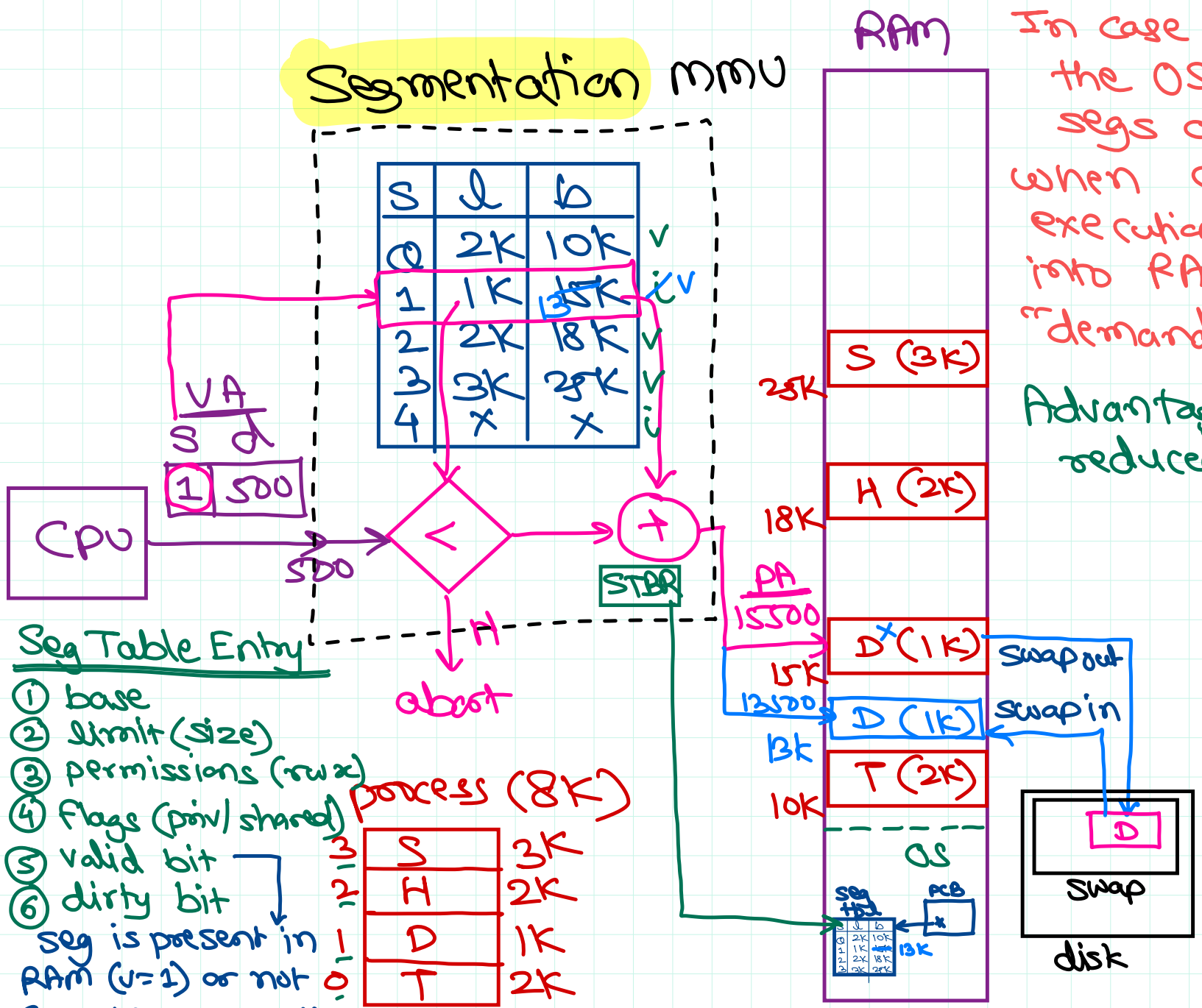
Embedded Operating Systems

Trainer: Nilesh Ghule

Segmentation MMU



S	l	b	
0	2K	10K	v
1	1K	13K	iv
2	2K	18K	v
3	3K	25K	v
4	x	x	i

Seg Table Entry
1. base
2. limit (size)
3. permissions (rwx)
4. Flags (priv/shared)
5. valid bit
6. dirty bit

seg is present in RAM (v=1) or not (v=0 i.e. swap out).

process (8K)

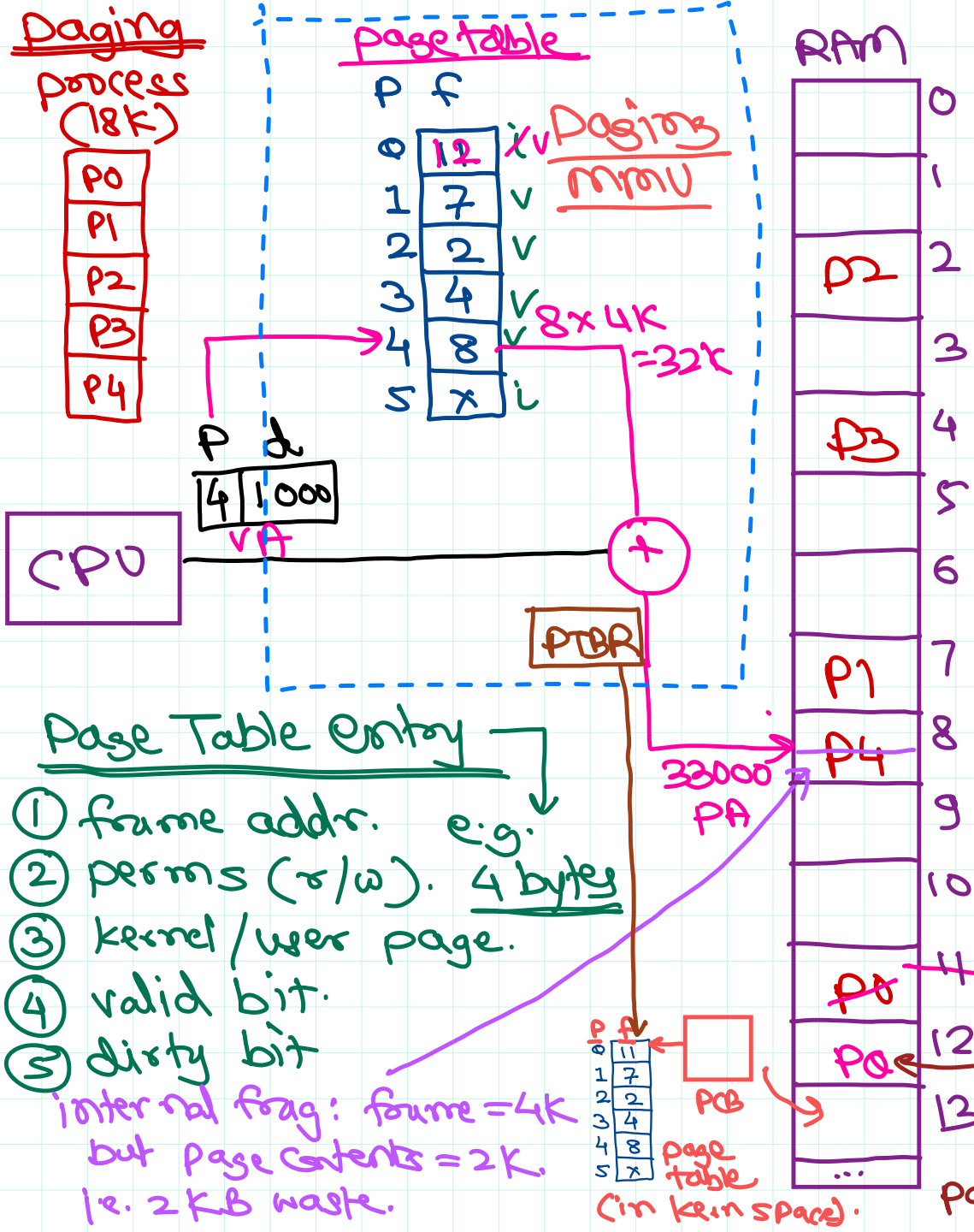
3	S	3K
2	H	2K
1	D	1K
0	T	2K

In case shortage of mem in RAM, the OS will swap out one/more segs of 'inactive process(es).
When CPU req it next time for execution, it will be loaded back into RAM (swap in). This is called as "demand segmentation".

Advantage of Seg: External frag is reduced.

Nilesh Ghule

Paging



Page Table Entry

1. frame addr. e.g. 4 bytes
2. perms (r/w).
3. kernel/user page.
4. valid bit.
5. dirty bit

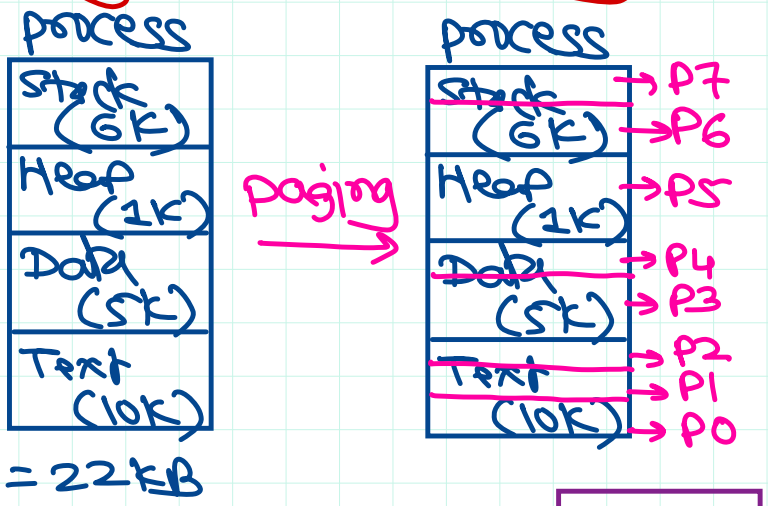
internal frag: fourne = 4K
but page contents = 2K.
i.e. 2KB waste.

1. RAM is divided in small equal sized partitions called as frames.
2. Process is divided in small equal sized parts called as pages. Page size is same as frame size.
3. OS allocated one page to one empty frame. To keep track, page & frame addr mapping is stored in a table - called as - page table.
4. page table is associated in PCB of process
5. During the switch page table entries of scheduled process will be loaded into mmu.
6. Page size depends on CPU arch
   - a. x86 arch → 4KB or 4MB.
   - b. ARM arch → 1K, 4K, 64K or 1MB.

In case of mem shortage, OS swap out one/more pages of inactive process. When CPU req page (that is not in RAM), the page will be loaded from disk/swap → Demand Paging

Nilesh Ghule

Segmented Paging

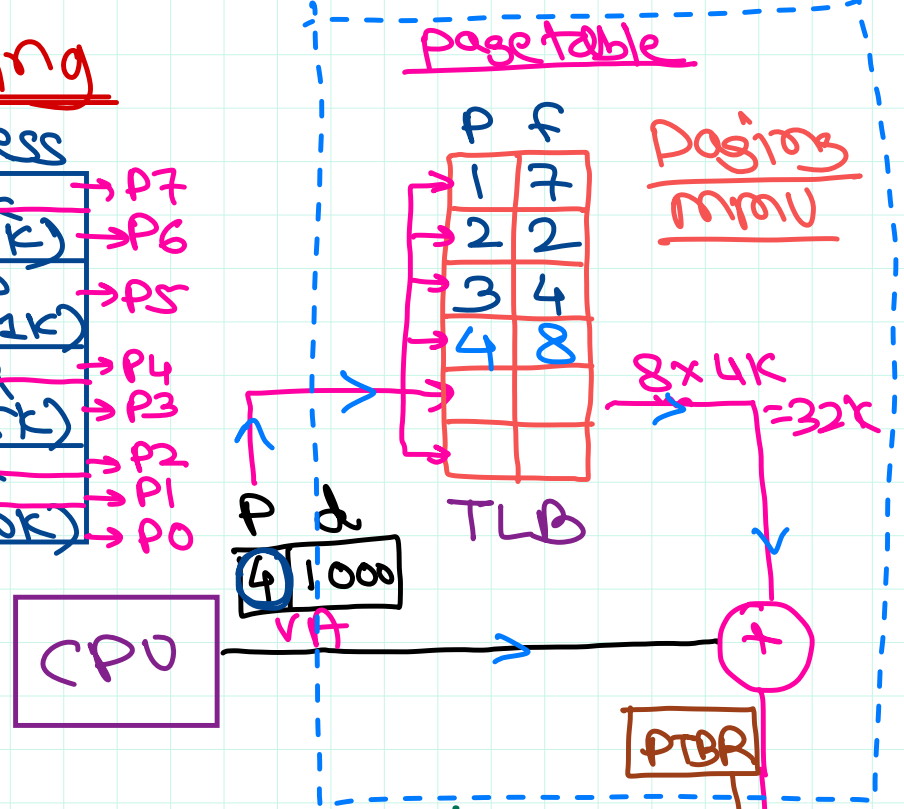


Page Table

e.g. process 40 MB
page size = 4 KB
then
num of pages = $\frac{40M}{4K}$

$= \frac{40 \times 1024 \times 1024}{4 \times 1024}$

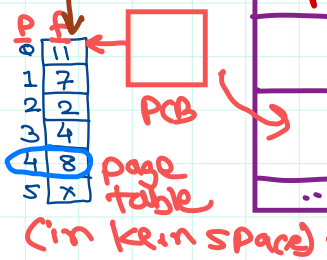
$= 10240$



During Context Switch

option 1: clear TLB.
- for new process initial page req will be TLB miss & get addr from in RAM page table - inefficient.

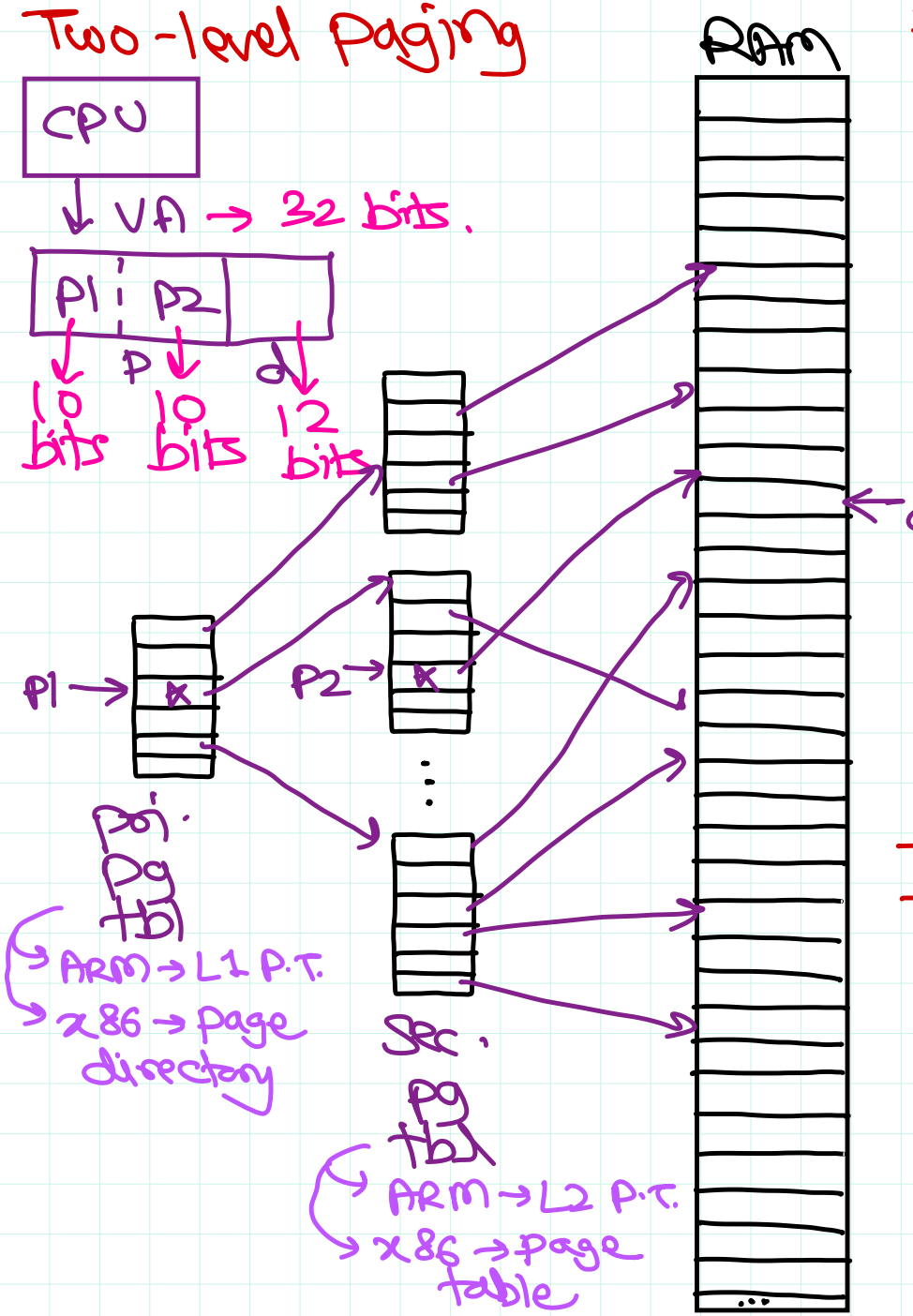
option2: keep pg addr with pid (to avoid pg addr conflict betn multiple processes).
pid + pg addr = asid (addr space id).



Paging MMU = TLB
- High Speed Associative → Key - value (page addr - frame addr)
- all entries compared at once to find frame addr
- num of entries are limited (e.g. 32 or 64).

If entry is found (TLB hit), it will access frame addr imm.
If page addr entry is not found (TLB miss), it will get actual entry from page table (using PTBR) - copy it into page table - calculate physical addr.
If cache is full, least recently used entry (LRU) is over-written with latest addr translation. Always keeps recently accessed entries.

Two-level paging



primary page table = 1 page (4KB).
num of max entries = 1024
⇒ max num of sec page tables = 1024

Secondary page table → per page (4KB)
num of max entries = 1024
⇒ max num of process pages = 1024

size of one process page = 4KB

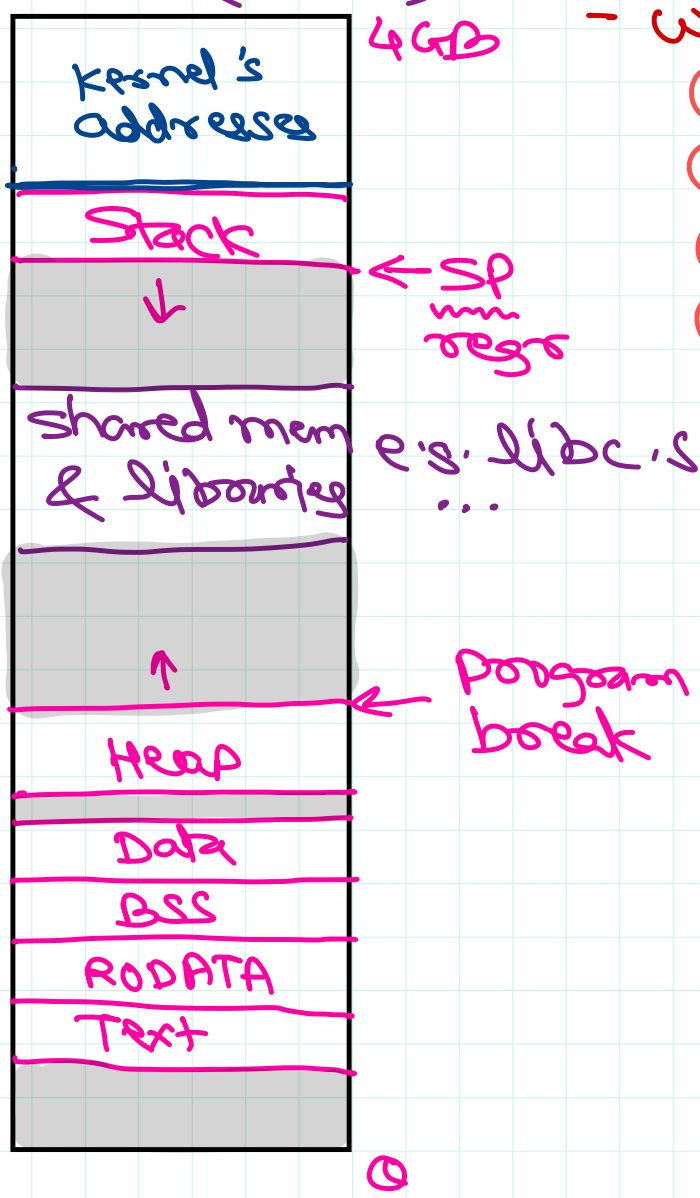
Max process size
= max num of sec table page * num of page for sec P.T. * size of page.
= 1024 × 1024 × 4 × 1024 = 4 GB

Three-level paging

L1 P.T. → L2 P.T. → L3 P.T. → process pages.

Page Table entry → Invalid ?

- when page is not present in RAM
  - (a) may be invalid page (dangling ptr).
  - (b) may not be allocated yet.
  - (c) may be in swap area
  - (d) may be in disk (in exe file)



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>