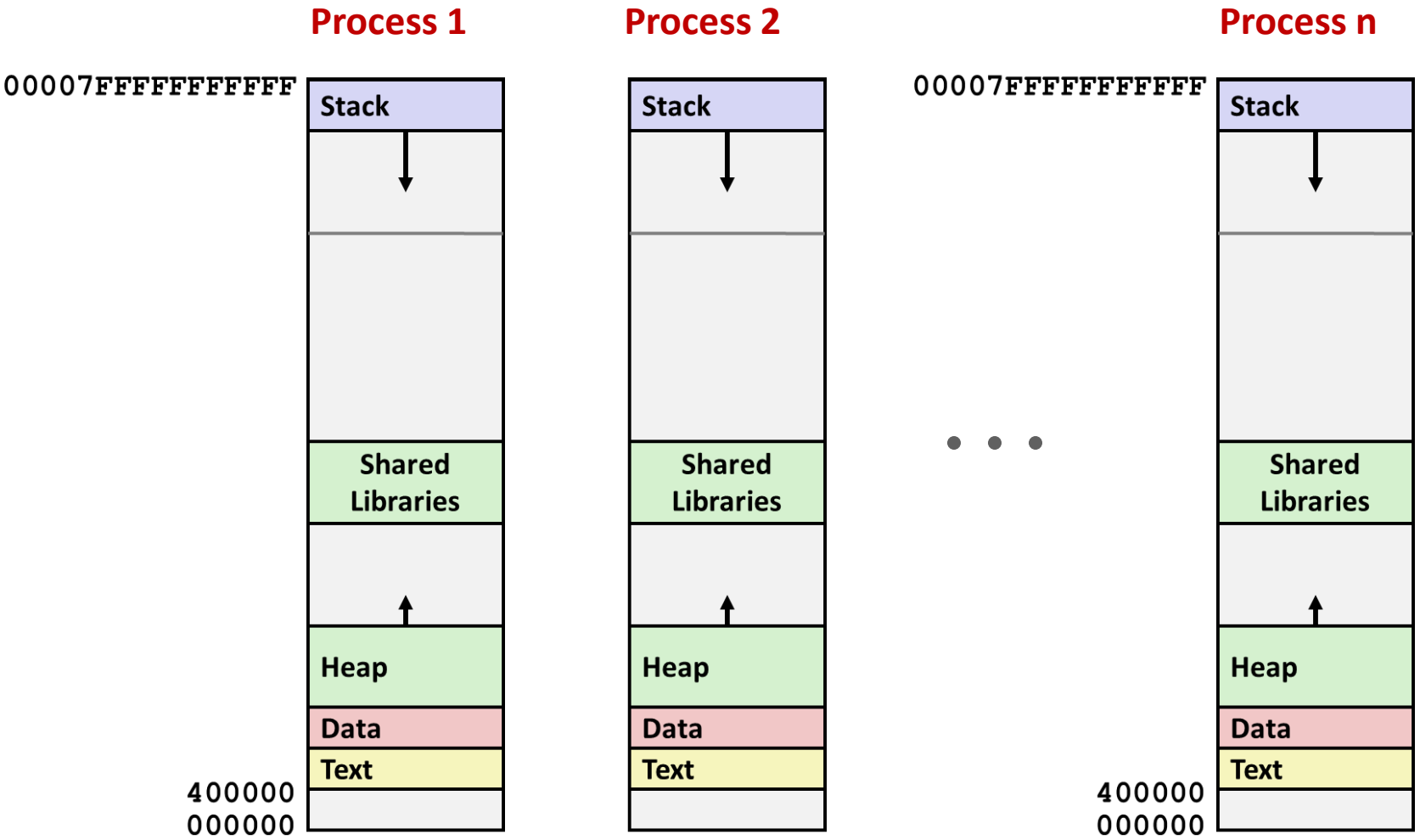
Systems Programming

Virtual Memory: Concepts

Byoungyoung Lee
Seoul National University
byoungyoung@snu.ac.kr
https://lifeasageek.github.io

Lecture slides are prepared based on materials provided by CSAPP authors.

Hmmm, How Does This Work?!

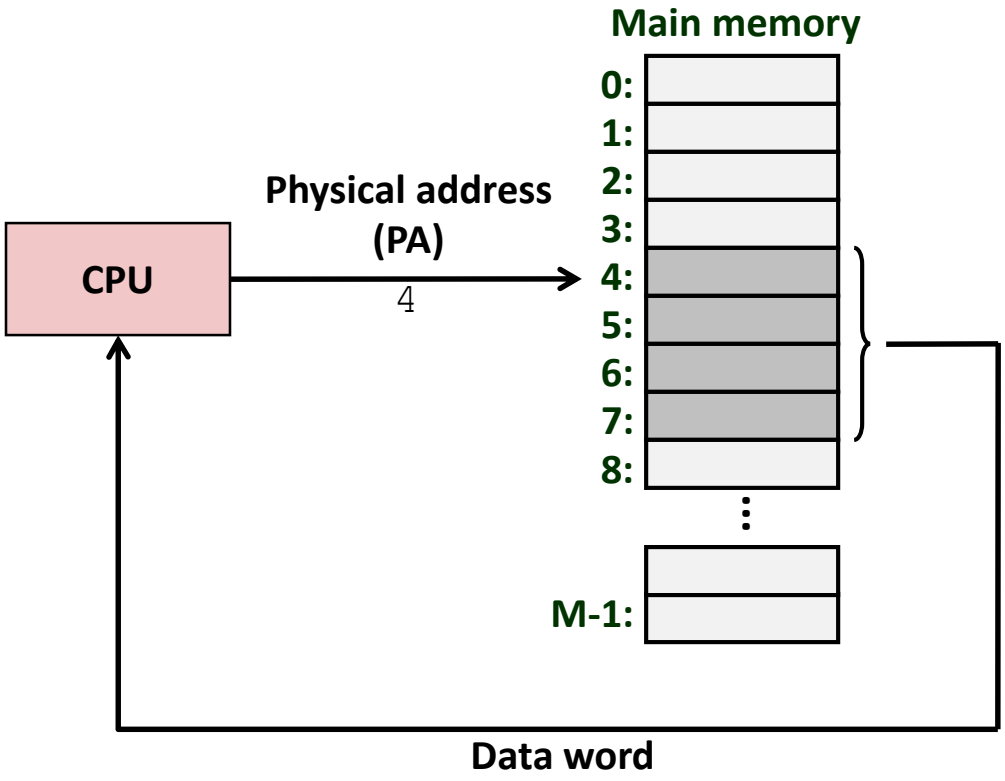


Solution: Virtual Memory

Today

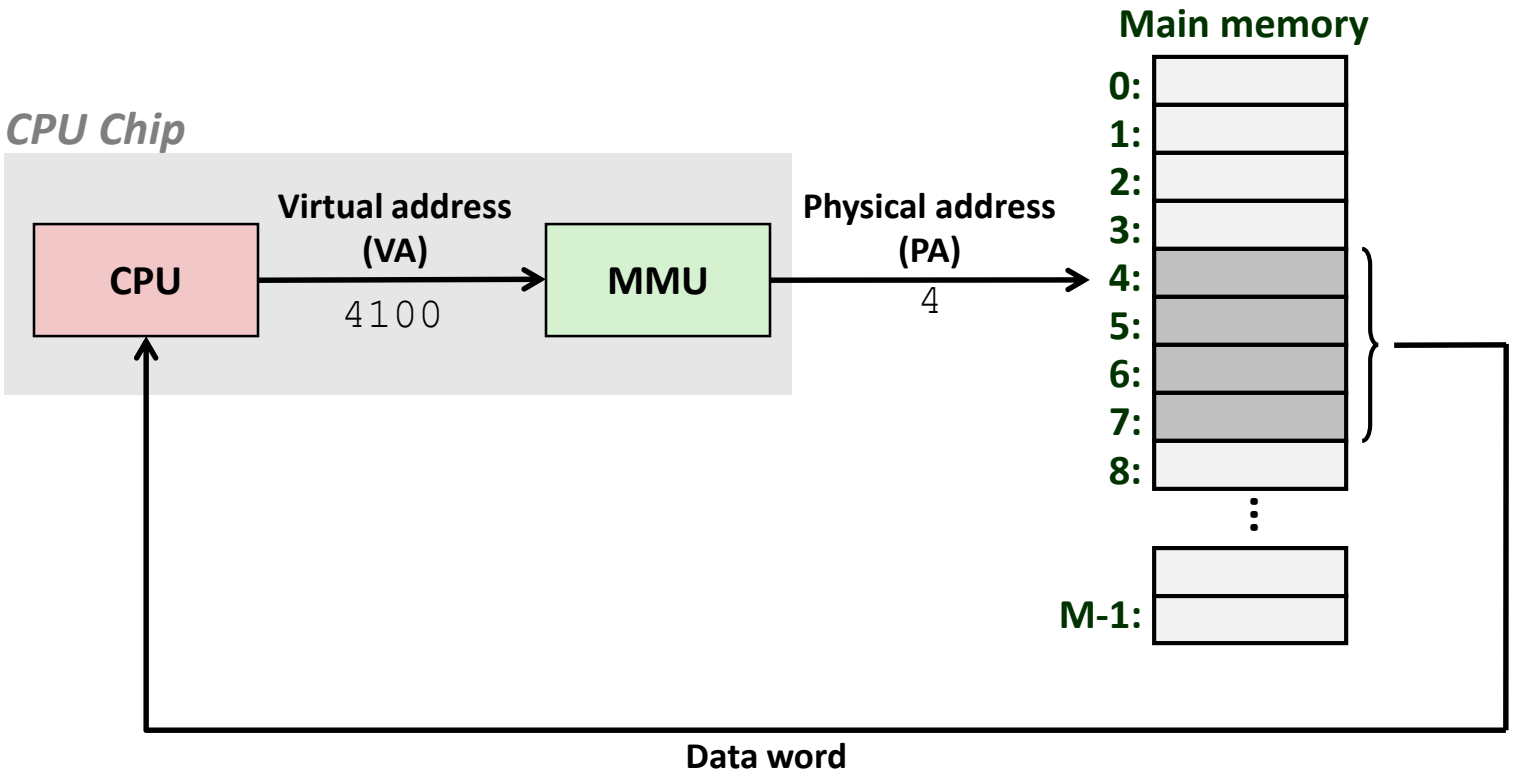
- **Address spaces** — CSAPP 9.1-9.2
- VM as a tool for caching — CSAPP 9.3
- VM as a tool for memory management — CSAPP 9.4
- VM as a tool for memory protection — CSAPP 9.5
- Address translation — CSAPP 9.6

Physical Addressing



- **Used in "simple" systems like embedded microcontrollers in devices**
  - ECUs in automobiles, elevators, sensors, and digital picture frames

Virtual Addressing



- Used in all modern servers, laptops, and smart phones
- One of the great ideas in computer science

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
  {0, 1, 2, 3, …, M-1}

- **In practice, N >> M**

Why Virtual Memory (VM)?

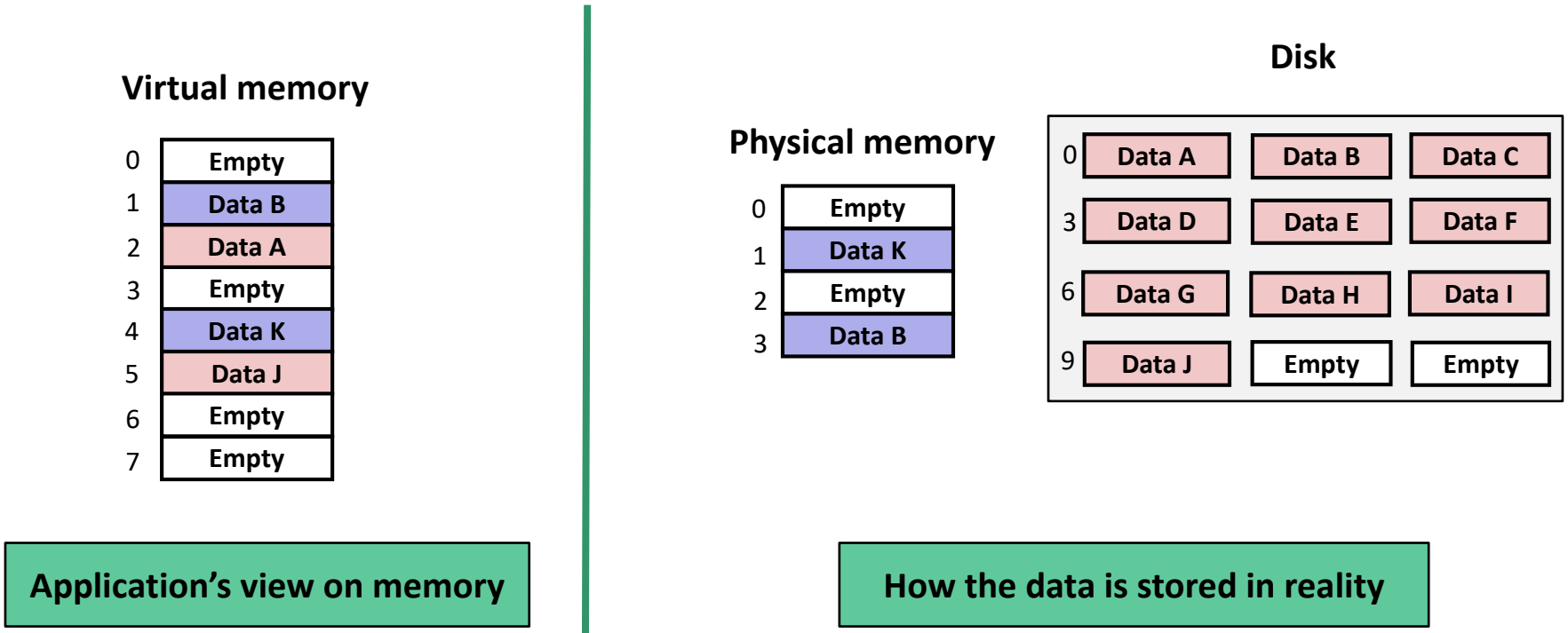
- **Uses main memory efficiently**
  - Addressing more than physical memory (e.g., DRAM) has
- **Simplifies memory management**
  - Each process gets the same uniform address space
- **Isolates address spaces**
  - One process can't interfere with another's memory
  - User program cannot access privileged 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

- **All the data is initially stored in disk**
- **The data in disk is cached in *physical memory* (*DRAM*)**
  - These cache blocks are called *pages* (the size of each page is 2^p bytes)
  - This DRAM cache is not CPU cache!
- **Virtual memory provides imaginary/virtual memory view for applications**



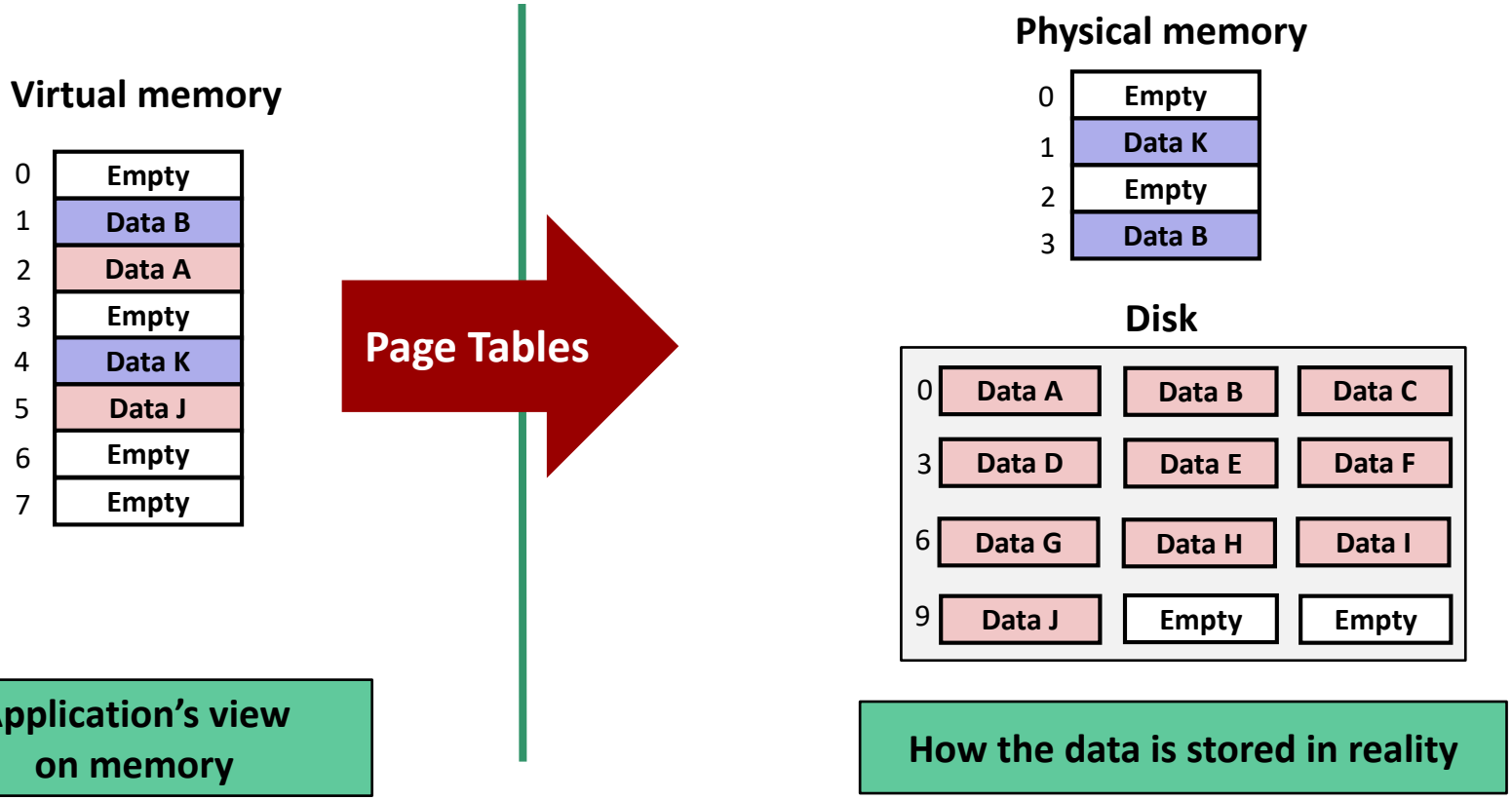
Design Consideration on Caching

- **DRAM is much faster than disk**
  - Disk is about ***10,000x*** slower than DRAM

- **Consequences**
  - Fully associative
    - Any virtual page (VP) can be placed in any physical page (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

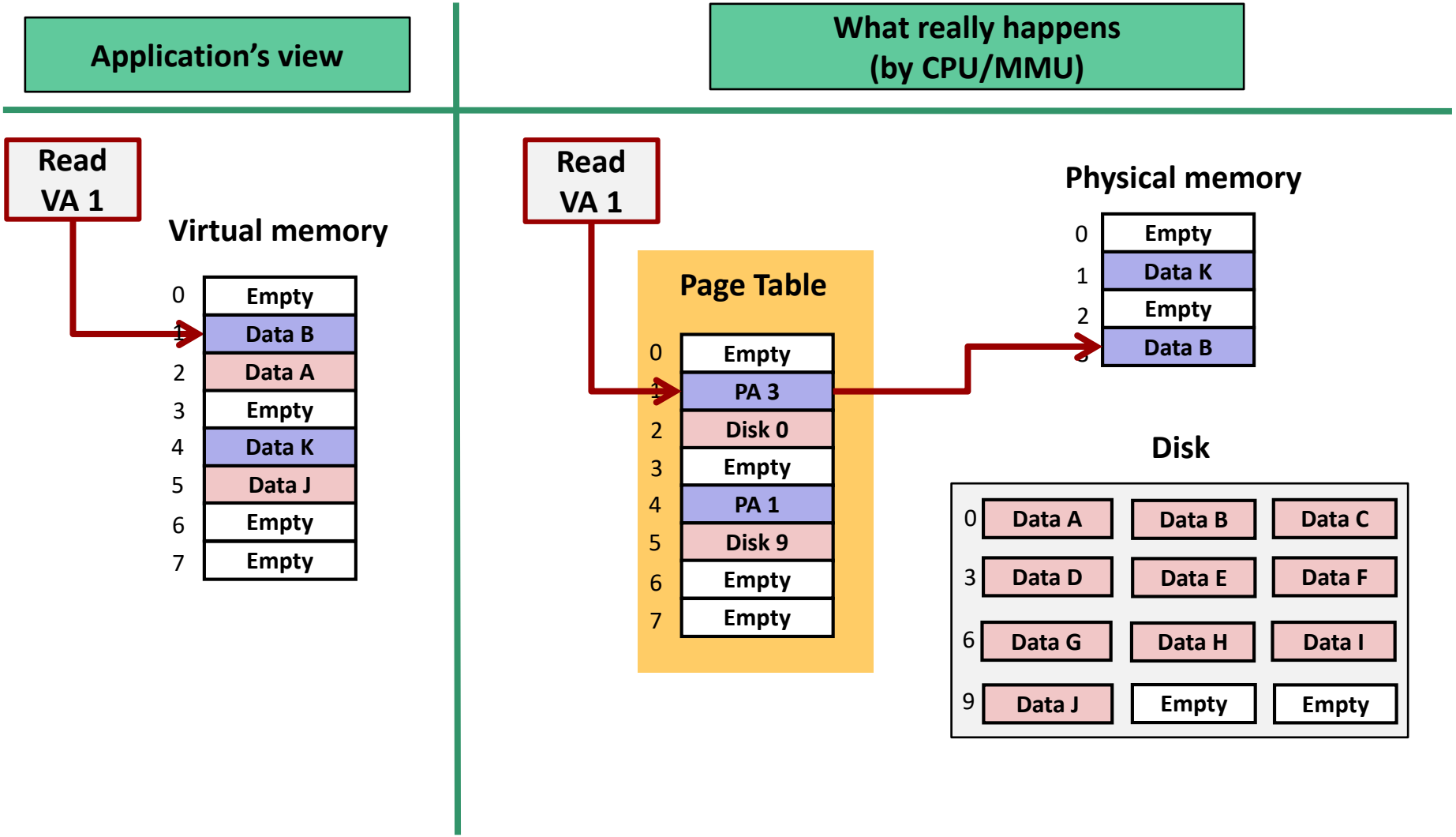
Enabling Data Structure: Page Table

- **A *page table* is an array of page table entries (PTEs)**
  - Per-process kernel data structure in the main memory
- **A page table maps virtual memory to physical memory.**
  - It maps from "imaginary memory view" to "real-world memory view"
  - **Note: page tables do not map to the disk, but here we simply assumed so for simplicity**



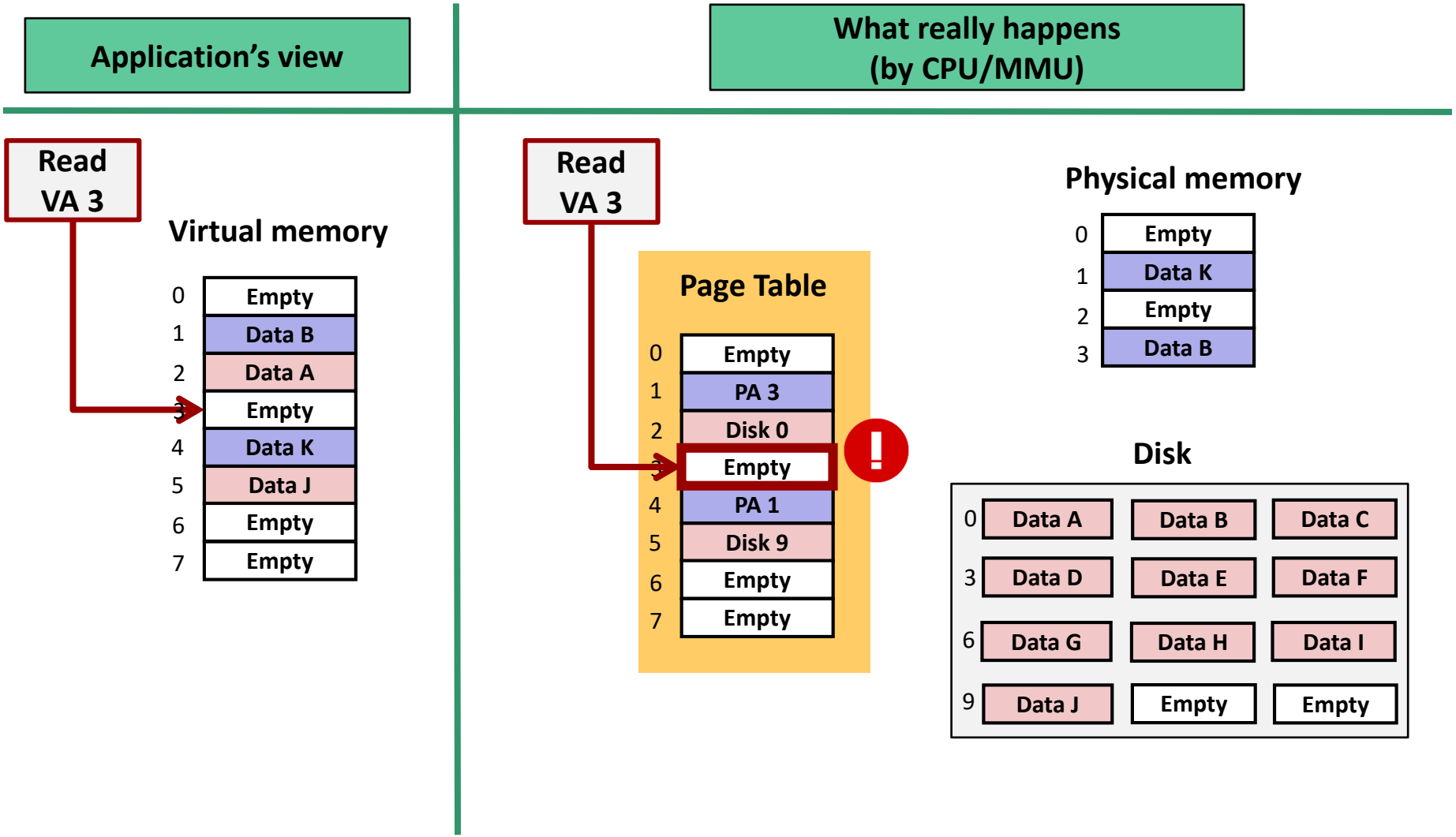
Page Hit

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



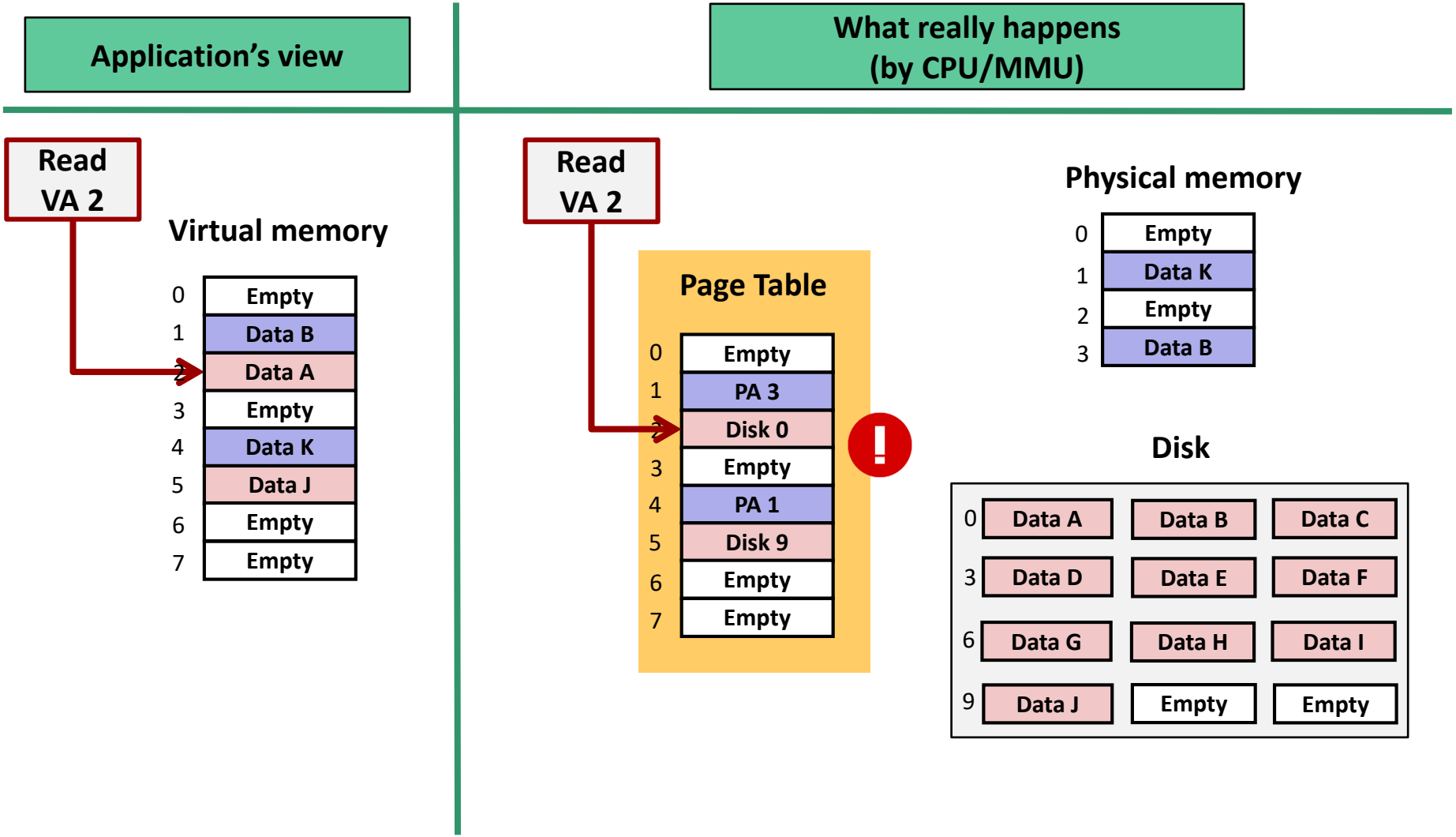
Page Fault (1)

- *Page fault:* reference to VM word that is neither in physical memory nor disk



Page Fault (2)

- *Page fault:* reference to VM word, not in physical memory but in disk

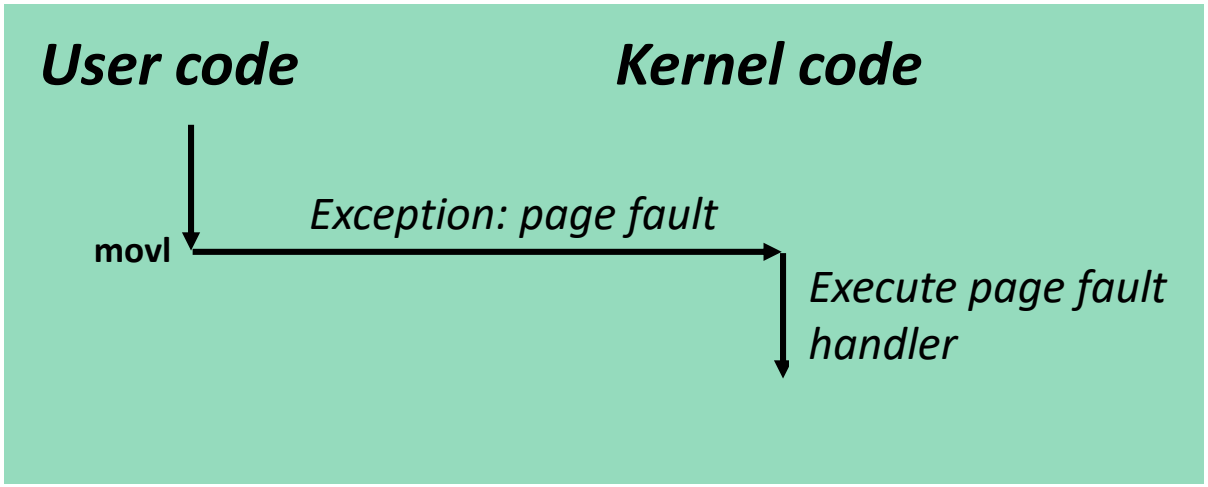


Triggering a Page Fault

- User writes to memory location

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

- That portion (page) of user's memory may be stored in disk
- MMU triggers page fault exception
  - (More details in later lecture)
  - Raise privilege level to supervisor mode
  - Causes procedure call to software page fault handler

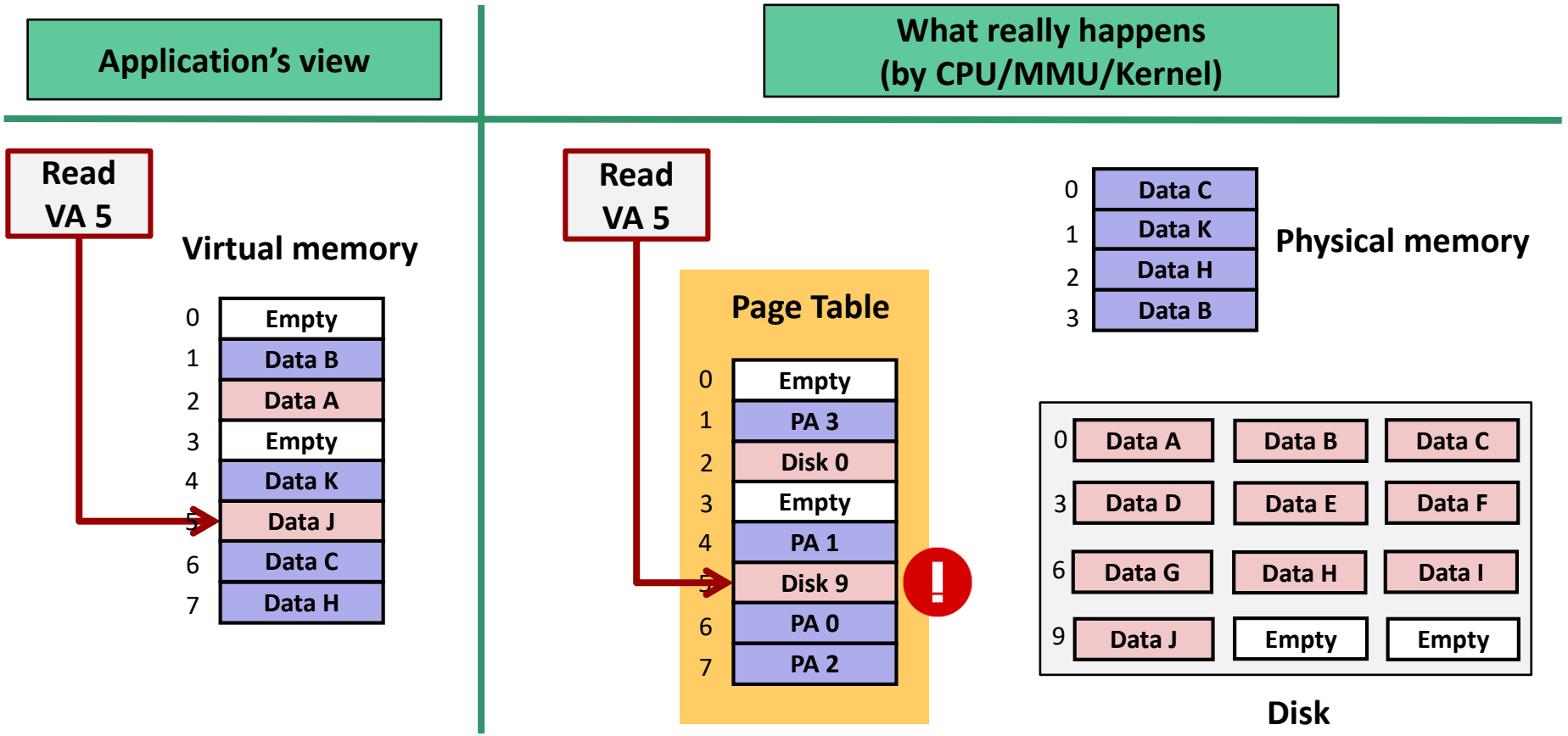


Handling Page Fault

- **If data is neither in physical memory nor in disk**
  - A page fault cannot be recovered
  - The user code may need to be terminated
- **If data is not in physical memory but in disk**
  - A page fault can be recovered
  - The kernel code will load the data in disk to the physical memory
  - Then re-execute the instruction, which raised the page fault

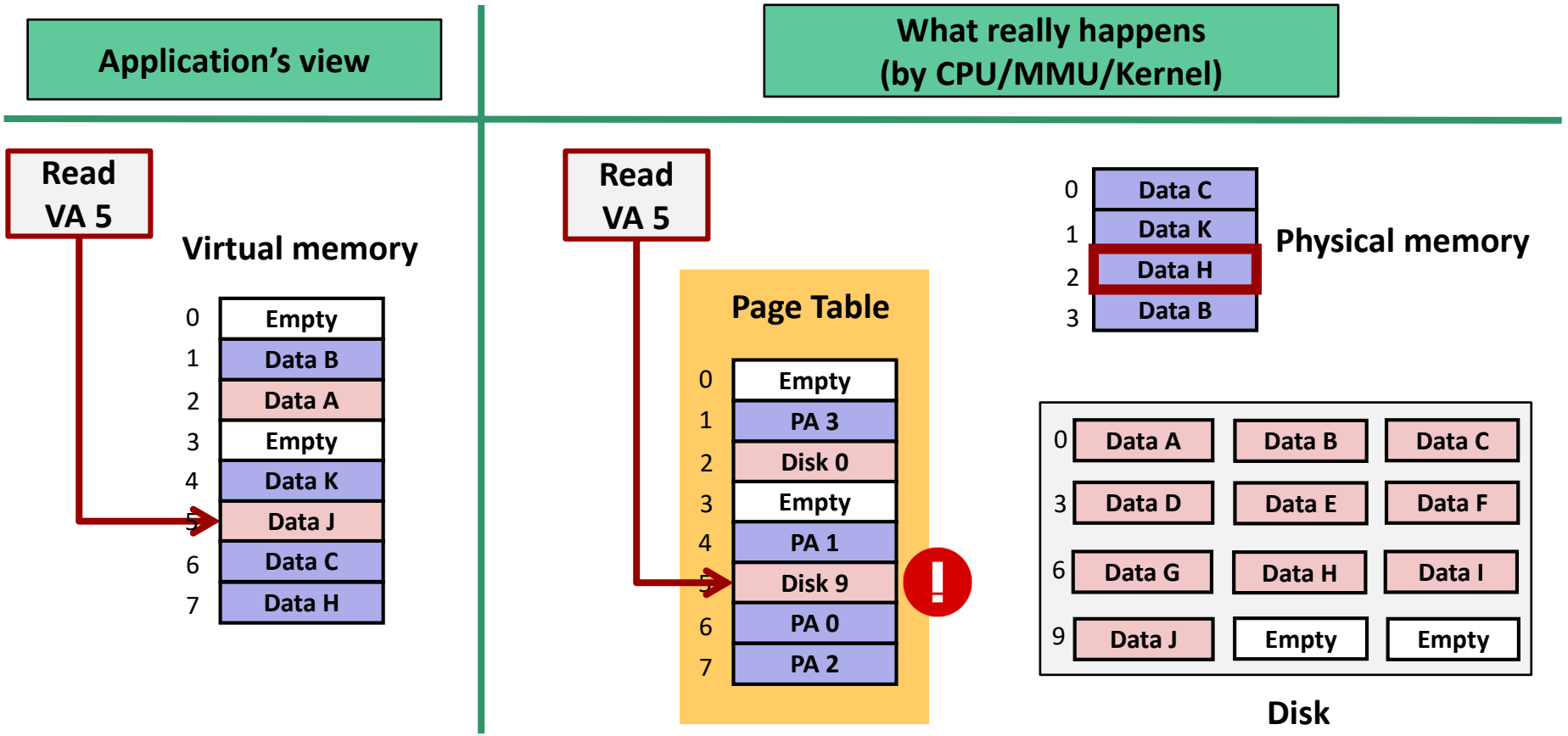
Handling Page Fault (1)

- Page miss causes page fault (an exception)



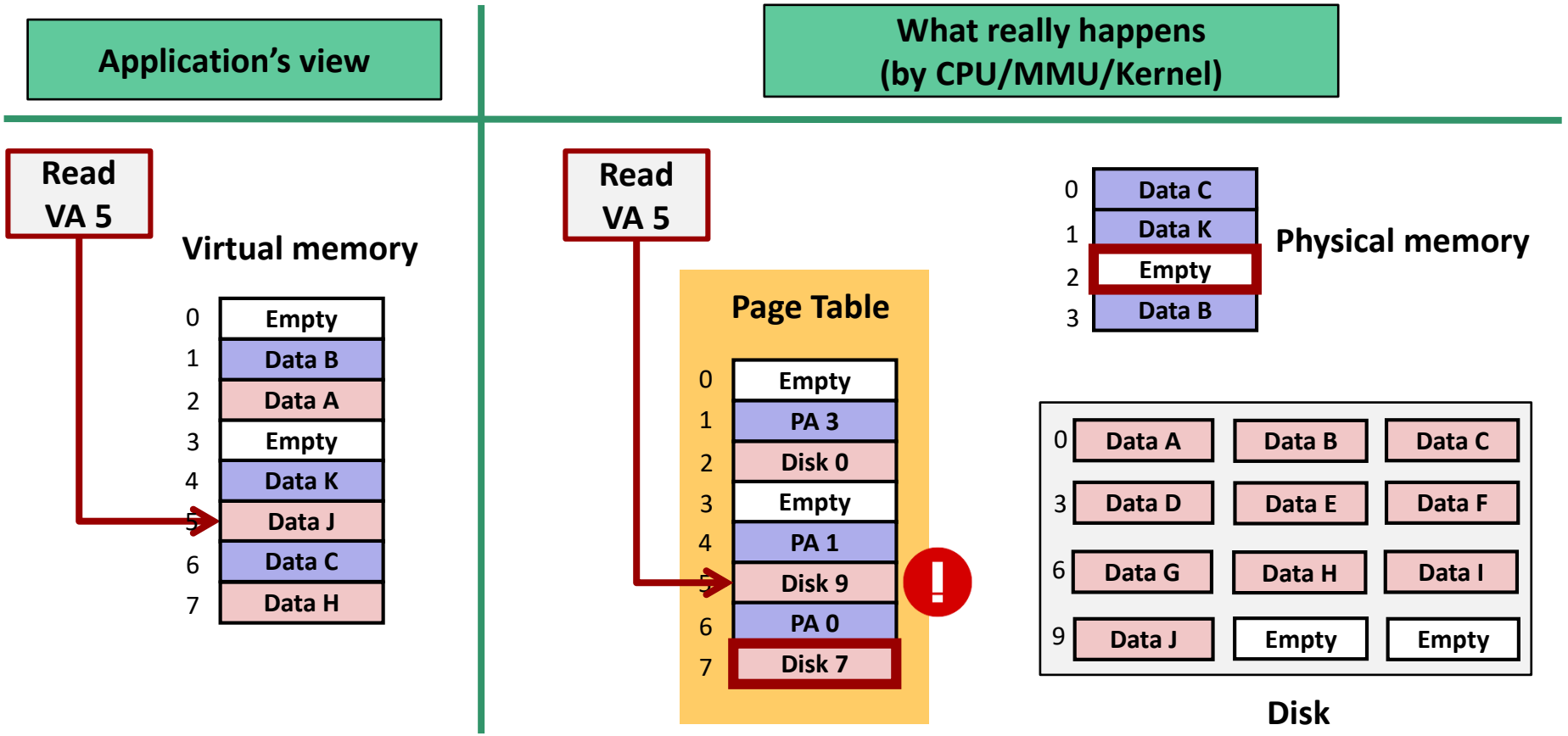
Handling Page Fault (2)

- Page miss causes page fault (an exception)
- **Page fault handler selects a victim to be evicted (here "Data H" at "VA 7" or "PA 2")**



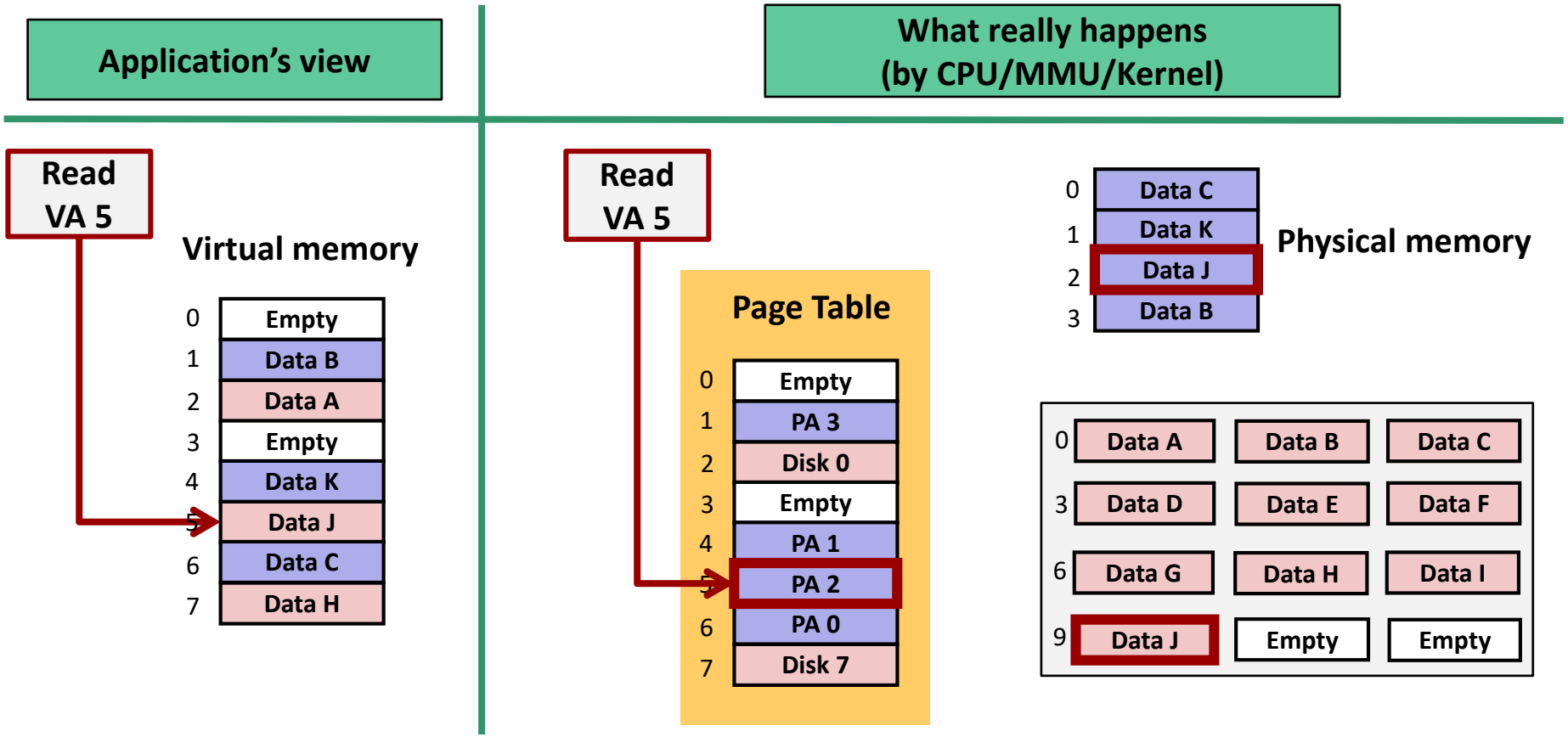
Handling Page Fault (3)

- Page miss causes page fault (an exception)
- **Page fault handler selects a victim to be evicted (here "Data H" at "VA 7" or "PA 2")**



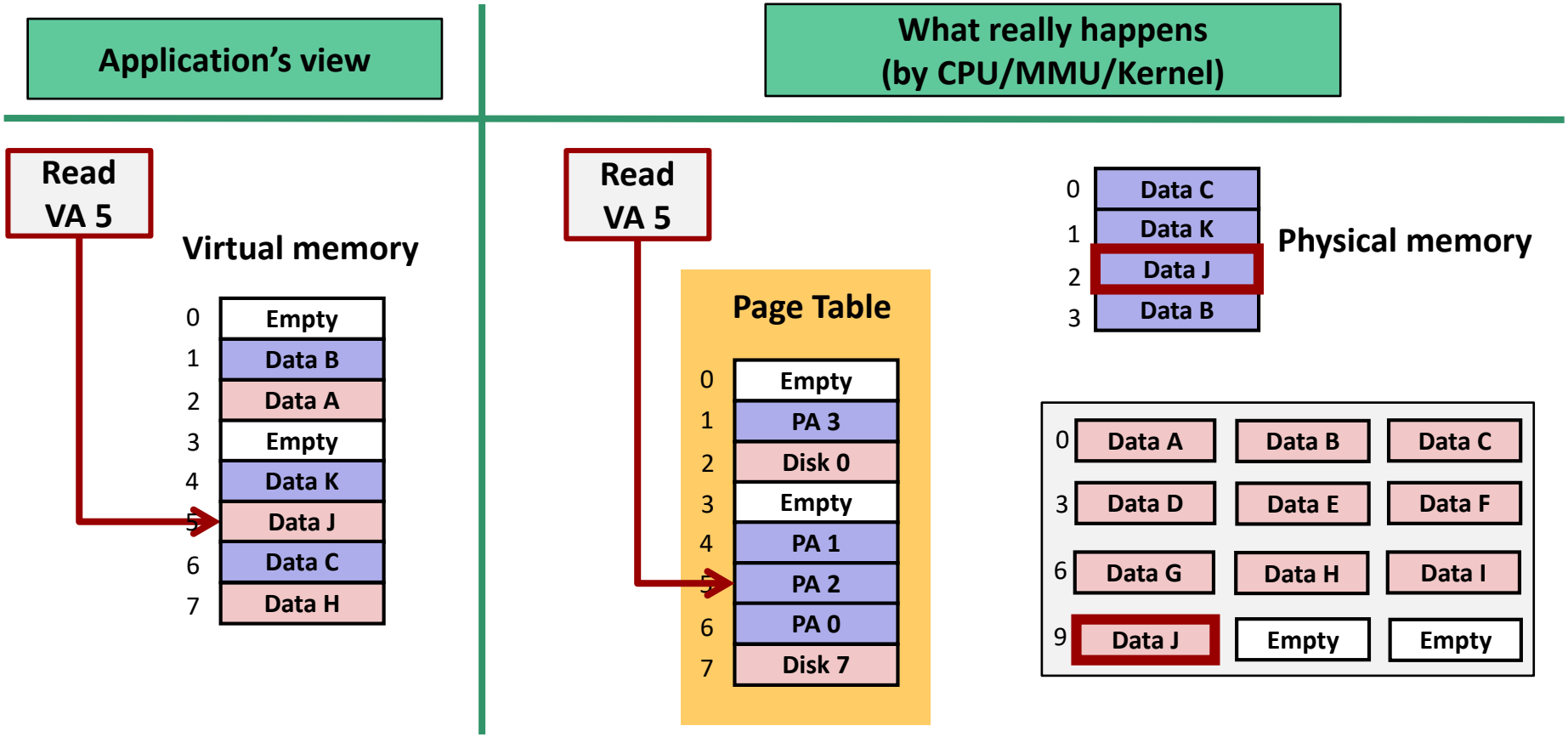
Handling Page Fault (4)

- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here “Data H” at “VA 7” or “PA 2”)
- **Page fault handler loads the target page from disk (here “Data J” at “Disk 9”)**



Handling Page Fault (5)

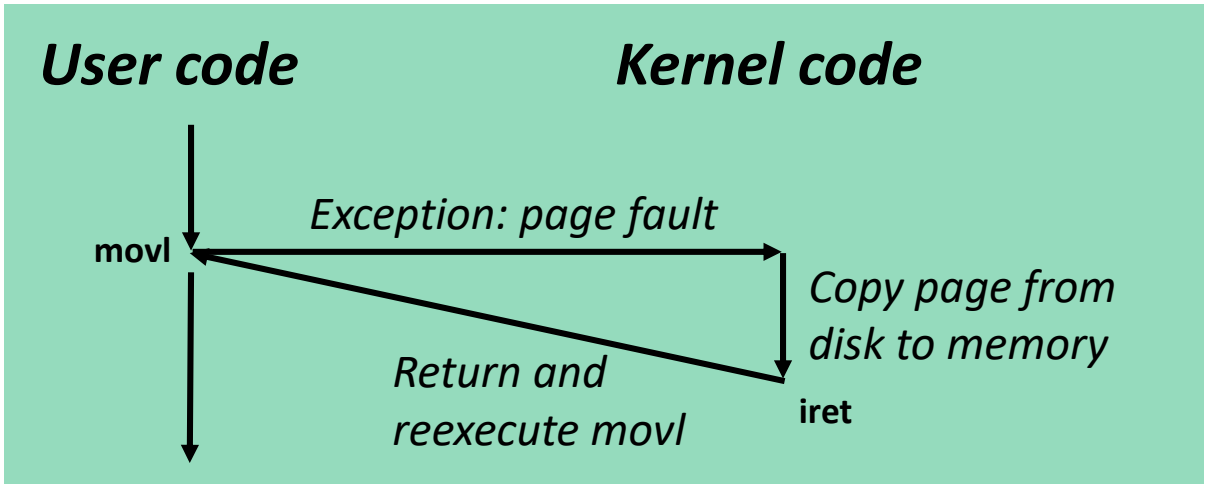
- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here “Data H” at “VA 7” or “PA 2”)
- Page fault handler loads the target page from disk (here “Data J” at “Disk 9”)
- **Offending instruction is restarted: page hit!**



Completing page fault

- Page fault handler executes “return from interrupt” (**`iret`**) instruction
  - Like **`ret`** instruction, but also restores privilege level
  - Return to instruction that caused fault
  - But, this time there is no page fault

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



Locality to the Rescue Again!

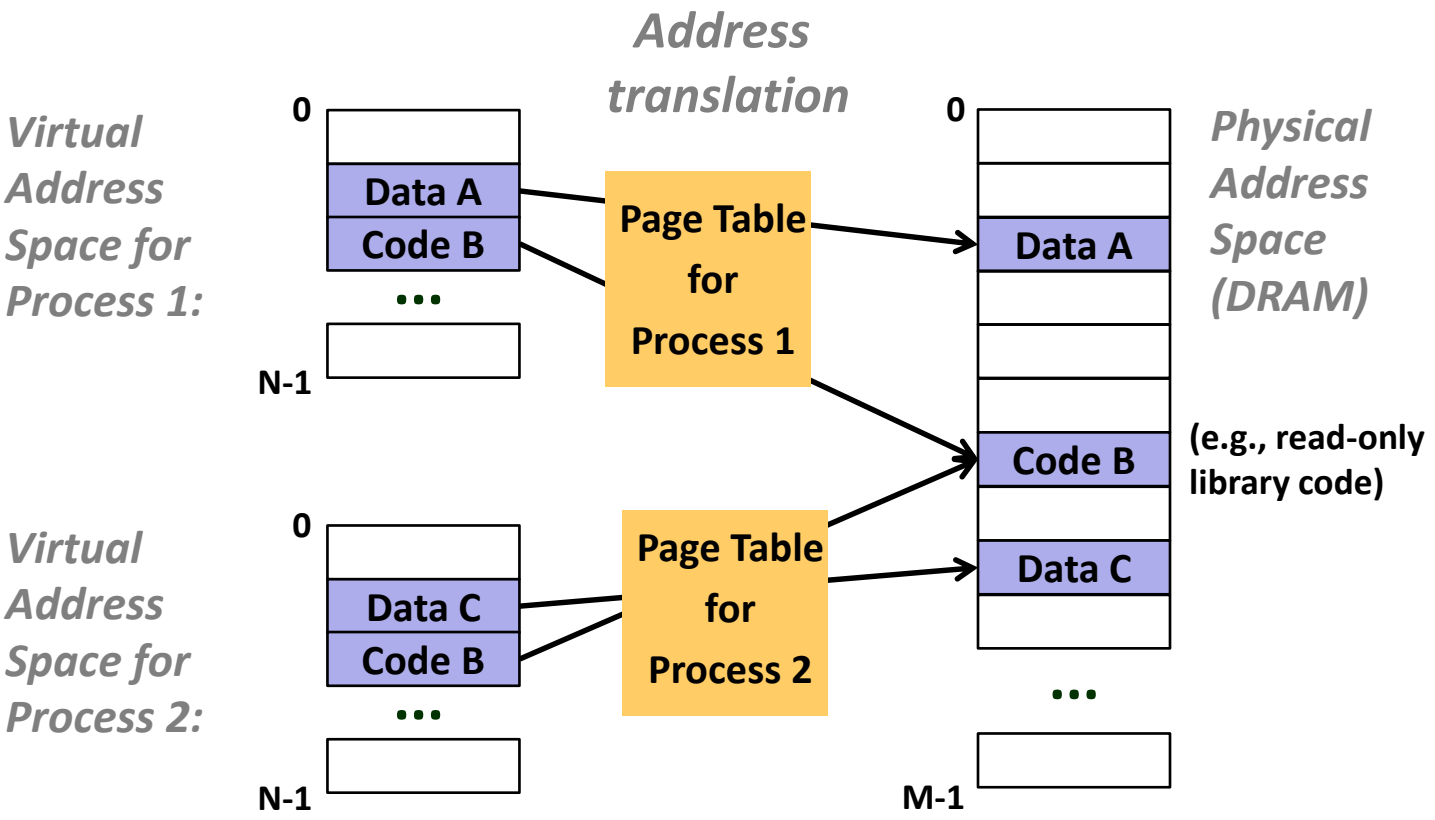
- **In practice, VM with paging surprisingly efficient**
  - because of **locality**.
- **At any point in time, programs tend to access a set of active virtual pages called the *working set***
  - Programs with better temporal locality will have smaller working sets
- **If (working set size < main memory size)**
  - Good performance for one process
- **If (working set size > main memory size)**
  - *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously
  - If multiple processes run at the same time, thrashing occurs if their total working set size > main memory size

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

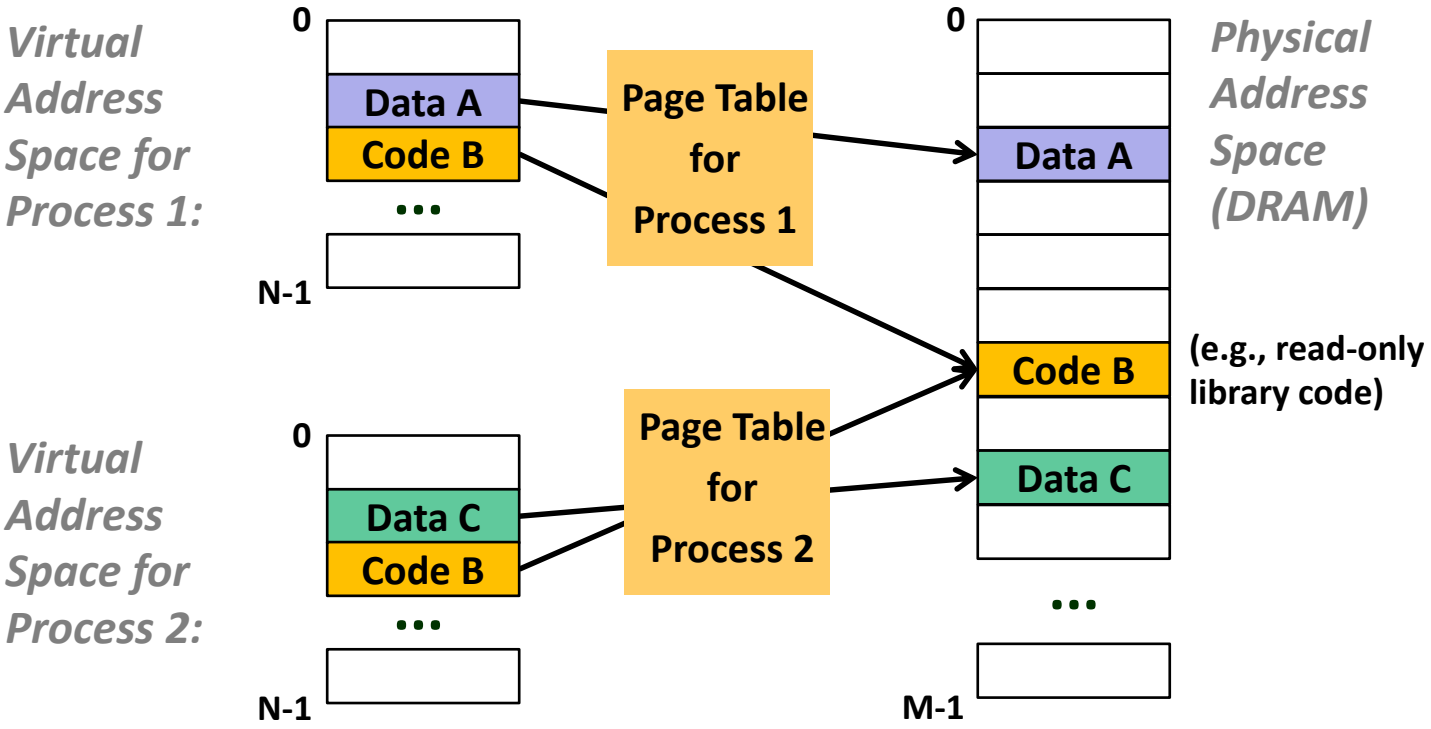
VM as a Tool for Memory Management

- **Key idea: each process has its own virtual address space**
  - It can view memory as a simple linear array



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 migrated to a different physical page
- **Sharing code and data among processes**
  - Map multiple virtual pages to the same physical page (here: PP 6)

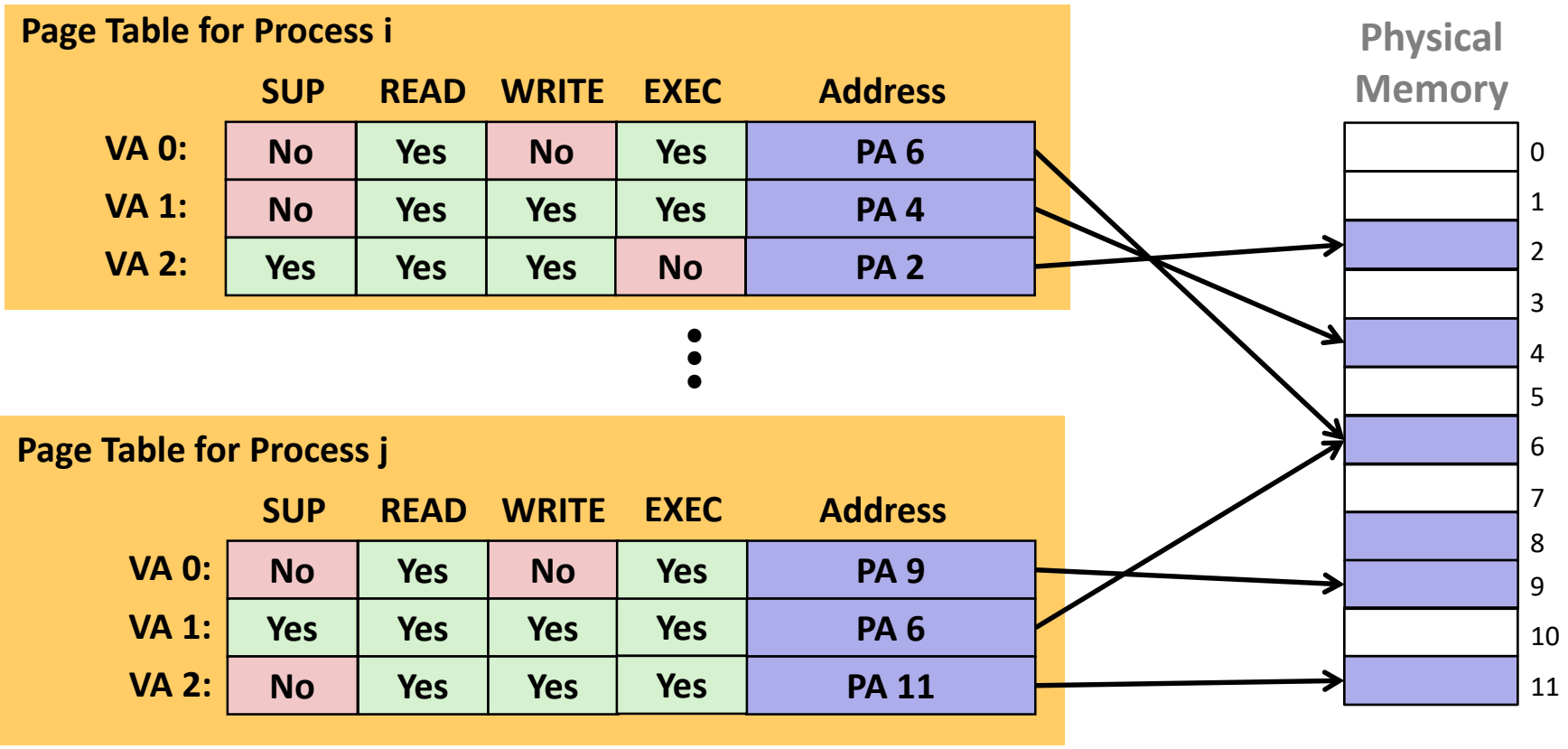


Today

- 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

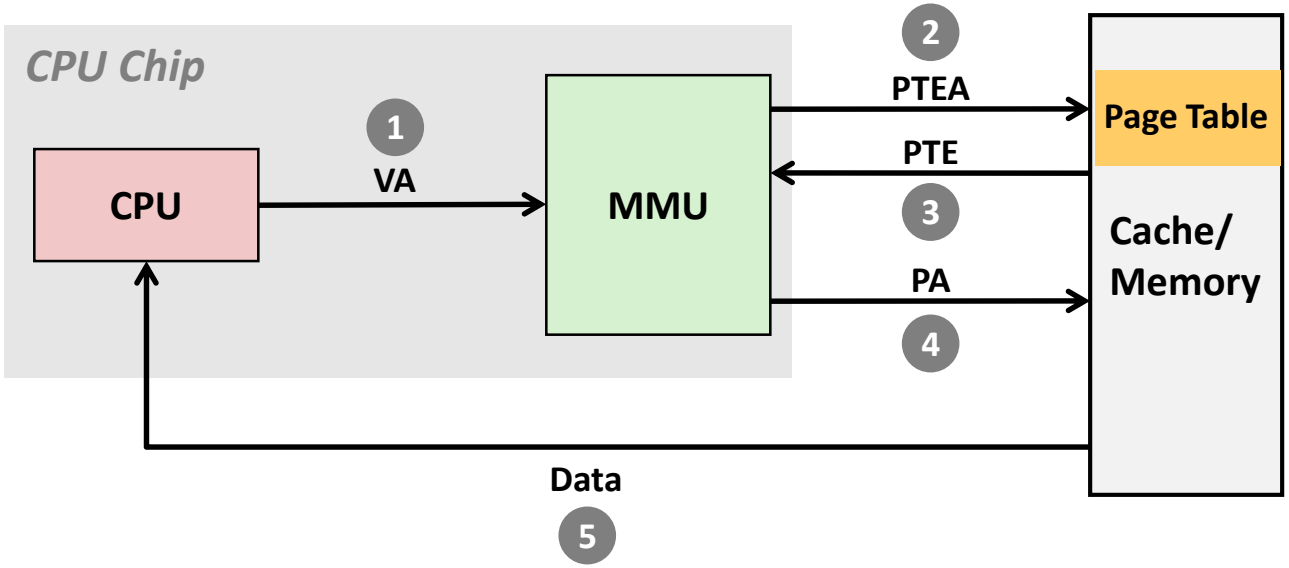


SUP: requires kernel mode

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**

Address Translation: Page Hit



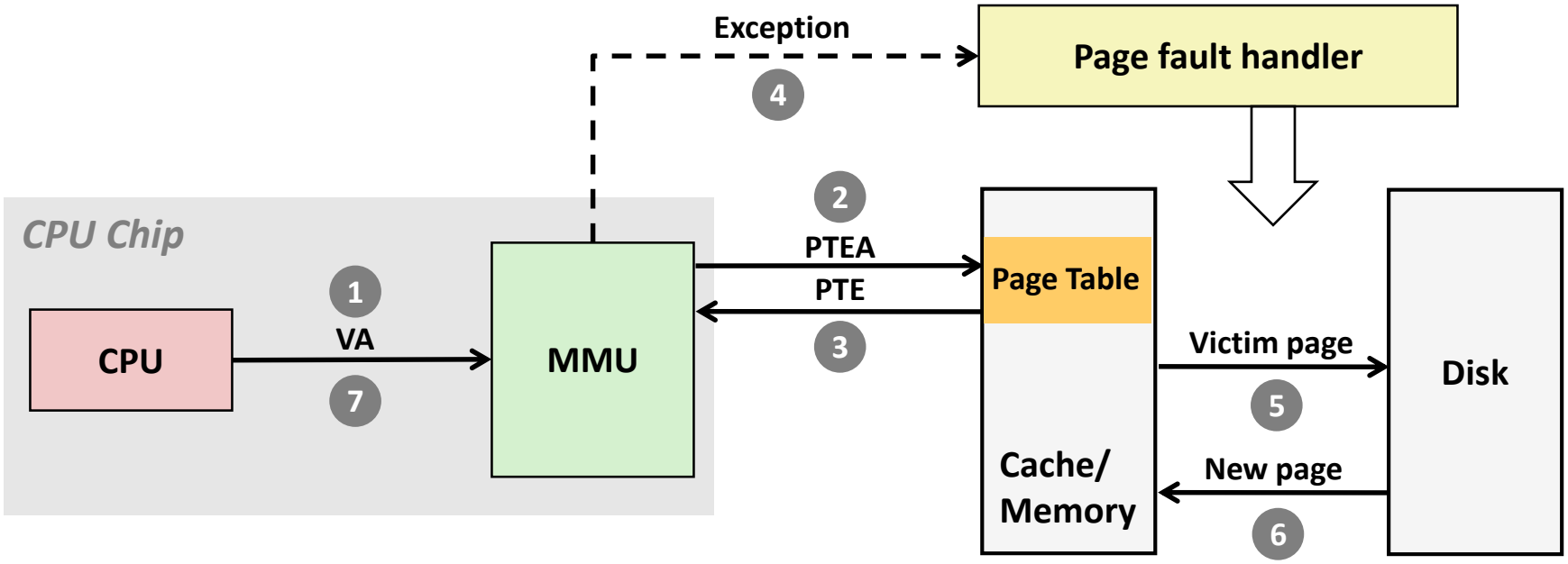
1) Processor sends virtual address to MMU

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)

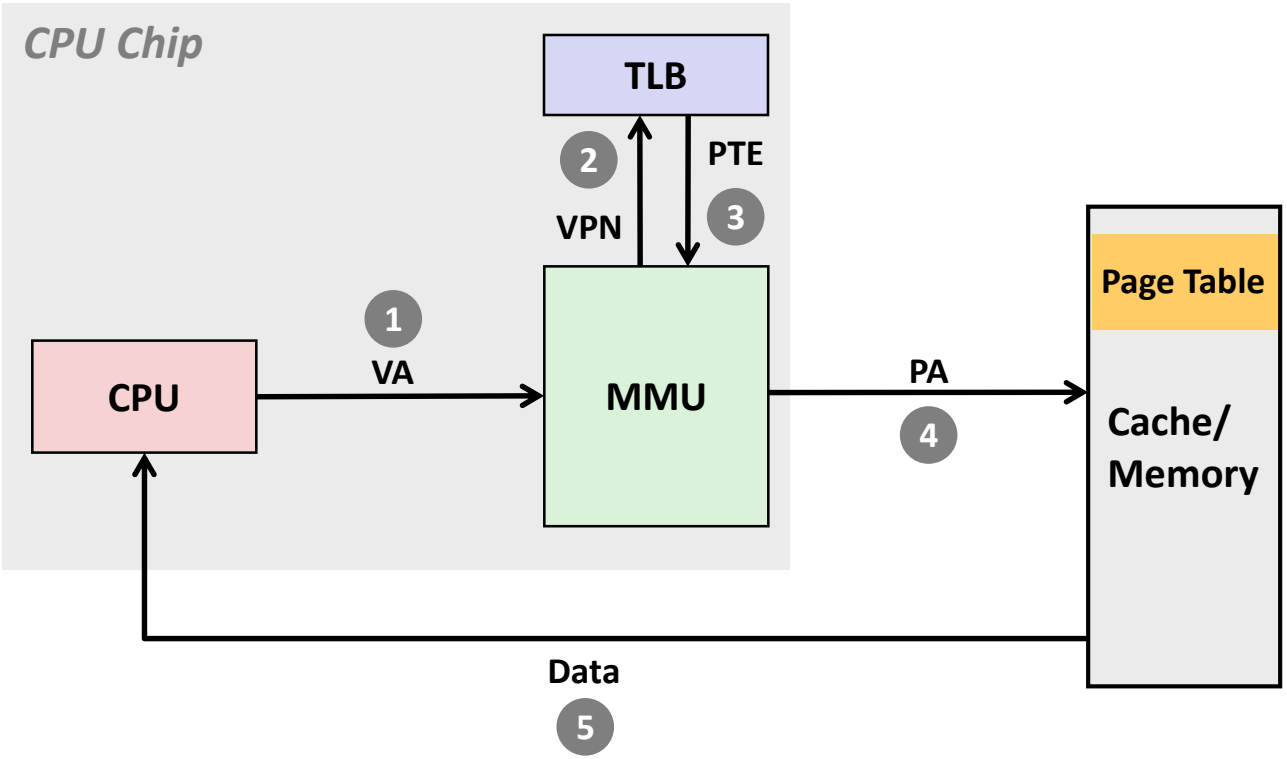
6) Handler pages in new page and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction

Speeding up Translation with a TLB

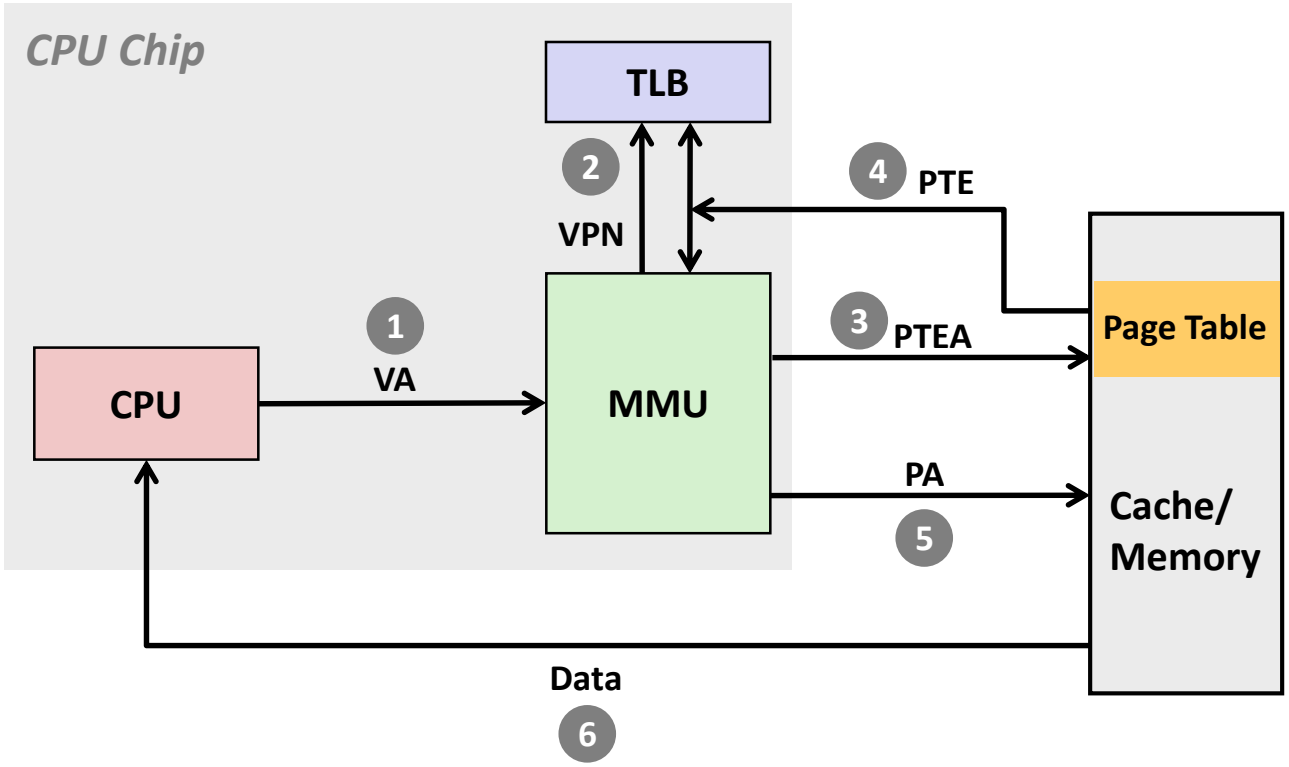
- **Address translation can be slow**
  - Page table entries (PTEs) should be stored in memory
  - Every memory access through VA would results in two memory accesses
    - #1. To translate VA to PA, accessing PTE
    - #2. Fetch the data using PA
  - Two memory accesses? Can we make it faster?
- **Idea: *Translation Lookaside Buffer* (TLB)**
  - Small set-associative hardware cache in MMU
  - Cache a subset of page table entries
    - virtual page numbers to physical page numbers

TLB Hit



A TLB hit eliminates a slow memory/cache access

TLB Miss

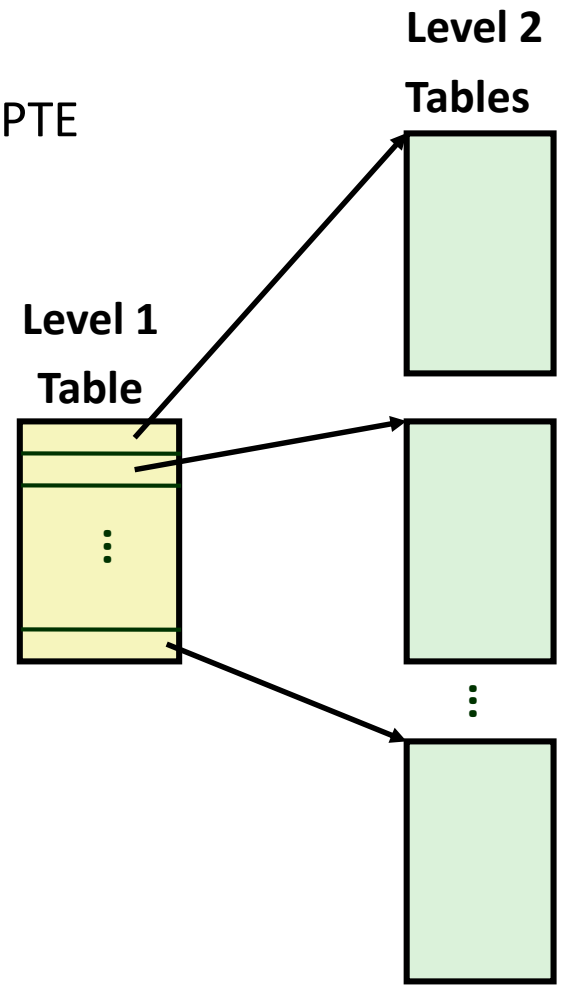


A TLB miss incurs an additional cache/memory access (the PTE)

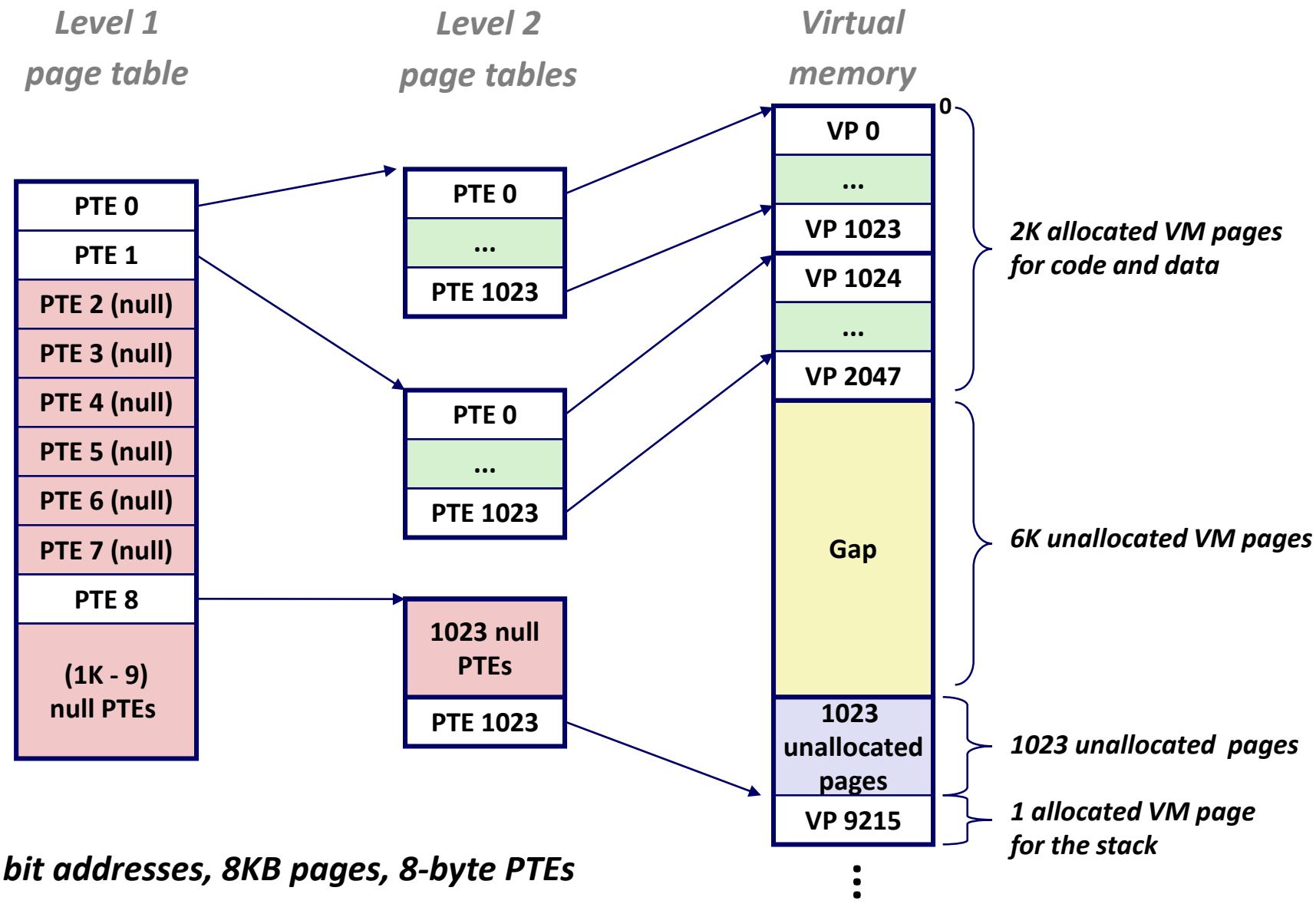
Fortunately, TLB misses are rare. Why?

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**
  - L1: each L1 PTE points to a L2 pagetable (always memory resident)
  - L2: each L2 PTE points to a page (paged in and out like any other data)



A Two-Level Page Table Hierarchy



64 bit addresses, 8KB pages, 8-byte PTEs

Translating with a k-level Page Table

Components of the virtual address (VA)

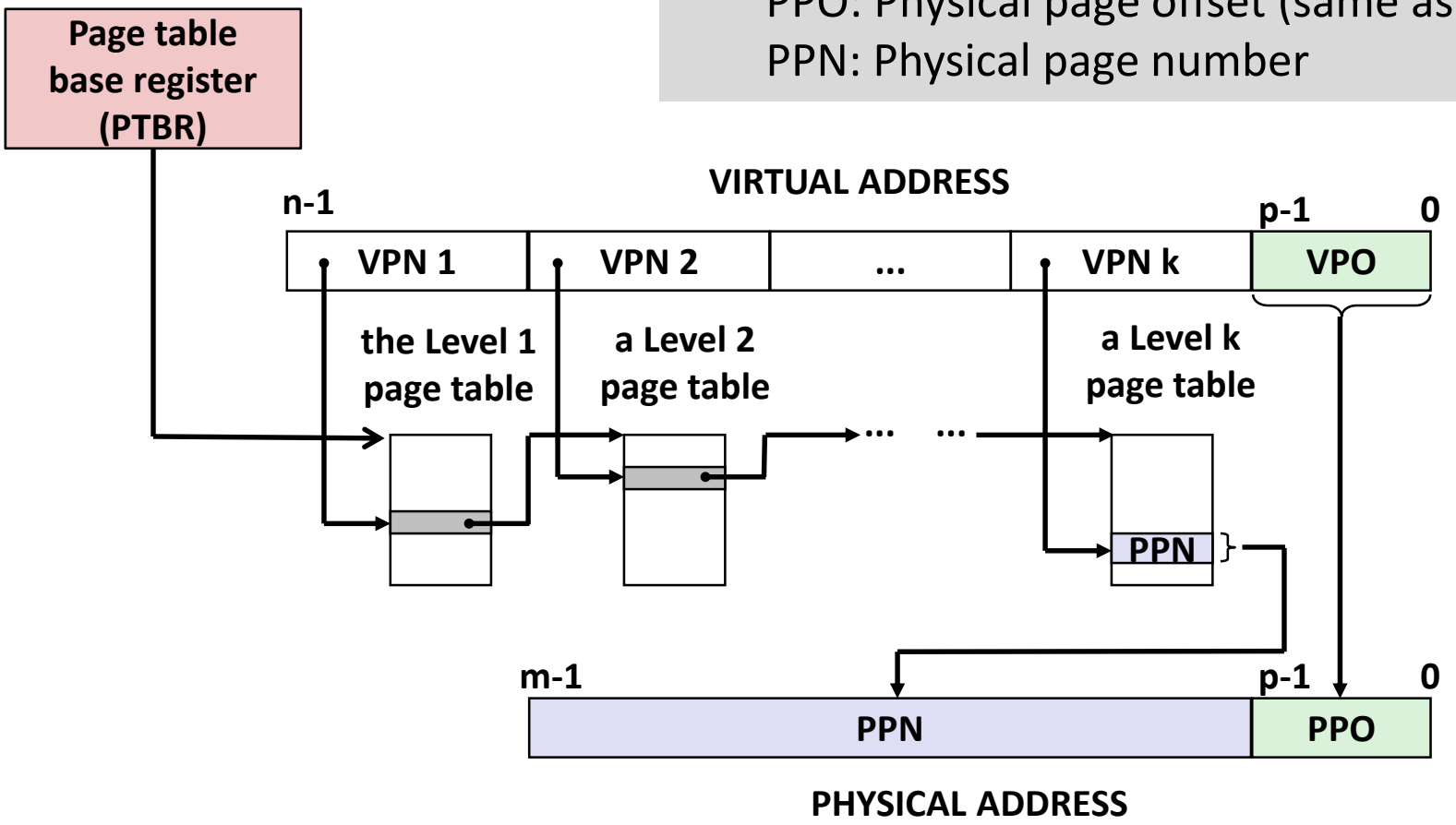
VPO: Virtual page offset

VPN: Virtual page number

Components of the physical address (PA)

PPO: Physical page offset (same as VPO)

PPN: Physical page number



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 because of locality
  - Simplifies memory management and programming
  - Simplifies protection by providing a convenient interpositioning point to check permissions
- **Implemented via combination of hardware & software**
  - MMU, TLB, exception handling mechanisms part of hardware
  - Page fault handlers, TLB management performed in software