Unit-IV

Memory Management Requirements, Memory Partitioning: Fixed Partitioning, Dynamic Partitioning, Memory Allocation Strategies: Best-Fit, First Fit, Worst Fit, Next Fit, Relocation, Paging, Segmentation.

Virtual Memory: Demand Paging, Structure of Page Tables, Page Replacement Strategies: FIFO, Optimal, LRU, LFU, Thrashing.

CO 3: Illustrate various memory management techniques.

❖ Memory Management Requirements

The memory management is intended to satisfy the following requirements:

1. Relocation
2. Protection
3. Sharing
4. Logical organization
5. Physical organization

Memory Management Requirements Continued…

1. Relocation

- In a multiprogramming system, the available **main memory is generally shared** among a number of processes.
- We need to swap **active processes** in and out of main memory **to maximize processor utilization** by providing a large pool of ready processes to execute.
- Once a program is swapped out to disk, it would be quite limiting to specify that when it is next swapped back in, it **must be placed in the same main memory region as before**.
- Instead, we may **need to relocate** the process to a different area of memory.

Relocation Continued…

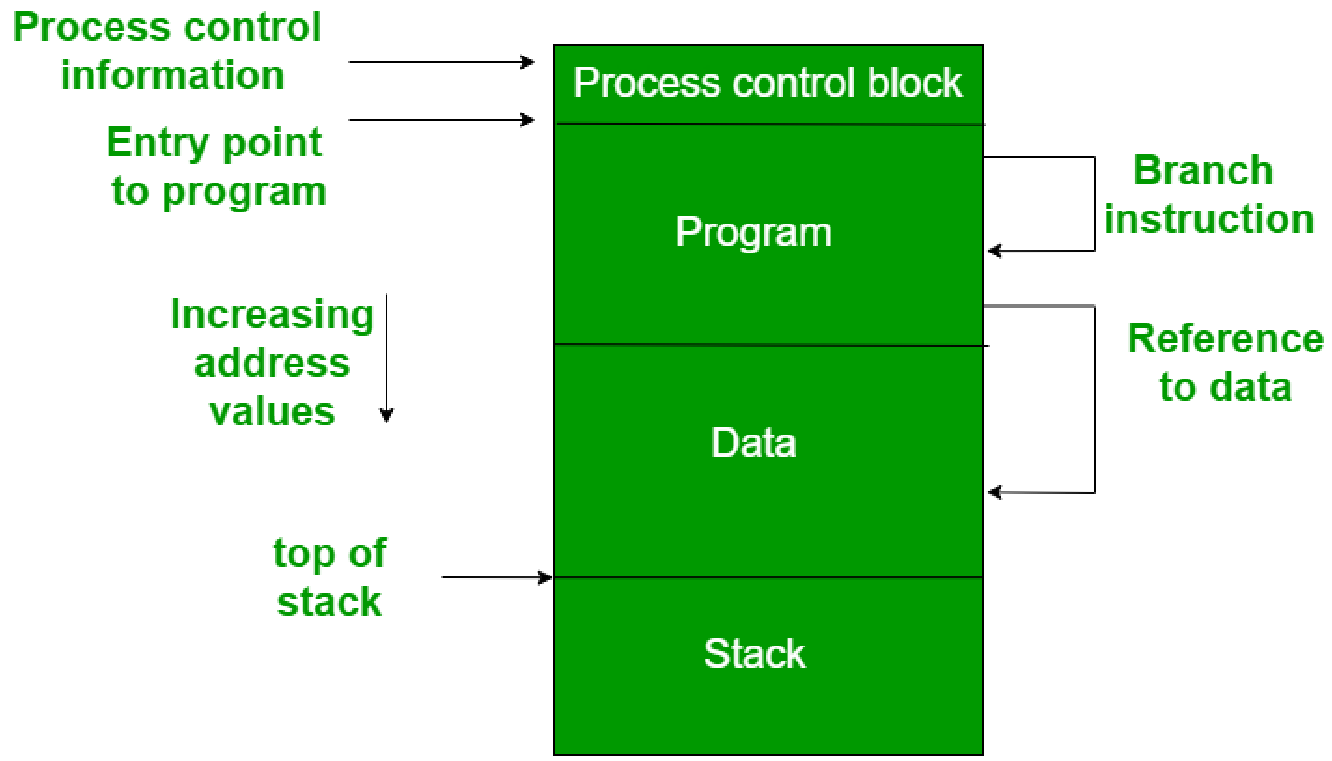


Figure: a process image

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

2. Protection

- Each process should be **protected against unwanted interference** by other processes, whether **accidental or intentional**.
- Thus, other processes **should not be able to reference memory locations** in a process **for reading or writing purposes without permission**.
- Because the location of a program in main memory is unpredictable, **it is impossible to check absolute addresses at compile time to assure protection.**
- Furthermore, most programming languages allow the **dynamic calculation of addresses at run time.** Hence all **memory references** generated by a process **must be checked at run time to ensure** that they refer only to the memory space allocated to that process.
- Without special arrangement, a program in one process **cannot access the data area of another process.** The processor must be able to abort such instructions at the point of execution.
- The memory protection requirement **must be satisfied by the processor** (hardware) rather than the operating system (software).

  (This is because the OS can’t anticipate all of the memory references that a program will make.)

3. Sharing

- Any protection mechanism must have the **flexibility to allow several processes to access the same portion of main memory.**

- For the number of processes executing the same program, it is advantageous **to allow each process to access the same copy of the program** rather than have its own separate copy.

- Processes that are cooperating on some task may **need to share access to the same data structure.**

- The memory management system must therefore allow **controlled access to shared areas of memory without compromising essential protection**.

4. Logical Organization

- **Main memory** in a computer system is **organized as** a linear or one-dimensional, address space, consisting of a sequence of bytes or words.
- **Secondary memory**, at its physical level, is **similarly organized**.
- Both these **do not correspond to** the way in which programs are typically constructed.
- Most **programs are organized into modules**, some of which are unmodifiable (read only, execute only) and some of which contain data that may be modified.
- The operating system and computer hardware are required to effectively **deal** with user programs and data **in the form of modules** with following advantages:
    - Modules can be **written and compiled independently**, with all references from one module to another resolved by the system at run time.
    - **Different degrees of protection** (read only, execute only) can be given to different modules.
    - It is possible to introduce mechanisms by which **modules can be shared** among processes.

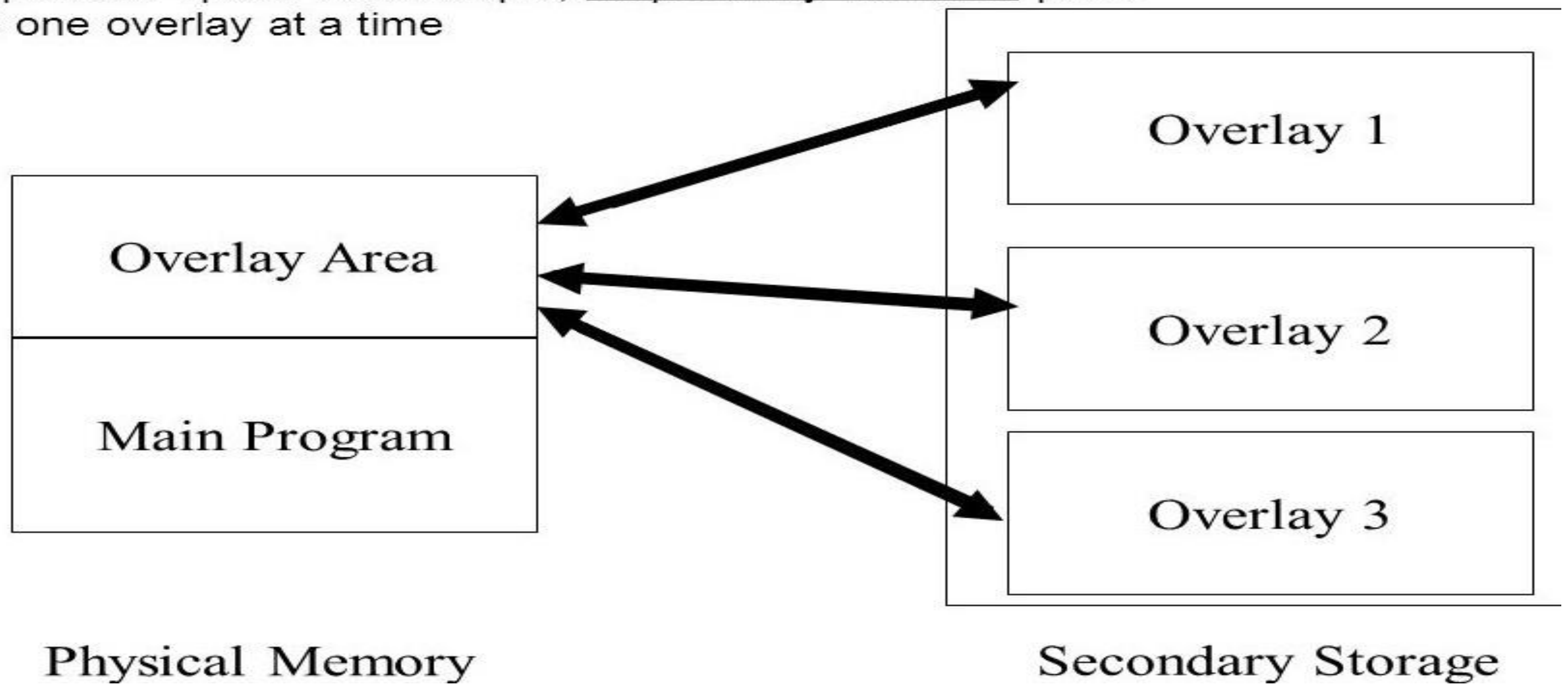
5. Physical Organization

- **Main memory** provides **fast access** at relatively high cost but its **volatile**.
- **Secondary memory is slower and cheaper** than main memory and is usually **not volatile**.
- Thus secondary memory of large capacity can be provided **for long-term storage of programs and data**, while a smaller main memory holds **programs and data currently in use.**
- The **flow of information** between main and secondary memory **is a major system concern**.
- The responsibility for this flow **could not be** assigned to the individual programmer as:
  - The main memory available for a program & its data may be insufficient; so programmer use **overlaying** , in which the **program and data** are organized in such a way that various modules can be **assigned the same region of memory**.
  - In a multiprogramming environment, the **programmer does not know** at the time of coding **how much space will be available** or where that space will be.

- The task of moving information between the two levels of memory should be a **system responsibility**. This task is the **essence of memory management**.

- **Overlaying:**

Used when process memory requirement exceeds the physical memory space
Split process space into multiple, <u>sequentially runnable</u> parts
Load one overlay at a time



- The idea behind overlays is **to only load the necessary parts of a program into memory at a given time**, freeing up memory for other tasks.
- The **unused portions of the program are kept on disk or other storage**, and are loaded into memory as needed.
- This **allows programs to be larger than the available memory**, but still run smoothly.

❖ Memory Partitioning

❑ Contiguous Allocation

- Processes placed in memory locations with consecutive memory addresses.
- It can be done in two ways:

▪ Static Partition (Fixed Partition)

fixed size partitions are used for allocation.

▪ Dynamic Partition (Variable Partition)

partitions are done as per requirement at the time of allocation.

❑ Static Partitioning (Fixed Partitioning)

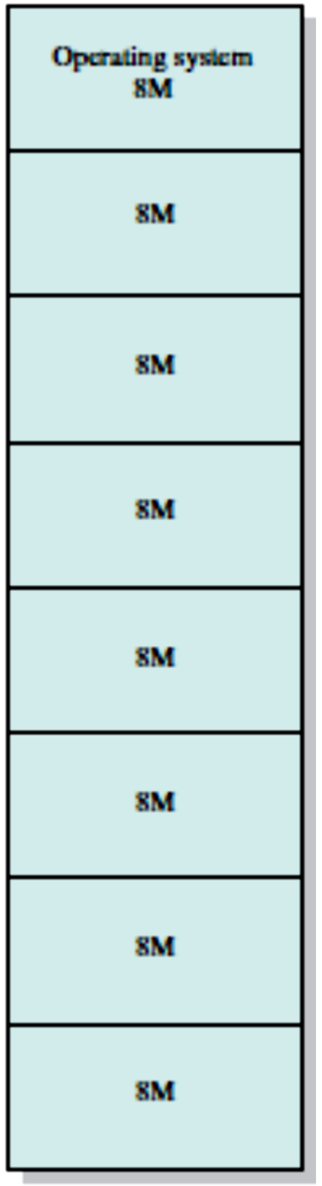
- The **OS occupies some fixed portion** of main memory and that the **rest of main memory** is available for use by multiple processes.
- **Partition Sizes:**
  i) **Equal-size partitions**   **ii) Unequal size partition**

➢ **Equal-size partitions**: any process **whose size is less than or equal to the partition size** can be loaded into any available partition.

- If all partitions are full and no process is in the Ready or Running state, OS can **swap a process** out of any of the partitions and load-in another process.

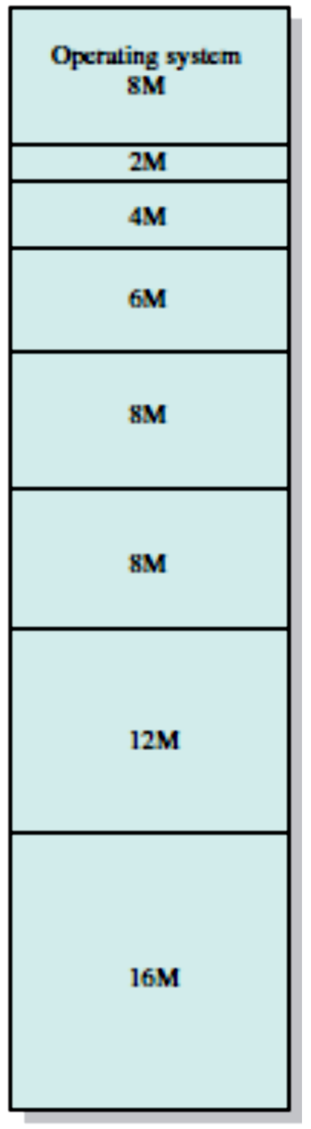
- Two difficulties:

1. **A program may be too big** to fit into a partition (solution: use of overlays).
2. **Internal fragmentation** may occur.



(a) Equal-size partitions

- **Unequal size partition** can be tried to solve both these difficulties but not solved completely.
- In this example, **programs as large as 16 Mb** can be accommodated without overlays.
- Partitions smaller than 8 Mbytes allow smaller programs to be accommodated with **less internal fragmentation**



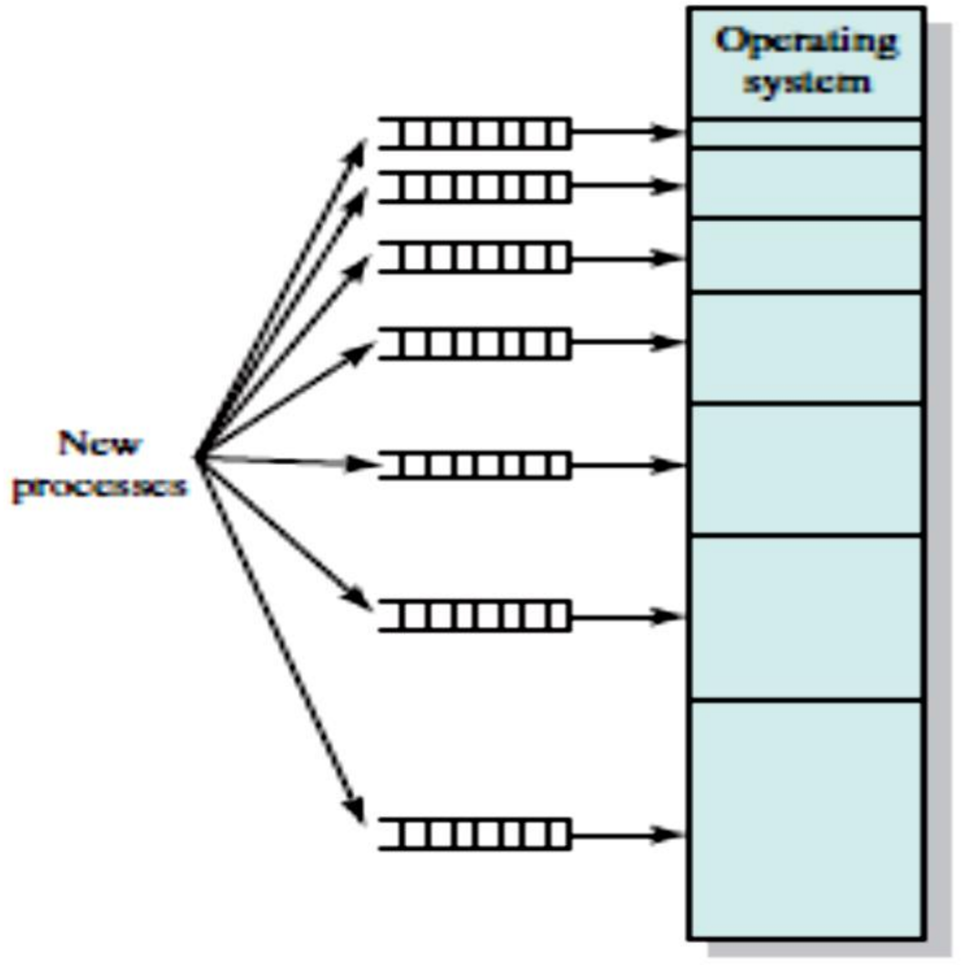
(b) Unequal-size partitions

➢**Unequal size partition:**

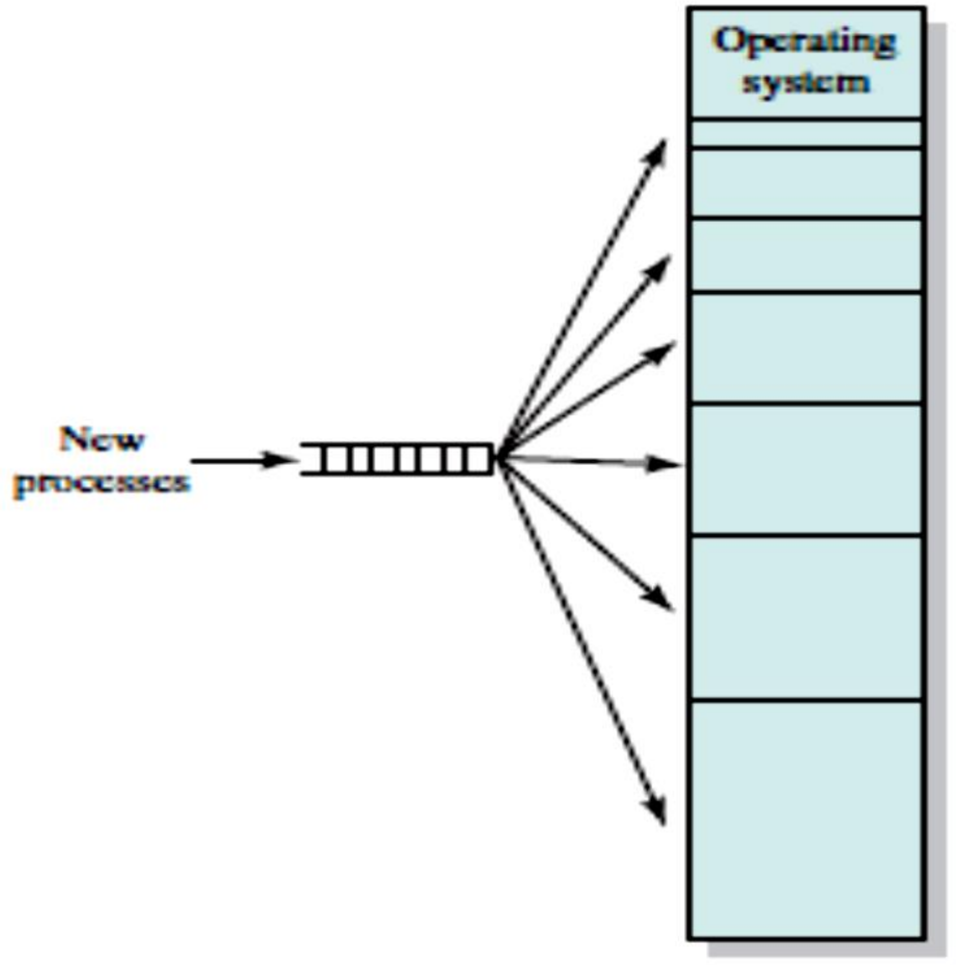
There are **two possible ways** to assign processes to partitions.

- The simplest way is **to assign each process to the smallest partition within which it will fit.**
- In this case, **a scheduling queue is needed for each partition**, to hold swapped-out processes intended for that partition.
- In this case **some partitions may remain unused.**
- So a preferable approach would be to **employ a single queue for all processes**.
- When it is time to load a process into main memory, the **smallest available partition** that will hold the process is selected. If all partitions are occupied, then a **swapping decision** must be made.

Unequal size partition continued…



(a) One process queue per partition

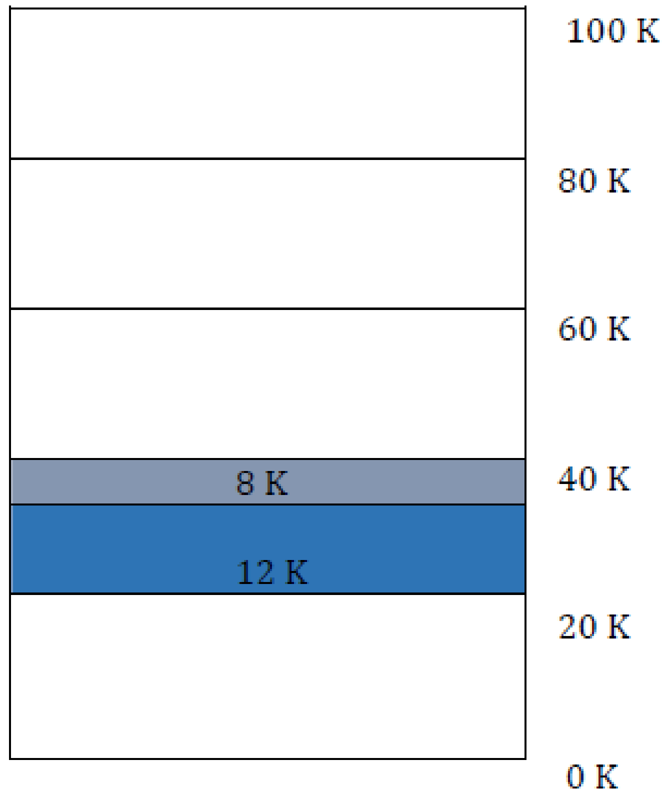


(b) Single queu

❖ **Fragmentation:** when partitioning is static then possibility of wastage of memory

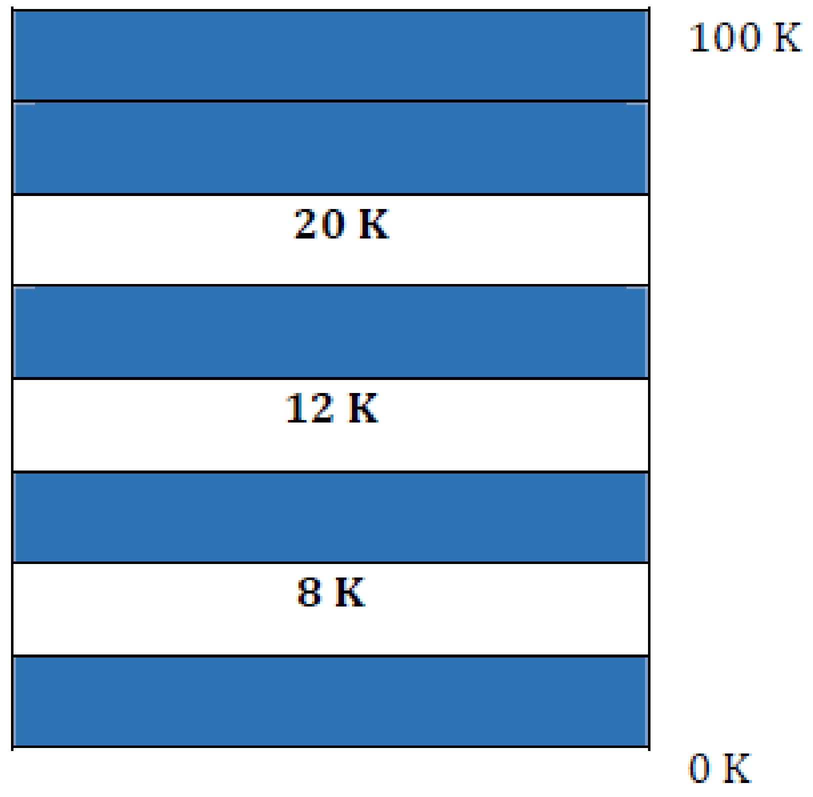
- Internal
- External

- Internal Fragmentation – If any object of smaller size than available partition is loaded in memory, this wastage of memory <u>within partition</u> is internal fragmentation

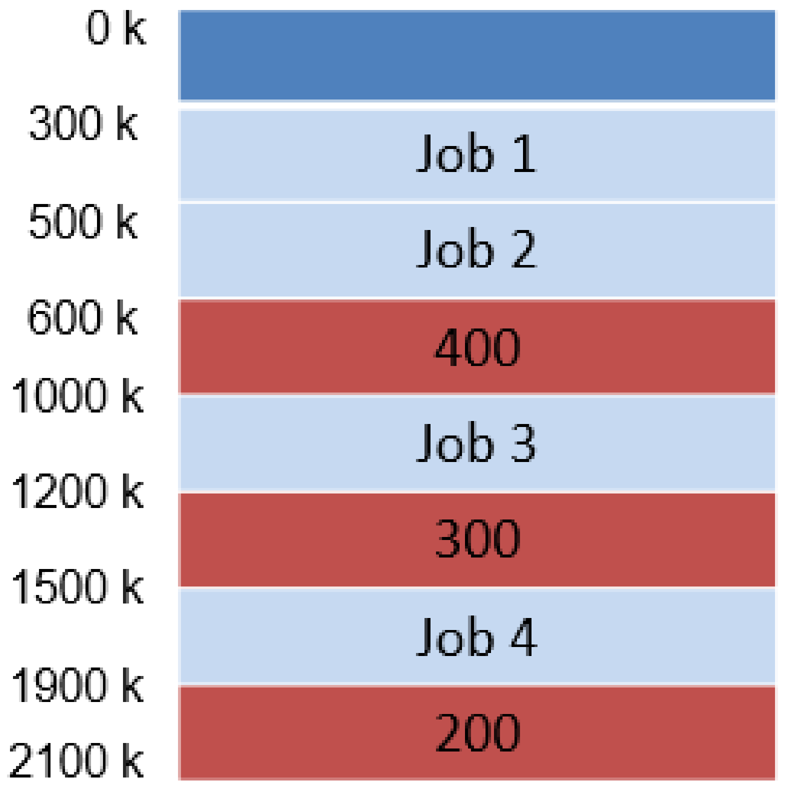


Fragmentation Continued..

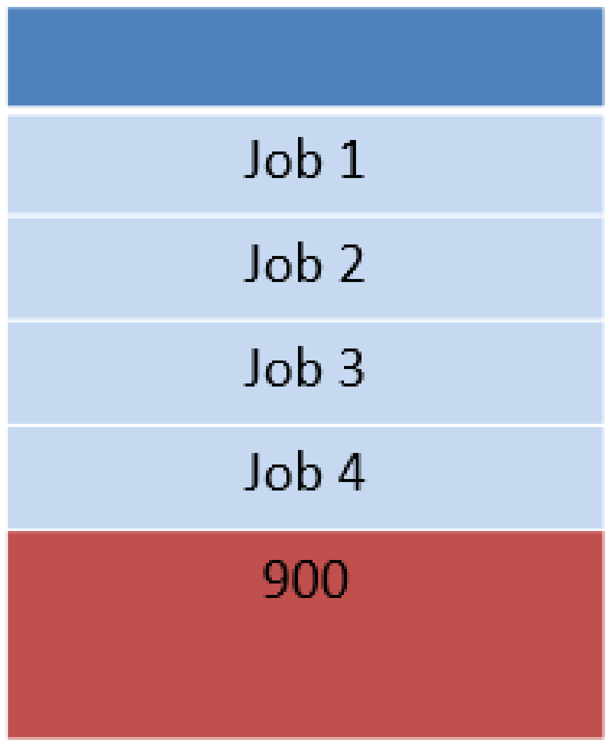
- **External Fragmentation –** total memory space exists to satisfy a request, but it is not contiguous



- **Compaction**
  - Solution to external fragmentation problem



Before Compaction



After Compaction

Relocation

Relocation is the ability to load & execute a program into an arbitrary space in memory.

- Static: Performed before or during loading program into memory by loader or linker.
- Dynamic: Mapping from virtual address space to physical address space performed at runtime during execution.

Dynamic Relocation Continued..

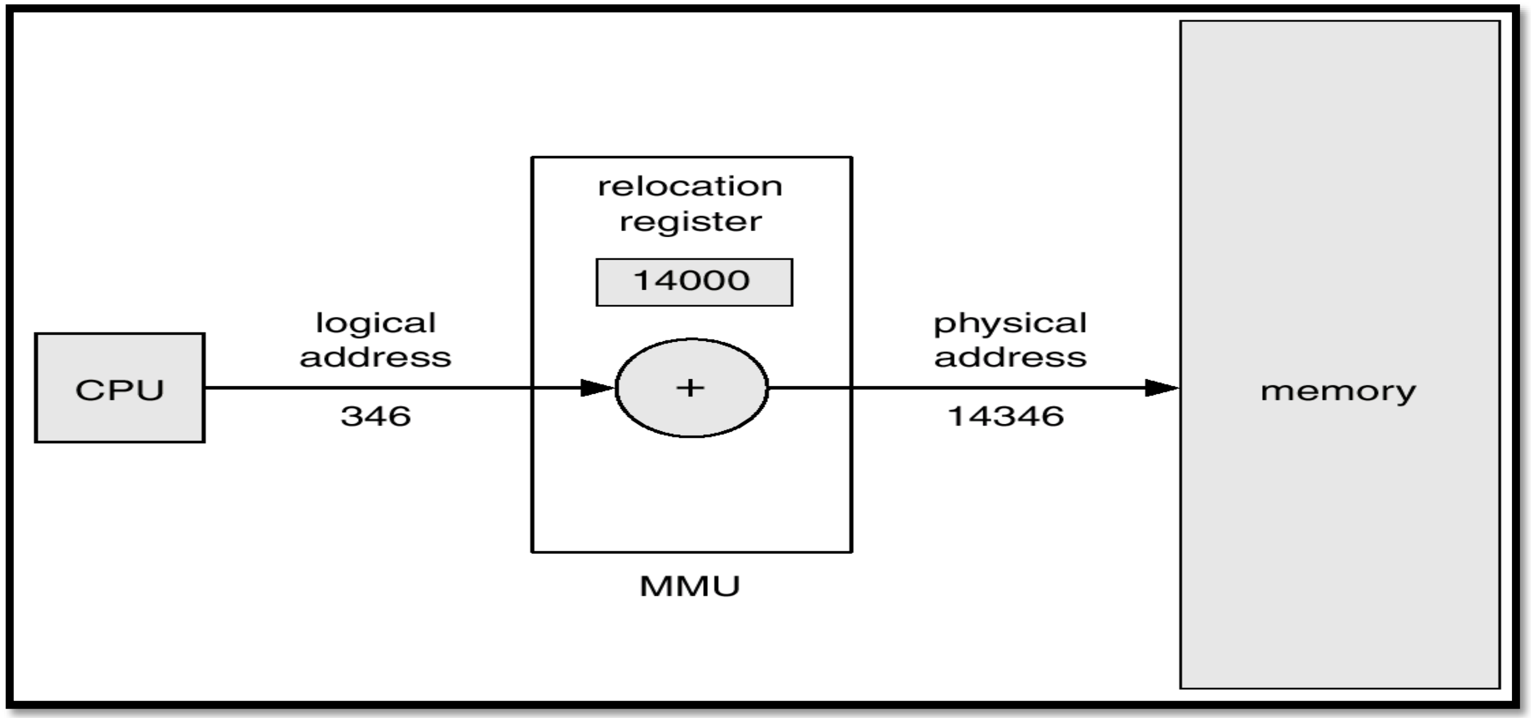


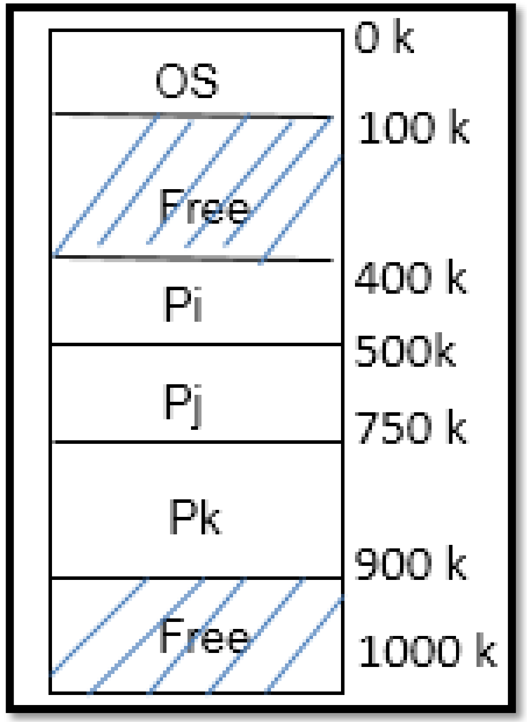
Figure: Dynamic Relocation

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

The user program deals with *logical* addresses; it never sees the *real* physical addresses

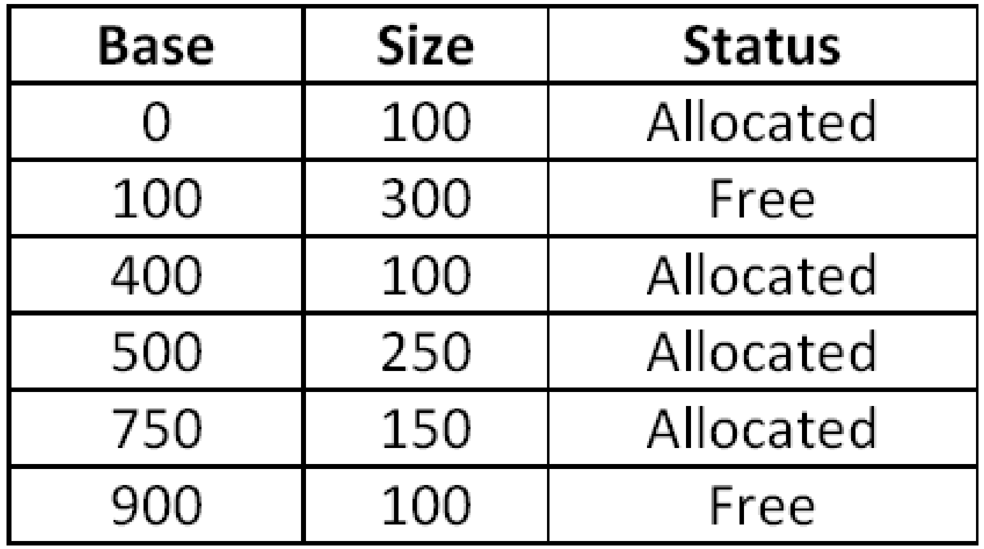
MMU- Memory Mgmt Unit

- **Partitioned Memory Allocation - Static**



Partition of memory is done before & remain fixed thereafter.

Partitioned Memory Allocation – Static continued…



Base	Size	Status
0	100	Allocated
100	300	Free
400	100	Allocated
500	250	Allocated
750	150	Allocated
900	100	Free

Table:- P.D.T.(Partition Description Table)

- Allocation of free partition can be made by :

▪ First-fit: terminates upon finding first search partition

▪ Best-fit: search for the tightest fit from PDT

▪ Partitioned Memory Allocation-Dynamic

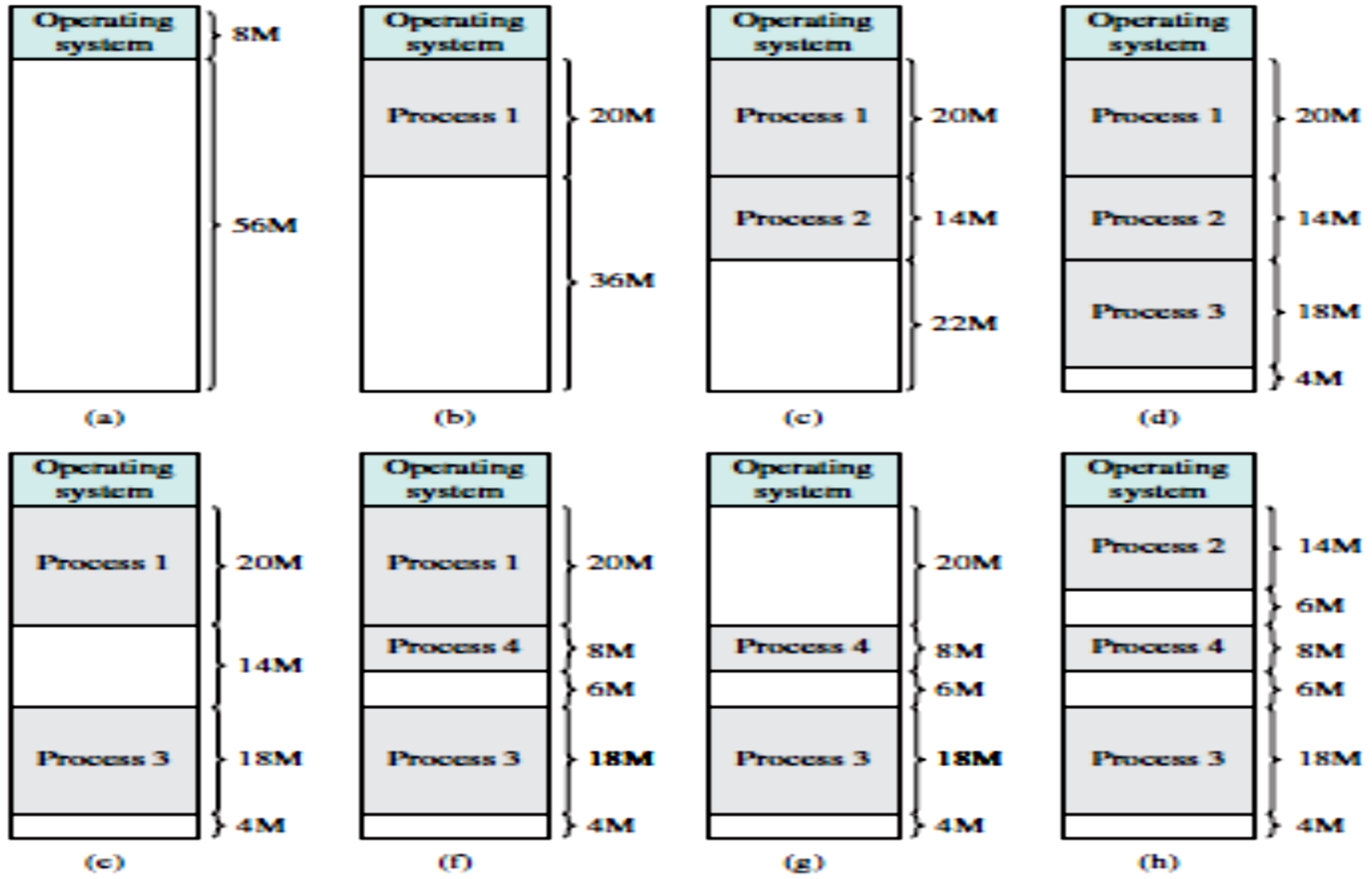


Figure: The Effect of Dynamic Partitioning

Partitioned Memory Allocation-Dynamic Continued..

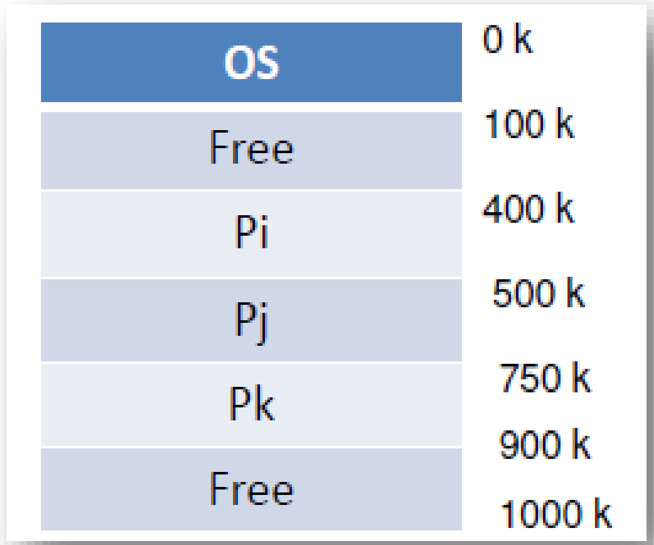
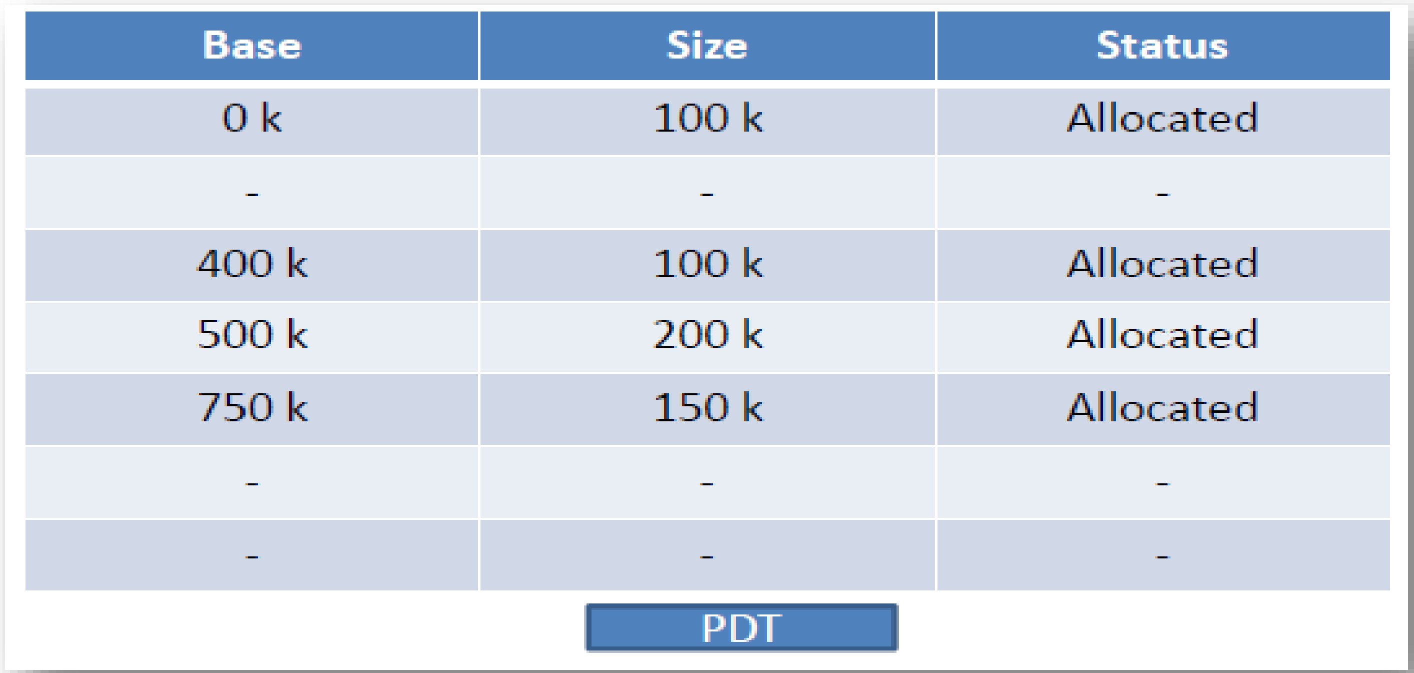


Figure : Definition of Base Address & Size



Base	Size	Status
0 k	100 k	Allocated
-	-	-
400 k	100 k	Allocated
500 k	200 k	Allocated
750 k	150 k	Allocated
-	-	-
-	-	-

PDT

Partitioned Memory Allocation-Dynamic Continued..

Location	Size	Status
100	300	Free
900	100	Free

Table: Unallocated Area Status Table (UAST)

- Memory Allocation Strategies:

In selection of free area for memory to create partitions:

1. **First Fit:** terminates upon finding **first search partition**
2. **Next Fit: keeps track of where it was** after finding suitable hole.
3. **Best Fit:** search for the **tightest fit** from PDT
4. **Worst Fit:** allocates **largest free block; opposite to best fit**.

- Consider the following sequence of holes

  10k, 4k, 20k, 18k, 4k, 9k, 12k, 15k

  How would the first-fit, best-fit, Next Fit and worst-fit algorithms place processes for 12k, 10k, 9k?

Sequence of holes: 10k, 4k, 20k, 18k, 4k, 9k, 12k, 15k

New requesting processes: 12k, 10k, 9k

First Fit:

S N	Process request	First Fit
1	12k	20k
2	10k	10k
3	9k	18k

First Fit: terminates upon finding first search partition

Sequence of holes: 10k, 4k, 20k, 18k, 4k, 9k, 12k, 15k

New requesting processes: 12k, 10k, 9k

Best Fit:

SN	Process request	Best Fit
1	12k	12k
2	10k	10k
3	9k	9k

Best Fit: search for the tightest fit from PDT

Sequence of holes: 10k, 4k, 20k, 18k, 4k, 9k, 12k, 15k

New requesting processes: 12k, 10k, 9k

Next Fit:

SN	Algorithm	Next Fit
1	12k	20k
2	10k	18k
3	9k	9k

Next Fit: keeps track of where it was after finding suitable hole

Sequence of holes: 10k, 4k, 20k, 18k, 4k, 9k, 12k, 15k

New requesting processes: 12k, 10k, 9k

Worst Fit:

SN	Algorithm	Worst Fit
1	12k	20k
2	10k	18k
3	9k	15k

Worst Fit: Allocates largest free block. Opposite to best fit.

10k, 4k, 20k, 18k, 4k, 9k, 12k, 15k

SN	Process Request	First Fit	Best Fit	Next Fit	Worst Fit
1	12k	20k	12k	20k	20k
2	10k	10k	10k	18k	18k
3	9k	18k	9k	9k	15k

- **Non Contiguous Memory Allocation**

Memory is allocated such that part of single process may be placed in noncontiguous area of physical memory.

It can be done by :

- Paging
- Segmentation

- **Paging:**

- Paging permits program memory **to be non contiguous** allowing program to be allocated in physical memory **wherever it is possible**.
- Thus paging can **solve external fragmentation problem**.

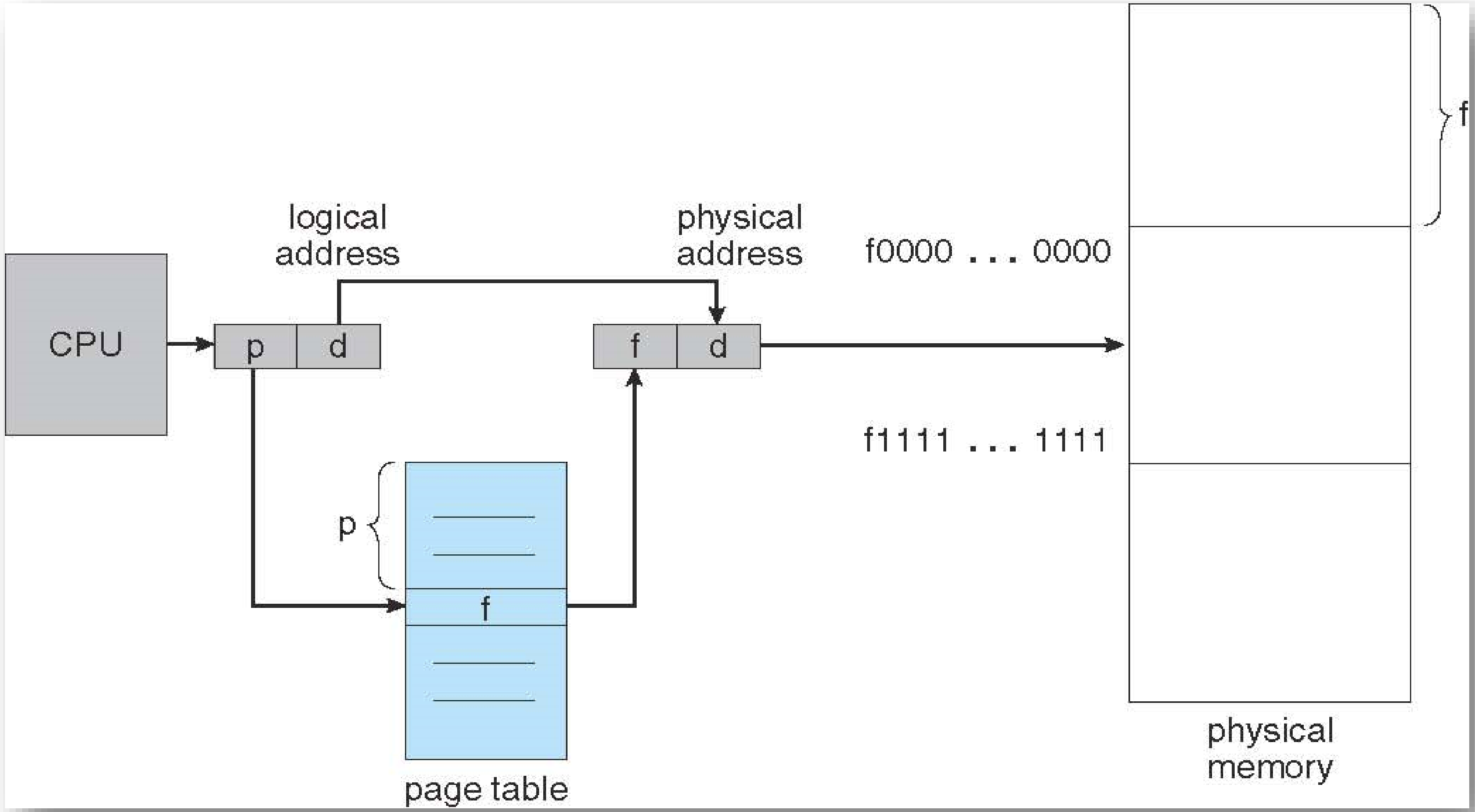
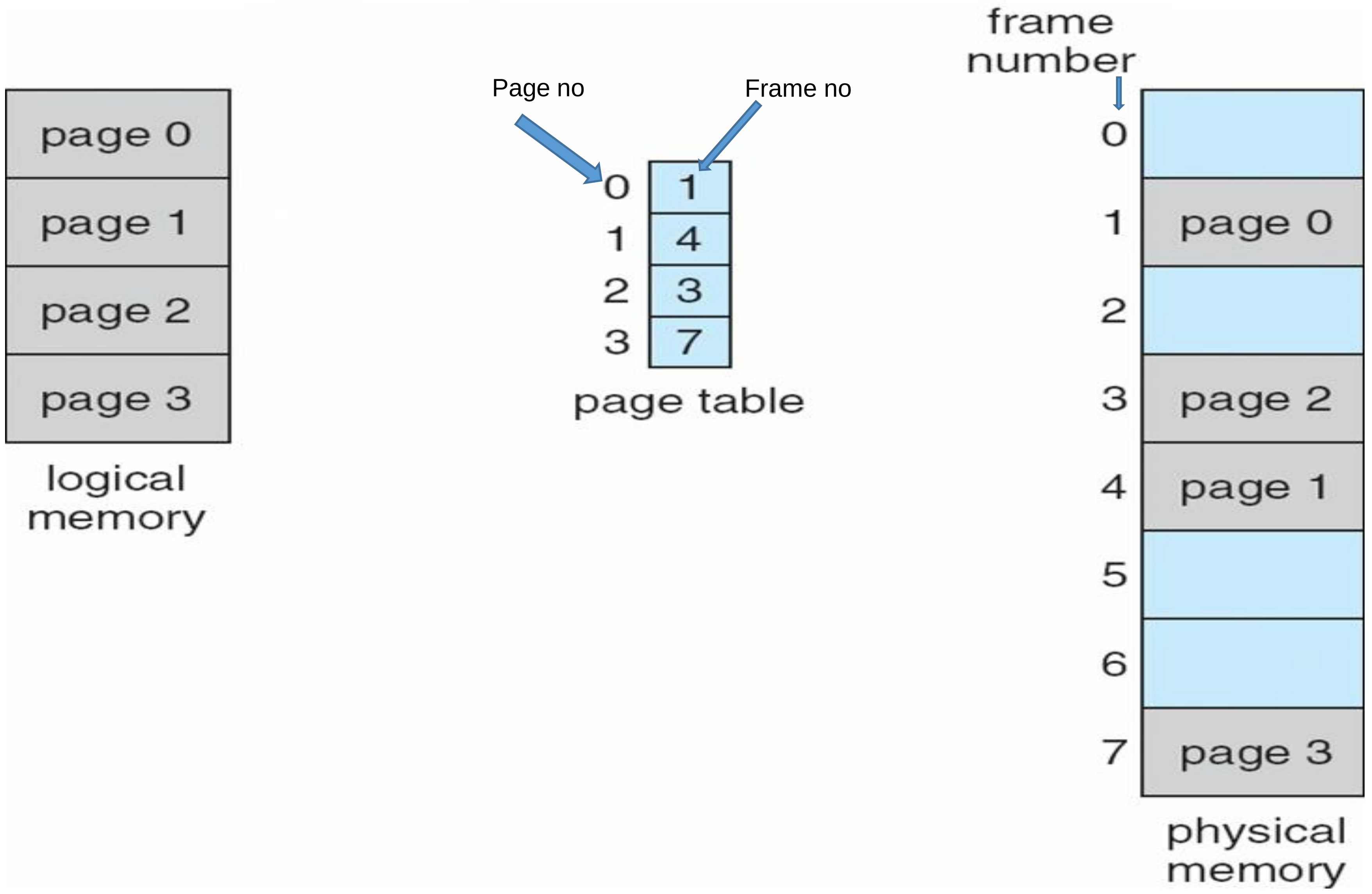


Figure: Paging Hardware (Paging)

Paging Continued….

Paging Example



Paging Continued….

• **Paging**

- Logical address space of a process can be noncontiguous; process is allocated physical memory whenever available.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, between 512 bytes and 1 GB per page depending on the computer architecture.).
- Divide logical memory into blocks of same size called **pages**.
- Keeps **track of all free frames**.
- To run a **program of size *n* pages**, need to **find *n* free frames** and load program.
- Sets up a page table to **translate logical to physical addresses**.
- Internal fragmentation problem can occur.
- #

• Paging Hardware With TLB (Translation Look Aside Buffer)

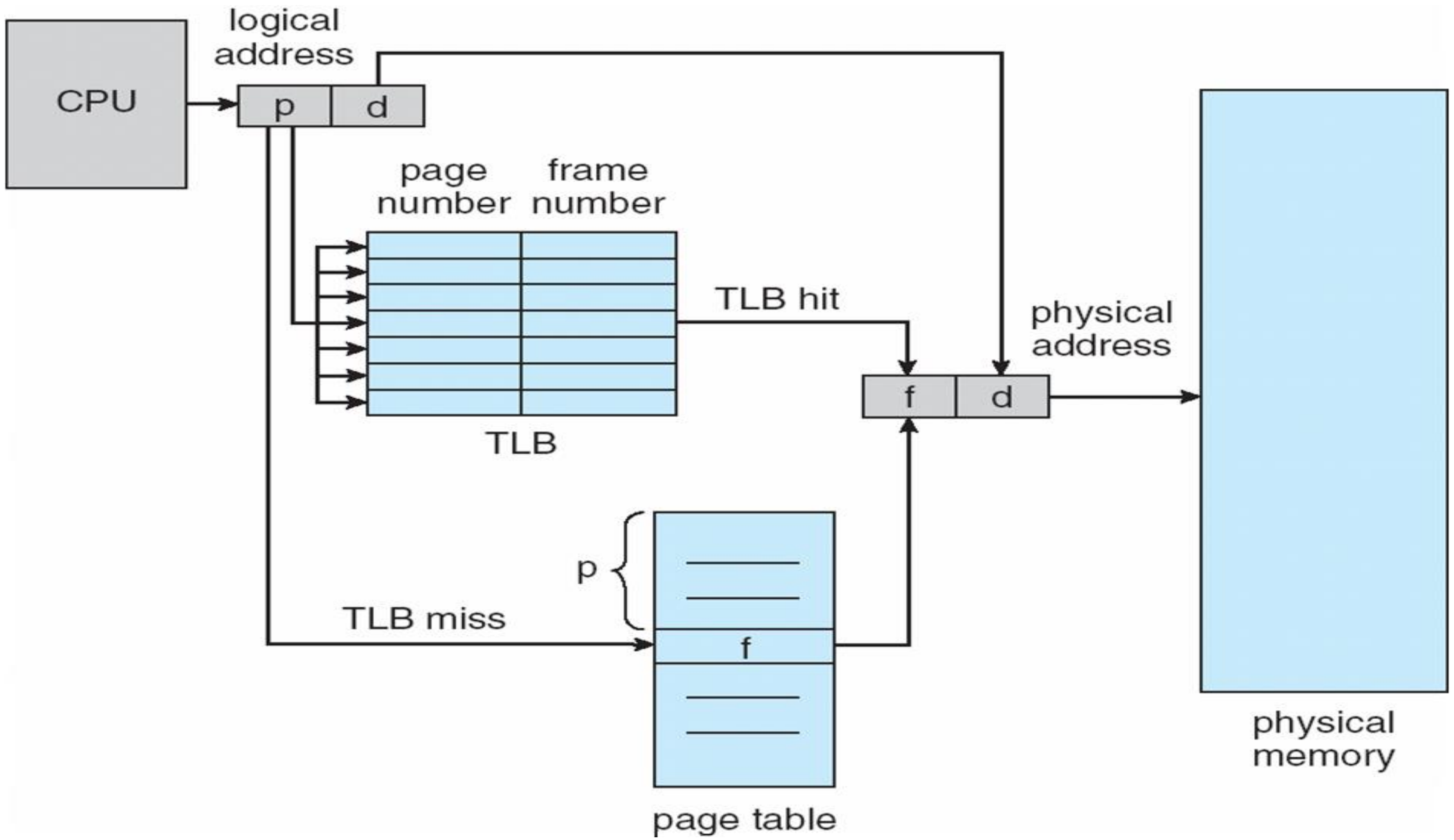
- The page table is kept in main memory and a **Page Table Base Register (PTBR)** points to the page table for faster access.
- In paging, **two memory accesses** are needed to access a byte: one for the page-table entry, one for the byte; which causes the access slowed by a factor of 2.
- Page table has very **large number of entries** (nearly 1 million entries)
- **Solution is** to use a special, small, fast lookup **hardware cache i.e. TLB**
- **TLB is associate, high speed memory**

Paging Hardware With TLB Continued….

- Each entry in the TLB consists of **two parts: a key (or tag) and a value.**
- When a **logical address is generated** by the CPU, its **page number is presented to the TLB.**
- If the page number is found (**TLB hit**), its frame number is immediately available and is used to access memory.
- If the page number is not in the TLB (**TLB miss**), a **memory reference** to the page table must be made.
- When the frame number is obtained, we can use it to access memory
- We **add the page number and frame number to the TLB**, so that they will be found **quickly on the next reference**.
- If the TLB is already full of entries, the OS must select one for **replacement**.

Paging Hardware With TLB Continued….

Paging Hardware With TLB (translation look aside buffer)



Next: Paging Protection…

Protection

- **Memory protection** in a paged environment is accomplished by **protection bits associated with each frame.**
- One additional bit is generally attached to each entry in the page table: a **valid-invalid** bit
- When this bit is set to **“valid”,** the associated page is in the process's logical address space and is thus a **legal (or valid) page**.
- When the bit is set to **“invalid”,** the page is **not in the process's logical address space**.
- **Illegal addresses are trapped** by use of the valid-invalid bit.
- **The OS sets this bit for each page to allow or disallow access to the page.**

Protection Continued…

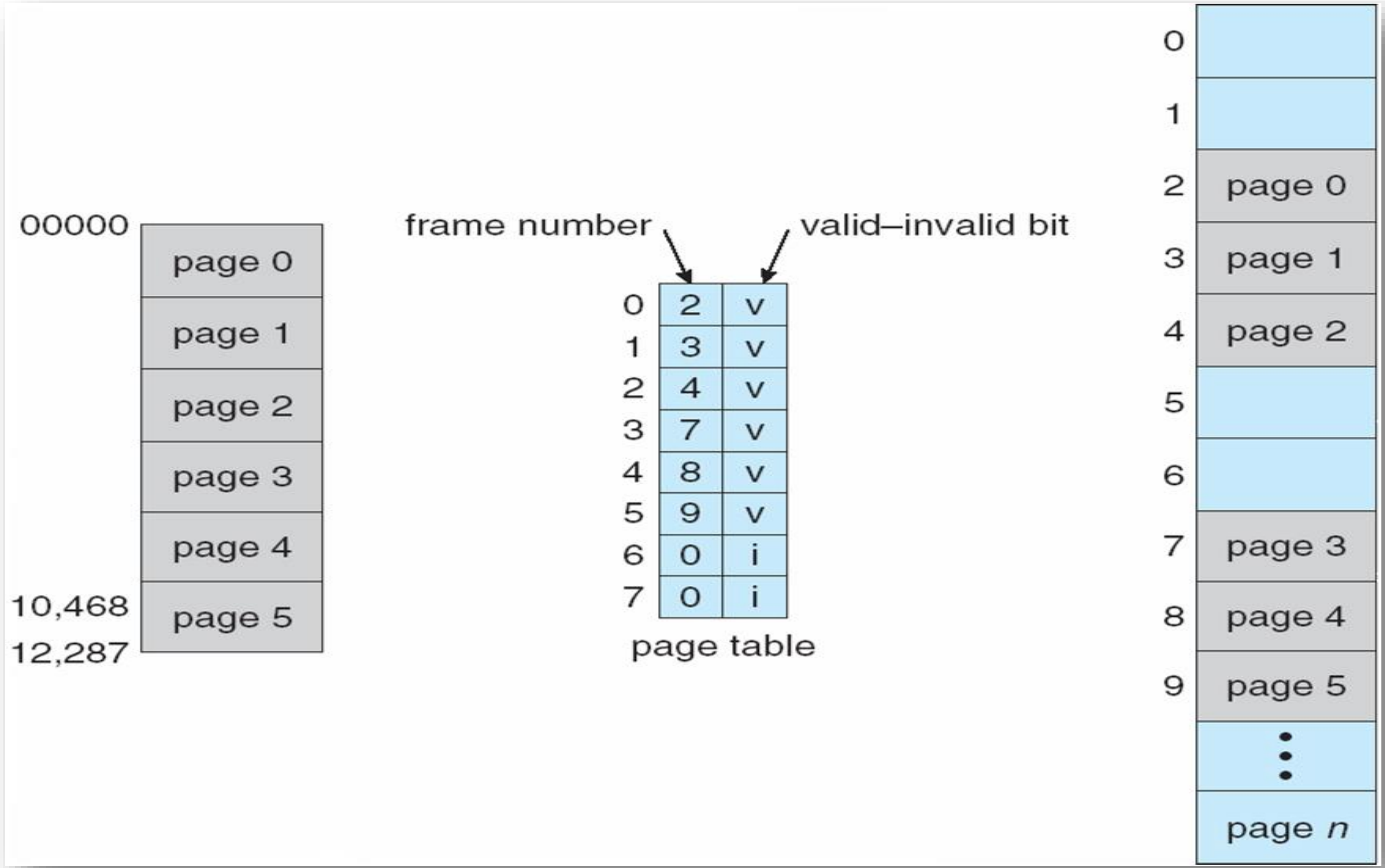


Figure: Valid (v) and Invalid (i) Bit in Page Table

- **Sharing**

- An **advantage of paging** is the possibility of ***sharing* common code**.
- **A Reentrant code is non-self-modifying code; it never changes during execution**
- If the code is **reentrant code** (or **pure code),** it can be shared.
- Thus, two or more processes can **execute the same code at the same time**.
- Each process has its **own copy** of registers and data storage to hold the data for the process's execution.
- The **data** for two different processes will be **different**.
- The shared pages example is as shown in Figure **below.**

Sharing Continued…

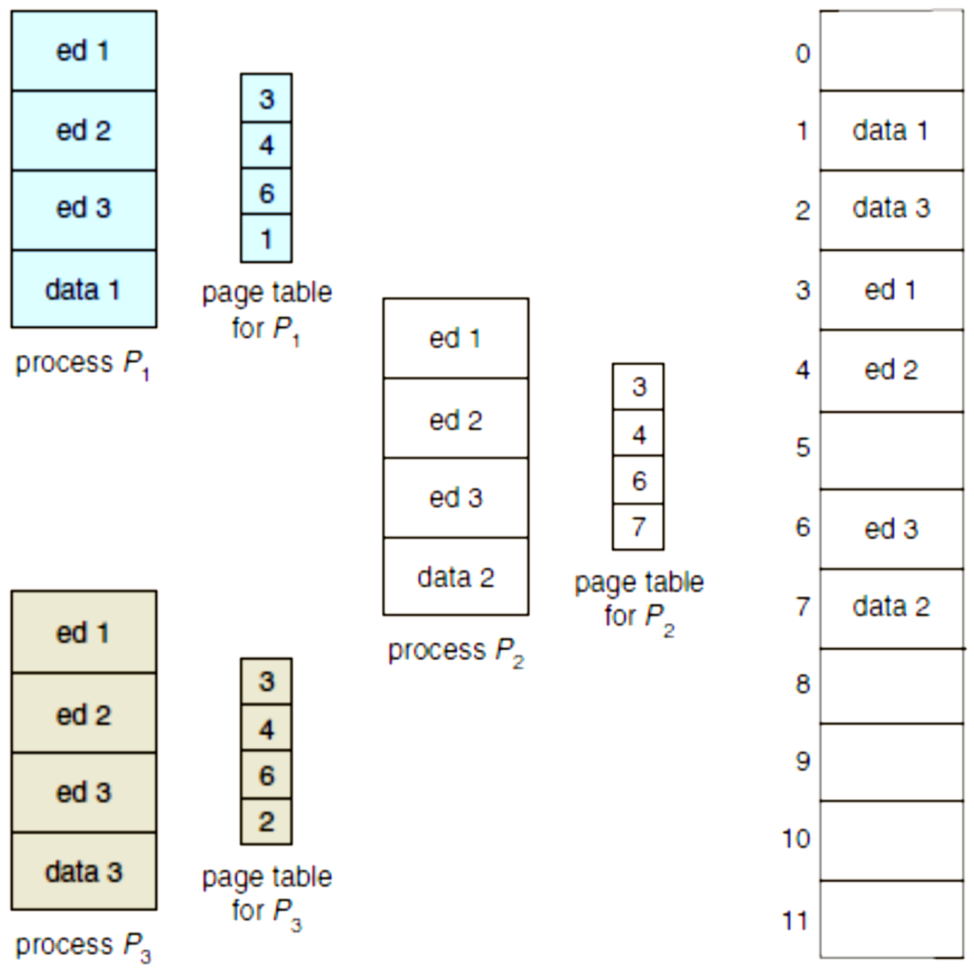


Figure: Shared Pages Example

A system that supports **40 users**, each of whom executes **a text editor**,

Code of Text editor: 150k
Data space: 50k
For 40 users: ?
= (40*50) + (150*40)
= 2000 + 6000
= 8000 k

If the code is reentrant code
Text editor: 150k
Data space: 50k
For 40 users: ?
= (40*50) + (150)
= 2000 + 150
= 2150 k

- Consider a system that supports 40 users, each of whom executes a text editor.
- If the text editor consists of 150 KB of code and 50 KB of data space, we need 8,000 KB to support the 40 users.
- **For Reentrant code**, only **one copy of the editor** need be kept in physical memory.
- Each user’s page table maps onto the same physical copy of the editor, but data pages are mapped onto different frames.
- 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.

- **Disadvantages of Paging**

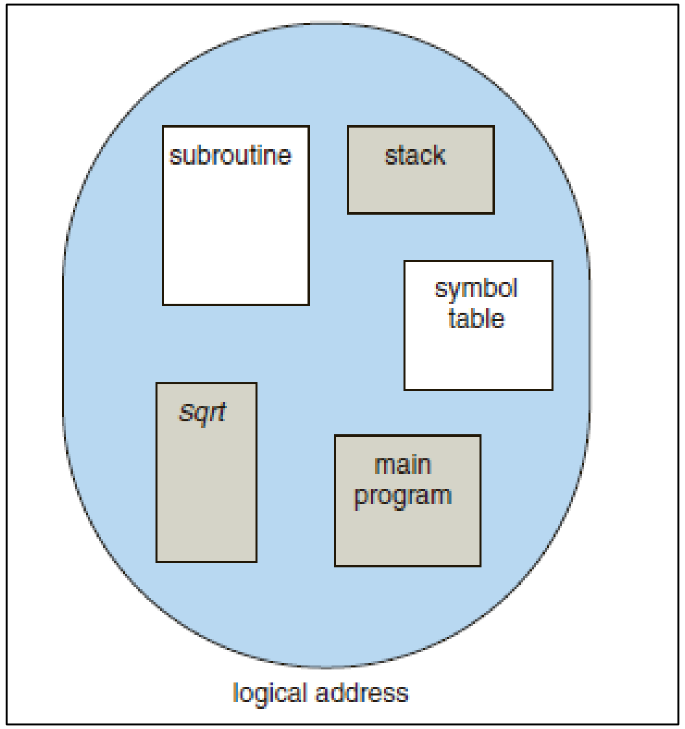
- Internal fragmentation
- Page Table may consume more memory

❖ Segmentation

- Memory-management scheme that **supports user view of memory**.
- A program is a **collection of segments**.
- A segment is a logical unit such as:

  main program,
  procedure,
  function,
  method,
  object,
  local variables, global variables,
  common block,
  stack,
  symbol table, arrays

- **A segment can have arbitrary size <u>unlike</u> paging.**
- Each process has its own segment table.



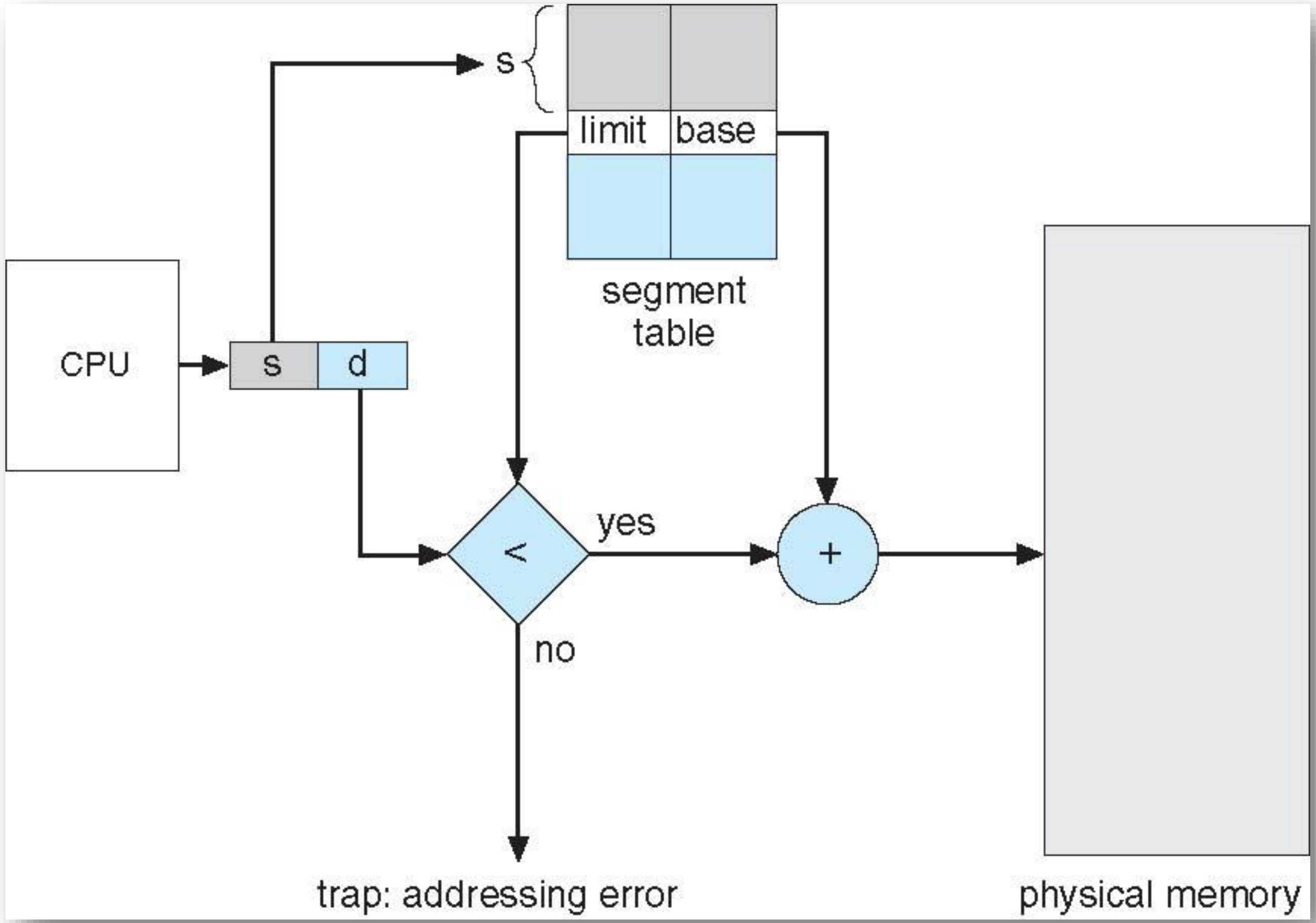


Figure: Segmentation Hardware

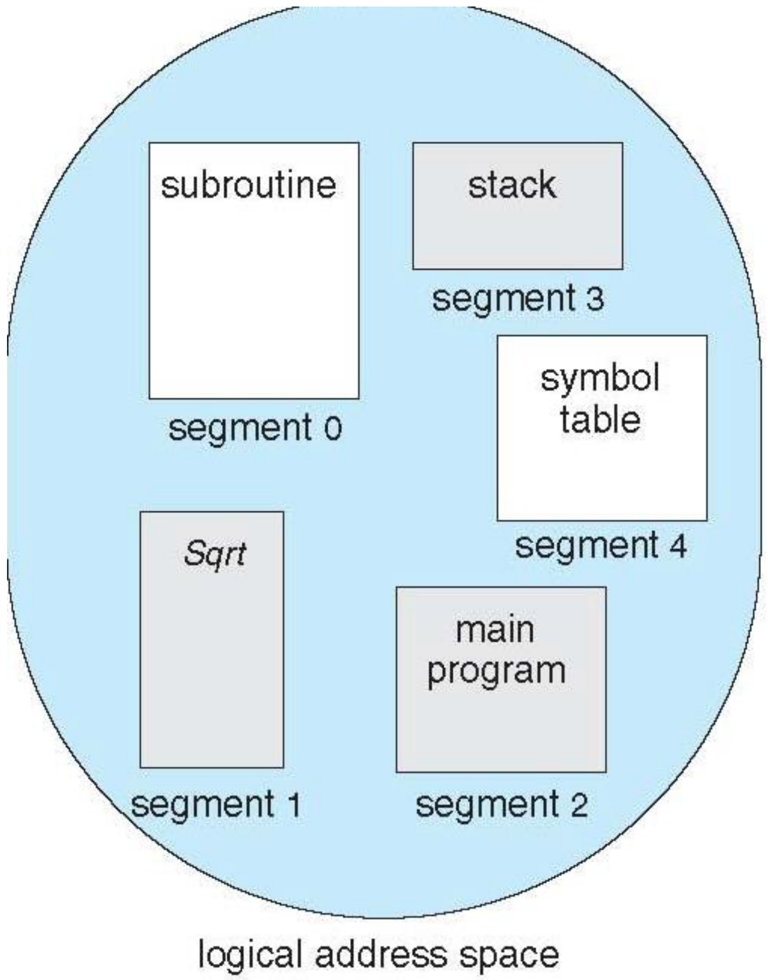
- Each Process has its segment table.
- Each entry in segment table has segment base & limit.
- The segment base contains starting physical address where segment resides in main memory.
- The segment limit specifies length of segment.
- The logical address generated by CPU has two parts:

i) Segment number (s)

ii) Offset into that segment (d)

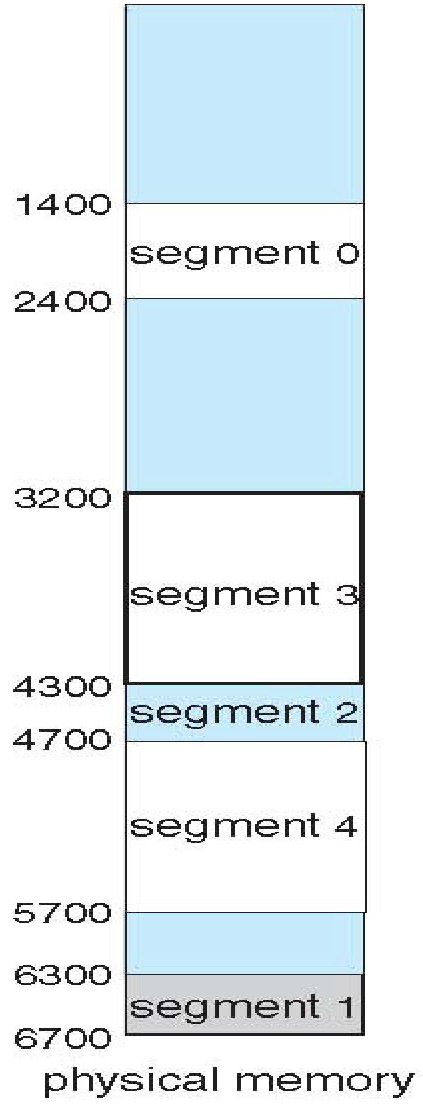
- The segment no is used as index into segment table.
- The offset d must be between 0 and segment limit.

  If it is not within limit, we trap to OS (logical addressing, attempts beyond end of segment).
- If this offset is legal, it is added to segment base to produce physical address of required bytes.



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

segment table



Example of Segmentation

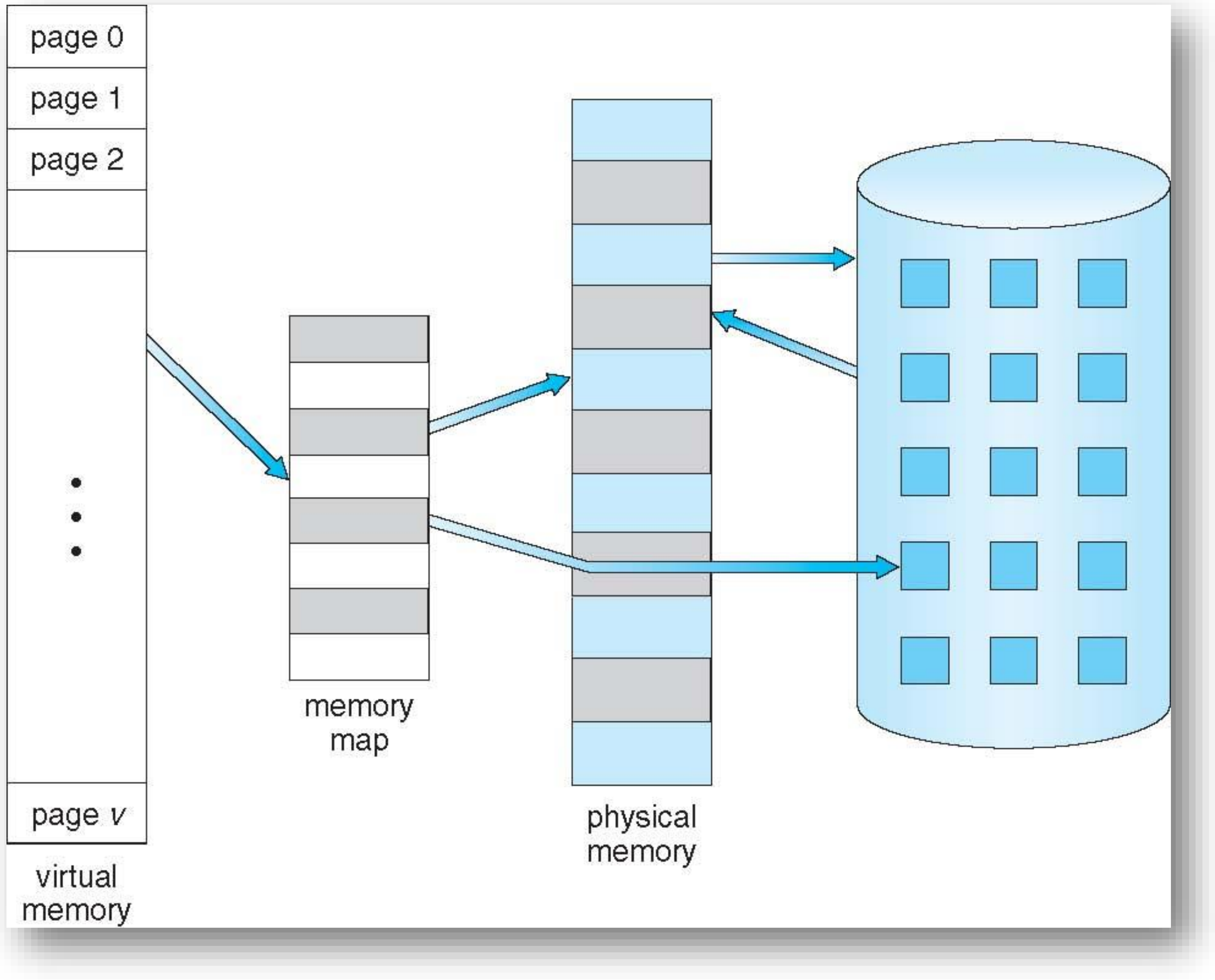
❖ Virtual Memory Management

- Virtual memory – It is a technique that allows the execution of processes that are not completely in memory.
  - Only **part of the program** needs to be in memory for execution.
  - Logical address space can therefore be **much larger** than physical address space.
  - Allows **address space to be shared** by several processes through page sharing.
  - Frees programmer from concerns of main memory storage.
  - Increased CPU utilization and throughput.

- Virtual memory can be implemented via:

- **Demand paging**
- Demand segmentation

- **Virtual Memory That is Larger Than Physical Memory**
  -



- **Demand Paging**
  (Paging with respect to virtual memory)

Transfer of a Paged Memory to Contiguous Disk Space

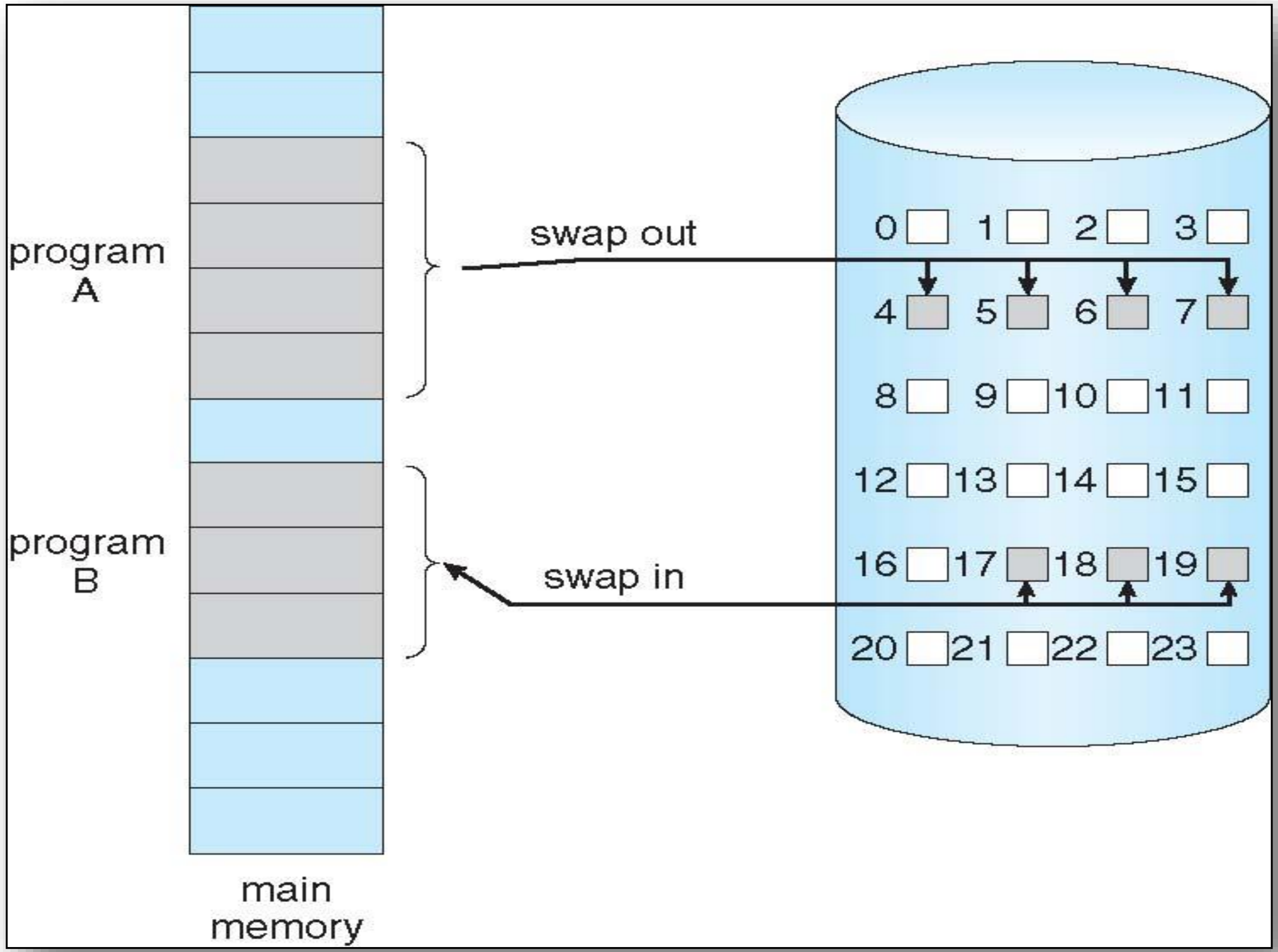


Figure: Demand Paging

Demand Paging Continued…

- Paging system with swapping.
- Process resides on secondary storage like disc; when required its swapped in.
- Lazy Swapper (pager) which brings only required pages in the memory .
- Avoids reading of not needed pages.
- Pager guesses the page which will be used before process is swapped out.
- Decrease swapping time & amount of memory needed.
- Needs H/W support to distinguish between pages (on disc & in memory).
- Page is needed $\Rightarrow$ reference to it
  - Bit set to valid - Page is legal and in memory
  - Bit set to invalid - Page is either not valid or valid but currently on disc
- **Page Fault**:
  - ➢ if the process tries to access a page that was not brought into memory?
  - ➢ Access to a page marked invalid causes a **page fault**.

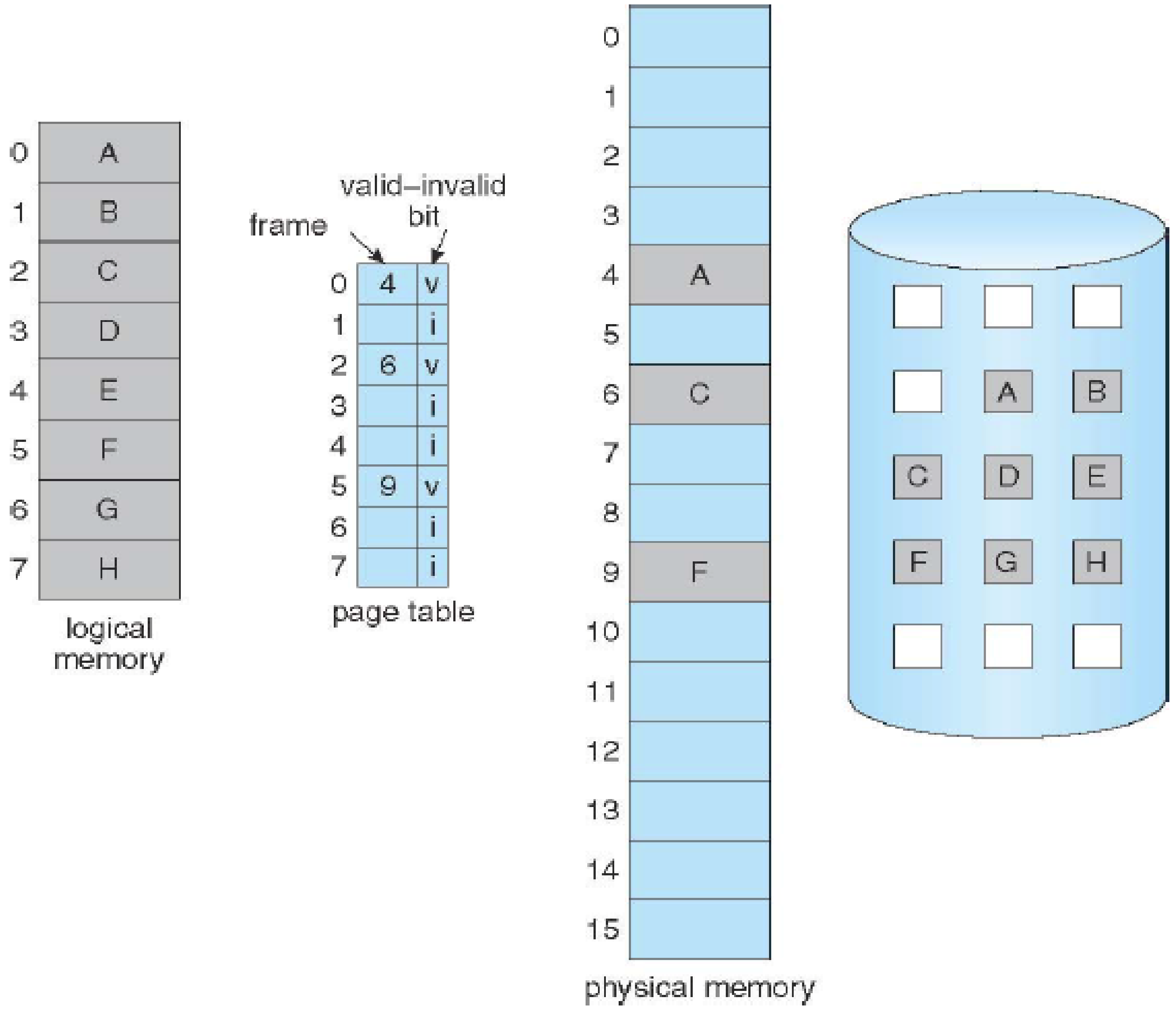


Figure: Page table when some pages are not in main memory

- **Steps in Handling a Page Fault**

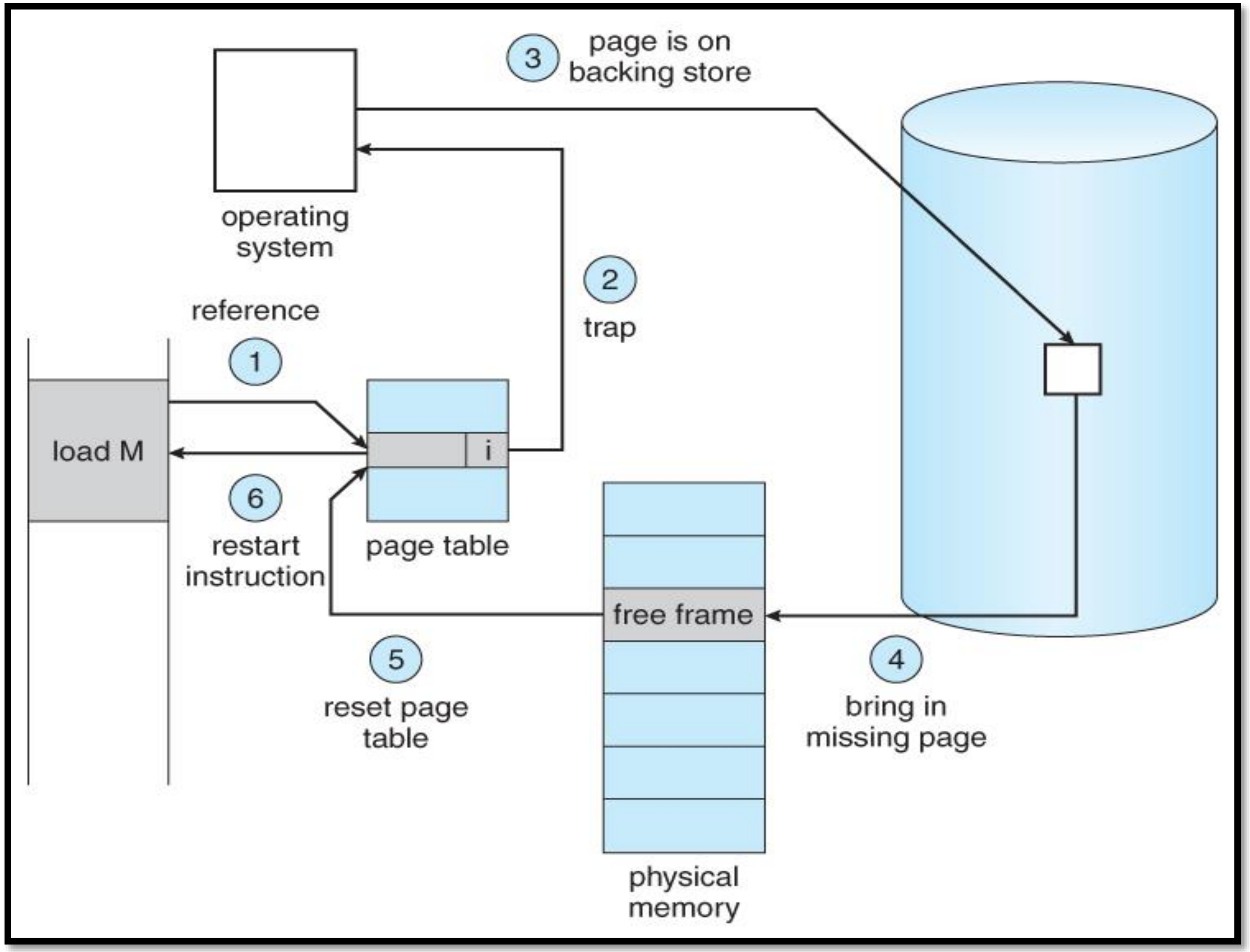


Figure: Procedure to handle page fault

1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk.
5. When the disk read is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning.

❖ Page Replacement Algorithms

Algorithms to implement demand paging & recover from page Faults.

- First-In-First-Out (FIFO) Algorithm
- Least Recently Used (LRU) Algorithm
- Optimal Page Replacement
- Least Frequently Used (LFU) Algorithm

- **First in First Out:**

▪ New pages are replaced with the oldest one.

▪ Consider the reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

- For **3 pages per frame**

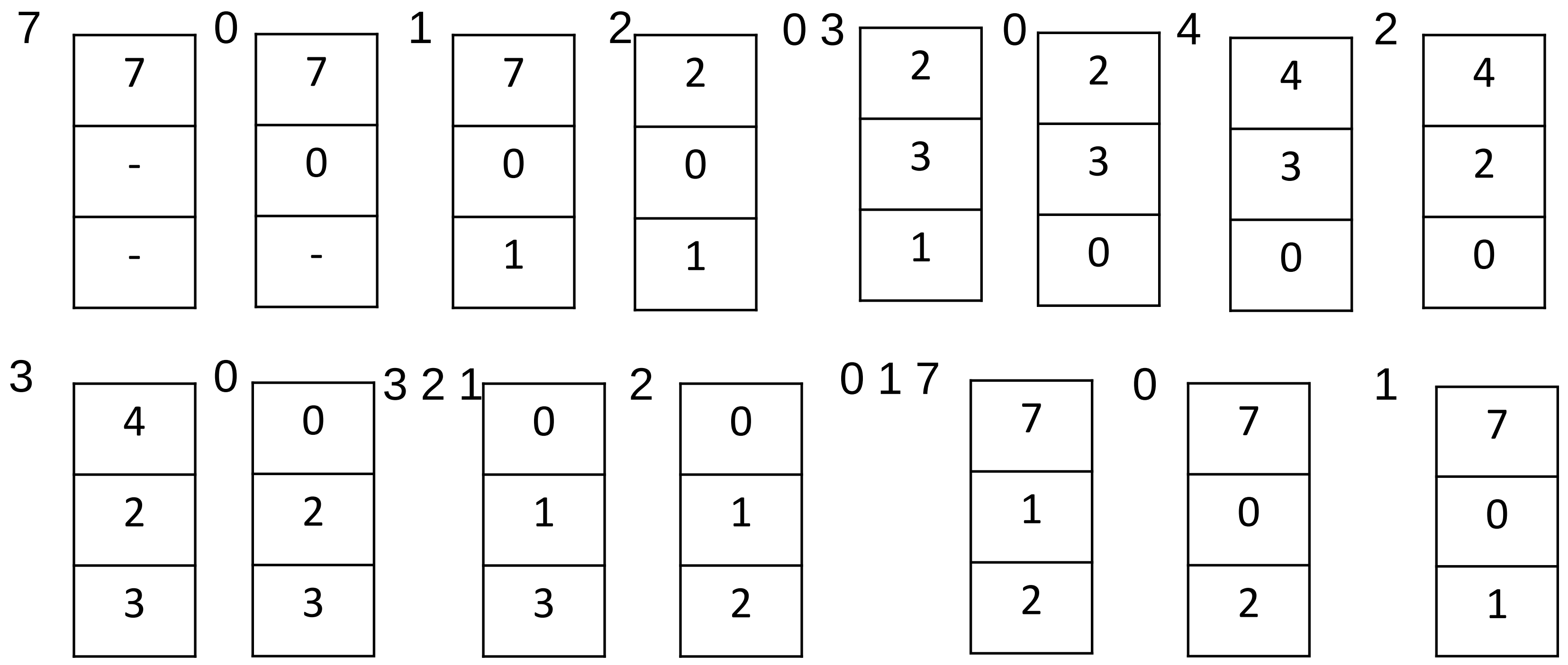
New pages are replaced with the oldest one

Reference String

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

First-In-First-Out (FIFO) Algorithm

For 3 pages per frame



Total No of Page Faults: 15

Hit Ratio: No of Page Hits/String Size: 5/20

Miss Ratio: No of Miss Hits/String Size: 15/20

- **Least Recently Used:**

▪ It is associated with each page used and the time of its last use.

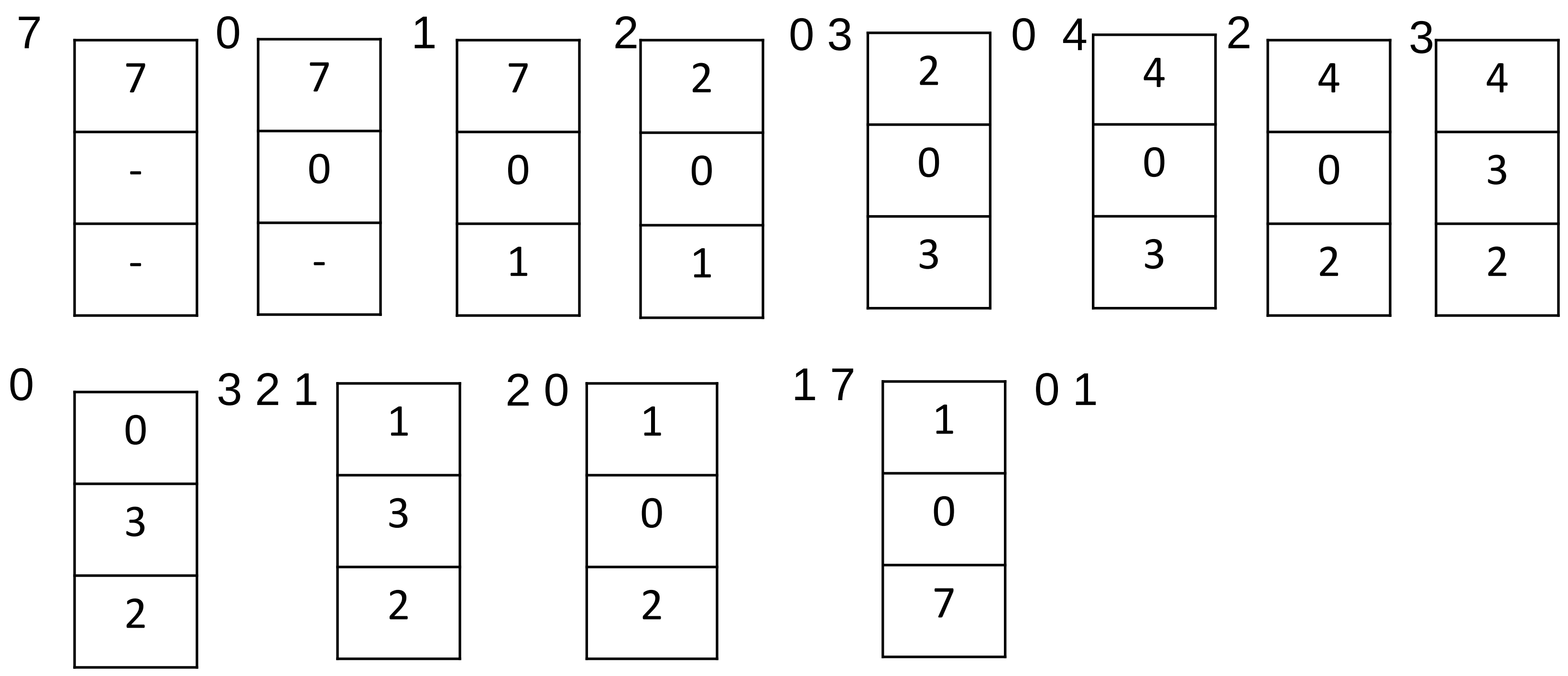
▪ New page is replaced with the page which has not been used for longest period of time.

LRU Page Replacement

Page which has not been used for longest period of time is replaced

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

For 3 pages per frame



Total No of Page Faults: 12
Hit Ratio: No of Page Hits/String Size: 8/20
Miss Ratio: No of Miss Hits/String Size: 12/20

- **Optimal Page Replacement**

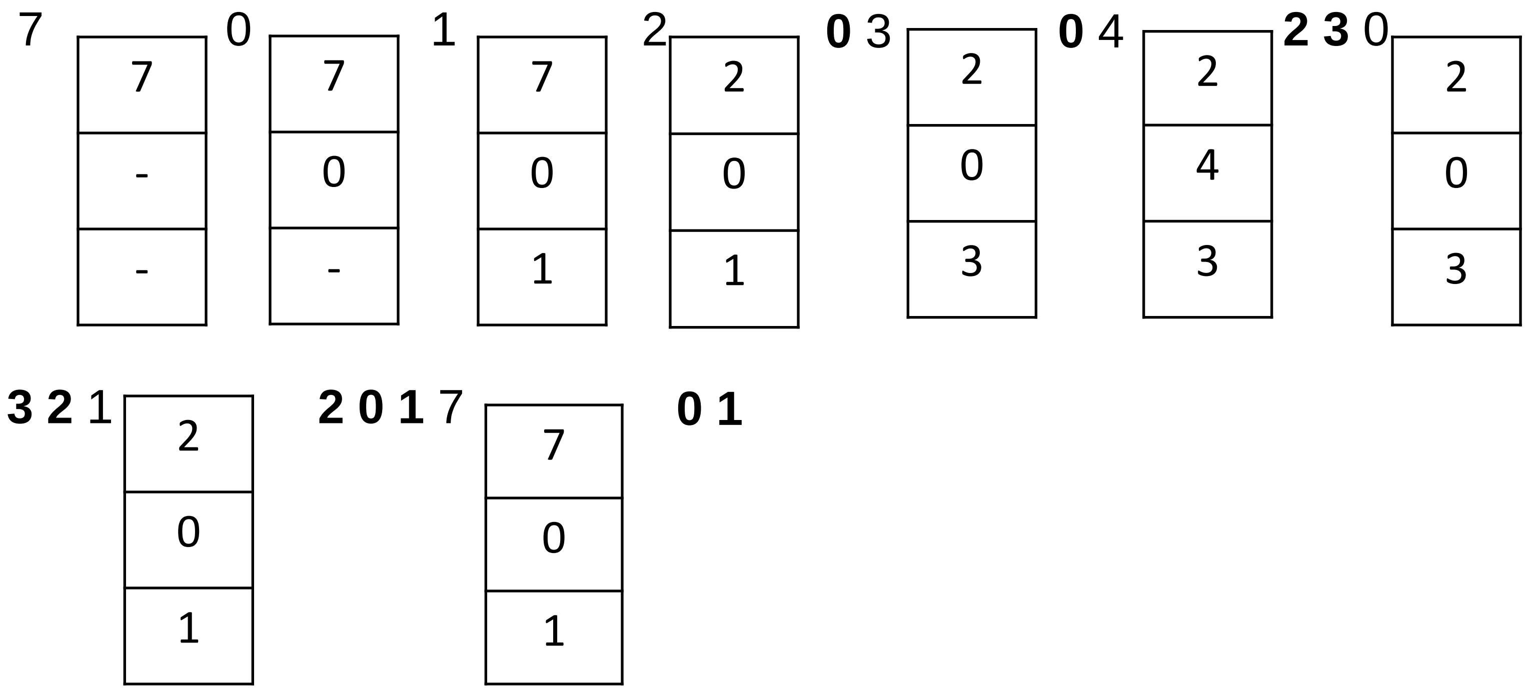
- It replaces the page that will be used not immediately but in distant future.
- i.e. it replaces the page that will not be used for longest period of time.

Page that **will not be used for longest period of time is replaced**

Optimum Page Replacement

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

For 3 pages per frame



Total No of Page Faults: 9
Hit Ratio: No of Page Hits/String Size: 11/20
Miss Ratio: No of Miss Hits/String Size: 9/20

- **Least Frequently Used (LFU) (Not Frequently Used)**

- Replace that block in the set that has experienced the fewest references.
- LFU could be implemented by associating a counter with each block.
- When a block is brought in, it is assigned a count of 1; with each reference to the block, its count is incremented by 1.
- At required replacement, the block with the smallest count is selected.

Replace that block in the set that has **experienced the fewest references**

LFU Page Replacement

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

For 3 pages per frame

7	0	1	2	0 3	0 4	2
7	7	7	2	2	4	4
-	0	0	0	0	0	0
-	-	1	1	3	3	2

3	0 3 2 1	2	0 1	7	0 1
3	3	3	3	3	3
0	0	0	0	0	0
2	1	2	1	7	1

Block	Count
7	
0	
1	
2	
3	
4	

Total No of Page Faults: 13

Hit Ratio: No of Page Hits/String Size: 7/20

Miss Ratio: No of Miss Hits/String Size: 13/20

Consider following page reference string:

0, 3, 2, 1, 4, 5, 0, 3, 1, 2, 1, 3, 4, 0, 2, 3, 4, 5, 1, 2

How many page faults would occur for the following page replacement algorithms if they are allocated 3 initial empty frames?

i) LRU ii) Optimal.

0, 3, 2, 1, 4, 5, 0, 3, 1, 2, 1, 3, 4, 0, 2, 3, 4, 5, 1, 2

i) LRU

Page Fault= 18

Hit Ratio: No of Page Hits/String Size: 2/20

Miss Ratio: No of Miss Hits/String Size: 18/20

i) LRU

0	3	2	1	4	5	0	3	1	2	1	3	4	0	2	3	4	5	1	2
0	0	0	1	1	1	0	0	0	2	2	2	4	4	4	3	3	3	1	1
	3	3	3	4	4	4	3	3	3	3	3	3	3	2	2	2	5	5	5
		2	2	2	5	5	5	1	1	1	1	1	0	0	0	4	4	4	2
F	F	F	F	F	F	F	F	F	F	H	H	F	F	F	F	F	F	F	F

Page Fault= 18
Hit Ratio: No of Page Hits/String Size: 2/20
Miss Ratio: No of Miss Hits/String Size: 18/20

0, 3, 2, 1, 4, 5, 0, 3, 1, 2, 1, 3, 4, 0, 2, 3, 4, 5, 1, 2

ii) Optimal

Page Fault= 13
Hit Ratio: No of Page Hits/String Size: 7/20
Miss Ratio: No of Miss Hits/String Size: 13/20

ii) Optimal

0	3	2	1	4	5	0	3	1	2	1	3	4	0	2	3	4	5	1	2
0	0	0	0	0	0	0	0	0	2	2	2	2	2	2	2	2	2	2	2
	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	4	1	1
		2	1	4	5	5	5	1	1	1	1	4	0	0	0	0	5	5	5
F	F	F	F	F	F	H	H	F	F	H	H	F	F	H	H	F	F	F	H

Page Fault= 13
Hit Ratio: No of Page Hits/String Size: 7/20
Miss Ratio: No of Miss Hits/String Size: 13/20

Question:

Assume that there are 4 page frames which are initially empty. If the page reference string is 1, 2, 3, 4, 2, 1, 5, 3, 2, 4, 6, Calculate No. of **Page Faults, Hit Ratio and Miss Ratio** using OPTIMAL page replacement algorithm:

i. First-In-First-Out (FIFO) Algorithm

ii. Least Recently Used (LRU) Algorithm

iii. Optimal Page Replacement

iv. Least Frequently Used (LFU) Algorithm

Question:

- Assume that there are 4 page frames which are initially empty.
  If the page reference string is:

  7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
- Calculate No. of **Page Faults, Hit Ratio and Miss Ratio** using following page replacement algorithm:

  i. First-In-First-Out (FIFO) Algorithm

  ii. Least Recently Used (LRU) Algorithm

  iii. Optimal Page Replacement

  iv. Least Frequently Used (LFU) Algorithm

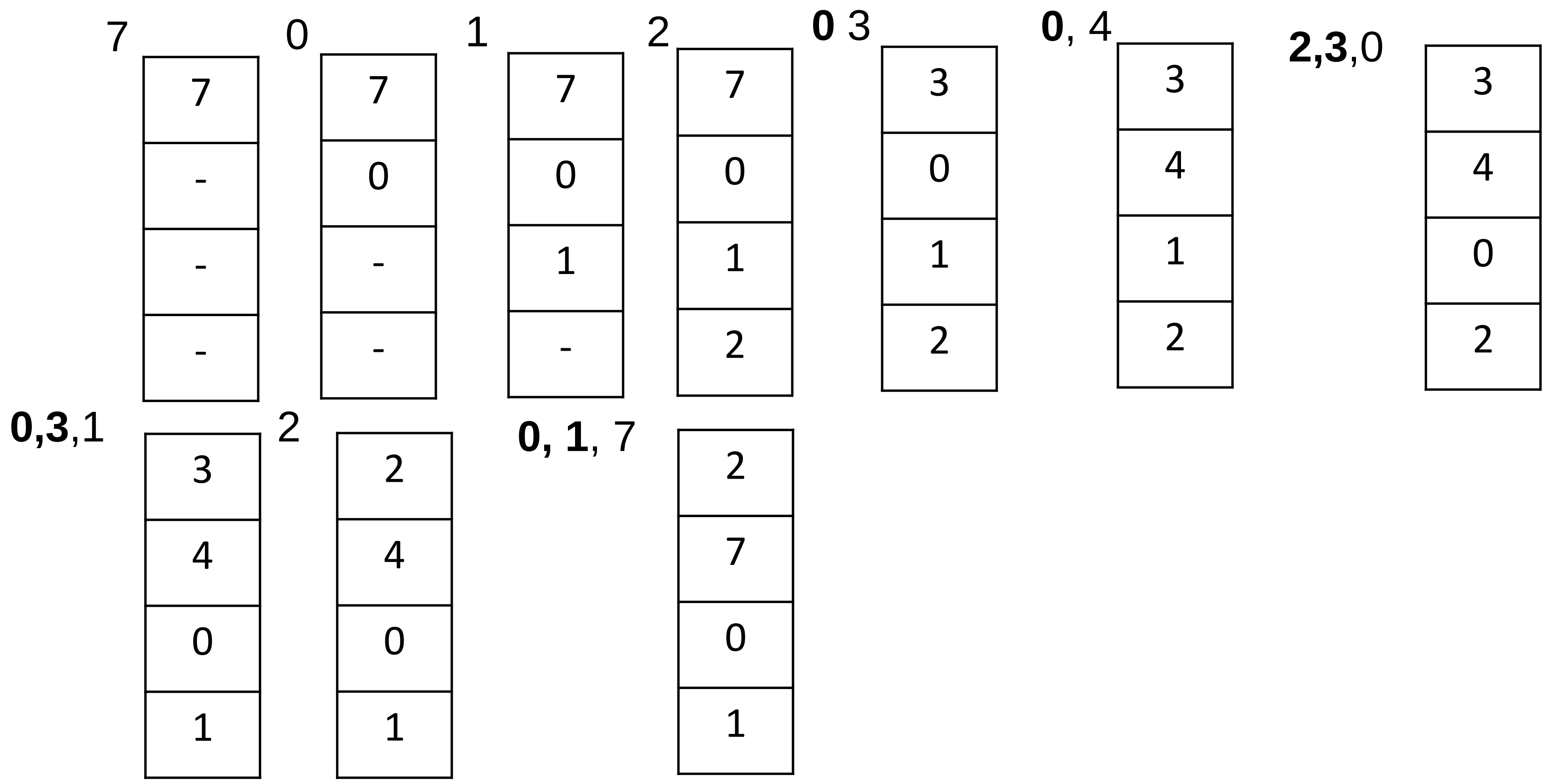
New pages are replaced with the oldest one

Reference String

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

First-In-First-Out (FIFO) Algorithm

For 4 pages per frame



Total No of Page Faults: 10

Hit Ratio: No of Page Hits/String Size: 8/20

Miss Ratio: No of Miss Hits/String Size: 10/20

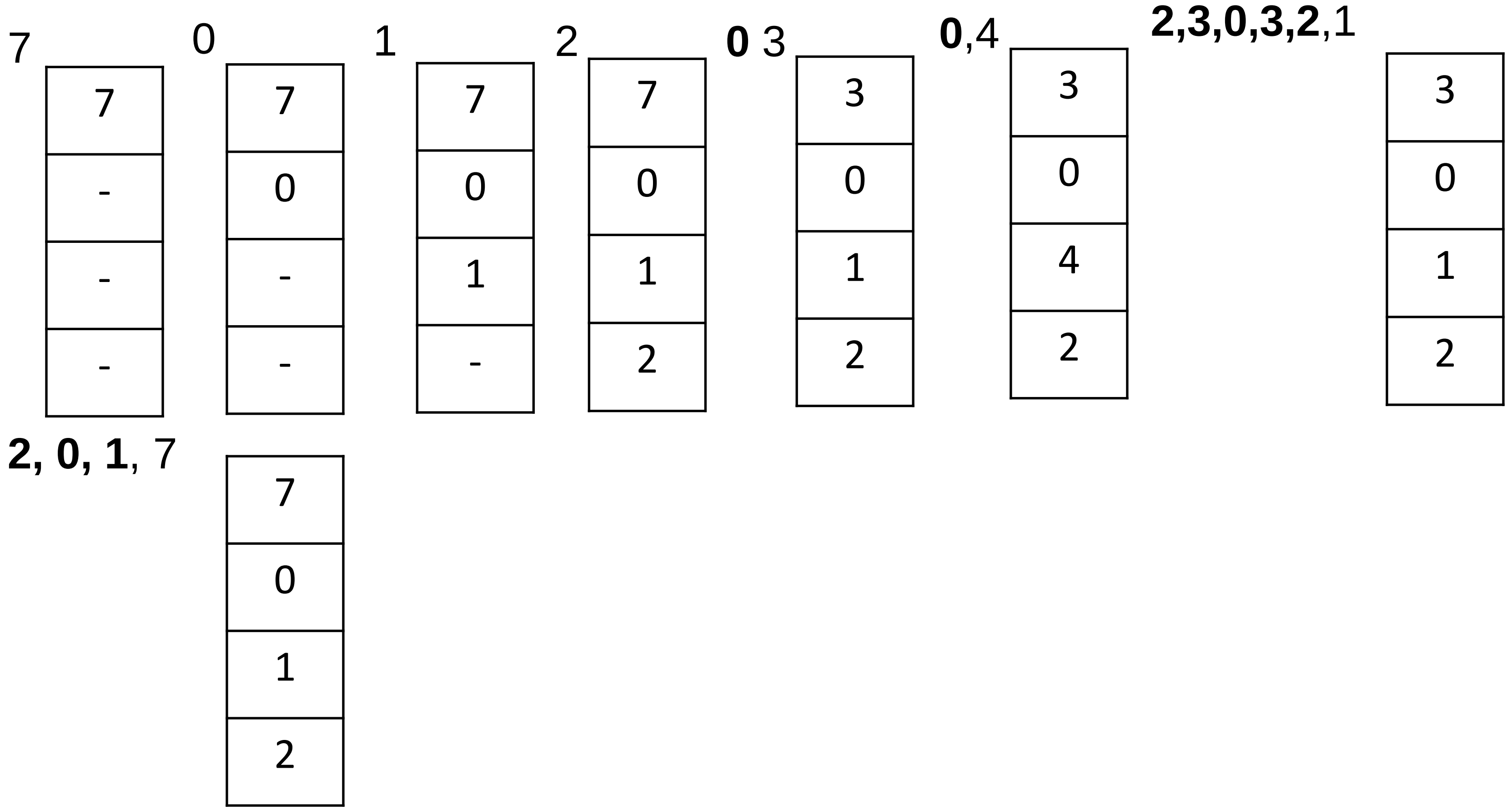
Page which has not been used for longest period of time is replaced

Reference String

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

LRU Page Replacement Algorithm

For **4 pages per frame**



Total No of Page Faults: 8

Hit Ratio: No of Page Hits/String Size: 10/20

Miss Ratio: No of Miss Hits/String Size: 8/20

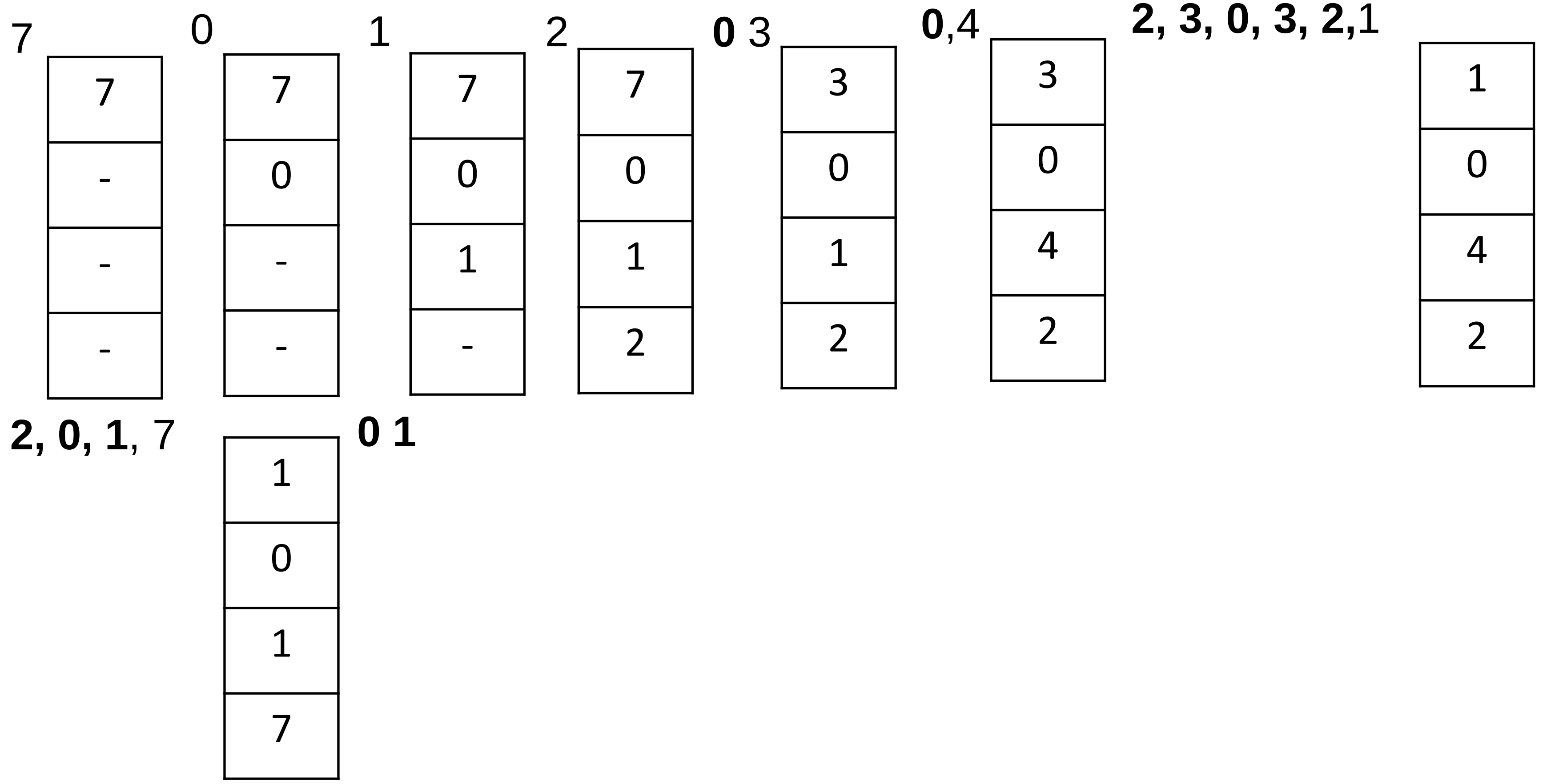
Page that will not be used for longest period of time is replaced

Reference String

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Optimum Page Replacement Algorithm

For 4 pages per frame



Total No of Page Faults: 8

Hit Ratio: No of Page Hits/String Size: 12/20

Miss Ratio: No of Miss Hits/String Size: 8/20

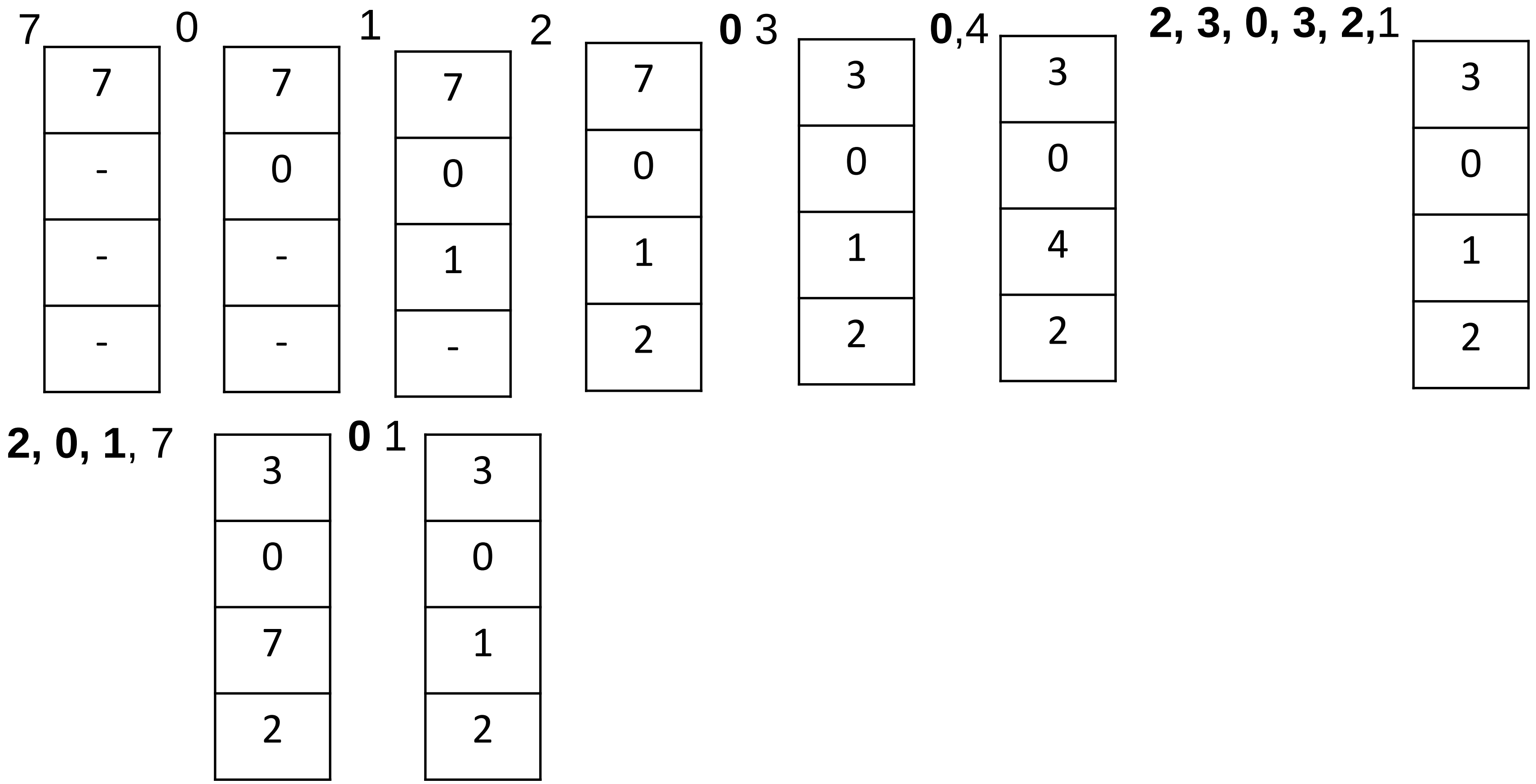
Replace that block in the set that has experienced the fewest references

Reference String

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

LFU Page Replacement Algorithm

For **4 pages per frame**



Block	Count
7	
0	
1	
2	
3	
4	

Total No of Page Faults: 9

Hit Ratio: No of Page Hits/String Size: 11/20

Miss Ratio: No of Miss Hits/String Size: 9/20

❖ Thrashing

- A process is thrashing if its **spending more time in paging than executing.**
- Low CPU Utilization -> OS increases degree of multiprogramming (by introducing new processes)
- A **global page replacement algo** replaces pages with no regard to which to process.
- If process enters new phase – needs more frames – so start faulting & taking pages away from another processes.
- These processes need those pages also – so they also fault & take pages from other processes
- As process waits for paging, the CPU utilization decreases
- So again CPU scheduler increases degree of multiprogramming – as a result cause more page faults
- Again CPU utilization drops – scheduler increases degree of multiprogramming even more – **thus Thrashing has occurred.**
- No work is getting done as time is spent in paging only.

- Local Page Replacement Algorithm can decrease effect of thrashing

  If process starts thrashing, it can't steal frames from other processes.

- Working Set Strategy – provide as many frames as a process needs

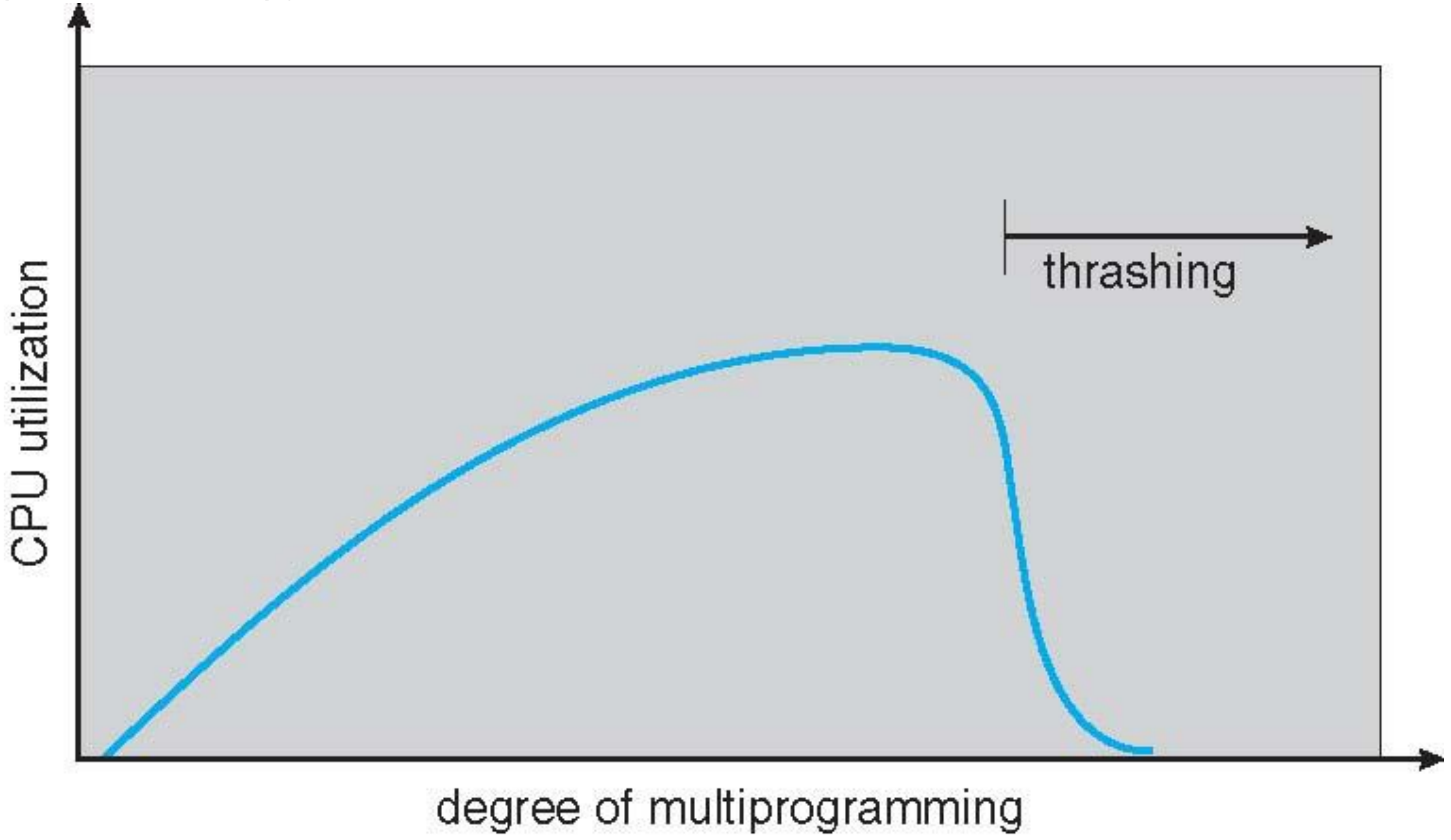


Figure: Thrashing

References

Text Books:

1. **William Stallings**, O**perating System: Internals and Design Principles**, 8th Edition, Prentice Hall, 2014.
2. Abraham Silberschatz, Peter Baer **Galvin** and Greg Gagne, **Operating System Concepts**, 9th Edition, John Wiley & Sons, Inc., 2016.
3. Andrew **Tannenbaum**, **Operating System Design and Implementation**, 3rd Edition, Pearson, 2015.

Reference Books:

1. Maurice J. Bach, Design of UNIX Operating System, 2nd Edition, PHI, 2004.
2. Achyut Godbole and Atul Kahate, Operating Systems, 3rd Edition, McGraw Hill Education, 2017.
3. The Linux Kernel Book, Remy Card, Eric Dumas, Frank Mevel, 1st Edition, Wiley Publications, 2013.