Memory (Part IV):
Virtual Memory

Professor Travis Peters
CSCI 460 Operating Systems
Fall 2019

Some slides & figures adapted from Stallings instructor resources.

Some slides adapted from Adam Bates's F'18 CS423 course @ UIUC
https://courses.engr.illinois.edu/cs423/sp2018/schedule.html

Today

Announcements

- Project Proposal ***Due Friday (11/01)!***
- HW5 posted — ***Also Due Friday (11/01)!***
- PA2 posted — ***Due NEXT Friday (11/08)***… get started ASAP… ;)

Goals & Learning Objectives

- Wrapping up virtual memory
- Discuss some issues closer to real-world implementations of virtual memory

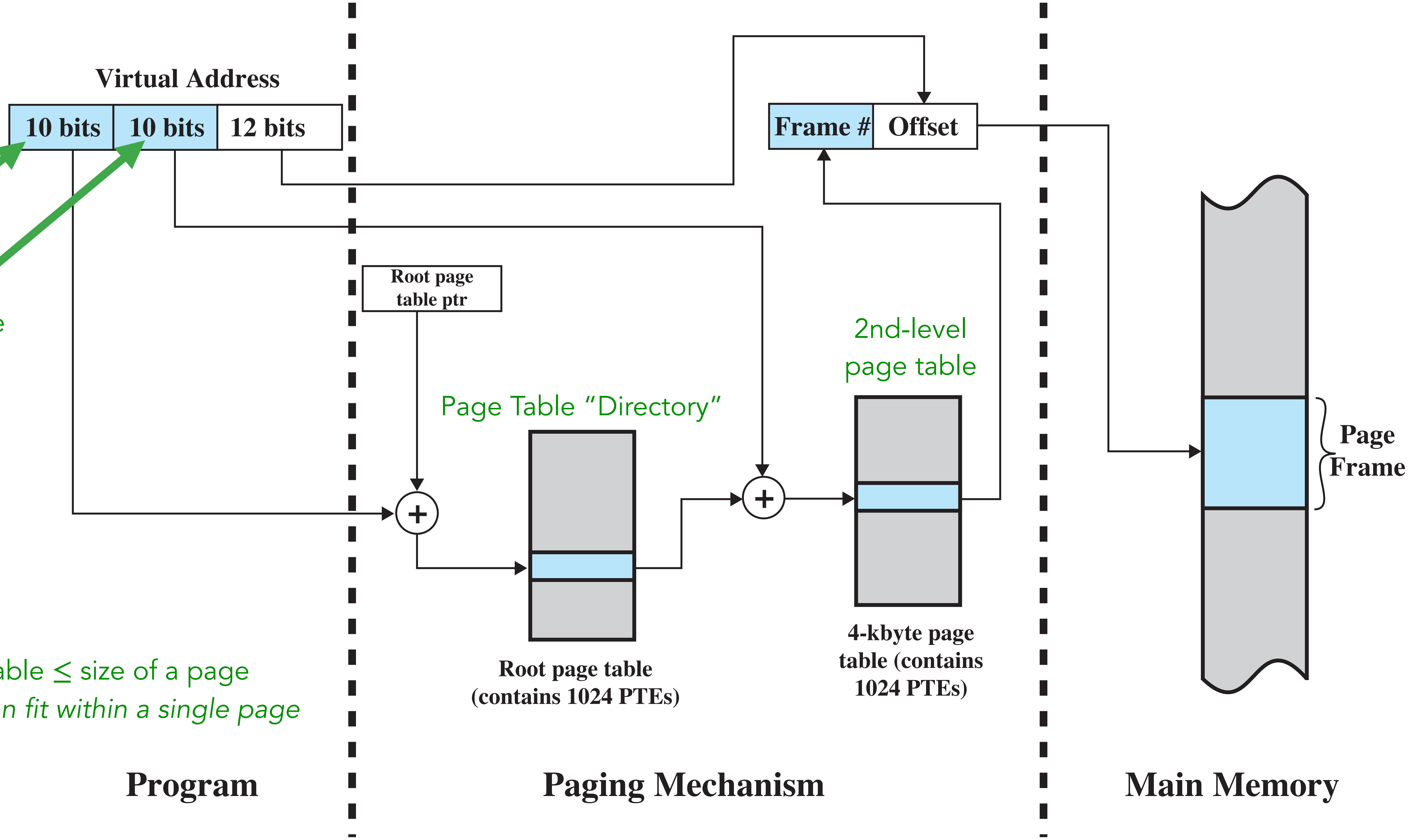
Address Translation in 2-Level Paging System

Page tables themselves could be paged in/out…
but at least *part* of a page table must be in MM for a process to run… → ***2-Level Paging System***

X page tables

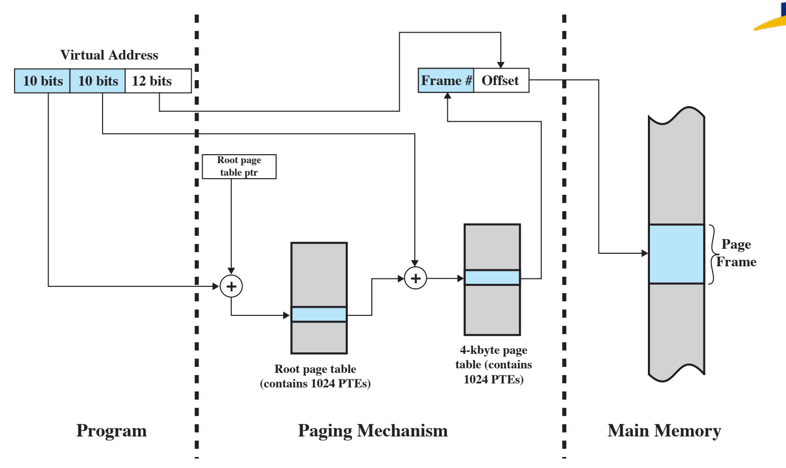
Y entires per page table

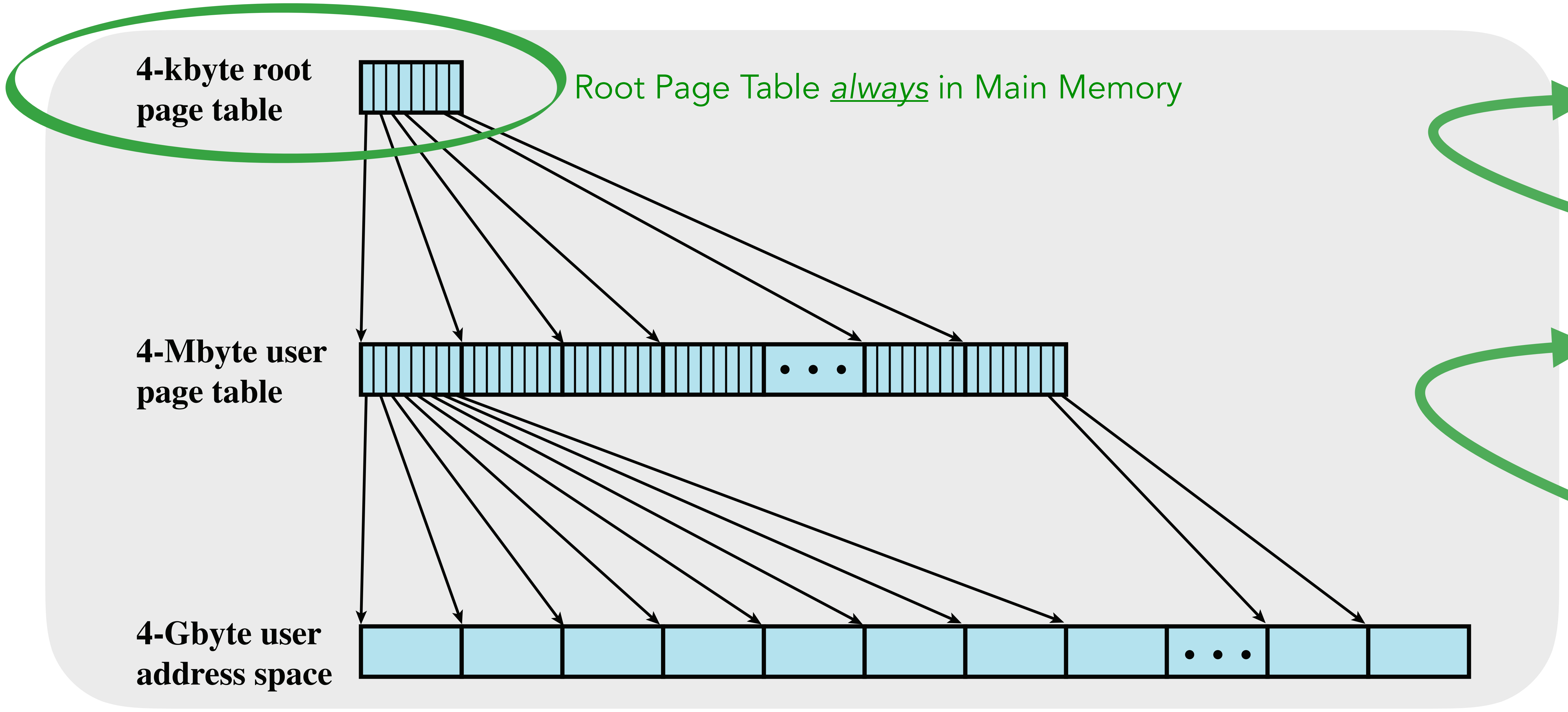
$= X \times Y$ total pages



Typically, size of a page table ≤ size of a page
i.e., a 2nd-level page table can fit within a single page

Address Translation in 2-Level Paging System *(cont.)*





Root Page Table *always* in Main Memory

Each 4 MB page table is mapped by root page table → 1024 page tables mapped by 4-byte PTE → root page table size = 4 kB

Each 4 kB page is mapped by 4-byte PTE → user page table requires 4 MB (2^{22}) … or according to this scheme, $2^{22} \div 2^{12} = 2^{10}$ = 1K pages

Example: 2-level scheme,

- 4 GB (2^{32}) address space,
- byte-level addressing,
- 4 kB (2^{12}) page size

  → 2^{20} total pages

Inverted Page Table Structure

Problem

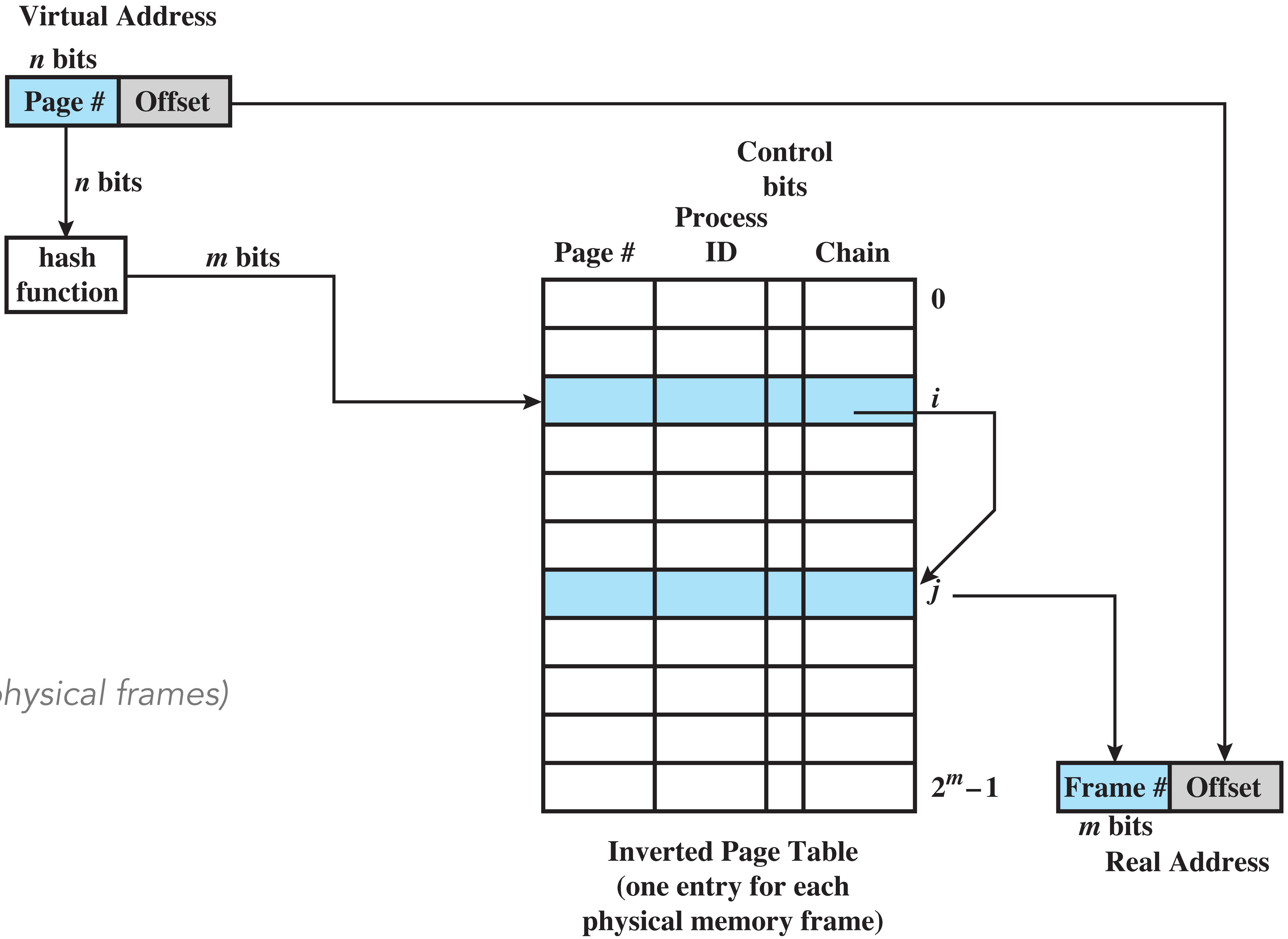
size of 1+ level page tables is that they are proportional to the size of the virtual address space

Idea!

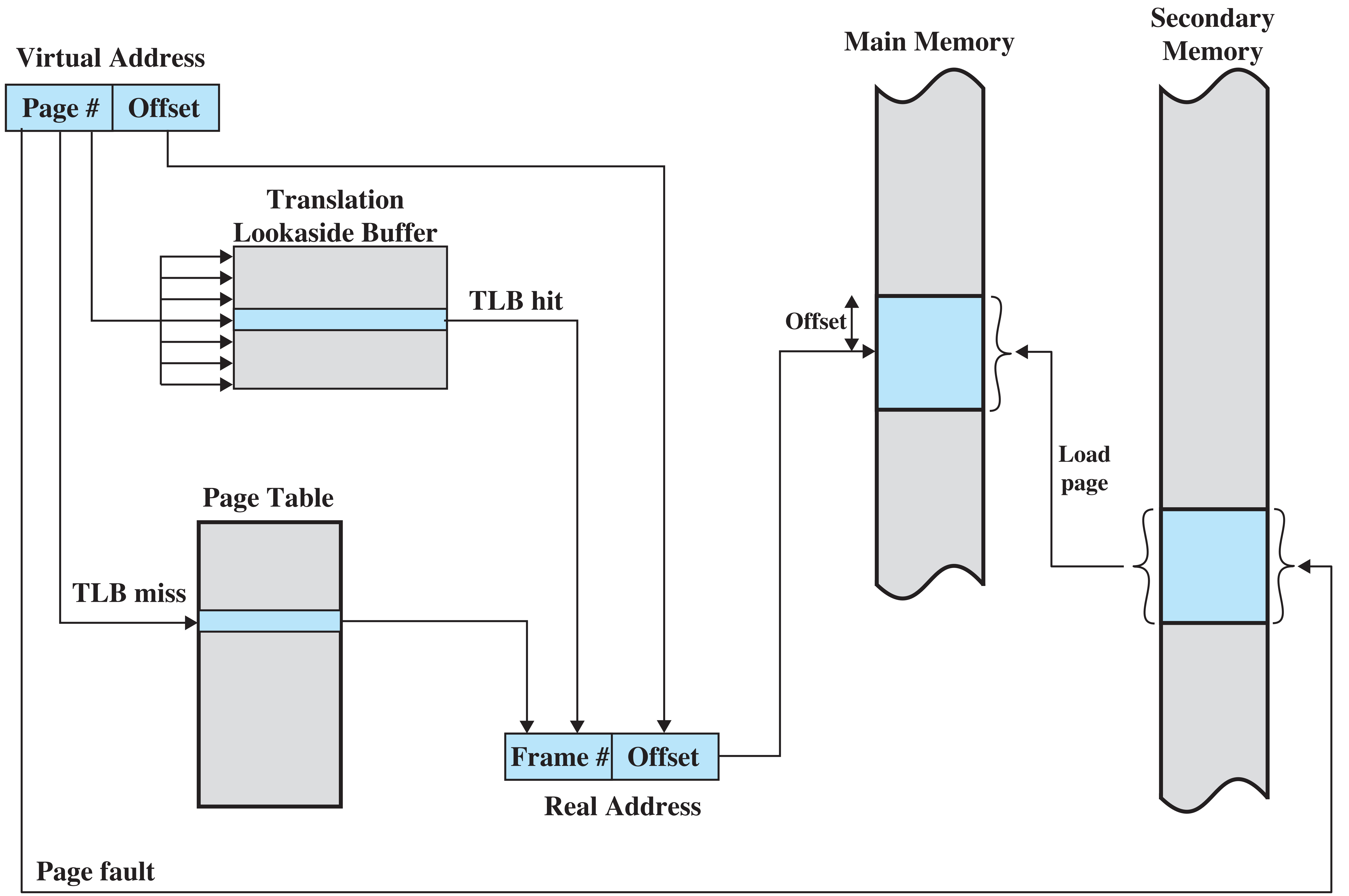
Invert page table *(index by physical frames)*

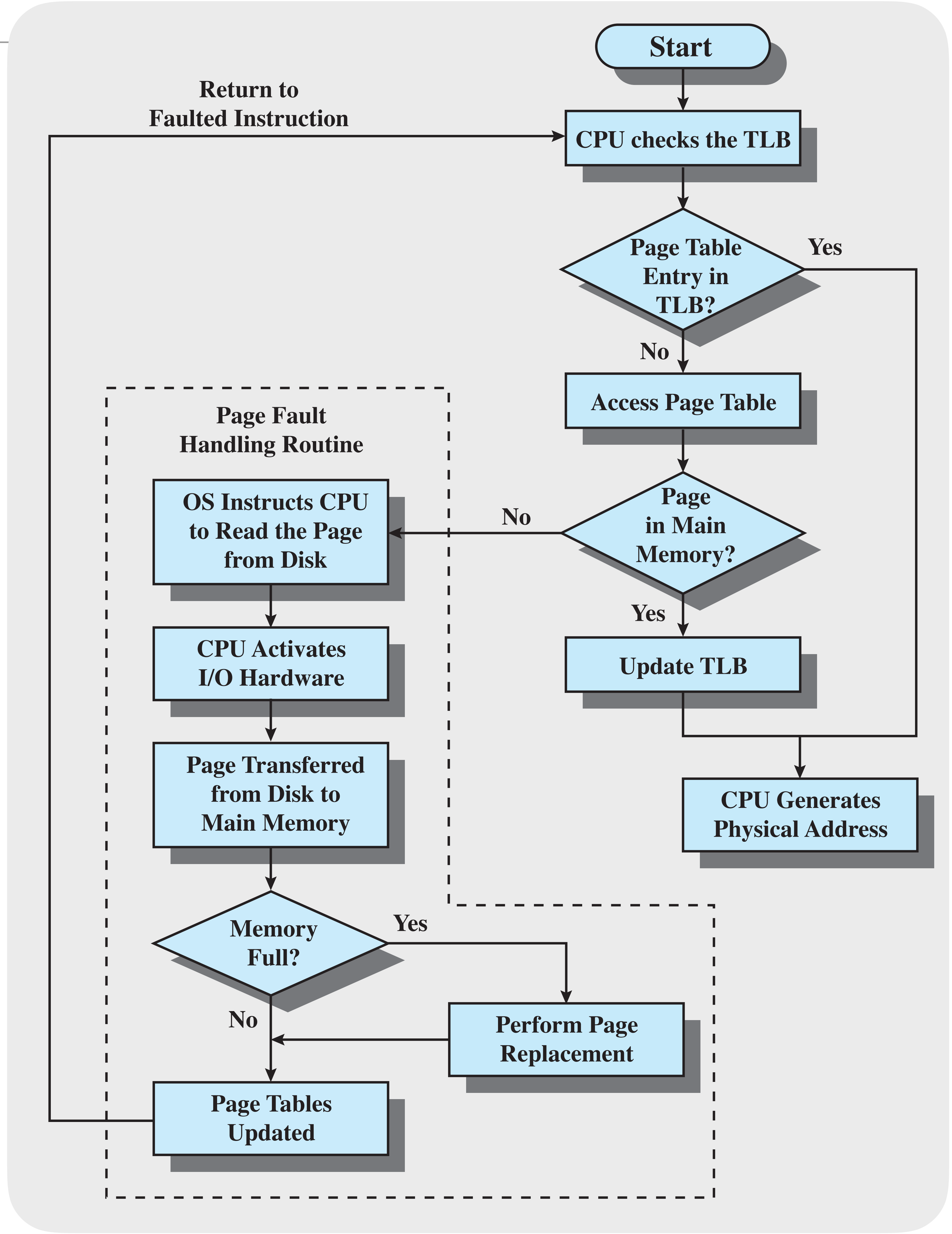
- *n >> m*
  → hash page # into smaller page table
  → use chaining for collisions

- *Regardless of virtual address space size,*
  the size of an inverted page table remains the same
  (i.e., inverted page table size is proportional to # of physical frames)

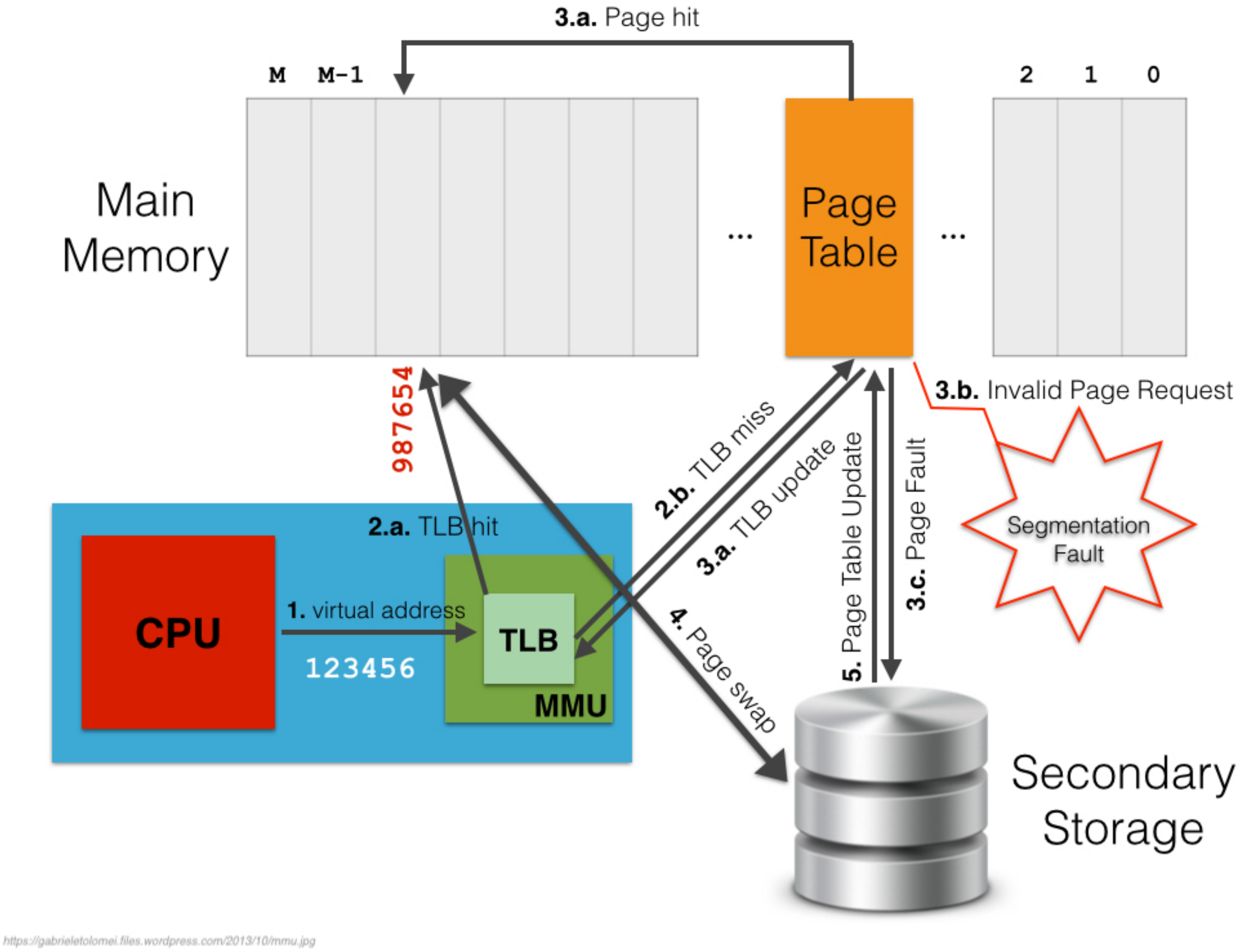


Using a Translation Lookaside Buffer (TLB)

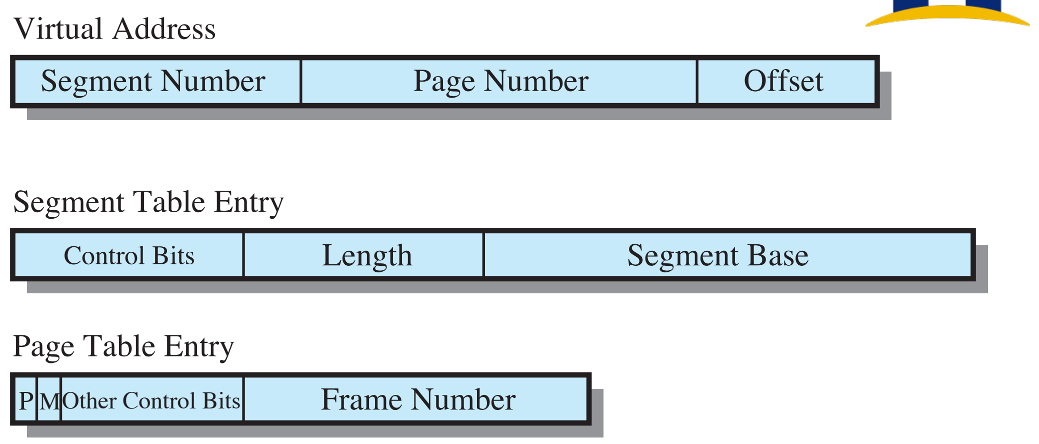


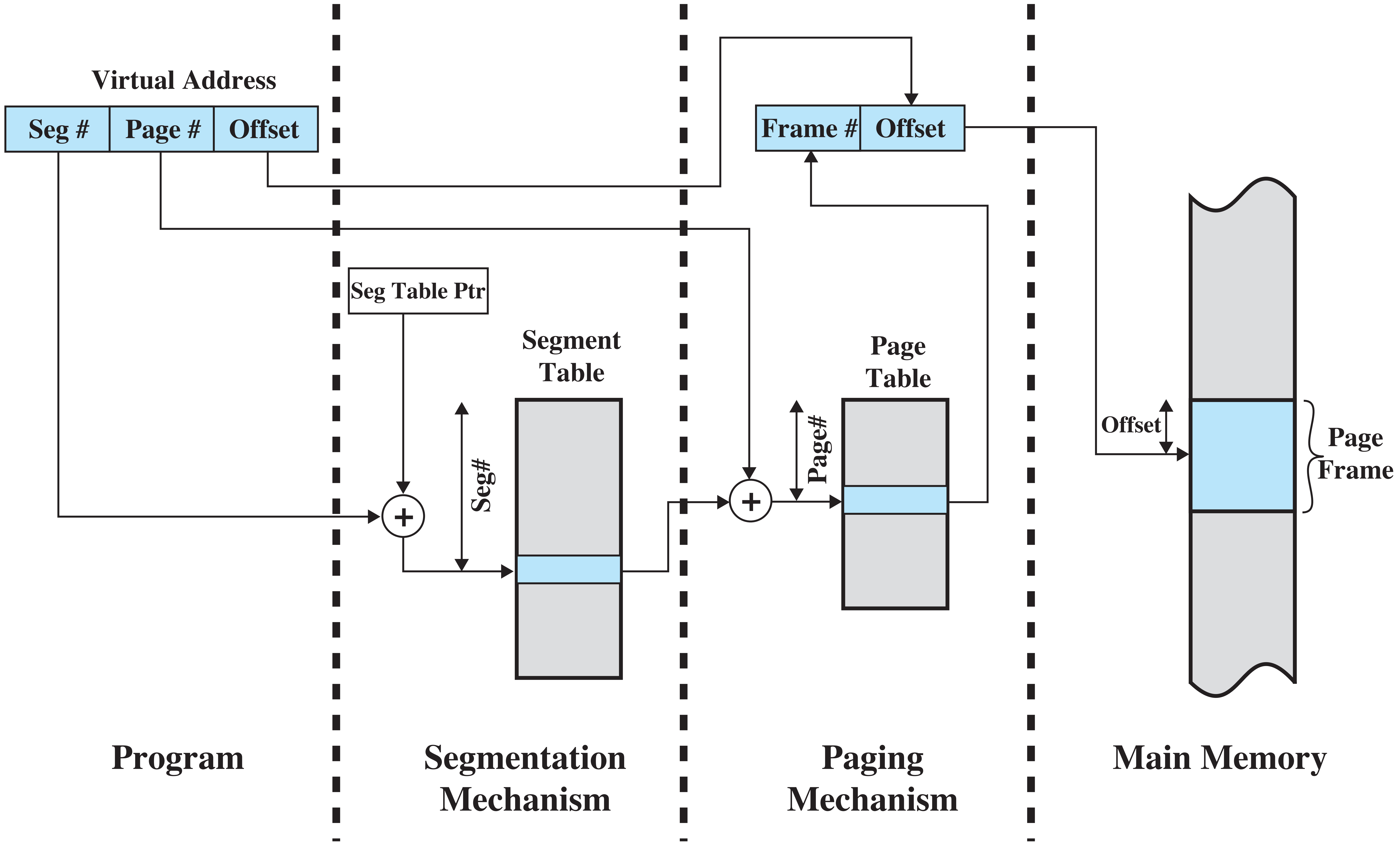


Using a Translation Lookaside Buffer (TLB) — *another look*

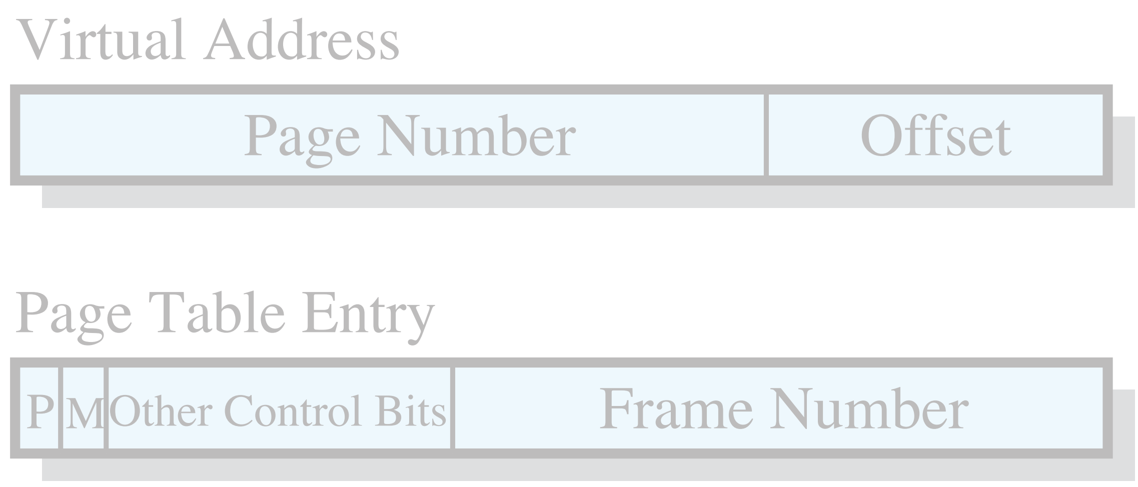


Address Translation in a Segmentation/Paging System

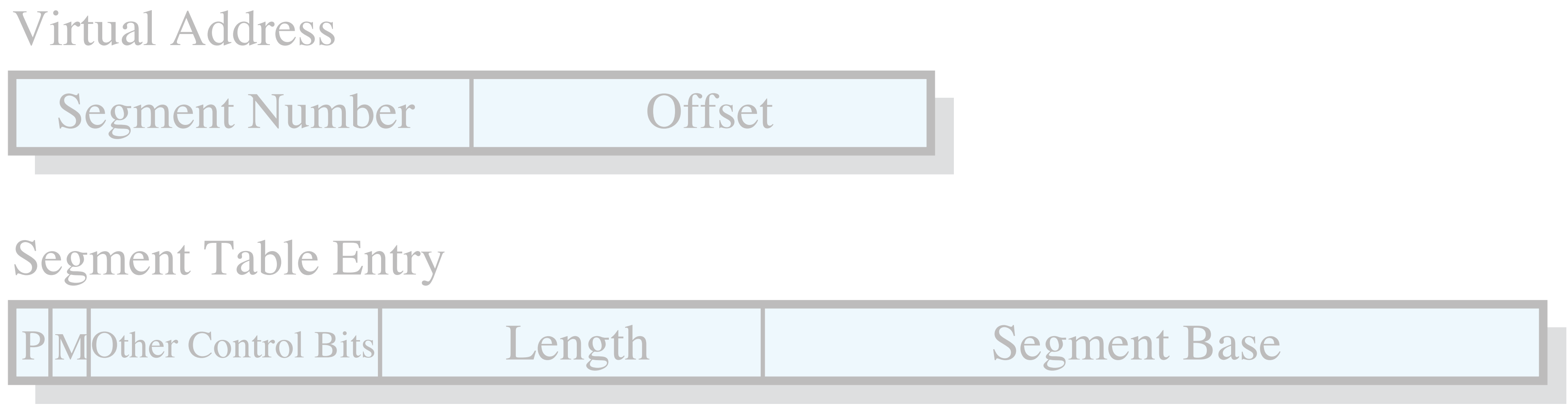




Memory Management Formats



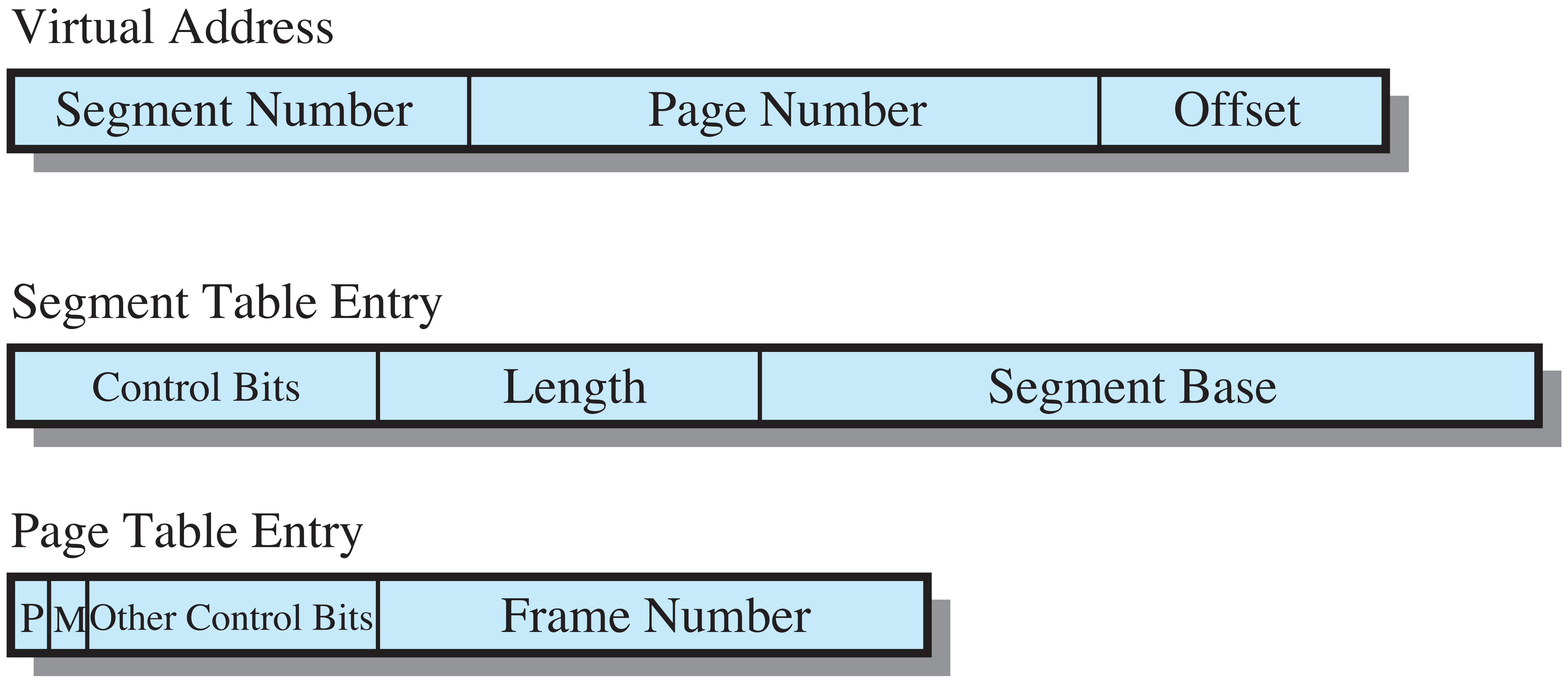
Paging



Segmentation

If Page is Present (P)
→ *use frame number*

If Page Modified (M)
→ *write out*



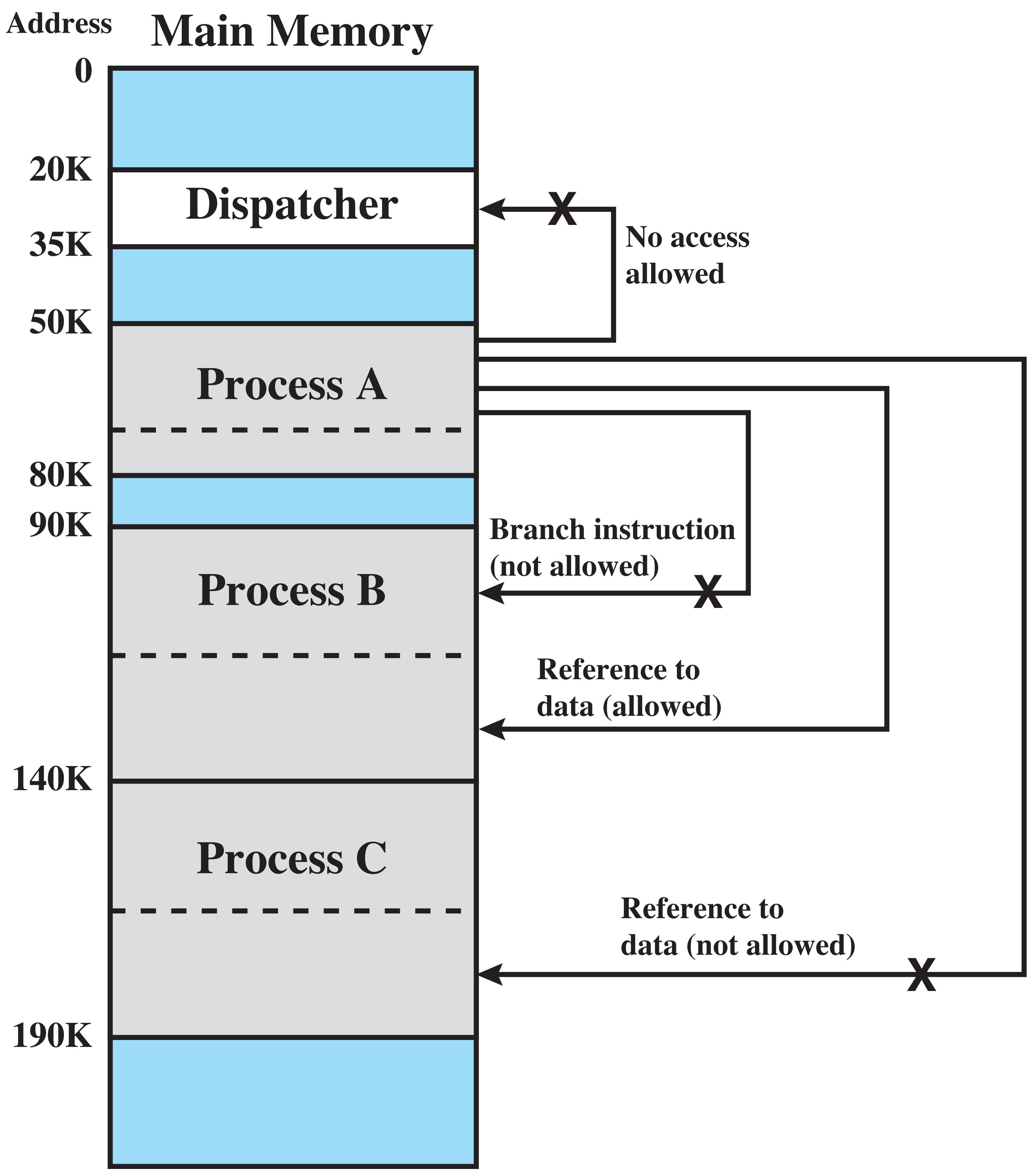
Paging *and* Segmentation

P= present bit
M = Modified bit

Protection Between Segments

- Configurable settings for segments
  - readable
  - writable
  - executable

- This is possible on a per-page level, but it is less intuitive/kind of awkward since paging structure is not visible to programmers.



Designing an OS with Virtual Memory

Goal: *Reduce Frequency of Page Faults*

Designing an OS with Virtual Memory

Goal: *Reduce Frequency of Page Faults*

Fetch Policy — *when should a page be brought into main memory?*

- ***Demand Paging***
  Bring pages in when a reference is made (as needed / on demand)

- ***Prepaging***
  Bring in a number of pages at once (e.g., other nearby pages on disk)

Designing an OS with Virtual Memory

Goal: *Reduce Frequency of Page Faults*

Placement Policy — *where should a page be put in main memory?*
(Re: Chapter 7)

- ***Segmentation***
  → *Best-Fit, First-Fit, Etc.*

- ***Paging***
  → *Not an issue (page-frame mappings are equally efficient)*

Designing an OS with Virtual Memory

Goal: *Reduce Frequency of Page Faults*

Replacement Policy — *which page* should be replaced when a new page must be brought into main memory?***

- **Random Page Replacement**
  Choose a page randomly
- **Optimal (OPT)**
  Replace the page that will not be used the most time in the future

  Benchmarks
- **LRU**
  Replace the page that has not been used for the longest time

  Nearly optimal
  Complex
- **FIFO**
  Replace the page that has been in primary memory the longest; use a circular buffer

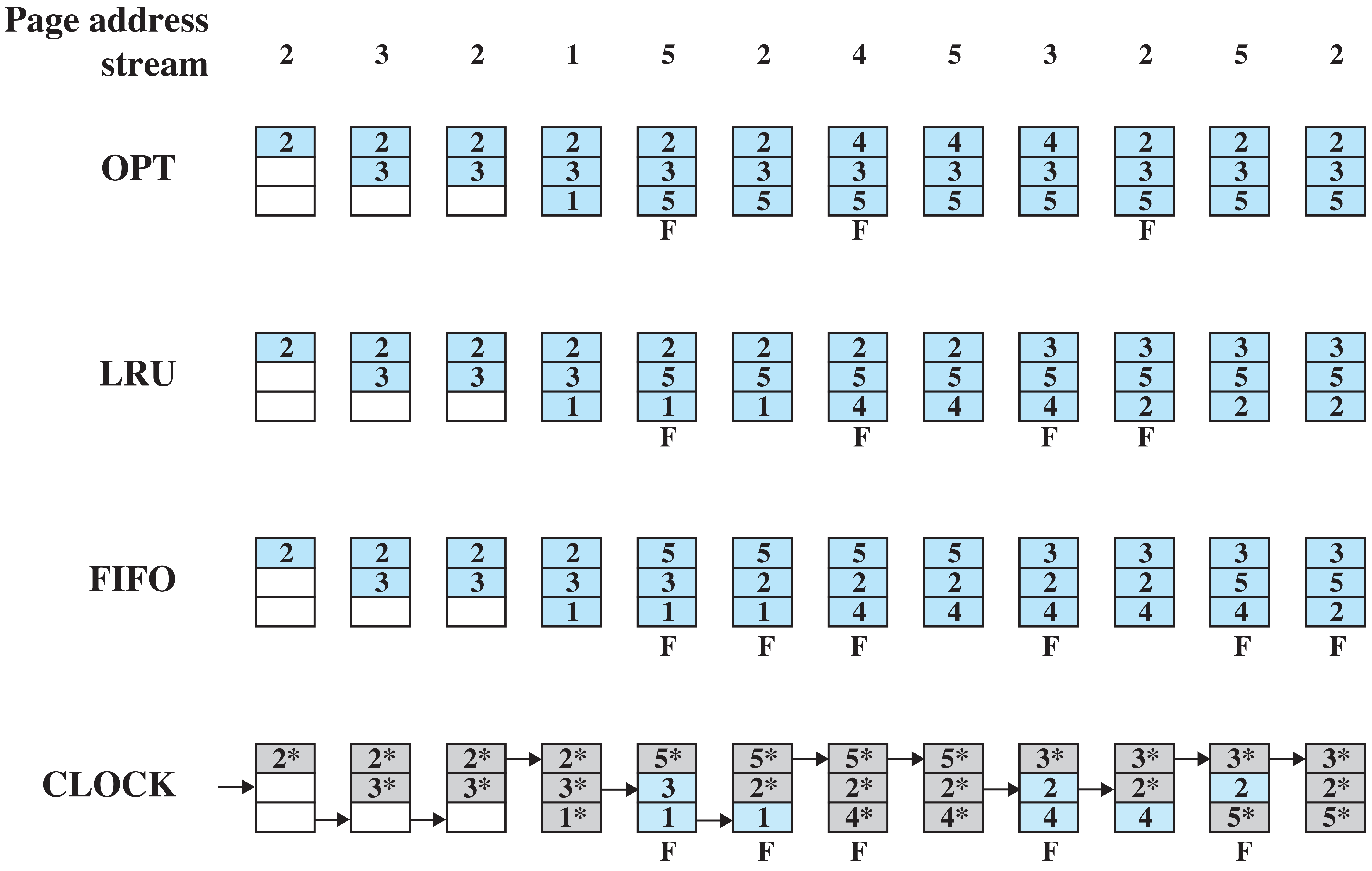
  Simple
  Often Incorrect
- **Clock** *(approximation of LRU)*
  Basically, FIFO w/ a "use" bit; cycle through pages, replace pages not used recently
- **LFO**
  Replace the page that is used least often
- **2nd Chance**
  Choose a victim to evict; "mark" it; spare it until later; if encountered again, then page out

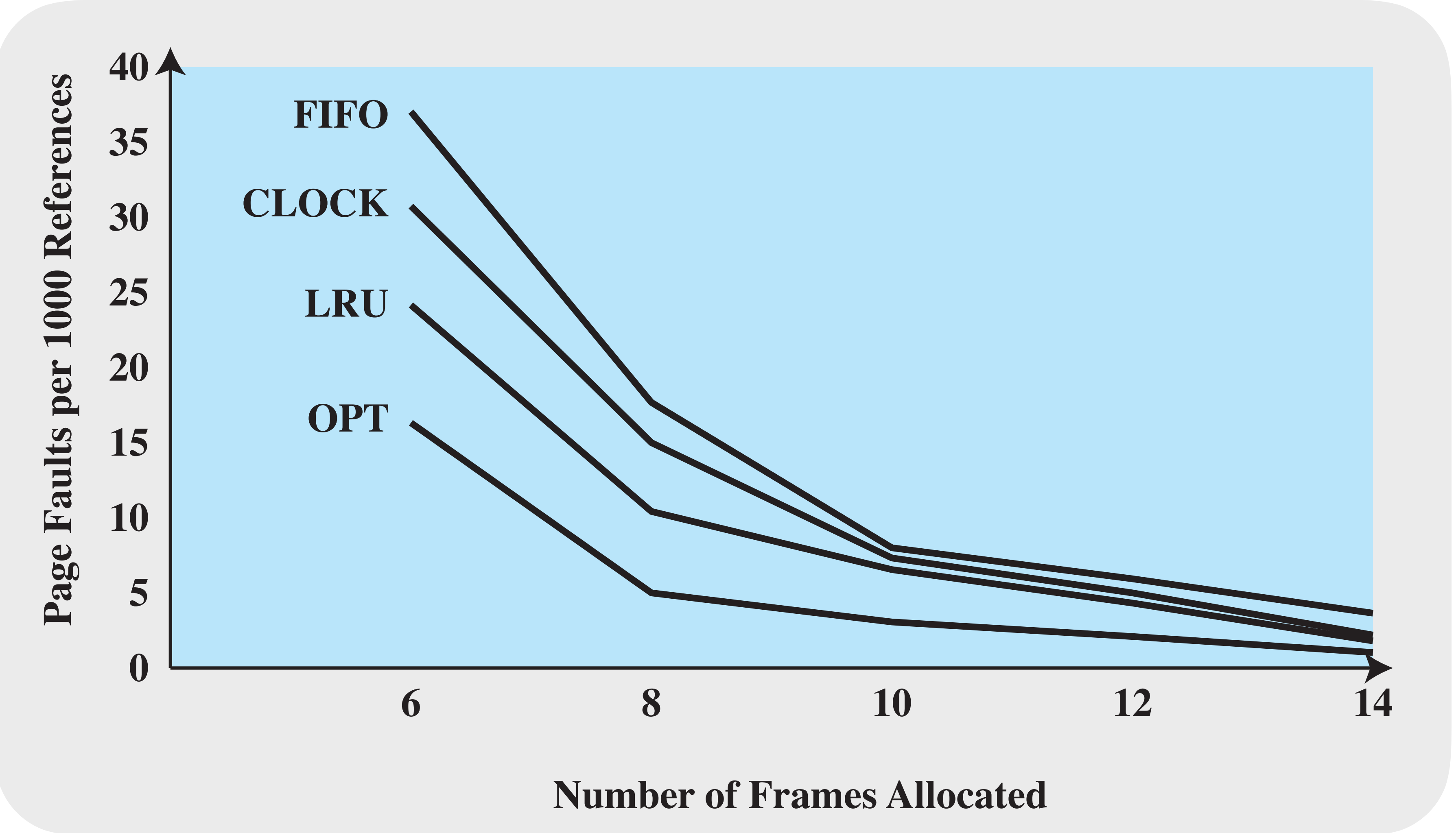
Want to replace an **unlocked page that is least likely to be referenced in the near future.*

***Most interesting if we assume that all frames in main memory are occupied — then a page fault occurs.*

Example: Page-Replacement Algorithms

Order of page references: 2 3 2 1 5 2 4 5 3 2 5 2





F = page fault occurring after the frame allocation is initially filled

Designing an OS with Virtual Memory

Goal: *Reduce Frequency of Page Faults*

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 to secondary memory? (opposite of "Fetch Policy")*

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