UNM SCHOOL of ENGINEERING
Department of Computer Science

Lecture #26
Virtual Memory

Prof. Soraya Abad-Mota, PhD

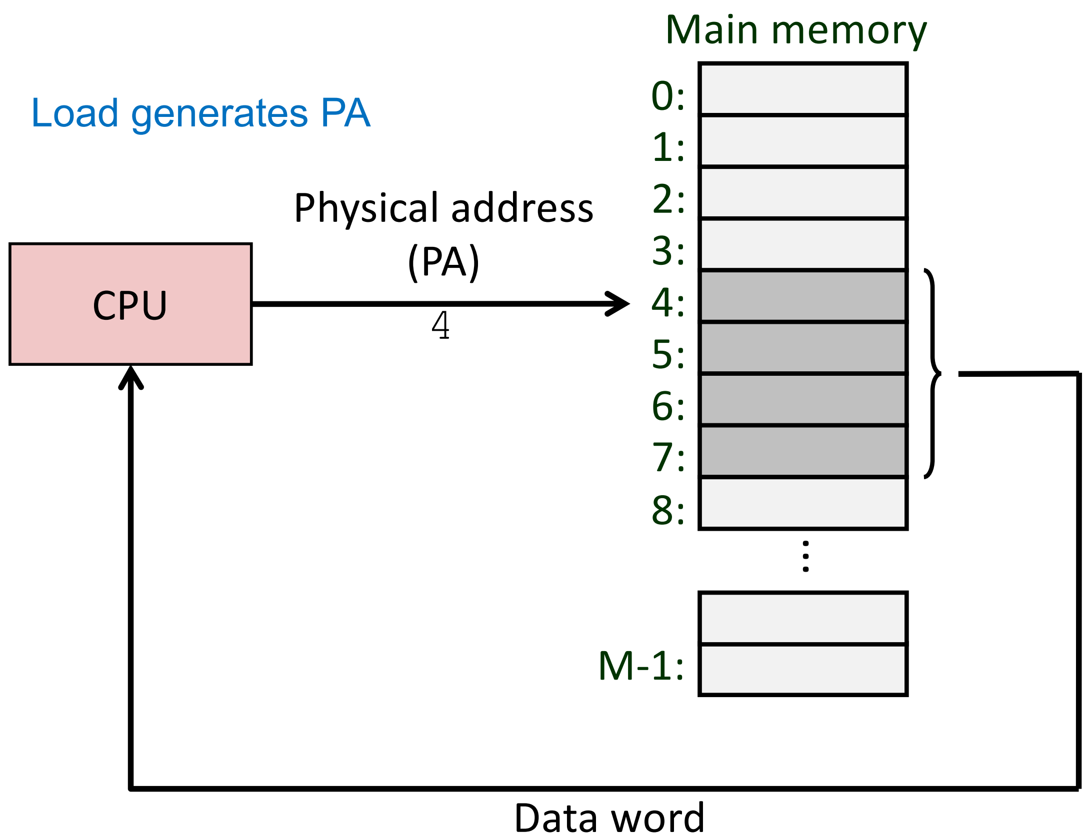
Topics

- Virtual Memory (material for the Final Exam)
  - Introduction
  - Address spaces
  - VM as a tool for caching
  - VM as a tool for memory management
  - VM as a tool for memory protection
  - Address translation (serves as review of cache)

Virtual Memory (VM)

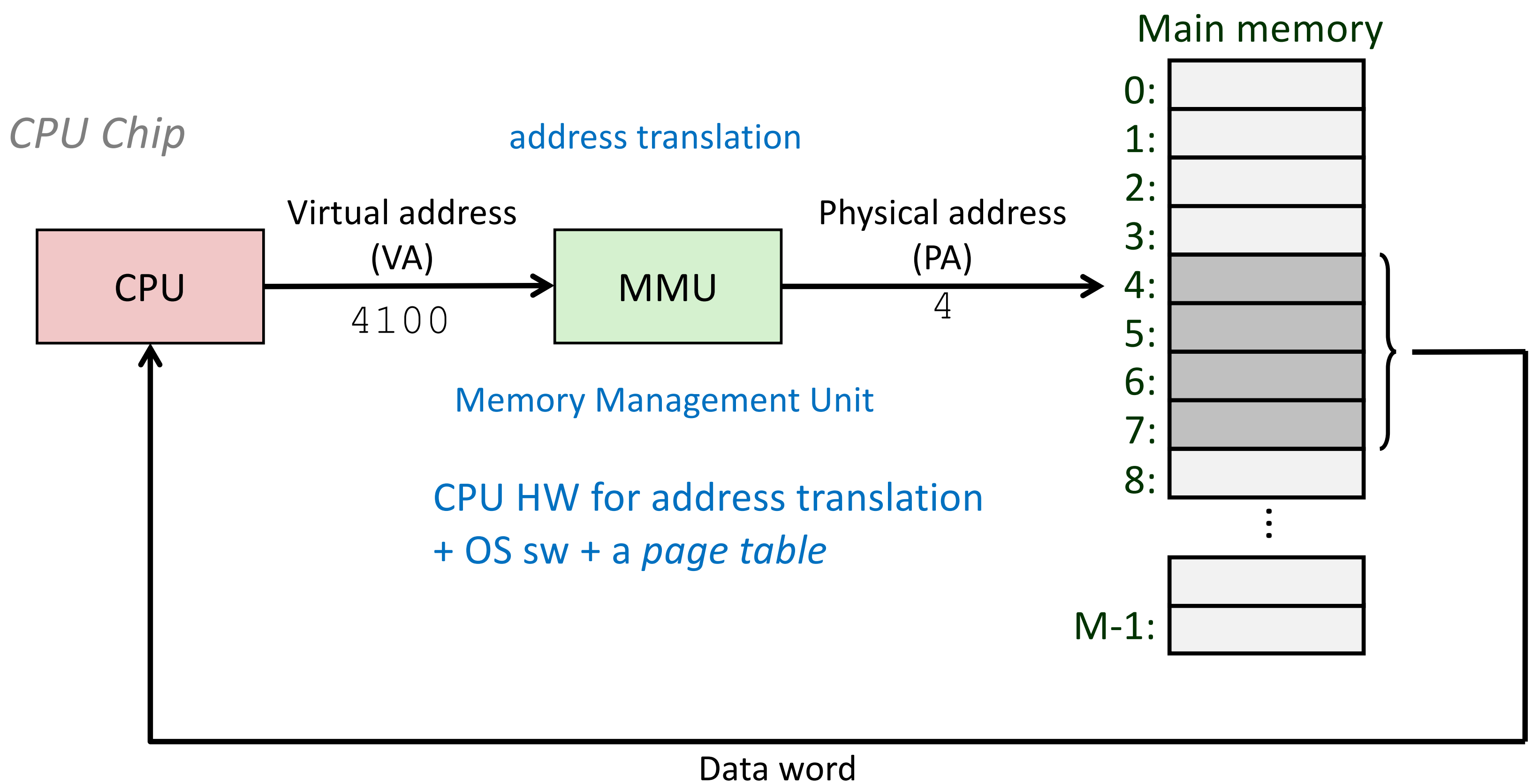
- One of the great ideas of computer systems
- Abstraction of main memory
- Designed to manage memory more efficiently and with fewer errors
- VM provides:
  - cache for address space stored in disk
  - to each process a uniform address space
  - protects the address space of each process from corruption by other processes.

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, laptops, and smart phones
- 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 … }

- Virtual address space: Set of $N = 2^n$ virtual addresses

  {0, 1, 2, 3, …, N-1}

  - n-bit address spaces, typical: 32-bit and 64-bit

- Physical address space: Set of $M = 2^m$ physical addresses

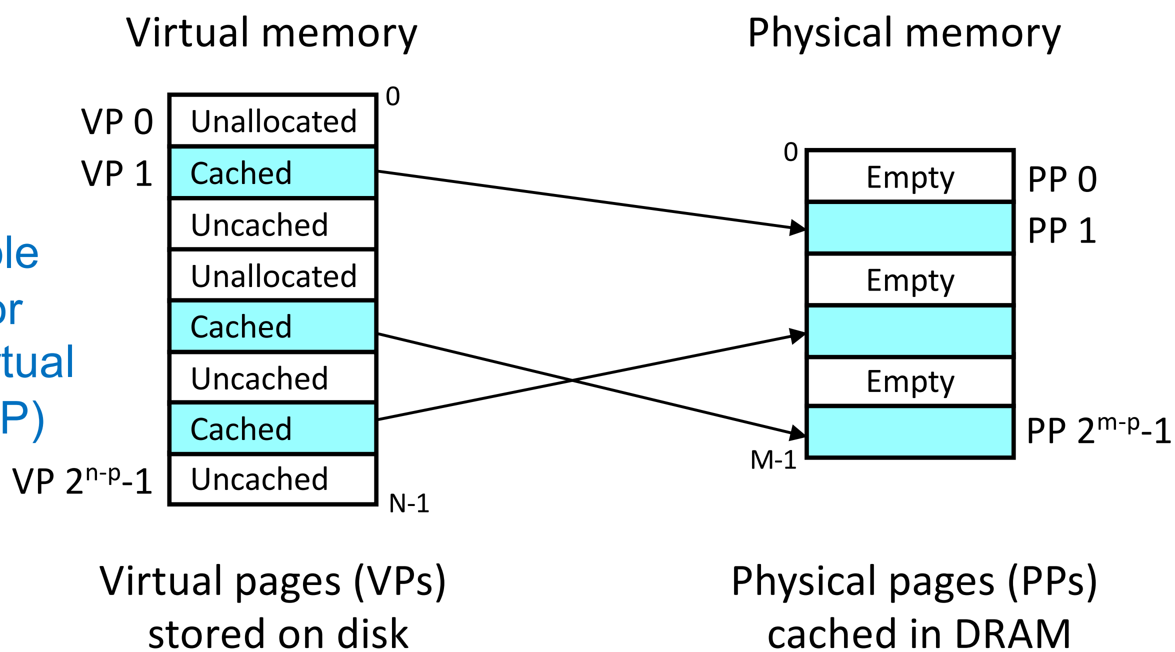
  {0, 1, 2, 3, …, M-1}

Why Virtual Memory (VM)?

- Uses main memory efficiently
  - Use *DRAM as a cache* for parts of a virtual address space
- Simplifies memory management
  - Each process gets the *same uniform linear address space*
- *Isolates address spaces*
  - One process can’t interfere with another’s memory
  - User program cannot access privileged kernel information and code

VM as a Tool for Caching

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



3 possible states for each Virtual Page (VP)

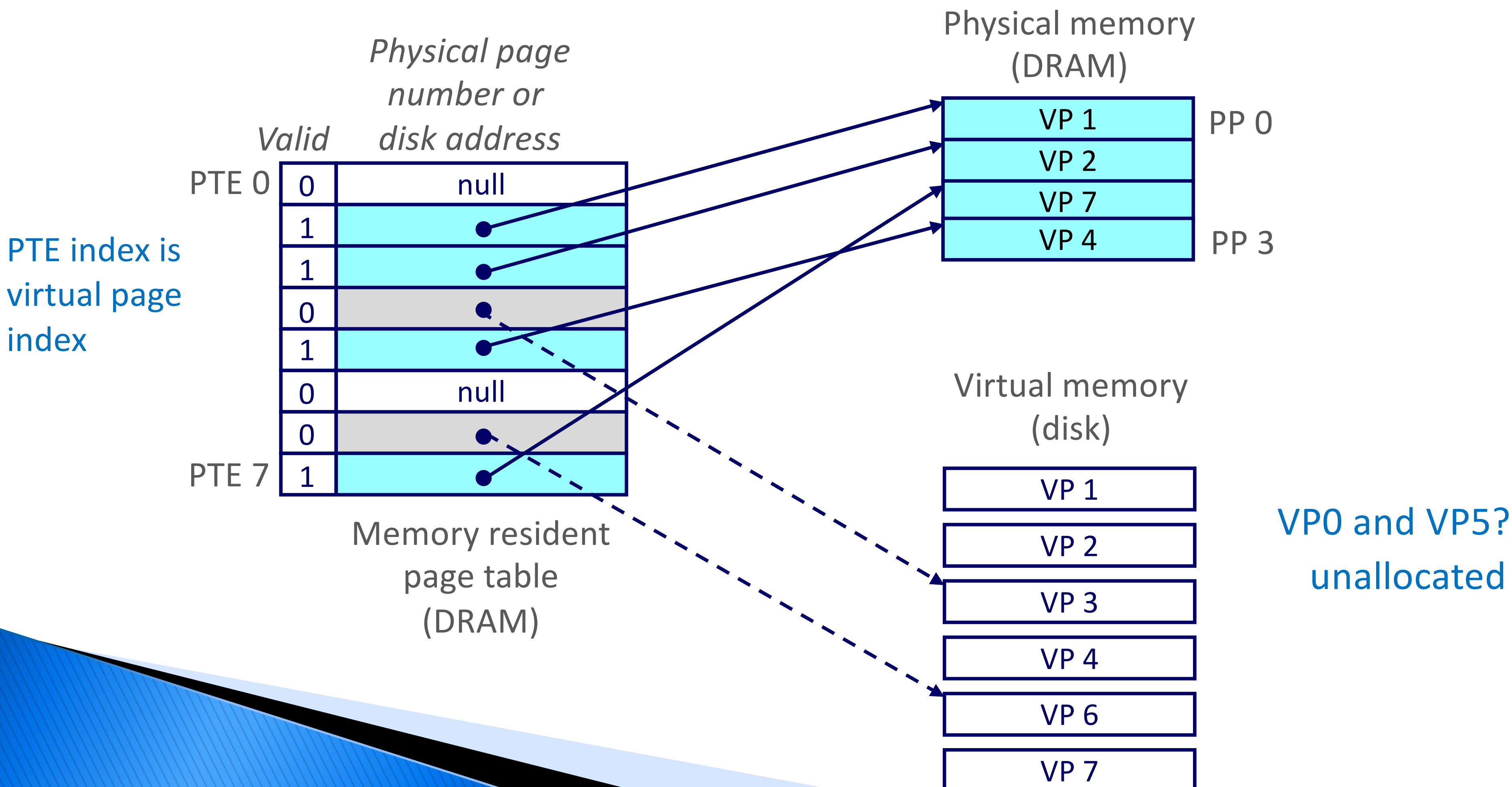
Figure 9.3

“DRAM Cache” Organization (sec 9.3.1)

- “DRAM cache” (main memory) organization driven by the enormous miss penalty
  - DRAM is about ***10x*** slower than SRAM (L1, L2, L3 level cache memories)
  - Disk is about ***10,000x*** slower than DRAM (main memory)
- Consequences
  - Large page (block) size: typically 4 KB, sometimes 4 MB
  - Fully associative (Recall Chap 6, p. 626, single set holds all the lines)
    - Any VP(virtual page) can be placed in any PP (physical page)
    - Requires a “large” mapping function – different from cache memories
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
  - Write-back (later) rather than write-through

Data Structure: Page Table (enables VM) (sec. 9.3.2)

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



Exercise

- Practice problem 9.2 (review terminology + sizes)

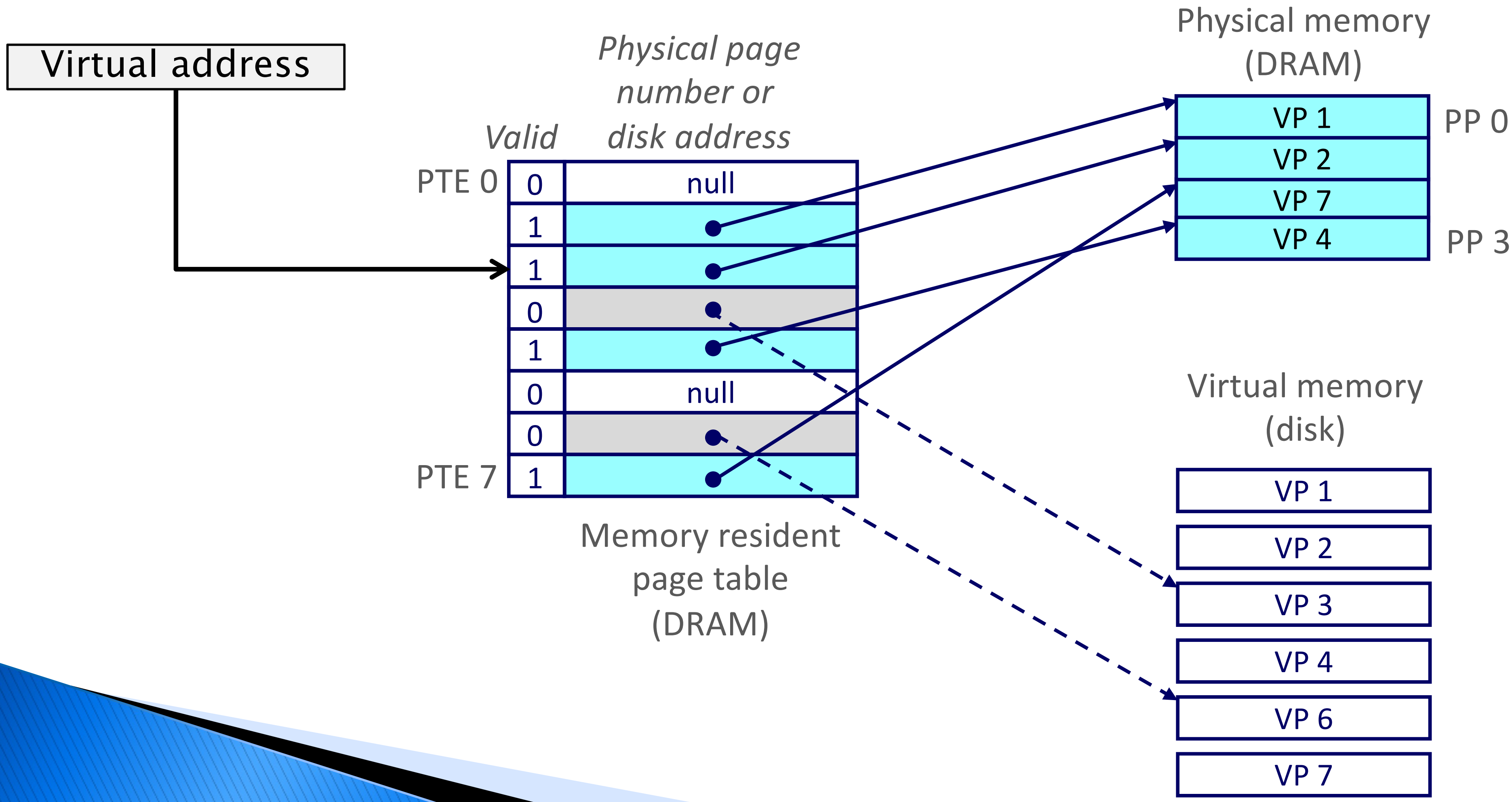
2^n size of virtual memory

n	P (size of page) = 2^p	number of PTE's			
16	4K	$2^{16}/2^{12} = 2^4$	=	16	entries
16	8K	$2^{16}/2^{13} = 2^3$	=	8	"
32	4K	$2^{32}/2^{12} = 2^{20}$	=	1M	"
32	8K	$2^{32}/2^{13} = 2^{19}$	=	512K	"

K = 2^{10} M = 2^{20} G = 2^{30} T = 2^{40} Peta = 2^{50} Exa = 2^{60}

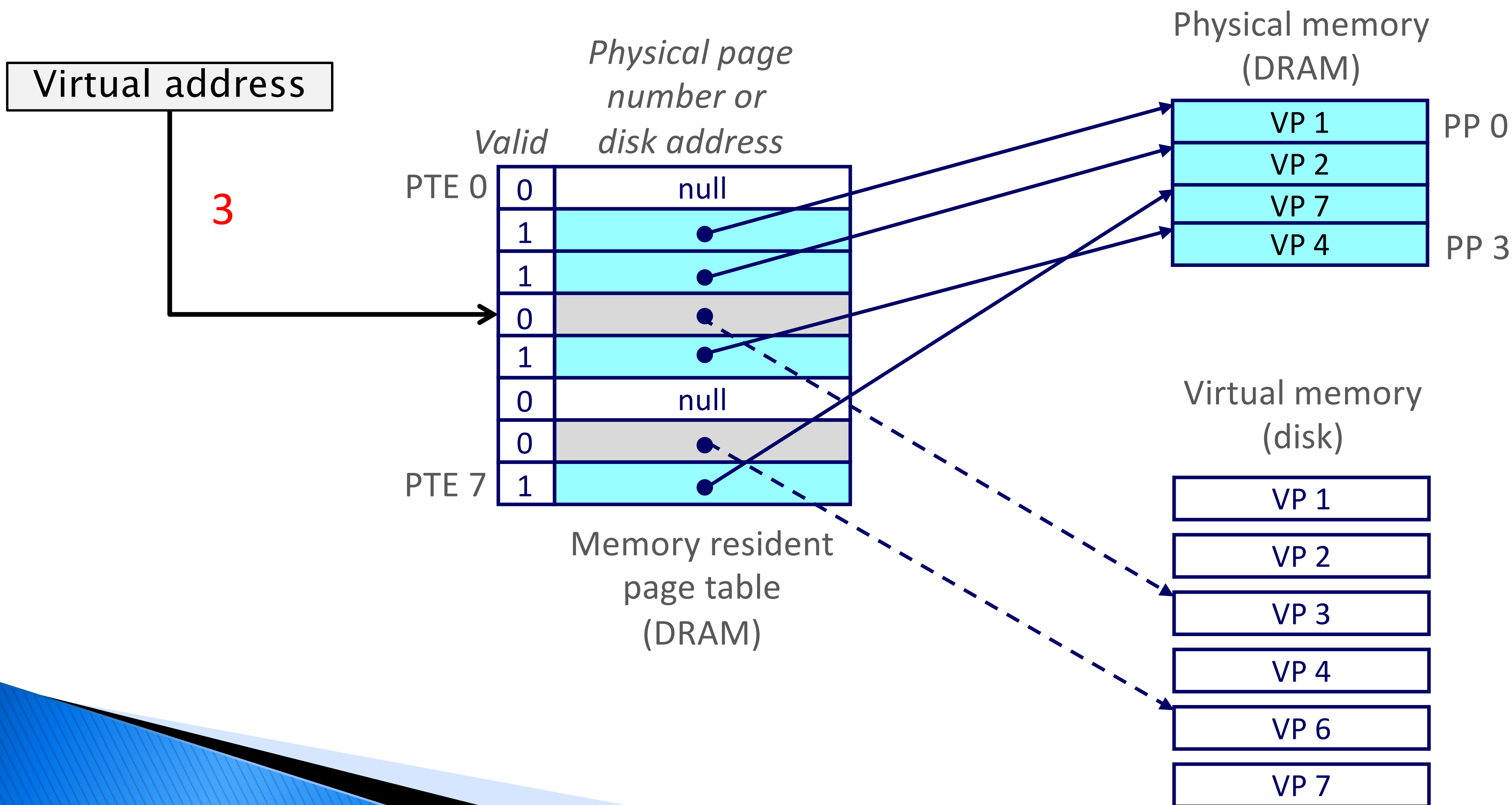
Page Hit (sec. 9.3.3)

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



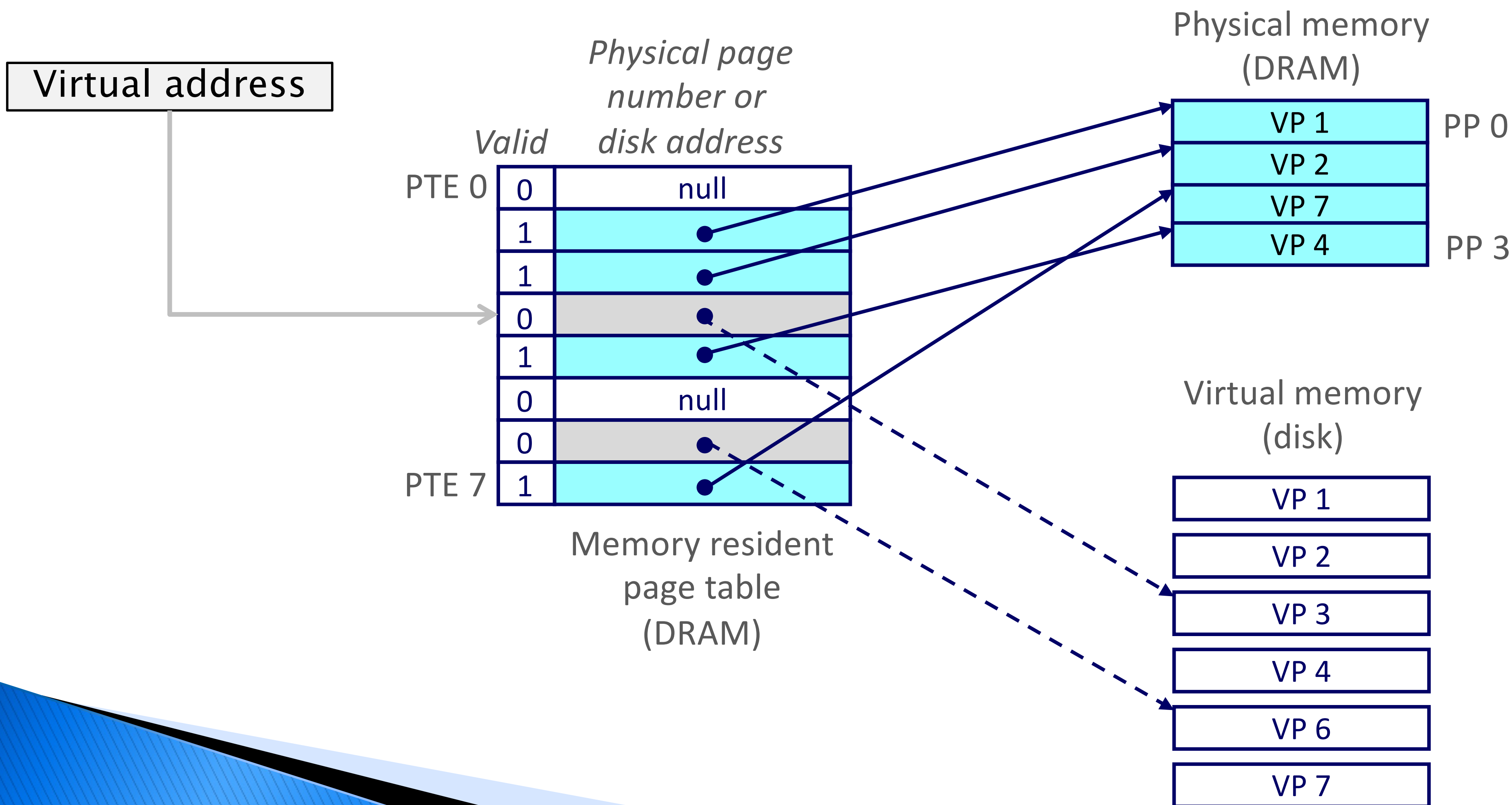
Page Fault (sec. 9.3.4)

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



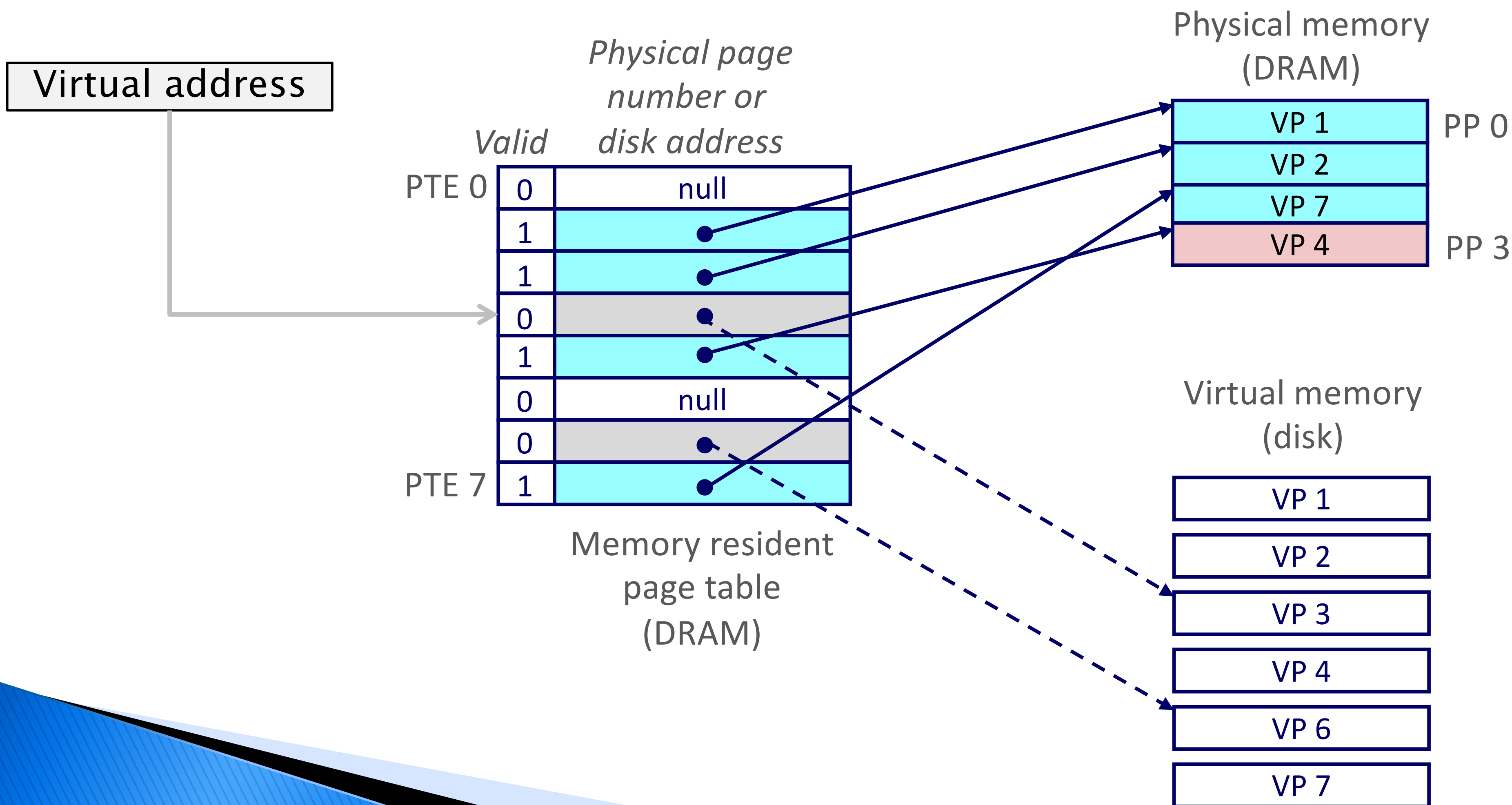
Handling Page Fault

- Page miss causes page fault (an exception)



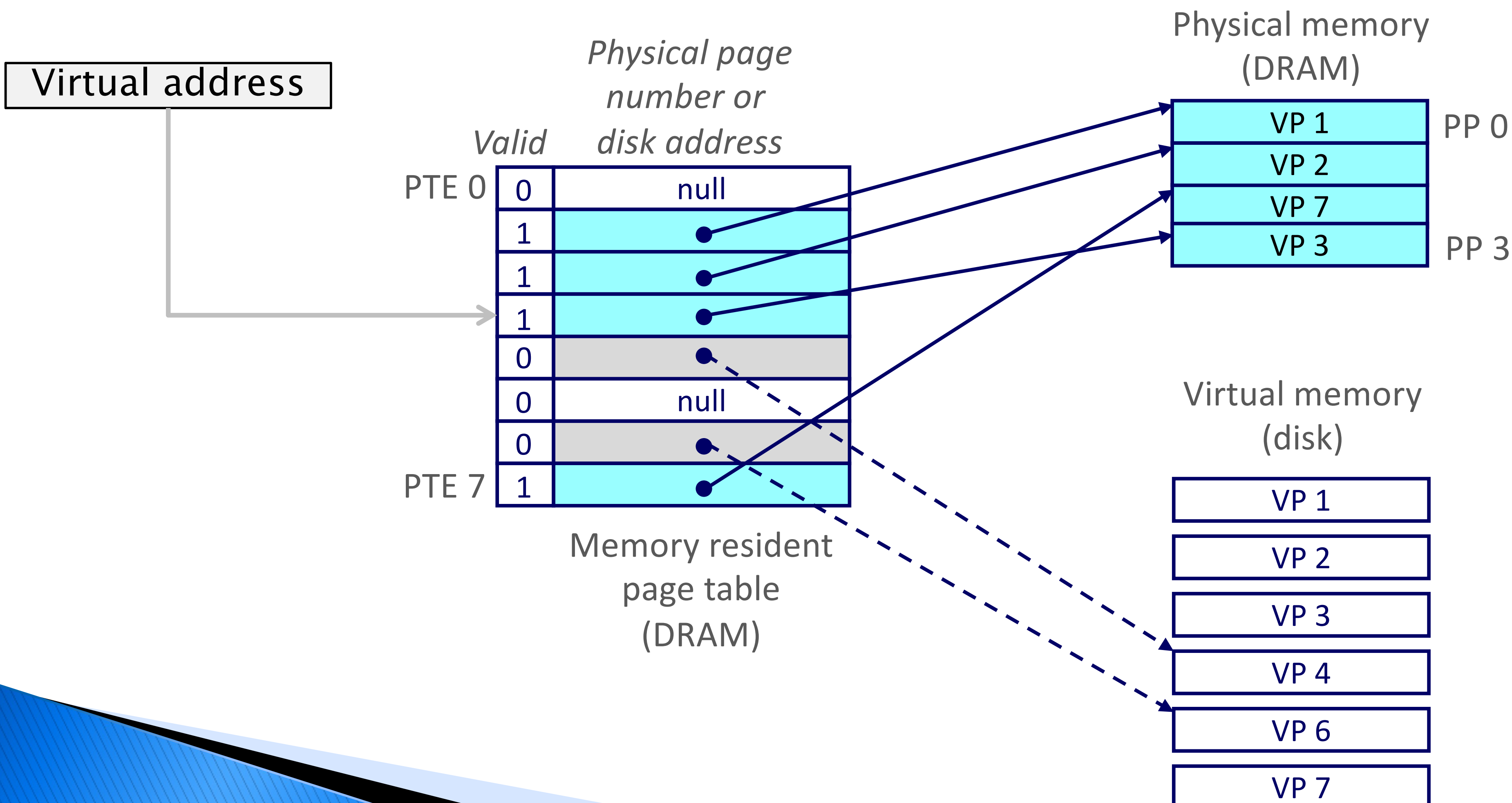
Handling Page Fault

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



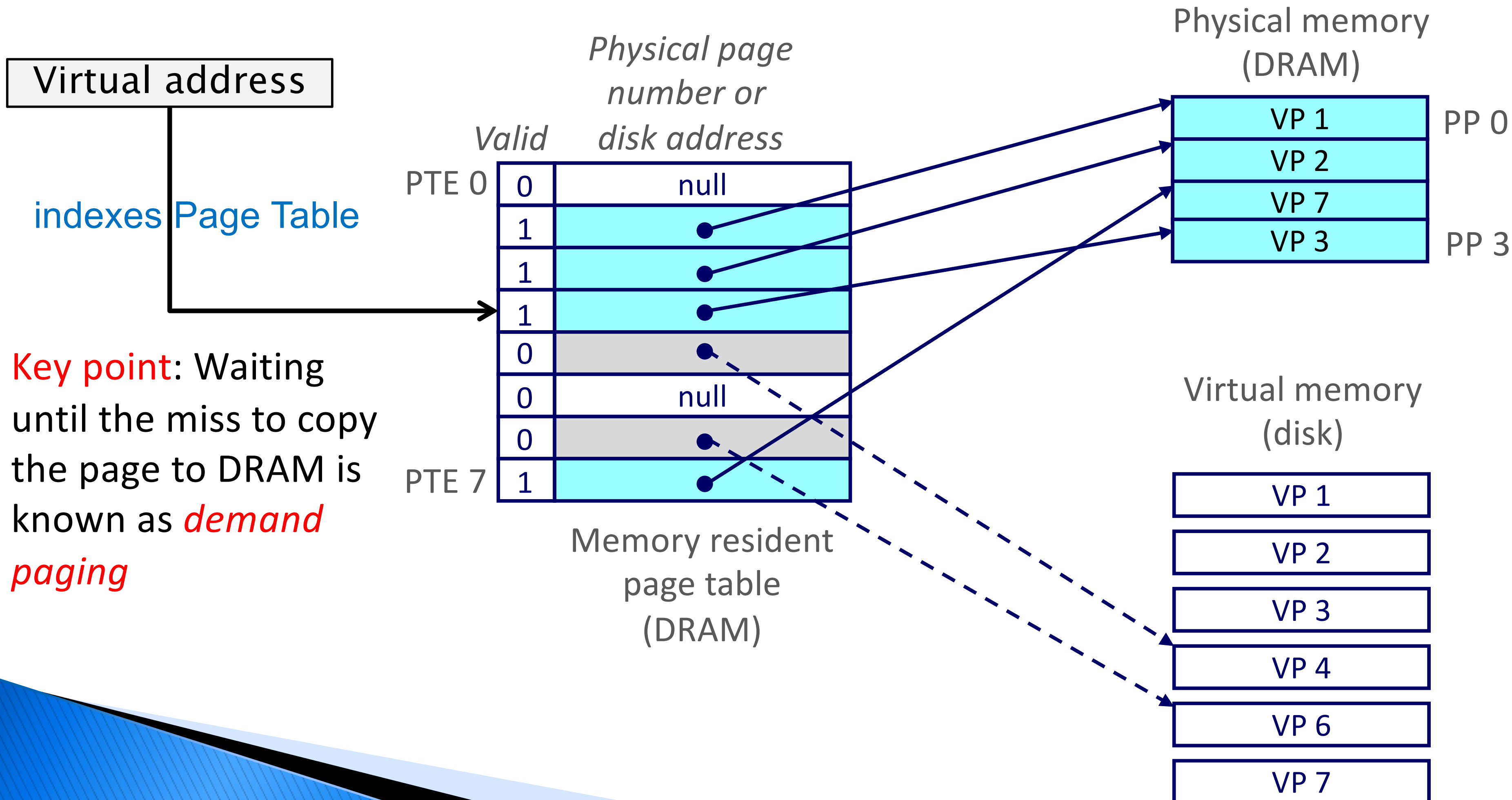
Handling Page Fault

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



Handling Page Fault

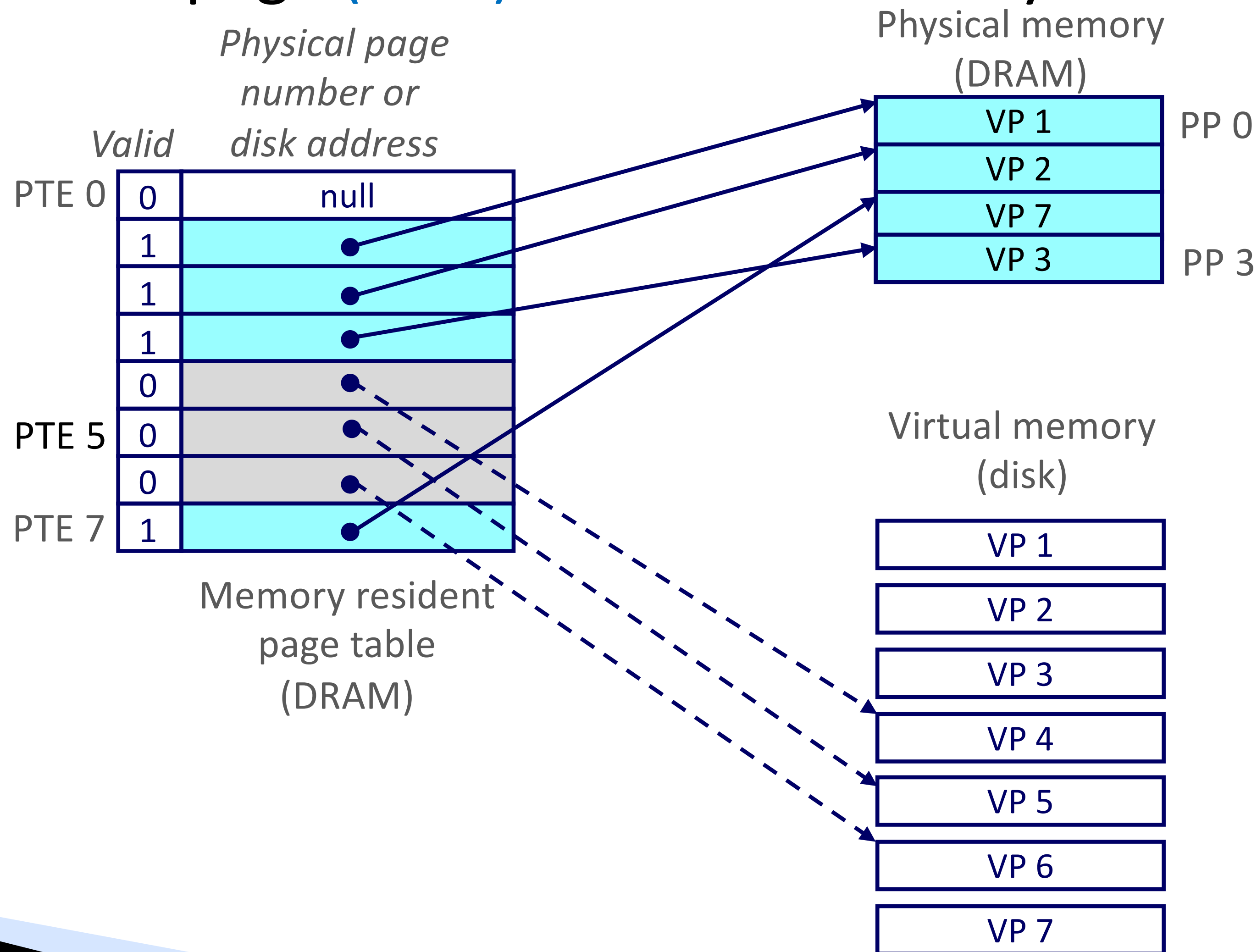
- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4) (write back!)
- Offending instruction is restarted: now it is a page hit!



Key point: Waiting until the miss to copy the page to DRAM is known as *demand paging*

Allocating Pages (sec. 9.3.5 + fig. 9.8)

- Allocating a new page (VP 5) of virtual memory.



Question (addressed with diagram on the board at the beginning)

- Given what we know of the cache, what would the process be to provide a Virtual Page to a process that needs data from it?
- First, recall how was the process to get data from main memory.
- We will explain it in great detail, but first get a clear picture of what we are trying to do.
- Keep adding more details to the general picture drawn at the beginning.

Topics

- Virtual Memory
  - VM as a tool for caching (cont.)
  - VM as a tool for memory management
  - VM as a tool for memory protection
  - Address translation
  - Simple memory system example

Locality to the Rescue Again! (sec. 9.3.6)

- Virtual memory seems terribly inefficient, but it 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 and VM works well.

- 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