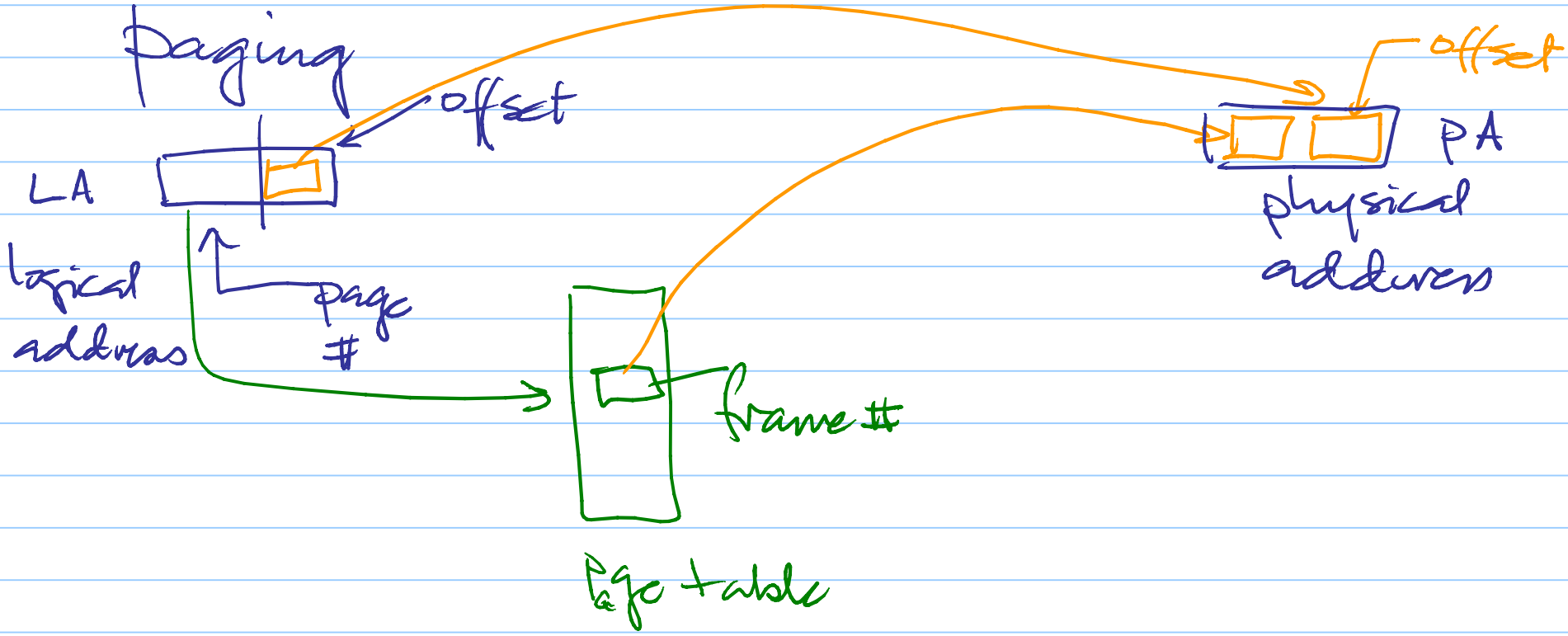
CSE 330: Operating Systems

Class: 18 **Date:** 3/20

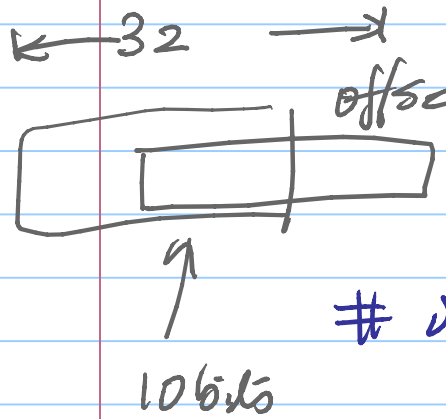
Spring 2018

Note Title



Size of page table

(depends on page size, logical memory size)



page size → 1KB

logical memory size → 1MB

$$\#\text{ of PTE} = \frac{1\,MB}{1KB} = \frac{\text{Logical mem size}}{\text{page size}}$$

$$\downarrow$$

1000

page tables are stored in memory

each mem fetch

= page table lookup
+ data lookup]

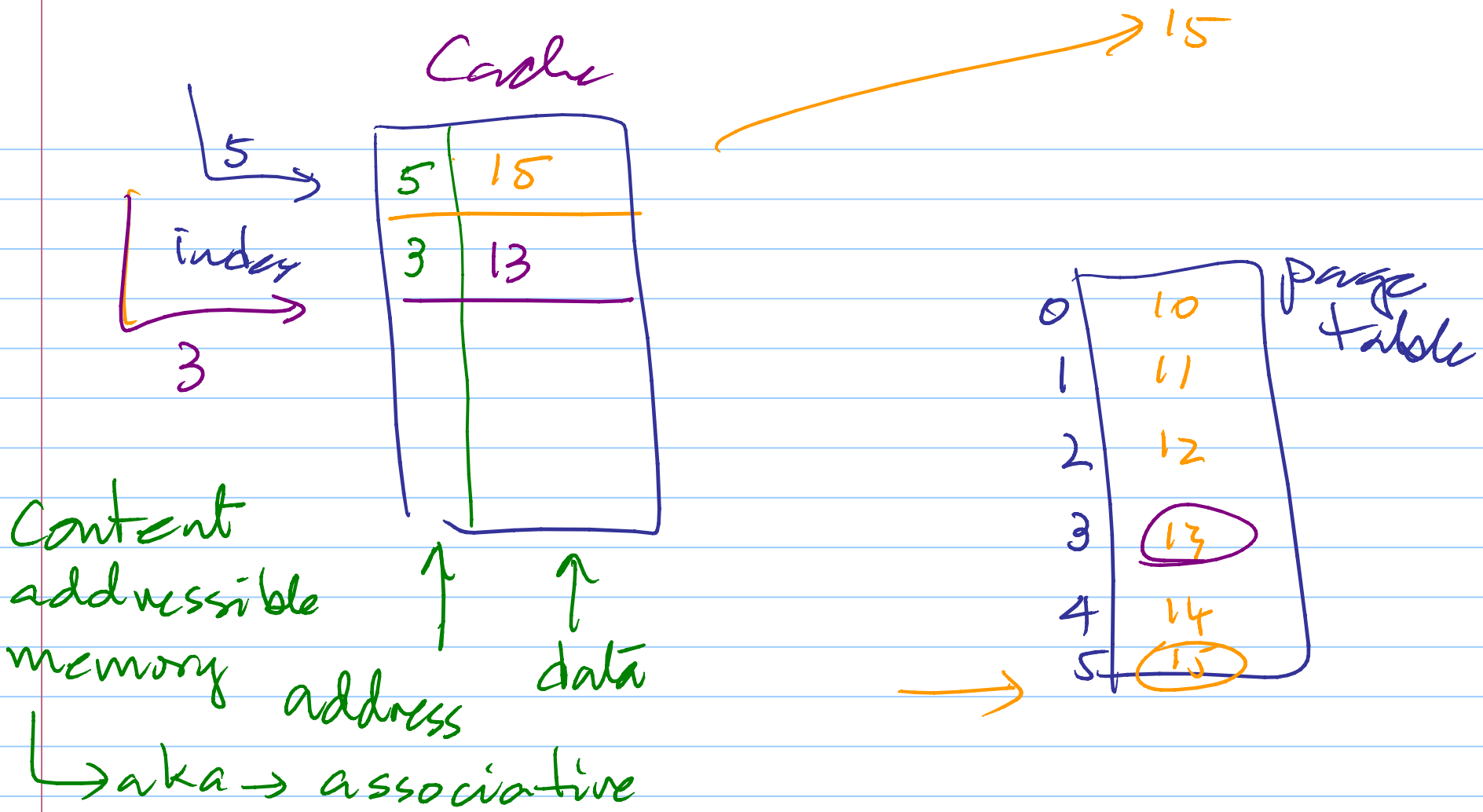
2 mem cycles → slow

Use a page table cache @ MMU

→ called a TLB

→ translation lookaside buffer

→ contains parts of the page table (populated on demand)


Cache
5
index
3
5
15
3
13
15
0
10
1
11
2
12
3
13
4
14
5
15
page table
Content addressible memory
aka → associative
address
data

get a translation from cache

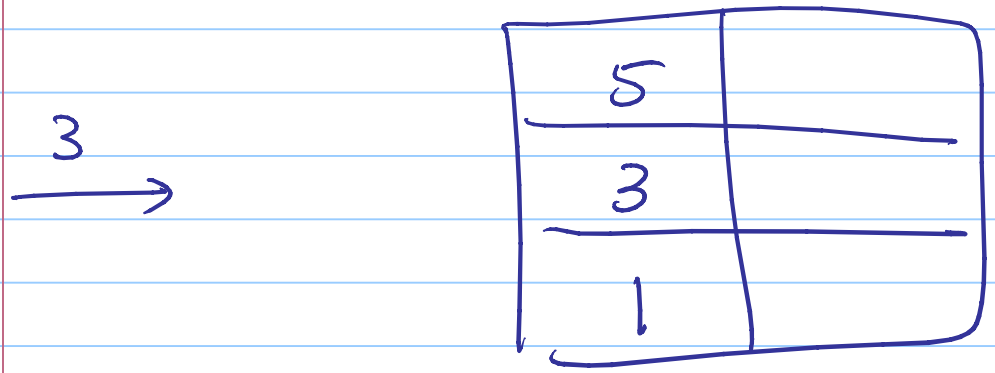
↳ cache hit — yes → fast

↳ cache miss — no

↳ load to cache slow

hit ratio >> 90%

↳



↳ parallel search in hardware ...

Caches can fill up?

→ yes → then → do cache replacement

delete old entries

+ add new entries

access speed of memory

hit ratio = h %

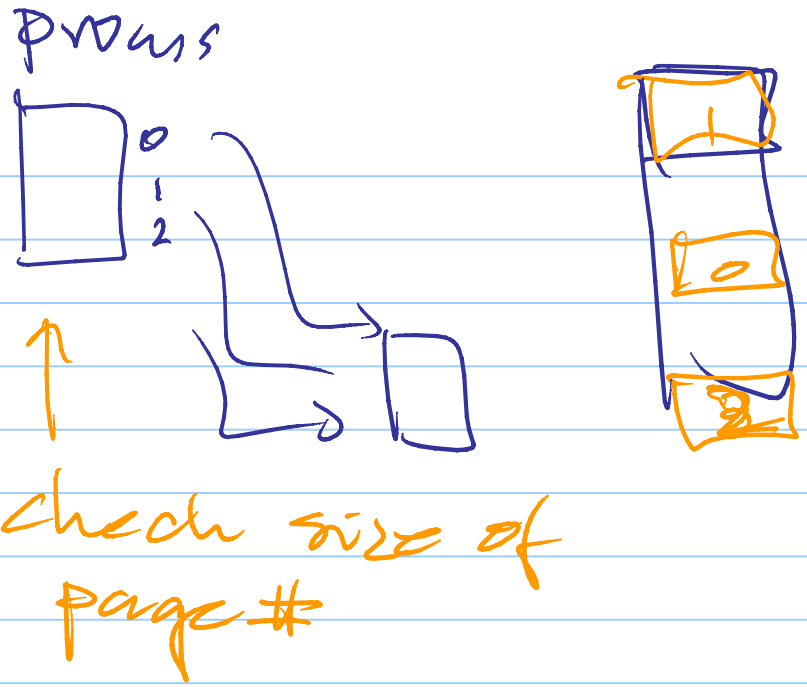
time = (h) * (mem cycle time)

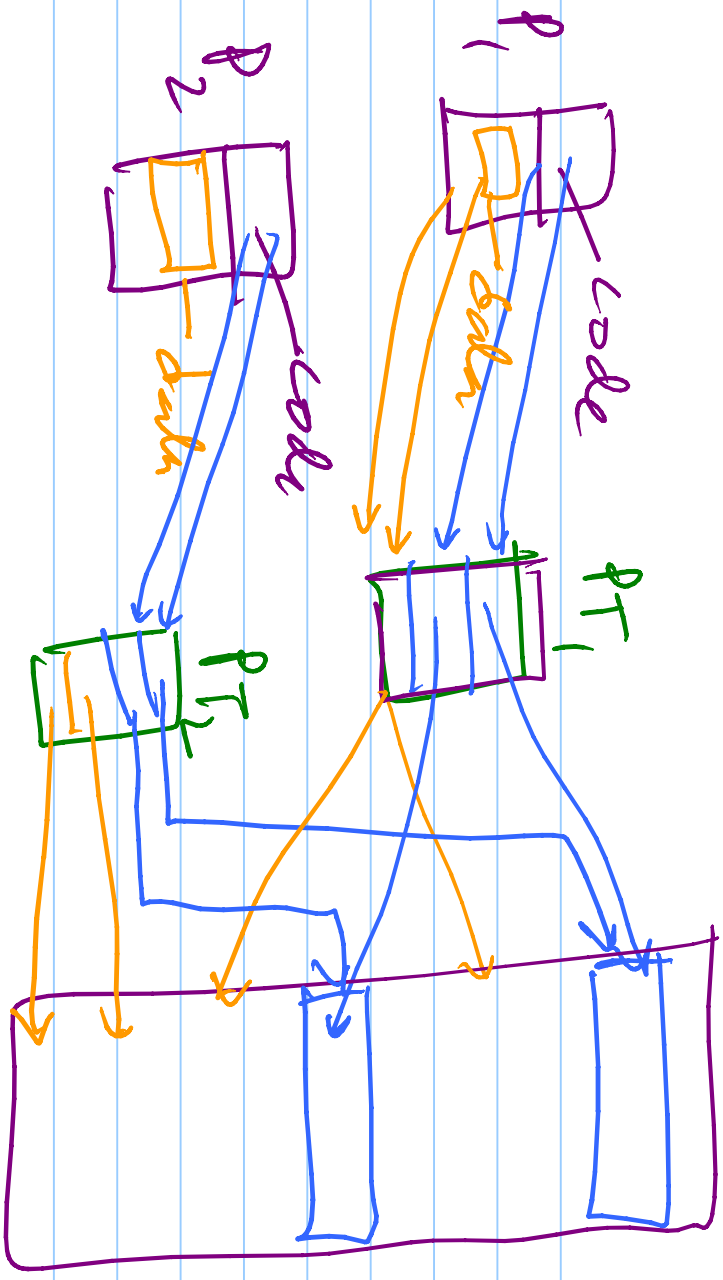
+ (1 − h) * (mem-cycle time) * 2

// assume cache hit takes no time

Paging benefits

1. protection
2. sharing
3. growth.
4. lack of fragmentation

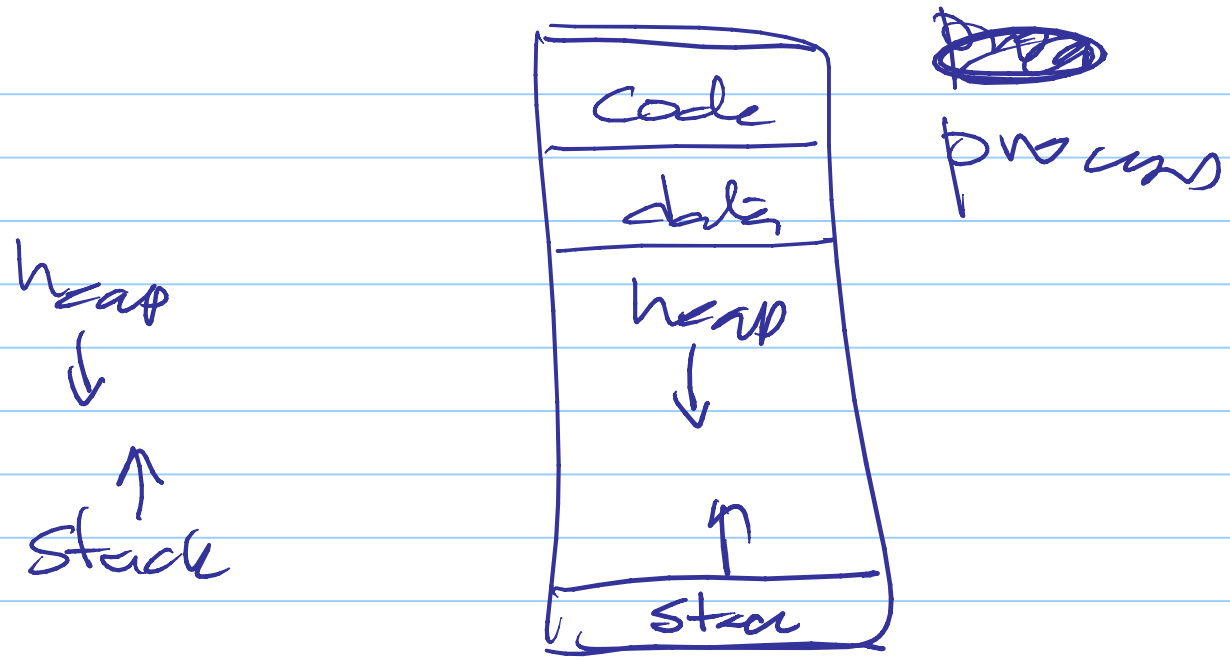



P1
P2
code
data
code
data
PT
PT

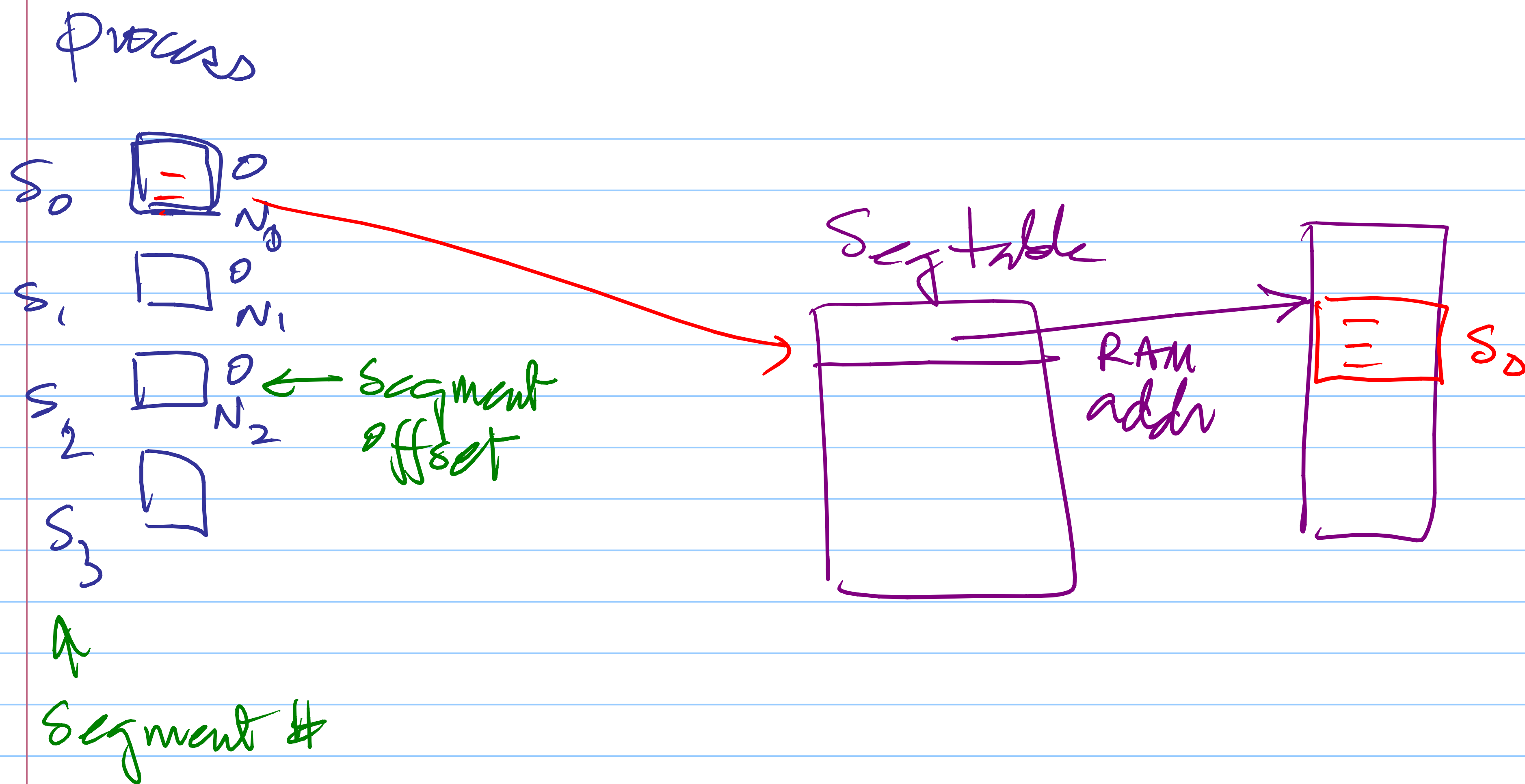
maybe the page
table layer at
runtime

Paging problems

1. arbitrary piece (pages)
2. poor sharing
3. No dynamic growth possible


heap
stack
code
data
heap
stack
process

- Segmentation


Process
S_0
S_1
S_2
S_3
0
N_0
0
N_1
0
N_2
segment offset
Segment #
Seg table
RAM addr
S_0

paging + segmentation

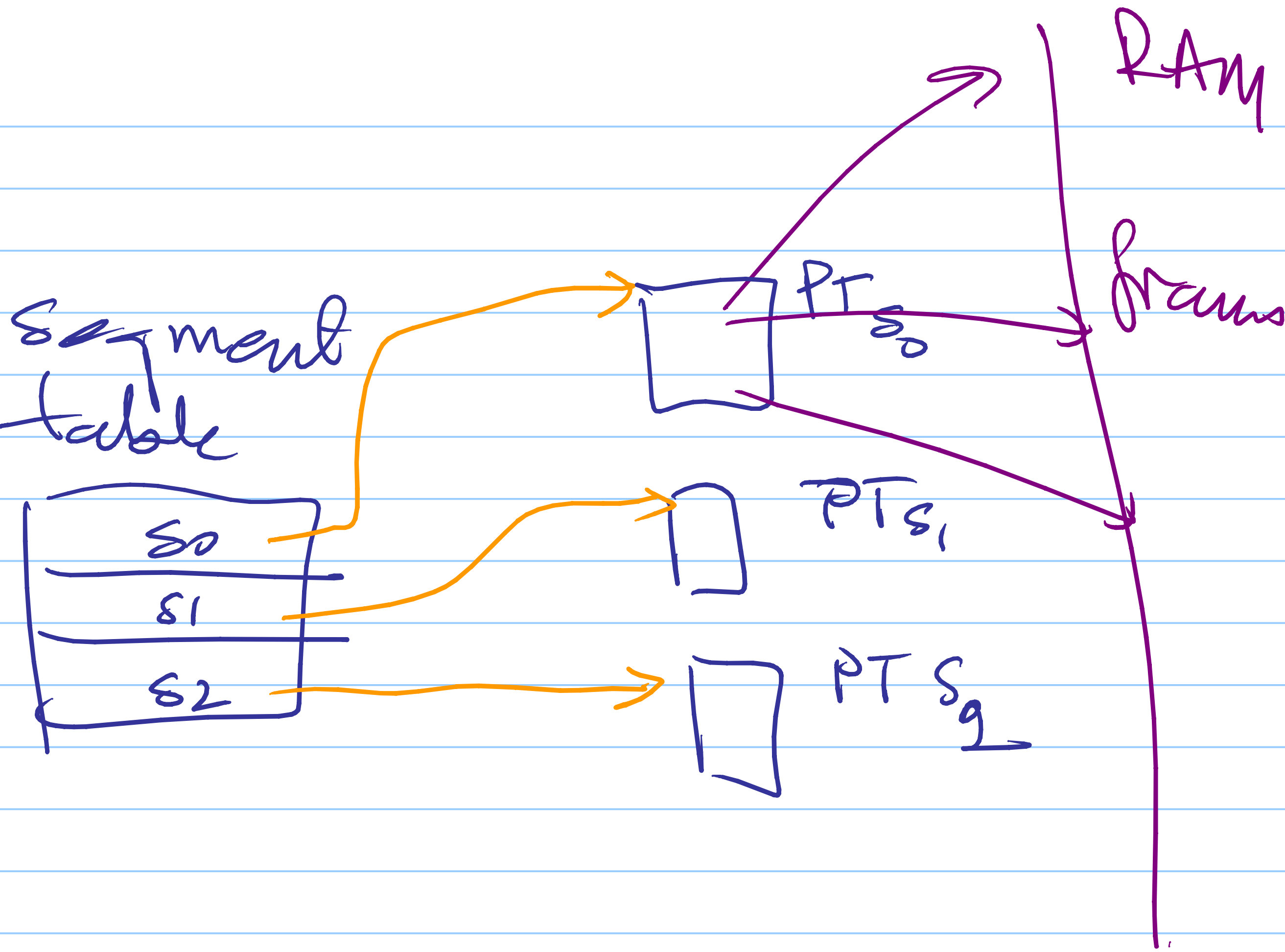
= paged segmentation

process → set of segments → set of pages

"page the segments"

1 process

↳ 1 segment table

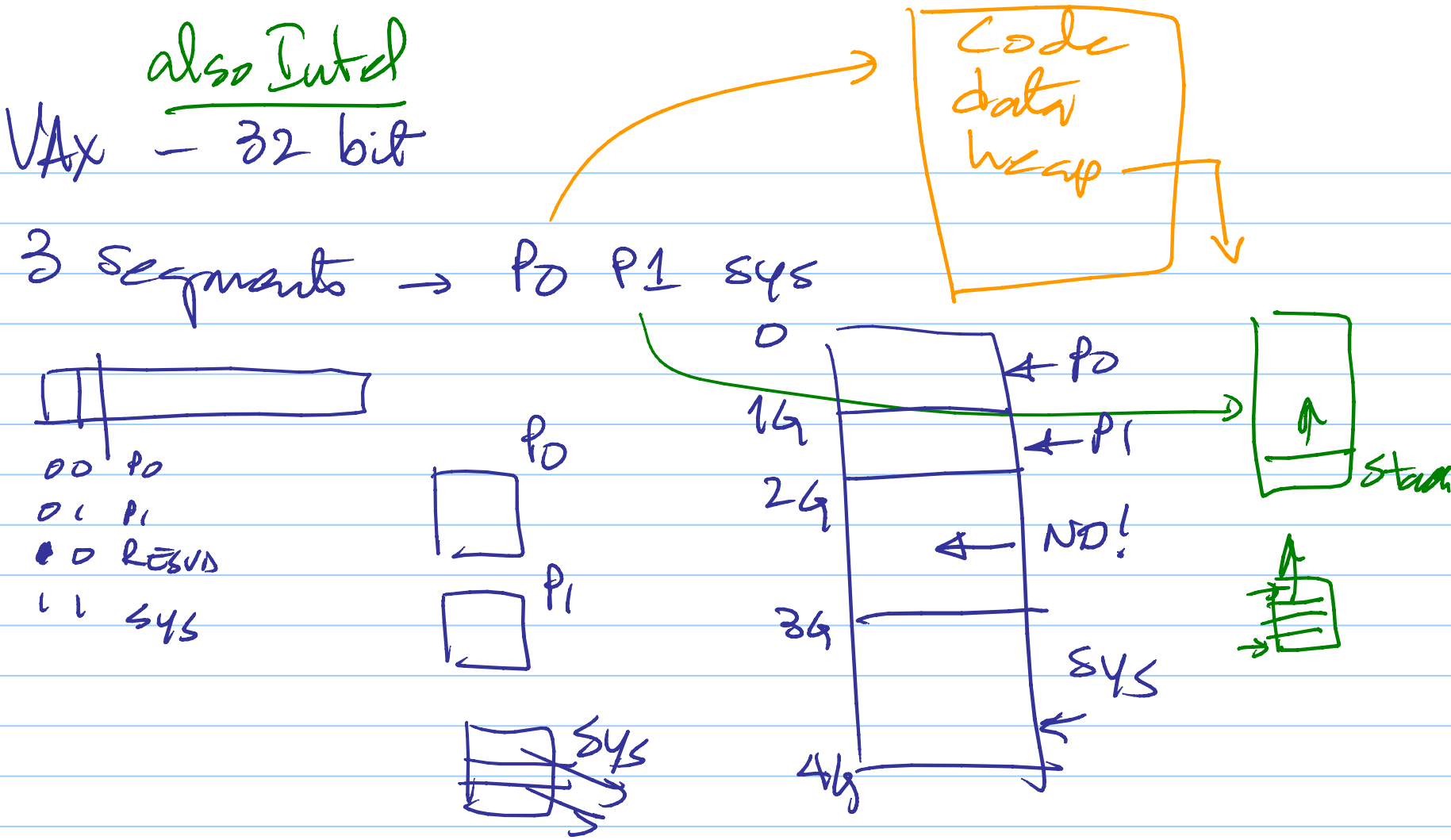


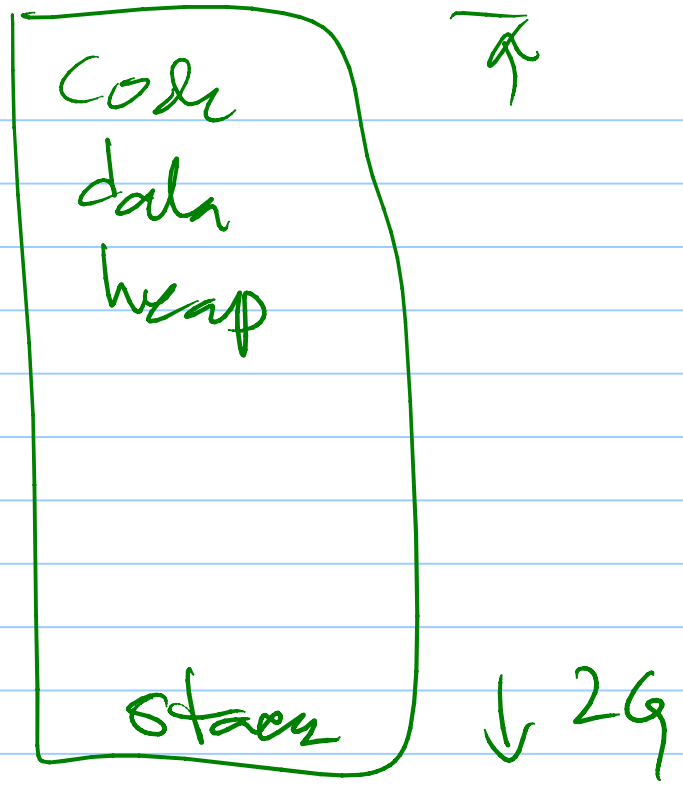
VAX memory architecture

↳ 1983

also Intel

VAX – 32 bit




code
data
heap
stack
↑
↓ 2G

P
PS
PC
P2
PS
PC