LECTURE 7:

Fancy Page Tables

Last Time:
Page Tables: Too ~~Big~~ Slow ⇒ solution TLB
This Time:
P.T. : Too Big!

Q0 why do we want (large virtual address spaces? (easy to program)

Example: (4GB) 32-bit AS w/ 512 byte pages [4 byte Page Table Entry]

Q1 ⇒ how many bytes / PT? (array)

A) 32 | 23 | 9

2^{23} entries in linear PT

$8 \cdot 2^{20} \Rightarrow \sim 8$ million

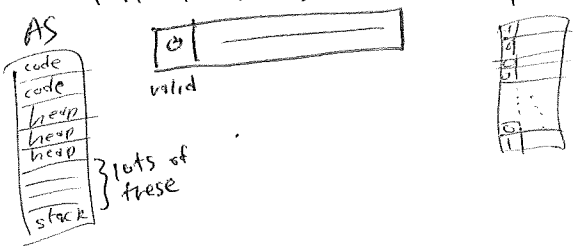
$\times 4$ b/PTE $\Rightarrow$ 32 MB / PT

~100 processes ⇒ 3200 MB ~ 3 GB of mem

500 processes ⇒ ~16 GB

Q2 ⇒ why so big?

full of PTEs like this:

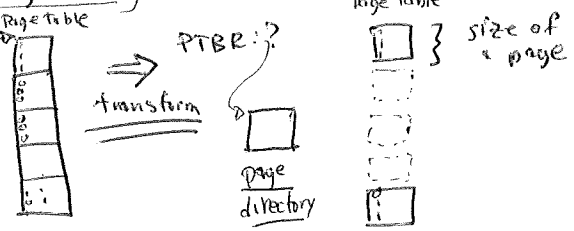


Q) How to make it smaller?

Today: a few different ways

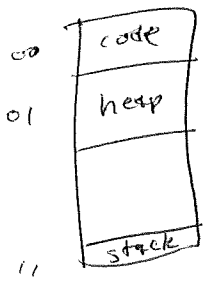
~~First~~ then: [multi-level page table]

goal: PTBR → Page Table ⇒ transform ⇒ PTBR: → page directory → Page Table (size of a page)



First: Hybrid w/ segmentation

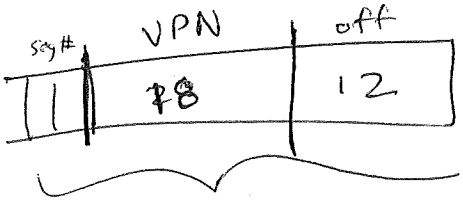
⇒ page table per segment



[Discuss: For Five Minutes]

EXAMPLE

1K AS, 16-byte page [2 byte PTE]



32-bit AS

base / bounds ⇒ holds # of (max) valid pages in segment

⇓

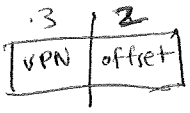
holds address of page table for this segment

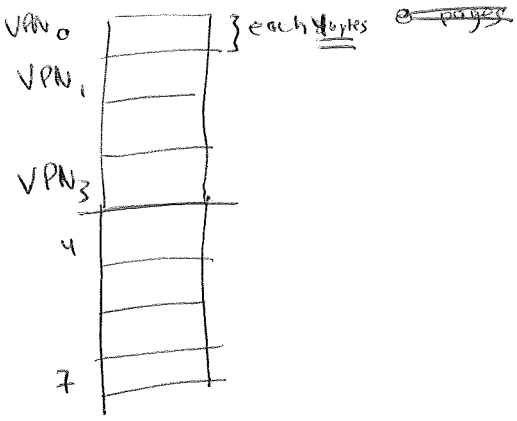
Other Ideas:
⇒ inverted
⇒ swapping pieces in/out of mem ⇒ disk

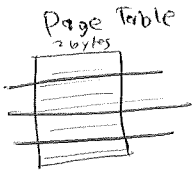
Summary:
Lots of tricks to
save space ← time (h/w TLB)
(data structures)

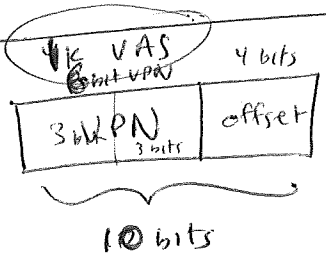
Simple Example:

32-byte AS



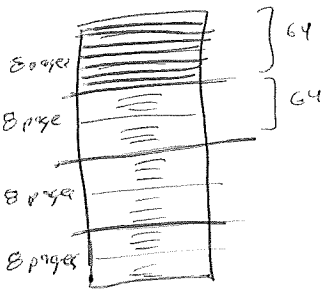






page size:
$2^4 \Rightarrow$ 16 bytes

PT

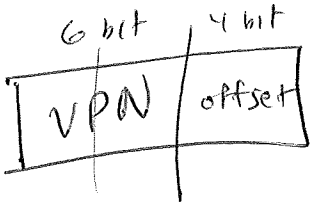


4 entries/page

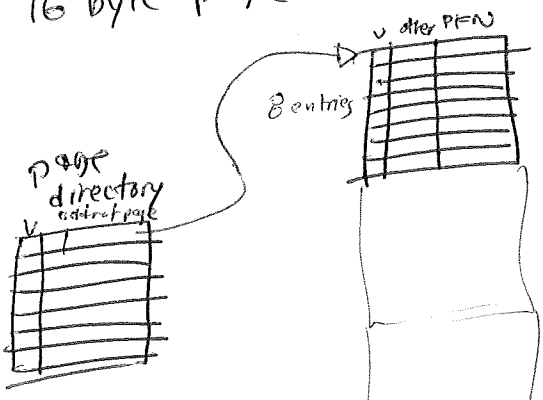
PT: 2^8 entries $\Rightarrow$ 256

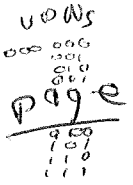
8 page

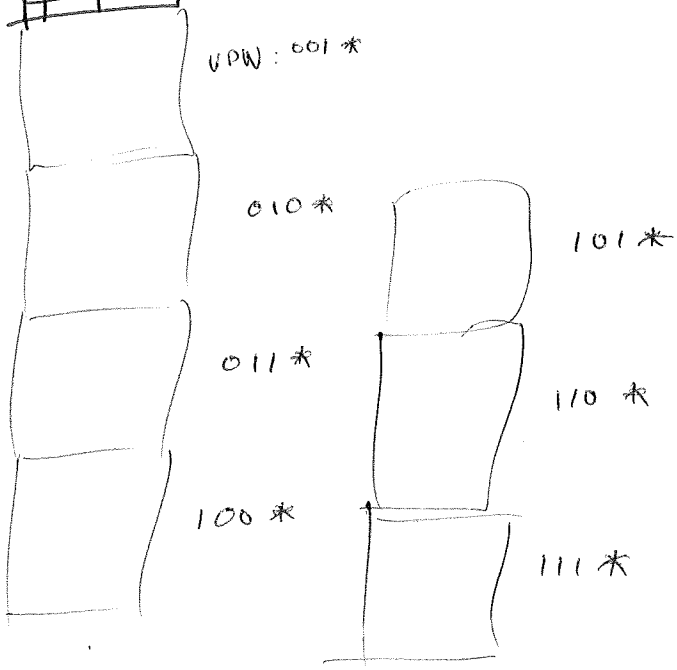
(PTE: 2 bytes)



16 byte page







64 entries
$\Rightarrow$ 64 pages