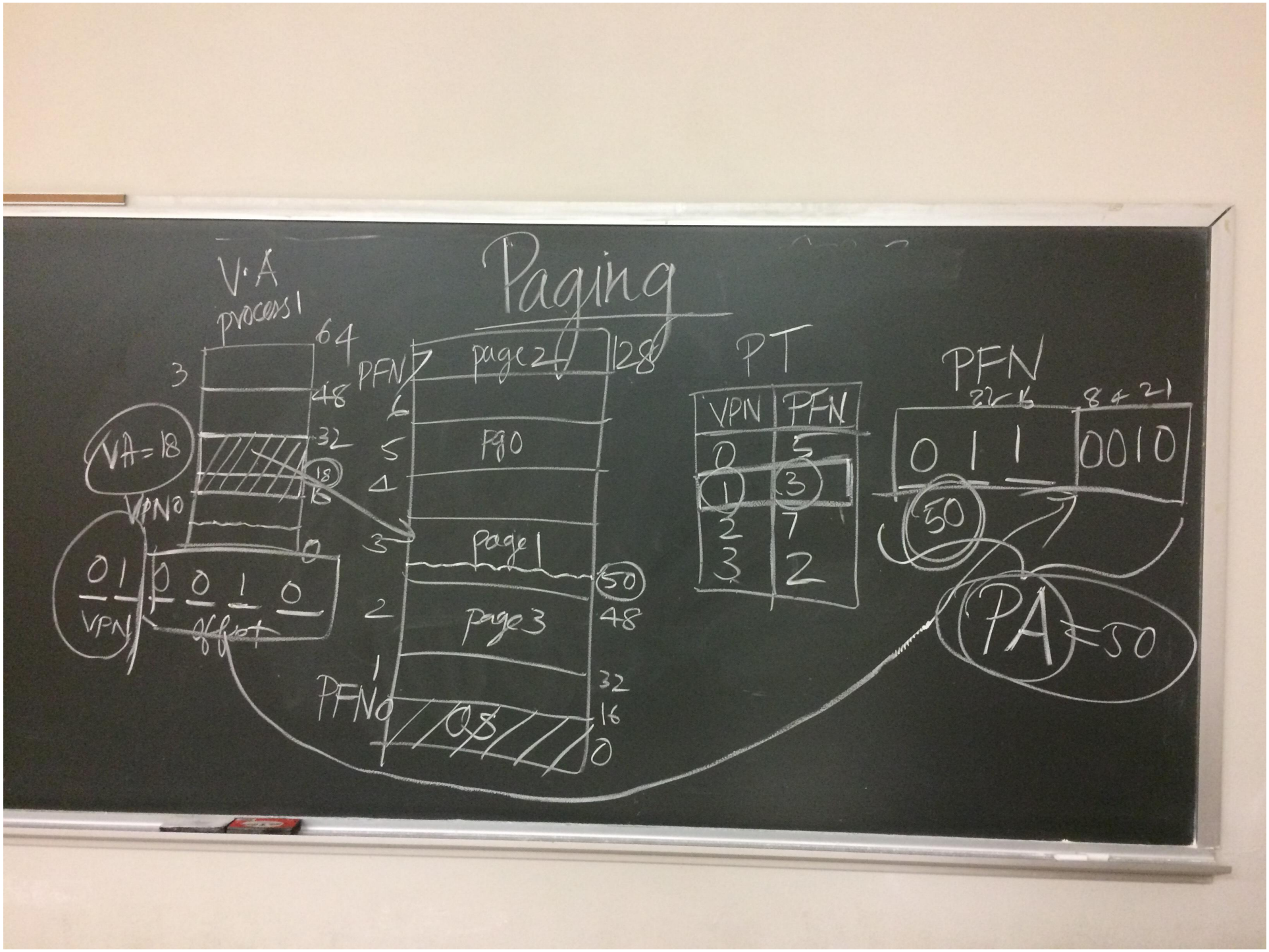
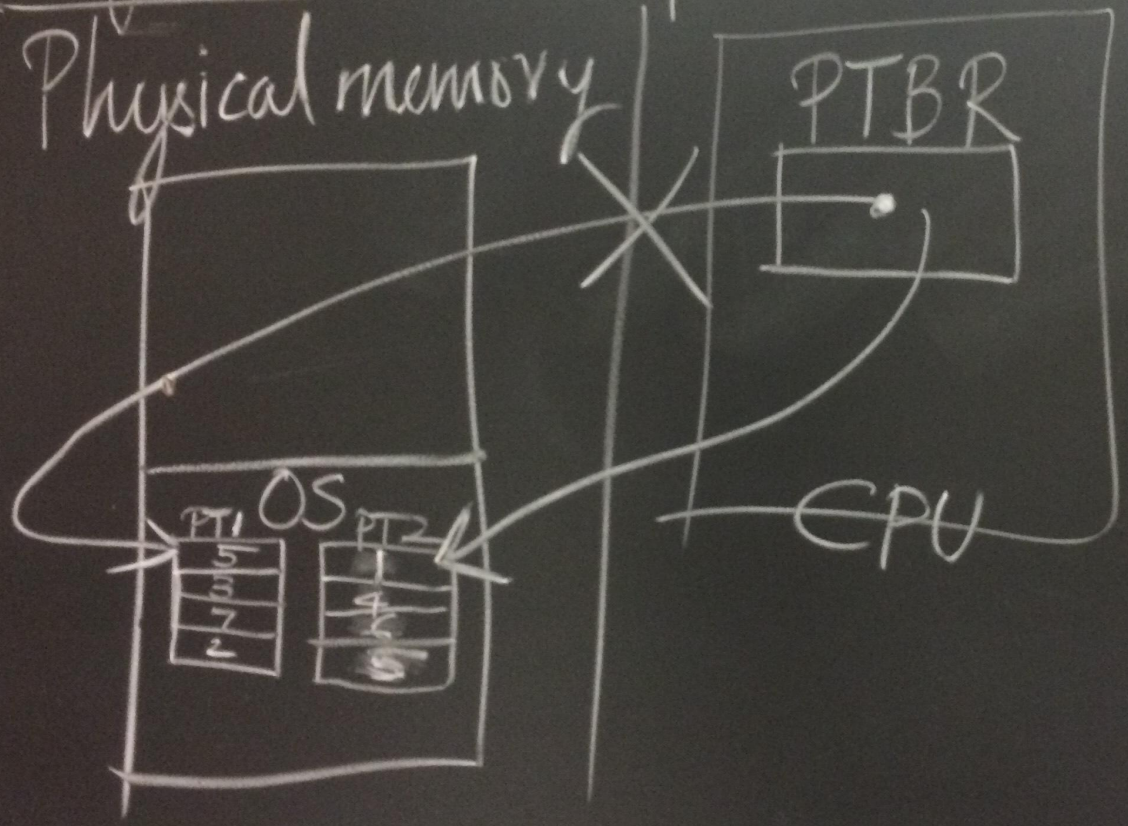
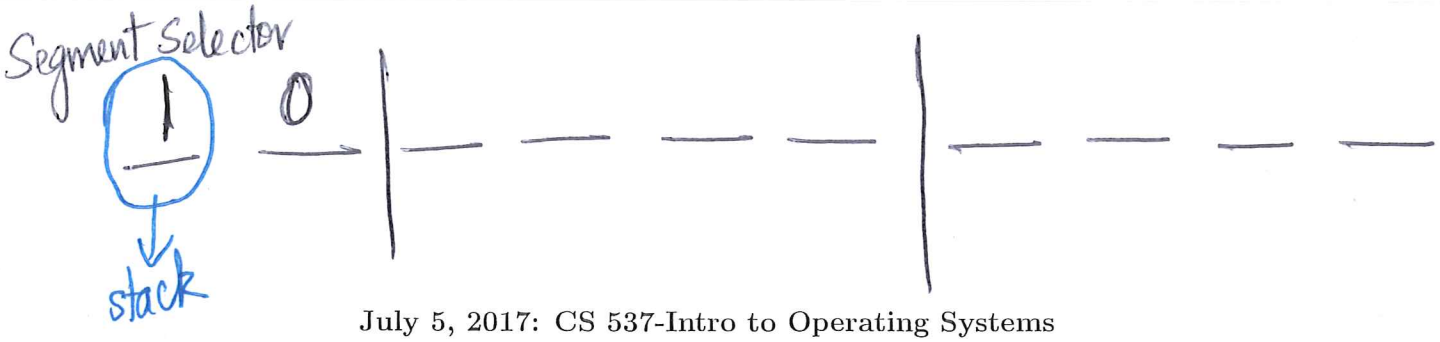

Paging
V.A
process1
64
48
32
18
16
0
3
VA=18
VPN
0 1
VPN
0 0 1 0
offset
PFN
6
5
4
3
2
1
0
page2
128
Pg0
page1
50
page3
48
32
16
0
PFN
0 8
PT
VPN
PFN
0 5
1 3
2 7
3 2
PFN
32 16
8 4 2 1
0 1 1
0 0 1 0
50
PA=50

NO SMOKING

Page Table - per process data structure

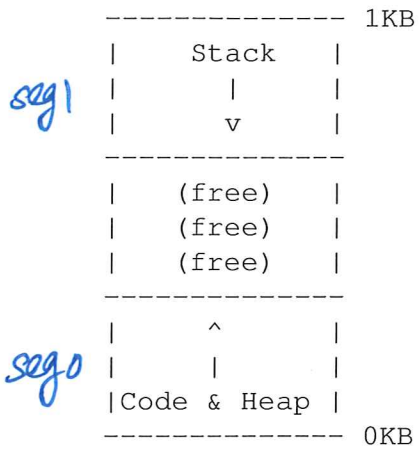




July 5, 2017: CS 537-Intro to Operating Systems

Worksheet 1 - Segmentation

Assume a system that uses **segmented virtual memory**. The segmentation that this system uses is pretty simple: an address space has just **two** segments: segment 0 and segment 1; further, the top bit of the virtual address generated by the process determines which segment the address is in: 0 for **segment 0** (where, say, **code and the heap** would reside) and 1 for **segment 1** (where the **stack** lives). **Segment 0** grows in a **positive** direction (towards higher addresses), whereas **segment 1** grows in the **negative** direction.



address space size = 1K.

phy. memory size = 16K.

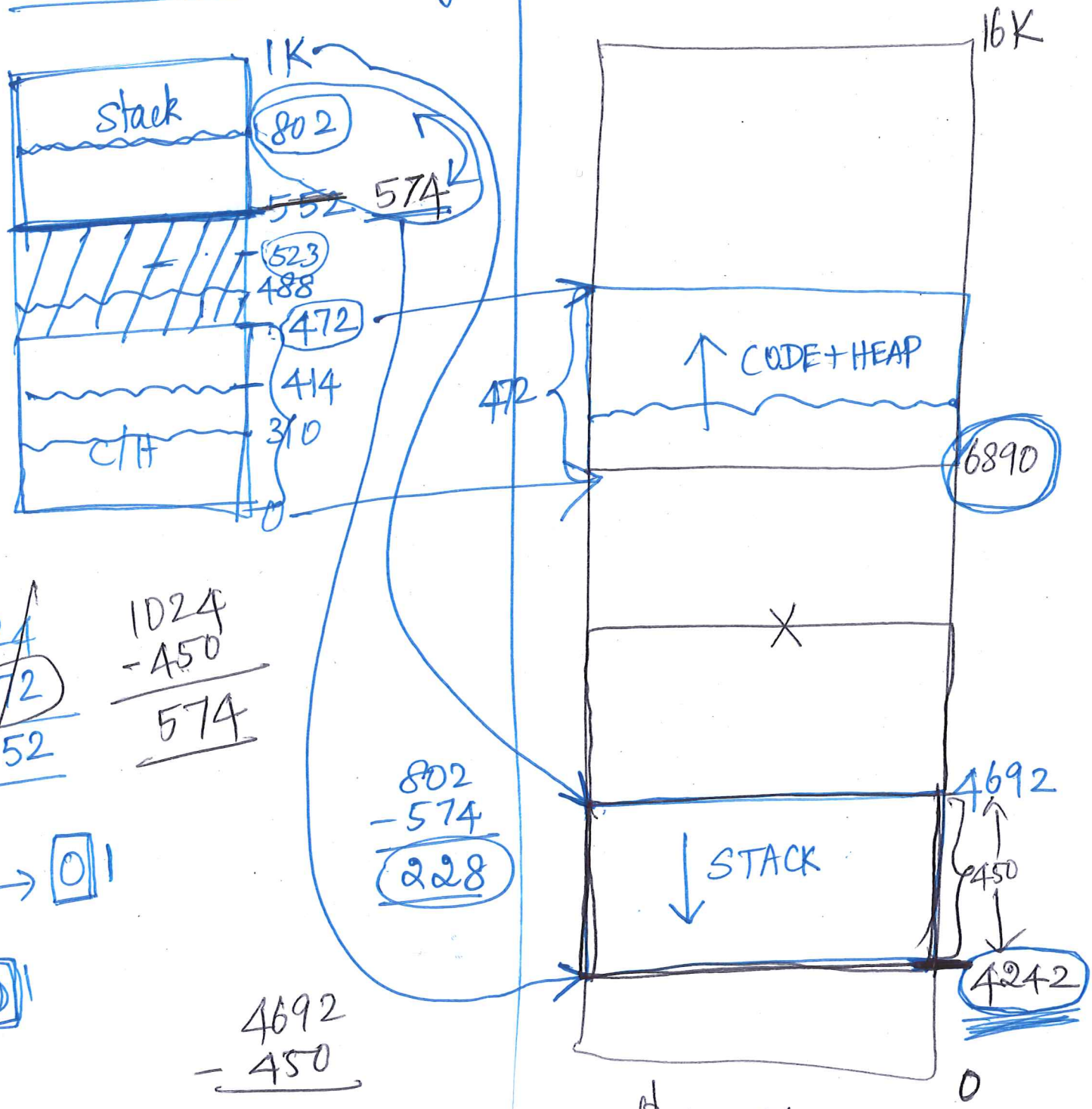
```
Segment register information:

Segment 0 base  (grows positive) : 0x00001aea (decimal 6890)
Segment 0 limit                  : 472

Segment 1 base  (grows negative) : 0x00001254 (decimal 4692)
Segment 1 limit                  : 450
```

For each virtual address, either write down the physical address it translates to OR write down that it is an out-of-bounds address (a segmentation violation).

Virtual Address	Physical Address OR Seg Fault
0x0000020b (decimal: 523)	Seg Fault.
0x0000019e (decimal: 414)	6890+414 = 7304
* 0x00000322 (decimal: 802)	4470.
0x00000136 (decimal: 310)	6890+310 = 7200
0x000001e8 (decimal: 488)	Seg Fault.

3 → 1 ⇒ stack.

Virtual A.S. Segmentation



1024 − 472 = 552

1024 − 450 = 574

802 − 574 = 228

4692 − 450 = 4242

1 → 0 | 1

0 | 1

3 → 1

1 → 0 | 1

4242 + 228 = 4470

ISSUES WITH PAGING!

32-bit machine

2^{32} = 4 GB $\quad (2^2 \times 2^{30})$ GB

page size = 4 KB = 2^{12}

$$\frac{2^{32}}{2^{12}} = 2^{20} \text{ Virtual Pages}$$

$2^{20} \simeq$ 1 million.

PT

5
3
7
2

PTE

1 PTE = 4 bytes

$2^{20} \times 4$ = 4 MB

10 processes

40 MB

400 MB

Memory Waste!

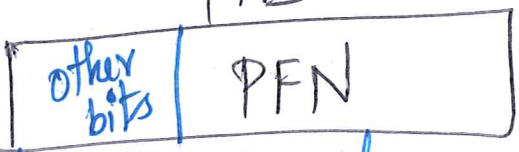
20 | mov 18, %eax | instr at addr 20.

What h/w does on each memory reference?

H/W:

1. extract VPN from V.A.
2. access memory to get the PTE

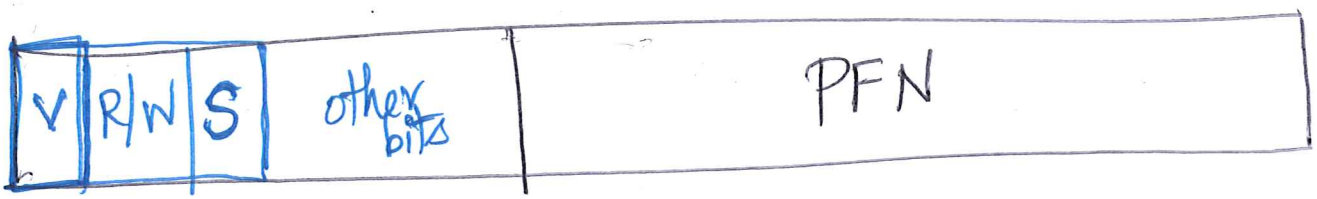
EXPENSIVE



3. Extract PFN from PTE.
4. Form the PA using PFN and offset (from V.A).
5. access memory
   PA ⟹ %eax

EXPENSIVE

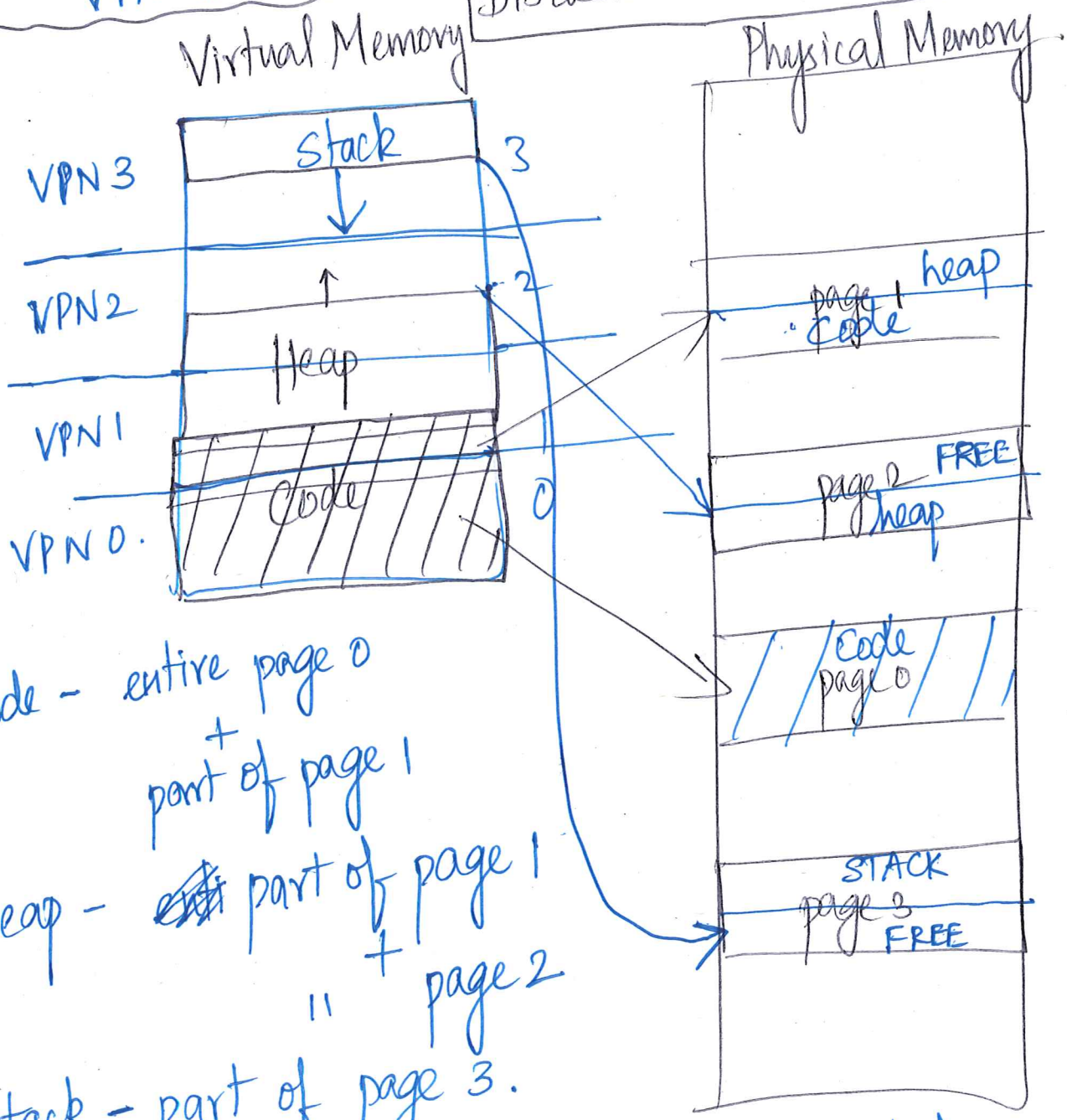
PTE



VPN 0, 3 → valid

VPN 1, 2 → invalid

DISCUSSION AFTER CLASS



Code - entire page 0
+
part of page 1

Heap - part of page 1
+
page 2

Stack - part of page 3.

FREE → part of page 2 & part of page 3.