Today (2/20)

[537]

Virtual Memory ✓

⇒ Page Tables

array-based
called "linear"
P.T.

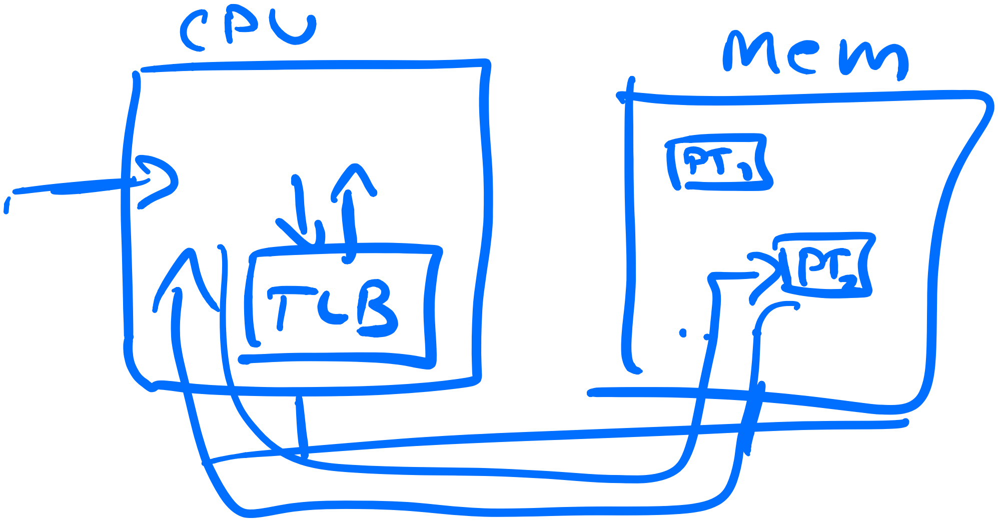
→ Problems

→ Paging: too slow

⇒ extra memory
ref ⇒ Page Table

solution: TLB

"address translation
cache"



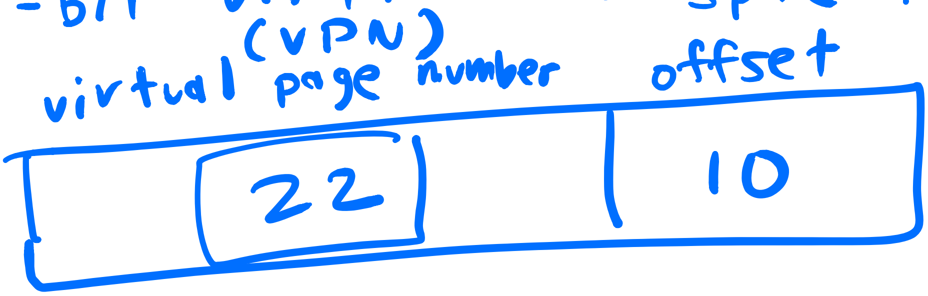
=> Problem #2: Page Tables too big

=> Problem #3: what if amt mem. accessed by prog > amt. of phys mem?

=> use "disk": [slower, larger Storage medium]

Problem #2: Page Tables (linear) are too big

32-bit virt. addr space:



Page size: 1 KB

=> how big is the page

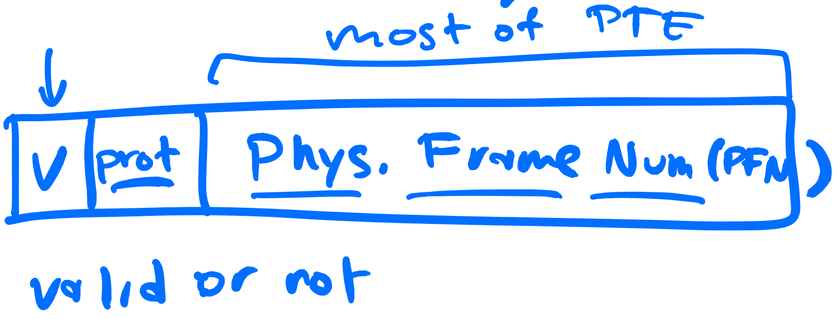
=> how big is the page table?

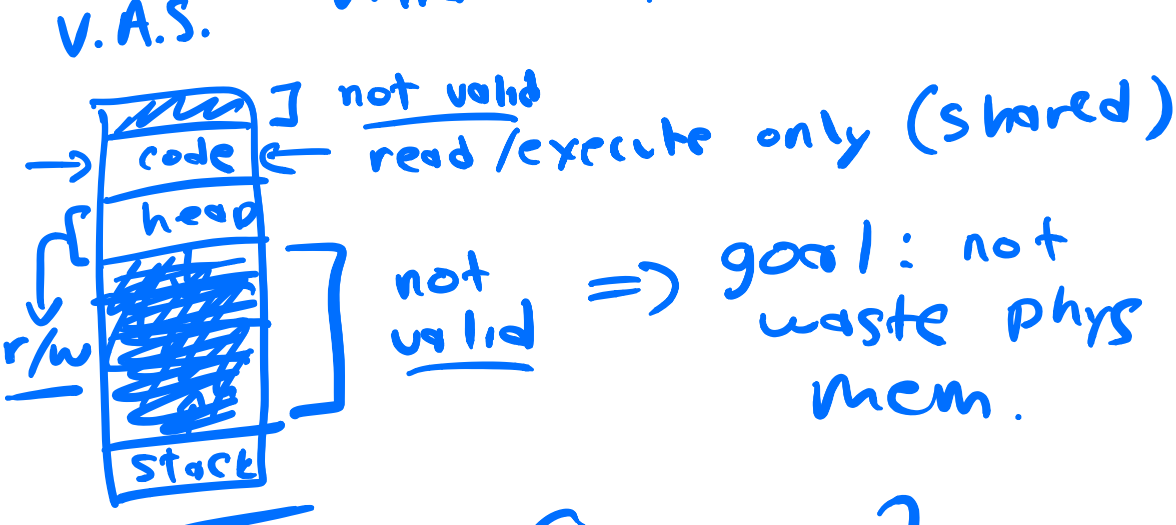
array: [one entry per VPN]

pages: 2^{22} ~ 4M

=> about 4 million entries

Page Table Entry (PTE)





=> goal: not waste phys mem.

PTE: [4 bytes]

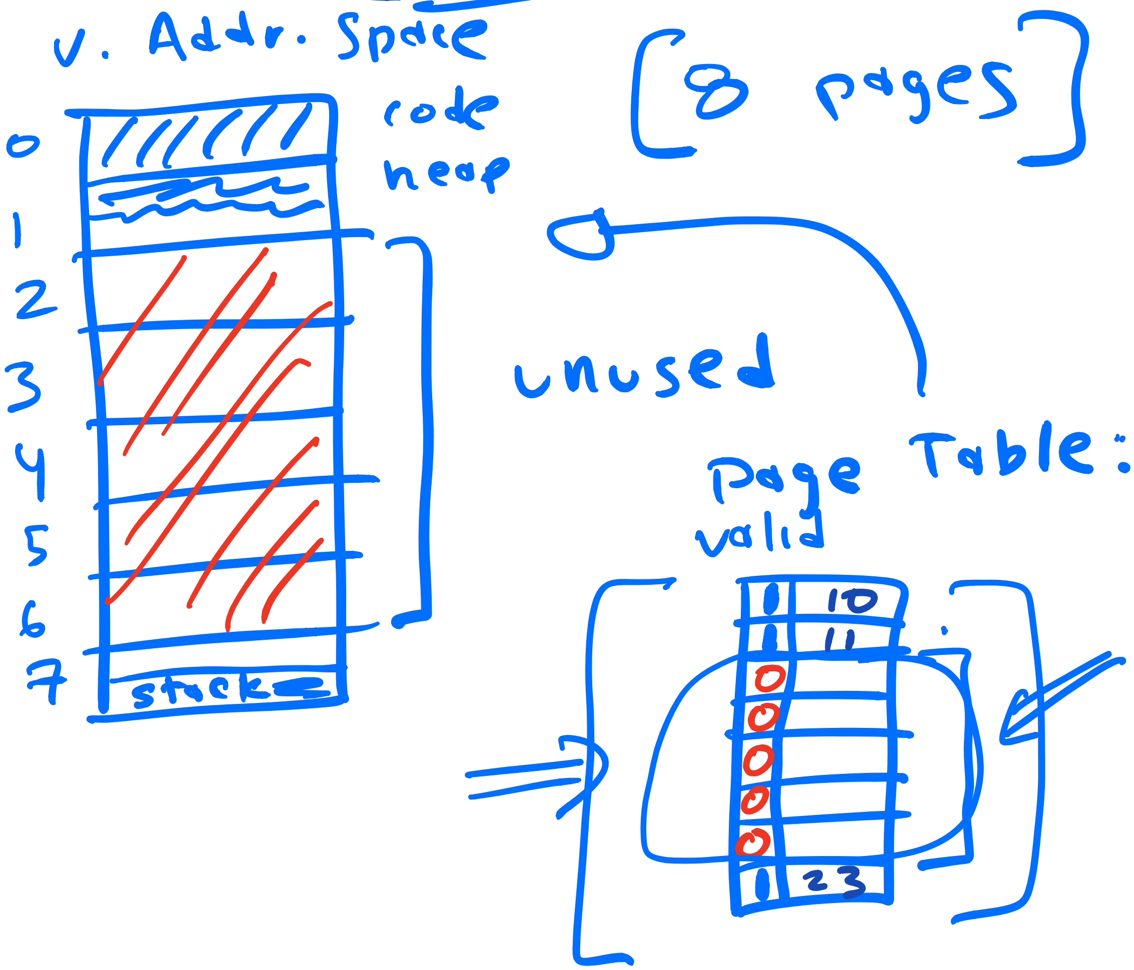
Size (Page Table): ~4M * 4b

=> ~16 MB

one Page Table / Process:

assume 100 processes

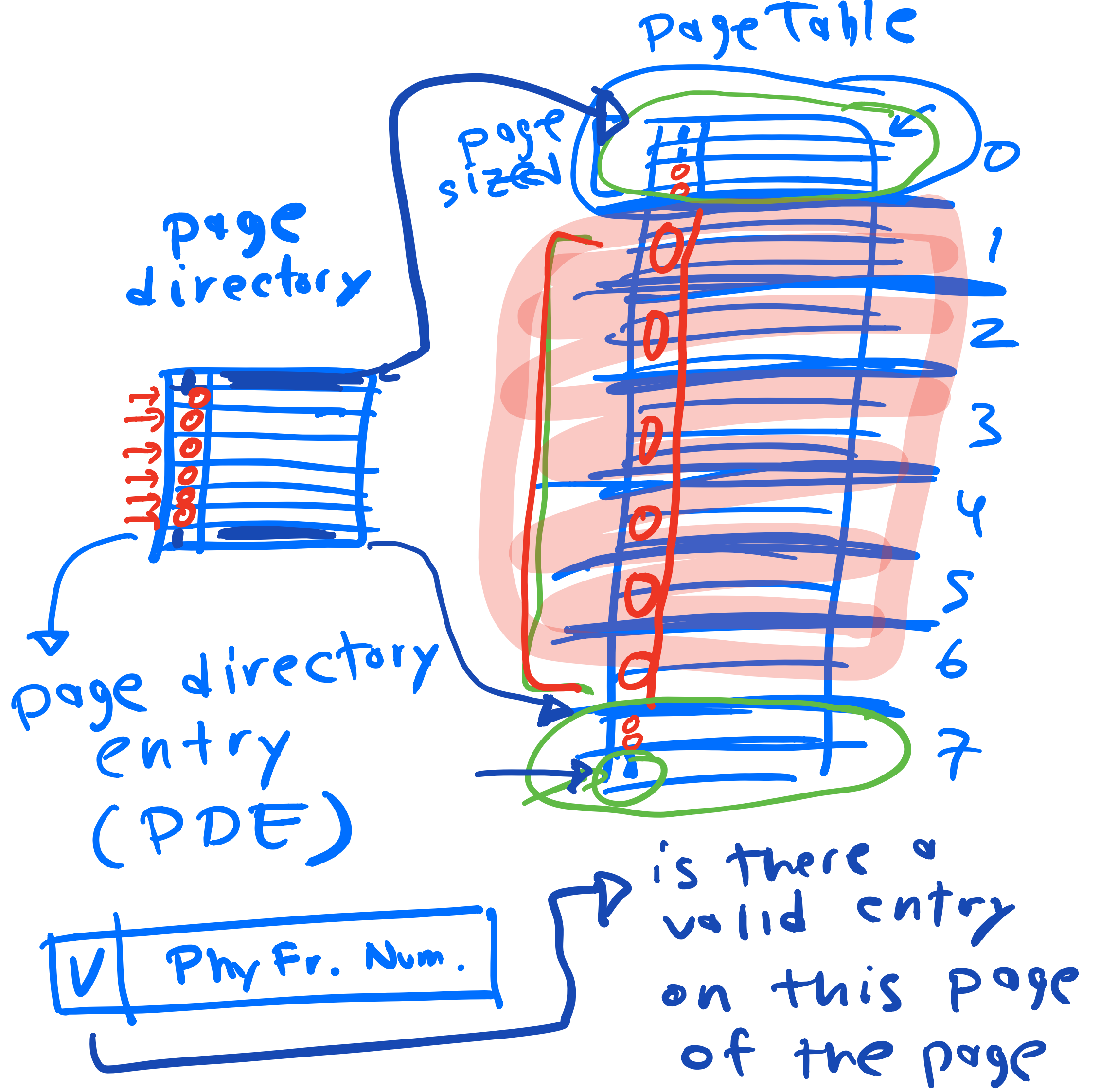
=> $\boxed{1600 \text{ MB}}$ of PTs(!)



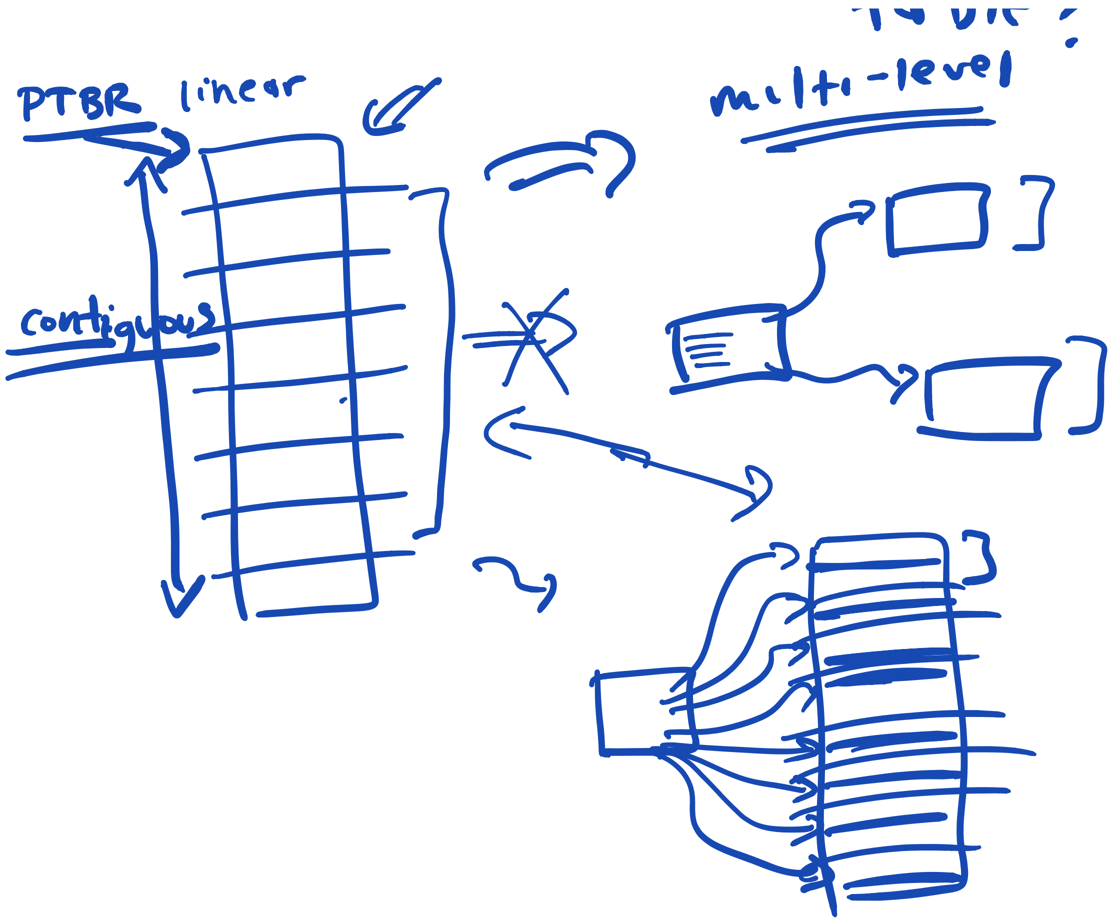
Solutions:

- → [Multi-level Page Table]
- → Hybrid: (segmentation + Paging)
- → Inverted : one per system

▷ [just data structures]



is there a valid entry on this page of the page table?

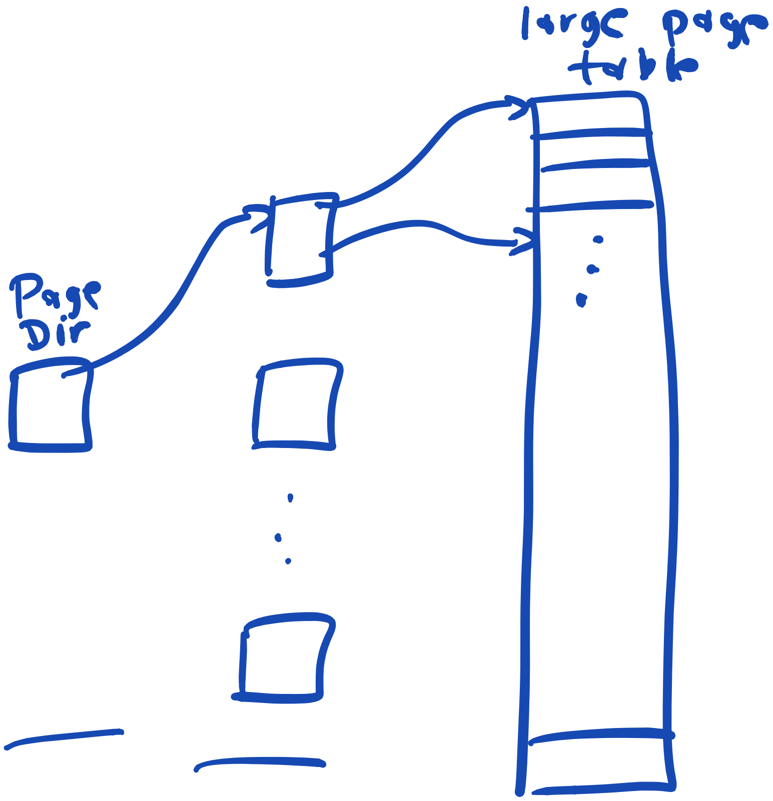


Q) why is OS mgmt of phys memory easier w/ multi-level PTs? (vs. linear?)

before

trade off: space vs. time

=) M.L. P.T. (slower but smaller)



Point of Discussion:

Best Broadway Musical?

HAMILTON

~~Sponge Bob the Musical~~

Book of Mormon

Cats

Lion King

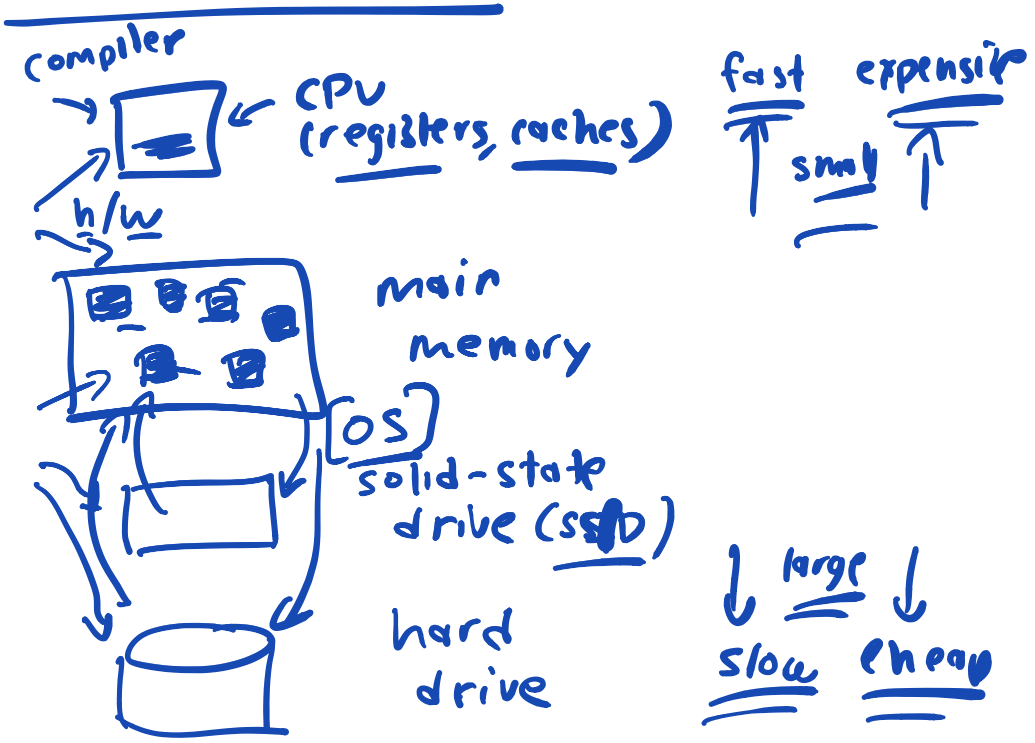
Wicked

Phantom of Opera

=> [Read book, do homework problem on own]

Problem #3: virt. mem accessed by programs > phys mem of system

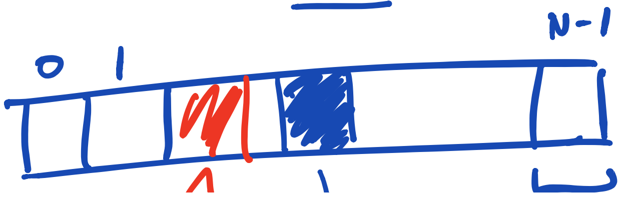
Memory Hierarchy:



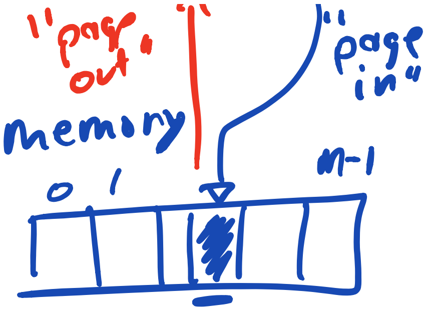
Mechanisms

Policy

-> swap space (on disk)



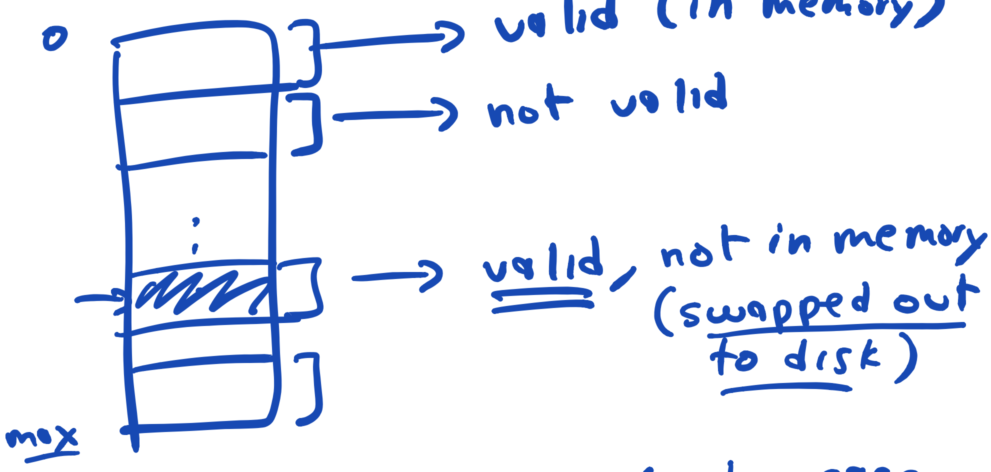
larger, slower



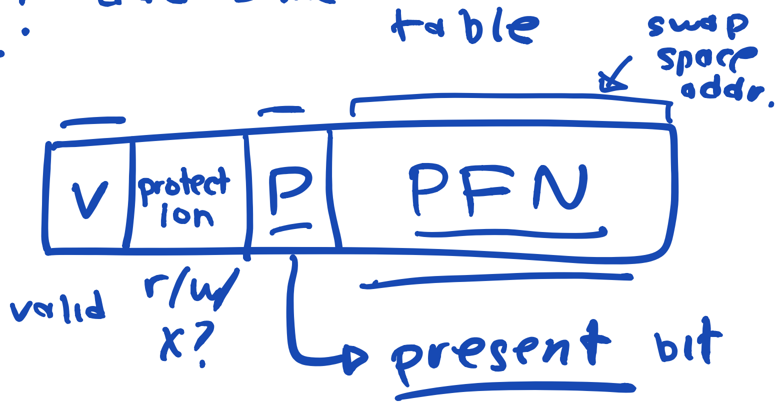
blocks => page sized (4 KB)

[M < N]

Virt Addr Space



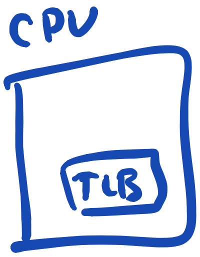
Track? add some info to page table



V=1, P=1 => valid, PFN is location

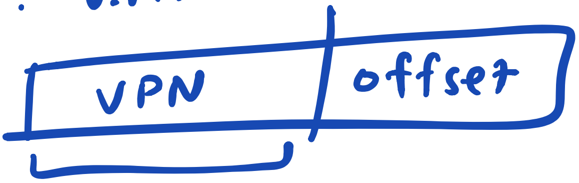
V=1, P=0 => valid, page on disk somewhere

=> page miss
(page fault)



every mem. ref :

H/W : V.A. virt. addr.



consults TLB :

TLB hit : [PFN | offset]
=> access memory

TLB miss :
=> Fetch PTE
=> extract PTE
is valid ? NOT => seg. Fault
is prot ok ? NOT => prot fault
is present? NOT => page fault

OS : page fault handler

=> Find free phys. frame

if NOT,

page/swap out existing page

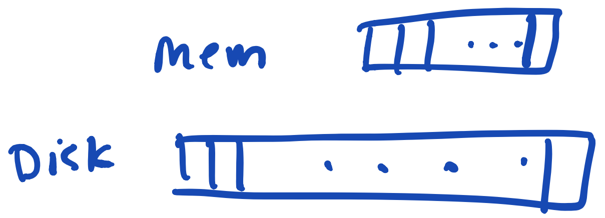
=> [page replacement policy]

=> bring in desired page,

update PTE => present=1,
PFN = ____

(retry inst)

Policy:



Page Replacement:

when OS needs to kick out page, which one?

=> Least Recently Used (LRU)

=> Frequently (LFU)

"better"
harder to build

=> Most Recently Used (MRU)

=> Random

=> FIFO

easy to implement

(Use History to predict future)

Comparison: <u>Optimal</u> <u>Policy</u>

⇒ kick out the page that will be used furthest in <u>future</u> →

<u>Example:</u> mem size : <u>3</u> (pages)

hit or miss? what's in memory?

page

	1	1	2	0	1	3	0	1	2	0
FIFO	m	h	m	m	h	m	h	m	m	m
	1	1	2 1	0,2,1		3,0,2		1,3,0	2,1,3	
LRU-				1	~~2~~	0	1	~~3~~	0	1
			1	2	0	1	3	0	1	2
MRU-	1	1	2	0	1	3	0	1	2	0
hit or misss	m	h	m	m	h	m	h	h	m	h
Opt?		_			_		_	_		_

How to implement "approximate" LRU ?

⇒ why approximate?

LRU requires knowledge of access time of each page

=> timestamp / page

=> h/w updates on each mem access

=> replacement : (**LRU**) search, finding **Least** R.U. page (**SLOW**)

approximate LRU:
"not recently used"

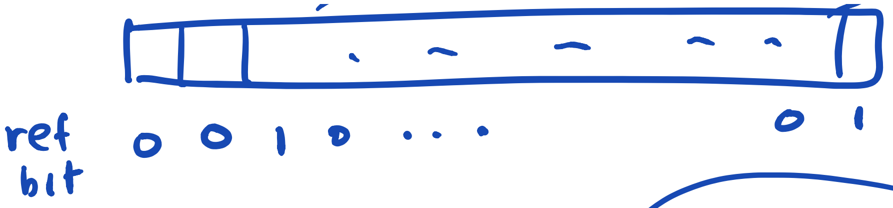
h/w: when page accessed, sets **bit** => (**reference bit**)

OS: replacement

-> scans through pages

if ref bit = 0, **kick out** : replace

if ref bit = 1, set ref bit to 0
cont. scanning



1) Technical Piece

2) Admin.

=> [p2b]

→ last solo

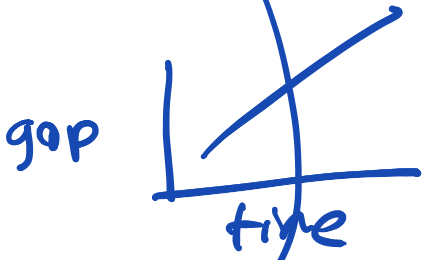
→ [video] ⇐

Problem w/ swapping:

"Disk" too slow

mem: nanoseconds

hard drive: milliseconds



Solution: Buy more memory
(don't swap)

More memory always lead to better "hit rate"?

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

$FIFO_3$

FIFO_4

↗

Belady's Anomaly

Some policies, no problem:

LRU: LRU_4 contains LRU_3