CSE 153
Design of Operating Systems

Winter 2023

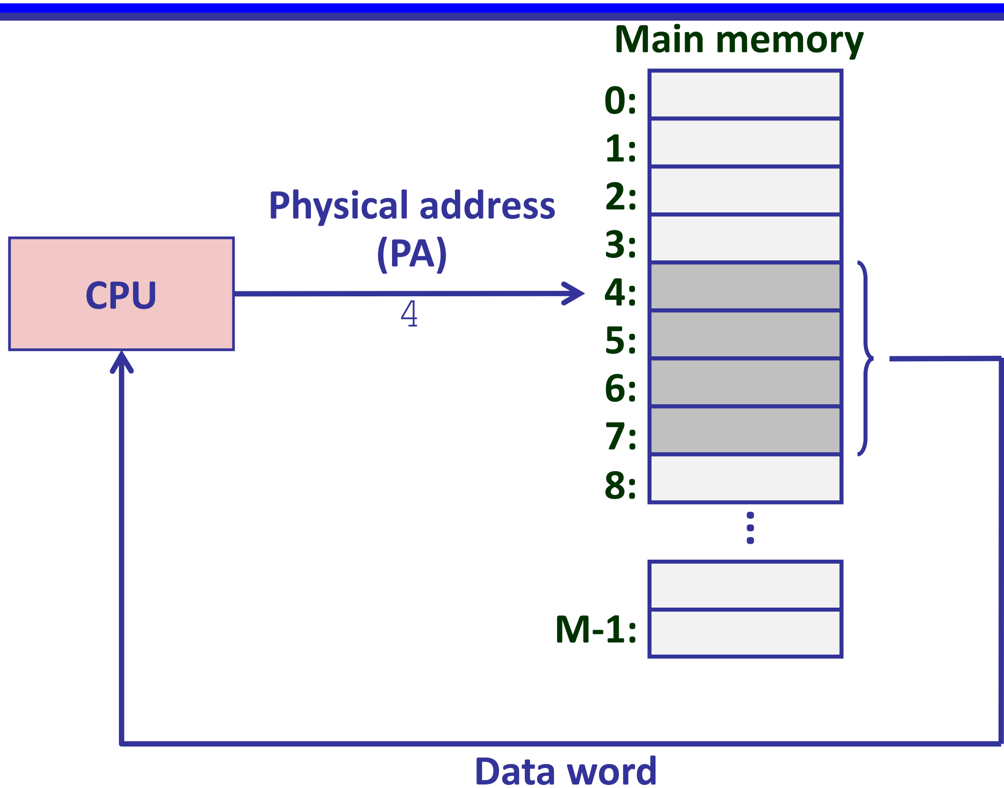
Lecture 15/16: Paging/Virtual Memory (1)

Some slides modified from originals by Dave O'hallaron

Today

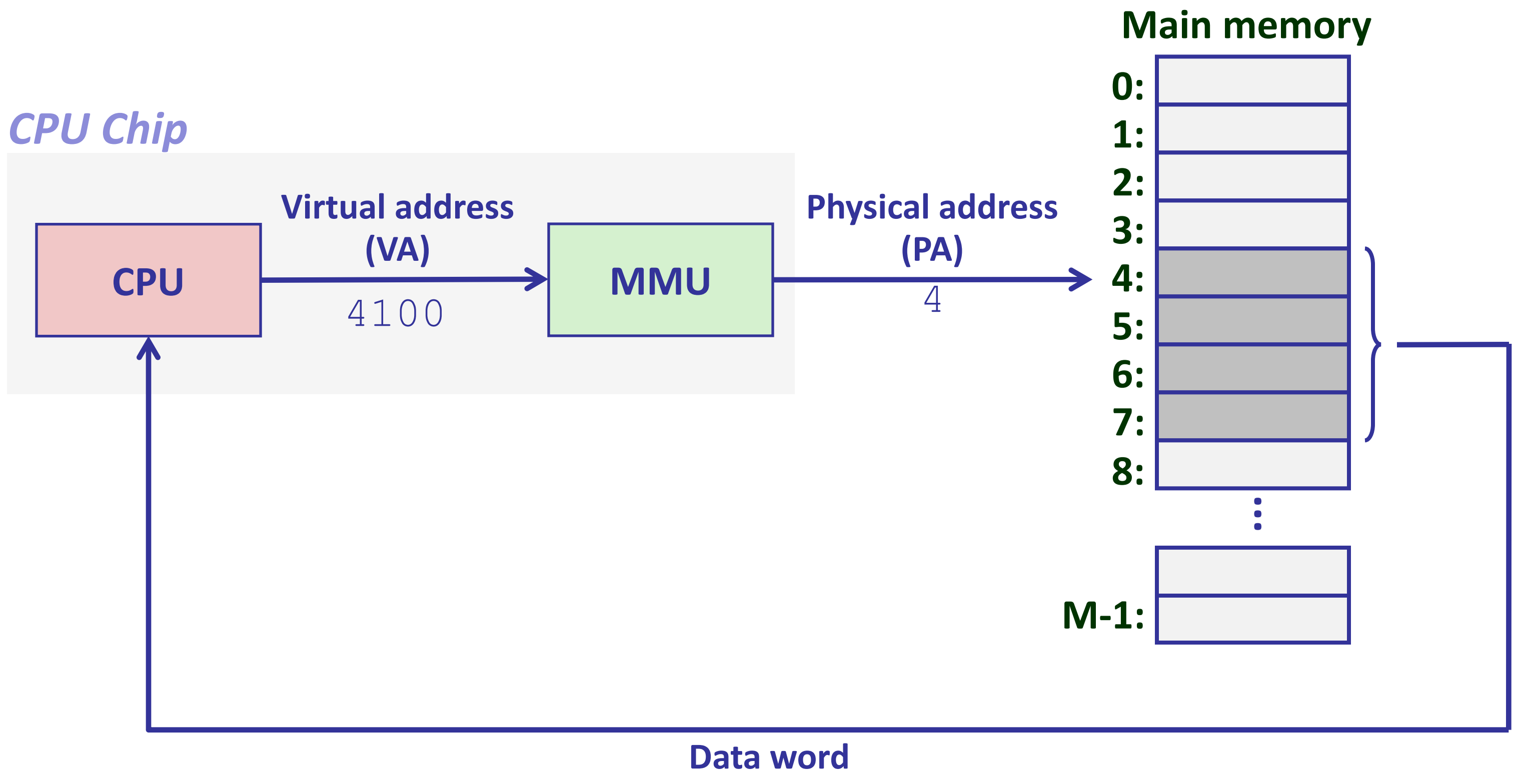
- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

A System Using Physical Addressing



- Used in "simple" systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames

A System Using Virtual Addressing



- Used in all modern servers, desktops, and laptops
- One of the great ideas in computer science

Address Spaces

- Linear address space: Ordered set of contiguous non-negative integer addresses:
  $\{0, 1, 2, 3 \ldots \}$
- Virtual address space: Set of $N = 2^n$ virtual addresses
  $\{0, 1, 2, 3, \ldots, N-1\}$
- Physical address space: Set of $M = 2^m$ physical addresses
  $\{0, 1, 2, 3, \ldots, M-1\}$
- Clean distinction between data (bytes) and their attributes (addresses)
- Each object can now have multiple addresses
- Every byte in main memory:
  one physical address, one (or more) virtual addresses

Why Virtual Memory (VM)?

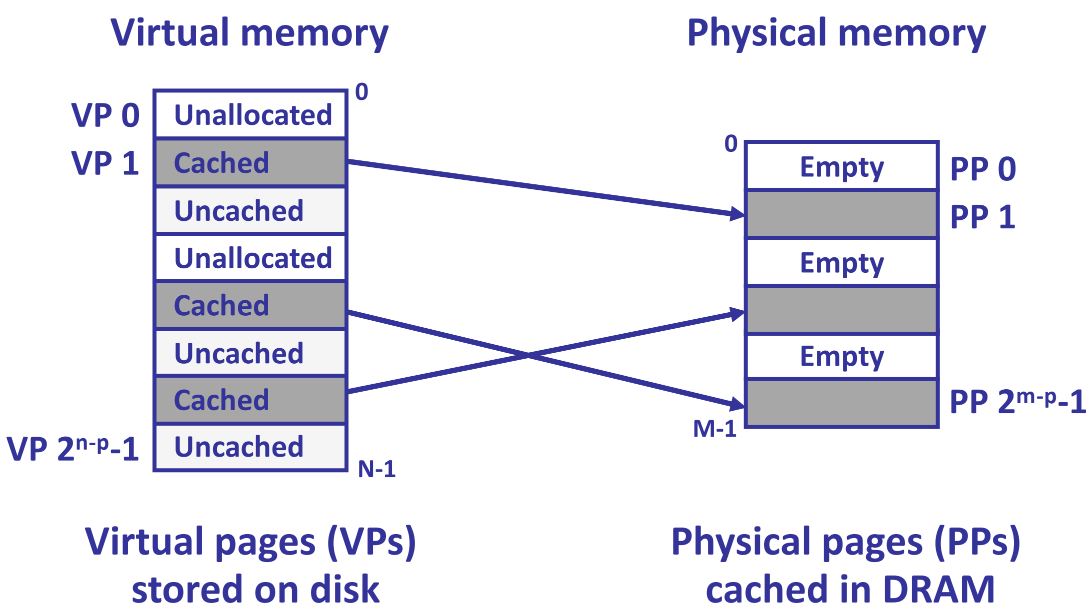
- Virtual memory is page with a new ingredient
  - Allow pages to be on disk
    - In a special partition (or file) called swap
- Motivation?
  - Uses main memory efficiently
  - Use DRAM as a cache for the parts of a virtual address space
- Simplifies memory management
  - Each process gets the same uniform linear address space
  - With VM, this can be big!

Today

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

VM as a Tool for Caching

- *Virtual memory* is an array of N contiguous bytes stored on disk.
- The contents of the array on disk are cached in *physical memory* (*DRAM cache*)
  - These cache blocks are called *pages* (size is $P = 2^p$ bytes)



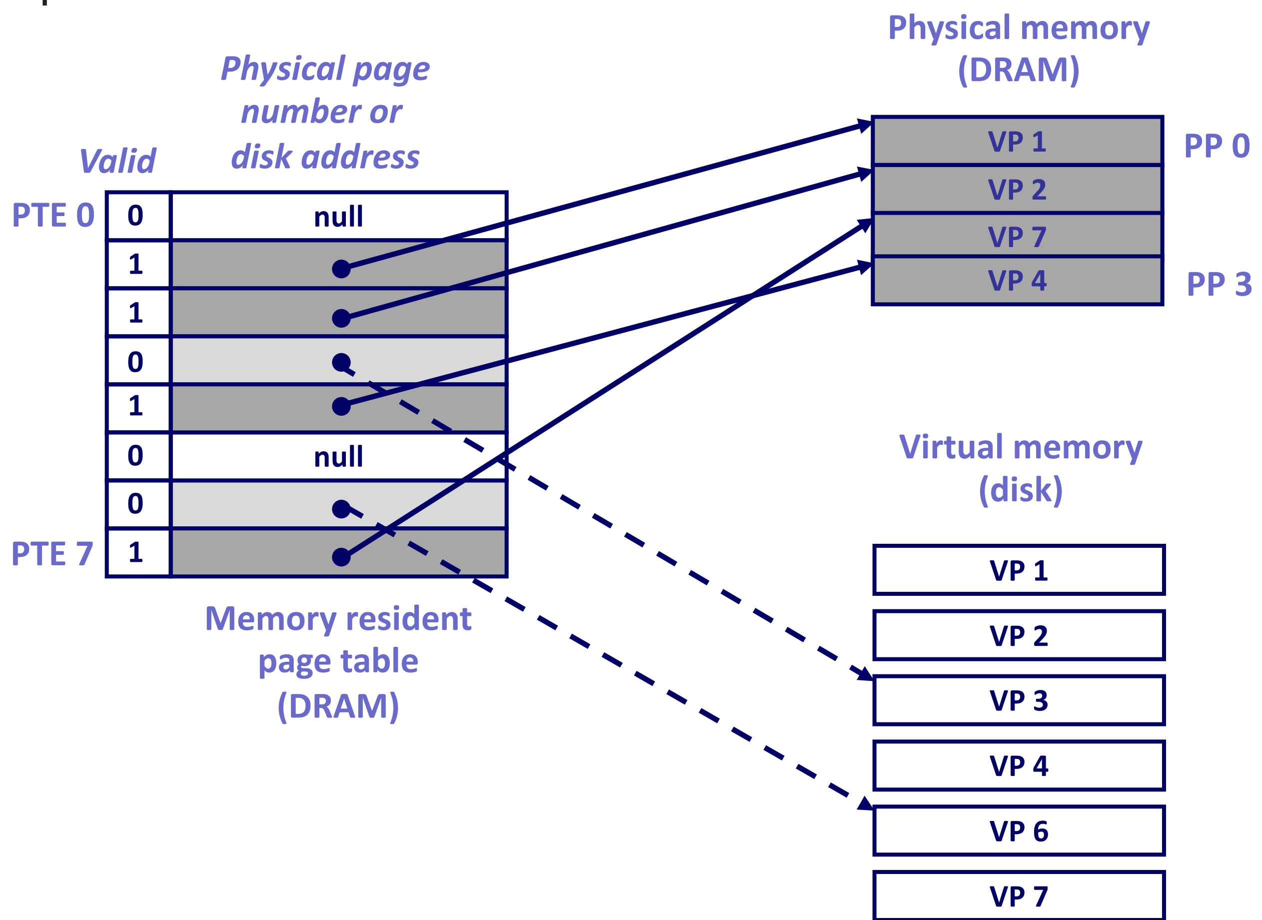
DRAM Cache Organization

- DRAM cache organization driven by the enormous miss penalty
  - DRAM is about ***10x*** slower than SRAM
  - Disk is about ***10,000x*** slower than DRAM

- Consequences
  - Large page (block) size: typically 4-8 KB, sometimes 4 MB
  - Fully associative
    - Any VP can be placed in any PP
    - Requires a “large” mapping function – different from CPU caches
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
  - Write-back rather than write-through

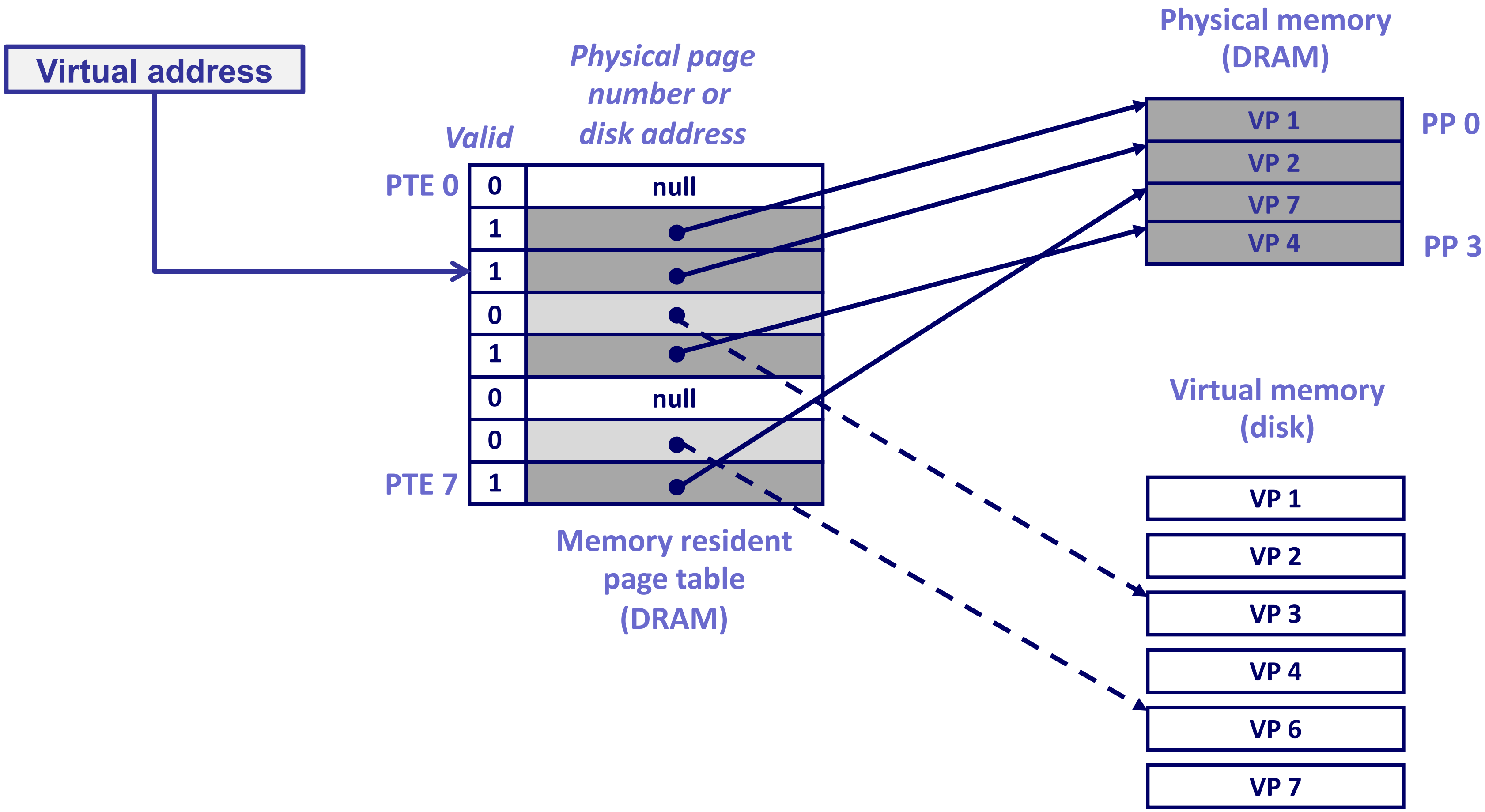
Page Tables

- A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.
  - Per-process kernel data structure in DRAM



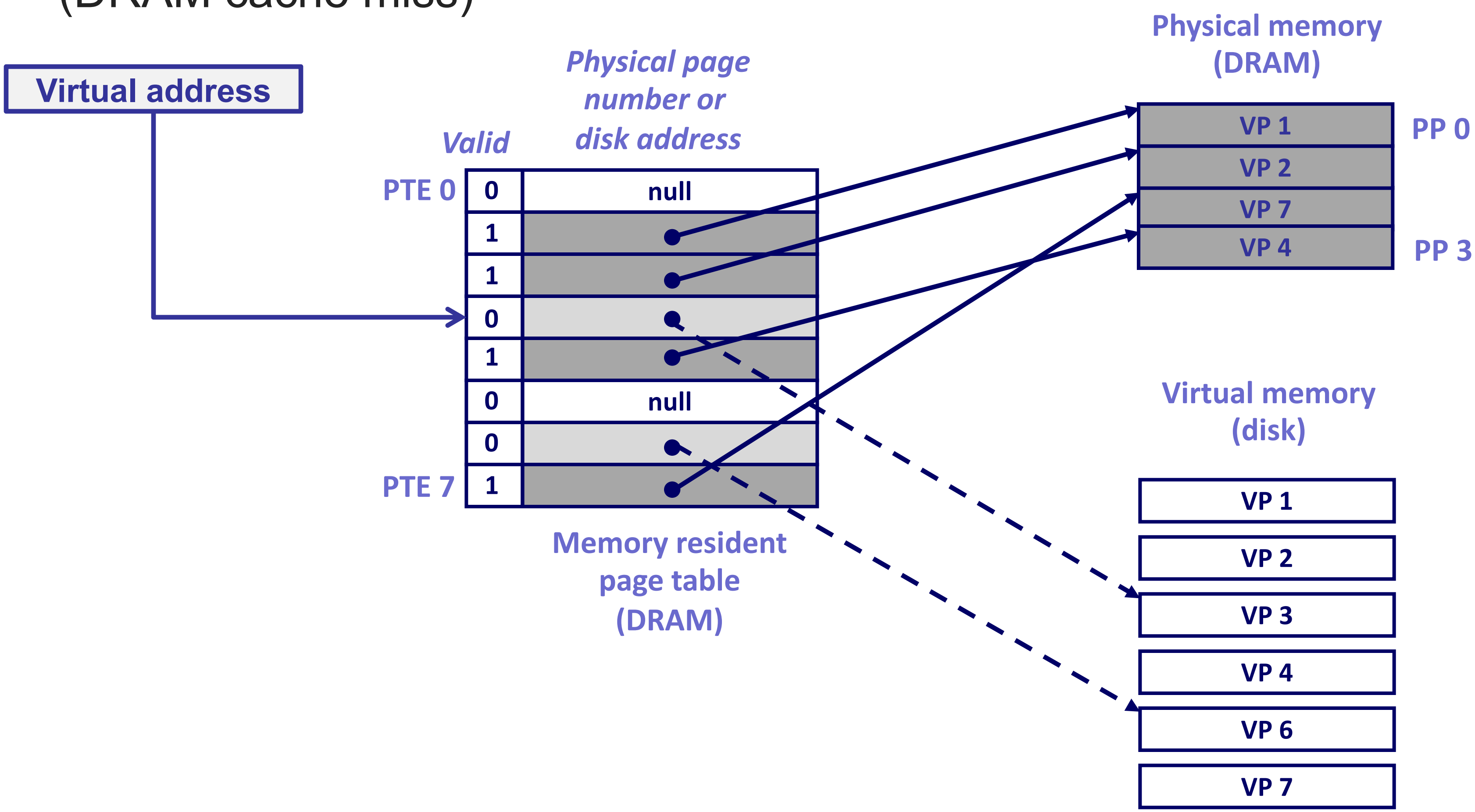
Page Hit

- *Page hit:* reference to VM word that is in physical memory (DRAM cache hit)



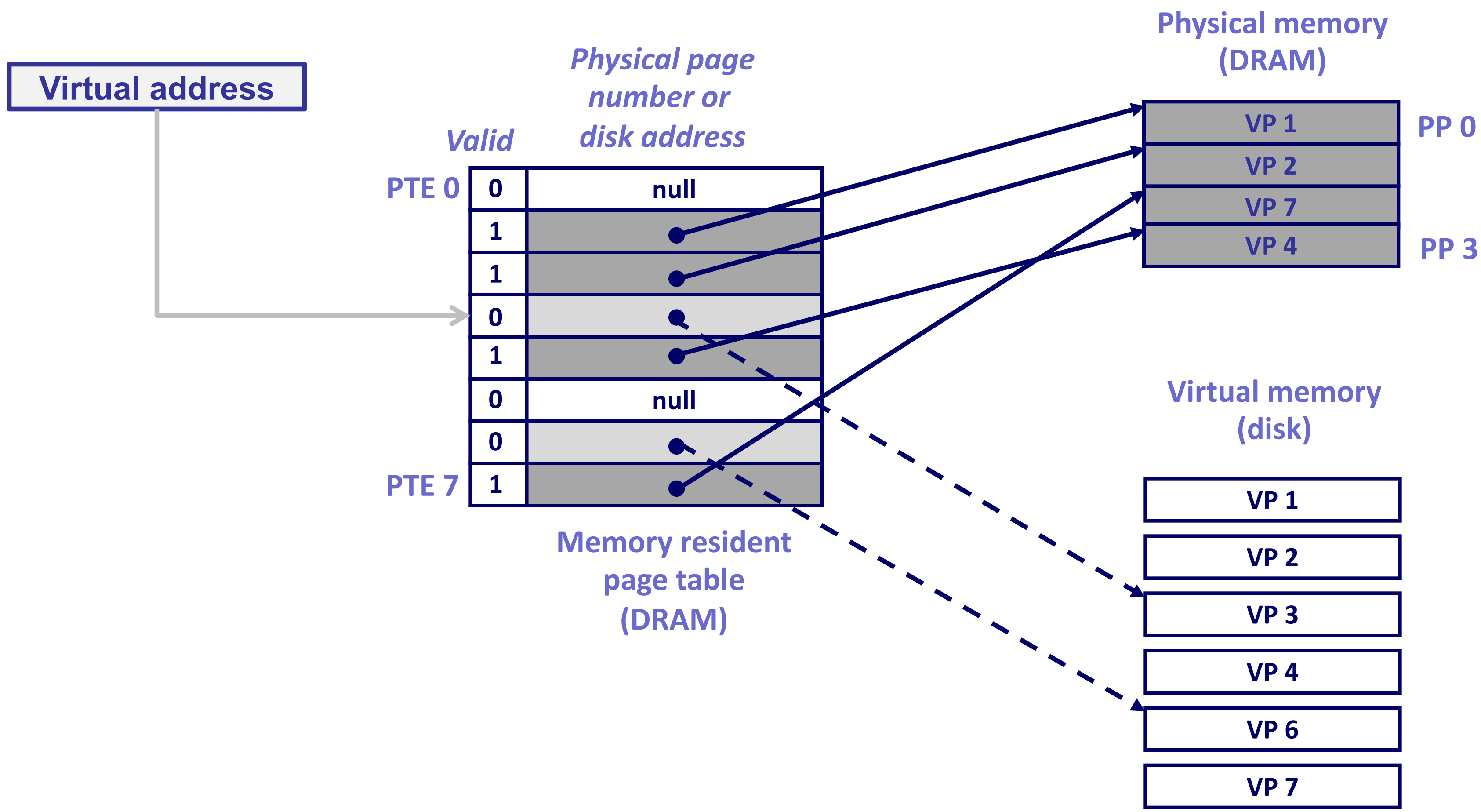
Page Fault

- *Page fault:* reference to VM word that is not in physical memory (DRAM cache miss)



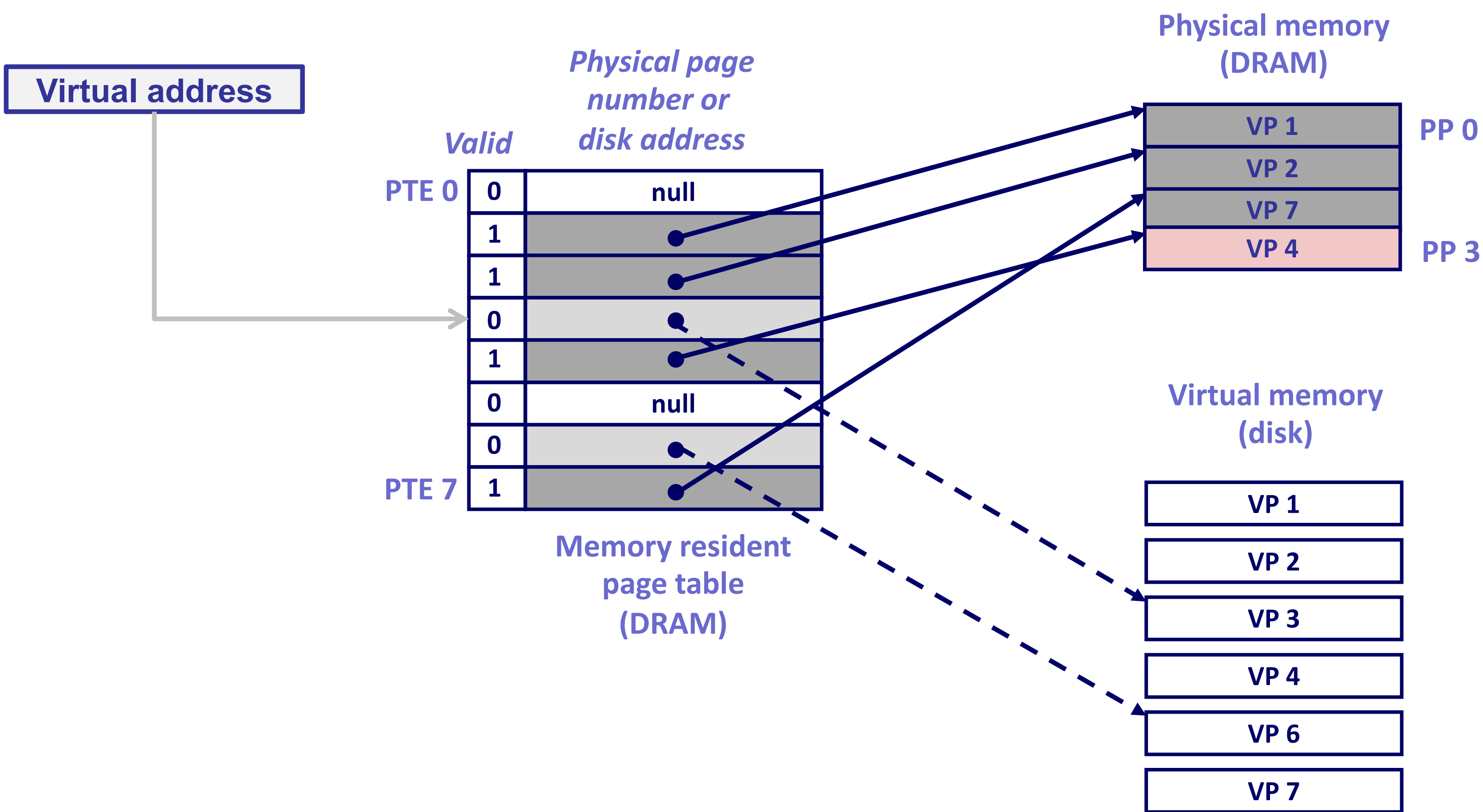
Handling Page Fault

- Page miss causes page fault (an exception)



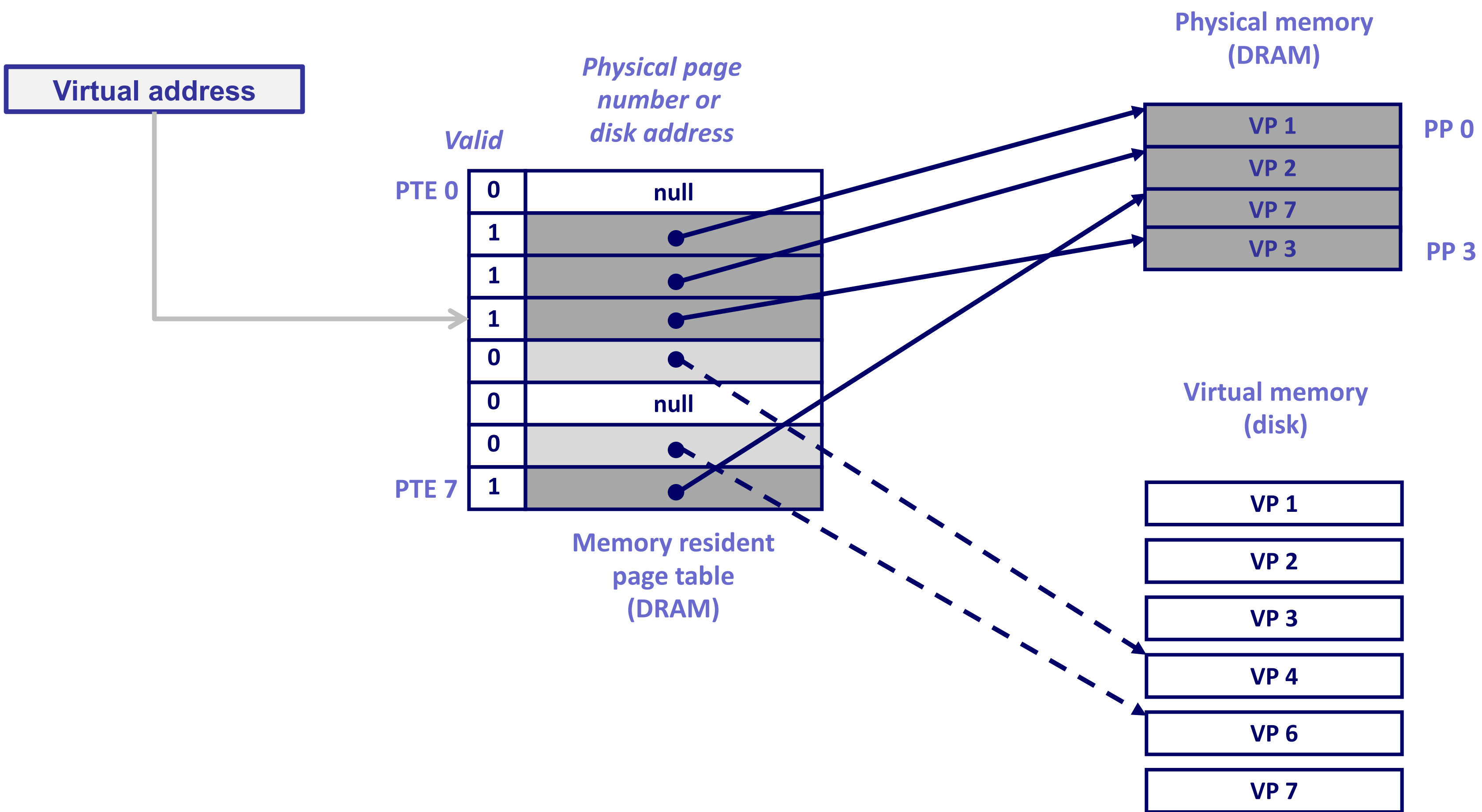
Handling Page Fault

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



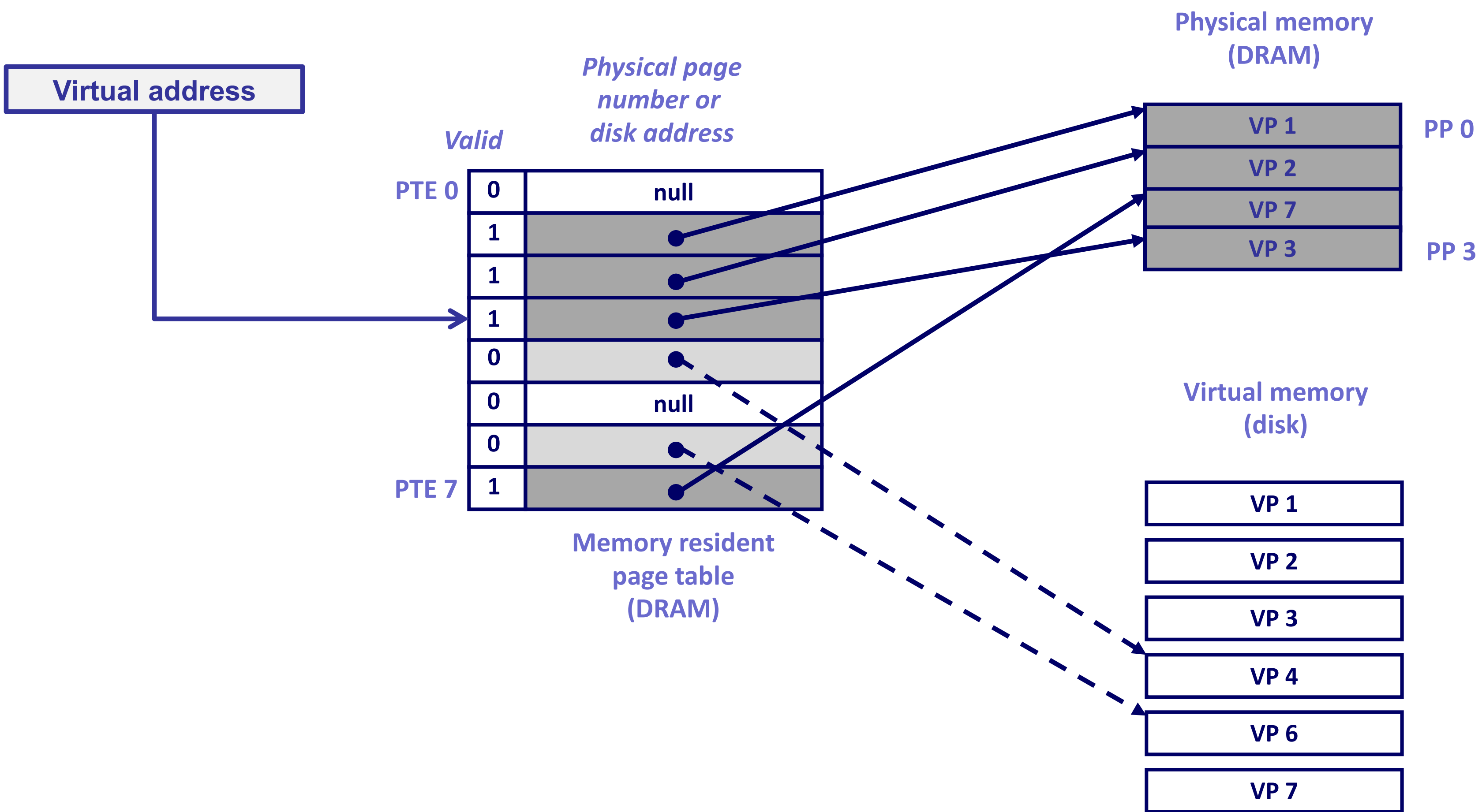
Handling Page Fault

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



Handling 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!



Locality to the Rescue!

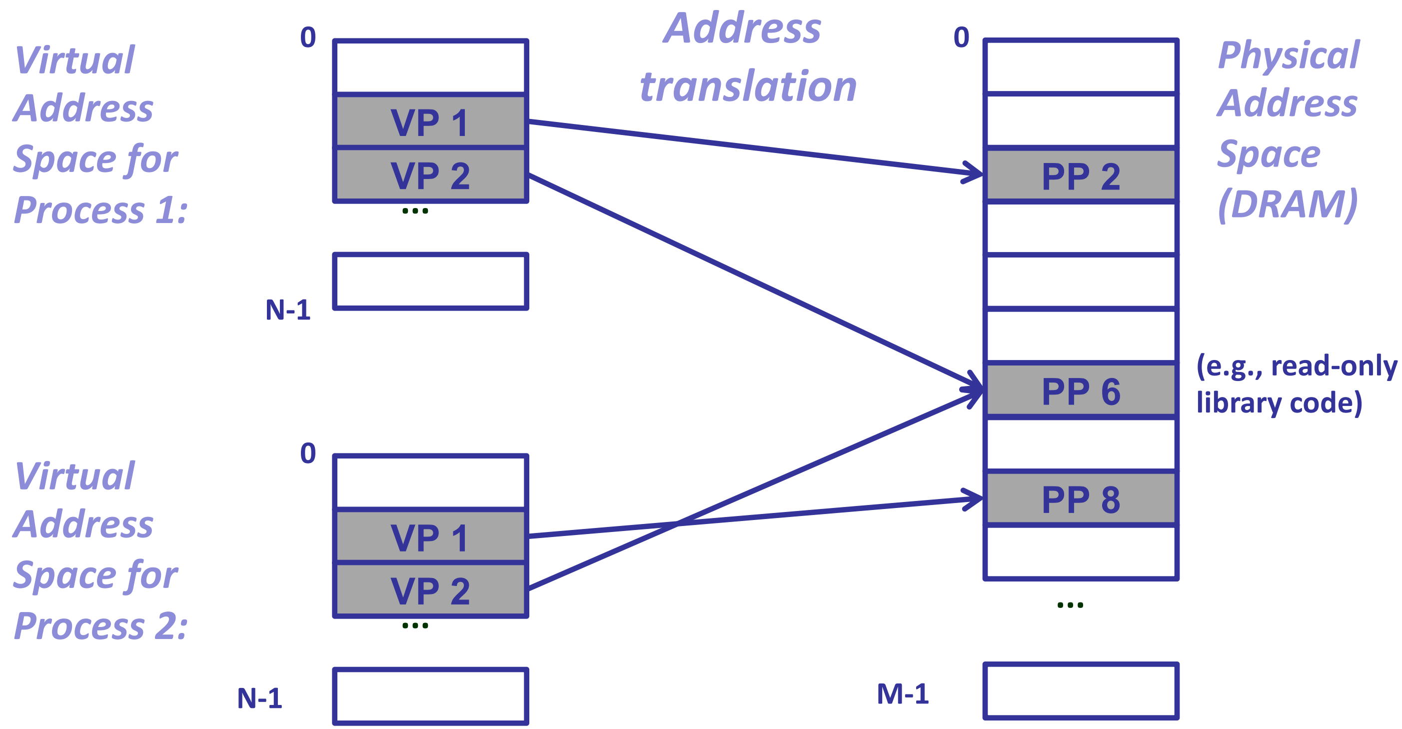
- Virtual memory works because of locality
- At any point in time, programs tend to access a set of active virtual pages called the *working set*
  - Programs with better temporal locality will have smaller working sets
- If (working set size < main memory size)
  - Good performance for one process after compulsory misses
- If (SUM(working set sizes) > main memory size)
  - *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously

Today

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

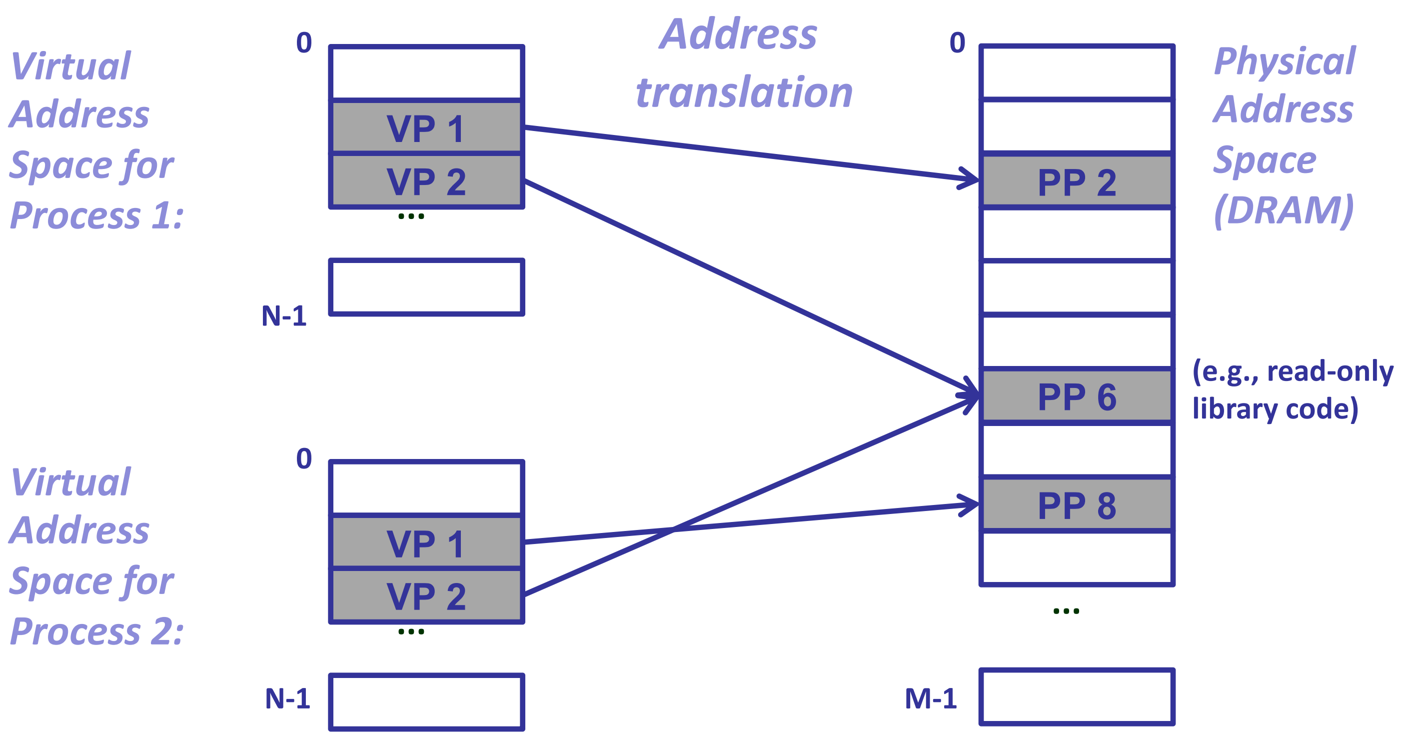
VM as a Tool for Memory Management

- Key idea: each process has its own virtual address space
  - It can view memory as a simple linear array
  - Mapping function scatters addresses through physical memory
    - » Well chosen mappings simplify memory allocation and management



VM as a Tool for Memory Management

- Memory allocation
  - Each virtual page can be mapped to any physical page
  - A virtual page can be stored in different physical pages at different times
- Sharing code and data among processes
  - Map virtual pages to the same physical page (here: PP 6)



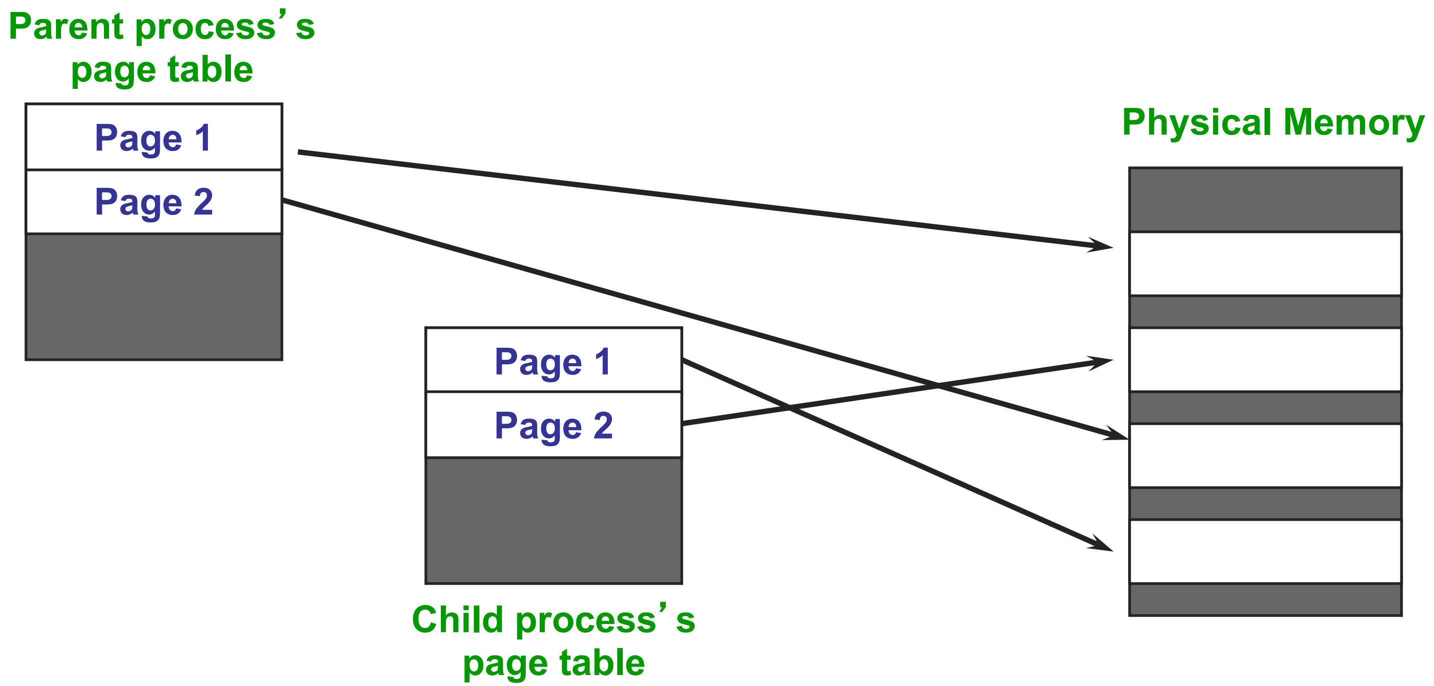
Sharing

- Can map shared memory at same or different virtual addresses in each process' address space
  - Different:
    - » 10^{th} virtual page in P1 and 7^{th} virtual page in P2 correspond to the 2^{nd} physical page
    - » Flexible (no address space conflicts), but pointers inside the shared memory segment are invalid
  - Same:
    - » 2^{nd} physical page corresponds to the 10^{th} virtual page in both P1 and P2
    - » Less flexible, but shared pointers are valid

Copy on Write

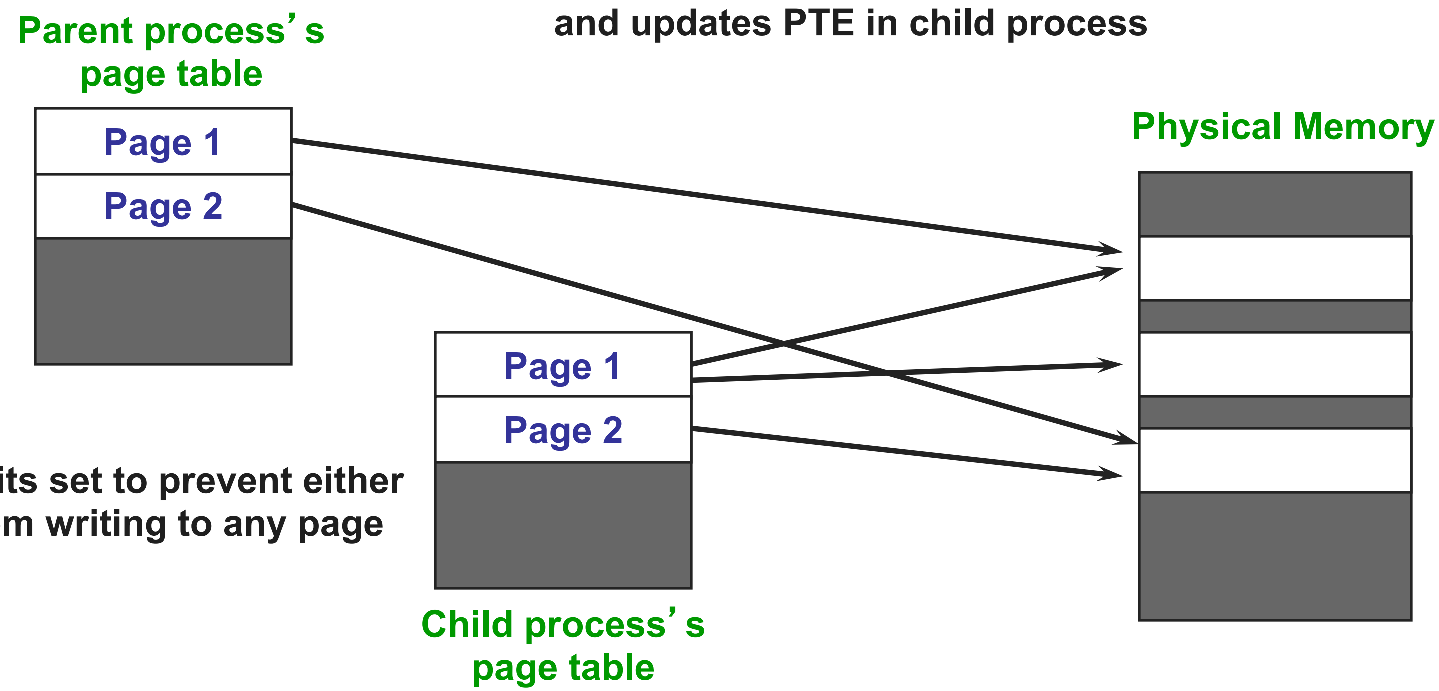
- OSes spend a lot of time copying data
  - System call arguments between user/kernel space
  - Entire address spaces to implement fork()
- Use Copy on Write (CoW) to defer large copies as long as possible, hoping to avoid them altogether
  - Instead of copying pages, create shared mappings of parent pages in child virtual address space
  - Shared pages are protected as read-only in parent and child
    - » Reads happen as usual
    - » Writes generate a protection fault, trap to OS, copy page, change page mapping in client page table, restart write instruction
  - How does this help fork()?

Execution of fork()



fork() with Copy on Write

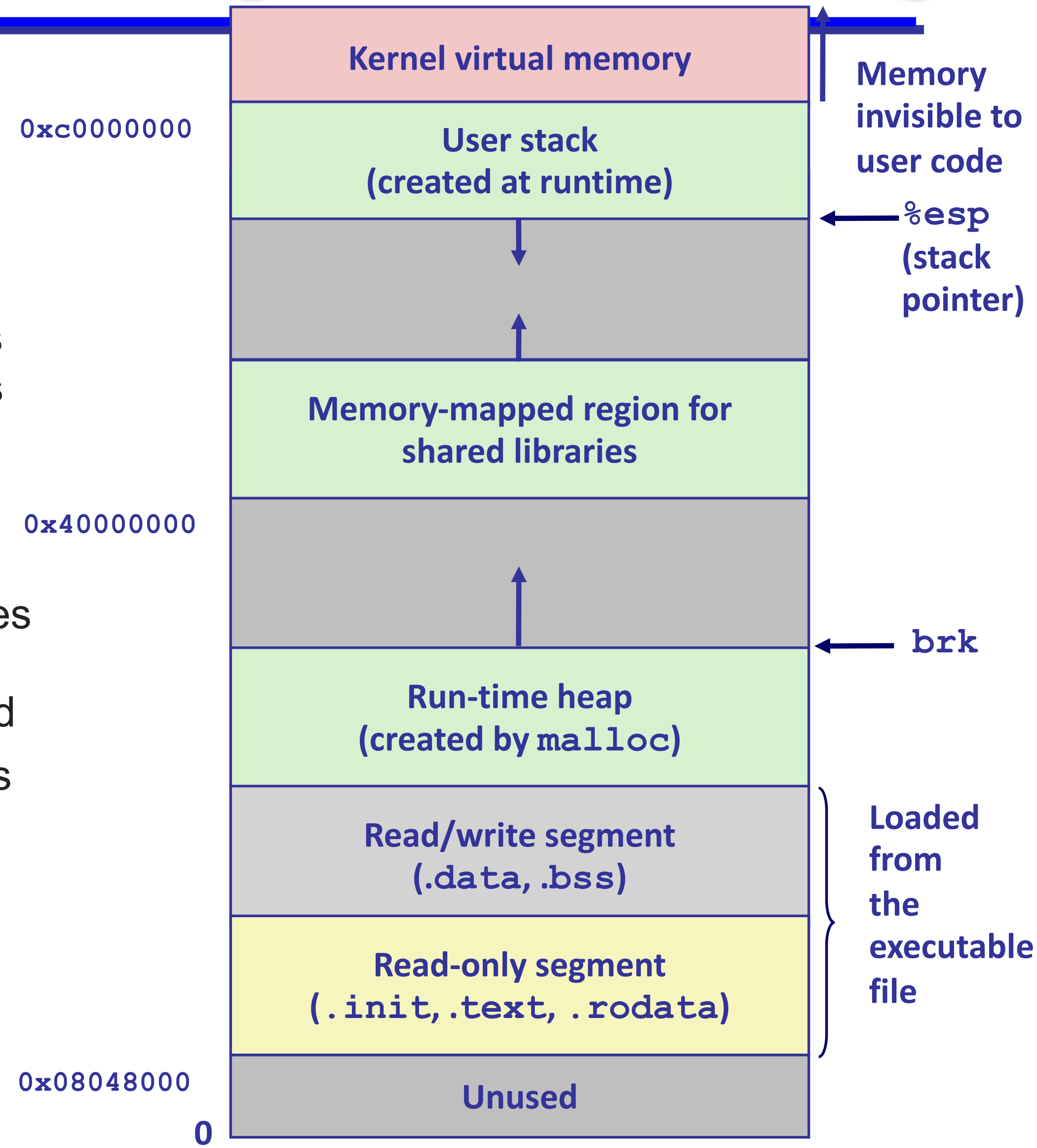
When either process modifies Page 1, page fault handler allocates new page and updates PTE in child process



Protection bits set to prevent either process from writing to any page

Simplifying Linking and Loading

- Linking
  - Each program has similar virtual address space
  - Code, stack, and shared libraries always start at the same address
- 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



Today

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

VM as a Tool for Memory Protection

- Extend PTEs with permission bits
- Page fault handler checks these before remapping
  - If violated, send process SIGSEGV (segmentation fault)

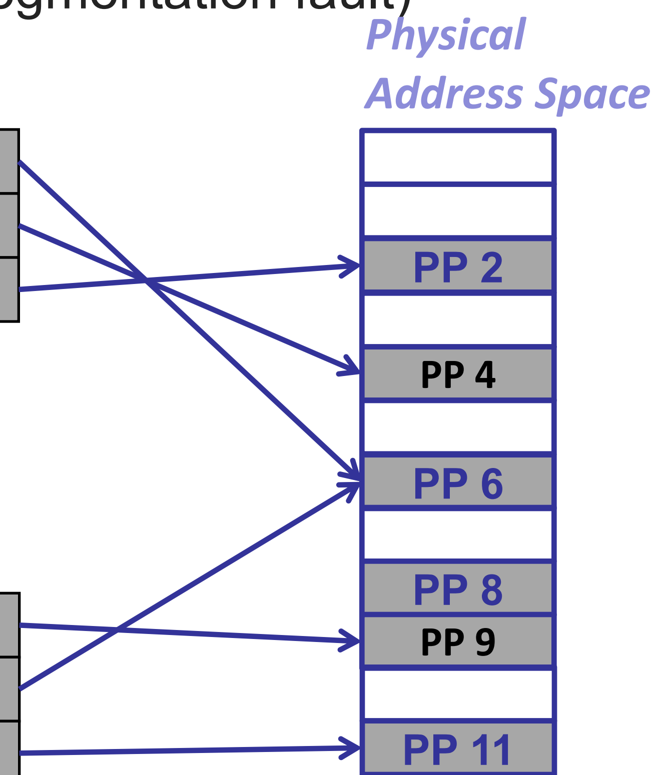
Process i:

	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 6
VP 1:	No	Yes	Yes	PP 4
VP 2:	Yes	Yes	Yes	PP 2

⋮

Process j:

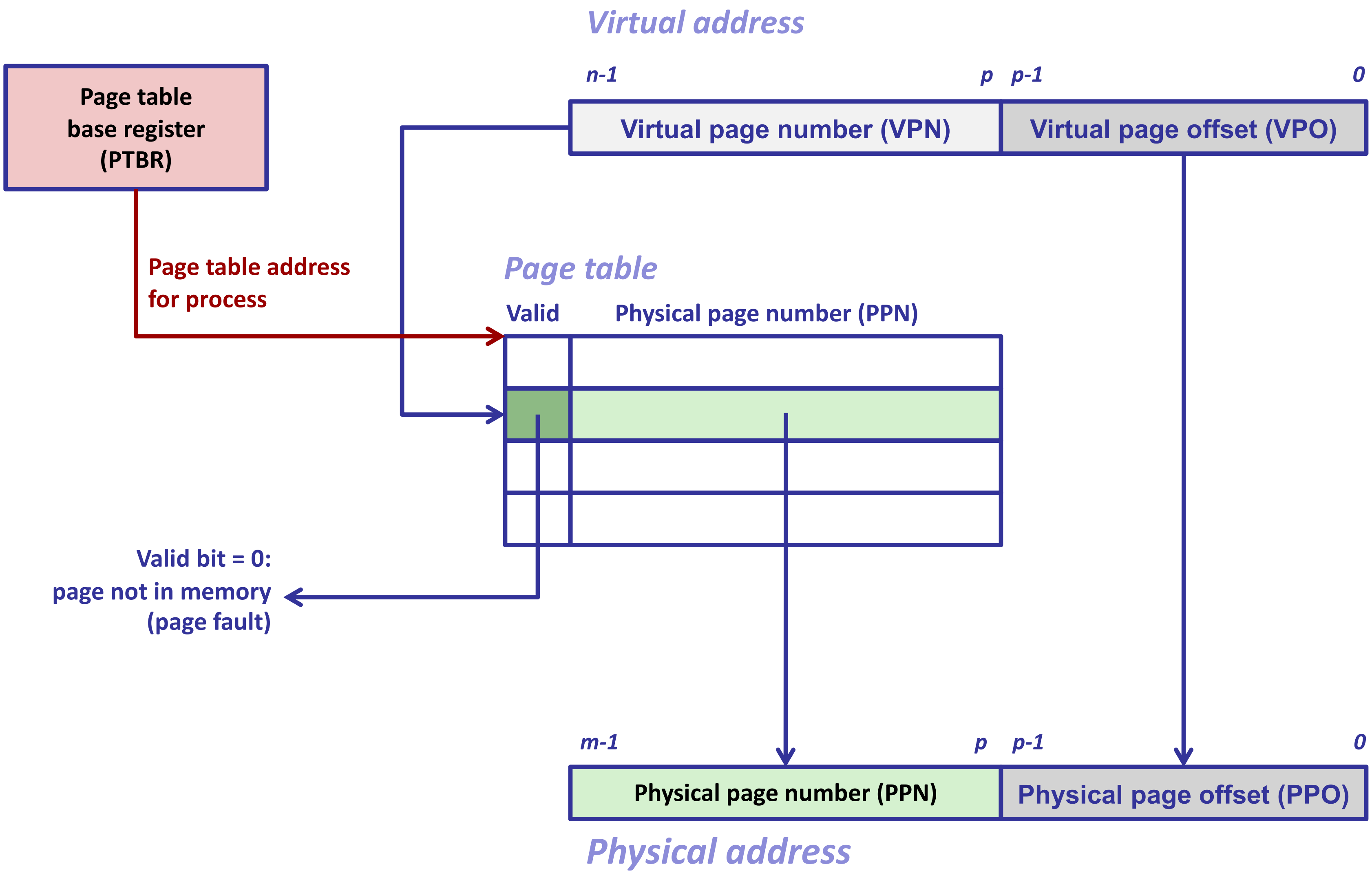
	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 9
VP 1:	Yes	Yes	Yes	PP 6
VP 2:	No	Yes	Yes	PP 11



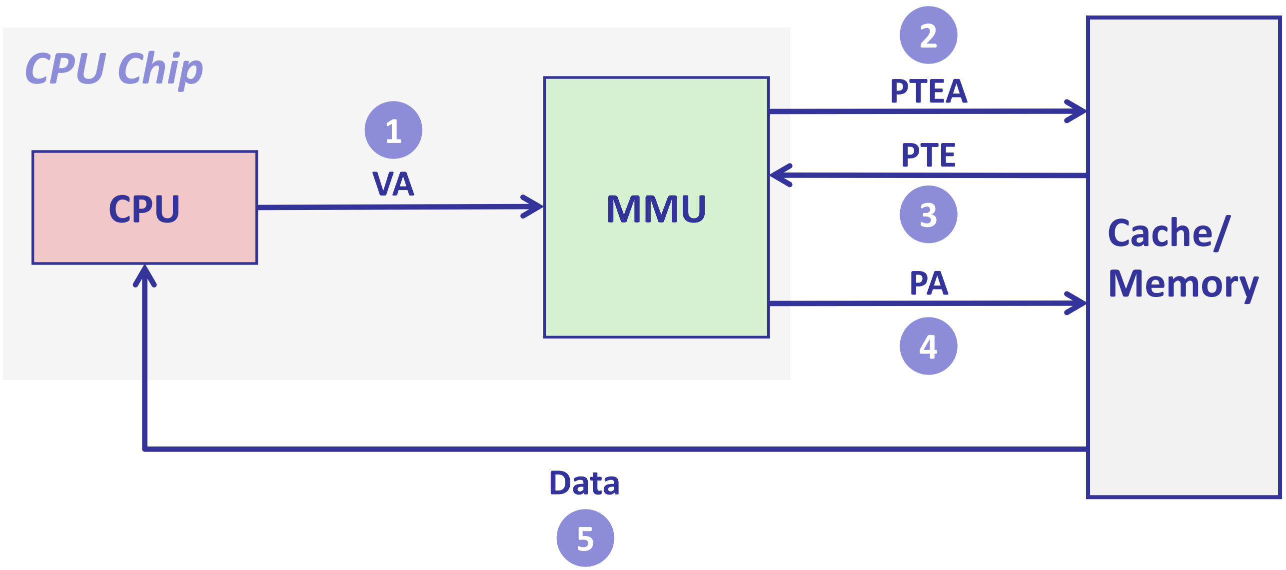
Today

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

Address Translation With a Page Table



Address Translation: Page Hit



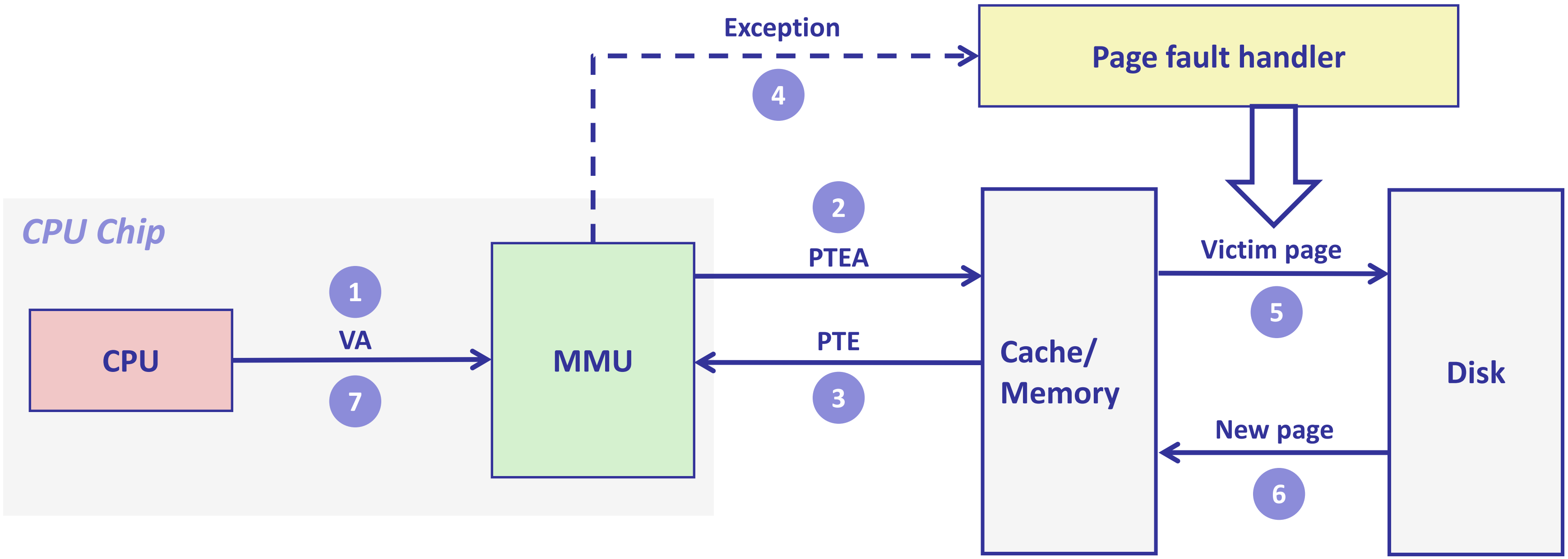
1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

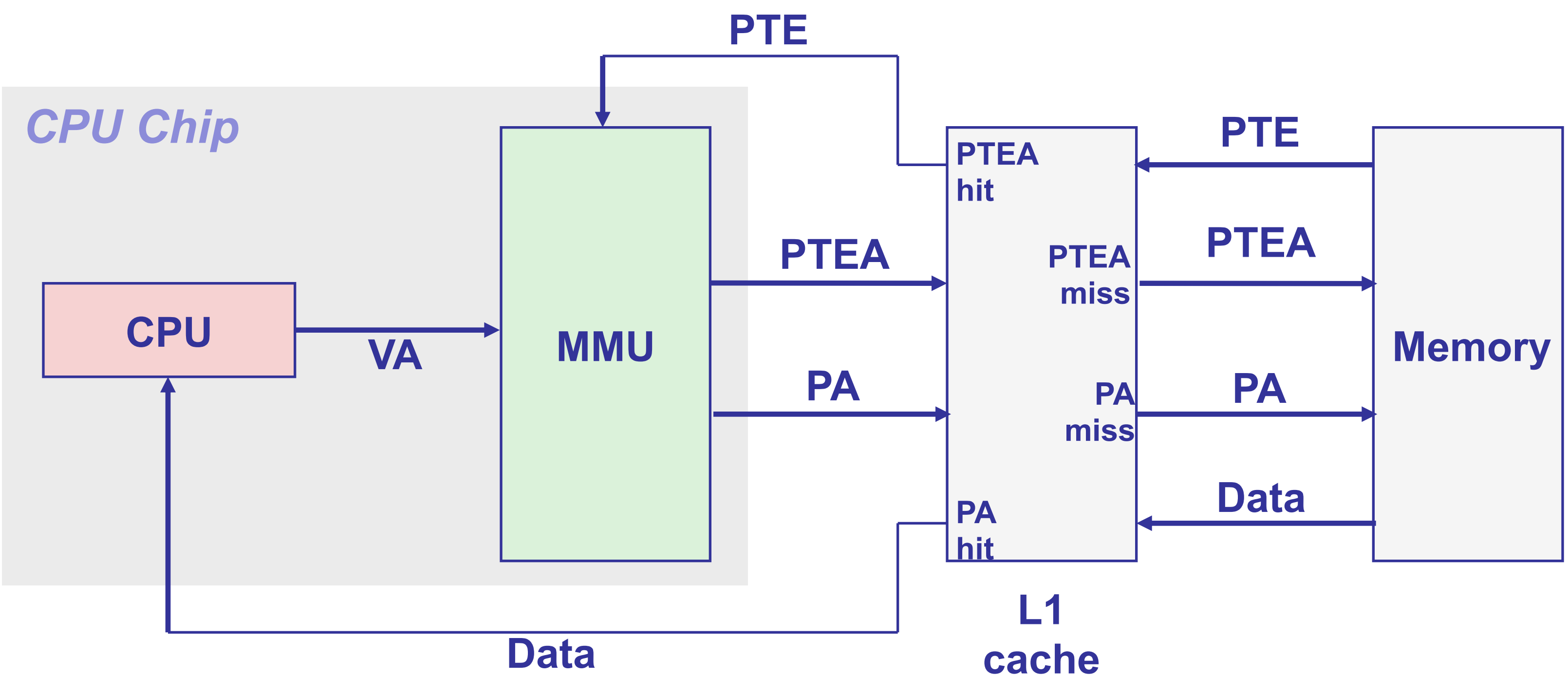
4) Valid bit is zero, so MMU triggers page fault exception

5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction

Integrating VM and Cache



VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Elephant(s) in the room

- Problem 1: Translation is slow!
  - Many memory accesses for each memory access
  - Caches are useless!

- Problem 2: Page table can be gigantic!
- We need one for each process
- All your memory are belong to us!

“Unfortunately, there’s another elephant in the room.”