Virtual Memory I

CSE 351 Summer 2022

Instructor:

Kyrie Dowling

Teaching Assistants:

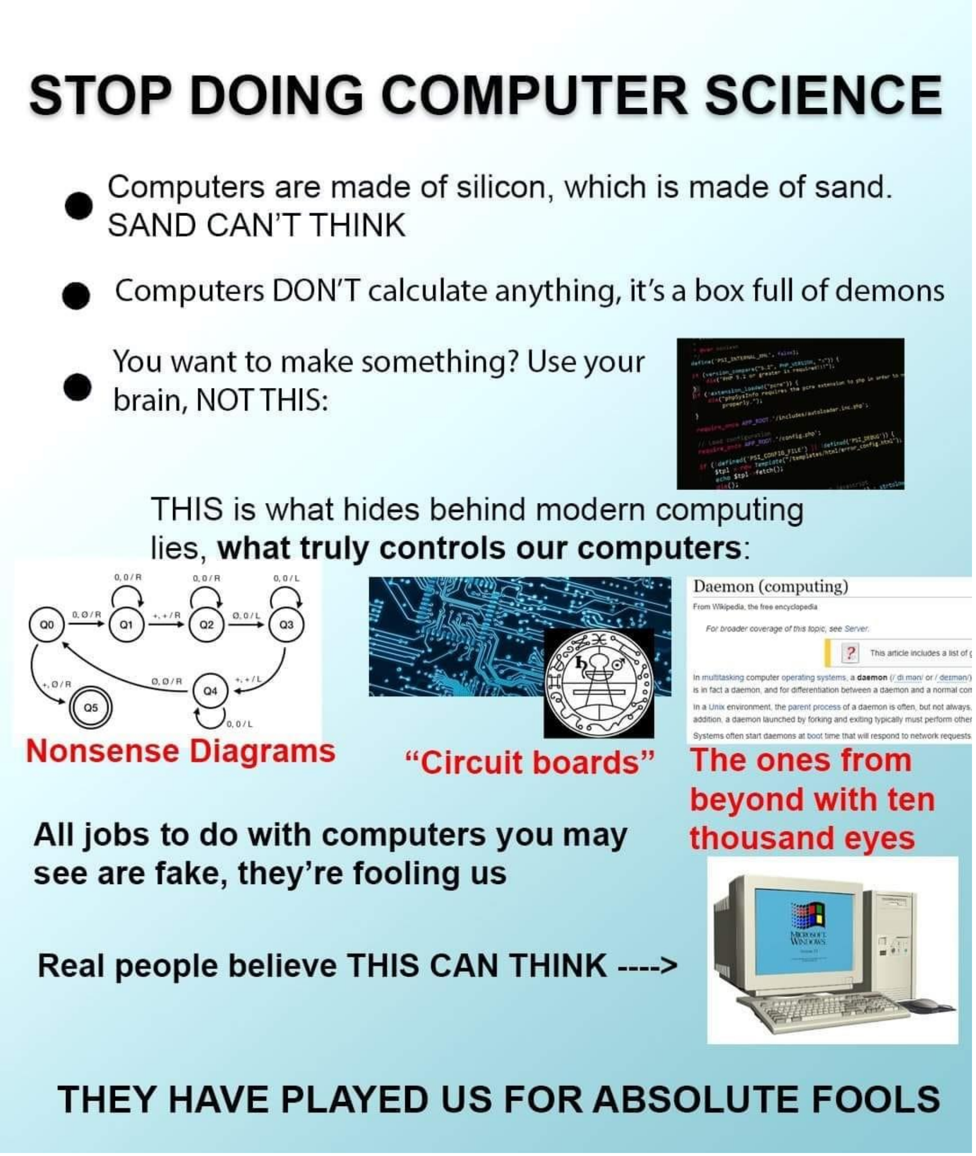
Aakash Srazali

Allie Pfleger

Angela Xu

Dara Stotland

Ellis Haker

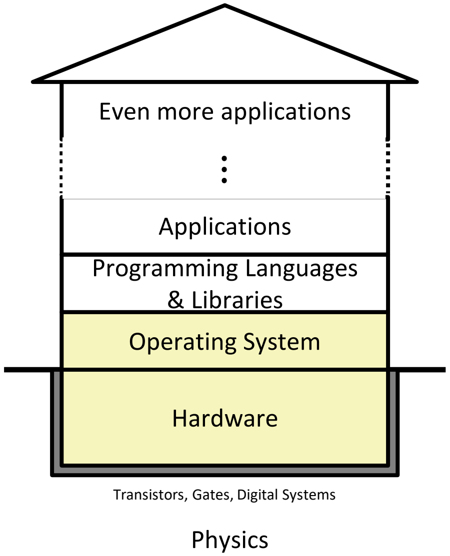


Relevant Course Information

- Only four more homeworks!
  - hw20 due tonight, hw21 due Monday, hw22 due Wednesday, hw23 due Friday
- Lab 4 late deadline tonight
  - The latest you can turn in lab 4 is tonight at 11:59 pm
- Lab 5 released, due next Friday (8/19) at 11:59 pm
  - ***Hard deadline – cannot be turned in late*** (not including extenuating circumstances, email me if there is an emergency)
- Unit Portfolio 3 due next Friday (8/19)
  - No problem videos for this one, only the reflection portion

The Hardware/Software Interface

- Topic Group 3: **Scale & Coherence**
  - Caches, Processes, **Virtual Memory**, Memory Allocation



- How do we maintain logical consistency in the face of more data and more processes?
  - How do we support control flow both within many processes and things external to the computer?
  - How do we support data access, including dynamic requests, across multiple processes?

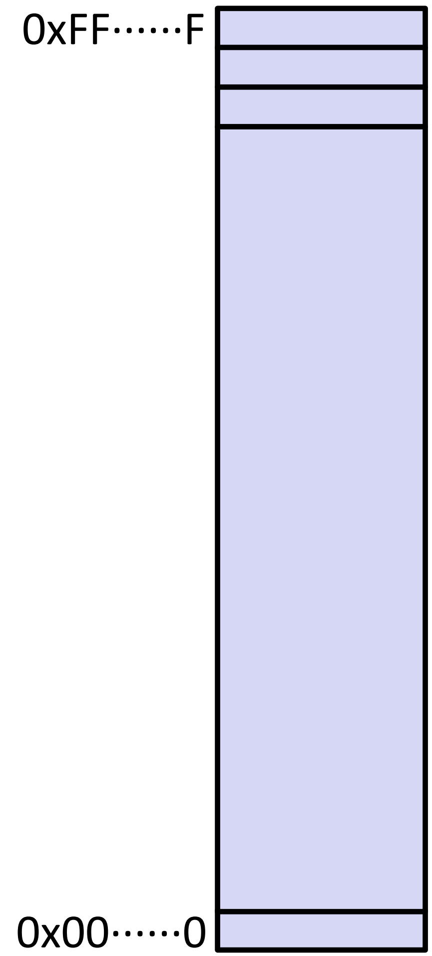
Virtual Memory (VM*)

- **Overview and motivation**
- **VM as a tool for caching**
- Address translation
- VM as a tool for memory management
- VM as a tool for memory protection

**Not to be confused with "Virtual Machine" which is a whole nother thing.*

Memory as we know it so far… is *virtual!*

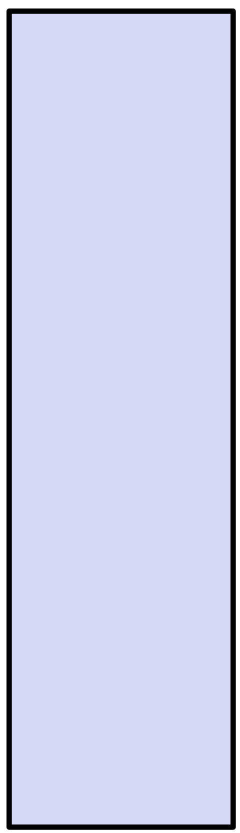
- Programs refer to virtual memory addresses
  - `movq (%rdi),%rax`
  - Conceptually memory is just a very large array of bytes
  - System provides private address space to each process
- Allocation: Compiler and run-time system
  - Where different program objects should be stored
  - All allocation within single virtual address space
- But…
  - We *probably* don't have 2^w bytes of physical memory
  - We *certainly* don't have 2^w bytes of physical memory ***<u>for every process</u>***
  - Processes should not interfere with one another
    - Except in certain cases where they want to share code or data



Problem 1: How Does Everything Fit?

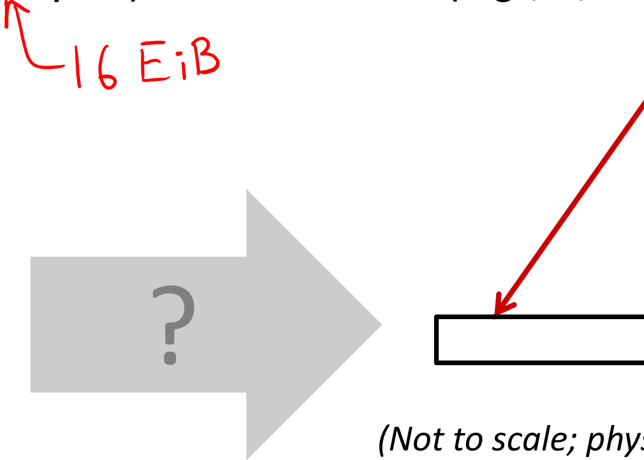
64-bit virtual addresses can address several exabytes (18,446,744,073,709,551,616 bytes)

↑ 16 EiB



Physical main memory offers a few gigabytes (*e.g.*, 8,589,934,592 bytes)

↖ 8 GiB



(Not to scale; physical memory would be smaller than the period at the end of this sentence compared to the virtual address space.)

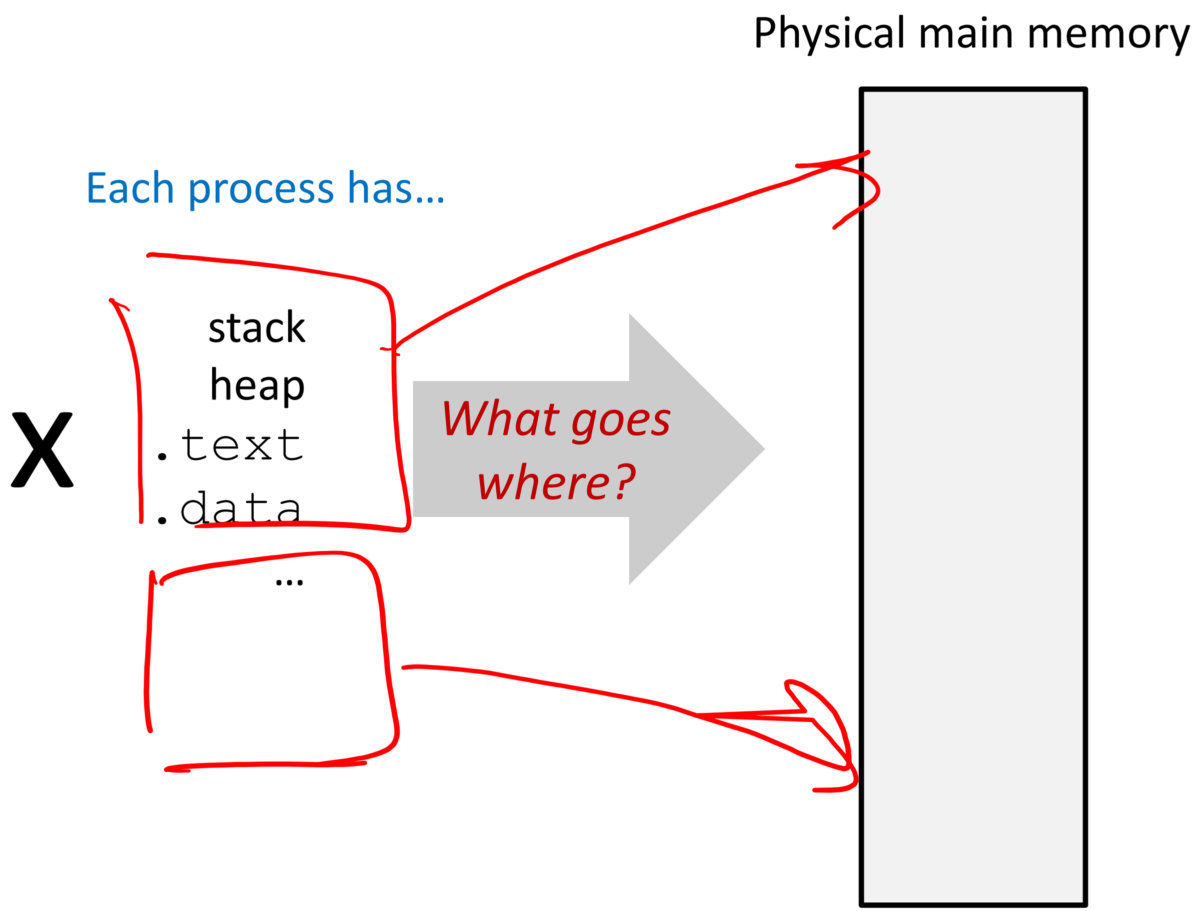
↑ smaller than this!

1 virtual address space per process, with many processes…

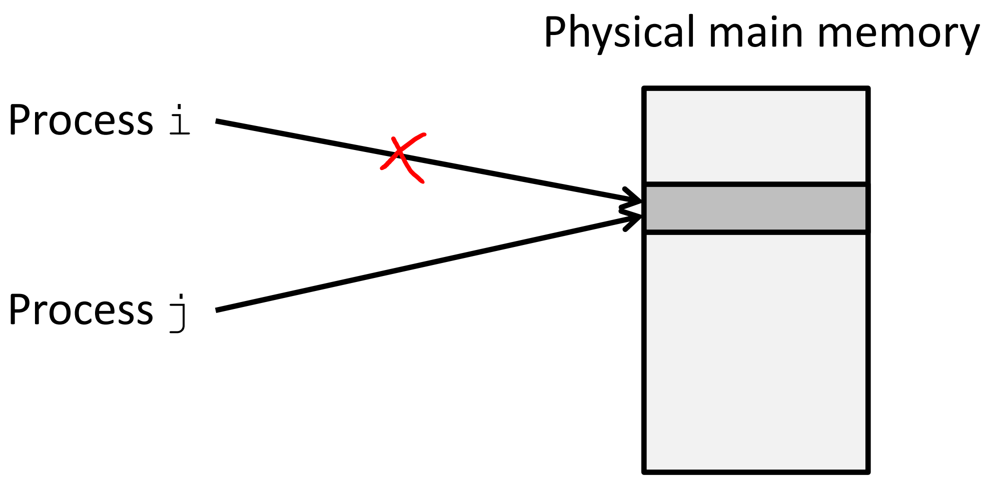
Problem 2: Memory Management

We have multiple processes:

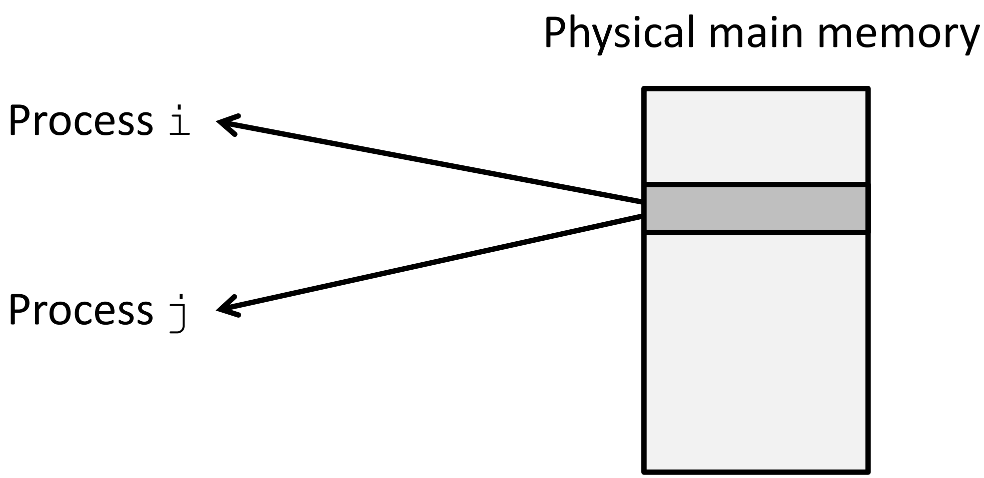
- Process 1
- Process 2
- Process 3
- …
- Process n



Problem 3: How To Protect

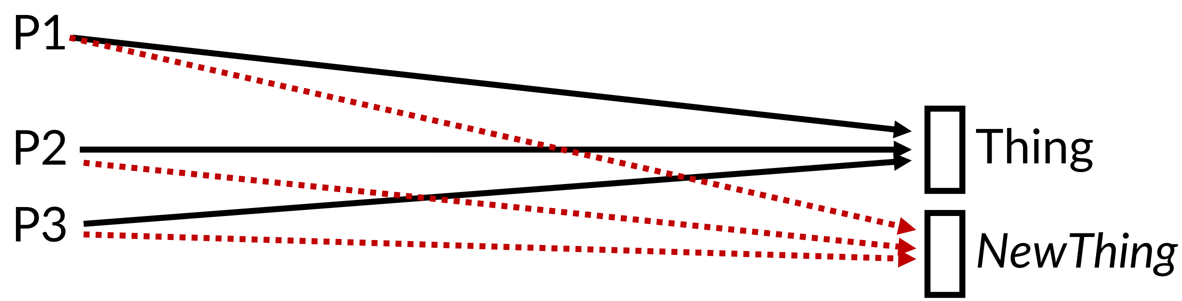


Problem 4: How To Share?

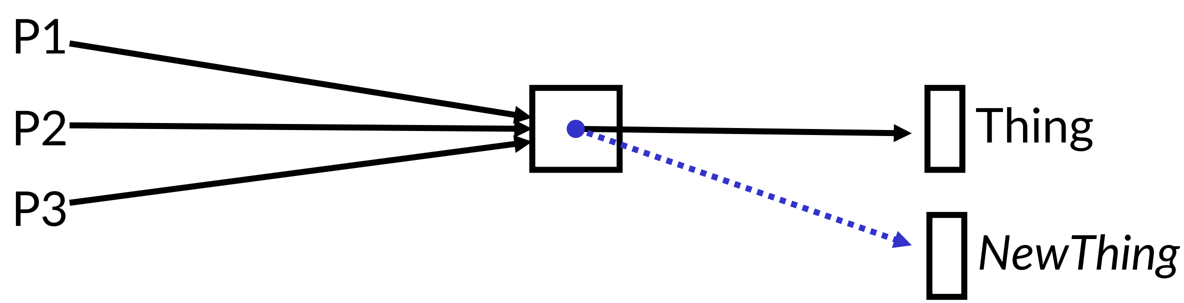


How can we solve these problems?

- "Any problem in computer science can be solved by adding another level of **indirection**." *– David Wheeler, inventor of the subroutine*
- Without Indirection



- With Indirection

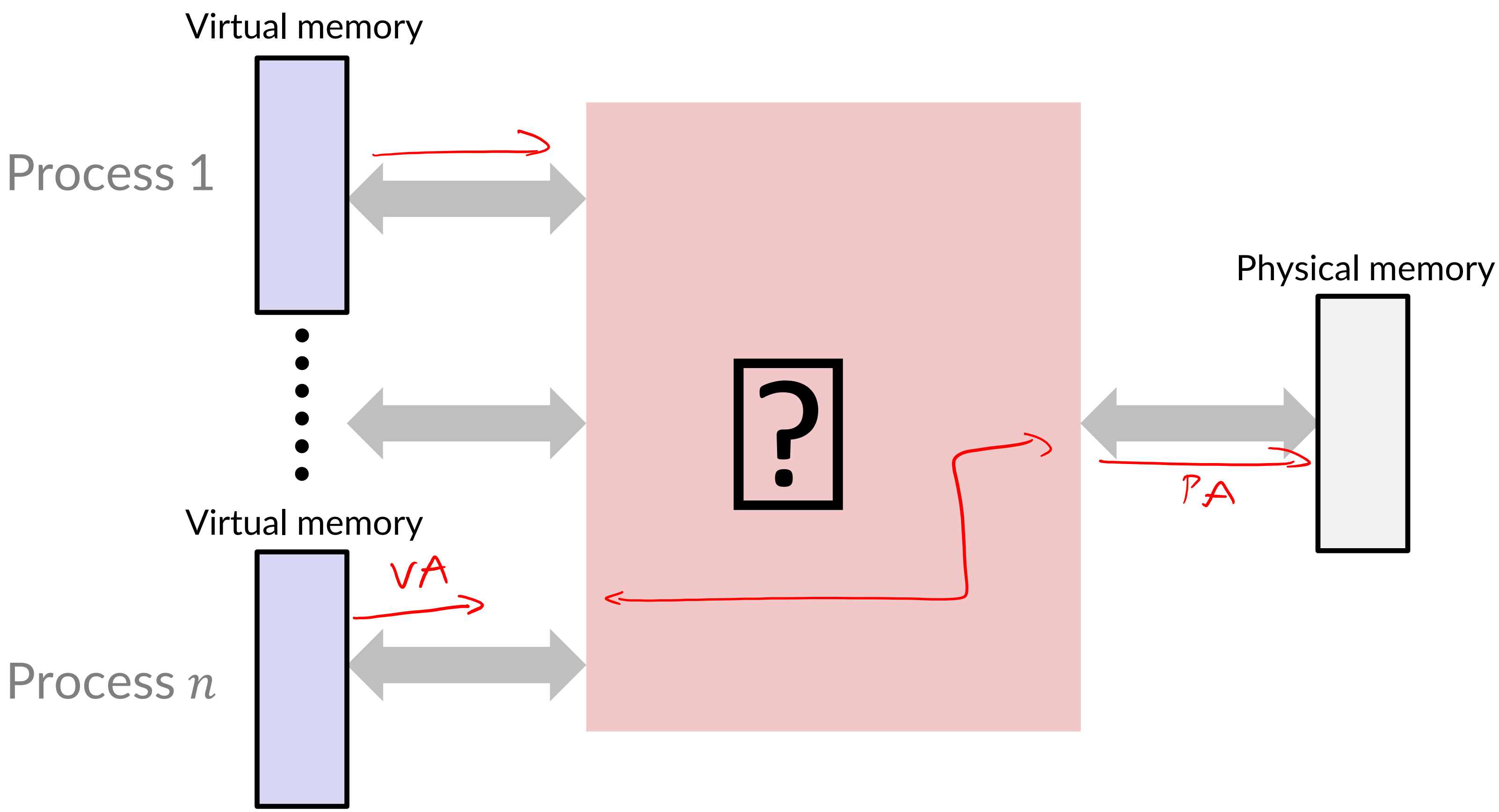


What if I want to move Thing?

Indirection

- *Indirection*: The ability to reference something using a name, reference, or container instead of the value itself. A flexible mapping between a name and a thing allows changing the thing without notifying holders of the name.
  - − Adds some work (now must look up 2 things instead of 1)
  - + But don't have to track all uses of name/address (single source!)
- Examples:
  - **Phone system:** cell phone number portability
  - **Domain Name Service (DNS):** translation from name to IP address
  - **Call centers:** route calls to available operators, etc.
  - **Dynamic Host Configuration Protocol (DHCP):** local network address assignment

Indirection in Virtual Memory



- Each process gets its own private virtual address space
- Solves the previous problems!

Address Spaces

bits → $n = \lceil \log_2 N \rceil$ ← ceiling function

- Virtual address space: Set of $N = 2^n$ virtual addr
  - {0, 1, 2, 3, …, N-1}

bytes → $m = \lceil \log_2 M \rceil$

- Physical address space: Set of $M = 2^m$ physical addr
  - {0, 1, 2, 3, …, M-1}

- Every byte in main memory has:
  - one physical address (PA)
  - zero, one, *or more* virtual addresses (VAs)

unused — used by one process — used by many processes

Polling Questions

- On a 64-bit machine currently running 8 processes, how much virtual memory is there?

word size is 64 bits, so $n = 64$ and $N = 2^{64}$ bytes per process.

$2^{64} \times 8 = 2^{67}$ bytes of virtual memory

- True or False: A 32-bit machine with 8 GiB of RAM installed would never use all of it (in theory).

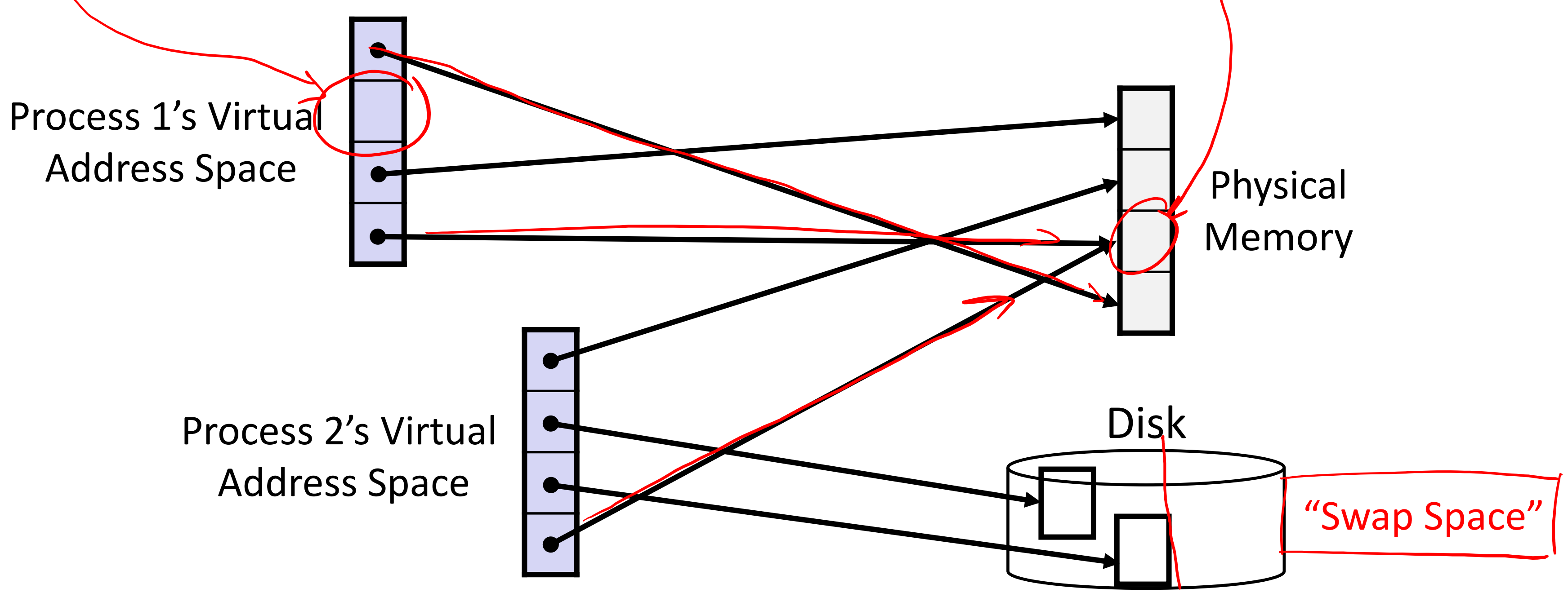
word size is 32 bits, so each process has 2^{32} bytes = 4 GiB of virtual memory

however, we have more than 1 process, so we can easily use up all 8 GiB of physical memory

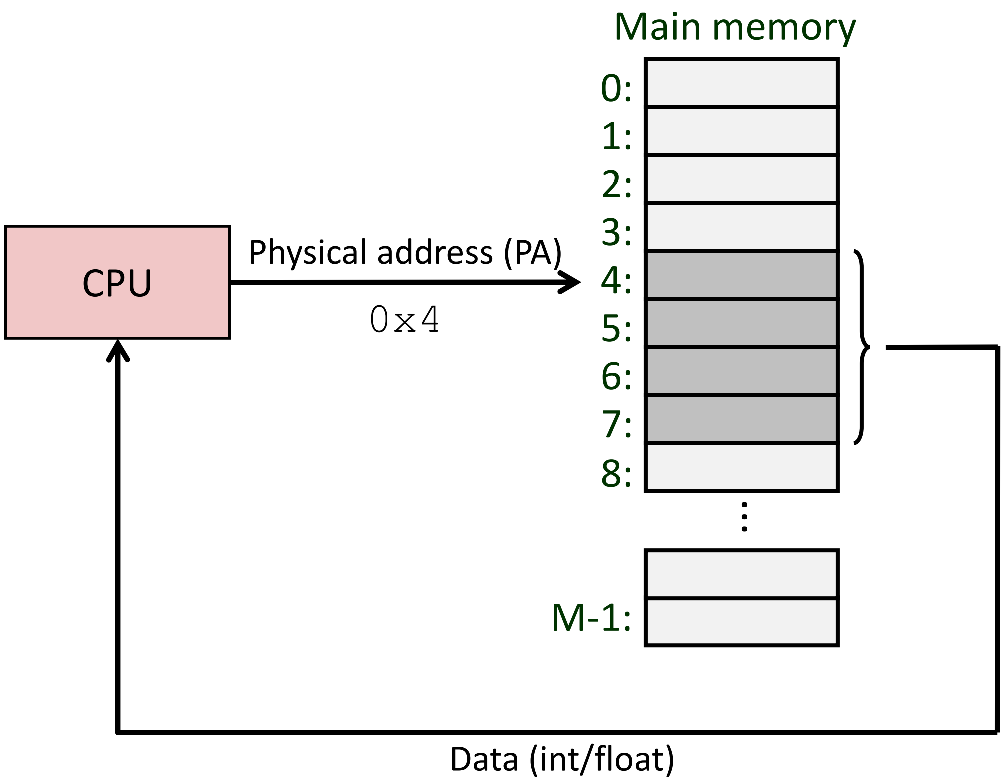
note: there are other limitations, (e.g., motherboard, OS) that restrict the maximum amount of usable RAM in practice

Mapping

- A virtual address (VA) can be mapped to either physical memory or disk
  - Unused VAs may not have a mapping
  - VAs from *different* processes may map to same location in memory/disk

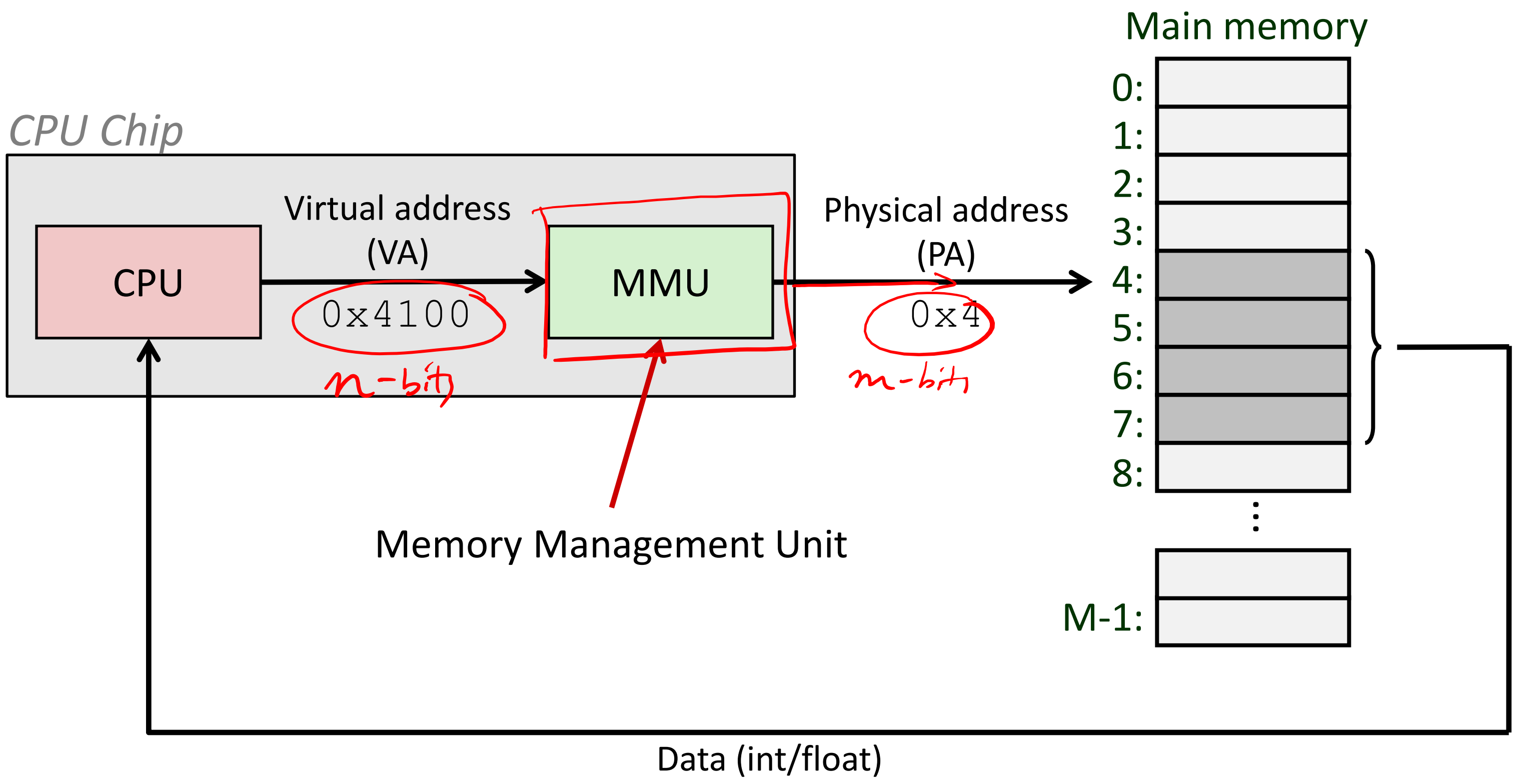


A System Using Physical Addressing



- Used in "simple" systems with (usually) just one process:
  - Embedded microcontrollers in devices like cars, elevators, and digital picture frames

A System Using Virtual Addressing



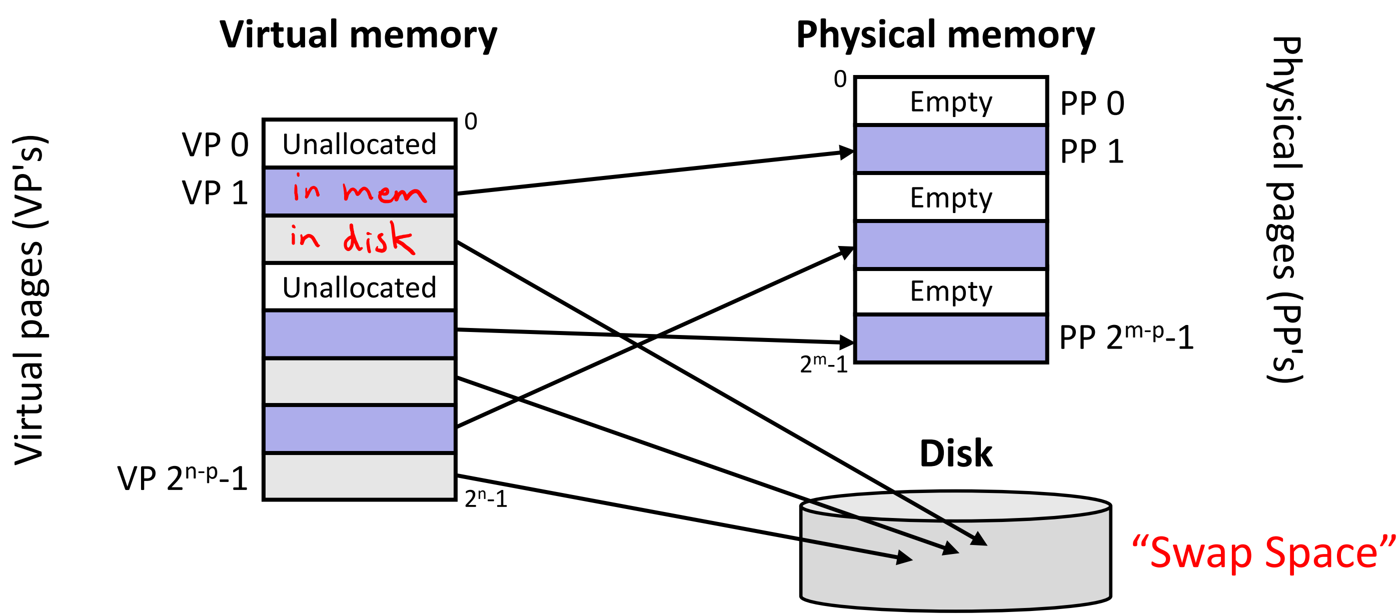
- Physical addresses are *completely invisible to programs*
  - Used in all modern desktops, laptops, servers, smartphones…

Why Virtual Memory (VM)?

- Efficient use of limited main memory (RAM)
  - Use RAM as a cache for the parts of a virtual address space
    - Some non-cached parts stored on disk
    - Some (unallocated) non-cached parts stored nowhere
  - Keep only active areas of virtual address space in memory
    - Transfer data back and forth as needed
- Simplifies memory management for programmers
  - Each process "gets" the same full, private linear address space
- Isolates address spaces (protection)
  - One process can't interfere with another's memory
    - They operate in *different address spaces*
  - User process cannot access privileged information
    - Different sections of address spaces have different permissions

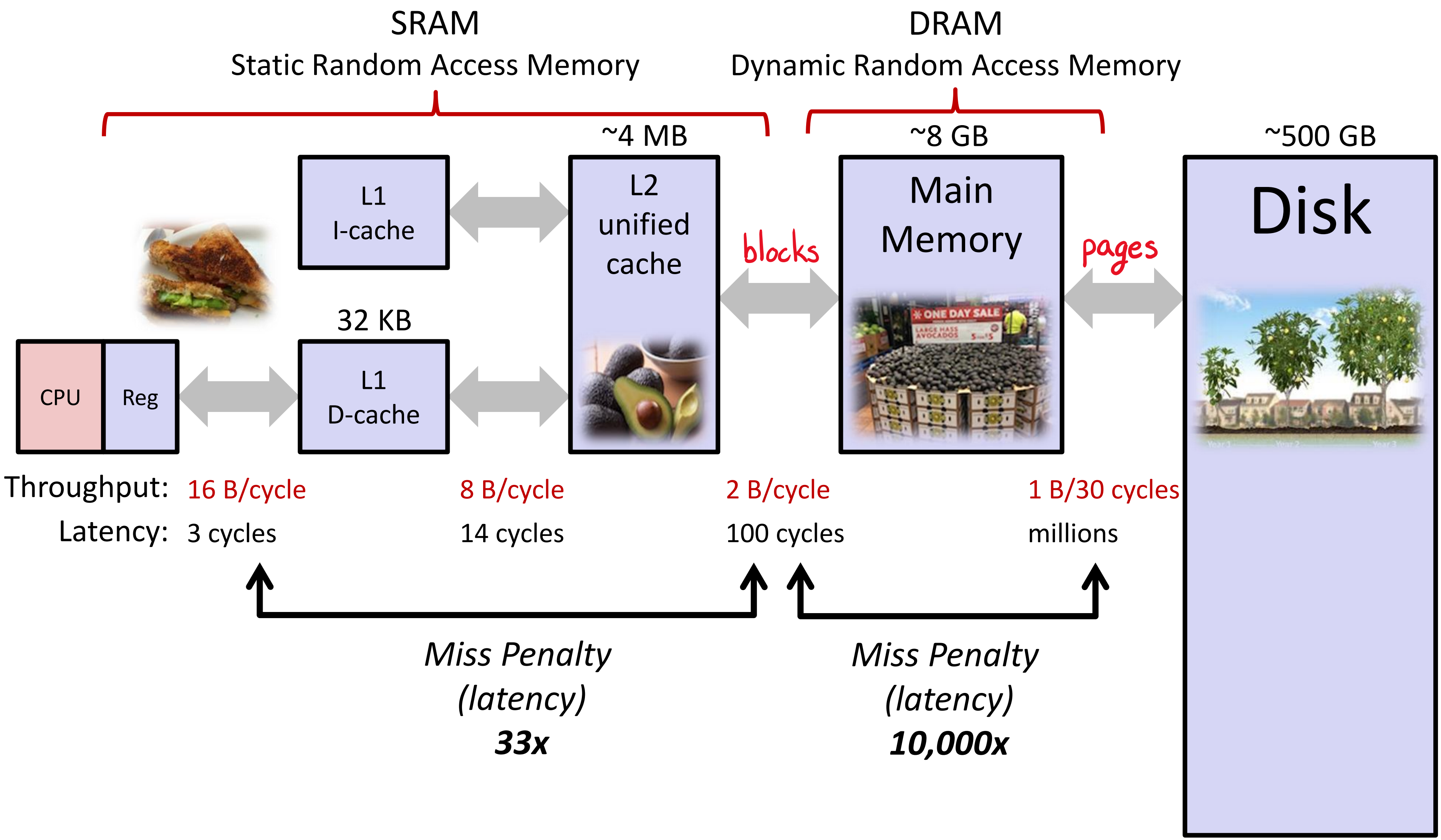
VM and the Memory Hierarchy

- Think of memory (virtual or physical) as an array of bytes, now split into *pages*
  - Pages are another unit of aligned memory (size is $P = 2^p$ bytes) $p = \lceil \log_2 P \rceil$
  - Each virtual page can be stored in *any* physical page (no fragmentation!) no wasted space/gaps
- Pages of virtual memory are usually stored in physical memory, but sometimes spill to disk



Memory Hierarchy: Core 2 Duo

Not drawn to scale



Virtual Memory Design Consequences

- Large page size: typically 4-8 KiB or 2-4 MiB
  - *Can* be up to 1 GiB (for "Big Data" apps on big computers)
  - Compared with 64-byte cache blocks
- Fully associative (physical memory is a single set)
  - Any virtual page can be placed in any physical page
  - Requires a "large" mapping function – different from CPU caches
- Highly sophisticated, expensive replacement algorithms in OS
  - Too complicated and open-ended to be implemented in hardware
- *Write-back* rather than *write-through* (track dirty pages)
  - *Really* don't want to write to disk every time we modify memory
  - Some things may never end up on disk (*e.g.*, stack for short-lived process)

Why does VM work on RAM/disk?

- Avoids disk accesses because of *locality*
  - Same reason that L1 / L2 / L3 caches work

- The set of virtual pages that a program is "actively" accessing at any point in time is called its *working set*
  - If (*working set of one process* ≤ *physical memory*):
    - Good performance for one process (after compulsory misses)
  - If (*working sets of all processes* > *physical memory*):
    - ***Thrashing:*** Performance meltdown where pages are swapped between memory and disk continuously (CPU always waiting or paging)
    - This is why your computer can feel faster when you add RAM

Virtual Memory (VM)

- Overview and motivation
- VM as a tool for caching
- **Address translation**
- VM as a tool for memory management
- VM as a tool for memory protection

Reading Review

- Terminology:
  - Paging: page size (P), page offset width (p) virtual page number (VPN), physical page numbers (PPN)
  - Page table (PT): page table entry (PTE), access rights (read, write, execute)

- Questions from the Reading?

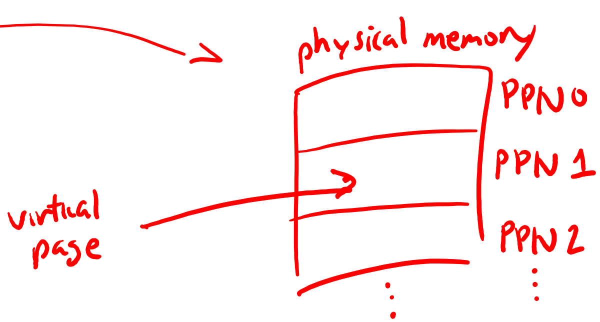
Review Questions

- Which terms from caching are most similar/analogous to the new virtual memory terms?
  - page size
    block size
  - page offset width
    (block) offset width
  - virtual page number
    block number
  - physical page number
    block number or cache set
  - page table entry
    cache line: data of interest + management bits
  - access rights
    management bits

VA: | virtual page number | page offset |

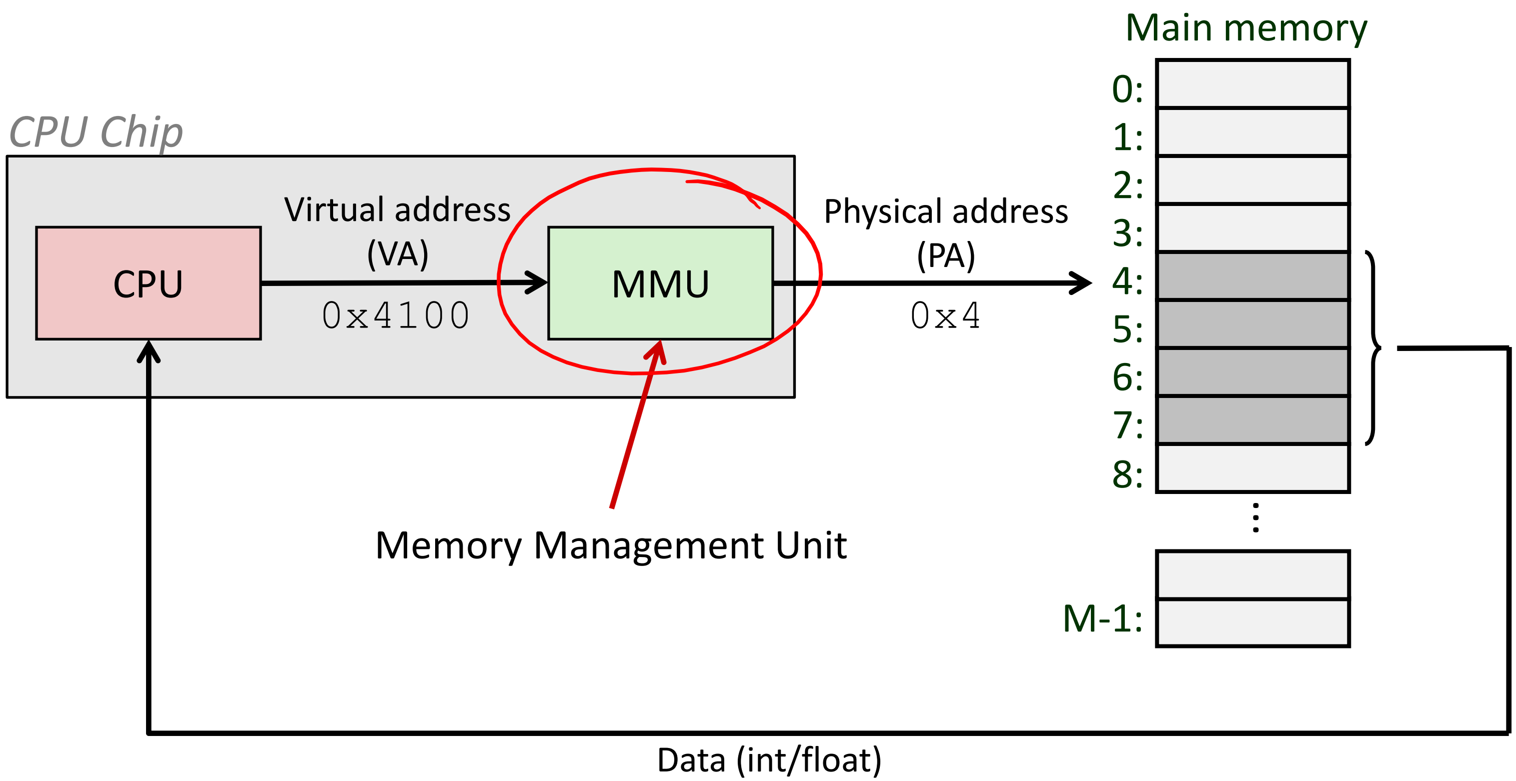
PA: | block number | offset |

PA: | physical page num | page offset |



Address Translation

How do we perform the virtual → physical address translation?



Address Translation: Page Tables

VPN width $n-p$ ⟺ we have 2^{n-p} pages in VA space

Page size P bytes ⟺ $p = \lceil \log_2 P \rceil$ bits

- CPU-generated address can be split into:

n-bit address:

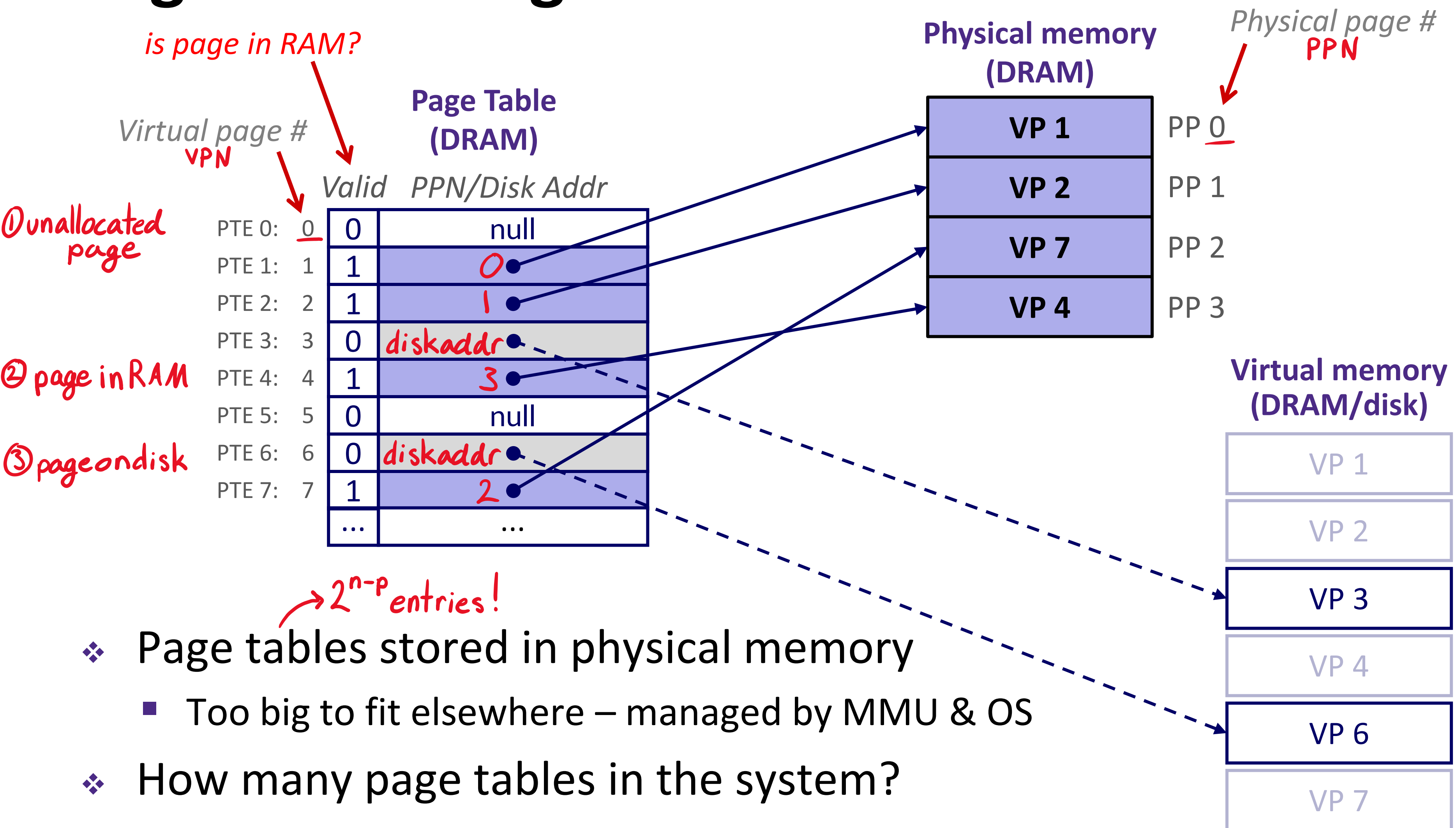
$n-p$ bits	p bits
Virtual Page Number	Page Offset
block number	block offset

analogus to: (block number / block offset) for caches

  - Request is Virtual Address (VA), want Physical Address (PA)
  - Note that Physical Offset = Virtual Offset (page-aligned)

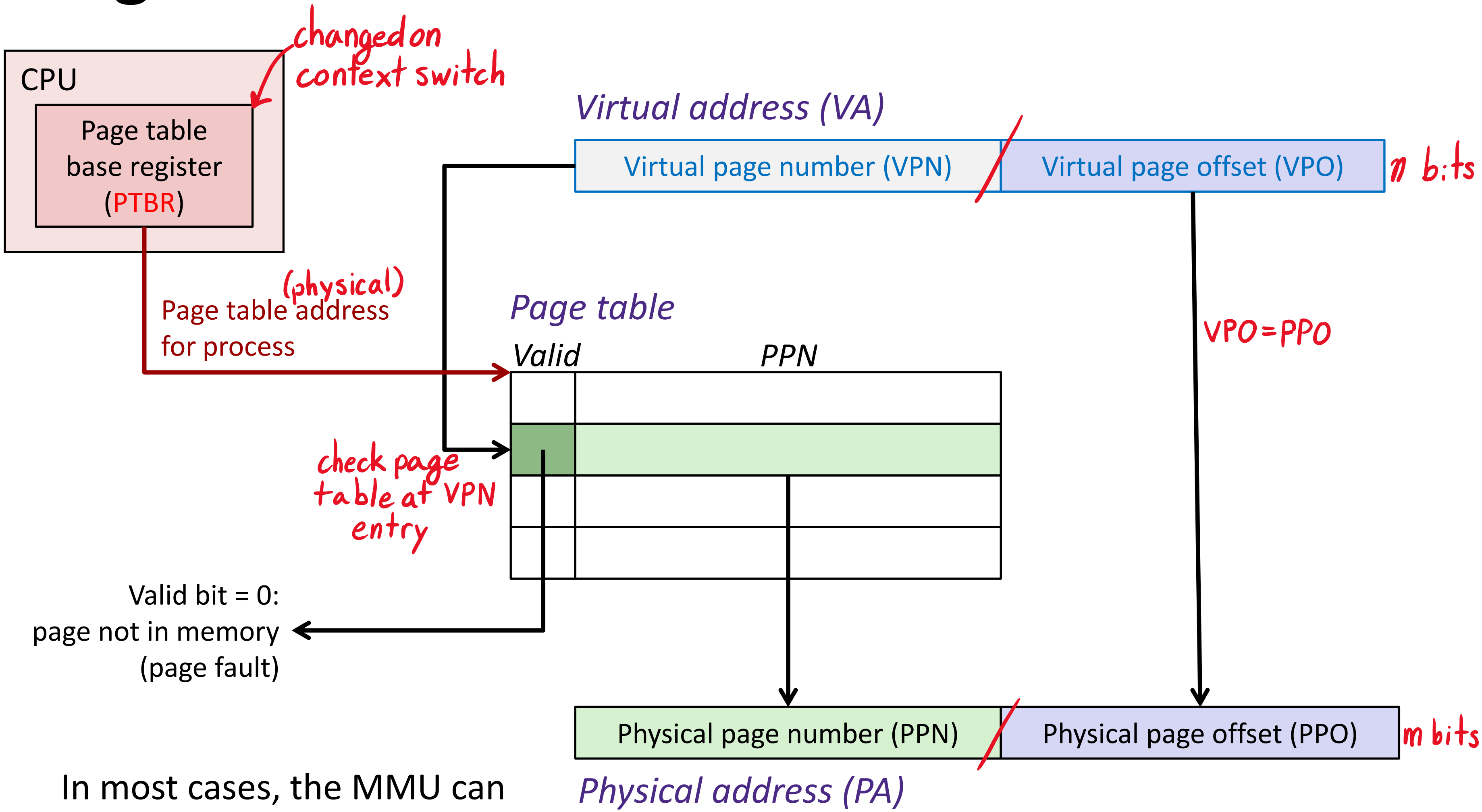
- Use lookup table that we call the *page table* (PT)
  - Replace Virtual Page Number (VPN) with Physical Page Number (PPN) to generate Physical Address
  - Index PT using VPN: page table entry (PTE) stores the PPN plus management bits (*e.g.*, Valid, Dirty, access rights)
  - Has an entry for *every* virtual page

Page Table Diagram



- Page tables stored in physical memory
  - Too big to fit elsewhere – managed by MMU & OS
- How many page tables in the system?
  - *One per process*

Page Table Address Translation



In most cases, the MMU can perform this translation without software assistance

Polling Question

- How many bits wide are the following fields?
  - 16 KiB pages (2^4 2^{10}) — $p = 14$ bits
  - 48-bit virtual addresses — $n = 48$ bits
  - 16 GiB physical memory (2^4 2^{30}) — $m = 34$ bits

VA: [VPN | PO]

VPN = $n - p = 34$ bits $\leftrightarrow 2^{34}$ pages in virtual address space

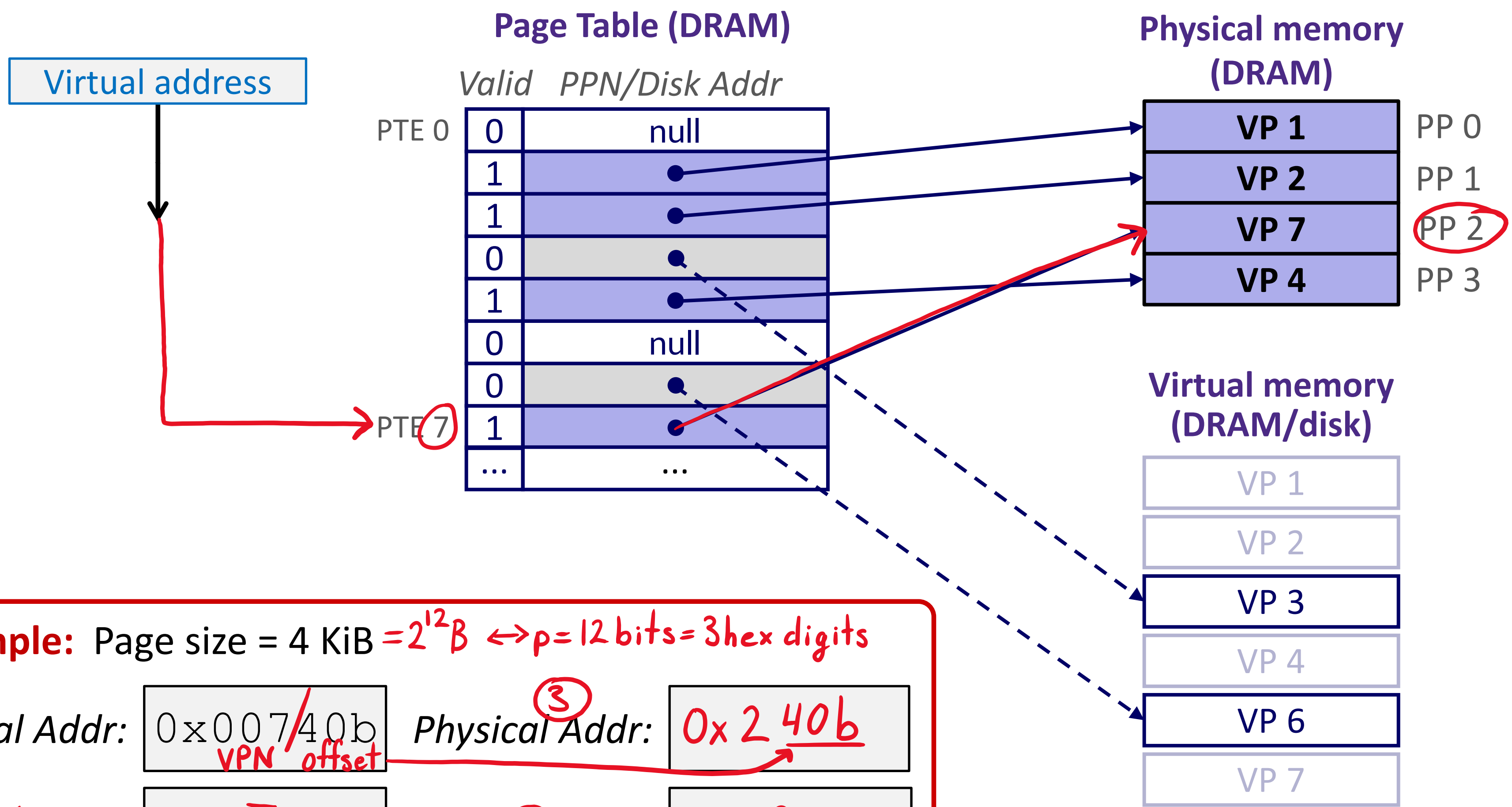
PA: [PPN | PO]

PPN = $m - p = 20$ bits $\leftrightarrow 2^{20}$ pages in physical address space

	VPN	PPN
(A)	34	24
(B)	32	18
(C)	30	20
(D)	34	20

Page Hit

- ***Page hit:*** VM reference is in physical memory



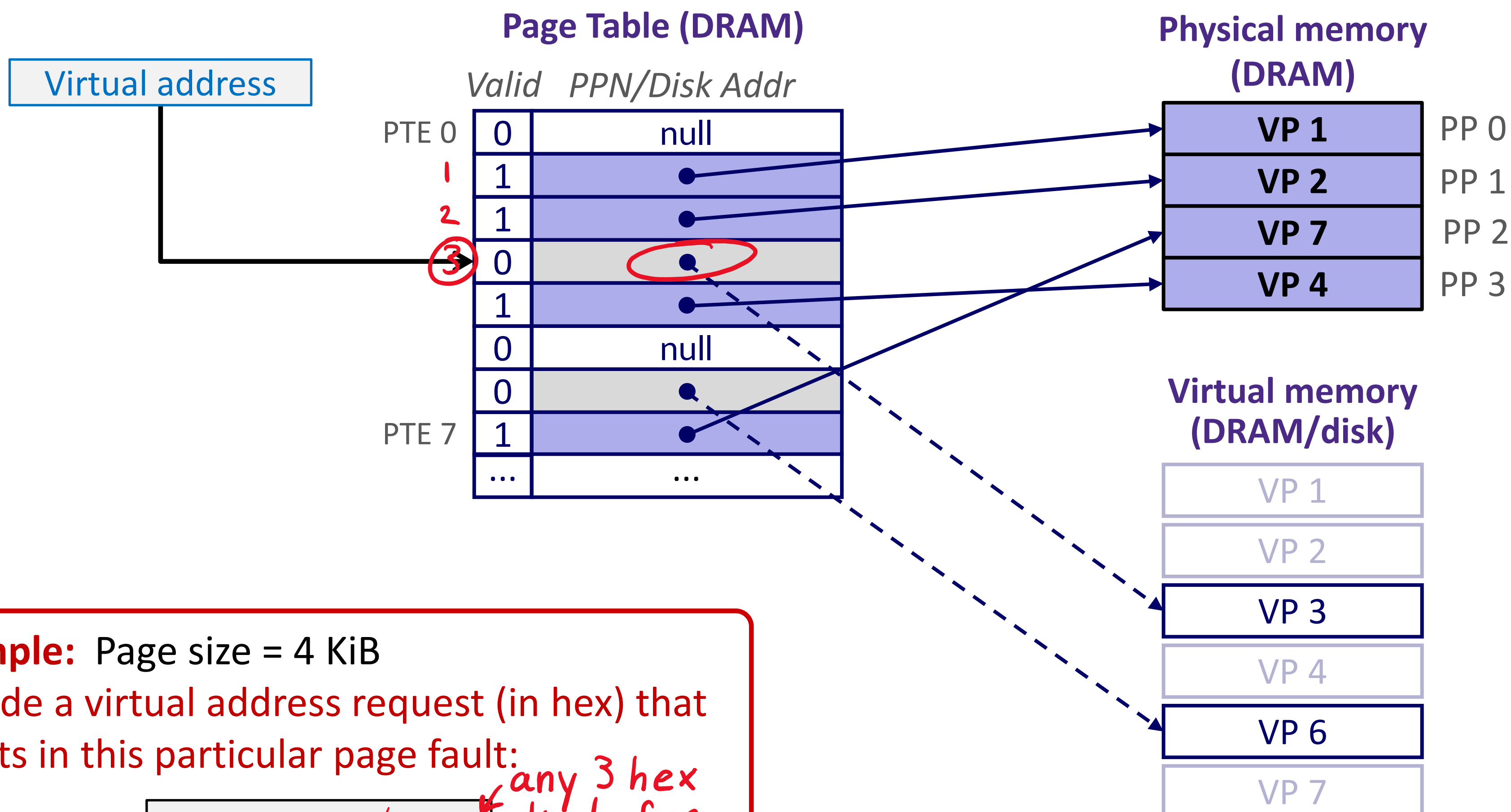
Example: Page size = 4 KiB $= 2^{12}$ B ↔ p = 12 bits = 3 hex digits

Virtual Addr: 0x00740b (VPN | offset)   ③ *Physical Addr:* 0x 2 40b

① *VPN:* 7   ② *PPN:* 2

Page Fault

- ***Page fault:*** VM reference is NOT in physical memory



Example: Page size = 4 KiB

Provide a virtual address request (in hex) that results in this particular page fault:

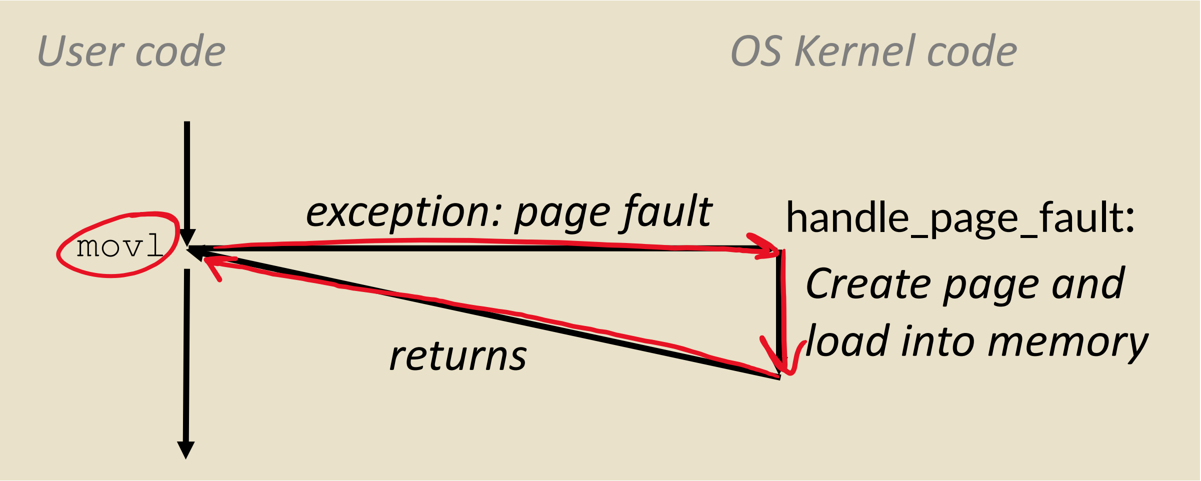
Virtual Addr: 0x003/000 ← any 3 hex digits for offset

Reminder: Page Fault Exception

- User writes to memory location
- That portion (page) of user's memory is currently on disk

```
int a[1000];
int main () {
    a[500] = 13;
}
```

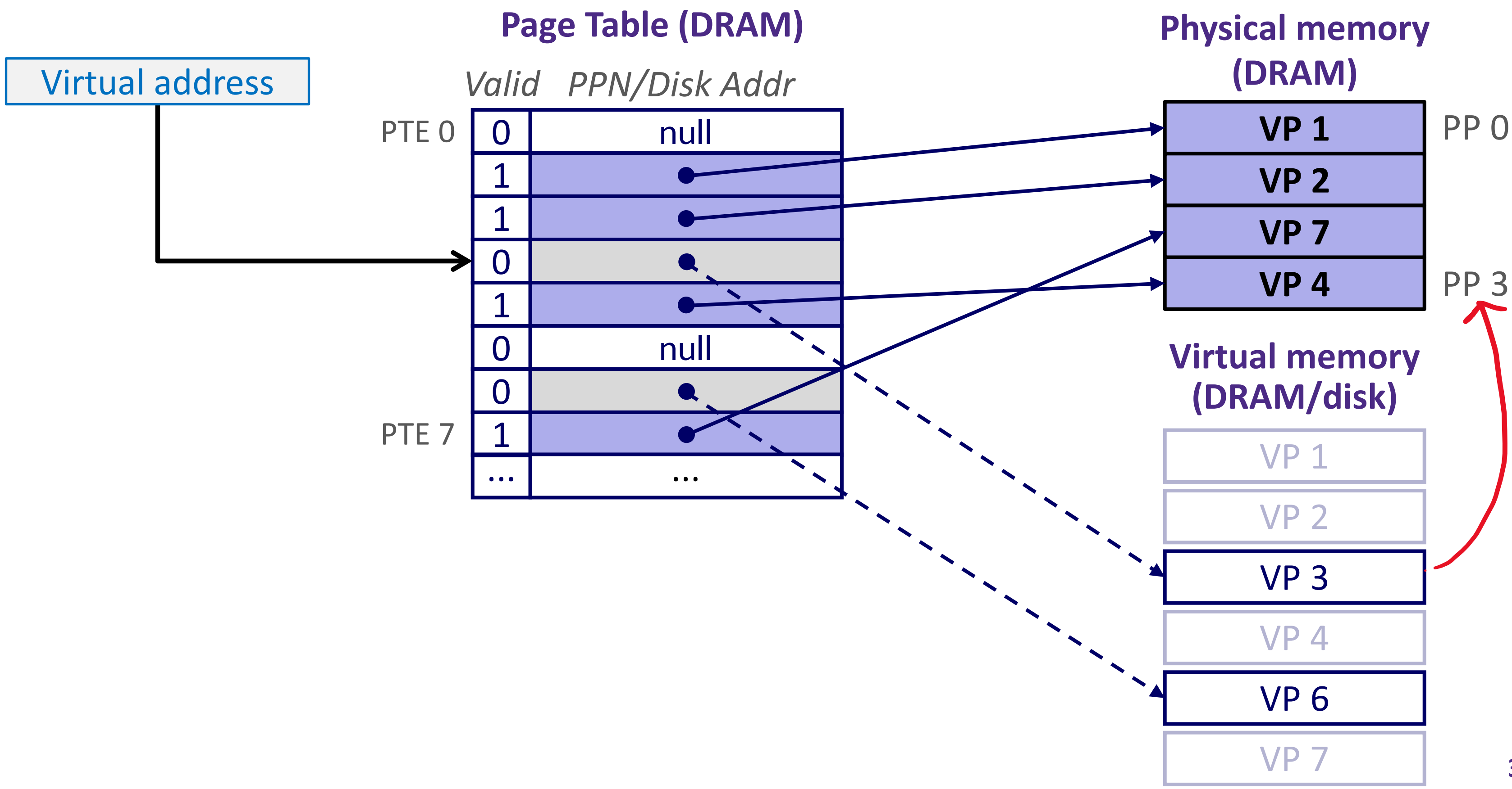
```
80483b7:        c7 05 10 9d 04 08 0d    movl    $0xd,0x8049d10
```



- Page fault handler must load page into physical memory
- Returns to faulting instruction: `mov` is executed again!
  - Successful on second try

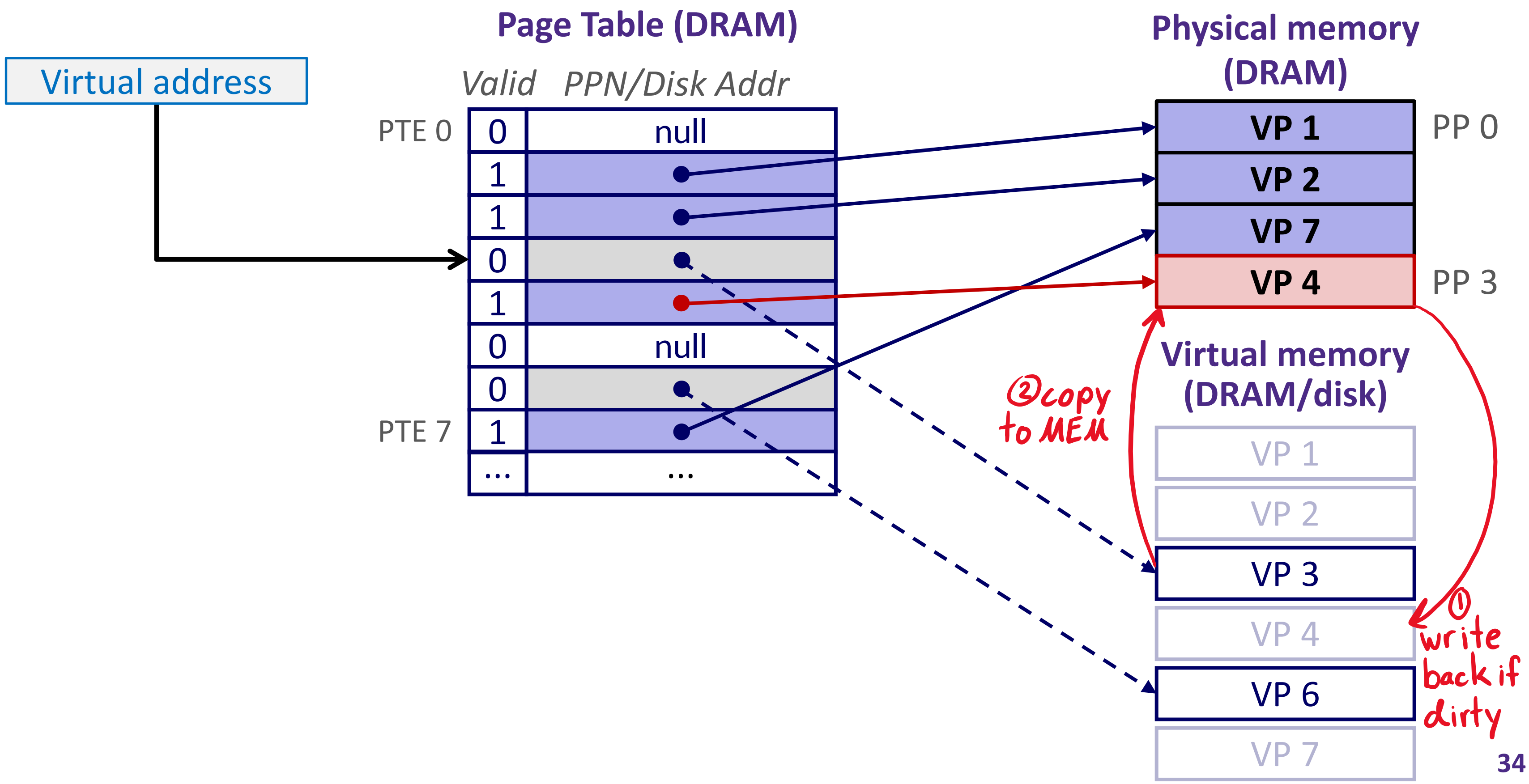
Handling a Page Fault

- Page miss causes page fault (an exception)



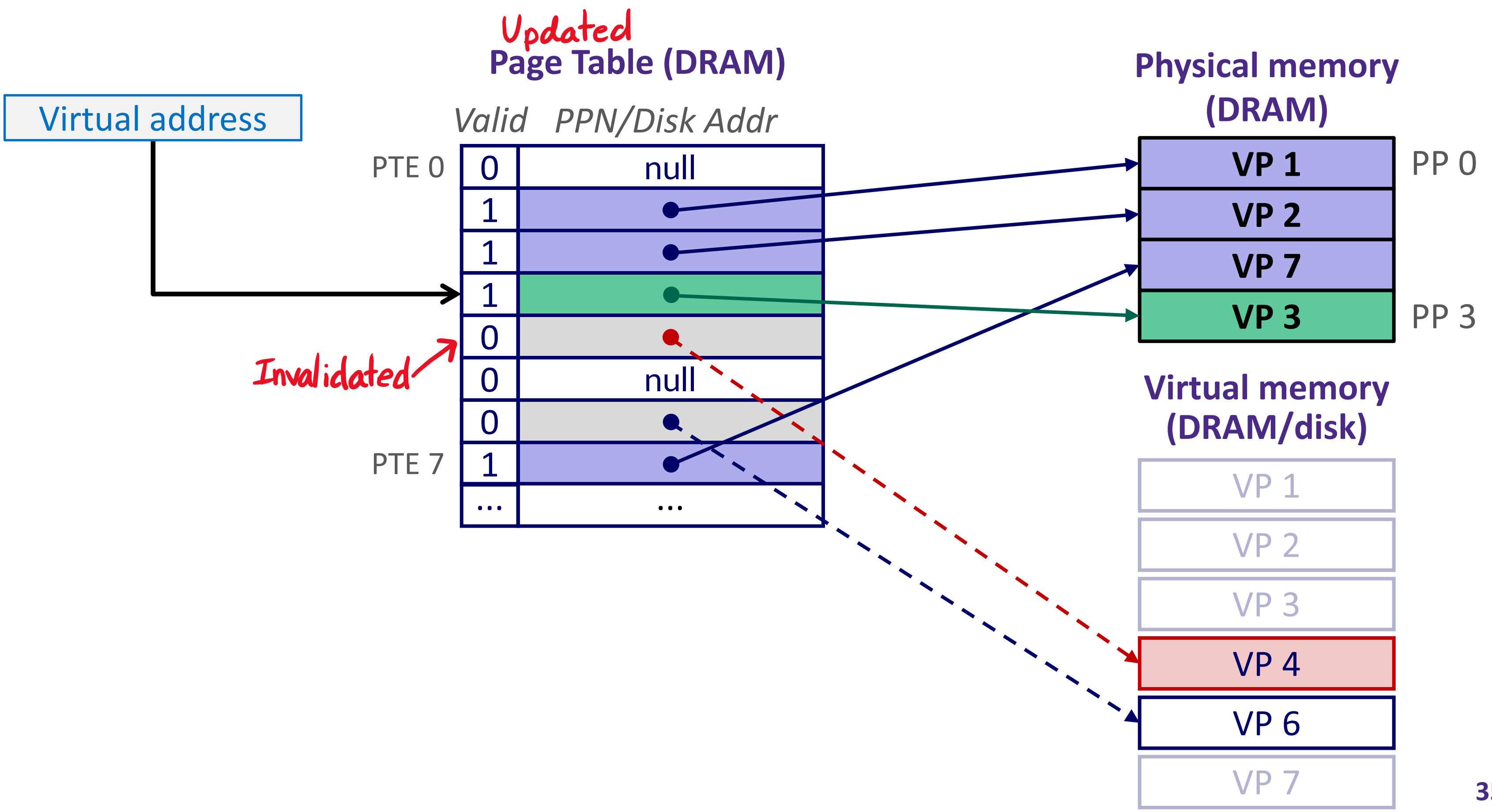
Handling a Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a *victim* to be evicted (here VP 4)



Handling a Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a *victim* to be evicted (here VP 4)



Handling a Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a *victim* to be evicted (here VP 4)
- Offending instruction is restarted: page hit!