(5) Memory Management

- memory is storage unit
- Program must be brought from secondory mem. to primary mem.
- CPU can access main memory and registers directly
  - ▷ Register access in 1 CPU clock (or less)
  - ▷ main memory access takes multiple clock cycle & it leads to stalls.

Cache : - stores frequently used data
- Speed up access time of memory
- Present betⁿ main memory & Registers.

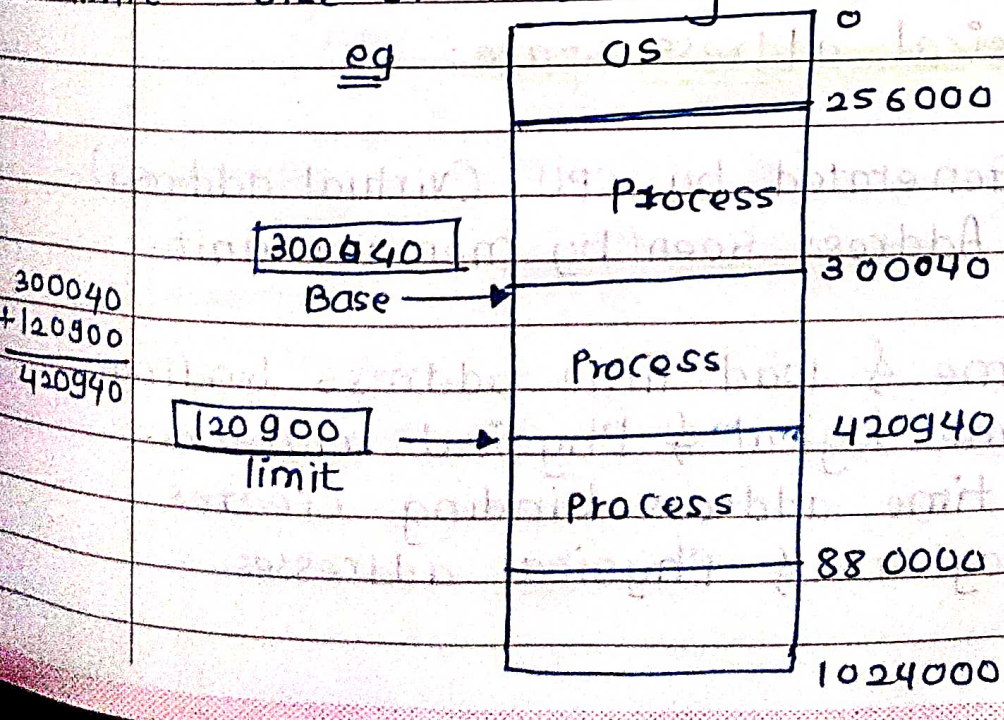
Base and limit Registers :

- Defines logical address space.

Base - Holds smallest legal physical address

limit - Size of the range.

eg



Address Binding:

- Programs on disk, ready to be brought into memory for execution.
- During program life cycle addresses are represented as follows:
  - Src code address - Symbolic
  - Compile code address - relocated
  - Linker and loader - absolute

- Binding of instruction & data to memory:
  Three stages:

1) Compile time:
   - If memory location known - absolute code generated
   - Recompilation required if location change.

2) Load time:
   - If mem location unknown at compile time - relocated code is generated.

3) Execution time:
   - Occurs when process moves from one to another mem. segment during execution.

Logical Vs Physical address space:

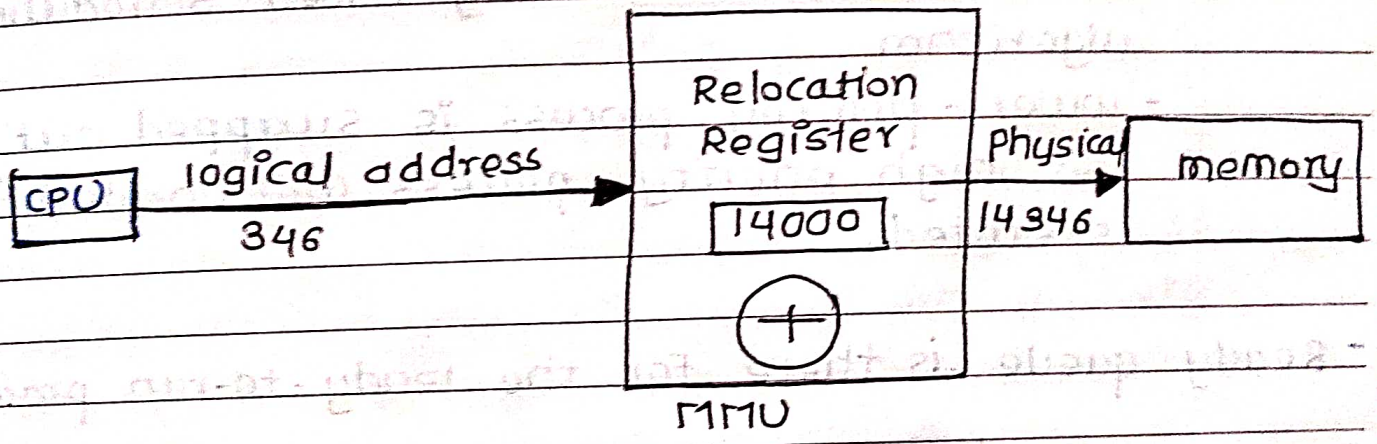
Logical address - Generated by CPU (Virtual address)
Physical address - Address seen by memory unit.

* Compile time & load time address binding creates same logical & Physical addresses.
* Execution time address binding creates different logical & Physical addresses.

- Logical address space:
  Set of all logical address generated by program.
- Physical address space:
  Set of all physical addresses corresponding to logical addresses.

Memory management Unit (MMU)

- H/w device that maps logical to physical address
- In MMU,
  value in relocation reg. is added to every logical address to generate physical.



Linking

Static linking

Process of combining all necessary libraries, routines & external references into executable File at compile time

Dynamic linking

—11— linking
all —11—
—11— references at runtime.

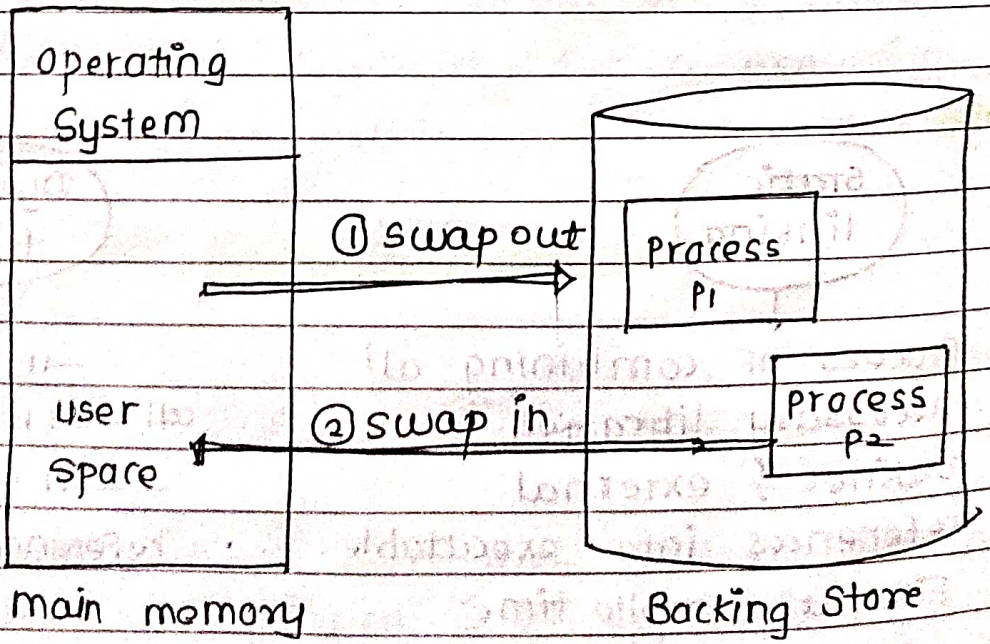
Swapping

- Process can be swapped temporarily out of memory to a backing store, and then brought back to memory for continued execution.

- Backing store : fast disk are large enough to accommodate copies of all memory images for all users.

- Roll out, Roll in :
  - Variant used for priority-based scheduling algorithm.
  - lower-priority process is swapped out so high priority process will loaded & executed.

- Ready queue is there for the ready-to-run processes.



Memory management techniques

1. Contigous Allocation
2. Non contigous Allocation

1. Contigous Allocation :

- Allows to store a process only in contiguous fashion.

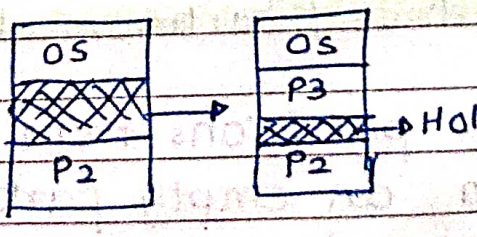
→ (A) Single contiguous memory allocation

- main mem. divided into two areas or partitions
  - os in one partition
  - And user process in another partition.

(B) Multiple partition :

- main memory is divided into multiple parts to load multiple processes into main memory.

- Now, after completion space occupied by P1 will be free, and that free space/partition called as hole
- when New process comes, it is allocated mem. from a hole large enough to accomodate it

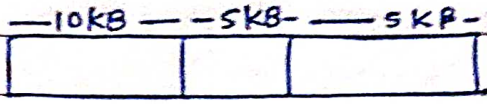


Multiple partition types : (A) Fixed / Static partitioning
(B) Variable / Dynamic partitioning

(A) Fixed Partitioning:

- main memory is divided into several fixed size partitions
- They can be of same or different sizes.
- Each partition holds single process
- No. of partition determines degree of multiprogramming

eg



memory of 20 KB

- Disadvantages:
  - Suffers from internal & external fragmentation
  - Inefficient memory utilization.
  - less degree of multiprogramming.

(B) Variable Size partitioning:

- designed to overcome problem of fixed partitioning
- Partition created as per the size of the process loaded for processing.
- Partition used are of variable size.

Disadvantage:

- Suffers from external fragmentation
- Allocation / Deallocation is complex.

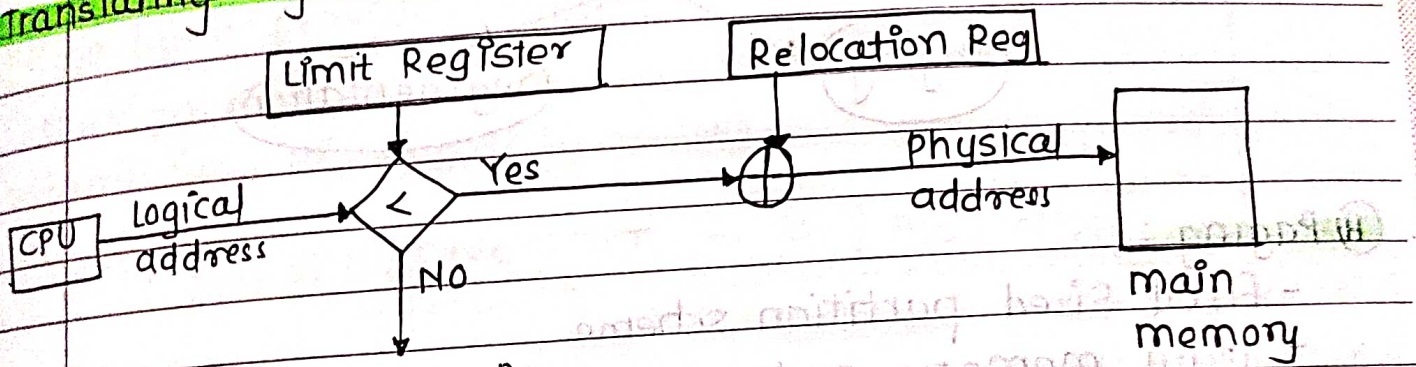
Memory Allocation Methods / Partition allocation methods

1) **First fit** - scans partitions from starting.
   - when an empty partition that is big enough to store the process is found, then it is allocated to that process.

② **Best Fit Algorithm**
- → Scan all empty partitions
- → Allocate smallest partition that is big enough, & produces smallest leftover partition.

③ **Worst fit**
- → Scan all empty partitions
- → Allocate biggest partition that is big enough & produces largest left over partition.

Translating logical address into physical address



Fragmentation:

① **Internal Fragmentation** :- Occurs only in static partitioning
- when allocated memory is larger than requested.
- Remaining space from that partition can't be used.

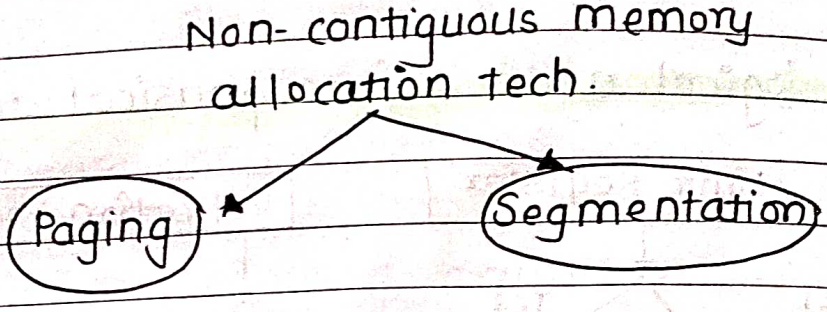
② **External Fragmentation** :- - When total amount of space required to store the process is available but not in contiguous.

↳ Solution : Compaction
i) shuffle memory contents to place all free memory together.

② Non-contiguous memory allocation :

- Program is divided into different blocks and loaded at different portions of the memory which need not necessarily be contiguous.

- stores the part of single process into a non-contiguous fashion.

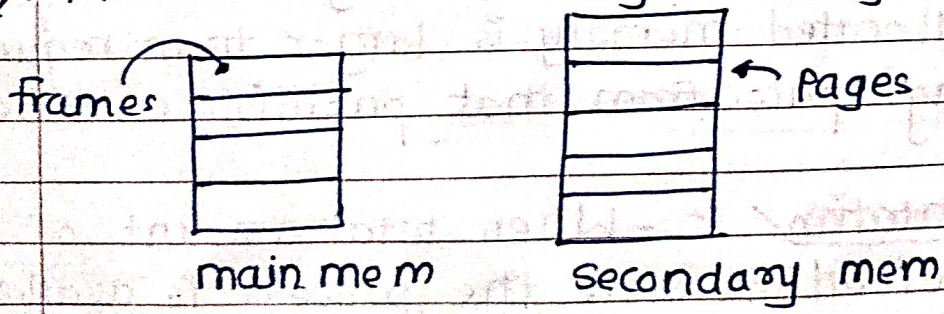


(A) Paging :

- Fixed sized partition scheme
- Main memory and secondary memory are divided into equal fixed size partitions.

Frames → Partitions of main memory / ~~logical~~ physical memory

Pages → Partitions of secondary memory / logical memory.



- Process is divided into pages of equal size.
- These pages are stored in main memory frames as per availability, in non-contiguous fashion.

Translating logical Address into physical address (done by MMU)

- CPU generates logical address

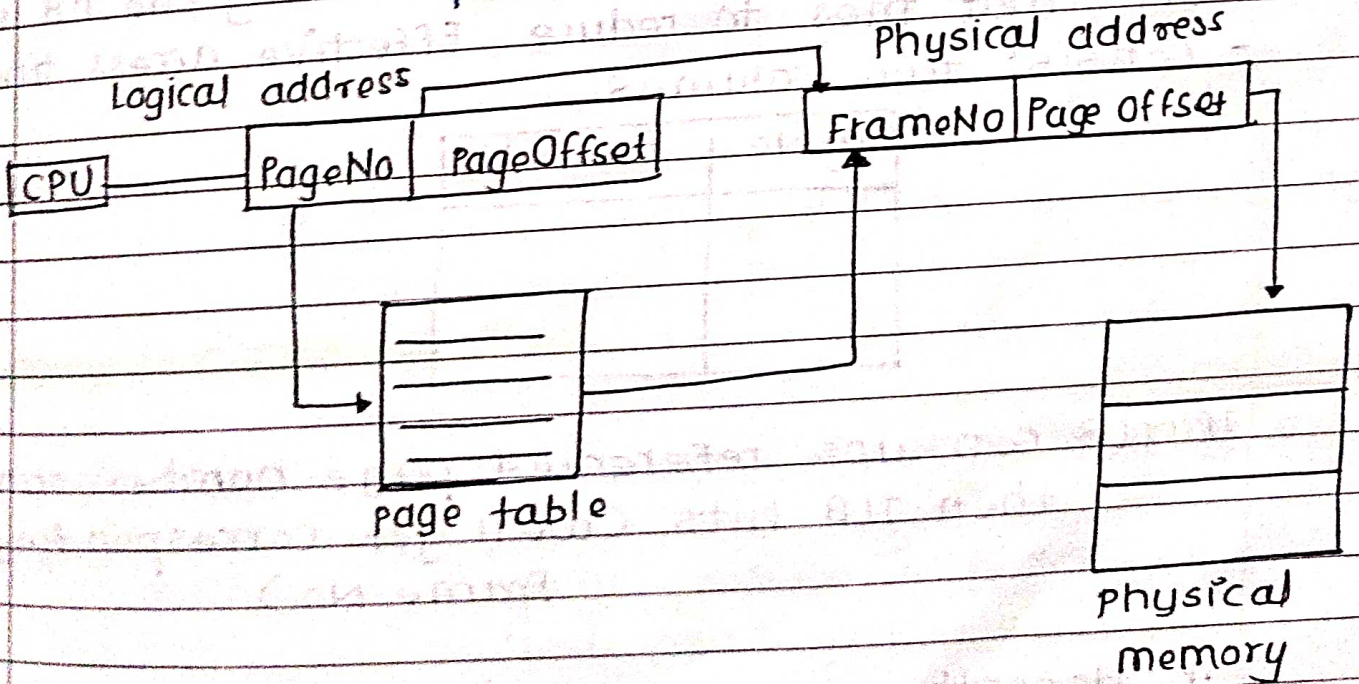
logical address : | Page No. | Page offset |

- Page No. → page from process from which CPU wants to read data
- Page offset → Specific word on that page wanted by CPU.

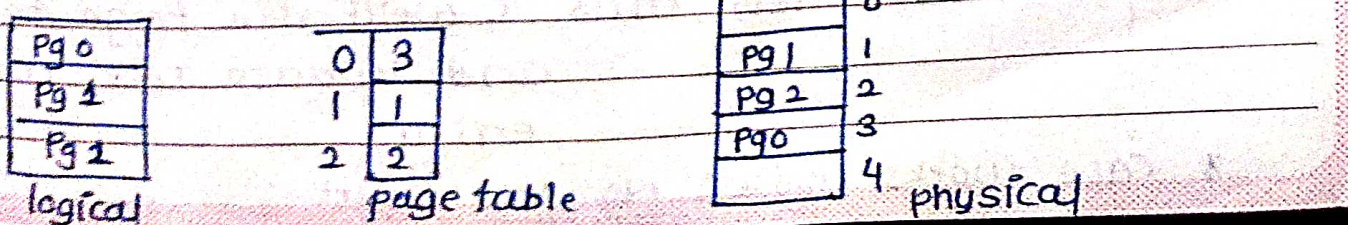
- Page table provides corresponding frame No. where that page is stored
- formation of physical address

pysical address : | Frame No | Page offset |

- Frame No → Frame where required page is present.
- Page offset → Specific word on that Page.



Example



Page Table

- maps page No with frame No.
- Stored in main memory
- No. of enteries in page table = Pages in process.
- Each process has its own table.

- PTBR (Page Table Base Reg) – Points to page table.
- PRLR (Page-table length Reg) – Points size of the page table.

- Every data access requires two times memory access i.e for page table & for data.

↳ This problem is solved by use of special fast-lookup H/w cache called associative memory or translation look-aside Buffer (TLB)

TLB (Translation look-aside Buffer : -only one TLB in sys

- Soln that tries to reduce Effective access time.
- Consist two columns

Pg No	Frame No

If TLB contains referenced page number entry then TLB hits (we'll get corresponding frame No.)

If doesn't
the TLB miss (we'll use Page table for it and update TLB with this entry)

* Case works when No page fault

Effective Access Time = (EAT)

$(1-P) \times ma + P \times$ page fault time

P = probability of page fault
ma = memory access time.

Ex: Hit Ratio = 80%
ma = 10 ns
page fault time = 2ma

→ Here $(1-P) = 80\% = 0.80$
$\therefore P = 20\% = 0.20$
$ma = 10$
page fault time = $2ma = 2 \times 10 = 20$

$$\therefore EAT = \left(\frac{0.80}{100} \times 10\right) + \left(\frac{0.20}{100} \times 20\right)$$

$$= \frac{0.80}{100} + \frac{0.40}{100}$$

$$= \frac{1.20}{100}$$

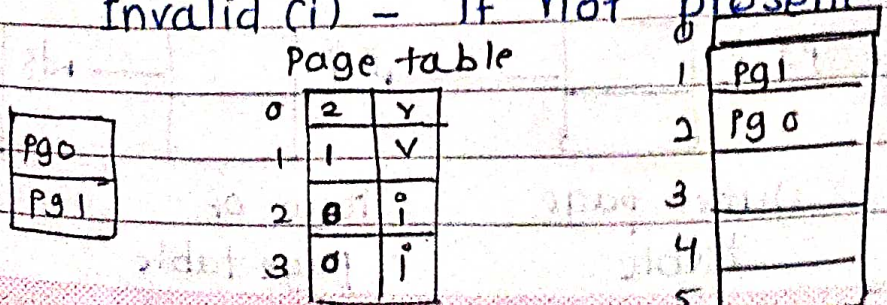
EAT = 12 ns

Memory protection in Paging

- Valid- Invalid bit is attached with each entry in page table

Valid (v) – if page present in logical address space

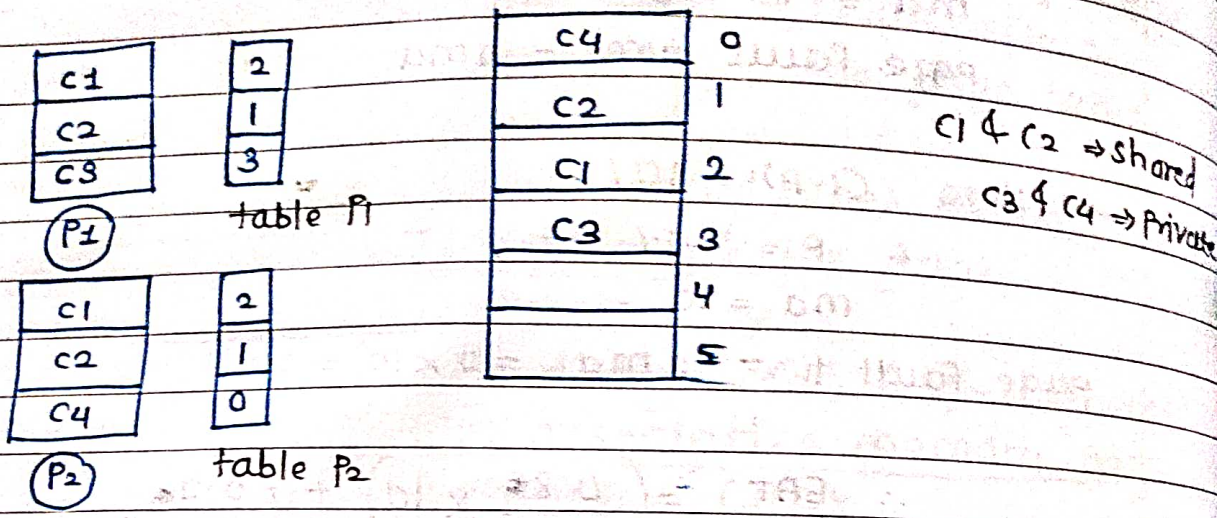
Invalid (i) – if not present in -11-



shared pages :

shared code
- Copy of code is shared among processes.
- Must appear in same location in logical address space

private code
- Each code has separate code also.
- stored on different location

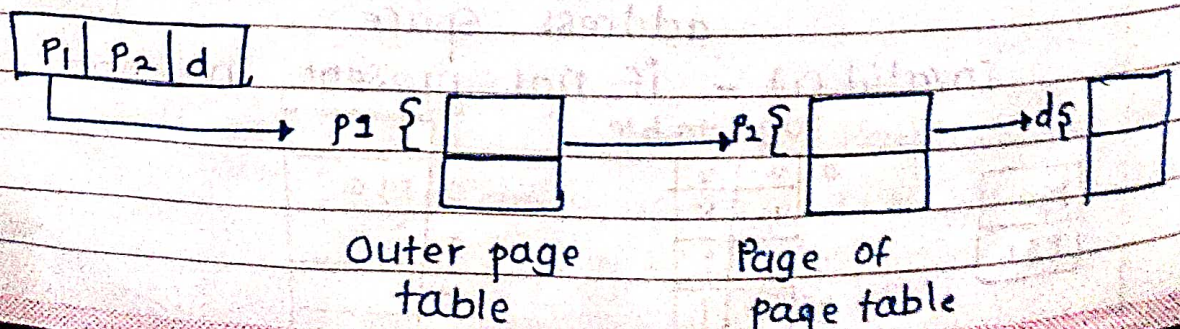


Structure of page table :

- In order to store page table, finding big continous space can be difficult
- So, divide page table in smaller units
  - (A) Hierarchical paging
  - (B) Hashed Page Tables
  - (C) Inverted Page Tables

(A) Hierarchical Paging

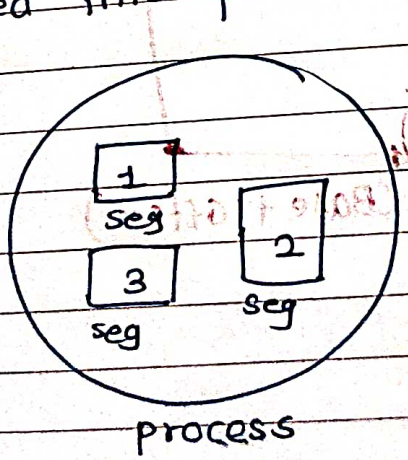
- Break up the logical address space into multiple page tables.

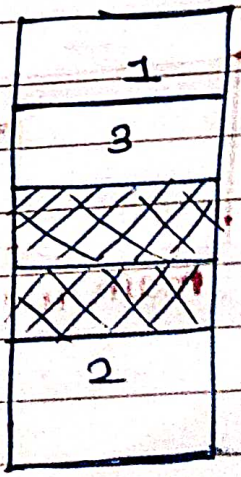


(b) Hashed Page Table :
- The virtual page no. is hashed into a page table - using linked list

(c) Inverted Page table :
- one entry for each real page.
- Entry consist of virtual address of the page stored in that real memory location.

(d) Segmentation :
- Another non-contiguous memory allocation technique.
- Process is not divided blindly into fixed size pages.
- Rather process is divided into modules / segments → logical unit
- solves prblm of internal fragmentation.
- Here main memory & secondary memory are divided into partitions of unequal size.



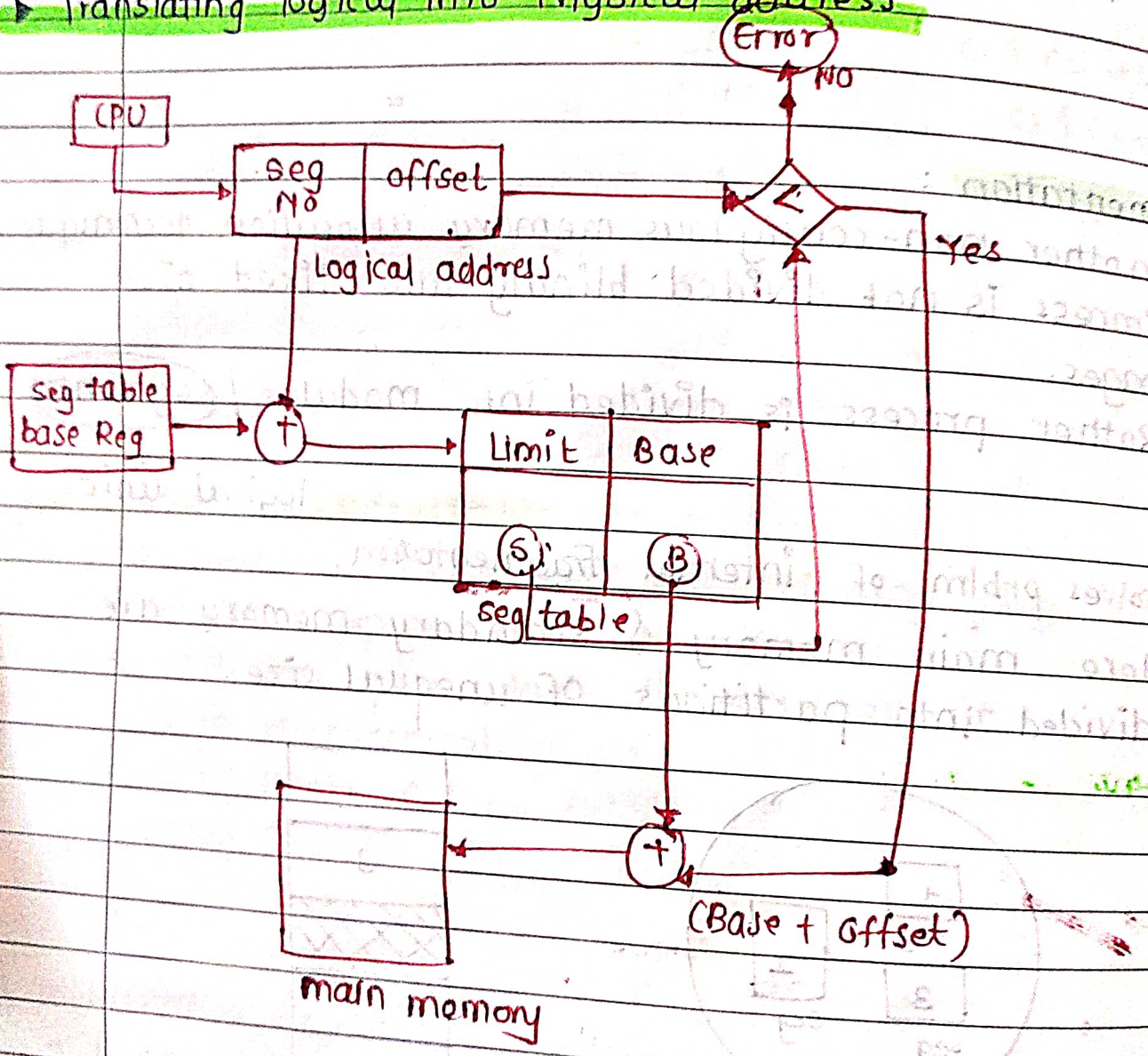


Segment Table :
- stores info about each segment.
- Has two columns :
  first column = size or length of segment (limit)

Second column = stores starting address of segment (base)

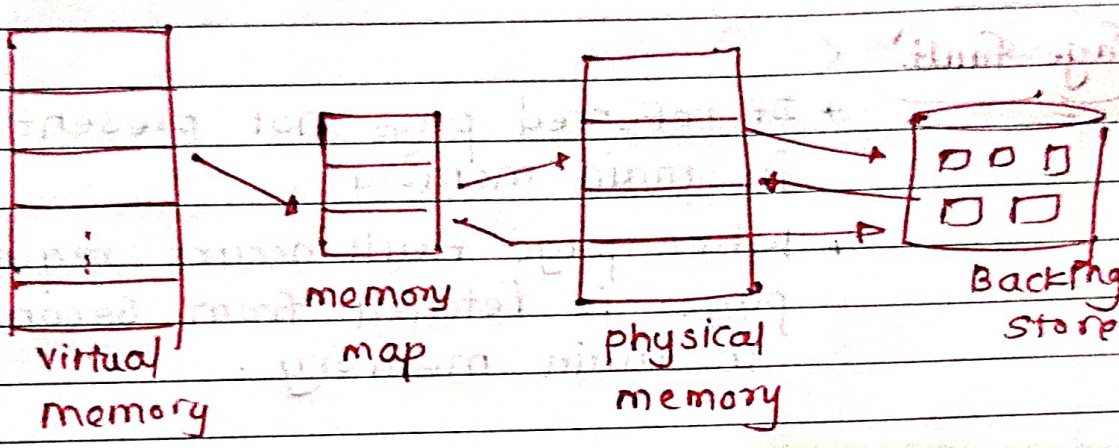
- segment table is stored as a separate segment in the main memory.
- STBR = Segment Table Base Register
  - stores base address of segment.

▶ Translating logical into Physical address



Virtual Memory

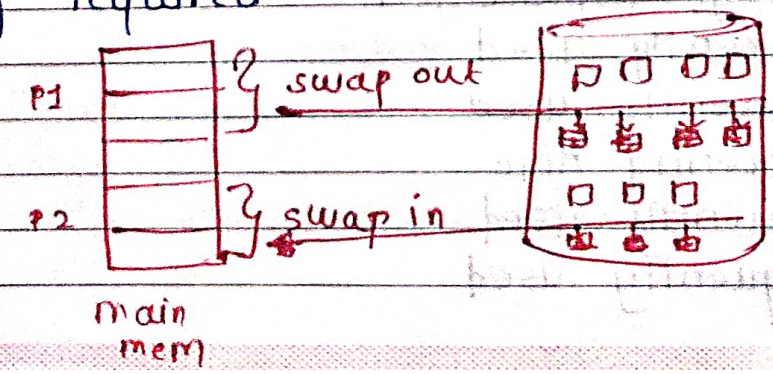
- Separation of user logical memory from physical memory
- Virtual memory is memory generated from physical devices like disk when less amount of main memory is available.
- Implemented using demand paging / demand segmentation
- In this very large virtual memory is provided for programmers on small physical memory.
- More programs runs concurrently.

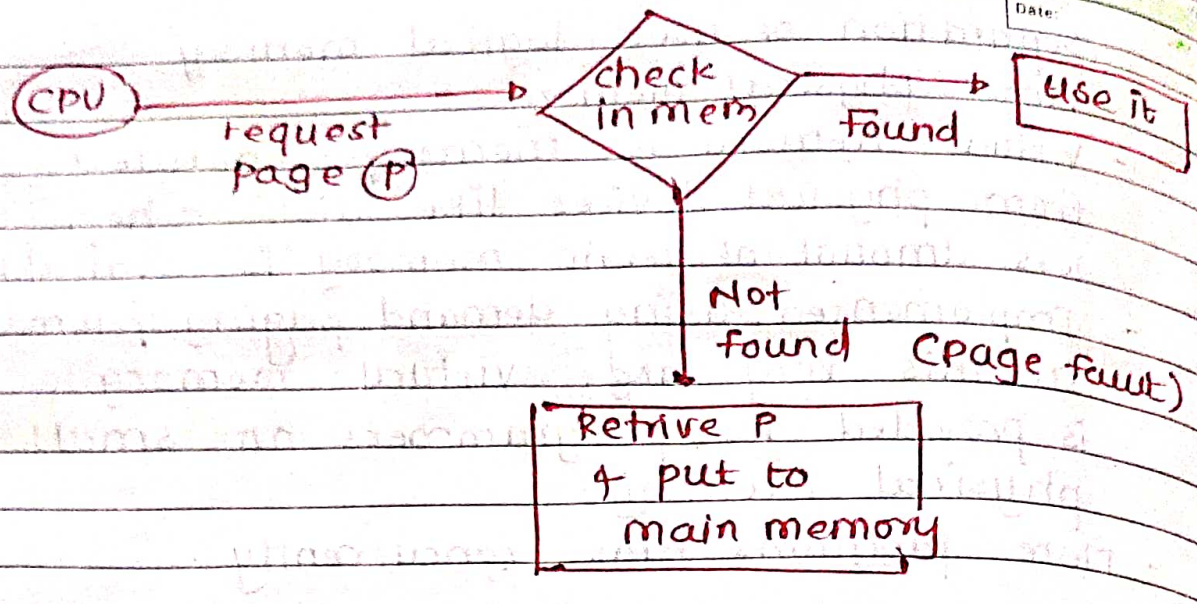


Demand Paging

→ According to concept of virtual memory only few pages of process will present in main memory at any time.

→ Deciding which page to keep in memory is difficult so to solve this problem demand paging required.





Page fault ?

→ If referred page not present in main memory.

→ When page fault occurs required page is fetched from secondary to main memory.

Page Replacement :

- Technique used to decide which memory page to swap.
- Used when page fault occurs

Page Replacement Algorithms :

- Help to decide which page must be swapped out

1. FIFO
2. Optimal Page Replacement
3. Least Recently used
4. Most Recently used
5. Page Buffering Algo.
6. less frequently used
7. Most frequently used

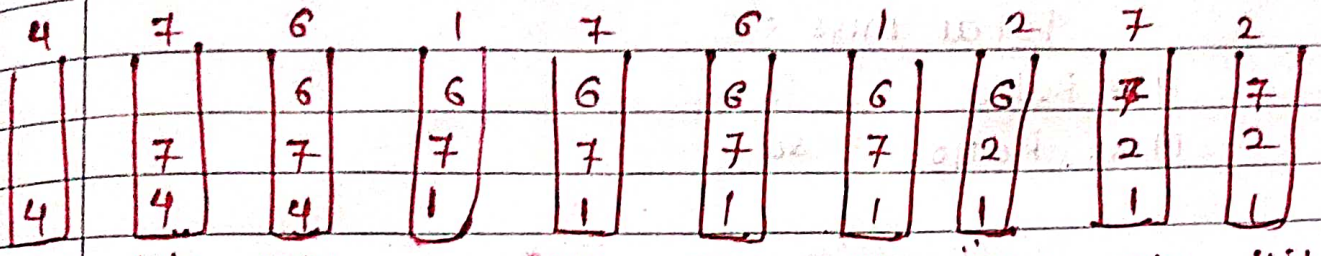
(1) First in First Out

- uses queue
- Replaces oldest page

EX: 4, 7, 6, 1, 7, 6, 1, 2, 7, 2

Consider 3 frames

Reference string = 4 7 6 1 7 6 1 2 7 2



∴ Total page fault (miss) = 6

page Hit (H) = 4

Total = 10

∴ Hit Ratio = 4/10

= 0.4 or 40%

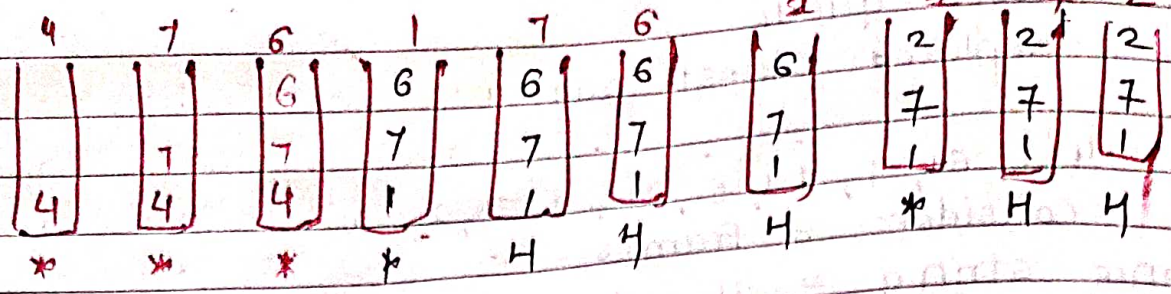
Miss Ratio = 60%

*By increasing no. of frames, page faults should be reduce but page faults gets increased this problem is Belady's Anomaly

(2) Optimal Page Replacement Algorithm :

- This algo replaces the page that will not be referred by the CPU in future for the longest time.
- Practically implementation is impossible. becaz prediction of future.
- Best Algo.

Ex : Reference String : 4, 7, 6, 1, 7, 6, 1, 2, 7, 2



∴ Total Hit = 5

Total miss = 5

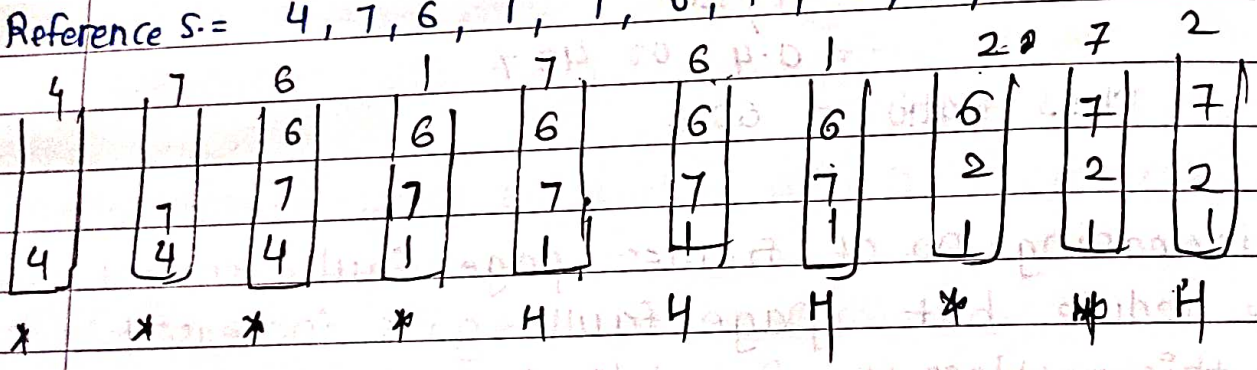
Hit Ratio = 50 %

Miss Ratio = 50 %

③ Least Recently Used (LRU)

- Page not used from longest time in main memory.

Reference S.= 4, 7, 6, 1, 7, 6, 1, 2, 7, 2



Hits = 4

Miss / faults = 6

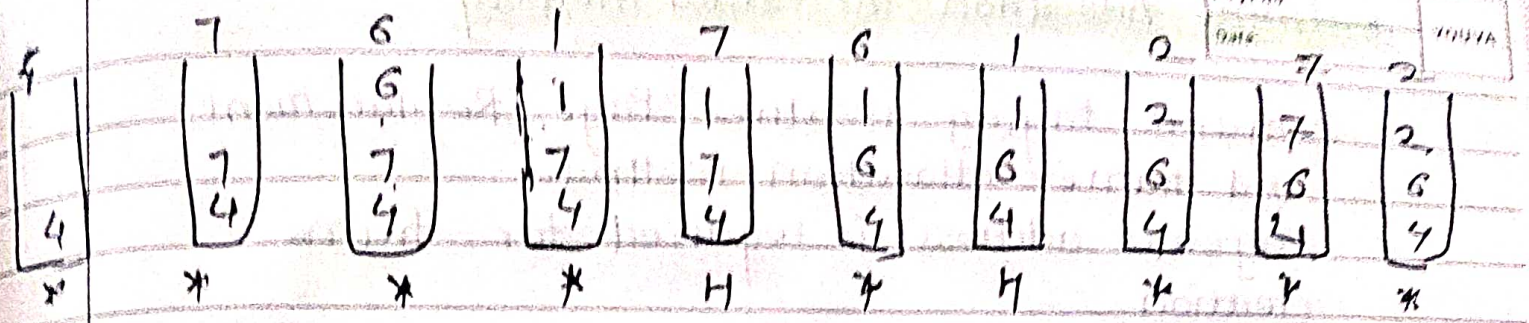
Hit ratio = 40 %

Miss ratio = 60 %.

④ Most Recently used (MRU)

-Page used recently in main memory.

4, 7, 6, 1, 7, 6, 1, 2, 7, 2



Hits = 2
miss = 8
Hit ratio = 20%.
miss ratio = 80%.

⑤ Page Buffering Algo:
- Used to get a process start quickly.
- keeps a pool of free frames.
- on page fault,
  Write a page in Frame of free pool,
  mark a page table,
  Restart the process
- Remove dirty page and place frame of replaced page in free pool.

⑥ LFU (Least Frequently Used)
- page with smallest count is replaced in past

⑦ MFU (Most Frequently used)
- Page with most highest count is replaced in past.

Frame Allocation in virtual memory

- Demand Paging requires Page Replacement and frame allocation methods
- Physical address is required for frame creation.

- Each process needs min no. of frames.
- Allocated frames can't exceed total no. of frames.
- If fewer frames = Page fault increases

Frame Allocation schemes

Fixed allocation

① Equal allocation :
- equal no. of frames to all processes.
- Wastage in case of small processes.

② Proportional allocation :
- dynamic allocation as per the size of the process.
- Wastage gets rare

Priority allocation

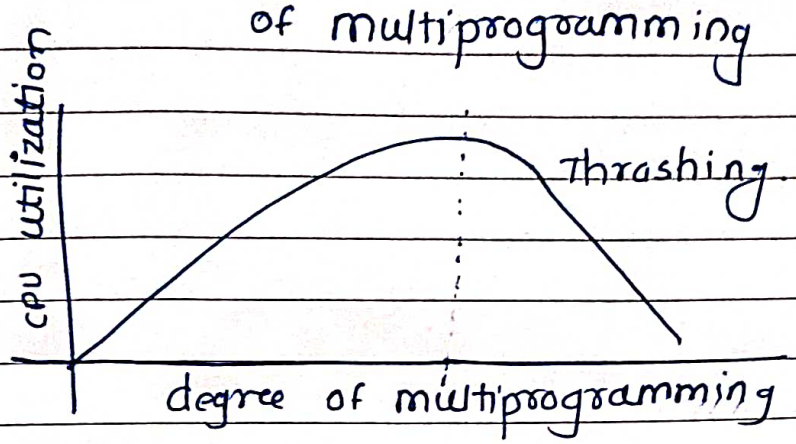
- Distribution of frame based on process priority
- High priority more frames.

Thrashing :

- A process is busy in swapping pages in and out.
- Condition where system is spending a major portion of its time in servicing

the page faults, but actual processing done is negligible.

Page replacement policy. — causes:
- lack of frames
- High degree of multiprogramming



Preventions:
- Increase physical memory
- Reduce degree of multiprogramming
- Use effective page replacement algo.