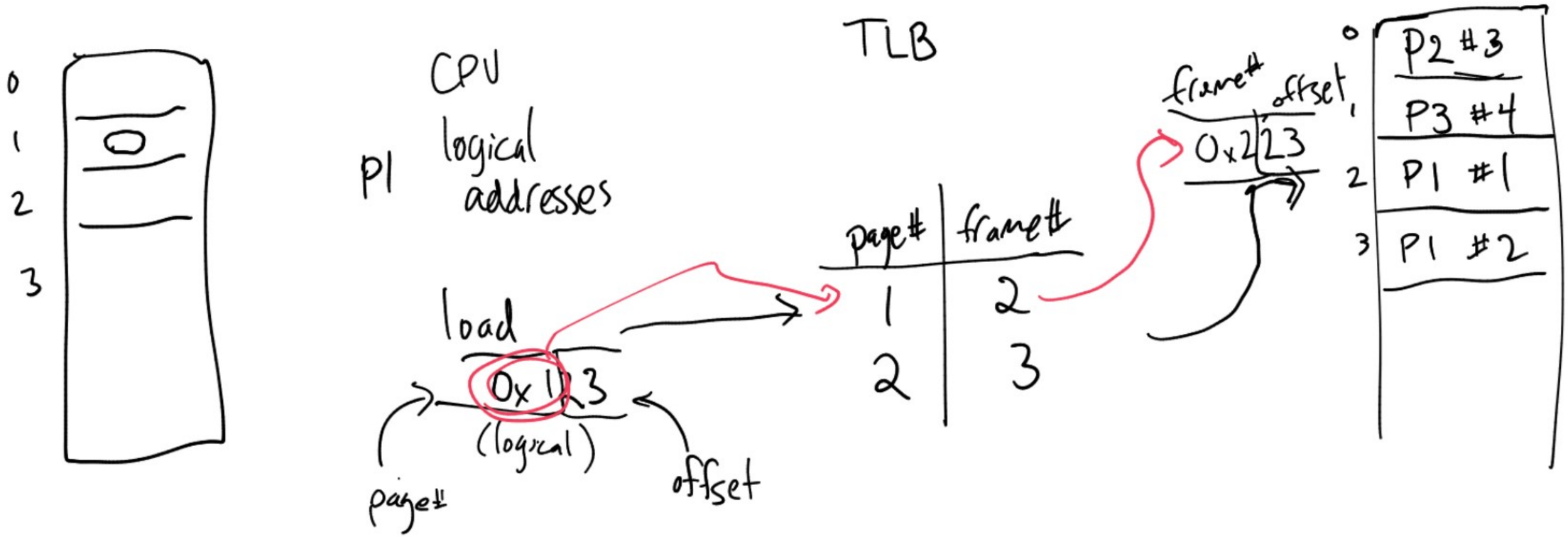
4410 Lecture 12: Page tables

- Computing sizes of data structures
- page table
- hierarchical page table
- inverted / hashed PT


0
1
2
3
CPU
P1
logical
addresses
load
0x123
(logical)
page#
offset
TLB
page#
frame#
1
2
2
3
frame#
offset
0x223
0
P2 #3
1
P3 #4
2
P1 #1
3
P1 #2

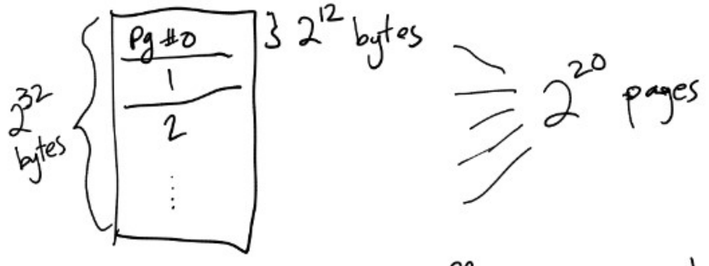
On ctxt switch, flush TLB (points to old procs pages, not new).

Tagged TLB: each entry has a process ID (3,4,5 bits) to identify process, only use entries for current process).

2^{32} addresses — 32-bit logical addr. space
$\Rightarrow 2^{32}$ bytes — 32-bit phys. "
4 kb page size

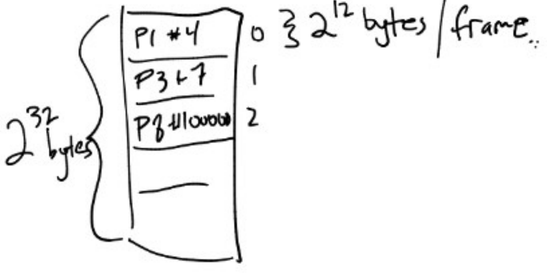
useful
bit: pointer sizes.
(large #)bytes: data sizes.
$2^{10} \approx 1000 = 1$k
$2^{20} \approx 1000$k $= 1$M (million)
$2^{30} \approx 1000$M $= 1$G (billion)
write down units.

Pl log. addr space



$\frac{2^{32}\ \text{bytes}}{\text{log. addr. space}} \cdot \frac{1\ \text{page}}{2^{12}\ \text{bytes}} = 2^{20}\ \frac{\text{page}}{\text{addr. space}}$.

phys. mem



2^{20} frames in (phys.) memory.

TLB

Page #	frame #	perms
	<20 bit frame #>	r x x v
	<20 bit frame #>	x x x x
	⋮	

1M

can't store this in TLB (hardware)

Page table (mapping page# → frame #s)
: 2^{20} entries (one per page) · $\frac{4 \text{ bytes}}{\text{entry}}$ ← frame # + perms, round up
$= 2^{24}$ bytes $= 16$ Mbytes.

TLB just contains handful of entries (e.g. 16 or 32),
raise HW exception if access page not in TLB (TLB miss)

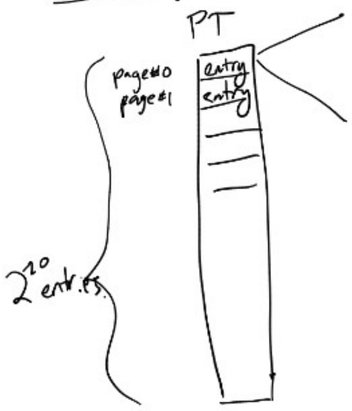
Page table stored in memory. ✭

OS, to handle TLB miss:
- look up page table (in memory),
- find frame # & permissions
- stick ↑ in TLB
- continue.

How to store page table? (mapping of page # → frame # (+perms)).

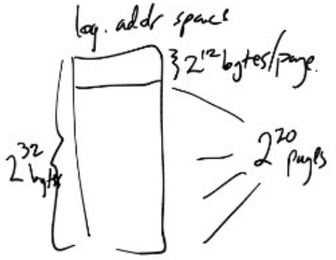
- Hashmap
- Array
- Tree

Store page table in an array?



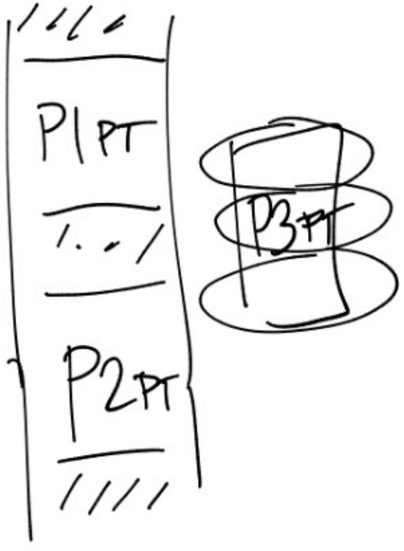
entries contain
- frame # (20 bits)
- + perms (4 bits)
- + other stuff

32 bits = 4 bytes

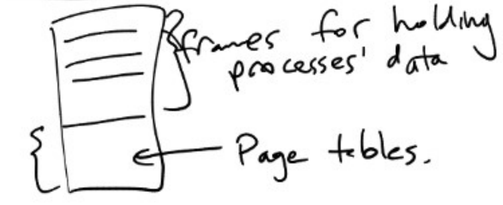


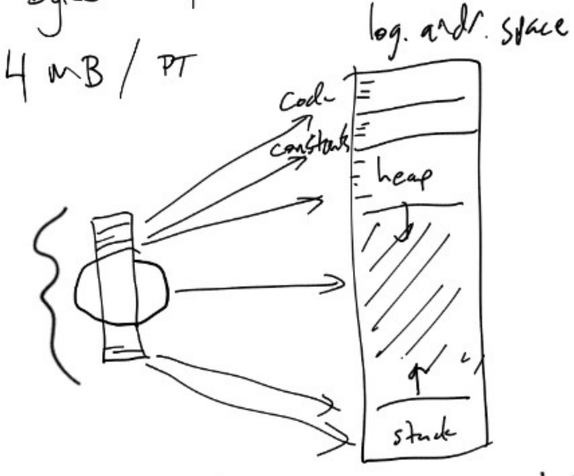
2^{20} entries $\cdot \frac{4 \text{ bytes}}{\text{entry}} = 2^{22}$ bytes in pt.

$= 4$ MB / PT

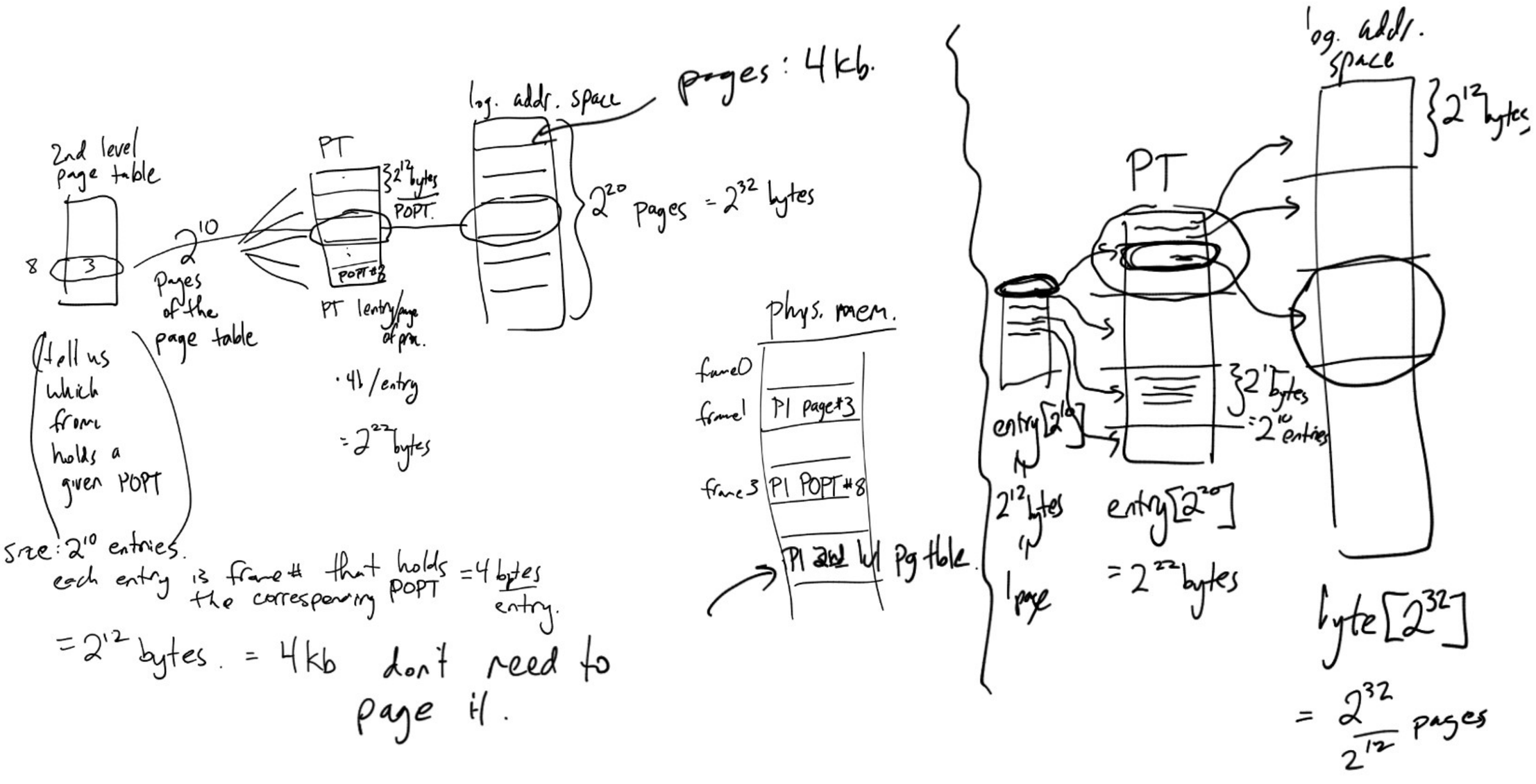


phys mem

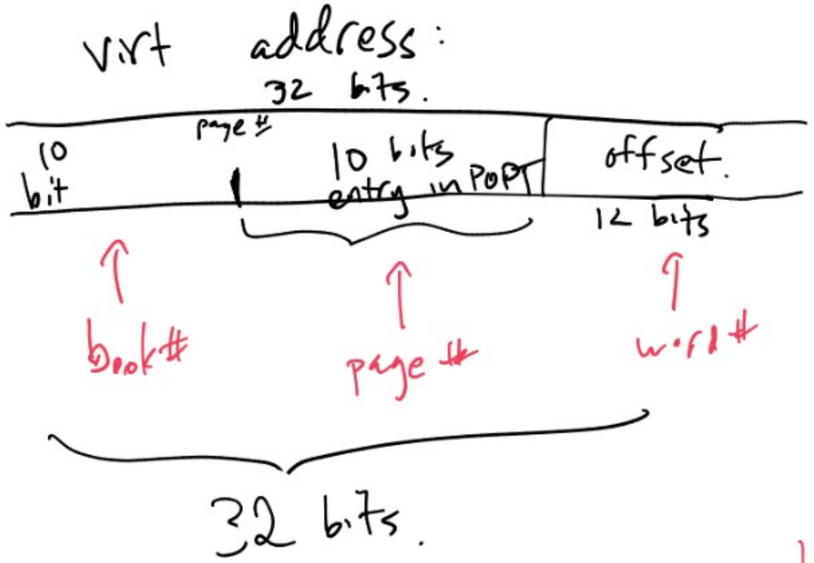


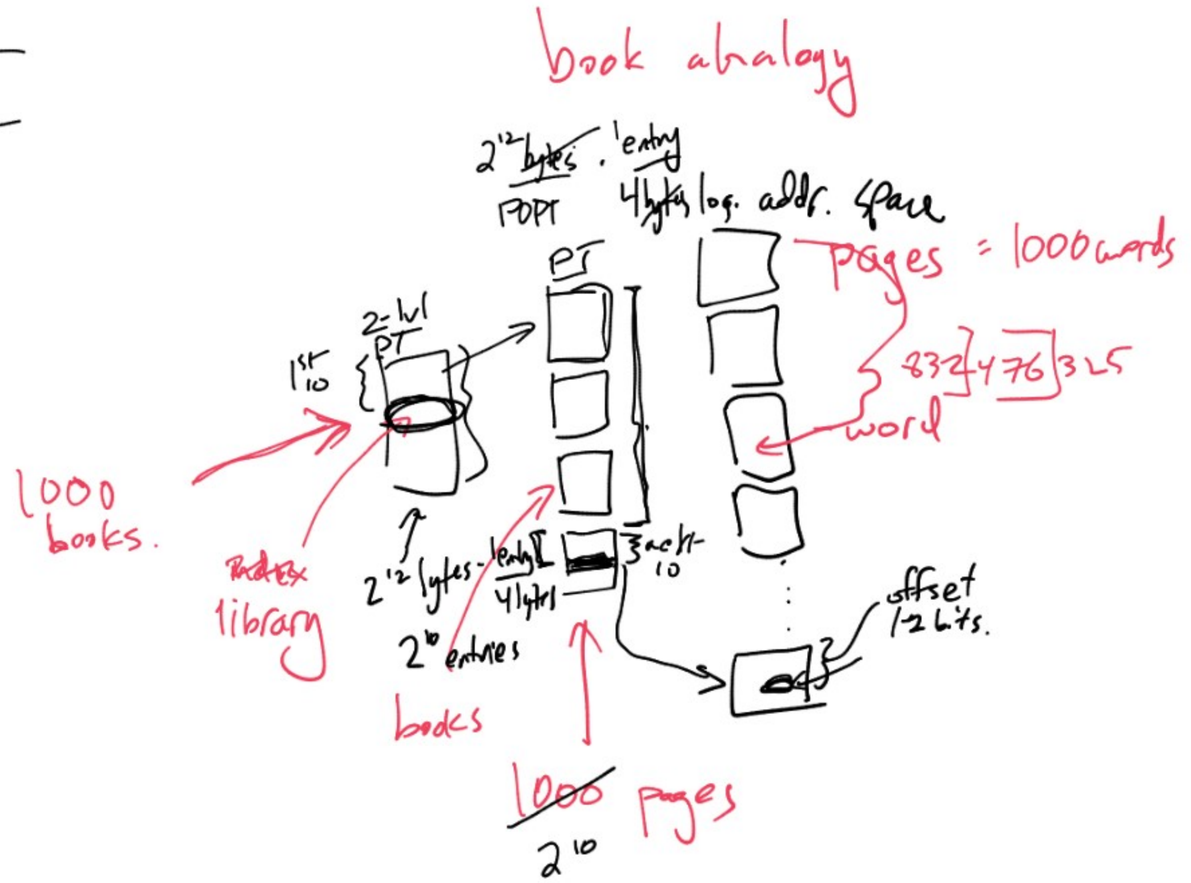


Instead of storing whole page table in one contiguous chunk, page it.


2nd level page table
8
3
2^{10} pages of the page table
PT
2^{12} bytes
POPT
POPT #3
PT entry/page of PTs
log. addr. space
pages: 4kb.
2^{20} pages = 2^{32} bytes
tell us which frame holds a given POPT
size: 2^{10} entries.
each entry is frame# that holds the corresponding POPT
= 4 bytes/entry.
= 2^{12} bytes. = 4kb don't need to page it.
· 4b/entry
= 2^{22} bytes
phys. mem.
frame0
frame1
P1 page#3
frame3
P1 POPT#8
P1 2nd lvl pg table.
log. addr. space
2^{12} bytes
PT
entry[2^{10}]
2^{12} bytes/page
2^{10} entries
entry[2^{20}]
= 2^{22} bytes
byte[2^{32}]
= $\frac{2^{32}}{2^{12}}$ pages

How to find page (phys. address) w/ 2-lvl page table.





OS algorithm for handling TLB miss. (to address a)

- look at first 10 bits of a, index into top level page table.
  ⇒ frame # holds correct POPT
- use next 10 bits of a to index into POPT.
  ⇒ frame # holds correct page.
- load that frame # (f.perms) into TLB.
- restart process just before access
  (process will access a again, TLB look@ last 12 bits to offset into frame)