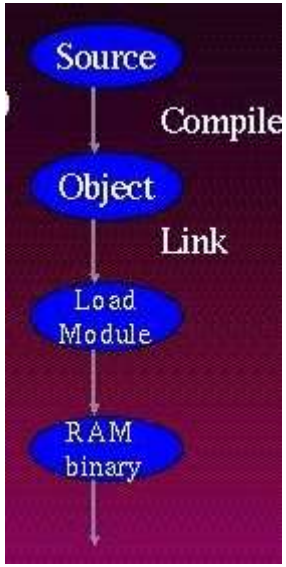
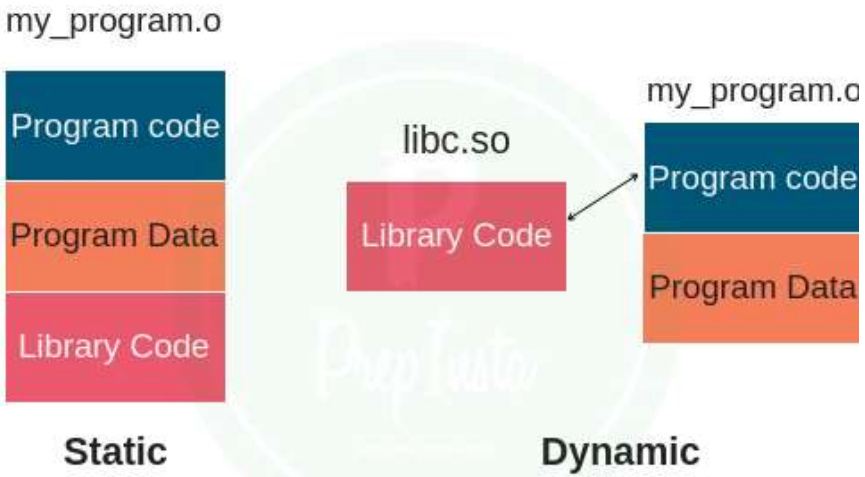
SNS COLLEGE OF TECHNOLOGY

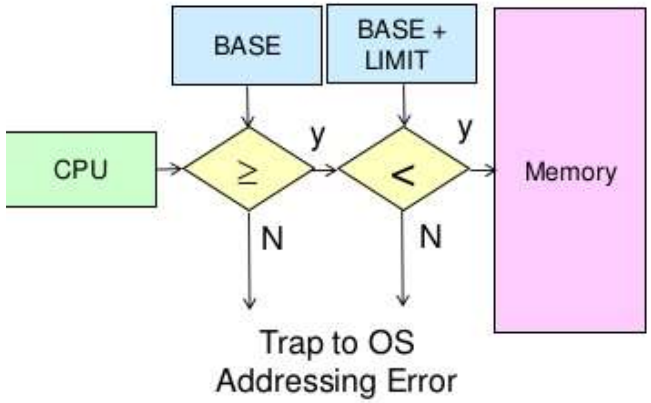
(Autonomous)

COIMBATORE-35

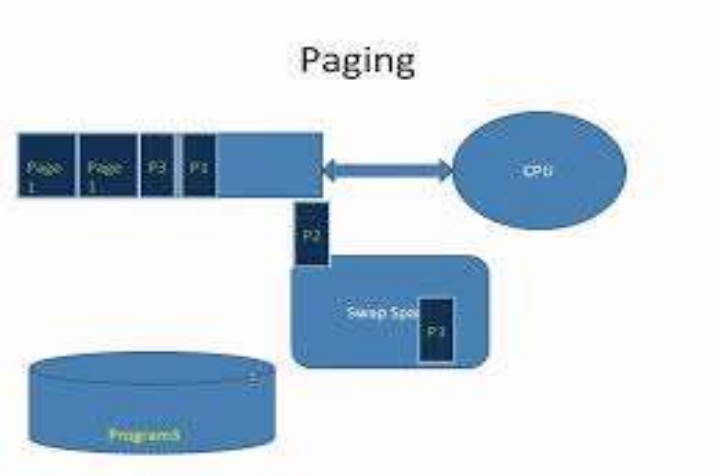
Memory Management Paging







Paging



Paging Basic Method

Hardware Support

Protection

Shared Pages

Structure of Page Table

Paging

- Memory management technique
- Divides a process's virtual memory into equal sized blocks called pages
- Physical memory into frames
- Allowing for non contiguous allocation and enabling virtual memory
- Page table used to map logical(virtual)
- Addresses to physical addresses

Page Fault-Page is not currently in memory

Paging

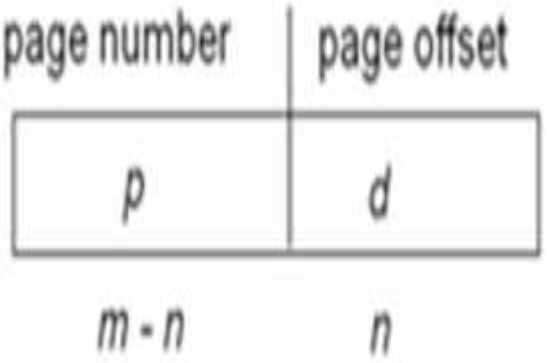
Physical address space of a process can be noncontiguous

- Avoids external fragmentation
- Avoids problem of varying sized memory chunks

- Divide physical memory into fixed-sized blocks called **frames**
  - Size is power of 2, between 512 bytes and 16 Mbytes
- Divide logical memory into blocks of same size called **pages**
- **page table** to translate logical to physical addresses
- **Disadvantage**
- Still have Internal fragmentation

Paging

- Permits the Physical Address space of a process to be noncontiguous
- Address Translation Scheme

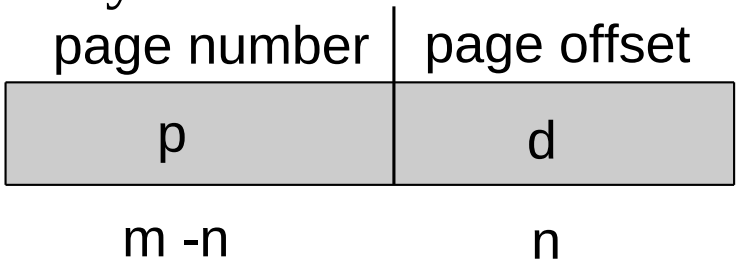


PAGE TABLE ENTRY

Frame No	Valid/Invalid	Protection(R/W/X)	Reference	Caching	Dirty

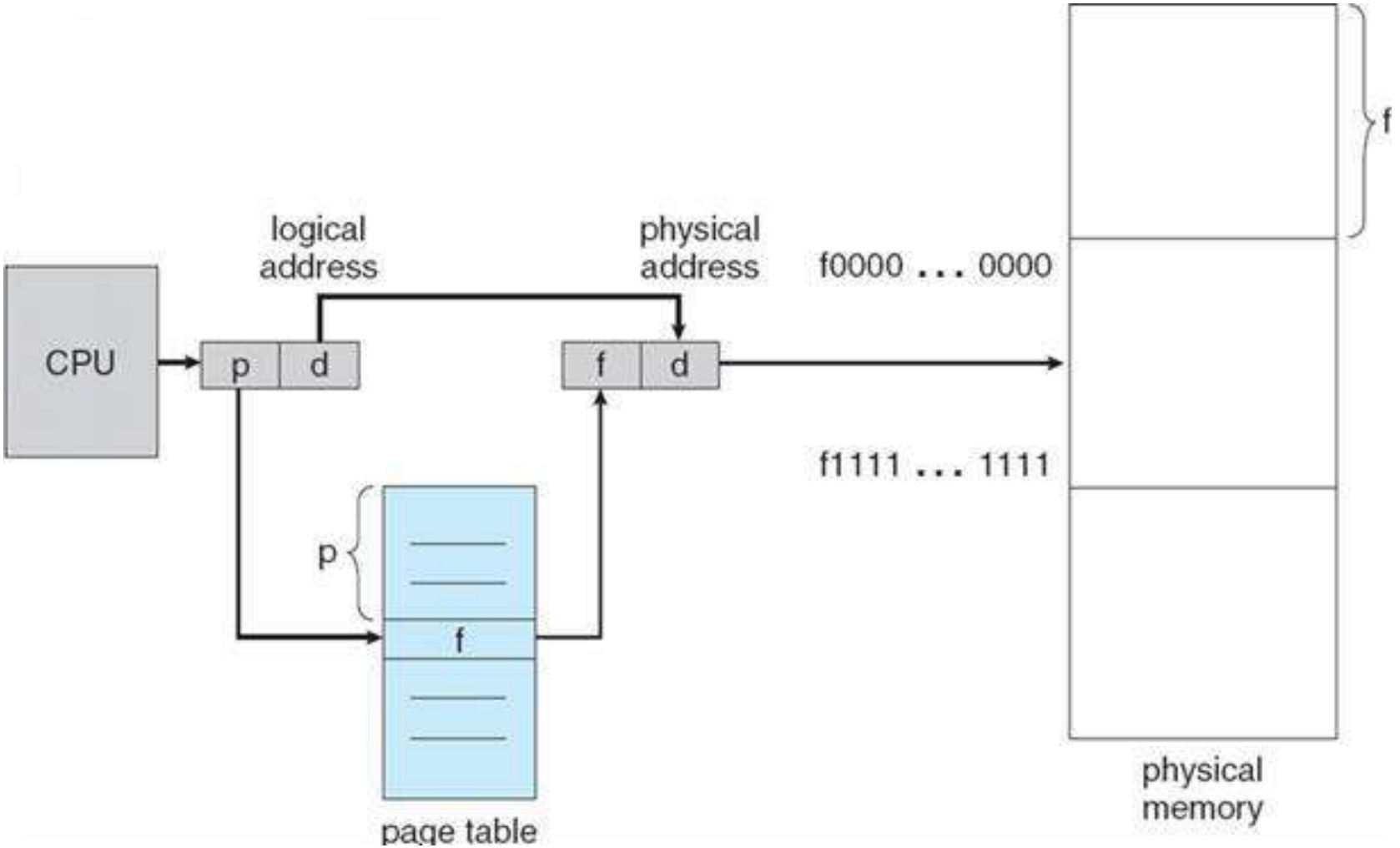
Address Translation Scheme

- Address generated by CPU is divided into:
  - Page number (p) – used as an index into a page table which contains base address of each page in physical memory
  - Page offset (d) – combined with base address to define the physical memory address that is sent to the memory unit

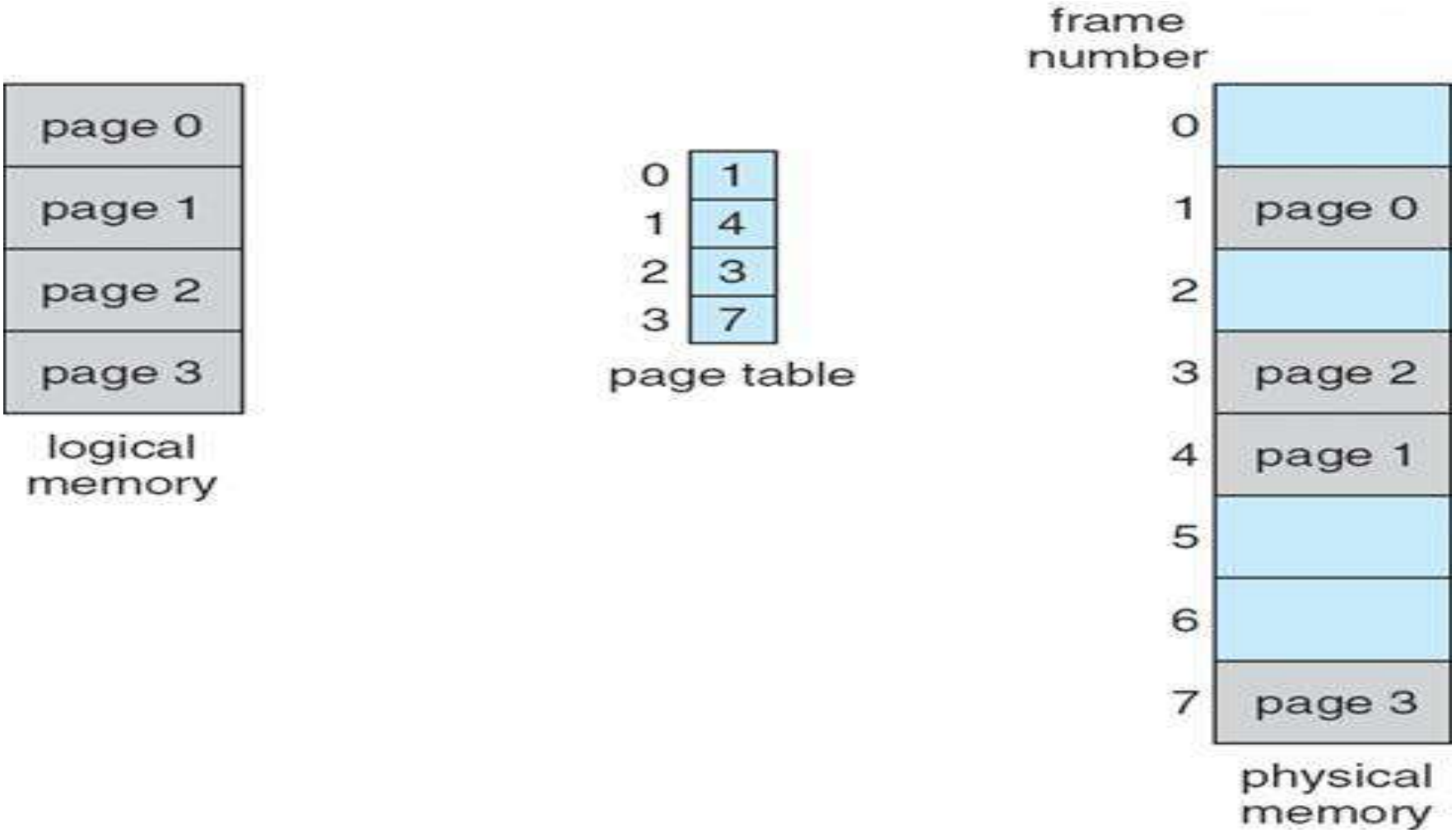


  - For given logical address space 2^m and page size 2^n

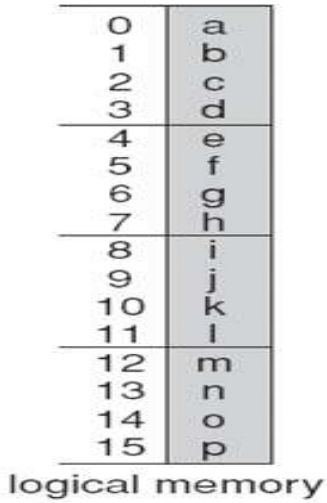
Paging Hardware

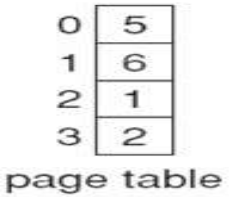


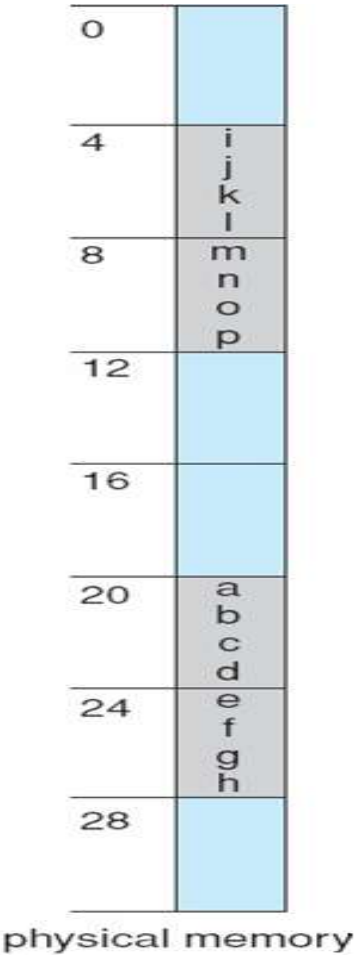
Paging model of logical and Physical Memory



Paging Example

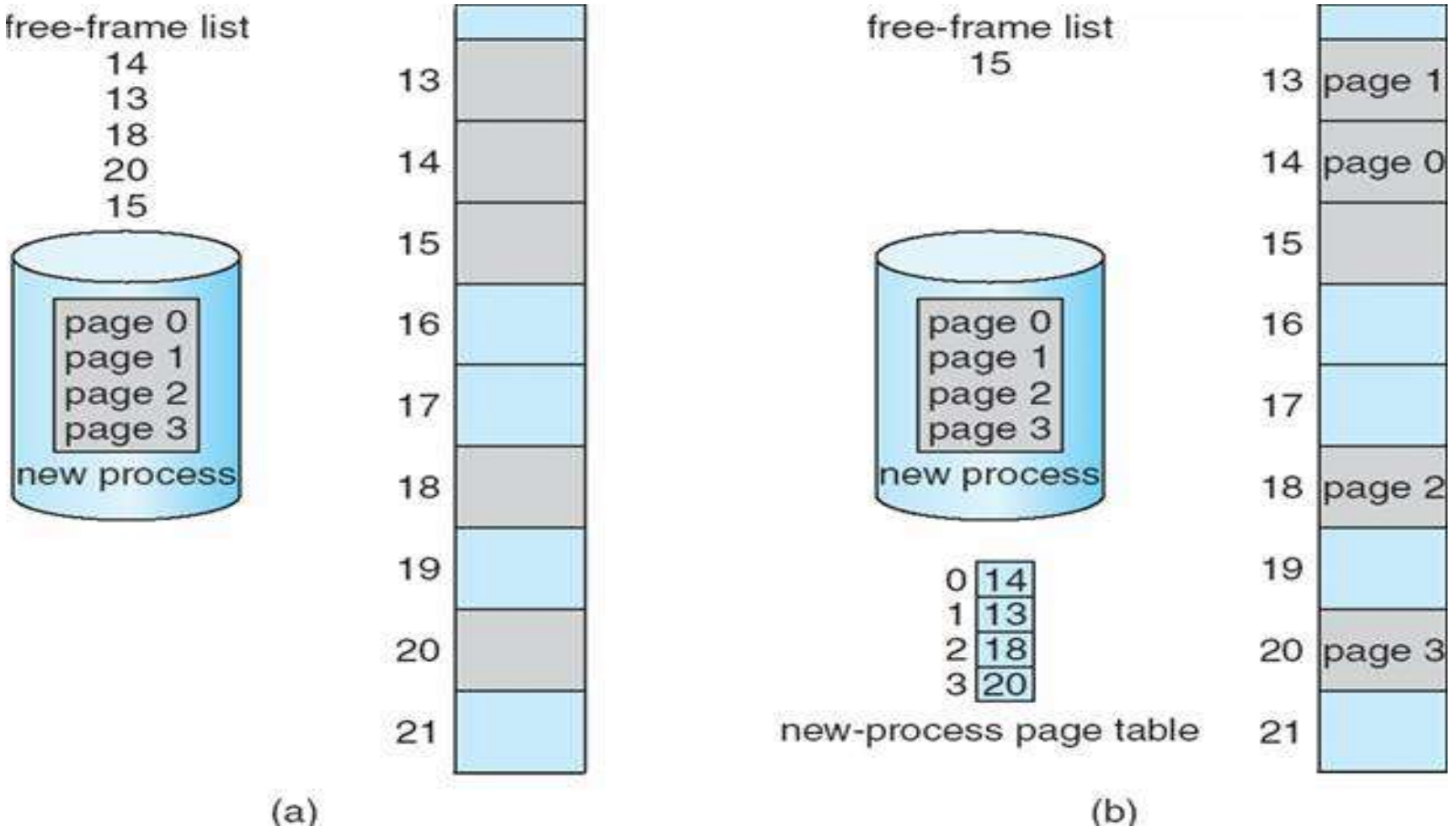






***n*=2 and *m*=4 32-byte memory and 4-byte pages**

Paging-Free frames



Implementation of Page Table

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

In this scheme every data/instruction access requires two memory accesses

- One for the page table and one for the data / instruction

➢ The two memory access problem can be solved by **associative memory or translation look-aside buffers (TLBs)**

Associative Memory

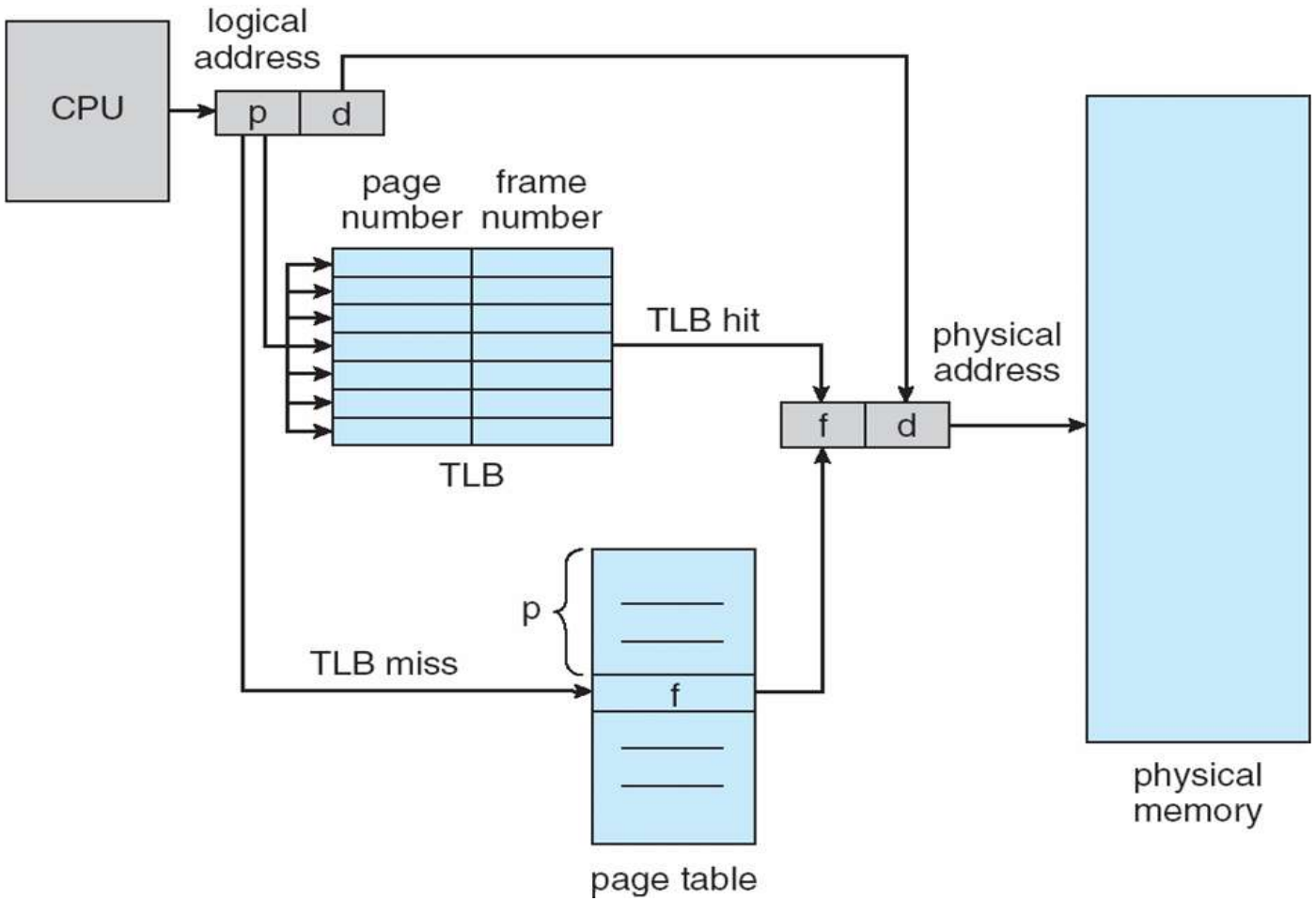
- Associative memory – parallel search

Page #	Frame #

Address translation (A′, A′′)

- If A′ is in associative register, get frame # out.
- Otherwise get frame # from page table in memory

Paging Hardware With TLB



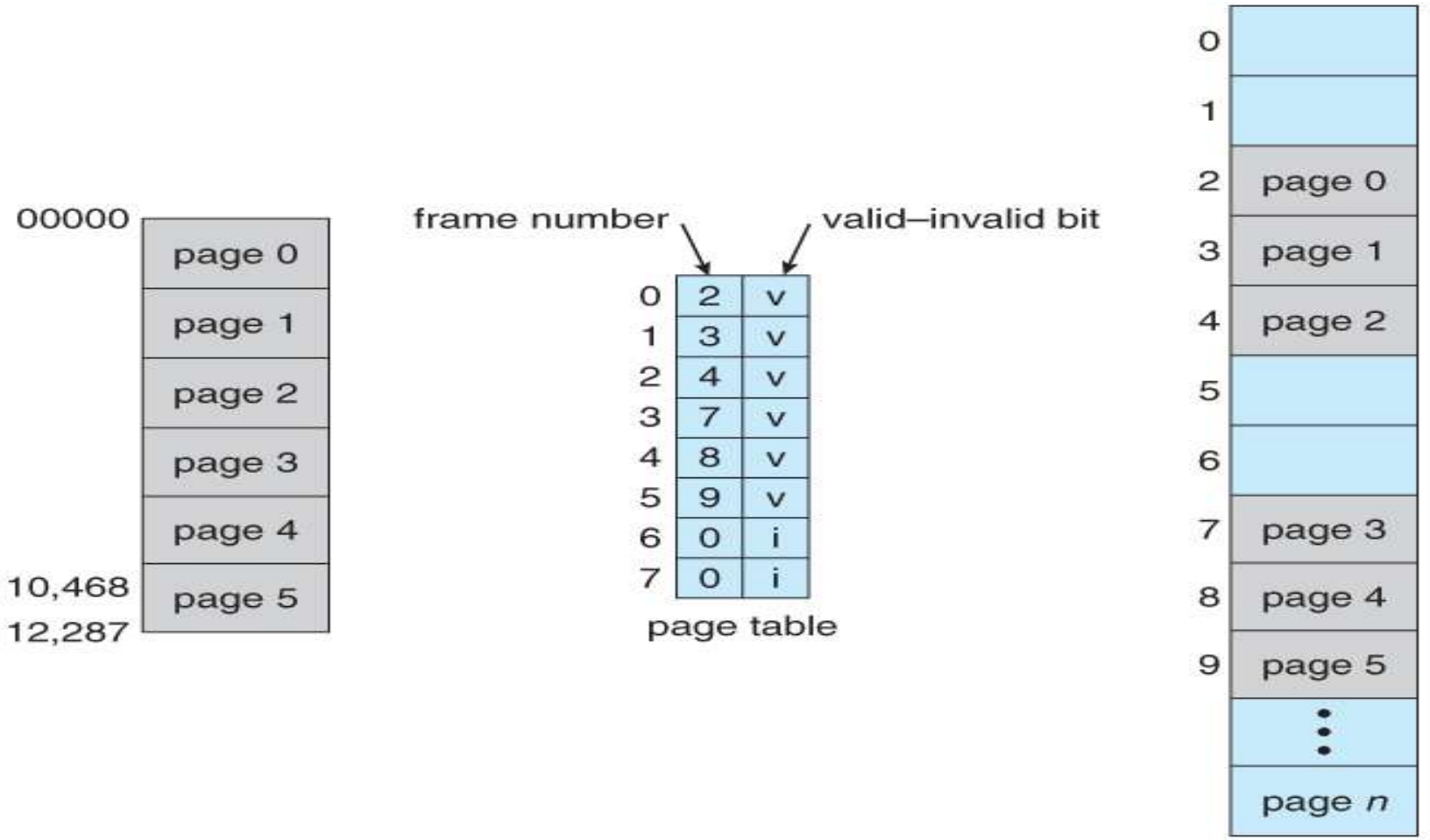
Memory Protection

Memory protection implemented by associating protection bit

- Valid-invalid bit attached to each entry in the page table:
- "valid" indicates that the associated page is in the process' logical address space, and is thus a legal page
- "invalid" indicates that the page is not in the process' logical address space
- Or use page-table length register (PTLR)

Paging –Protection

Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

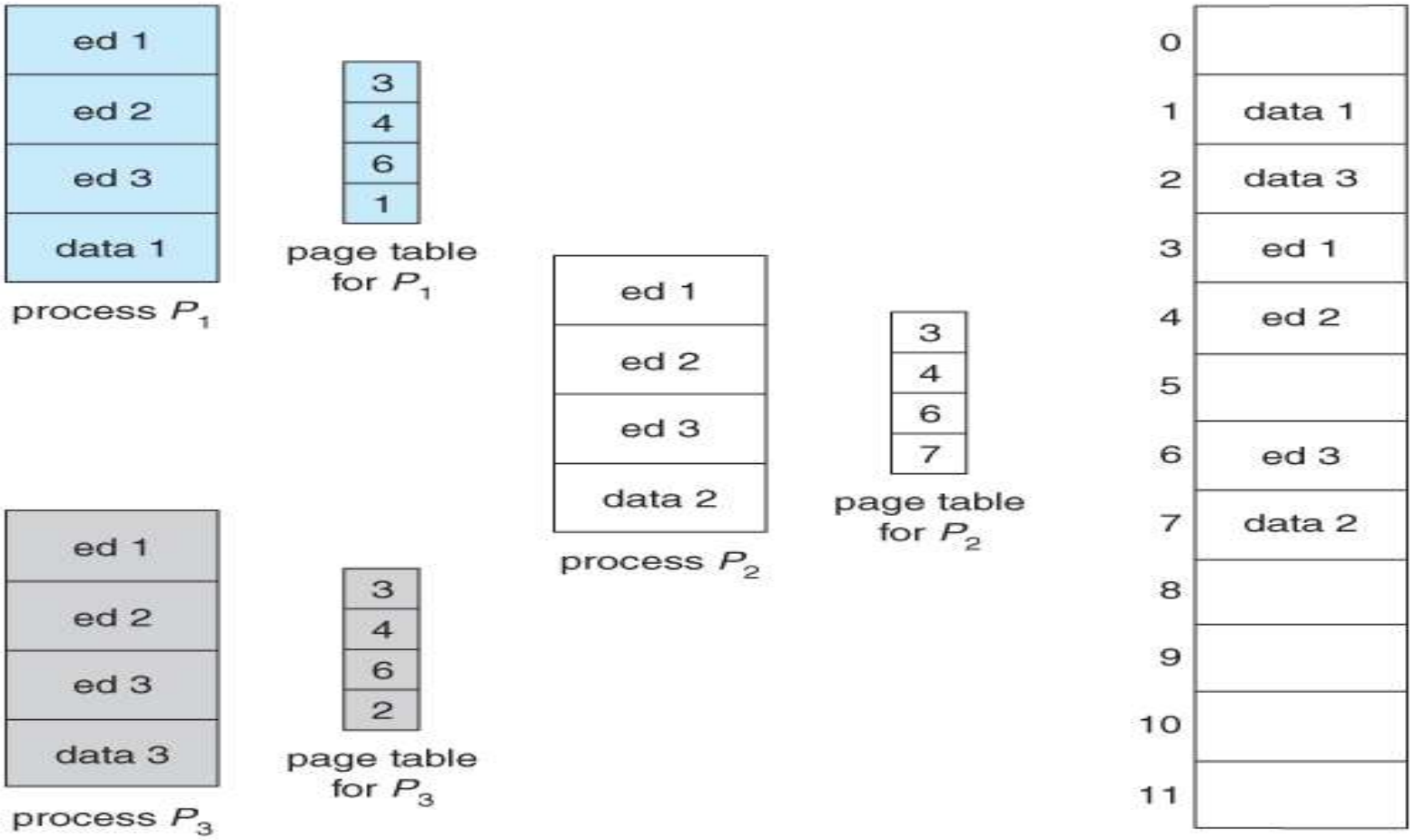
Shared code

- One copy of read-only **(reentrant)** code shared among processes (i.e., text editors, compilers, window systems)
- Similar to multiple threads sharing the same process space
- Also useful for inter process communication if sharing of read-write pages is allowed

Private code and data

- Each process keeps a separate copy of the code and data
- The pages for the private code and data can appear anywhere in the logical address space

Paging –Shared Pages



Problems

Q] Consider a logical address space of 8 pages of 1024 words each, mapped onto a physical memory of 32 frames. How many bits are there in physical address and logical address respectively?

(a) 5, 3 (b) 10, 10 (c) 15, 13 (d) 15, 15.

Logical Address Space Size = 8×1024

$= 2^3 \times 2^{10} = 2^{13}$

$\rightarrow$ 13 bits.

Physical Address Space Size = 32 frames $\times$ Size of 1 Page (or 1 frame)

$= 32 \times 1024$

$= 2^5 \times 2^{10} = 2^{15}$

$\rightarrow$ 15 bits

Problems

Consider a system which has LA = 7 bits, PA = 6 bits, page size = 8 words, then calculate number of pages and number of frames.

Answer: Total bits of LA = 7 bits, which means it is equal to 2^7

The page size = 8 words which means 2^3 so we can say it is of 3 bits
Therefore, Page No. = 4 which we can say is 2^4 and then total number of pages = $2^4 = 16$
While total number of bits required to respond total number of page = 4 bits.
PA = 6 bits, total size of PA = $2^6 = 64$
Now, we know that frame offset = Page offset
Number of frames = $2^3 = 8$
Total number of entries in page table = number of pages
Total number of entries = 16
Total size of page table = 24 x 8

Problems-Effective Access time-No Page fault

If main memory access time is 400ms, TLB access time is 50ms,considering a TLB hit as 90%, What will be the overall Effective Access Time(EAT)?

Formula

EAT = (TLB Hit Rate) * (TLB Access Time + Main Memory Access Time) + (1 - TLB Hit Rate) * (TLB Access Time + Main Memory Access Time + Main Memory Access Time)

The overall Effective Access Time (EAT) is calculated as 0.9 * (50 + 400) + 0.1 * (50 + 400 + 400) = 490ms.

1.If main memory access time is 50ms, TLB access time is 10ms,considering a TLB hit as 90% and there is no page fault, What will be the overall Effective Access Time(EAT)?

.

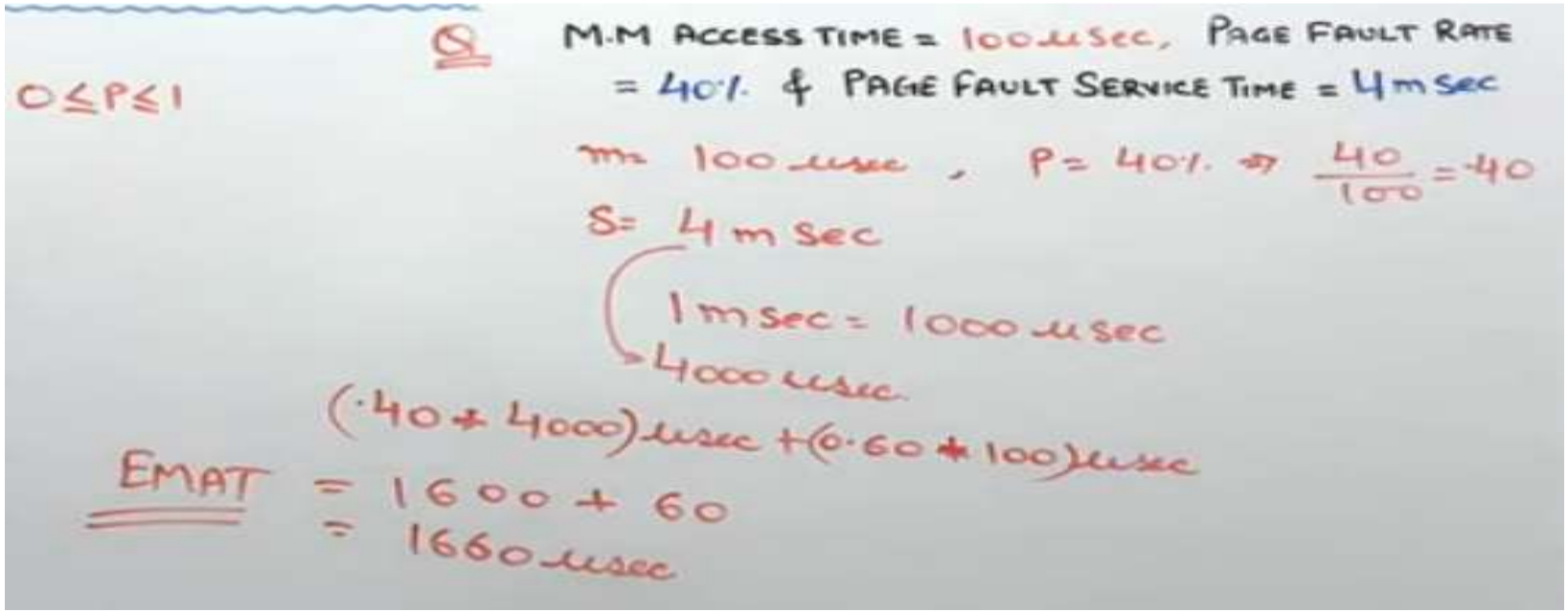
Problems-Effective Access time

Main memory access time=‘m’

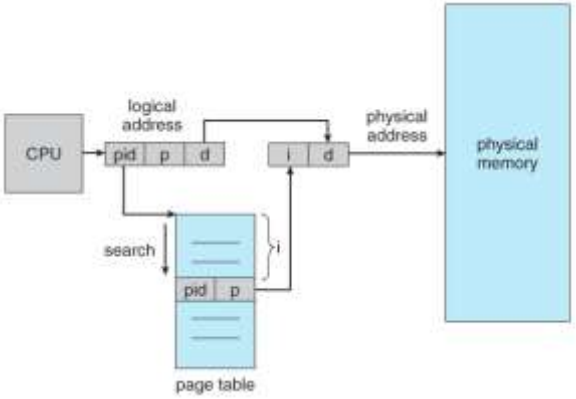
Page Fault Rate=p

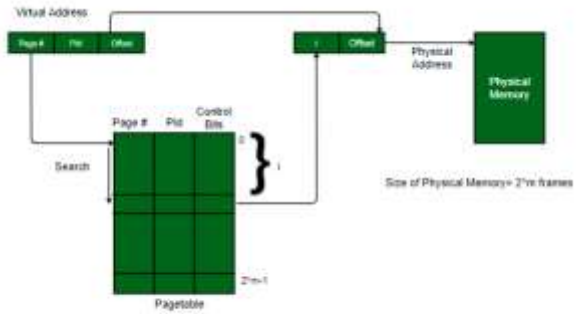
Page fault service time=s

Effective memory access time=p*s+(1-p)*m

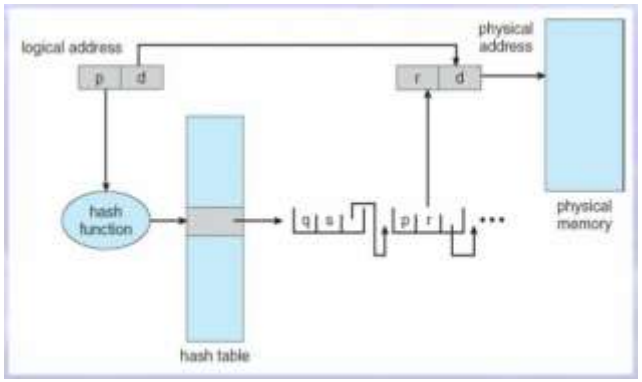


Structure of the Page Table





Inverted PageTable



Structure of the Page Table

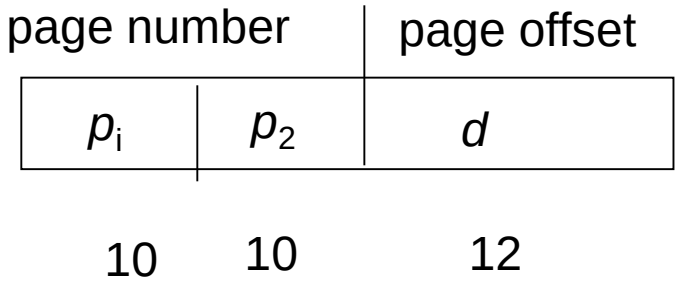
- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

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

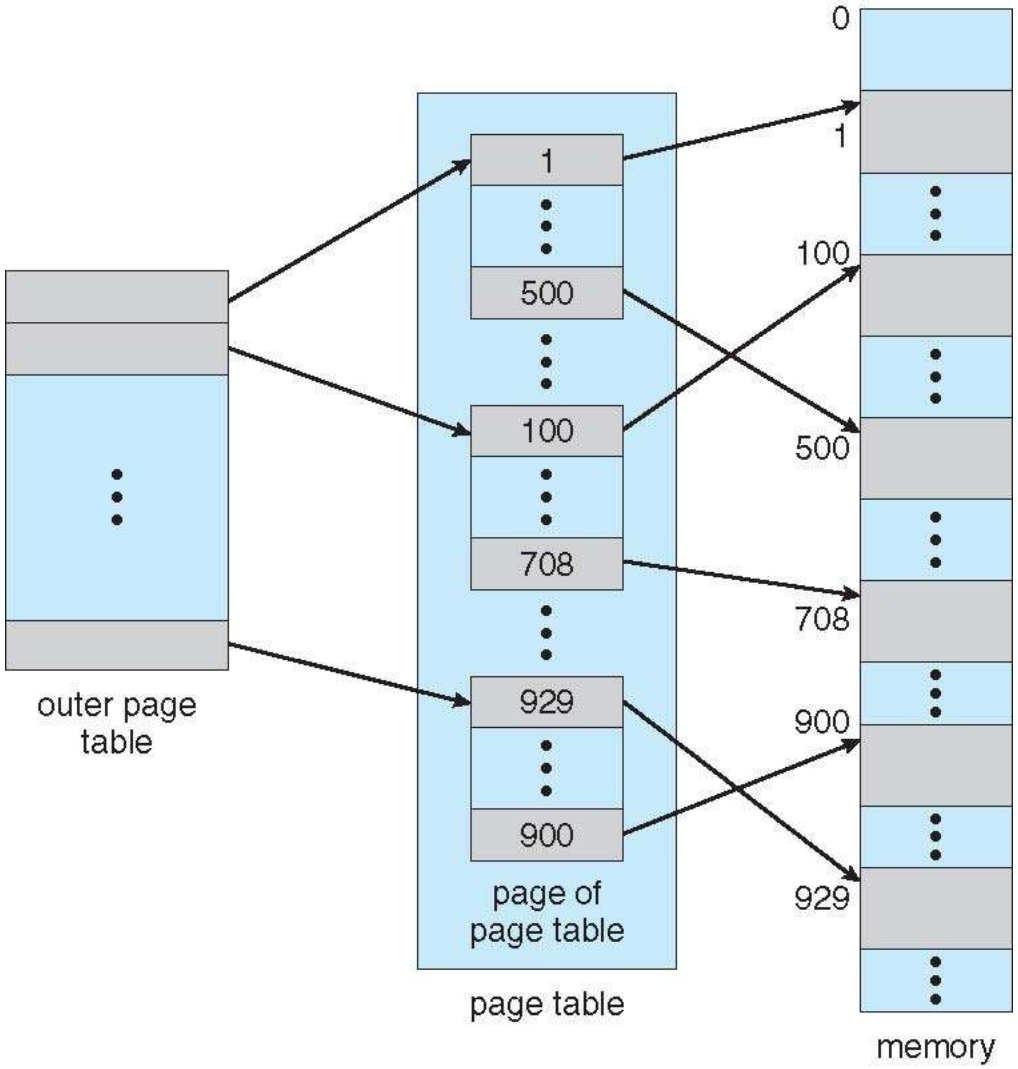
Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:
  - a page number consisting of 20 bits.
  - a page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - a 10-bit page number.
  - a 10-bit page offset.
- Thus, a logical address is as follows:

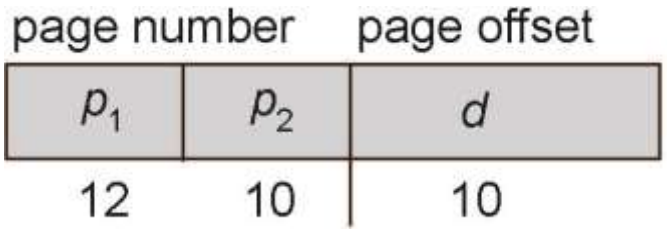


where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.

Two-Level Page-Table Scheme

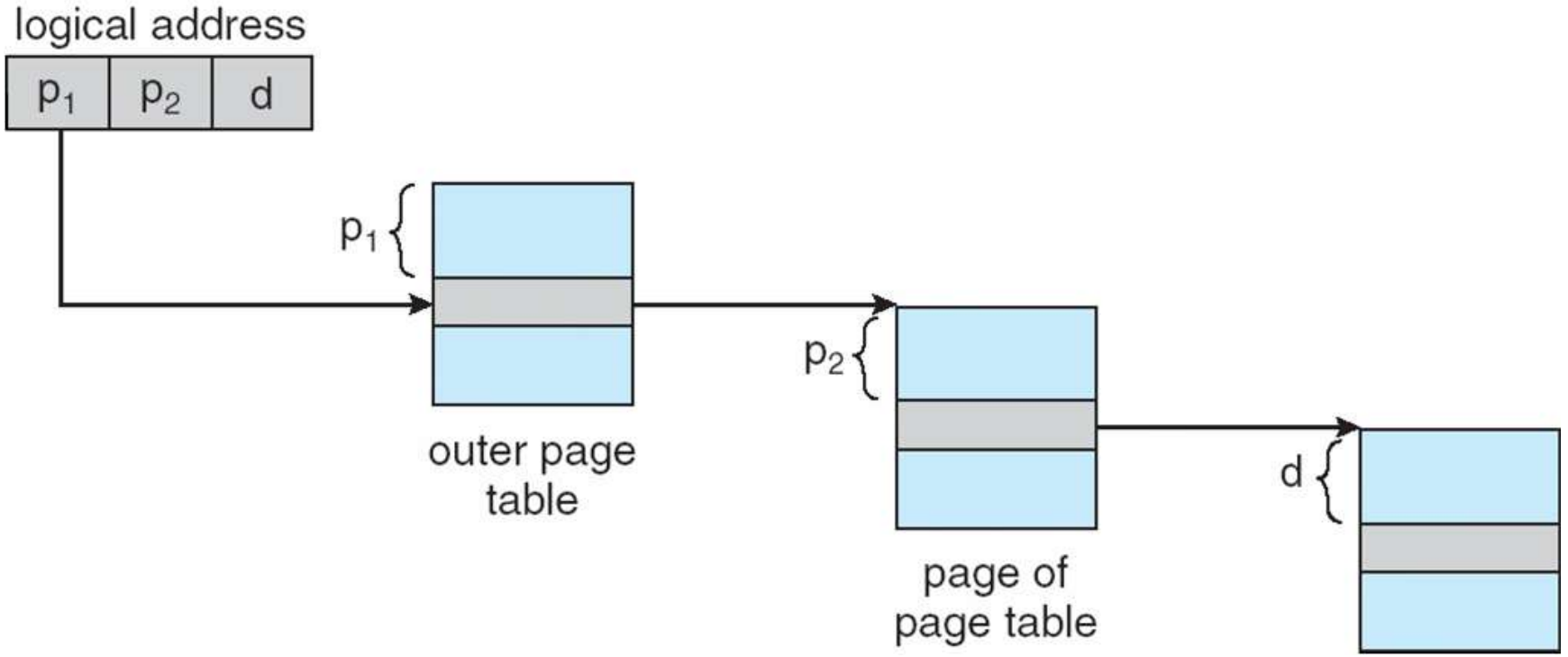


- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits
- A logical address (on 32-bit machine with 1K page size) is divided into:
  - a page number consisting of 22 bits
  - a page offset consisting of 10 bits



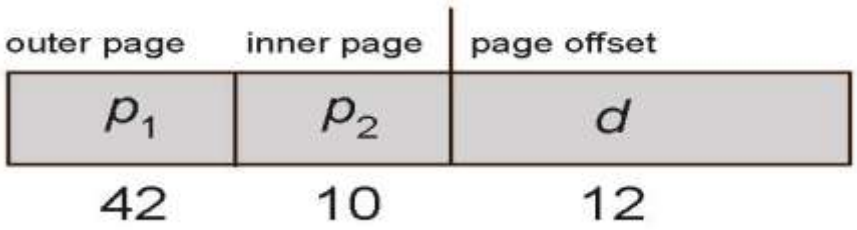
- p_1 is an index into the outer page table, and p_2 is the displacement within the page of the inner page table known as **forward-mapped page table**

Address-Translation Scheme

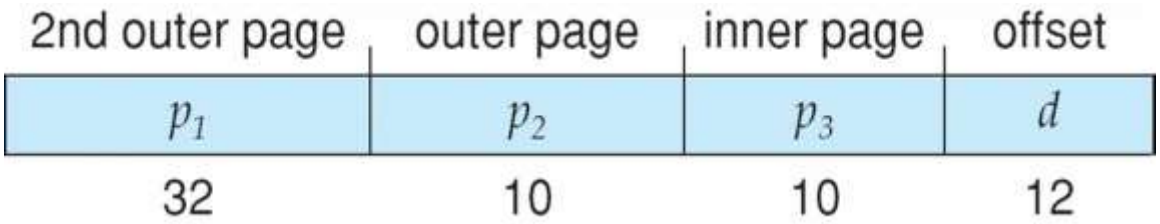


64-bit Logical Address Space

- Two-level paging scheme not sufficient
- Outer page table has 2^{42} entries or 2^{44} bytes



- Three-level Paging Scheme



Hashed Page Tables

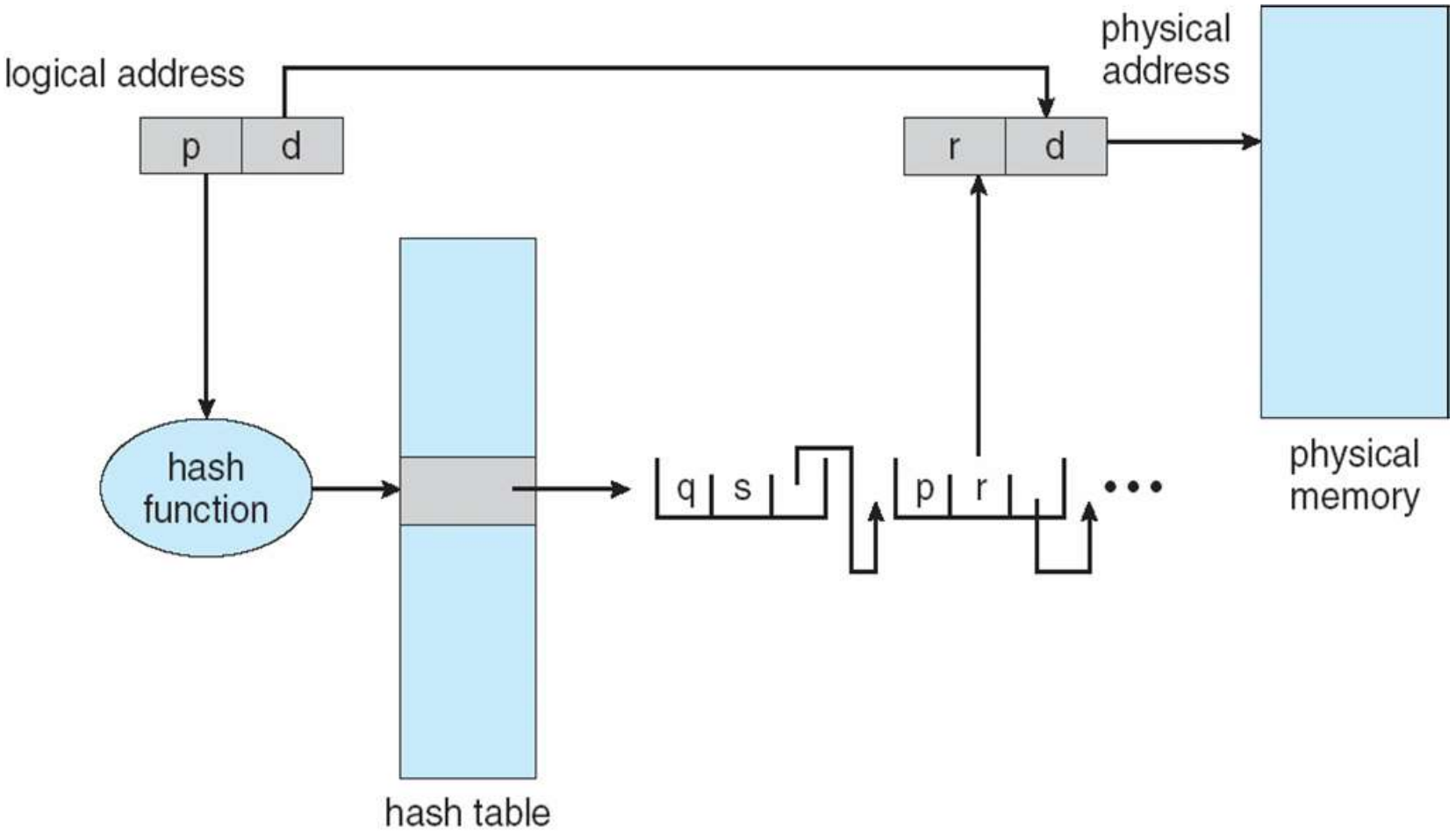
- Virtual page number is **hashed** into a page table
- Each element contains

  (1) the virtual page number

  (2) the value of the mapped page frame

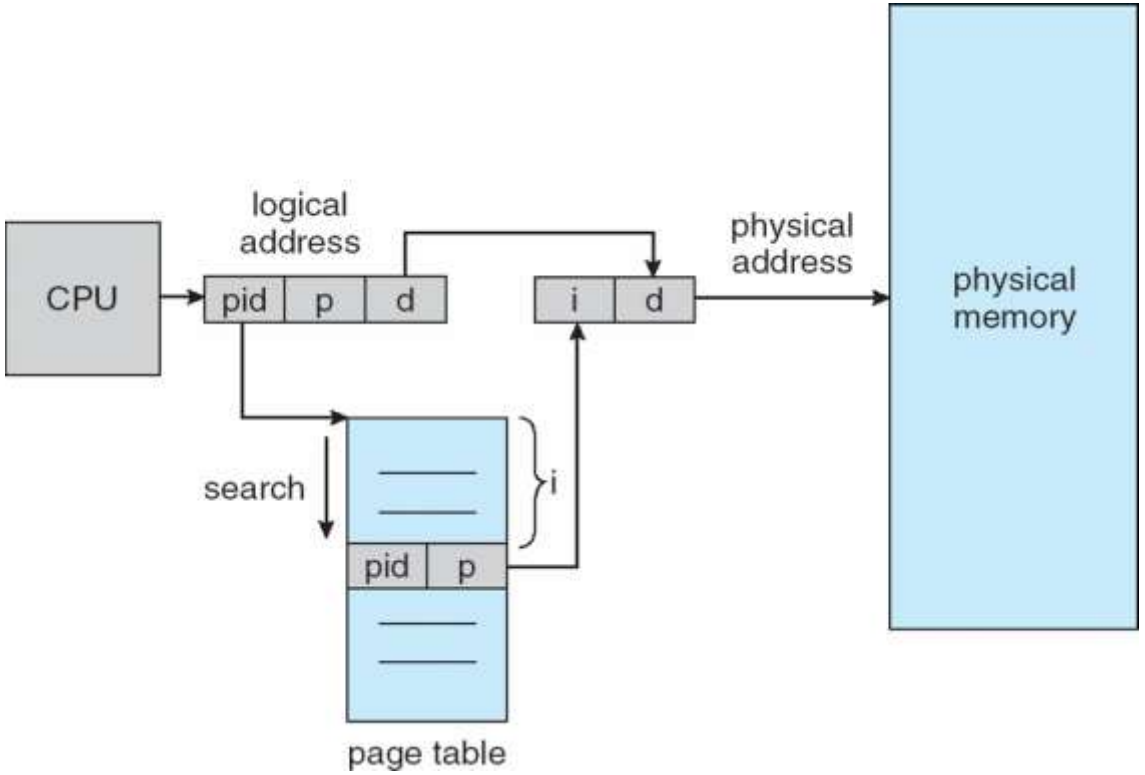
  (3) a pointer to the next element
- Virtual page numbers are compared in this chain searching for a match
  - If a match is found, the corresponding physical frame is extracted

Hashed Page Table



Inverted Page Table

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



Summarization