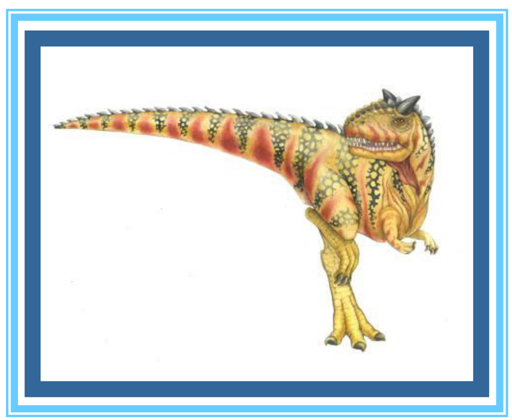
# Memory Management



# Contents

# *Memory Management*

- *Efficient utilization of main memory* ***(Goal)***.
- *Method of managing* ***primary memory***.
- *Each approach has its own advantages and disadvantages.*

# Memory Management

- ***Main memory*** *and the* ***registers*** *built into the processor itself are the only general-purpose storage that the* ***CPU can access directly****.*
- *CPU access to primary memory is many times slower than access to registers, but* ***caches*** *help speed things up.*
- *Any instructions in execution, and any data being used by the instructions, must be in one of these direct-access storage devices*
- *We are not only concerned with the relative speed of accessing physical memory, we must ensure correct operation.*

# *Memory Management*

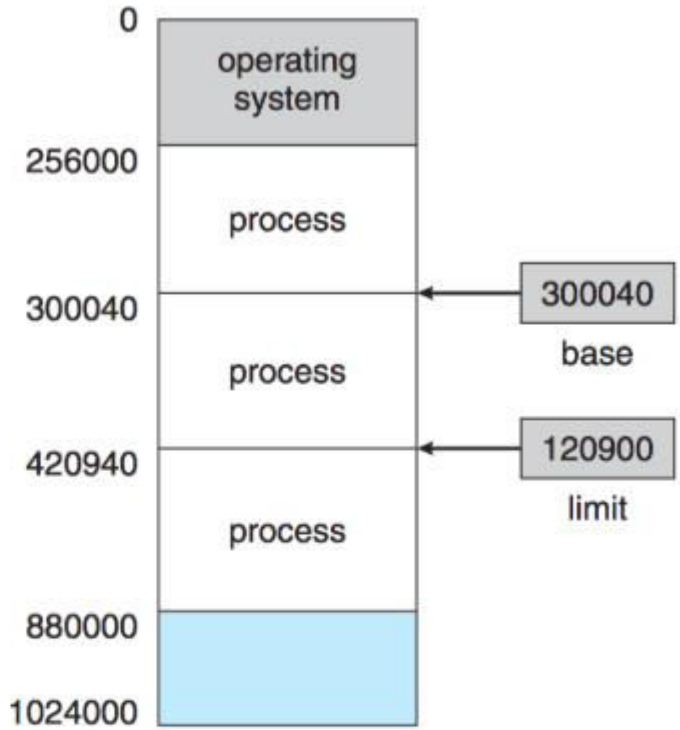
- *For proper system operation we must protect the operating system from access by user processes and should protect user processes from one another.*
- ***Hardware** must **protect user processes** from one another (because OS doesn't intervene between CPU and its memory access).*
- *The **performance penalty** would be very extreme if designers assigned this task to the operating system - it would simply not be practical.*

# *Memory Management*

- *Each process must have a separate memory space which protects the processes from each other*
- *We can provide this protection by using two registers, usually a* ***base*** *and a* ***limit****.*

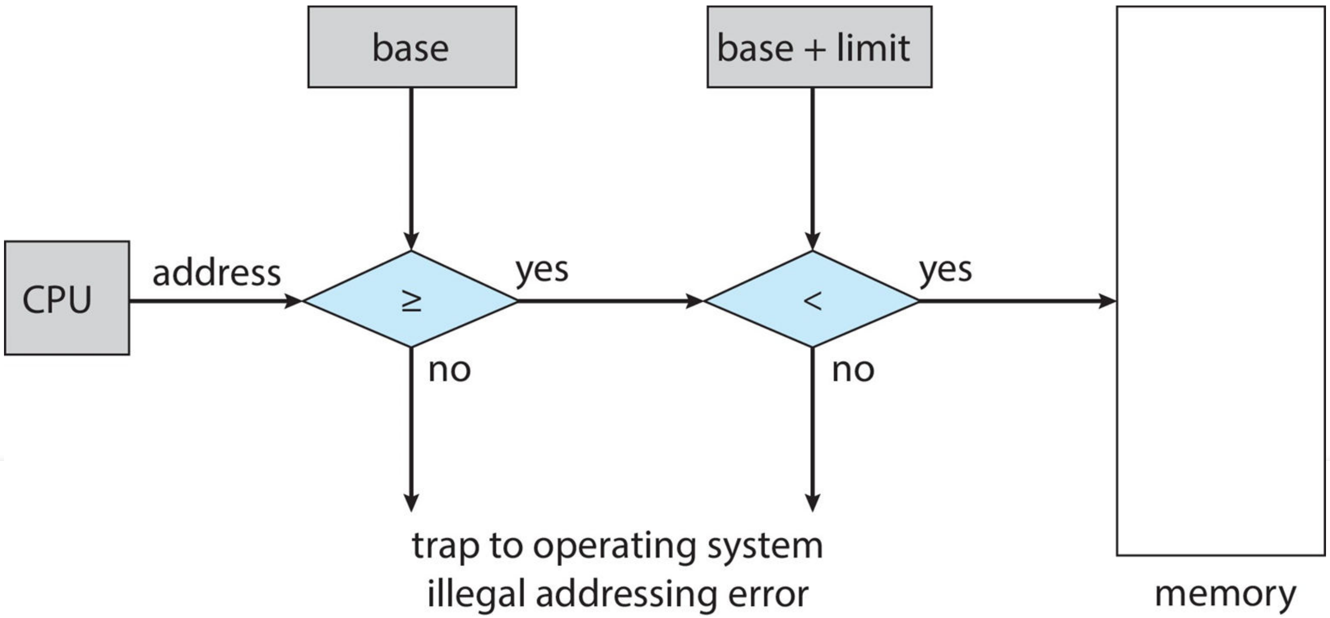
# Basic Hardware

- *The OS allocates one **contiguous span** of primary memory to a process P.*
- *The **base register** contains the **lowest address** allocated to P.*
- *The **limit register** contains the number of bytes **(size)** in the allocation.*
- *Using the values in the base and limit registers, hardware checks every address generated in user mode.*

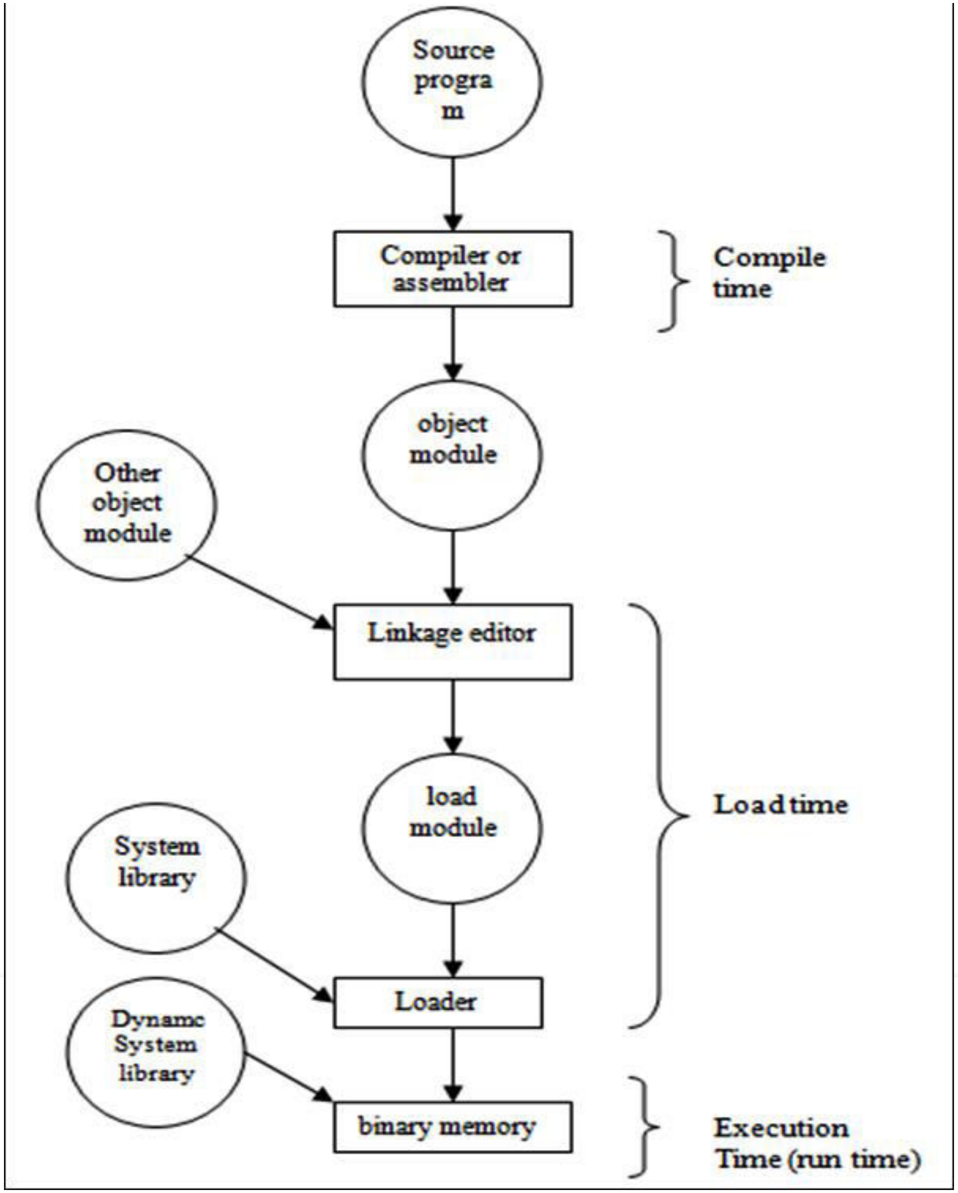


# Basic Hardware

- *Any attempt by a program executing in user mode to access operating-system memory or other users' memory results in a **trap** to the operating system, which treats the attempt as a fatal error*
- *This scheme prevents a user program from **(accidentally or deliberately)** modifying the code of either the operating system or other users*
- *The base and limit registers can be loaded only by the operating system **(privileged instruction).***



# Program Execution Steps

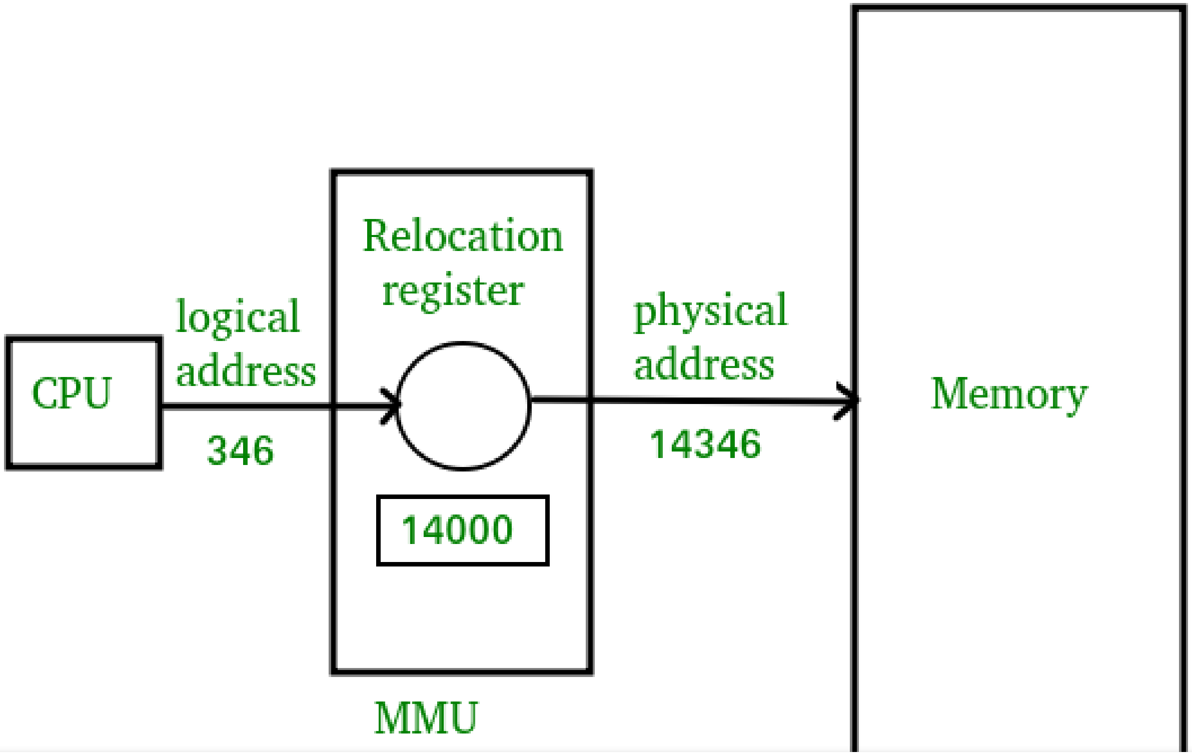


# *Memory-Management Unit (MMU)*

- ***Hardware device*** *that maps virtual address to physical address.*
- *In* ***MMU*** *scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.*
- *The user program deals with logical addresses; it never sees the real physical addresses.*

# *Memory Management Unit*

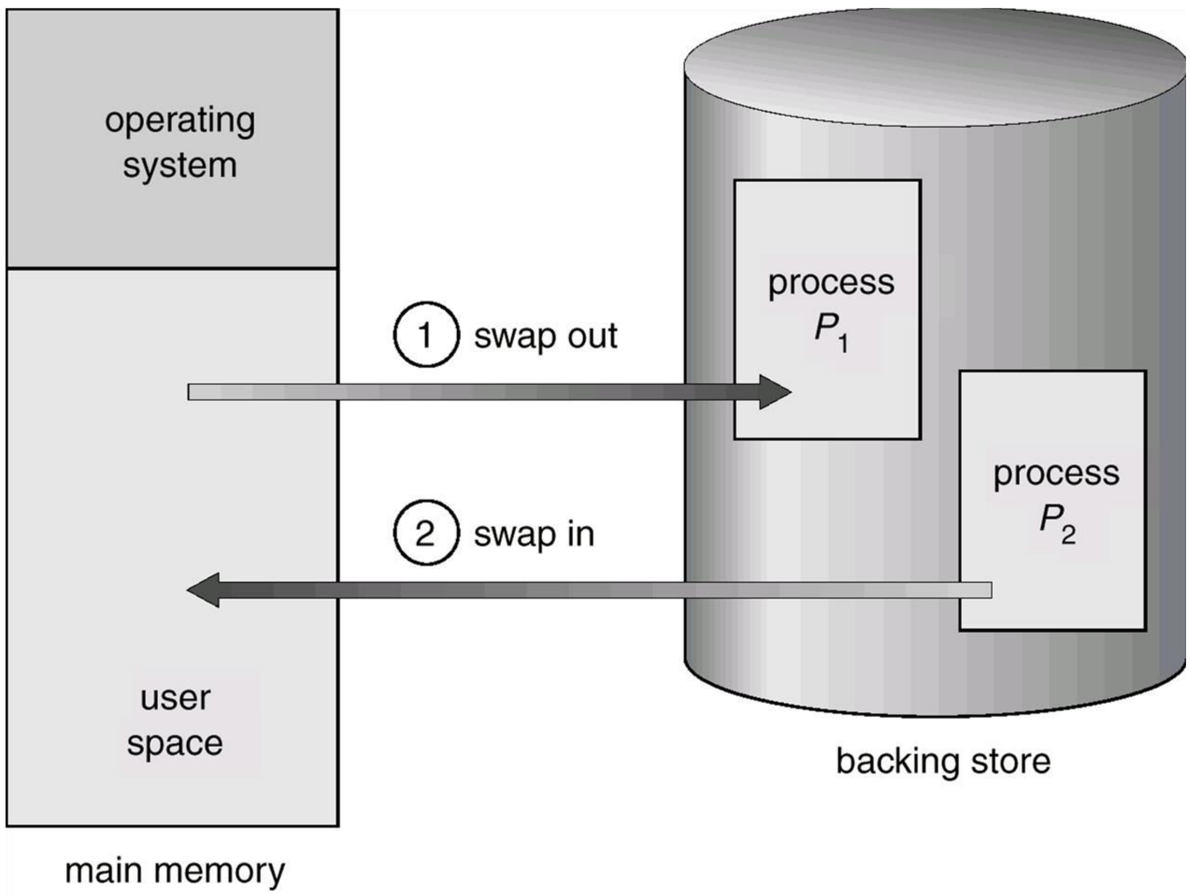
A ***hardware component*** referred to as the ***Memory Management Unit (MMU)*** is responsible for ***translating*** a *virtual address to a physical address*.



| BASIS FOR COMPARISON | LOGICAL ADDRESS | PHYSICAL ADDRESS |
|---|---|---|
| **Basic** | It is the virtual address generated by CPU | The physical address is a location in a memory unit. |
| **Address Space** | Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space. | Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address. |
| **Visibility** | The user can view the logical address of a program. | The user can never view physical address of program |
| **Access** | The user uses the logical address to access the physical address. | The user can not directly access physical address. |
| **Generation** | The Logical Address is generated by the CPU | Physical Address is Computed by MMU |
| **Rebooting** | Logic address is erased when the system is rebooted. | Physical address is not affected when the system is rebooted. |

# *Schematic View of Swapping*

***Swapping makes it possible for the total physical address space of all processes to exceed the real physical memory of the system. Increase Degree of multiprogramming.***



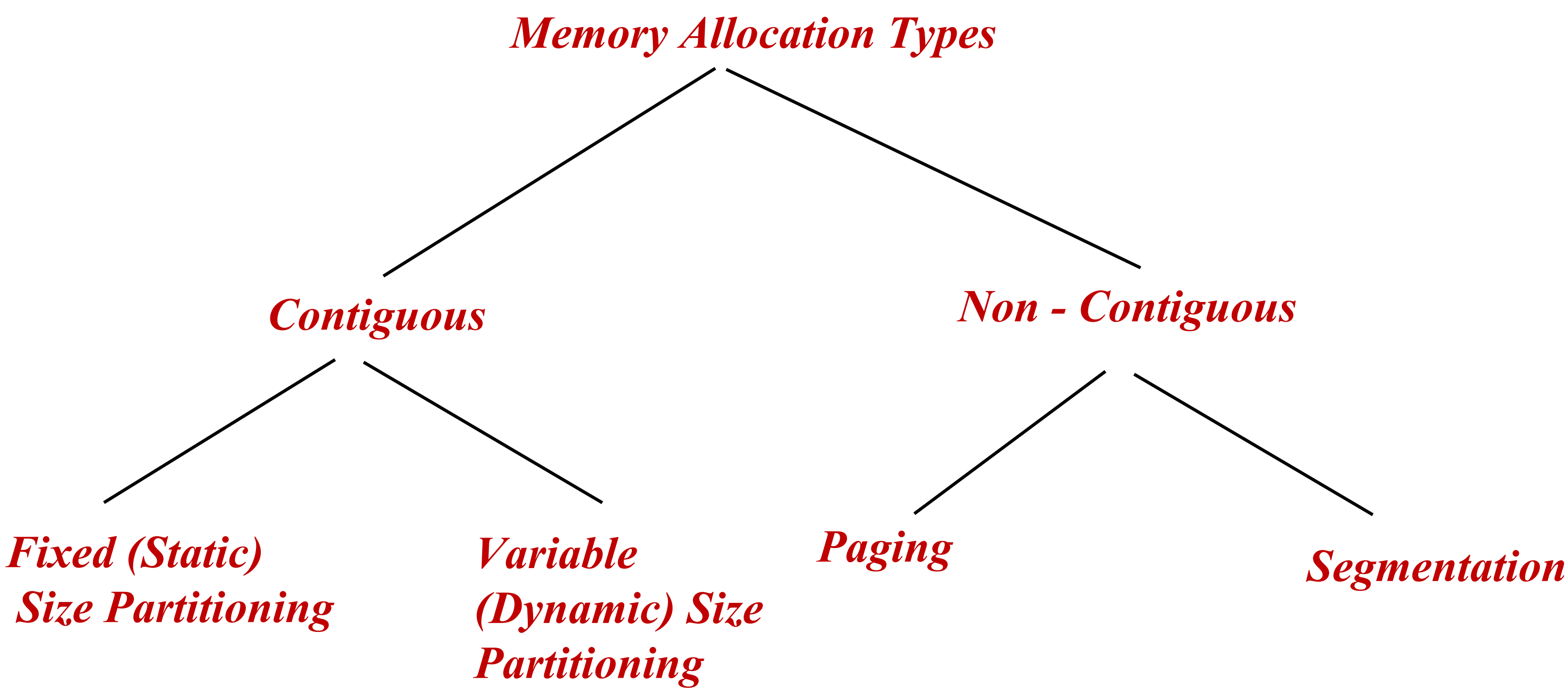
# Swapping

- *A process can be* ***swapped*** *temporarily out of memory to a* ***backing store****, and then* ***brought back*** *into memory for continued execution.*
- ***Backing store*** *– fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.*
- ***Roll out, roll in*** *– swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed.*
- *Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.*

## Swapping

- *Before swapping a process we must ensure it is idle not waiting for I/O. If we swapped out a process p1 waiting for I/O and swap in process p2 then I/O operation will try to access memory that now belongs to p2. Solution to this problem is either do* ***double buffering*** ***(use two buffers to speed up a computer that can overlap I/O with processing),*** *execute I/O operations in OS buffer only, Transfer between operating system buffer and process memory will occur only when p1 swapped in or never swap a process with pending I/O*

# Types of Memory Allocation



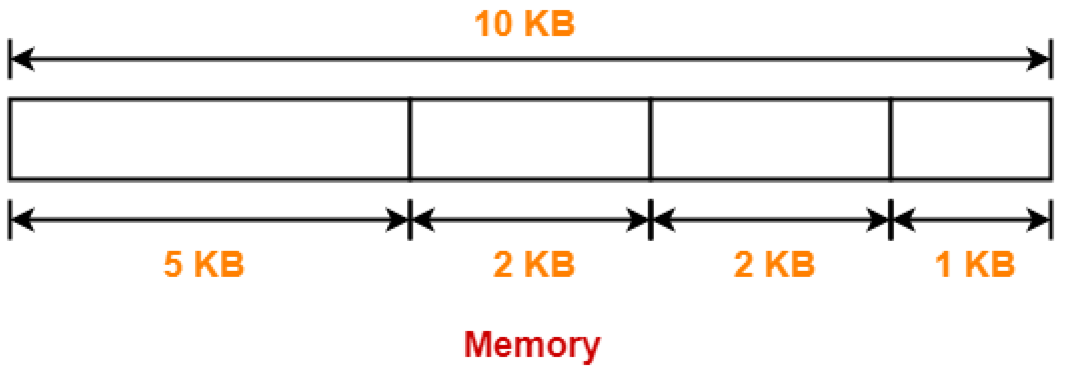
# Contiguous Allocation

- ***Contiguous memory allocation:*** *Each process is contained in* ***single section of memory*** *that is contiguous.*
- ***Non contiguous memory allocation:*** *a process will acquire the memory space but it is not at one place it is at the different locations according to the process requirement.*

# Contiguous Allocation (Cont.)

- ***Fixed (Static) Size Partitioning:***
  - *In this technique, main memory is pre-divided into fixed size partitions, **size of partitions could be different**.*
  - *The size of each partition is fixed and can not be changed.*
  - *Each partition is allowed to store **exactly one process**.*
  - *Degree of multiprogramming depends on number of partitions.*
  - *Internal Fragmentation.*

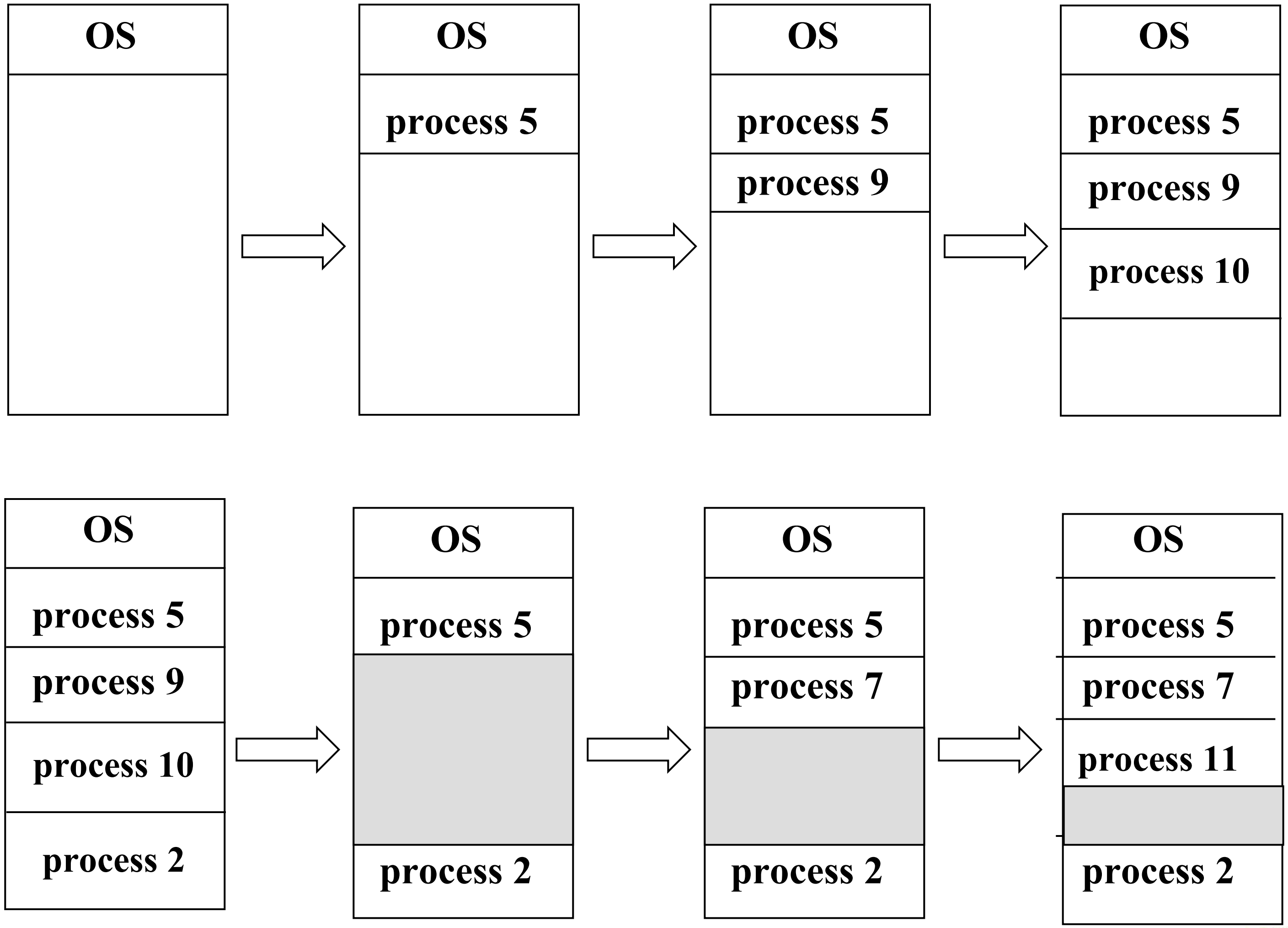
# Contiguous Allocation (Cont.)



# Contiguous Allocation (Cont.)

- ***Variable (dynamic) Size Partitioning:***
  - *It performs the allocation dynamically.*
  - *Initially all memory is available for the processes, it is considered as one large block of available memory, a* ***Hole****.*
  - *When a process arrives, a partition of size equal to the size of process is created, Then, that partition is allocated to the process.*
  - *Operating system maintains information about: a) allocated partitions b) free partitions (hole*

# Contiguous Allocation (Cont.)

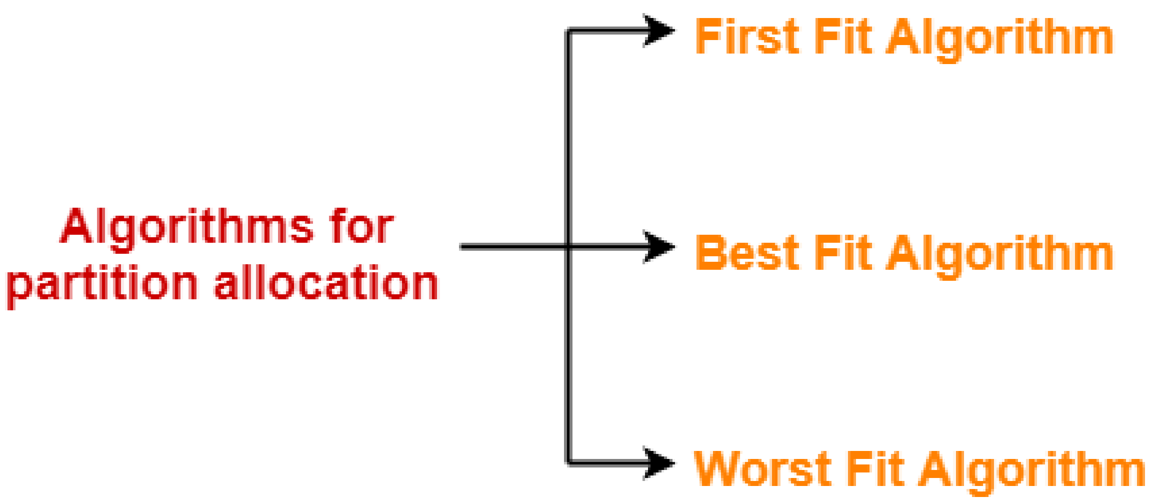


# Contiguous Allocation (Cont.)

- *The processes arrive and leave the main memory.*
- *As a result, holes of different size are created in the main memory.*
- *These holes are allocated to the processes that arrive in future.*

# Partition Allocation Algorithms

*Partition allocation algorithms are used to decide which hole should be allocated to the arrived process..*



# *Partition Allocation Algorithms*

## ***First-fit Algorithm*:**

- *This algorithm starts scanning the partitions serially from the starting.*
- *When an empty partition that is big enough to store the process is found, it is allocated to the process.*
- *Obviously, the partition size has to be greater than or at least equal to the process size.*

# Partition Allocation Algorithms

***Best-fit Algorithm**:*

- *This algorithm first scans all the empty partitions.*
- *It then allocates the smallest size partition to the process.*

***Worst-fit Algorithm**:*

- *This algorithm first scans all the empty partitions.*

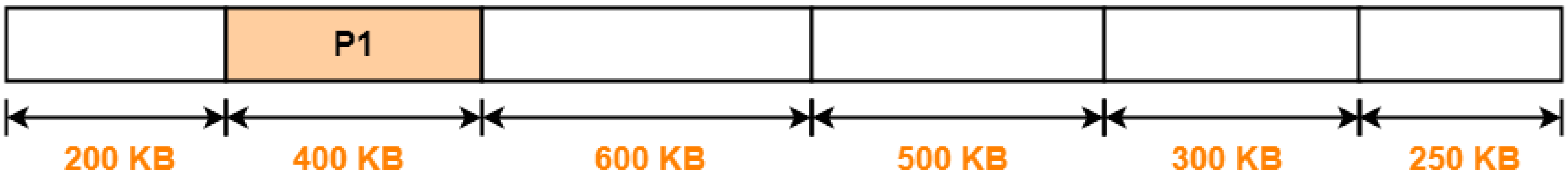
# Static Size Partitioning Problem

*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 P1- 357 KB, P2- 210 KB, P3- 468 KB and P4-491 KB in that order.*

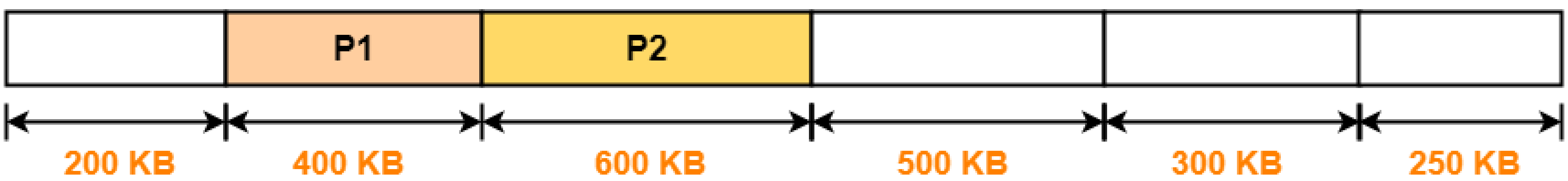
*Perform the allocation of processes using-*

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

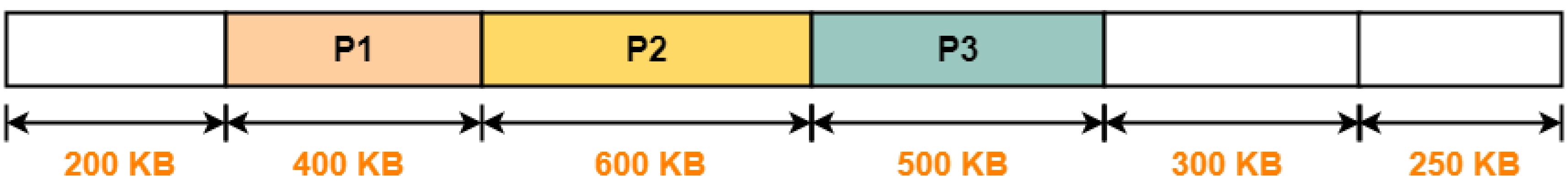
# *First Fit*




Main Memory



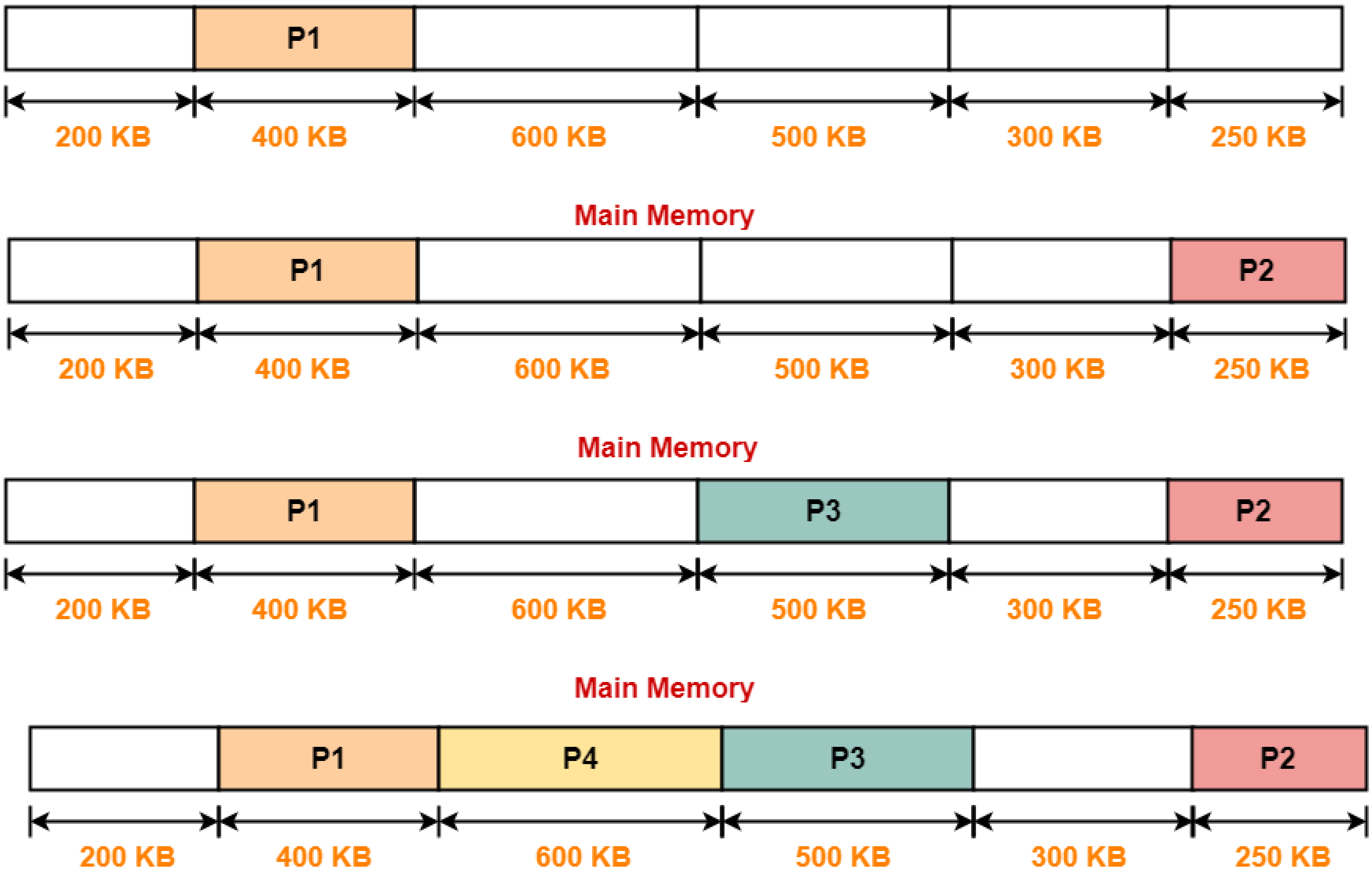

Main Memory




Main Memory

- *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.*

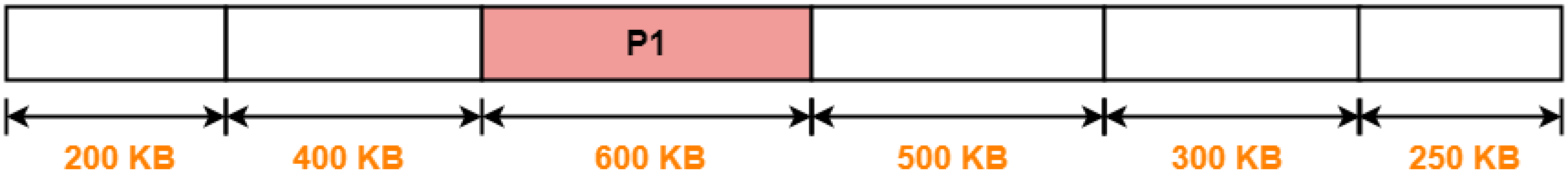
# Best Fit



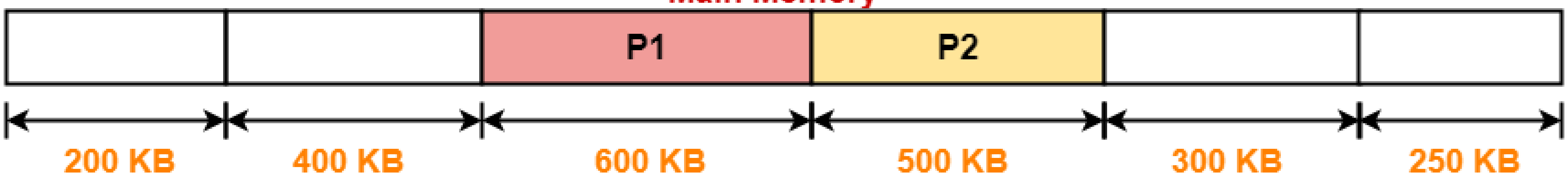
P1
200 KB
400 KB
600 KB
500 KB
300 KB
250 KB
Main Memory
P1
P2
200 KB
400 KB
600 KB
500 KB
300 KB
250 KB
Main Memory
P1
P3
P2
200 KB
400 KB
600 KB
500 KB
300 KB
250 KB
Main Memory
P1
P4
P3
P2
200 KB
400 KB
600 KB
500 KB
300 KB
250 KB

# Worst Fit




Main Memory




Main Memory

- Process P3 and Process P4 can not be allocated the memory.
- This is because no partition of size greater than or equal to the size of process P3 and process P4 is available.

# Important Points about static Partitioning

- *Best Fit Algorithm works best, because space left after the allocation inside the partition is of very small size. Thus, internal fragmentation is least.*

- *Worst Fit Algorithm works worst. This is because space left after the allocation inside the partition is of very large size. Thus, internal fragmentation is maximum.*

# Advantages of Static Partitioning

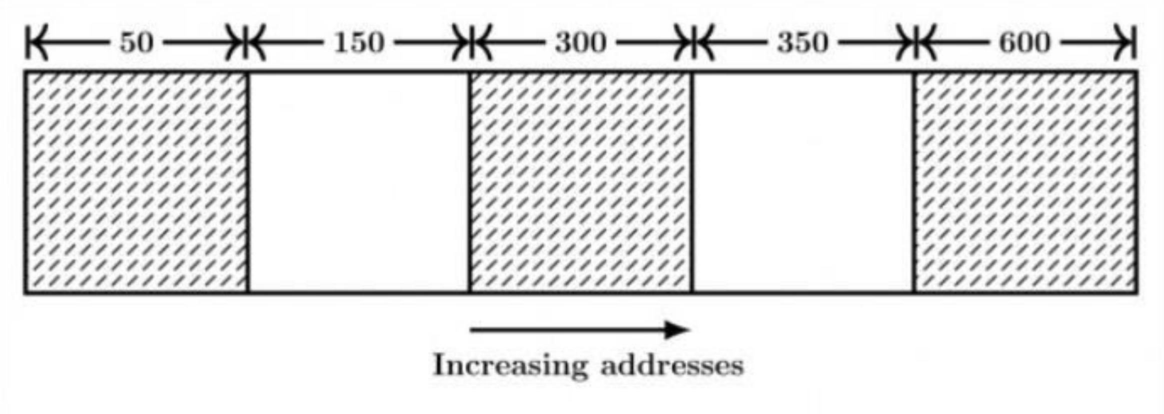
- *It is simple and easy to implement.*
- *It supports multiprogramming since multiple processes can be stored inside the main memory.*
- *Only one memory access is required which reduces the access time.*

# Disadvantages of Static Partitioning

- *It suffers from both **internal fragmentation** and **external fragmentation**.*
- *It utilizes memory inefficiently.*
- *The degree of multiprogramming is **limited** equal to number of partitions.*
- *There is a **limitation on the size of process** since processes with size greater than the size of largest partition can't be stored and executed.*

# Dynamic Storage - Allocation Problem

Consider the following heap (figure) in which blank regions are not in use and hatched region are in use.




The sequence of requests for blocks of sizes 300, 25, 125, 50 can be satisfied if we use

A. either first fit or best fit policy (any one)

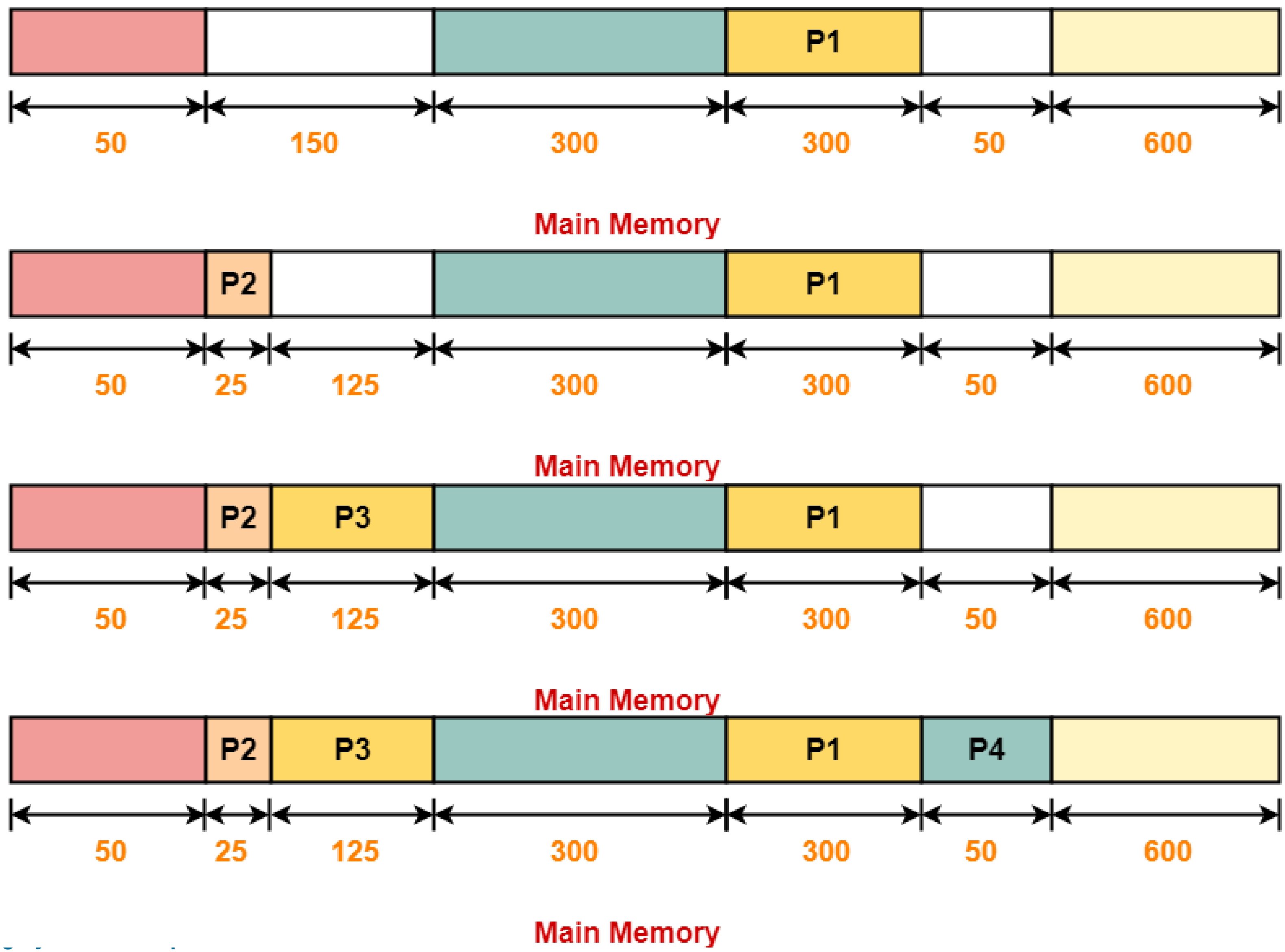
B. first fit but not best fit policy

C. best fit but not first fit policy

D. None of the above

# Dynamic Storage-Allocation Problem

**Allocation Using First Fit Algorithm**

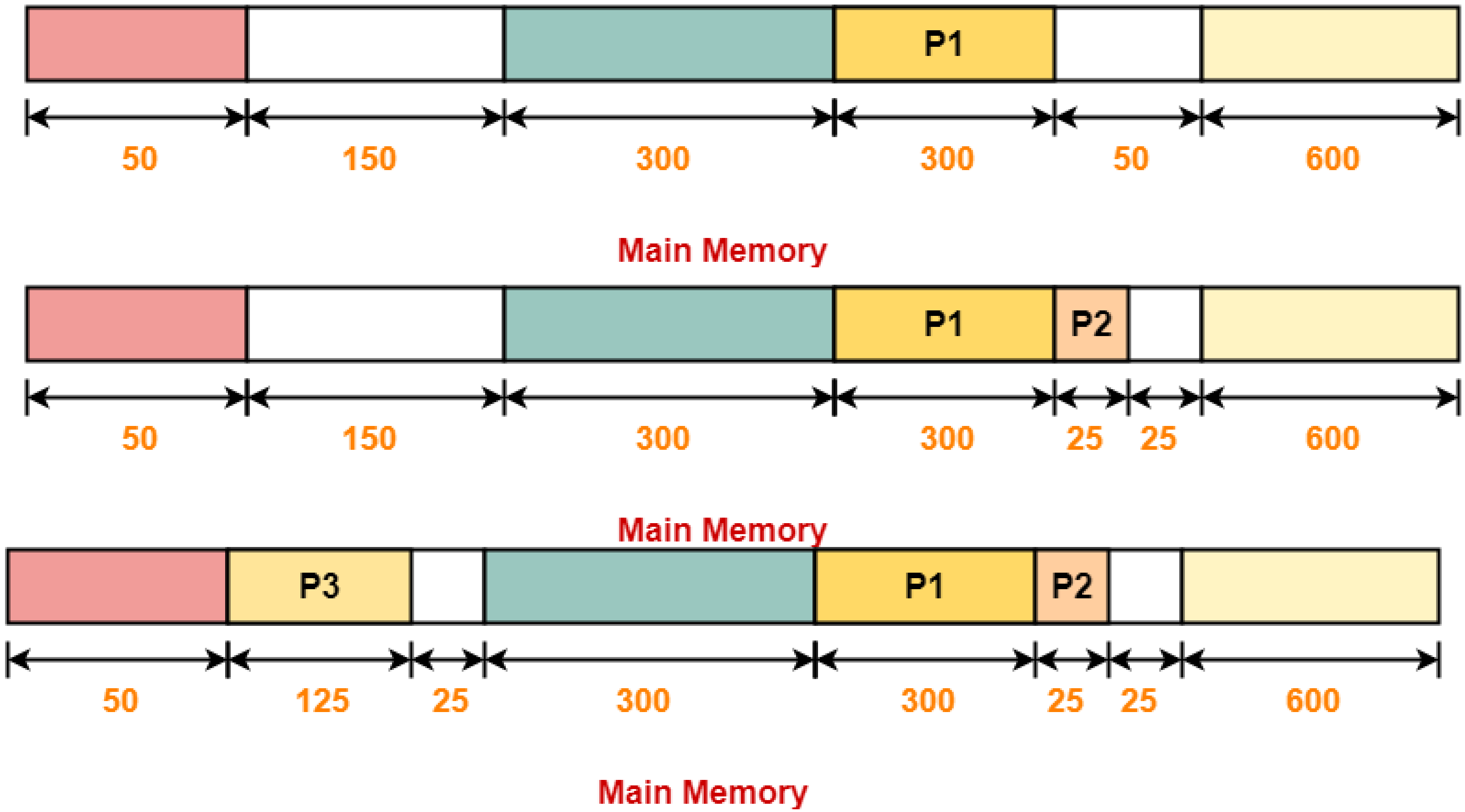


# *Important Points about Dynamic Size Partitioning*

- *Worst Fit Algorithm works best, This is because space left after allocation inside the partition is of large size, There is a high probability that this space might suit the requirement of arriving processes.*
- *Best Fit Algorithm works worst. This is because space left after allocation inside the partition is of very small size and there is a low probability that this space might suit the requirement of arriving processes.*

# Dynamic Storage-Allocation Problem

***Allocation Using Best Fit Algorithm-***



# *Advantages of Dynamic Size Partitioning*

- *It does not suffer from internal fragmentation.*
- *Degree of multiprogramming is dynamic.*
- *There is no limitation on the size of processes.*

# Disadvantages of Dynamic Size Partitioning

- *It suffers from external fragmentation.*
- *Allocation and deallocation of memory is complex.*

# Problem-4

In a computer system where the 'best fit' algorithm is used for allocating 'jobs' to 'memory partition', the following situation was encountered.

| Partition size in KB | 4 KB | | 8 KB | | 20 KB | | 2 KB |
|---|---|---|---|---|---|---|---|
| Jobs | 2 K | 14 K | 3 K | 6 K | 10 K | 20 K | 2 K |
| Times for execution | 4 | 10 | 2 | 1 | 1 | 8 | 6 |

When will the 20 K job complete?

# Structure of Page Table

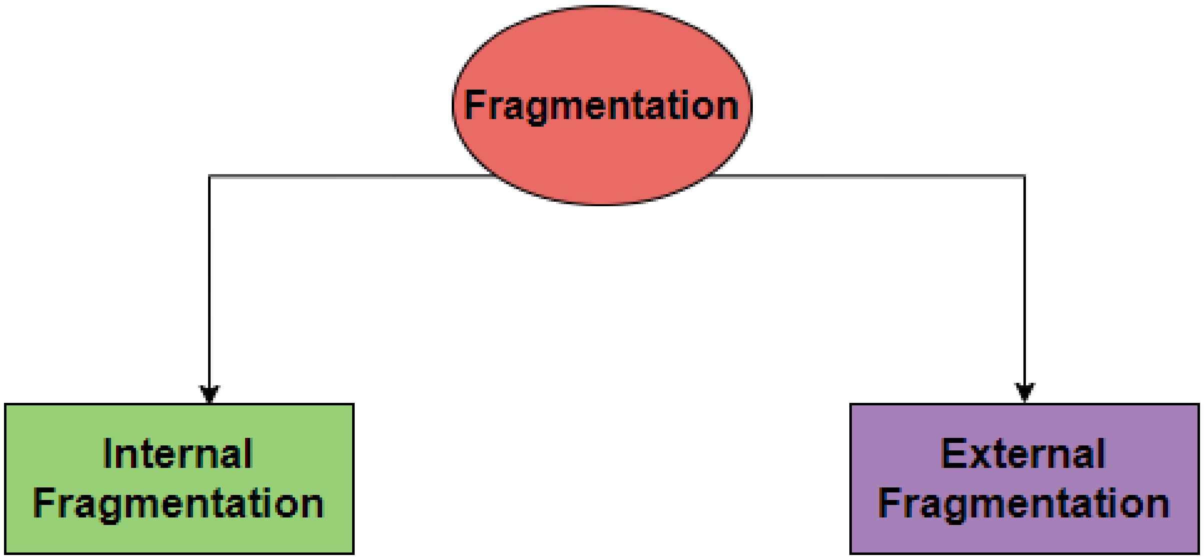
| Jobs | Job size | Ex. Time |
|---|---|---|
| 1 | 2k | 4 |
| 2 | 14k | 10 |
| 3 | 3k | 2 |
| 4 | 6k | 1 |
| 5 | 6k | 4 |
| 6 | 10k | 1 |
| 7 | 20k | 8 |
| 8 | 2k | 6 |

# Structure of Page Table

| Jobs | Job size | Ex. Time |
|---|---|---|
| 1 | 2k | 4 |
| 2 | 14k | 10 |
| 3 | 3k | 2 |
| 4 | 6k | 1 |
| 5 | 6k | 4 |
| 6 | 10k | 1 |
| 7 | 20k | 8 |
| 8 | 2k | 6 |

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | | | | | | | | | | | |
| 4k | 3 | 3 | 8 | 8 | 8 | 8 | 8 | 8 | | | | | | | | | | | |
| 8k | 4 | 5 | 5 | 5 | 5 | | | | | | | | | | | | | | |
| 20k | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 6 | 7 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 2k | 1 | 1 | 1 | 1 | | | | | | | | | | | | | | | |

# Fragmentation



# Internal Fragmentation

- *It occurs when the space is left inside the partition after allocating the partition to a process.*
- *This space is called as internally fragmented space.*
- *This space can not be allocated to any other process. This is because only static partitioning allows to store only one process in each partition.*
- *Internal Fragmentation occurs only in static partitioning.*

# External Fragmentation

- *It occurs when the total amount of empty space required to store the process is available in the main memory.*
- *But because the space is not contiguous, so the process can not be stored.*

# Fragmentation

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

.

# *Solution for External Fragmentation*

- *Reduce external fragmentation by compaction*
  - *Shuffle memory contents to place all free memory together in one large block.*

- *Permit the logical address space to be noncontiguous, allow process to allocate physical memory wherever it is available: Paging and segmentation.*

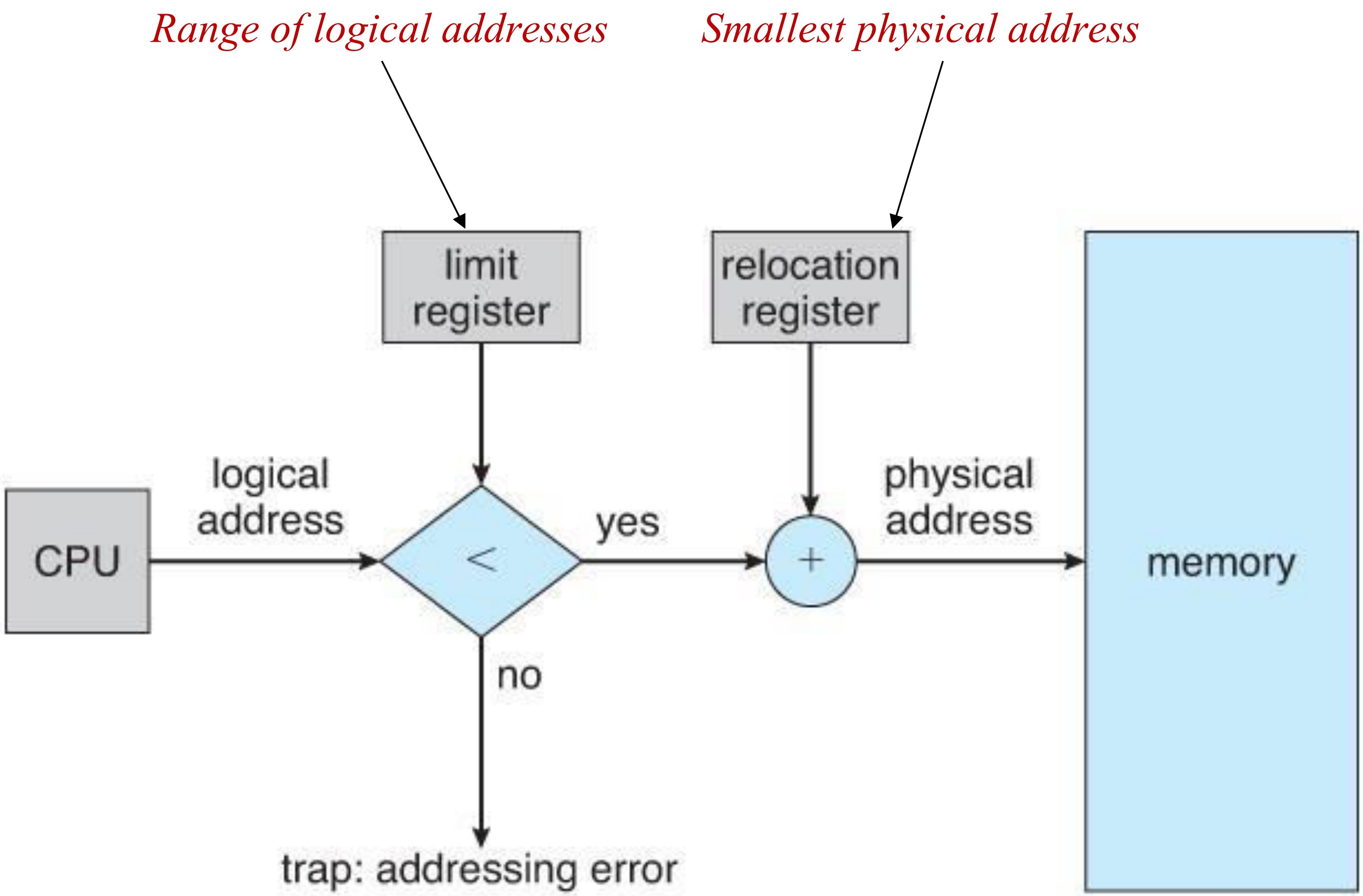
# *Translating Logical to Physical Address*

- *CPU always **generates** a logical address.*
- *A **physical address is needed to access the main memory**.*
- *During **context switching**, the values corresponding to the process being loaded are set in the registers **(Relocation Register, Limit Register).***
- *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 scheme used to protect user processes from each other, and from changing operating-system code and data.*

## *Translating Logical to Physical Address*

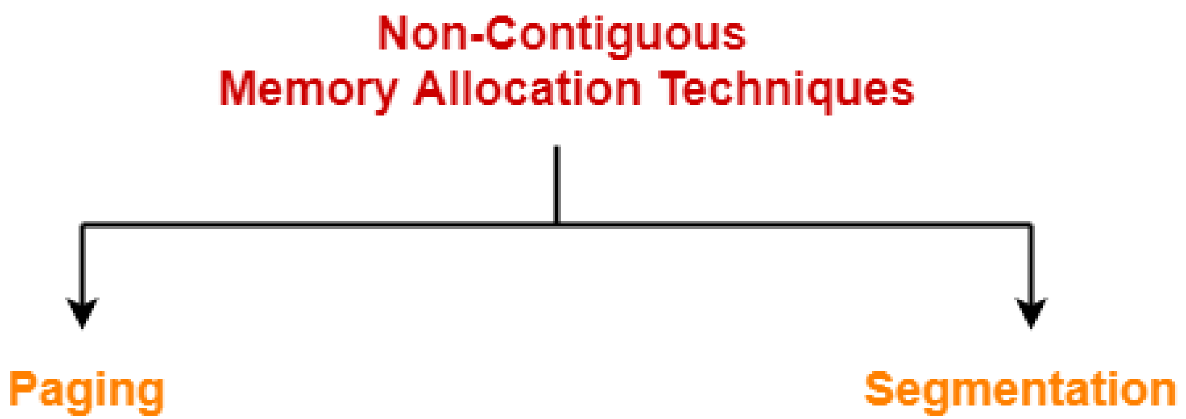
- *CPU generates a* ***logical address containing*** *the address of the instruction that it wants to read.*
- *The address must always lie in the range [0, limit-1]. If address is found to be smaller than the limit, then the request is treated as a valid request. Then, generated address is added with the base address of the process.*
- *The result obtained after addition is the address of the memory location storing the required word.*

# Contiguous Allocation: Memory Protection



# *Non-Contiguous Memory Allocation*

- *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**.*



# *Segmentation*

- *Divide process in logically related partition.*
- *The process is divided into **modules** for better visualization.*
- *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**.*

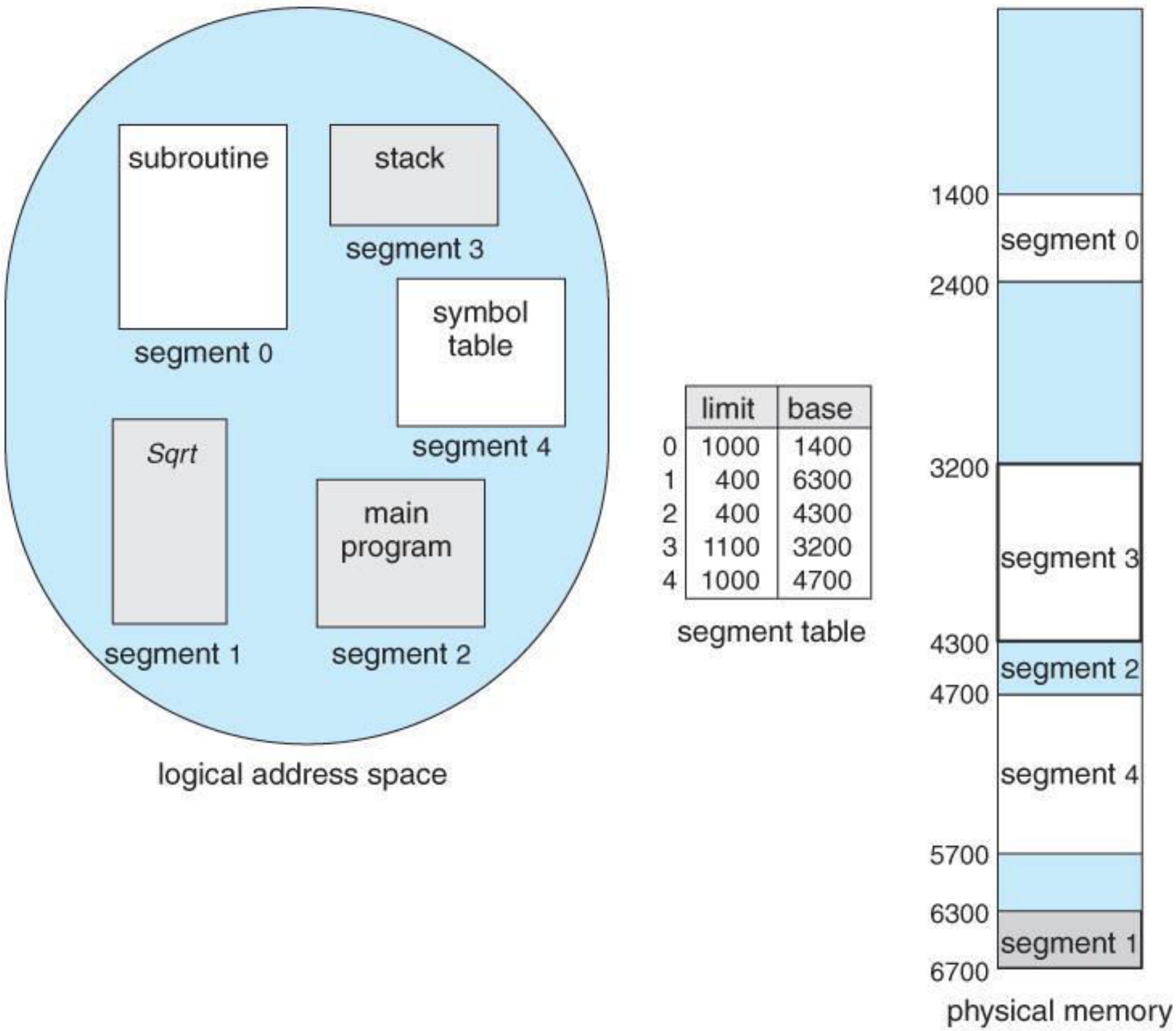
# Segmentation

- *Thus, a logical address consists of a two tuple:*

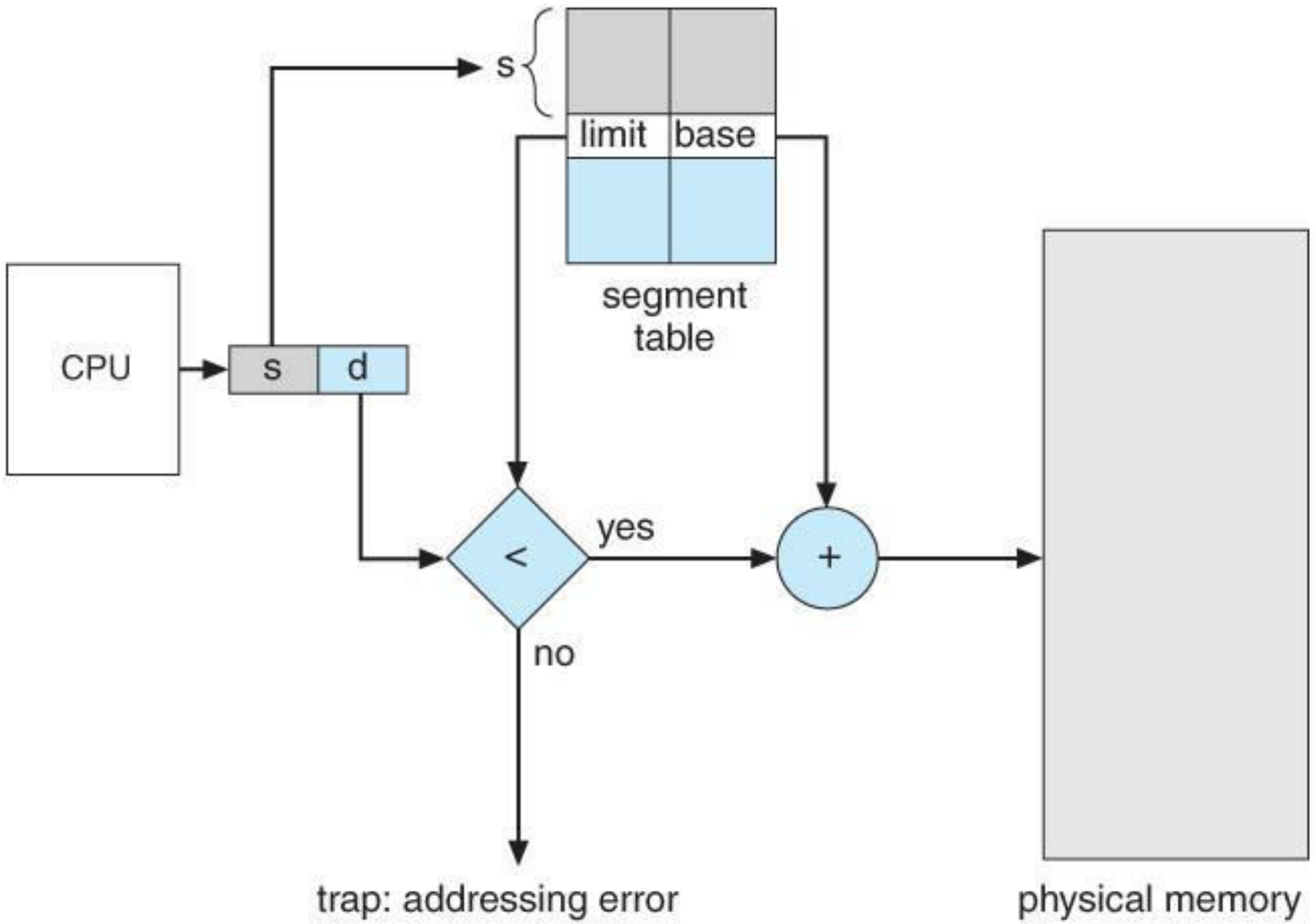
  ***<segment-number, offset>.***

- *This mapping is effected by a segment table. Each entry in the segment table has a* ***segment base and a segment limit****.*
- ***Segment base****: Contains the starting physical address where the segment resides in memory.*
- ***Segment limit:*** *specifies the length of the segment.*
- *Segment table is kept in memory: Segment table base register, segment table length register.*

# *Segmentation hardware*



# Segmentation Hardware



## *Segmentation Advantages*

- *It allows to divide the program into modules which* ***provides better visualization****.*
- *It solves the problem of* ***internal fragmentation****.*

# *Segmentation Disadvantages*

- *Segments of unequal size are not suited for swapping.*
- *It **suffers from external fragmentation** as the free space gets broken down into smaller pieces with the processes being loaded and removed from the main memory.*

# Segmentation

***Consider the segment table given below:***

| *Segment No.* | *Base* | *Limit* |
|---|---|---|
| *0* | *1219* | *700* |
| *1* | *2300* | *14* |
| *2* | *90* | *100* |
| *3* | *1327* | *580* |
| *4* | *1952* | *96* |

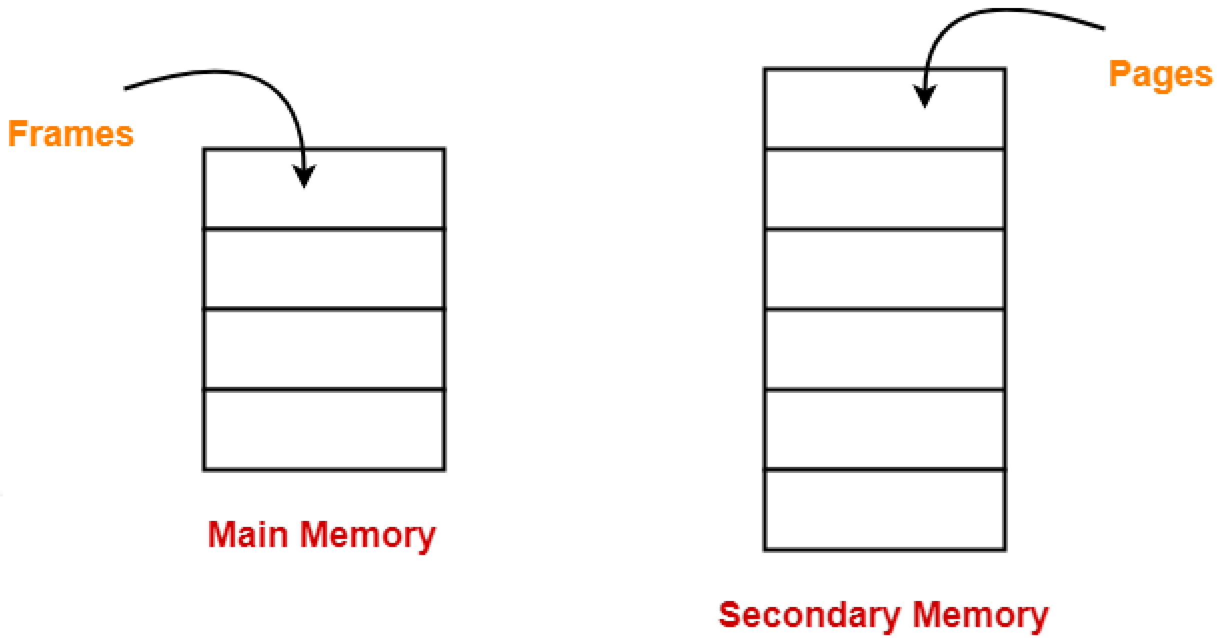
*Which of the following logical address will produce trap addressing error?*

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

***Calculate the physical address if no trap is produced.***

# Paging

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

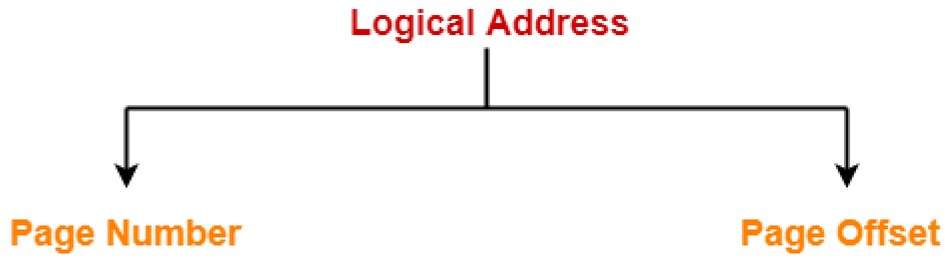


# Paging

- *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****.*

# *Translating Logical Address into Physical Address*

- *CPU always generates a* ***logical address****.*



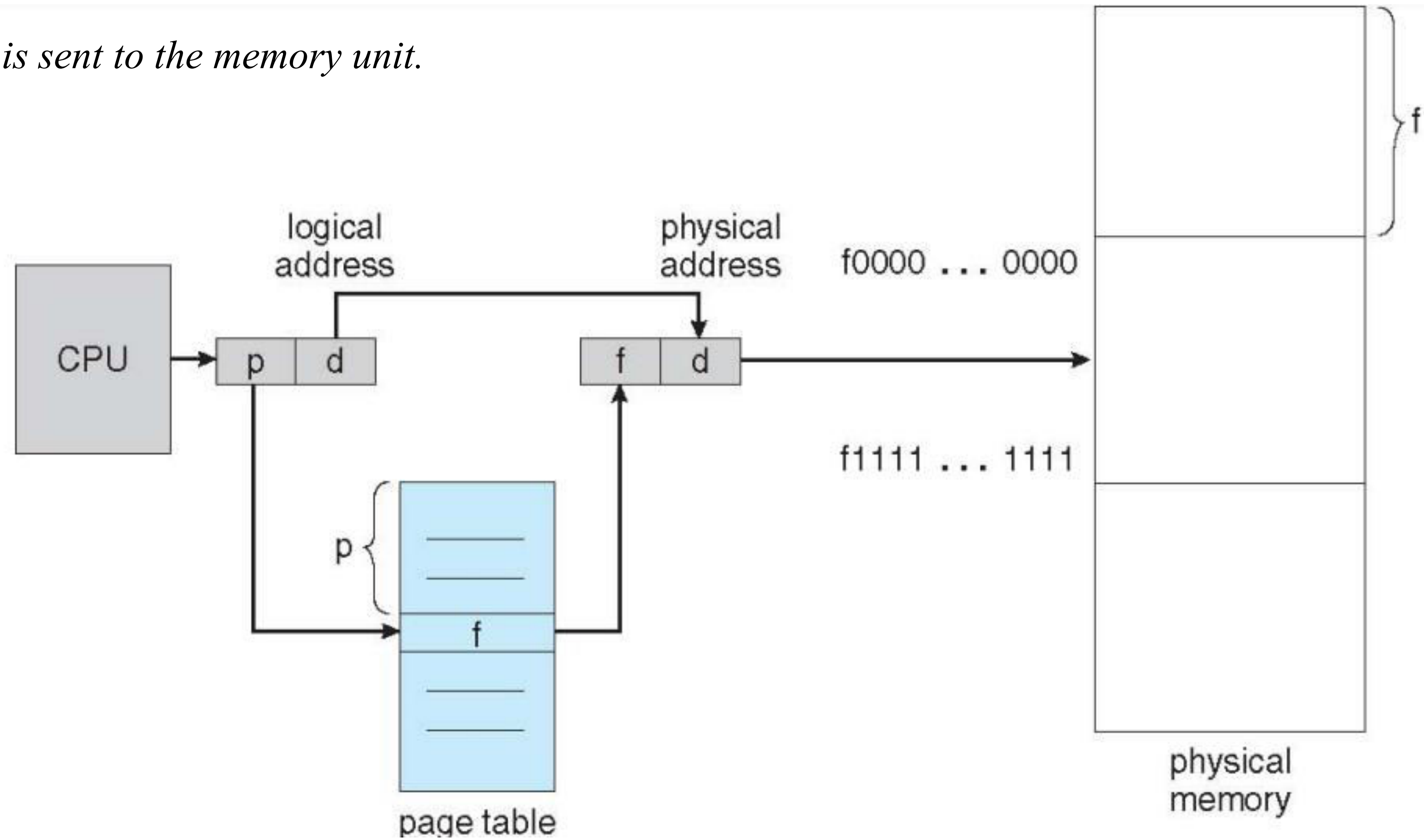

- ***Page Number*** *specifies the specific page of the process from which CPU wants to read the data.*
- ***Page Offset*** *specifies the specific instruction on the page that CPU wants to read.*

## *Translating Logical Address into Physical Address*

- *A **physical address** is **needed** to **access the main memory***
- *For the page number generated by the CPU, **Page Table** provides the corresponding frame number (base address of the frame) where that page is stored in the main memory.*

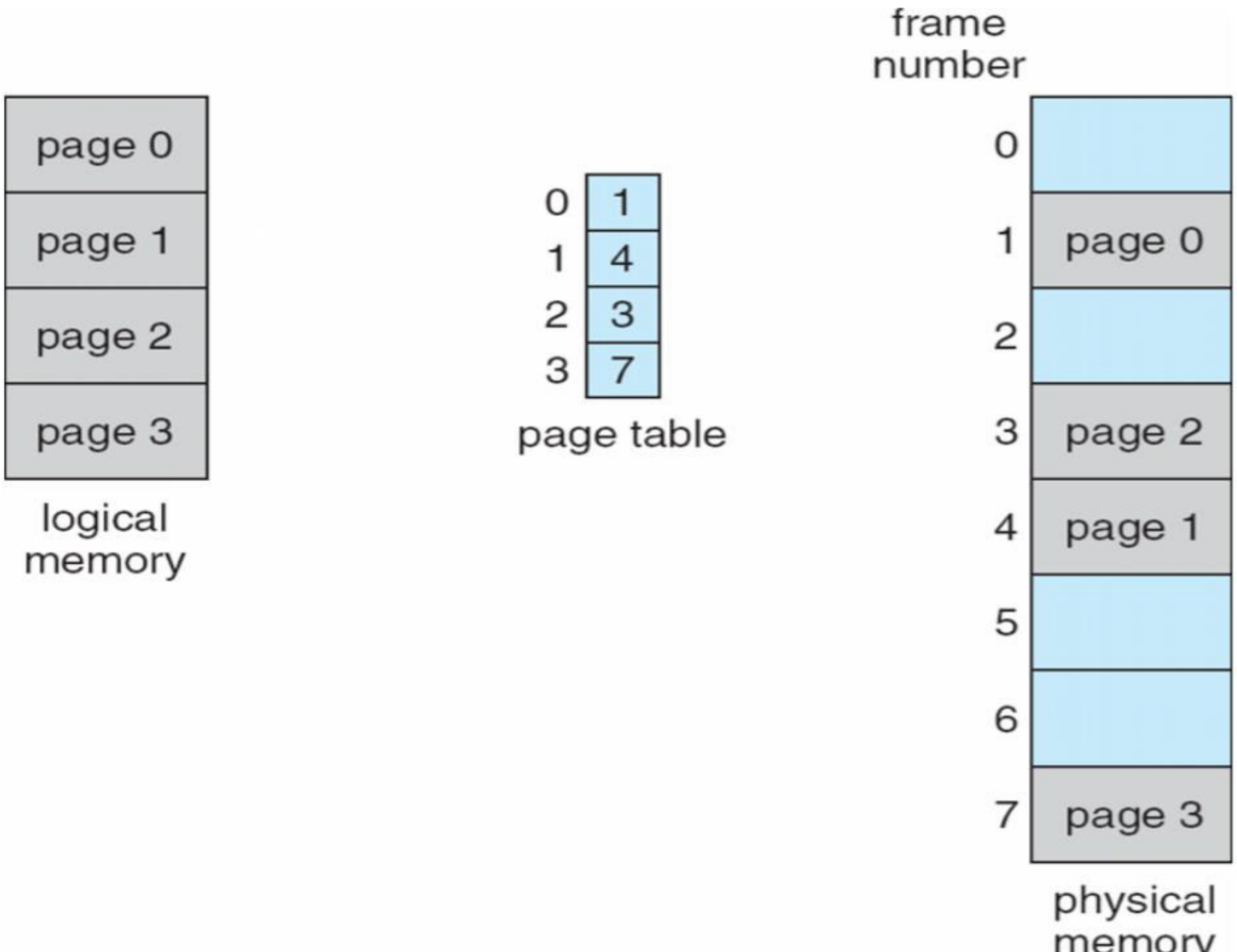
# Paging Hardware (MMU)

- ***Page number (p)*** *– Used as an index into a page table.*
- ***Page offset (d) –*** *Combined with base address to define the physical memory address that is sent to the memory unit.*



# *Paging model of logical and physical Memory*

- *Page 0 is in Frame 1, Page 1 is placed in frame 4.*
- *Page 2 is placed in frame 3, Page 3 is placed in Frame 7.*



# *Fragmentation by paging*

- *When we use a paging scheme, we have* ***no external fragmentation****: However, we* ***may have some internal fragmentation****.*
- *In the* ***worst case****, a process would need* ***n pages plus 1 byte****. It would be allocated n + 1 frames, resulting in internal fragmentation of almost an entire frame.*

# Question

***If Page size = 2,048 bytes and Process size = 72,766 bytes then find out internal fragmentation***

# *Paging*

- *In this scheme every data/instruction access requires* ***two memory accesses****. One for the page table and one for the data/instruction. Thus, memory access is slowed by a* ***factor of 2****.*
- *The two memory access problem can be solved by the use of a special, small, fast (high-speed) lookup hardware called* ***translation look-aside buffers (TLBs).***

# Paging

- *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.*

# Paging

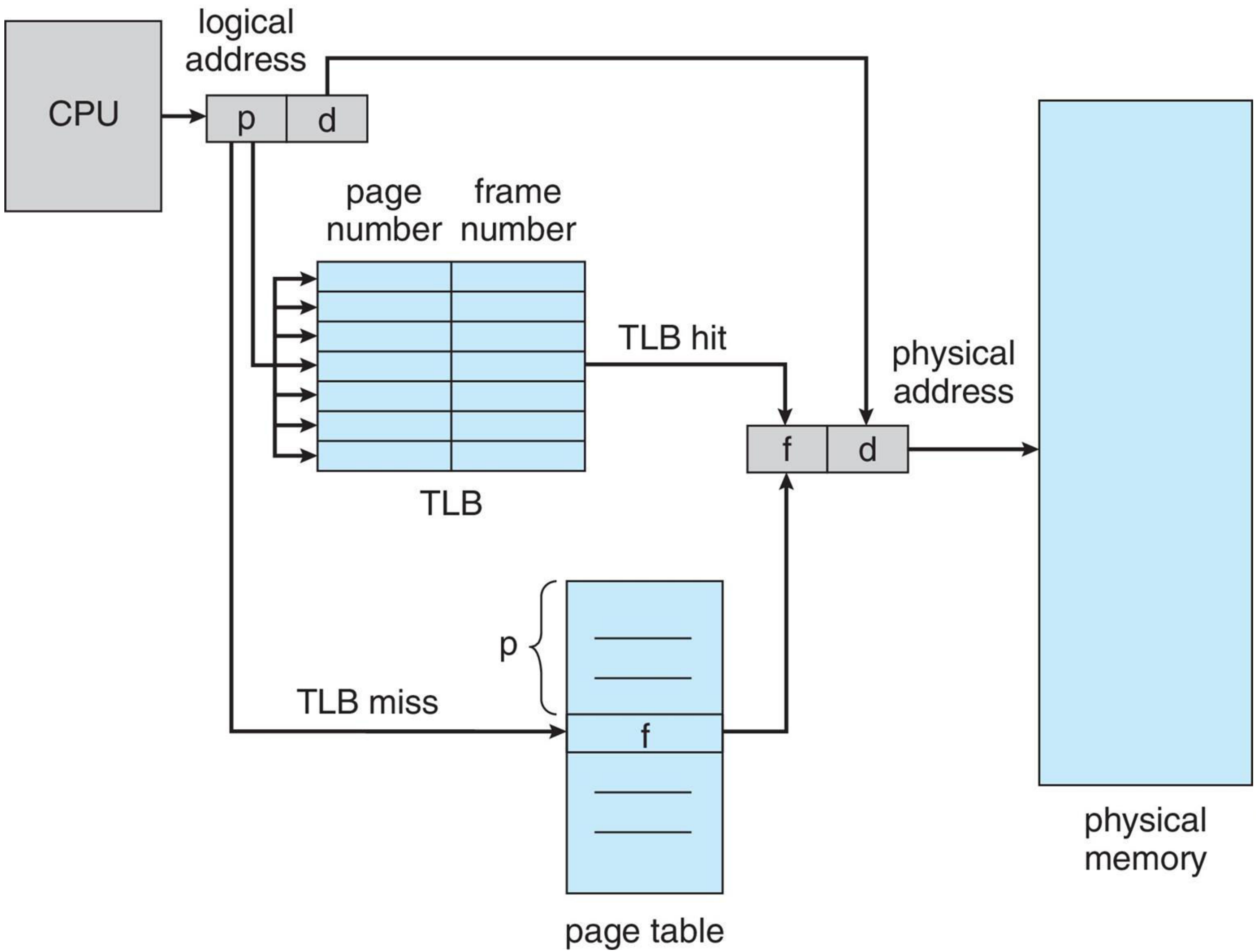
- *TLBs allow certain entries to be* ***wired down****, meaning that they cannot be removed from the TLB. Typically, TLB entries for key kernel code are wired down.*
- *TLBs store* ***address-space identifiers (ASIDs)*** *in each TLB entry. An ASID uniquely identifies each process and is used to provide address-space protection for that process.*
- *ASID allows the TLB to contain entries for several different processes simultaneously*

# *Associative Memory*

- *Translation Lookaside Buffer (TLB) consists of two columns-*
    1. *Page Number*
    2. *Frame Number*
- *Associative memory – Parallel search*

| Page # | Frame # |
|---|---|
| | |
| | |
| | |
| | |

- *TLB is checked to see if it contains an entry for the referenced page number.*
- *The referenced page number is compared with the TLB entries all at once.*



CPU
logical address
p
d
page number
frame number
TLB hit
physical address
f
d
TLB
p
TLB miss
f
physical memory
page table

# Paging

*When a new process gets scheduled-*

- *Initially, TLB is empty. So, TLB misses are frequent.*
- *With every access from the page table, TLB is updated.*
- *After some time, TLB hits increases and TLB misses reduces.*

# Paging

- ***Hit Ratio:*** *Percentage of times that the page number of interest is found in the TLB is called the **hit ratio**.*
- ***Miss Ratio:*** *Percentage of times that the page number of interest is not found in the TLB is called the **miss ratio (100-hit ratio)**.*

**Effective Access Time =**

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

**+**

**Miss ratio of TLB x { Access time of TLB + (L+1) x Access time of main memory }**

**where L = Number of levels of page table**

# Paging

***Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 10 ns to search the TLB and 200 ns to access the physical memory. If TLB hit ratio is 70%, the effective memory access time is _______ nsec.***

*Substituting values in the above formula, we get-*

*Effective Access Time*

*= **0.7** * { 10 ns + 200 ns } + **0.3** * { 10 ns + (1+1) * 200 ns }*

*= 0.7 * 210 ns + 0.3 + 410 ns*

*= 147 ns + 123 ns*

*= 270 ns*

*Thus, effective memory access time = **270 ns**.*

# Paging

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

1. 54
2. 60
3. 20
4. 75

160 ns = 0.6 x { T + 100 ns } + 0.4 x { T + 2 x 100 ns }

160 = 0.6 x T + 60 + 0.4 x T + 80

160 = T + 140

T = 160 – 140

T = 20

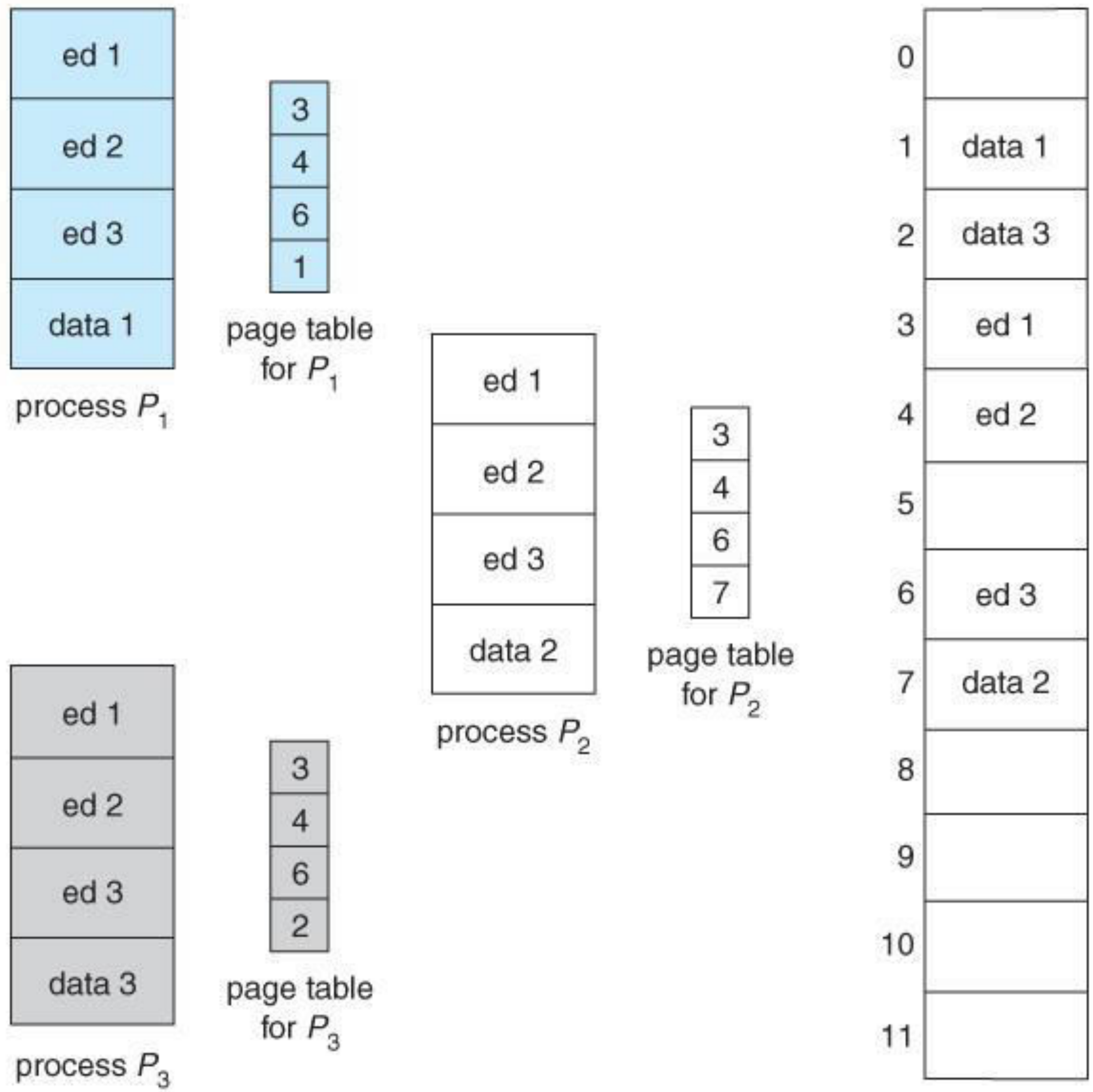
# *Shared Pages*

- *An advantage of paging is possibility of* ***sharing common codes***
- ***Reentrant code (read only)*** *is non-self-modifying code: it never changes during execution. Thus, two or more processes can execute the same code at the same time.*
- .

# Shared Pages

- *Consider a system that supports 40 users, each of whom executes a text editor. The text editor consists of 150 KB of code and 50 KB of data space.*
- *Then we need 8,000 KB to support the 40 users.*
- *If the code is* ***reentrant code*** *(or* ***pure code****), it can be shared.*

# Shared Pages



# Shared Pages

*Thus, to support 40 users, we need only one copy of the editor (150 KB), plus 40 copies of the 50 KB of data space per user. The total space required is now 2,150 KB instead of 8,000 KB—a significant savings.*

# Problem-1

***Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 20 ns to search the TLB. If TLB hit ratio is 50% and effective memory access time is 170 ns, main memory access time is ______.***

# *Problem-2*

Consider a paging hardware with a TLB. Assume that the entire page table and all the pages are in the physical memory. It takes 10 milliseconds to search the TLB and 80 milliseconds to access the physical memory. If the TLB hit ratio is 0.6, the effective memory access time (in milliseconds) is __________.

**(A)** 120

**(B)** 122

**(C)** 124

**(D)** 118

# Problem-3

***Consider a single level paging scheme with a TLB. Assume no page fault occurs. It takes 10 ns to search the TLB and 200 ns to access the physical memory. If effective memory access time is 200 ns, TLB hit ratio is ______.***

# Important Formulas for Paging

## For Main Memory-

- *Physical Address Space = Size of main memory*
- *Size of main memory = Total number of frames * Frame 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*

# Important Formulas for Paging

**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*

**For Page Table-**

- *Size of page table = Number of entries in page table * Page table entry size*
- *Number of entries in pages table = Number of pages the process is divided*
- *Page table entry size = Number of bits in frame number + Number of bits used for optional fields if any*

# Problem-1

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

*a)2*

*b)4*

*c)8*

*d)1*

*Number of bits in logical address = 32 bits, Thus, Process size=* $2^{32}$ *B*

*Number of pages the process is divided = Process size / Page size =* $2^{32} / 2^{12}$

$= 2^{20}$ *pages*

*Thus, Number of entries in page table =* $2^{20}$ *entries*

*Page table size = Number of entries in page table x Page table entry size*

$= 2^{20} \times 4$ *bytes*

*= 4 MB*

# *Important Formulas for Paging*

- *In general, if the given address consists of 'n' bits, then using 'n' bits, $2^n$ locations are possible.*
- *Then, size of memory = $2^n$ * Size of one location.*
- *If the memory is byte-addressable, then size of one location = 1 byte.*
- *Thus, size of memory = $2^n$ bytes.*
- *If the memory is word-addressable where 1 word = m bytes, then size of one location = m bytes.*
- *Thus, size of memory = $2^n$ * m bytes.*

# Problem-2

*Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?*

*a)16 MB*

*b)8 MB*

*c)2 MB*

*d)24 MB*

*Total physical memory = 2^26 byte*

*Process Size = 2^32 byte*

*Page size = 2^12 byte*

*Number of frames = 2^14*

*Bits in frame address = 14*

*No of Pages = 2^20*

*Page Table Size = 2^20 * 14 bits (approximating 14bits to 16 bits)*

*= 2 MB*

# Problem-3

*Consider a single level paging scheme. The virtual address space is 4 MB and page size is 4 KB. What is the maximum page table entry size possible such that the entire page table fits well in one page?*

*Process Size = 2^22 byte*

*Page size = 2^12 byte*

*No of pages = 2^10*

*Page table size = No of page table entries * page table entry size*

*Let's assume Page table entry size = x*

*2^10*x = 2^12*

*X = 2^2*

*4 Byte*

# Problem-4

*In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is the maximum number of bits that can be used for storing protection and other information in each page table entry*

*a)2*

*b)10*

*c)12*

*d)14*

*Total physical memory = 2^30 byte*

*Process Size = 2^32 byte*

*Page size = 2^12 byte*

*Page table entry = 32 bit= 4 byte*

*No. of Frames = 2^30/2^12 = 2^18*

*Frame address 18 bits*

*X=32-18 = 14*