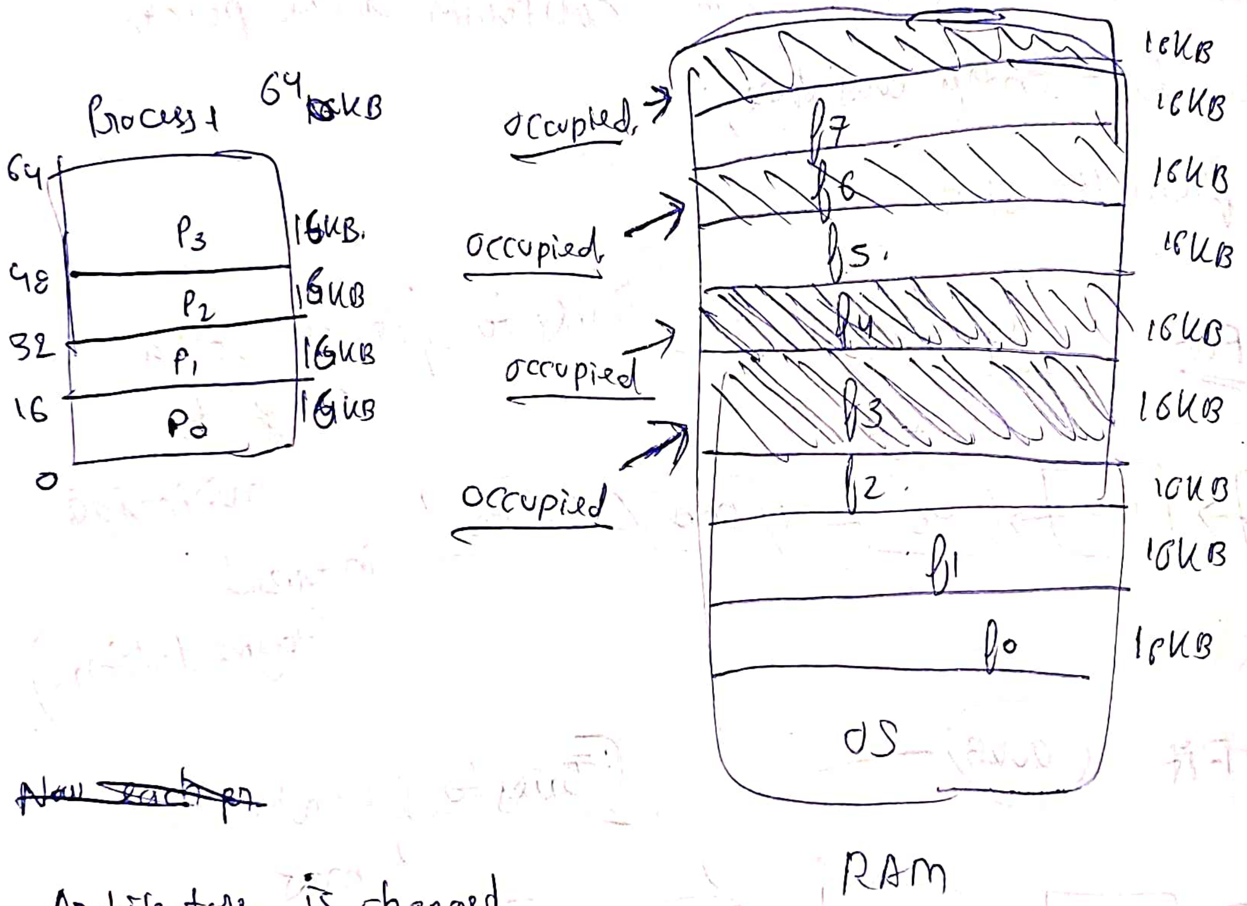
L26: Paging (Non-Contiguous Memory Allocation)

We saw compaction or defragmentation is slowing the performance of process allocation and process running.

So new changes were introduced.



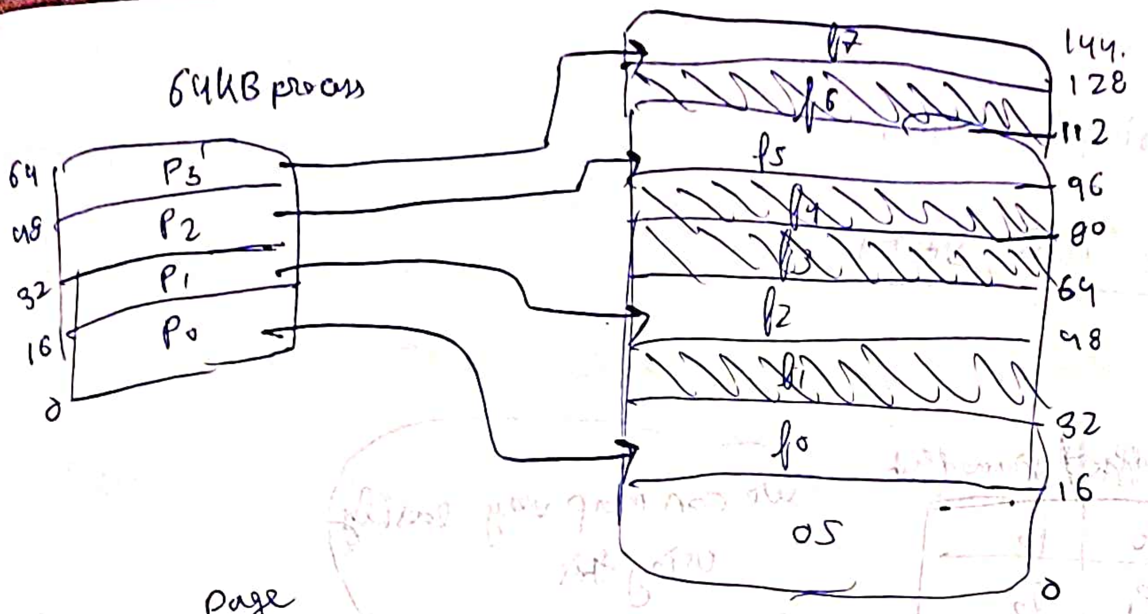
~~New Each pr~~

Architecture is changed,

Process is split in equal segments called pages,
RAM is split into equal segments called frames.

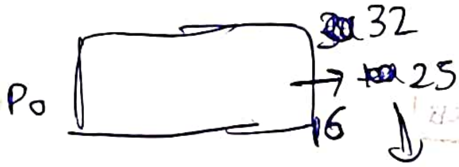
$\boxed{\text{frame Size} == \text{Page Size}}$ We decided on 16KB here

64KB process



Each ~~frame~~ page is mapped ~~to~~ a frame

$P_0 \rightarrow f_0$
$P_1 \rightarrow f_2$
$P_2 \rightarrow f_5$
$P_3 \rightarrow f_7$



lets say

I want to know where is 25 virtual address is located in physical memory

binary

25 → 1|1001

this can be used to identify page Number.

this can be considered offset

page → frame

$\boxed{01} \rightarrow \boxed{16\ (10000)}$ + offset = 9 = 16+9=25

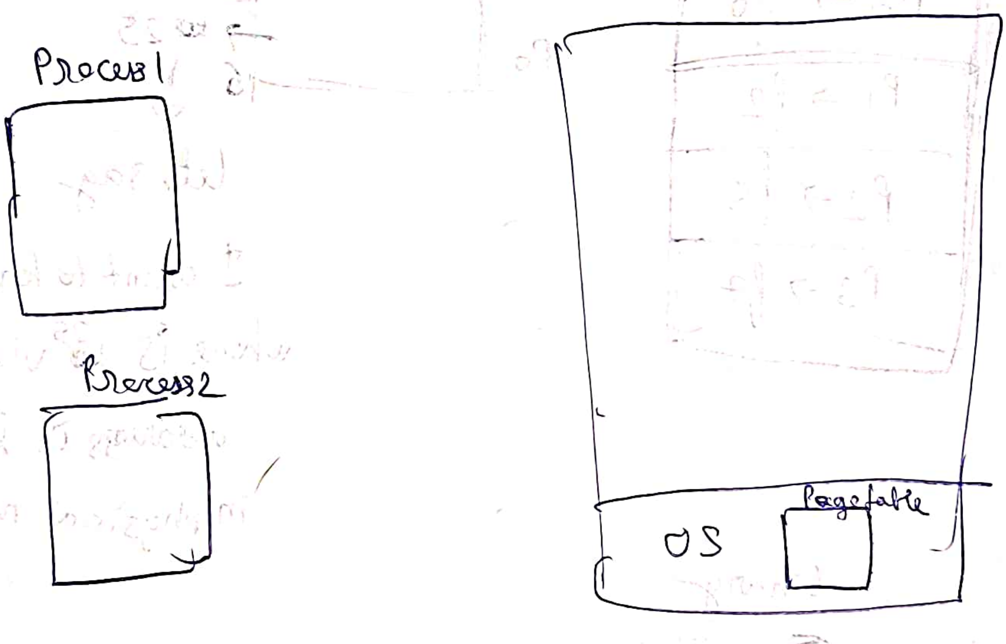
in physical memory

So in simple terms,

Page table stores.

Page offset	frame offset
P_0	f_0
P_1	f_2
...	...

we can map very easily using this



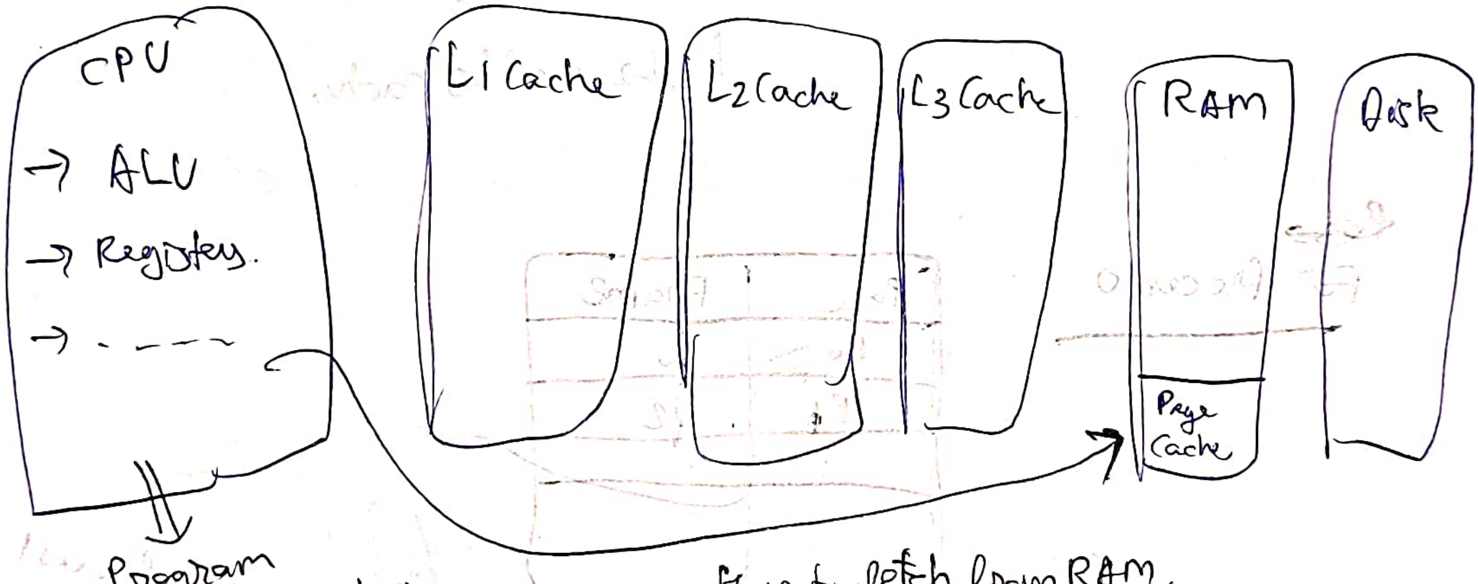
Page table is stored in-memory (RAM).

and hence its a bit slow,

Honestly I was shocked, even considering page table in RAM is slow, So RAM is not upto the standard

Then chat gpt explain, since lots of context switch happens and pages have to be queried from RAM, and sent to CPU for execution from RAM thats a slow operation.

This statement made me curious and I asked for the architecture

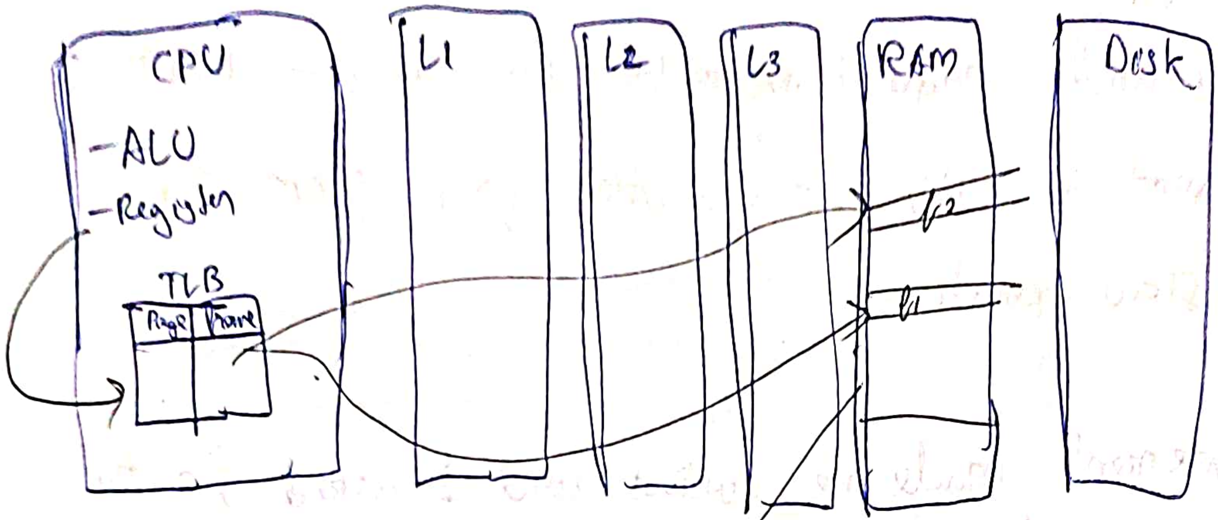


Program execution, operations happen here, So to execute a snippit of code for a process.

Have to fetch from RAM, with L1, L2, L3 cache miss.

So what happens is that for faster lookup a Hardware Cache called TLB.

TLB (Translation LookAhead Buffer Cache) is added



Fetch of frame
the lookup directly
happens here

Most frequently
accessed ~~caches~~ frames
are loaded into
L1, L2 and L3 cache.

For Process 0

Page	Frame
P_0	f_0
P_1	f_2
⋮	⋮

For Process 1

Page	Frame

Now Context
switch Process 1
is to be
executed.

Now P_0 for Process 1
will see if TLB has
P_0 or not,
It will have got that's
the P_0 of Process 0,
So to add a
distinguisher another
column is added in TLB.

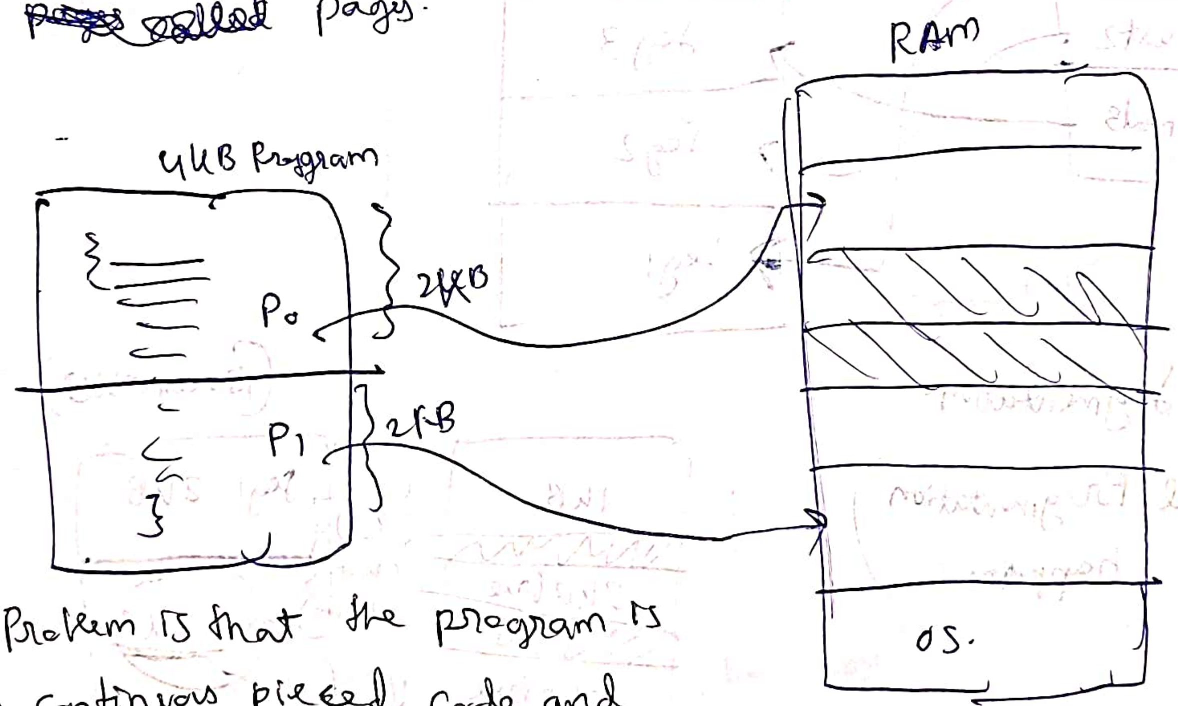
ASCId	Page	Frame
0	P_0	f_0
0	P_1	f_2

→ So Now Process 1 will check if process id == 1
Not equal;

So TLB miss, then it will do a pagetable in RAM Lookup.

Lec27: Segmentation.

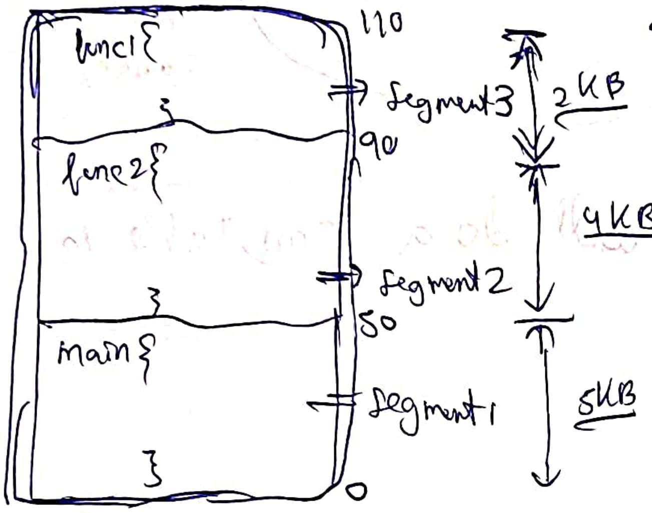
Well paging has a problem. In the last lecture, we saw paging will require spliting process into equal sized ~~pages called~~ pages.



Problem is that the program is a continuous piece of code and while executing when

P_0 gets over then to ~~go~~ to next page P_1 I have to do TLB, worst case PageTable Lookup, slowing down my execution.

to solve this segmentation or concept of segments is introduced.



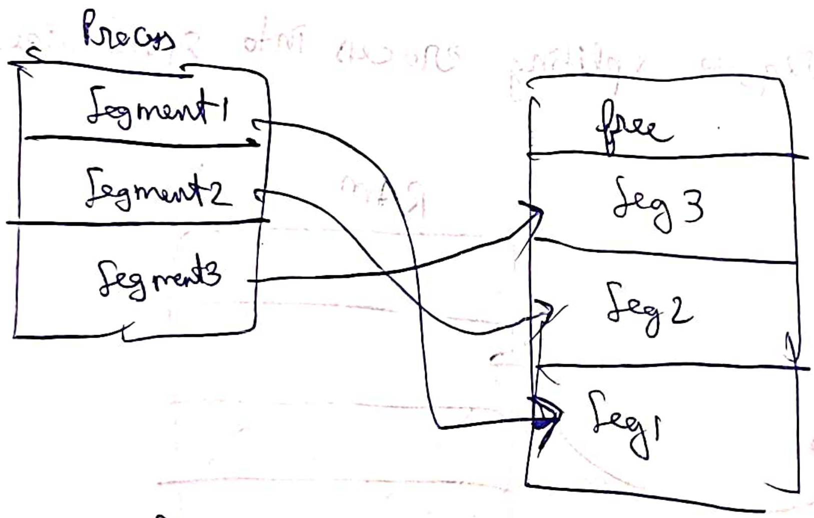
So now instead of pagetable there is a segment Table

Segment Table

limit	Base
50	0
40	50
20	90

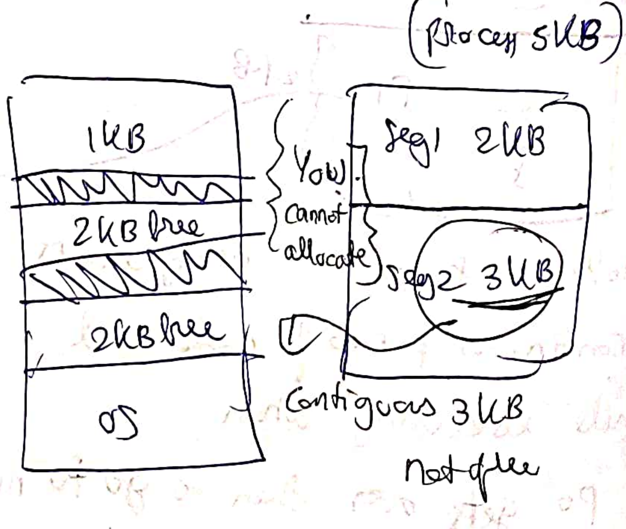
Used for checking isolation, that 1 segment is not accessing out of bounds.

Now ~~pages~~ frames are not there in RAM.



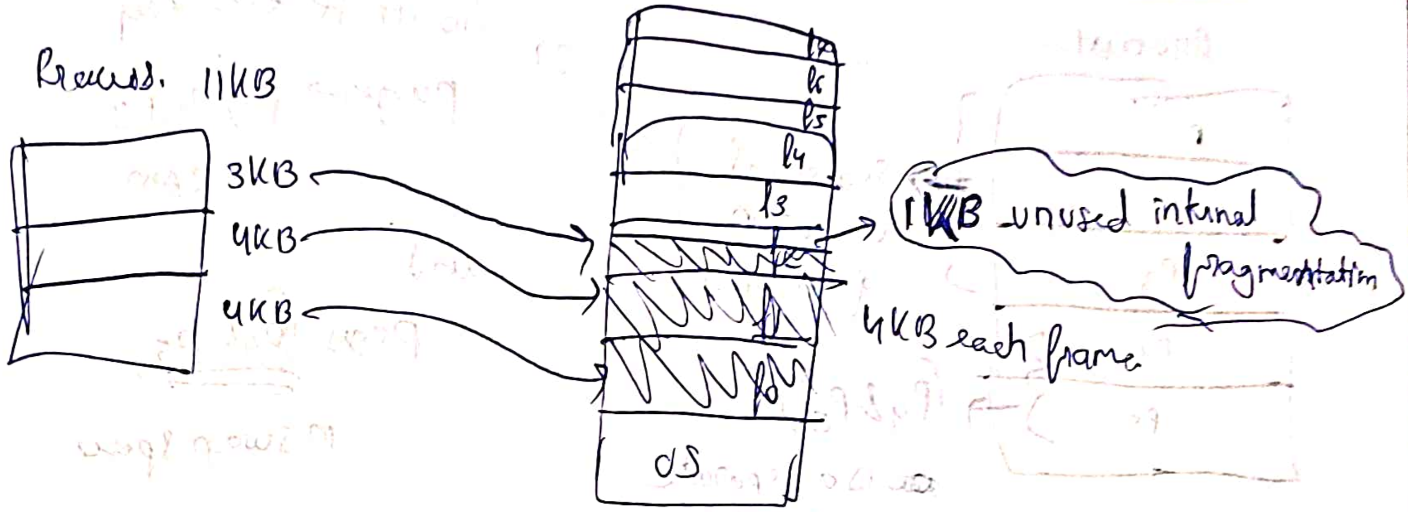
But in Segmentation.

(External Fragmentation happens)



In modern systems mix of Segmentation and Paging is used.

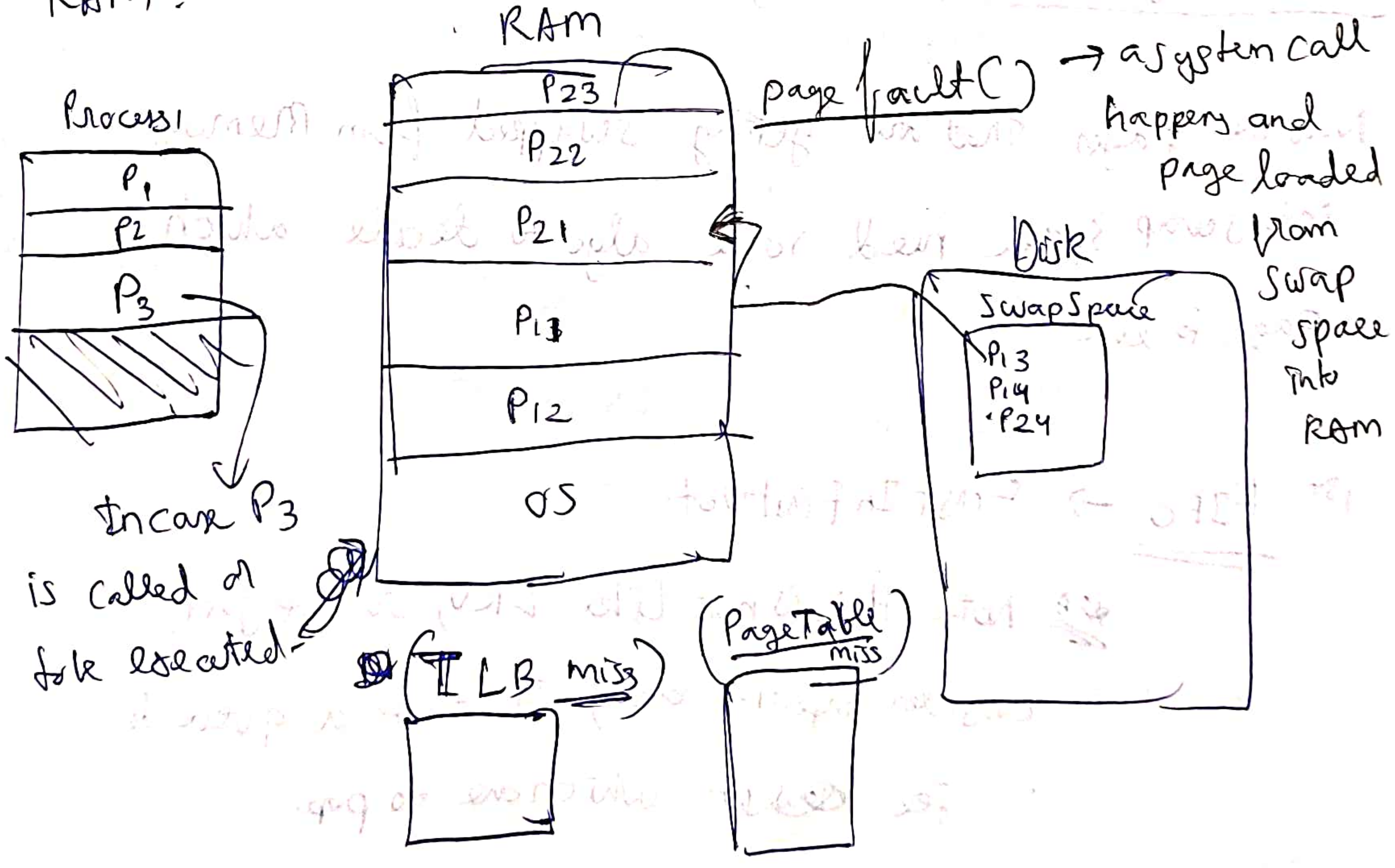
And in paging. ~~system~~ internal fragmentation happens.



Lec 28 Virtual Memory

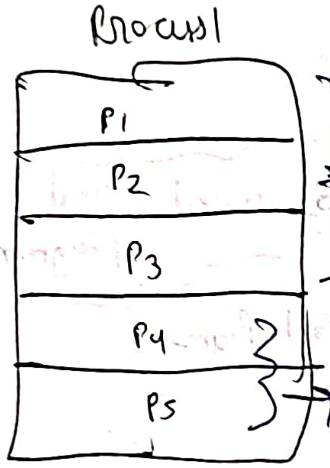
This we studied recently even in Kafka

Basically processes bigger than RAM are able to run on RAM.



One way to minimize page faults, is by locality references.

meaning



Sequential execution of a method

(P4 & P5) is a separate method call

So its better to keep pages → P1, P2, P3 in RAM

and pages (P4 & P5) in swap space

Pages which are in nearby locality of executable section, keep those in RAM.

Lec 29: Page Replacement Algos.

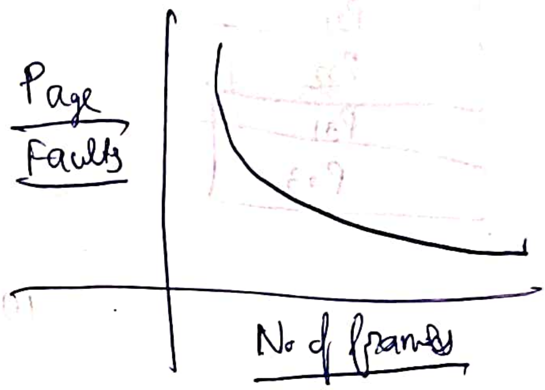
Now the pages that are getting swapped from memory into swap space need some algo to decide which page to evict

1st **FIFO** → First In First Out

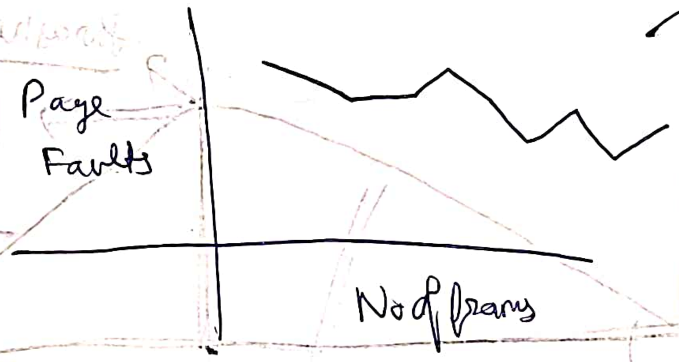
* Note this is not like LRU, so we just consider insertion order, Just use a queue to see which one to pop.

In FIFO one would assume, if we increase the number of frames in RAM, then page fault would decrease,

(Expected graph way)



(Reality graph)



→ Unexpect Behaviour.
(Belady's Anomaly)

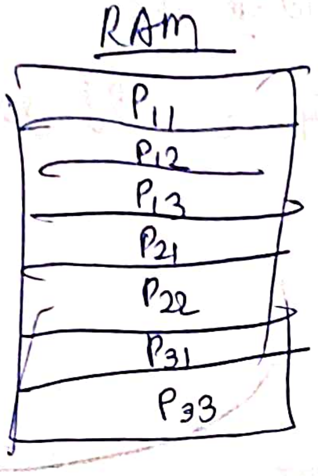
Rest we know about

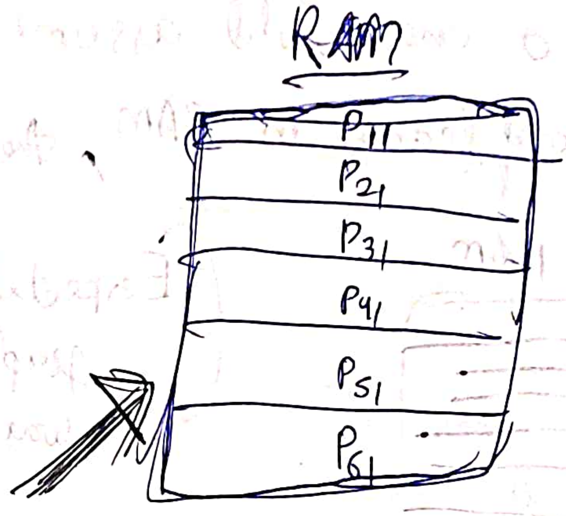
[LRU, LFU, MFU] → (pop most frequently used)

Lec 30: Thrashing → Scenario where page fault and page swapping has increased so much that CPU is not getting time to execute the program.

The more the multiprogramming, the more chances of thrashing and swapping

(P.T.O)

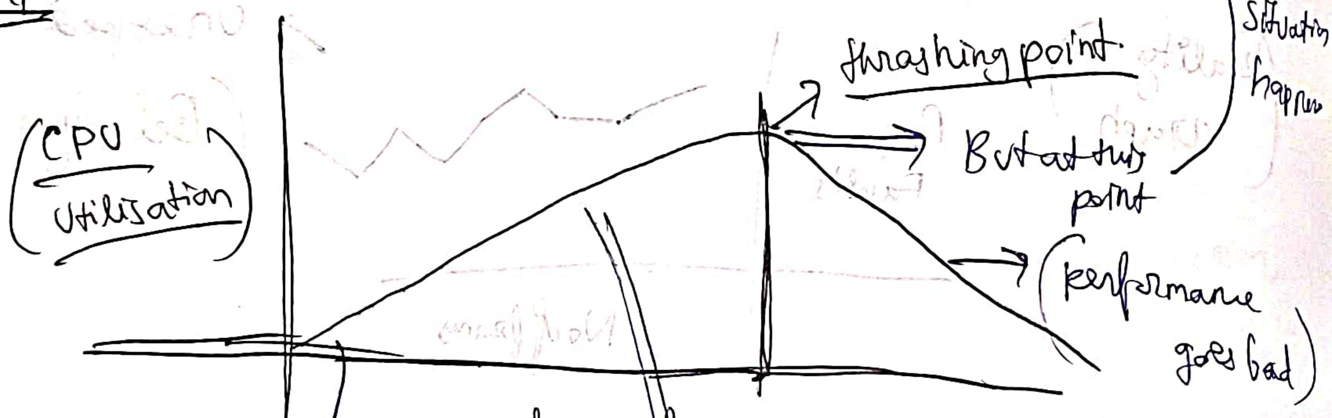




more multi-programing more thrashing

this situation happens

Graph



degree of multiprogram

Since degree of multi-programming is less so if programs are doing I/O then CPU is Idle

As we increase programs CPU utilised more