1 bit

4 bit ⇒ nibble

8 bit ⇒ 1 byte

16 bit ⇒ 2 byte ⇒ 1 word

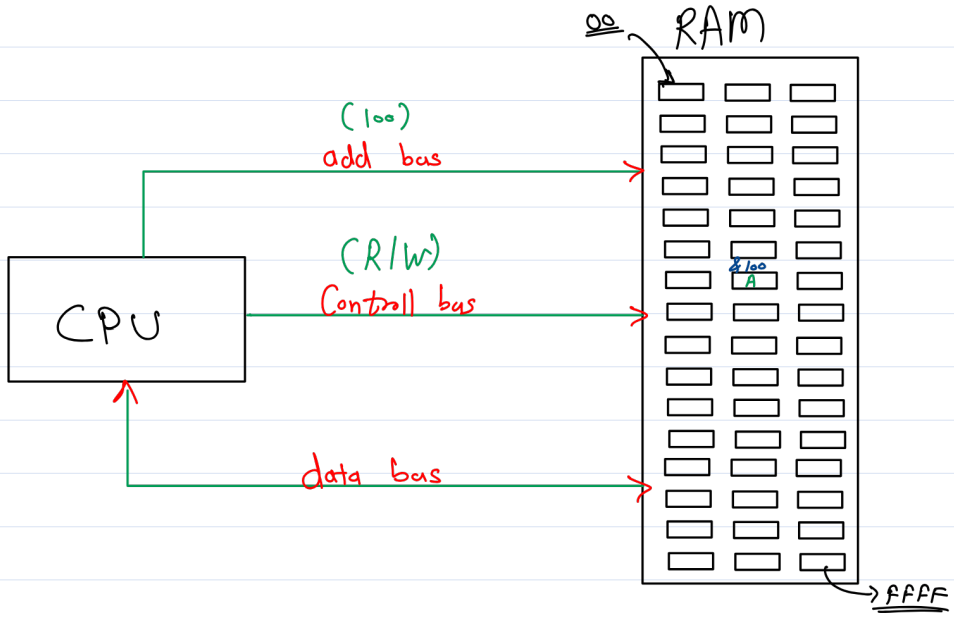
1 byte * 1024 ⇒ 1024 byte ⇒ 1 kb
(2^{10})

1 kb * 1024 ⇒ 1024 kb ⇒ 1 mb [1 byte * (2^{20})]
(2^{20})

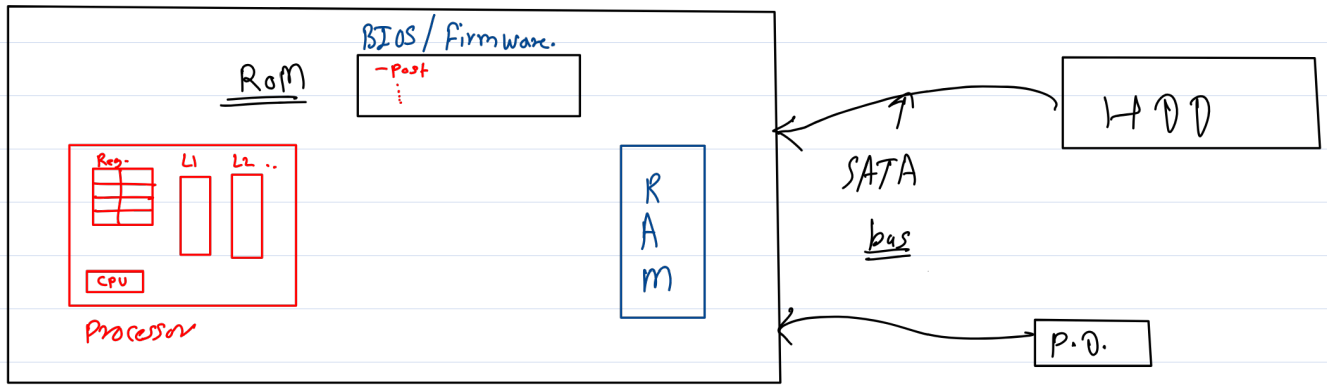
1 mb * 1024 ⇒ 1024 mb ⇒ 1 GB
(2^{30})

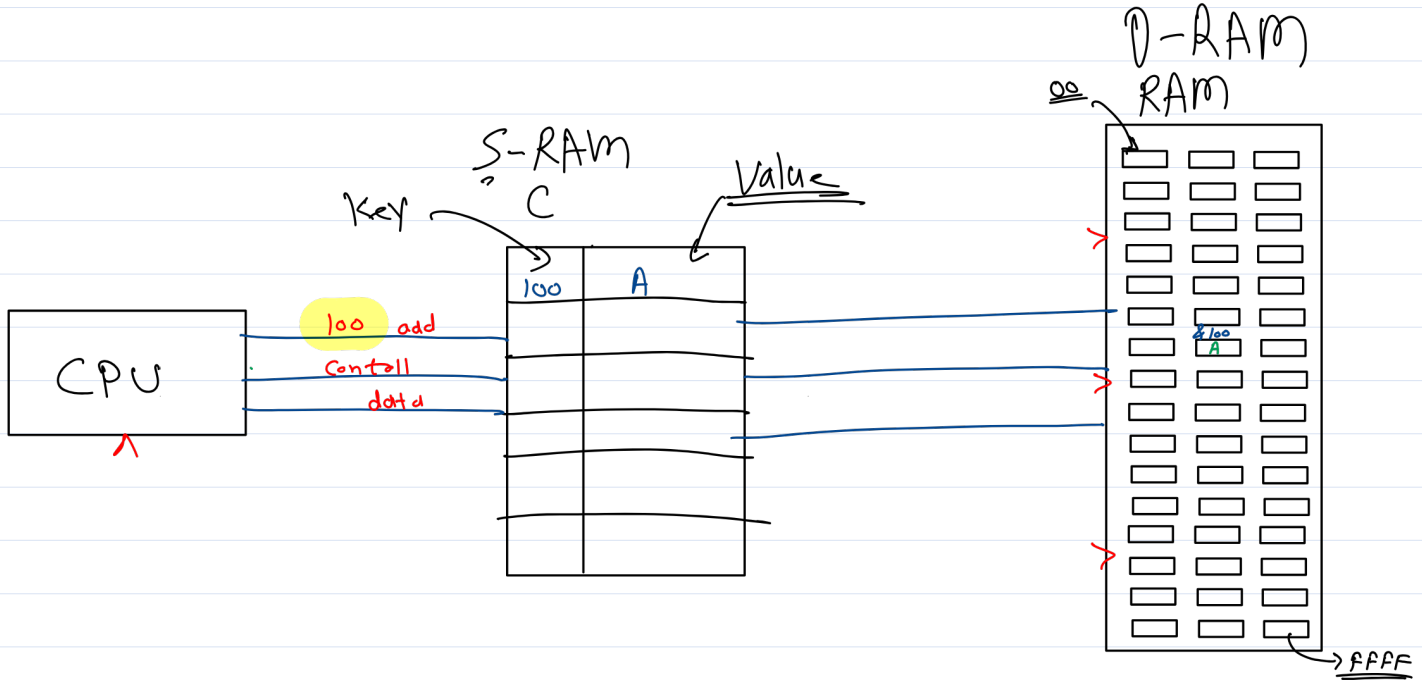
1 Gb * 1024 ⇒ 1024 Gb ⇒ 1 Tb
(2^{40})

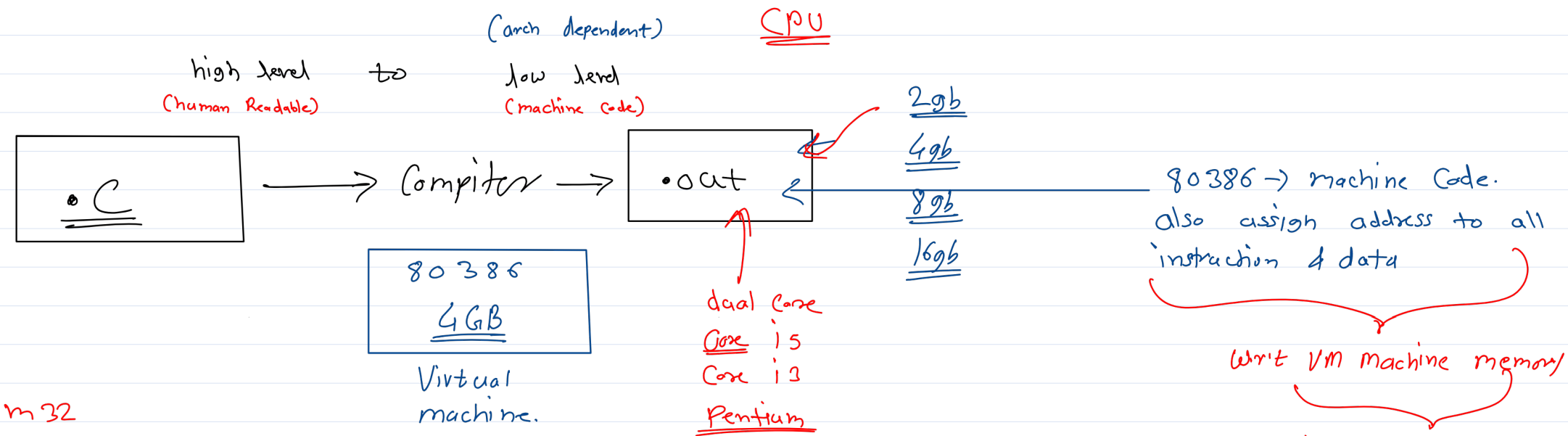
1 Tb * 1024 ⇒ 1024 Tb ⇒ 1 Pb
(2^{50})



Mother board



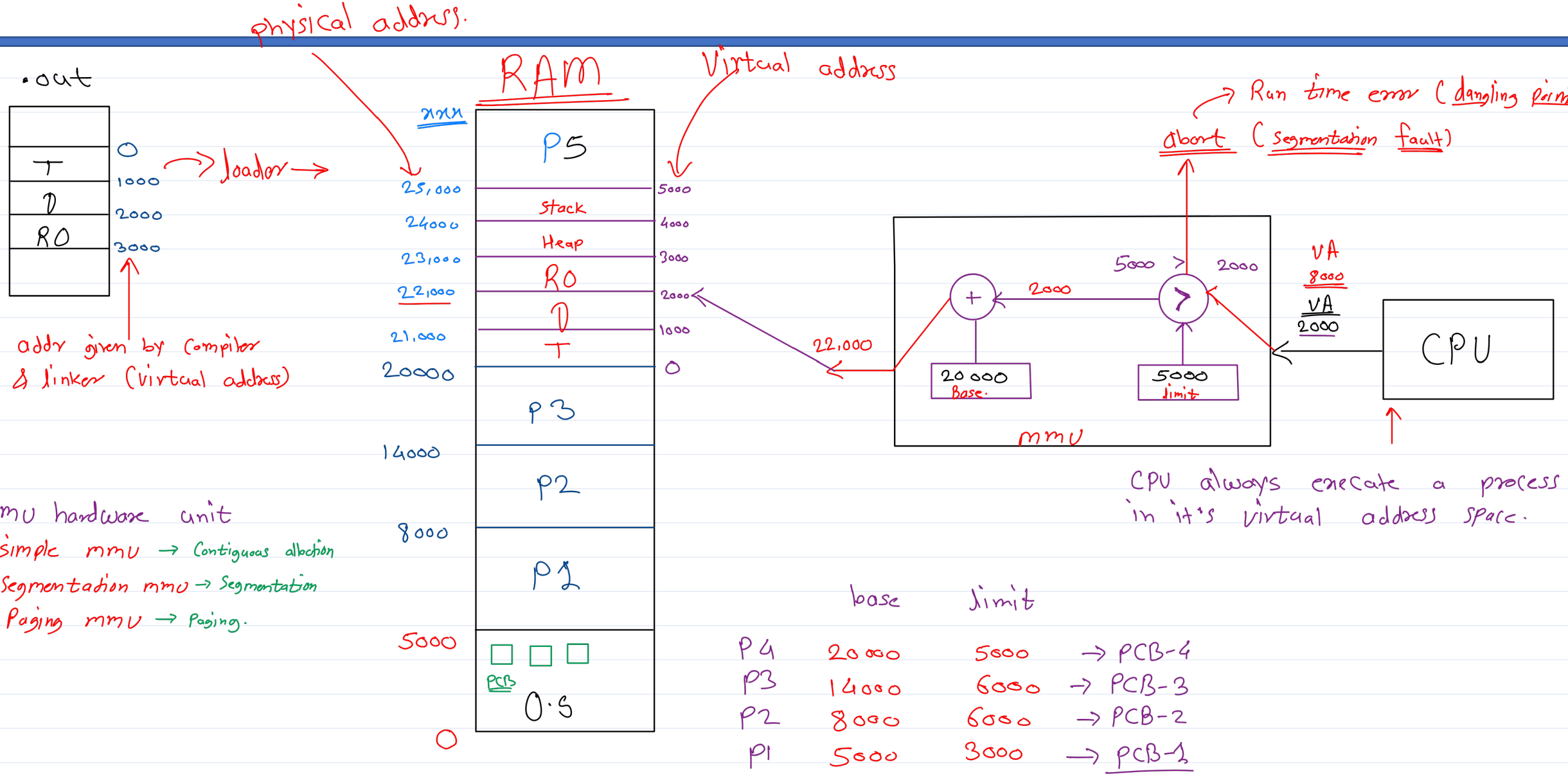

D-RAM
RAM
00
S-RAM
C
Key
Value
100
A
CPU
100
add
Control
data
&100
A
FFFF



gcc -m32

pointer → 8 byte

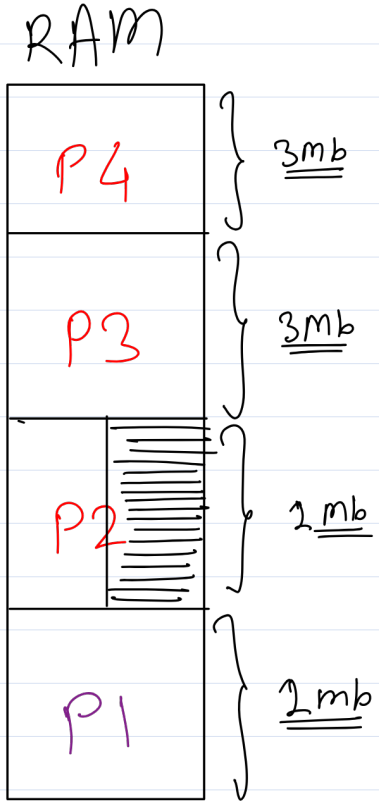
pointer → 4byte


physical address.
RAM
Virtual address
.out
T
D
RO
0
1000
2000
3000
loader
addr given by compiler & linker (virtual address)
xxxx
25,000
24000
23,000
22,000
21,000
20000
14000
8000
5000
0
P5
Stack
Heap
RO
D
T
P3
P2
P1
PCB
O.S
5000
4000
3000
2000
1000
0
22,000
+
>
2000
5000
2000
20 000
Base.
5000
limit
mmu
VA
8000
VA
2000
CPU
Run time error (dangling pointer)
abort (segmentation fault)
CPU always execute a process in it's virtual address space.
mmu hardware unit
→ simple mmu → Contiguous allocation
→ Segmentation mmu → Segmentation
→ Paging mmu → Paging.
base
limit
P4 20000 5000 → PCB-4
P3 14000 6000 → PCB-3
P2 8000 6000 → PCB-2
P1 5000 3000 → PCB-1

(1) Fixed size Partition.

P1 → 1mb
P2 → 500kb
P3 → 3mb
P4 → 3mb

P5 → 4mb ✗



✓ simple to implement.

✗ process not utilizing all memory assigned to it — internal fragmentation.

✗ max num of processeses is fixed
i.e. max num of partitions.
(Degree of multi-prog is fixed)

✗ max size of process
= max size of partition.

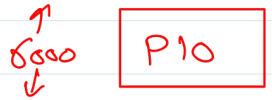
(2) Variable size Partition scheme.

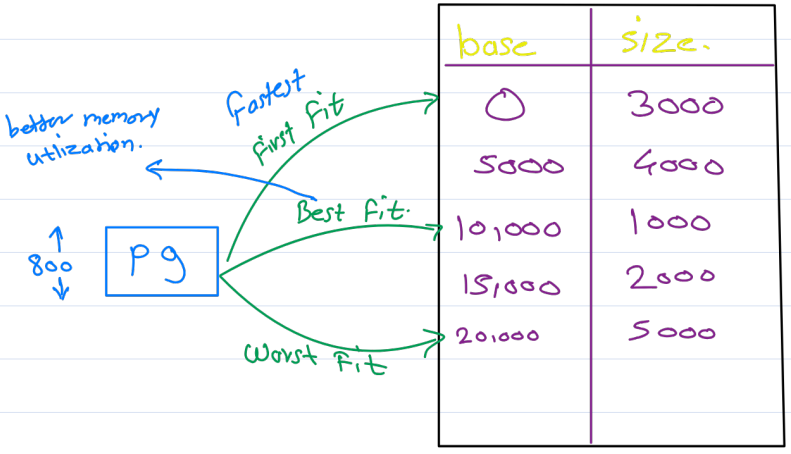
✓ No internal frag
✓ Degree of M.P. is not fixed.
✓ Process size is not fixed.

✗ External frag.

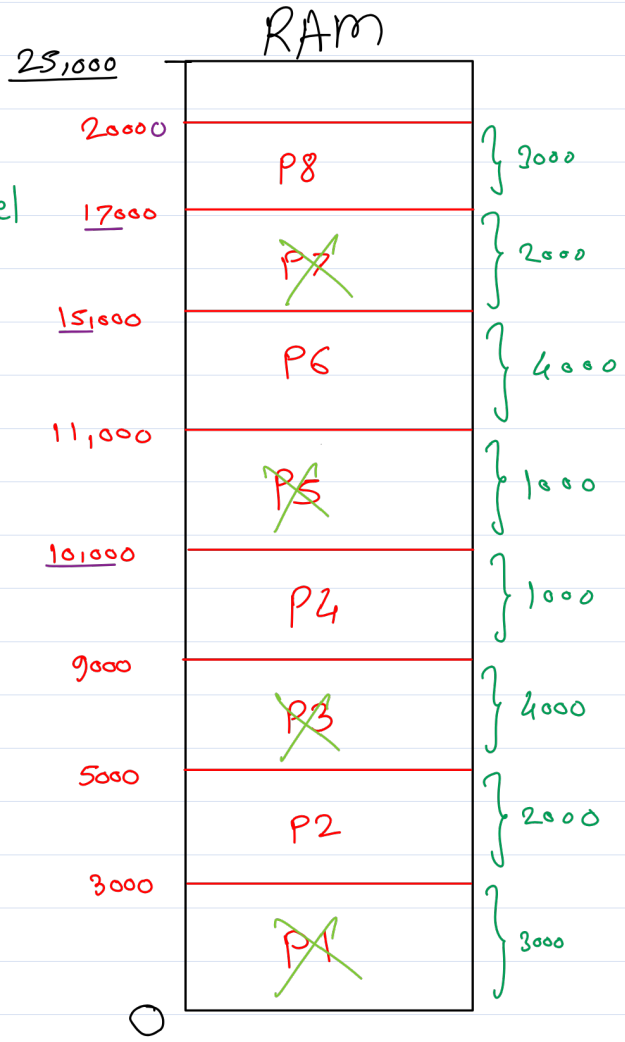
Soln Compaction:

✗ External frag : memory Req. is available but not contigous order.

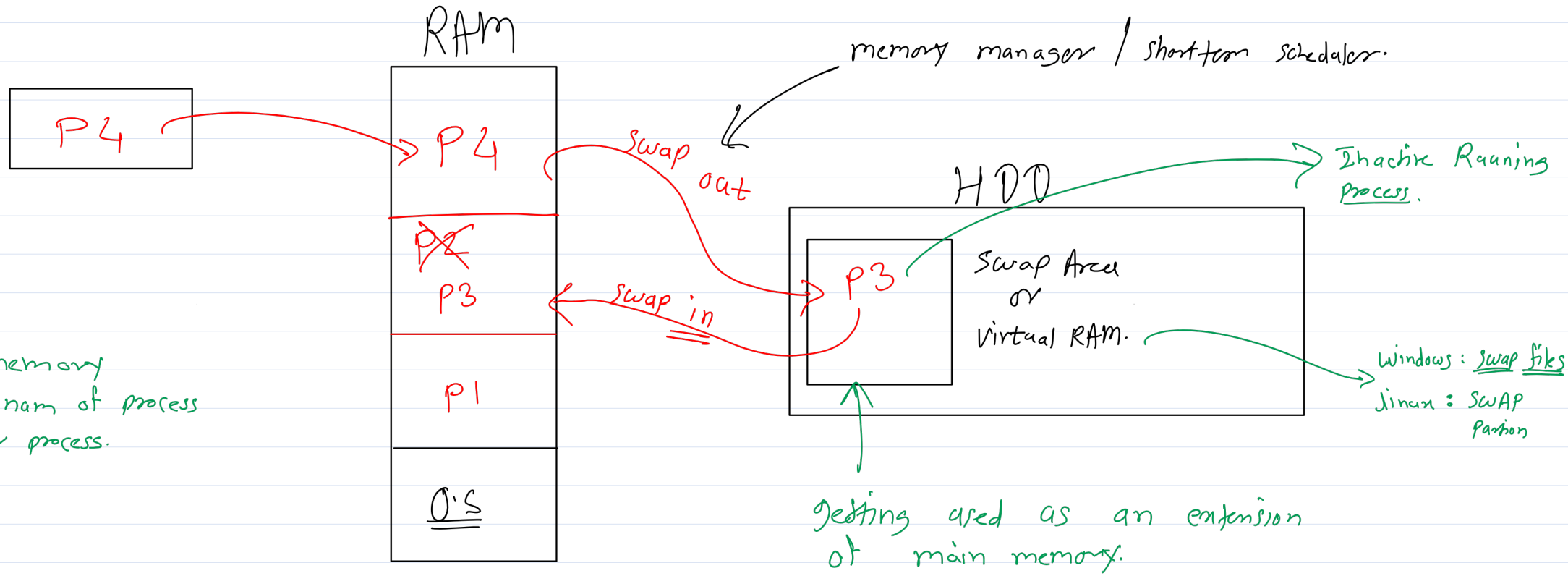




Free slot table.

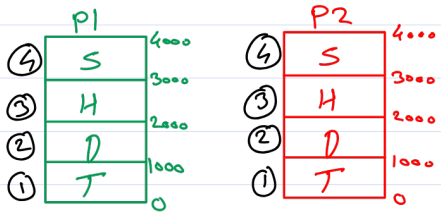


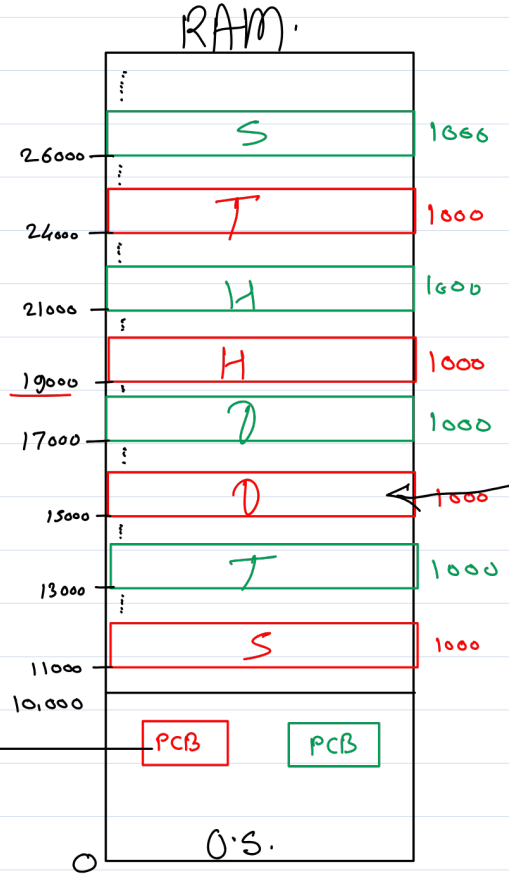
* Virtual memory / Swap Area.



* Using Virtual memory
  - Create max num of process
  - Create bigger process.

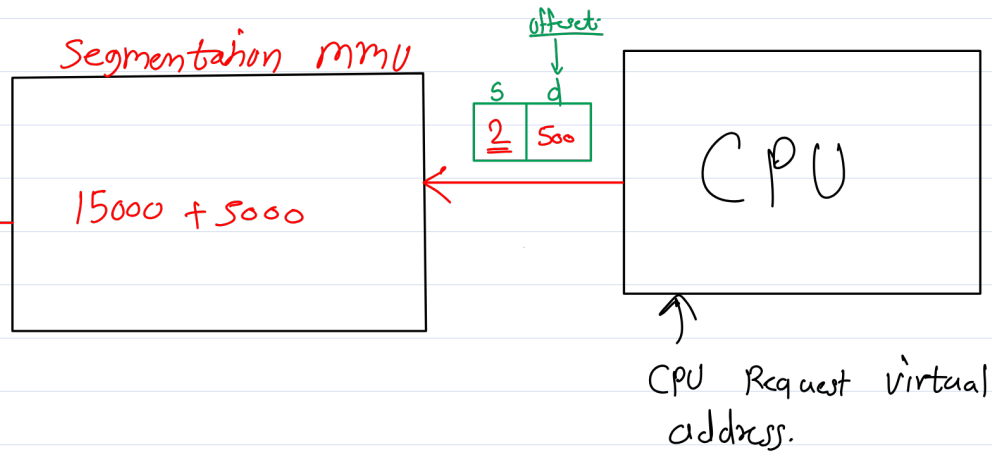
(1) Segmentation:





Segmentation table.

	base.	limit
1	24000	1000
2	15000	1000
3	19000	1000
4	11,1000	1000

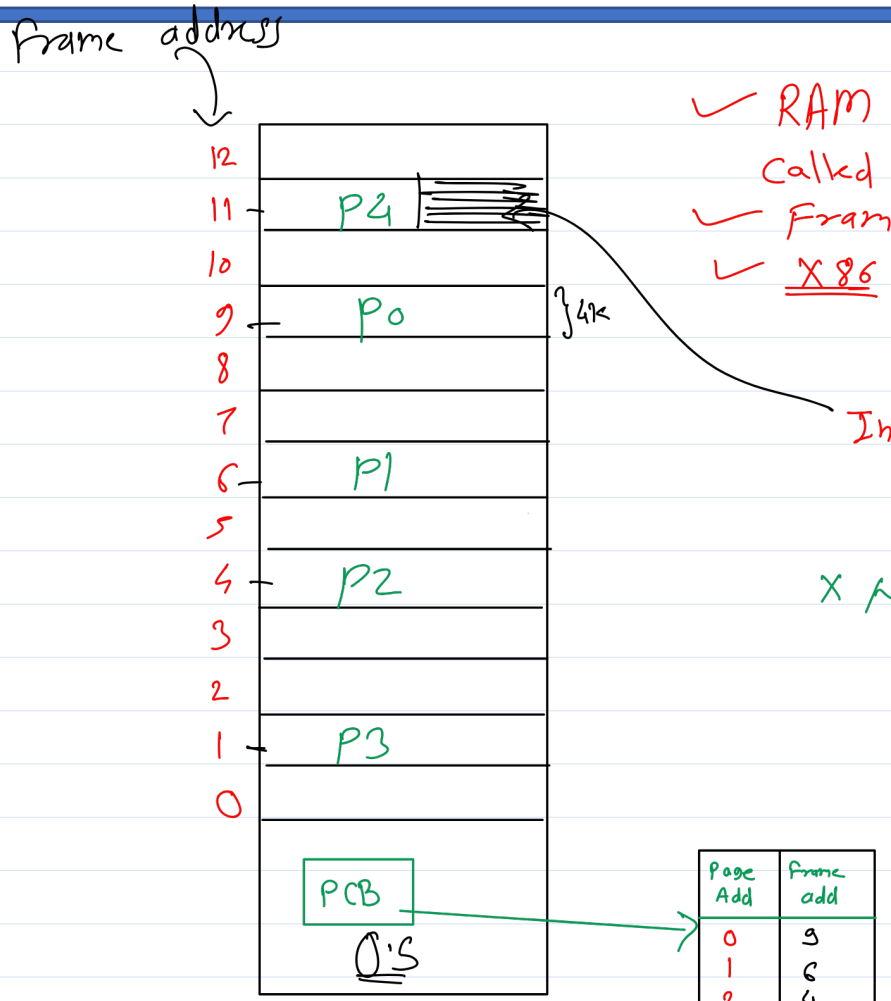


(2) Paging

Proces (18 kb)

P0	4K
P1	4K
P2	4K
P3	4K
P4	2kb

Process is divided into small parts (same as frame size), called as **Pages**.



- RAM divided into small equal size parts - called as **frame**.
- Frame size depends on mmu.
- X86 arch. - frame size ⇒ 4kb

Internal frag. happens: if a page size is less than frame size.

X No external frag..

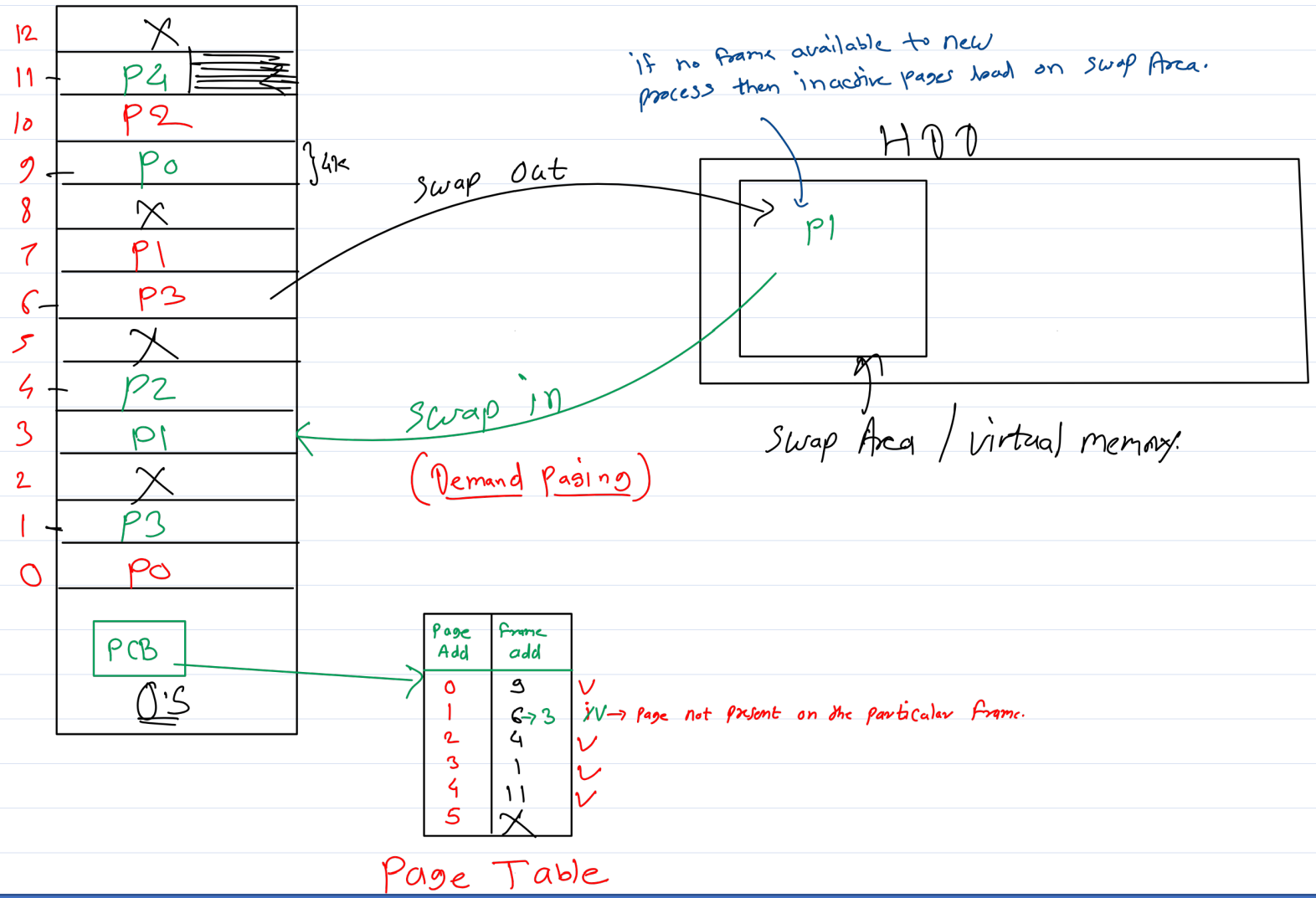
Page Add	Frame add
0	9
1	6
2	4
3	1
4	11
5	X

Page Table

Virtual RAM + Paging

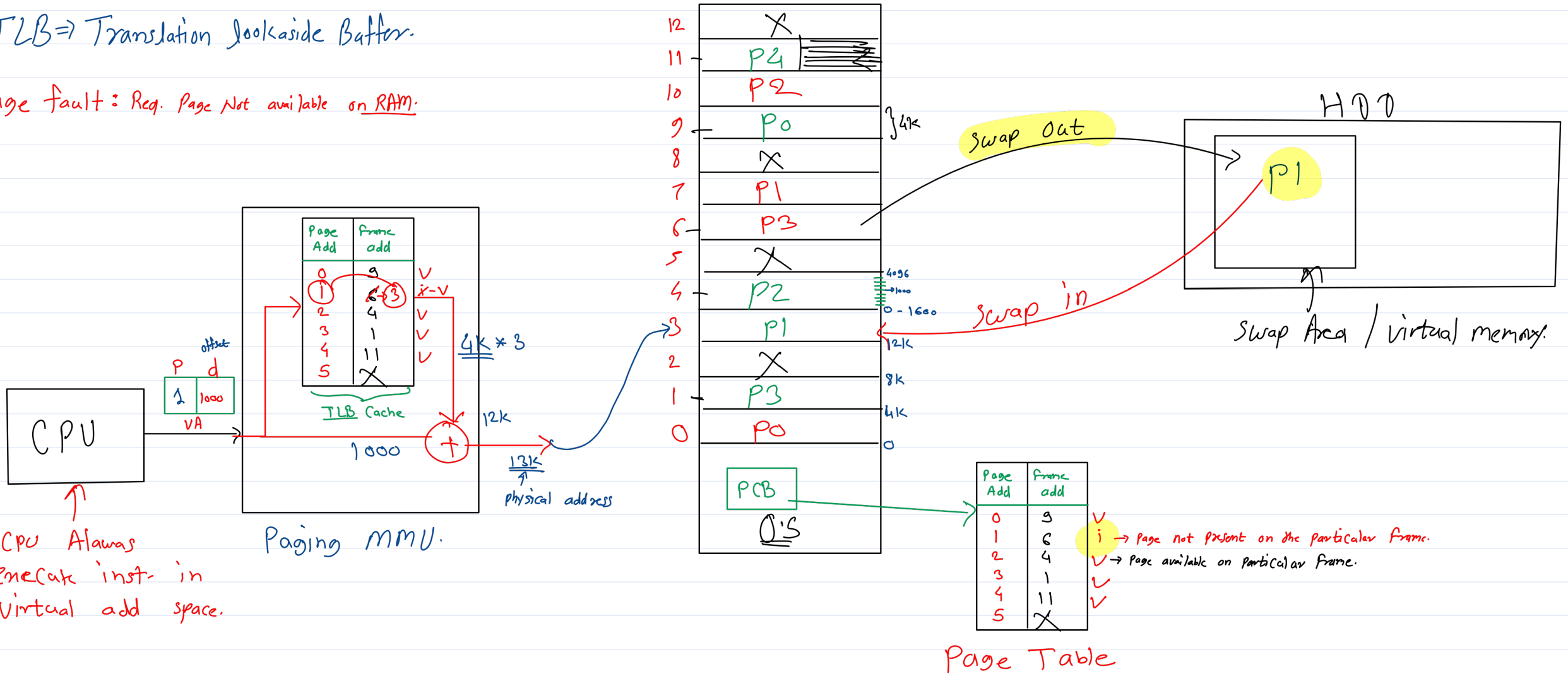
P2

P0
P1
P2
P3



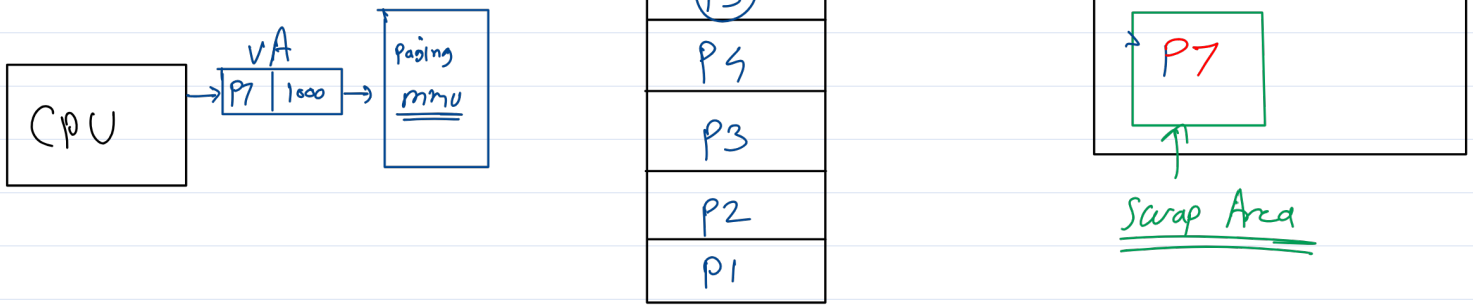
TLB ⇒ Translation lookaside Buffer.

Page fault : Req. Page Not available on RAM.



* Page Fault: if CPU Req. any page and it is not available on main memory (i.e. PTE is invalid), then it is page fault.

* Page Fault:



* Page Relpacement algo.

* Page Fault Handling (OS)

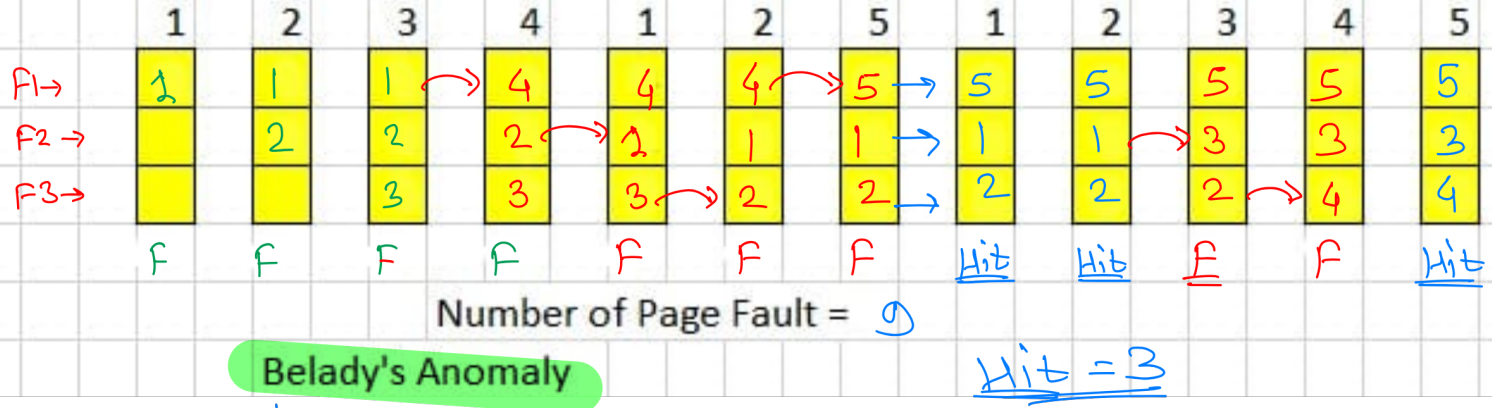
1. Check if address due to which Page fault occar is valid. if not abort the process.
2. Allocate an empty frame.
   - ↳ IF empty frame is not available. then OS needs to swapout some pages from main memory. To deside these OS use Page Relpacement algo.
     1. FIFO
     2. Optimal
     3. LRU.
3. if page in swap area swap in Page in allocated frame.
4. Update Page table entry (PTE)
5. Reexecute instruction due to page fault occured.

Given:

I/P Page Seq : 1 2 3 4 1 2 5 1 2 3 4 5

Num of Frame: 3 & Find Number of Page Fault and Page Hit.

FIFO Page Replacement



Number of Page Fault = 9

Hit = 3

Belady's Anomaly

↳ In FIFO page Replacement algo. it we increase Frame Count the Page Fault will also increase.

Optimal Page Replacement

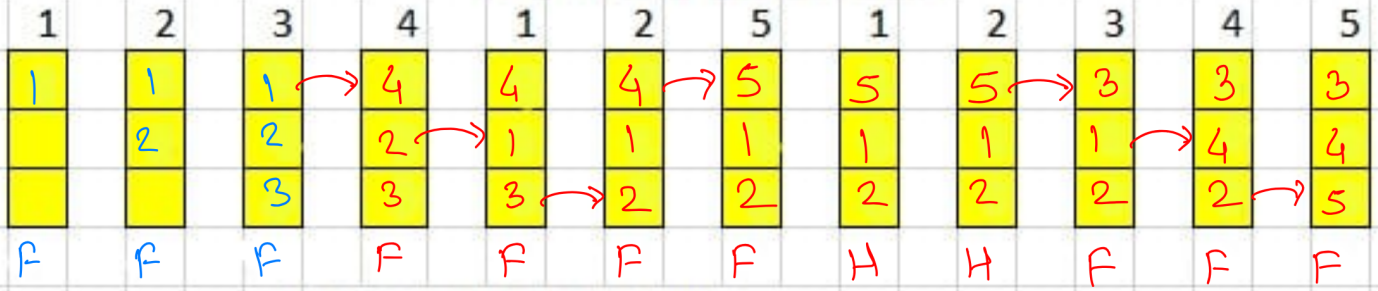
→ practical implementation

Not possible

Number of Page Fault = 7

Hit = 5

LRU Page Replacement



Number of Page Fault = 10 , Hit = 2

Implementation: Stack based approach -- O(n)

Second chance LRU -- Approximate LRU (similar) -- O(1)