UNM SCHOOL of ENGINEERING
Department of Computer Science

Lecture #27
Virtual Memory (2)

Prof. Soraya Abad-Mota, PhD

Refresher (0)

Virtual address space:

1. Linear addresses
2. Total size of the space is a power of 2 ($N = 2^n$)
3. Organization
   - Divided into *virtual pages* of size $P = 2^p$
   - A *virtual address* points to a page (virtual page number) and an offset within that page (virtual page offset)
   - The virtual space is page-addressable (as opposed to byte-addressable our previous assumption w/MM)

Exercise

- Review terminology + sizes

  2^n size of virtual memory

n	P (size of page) = 2^p	number of PTE's
32	4K	? entries
32	8K	? entries
64	8K	? entries

Number of PTEs = number of virtual pages = number of addresses?

Topics

- Virtual Memory
  - VM as a tool for caching (cont.)
  - VM as a tool for memory management
  - VM as a tool for memory protection
  - Address translation
  - Simple memory system example

From the index cards

- Some of your questions on the index cards from Monday, will be answered in the following 3 slides

VM as a Tool for Memory Management (sec. 9.4)

- 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 can improve locality

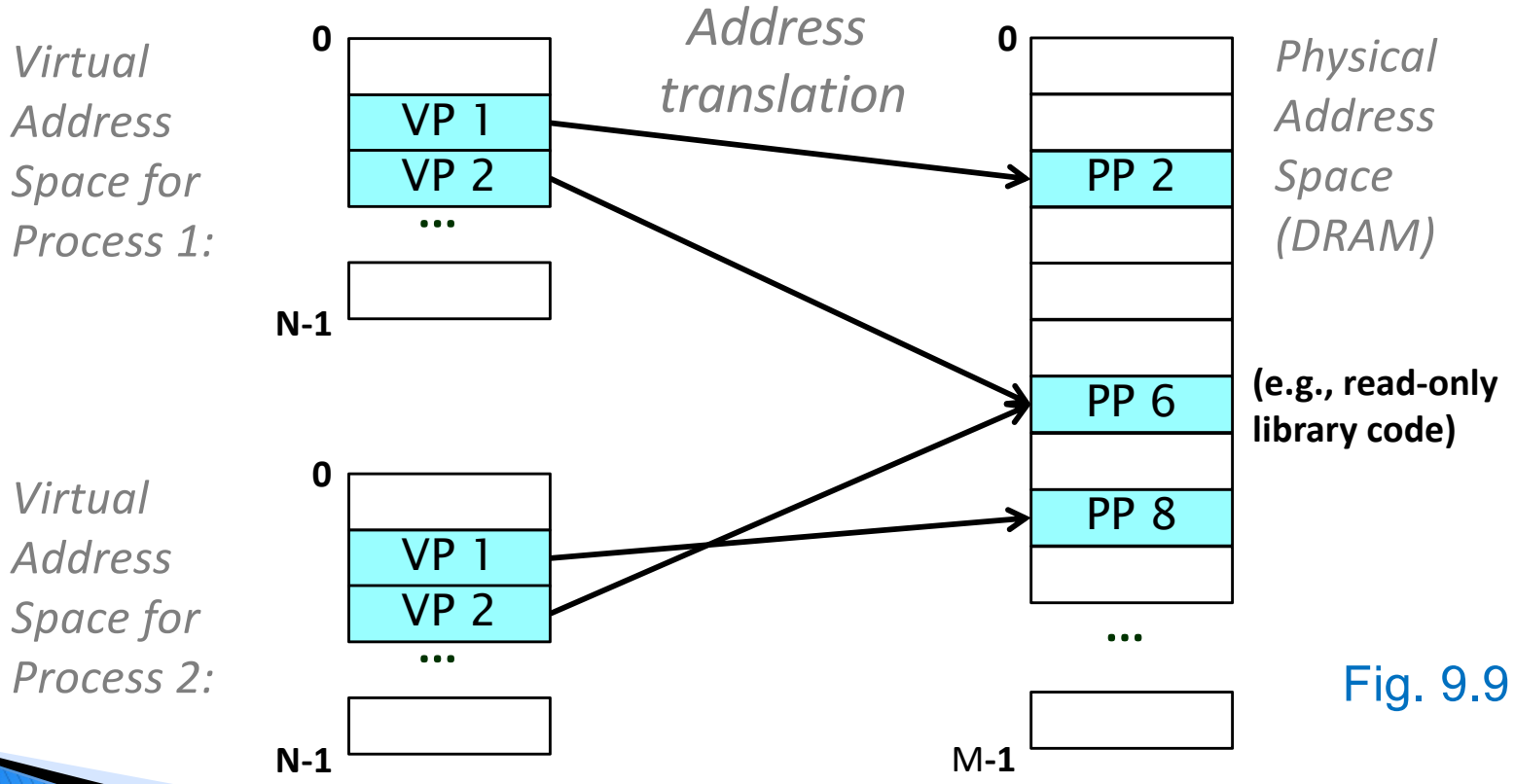
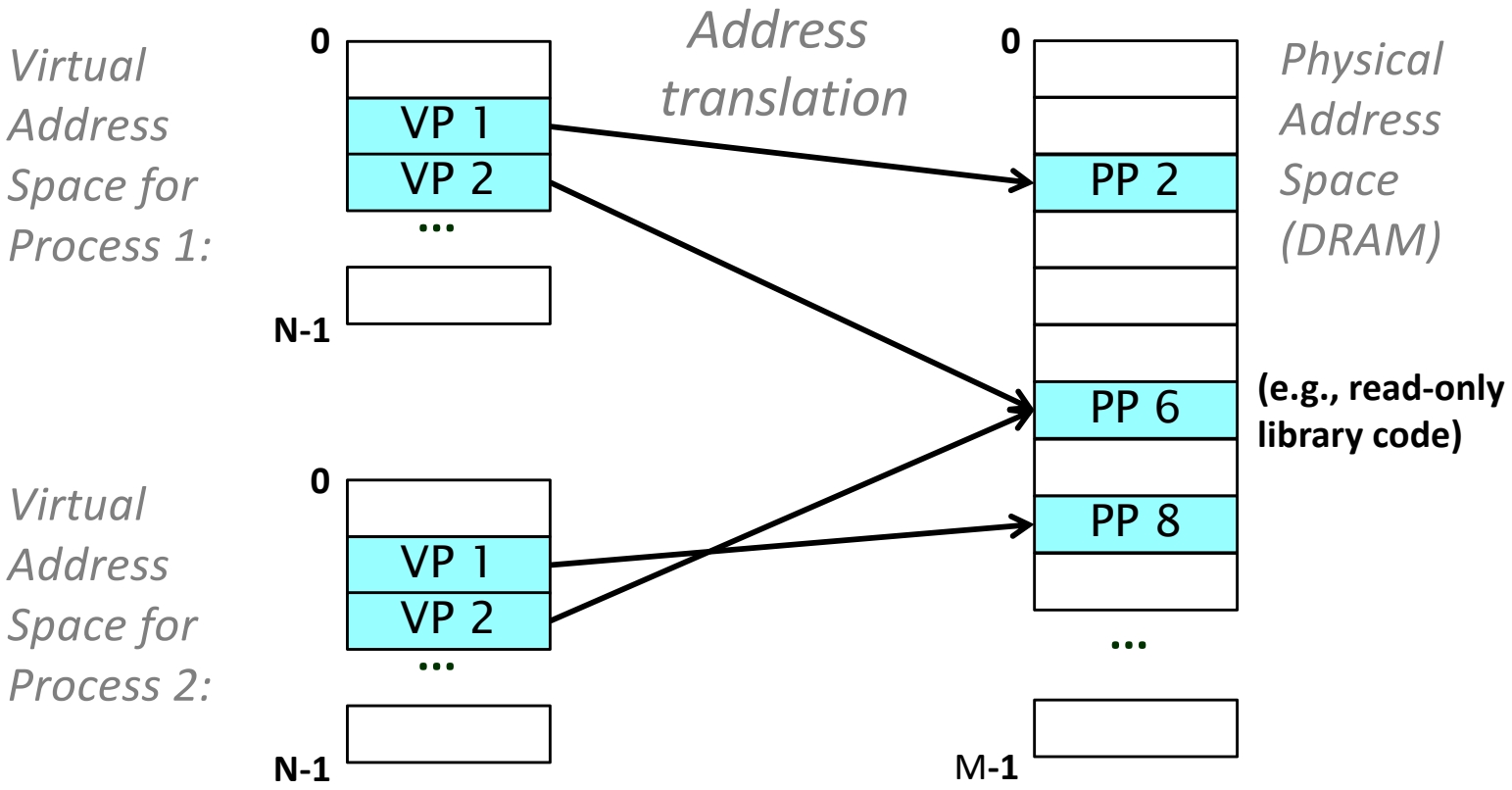


Fig. 9.9

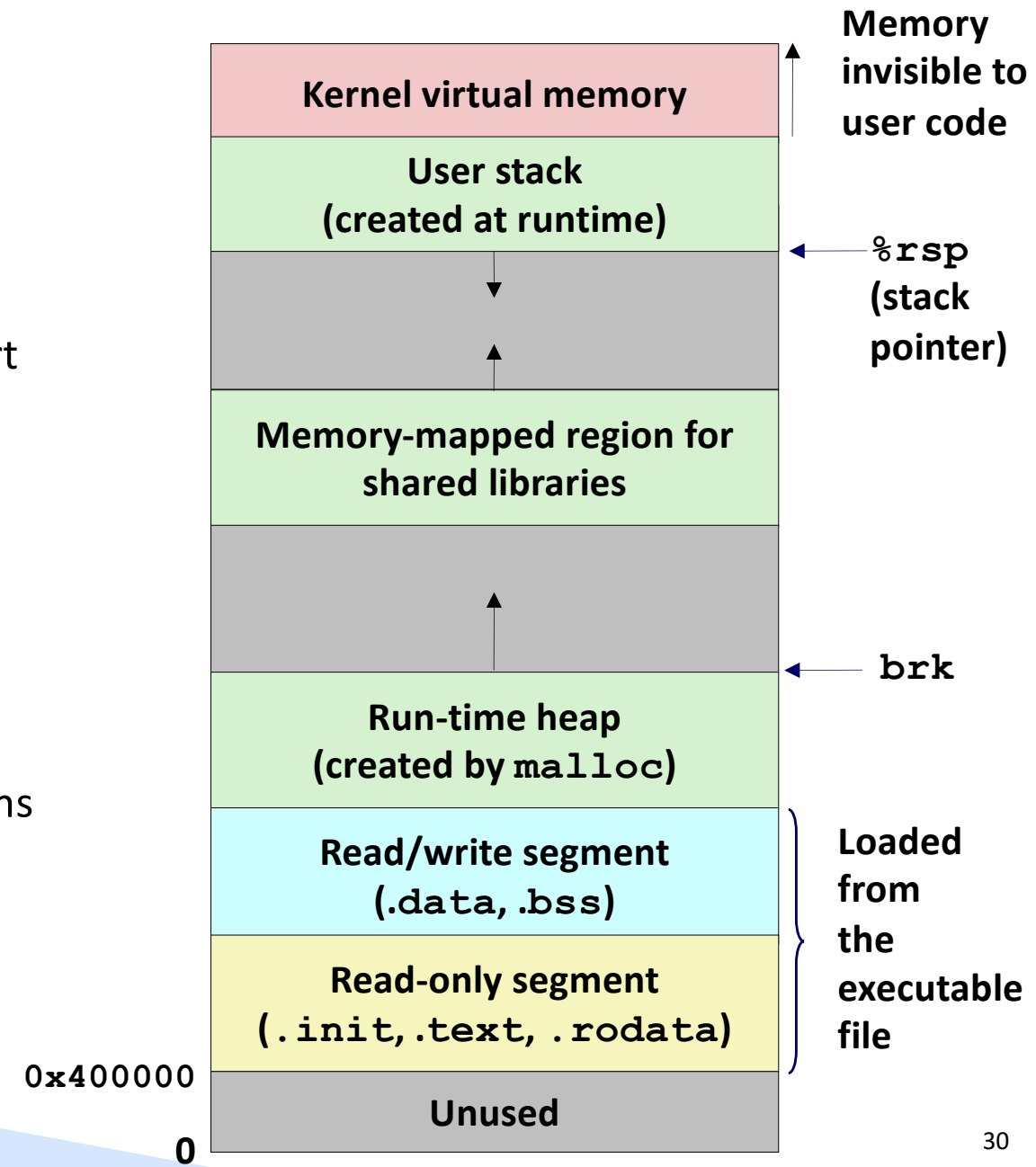
VM as a Tool for Memory Management

- Simplifying 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)



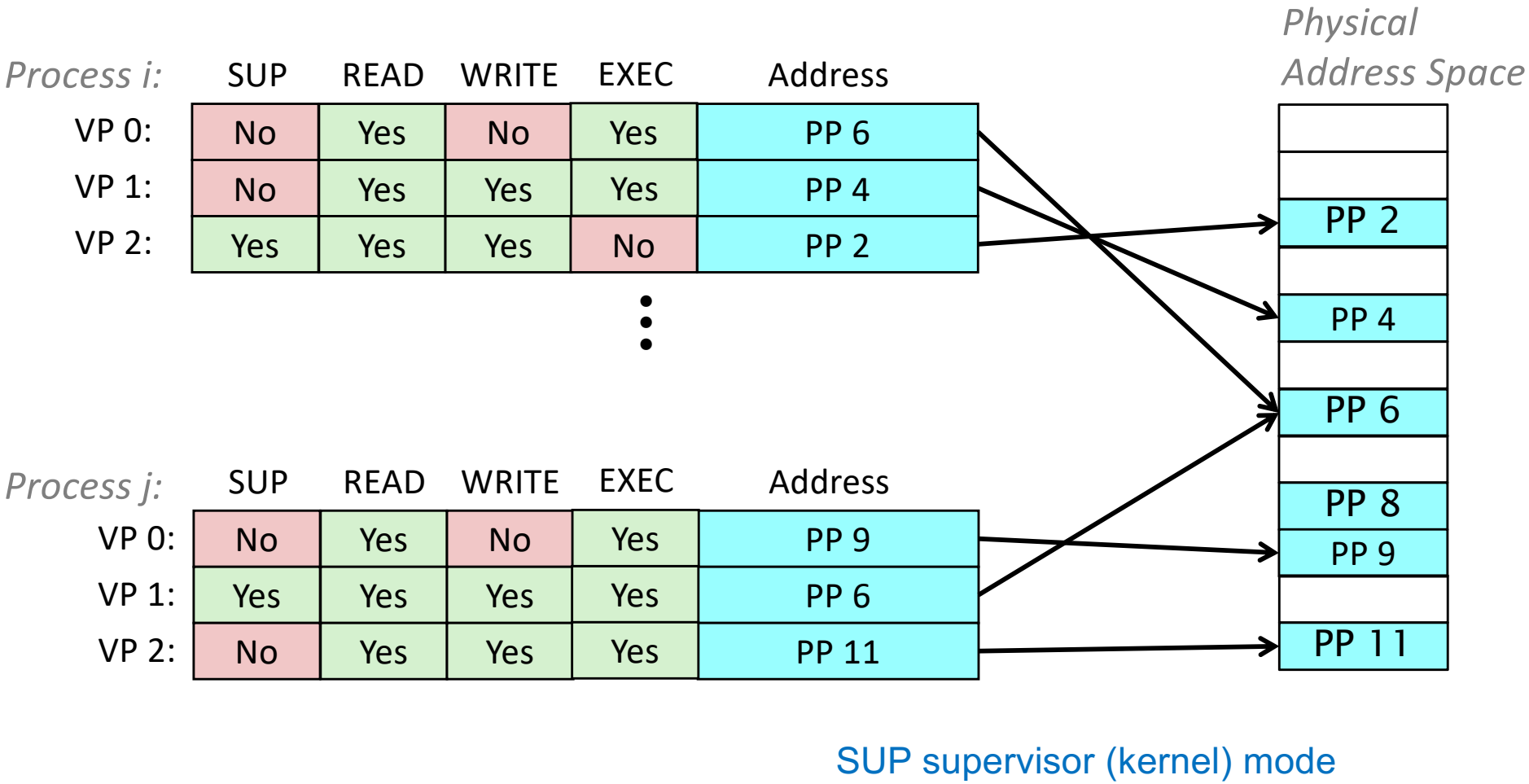
Simplifying Linking and Loading

- Linking
  - Each program has similar virtual address space
  - Code, data, and heap always start at the same addresses.
- 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 (memory mapping)



VM: Tool for Memory Protection (sec. 9.5)

- Extend PTEs with permission bits
- MMU checks these bits on each access
- attempts to violate permissions ➔ protection fault ("segmentation fault")



SUP supervisor (kernel) mode

Topics

- Virtual Memory
  - VM as a tool for memory management
  - VM as a tool for memory protection
  - Address translation
  - Simple memory system example
  - Case study: Core i7/Linux memory system (skip)

VM Address Translation

- Virtual Address Space
  - *V = {0, 1, …, N–1}* (virtual page 0 to virtual page N-1)
- Physical Address Space
  - *P = {0, 1, …, M–1}* (physical page 0 to physical page M-1)
- Address Translation
  - ***MAP: V → P U {∅}***
  - For virtual address ***a***:
    - ***MAP(a) = a'***

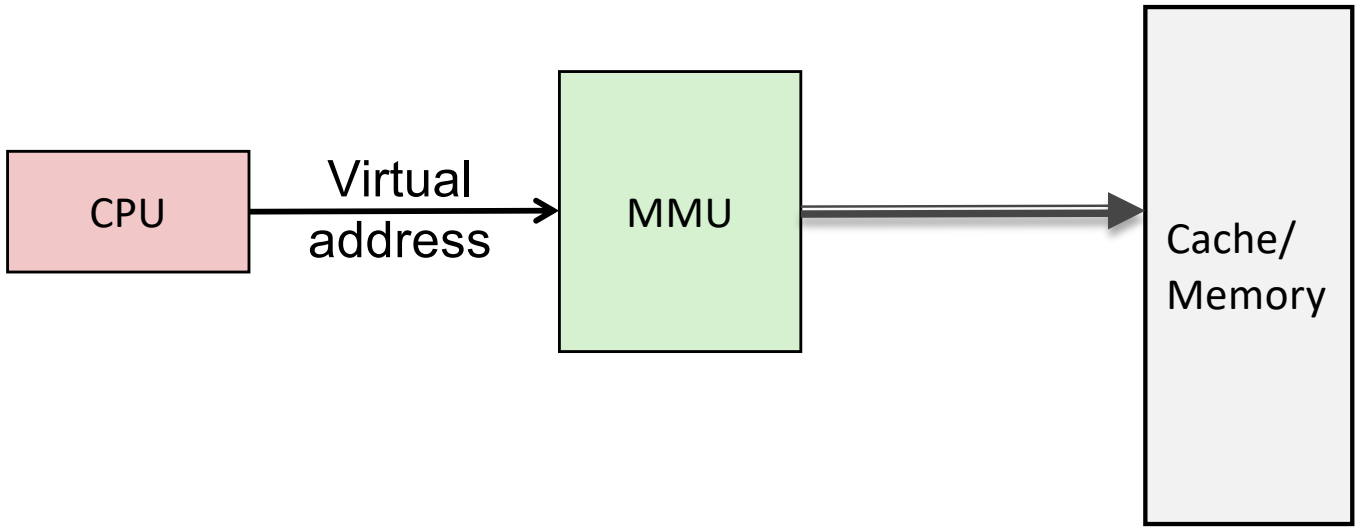
      if data at virtual address ***a*** is at physical address ***a'*** in ***P***
    - ***MAP(a) = ∅***

      if data at virtual address ***a*** is not in physical memory
      - Either invalid or stored on disk (==> PTE points to null or to disk address)

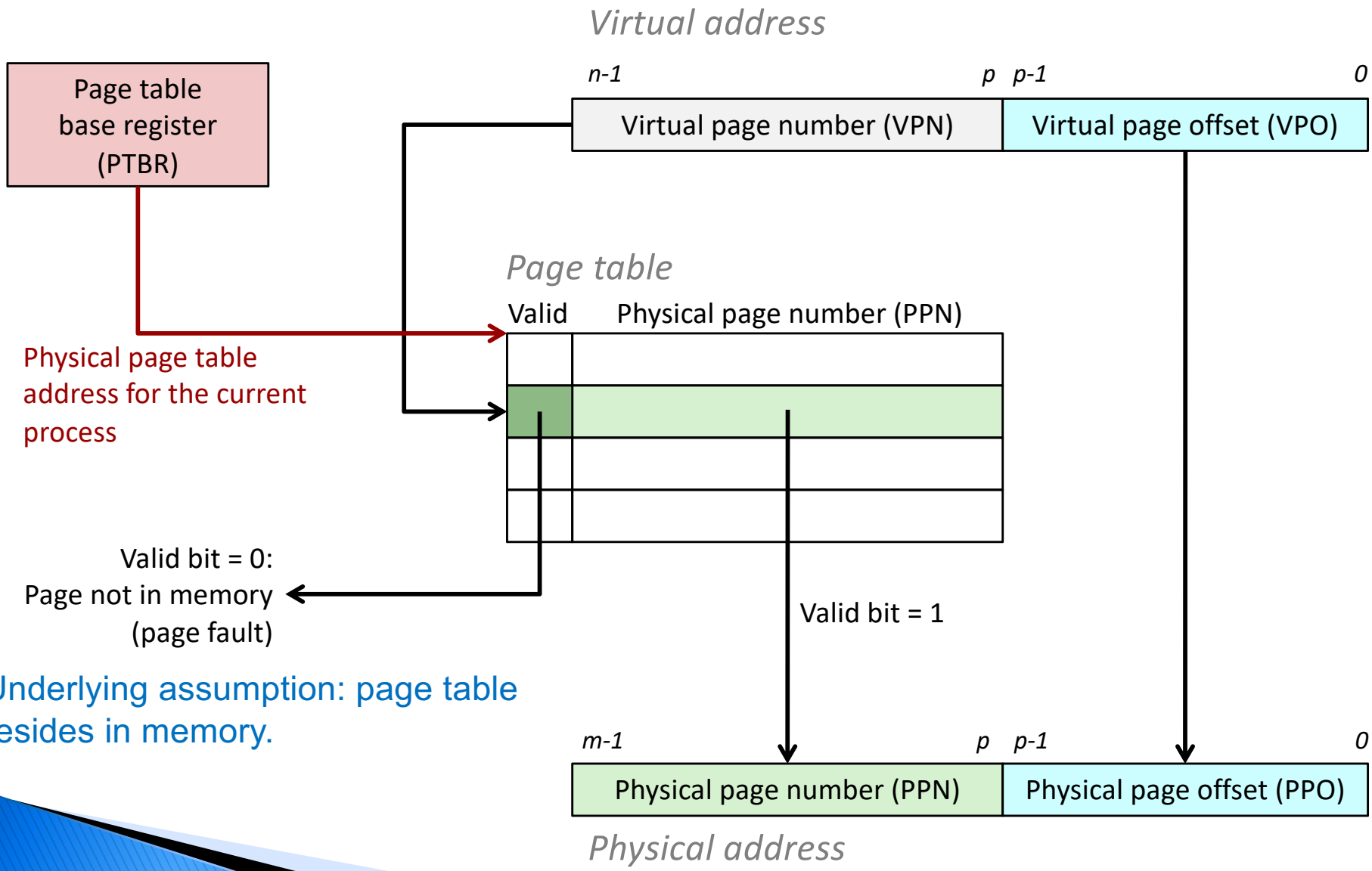
Address Translation Symbols
(fig. 9.11, p. 814)

- 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)
- Components of the virtual address (VA)
  - **VPO**: Virtual page offset (to locate the word inside the page)
  - **VPN**: Virtual page number
- Components of the physical address (PA)
  - **PPO**: Physical page offset (same as VPO)
  - **PPN:** Physical page number

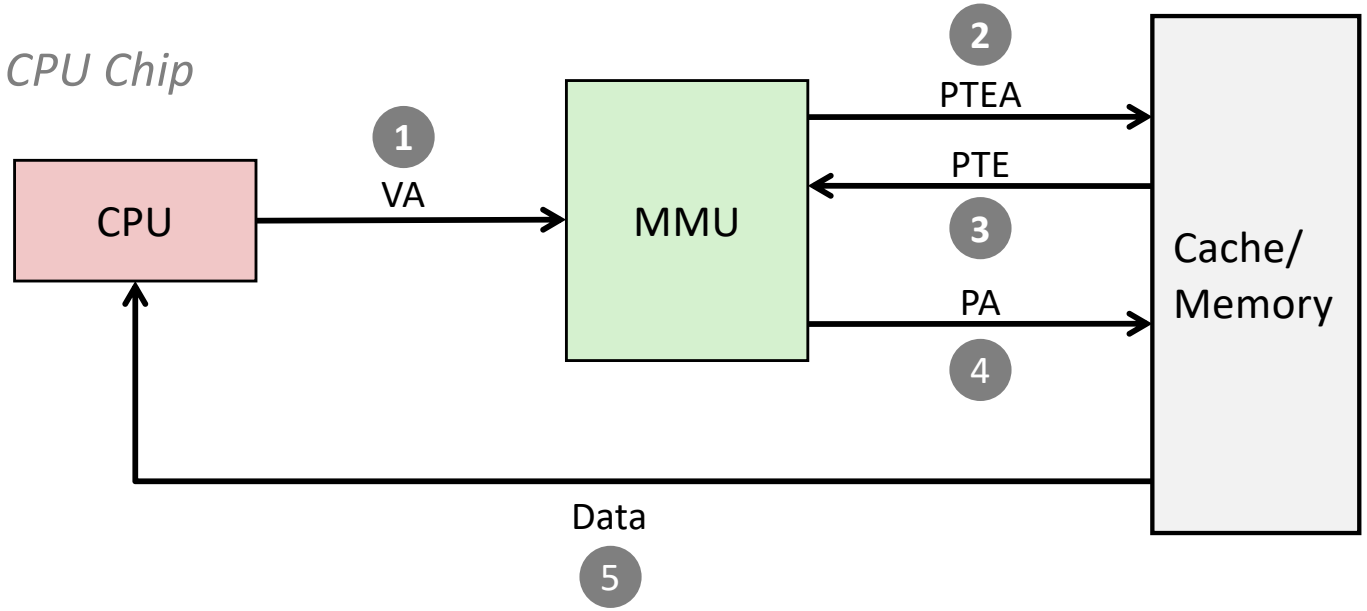
The Memory Management Unit



Address Translation With a Page Table



Address Translation: Page Hit



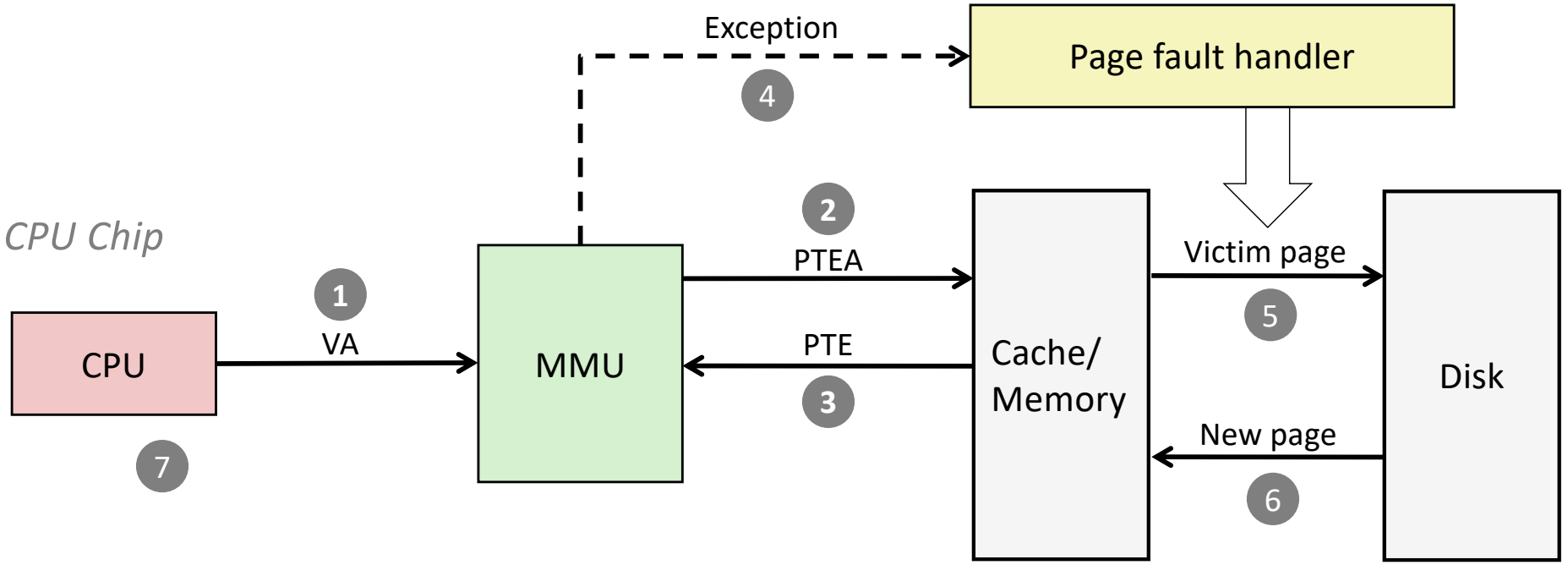
1) Processor sends virtual address to MMU

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

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, places page on disk: write-back)

6) Handler brings in *new page* and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction

(will now get a page hit)

Refresher (1)

1. What is a virtual address (VA)?
2. Which are the components of a VA?
3. What is a Physical Address (PA) and its components?
4. What is a page table and what is it used for?
5. What does it mean for a virtual page to be cached?

Refresher (2)

6. What is the MMU?
7. What is the main task performed by the MMU?
8. What is the relationship between a virtual address (VA) and a physical address (PA)?

Address translation

Goal: to map a virtual address (VA) to a physical memory address

- The memory serves as a staging area for the virtual pages that live permanently on disk.
- Main memory stores virtual pages

Physical address (PA): points to a memory address were the page starts, it contains P bytes, and the physical page offset, is where the data that we are looking for starts within that page.

Exercise

- Practice problem 9.3 p.816: 32-bit VA, 24-bit PA

P	VPN bits	VPO bits	PPN bits	PPO bits
1K				
2K				
4K				
8K				

first, calculate p ($\log_2 P$)

Exercise

- Practice problem 9.3 p.816: 32-bit VA, 24-bit PA

P	VPN bits	VPO bits	PPN bits	PPO bits
1K	32-10	10	24-10	10
2K				
4K				
8K				

first, calculate p ($\log_2 P$) = VPO bits

Procedure to answer

- To address P bytes in one page, we need $\log_2 P$ which gives the number of bits necessary, this is number of bits for the offset, which of the two offsets?
  (PPO or VPO? why?)

- VAS: 32 – offset bits = bits for VPN
- PAS: 24 – offset bits = bits for PPN

The S is for space

Topics

- Virtual Memory
  - Integrating VM and cache
  - Speeding up address translation with TLB
  - A full translation exercise
  - Simple memory system example
  - Multi-level page tables
  - Case study: Core i7/Linux memory system (skip)

Speeding up Translation with a TLB (sec. 9.6.2)

- 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 L1 delay
- Solution (to speed up): *Translation Lookaside Buffer* (TLB)
  - Small set-associative hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages
  - (After understanding the TLB, we can go back to table 6.23 on page 614, and understand the TLB info. in that table.)

Integrating VM and Cache (sec. 9.6.1)

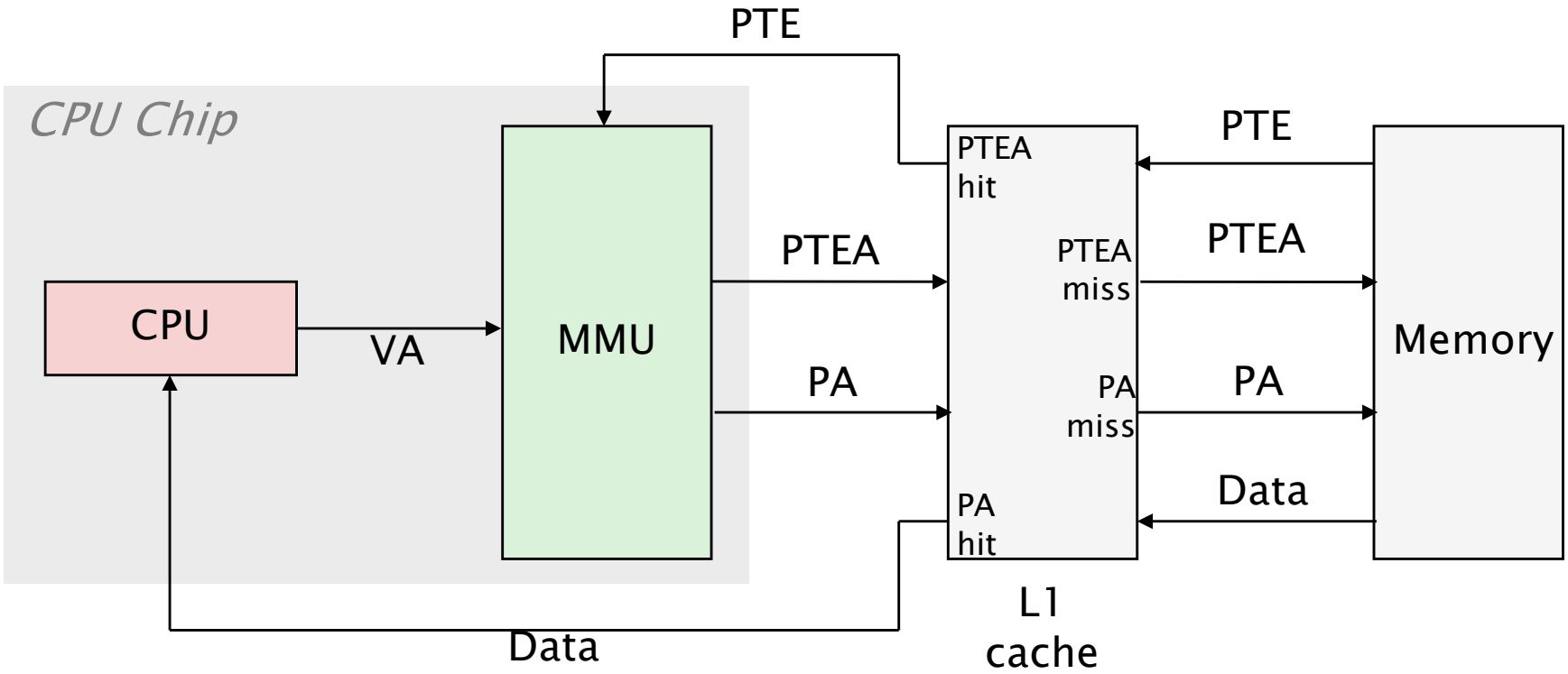
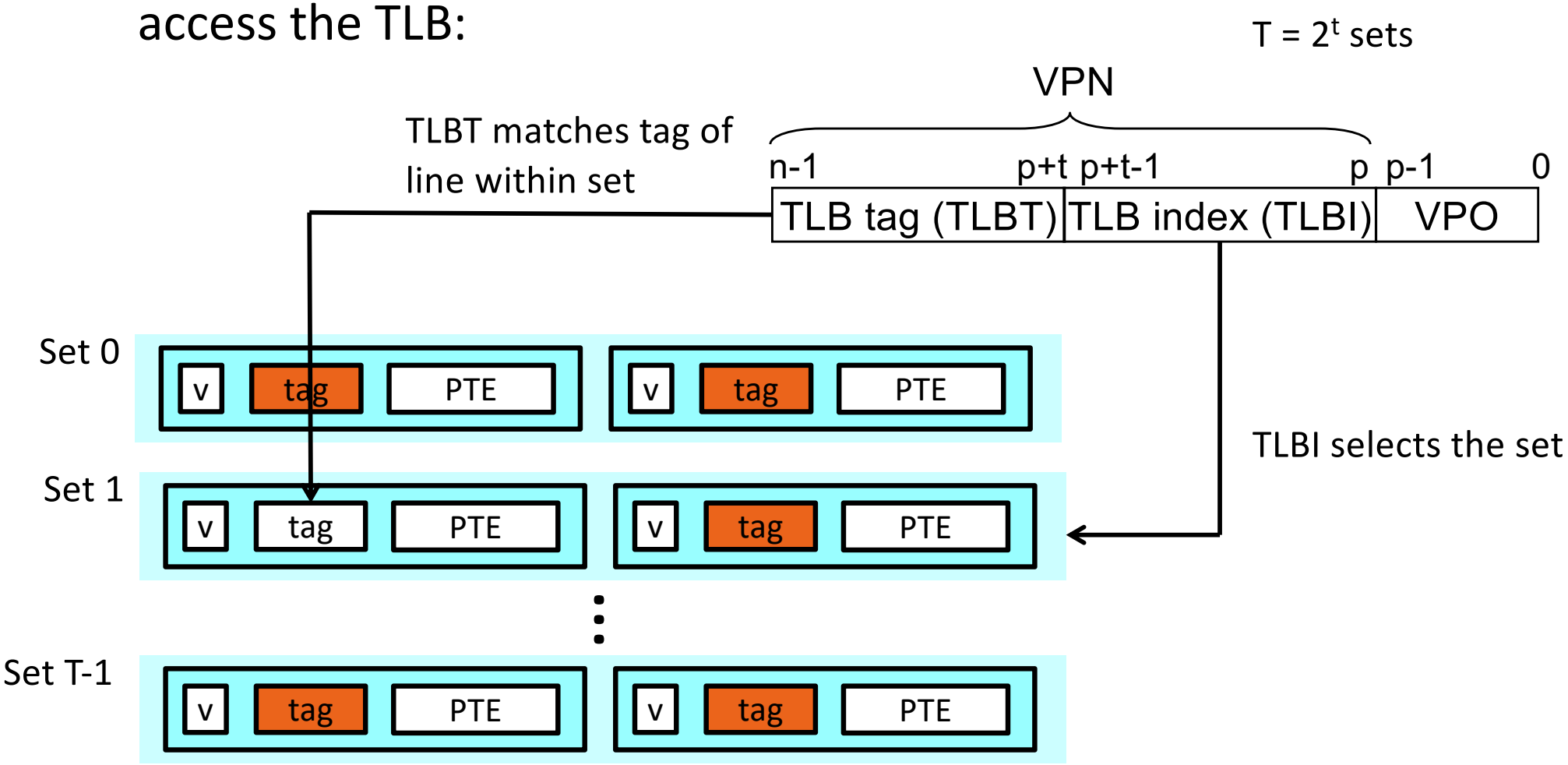


fig. 9.14

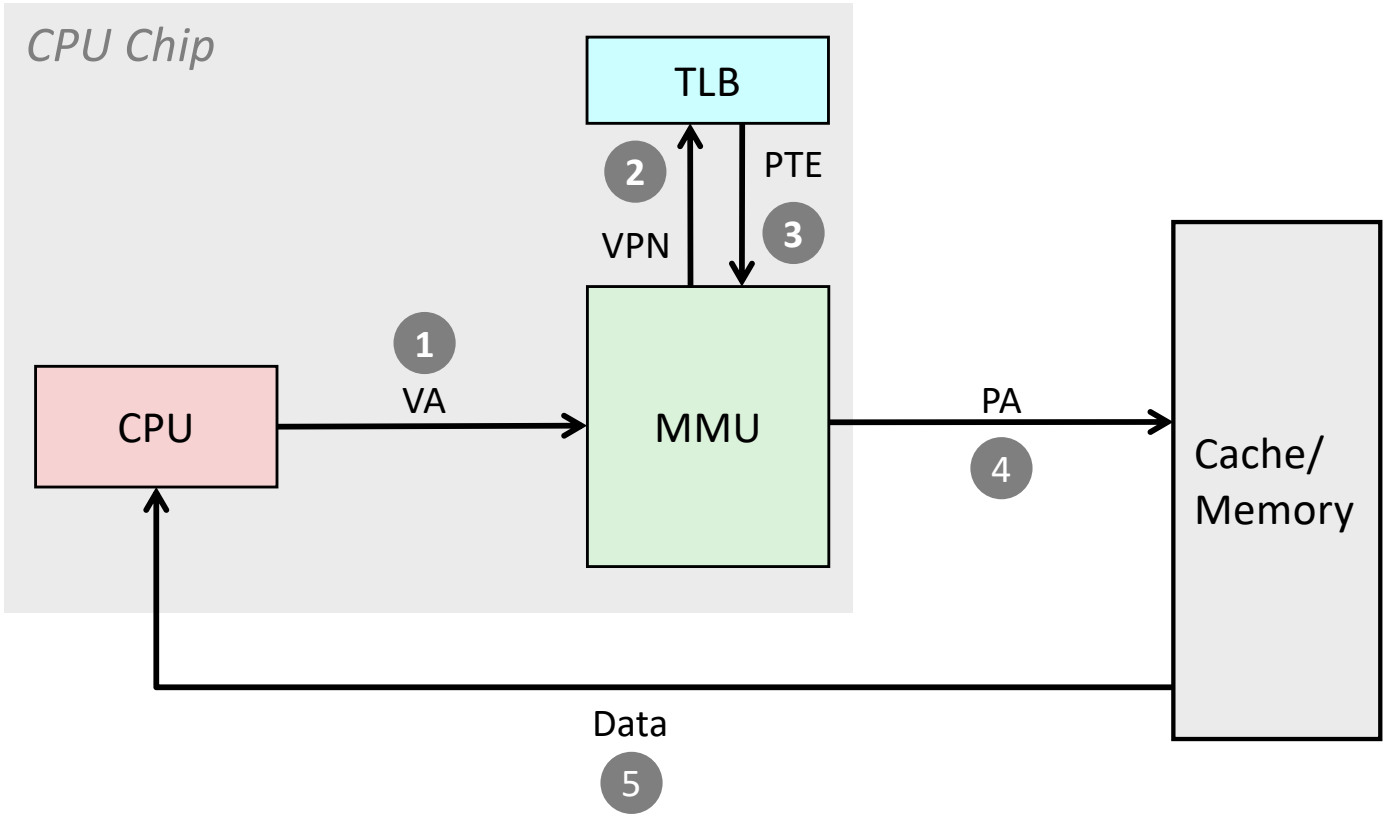
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Accessing the TLB

- MMU uses the VPN portion (only) of the virtual address to access the TLB:

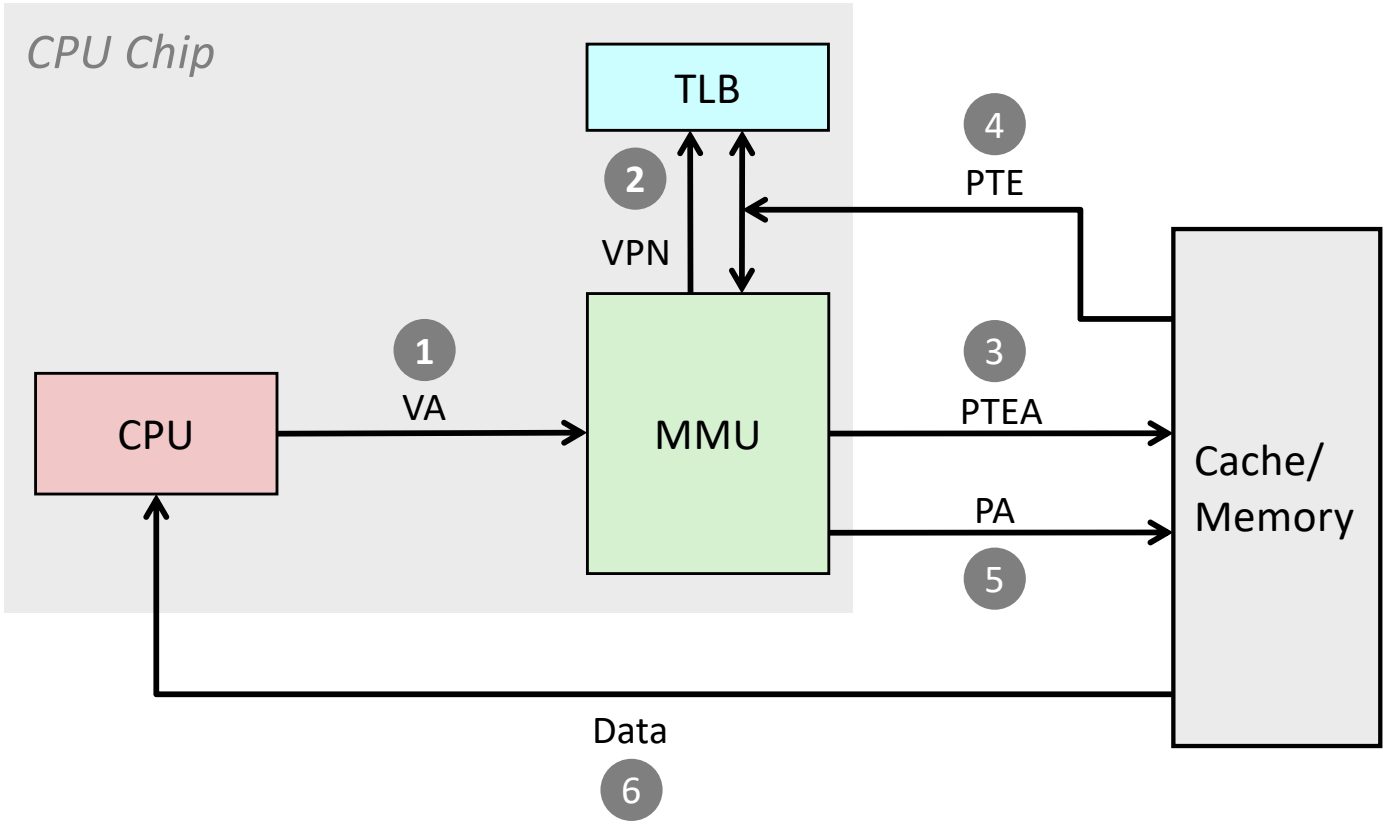


TLB Hit



A TLB hit eliminates a memory access

TLB Miss



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

Topics

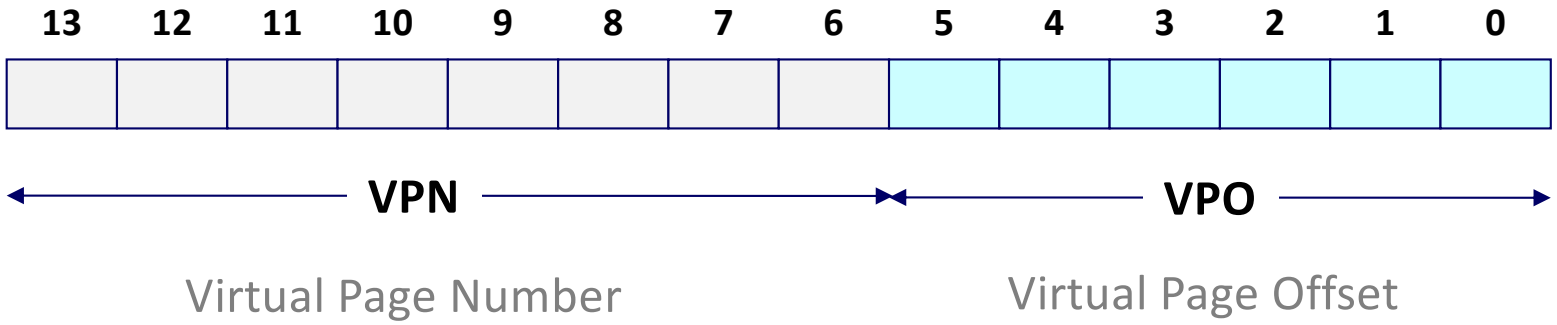
- Virtual Memory
  - Integrating VM and cache
  - Speeding up address translation with TLB
  - Simple memory system example
  - Two full translation exercises
  - Multi-level page tables
  - Case study:
    - Core i7/Linux memory system (on your own)

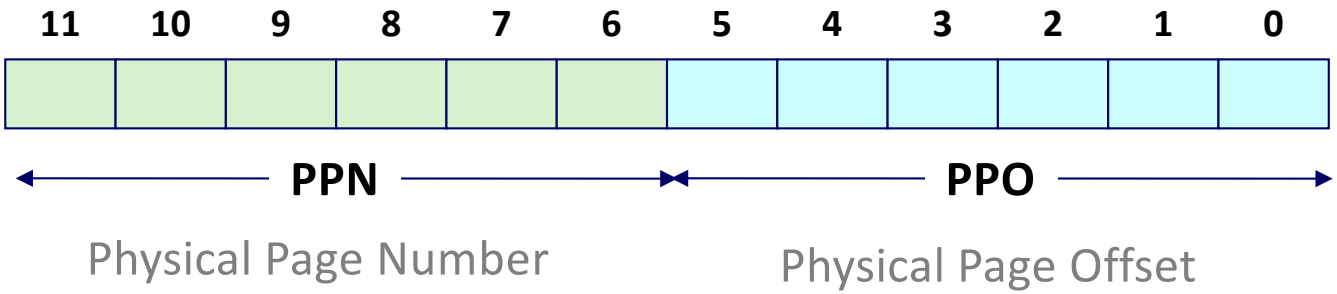
Review of Symbols

- 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)
- Components of the virtual address (VA)
  - **TLBI**: TLB index (TLB =Translation Lookaside Buffer)
  - **TLBT**: TLB tag
  - **VPO**: Virtual page offset
  - **VPN**: Virtual page number (decomposed into TLBT and TLBI)
- Components of the physical address (PA)
  - **PPO**: Physical page offset (same as VPO)
  - **PPN:** Physical page number (decomposed into CT and CI, as usual)
  - **CO**: Byte offset within cache line
  - **CI:** Cache index
  - **CT**: Cache tag

Simple Memory System Example

- Addressing (memory is byte-addressable)
  - 14-bit virtual addresses
  - 12-bit physical address
  - Page size = 64 bytes (2^6)



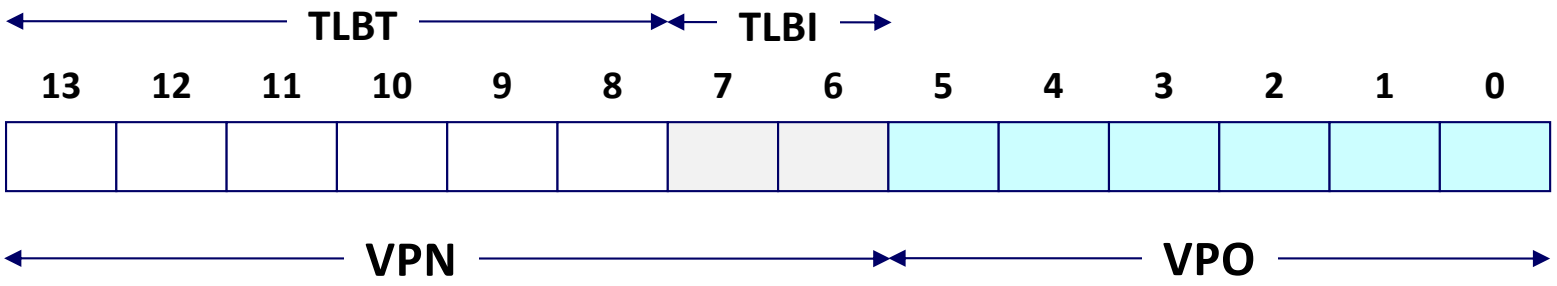


Memory/cache content for examples

- The next three slides contain TLB, Page Table, and cache, content for the two complete address translation examples that we will do today
- Review questions
  - what is stored in TLB? (explain 42-43, e.g. shown in slide 50)
  - what is the content of the Page Table and where is it stored? (explain slide 10 from lecture #24, shown slide 51)
  - what is stored in the cache on the third slide (slide 52)?

1. 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

2. Simple Memory System Page Table

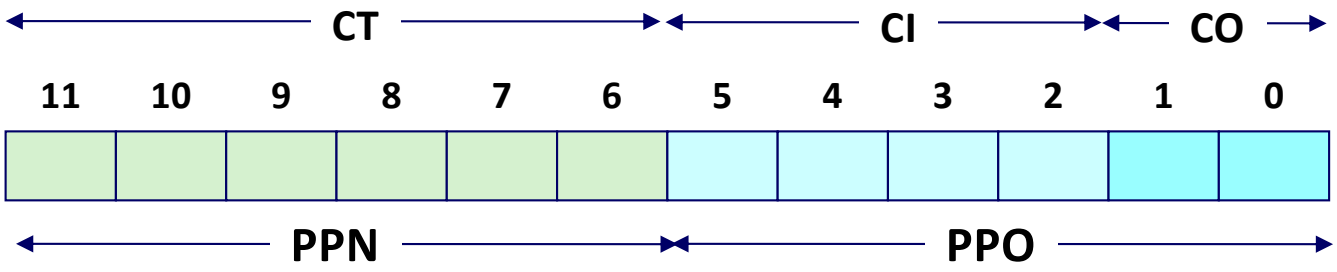
Only show first 16 entries (out of 256)

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

3. Simple Memory System Cache

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

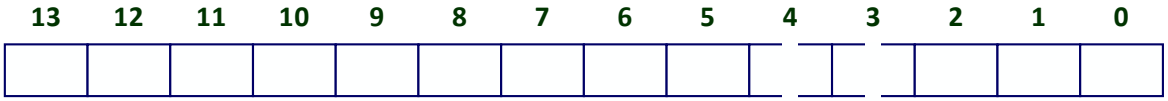


Idx	*Tag*	*Valid*	*B0*	*B1*	*B2*	*B3*
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

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	–	–	–	–

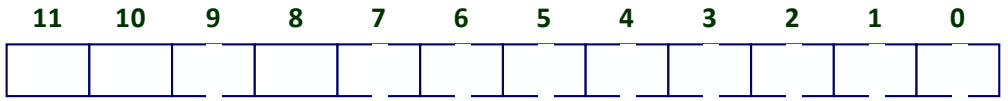
Address Translation Example #1 (in the book)

Virtual Address: `0x03D4`



VPN ___ TLBI ___ TLBT ___ TLB Hit? __ Page Fault? __ PPN: ____

Physical Address



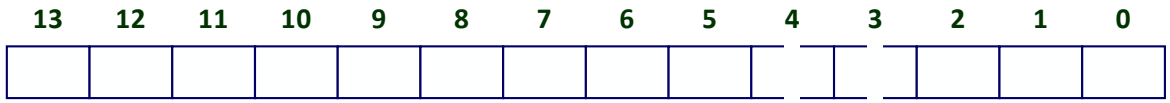
CO ___ CI__ CT ___ Hit? __ Byte: ____

Topics

- Virtual Memory
- Address translation (examples 2 and 3)
- Multi-level page tables
- Case study: Core i7/Linux memory system (example to study on your own)
- Lect #26:
  - System-Level Input/output (Chap. 10)

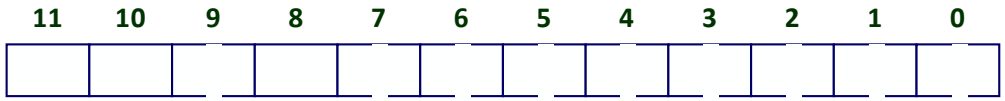
Address Translation Example #2

Virtual Address: 0x0020



VPN ___ TLBI ___ TLBT ___ TLB Hit? __ Page Fault? __ PPN: ____

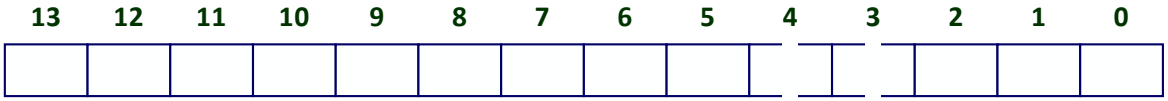
Physical Address



CO ___ CI__ CT ___ Hit? __ Byte: ____

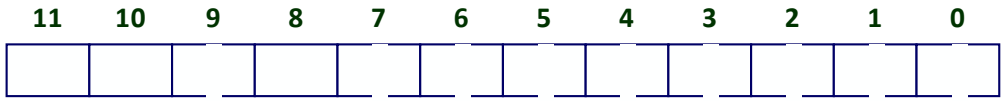
Address Translation Example #3 (problem 9.4)

Virtual Address: `0x03D7`



VPN ___ TLBI ___ TLBT ___ TLB Hit? __ Page Fault? __ PPN: ____

Physical Address



CO ___ CI__ CT ___ Hit? __ Byte: ____