UNIT III

STORAGE MANAGEMENT

APEC

Introduction

- CPU ➔ shared by many processes (or) set of processes.

- Aim:
  - Maximize CPU utilization ➔ keep more processes in memory.

APEC

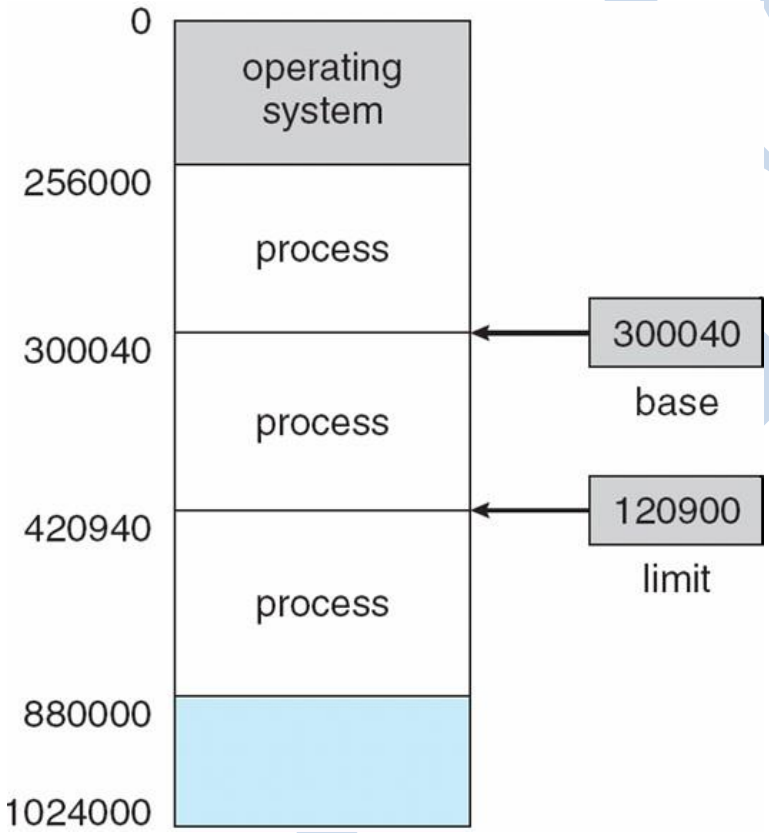
Background

- Program must be brought (from disk) into memory for execution
- Memory consists of large array of words (or) bytes, where each word has its own address.
- **Cache** sits between main memory and CPU registers

APEC

Base and Limit Registers

- A pair of **base** and **limit** registers define the logical address space



Base = 300 040
Limit = 120 900
---
420 940

Contd…

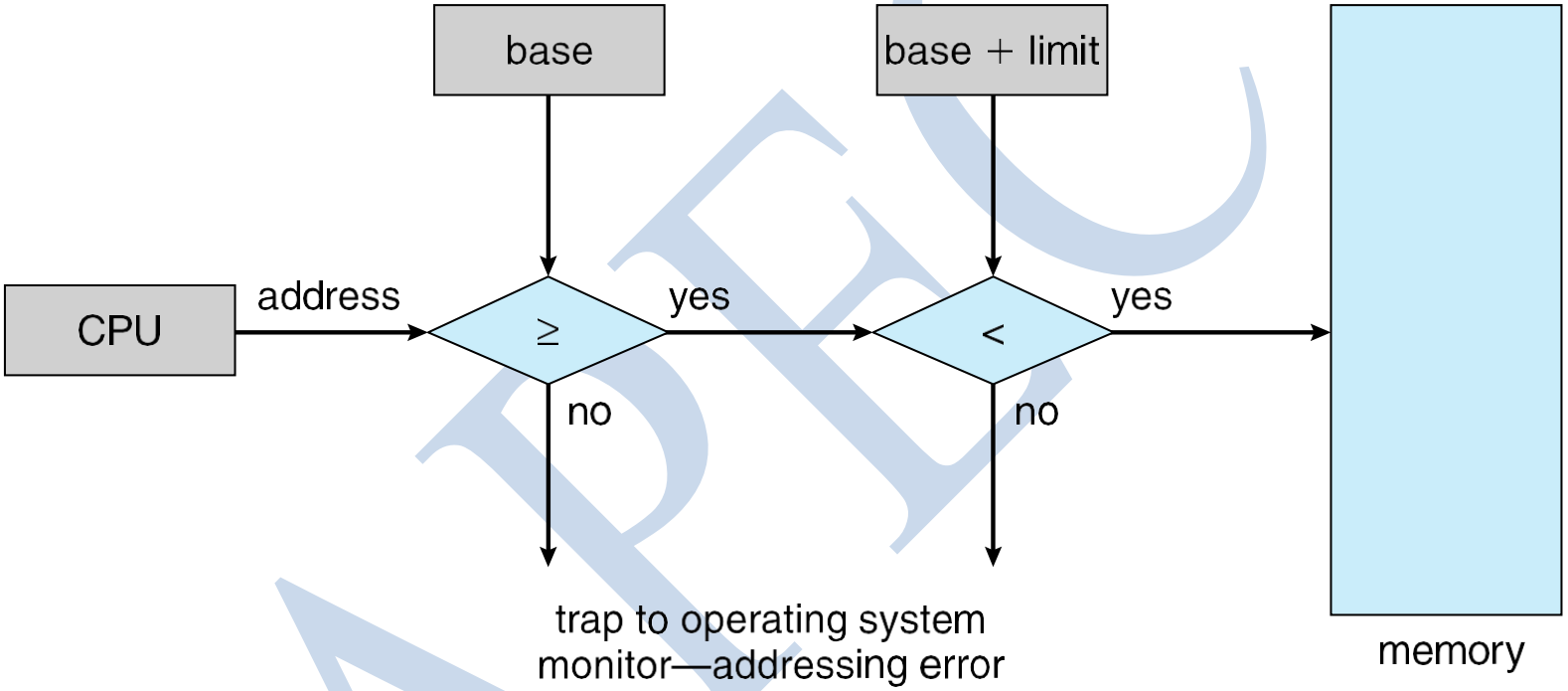
- In order to store temporary data, CACHE is used.
  - 1st thing, we need to check whether each process has enough memory space.

To access memory, 2 registers are used.

- Base Register:
  - Holds the base physical memory address
- Limit Register:
  - Specifies the range

APEC

Hardware Address Protection



CPU must check every memory access is in between base and limit register

Contd...

- Protection of memory space is done by comparing every address with the registers. Any attempt by a program executing in user mode to access OS memory (or) other programs memory results in a trap, which results in a fatal error.

APEC

Disk ☒➔ Memory

- Processes shift between hard disk & main memory
  - Whenever any process need to be executed it should be brought to main memory for execution.

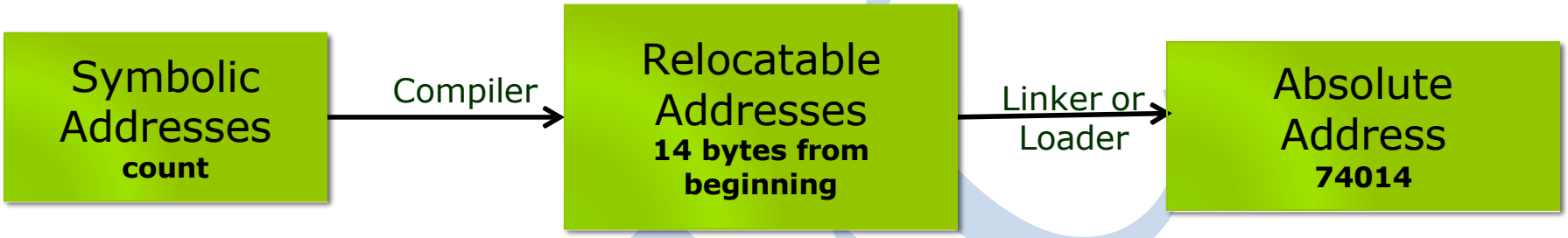
APEC

Address Binding

- In most cases, a user program goes through several steps before being executed
  - Compilation, linking, executable file, loader creates a process
- Addresses are represented in different ways at different stages of a program's life
  - Each binding maps one address space to another
- Source code -- addresses are usually symbolic
  - E.g., variable `count`
- A compiler typically binds these symbolic addresses to relocatable addresses
  - E.g., "14 bytes from beginning of this module"
- Linker or loader will bind relocatable addresses to absolute addresses
  - E.g., 74014

APEC

Addresses used:



APEC

Address Binding

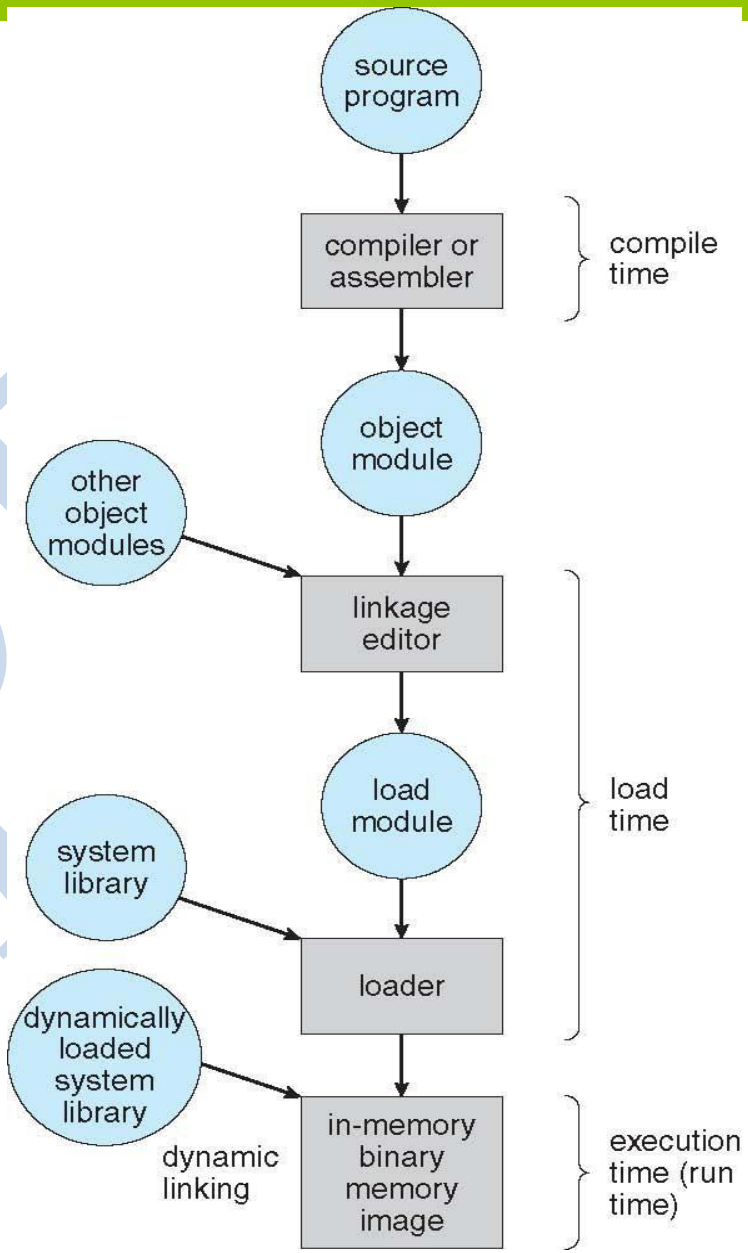


Fig. Multistep Processing of a User Program

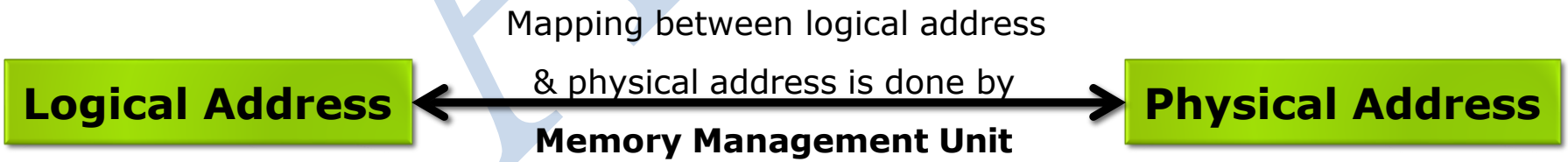
Binding of Instructions and Data to Memory

- Address binding of instructions and data to memory addresses can happen at three different stages
  - **Compile time**: If we know at compile time where the process will reside in memory, **absolute code** can be generated
  - **Load time**: If its not known at compile time where the process reside in memory, the compiler generate **relocatable code**
  - **Execution time**: Actual Execution

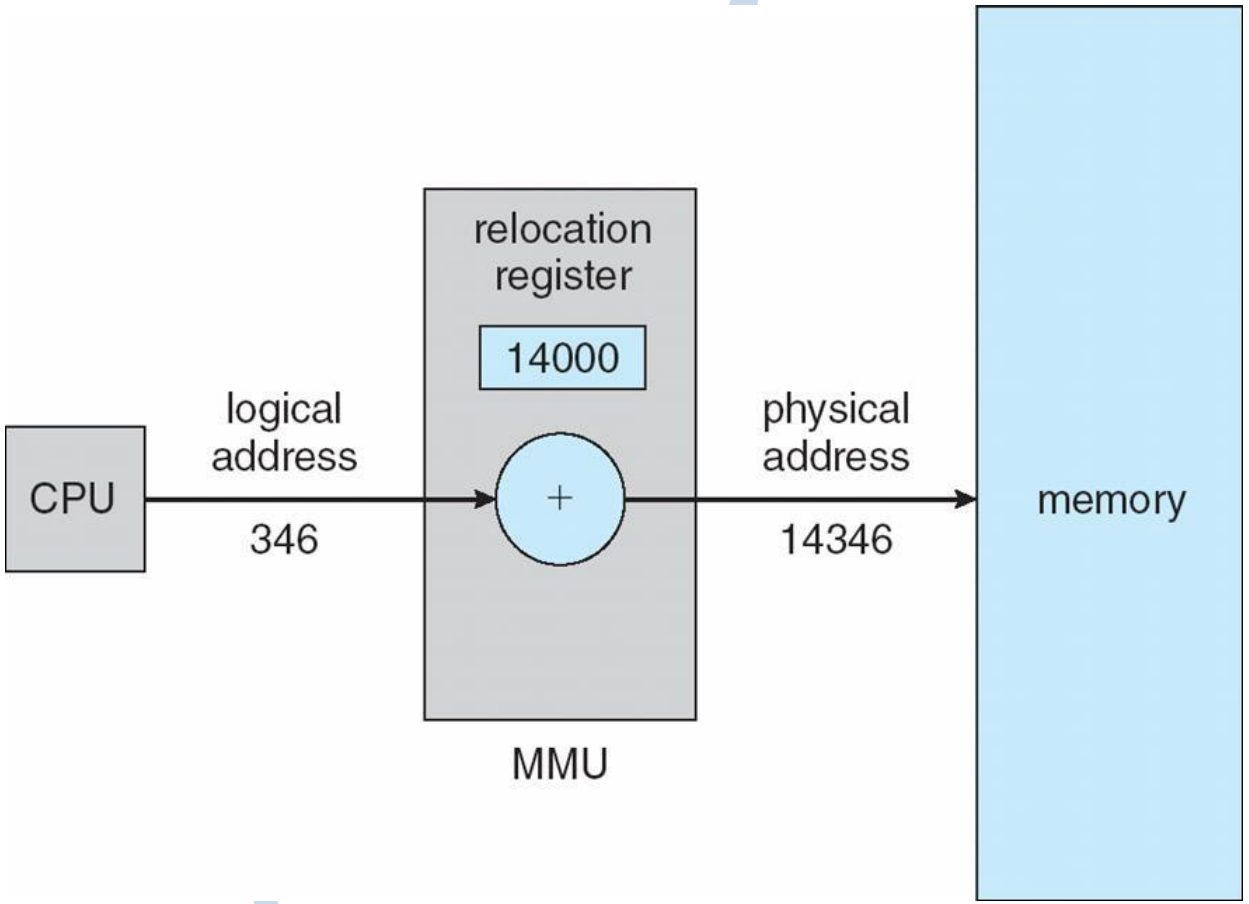
APEC

Logical vs. Physical Address Space

- Logical address or virtual address
  - Logical address space.
  - Generated by the CPU.
  - Ex: 0 – Max
- Physical address or memory address
  - Physical address space.
  - Address seen by the memory unit
  - Ex: R+0 - R+Max, where R ➔ Base Register

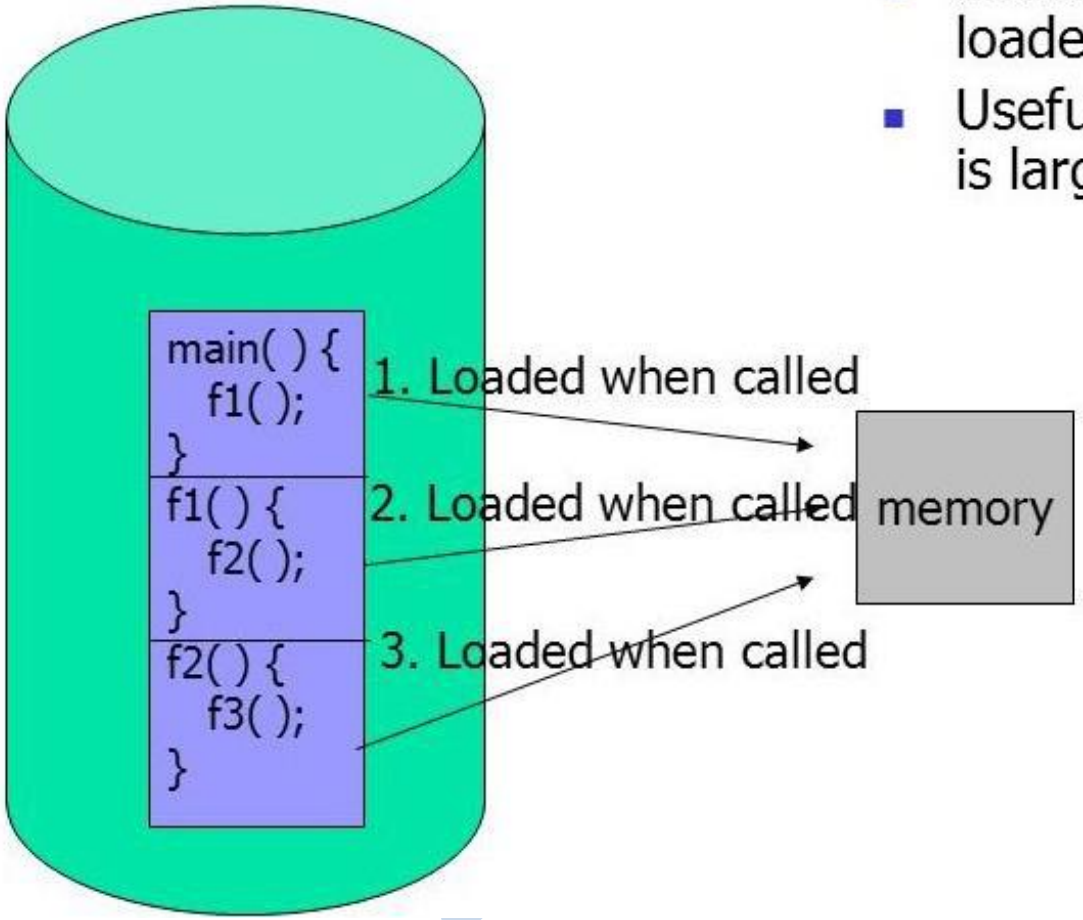


Dynamic relocation using a relocation register



Dynamic Loading

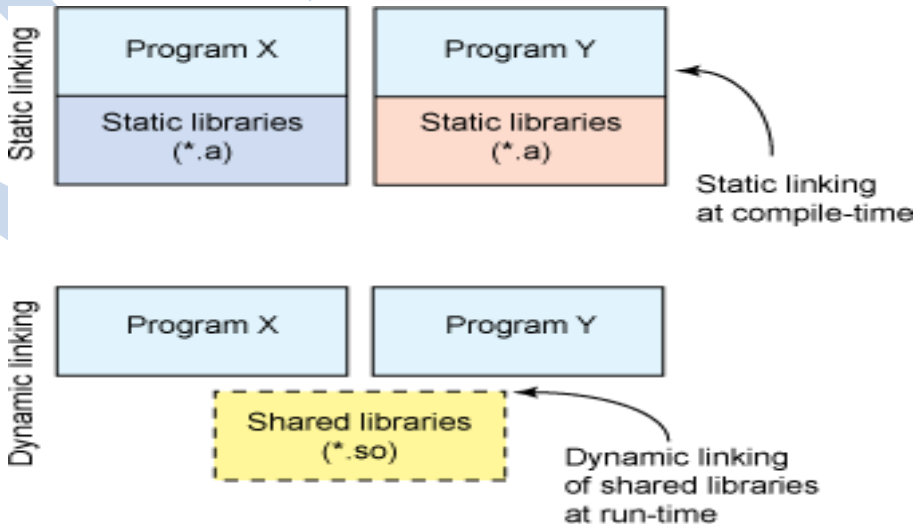
- Unused routine is never loaded
- Useful when the code size is large



Dynamic linking

- Static linking
- System libraries and program code seperately
- Some OS support only static linking

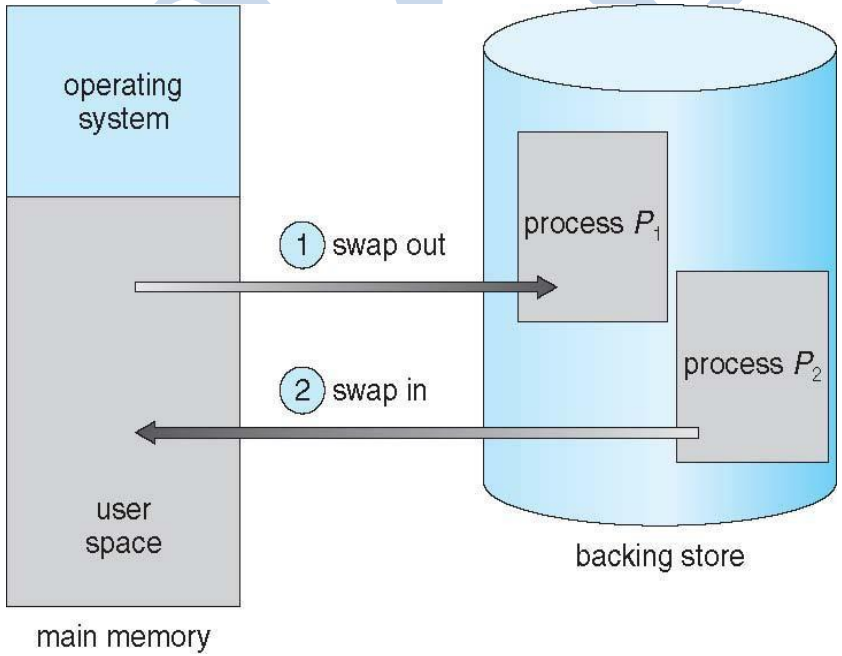
- Dynamic linking
- Linking is postponed until execution time
- Wastage of both disk space and main memory is avoided



APEC

Swapping

- A process can be **swapped** temporarily out of memory to a backing store, and then brought back into memory for continued execution
  - Total physical memory space of processes can exceed physical memory
  - This increases the degree of multiprogramming in a system
    - **Backing store** – Normally its hard disk



Memory Resident

- Certain programs, can be marked as being **memory resident**, which means that the operating system is not permitted to swap them out to a storage device; they will always remain in memory.
  - Example: Calendars, calculators

APEC

Transient operating -system code

- OS contains space for device drivers. If device driver not commonly used, no need to keep the code and data in memory.
  - Such code that comes and goes as needed is called transient operating system codes.

APEC

Contiguous Memory Allocation

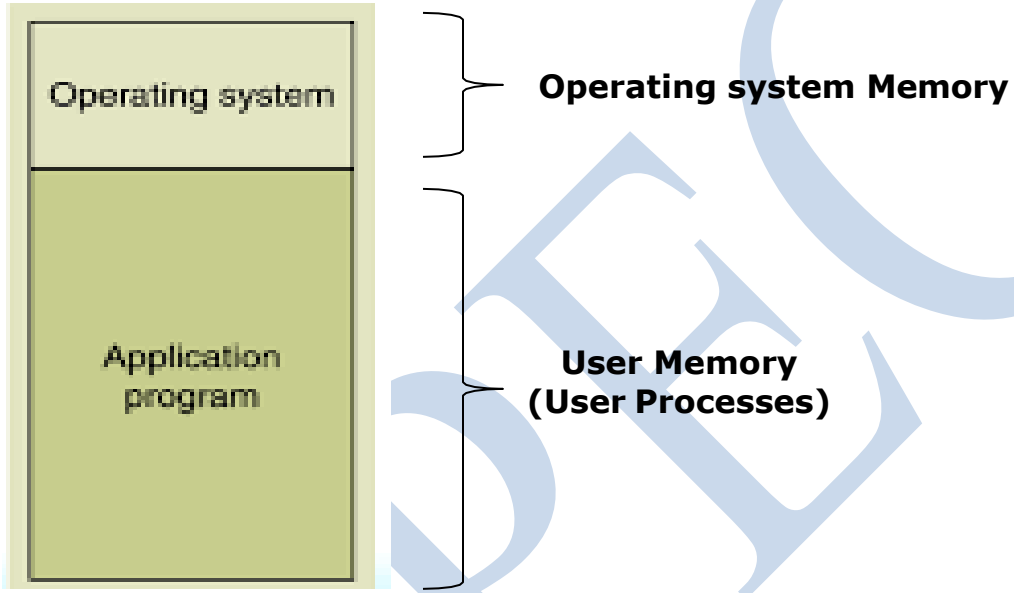


Fig. Single-partition allocation

- There are only two programs in memory
  - The Operating System
  - The application program
- This approach is called single contiguous memory management.

Contiguous Memory Allocation

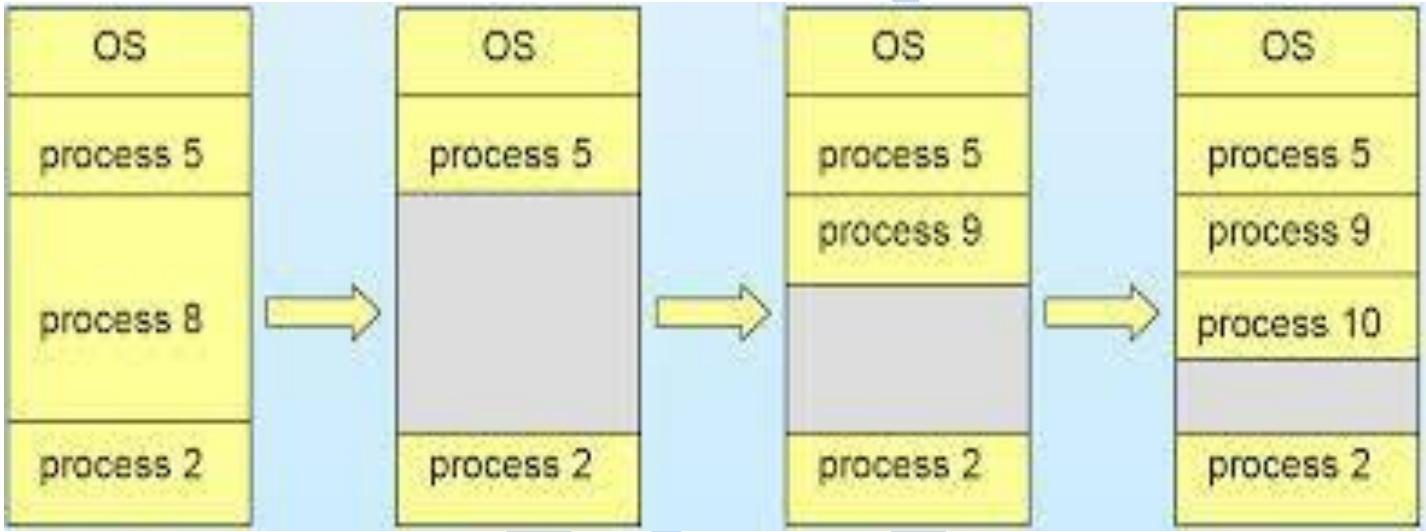


Fig. Multiple-partition allocation

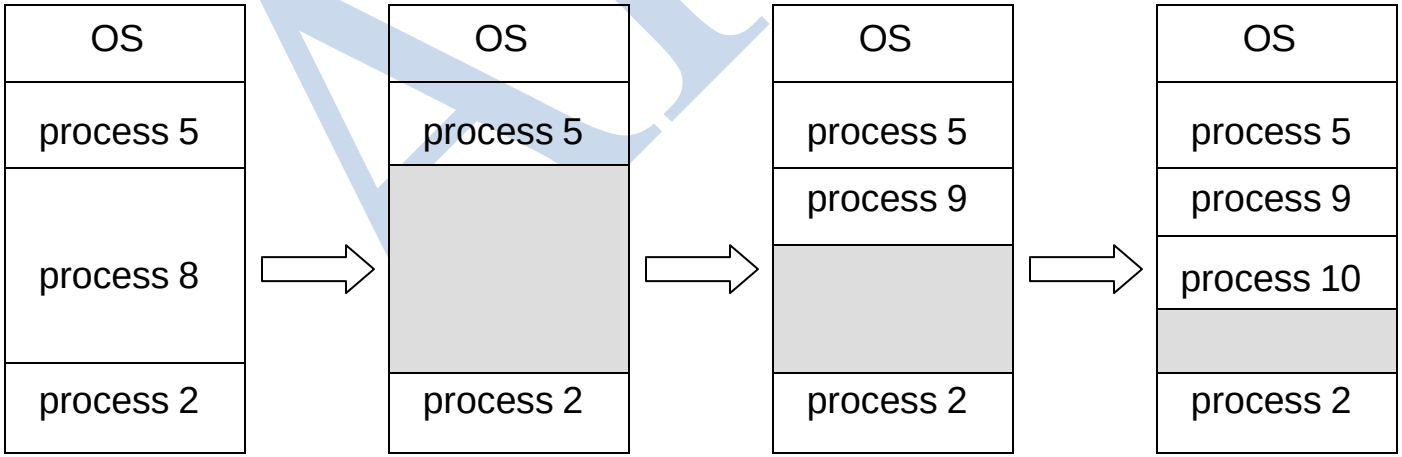
Partitioning

- Memory having single partition
- Equal size partition → **Multiprogramming Fixed Tasks(MFT). Used in IBM OS/360. Memory is divided into equal sized partitions. Not used now**
- Unequal size partition → **Multiprogramming Variable Tasks(MVT).**

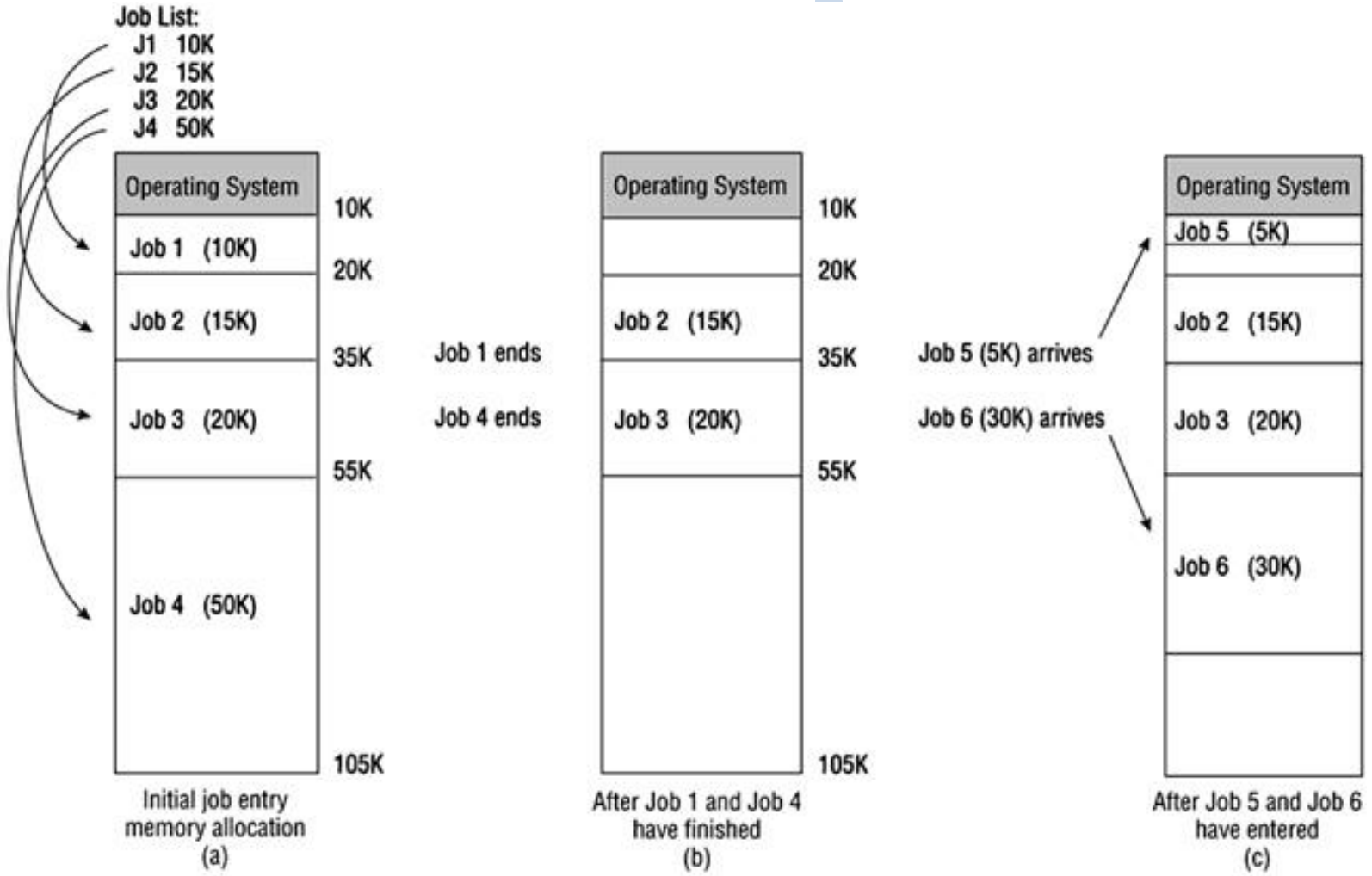
Contiguous Allocation (Cont)

Hole: **Free part of memory is called a hole.**

- Multiple-partition allocation
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it
  - Operating system maintains information about:
    a) allocated partitions b) free partitions (hole)



Examples


Job List:
J1 10K
J2 15K
J3 20K
J4 50K
Operating System
Job 1 (10K)
Job 2 (15K)
Job 3 (20K)
Job 4 (50K)
10K
20K
35K
55K
105K
Initial job entry memory allocation
(a)
Job 1 ends
Job 4 ends
Operating System
Job 2 (15K)
Job 3 (20K)
10K
20K
35K
55K
105K
After Job 1 and Job 4 have finished
(b)
Job 5 (5K) arrives
Job 6 (30K) arrives
Operating System
Job 5 (5K)
Job 2 (15K)
Job 3 (20K)
Job 6 (30K)
After Job 5 and Job 6 have entered
(c)

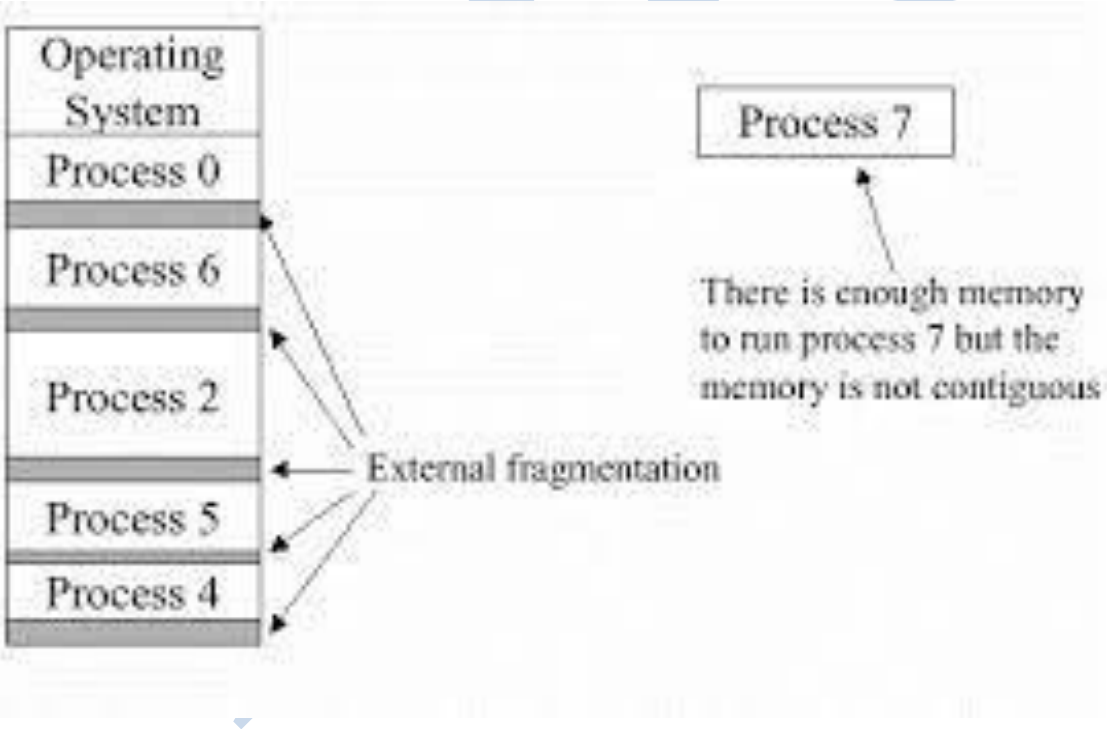
Internal Fragmentation

- Main memory is broken into fixed sized partitions.
- The process size is smaller than the size of the partition. So, rest space in partition is wasted.
- It is called internal fragmentation.



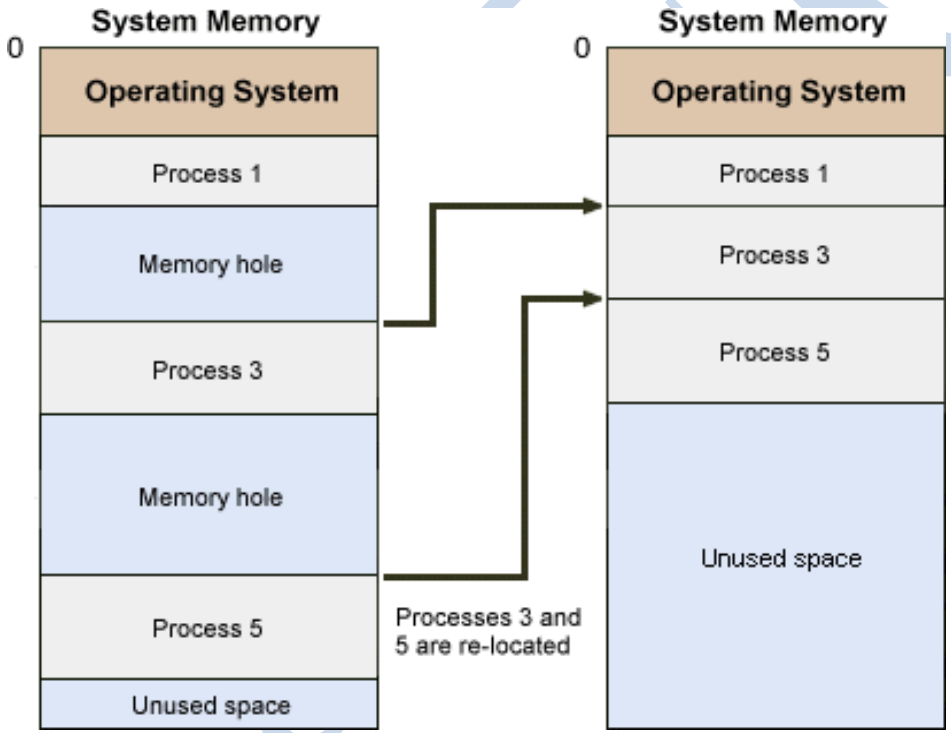
External Fragmentation

- Various free memory blocks are available, but not contiguous ➔ External Fragmentation.



Compaction

- To avoid the external fragmentation problem, shuffle all the free holes to one part.



Reforming the memory is called **COMPACTION**

Partition Selection Algorithms or Memory allocation algorithms or Dynamic Storage Allocation Methods

- Whenever process comes, many holes are available. The OS selects any of the hole based on the following algorithms.
  - First Fit algorithm
    - Allocate the first hole that is big enough
  - Best Fit algorithm
    - Allocate the smallest hole that is big enough
  - Worst Fit algorithm
    - Allocate the largest hole

Example 1

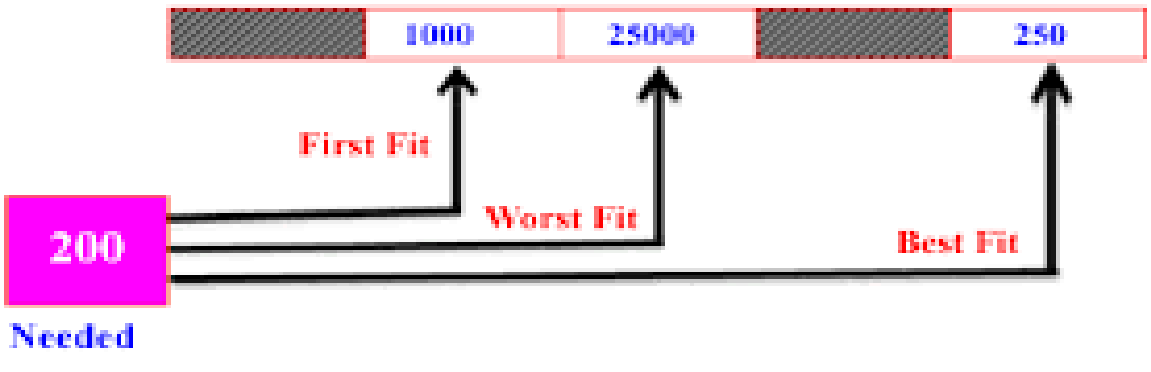
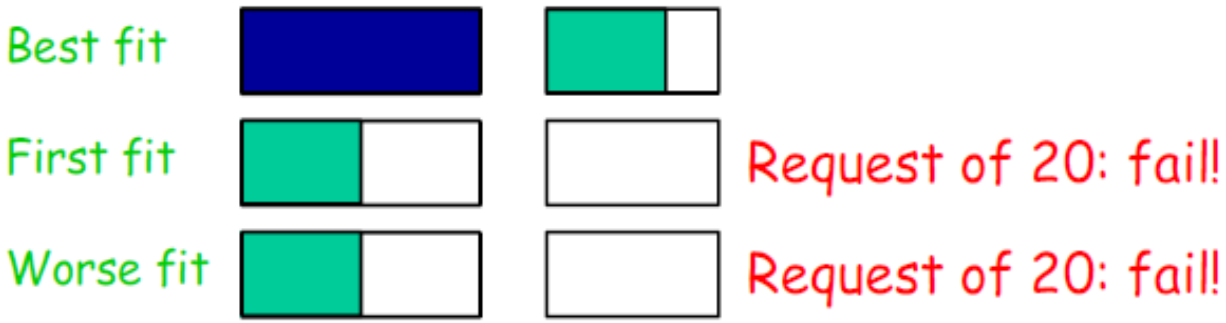


Fig: Showing First Fit, Worst Fit and Best Fit Memory Allocations

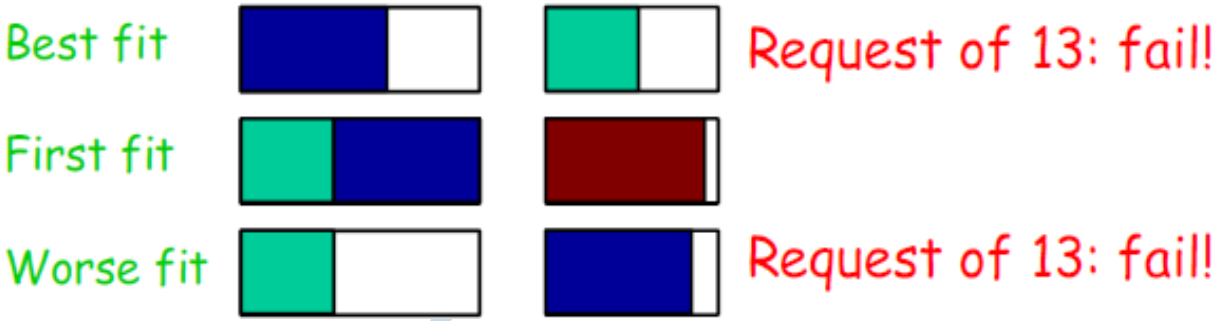
First-fit and best-fit better than worst-fit in terms of speed and storage utilization

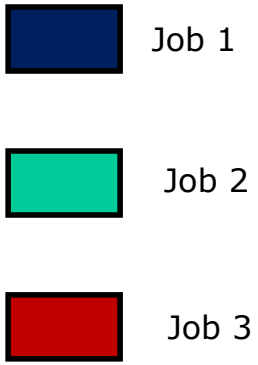
Example 2

- Free space: 2 blocks, size 20 and 15
- Workload 1: allocation requests: 10 then 20



- Workload 2: allocation requests: 8, 12, then 13





Paging

- In Paging, Physical address space ➔ Noncontiguous;
  - Hence avoids external fragmentation & compaction.

- Here,
- Physical memory is divided into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes

- Logical memory is divided into blocks of same size called **pages**

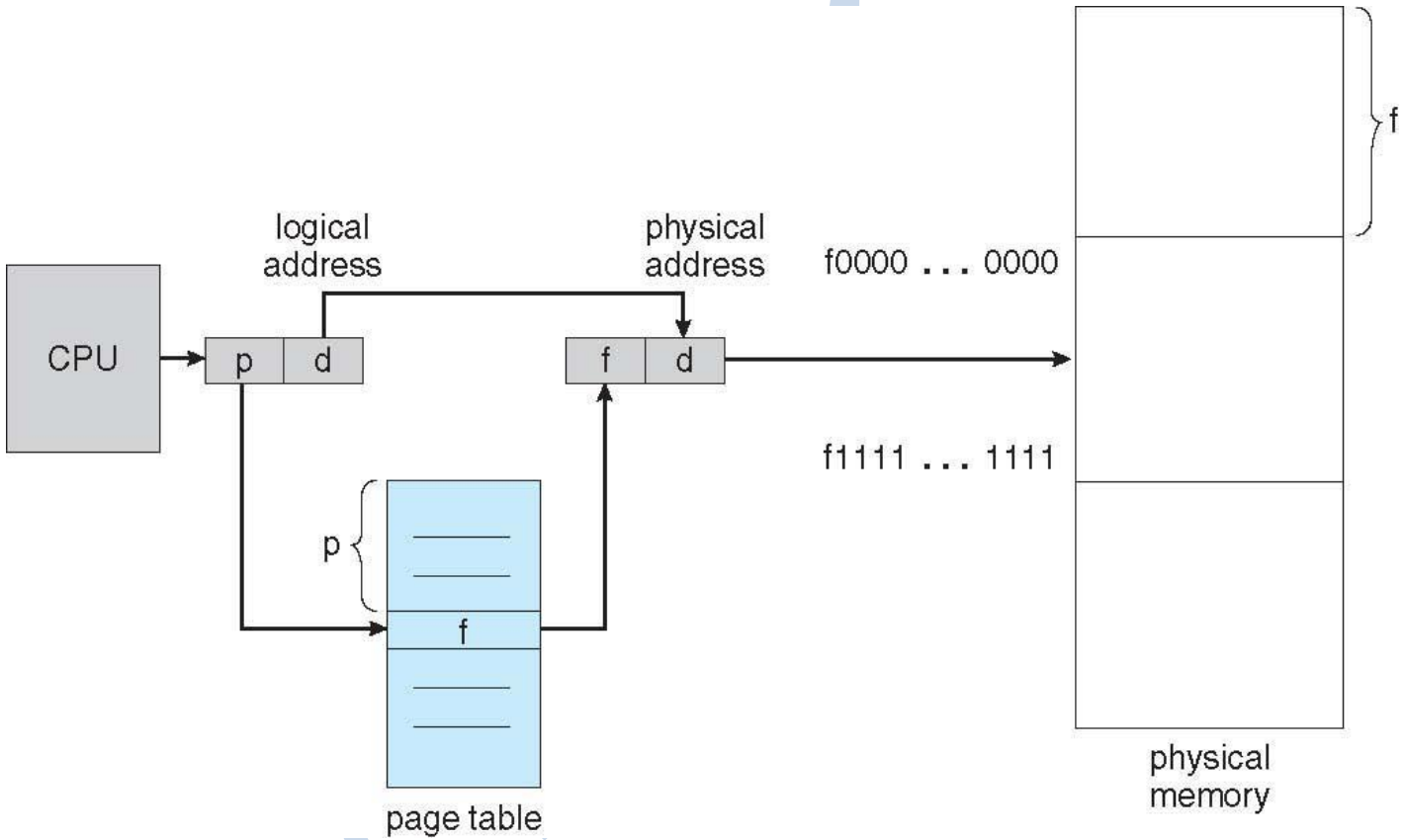
Hardware Support:

- The address generated by CPU has 2 parts
  - Page Number (P)
  - Page Offset (d)

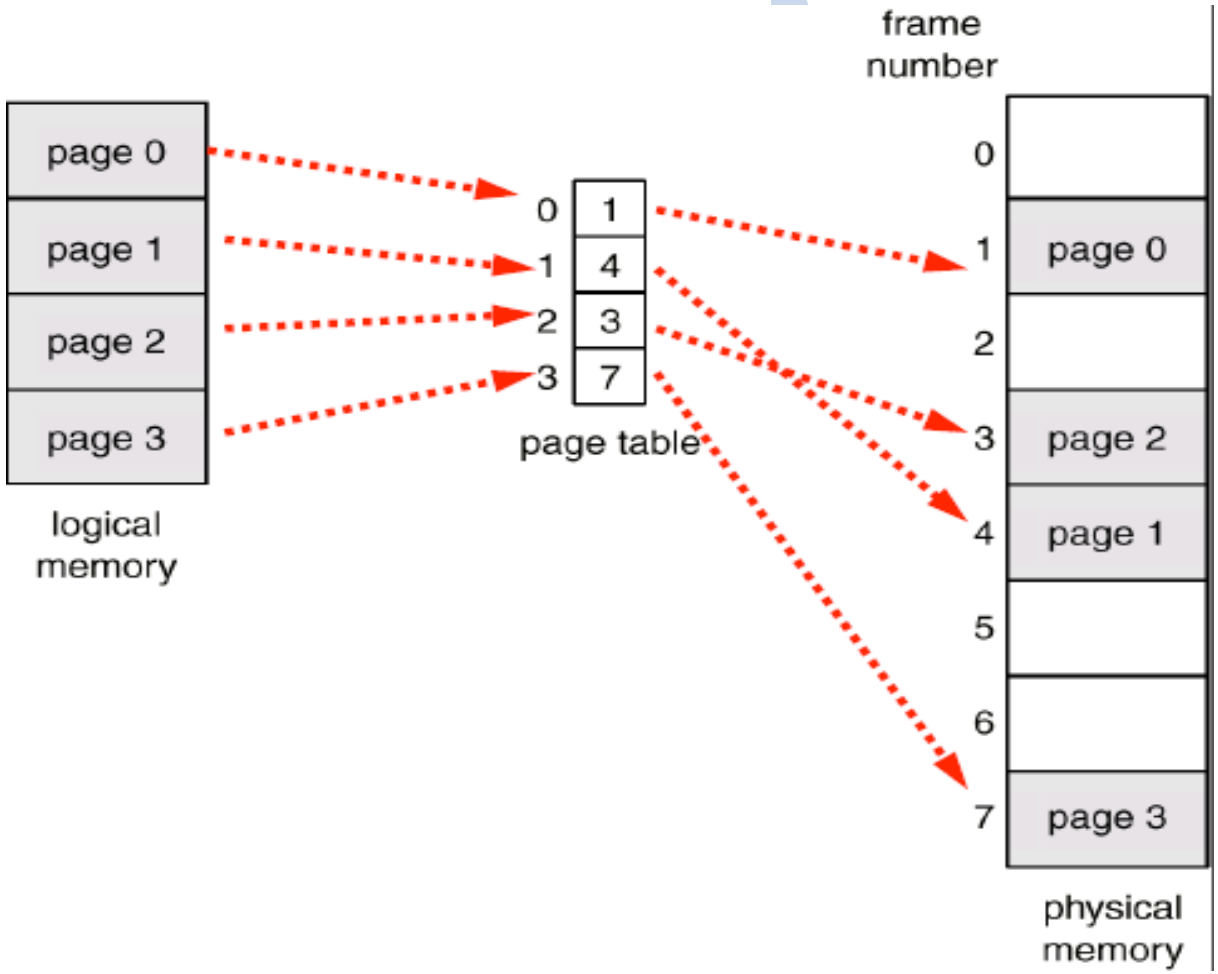
- The page number is used as an index in the page table. The page table contains the base address of each page in physical memory. Then the base address is combined with the page offset to define physical memory.

APEC

Paging Hardware

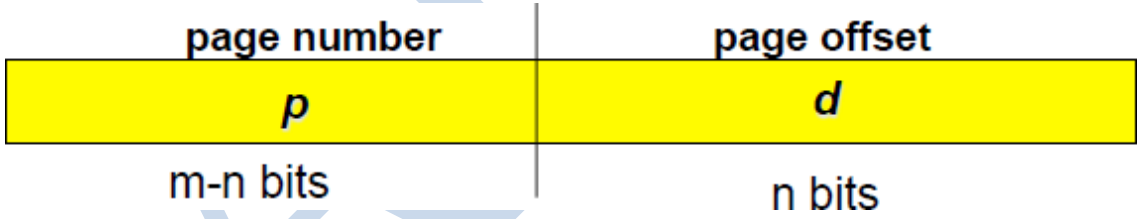


Paging Model of Logical and Physical Memory



Contd...

- If size of logical address is 2^m, and the page size is 2^n, then, m-n bits is the page number and n bits is the page offset.

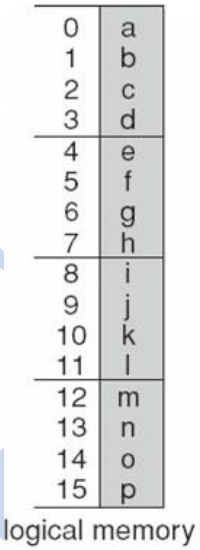


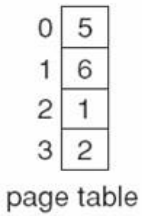
Paging Example

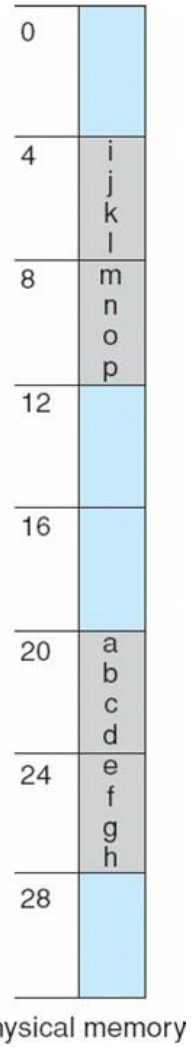
n=2 ; *m*=4;

Page size = 4 bytes;

Physical memory = 32 bytes (8 pages)







How to map logical memory ➔ Physical memory?

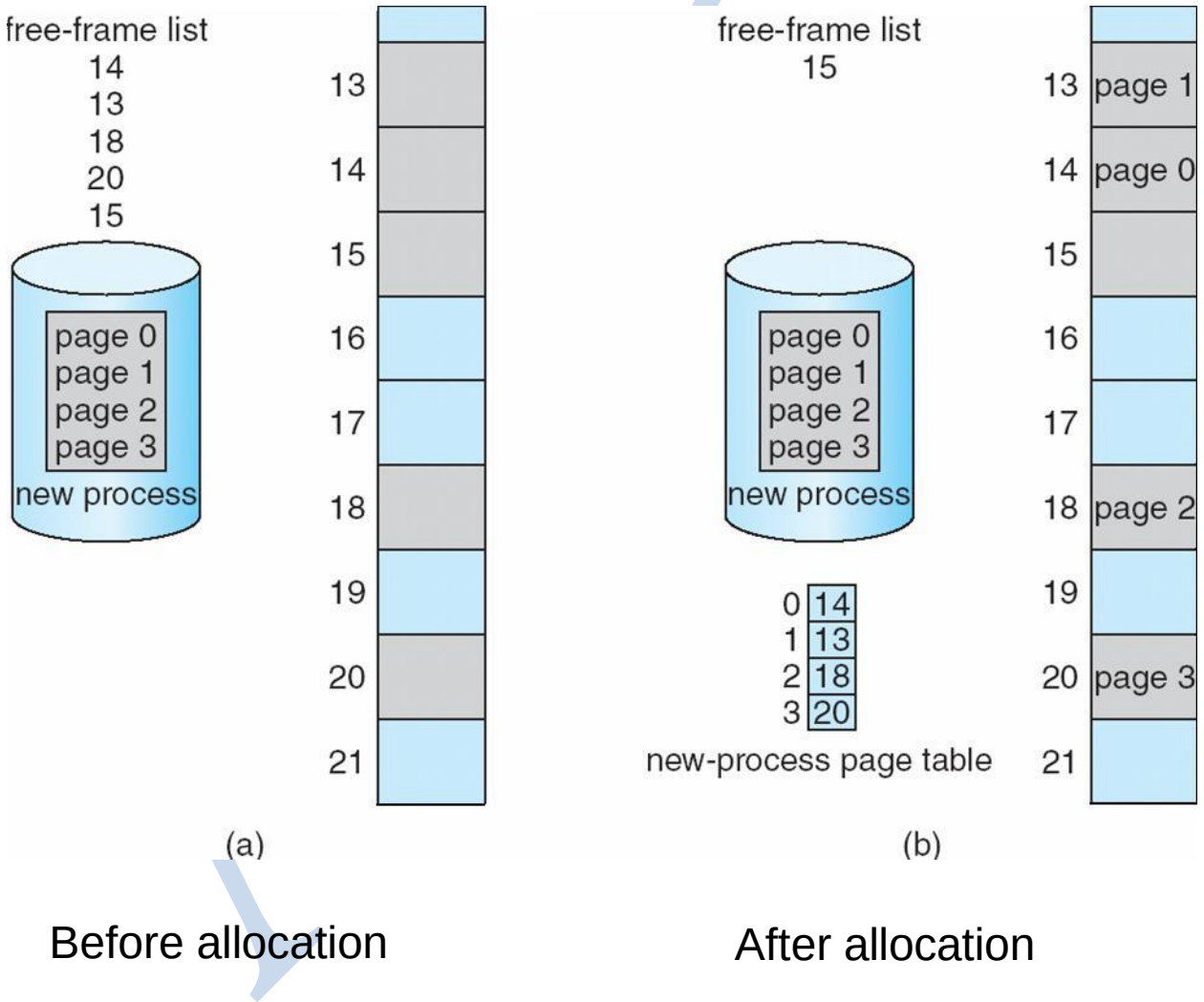
Logical Address = 0 ➔Page =0 ; offset = 0

Page 0 is in Frame 5

Logical address =0 maps to Physical address = 20 [(5x4) +0]

Logical address =3 (Page 0, offset 3) maps to Physical address = 23 [(5x4) +3]

Free Frames



Before allocation

After allocation

Implementation of Page Table

- Page table is kept in main memory via Registers
  - **Page-table base register** (**PTBR**) points to the page table
  - **Page-table length register** (**PTLR**) indicates size of the page table

- Normally two memory accesses to access a byte.
  1) One for the page table entry
  2) One for the byte

- The two memory access problem can be solved by the use of a special hardware cache called associative memory or translation look-aside buffers (**TLBs**)

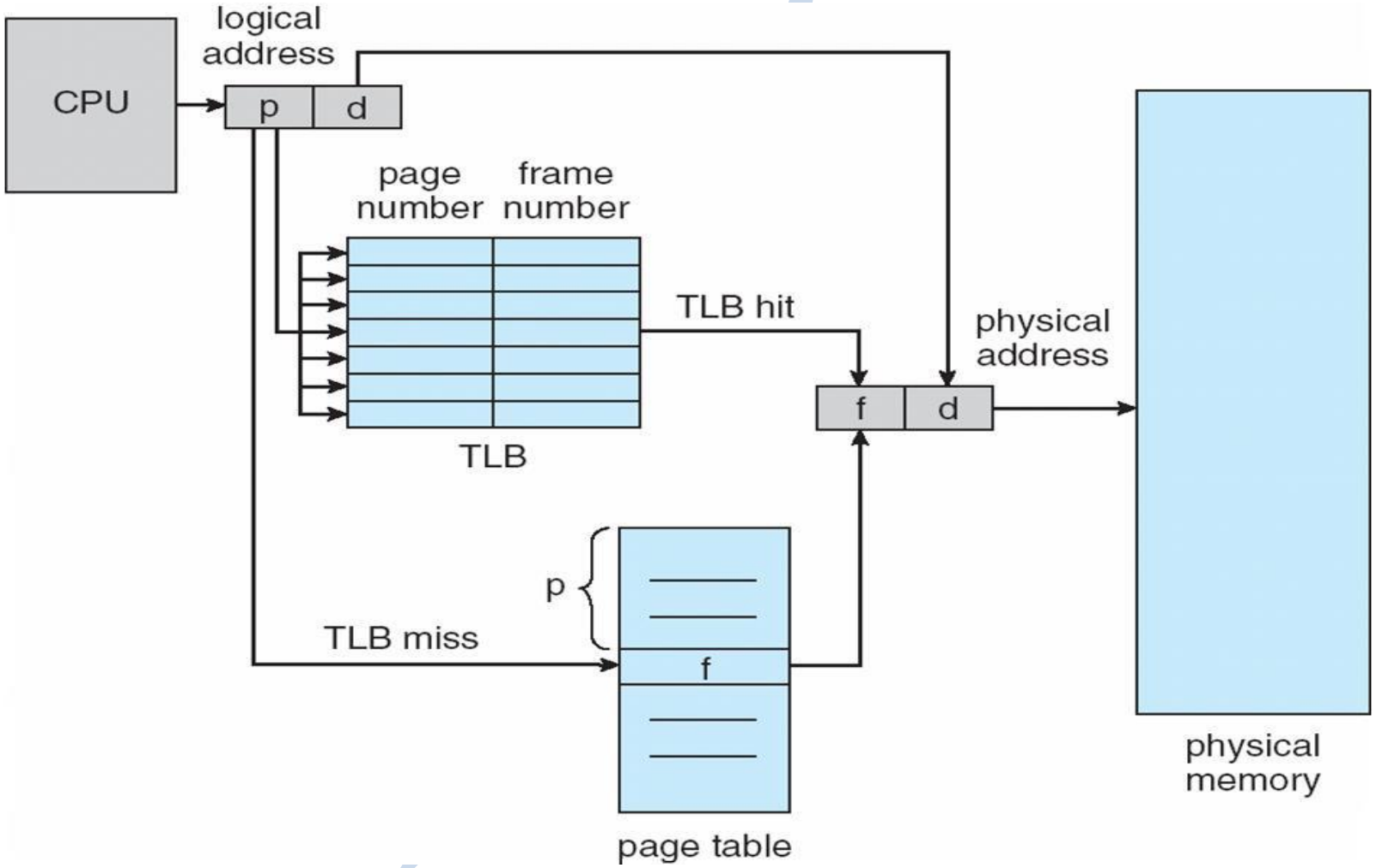
APEC

Implementation of Page Table (Cont.)

- Some TLBs store **address-space identifiers** (**ASIDs**) in each TLB entry.
- **ASI:** Uniquely identifies each process to provide address-space protection for that process

APEC

Paging Hardware With TLB



Effective Access Time (EAT)

Hit ratio: Percentage of times that a page number is found in the TLB.

Example 1:

Consider α = 80%, ε = 20ns for TLB search, 100ns for memory access

- EAT = 0.80 x 100 + 0.20 x 200 = 120ns
- (ie) 80% of TLB Hit + 20% of TLB miss

Example 2:

- α = 99%, ε = 20ns for TLB search, 100ns for memory access
  - EAT = 0.99 x 100 + 0.01 x 200 = 101ns

APEC

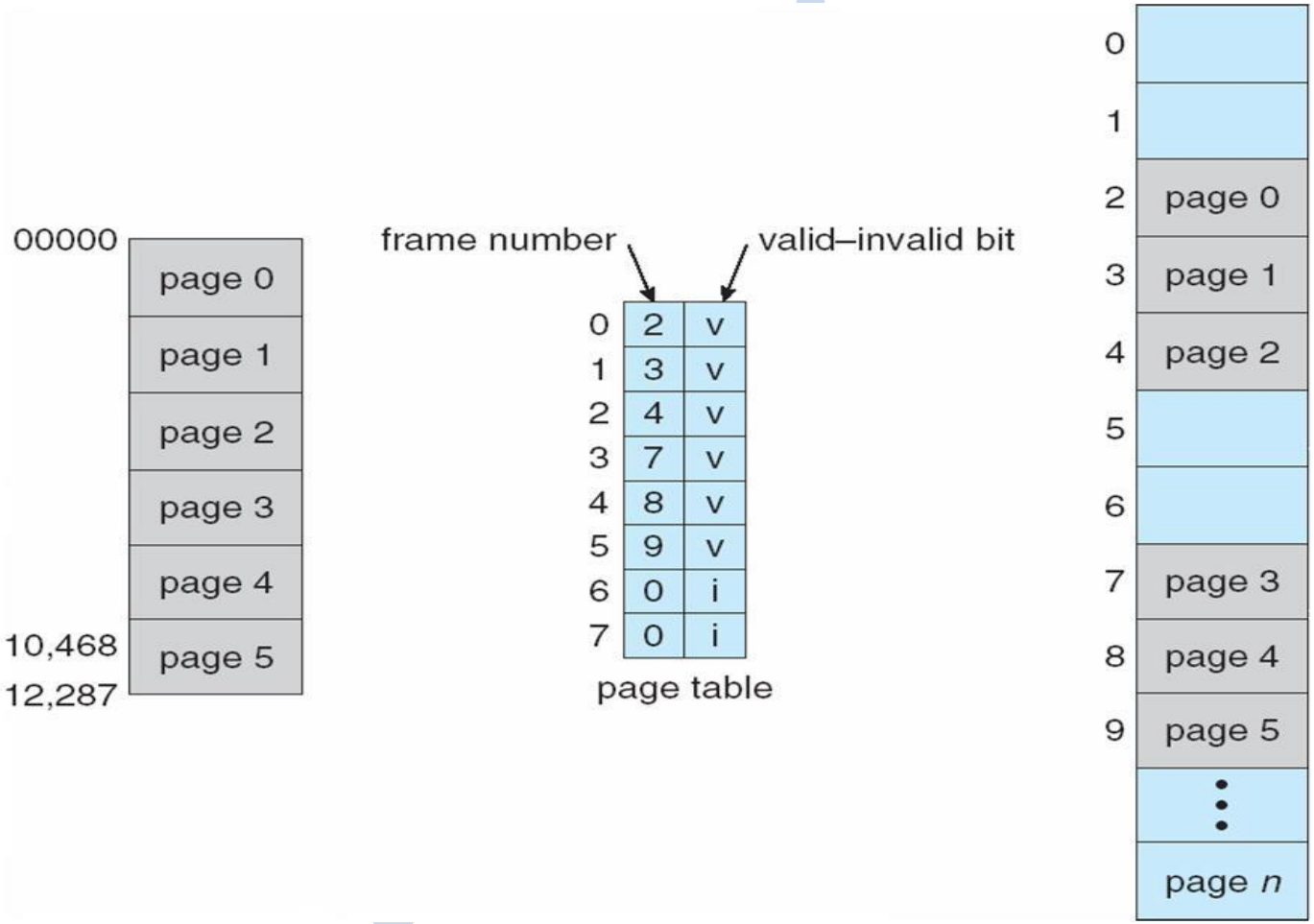
Memory Protection

- Protect the pages through hardware
- Its done by adding a extra bit **(Valid – Invalid bit)**

- When the bit is valid ➔ its valid page
  - (ie) corresponding logical address space is legal.
- When the bit is not valid ➔ the page is not available
  - (ie) trap to OS.

APEC

Valid (v) or Invalid (i) Bit in a Page Table



Shared Pages

- Assume 40 users ➔ each user needs text editor.
  - Text Editor ➔ 150 KB of code
  - Data ➔ 50 KB of code
- If the code is **reentrant code** (or) **pure code** ➔ it can be shared.

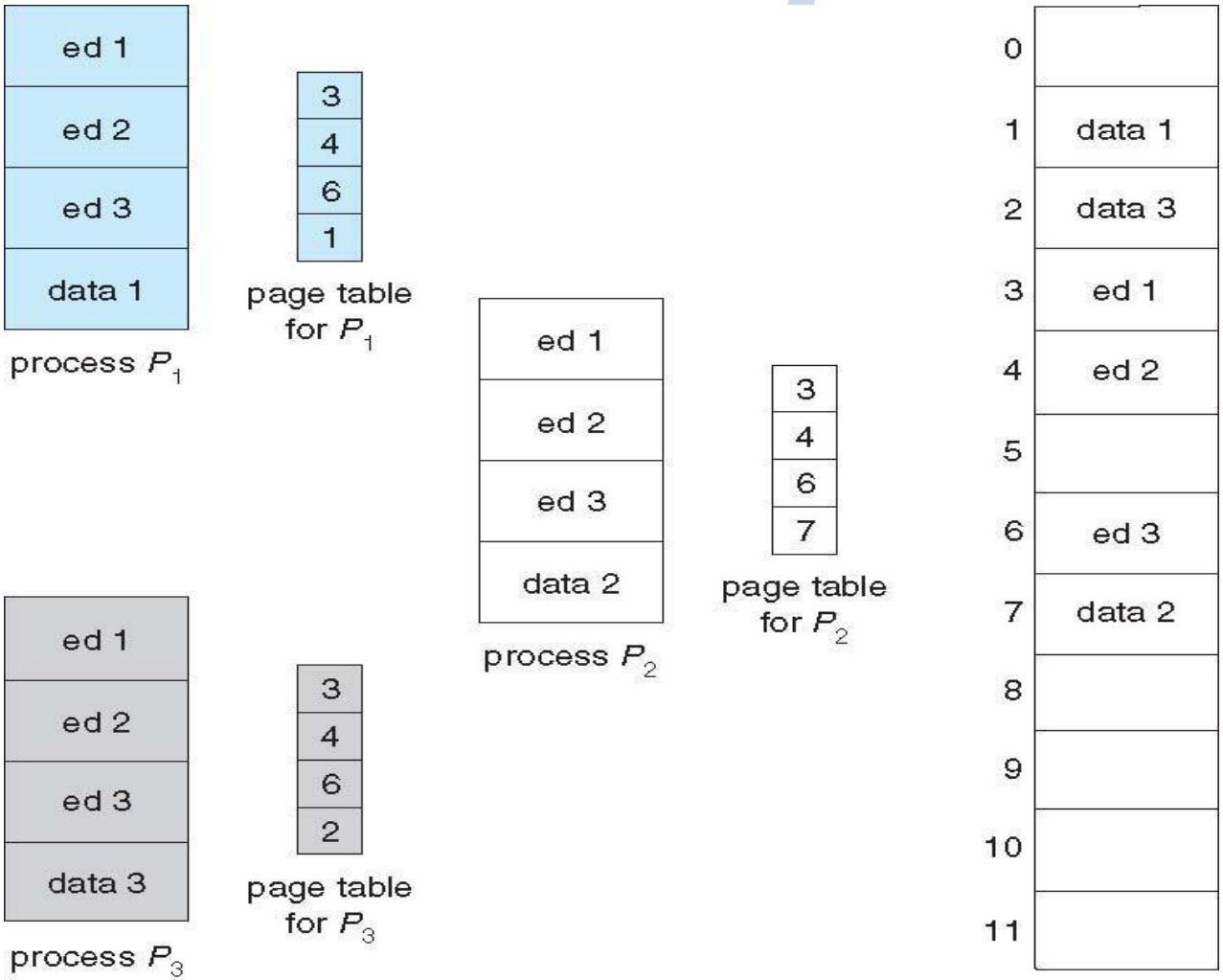
Reentrant code:

- It is non self modifying code; (ie) it never changes during execution.
  - (40 users x 150 KB) + 150 KB) = 2000 + 150 = 2150 KB
- 2150 KB is needed instead of 8000 KB

Sharing Memory among Processes

APEC

Shared Pages Example



Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

APEC

Hierarchical Page Tables

- Break up the logical address space into multiple page tables
- A simple technique is a two-level page table

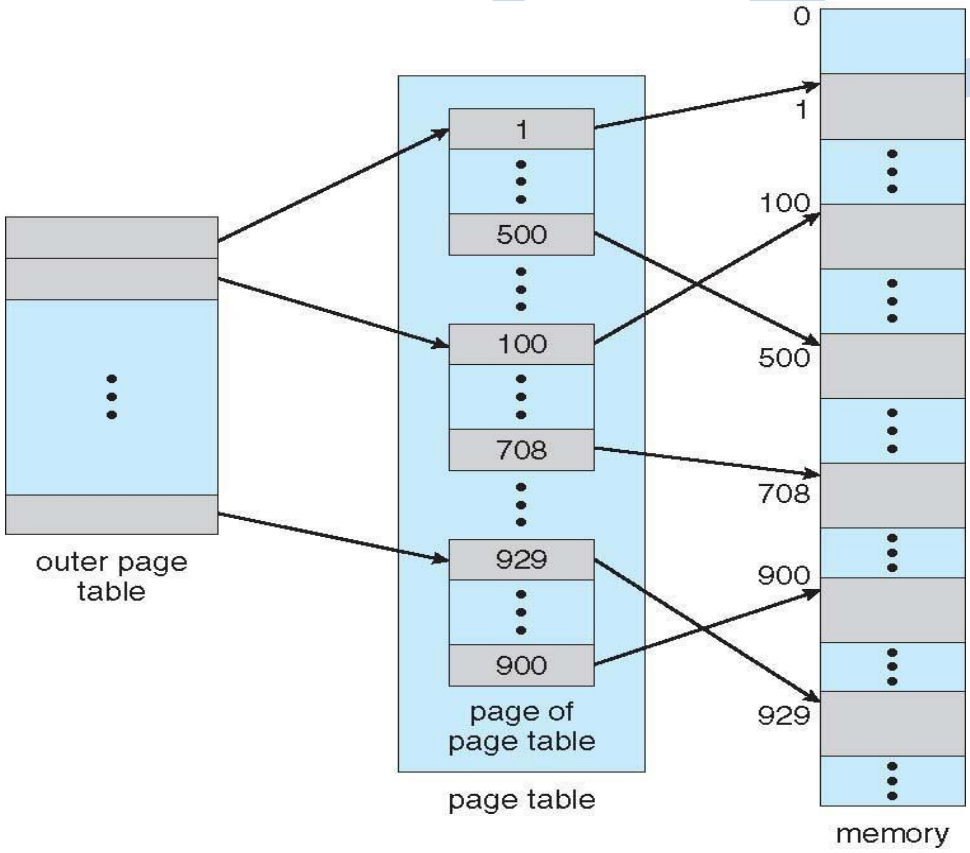
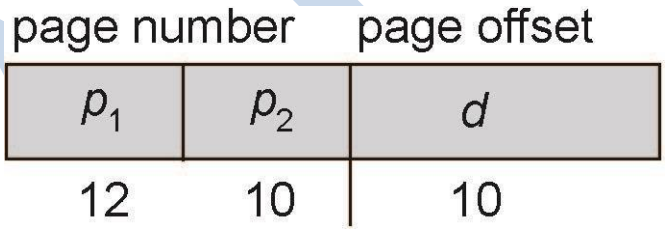


Fig. Two Level Page Table

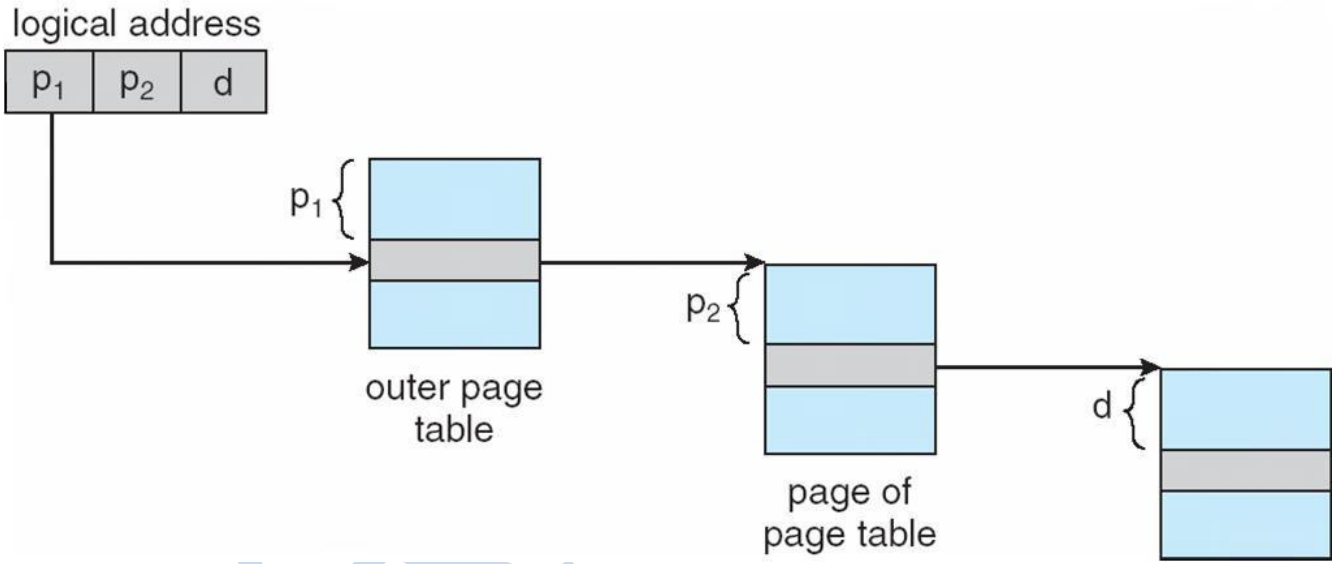
Two-Level Paging Example

- A logical address is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
  - a 12-bit page number
  - a 10-bit page offset
- Thus, a logical address is as follows:

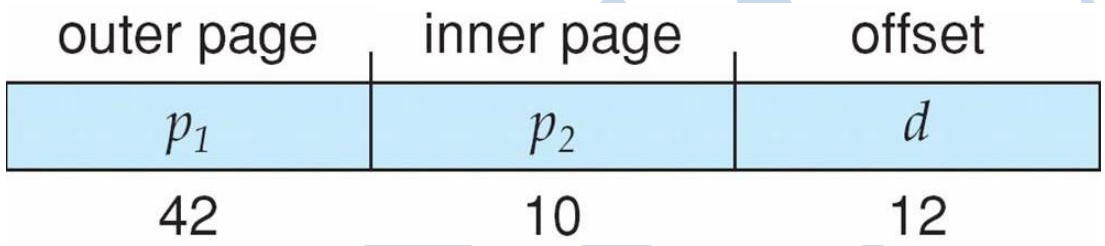


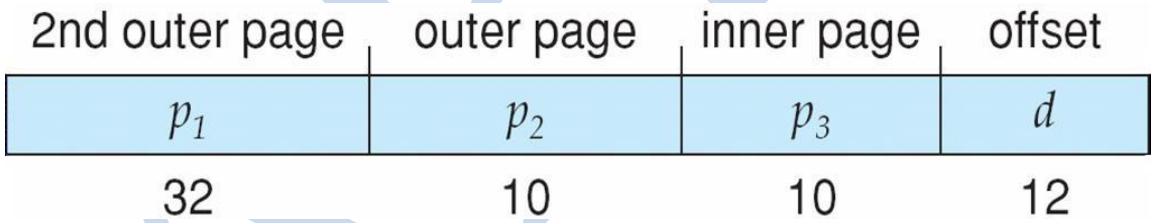
- where p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table
  - Known as forward-mapped page table

Address-Translation Scheme



Three-level Paging Scheme





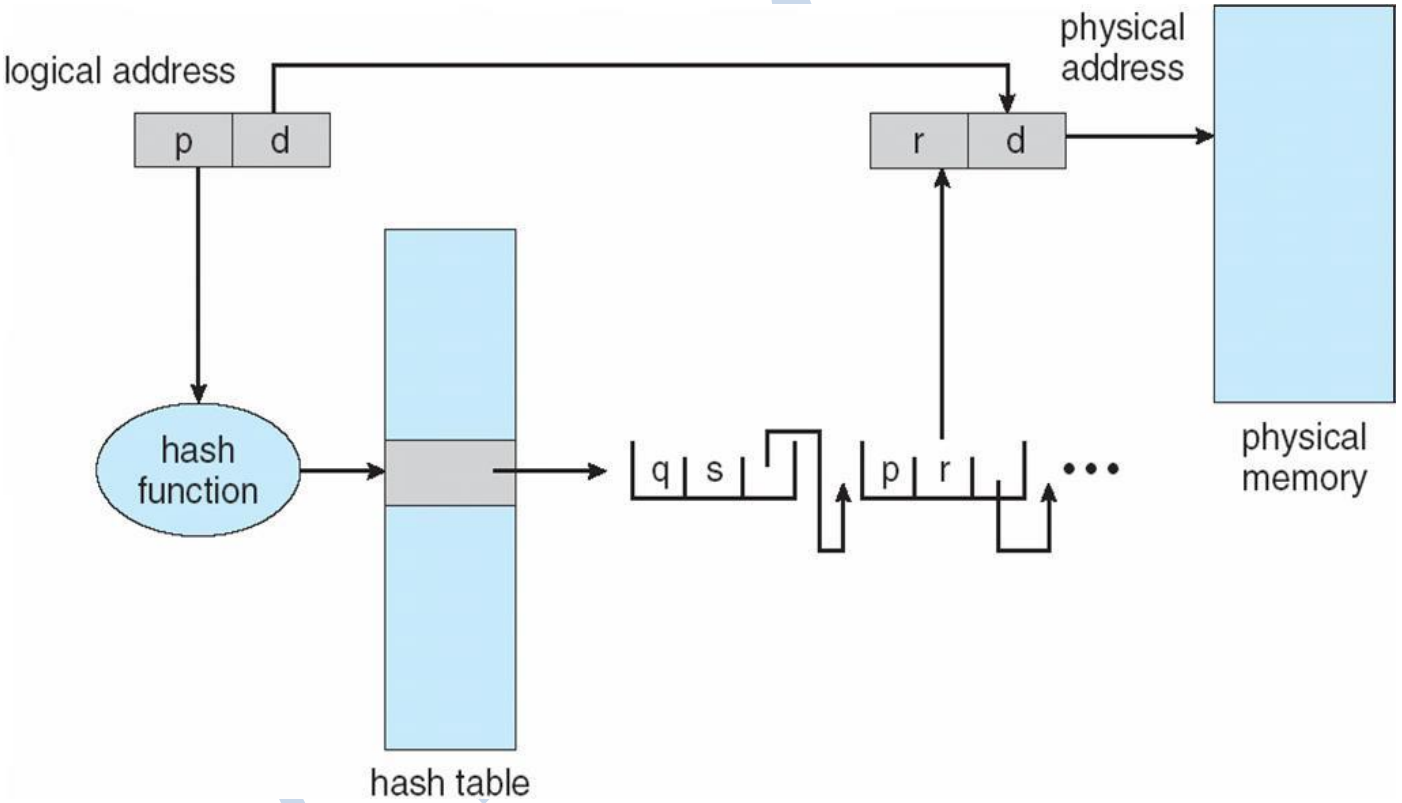
Hashed Page Tables

- To handle larger address spaces, we use **Hashed Page Tables** with the hash value being the virtual page number. Each entry in the hash table, contain a linked list of elements (To avoid collisions).

- Each element consists of
- (1) the virtual page number
- (2) the value of the mapped page frame
- (3) a pointer to next element in Linked List

APEC

Hashed Page Table



Disadvantages: Page table has million of entries

Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages.

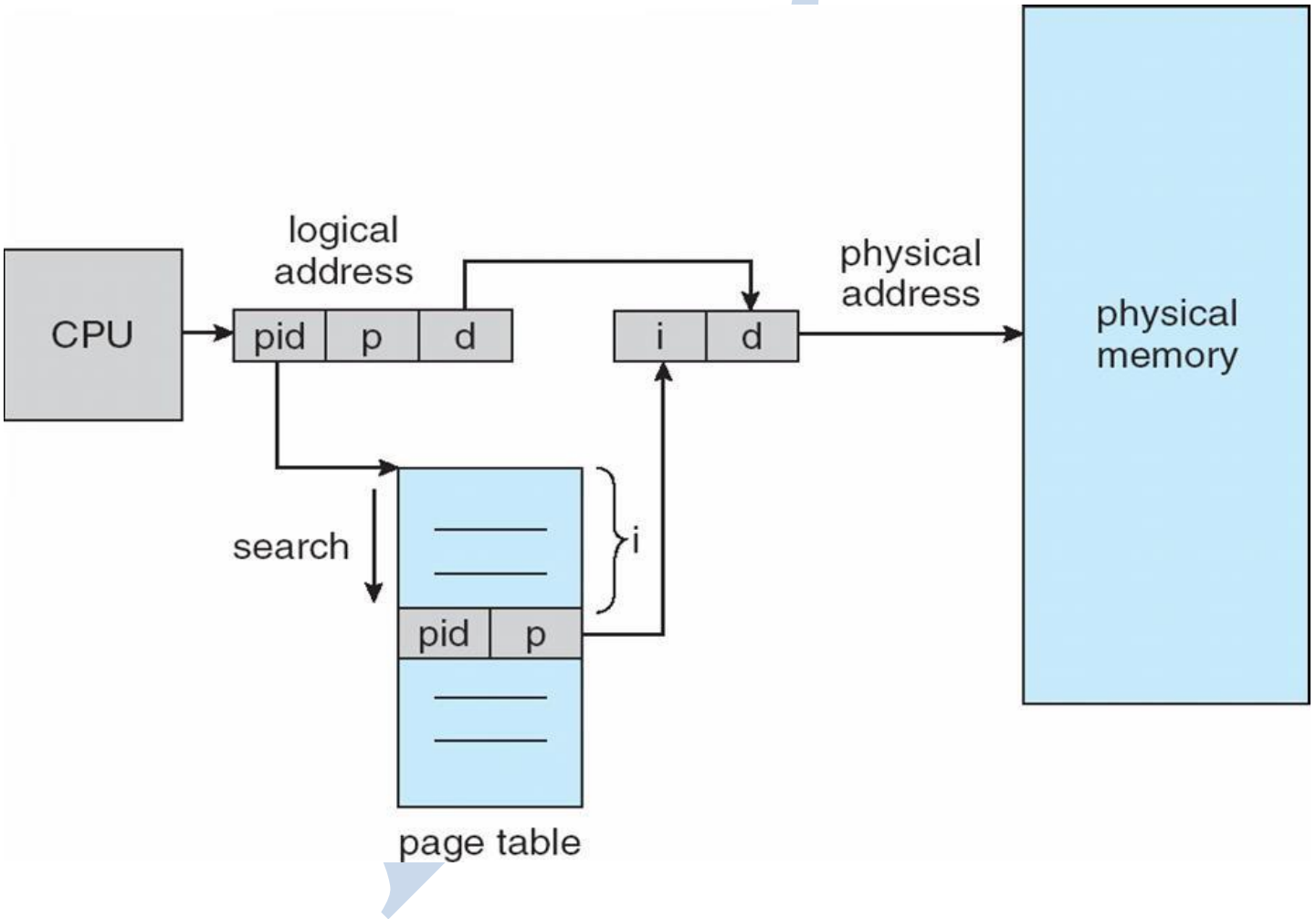
- The inverted page table has one entry for each page (or) frame. Each entry consists of a triple.

  <process id, page number, offset>

  - The inverted page table, check for the match.
  - If match found at entry i, then physical address <i,d> is generated.

APEC

Inverted Page Table



Segmentation

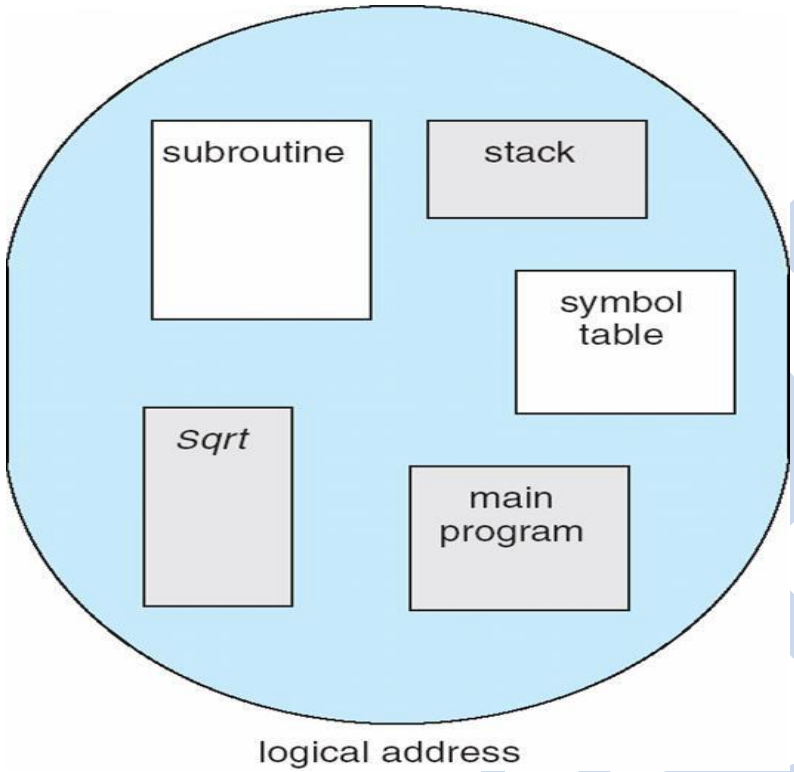
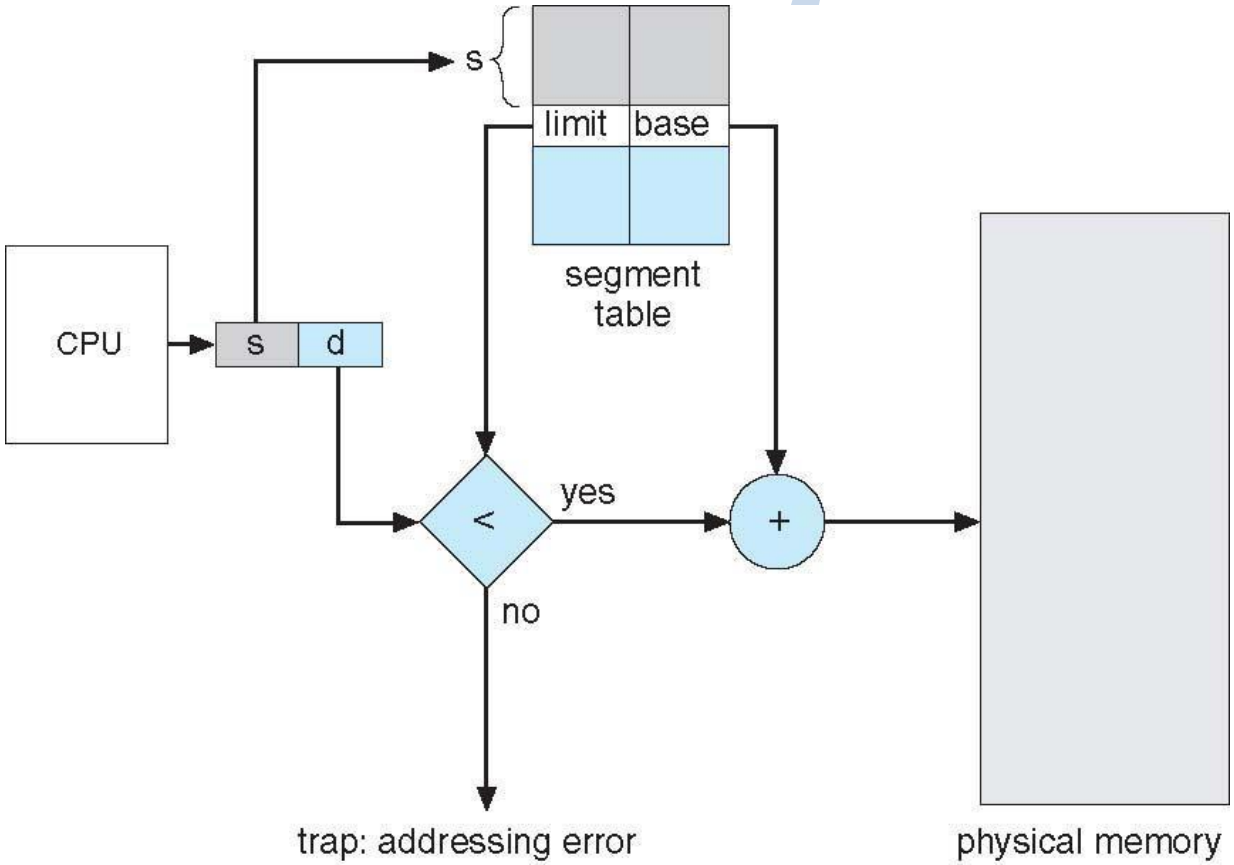


Fig. User's View of a Program

Segmentation:

- Treats logical address space as a collection of segments.
- Where each segment has a name & length.

Segmentation Hardware



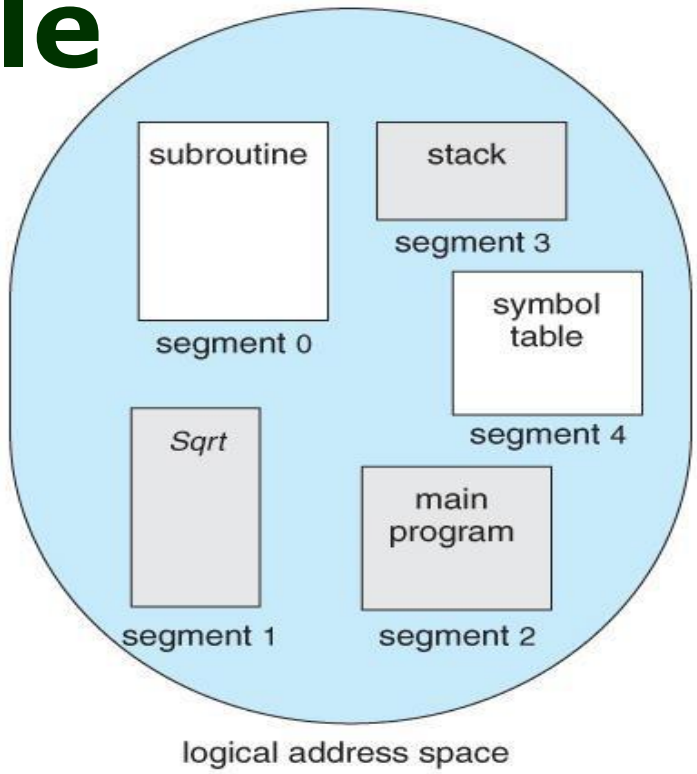
The logical address consists of <segment-number, offset>

Segmentation Contd...

- Logical address is mapped to physical address with the help of the segment table.
- Each entry in the segment table consists of segment base & segment limit.
  - Segment Base: starting physical address where the segment resides
  - Segment limit: specifies the length of the segment

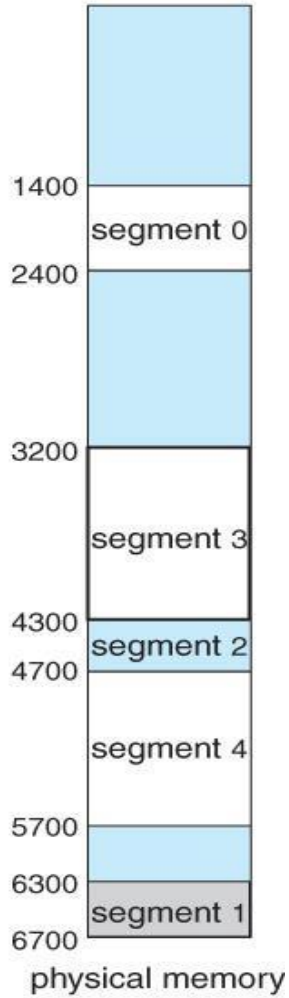
APEC

Example



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

segment table



Segment 2 is 400 bytes long and begins at location 4300.
Reference to 53rd byte is mapped to 4300 + 53 = 4353

Similarly, reference to 1222nd byte of segment 0 ➔ its trap to OS, because segment 0 limit = 1000 bytes only.

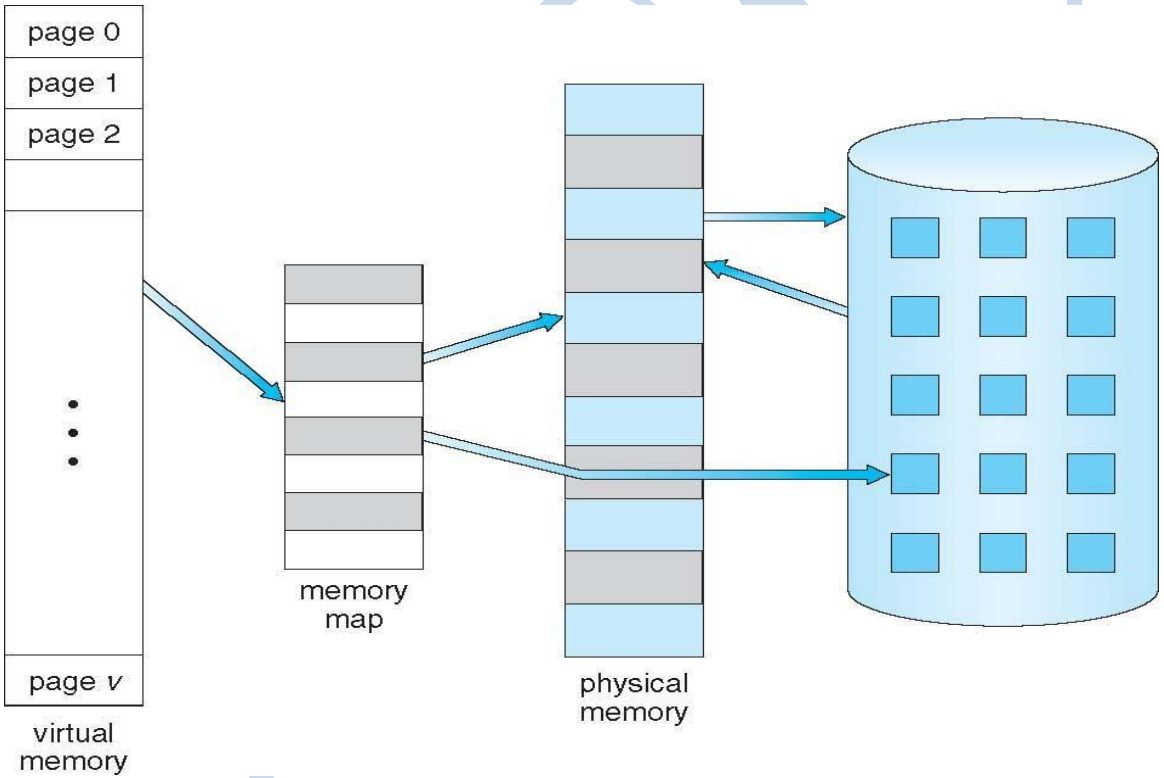
VIRTUAL MEMORY

APEC

Virtual Memory That is Larger Than Physical Memory

Virtual Memory:

It is a technique that allows the execution of processes that are not completely in memory.



When entire programs is not needed?

Array ➔ a[100]

But actually 10 elements are used.

Demand Paging

- Pages are loaded only when they are demanded. A demand paging is, where processes reside in secondary memory (usually the Disk).
- Lazy swapper – never swaps a page into memory unless its needed.

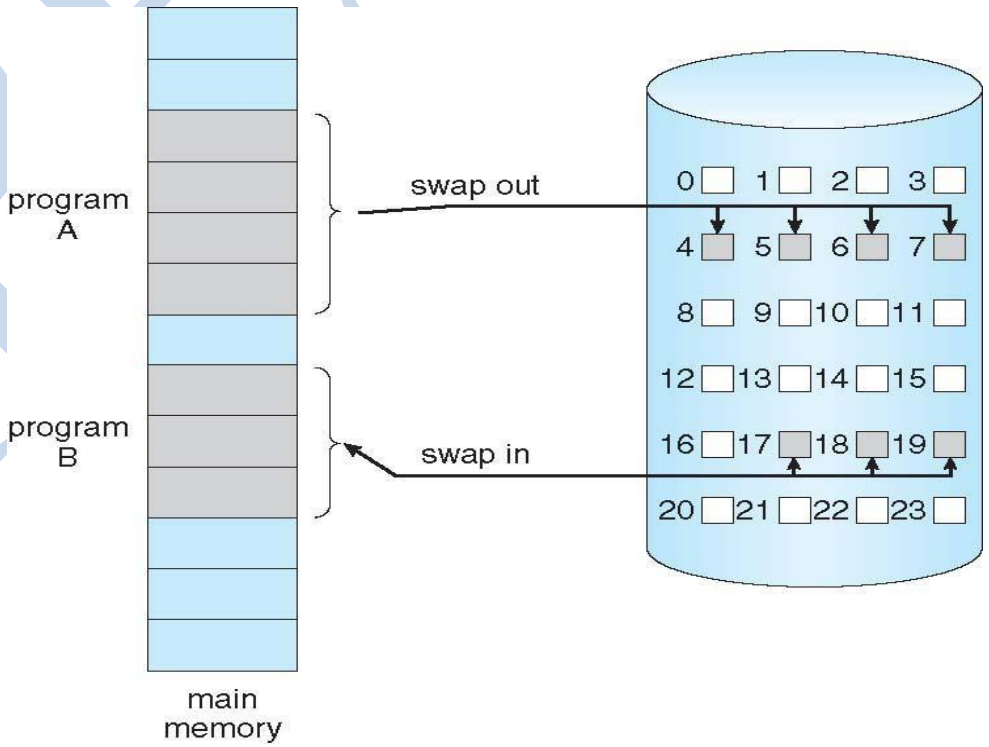


Fig. Transfer of Paged memory to contiguous disk space

NOTE:
Swapper: swap the entire process
Pager : concerned with individual pages.

Lazy swapper = pager

Pure demand paging: starts a process with no pages in memory.

Basic Concepts

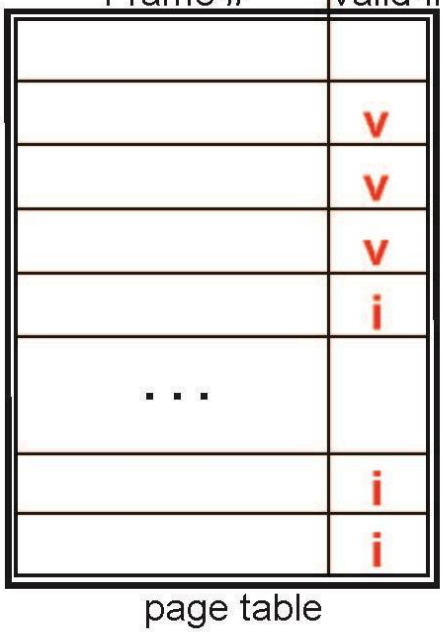
- When a process needs to be swapped in, the pager knows which pages will be used.
- Only that particular pages are brought into memory.

- Advantages:
  - Memory is less used.
  - Swap time is reduced.

APEC

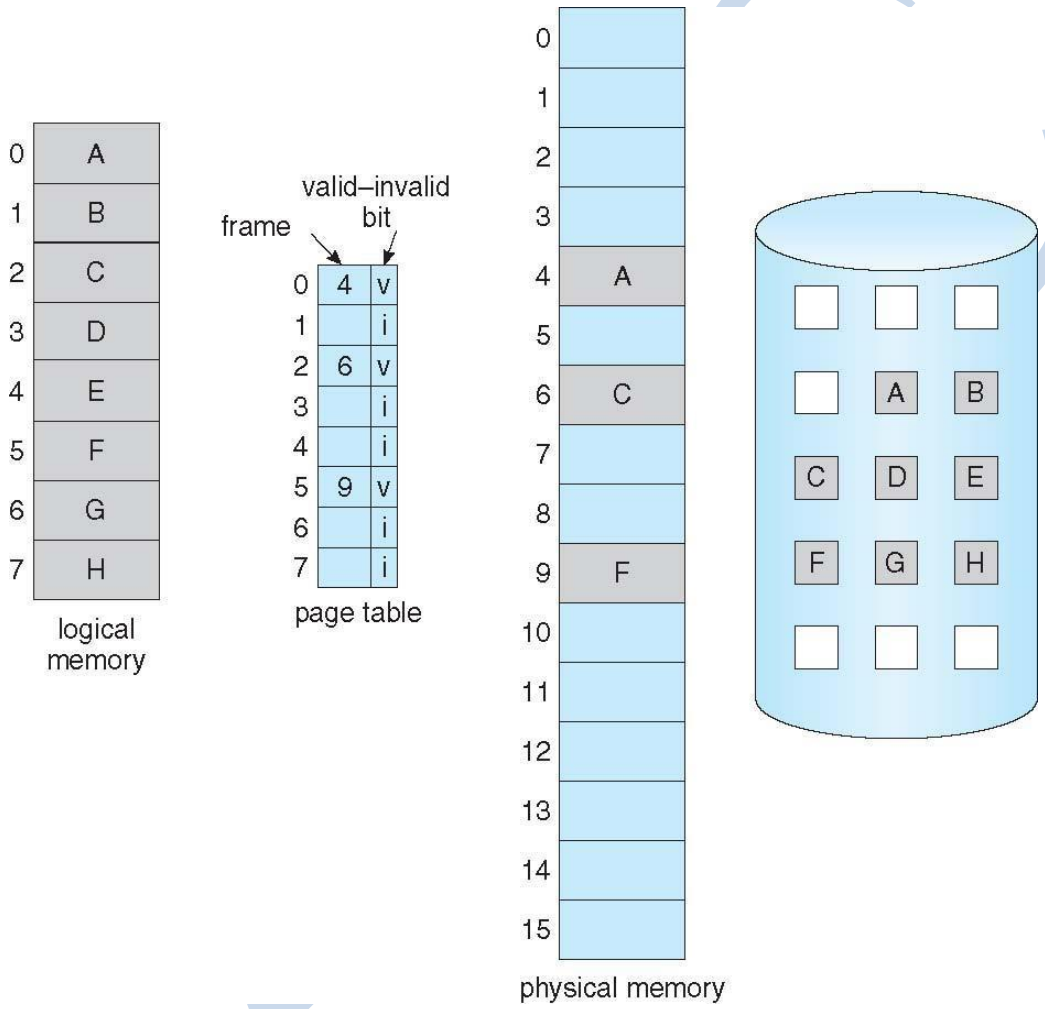
Valid-Invalid Bit

- With each page table entry a valid–invalid bit is associated
  (**v** ⇒ in-memory – **memory resident**, **i** ⇒ not-in-memory)
- Initially vali[illegible] **i** on all entries
- Example of [illegible]ot:



- During MMU address translation, if valid–invalid bit in page table entry is **i** ⇒ page fault

Page Table When Some Pages Are Not in Main Memory



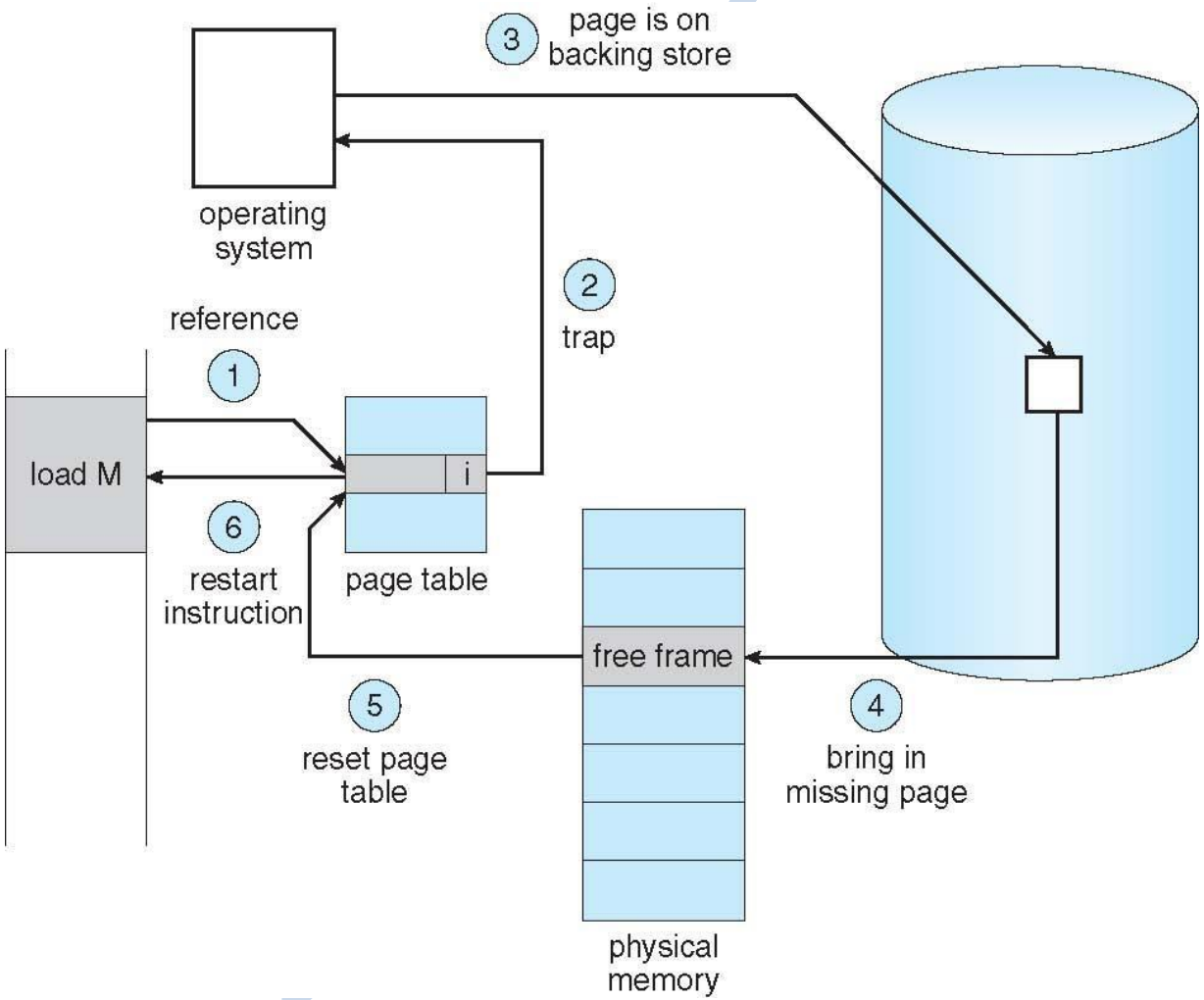
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:
  page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Find free frame
3. Swap page into frame via scheduled disk operation
4. Reset tables to indicate page now in memory
   Set validation bit = **v**
5. Restart the instruction that caused the page fault

APEC

Steps in Handling a Page Fault



APEC