Virtual Memory II

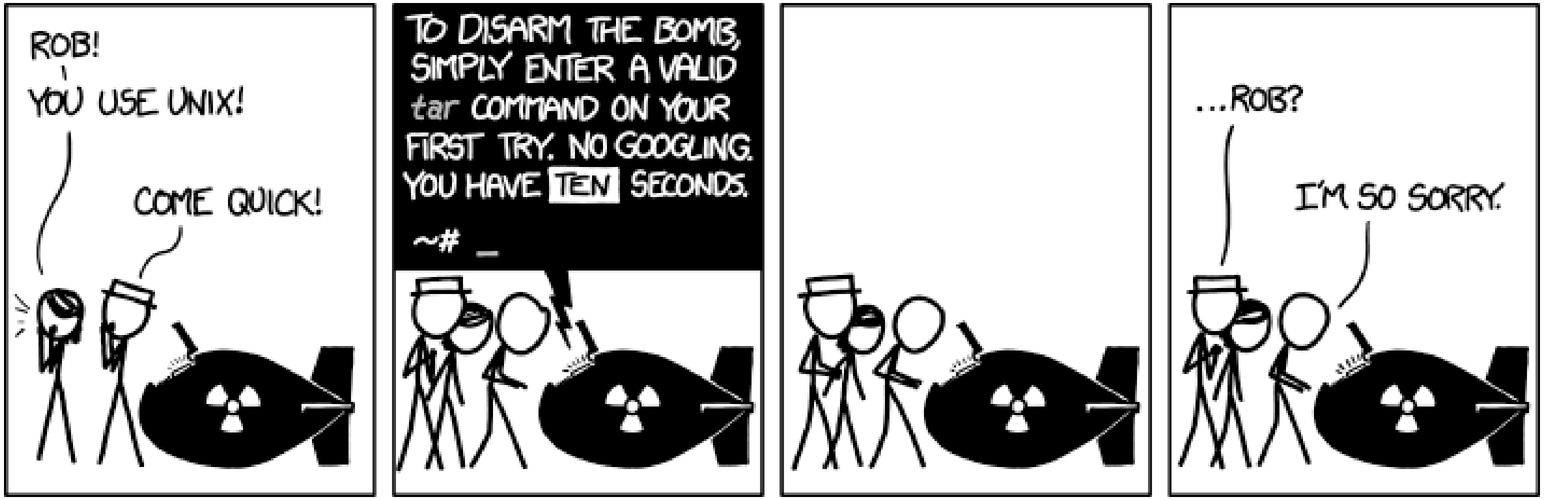
CSE 351 Autumn 2021

Instructor:

Justin Hsia

Teaching Assistants:

Allie Pfleger, Anirudh Kumar, Assaf Vayner, Atharva Deodhar, Celeste Zeng, Dominick Ta, Francesca Wang, Hamsa Shankar, Isabella Nguyen, Joy Dang, Julia Wang, Maggie Jiang, Monty Nitschke, Morel Fotsing, Sanjana Chintalapati



http://xkcd.com/1831/

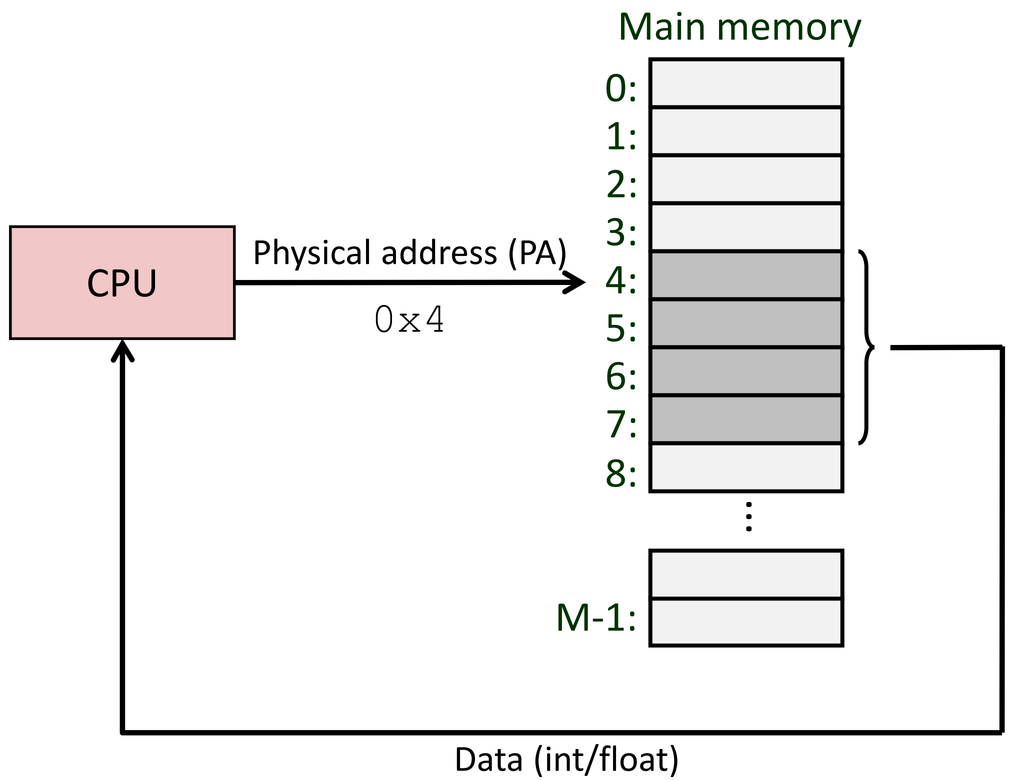
Relevant Course Information

- hw22 due *Friday* (11/26)
- hw23 due next Wednesday (12/1)
  - Another double-lecture hw
- Lab 4 due Monday (11/29)

- Virtual section this week on virtual memory (videos)

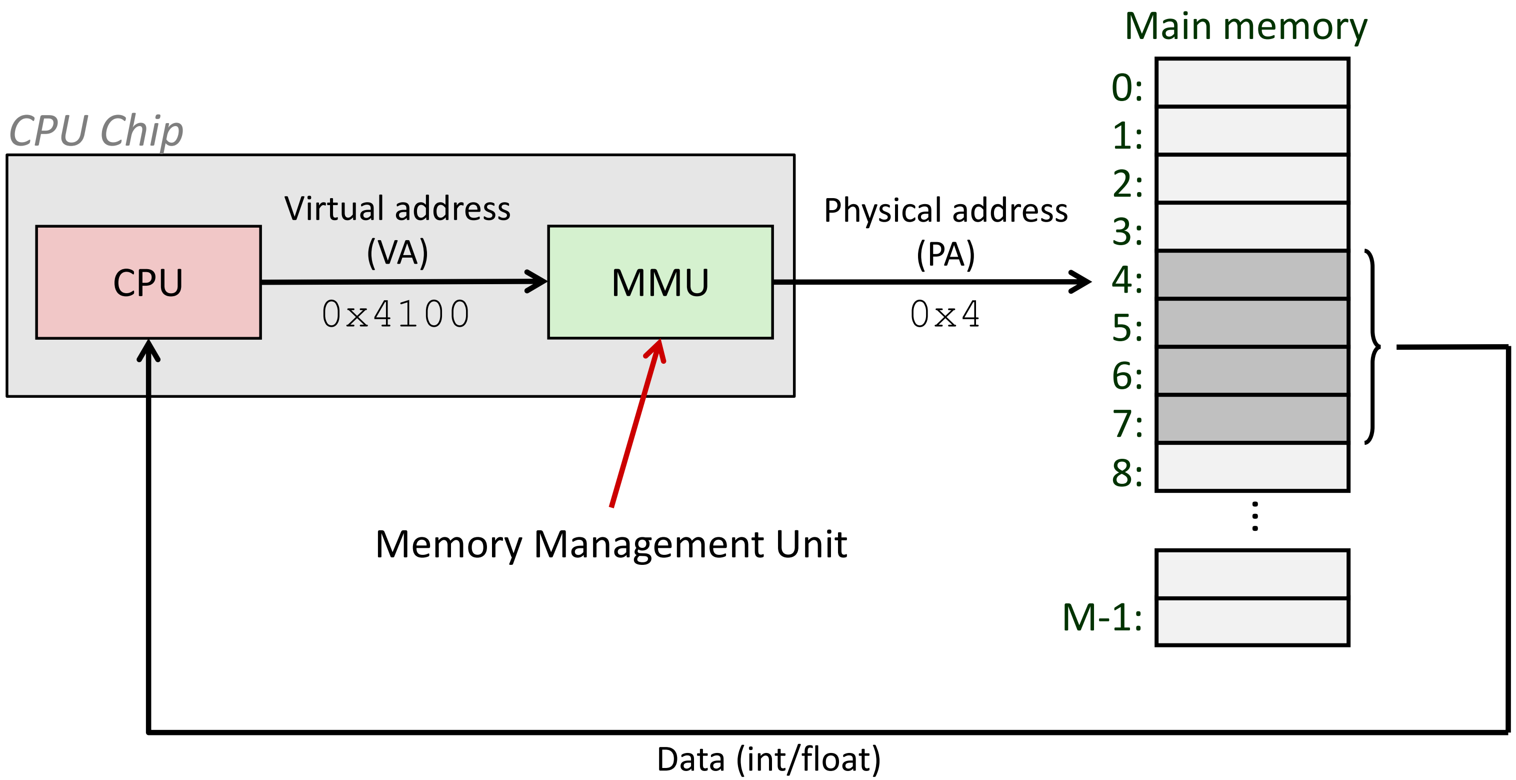
- Looking ahead
  - Final Dec. 13-15, regrade requests Dec. 18-19
  - Check your grades in Canvas as we go

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

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)

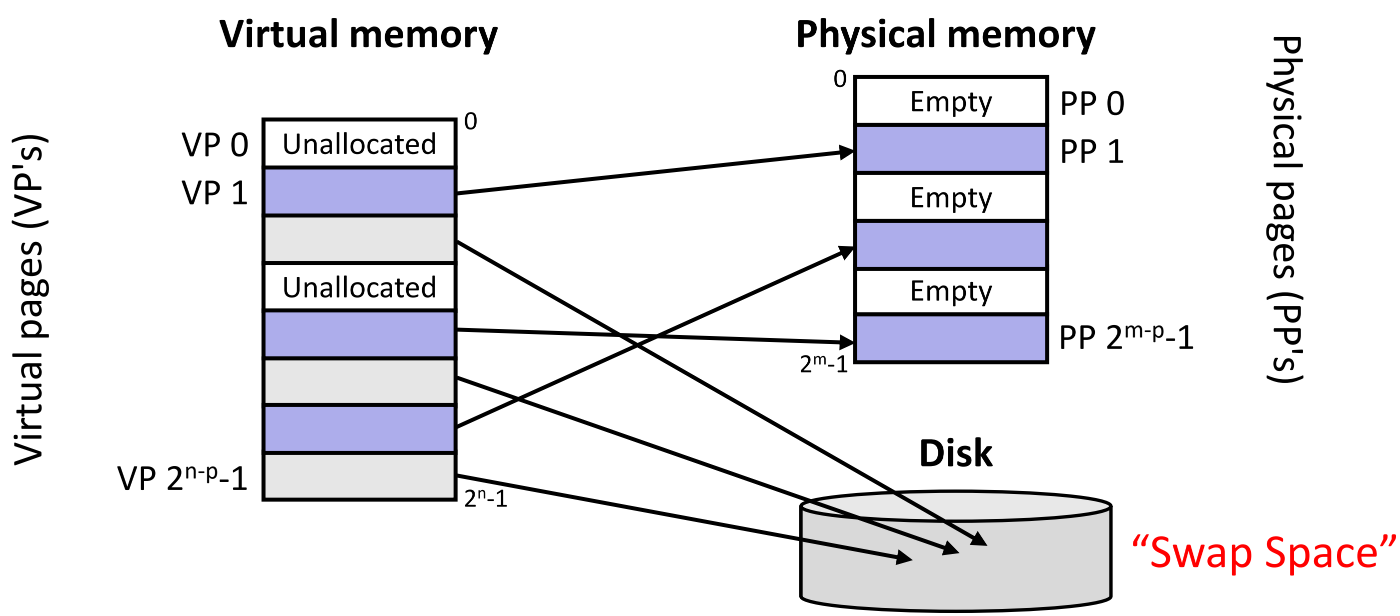
- Questions from the Reading?

Review Questions

- Which terms from caching are most similar/analogous to the new virtual memory terms?
  - page size
  - page offset width
  - virtual page number
  - physical page number
  - page table entry
  - access rights

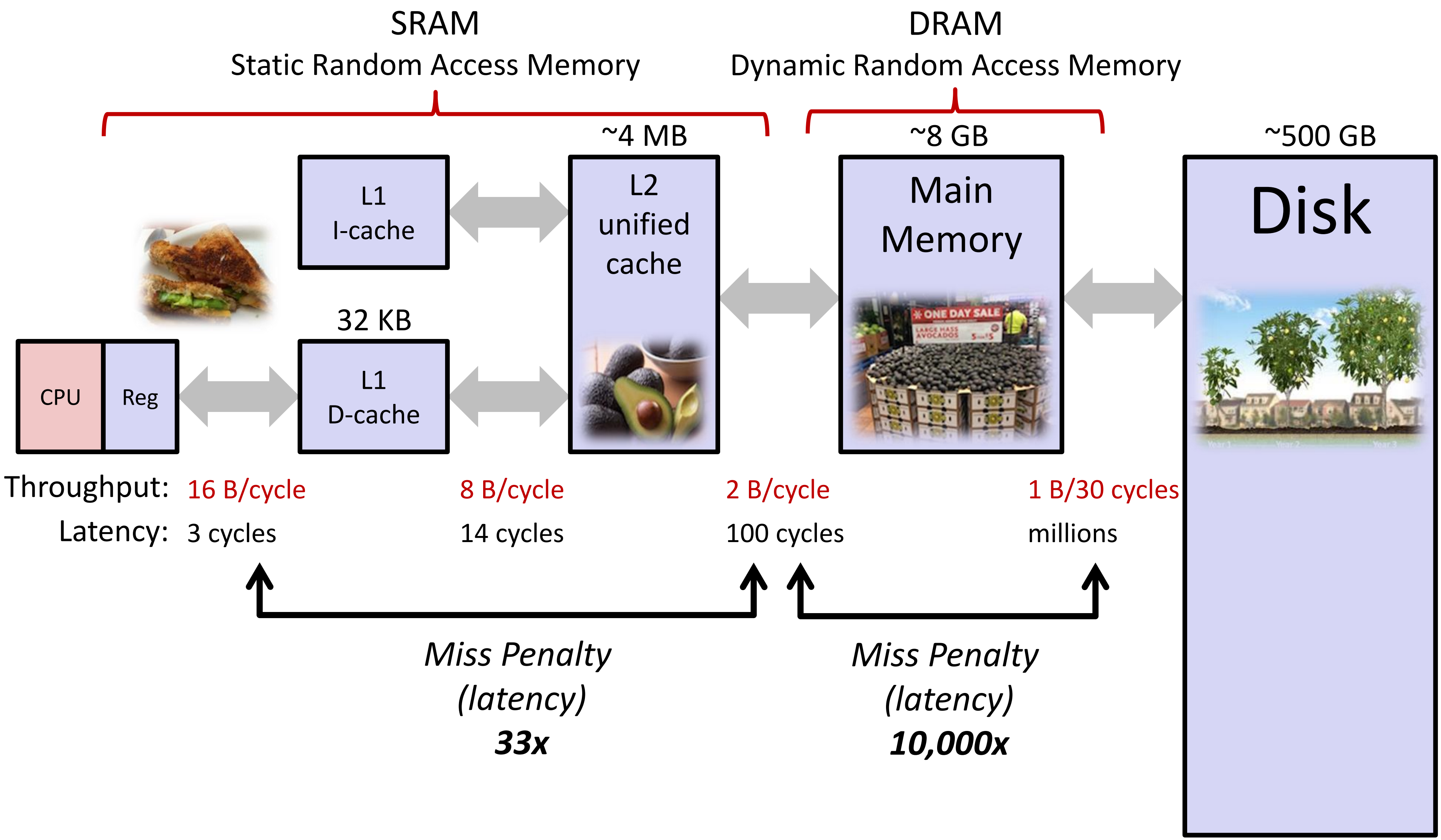
VM and the Memory Hierarchy

- Think of memory (virtual or physical) as an array of bytes, now split into *pages*
  - Pages are another unit of aligned memory (size is $\mathrm{P} = 2^p$ 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)

Why does VM work on RAM/disk?

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

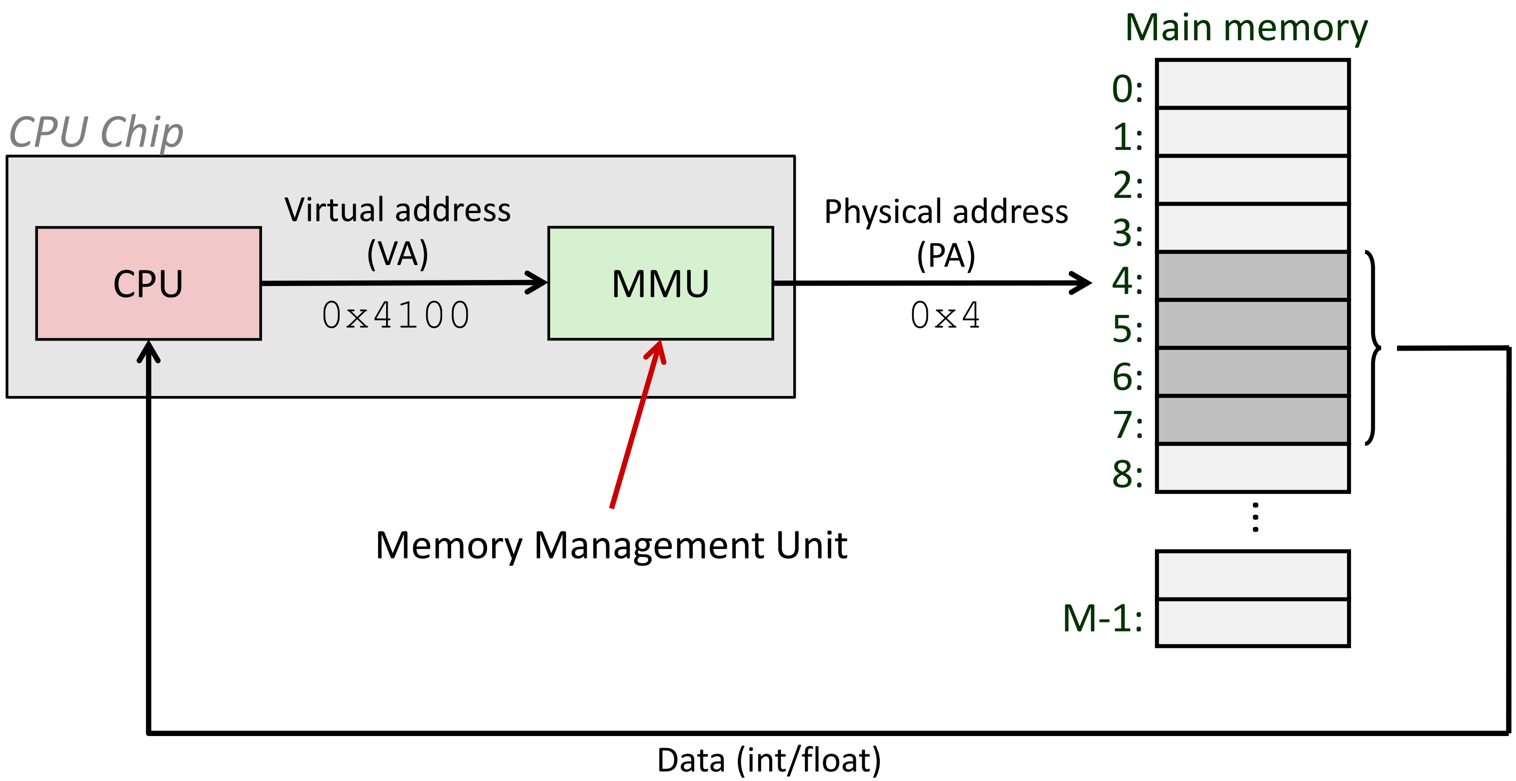
- The set of virtual pages that a program is "actively" accessing at any point in time is called its *working set*
  - If (*working set of one process* ≤ *physical memory*):
    - Good performance for one process (after compulsory misses)
  - If (*working sets of all processes* > *physical memory*):
    - ***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

Address Translation

How do we perform the virtual → physical address translation?



Address Translation: Page Tables

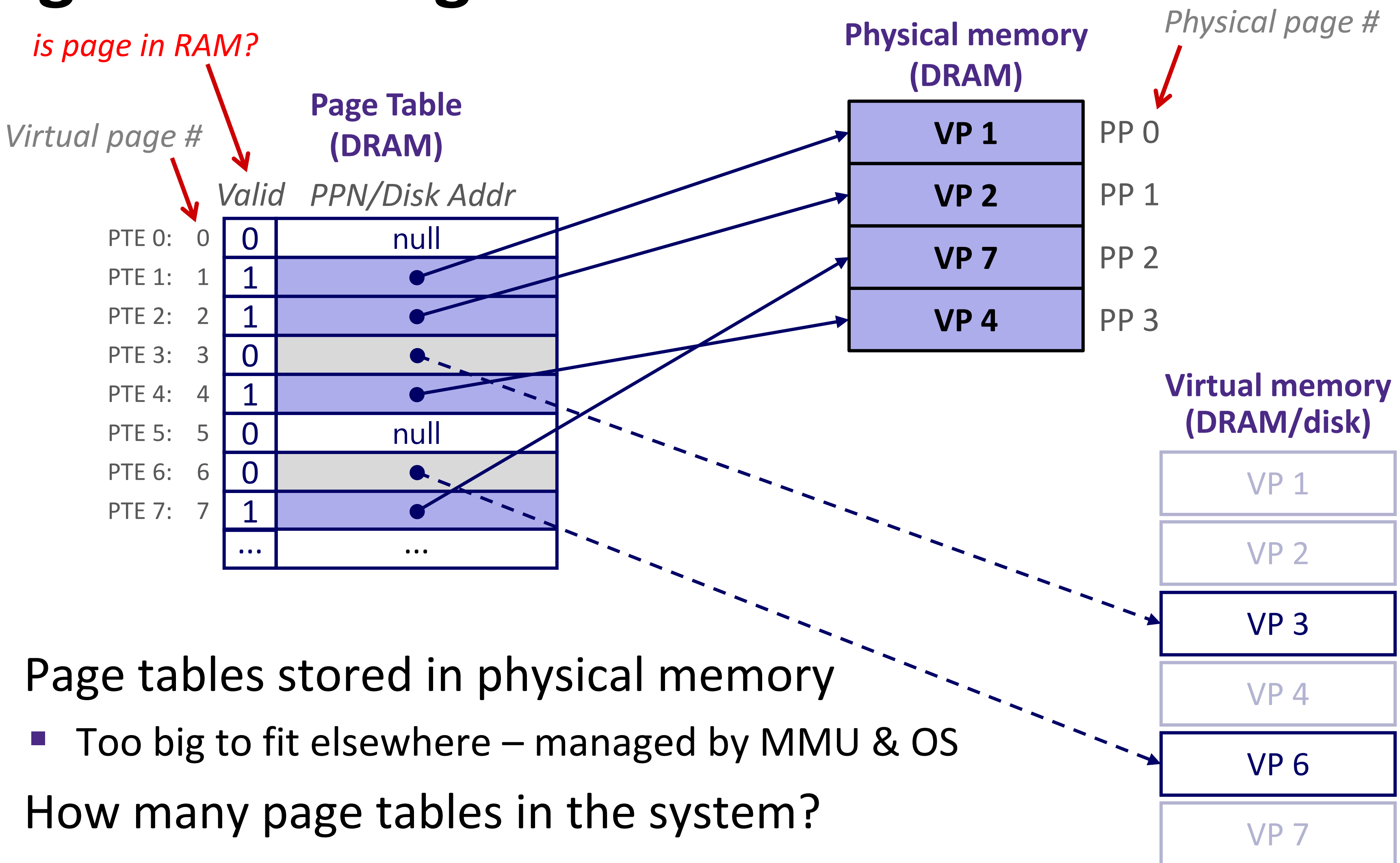
- CPU-generated address can be split into:

n-bit address:

Virtual Page Number	Page Offset

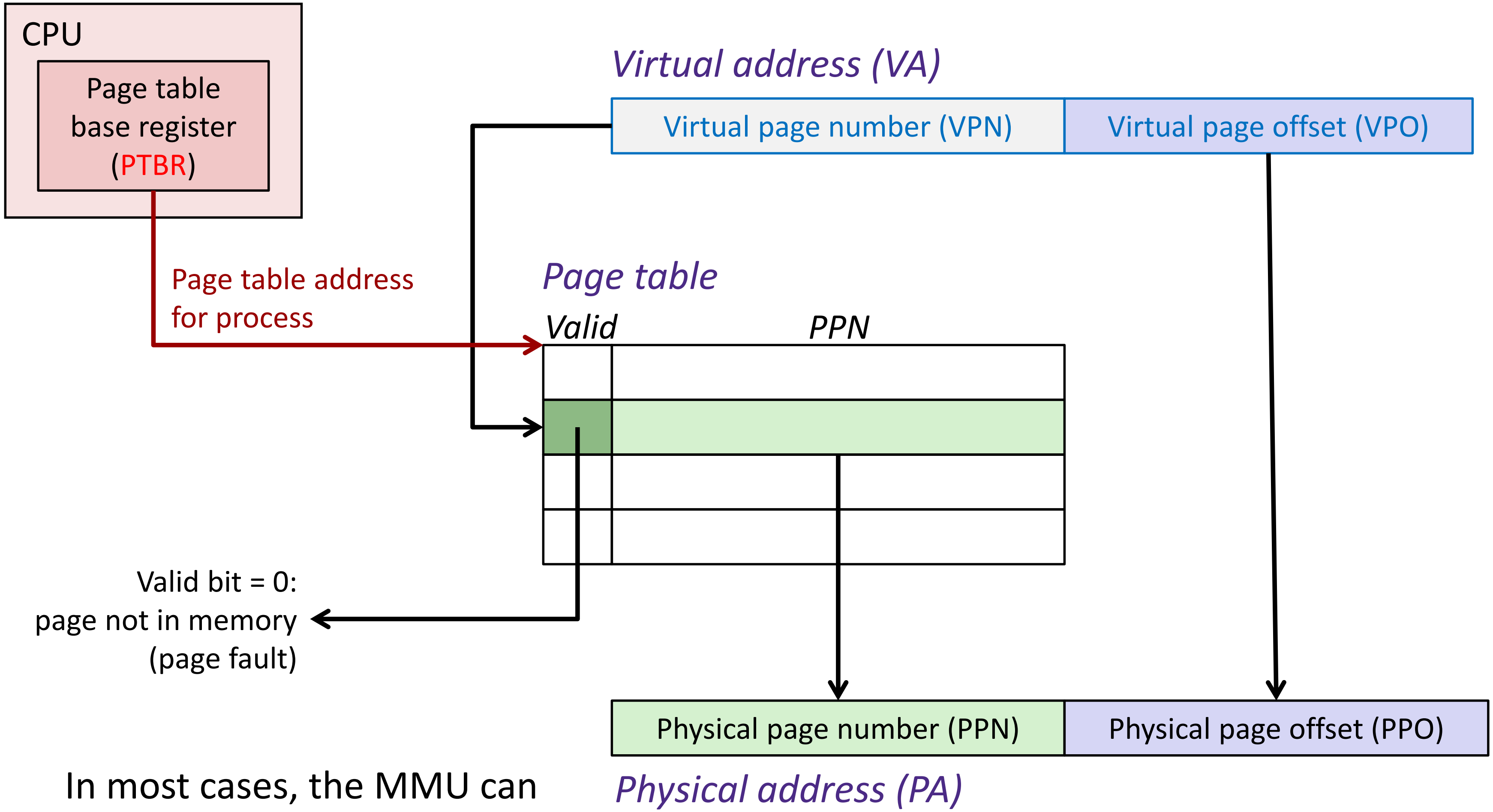
  - Request is Virtual Address (VA), want Physical Address (PA)
  - Note that Physical Offset = Virtual Offset (page-aligned)
- Use lookup table that we call the *page table* (PT)
  - Replace Virtual Page Number (VPN) for Physical Page Number (PPN) to generate Physical Address
  - Index PT using VPN: page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



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

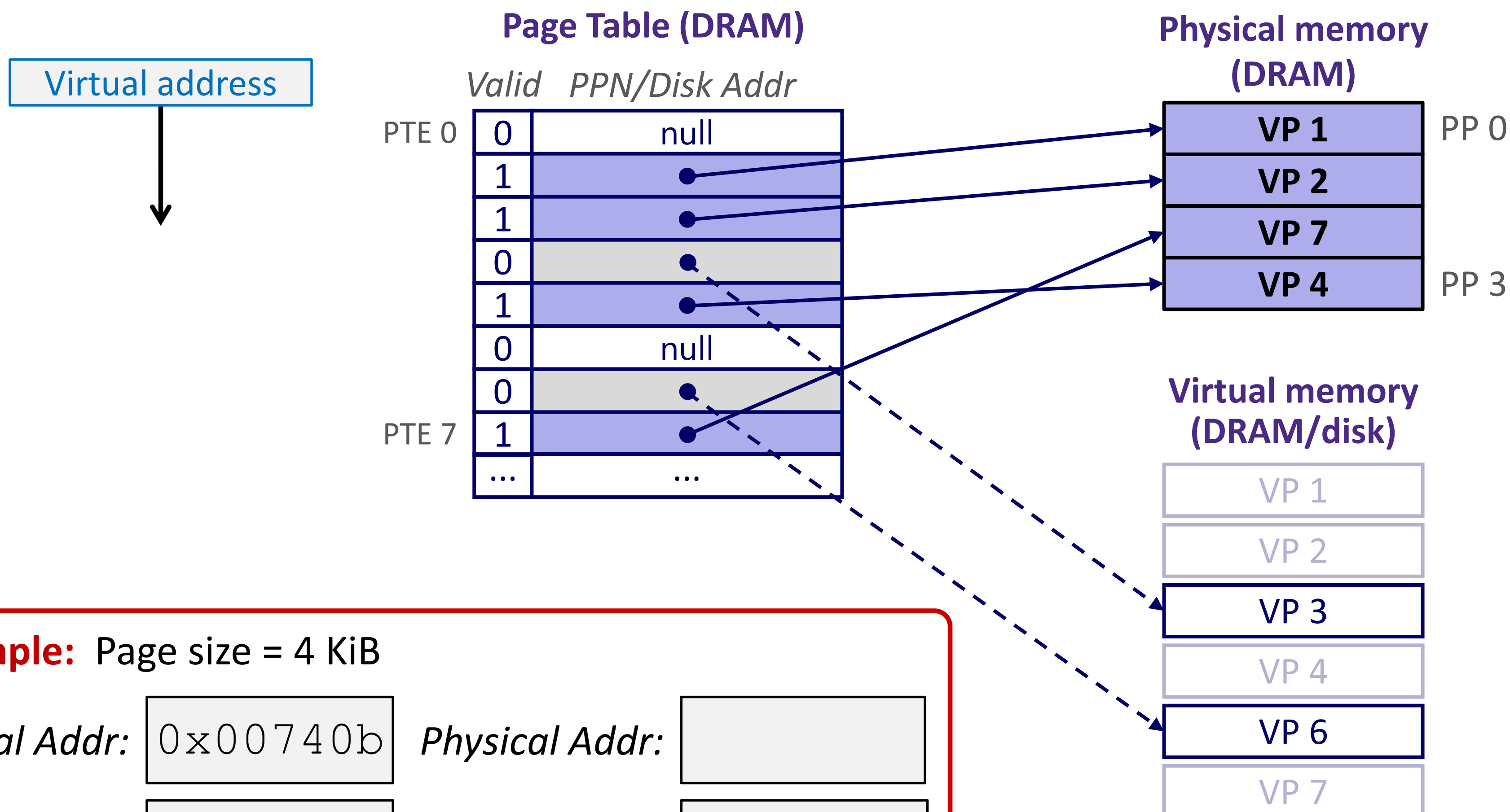
Polling Question

- How many bits wide are the following fields?
  - 16 KiB pages
  - 48-bit virtual addresses
  - 16 GiB physical memory
  - 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



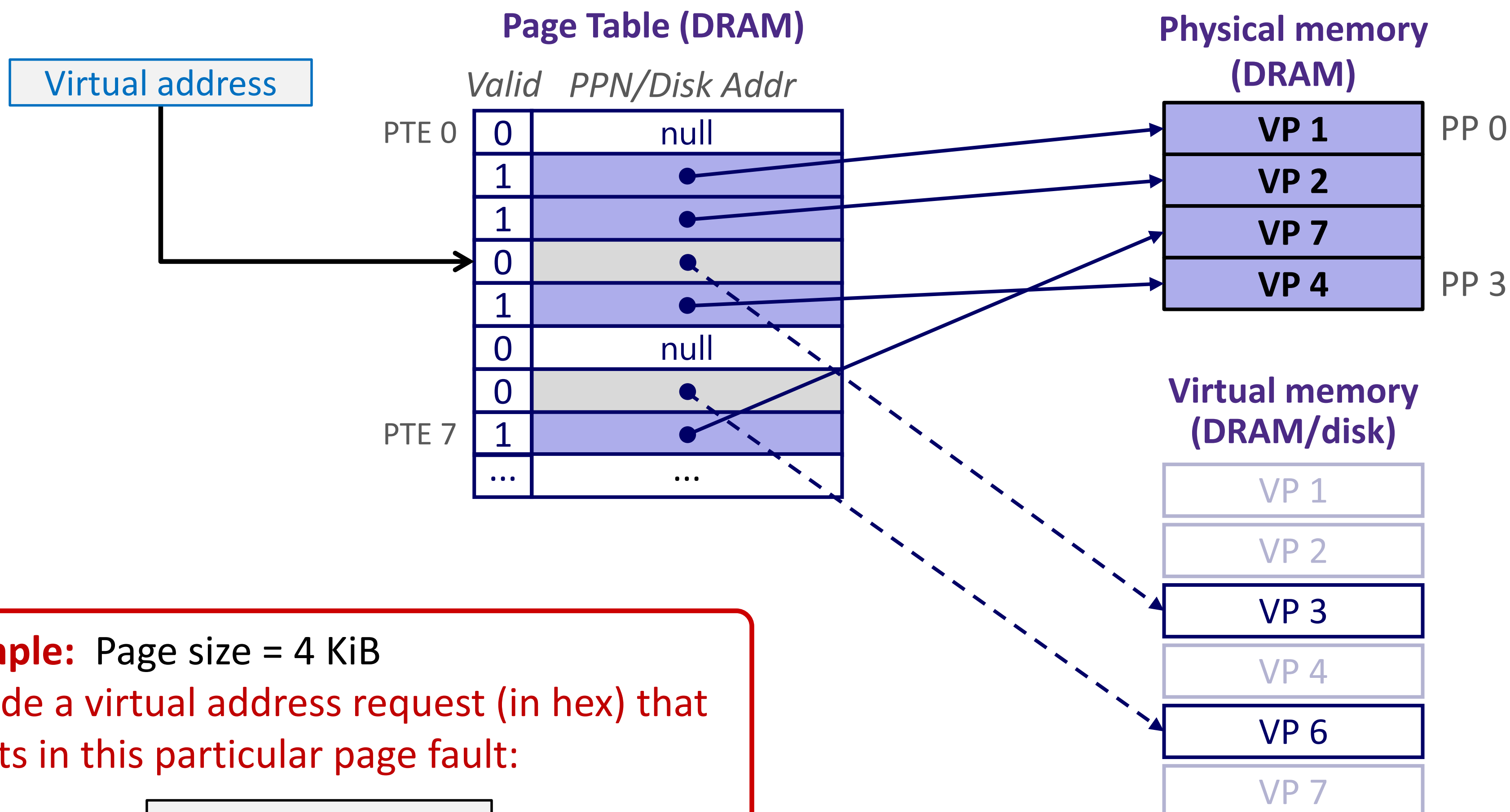
Example: Page size = 4 KiB

Virtual Addr: `0x00740b` *Physical Addr:*

VPN: *PPN:*

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:

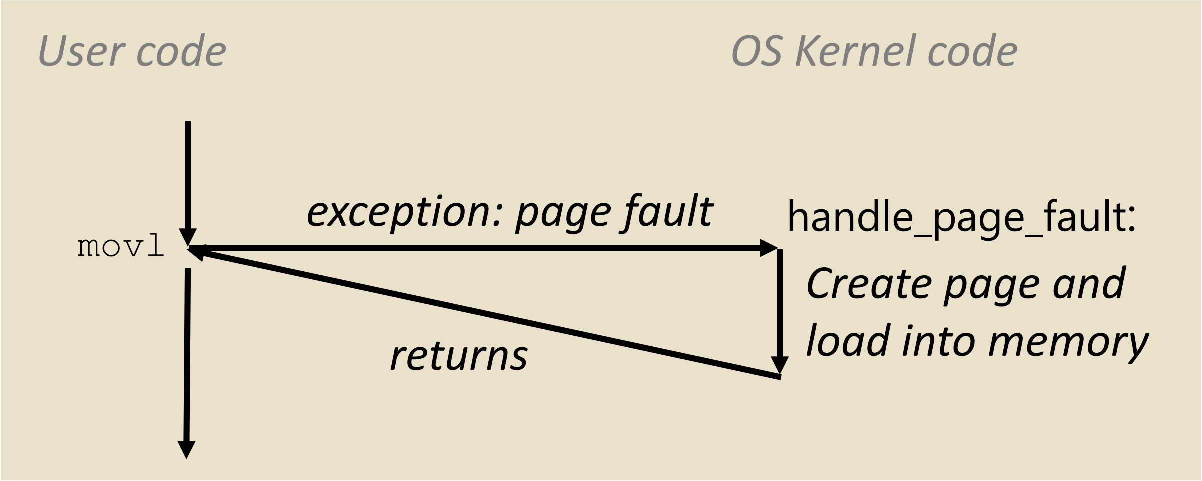
Virtual Addr: ________

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;
}
```

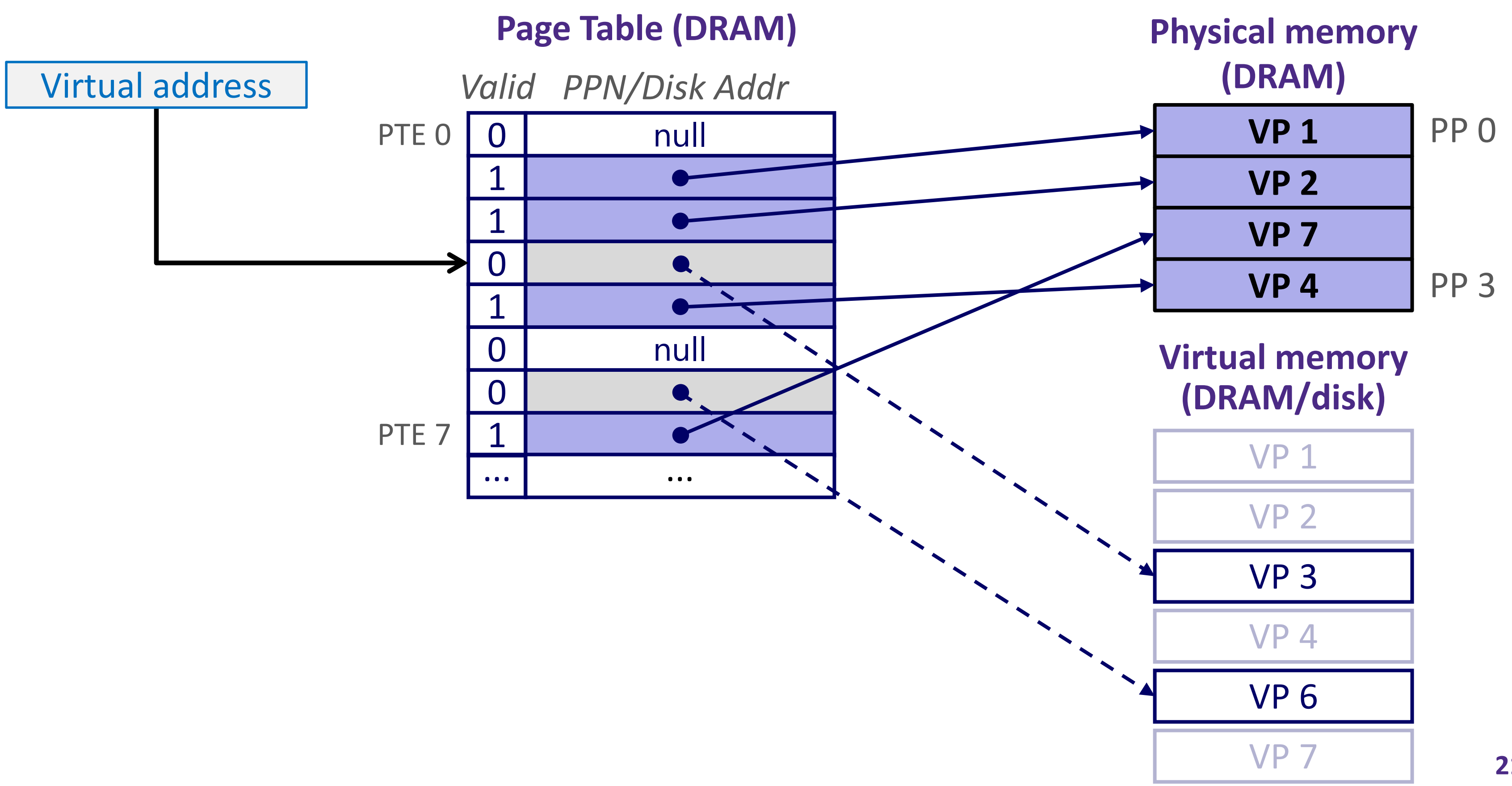
```
80483b7:      c7 05 10 9d 04 08 0d    movl    $0xd,0x8049d10
```



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

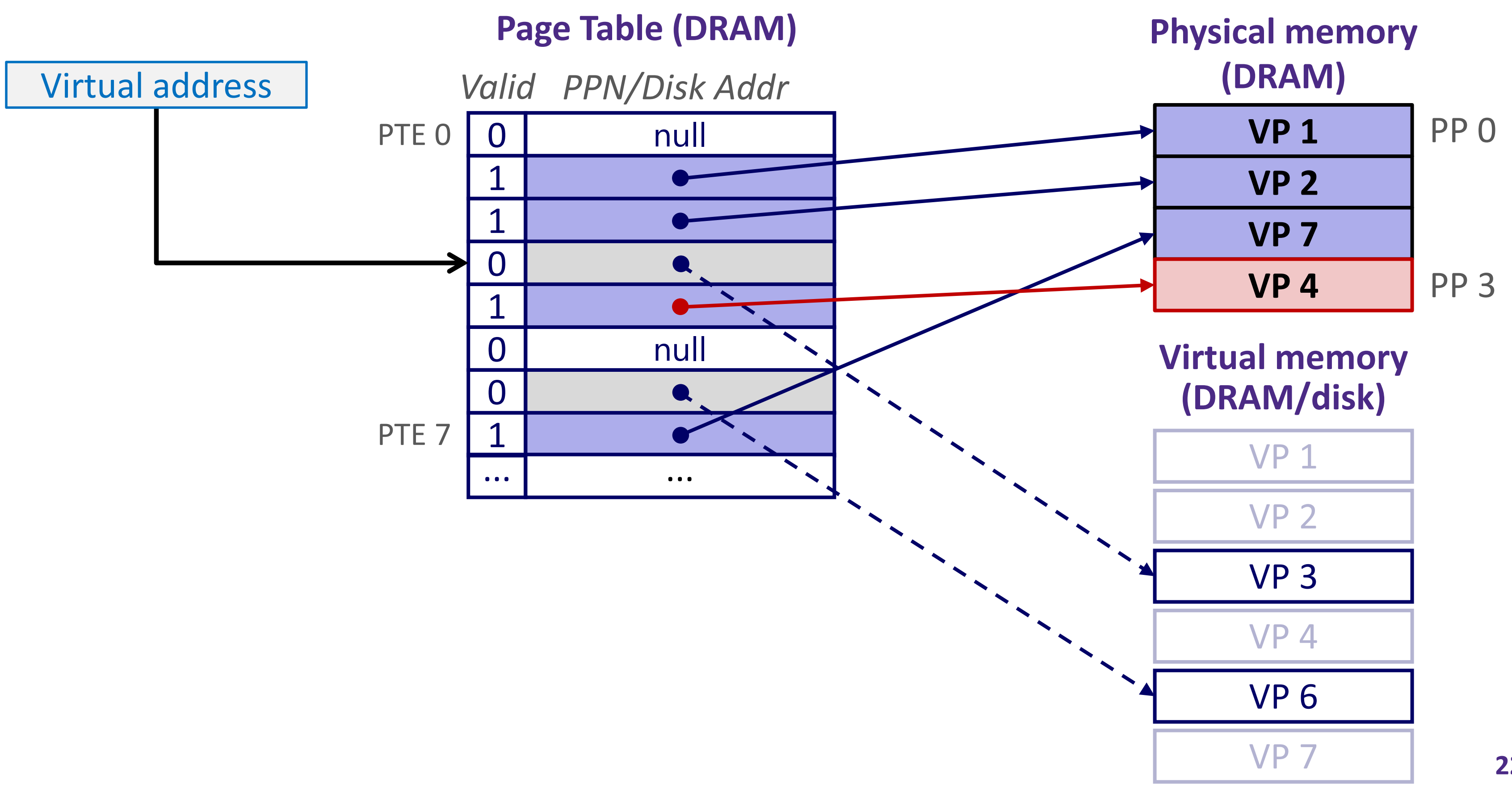
Handling a Page Fault

- Page miss causes page fault (an exception)



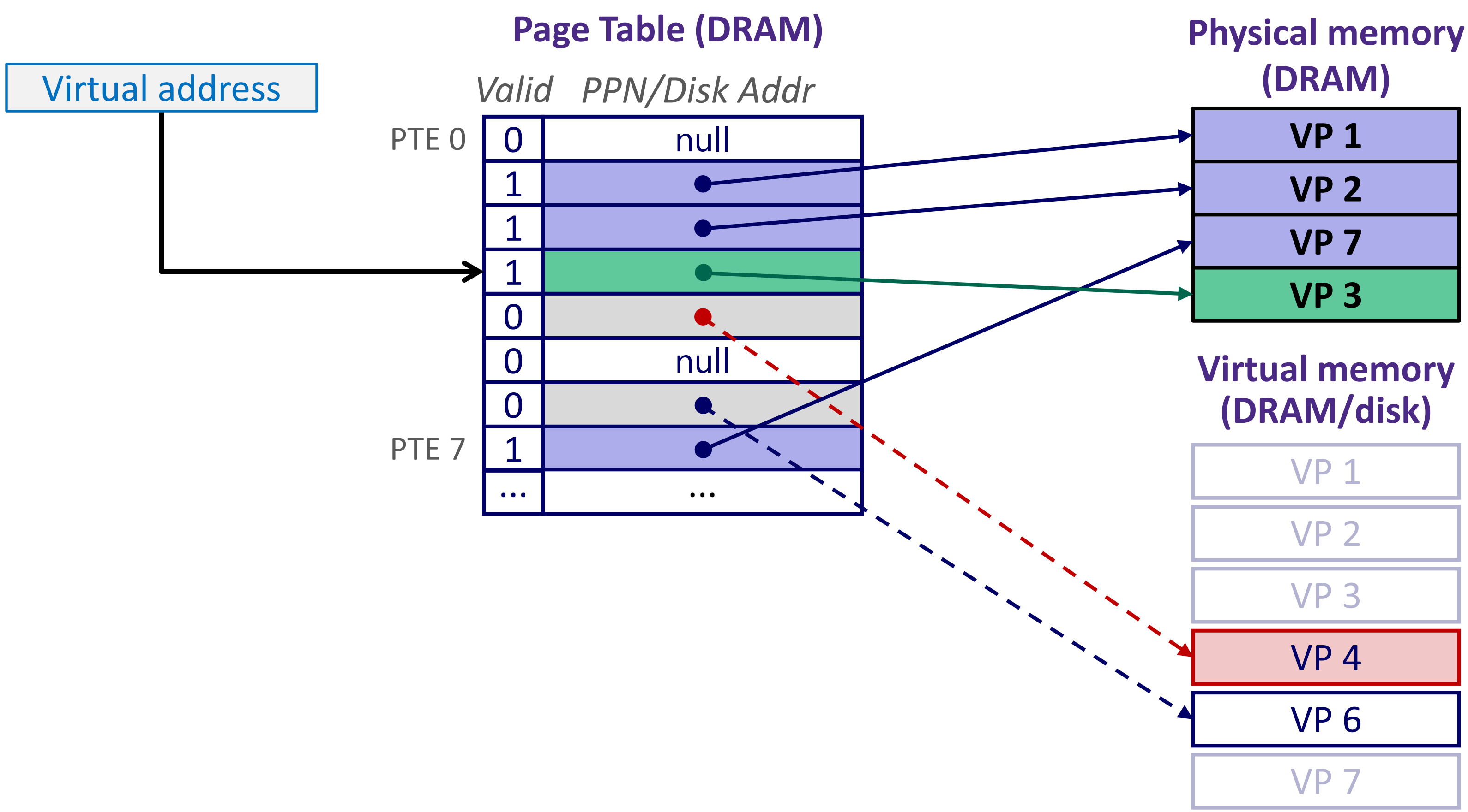
Handling a Page Fault

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



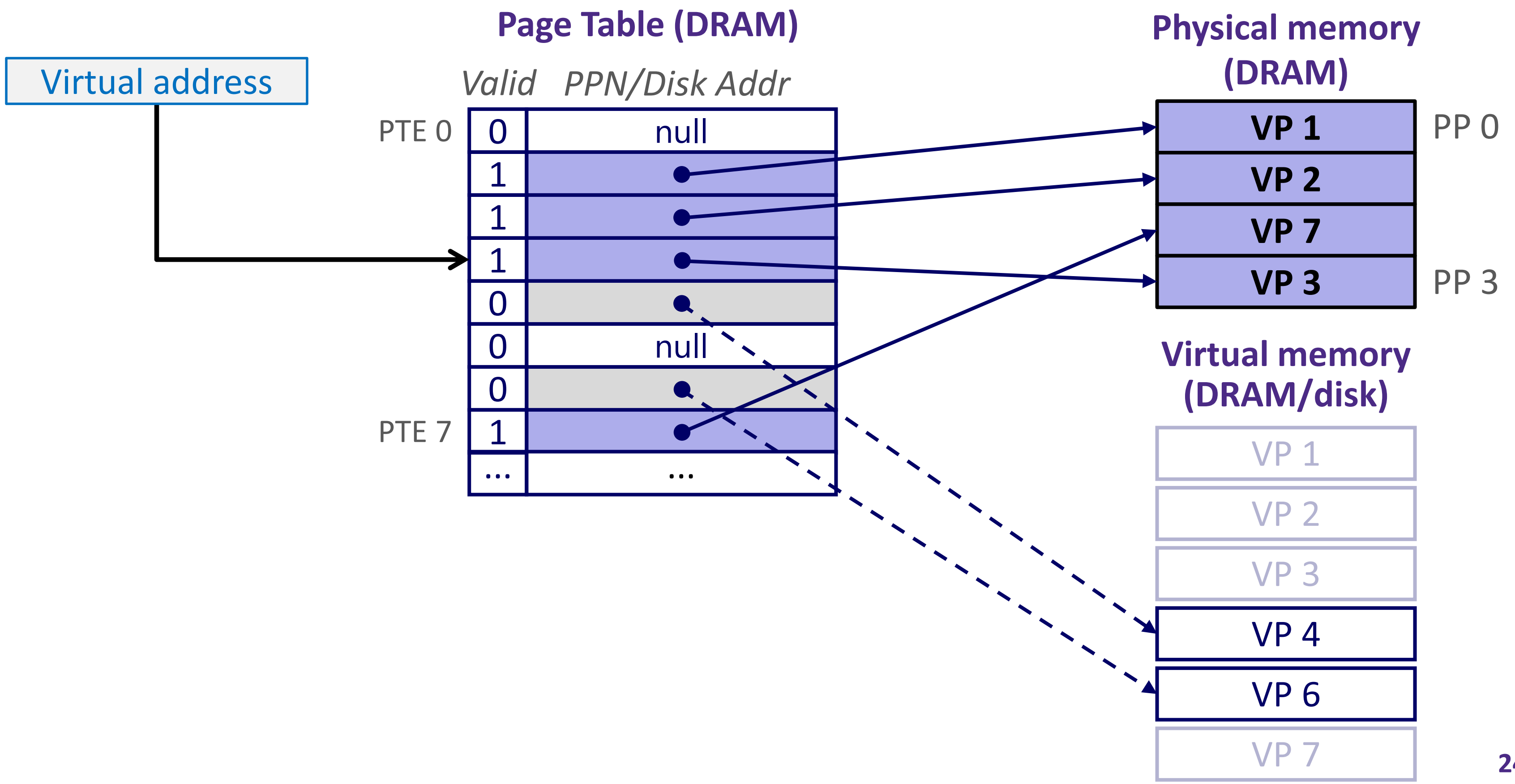
Handling a Page Fault

- 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!

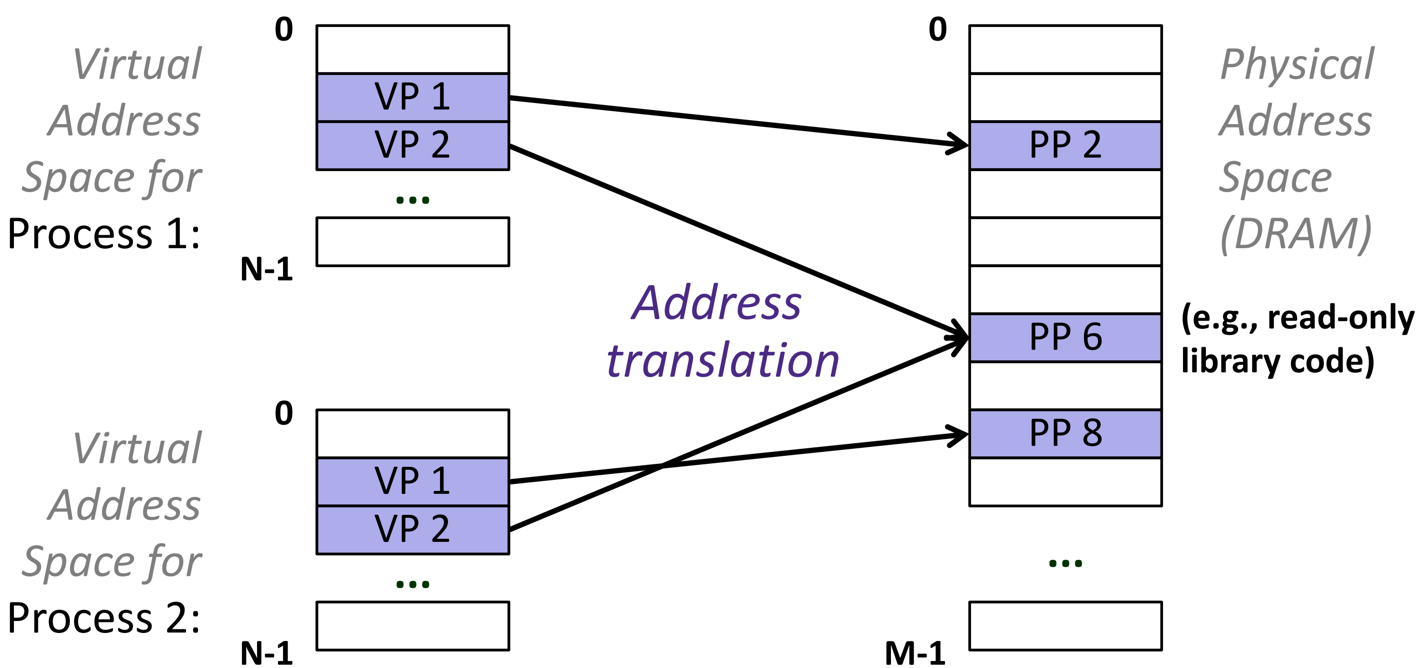


Virtual Memory (VM)

- 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**

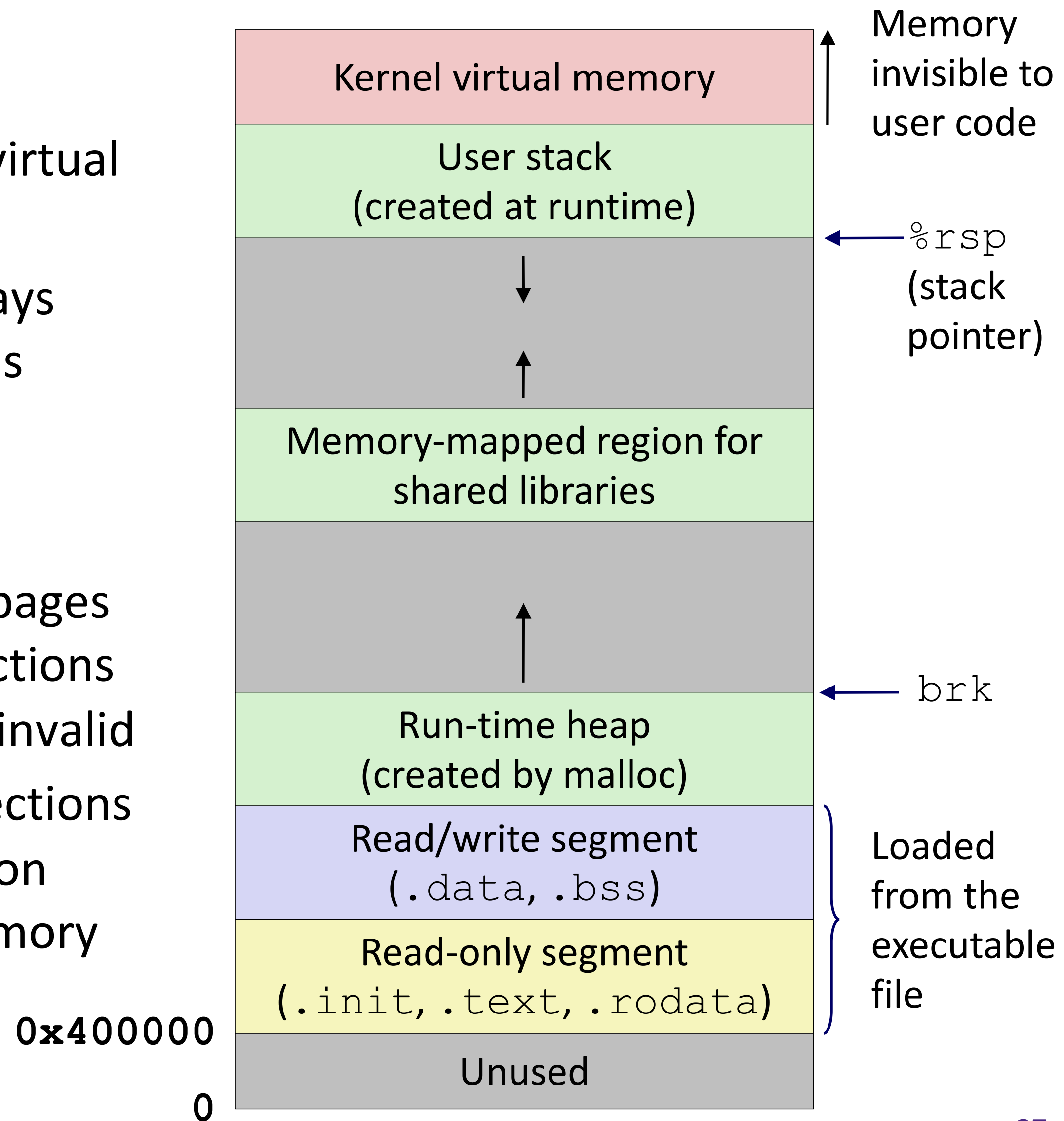
VM for Managing Multiple Processes

- 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!



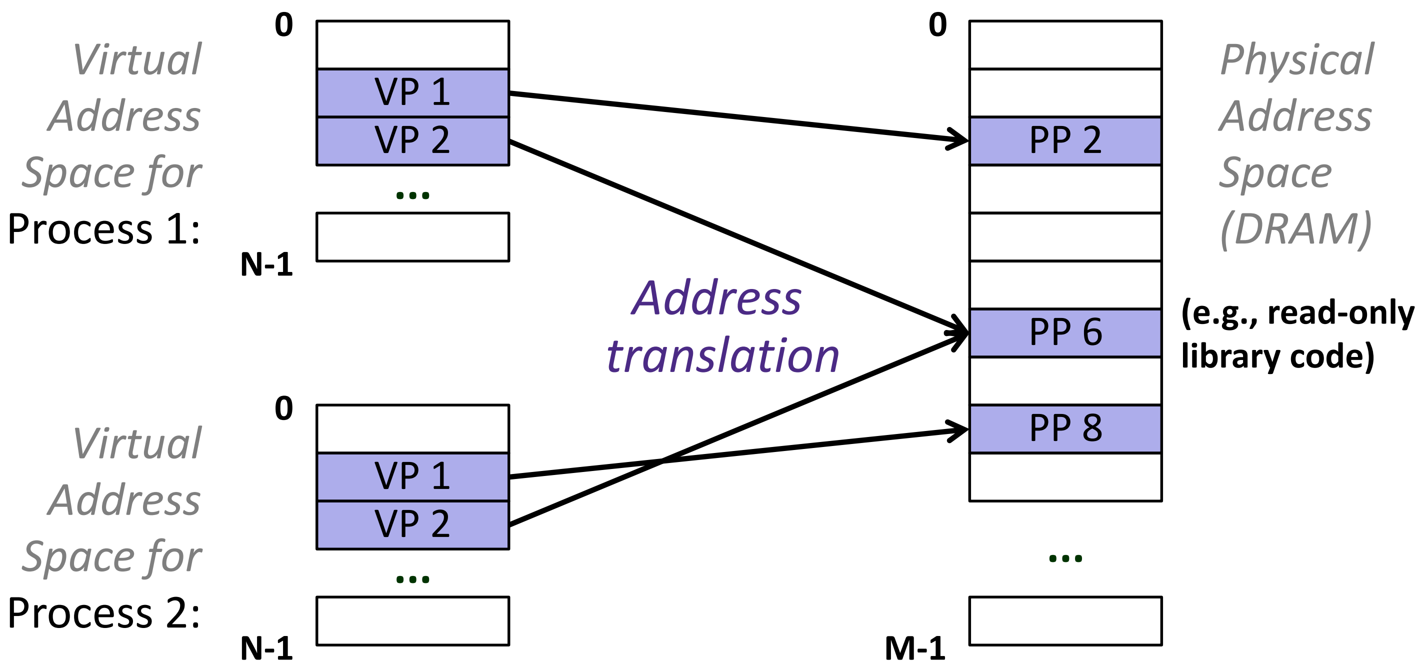
Simplifying Linking and Loading

- 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)

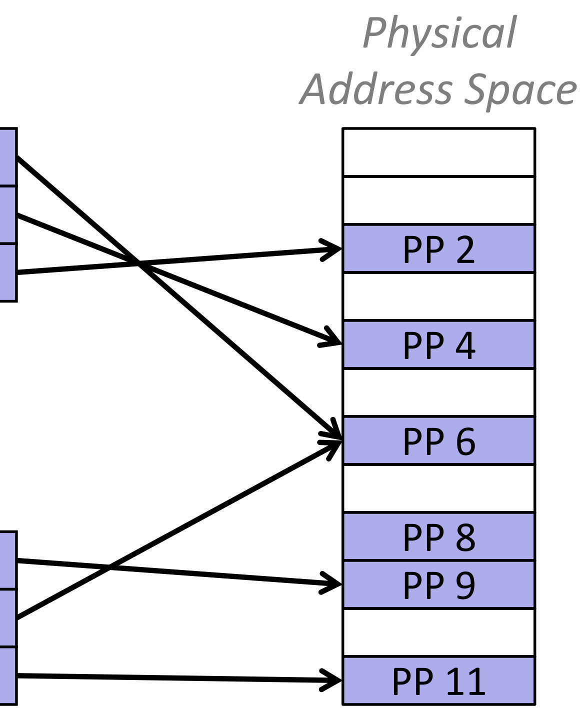
Process `i`:

	Valid	READ	WRITE	EXEC	PPN
VP 0:	Yes	Yes	No	No	PP 6
VP 1:	Yes	Yes	No	Yes	PP 4
VP 2:	Yes	Yes	Yes	No	PP 2

⋮

Process `j`:

	Valid	READ	WRITE	EXEC	PPN
VP 0:	Yes	Yes	Yes	No	PP 9
VP 1:	Yes	Yes	No	No	PP 6
VP 2:	Yes	Yes	Yes	No	PP 11



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			