Last class

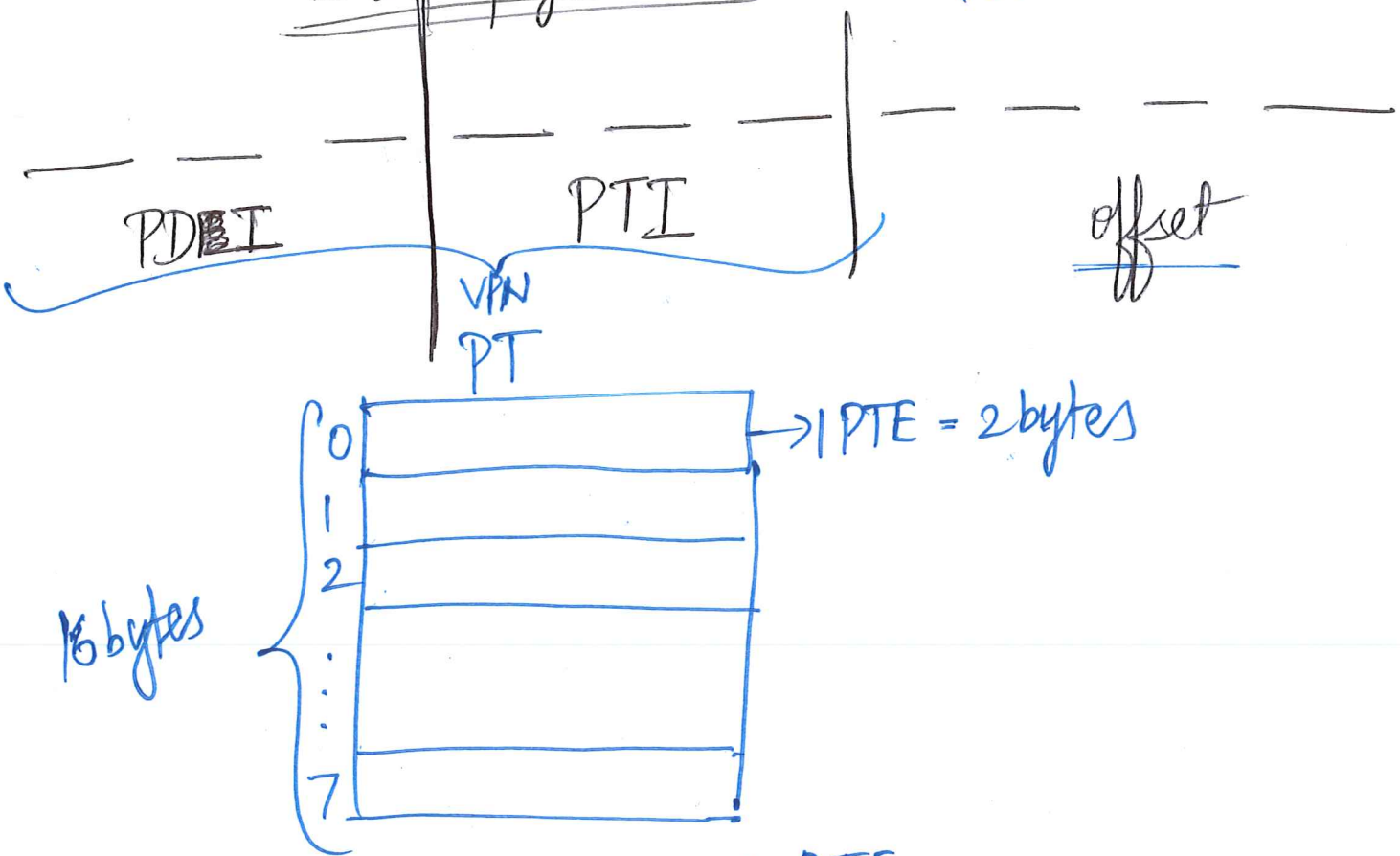
1. Paging < space → multi-level page tables
time → TLB

Virt. addr space = 1KB = 2^{10} page size = 16 bytes = 2^4

phy addr " = 4KB = 2^{12} sizeof(PTE) = 2 bytes

" PDE = "

2 level page table



PT

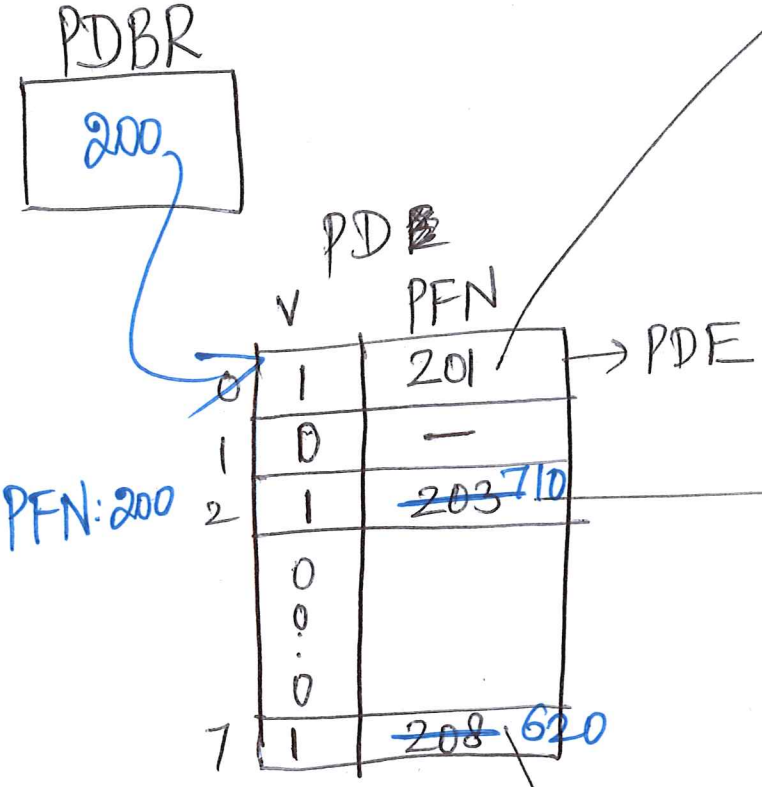
1 page = 8 PTEs

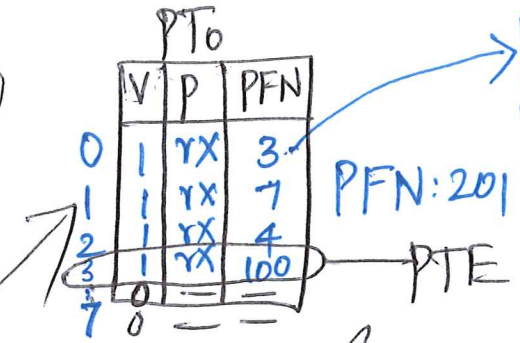
PTEs = 2^6 = 64

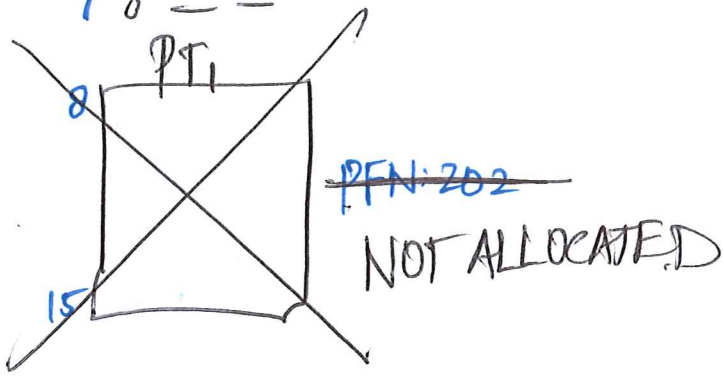
pages for storing the Page Table = $\frac{64}{8}$ = 8 pages

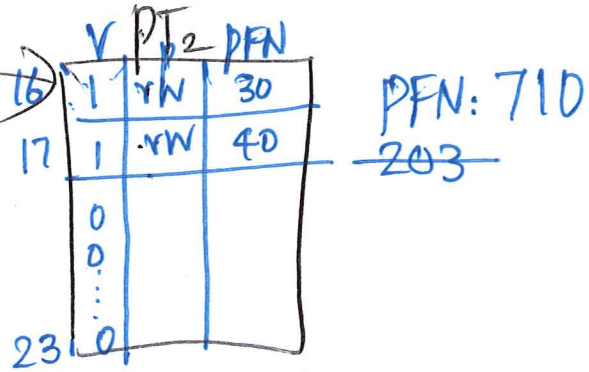
Assumption:

VPNs (0, 1, 2, 3, 16, 17, 62, 63.) are valid.



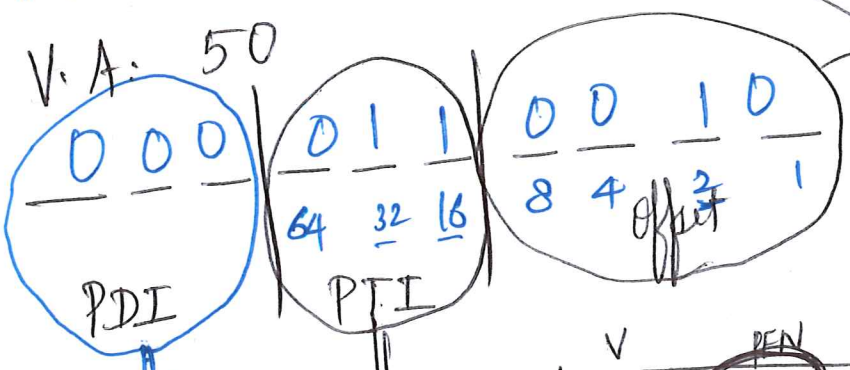


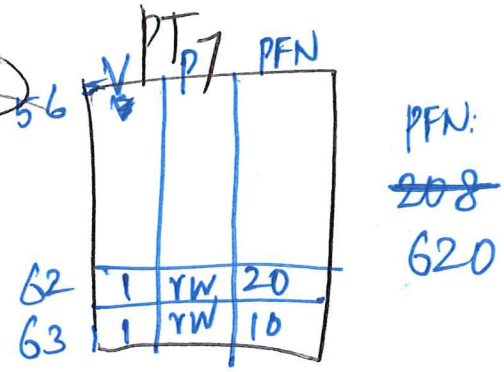




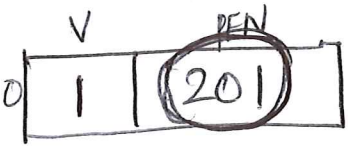
8 + 3 + 1 = 12 pages

pages PTs PD

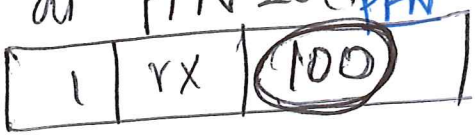




0th entry in the PD



3rd PTE in PT0 which is at PFN 201.



PA = PFN + offset

100

0x64 → 4bits 4bits

0x002

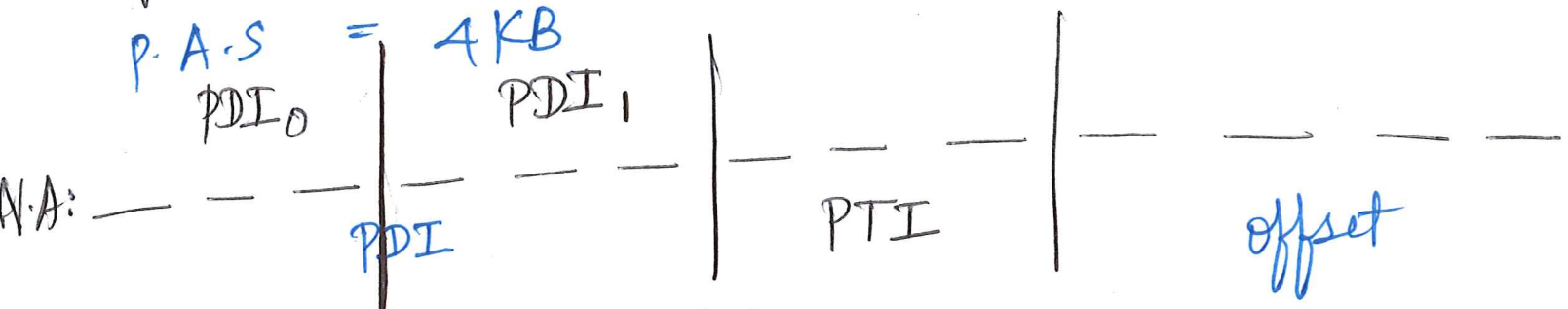
PA = 0x642

16 | 100
6 – 4

0x64

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

P.A.S = 4 KB

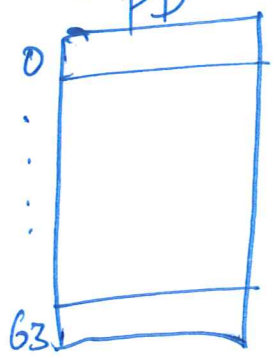


page size = 16 bytes.

1 PTE = 1 PDE = 2 bytes

$2^6 = 64$

PD



PDEs in the PD = $\frac{64}{2}$ = 32 PDEs

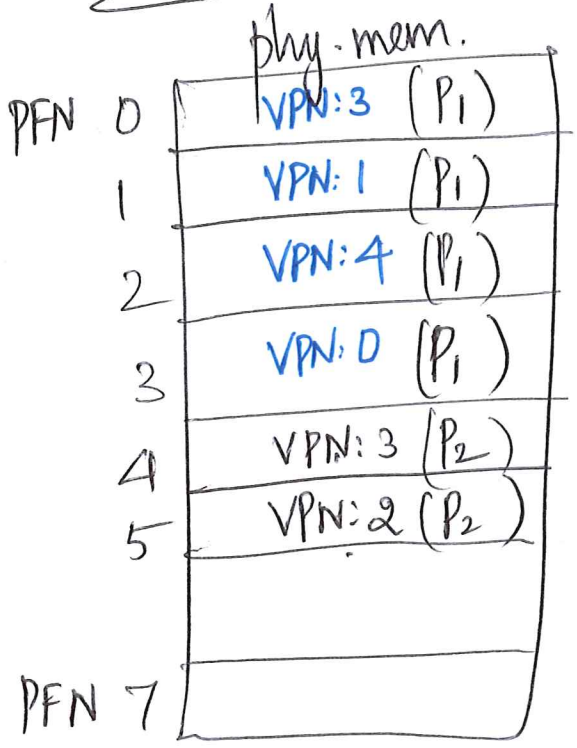
4 pages

64 PDEs X 2 bytes

= 128 bytes

pages for the PD = $\frac{128}{16} = \frac{2^7}{2^4} = 2^3 = 8$ pages

Inverted Page Table



PFN	VPN	V	P	pid
0	3			1
1	1			1
2	4			1
3	0			1
4	3			2
5	2			2
6				
7				

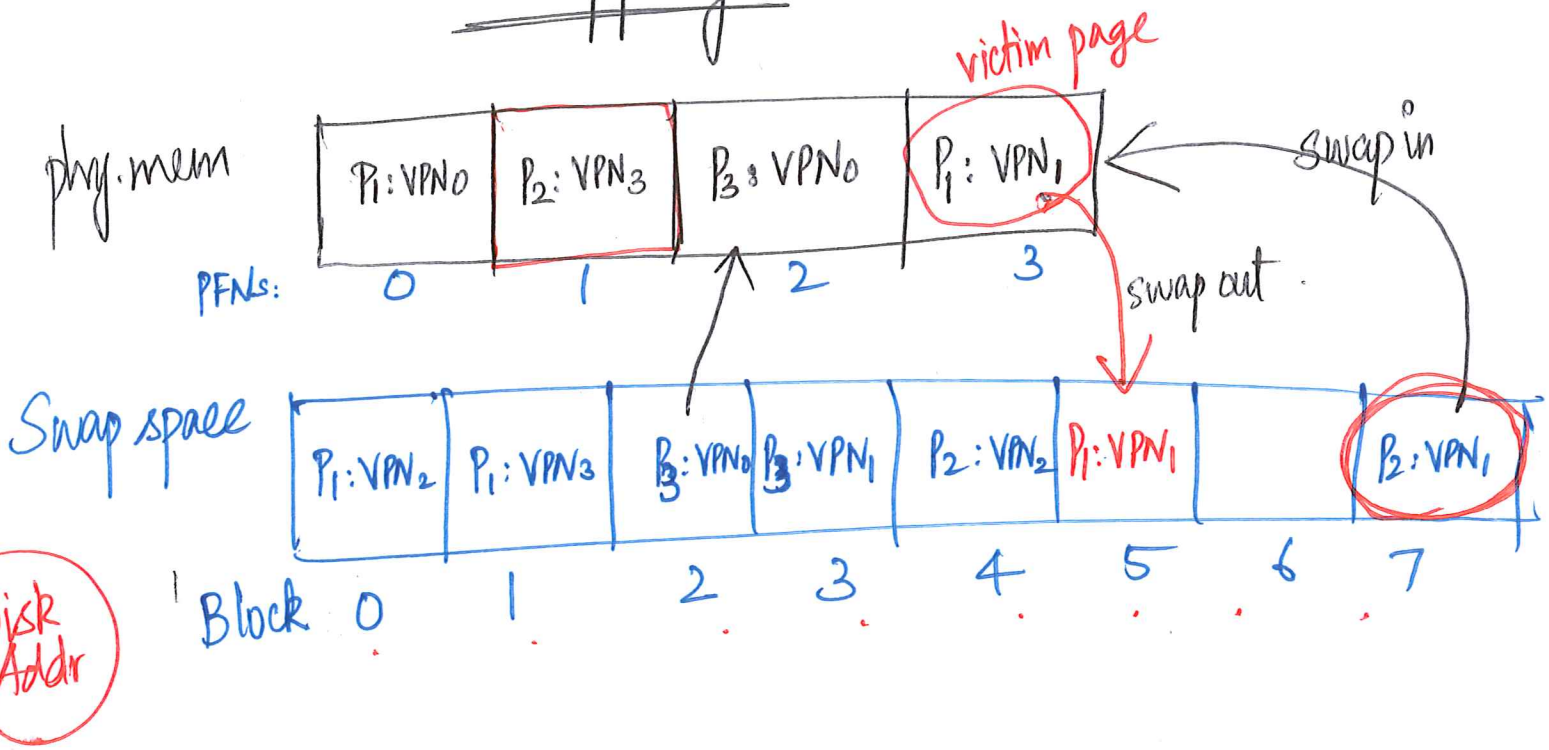
phy. mem = 8GB

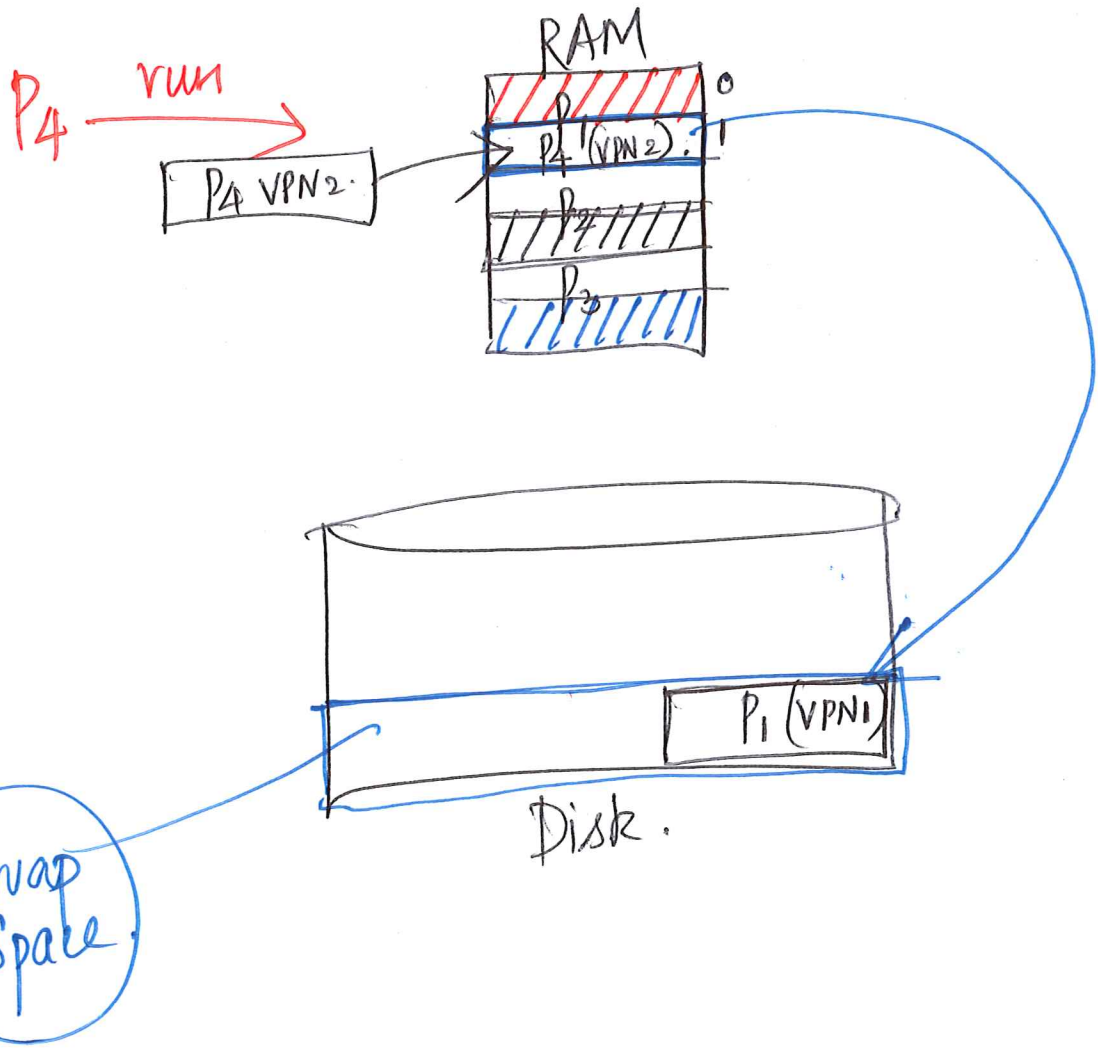
page size = 4KB

$$\#\text{ phy. frames} = \frac{2^{30}}{4 \times 2^{10}} = \frac{8 \times 2^{20}}{4 \times 2^{10}} = 2 \times 2^{20}$$

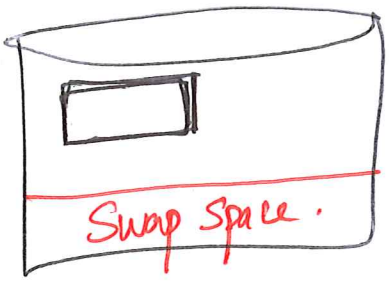
$\simeq$ 2 million.

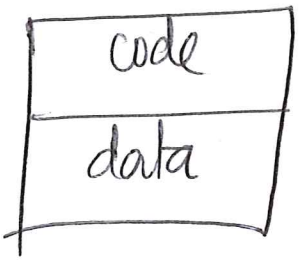
Swapping


victim page
phy. mem
P1: VPN0
P2: VPN3
P3: VPN0
P1: VPN1
swap in
PFNs: 0 1 2 3
swap out
Swap space
P1: VPN2
P1: VPN3
P3: VPN0
P3: VPN1
P2: VPN2
P1: VPN1
P2: VPN1
Disk Addr
Block 0 1 2 3 4 5 6 7

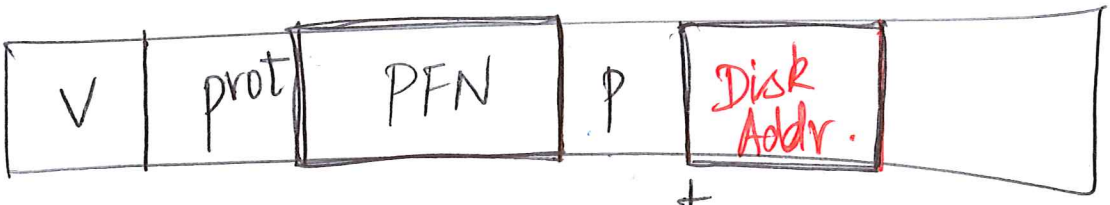

RAM
P4
run
P4 VPN2
P4 (VPN2)
0
1
P2
P3
P1 (VPN1)
Disk.
Swap Space

a.out





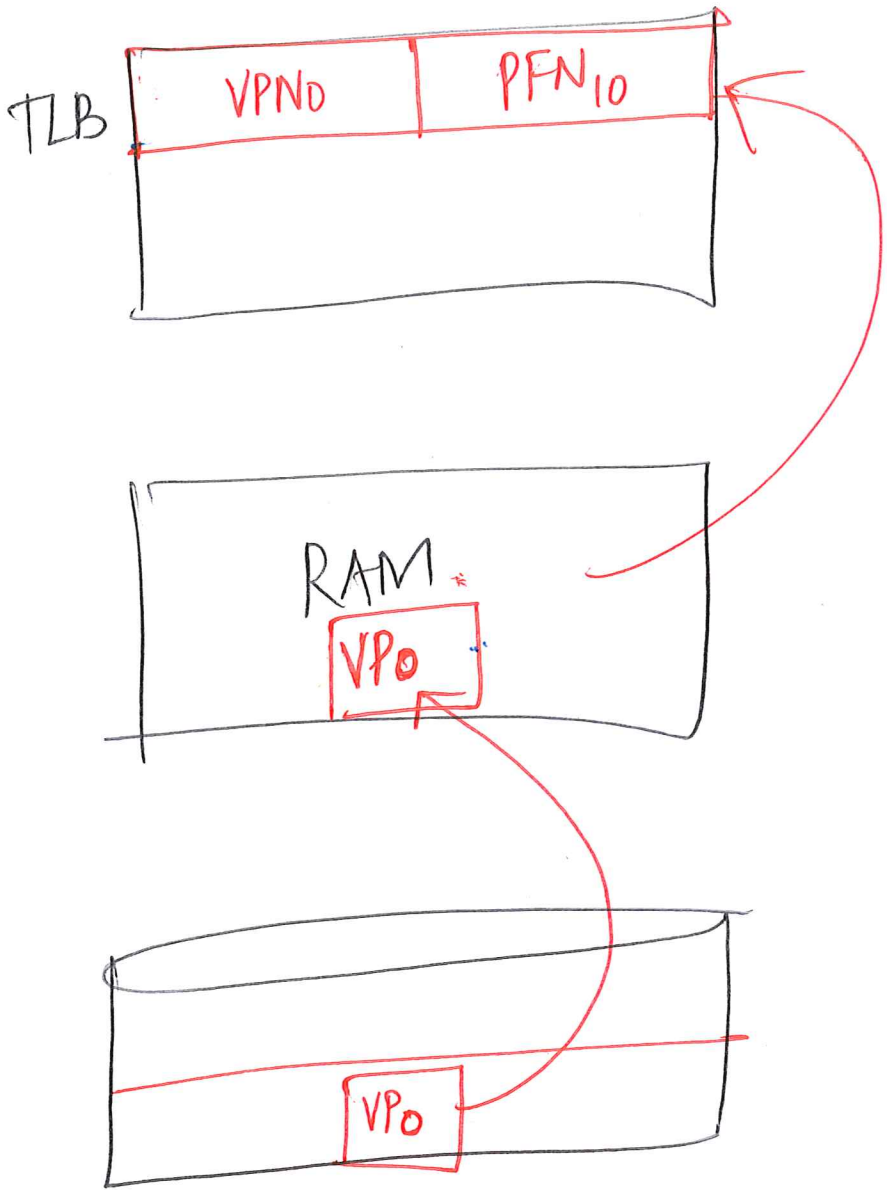
PTE:



P = 1 ⇒ page is in phy. mem.

P = 0 ⇒ " " " swap space.

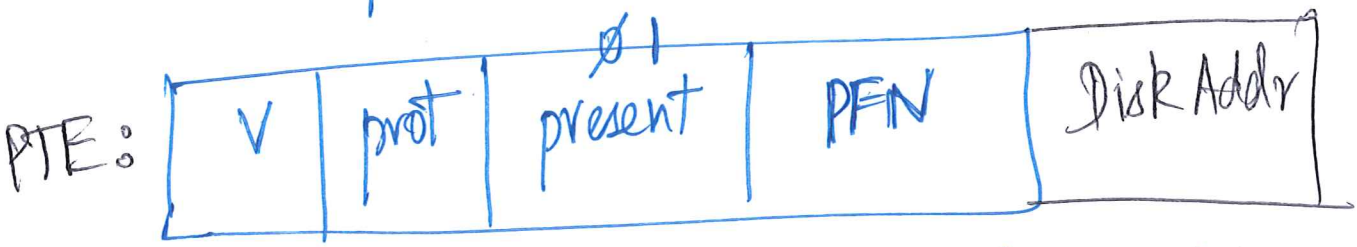
"Page Fault"


TLB
VPN0
PFN10
RAM
VP0
VP0

Present Bit

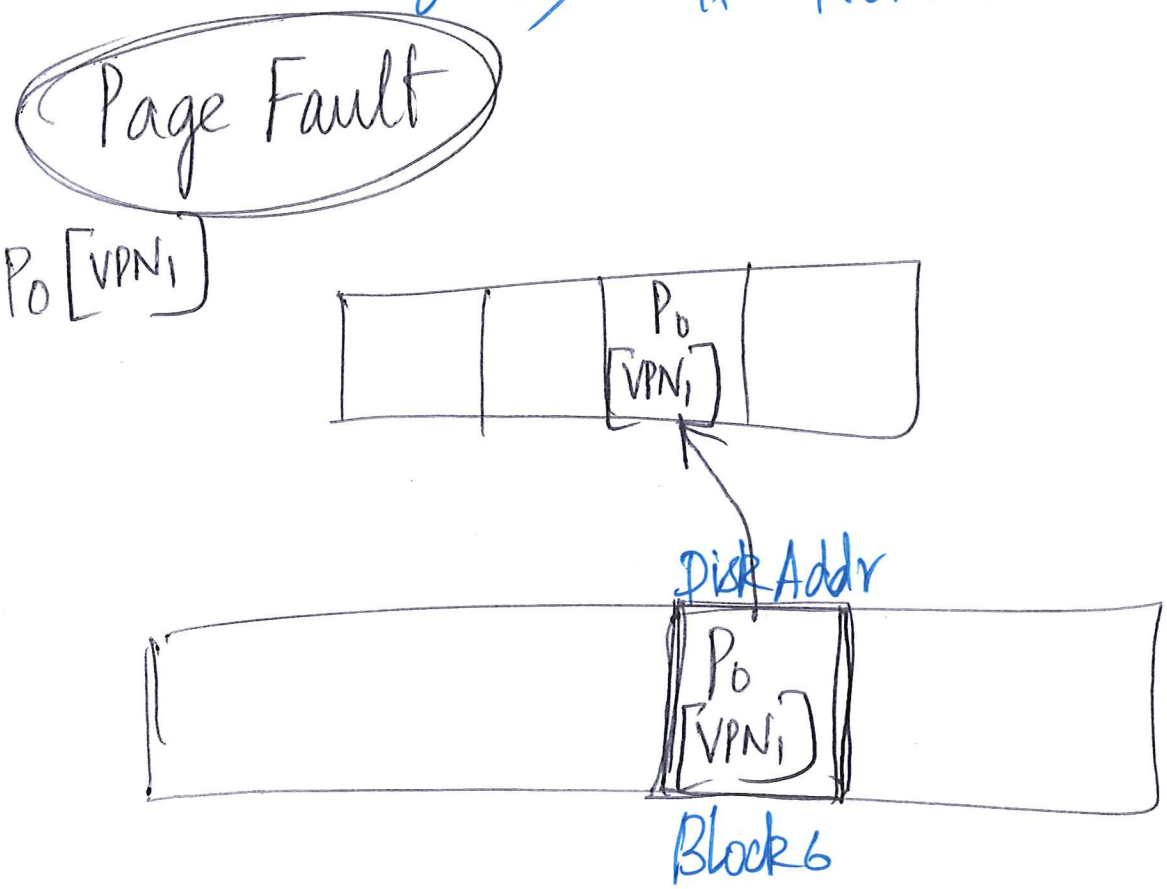
PTE

v prot PFN



present = 1 ⇒ page is in memory

0 ⇒ " NOT in "



V.A: | VPN | offset |

Assumptions:
1. simple linear PT
2. HW managed TLB.

1. Extract VPN from VA
2. Check TLB

TLB hit → PA = PFN + offset → Access Memory (PA)

TLB miss → PTE_{addr} = PTBR + sizeof(PTE) * VPN.

PTE | PFN |

if (present bit == 1)
→ extract the PFN from the PTE.
→ update the TLB with this translation.
→ retry instr.

else : // present bit == 0.
→ raise (Page Fault Exception).

Page Fault Handler

OS:

```
PFN = FindFreePhysicalPage().
if (PFN == -1)
    PFN = EvictPage() // Page replacement policies.
DiskRead(PTE.DiskAddr, PFN)

PTE.PFN = PFN;
PTE.present = 1;
Return from trap
```

HW: retries the instr.

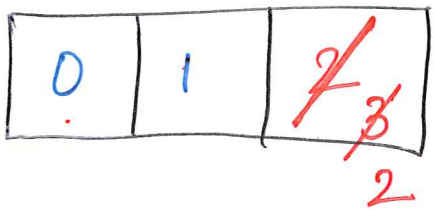
Swapping - Policies

1. OPT

cold miss

0	1	2	0	1	3	0	3	1	2	1
m	m	m	h	h	m	h	h	h	m	h

Cache: eg. RAM.



hit rate = $\frac{6}{11}$

2. FIFO

0	1	2	0	1	3	0	3	1	2	1
m	m	m	h	h	m	m	h	m	m	h

Cache: ~~∅~~ ~~1~~ ~~2~~ ~~3~~ 0 1 2

hit rate = $\frac{4}{11}$

3. Random

0	1	2	0	1	3	0	3	1	2	1
m	m	m	h	h	m	h	h	m	m	h

Cache: ~~∅~~ ~~1~~ ~~2~~
2 3 1

hit rate = $\frac{5}{11}$

4. LRU - Least Recently Used.

0	1	2	0	1	3	0	3	1	2	1
m	m	m	h	h	m	h	h	h	m	h

Cache: ~~∅~~ 1 ~~2~~
2 3

hit rate = $\frac{6}{11}$

0 1 2 3 0 1 2 3 . . .
m m m m

LRU: ~~0~~ 1 2
3 0

---

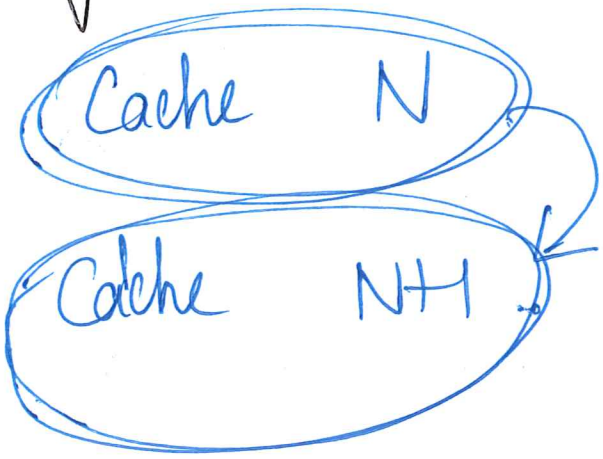
1 2 3 4 1 2 5 1 2 3 4 5

FIFO(3):

FIFO(4):

"Belady's anomaly"

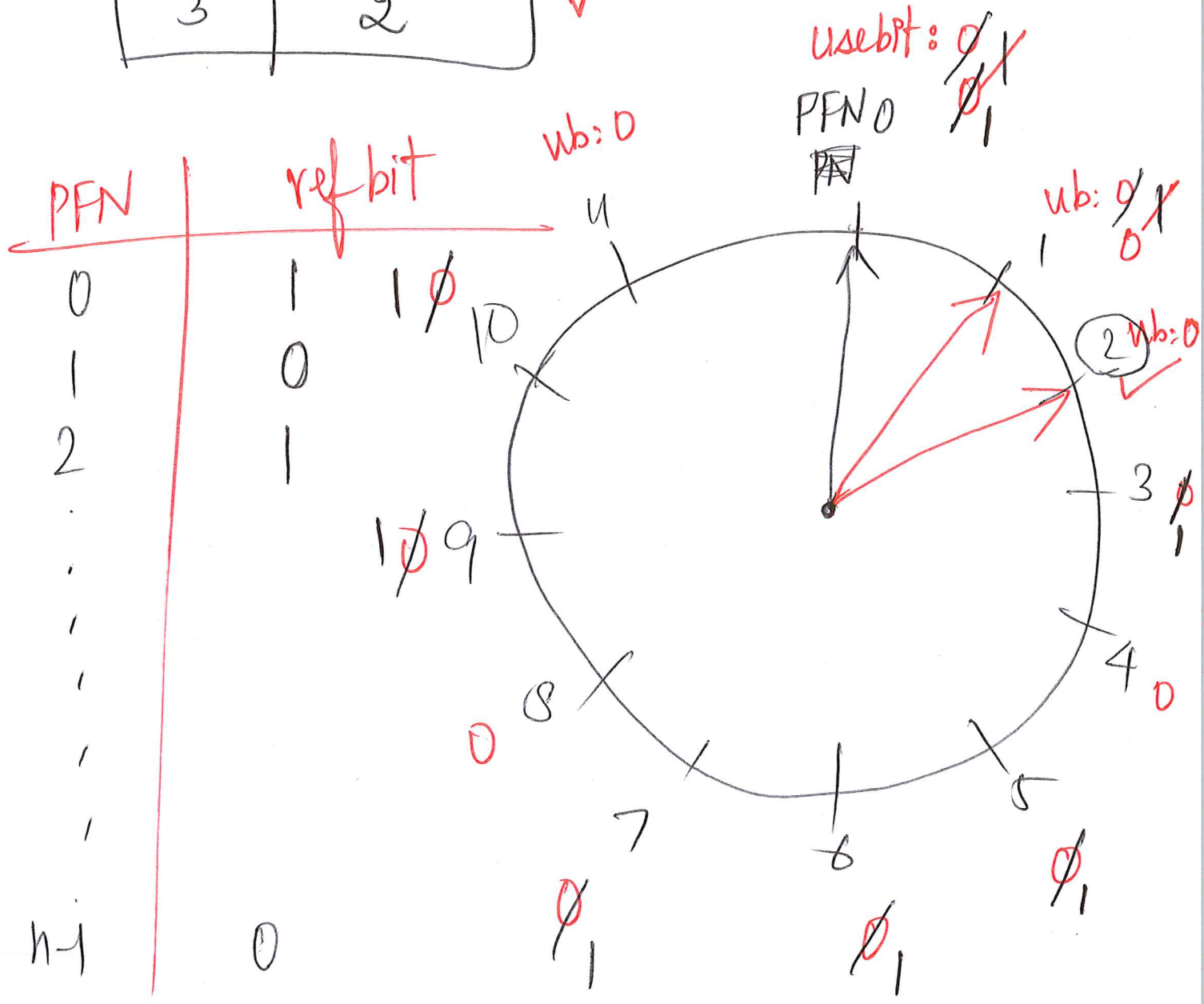
Stack property:

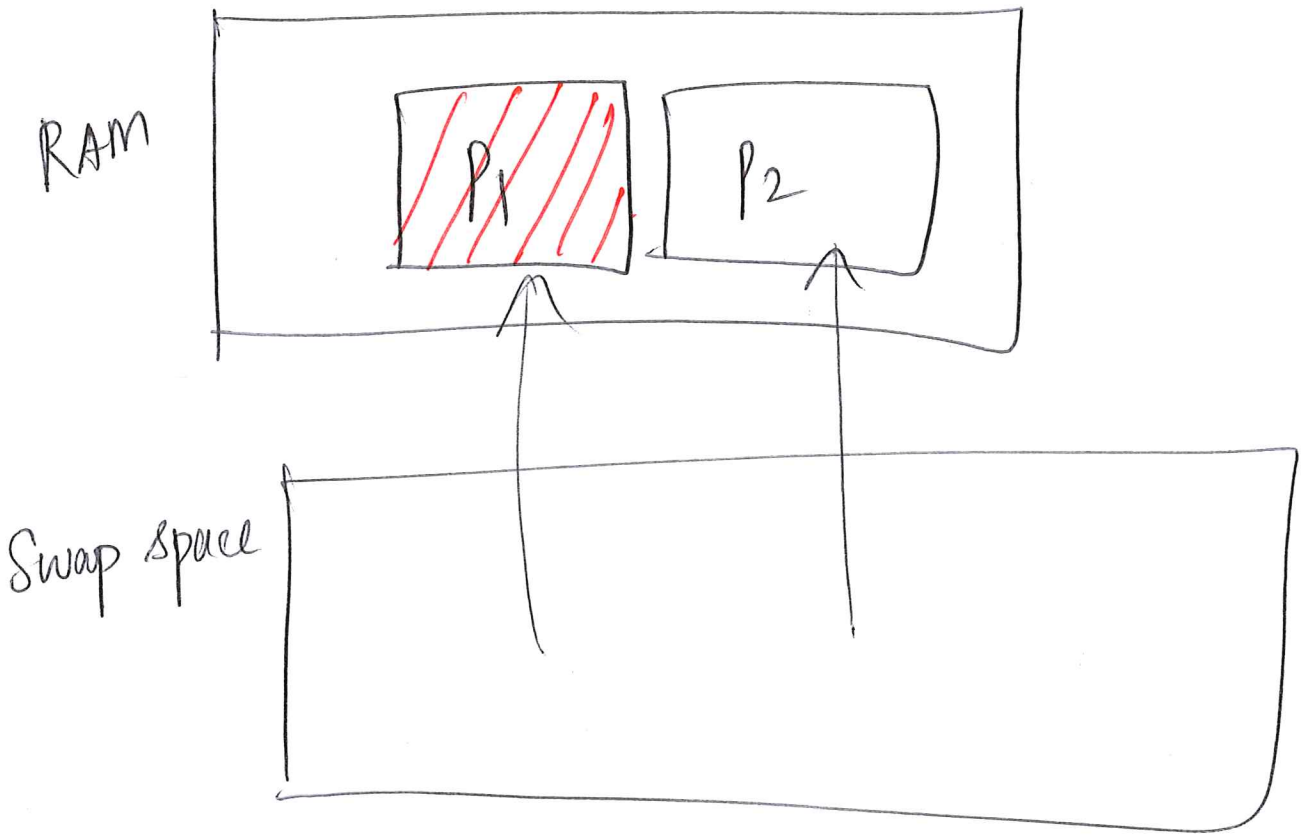


How to implement LRU?

PFN	timestamp
0	3
1	1
2	7
3	2

PFN	refbit
0	1
1	0
2	1
...	
n-1	0





Trashing

Working Set > memory size

1. Admission control.
2. Out-of-memory killer!