Operating Systems!

Memory: Approaches to Memory Management (Part 6)

Prof. Travis Peters

Montana State University

CS 460 - Operating Systems

Fall 2020

https://www.cs.montana.edu/cs460

Some diagrams and notes used in this slide deck have been adapted from Sean Smith's OS courses @ Dartmouth. Thanks, Sean!

Today

- Announcements
  - ~~Yalnix~~ How are projects/proposals going?! ☺
  - *Congrats on submitting PA3!*

- Upcoming Deadlines
  - **Sunday [10/25/2020] @ 11:59 PM (MST)** - Project Proposal

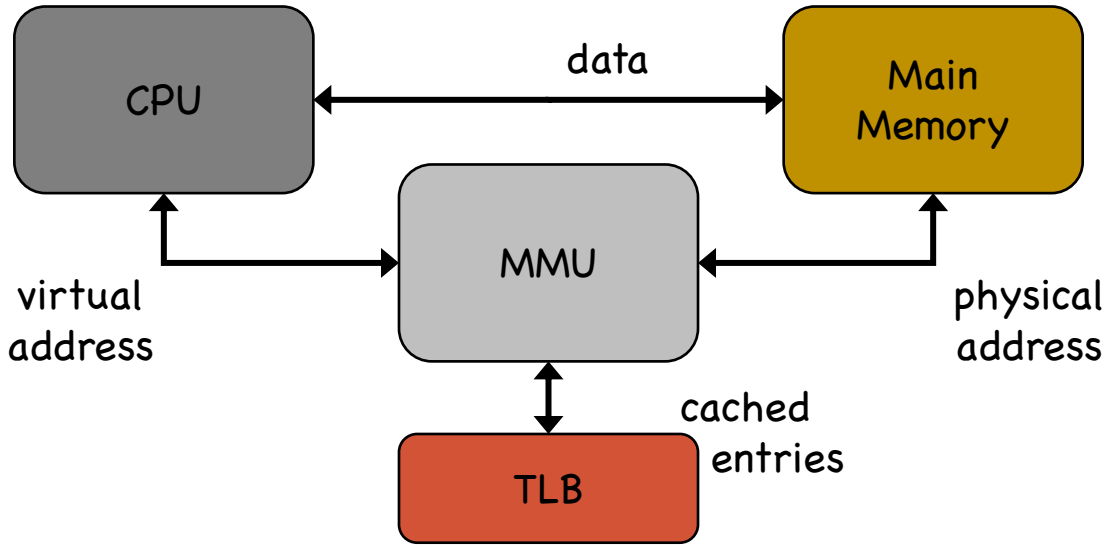
Yalnix teams:
you should plan to be near completing checkpoint 3 at this point!

Non-Yalnix teams:
let me know if you want to discuss projects before submitting your proposal!

Today *(cont.)*

- Agenda
  - ~~Page Tables~~
  - ~~Heaps~~ → ~~review fault.c 3,4~~
  - ~~Stacks~~ → ~~(quickly) review fault.c 5,6,7 + sf.c~~
  - ~~Page Tables (Practical Issues)~~
  - ~~The TLB~~
  - ~~Switching Processes & the TLB~~
  - ~~Revisiting Syscalls~~
  - Memory Variations
  - Paging to Disk

  ➔ Virtual Memory

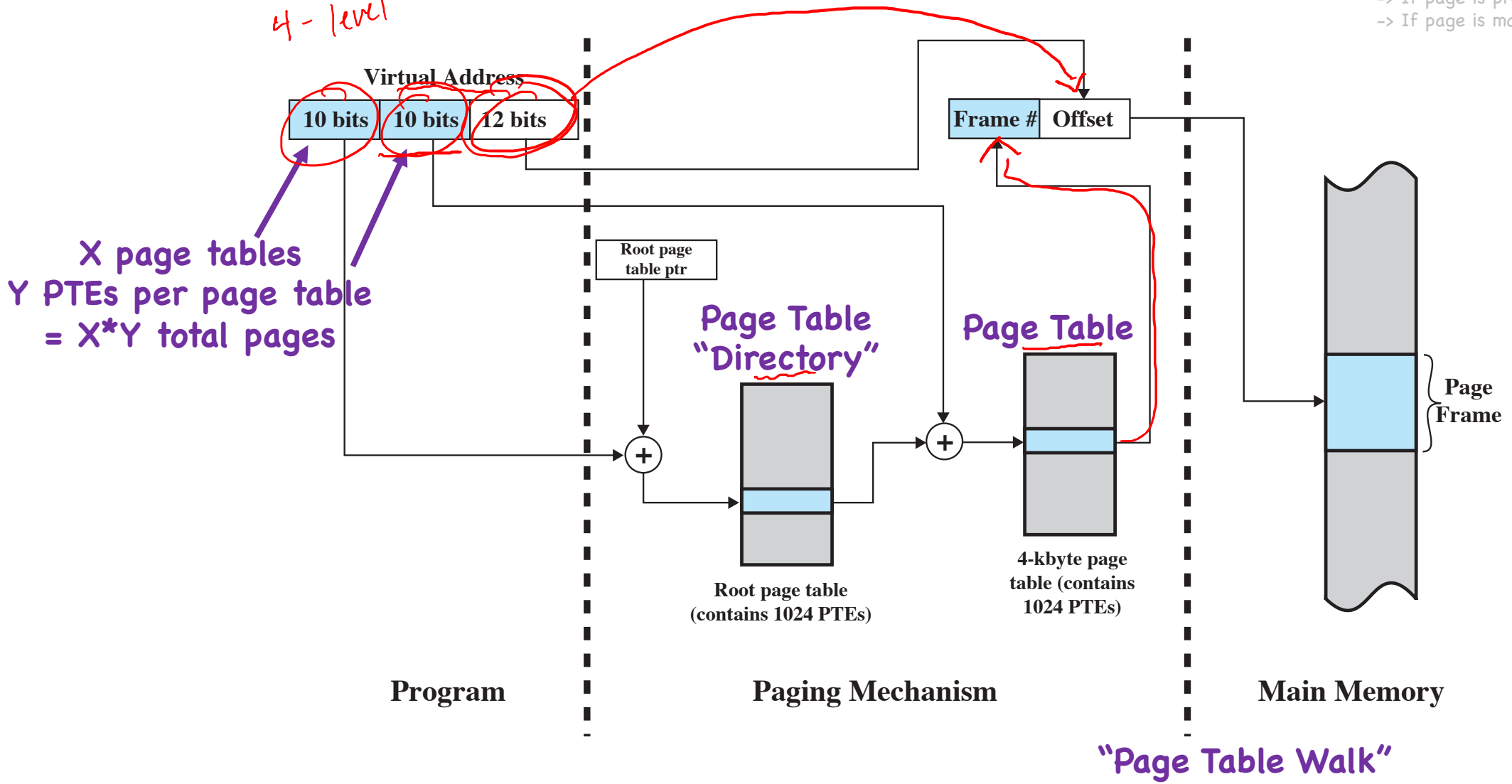


Memory Variations

A quick look at other variations in memory management

Recall: 2-Level Paging System

1 - Level
4 - Level



Virtual Address

Page Number	Offset

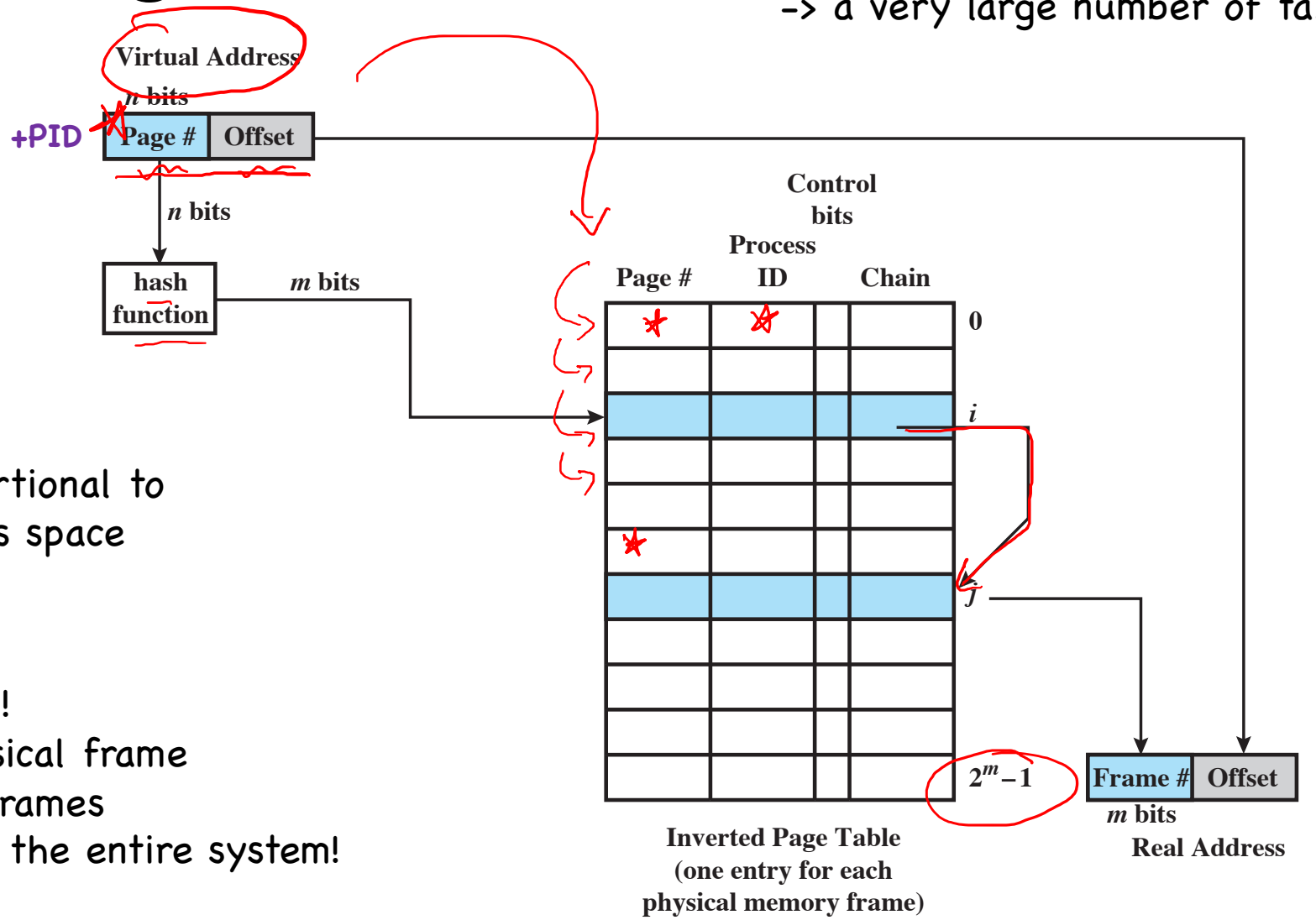
Page Table Entry

P	M	Other Control Bits	Frame Number

-> If page is present (P), use frame number
-> If page is modified (M), write out

Inverted Page Tables

Traditional Paging Systems
One slot for each virtual page, used or not
-> a very large number of table entries!



Problem
Page table size proportional to size of virtual address space

Idea!
Invert the page table!
-> one entry per physical frame
-> index by physical frames
-> one page table for the entire system!

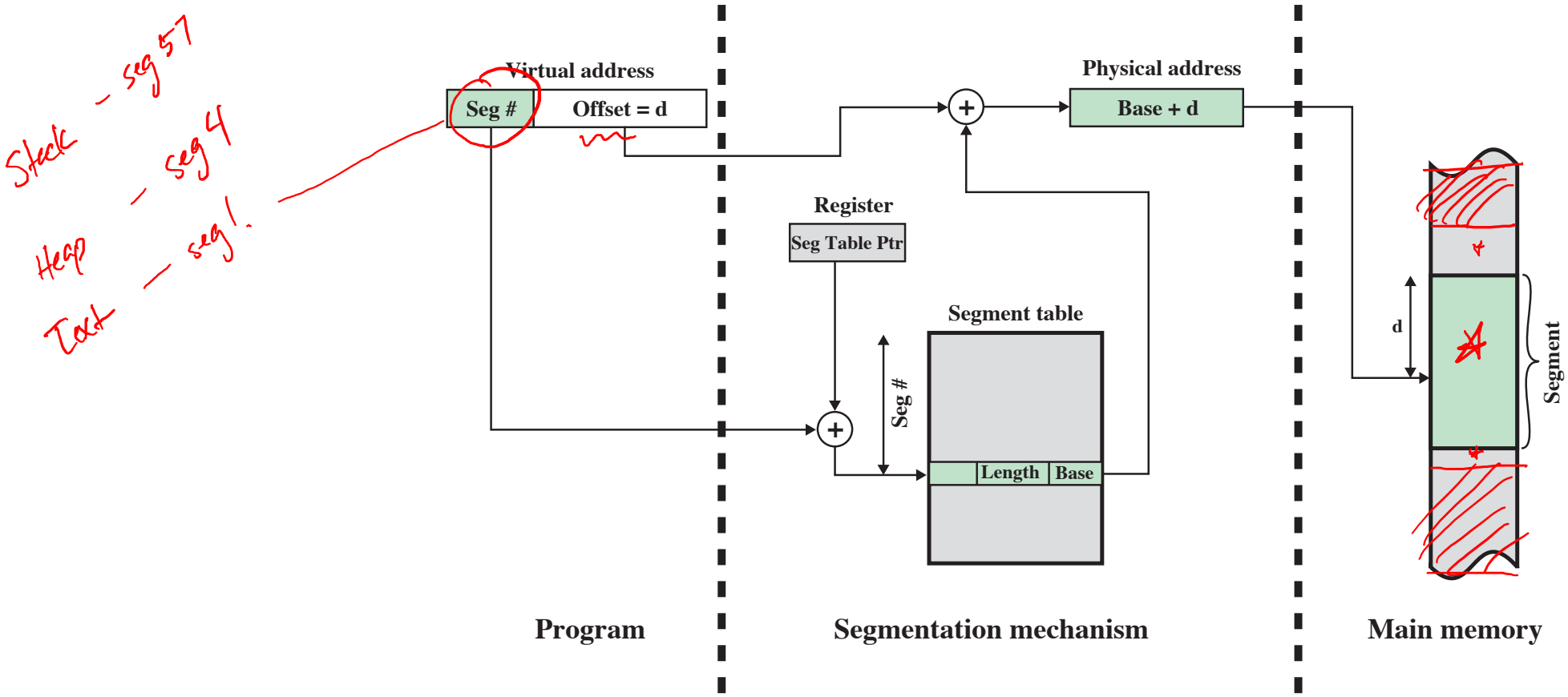
Segmentation

Virtual Address

Segment Table Entry

-> If seg. is present (P), use seg. number
-> If seg. is modified (M), write out

Observation:
Programmers don't think about "pages" and "frames"...
... they think about regions (**segments**) such as "code" and "stack" and "heap"



Paging & Segmentation

Virtual Address

Segment Number	Page Number	Offset

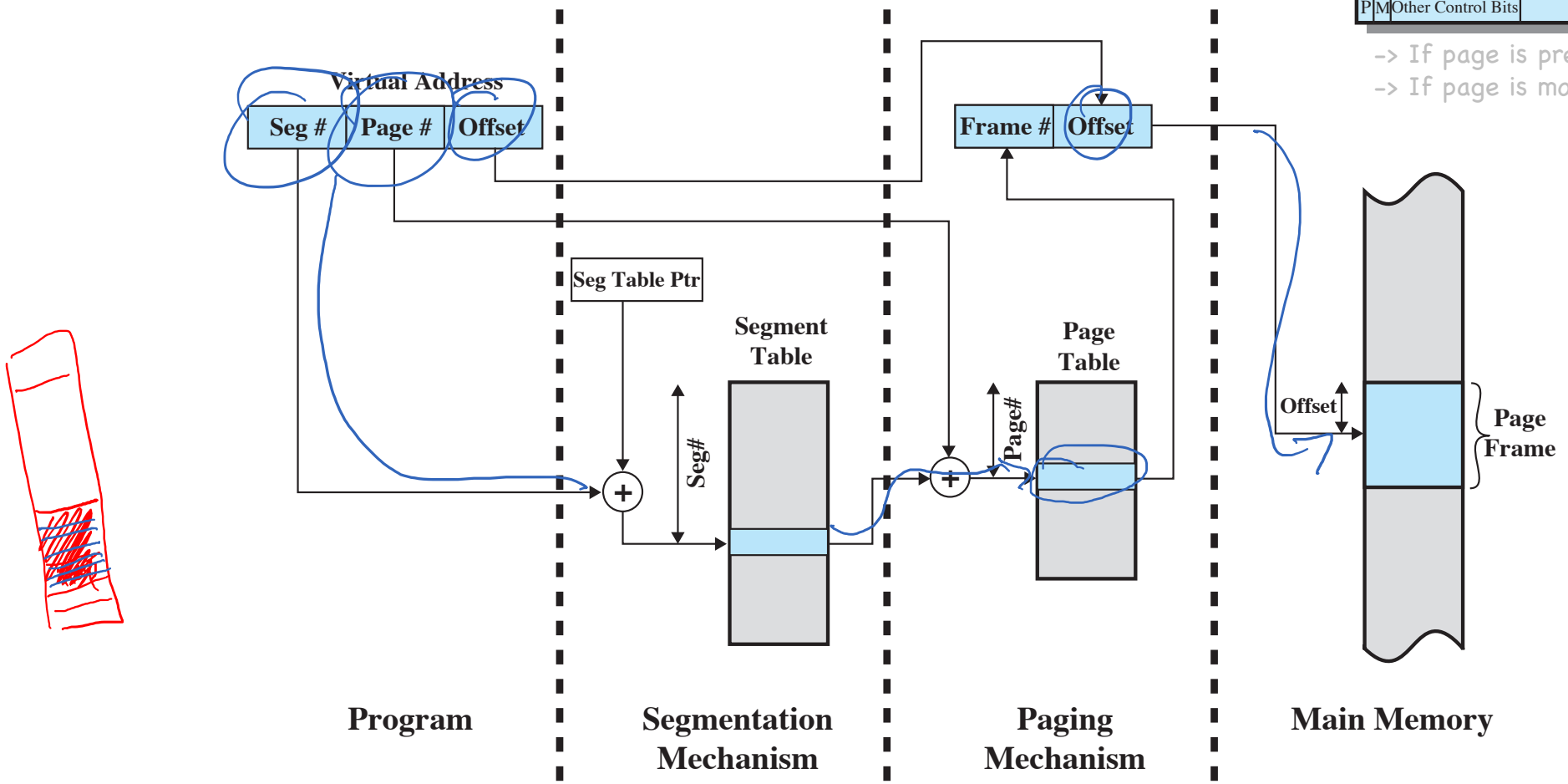
Segment Table Entry

Control Bits	Length	Segment Base

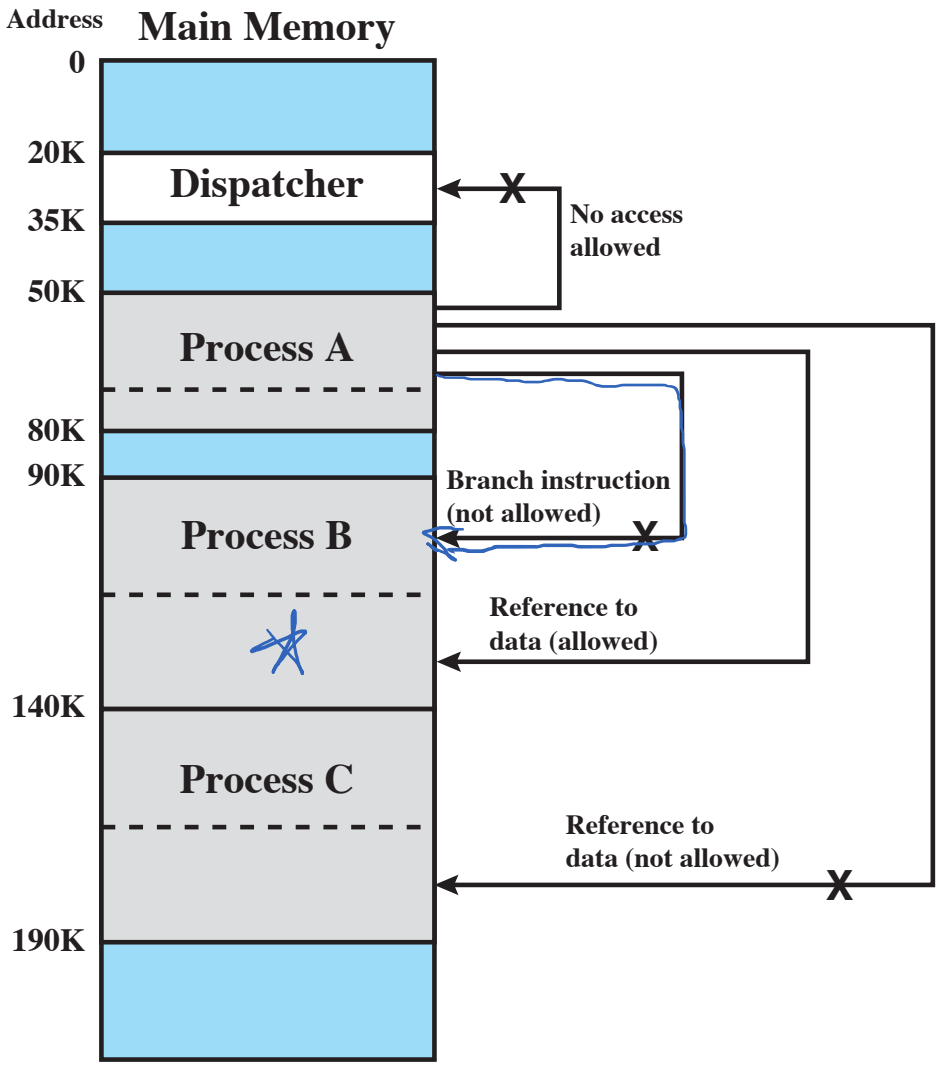
Page Table Entry

P	M	Other Control Bits	Frame Number

-> If page is present (P), use frame number
-> If page is modified (M), write out

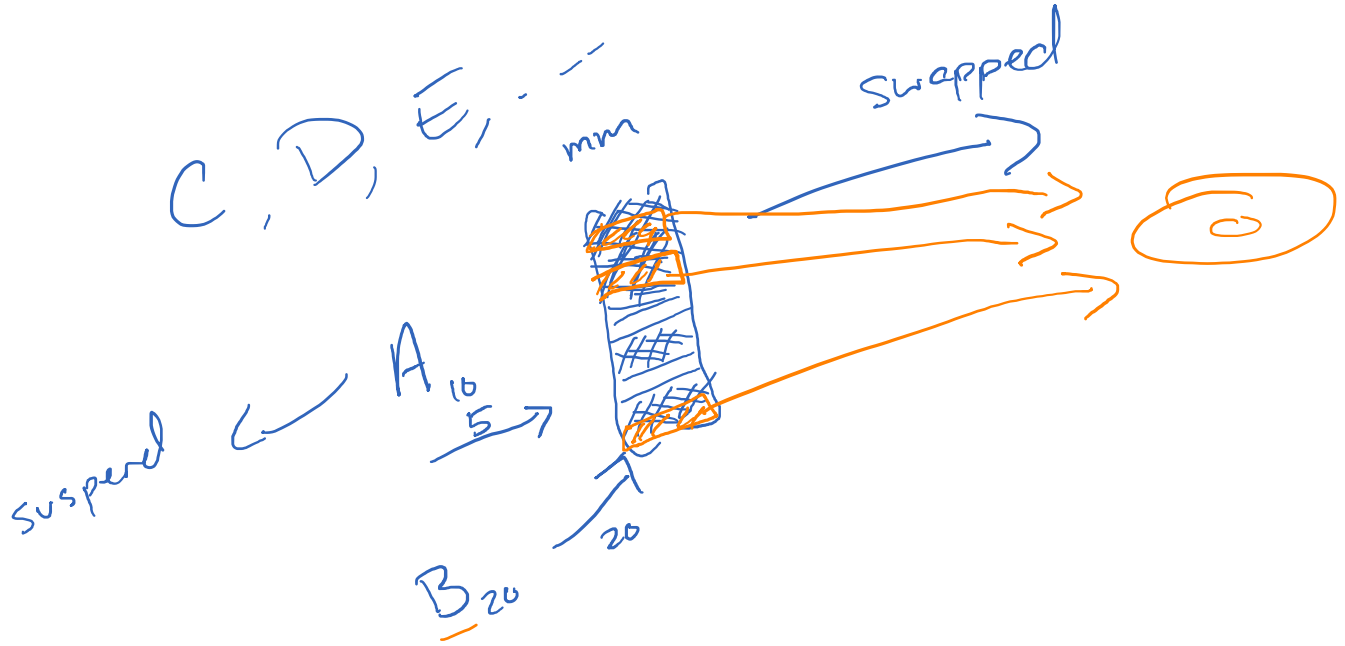


Paging & Segmentation *(cont.)*



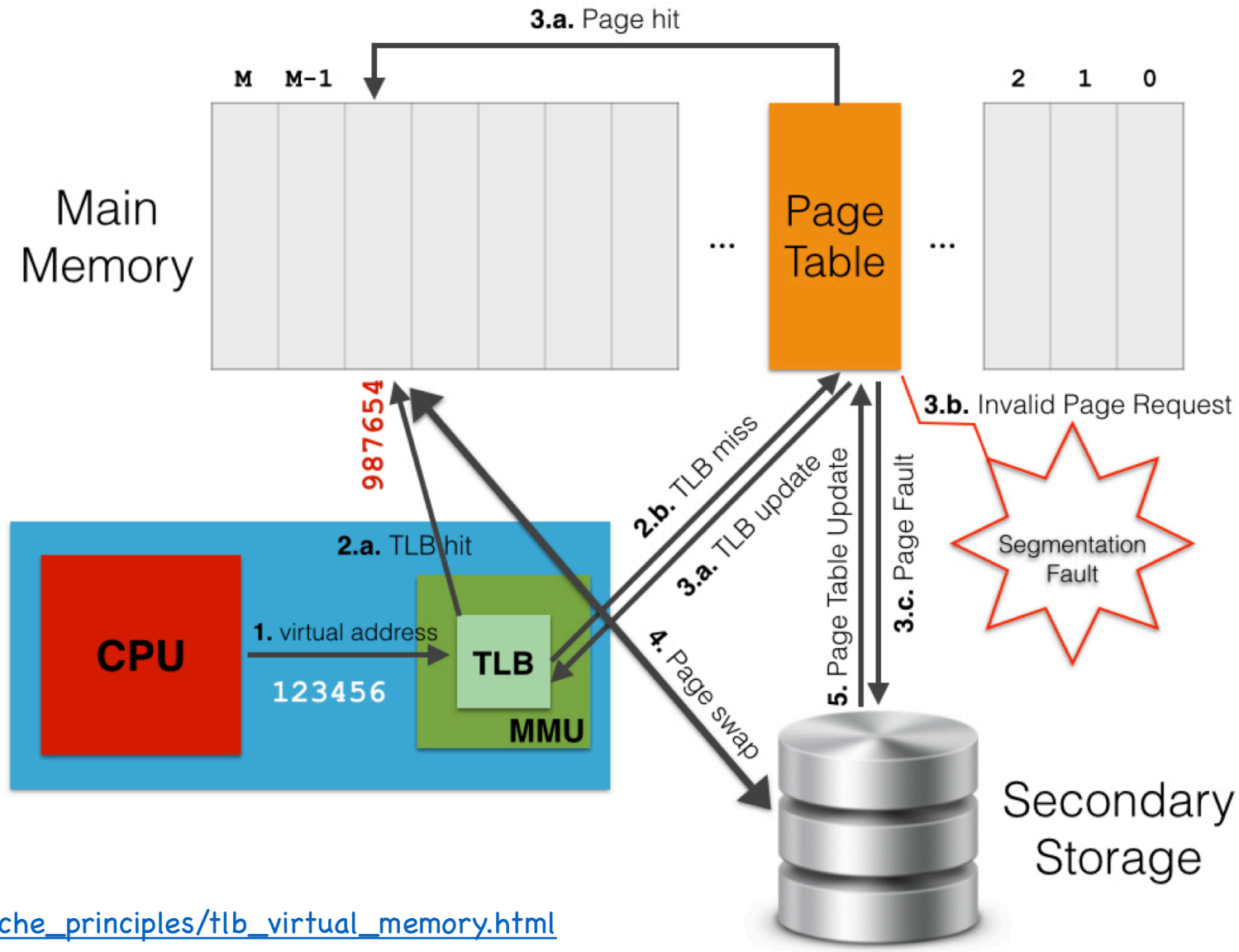
-> Configure settings for **segments** (r/w/x)

-> Sure we can do this on a per-page level, but it is less intuitive



Paging to Disk

Paging to Disk



Further Reading:
http://alasir.com/articles/cache_principles/tlb_virtual_memory.html

https://gabrieletolomei.files.wordpress.com/2013/10/mmu.jpg

Designing an OS with Virtual Memory

- **Goal:** Reduce Frequency of Page Faults

- *Why? What's the cost?*

r/w to/from disk

Designing an OS with Virtual Memory

- **Goal:** Reduce Frequency of Page Faults

- *Why? What's the cost?*
  - Must make decision for which page(s) to swap in/out + associated I/O
  - Process switch

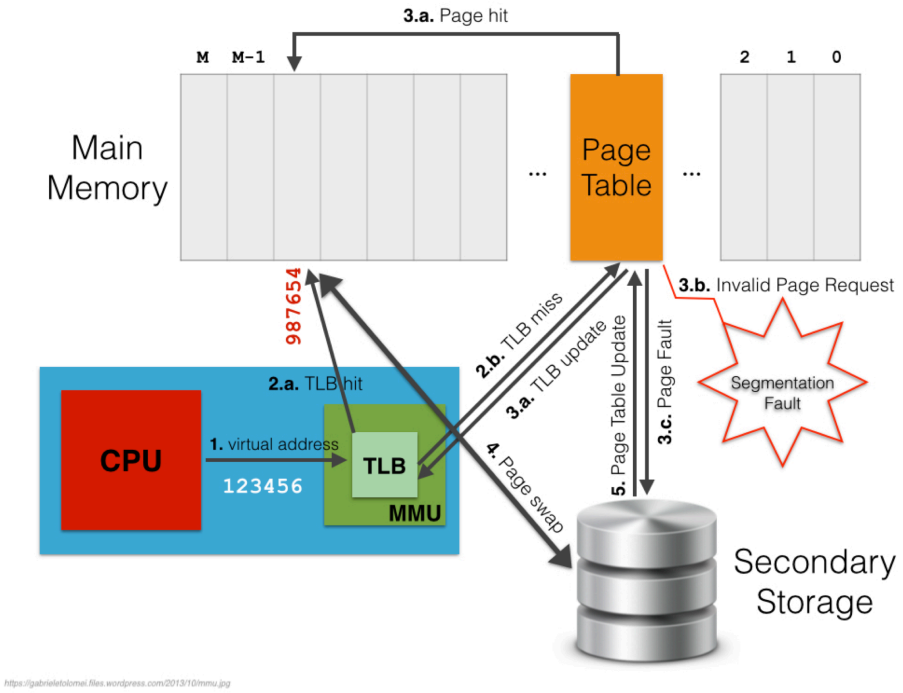
What issues must we consider to address this goal?

Fetch Policy

When should a page be brought into main memory?

- Demand Paging
  - *Bring pages in when a referenced*
  - *as needed / on demand*

- Pre-paging
  - *Bring in several pages at once*
  - *Ex: other nearby pages on disk*



Placement Policy

Where should a page be put in main memory?

- Segmentation?
  - Best-Fit, First-Fit, Etc.

- Paging?
  - *Not really an an issue: page-frame mappings are equally efficient*
  - *Just need a free frame...*

Replacement Policy

Which page should be replaced when a new page must be brought into main memory**?*

- Strategies:
  - Optimal (OPT) --- sometimes called "MIN"
  - LRU
  - FIFO
  - Clock ("2nd Chance")
  - + Random Placement, ...

** Want to replace an **"unlocked"** page that is least likely to be referenced in the near future.*
***Most interesting if we assume that all frames in MM are occupied, then a page fault occurs.*

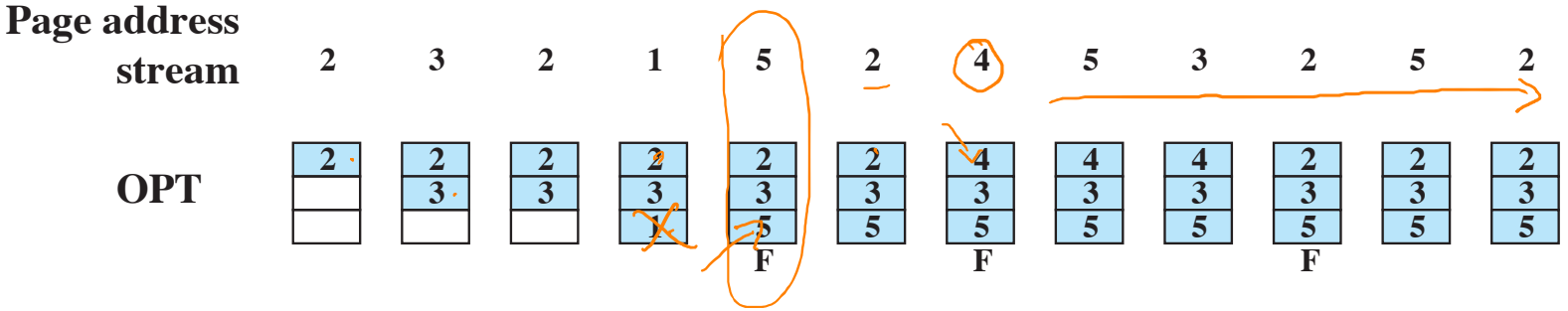
Example: Page-Replacement Algorithms

Reference 5 distinct pages…

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2

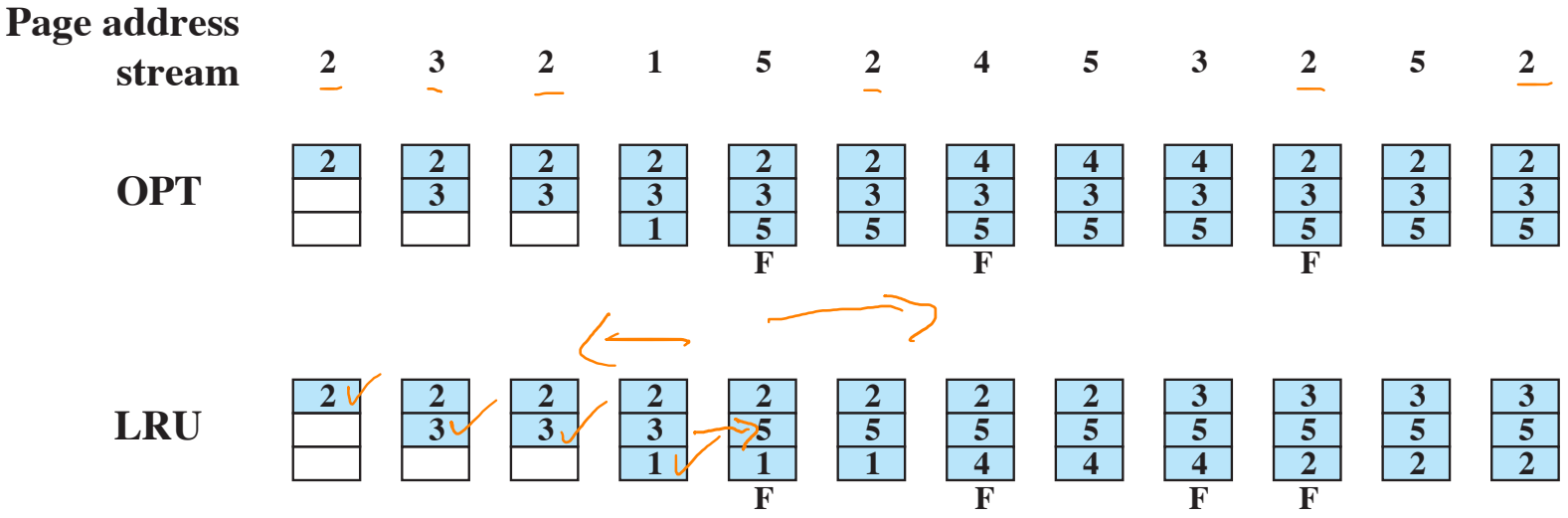
Example: Page-Replacement Algorithms

Reference 5 distinct pages…



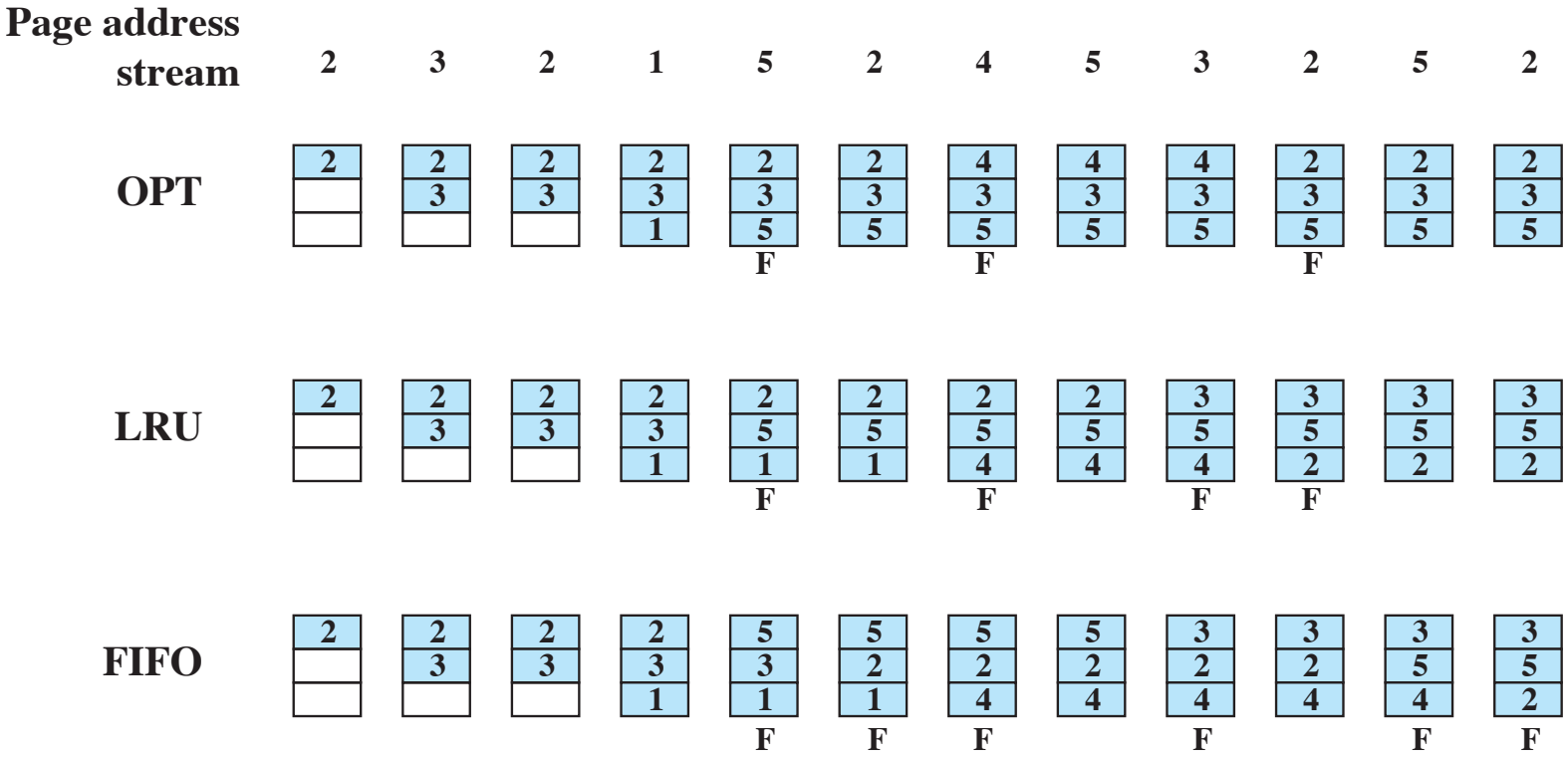
Example: Page-Replacement Algorithms

Reference 5 distinct pages…



Example: Page-Replacement Algorithms

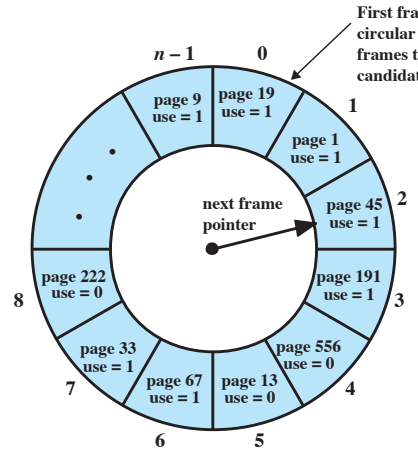
Reference 5 distinct pages…



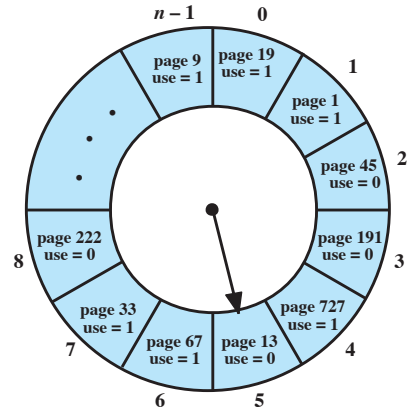
Example: Page-Replacement Algorithms

Reference 5 distinct pages…

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2	2	2	2	2	2	4	4	4	2	2	2
		3	3	3	3	3	3	3	3	3	3	3
				1	5	5	5	5	5	5	5	5
					F		F			F		
LRU	2	2	2	2	2	2	2	2	3	3	3	3
		3	3	3	5	5	5	5	5	5	5	5
				1	1	1	4	4	4	2	2	2
					F		F		F	F		
FIFO	2	2	2	2	5	5	5	5	3	3	3	3
		3	3	3	3	2	2	2	2	2	5	5
				1	1	1	4	4	4	4	4	2
					F	F	F		F		F	F
CLOCK	2*	2*	2*	2*	5*	5*	5*	5*	3*	3*	3*	3*
		3*	3*	3*	3	2*	2*	2*	2	2*	2	2*
				1*	1	1	4*	4*	4	4	5*	5*
					F	F	F		F		F	



(a) State of buffer just prior to a page replacement



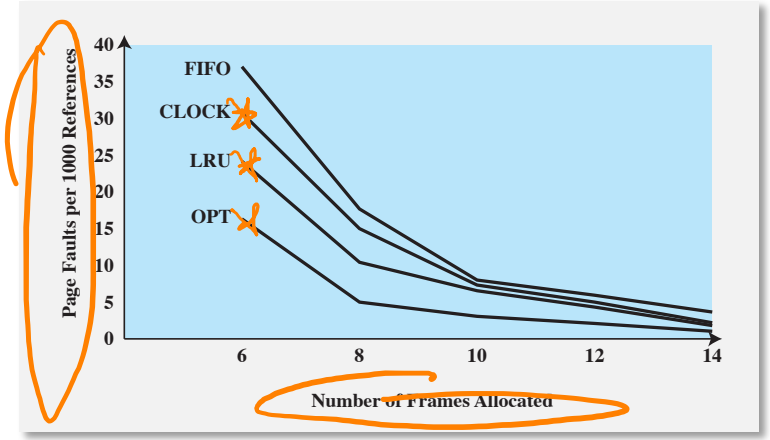
(b) State of buffer just after the next page replacement

Figure 8.15 Example of Clock Policy Operation

Example: Page-Replacement Algorithms

Reference 5 distinct pages…

Page address stream	2	3	2	1	5	2	4	5	3	2	5	2
OPT	2 / 3 /	2 / 3 /	2 / 3 /	2 / 3 / 1	2 / 3 / 5 F	2 / 3 / 5	4 / 3 / 5 F	4 / 3 / 5	4 / 3 / 5	2 / 3 / 5 F	2 / 3 / 5	2 / 3 / 5
LRU	2 / 3 /	2 / 3 /	2 / 3 /	2 / 3 / 1	2 / 5 / 1 F	2 / 5 / 1	2 / 5 / 4 F	2 / 5 / 4	3 / 5 / 4 F	3 / 5 / 2 F	3 / 5 / 2	3 / 5 / 2
FIFO	2 / 3 /	2 / 3 /	2 / 3 /	2 / 3 / 1	5 / 3 / 1 F	5 / 2 / 1 F	5 / 2 / 4 F	5 / 2 / 4	3 / 2 / 4 F	3 / 2 / 4	3 / 5 / 4 F	3 / 5 / 2 F
CLOCK	2* /	2* / 3*	2* / 3*	2* / 3* / 1*	5* / 3 / 1 F	5* / 2* / 1 F	5* / 2* / 4* F	5* / 2* / 4*	3* / 2 / 4 F	3* / 2* / 4	3* / 2 / 5* F	3* / 2* / 5*



Going Further...

- Resident Set Management
  — how many pages should be in main memory?
  - Resident Set Size: **Fixed** vs. **Variable**
  - Replacement Scope: **Local** (current process) vs. **Global** (everything)

- Cleaning Policy
  — when should a page be written out from main memory?
  (opposite of "Fetch Policy")

- Load Control
  — how many processes should be in main memory?
  (i.e., the level/degree of multiprogramming)

Virtual Memory

Putting it all together!

Virtual Memory

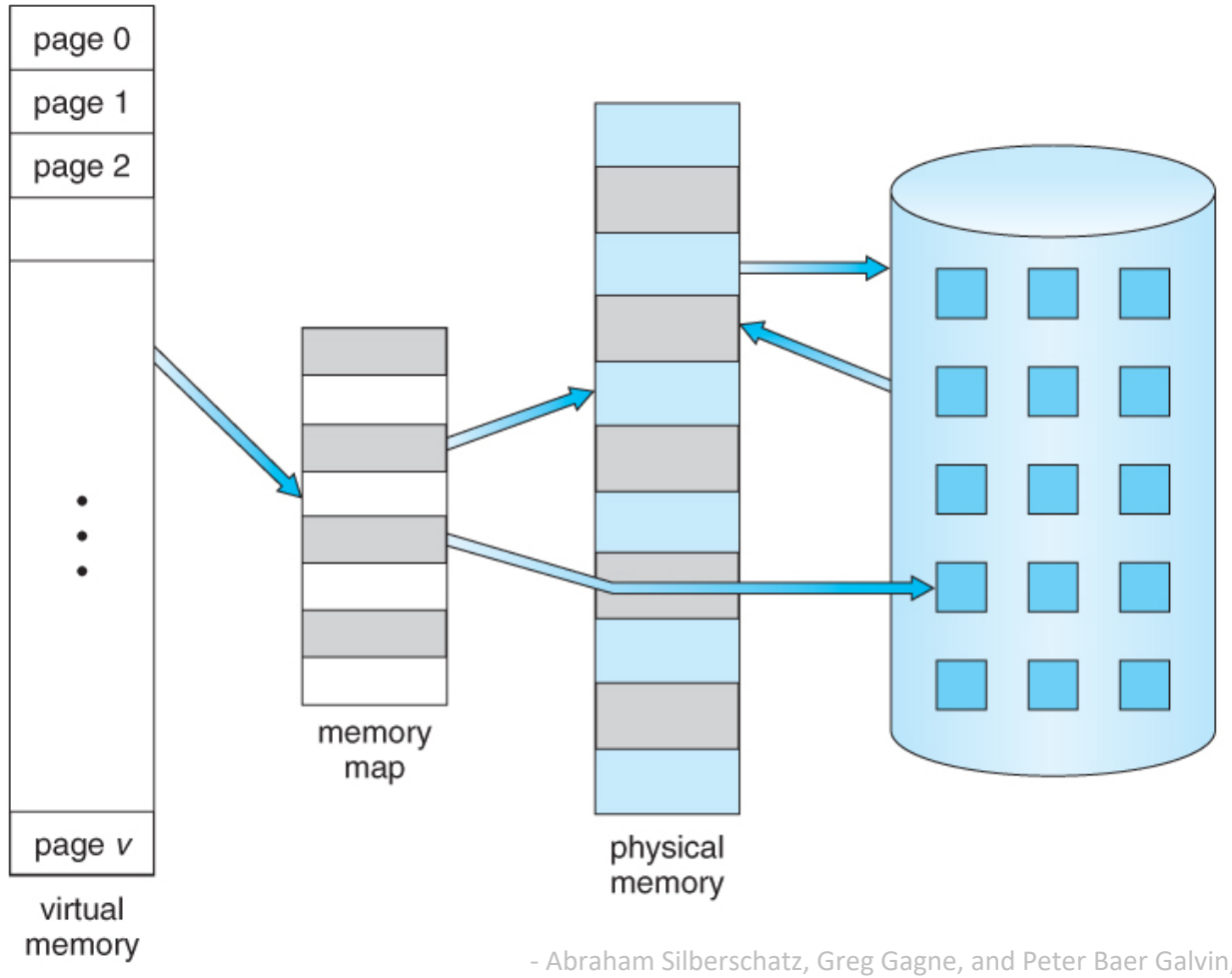
1. Bring bare minimum of process into main memory (MM)

2. Run process in the normal way

3. Use page tables to keep track of pages

4. Page not in MM?
   -> process moved to "blocked" state
   -> OS issues "read" from disk (re: page table)
   -> schedule other processes...
   -> I/O interrupt indicates when page is in MM
   -> blocked process moved back to "ready" state

5. Use replacement strategy to make space as needed



- Abraham Silberschatz, Greg Gagne, and Peter Baer Galvin, "Operating System Concepts," Ninth Edition, Chapter 9

Summary

Big Ideas

- Programs broken into non-contiguous pieces (pages or segments)
- Logical addresses dynamically translated into physical addresses at run-time
- Not all memory needs to be in MM during execution!
- Special HW/caches help to speed up memory access

So what is Virtual Memory?

- All memory can be addressed as if it were part of main memory
- References to memory are dynamically translated
- Limited only by the addressing scheme of the system & amount of secondary memory