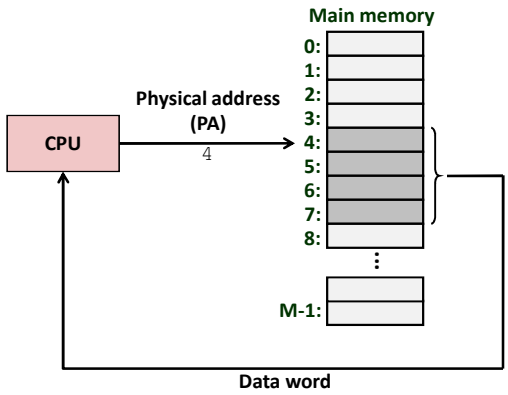
Virtual Memory

adapted for CS367@GMU

Today

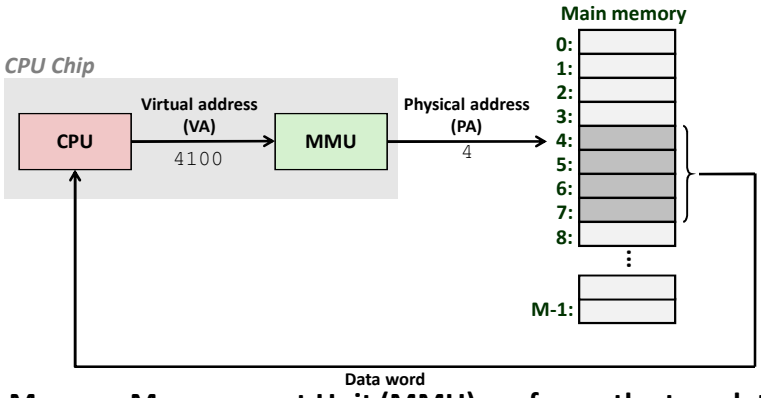
- **Basic Concepts & Address Spaces**
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

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



- Memory Management Unit (MMU) performs the translation
- 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
  V = {0, 1, 2, 3, …, N-1}
- **Physical address space:** Set of M = 2^m physical addresses
  P = {0, 1, 2, 3, …, M-1}
- **Clean distinction between data (bytes) and their attributes (addresses)**
- **Every byte in main memory has one physical address and zero or more virtual addresses**

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition 5

Address Spaces

- **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 invalid

Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition 6

Why Virtual Memory (VM)?

- **Uses main memory efficiently**
  - Avoids ineffective use of DRAM (main memory) by reducing "fragmentation"
  - Uses DRAM as a cache for parts of a virtual address space
- **Simplifies memory management**
  - Each process gets the same uniform linear address space
  - Simplifies linking and loading
- **Isolates address spaces**
  - One process can't interfere with another's memory
  - User program cannot access privileged kernel information and code

Memory at Run-time

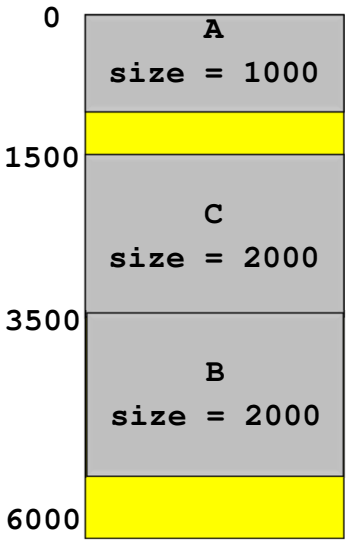
- **Each *process* (running program) has a private and distinct address space**
- **There are (typically) many different active processes sharing runtime memory.**
- **Need a way to enable this sharing in a way that is**
  - Efficient – have to be able to quickly map from the process's address (virtual address) to the physical (real) address.
  - Safe – processes should not be able to interfere with each other.

Contiguous Memory Allocation

A process can run if there is enough contiguous space in memory to hold its entire process space

Address Translation – PA = VA + offset (where the process space starts in the memory)

Fragmentation – big problem! Process can't start to run because there is no large enough contiguous free region (consider Process D of size 800)

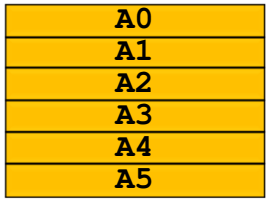


Paged Memory

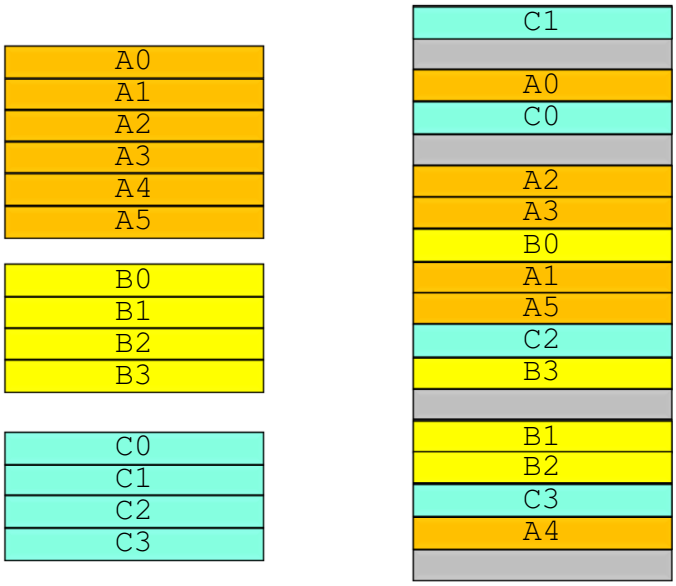
Divide physical address space in the main memory into equal-size chunks of size 2^n (called Physical Pages)

Divide the virtual address space of the process into equal-size chunks of size 2^n

(called Virtual Pages)



Paged Memory (Multiple Running Processes)

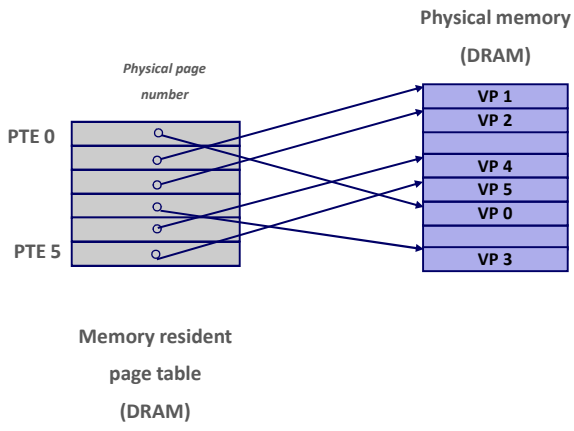


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition 11

Address Translation: Page Tables

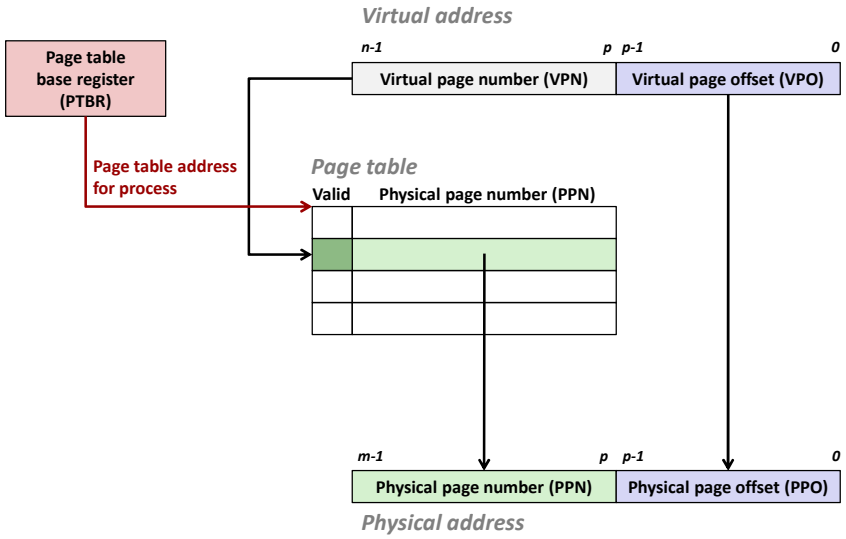
A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.

- Per-process data structure in DRAM (managed by the operating system)
- Dedicated register keeps address of page table in DRAM



Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition 12

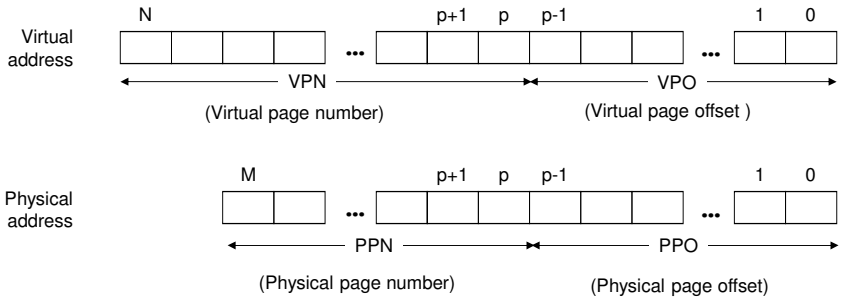
Address Translation With a Page Table



Address Translation

Size of VPN
– dependent on number of pages for the process

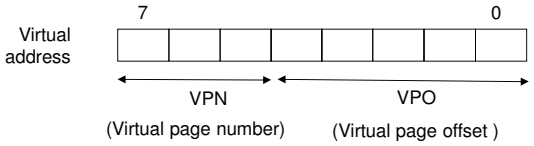
Size of VPO = size of PPO
– dependent on physical page size



Size of PPN
– dependent number of physical pages in DRAM

Example:

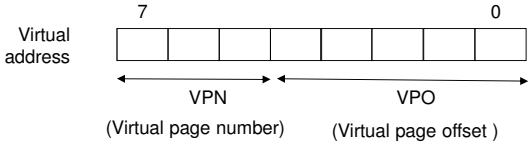
8-bit virtual address with a 3-bit VPN …



What is the page size? $2^p = 2^5 = 32$

How many pages can we have? Up to $2^3 = 8$

Example:



Page Table:

000	0111100
001	1101100
010	0110101
011	1000100
100	0101110
101	0110000

How many bits in the physical address?

$7_{ppn} + 5_{ppo} = 12$ bits

What is the physical address for:

- **virtual address 01011011?**
  0110101 11011
- **virtual address 10110001?**
  0110000 10001

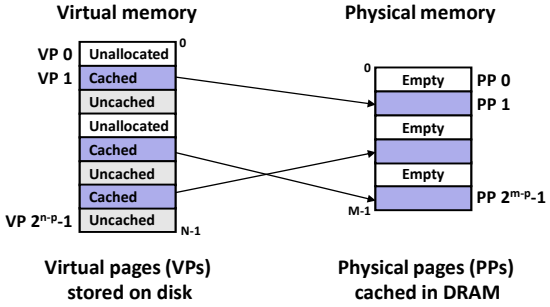
6 entries <= 8 maximum: OK

Today

- Address spaces & Basic Concepts
- **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, we can view the total *virtual memory* as an array of N contiguous bytes stored on disk at all times.**
- **The contents of the array on disk are cached in *physical memory* (which can be called the *DRAM cache*)**
  - These cache blocks in this case correspond to *pages* (size is $P = 2^p$ bytes)



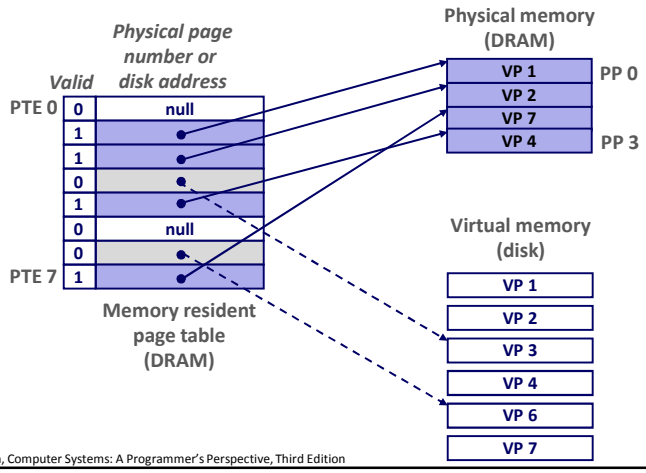
DRAM Cache Organization

- **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

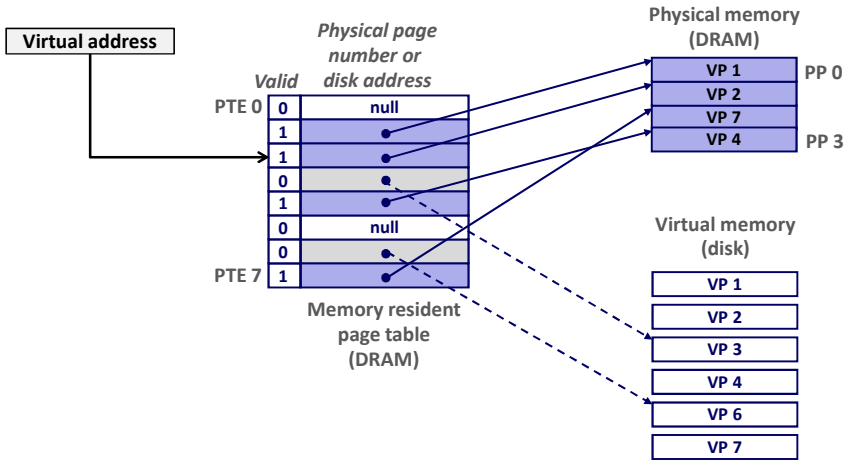
Extending Page Table Structure

- **Page table entries (PTEs) may contain the disk address if VP is not in the physical memory (DRAM)**
  - Valid bit will be 0 for such entries



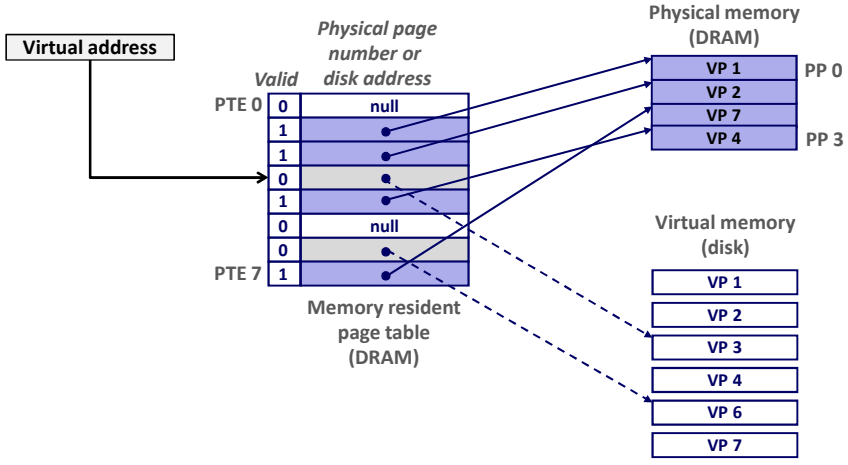
Page Hit

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



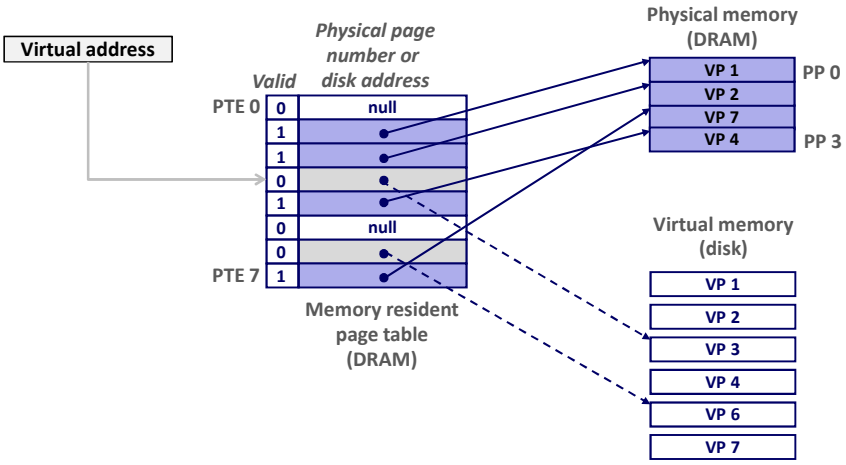
Page Fault

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



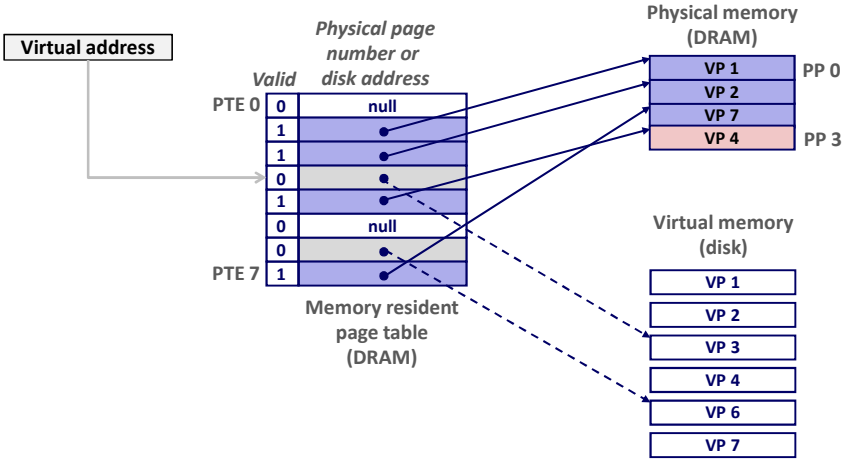
Handling Page Fault

- Page miss causes **page fault** (an "exception" -- a transfer of control to the Operating System in response to some *event*)



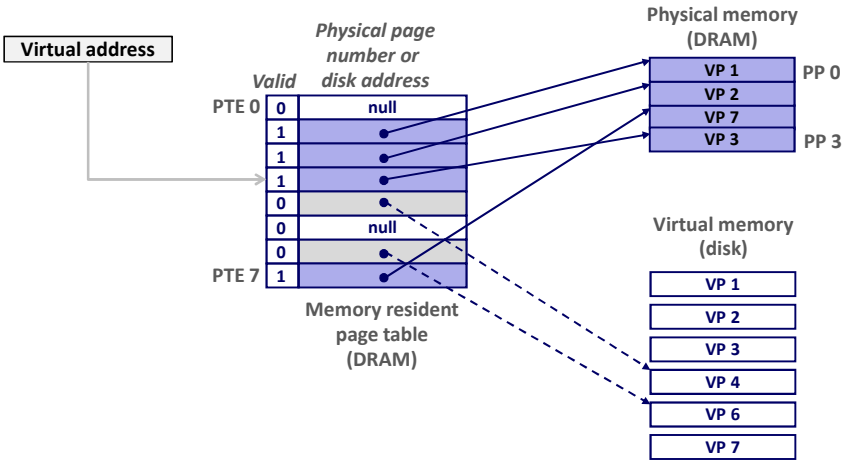
Handling Page Fault

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



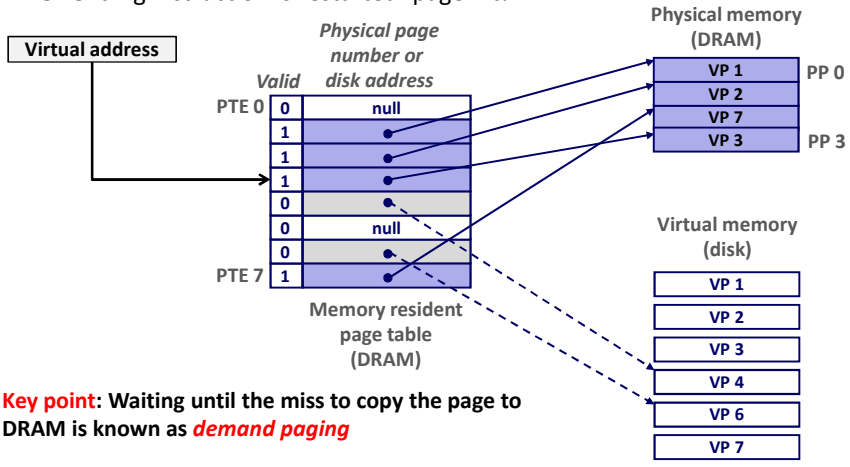
Handling Page Fault

- 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!



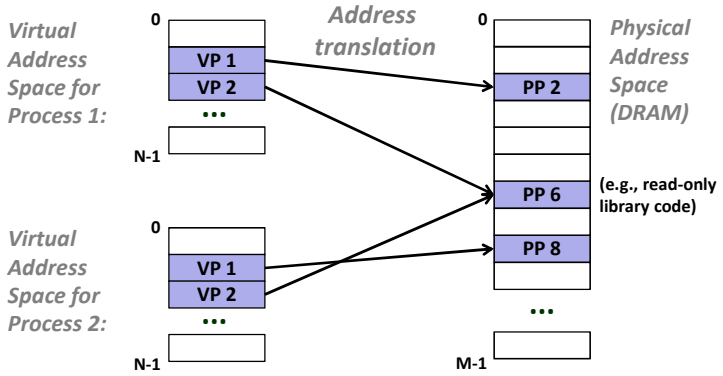
Key point: Waiting until the miss to copy the page to DRAM is known as *demand paging*

Today

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

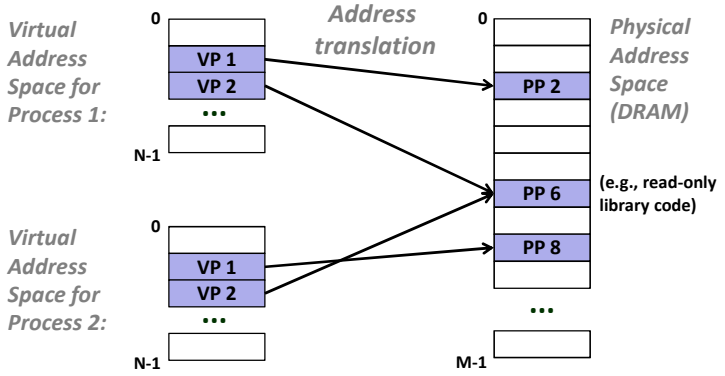
VM as a Tool for Memory Management

- **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 can improve locality



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)

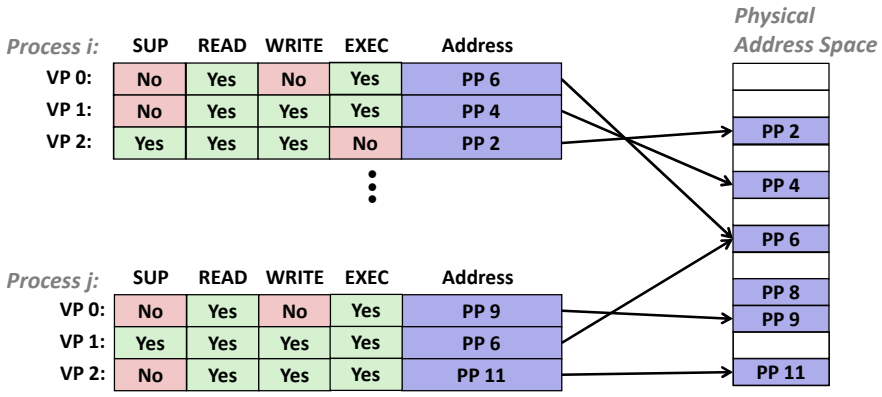


Today

- Address spaces & Basic Concepts
- 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, "segmentation fault" occurs!



SUP: supervisor (kernel) mode

Today

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

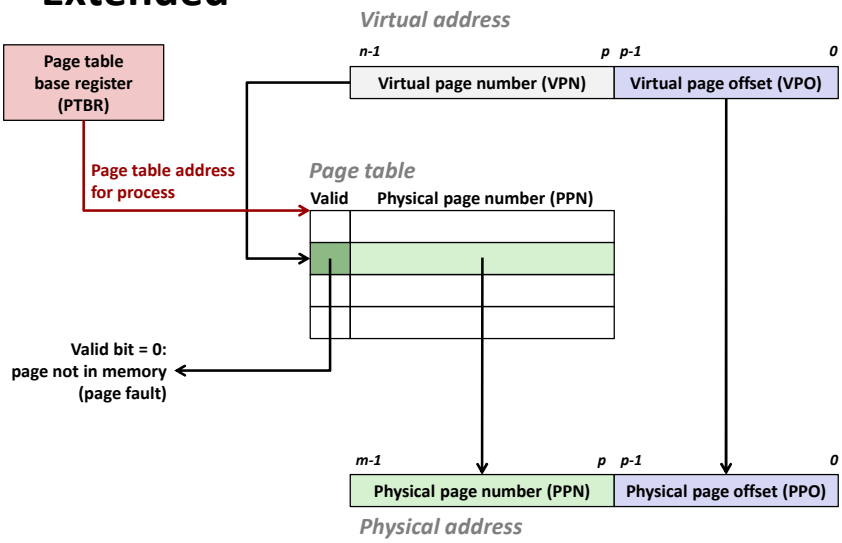
VM Address Translation - Extended

- **Virtual Address Space**
  - $V = \{0, 1, \ldots, N-1\}$
- **Physical Address Space**
  - $P = \{0, 1, \ldots, M-1\}$
- **Address Translation**
  - $MAP: V \rightarrow P \cup \{\varnothing\}$
  - For virtual address a:
    - $MAP(a) = a'$ if data at virtual address a is at physical address a' in P
    - $MAP(a) = \varnothing$ if data at virtual address a is not in physical memory
      - Either invalid or stored on disk

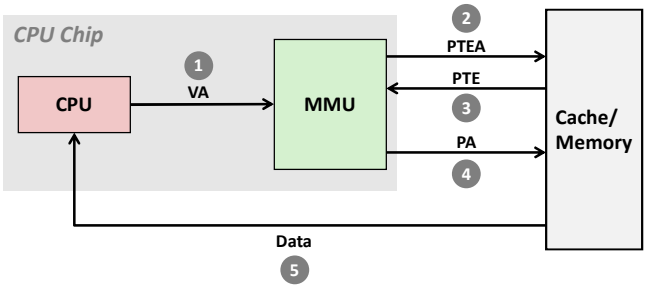
Summary of Address Translation Symbols

- **Basic Parameters**
  - $\mathbf{N = 2^n}$: Number of addresses in virtual address space
  - $\mathbf{M = 2^m}$: Number of addresses in physical address space
  - $\mathbf{P = 2^p}$: 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 - Extended



Address Translation: Page Hit



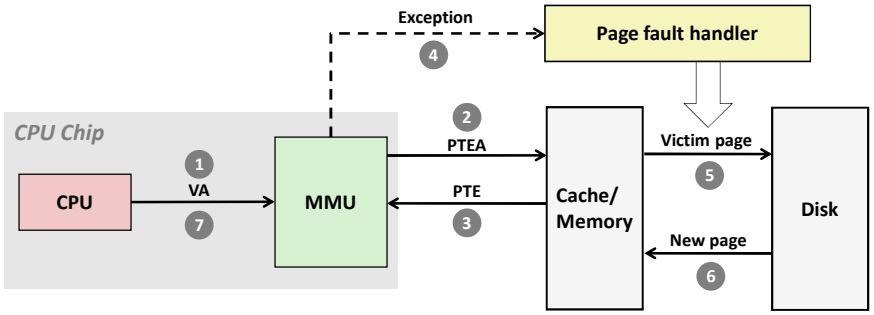
1) Processor sends virtual address to MMU

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

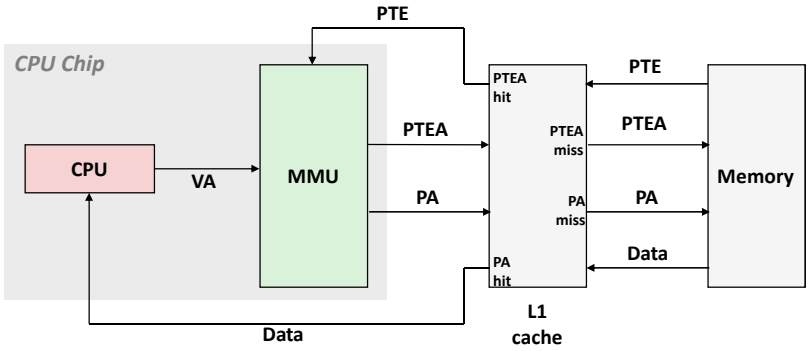
4) Valid bit is zero, so MMU triggers page fault exception

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



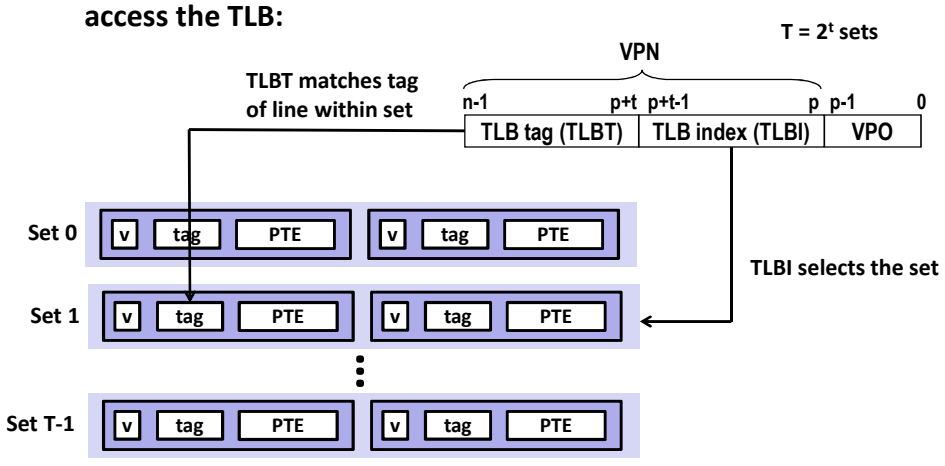
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Speeding up Translation with a TLB

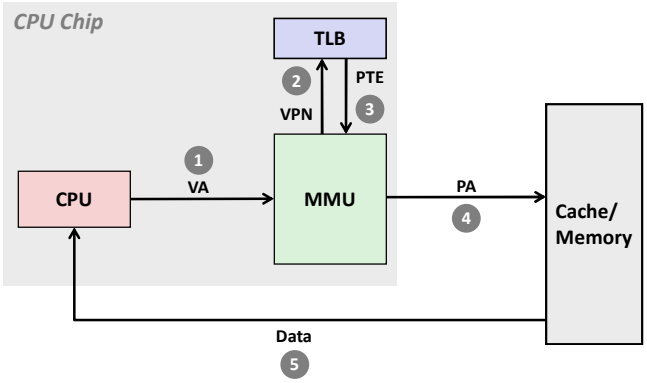
- **Page table entries (PTEs) might be cached in L1 like any other memory word**
  - PTEs may be evicted by other data references
  - PTE hit would still require a small L1 delay
- **Solution: *Translation Lookaside Buffer* (TLB)**
  - Small set-associative hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

Accessing the TLB

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

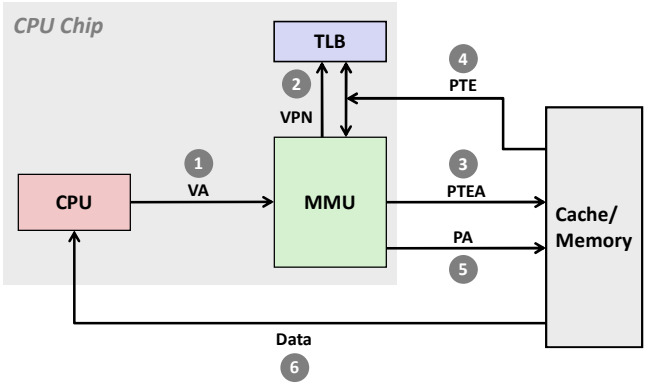


TLB Hit



A TLB hit eliminates a memory access

TLB Miss

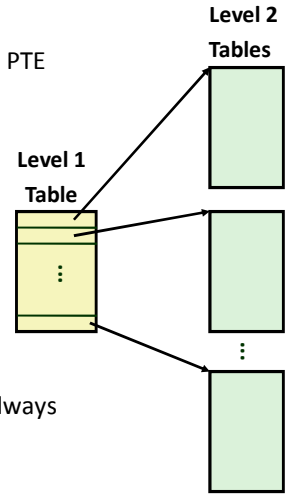


A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare: pages are large.

Multi-Level Page Tables

- **Suppose:**
  - 4KB (2^{12}) page size, 48-bit address space, 8-byte PTE
- **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 page (paged in and out like any other data)



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
  - Simplifies memory management and programming
  - Simplifies protection by providing a convenient interpositioning point to check permissions