CS3281 / CS5281

Virtual Memory

Will Hedgecock
Sandeep Neema
Bryan Ward

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

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

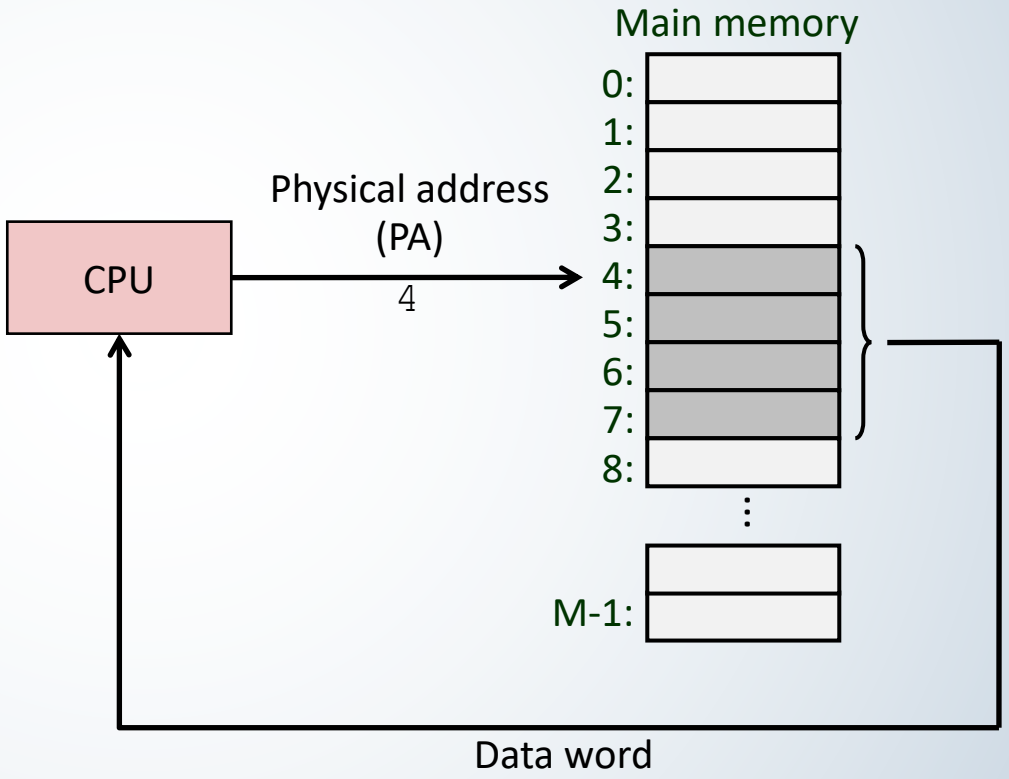
VANDERBILT UNIVERSITY

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

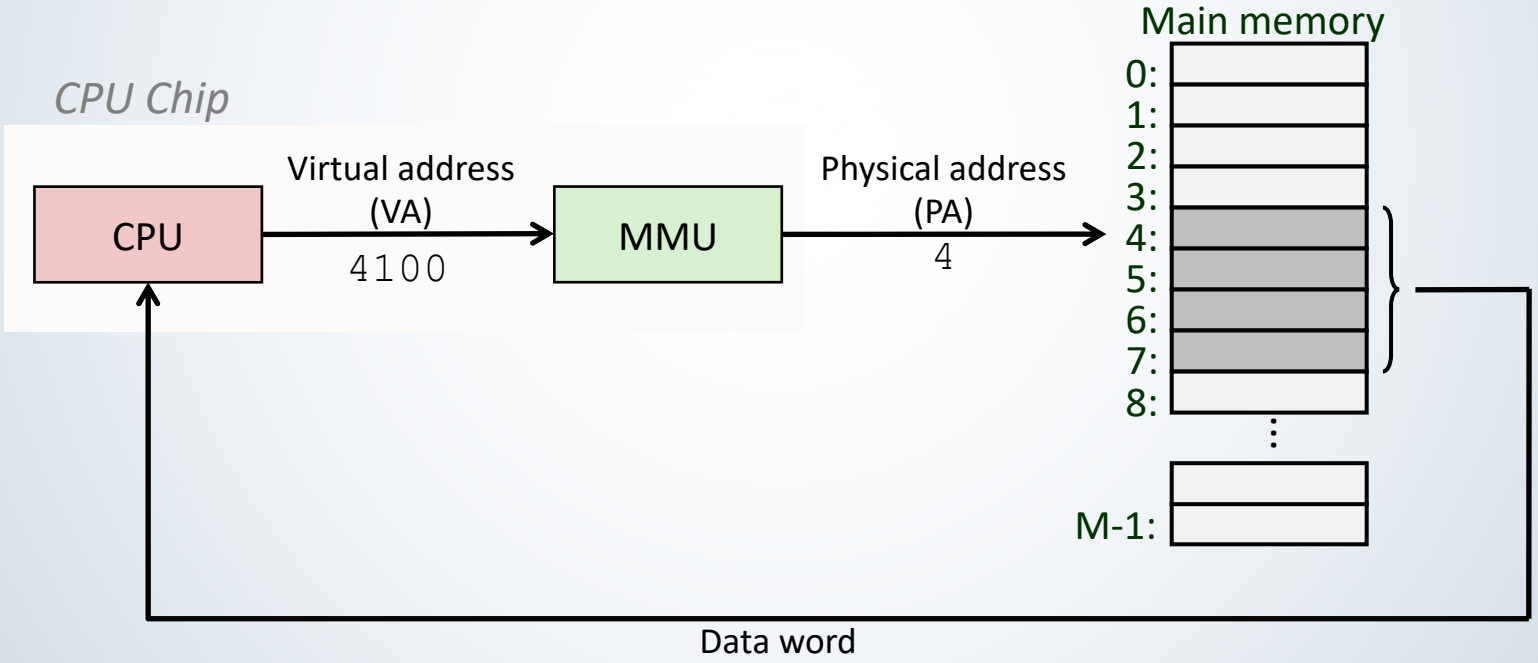
A System Using Physical Addressing

- 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, 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}

- Physical address space: Set of $M = 2^m$ physical addresses
  {0, 1, 2, 3, …, M-1}

Why Virtual Memory (VM)?

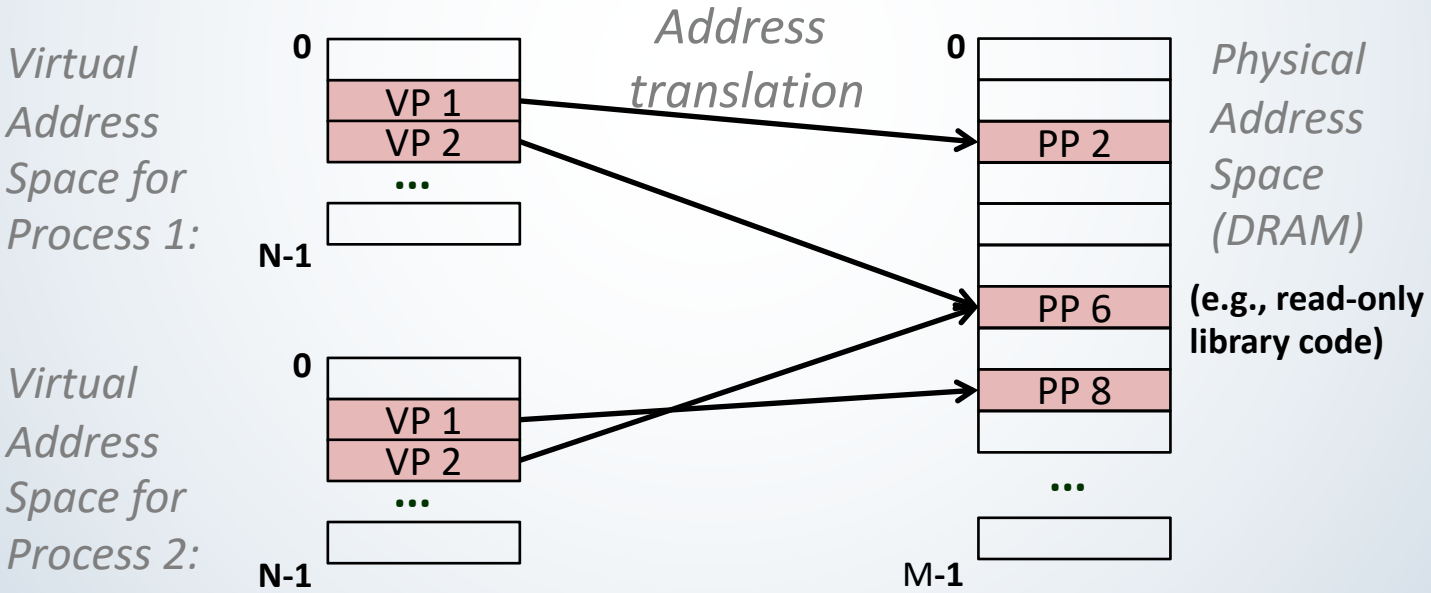
- Simplifies memory management
  - Each process gets the same uniform linear address space
- Isolates address spaces
  - One process can't interfere with another's memory
  - User program cannot access privileged kernel information and code
- Uses main memory (RAM) efficiently
  - Use DRAM as a cache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

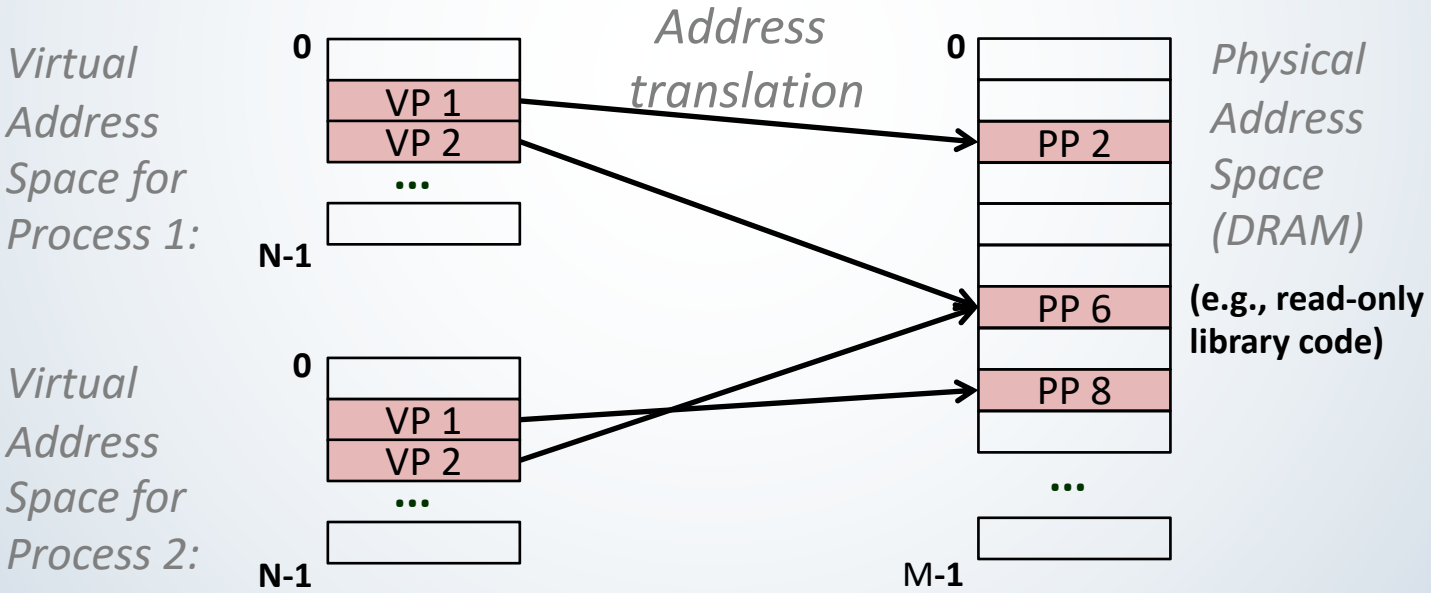
VM as a Tool for Memory Management

- **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
    - Well-chosen mappings can improve locality



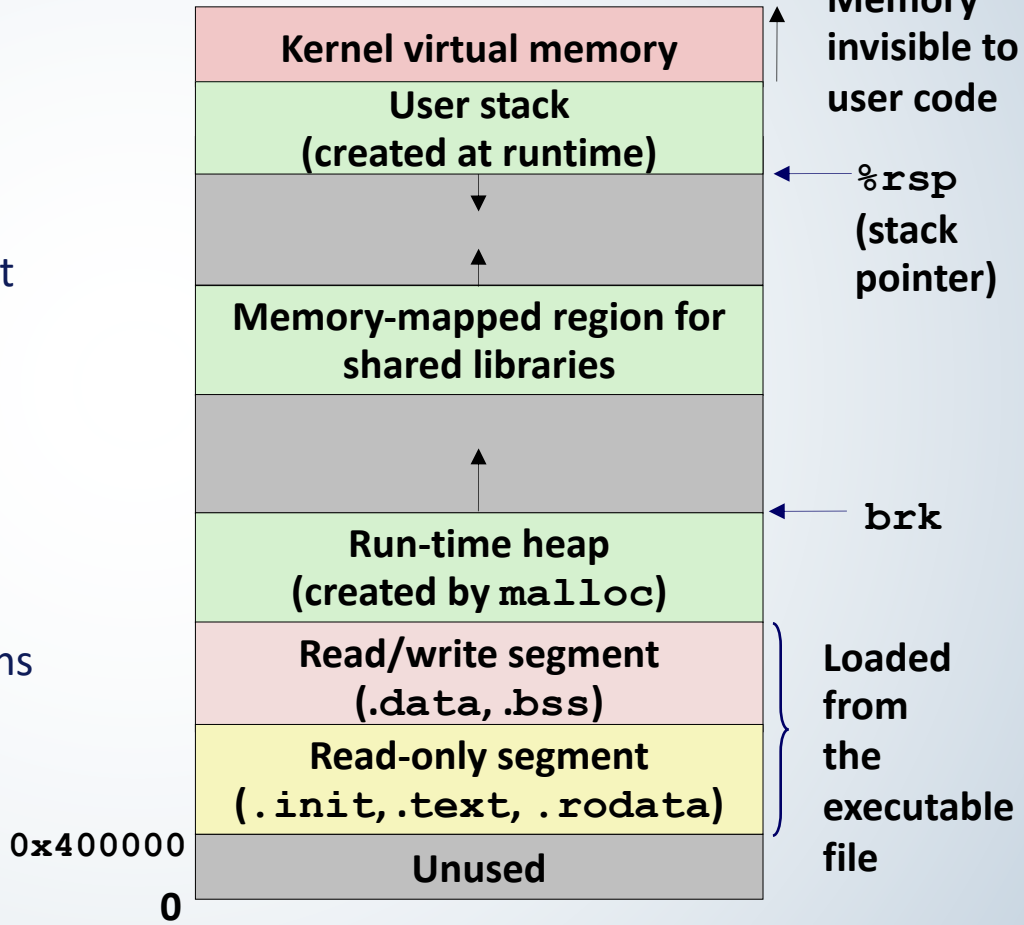
VM as a Tool for Memory Management

- **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)



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

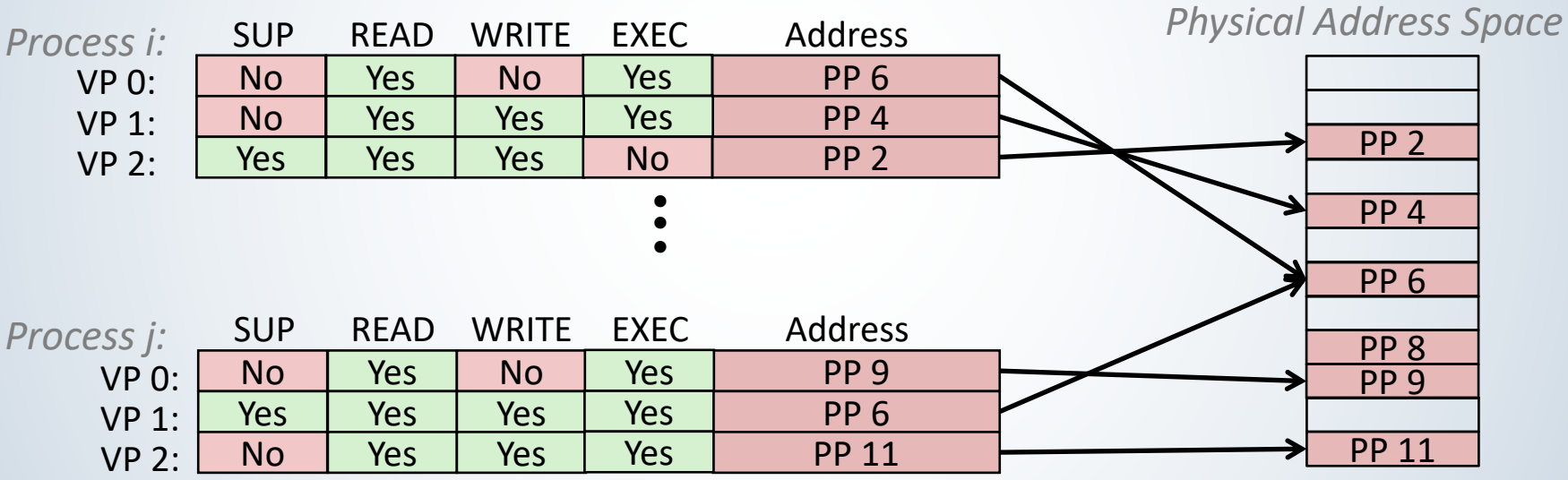


Today

- 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 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

Memory Hierarchy

"Ideally one would desire an indefinitely large memory capacity such that any particular 40 binary digit number or word would be immediately – i.e., on the order of 1 to 100us – available and that words could be replaced with new words at about the same rate. It does not seem possible physically to achieve such a capacity. We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding, but which is less quickly accessible."

Preliminary discussion of the logical design of
an electronic computing instrument

Burks, Goldstine, von Neuman 1946

Latency Numbers Every Programmer Should Know

2020

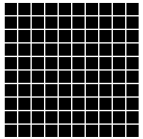

1ns


L1 cache reference: 1ns


Branch mispredict: 3ns


L2 cache reference: 4ns

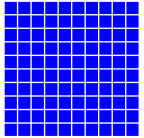

Mutex lock/unlock: 17ns


100ns = 


Main memory reference: 100ns

1,000ns ≈ 1µs

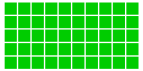

Compress 1KB wth Zippy: 2,000ns ≈ 2µs

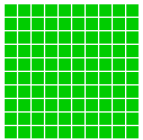

10,000ns ≈ 10µs = 

Send 2,000 bytes over commodity network: 16ns

SSD random read: 16,000ns ≈ 16µs

Read 1,000,000 bytes sequentially from memory: 1,000ns ≈ 1µs

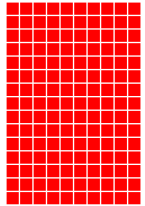

Round trip in same datacenter: 500,000ns ≈ 500µs


1,000,000ns = 1ms =

Read 1,000,000 bytes sequentially from SSD: 24,000ns ≈ 24µs

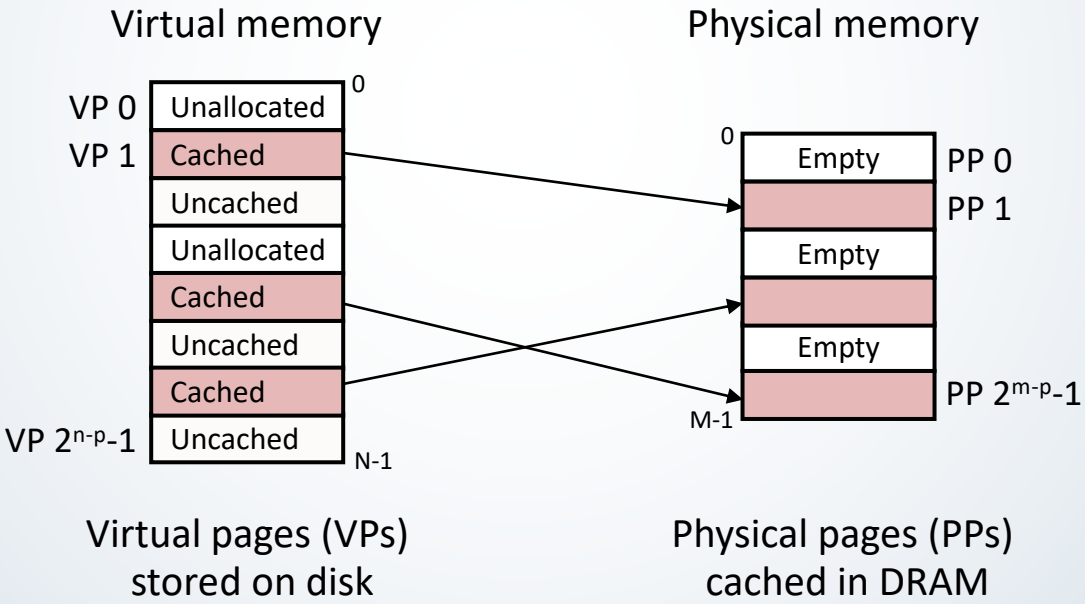

Disk seek: 2,000,000ns ≈ 2ms

Read 1,000,000 bytes sequentially from disk: 544,000ns ≈ 544µs


Packet roundtrip CA to Netherlands: 150,000,000ns ≈ 150ms

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)



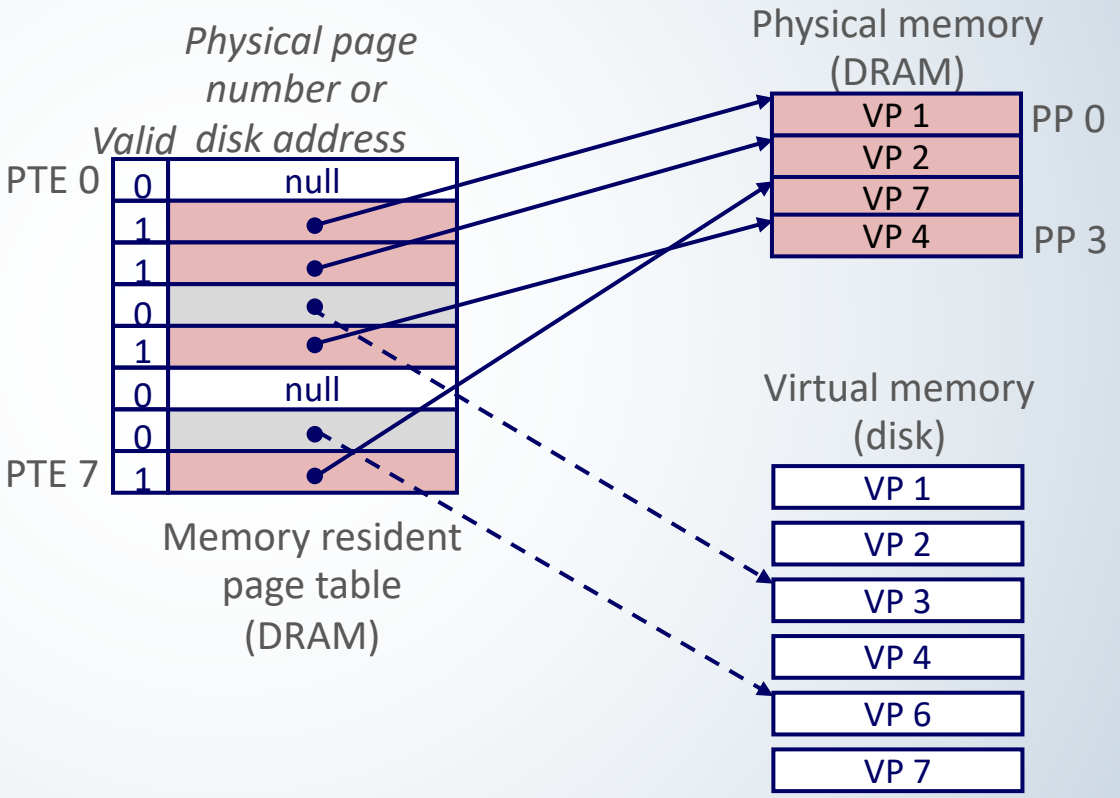
DRAM Cache Organization

- 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 KB (Huge pages are 2MB – 1GB.)
  - Fully associative
    - Any VP can be placed in any PP
    - 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 rather than write-through

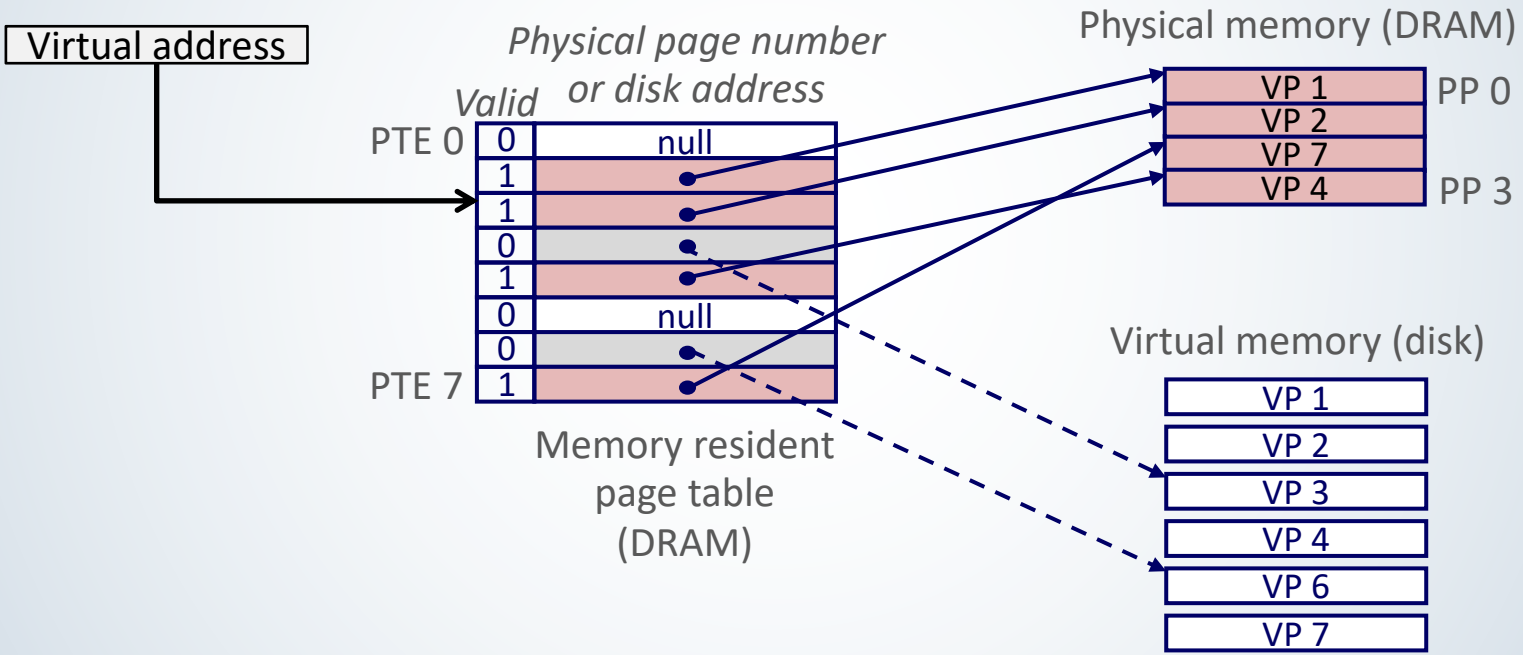
Enabling Data Structure: Page Table

- 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



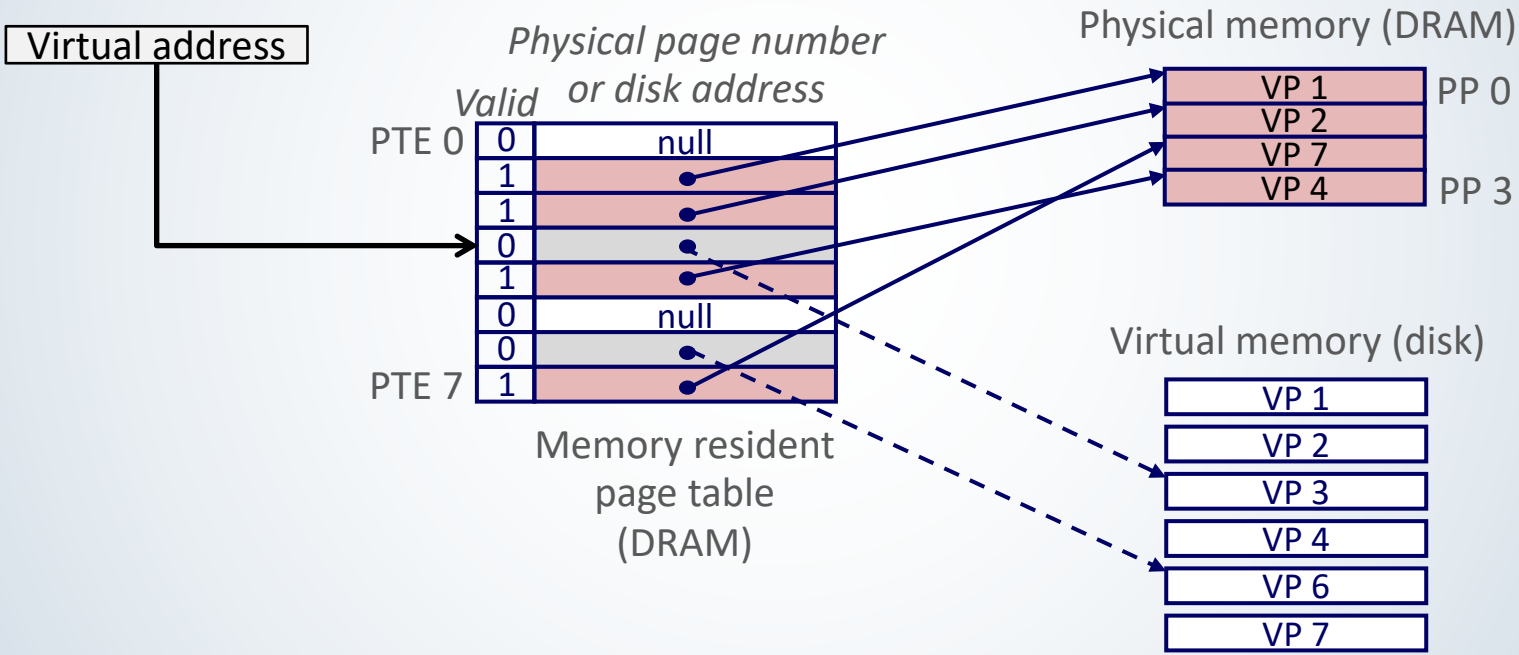
Page Hit

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



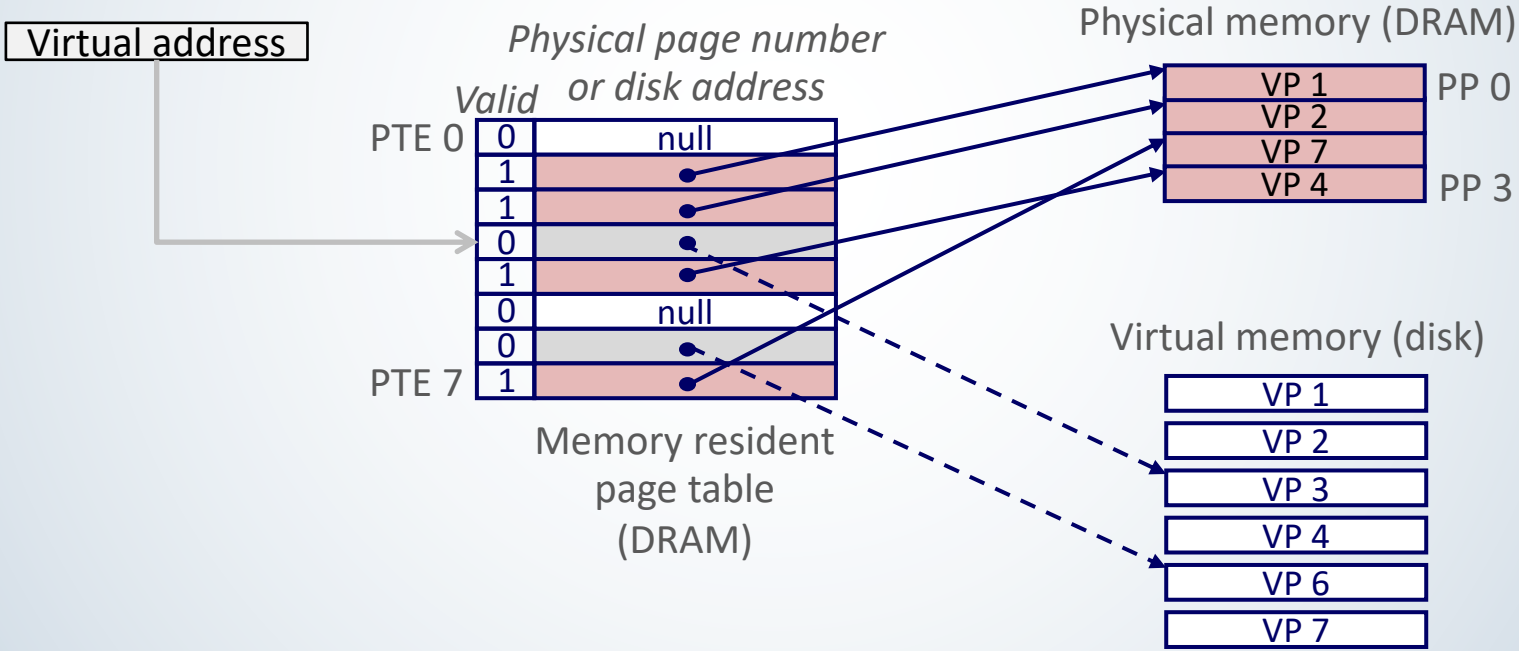
Page Fault

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



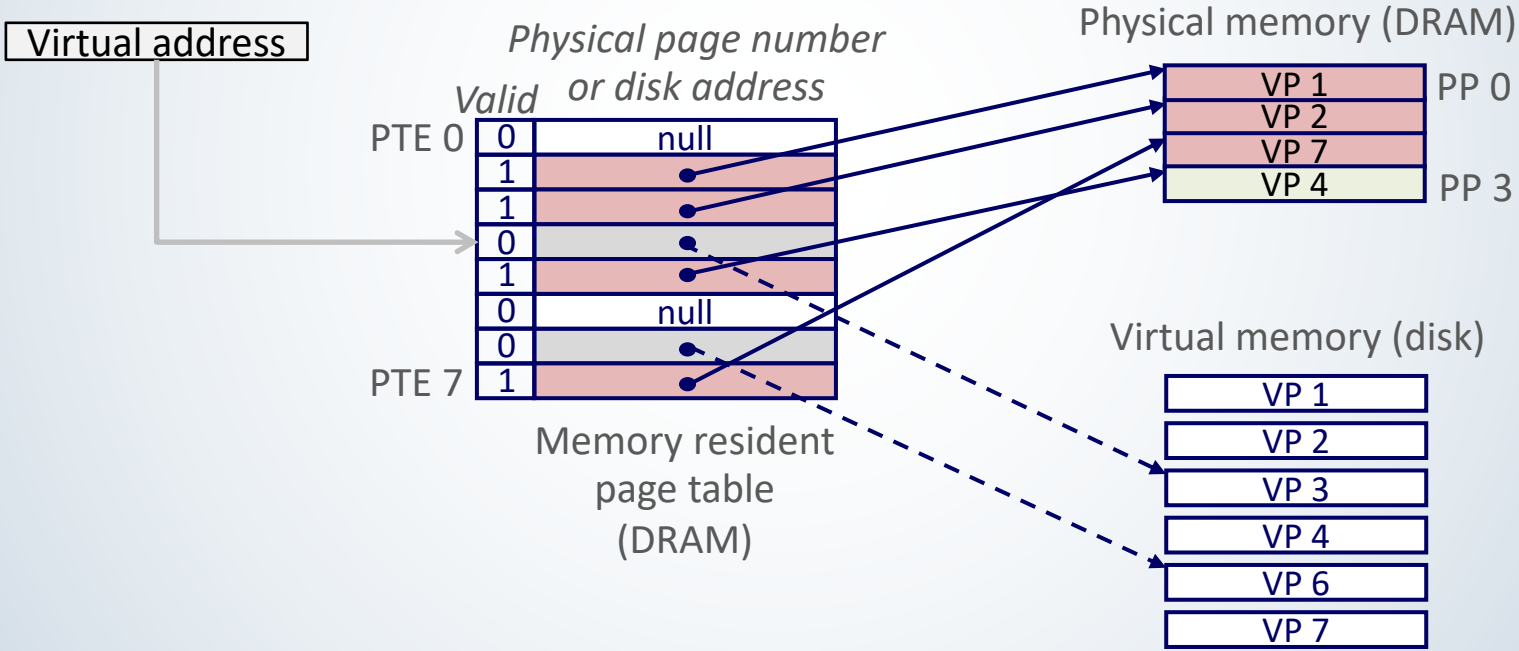
Handling Page Fault

- Page miss causes page fault (an exception)



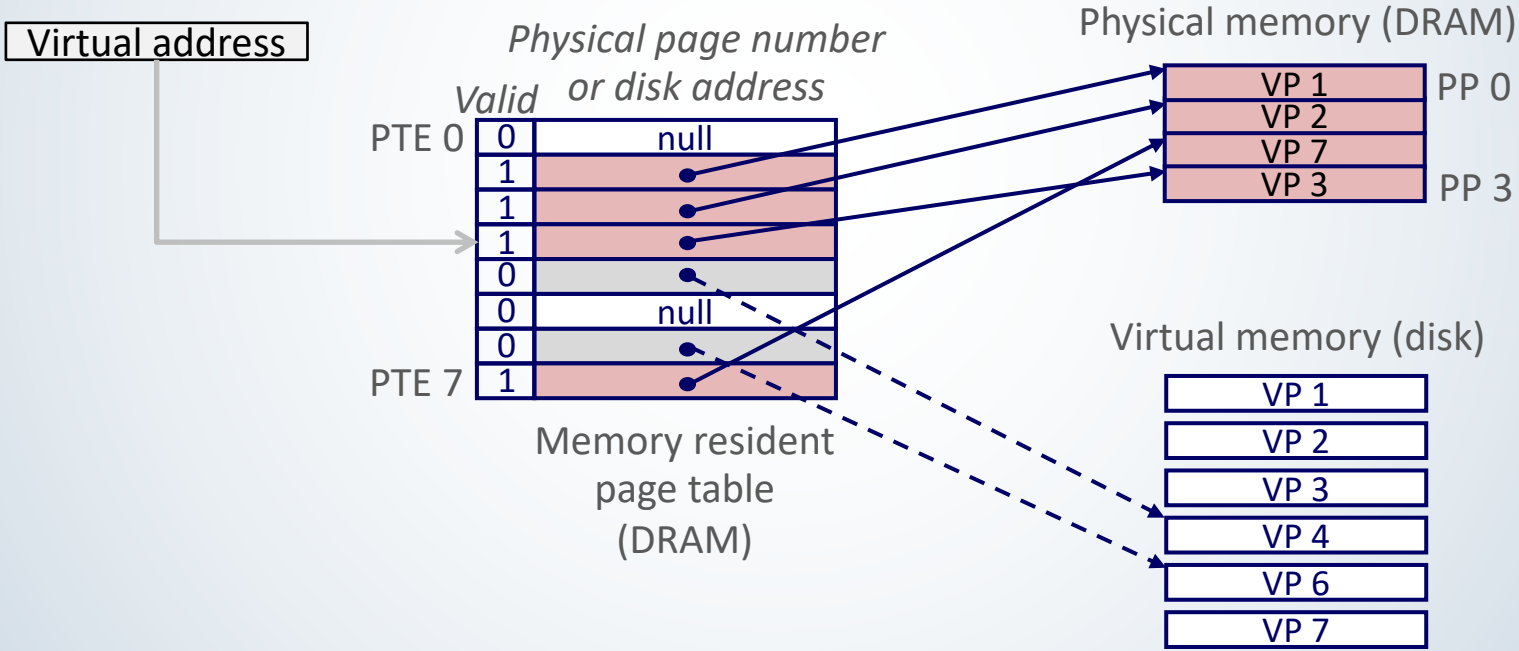
Handling Page Fault

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



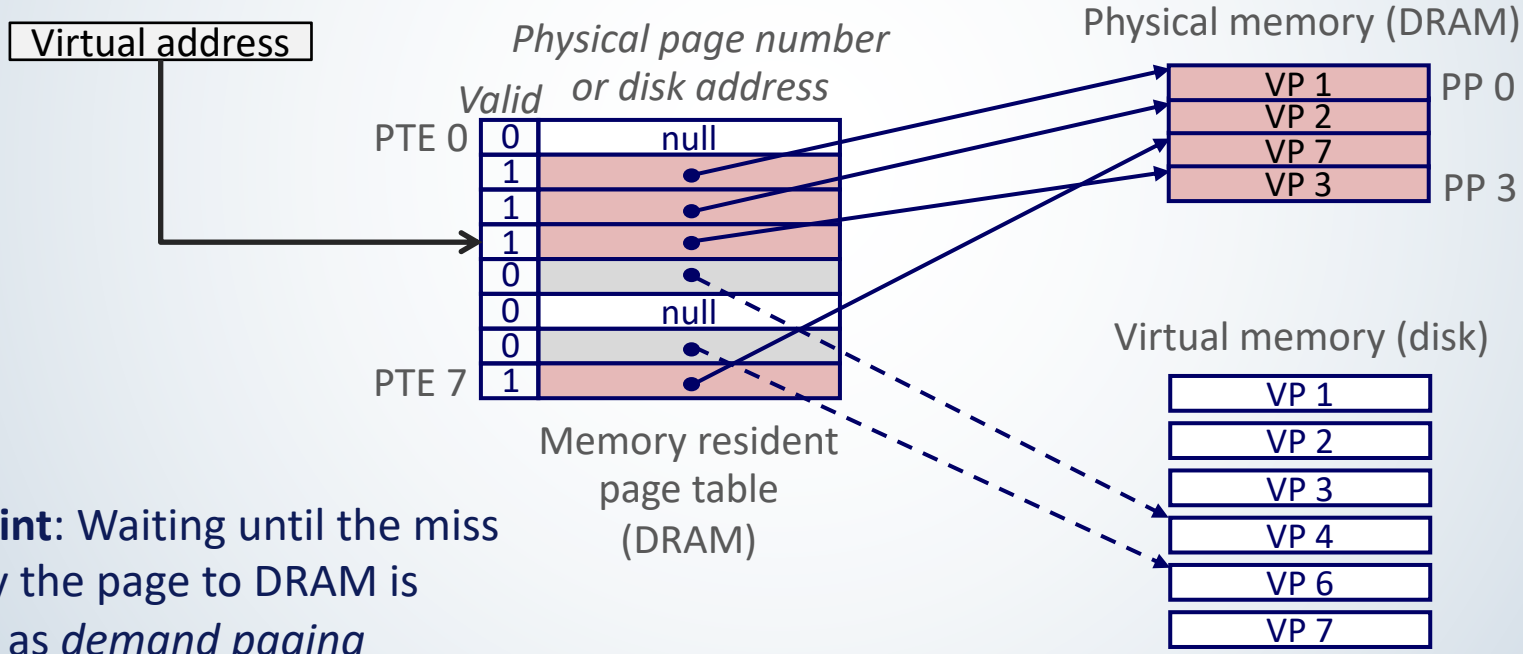
Handling Page Fault

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



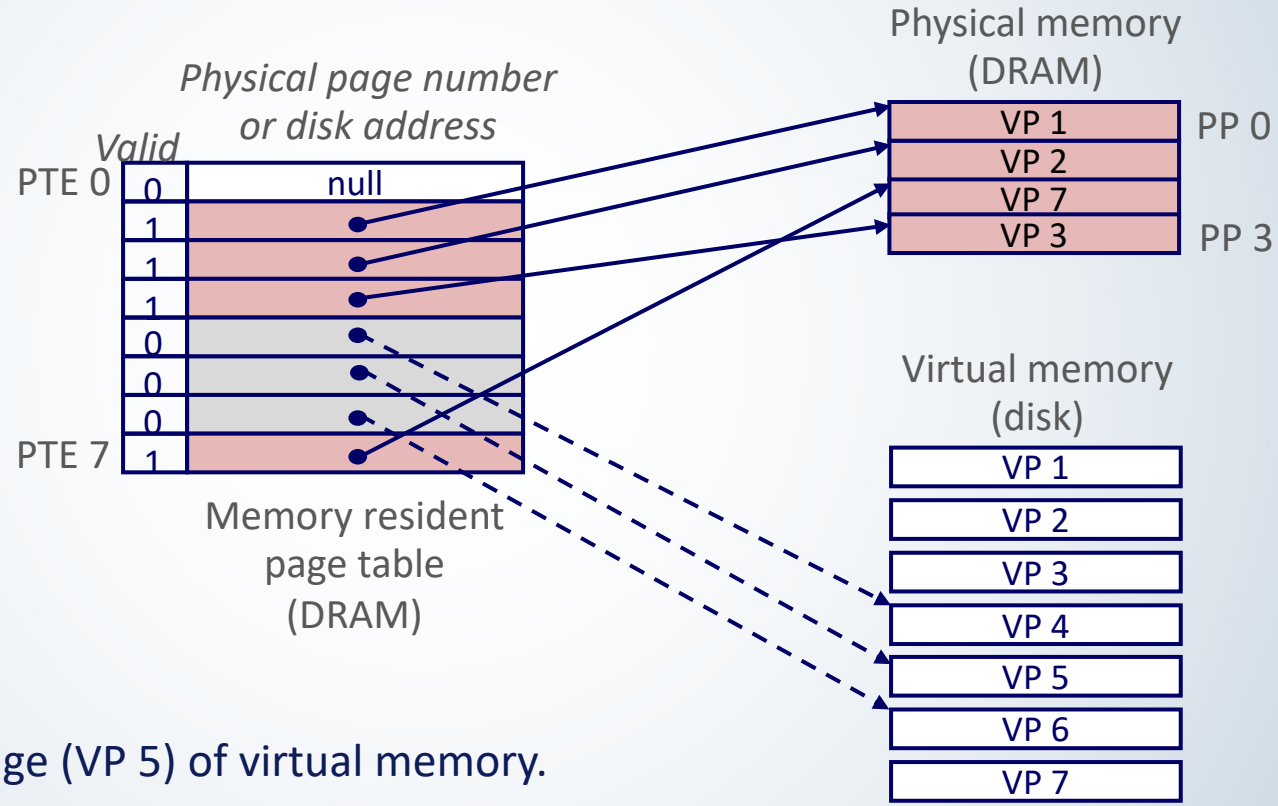
Handling Page Fault

- 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!



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

Allocating Pages



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

Locality to the Rescue

- 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
- 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

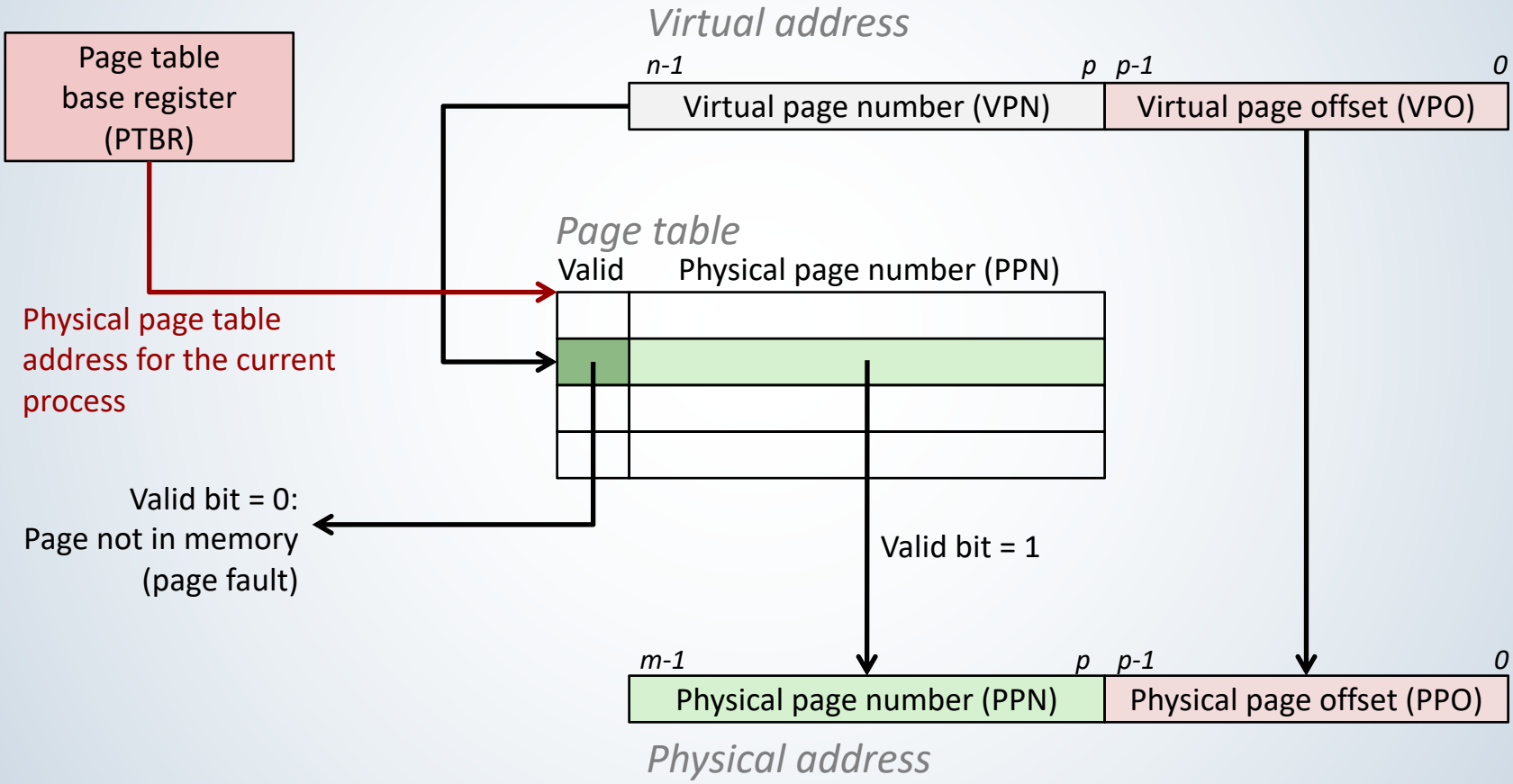
VM Address Translation

- Virtual Address Space
  - *V = {0, 1, …, N–1}*
- Physical Address Space
  - *P = {0, 1, …, M–1}*
- Address Translation
  - ***MAP: V → P U {∅}***
  - 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

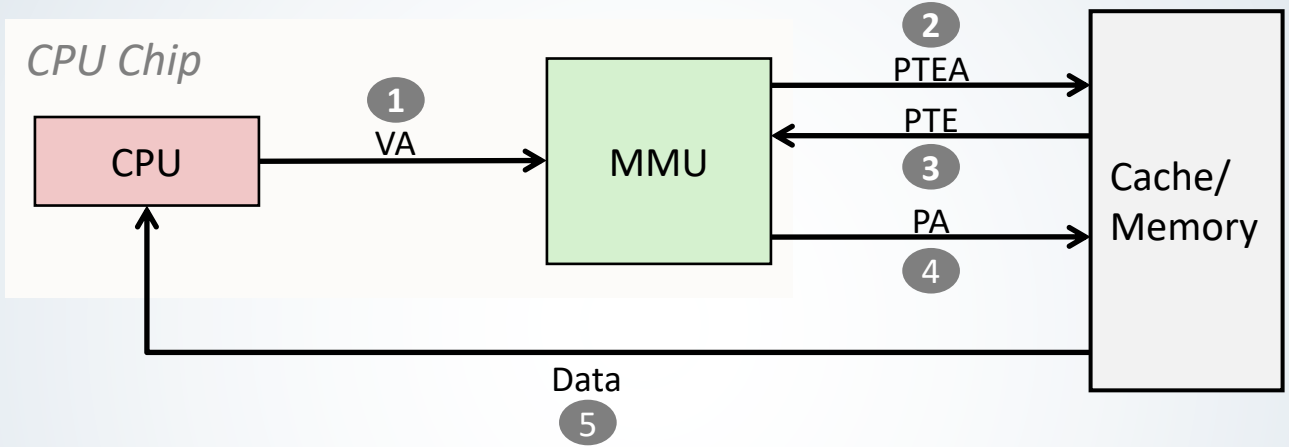
Summary of Address Translation Symbols

- Basic Parameters
  - **N = 2^n** : Number of addresses in virtual address space
  - **M = 2^m** : Number of addresses in physical address space
  - **P = 2^p** : 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

Address Translation with a Page Table

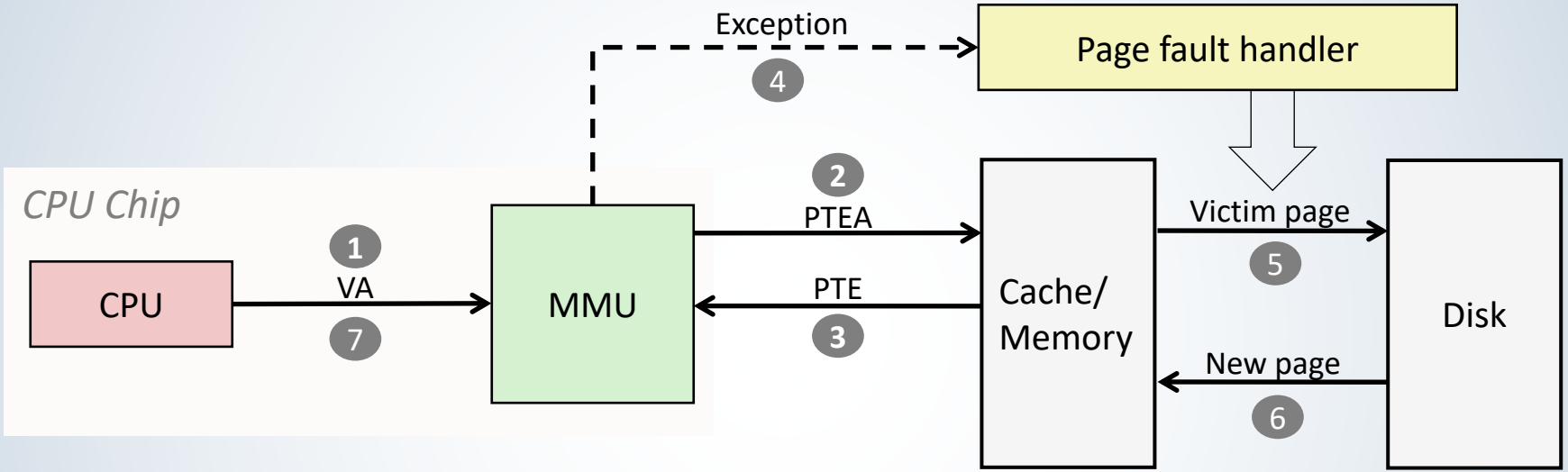


Address Translation: Page Hit



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

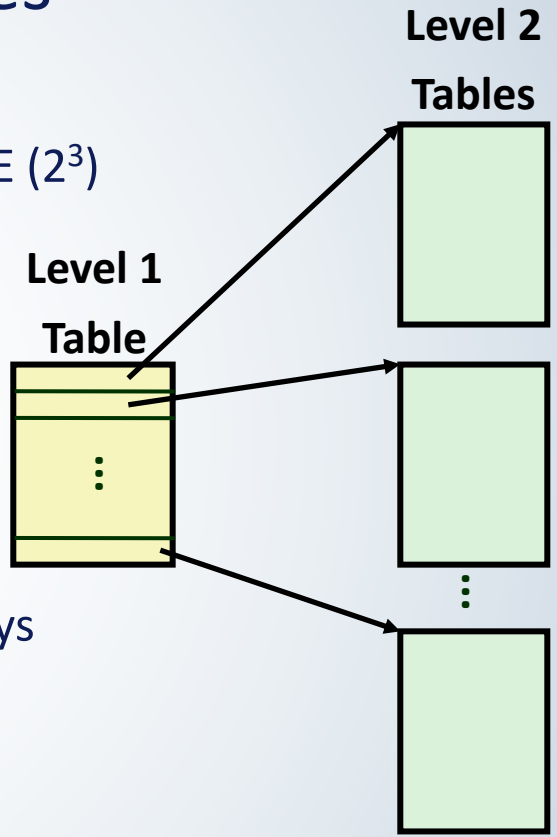
Address Translation: Page Fault



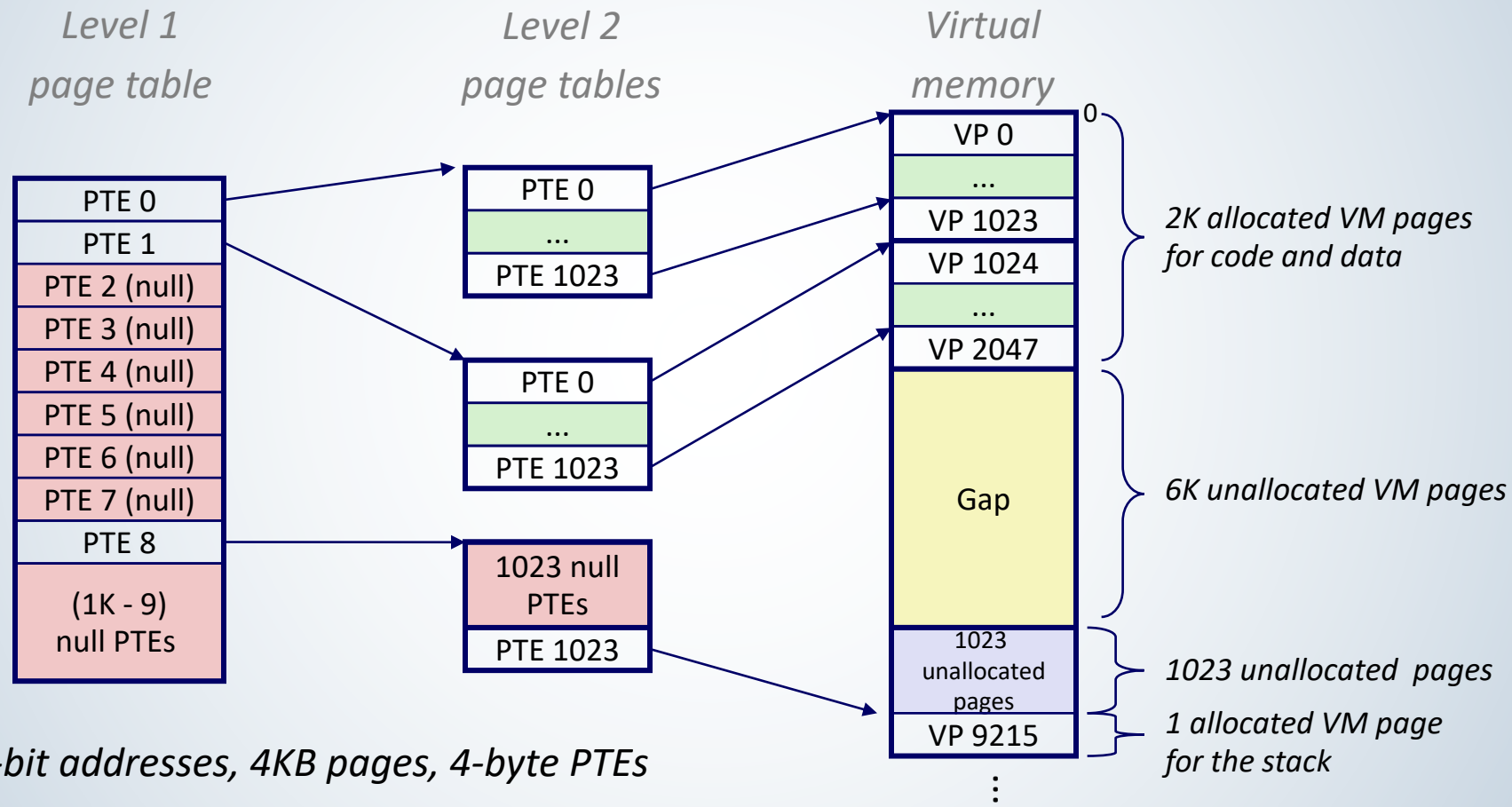
1) Processor sends virtual address to MMU
2-3) MMU fetches PTE from page table in memory
4) Valid bit is zero, so MMU triggers page fault exception
5) Handler identifies victim (and, if dirty, pages it out to disk)
6) Handler pages in new page and updates PTE in memory
7) Handler returns to original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

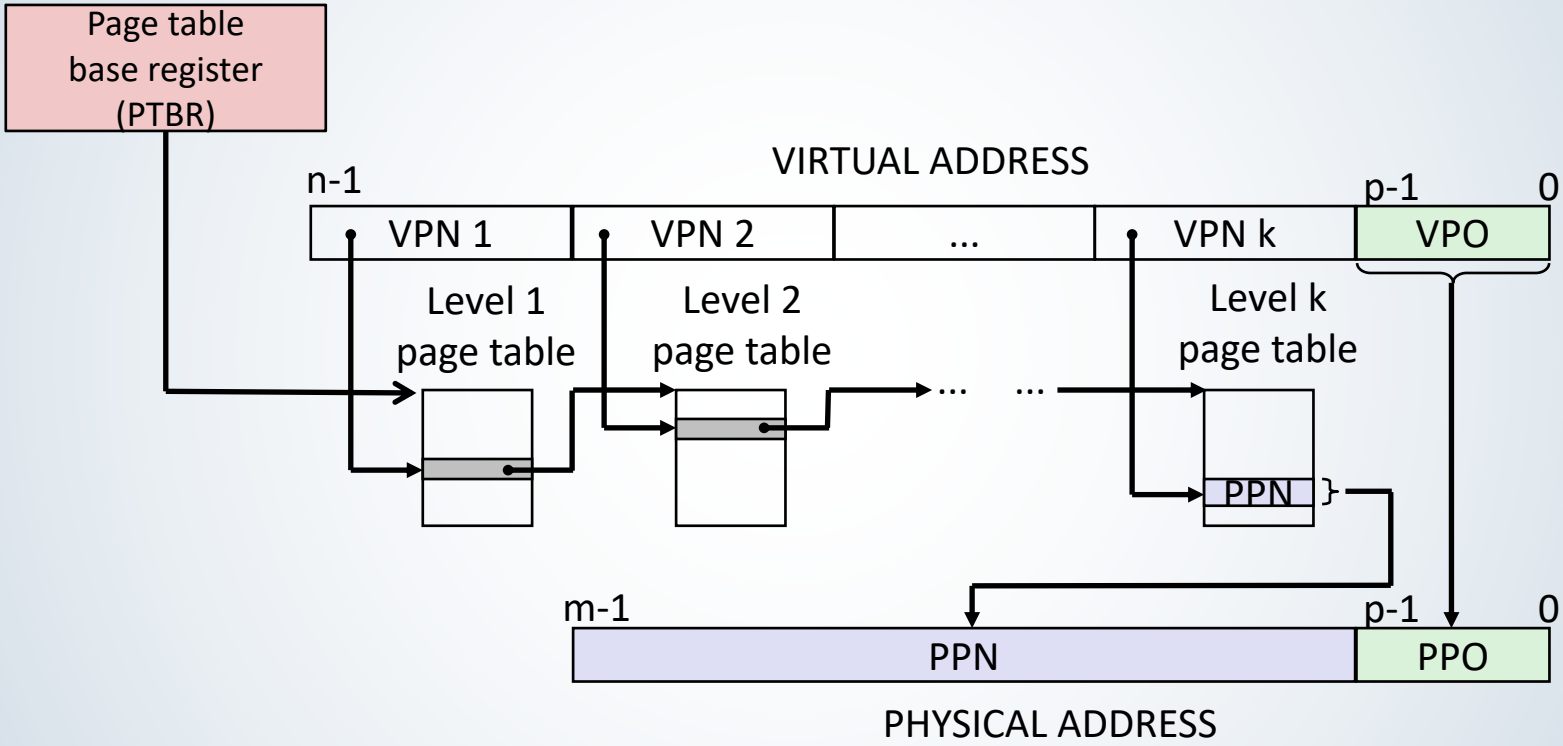


Two-Level Page-Table Hierarchy



32-bit addresses, 4KB pages, 4-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 point to check permissions