CMPE 110: Computer Architecture

Week 9
Virtual Memory

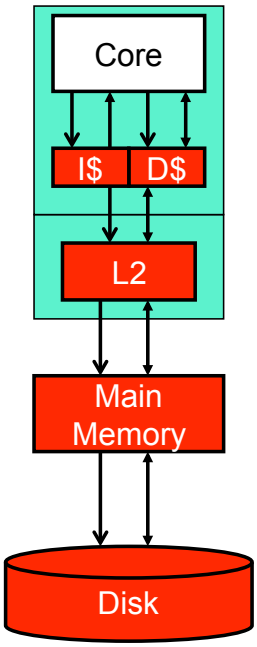
Jishen Zhao (http://users.soe.ucsc.edu/~jzhao/)

[Adapted in part from Jose Renau, Mary Jane Irwin, Joe Devietti, Onur Mutlu, and others]

Reminder

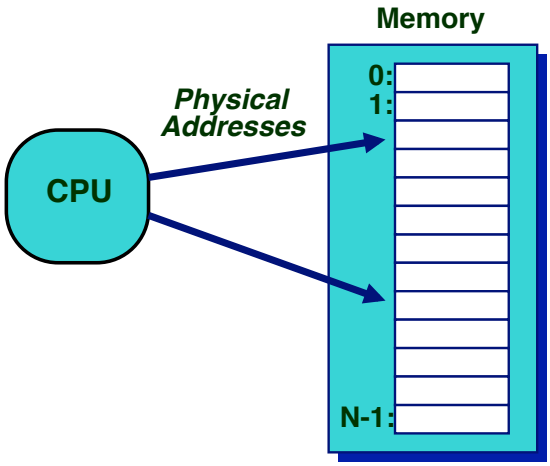
- Homework 4 will be posted on today

Today

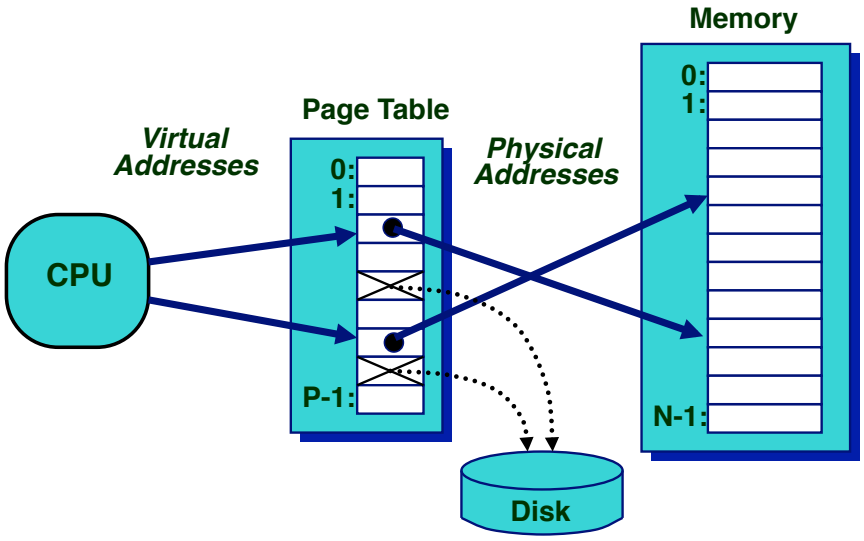


Virtualizing the memory hierarchy: An Infinite capacity

Review: If no virtual memory



Review: A System with Virtual Memory



- Address Translation: The hardware converts virtual addresses into physical addresses via an OS-managed lookup table (page table)

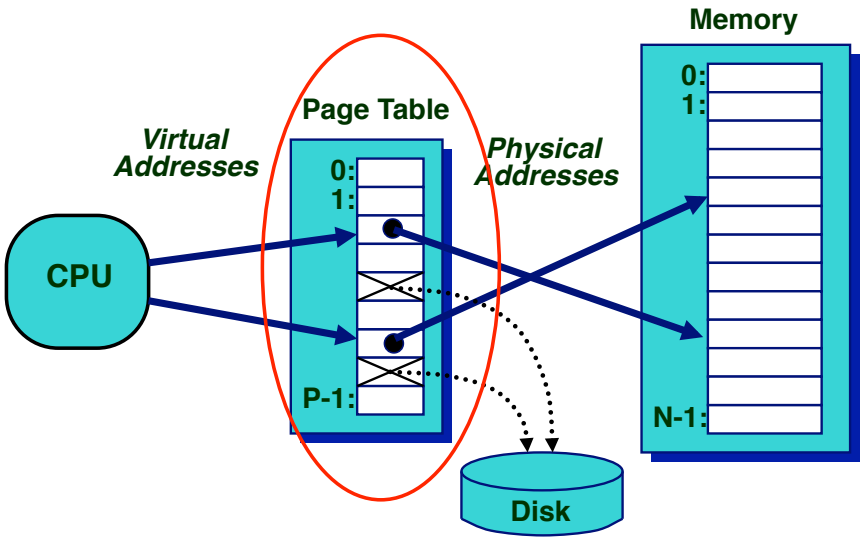
Virtual Memory

- Idea: Give the programmer the illusion of a large address space while having a small physical memory
  - So that the programmer does not worry about managing physical memory
- Programmer can assume he/she has "infinite" amount of physical memory
- Hardware and software cooperatively and automatically manage the physical memory space to provide the illusion
  - Illusion is maintained for each independent process

Today

- Virtual memory
  - "Infinite" memory, isolation, protection, inter-process communication
  - Virtual-physical address translation
  - Page tables
  - TLBs
  - Page faults
  - DMA
  - Virtual memory and cache
- Multicore

A logical view



Virtual Pages, Physical Frames

- Virtual address space divided into pages
- Physical address space divided into frames
- A virtual page is mapped to
  - A physical frame, if the page is in physical memory
  - A location in disk, otherwise

- If an accessed virtual page is not in memory, but on disk
  - Get a page fault
  - Virtual memory system brings the page into a physical frame and adjusts the mapping → this is called demand paging

- Page table is the table that stores the mapping of virtual pages to physical frames

Physical Memory as a Cache

- In other words…

- Physical memory is a cache for pages stored on disk
  - In fact, it is a fully associative cache in modern systems (a virtual page can be mapped to any physical frame)

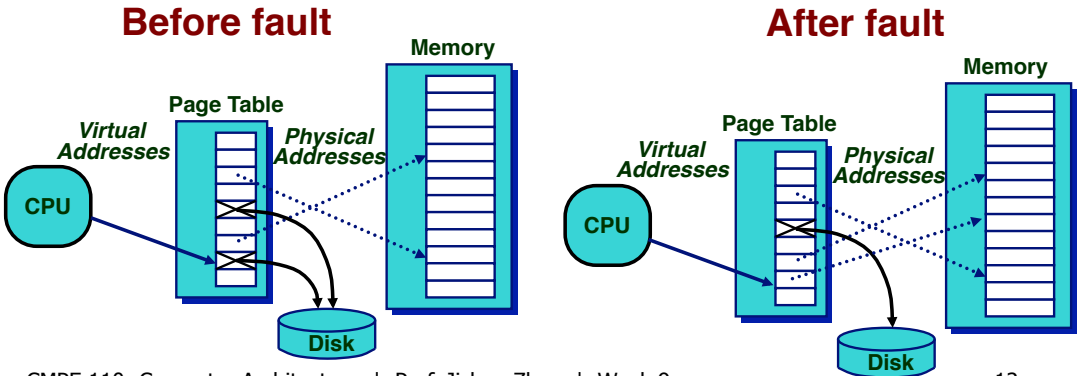
- Similar caching issues exist as we have covered earlier:
  - Placement: where and how to place/find a page in cache?
  - Replacement: what page to remove to make room in cache?
  - Granularity of management: large, small, uniform pages?
  - Write policy: what do we do about writes? Write back?

Supporting Virtual Memory

- Virtual memory requires both HW+SW support
  - Page Table is in memory
  - Can be cached in special hardware structures called Translation Lookaside Buffers (TLBs)
- The hardware component is called the MMU (memory management unit)
  - Includes Page Table Base Register(s), TLBs, page walkers
- It is the job of the software to leverage the MMU to
  - Populate page tables, decide what to replace in physical memory
  - Change the Page Table Register on context switch (to use the running thread's page table)
  - Handle page faults and ensure correct mapping

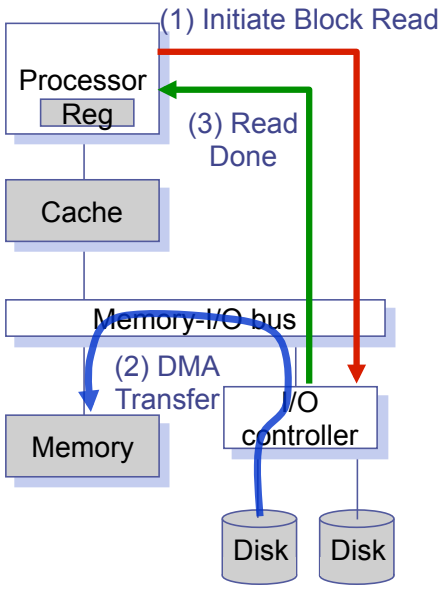
Page Fault ("A Miss in Physical Memory")

- If a page is not in physical memory but disk
  - Page table entry indicates virtual page not in memory
  - Access to such a page triggers a page fault exception
  - OS trap handler invoked to move data from disk into memory
    - Other processes can continue executing
    - OS has full control over placement



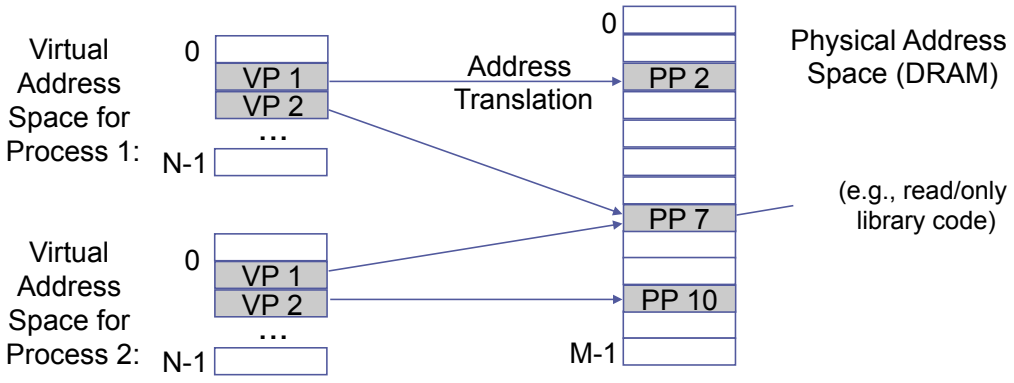
Servicing a Page Fault and DMA

- (1) Processor signals controller
  - Read block of length P starting at disk address X and store starting at memory address Y
- (2) Read occurs
  - Direct Memory Access (DMA)
  - Under control of I/O controller
- (3) Controller signals completion
  - Interrupt processor
  - OS resumes suspended process



Page Table is Per Process

- Each process has its own virtual address space
  - Full address space for each program
  - Simplifies memory allocation, sharing, linking and loading.

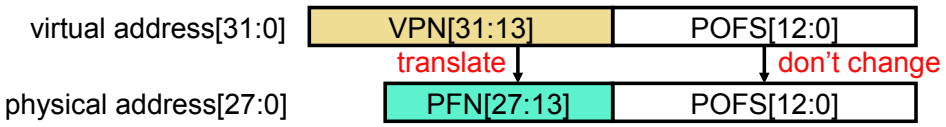


Virtual to Physical
Address translation

Address Translation

- How to obtain the physical address from a virtual address?
- Page size specified by the ISA
  - VAX: 512 bytes (by DEC in 1970s)
  - Today: 4KB, 8KB, 2GB, … (small and large pages mixed together)
- Page Table contains an entry for each virtual page
  - Called Page Table Entry (PTE)
  - What is in a PTE?

Address Translation (II)

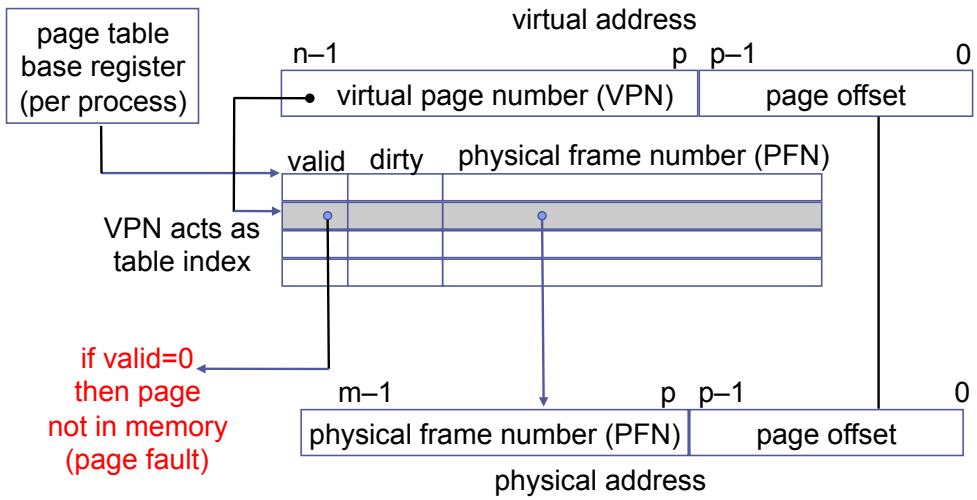


- VA→PA mapping called **address translation**
  - Split VA into **virtual page number (VPN)** & **page offset (POFS)**
  - Translate VFN into **physical frame number (PFN)**
  - POFS is not translated

- Example above
  - 8KB page size
  - 32-bit virtual address
  - VPN 19 bits →2^19 virtual pages →
    2^19 PTEs in the page table (for each process)

What's in a page table

- Separate (set of) page table(s) per process
- VPN forms index into page table (points to a page table entry)
- Page Table Entry (PTE) provides information about page



Virtual Memory Issue

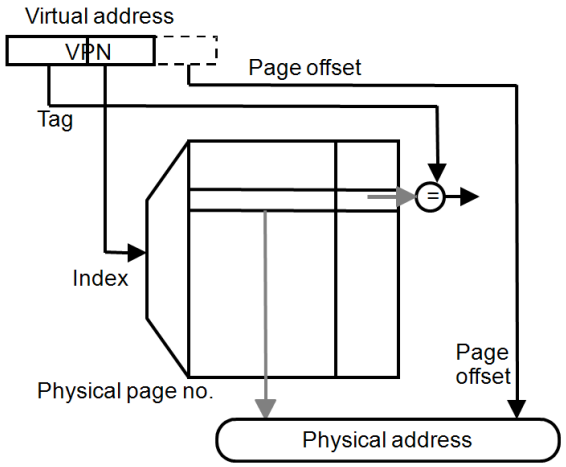
- How fast is the address translation?
  - Cost at least one memory read
  - How can we make it fast?
- Idea: Use a hardware structure that caches PTEs →
  - Translation lookaside buffer (TLB)
  - Actually a cache, not a buffer

Speeding up Translation with a TLB

- Essentially a cache of recent address translations
  - Avoids going to the page table on every reference

- **Index** = lower bits of VPN (virtual page #)
- **Tag** = unused bits of VPN + process ID
- **Data** = a page-table entry
- **Status** = valid, dirty

The usual cache design choices (placement, replacement policy, multi-level, etc.) apply here too.



Handling TLB Misses

- The TLB is small; it cannot hold all PTEs
  - Some translations will inevitably miss in the TLB
  - Must access memory to find the appropriate PTE
    - Called walking the page directory/table
    - Large performance penalty

- Who handles TLB misses? Hardware or software?

Handling TLB Misses (II)

- Approach #1. **Hardware-Managed** (e.g., x86)
  - The hardware does the **page walk**
  - The hardware fetches the PTE and inserts it into the TLB
    - If the TLB is full, the entry **replaces** another entry
  - Done transparently to system software

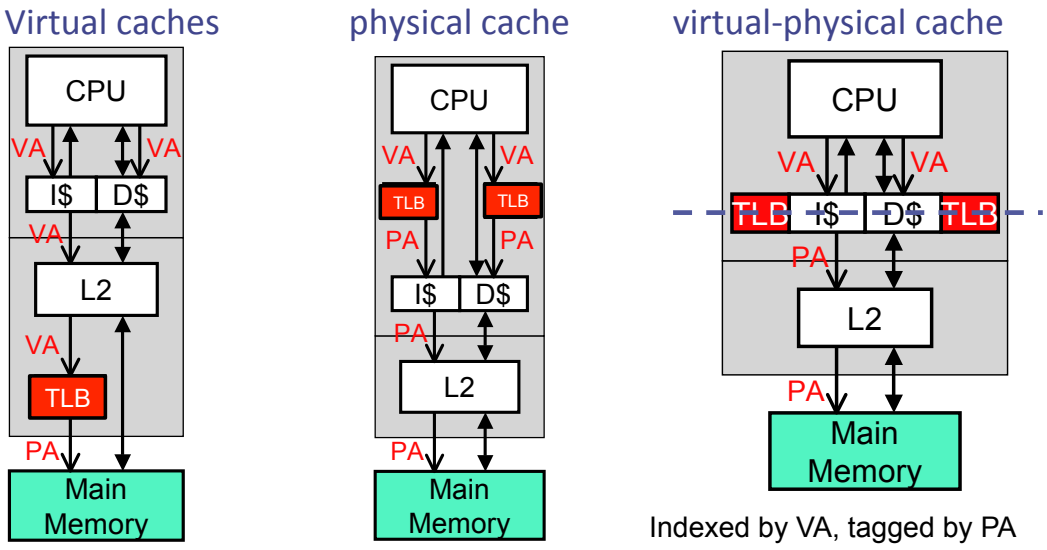
- Approach #2. **Software-Managed** (e.g., MIPS)
  - The hardware raises an exception
  - The operating system does the **page walk**
  - The operating system fetches the PTE
  - The operating system inserts/evicts entries in the TLB

Handling TLB Misses (III)

- Hardware-Managed TLB
  - + No exception on TLB miss. Instruction just stalls
  - + Independent instructions may continue
  - + No extra instructions/data brought into caches.
  - - Page table organization is etched into the system: OS has little flexibility in deciding these

- Software-Managed TLB
  - + The OS can define page table organization
  - + More sophisticated TLB replacement policies are possible
  - - Need to generate an exception → performance overhead due to pipeline flush, exception handler execution, extra instructions brought to caches

Cache-VM Interaction: Locations of TLB



What we learned

- OS virtualizes memory and I/O devices

- Virtual memory
  - "infinite" memory, isolation, protection, inter-process communication
  - Virtual-physical address translation
  - Page tables
  - TLBs
    - It is a cache, not a buffer
  - Page faults
  - DMA
  - Virtual memory and cache

What we learned

- Why do need virtual memory?
- What is page table? Where is it?
- What is TLB? Is it a buffer or a cache?
- What is page fault? How is page fault handled? What is DMA?
- What is MMU? What does it do?
- What is virtual caches? What is physical cache? What is virtual-physical cache?
- Virtual-physical address translation. Example question on the next slide…