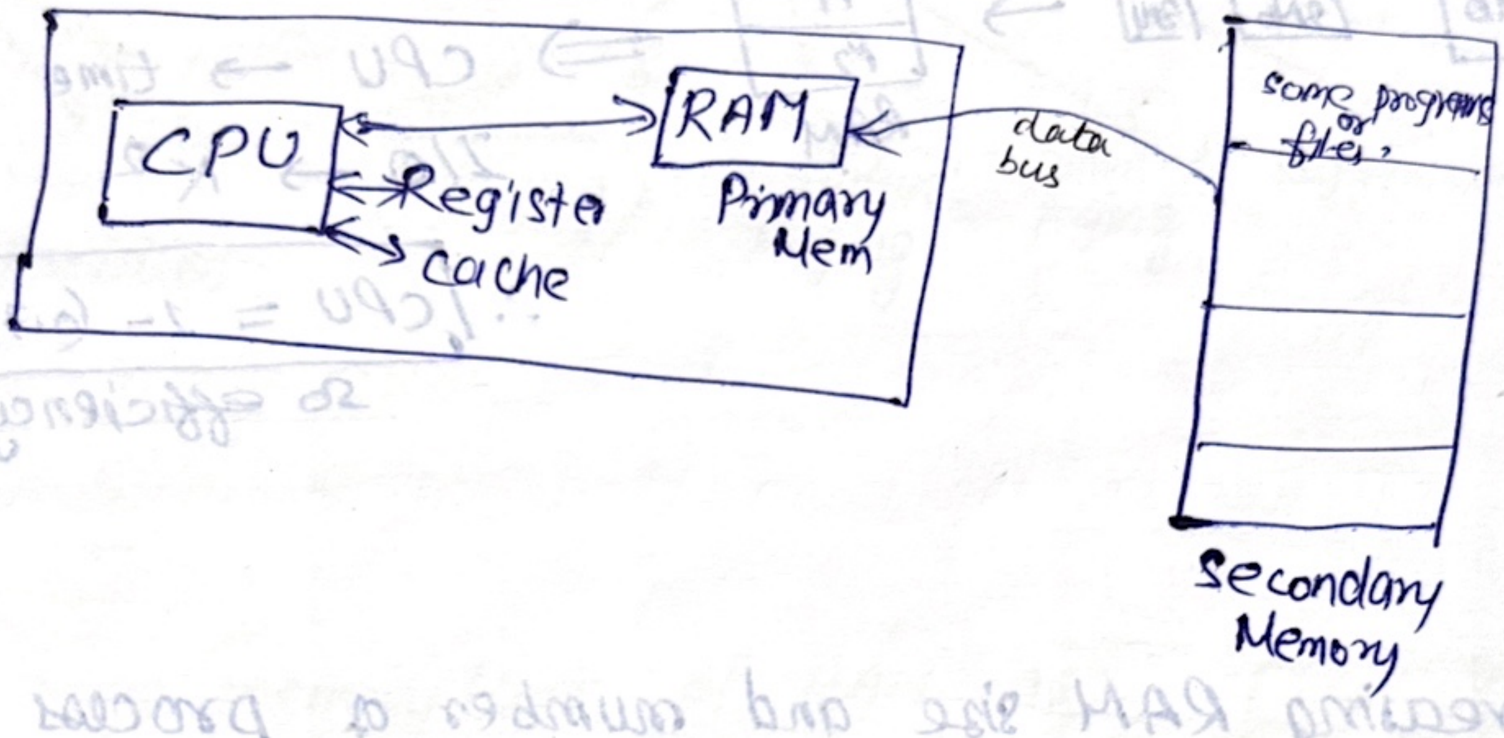
(38) (#) Memory Management & degree of Multiprogramming

⇒ How manage memory in ous system iis done by OS.

⇒ Mostly we will deal with primary memory management (RAM).

⇒ Goal :- Efficient utilization of memory.



⇒ Register :- is one small set of data holding, part of computer Processor (CPU). A register may hold an instruction, a storage address, or any kind of data. we could also use register like ram but its size is small so we can't do that. they are fastest but size in kb.

(Multiprogramming vs Multitasking)

⇒ Multiprogramming :- Means when we are trying to run some program then try to bring as much as possible process in RAM. This is called multiprogramming. (multiple programs)

⇒ Advantage :- More process (program) in RAM, Higher the utilization of CPU. If CPU Idle then performance down.

⇒ e.g. Process → CPU
→ I/O → If here then so it will we some other process in CPU.

⇒ Degree of multiprogramming :- More & More process in RAM and in CPU.

e.g. How more process in RAM can increas efficiency of CPU.

(i) 8Mb RAM — Process 8Mb P_1 ⇒ RAM ⇒ CPU → time (1-K)

I/O → time K

⇒ If K = 70% then

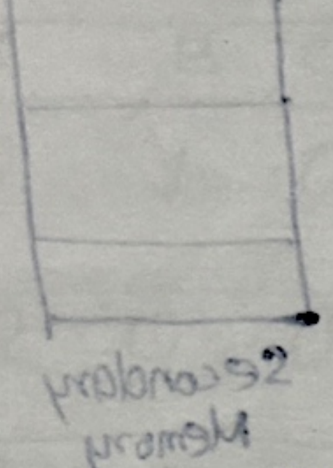
CPU = 30%

if (ii) 16Mb RAM — P_1 8Mb, P_2 8Mb → RAM ⇒ CPU → time $1-K^2$

I/O → K^2

$\therefore$ CPU = $1-(0.7)^2$ = 76%

So efficiency increase.



⇒ So increasing RAM size and number of process in RAM then we can increase the efficiency of CPU.

If 4 process CPU = $1-K^4$

$1-K^n$ n Process

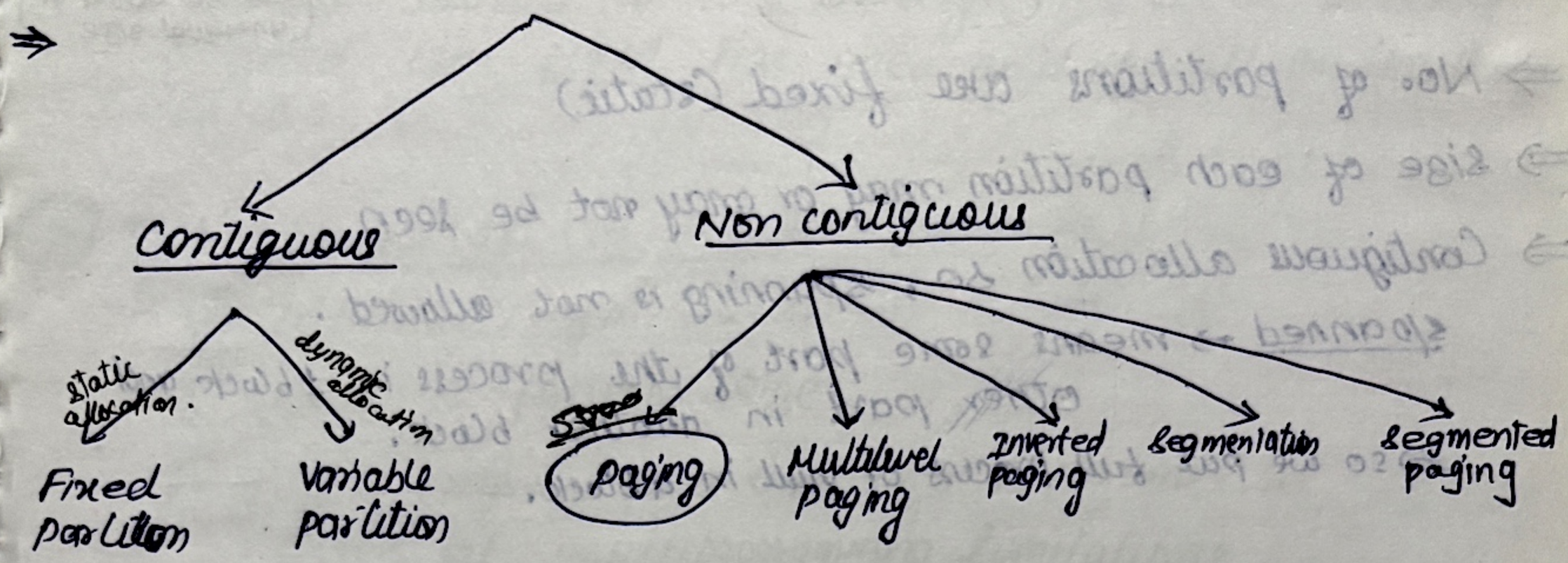
→ If all of a time will I/o then, we got approx = 99% efficient.

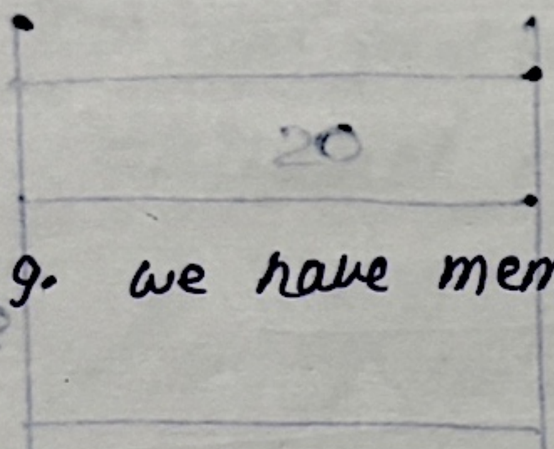
⇒ To bring more process we need to increase size of the RAM.

So

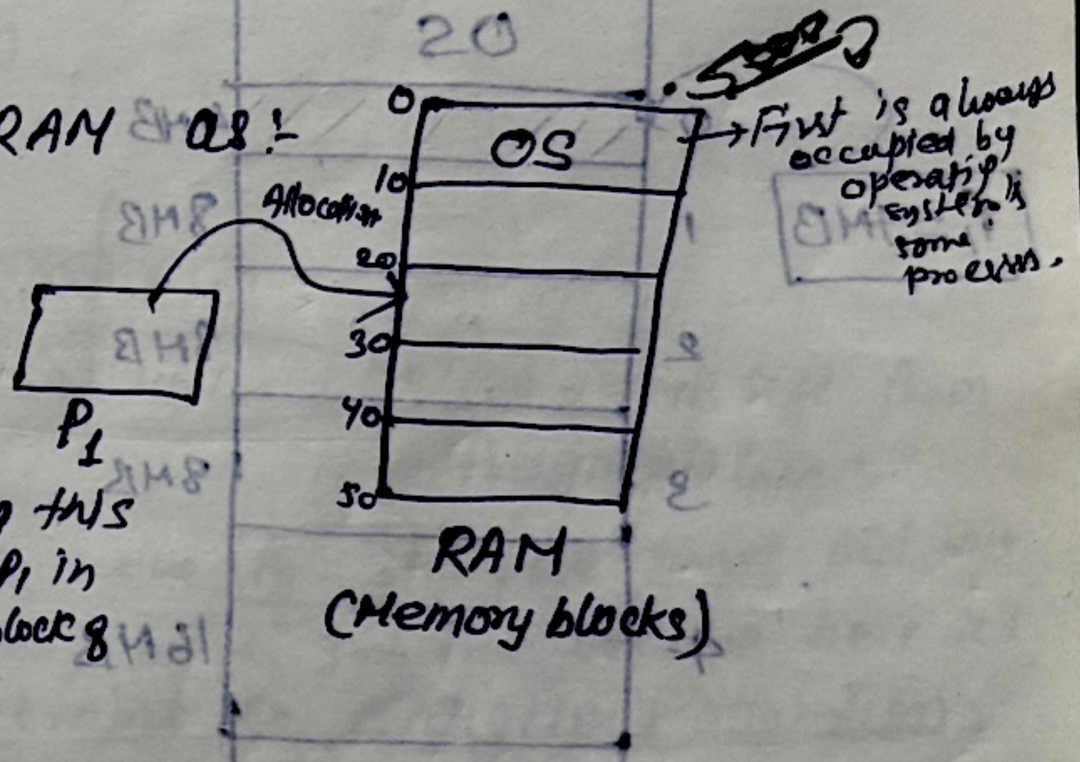
⇒ so memory management of RAM is most important.

89 # Memory Management Techniques :-





e.g. we have memory block of RAM as :-



∴ (1) contiguous → In this we fill process in continious address. manner in memory block of RAM

(a) fixed partition → we already make the fixed block size of RAM

(b) dynamic partition → during runtime only first check size of process and allocate that much size block in RAM and allocatee to it.

10=00

⇒ contiguous is absolete. Nowadays we use non-contiguous.

(2) Non contiguous → we can put process in different location in memory. Not putting in continous.

(40) # Fixed Size Partitioning / Internal Fragmentation

(Static)

5* (can be equal or unequal size)

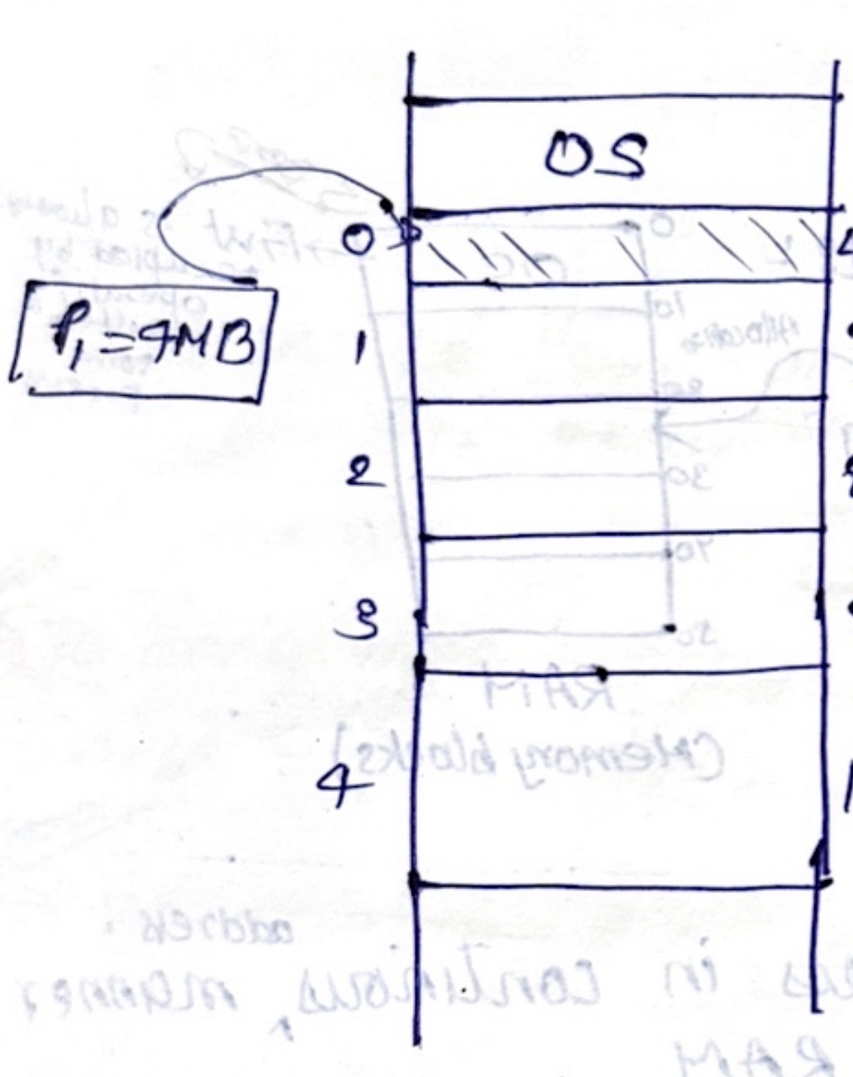
⇒ No. of partitions are fixed (static)

⇒ Size of each partition may or may not be seen

⇒ Contiguous allocation so, spanning is not allowed.

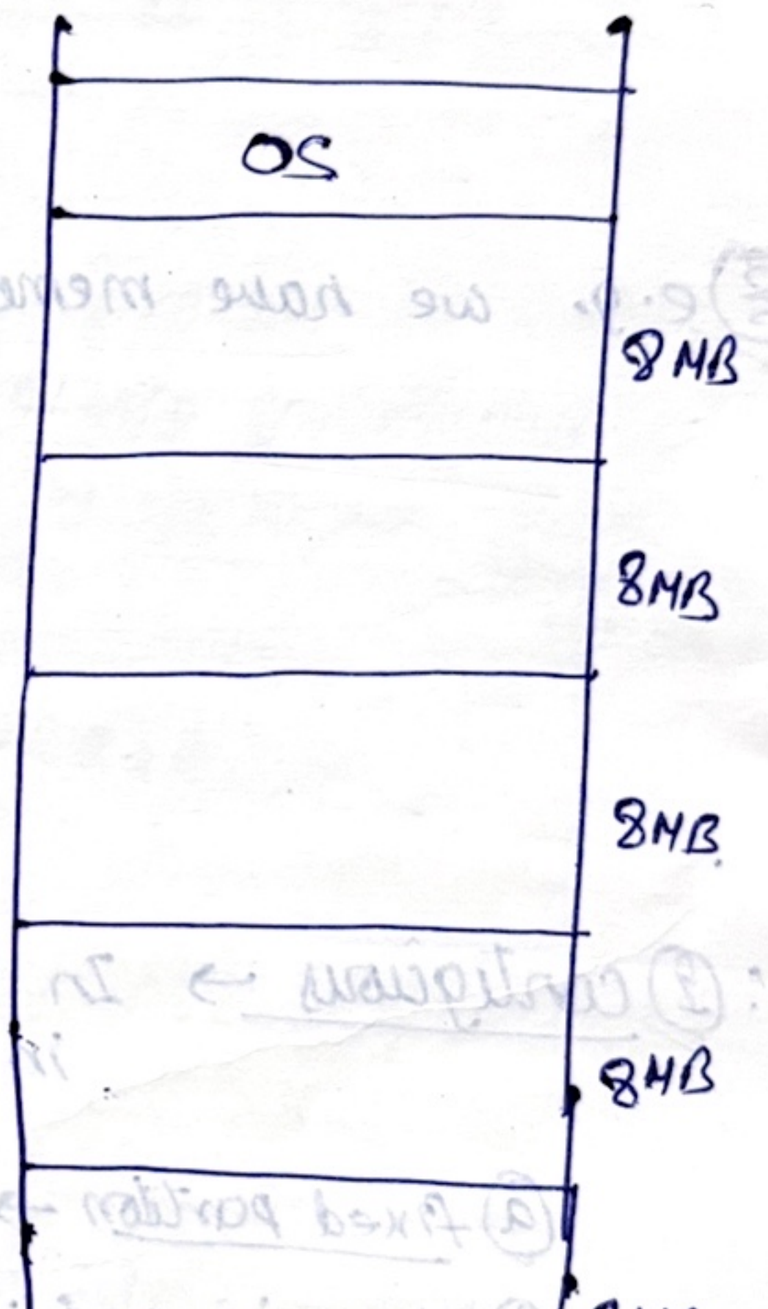
Spanned → means some part of the process in 1 block and other part in another block.

⇒ so we put full process or null in a block.



5 partitions

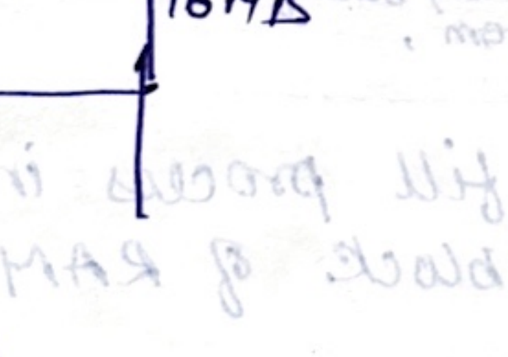
(fixed size ~~varied~~ equal)
unequal size



5 partitions

(fixed size ~~varied~~ all equal)

But if $P_1 = 2MB$ →

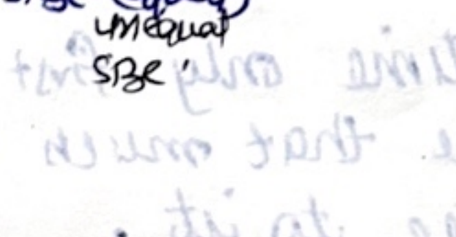


Now we cannot put other process here by spanning in each block 1 process only.

So,

↳ So this is called internal fragmentation.

also again if $P_2 = 7MB$



⇒ Also if we've process of size 20MB then we cannot allocate in the memory since we've already decided the fixed size memory block all are less than 20MB.

Disadvantages (For both fixed size & variable size) :-

① Internal fragmentation

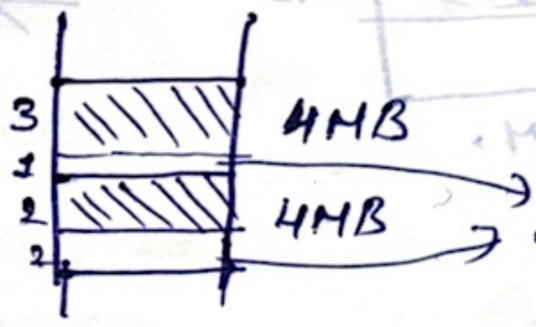
② Limit in process size.

③ Only 5 programs at a time since 5 partitions. So degree of multiprogramming limitations.

④ In vatta fixed equal size one more limitation & process size.

④ External fragmentation

Now we've situation



∴ 1+2 = 3 MB left from fragmentation.

So if we've $P_3 = 3MB$ we cannot put this although we've 3mb free because here it's constraint to contiguous allocation.

⇒ And this problem is called external fragmentation.

⇒ External fragmentation :- Although we are having available of the memory in different slots or block and on addition of all these is equivalent or greater then new process size, but still we are not able to accomodate, because of the contiguous method.

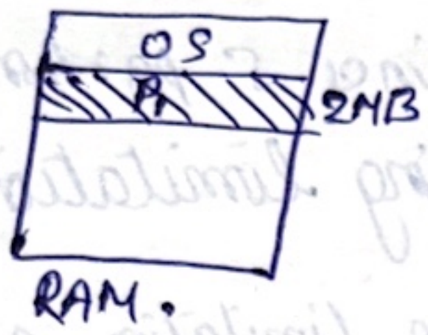
Advantage :-

① Easy to implement :- Because already decided number of slots. allocation & deallocation is easy method.

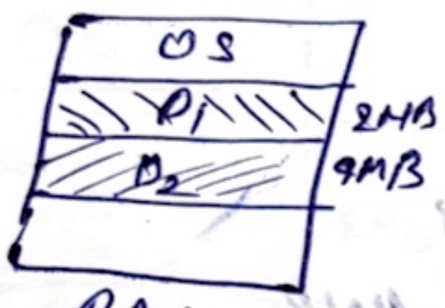
41 # Variable size partitioning :- (Dynamic (runtime) partitioning)
(Dynamic)

⇒ In dynamic size partitioning we keep the whole memory empty, non partitioned and whenever a process comes in then we will allocate memory to it dynamically as its size.

⇒ e.g. $P_1 = 2MB$



$P_2 = 4MB$



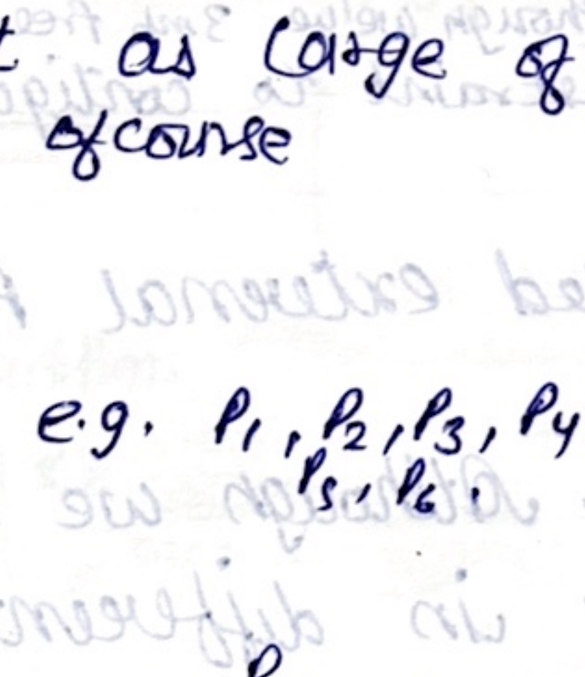
5 ⇒ So here no chance of internal fragmentation.

⇒ Also no limitations on number of process.

⇒ We can put as large of process size but less than RAM size ofcourse

⇒ Problem :- e.g. P_1, P_2, P_3, P_4 in RAM as P_5, P_6.

Now if P_2, P_4 is done than.

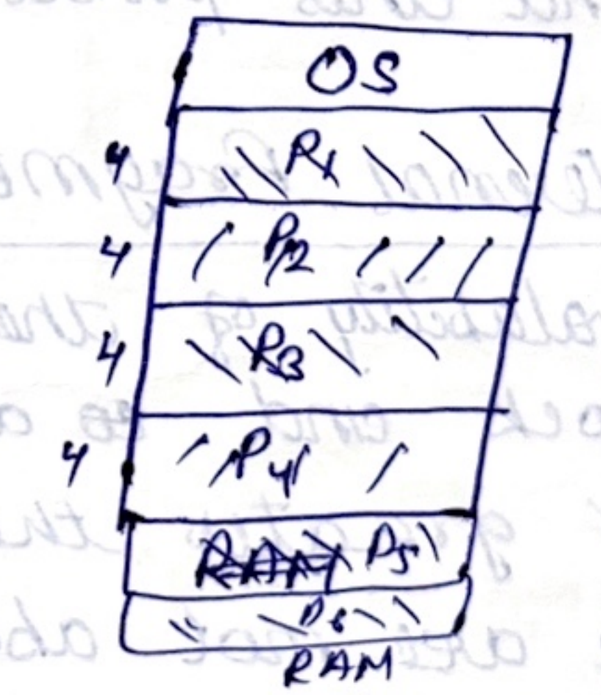


Hole created. (8mb space) available

and we've new process $P_7 = 8mb$ we cannot put this because it should go in contiguous only.

So 8mb process won't go.

So external fragmentation still occurs here.



⇒ Now how to remove this external fragmentation :-

⇒ By using compaction method :- Means moving empty spaces (holes) in one side and occupied in another side together. so it's like copy paste of address.

⇒ But this is undesirable bcoz first we've to stop all working process and takes a lot of time.

(5) Disadvantages :-

(1) External Fragmentation

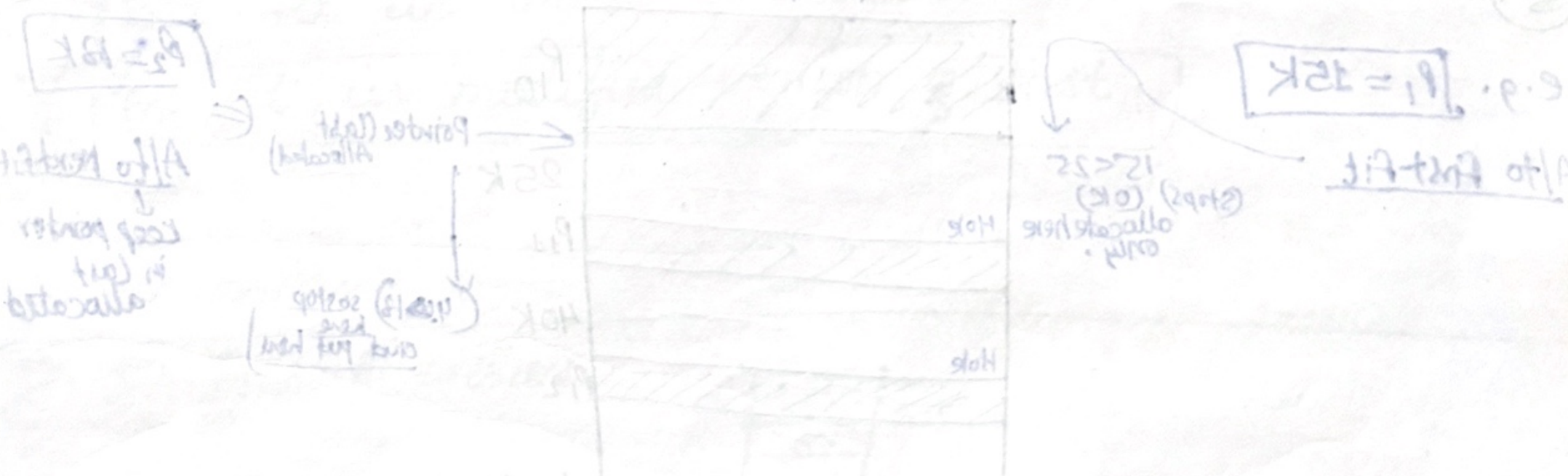
(2) Allocation / Deallocation (After we're hole) so. to remove this we use bitmap, LinkedList to manage this.

(6) Advantages :-

(1) No internal Fragmentation

(2) No limitations to number of process

(3) No process size limitations.



(42) (#) First Fit, Next Fit, Best Fit, Worst Fit Memory Allocation:-

⇒ Now how to allocate process in these holes. we will dealt about it.

⇒ we've four algorithms :-

(I) First-Fit ⇒ Allocate the first hole that is big enough.

(2) Next-Fit ⇒ Same as first fit but starts search always from last allocated hole.

(3) Best-fit ⇒ Allocate the smallest hole that is big enough.

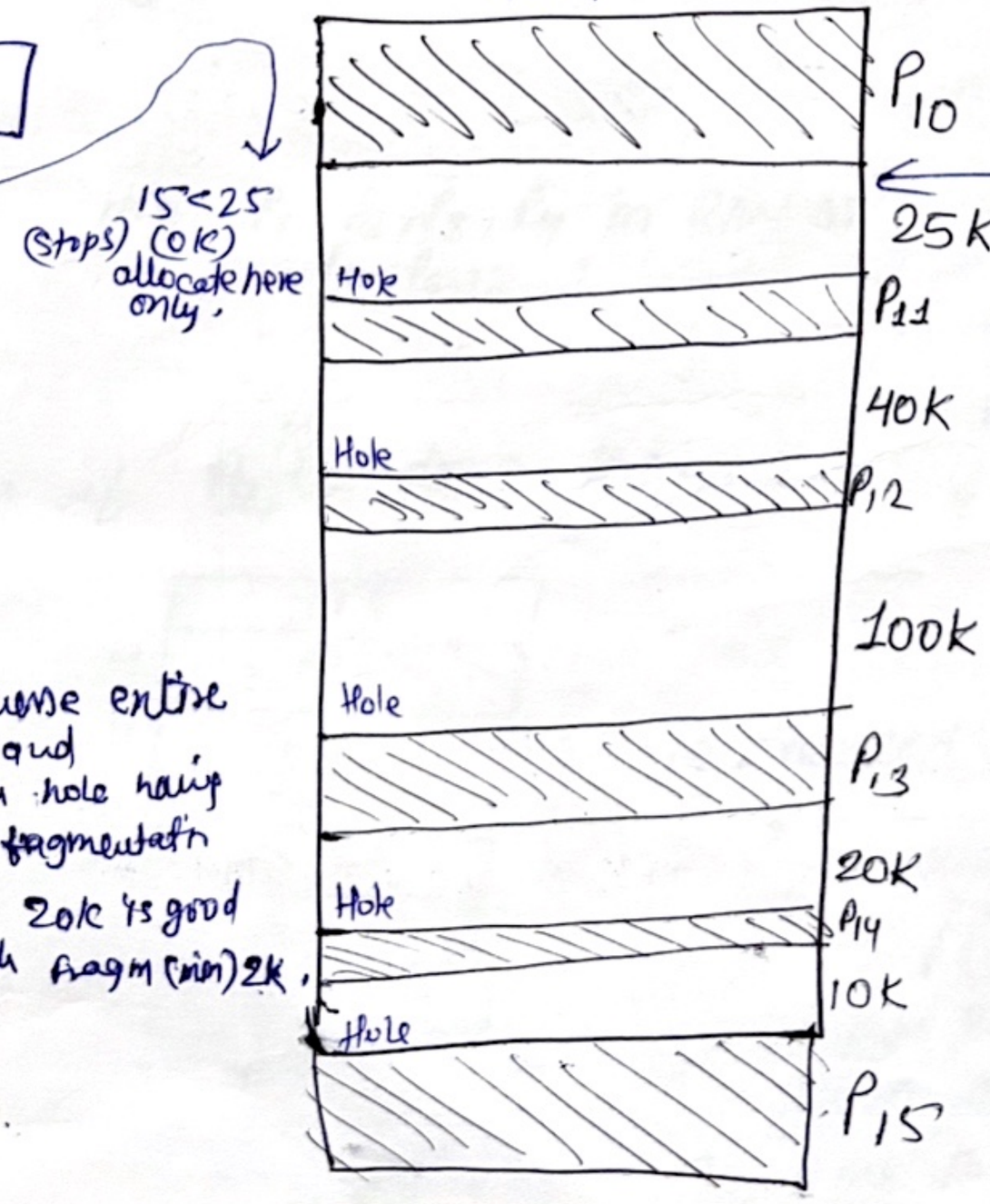
(4) Worst-fit ⇒ Allocate the largest hole.

---

(5)

e.g. $P_1 = 15K$

A/to first-fit



Best fit → Traverse entire list and search hole having minm fragmentation

e.g. $P_3 = 18K$ 20K is good with fragm (min) 2K.

Worst-fit

- Look for largest hole. (reverse of best fit)

e.g. here 100K

$P_4 = 12K$ ∴ 82K

Internal Fragmentation

start search from 0

⇒ ∴ First-fit → Advantages → very fast (since first only)
→ Disadvantage → still fragmentation changes.

⇒ Next-fit → Advantage → further faster than first-fit. start search from last allocate instead from 0.

⇒ Best-fit → Advantages → very less internall fragmentation or no fragmentation.
→ disadvantage → will create very little interfragment so they are of no use for large size process
→ Search the whole list. so search is slow very slow

⇒ Worst-fit → Advantage → None.
→ Disadvantage → search very slow) large size holes.

⇒ Best & worst are opposite of each other.

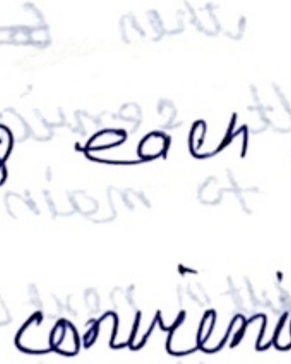
⇒** First-fit is most convenient Alto time.

⇒ Next-fit is modified version of first.

5 ***

(43) (44) # worst fit questions

125 10, 51K, 15K
Put and see for various Algo so that we get min fragmentation those questions



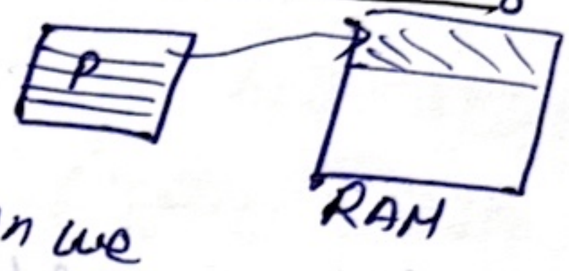
45 # Thrashing in Operating System :- (Thrashing, Pagination in RAM, Page Fault)

⇒ Thrashing is directly linked with degree of multiprogramming.

⇒ More process in RAM more CPU utilization.

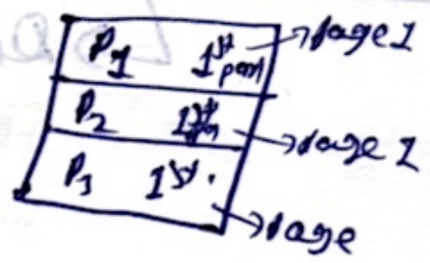
But sometimes process demands for I/O requests then in that case CPU is left Idle and hence throughput decreases.

⇒ So to handle this we makes use of concept of paging. In this concept we divide process in pages and then bring it in ram. So we will bring only some part of process inside RAM and not fully.



so if CPU ask for other part of process then we can give to it.

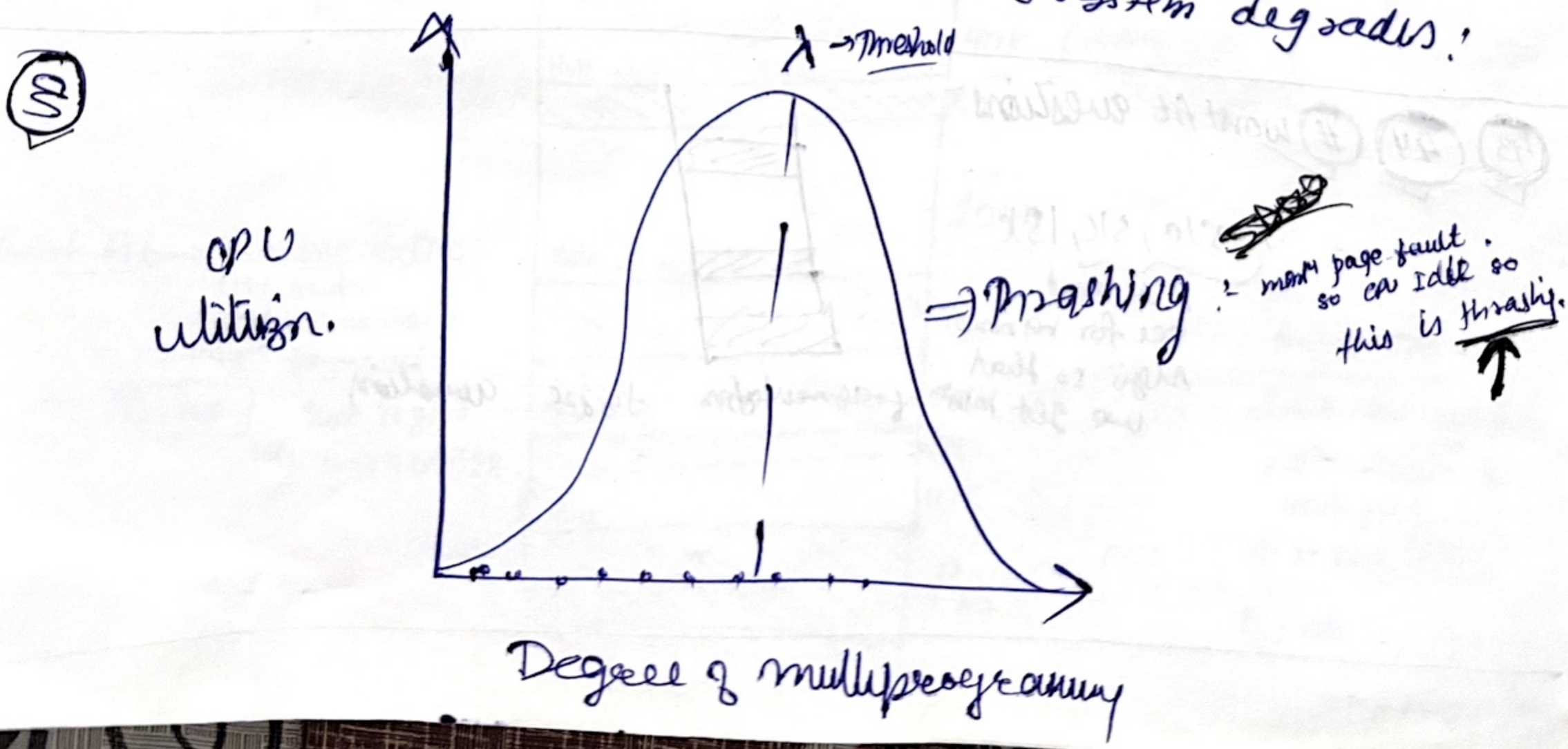
⇒ But in worst case scenario:-



If now CPU requires other part of these process P_1 & P_2, or P_3 then page fault occurs.

So we use page fault service time. Means bring that required page from harddisk to main memory and it will take lot of time. (See page fault in next lecture). If page required not found then page fault occurs.

⇒ So in here we've maximized degree of multiprogramming but CPU demand not present in CPU, so it will take lot of time to service this page fault, so performance of system degrades.



So how to remove thrashing.

(1) Increase Ram size

(2) Long term scheduler (Bring more process in RAM ready state)

↳ we can control to bring only some limit max process in RAM to minimize page fault.

(3) In OS three types of scheduler :-

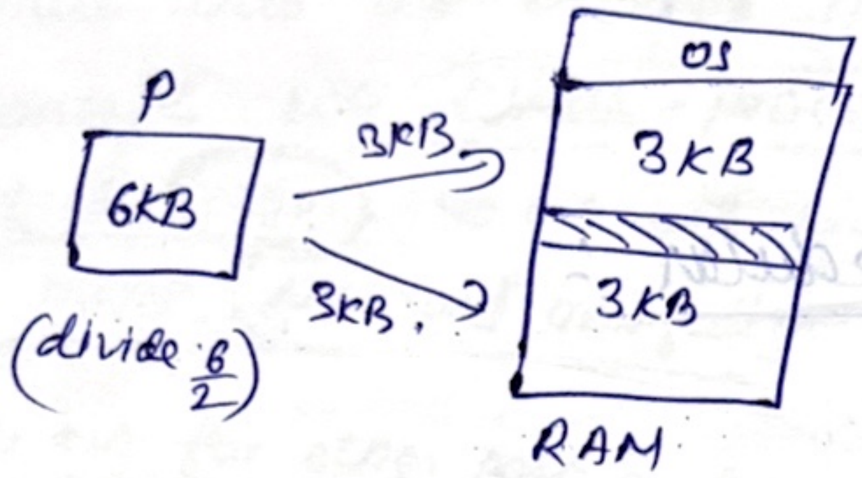
(1) long term
(2) short term
(3) middle term.

(46) # Non-contiguous memory allocation

Need of Paging

⇒ Here process can be spanned (divide) and then put in different locations in RAM.

⇒ e.g.



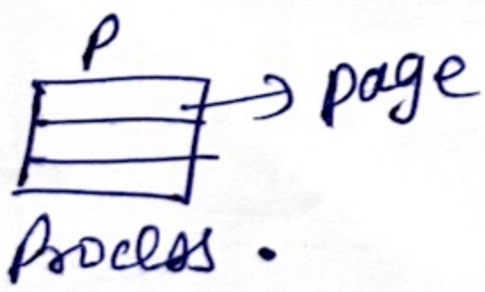
So here we can remove external fragmentation also.

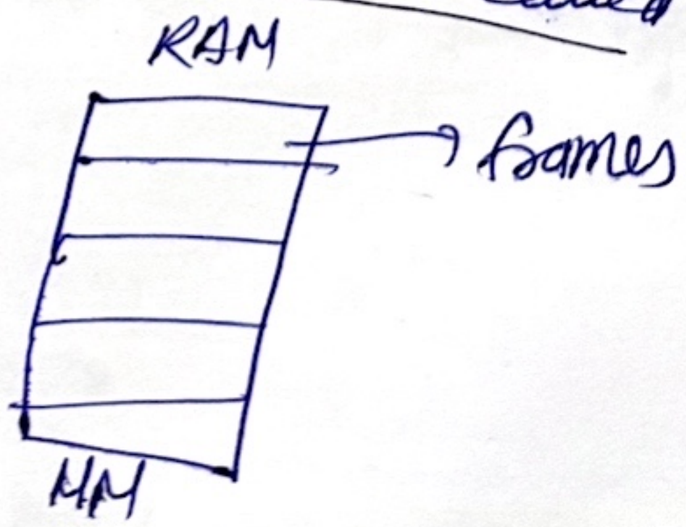
⇒ But this division is very costly method. Because whole are created dynamically so we don't know if again other some process left the memory then so which is to be chosen better. It's very time consuming.

⇒ So to deal with above problem we will already divide the process before bringing it to main memory.

So this partition is called the page.
we are doing this partition inside secondary memory only.

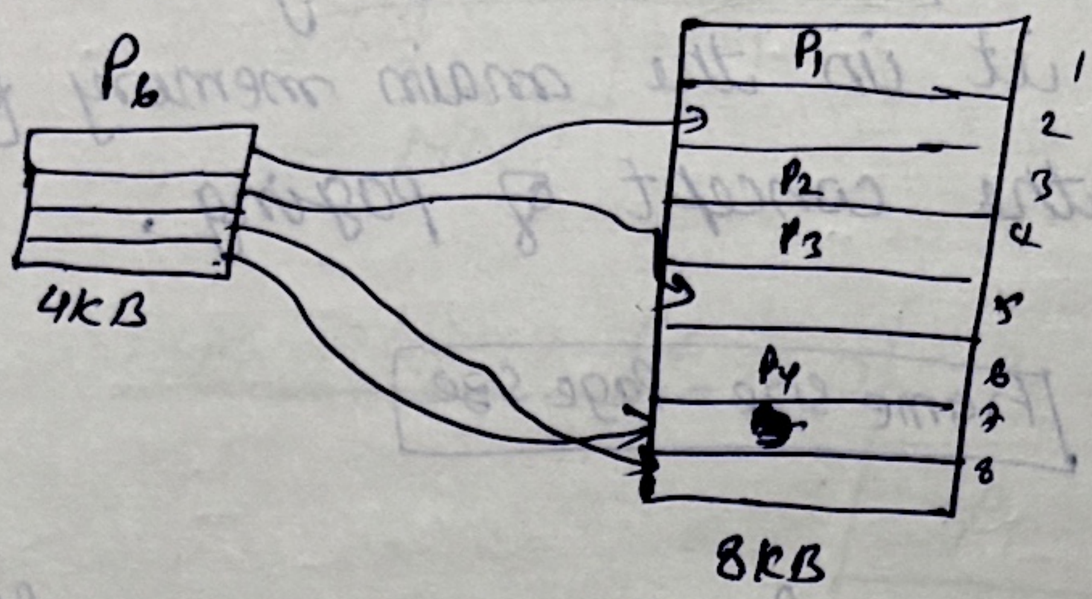
⇒ Also we can also divide main memory in slots called frame.





⇒ **The size of page = The size of frame**

eg



Sol:

So by use of paging concept we can remove external fragmentation.

⇒ CPU must aware of paging system is being involved.

So if it asks for page P1, then in MM it is randomly allocated.

To get that 3 we need to do mapping to get that data. eg consist of P3.

And this mapping is done by Memory Management Unit - (MMU).

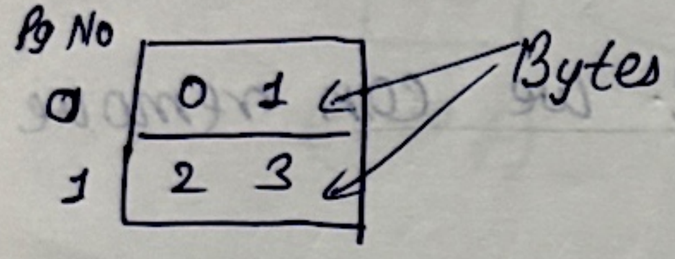
CPU

(47) # Paging / Memory management :-

⇒ In paging; we divide process in equally sized pages and then put it in the main memory frames. So this is the concept of paging.

Frame size = Page size

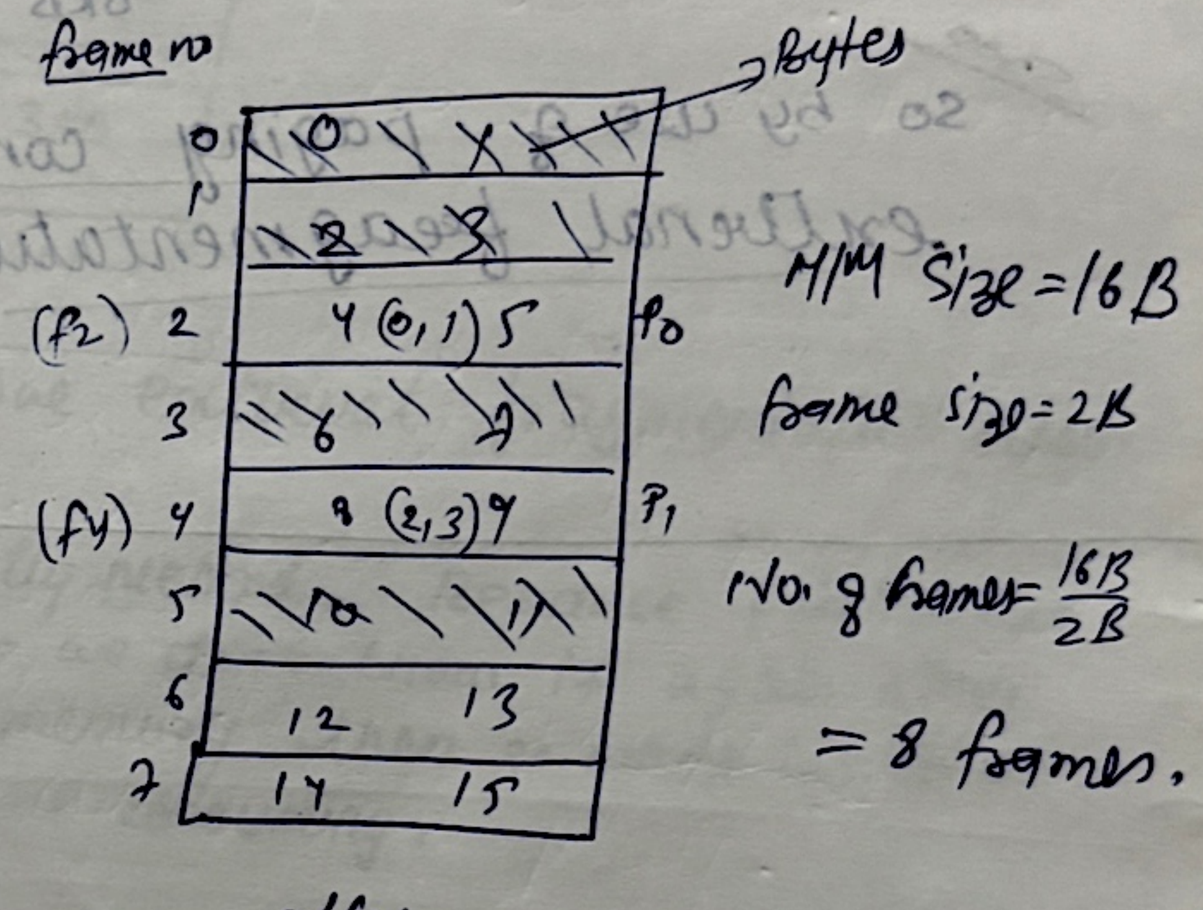
e.g.



Process size = 4B

Page size = 2B

No. of pages = $\frac{4B}{2B}$ = (2)



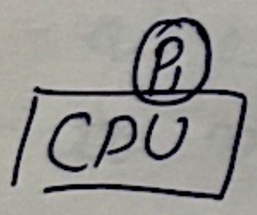
M/M Size = 16B

Frame size = 2B

No. of frames = $\frac{16B}{2B}$

= 8 frames.

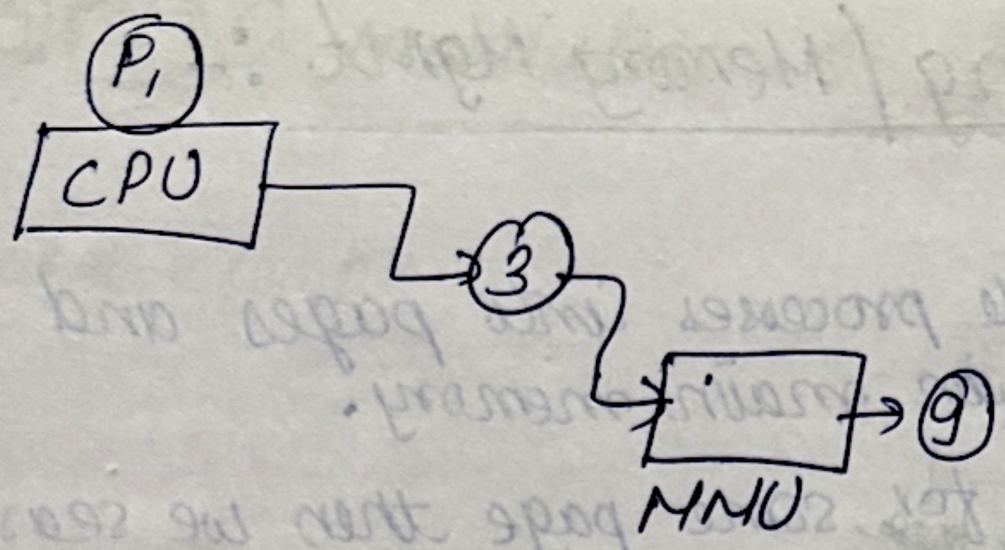
M/M



⇒ CPU not aware of Paging system is being involved. So if it asks 3rd Bytes of (P1), then in M/M it is randomly allocated. To get that 3 we need to do mapping to get that data. e.g. convert 2 to 3. (means 9 present in MM so this will replace 9 to 3 from P1 page in Second Memory)

And this Mapping is done by using Memory Management Unit. (MMU).

So for that MMU use page table. Page table contains frame number.



Page Table & P_1 ⇒

0	f2 → frame number
1	f4

P_1

⇒ CPU always works on logical address :-
(logical address → page no + page offset.)

Logical Address

Page No.	page offset.

↓ Page size 2=(0,1)

⇒

1	1

0 (0)	1 (1)
2 (0)4	3 (1)5

∴ we got ③

⇒ Physical Address :- 4bit (0-15)
For MainMemory

Frame No.	Frame offset
(0-1)	(0-1) → Frame size (0-1) = 2.

MM.

so understood.

(48) # Question explaining on paging.

(49) (#) Inverted Paging / Memory Mgmt :-

⇒ In normal paging we divide processes into pages and bring few of those pages in main memory.

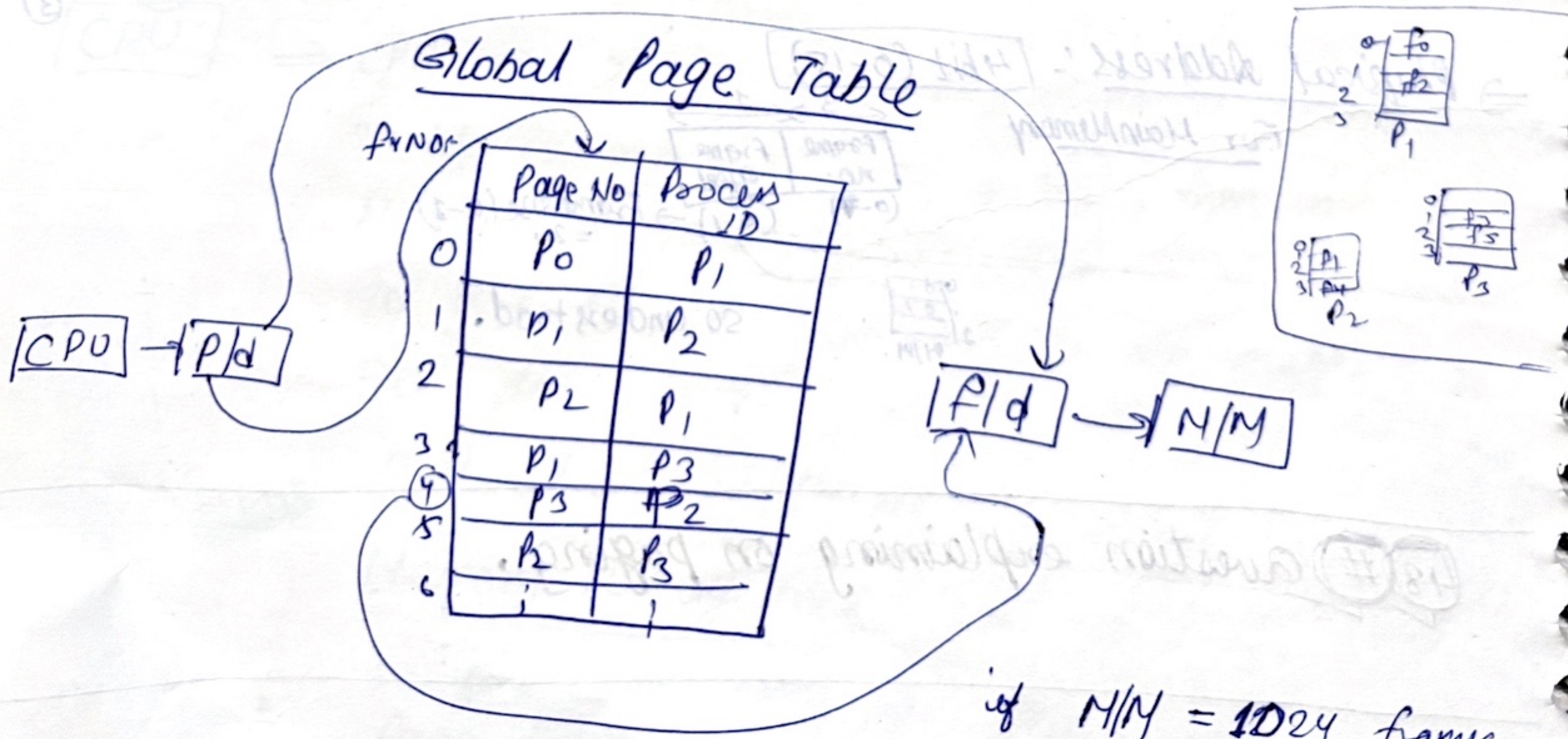
⇒ And whenever CPU demands for some page then we search for it in page table. Each process has it's page table. And from their we fetch the bytes in M/M.

∴ (1) Each process has its own page table.
(2) Also this page table must be in Main Memory in some frame.

↳ So by keeping many page table for many process in main memory is only taking or occupying unnecessary space in our limited Main Memory.

⇒ So to remove such things, concept of inverted Paging comes into play.

⇒ In this we make a single global page table for all the process.

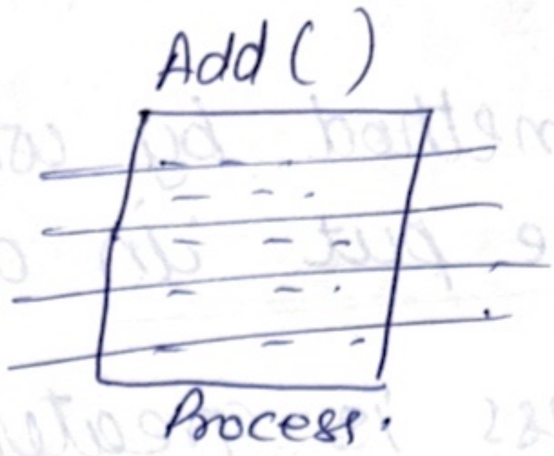


⇒ Searching Time (disadvantage) (Linearly) (So it's not so popular).

if M/M = 1024 frames
No. of entries in Inverted Pag Table = 1024.

(50) (#) Segmentation vs Paging | segmentation working

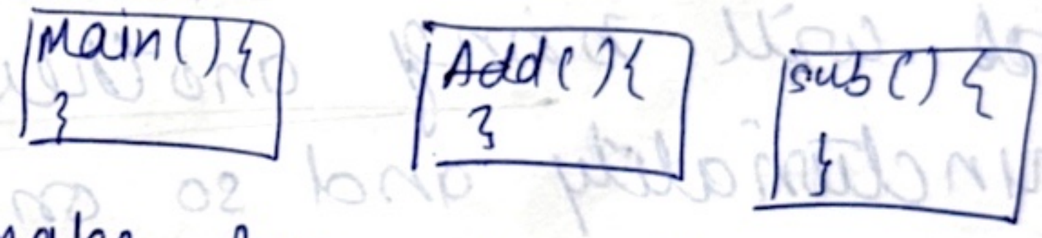
⇒ Take example code :-



⇒ Paging will randomly divide this process code without knowing that if some part of code is to be kept in same.

⇒ whereas Segmentation is same like paging, but here division is based on the user criteria. so that same code remains in same page (segment) and page fault can be minimized.

It will make segment of

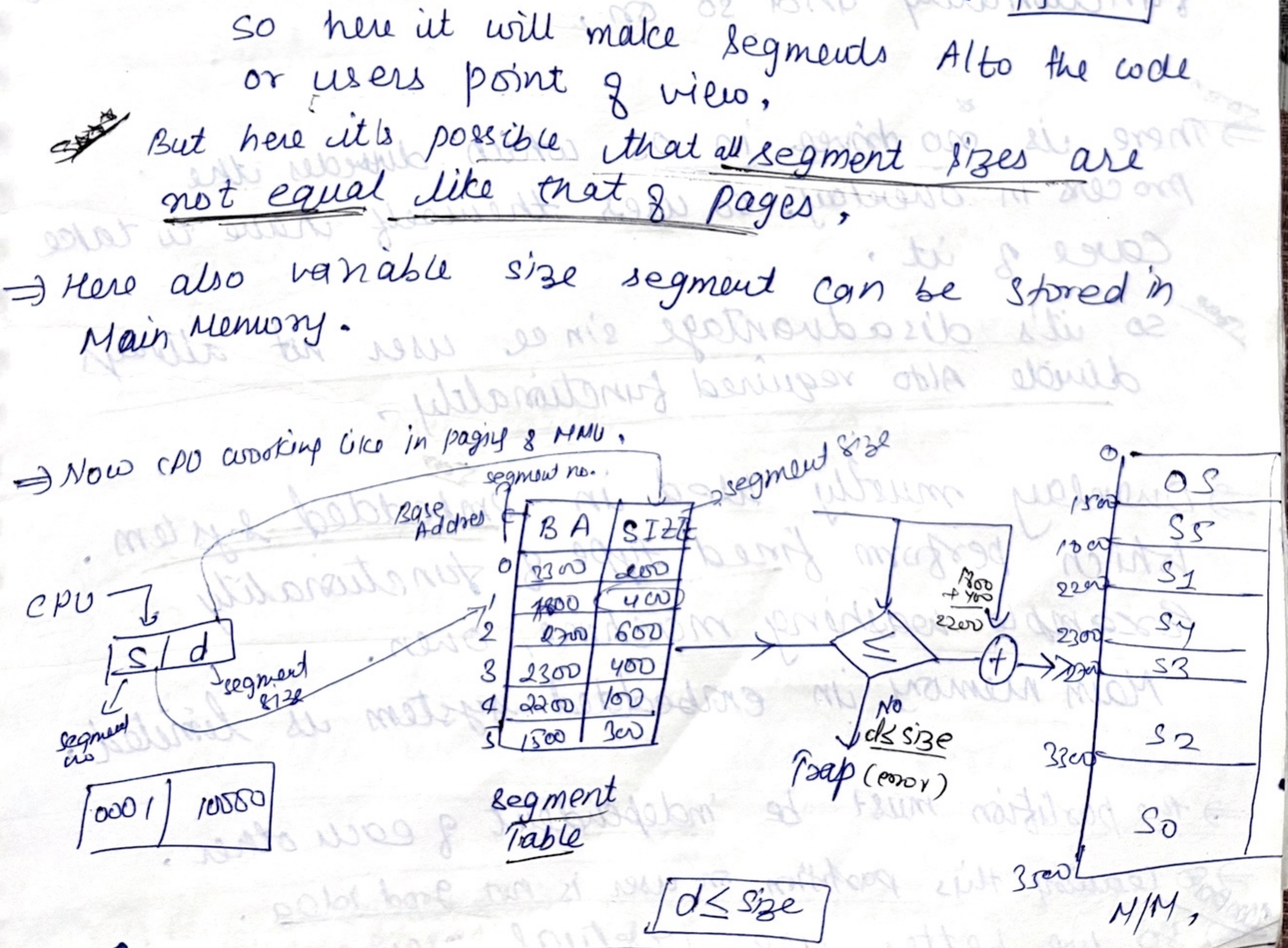


so here it will make segments A to the code or users point of view.

But here it is possible that all segment sizes are not equal like that of pages.

⇒ Here also variable size segment can be stored in Main Memory.

⇒ Now CPU working like in paging & MMU.



51 # Overlay / Memory Mgmt :-

⇒ Overlay is a method by which a large size process can be put in a main memory.

⇒ If size of process is greater than main memory than still we can put accomodate our process in main memory by the concept of overlay.

⇒ Using this overlay concept :- firstly we will divide that large size and bring only those usable functionality in our main memory.
If the currently partition work is done then work will bring another partition and use that functionality and so on,

⇒ There is no driver in OS which divides the process in overlay. so user themself have to take care of it,

⇒ so it's disadvantage since user not allways divide Also required functionality,

⇒ Overlay mostly used in embedded system. which perform fixed type of functionality,
Example washing machine, oven.
Main memory in embedded system is limited,

⇒ the partition must be independent of each other,

⇒ so leaving this partition on user is not good idea,

⇒ so we better use virtual memory concept,

Q Consider a two Pass assembler pass 1: 80 KB, Pass 2: 90 KB.
Symbol table : 30 KB
common routine : 20 KB.

At a time only one Pass is in use
what is min partition size required if overlay driver is 10 KB size.

80 KB
90
30
20
10
× 230 KB

Pass 1 (Memory required)
30
30
20
10
140 KB

Pass 2
90
30
20
10
150 KB

Max is chosen

Pass 2 150 KB

∴ Min Partition = 150 KB.

100%

⇒ so overlay is used in Embedded System. for real memory or real system we use concept of virtual memory

52 # Question explanation on logical address and Physical address space

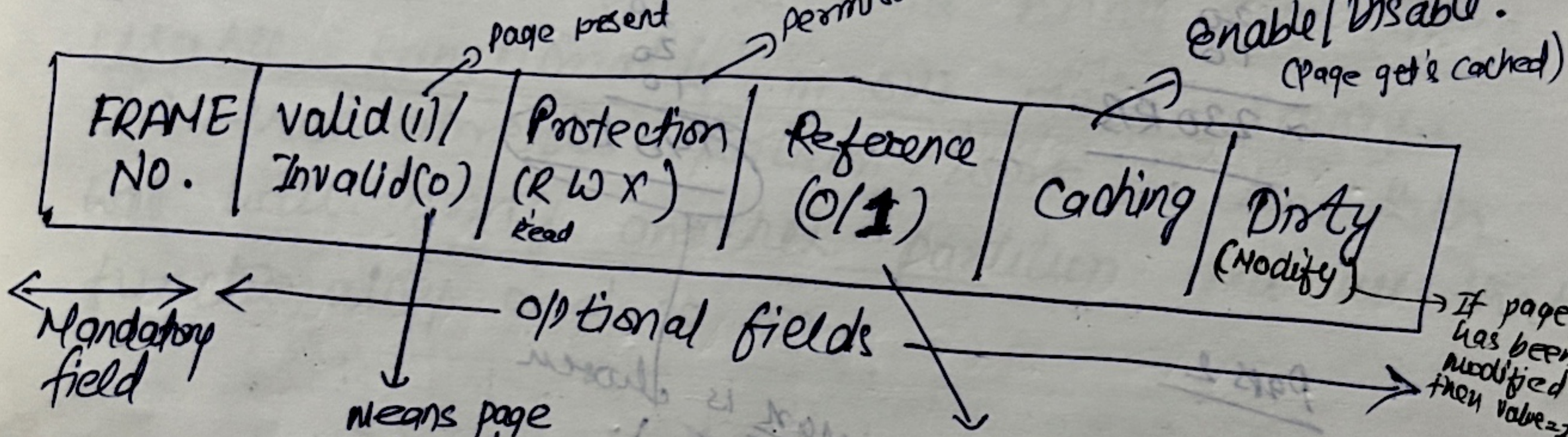
#53 Format of Page Table Entry | Memory Mgmt

⇒ In a page table in singe entry, there are lot of things we've to put.

⇒ we will see here significance of all these values.

⇒ Page table is used by MMU to map physical address to logical address.

⇒ Page table entry :-



(54) # Virtual Memory / Page fault / Significance of virtual Memory

⇒ Virtual Memory provides an illusion to the programmer that a process whose size is largere than the size of main memory can also be executed. (Like mouse lag)

⇒ It also provides an illusion of providing more and more process in Main memory to increase degree of multiprogramming..

⇒ How Virtual Memory Works:-

⇒ So when page demanded by CPU not present in Page table (or Main Memory) then it will generate a page ~~default~~ & fault and trap generated

⇒ So how to remove page fault. ?

where multiple Process are divided into pages.

⇒ so whenever trap (interrupt) generate, control goes from user's process to OS. This is called context switching.
Here we remove the current running process and bring another process to running state.

⇒ Now OS get's actuated and check first for authentication & that demanded page and bring that page from hard disk (LAS = Logical Address space) to Main Memory. And then update it in page Table saying P_1 present at frame 11 in M/M.
Now again control comes back to user.

⇒ Here time taken to do this is called Effective Memory Access Time = EMAT.

Let p' → probability to occur page fault.

then

$$EMAT = p\ (\text{page fault service time}) + (1-p)\ (\text{main memory access time (MA)})$$

(in milliseconds) (in ns (nanosecond))

So this is how things work where we seem like all the pa process page is present for a large program in our MainMemory (Page Table).
So this is how concept of Virtual Memory works. This swapping happens so fast that it looks like all are present.

(55) # Page Replacement Algorithms :-

(Virtual Memory) Page Replacement from Main Memory to virtual Memory (A part of secondary memory)

In virtual Memory

⇒ We never bring all parts of the single process in main memory. We just provide illusion to the programmer through virtual memory.

⇒ So we only bring few pages of those processes. When their need is finished then we remove them from main memory and bring those whose are demanded.

⇒ So to replace this pages there are three methods.

Page Replacement algo :-

1. FIFO (First In First Out)
2. Optimal Page Replacement
3. Least Recently Used (LRU).

(The main motive of Page Replacement Algorithm is to minimize Page fault)

(#) We've been given 3 frames in Main Memory. (For understanding problem let)

f_1		7	1	1	1	1	0	0	0	3	1	3	3	2	2
f_2		0	0	0	0	3	3	3	2	2	2	2	1	1	1
f_3	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0
	* (Page fault)	* (Page fault)	*	*	Hit	*	*	*	*	*	*	*	HIT	*	HIT

(Let's see. How many page fault and page hit present here in this problem)

FIFO → whichever frame comes first replace with that and so on.

Reference string :- (P1) (demands by CPU)

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 1, 2, 0

(0 present in M/M (Page Hit))

* → Pagefault → 12
HIT → Page Hit → 3

Let's see How FIFO works here.

Now first CPU demands page '7'. Since it's not present in Main Memory initially hence it's page fault.

(8)

$$\text{Hit ratio} = \frac{\text{Number of Hit}}{\text{Total number of reference}} = \frac{3}{15} \times 100$$

$$\text{Miss ratio} = \frac{\text{Number of Pagefault}}{\text{Number of reference.}} = \frac{12}{15} \times 100$$

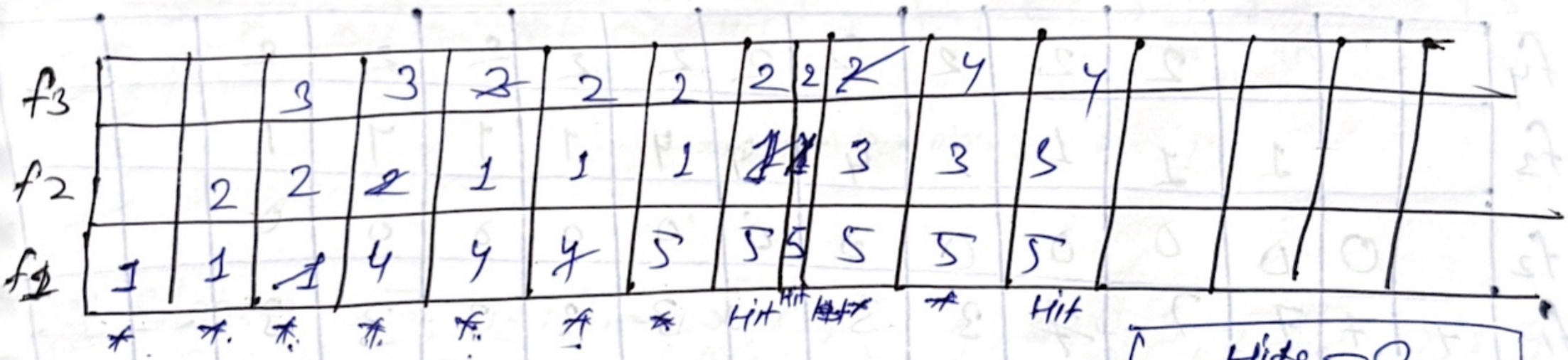
(56) # Belady's Anomaly in FIFO page replacement with Example :-

⇒ Belady's anomaly is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns.

This phenomenon is commonly experienced when using the first-in-first-out (FIFO) page replacement algorithm.

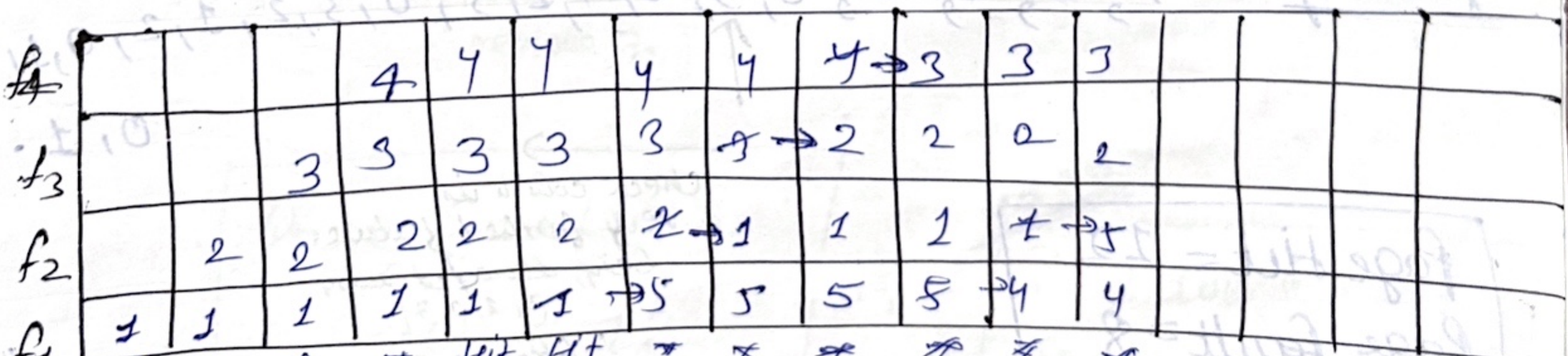
Using FIFO Page Replacement Algo

e.g.



Hits = 3
Page Fault = 9

Ref → 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.



Hits = 2
Page Fault = 10

(57) # Optimal Page Replacement :-

All this about {to bring page from Hard disk to M/M}

⇒ In this algorithm, whenever the page fault occurs we will replace it with the one which is not used or whose demand will be very late in the future.

⇒ It means :- When a page needs to be swapped in, the OS swaps out the page whose next use will occur farthest in the future.

e.g

f_4	·	·		2	2	2	2	2	2	2	2	2	2		
f_3			1	1	1	1	4	4	4	1	1	1	1		
f_2		0	0	0	0	0	0	0	0	0	0	0	0		
f_1	7	7	7	7	7	3	3	3	3	3	3	3	3	→	→
	*	*	*	*	Hit	*	Hit	Hit	Hit	Hit	Hit	Hit	Hit		

Ref string :- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

↑ → check column in very farthest future being demanded among (2, 1, 0, 7)

∴ Of course 7 so replace '7' by 3 and so on this algorithm replacement works.

Page Hit = 12
Page fault = 8

(58) (#) Least Recently Used Page Replacement Algo.

⇒ In the LRU page replacement algo, the page that is used least recently will be ~~pr~~ replaced.

(whichever was demanded much earlier or first among them)

e.g.:

			2	2	2	2	2	2	1	2	2	2	2
		1	1	1	1	1	4	4	4	4	4	4	1
	0	0	0	0	0	0	0	0	0	0	0	0	0
7	7	7	7	7	3	3	3	3	3	3	3	3	3
*	*	*	*	Hit	*	Hit	*	Hit	Hit	Hit	Hit	Hit	Hit

Now which to replace.

Ref string → 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

(2, 1, 0, 3)
↑
at very first amongst these 4 currently presently in frames. So swap 1 to 4.

among (7, 0, 1, 2) check which was first used amongst them. or which initially appeared. ofcourse it's '7'.
∴ replace 7.

∴ Page fault = 8
Hits = 12

speed → FIFO (more)
Hits → LRU (more)

Note:
LRU does searching. So LRU takes more time than FIFO. But in page fault it performs better than FIFO

(59) # Most Recently Used Page Repl. Algorithm

⇒ This algorithm replaces the most recently used page to minimize the page faults.

⇒ This is because most recently used page will be required after the longest time.

f_4				2	2	2	2	2	2	3	0			
f_3			2	1	1	1	1	1	1	1	1			
f_2		0	0	0	0	3	0	4	4	4	4			
f_1	7	7	7	7	7	7	7	7	7	7	7			
	*	*	*	*	Hit	*	*	*	Hit	*	*			

which to replace from (7,0,1,2)

Ref string - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

the oldest which was just recently used among (7,0,1,2) always it's '0' ∴ replace '0'.

Page fault = 12

Hits = 8