

Virtual Memory II

CSE 351 Winter 2021

Instructor:

Mark Wyse

Teaching Assistants:

Kyrie Dowling

Catherine Guevara

Ian Hsiao

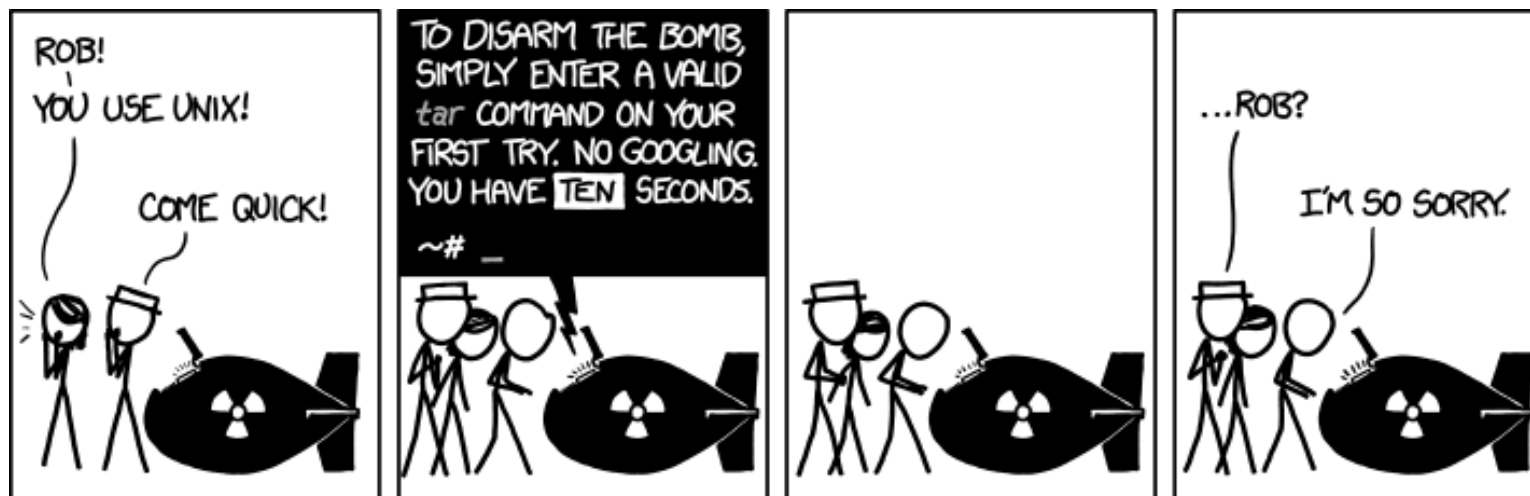
Jim Limprasert

Armin Magness

Allie Pfleger

Cosmo Wang

Ronald Widjaja



<http://xkcd.com/1831/>

Administrivia

- ❖ hw18 due *Tonight*, hw19 due Monday (3/1)
- ❖ hw20 due next Friday (3/5)
 - larger homework on VM II & III, start early!
- ❖ Study Guide 2 due Monday (3/1)
- ❖ Lab 4 due next Friday (3/5)

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)

$$p = \log_2 P$$

❖ Questions from the Reading?

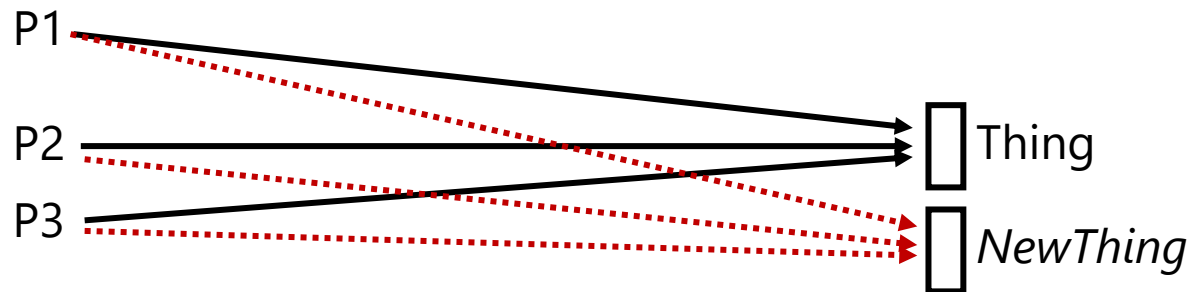
VM Motivation - Four Problems

- ❖ Problem 1: How does it all fit?
 - Virtual Memory >> Physical Memory
- ❖ Problem 2: Memory Management
 - Many processes (with large VA spaces) sharing small physical memory
 - Where does each process' code, data, etc. get placed?
- ❖ Problem 3: Protection
 - How do we prevent one process from accessing another process' memory?
- ❖ Problem 4: Sharing
 - How do we allow two processes to share code or data?

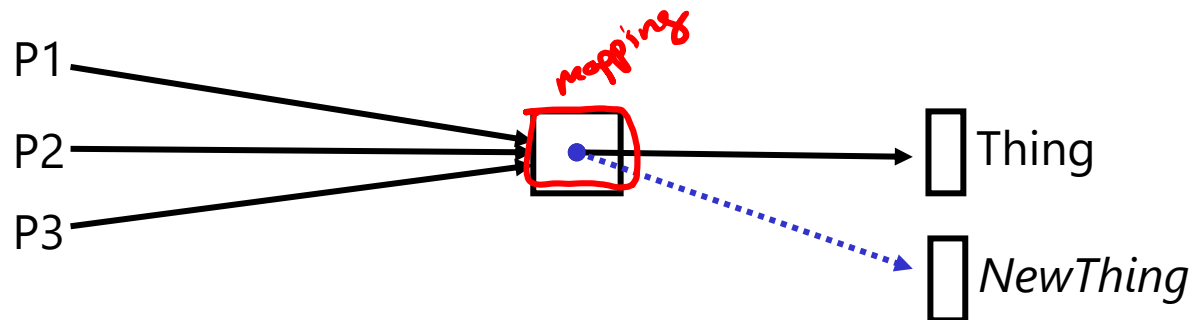
How can we solve these problems?

- ❖ “Any problem in computer science can be solved by adding another level of **indirection**.” – *David Wheeler, inventor of the subroutine*

- ❖ Without Indirection



- ❖ With Indirection

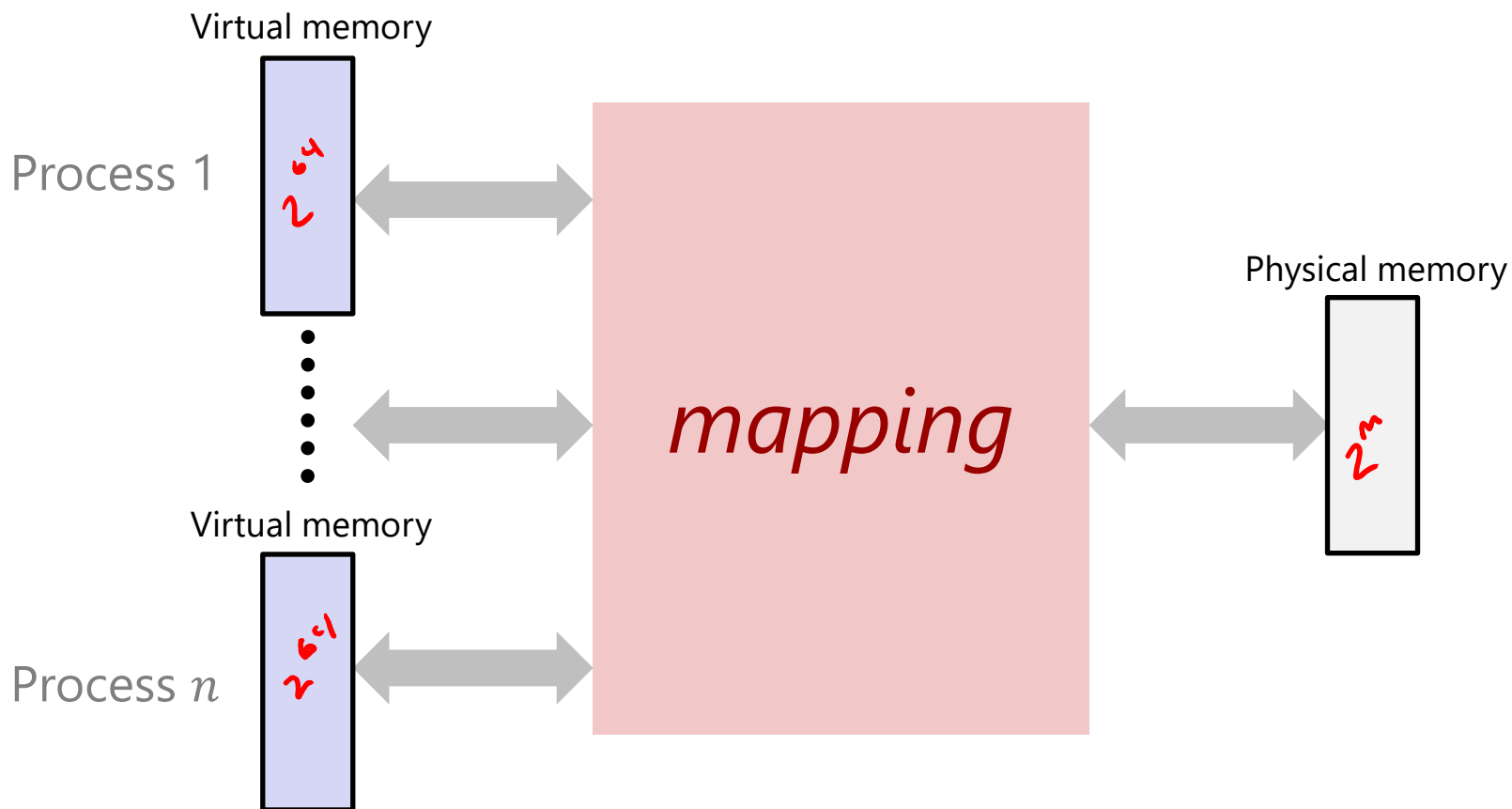


What if I want to move Thing?

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 have to 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

- ❖ **Virtual address space:** Set of $N = 2^n$ virtual addr 2^{64}
 $n = 64$
 - $\{0, 1, 2, 3, \dots, N-1\}$
- ❖ **Physical address space:** Set of $M = 2^m$ physical addr
 - $\{0, 1, 2, 3, \dots, M-1\}$
- ❖ Every byte in main memory has:
 - one physical address (PA)
 - zero, one, or more virtual addresses (VAs)
 - unused
 - used by 1 process
 - 2+ processes

Polling Questions

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

$$\text{per. process: } 2^{64} \text{ B} \quad * 2^3 = 2^{67} \text{ B}$$

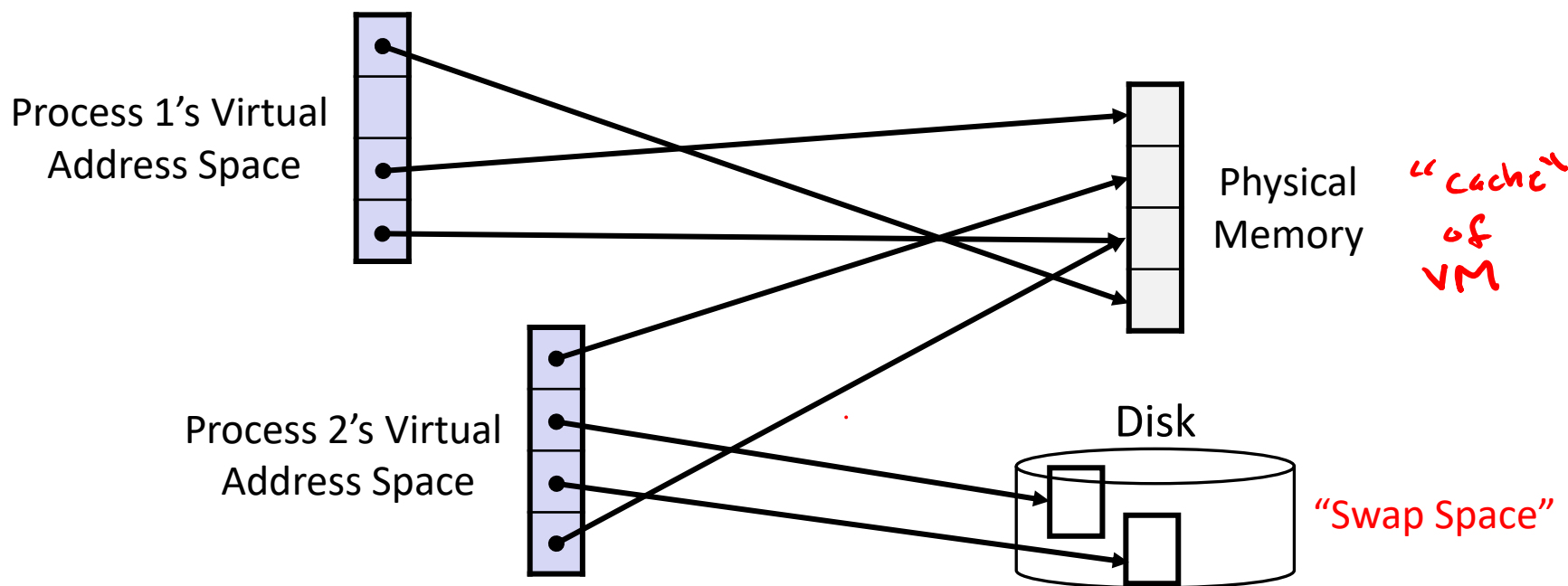
- ❖ True or False. A 32-bit machine with 8 GiB of RAM installed would never use all of it (in theory).

$$\text{VA space: } 2^{32} \text{ B per process} \rightarrow 4 \text{ GiB}$$

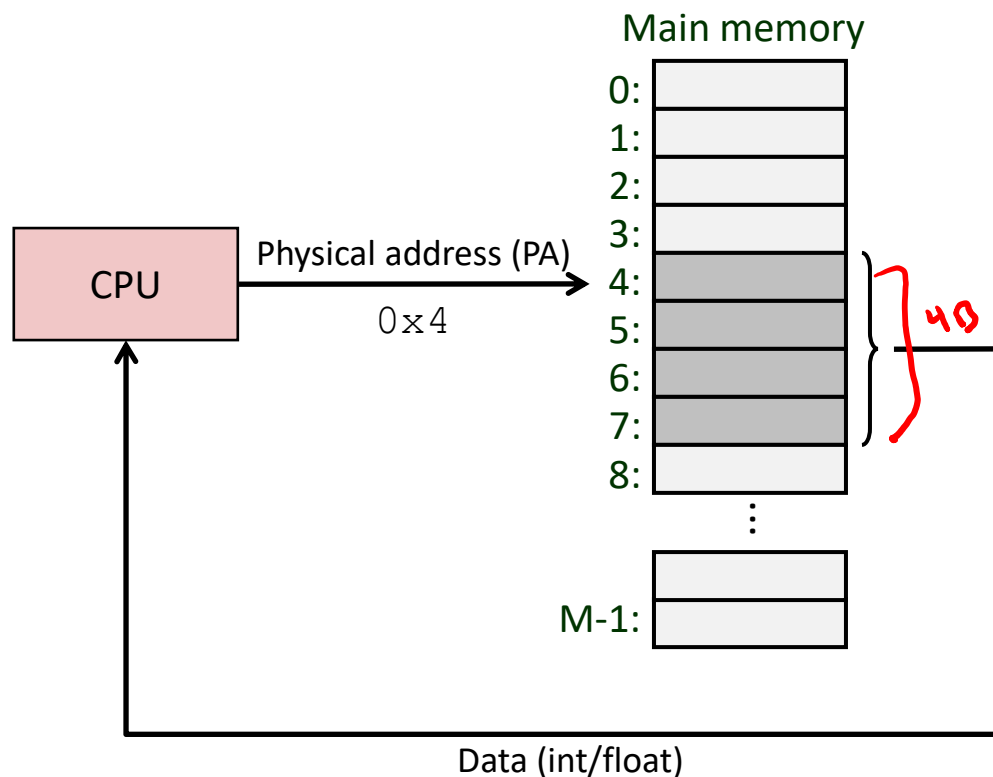
$$\geq 2 \text{ processes} \rightarrow \geq 8 \text{ GiB VM}$$

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

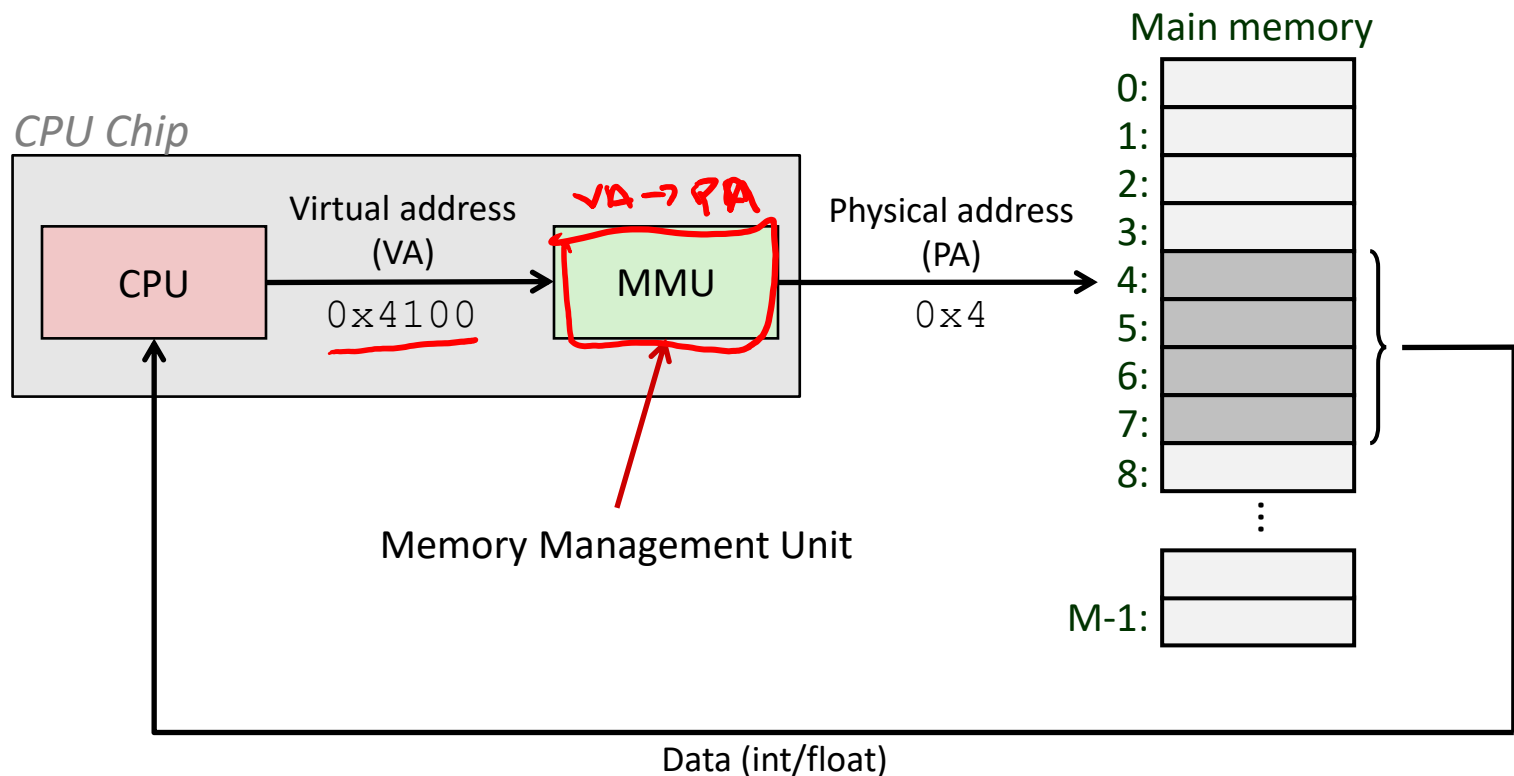


A System Using Physical Addressing



- ❖ 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...
 - One of the great ideas in computer science

Why Virtual Memory (VM)?

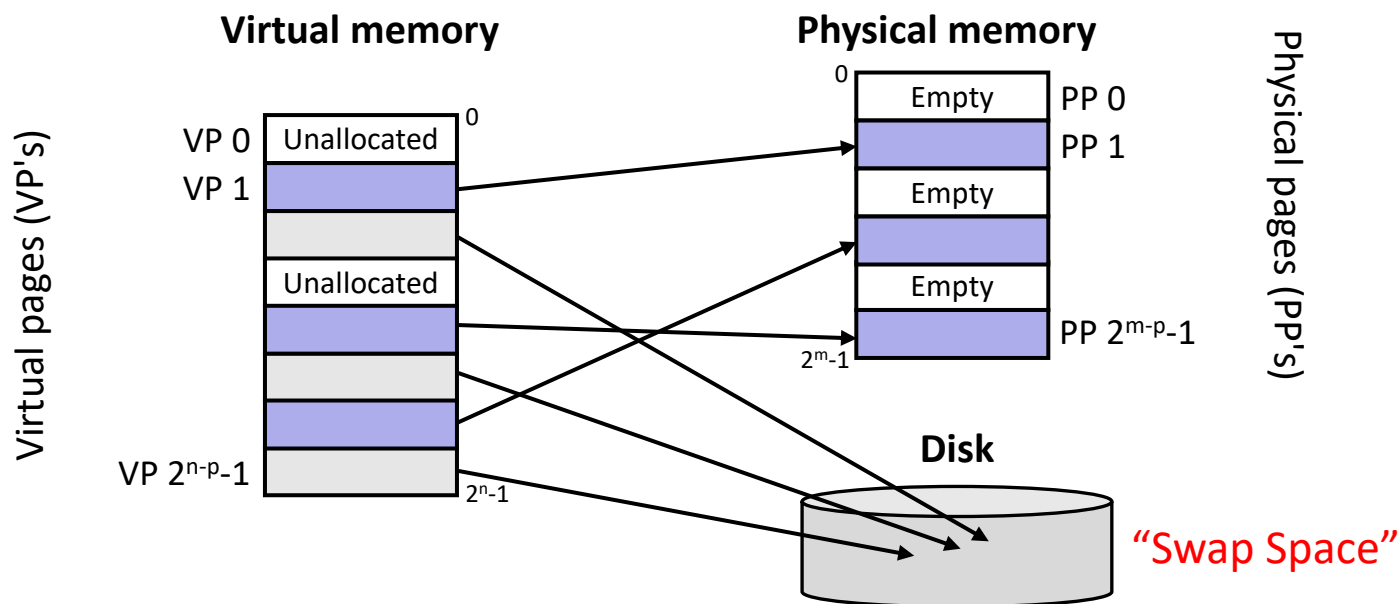
- ❖ Efficient use of limited main memory (RAM)
 - Use RAM as a cache for the parts of a virtual address space
 - Some non-cached parts stored on disk
 - Some (unallocated) non-cached parts stored nowhere
 - Keep only active areas of virtual address space in memory
 - Transfer data back and forth as needed
- ❖ **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

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, cache set*
 - page table entry — *cache line (data + request bits)*
 - access rights — *request bits*

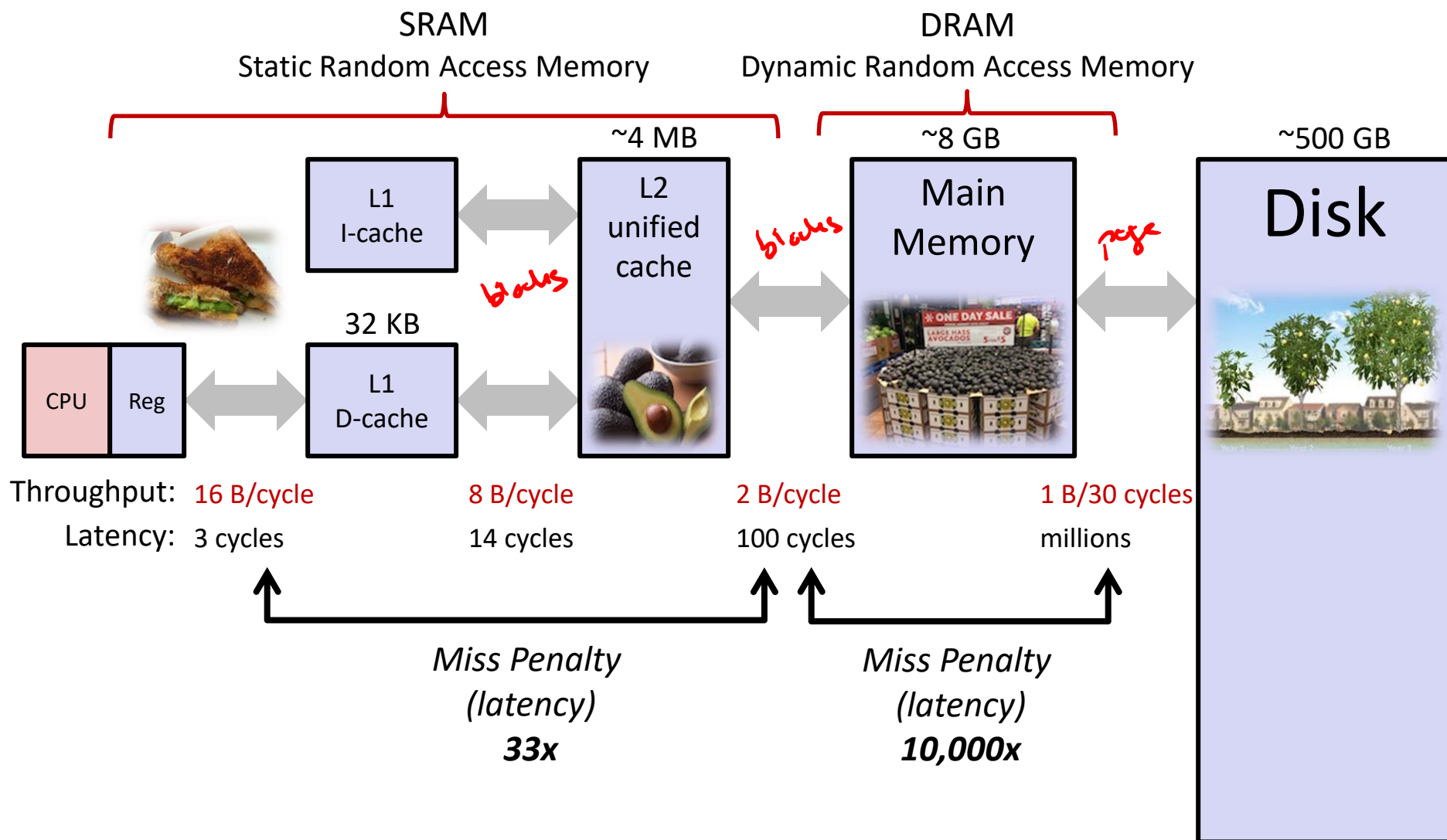
VM and the Memory Hierarchy

- ❖ 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)
 - Each virtual page can be stored in *any* physical page (no fragmentation!)
- ❖ 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
 - 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*
 - *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)

4 KiB

most common

Why does VM work on RAM/disk?

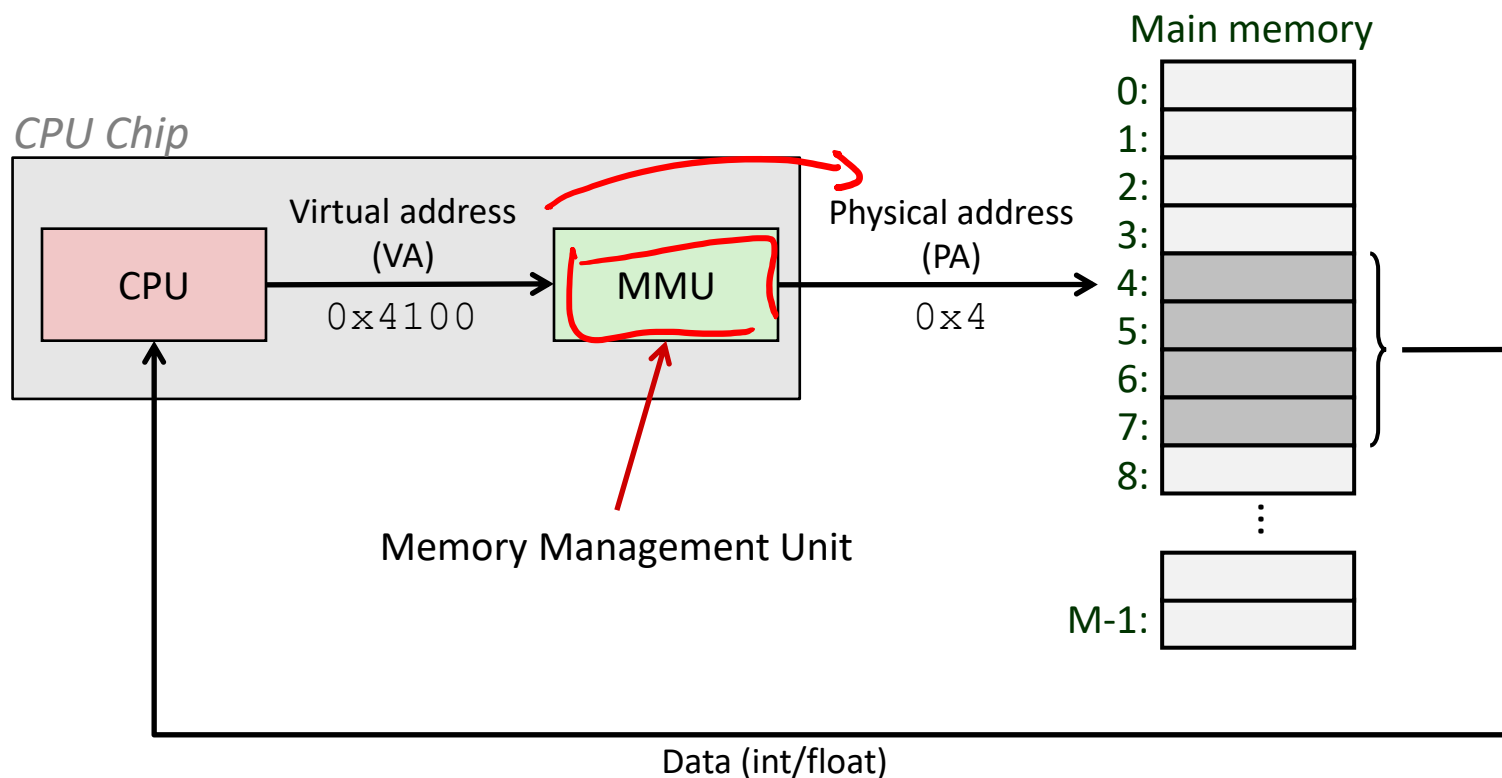
- ❖ Avoids disk accesses because of locality
 - Same reason that L1 / L2 / L3 caches work
- ❖ The set of virtual pages that a program is “actively” accessing at any point in time is called its *working set*
 - ✓ ■ If (*working set of one process* \leq *physical memory*):
 - Good performance for one process (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

Address Translation

*How do we perform the virtual
→ physical address translation?*



Address Translation: Page Tables

- ❖ CPU-generated address can be split into:

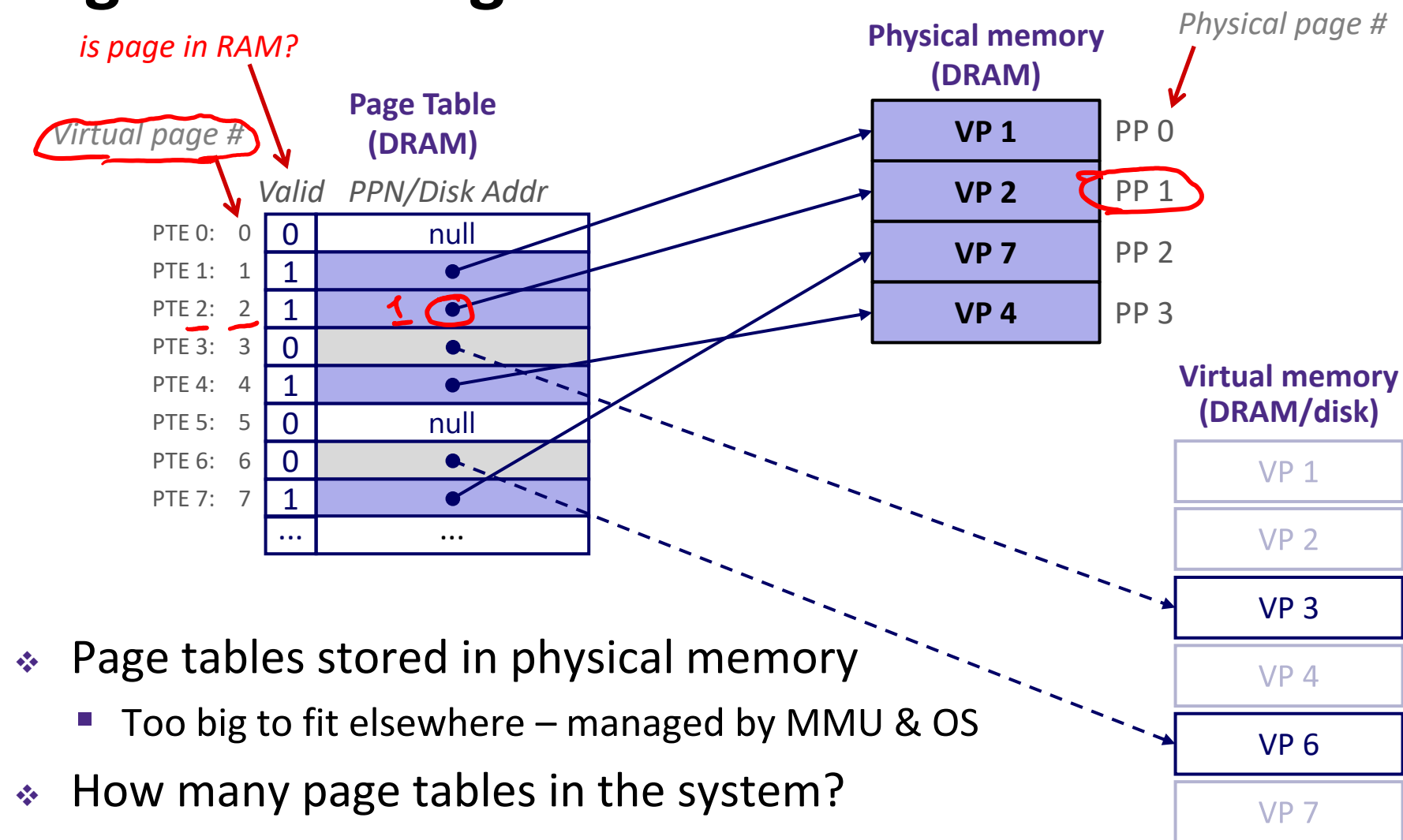
n -bit address:

VPN Virtual Page Number	$p = \log_2 P$ Page Offset
-------------------------------------	-------------------------------

 $\text{physical} \leftrightarrow \text{virtual}$

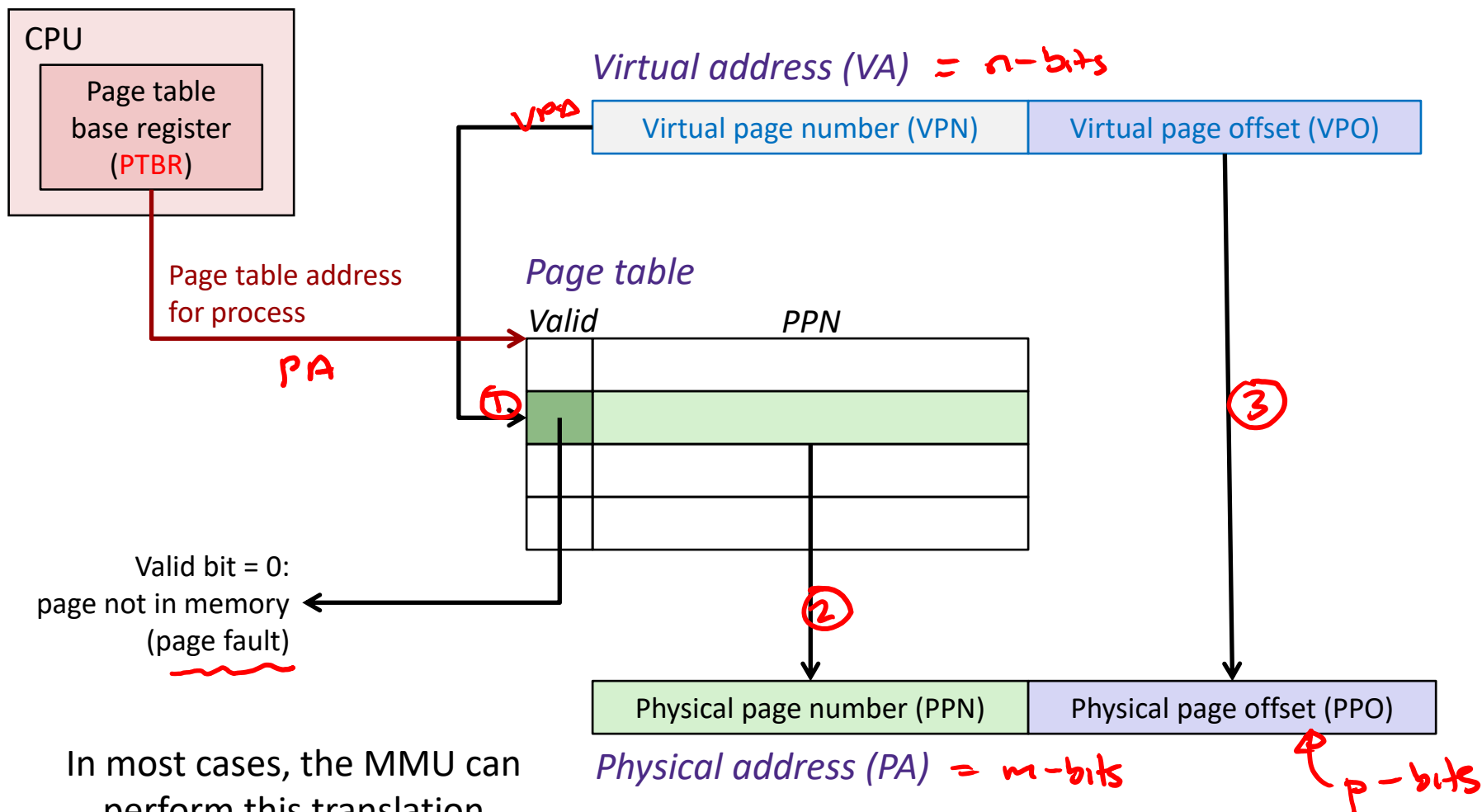
- Request is Virtual Address (**VA**), want Physical Address (**PA**)
- Note that Physical Offset = Virtual Offset (page-aligned)
- ❖ Use lookup table that we call the **page table** (**PT**) \rightarrow per process
 - Replace Virtual Page Number (**VPN**) for Physical Page Number (**PPN**) to generate 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



Polling Question

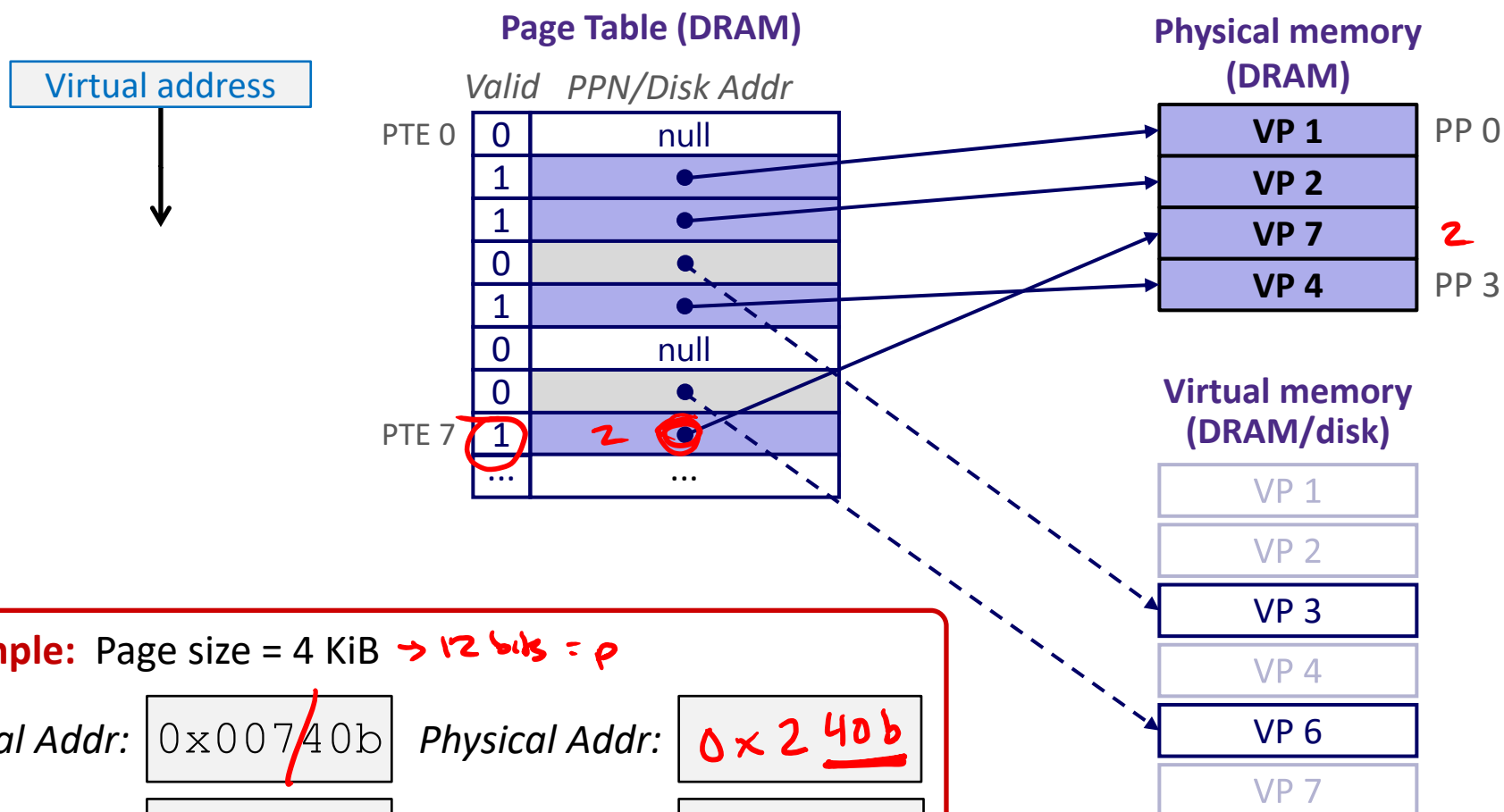
❖ How many bits wide are the following fields?

- 16 KiB pages $= 2^4 2^{10} \rightarrow 2^{14} \text{ B} \rightarrow p = 14\text{-bits}$
- 48-bit virtual addresses $n = 48\text{-bits} \rightarrow \text{VPN} = n - p = 48 - 14 = 34$
- 16 GiB physical memory $2^4 2^{30} \rightarrow 2^{34} \text{ B} \rightarrow m = 34$
 $\text{PPN} = m - p = 34 - 14 = 20$
- Vote in Ed Lessons

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

Page Hit

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



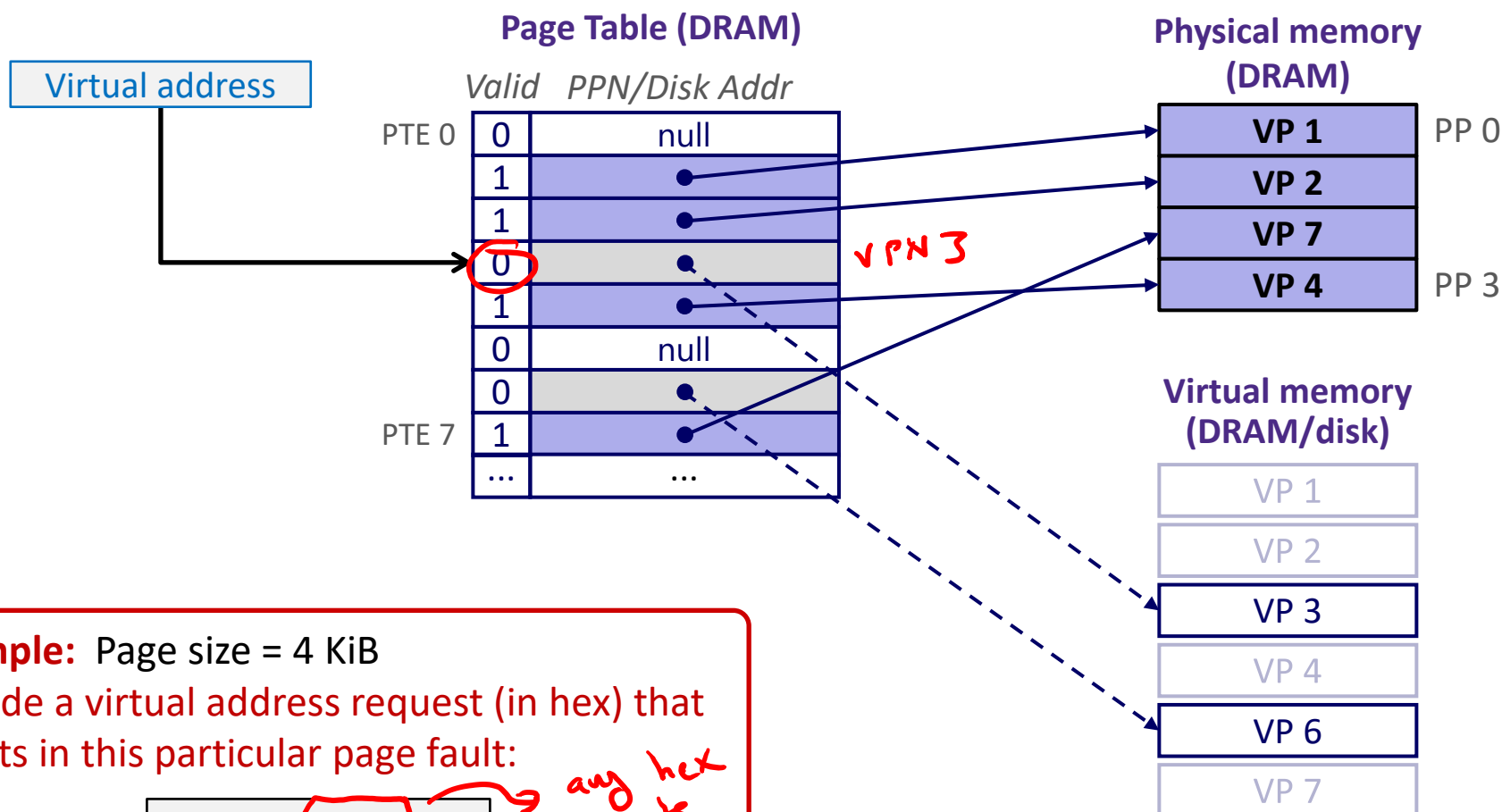
Example: Page size = 4 KiB $\rightarrow 12 \text{ bits} = p$

Virtual Addr: $0x00740b$ Physical Addr: $0x240b$

VPN: 7 PPN: 2

Page Fault

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

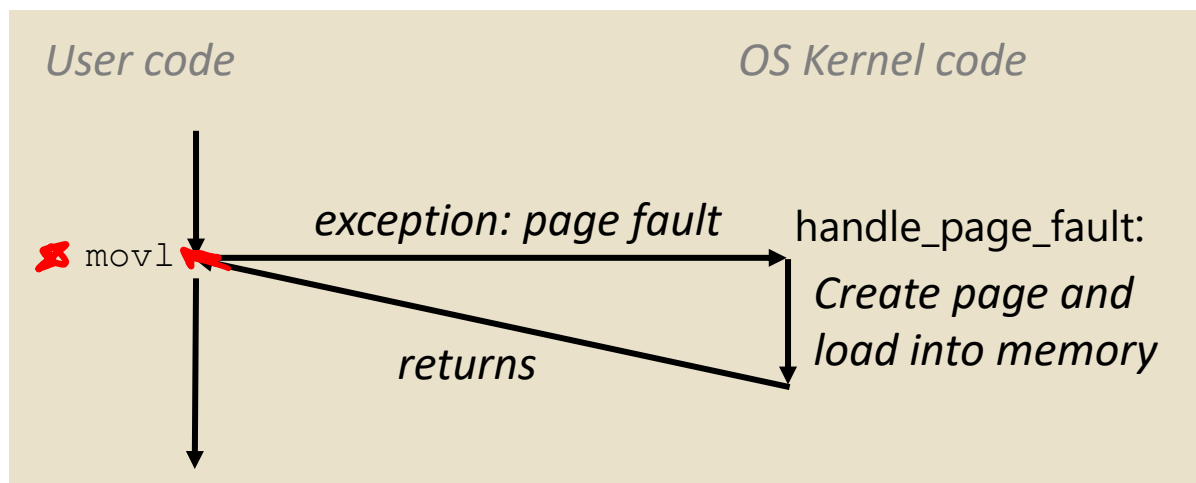


Reminder: Page Fault Exception

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

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

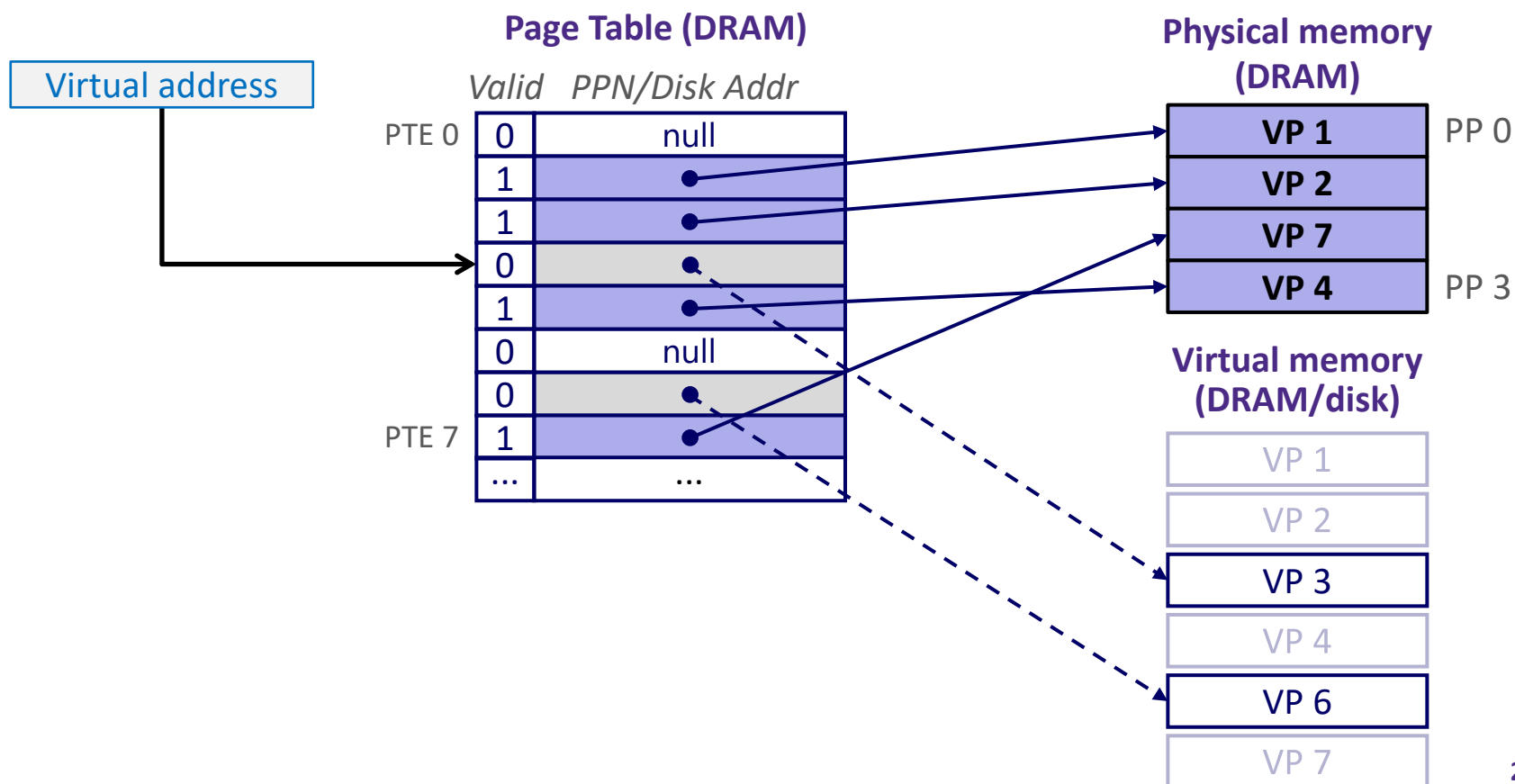
```
80483b7:      c7 05 10 9d 04 08 0d  movl    $0xd,0x8049d10
```



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

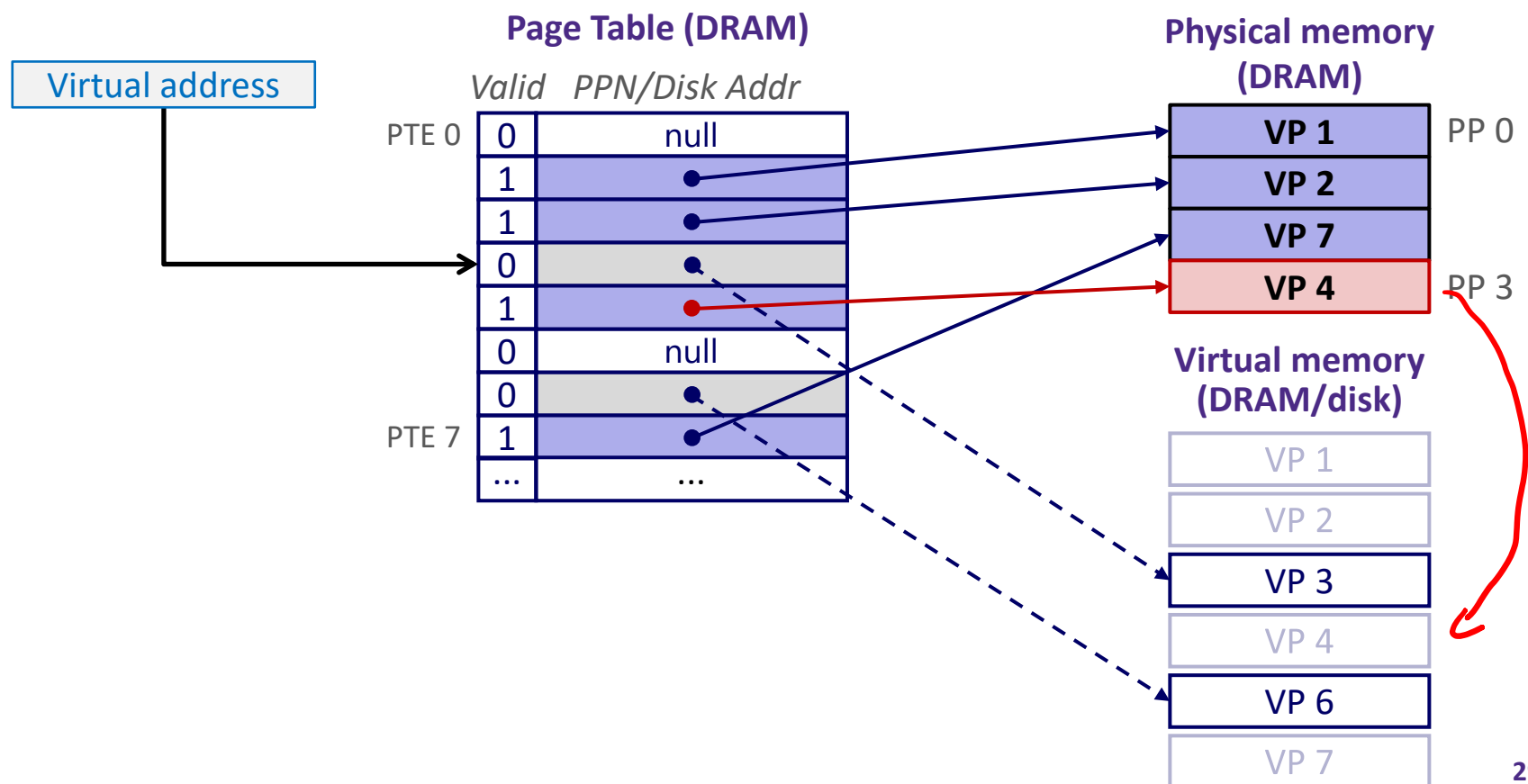
Handling a Page Fault

- ❖ Page miss causes page fault (an exception)



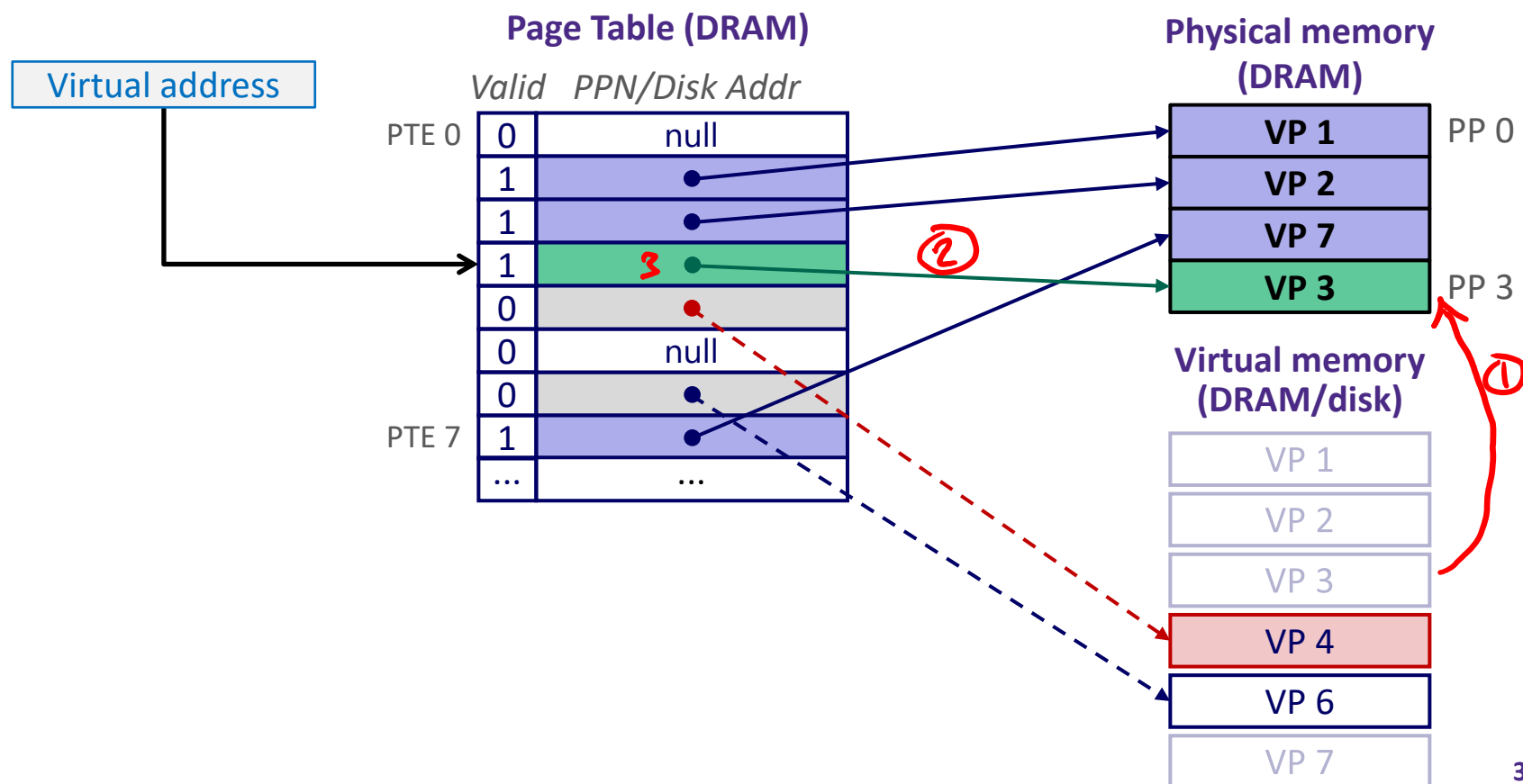
Handling a Page Fault

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



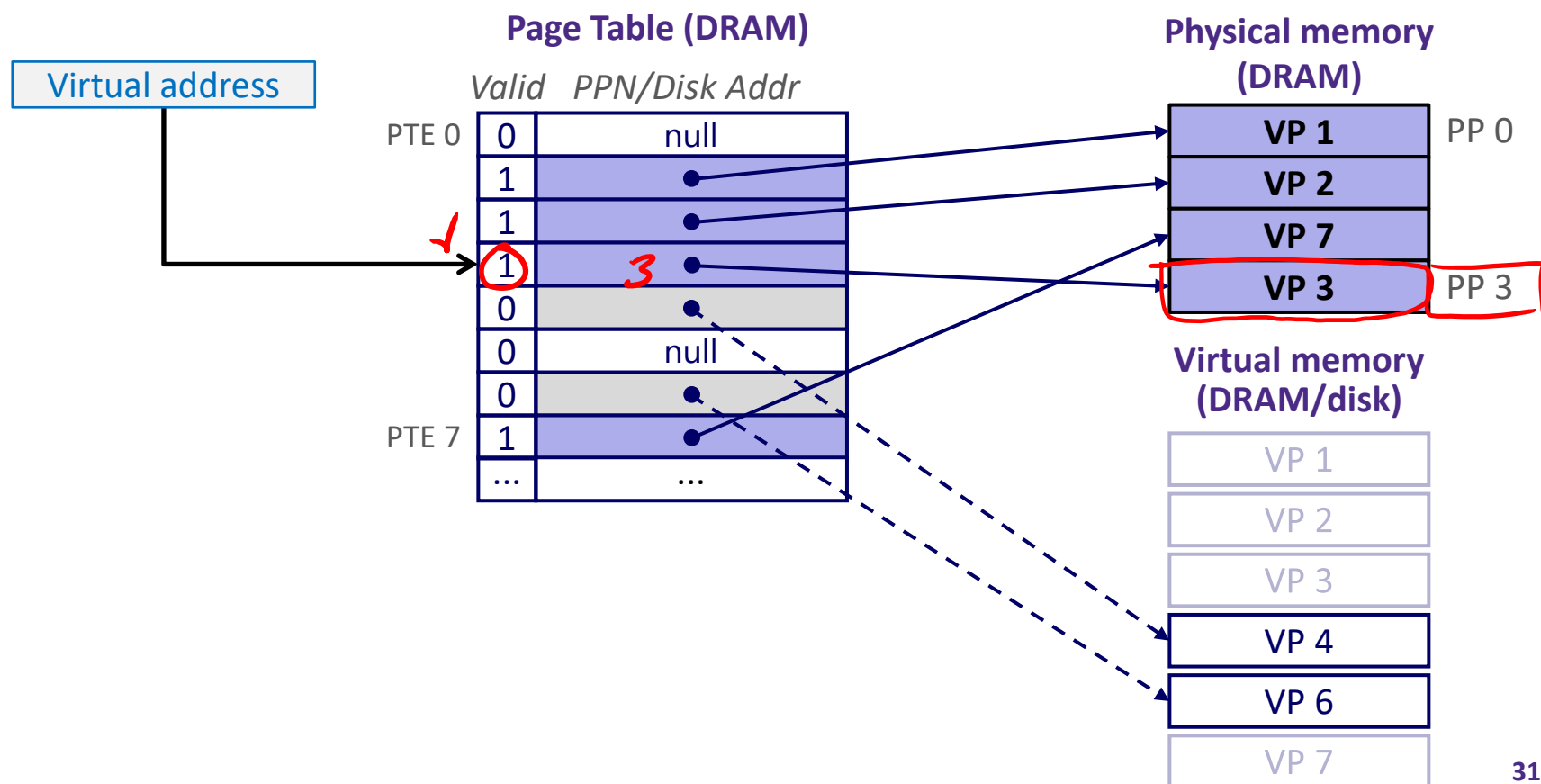
Handling a Page Fault

- ❖ 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!

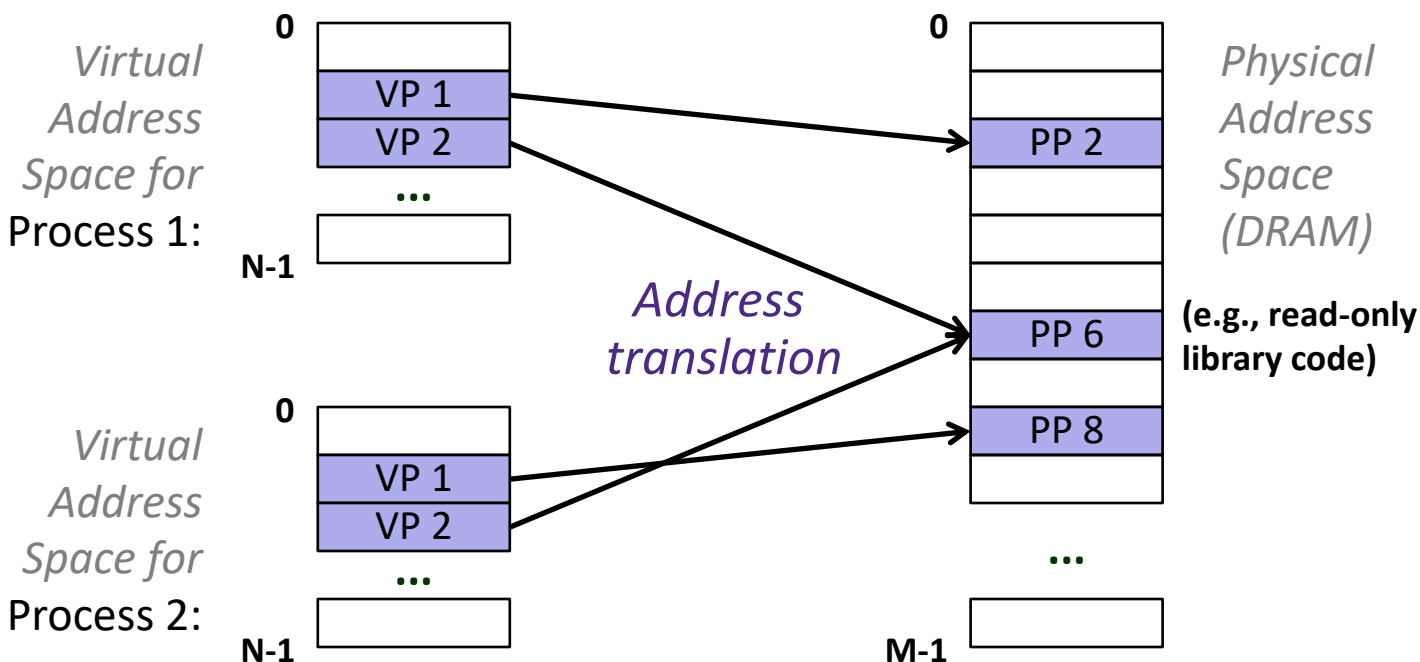


Virtual Memory (VM)

- ❖ 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**

VM for Managing Multiple Processes

- ❖ Key abstraction: each process has its own virtual address space
 - It can view memory as *a simple linear array*
- ❖ With virtual memory, this simple linear virtual address space **need not be contiguous in physical memory**
 - Process needs to store data in another VP? Just map it to *any* PP!



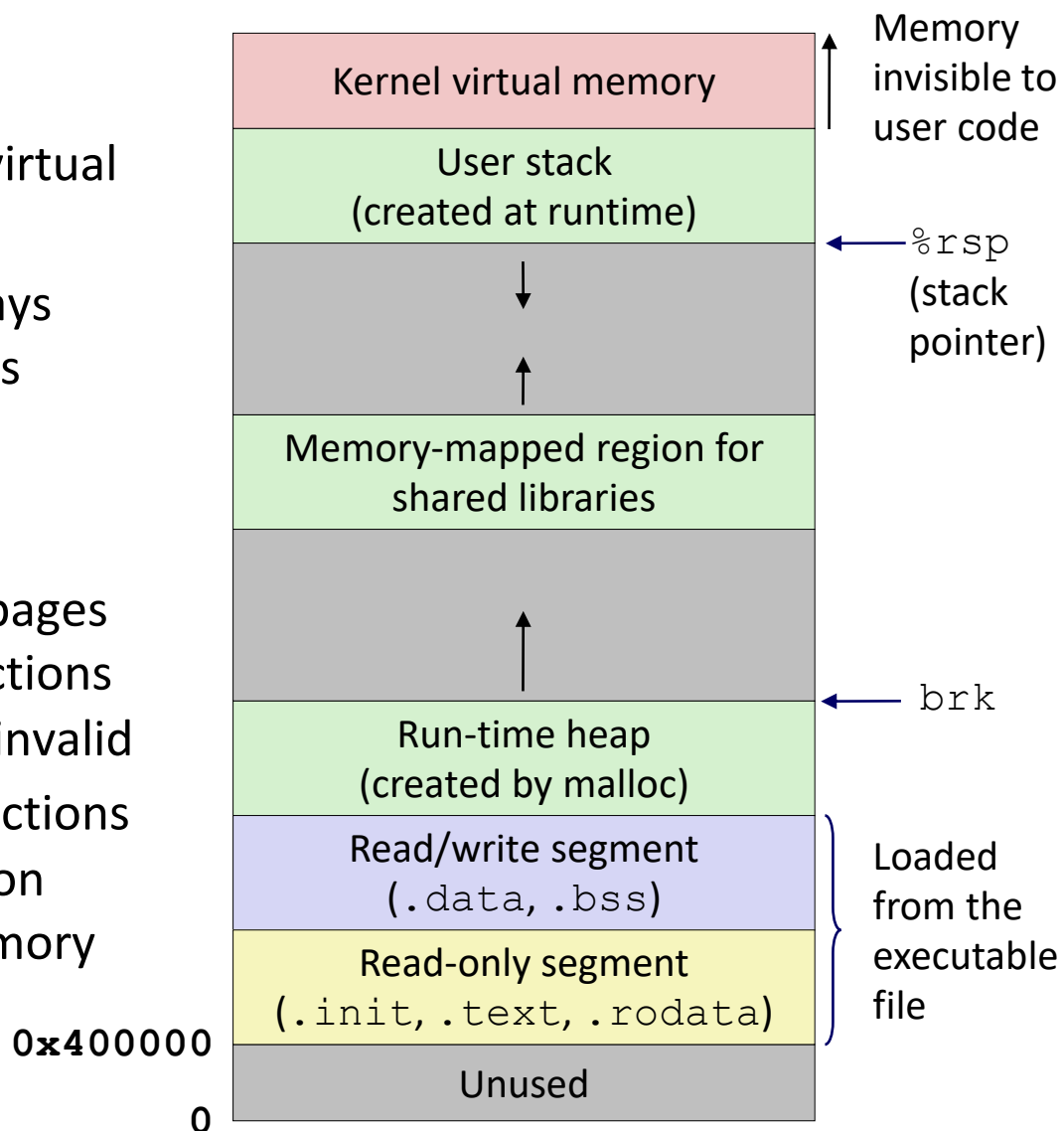
Simplifying Linking and Loading

❖ Linking

- Each program has similar virtual address space
- Code, Data, and Heap always start at the same addresses

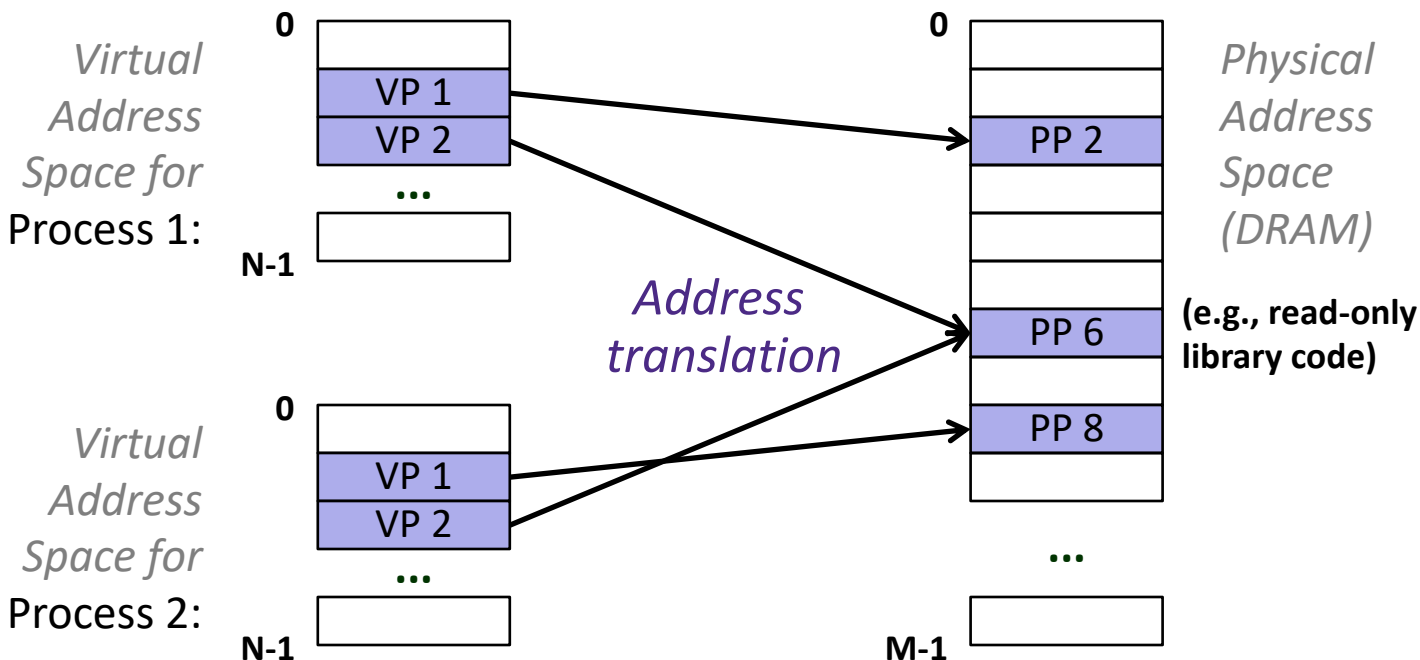
❖ Loading

- `execve` allocates virtual pages for `.text` and `.data` sections & creates PTEs marked as invalid
- The `.text` and `.data` sections are copied, page by page, on demand by the virtual memory system



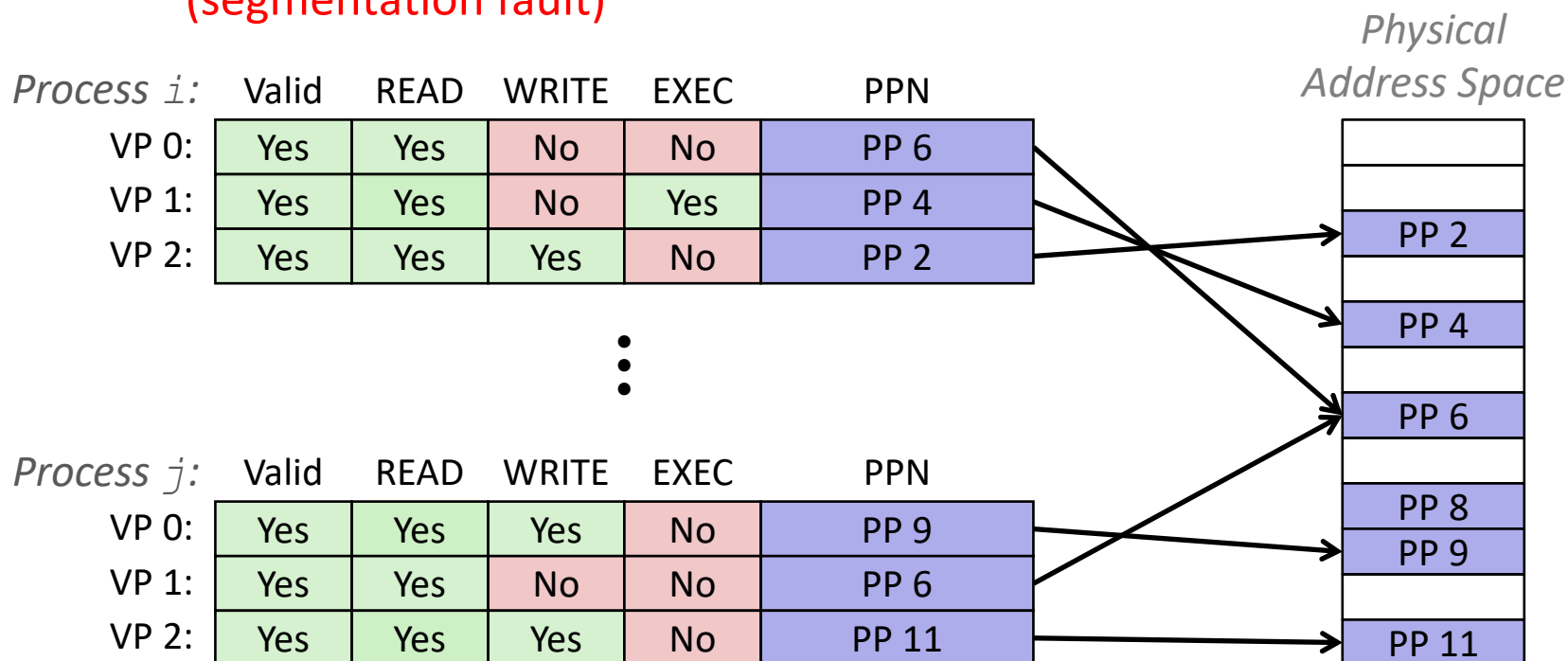
VM for Protection and Sharing

- ❖ The mapping of VPs to PPs provides a simple mechanism to *protect* memory and to *share* memory between processes
 - **Sharing:** map virtual pages in separate address spaces to the same physical page (here: PP 6)
 - **Protection:** process can't access physical pages to which none of its virtual pages are mapped (here: Process 2 can't access PP 2)



Memory Protection Within Process

- ❖ VM implements read/write/execute permissions
 - Extend page table entries with permission bits
 - MMU checks these permission bits on every memory access
 - If violated, raises exception and OS sends SIGSEGV signal to process (segmentation fault)



Memory Review Question

- ❖ What should the permission bits be for pages from the following sections of virtual memory?

Section	Read	Write	Execute
Stack			
Heap			
Static Data			
Literals			
Instructions			