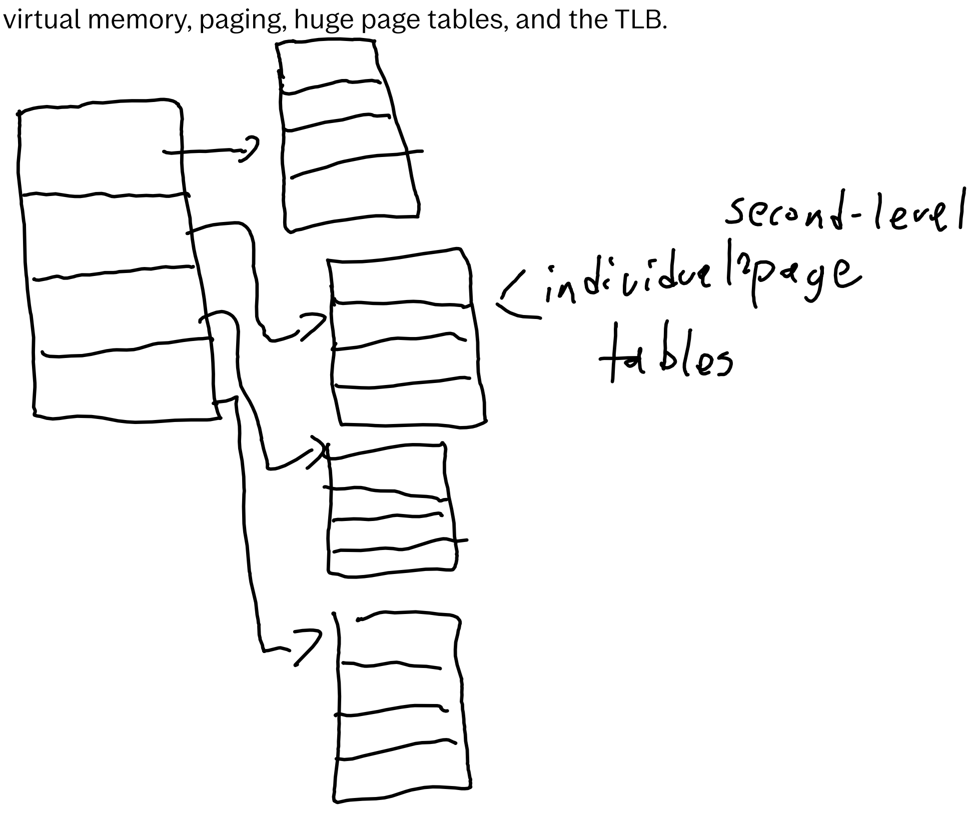
Page tables, virtual memory, and mmap

Last [few] times: virtual memory, paging, huge page tables, and the TLB.

multi-level

page table



looking at a single row of a page table, entries are:

[virtual-address; physical-address; valid-bit; present-bit; protections; clock-bit]

what do these mean?

- virtual-address: what the program sees. Address *of the page*; need to mask to lookup. (Should walk through this, with the masking, in lecture---but not right now)
- physical-address: corresponding physical page addr
- valid-bit: is this page mapped? i.e. *should* the program have access to it?
- present-bit: is this page present in physical memory, or is it on disk?
- protections: can we read? write? execute? Note: overlaps with valid a bit (in that "not valid" is "no read / write / execute)
- clock-bit: have we accessed this page since the last page fault?

Now: walk through (on the board) an address translation, with masking, for a hypothetical (all-X) address:

- mask off the page offset (just preserve page-identifying bits), search TLB
- TLB hit: done
- TLB miss:
  - search page table; in multi-level, mask off first few bits to find entry in outer table, next few bits to find entry in inner table
  - check entry:
  - valid? if no, segfault
  - present? if no, pagefault
  - protected? if yes, protfault
  - set clock-bit = 1
  - load entry into TLB
- fault: goto OS
- no fault: retry TLB access (hit)

Sidebar: what does "physical addr" mean when valid=0? when present=0?

(present=0: contains disk mapping)

segfault: program handles it (segfault handler) default: kill

pagefault: OS handles it (needs to sync in page from disk)

protfault: *either* program handles it (still segfault handler, default: kill) *or* autokill

Let's talk about MMAP!

- System call: *manually add entry to page table*
- Why do we do it? Big reason: loading from disk!
  - set "present" to 0, set "physical addr" to desired location on disk.
- Also: setting up *shared memory regions* (this is a linux thing!)
- Also: getting memory associated with a *paritcular desired* set of virtual pages!
- Finally: we know how to load programs :)

Final important thing: where is the page table?

Multi-level page tables: often in *dedicated physical memory* (its why they need to be multi-level).

Why not just ... put it in virtual memory? What could happen if we did this?

(Walk through an access to a page table in *virtual memory.*)