



CS3281 / CS5281

# Virtual Memory

CS3281 / CS5281

Fall 2025

*\*Some lecture slides borrowed and adapted from CMU's  
"Computer Systems: A Programmer's Perspective"*



Tel (615) 343-7472 | Fax (615) 343-7440  
1025 16th Avenue South Nashville, TN 37212  
[www.isis.vanderbilt.edu](http://www.isis.vanderbilt.edu)



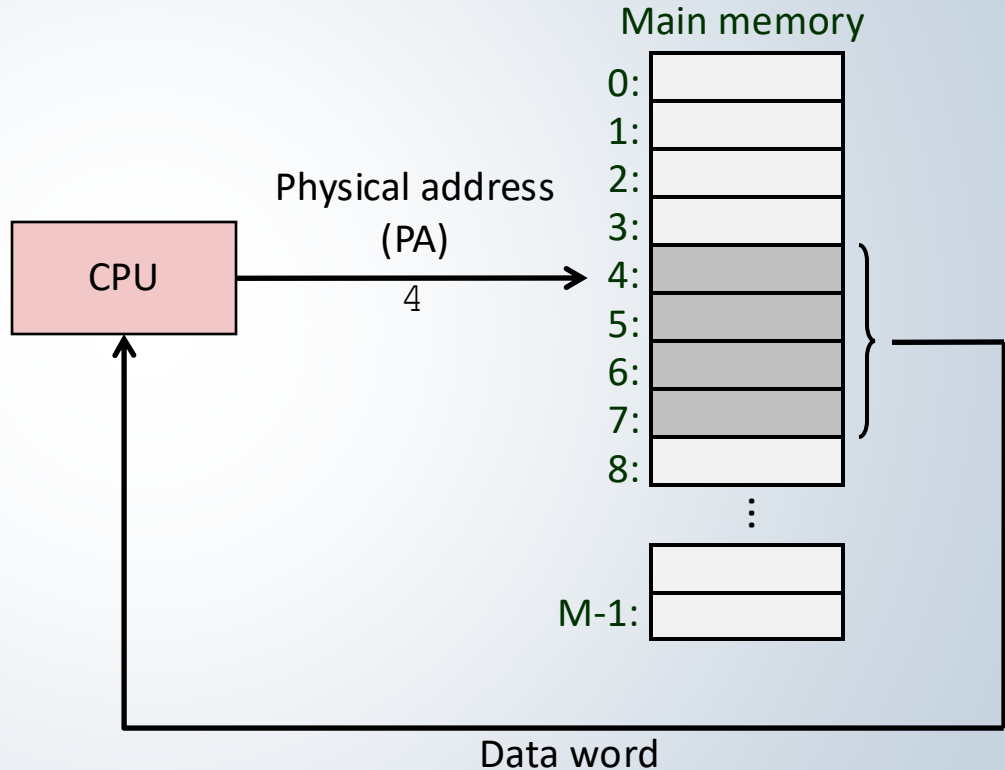
# Today

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



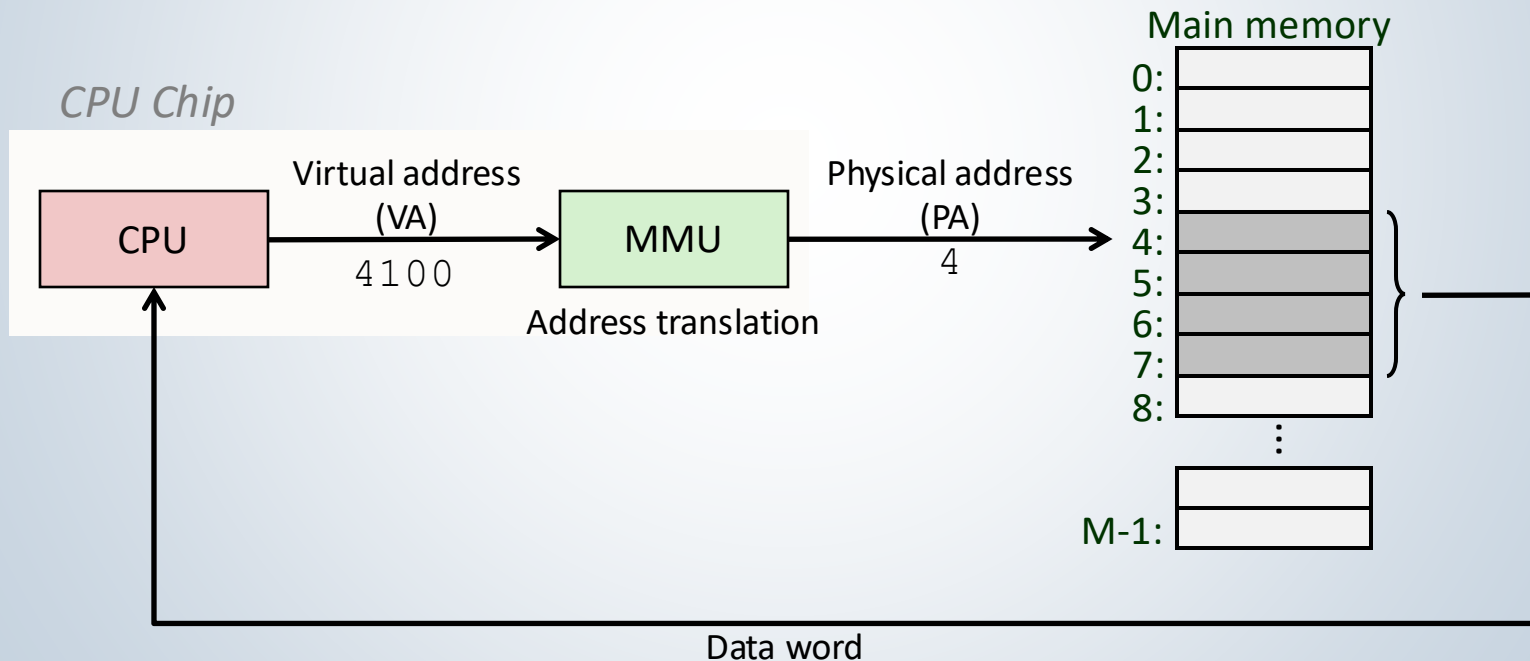
# A System Using Physical Addressing

- 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, 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:  
 $\{0, 1, 2, 3 \dots \}$
- **Virtual address space:** Set of  $N = 2^n$  virtual addresses  
 $\{0, 1, 2, 3, \dots, N-1\}$
- **Physical address space:** Set of  $M = 2^m$  physical addresses  
 $\{0, 1, 2, 3, \dots, M-1\}$

# Why Virtual Memory (VM)?

- Simplifies memory management
  - Each process gets the same linear address space
- Isolates address spaces
  - One process can't interfere with another's memory
- Uses main memory (RAM) efficiently
  - Use DRAM as a cache for parts of a virtual address space. A very large virtual memory is perceived by programmers when only a small physical memory is available
- Many other benefits (some discussed later)
  - Shared memory, memory deduplication, lazy allocation, etc.

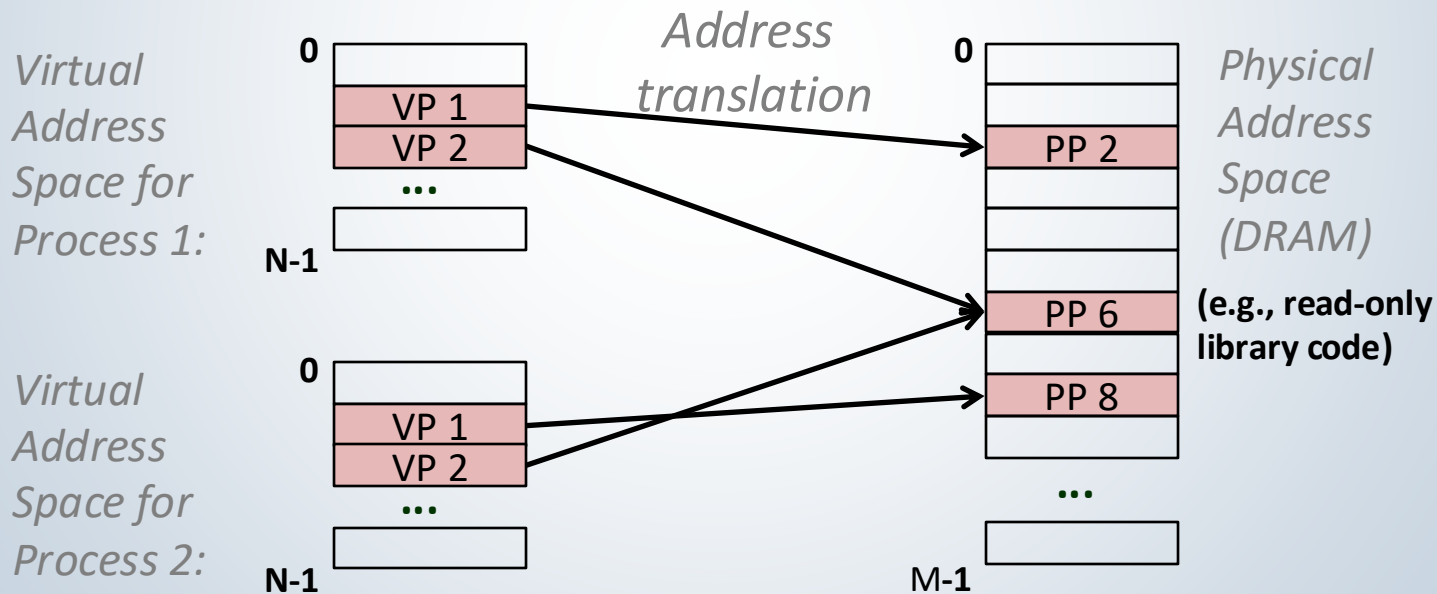
# Today

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



# VM as a Tool for Memory Management

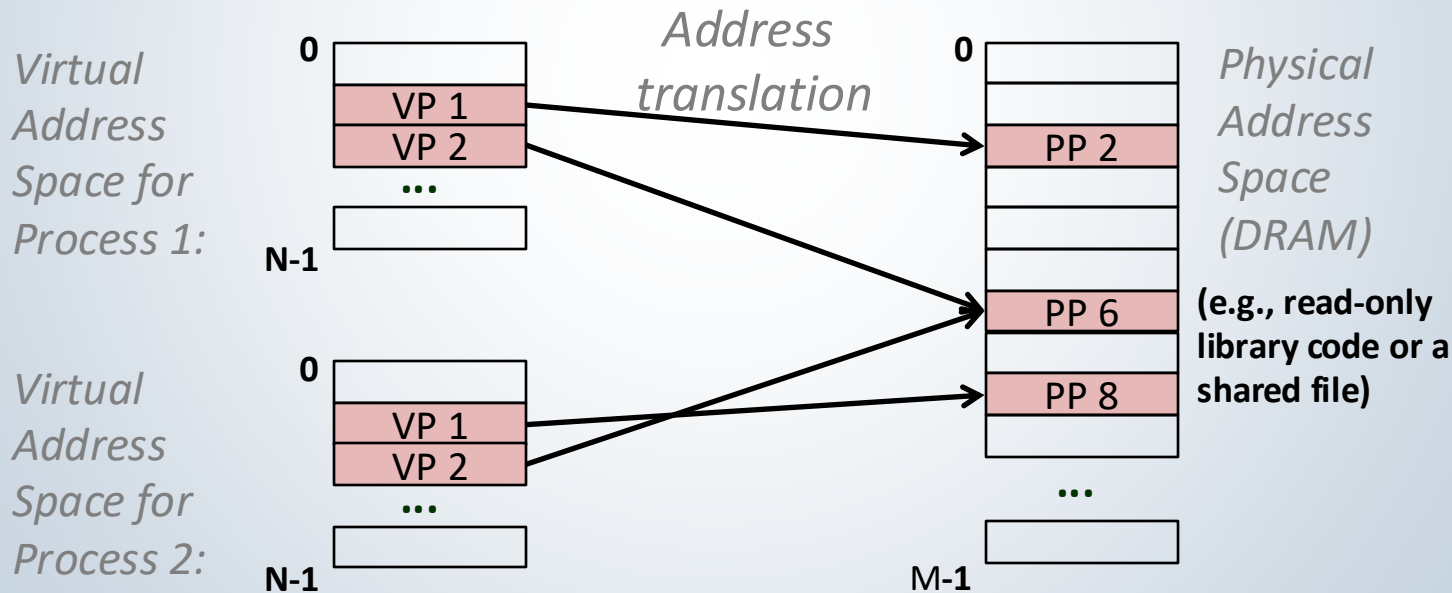
- **Key idea: each process has its own virtual address space**
  - It can view memory as a simple linear array
  - Mapping function scatters addresses through physical memory





# VM as a Tool for Memory Management

- **Simplifying 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)



# Today

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



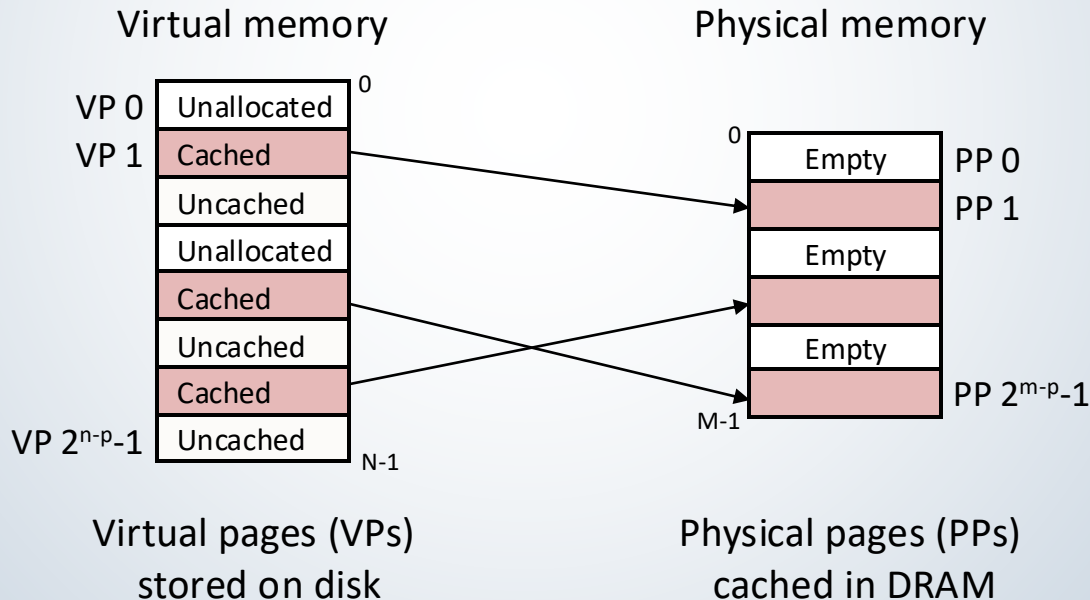
# Today

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



# VM as a Tool for Caching

- Conceptually, *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). Pages may (not) be contiguous in the physical memory



# DRAM Cache Organization

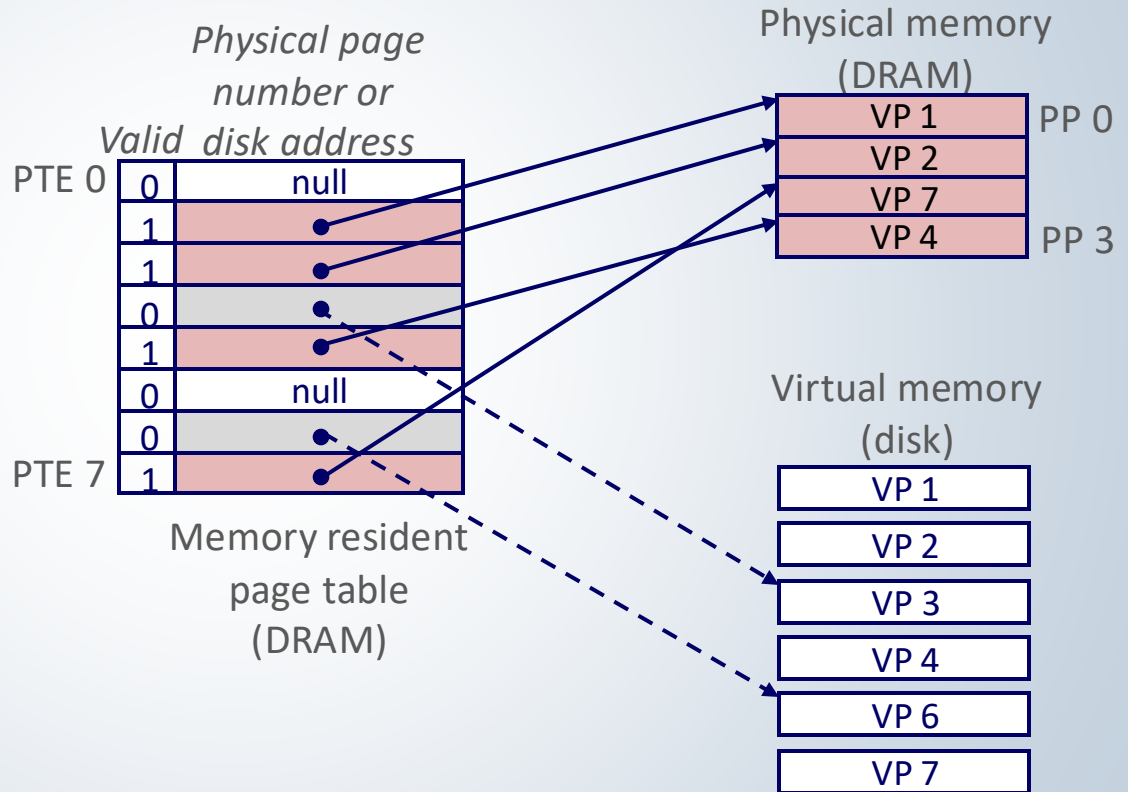
- DRAM cache organization driven by the enormous miss penalty
  - DRAM is about **10x** slower than SRAM (cache)
  - Disk is about **10,000x** slower than DRAM
- Consequences
  - Large page (block) size: typically 4 KB (Huge pages are 2MB – 1GB.)
  - Only a subset of virtual pages are stored in the main memory
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware

# Internal Fragmentation

- Depending on a page size, a program size may not be a multiple of the number of pages. Thus, the last page is partially filled. This loss of usable memory is known as internal fragmentation
- To avoid accessing the unused fragment of the last page, the logical addresses are compared to the program size. A logical address is rejected if it is larger than the program size

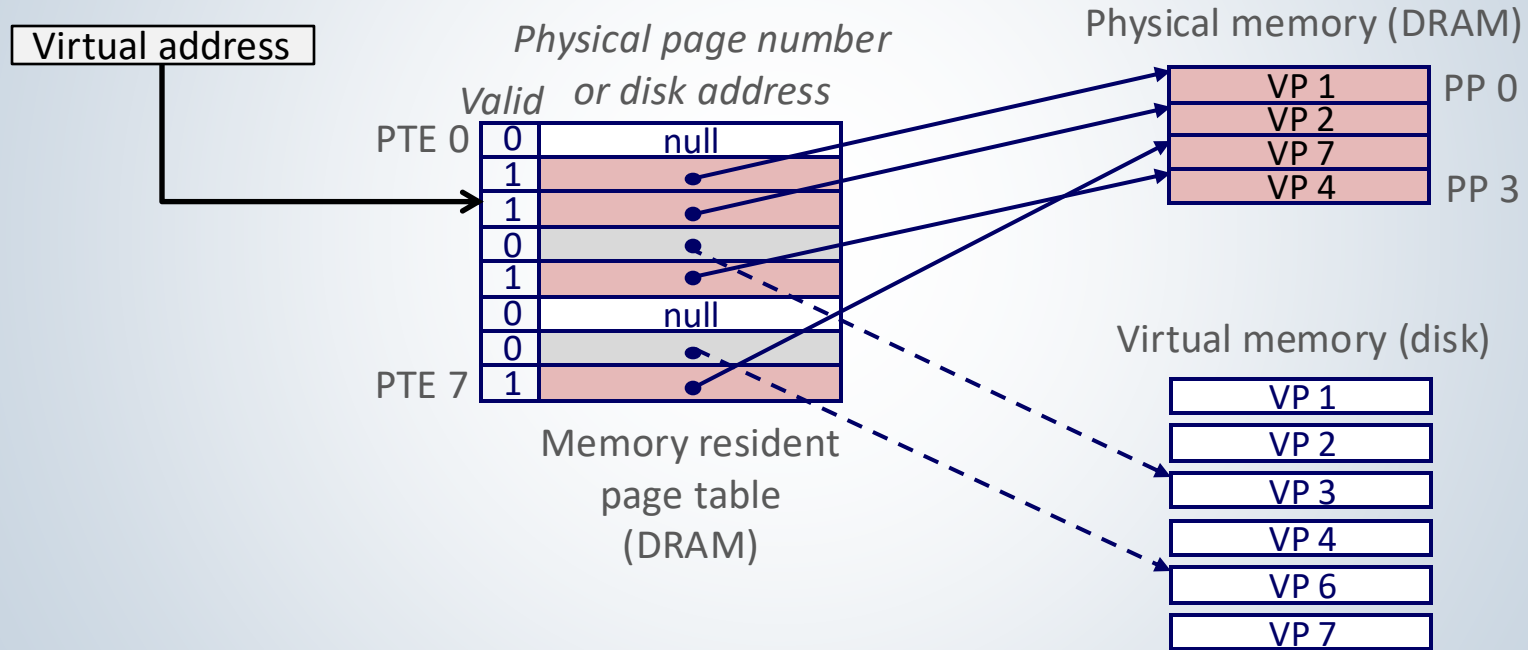
# Enabling Data Structure: Page Table

- 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

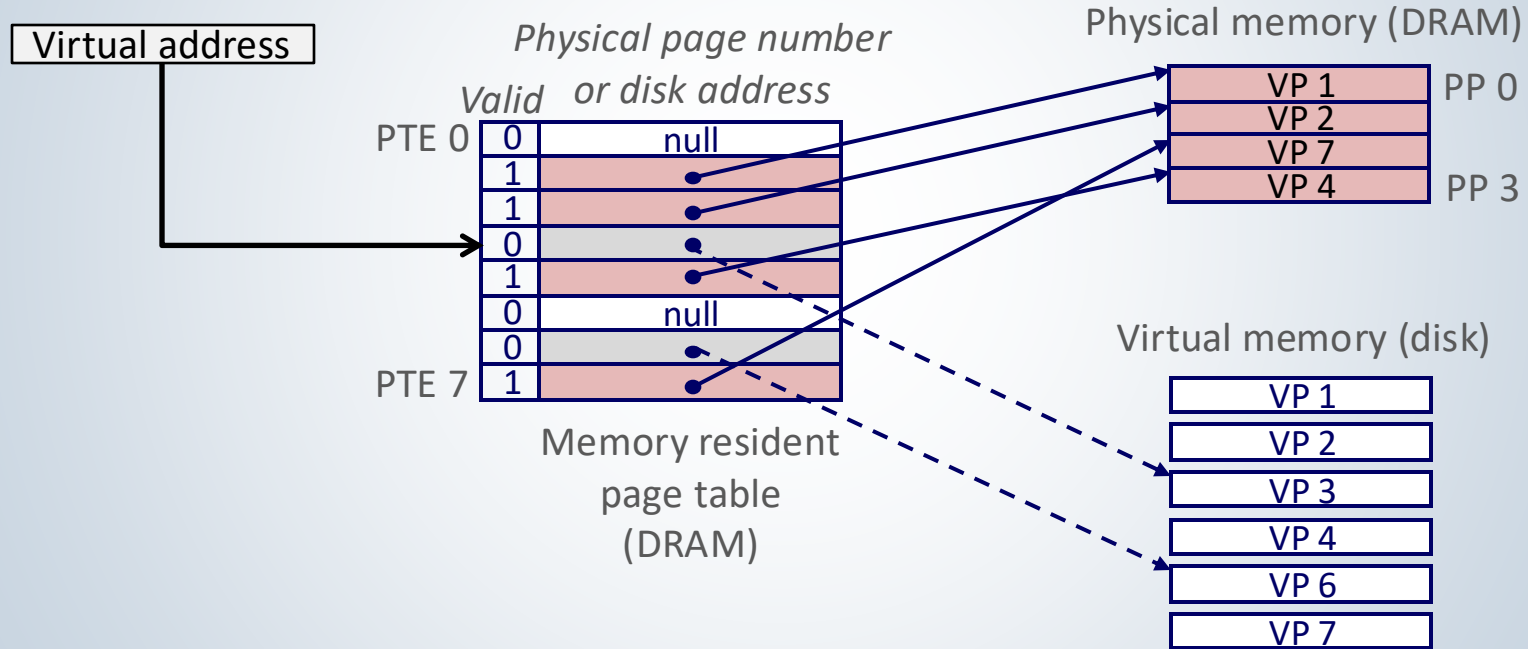
- *Page hit*: reference to VM word that is in physical memory (DRAM cache hit)





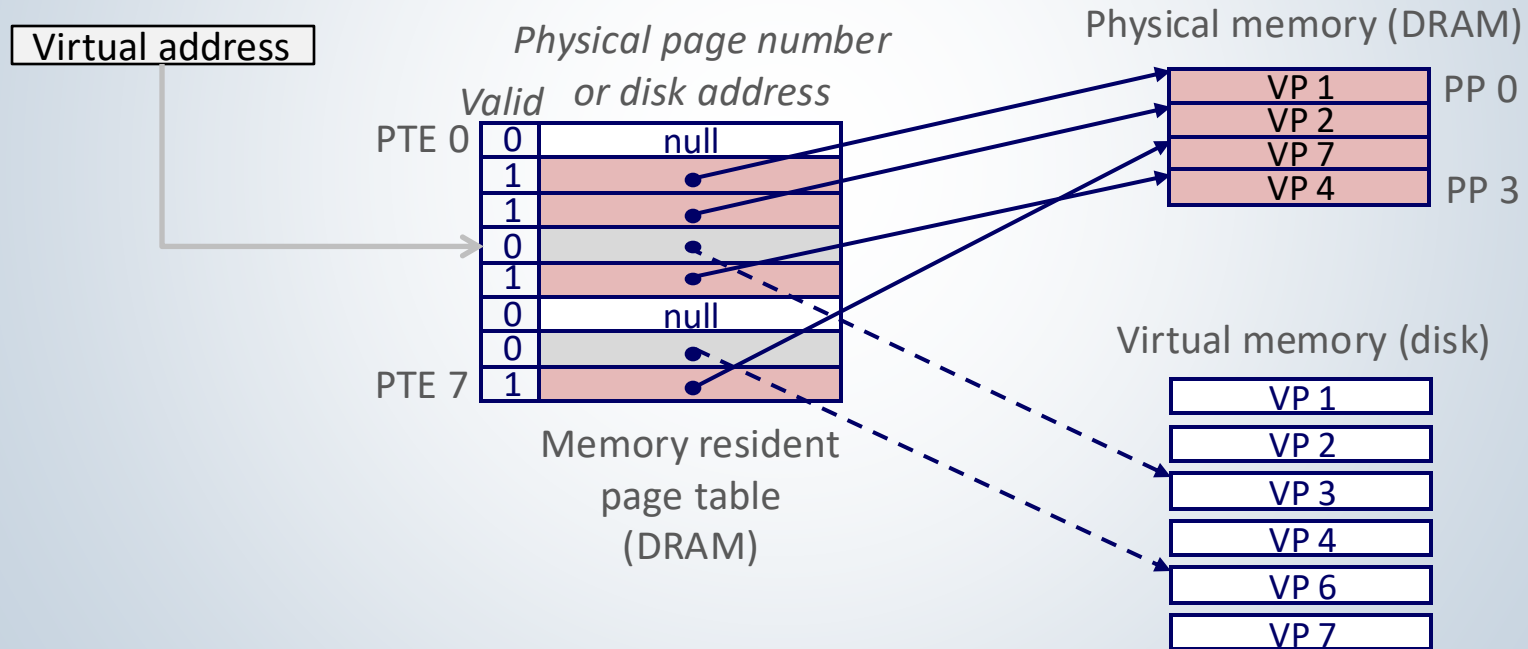
# Page Fault

- Page fault:** reference to VM word not in physical memory (DRAM cache miss). The VM word can be instruction or data



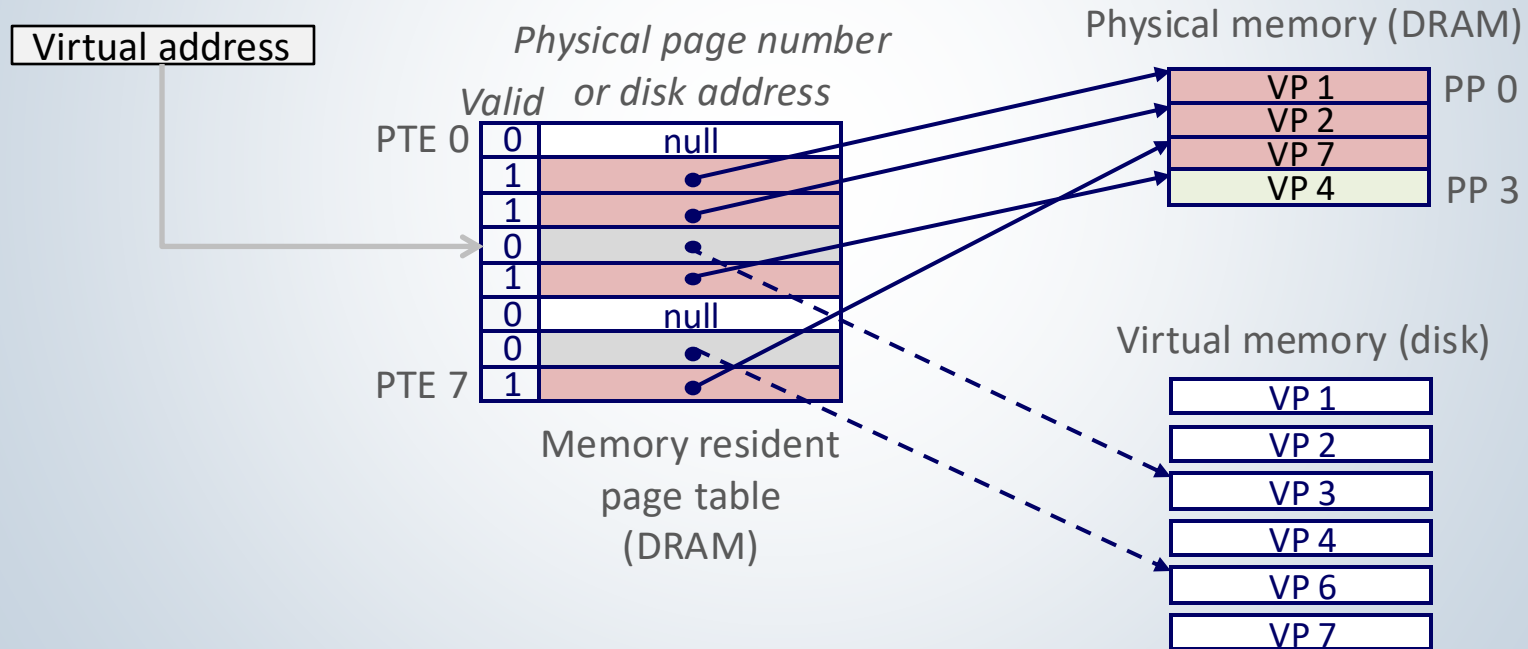
# Handling Page Fault

- Page miss causes page fault (an exception)



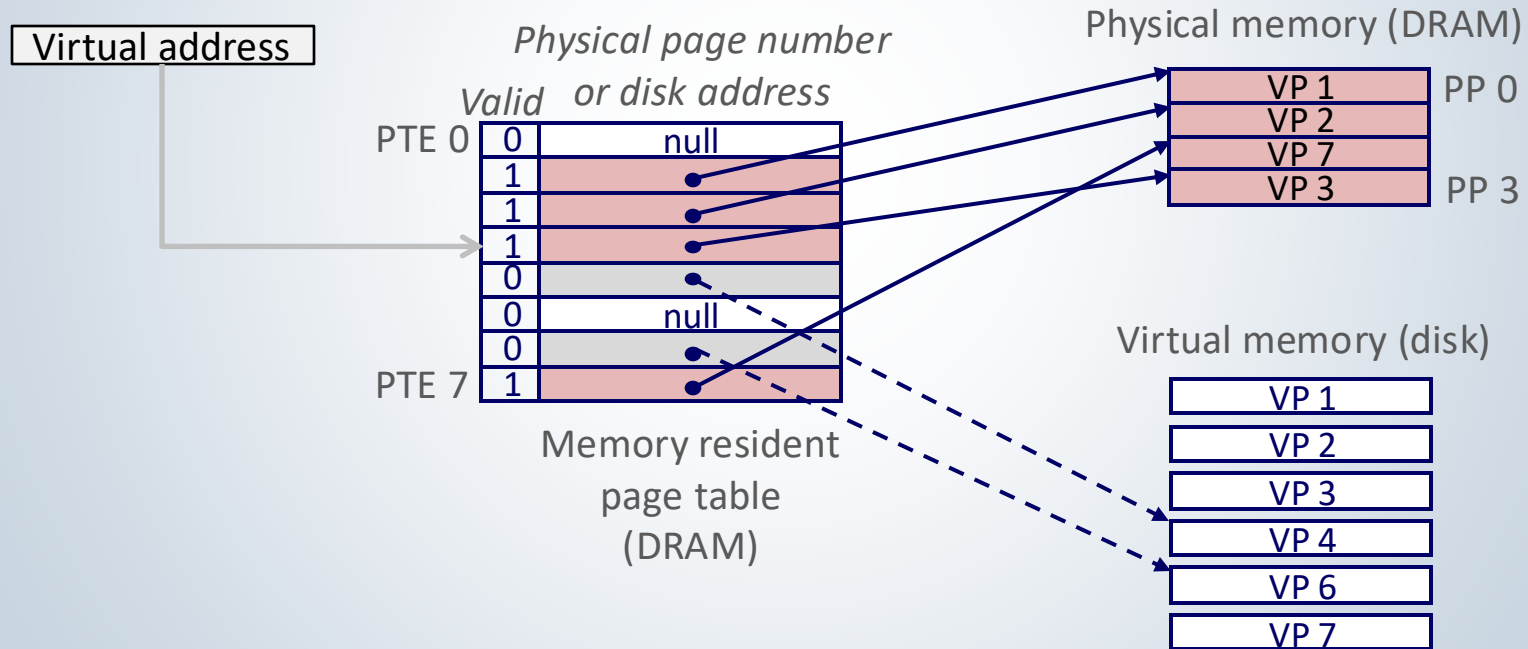
# Handling Page Fault

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



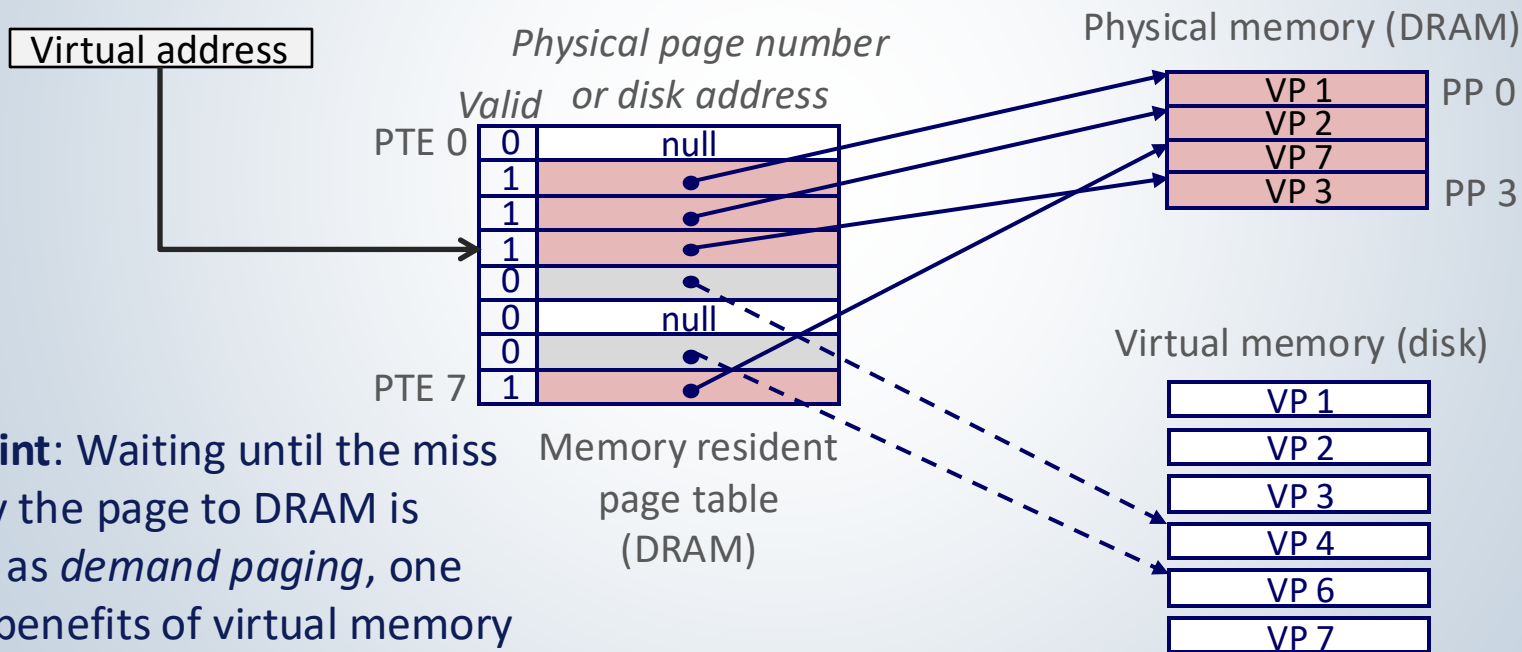
# Handling Page Fault

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

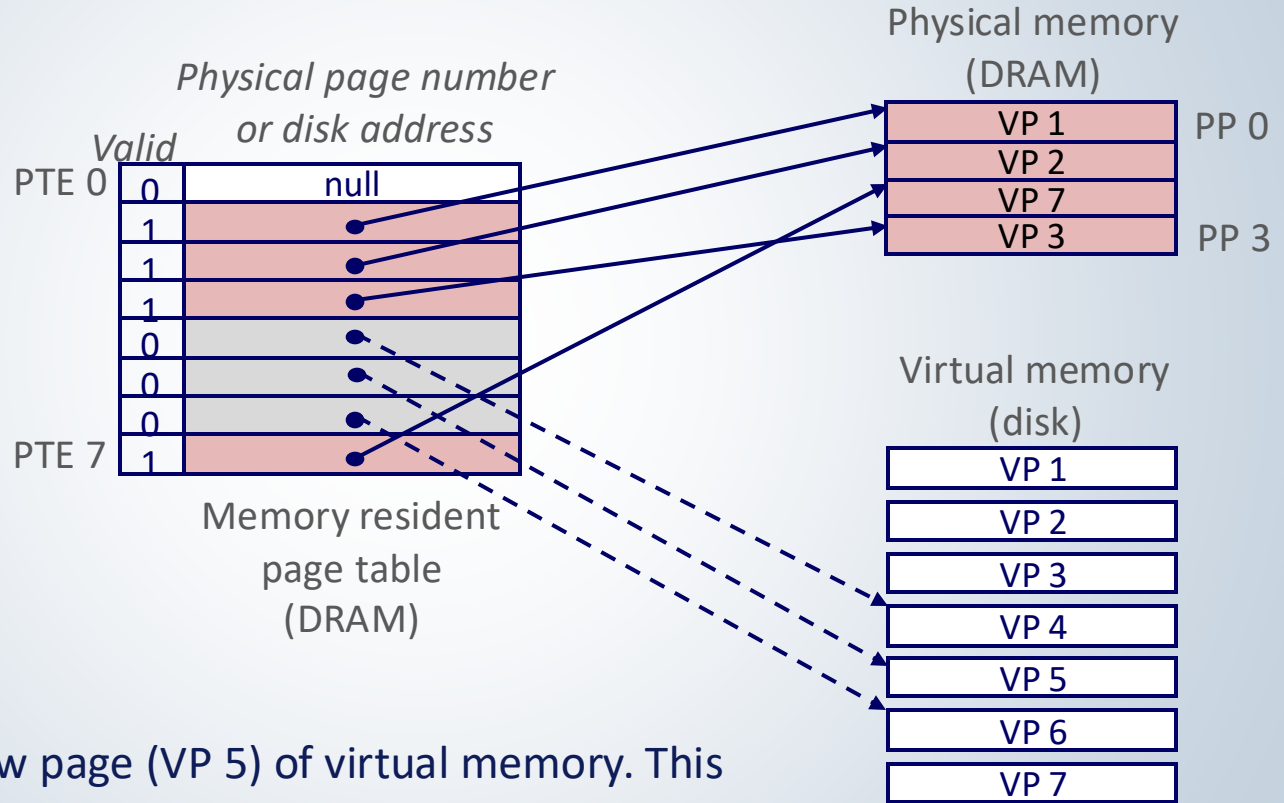


# Handling Page Fault

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- Offending (faulting) instruction is restarted: page hit! If the page fault is caused by memory miss, the data operand is fetched from memory



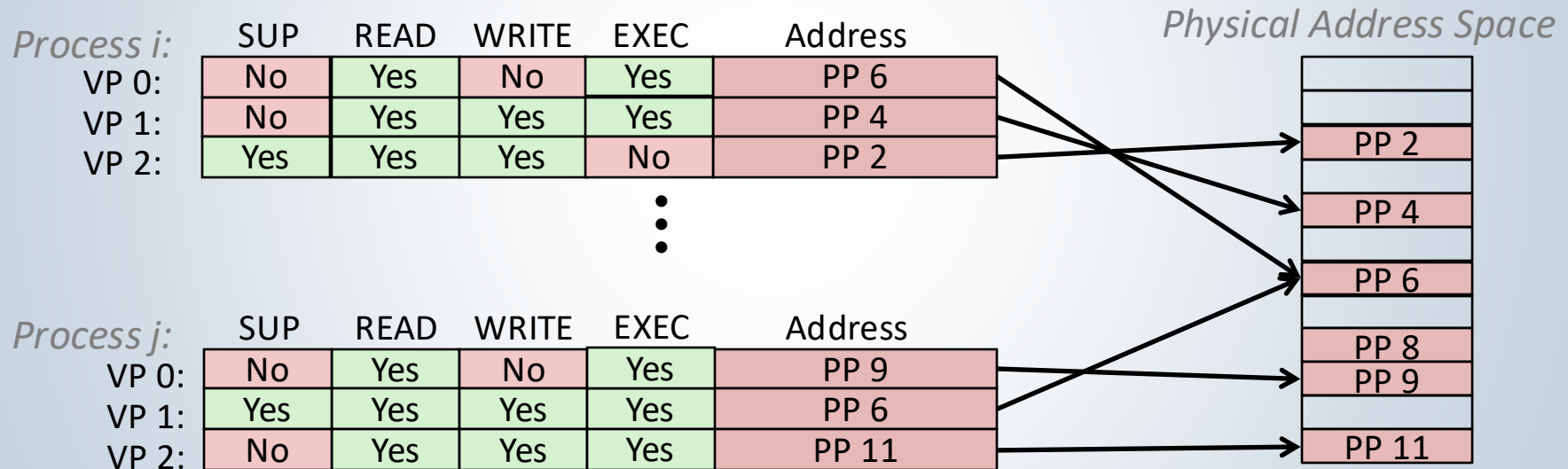
# Allocating Pages



malloc() allocates a new page (VP 5) of virtual memory. This page stays in disk until it is used

# VM as a Tool for Memory Protection

- Extend PTEs with permission bits
- MMU checks these bits on each access



# Today

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



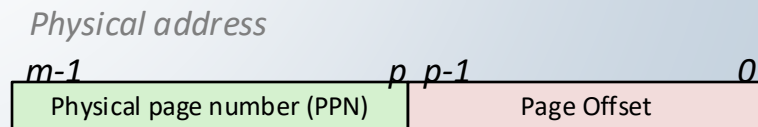
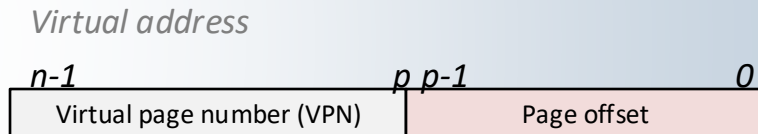


# VM Address Translation

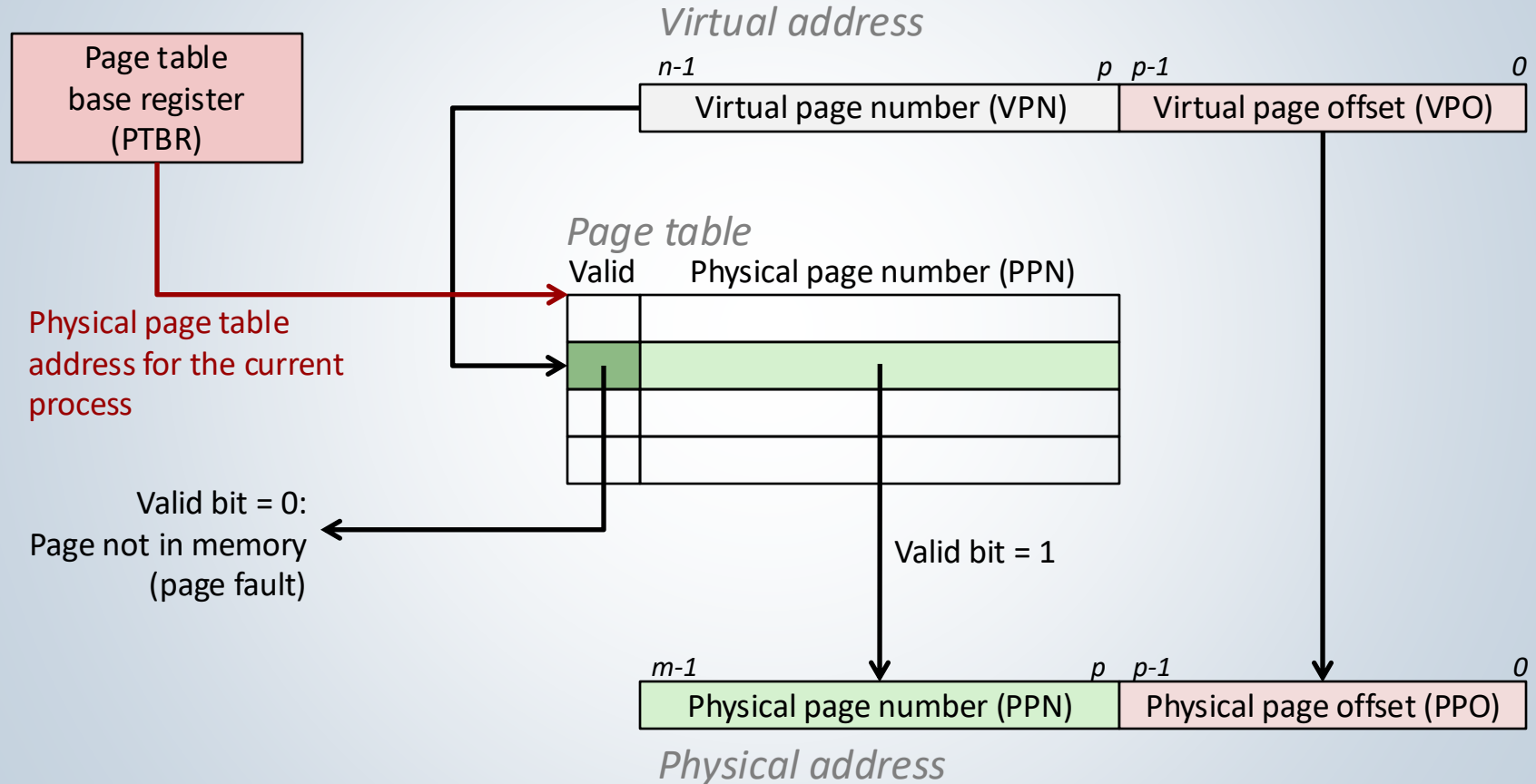
- Virtual Address Space
  - $V = \{0, 1, \dots, N-1\}$
- Physical Address Space
  - $P = \{0, 1, \dots, M-1\}$
- Address Translation
  - **$MAP: V \rightarrow P \cup \{\emptyset\}$**
  - For virtual address  $\alpha$ :
    - **$MAP(\alpha) = \alpha'$**  if data at virtual address  $\alpha$  is at physical address  $\alpha'$  in  $P$
    - **$MAP(\alpha) = \emptyset$**  if data at virtual address  $\alpha$  is not in physical memory (page fault)

# Breaking down virtual addresses

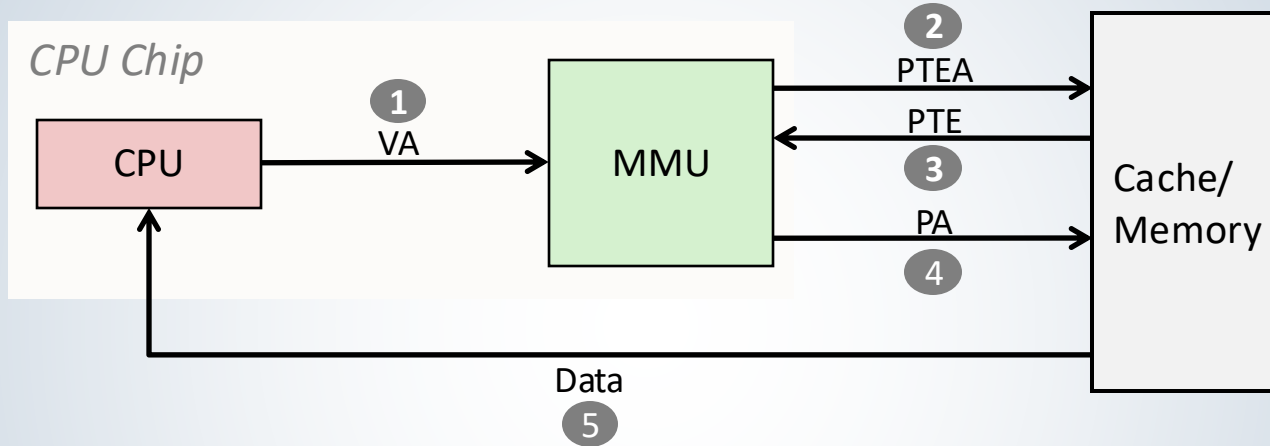
- Basic Parameters
  - $N = 2^n$  : Number of addresses in virtual address space
  - $M = 2^m$  : Number of addresses in physical address space.  $m \leq n$  (usually much less)
  - $P = 2^p$  : Page size (bytes)
- Components of the virtual address (VA)
  - Virtual page number (VPN):  $n-p$  bits
  - Page Offset:  $p$  bits
- Components of the physical address (PA)
  - Physical page number (PPN):  $m-p$  bits
  - Page Offset (same offset as VA):  $p$  bits



# Address Translation with a Page Table

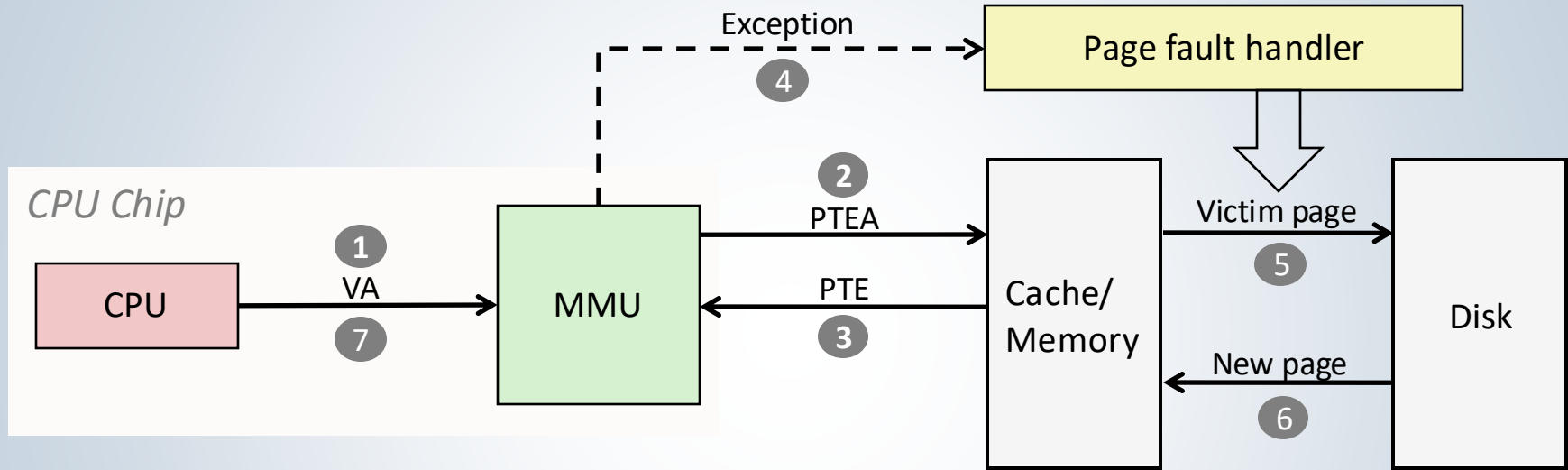


# Address Translation: Page Hit



- 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

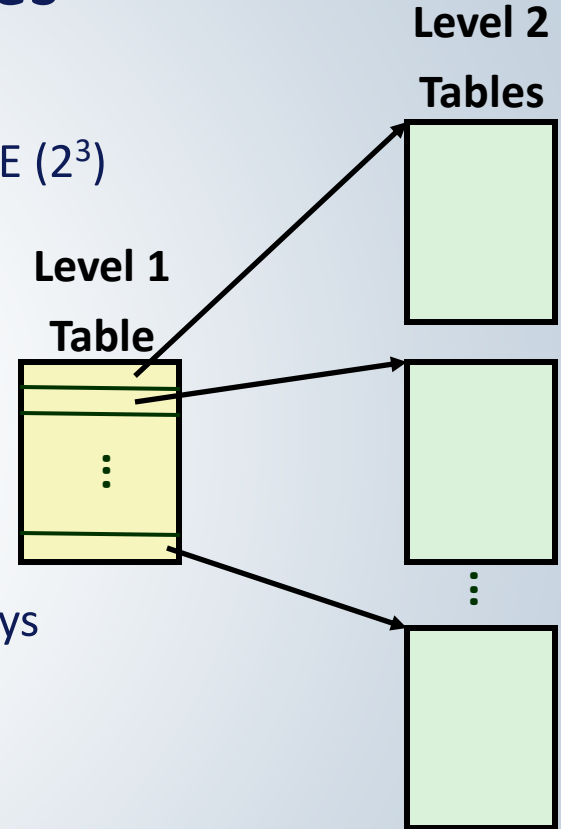
# Address Translation: Page Fault



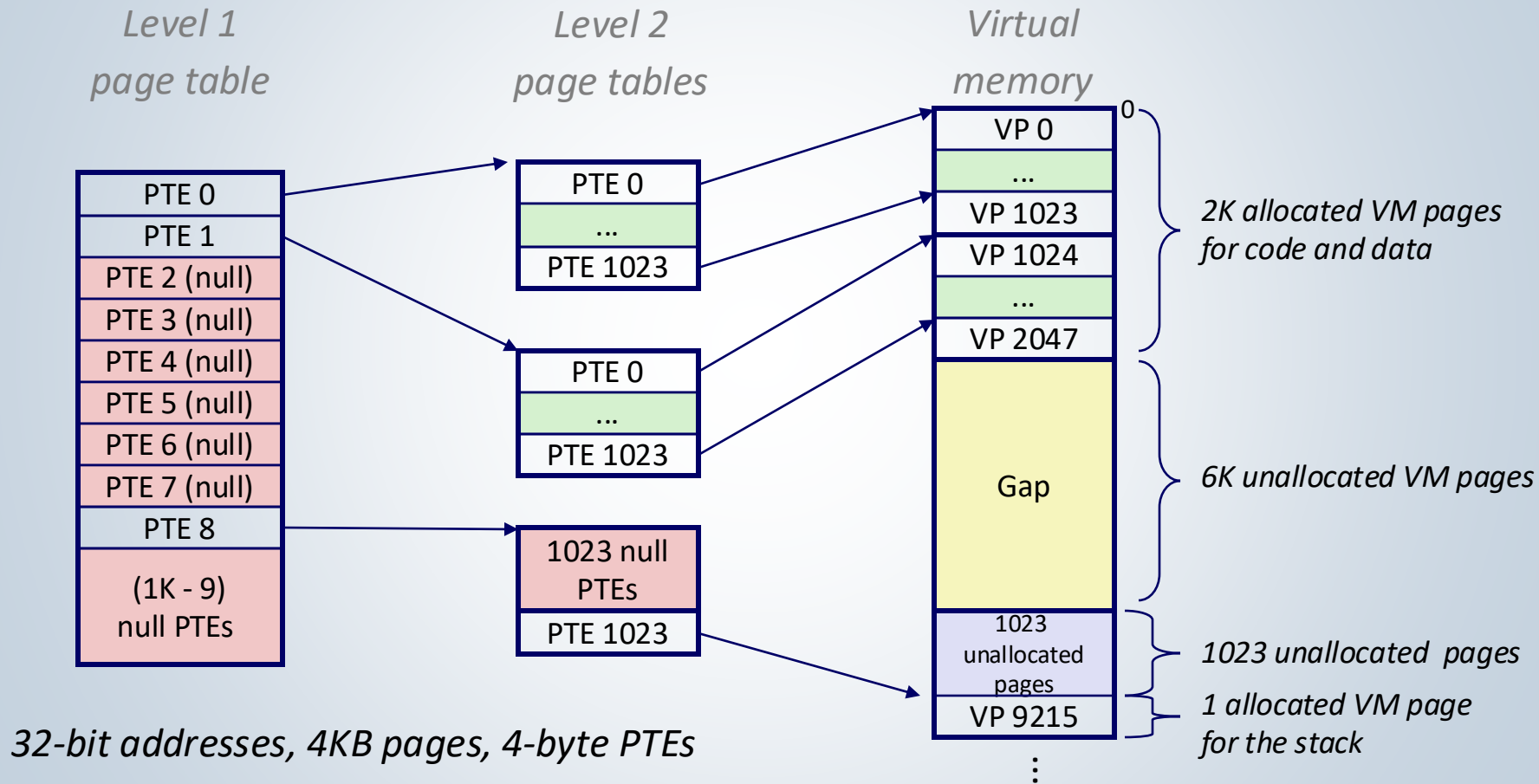
- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) Valid bit is zero, so MMU triggers page fault exception
- 5) Handler identifies victim (and, if dirty, pages it out to disk)
- 6) Handler pages in new page and updates PTE in memory
- 7) Handler returns to original process, restarting faulting instruction

# Multi-Level Page Tables

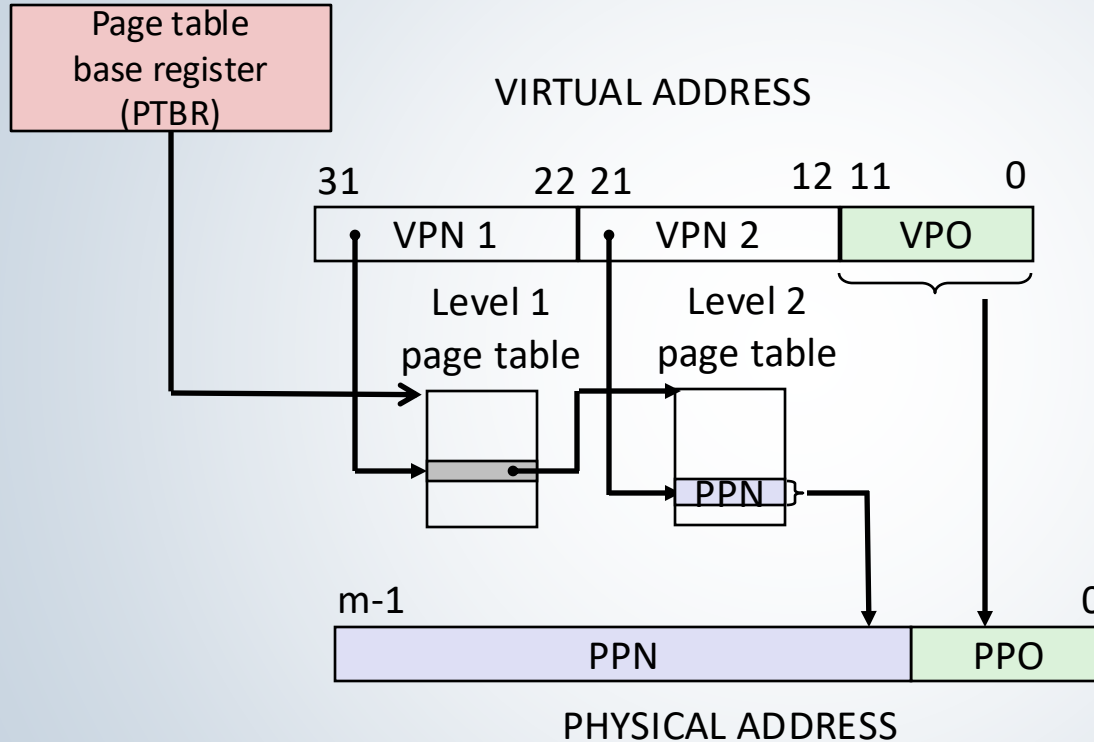
- Suppose:
  - 4KB ( $2^{12}$ ) page size, 48-bit address space, 8-byte PTE ( $2^3$ )
- Problem:
  - Would need a 512 GB page table!
    - $2^{48} * 2^{-12} * 2^3 = 2^{39}$  bytes
- Common solution: Multi-level page table
- Example: 2-level page table
  - Level 1 table: each PTE points to a page table (always memory resident)
  - Level 2 table: each PTE points to a data page (paged in and out like any other data)



# Two-Level Page-Table Hierarchy



# Translating with a 2-Level Page Table





# Multi-level page table (example)

- Consider a 3-level page table. A page table has 64 entries. If the page size is 4K bytes, how much is the size of the virtual address space?

# Multi-level page table (example)

- Another view:
- L3 table maps 64 pages:  $64 \times 4\text{KB} = 256\text{KB}$
- L2 table has 64 entries, each mapping to an L3:  $64 \times 256\text{KB} = 16\text{MB}$
- L1 has 64 entries, each mapping to an L2:  $64 \times 16\text{MB} = 1\text{GB}$

# Multi-level page table (example)

- Consider a 3-level page table. A page table has 64 entries. If the page size is 4K bytes, how much is the size of the virtual address space?
- 64 entries per table: 6 bits of index per level
- Page size = 4KB,  $2^{12}$   $\rightarrow$  12 offset bits
- Bits view
  - 3 levels \* 6 bits = 18 + offset = 30 bits
  - The virtual address space size is  $2^{30}$  bytes = 1GB

# Summary

- Programmer's view of virtual memory
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes
- System view of virtual memory
  - Uses memory efficiently by caching virtual memory pages
    - Efficient only because of locality
  - Simplifies memory management and programming
  - Simplifies protection by providing a convenient point to check permissions