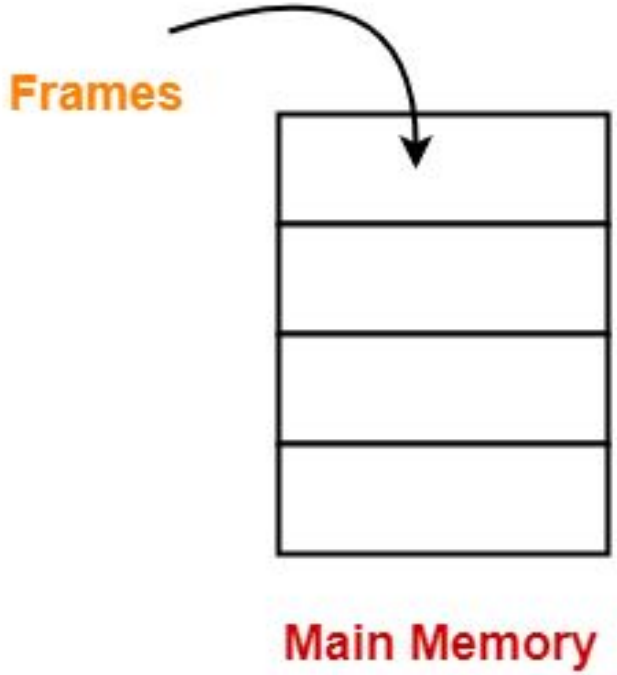


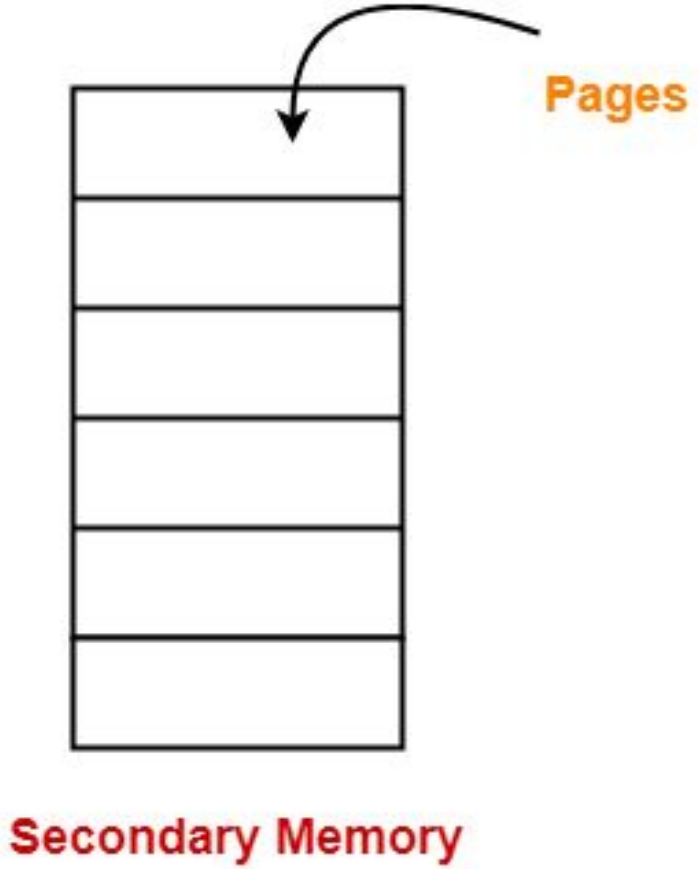
Non-Contiguous Allocation in Operating System

- In non-contiguous allocation, Operating system needs to maintain the table which is called **Page Table** for each process which contains the **base address** of the each block which is acquired by the process in memory space.
- In non-contiguous memory allocation, different parts of a process is allocated different places in Main Memory.
- **Spanning is allowed** which is not possible in other techniques like Dynamic or Static Contiguous memory allocation.

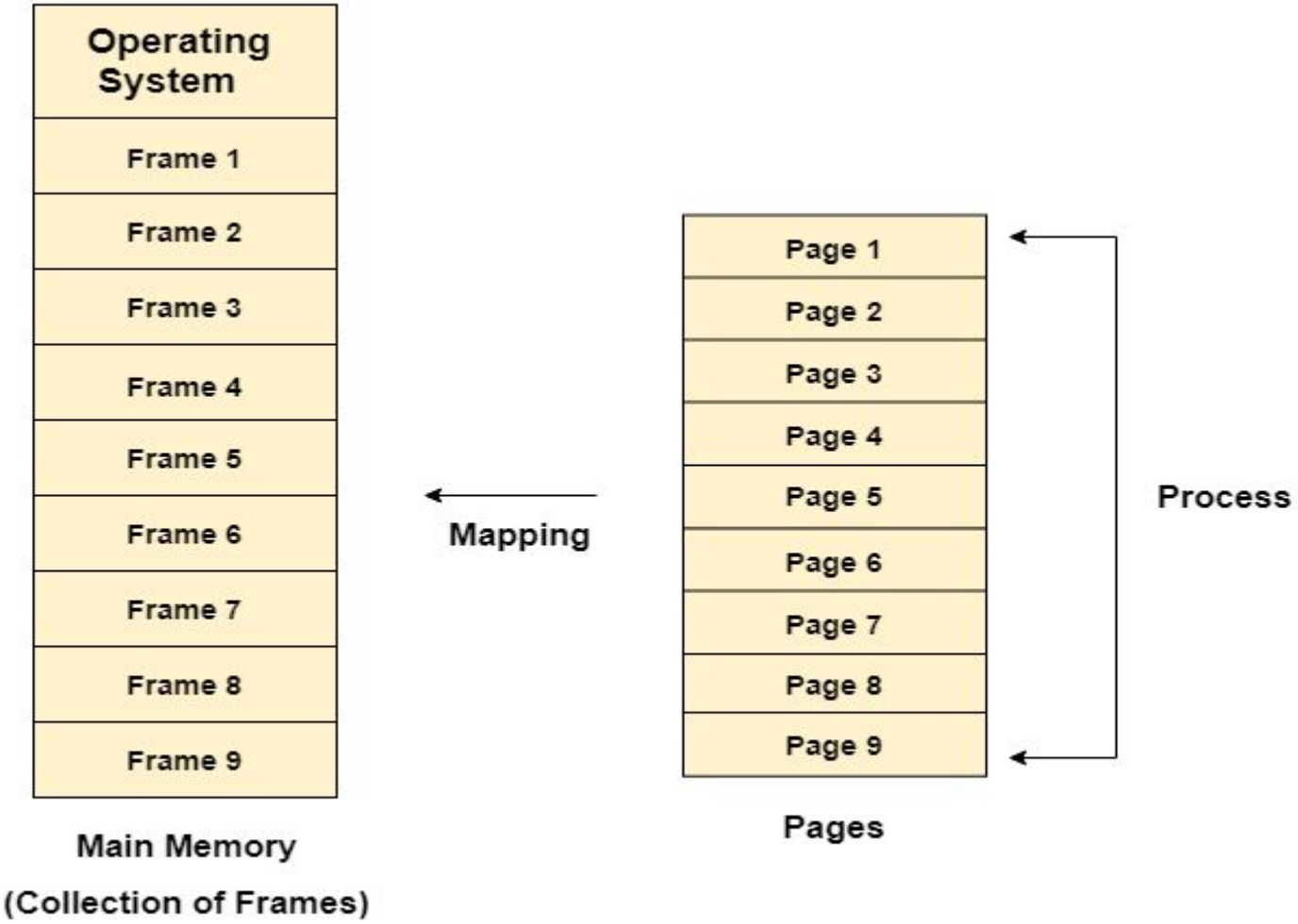
Paging

- Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory.
- 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.**
- One page of the process is to be stored in one of the frames of the memory.
- The size of the last part may be less than the page size.


Frames
Main Memory

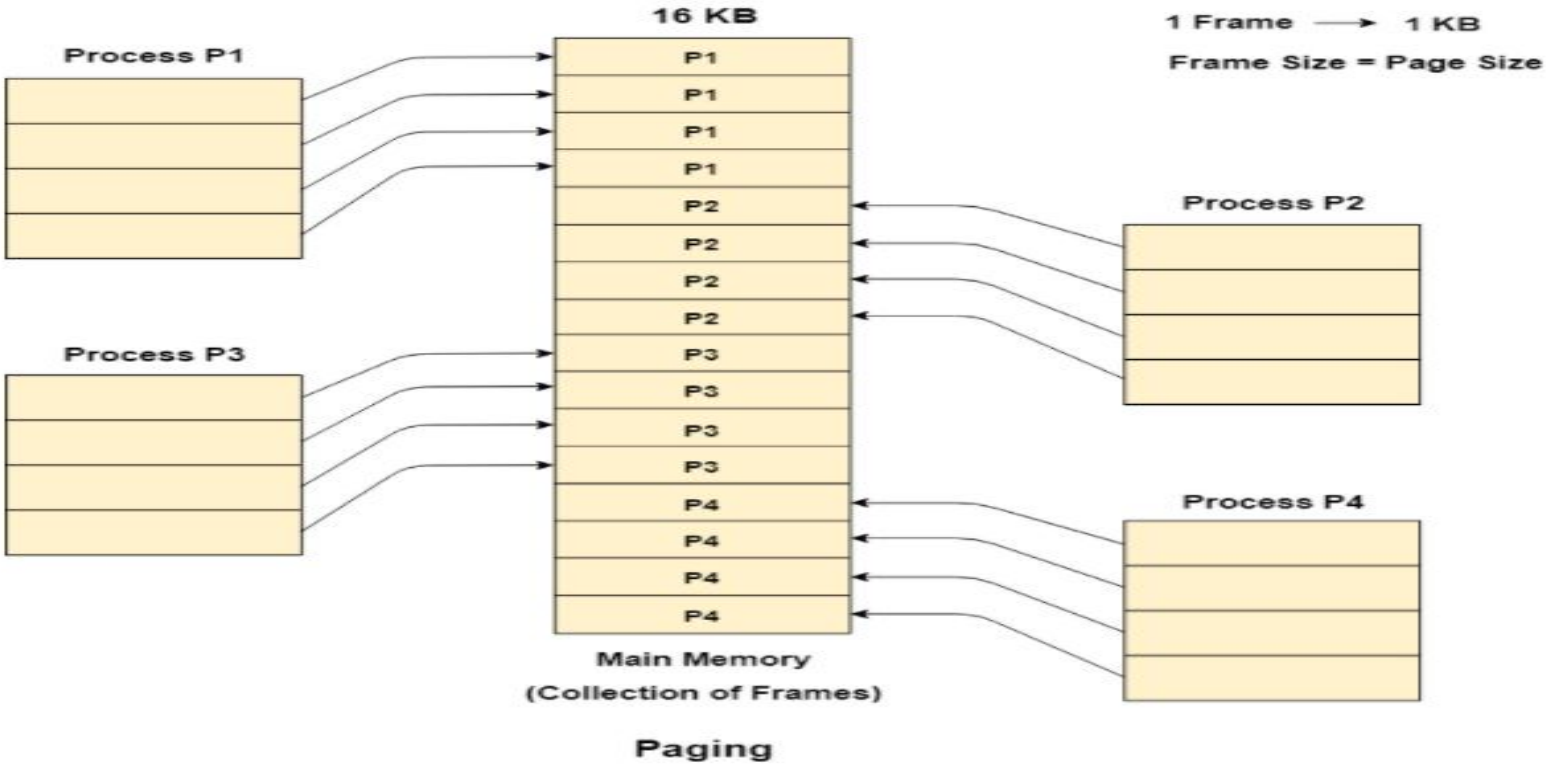

Pages
Secondary Memory

- The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.
- Paging is done to **remove External Fragmentation.**
- Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.
- Different operating system defines different frame sizes. **The sizes of each frame must be equal**


Operating System
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Frame 7
Frame 8
Frame 9
Main Memory
(Collection of Frames)
Mapping
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
Page 8
Page 9
Pages
Process

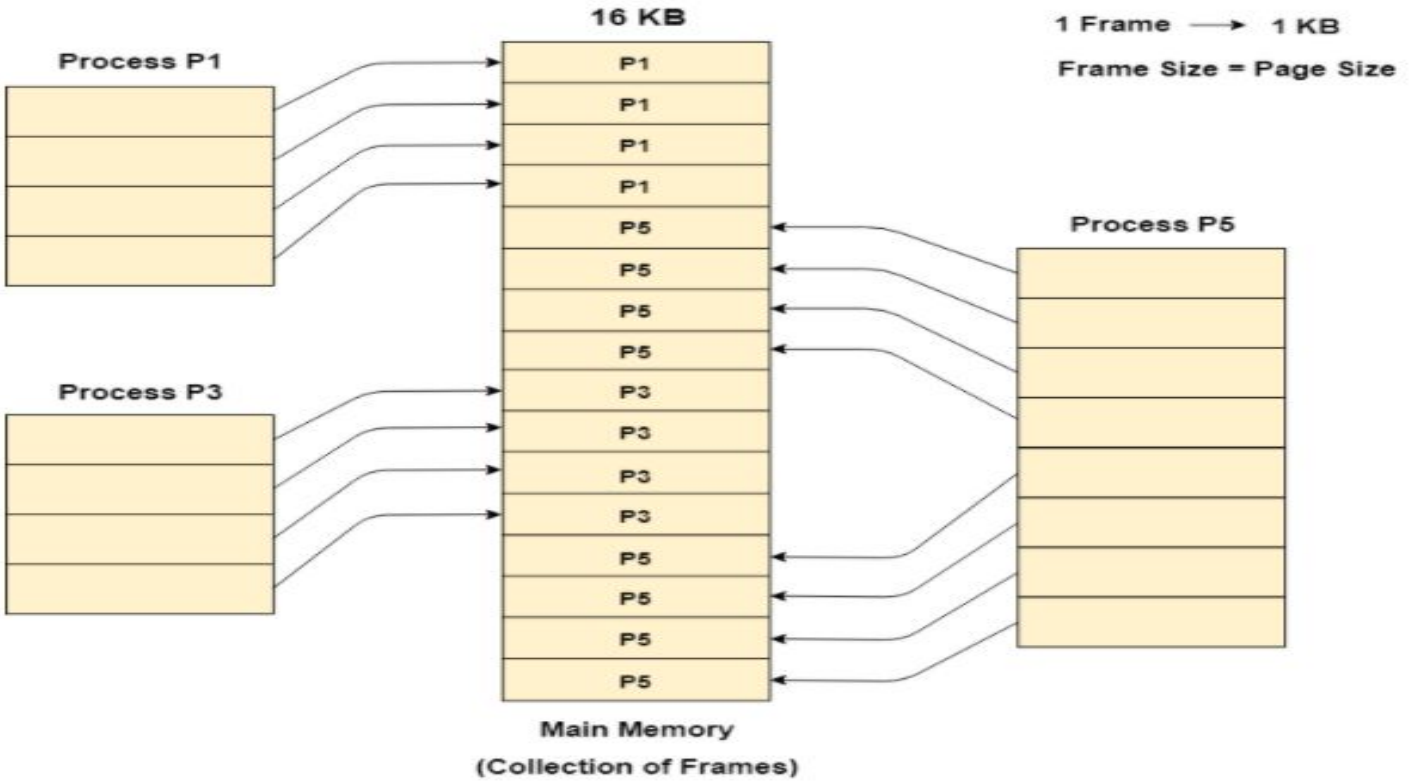
Example

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.



Paging

Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.



Paging

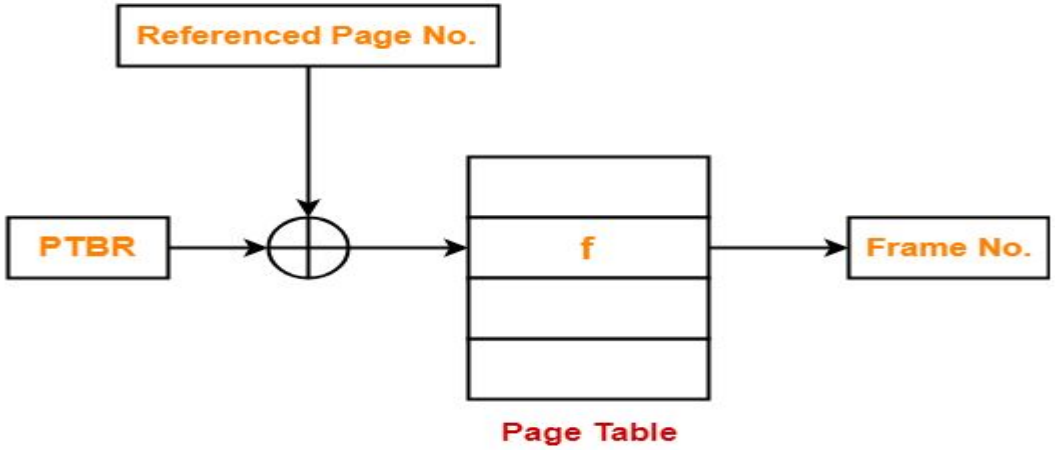
Page Table-

- Page table is a data structure.
- It maps the page number referenced by the CPU to the frame number where that page is stored.

Characteristics-

- Page table is stored in the main memory.
- 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.
- Each process has its own independent page table.

- Page Table Base Register (PTBR) provides the base address of the page table.
- 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.



Obtaining Frame Number Using Page Table

Memory Management Unit

The purpose of Memory Management Unit (MMU) is to convert the logical address into the physical address. The logical address is the address generated by the CPU for every page while the physical address is the actual address of the frame where each page will be stored.

When a page is to be accessed by the CPU by using the logical address, the operating system needs to obtain the physical address to access that page physically.

The logical address has two parts.

1. Page Number
2. Offset

Memory management unit of OS needs to convert the page number to the frame number.

Address generated by CPU (logical address) is divided into

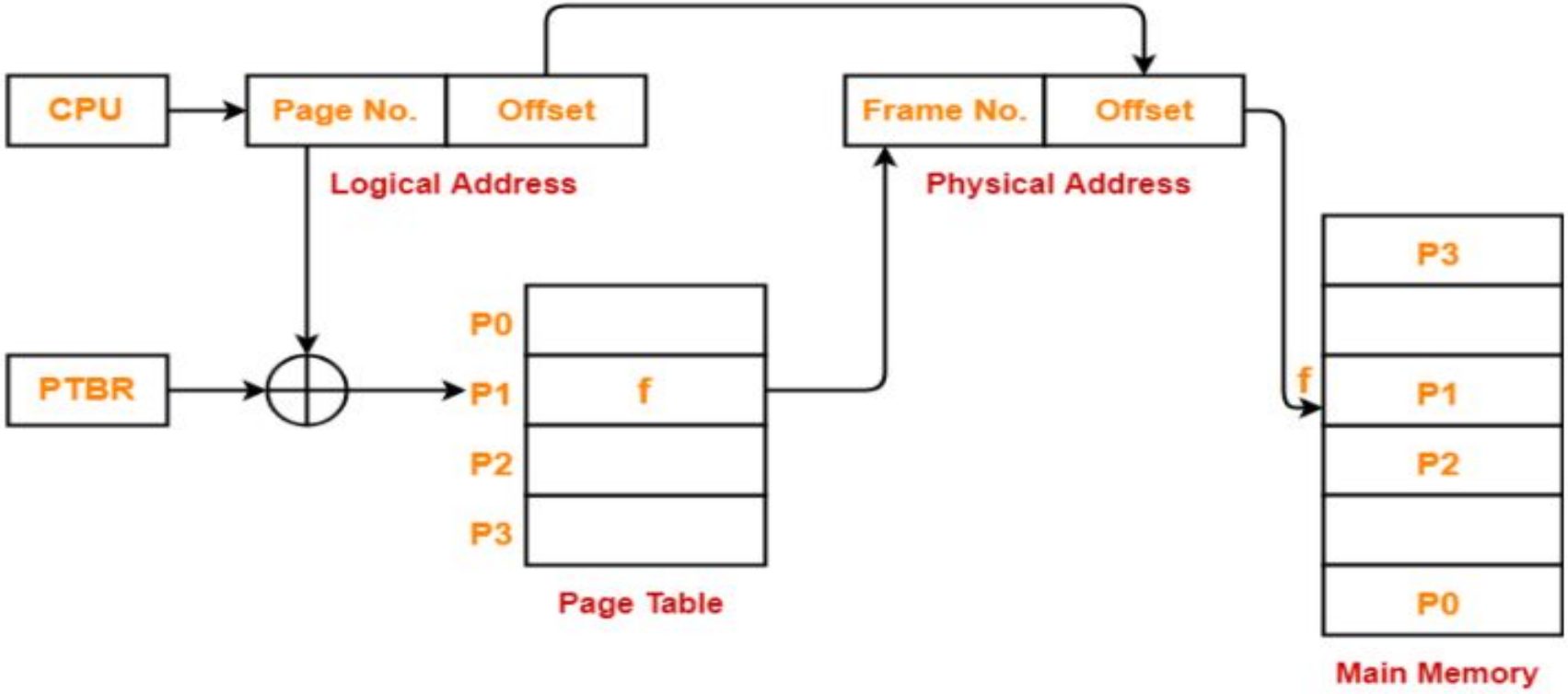
- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

LA=p+d

Physical Address is divided into

- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

PA=f+d



Translating Logical Address into Physical Address

Important Formulas-

For Main Memory-

- Physical Address Space = Size of main memory
- Frame size = Page size
- Size of main memory = Total number of frames x Page size
- If number of frames in main memory = 2^X, then number of bits in frame number = X bits
- If Page size = 2^X Bytes, then number of bits in page offset = X bits
- If size of main memory = 2^X Bytes, then number of bits in physical address = X bits

For Process-

- Virtual Address Space = Size of process
- Number of pages the process is divided = Process size / Page size
- If process size = 2^X bytes, then number of bits in virtual address space = X bits

Problem-01:

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

Solution-

We have-

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

Thus, Size of memory

= 2^{22} x 2 bytes

= 2^{23} bytes

= 8 MB

Problem-02:

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

Solution-

Let ‘n’ number of bits are required. Then, Size of memory = 2^n x 4 bytes.

Since, the given memory has size of 16 GB, so we have-

2^n x 4 bytes = 16 GB

2^n x 4 = 16 G

2^n x 2^2 = 2^{34}

2^n = 2^{32}

∴ n = 32 bits

PHYSICAL ADDRESS CALCULATION:

Given-

- Size of main memory = 64 MB
- Number of bits in virtual address space = 32 bits
- Page size = 4 KB

We will consider that the memory is byte addressable.

Number of Bits in Physical Address-

Size of main memory

= 64 MB =2^6* 2^20

= 2^{26} B

Thus, Number of bits in physical address = 26 bits

Number of Frames in Main Memory-

Number of frames in main memory = Size of main memory / Frame size

$= 64\ MB / 4\ KB$

$= 2^{26}\ B / 2^{12}\ B$

$= 2^{14}$

Thus, Number of bits in frame number = 14 bits

Number of Bits in Page Offset-

We have,

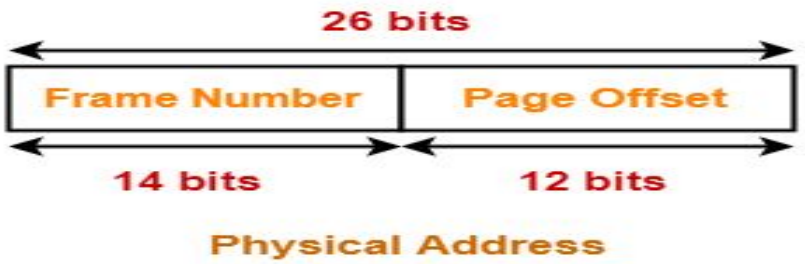
Page size

= 4 KB

= 2^{12} B

Thus, Number of bits in page offset = 12 bits

So, Physical address is-



Translation Lookaside Buffer-

Drawbacks of Paging

1. Size of Page table can be **very big** and therefore it wastes main memory.
2. CPU will take more time to read a single word from the main memory.

- To overcome this problem a **high-speed cache** is set up for page table entries called a Translation Lookaside Buffer (TLB). **Translation Lookaside Buffer (TLB) is nothing but a special cache used to keep track of recently used transactions.**
- TLB contains page table entries that have been most recently used.
- Given a virtual address, the processor examines the TLB if a page table entry is present (TLB hit), the frame number is retrieved and the real address is formed.
- If a page table entry is not found in the TLB (TLB miss), the page number is used as index while processing **page table.**

- TLB first checks if the page is already in main memory, if not in main memory a page fault is issued then the TLB is updated to include the new page entry.
- Translation Lookaside Buffer (TLB) is a solution that tries to reduce the effective access time.
- **Being a hardware, the access time of TLB is very less as compared to the main memory.**

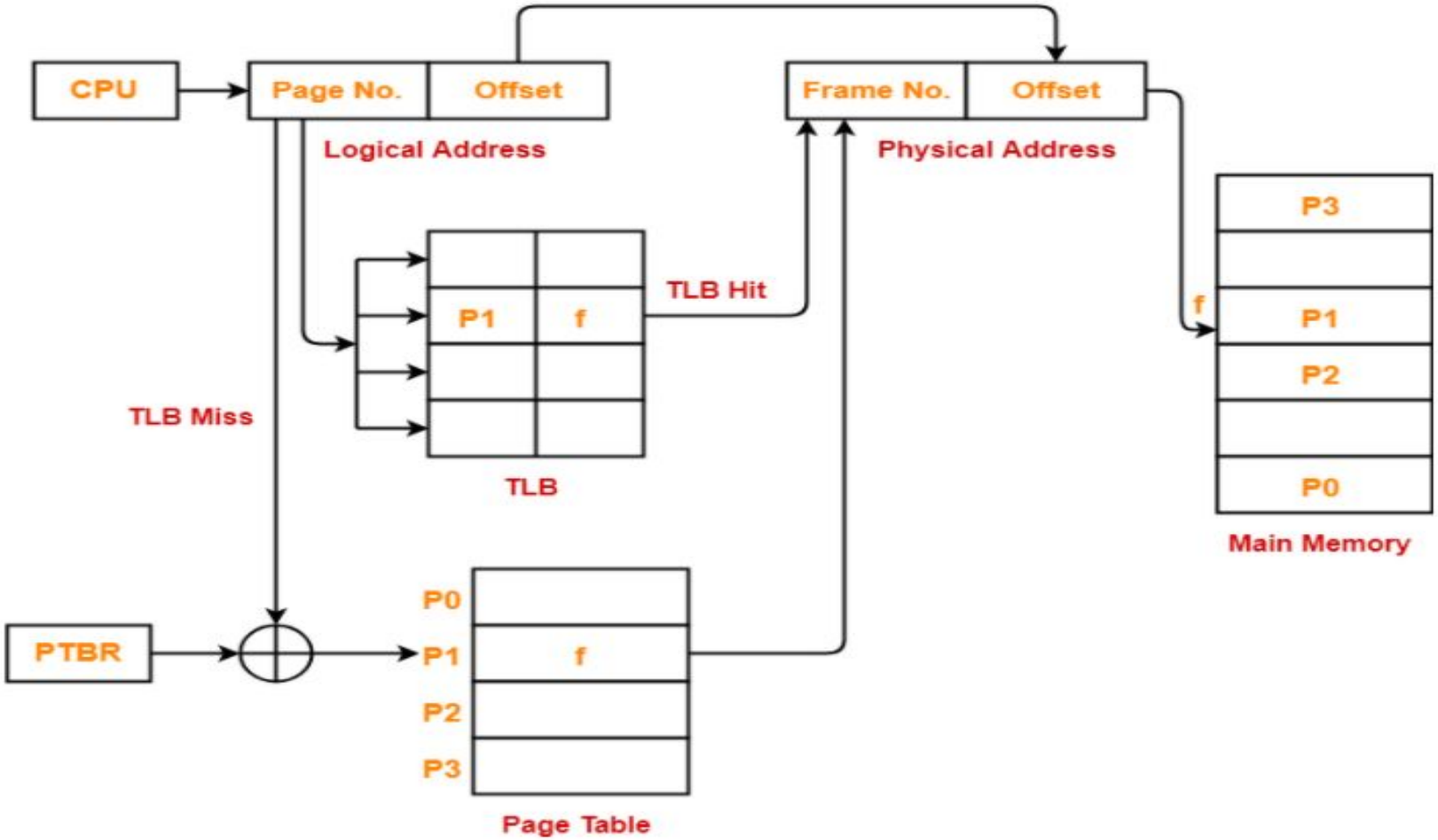
Structure-

Translation Lookaside Buffer (TLB) consists of two columns-

1. Page Number
2. Frame Number

Page Number	Frame Number

Translation Lookaside Buffer


CPU
Page No.
Offset
Logical Address
Frame No.
Offset
Physical Address
TLB Hit
P1
f
TLB Miss
TLB
P3
f
P1
P2
P0
Main Memory
PTBR
P0
P1
f
P2
P3
Page Table

EXAMPLE:

Main memory access=400MS

TLB=50MS

h=90%

WITHOUT TLB

=2MM

=2* 400

=800MS

WITH TLB

=hit[TLB+MM]+(1-hit)[TLB+MM+MM]

=0.9[50+400]+0.1[50+400+400]

=0.9*450+0.1*850

=405+85

=490MS

Segmentation

- 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.

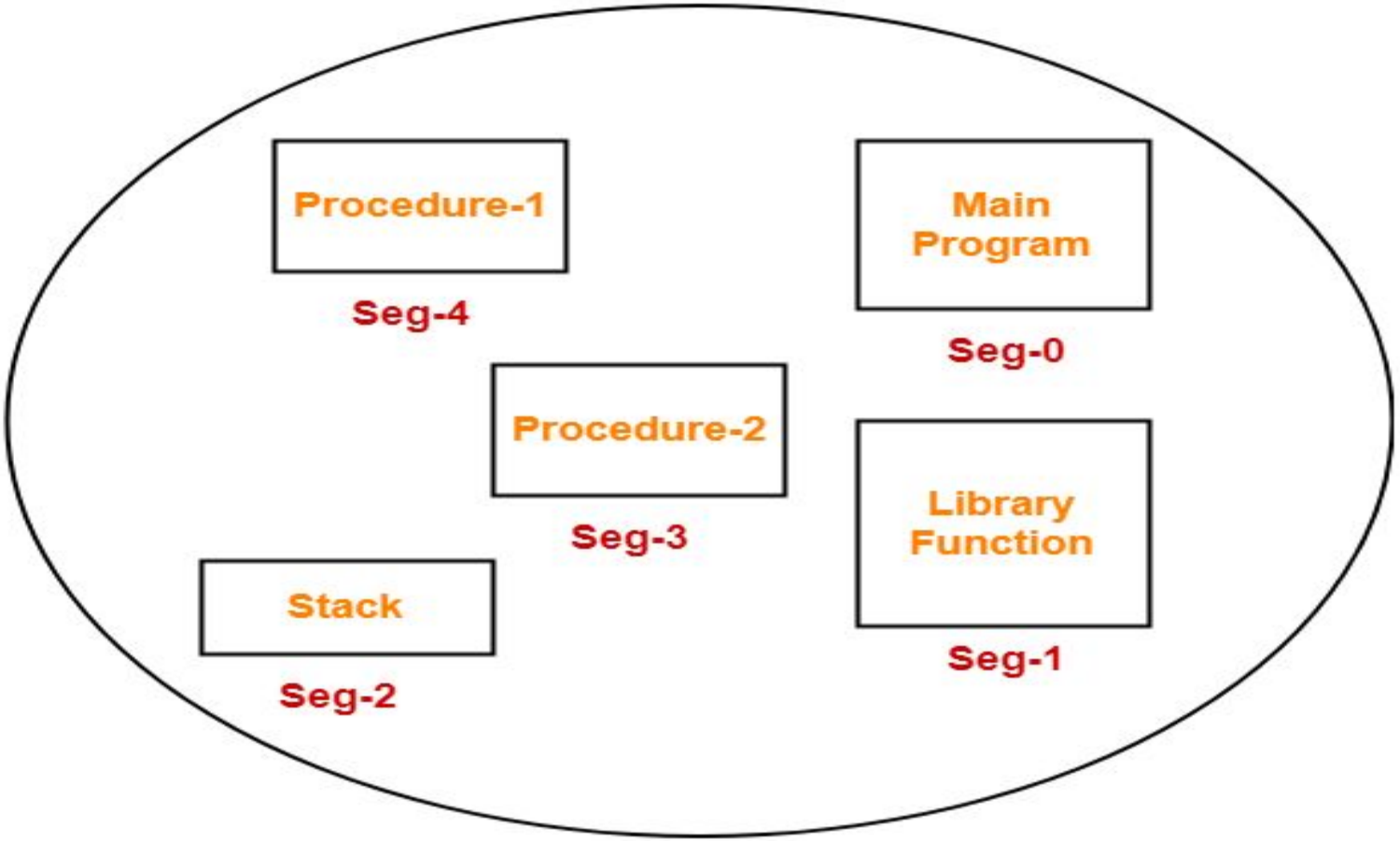
Characteristics-

- 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**.

o.h>
io.h>

esult;

n is

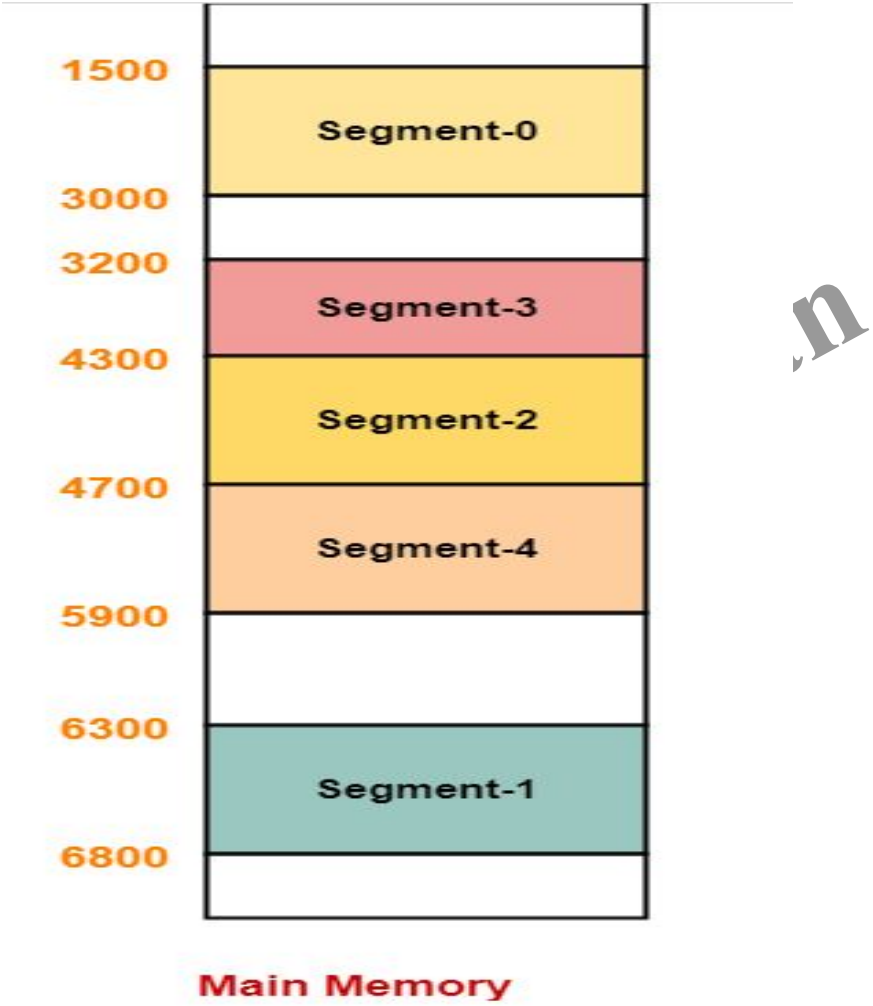


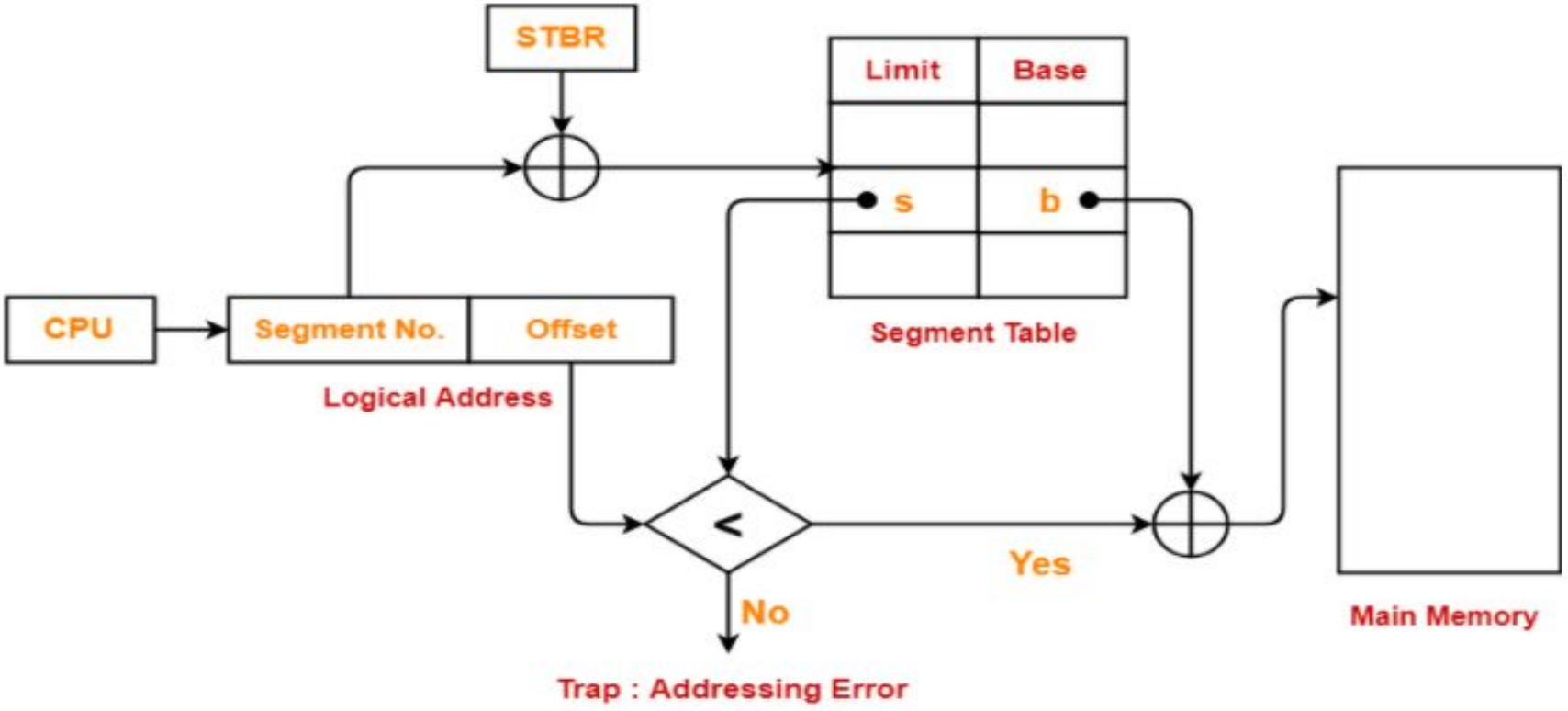
Segment Table-

- 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.

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

Segment Table


1500
Segment-0
3000
3200
Segment-3
4300
Segment-2
4700
Segment-4
5900
6300
Segment-1
6800
Main Memory



Translating Logical Address into Physical Address

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

Problem-

Consider the following segment table-

Segment No.	Base	Length
0	1219	700
1	2300	14
2	90	100
3	1327	580
4	1952	96

Which of the following logical address will produce trap addressing error?

1. 0, 430
2. 1, 11
3. 2, 100
4. 3, 425
5. 4, 95

Solution-

In a segmentation scheme, the generated logical address consists of two parts-

1. Segment Number
2. Segment Offset

We know-

- Segment Offset must always lie in the range [0, limit-1].
- If segment offset becomes greater than or equal to the limit of segment, then trap addressing error is produced.

Option-A: 0, 430-

Here,

- Segment Number = 0
- Segment Offset = 430

We have,

- In the segment table, limit of segment-0 is 700.
- Thus, segment offset must always lie in the range = [0, 700-1] = [0, 699]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1219 + 430 = 1649

Option-B: 1, 11-

Here,

- Segment Number = 1
- Segment Offset = 11

We have,

- In the segment table, limit of segment-1 is 14.
- Thus, segment offset must always lie in the range = [0, 14-1] = [0, 13]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 2300 + 11 = 2311

Option-C: 2, 100-

Here,

- Segment Number = 2
- Segment Offset = 100

We have,

- In the segment table, limit of segment-2 is 100.
- Thus, segment offset must always lie in the range = [0, 100-1] = [0, 99]

Now,

- Since generated segment offset does not lie in the above range, so request generated is invalid.
- Therefore, trap will be produced.

Option-D: 3, 425-

Here,

- Segment Number = 3
- Segment Offset = 425

We have,

- In the segment table, limit of segment-3 is 580.
- Thus, segment offset must always lie in the range = [0, 580-1] = [0, 579]

Now,

- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1327 + 425 = 1752

Option-E: 4, 95-

Here,

- Segment Number = 4
- Segment Offset = 95

We have,

- In the segment table, limit of segment-4 is 96.
- Thus, segment offset must always lie in the range = [0, 96-1] = [0, 95]

Now,

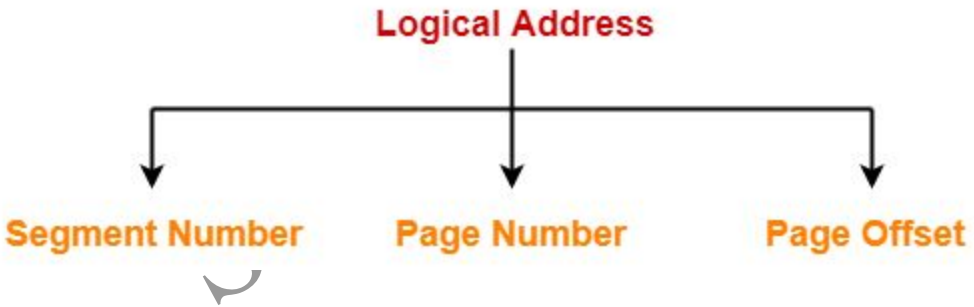
- Since generated segment offset lies in the above range, so request generated is valid.
- Therefore, no trap will be produced.
- Physical Address = 1952 + 95 = 2047

Segmented Paging-

- Segmented paging is a scheme that implements the combination of segmentation and paging.
- Process is **first divided into segments** and then **each segment is divided into pages.**
- These **pages are then stored in the frames** of main memory.
- A page table exists for each segment that keeps track of the frames storing the pages of that segment.
- Each page table occupies one frame in the main memory.
- Number of entries in the page table of a segment = Number of pages that segment is divided.
- A segment table exists that keeps track of the frames storing the page tables of segments.
- Number of entries in the segment table of a process = Number of segments that process is divided.
- The base address of the segment table is stored in the segment table base register

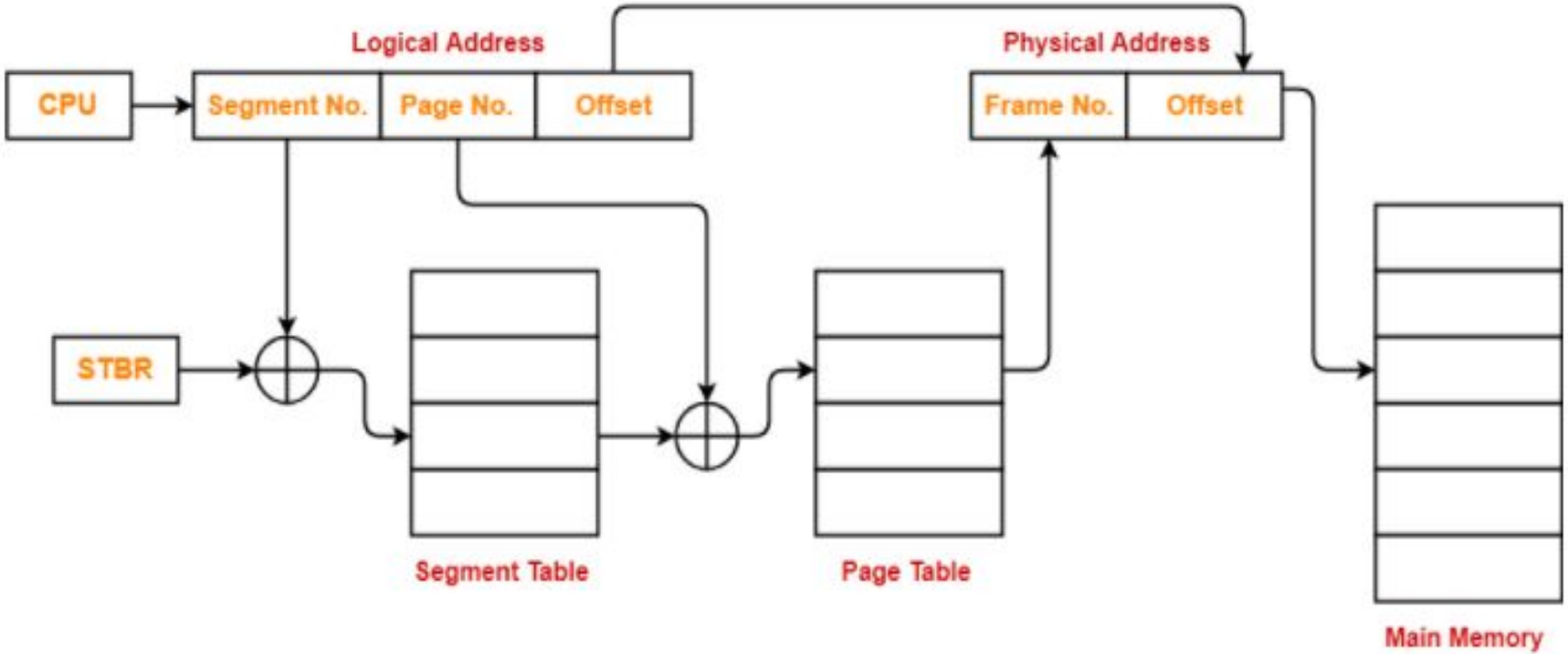
CPU generates a logical address consisting of three parts-

1. Segment Number
2. Page Number
3. Page Offset



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

- For the generated segment number, corresponding entry is located in the segment table.
- Segment table provides the frame number of the frame storing the page table of the referred segment.
- The frame containing the page table is located.
- For the generated page number, corresponding entry is located in the page table.
- Page table provides the frame number of the frame storing the required page of the referred segment.
- The frame containing the required page is located.



Translating Logical Address into Physical Address

Virtual Memory

Real and Virtual Memory

Real memory

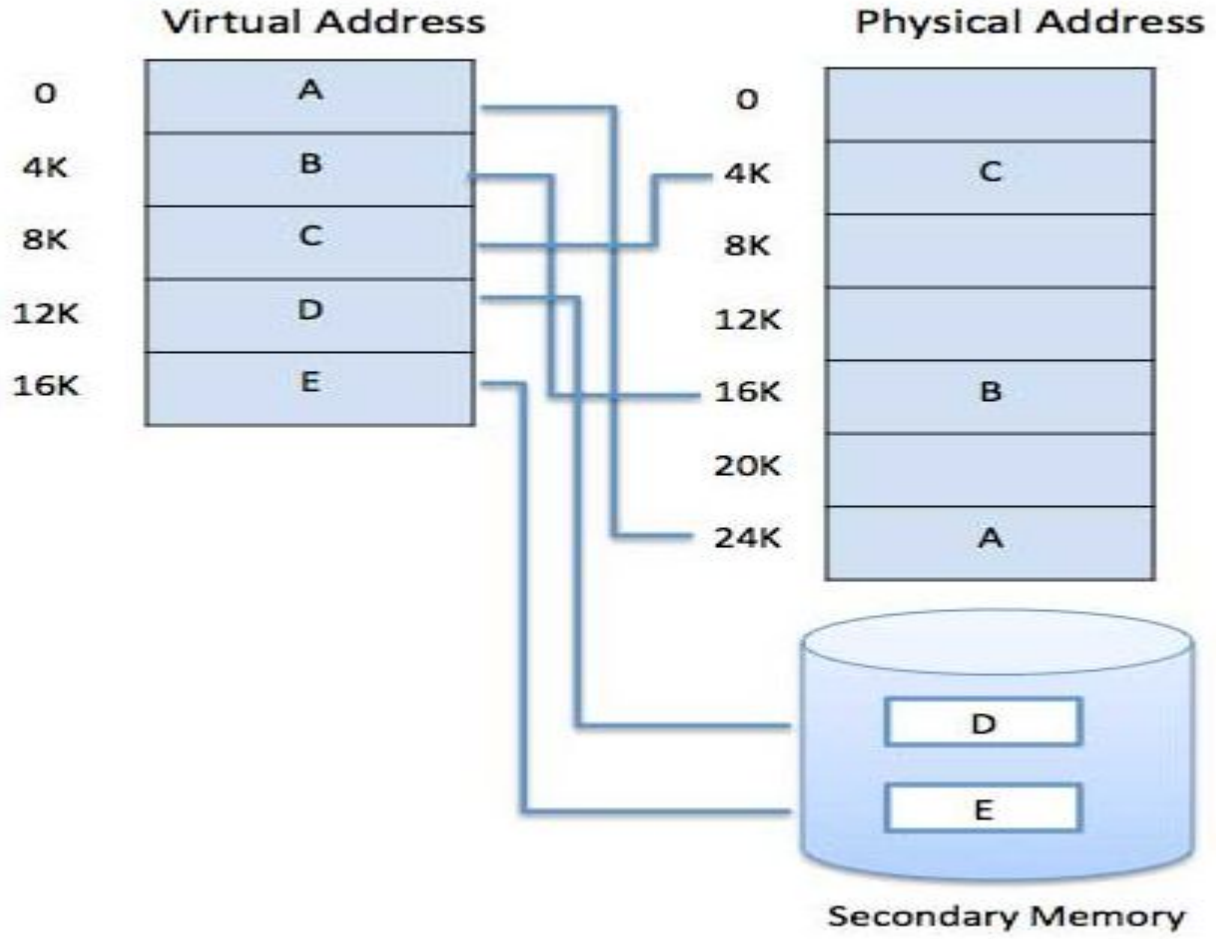
- main memory, the actual RAM

Virtual memory

- memory on disk
- allows for effective multiprogramming and relieves the user of tight constraints of main memory

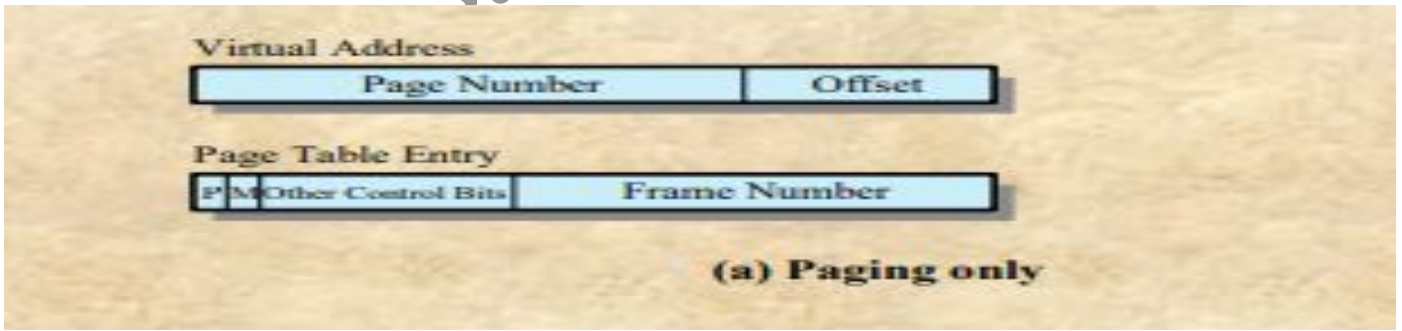
Virtual Memory

- 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.
- **Virtual memory is commonly implemented by demand paging**. It can also be implemented in a segmentation system.


Virtual Address
Physical Address
0
4K
8K
12K
16K
A
B
C
D
E
0
4K
8K
12K
16K
20K
24K
C
B
A
D
E
Secondary Memory

Paging in Virtual Memory

- The term virtual memory is usually associated with systems that employ paging
- Use of paging to achieve virtual memory was first reported for the **Atlas computer**
- Each process has its own page table each page table entry contains the frame number of the corresponding page in main memory



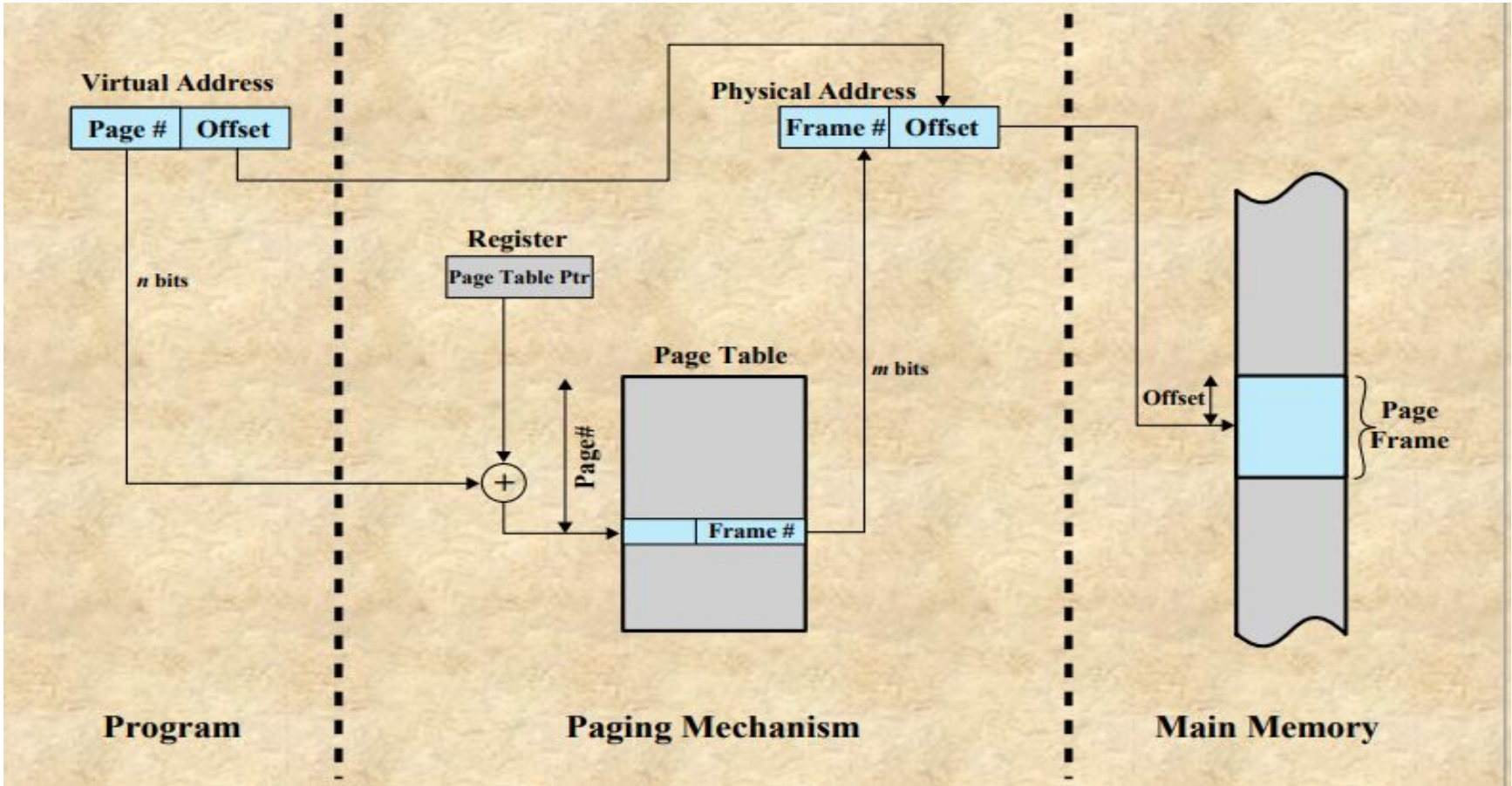
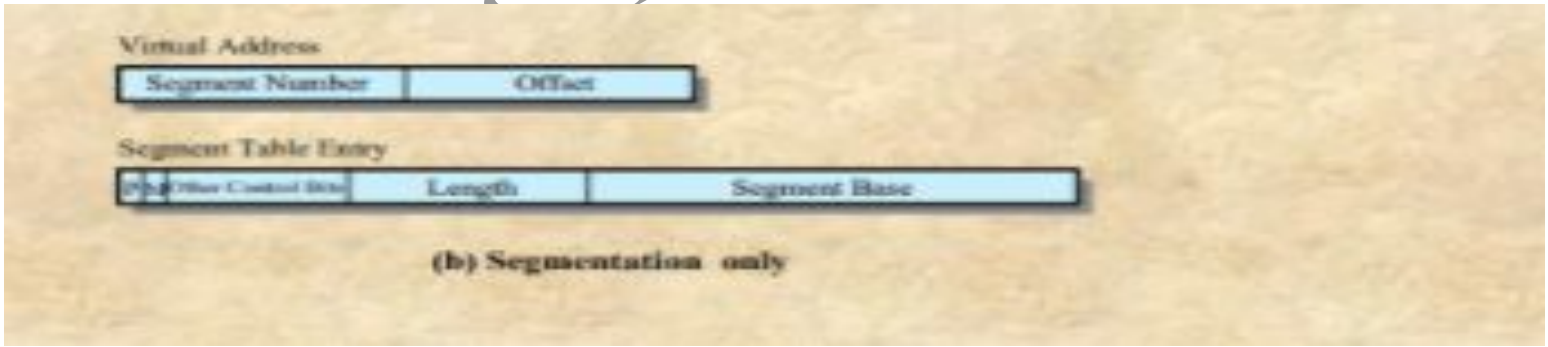


Figure 8.2 Address Translation in a Paging System

Segmentation in Virtual Memory

- Each segment table entry contains the starting address of the corresponding segment in main memory and the length of the segment
- A bit is needed to determine if **the segment is already in main memory**
- Another bit is needed to determine if the **segment has been modified** since it was loaded in main memory



(b) Segmentation only

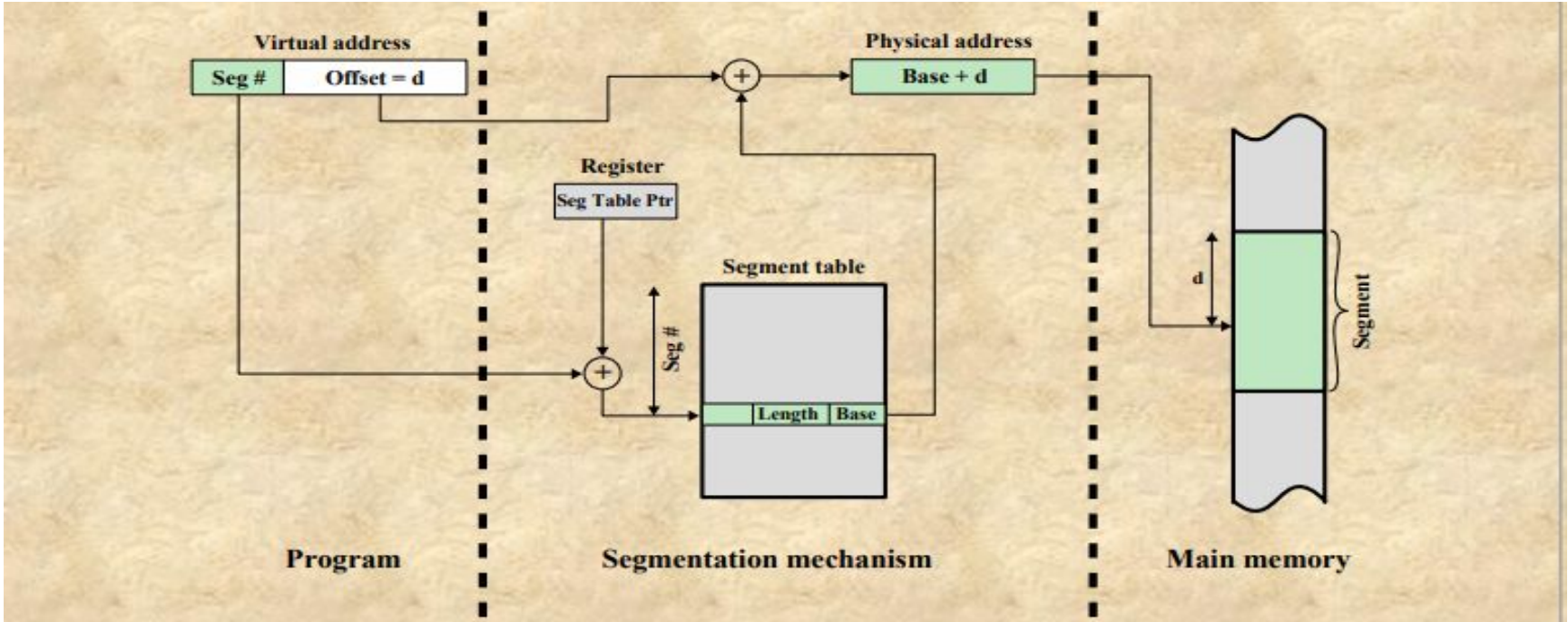


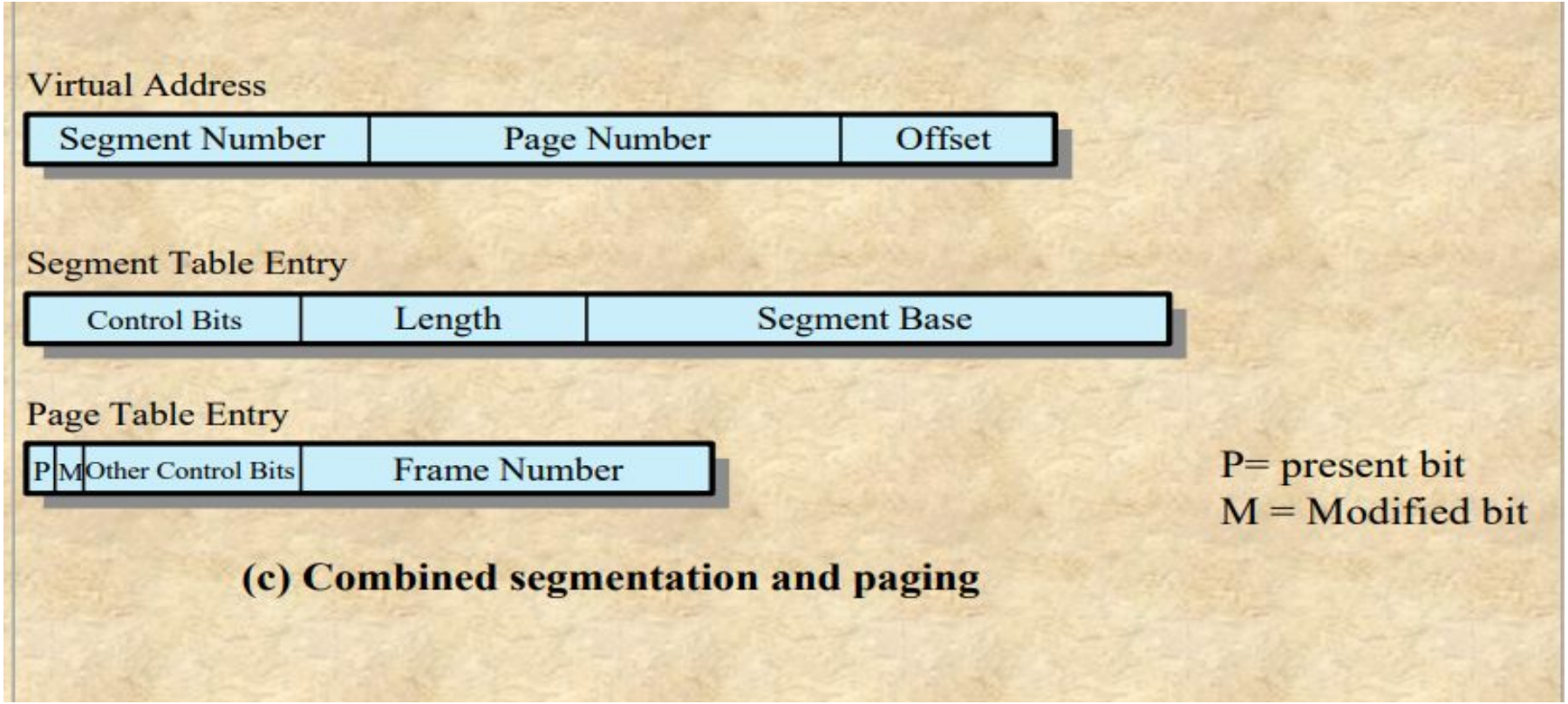
Figure 8.11 Address Translation in a Segmentation System

Combined Paging and Segmentation in Virtual Memory

In a combined paging/segmentation system a user's address space is broken up into a number of segments. Each segment is broken up into a number of fixed-sized pages which are equal in length to a main memory frame

- Segmentation is visible to the programmer
- Paging is transparent to the programmer

Combined Paging and Segmentation in Virtual Memory



(c) Combined segmentation and paging

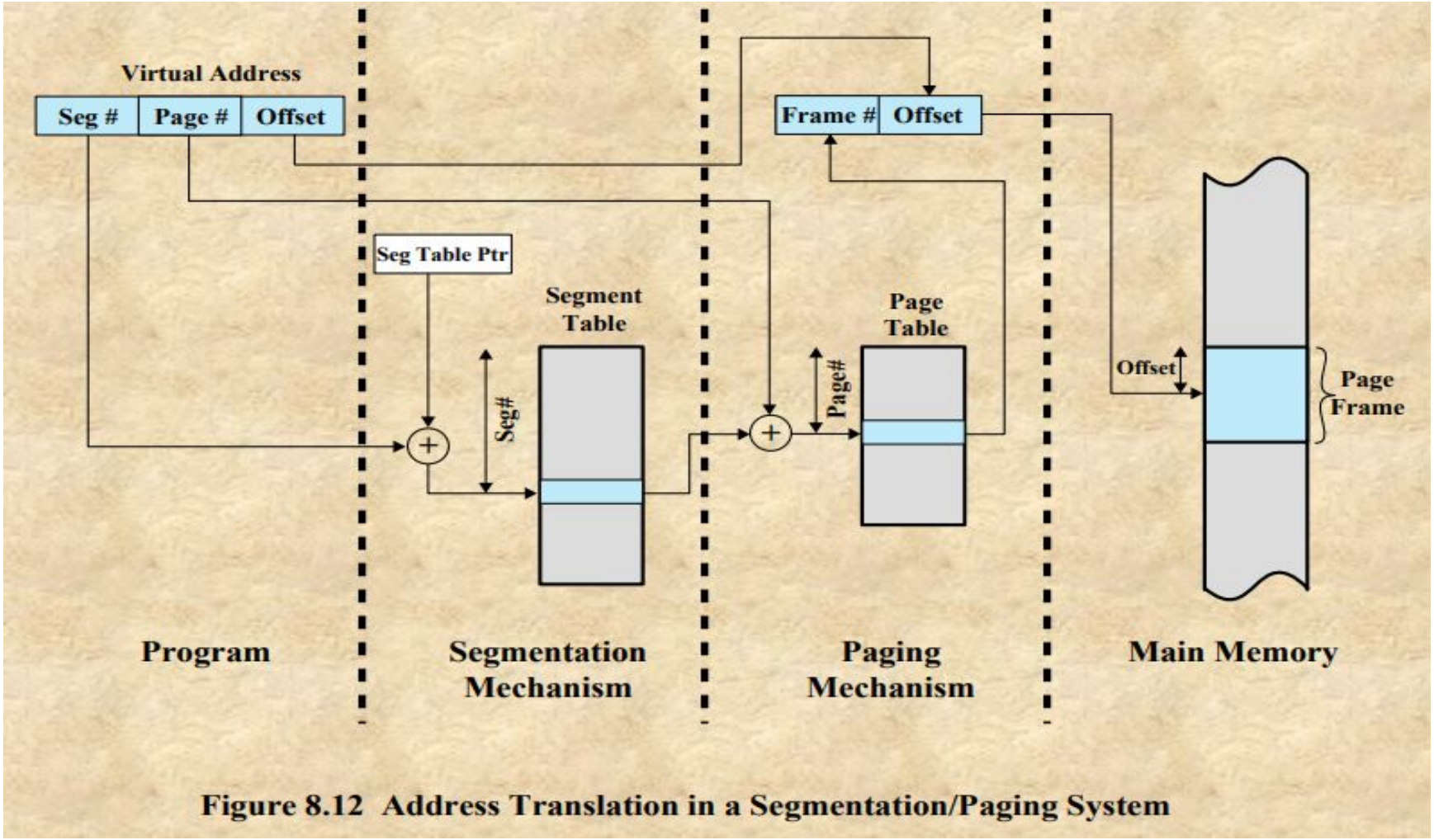


Figure 8.12 Address Translation in a Segmentation/Paging System

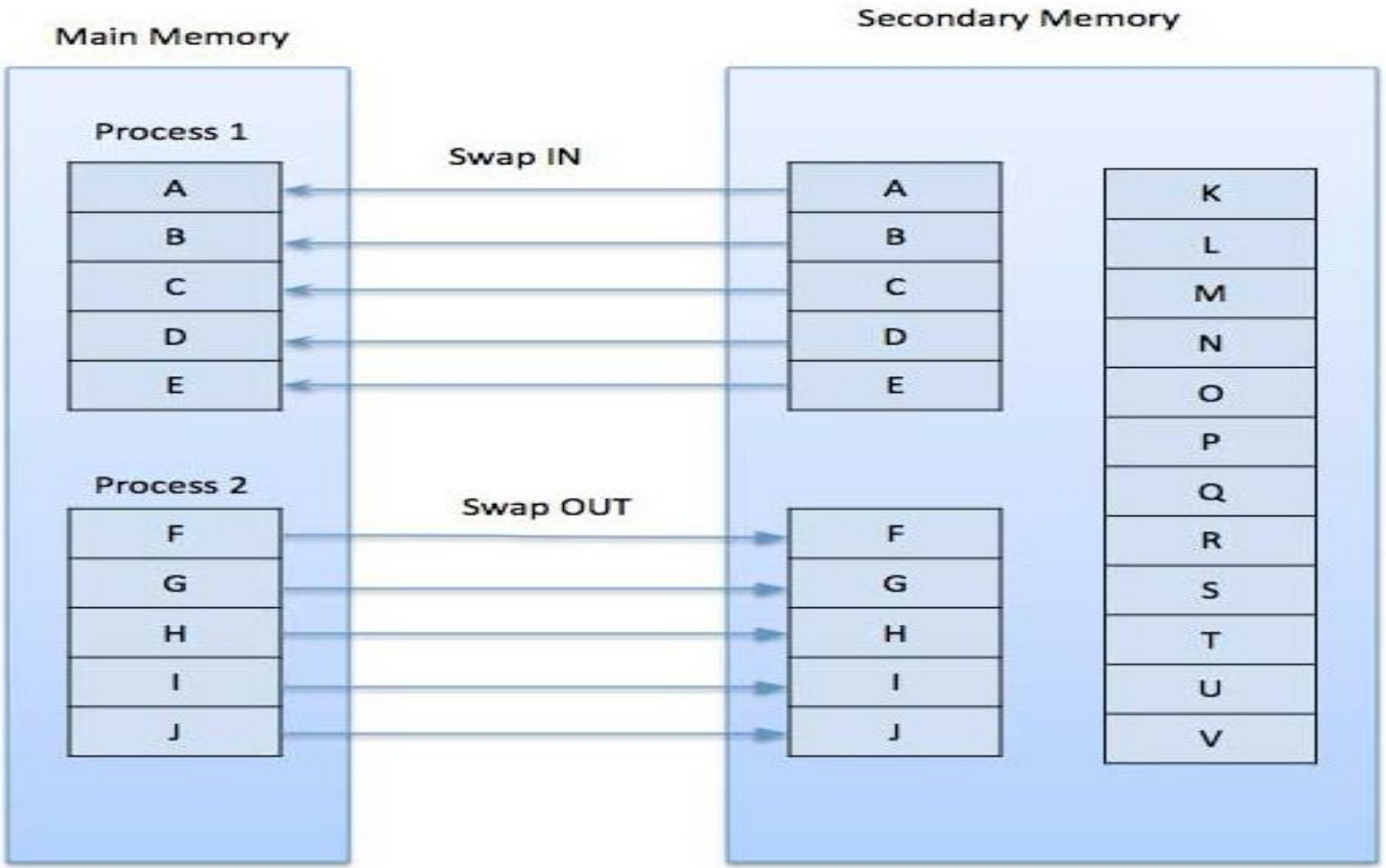
Demand Paging

- 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.

- Deciding, **which pages need to be kept in the main memory and which need to be kept in the secondary memory,** is going to be difficult because we cannot say in advance that a process will require a particular page at particular time.

- Therefore, to overcome this problem, there is a concept called Demand Paging is introduced.

- **Demand Paging is a method in which a page is only brought into main memory when the CPU requests it.**


Main Memory
Secondary Memory
Process 1
A
B
C
D
E
Swap IN
A
B
C
D
E
K
L
M
N
O
P
Q
R
S
T
U
V
Process 2
F
G
H
I
J
Swap OUT
F
G
H
I
J

Page Replacement-

- Page replacement is a process of **swapping out an existing page from the frame of a main memory and replacing it with the required page.**
- Page Replacement Algorithm decides which page to remove, also called swap out when a new page needs to be loaded into the main memory.
- Page Replacement happens when a requested page is not present in the main memory and the available space is not sufficient for allocation to the requested page.

*If a process requests for page and that page is found in the main memory then it is called page hit, otherwise page miss or pa**ge fault**.*

Reference String

The string of memory references is called reference string.

Page Replacement Algorithms-

1. FIFO Page Replacement Algorithm
2. LRU Page Replacement Algorithm
3. Optimal Page Replacement Algorithm

FIFO Page Replacement Algorithm-

- As the name suggests, this algorithm works on the principle of "**First in First out**".
- It replaces the oldest page that has been present in the main memory for the longest time.
- It is implemented by keeping track of all the pages in a queue.

Example

Consider the page reference string of size 12: 1, 2, 3, 4, 5, 1, 3, 1, 6, 3, 2, 3 with frame size 4(i.e. maximum 4 pages in a frame).

1	2	3	4	5	1	3	1	6	3	2	3
1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	6	6	6	6
			4	4	4	4	4	4	3	3	3
M	M	M	M	M	M	H	H	M	M	M	H

M = Miss
H = Hit

Total Page Fault = 9

Page Fault ratio = 9/12 i.e. total miss/total possible cases

LRU Page Replacement Algorithm-

- As the name suggests, this algorithm works on the principle of "**Least Recently Used**".
- It replaces the page that has not been referred by the CPU for the longest time.

Example

Consider the page reference string of size 12: 1, 2, 3, 4, 5, 1, 3, 1, 6, 3, 2, 3 with frame size 4(i.e. maximum 4 pages in a frame).

Example

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	5	5	5	5	5	5	2	2
	2	2	2	2	1	1	1	1	1	1	1
		3	3	3	3	3	3	3	3	3	3
			4	4	4	4	4	6	6	6	6
M	M	M	M	M	M	H	H	M	H	M	H

M = Miss
H = Hit

Total Page Fault = 8

Page Fault ratio = 8/12

Optimal Page Replacement Algorithm-

- This algorithm replaces the page that will not be referred by the CPU in future for the longest time.
- It is practically impossible to implement this algorithm.
- This is because the pages that will not be used in future for the longest time can not be predicted.
- However, it is the best known algorithm and gives the least number of page faults.
- Hence, it is used as a performance measure criterion for other algorithms.

Example

1	2	3	4	5	1	3	1	6	3	2	3

1	1	1	1	1	1	1	1	6	6	6	6
	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	3	3	3	3	3
			4	5	5	5	5	5	5	5	5
M	M	M	M	M	H	H	H	M	H	H	H

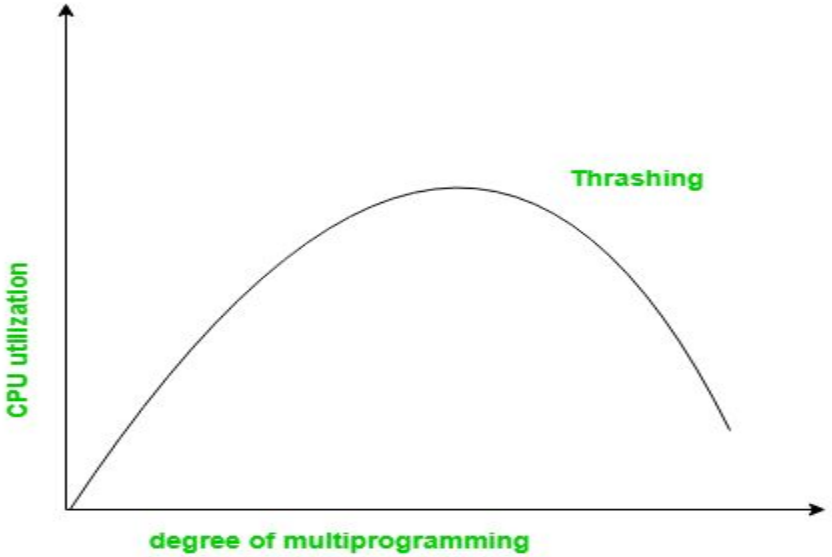
M = Miss
H = Hit

Total Page Fault = 6

Page Fault ratio = 6/12

Thrashing

- Thrashing occurs when a **Process** is spending more time in paging or **Swapping** activities rather than its execution.
- In thrashing, state CPU is so much busy in swapping that it cannot respond to user program as much as it required.



Causes of Thrashing

Thrashing affects the performance of execution.

- Initially when the CPU utilization is low, then process scheduling mechanism, loads many processes into the memory at the same time so that **Degree of Multiprogramming** can be increased.
- So now in this situation, we have more number of processes in memory as compare to the available number of frames in memory. Allocation of the limited amount of frames to each process.
- When any higher priority **Process** arrives in memory and if the frame is not freely available at that time then the other process that occupied the frame which resides in the frame will move to secondary storage and this free frame is now allocated to higher priority process.

As the degree of multiprogramming increases to increase the CPU Utilization then after some time it causes Thrashing.

Effect of Thrashing

Whenever thrashing starts, operating system tries to apply either **Global page replacement** Algorithm or **Local page replacement** algorithm.

Global Page Replacement

Since global page replacement **can access to bring any page,** it tries to bring more pages whenever thrashing found. But what actually will happen is, due to this, no process gets enough frames and by result thrashing will be increase more and more. So global page replacement algorithm is not suitable when thrashing happens.

Local Page Replacement

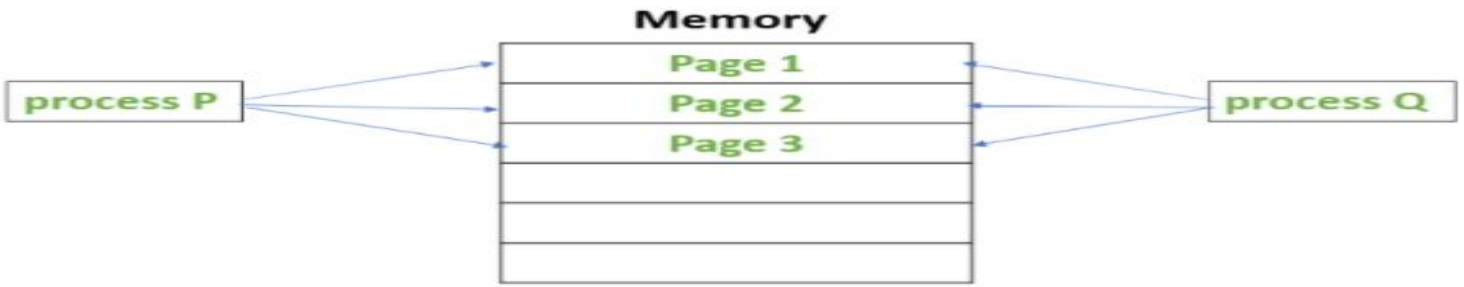
Unlike global page replacement algorithm, local page replacement **will select pages which only belongs to that process.** So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. So local page replacement is just alternative than global page replacement in thrashing scenario.

Copy on Write

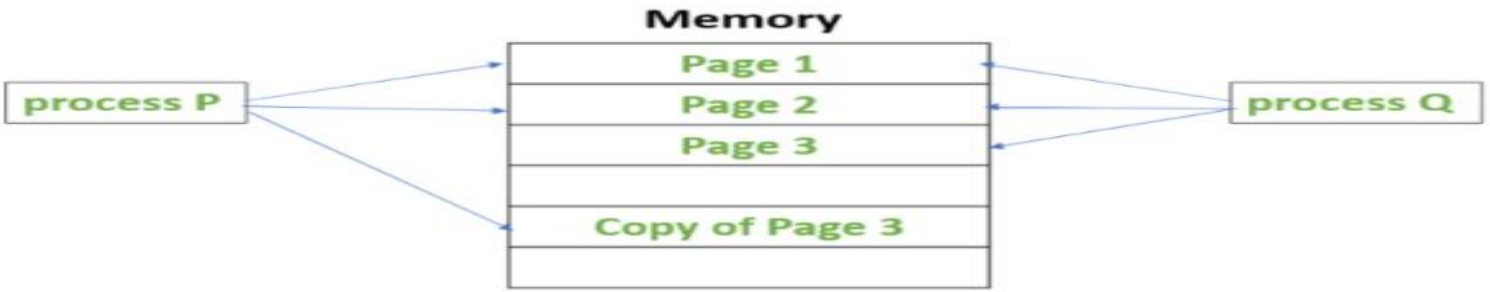
- **Copy on Write** or simply COW is a resource management technique. One of its main use is in the implementation of the fork system call in which it shares the virtual memory(pages) of the OS.
- In UNIX like OS, fork() system call **creates a duplicate process of the parent process which is called as the child process.**
- copy-on-write is that when a parent process creates a child process then **both of these processes initially will share the same pages** in memory and these shared pages will be marked as copy-on-write .
- which means that **if any of these processes will try to modify the shared pages then only a copy of these pages will be created and the modifications will be done on the copy of pages** by that process and thus not affecting the other process.

Suppose, there is a process P that creates a new process Q and then process P modifies page 3.

The below figures shows what happens before and after process P modifies page 3.



Before process P modifies Page 3



After process P modifies Page 3

THANK YOU