[CSED211] Introduction to Computer Software Systems

Lecture 15: Virtual Memory - Concepts

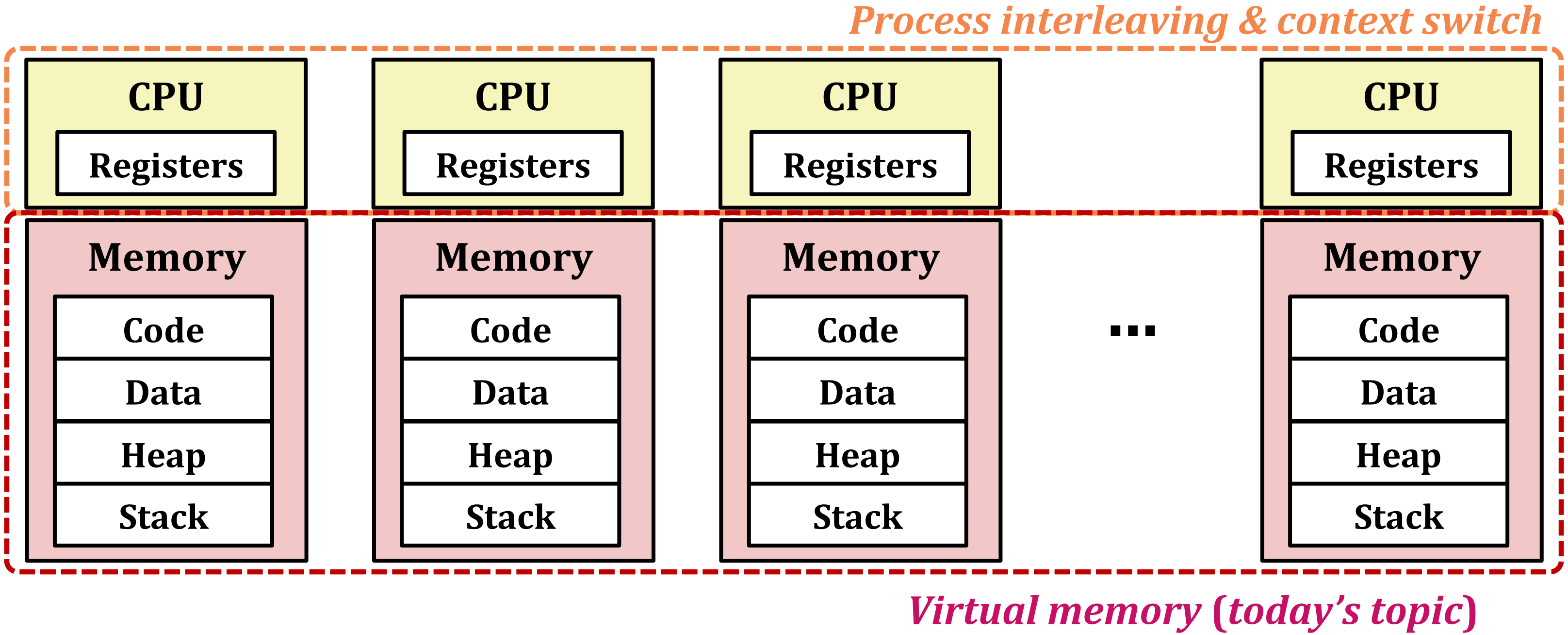
Prof. Jisung Park

CAOS
COMPUTER ARCHITECTURE &
OPERATING SYSTEMS LABORATORY

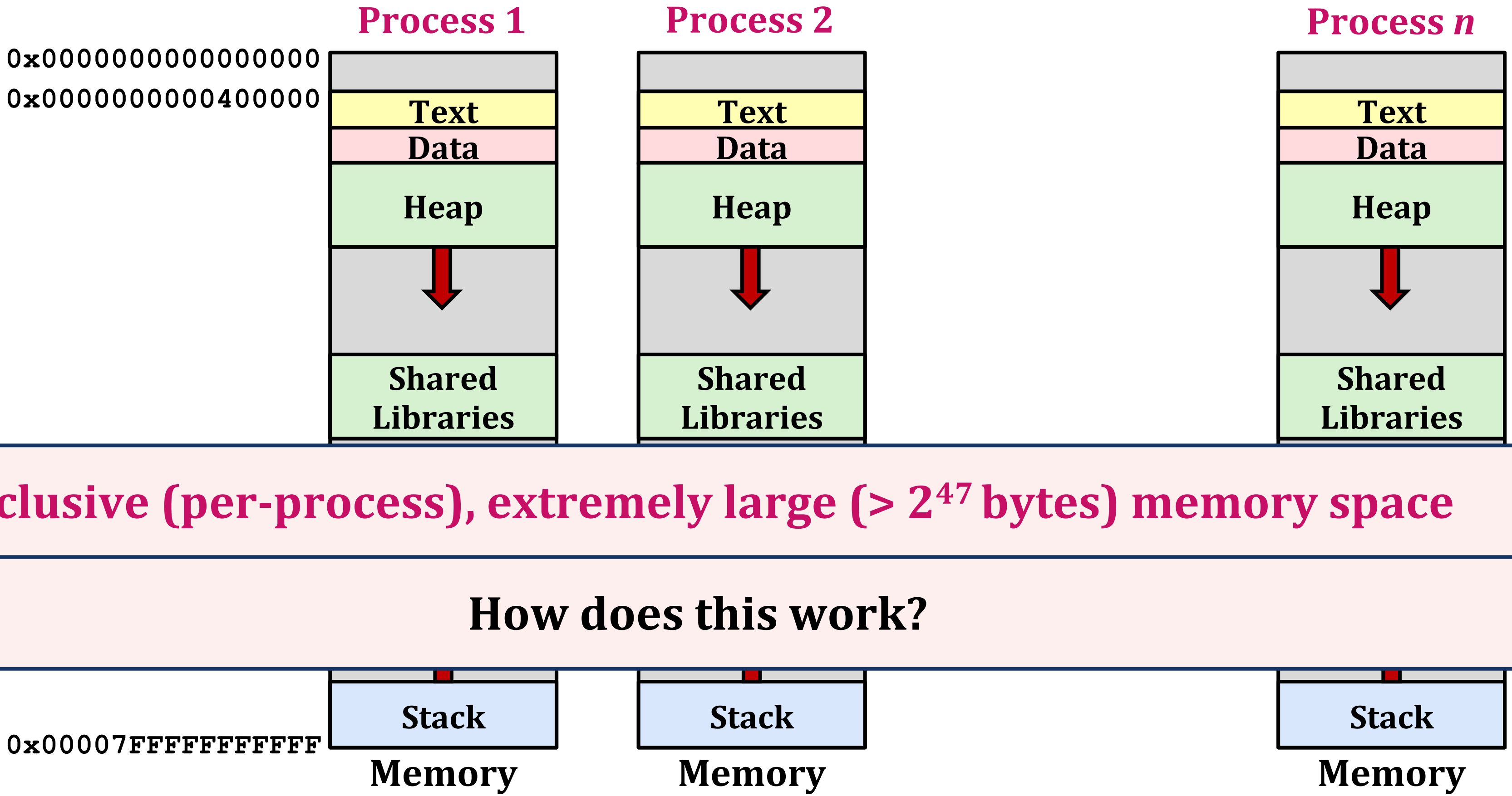
2023.12.06

Recall – Multiprocessing: The Illusion

- Computer runs many processes simultaneously
  - Applications for one or more users: e.g., web browsers, email clients, editors, etc.
  - Background tasks: e.g., monitoring network & I/O devices



Virtual Memory: Concept



Exclusive (per-process), extremely large (> 2^{47} bytes) memory space

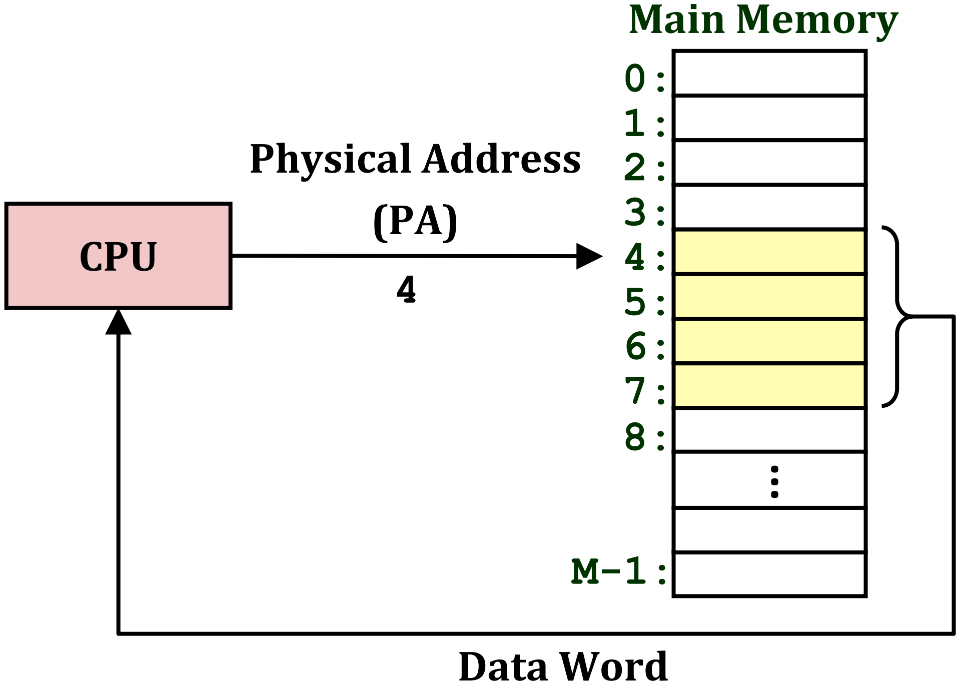
How does this work?

Lecture Agenda

- Address Spaces
- VM as a Tool for Caching
- VM as a Tool for Memory Management
- VM as a Tool for Memory Protection
- Address Translation

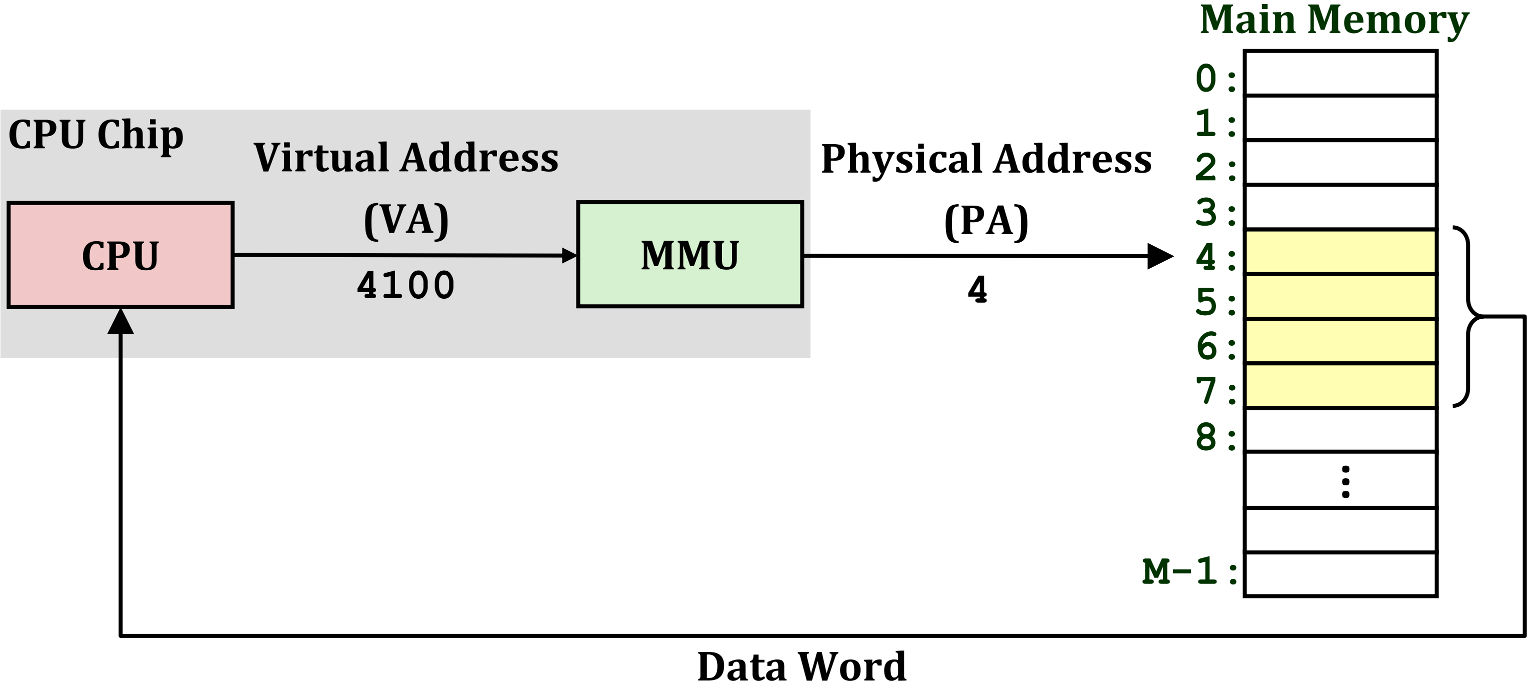
A System Using Physical Addressing

- Used in simple systems, e.g., cars, elevators, and digital picture frames



A System Using Physical Addressing

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



Address Spaces

- Linear address space: an ordered set of contiguous non-negative interger addresses, i.e., {0, 1, 2, 3, … }

- Virtual address space: a set of $N = 2^n$ virtual addresses, i.e., {0, 1, 2, 3, … , N-1}

- Physical address space: a set of $M = 2^m$ physical addresses, i.e., {0, 1, 2, 3, … , M-1}

Why Virtual Memory (VM)?

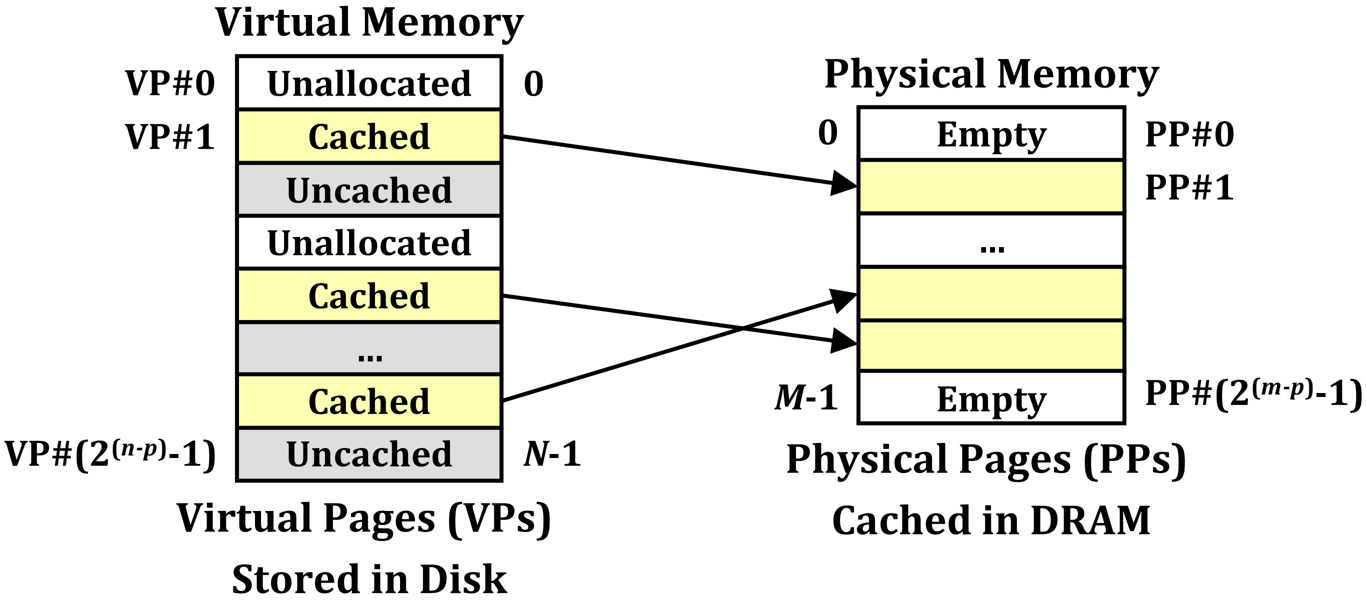
- Efficiently uses main memory
  - Uses DRAM as a cache for the parts of a virtual address space
- Simplifies memory management
  - Each process gets the same uniform linear address space
- Isolates address spaces
  - One process cannot interfere with another's memory
  - User programs cannot access privileged kernel informaion

Lecture Agenda

- 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

- Conceptually, virtual memory is an N-byte array stored in a disk
    - Part of the array is cached in (M-byte) physical memory (e.g., in DRAM)
    - These cache blocks are called pages (size is $P = 2^p$ bytes)



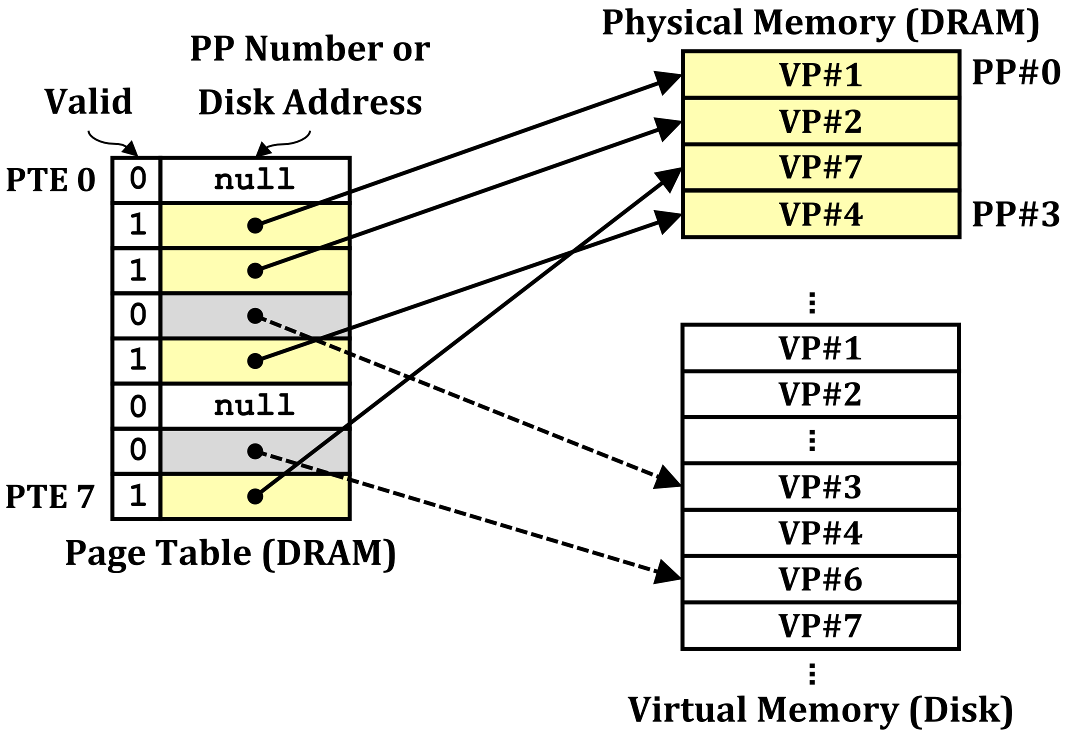
DRAM Cache Organization

- Driven by the enormous miss penalty
  - DRAM is about 10× slower than SRAM
  - Disk is about 10,000× slower than DRAM

- Consequences
  - Large page (i.e., cache block) size: typically 4-8 KiB, sometimes 4 MiB
  - 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

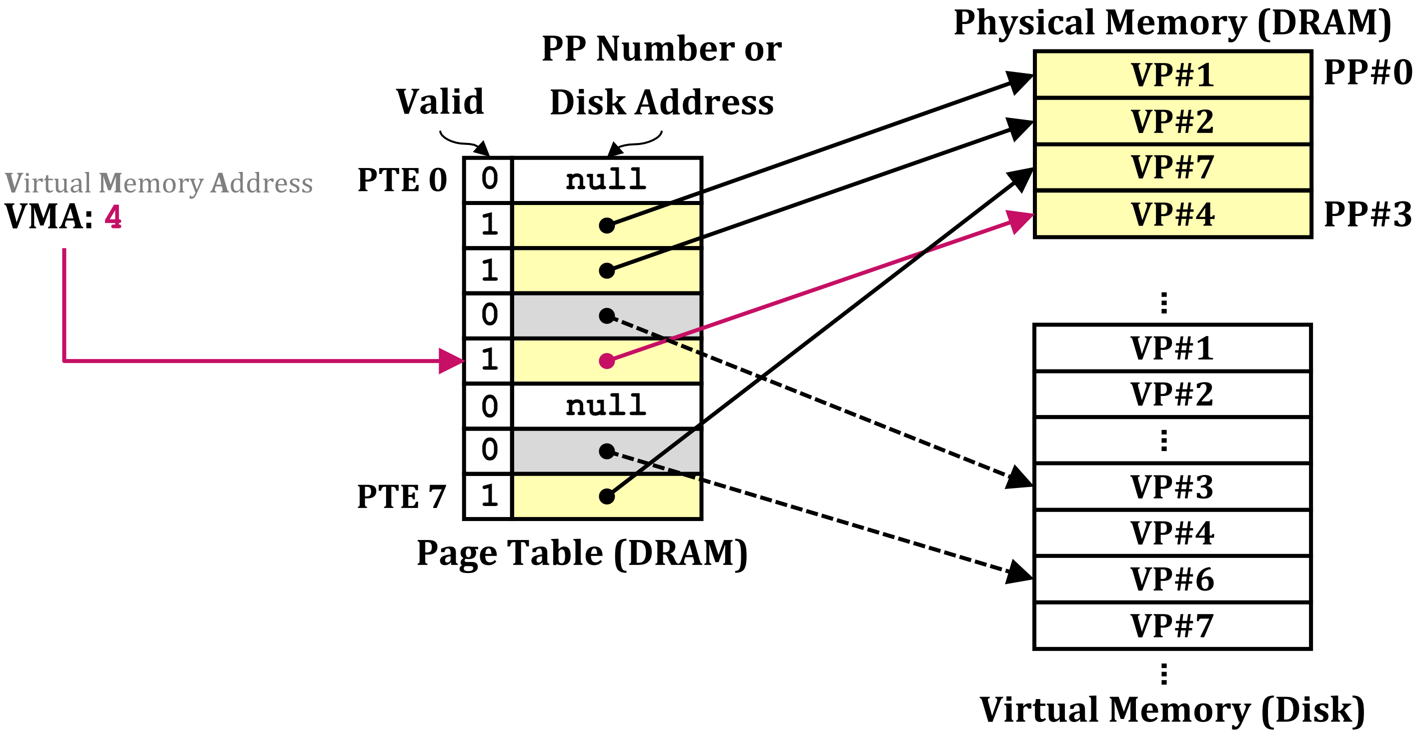
Enabling Data Structure: Page Table

- An array of page table entries (PTEs) storing virtual-to-physical mappings
  - Per-process kernel data structure in DRAM



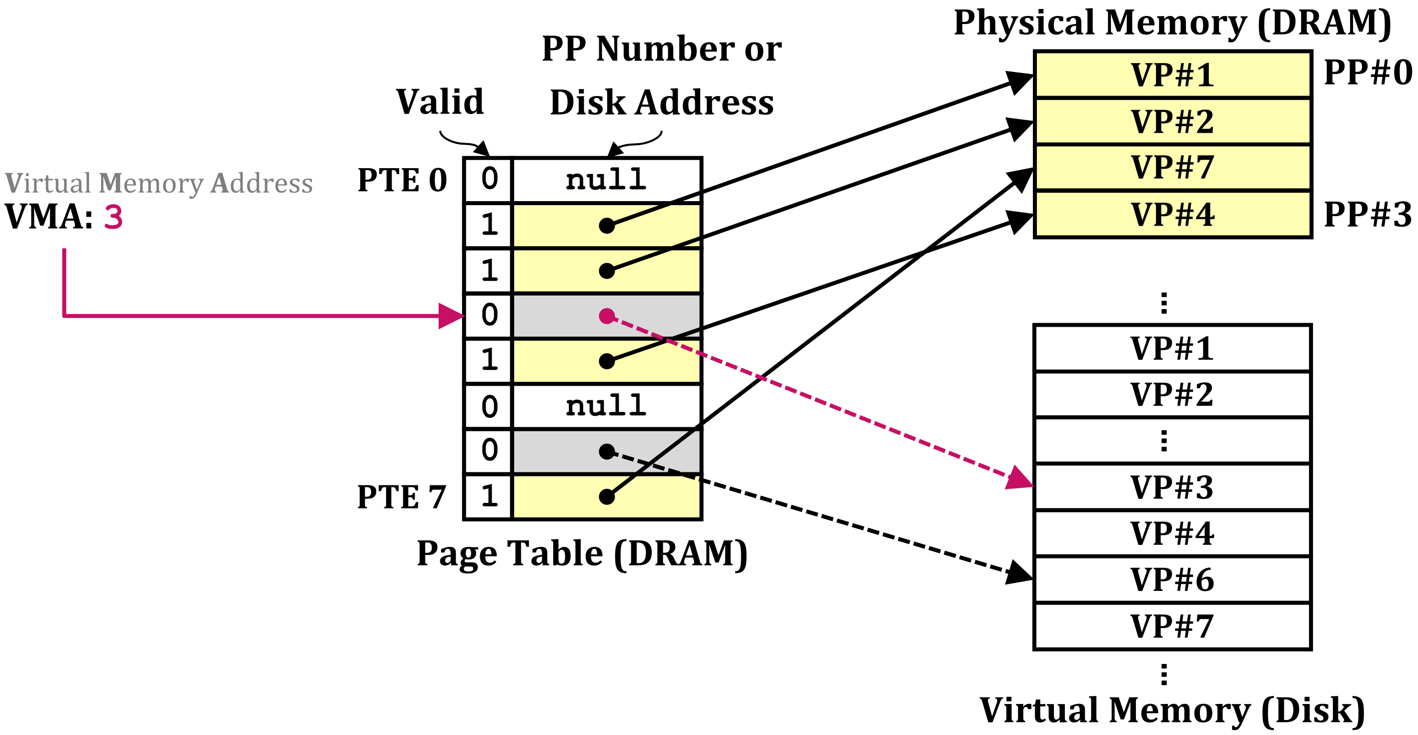
Page Hit

- Reference to a VM word in physical memory, i.e., DRAM cache hit



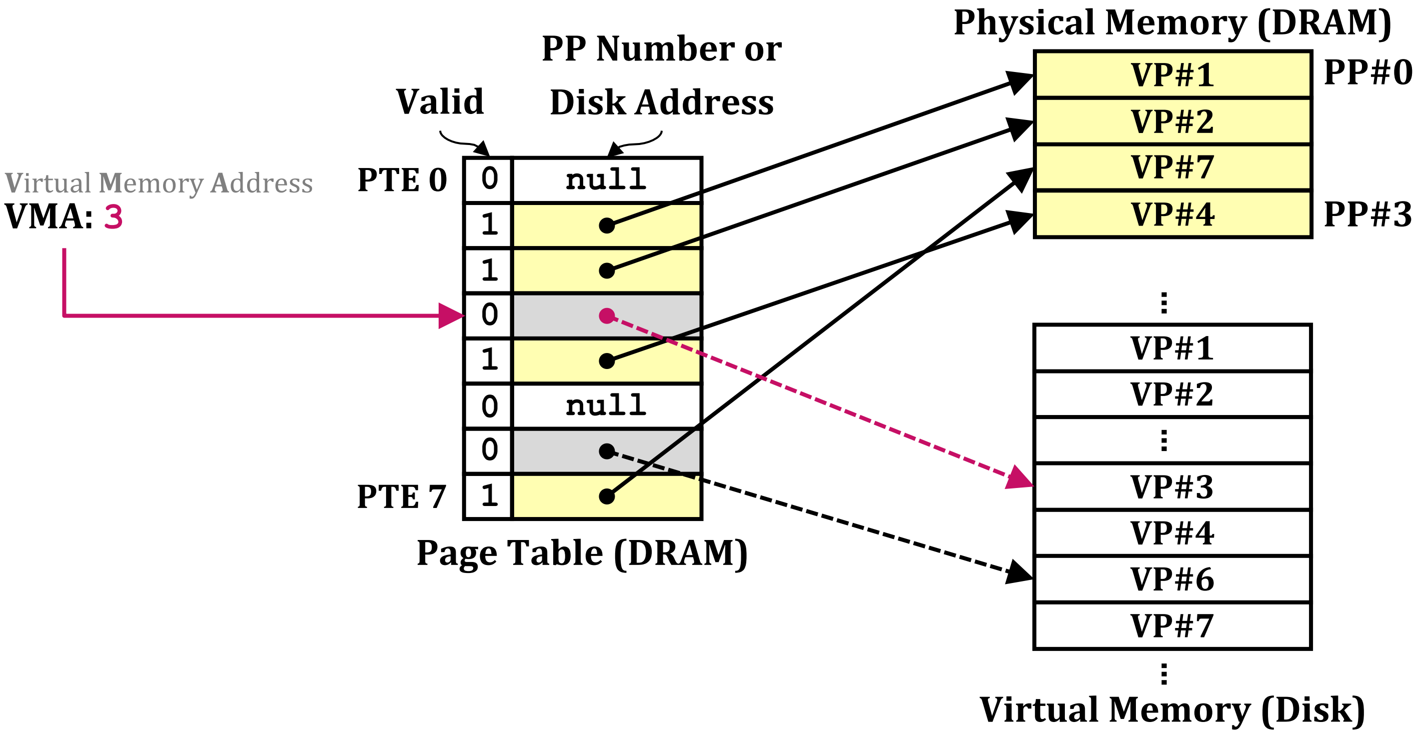
Page Fault

- Reference to a VM word not in physical memory, i.e., DRAM cache miss



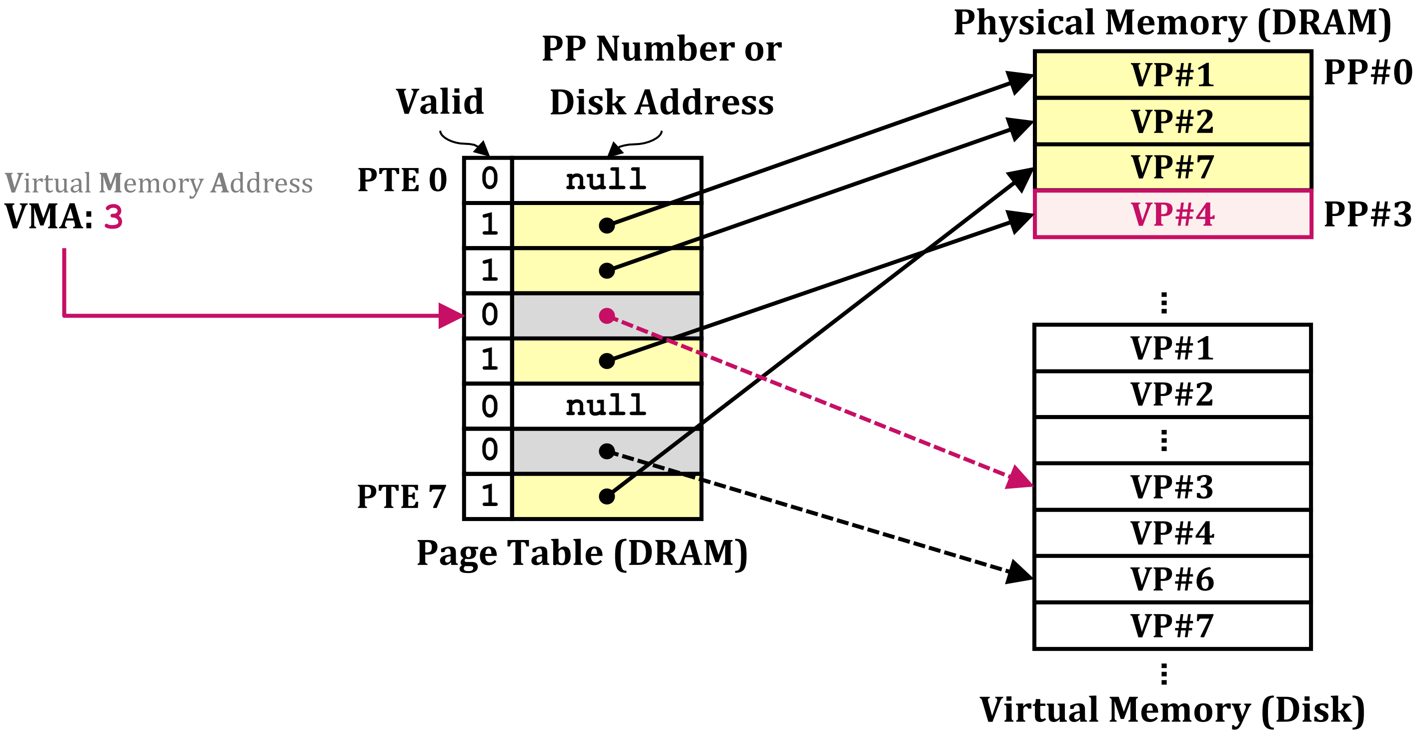
Page Fault Handling

- A page fault is treated as an exception (caused by a page miss)



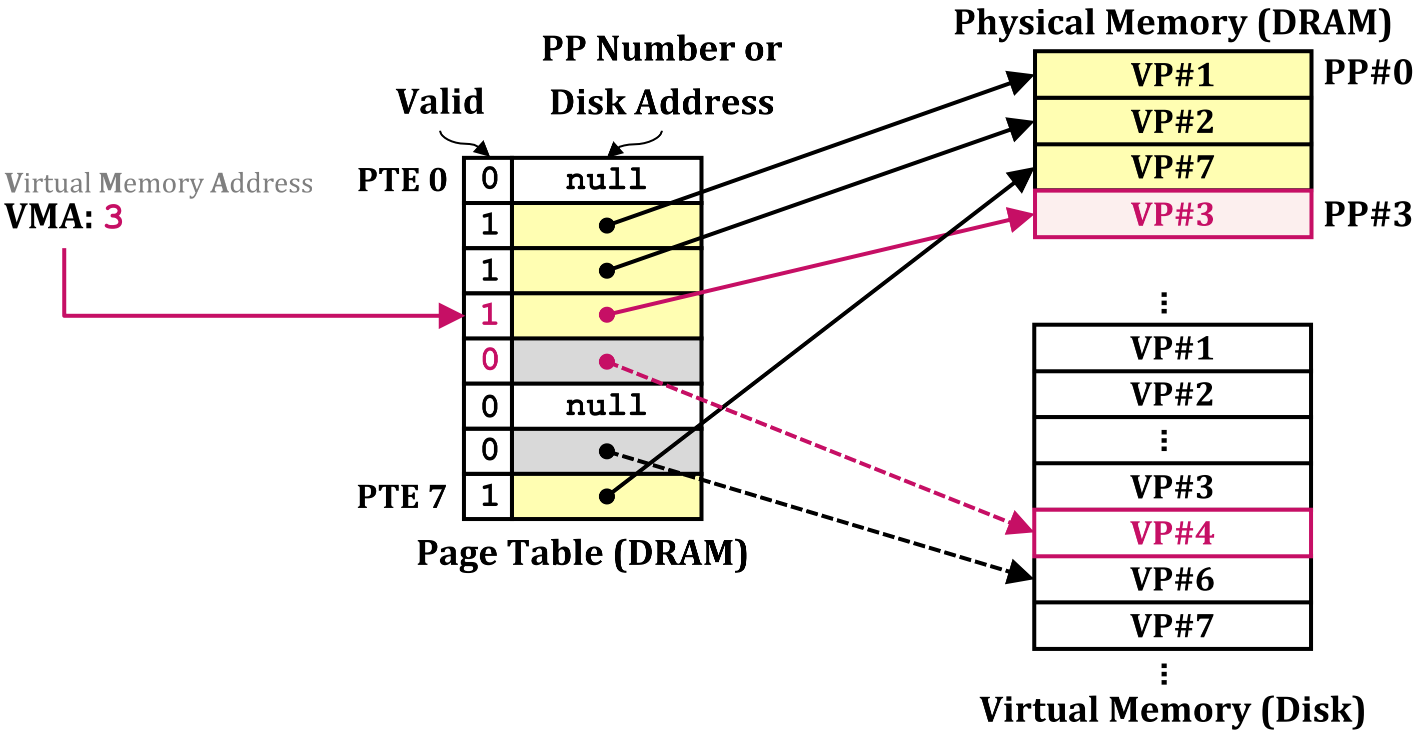
Page Fault Handling

- A page fault is treated as an exception (caused by a page miss)
  - Page fault handler evicts a victim (e.g., VP#4)



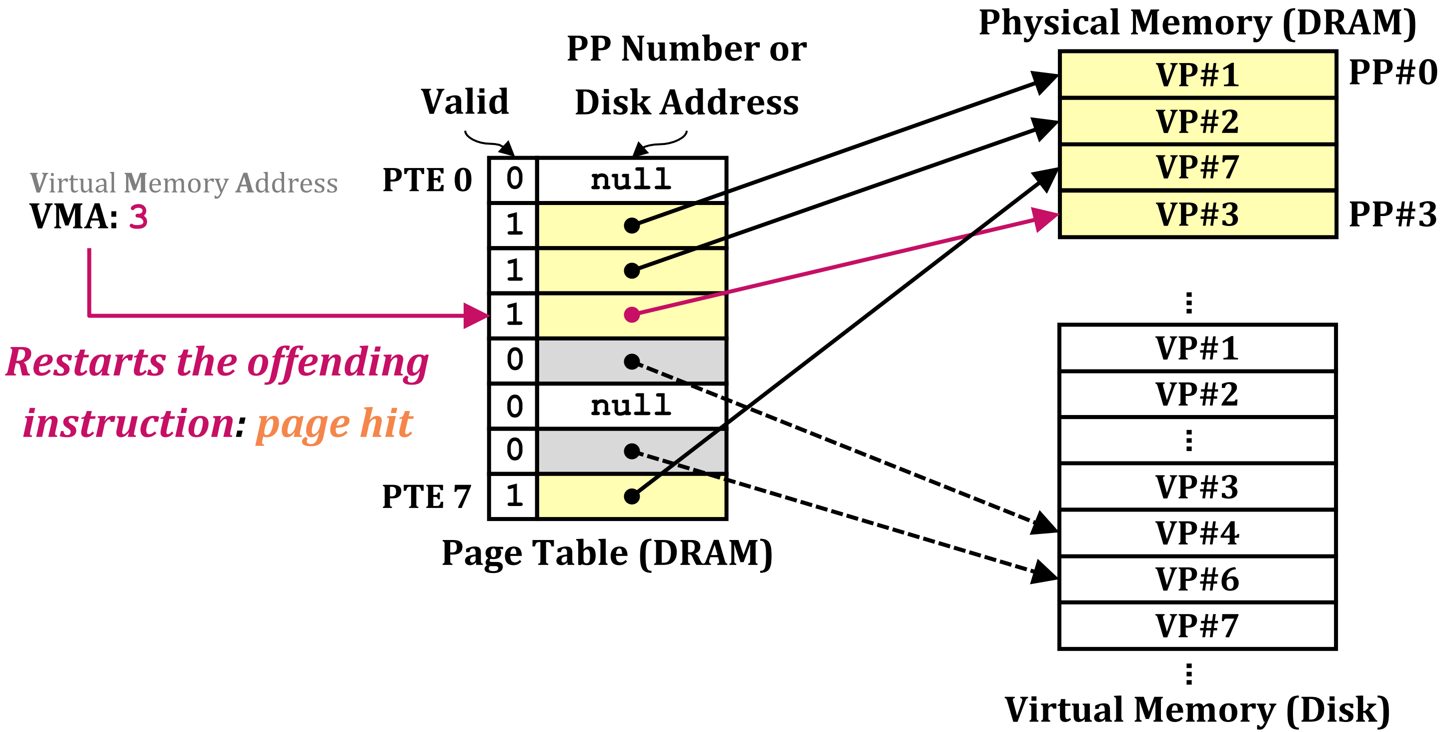
Page Fault Handling

- A page fault is treated as an exception (caused by a page miss)
  - Page fault handler evicts a victim (e.g., VP#4) and fetch the target (e.g., VP#3)



Page Fault Handling

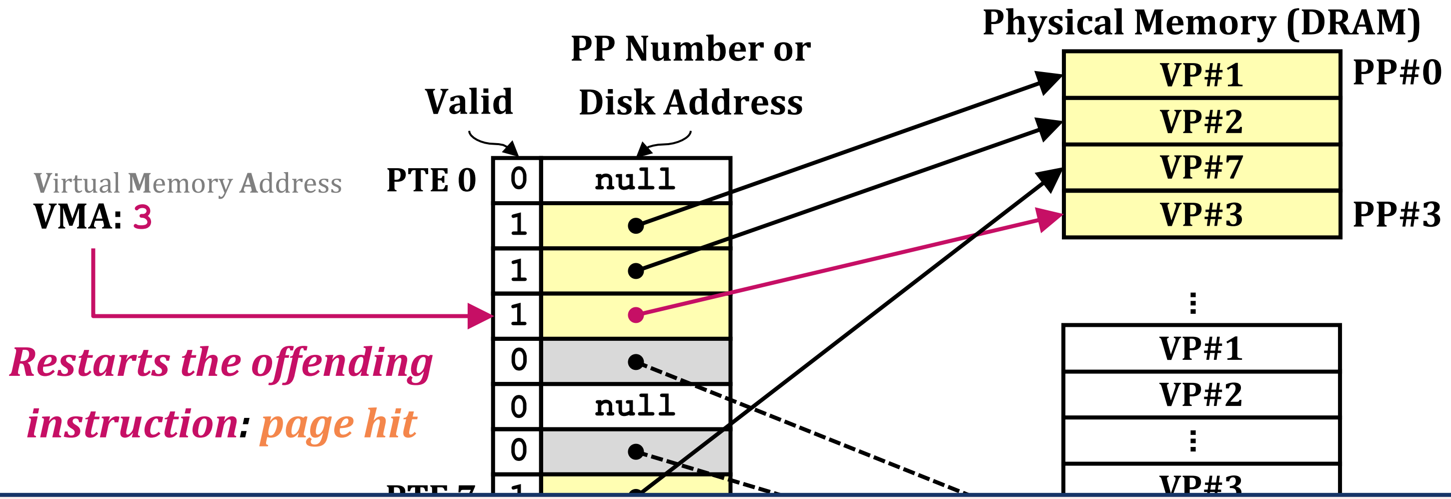
- A page fault is treated as an exception (caused by a page miss)
    - Page fault handler evicts a victim (e.g., VP#4) and fetch the target (e.g., VP#3)



Restarts the offending instruction: page hit

Page Fault Handling

- A page fault is treated as an exception (caused by a page miss)
    - Page fault handler evicts a victim (e.g., VP#4) and fetch the target (e.g., VP#3)



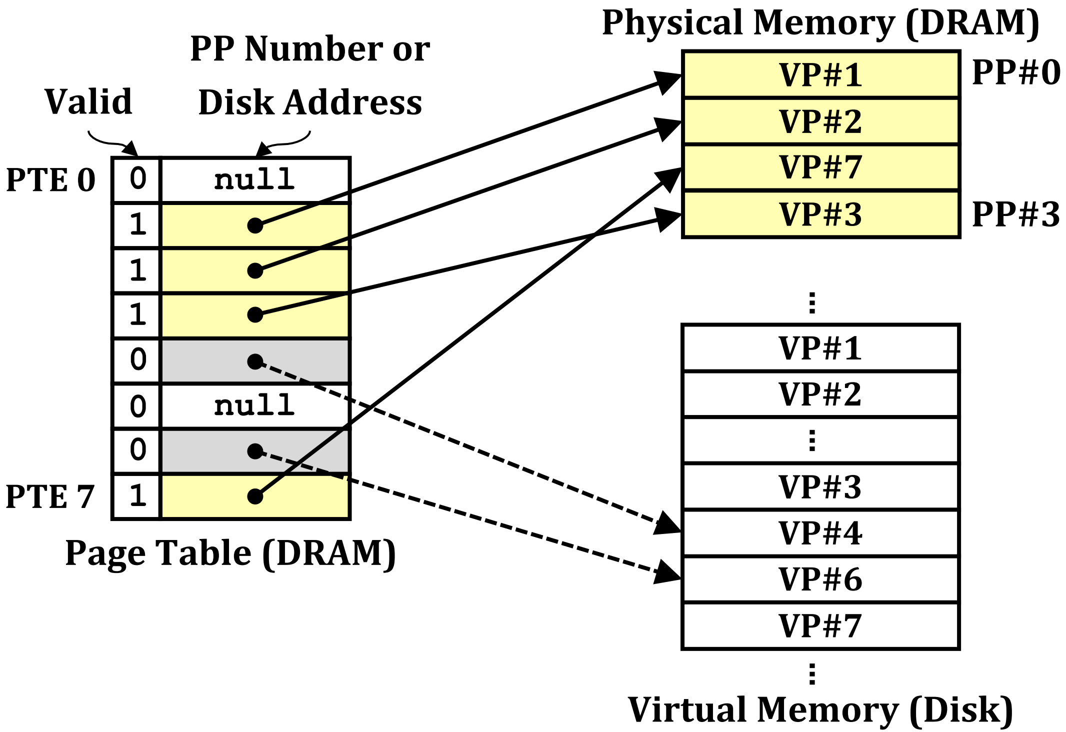
Restarts the offending instruction: page hit

Demand paging: waiting until the miss to copy a page to DRAM

Virtual Memory (Disk)

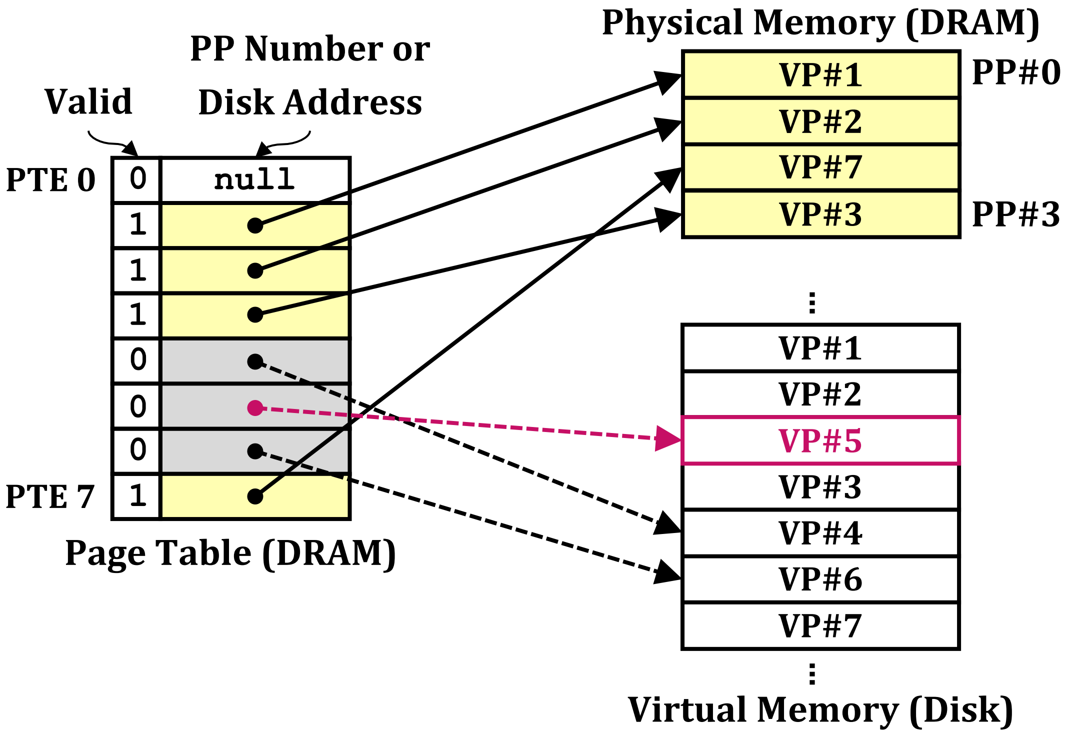
Page Allocation

- Only sets the corresponding PTE (e.g., for VP#5)
  - Subsequent page fault will bring the page into memory



Page Allocation

- Only sets the corresponding PTE (e.g., for VP#5)
  - Subsequent page fault will bring the page into memory



Locality to the Rescue Again

- Virtual memory may seem terribly inefficient but works because of locality

- At any point in time, programs tend to access a set of active virtual pages
  - Which is 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

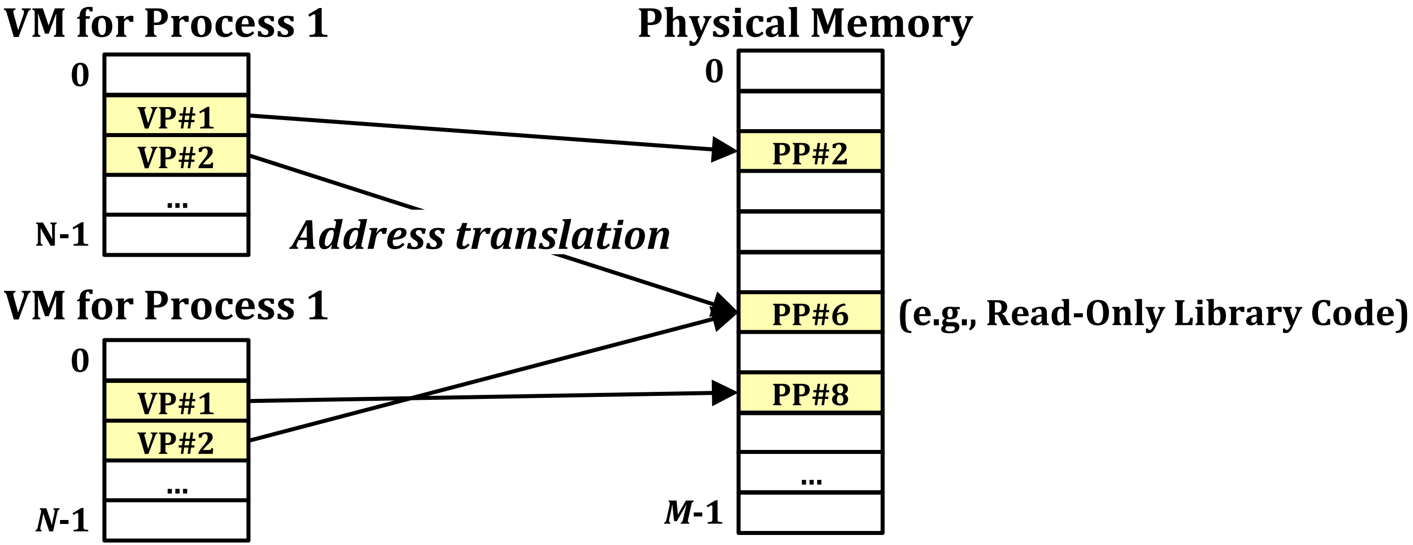
- If (total working set size > main memory size)
  - Cache thrashing: performance meltdown where pares are swapped (copied) in and out continuously

Lecture Agenda

- Address Spaces
- VM as a Tool for Caching
- VM as a Tool for Memory Management
- VM as a Tool for Memory Protection
- Address Translation

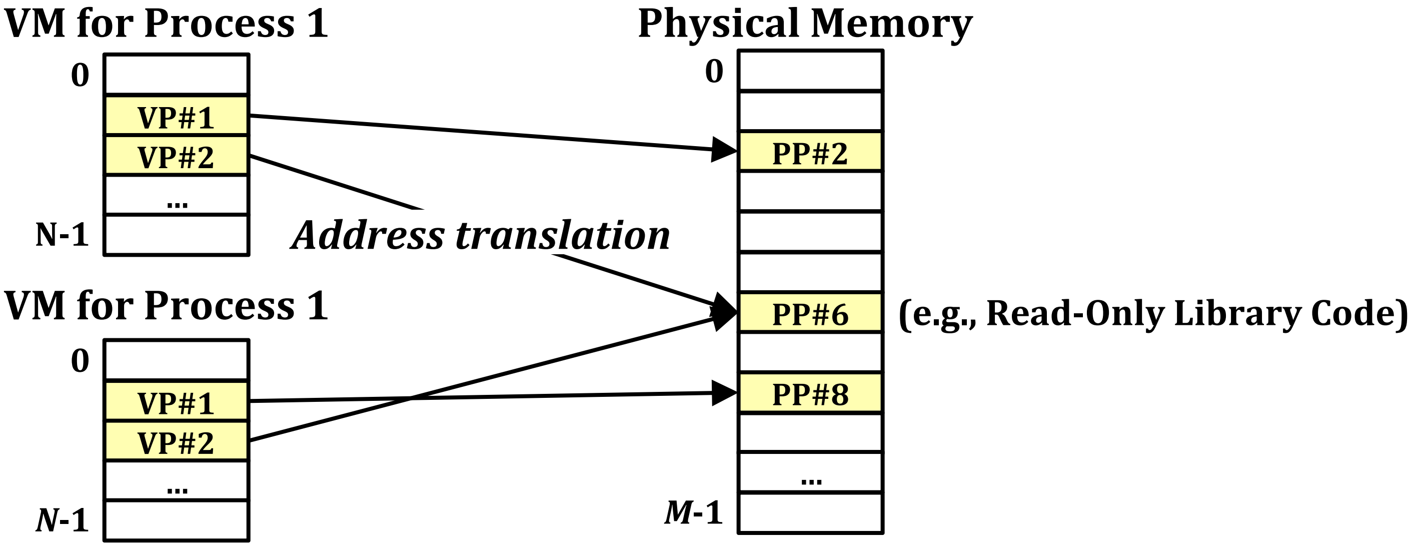
VM as a Tool for Memory Management

- Each process has its own virtual address space
  - Allows each process to 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 (Cont.)

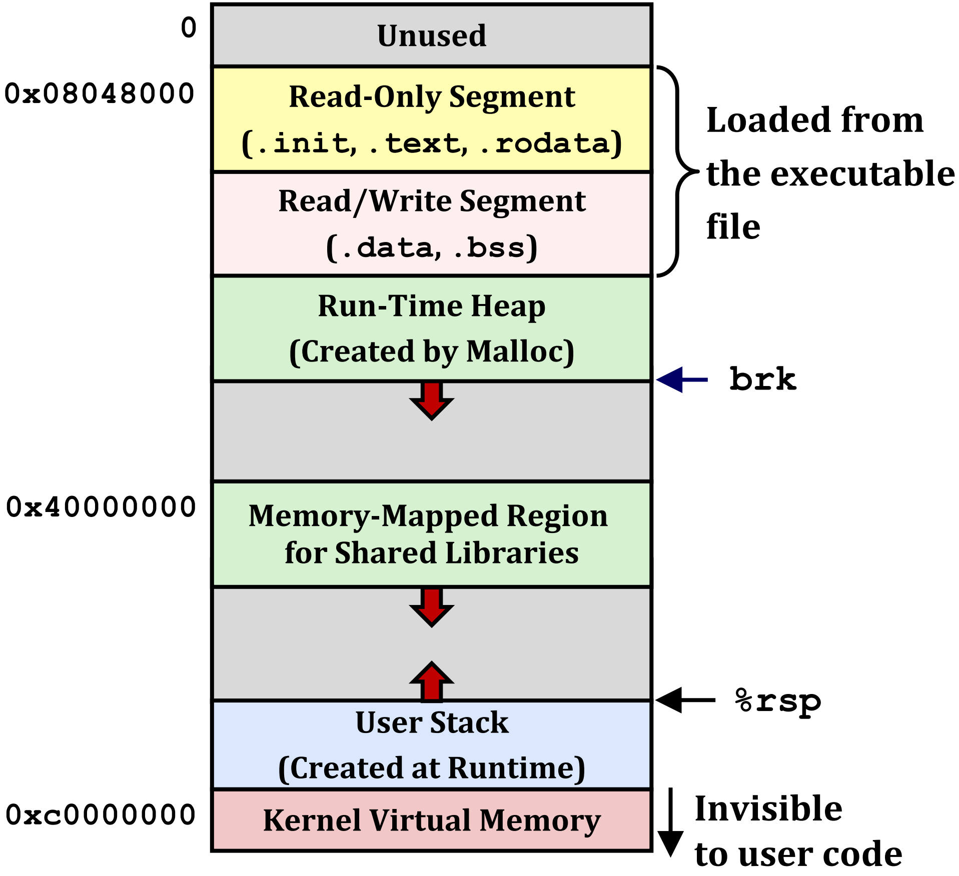
- Simplifying memory allocation
  - Each virtual page can be mapped to any physical page
  - A virtual page can be stored in different physical pages at different times
- Sharing code and data among processes
  - Map virtual pages to the same physical page (e.g., PP#6)



Simplifying Linking and Loading

- Linking
  - Each program has similar virtual address space
  - Code, stack, and shared libraries always 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, paged by page, on demand by the virtual memory system

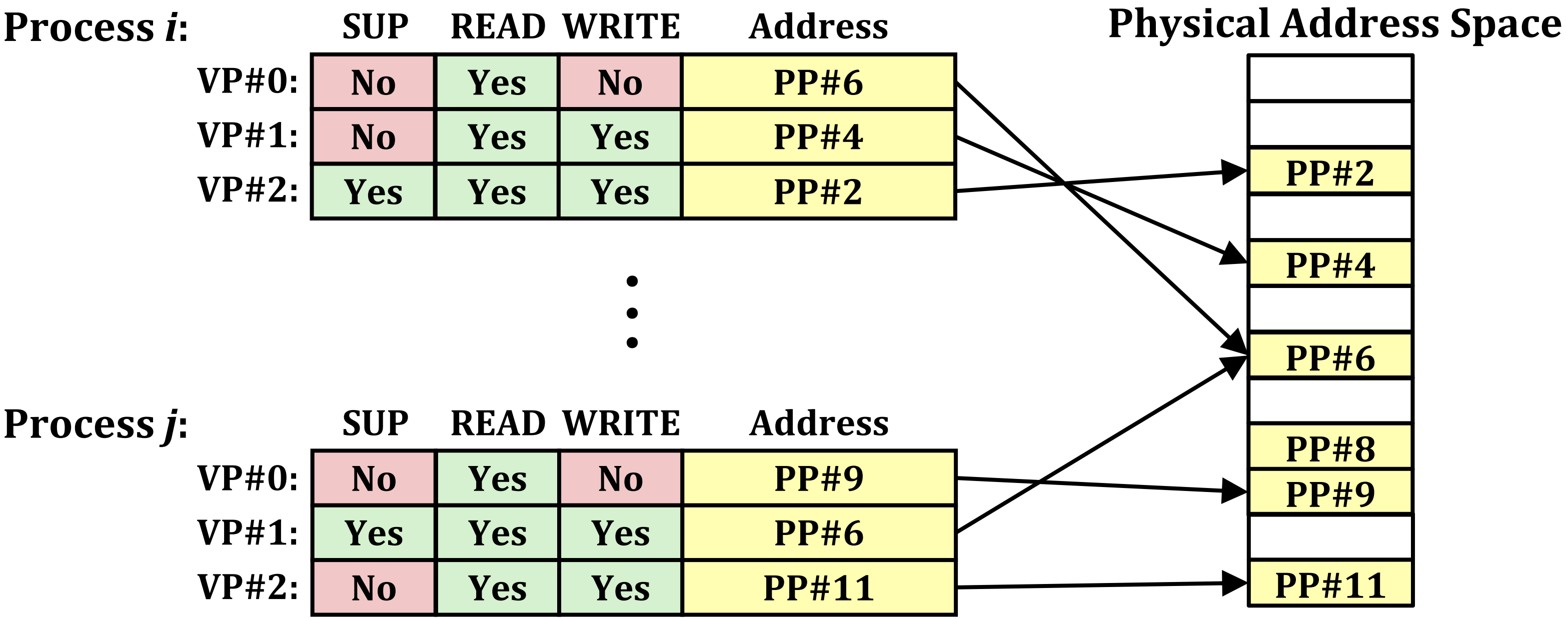


Lecture Agenda

- 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

Physical Address Space

Lecture Agenda

- 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 Address Translation

- Virtual address space: V = {0, 1, …, N−1}

- Physical address space: P = {0, 1, …, M−1}

- Address translation
  - MAP: V → P ∪ {∅}
  - For virtual address a:
    - MAP(a) = a' if data at virtual address a is at physical address $a' \in P$
    - MAP(a) = ∅ if data at virtual address a is not in physical memory
      - Either invalid or stored on disk

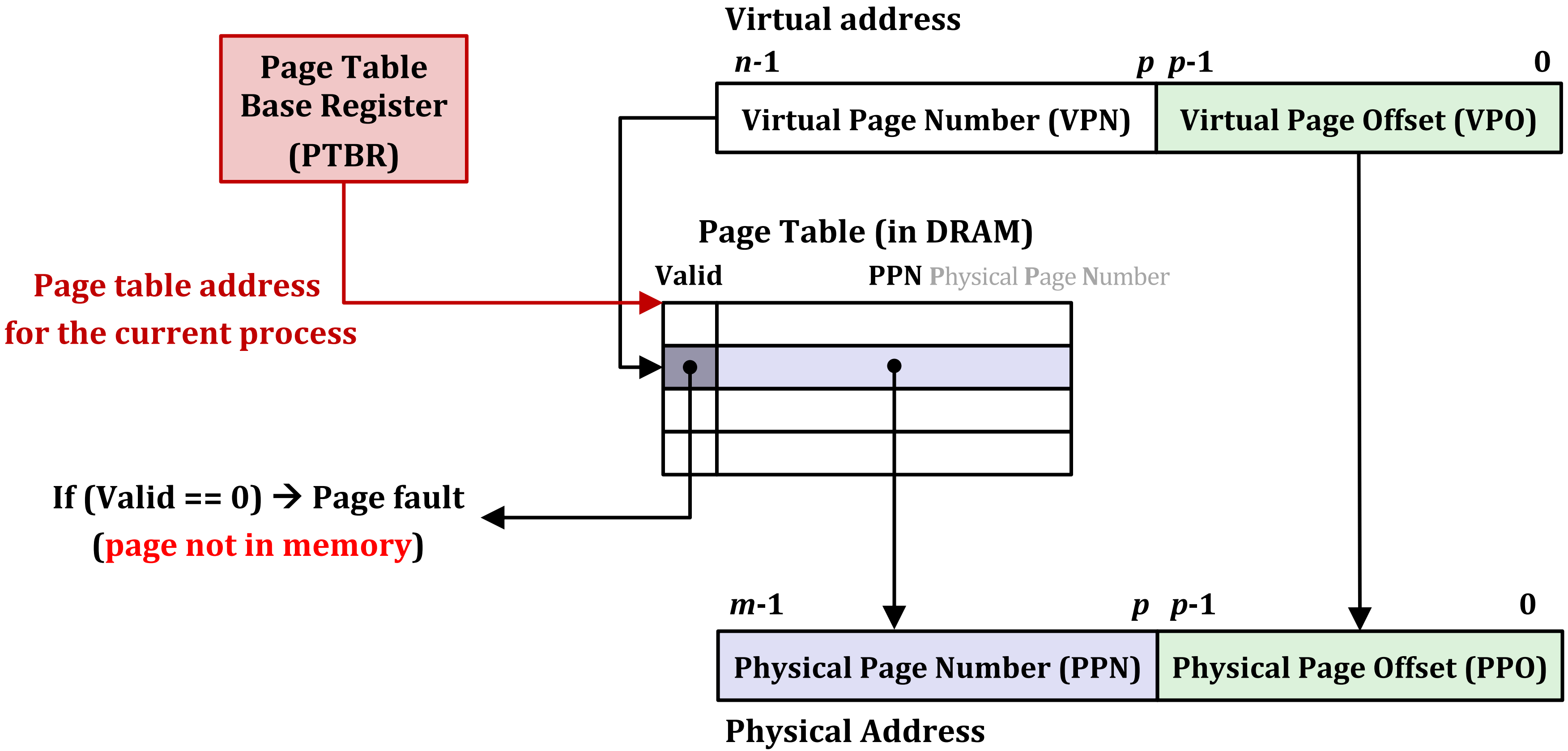
Summary of Address Translation Symbols

- Basic Parameters
  - N = 2^n: the number of addresses in virtual address space
  - M = 2^m: the number of addresses in physical address space
  - P = 2^p : the page size (bytes)

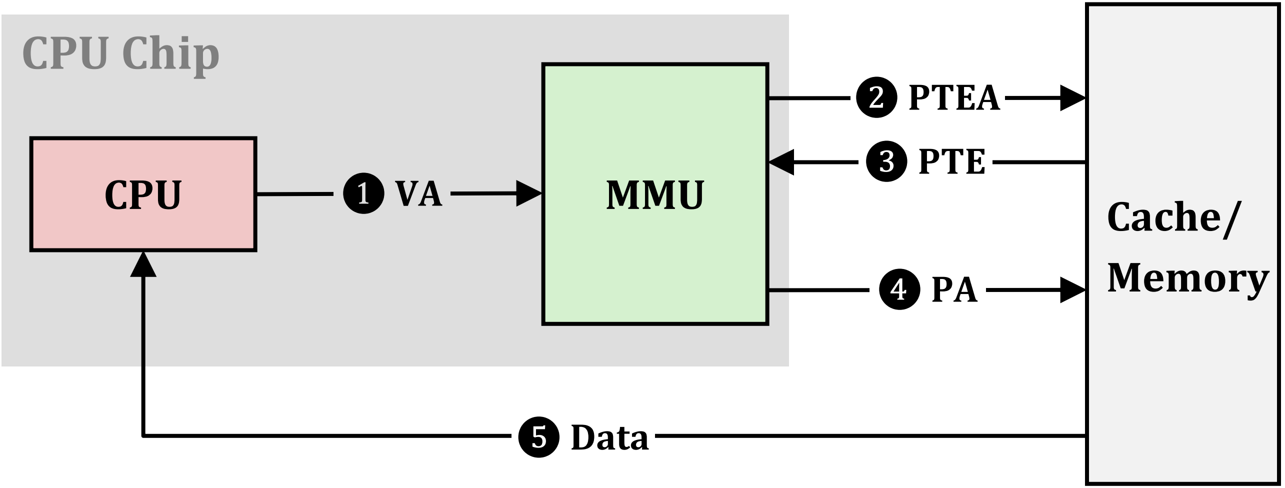
- Components of the virtual address (VA)
  - VPO: virtual page offset
  - VPN: virtual page number

- Components of the physical address (PA)
  - PPO: physical page offset (same as VPO)
  - PPN: physical page number
  - CO: byte offset within cache line
  - CI: cache index
  - CT: cache tag

Address Translation with a Page Table


Virtual address
n-1
p
p-1
0
Virtual Page Number (VPN)
Virtual Page Offset (VPO)
Page Table Base Register (PTBR)
Page table address for the current process
Page Table (in DRAM)
Valid
PPN Physical Page Number
If (Valid == 0) → Page fault
(page not in memory)
m-1
p
p-1
0
Physical Page Number (PPN)
Physical Page Offset (PPO)
Physical Address

Address Translation: Page Hit



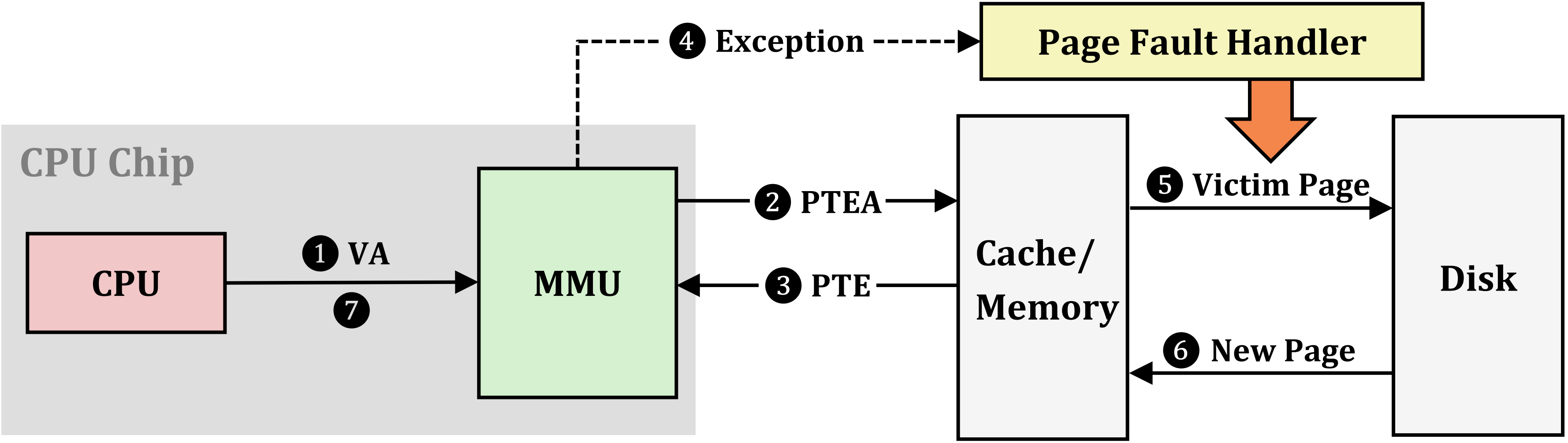
❶ Processor sends virtual address to MMU

❷, ❸ MMU fetches PTE from page table in memory

❹ MMU sends physical address to cache/memory

❺ Cache/memory sends data word to processor

Address Translation: Page Fault



❶ Processor sends virtual address to MMU

❷, ❸ MMU fetches PTE from page table in memory

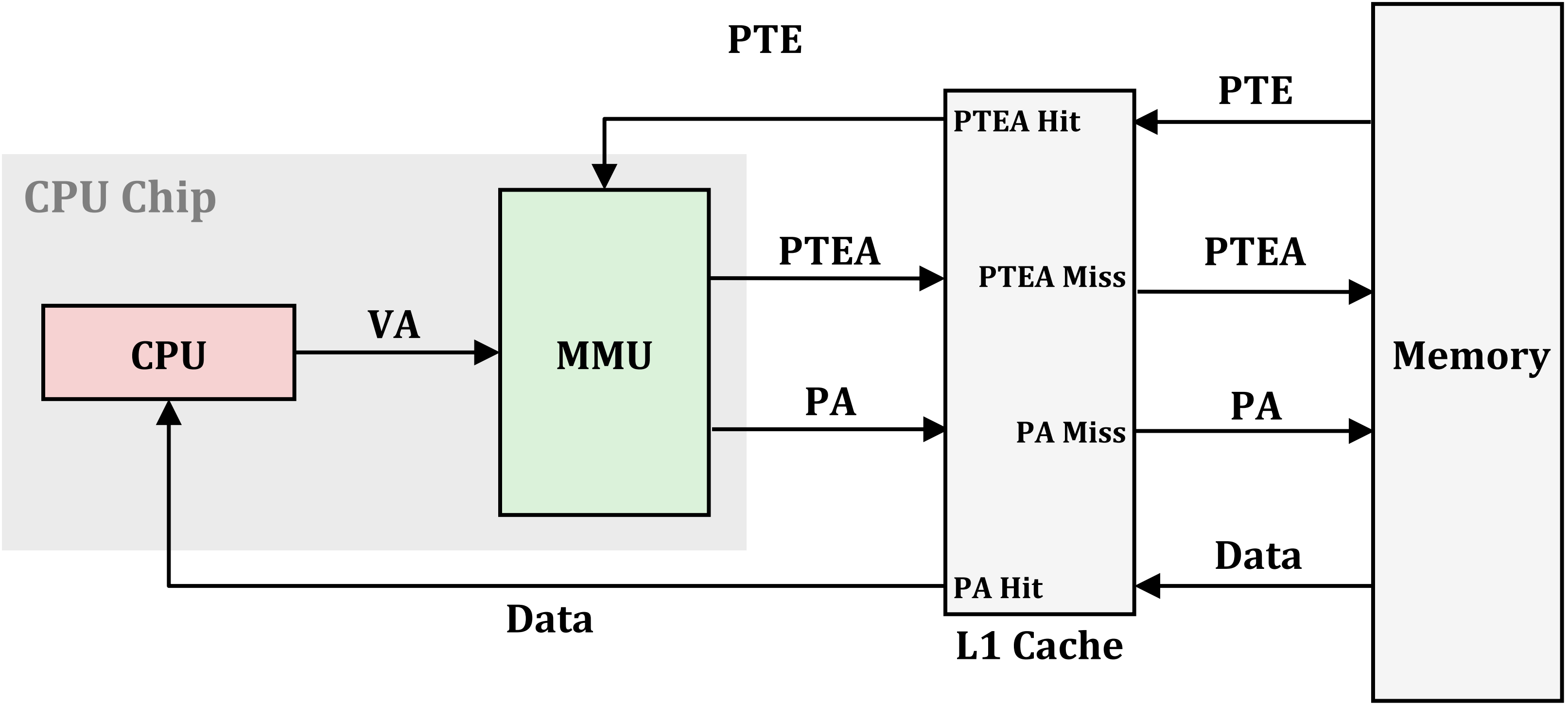
❹ Valid bit is zero, so MMU triggers page fault exception

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

❻ Handler pages in new page and updates PTE in memory

❼ 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

Speeding Up Translation with a TLB

- Problem: PTEs are cached in L1 like any other memory word
  - PTEs may be evicted by other data references
  - PTE hit still requires a small L1 delay

- Solution: translation lookaside buffer (TLB)
  - Small hardware cache in MMU
  - Stores virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

Summary of Address Translation Symbols

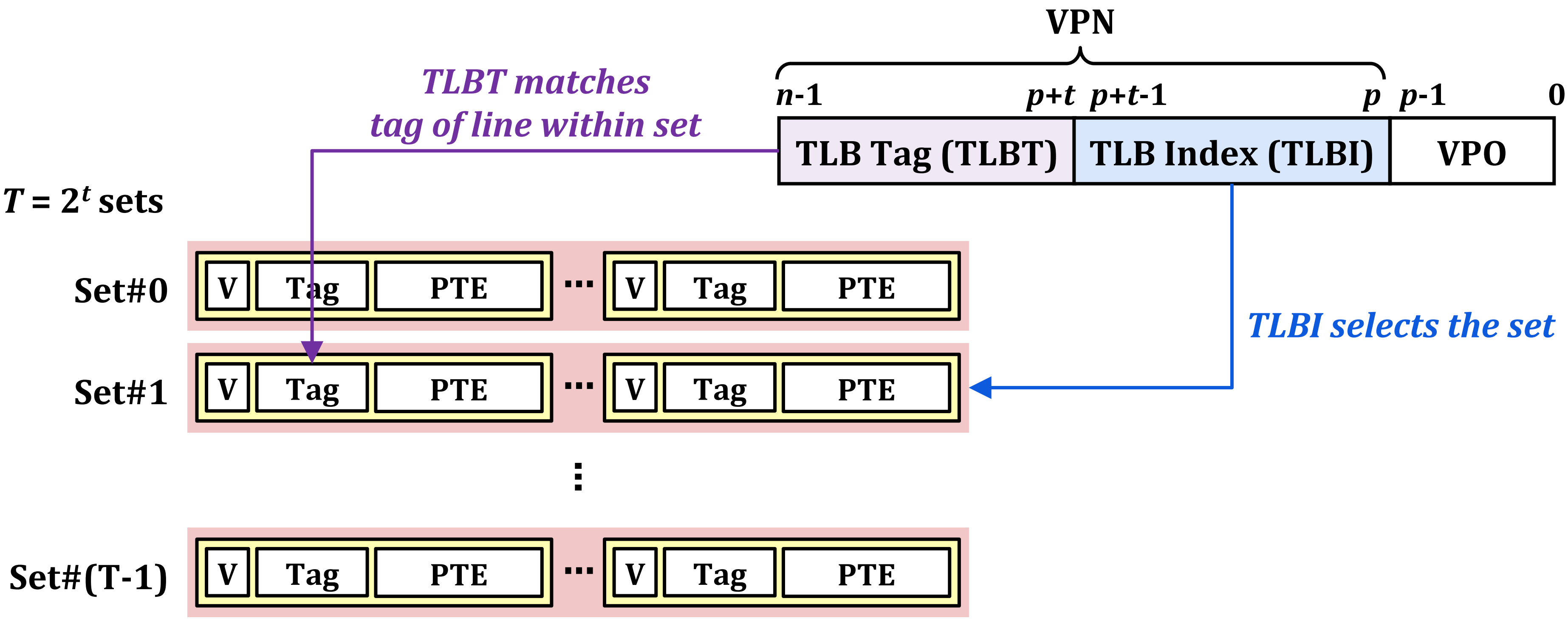
- Basic Parameters
  - N = 2^n: the number of addresses in virtual address space
  - M = 2^m: the number of addresses in physical address space
  - P = 2^p: the page size (bytes)

- Components of the virtual address (VA)
  - TLBI: TLB index
  - TLBT: TLB tag
  - VPO: virtual page offset
  - VPN: virtual page number

- Components of the physical address (PA)
  - PPO: physical page offset (same as VPO)
  - PPN: physical page number

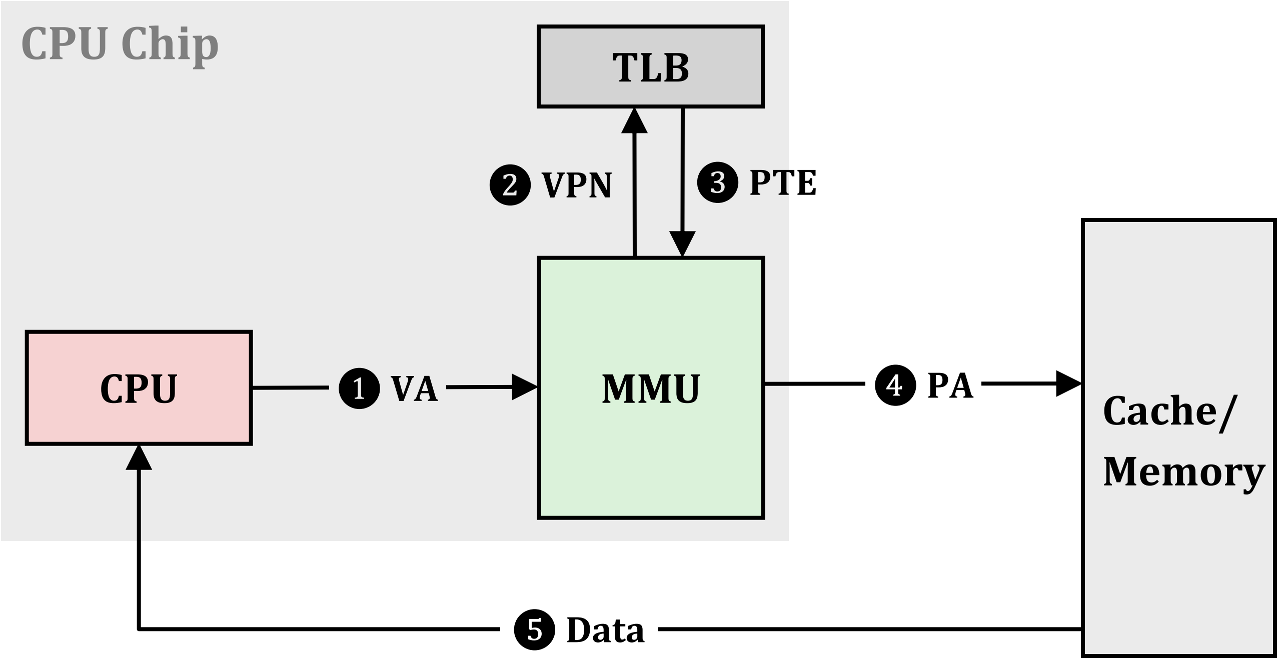
Accessing the TLB

- MMU uses the VPN portion of the virtual address to access the TLB:



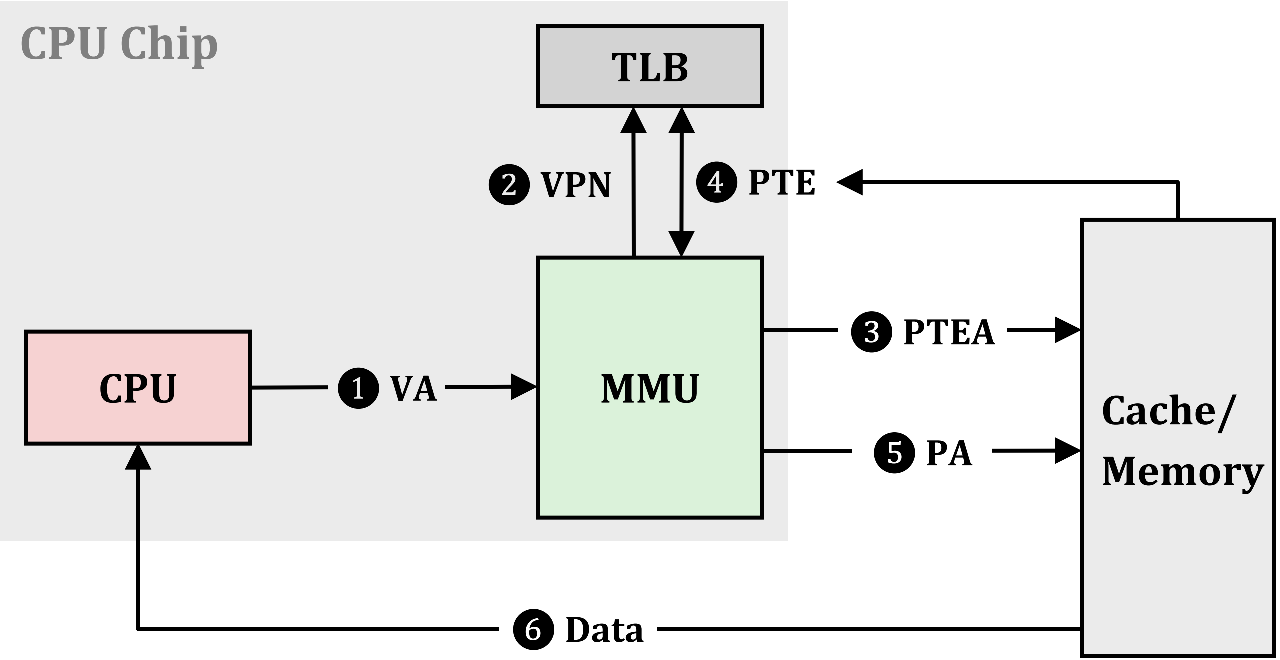
TLB Hit

- A TLB hit eliminates a memory access



TLB Miss

- A TLB miss incurs an additional memory access (the PTE)
  - Fortunately, TLB misses are rare. Why?



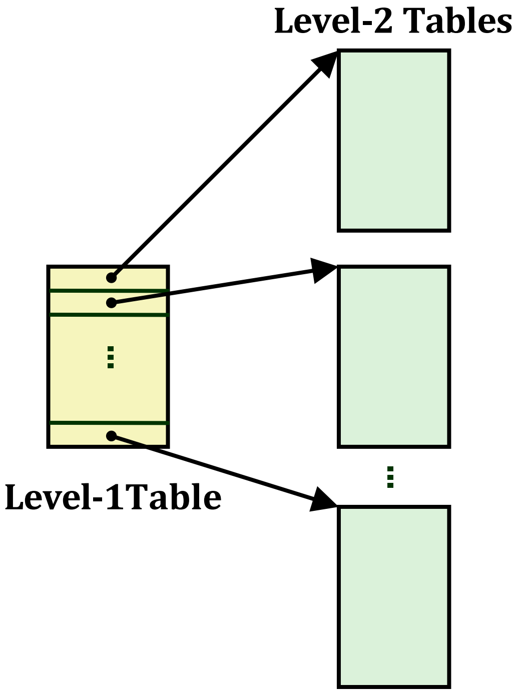
Multi-Level Page Tables

- Assumptions: 4-KiB (2^{12}) page size, 48-bit address space, 8-byte PTE

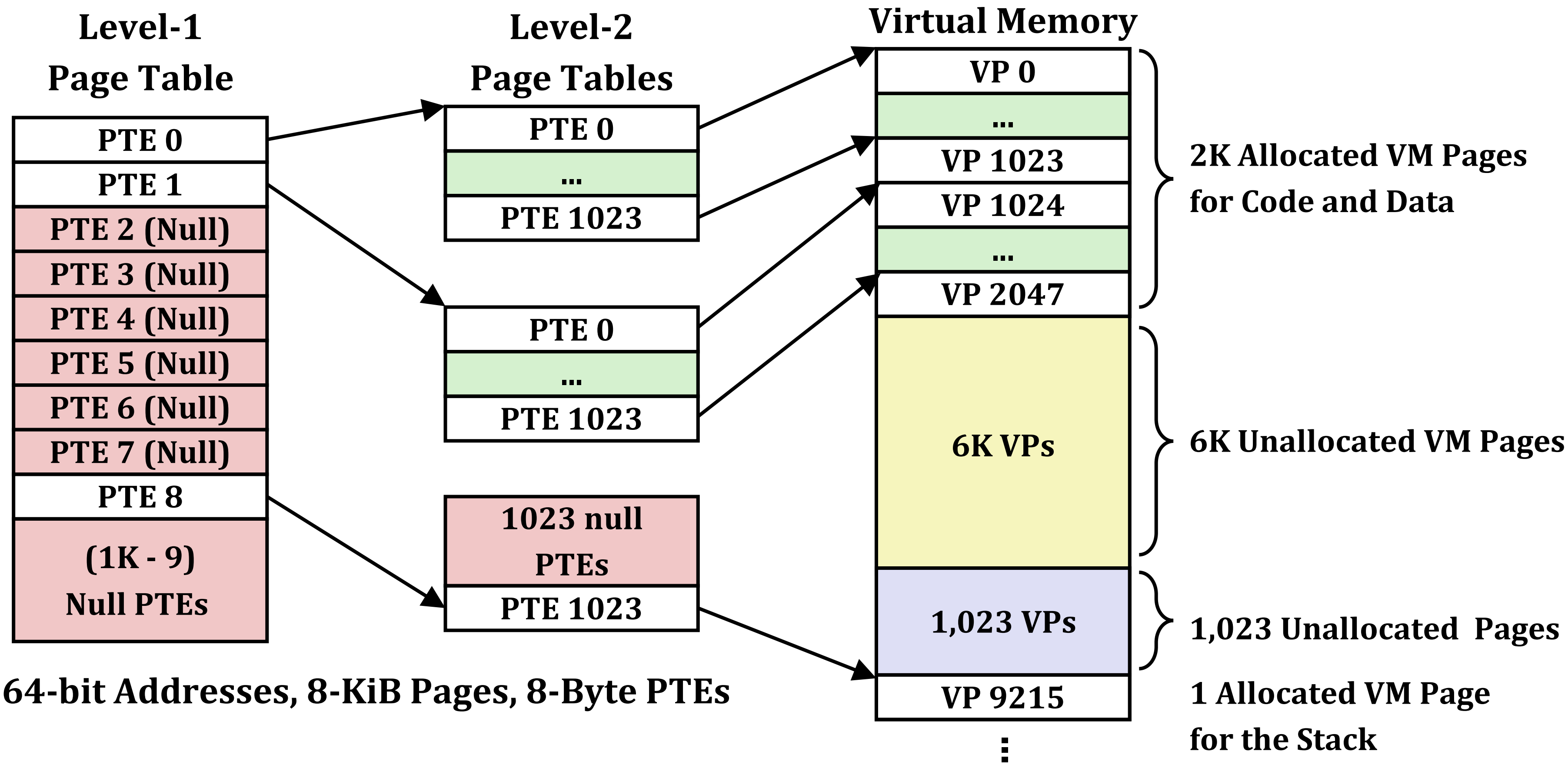
- Problem: Would need a 512 GB page table!
    - $2^{48} \times 2^{-12} \times 2^{3} = 2^{39}$ bytes

- Common solution: Multi-level page tables

- Example: 2-level page table
    - Level-1 table: each PTE points to a page table, i.e., a page that stores part of Level-2 page table (always memory resident)
    - Level-2 table: each PTE points to a page (paged in and out like any other data)

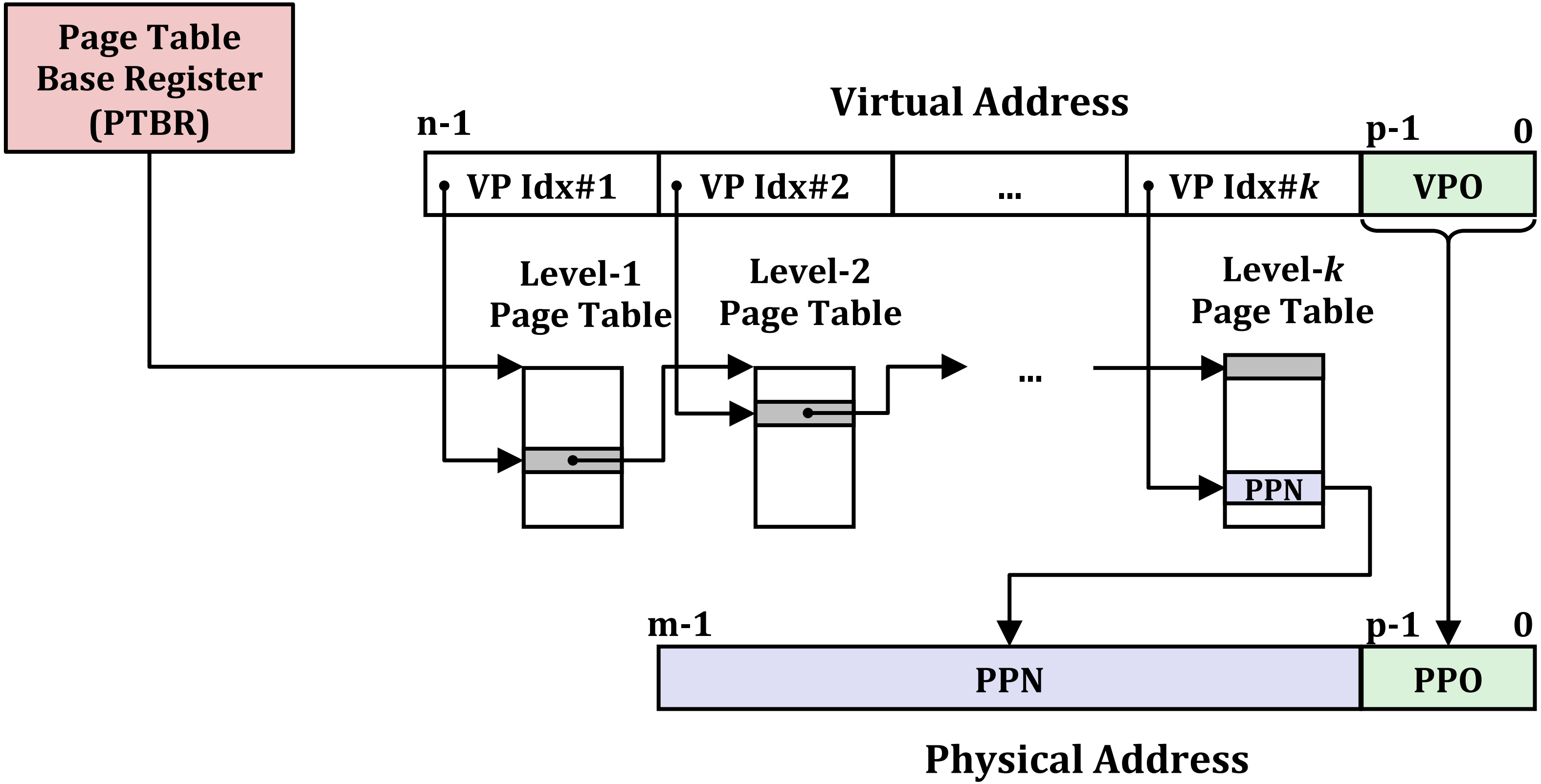


A Two-Level Page Table Hierarchy



64-bit Addresses, 8-KiB Pages, 8-Byte PTEs

Translating with a *k*-Level Page Table



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 inter-positioning point to check permissions

[CSED211] Introduction to Computer Software Systems

Lecture 15: Virtual Memory - Concepts

Prof. Jisung Park

POHANG UNIVERSITY OF SCIENCE AND TECHNOLOGY 1986 포항공과대학교

CAOS
COMPUTER ARCHITECTURE &
OPERATING SYSTEMS LABORATORY

2023.12.06