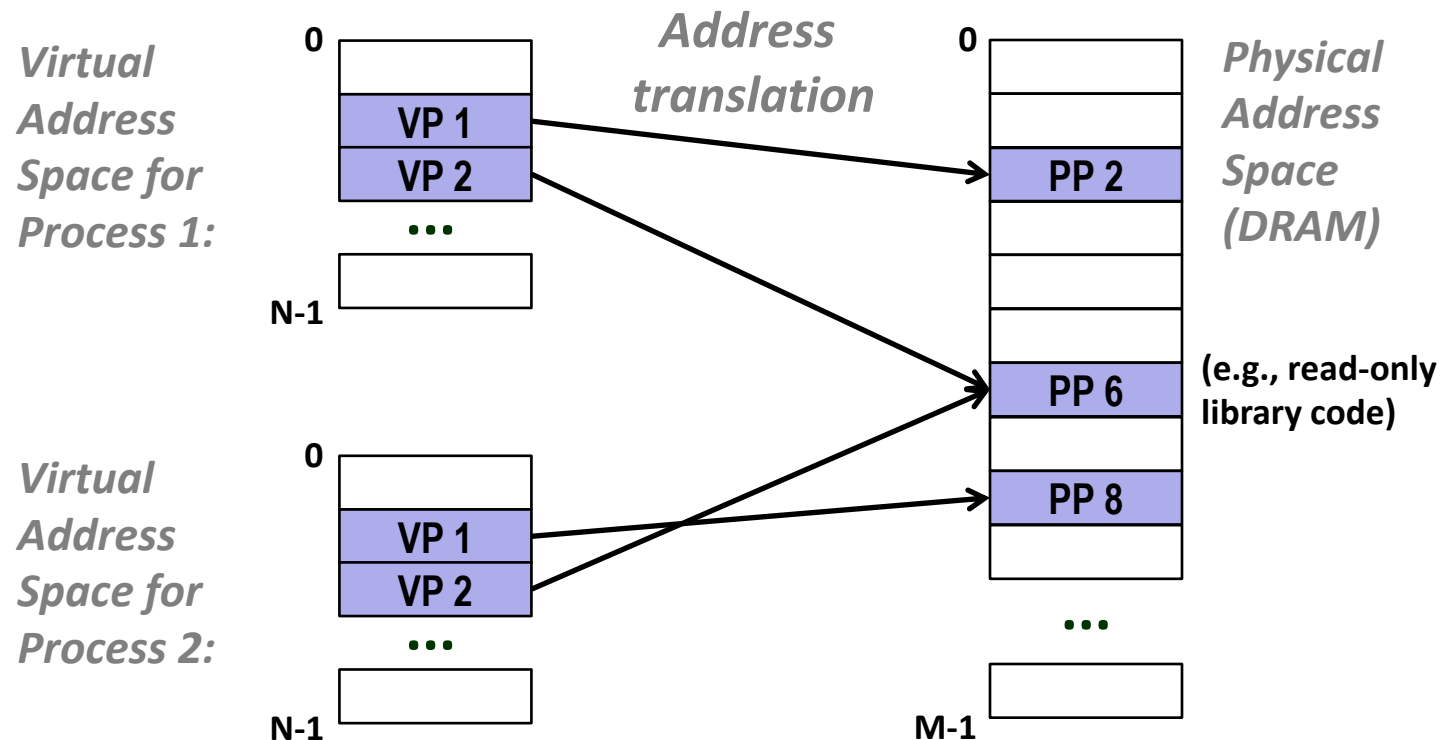


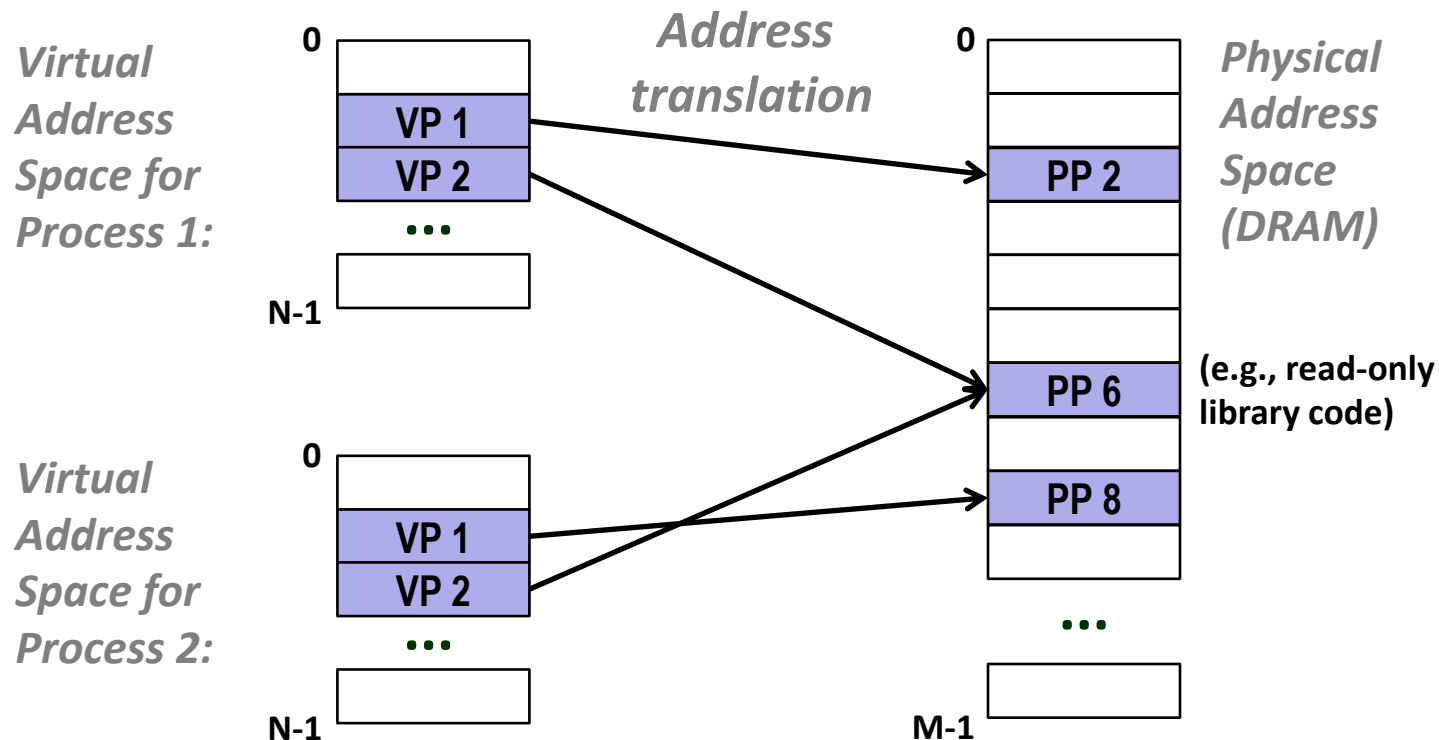
VM for Managing Multiple Processes

- **Key abstraction: each process has its own virtual address space**
 - It can view memory as *a simple linear array*
- **With virtual memory, this simple linear virtual address space need not be contiguous in physical memory**
 - Process needs to store data in another VP? Just map it to *any* PP!



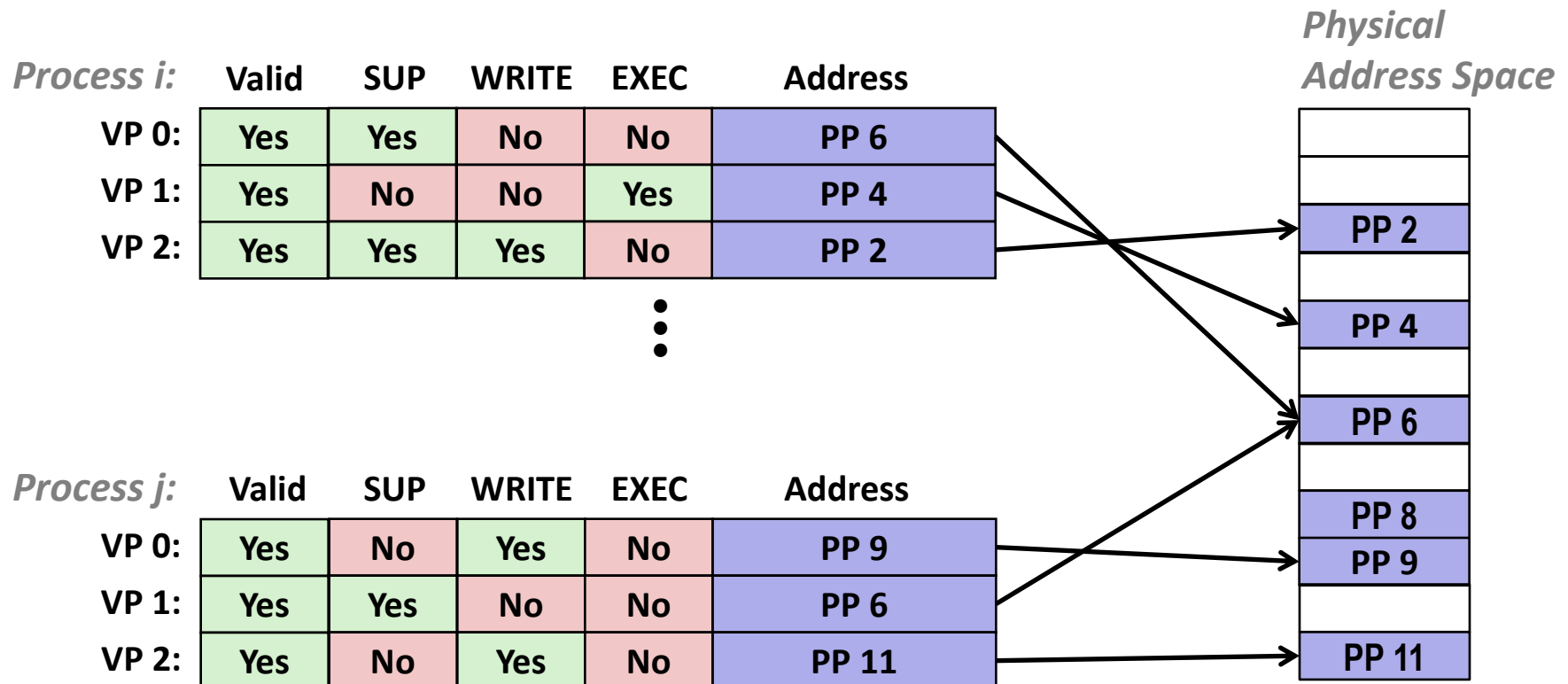
VM for Protection and Sharing

- The mapping of VPs to PPs provides a simple mechanism for *protecting* memory and for *sharing* memory btw. processes
 - Sharing: just map virtual pages in separate address spaces to the same physical page (here: PP 6)
 - Protection: process simply can't access physical pages it doesn't have a mapping for (here: Process 2 can't access PP 2)

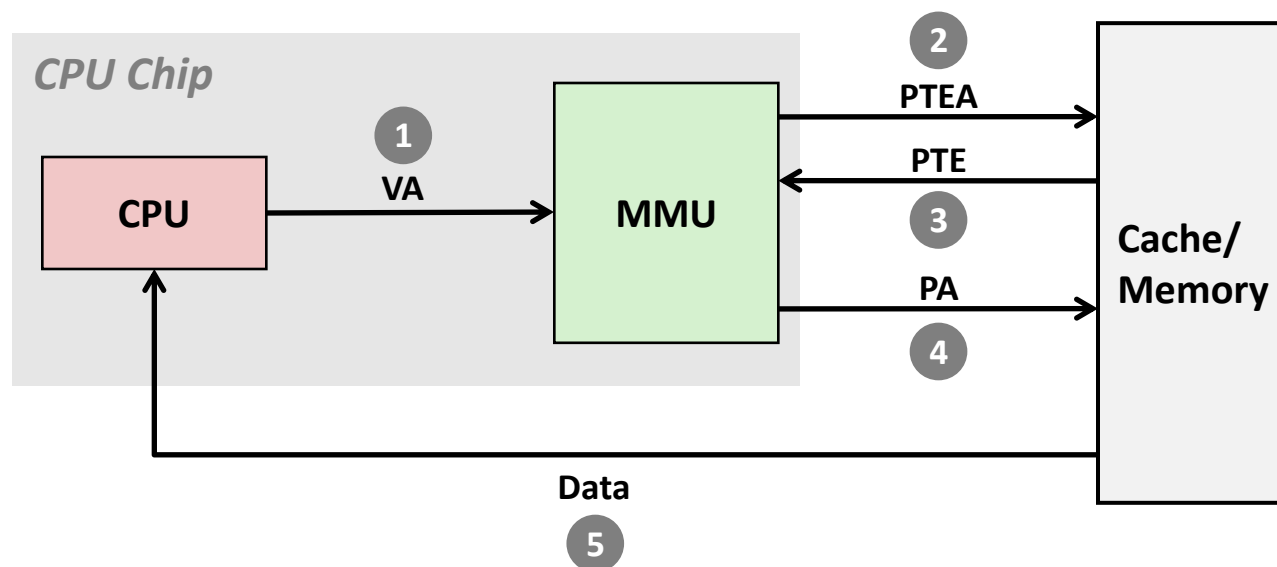


Memory Protection Within a Single Process

- Extend PTEs with permission bits
- MMU checks these permission bits on every memory access
 - If violated, raises exception and OS sends SIGSEGV signal to process

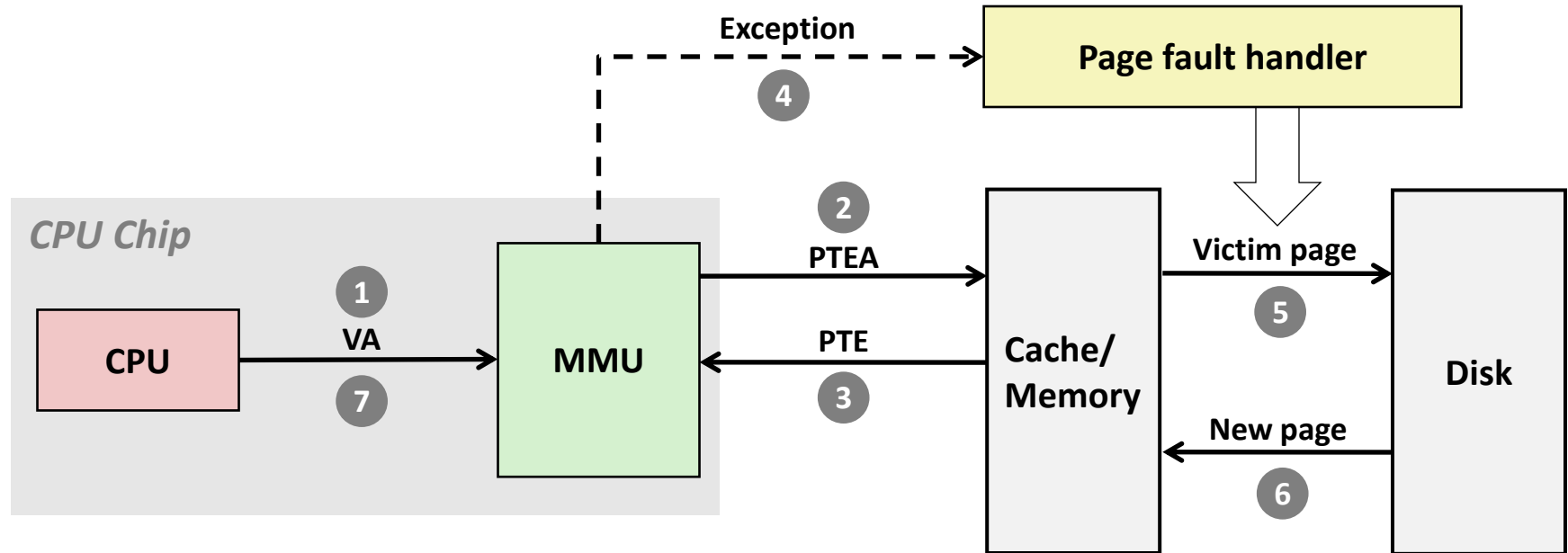


Address Translation: Page Hit



- 1) Processor sends virtual address to MMU (*memory management unit*)
- 2-3) MMU fetches PTE from page table in cache/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 cache/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

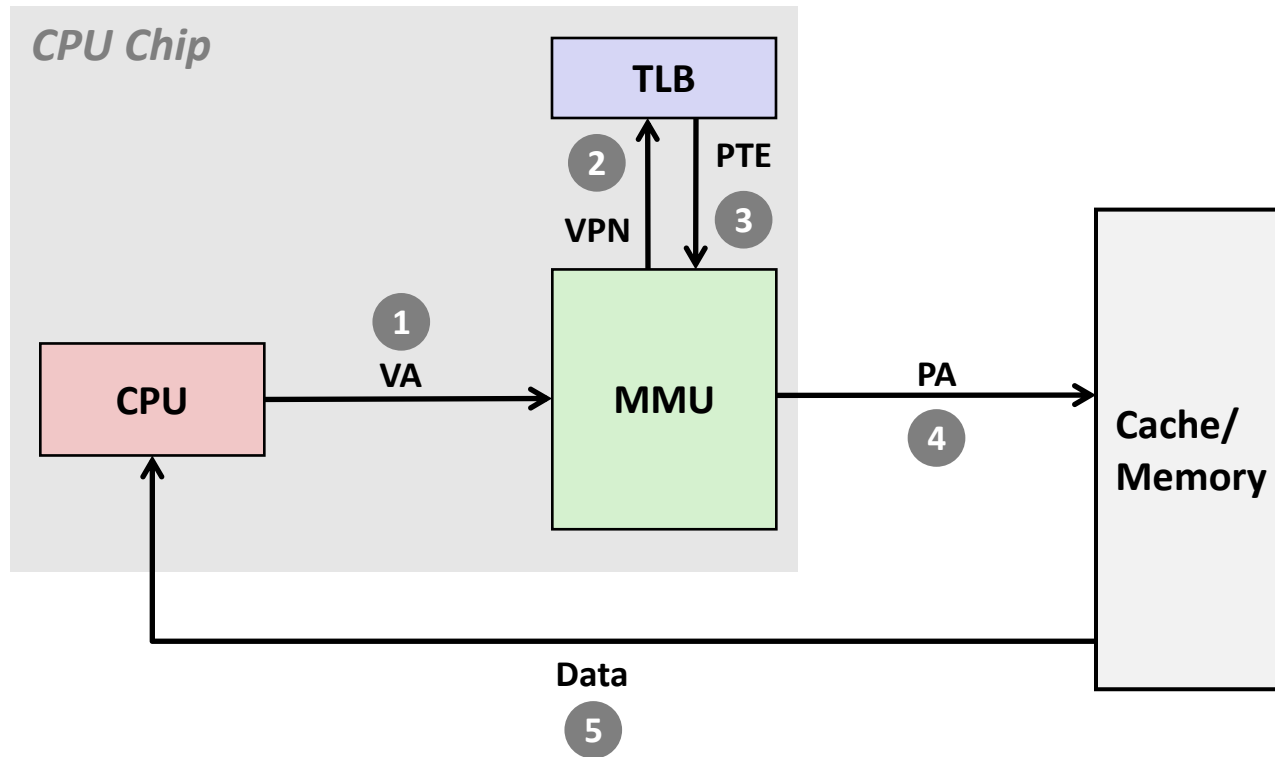
Hmm... Translation Sounds Slow!

- The MMU accesses memory *twice*: once to first get the PTE for translation, and then again for the actual memory request from the CPU
 - The PTEs *may* be cached in L1 like any other memory word
 - But they may be evicted by other data references
 - And a hit in the L1 cache still requires 1-3 cycles
- *What can we do to make this faster?*

Speeding up Translation with a TLB

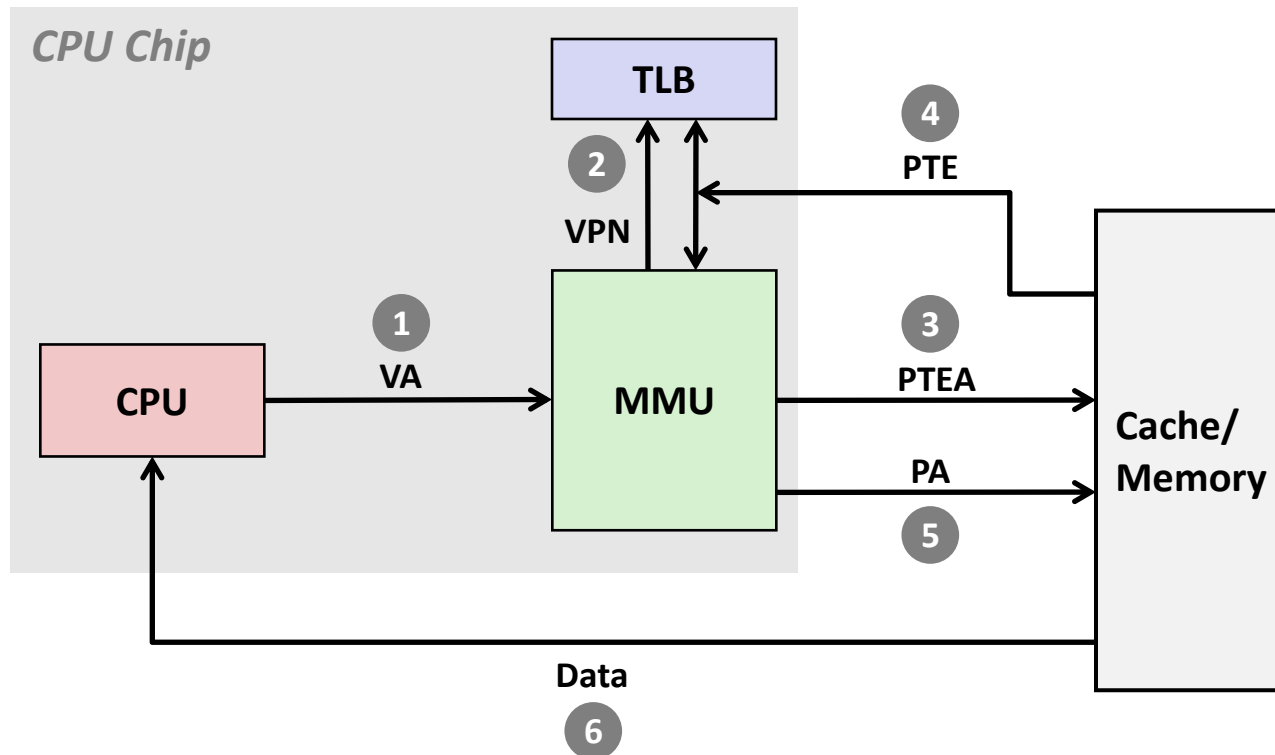
- Solution: add another cache!
- ***Translation Lookaside Buffer*** (TLB):
 - Small hardware cache in MMU
 - Maps virtual page numbers to physical page numbers
 - Contains complete *page table entries* for small number of pages
 - Modern Intel processors: 128 or 256 entries in TLB

TLB Hit



A TLB hit eliminates a memory access

TLB Miss



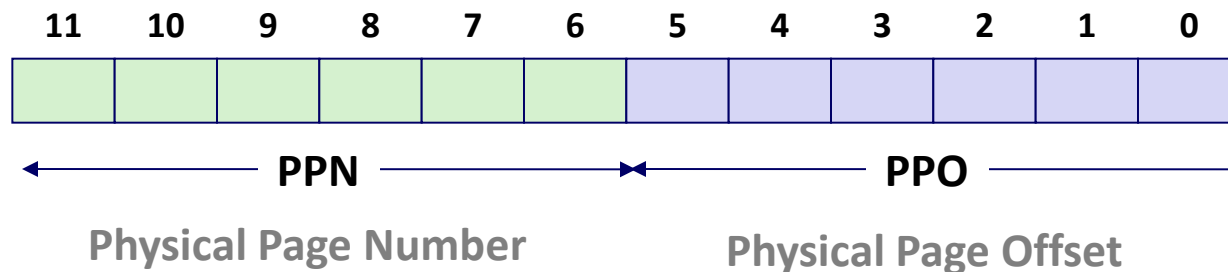
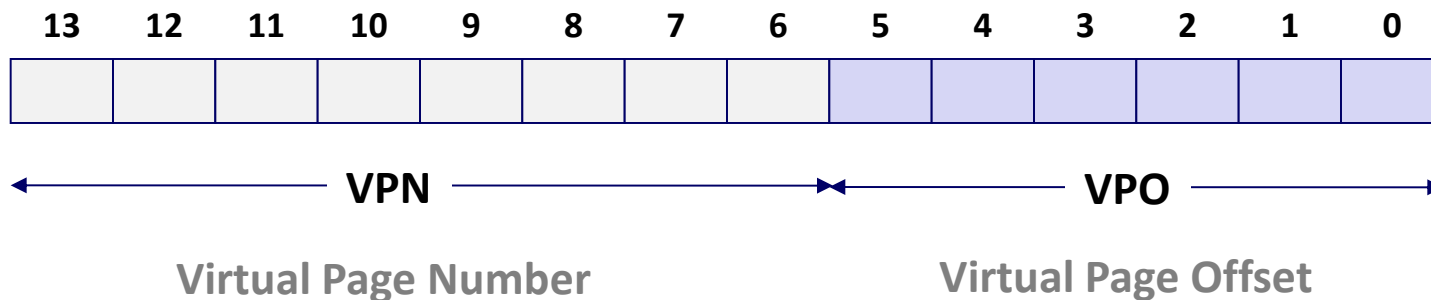
A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare

Simple Memory System Example

■ Addressing

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



Simple Memory System Page Table

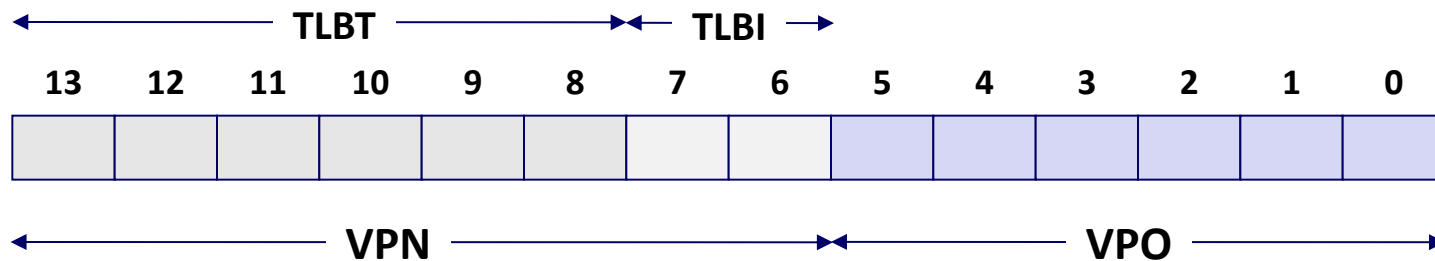
- Only showing first 16 entries (out of 256)

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
00	28	1
01	–	0
02	33	1
03	02	1
04	–	0
05	16	1
06	–	0
07	–	0

<i>VPN</i>	<i>PPN</i>	<i>Valid</i>
08	13	1
09	17	1
0A	09	1
0B	–	0
0C	–	0
0D	2D	1
0E	11	1
0F	0D	1

Simple Memory System TLB

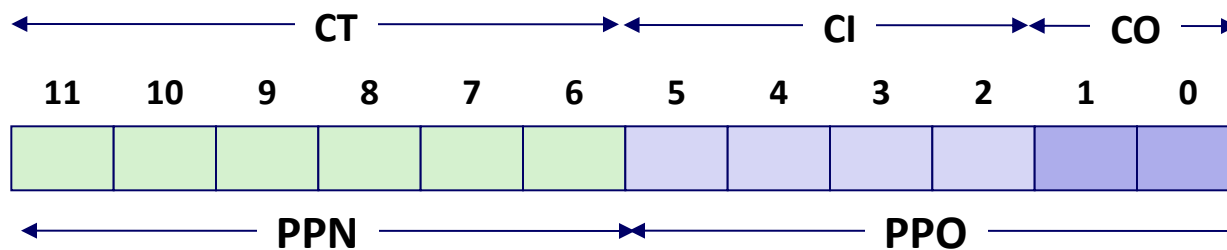
- 16 entries
- 4-way associative



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

Simple Memory System Cache

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



<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
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

<i>Idx</i>	<i>Tag</i>	<i>Valid</i>	<i>B0</i>	<i>B1</i>	<i>B2</i>	<i>B3</i>
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	–	–	–	–

Current state of caches/tables

page size = 64 bytes

TLB

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

VPN	PPN	Valid	VPN	PPN	Valid
00	28	1	08	13	1
01	–	0	09	17	1
02	33	1	0A	09	1
03	02	1	0B	–	0
04	–	0	0C	–	0
05	16	1	0D	2D	1
06	–	0	0E	11	1
07	–	0	0F	0D	1

Page table

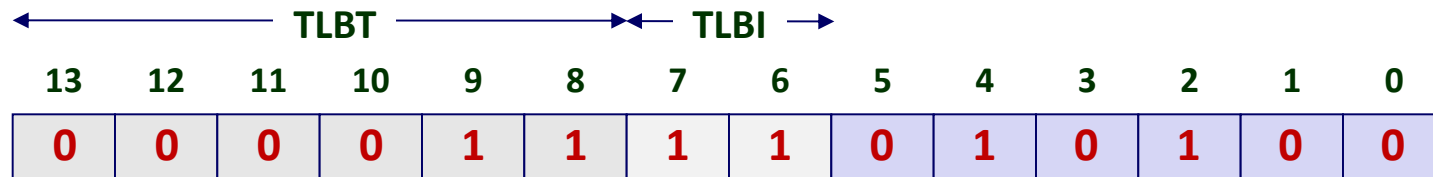
Cache

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	–	–	–	–

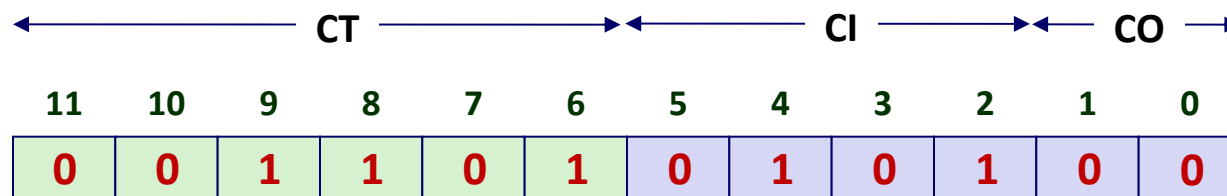
Address Translation Example #1

Virtual Address: 0x03D4



VPN 0x0F TLBI 3 TLBT 0x03 TLB Hit? Y Page Fault? N PPN: 0x0D

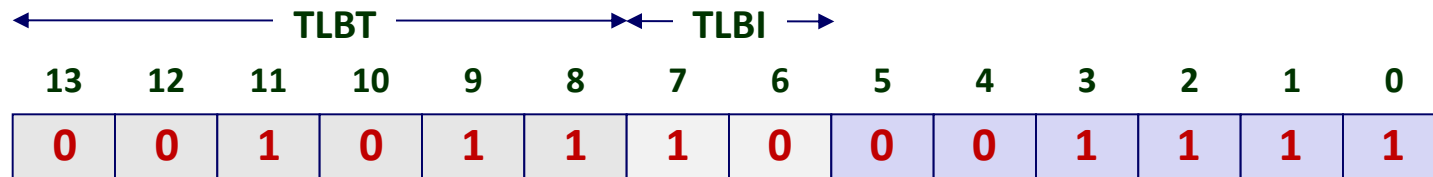
Physical Address



CO 0 CI 0x5 CT 0x0D Hit? Y Byte: 0x36

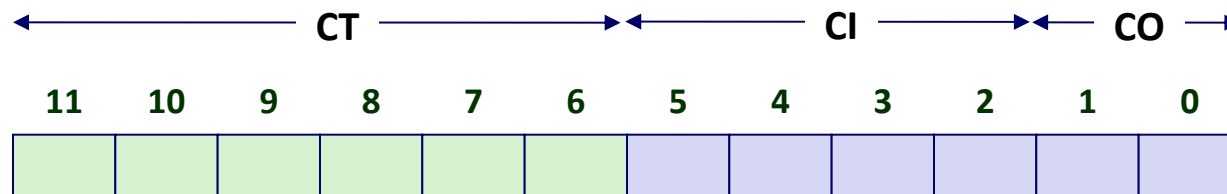
Address Translation Example #2

Virtual Address: 0x0B8F



VPN 0x2E TLBI 2 TLBT 0x0B TLB Hit? N Page Fault? ? PPN: TBD

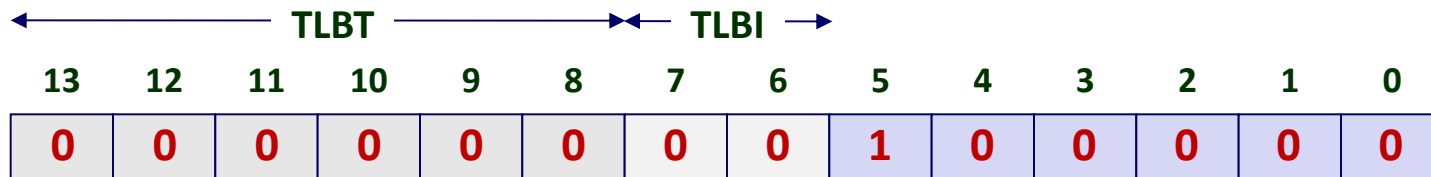
Physical Address



CO ____ CI ____ CT ____ Hit? ____ Byte: ____

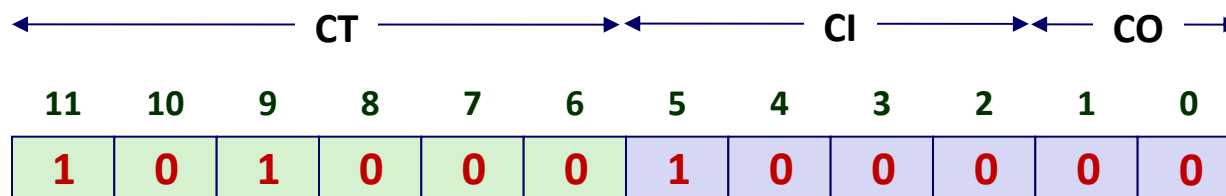
Address Translation Example #3

Virtual Address: 0x0020



VPN 0x00 TLBI 0 TLBT 0x00 TLB Hit? N Page Fault? N PPN: 0x28

Physical Address



CO 0 CI 0x8 CT 0x28 Hit? N Byte: Mem

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 sharing
- Simplifies protection by providing a convenient interpositioning point to check permissions

Memory System Summary

■ L1/L2 Memory Cache

- Purely a speed-up technique
- Behavior invisible to application programmer and (mostly) OS
- Implemented totally in hardware

■ Virtual Memory

- Supports many OS-related functions
 - Process creation, task switching, protection
- Software
 - Allocates/shares physical memory among processes
 - Maintains high-level tables tracking memory type, source, sharing
 - Handles exceptions, fills in hardware-defined mapping tables
- Hardware
 - Translates virtual addresses via mapping tables, enforcing permissions
 - Accelerates mapping via translation cache (TLB)

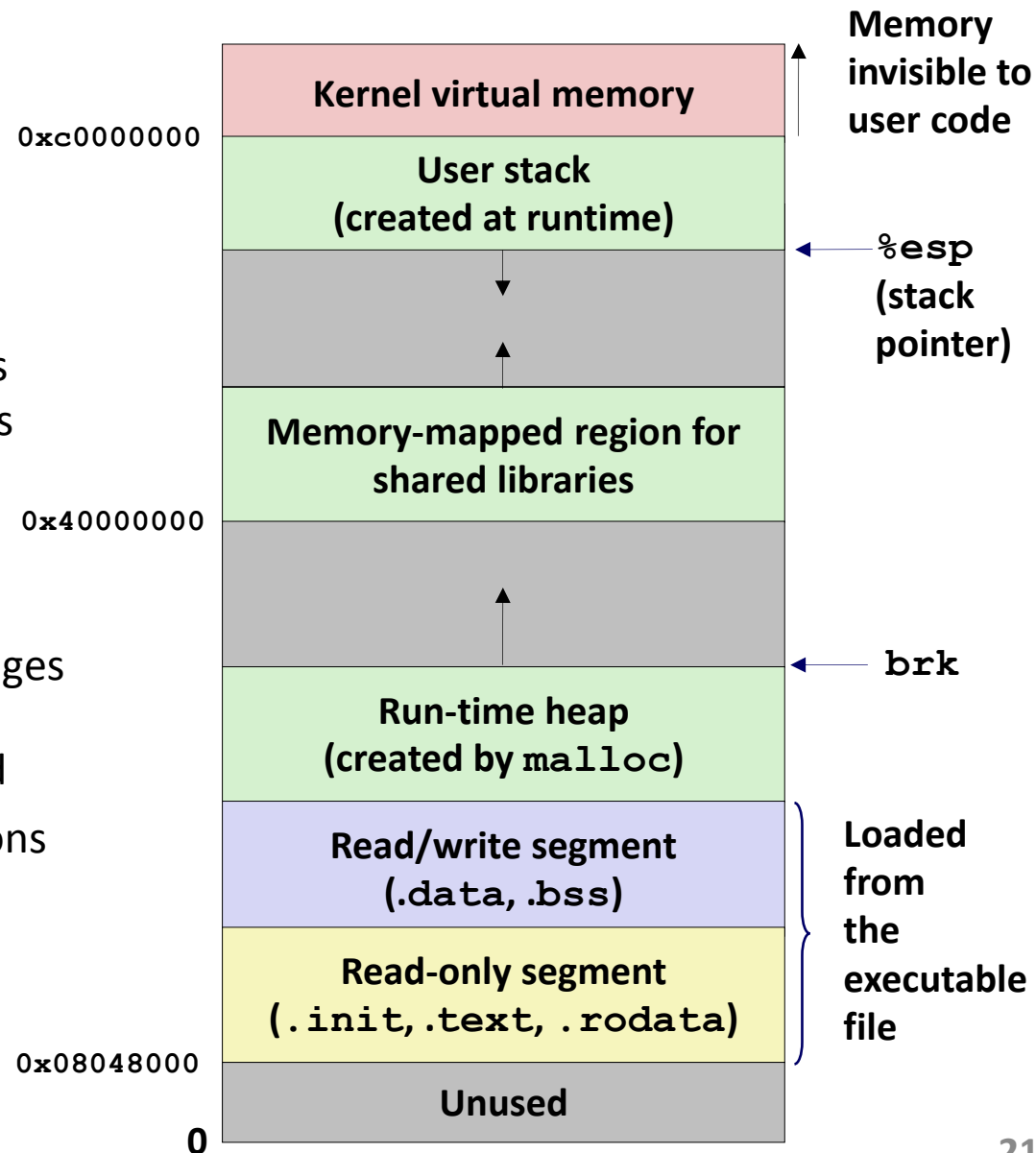
Simplifying Linking and Loading

■ Linking

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

■ 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



Servicing a Page Fault

■ (1) Processor signals disk controller

- Read block of length P starting at disk address X
- Store starting at memory address Y

■ (2) Read occurs

- Direct Memory Access (DMA)
- Under control of I/O controller

■ (3) Controller signals completion

- Interrupts processor
- OS resumes suspended process

