Marwadi University

3140702
Operating System

Unit – 6
Memory Management

Topics to be covered

- Basics of Memory Management
- Memory partitioning:
  - Fixed Size Partitioning
  - Variable Size Partitioning
- Memory Allocation Strategies:
  - First Fit
  - Best Fit
  - Worst Fit
- Swapping and Fragmentation
- Paging and Demand Paging
- Segmentation
- Concepts of Virtual Memory

Topics to be covered

- Page Replacement Policies
  - FIFO
  - LRU
  - Optimal
- Thrashing

Memory Management: Definition

- Second major component of computer system is memory after processors.
- CPU can only access directly main memory and register.
- Thus if we want to process some data then that data must be fetched into main memory from disk.

Memory Management: Definition

As the main memory have limited space (generally less than the hard disk), it may causes some issues related to memory allocation and deallocation.

Is the task carried out by the OS and hardware to accommodate multiple processes in main memory

Memory Management: Function

1. Keep track of what parts of memory are in use.
2. Allocate memory to processes when needed
3. Deallocate when processes are done.
4. Swapping, or paging, between main memory and disk, when main memory is too small to hold all current processes.

Logical and Physical Address Map

Logical address/ virtual address

- address generated by the CPU
- Range of logical address are limited to the size of processor.
- Example: 32-bit processor then address range would be up to 2^32 (4GB).

Physical address

- Address seen by Main memory unit.
- Range is depended on the size of memory.

Logical and Physical Address Map

- The basic **difference between Logical and physical address** is that **Logical address** is generated by CPU in perspective of a program.
- On the other hand, the **physical address** is a location that exists in the memory unit.

Logical and Physical Address Map

Logical address space

- The set of all logical addresses generated by a program is known as logical address space.

Physical address space

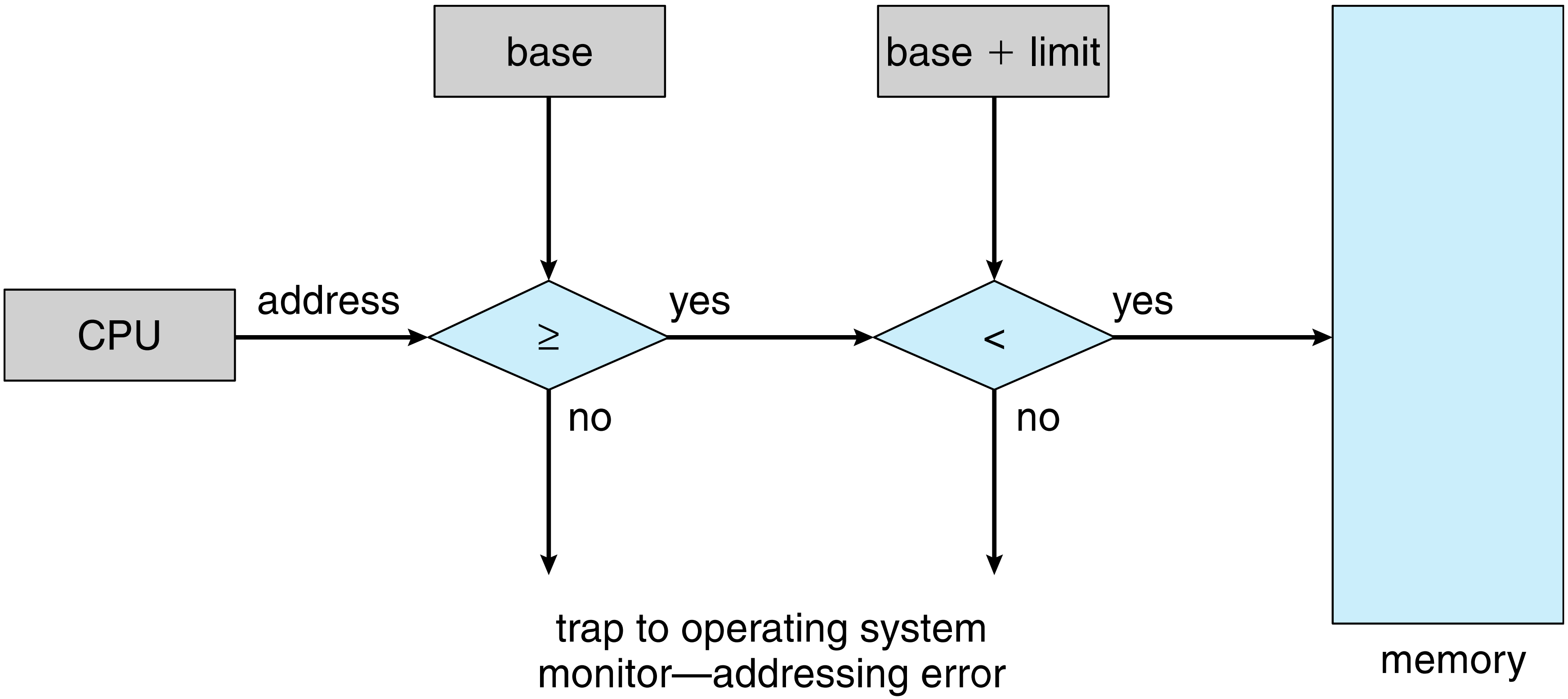
- The set of all physical addresses corresponding to the logical addresses is known as physical address space.

Logical and Physical Address Map

Hardware Address Protection

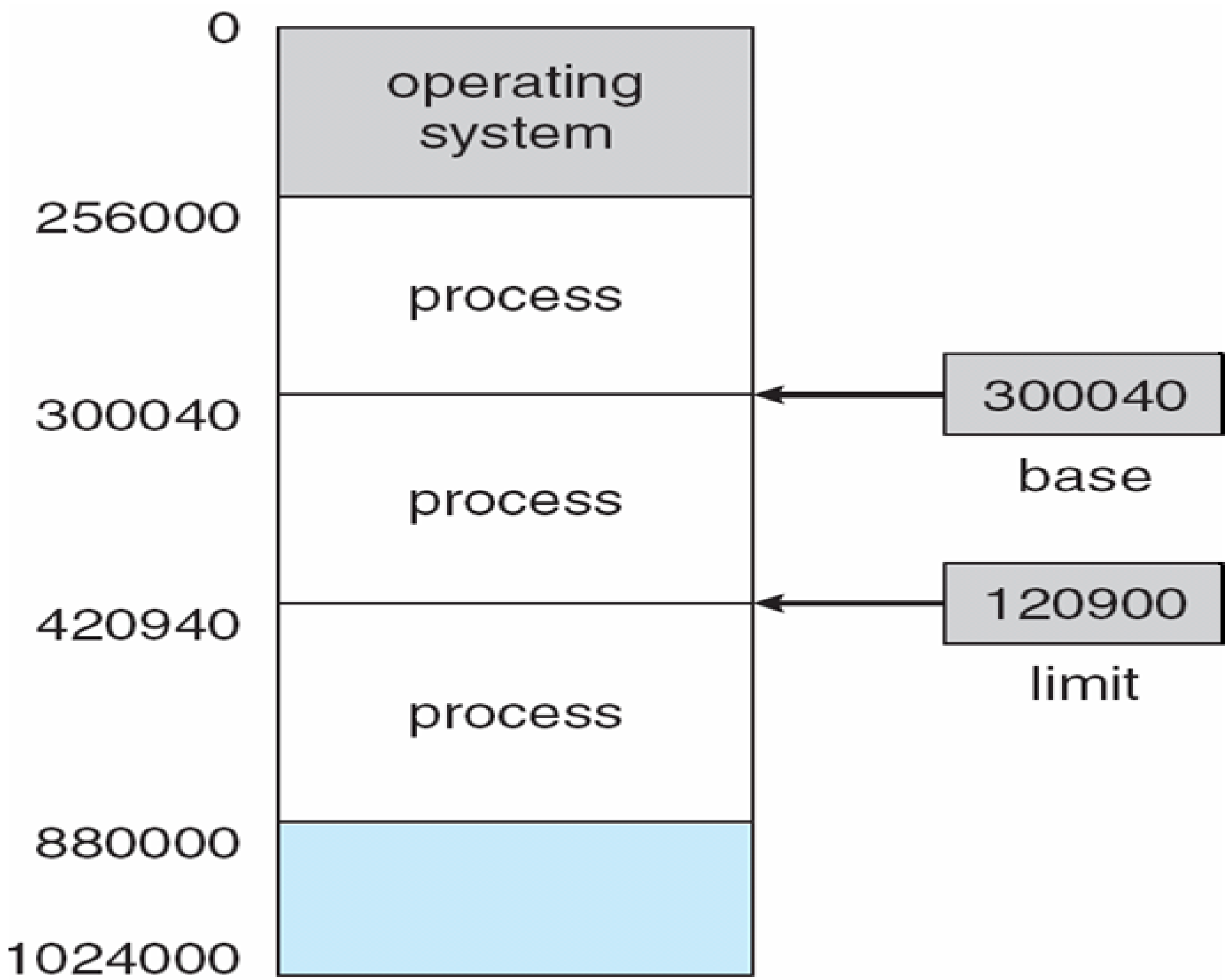
Base register: Starting legal address.

Limit register: size of range.



Logical and Physical Address Map

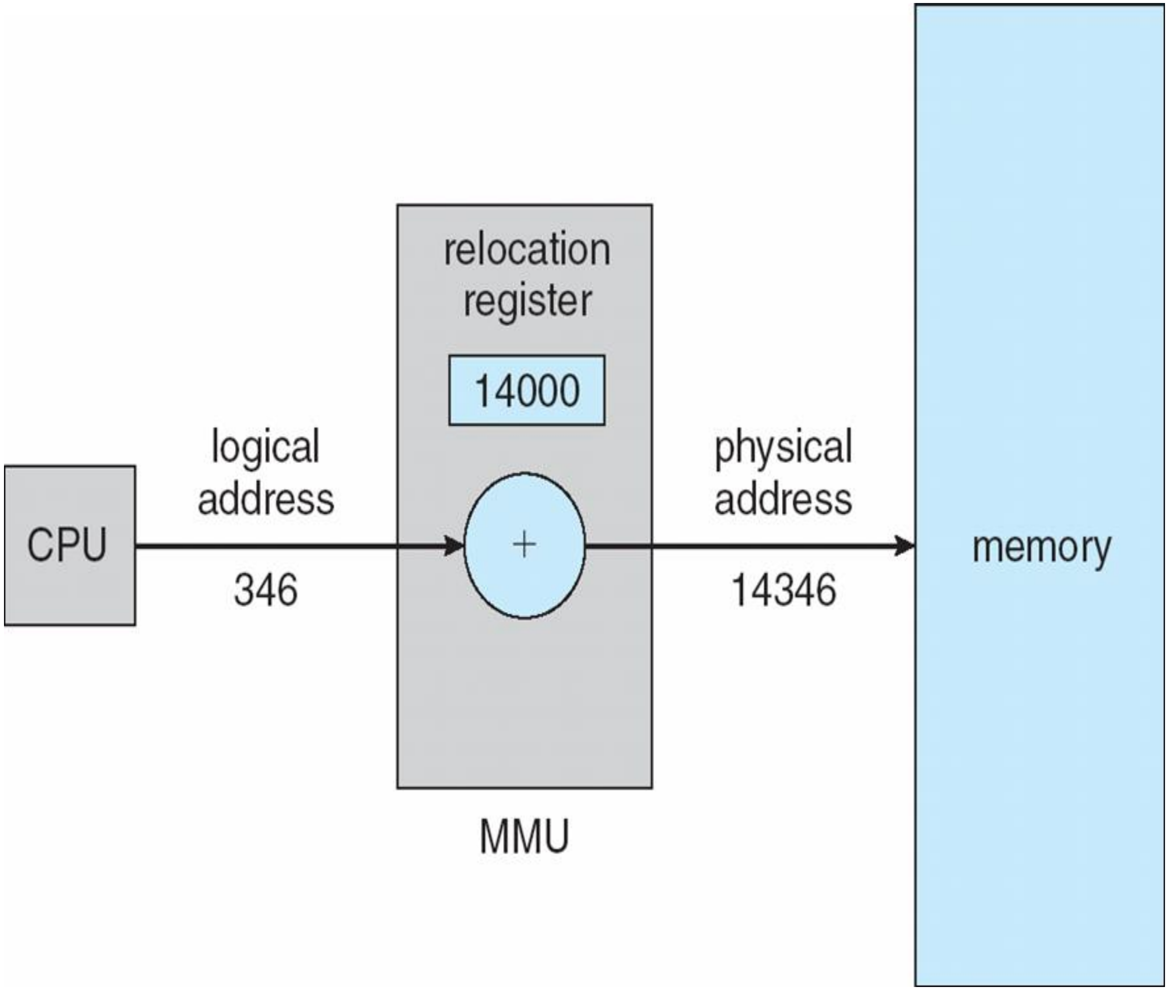
Hardware Address Protection



Logical and Physical Address Map

- For any process to get executed it should be in main memory.
- Thus whenever process (code, data, stack) fetched in main memory, it requires some storage space.
- In main memory physical address of process may not exactly match with the logical address that we need mapping between logical and physical address.
- Part of the OS manages the main memory is know as Memory Management Unit (MMU).

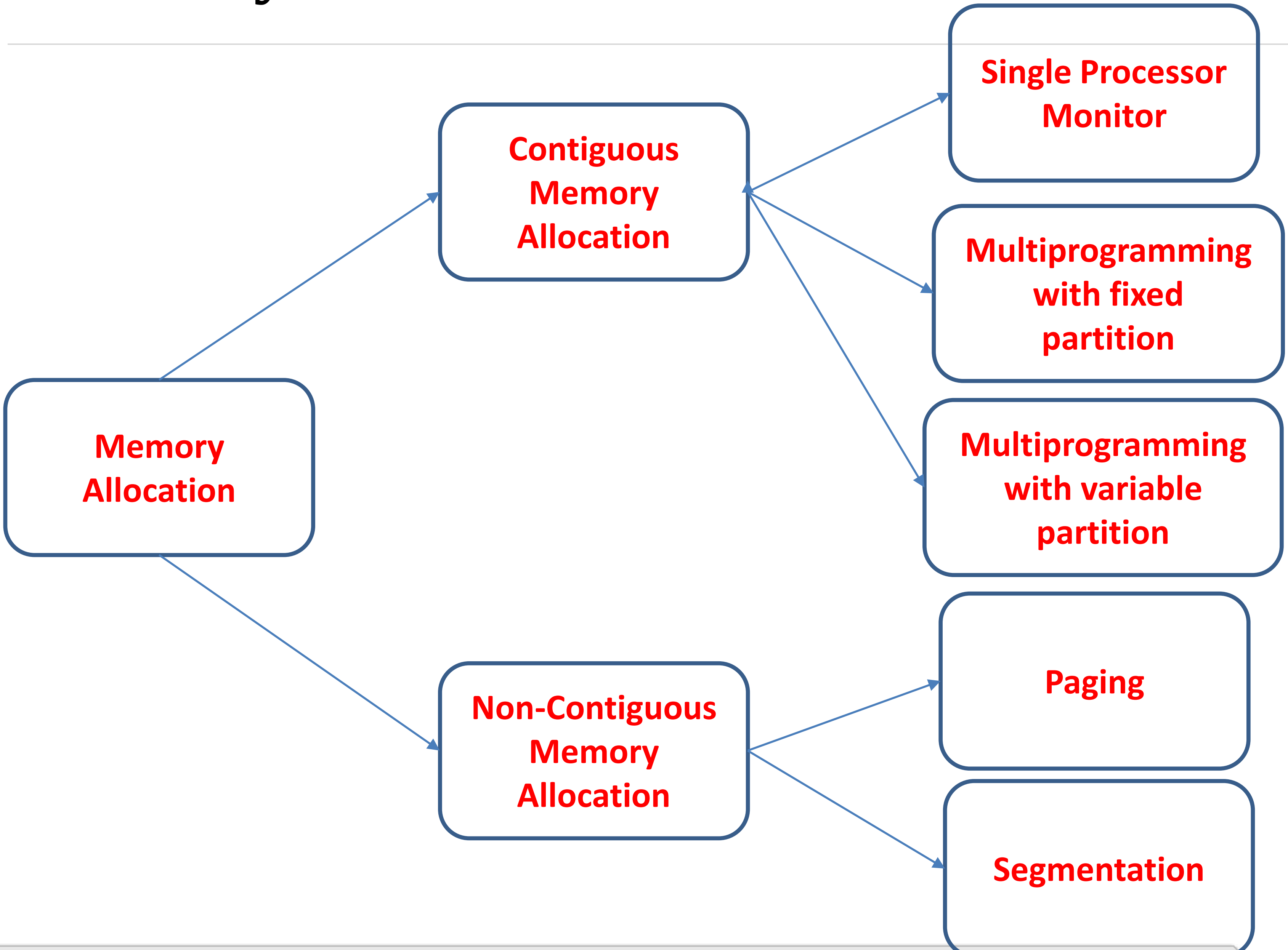
Logical and Physical Address Map



Base register is now known as relocation register.

The value in the relocation register added to every address generated by a user process when process is sent to memory.

Memory Allocation



Memory Allocation Techniques

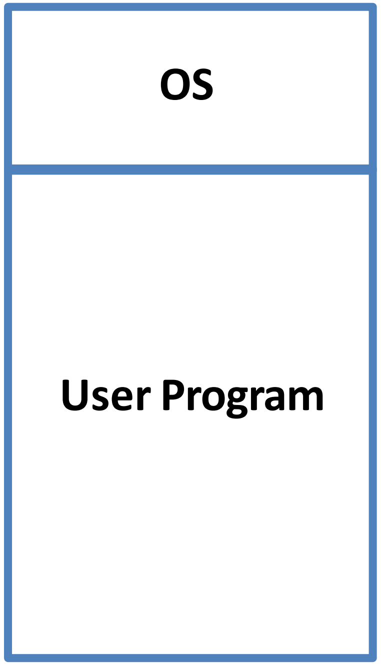
Contiguous Memory allocation

- Simple and old method.
- Here each process occupies contiguous block of main memory.
- When process is brought in memory, a memory is searched to find out a chunk of free memory having enough size to hold a process.

Memory Allocation Techniques

1. Single processor monitor

- Simplest possible schema.
- Only single process is allowed to run.
- Memory is only shared between single process and OS.



Contiguous Memory allocation: Fixed Partition

2. Multi programming with fixed partition

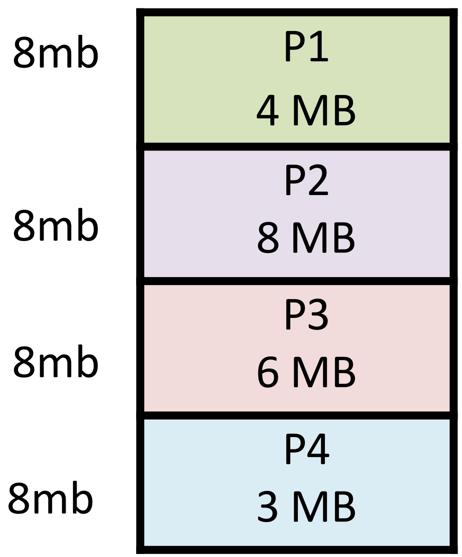
- Numbers of **partitions are fixed.**
- Here, memory is divided **into fixed size partition.**
- **Each partition** may contain **exactly one process**.
- Size of each partition is **not requires to be same.**
- When a partition is free, process is selected from the input queue and it is loaded into free partition.

Contiguous Memory allocation: Fixed Partition

Multi programming with fixed partition:

- **Advantage:**
  - Implementation is simple.
  - Processing overhead is low.
- **Disadvantage:**
  - Limit in **process size**.
  - **Degree of multiprogramming** is also limited.
  - Causes **External fragmentation** because of contiguous memory allocation.
  - Causes **Internal fragmentation** due to fixed partition of memory.

Contiguous Memory allocation: Fixed Partition



- ***Internal fragmentation***:
  because Partition are of fixed size thus any space in a partition which is not used by process is wasted.
- ***Example:***
  - P1 requires 4 MB, P2 requires 8mb, P3 requires 6 MB, P4 requires 3 MB, P5 requires 8MB.
  - Here in total, we have 11 MB free in the memory but due to internal fragmentation we can not assign that to P5.

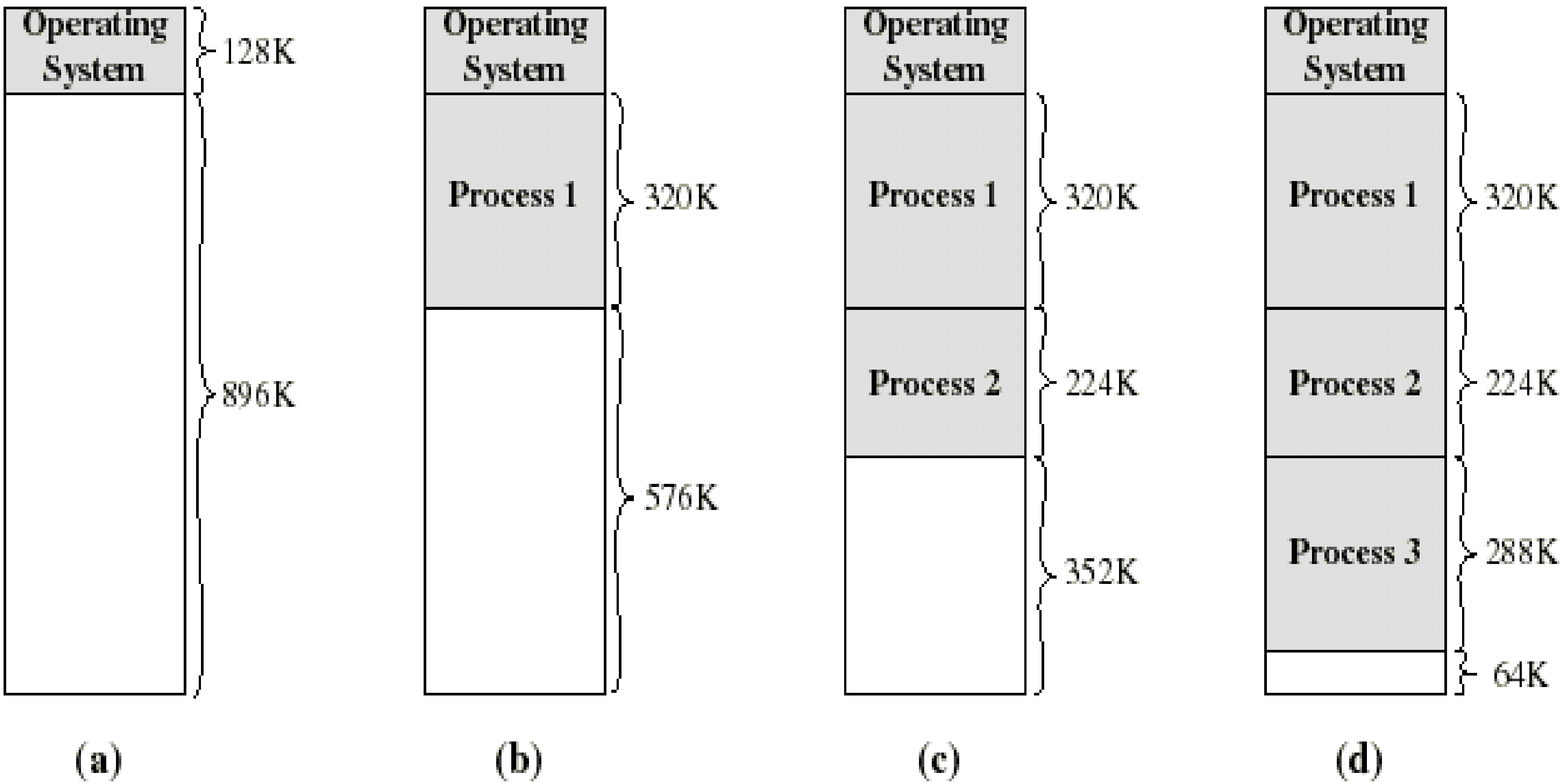
Contiguous allocation: Variable Size Partition

3. Multi programming with variable/dynamic partition

- Here memory **is not divided into fixed partition**, also the number of partition is not fixed.
- Only required memory is allocated to process at runtime.
- Whenever any process enter in a system, a chunk of memory big enough to fit the process is found and allocated. And the remaining unoccupied space is treated as another free partition.
- When process get terminated it releases the space occupied and it that free partition is contiguous to another free partition then that both free partition can be merge.

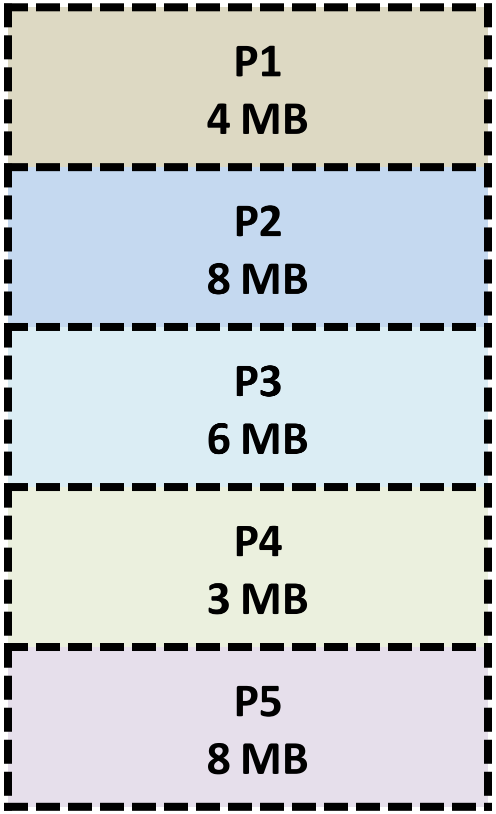
Multi programming with variable/dynamic partition

- Example:



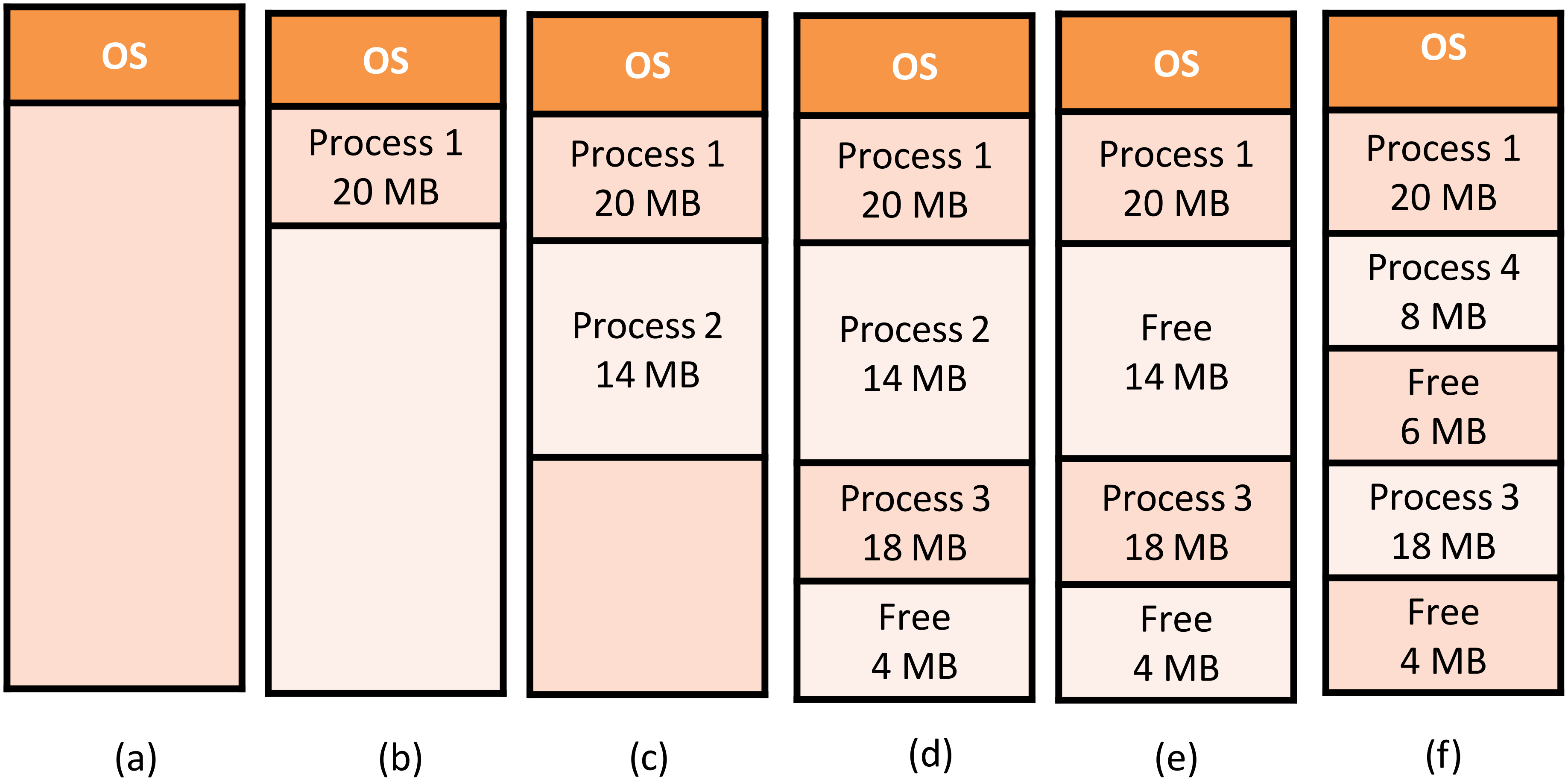
Multi Programming with Variable/Dynamic Partition

- Example:
- P1 needs 4 MB, P2 needs 8 MB, P3 needs 6 MB, P4 needs 3 MB, P5 needs 8 MB.



Multi Programming with Variable/Dynamic Partition

- Disadvantage: Causes External fragmentation:



(a) Initial state (b) Process 1 enters. (c) Process 2 enters. (d) Process 3 enters. (e) Process 2 terminates. (f) Process 4 enters (g) Process 5 needs 10 MB

Contiguous allocation: Variable Size Partition

Multi programming with variable/dynamic partition:

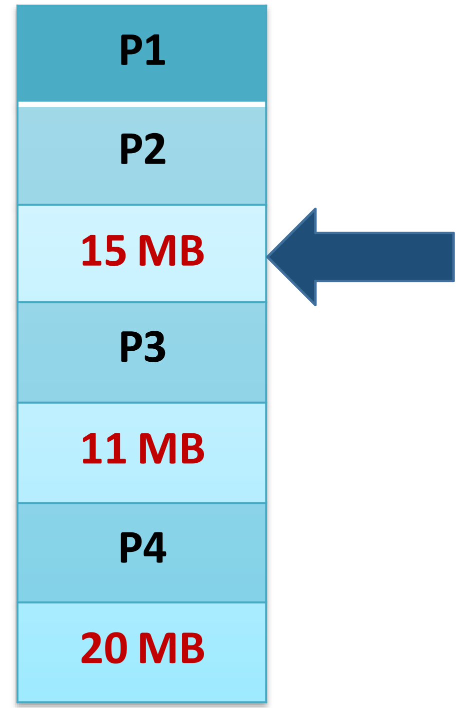
- **Advantage:**
  - No internal fragmentation.
  - No limitation on number of processes.
  - No limitation on process size.
- **Disadvantage:**
  - Causes **External fragmentation:**
    - Memory is allocated when process enters into system, and deallocated when terminates. This operation may leads to small holes in the memory.
    - This holes will be so small that no process can be loaded in it..
    - But total size of all holes may be big enough to hold any process.

Memory Allocation Strategies

- In **Partition Allocation**, when there is **more than one partition freely available** to accommodate a process's request, **a partition must be selected.**
- To **choose a particular partition**, a partition allocation method is needed. A partition allocation method is **considered better** if it **avoids internal fragmentation.**
- There are different Memory allocation Algorithm are:
  1. **First Fit**
  2. **Best Fit**
  3. **Worst Fit**

Memory Allocation Strategies

1. First Fit:

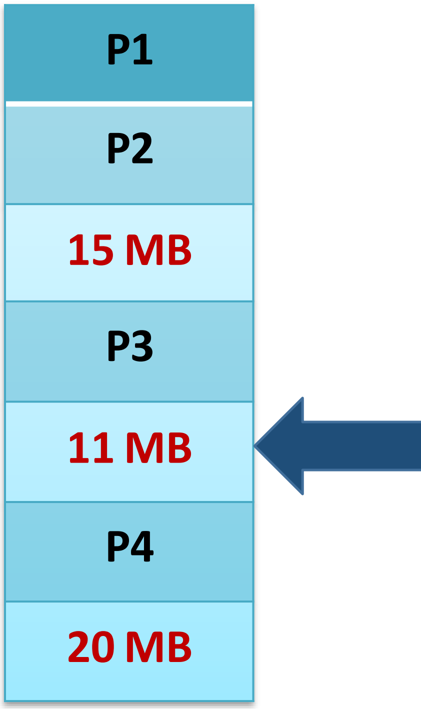


- In the first fit, the **partition is allocated** which is **the first sufficient block from the top** of Main Memory.
- It scans **memory from the beginning** and **chooses the first available block that is large enough.** Thus it allocates the first partition that is large enough.
- Example: processes P1, P2, P3, P4
- Are already in memory 3 free holes of 15 MB, 11 MB, 20 MB.
- **Now process P5 comes with needed memory is 10MB**

Memory Allocation Strategies

2. 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 partition to find the **smallest partition whose size is greater than or equal** to the **size of the process**.
- Example: processes P1, P2, P3, P4
- Are already in memory 3 free holes of 15 MB, 11 MB, 20 MB.
- **Now process P5 comes with needed memory is 10MB**



Memory Allocation Strategies

3. 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 partitions to **find the largest partition** and allocate it to process.
- Example: processes P1, P2, P3, P4
- Are already in memory 3 free holes of 15 MB, 11 MB, 20 MB.
- **Now process P5 comes with needed memory is 10MB**

P1
P2
15 MB
P3
11 MB
P4
20 MB

Fragmentation

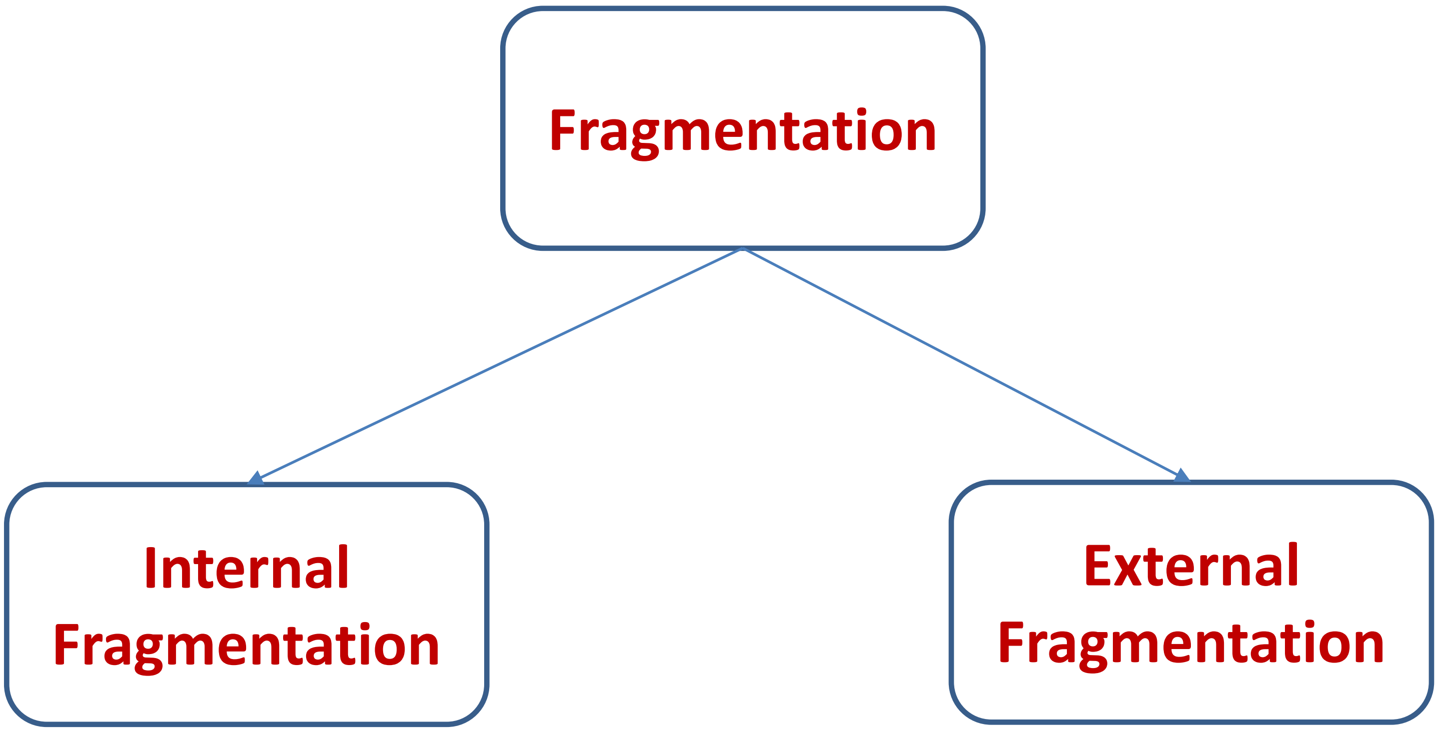
- **Fragmentation** is an unwanted problem in the operating system in **which the processes are loaded and unloaded from memory**, and free memory space is fragmented.

- Processes can't be assigned to memory blocks due **to their small size, and the memory blocks stay unused**.

- It is also necessary to understand that as programs are loaded and deleted from memory, they generate free space or a hole in the memory.

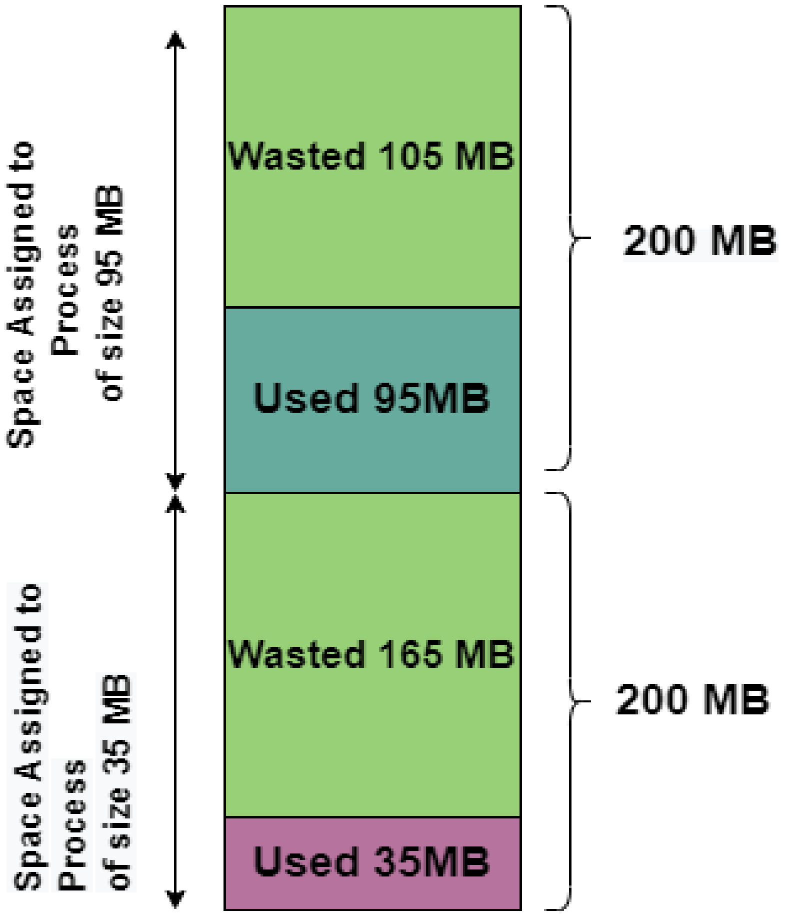
- **These small blocks cannot be allotted to new arriving processes, resulting in inefficient memory use**.

Fragmentation



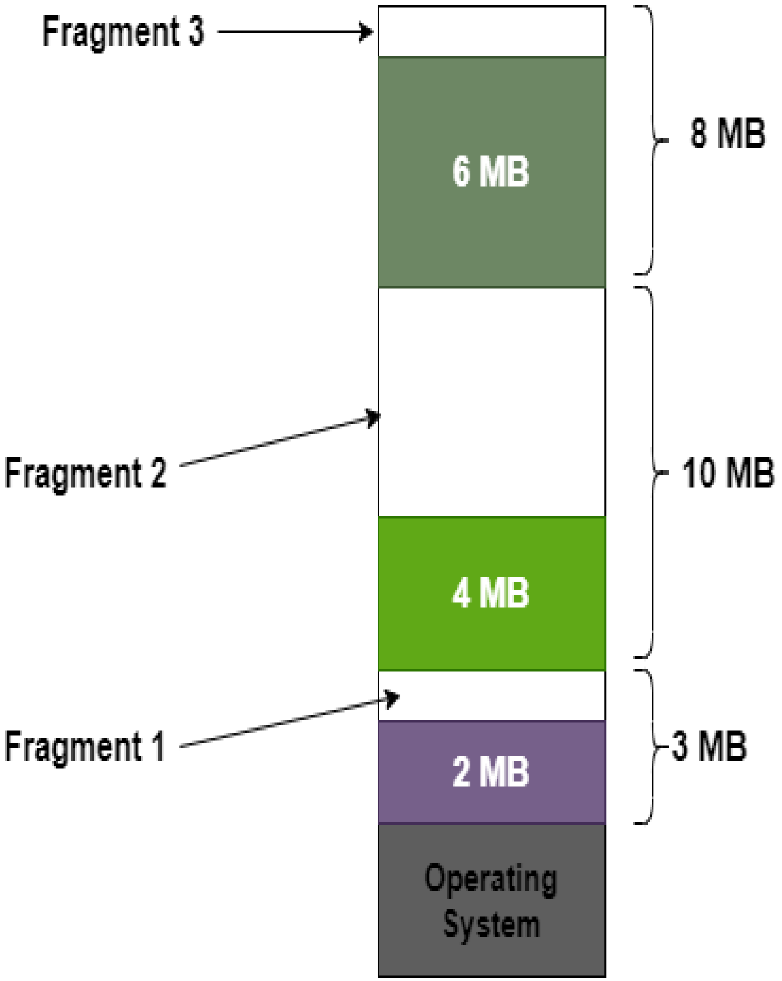
Internal Fragmentation

- **Internal Fragmentation** is a problem that occurs **when the process is allocated to a memory block whose size is more than the size of that process** and due to which some part of the memory is left unused.
- Thus, the space wasted inside the allocated memory block is due to the restriction on the allowed sizes of allocated blocks.



External Fragmentation

- When the memory space in the system can easily satisfy the requirement of the processes, but this available memory space is non-contiguous, So it can't be utilized further.
- Then this problem is referred to as **External Fragmentation.**



Internal Vs. External Fragmentation

S.NO	Internal fragmentation	External fragmentation
1.	In internal fragmentation **fixed-sized memory**, blocks square measure appointed to process.	In external fragmentation, **variable-sized memory blocks** square measure appointed to the method.
2.	Internal fragmentation happens when the method or process is **smaller than the memory**.	External fragmentation happens when the **method or process is removed**.
3.	The solution of internal fragmentation is the **best-fit block.**	The solution to external fragmentation is **compaction and paging.**
4.	Internal fragmentation occurs when memory is divided into **fixed-sized partitions.**	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.

Internal Vs. External Fragmentation

S.NO	Internal fragmentation	External fragmentation
5.	**The difference between memory allocated and required space or memory** is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are **too small to serve a new process**, which is called External fragmentation.
6.	Internal fragmentation occurs with **paging and fixed partitioning**.	External fragmentation occurs **with segmentation and dynamic partitioning**.
7.	It occurs on the allocation of a process to a partition greater than the process's requirement. The leftover space causes degradation system performance.	It occurs on the allocation of a process to a partition greater which is exactly the same memory space as it is required.
8.	It occurs in worst fit memory allocation method.	It occurs in best fit and first fit memory allocation method.

Non-Contiguous M/m Allocation

- Non-Contiguous memory allocation techniques are basically of two types:
  1. **Paging**
  2. **Segmentation**

- The main disadvantage of Dynamic Partitioning is **External fragmentation.**
- Although, this can be **removed by Compaction** but as we have discussed earlier, the compaction makes the system inefficient.
- That's why we come up with an idea of non-contiguous memory allocation.

Paging – Basic Idea and its Need

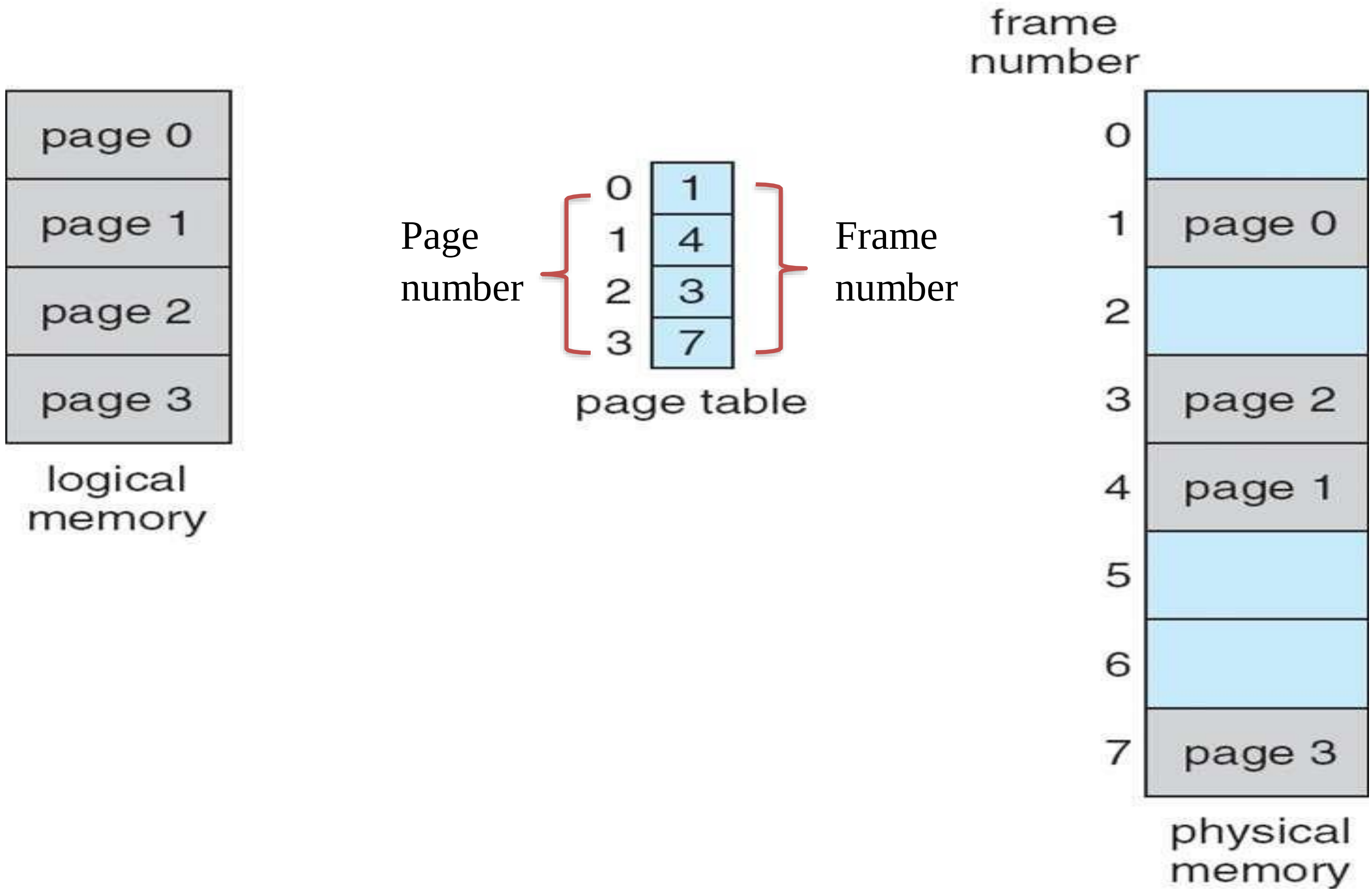
- **Physical memory (Main Memory)** is divided into fixed sized block called **frames.**
- **Logical address space (Secondary Memory)** is divided into blocks of fixed size called **pages.**
- **Page and frame will be of same size.**
- Whenever a process needs to get execute on CPU, its pages are moved from **hard disk** to available **frame in main memory.**

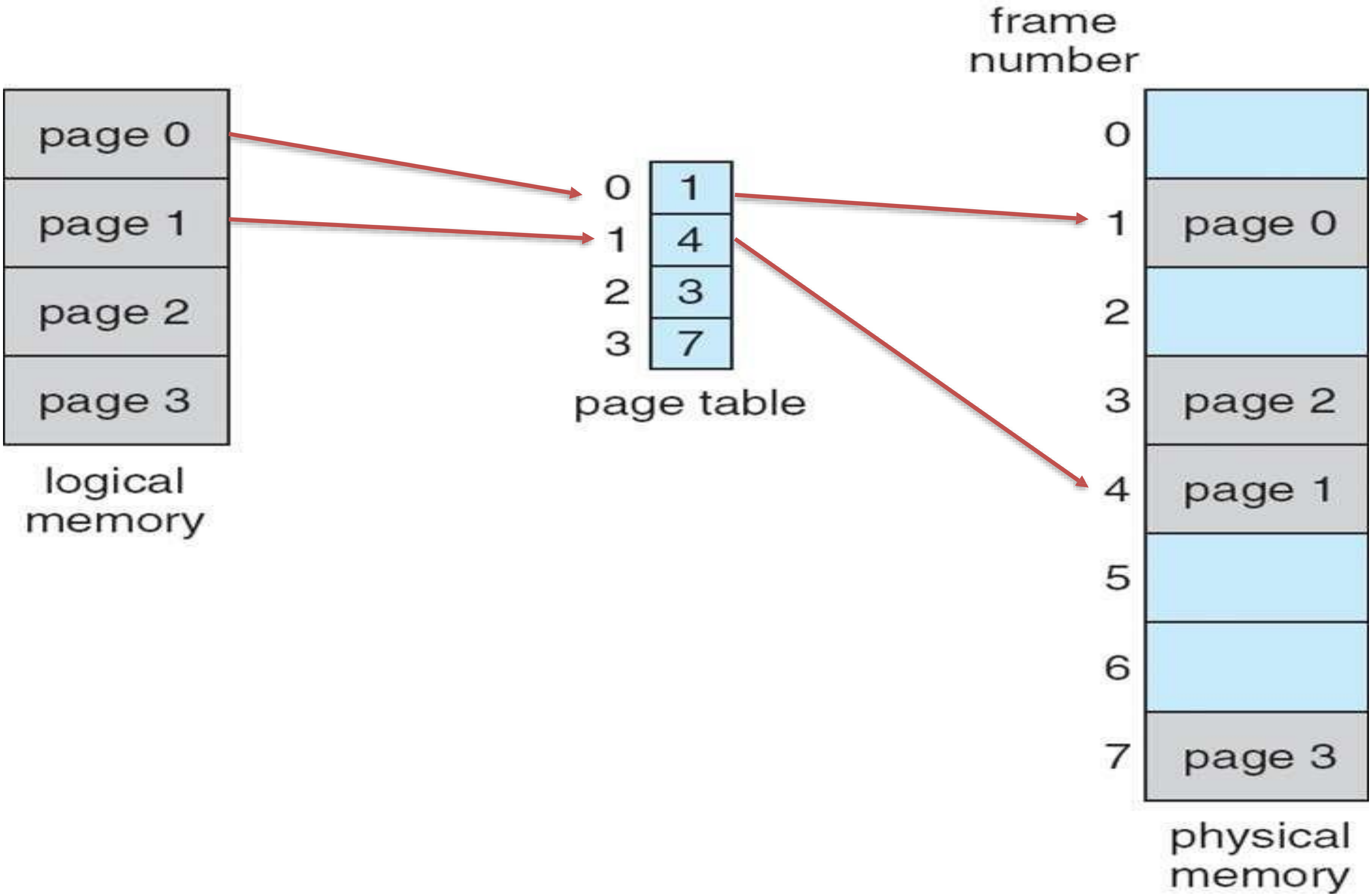
Paging

Thus here **memory management task** is:

- ✓ to **find free frame** in main memory,
- ✓ **allocate appropriate frame** to the page,
- ✓ **keeping track** of which **page belong** to **which frame.**

- OS maintains a **table called page table,** for each process.
- Page table is index by page number and stores the information about frame number.

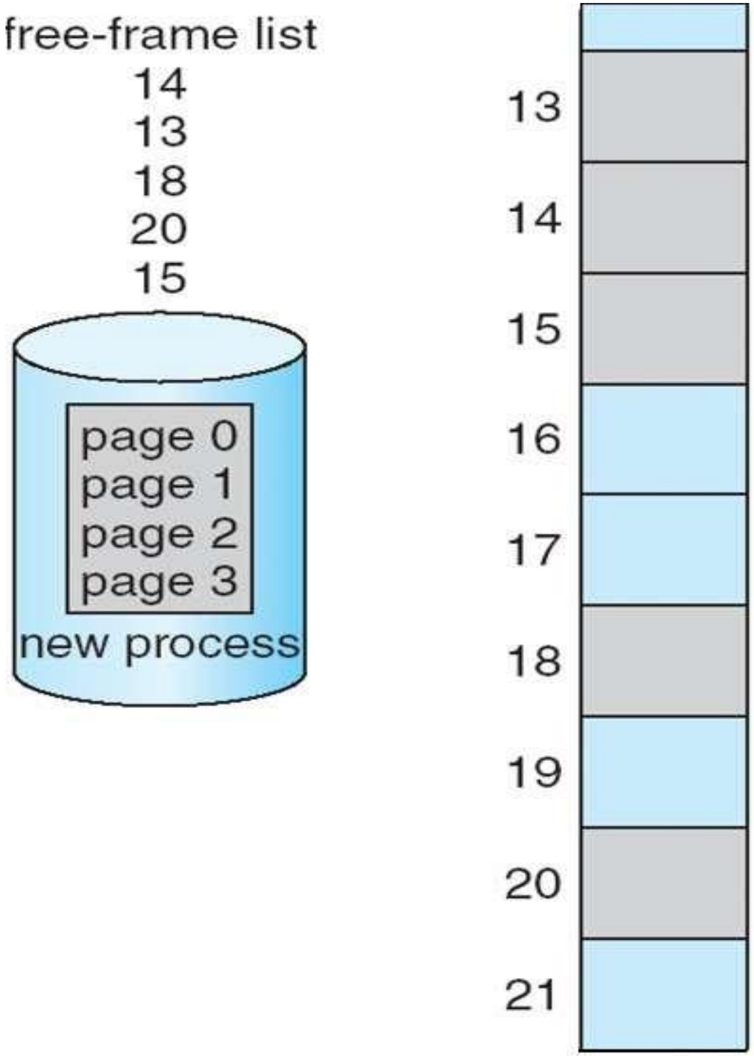

page 0
page 1
page 2
page 3
logical memory
Page number
0
1
2
3
1
4
3
7
Frame number
page table
frame number
0
1
2
3
4
5
6
7
page 0
page 2
page 1
page 3
physical memory


page 0
page 1
page 2
page 3
logical memory
0
1
2
3
1
4
3
7
page table
frame number
0
1
2
3
4
5
6
7
page 0
page 2
page 1
page 3
physical memory

Non-Contiguous Memory Allocation

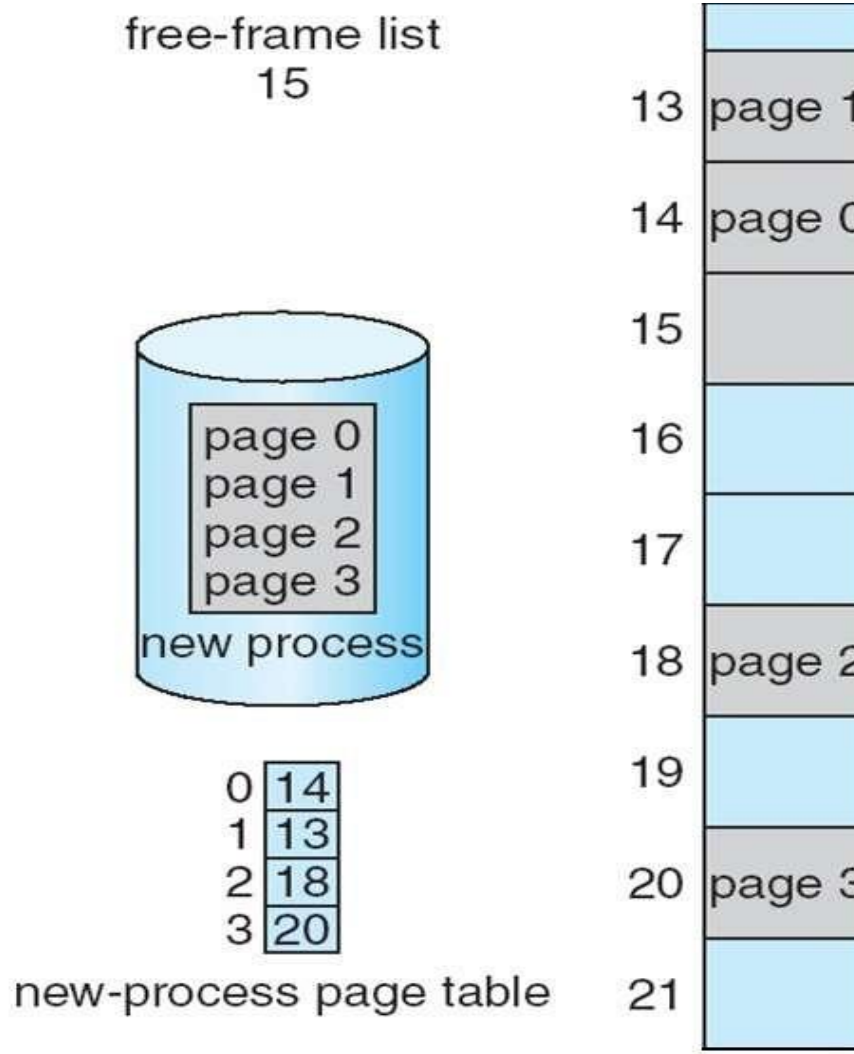
Page allocation in Paging system

- Here one ***free frame list*** is maintain.
- When a process arrives in the system to be executed, **size of process is expressed in terms of number of pages**.
- Each page of the process needs **one frame**.
- Thus if process requires n pages then n frames must be free in memory.



(a)

Before allocation



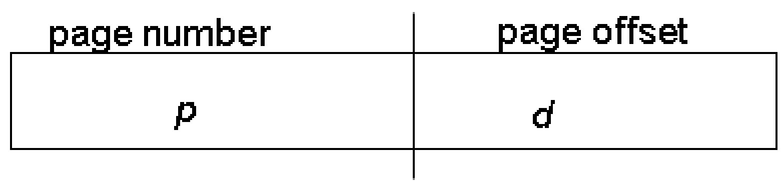
(b)

After allocation

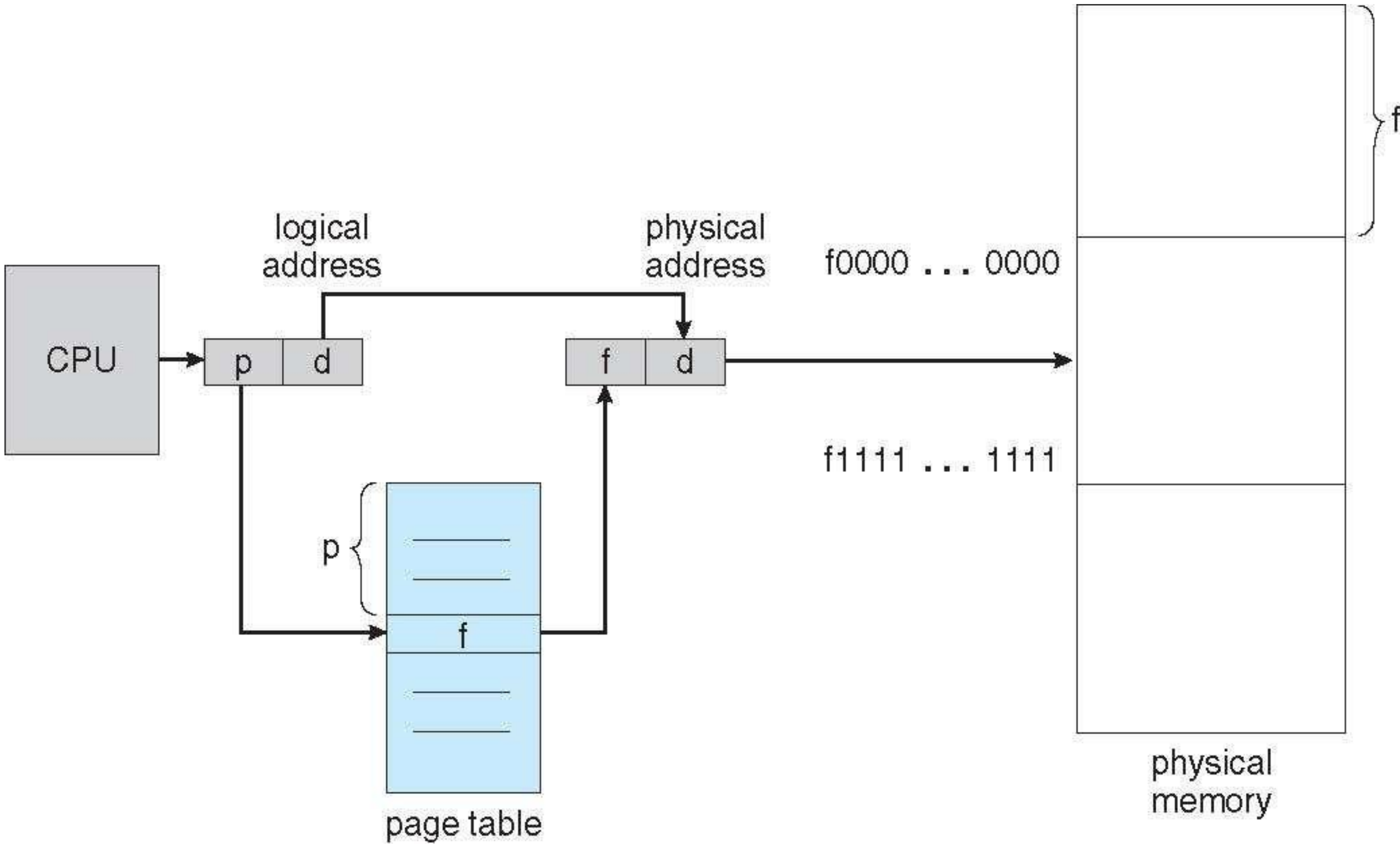
Address Mapping in Paging

- Here logical address (L) is divided into two parts:

1. **Page number (p)**
2. **Offset (d):** which gives us actual position in the page.



Address Mapping in Paging



Paging: Hardware support

- Modern OS uses variations of Paging which are:
  - **Translation look aside buffer**
  - **Hierarchical paging**
  - **Inverted page table**

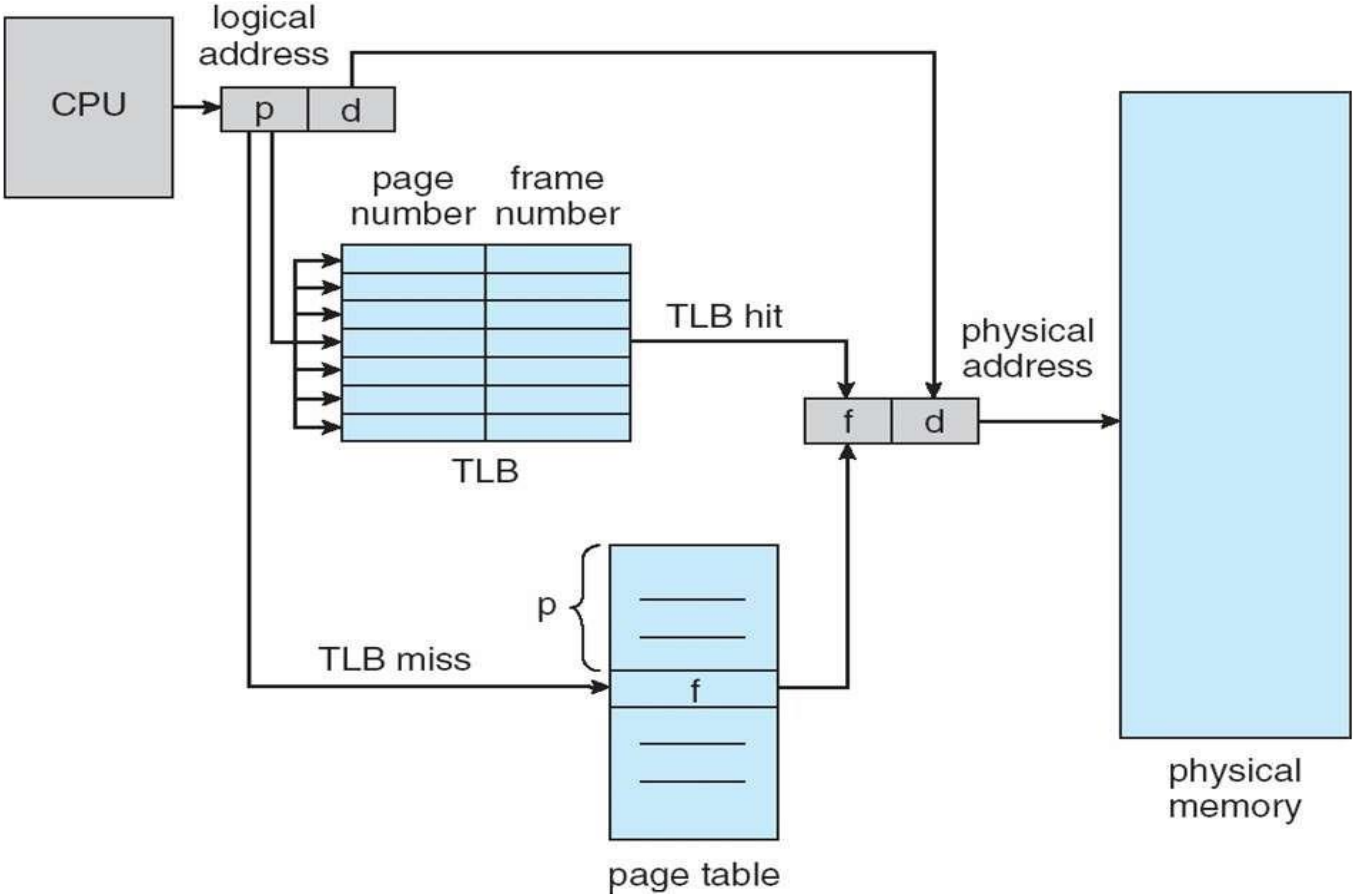
Translation look aside buffer

- **Used to overcome the slower access problem**.
- **TLB (Translation Look Aside Buffer)** is page table cache, which is implemented in **fast associative memory.**
- Its cost is high, so capacity is limited, thus only **subset of page** table is kept in memory.
- Each TLB contains a page number and a frame number, where the
  page is stored in the memory.

TLB - Working

1. Whenever a logical address is generated, the **page number of logical address is searched in the TLB.**
2. If the page number is **found**, then it is known as TLB hit. In this case corresponding frame number **is fetched from TLB entry and used to get physical address**.
3. If a match is **not found** then it is termed as **TLB miss**, in this case a memory reference to that page must be made. page table is used to get the frame number. **And this entry id moved to TLB.**
4. If TLB is full while moving the entry, then some of the existing entry in the the **TLB ae removed**. (strategy can be Least Recently Used(LRU) to random).

TLB - Working



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

TLB

- **Effective Access Time calculation:**
  - Q- 80 percent hit ratio in TLB, if it takes 20 nanosecond to search in TLB, 100 nanosecond to access main memory, then what is the effective access time to find a page?
  - A- Hit ratio is 80 percent and miss ratio is 20 percent

- Effective access time = **H(TLB + MM)+ M(TLB + PT + MM)**

  = H(TLB+MM)+ M(TLB+2MM)

  = 0.8(20+100) + 0.2(20+100+100)

  = 0.8 (120) + 0.2 (220)

  = 96 + 44 = **140 nano second**

Hierarchical Page Table

- Most modern OS supports large logical address space. In such case the page table itself become excessive large to store in contiguous space.
- Example system with 32-bit (2^32) =256 MB logical address space, if the page size if 4kb (4096=2^12), Then page table size would be 1 million entry (2^32/2^12).
- One of the solution is to **use two level paging algorithm**, in which **page table itself is also page.**

Hierarchical Page Table

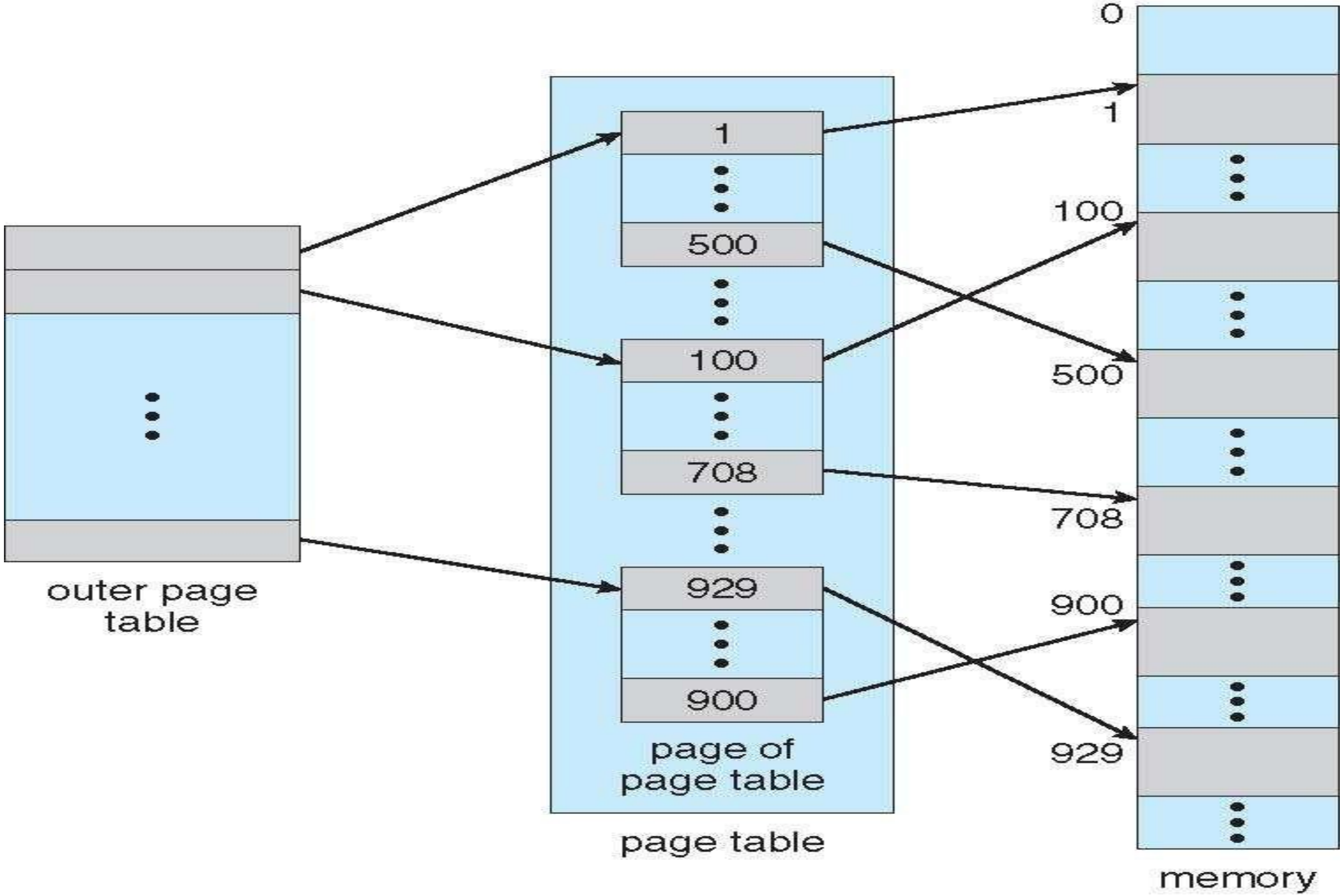
- Here the page table is divided into various **inner page table.** Also the extra **outer page table** is maintained.
- The outer table contains very limited entry, and points to the inner page table.
- And the inner page table in turn gives actual frame number.
- Here logical address is divided into three parts:

  Page number (p1) (outer page table page number)

  Page number (p2) (page table page number)

  Offset (d)

Hierarchical Page Table



Hierarchical Page Table

- **Advantage:**

All the inner pages need not be in memory simultaneously, thus reduce memory overhead.

- **Disadvantage:**

Extra memory access is required in each level of paging. Here total three memory access is required to get data.

Inverted Page Table

- Is also used to **overcome the problem** of memory overhead **due to large size of page table.**
- **In the previous approaches every process has a individual page table.** If the size of process is high then the number of entry in page table increases. Thus it become difficult to store more than one entry for single process.
- Inverted page table solve this problem: there is one entry per frame of physical memory in table, rather than one entry per page of logical address space.
- So the size of page table is depends on the size of physical memory.
- It will create one system wide table known as **global table.**

Inverted Page Table

- We organize the inverted page table as a hash table.
- We use hash function for given (process id, page number), we will apply hash function and look for match of process id and page number.
- If we have a match, translate the address. If not, use collision resolution technique (rehash, search, linear probing) and search again.
- There is just one page table in the entire system, implying that additional information needs to be stored in the page table to identify page table entries corresponding to each process(i.e. PID).

Inverted Page Table

Current process ID - 245

Page No 2	Offset

Hash Function

Process IDs do not match.
So, follow chaining pointer

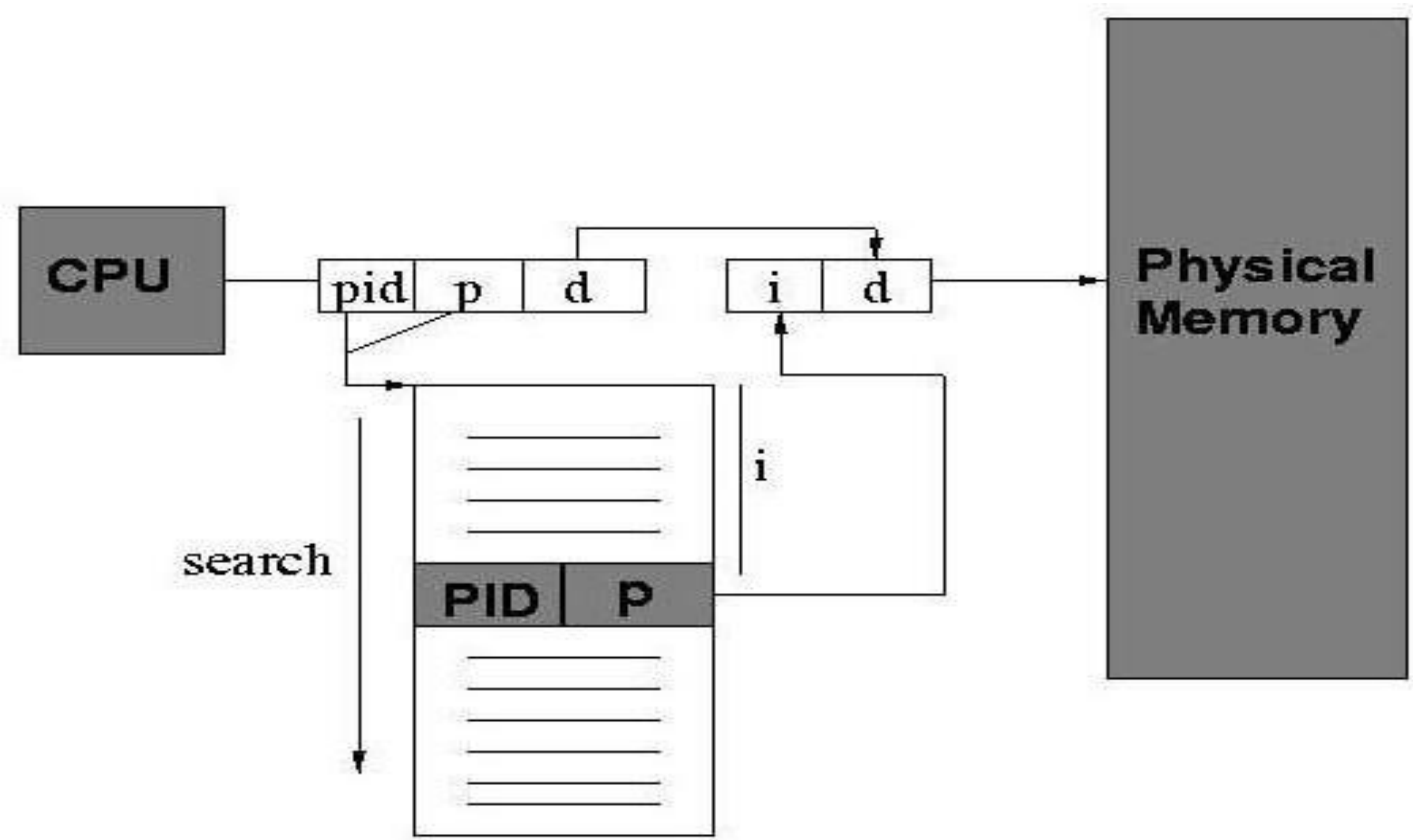
Frame No	Page No	Process ID	Pointer
1	2	211	
2			
3			
4	2	245	
5			
6			

Process IDs match.
So, frame no added to physical address.

Frame No 4	Offset

Physical Address

Inverted Page Table



Disadvantages of Paging

1. **Additional memory reference**

- Its required to read information from page table.
- Every instruction requires two memory accesses: one for page table, and one for instruction or data.

2. **Size of page table**

- Page tables are too large to keep it in main memory.
- Page table contain all pages in logical address space thus larger process page table will be large.

3. **Internal fragmentation**

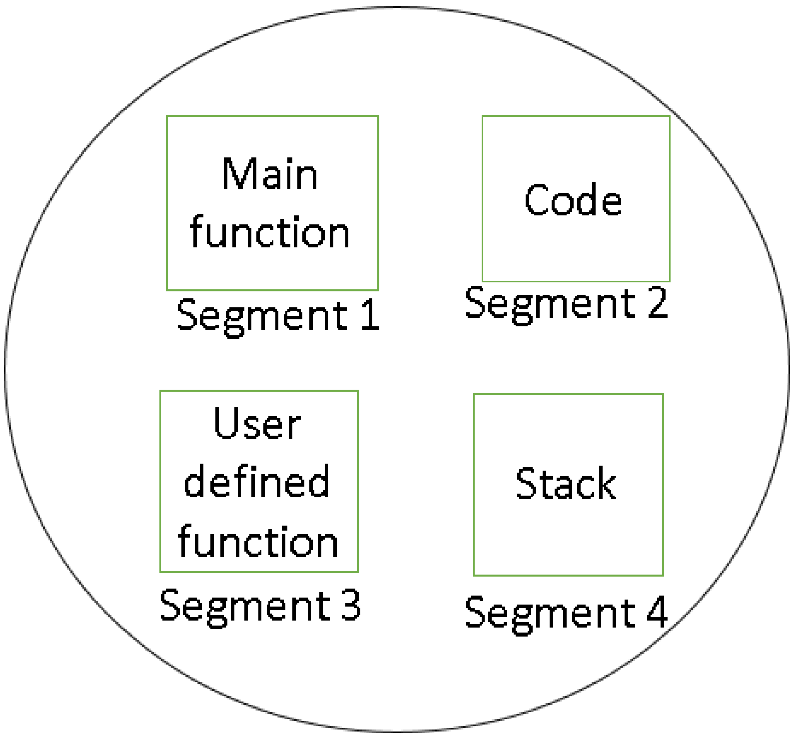
- A process size may not be exactly of the page size.
- So some space would remain unoccupied in the last page of a process. This result in internal fragmentation.

Segmentation

- Works on the user point of view, logical address space of any process is a collection of code, data and stack.
- Here the **logical address space** of a process is divided into **blocks of varying size, called segments.**
- **Each segment** contains a **logical unit** of process.
- When ever a process is to be executed, **its segments** are moved from **secondary storage** to the **main memory.**
- **Each segment** is allocated a ***chunk of free memory*** *of the* ***size equal to that segment*****.**
- OS maintains **one table** known as ***segment table,*** for each process. It includes **size of segment** and **location in memory** where the segment has been loaded.

Segmentation

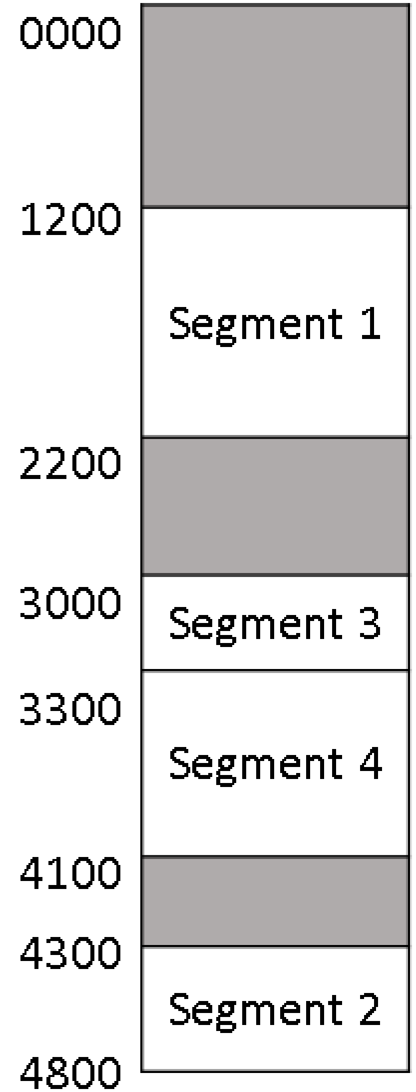
- Logical address is divided in to two parts:
  1. **Segment number:** identifier for segment.
  2. **Offset:** actual location within a segment.



Logical Address Space

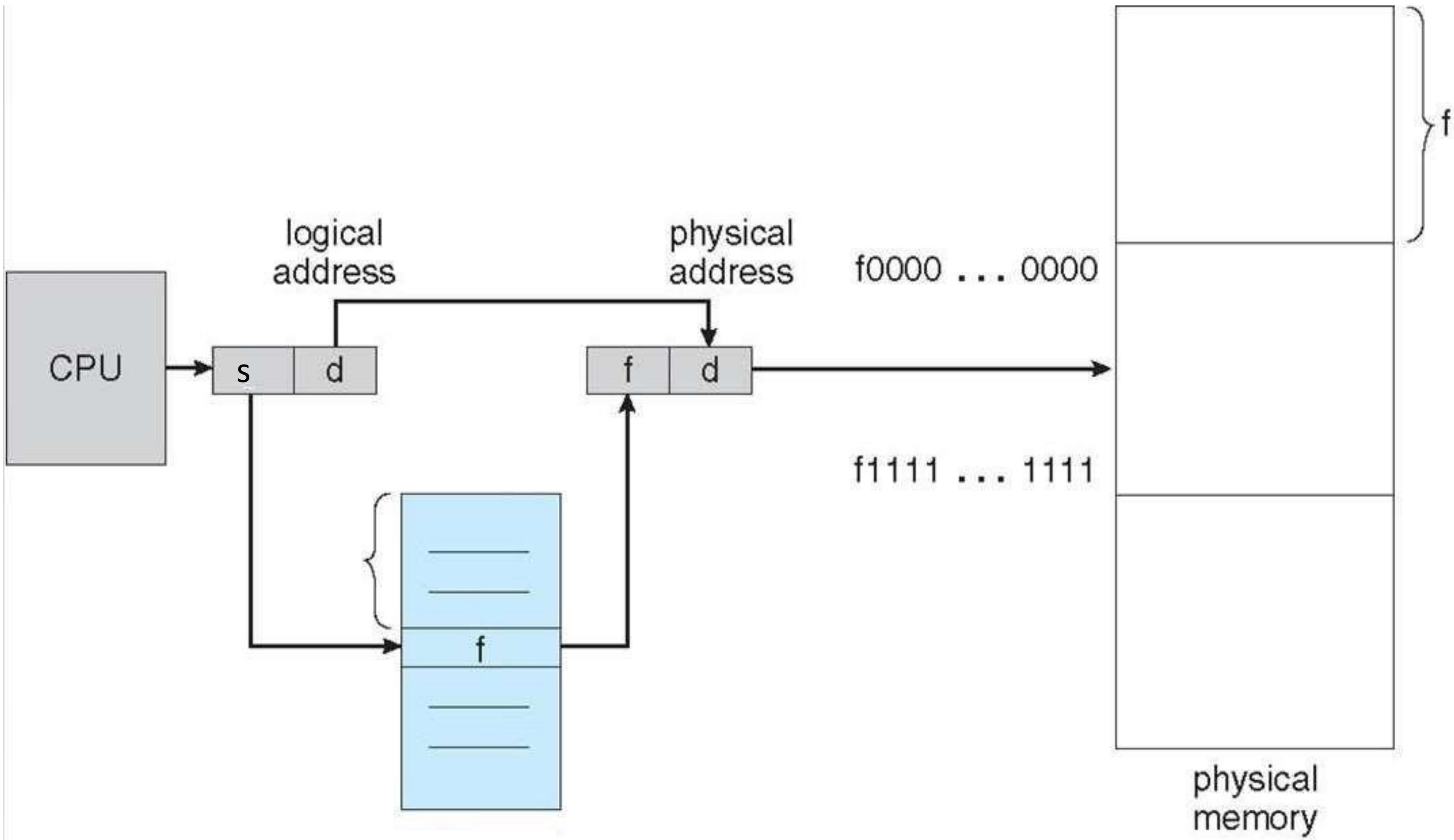
Segment No.	**Size of Segment**	**Base address**
1	1000	1200
2	500	4300
3	300	3000
4	800	3300

Segment Table

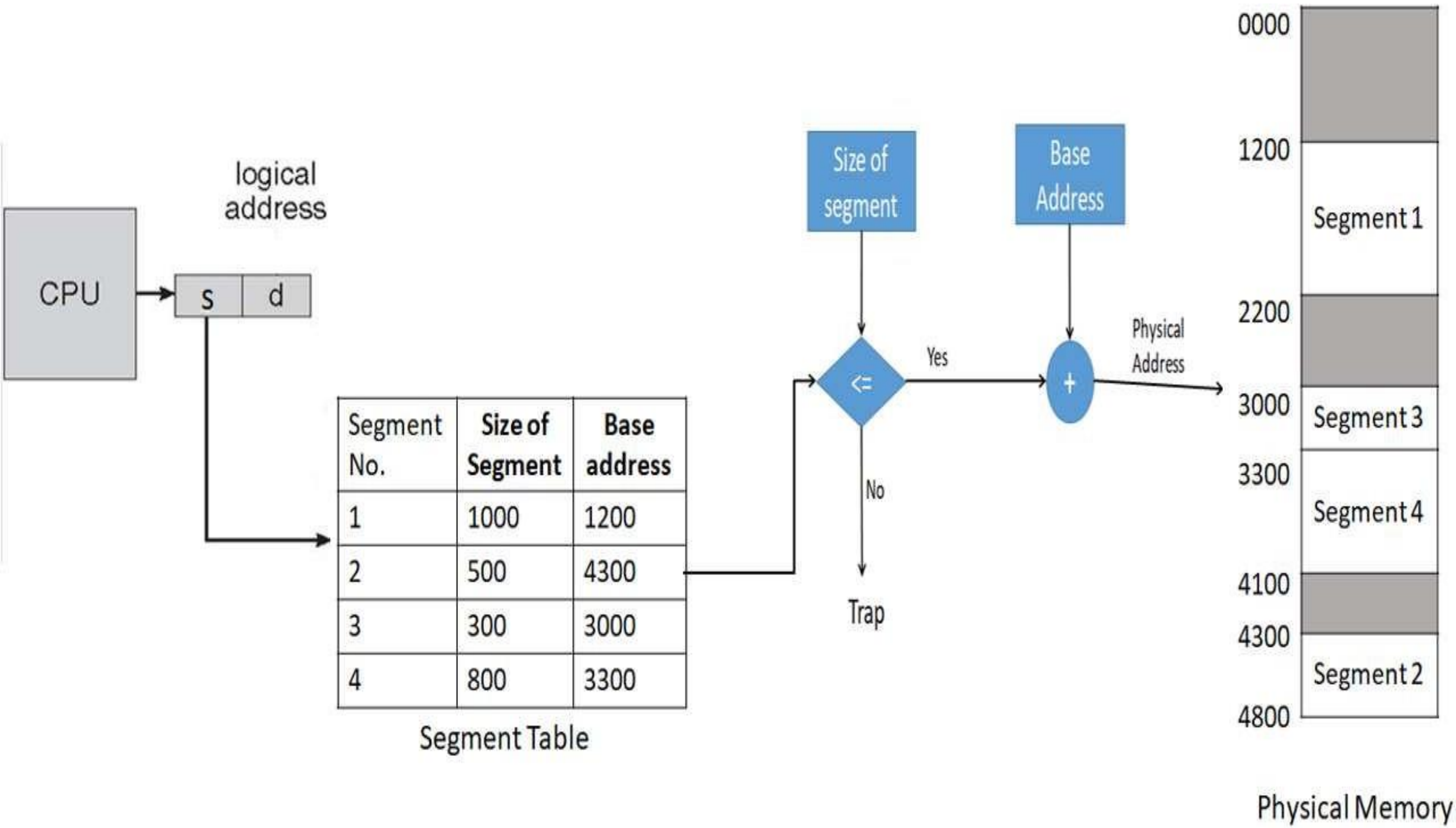


Physical Memory

Address Mapping in Segmentation



Address Mapping in Segmentation



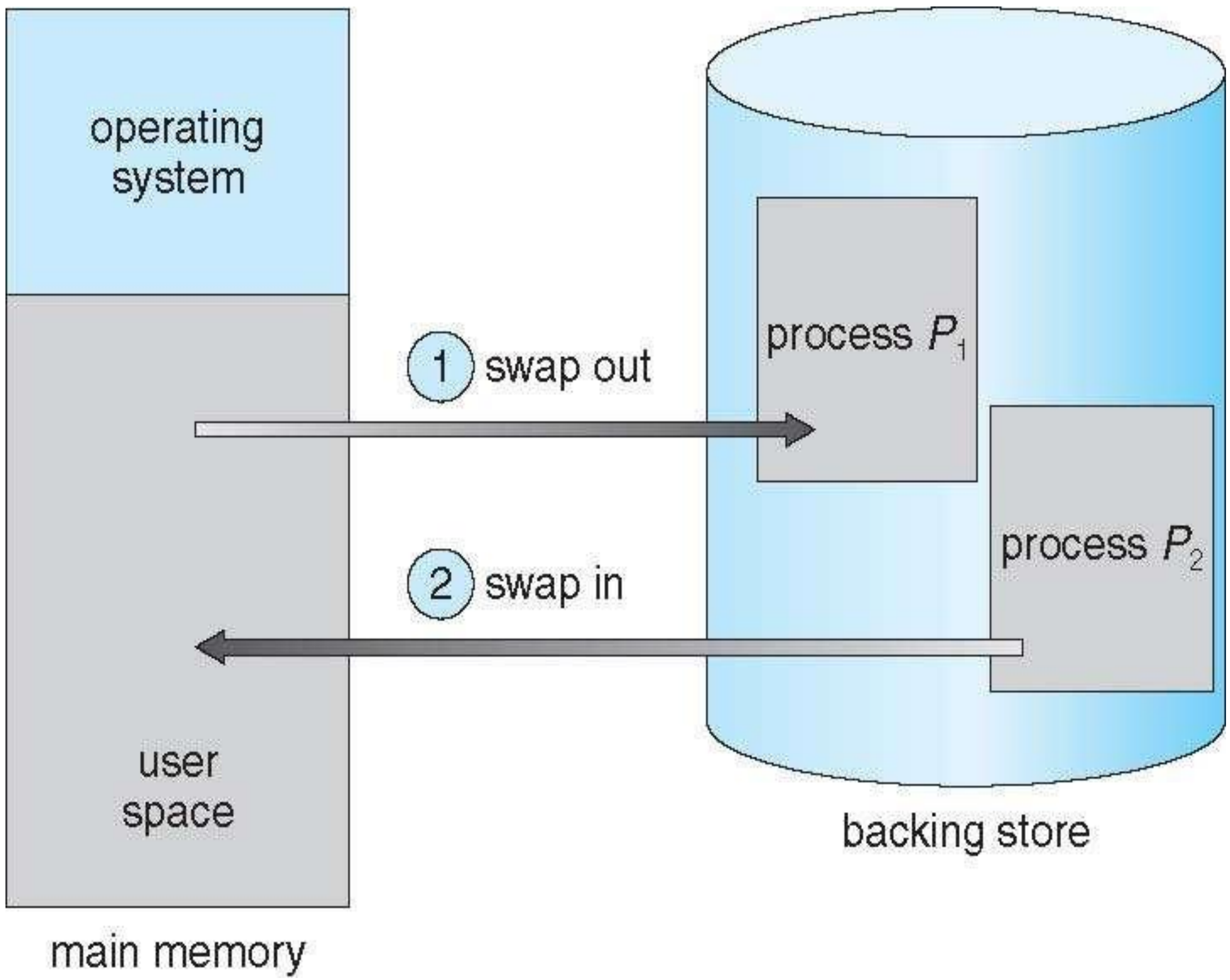
Segment No.	**Size of Segment**	**Base address**
1	1000	1200
2	500	4300
3	300	3000
4	800	3300

Segment Table

Swapping

- Example : A system has a physical memory of size 32-MB. Now suppose there are 5 process each having size 8MB that all want to execute simultaneously. How it is possible???
- **The solution is to use swapping.** Swapping is technique in which **process are moved** between **main memory** and **secondary memory or disk.**
- Swapping use some portion of secondary memory as backing store known as swapping area.
- **Operation of moving process** from **memory to swap area** is called **“swap out".** And **moving from swap area** to **memory** is known as **“swap in”.**

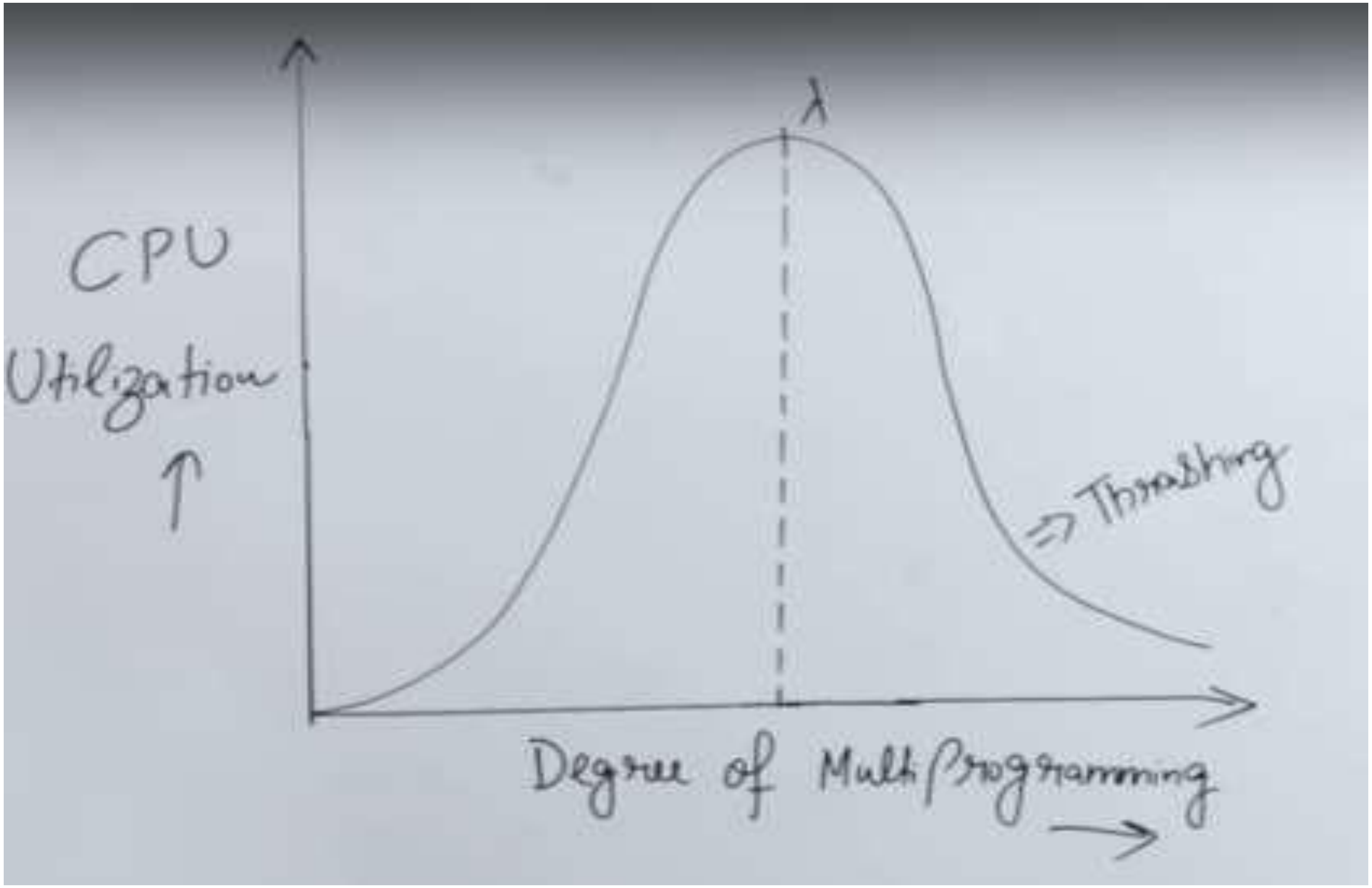
Swapping



operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Thrashing

- CPU utilization is directly linked to degree of multi programming.
- As RAM is limited, so we are using paging concept here.
- For e.g. I have 100 processes and each process is divided into some number of pages.
- Degree of multiprogramming is maximum, if I will place one page of each process into RAM.
- Degree of multiprogramming is achieved here as every process have its one page available in the RAM, but it causes maximum page faults.


λ
CPU
Utilization
⇒ Thrashing
Degree of MultiProgramming

Thrashing

- Due to this **performance** of the **system is decreased** .
- After a certain limit λ thrashing occurs.
- **To avoid this problem :**

i. **Increase the main memory size**

ii. **Efficiently use long term scheduler**.

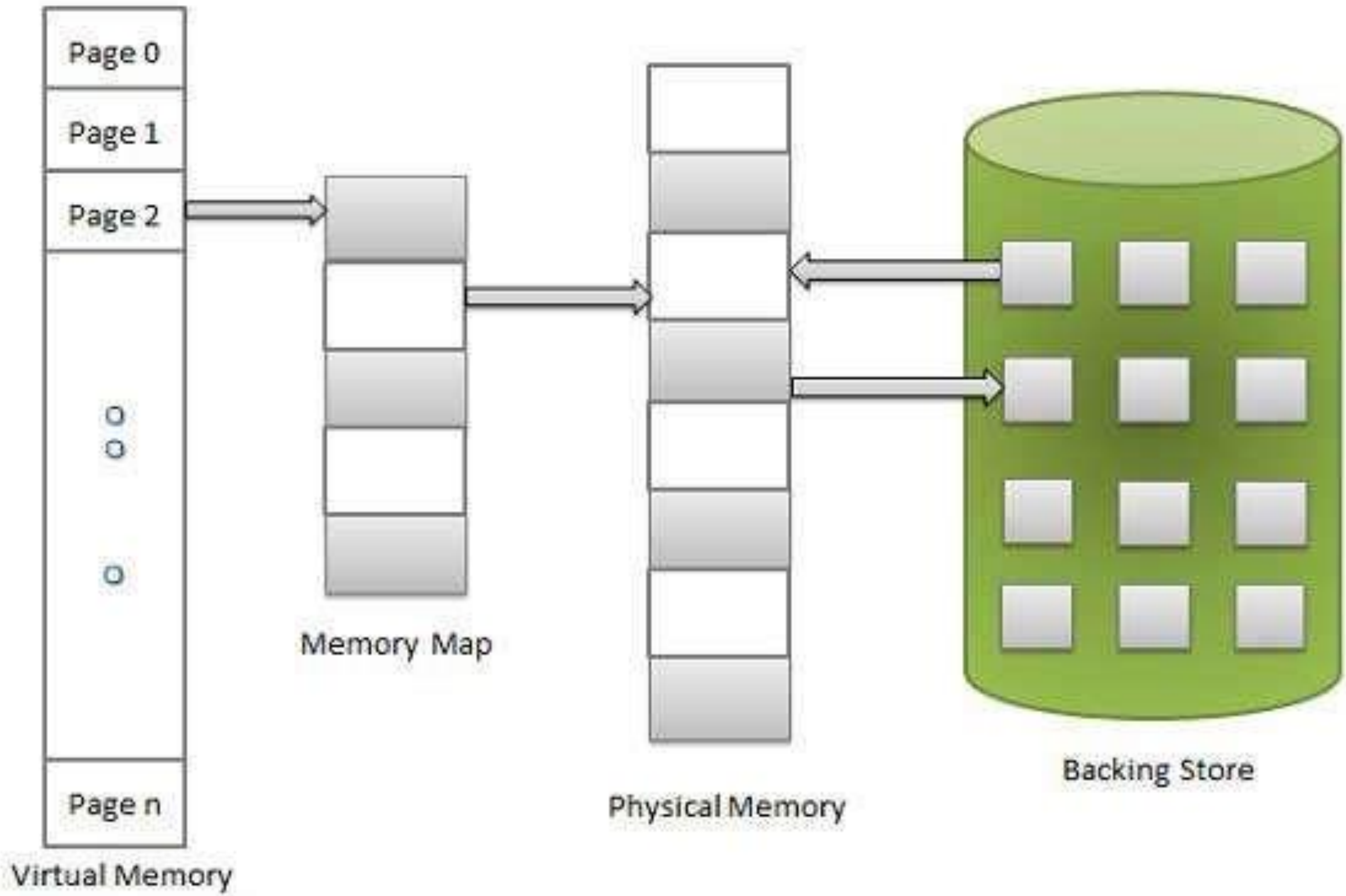
P1 – page1
P2– page1
P3 – page1
P4 – page1
…
P100 – page1

Virtual Memory

- A virtual memory is technique that allows a process to execute even though it is partially loaded in main memory.
- The basic idea behind virtual memory is that the combined size of the program, data, and stack may exceed the amount of physical memory (main memory) available for it.
- The operating system keeps those parts of the program currently in use in main memory, and the rest on the disk.
- These program-generated addresses are called virtual addresses and form the virtual address space.
- **MMU (Memory Management Unit)** maps the virtual addresses onto the physical memory addresses

Advantage of Virtual Memory

- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available. User would be able to write programs for an extremely large virtual address space.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.


Page 0
Page 1
Page 2
Page n
Virtual Memory
Memory Map
Physical Memory
Backing Store

Virtual Memory: Hardware and control structures

- Virtual memory involves the separation of logical memory perceived(aware) by user from physical memory.
- This separation allows extremely large virtual memory to be provided for programmers when only smaller amount of physical memory is available.
- Thus programmer need not to worry about the amount of memory available.

Virtual Memory

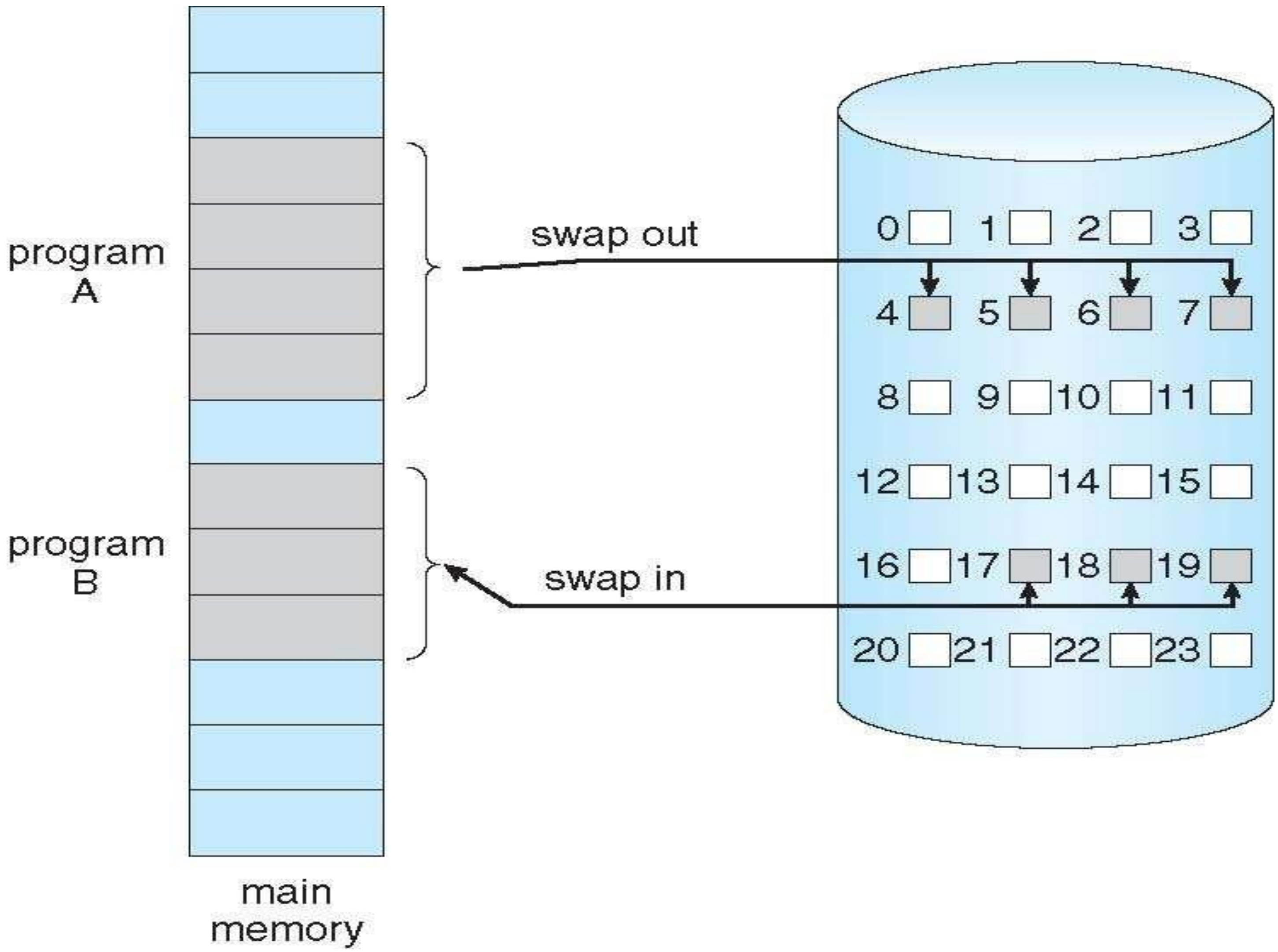
Virtual memory can be implemented in following three ways:

- Demand paging
- Demand Segmentation
- Segmentation with Paging

Demand Paging

- Demand paging is similar to a paging system with swapping where processes may reside in secondary memory. When we want to execute a process, we swap it into the main memory.
- Rather than swapping entire process into memory we use lazy swapper.
- A lazy swapper never swaps page into memory, unless that page will be needed.
- If some process is needs to be swap in, then pager(page table handler) brings only those pages which will be used by process.
- Thus avoid reading unused pages and decrease swap time and amount of physical memory needed.

Demand Paging



Demand Paging- Page Fault

If a process tries to access a page that is not in main memory then it causes page fault.

Pager will generate trap to the OS, and tries to swap in.

Page table includes the **valid-invalid bit** for each page entry.

If the bit is valid then page is currently available in to the memory.

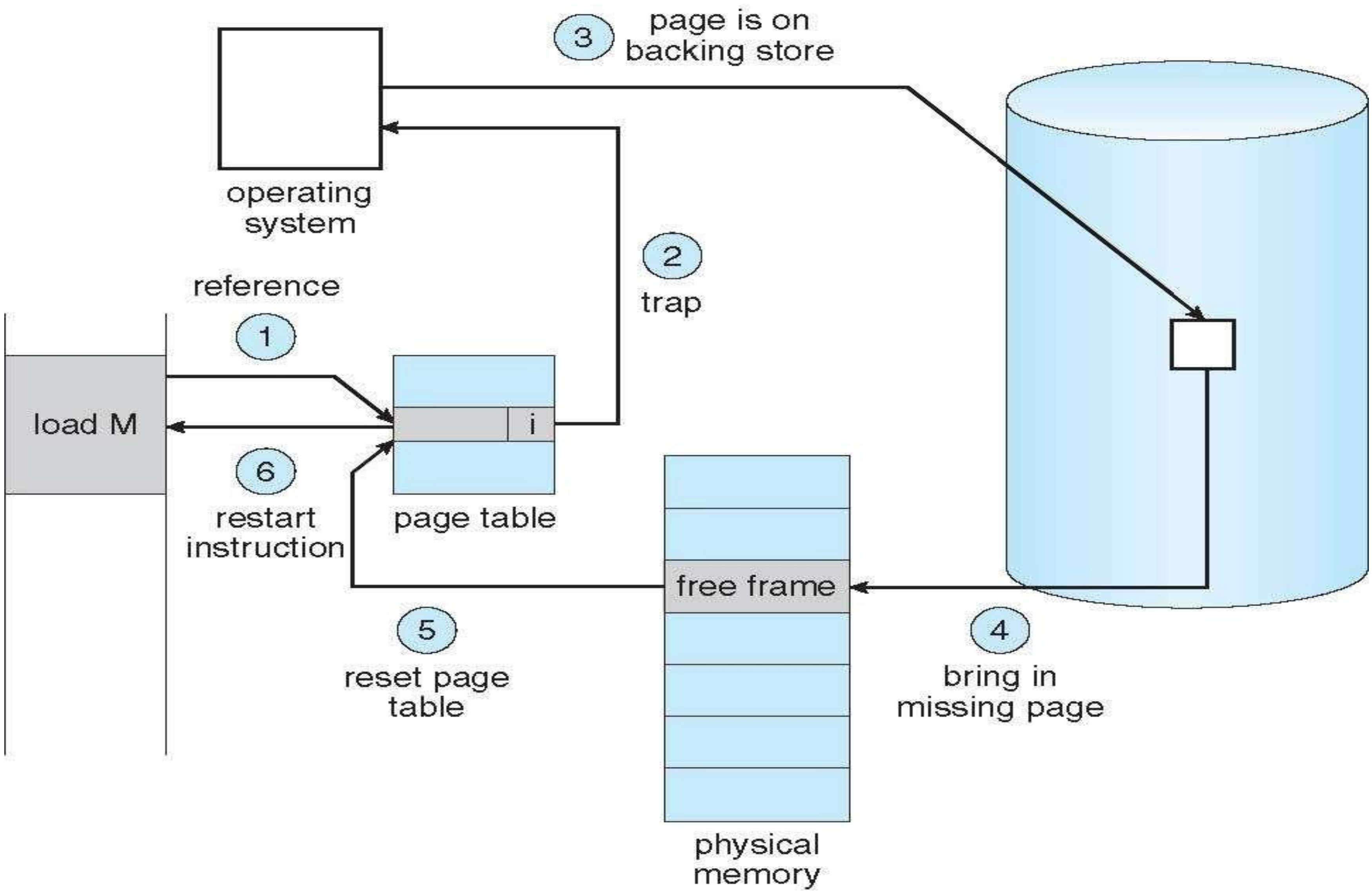
If it is set to invalid then page is either invalid or not present in main memory.

Demand Paging – Page Fault

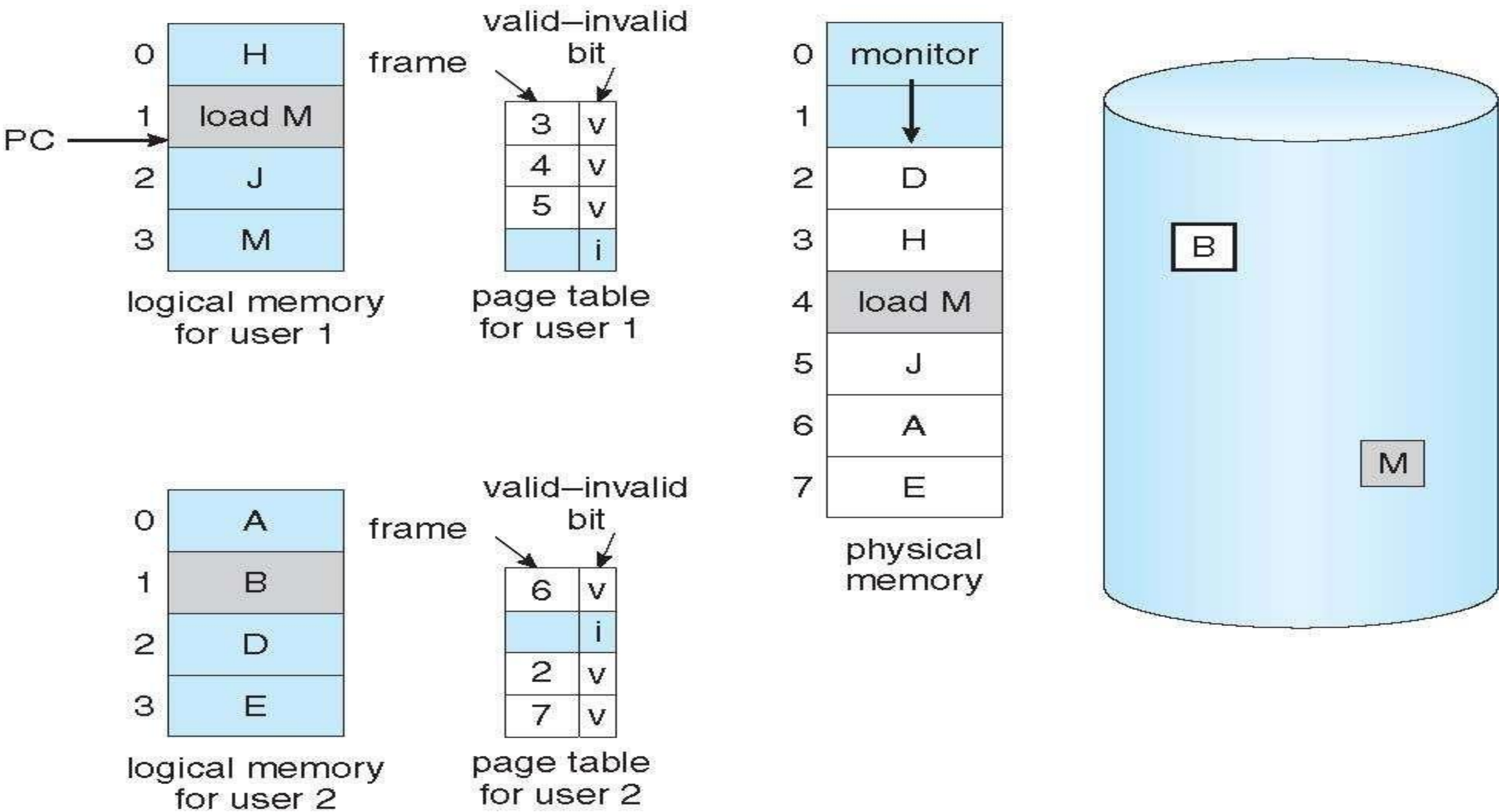
Following steps are followed to manage page fault:

1. Check page table for the process to determine whether the reference is valid or invalid.
2. If the page is invalid then terminate the process, but if the page is valid but currently not available in main memory, then generate trap instruction.
3. OS determines the location of that page on swap area.
4. Then it will use free frame list to find out free frame. OS will schedule disk operation to read desired page into newly allocated memory.
5. When disk read is complete modify the page table and set reference bit to valid.
6. Restart the instruction.

Demand Paging – Page Fault



Page Replacement Policies - Need



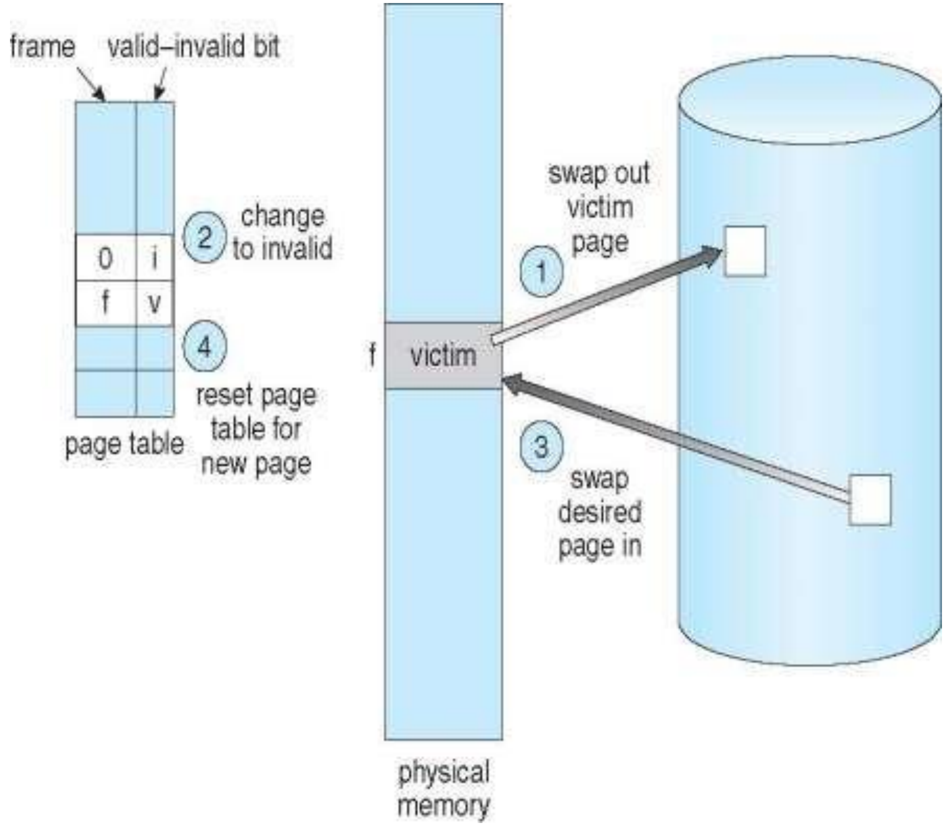
Page Replacement Policies - Basics

1. Find the location of the desired page on disk.

2. Find a free frame:
   - If there is a free frame, use it.
   - If there is no free frame, use a page replacement algorithm to select a victim frame
   - Write victim frame to disk if dirty

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Continue the process by restarting the instruction that caused the trap



First In First Out (FIFO)

- The simplest page replacement algorithm.
- When the page must be replaced the oldest page is chosen.
- one FIFO queue is maintained to hold all pages in memory.
- Replace the page which at the top of the queue and add new pages from rear end (tail)of the queue.

Reference String : 2 3 2 1 5 2 4 5 3 2 5 2

- 3 frames (3 pages can be in memory at a time per process)

- **Reference String : 7,0,1,2,0,3,0,4,2,3,0,3,1,2,0**
- 3 frames (3 pages can be in memory at a time per process) FIFO

F3			1	1	1	1	0	0	0	3	3	3	3	2	2
F2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	F	*	*	*	hit	*	*	*	*	*	*	2hit	*	*	Hit

First In First Out (FIFO)

- Reference String : **7,0,1,2,0,3,0,4,2,3,0,3,1,2,0**
- 3 frames (3 pages can be in memory at a time)

F3			1	1	1	1	0	0	0	3	3	3	3	2	2
F2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
F1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	F	F	F	F	Hit	F	F	F	F	F	F	hit	F	F	hit

Page faults/Page Miss = 12 **Page hit** = 3

- **Miss Ratio** = Num of miss/ num of reference => (12/15)*100= 80%
- **Hit Ratio** = Num of hit/ num of reference => (03/15)*100= 20%

First In First Out (FIFO)

- Reference string: **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**
- 3 frames (3 pages can be in memory at a time per process)

9 page faults

- **Find out the total page faults in the given reference string?**

By using 4 frames

- **10 page faults and 2 page hits**

Belady’s Anomaly in FIFO

time →

Access pattern	0	1	2	3	0	1	4	0	1	2	3	4
Physical memory (3 page frames)	0	0	0	1	2	3	0	0	0	1	4	4
		1	1	2	3	0	1	1	1	4	2	2
			2	3	0	1	4	4	4	2	3	3

9 page faults!

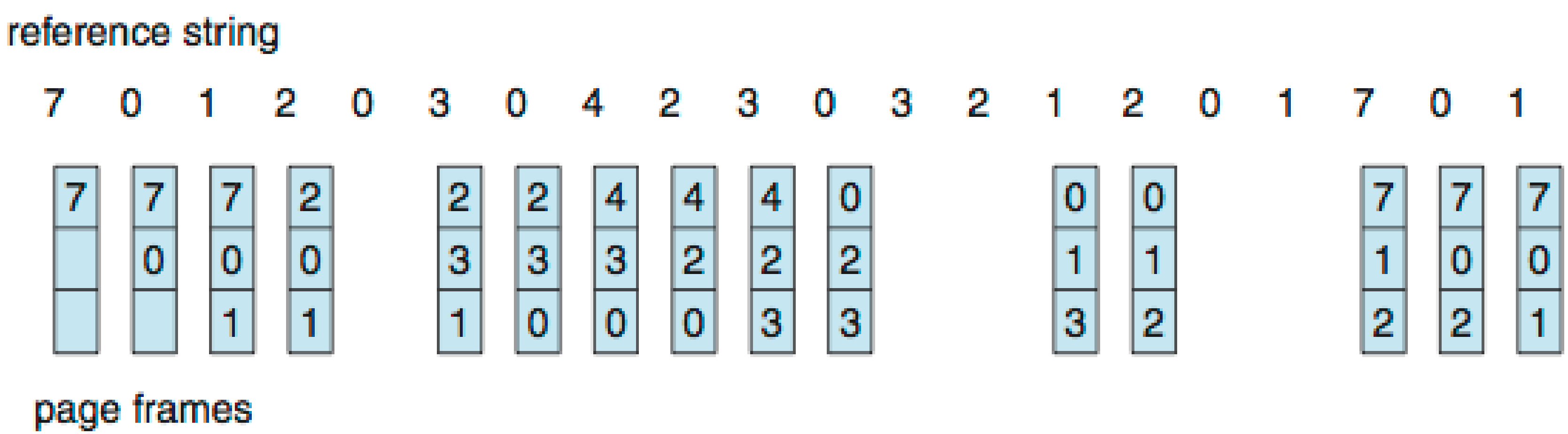
time →

Access pattern	0	1	2	3	0	1	4	0	1	2	3	4
Physical memory (4 page frames)	0	0	0	0	0	0	1	2	3	4	0	1
		1	1	1	1	1	2	3	4	0	1	2
			2	2	2	2	3	4	0	1	2	3
				3	3	3	4	0	1	2	3	4

10 page faults!

First In First Out (FIFO)

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)



- 15 page faults

First In First Out (FIFO)

Advantage

- Very simple.
- Easy to implement.

Disadvantage

- A page fetched into memory a long time ago may have now fallen out of use.
- This reasoning will often be wrong, because there will often be regions of program or data that are heavily used throughout the life of a program.
- Those pages will be repeatedly paged in and out by the FIFO algorithm.

Least Recently Used (LRU)

It is based on the observation that if pages that have been *heavily used in the last few instructions will probably be heavily used again in the next few.*

Conversely, pages that have not been used for ages will probably remain unused for a long time.

This idea suggests a realizable algorithm: when a page fault occurs, throw out the page that has been *unused for the longest time*.

Least Recently Used (LRU)

- Example-1 Reference string: 1, 2, 3, 4, 1, 2, **5**, 1, 2, **3**, **4**, **5**
- **8 page faults**

(replace the least recently used pages in past)

F1	**1**	**1**	**1**	**1**	**1**	**1**	**5**
F2	2	2	2	2	2	2	2
F3	3	**3**	5	5	**5**	4	4
f4	4	4	4	**4**	3	3	3
	4*	2hit	*	2hit	*	*	*

Least Recently Used (LRU)

- Example-2
- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per process)

F1	7	2	2	2	2	4	4	4	0	0	1	1	1	1	1	1
F2	0	0	0	0	0	0	0	3	3	3	3	3	0	0	0	0
F3	1	1	1	3	3	3	2	2	2	2	2	2	2	2	7	7
	3*	*	hit	*	hit	*	*	*	*	4hit	*	hit	*	hit	*	2hit

Least Recently Used (LRU)

1. Reference string: **1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6**

2. Reference string: **0 1 7 2 3 2 7 1 0 3**

- 4 frames (4 pages can be in memory at a time per process)

- 10 page faults

- 3 frames (3 pages can be in memory at a time per process)

- 15 page faults

Most Recently Used (MRU)

Idea of MRU- Replace the page which is most recently used in past.

- Example
- 1. Reference string: **7,0,1,2,0,3,0,4,2,7,3**
- 2. Reference string: **0 1 7 2 3 2 7 1 0 3**

3 frames (3 pages can be in memory at a time per process)

F1	7	7	7	7	7	7	7	3
F2	0	0	0	3	0	4	4	4
F3	1	2	2	2	2	2	2	2
	3*	*	hit	*	*	*	2hit	*

Optimal Page Replacement

- Its best page replacement policy.
- The Optimal policy selects for replacement the page that will *not be used for longest period* of time.
- Impossible to implement (need to know the future) but serves as a standard to compare with the other algorithms we shall study.

Optimal Page Replacement

- **For example: 3 frames**
- Reference string: **1,2,3,4,2,1,5,6,2,1,2,3,7,6,3,2,1,2,3,6**

F1	1	1	1	1	1	1	3	3	3	3	3	3	6
f2	2	2	2	2	2	2	2	7	7	2	2	2	2
f3	3	4	4	5	6	6	6	6	6	6	1	1	1
	3*	*	2hit	*	*	3hit	*	*	2hit	*	*	2hit	*

Optimal Page Replacement

- Reference string : **1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5**(check next pages)
- 4 frames

F1	1	1	1	1	4	
F2	2	2	2	2	2	
F3	3	3	3	3	3	
f4	4	4	5	5	5	
	4*	2 hit	*	3hit	*	

Optimal Page Replacement

- Need an approximation of how likely each frame is to be accessed in the future
- If we base this on past behavior we got a way to track future behavior
- Tracking memory accesses requires hardware support to be efficient

Optimal Page Replacement

Advantage:

- Lowest page faults.
- Can Improves performance of system as it reduces number of page faults so requires less swapping.

Disadvantage:

- Very difficult to implement.


THANK
YOU