

VIRTUAL MEMORY: CONCEPTS

CS 045

Computer Organization and
Architecture

Prof. Donald J. Patterson

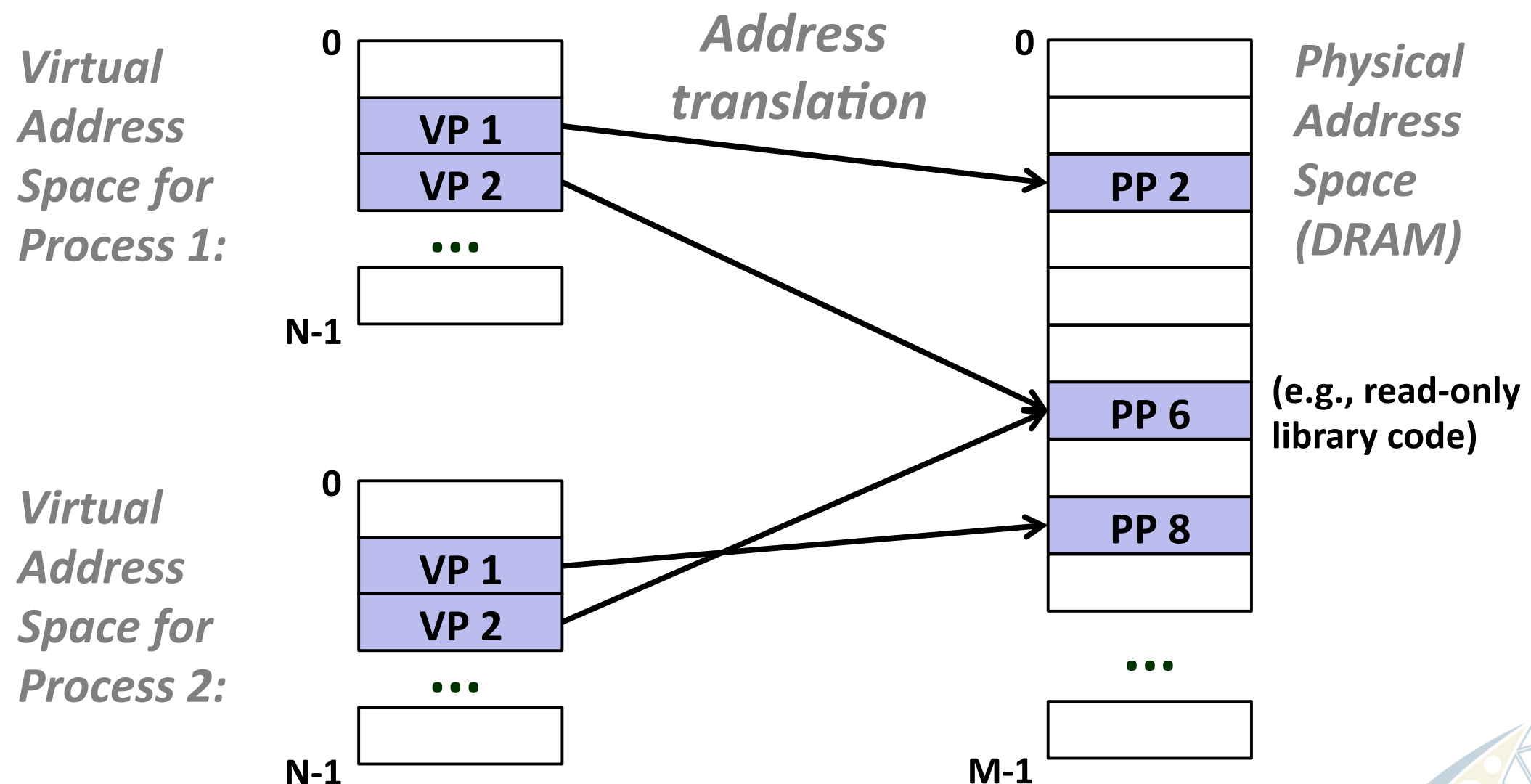
Adapted from Bryant and O'Hallaron,
Computer Systems:
A Programmer's Perspective, Third Edition

VIRTUAL MEMORY: CONCEPTS

- ADDRESS SPACES
- VM AS A TOOL FOR CACHING
- VM AS A TOOL FOR MEMORY MANAGEMENT
- VM AS A TOOL FOR MEMORY PROTECTION
- ADDRESS TRANSLATION

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 can improve locality



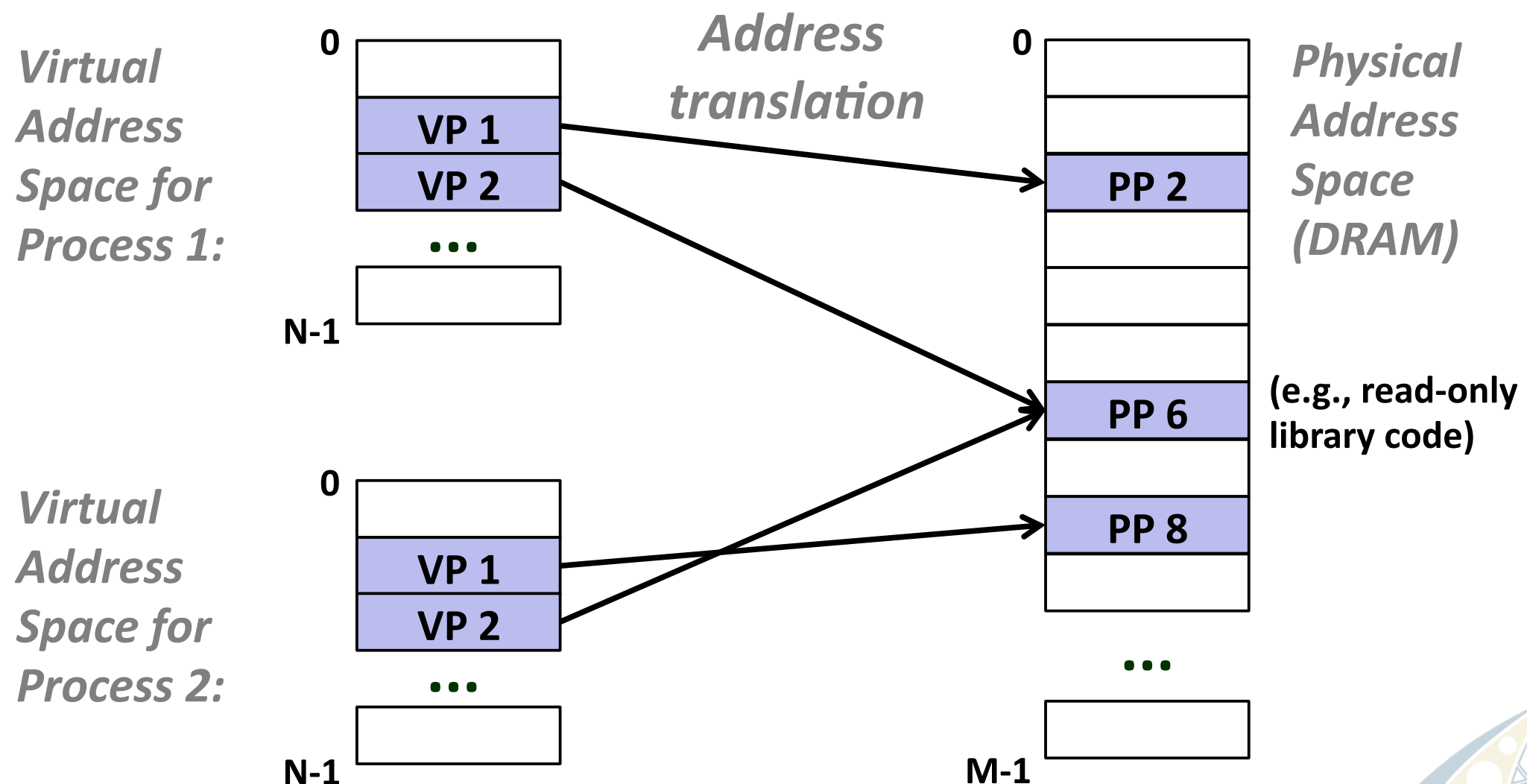
VM AS A TOOL FOR MEMORY MANAGEMENT

■ 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)



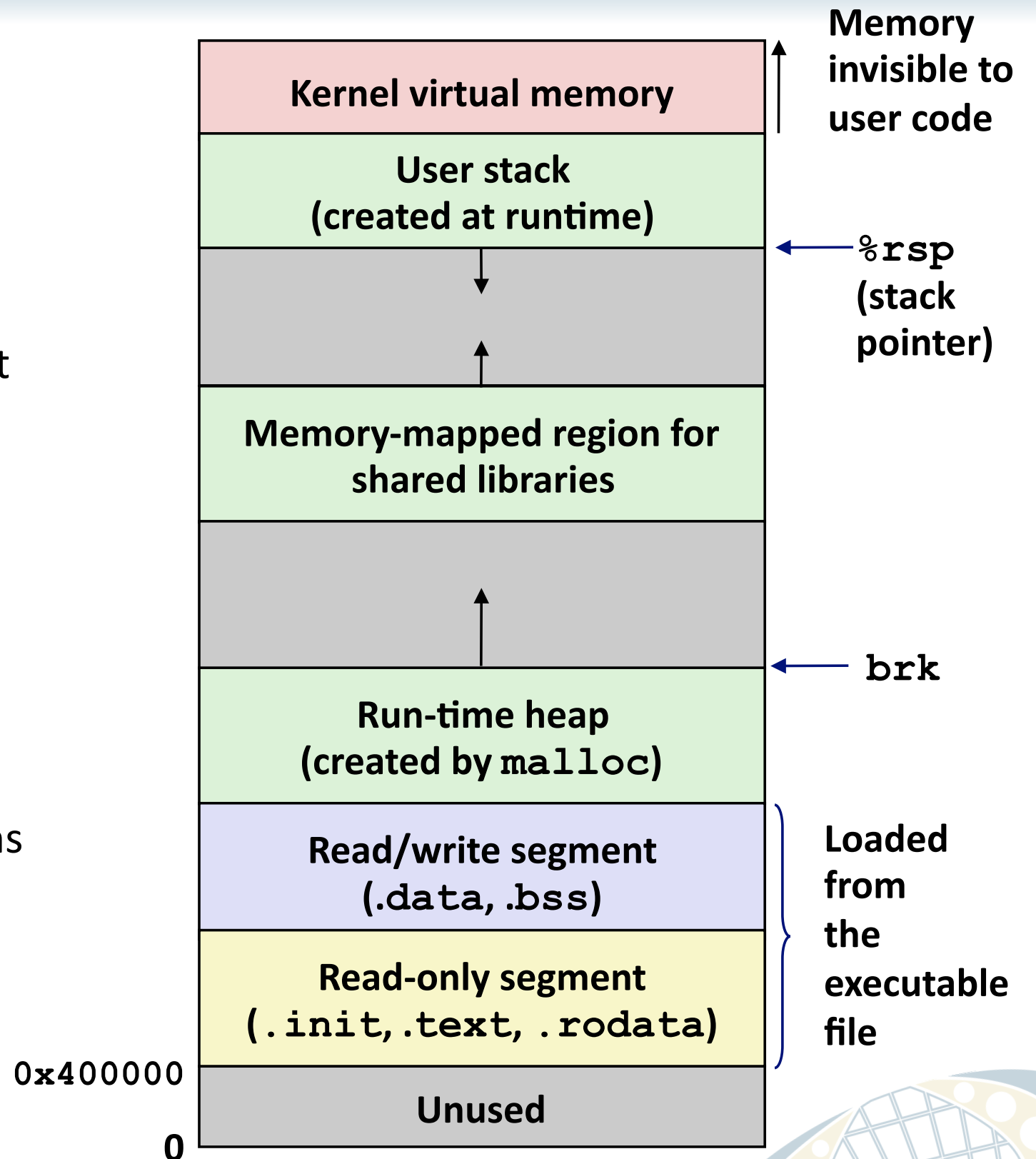
SIMPLIFYING LINKING AND LOADING

■ 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

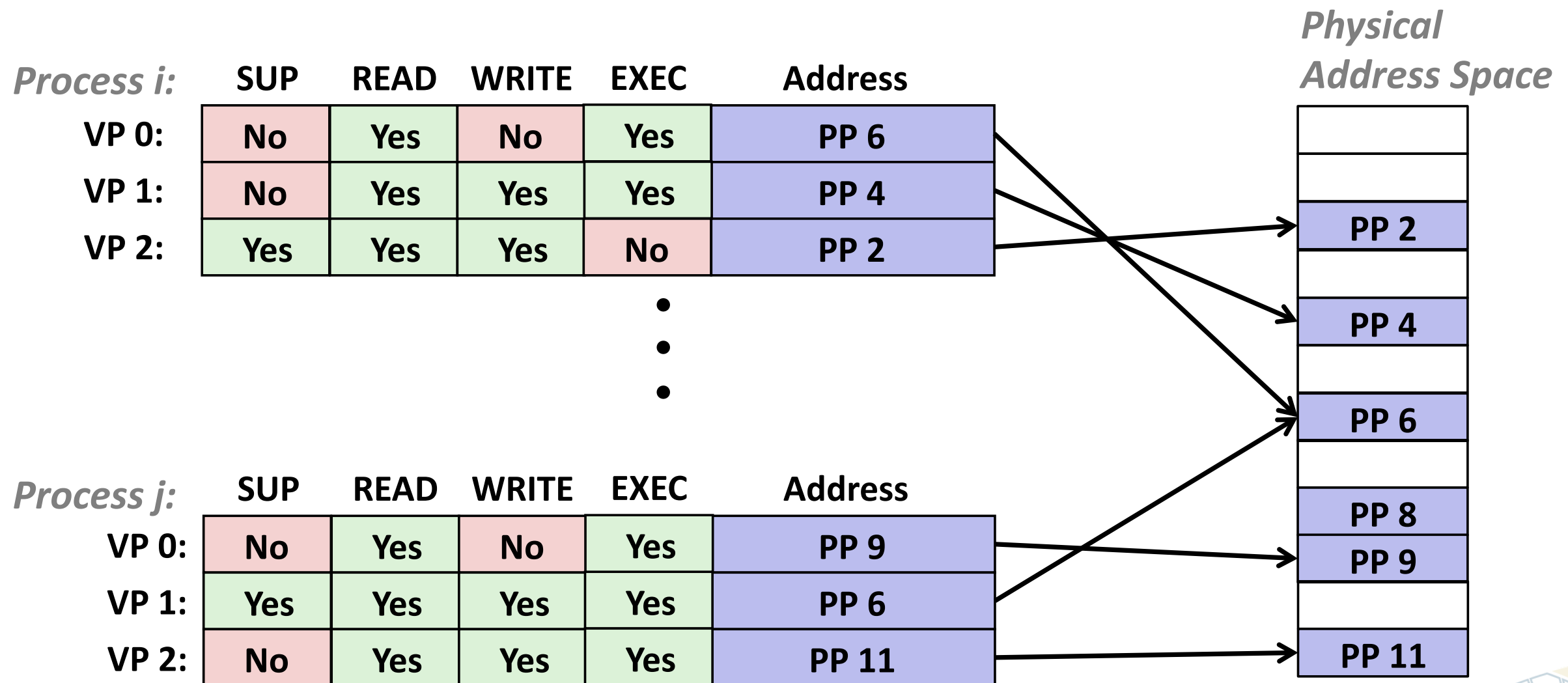


VIRTUAL MEMORY: CONCEPTS

- ADDRESS SPACES
- VM AS A TOOL FOR CACHING
- VM AS A TOOL FOR MEMORY MANAGEMENT
- VM AS A TOOL FOR MEMORY PROTECTION
- ADDRESS TRANSLATION

VM AS A TOOL FOR MEMORY PROTECTION

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



VIRTUAL MEMORY: CONCEPTS

- ADDRESS SPACES
- VM AS A TOOL FOR CACHING
- VM AS A TOOL FOR MEMORY MANAGEMENT
- VM AS A TOOL FOR MEMORY PROTECTION
- ADDRESS TRANSLATION

VM ADDRESS TRANSLATION

■ Virtual Address Space

- $V = \{0, 1, \dots, N-1\}$

■ Physical Address Space

- $P = \{0, 1, \dots, M-1\}$

■ Address Translation

- $MAP: V \rightarrow P \cup \{\emptyset\}$
- For virtual address a :
 - $MAP(a) = a'$ if data at virtual address a is at physical address a' in P
 - $MAP(a) = \emptyset$ if data at virtual address a is not in physical memory
 - Either invalid or stored on disk



SUMMARY OF ADDRESS TRANSLATION 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
- TLBT: TLB tag
- VPO: Virtual page offset
- VPN: Virtual page number

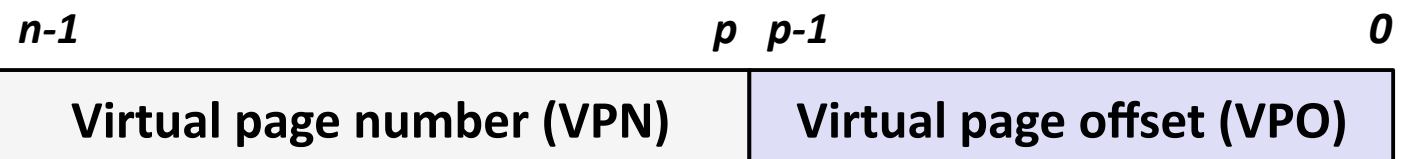
■ Components of the physical address (PA)

- PPO: Physical page offset (same as VPO)
- PPN: Physical page number



ADDRESS TRANSLATION WITH A PAGE TABLE

Virtual address



Page table
base register
(PTBR)

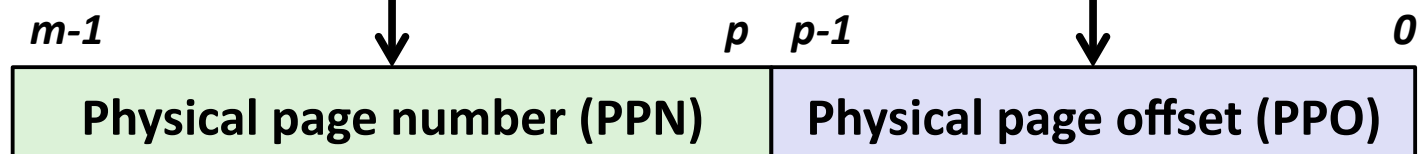
Physical page table
address for the current
process

Page table

Valid	Physical page number (PPN)

Valid bit = 0:
Page not in memory
(page fault)

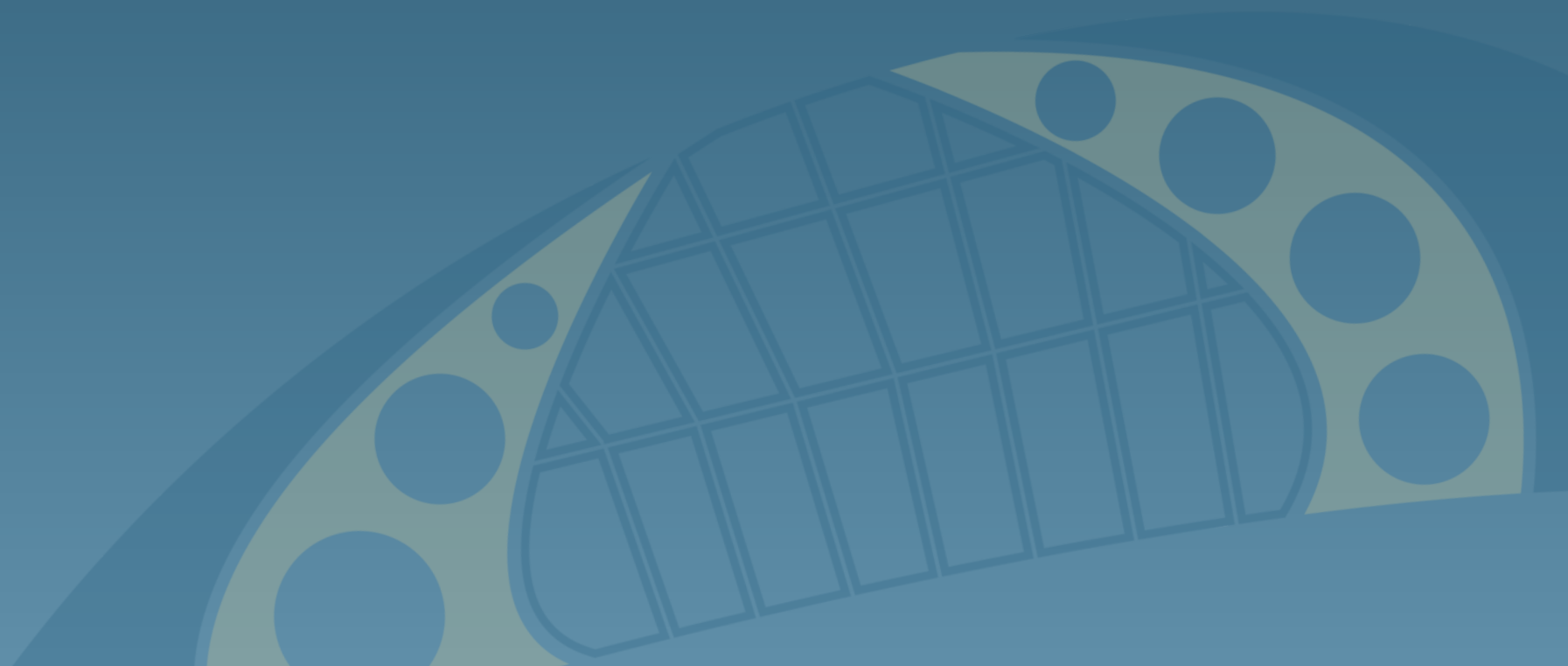
Valid bit = 1



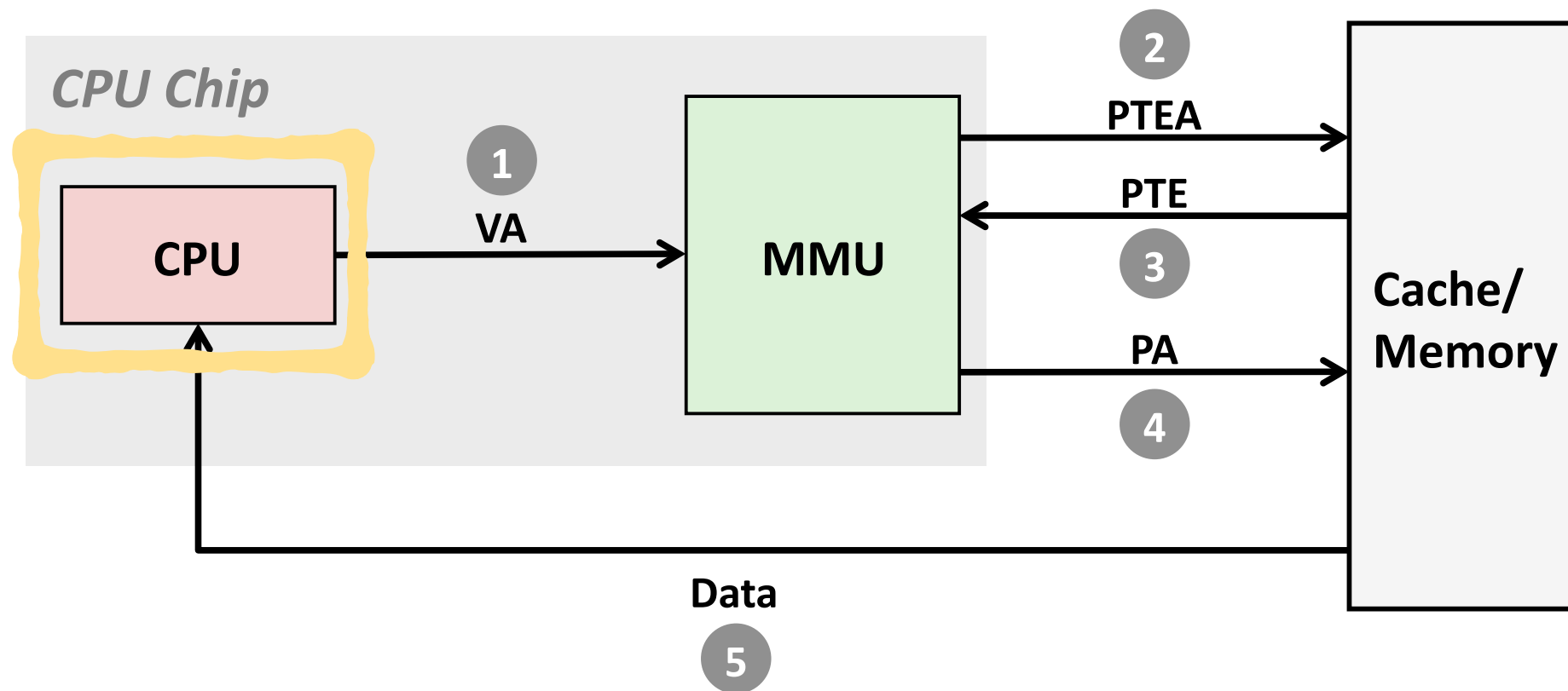
Physical address



ADDRESS TRANSLATION: PAGE HIT

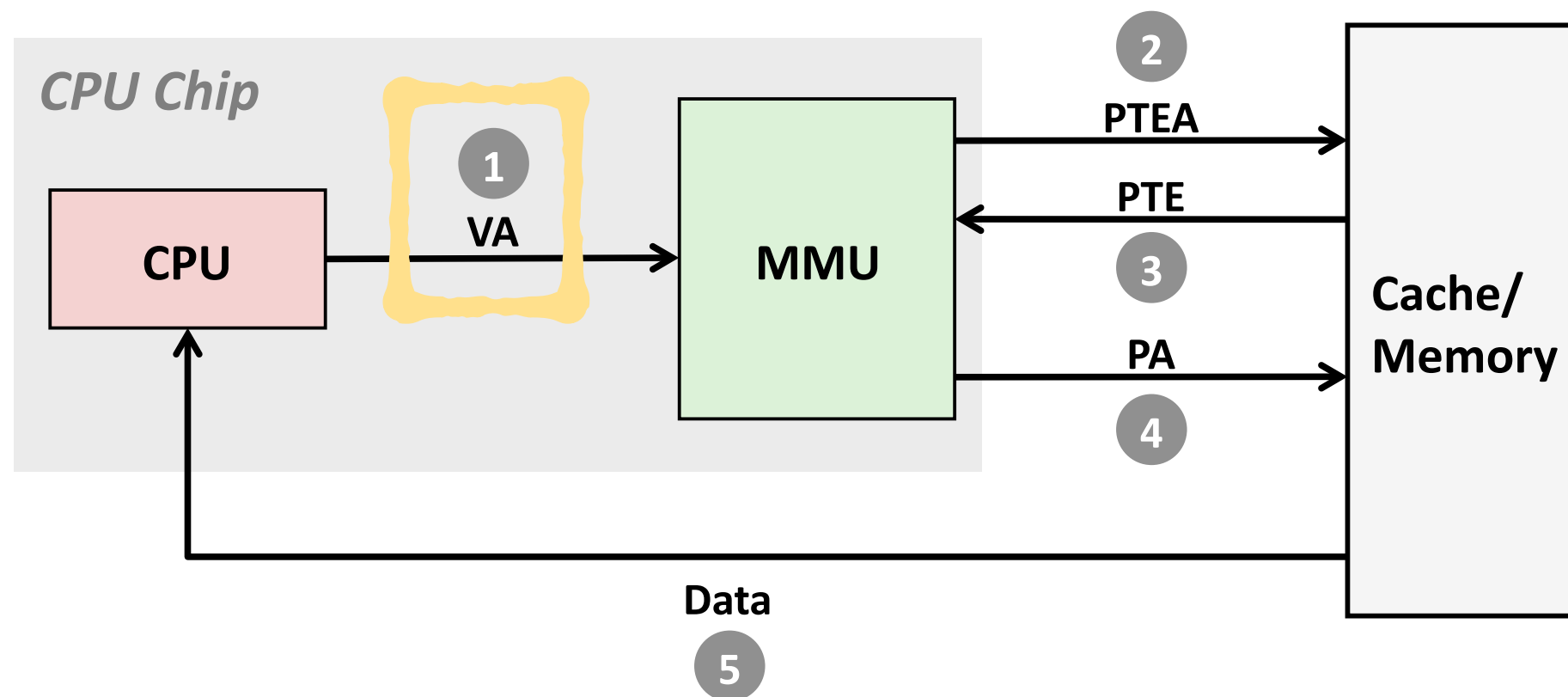


ADDRESS TRANSLATION: PAGE HIT



- 1) Processor sends virtual address to MMU `mov (%rax), %rdx`
- 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 HIT



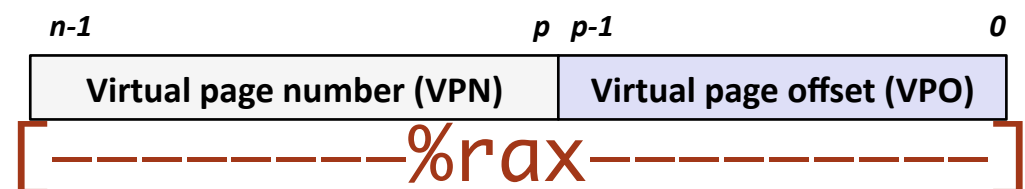
1) Processor sends virtual address to MMU

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

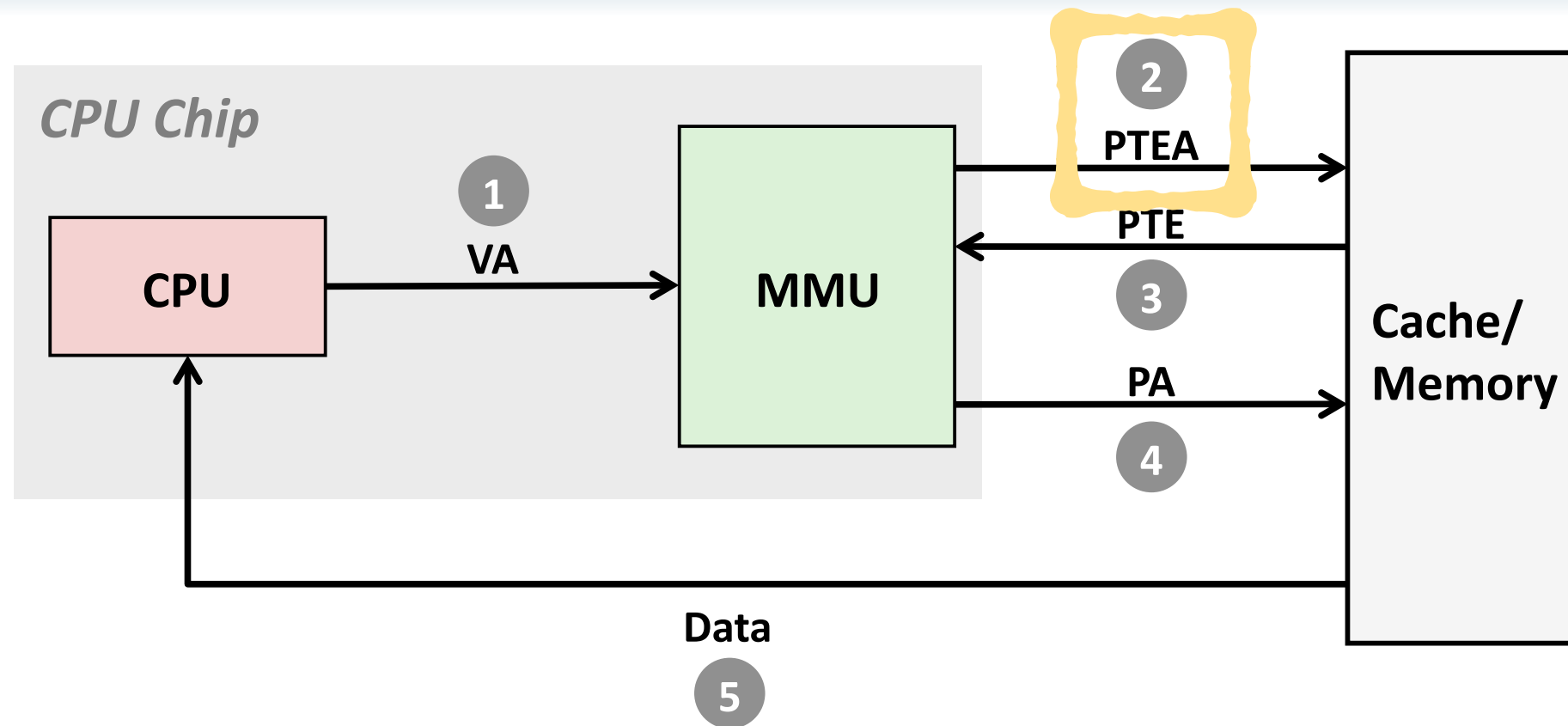
4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Virtual address



ADDRESS TRANSLATION: PAGE HIT



1) Processor sends virtual address to MMU

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

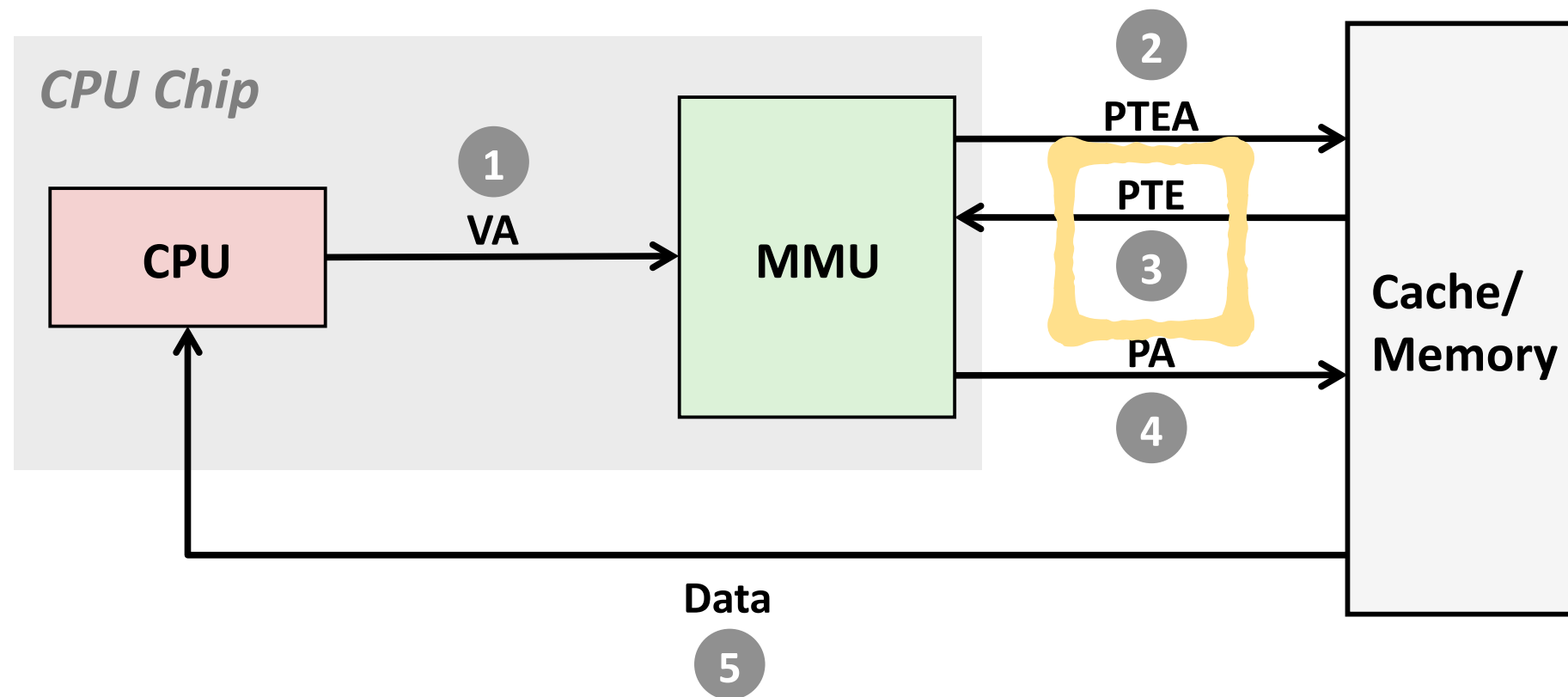
4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Page table

Valid	Physical page number (PPN)

ADDRESS TRANSLATION: PAGE HIT



1) Processor sends virtual address to MMU

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

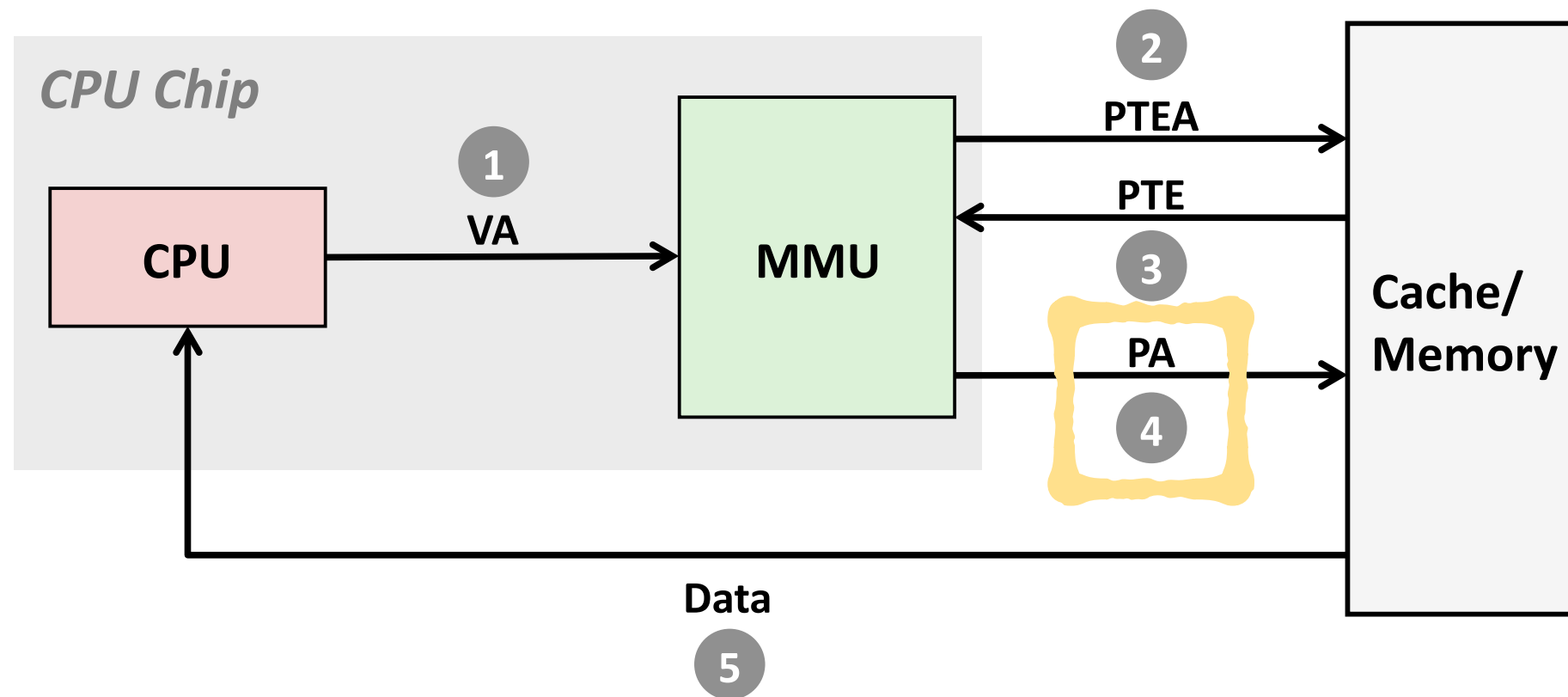
4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Page table

Valid	Physical page number (PPN)
	$m-1$ p
	Physical page number (PPN)

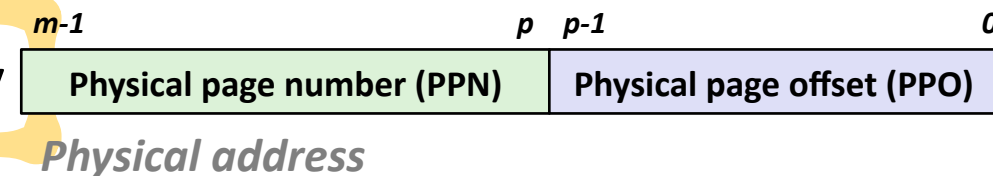
ADDRESS TRANSLATION: PAGE HIT



1) Processor sends virtual address to MMU

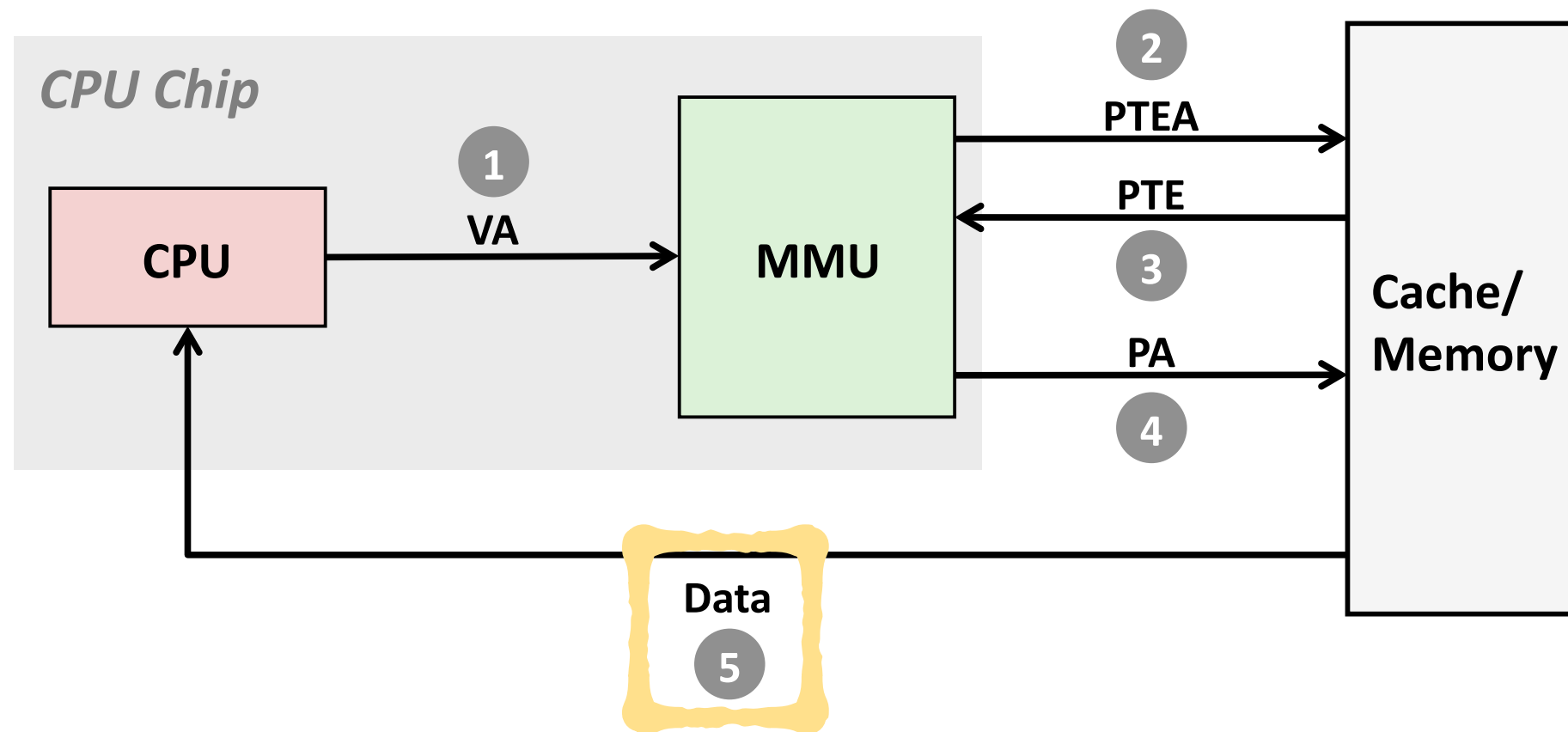
2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory



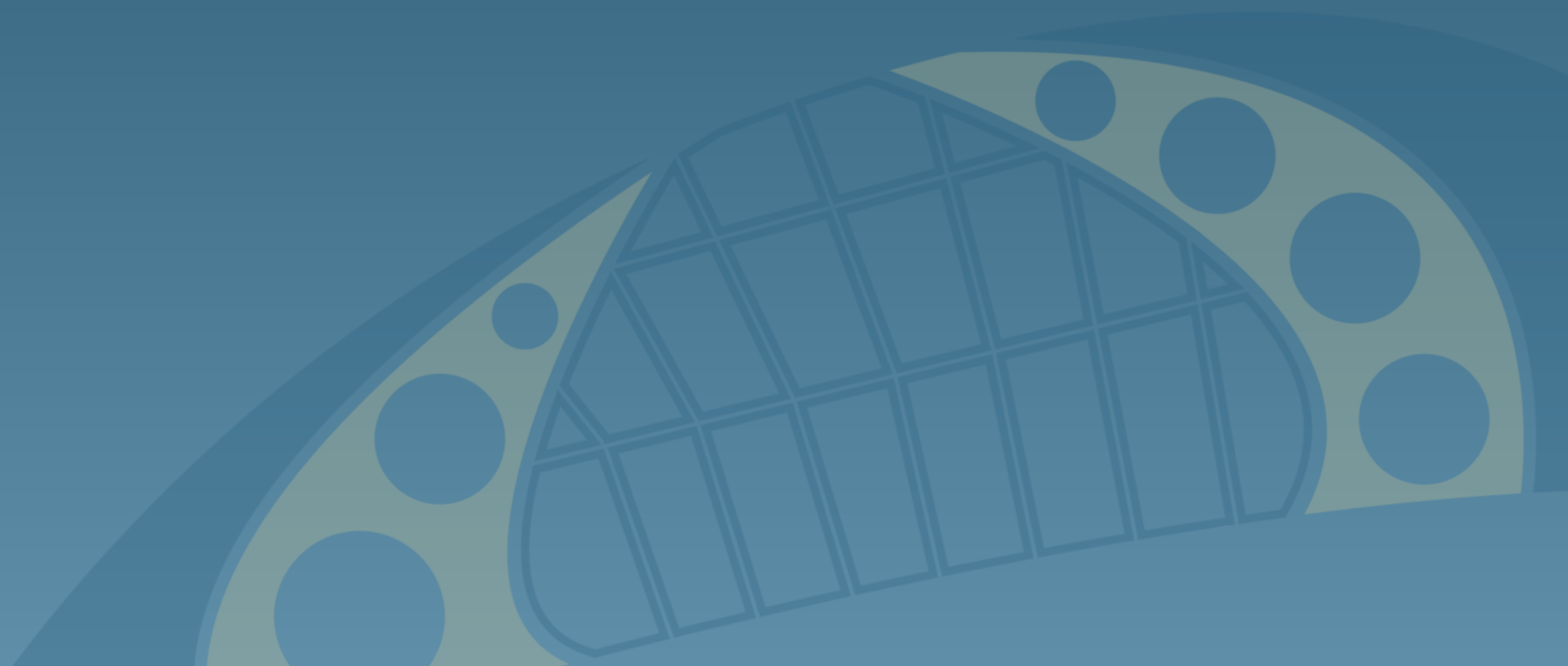
5) Cache/memory sends data word to processor

ADDRESS TRANSLATION: PAGE HIT

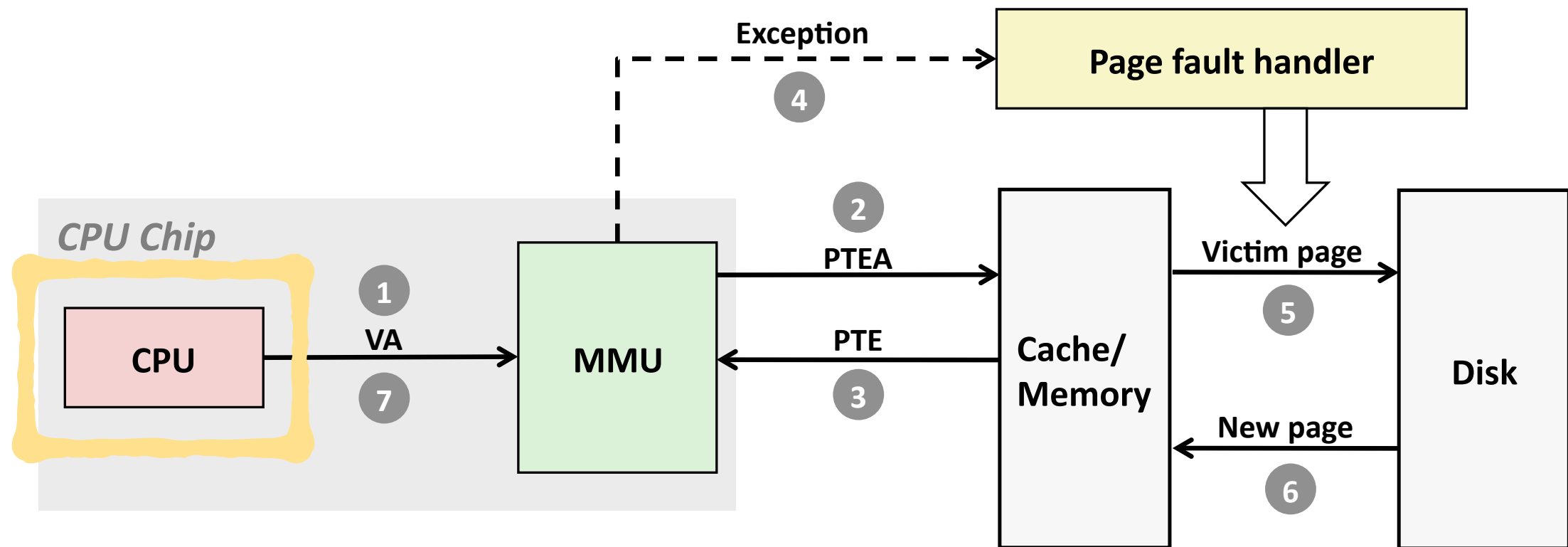


- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) MMU sends physical address to cache/memory
- 5) Cache/memory sends data word to processor

ADDRESS TRANSLATION: PAGE FAULT



ADDRESS TRANSLATION: PAGE FAULT



1) Processor sends virtual address to MMU

`mov (%rax), %rdx`

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

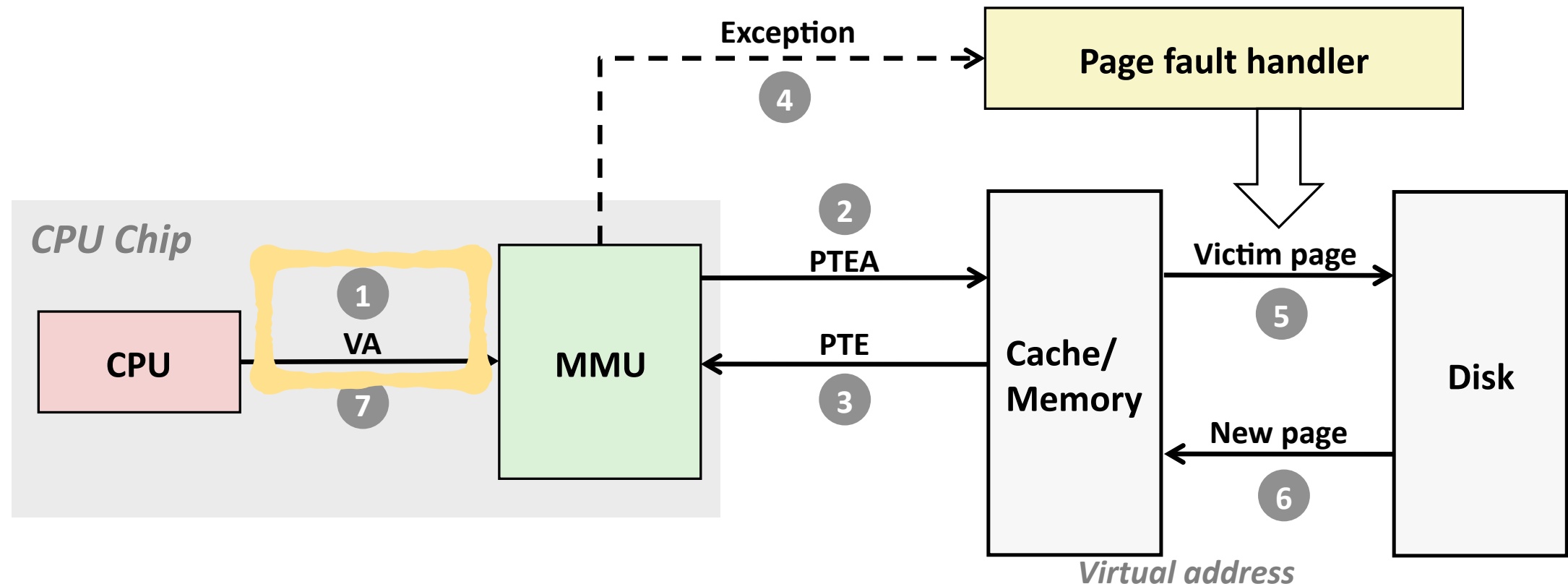
4) Valid bit is zero, so MMU triggers page fault exception

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

ADDRESS TRANSLATION: PAGE FAULT



1) Processor sends virtual address to MMU

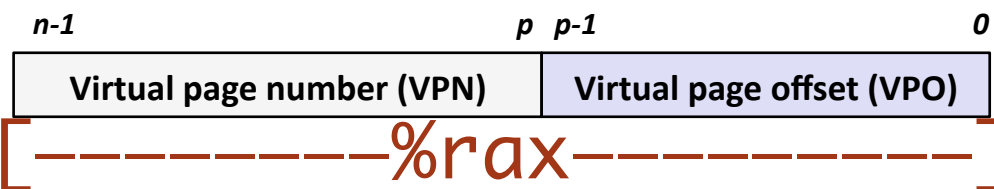
2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception

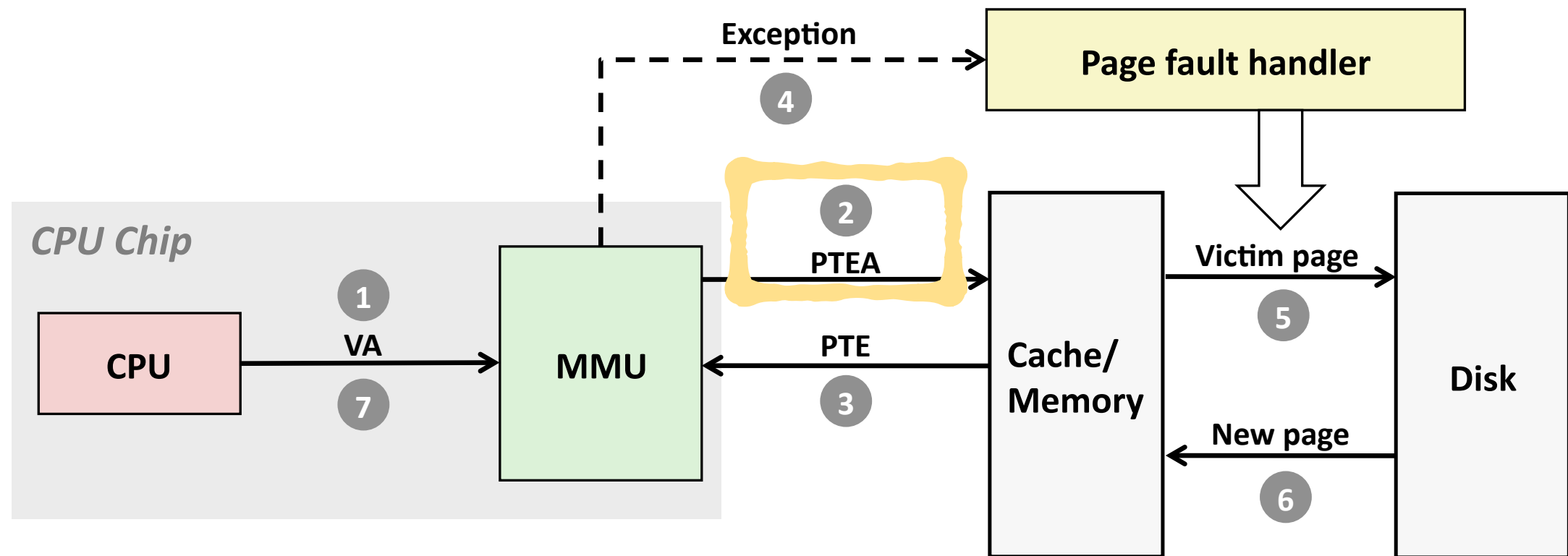
5) Handler identifies victim (and, if dirty, pages it out to disk)

6) Handler pages in new page and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction



ADDRESS TRANSLATION: PAGE FAULT

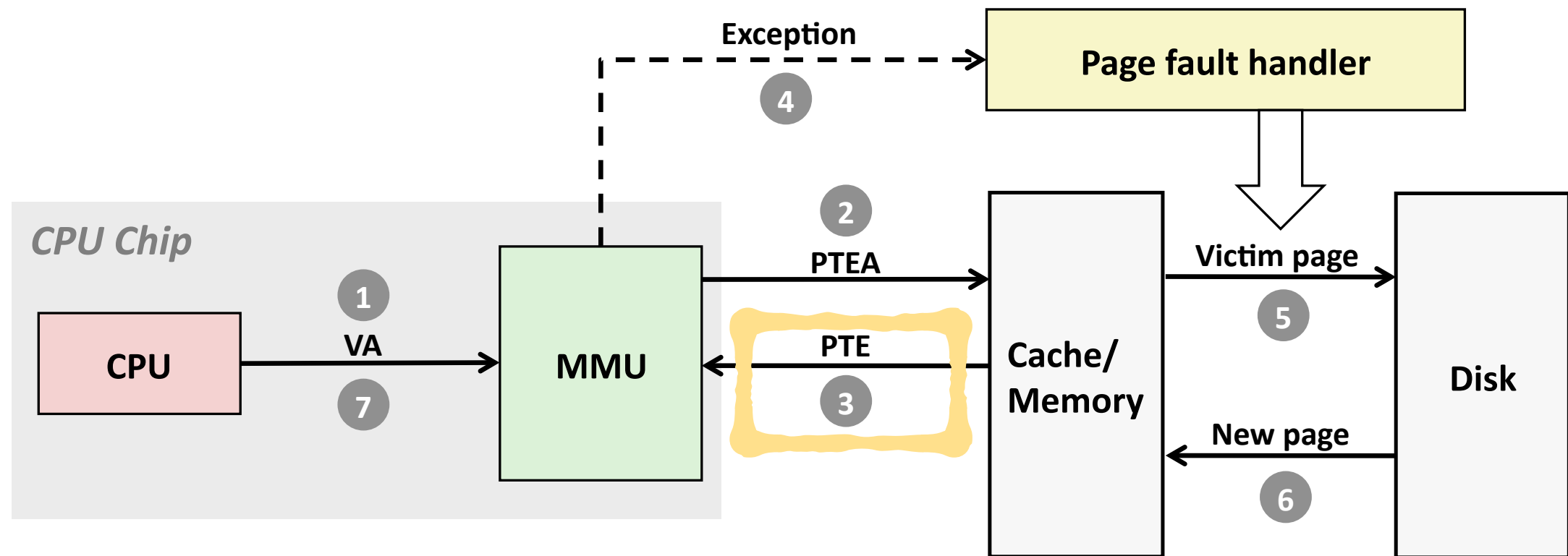


Page table

Valid	Physical page number (PPN)

- 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

ADDRESS TRANSLATION: PAGE FAULT

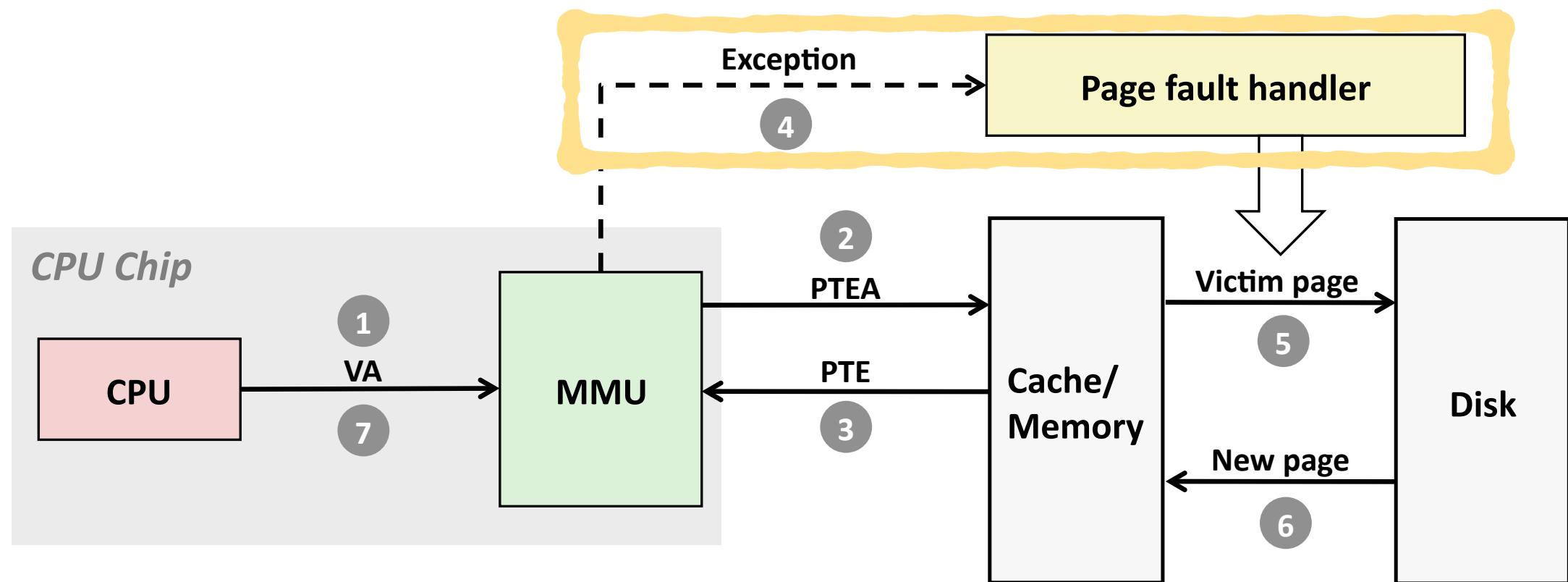


Page table

Valid	Physical page number (PPN)
0	

- 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

ADDRESS TRANSLATION: PAGE FAULT

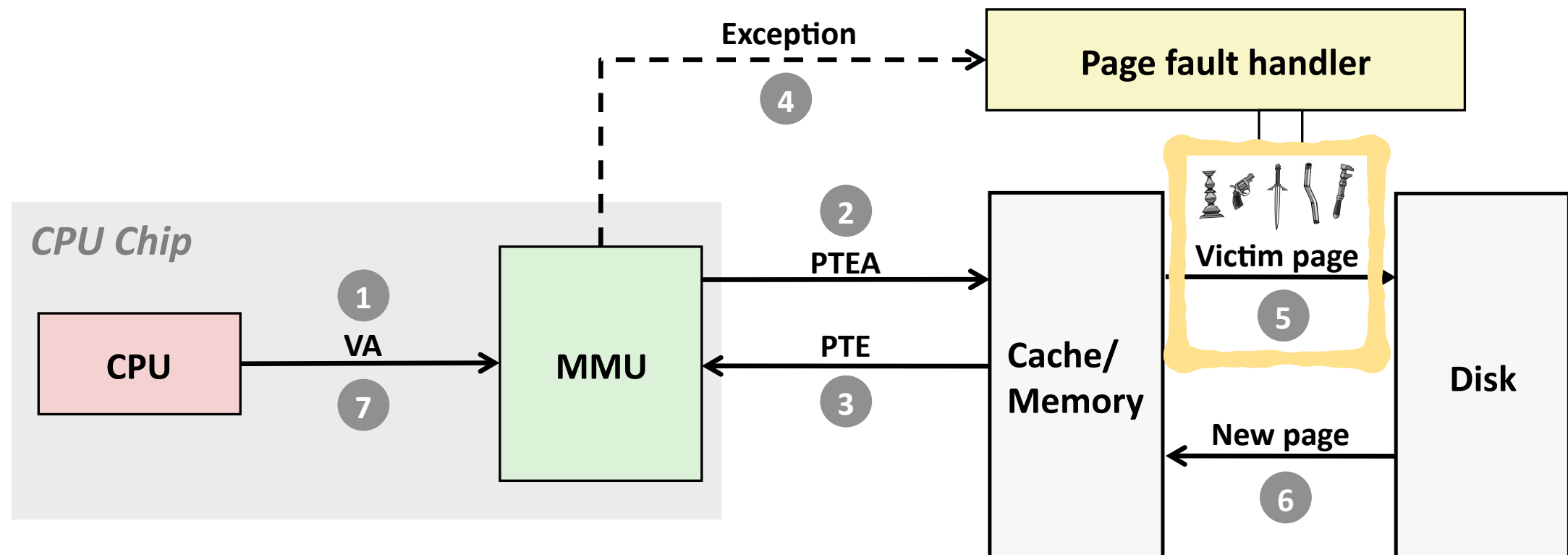


Page table

Valid	Physical page number (PPN)
0	

- 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

ADDRESS TRANSLATION: PAGE FAULT

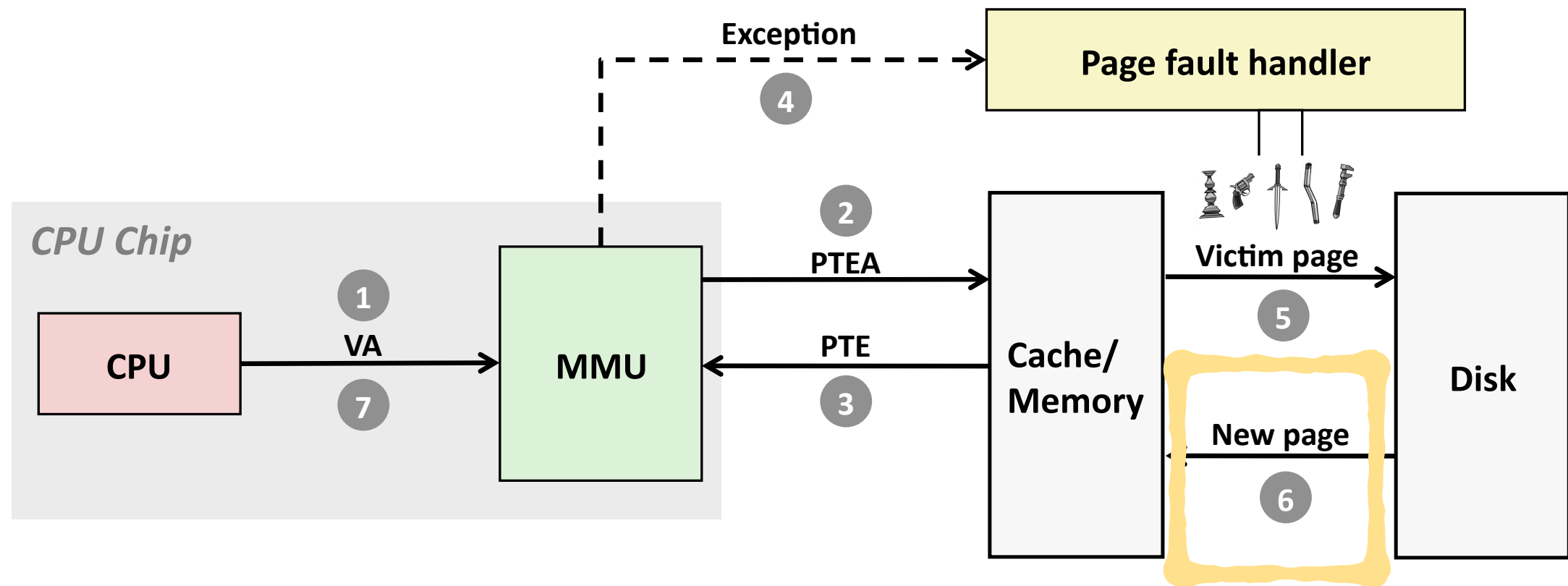


Page table

Valid	Physical page number (PPN)
0	

- 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

ADDRESS TRANSLATION: PAGE FAULT

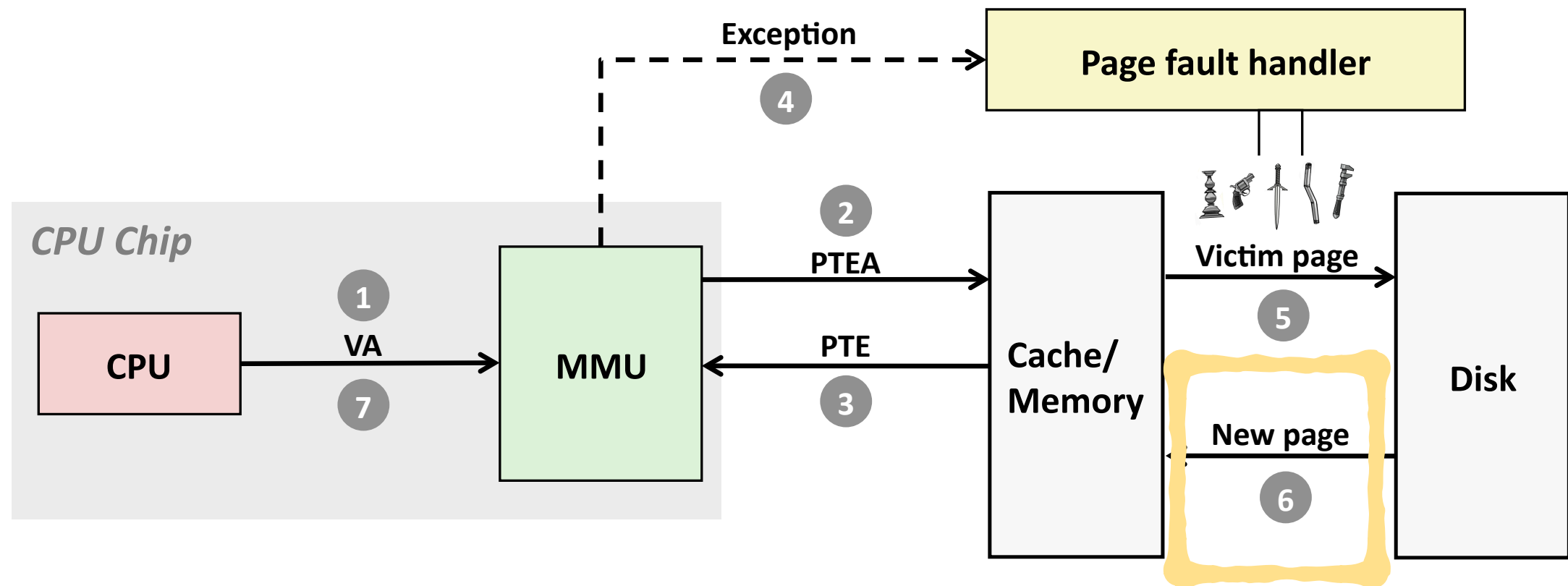


- 1) Processor sends virtual address to MMU
- 2-3) MMU fetches PTE from page table in memory
- 4) Valid bit is zero, so MMU triggers page fault exception
- 5) Handler identifies victim (and, if dirty, pages it out to disk)
- 6) Handler pages in new page and updates PTE in memory
- 7) Handler returns to original process, restarting faulting instruction

Page table

Valid	Physical page number (PPN)	
	$m-1$	p
	Physical page number (PPN)	

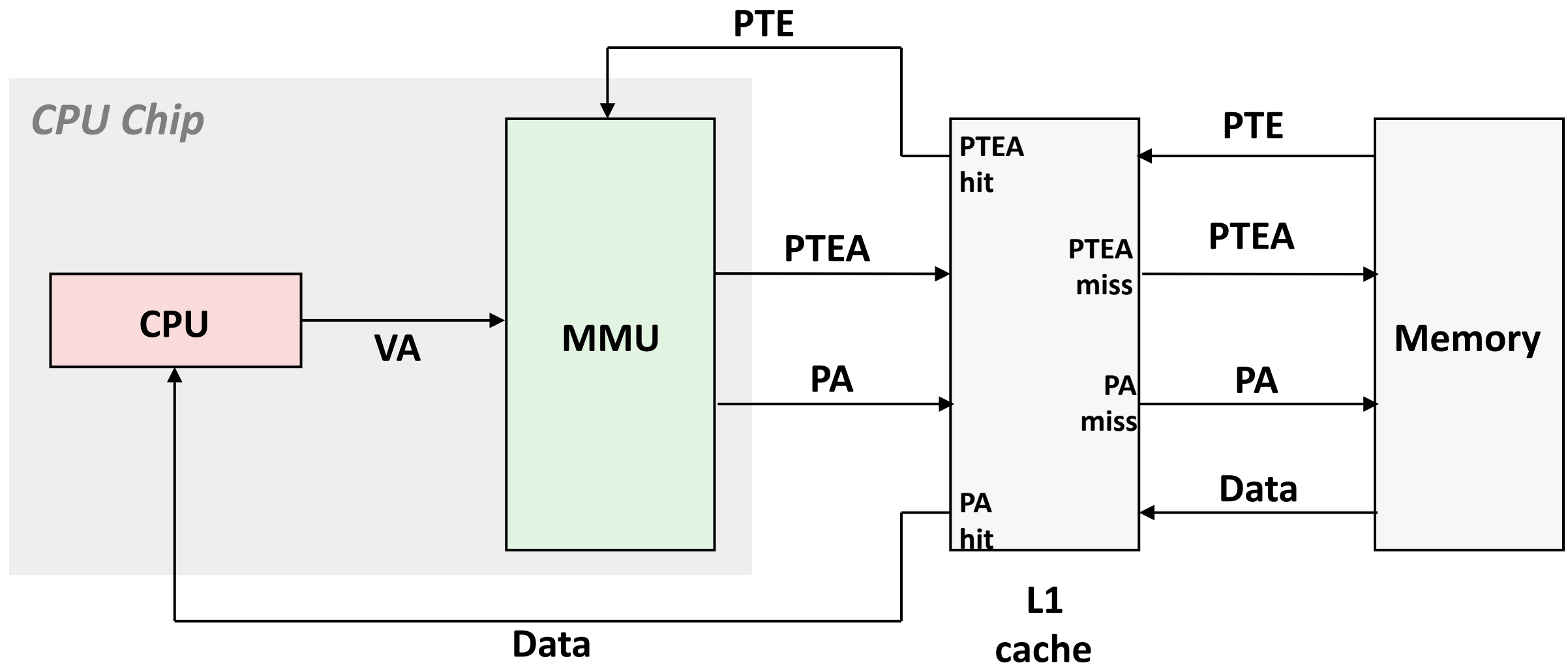
ADDRESS TRANSLATION: PAGE FAULT



- 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

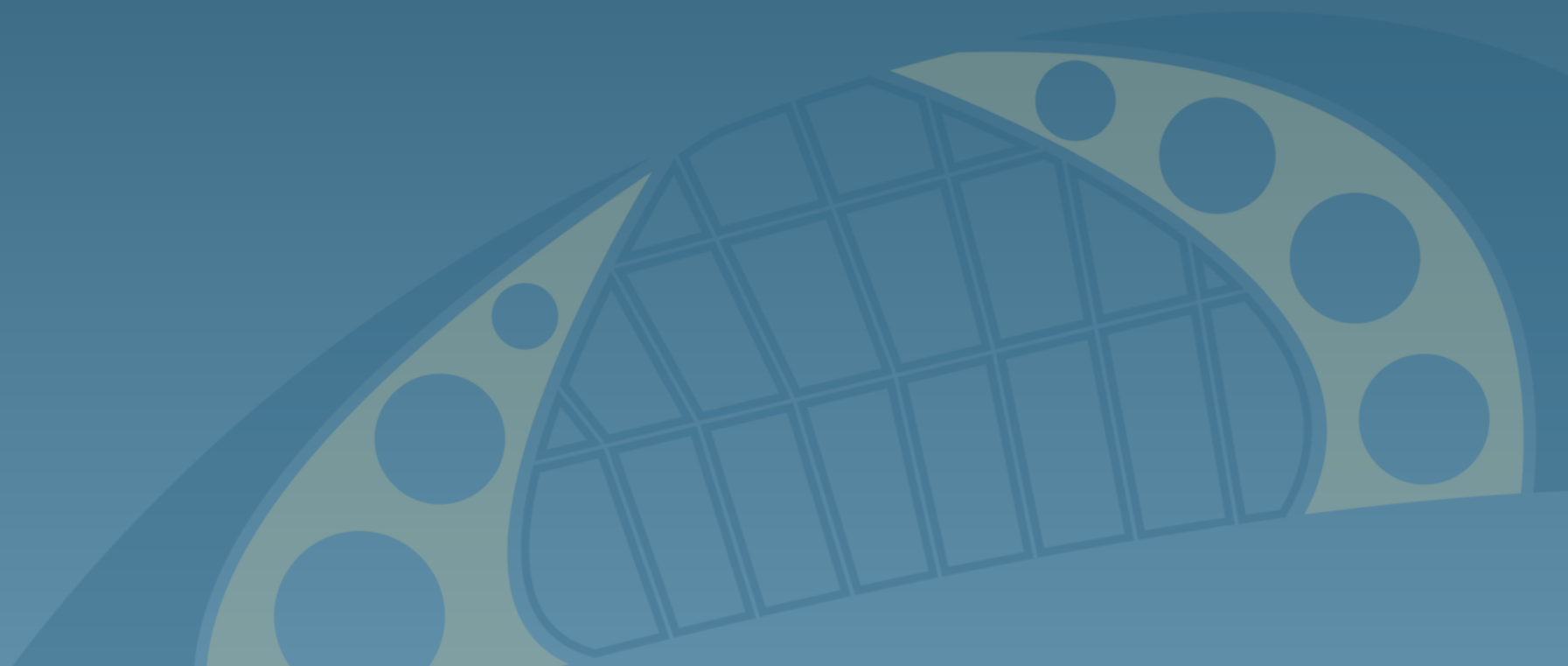
`mov (%rax), %rdx`

INTEGRATING VM AND CACHE



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

TRANSLATION LOOKASIDE BUFFER



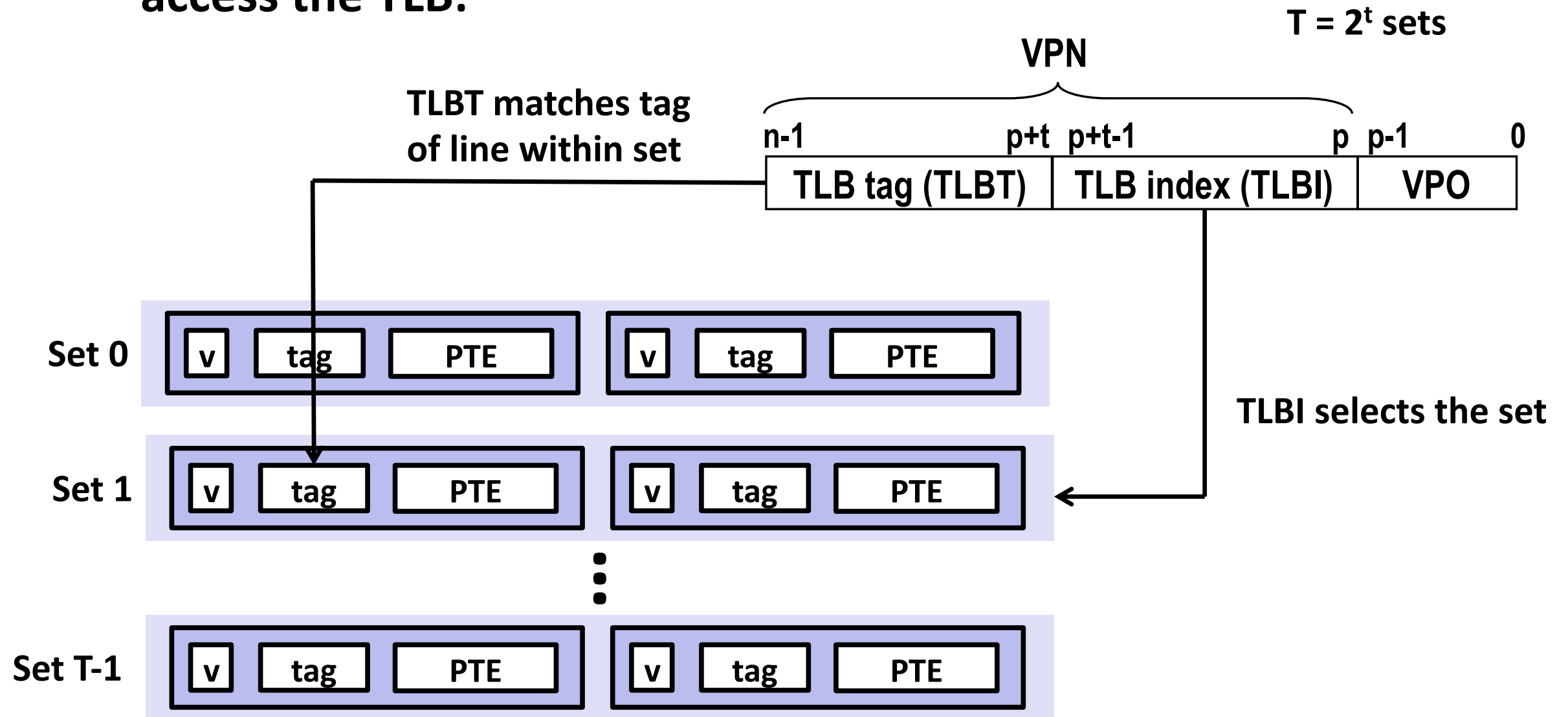
SPEEDING UP TRANSLATION WITH A TLB

- **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 small L1 delay
- **Solution: *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

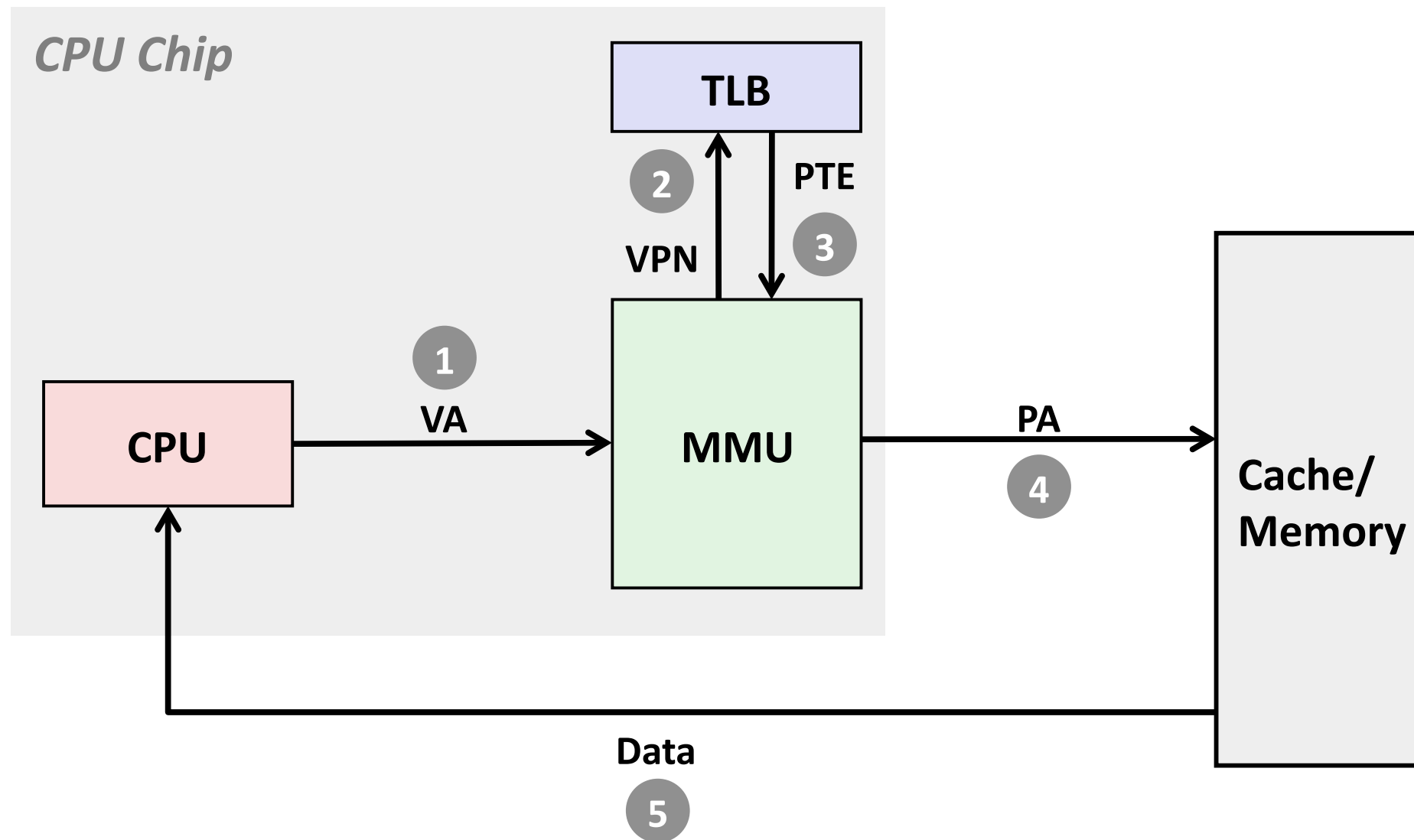


ACCESSING THE TLB

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

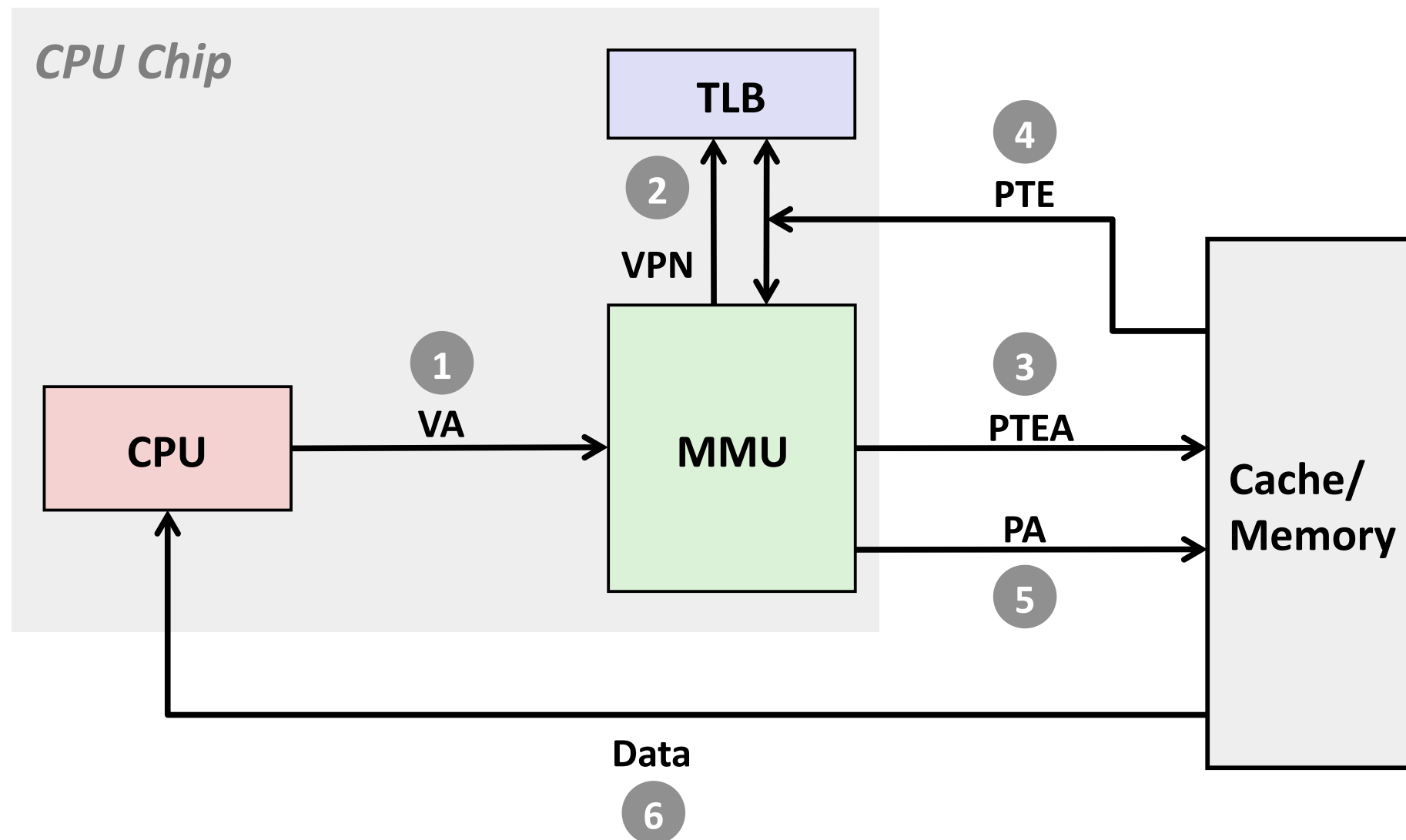


TLB HIT



A TLB hit eliminates a memory access

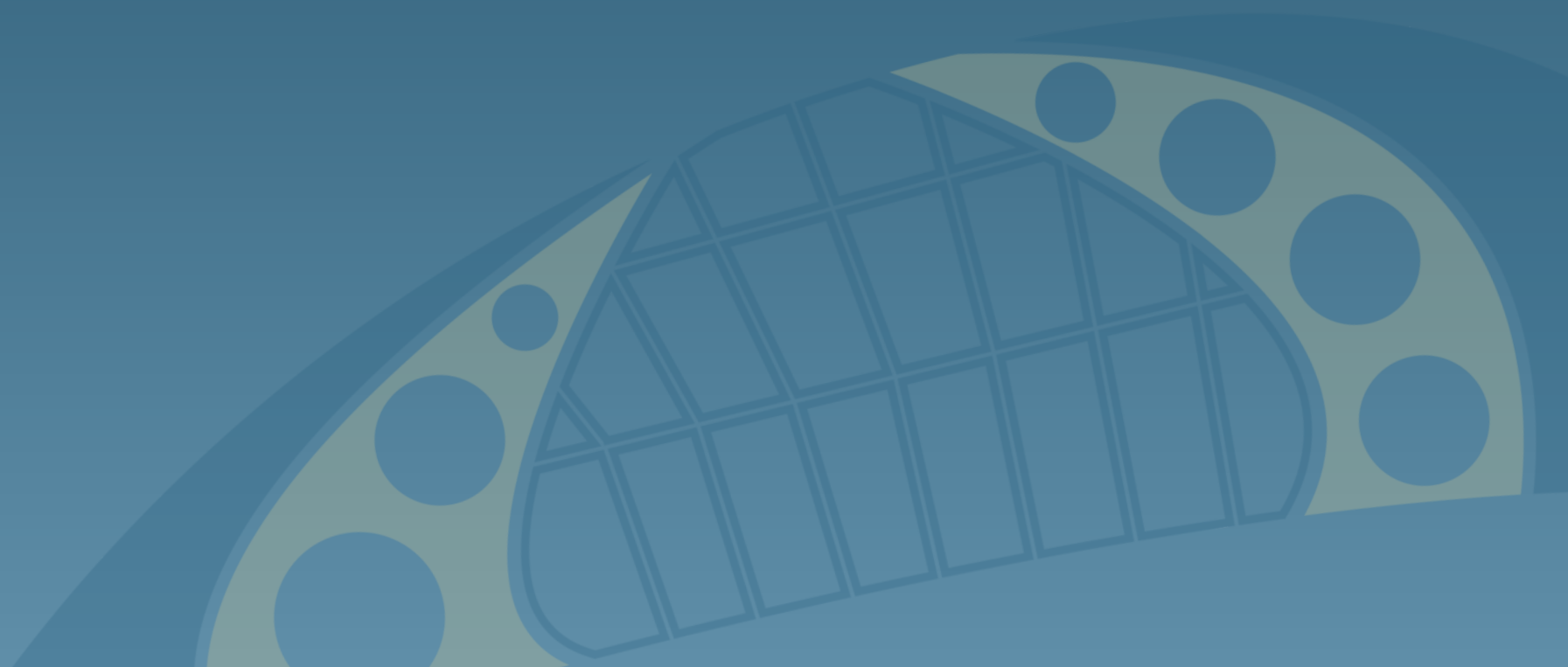
TLB MISS



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

Fortunately, TLB misses are rare. Why?

SUMMARY

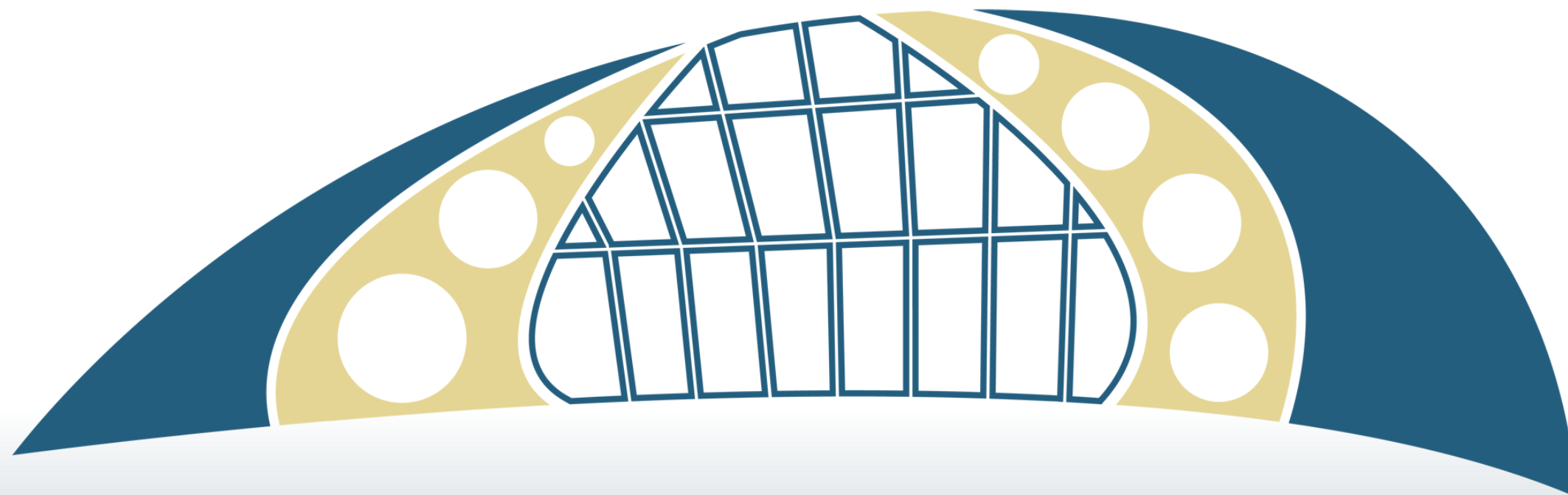


VIRTUAL MEMORY CONCEPTS

SUMMARY

- **Programmer's view of virtual memory**
 - Each process has its own private linear address space
 - Cannot be corrupted by other processes
- **System view of virtual memory**
 - Uses memory efficiently by caching virtual memory pages
 - Efficient only because of locality
 - Simplifies memory management and programming
 - Simplifies protection by providing a convenient interpositioning point to check permissions





WESTMONT **INSPIRED**
— COMPUTING LAB —