Virtual Memory

CSCI 237: Computer Organization
30th Lecture, Monday, November 17, 2025

Kelly Shaw

Administrative Details

- Read CSAPP 9.1-9.6 (Ch. 9 sections are short)
- Lab #5 due Tueday at 11pm
- Sign up for partner on lab #6 by Wednesday at 8am
- Colloquium talk on Tuesday at 2:35pm in Wege
  - David Paulius, postdoc in Intelligent Robot Lab at Brown University
  - "Object-level Planning: Bridging Human Knowledge and Task and Motion Planning"
- Book extra credit
  - Need to sign up at least week in advance
  - Last days to talk to me are reading period

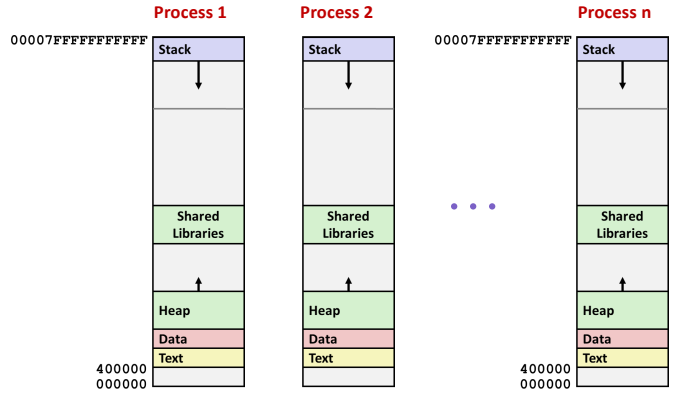
Last Time: Caches

- Cache memory organization and operation (Ch 6.4)
- Caches in real systems
- Performance impact of caches
  - The memory mountain

Today: Virtual Memory

- Address spaces (Ch 9.2)
- VM as a tool for caching (Ch 9.3)
- VM as a tool for memory management (Ch 9.4)
- VM as a tool for memory protection (Ch 9.5)
- Address translation (Ch 9.6)

Question: How Does This Work?!

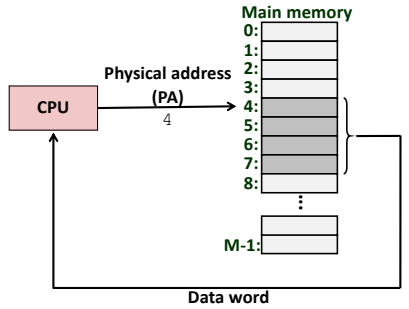


Solution: Virtual Memory

5

5

A System Using Physical Addressing

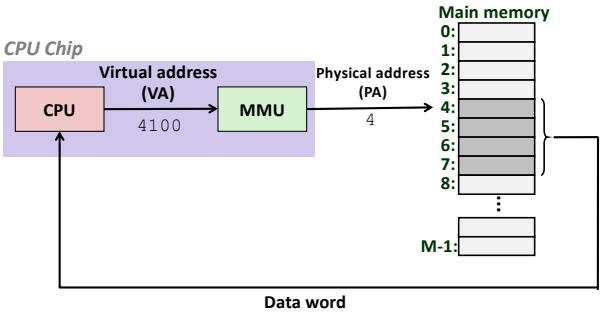


- Used in "simple" systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames

6

6

A System Using Virtual Addressing



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

7

7

Address Spaces: Terminology

- Linear address space: Ordered set of contiguous non-negative integer addresses (we always assume this):
  {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}

8

8

Why Virtual Memory (VM)?

- Uses main memory efficiently
  - Use DRAM as a cache for parts of a virtual address space
- Simplifies memory management
  - Each process gets the same uniform linear address space
- Isolates address spaces
  - Enables multiple processes to execute simultaneously
  - One process can't interfere with another's memory
  - Processes can allocate memory dynamically
  - *User* program cannot access privileged *kernel* information and code
- Total virtual memory in use can exceed physical memory

9

9

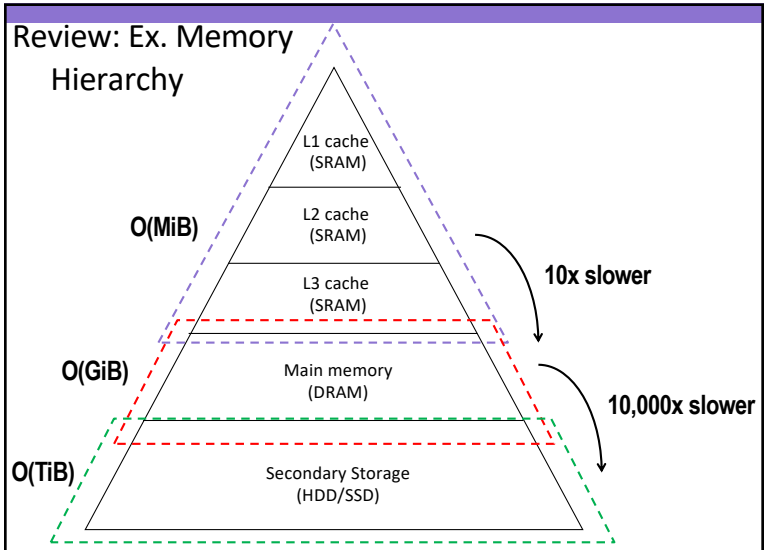
Today: Virtual Memory

- Address spaces (Ch 9.2)
- VM as a tool for caching (Ch 9.3)
- VM as a tool for memory management (Ch 9.4)
- VM as a tool for memory protection (Ch 9.5)
- Address translation (Ch 9.6)

10

10

Review: Ex. Memory Hierarchy

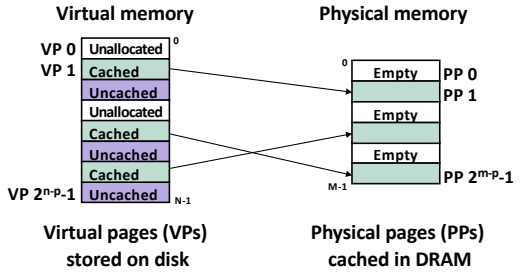


11

11

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)



12

12

DRAM Cache Organization

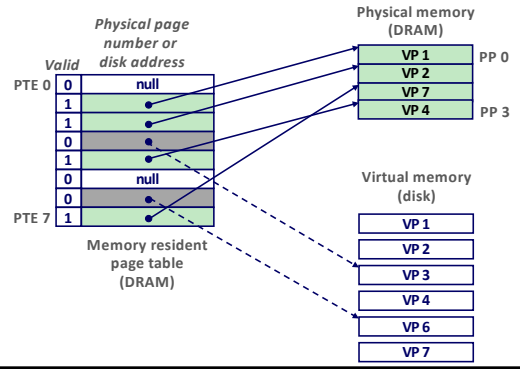
- DRAM cache organization driven by the enormous miss penalty
  - DRAM is about ***10x*** slower than SRAM
  - Disk is about ***10,000x*** slower than DRAM
- Consequences
  - Large page (block) size: typically 4 KB, sometimes 4 MB
  - Fully associative
    - Any VP can be placed in any PP
    - Requires 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

13

13

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 (i.e., OS) data structure in DRAM

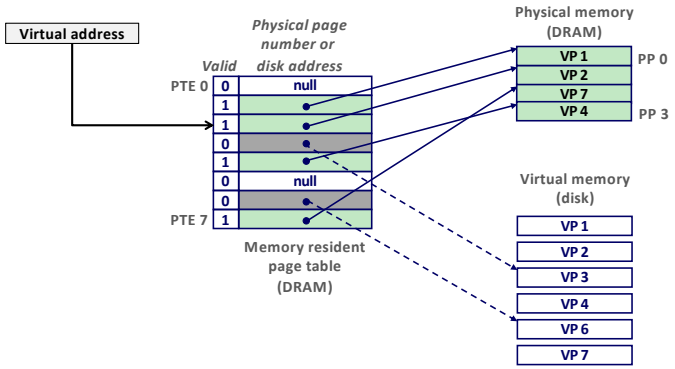


14

14

Page Hit

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

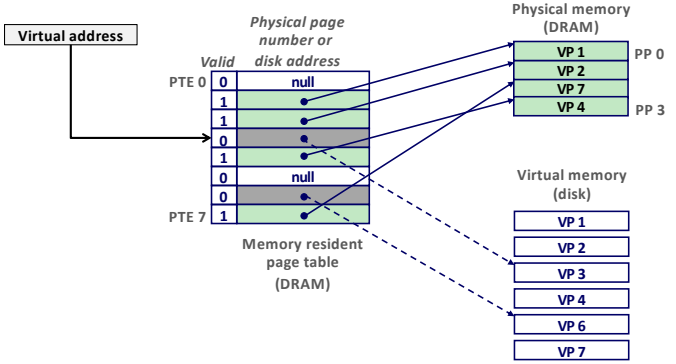


15

15

Page Fault

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

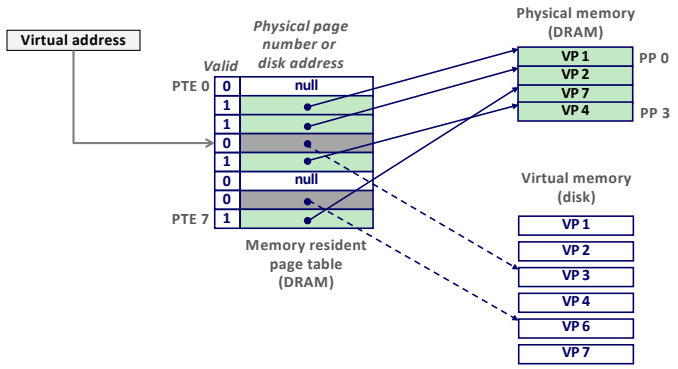


16

16

Handling Page Fault

- Page miss causes page fault (an exception)

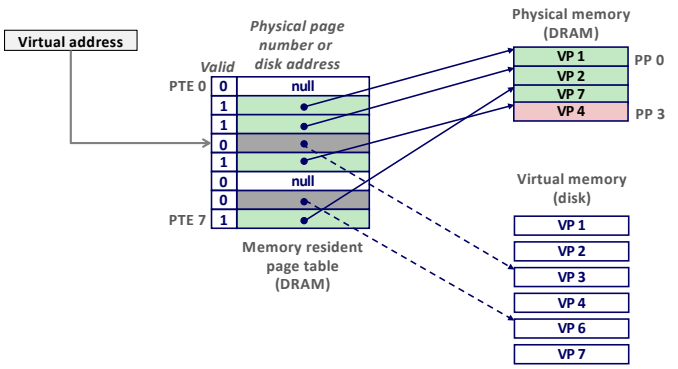


17

17

Handling Page Fault

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

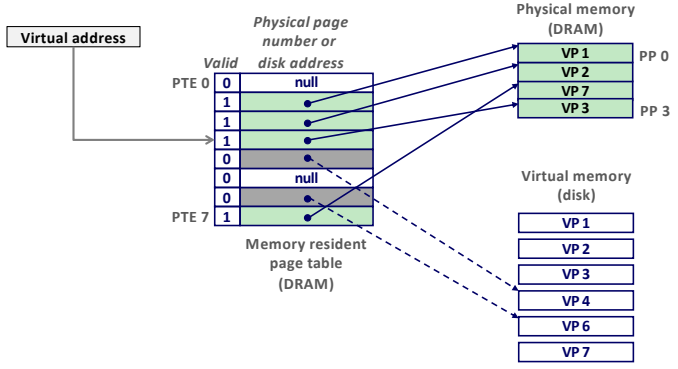


18

18

Handling Page Fault

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

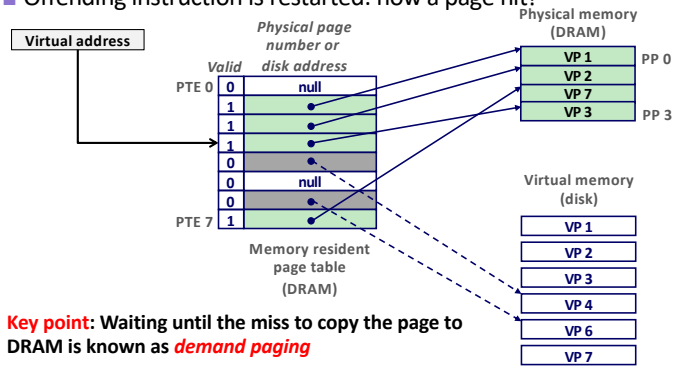


19

19

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: now a page hit!



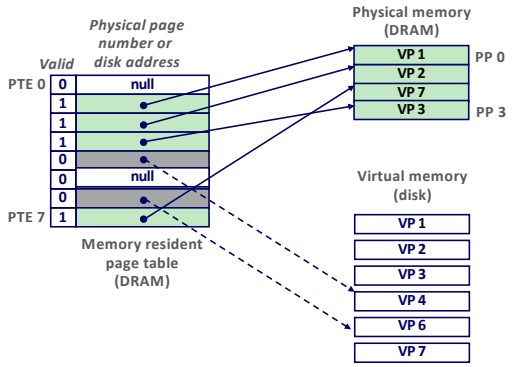
Key point: Waiting until the miss to copy the page to DRAM is known as *demand paging*

20

20

Allocating Pages

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

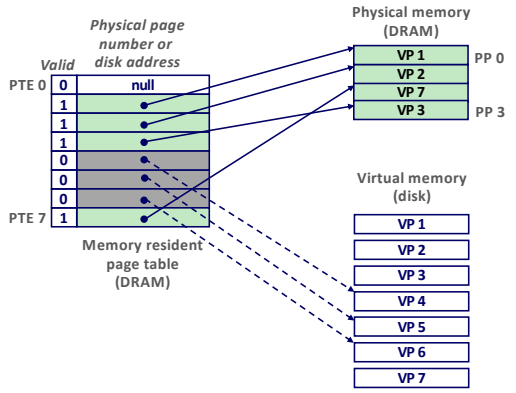


21

21

Allocating Pages

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



22

22

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* initial compulsory misses
- If (SUM(working set sizes) > main memory size)
  - *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously

23

23

Today: Virtual Memory

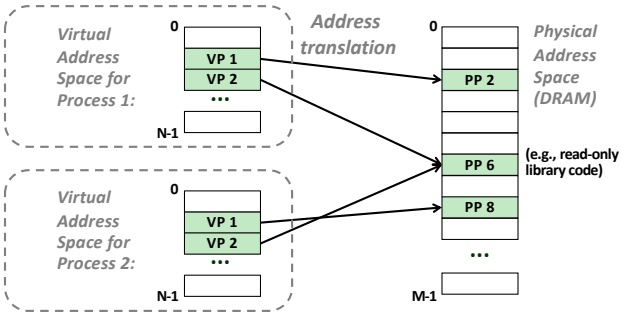
- Address spaces (Ch 9.2)
- VM as a tool for caching (Ch 9.3)
- VM as a tool for memory management (Ch 9.4)
- VM as a tool for memory protection (Ch 9.5)
- Address translation (Ch 9.6)

24

24

VM as a Tool for Memory Management

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



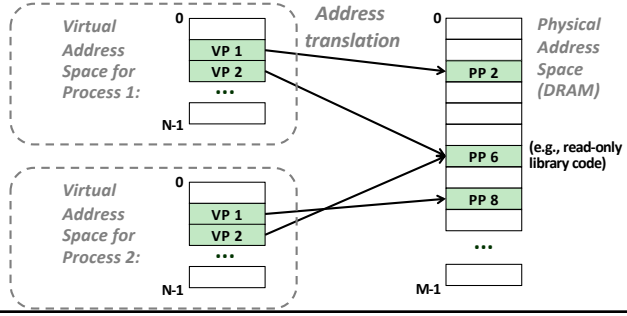
25

25

VM as a Tool for Memory Management

- Simplifies 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
- Allows sharing code and data among processes
  - Can map virtual pages to the same physical page (here: PP 6)

Why?



26

26

Today: Virtual Memory

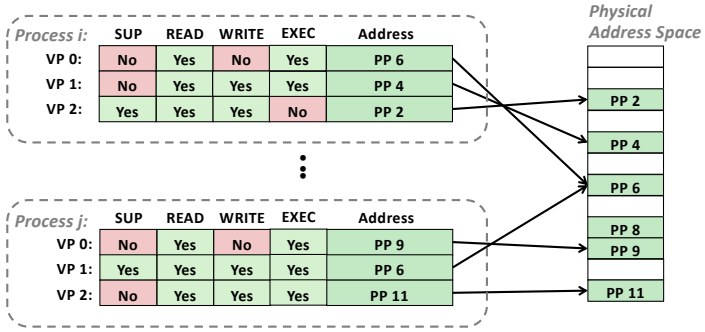
- Address spaces (Ch 9.2)
- VM as a tool for caching (Ch 9.3)
- VM as a tool for memory management (Ch 9.4)
- VM as a tool for memory protection (Ch 9.5)
- Address translation (Ch 9.6)

28

28

VM as a Tool for Memory Protection

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



29

29

Today: Virtual Memory

- VM as a tool for caching (Ch 9.3)
- VM as a tool for memory management (Ch 9.4)
- VM as a tool for memory protection (Ch 9.5)
- Address translation (Ch 9.6)

30

30

VM Address Translation (formally)

- 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 in V is at physical address a' in P
    - $MAP(a) = \varnothing$ if data at virtual address a is not in physical memory (either unallocated or stored on disk)

31

31

Summary of Address Translation Jargon

- 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 (in bytes) of physical and virtual pages
- Components of the virtual address (VA)
  - **VPN**: Virtual page number
  - **VPO**: Virtual page offset
  - **TLBI**: TLB (Translation Lookaside Buffer) index
  - **TLBT**: TLB tag
- Components of the physical address (PA)
  - **PPN:** Physical page number
  - **PPO**: Physical page offset (same as VPO)

32

32