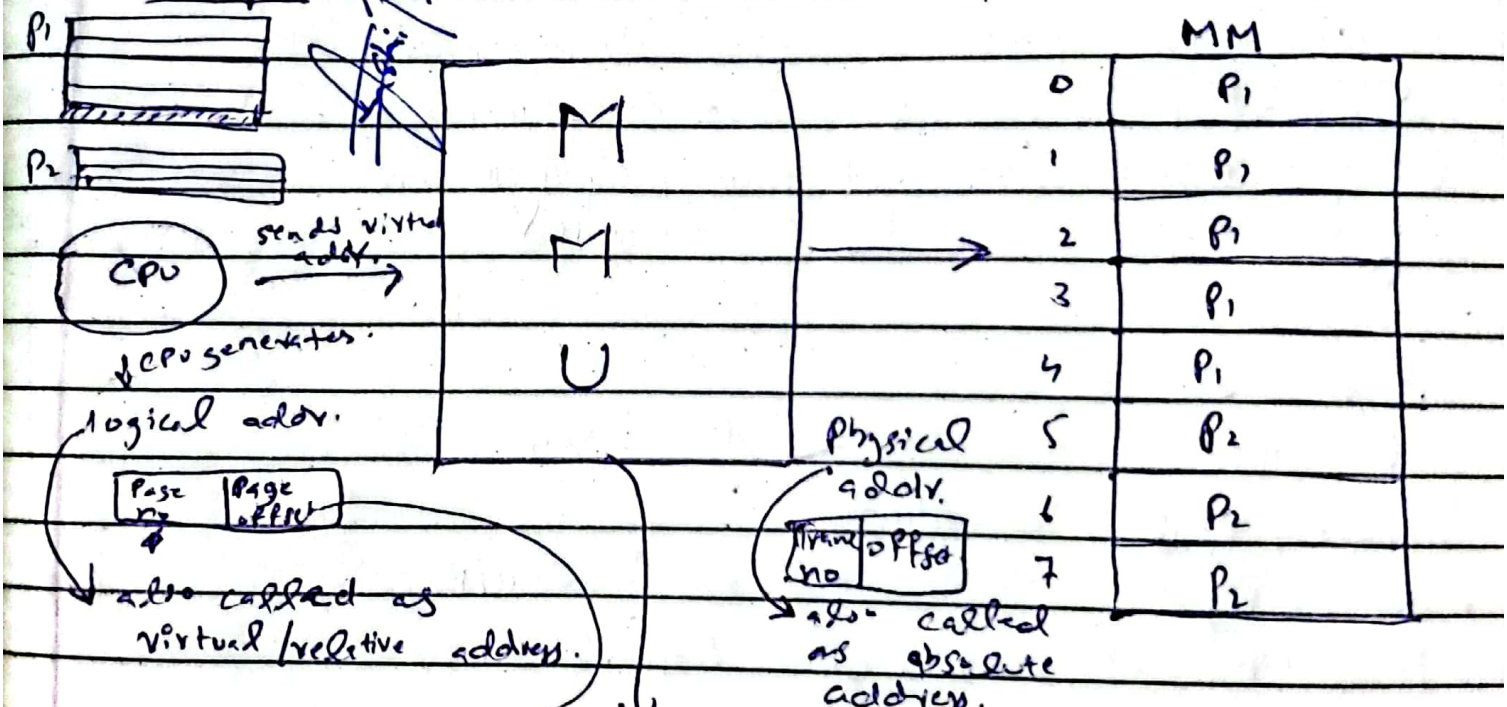
Tashfeen Abbasi NUCES, FAST, PAKISTAN.

Memory Managment

- Requirements for Memory Management Unit (MMU)
  1. Relocation
  2. Protection
  3. Sharing
  4. logical organization
  5. physical organization

- Programs are stored on disk & need to be loaded into the memory to run.
- Address binding of a program:
  - Compile time → Program is fixed to specific memory location
    ↓ static & can't move.
  - load time → (Program loaded → memory location determined)
  - Run time → Program moves to different memory location
    ↓ dynamic. during execution managed by MMU.

Example:-



•— Swapping

Swap out → memory full → process temporarily moved to lower storage (disk)

Swap in → When needed again brought back to memory.

•— Relocation

- Swapping (Program moves in & out of memory, location may change)
- Uncertainty (Program does not know where it will be at any time in memory)
- Addressing (relative addresses are used which are adjusted during execution as program moves and changes position)

•— Protection

- Ensures one program does not interfere with another's memory

① Base register (marks starting address where a process can access memory)
(smallest physical addr)

② Limit register (how much memory process can use beyond the base)
(specifies size of range)

- Base register adds a logical address to give physical address.
- Limit register checks if logical address is within allowed range. If out of range MMU raises an interrupt to prevent access.

① no of partitions same
② Size of partitions can be different

Internal Fragmentation
↓ solution
External
↓ solution
Compactation.

• Fixed Partitioning / Static Partition

- Memory is divided into fixed-size blocks. Any program regardless of its size, occupies an entire block. This causes a problem known as internal fragmentation.

Internal Fragmentation → (fixed size)
- If program doesn't use full block, unused memory is wasted.
- Requested memory can be smaller than allocated memory.

- Small process have to wait, even though plenty of memory is free. If large partition is empty and queues for small partition is full but everyone has its own queue and can not access other memory. We use single queue for all partition

Multiple queue

- Whenever, partition becomes free, a process is selected
  ① Closest to front of queue
  ② Smaller than partition size.

Single queue

Maybe space wasted as above mention

• Dynamic Partitioning (we allocate process according to size of our process so no internal fragmentation)

External Fragmentation → Occurs when variable size memory space are allocated. (happens when sufficient memory but can not be fulfilled as it is non-contiguous memory)

(Variable size)

Internal

4MB) 0	//// 2MB ////	→ unused.
8MB) 1		
2		
3		

External

40	P1	40	////	Free
60	P2	60	P2	
10	P3	10	////	Free.
20	P4	20	P4	

Now, P5 process aya of 50 size, but can not be ad
but we have 50MB spa

Separating allocating and unallocating but reduces
↑ CPU efficiency or time.

Compaction → Processes are shifted to remove gaps.
(defragmentation) creating larger continuous free space.
↓ (Example of vacuum pump as it sucks)
relation to external.

- Memory Management with Bitmap.

(1) • Memory divided into allocation units.
1 → occupied / process.
0 → free / hole.

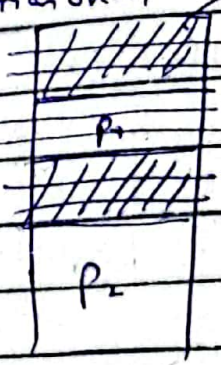
Smaller allocation unit → larger bitmap
larger allocation unit → smaller bitmap.

Example :-

allocation unit

H	H	H	P_1	P_1	P_1	P_1	H	H	H	P_2	P_2	R

0	0	0	1	1	1	1	0	0	0	1	1	1



Unallocated memory. now called as hole or free.

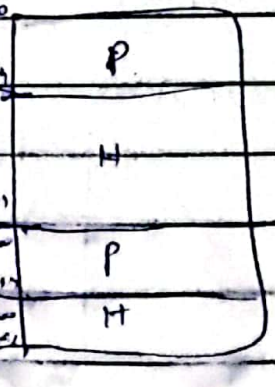
(2) With Link list.

Example:-

P/H	Start	End.

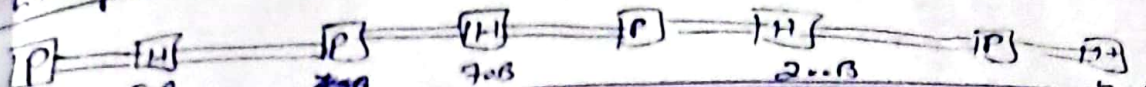
P	100	199	→	H	200	299	→	P	300	325

H	400	401



but it can be merged if there are 2 holes or 3 or more.
but single linked list will see the holes from its agg not previous
we use double linked to see both ways

Example



- Dynamic Partitioning : Placement Algo.

(1) First-fit (first block/hole j is main fit as jae)
(2) Next fit (start from last allocated block, use next block that is big enough)
no better performance than First-fit.
(3) Best fit (scans full linked list, time more, performance le
Consider all holes, and take shortest possible hole)
• Fill up memory with tiny useless holes.
• First fit generates larger holes on average.
(4) Worst fit. (Best fit reverse).

• Search time of all four placement algo can be improved by
① Keep separate lists for process & holes.
② while allocating search only holes.
③ hole list can be sorted by size

- Paging (non-contigous) (Avoid external fragmentation)
• Divide process into same size chunk (pages) Secondary memo
• Divide memory into equal-size chunks (frames) Size should be same
• OS maintains a page table for each process.
• Pages of a process forms a logical memory.

A1		A1	Now D is added but has D1 D2 D3	A1
A2		A2		A2
B1				D1
B2				D2
C1		C1		C1
C2		C2		C2
				D3

leftmost ← no. of Pages 2^{m-n}.

logical address space 2^m

rightmost ← page size / 2^n.

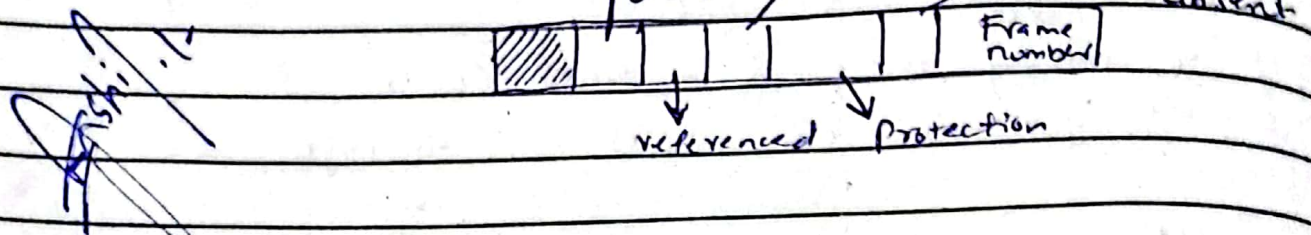
offset

Q: Do we avoid fragmentation completely in Paging?

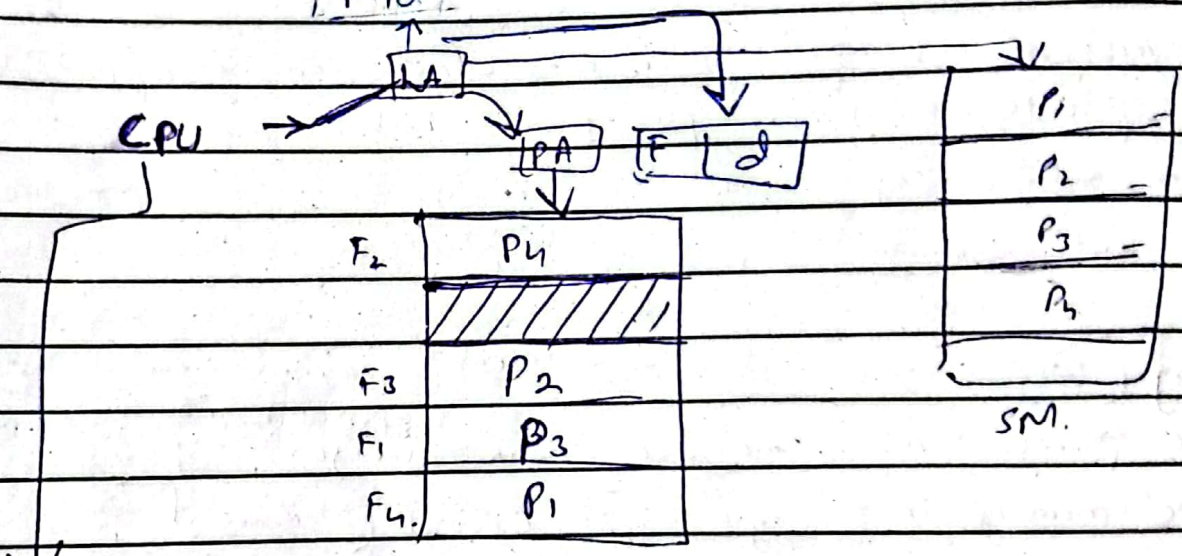
No. While Paging eliminates external fragmentation, it introduces internal fragmentation.

last page of process might not fully use a frame.

- Page Table Entry.



- Address translation in Paging



P1	F4
P2	F3
P3	F1
P4	F2

Page Table.

(kn se page kn key frame pr hy)

- Every Process has individual Page table
- Page table has Frame no
- If process has 6 entries then ~~Page table has~~ also have 6 entries

Example

m = 16, n = 12.

- logical address = 16 =
- Page number = 16 - 12 = 4 = 2^n = 16
- Page size = 12 = 2^n = 4096.
-

~~Example~~ 8196

- Binary = 0010000000001100

Page no = (leftmost 4 bits) = $0\times2^3 + 0\times2^2 + 1\times2^1 + 0\times2^0$

offset = (leftmost 12 bits) = $0\times2^{\cdots} \cdots 1\times2^2 + 0\times2^1 + 0\times2^0$

= 4

m = 16, n = 10.

logical address = 16

page size = 10 = ~~2^{10}~~ =

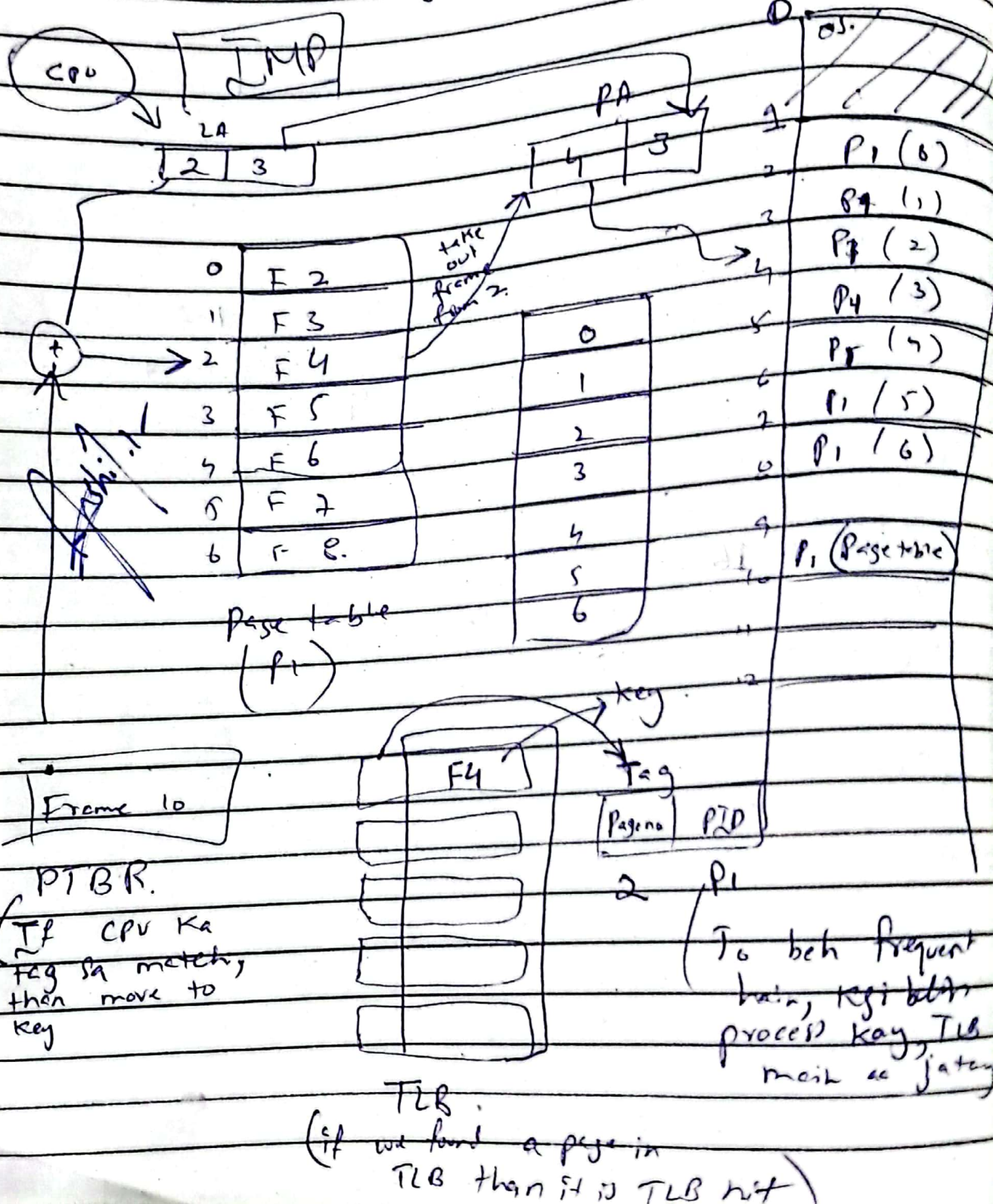
page number = 16 - 10 = 6 = ~~2^6~~ =

000001 011101110

page no. page size.

•-IMPLEMENTATION OF PAGE TABLE

- Page table is kept in main memory
  - Page table base register (PTBR) - point to page table
  - Page table length register (PTLR) - indicates size of page table.



•— (TLB) Translation lookaside buffer (TLB)

- TLB miss (when miss than all Page process)
- TLB hit (when hit, no need to access page table)
- Parallel search (code for page table)
- resides usually in MMU.

- Associative Lookup (ε) → time to check TLB. Very small
  Hit ratio (α) → %age of times required page is found in TLB.

Effective Access Time (EAT)

$$EAT = (1+\varepsilon)\cdot\alpha + (2+\varepsilon)\cdot(1-\alpha)$$

•— Single-Level Page Table limitations. (memory issues)

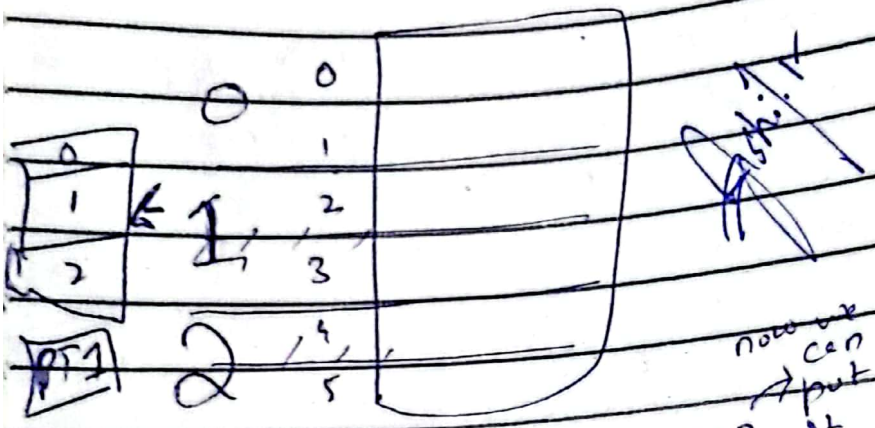
•— Two-level Page Table → • Break large Page table into smaller subtables (outer & inner)

• outer Page table points to smaller page tables that stores actual frame numbers

• logical memory has | outer | inner | offset |

•- Multilevel Paging (12 Kb ka Page table hy, 4 Kb kay ~~[illegible]~~ 1 frame main kajay ~~[illegible]~~ gay)

• We divide Page table in multiple levels.



$PT = \frac{12KB}{3} = 4KB$ now we can put it in memory.

•- Inverted Page table. (contains all processes ~~[illegible]~~ global Page table)

Inverted PT.

fr	Page No	PID
	P_0	P_1
	P_1	P_2
	P_2	P_1
	P_1	P_3
	P_3	P_2
	P_2	P_3
	:	:
	:	:

0	f_0
1	X
2	f_2
3	X

Page table 1

0	X
1	f_3
2	f_5
3	X

PT 2

0	X
1	f_1
2	X
3	f_4

PT 3

Main Memory = cache of disk space.

- Decreases memory needed to store each page table
- Increase in time for search.

Virtual Memory

- OS brings few pieces of program into main memory
- resident Set → Portion of process that is in main memory.

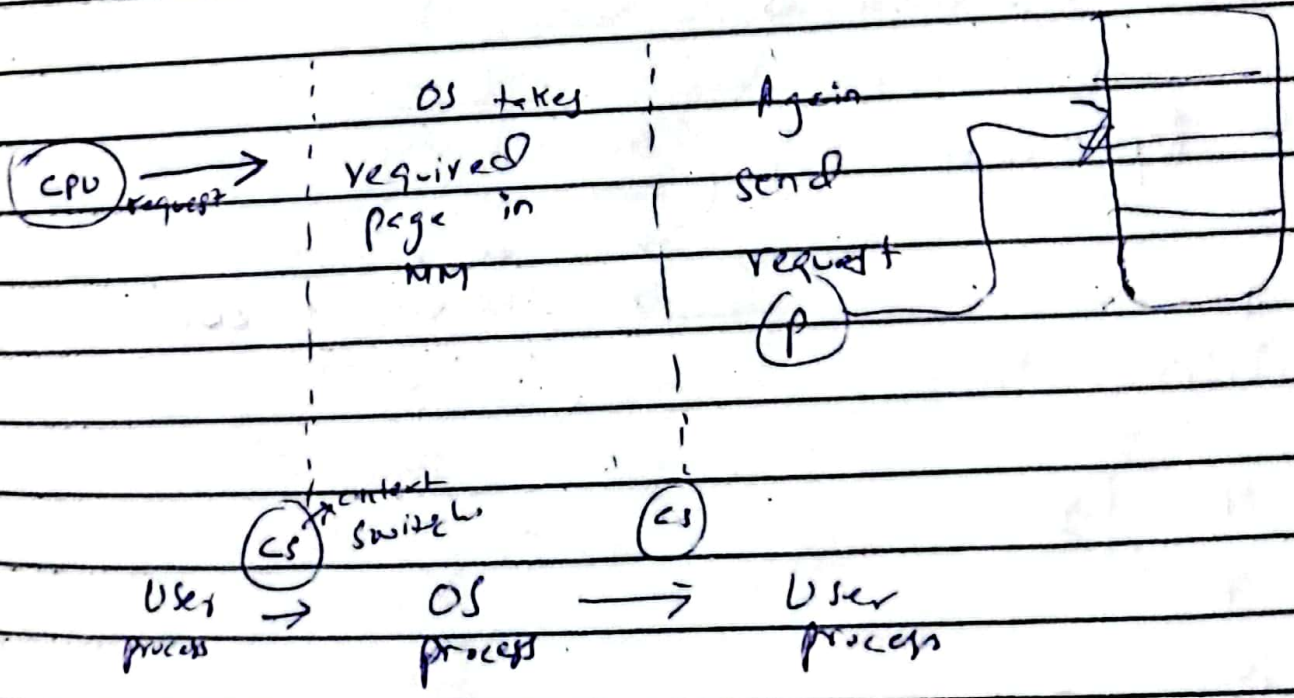
1 → in memory
0 → not in memory

- Page Fault & Address Translation

Demand Paging & Page Fault.

To fetch 1 page in requirement no eye. Do not load it in main memory

if CPU say that I need this page but it is not in entire MM.



Belady's Anomaly → More frames, sometimes more page faults

Page Replacement Algo

1) • FIFO

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

F = 3

4	1
7	2
6	7

4 4 6 1 2 7

pf = 6

• 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

• 4 frames

1	8 4
2	1 5
3	2
4	3

1 2 3 4 1 2 5 1 2 3 4 5

Total Page faults = 10.

2) Optimal Replacement Algo (See after the memory fuels, insert 5th then see in MM that the elements which immediately after that node, do not replace them)

↓

Future

F = 3

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

4	1
7	
6	2

Page fault = 5.

(Used as a benchmark because it provides best performance)

OPT
LRU
Clock
FIFO.

3) LRU (See previous) (Use Stack)

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

F = 3

~~4~~	1
~~7~~	2
~~6~~	7

Pf = 6.

Problems:

- OS kernel involvement at every memory reference.

- Uses reference bit
- initially 0
- 1, when page is referred

All bits set → FIFO