VIRTUAL MEMORY I

CAS CS 210

9.1 - 9.6.1

Virtual Memory (Previous Lectures)

- **Programs refer to virtual memory addresses**
  - `movl (%ecx),%eax`
  - Conceptually very large array of bytes
  - Each byte has its own address
  - Actually implemented with hierarchy of different memory types
  - System provides address space private to particular "process"
- **Allocation: Compiler and run-time system**
  - Where different program objects should be stored
  - All allocation within single virtual address space
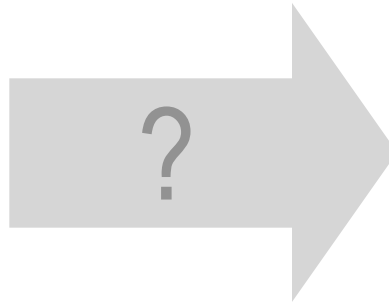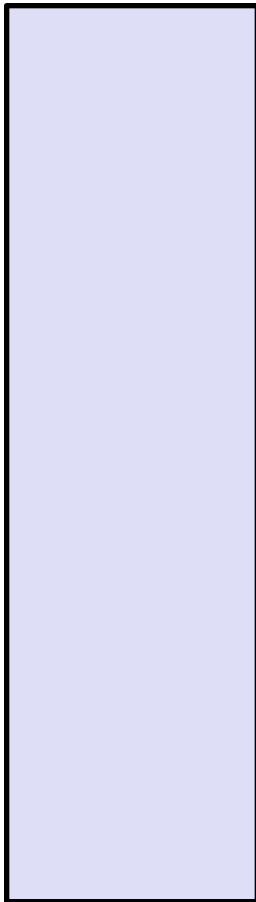- ***But why virtual memory?***
- ***Why not physical memory?***

00⋯⋯0

FF⋯⋯F
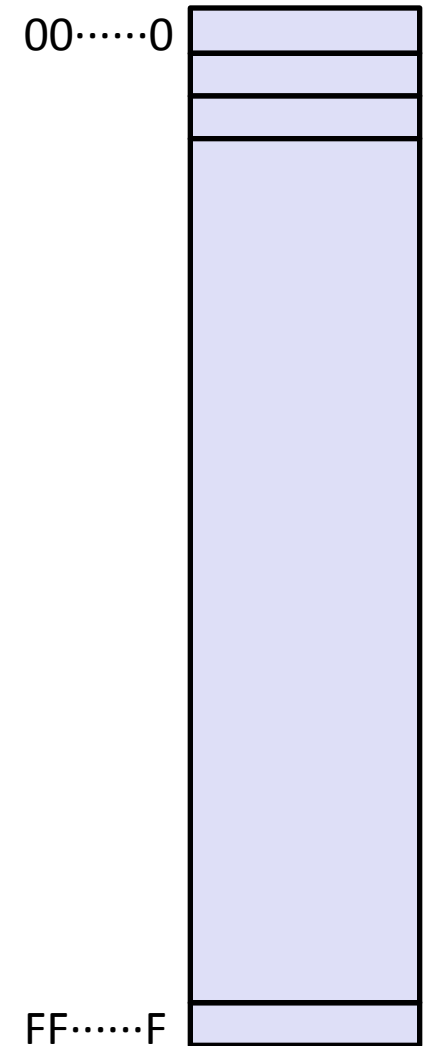
Problem 1: How Does Everything Fit?

64-bit addresses:
16 Exabyte

Physical main memory:
Few Gigabytes

?

And there are many processes ....

Problem 2: Memory Management

Process 1
Process 2
Process 3
…
Process n

X

stack
heap
.text
.data
…

What goes where?

Physical main memory

Problem 3: How To Protect

Physical main memory
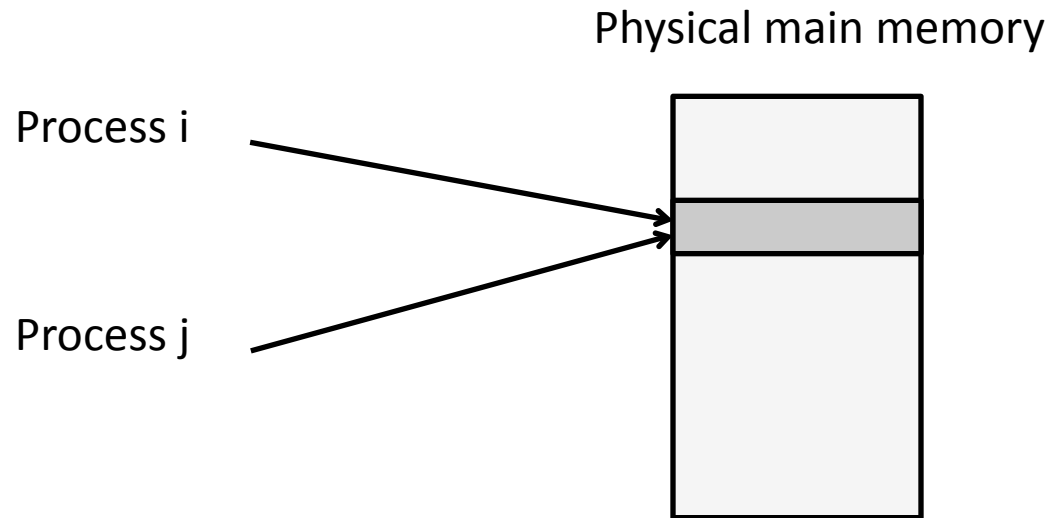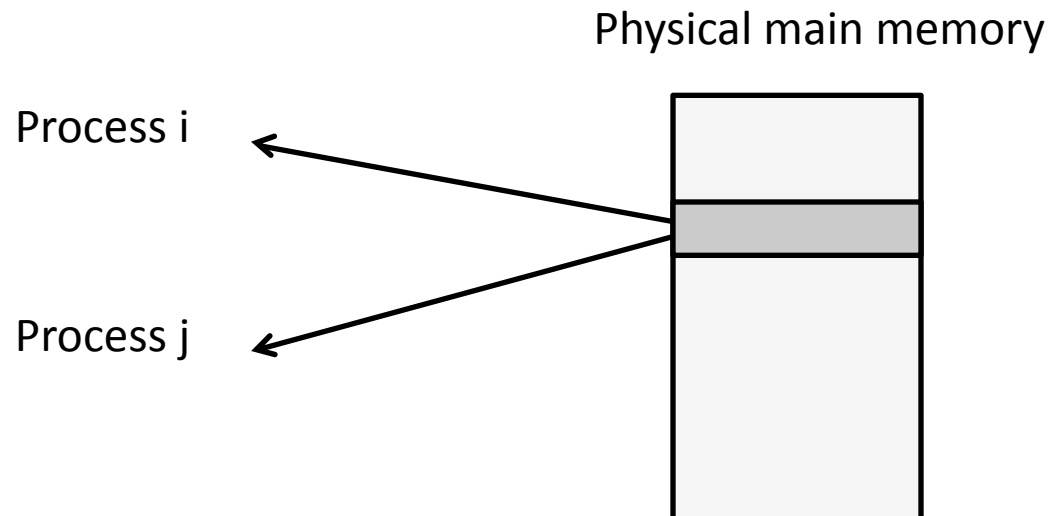
Process i

Process j

Solution: Level Of Indirection

Virtual memory

Process 1

Physical memory

mapping

Virtual memory
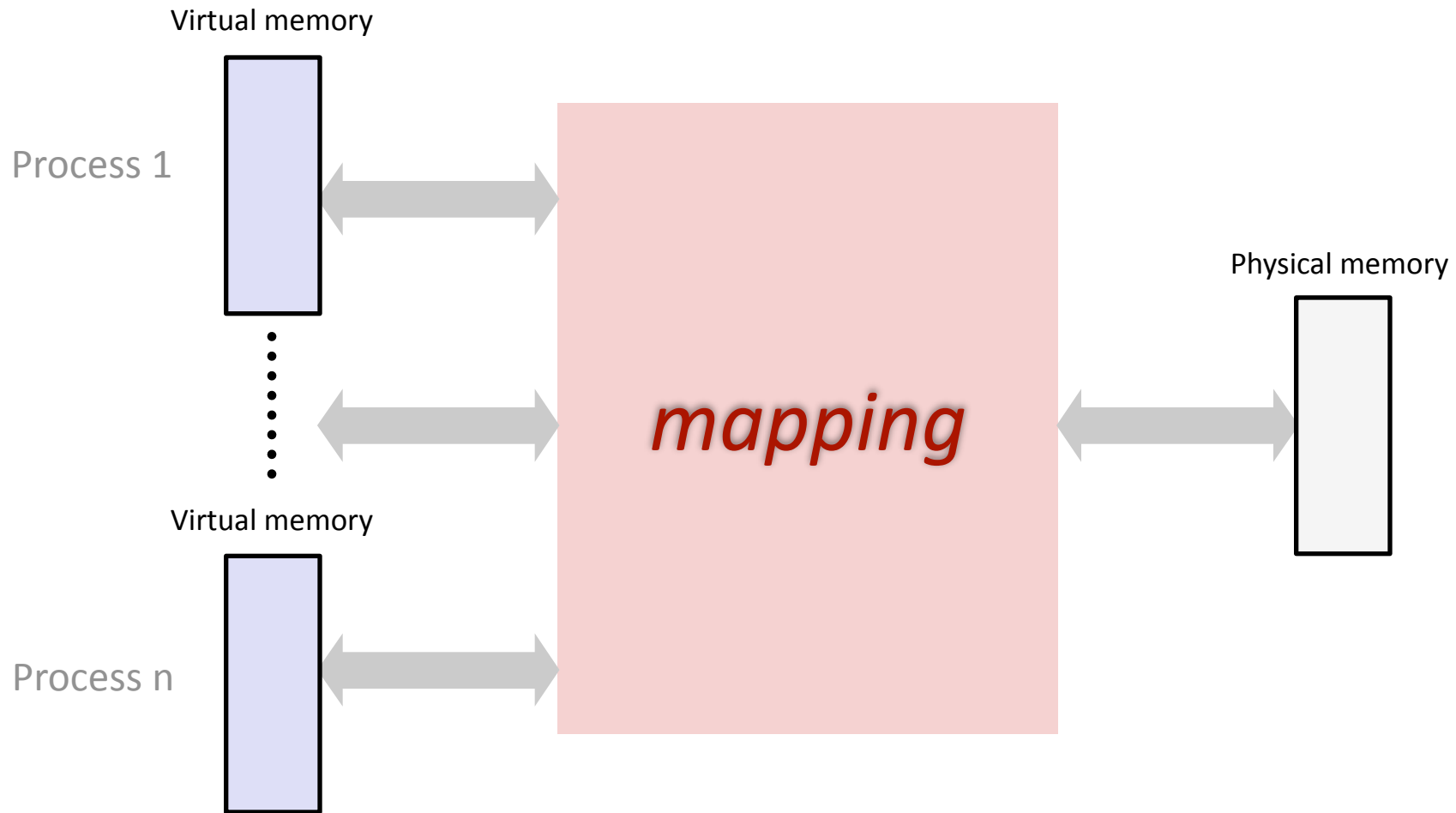
Process n

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

Address Spaces

- **Linear address space:** Ordered set of contiguous non-negative integer addresses:

$$\{0, 1, 2, 3 \dots \}$$

- **Virtual address space:** Set of $N = 2^n$ virtual addresses

$$\{0, 1, 2, 3, \dots, N-1\}$$

- **Physical address space:** Set of $M = 2^m$ physical addresses

$$\{0, 1, 2, 3, \dots, M-1\}$$

- **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

A System Using Physical Addressing

Main memory

0:	
1:	
2:	
3:	
4:	
5:	
6:	
7:	
8:	

CPU

Physical address
(PA)

M-1:

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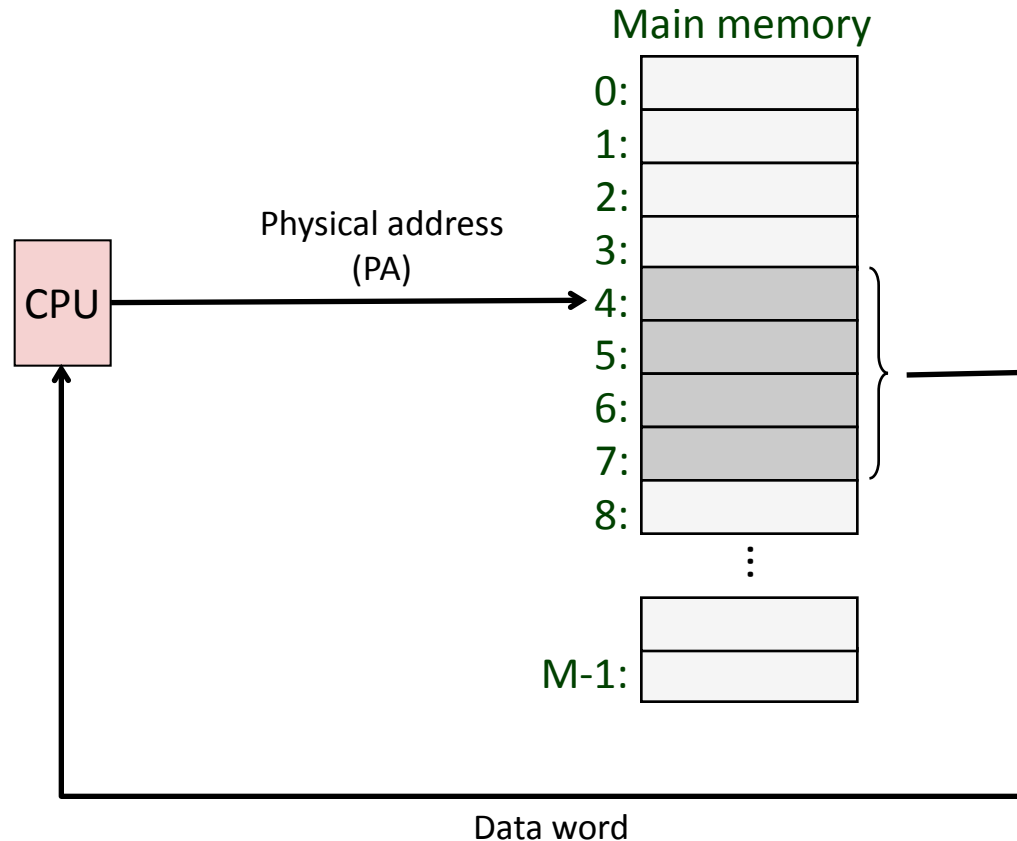
- **Used in "simple" systems like embedded microcontrollers in devices like cars, elevators, and digital picture frames**

A System Using Virtual Addressing

Main memory

CPU Chip

CPU → Virtual address (VA) → MMU → Physical address (PA) → Main memory

0:
1:
2:
3:
4:
5:
6:
7:
8:
...
M-1:

Data word
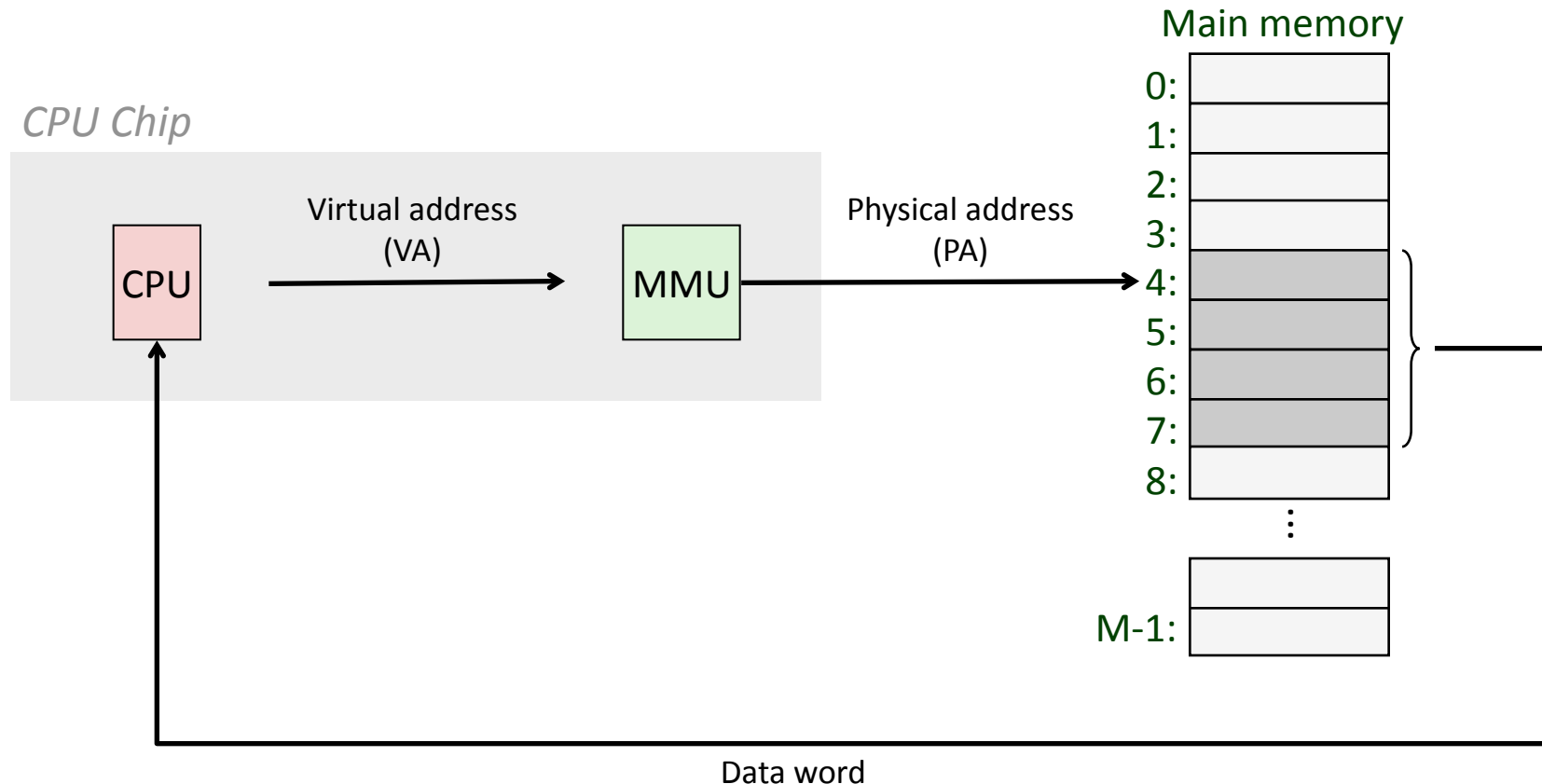
- **Used in all modern desktops, laptops, workstations**
- **One of the great ideas in computer science**
- *MMU checks the cache*

Why 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

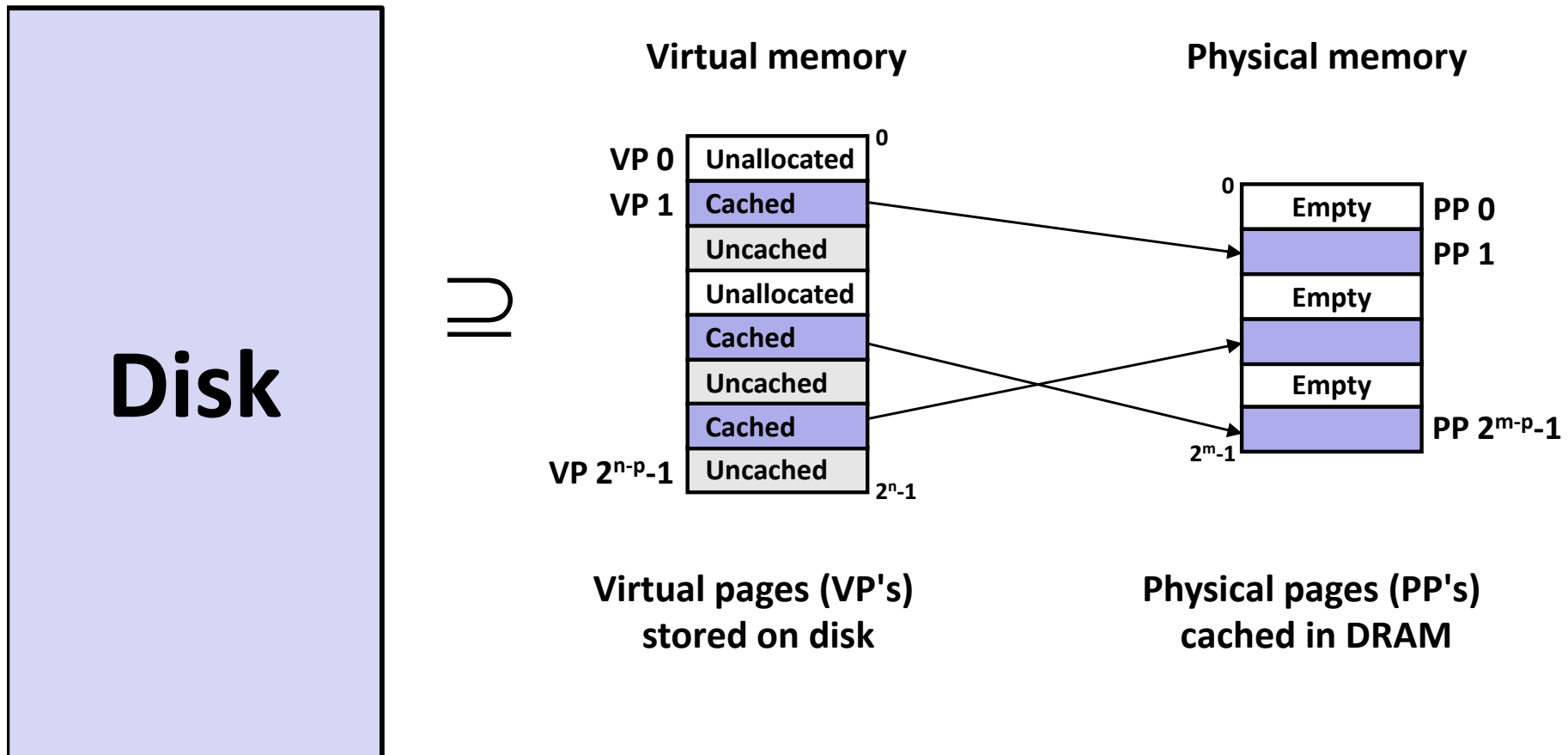- **Simplifies memory management for programmers**
  - Each process gets the same full, private linear address space

- **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

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**

Virtual memory　　　　　　**Physical memory**

VP 0	Unallocated	0
VP 1	Cached	
	Uncached	
	Unallocated	
	Cached	
	Uncached	
	Cached	
VP $2^{n-p}-1$	Uncached	2^n-1

Physical memory:
- 0
- Empty — PP 0
- (cached) — PP 1
- Empty
- (cached)
- Empty
- (cached) — PP $2^{m-p}-1$
- 2^m-1

Disk $\supseteq$

Virtual pages (VP's)
stored on disk

Physical pages (PP's)
cached in DRAM

Memory Hierarchy: Core 2 Duo

Not drawn to scale

L1/L2 cache: 64 B blocks

~4 MB

~4 GB

~500 GB

L1
I-cache

L2
unified
cache

Main
Memory

Disk

32 KB

L1
D-cache

CPU | Reg

Throughput: **16 B/cycle**   **8 B/cycle**   **2 B/cycle**   **1 B/30 cycles**
Latency: 3 cycles   14 cycles   100 cycles   millions

Miss penalty (latency): 30x

Miss penalty (latency): 10,000x
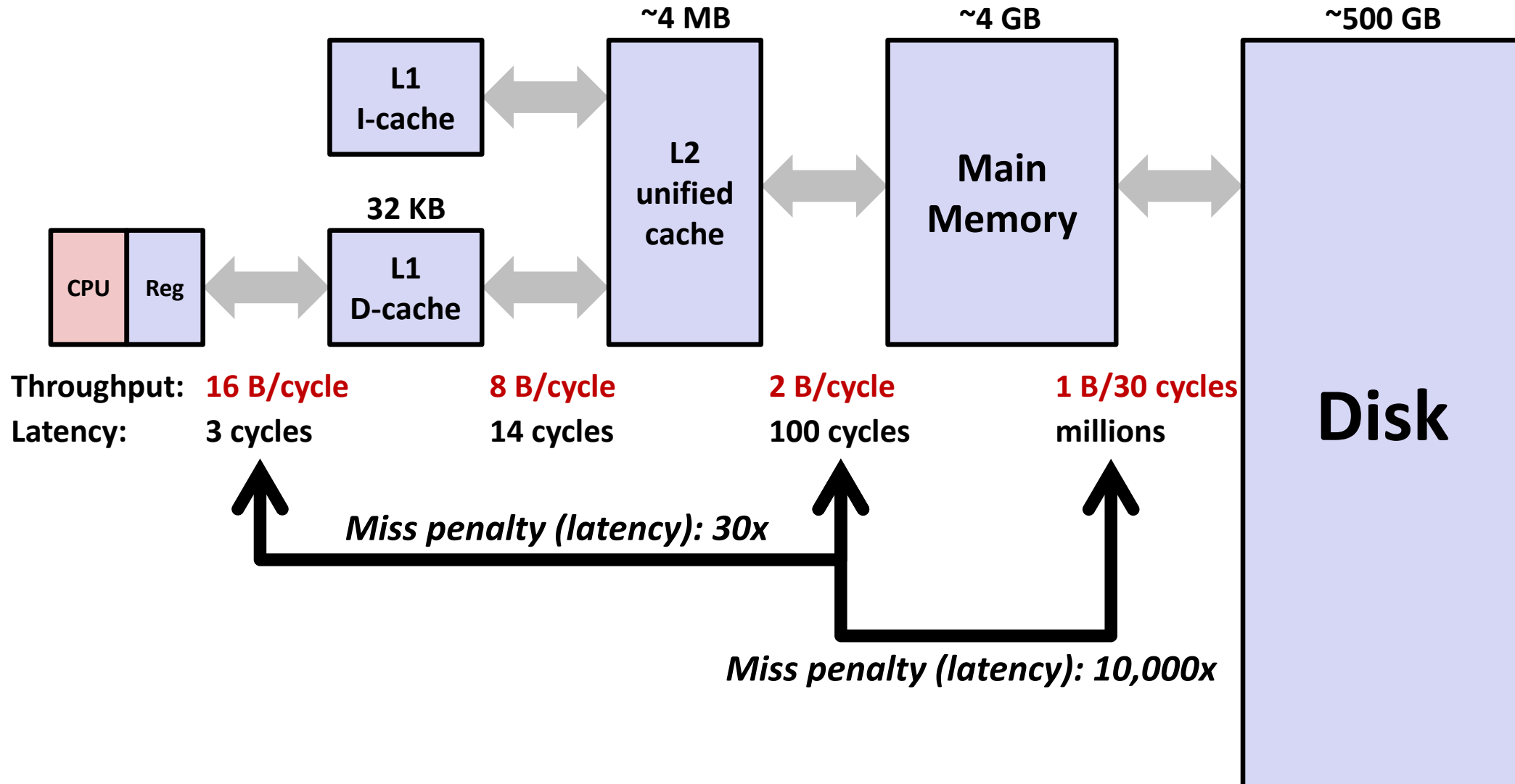
DRAM Cache Organization

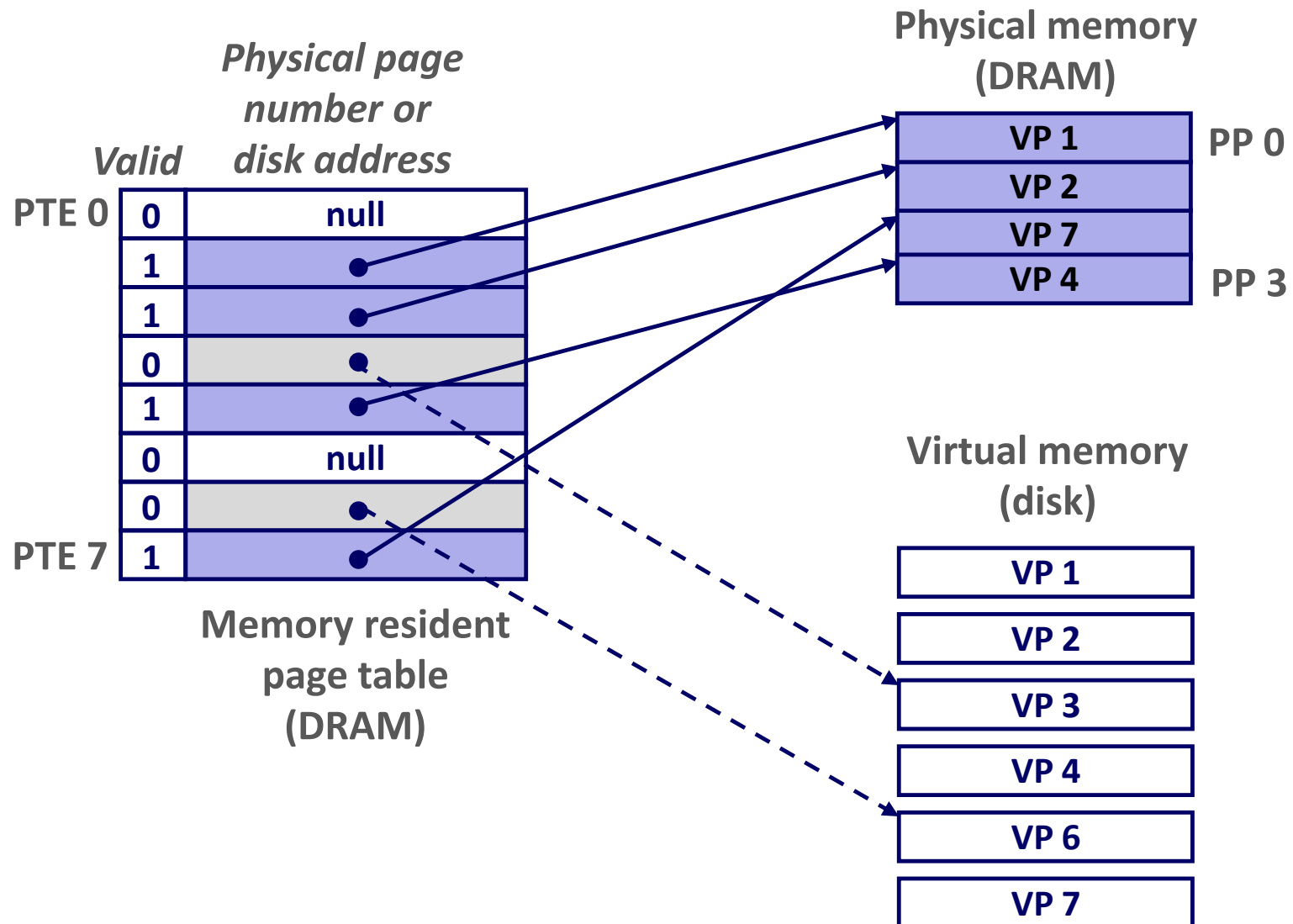- **DRAM cache organization driven by the enormous miss penalty**
  - DRAM is about *10x* slower than SRAM
  - Disk is about *10,000x* slower than DRAM
    - For first byte, faster for next byte

- **Consequences**
  - Large page (block) size: typically 4-8 KB, sometimes 4 MB
  - Fully associative
    - Any VP can be placed in any PP
    - 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

Address Translation: 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

Address Translation With a Page Table

Page table base register (PTBR)

Page table address for process

Virtual address

Virtual page number (VPN)	Virtual page offset (VPO)

Page table

Valid | **Physical page number (PPN)**

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

Physical page number (PPN)	Physical page offset (PPO)

Physical address
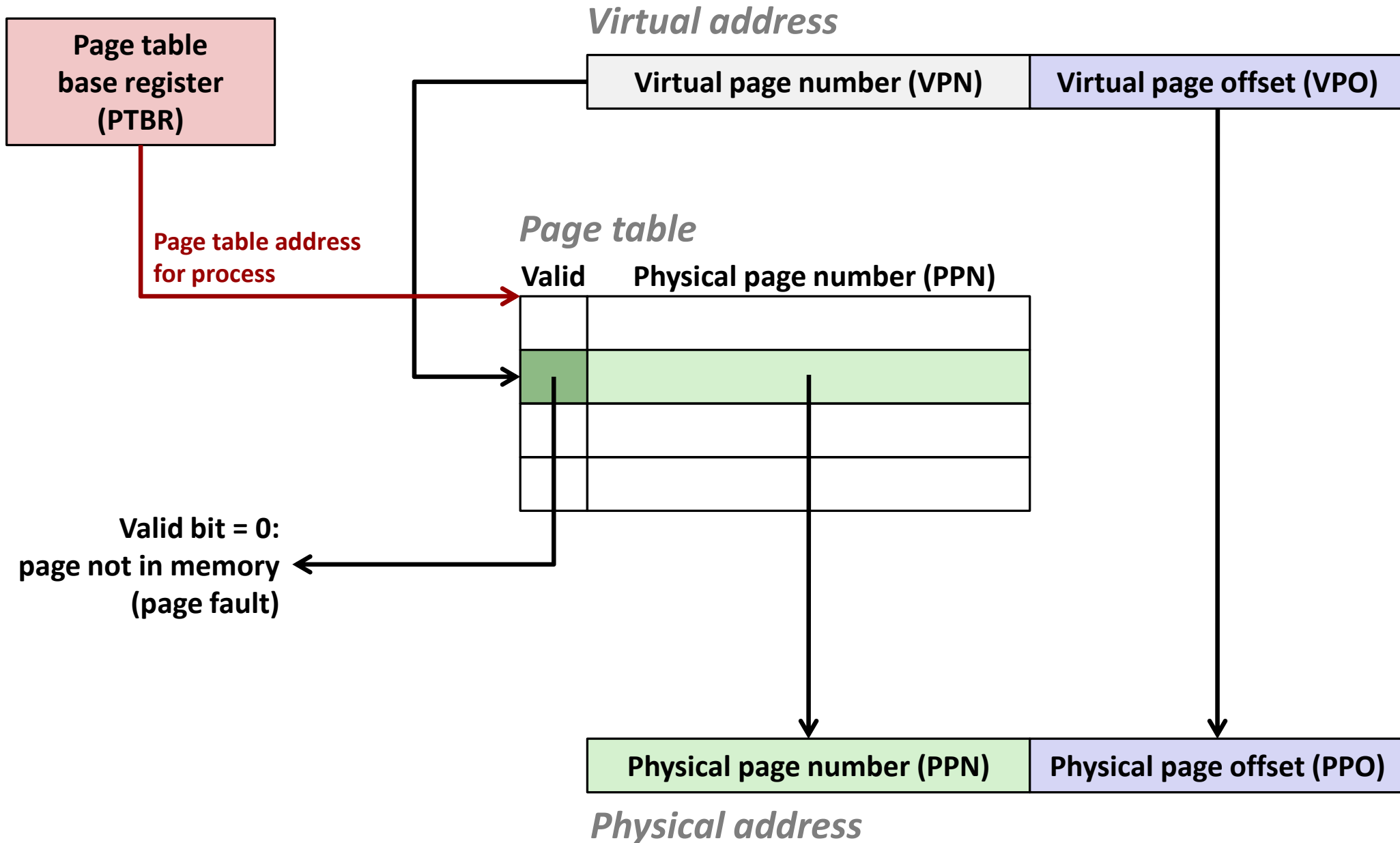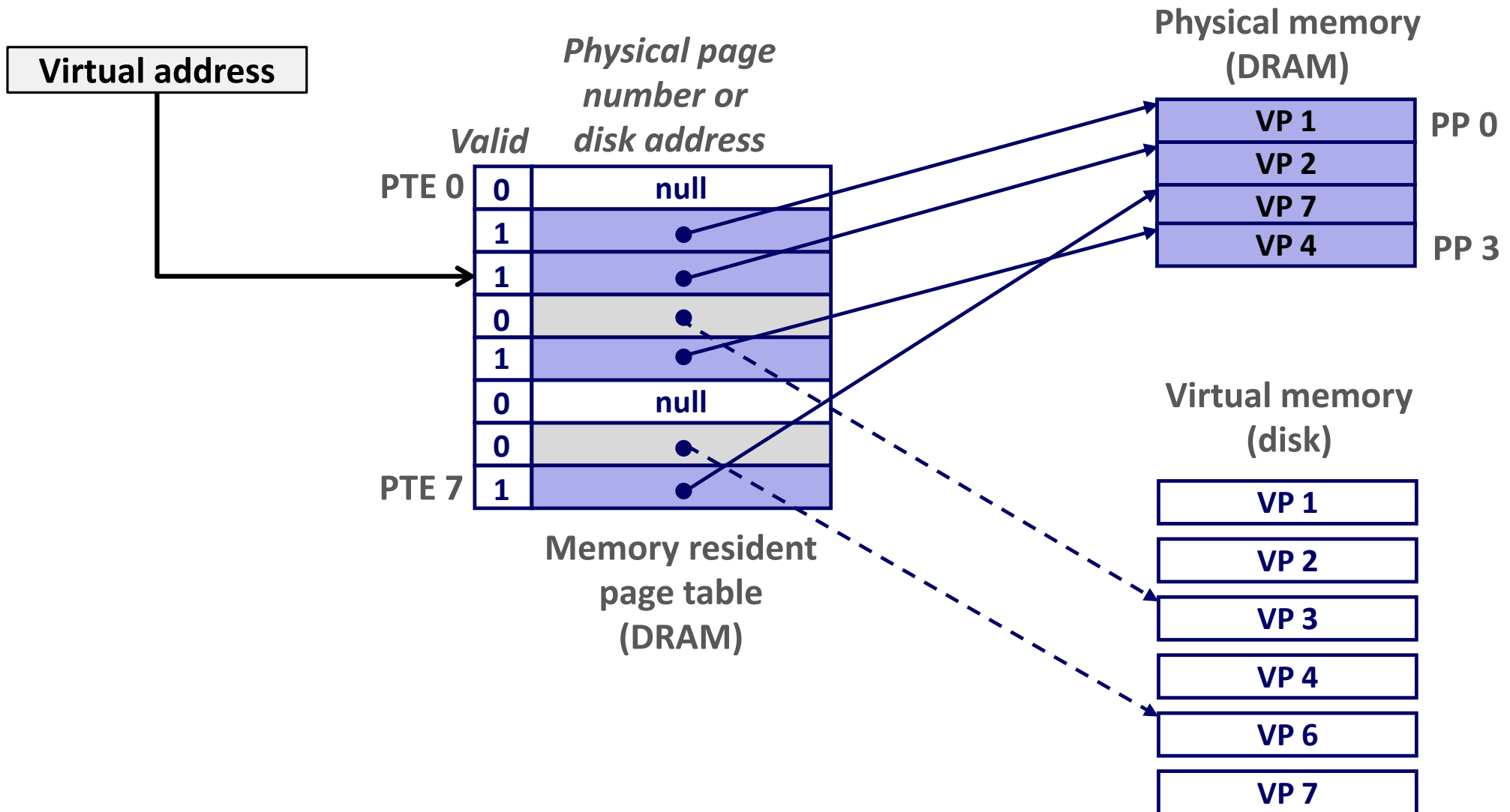
Page Hit

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

Page Miss

- *Page miss:* **reference to VM word that is not in physical memory**
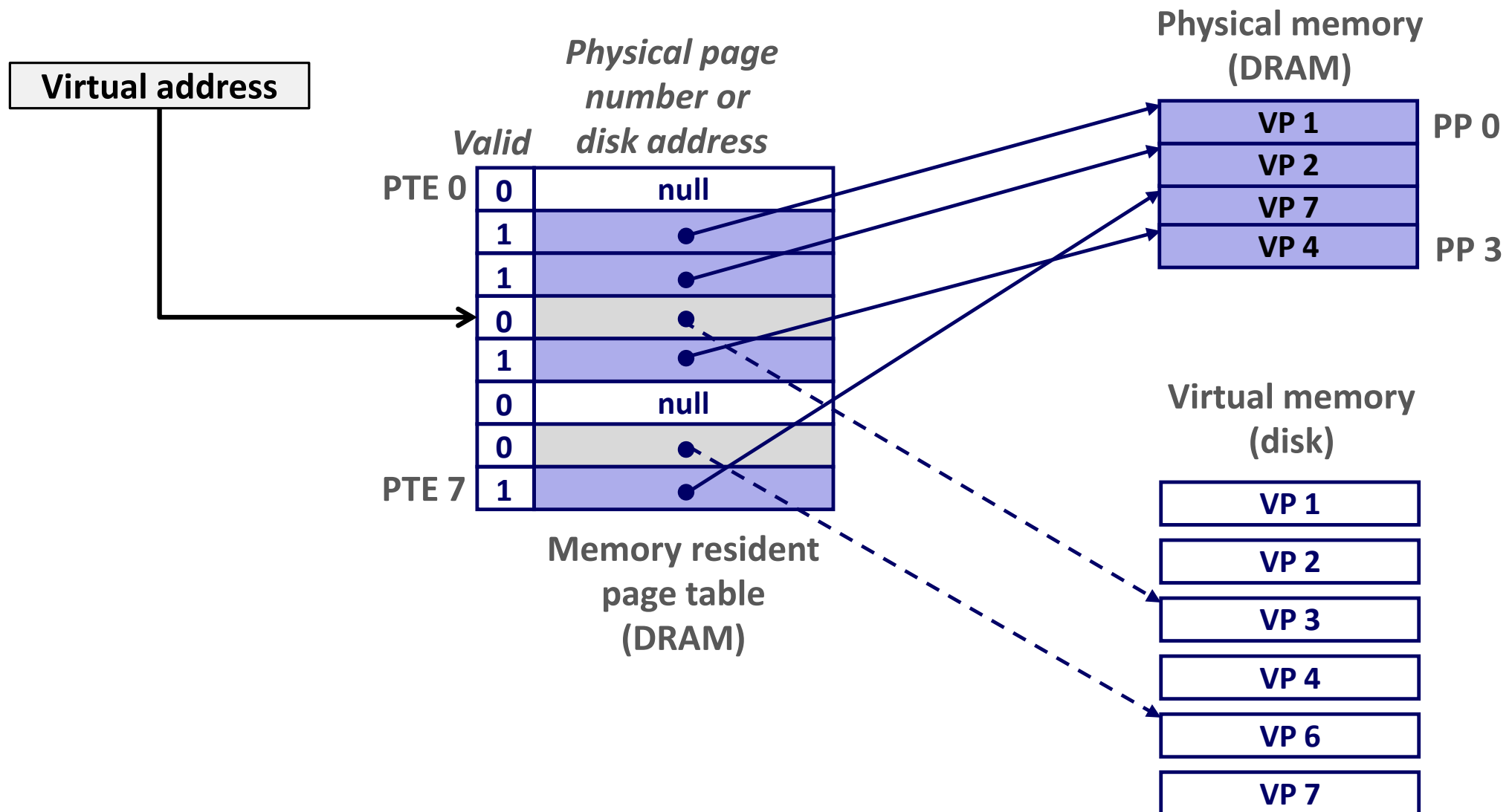
Handling Page Fault
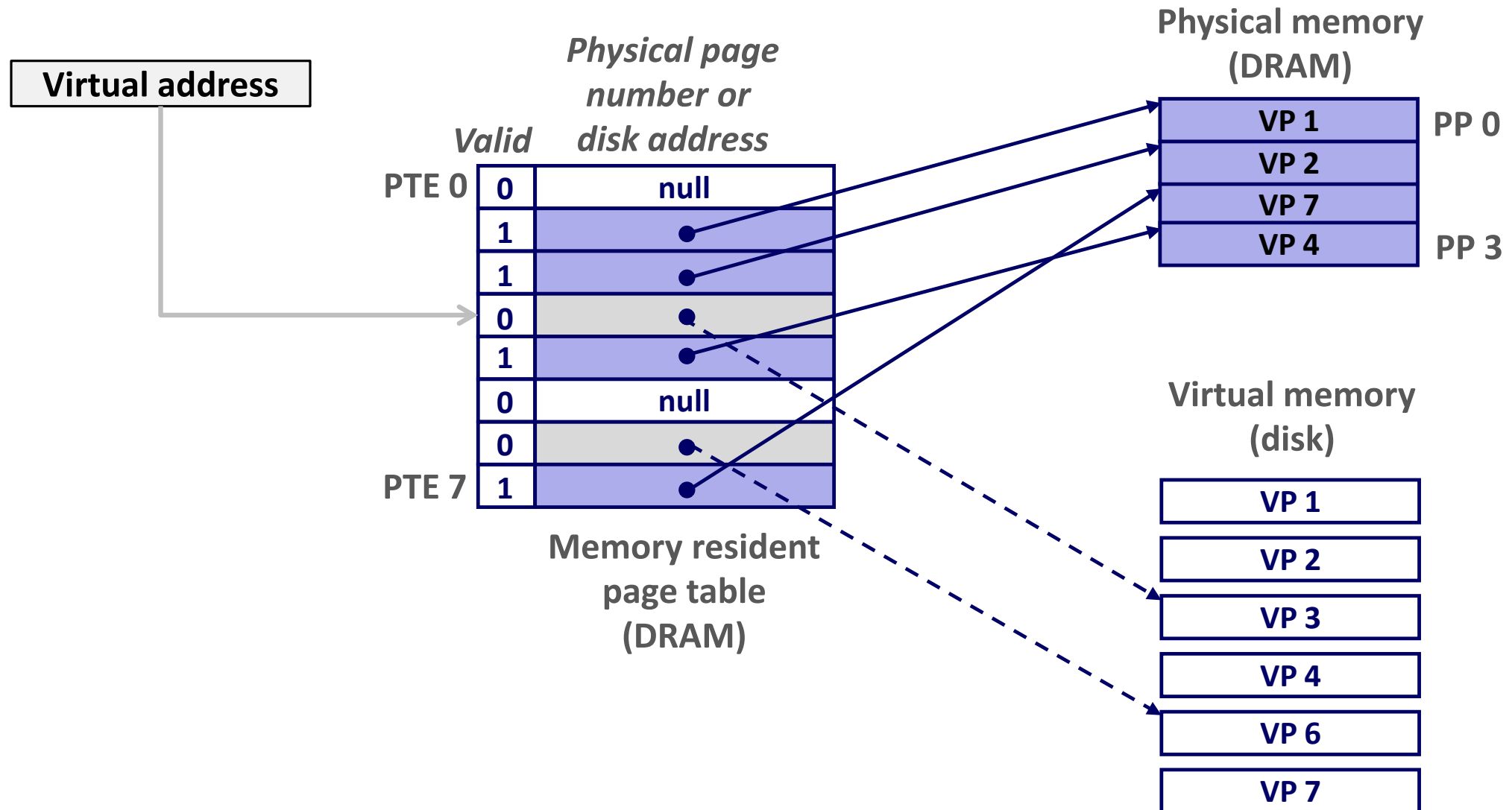
- Page miss causes page fault (an exception)

Handling Page Fault

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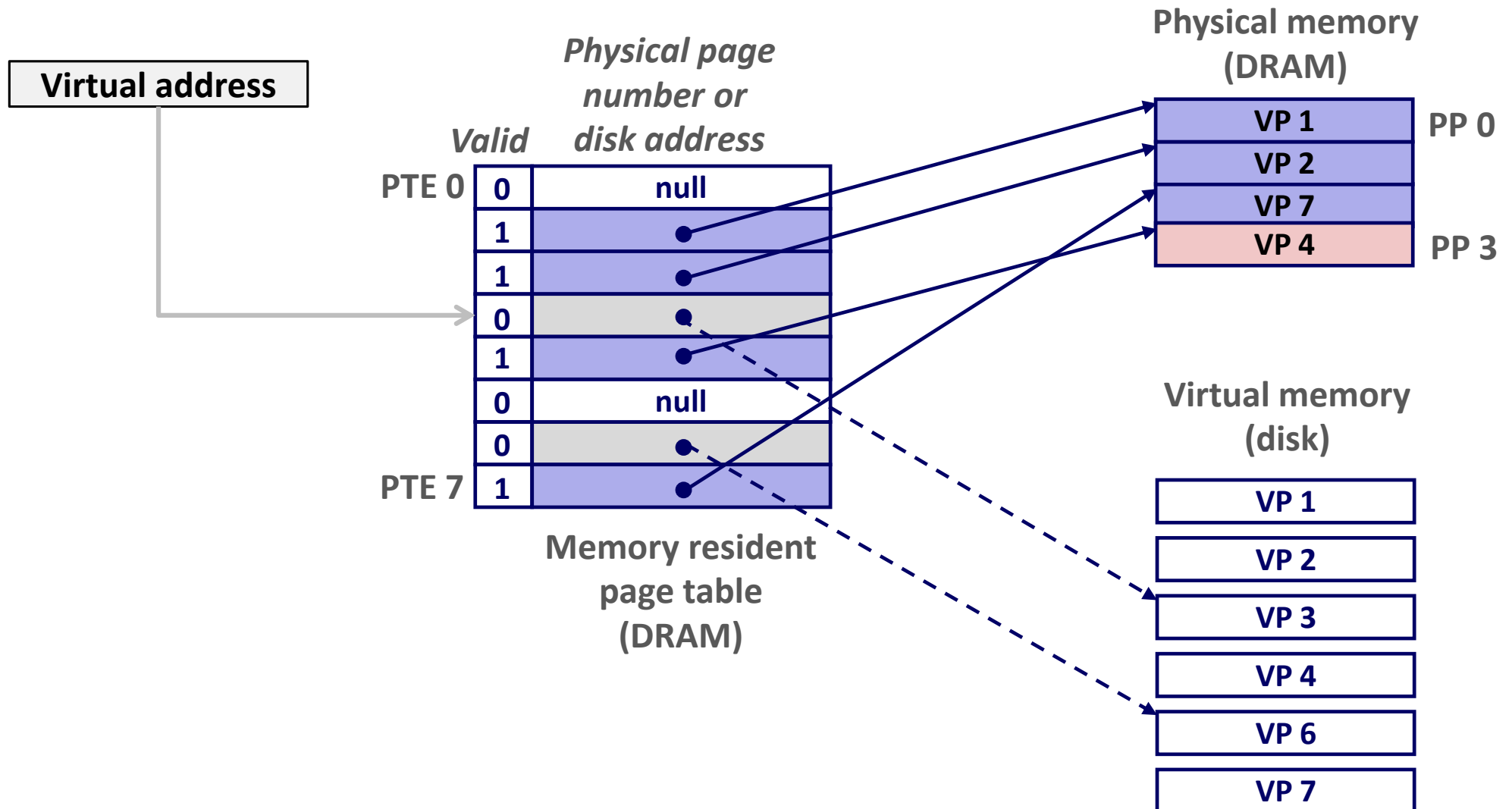
Handling Page Fault

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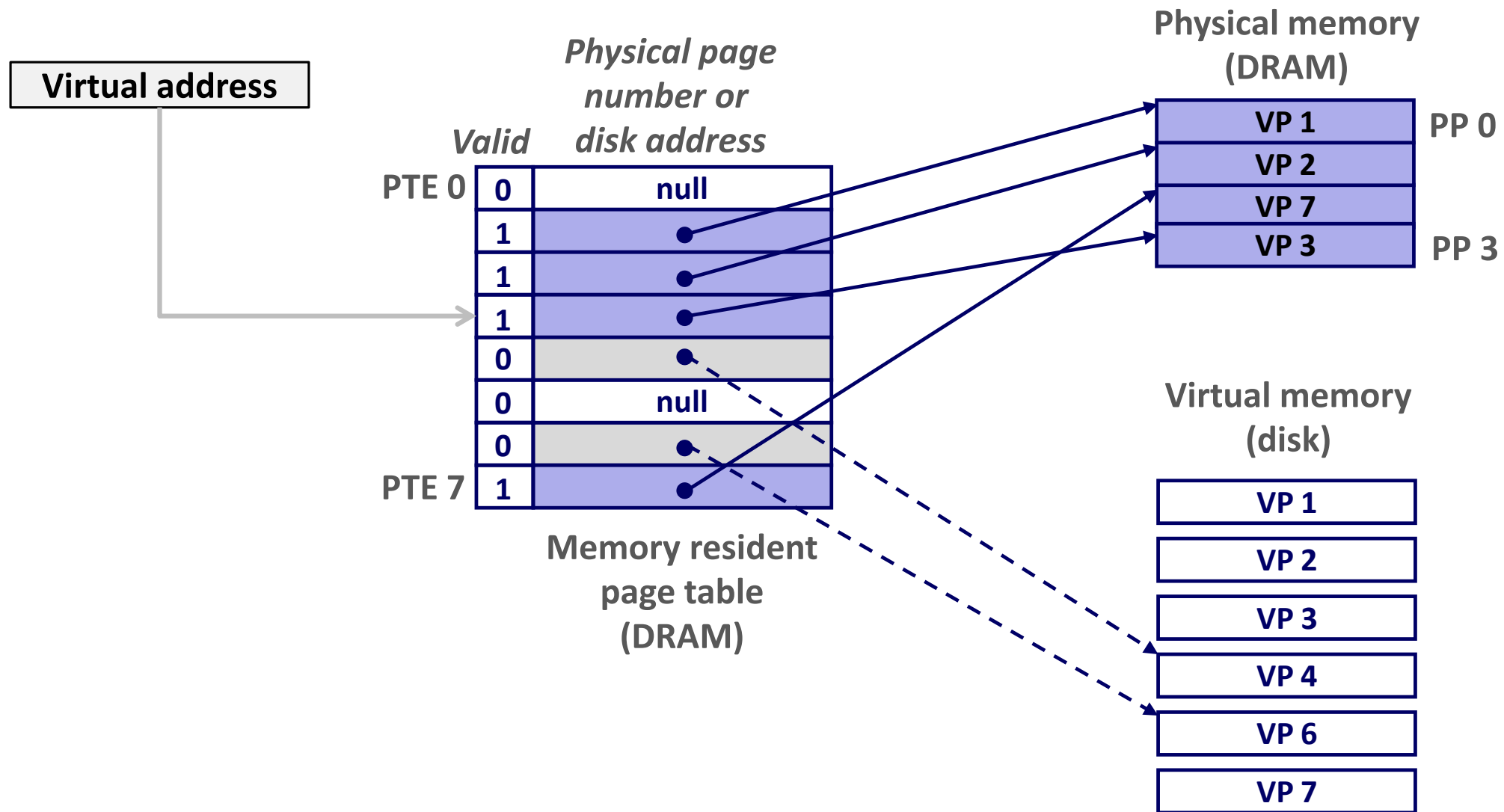
Handling Page Fault

- 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!
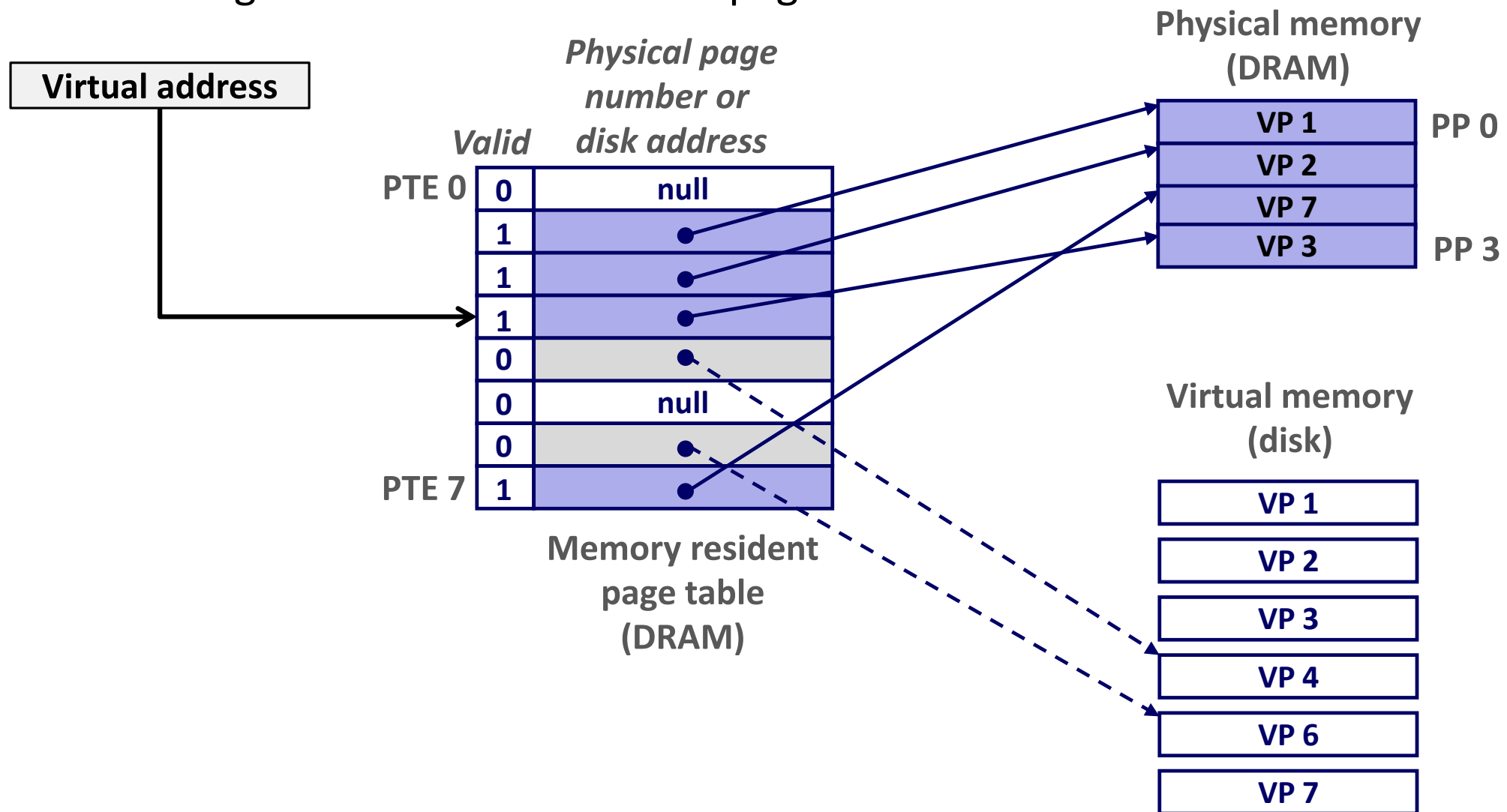
Why does it work? Locality

- **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

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
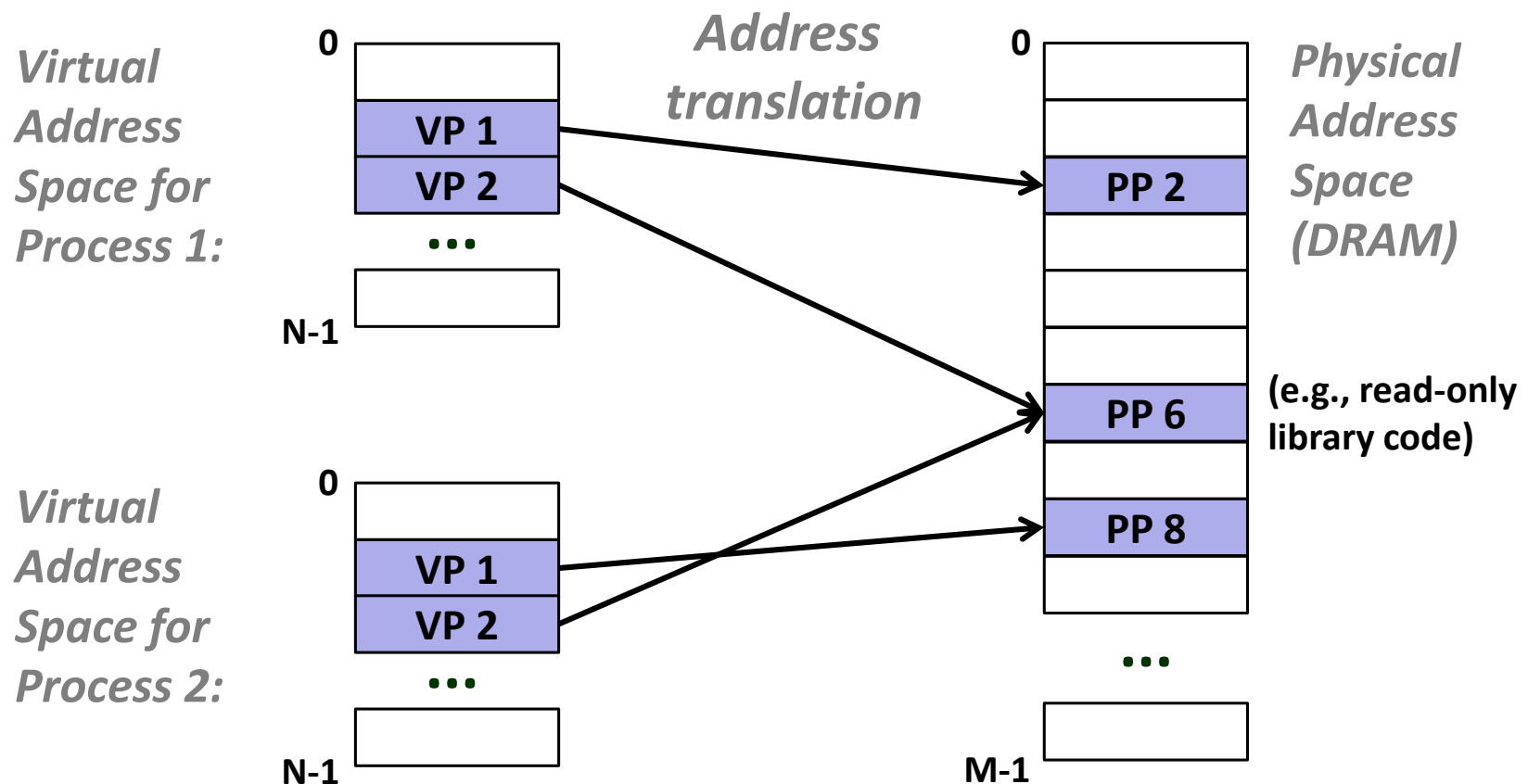
VM as a Tool for Memory Management

- **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)
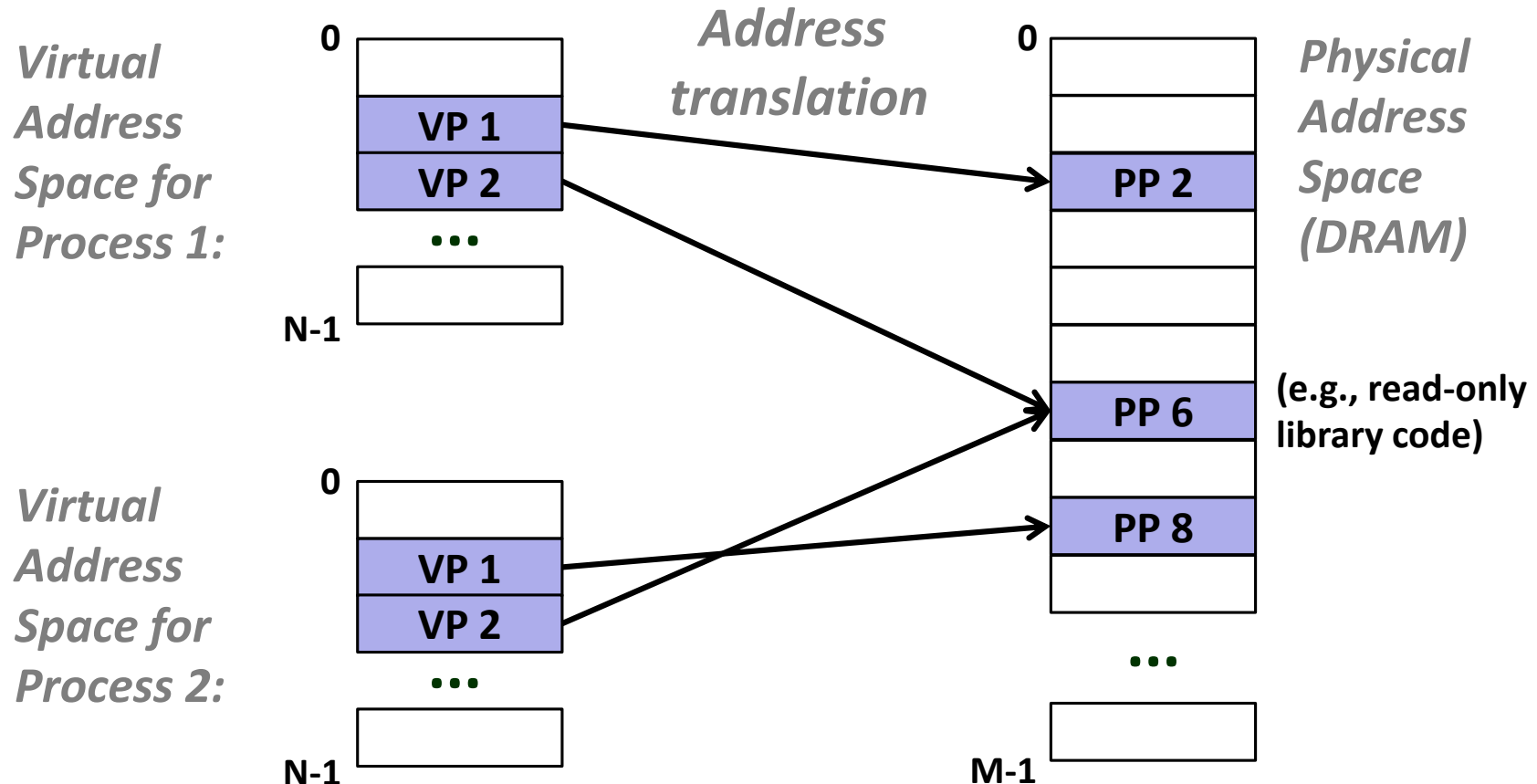
Simplifying Linking and Loading

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

Kernel virtual memory	Memory invisible to user code
User stack (created at runtime)	
	%esp (stack pointer)
Memory-mapped region for shared libraries	
Run-time heap (created by malloc)	brk
Read/write segment (.data, .bss)	Loaded from the executable file
Read-only segment (.init, .text, .rodata)	
Unused	

0xc0000000

0x40000000

0x08048000

0
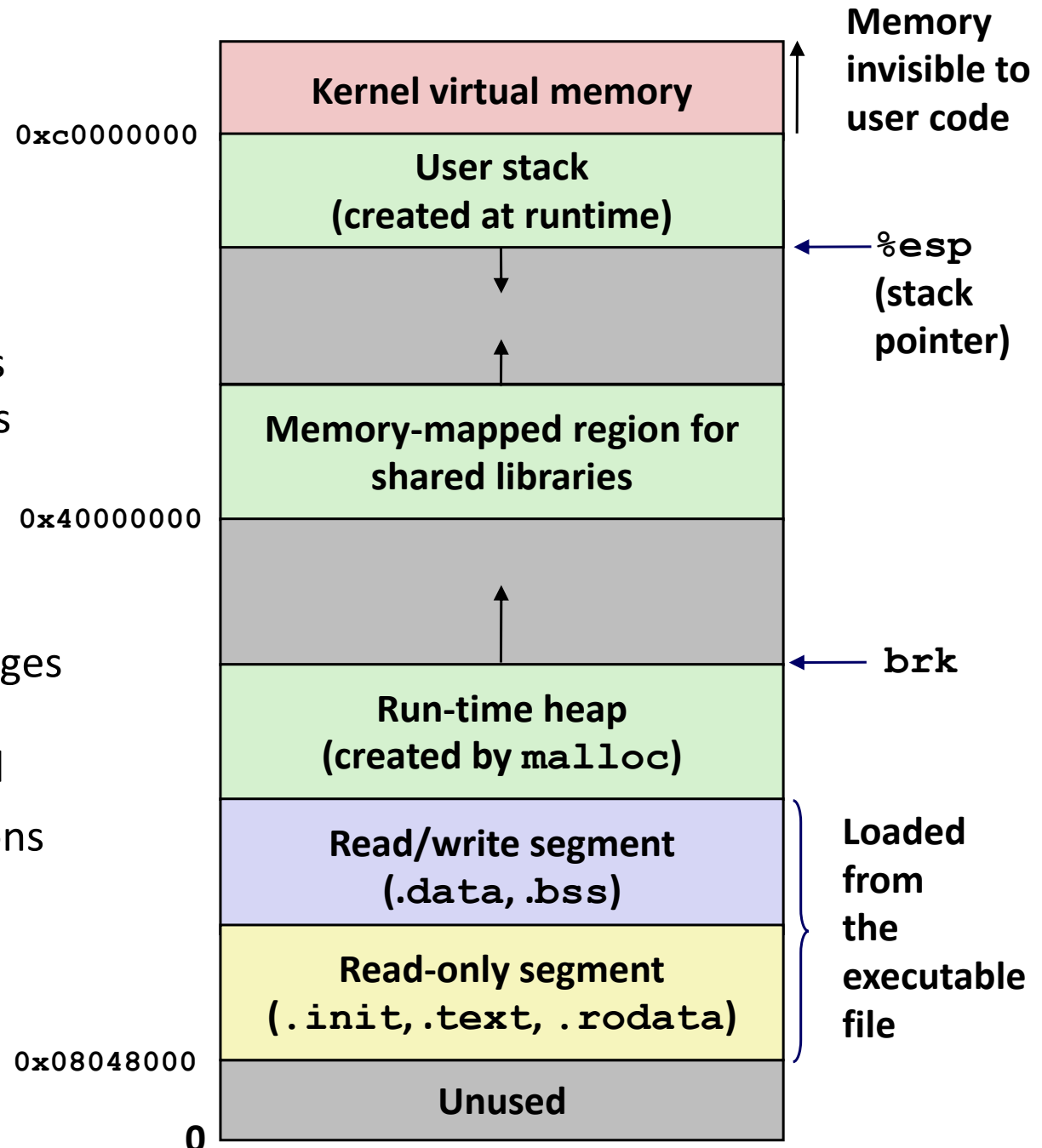
VM as a Tool for Memory Protection

- **Extend PTEs with permission bits**

- **Page fault handler checks these before remapping**
  - If violated, send process SIGSEGV (segmentation fault)

Physical Address Space

Process i:

	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 6
VP 1:	No	Yes	Yes	PP 4
VP 2:	Yes	Yes	Yes	PP 2

⋮

Process j:

	SUP	READ	WRITE	Address
VP 0:	No	Yes	No	PP 9
VP 1:	Yes	Yes	Yes	PP 6
VP 2:	No	Yes	Yes	PP 11

PP 2

PP 4

PP 6

PP 8
PP 9

PP 11
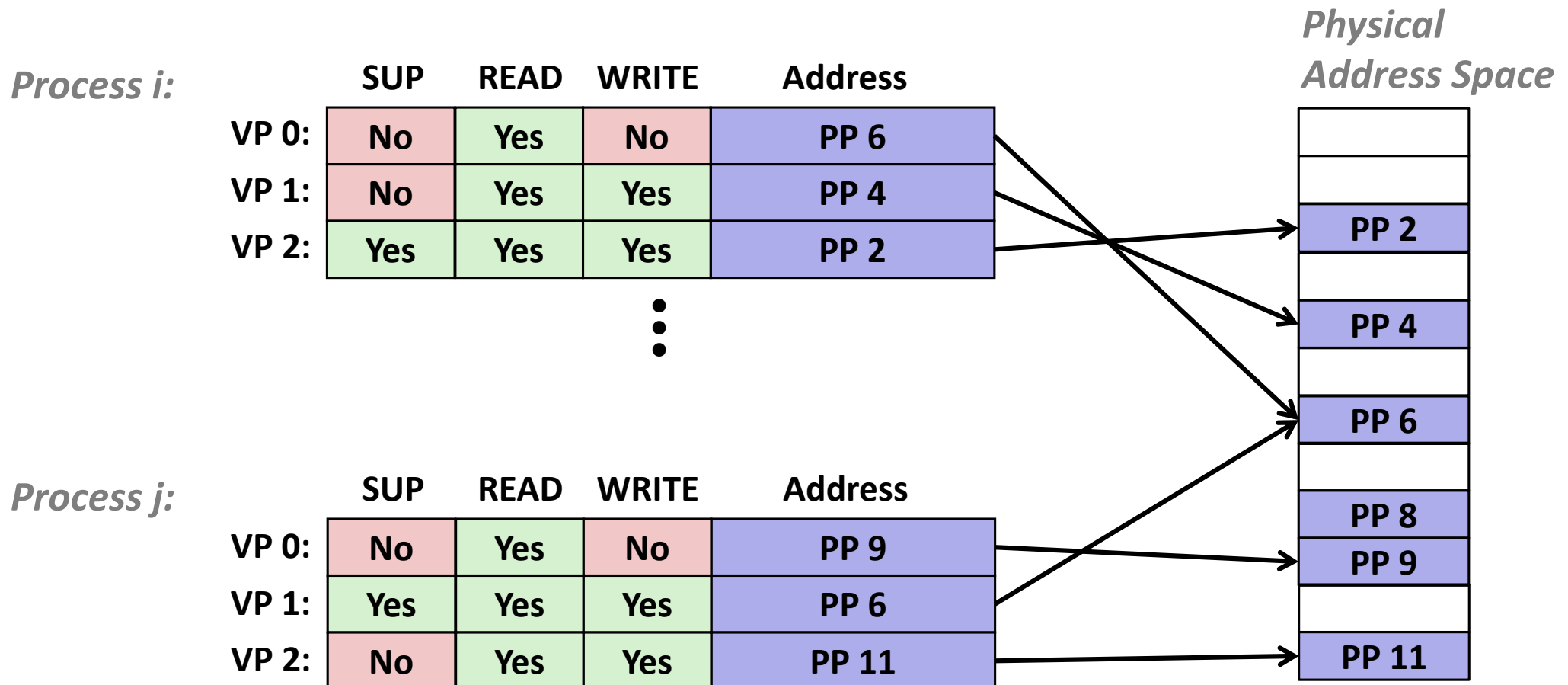
VM Address Translation

- **Virtual Address Space**
  - $V = \{0, 1, ..., N-1\}$

- **Physical Address Space**
  - $P = \{0, 1, ..., M-1\}$

- **Address Translation**
  - *MAP: $V \rightarrow P\ U\ \{\varnothing\}$*
  - For virtual address **a**:
    - *MAP(a) = a'* if data at virtual address **a** is at physical address **a'** in **P**
    - *MAP(a) = $\varnothing$* 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)**

  - **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

Address Translation With a Page Table

Virtual address

	Virtual page number (VPN)	Virtual page offset (VPO)

n-1 ⟶ *p* *p-1* ⟶ *0*

Page table base register (PTBR)

Page table address for process

Page table

Valid **Physical page number (PPN)**

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

m-1 ⟶ *p* *p-1* ⟶ *0*

Physical page number (PPN)	Physical page offset (PPO)
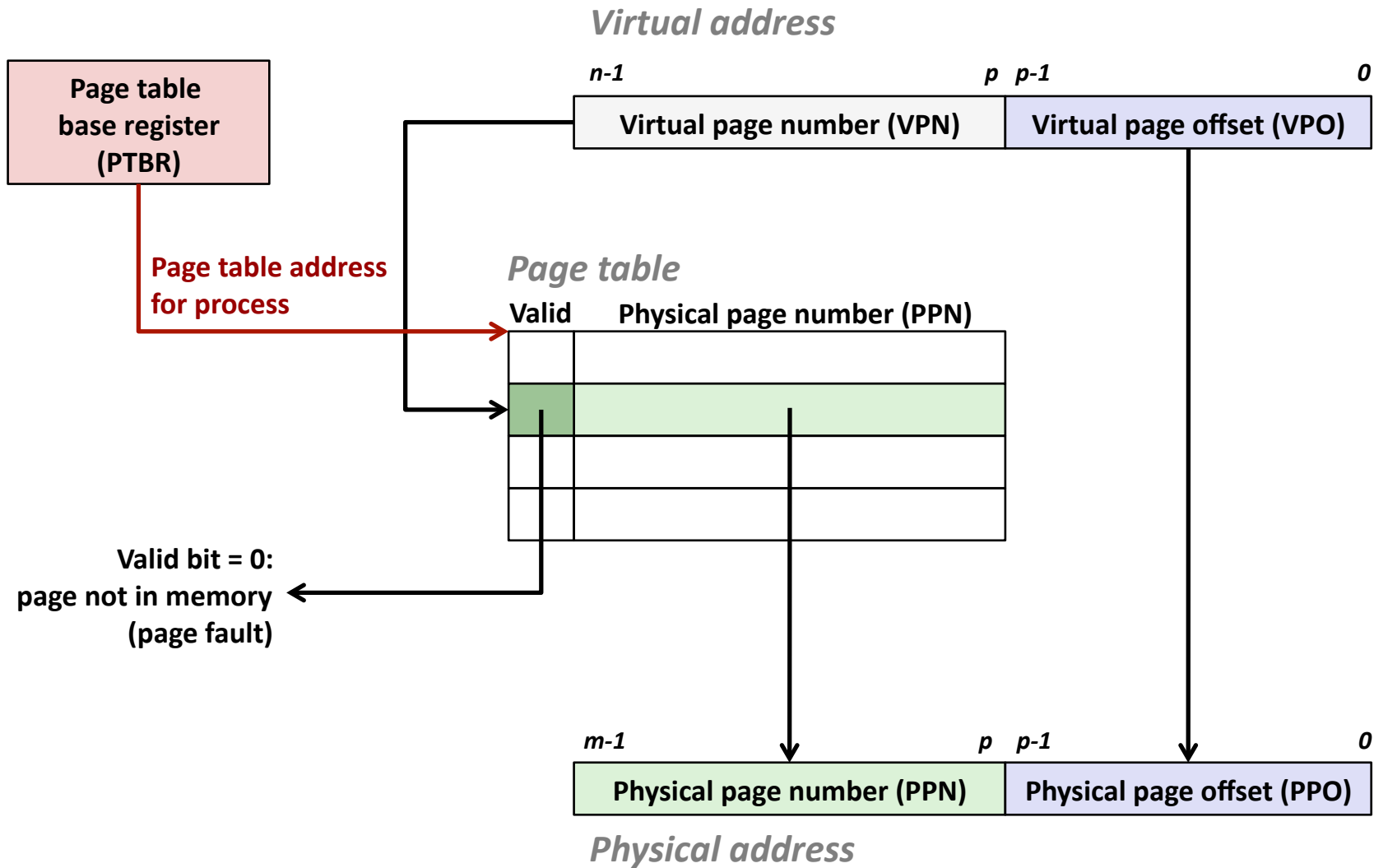
Physical address

Address Translation: Page Hit
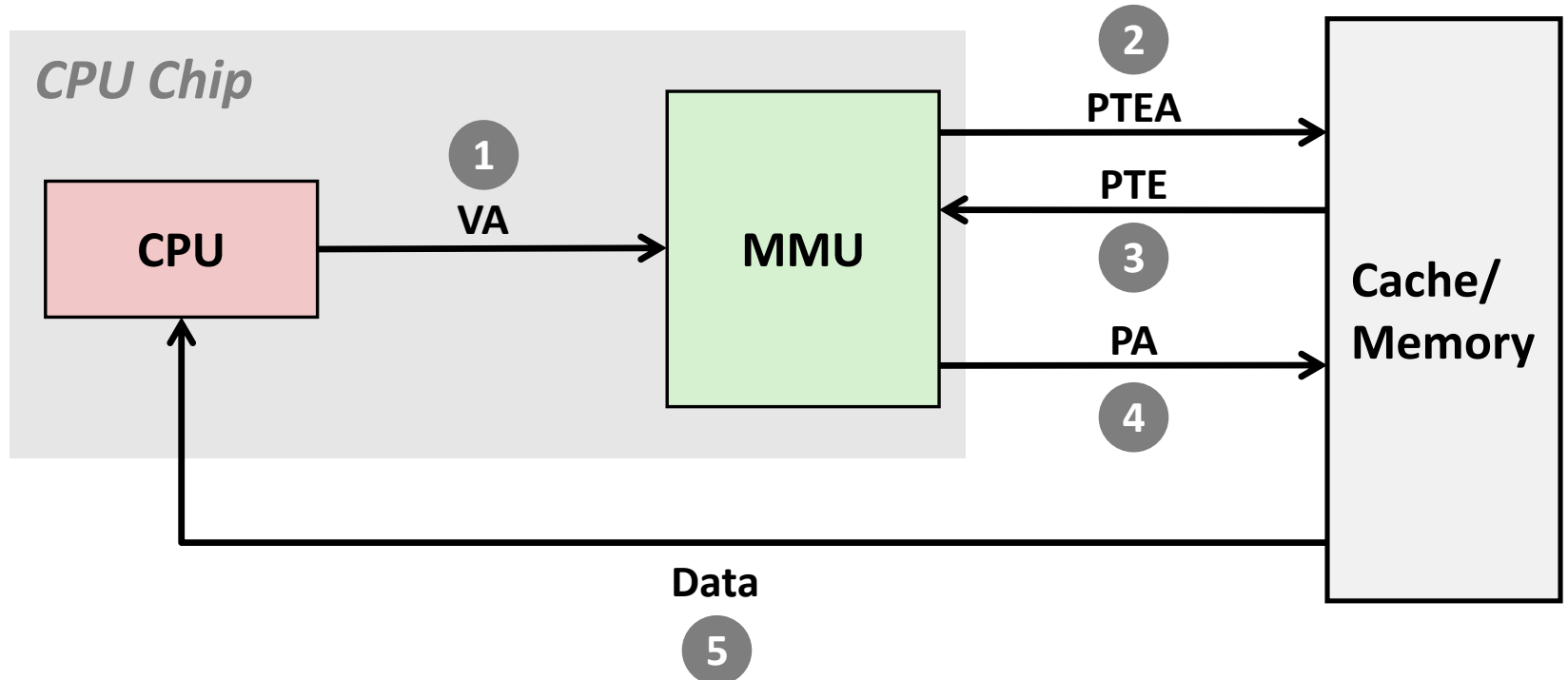
1) Processor sends virtual address to MMU

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

Address Translation: Page Fault
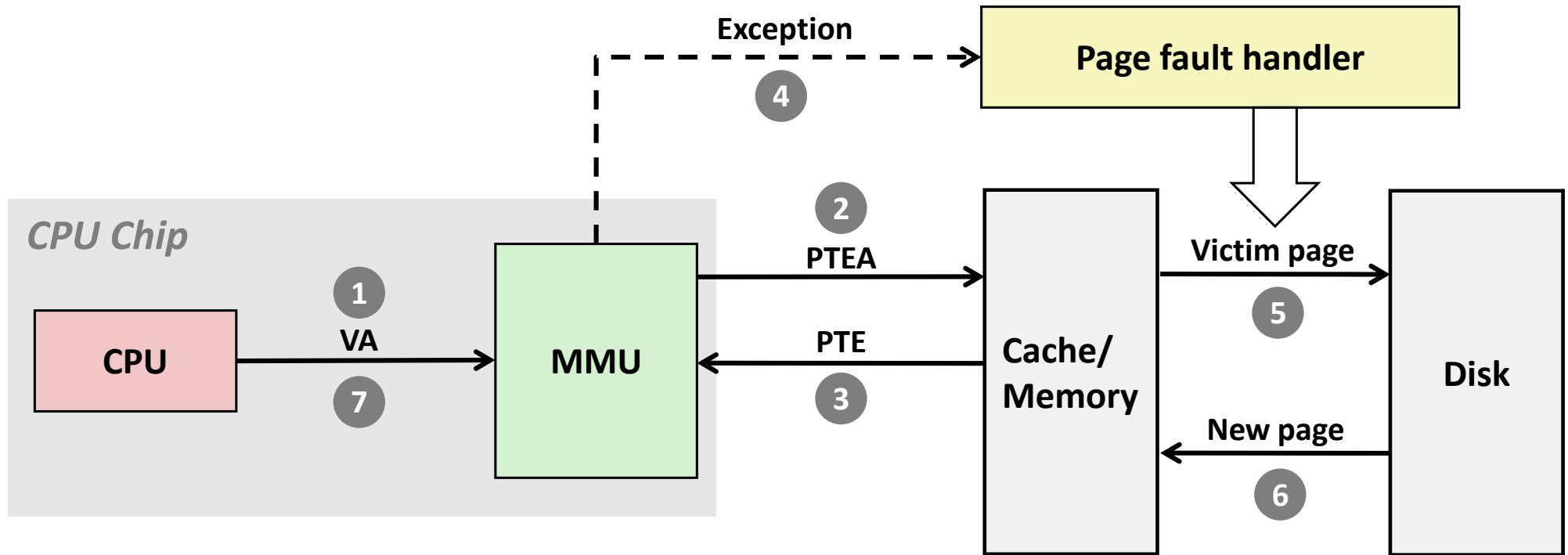
1) Processor sends virtual address to MMU

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

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

TLB Hit
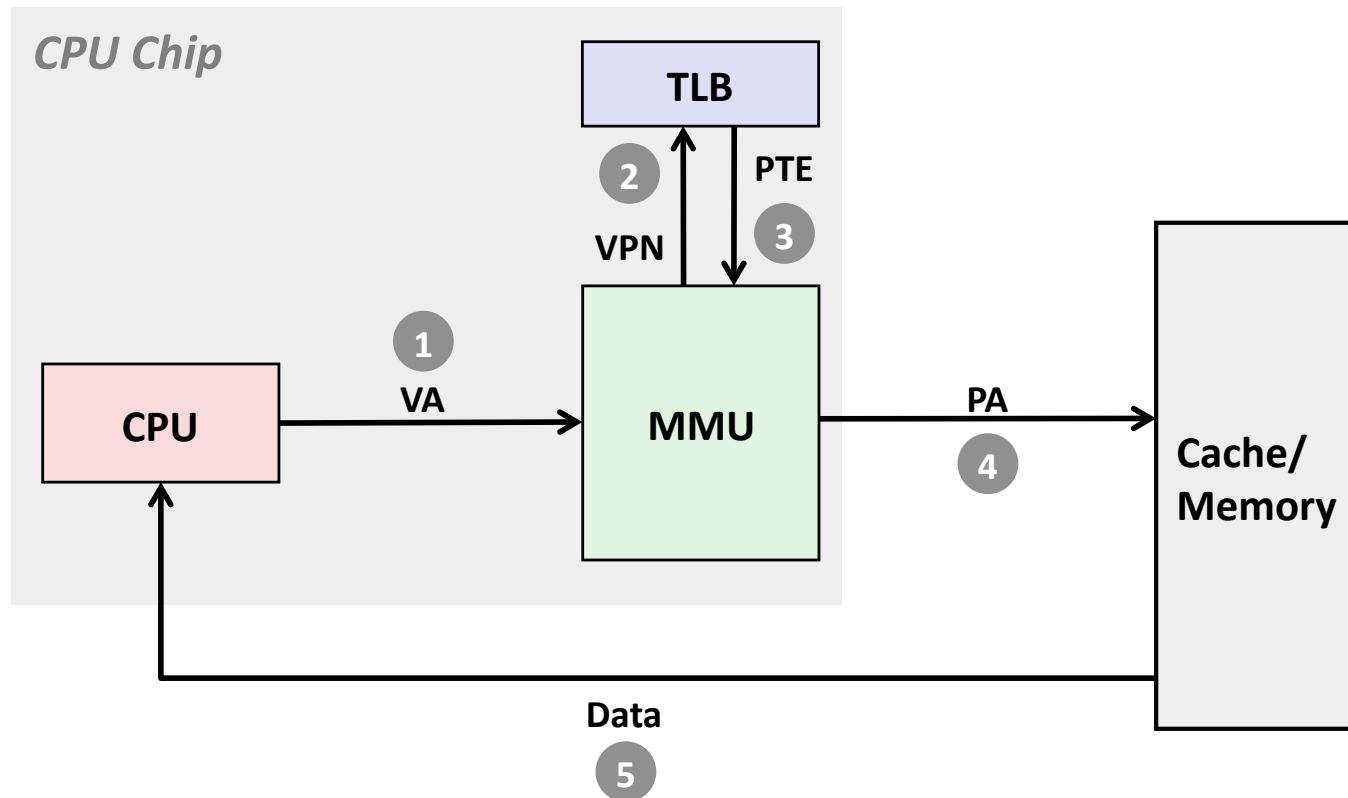
A TLB hit eliminates a memory access

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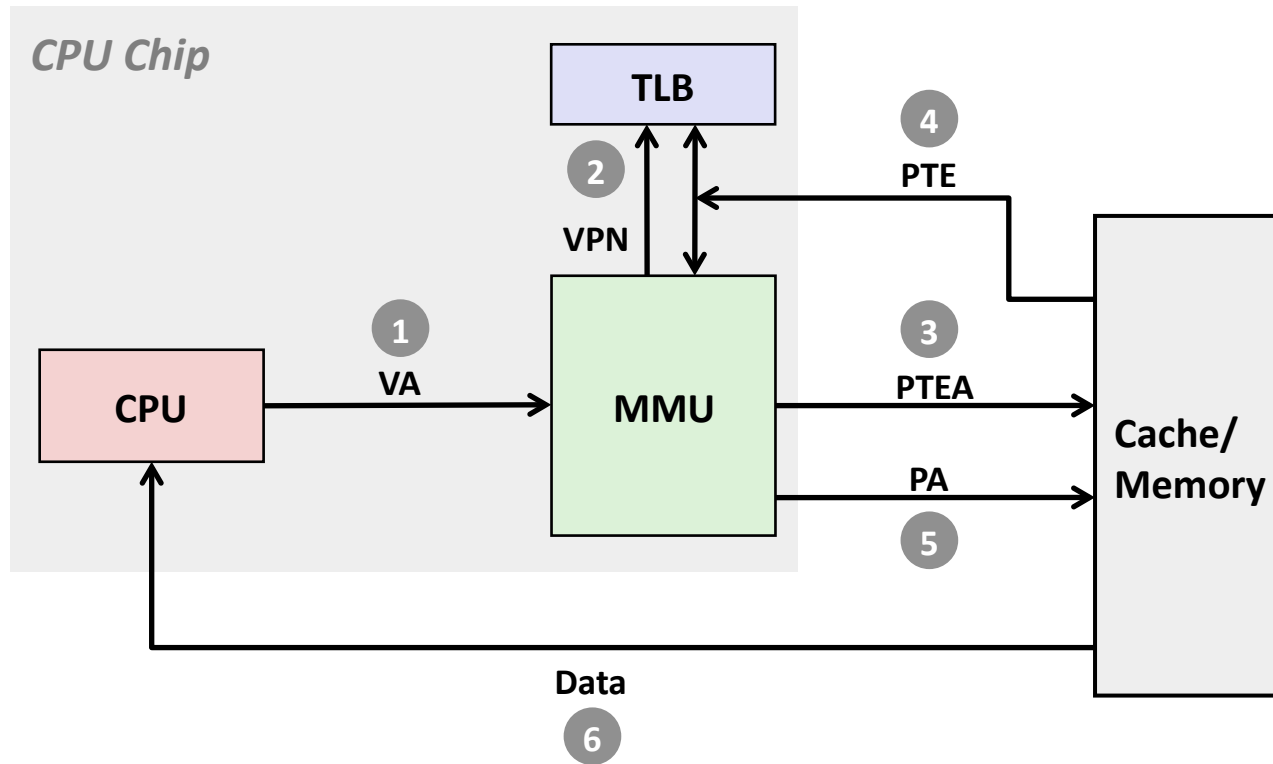
A TLB miss incurs an additional memory access (the PTE)

Fortunately, TLB misses are rare. Why?

Multi-Level Page Tables

- **Suppose:**
  - 4KB (2^{12}) page size, 48-bit address space, 8-byte PTE

- **Problem:**
  - Would need a 512 GB page table!
    - $2^{48} * 2^{-12} * 2^3 = 2^{39}$ bytes

- **Common solution:**
  - Multi-level page tables
  - 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)

Level 2 Tables

Level 1 Table

...
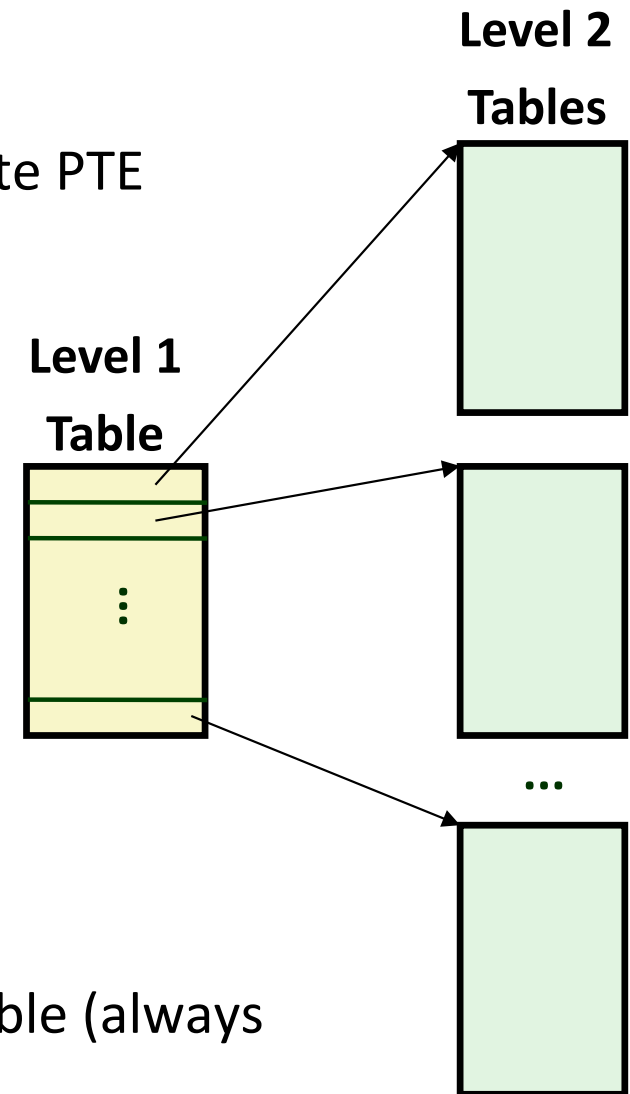
A Two-Level Page Table Hierarchy

Level 1 page table — PTE 0, PTE 1, PTE 2 (null), PTE 3 (null), PTE 4 (null), PTE 5 (null), PTE 6 (null), PTE 7 (null), PTE 8, (1K - 9) null PTEs

Level 2 page tables — PTE 0, ..., PTE 1023; PTE 0, ..., PTE 1023; 1023 null PTEs, PTE 1023

Virtual memory — VP 0, ..., VP 1023, VP 1024, ..., VP 2047 (2K allocated VM pages for code and data); Gap (6K unallocated VM pages); 1023 unallocated pages (1023 unallocated pages); VP 9215 (1 allocated VM page for the stack)

32 bit addresses, 4KB pages, 4-byte PTEs

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

Integrating VM and Cache
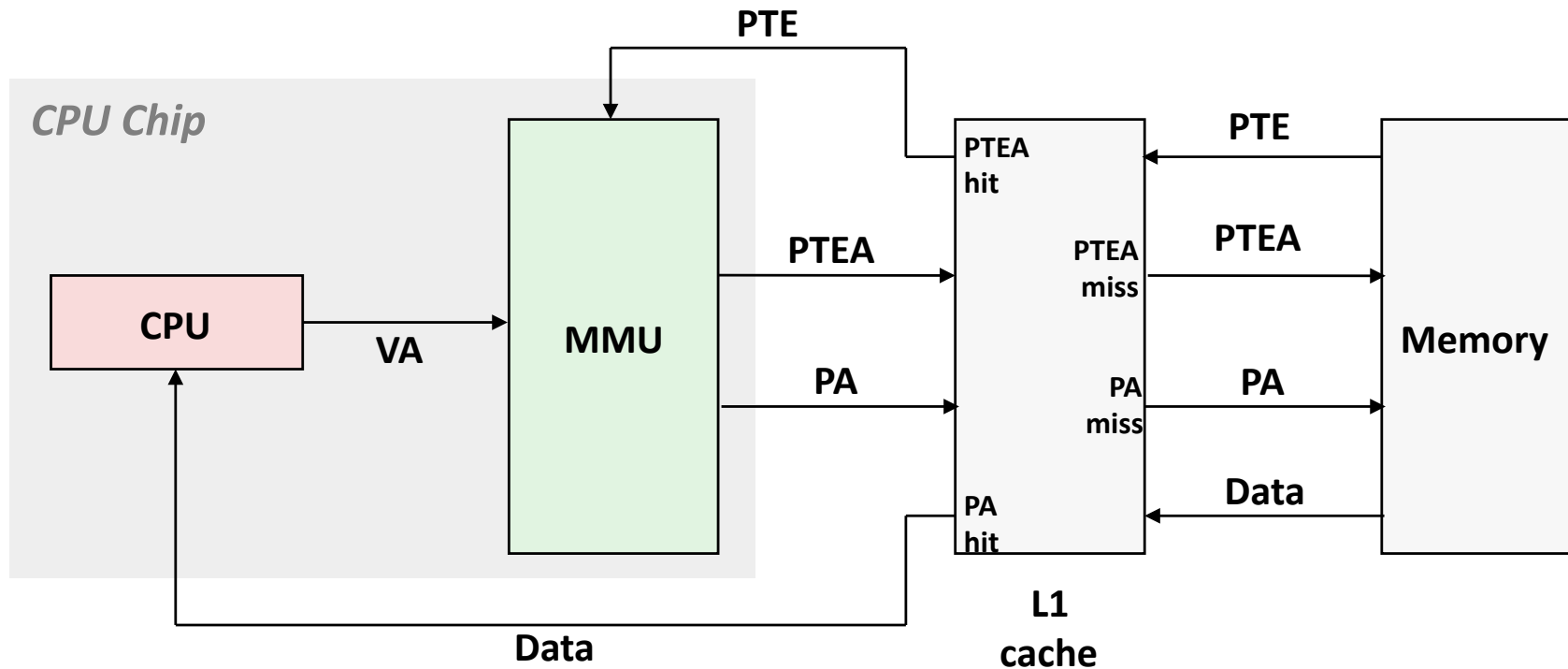
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

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

Simple Memory System Example

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

13	12	11	10	9	8	7	6	5	4	3	2	1	0

← ————————————— VPN ————————————— →← ————————— VPO ————————— →

Virtual Page Number   **Virtual Page Offset**

11	10	9	8	7	6	5	4	3	2	1	0

← ————————————— PPN ————————————— →← ————————— PPO ————————— →

Physical Page Number   **Physical Page Offset**
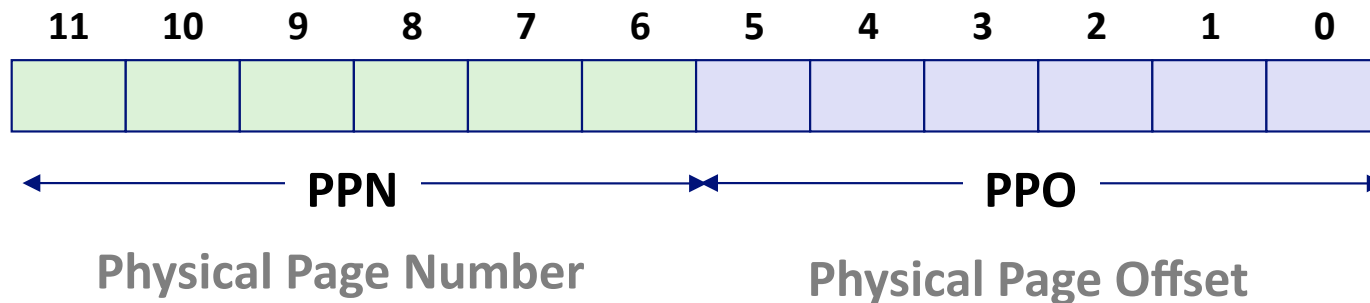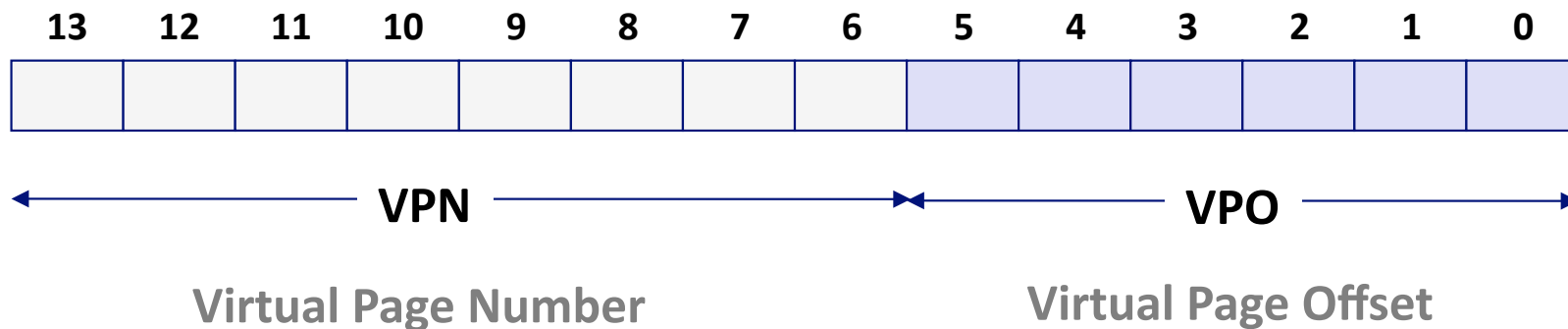
Simple Memory System Page Table

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

Simple Memory System TLB

- **16 entries**
- **4-way associative**

	13	12	11	10	9	8	7	6	5	4	3	2	1	0

TLBT ← bits 13–7 → | TLBI ← bits 6–5 →

VPN ← bits 13–6 → | VPO ← bits 5–0 →
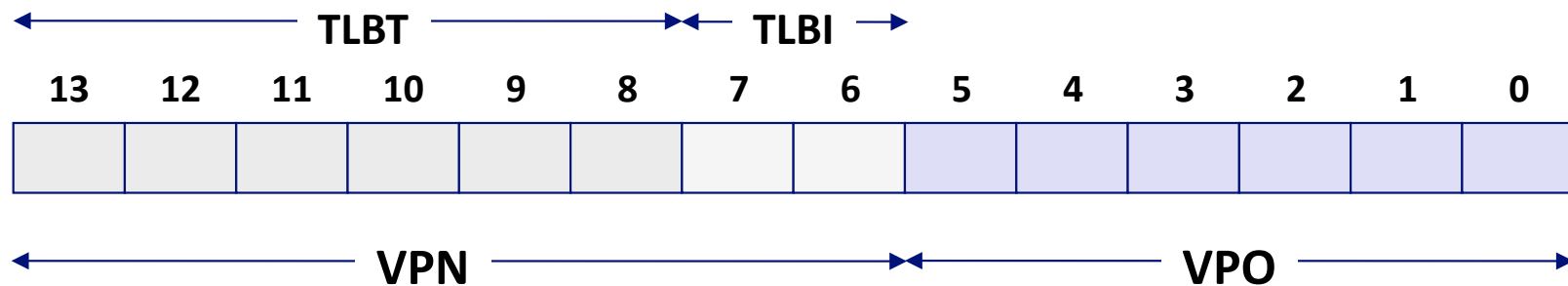
Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	03	–	0	09	0D	1	00	–	0	07	02	1
1	03	2D	1	02	–	0	04	–	0	0A	–	0
2	02	–	0	08	–	0	06	–	0	03	–	0
3	07	–	0	03	0D	1	0A	34	1	02	–	0

Simple Memory System Cache

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

	CT						CI				CO	
11	10	9	8	7	6	5	4	3	2	1	0

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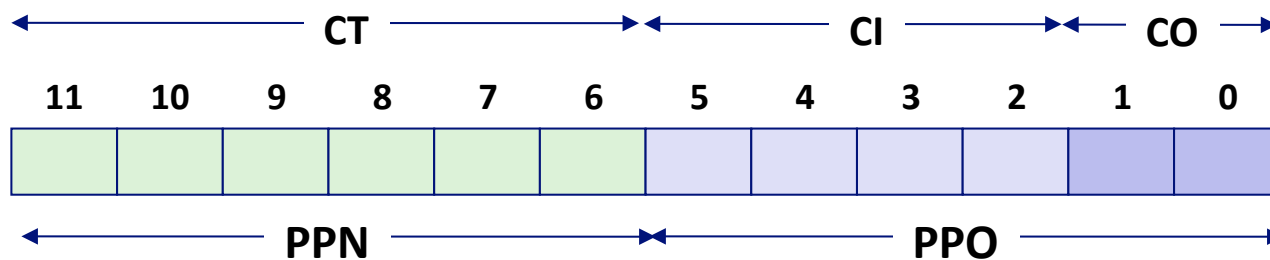
Idx	Tag	Valid	B0	B1	B2	B3
0	19	1	99	11	23	11
1	15	0	–	–	–	–
2	1B	1	00	02	04	08
3	36	0	–	–	–	–
4	32	1	43	6D	8F	09
5	0D	1	36	72	F0	1D
6	31	0	–	–	–	–
7	16	1	11	C2	DF	03

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

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	1	1	1	1	0	1	0	1	0	0

← TLBT → ← TLBI →

← VPN → ← VPO →
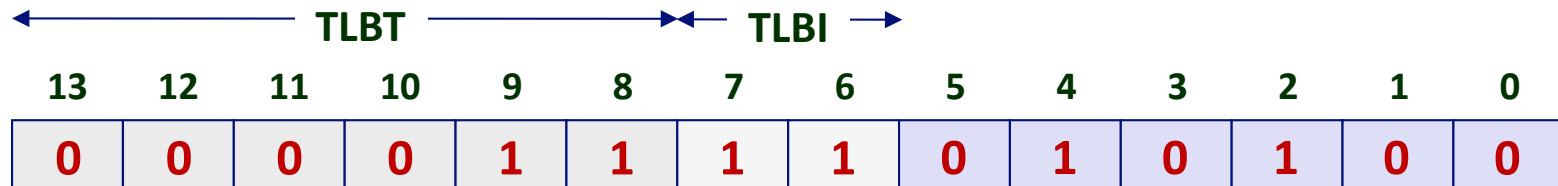
VPN **0x0F**   TLBI **0x3**   TLBT **0x03**   TLB Hit? **Y**   Page Fault? **N**   PPN: **0x0D**

Physical Address

11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	1	0	1	0	1	0	0

← CT → ← CI → ← CO →

← PPN → ← PPO →
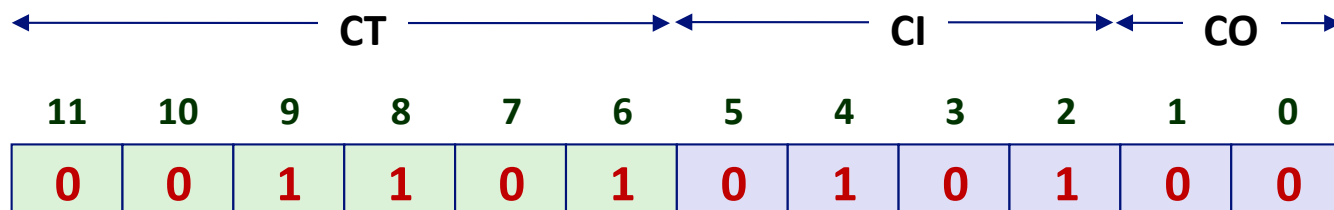
CO **0**   CI **0x5**   CT **0x0D**   Hit? **Y**   Byte: **0x36**

Address Translation Example #2

Virtual Address: `0x010F`

TLBT ← → TLBI →

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	0	0	0	1	1	1	1

← VPN → ← VPO →
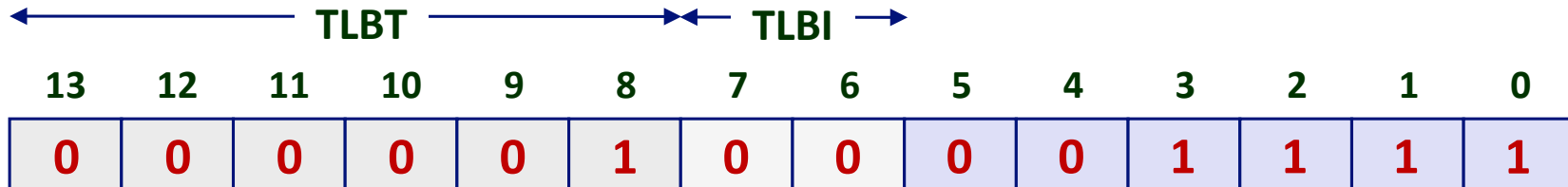
VPN **0x04**   TLBI **0**   TLBT **0x01**   TLB Hit? **N**   Page Fault? **Y**   PPN: **TBD**

Physical Address

← CT → ← CI → ← CO →

11	10	9	8	7	6	5	4	3	2	1	0

← PPN → ← PPO →
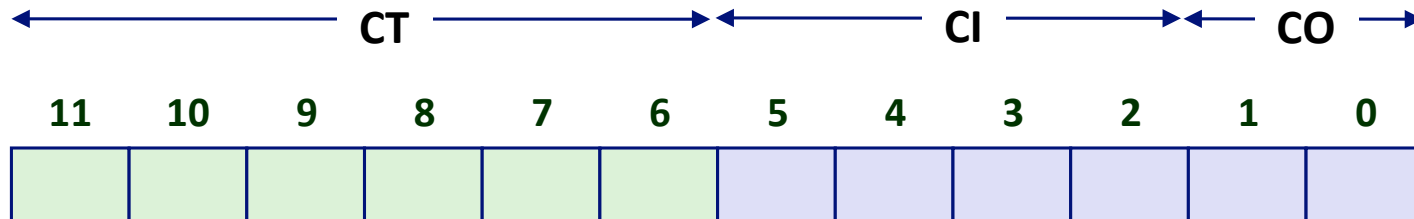
CO ___   CI ___   CT ____   Hit? __   Byte: ____

Address Translation Example #3

Virtual Address: 0x0020

TLBT ←——————————→ | TLBI →

13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0

←——————— VPN ———————→ | ←——————— VPO ———————→
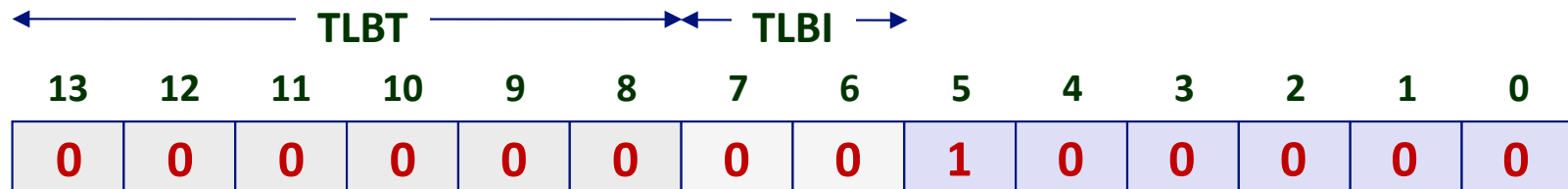
VPN **0x00**   TLBI **0**   TLBT **0x00**   TLB Hit? **N**   Page Fault? **N**   PPN: **0x28**

Physical Address

←—————————— CT ——————————→ | ←——— CI ———→ | ← CO →

11	10	9	8	7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	0	0	0	0	0

←——————— PPN ———————→ | ←——————— PPO ———————→
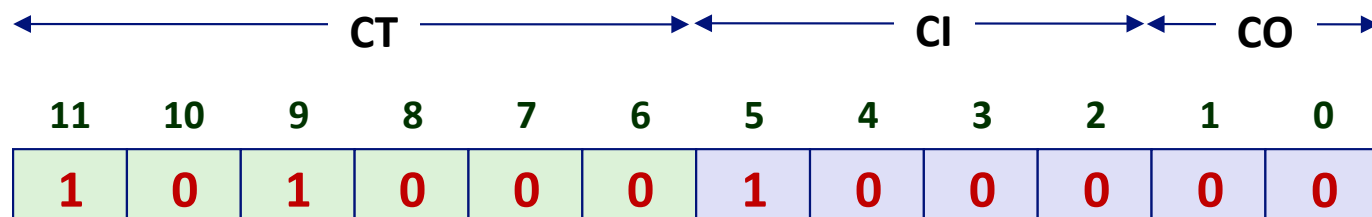
CO **0**   CI **0x8**   CT **0x28**   Hit? **N**   Byte: **Mem**