Computer Systems

CSE 410 Autumn 2013

12 – Virtual Memory

Roadmap

C:

```
car *c = malloc(sizeof(car));
c->miles = 100;
c->gals = 17;
float mpg = get_mpg(c);
free(c);
```

Java:

```
Car c = new Car();
c.setMiles(100);
c.setGals(17);
float mpg =
    c.getMPG();
```

Assembly language:

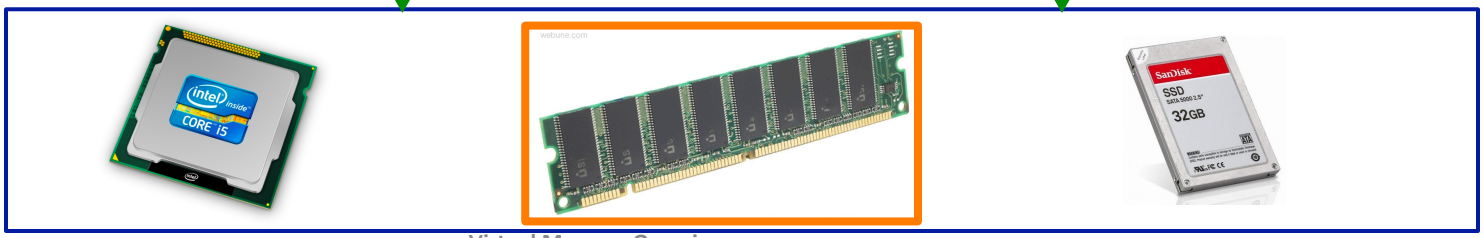
```
get_mpg:
    pushq   %rbp
    movq    %rsp, %rbp
    ...
    popq    %rbp
    ret
```

Machine code:

```
0111010000011000
100011010000010000000010
1000100111000010
11000001111110100011111
```

OS:

Computer system:



Memory & data
Integers & floats
Machine code & C
x86 assembly
Procedures & stacks
Arrays & structs
Memory & caches
Processes
Virtual memory
Memory allocation
Java vs. C

Virtual Memory (VM)

- Overview and motivation
- Indirection
- VM as a tool for caching
- Memory management/protection and address translation
- Virtual memory example

Processes

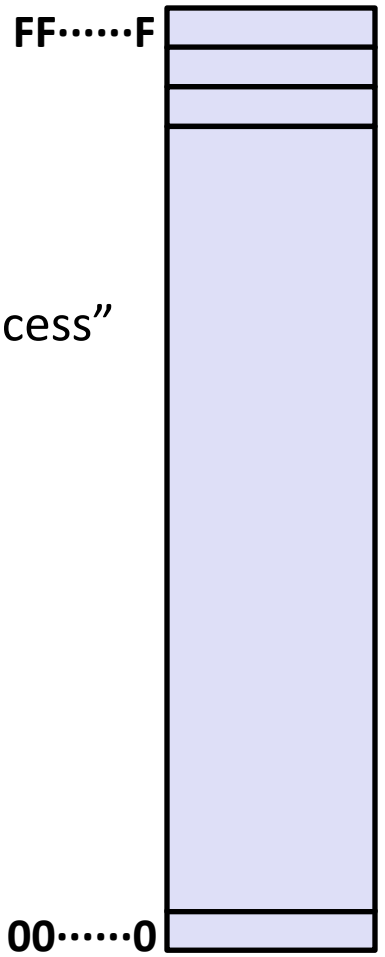
- **Definition: A *process* is an instance of a running program**
  - One of the most important ideas in computer science
  - Not the same as "program" or "processor"

- **Process provides each program with two key abstractions:**
  - Logical control flow
    - Each process seems to have exclusive use of the CPU
  - Private virtual address space
    - Each process seems to have exclusive use of main memory

- **How are these illusions maintained?**
  - Process executions interleaved (multi-tasking) – last section
  - Address spaces managed by virtual memory system – this section!

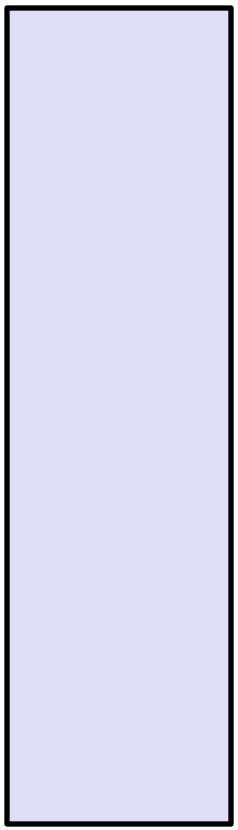
Virtual Memory (Previous Lectures)

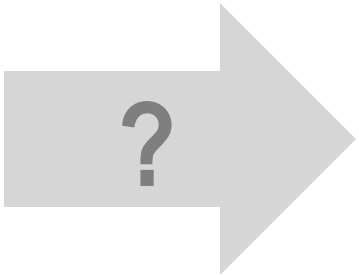
- **Programs refer to virtual memory addresses**
  - `movl (%ecx),%eax`
  - Conceptually memory is just a very large array of bytes
  - Each byte has its own address
  - System provides address space private to particular "process"
- **Allocation: Compiler and run-time system**
  - Where different program objects should be stored
  - All allocation within single virtual address space
- ***What problems does virtual memory solve?***



Problem 1: How Does Everything Fit?

64-bit addresses:
16 Exabyte

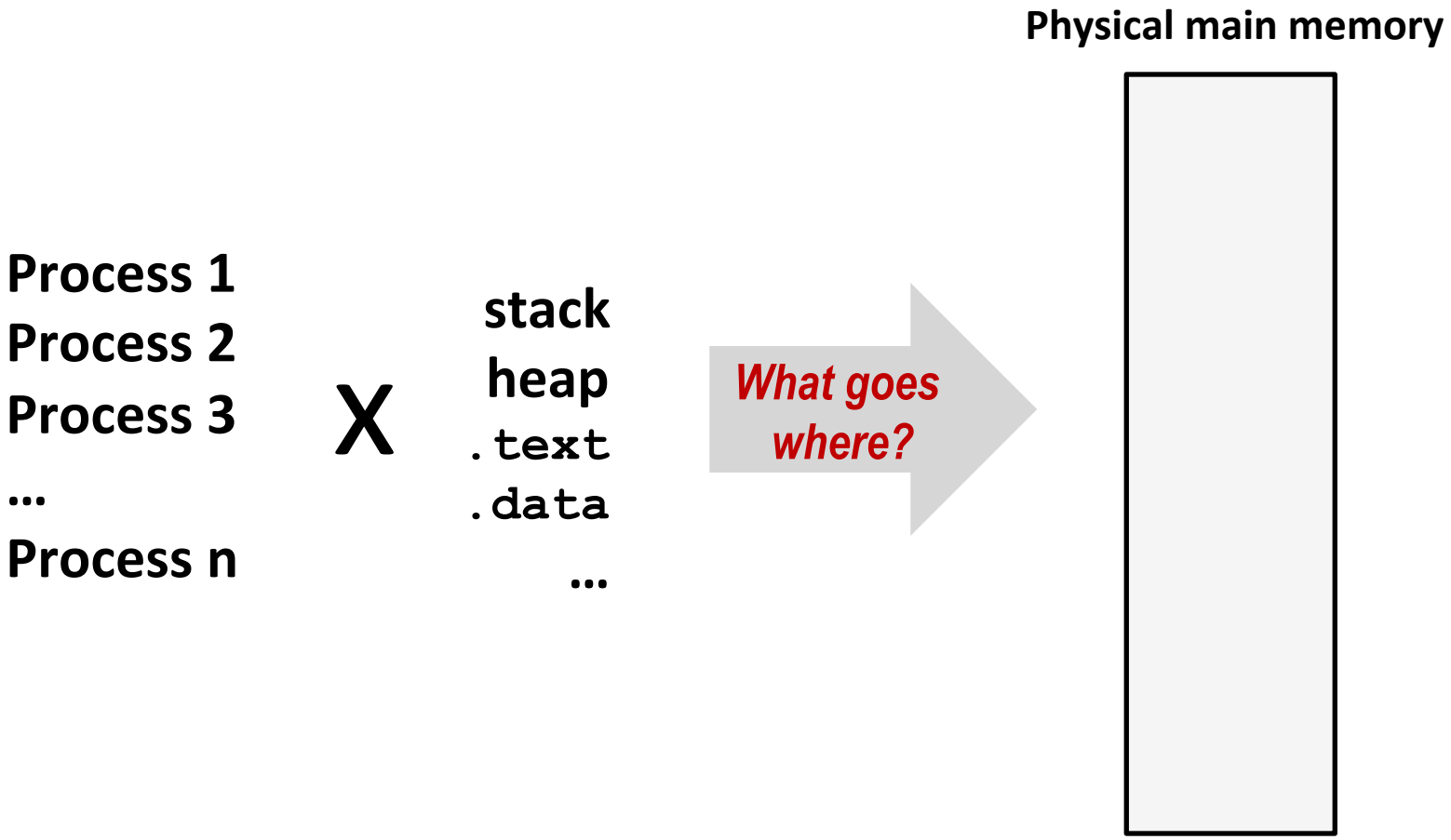




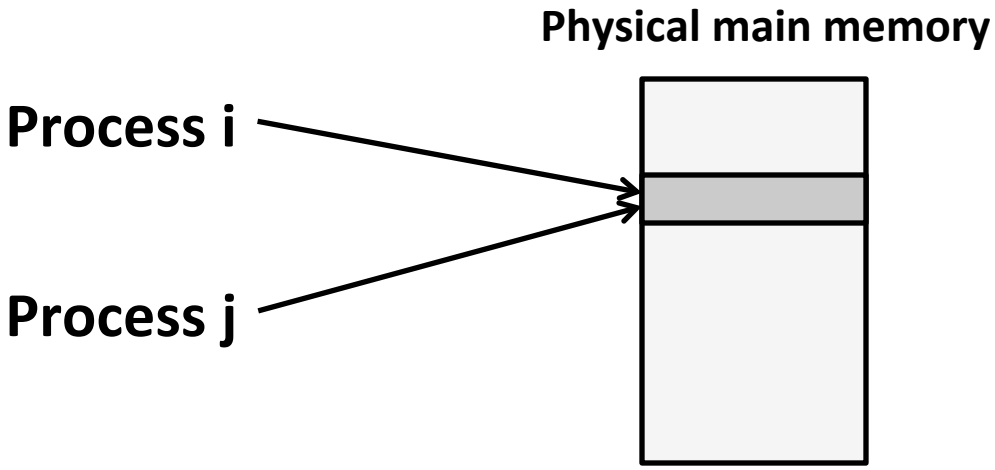
Physical main memory:
Few Gigabytes

And there are many processes ….

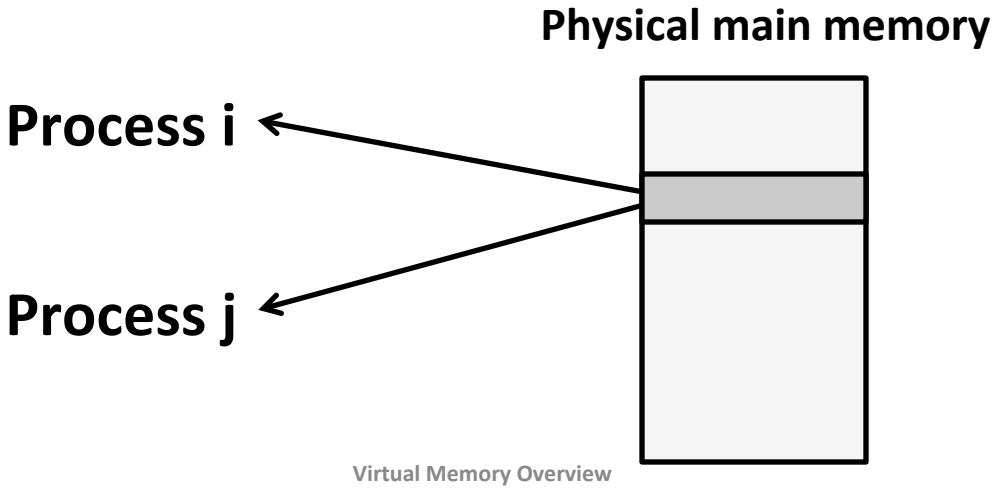
Problem 2: Memory Management



Problem 3: How To Protect



Problem 4: How To Share?



Virtual Memory (VM)

- Overview and motivation
- Indirection
- VM as a tool for caching
- Memory management/protection and address translation
- Virtual memory example

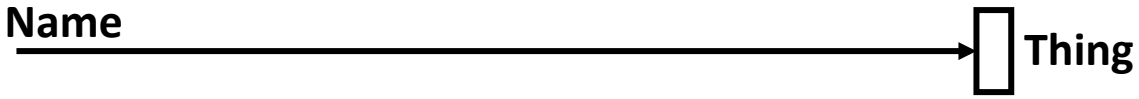
How would you solve those problems?

- Fitting a huge memory into a tiny physical memory
- Managing the memory spaces of multiple processes
- Protecting processing from stepping on each other's memory
- Allowing processes to share common parts of memory

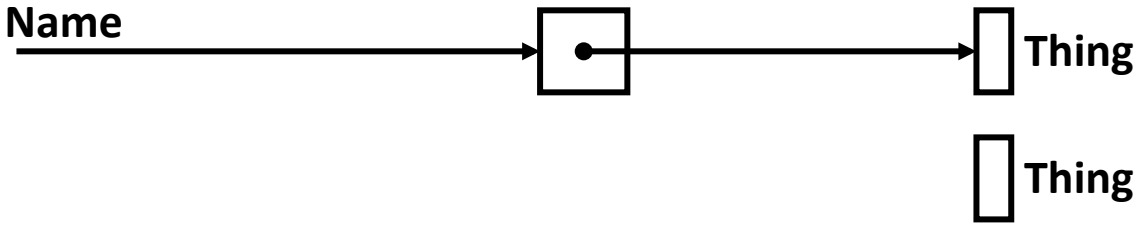
Indirection

- "Any problem in computer science can be solved by adding another level of indirection"

- Without Indirection



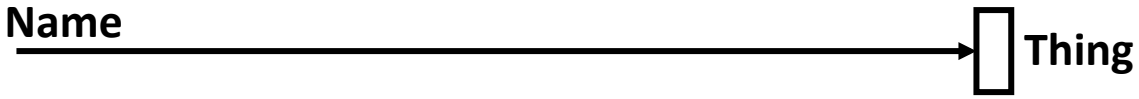
- With Indirection



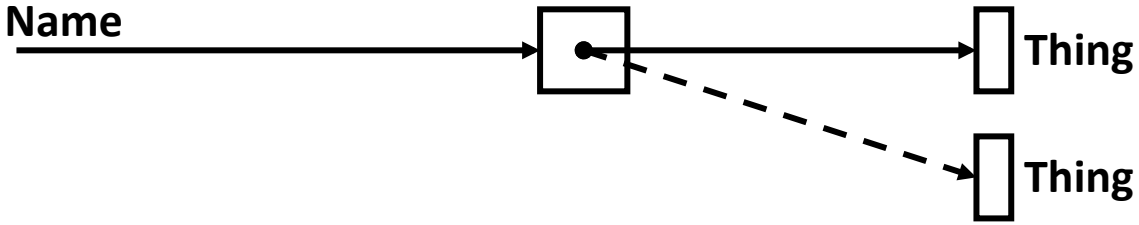
Indirection

- Indirection: the ability to reference something using a name, reference, or container instead the value itself. A flexible mapping between a name and a thing allows changing the thing without notifying holders of the name.

- Without Indirection

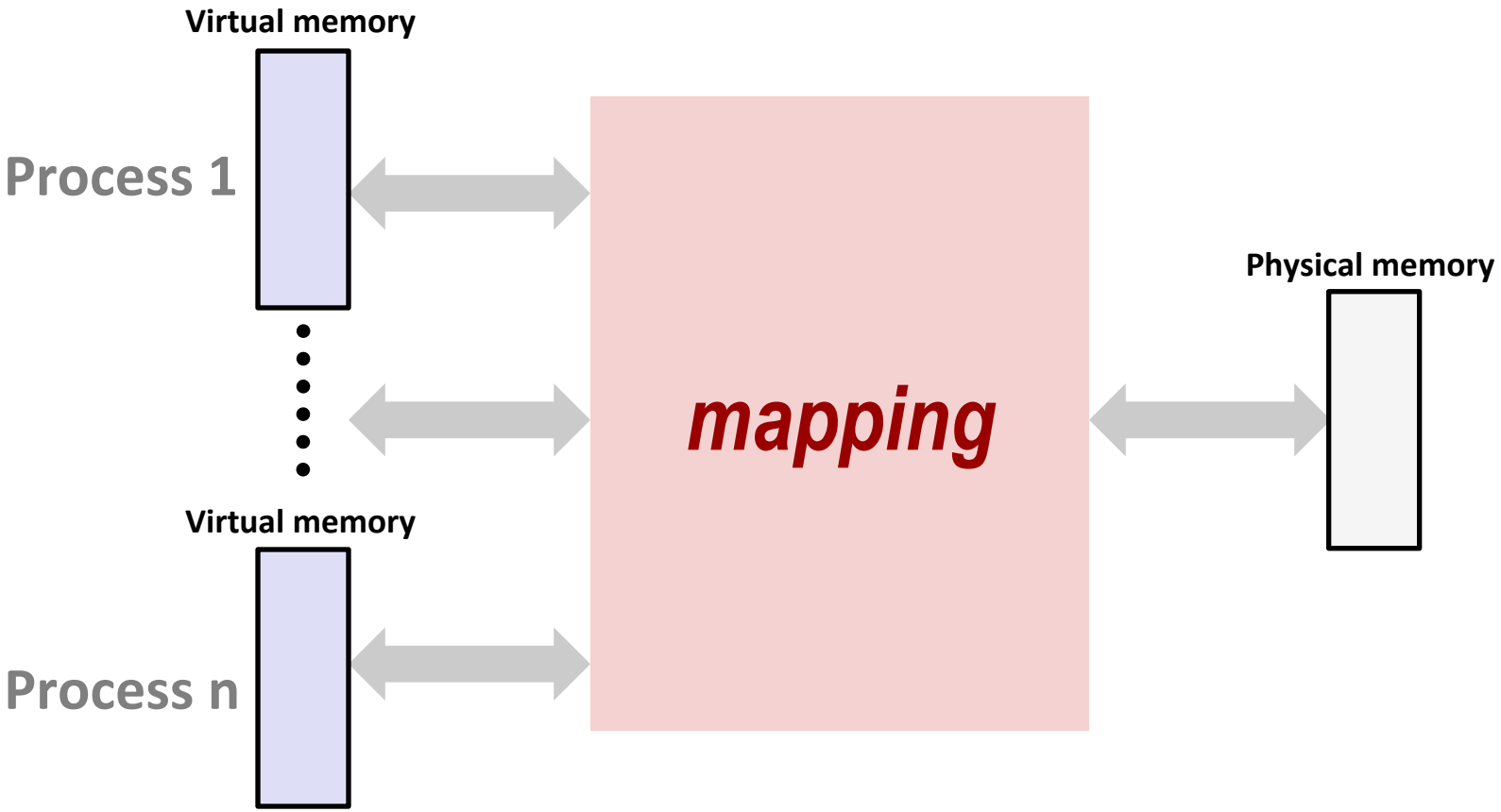


- With Indirection



- Examples:
  Domain Name Service (DNS) name->IP address, phone system (e.g., cell phone number portability), snail mail (e.g., mail forwarding), 911 (routed to local office), DHCP, call centers that route calls to available operators, etc.

Solution: Level Of Indirection



- Each process gets its own private virtual address space
- Solves the previous problems

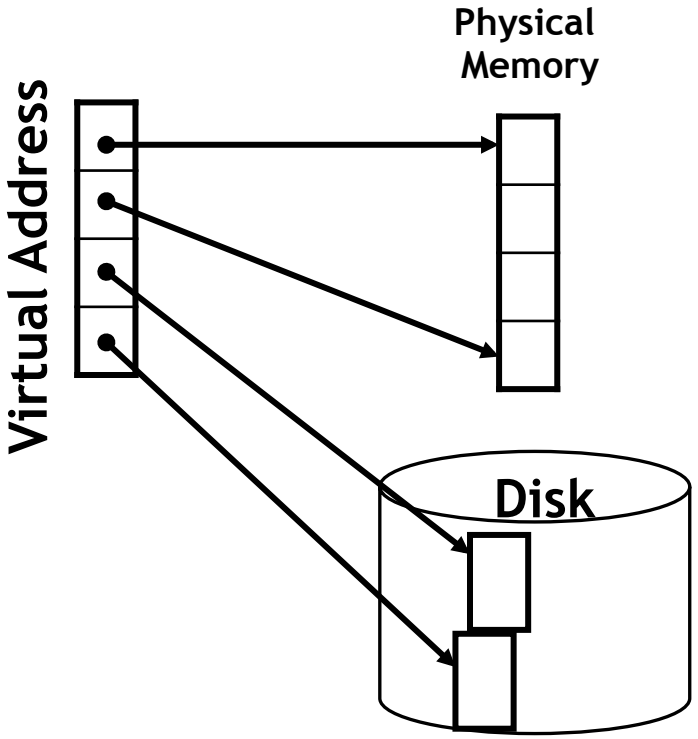
Address Spaces

- **Virtual address space:** Set of $N = 2^n$ virtual addresses
  {0, 1, 2, 3, …, N-1}

- **Physical address space:** Set of $M = 2^m$ physical addresses (n > m)
  {0, 1, 2, 3, …, M-1}

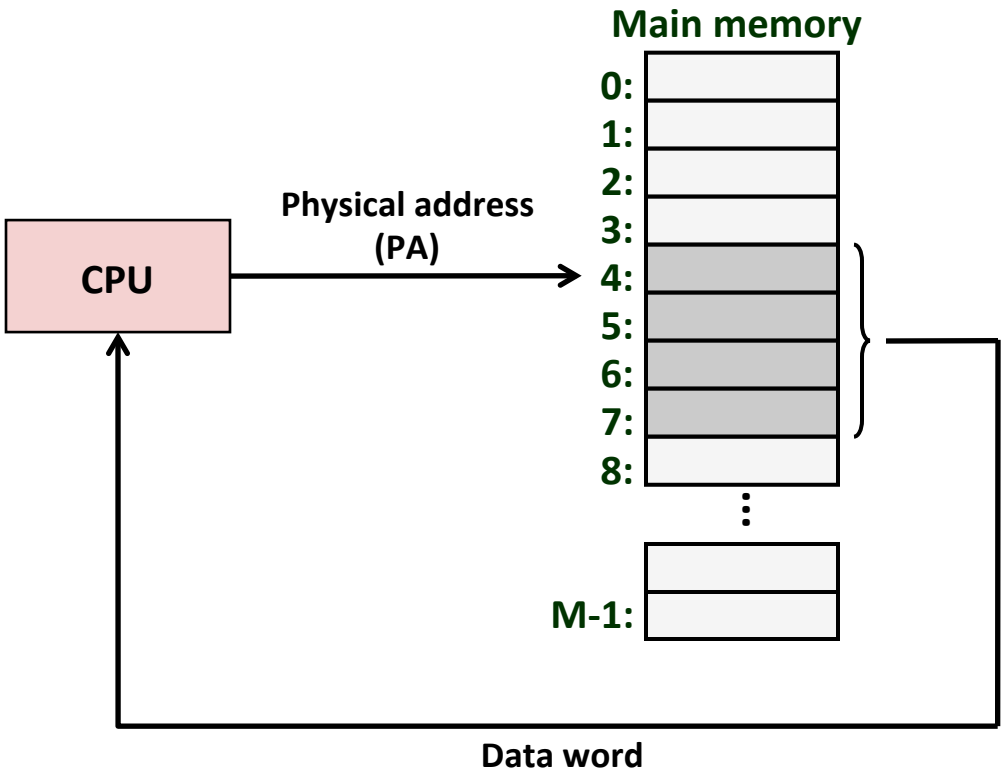
- **Every byte in main memory:**
  one physical address; zero, one, or more virtual addresses

Mapping



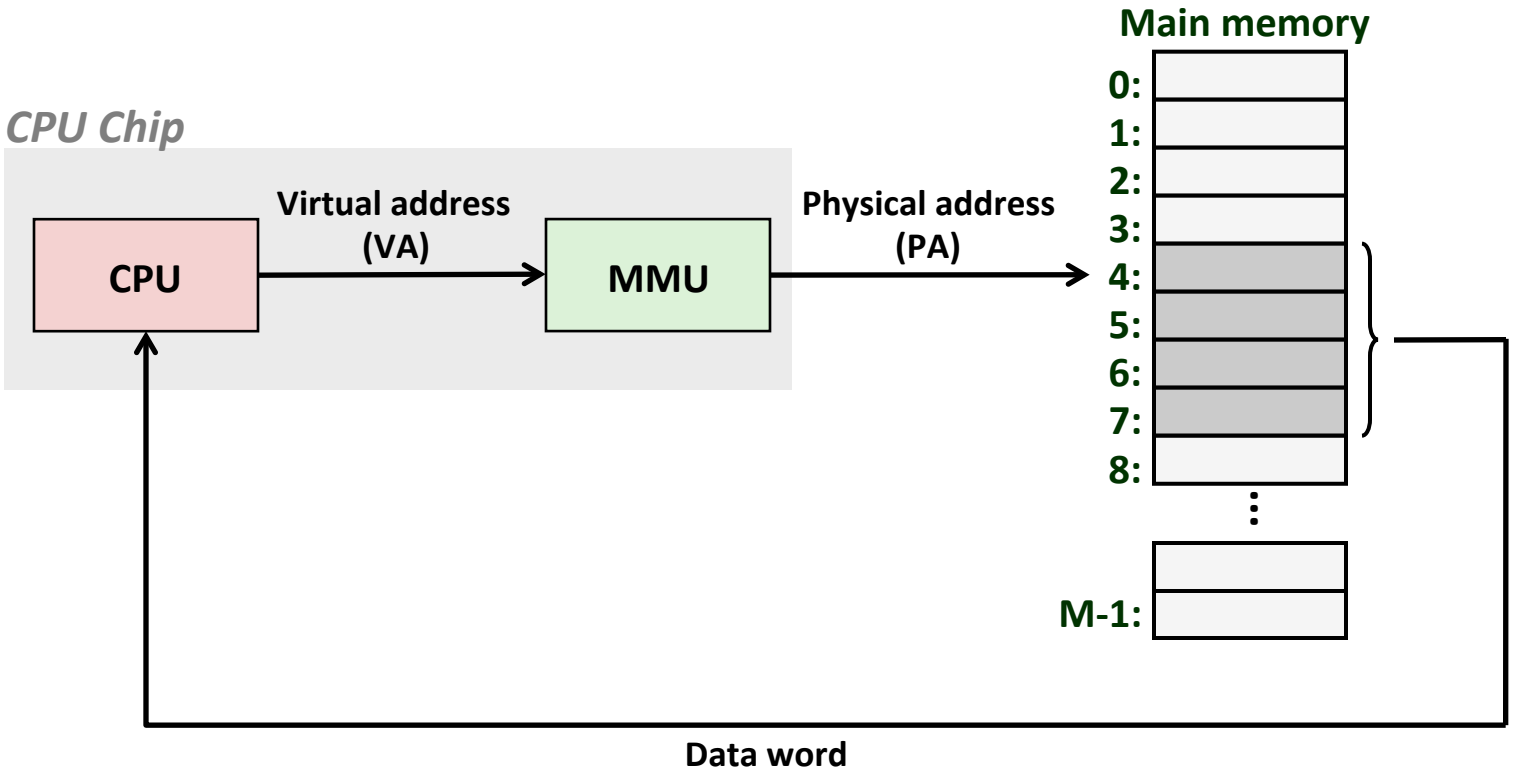
A virtual address can be mapped to either physical memory or disk.

A System Using Physical Addressing



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

A System Using Virtual Addressing



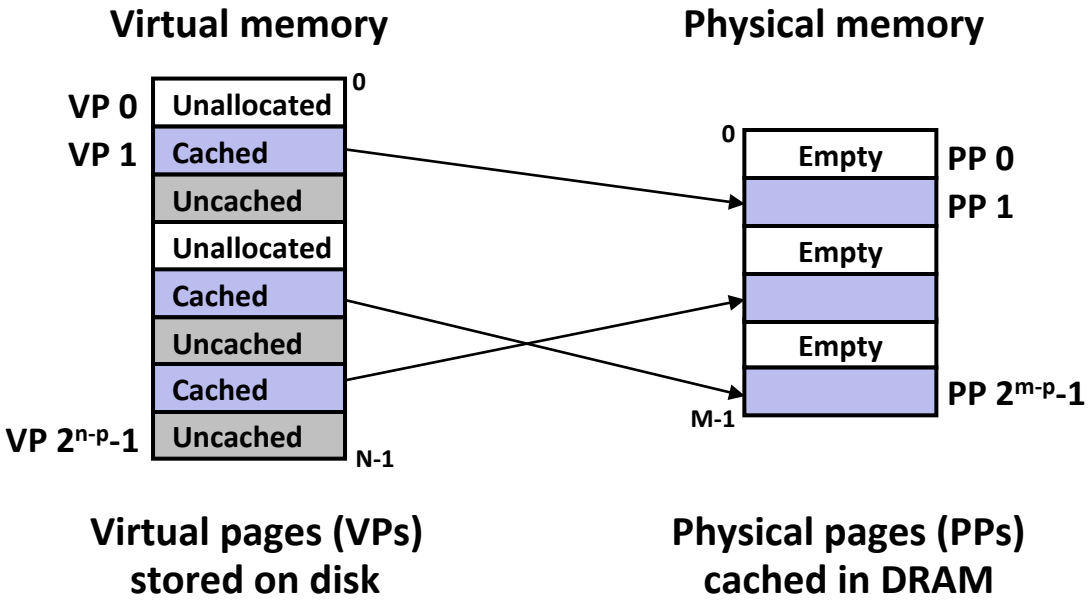
- Used in all modern desktops, laptops, servers
- One of the great ideas in computer science

Virtual Memory (VM)

- Overview and motivation
- Indirection
- VM as a tool for caching
- Memory management/protection and address translation
- Virtual memory example

VM and the Memory Hierarchy

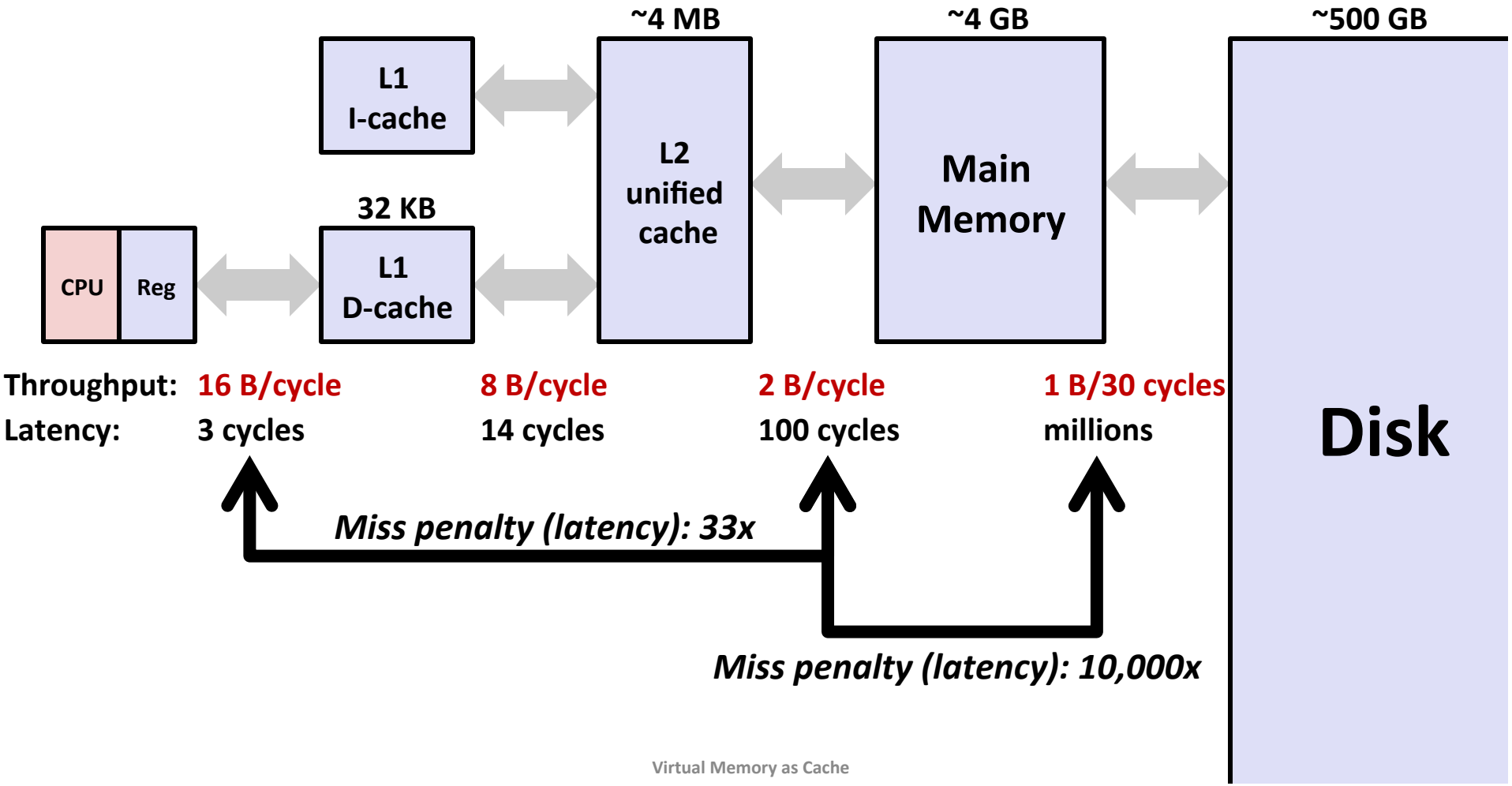
- **Think of virtual memory as an array of N = 2^n contiguous bytes stored *on a disk***
- **Then physical main memory (DRAM) is used as a *cache* for the virtual memory array**
  - The cache blocks are called *pages* (size is P = 2^p bytes)



Memory Hierarchy: Core 2 Duo

Not drawn to scale

L1/L2 cache: 64 B blocks



DRAM Cache Organization

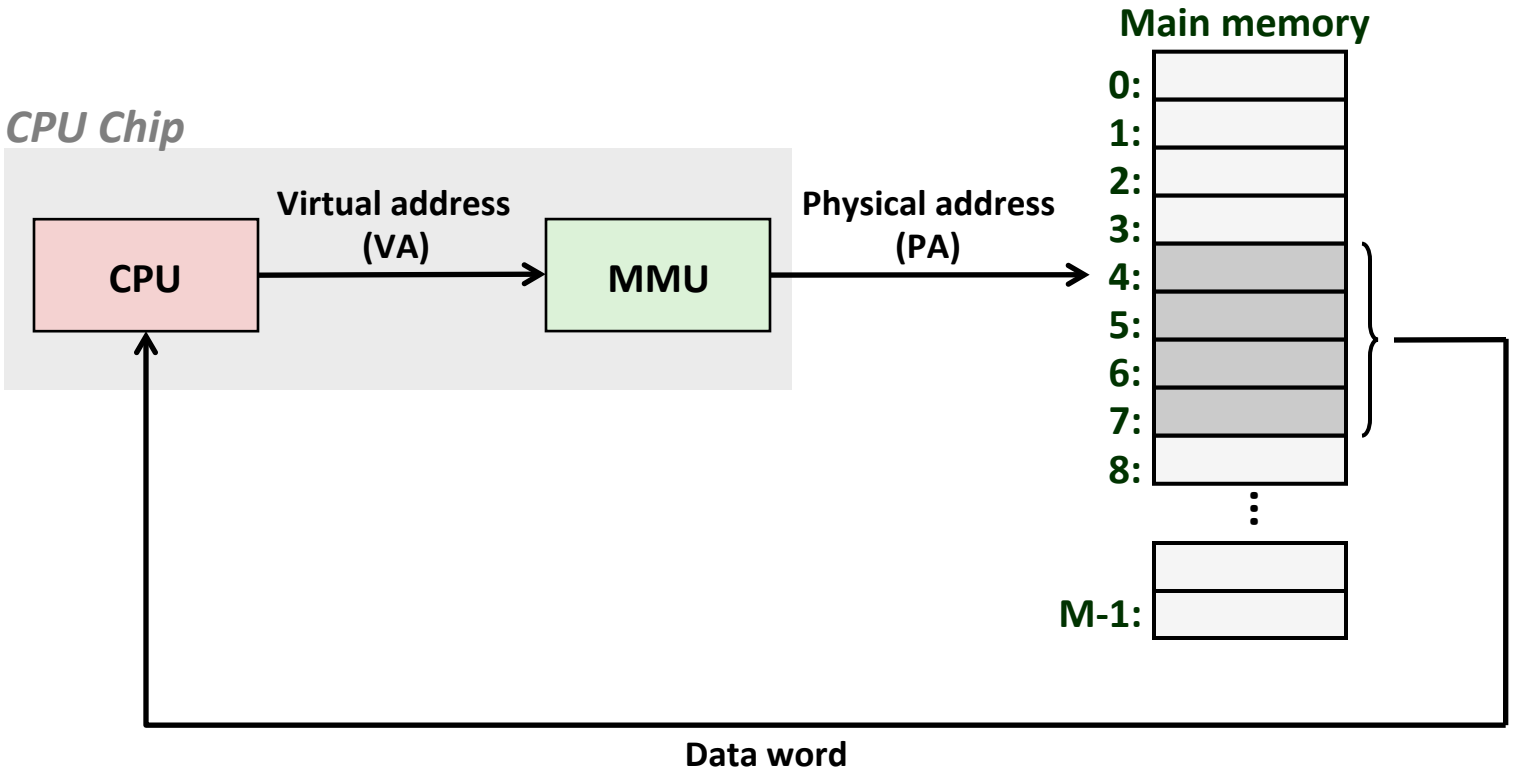
- **DRAM cache organization driven by the enormous miss penalty**
  - DRAM is about 10x slower than SRAM
  - Disk is about 10,000x slower than DRAM
    - (for first byte; faster for next byte)
- **Consequences?**
  - Block size?
  - Associativity?
  - Write-through or write-back?

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

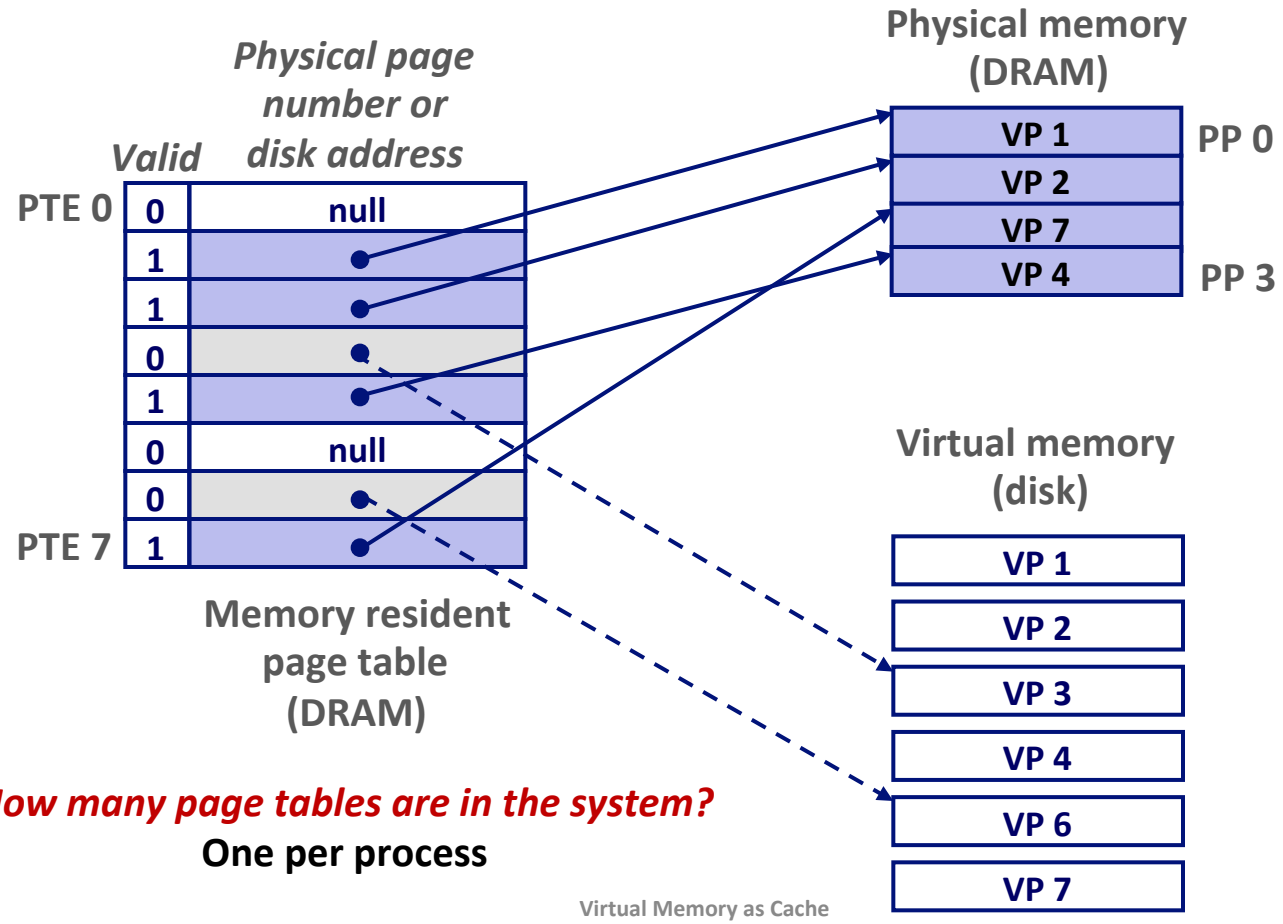
Indexing into the "DRAM Cache"



How do we perform the VA -> PA translation?

Address Translation: Page Tables

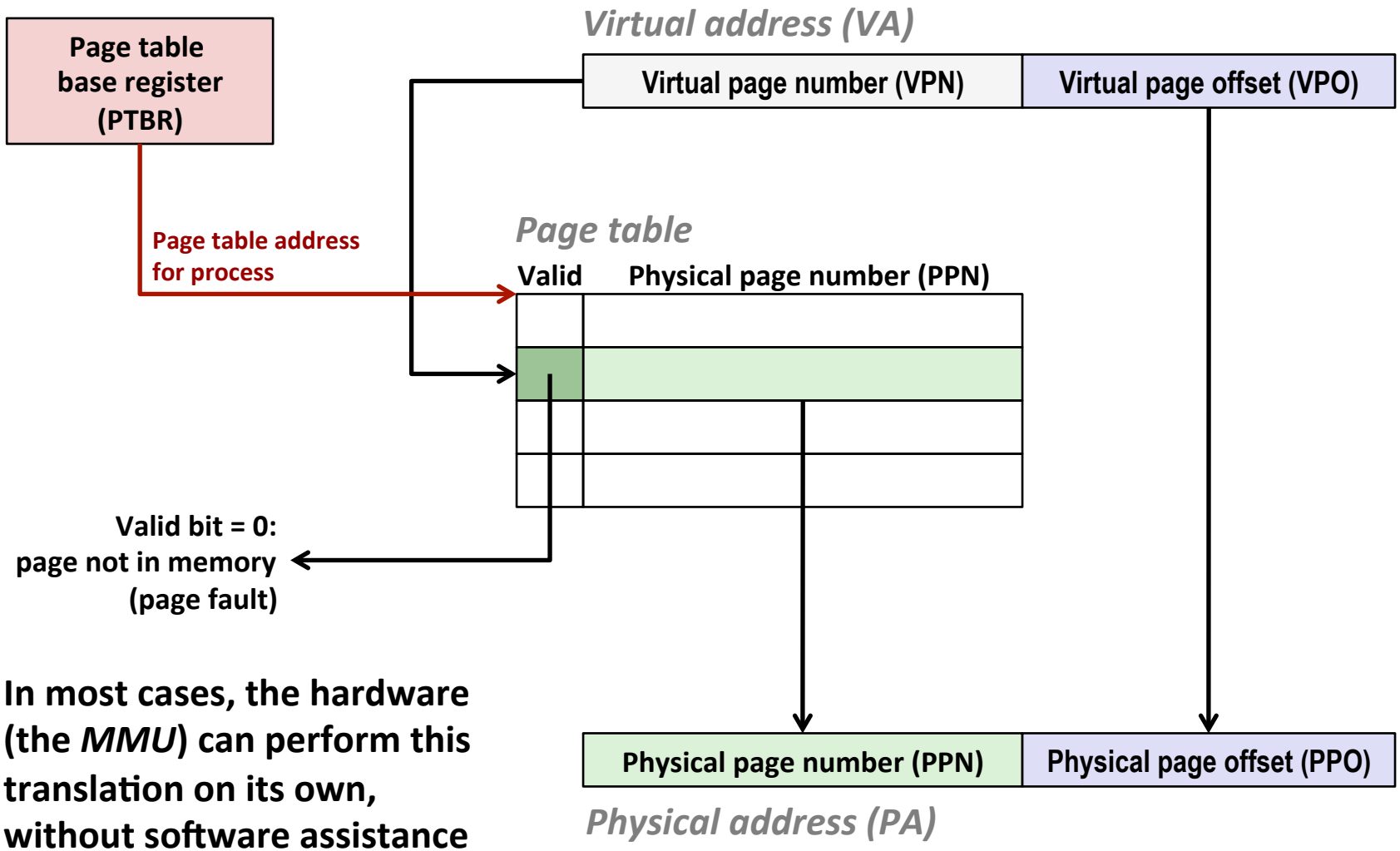
- A *page table* (PT) is an array of *page table entries* (PTEs) that maps virtual pages to physical pages.



How many page tables are in the system?

One per process

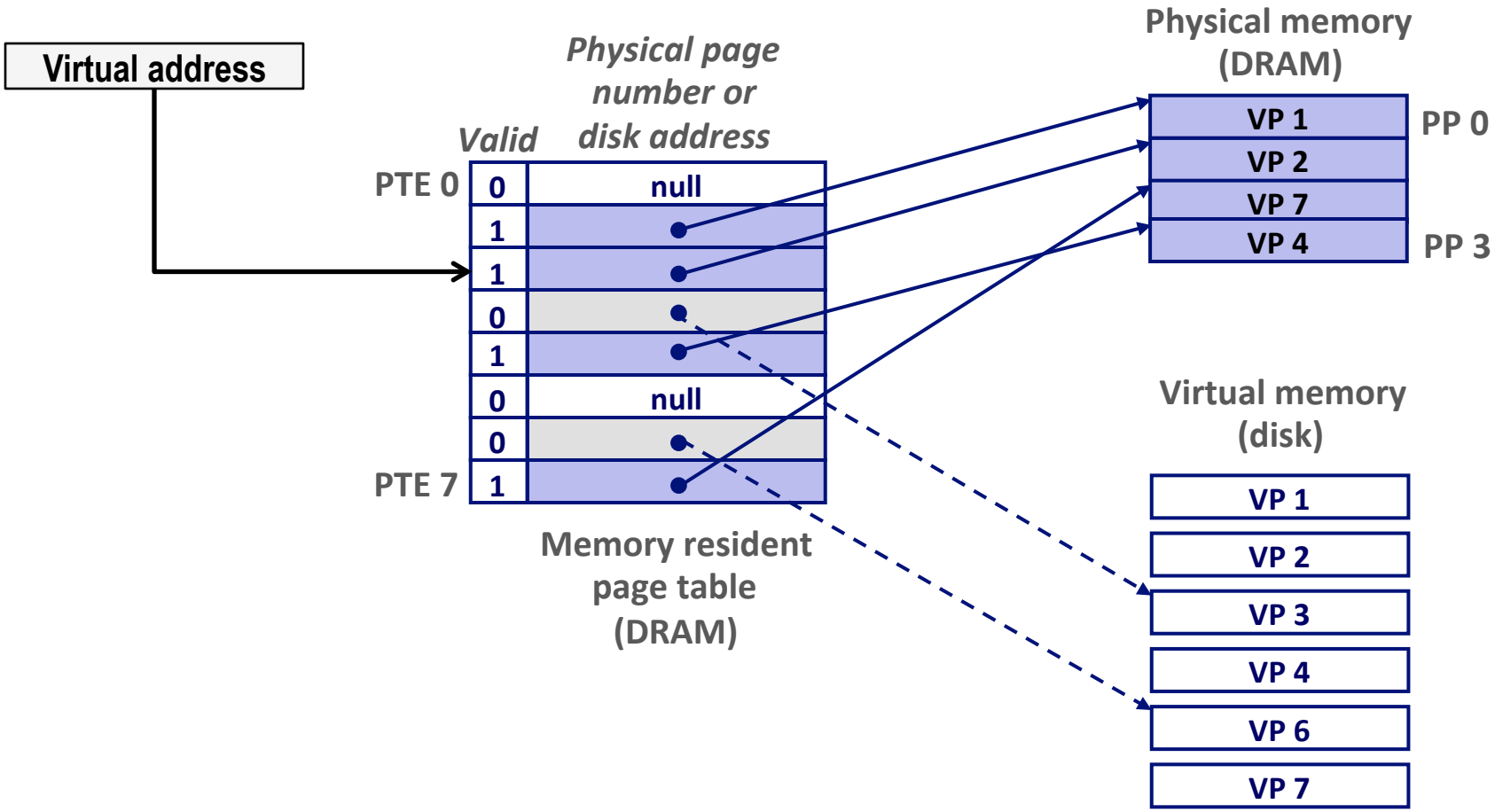
Address Translation With a Page Table



In most cases, the hardware (the *MMU*) can perform this translation on its own, without software assistance

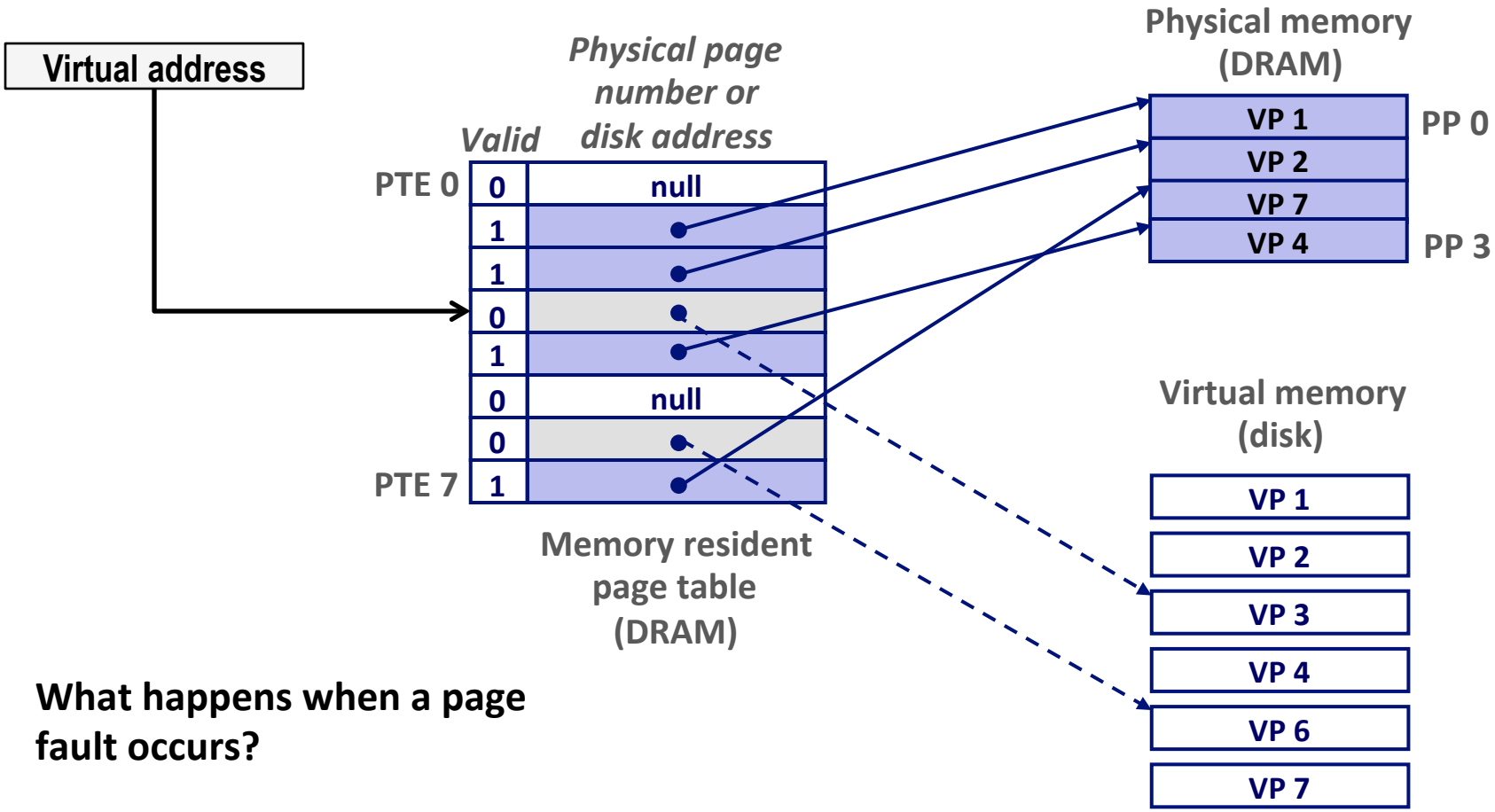
Page Hit

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



Page Fault

- ***Page fault:*** **reference to VM byte that is NOT in physical memory**



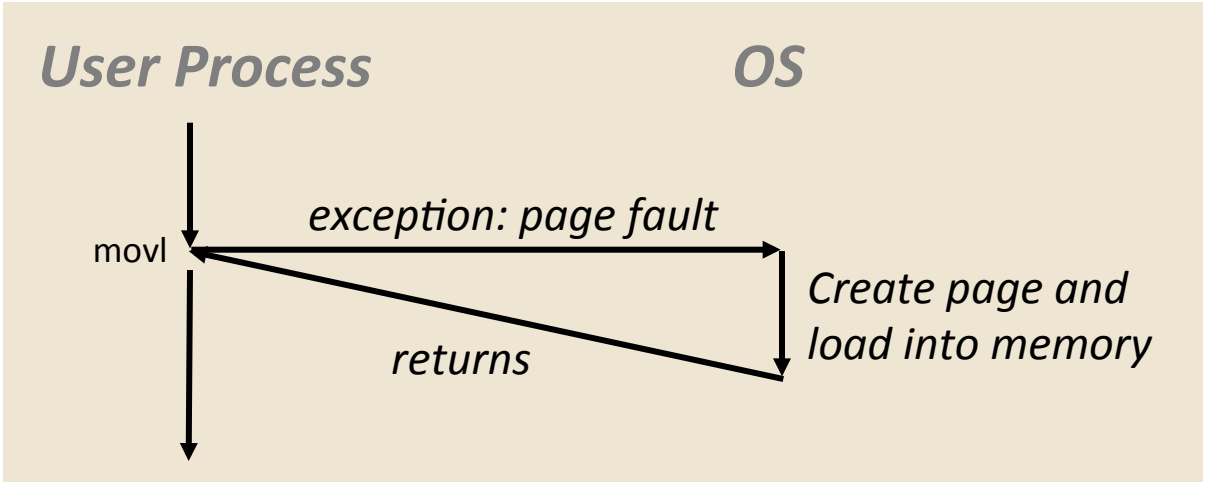
What happens when a page fault occurs?

Fault Example: Page Fault

- User writes to memory location
- That portion (page) of user's memory is currently on disk

```
int a[1000];
main ()
{
    a[500] = 13;
}
```

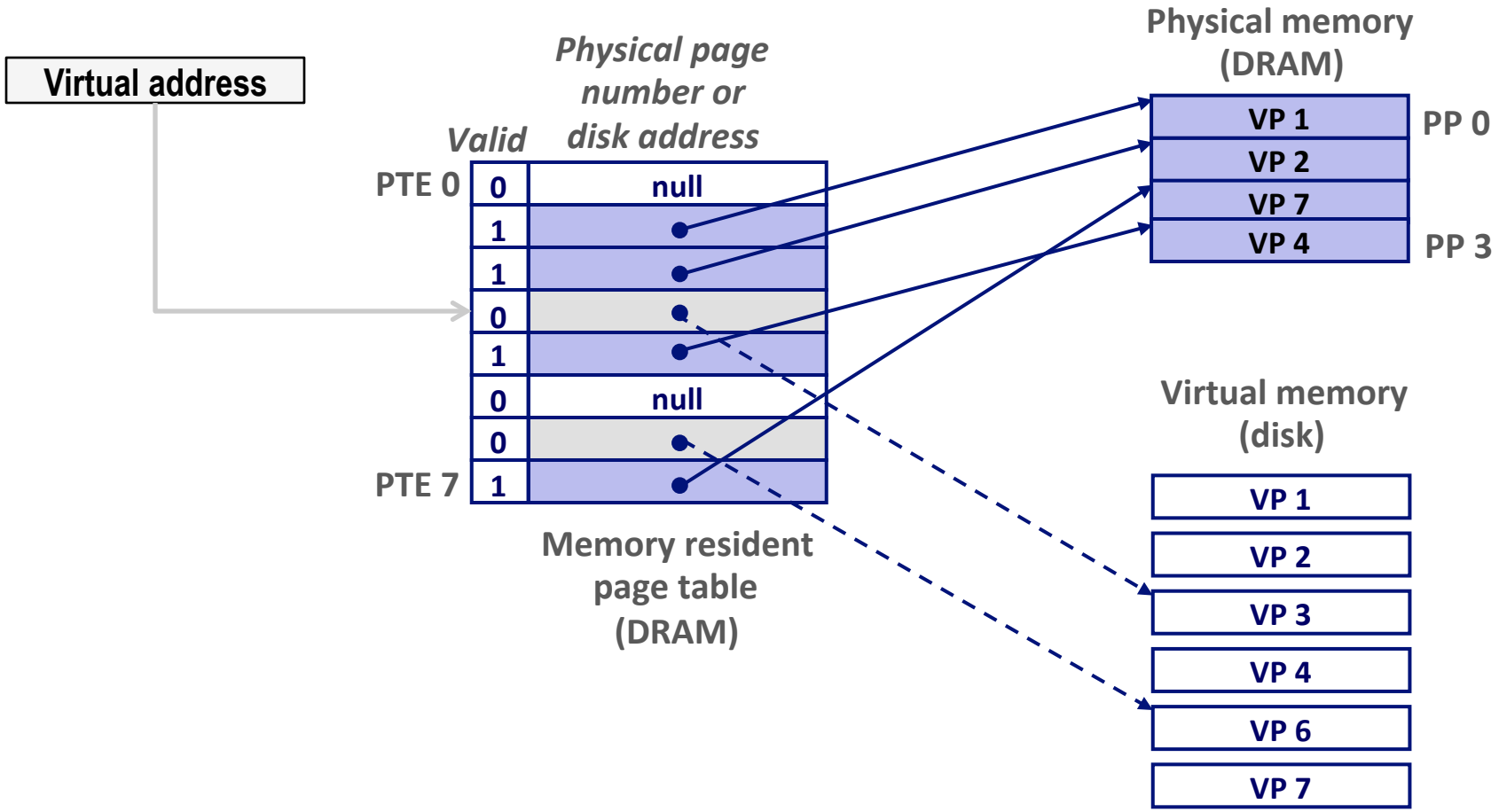
```
80483b7:      c7 05 10 9d 04 08 0d   movl   $0xd,0x8049d10
```



- Page handler must load page into physical memory
- Returns to faulting instruction: **`mov`** is executed again!
- Successful on second try

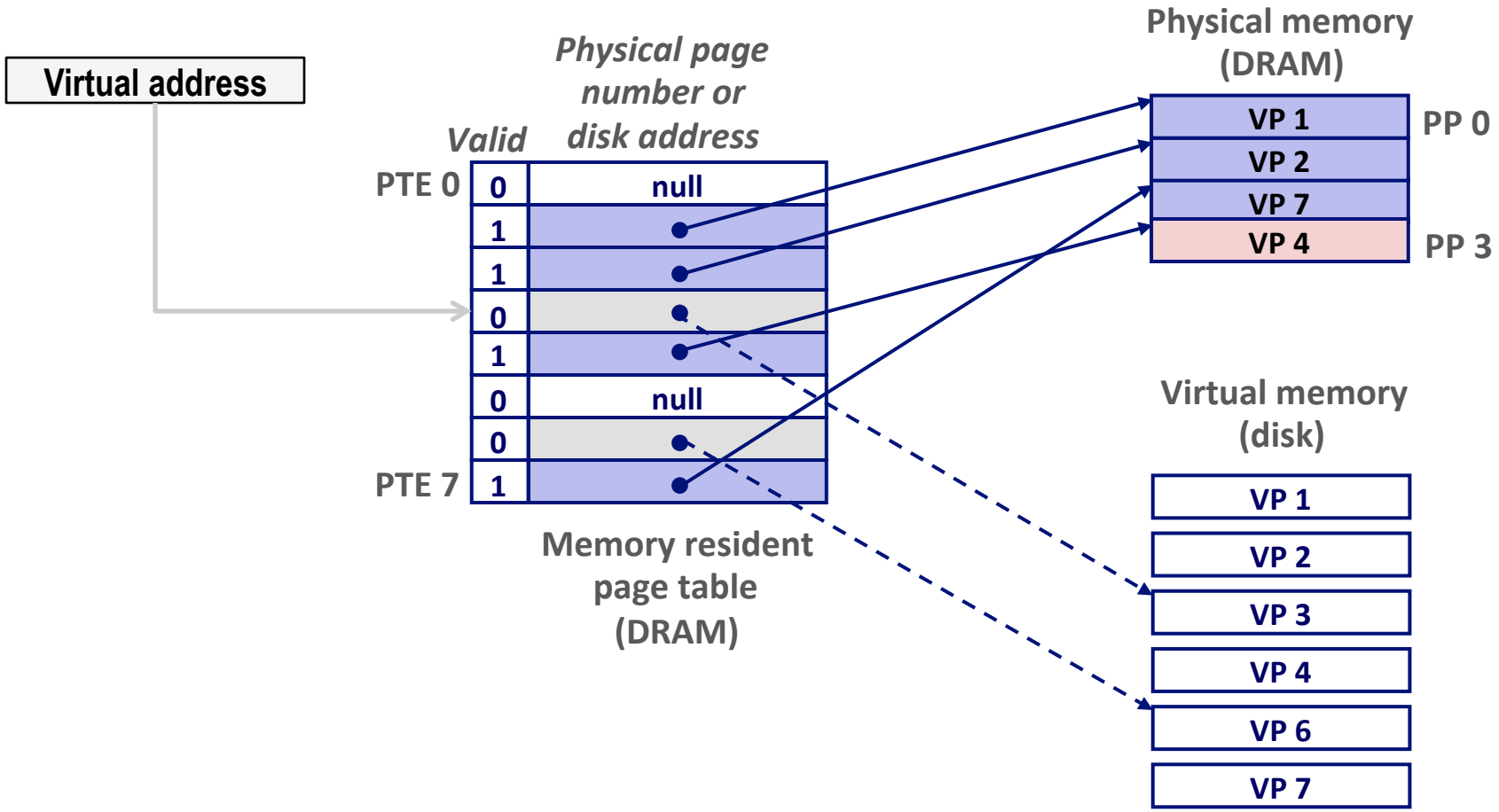
Handling Page Fault

- Page miss causes page fault (an exception)



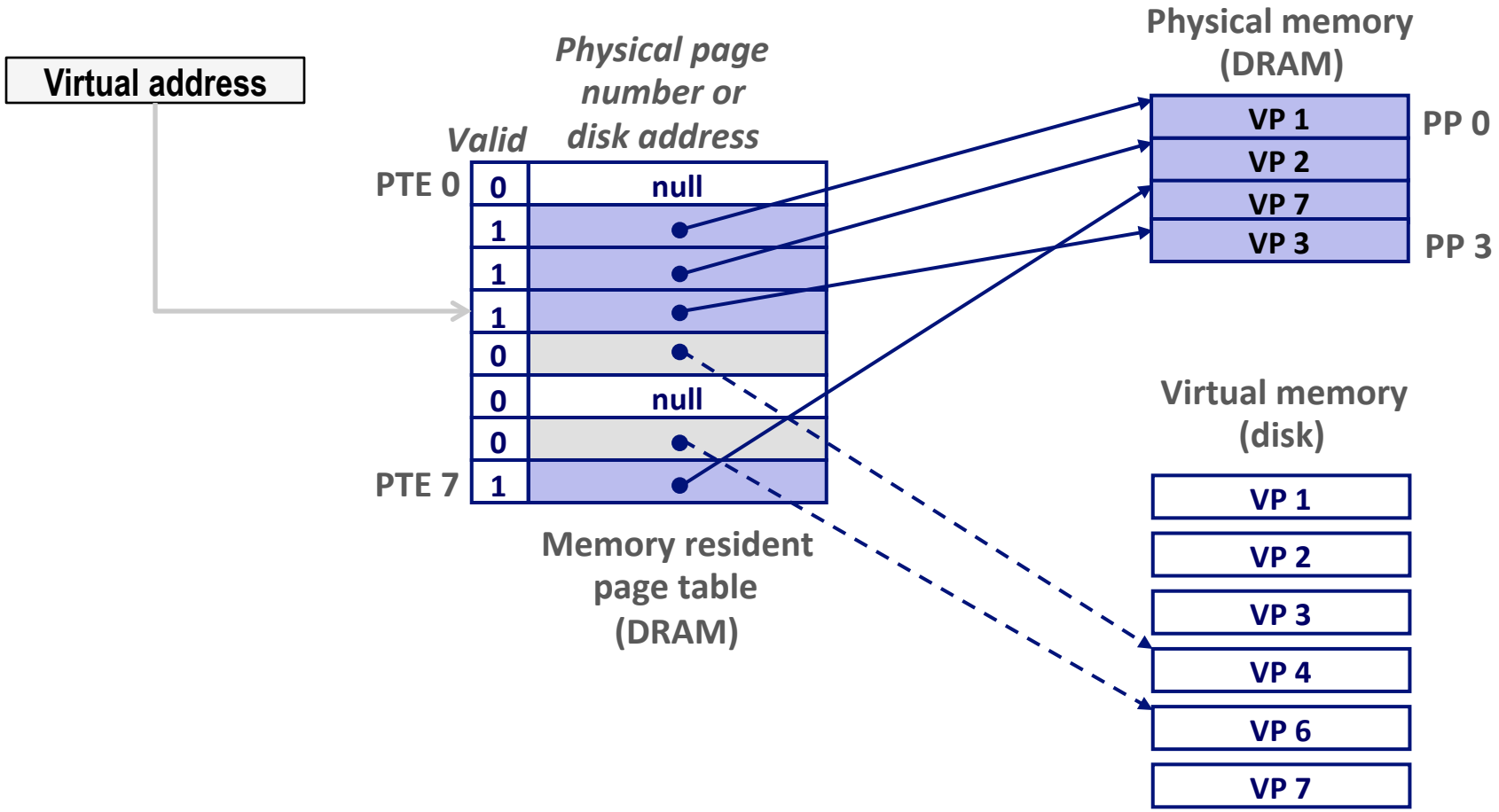
Handling Page Fault

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



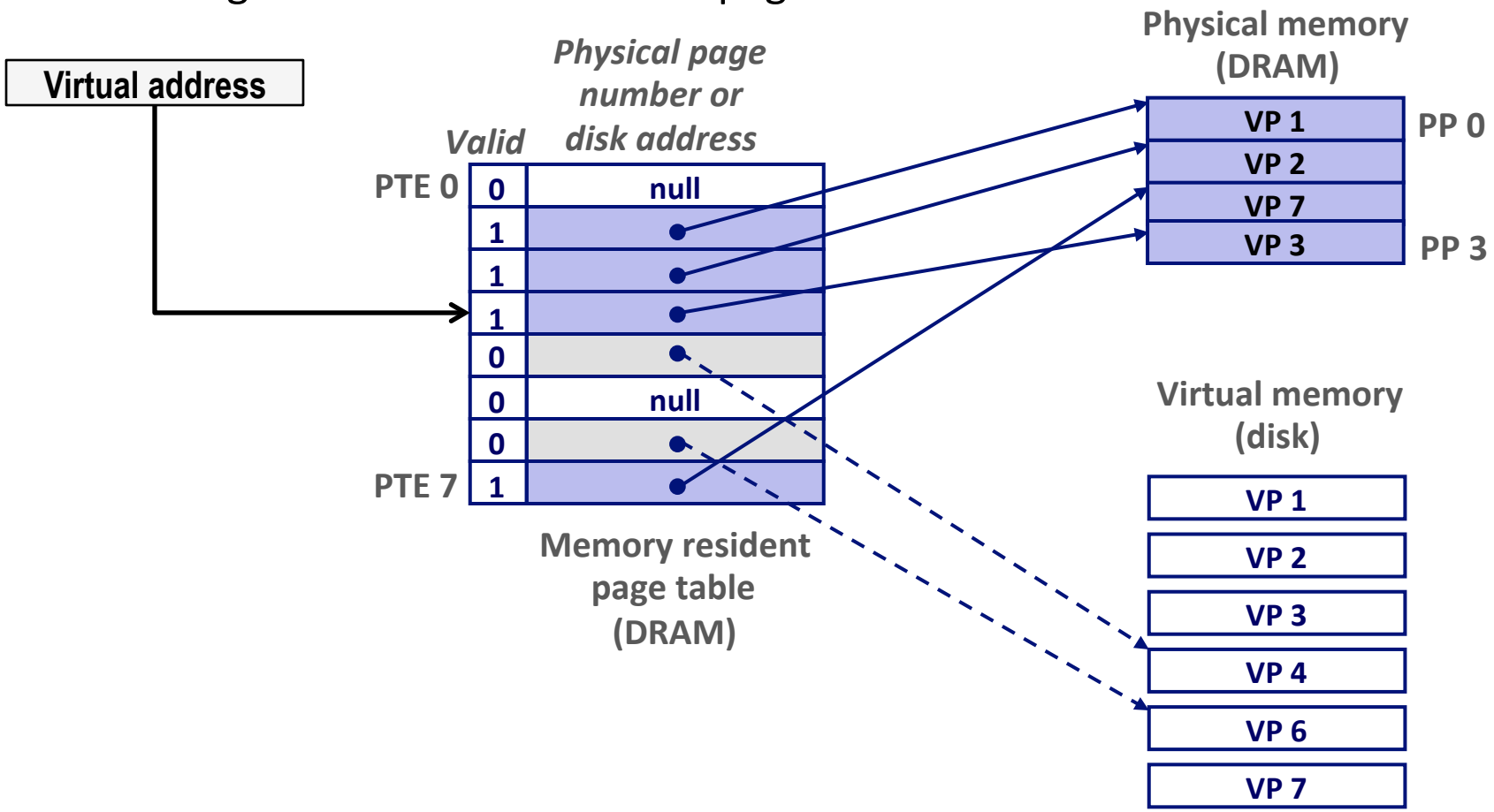
Handling Page Fault

- 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

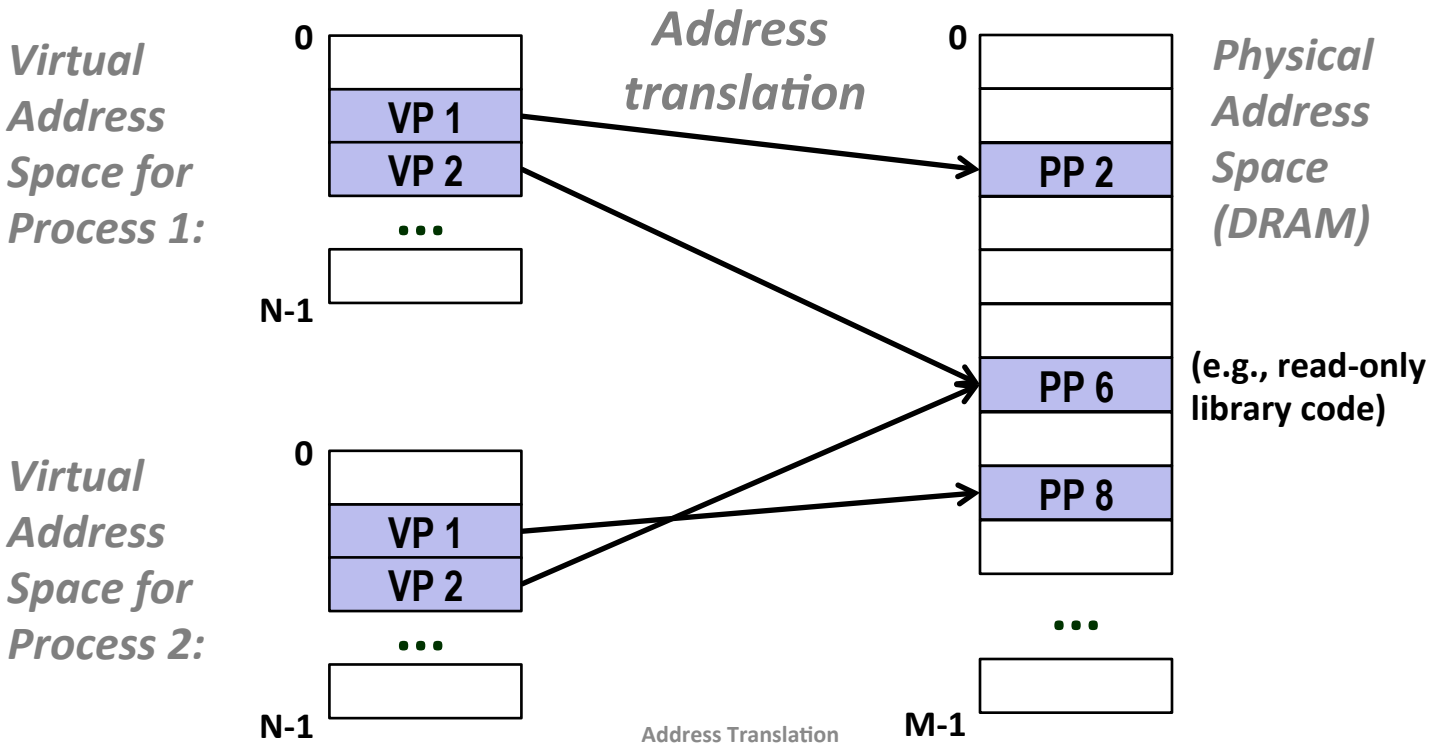
- **Virtual memory works well because of locality**
  - Same reason that L1 / L2 / L3 caches work
- **The set of virtual pages that a program is "actively" accessing at any point in time is called its *working set***
  - Programs with better temporal locality will have smaller working sets
- **If (working 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

Virtual Memory (VM)

- Overview and motivation
- Indirection
- VM as a tool for caching
- Memory management/protection and address translation
- Virtual memory example

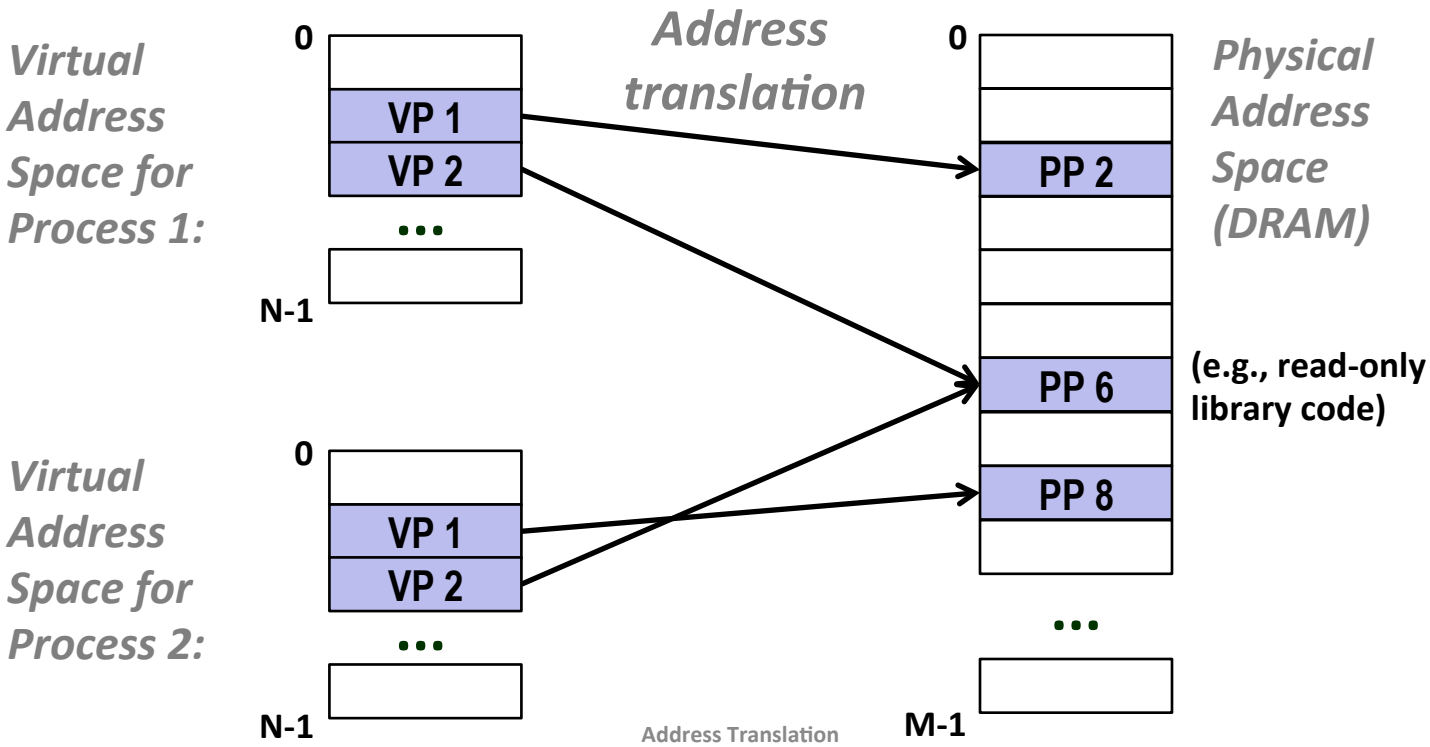
VM for Managing Multiple Processes

- **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!



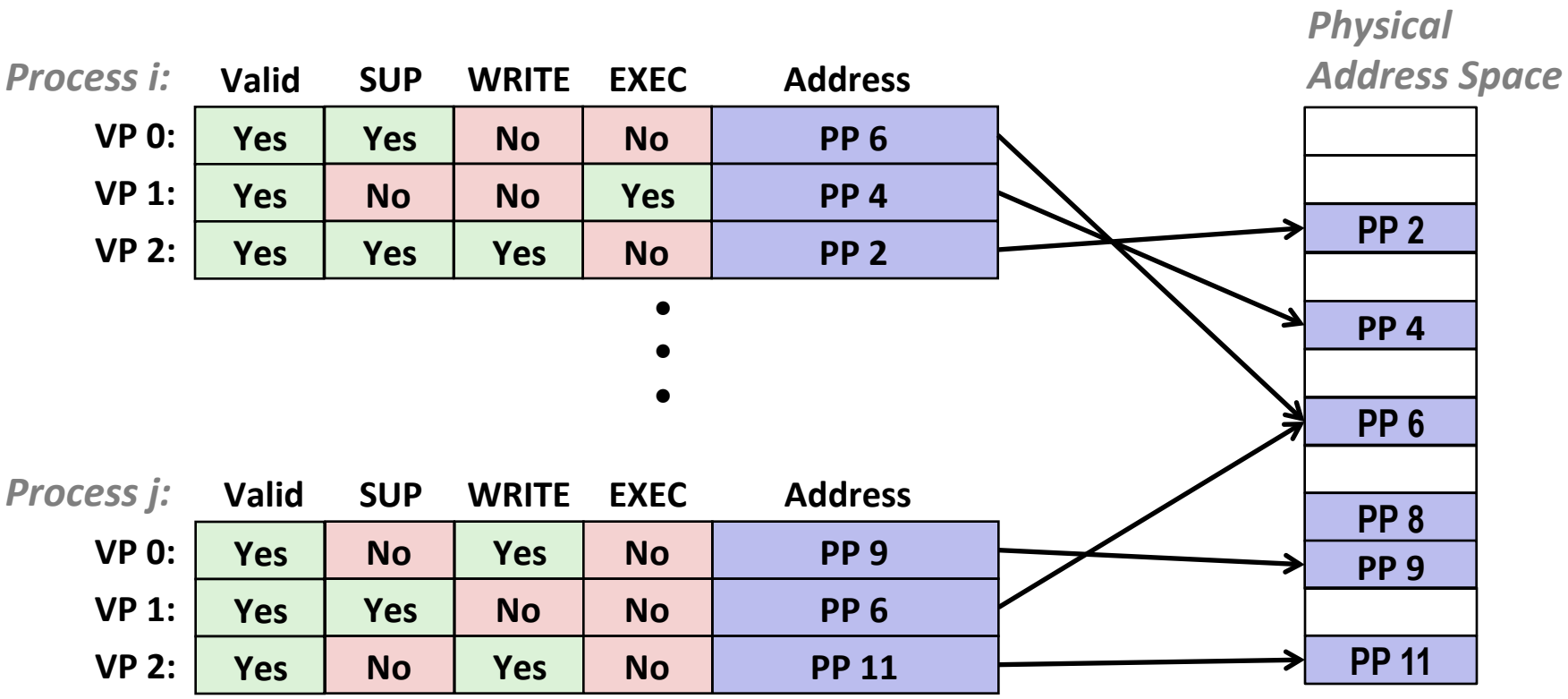
VM for Protection and Sharing

- **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)

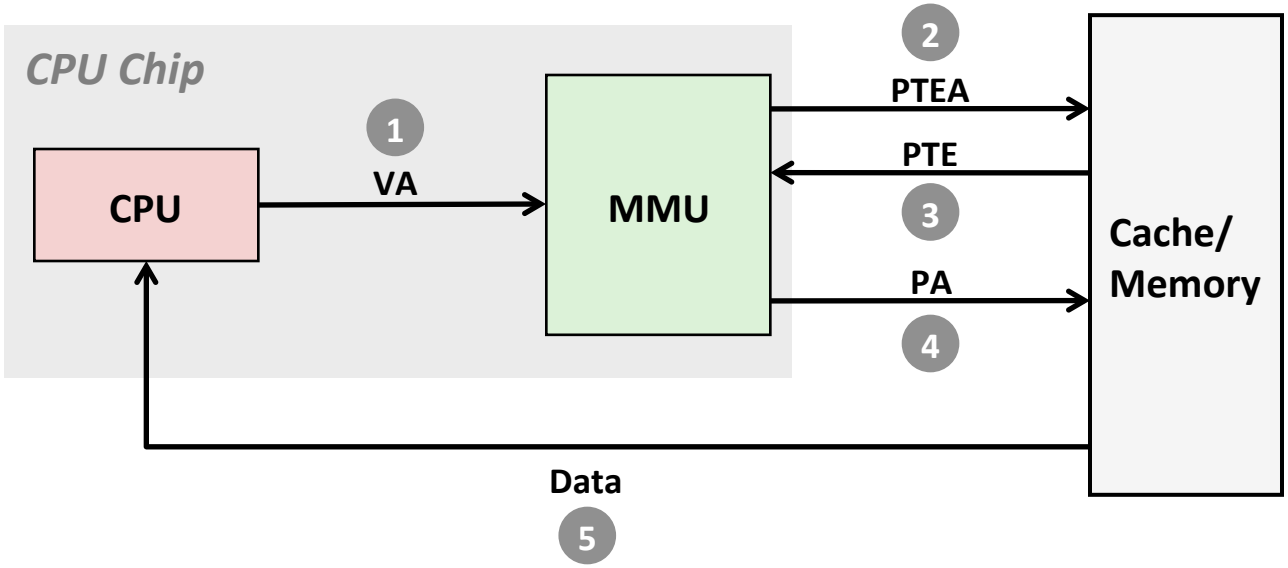


Memory Protection Within a Single Process

- 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



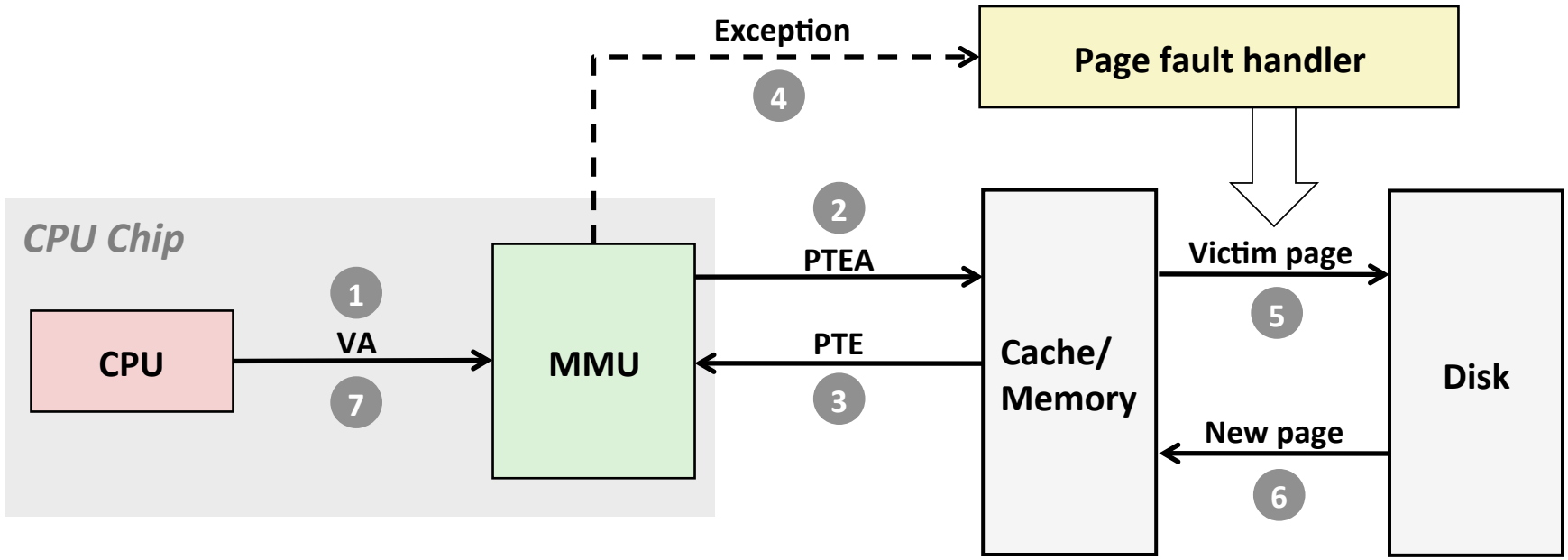
1) Processor sends virtual address to MMU (*memory management unit*)

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!

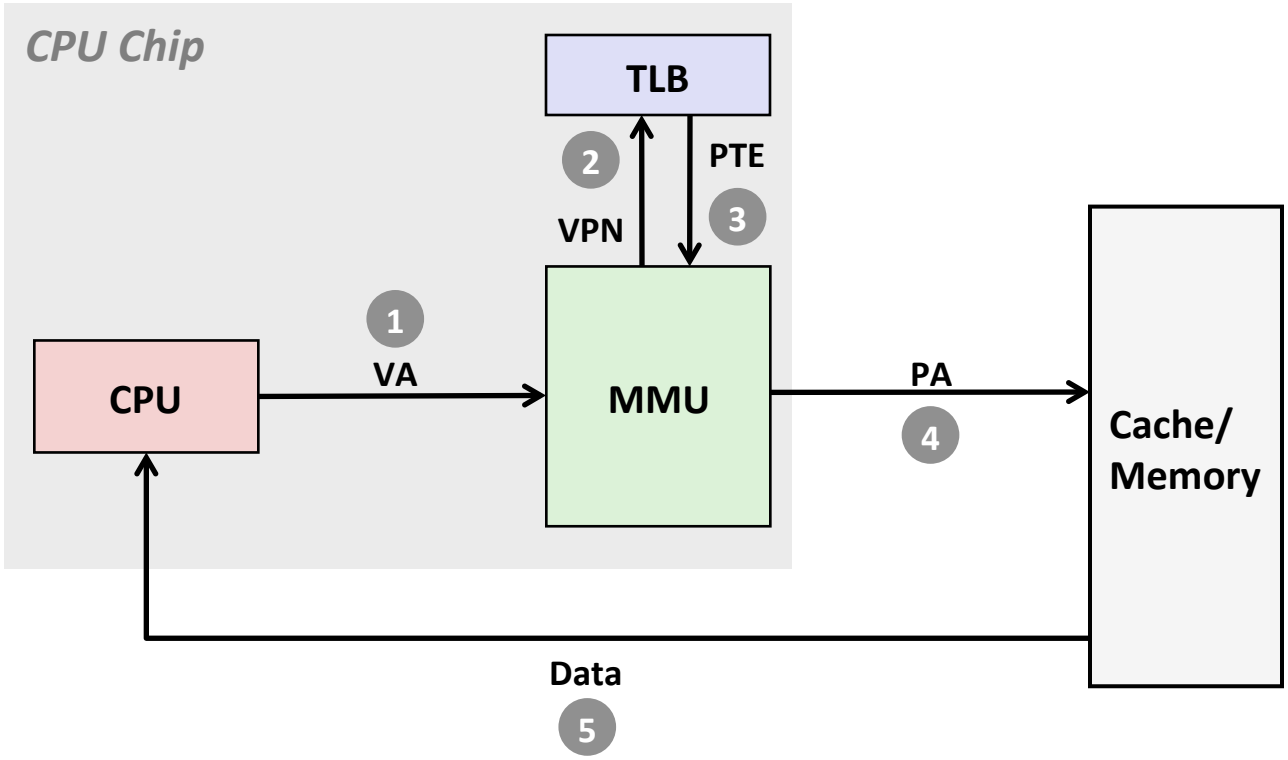
- **The MMU accesses memory *twice*: once to first get the PTE for translation, and then again for the actual memory request from the CPU**
  - The PTEs *may* be cached in L1 like any other memory word
    - But they may be evicted by other data references
    - And a hit in the L1 cache still requires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

- *What can we do to make this faster?*

Speeding up Translation with a TLB

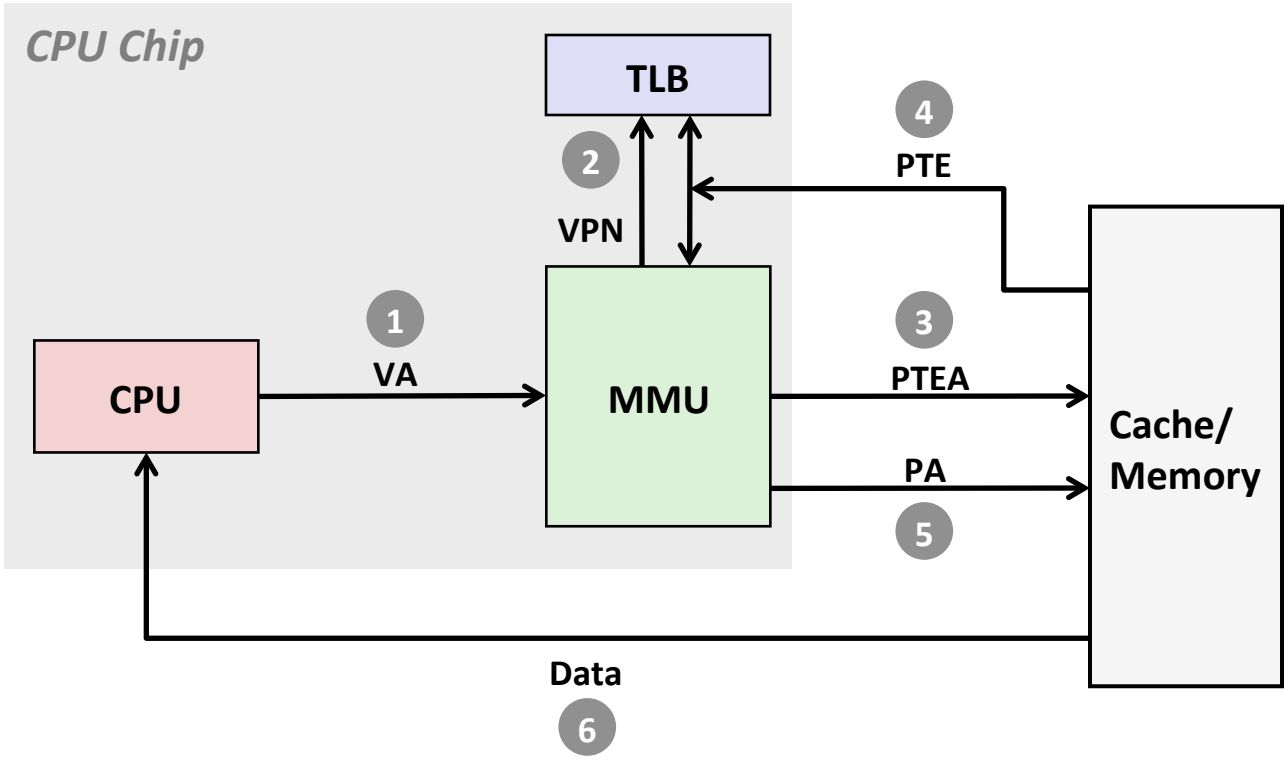
- **Solution: add another cache!**
- ***Translation Lookaside Buffer*** **(TLB):**
  - Small hardware cache in MMU
  - Maps virtual page numbers to physical page numbers
  - Contains complete *page table entries* for small number of pages
    - Modern Intel processors: 128 or 256 entries in TLB

TLB Hit



A TLB hit eliminates a memory access

TLB Miss



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

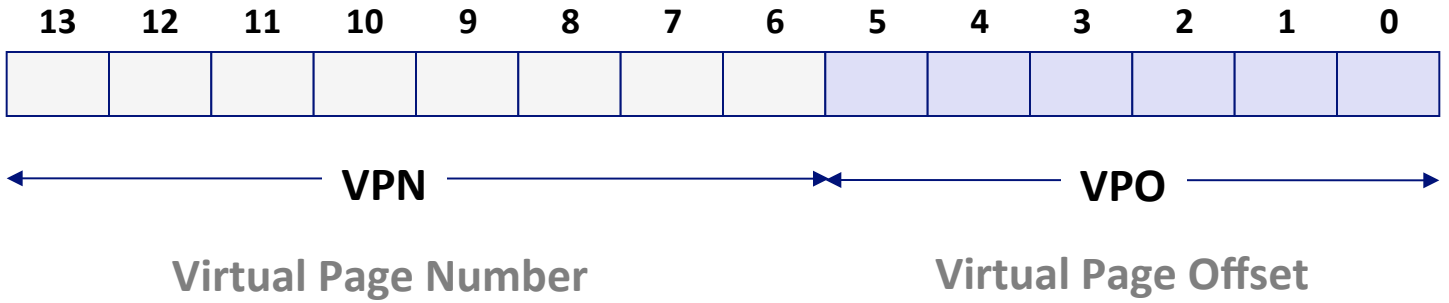
Fortunately, TLB misses are rare

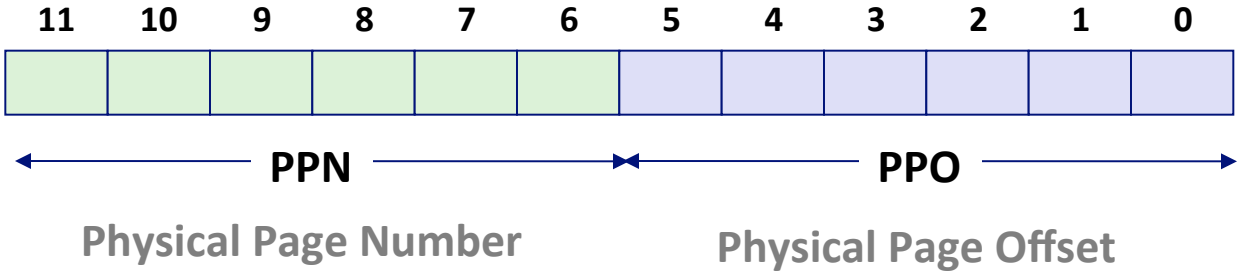
Virtual Memory (VM)

- Overview and motivation
- Indirection
- VM as a tool for caching
- Memory management/protection and address translation
- Virtual memory example

Simple Memory System Example

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





Simple Memory System Page Table

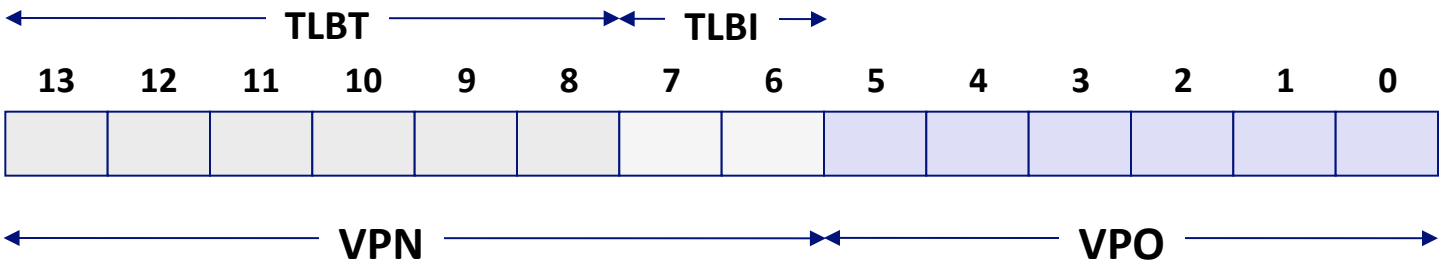
- Only showing first 16 entries (out of 256)

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

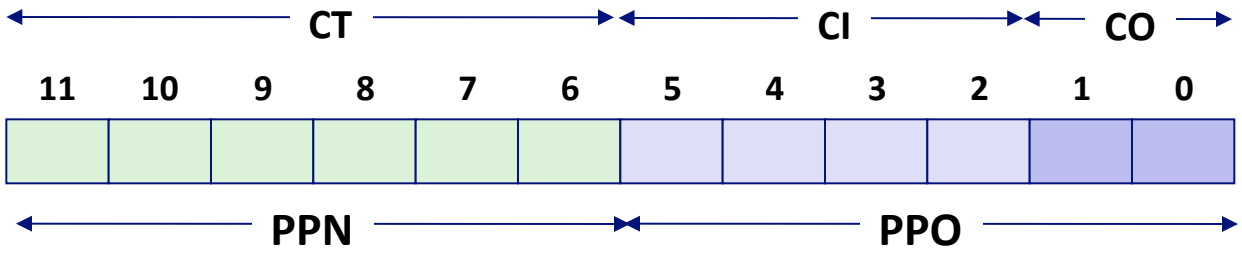
- 16 entries
- 4-way associative



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



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	–	–	–	–

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
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

Page table

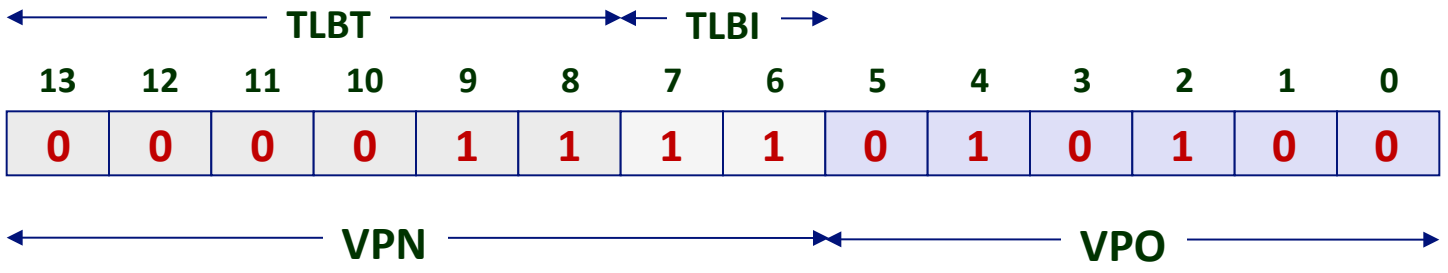
Cache

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	–	–	–	–

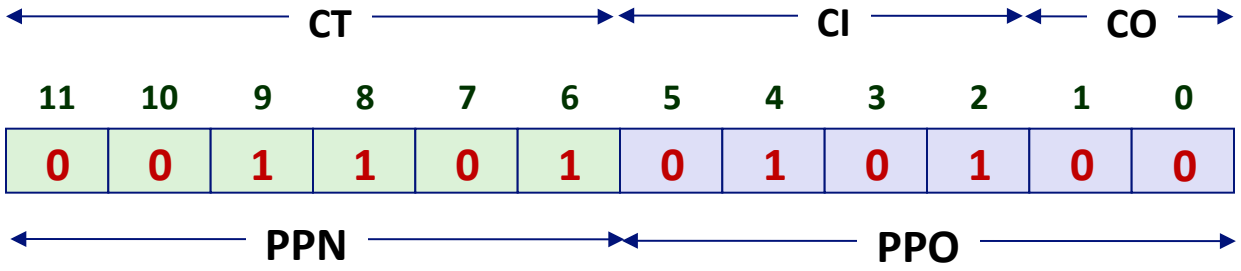
Address Translation Example #1

Virtual Address: 0x03D4



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

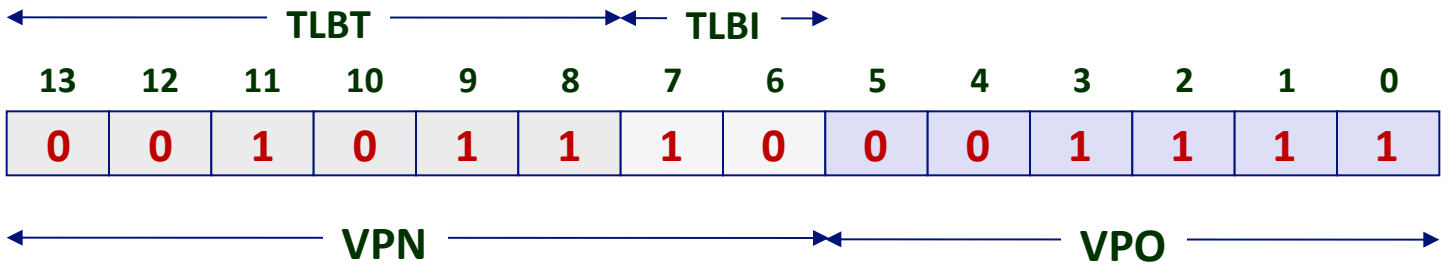
Physical Address



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

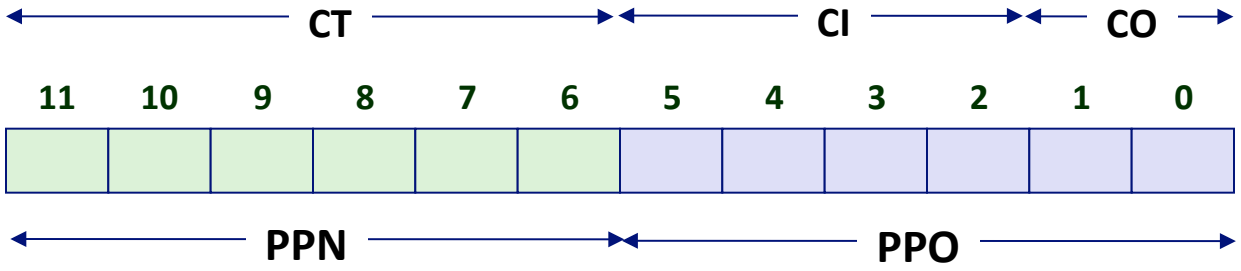
Address Translation Example #2

Virtual Address: 0x0B8F



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

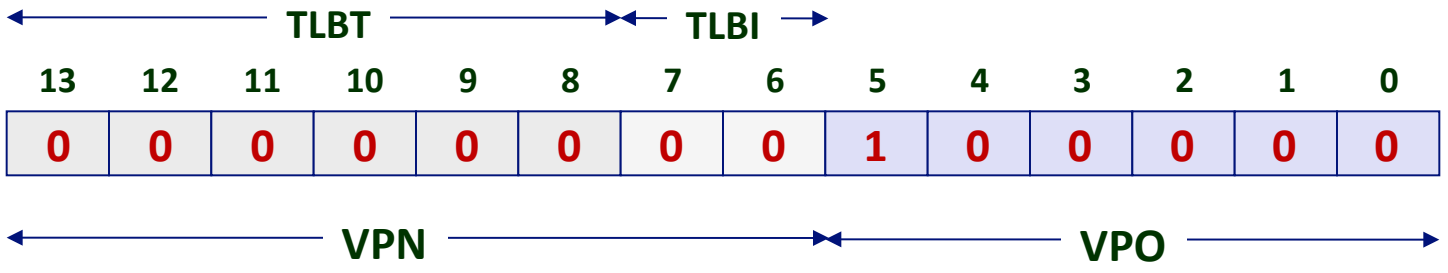
Physical Address



CO ___ CI___ CT ____ Hit? __ Byte: ____

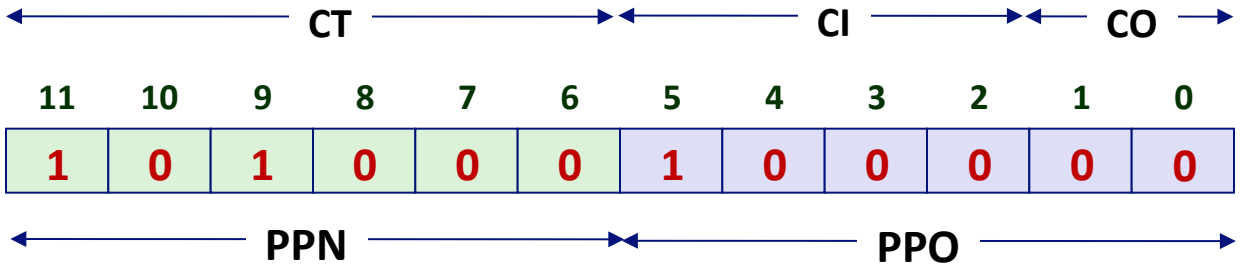
Address Translation Example #3

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)