CSE 330: Operating Systems

Class: 18 Date: 10/25

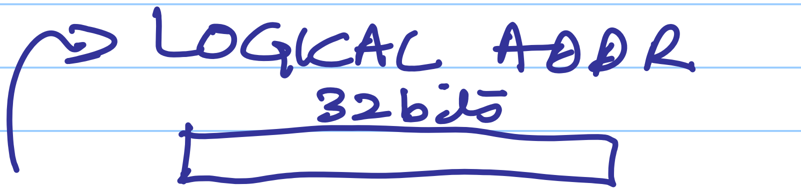
Fall 2016

Note Title

Paging

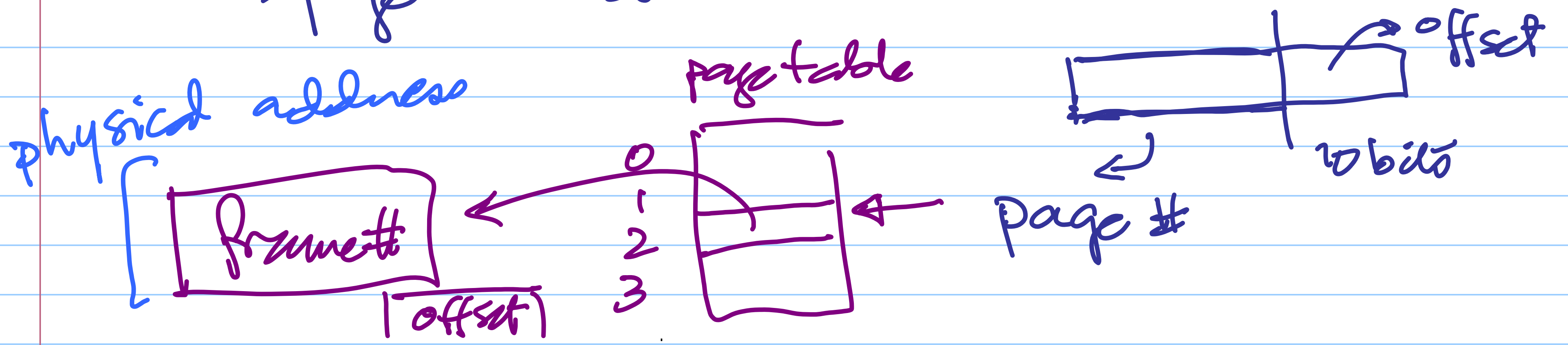
→ address translation

→ page tables



div by pagesize

1024



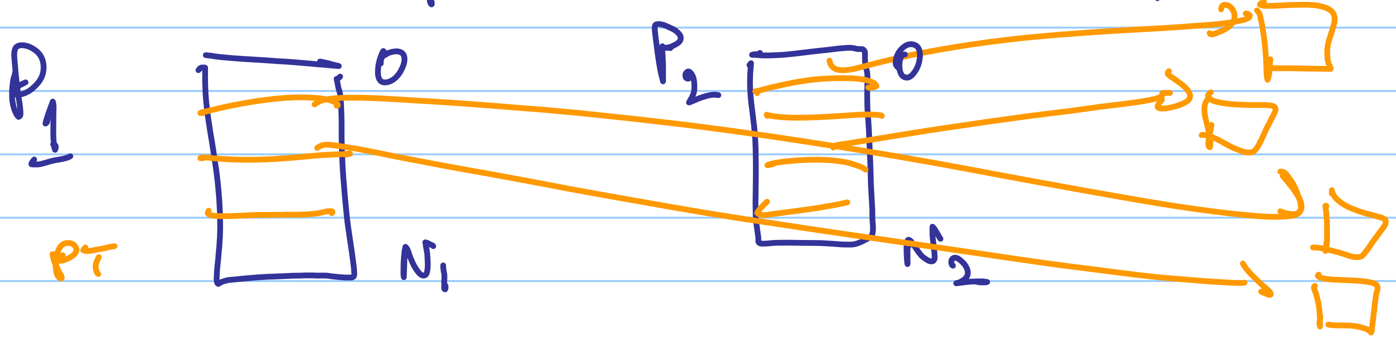
page tables

- large
- one per process

stored in RAM

physical address

address space of each process



Speed of paging

- one memory lookup

= one PTE lookup → 1 mem lookup

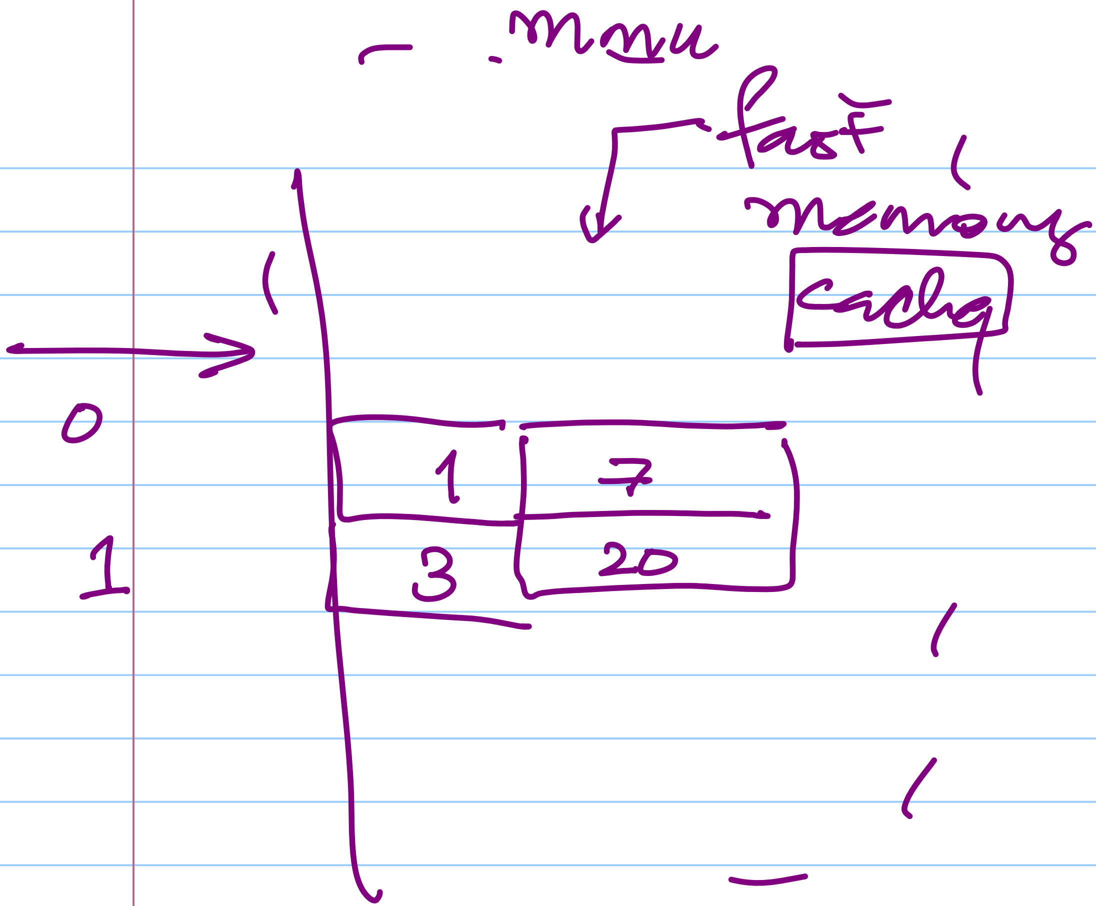
+ 1 mem lookup

= 2 mem lookup

→ BAD

PAGE lookups use caches

TLB → Translation
Lookaside
Buffer



1	7
3	20

Page table

0	5
1	7
2	3
3	20
4	9

Cache

Content addressible memory

addr	value
1	7
3	20

page #

0 → not in cache → lookup page table in memory

MISS

1 → 7

3 → 20

→ frame #] HIT

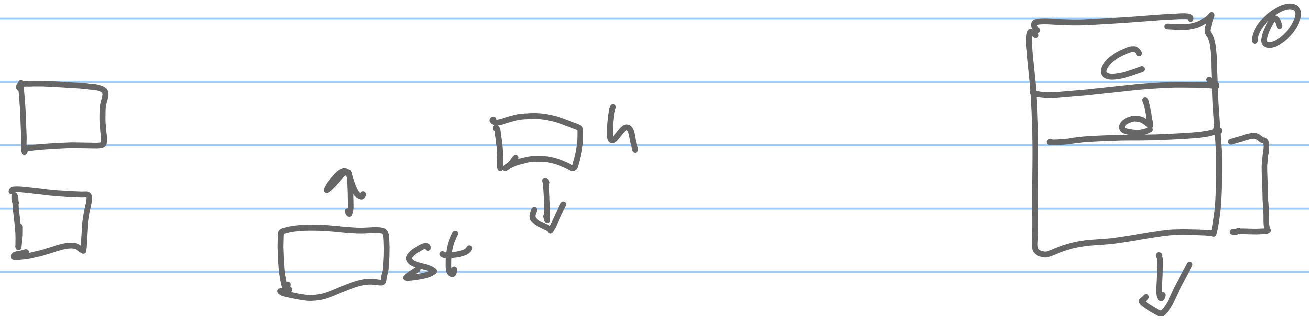
Use a TLB with hit ration

$$= h \rightarrow \frac{\#\text{ of hits}}{(\#\text{ of hits}) + (\#\text{ of misses})}$$

time per mem access $= \left[1 + (1-h)\right]$ memory cycles problems with paging

→ arbitrary divisions of Code, data, etc

→ cannot handle growth.



→ code sharing]
→ library "] hard with paging

Segmentation → no pages

program has segments

↳ logical chunks

program / process

→ code segment → modules
  - → libraries

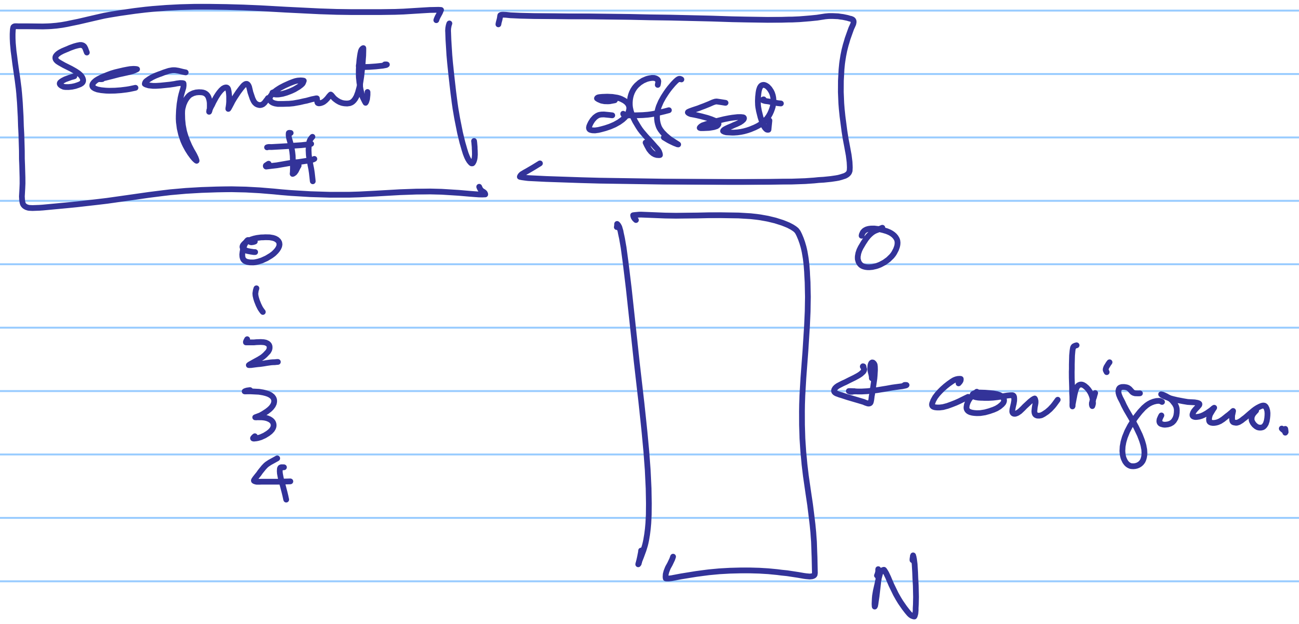
data " ≡

heap "

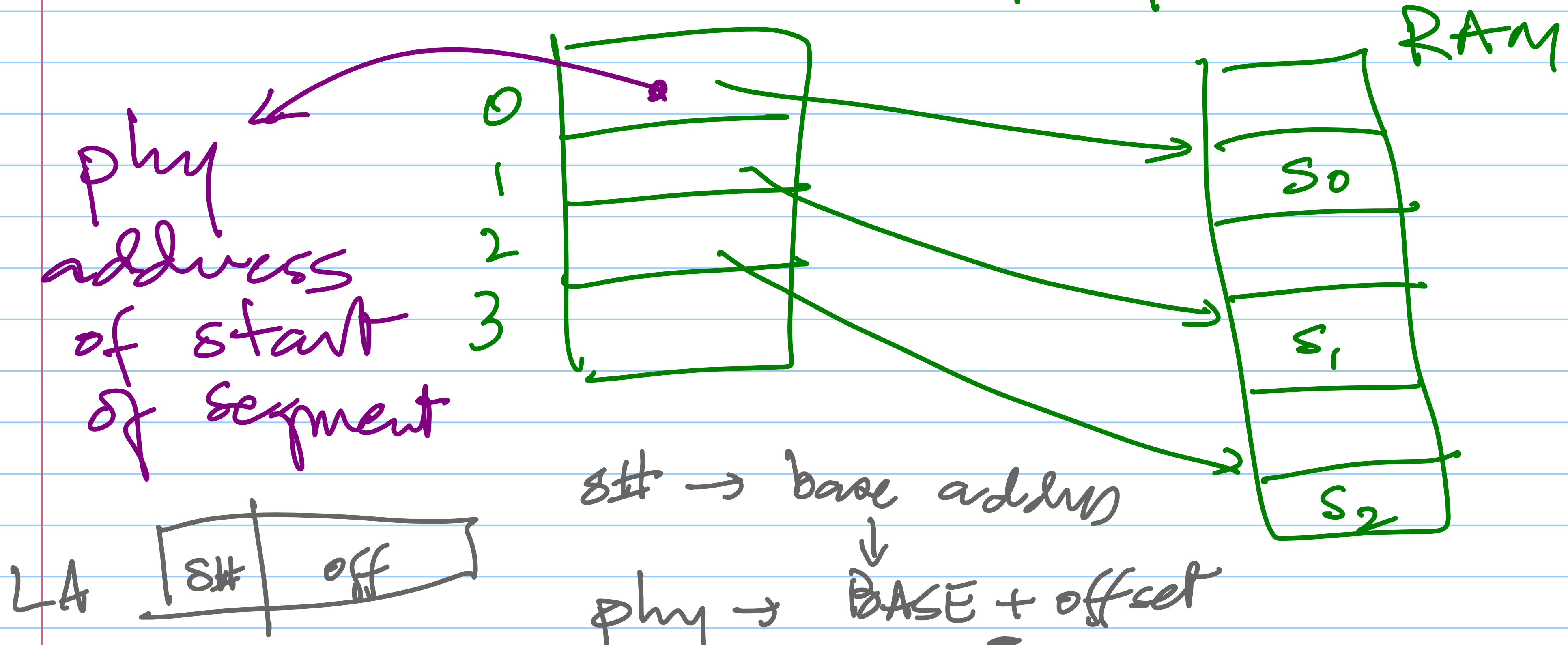
stack "

Addressing

Logical address =



Segment table per process



s# → base addr

↓

phy → BASE + offset

Segments — contiguous in phy mem

→ all kinds of sizes

→ dynamic → grow/shrink

→ create/delete

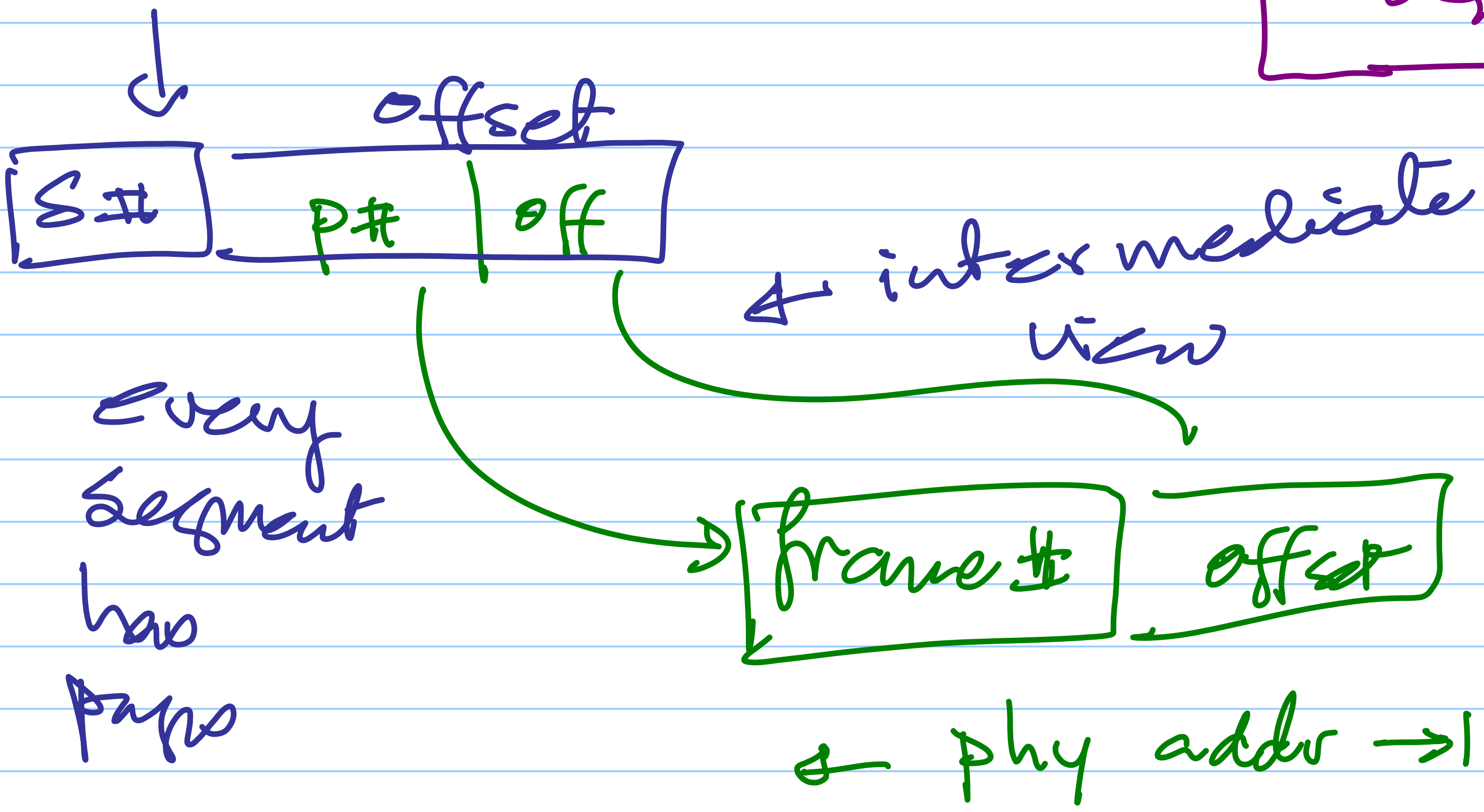
Use segmentation at "top"

" paging " "bottom"

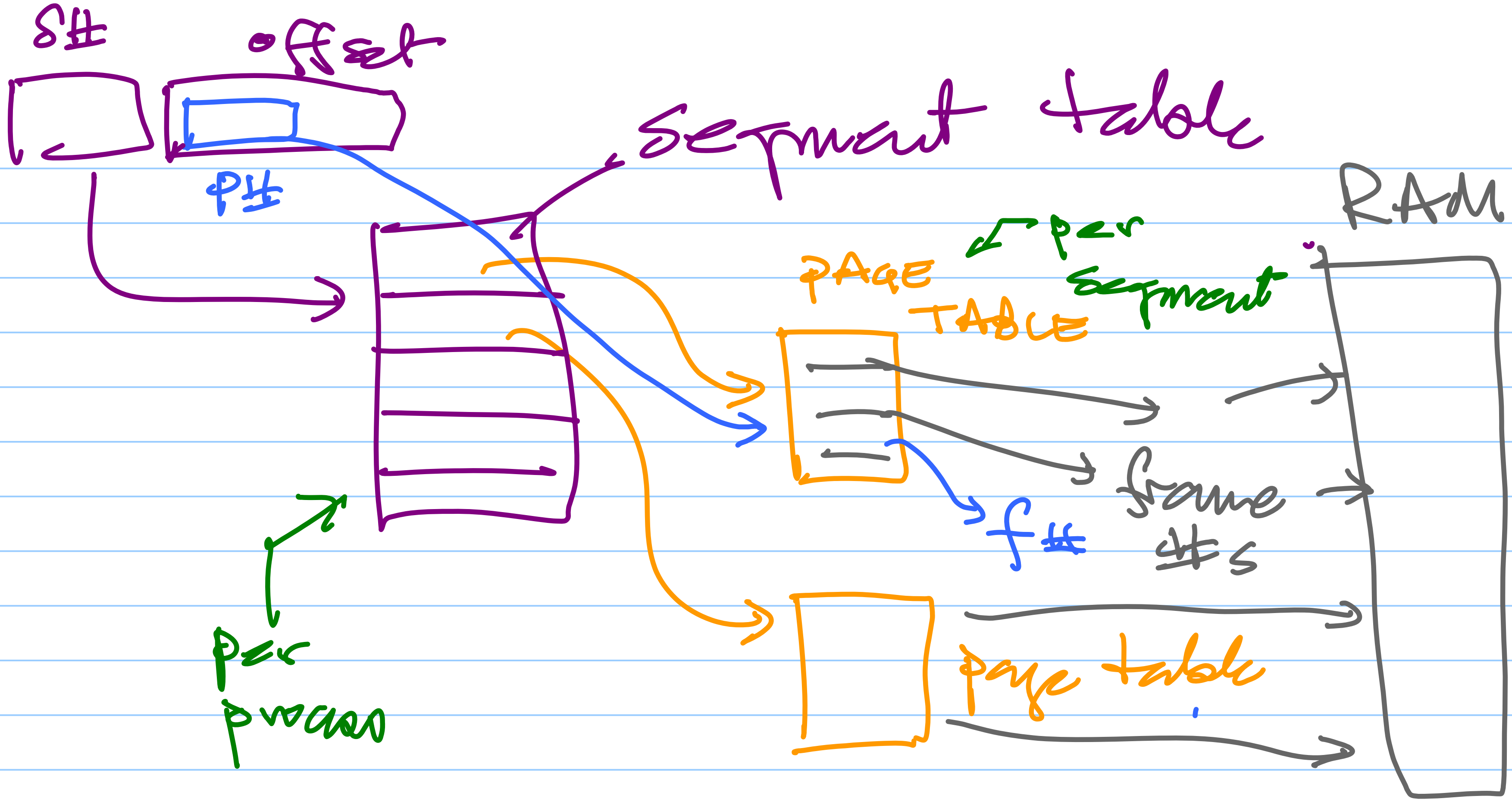
↳ page the segments

PAGED SEGMENTATION

process/CPU view (logical)



every segment has pages


S#
offset
P#
Segment table
RAM
per segment
PAGE TABLE
f#
frame #s
page table
per process

VAX memory management → late 1970's

↳ 32 bit machine

↳ is v. similar to Intel 32 bit processors.

32 bit – 4GIG

↳ 4 segments →

processes

P0

P1

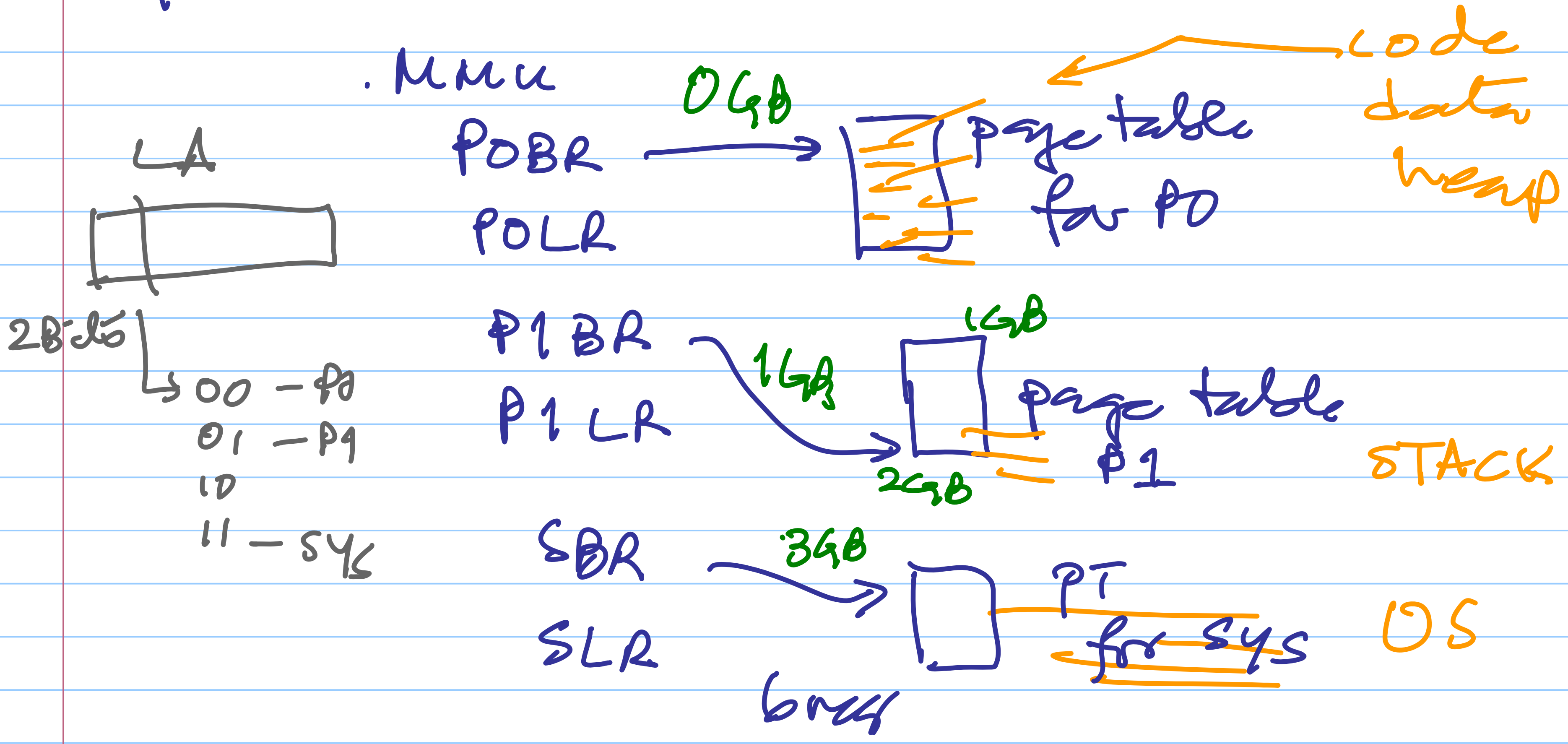
R

SYS

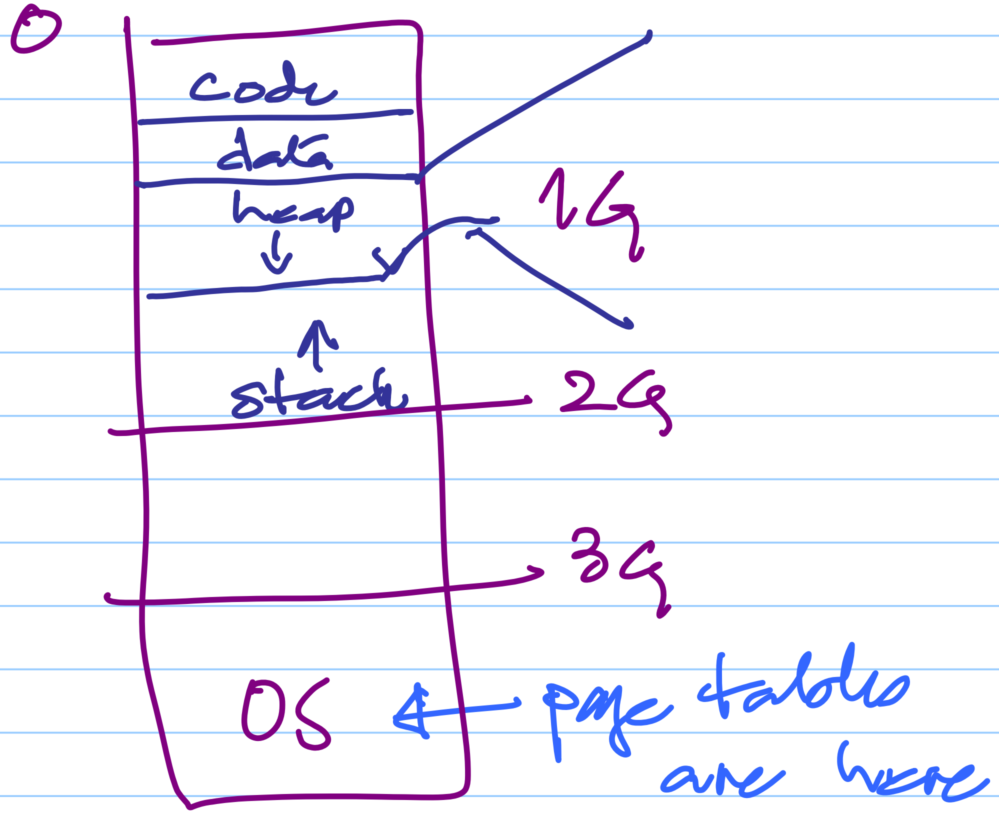
1G each.

OS

process → 3 tables P0, P1, SYS



process addr space


0
code
data
heap
stack
1G
2G
3G
OS
page tables are here