

CS 105  
 "Tour of the Black Holes of Computing!"

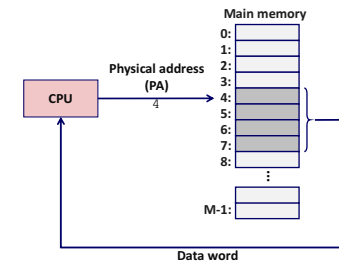
## Virtual Memory

### Topics

- Address translation
- Motivations for VM
- Accelerating translation with TLBs

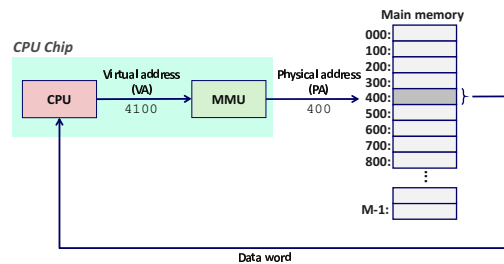


## Physically Addressed System



Used in "simple" systems like embedded microcontrollers in devices such as thermostats, car tires, elevators, digital picture frames

## Virtually Addressed System



Used in all modern servers, laptops, and smart phones  
 One of the great ideas in computer science



## What Is Virtual Memory?

If you think it's there, and it is there...it's *real*.

If you think it's not there, and it really isn't there...it's *nonexistent*.

If you think it's not there, and it really is there...it's *transparent*.

If you think it's there, and it's not really there...it's *imaginary*.

*Virtual memory* is imaginary memory: it gives you the illusion of a memory arrangement that's not physically there.





## 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 (typically has inaccessible "holes")  
{0, 1, 2, 3, ..., N-1}

**Physical address space:** Set of  $M = 2^m$  physical addresses (may also have "holes")  
{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



## Why Virtual Memory (VM)?

**Uses main memory efficiently**  
■ Use DRAM as a cache for parts of a large 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 can't access privileged kernel information and code

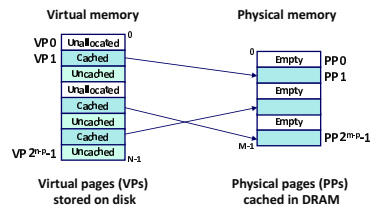


## VM as Tool for Caching

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 (main memory) "cache" organization driven by the enormous miss penalty

- DRAM is about  $10x$  slower than SRAM
- Hard disk is about  $10,000x$  slower than DRAM

### Consequences

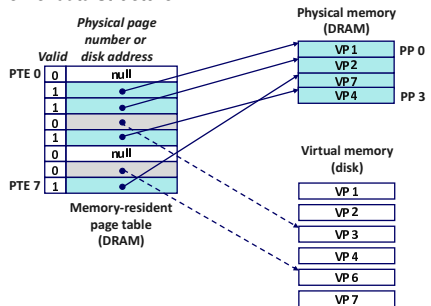
- Large page (block) size: typically 4-8 KB, sometimes 4 MB or more
- Fully associative
  - Any VP can be placed in any PP
  - Requires a "large" mapping function—different from CPU caches
- Highly sophisticated, expensive replacement algorithms
  - Too complicated and open-ended to be implemented in hardware
- Write-back rather than write-through

## Enabling Data Structure: Page Table



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



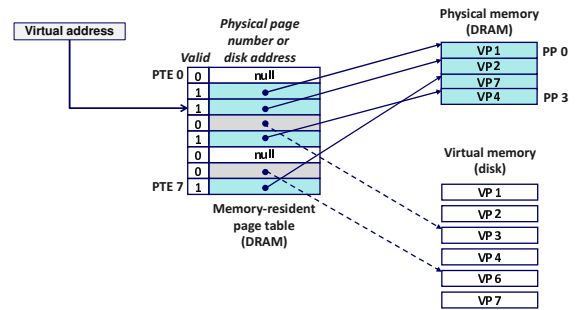
- 9 -

CS 105

## Page Hit



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



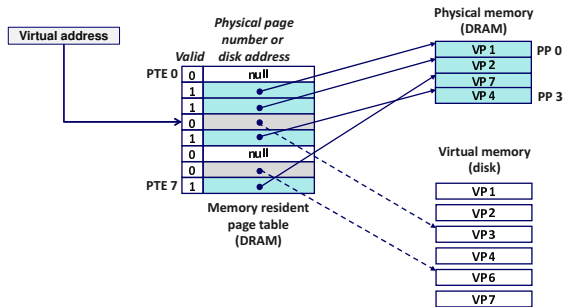
- 10 -

CS 105

## Page Fault



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



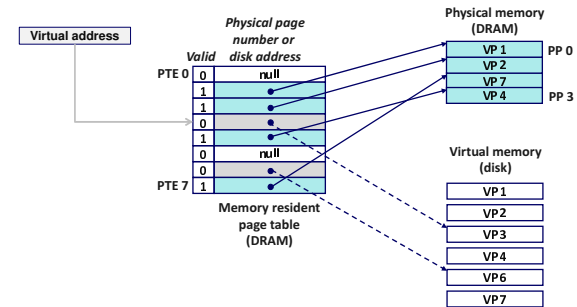
- 11 -

CS 105

## Handling Page Fault



Page miss causes page fault (an exception)



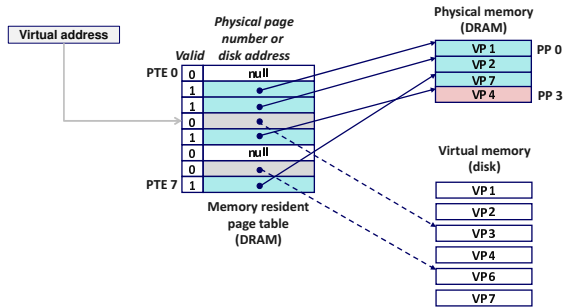
- 12 -

CS 105

## Handling Page Fault

Page miss causes page fault (an exception)

Page fault handler selects a victim to be evicted (here VP 4)



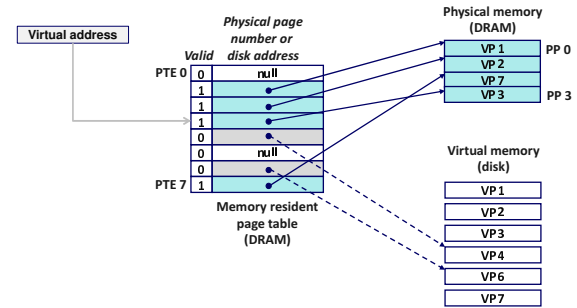
- 13 -

CS 105

## Handling Page Fault

Page miss causes page fault (an exception)

Page fault handler selects a victim to be evicted (here VP 4)



- 14 -

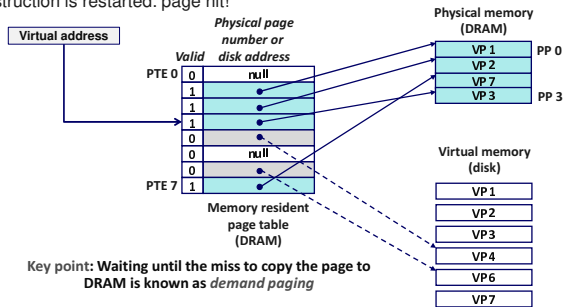
CS 105

## 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!

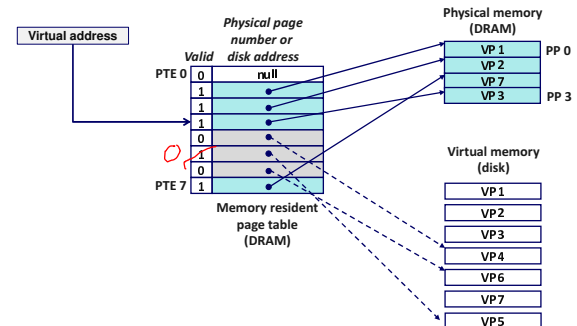


- 15 -

CS 105

## Allocating Pages

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



- 16 -

CS 105

## Locality to the Rescue Again!



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  $\text{SUM}(\text{working set sizes}) > \text{main memory size}$

- Thrashing:** Performance meltdown where pages are swapped (copied) in and out continuously

- 17 -

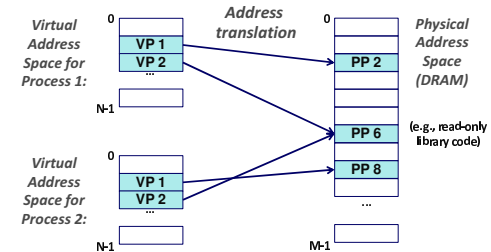
CS 105

## VM as Tool for Memory Management



Key idea: each process has own virtual address space

- Can view memory as a simple linear array
- Mapping function scatters addresses through physical memory
  - (But well-chosen mappings can improve locality in L1-L3 caches)



- 18 -

CS 105

## VM as Tool for Memory Management

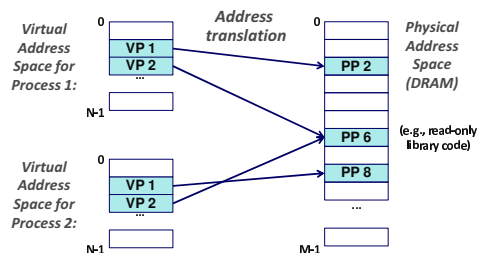


Memory allocation

- 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 multiple virtual pages to the same physical page (here: PP 6)



- 19 -

CS 105

## Simplifying Linking and Loading

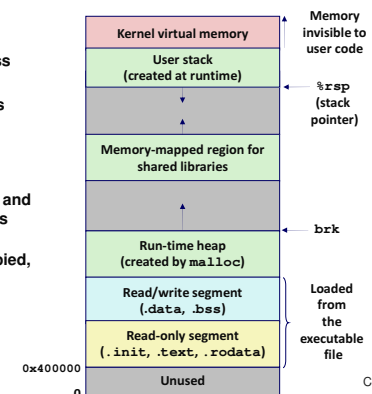


Linking

- Each program has similar virtual address space
- Code, stack, and shared libraries always start at same virtual address

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
  - Called "paging in" the program

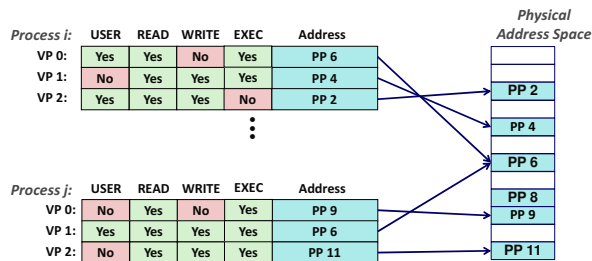


- 20 -

CS 105

# VM as Tool for Memory Protection

Extend PTEs (page table entries) with permission bits  
 Page-fault handler checks these bits before remapping  
 ■ If violated, send process SIGSEGV (segmentation fault)



# 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  $a$ :

- $MAP(a) = a'$  if data at virtual address  $a$  is at physical address  $a'$  in  $P$
- $MAP(a) = \emptyset$  if data at virtual address  $a$  is not in physical memory  
 » Either invalid or stored on disk

# 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)
- $M' = M / P$ : Number of physical pages

Components of the virtual address (VA)

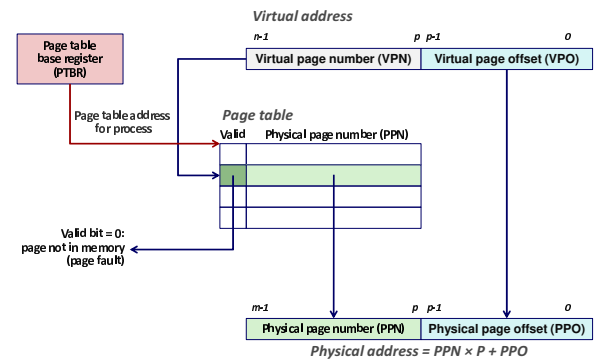
- VPN: Virtual page number
- VPO: Virtual page offset
- TLBI: TLB index
- TLBT: TLB tag

Components of the physical address (PA)

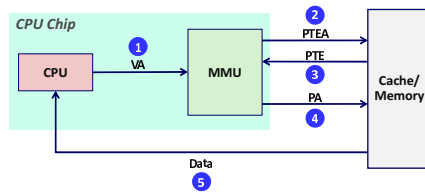
- PPN: Physical page number
- PPO: Physical page offset (same as VPO)
- PTE: Page table entry
- PTEA: Address of PTE

There's a bunch of these:  
 It'll take time to learn them.  
 The highlighted ones are the 8 most important. The greyed ones are the least.

# Address Translation With a Page Table



## Address Translation: Page Hit

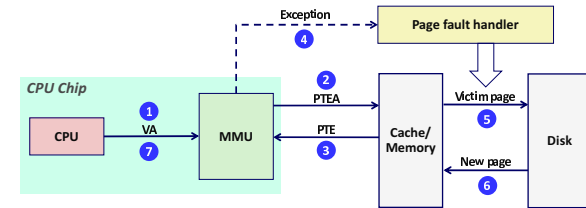


- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) MMU sends physical address to cache/memory
- 5) Cache/memory sends data word to processor

- 25 -

CS 105

## Address Translation: Page Fault

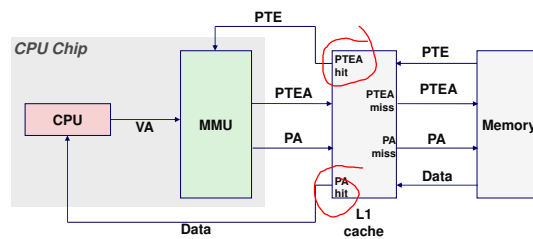


- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) Valid bit is zero, so MMU triggers page fault exception
- 5) Handler identifies victim (and, if dirty, pages it out to disk)
- 6) Handler "pages in" (reads) new page and updates PTE in memory
- 7) Handler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

- 26 -

CS 105

## Integrating VM and Cache



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

- 27 -

CS 105

## Speeding up Translation With a TLB

Page table entries (PTEs) are cached in L1 like any other memory word

- PTEs may be evicted by other data references
- PTE hit still requires a small but significant L1 delay (3-4 cycles)
  - Net effect is to double time needed to access data in L1 cache!

**Solution: Translation Lookaside Buffer (TLB)**

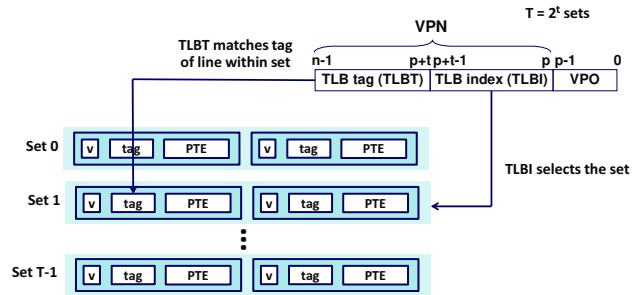
- Tiny set-associative (or fully associative) hardware cache inside MMU
- Maps virtual page numbers to physical page numbers
- Contains complete page table entries for small number of pages

- 28 -

CS 105

## Accessing the TLB

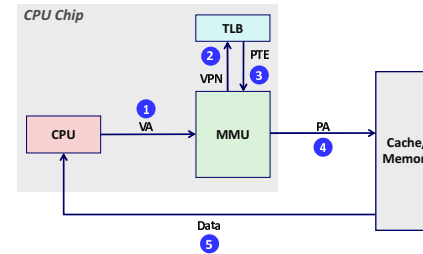
MMU uses the VPN portion of the virtual address to access the TLB:



- 29 -

CS 105

## TLB Hit

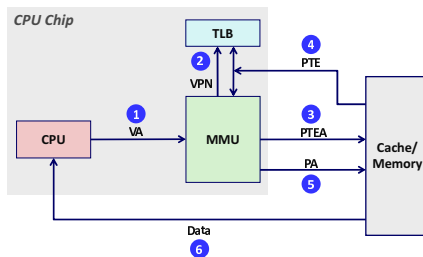


A TLB hit eliminates a cache or memory access to get the PTE

- 30 -

CS 105

## TLB Miss



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

- 31 -

CS 105

## Multi-Level Page Tables

Suppose:

- 4KB ( $2^{12}$ ) page size, 48-bit virtual address space, 8-byte PTE

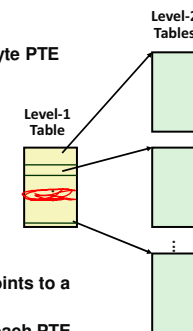
Problem:

- 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 (always memory-resident): each PTE points to a page table
- Level 2 table (paged in and out like any other data): each PTE points to a page



- 32 -

CS 105



