Computer Architecture

Lecture 8: Virtual Memory

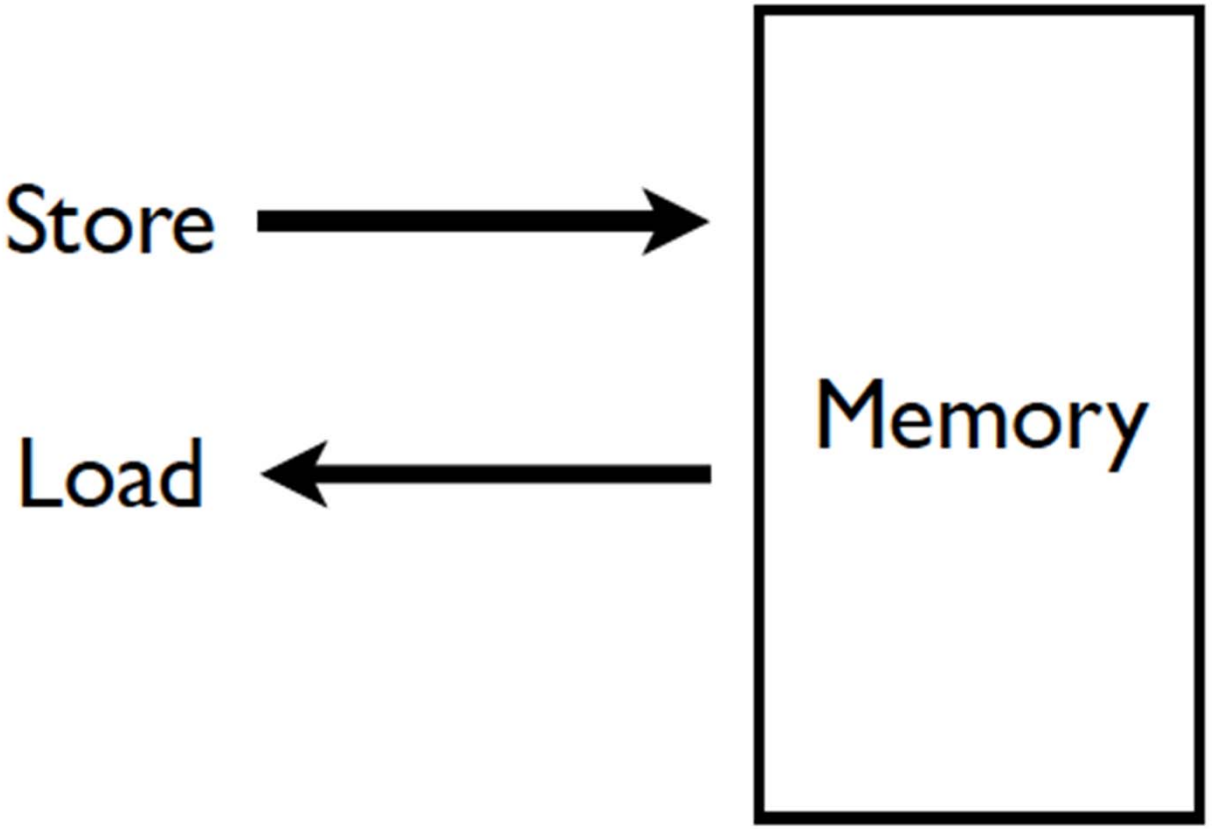
Dr. Ahmed Sallam

Suez Canal University

Spring 2015

Based on original slides by Prof. Onur Mutlu

Memory (Programmer's View)



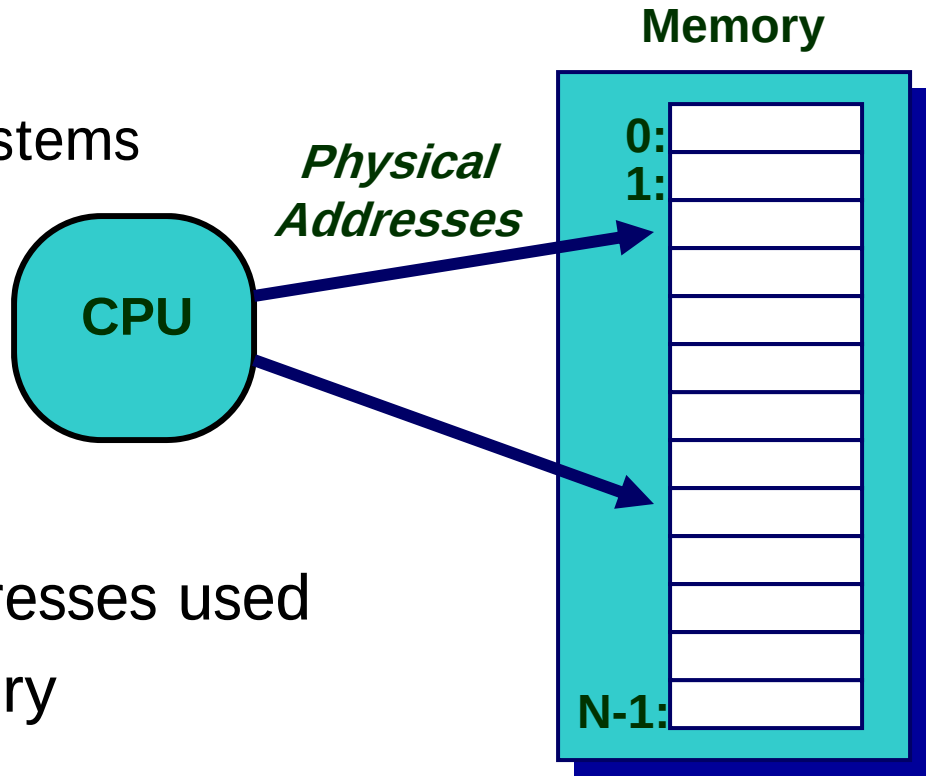
Ideal Memory

- Zero access time (latency)
- Infinite capacity
- Zero cost
- Infinite bandwidth (to support multiple accesses in parallel)

THE NEED TO VIRTUAL MEMORY

A System with Physical Memory Only

- Examples:
  - most Cray machines
  - early PCs
  - nearly all embedded systems



CPU's load or store addresses used directly to access memory

The Problem

- Physical memory is of limited size (cost)
  - What if you need more?
  - Should the programmer be concerned about the size of code/data blocks fitting physical memory?
  - Should the programmer manage data movement from disk to physical memory?
  - Should the programmer ensure two processes do not use the same physical memory?

- Also, ISA can have an address space greater than the physical memory size
  - E.g., a 64-bit address space with byte addressability
  - What if you do not have enough physical memory?

Difficulties of Direct Physical Addressing

- Programmer needs to manage physical memory space
  - Inconvenient & hard
  - Harder when you have multiple processes
- Difficult to support code and data relocation
- Difficult to support multiple processes
  - Protection and isolation between multiple processes
  - Sharing of physical memory space
- Difficult to support data/code sharing across processes

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

Abstraction: Virtual vs. Physical Memory

- Programmer sees virtual memory
  - Can assume the memory is "infinite"
- Reality: Physical memory size is much smaller than what the programmer assumes
- The system (system software + hardware, cooperatively) maps virtual memory addresses are to physical memory
  - The system automatically manages the physical memory space transparently to the programmer

+ Programmer does not need to know the physical size of memory nor manage it → A small physical memory can appear as a huge one to the programmer → Life is easier for the programmer

-- More complex system software and architecture

A classic example of the programmer/(micro)architect tradeoff

Benefits of Automatic Management of Memory

- Programmer does not deal with physical addresses
- Each process has its own mapping from virtual→physical addresses

- Enables
  - Code and data to be located anywhere in physical memory (relocation)
  - Isolation/separation of code and data of different processes in physical processes (protection and isolation)
  - Code and data sharing between multiple processes (sharing)

Basic Mechanism

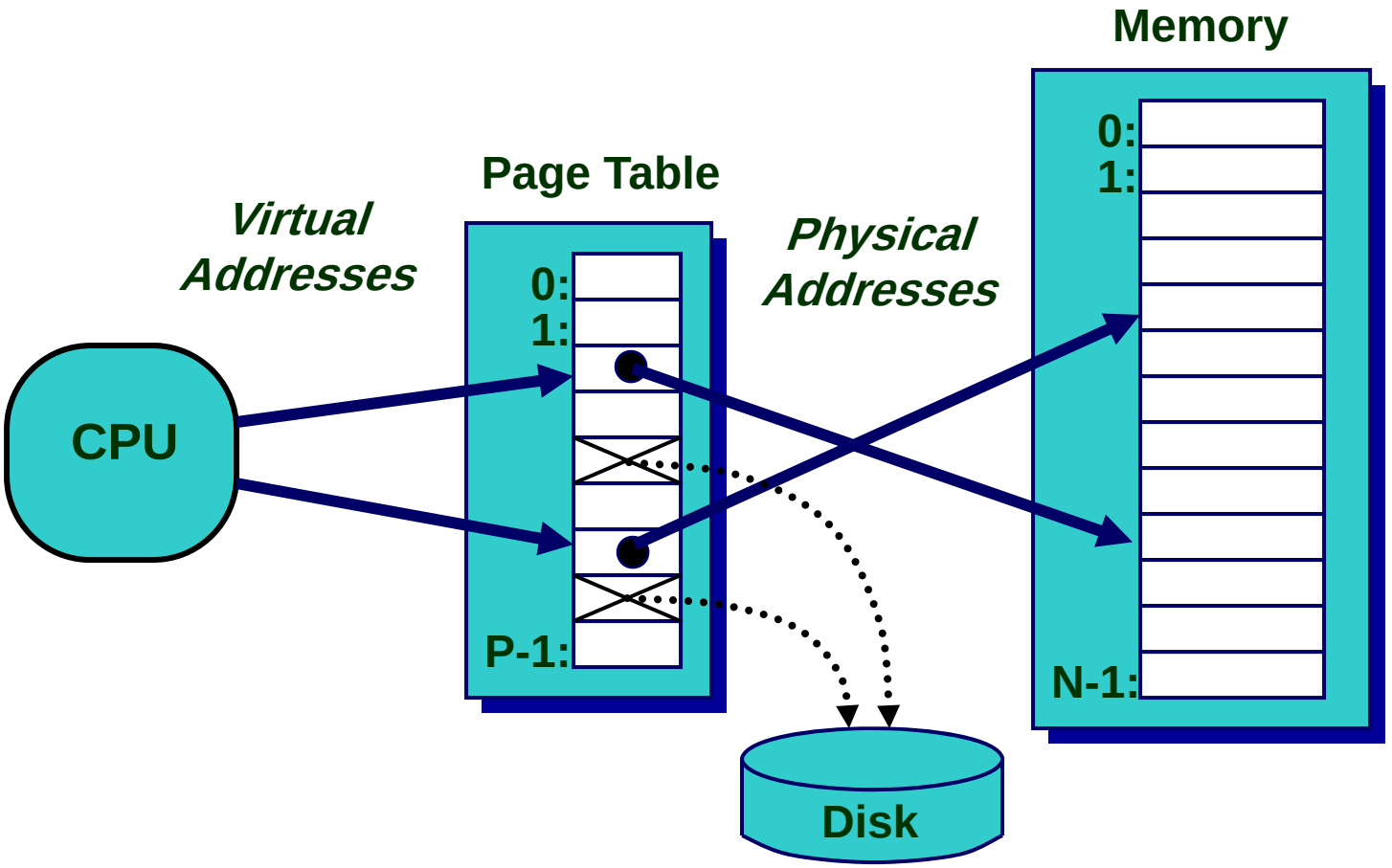
- Indirection (in addressing)

- Address generated by each instruction in a program is a "virtual address"
  - i.e., it is not the physical address used to address main memory

- An "address translation" mechanism maps this address to a "physical address"
  - Address translation mechanism can be implemented in hardware and software together

VIRTUAL MEMORY SYSTEM

A System with Virtual Memory (Page based)



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

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
  - 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

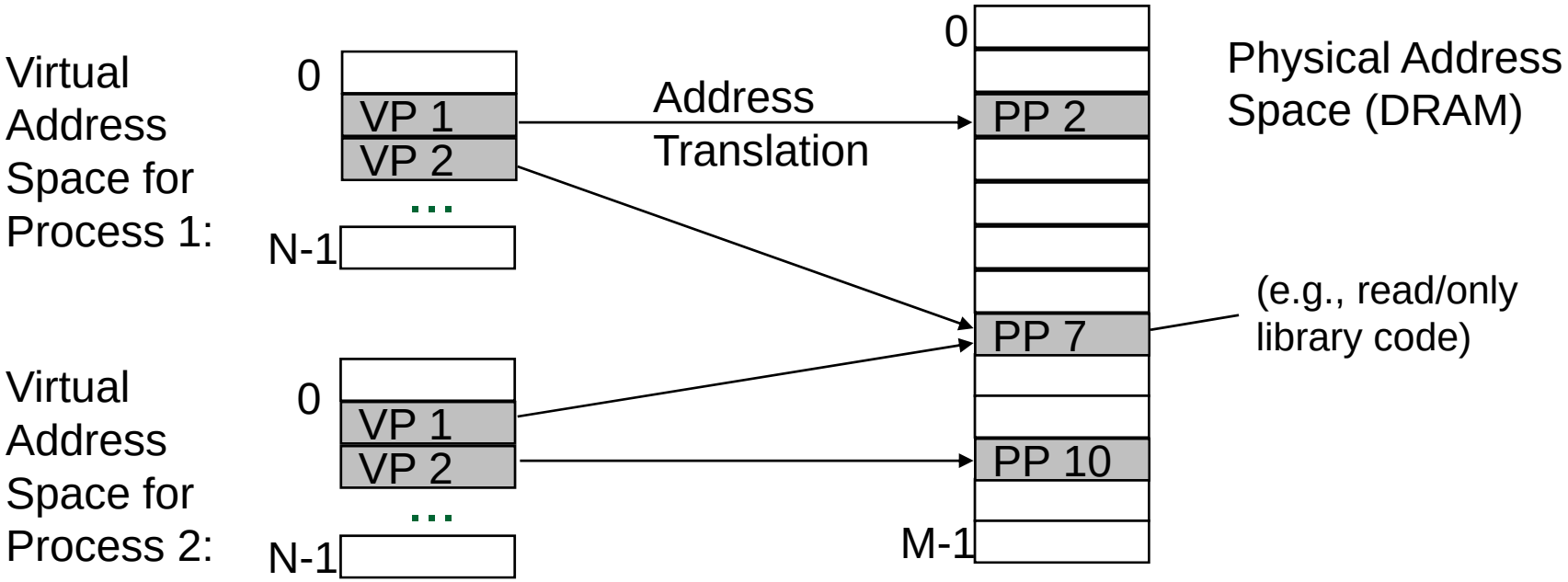
- It is the job of the software to leverage the MMU to
  - Populate page tables, decide what to replace in physical memory
  - Change the Page Table Register on context switch (to use the running thread's page table)
  - Handle page faults and ensure correct mapping

Some System Software Jobs for VM

- Keeping track of which physical frames are free
- Allocating free physical frames to virtual pages
- Page replacement policy
  - When no physical frame is free, what should be swapped out?
- Sharing pages between processes
- Copy-on-write optimization
- Page-flip optimization

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.



VM OPERATIONS

Address Translation

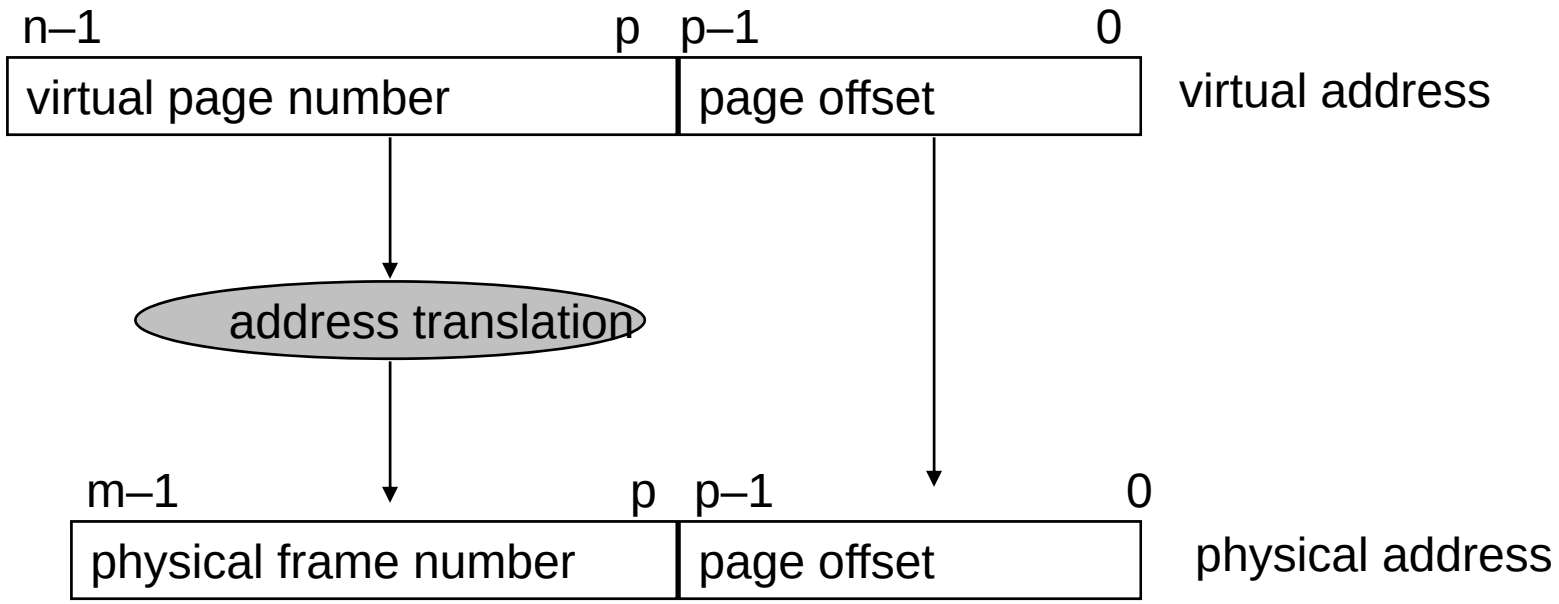
- How to obtain the physical address from a virtual address?

- Page size specified by the ISA
  - VAX: 512 bytes
  - Today: 4KB, 8KB, 2GB, ... (small and large pages mixed together)
  - Trade-offs? (remember cache lectures)

- Page Table contains an entry for each virtual page
  - Called Page Table Entry (PTE)
  - What is in a PTE?

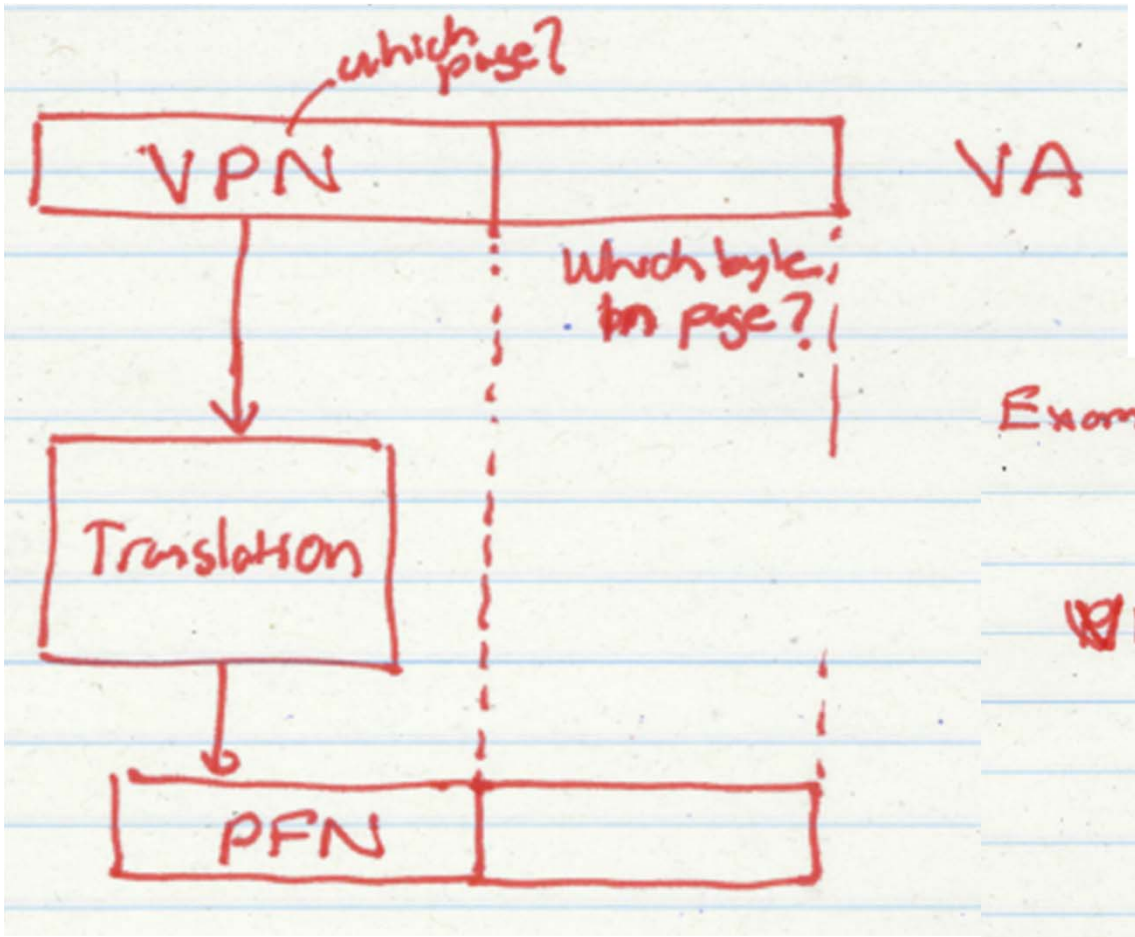
Address Translation (III)

- Parameters
  - P = 2^p = page size (bytes).
  - N = 2^n = Virtual-address limit
  - M = 2^m = Physical-address limit



Page offset bits don't change as a result of translation

Address Translation (II)

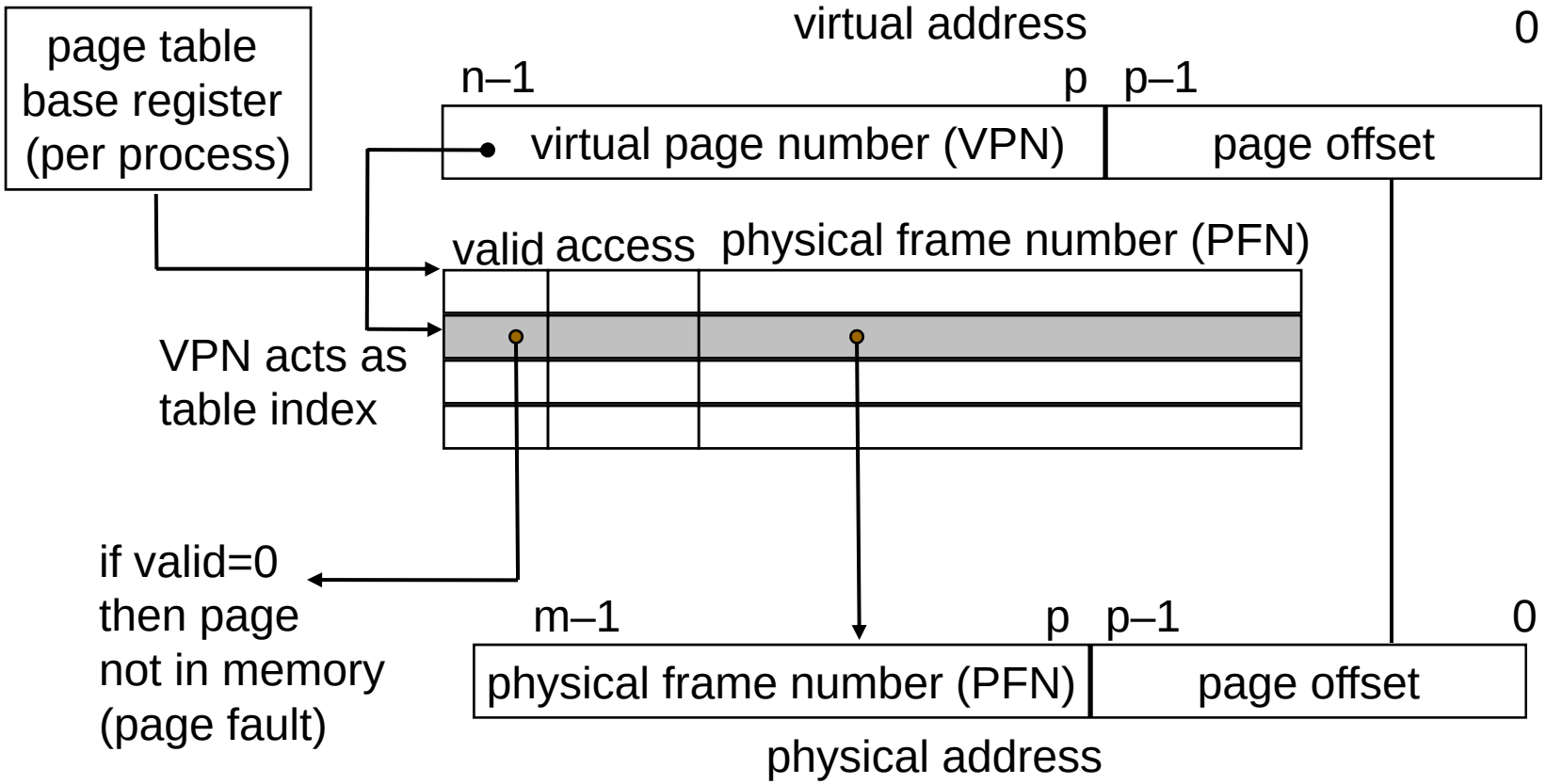


Example: 8K page size
32-bit virtual address space

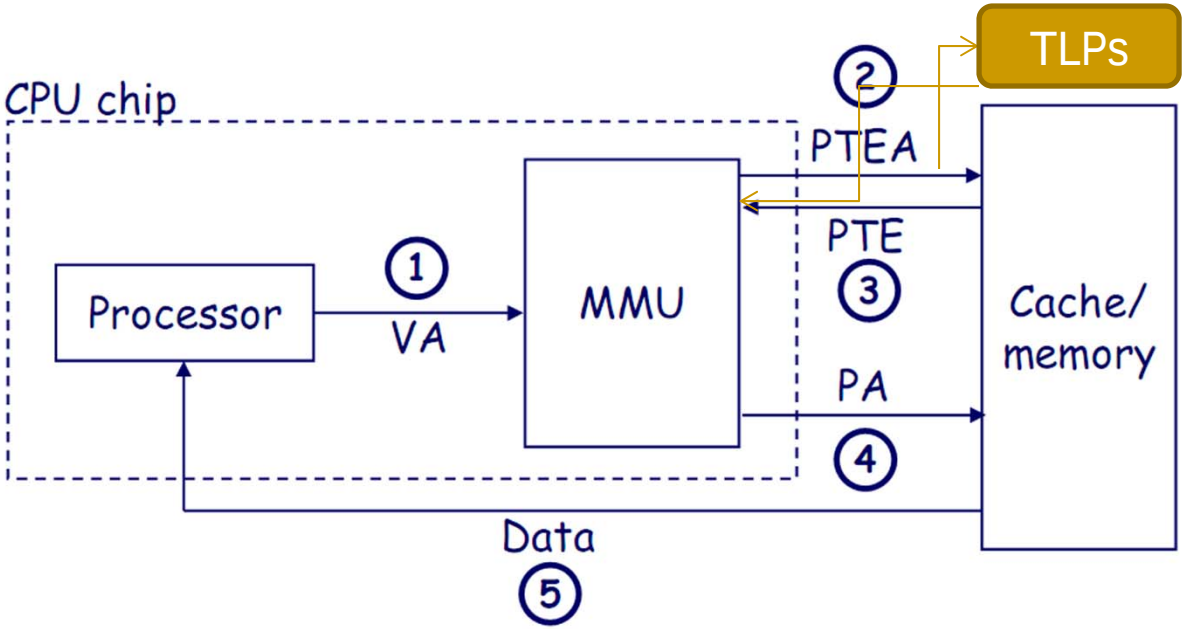
VPN → 19 bits
→ 2^{19} virtual pages
→ 2^{19} PTEs in page table
(for each process)

Address Translation (IV)

- 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



Address Translation: Page Hit



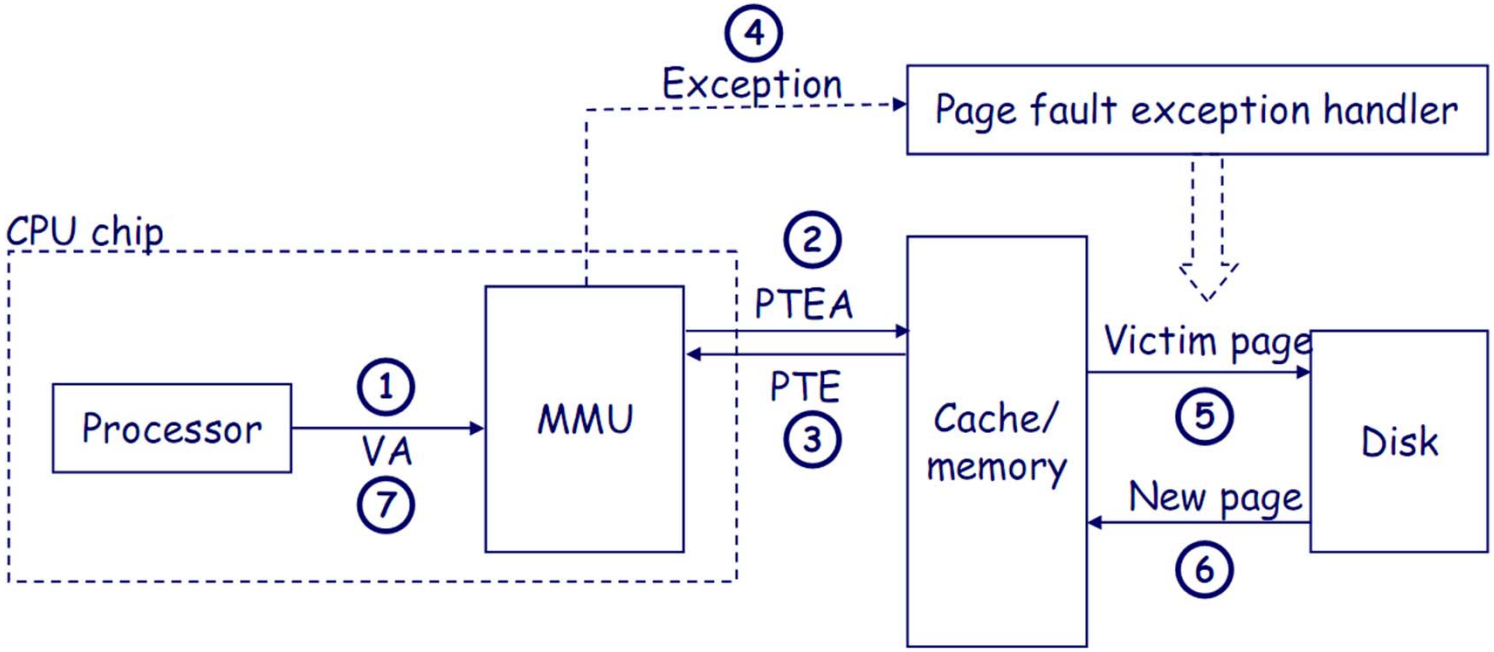
1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to L1 cache

5) L1 cache sends data word to processor

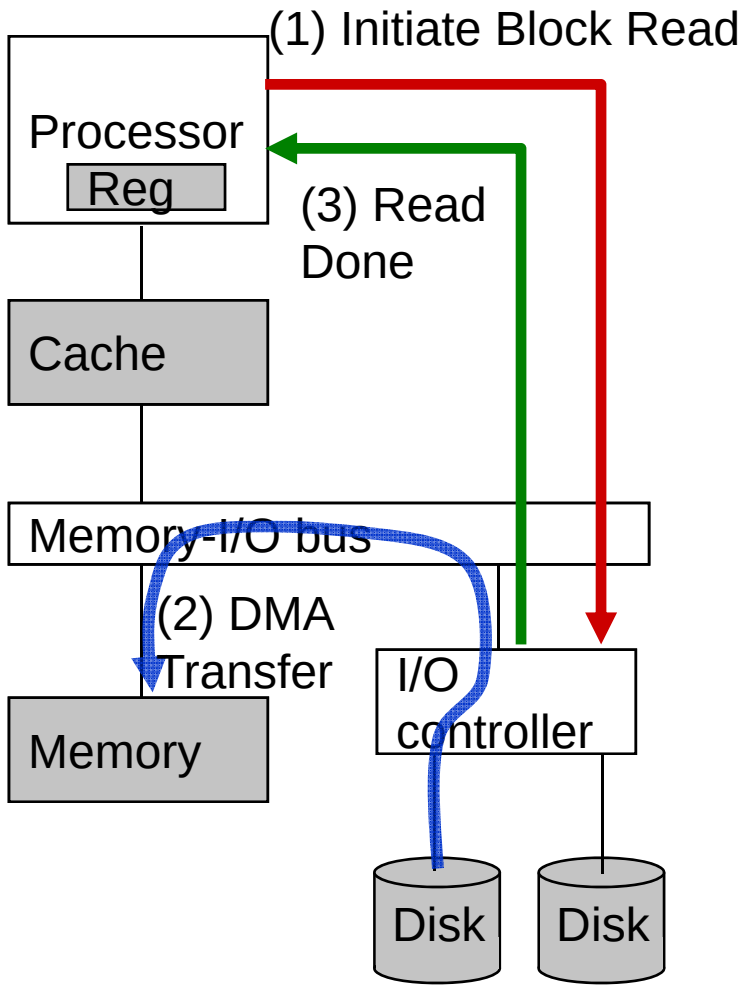
Address Translation: Page Fault



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.

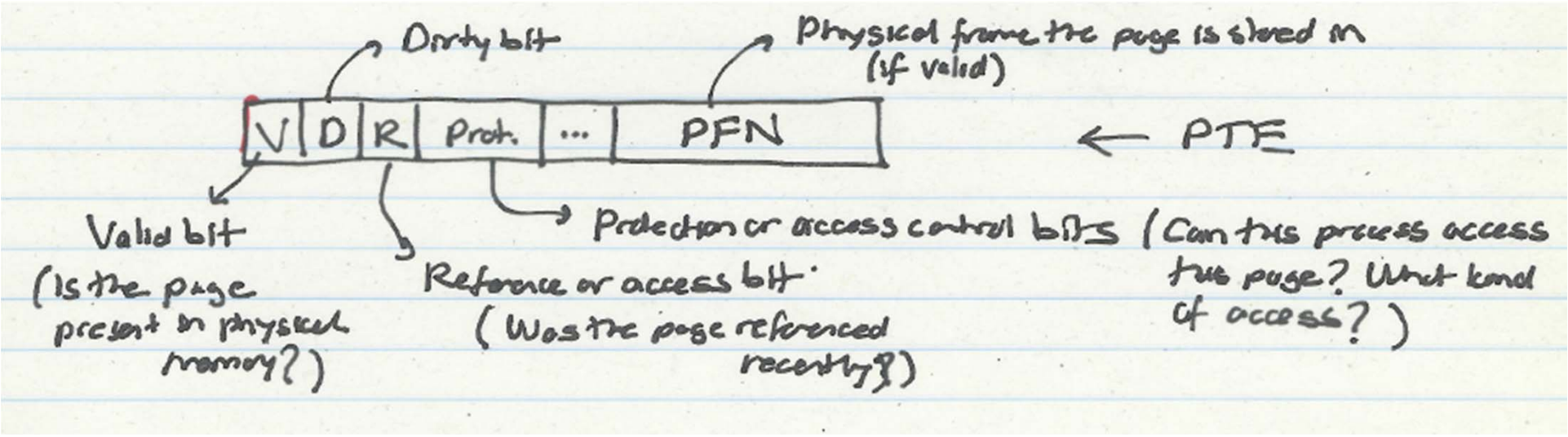
Servicing a Page Fault

- (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



What Is in a Page Table Entry (PTE)?

- Page table is the "tag store" for the physical memory data store
  - A mapping table between virtual memory and physical memory
- PTE is the "tag store entry" for a virtual page in memory
  - Need a valid bit → to indicate validity/presence in physical memory
  - Need tag bits (PFN) → to support translation
  - Need bits to support replacement
  - Need a dirty bit to support "write back caching"
  - Need protection bits to enable access control and protection



PAGE REPLACEMENT

Remember: Cache versus Page Replacement

- Physical memory (DRAM) is a cache for disk
  - Usually managed by system software via the virtual memory subsystem

- Page replacement is similar to cache replacement
- Page table is the "tag store" for physical memory data store

- What is the difference?
  - Required speed of access to cache vs. physical memory
  - Number of blocks in a cache vs. physical memory
  - "Tolerable" amount of time to find a replacement candidate (disk versus memory access latency)
  - Role of hardware versus software

Page Replacement Algorithms

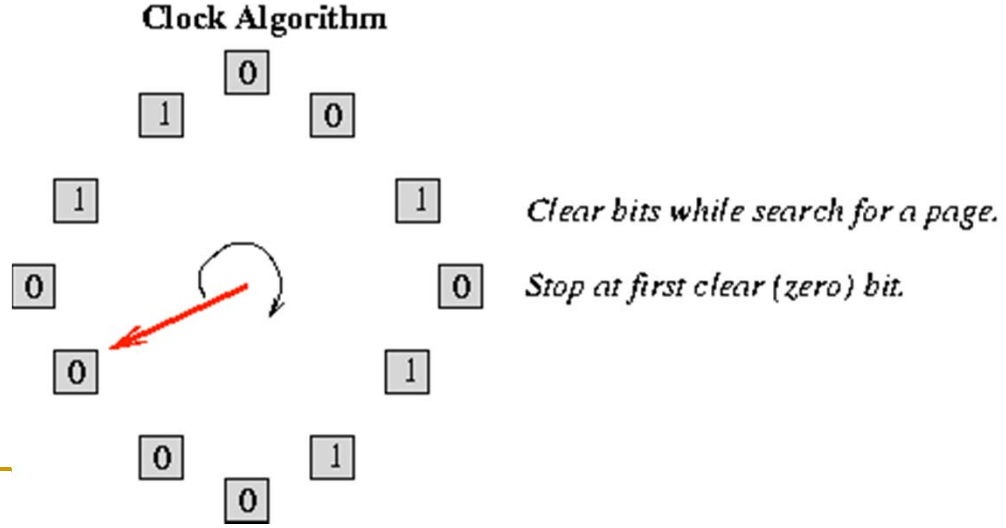
- If physical memory is full (i.e., list of free physical pages is empty), which physical frame to replace on a page fault?

- Is True LRU feasible?
  - 4GB memory, 4KB pages, how many possibilities of ordering?

- Modern systems use approximations of LRU
  - E.g., the CLOCK algorithm
- And, more sophisticated algorithms to take into account "frequency" of use
  - E.g., the ARC algorithm
  - Megiddo and Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," FAST 2003.

CLOCK Page Replacement Algorithm

- Keep a circular list of physical frames in memory
- Keep a pointer (hand) to the last-examined frame in the list
- When a page is accessed, set the R bit in the PTE
- When a frame needs to be replaced, replace the first frame that has the reference (R) bit not set, traversing the circular list starting from the pointer (hand) clockwise
  - During traversal, clear the R bits of examined frames
  - Set the hand pointer to the next frame in the list



Aside: Page Size Trade Offs

- What is the granularity of management of physical memory?
- Large vs. small pages
- Tradeoffs have analogies to large vs. small cache blocks

- Many different tradeoffs with advantages and disadvantages
  - Size of the Page Table (tag store)
  - Reach of the Translation Lookaside Buffer (we will see this later)
  - Transfer size from disk to memory (waste of bandwidth?)
  - Waste of space within a page (internal fragmentation)
  - Waste of space within the entire physical memory (external fragmentation)
  - Granularity of access protection
  - ...

Readings

- Section 5.4 in P&H