Virtual Memory

CS230 System Programming
12th Lecture

Instructors:

Jongse Park

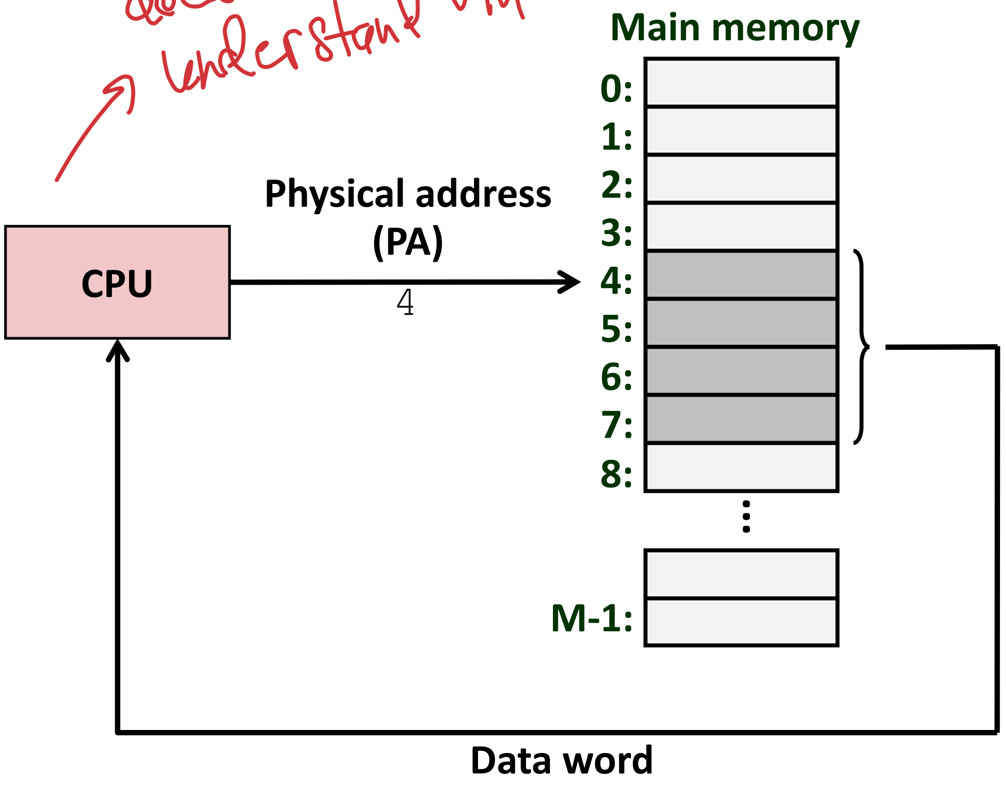
Today

- **Address spaces** VM, PM
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- Address translation

} why VM is uniquely good

A System Using Physical Addressing

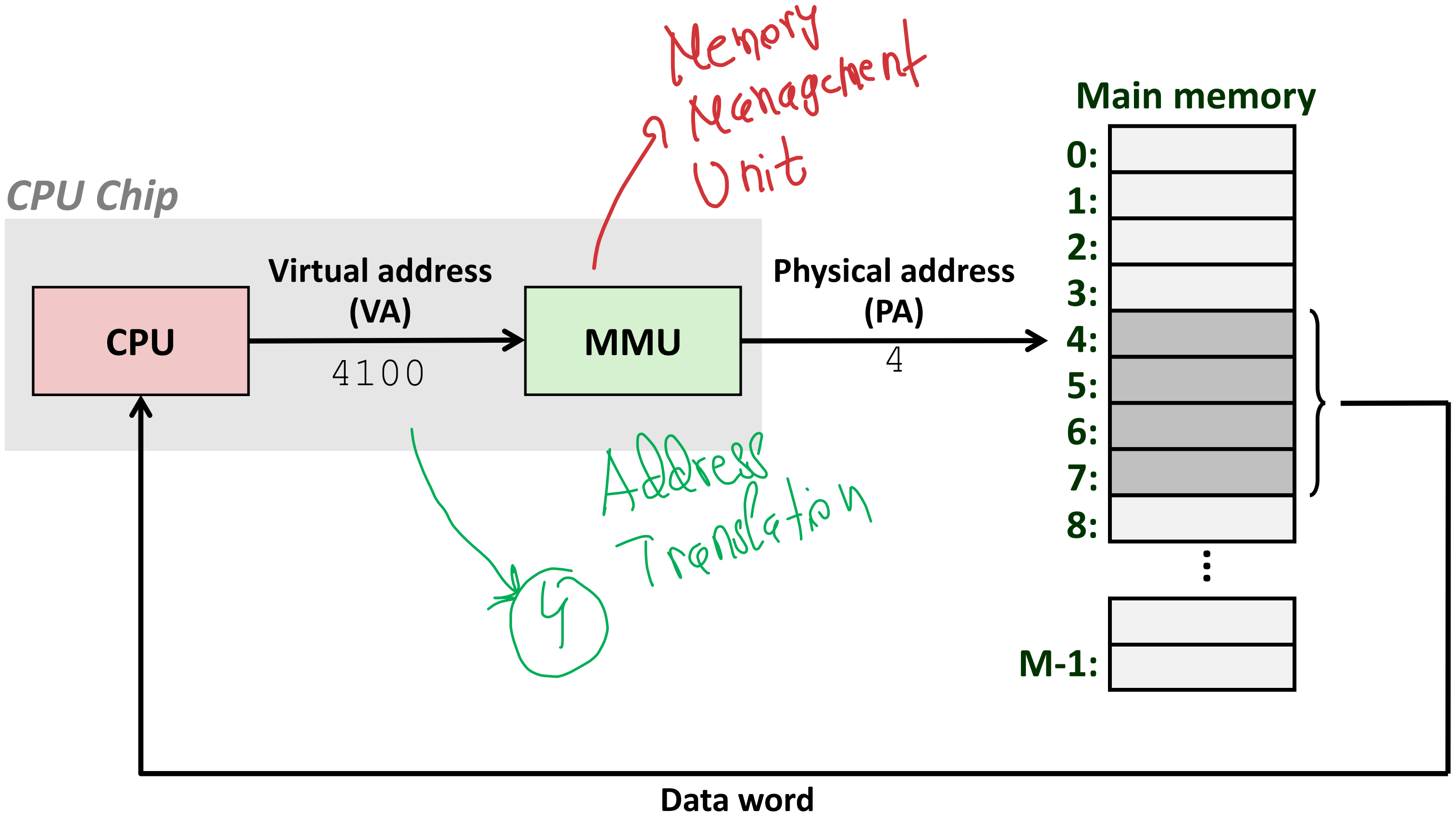
does not understand VM



2^{48}
heap
text
0

- **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 servers, laptops, and smart phones
- One of the great ideas in computer science

Address Spaces

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

  {0, 1, 2, 3 … }

  # of bits

- **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

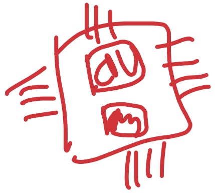
  {0, 1, 2, 3, …, M-1}

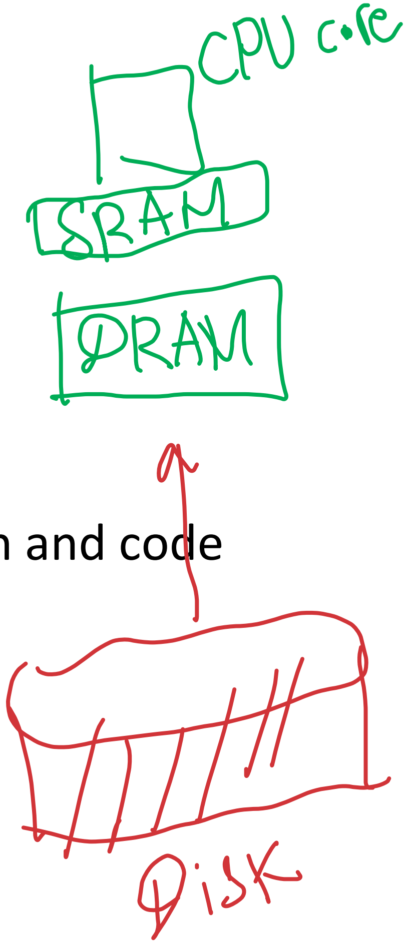
both are linear address space

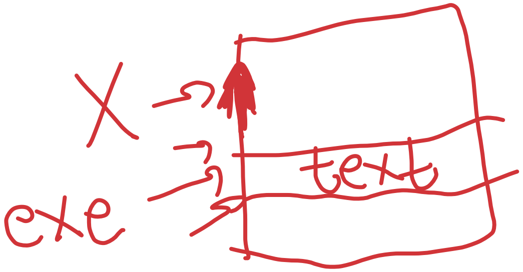
Why Virtual Memory (VM)?

Majority of nonessentials into disk

- **Uses main memory efficiently**
  - Use DRAM as a cache for parts of a virtual address space
- **Simplifies memory management**
  - Each process gets the same uniform linear address space
- **Isolates address spaces**
  - One process can't interfere with another's memory
  - User program cannot access privileged kernel information and code





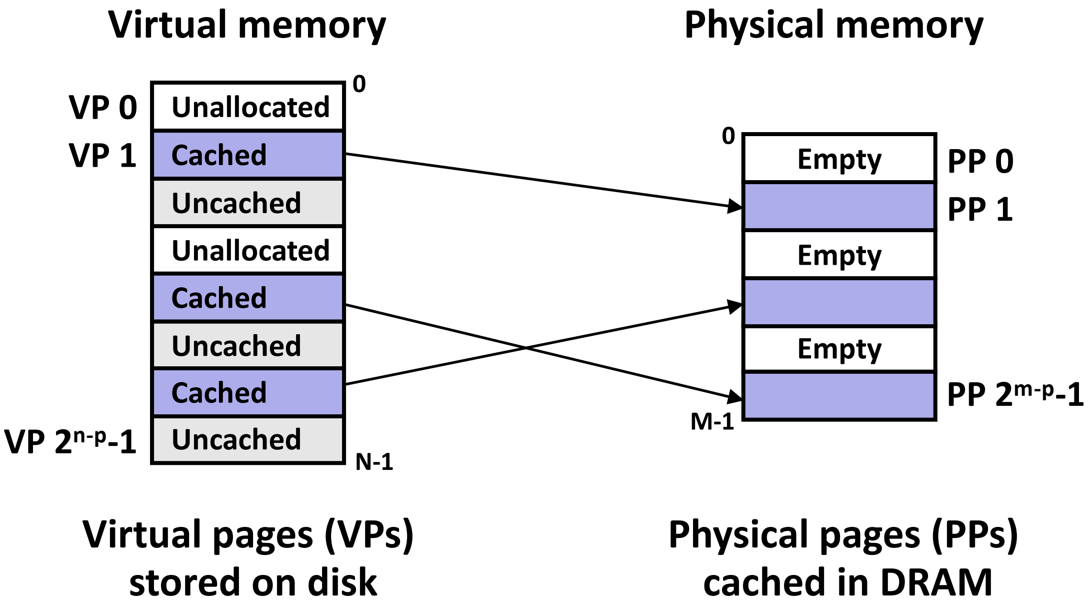


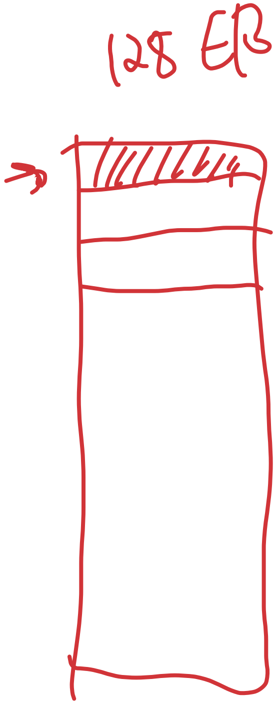
Today

- 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 Caching

- **Conceptually, *virtual memory* is an array of N contiguous bytes stored on disk.**
- **The contents of the array on disk are cached in *physical memory* (*DRAM cache*)**
  - These cache blocks are called *pages* (size is P = 2^p bytes)





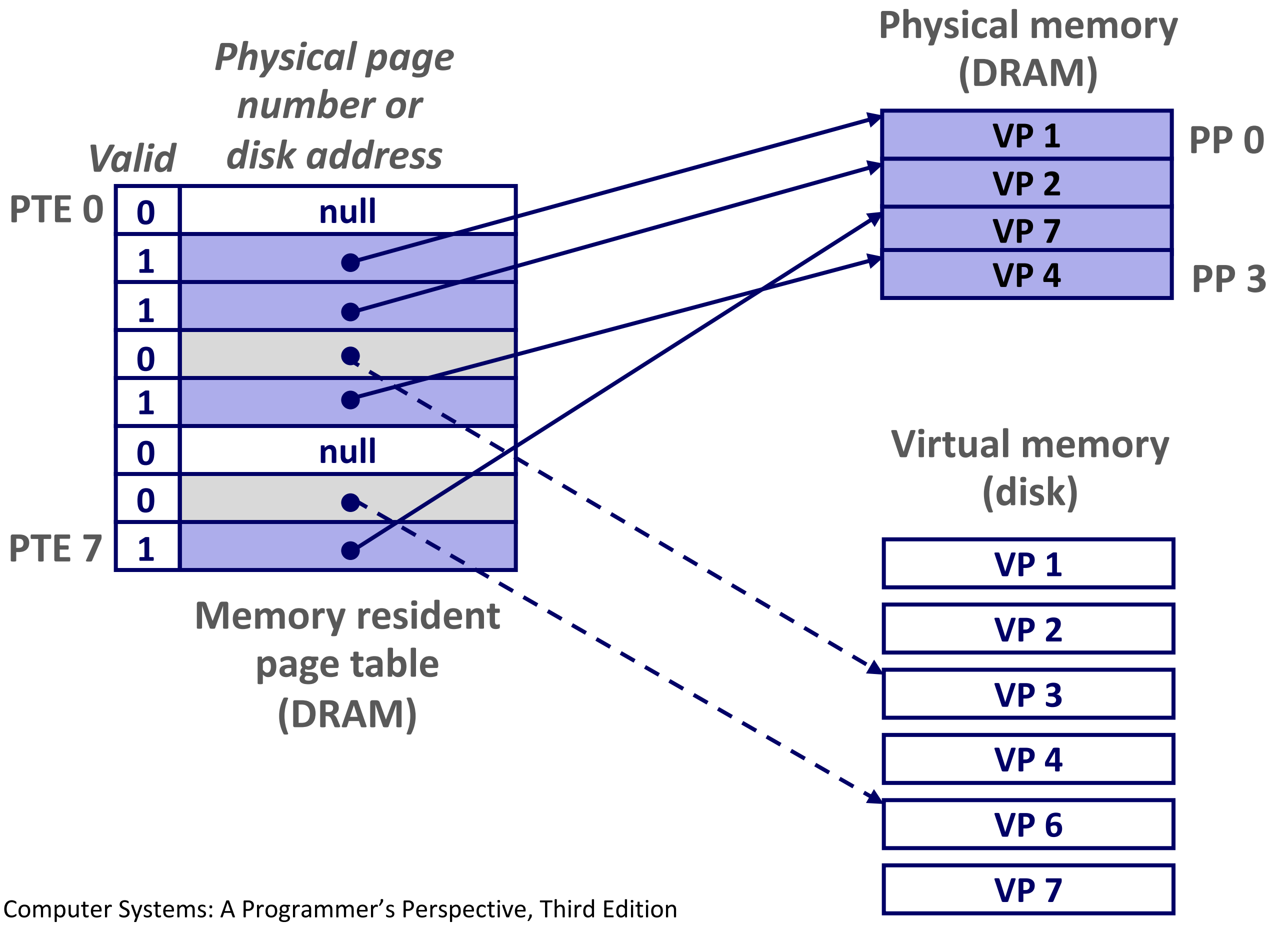
DRAM Cache Organization

- **DRAM cache organization driven by the enormous miss penalty**
  - DRAM is about ***10x*** slower than SRAM
  - Disk is about ***10,000x*** slower than DRAM

- **Consequences**
  - Large page (block) size: typically 4 KB, sometimes 4 MB
  - Fully associative
    - Any VP can be placed in any PP
    - Requires a “large” mapping function – different from cache memories
  - Highly sophisticated, expensive replacement algorithms
    - Too complicated and open-ended to be implemented in hardware
  - Write-back rather than write-through

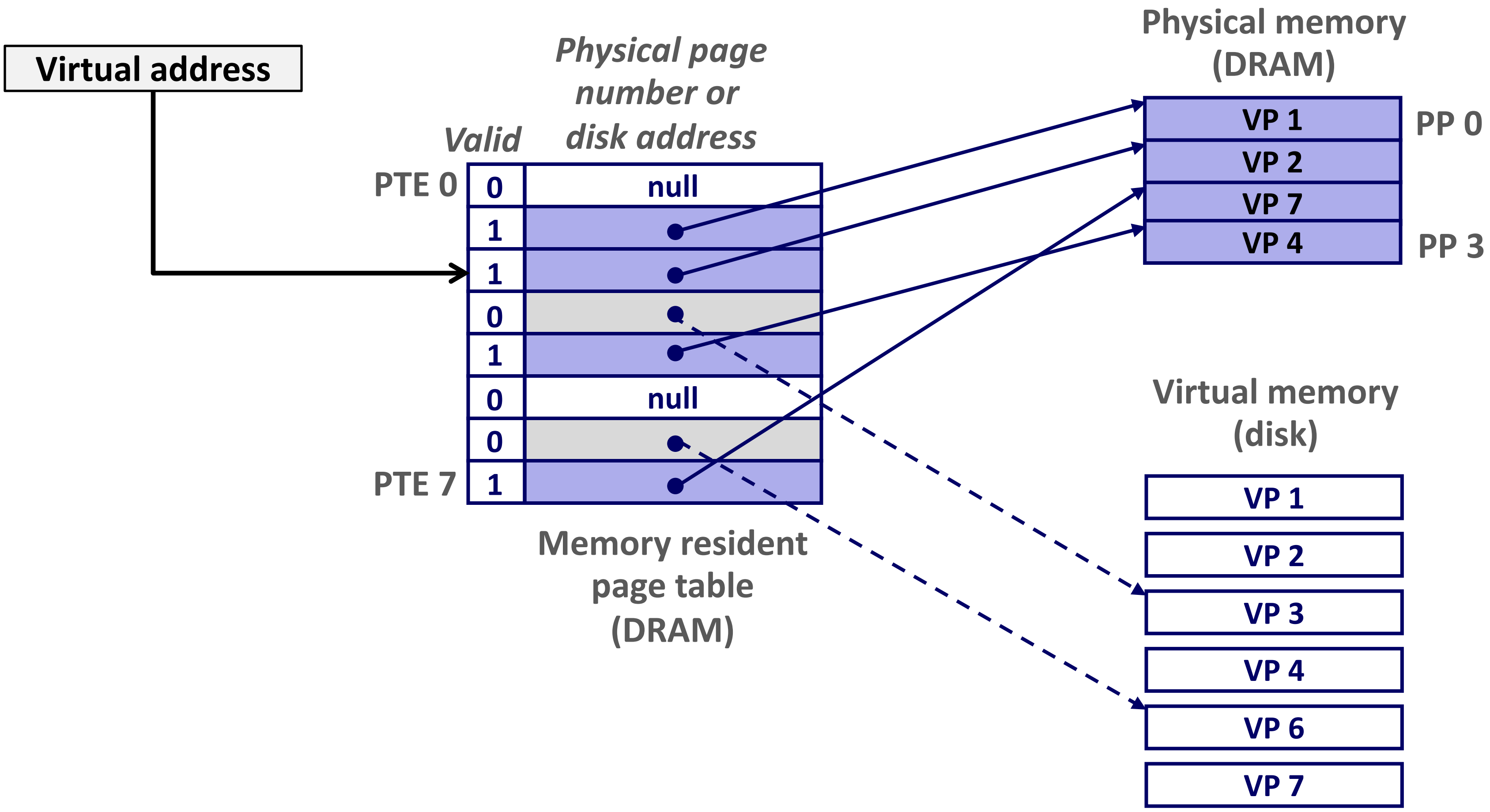
Enabling Data Structure: Page Table

- **A *page table* is an array of page table entries (PTEs) that maps virtual pages to physical pages.**
  - Per-process kernel data structure in DRAM



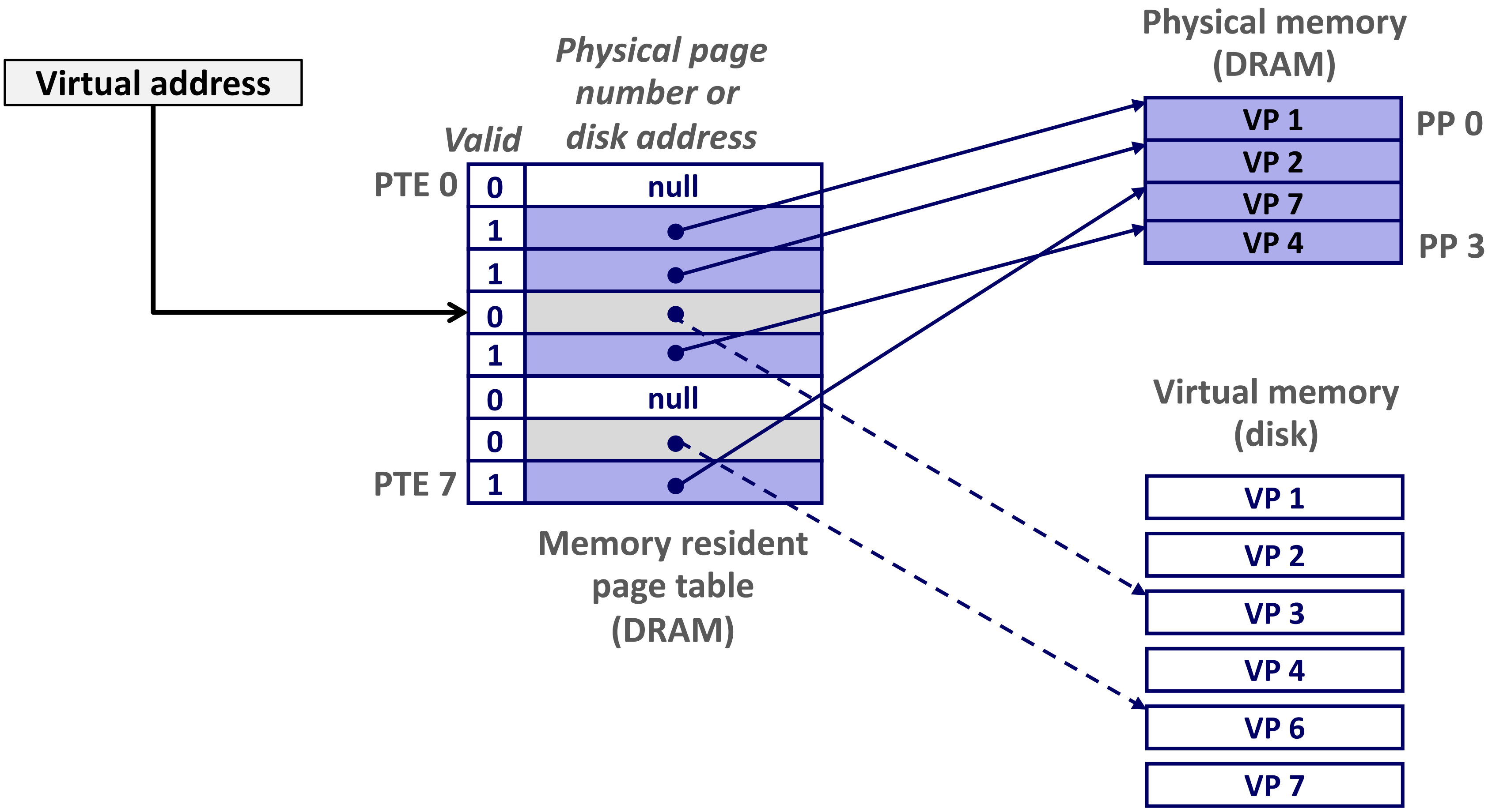
Page Hit

- ***Page hit:*** **reference to VM word that is in physical memory (DRAM cache hit)**



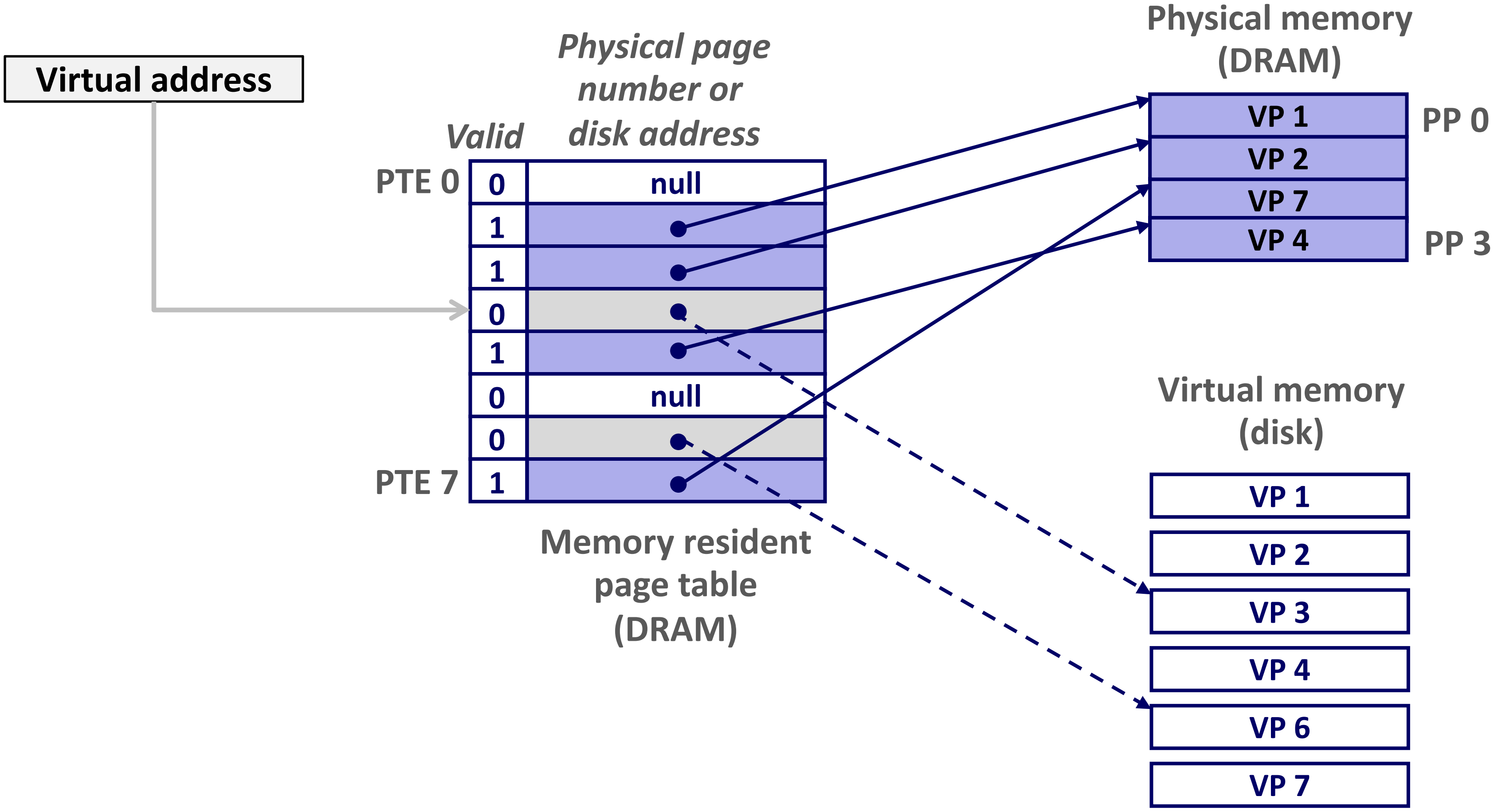
Page Fault

- ***Page fault:*** **reference to VM word that is not in physical memory (DRAM cache miss)**



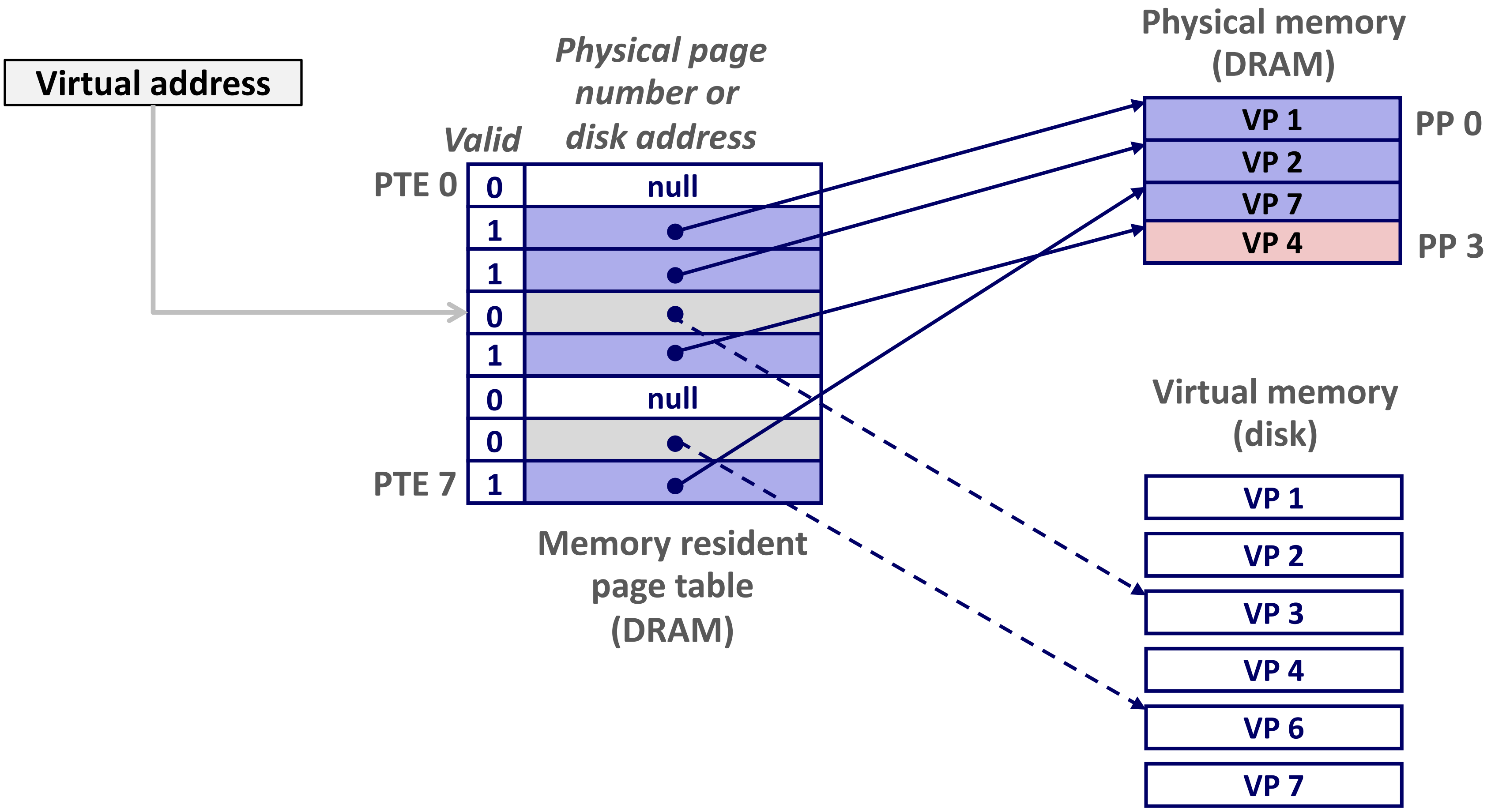
Handling Page Fault

- Page miss causes page fault (an exception)



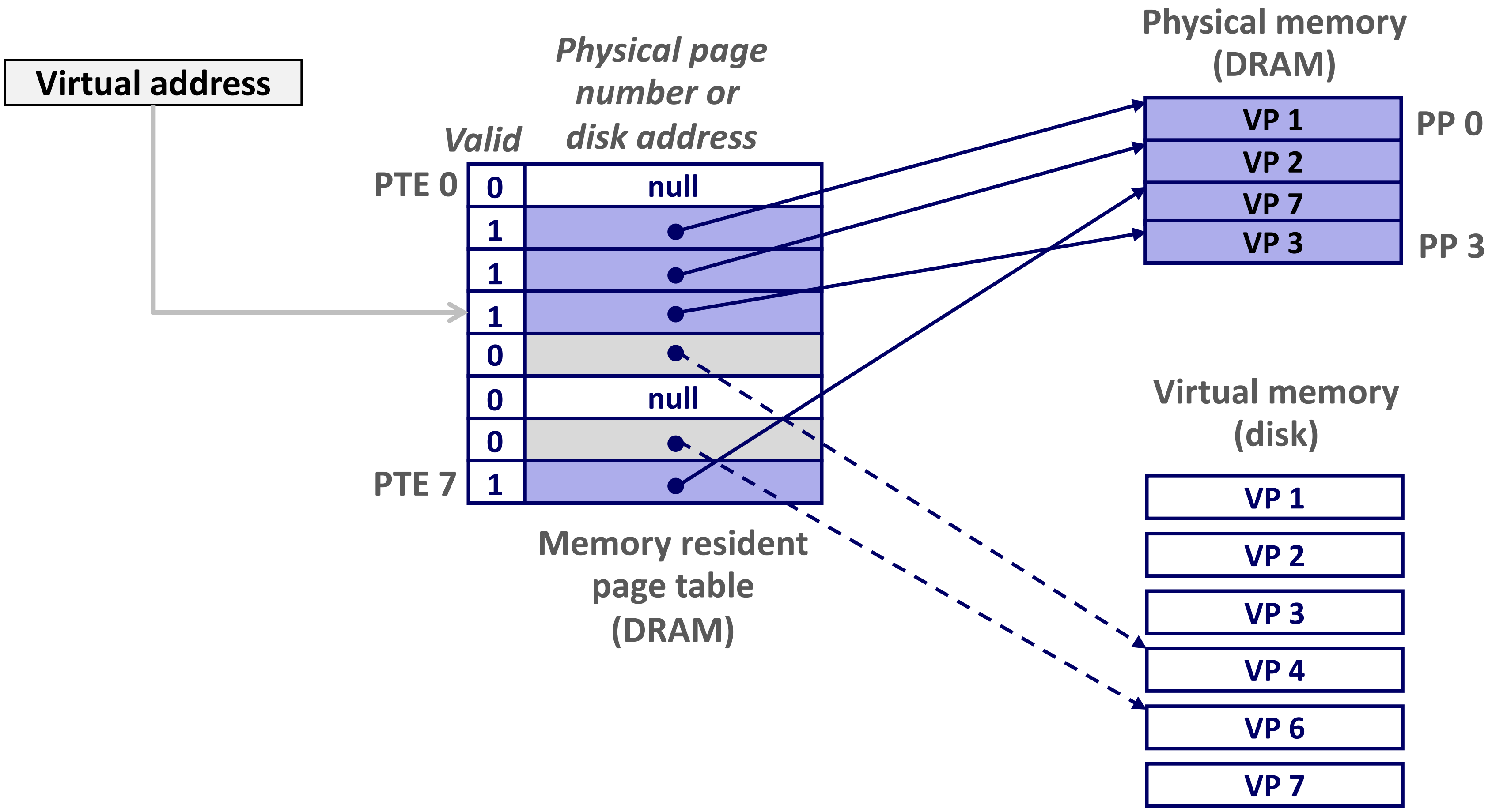
Handling Page Fault

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



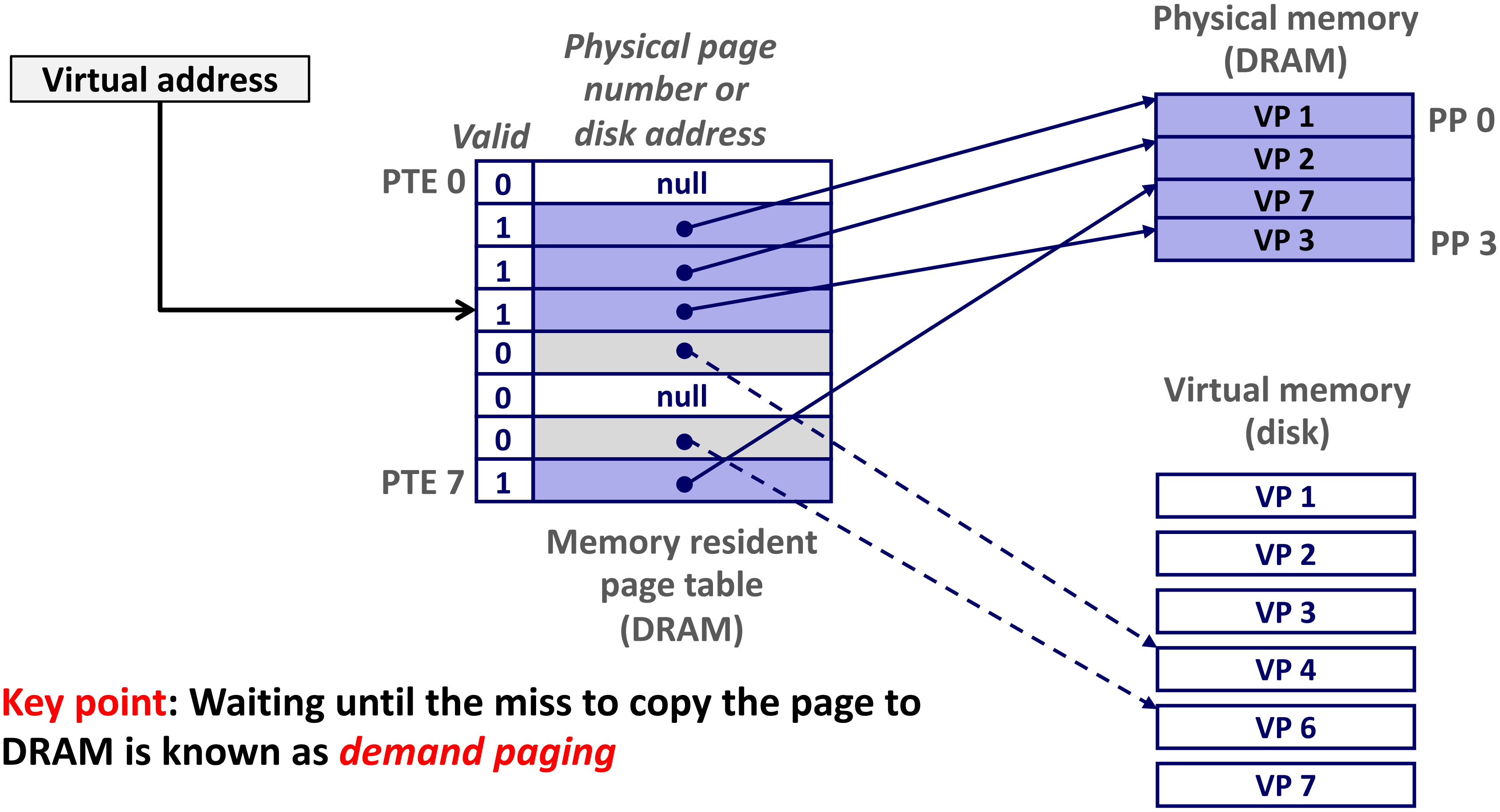
Handling Page Fault

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



Handling Page Fault

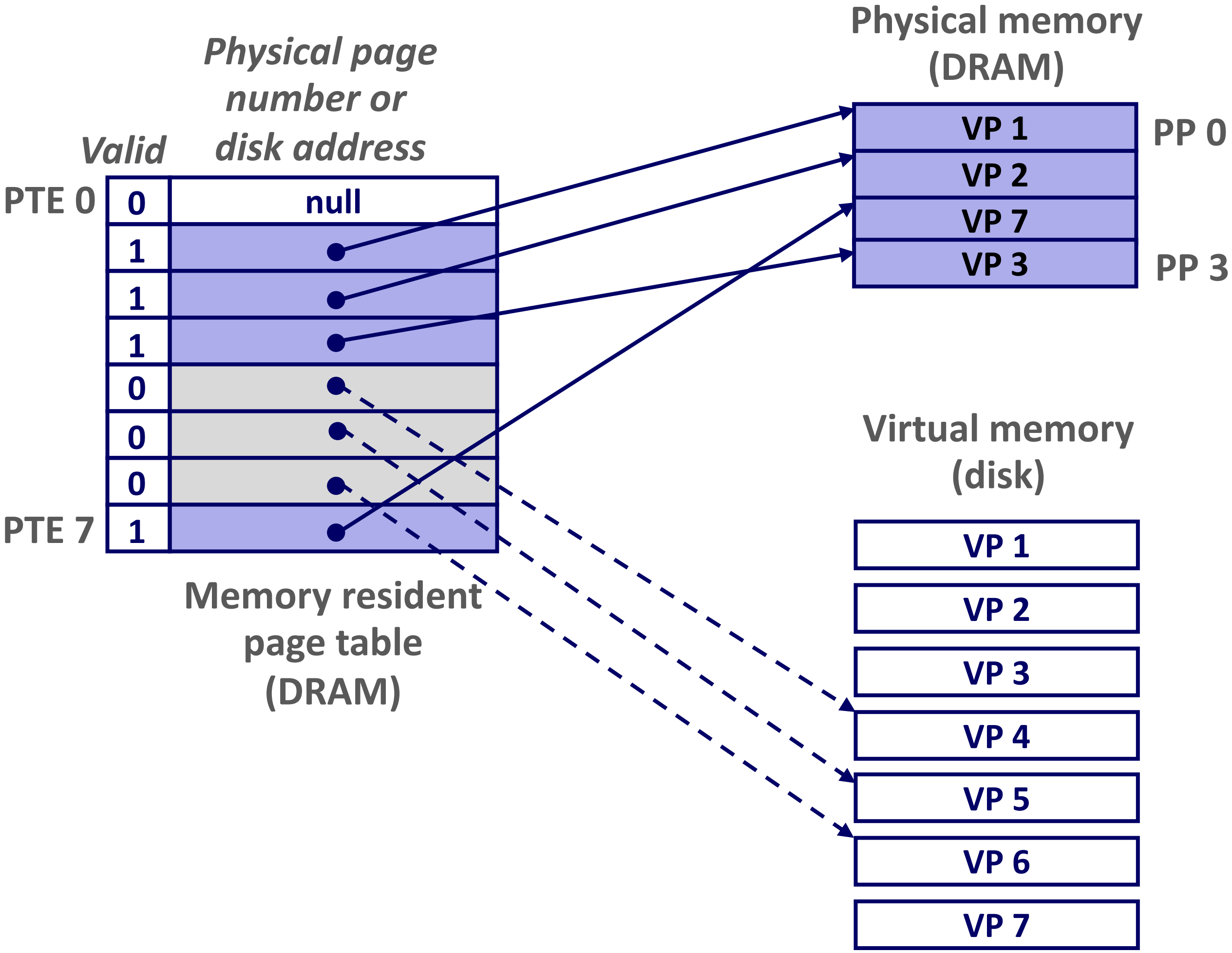
- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!



Key point: Waiting until the miss to copy the page to DRAM is known as *demand paging*

Allocating Pages

- **Allocating a new page (VP 5) of virtual memory.**



Locality to the Rescue Again!

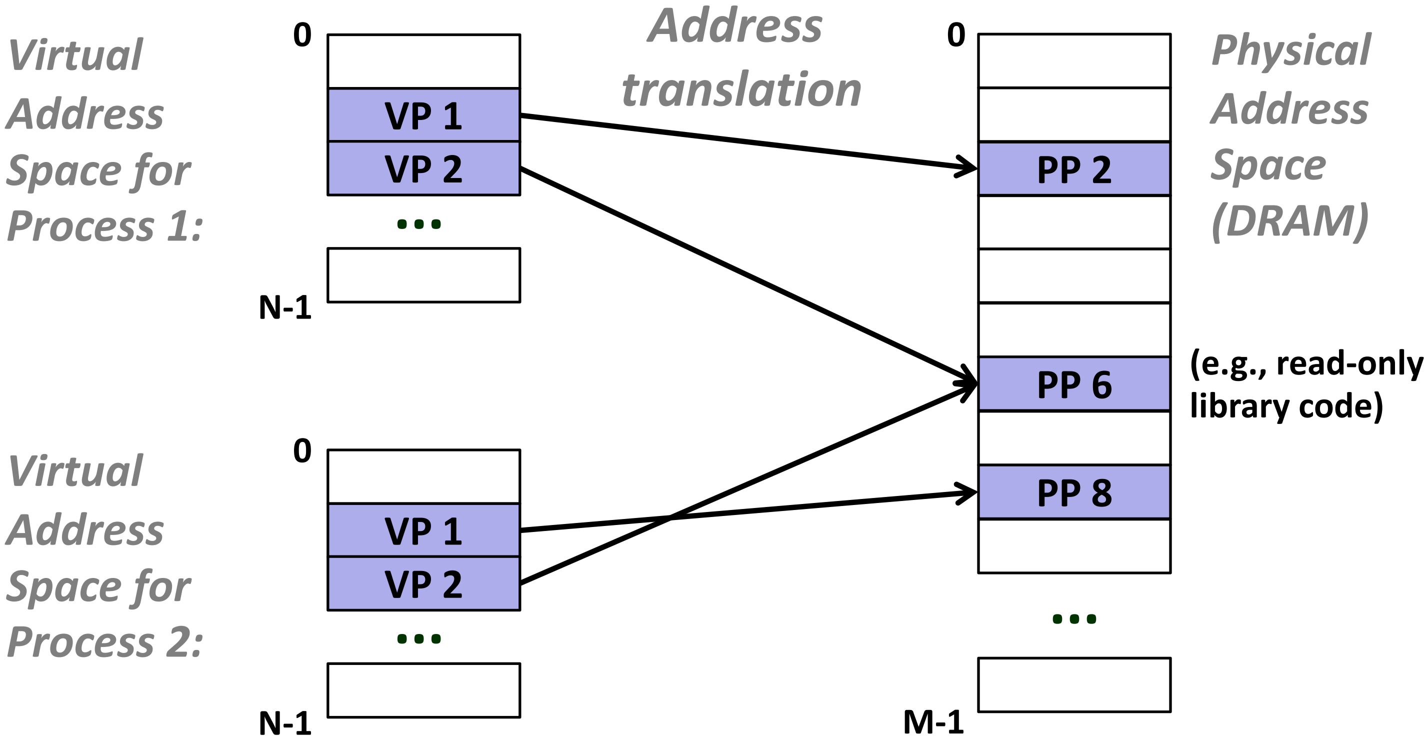
- **Virtual memory seems terribly inefficient, but it works because of locality.**
- **At any point in time, programs tend to access a set of active virtual pages called the *working set***
  - Programs with better temporal locality will have smaller working sets
- **If (working set size < main memory 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

Today

- Address spaces
- VM as a tool for caching
- **VM as a tool for memory management**
- VM as a tool for memory protection
- Address translation

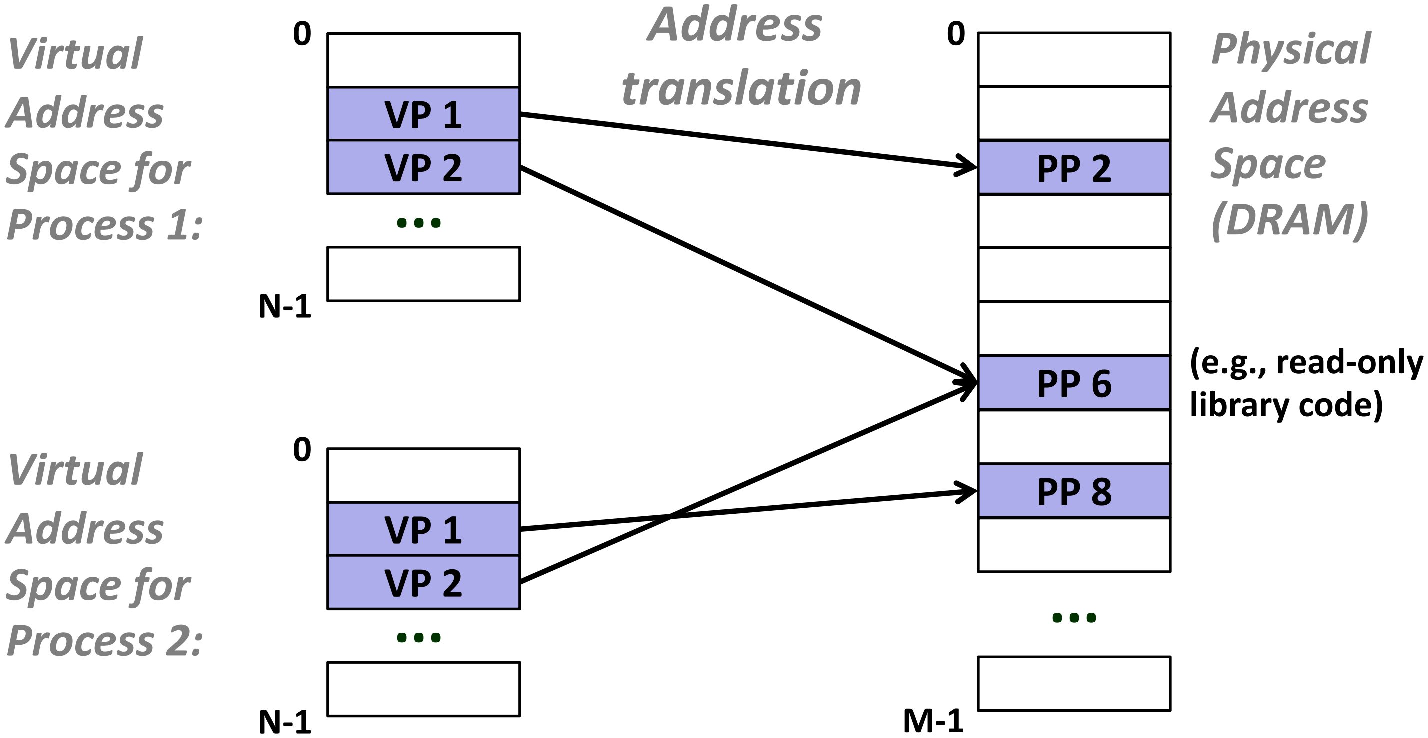
VM as a Tool for Memory Management

- **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


Virtual Address Space for Process 1:
0
VP 1
VP 2
...
N-1
Address translation
0
PP 2
PP 6
PP 8
...
M-1
Physical Address Space (DRAM)
(e.g., read-only library code)
Virtual Address Space for Process 2:
0
VP 1
VP 2
...
N-1

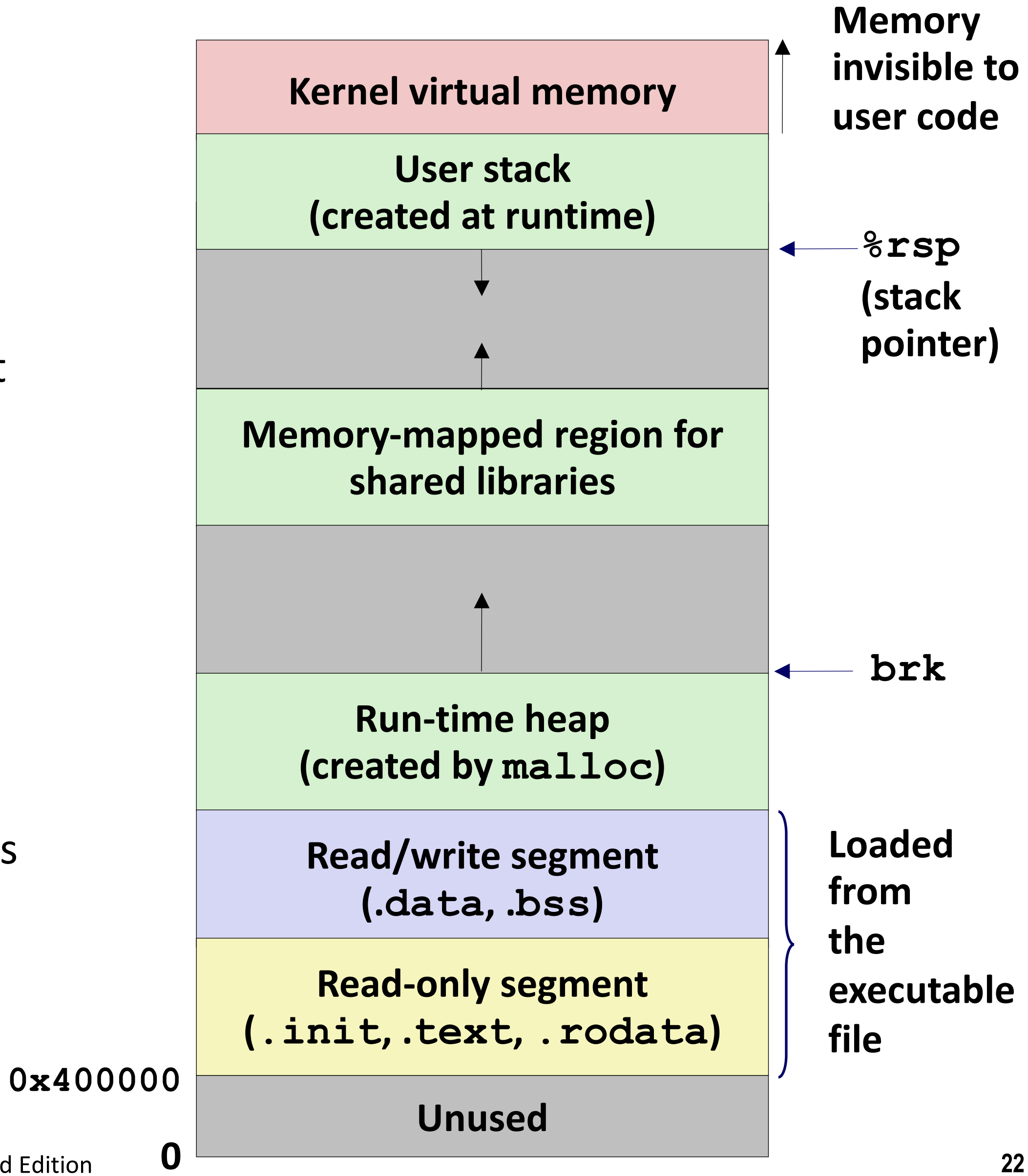
VM as a Tool for Memory Management

- **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)


Virtual Address Space for Process 1:
0
VP 1
VP 2
...
N-1
Address translation
0
PP 2
PP 6
PP 8
...
M-1
Physical Address Space (DRAM)
(e.g., read-only library code)
Virtual Address Space for Process 2:
0
VP 1
VP 2
...
N-1

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


Memory invisible to user code
Kernel virtual memory
User stack (created at runtime)
%rsp (stack pointer)
Memory-mapped region for shared libraries
brk
Run-time heap (created by malloc)
Read/write segment (.data, .bss)
Loaded from the executable file
Read-only segment (.init, .text, .rodata)
0x400000
Unused
0

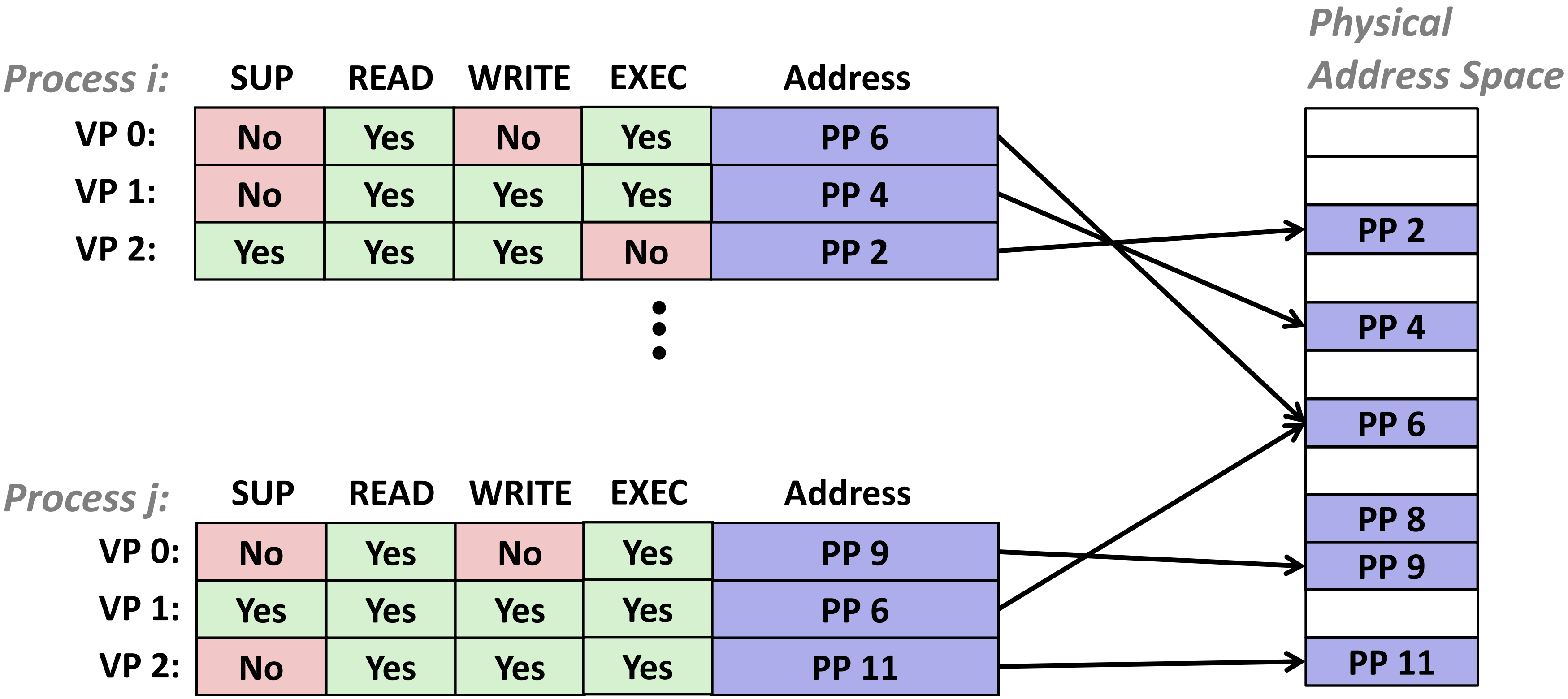
Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Today

- 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



Process i:

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

⋮

Process j:

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

Today

- Address spaces
- VM as a tool for caching
- VM as a tool for memory management
- VM as a tool for memory protection
- **Address translation**

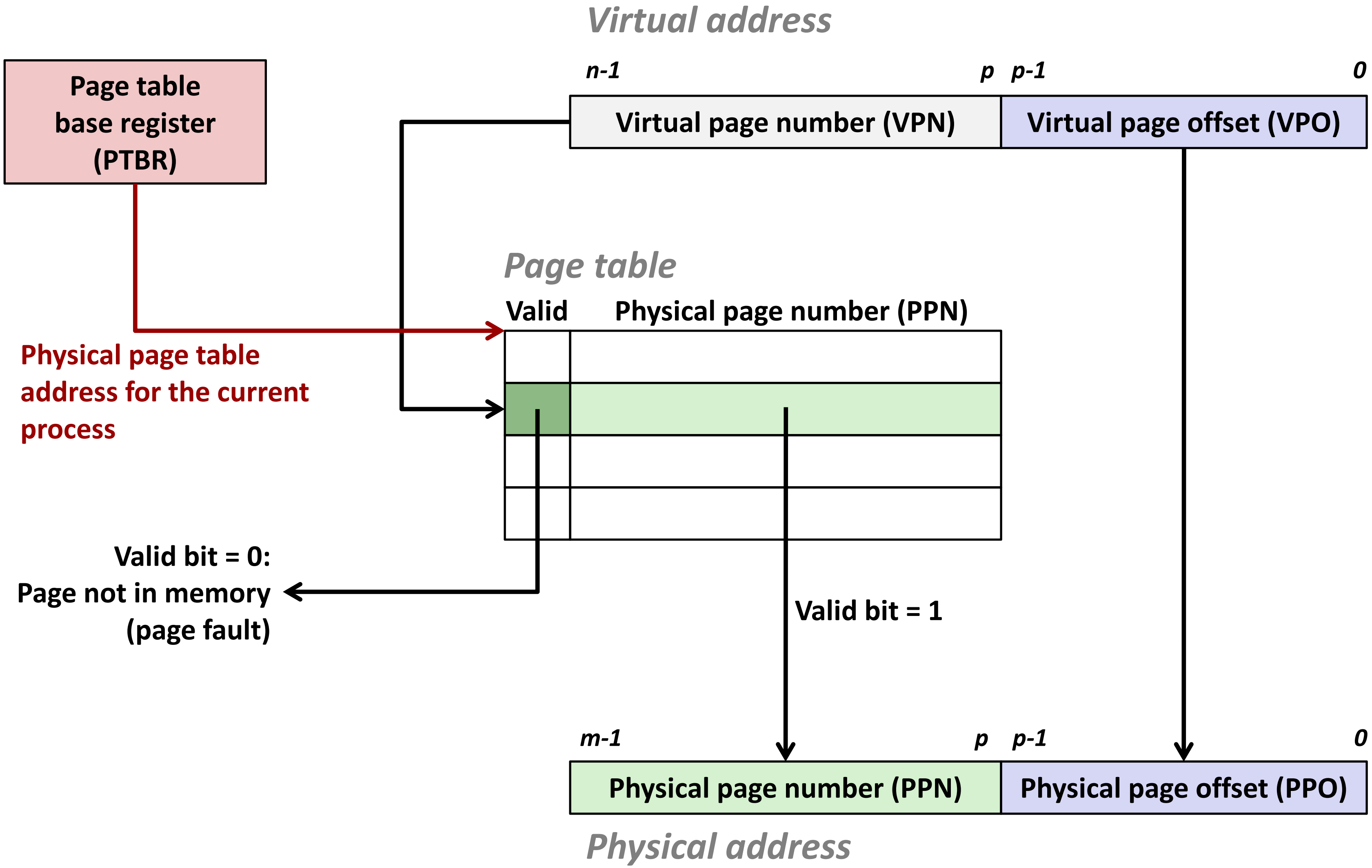
VM Address Translation

- **Virtual Address Space**
  - $V = \{0, 1, \ldots, N-1\}$
- **Physical Address Space**
  - $P = \{0, 1, \ldots, M-1\}$
- **Address Translation**
  - $\boldsymbol{MAP: V \rightarrow P \cup \{\varnothing\}}$
  - 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

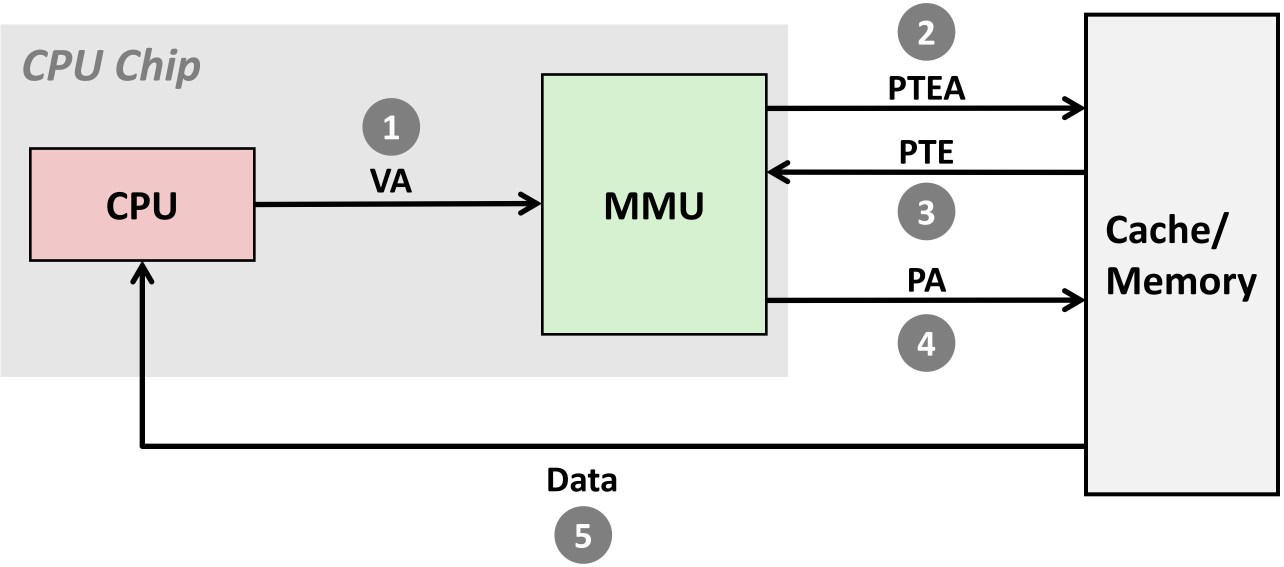
Summary of Address Translation Symbols

- **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)**
  - **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
n-1
p
p-1
0
Virtual page number (VPN)
Virtual page offset (VPO)
Page table
base register
(PTBR)
Page table
Valid
Physical page number (PPN)
Physical page table address for the current process
Valid bit = 0:
Page not in memory
(page fault)
Valid bit = 1
m-1
p
p-1
0
Physical page number (PPN)
Physical page offset (PPO)
Physical address

Address Translation: Page Hit



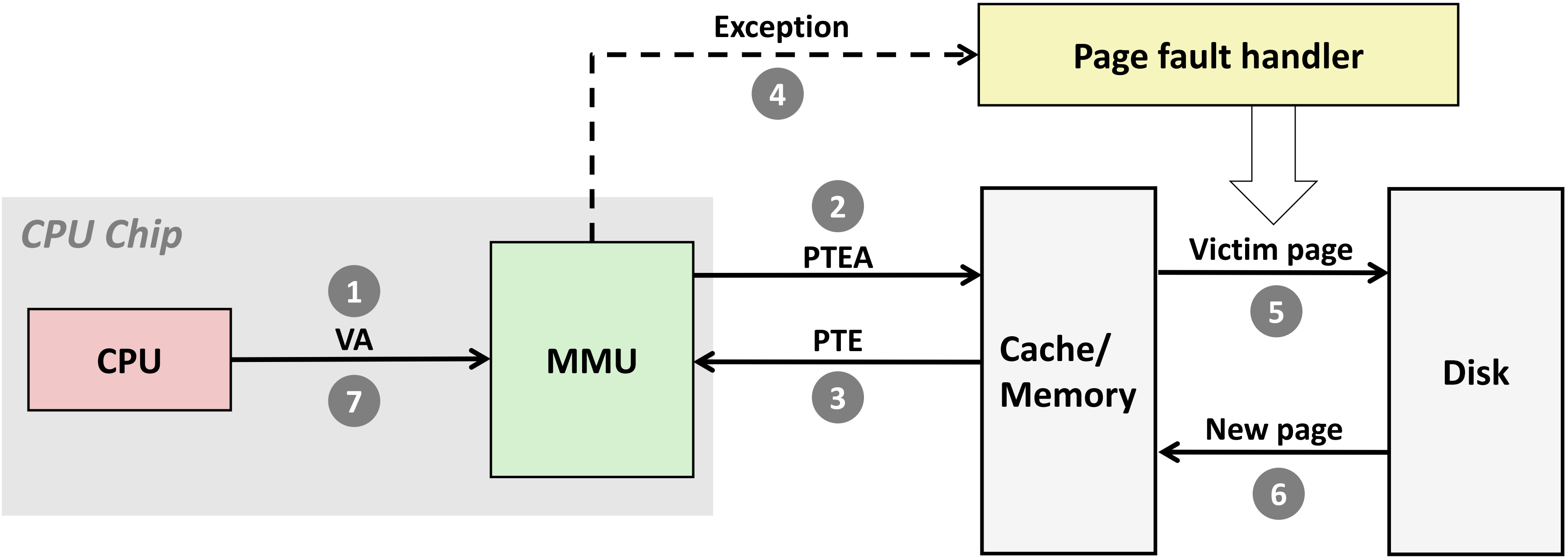
1) Processor sends virtual address to MMU

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

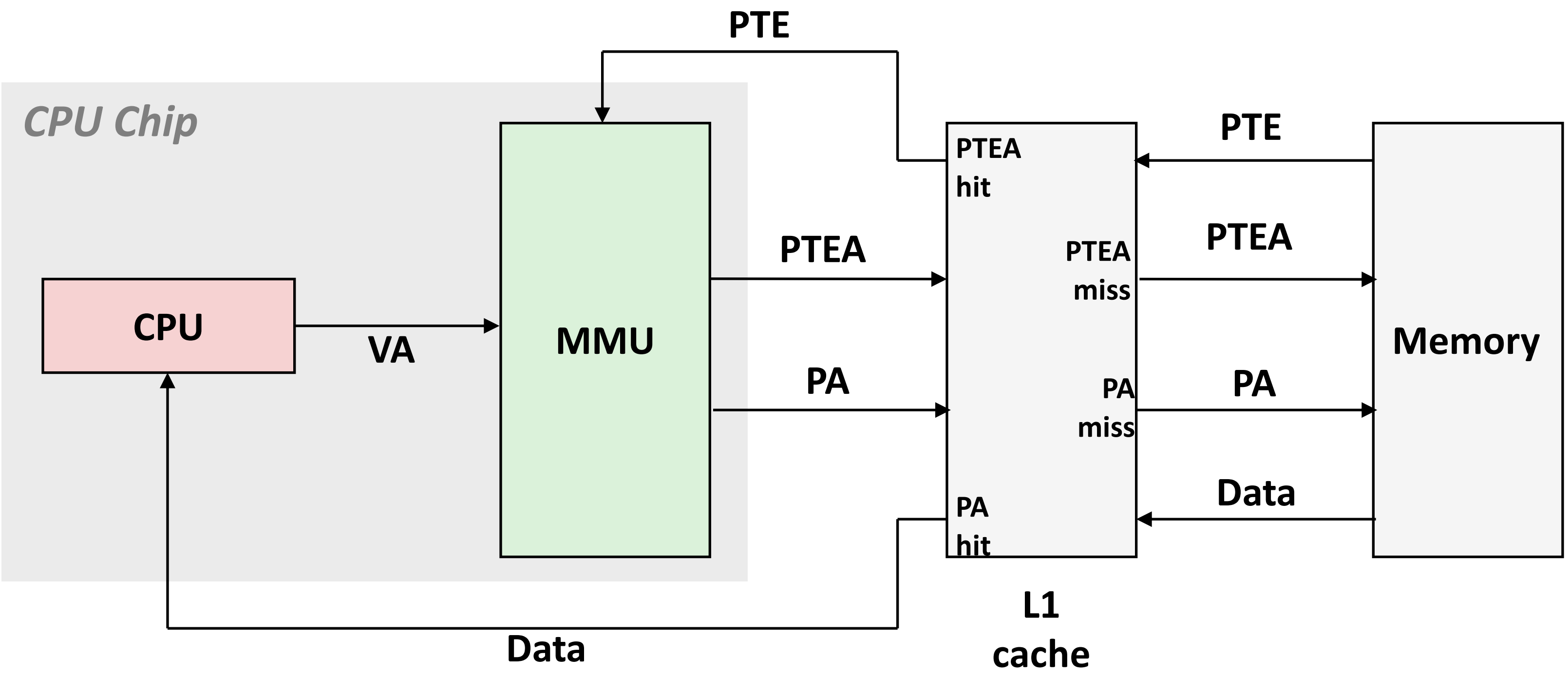
4) Valid bit is zero, so MMU triggers page fault exception

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

Integrating VM and Cache



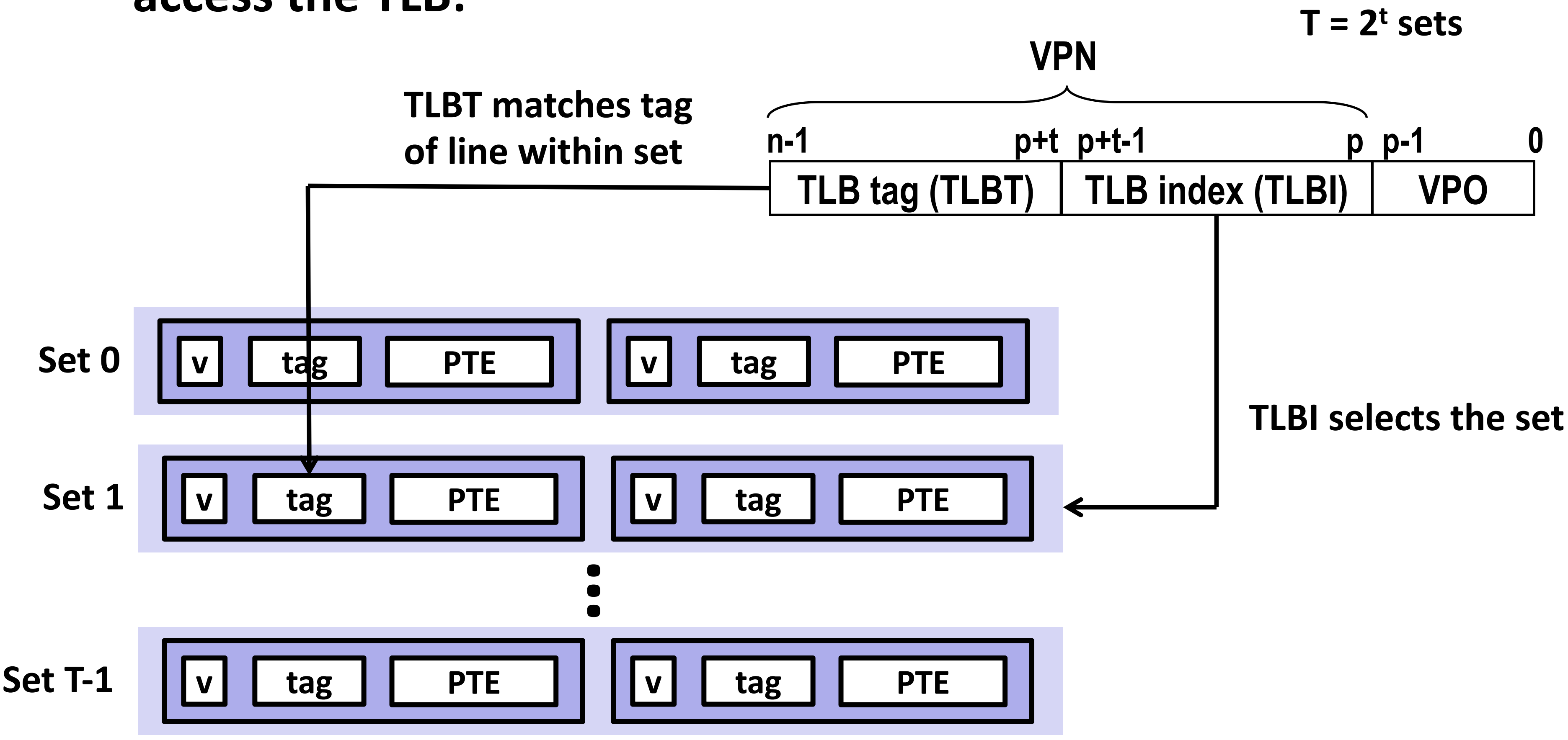
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Speeding up Translation with a TLB

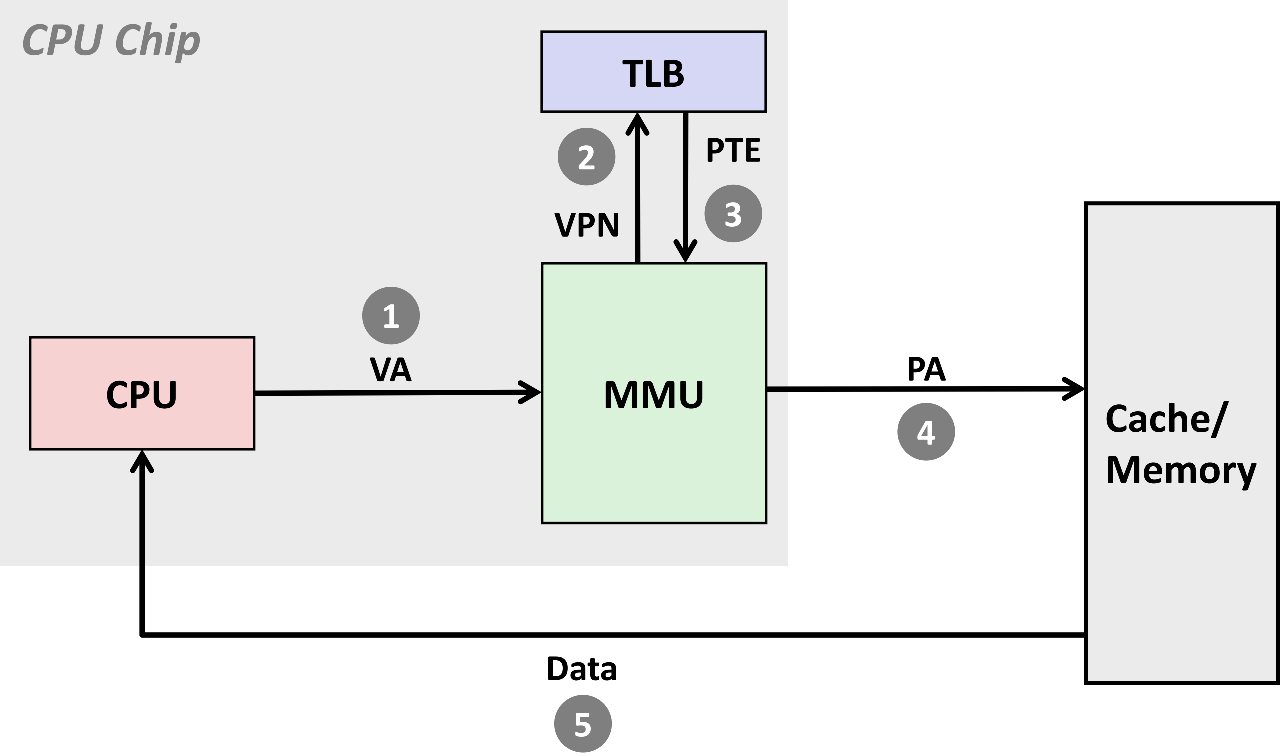
- **Page table entries (PTEs) are cached in L1 like any other memory word**
  - PTEs may be evicted by other data references
  - PTE hit still requires a small L1 delay
- **Solution: *Translation Lookaside Buffer* (TLB)**
  - Small set-associative hardware cache in MMU
  - Maps virtual page numbers to 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:

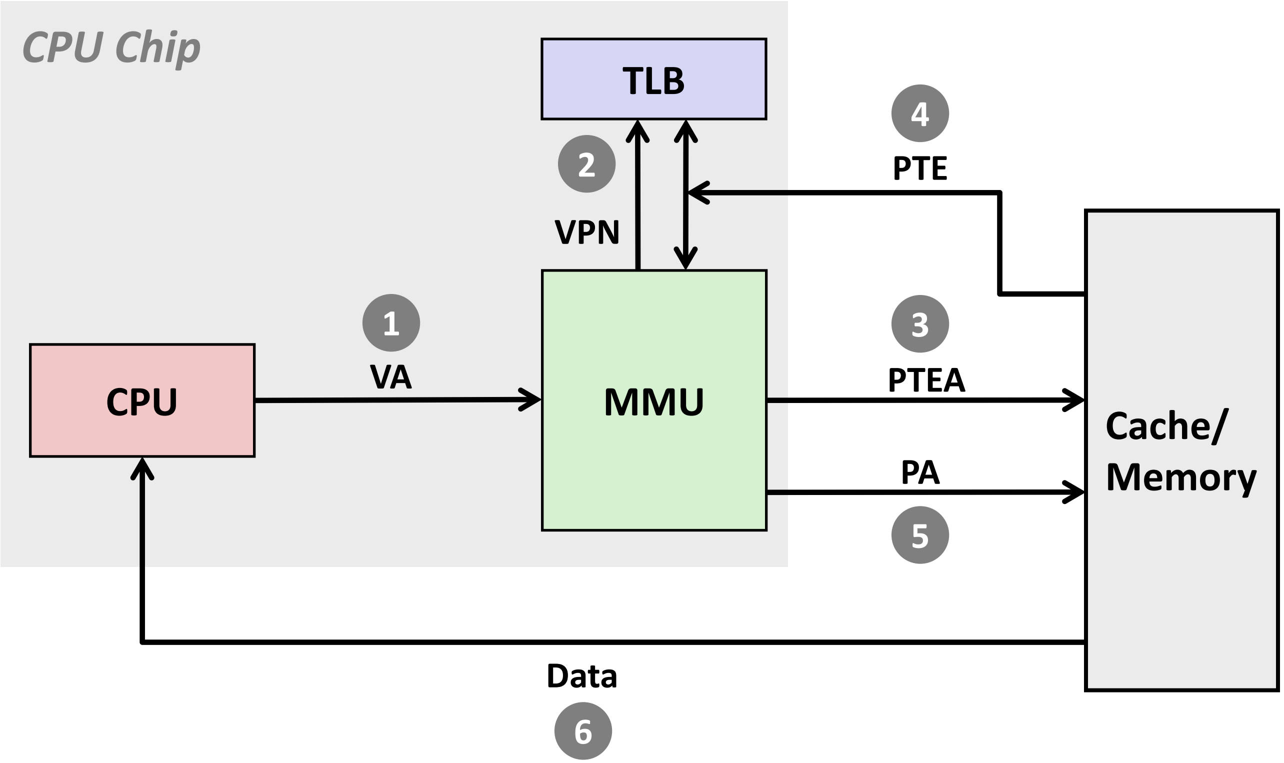


TLB Hit



A TLB hit eliminates a memory access

TLB Miss

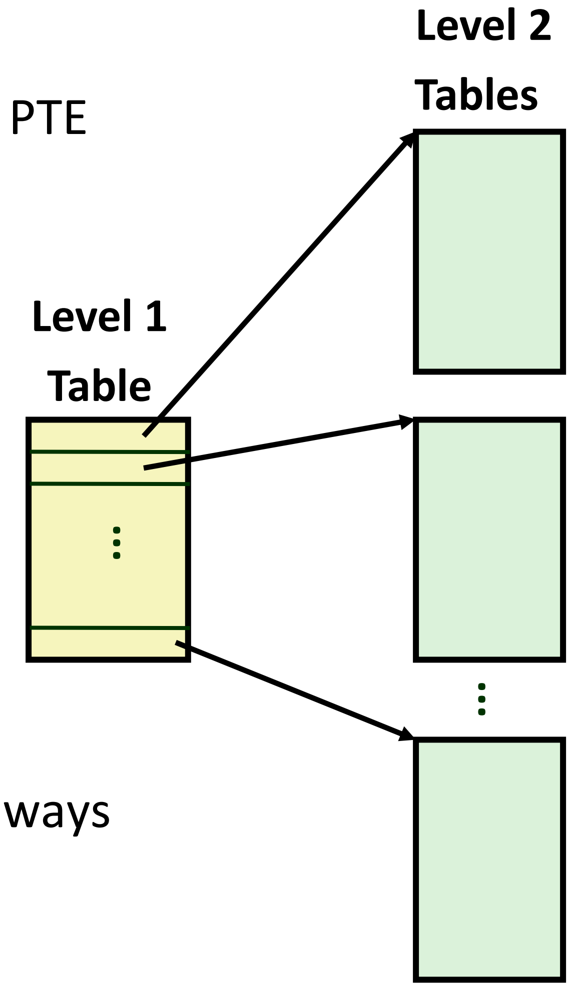


A TLB miss incurs an additional memory access (the PTE)

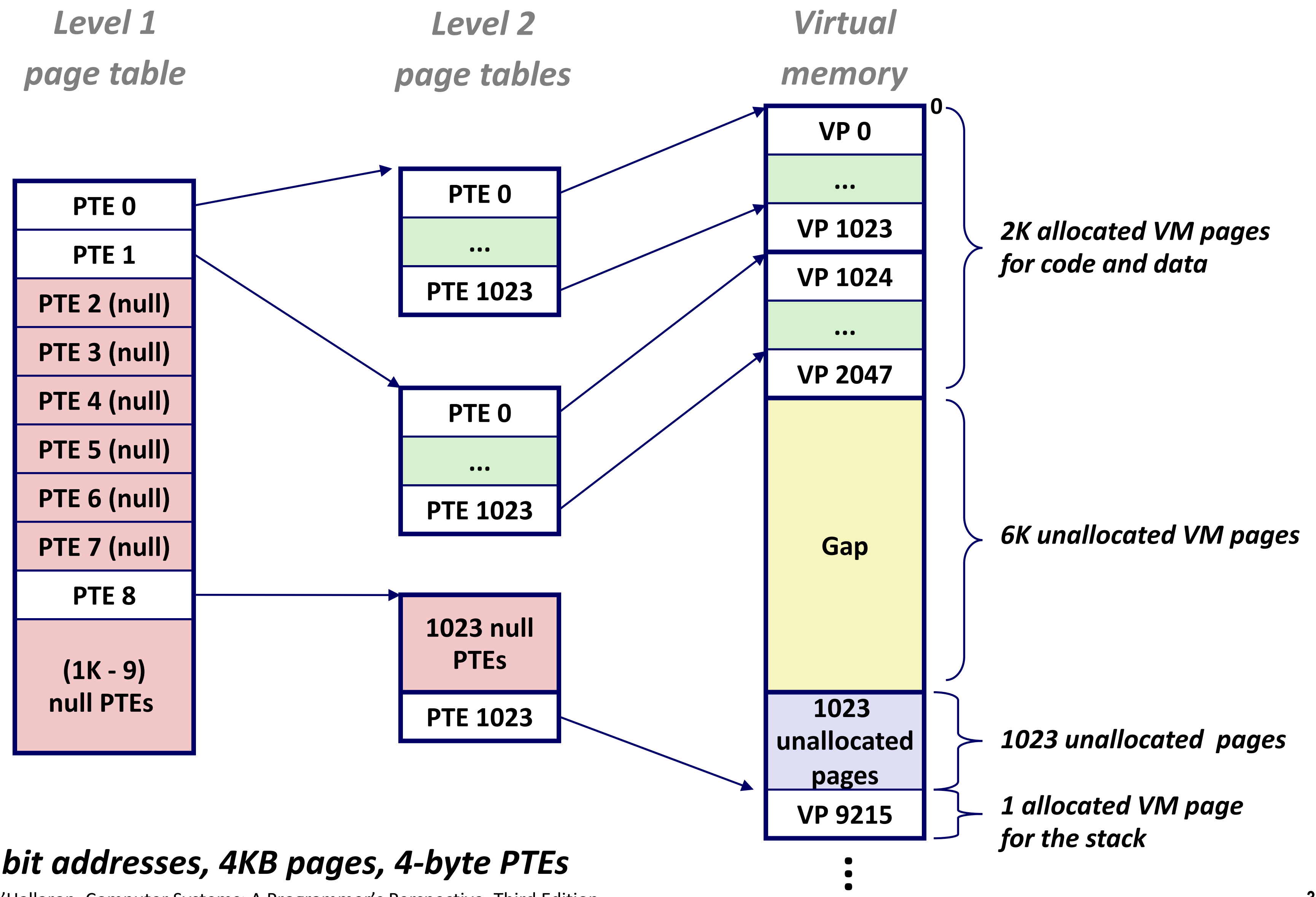
Fortunately, TLB misses are rare. Why?

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 table**
- **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)



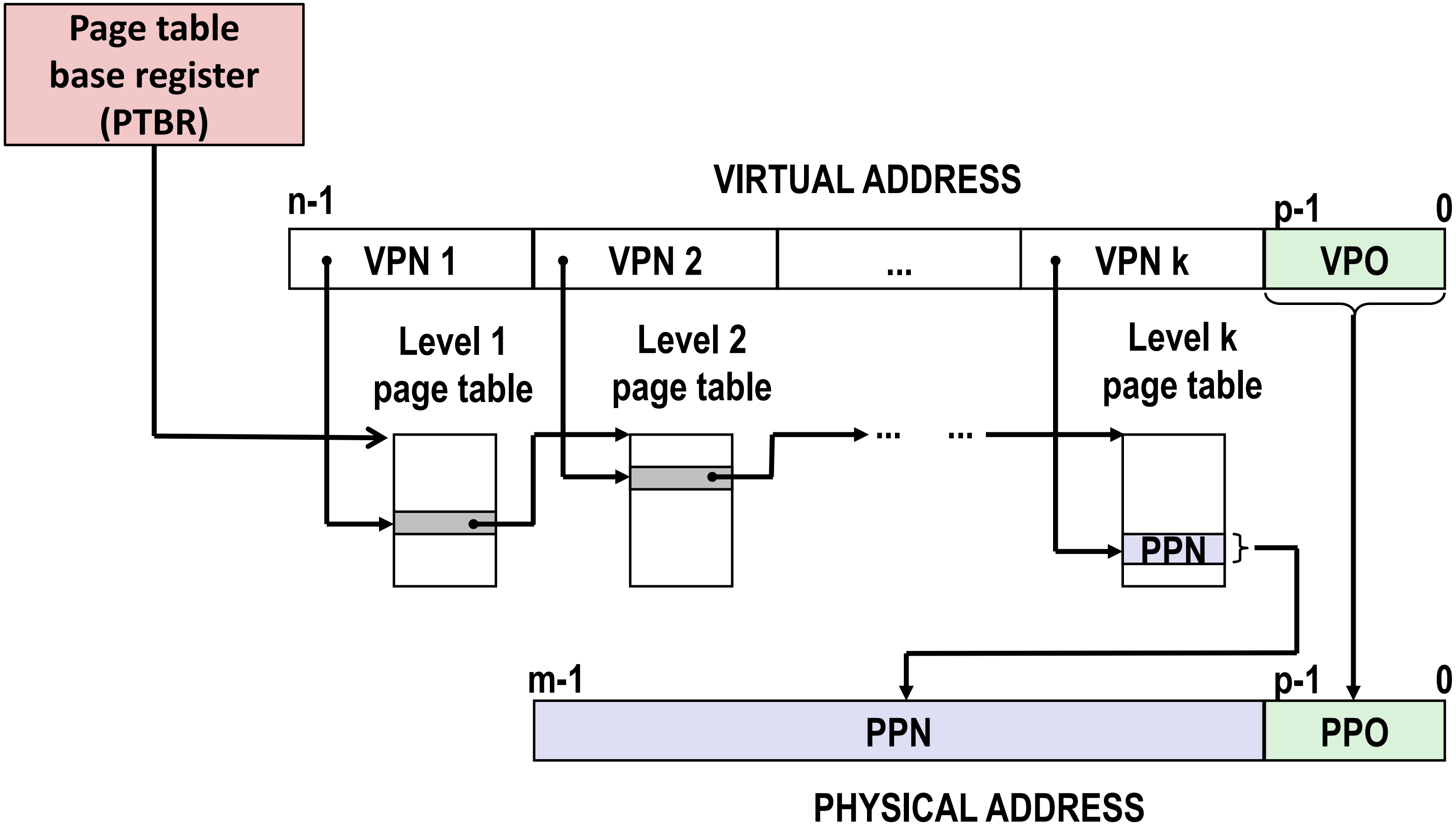
A Two-Level Page Table Hierarchy



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

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

Translating with a k-level Page Table



Summary

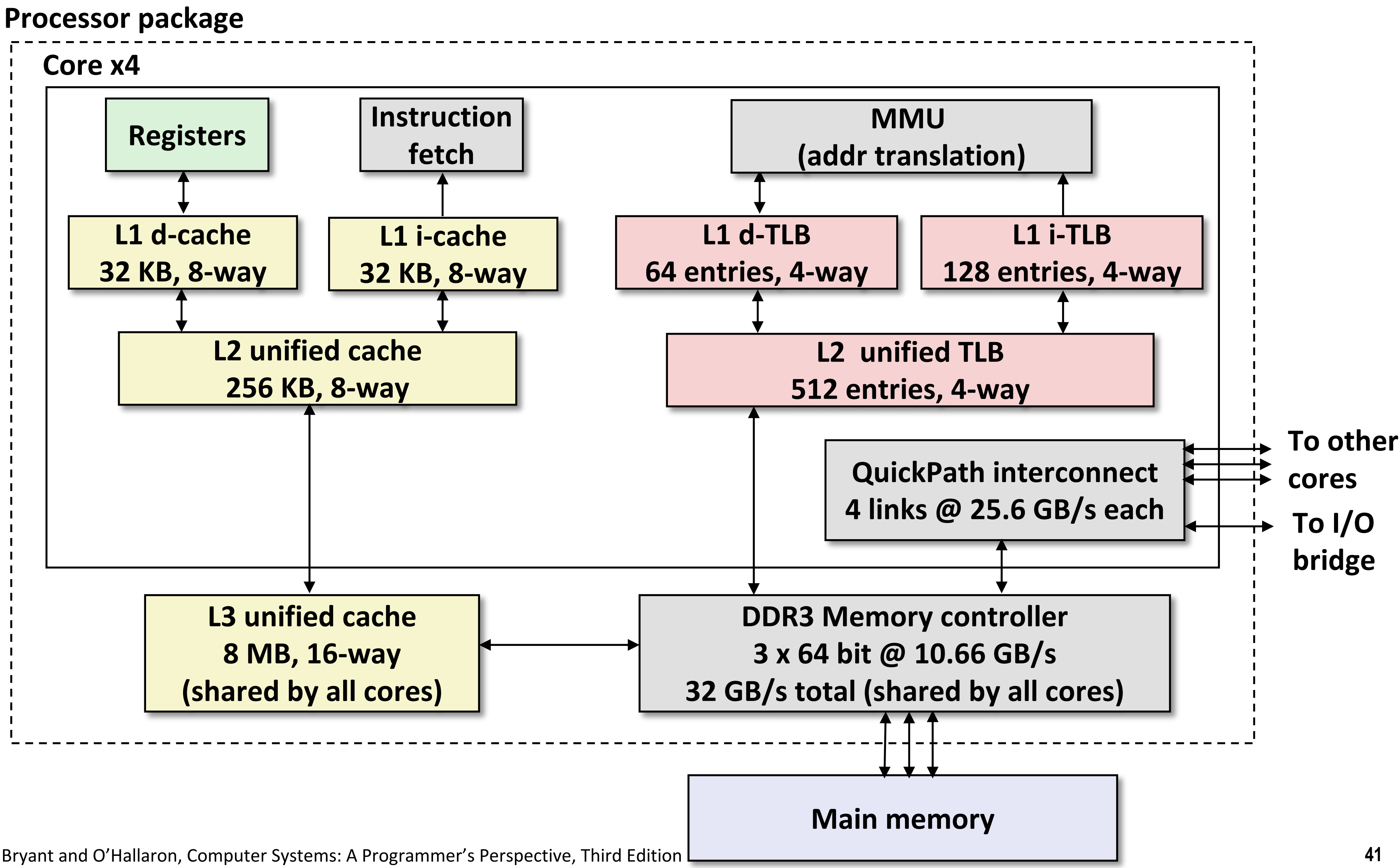
- **Programmer's view of virtual memory**
  - Each process has its own private linear address space
  - Cannot be corrupted by other processes

- **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

Extra Slides

- Case study: Core i7/Linux memory system

Intel Core i7 Memory System

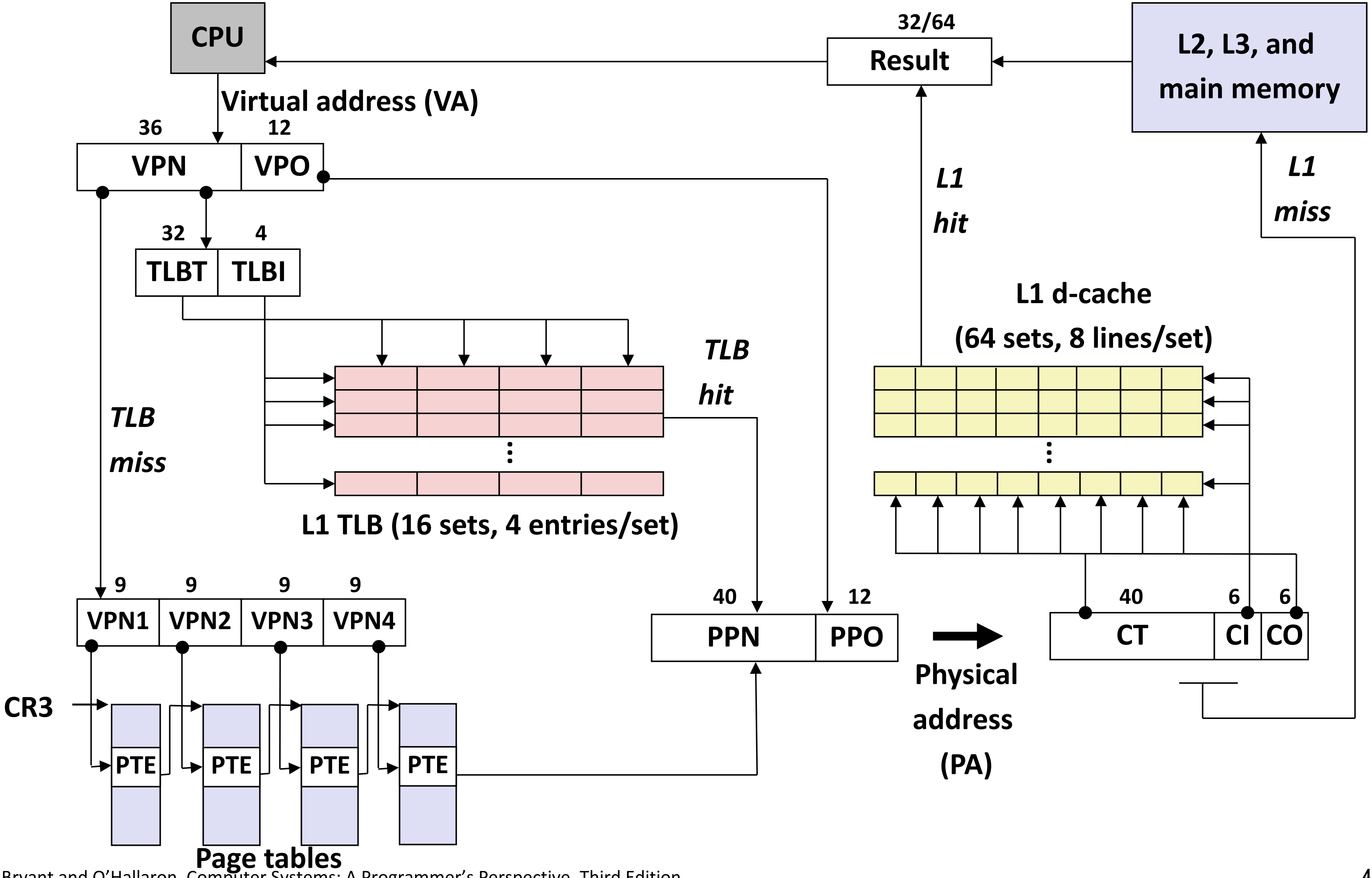

Processor package
Core x4
Registers
Instruction fetch
MMU (addr translation)
L1 d-cache 32 KB, 8-way
L1 i-cache 32 KB, 8-way
L1 d-TLB 64 entries, 4-way
L1 i-TLB 128 entries, 4-way
L2 unified cache 256 KB, 8-way
L2 unified TLB 512 entries, 4-way
QuickPath interconnect 4 links @ 25.6 GB/s each
To other cores
To I/O bridge
L3 unified cache 8 MB, 16-way (shared by all cores)
DDR3 Memory controller 3 x 64 bit @ 10.66 GB/s 32 GB/s total (shared by all cores)
Main memory

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

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

End-to-end Core i7 Address Translation



Core i7 Level 1-3 Page Table Entries

63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
XD	Unused	Page table physical base address	Unused	G	PS		A	CD	WT	U/S	R/W	P=1

Available for OS (page table location on disk)	P=0

Each entry references a 4K child page table. Significant fields:

P: Child page table present in physical memory (1) or not (0).

R/W: Read-only or read-write access access permission for all reachable pages.

U/S: user or supervisor (kernel) mode access permission for all reachable pages.

WT: Write-through or write-back cache policy for the child page table.

A: Reference bit (set by MMU on reads and writes, cleared by software).

PS: Page size either 4 KB or 4 MB (defined for Level 1 PTEs only).

Page table physical base address: 40 most significant bits of physical page table address (forces page tables to be 4KB aligned)

XD: Disable or enable instruction fetches from all pages reachable from this PTE.

Core i7 Level 4 Page Table Entries

63	62 52	51 12	11 9	8	7	6	5	4	3	2	1	0
XD	Unused	Page physical base address	Unused	G		D	A	CD	WT	U/S	R/W	P=1

Available for OS (page location on disk)	P=0

Each entry references a 4K child page. Significant fields:

P: Child page is present in memory (1) or not (0)

R/W: Read-only or read-write access permission for child page

U/S: User or supervisor mode access

WT: Write-through or write-back cache policy for this page

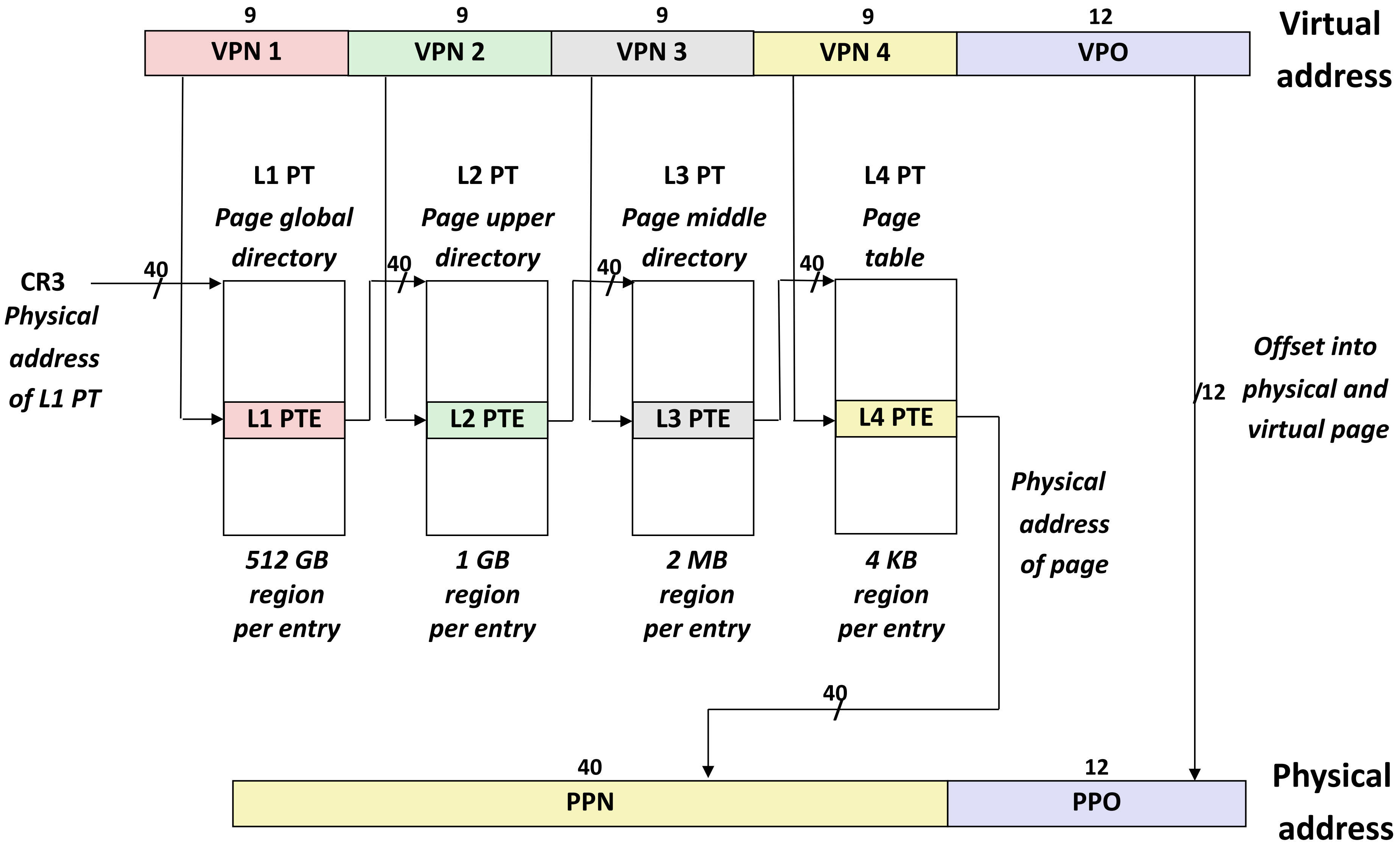
A: Reference bit (set by MMU on reads and writes, cleared by software)

D: Dirty bit (set by MMU on writes, cleared by software)

Page physical base address: 40 most significant bits of physical page address
(forces pages to be 4KB aligned)

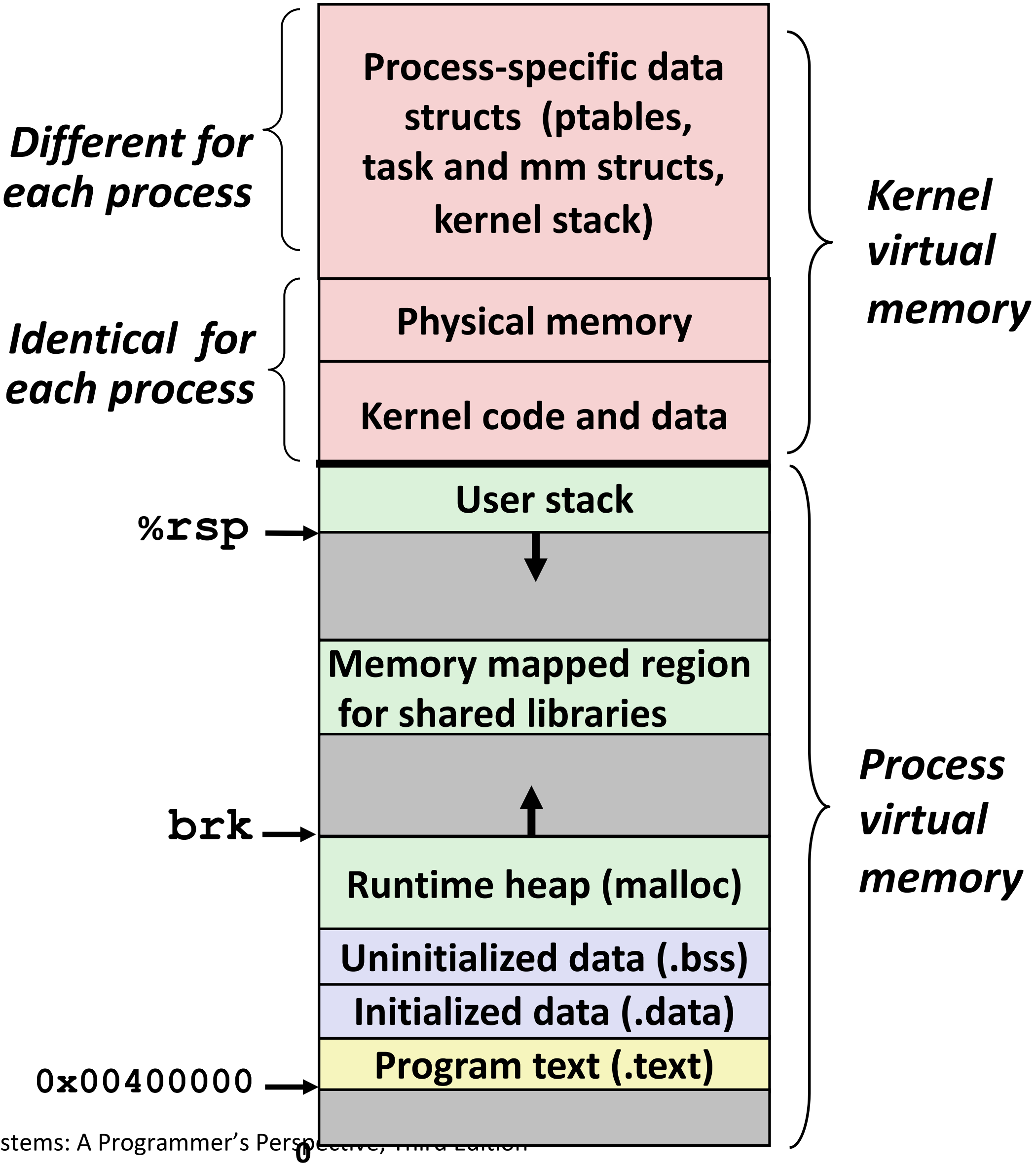
XD: Disable or enable instruction fetches from this page.

Core i7 Page Table Translation

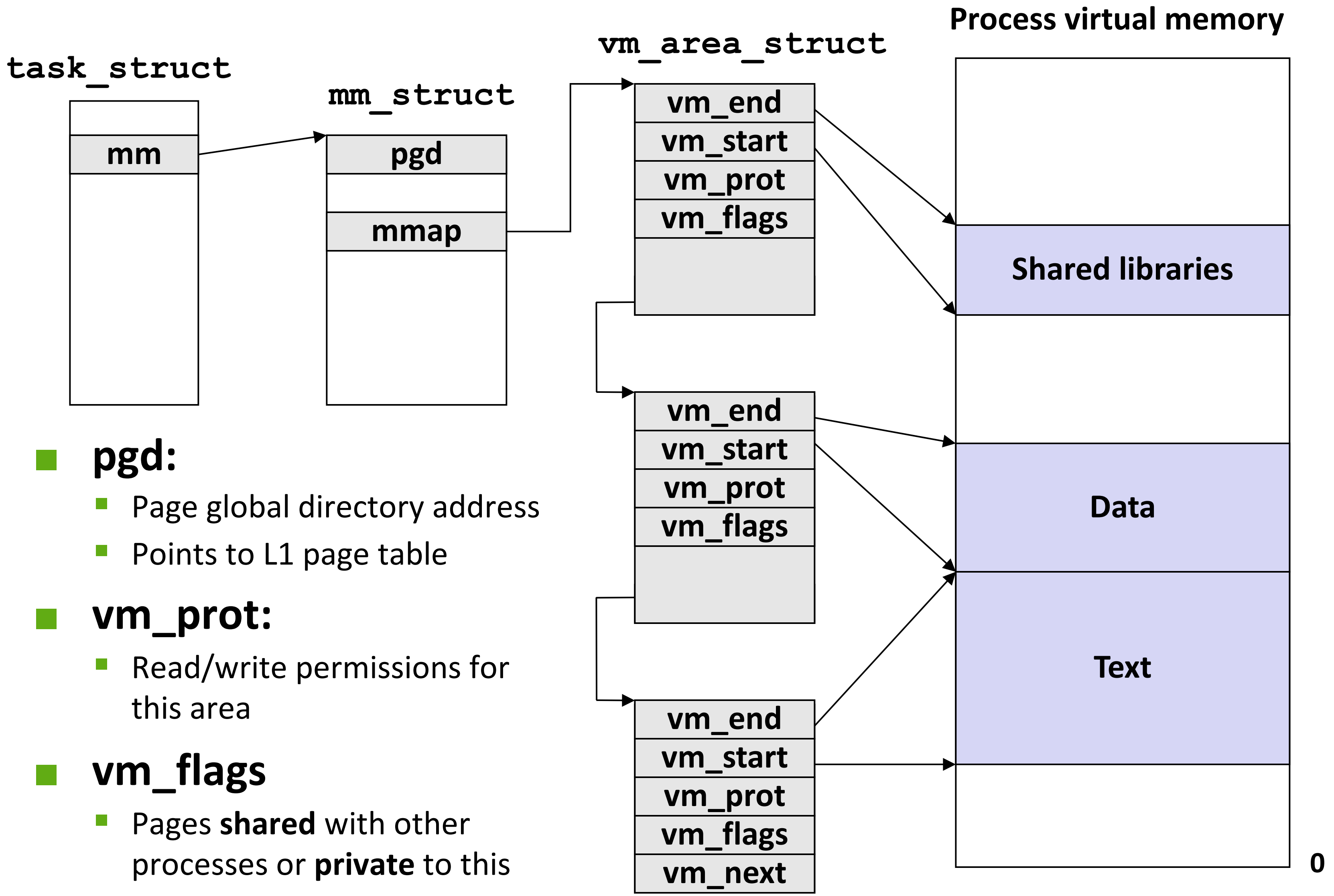


Bryant and O'Hallaron, Computer Systems: A Programmer's Perspective, Third Edition

Virtual Address Space of a Linux Process

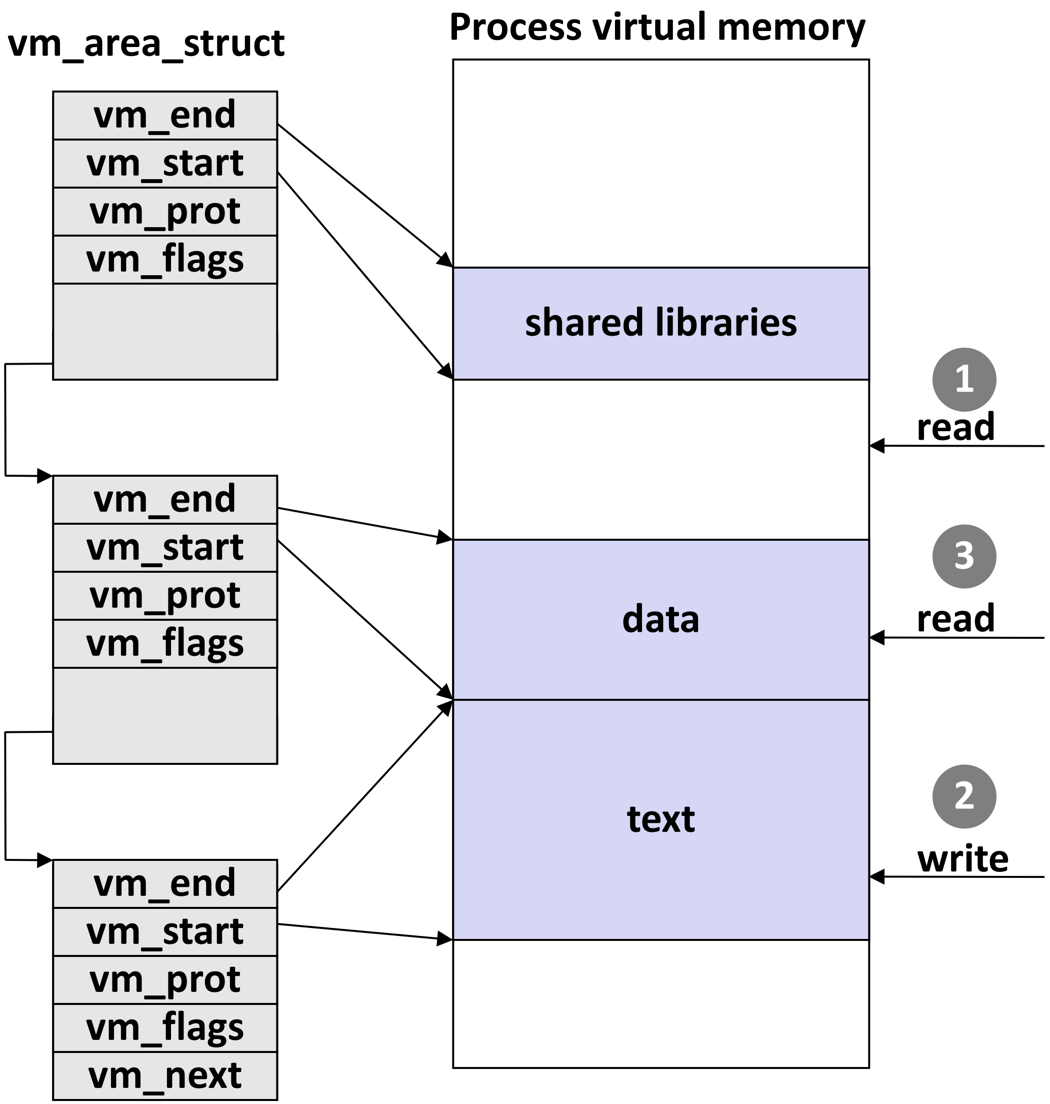

Different for each process
Identical for each process
Process-specific data structs (ptables, task and mm structs, kernel stack)
Physical memory
Kernel code and data
Kernel virtual memory
%rsp
User stack
Memory mapped region for shared libraries
brk
Runtime heap (malloc)
Uninitialized data (.bss)
Initialized data (.data)
0x00400000
Program text (.text)
0
Process virtual memory

Linux Organizes VM as Collection of "Areas"



- **pgd:**
  - Page global directory address
  - Points to L1 page table
- **vm_prot:**
  - Read/write permissions for this area
- **vm_flags**
  - Pages **shared** with other processes or **private** to this process

Linux Page Fault Handling



Segmentation fault:
accessing a non-existing page

Normal page fault

Protection exception:
e.g., violating permission by writing to a read-only page (Linux reports as Segmentation fault)

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