PAGING

- It is a memory management scheme that permits the physical address space of a process to be noncontiguous.
- It avoids the considerable problem of fitting the varying size memory chunks on to the backing store.

(i) Basic Method:

- Divide logical memory into blocks of same size called **“pages”**.
- Divide physical memory into fixed-sized blocks called **“frames”**
- Page size is a power of 2, between 512 bytes and 16MB.

Address Translation Scheme

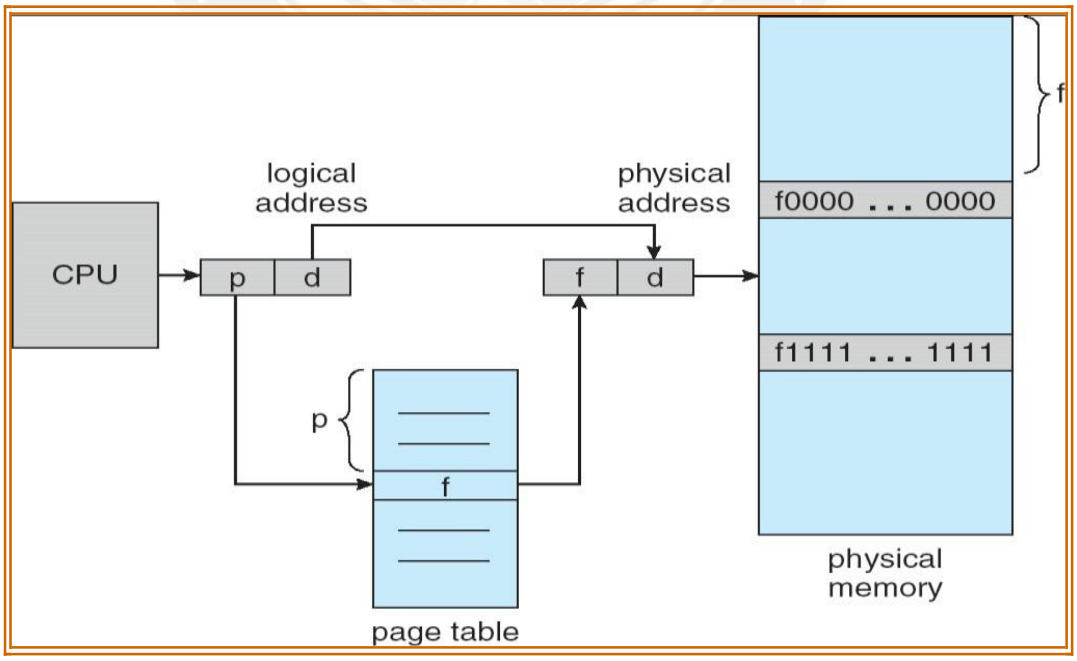
Address generated by CPU(logical address) is divided into:

Page number *(p)* – used as an index into a page table which contains base address of each page in physical memory

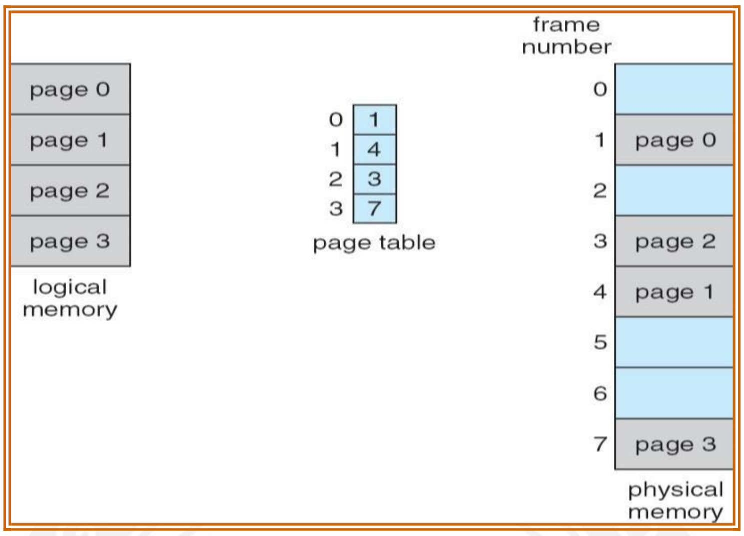
Page offset *(d)* – combined with base address to define the physical address i.e.,

Physical address = base address + offset

Paging Hardware



Paging model of logical and physical memory



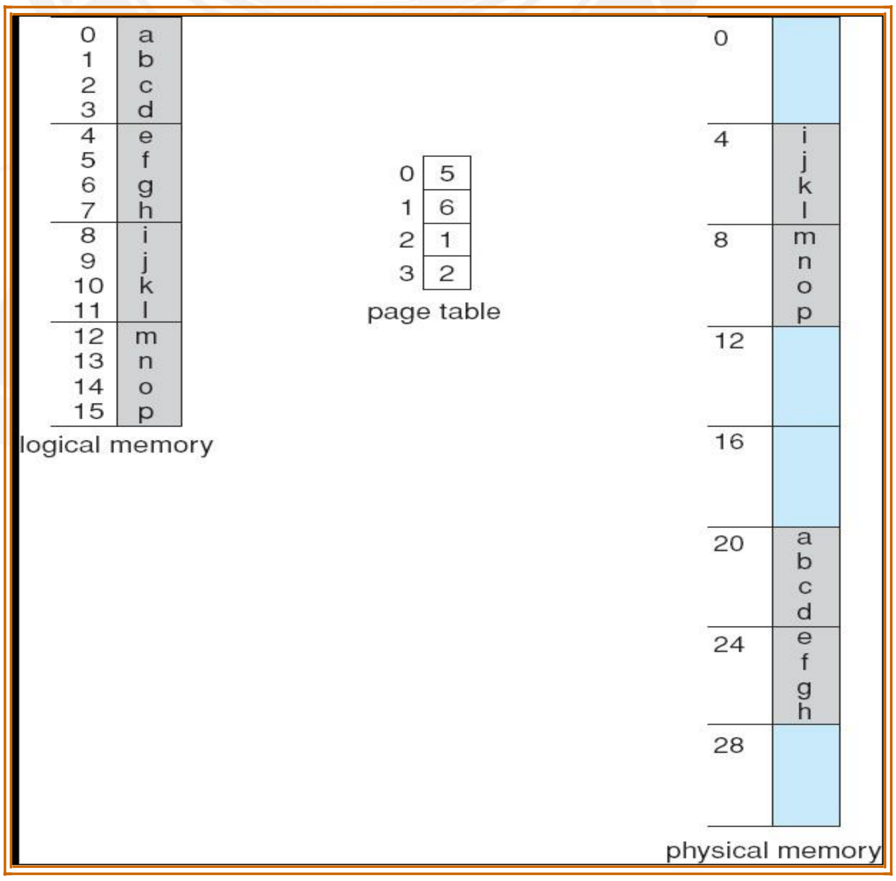
<u>Paging example for a 32-byte memory with 4-byte pages</u>

Page size = 4 bytes

Physical memory size = 32 bytes i.e (4 X 8 = 32 so, 8 pages) Logical address "0' maps to physical address 20 i.e ((5 X 4) +0) Where Frame no = 5,

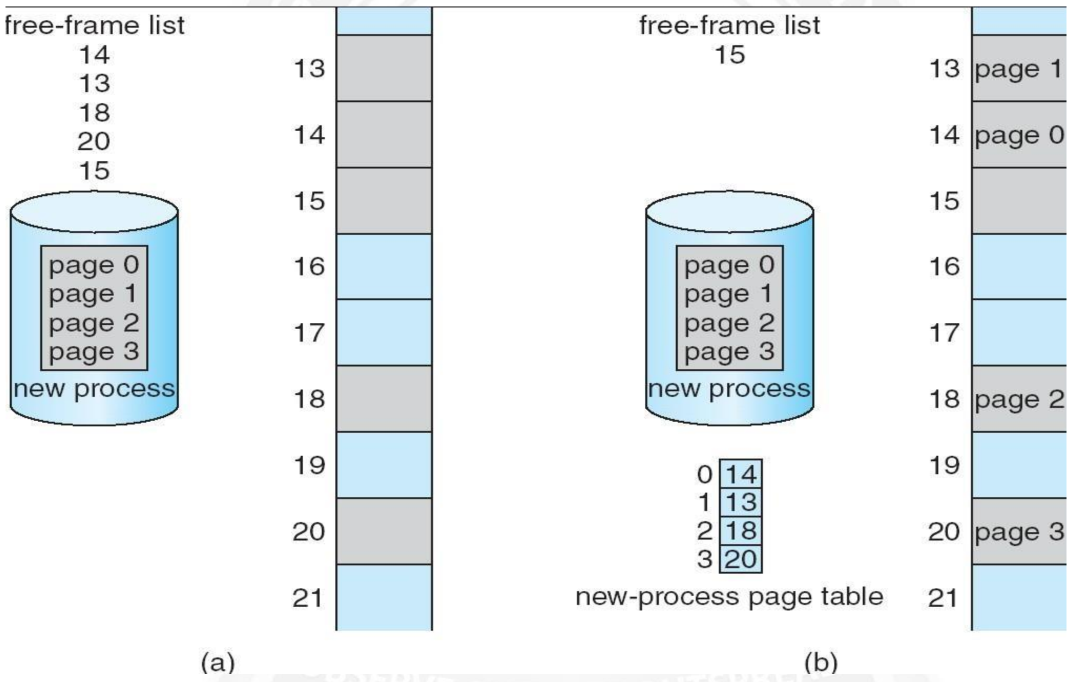
Page size = 4,

Offset = 0



Allocation

- When a process arrives into the system, its size (expressed in pages) is examined.
- Each page of process needs one frame. Thus if the process requires ‗n‘ pages, at least ‗n‘ frames must be available in memory.
- If ‗n‘ frames are available, they are allocated to this arriving process.
- The 1st page of the process is loaded into one of the allocated frames & the frame number is put into the page table.
- Repeat the above step for the next pages & so on.



(a) Before Allocation (b) After Allocation

Frame table: It is used to determine which frames are allocated, which frames are available, how many total frames are there, and so on.(ie) It contains all the information about the frames in the physical memory.

(ii) Hardware implementation of Page Table

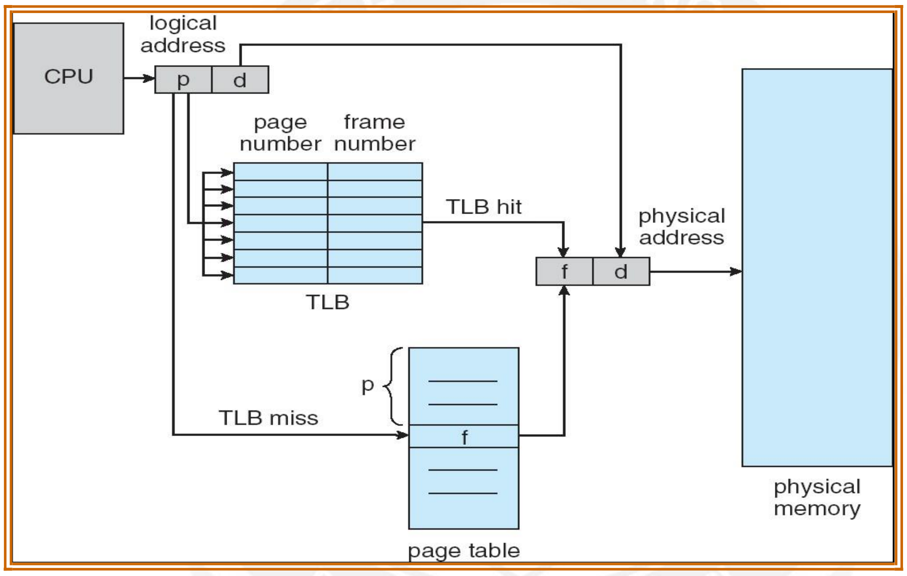
This can be done in several ways :

1. Using PTBR
2. TLB

The simplest case is **Page-table base register (PTBR**), is an index to point the page table.

1. TLB (Translation Look-aside Buffer)

- It is a fast lookup hardware cache.
- It contains the recently or frequently used page table entries.
- It has two parts: Key (tag) & Value.
- More expensive.



<u>Paging Hardware with TLB</u>

When a logical address is generated by CPU, its page number is presented to TLB.

<u>TLB hit</u>: If the page number is found, its frame number is immediately available & is used to access memory

<u>TLB miss</u>: If the page number is not in the TLB, a memory reference to the page table must be made.

<u>Hit ratio:</u> Percentage of times that a particular page is found in the TLB.

For example hit ratio is 80% means that the desired page number in the TLB is 80% of the time.

<u>Effective Access Time:</u>

Assume hit ratio is 80%.

- If it takes 20ns to search TLB & 100ns to access memory, then the memory access takes 120ns(TLB hit)
- If we fail to find page no. in TLB (20ns), then we must 1st access memory for page table (100ns) & then access the desired byte in memory (100ns).
- Therefore Total = 20 + 100 + 100

  = 220 ns(TLB miss).
- Then Effective Access Time (EAT) = 0.80 X (120 + 0.20) X 220.

  = 140 ns.

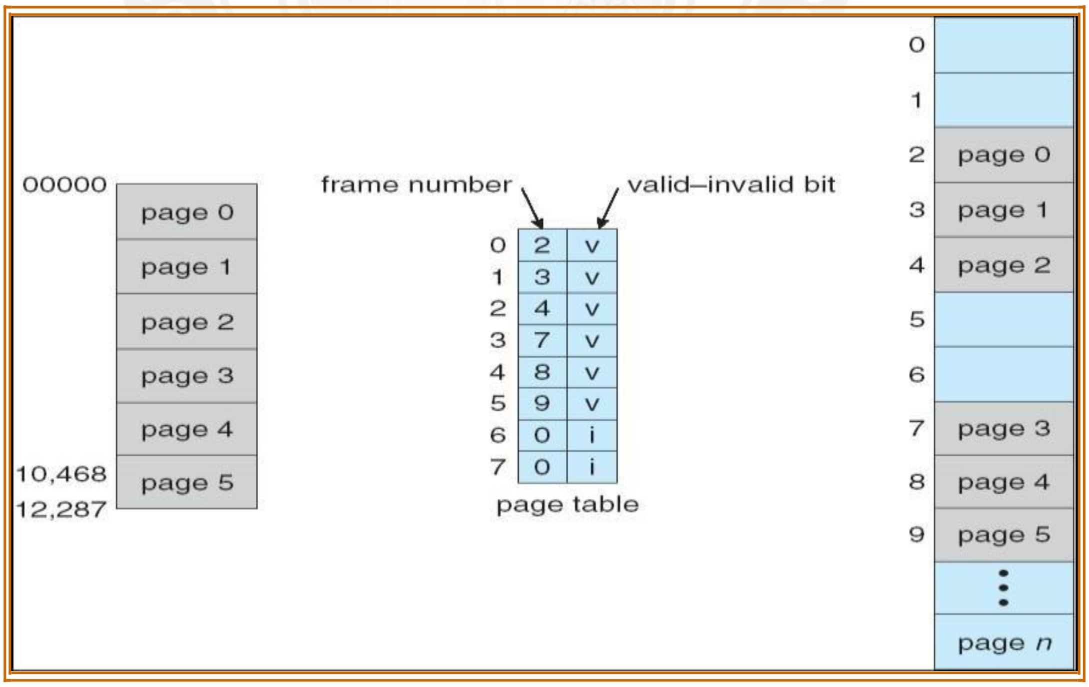
(iii) Memory Protection

Memory protection implemented by associating protection bit with each frame

Valid-invalid bit attached to each entry in the page table:

"valid (v)" indicates that the associated page is in the process' logical address space, and is thus a legal page

"invalid (i)" indicates that the page is not in the process' logical address space



(iv) Structures of the Page Table

a) Hierarchical Paging

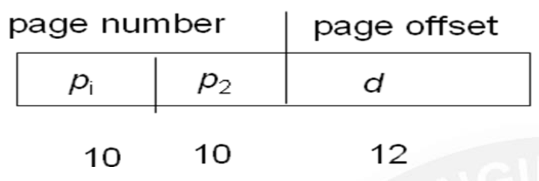
b) Hashed Page Tables

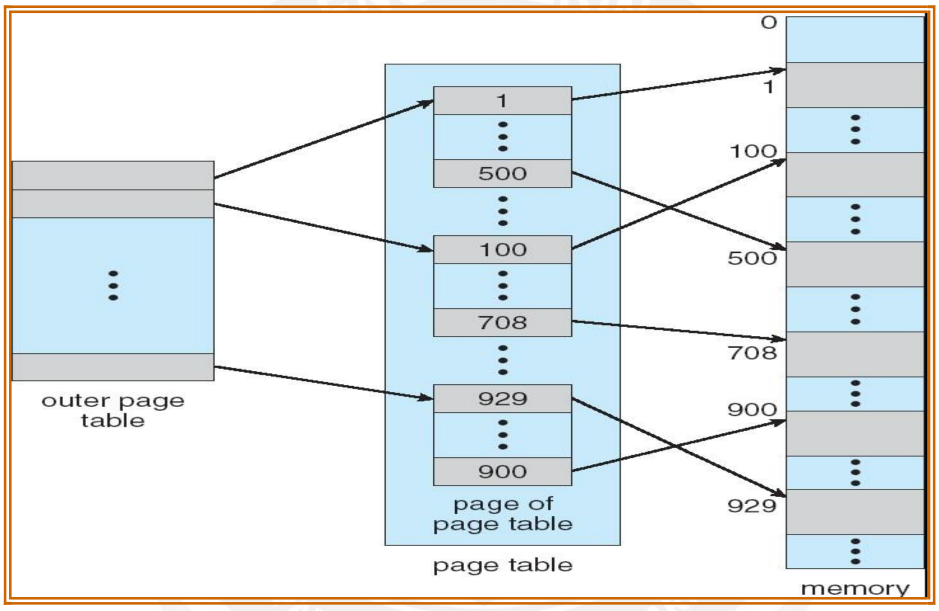
c) Inverted Page Tables

a) <u>Hierarchical Paging</u>

Break up the Page table into smaller pieces. Because if the page table is too large then it is quit difficult to search the page number.

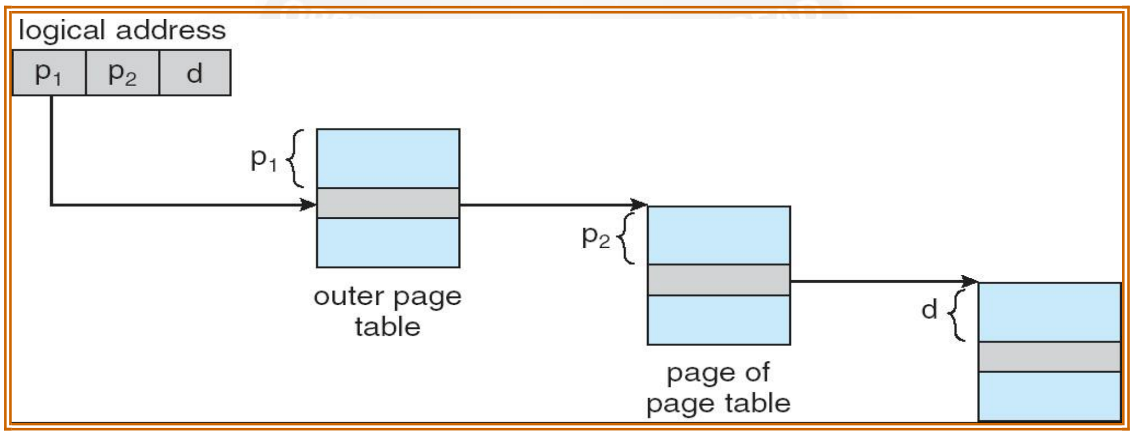
Example: "Two-Level Paging "





Address-Translation Scheme

Address-translation scheme for a two-level 32-bit paging architecture



It requires more number of memory accesses, when the number of levels is increased.

(b) Hashed Page Tables

Each entry in hash table contains a linked list of elements that hash to the same location.

Each entry consists of;

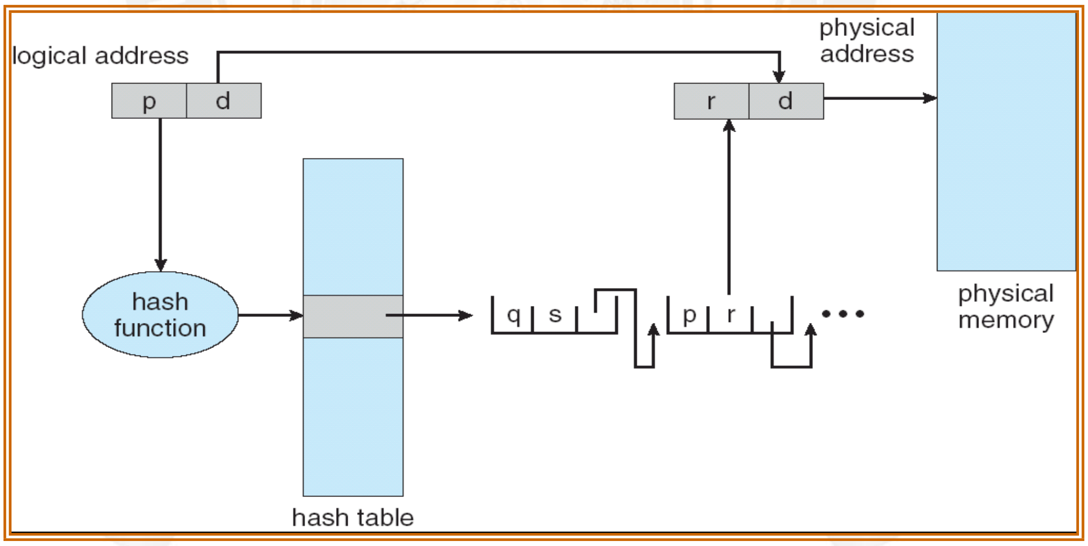
(a) Virtual page numbers

(b) Value of mapped page frame.

(c) Pointer to the next element in the linked list.

Working Procedure:

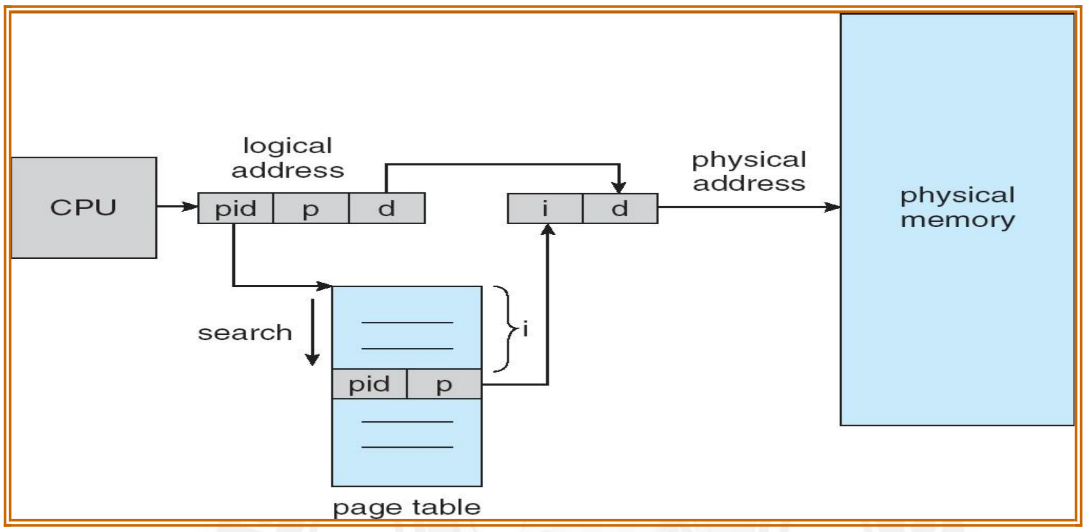
- The virtual page number in the virtual address is hashed into the hash table.
- Virtual page number is compared to field (a) in the 1st element in the linked list.
- If there is a match, the corresponding page frame (field (b)) is used to form the desired physical address.
- If there is no match, subsequent entries in the linked list are searched for a matching virtual page number.



Clustered page table: It is a variation of hashed page table & is similar to hashed page table except that each entry in the hash table refers to several pages rather than a single page.

(c) Inverted Page Table

It has one entry for each real page (frame) of memory & each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page. So, only one page table is in the system.



- When a memory reference occurs, part of the virtual address ,consisting of <Process-id, Page-no> is presented to the memory sub-system.
- Then the inverted page table is searched for match:

(i) If a match is found, then the physical address is generated.

(ii) If no match is found, then an illegal address access has been attempted.

Merit: Reduce the amount of memory needed.

Demerit: Improve the amount of time needed to search the table when a page reference oocurs.

(v) Shared Pages

One advantage of paging is the possibility of sharing common code.

Shared code

One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

Shared code must appear in same location in the logical address space of all processes

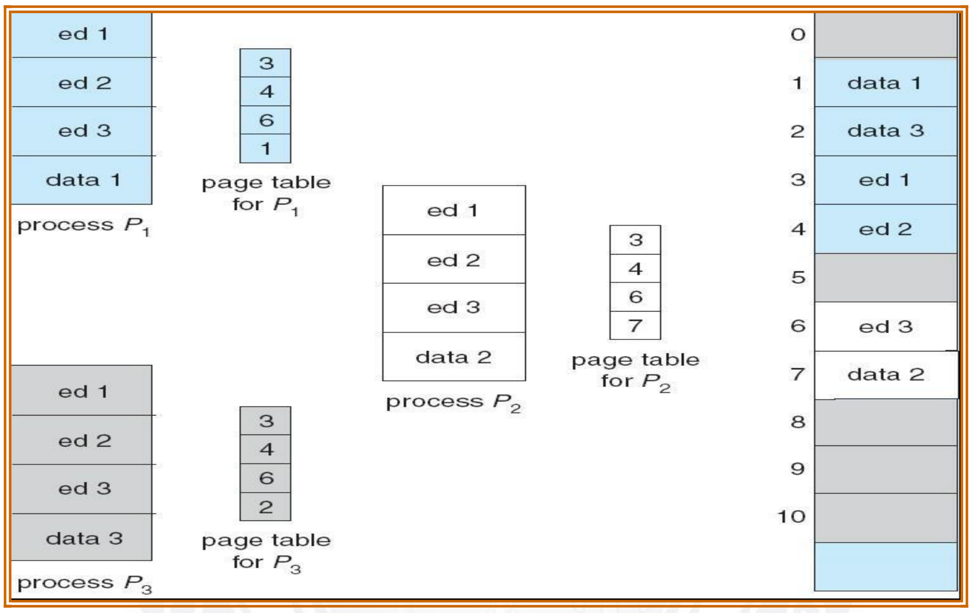
Reentrant code (Pure code): Non-self modifying code. If the code is reentrant, then

it never changes during execution. Thus two or more processes can execute the same code at the same time.

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

EXAMPLE:



Drawback of Paging – Internal fragmentation

In the worst case a process would need n pages plus one byte.It would be allocated n+1 frames resulting in an **internal fragmentation** of almost an entire frame.

Example:

Page size = 2048 bytes

Process size= 72766 bytes

Process needs 35 pages plus 1086 bytes.

It is allocated 36 frames resulting in an internal fragmentation of 962 bytes.

OBSERVE OPTIMIZE OUTSPREAD