HARVARD

School of Engineering
and Applied Sciences

Virtual memory

CS61, Lecture 14

Prof. Stephen Chong

October 19, 2010

Announcements

- Lab 3 due tonight!
- In-class **midterm exam**, Thursday 28 Oct
  - Open book, closed note
    - *Note: we will provide you with useful reference material, such as the stack frame cheat sheet, description of assembly commands, etc.*
  - No computers, no smartphones, no internet access
- Midterm review session, Monday 25 Oct
  - 7pm to 8:30pm, location TBA
  - Optional

© 2010 Stephen Chong, Harvard University

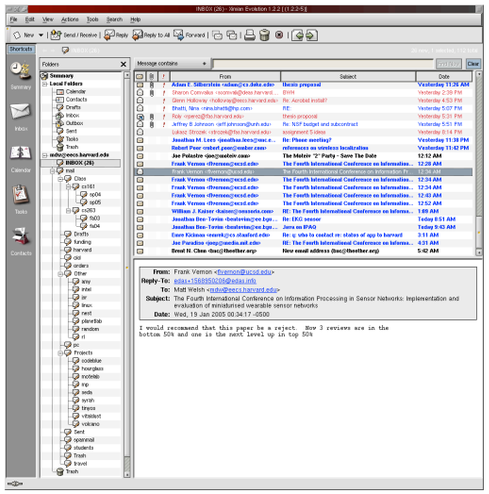
Today

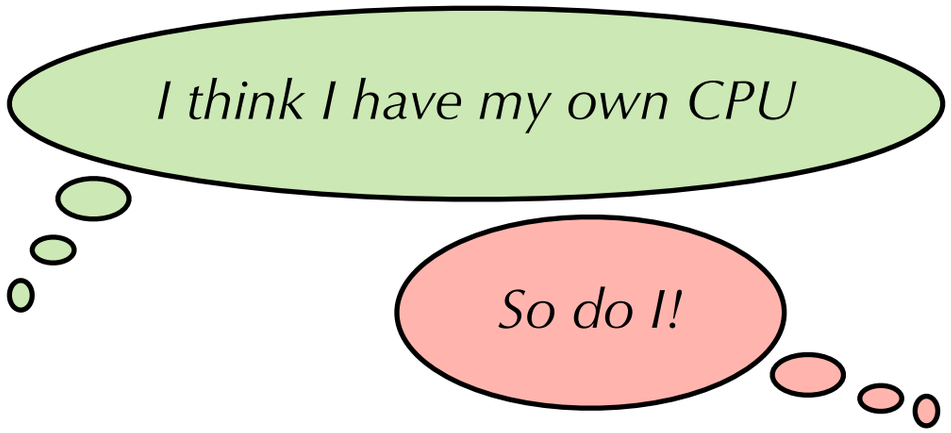
- Running multiple programs at once
- Virtual memory
  - Address spaces
  - Benefits
  - Physical memory as a cache for virtual memory
  - Page tables
  - Page hits and page faults
  - Virtual memory as a tool for memory management
  - Virtual memory as a tool for memory protection
  - Address translation
  - Table Lookaside Buffer (TLB)
  - Example

© 2010 Stephen Chong, Harvard University

Running multiple programs at once

- So far, we have discussed memory management for a single program running on a computer.
- Most modern computers run multiple programs simultaneously.
  - This is called **multiprogramming** or **multitasking**.

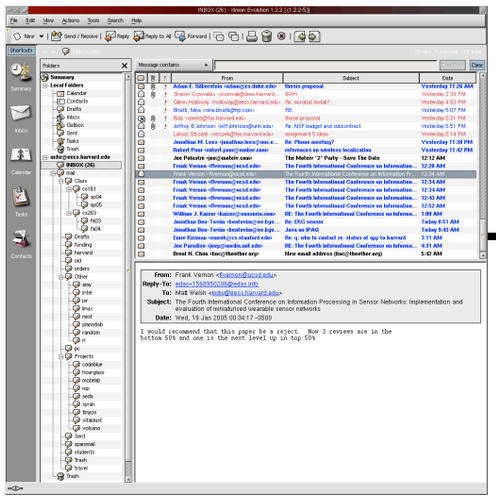




© 2010 Stephen Chong, Harvard University

Timeslicing

- The OS **timeslices** multiple applications on a single CPU
  - Switches between applications extremely rapidly, i.e., 100 times/sec
  - Each switch between two applications is called a context switch.



Operating System

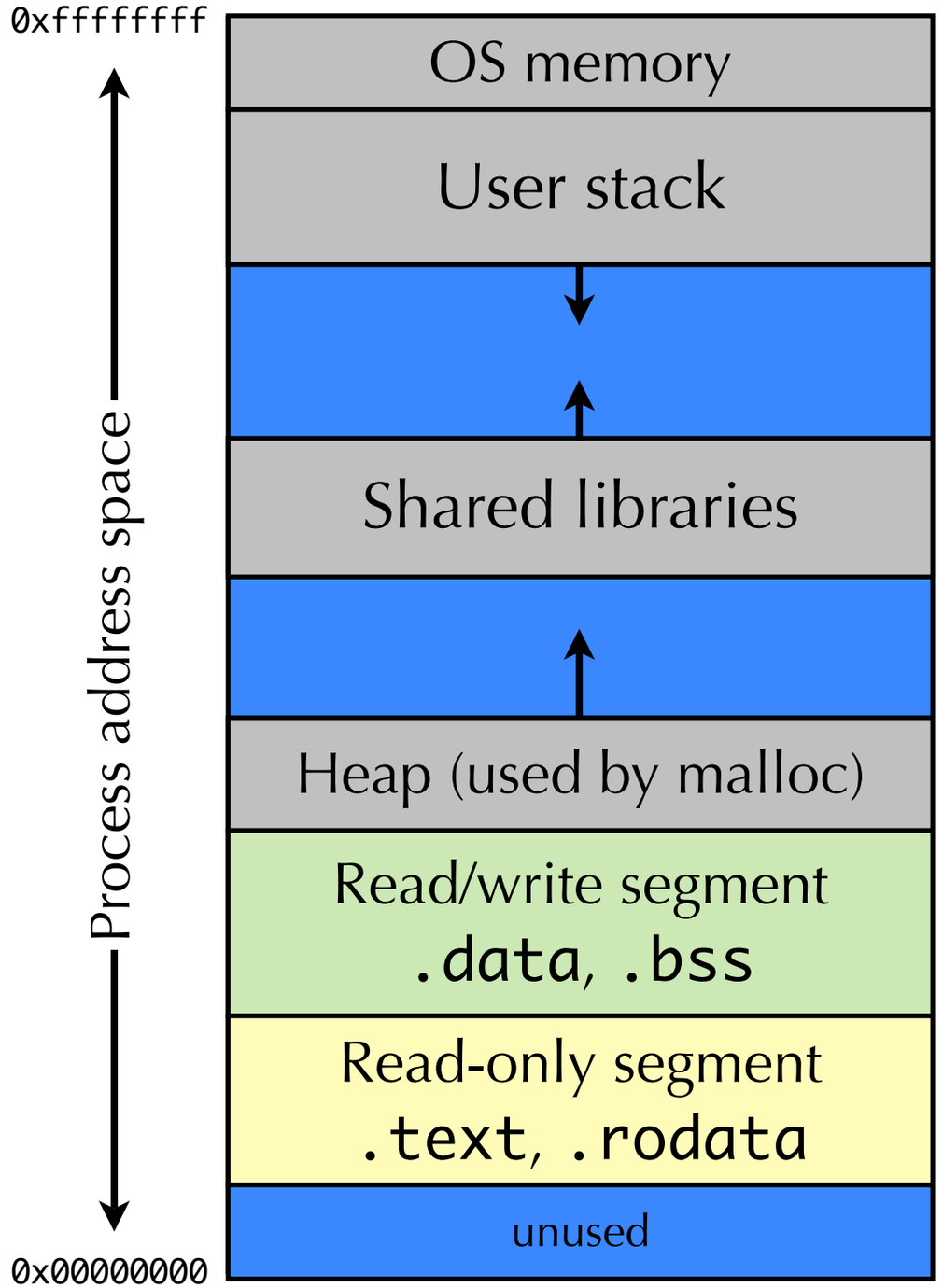
time

Timeslice on single CPU system

© 2010 Stephen Chong, Harvard University

Virtual memory

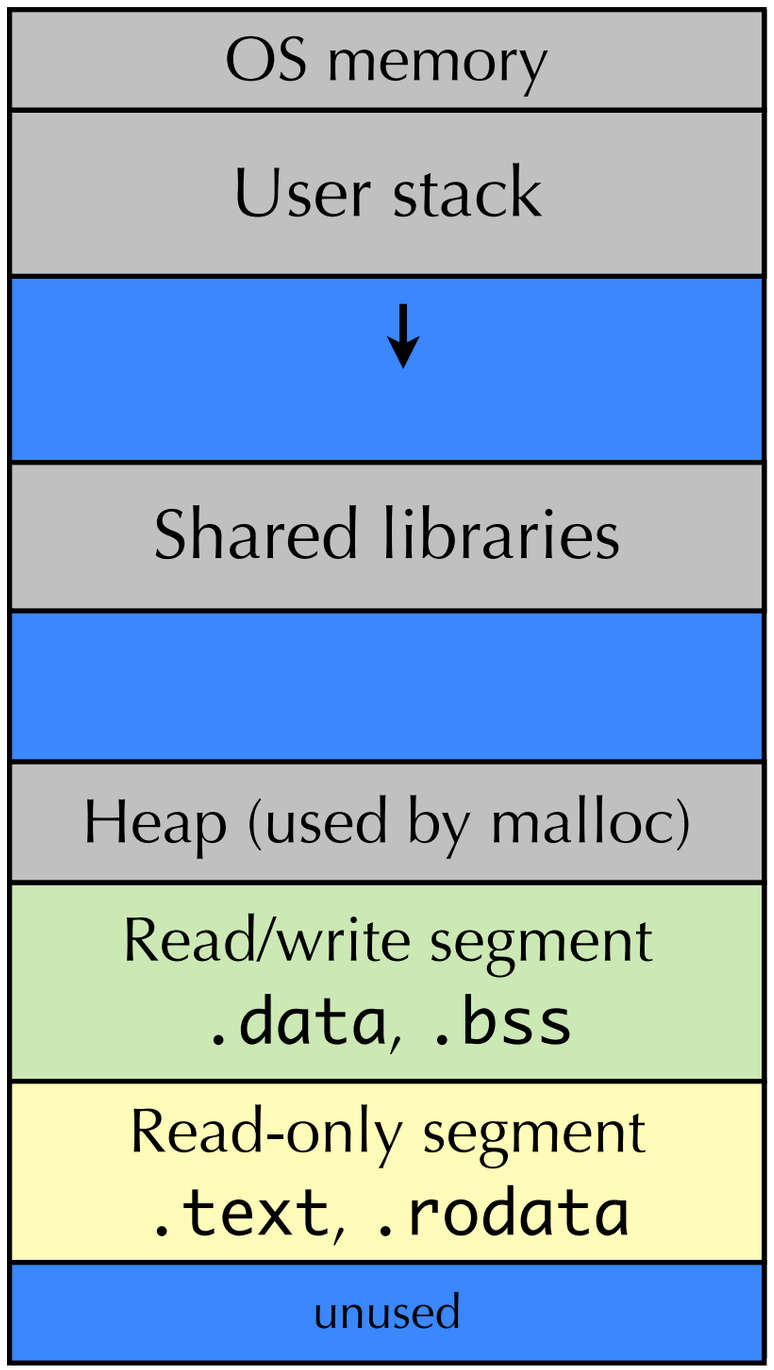
- OS also gives every program illusion of having it's own memory!
- Each program running on the machine is called a **process**.
  - Each process has a **process ID** (a number) and an **owner** – the user ID running the process.
  - UNIX command "ps" prints out info about the running processes.
- Each process has its own **address space**
  - Contents of that process' memory, visible only to that process.
  - 4GB on 32 bit machine, ~16 TB on 64 bit machine



© 2010 Stephen Chong, Harvard University

Problem 1: How Does Everything Fit?

4GB to 16 EB virtual memory

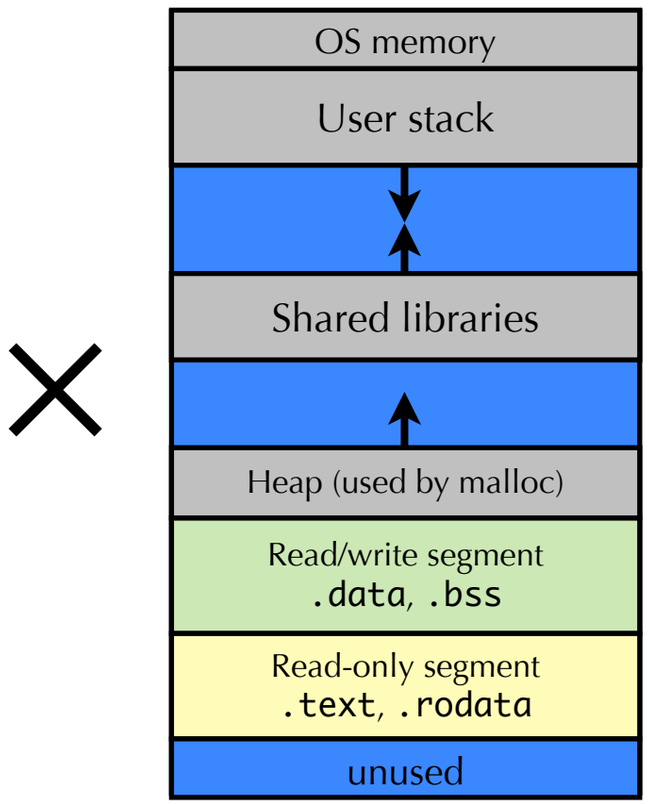


Physical main memory:
Few Gigabytes

© 2010 Stephen Chong, Harvard University

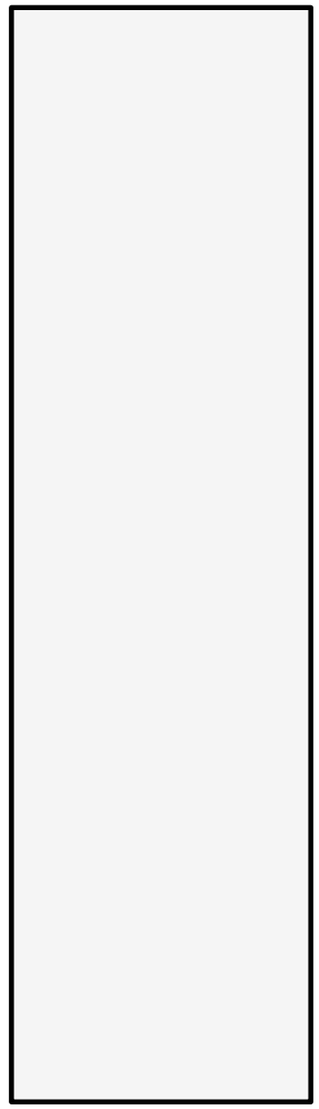
Problem 2: Memory Management

Process 1
Process 2
Process 3
…
Process n



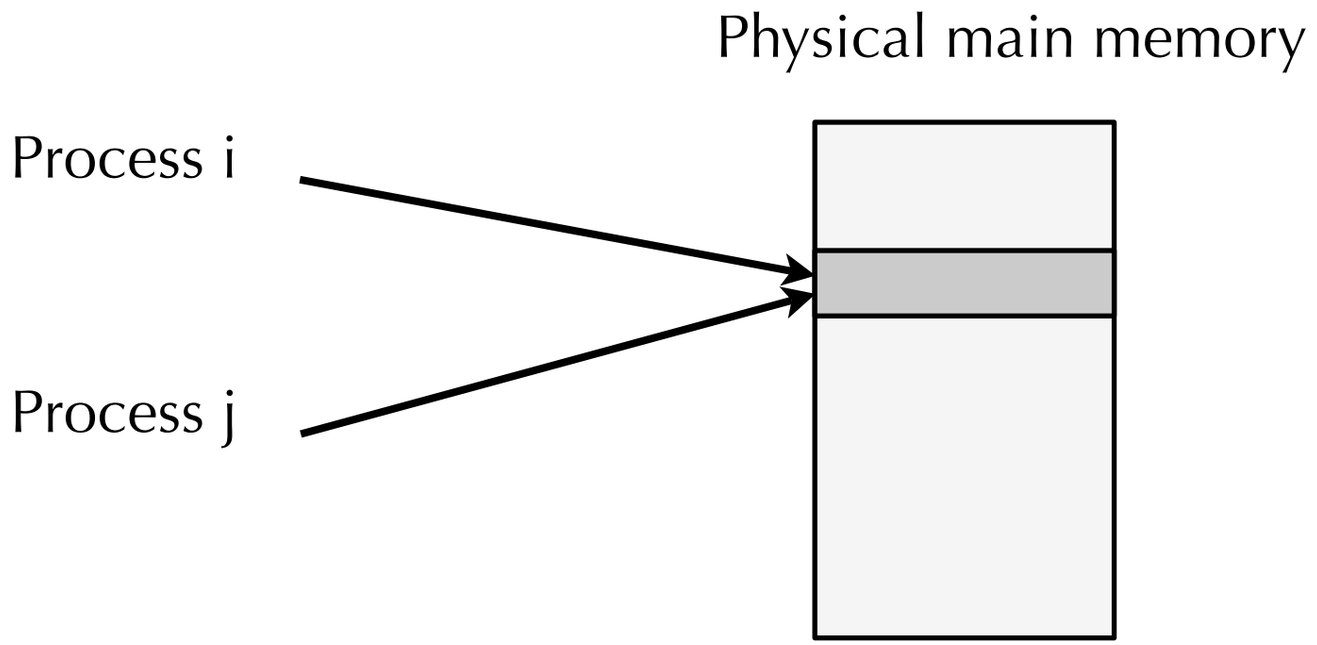
What goes where?

Physical main memory



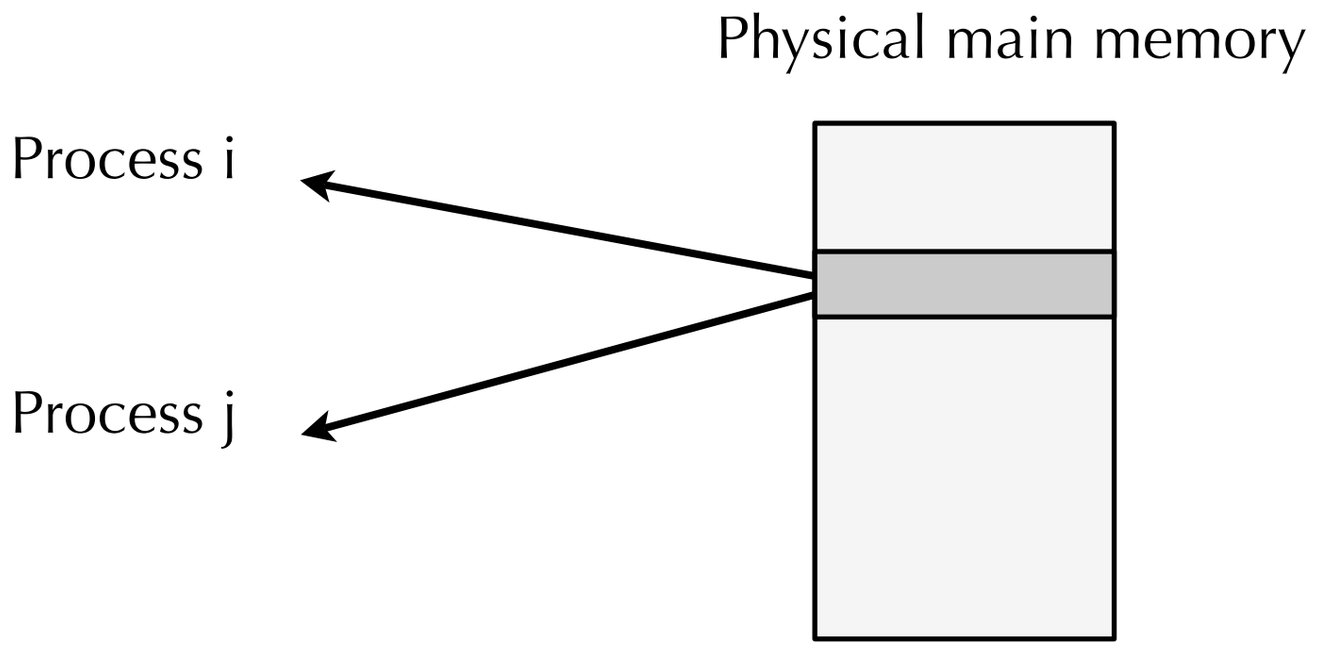
© 2010 Stephen Chong, Harvard University

Problem 3: How To Protect



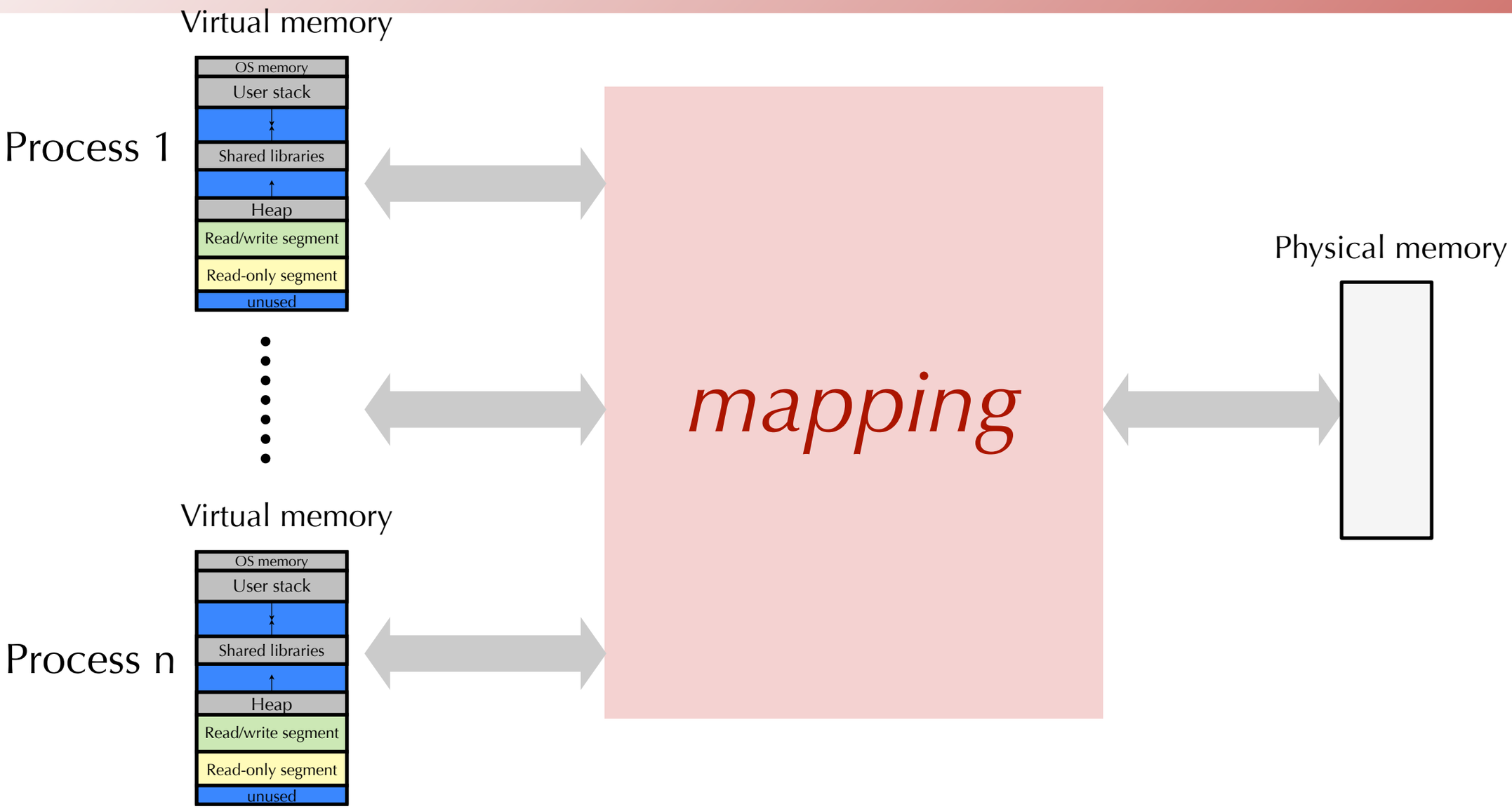
© 2010 Stephen Chong, Harvard University

Problem 4: How To Share



© 2010 Stephen Chong, Harvard University

Solution: Level Of Indirection



- Each process gets its own private memory space
- Solves the previous problems

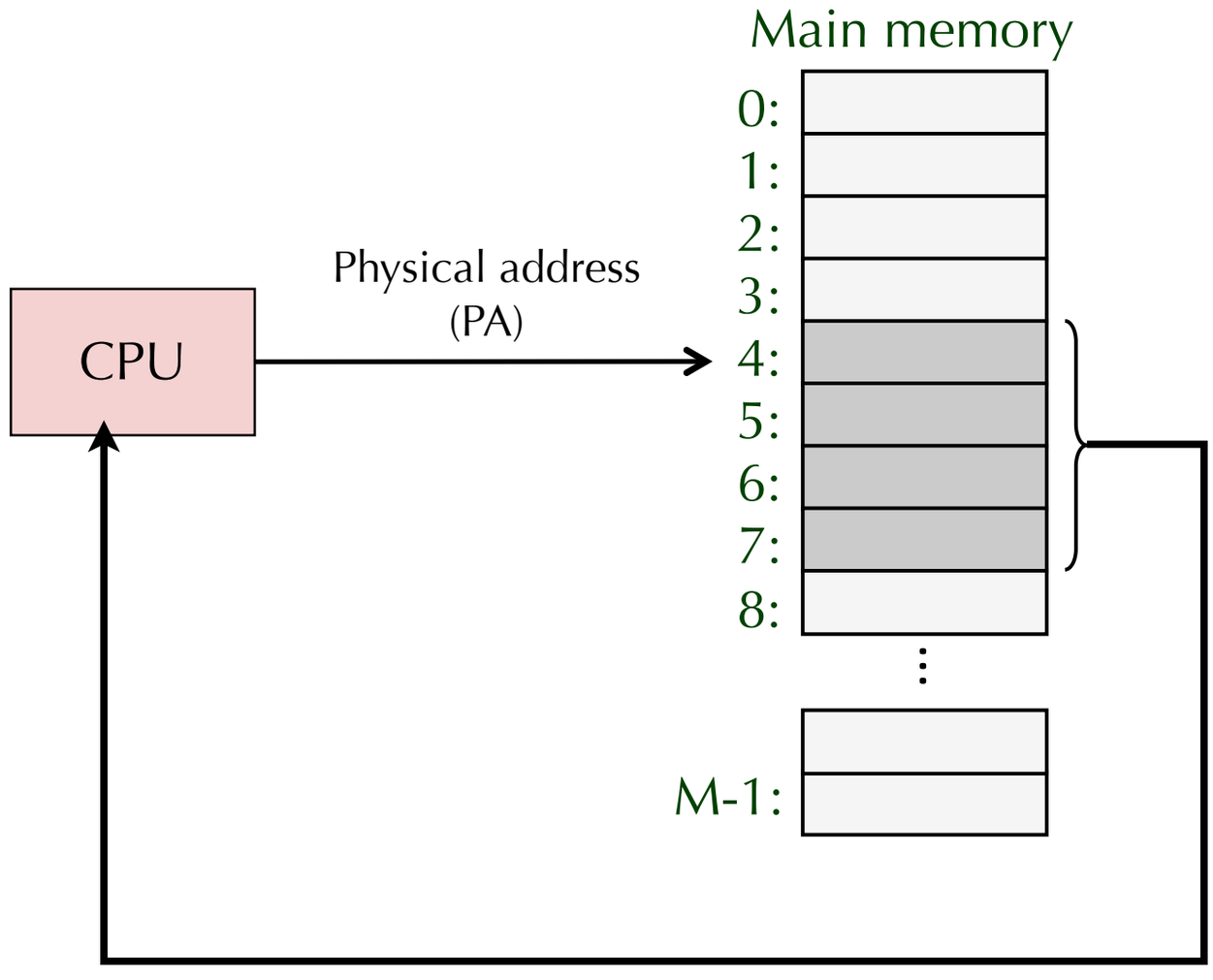
© 2010 Stephen Chong, Harvard University

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
    {0, 1, 2, 3, …, N-1}
- **Physical address space:**
  - Set of $M = 2^m$ physical addresses
    {0, 1, 2, 3, …, M-1}
- Clean distinction between data (bytes) and their attributes (addresses)
- Each object can now have multiple addresses
- Every byte in main memory: one physical address, one (or more) virtual addresses

© 2010 Stephen Chong, Harvard University

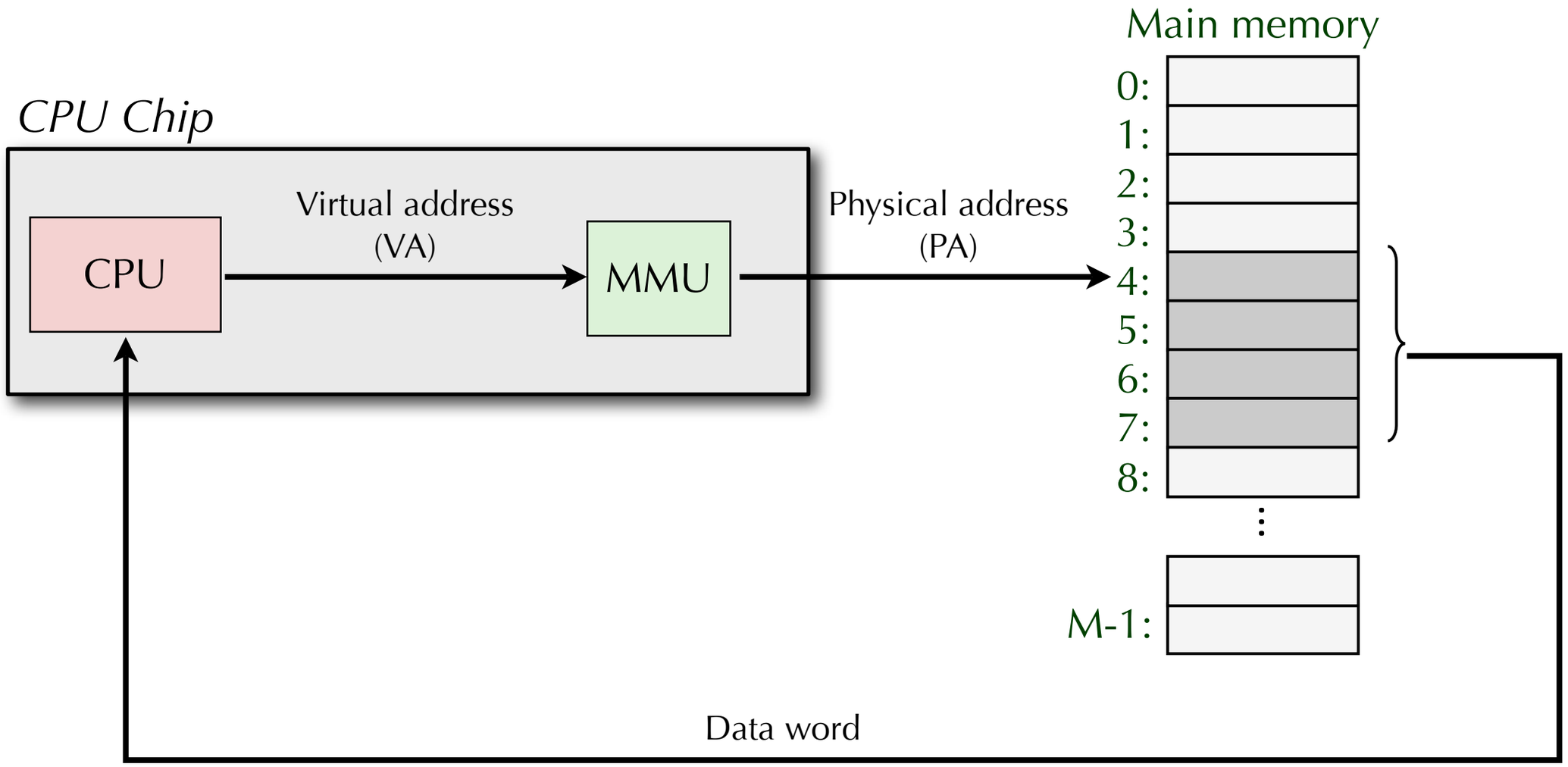
A System Using Physical Addressing



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

© 2010 Stephen Chong, Harvard University

A System Using Virtual Addressing



- Used in all modern desktops, laptops, workstations
- One of the great ideas in computer science

© 2010 Stephen Chong, Harvard University

Benefits of Virtual Memory (VM)

- Efficient use of limited main memory (RAM)
  - Use RAM as a cache for the parts of a virtual address space
    - some non-cached parts stored on disk
    - some (unallocated) non-cached parts stored nowhere
  - Keep only active areas of virtual address space in memory
    - transfer data back and forth as needed
- Simplifies memory management for programmers
  - Each process gets the same full, private linear address space
- Isolates address spaces
  - One process can't interfere with another's memory
    - because they operate in different address spaces
  - User process cannot access privileged information
    - different sections of address spaces have different permissions

© 2010 Stephen Chong, Harvard University

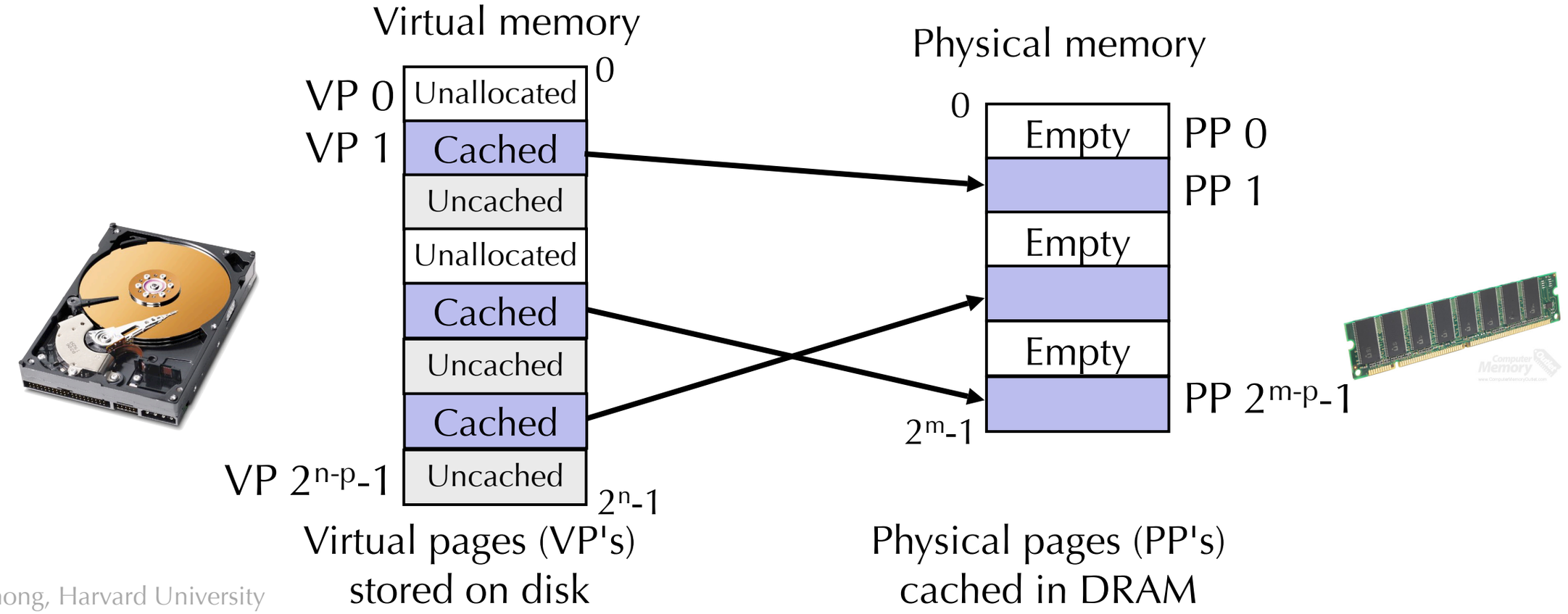
Today

- Running multiple programs at once
- Virtual memory
  - Address spaces
  - Benefits
  - Physical memory as a cache for virtual memory
  - Page tables
  - Page hits and page faults
  - Virtual memory as a tool for memory management
  - Virtual memory as a tool for memory protection
  - Address translation
  - Table Lookaside Buffer (TLB)
  - Example

© 2010 Stephen Chong, Harvard University

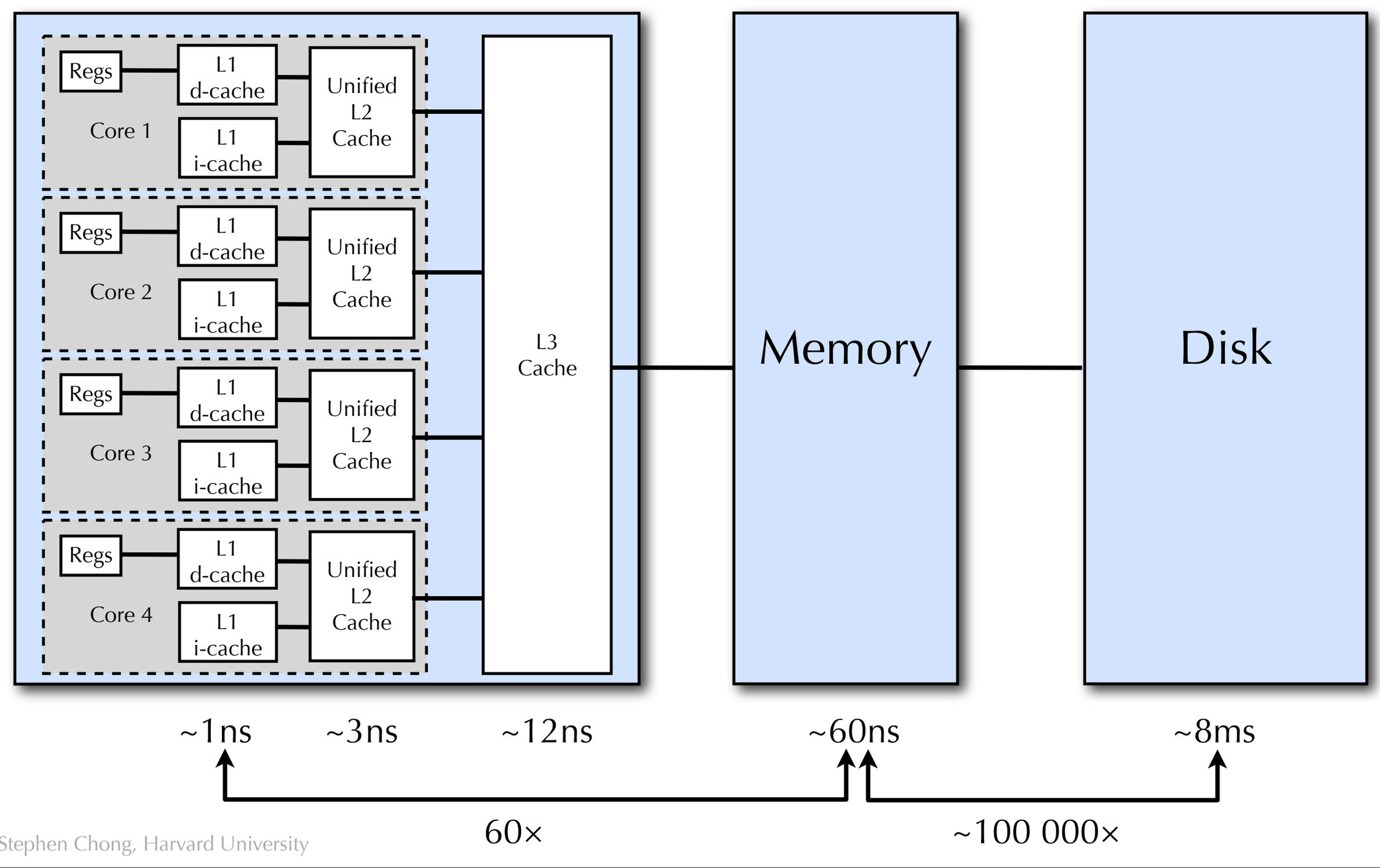
VM as a Tool for Caching

- Virtual memory: array of $N = 2^n$ contiguous bytes
  - think of the array (allocated part) as being stored on disk
- Physical main memory (DRAM) = cache for allocated virtual memory
- Blocks are called **pages**; size = 2^p



© 2010 Stephen Chong, Harvard University

Memory hierarchy: Intel Core i7



© 2010 Stephen Chong, Harvard University

DRAM Cache Organization

- DRAM cache organization driven by the enormous miss penalty
  - DRAM is about 10x slower than SRAM
  - Disk is about 100,000x slower than DRAM
    - For first byte, faster for next byte
- Consequences
  - Large page (block) size: typically 4-8 KB, sometimes 4 MB
  - **Fully associative**
    - Fully associate cache: A cache with only one set
    - Any virtual page can be placed in any physical page
    - 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

© 2010 Stephen Chong, Harvard University

Why does virtual memory work?

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

© 2010 Stephen Chong, Harvard University

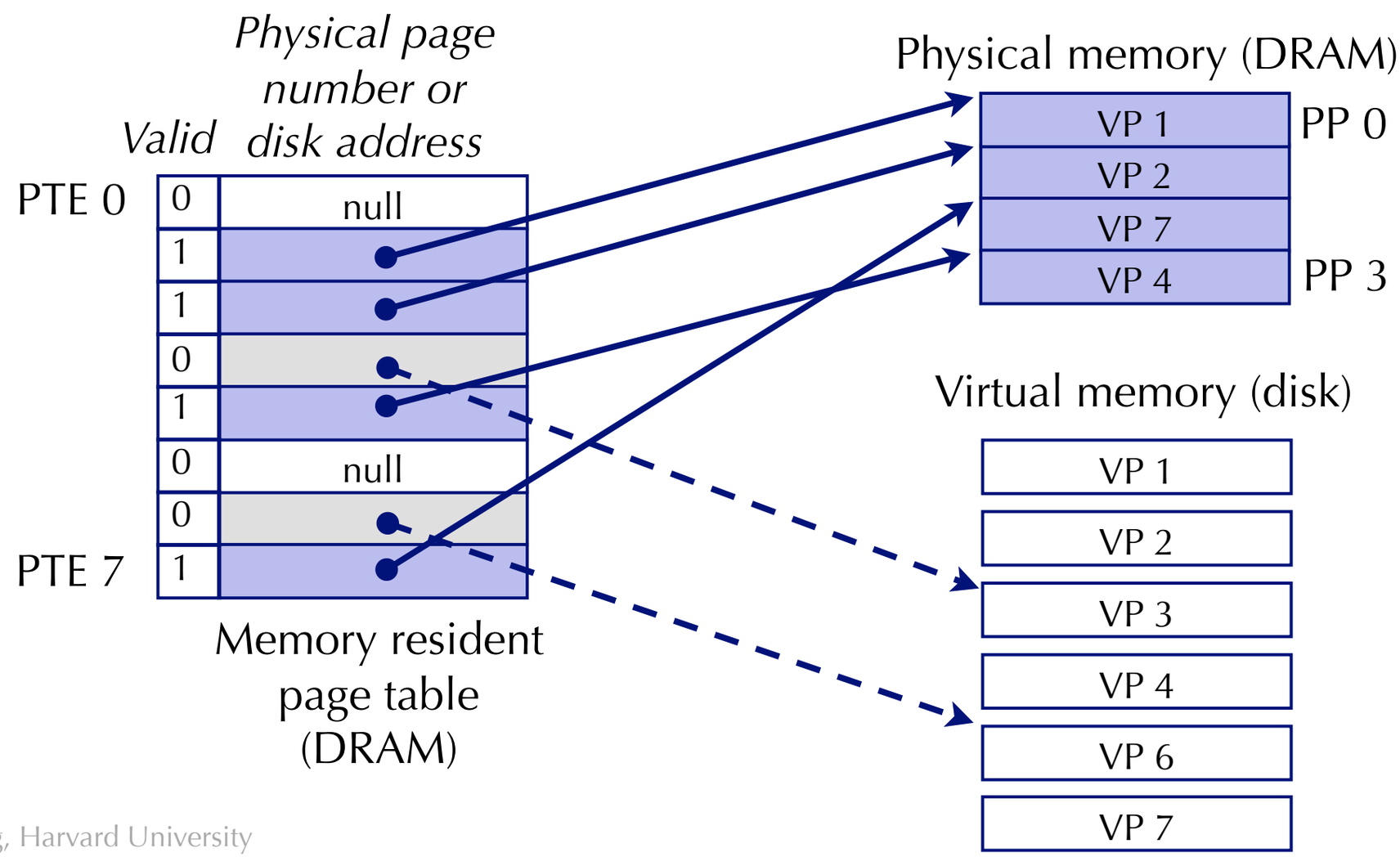
Today

- Running multiple programs at once
- Virtual memory
  - Address spaces
  - Benefits
  - Physical memory as a cache for virtual memory
  - Page tables
  - Page hits and page faults
  - Virtual memory as a tool for memory management
  - Virtual memory as a tool for memory protection
  - Address translation
  - Table Lookaside Buffer (TLB)
  - Example

© 2010 Stephen Chong, Harvard University

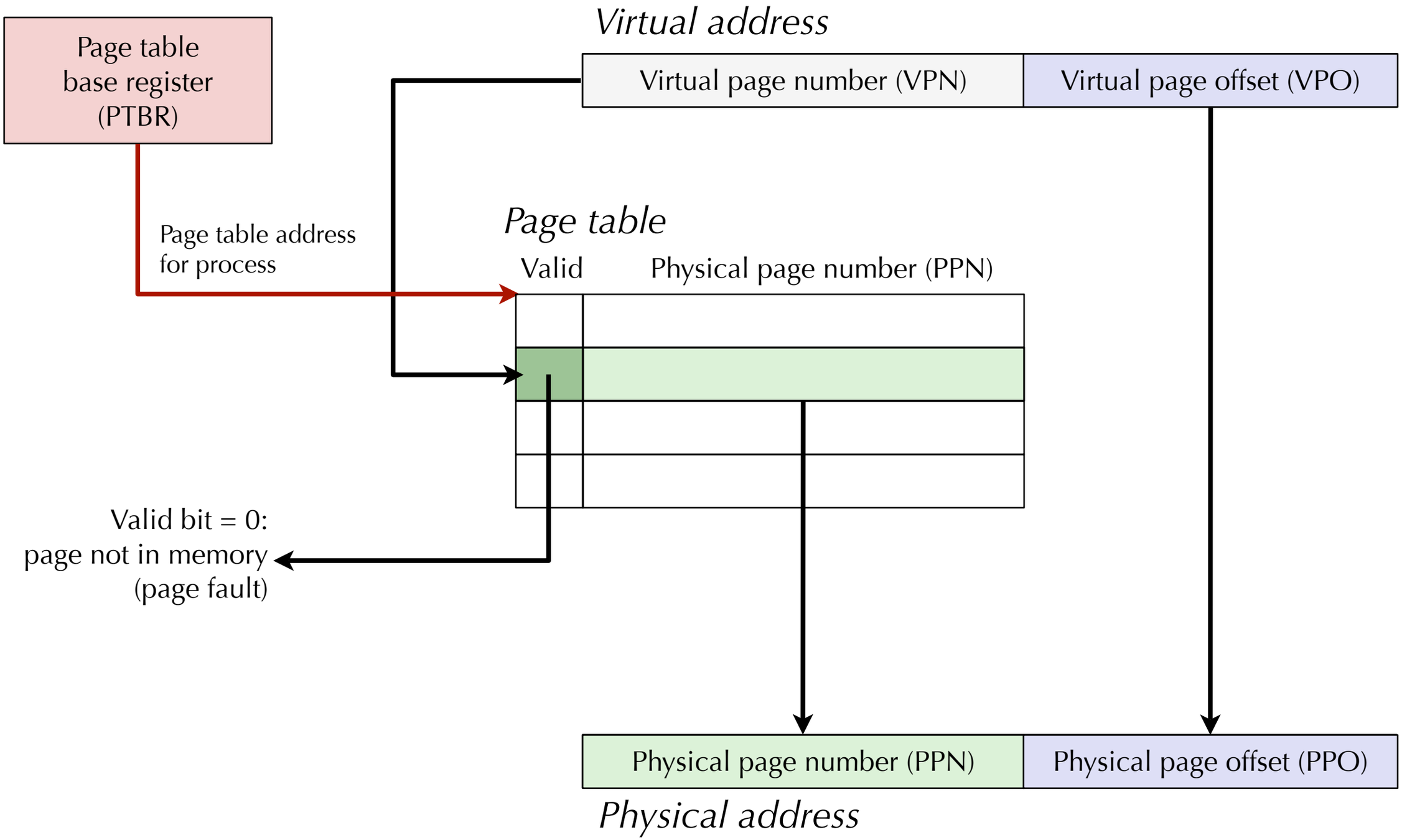
Page Tables

- A **page table** is an array of page table entries (PTEs) that maps virtual pages to physical pages. Here: 8 VPs
  - Per-process kernel data structure in DRAM



© 2010 Stephen Chong, Harvard University

Address Translation With a Page Table



© 2010 Stephen Chong, Harvard University

Page table size

- How big is the page table for a process?
- Well ... we need one PTE per page.
- Say we have a 32-bit address space, and the page size is 4KB
- How many pages?
  - 2^{32} = 4 GB total memory
  - 4GB / 4KB = 1,048,576 (= 1M) pages = 1M PTEs
- How big is a PTE?
  - Depends on CPU architecture. On x86 it is 4 bytes
- Page table size for one process: 1M PTEs × 4 bytes/PTE = 4 MB
- For 100 processes that's 400 MB of memory just for page tables!!!
- Solution: swap page tables out to disk...

© 2010 Stephen Chong, Harvard University

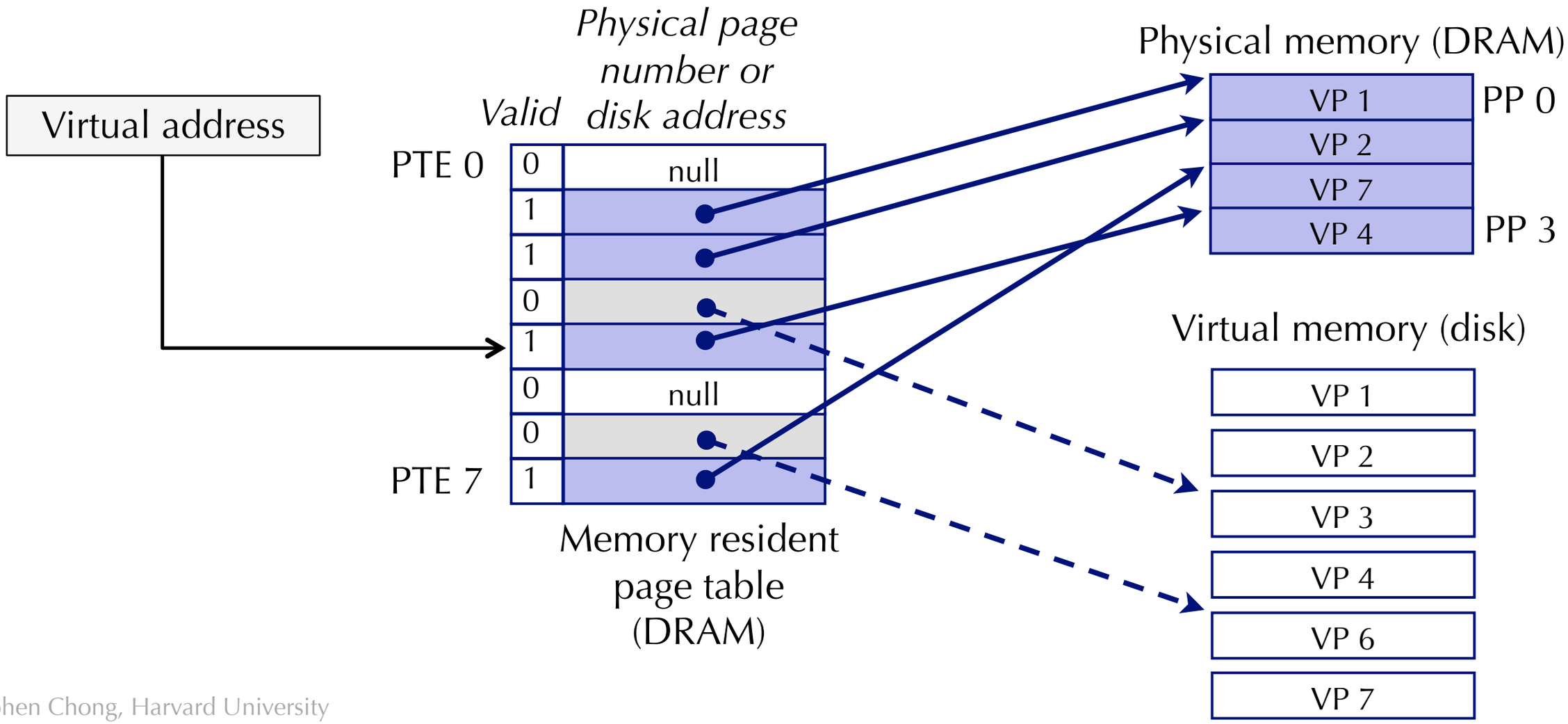
Today

- Running multiple programs at once
- Virtual memory
  - Address spaces
  - Benefits
  - Physical memory as a cache for virtual memory
  - Page tables
  - Page hits and page faults
  - Virtual memory as a tool for memory management
  - Virtual memory as a tool for memory protection
  - Address translation
  - Table Lookaside Buffer (TLB)
  - Example

© 2010 Stephen Chong, Harvard University

Page Hit

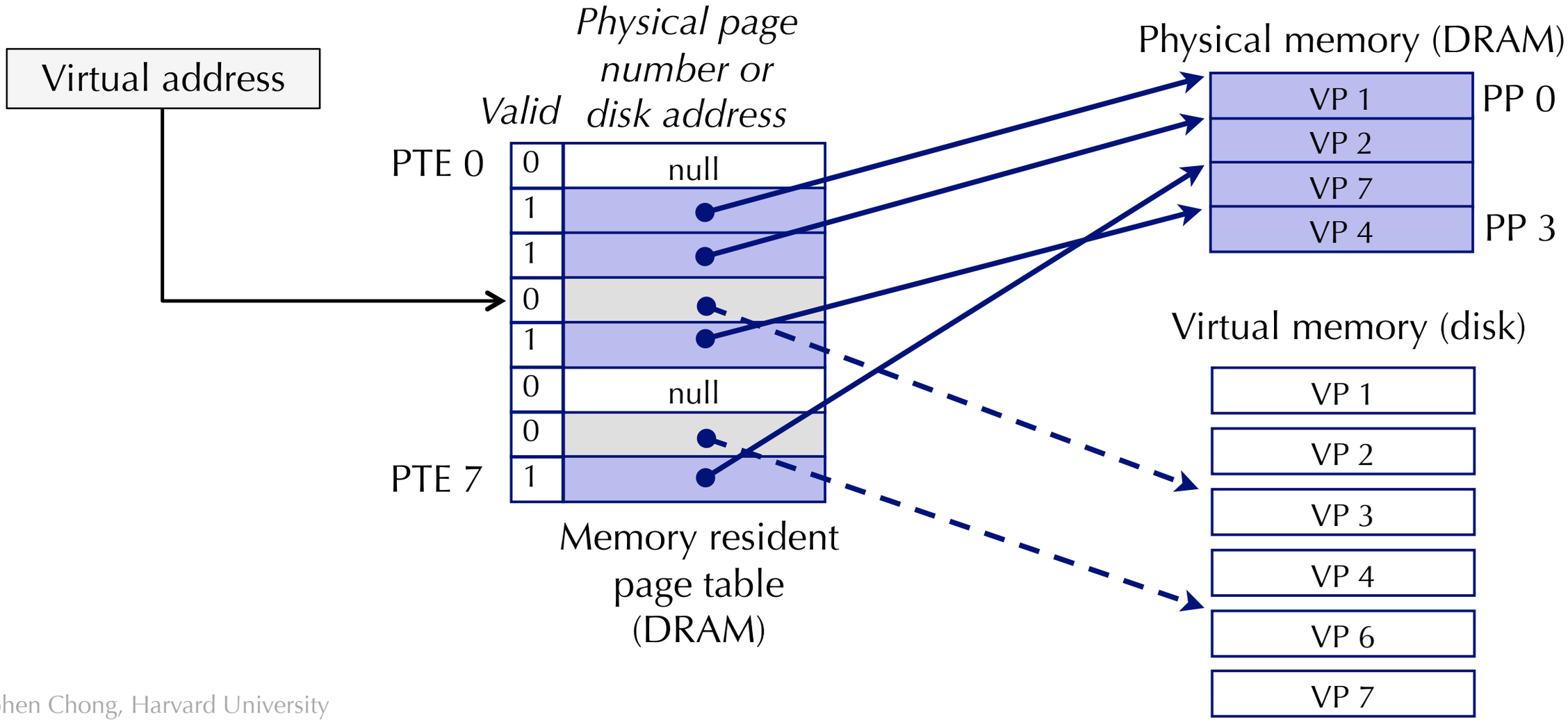
- **Page hit**: reference to VM word that is in physical memory



© 2010 Stephen Chong, Harvard University

Page fault

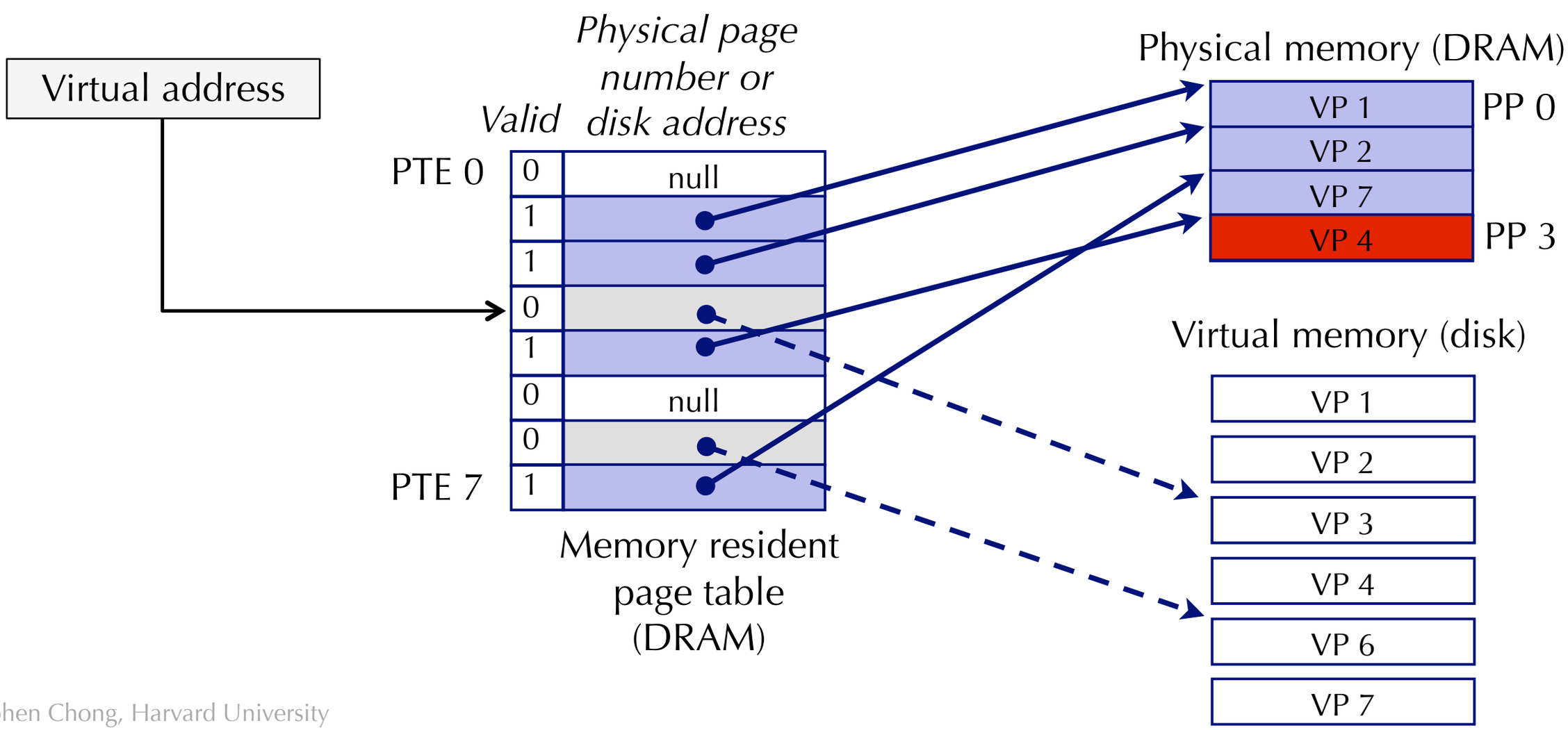
- **Page fault**: reference to VM word that is not in physical memory
  - i.e., invalid entry in page table
  - Could be bad memory address, or page table currently on disk



© 2010 Stephen Chong, Harvard University

Handling page faults

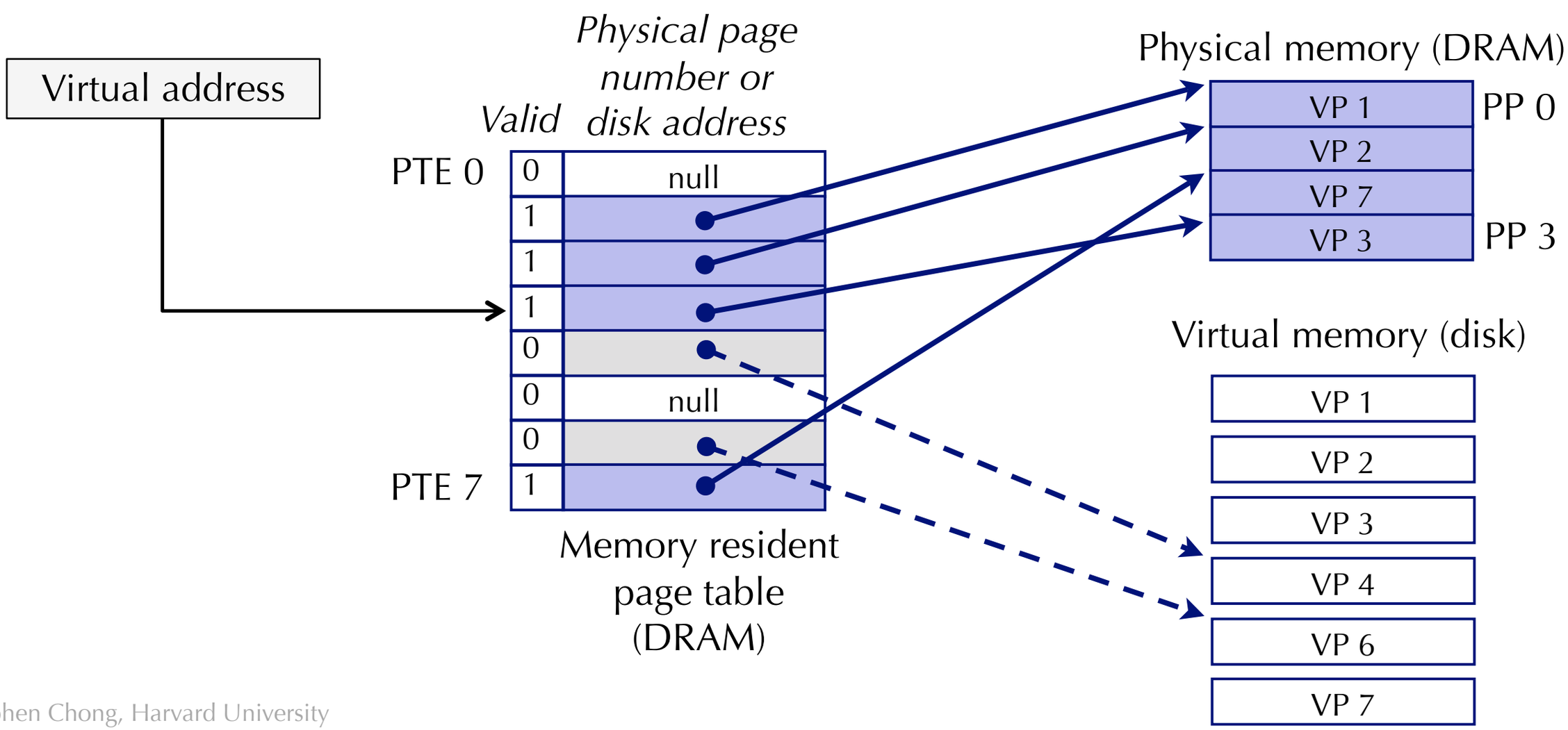
- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)



© 2010 Stephen Chong, Harvard University

Handling page faults

- 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!



© 2010 Stephen Chong, Harvard University

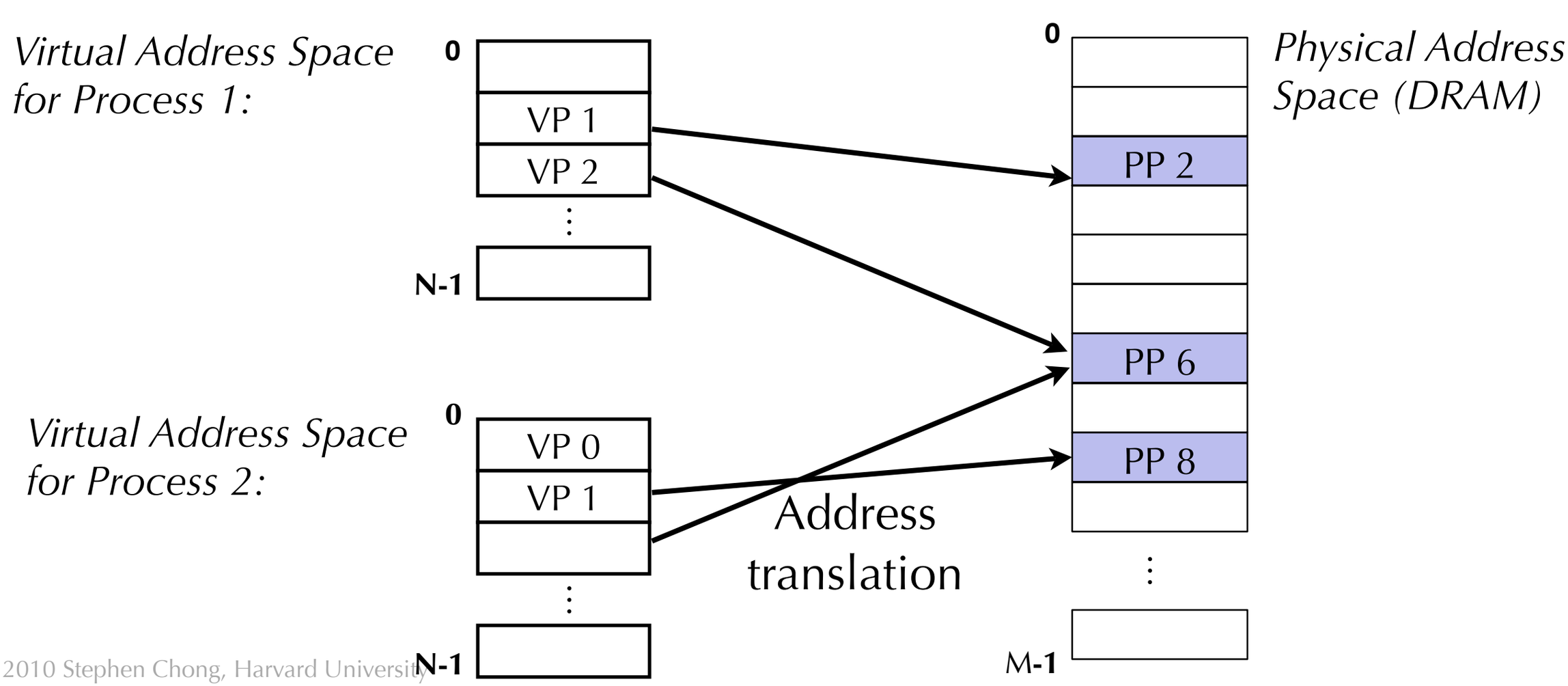
Today

- Running multiple programs at once
- Virtual memory
  - Address spaces
  - Benefits
  - Physical memory as a cache for virtual memory
  - Page tables
  - Page hits and page faults
  - Virtual memory as a tool for memory management
  - Virtual memory as a tool for memory protection
  - Address translation
  - Table Lookaside Buffer (TLB)
  - Example

© 2010 Stephen Chong, Harvard University

VM as a Tool for Memory Management

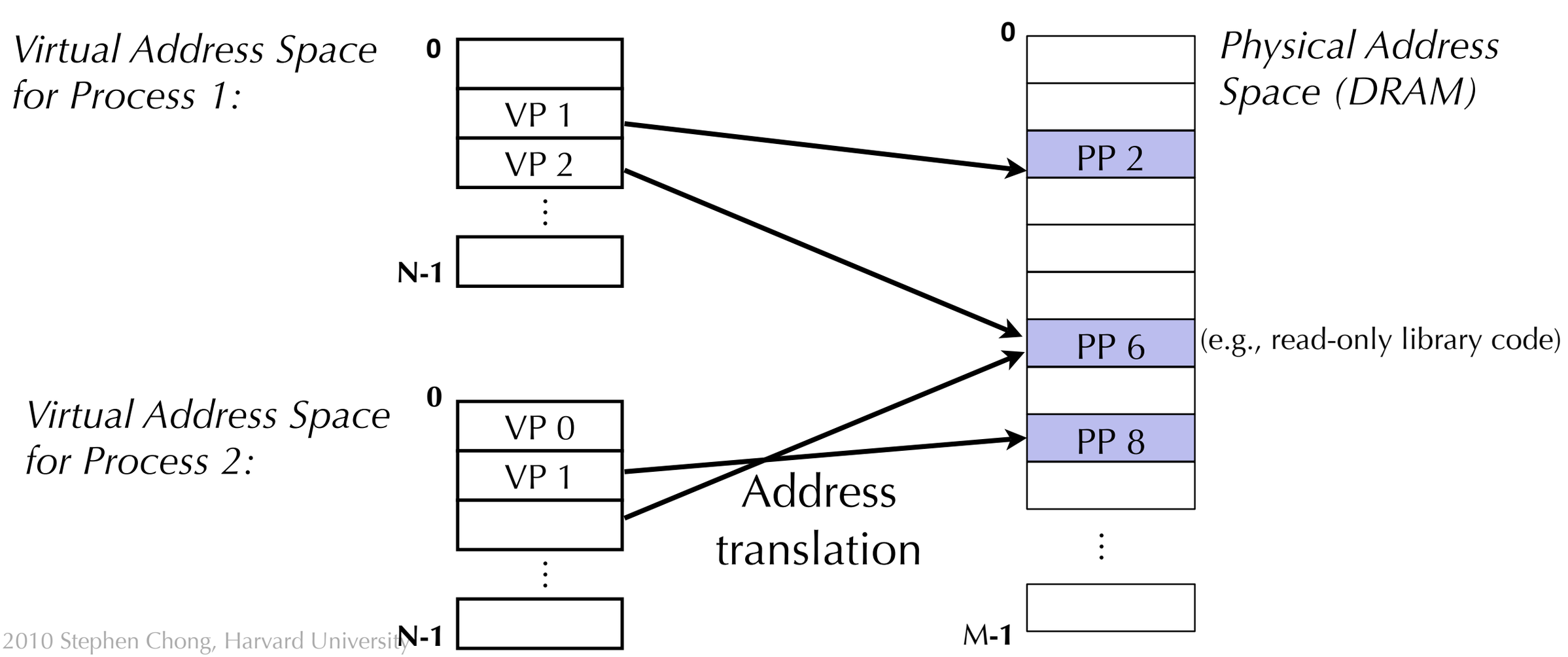
- Key idea: each process has its own virtual address space
  - It can view memory as a simple linear array
  - Mapping function scatters addresses through physical memory
    - Well chosen mappings simplify memory allocation and management



© 2010 Stephen Chong, Harvard University

VM as a Tool for Memory Management

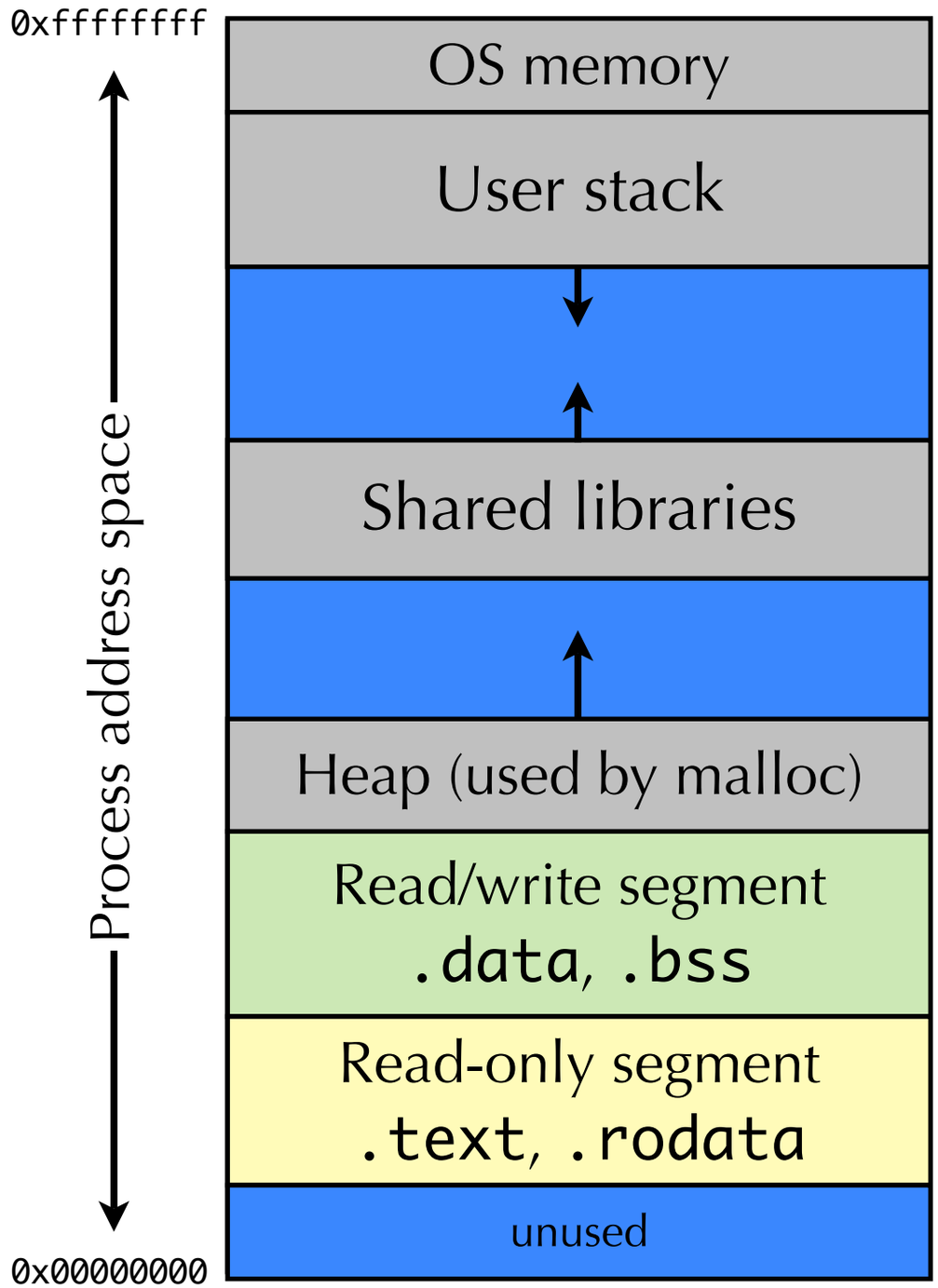
- Memory allocation
  - Each virtual page can be mapped to any physical page
  - A virtual page can be stored in different physical pages at different times
- Sharing code and data among processes
  - Map virtual pages to the same physical page (here: PP 6)



© 2010 Stephen Chong, Harvard University

Simplifying linking

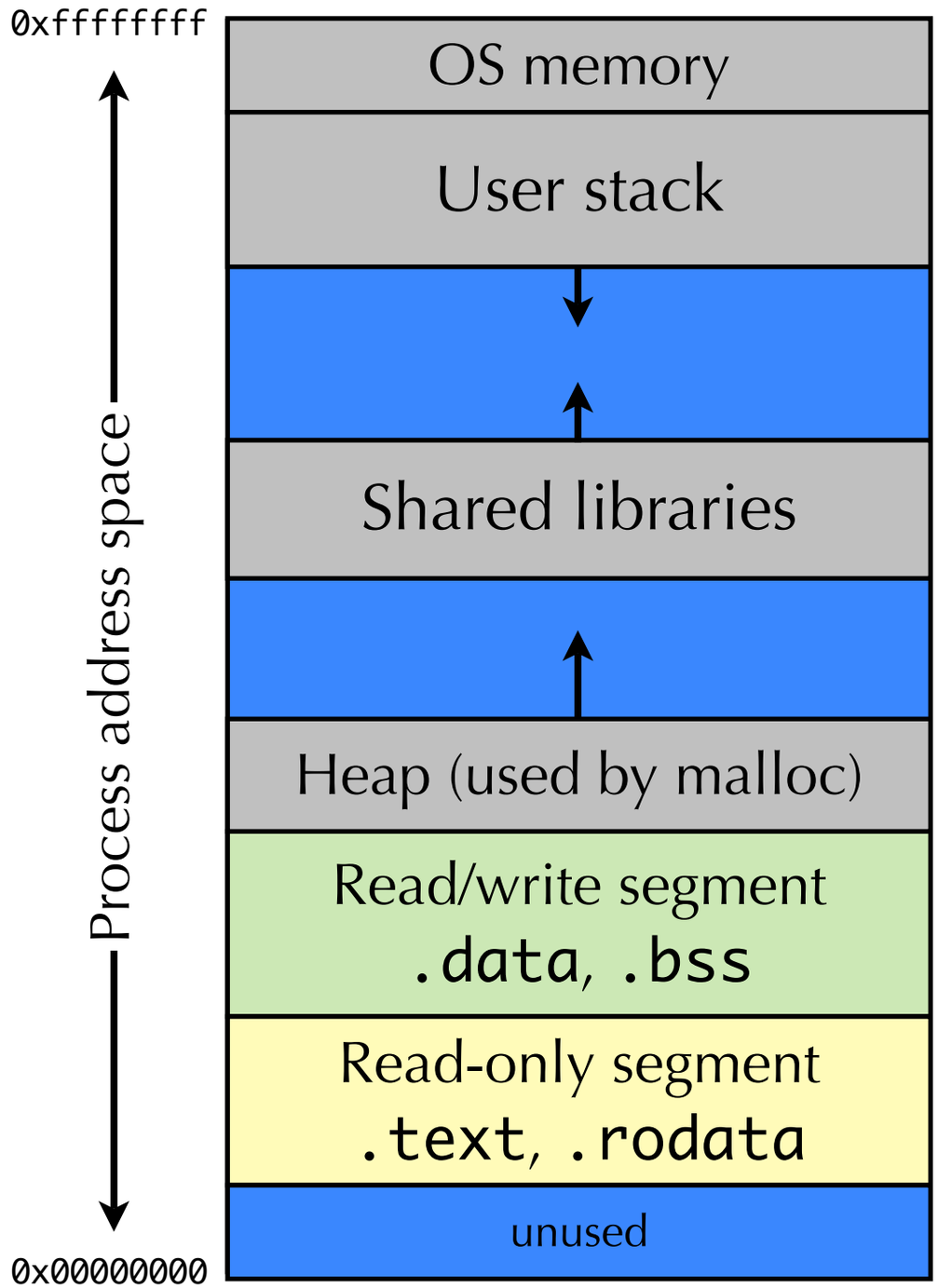
- Linking
  - Each program has similar virtual address space
  - Code, stack, and shared libraries always start at the same address



© 2010 Stephen Chong, Harvard University

Simplifying loading

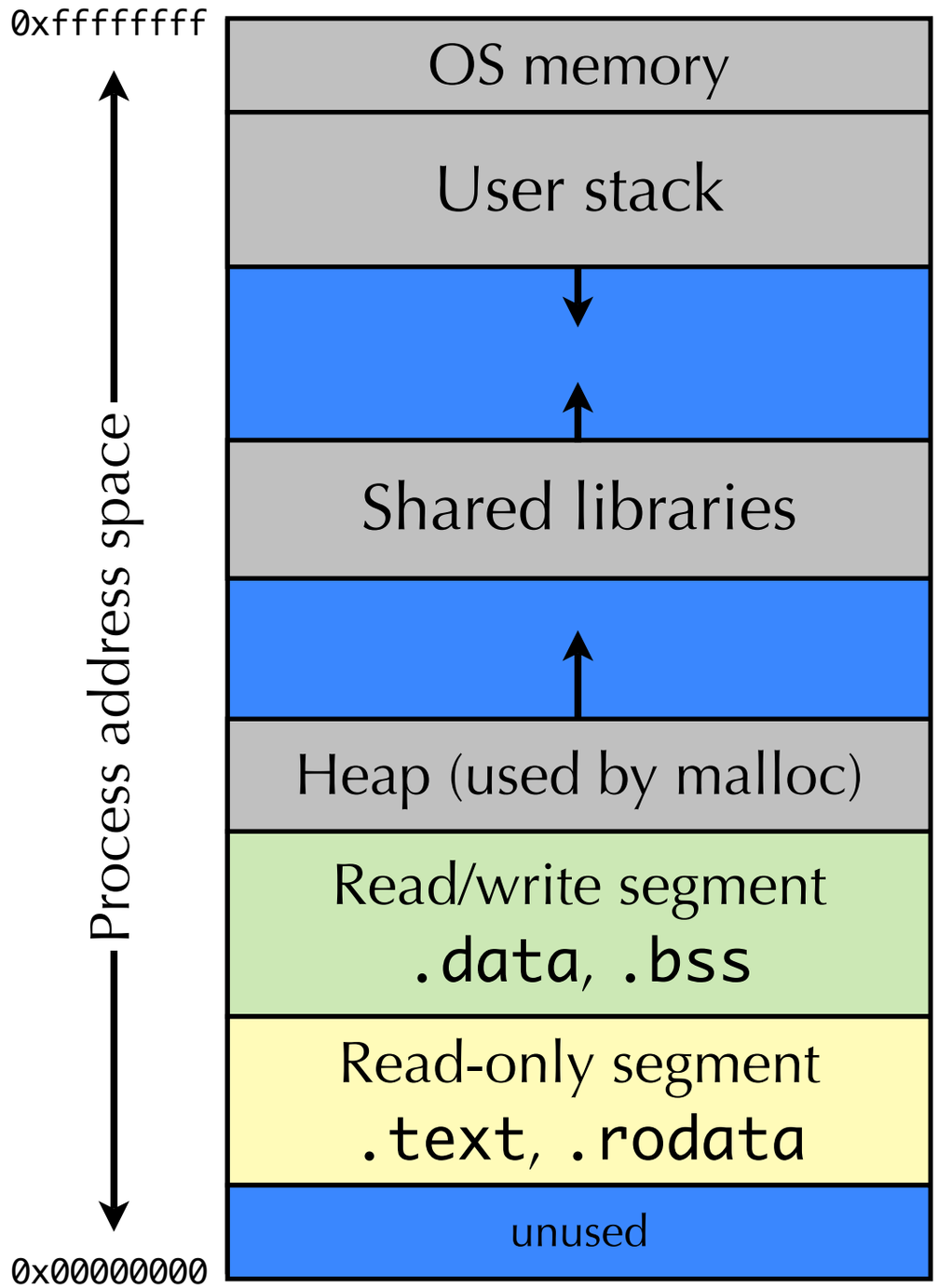
- Doesn't make sense to immediately load all of a program into physical memory
  - Lots of code and data may not be used
  - E.g., rarely used functionality
- Initially, page table maps data, text, rodata, etc., segments to disk
  - i.e., PTE are marked as invalid
- As segments are used (e.g., code accessed for the first time), page fault will bring them into physical memory
- **Demand paging**: pages copied into memory only when they are needed



© 2010 Stephen Chong, Harvard University

Demand-zero pages

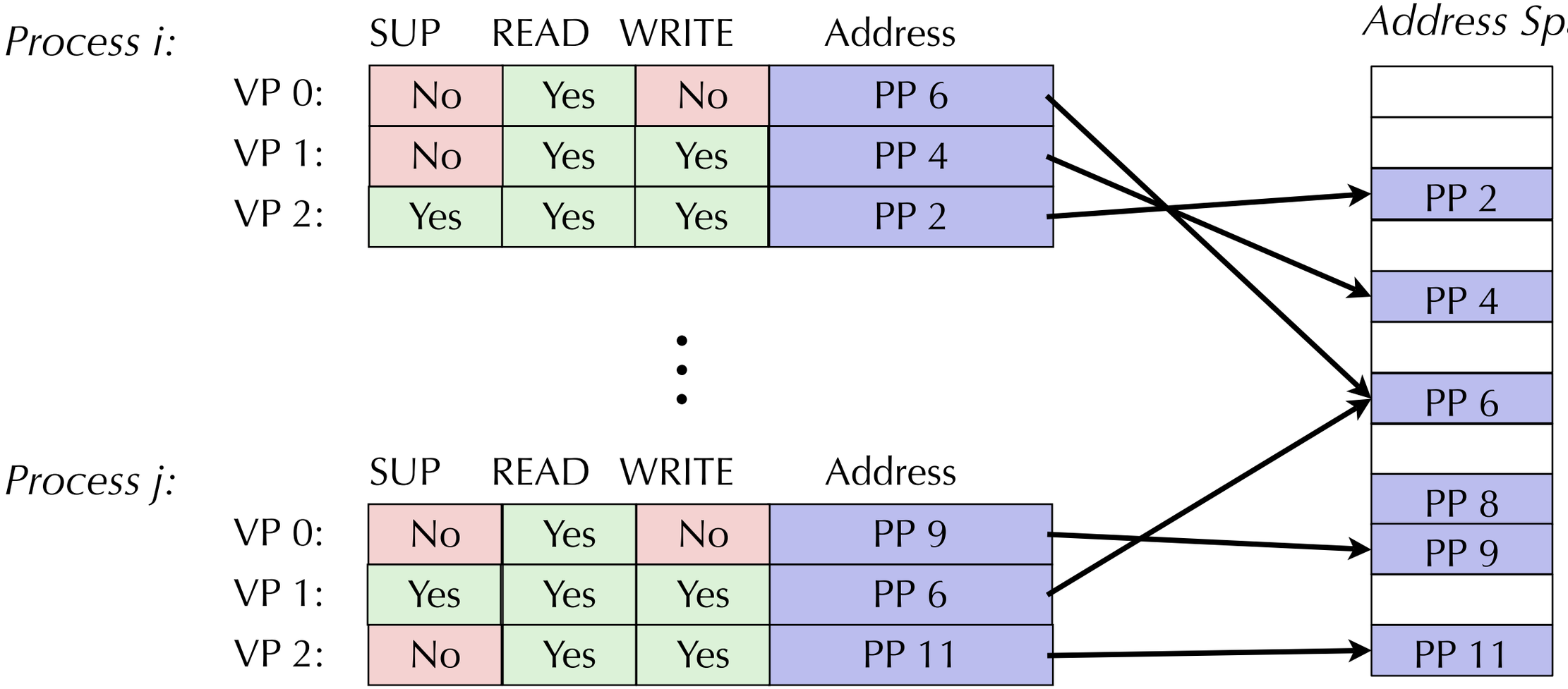
- When page from bss segment (and new memory for the heap, stack, etc.) faulted for the first time, physical page of all zeros allocated.
  - Once page is written, like any other page.
  - **"Demand-zero" pages**



© 2010 Stephen Chong, Harvard University

VM as a Tool for Memory Protection

- Extend PTEs with permission bits
- Page fault handler checks these before remapping



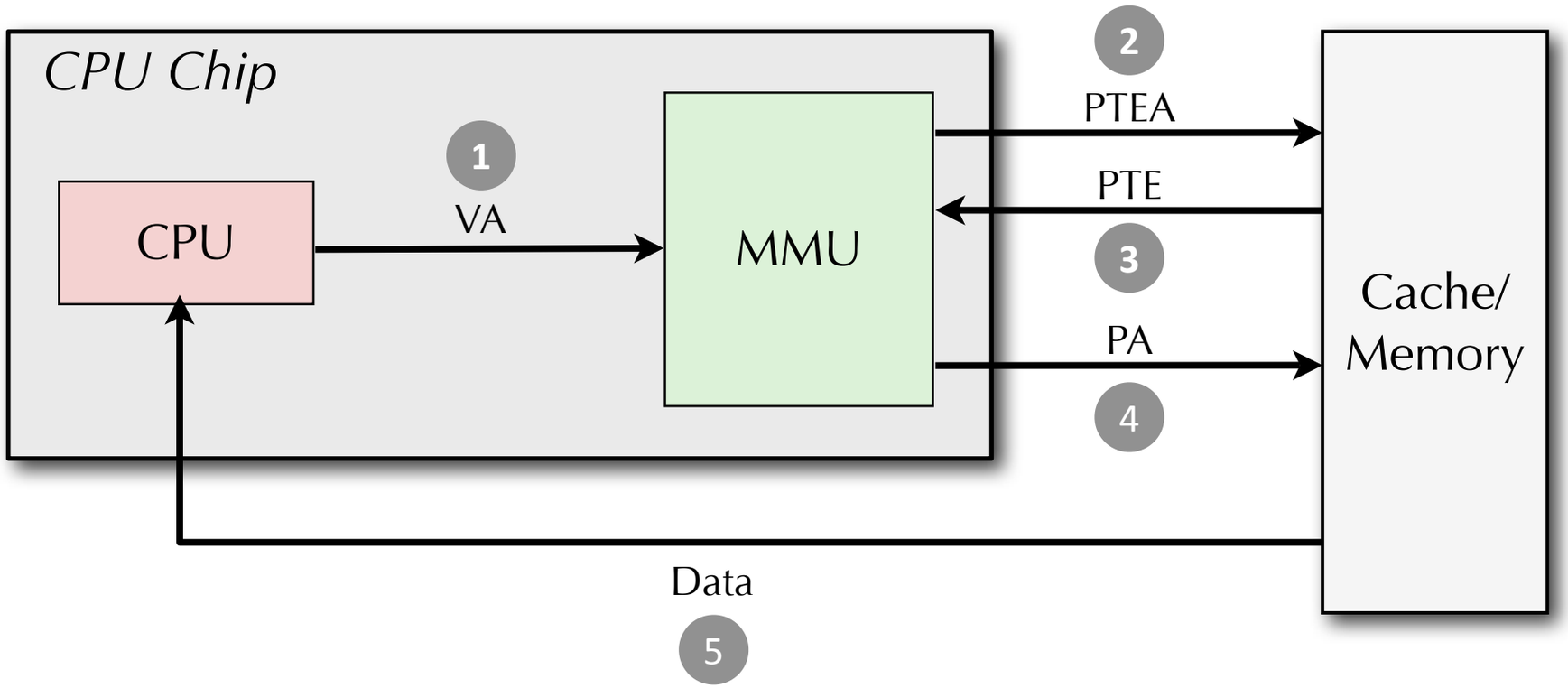
© 2010 Stephen Chong, Harvard University

Today

- Running multiple programs at once
- Virtual memory
  - Address spaces
  - Benefits
  - Physical memory as a cache for virtual memory
  - Page tables
  - Page hits and page faults
  - Virtual memory as a tool for memory management
  - Virtual memory as a tool for memory protection
  - Address translation
  - Table Lookaside Buffer (TLB)
  - Example

© 2010 Stephen Chong, Harvard University

Address Translation: Page Hit



1) Processor sends virtual address to MMU

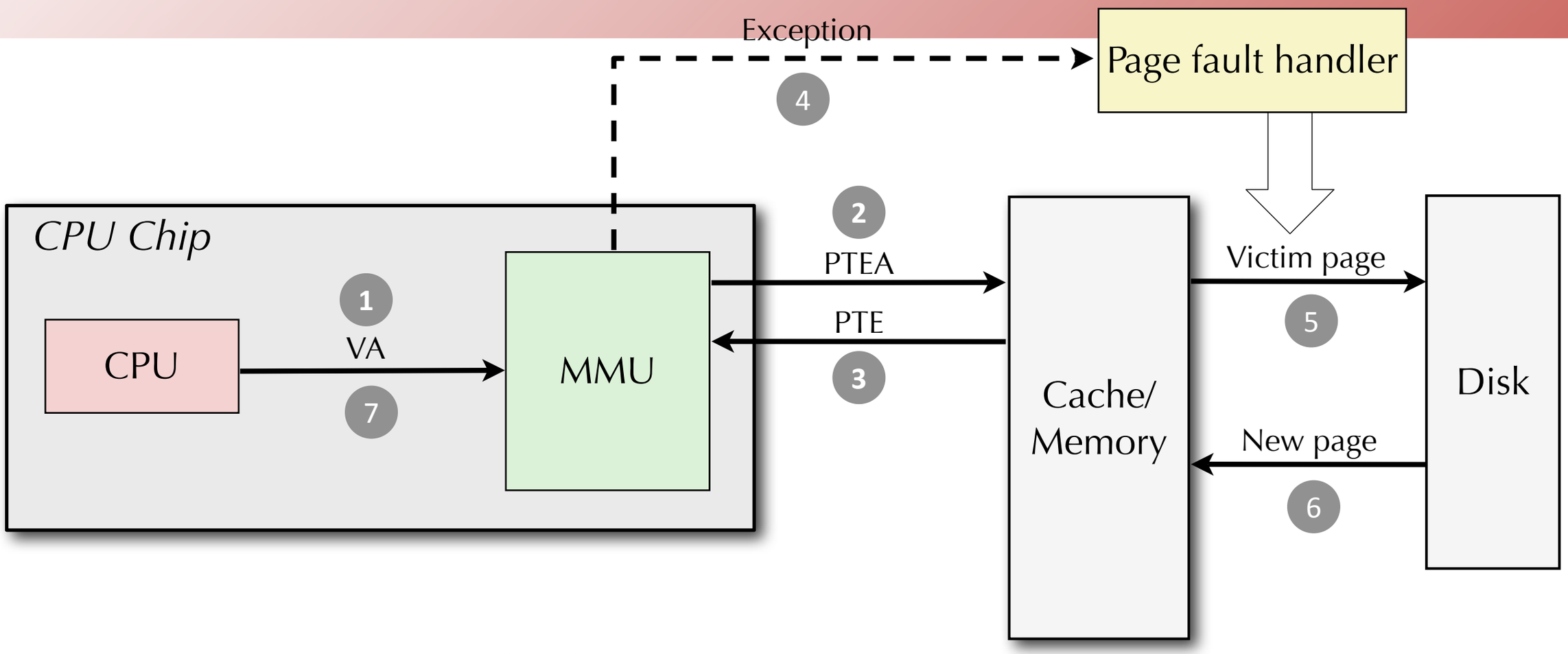
2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

© 2010 Stephen Chong, Harvard University

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

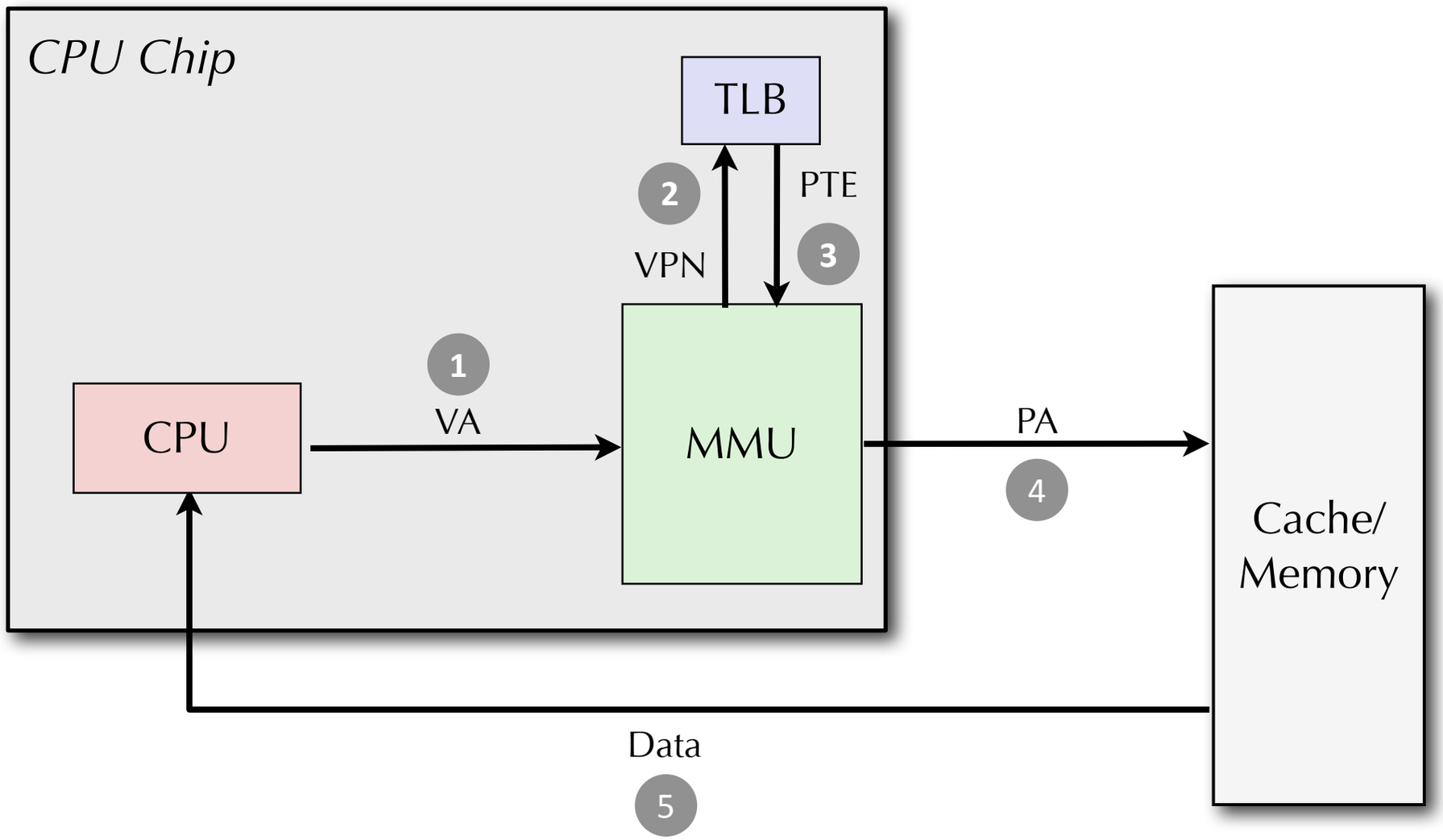
© 2010 Stephen Chong, Harvard University

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 1-cycle delay
- Solution: **Translation Lookaside Buffer** (**TLB**)
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages

© 2010 Stephen Chong, Harvard University

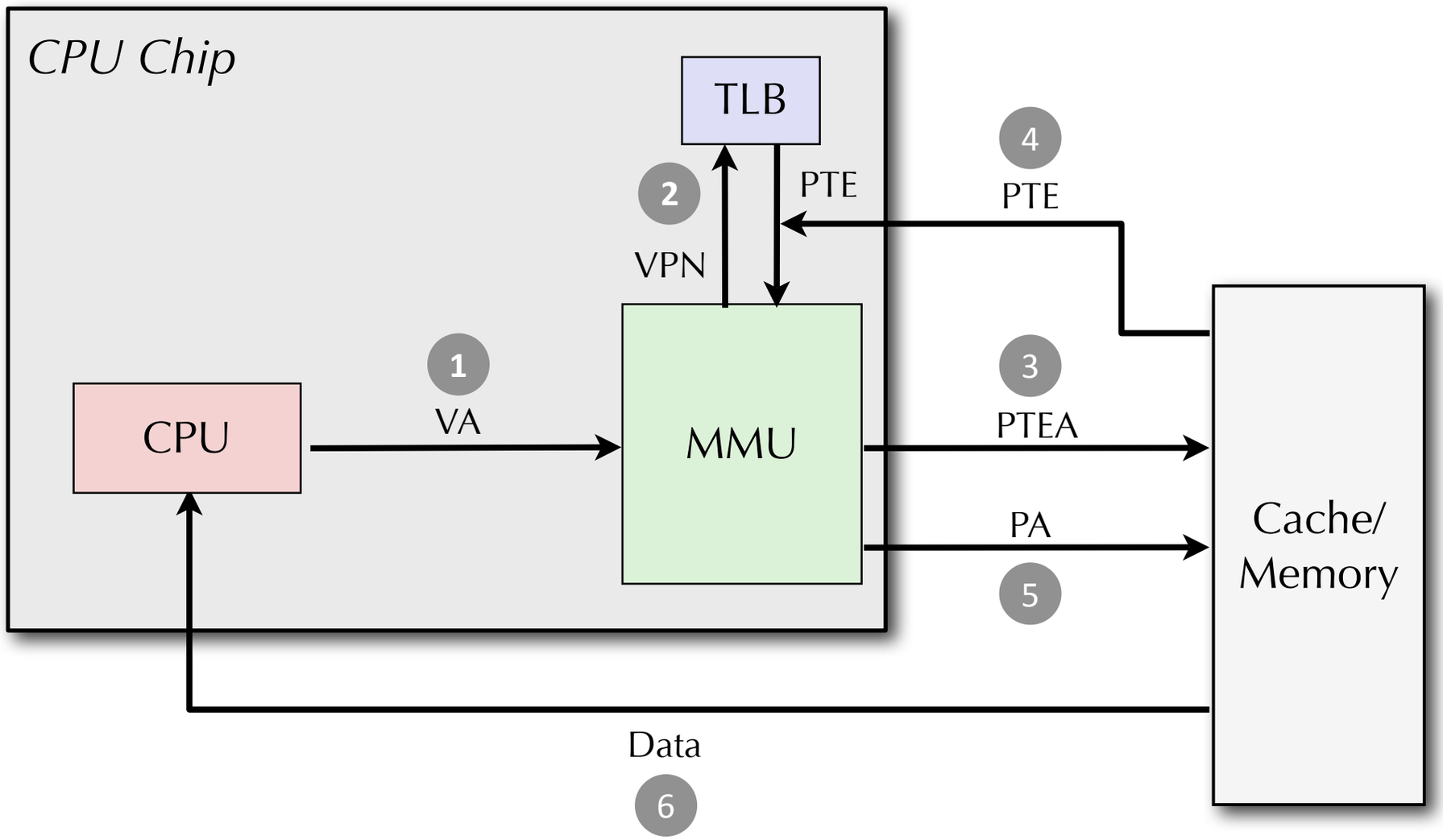
TLB Hit



- A TLB hit eliminates a memory access

© 2010 Stephen Chong, Harvard University

TLB Miss



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

© 2010 Stephen Chong, Harvard University

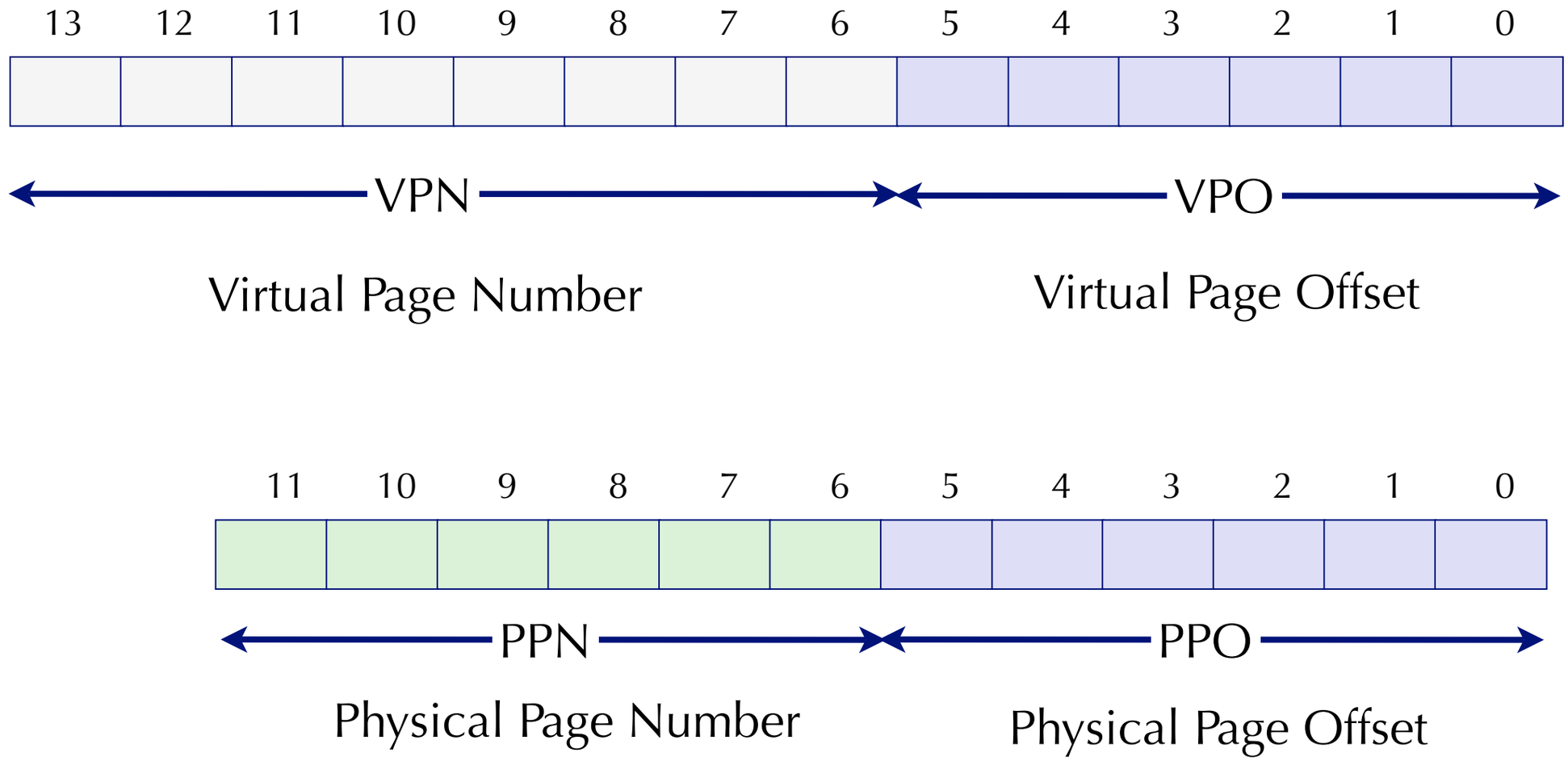
Today

- Running multiple programs at once
- Virtual memory
  - Address spaces
  - Benefits
  - Physical memory as a cache for virtual memory
  - Page tables
  - Page hits and page faults
  - Virtual memory as a tool for memory management
  - Virtual memory as a tool for memory protection
  - Address translation
  - Table Lookaside Buffer (TLB)
  - Example

© 2010 Stephen Chong, Harvard University

Simple Memory System Example

- Addressing
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes



© 2010 Stephen Chong, Harvard University

Simple Memory System Page Table

- Only show first 16 entries (out of 256)

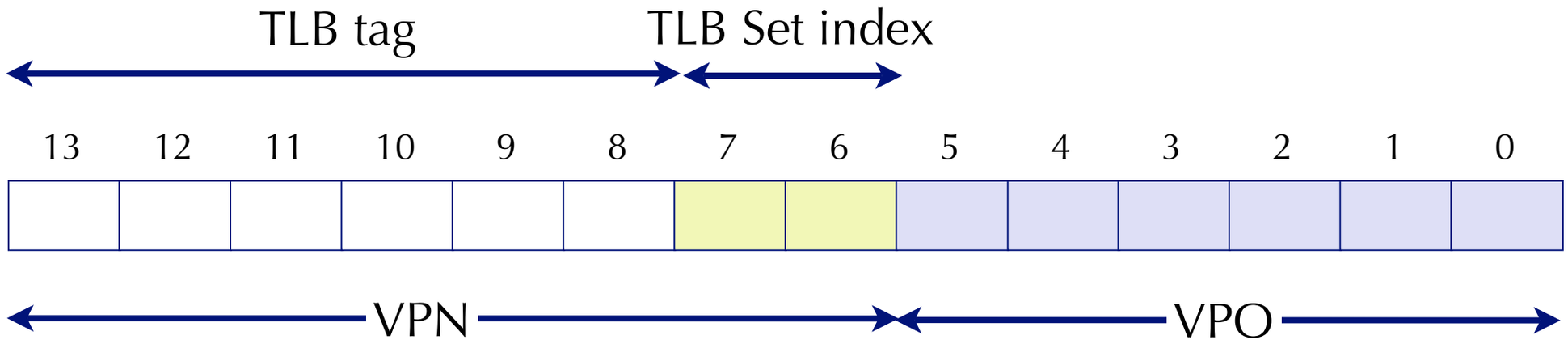
VPN	*PPN*	*Valid*
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

VPN	*PPN*	*Valid*
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

© 2010 Stephen Chong, Harvard University

Simple Memory System TLB

- 16 entries
- 4-way associative

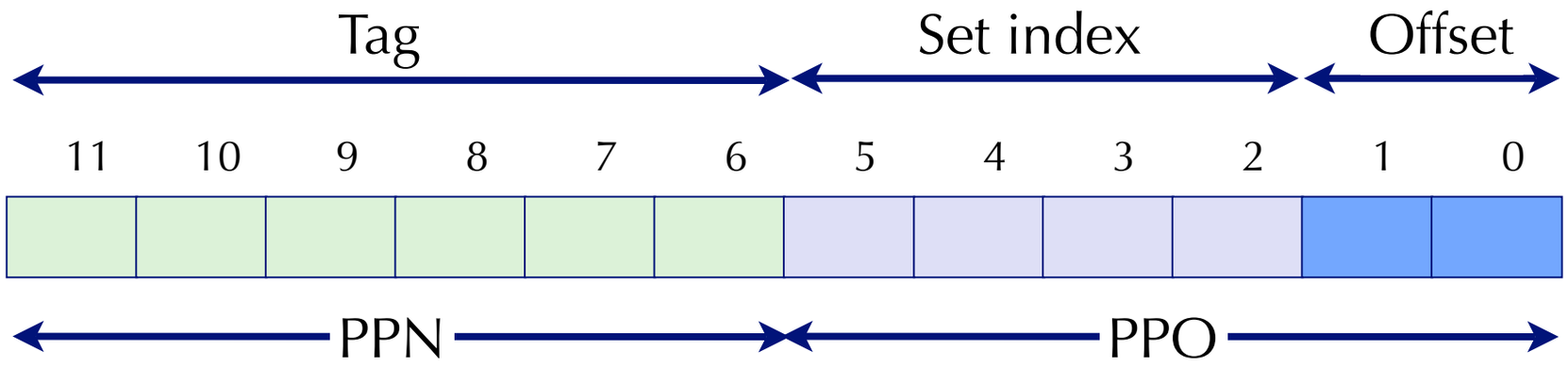


Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

© 2010 Stephen Chong, Harvard University

Simple Memory System Cache

- 16 lines, 4-byte block size
- **Physically addressed**
- Direct mapped



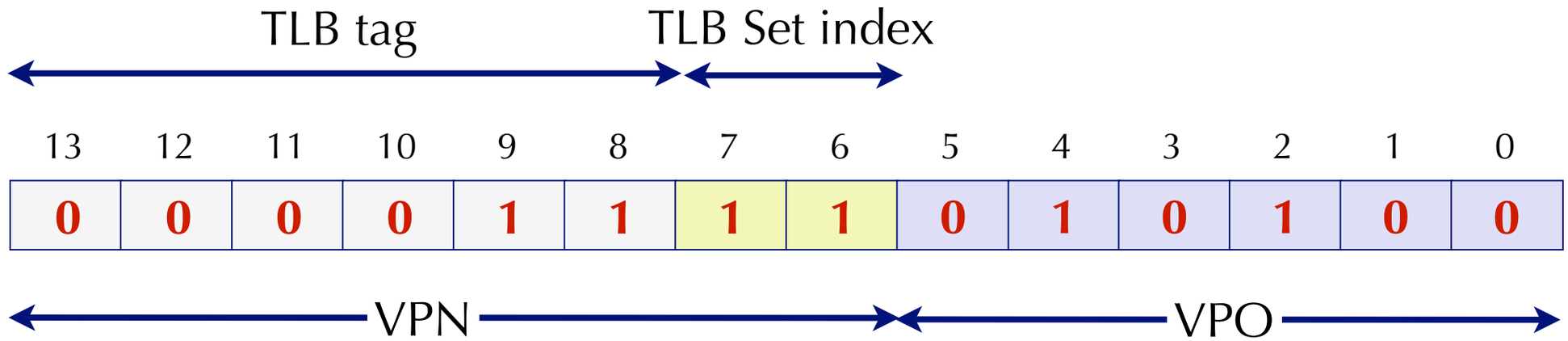
Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

Idx	Tag	Valid	B0	B1	B2	B3
8	24	1	3A	00	51	89
9	2D	0	–	–	–	–
A	2D	1	93	15	DA	3B
B	0B	0	–	–	–	–
C	12	0	–	–	–	–
D	16	1	04	96	34	15
E	13	1	83	77	1B	D3
F	14	0	–	–	–	–

© 2010 Stephen Chong, Harvard University

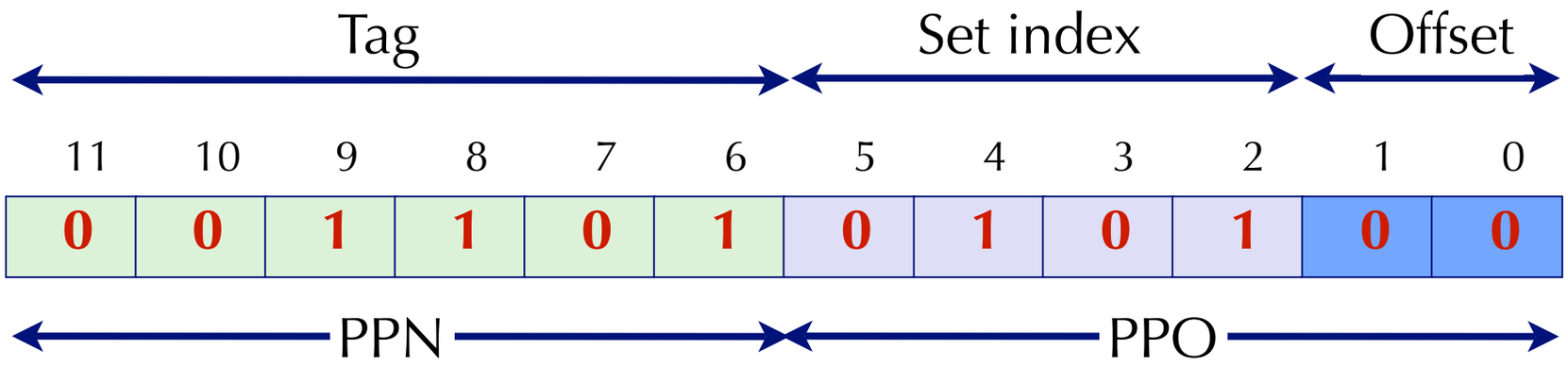
Address Translation Example #1

Virtual Address: `0x03D4`



VPN: 0x0F TLB Set index: 3 TLB Tag: 0x03 TLB Hit? Y Page fault? N PPN: 0x0D

Physical Address

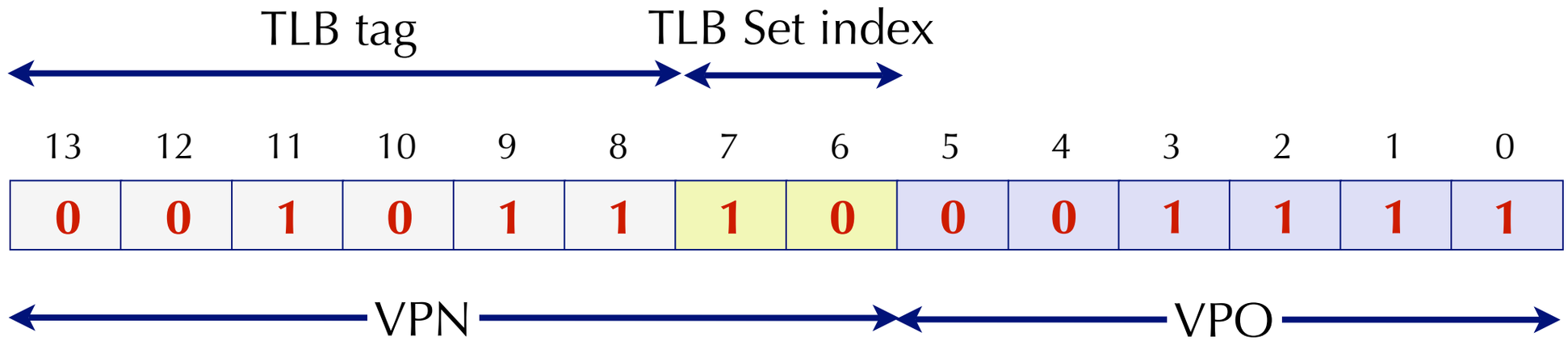


Offset: 0 Set index: 0x5 Tag: 0x0D Hit? Y Byte: 0x36

© 2010 Stephen Chong, Harvard University

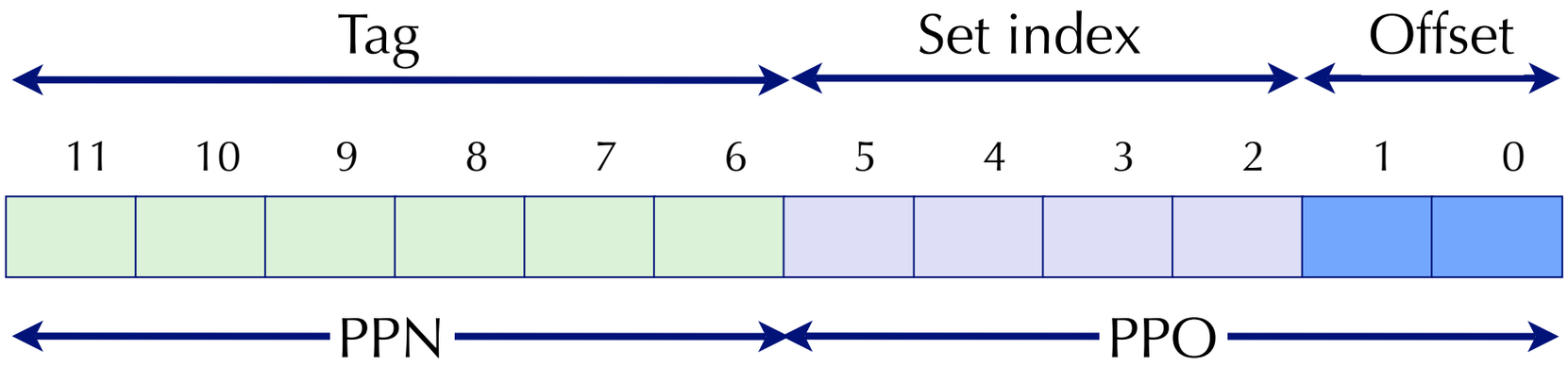
Address Translation Example #2

Virtual Address: `0x0B8F`



VPN: 0x0E TLB Set index: 2 TLB Tag: 0x0B TLB Hit? N Page fault? Y PPN: TBD

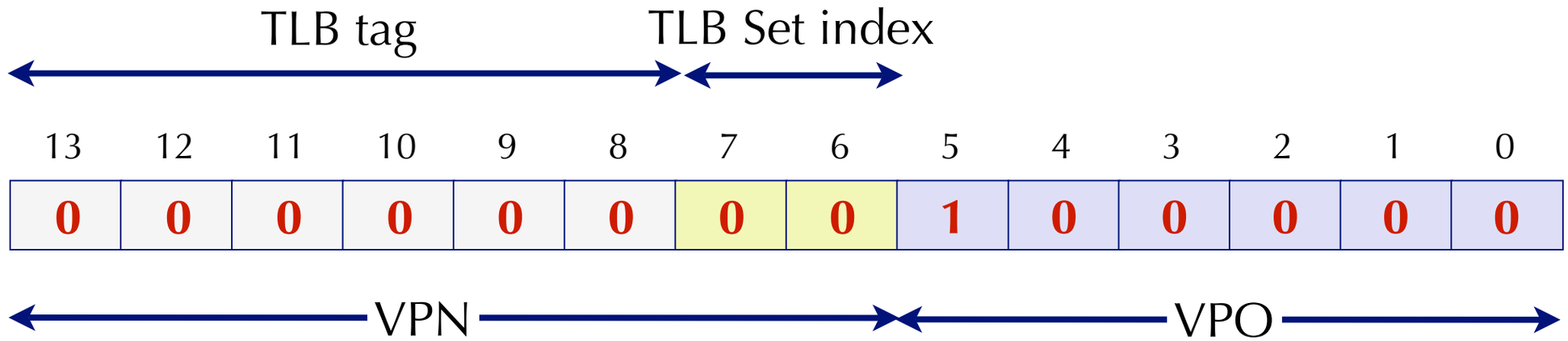
Physical Address



© 2010 Stephen Chong, Harvard University

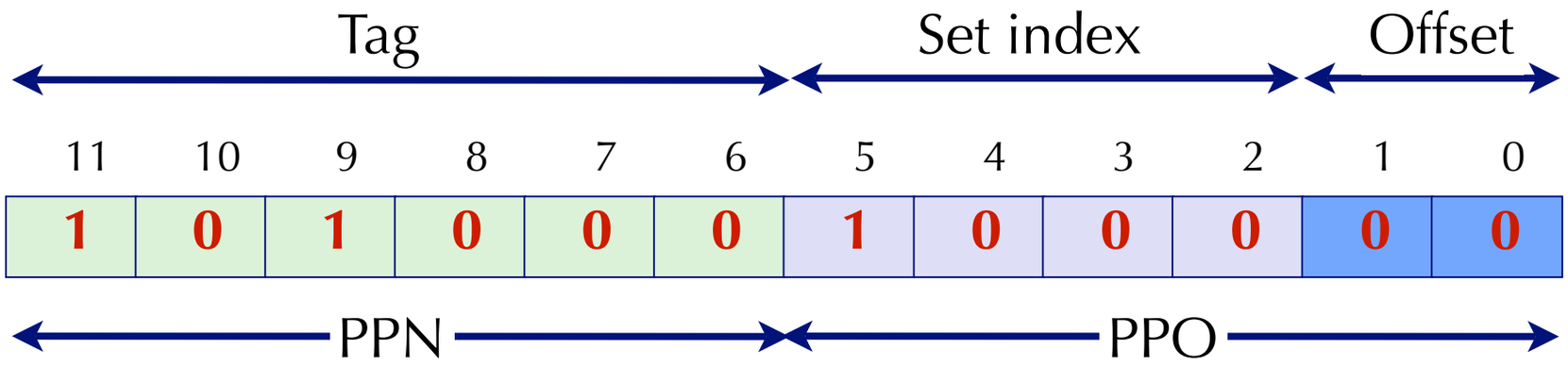
Address Translation Example #3

Virtual Address: `0x0020`



VPN: 0x00 TLB Set index: 0 TLB Tag: 0x00 TLB Hit? N Page fault? N PPN: 0x28

Physical Address



Offset: 0 Set index: 0x8 Tag: 0x28 Hit? N Byte: Mem...

© 2010 Stephen Chong, Harvard University

What happens on a context switch?

- Context switch is fast. Why?
- Only need to change page table base register, and flush the TLB
- Why don't we need to flush L1, L2, L3 cache?
  - CPU caches use *physical addresses*
  - So single physical page with multiple virtual addresses will get reused across processes!

© 2010 Stephen Chong, Harvard University

Summary of VM benefits

- Isolation
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes
- Simplifies memory management and programming
  - Allows multiple concurrent programs to share limited DRAM
- Simplifies protection by providing a convenient method to check permissions
- Efficient
  - Fast translation
    - MMU on CPU, and TLB provides cache of recent translation
  - Fast context switch

© 2010 Stephen Chong, Harvard University

Summary of VM mechanism

- VM is implemented by the MMU and OS working together.
  - MMU does virtual-to-physical address translations.
  - OS sets up the page tables that control those translations.
- The TLB is a cache for recent virtual-to-physical mappings.
  - Avoids lookup in page tables for each memory access.
- The OS handles page faults triggered by the MMU.
  - This is the basis for swapping and demand paging.

© 2010 Stephen Chong, Harvard University

Topics for next time

- Systems programming in UNIX
- File abstraction and I/O operations
- Robust and buffered I/O
- Accessing metadata and directories
- Fun with filehandles

© 2010 Stephen Chong, Harvard University