Today: More of the same

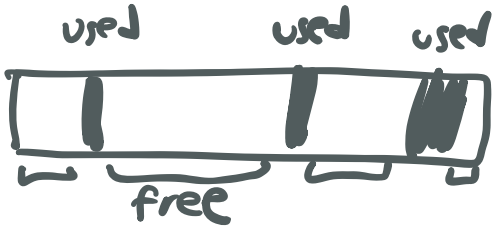
Virtual Memory:

Goal: each process w/ illusion (own large private memory)

Three techniques
- => Base/Bounds
- => Segmentation
- => Paging

Paging → fixed-sized units (pages)
- no external frag.

Base/Bounds, Segmentation → variable-sized allocation
- ⇓ external fragmentation



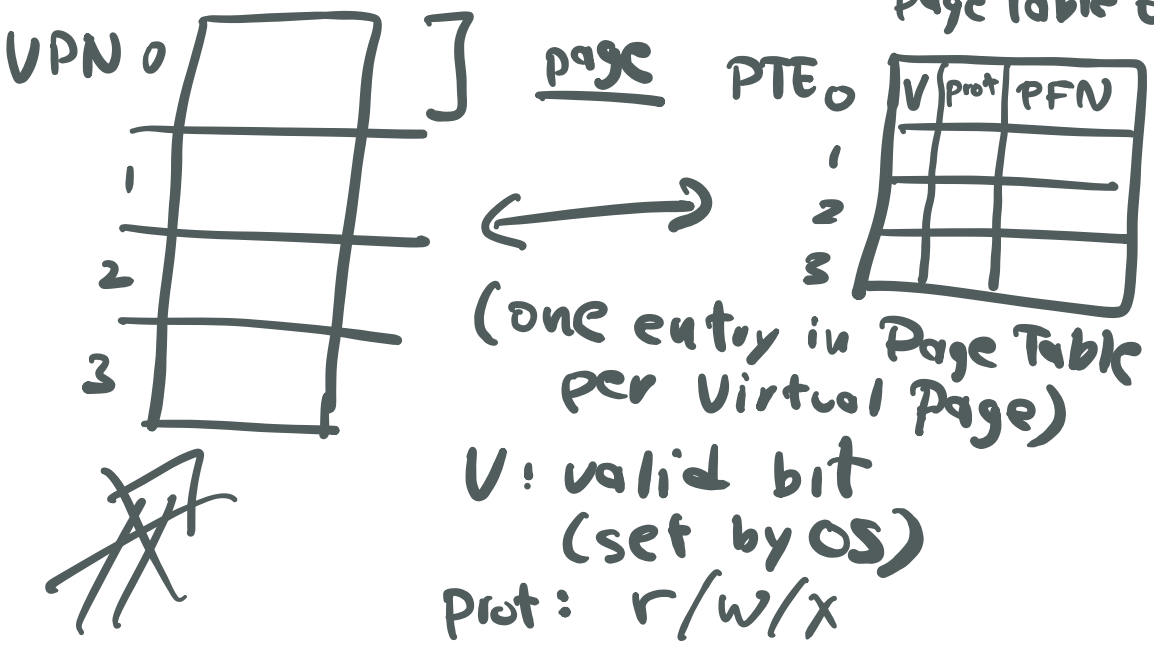
Paging: (TLB)
- => Too slow
- => Too big ← focus (today)

Structure of Page Table:
- Linear (Arrays)

Virtual Address Space

Page Table (for that process)

Page Table Entry



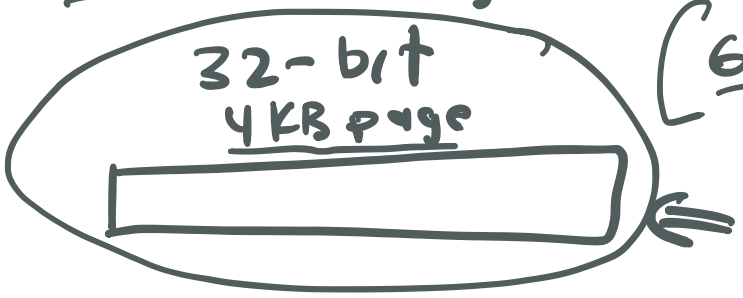
(one entry in Page Table per Virtual Page)

V: valid bit (set by OS)

Prot: r/w/x

PFN: phys. frame number

Problem: large virt. addr. space [64-bit]



how many virtual pages?

$$\frac{2^{32}}{2^{12}} = 2^{20}$$

2^{20} = 1 Million

$$\frac{2 \times 2 \times 1 \ldots \times 2}{2 \times \ldots \times 2}$$

2^{20} entries per page table

Size: 2^{20} x PTE size

(4 B)

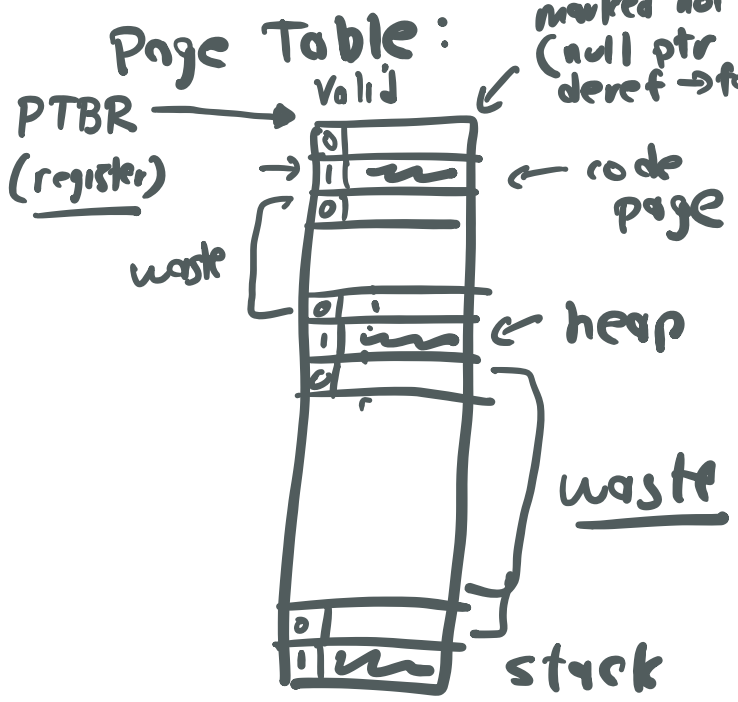
: [4 MB]

e.g.
300 processes => 300 x 4 MB

~1.2 GB of memory

=> 32-bit VA space, 4KB pages

Page Table:

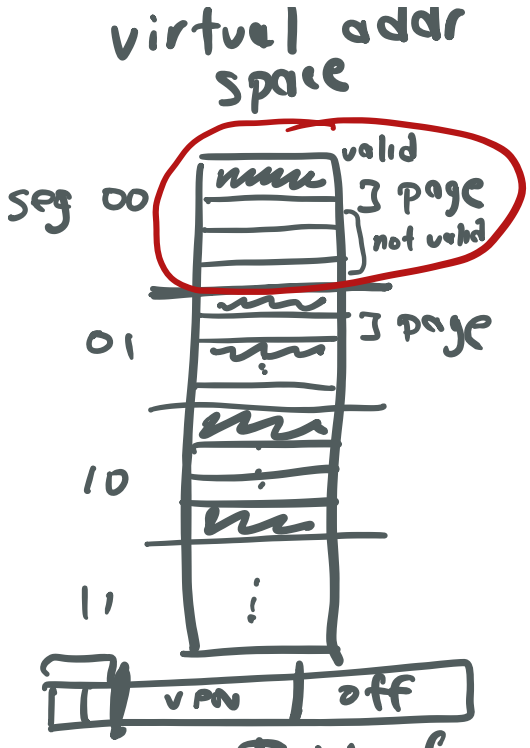


typical usage:
=> few code pages
small heap
small stack

How to make smaller tables?

Technique #2: Hybrid

Paging and Segmentation



Page Table for Segment 0:

base → [V | prot | PFN] one entry

bounds: 7

(somewhere in phys memory)

1 page table per segment (per process)

per segment:

base / bounds

⇓ page table for this segment resides

⇓ "size" of page table (of that segment)

Midterm: ⇒ tomorrow, 7:15pm (2 hours)

review: today in [1125 Biochem]

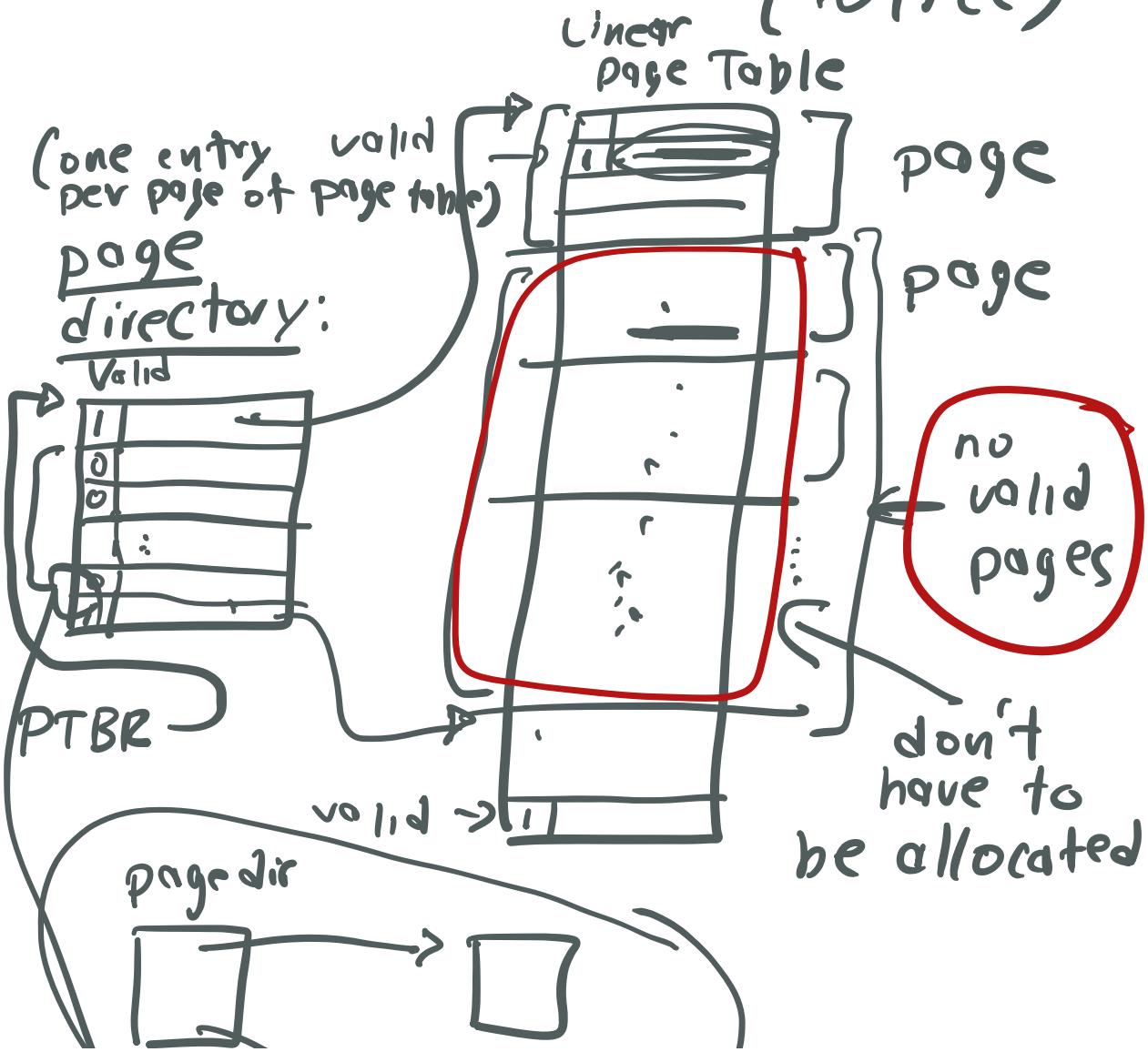
⇒ pencil(s)

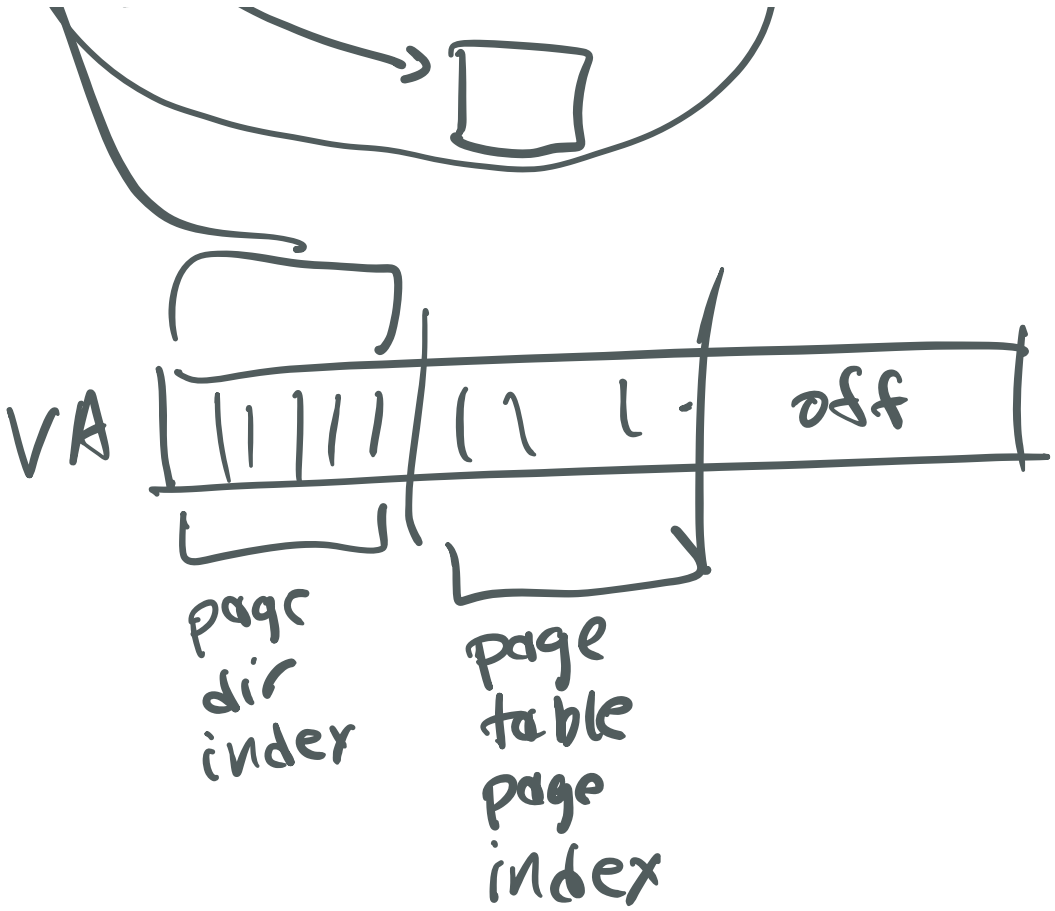
⇒ "helper" sheet: just 1

Problem w/ Seg + Paging:

=) ext. fragmentation
(just for pieces of page table)

Technique #2: Multi-level Page Table (~Tree)




VA
off
page dir index
page table page index