Virtual Memory: Systems II

CSCI 237: Computer Organization
26th Lecture, Apr 23, 2025

Jeannie Albrecht

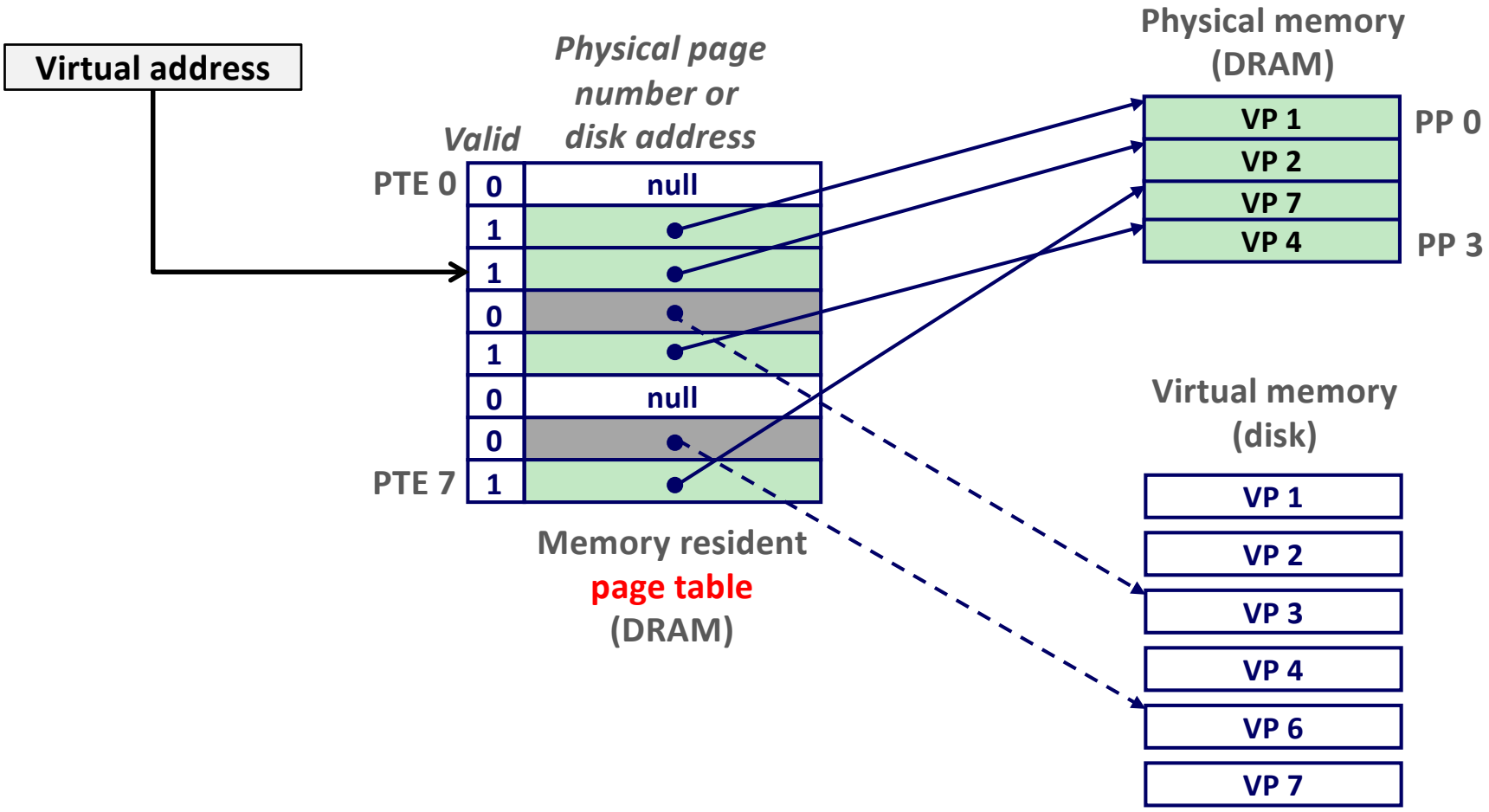
Administrative Details

- Lab 5 – Cache simulator, due next week
- Glow HW coming soon, but will be due next week
- Advising info session at 2:30 on Friday (in Wege)
  - Submit planning sheet beforehand
  - Parallel advising at info session
  - Can declare major and remove holds

Last time

- Introduction to virtual memory
- VM as a tool for caching
- VM as a tool for memory protection

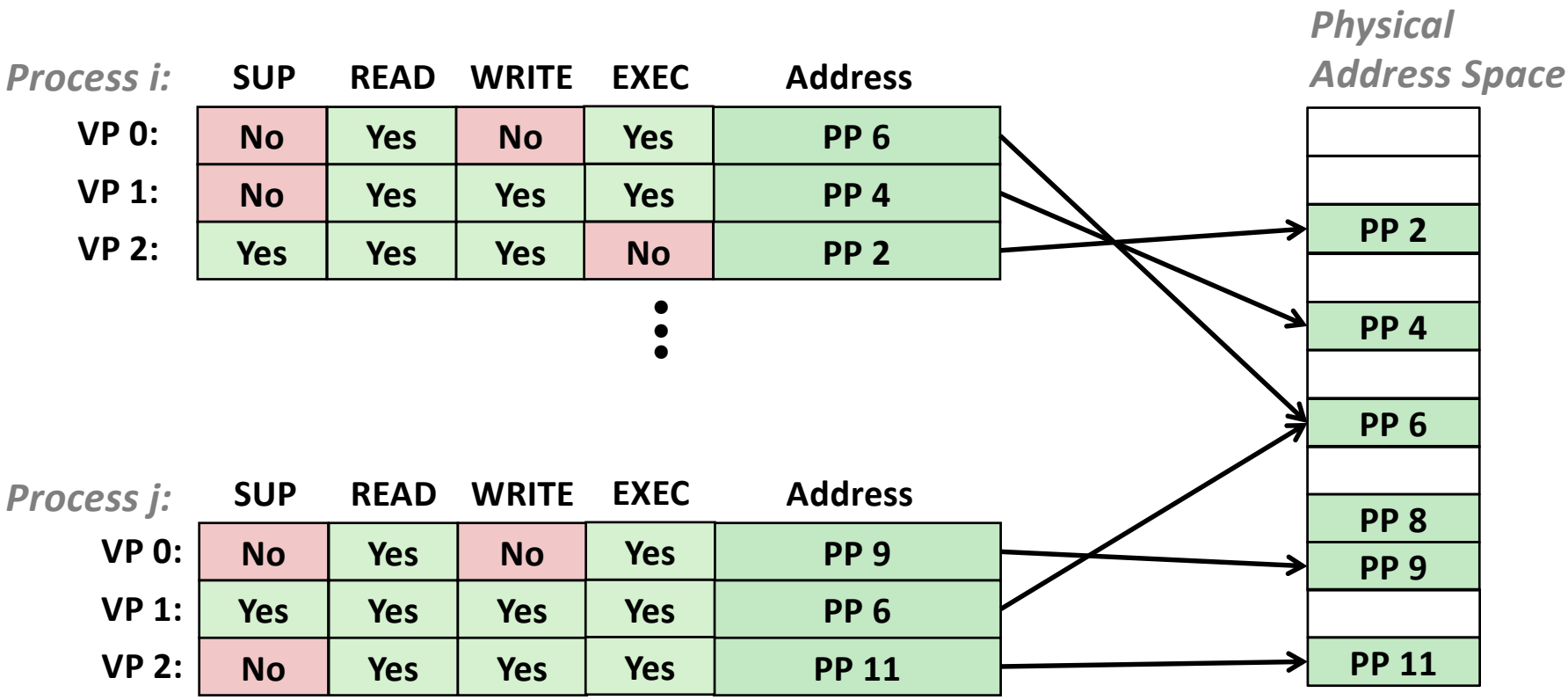
Recap: Virtual Memory & Physical Memory



- A *page table* contains page table entries (PTEs) that map virtual pages to physical pages.

Recap: VM as a Tool for Memory Protection

- Extend PTEs with permission bits (similar to UNIX file permissions!)
- MMU checks these bits on each access



Today

- Address translation (Ch 9.6)
- “Simple” memory system example

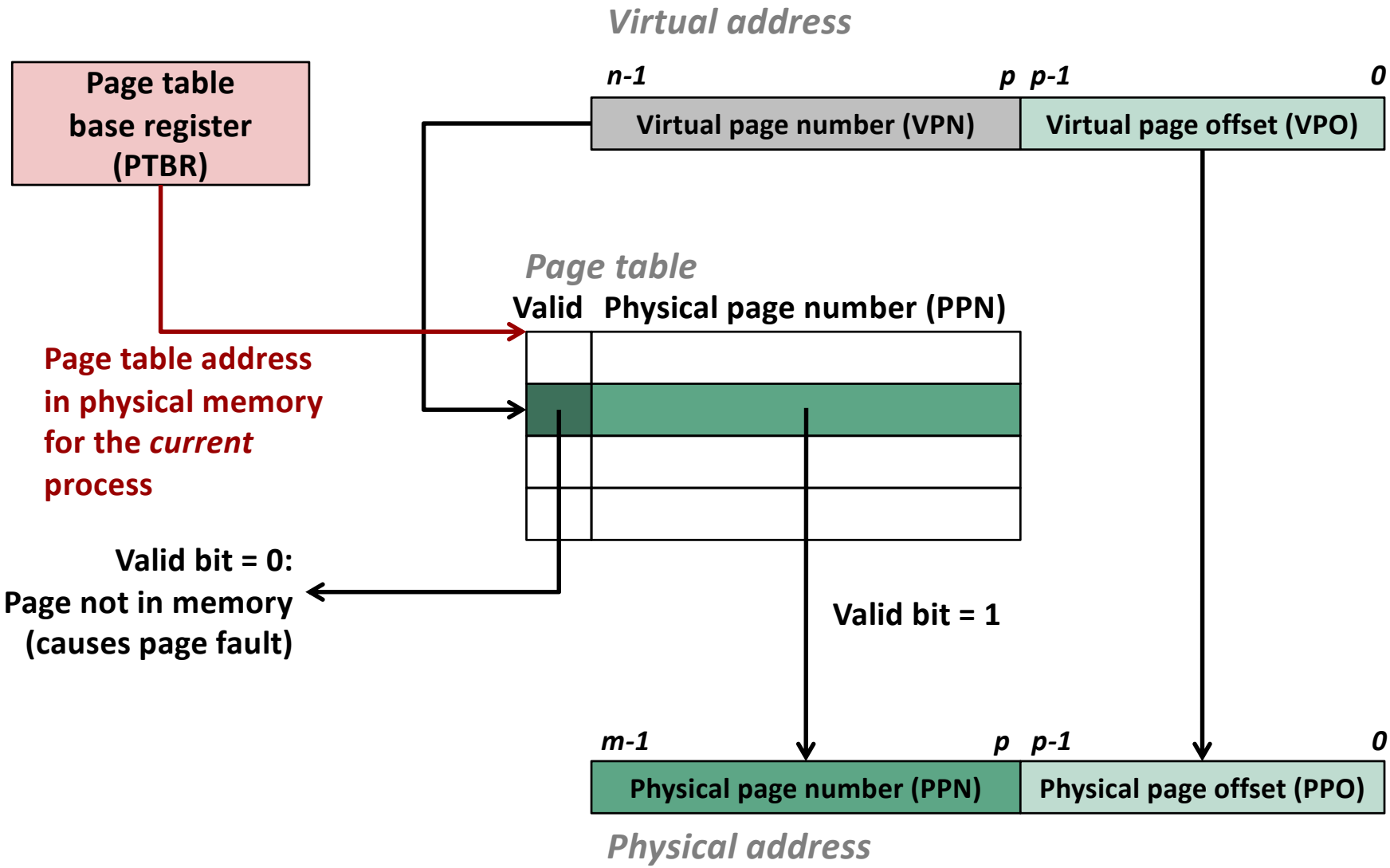
VM Address Translation

- Virtual Address Space (N pages)
  - *V = {0, 1, …, N–1}*
- Physical Address Space (M pages)
  - *P = {0, 1, …, M–1}*
- Address Translation (map virtual addr to physical addr)
  - ***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 an invalid addr or page is currently stored on disk

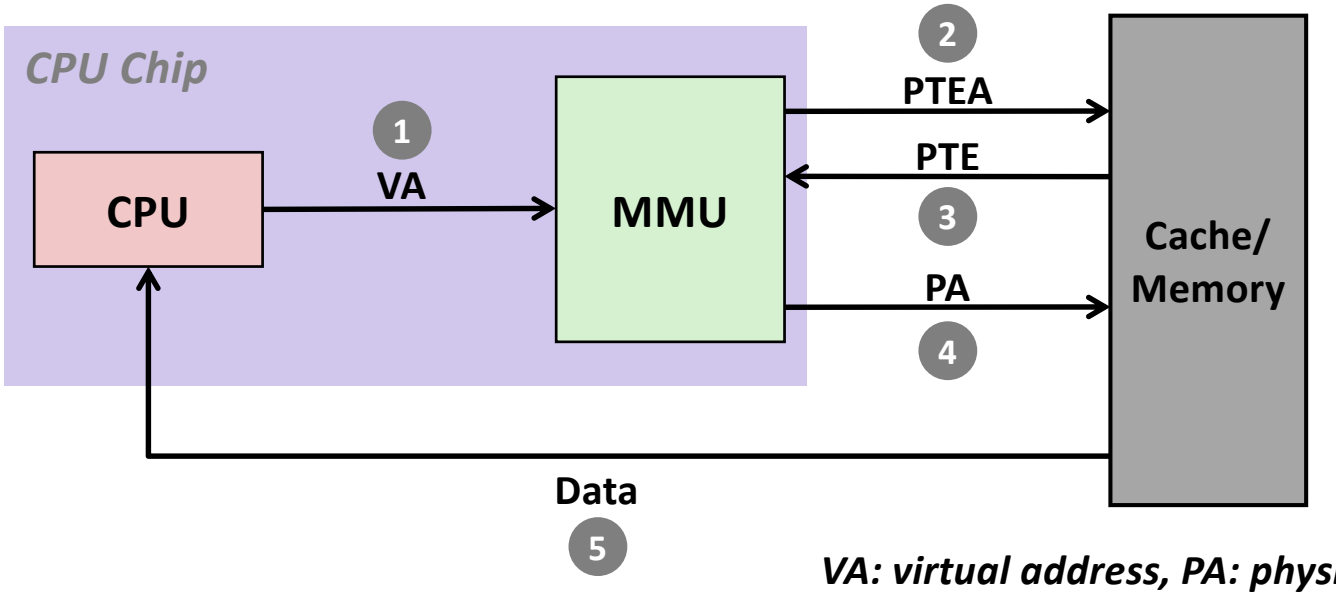
Summary of Address Translation Symbols

- 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)
  - **VPO**: Virtual page offset
  - **VPN**: Virtual page number
  - **TLBI**: TLB (Translation Lookaside Buffer) index
  - **TLBT**: TLB tag
- Components of the physical address (PA)
  - **PPO**: Physical page offset (same as VPO since page sizes are same)
  - **PPN:** Physical page number

Address Translation With a Page Table



Address Translation: Page Hit



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

1) Processor sends virtual address to MMU

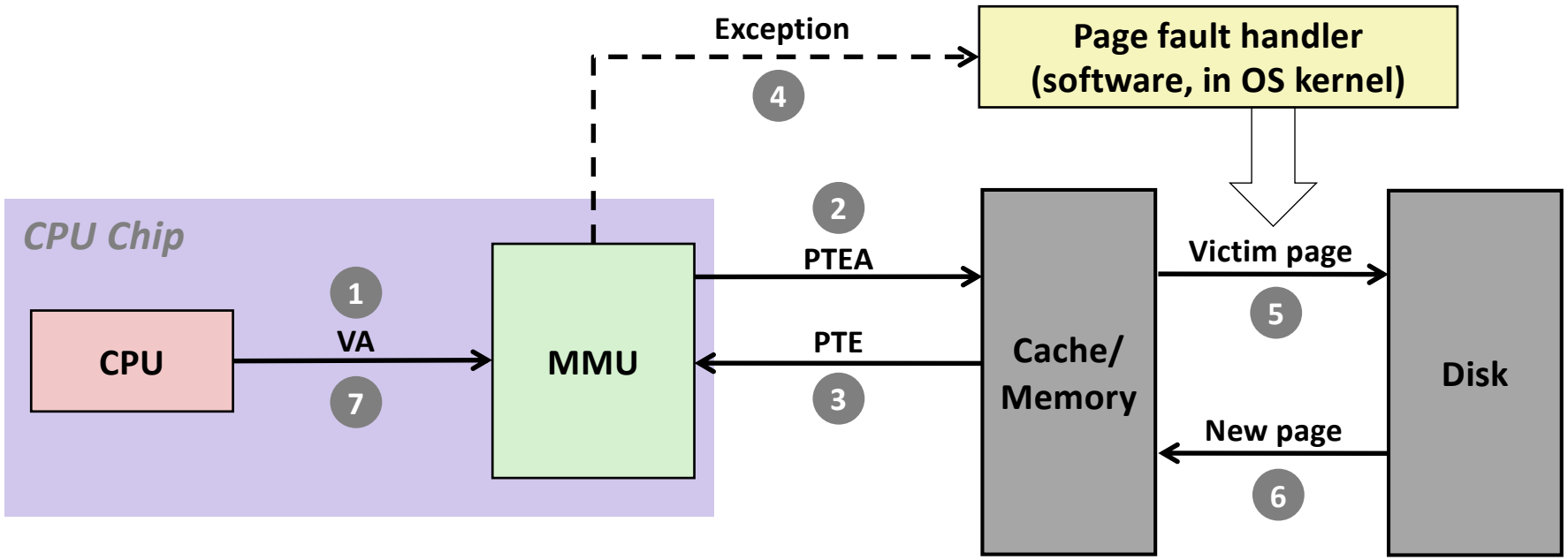
2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

5) Cache/memory sends data word to processor

Handled entirely by hardware (so very fast)!

Address Translation: Page Fault



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

1) Processor sends virtual address to MMU

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

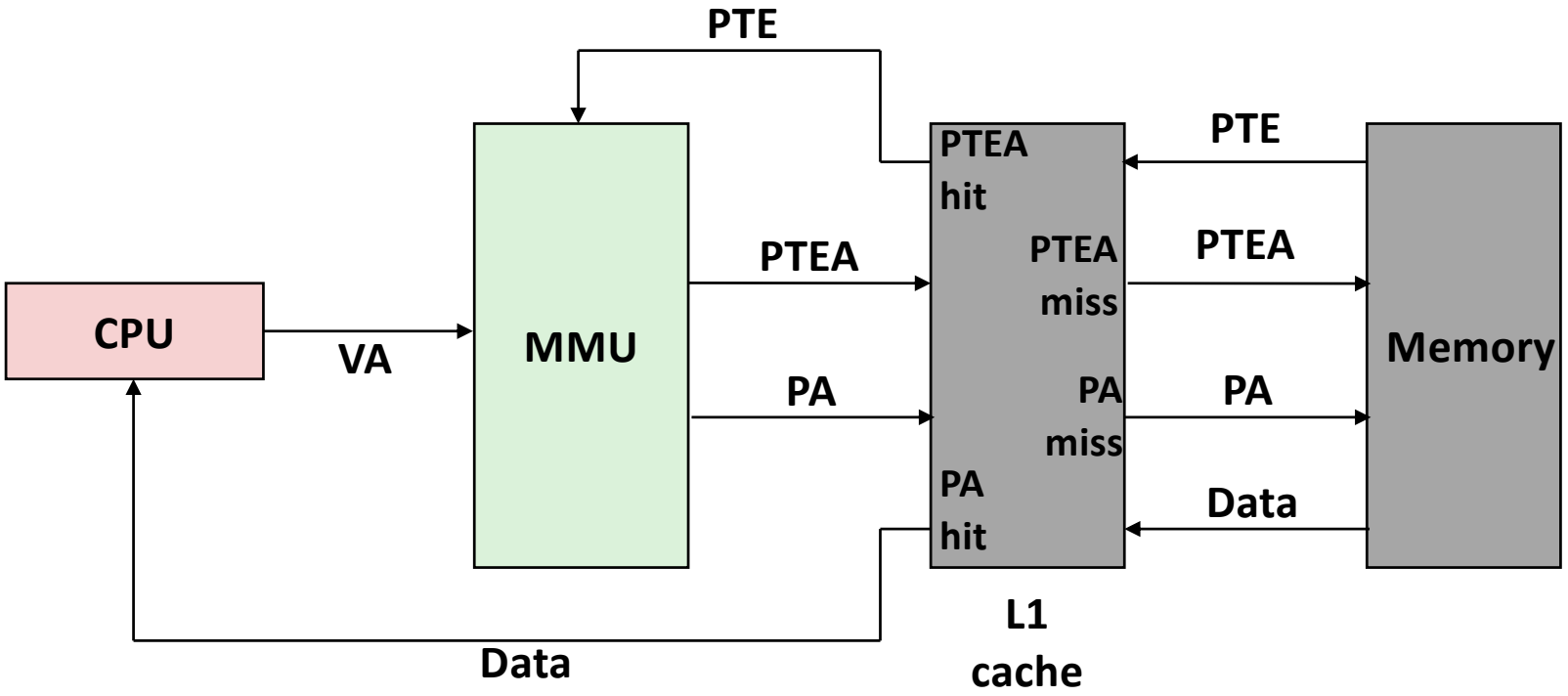
4) Valid bit is zero, so MMU triggers page fault exception

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

Integrating VM and Cache



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

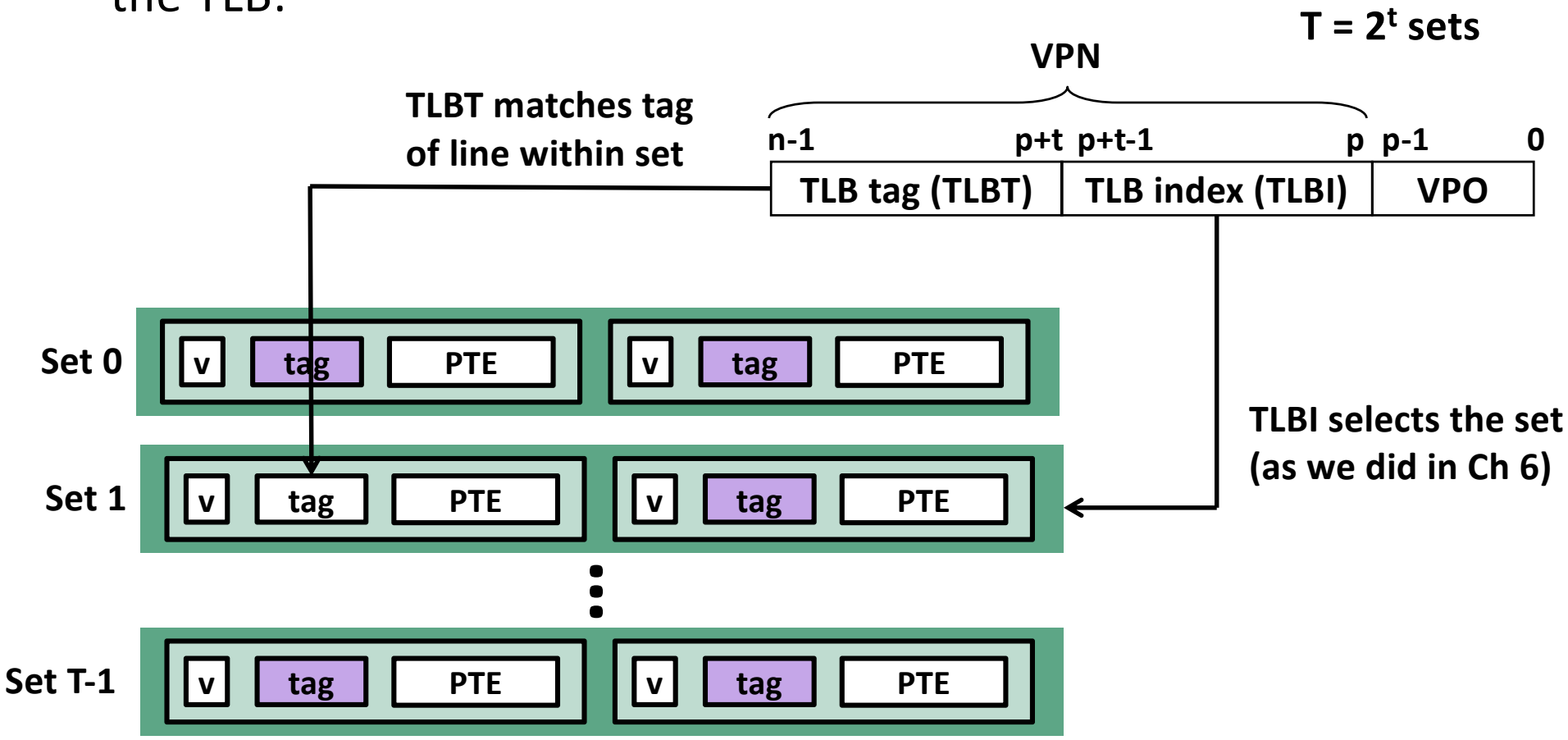
Cache can hold page table entries, like any other data word! Note that address translation usually happens before cache lookup. Cache uses physical addresses.

Speeding up Translation with a TLB

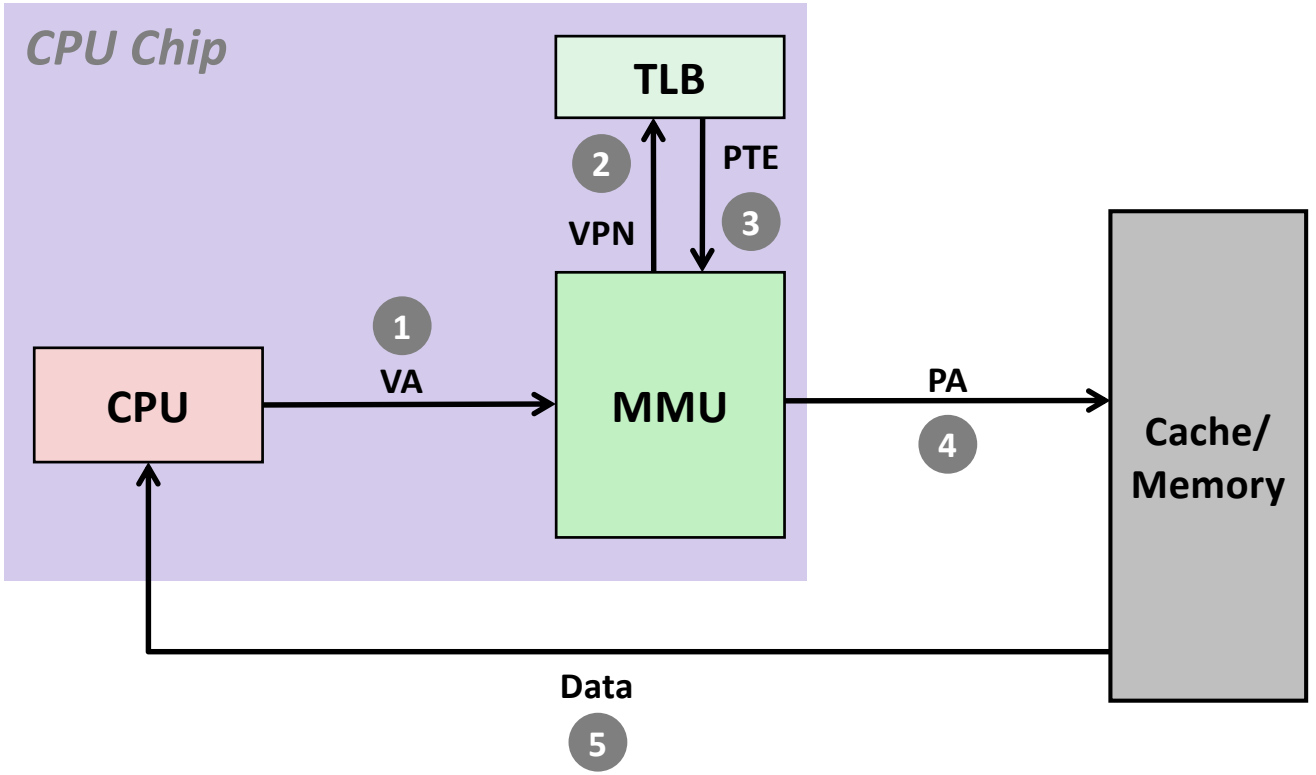
- Page table entries (PTEs) are cached in L1/L2/L3 like any other memory word
  - So PTEs may be evicted by other data references
  - Even PTE hit still requires a small L1/L2/L3 delay
- Solution: *Translation Lookaside Buffer* (TLB)
  - Small set-associative **hardware** cache *in* MMU (part of CPU chip)
  - Maps virtual page numbers to physical page numbers
  - Contains complete page table entries for small number of pages
  - Each line holds a block consisting of single PTE

Accessing the TLB

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



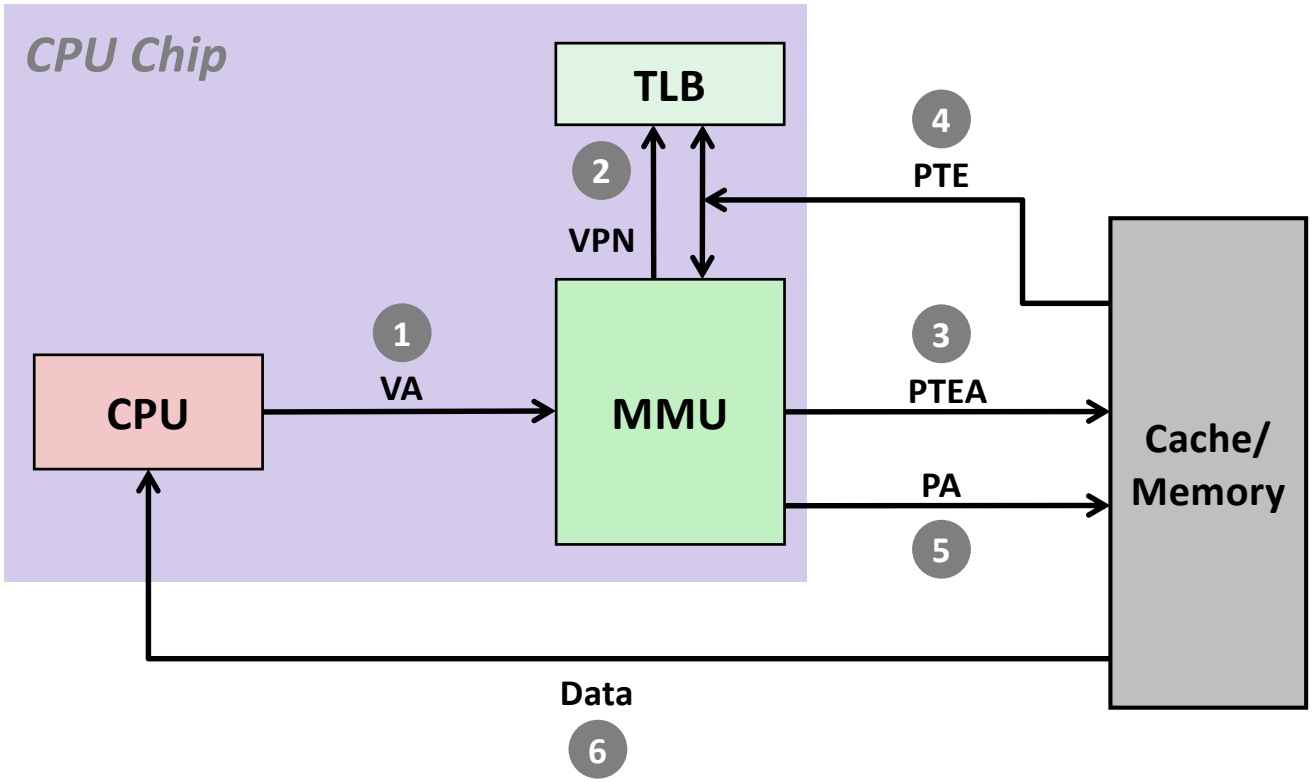
TLB Hit



A TLB hit eliminates a memory access.

All steps in address *translation* happen inside MMU and are fast.

TLB Miss

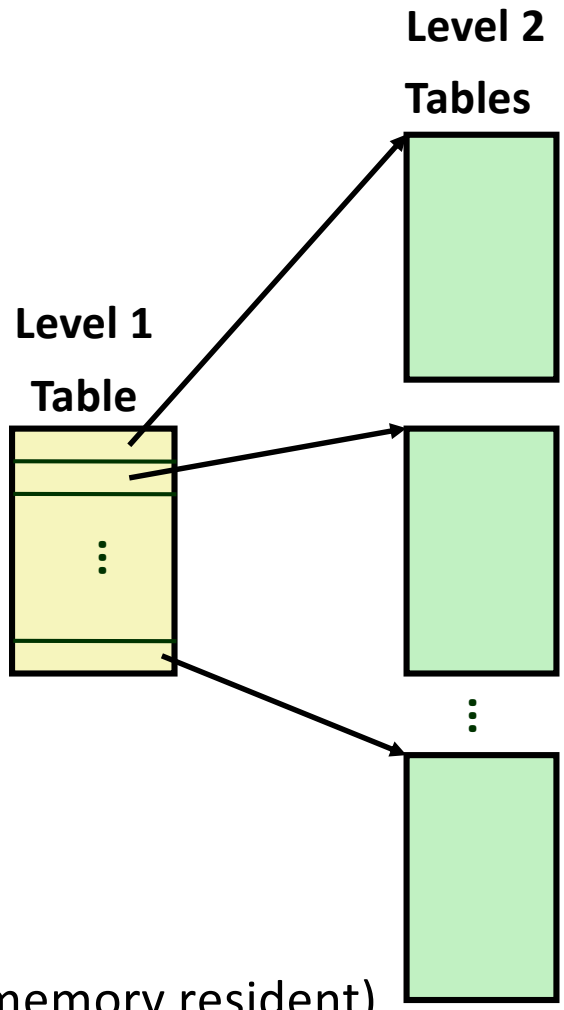


A TLB miss incurs an additional memory access (the PTE).

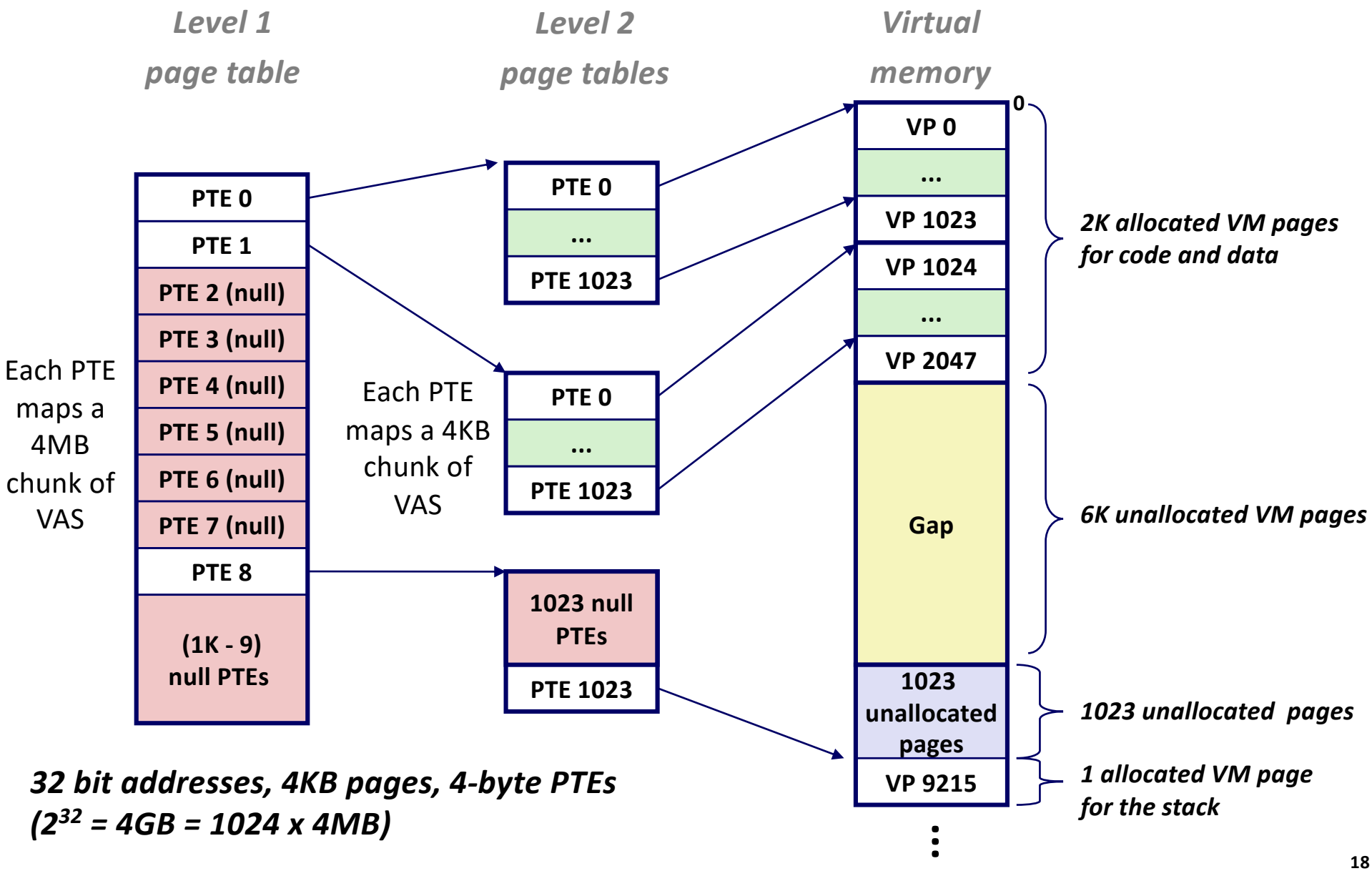
Fortunately, TLB misses are rare.

Moving on…
Multi-Level Page Tables (tricky concept!)

- Suppose:
  - 4KB (2^{12}) page size, 48-bit address space, 8-byte PTE
- Problem:
  - Would need a 512 GB page table!
    - 2^{48} addr / (2^{12} addr/page) = # entries in page table
    - 2^3 bytes per entry
    - ($2^{48} * 2^{-12}$) entries * 2^3 (bytes/entry)
      = 2^{39} bytes in every page table (BIG!)
- Common solution: Multi-level page table
  - No need to waste space for unallocated pages!
  - Indirection to the rescue!
- Example: 2-level page table
  - Level 1 table: each PTE points to a page table (always memory resident)
  - Level 2 table: each PTE points to a page (paged in and out like any other data)

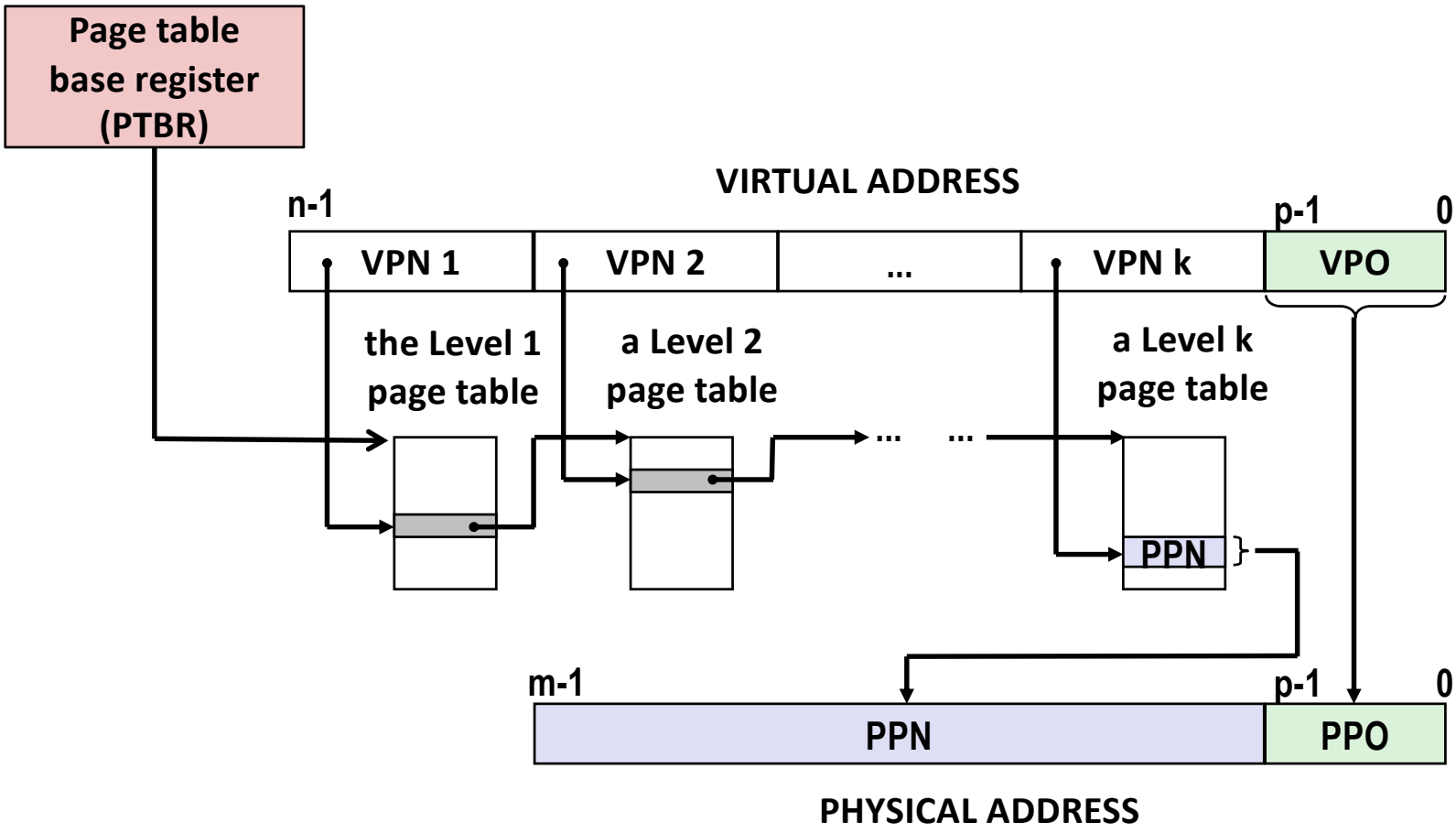


A Two-Level Page Table Hierarchy



32 bit addresses, 4KB pages, 4-byte PTEs
(2^{32} = 4GB = 1024 x 4MB)

Translating with a k-level Page Table



Address Translation Summary (Ch 9.6)

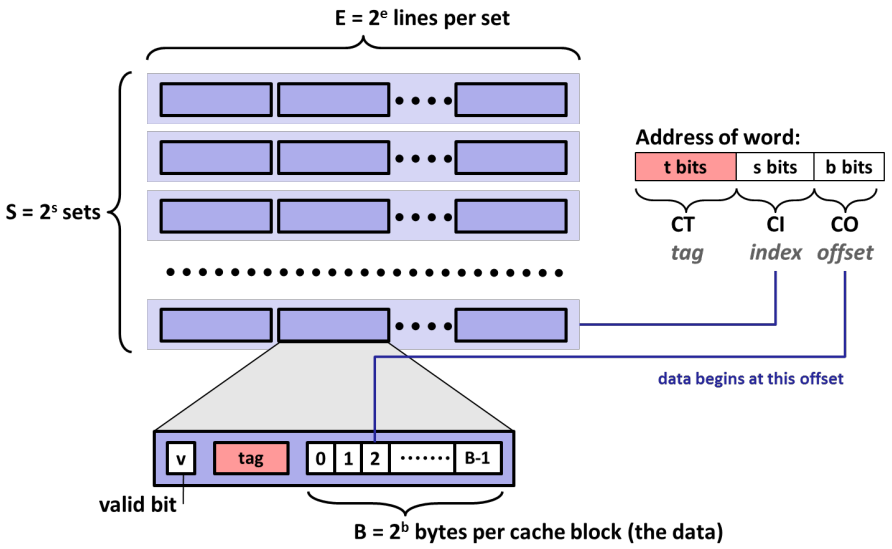
- Programmer's view of virtual memory
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes

- System (OS) 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 inter-positioning point to check permissions

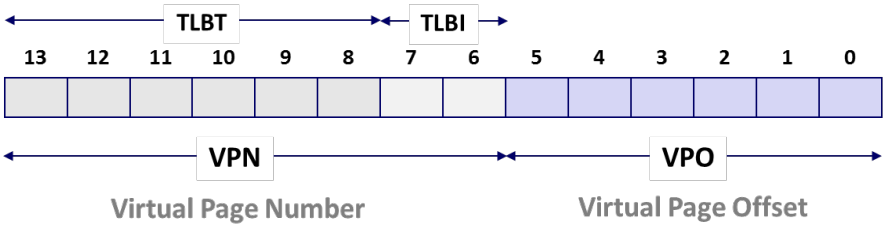
Let’s put it all together in an example!

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)

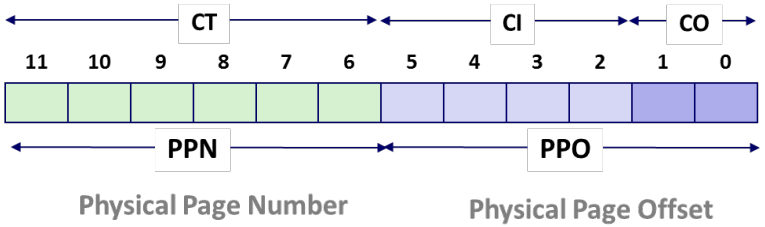


- Components of the *virtual address* (VA)
  - **TLBI**: TLB index
  - **TLBT**: TLB tag
  - **VPO**: Virtual page offset
  - **VPN**: Virtual page number



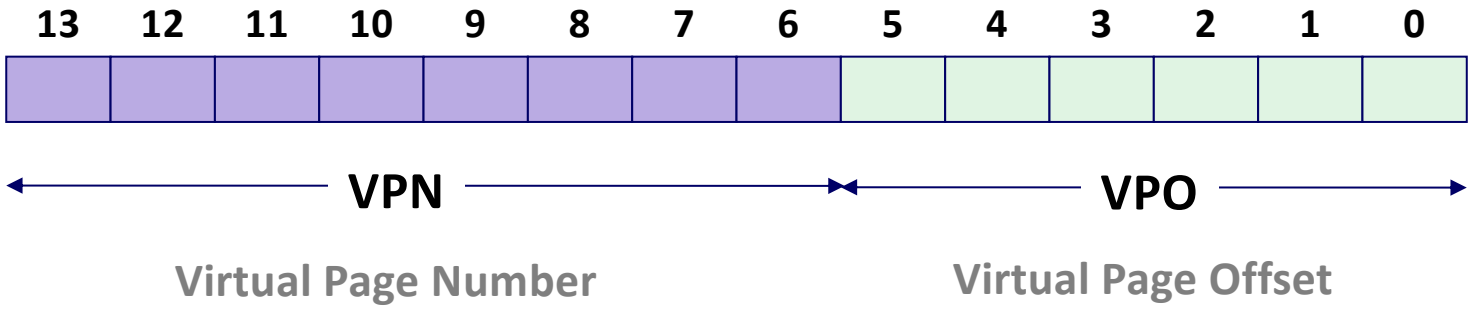
- Components of the *physical address* (PA)
  - **PPO**: Physical page offset (same as VPO)
  - **PPN:** Physical page number
  - **CO**: Byte offset within cache line
  - **CI:** Cache index
  - **CT**: Cache tag

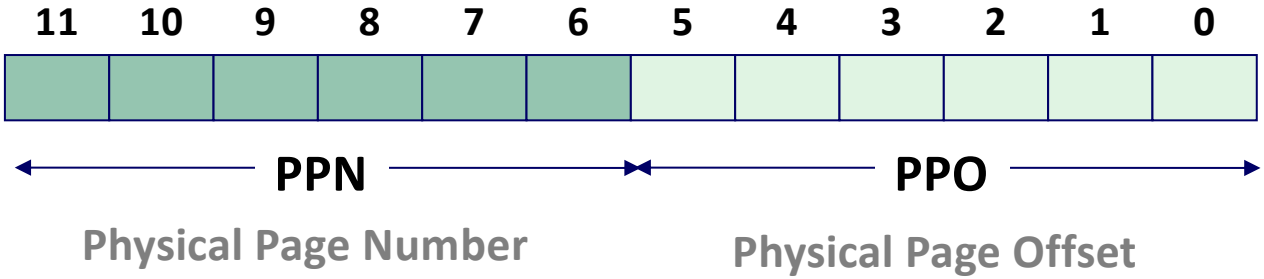
(bits per field for our simple example)



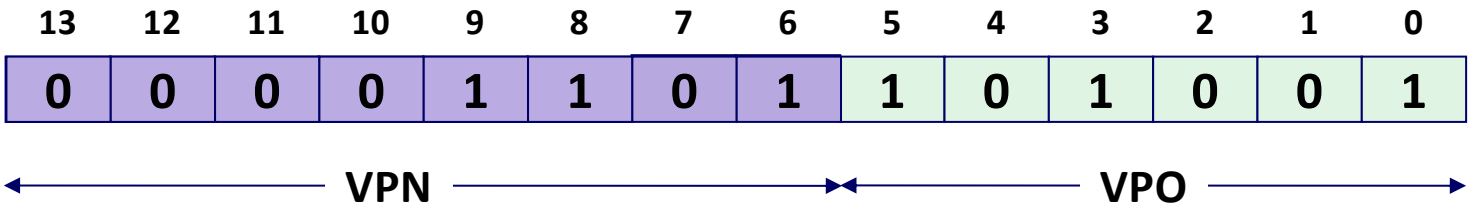
Simple Memory System Example

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





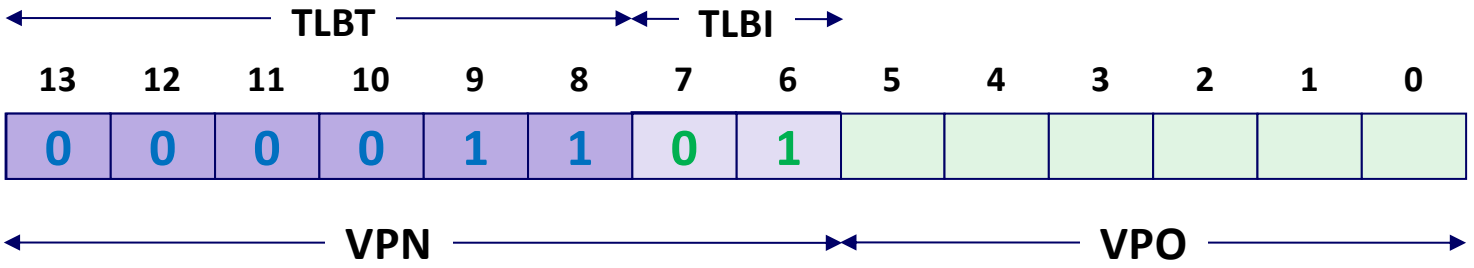
Simple Memory System



VA = 0x0369 = 0b00 0011 0110 1001

Simple Memory System TLB (VPN➔PPN)

- 16 entries (4 sets)
- 4-way associative



VPN = 0b00001101 = 0x0D

Translation Lookaside Buffer (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

Simple Memory System Page Table

What happens on TLB miss?

If PTE not in TLB, need to get PTE from main memory

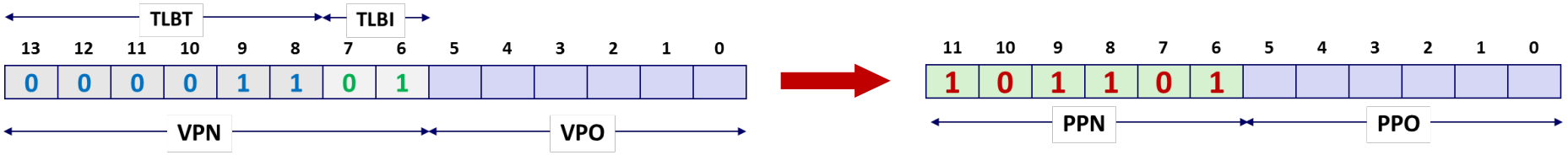
Only showing the first 16 entries (out of $2^8 = 256$)

Note: VPN not actually part of page table (shown for reference...it's just an index)

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

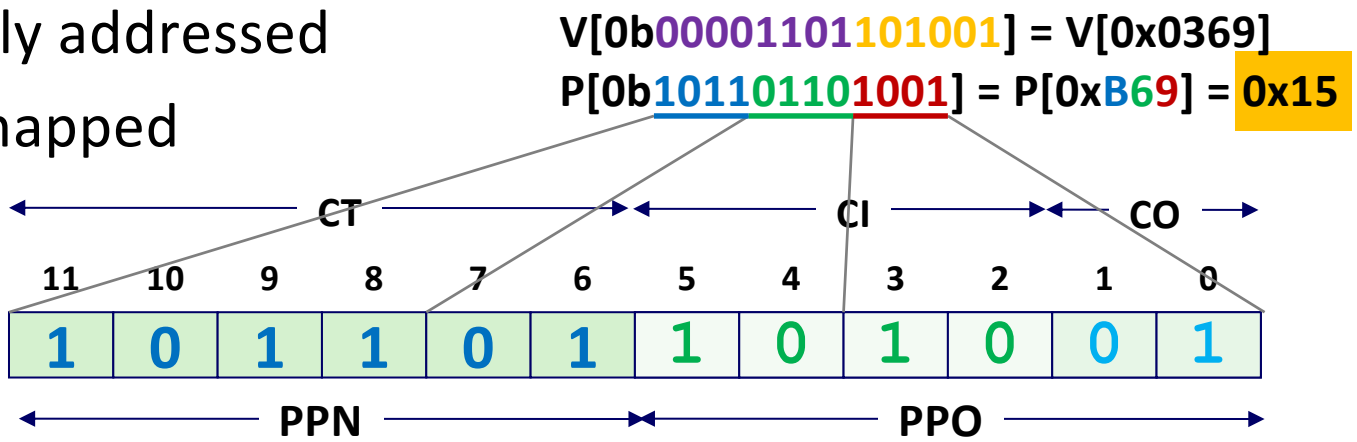
0x0D → 0x2D



Simple Memory System Cache

(*after* finding PPN in TLB or PT in mem)

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



Tag = 10 1101=2D; Set (Idx) = 1010 = A

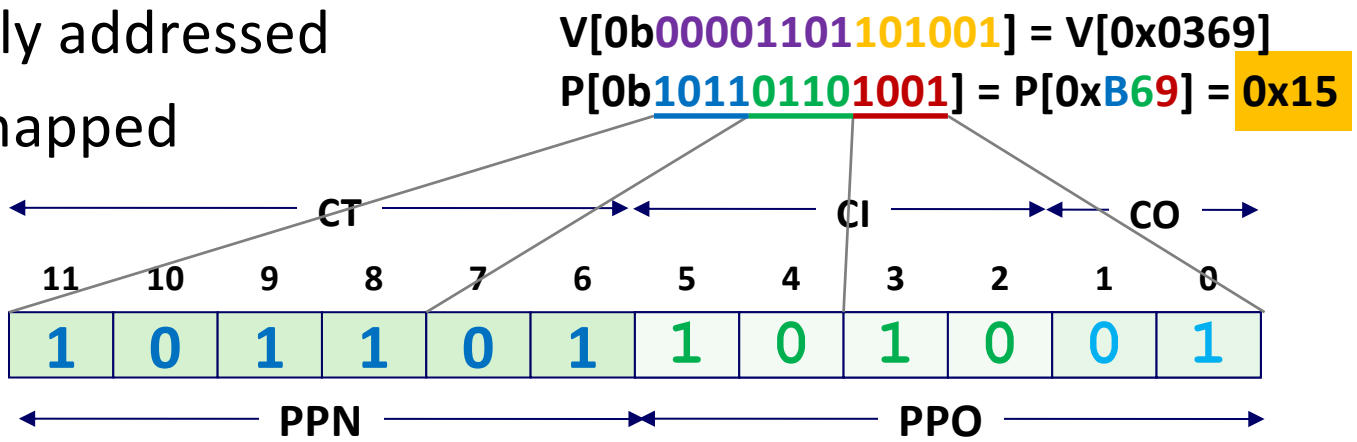
Idx	*Tag*	*Valid*	*B0*	*B1*	*B2*	*B3*
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

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	–	–	–	–

Simple Memory System Cache

(*after* finding PPN in TLB or PT in mem)

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



Tag = 10 1101=2D; Set (Idx) = 1010 = A

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	–	–	–	–