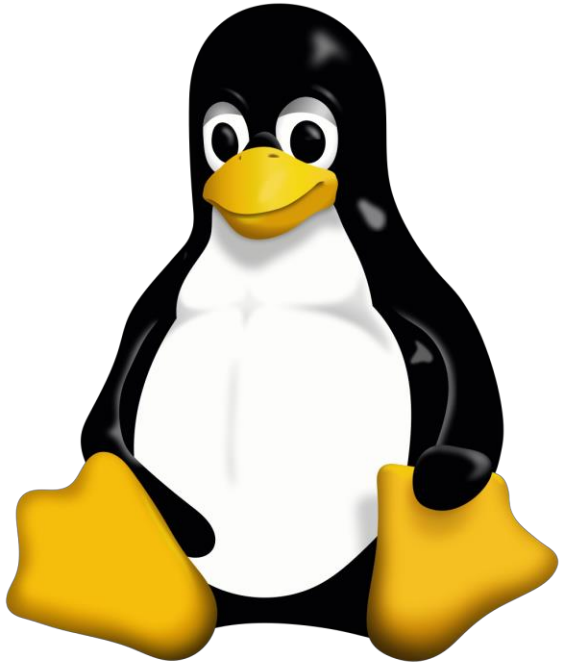
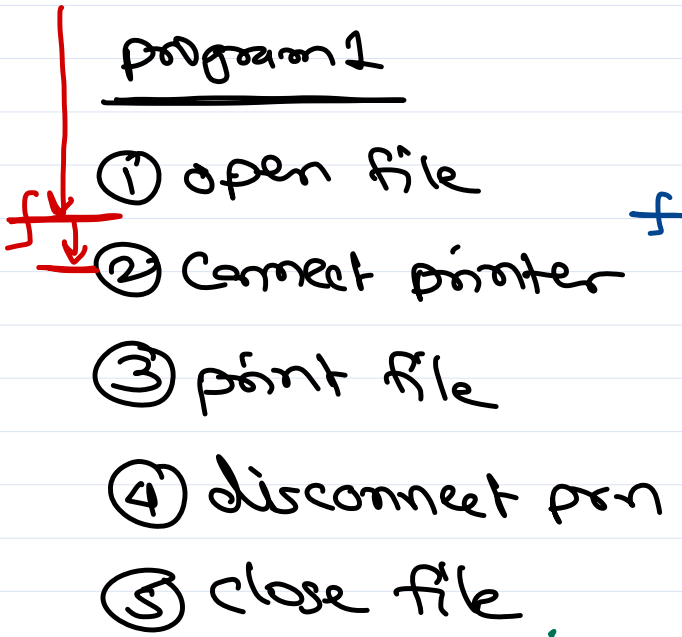


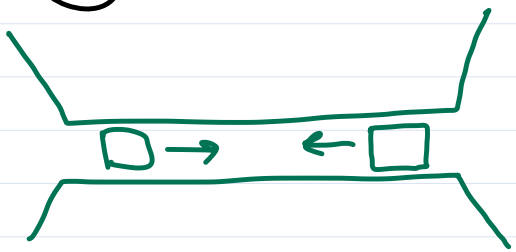
Operating System Concepts

Sunbeam Infotech

Deadlock

deadlock: processes involved in deadlock are permanently waiting
Starvation: process in ready queue not get cpu time due to low pri. (in wait que).





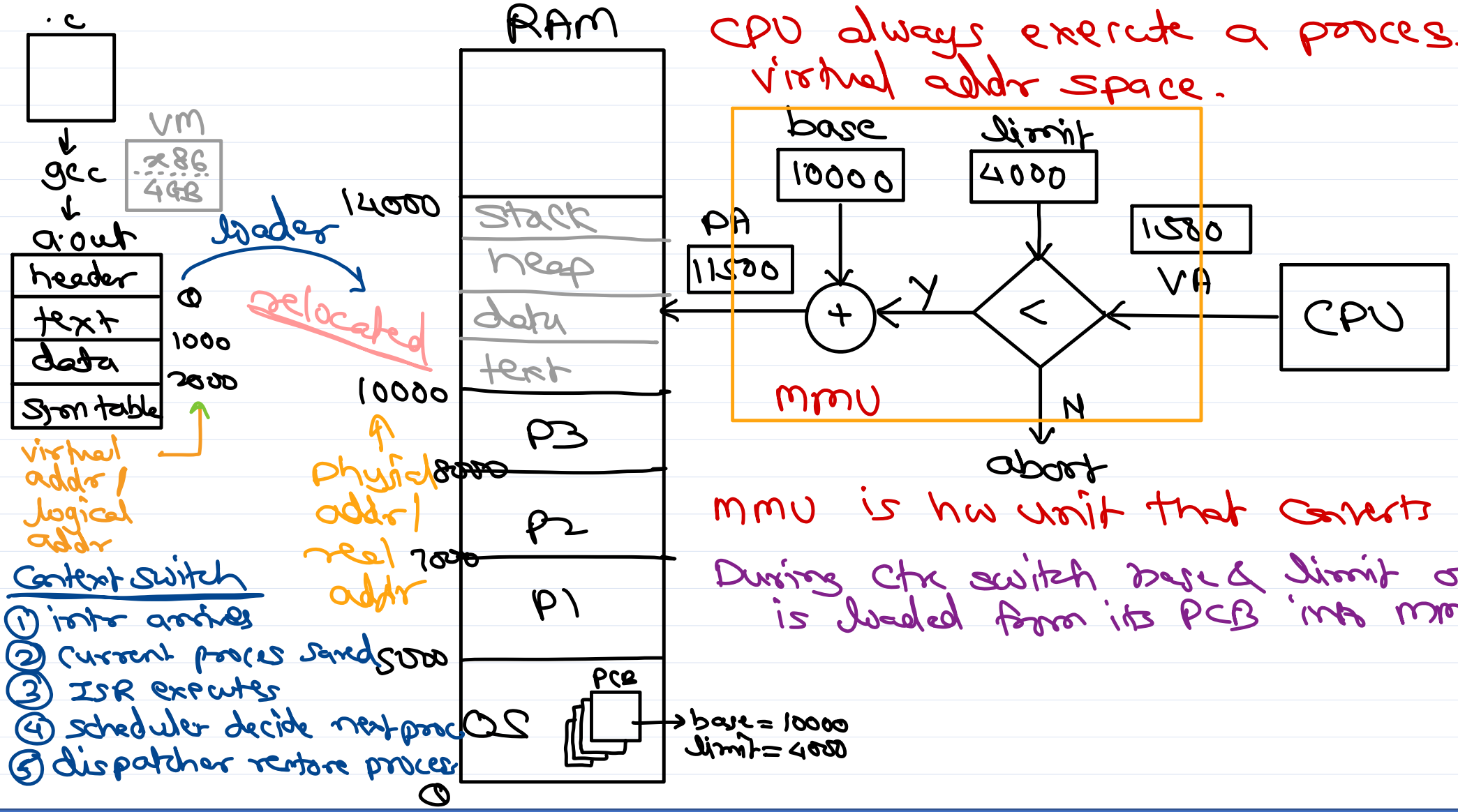
deadlock recovery:
forcible preempt resources & then reassign to processes.

deadlock characteristics
1. no preemption
2. mutual exclusion
3. hold & wait
4. circular wait

deadlock prevention:
- design system (OS + app) in a way that atleast one deadlock cond never holds true.
UNIX OS provide this facility-
can operate multiple sem at once.
i.e. acquire multiple resources at once ⇒ No Hold & wait.

deadlock avoidence:
- processes first informs OS about resources that it will need.
- later processes req resources to OS as and when needed.
- when process req resource, OS checks if this may result in deadlock and if there are chances, OS may deny the req upto some timeout.
- algorithms:
  1. Safe state 2. resource alloc graph
  3. banker's algo.

Memory management



Memory management schemes

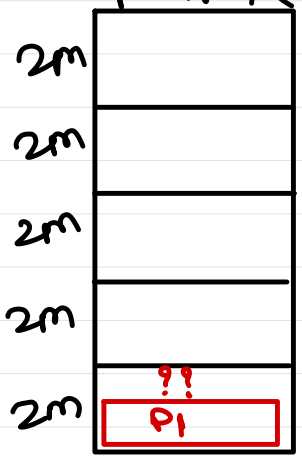
MM schemes depends on avail MMU hw:

MMU hw	MM scheme
① simple	① Contiguous alloc
② segmentation	② segmentation
③ paging	③ paging

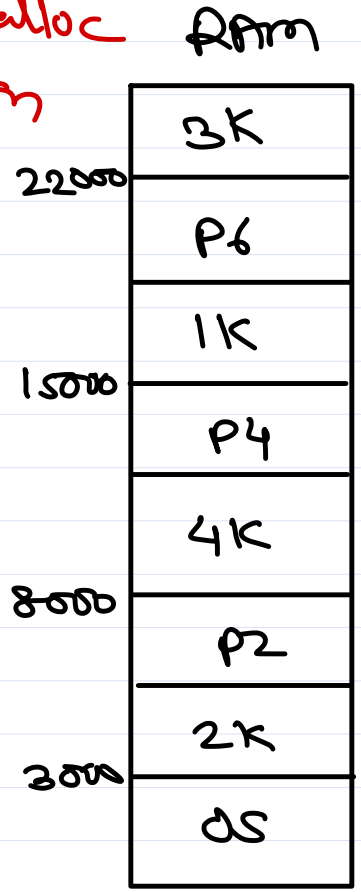
Contiguous allocation:

① fixed Part. method.

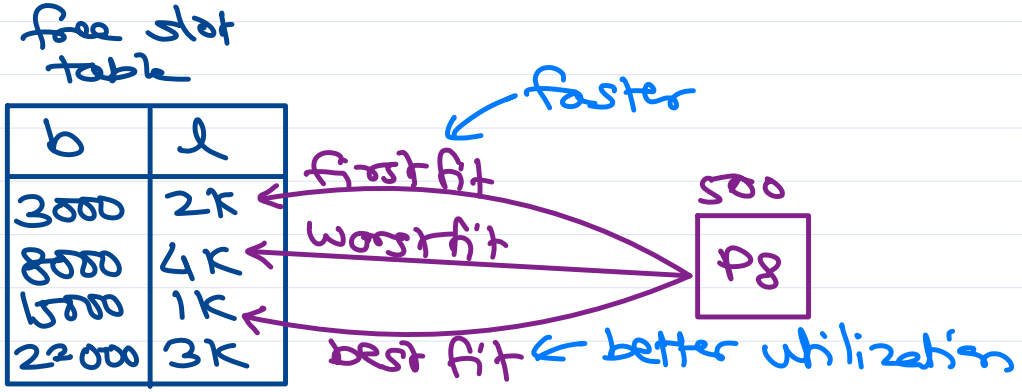
RAM (10M) - fixed num of fixed sized parts.



- 1 proc → 1 Part.
- ✓ simple impl.
- ✗ max processes = num of parts.
- ✗ max proc size = max part size.
- ✗ internal frag.

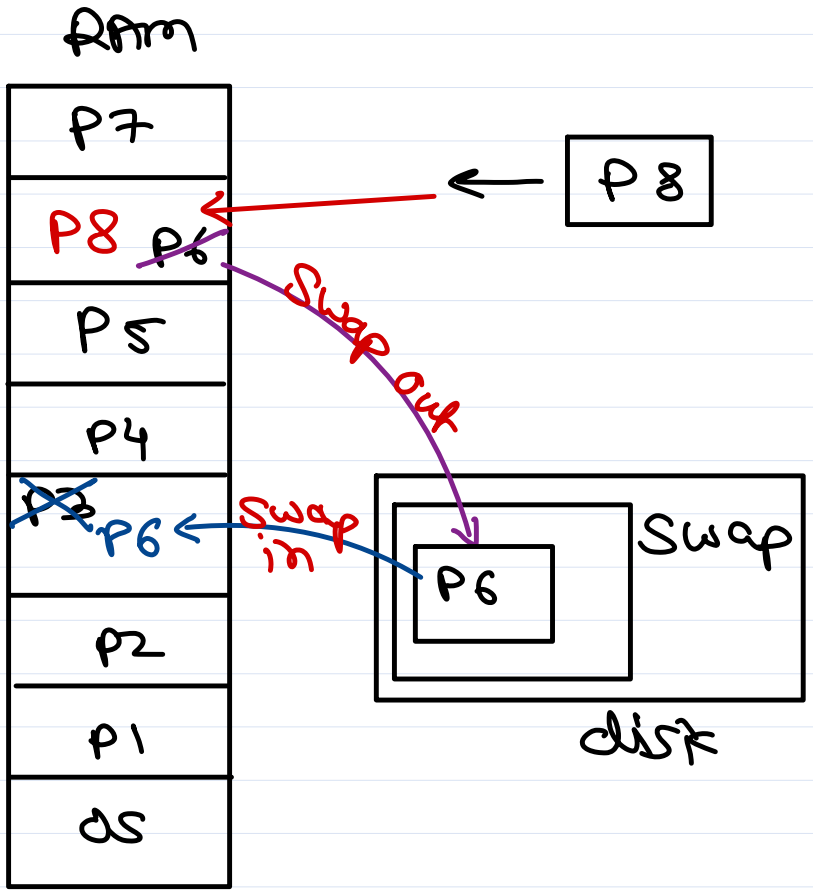


② variable part method/ dynamic alloc method



- ✓ process → avail slot
- ✓ simple (w.r.t. seg. & paging)
- ✗ num of processes = avail RAM size.
- ✗ max proc size = avail RAM size.
- ✓ no 'internal frag.
- ✗ external frag → soln: compaction.

Virtual memory



Portion of disk can be used as extension of main memory to store inactive processes in case of shortage of memory - swap area

a.k.a. virtual memory

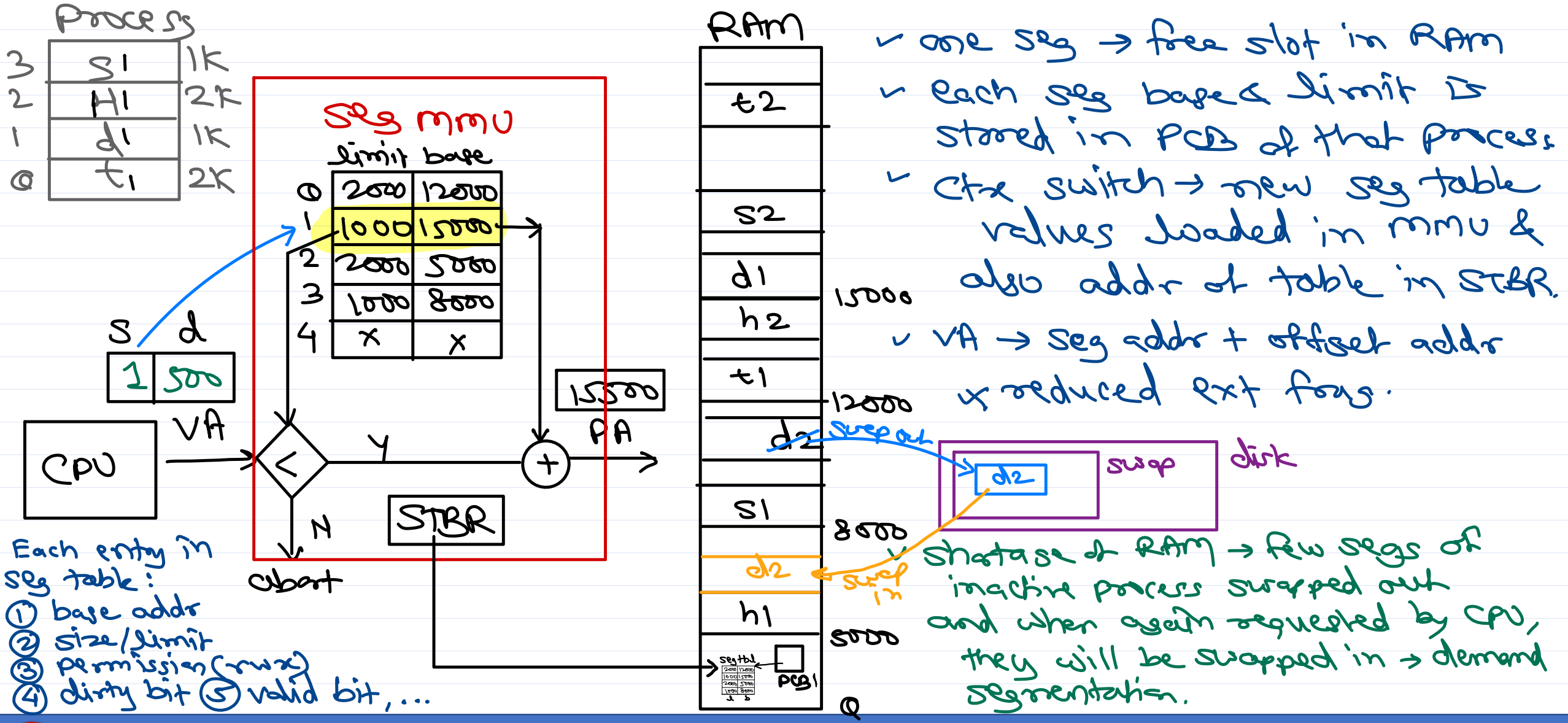
RAM ⇄ (swap out / swap in) Swap area

- Swap file (e.g. Windows) c:\pagefile.sys
- Swap part (e.g. Linux)

OS schedulers

1. job sched → long term sched
2. cpu sched → short term sched
3. swapper → medium term sched

Segmentation

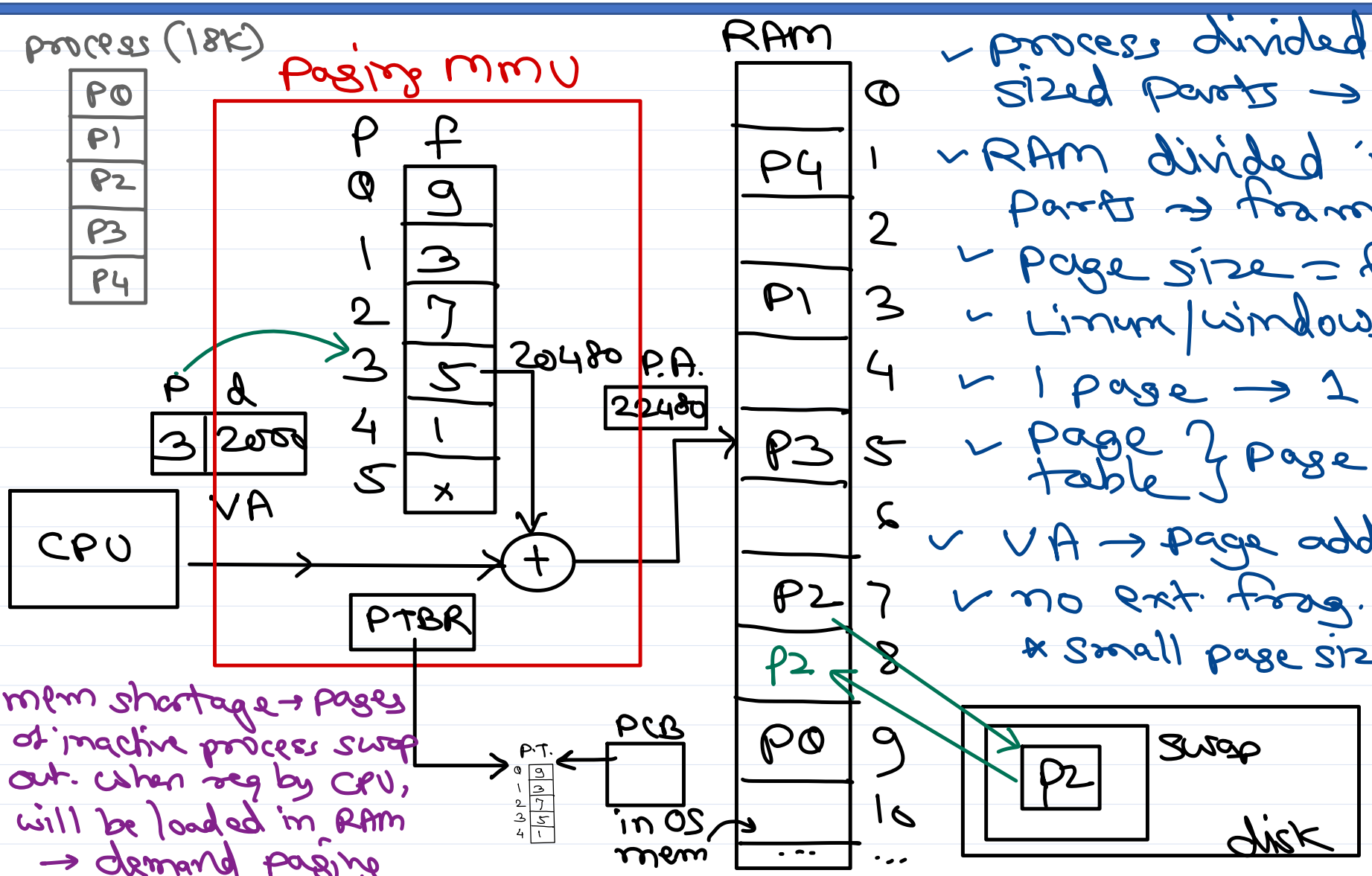


Each entry in seg table:
① base addr
② size/limit
③ permission (rwx)
④ dirty bit ⑤ valid bit, ...

- one seg → free slot in RAM
- each seg base & limit is stored in PCB of that process.
- Ctx switch → new seg table values loaded in mmu & also addr of table in STBR.
- VA → seg addr + offset addr
  - reduced ext frag.

Shortage of RAM → few segs of inactive process swapped out and when again requested by CPU, they will be swapped in → demand segmentation.

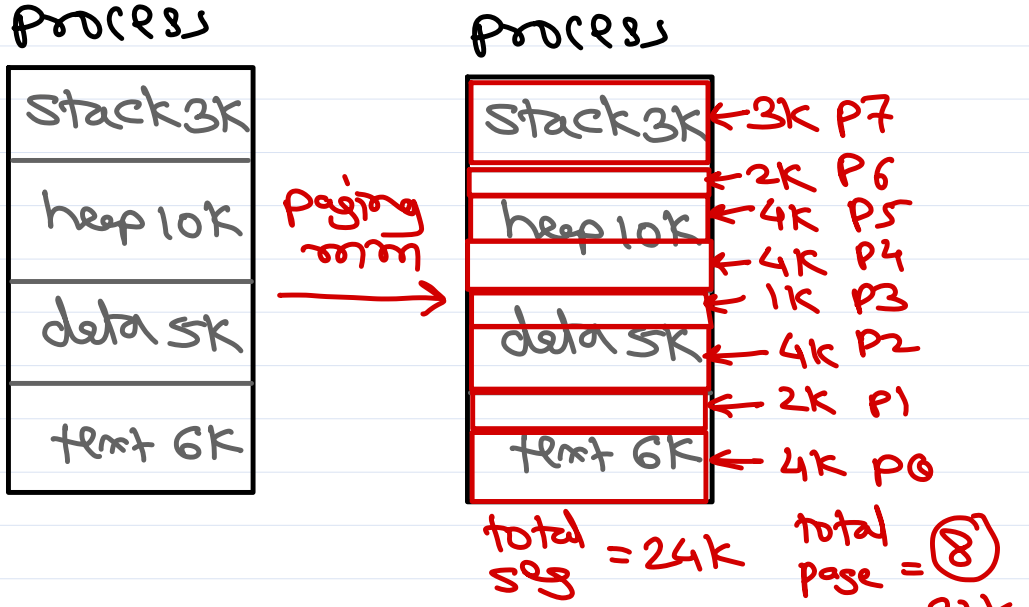
Paging



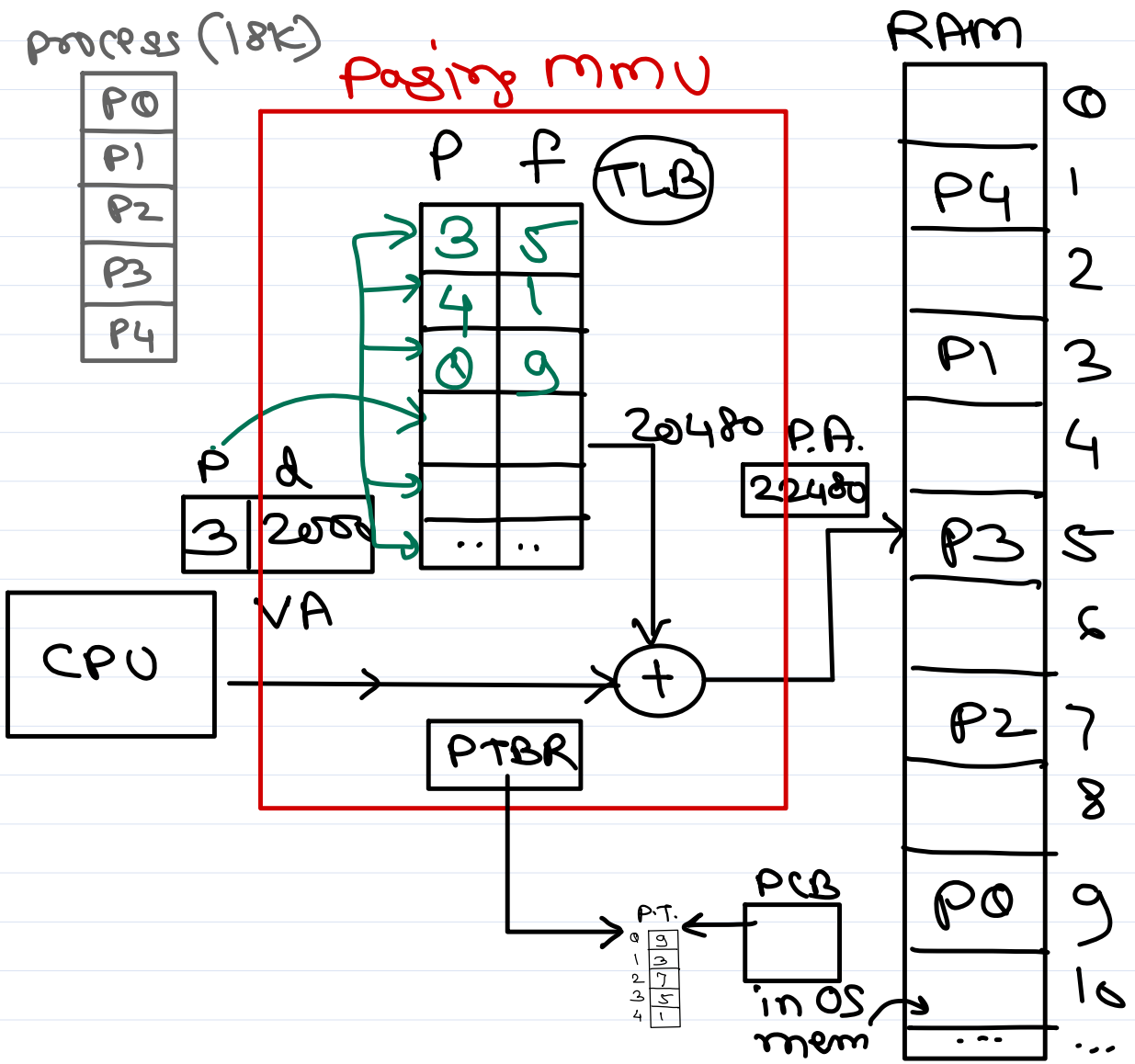
- process divided into small equal sized parts → pages.
- RAM divided into small equal parts → frames.
- page size = frame size.
- Linux/windows: page size = 4KB.
- 1 page → 1 empty frame.
- page table } page addr → frame addr
- VA → page addr + offset addr.
- no ext. frag. x internal frag.
  - * small page size ⇒ less int. frag but bigger page table.

Thrashing: more time spent in swapping (disk IO) than actual CPU execution. Overall system is slow.
Soln: Increase RAM.

Segmented Paging



TLB

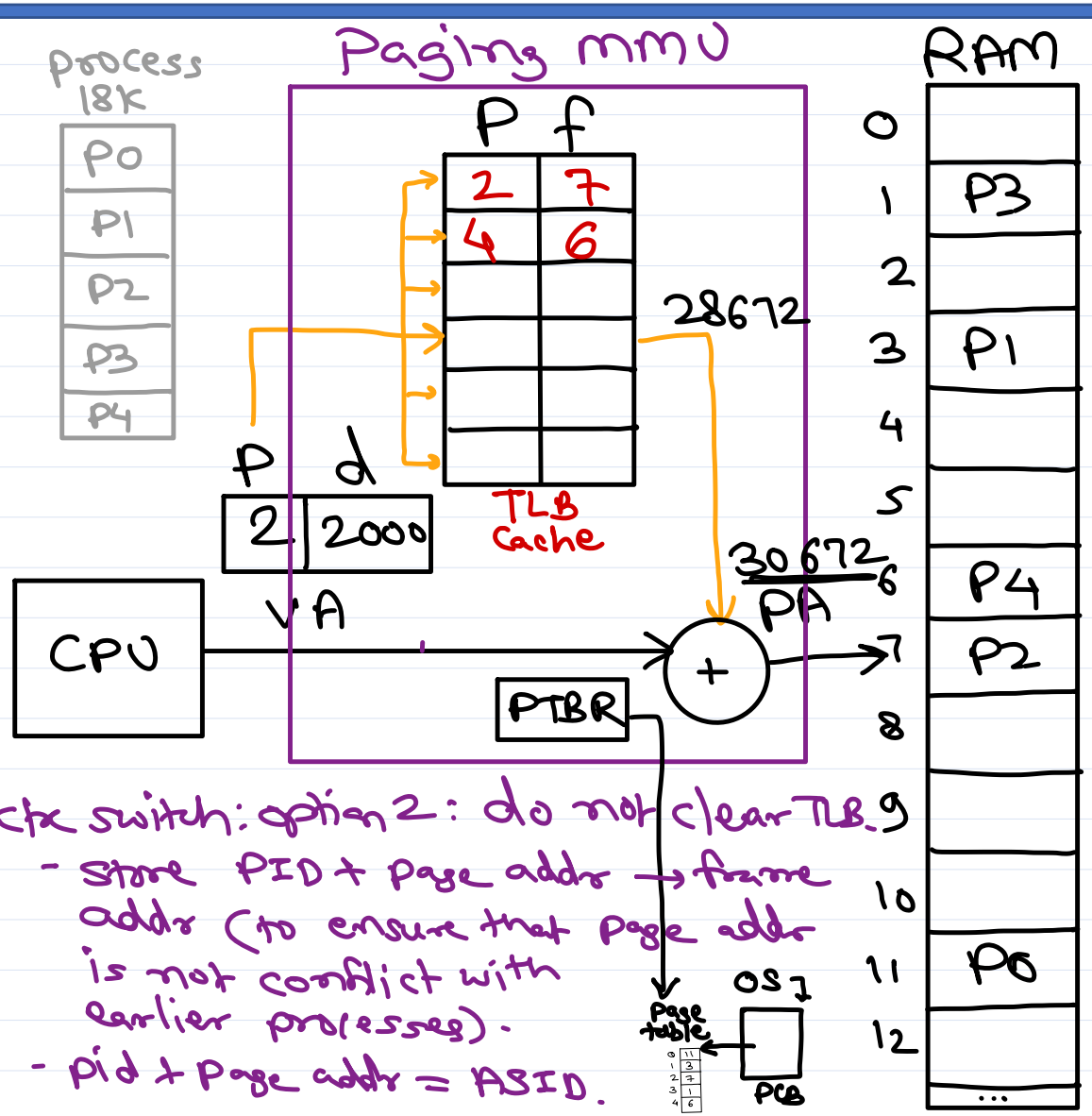


Translation Lookaside Buffer.
- high speed associative cache memory.

Page table entry :

1. frame addr
2. permission (r-w)
3. valid bit
4. dirty bit

TLB cache

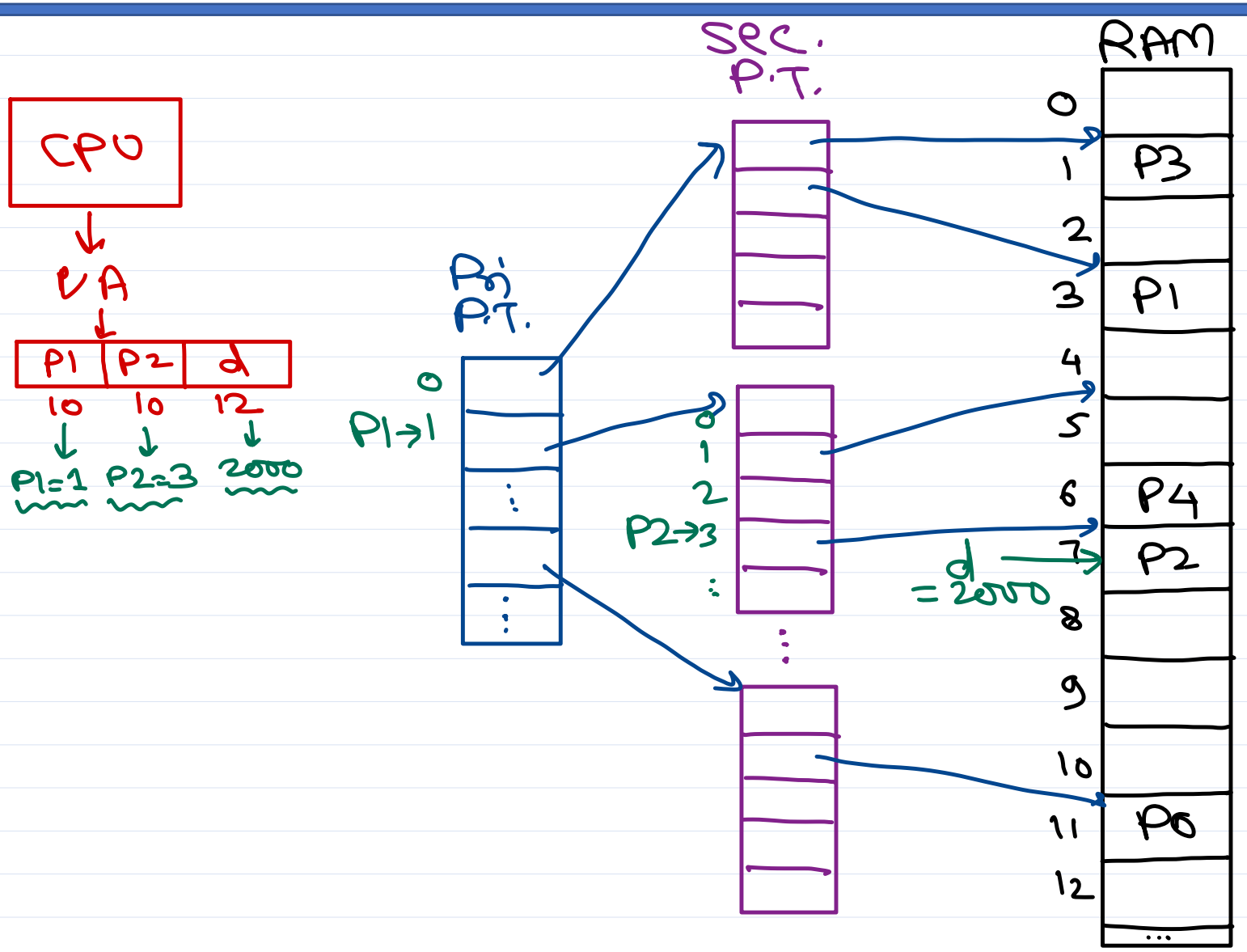


* Paging MMU = TLB Cache.
* TLB cache = High-Speed associative Cache.
  - ✓ Associative = page addr & frame addr mapping
  - ✓ High Speed = page addr compared with all entries at once to find corresponding frame addr
  - ✓ Cache = Limited entries (32/64 entries).
    - If page addr found (TLB hit), frame addr is retrieved.
    - If page addr not found (TLB miss), the process page table is read (using PTBR) and actual entry is copied into TLB cache.
    - If TLB is full, least recently used (LRU) entry is overwritten. Thus contains only recent entries.

Ctx switch: option 1: flush/clear TLB.
- New process CPU req will be TLB miss and then filled from process's page table (in RAM)

Ctx switch: option 2: do not clear TLB.
- store PID + page addr → frame addr (to ensure that page addr is not conflict with earlier processes).
- pid + page addr = ASID.

Two level paging



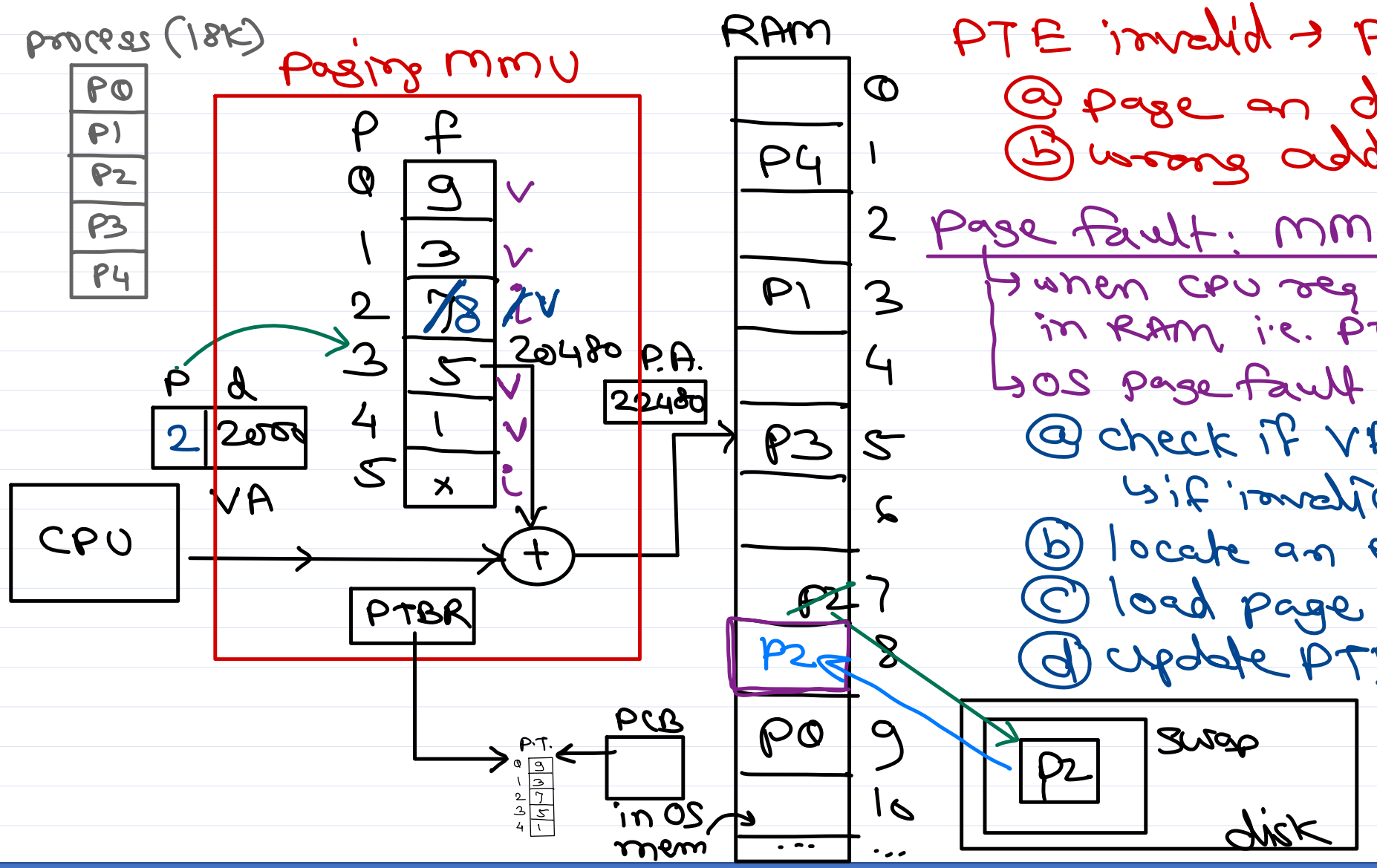
max virtual addresses
= 1024 × 1024 × 4 × 1024
= 4 GB

So virtual addr space of 32-bit processes = 4 GB

Pri PT a.k.a. → Page directory in x86 / L1 page table in ARM

Sec PT a.k.a. → Page table in x86 / L2 page table in ARM

Page Fault



PTE invalid → page not in RAM.

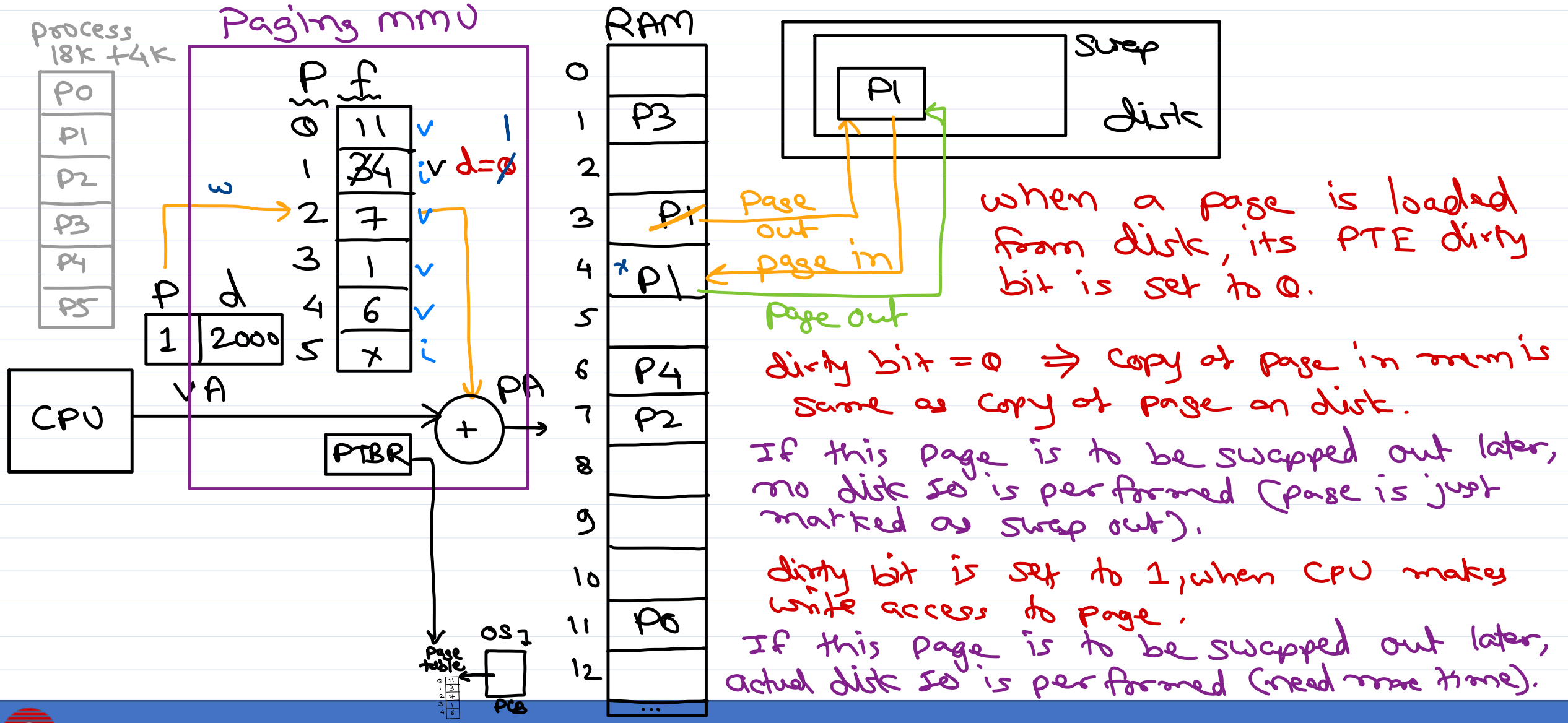
ⓐ page on disk/swap

ⓑ wrong addr (page not in proc)

Page fault: MMU interrupt/exception.

- when CPU req a page that is not in RAM, i.e. PTE is 'invalid'.
- OS page fault handler executed.
  - ⓐ check if VA is valid (dangling ptr).
    - if invalid, abort process.
  - ⓑ locate an empty frame.
  - ⓒ load page in that frame.
  - ⓓ update PTE.
  - ⓔ restart instruct at which fault occurred.

Dirty bit



when a page is loaded from disk, its PTE dirty bit is set to 0.

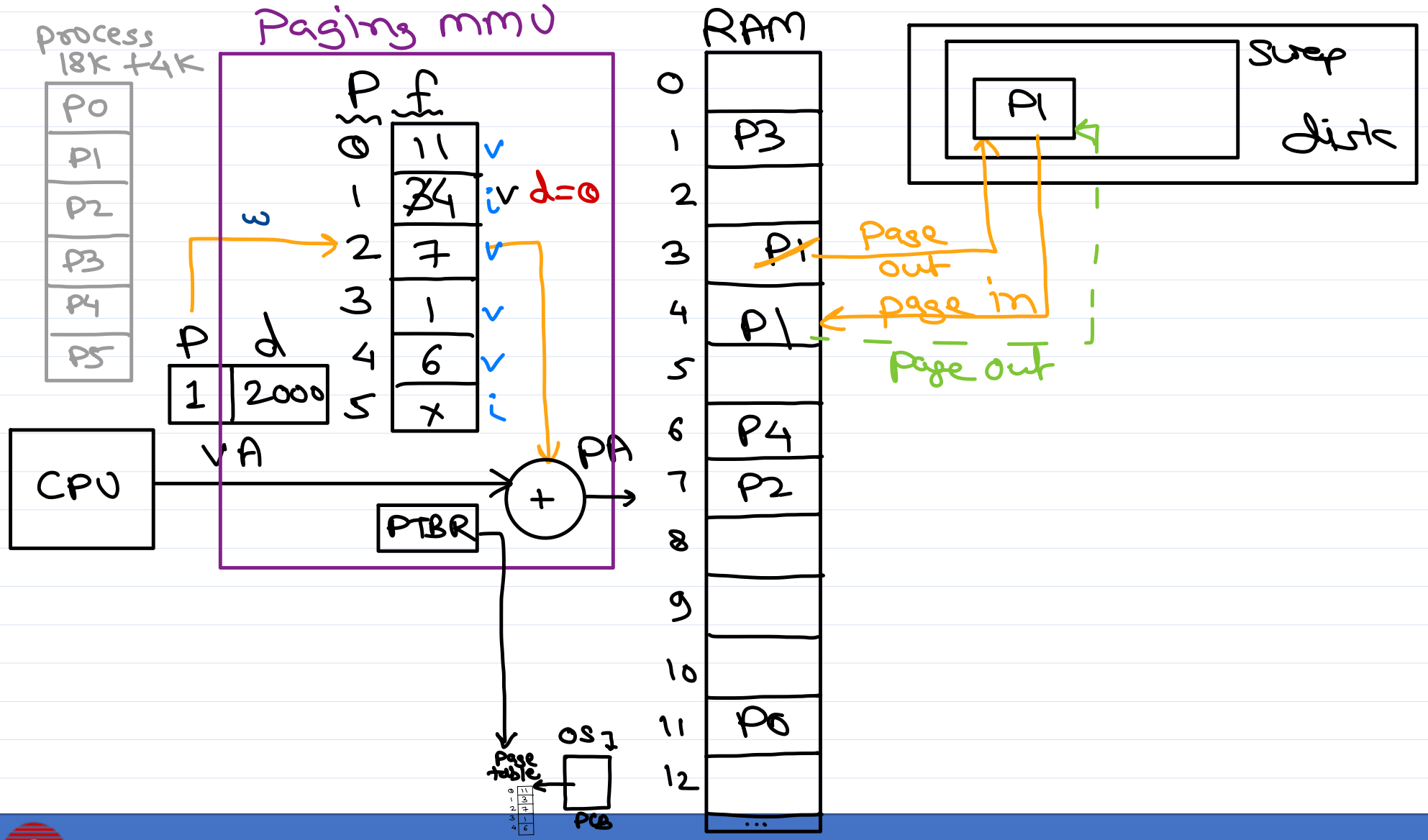
dirty bit = 0 ⇒ Copy of page in mem is same as copy of page on disk.

If this page is to be swapped out later, no disk IO is performed (page is just marked as swap out).

dirty bit is set to 1, when CPU makes write access to page.

If this page is to be swapped out later, actual disk IO is performed (need more time).

Dirty bit


process
18K + 4K
P0
P1
P2
P3
P4
P5
Paging mmu
P
f
0
1
2
3
4
5
11
34
7
1
6
x
v
iv
i
d=0
w
P
d
1
2000
VA
CPU
PA
+
PTBR
Page table
OS
PCB
RAM
0
1
2
3
4
5
6
7
8
9
10
11
12
P3
P1
P1
P4
P2
P0
Swap
disk
P1
Page out
page in
Page out

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>