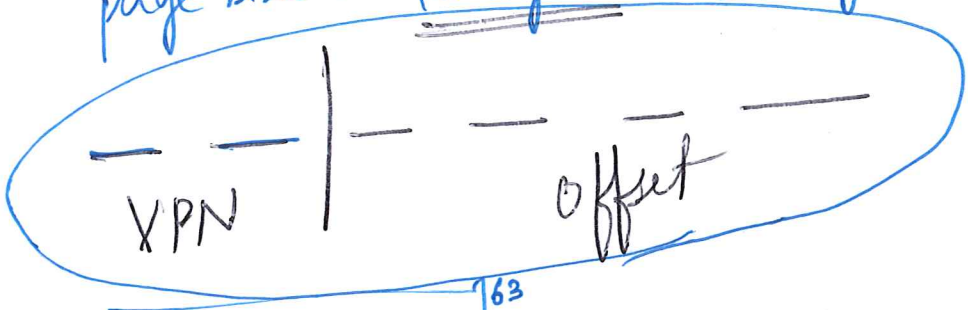
Lecture - 5

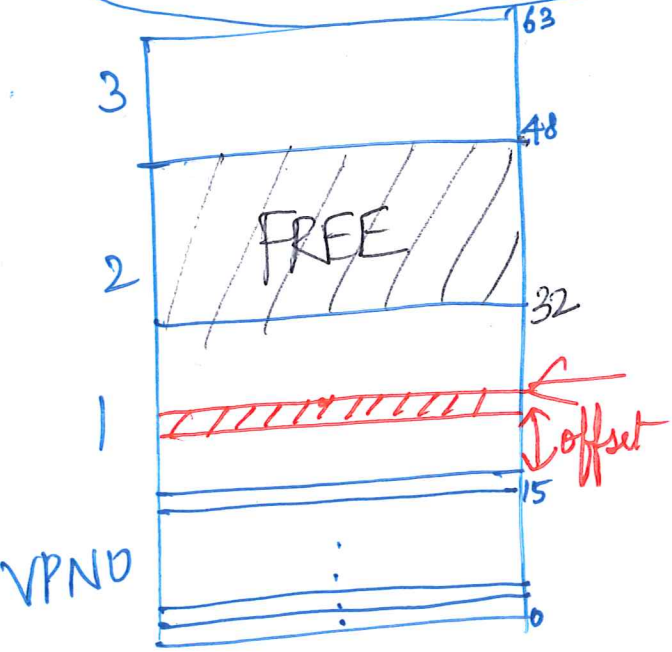
1. Paging.
   - Issues
     - Speed
     - Space.
2. TLBs.
3. _____

Paging

V.A.S. = 64 bytes = 2^6 bytes

page size = 16 bytes = 2^4 bytes

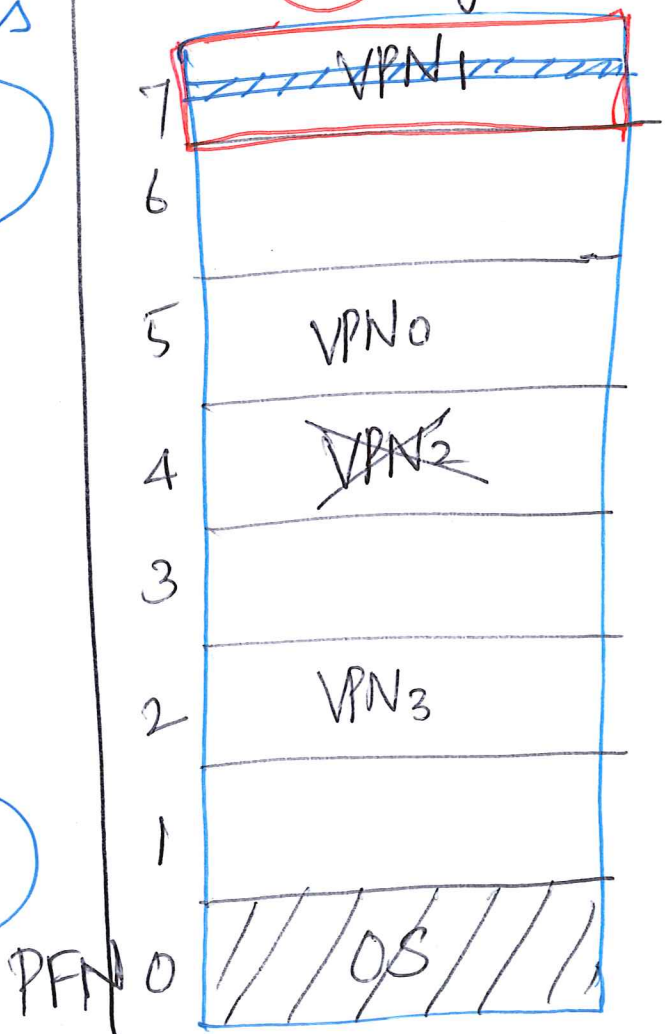




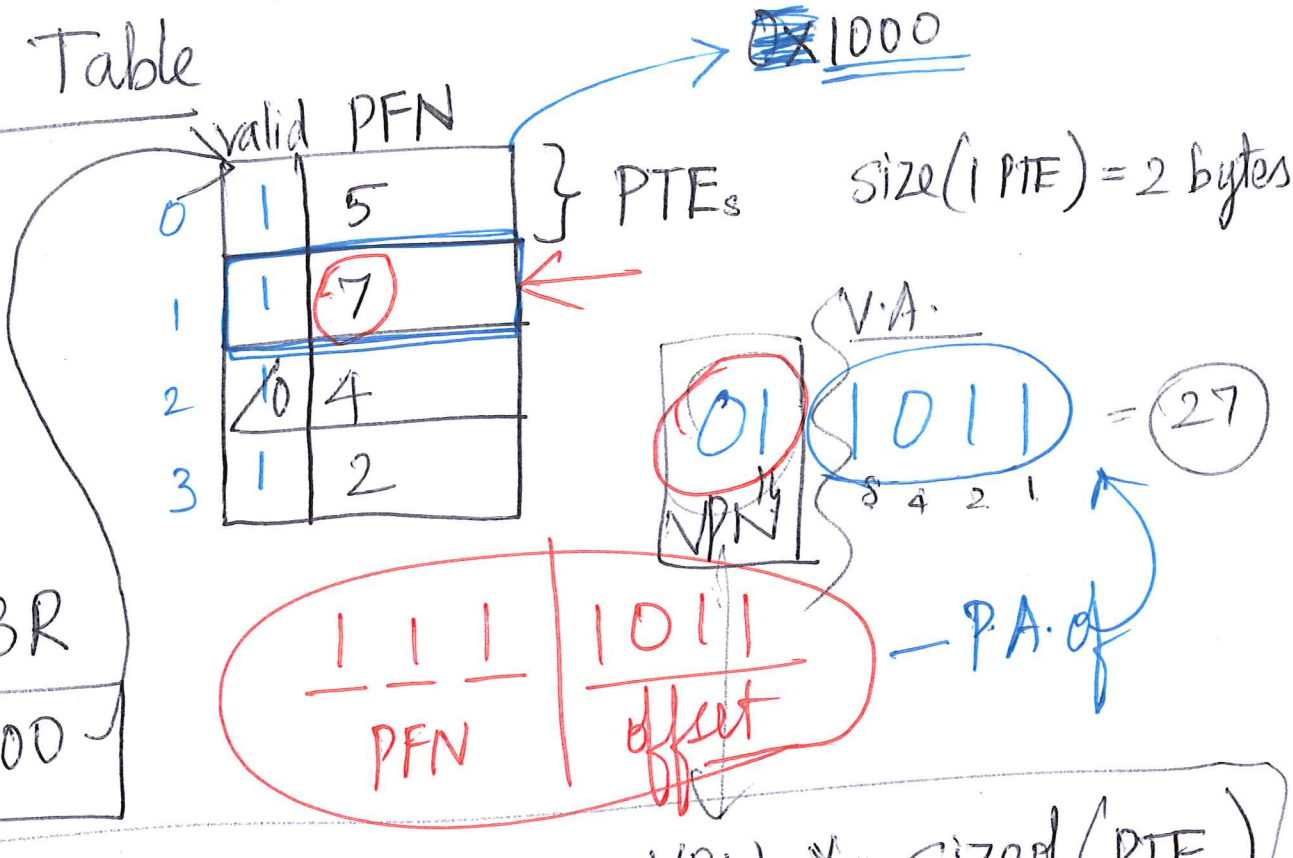
PPN

Physical memory

128 bytes



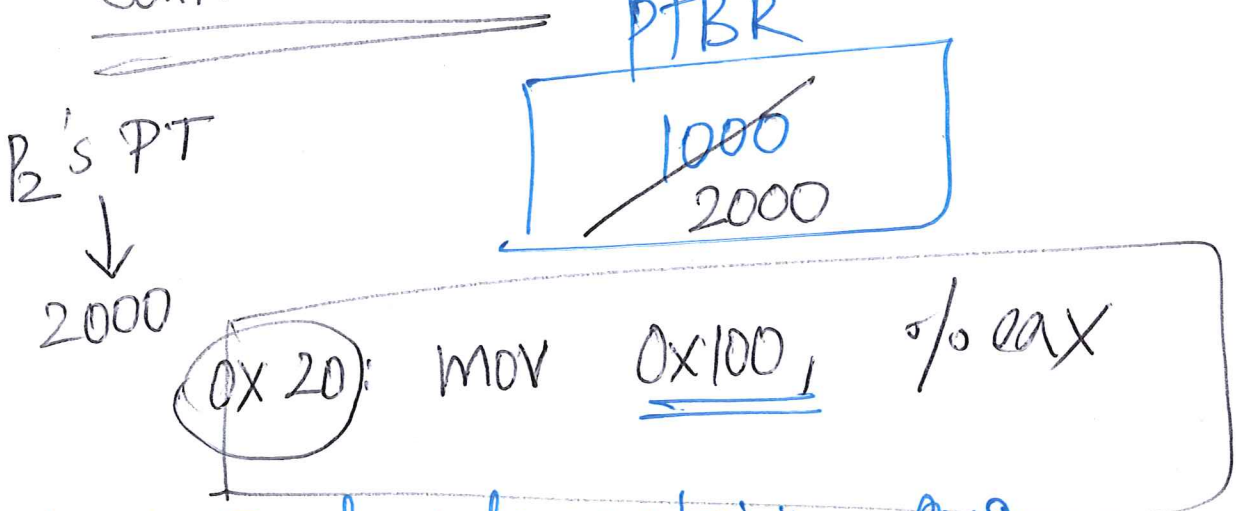
Page Table



PTE_{Addr} = PTBR + VPN * sizeof(PTE)

e.g., 1002 — 1000 — 1 — 2

Context Switch



1. PTE of code containing 0x20.
2. PA of 0x20.
3. PTE for data " 0x100
4. PA of 0x100.

Memory Access

V.A.

1. Extract the VPN from V.A.
2. PTE_{Addr} = PTBR + VPN * size of (PTE)
3. PTE ← MemAccess (PTE Addr) EXPENSIVE
4. Extract the PFN from PTE.

V	PFN

5. PA = PFN + offset
6. Reg ← MemAccess (PA) EXPENSIVE

HW

MMU

TLB

Translation Lookaside Buffer

valid	VPNs	PFNs
1	0	5
0	1	7
1	3	2

Cache Address Translation Cache

16, 32, 64, ... 512.

Mem

Memory Access (w/ TLBs) HW managed TLB

Given: V.A.

1. Extract the (VPN) from V.A.
2. check TLB: if VPN is present.
   a) "TLB Hit"

   PA = PFN + offset

   Access Memory (PA).

3. b) if VPN is NOT present.
   "TLB Miss"

   PTE_{Addr} = PTBR + VPN * sizeof(PTE)

   - Get the PTE and update the TLB with the translation

     VPN → PFN

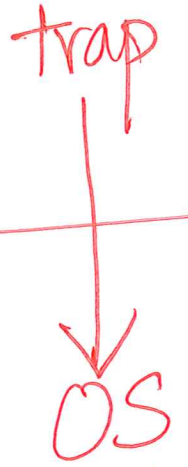
   - retry the instruction.

Mem. Access in a SW (OS) managed TLB

1. same

2. same

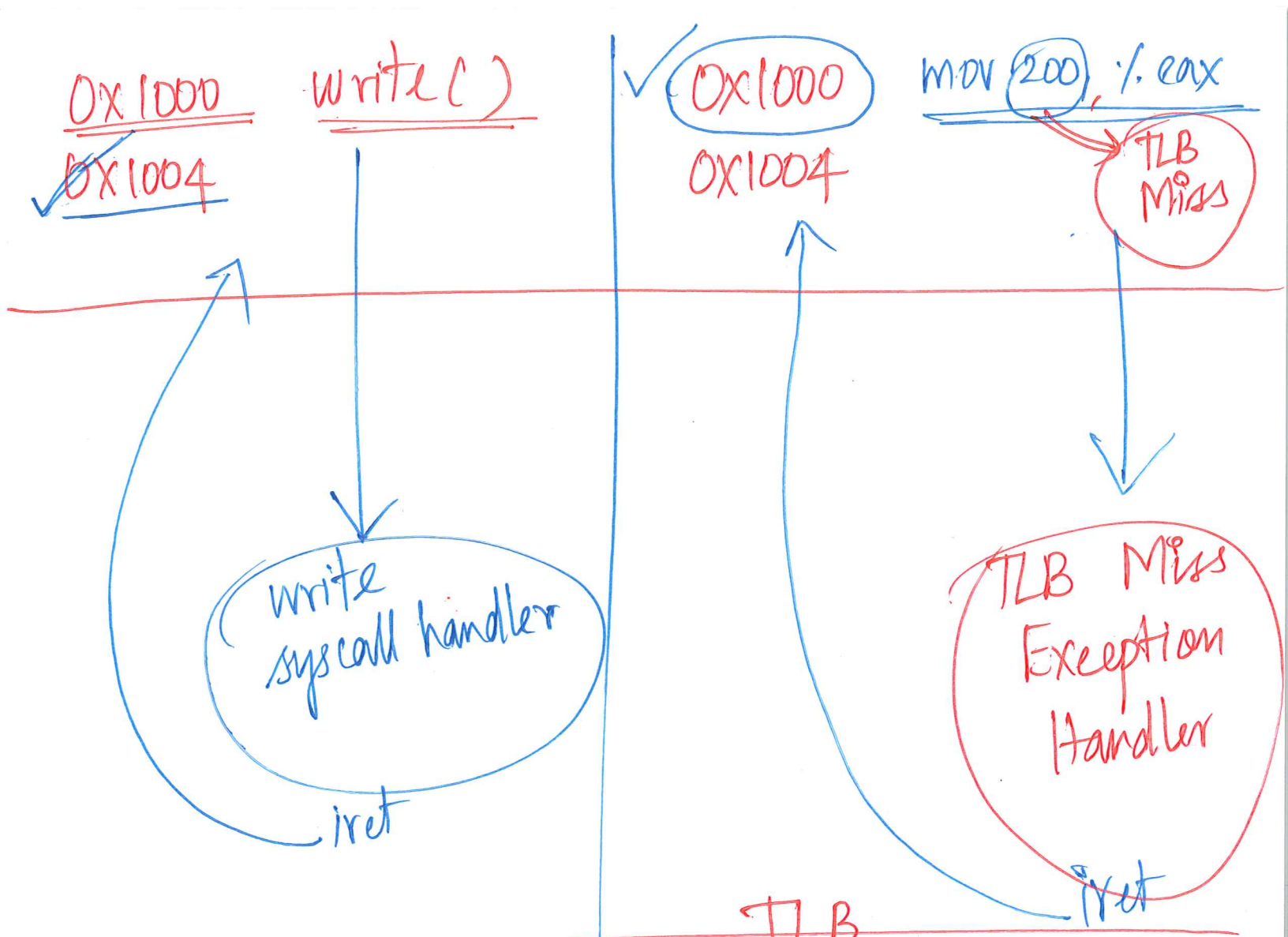
3. TLB Miss ⇒ TLB Miss Exception



1. run a TLB miss exception handler.

2. check PT for PTE.

3. if valid, prot are OK:
   - extract the PFN from the PTE
   - update the TLB with this translation
     instructions → preledged instr.
   - return from trap (iret)



TLB

V	VPN	PFN
~~1~~ 0	3	7
~~1~~ 0	1	2
~~1~~ 0	2	1
~~1~~ 0	0	0

V	VPN	PFN
1	2	7
1	3	8
0	—	—
0	—	—

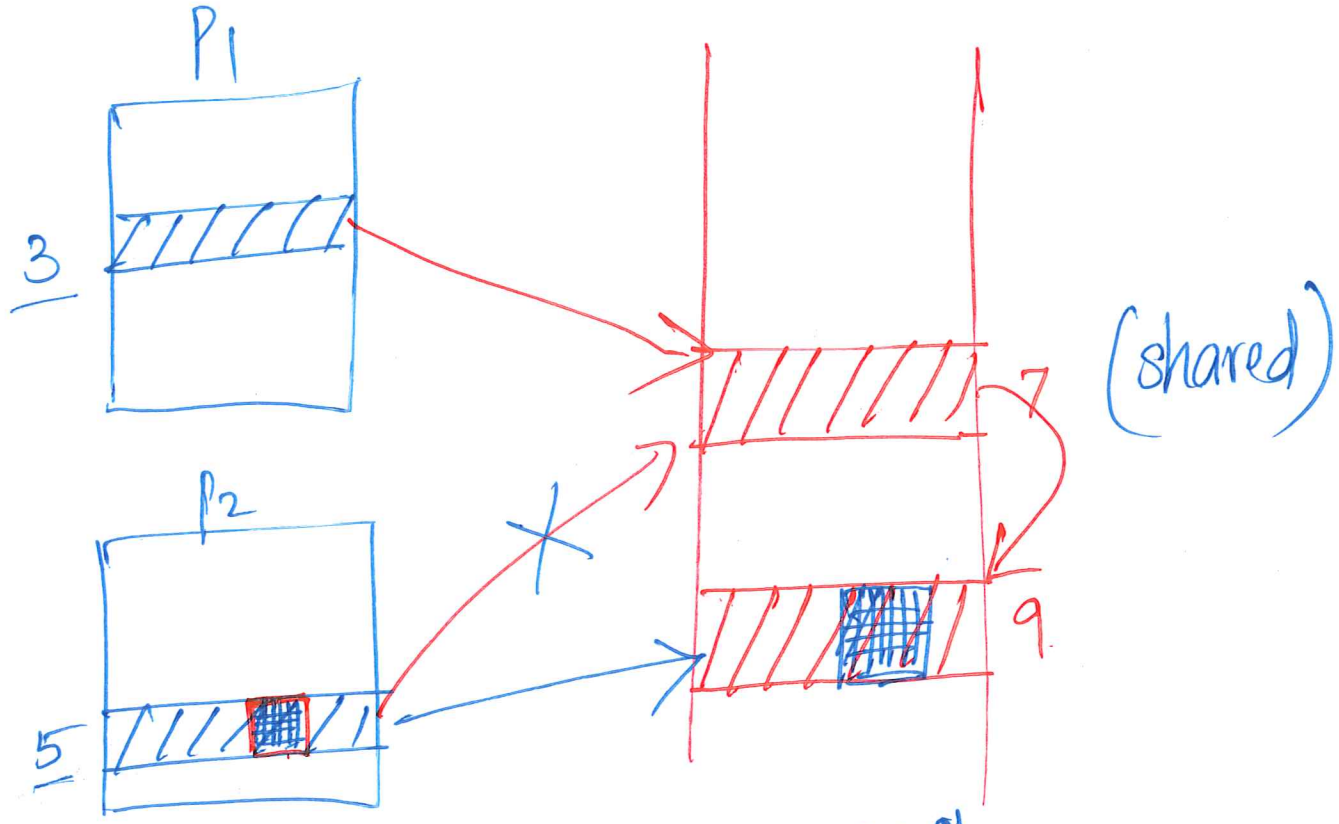
Context Switch

$P_1 \longrightarrow P_2$

1. Flush
   set valid bits = 0.
2. ASID
   (like to pid)

TLB

V	VPN	PFN	ASID
1	3	7	1
1	5	8	1
1	3	9	2
1	5	7	2



COW - Copy On Write

Space

V.A.S. = ~~32 GB~~ 4GB

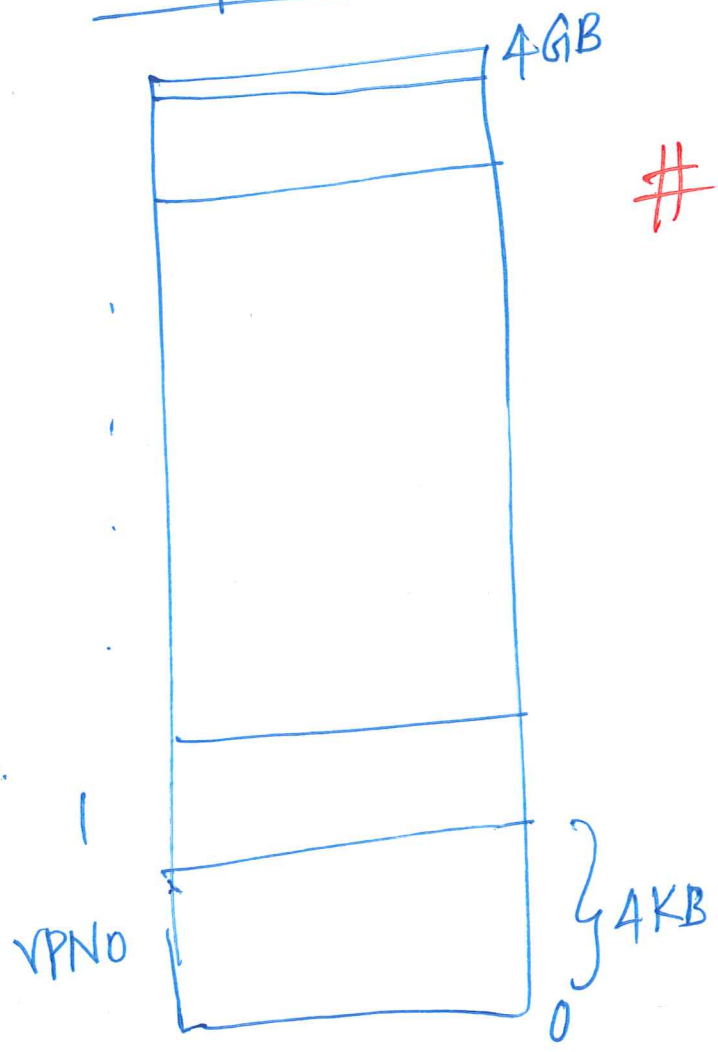
32-bit machine

Page Size = 4KB

2^{32} bytes = $2^2 \times 2^{30}$ bytes

= 4 GB

Simple Linear PT



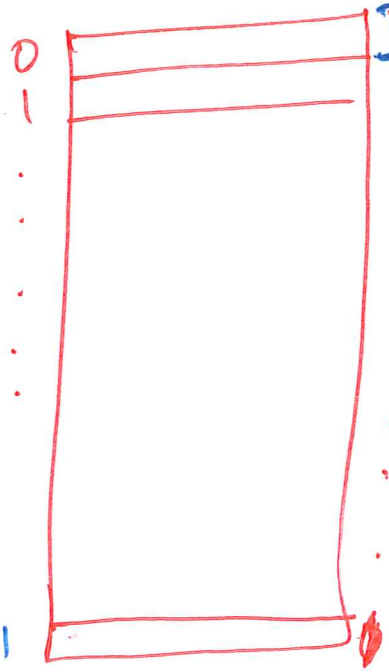
# of VPs = $\frac{4GB\,(2^{30})}{4KB\,(2^{10})}$ = 1MB

= 1024 X 1024 VPs

≃ 1 million VPs.

PT

1 PTE = 4 bytes



1 PT = 4 X 1MB

= 4MB

100 procs.

= 400 MB

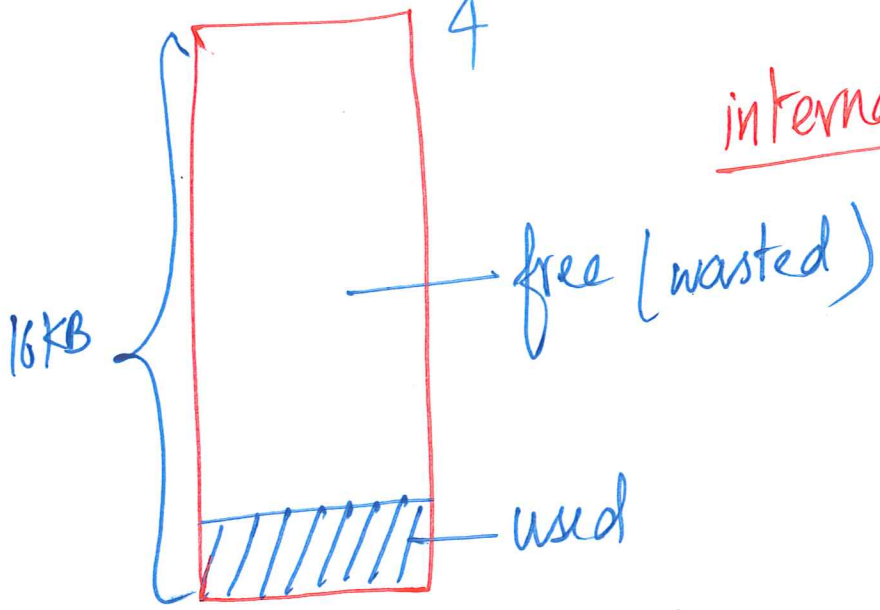
1000 procs

= 4GB

1. Larger Pages

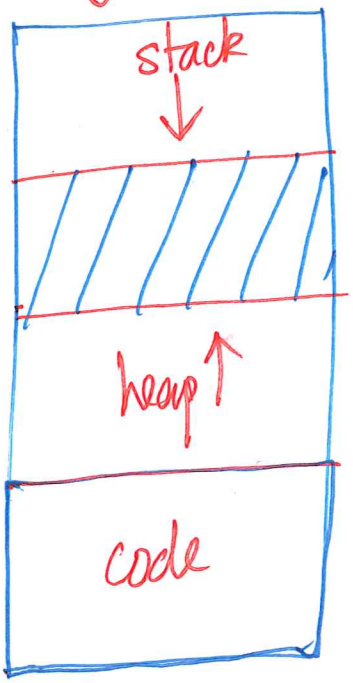
page size = 4KB ⟶ 16 KB

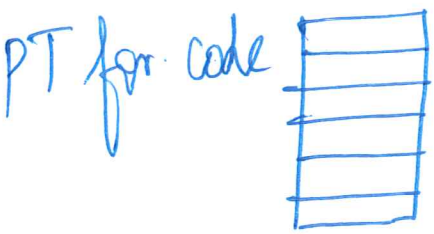
$$\#\ PTEs = \frac{4GB}{16KB} = \frac{2^{20}}{4} = 2^{18} = 256\ KB$$



internal fragmentation

2. Segmentation + Paging

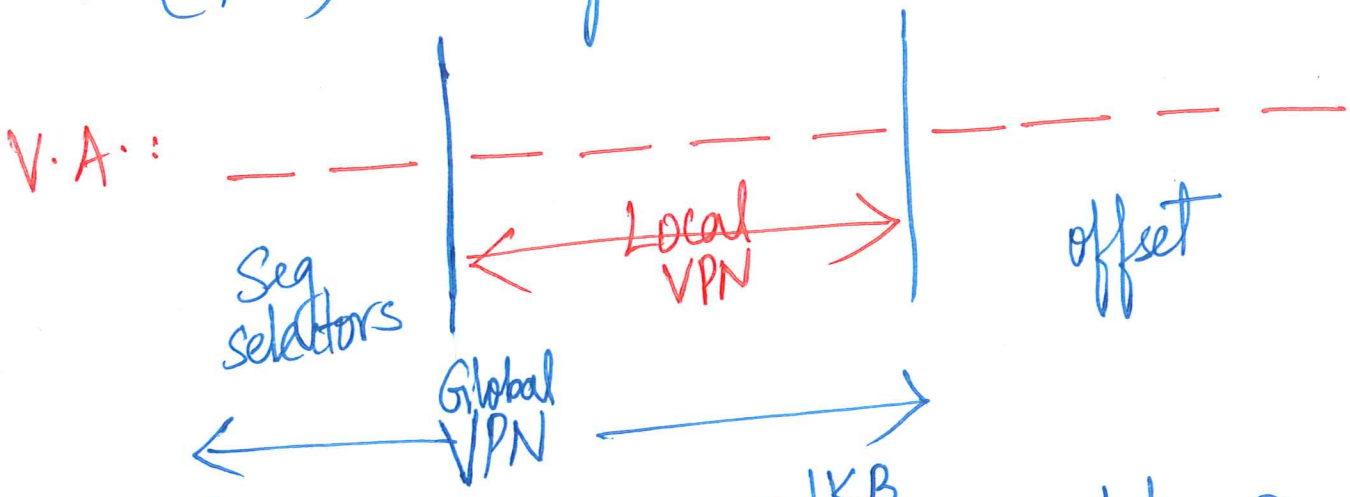


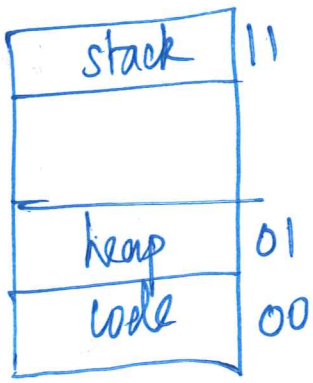


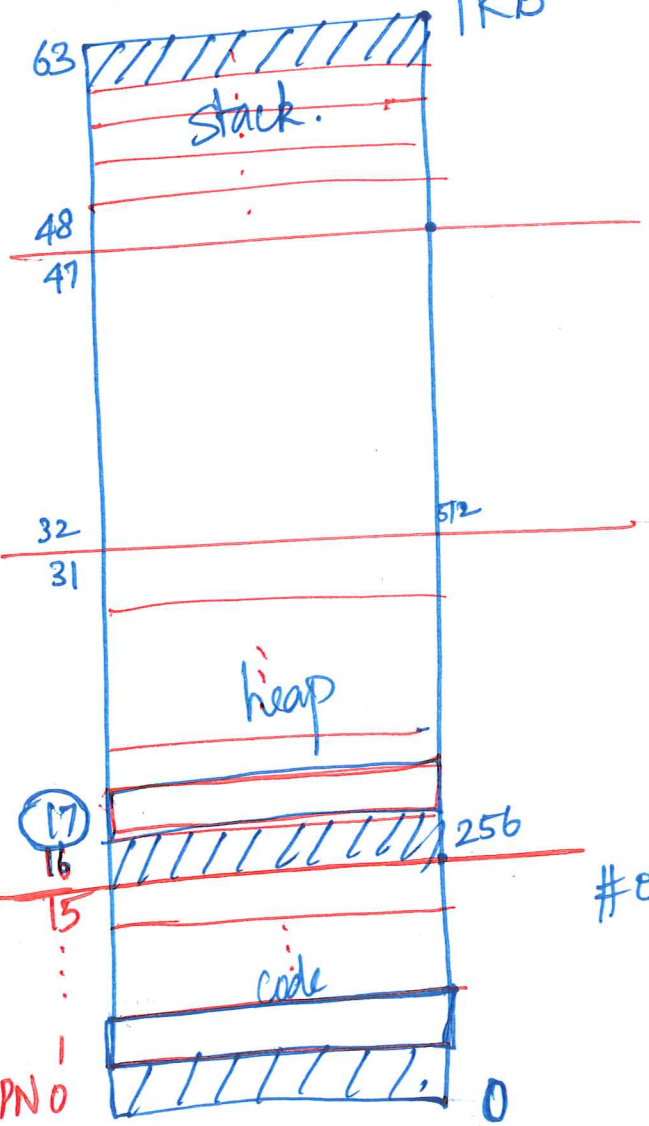
V.A.S = 1KB = 2^{10} bytes

page size = 16 bytes = 2^4 bytes

size(PTE) = 2 bytes







valid VPs = 0, 16, 63

# of V.P.s in code seg. $= \frac{256}{16} = \frac{2^8}{2^4} = 2^4 = 16$

PTBR.

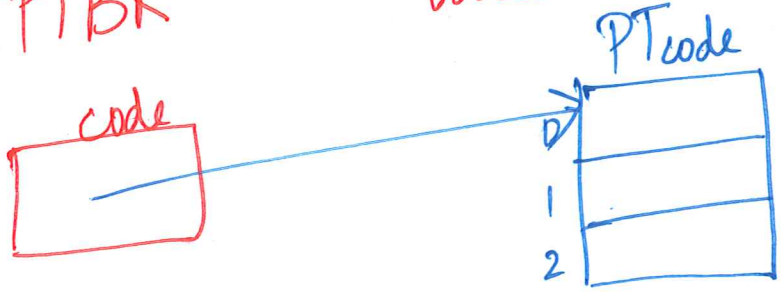
stack

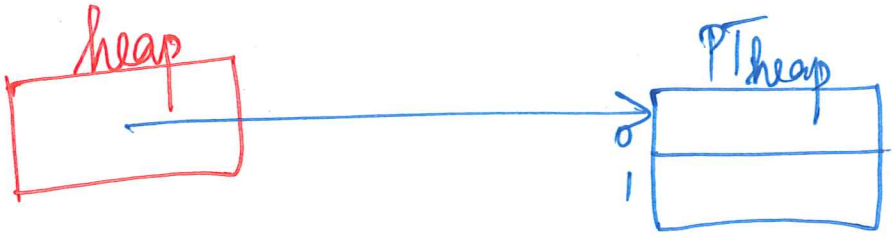
code heap

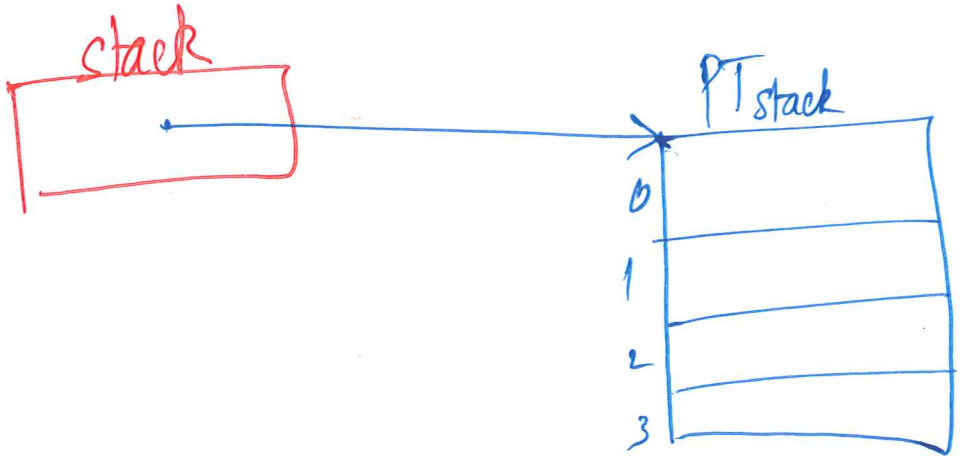
Base	Bounds	Seg
0x1000	✗ 3	code
0x2000	✗ 2	heap
0x3000	✗ 4	stack.

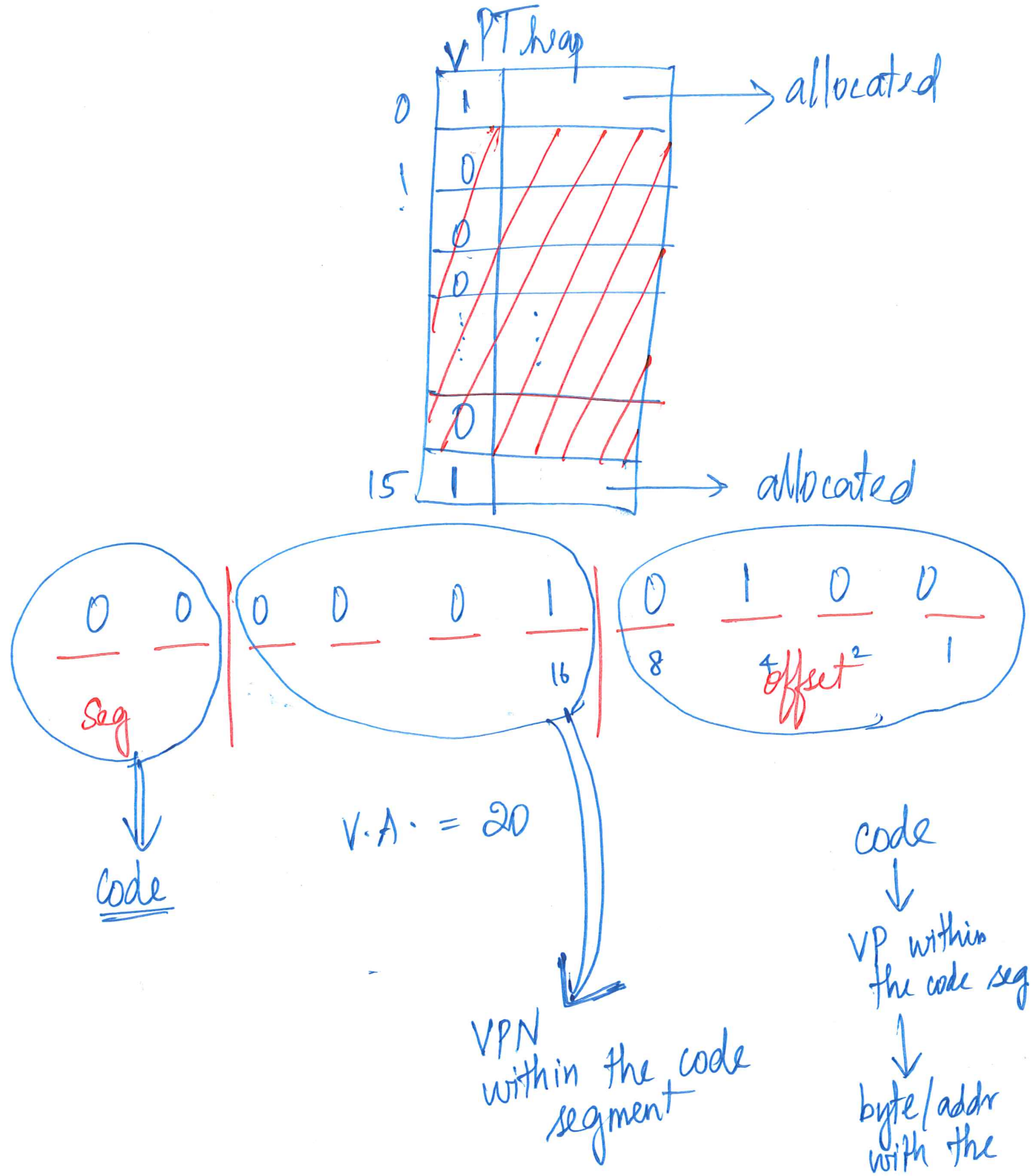
VP.s valid = 0, 1, 2, (code) | 16, 17, (heap) | 63, 62, 61, 60 (stack)

PTBR







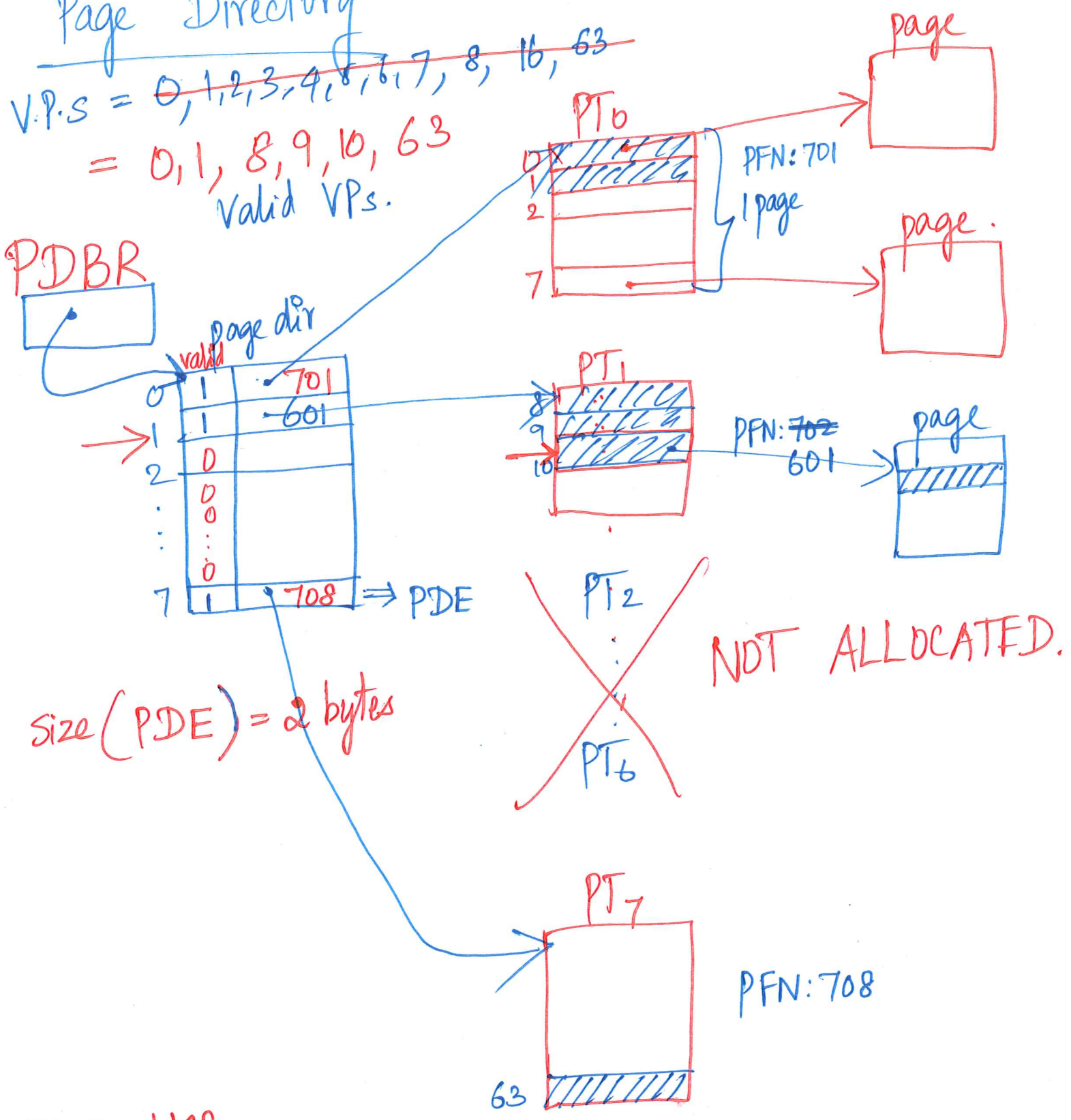

PT heap
V
0
1
15
1
0
0
0
0
1
allocated
allocated
0 0
0 0 0 1
0 1 0 0
16
8
1
offset
Seg
code
V.A. = 20
VPN within the code segment
code
VP within the code seg
byte/addr with the VP.

Page Directory

V.P.s = ~~0, 1, 2, 3, 4, 8, 6, 7, 8, 16, 63~~

= 0, 1, 8, 9, 10, 63

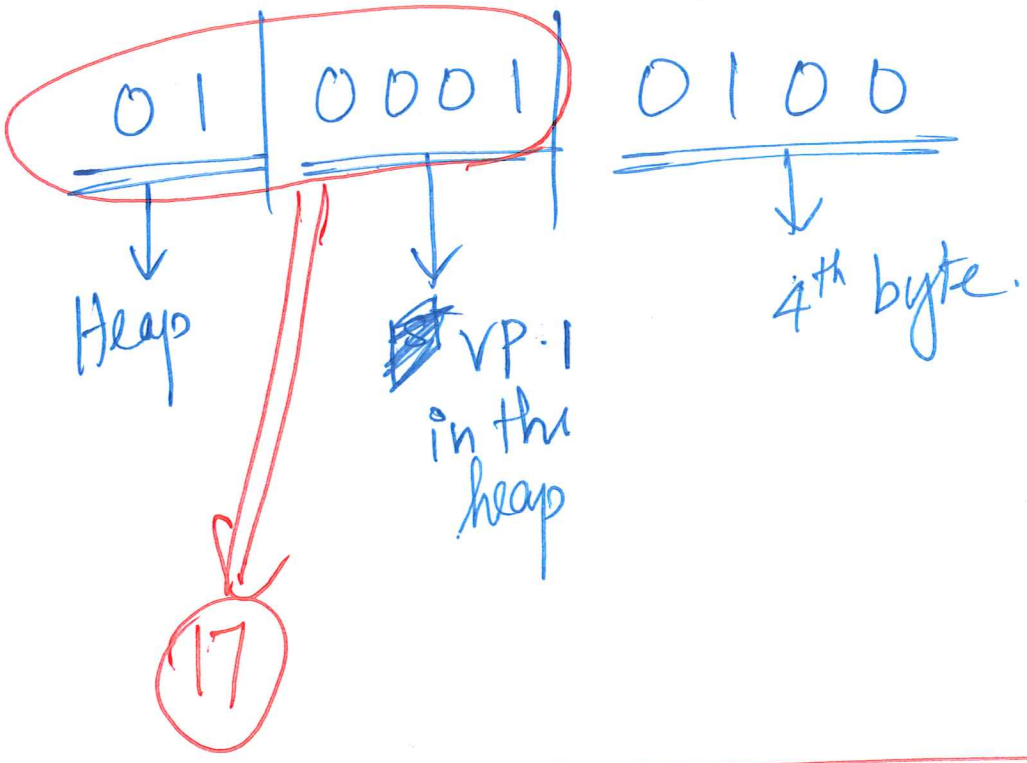
Valid VPs.

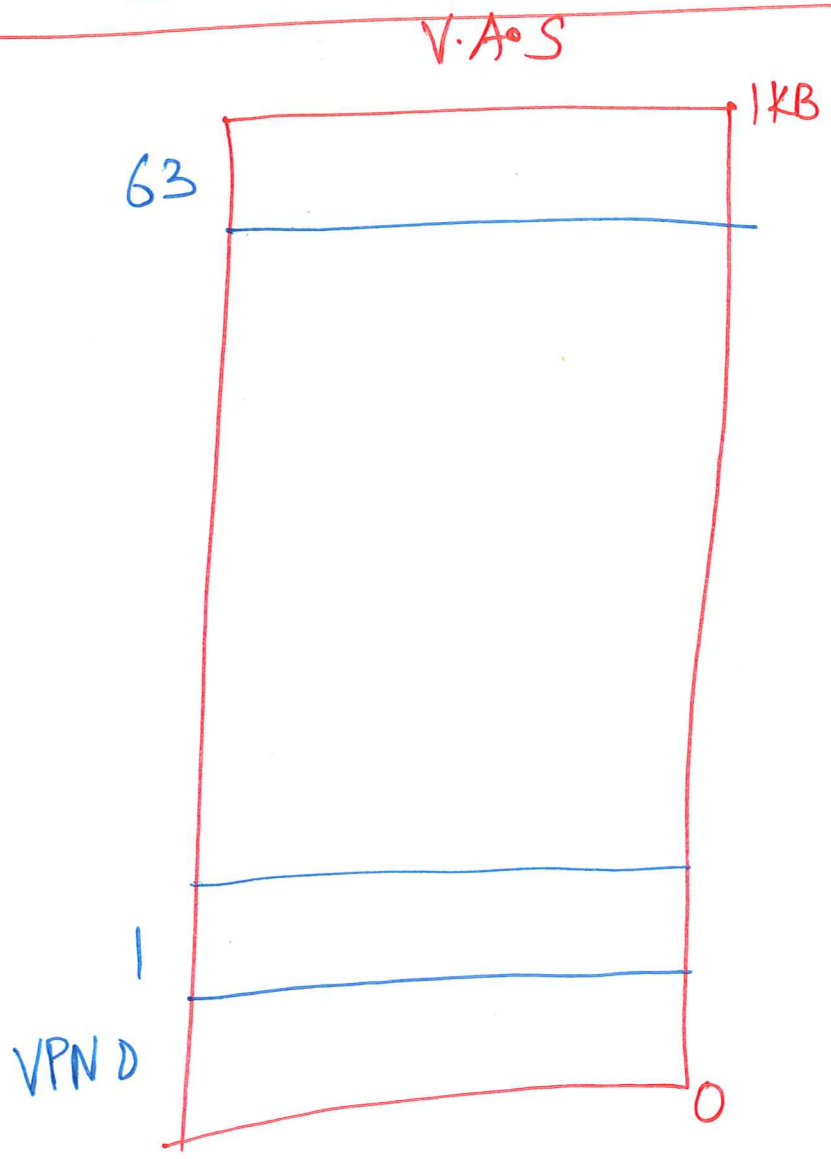


size (PDE) = 2 bytes

VAS: 1KB

0	0	1	0	1	0	0	0	1	0
PDI			PTI			offset			





V.A.S = 1KB

page size = 16 bytes

size(PTE) = 2 bytes

Phy. addr = 4KB.

$$\#\ VPs = \frac{1KB}{16\ bytes}$$

$$= \frac{2^{10}}{2^4} = 2^6$$

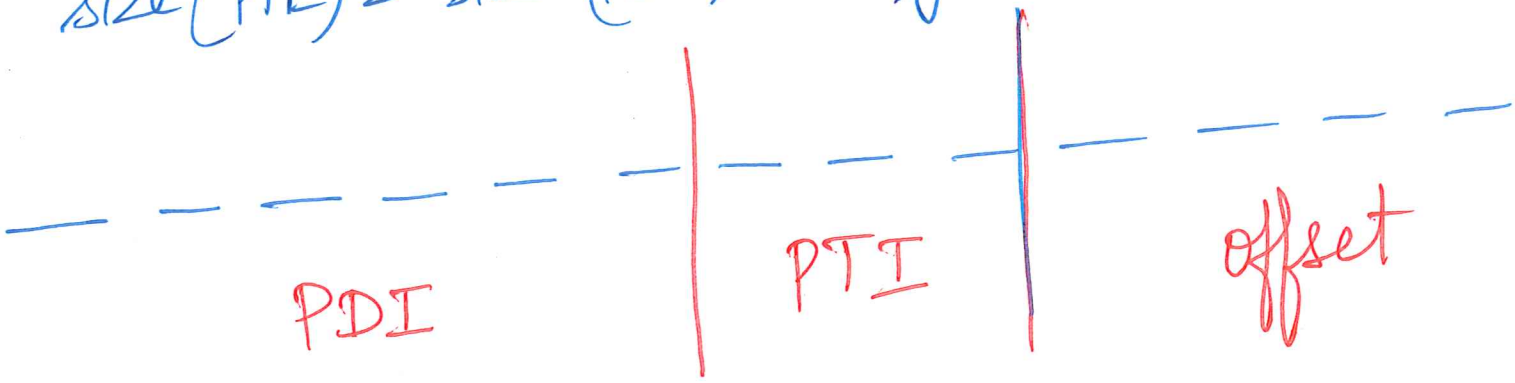
$$= 64$$

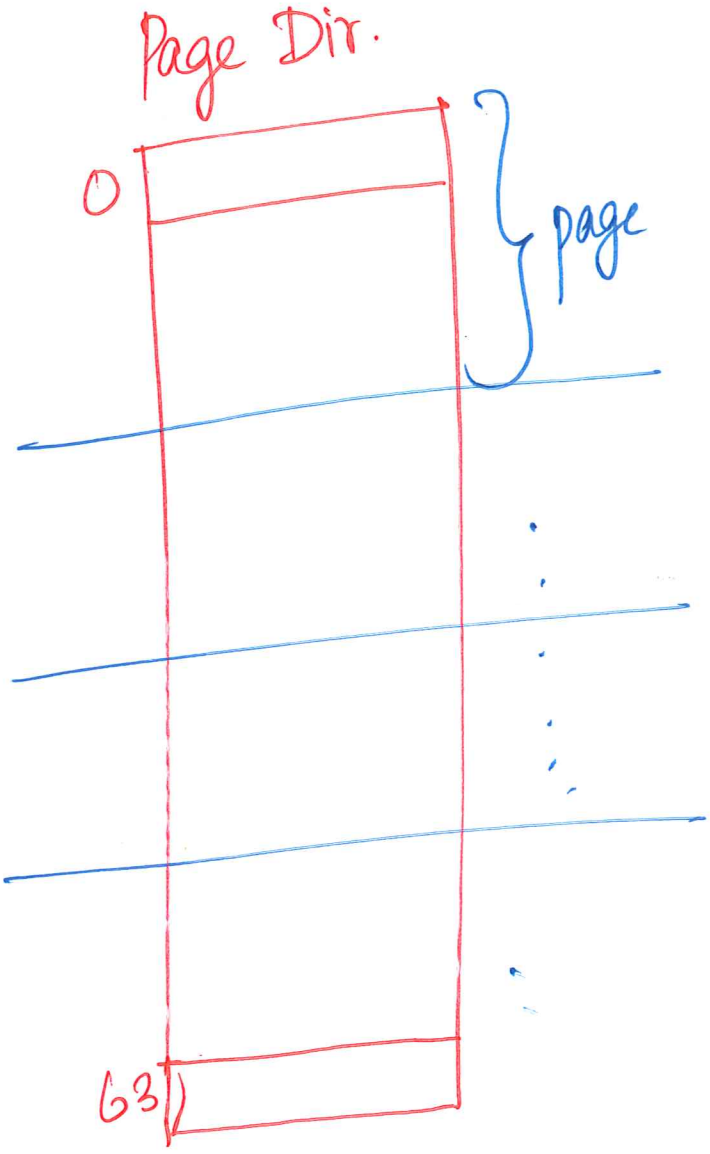
V.A.S = ~~1KB~~ 8KB = 2^{13} bytes

P.A.S = 4KB

page size = 16 bytes

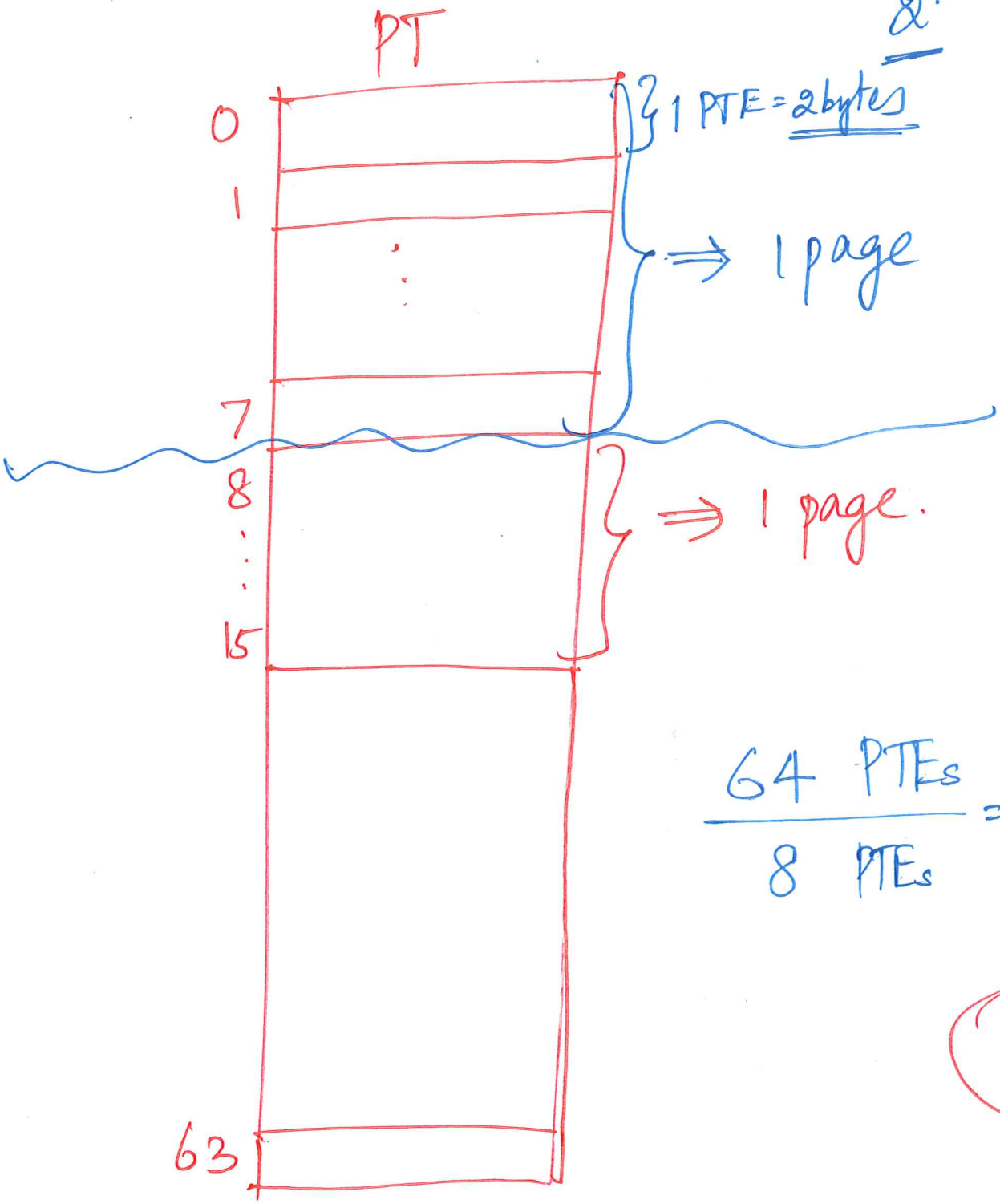
size(PTE) = size (PDE) = 2 bytes.





64 PTEs × $\frac{16}{2}$ = $2^6 \times \frac{2^4}{2} = 2^{10}$ bytes

2^7



$\frac{64 \text{ PTEs}}{8 \text{ PTEs}} = 8 \text{ pages}$

8 PTs