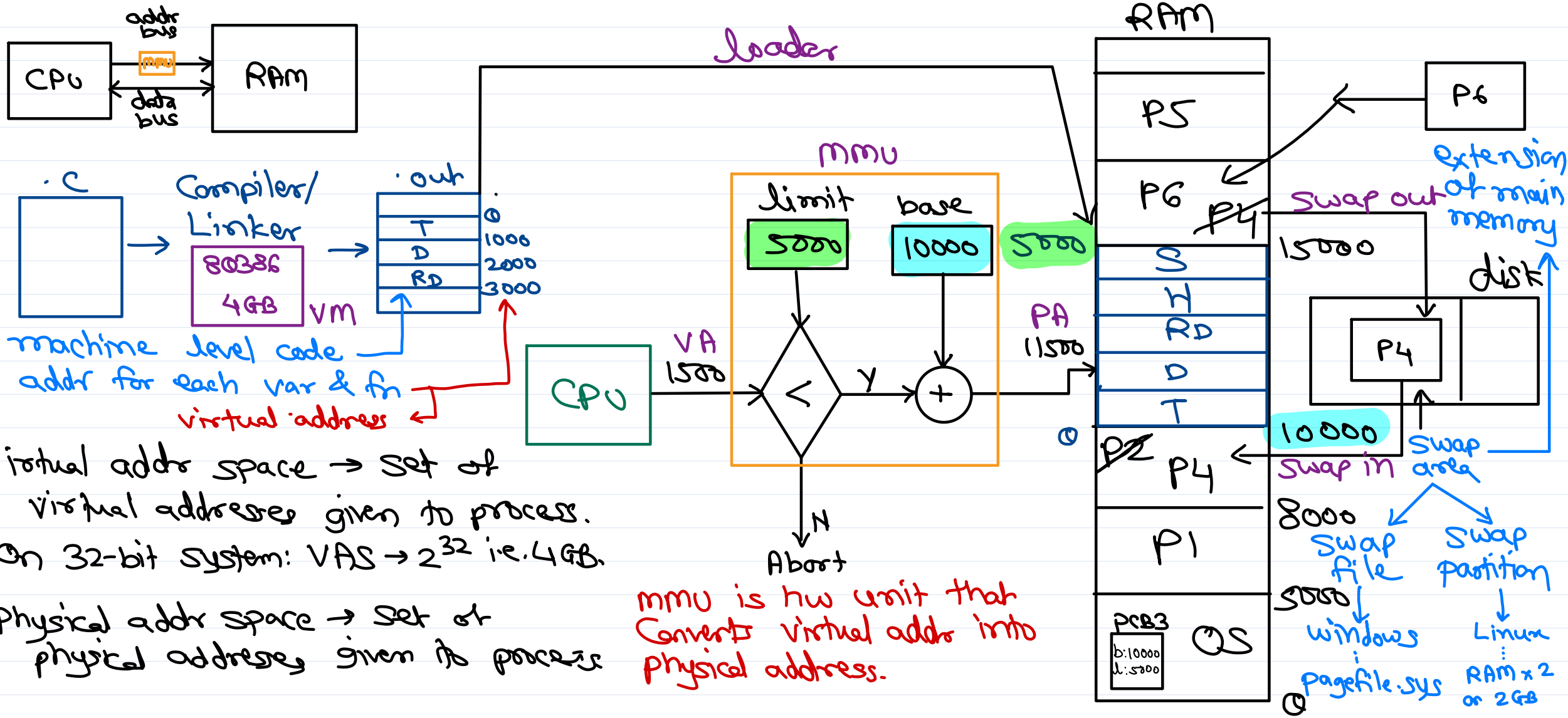
Embedded Operating Systems

Trainer: Nilesh Ghule

CPU always execute a process in its virtual addr space.



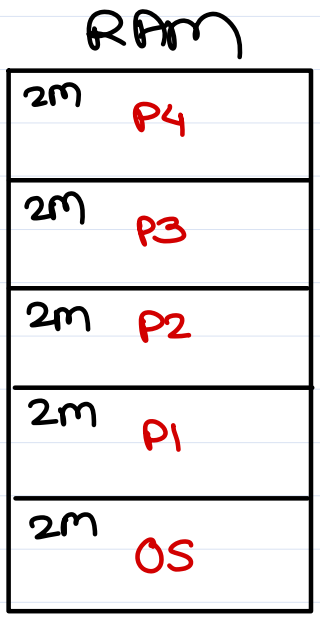
Virtual addr space → Set of virtual addresses given to process.

On 32-bit system: VAS → 2^{32} i.e. 4GB.

Physical addr space → Set of physical addresses given to process

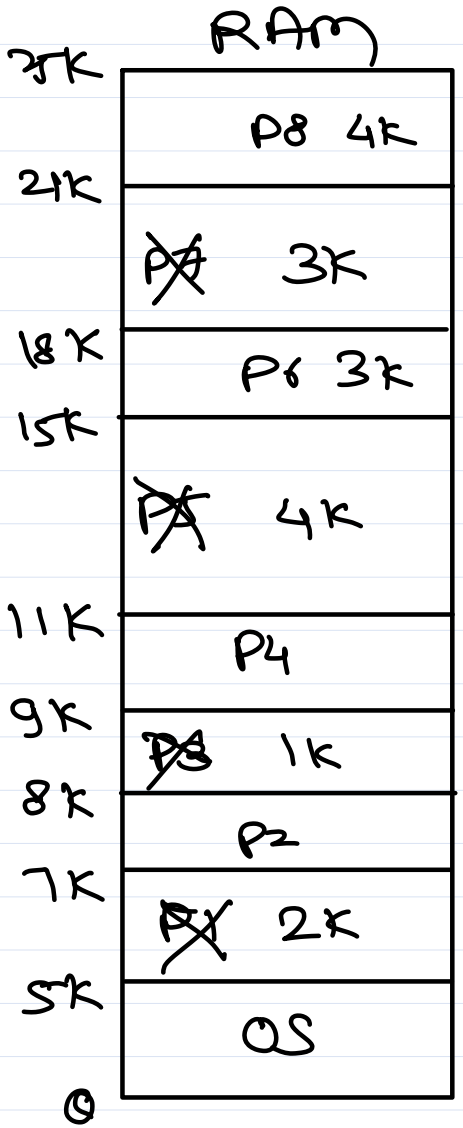
mmu is hw unit that converts virtual addr into physical address.

Fixed partition



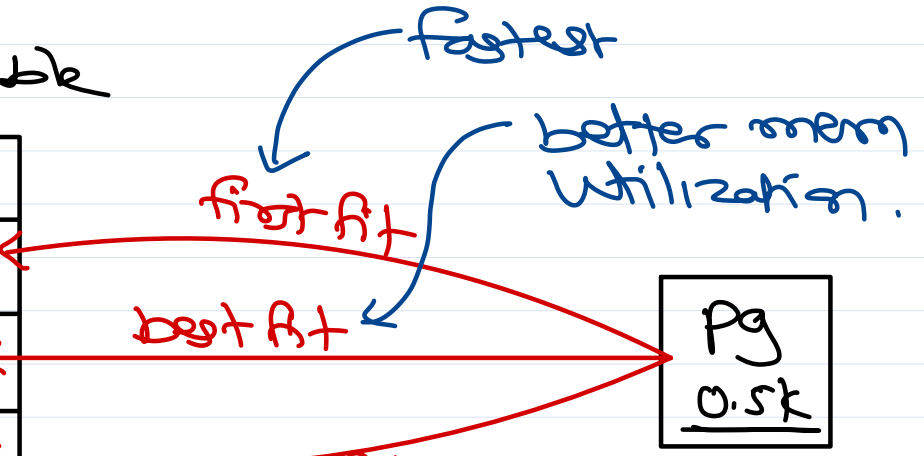
- ✓ Fixed number of fixed partitions.
- ✓ Simple mem mgmt
- ✗ Max num of processes = num of partitions.
- ✗ Max size of process = max size of part.
- ✗ internal fragmentation.

Variable partition



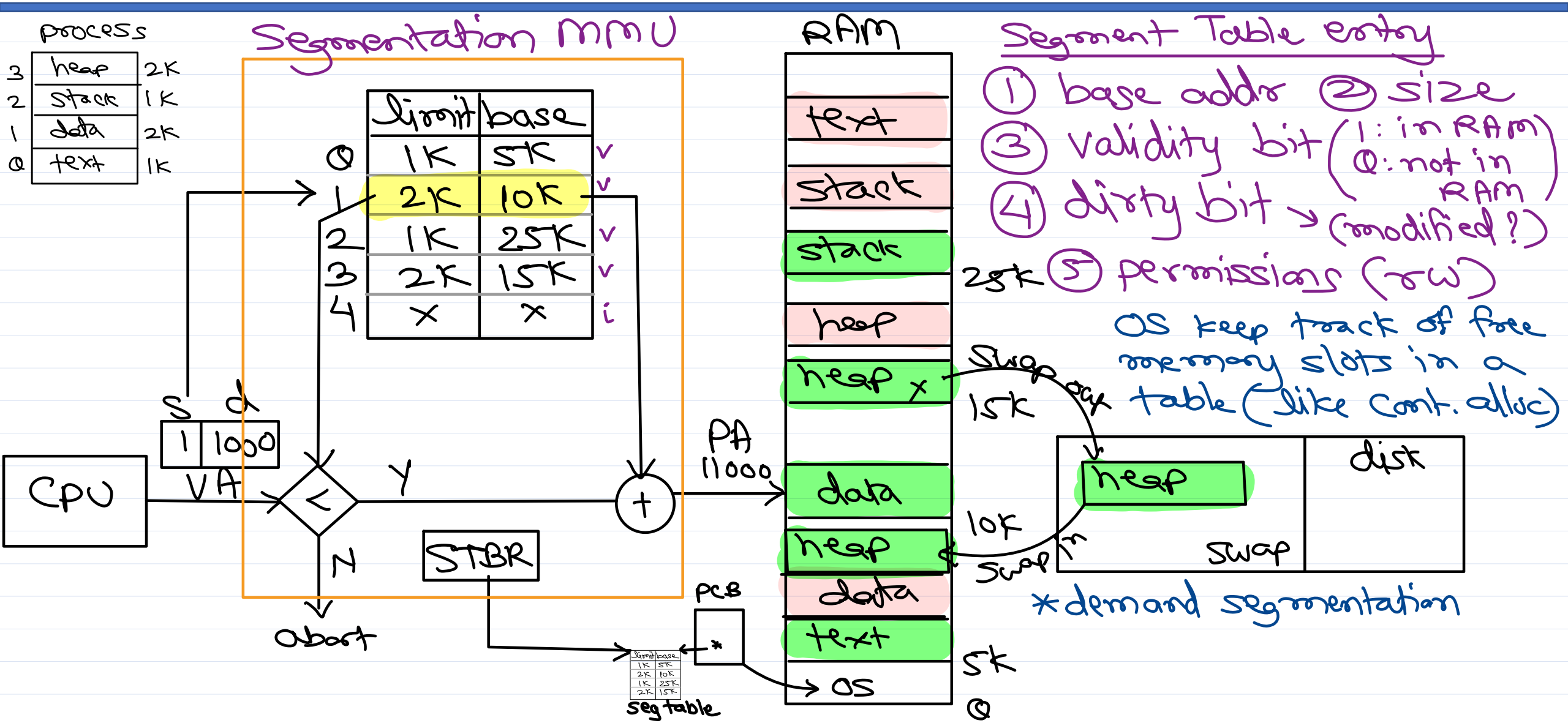
free slot table

base	size
5K	2K
8K	1K
11K	4K
18K	3K

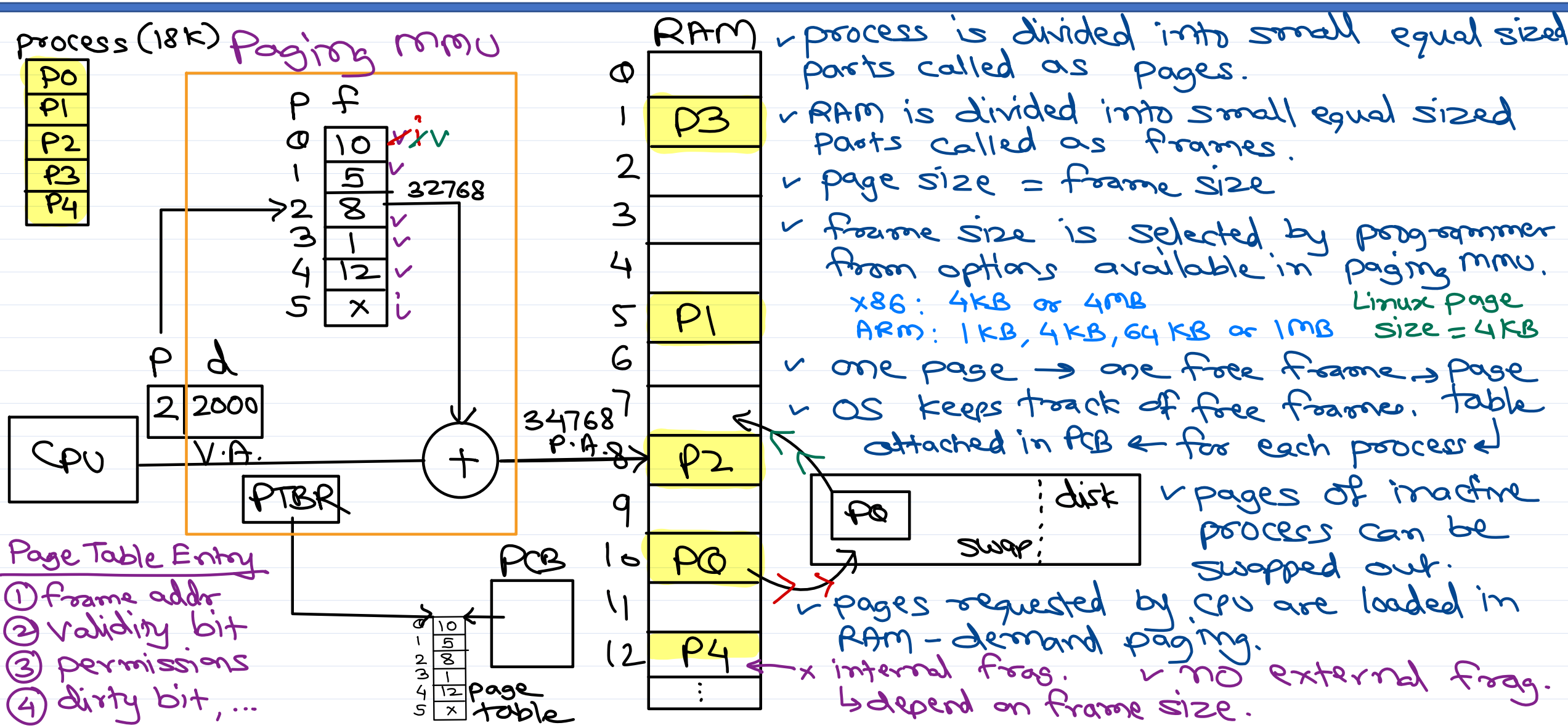


- ✓ No fixed partition. Allocate mem to the process when required as per availability.
- ✓ Max num of processes = depends on RAM size.
- ✓ Max size of processes = depends on avail RAM.
- ✓ No 'internal frag.
- ✗ Process may not grow at runtime.
- ✗ External frag: Amount mem required for process is avail but not contiguous.
  - Compaction - moving processes in RAM to get max free block.

Segmentation

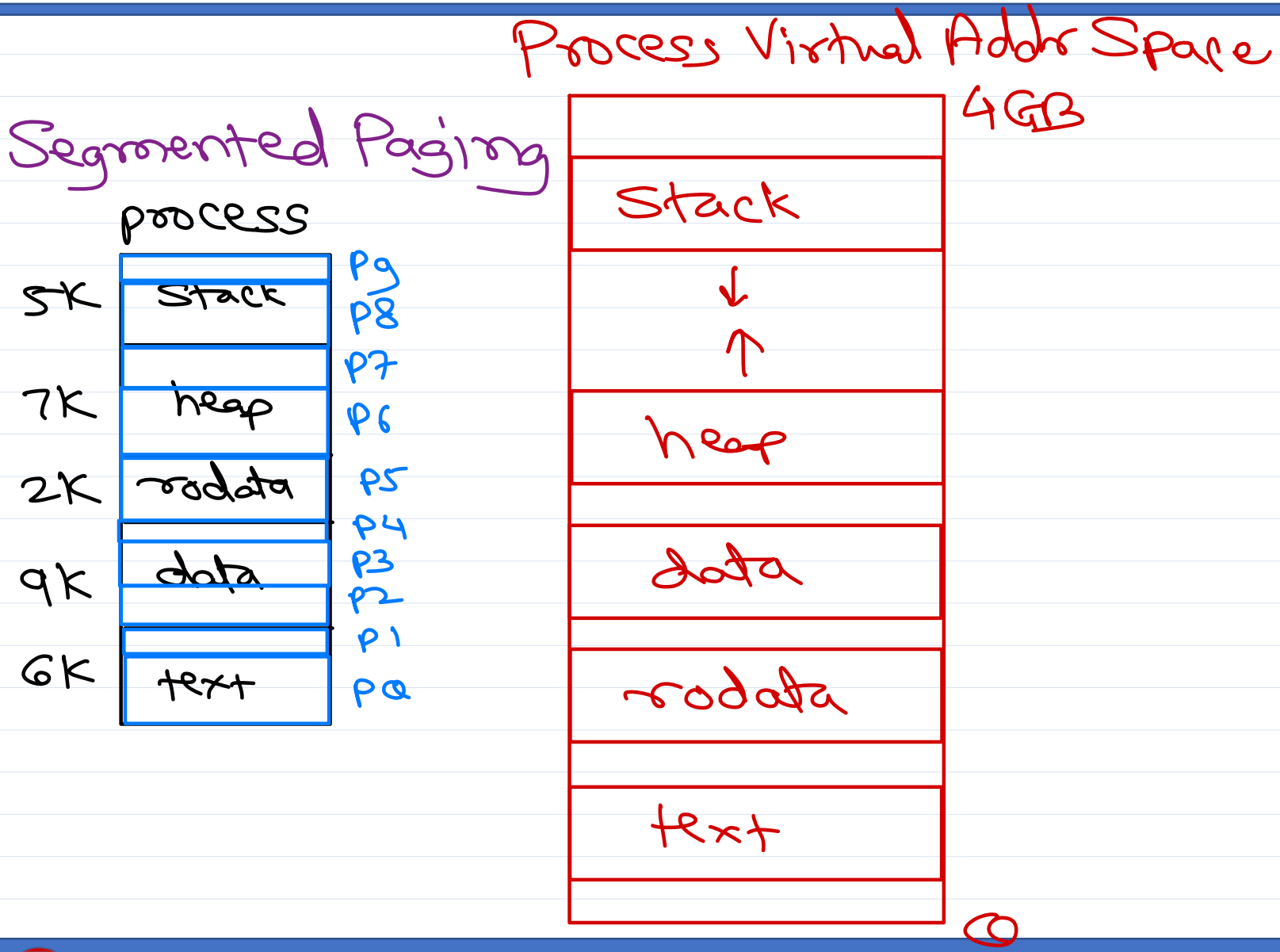


Paging



- process is divided into small equal sized parts called as pages.
- RAM is divided into small equal sized parts called as frames.
- page size = frame size
- frame size is selected by programmer from options available in paging mmu.
  - x86: 4KB or 4MB
  - ARM: 1KB, 4KB, 64KB or 1MB
  - Linux page size = 4KB
- one page → one free frame → page
- OS keeps track of free frames. table attached in PCB ← for each process
- pages of inactive process can be swapped out.
- pages requested by CPU are loaded in RAM - demand paging.
- x internal frag. ✓ no external frag.
  - ↳ depend on frame size.

Segmented paging



Process size = 40 MB

Page size = 4 KB

Num of pages = $\frac{40M}{4K} = \frac{10 \times 1024 \times 1024}{4 \times 1024}$

= 10240 pages.

1 P.T. entry size = 4 bytes

Num of PTEs = 10240

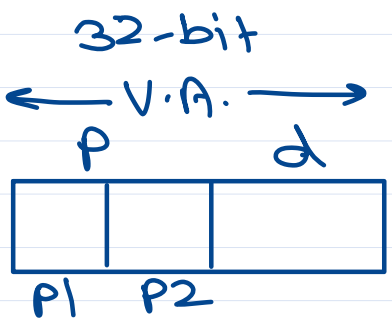
Size of P.T. = 10240 × 4

= 40960 = 40 KB

Num of pages for P.T. = $\frac{40\ KB}{4\ KB}$

= 10 pages.

Two-level paging



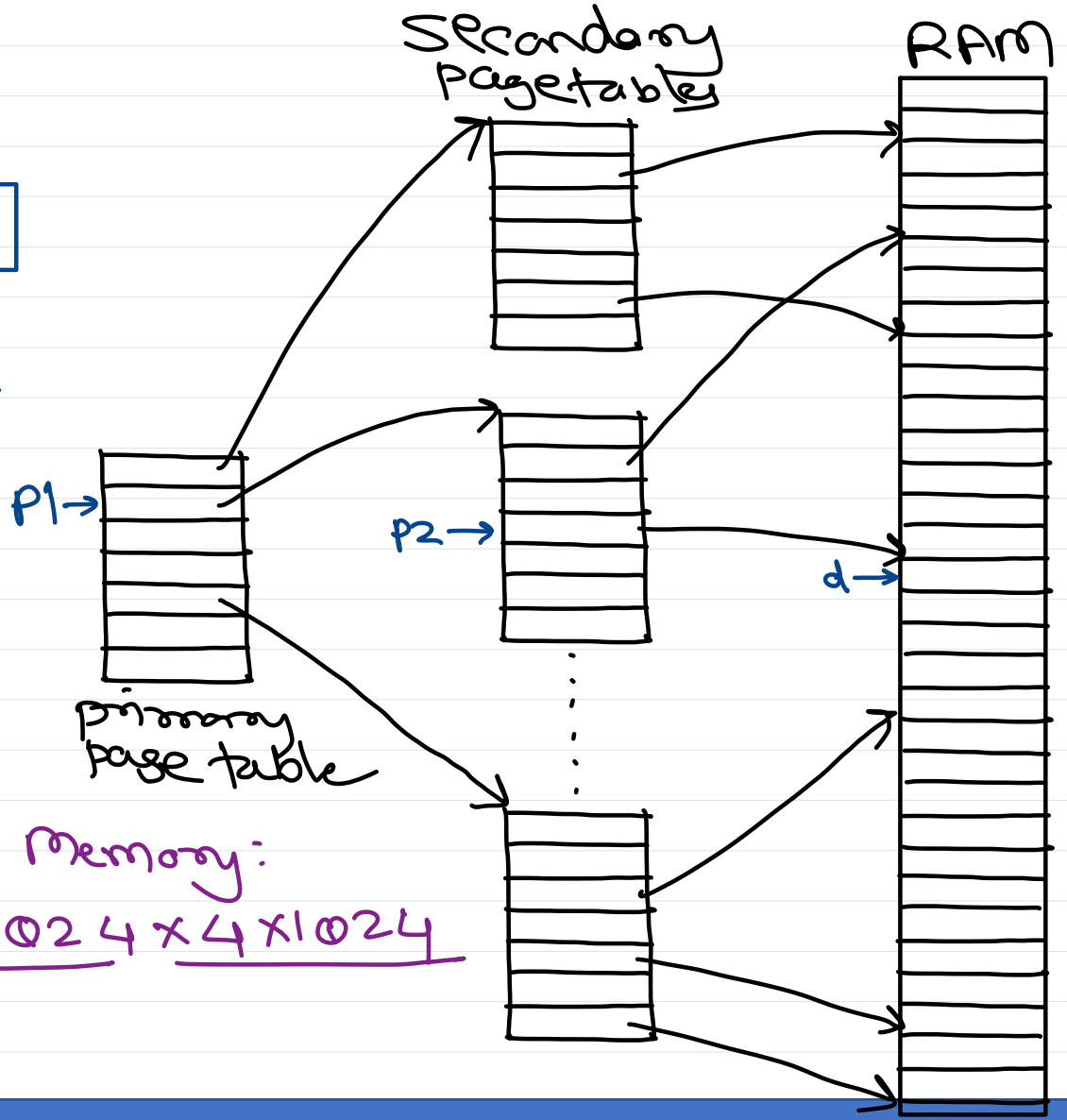
$2^{32} = 4GB$

Virtual address space

Max Process Memory:

= 1024 x 1024 x 4 x 1024

= 4 GB



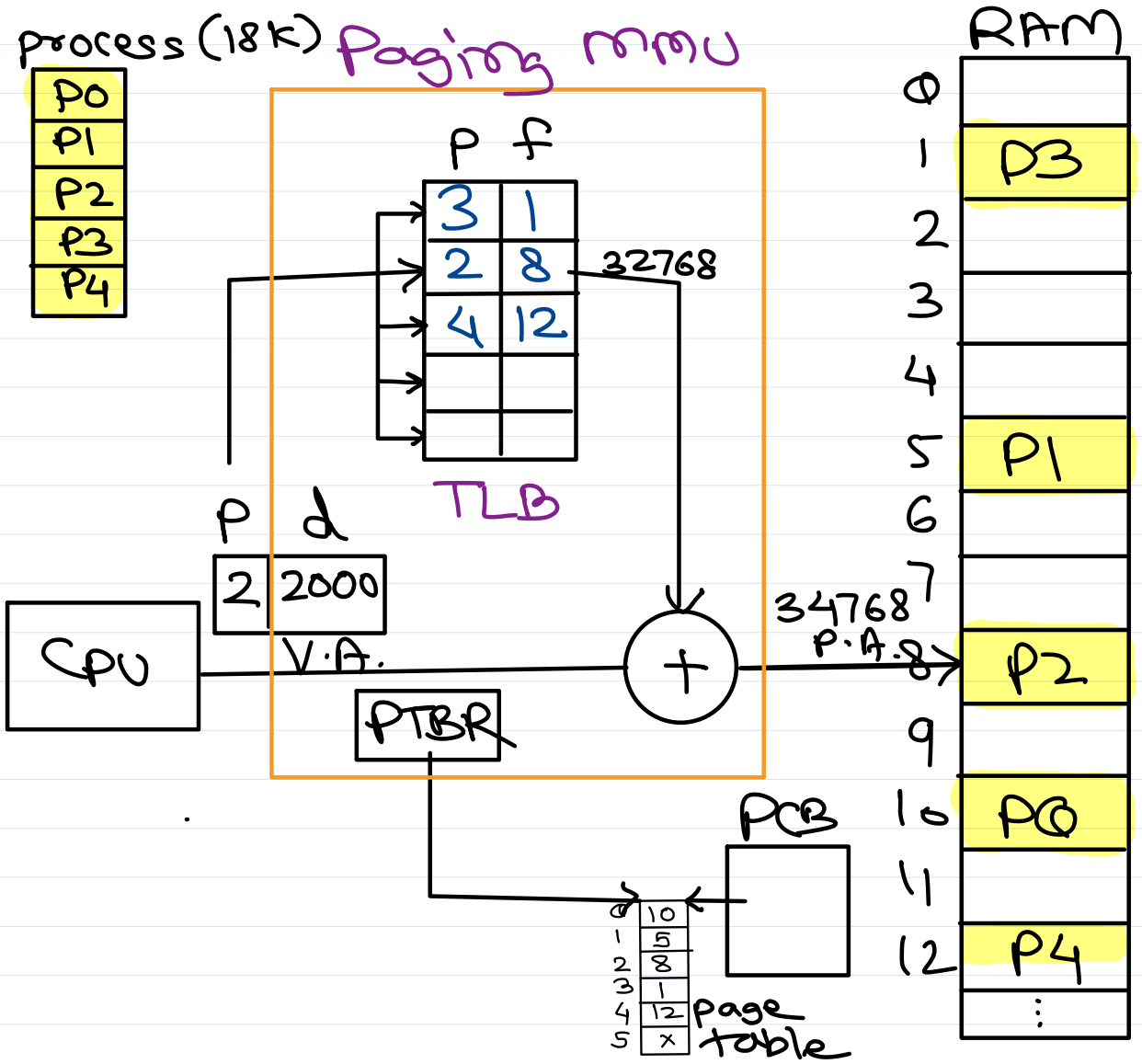
arm x86

sec PT = L2 PT = page table

pri PT = L1 PT = page dir.

* On x86_64 architecture, 2-level paging is not sufficient/applicable.
* Here 3-level paging is used in Paging mmu and Linux OS.

Translation Look-aside Buffer



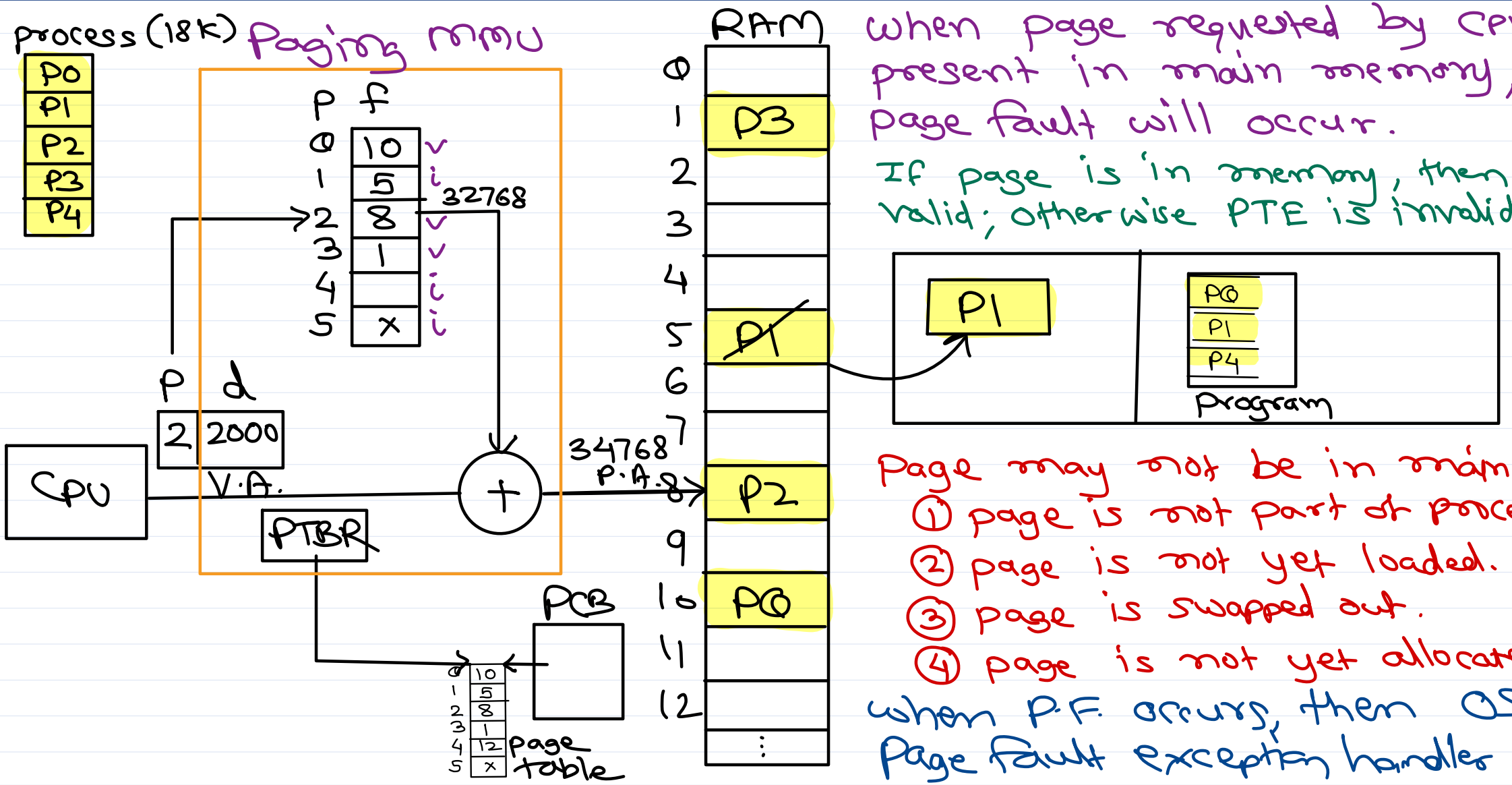
High Speed Associative Cache

page addr ↔ frame addr

↳ all entries are compared at once with requested page addr.

- limited storage - to store recently accessed page & frame addresses.
- when cache is full, old entries are over written (LRU).
- if req addr is found in cache, its frame addr will get immediately (hit); if req addr is not found, its frame addr taken from page table in RAM (using PTBR) and copied into cache (miss)

Page fault



When page requested by CPU is not present in main memory, then page fault will occur.

If page is in memory, then PTE is valid; otherwise PTE is invalid.

Page may not be in main mem:

1. page is not part of process's VAS.
2. page is not yet loaded.
3. page is swapped out.
4. page is not yet allocated.

When P.F. occurs, then OS's Page fault exception handler is executed.

Page fault handling

① check if VA for which fault occured is valid (in VAS) or not. ← validity fault
   if not, send SIGSEGV signal to process.

② check if appropriate perms are available for requested page. ← protection fault
   if not, send SIGSEGV signal to process.

③ allocate an empty frame.

④ if page is on disk, then load it in that frame.

⑤ Modify Page table entry.
   - frame address
   - valid bit = 1

⑥ Restart the instruction at which page fault occurred.

→ check from process's VAD list. If addr is not in range of any seg addr, then the addr is invalid.

→ check from process's VAD list. If seg. perm are read only (r--) and write operation is done, then protection fault.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>