Operating Systems!

Memory: Approaches to Memory Management (Part 4)

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?! ☺
  - *Exam 1: Nice work! **(Still)** Working on grading…*

- Upcoming Deadlines
  - **Sunday [10/18/2020] @ 11:59 PM (MST)** - PA3 (Group Assignment)

    Yalnix teams: *While PA3 == yalnix checkpoint 1, you should plan to be near completing checkpoint 2 at this point!*

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

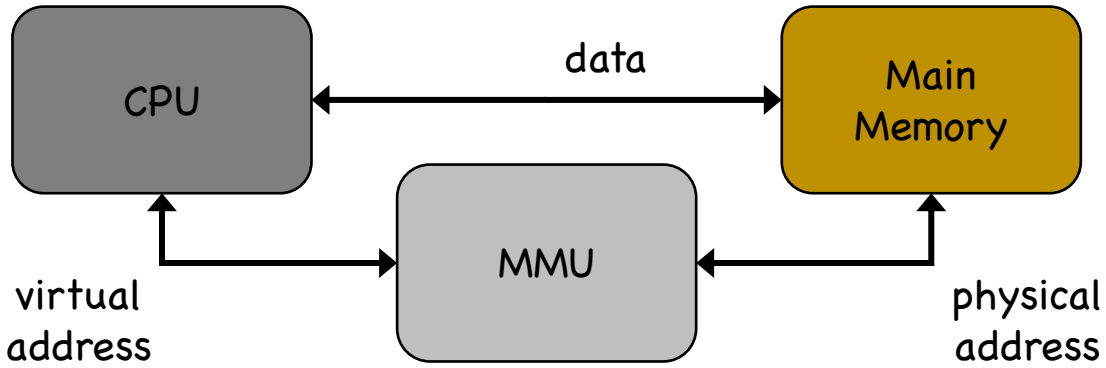
    Yalnix teams: *you should plan to be near completing checkpoint 3 at this point!*

Today *(cont.)*

Last Week: Logical vs. Physical Addresses, Paging, Address Translation, ...

- 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

➔ Virtual Memory



The Stack

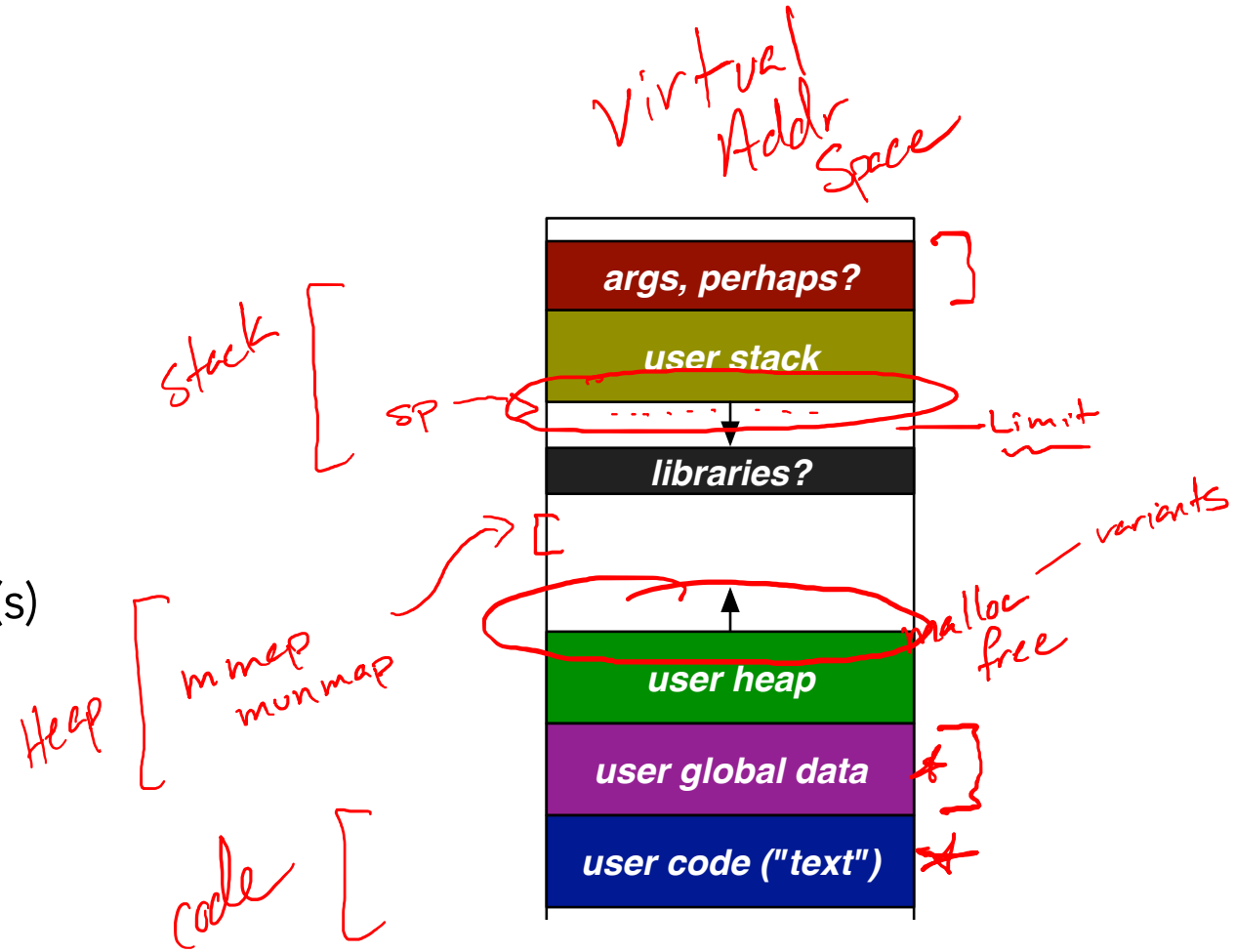
Stacks of Mystery

Bottom of the Stack

- Stacks "grow" in a different way...
  -> implicit growth!

- Basic idea:
  - *if* access is within max limit, allocate page(s)
  - ***else*** send SIGSEGV (which typically kills it)

- *How?* Page Faults!
  - **major fault**
    vs.
  - **minor fault**



See:
SE posts on stack allocation in Linux
https://unix.stackexchange.com/a/239323
https://unix.stackexchange.com/a/280865

see: **fault.c (5,6,7)**
*(Why don't we see syscalls when running **strace**...)*

Page Tables

Practical Issues

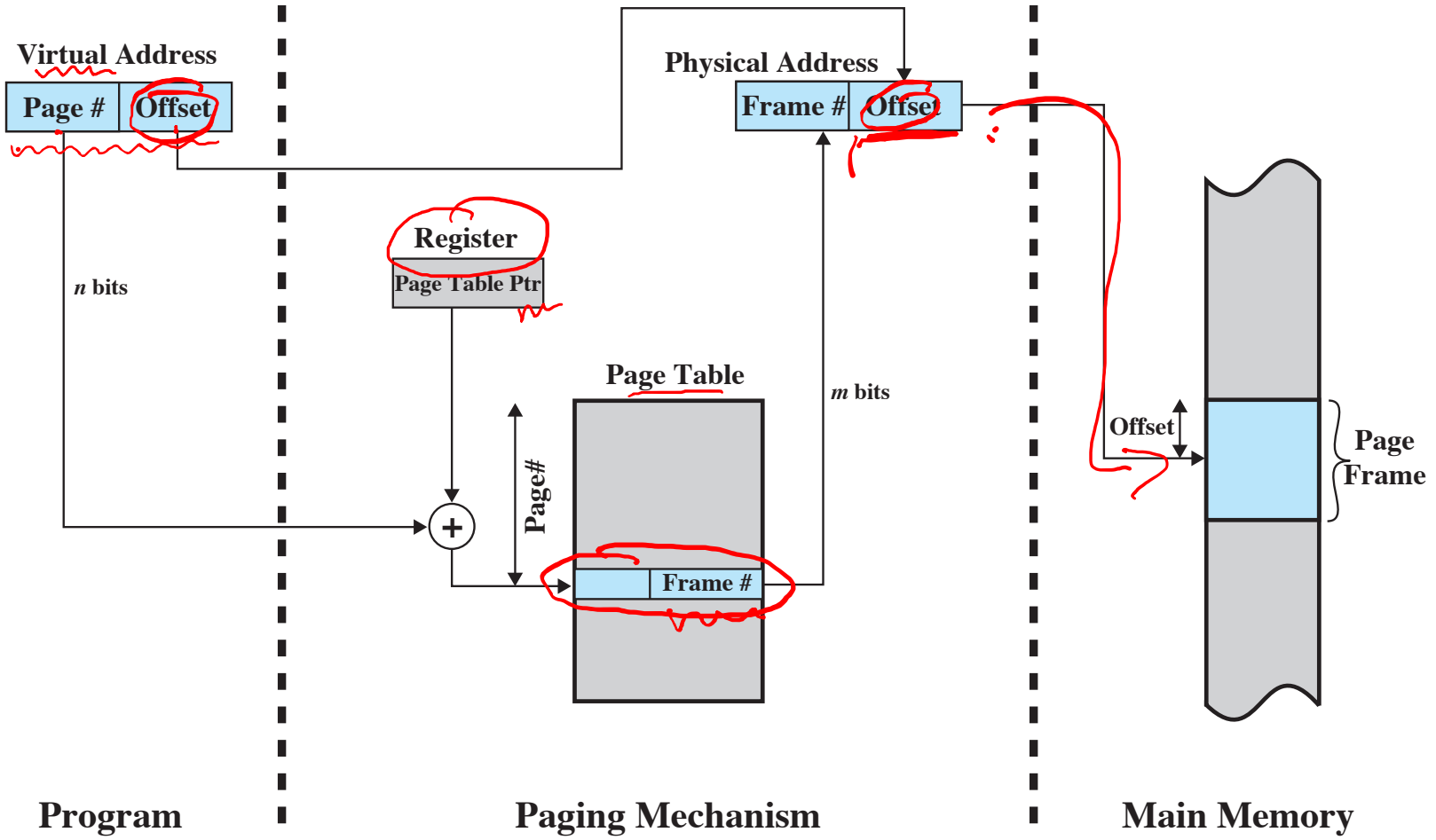
Address Translation:
1-Level Paging System

Virtual Address

Page Number	Offset

Page Table Entry

P	M	Other Control Bits	Frame Number

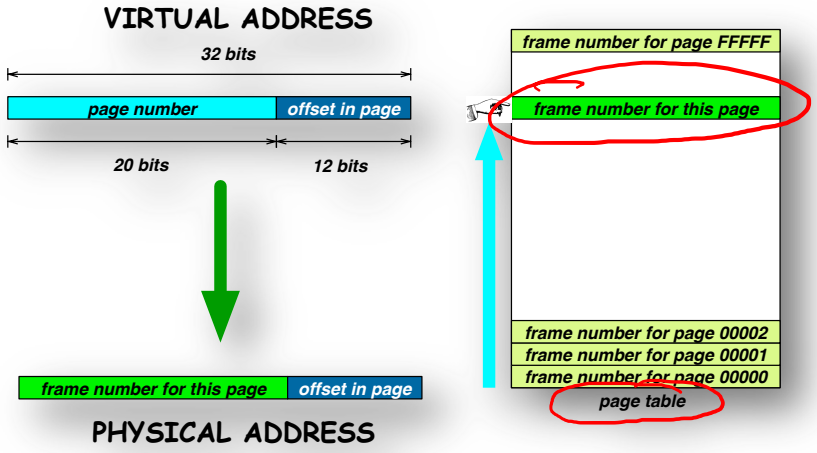


How big are these page tables?

Activity! (Think-Pair-Share)

- Suppose…
  - Virtual Memory = 4GB (2^32)
  - Page Size = 4KB (2^12)
  - PTE = 4B (2^2)

4GB RAM



- How many page table entries?

$2^{32} / 2^{12} = 2^{20}$

- How much memory is needed to store such a page table?

$2^{20} \cdot 2^{2} = 2^{22}$ (4MB)

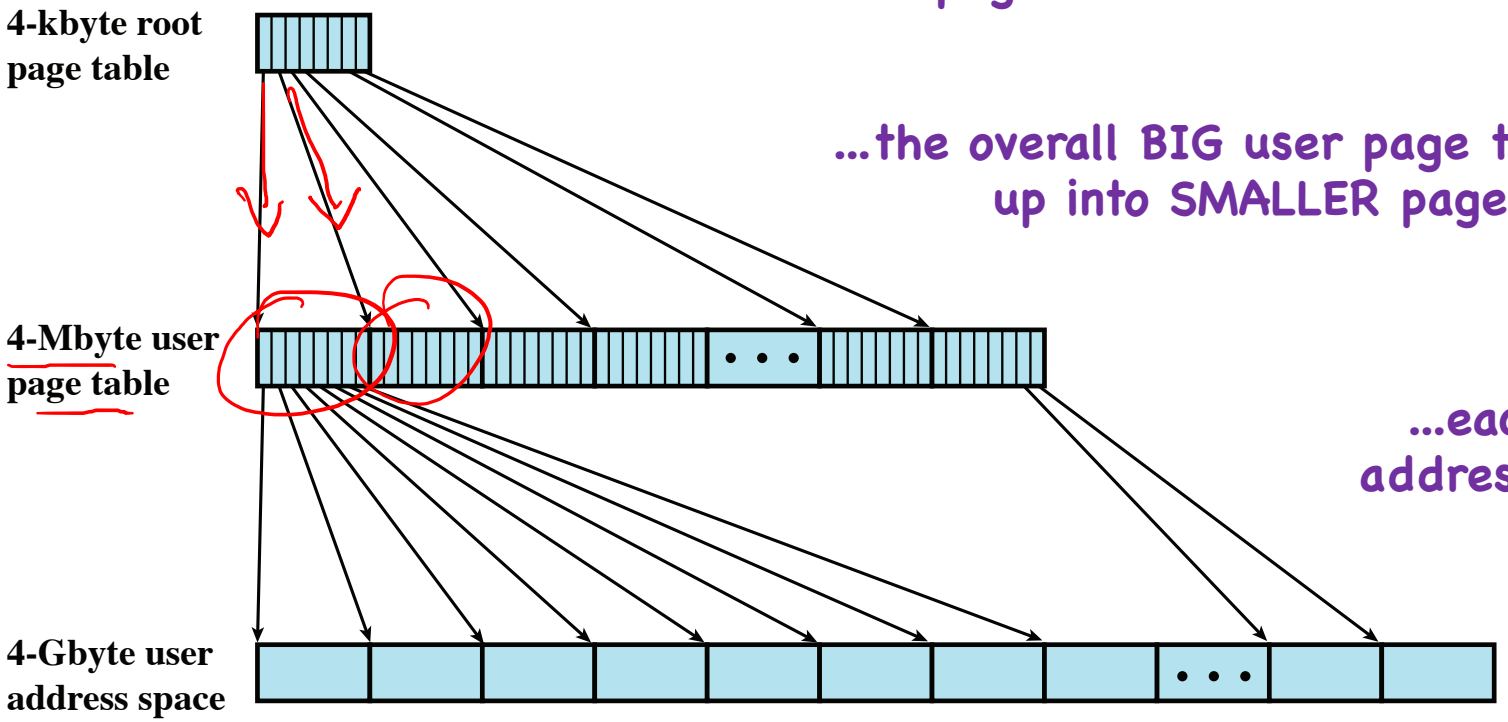
- How many pages are needed to hold the page table?

$2^{22} / 2^{12} = 2^{10}$

Address Translation:

2-Level Paging System (Overview)

Each entry of the root/directory addresses one user page table...



...the overall BIG user page table is divided up into SMALLER page tables...

...each PTE in a user page table addresses one physical memory page

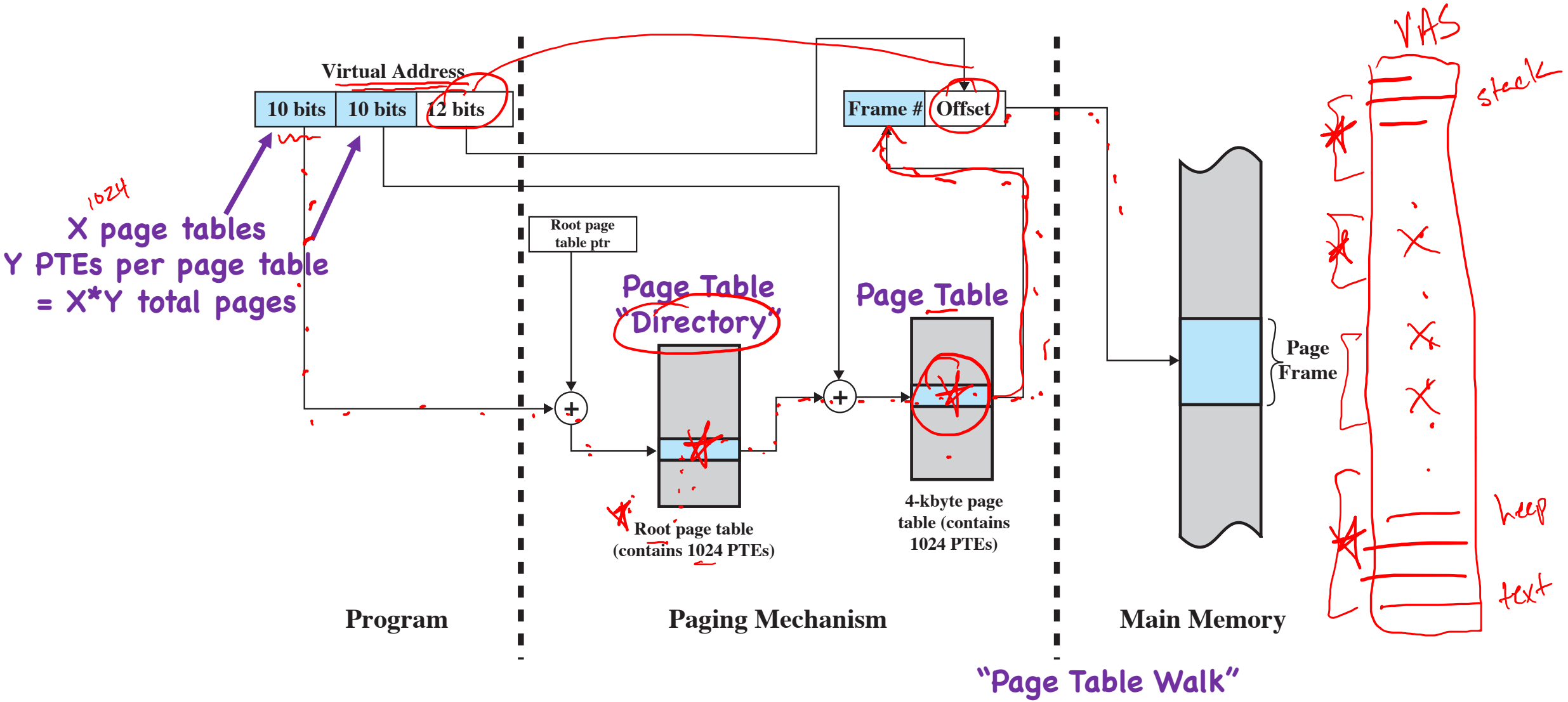
Address Translation:
2-Level Paging System

Virtual Address

Page Number	Offset

Page Table Entry

P	M	Other Control Bits	Frame Number

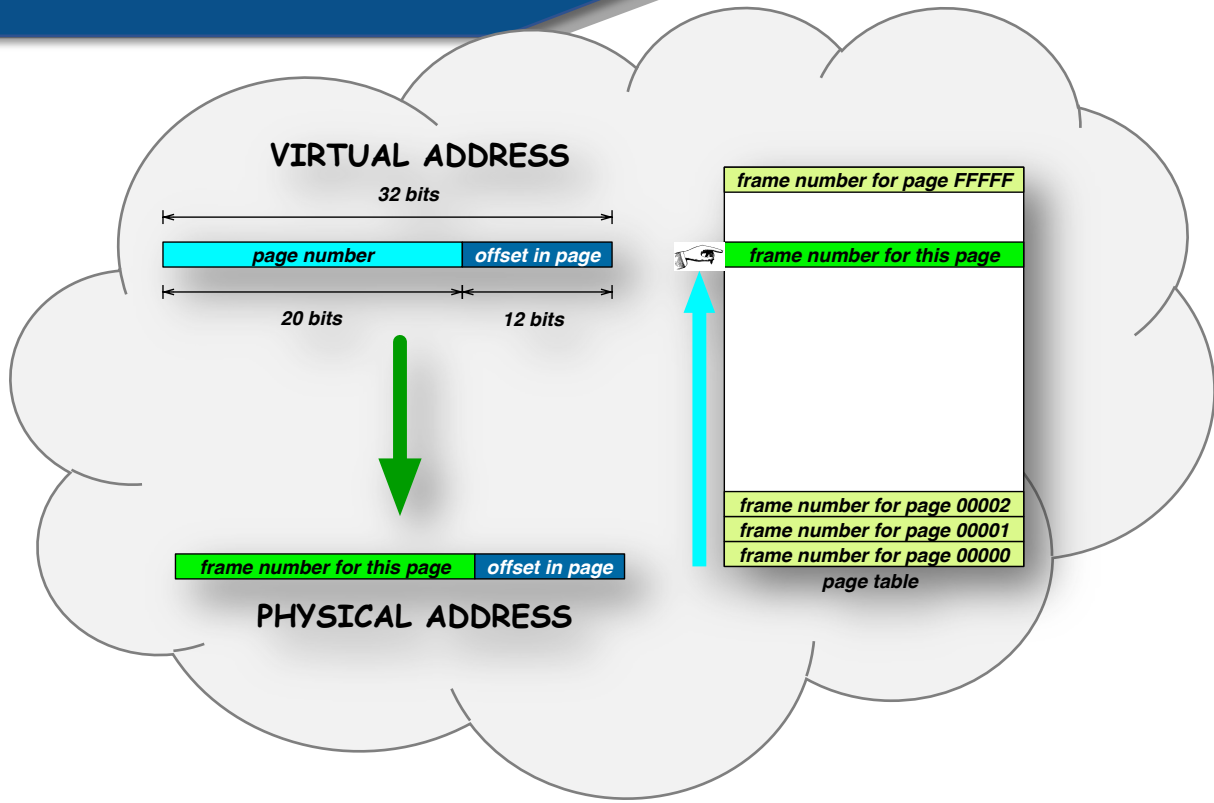


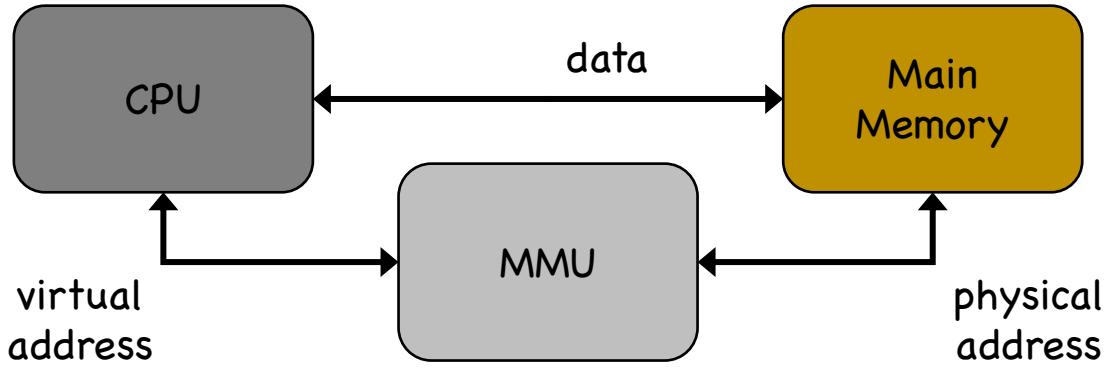
The TLB

An overview of the page table cache

So...

***Q:** Where should the page table live?*

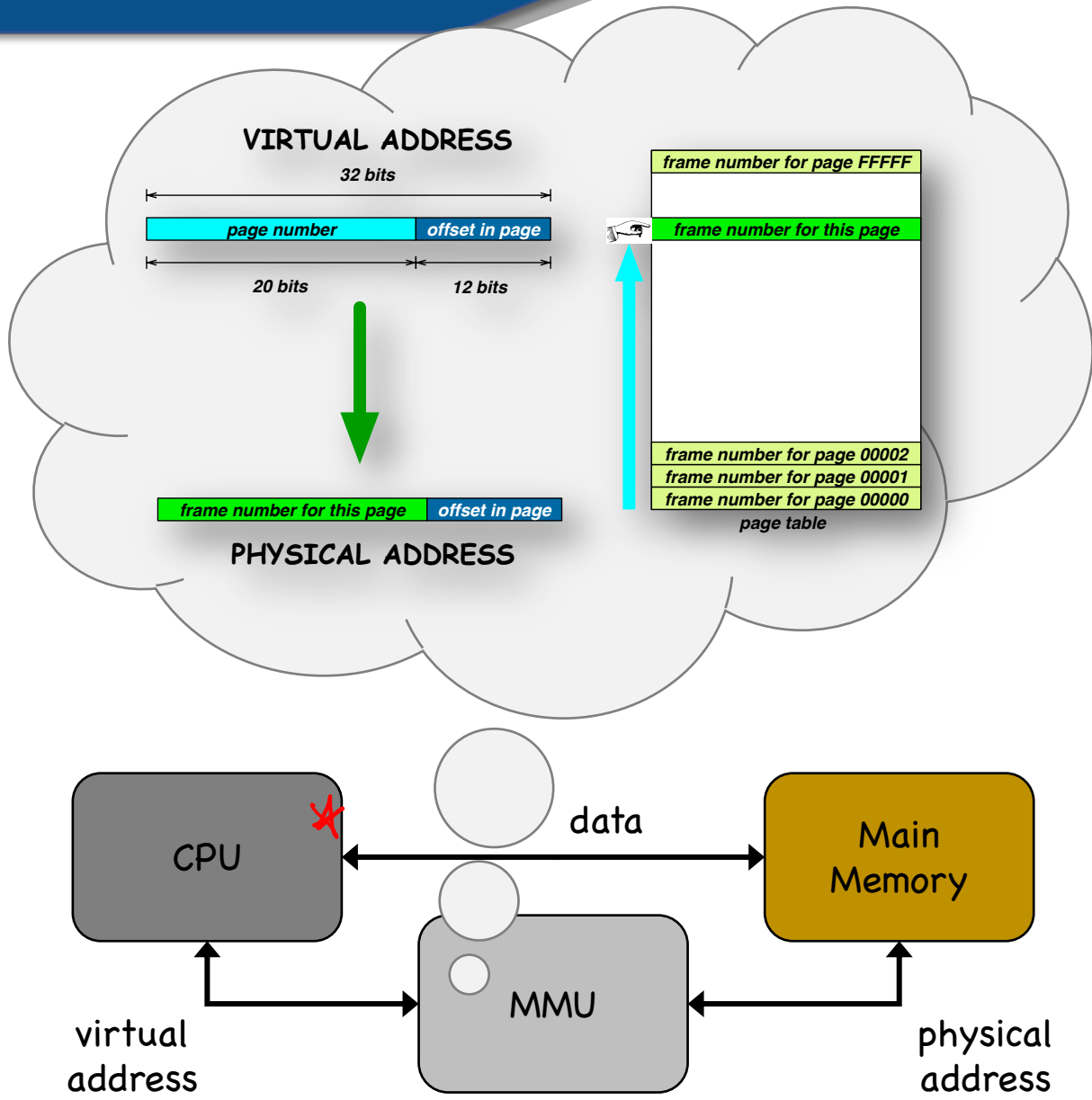




Option 1: CPU/MMU

Q: Where should the page table live?

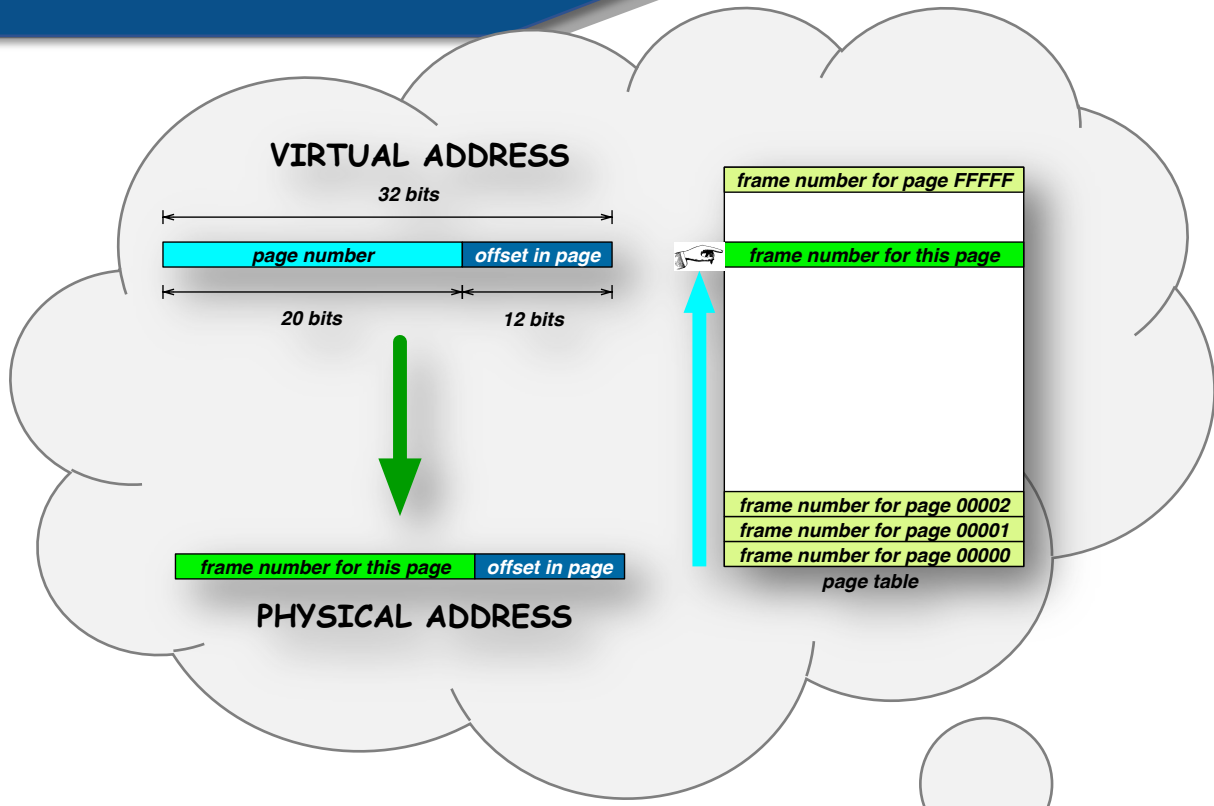
- Within special register(s) inside the MMU?
  - Too small... ☹



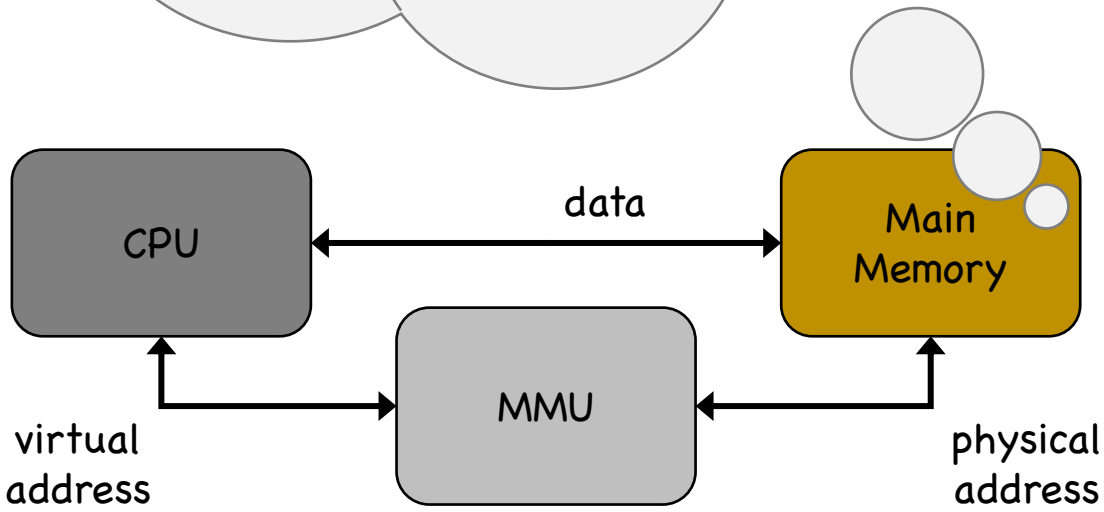
Option 2: MM

Q: Where should the page table live?

- Within special register(s) inside the MMU?
  - Too small... ☹



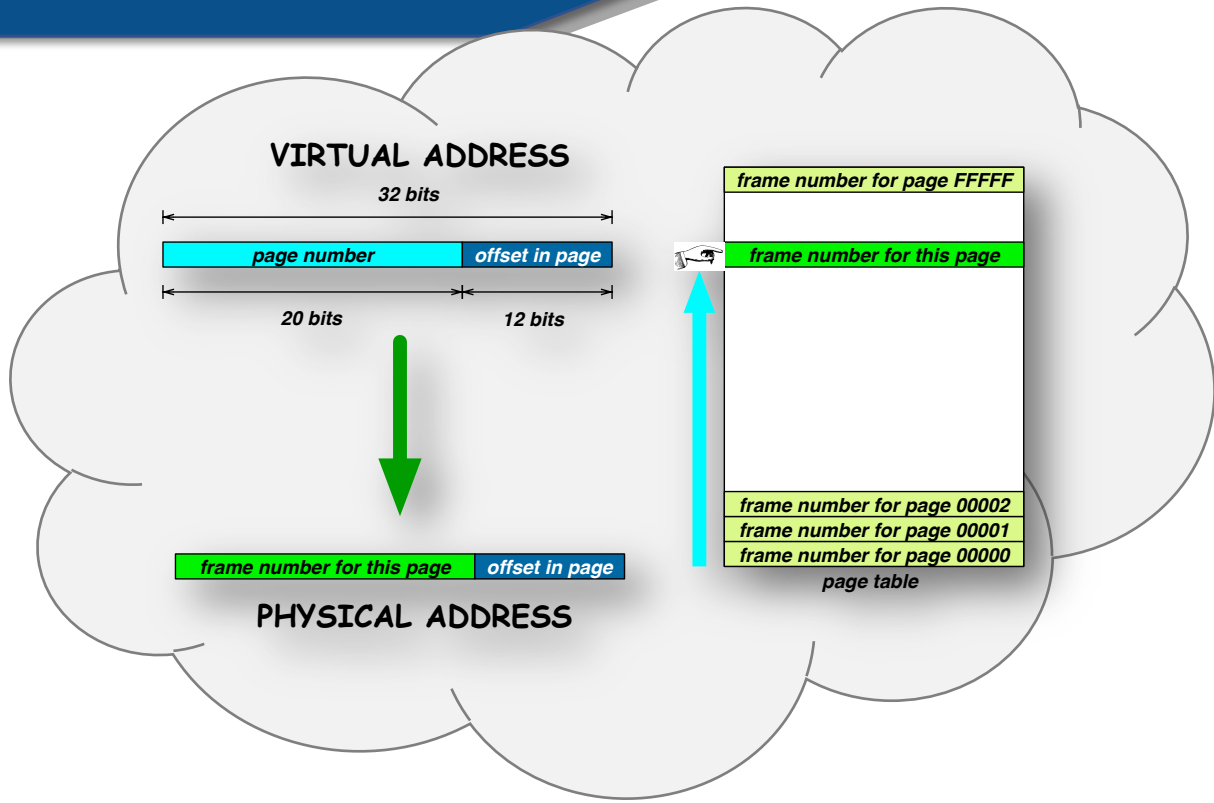
- Outside in physical memory?
  - How does MMU know where the page table is?
  - What about performance...?

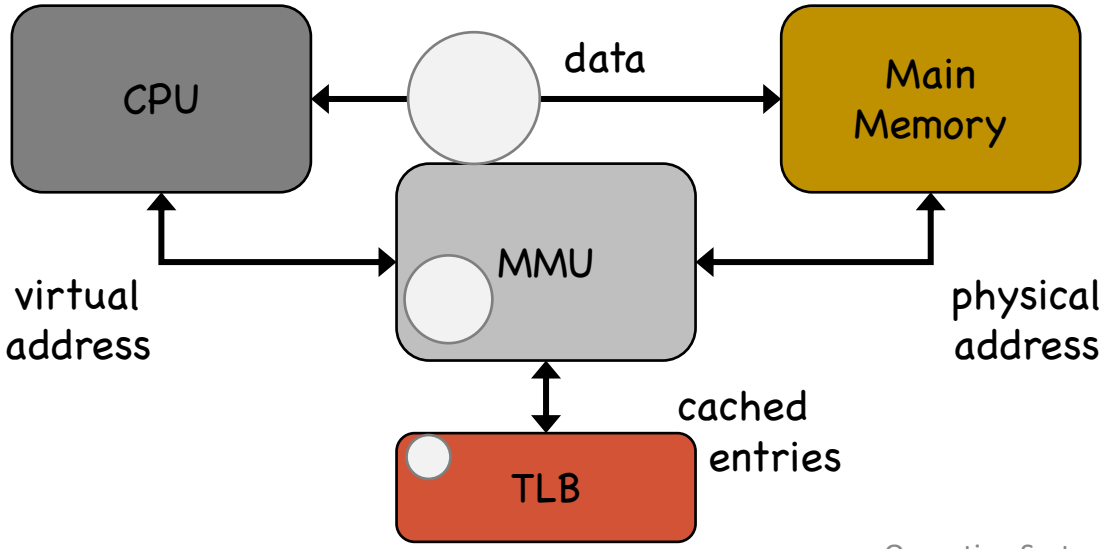


The TLB

Q: Where should the page table live?

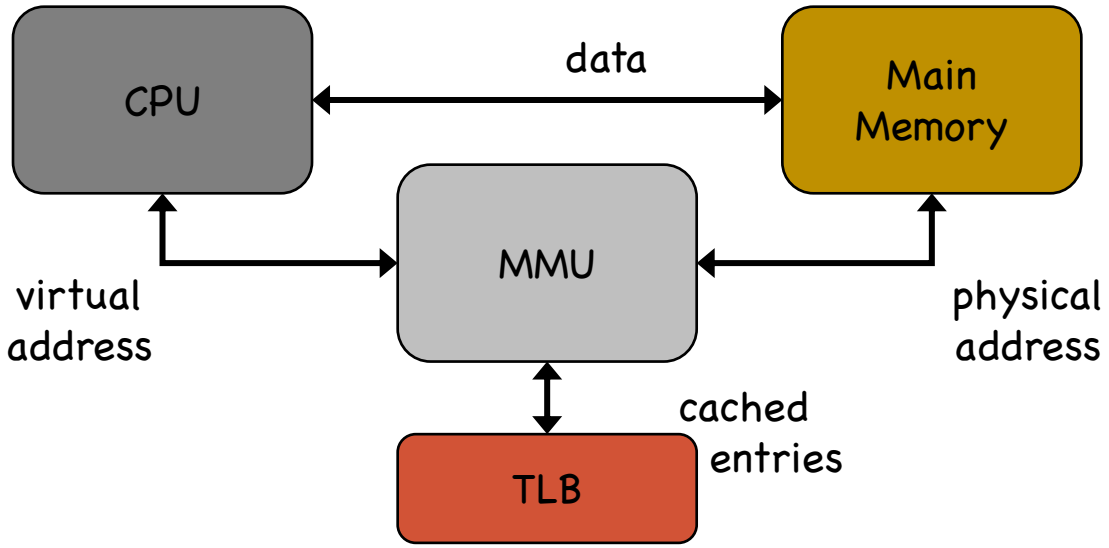
- Within special register(s) inside the MMU?
  - Too small... ☹
- Outside in physical memory?
  - How does MMU know where the page table is?
  - What about performance...?



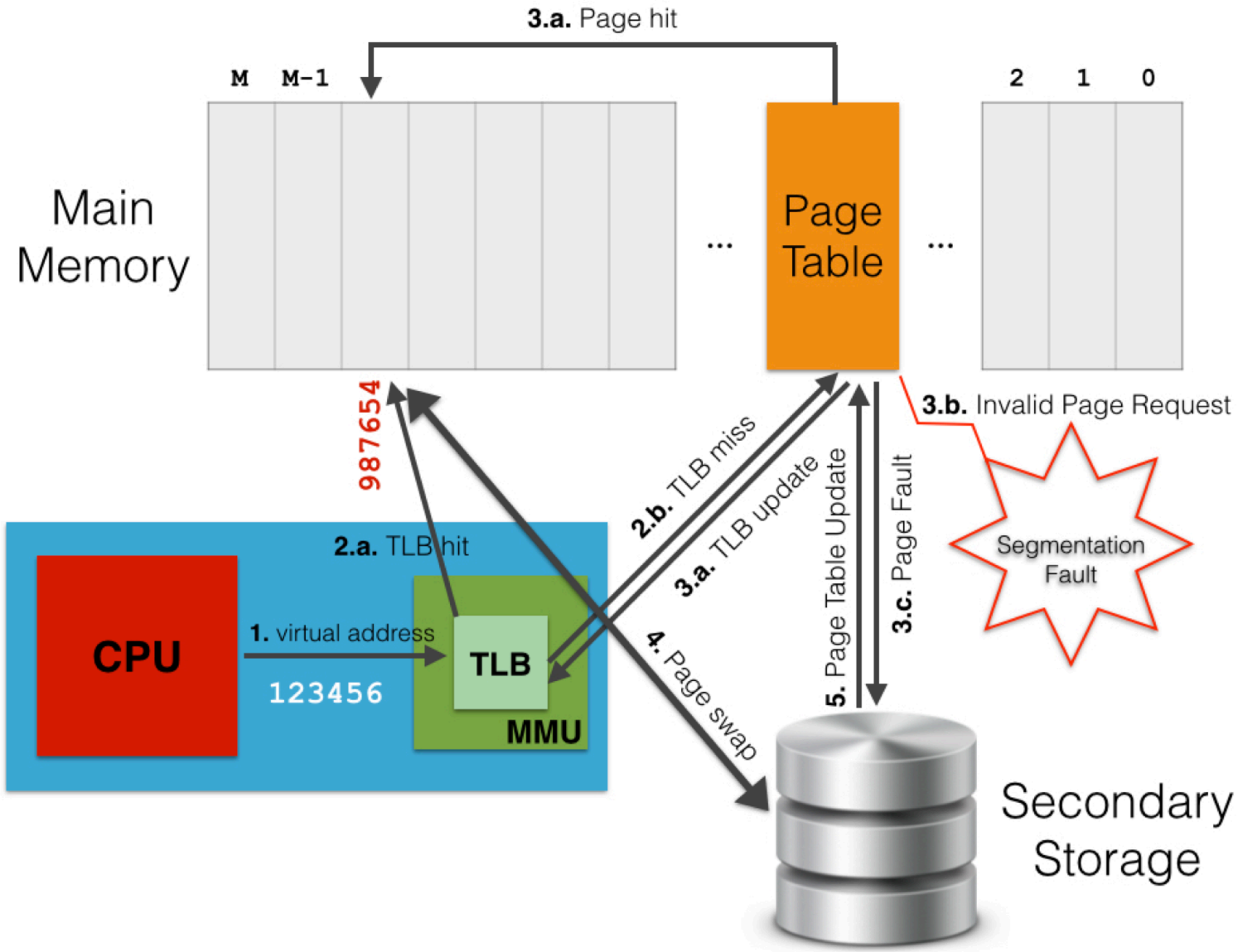


The
Translation Lookaside Buffer (TLB)
to the rescue!

Using a TLB *in general...*



- **TLB as a cache for PTEs**
  - TLB saves N <p,pte> pairs
  - check TLB first...
- **TLB Hit**
  - p exists in TLB
  - quickly returns pte (frame #)
- **TLB Miss**
  - p does not exist in TLB
  - search page table in MM
  - update TLB accordingly
  - ***page fault*** *if pte not in MM!*
  - ***effective access time*** *– see: strider.c*



Further Reading:

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