# Virtual Memory (VM)

→ VM provides the illusion to the programmer that a process whose size is larger than size of M/M can also be executed

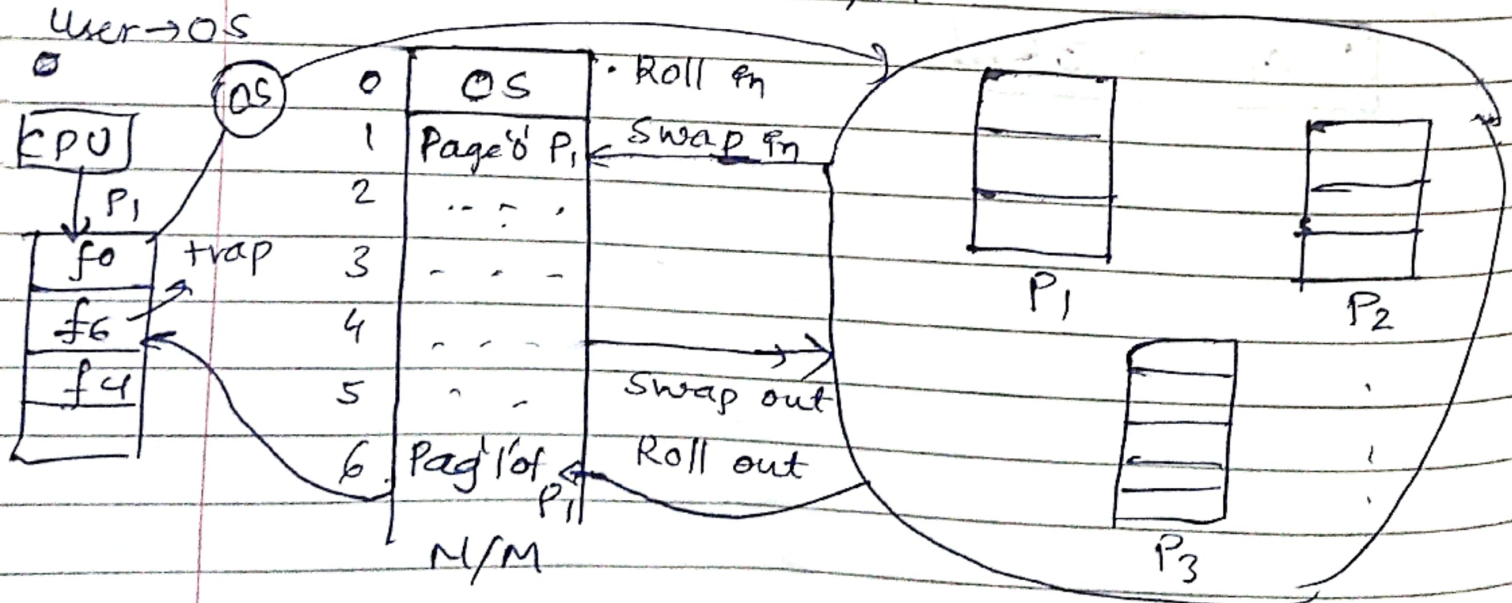
→ How we would bring maxm processes in M/M & how would we execute them is also a illusion ~~provided~~ provided by V.M.

So what do we do here,
Instead of bringing the entire process into M/M we bring only the reqd pages into M/M

## Locality of reference / Principle of locality:

Tendency of a processor to access the same set of memory locations repititively over a short period of time

→ So by this we bring parts of maxm proc. in M/M so that user thinks that entire process is there in M/M.



→ When CPU asks for a page & it's not present in page table of $P_1$, an interrup called trap is generated & control goes back to OS.
→ Now the OS checks whether the user is a valid user or not

Effective memory access time =

$$p\left(\begin{array}{l}\text{page fault}\\ \text{service time}\end{array}\right) + (1-p)\left(\begin{array}{l}\text{main mem. access}\\ \text{time}\end{array}\right)$$

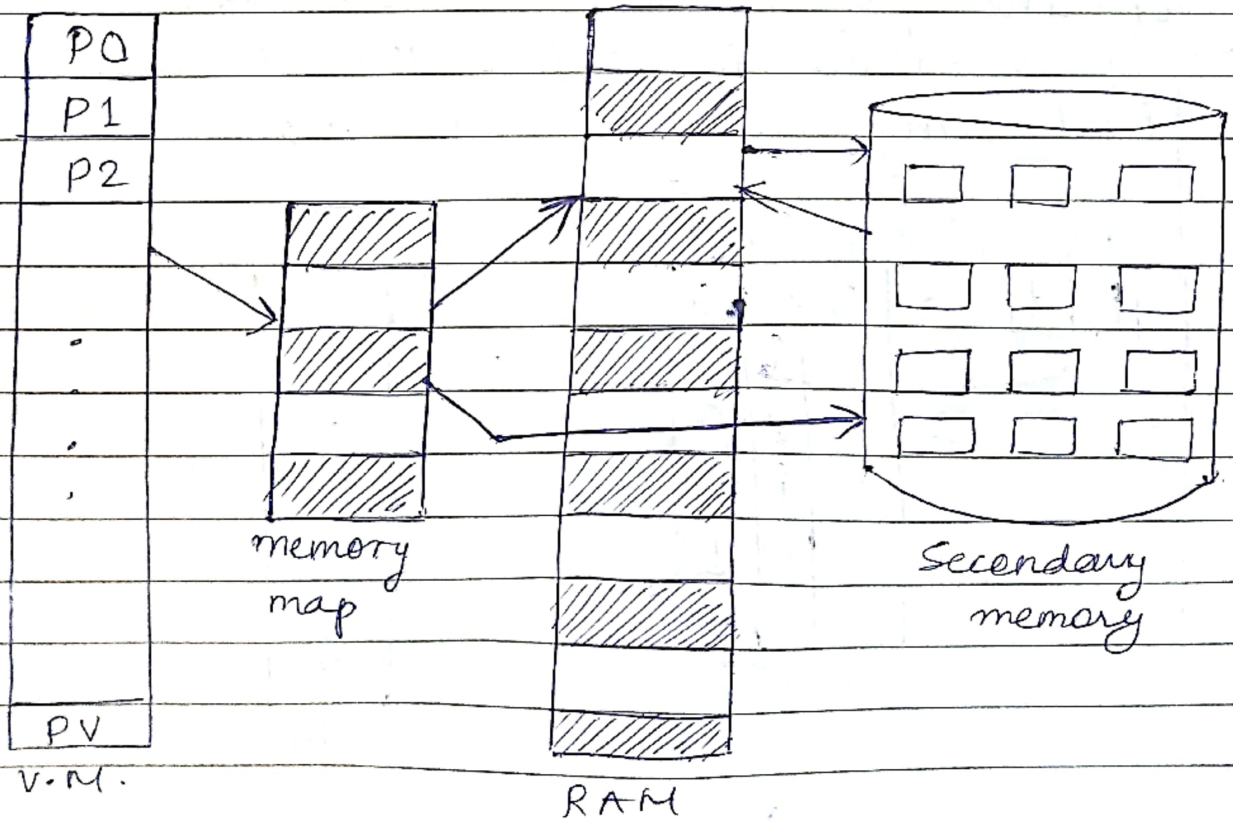
(page fault service time: gen. in 'msec'; main mem. access time: generally in 'n.s')

'p' = prob. of occurrence of page fault

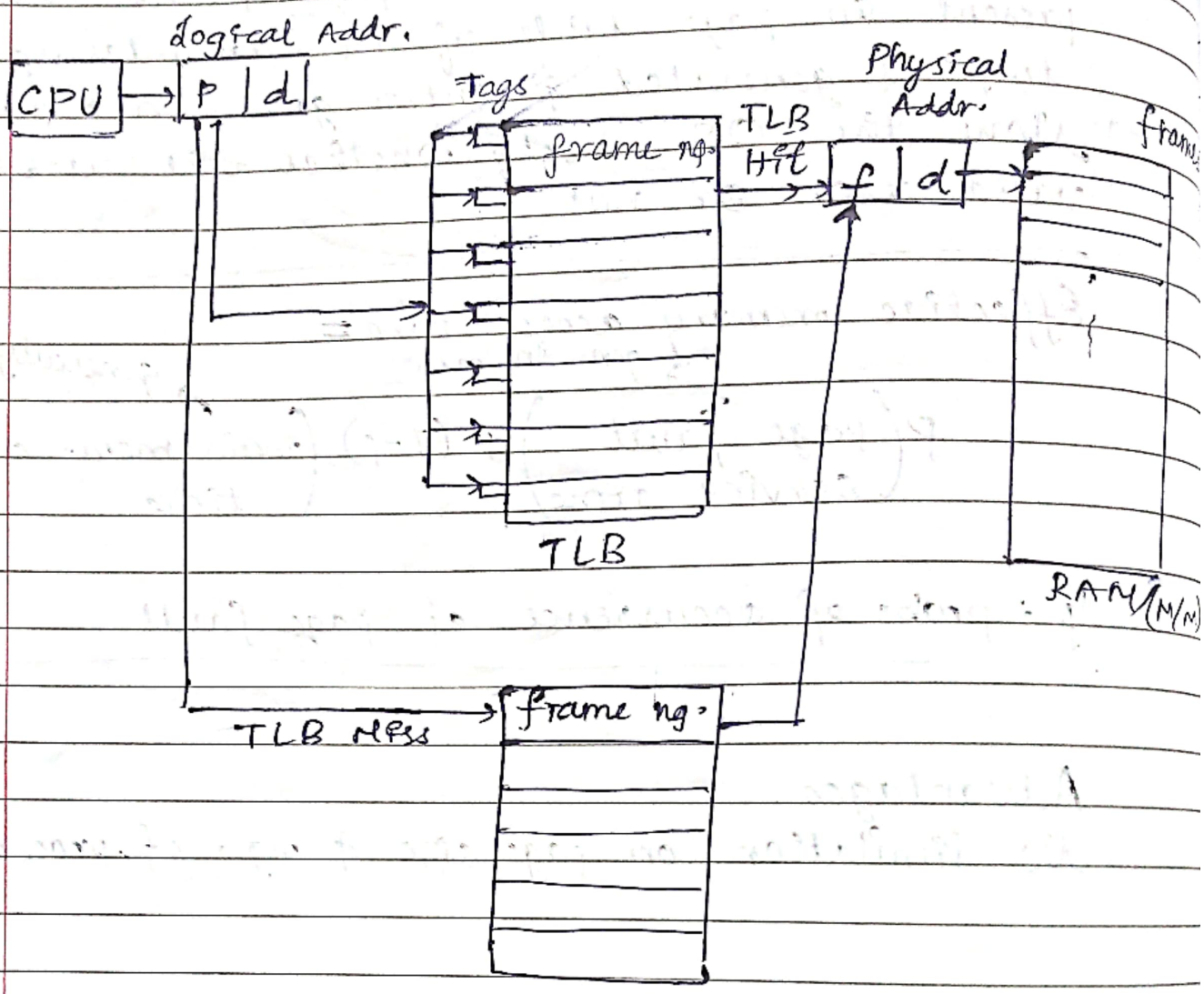
It is a technique that allows execution of processes not completely in memory

Advantages:

→ Programmer should not worry about the size of available physical memory.
→ Bigger size programs now can be easily executed
→ Reduces external fragmentation.
→ The amount of space in use by a process may be varied during its memory residence.
→ As result execution of important ~~processes~~ processes may speed up by allocating them more real memory.
→ Degree of multiprogramming ~~is~~ increased
→ Implemented by 'DEMAND PAGING'.
→ It can be implemented on 'SEGMENTATION' & 'PAGED SEGMENT' but it is quite complex



# Translation Lookaside buffer




& P.T. & pages both are on M/M.
Time to access M/M is 'x'
C-T → No page fault
→ So 1st 9'll need to access P.T. on M/M & then 9'll need to access the M/M again for frame
→ ∴ We need to access the M/M twice, ∴ we reqd 2x time
→ 9f page table is multi-level time would also go on enc.

→ TLB is faster than M/M

→ So we keep the entries of page table in TLB

→ So, first we search for the corr. frame no. in TLB & if we don't get the page there we search for it in the P.T.

→ TLB has limited size

Hit ↘

EMAT = (TLB + x) +

time to access TLB | Time to access M/M

∴ We need to access M/M twice once for P.T & once for frame

$$EMAT = Hit(TLB + x) + Miss(TLB + 2x)$$

Q A paging scheme using TLB. TLB access time 10ns & M/M access time takes 50ns. What is EMAT (in ns) if TLB hit ratio is 90% & there is no page fault.

x = time to access M/M

$$EMAT = Hit(TLB + M/M) + Miss(TLB + PT + M/M)$$
(M/M)

$$= 0.9(10 + 50) + 0.1(10 + 50 + 50)$$
$$= 0.9 \times 60 + 0.1 \times 110$$
$$= 65ns$$

# DEMAND PAGING - V.M

→ Combination of 'PAGING & SWAPPING' - 'LAZY-SWAPPER' ~~ISSIITS US~~ is used for swapping.

→ Page is brought when only demanded

→ PMT will have entries of those pages which are brought in M/M.

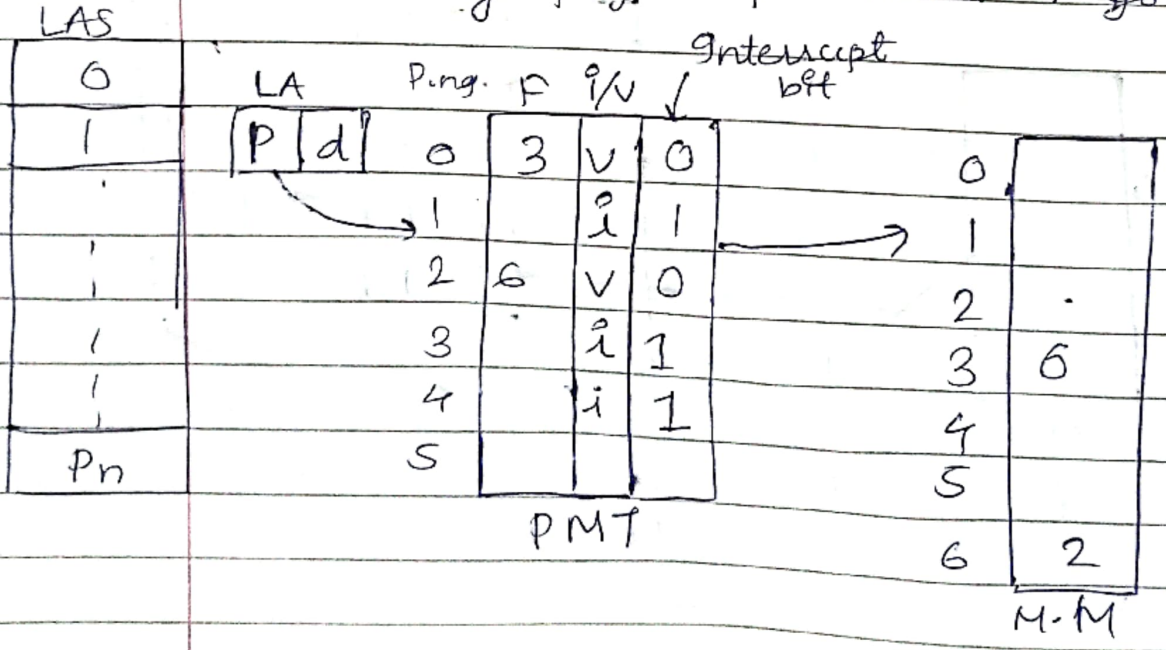
→ Other bit will be 'VALID' or 'INVALID' for protection of job.

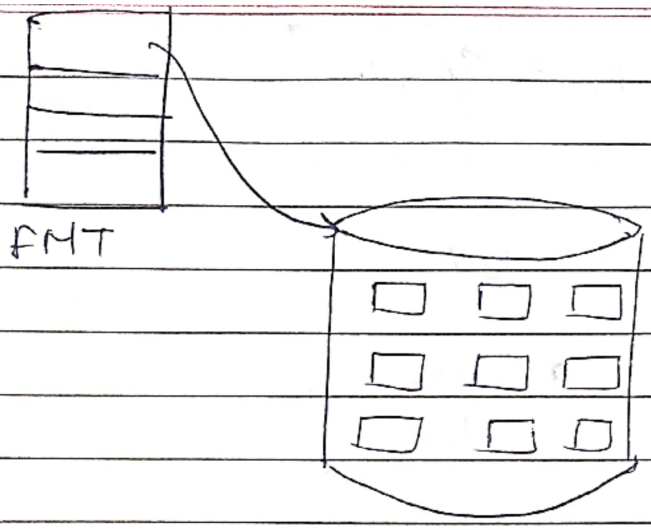
→ File map table (FMT) give address of secondary storage containg pages of logical address space.

→ In PMT there is a bit called interrupt bit, if its '0' means page in M/M, else if its '1', the page is not in M/M & its a 'page fault'.

→ Now OS goes to ~~M/M~~ FMT & gets address of the page & loads it into M/M.

→ If a frame is not free then it can be freed using page replacement Algo.






PMT (Page map table)

## FRAME ALLOCATION:

Fixed allocation:

~~Give eq~~

→ Equal allocation:

→ Give each process equal ng. of frames.
For e.g. If there are 100 frames & 5 process
∴ the give each process 100/5 = 20 frames

Disadvantage: Every process may not need 20 frames, so there is a wastage of frames.

→ Proportional Allocation:
Allocate acc. to size of process
$s_i$ = size of process $p_i$
$S = \Sigma s_i$
m = total ng. of frames

$a_i$ = allocation for $p_i = \frac{s_i}{S} \times m$

$m = 64 \quad s_1 = 10 \quad s_2 = 127$

$a_1 = \frac{10}{137} \times 62 = 4 \qquad a_2 = \frac{127}{137} \times 62 \approx 57$

# PRIORITY ALLOCATION:

It assigns frames based on no. of page allocations & processes.

Global replacement allocation:
When a process req. a page which is not in M/M mem, it can bring in the new page & allocate it a frame from the set of all frames, even if that frame is currently allocated to some other process; i.e. one process can take frame from another.

Advantage - Does not hinder performance of processes & hence results in greater system throughput.

Disadvantage: The page fault rate can be solely controlled by the process itself. The pages in memory for a process depends on the paging behaviour of other processes too.

Local replacement:
When a process needs a page which is not in memory, it can bring in the new page & allocate it a frame from its own set of allocated frames only.

*

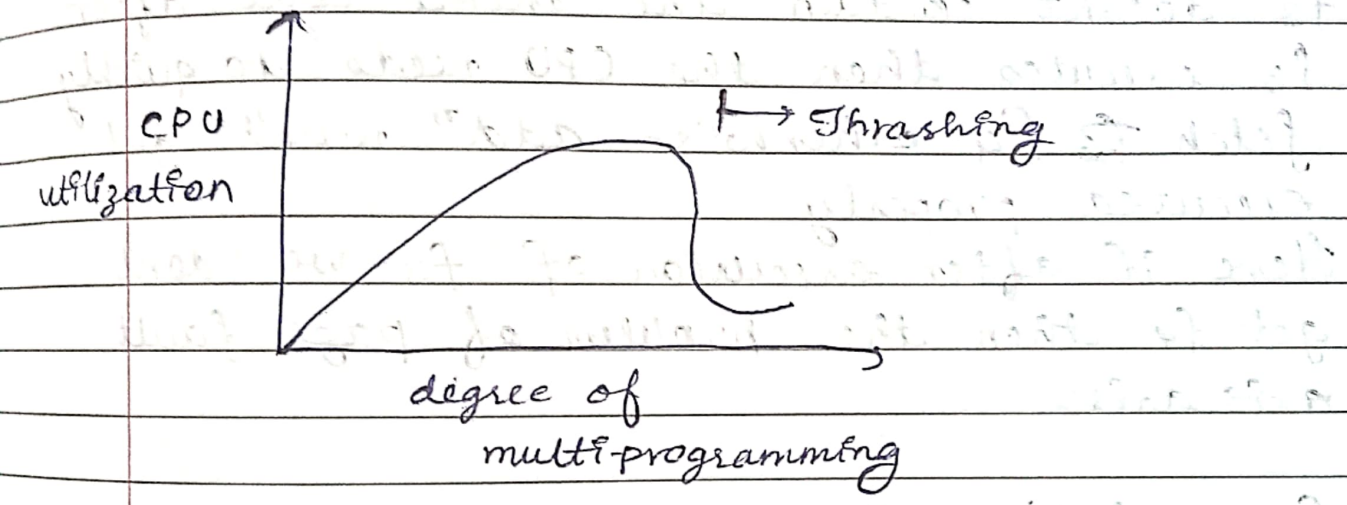
Advantages: The pages in memory for a past process & the page fault ratio is affected by the paging behaviour of only that process.

Disadvantage: A low priority process may hinder a high priority process by not making its frames available to the high priority process.

# Thrashing:

Thrashing is when page fault & swapping happens very frequently at a higher rate, & then the OS has to spend more time swapping these pages.

→ CPU utilization is reduced




Page fault: If program ~~tries~~ attempts to access data or code in it's address space but is not currently located in system RAM.

Swapping: On the occurence of a page fault, OS tries to fetch that page from secondary memory & try to swap it with one of the pages in RAM.