CSC 252: Computer Organization
Spring 2018: Lecture 22

Instructor: Yuhao Zhu

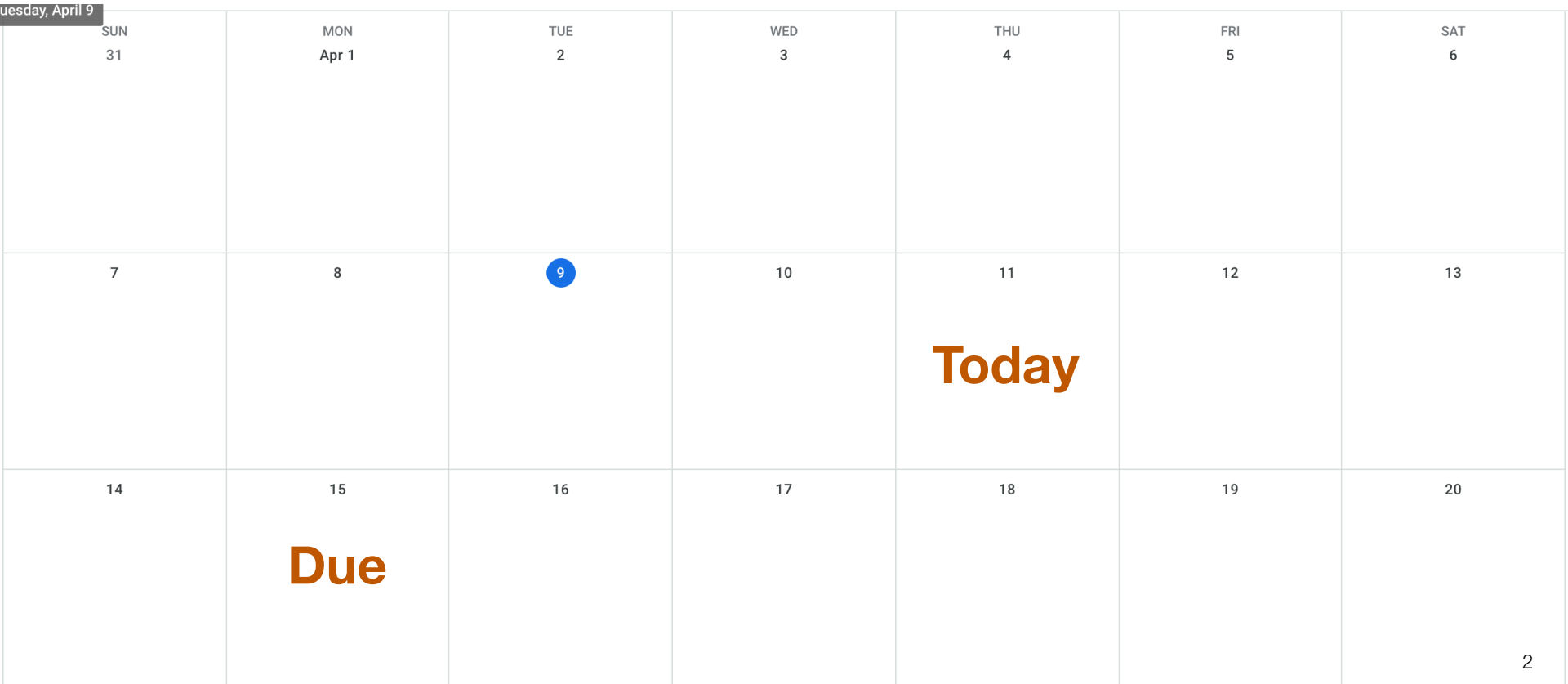
Department of Computer Science
University of Rochester

Action Items:

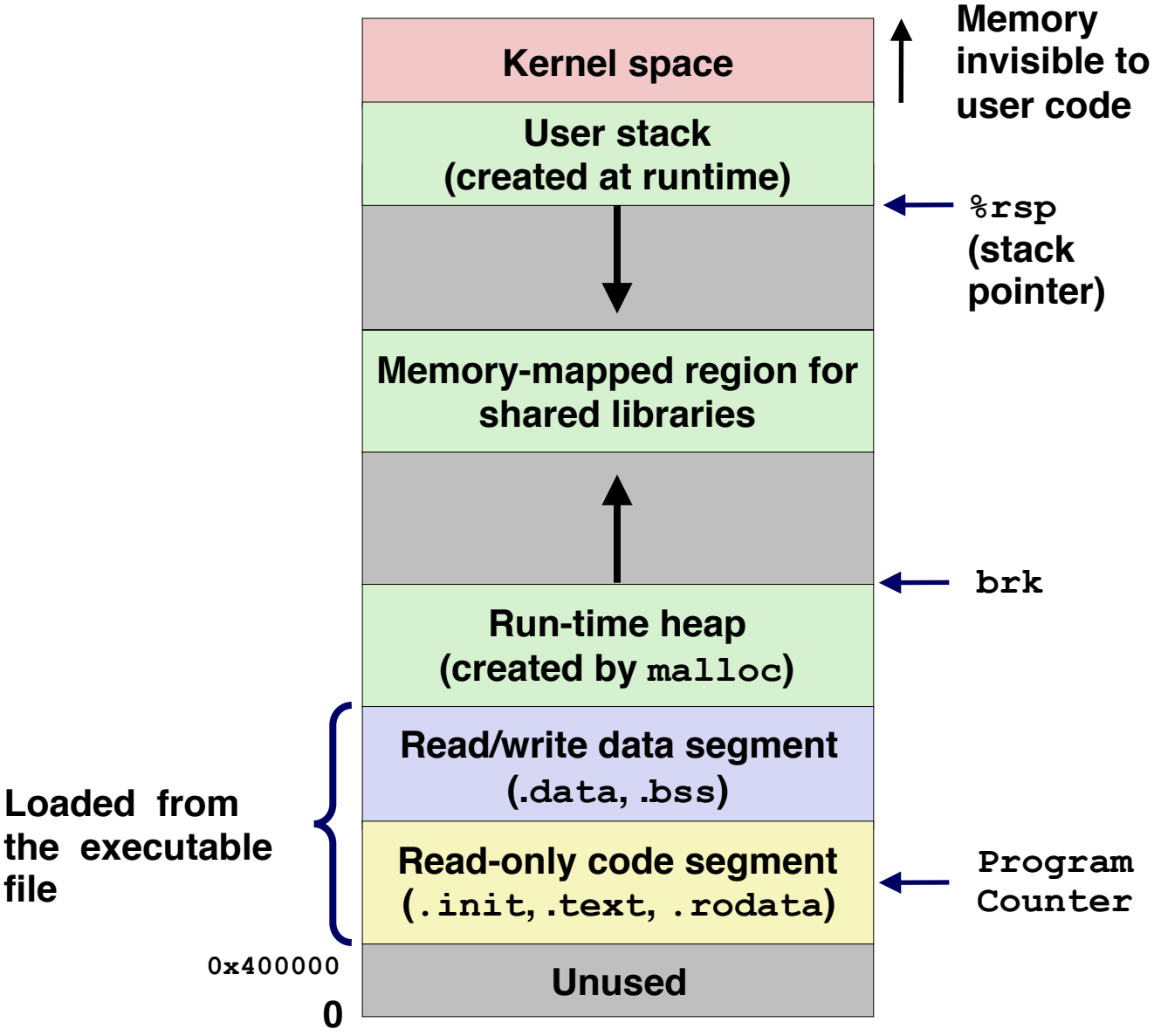
- **Programming Assignment 5 is out**

Announcement

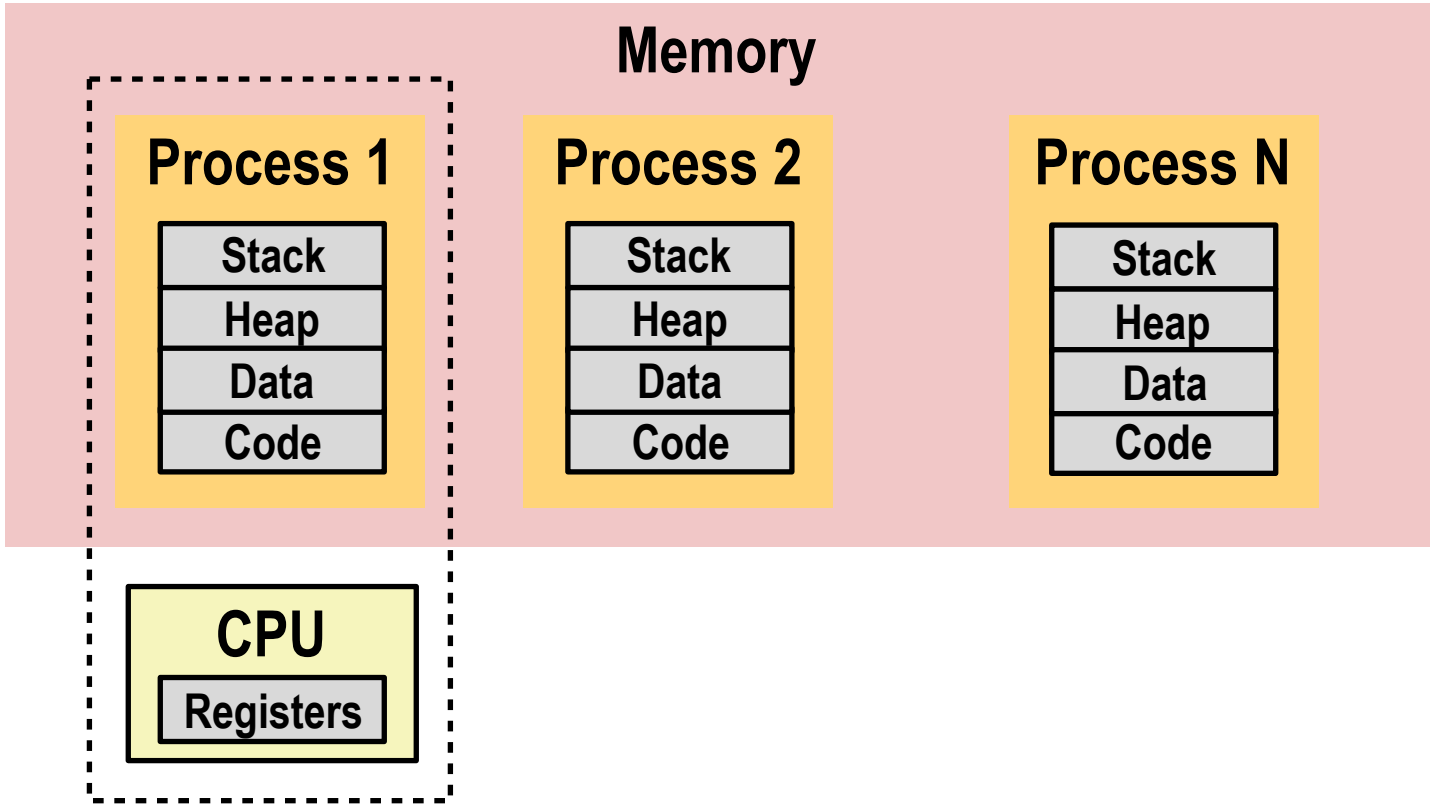
- Programming Assignment 5 is due 11:59pm, Monday, April 15.
- Your shell should emit output that is identical to the reference solution (except the PID). If you output does not match the reference solution, you get a 0. **NO exceptions will be made.**



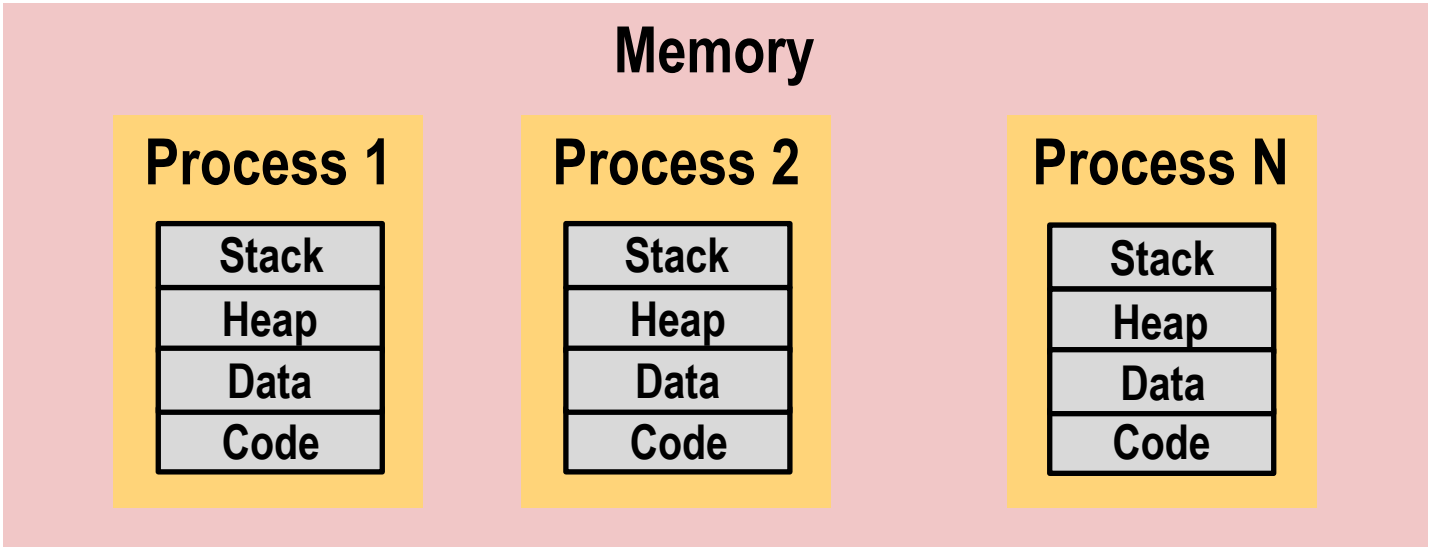
Process Address Space



Multiprocessing Illustration

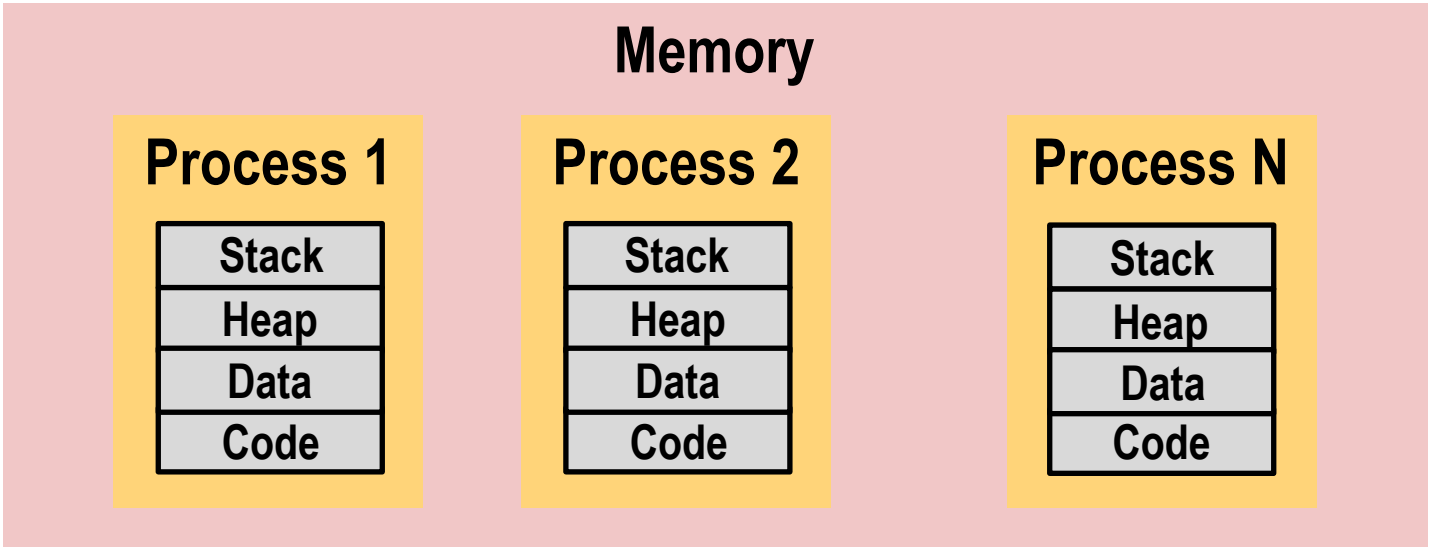


Problem



- Space:
  - Each process's address space is huge (64-bit): can memory hold it (16GB is just 34-bit)?
  - There are multiple processes, increasing the overhead further

Problem



- Space:
  - Each process's address space is huge (64-bit): can memory hold it (16GB is just 34-bit)?
  - There are multiple processes, increasing the overhead further
- Solution: store all the data in disk, and use memory only for most recently used data
  - Does this sound similar?

The Big Idea: Virtual Memory

The Big Idea: Virtual Memory

- What Does a Programmer Want?

The Big Idea: Virtual Memory

- What Does a Programmer Want?
- Infinitely large, infinitely fast memory
  - Preferably automatically moved to where it is needed

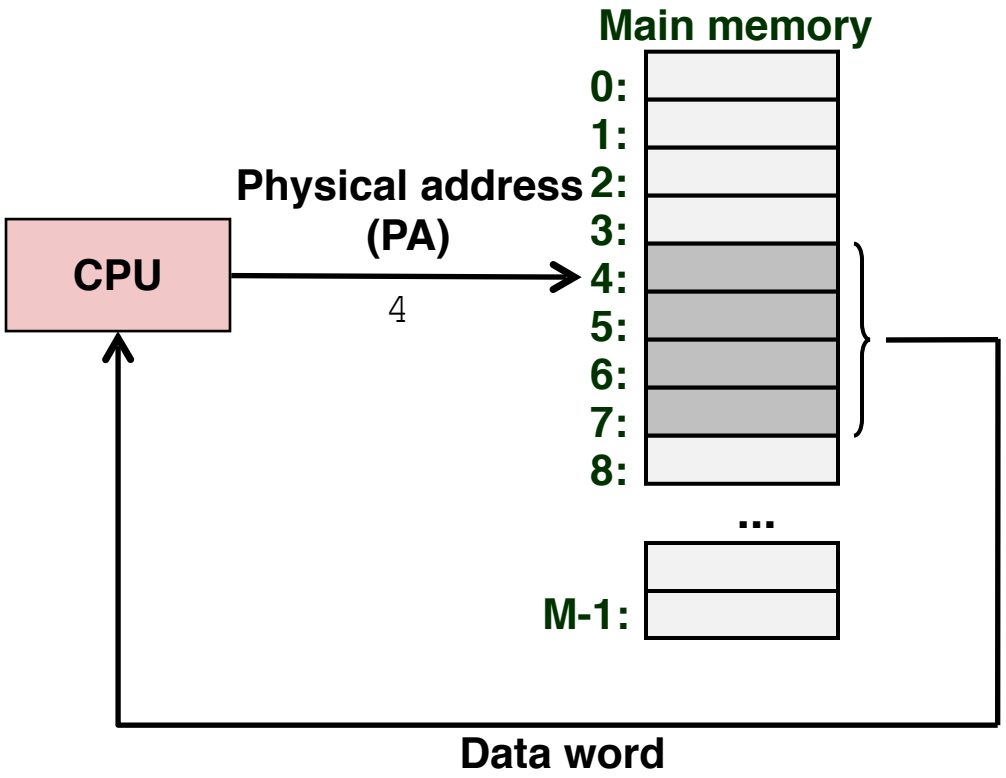
The Big Idea: Virtual Memory

- What Does a Programmer Want?
- Infinitely large, infinitely fast memory
  - Preferably automatically moved to where it is needed
- Virtual memory to the rescue
  - Present a large, uniform memory to programmers
  - Data in virtual memory by default stays in disk
  - Data moves to physical memory (DRAM) "**on demand**"
  - Disks (~TBs) are much larger than DRAM (~GBs), but 10,000x slower.
  - Effectively, virtual memory system transparently share the physical memory across different processes
  - Manage the sharing automatically: hardware-software collaborative strategy (too complex for hardware alone)

Today

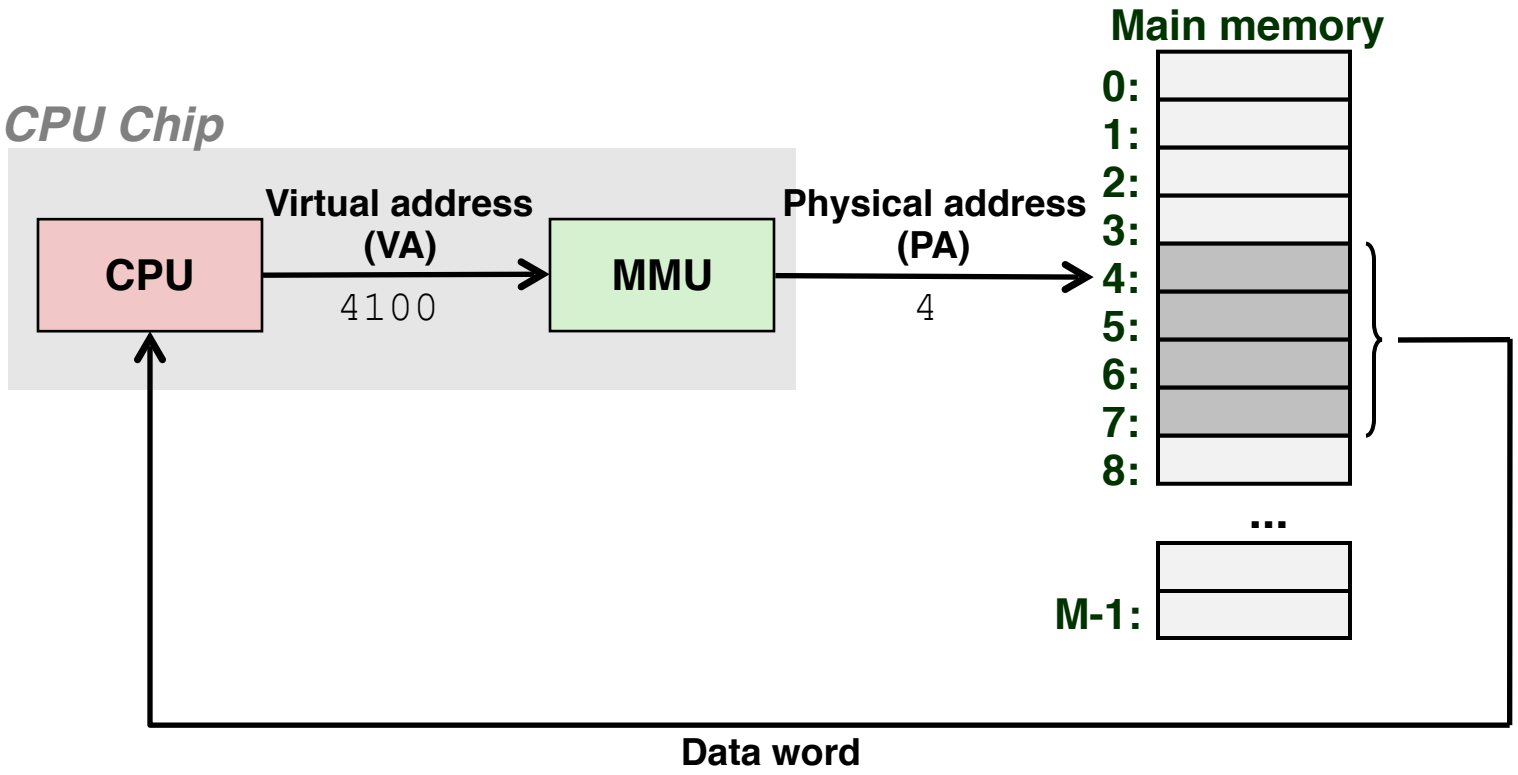
- Virtual memory (VM) illustration
- VM basic concepts and operation
- Other critical benefits of VM
- Address translation

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 servers, laptops, and smart phones
- One of the great ideas in computer science
- MMU: Memory Management Unit

Address Spaces

Address Spaces

- Virtual address space: Set of N = 2^n virtual addresses
  - Virtual address space is a linear address space, but limited

    {0, 1, 2, 3, …, N-1}

Address Spaces

- Virtual address space: Set of $N = 2^n$ virtual addresses
  - Virtual address space is a linear address space, but limited

    {0, 1, 2, 3, …, N-1}

- Physical address space: Set of $M = 2^m$ physical addresses
  - Physical address space is a also linear address space, but smaller than virtual address space

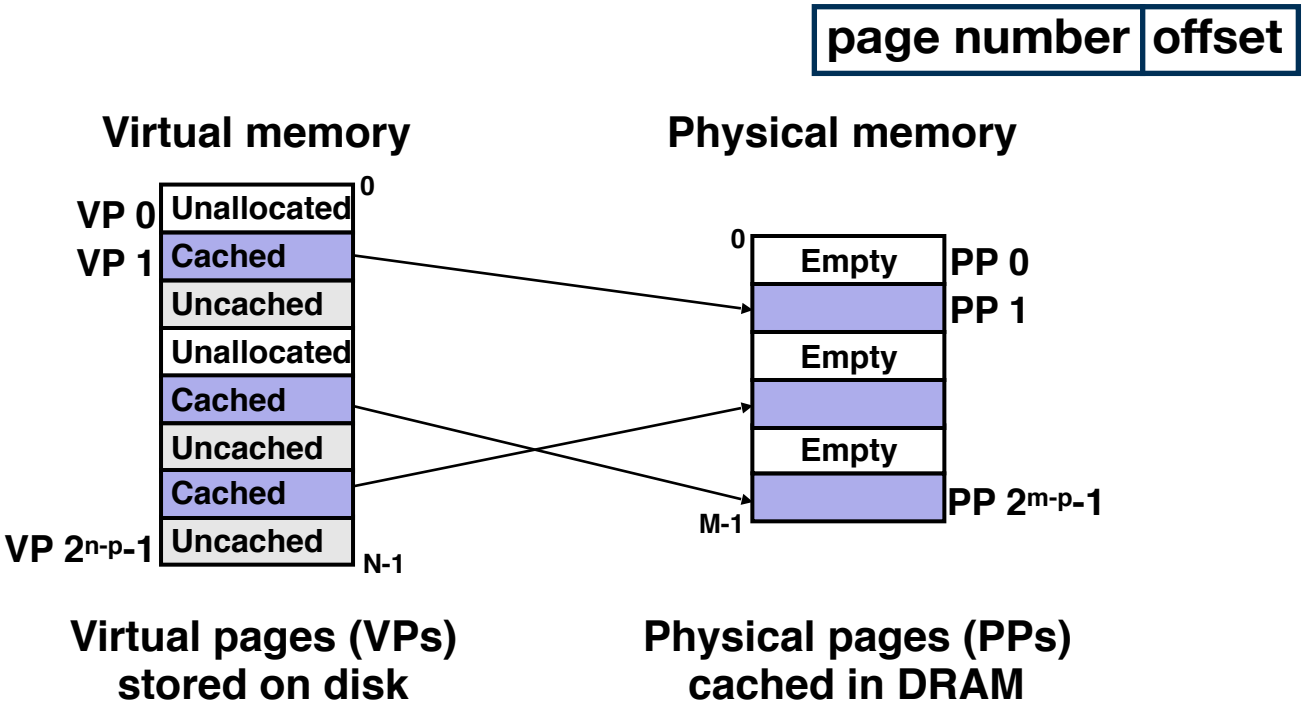
    {0, 1, 2, 3, …, M-1}

Today

- Virtual memory (VM) illustration
- VM basic concepts and operation
- Other critical benefits of VM
- Address translation

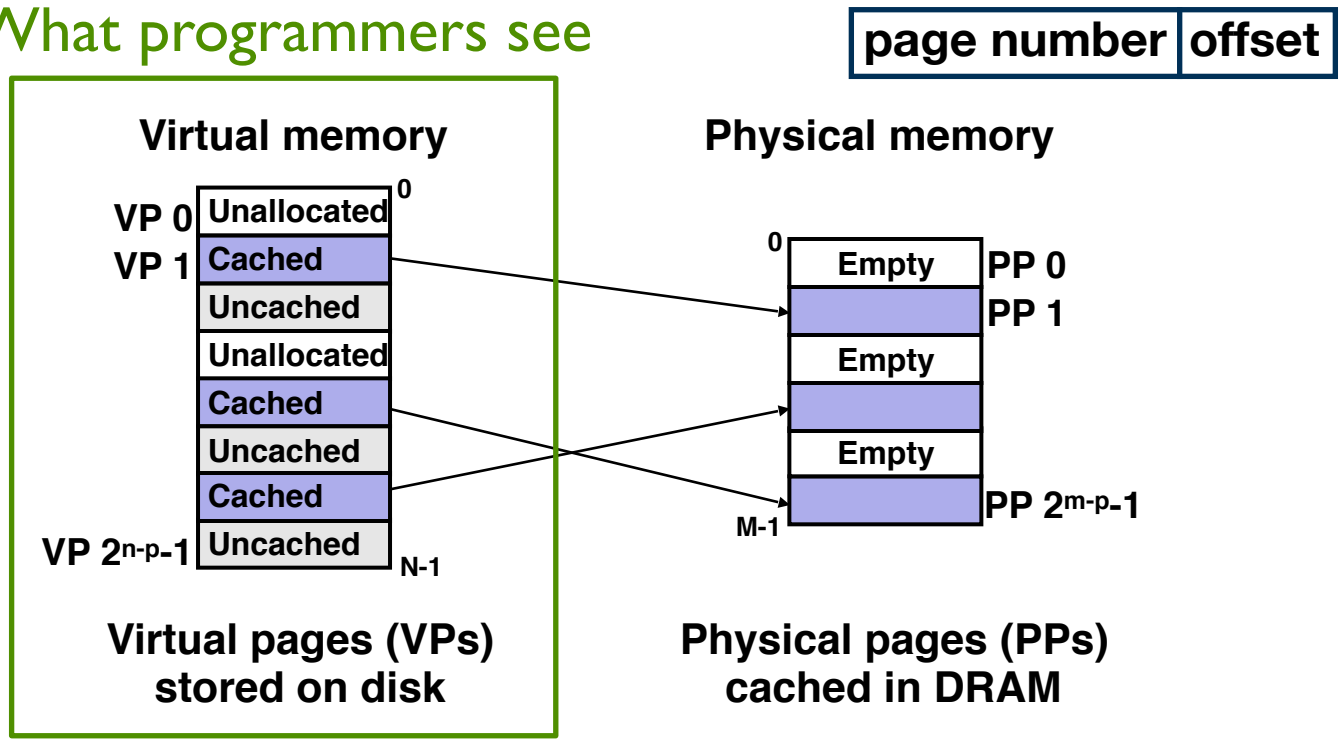
VM Concepts

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



VM Concepts

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



Analogy for Address Translation: A Secure Hotel

Analogy for Address Translation: A Secure Hotel

- Call a hotel looking for a guest; what happens?
  - Front desk routes call to room, does not give out room number
  - Guest's name is a virtual address
  - Room number is physical address
  - Front desk is doing address translation!

Analogy for Address Translation: A Secure Hotel

- Call a hotel looking for a guest; what happens?
  - Front desk routes call to room, does not give out room number
  - Guest's name is a virtual address
  - Room number is physical address
  - Front desk is doing address translation!

- Benefits
  - **Ease of management**: Guest could change rooms (physical address). You can still find her without knowing it
  - **Protection**: Guest could have block on calls, block on calls from specific callers (permissions)
  - **Sharing**: Multiple guests (virtual addresses) can share the same room (physical address)

Different Names in Different Places

Symbolic
address

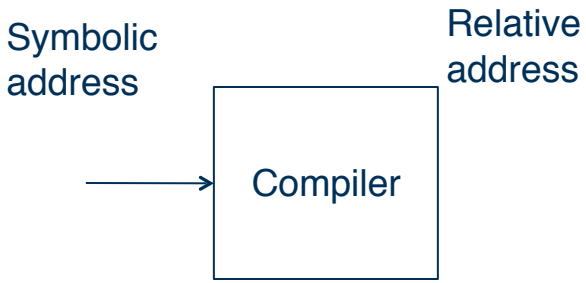
- Programmer uses text-based names (symbolic address)
  - int array[100];

Different Names in Different Places

Symbolic
address

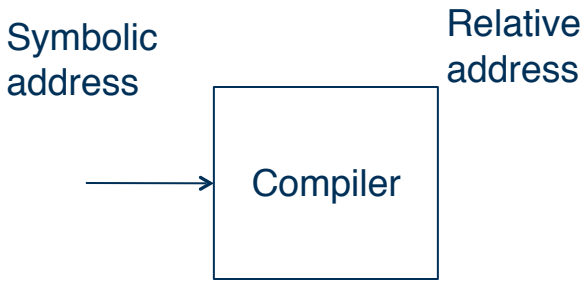
- Programmer uses text-based names (symbolic address)
  - int array[100];
- Compiler maps names to flat, uniform space
  - Starting point is relative, size specified

Different Names in Different Places



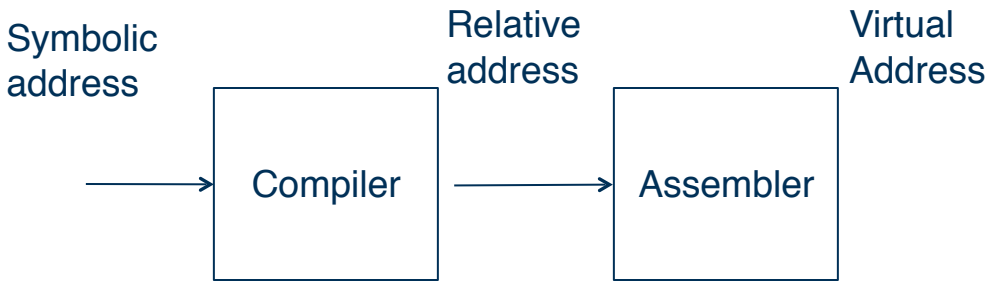
- Programmer uses text-based names (symbolic address)
  - int array[100];
- Compiler maps names to flat, uniform space
  - Starting point is relative, size specified

Different Names in Different Places



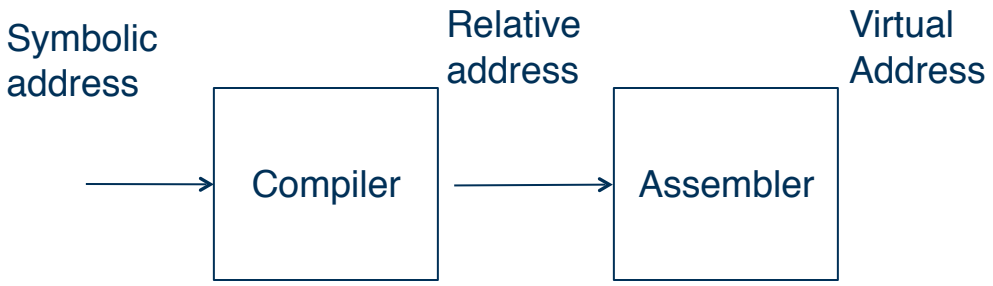
- Programmer uses text-based names (symbolic address)
  - int array[100];
- Compiler maps names to flat, uniform space
  - Starting point is relative, size specified
- Assembler maps uniform space to virtual addresses
  - Mechanical transformation (assume a start address)

Different Names in Different Places



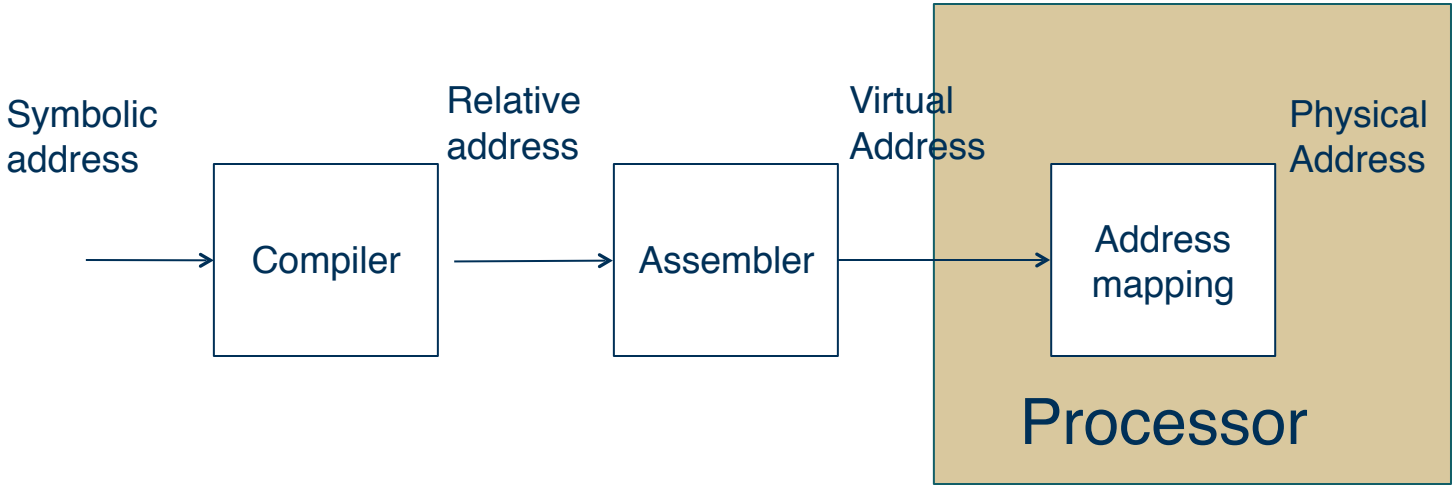
- Programmer uses text-based names (symbolic address)
  - int array[100];
- Compiler maps names to flat, uniform space
  - Starting point is relative, size specified
- Assembler maps uniform space to virtual addresses
  - Mechanical transformation (assume a start address)

Different Names in Different Places



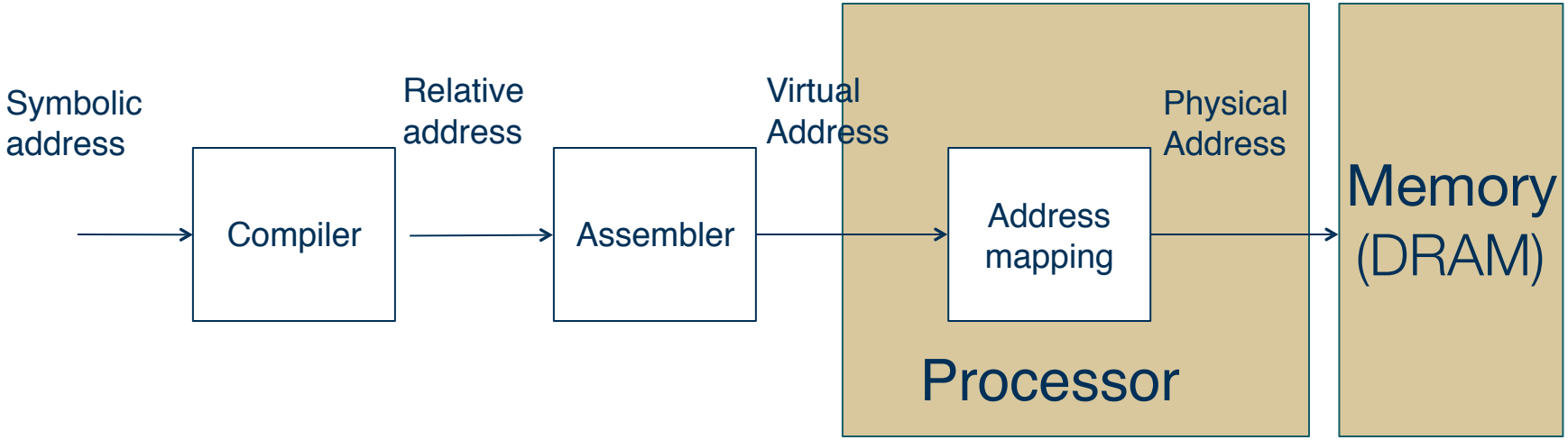
- Programmer uses text-based names (symbolic address)
  - int array[100];
- Compiler maps names to flat, uniform space
  - Starting point is relative, size specified
- Assembler maps uniform space to virtual addresses
  - Mechanical transformation (assume a start address)
- Processor instructions use virtual addresses, translates to physical addresses

Different Names in Different Places



- Programmer uses text-based names (symbolic address)
  - int array[100];
- Compiler maps names to flat, uniform space
  - Starting point is relative, size specified
- Assembler maps uniform space to virtual addresses
  - Mechanical transformation (assume a start address)
- Processor instructions use virtual addresses, translates to physical addresses

Different Names in Different Places



- Programmer uses text-based names (symbolic address)
  - int array[100];
- Compiler maps names to flat, uniform space
  - Starting point is relative, size specified
- Assembler maps uniform space to virtual addresses
  - Mechanical transformation (assume a start address)
- Processor instructions use virtual addresses, translates to physical addresses

Enabling Data Structure: Page Table

- How do we track which virtual pages are mapped to physical pages, and where they are mapped?

Enabling Data Structure: Page Table

- How do we track which virtual pages are mapped to physical pages, and where they are mapped?
- Use a table to track this. The table is called page table, in which each virtual page has an entry

Enabling Data Structure: Page Table

- How do we track which virtual pages are mapped to physical pages, and where they are mapped?
- Use a table to track this. The table is called page table, in which each virtual page has an entry
- Each entry records whether the particular virtual page is mapped to the physical memory

Enabling Data Structure: Page Table

- How do we track which virtual pages are mapped to physical pages, and where they are mapped?
- Use a table to track this. The table is called page table, in which each virtual page has an entry
- Each entry records whether the particular virtual page is mapped to the physical memory
  - If mapped, where in the physical memory it is mapped to?

Enabling Data Structure: Page Table

- How do we track which virtual pages are mapped to physical pages, and where they are mapped?
- Use a table to track this. The table is called page table, in which each virtual page has an entry
- Each entry records whether the particular virtual page is mapped to the physical memory
  - If mapped, where in the physical memory it is mapped to?
  - If not mapped, where on the disk is the virtual page?

Enabling Data Structure: Page Table

- How do we track which virtual pages are mapped to physical pages, and where they are mapped?
- Use a table to track this. The table is called page table, in which each virtual page has an entry
- Each entry records whether the particular virtual page is mapped to the physical memory
  - If mapped, where in the physical memory it is mapped to?
  - If not mapped, where on the disk is the virtual page?
- Do you need a page table for each process?

Enabling Data Structure: Page Table

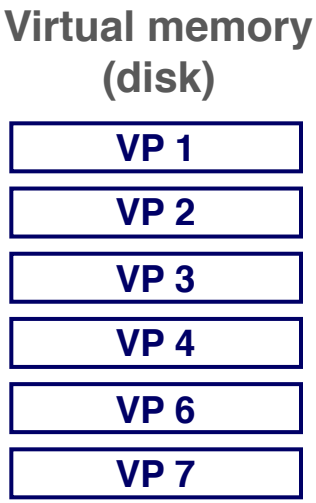
- How do we track which virtual pages are mapped to physical pages, and where they are mapped?
- Use a table to track this. The table is called page table, in which each virtual page has an entry
- Each entry records whether the particular virtual page is mapped to the physical memory
  - If mapped, where in the physical memory it is mapped to?
  - If not mapped, where on the disk is the virtual page?
- Do you need a page table for each process?
  - Per-process data structure; managed by the OS kernel

Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.

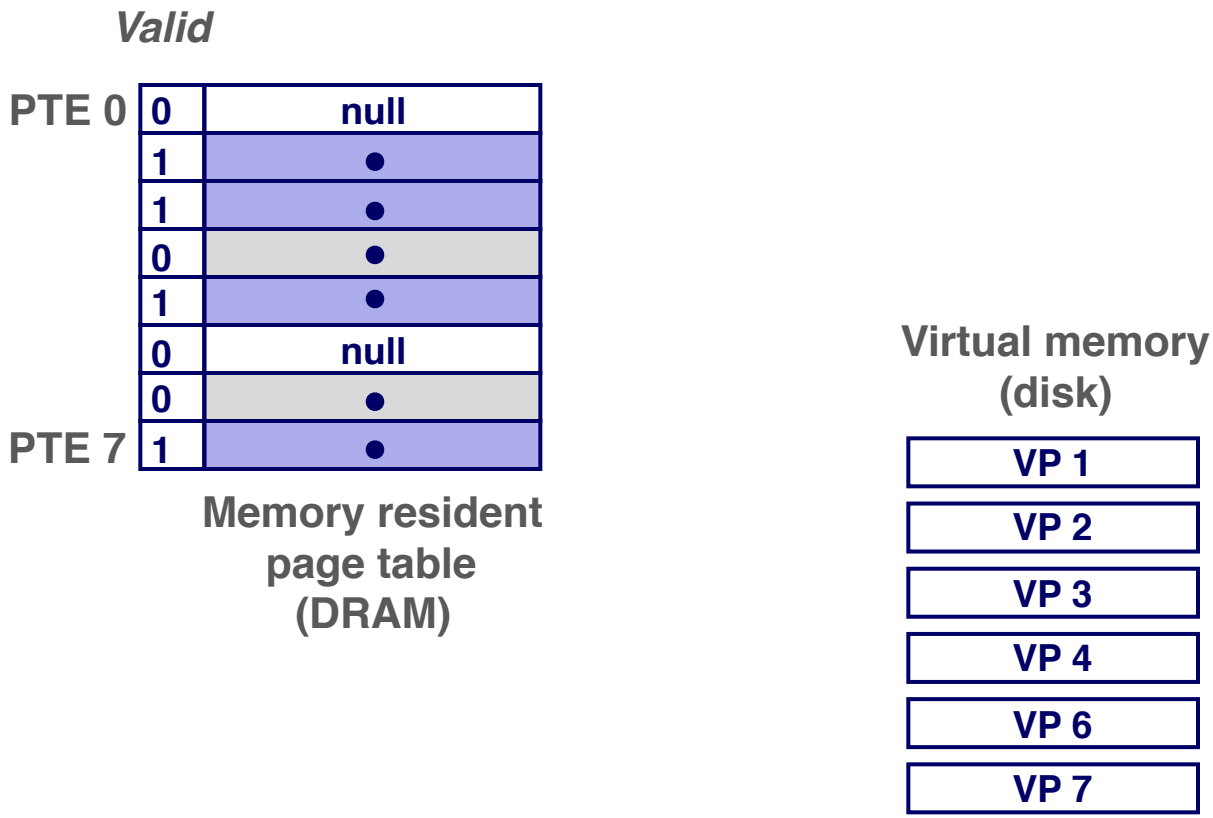
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



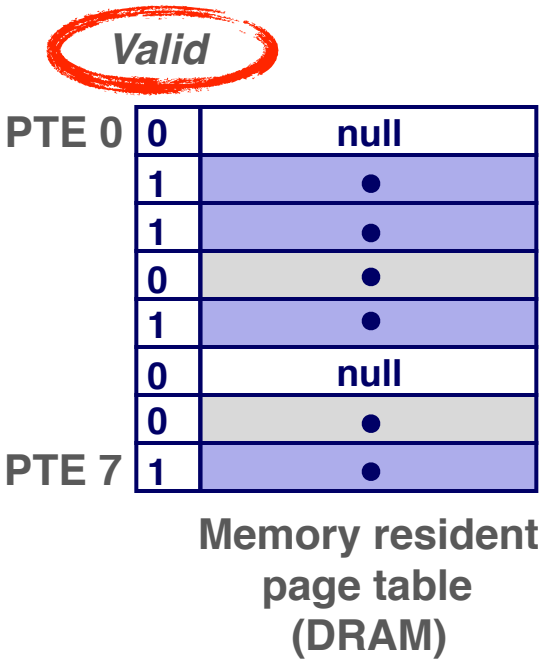
Enabling Data Structure: Page Table

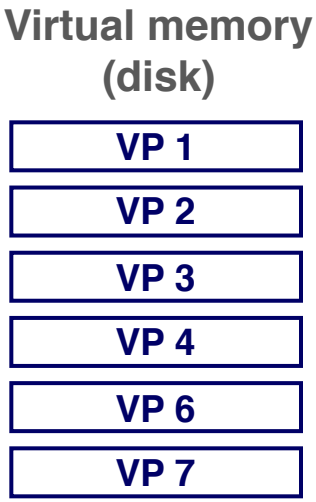
- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



Enabling Data Structure: Page Table

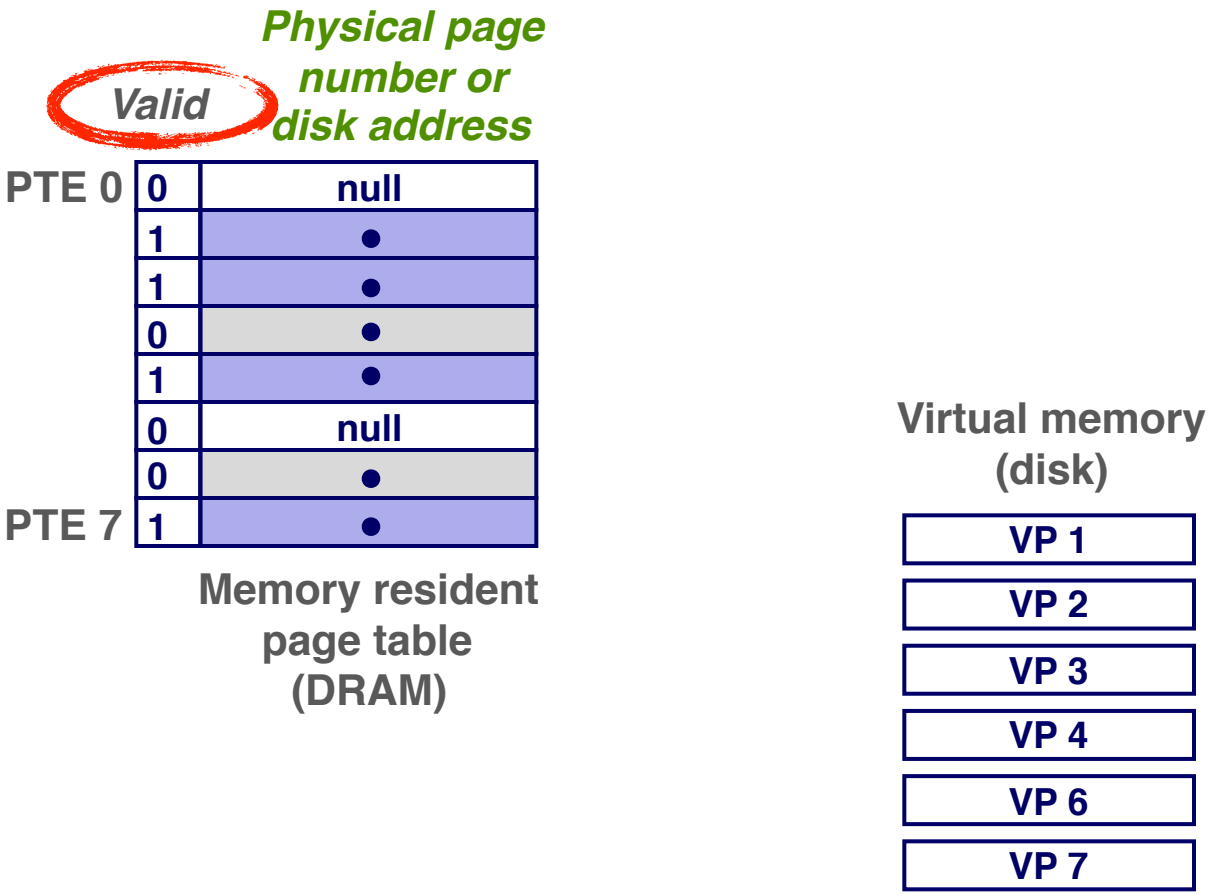
- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.





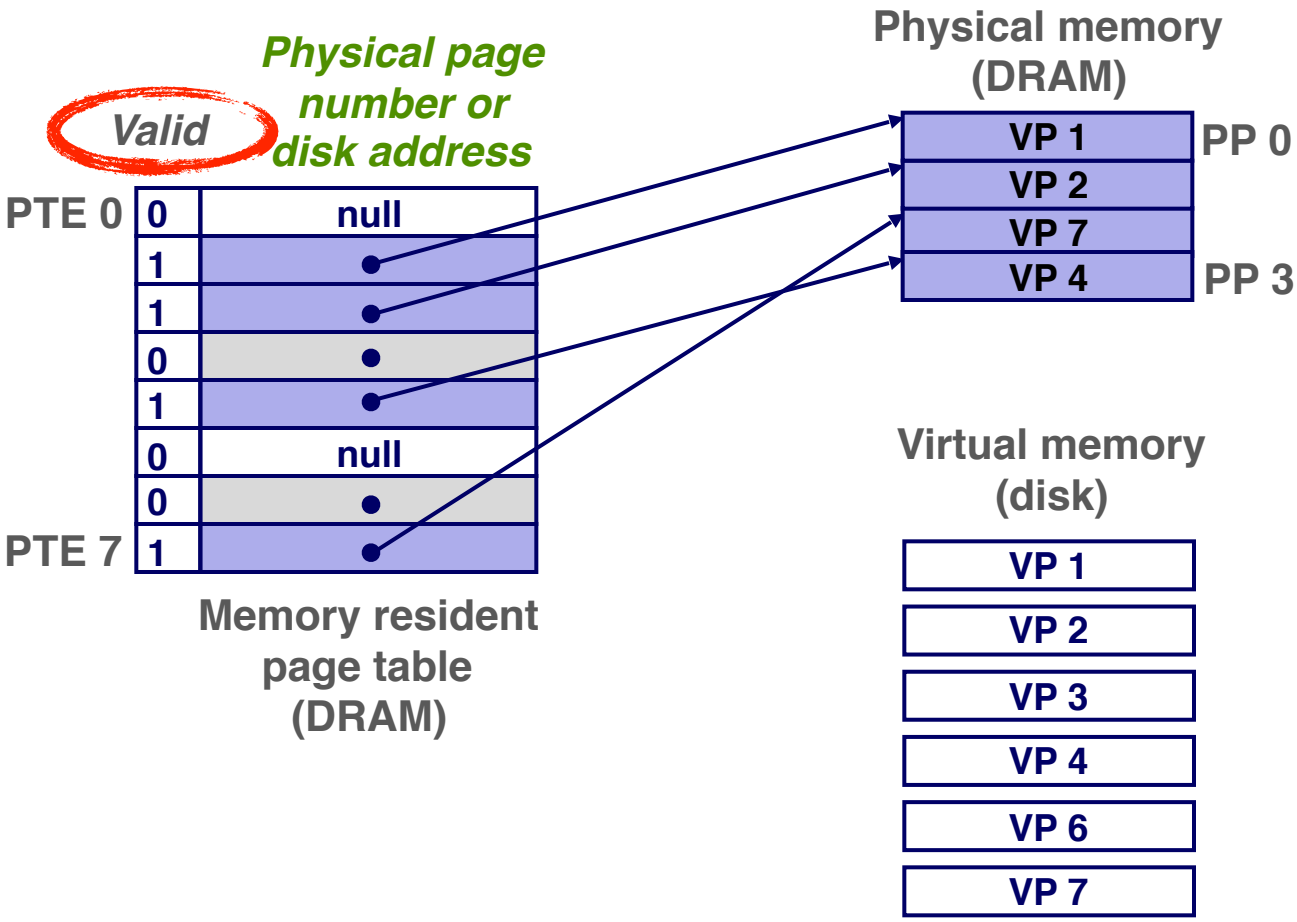
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



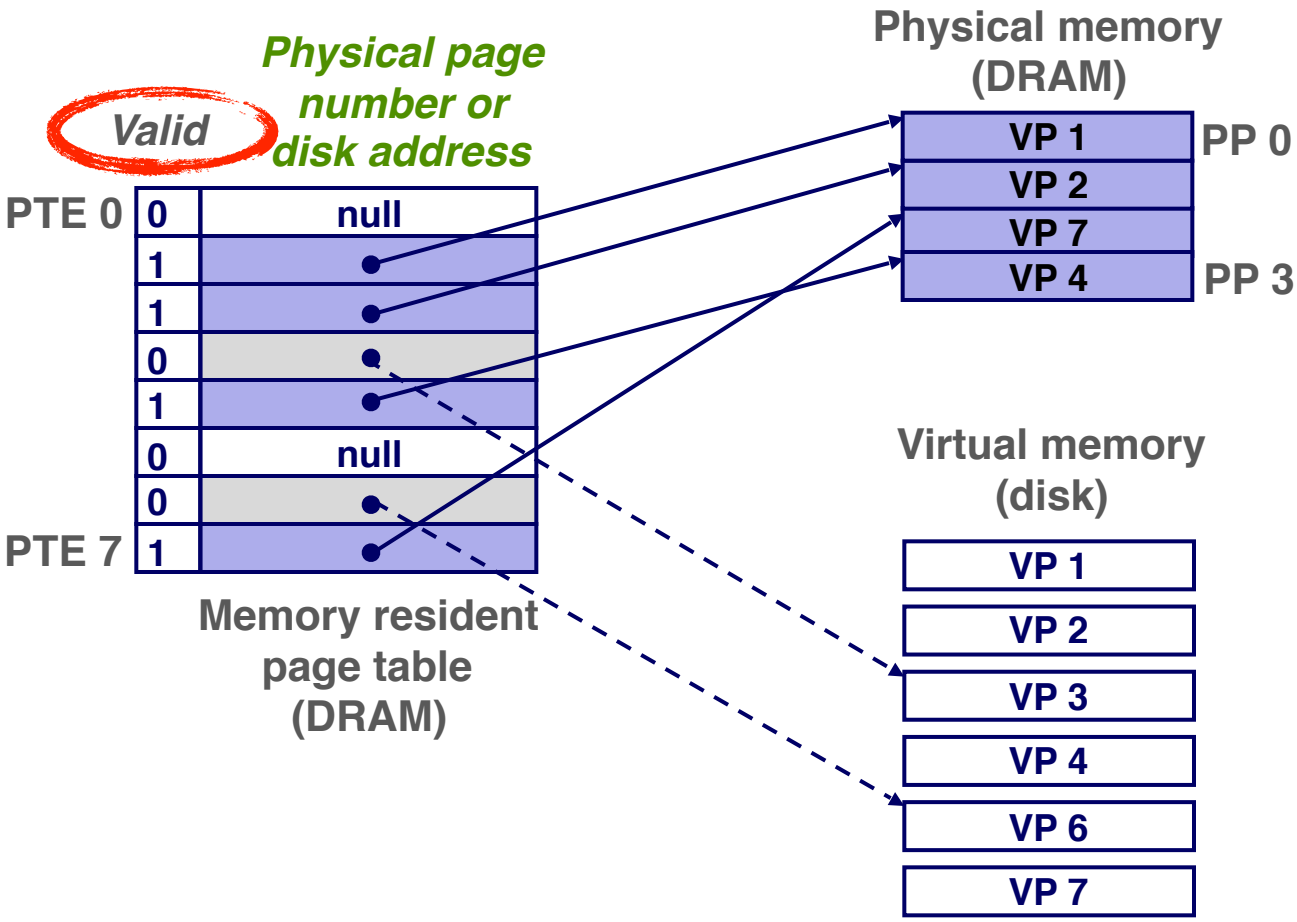
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



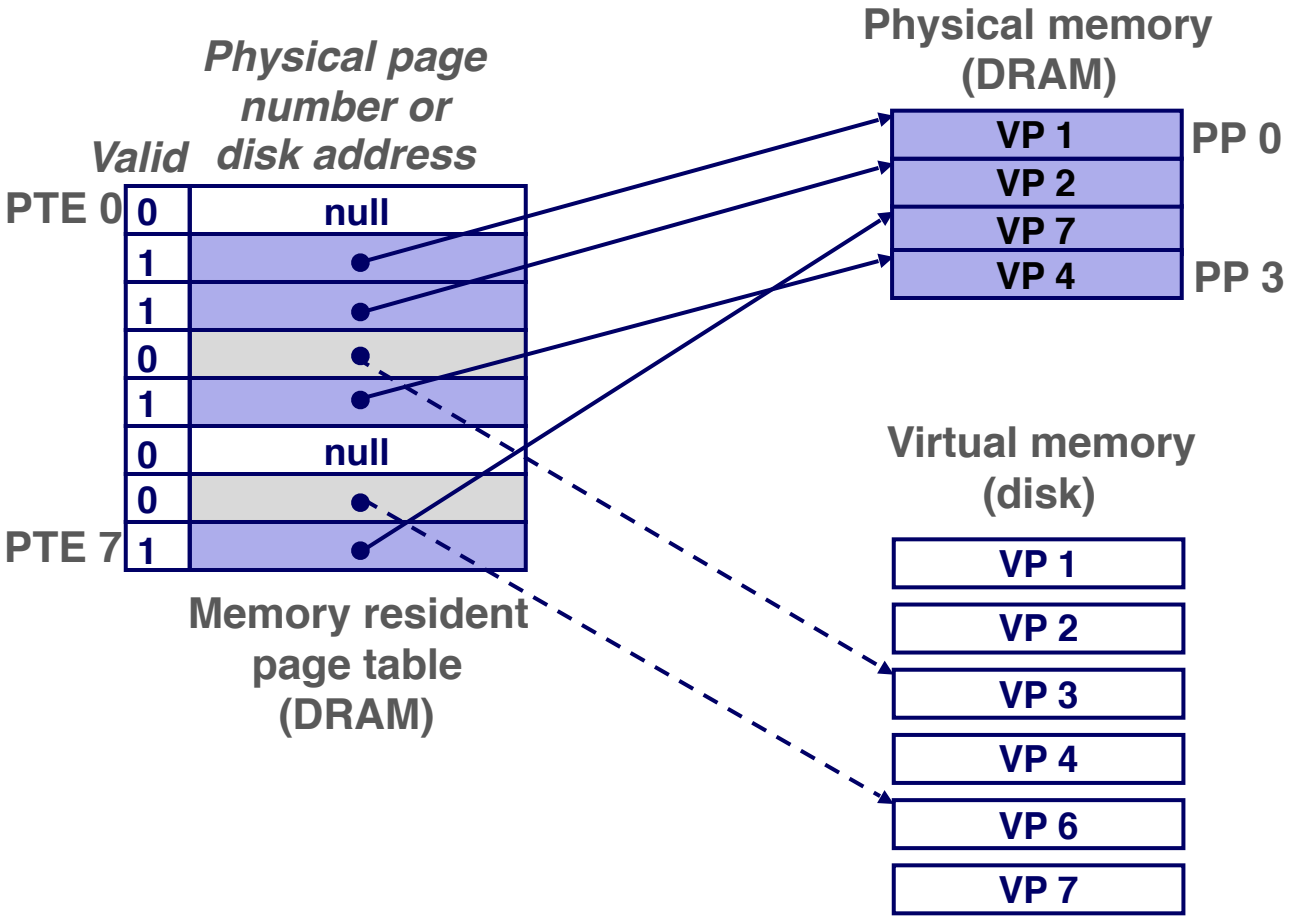
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page.



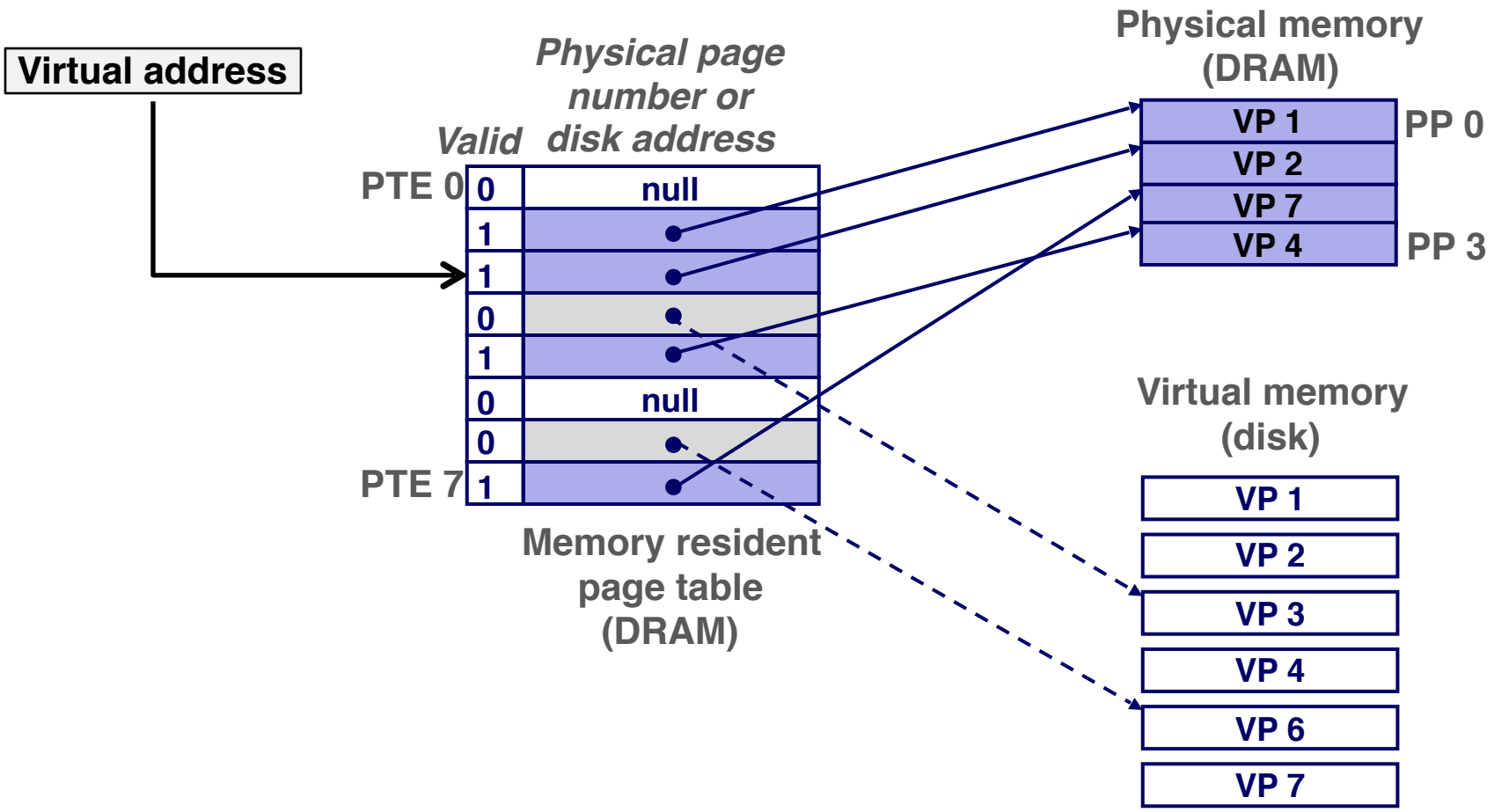
Page Hit

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



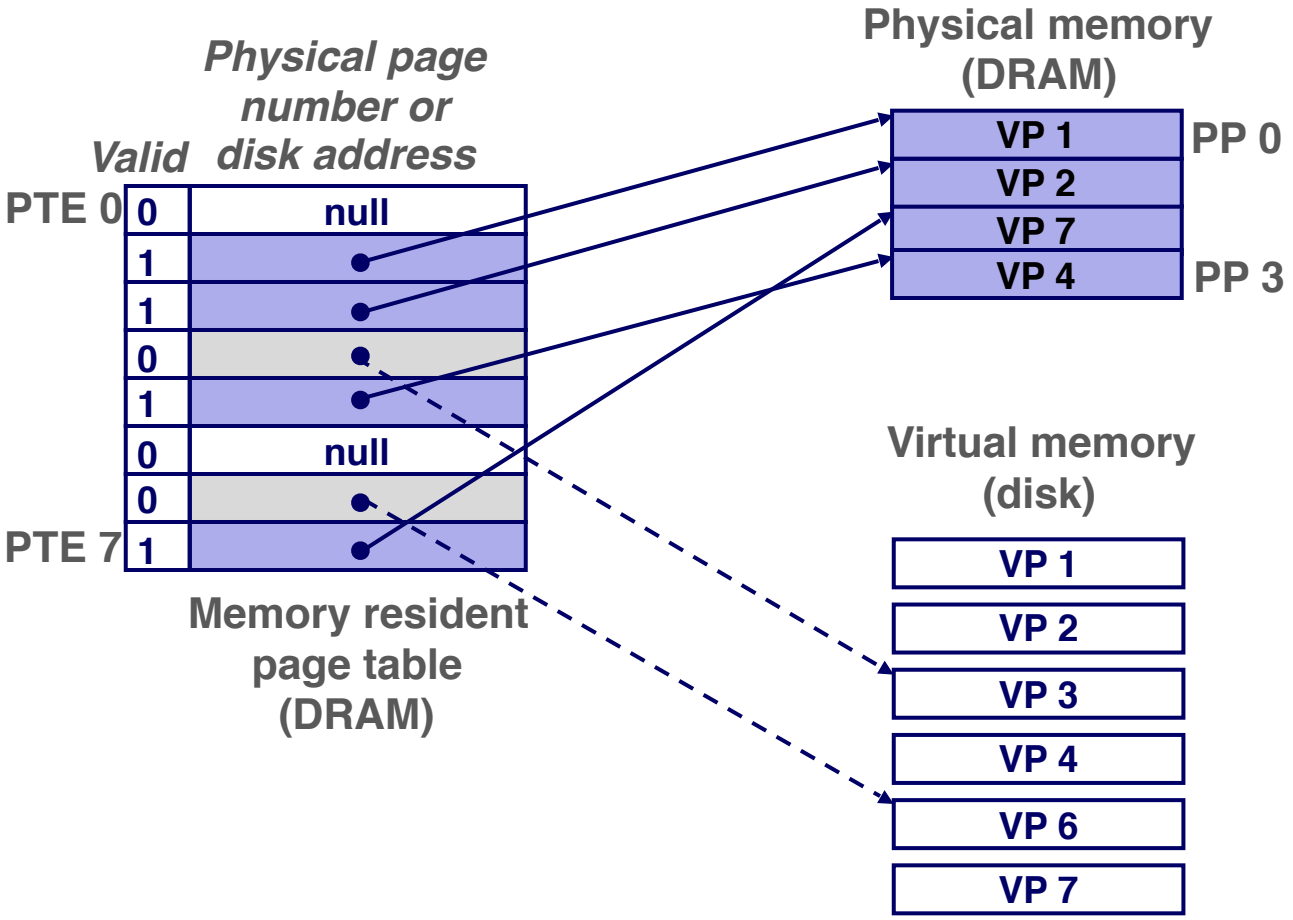
Page Hit

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



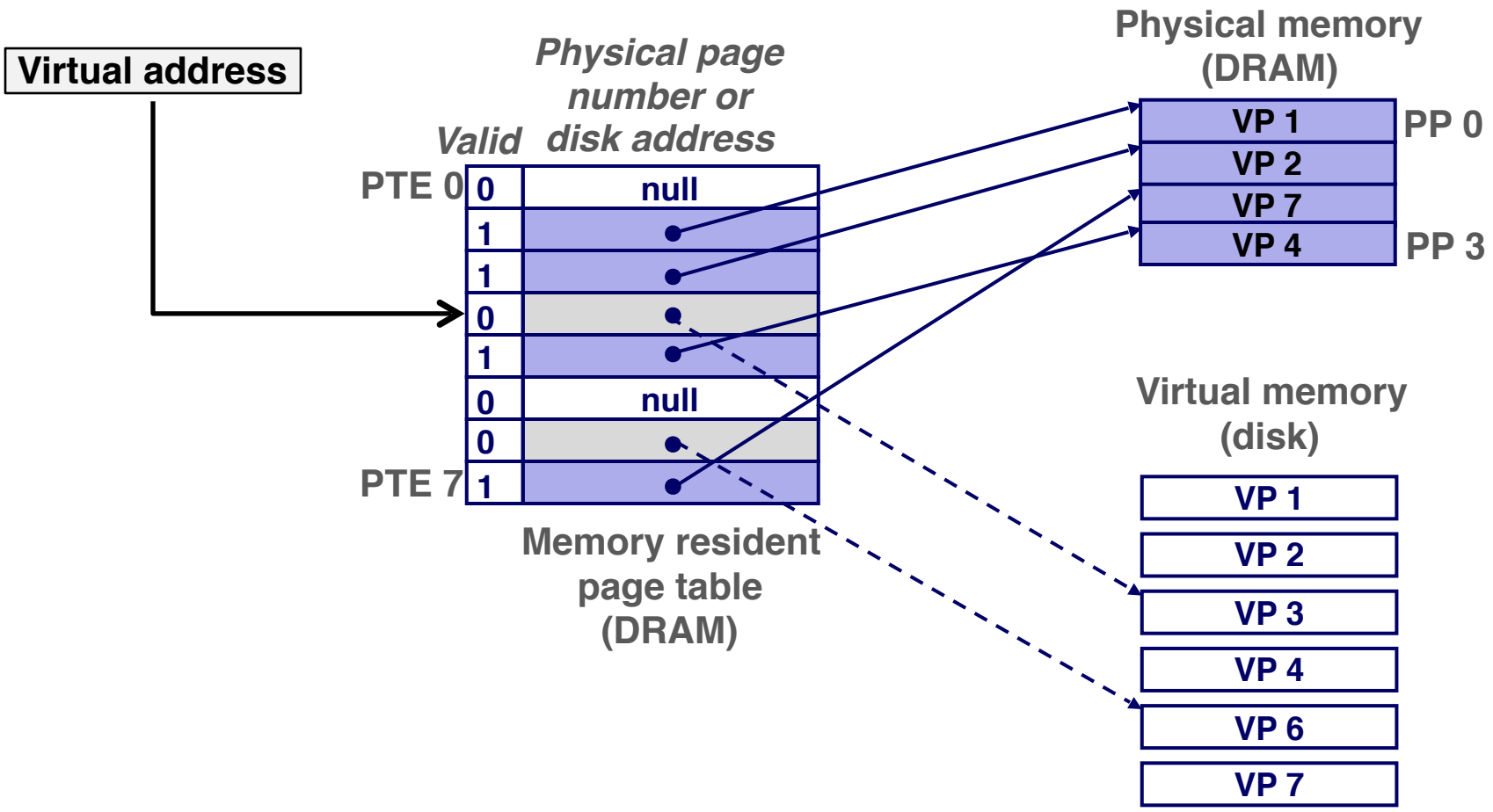
Page Fault

- *Page fault:* reference to VM word that is not in physical memory



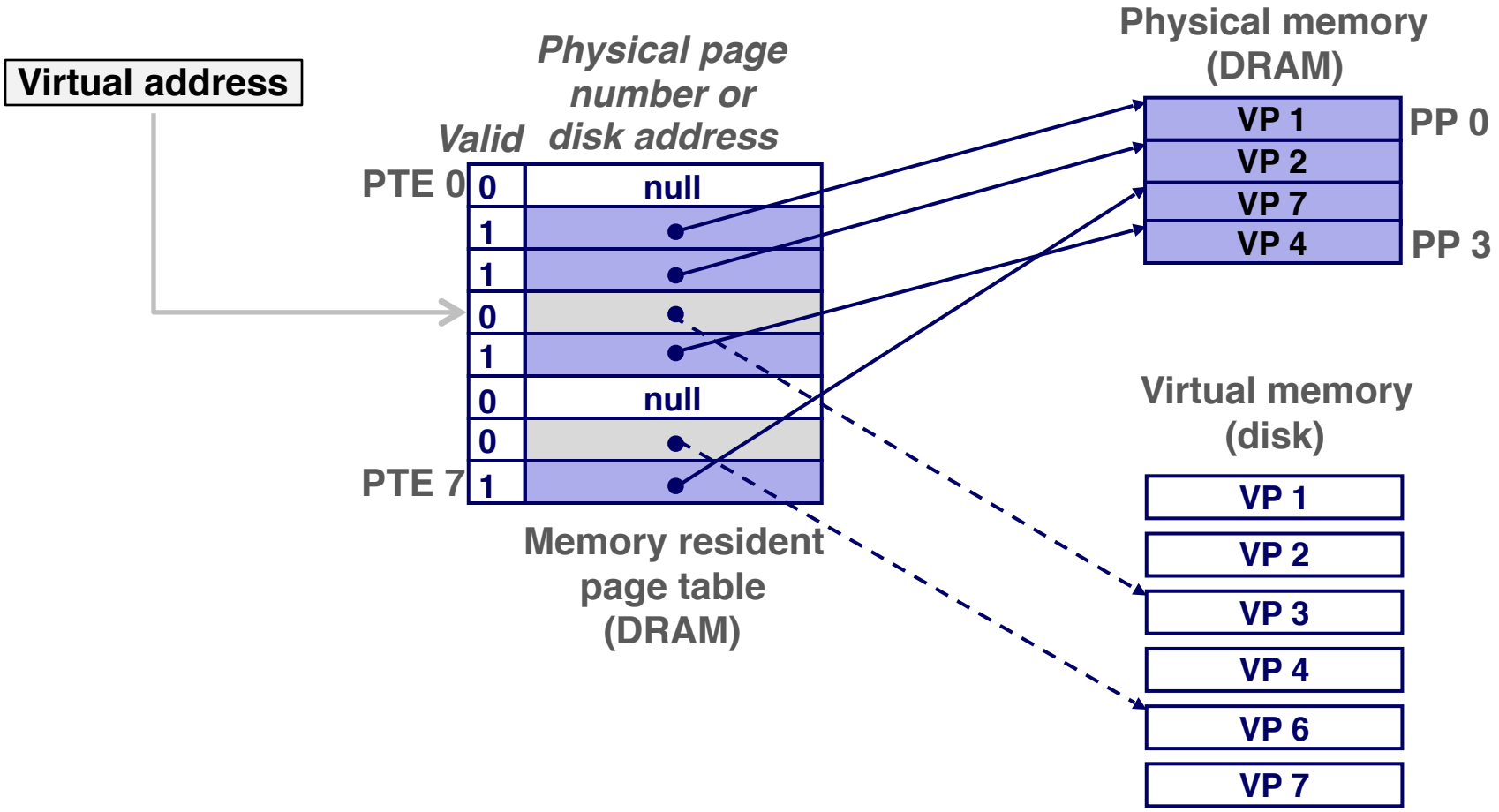
Page Fault

- *Page fault:* reference to VM word that is not in physical memory



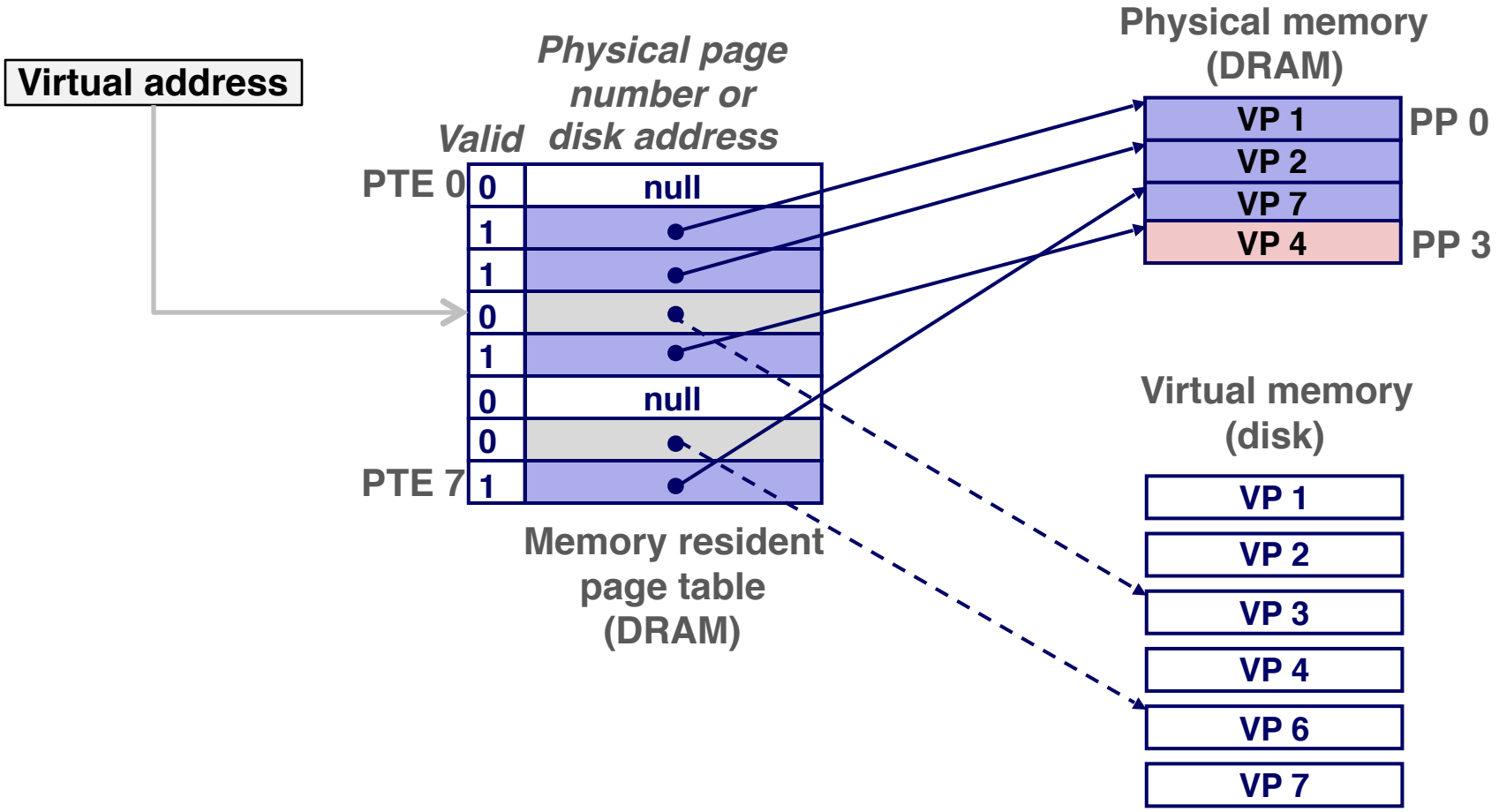
Handling Page Fault

- Page miss causes page fault (an exception)



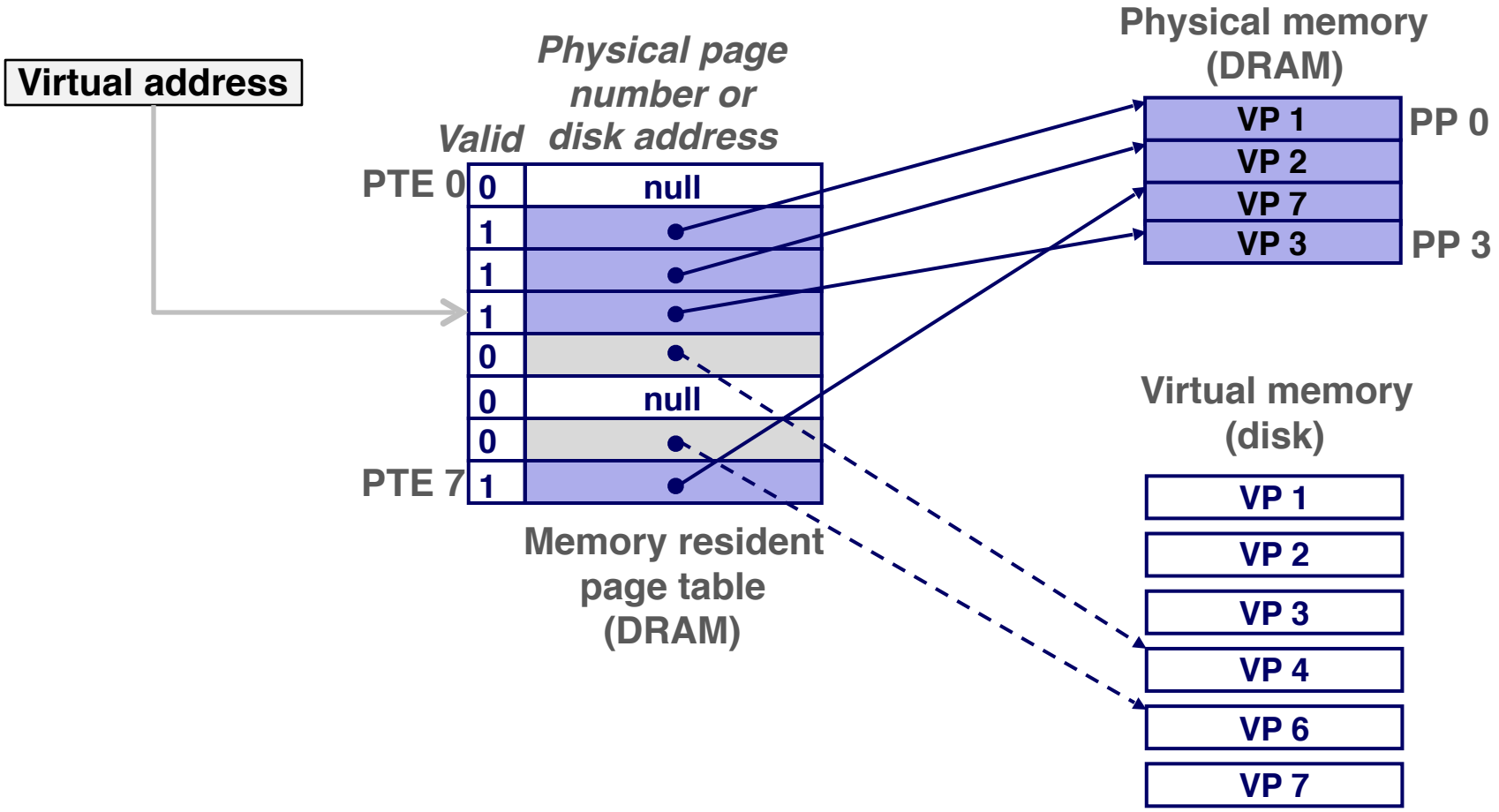
Handling Page Fault

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



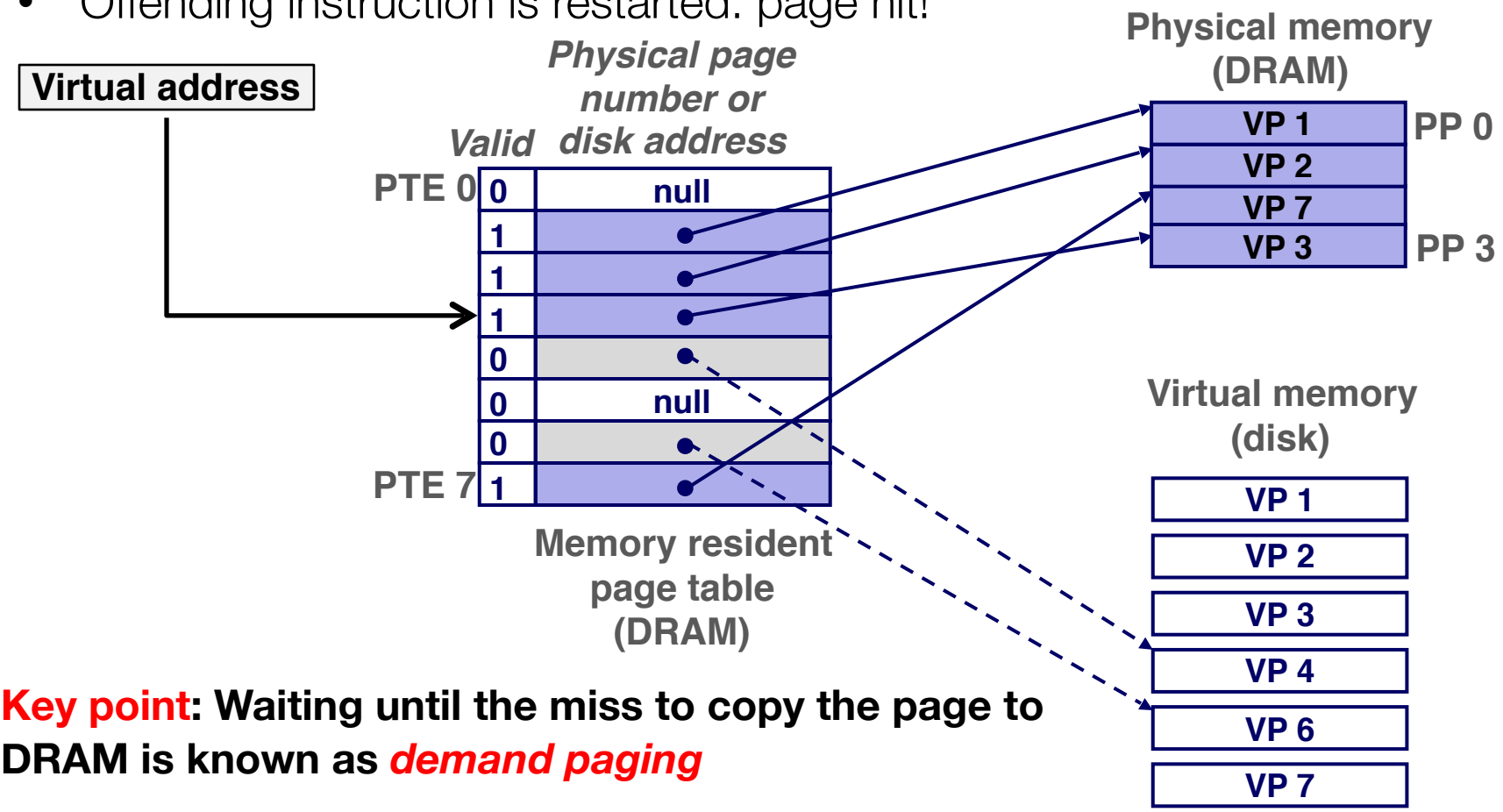
Handling Page Fault

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



Handling Page Fault

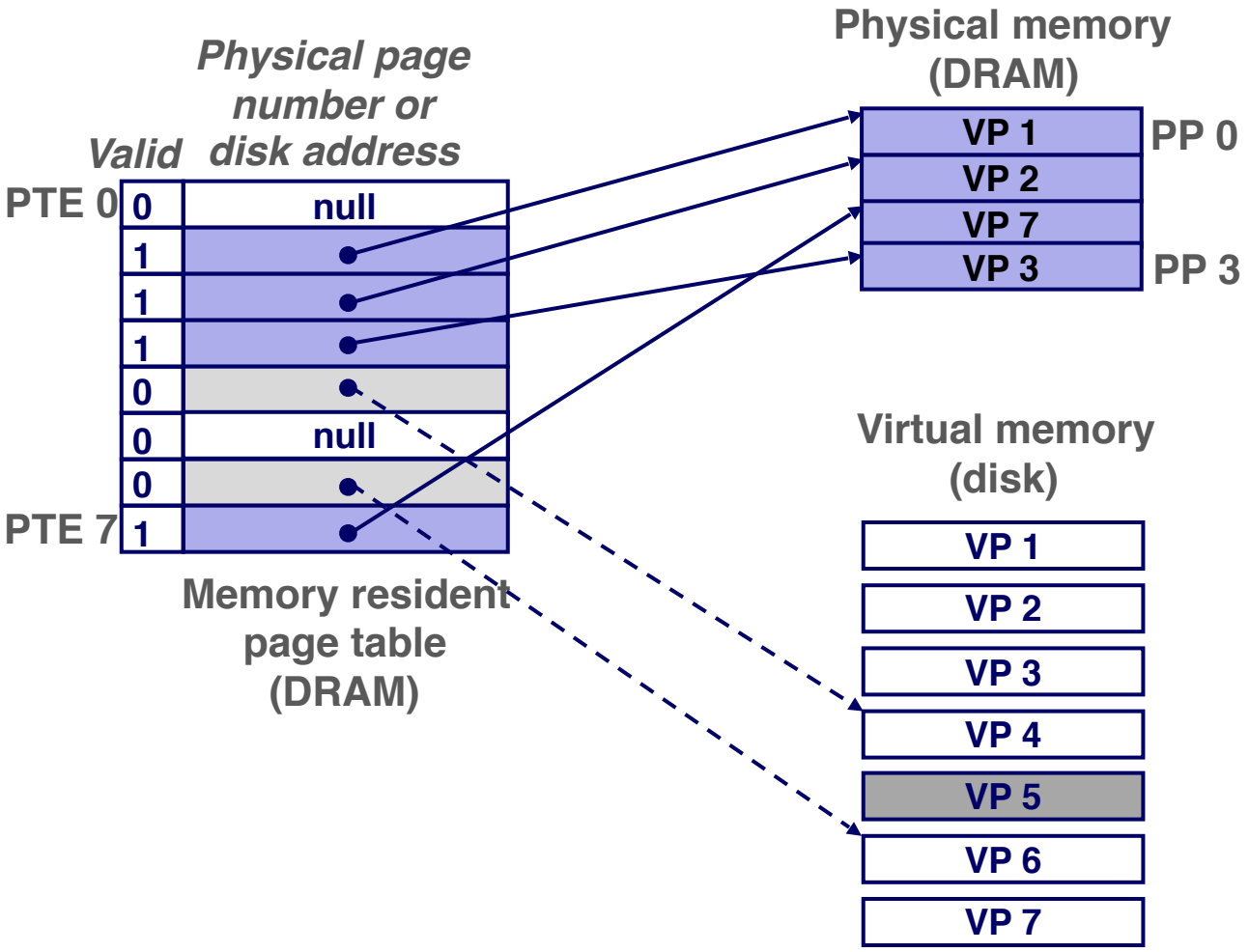
- Page miss causes page fault (an exception)
- Page fault handler selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!



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

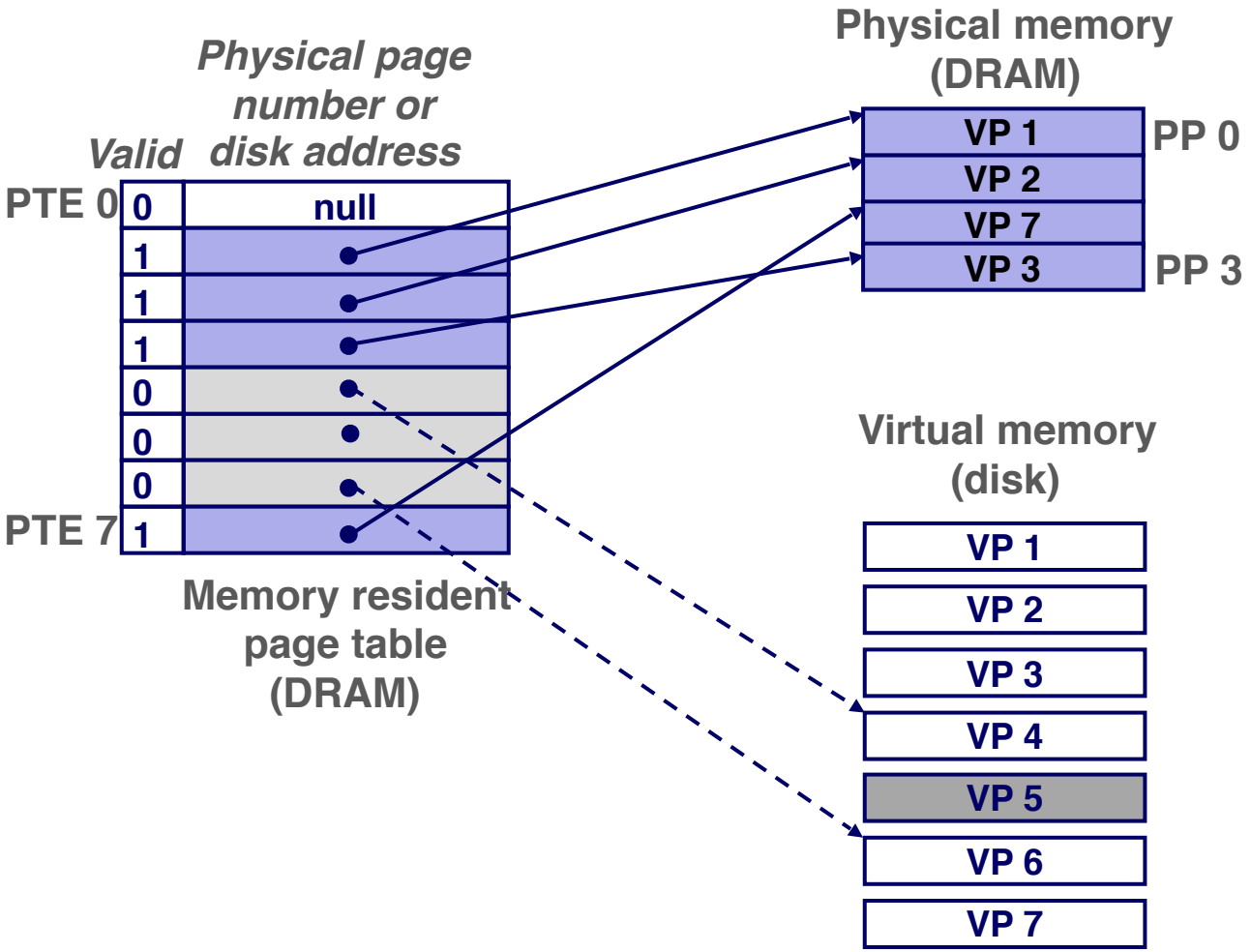
Allocating Pages

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



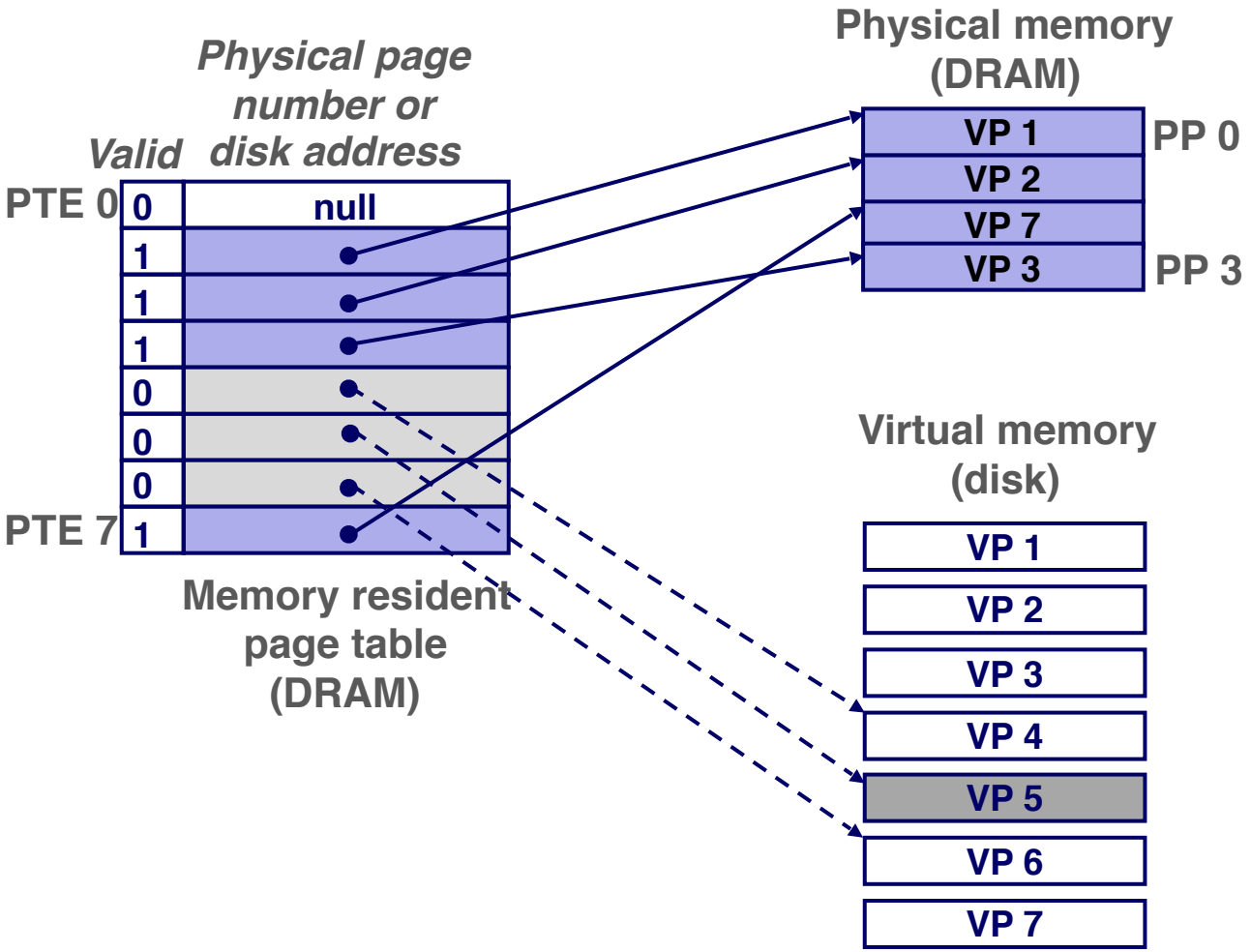
Allocating Pages

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



Allocating Pages

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



Virtual Memory Exploits Locality (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 initial misses

- If (SUM(working set sizes) > main memory size)
  - *Thrashing:* Performance meltdown where pages are swapped (copied) in and out continuously

Where Does Page Table Live?

Where Does Page Table Live?

- It needs to be at a specific location where we can find it
  - Some special SRAM?
  - In main memory?
  - On disk?

Where Does Page Table Live?

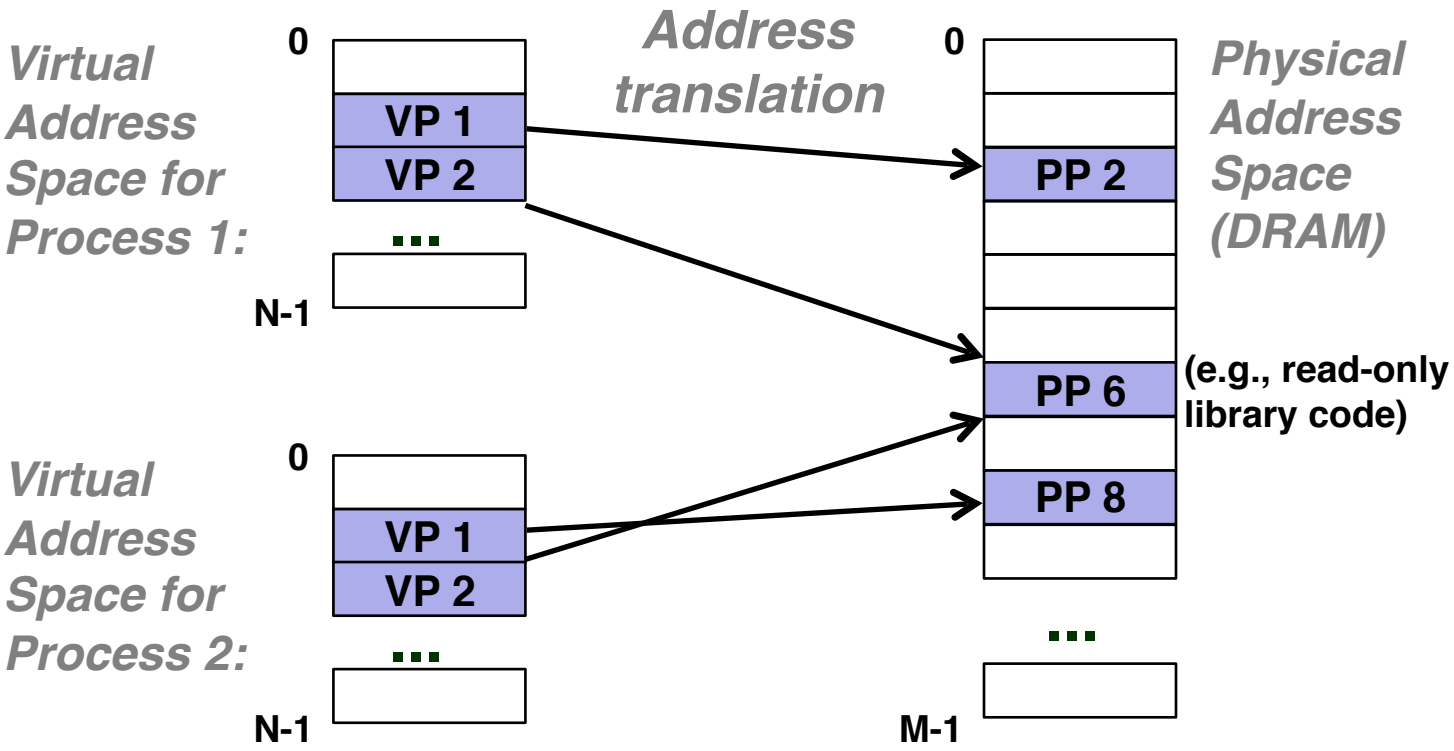
- It needs to be at a specific location where we can find it
  - Some special SRAM?
  - In main memory?
  - On disk?
- Assume 4KB page, 4GB main memory, each PTE is 8 Bytes
  - 1M PTEs in a page table
  - 8MB total size per page table
  - Too big for on-chip SRAM
  - Too slow to access in disk
  - Put the page table in DRAM, with its start address stored in a special register (Page Table Base Register)

Today

- Virtual memory (VM) illustration
- VM basic concepts and operation
- Other critical benefits of VM
- 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 scatters addresses through physical memory
    - Well-chosen mappings can improve locality



Virtual Memory Enables Isolations

- If all processes use physical address, it would be easy for one program to modify the data of another program. This is obviously a huge security and privacy issue.

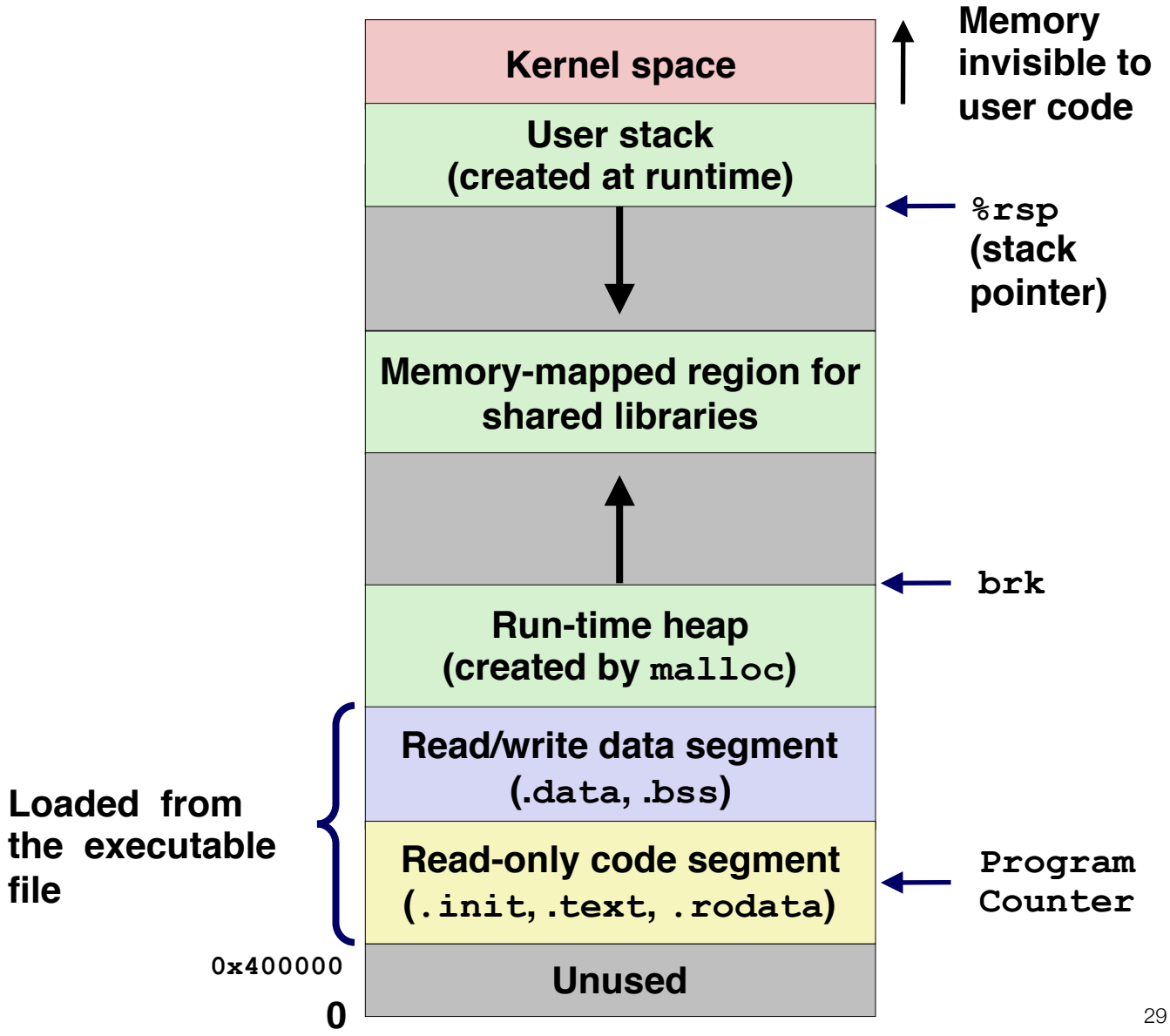
Virtual Memory Enables Isolations

- If all processes use physical address, it would be easy for one program to modify the data of another program. This is obviously a huge security and privacy issue.
- Early days (e.g., EDSAC in 50's), ISA use physical address. To address the security issue, a program is loaded to a different address in memory every time it runs.
  - not ideal: address in programs depend on where the program is loaded in memory

Virtual Memory Enables Isolations

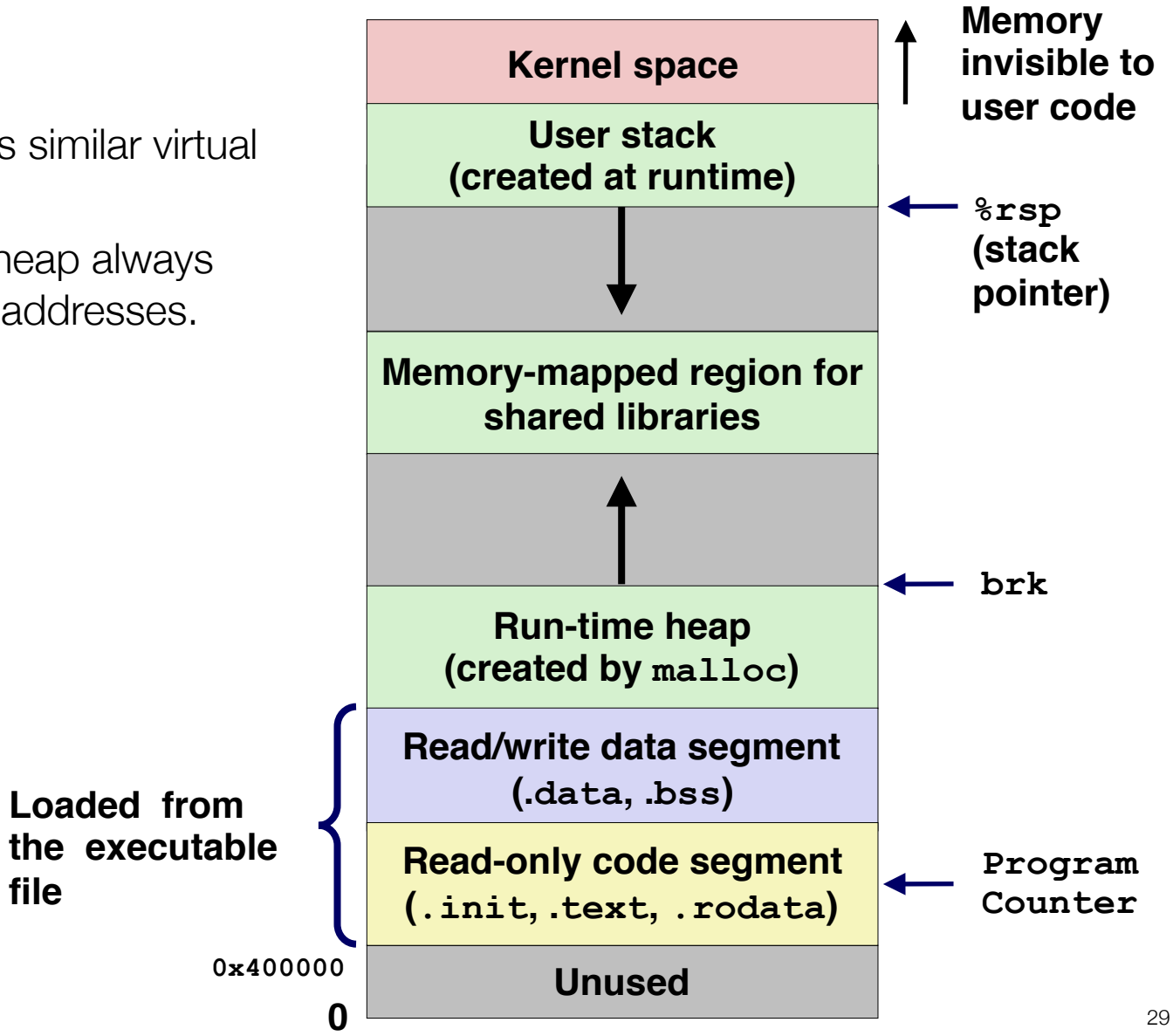
- If all processes use physical address, it would be easy for one program to modify the data of another program. This is obviously a huge security and privacy issue.
- Early days (e.g., EDSAC in 50's), ISA use physical address. To address the security issue, a program is loaded to a different address in memory every time it runs.
  - not ideal: address in programs depend on where the program is loaded in memory
- With virtual memory, addresses used by program are not the same as what the processor uses to actually access memory. This naturally isolates/protect programs.

Simplifying Linking and Loading



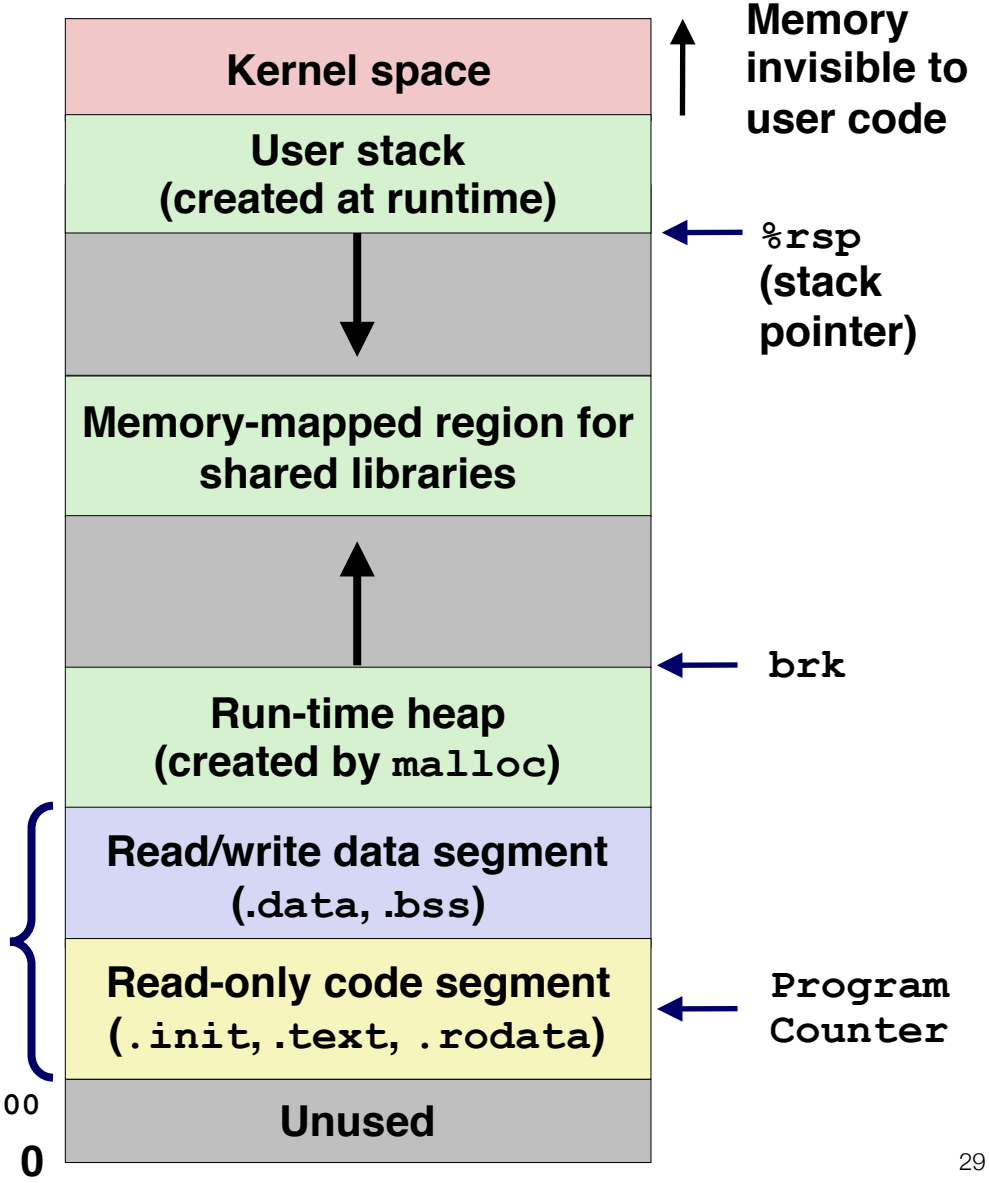
Simplifying Linking and Loading

- Linking
  - Each program has similar virtual address space
  - Code, data, and heap always start at the same addresses.



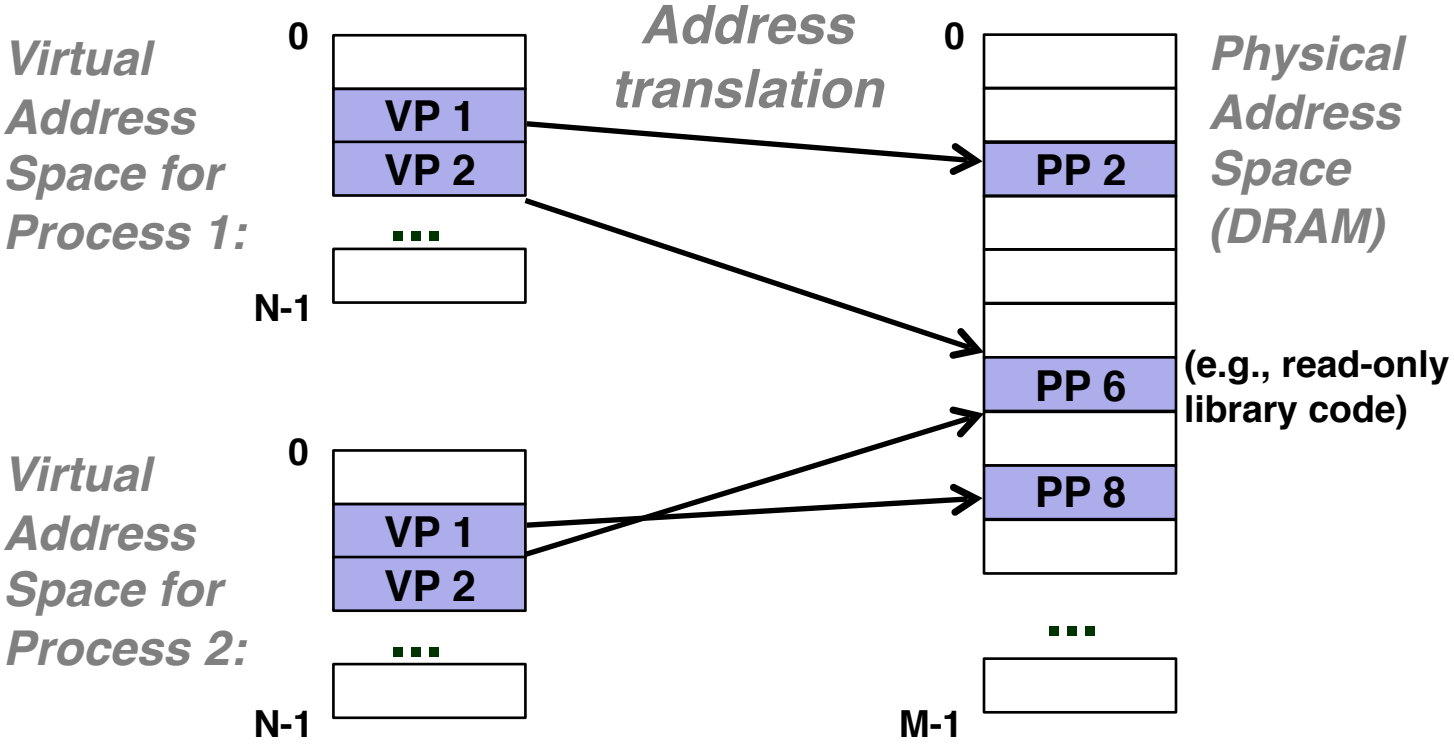
Simplifying Linking and Loading

- 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 VM system



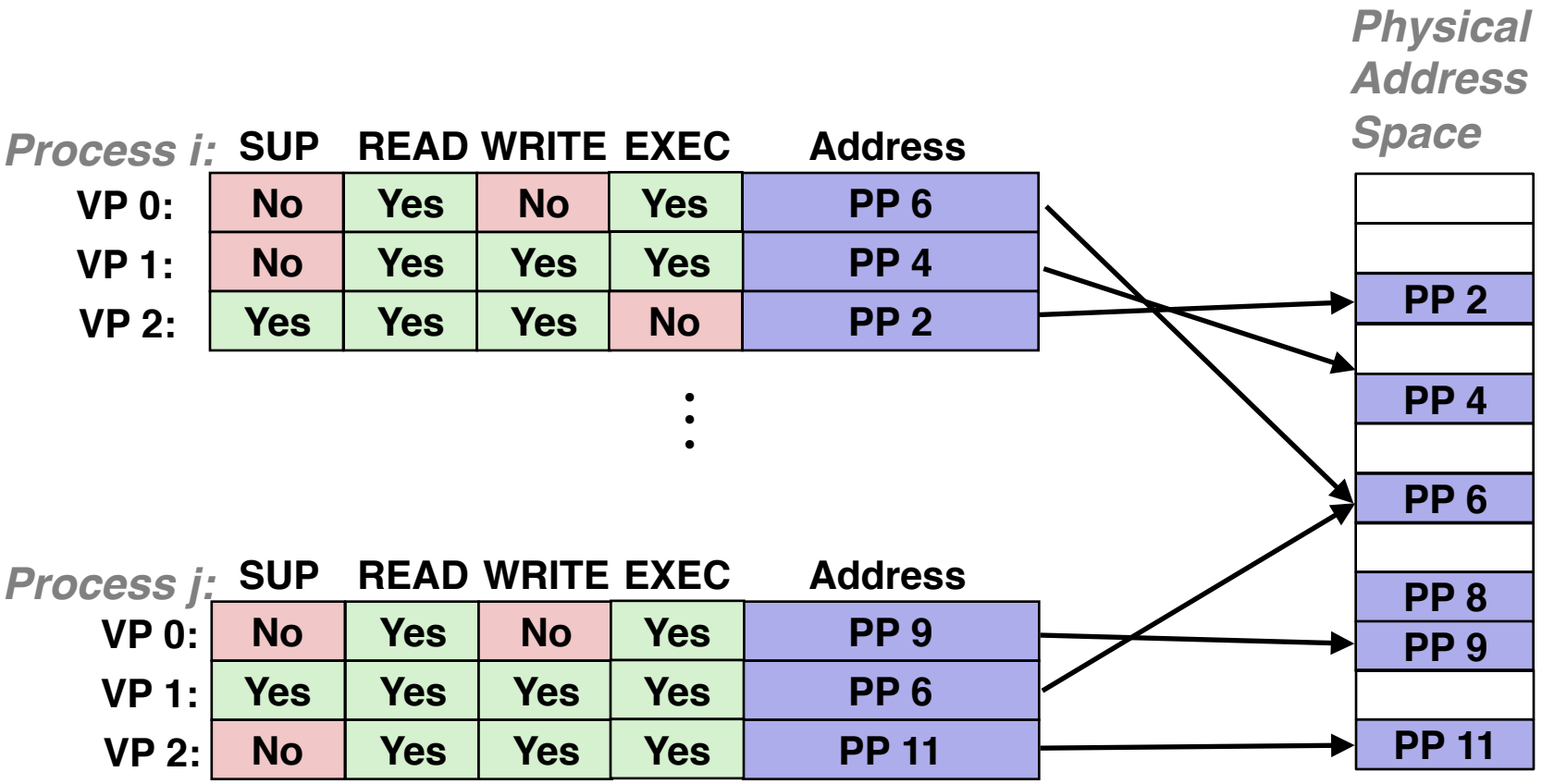
Virtual Memory Enables Sharing

- 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)



VM Provides Further Protection Opportunities

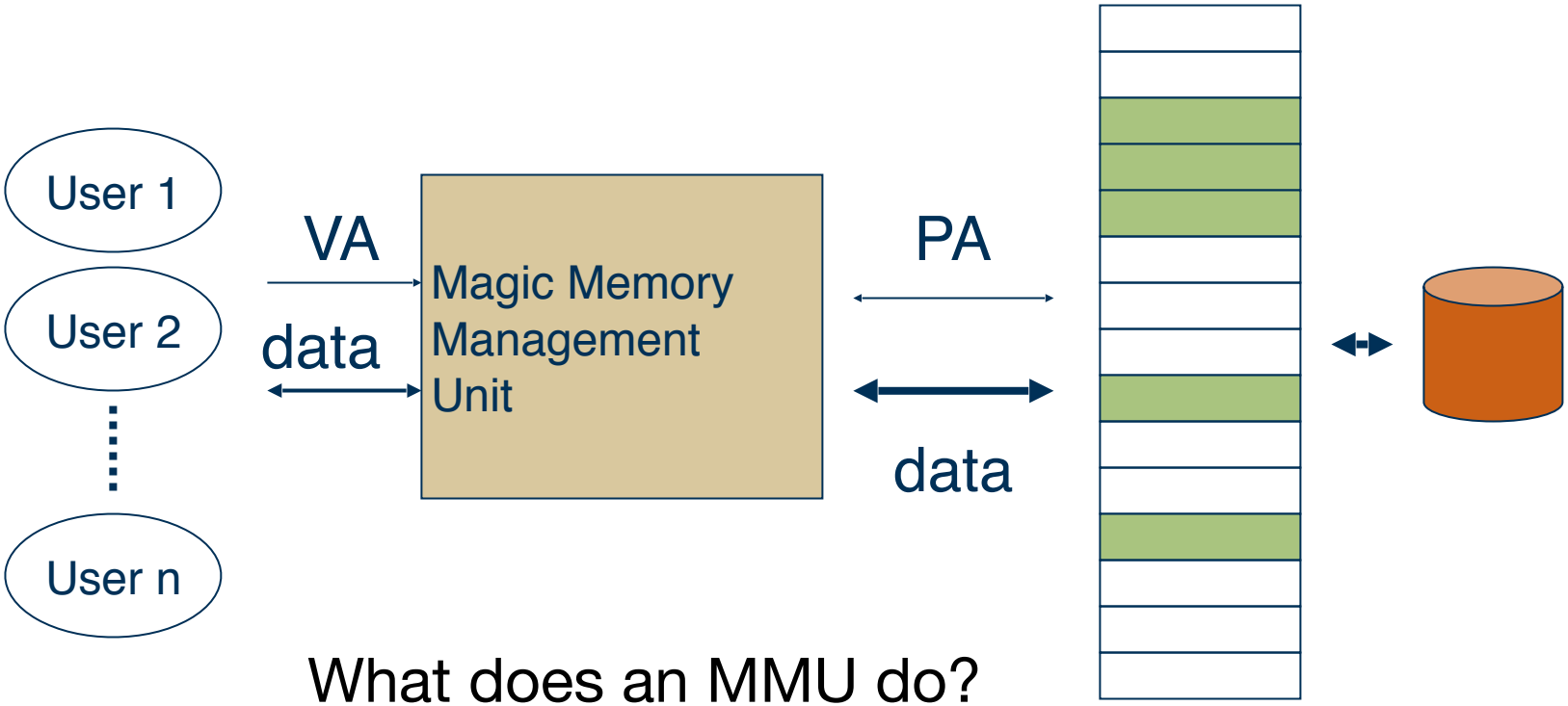
- Extend PTEs with permission bits
- MMU checks these bits on each access



Today

- Virtual memory (VM) illustration
- VM basic concepts and operation
- Other critical benefits of VM
- Address translation

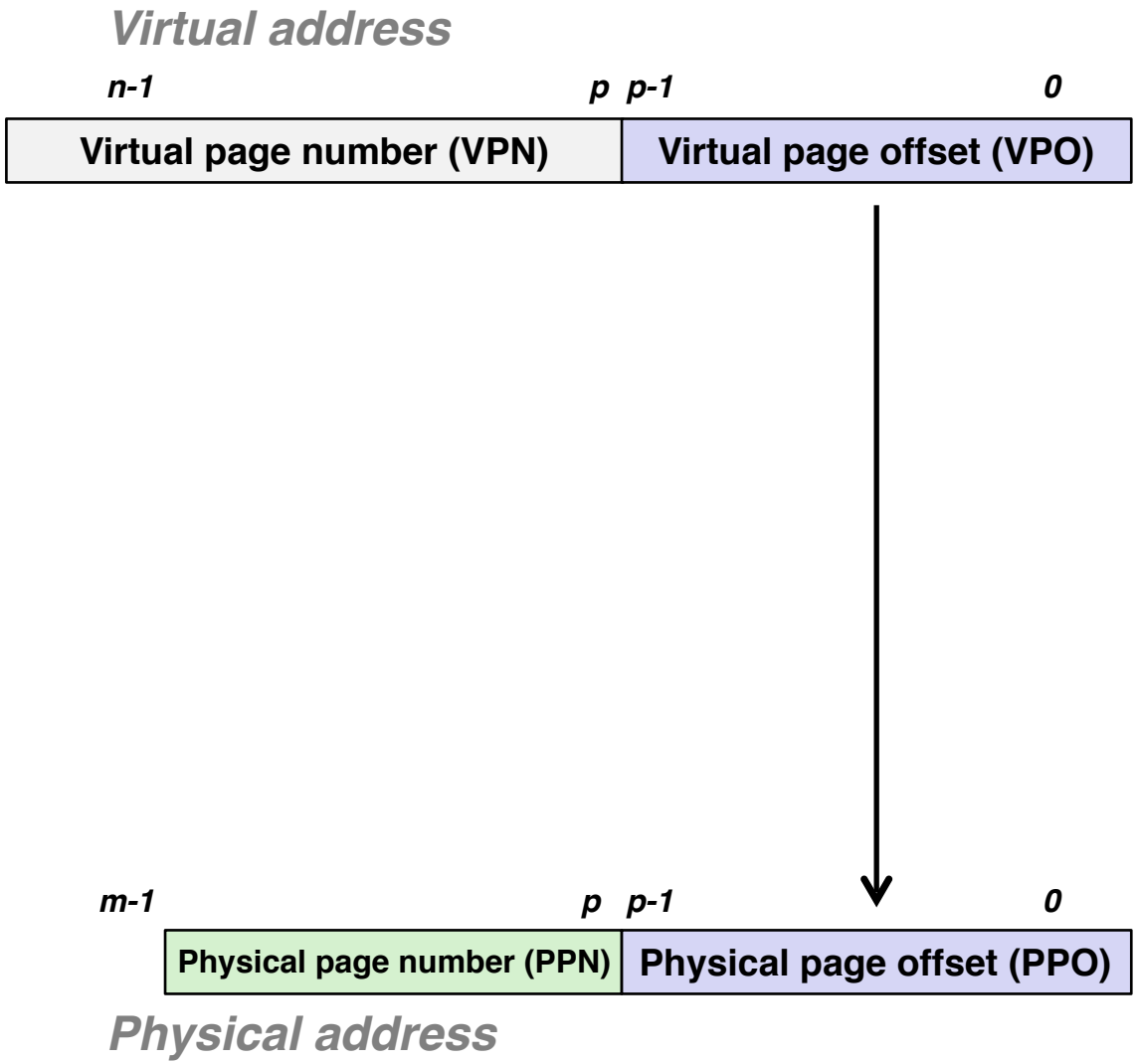
So Far…



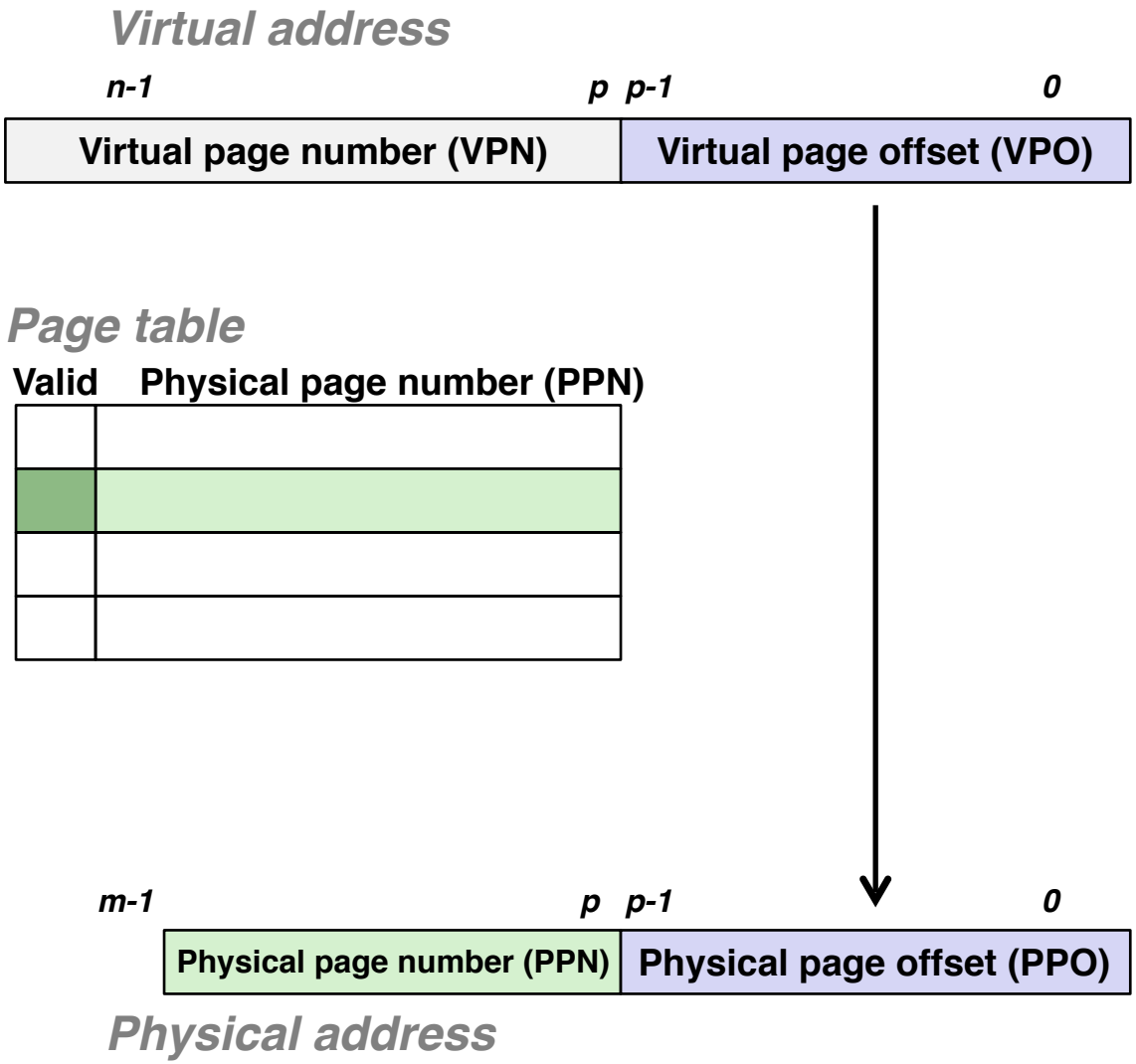
What does an MMU do?

- Translate address
  - Enforce permissions
  - Fetch from disk

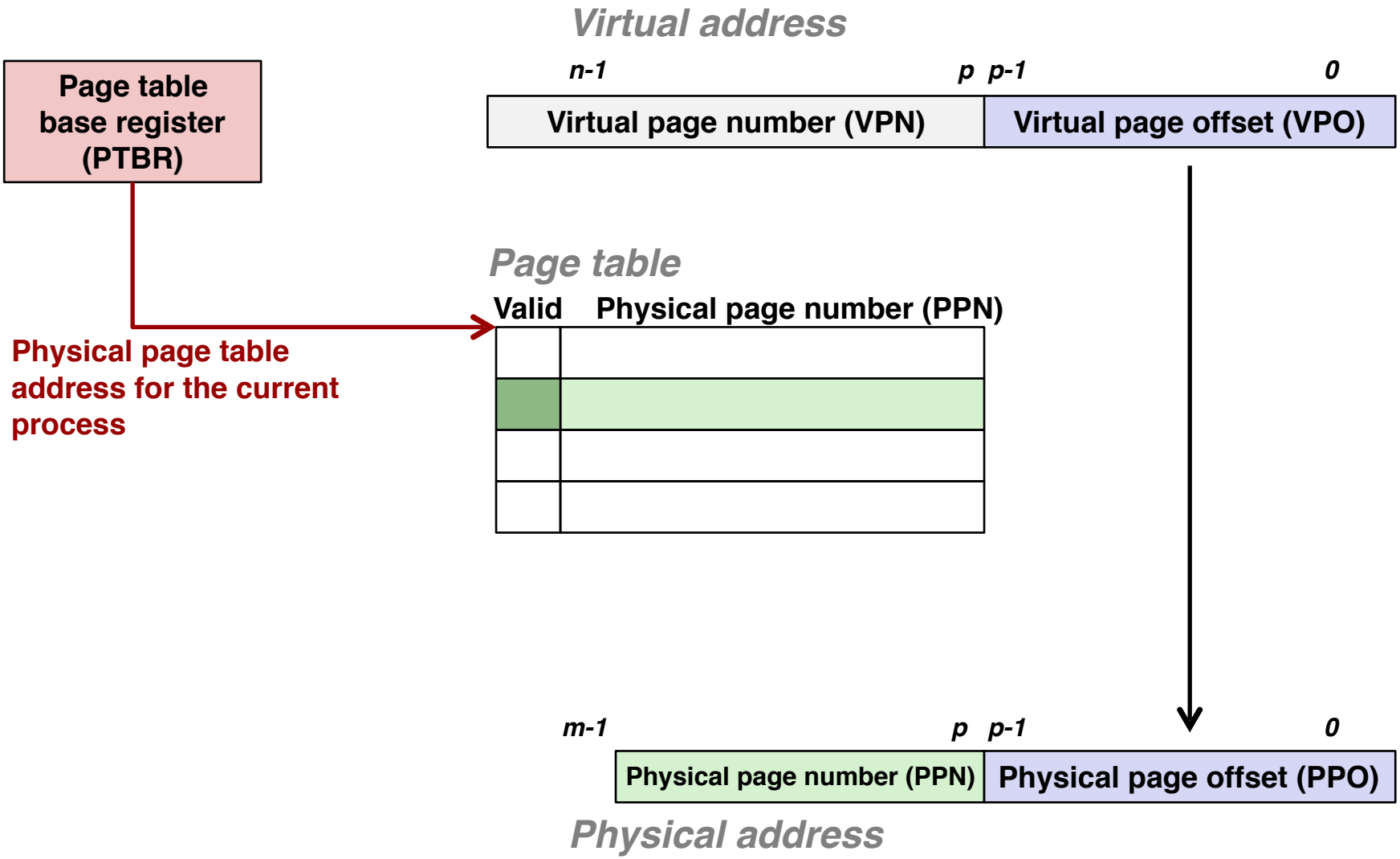
Address Translation With a Page Table



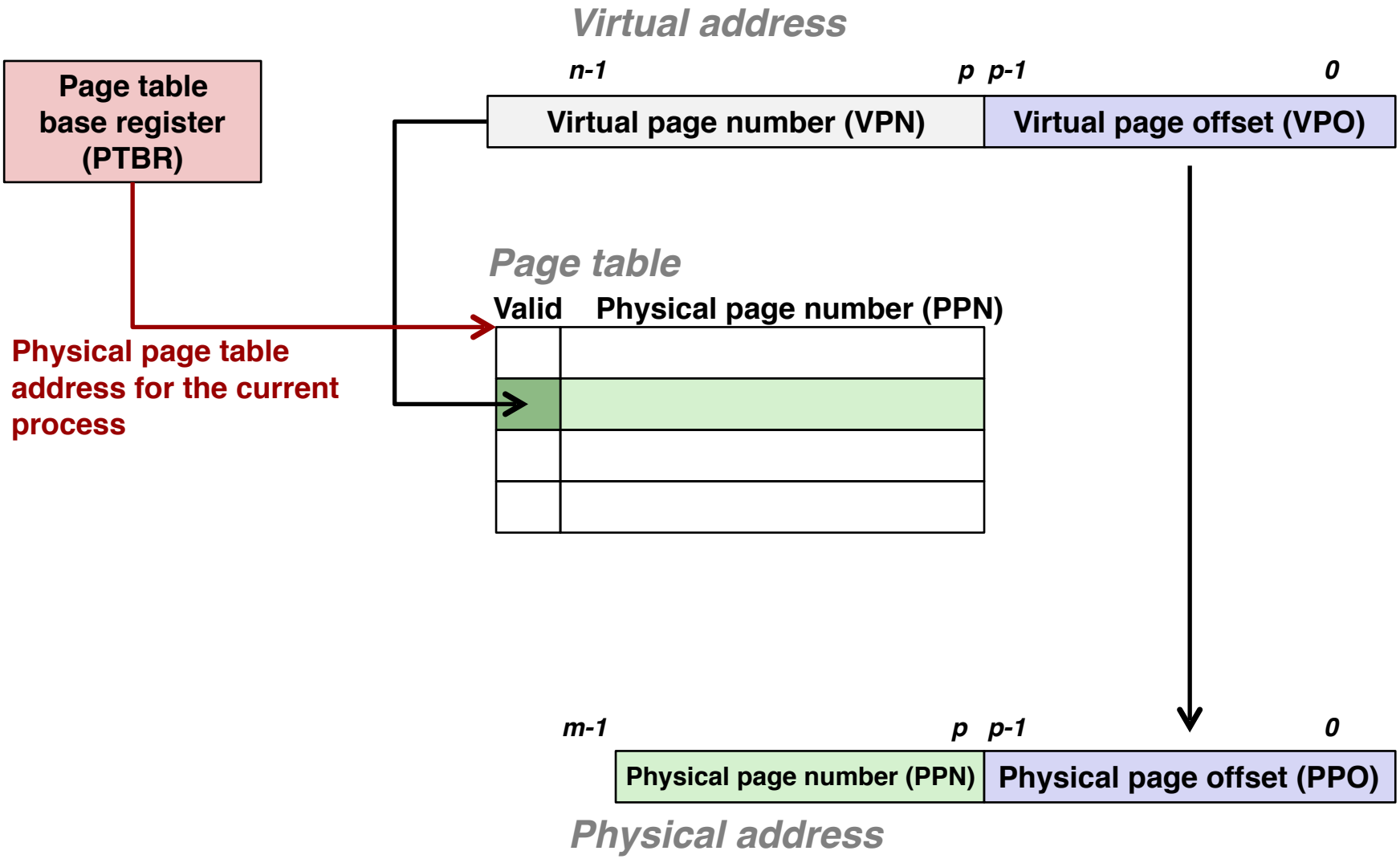
Address Translation With a Page Table



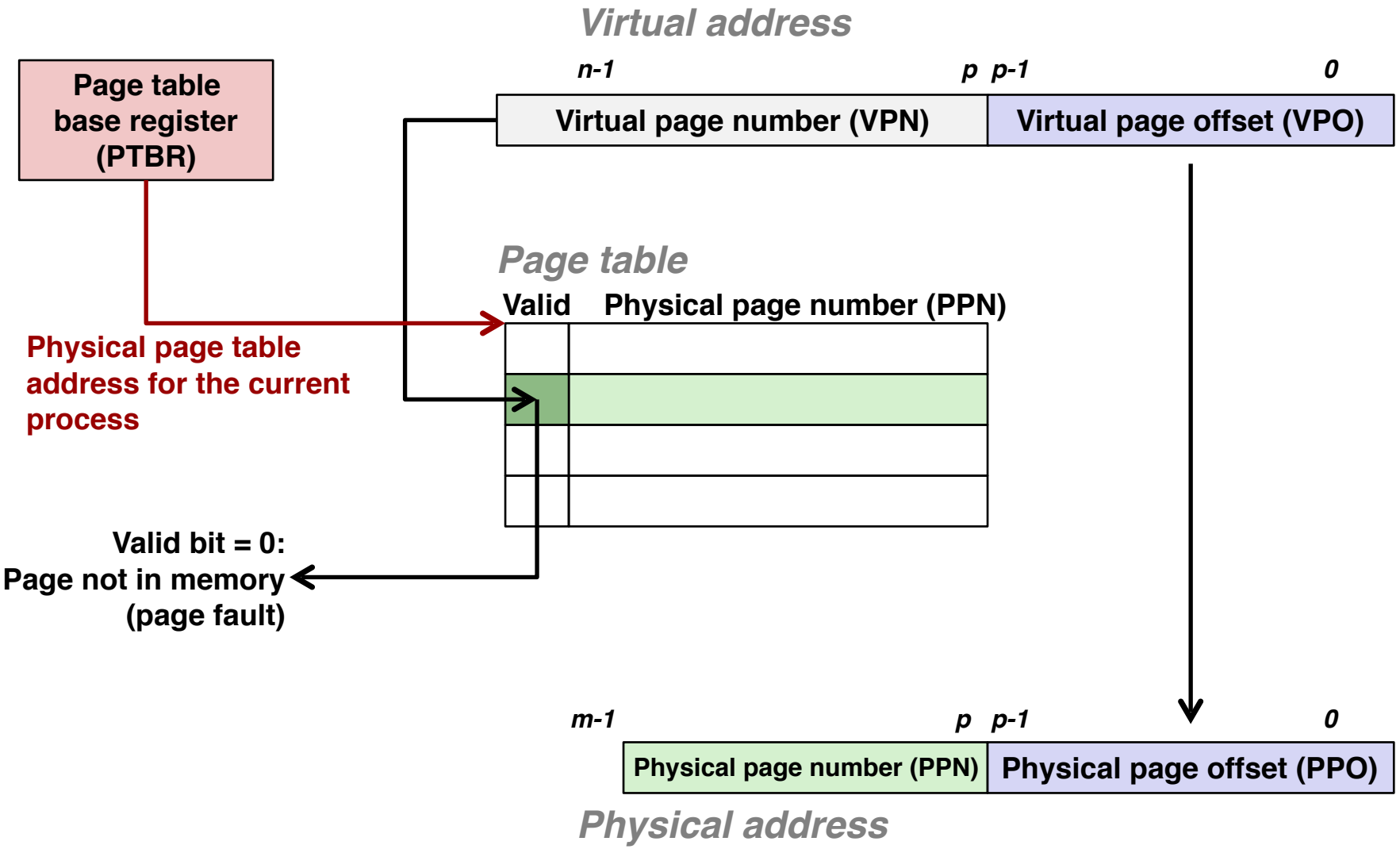
Address Translation With a Page Table



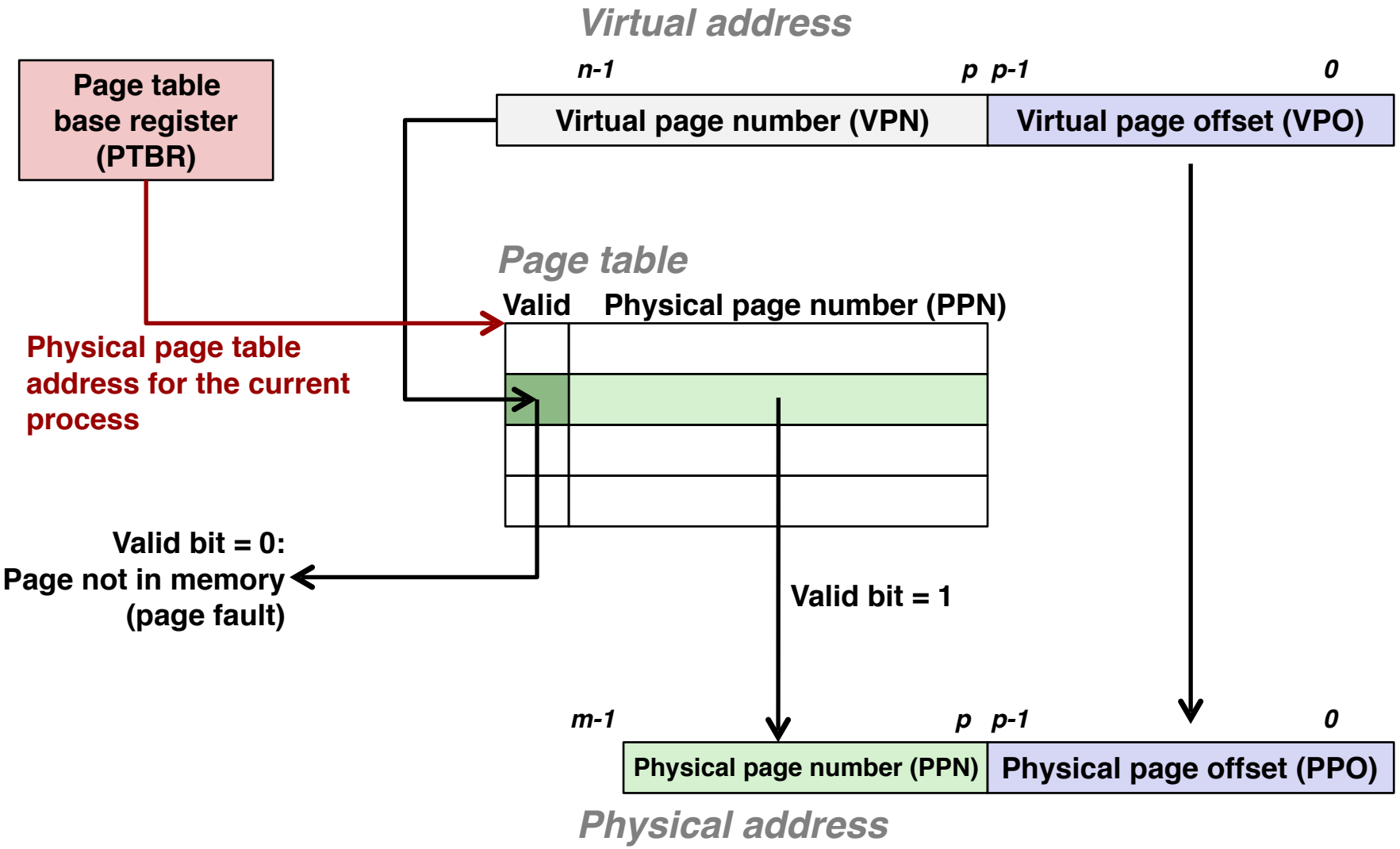
Address Translation With a Page Table



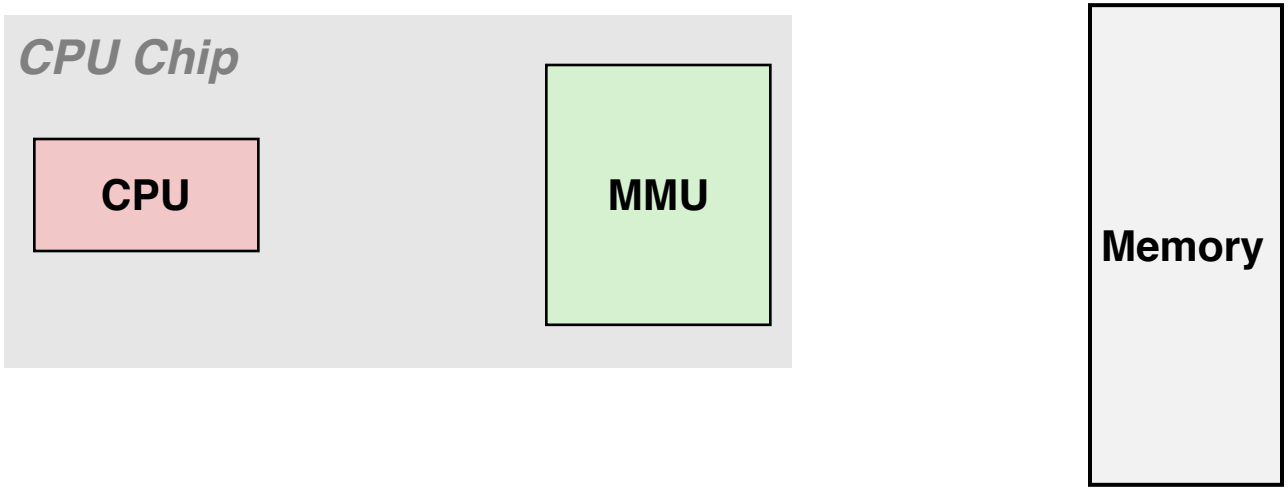
Address Translation With a Page Table



Address Translation With a Page Table

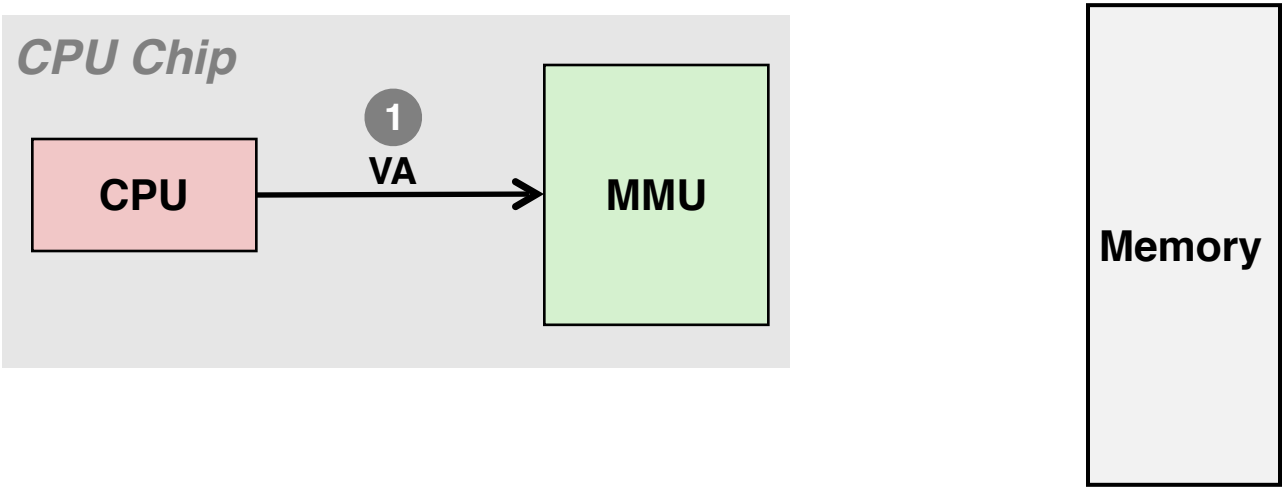


Address Translation: Page Hit



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

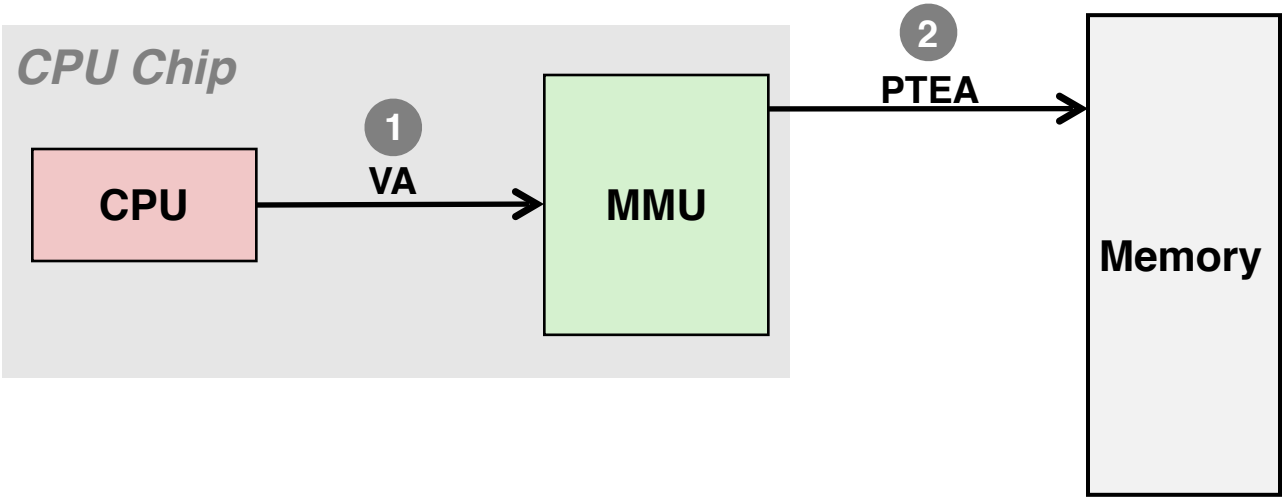
Address Translation: Page Hit



1) Processor sends virtual address to MMU

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

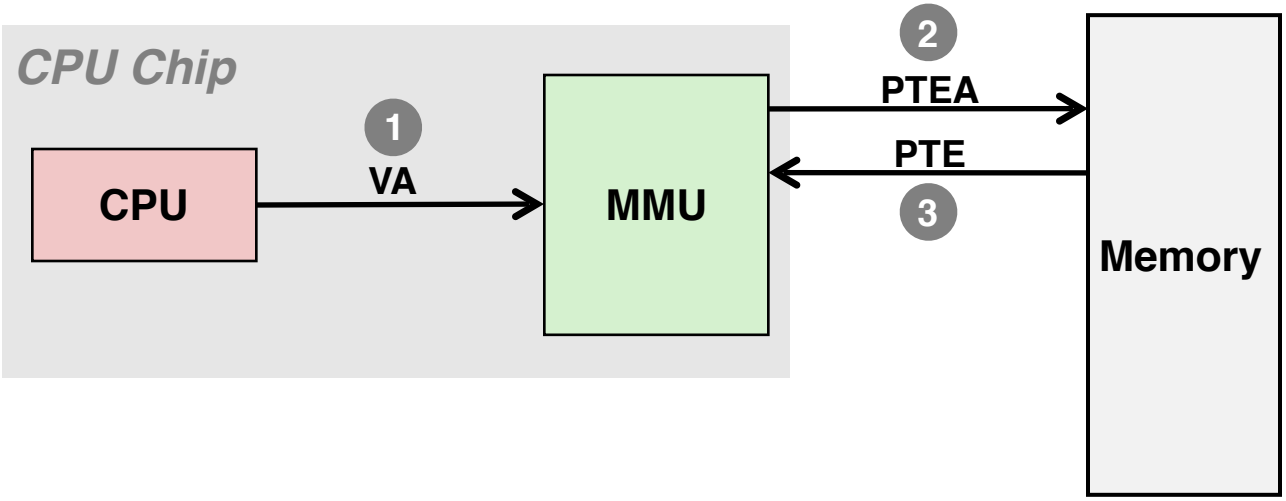
Address Translation: Page Hit



1) Processor sends virtual address to MMU

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

Address Translation: Page Hit

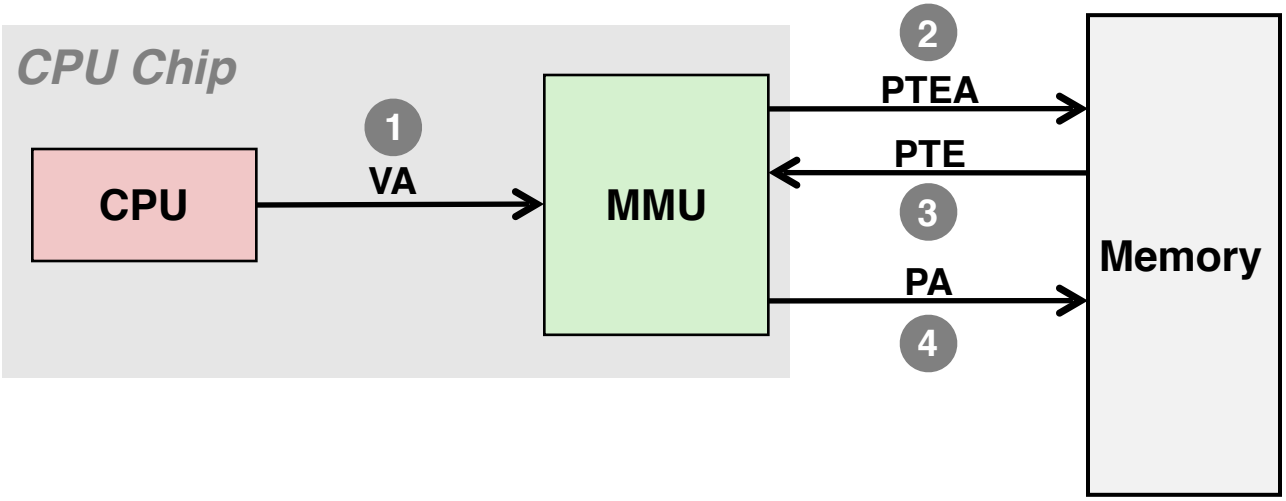


1) Processor sends virtual address to MMU

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

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

Address Translation: Page Hit



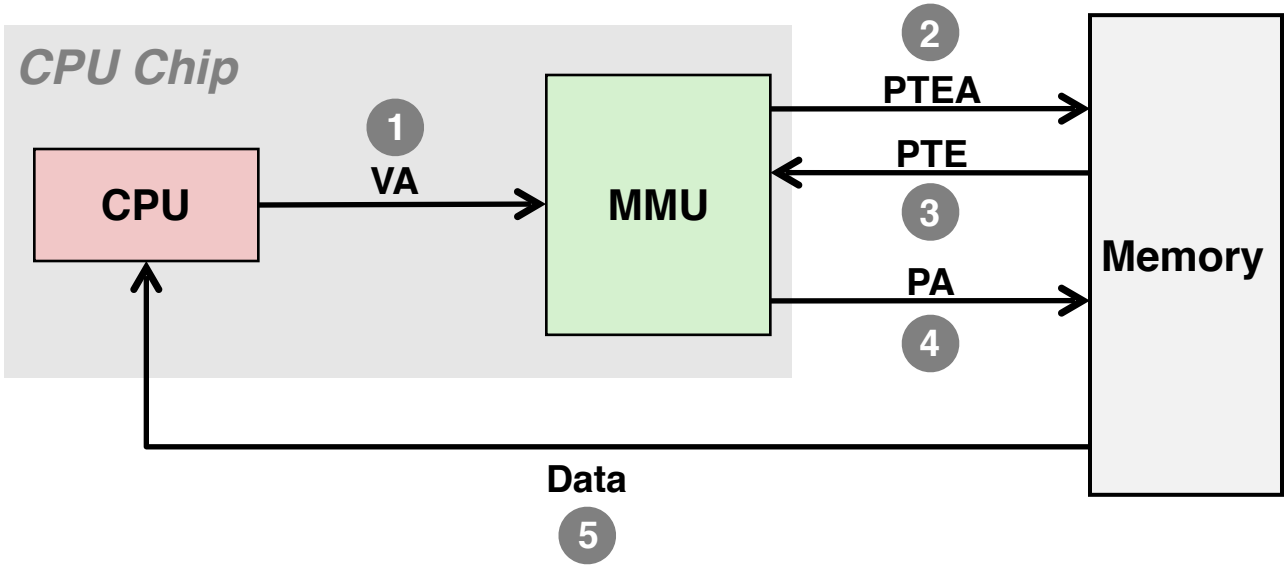
1) Processor sends virtual address to MMU

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

4) MMU sends physical address to cache/memory

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

Address Translation: Page Hit



1) Processor sends virtual address to MMU

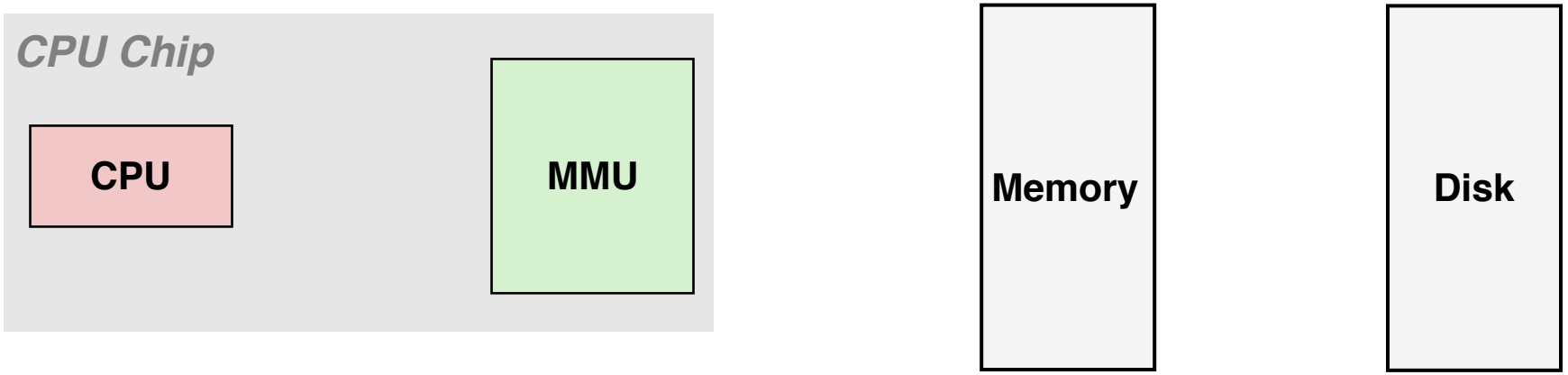
2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to cache/memory

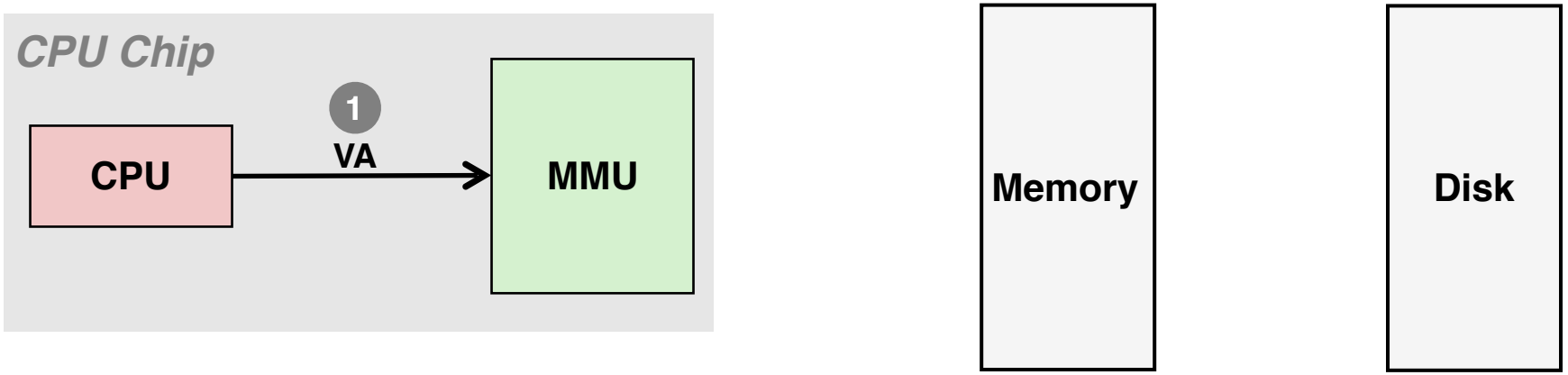
5) Cache/memory sends data word to processor

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

Address Translation: Page Fault

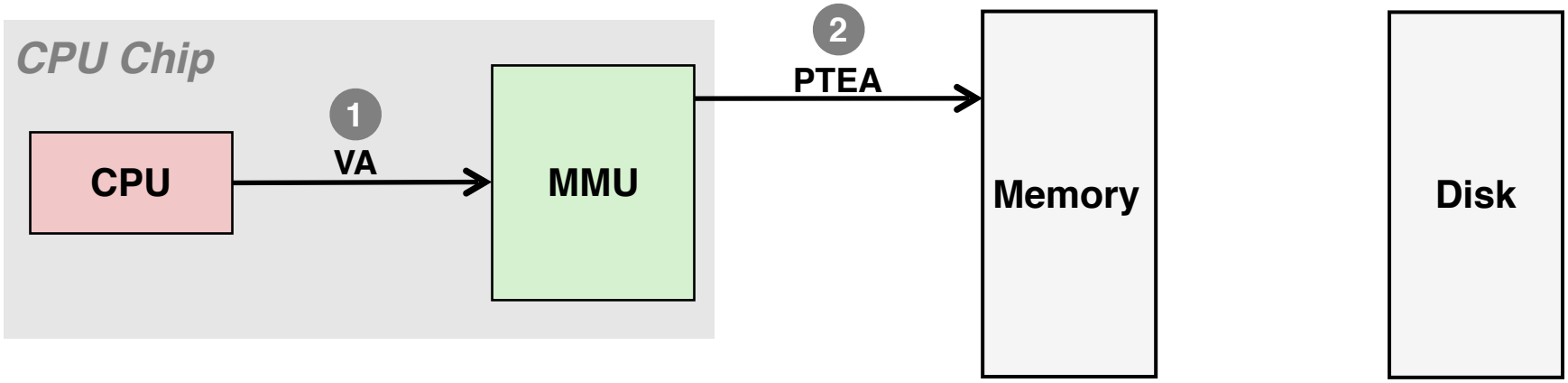


Address Translation: Page Fault



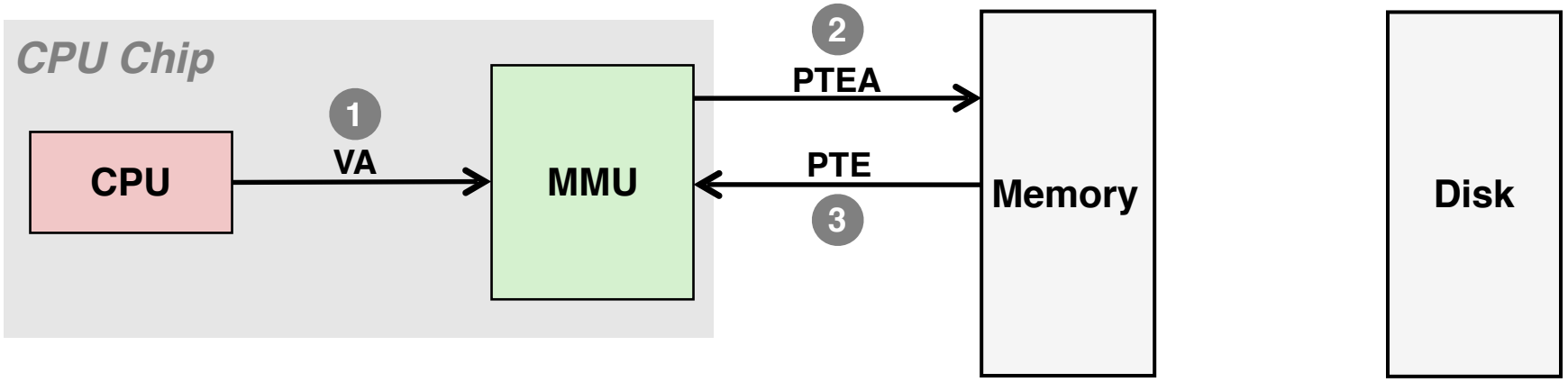
1) Processor sends virtual address to MMU

Address Translation: Page Fault



1) Processor sends virtual address to MMU

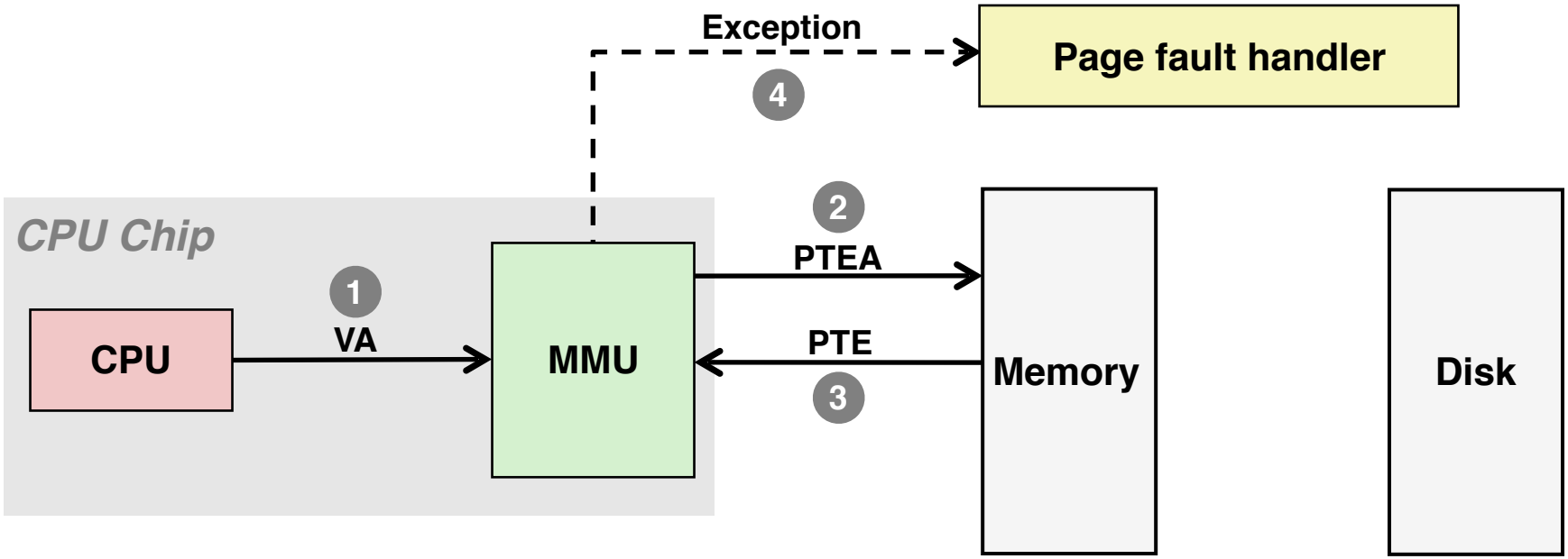
Address Translation: Page Fault



1) Processor sends virtual address to MMU

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

Address Translation: Page Fault

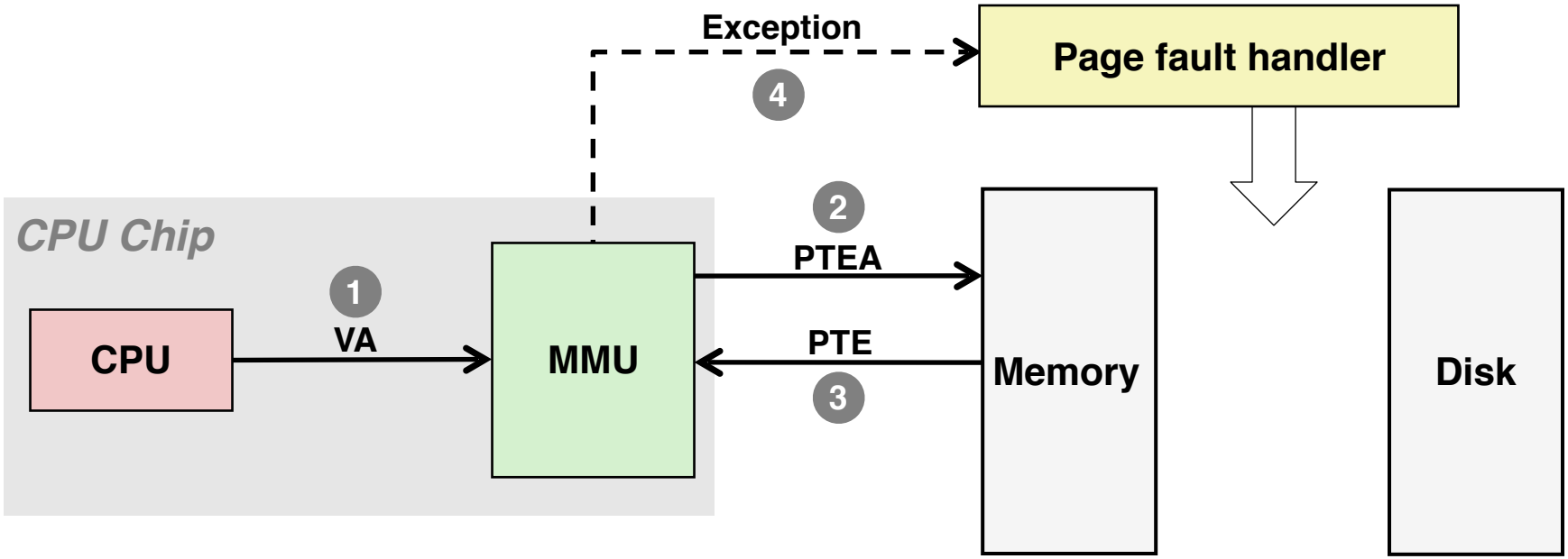


1) Processor sends virtual address to MMU

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

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

Address Translation: Page Fault

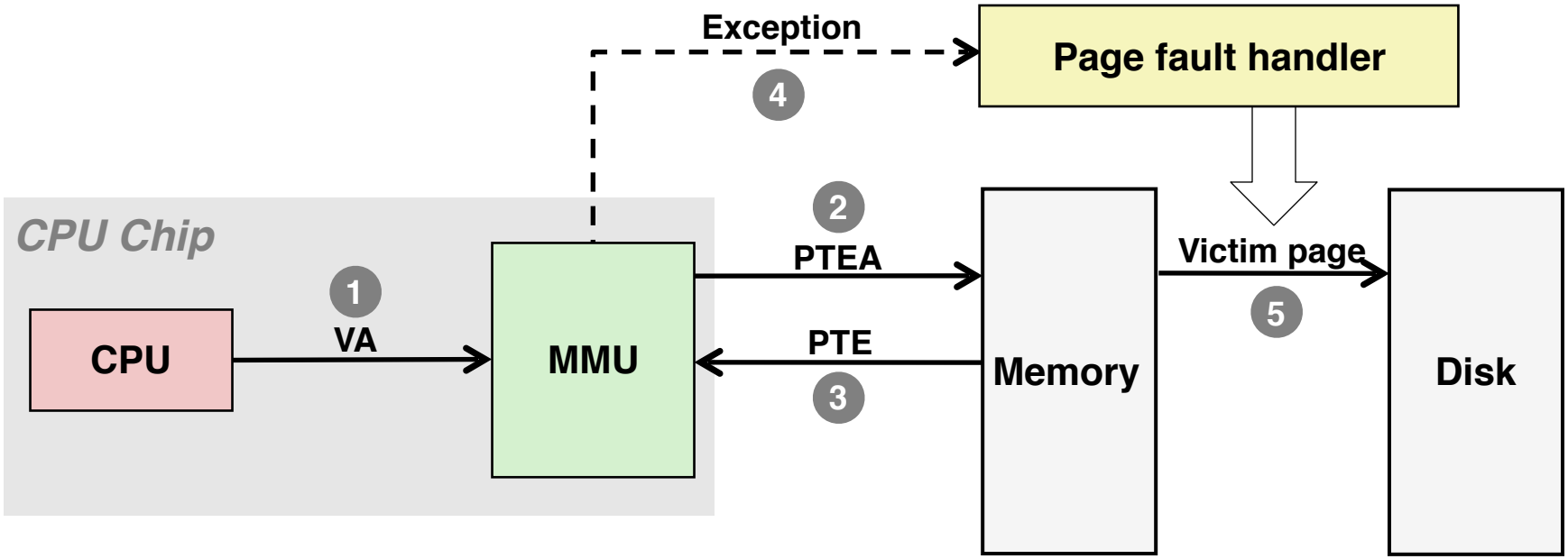


1) Processor sends virtual address to MMU

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

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

Address Translation: Page Fault



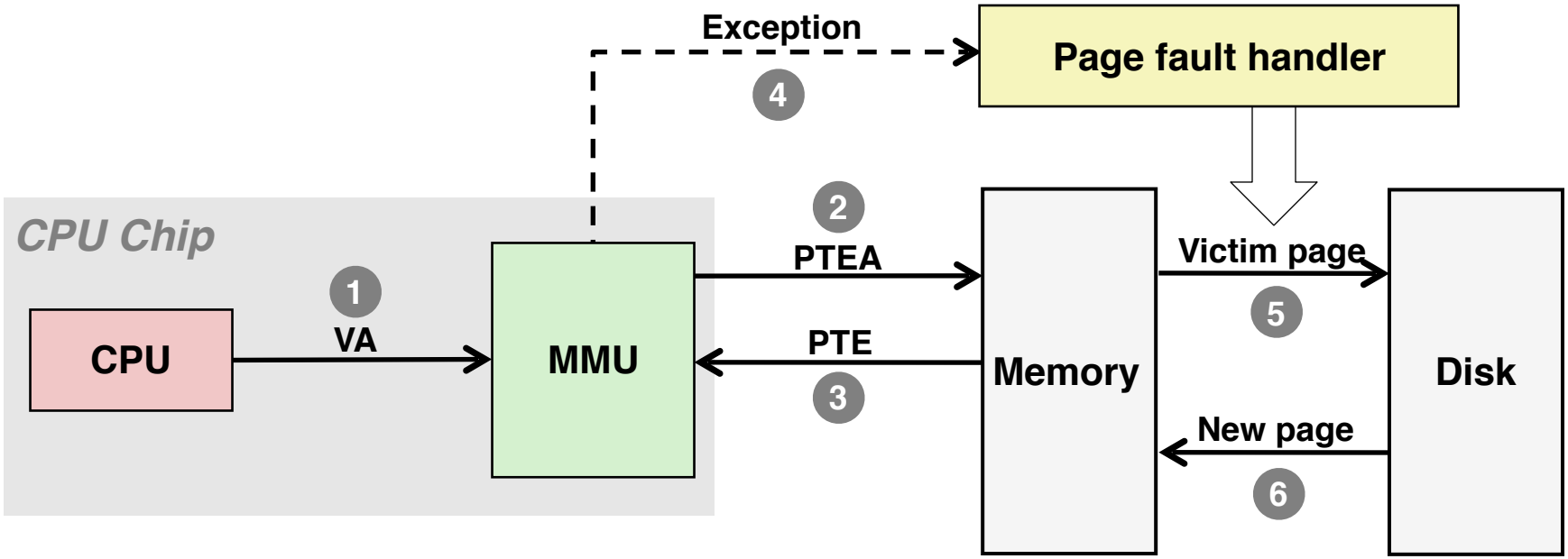
1) Processor sends virtual address to MMU

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)

Address Translation: Page Fault



1) Processor sends virtual address to MMU

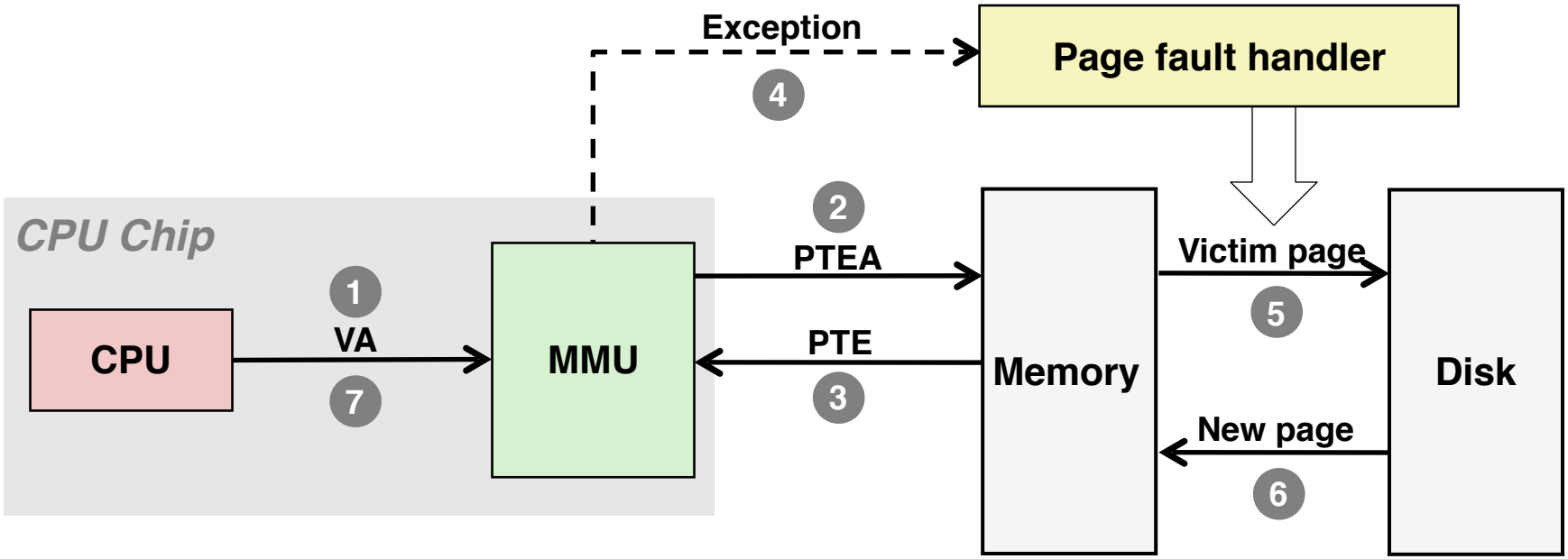
2-3) MMU fetches PTE from page table in memory

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

Address Translation: Page Fault



1) Processor sends virtual address to MMU

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

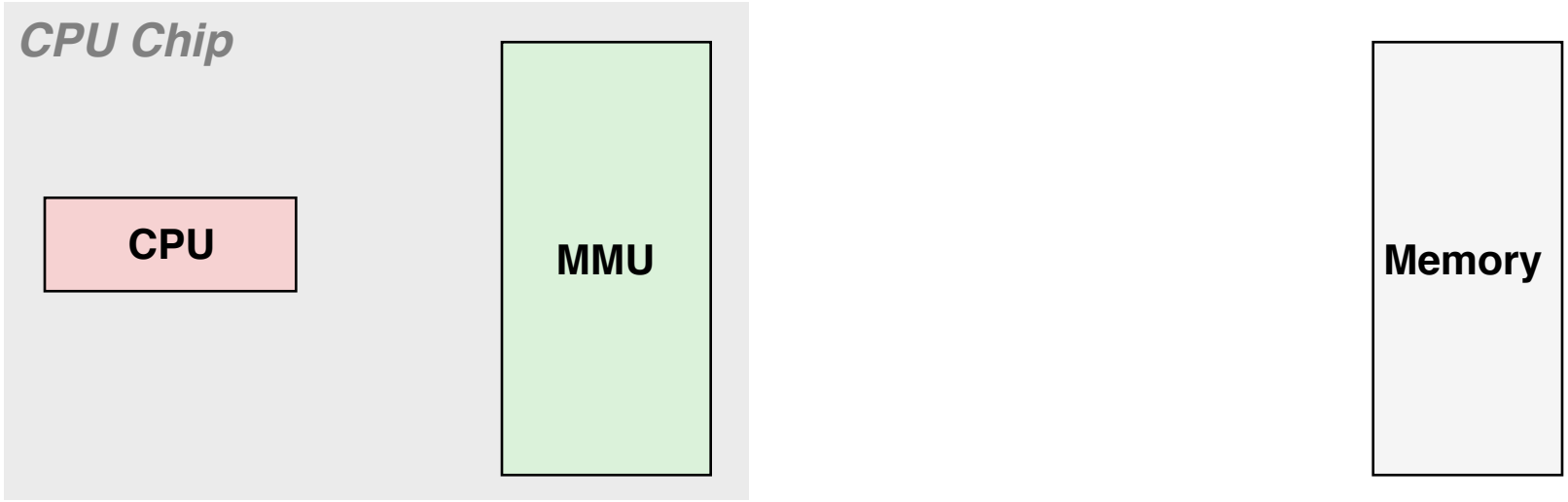
4) Valid bit is zero, so MMU triggers page fault exception

5) Handler identifies victim (and, if dirty, pages it out to disk)

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

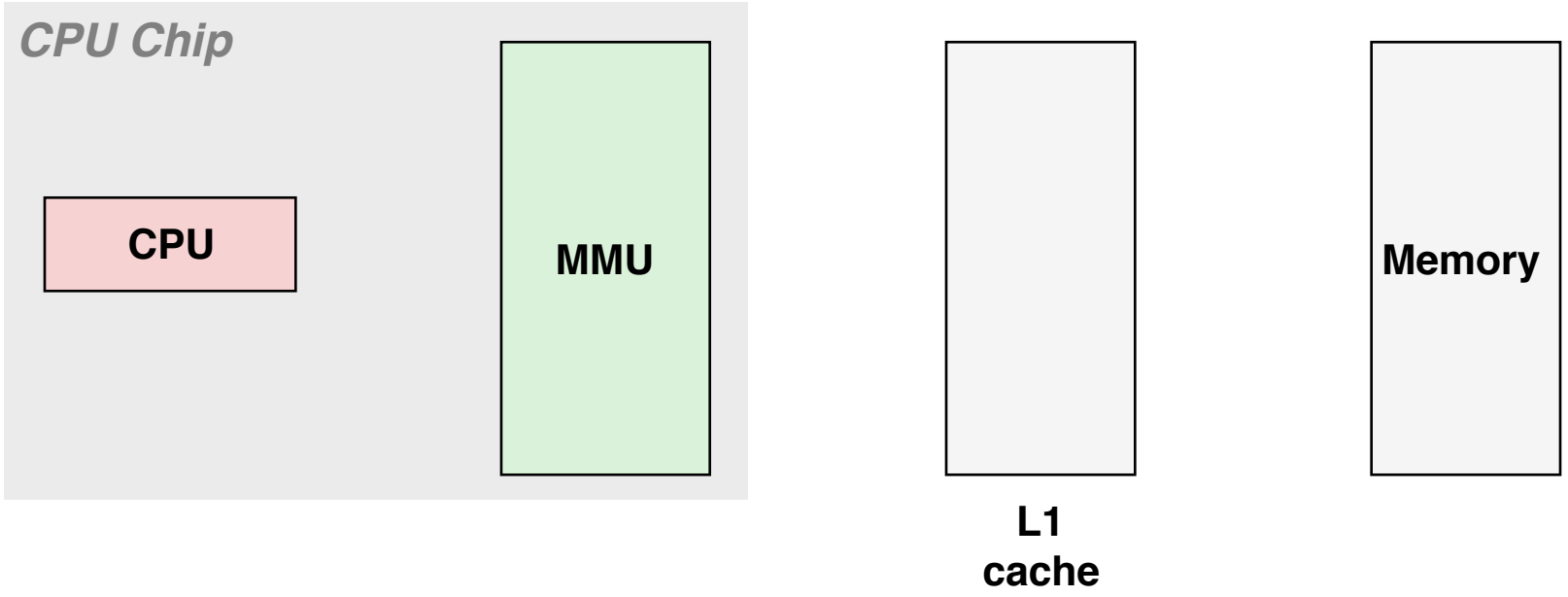
7) Handler returns to original process, restarting faulting instruction

Integrating VM and Cache



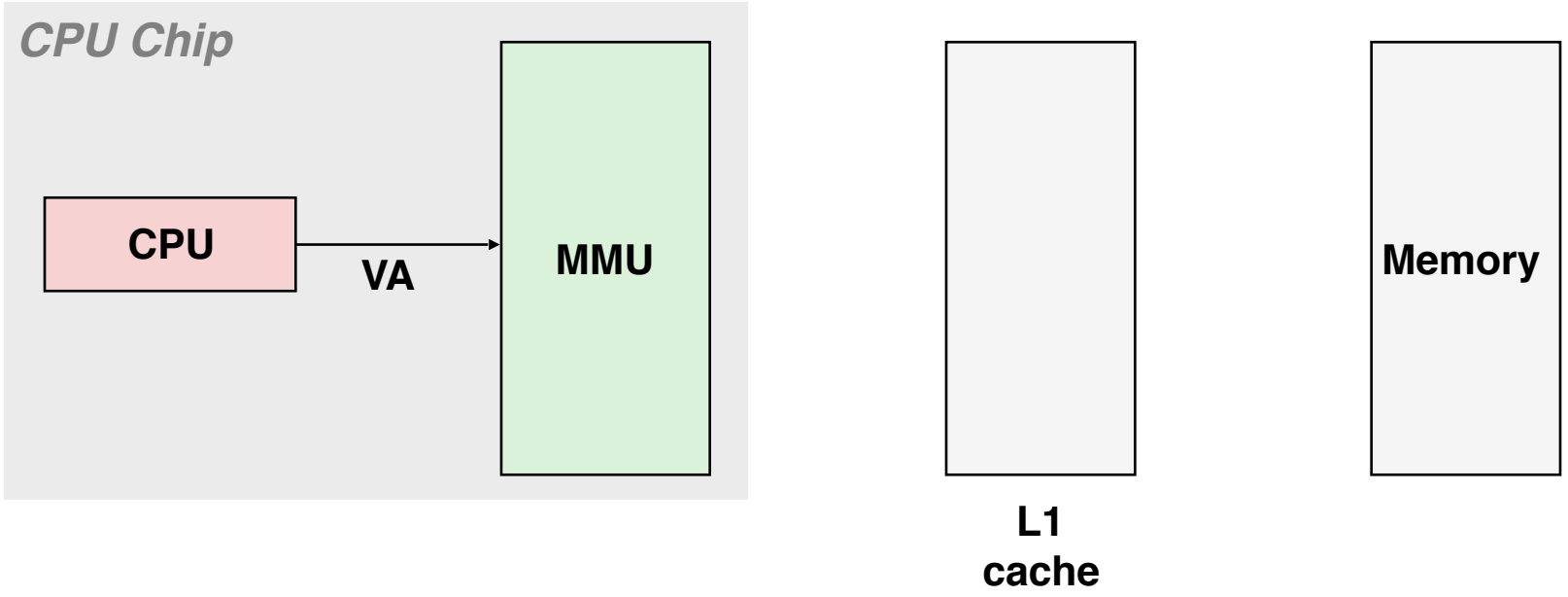
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



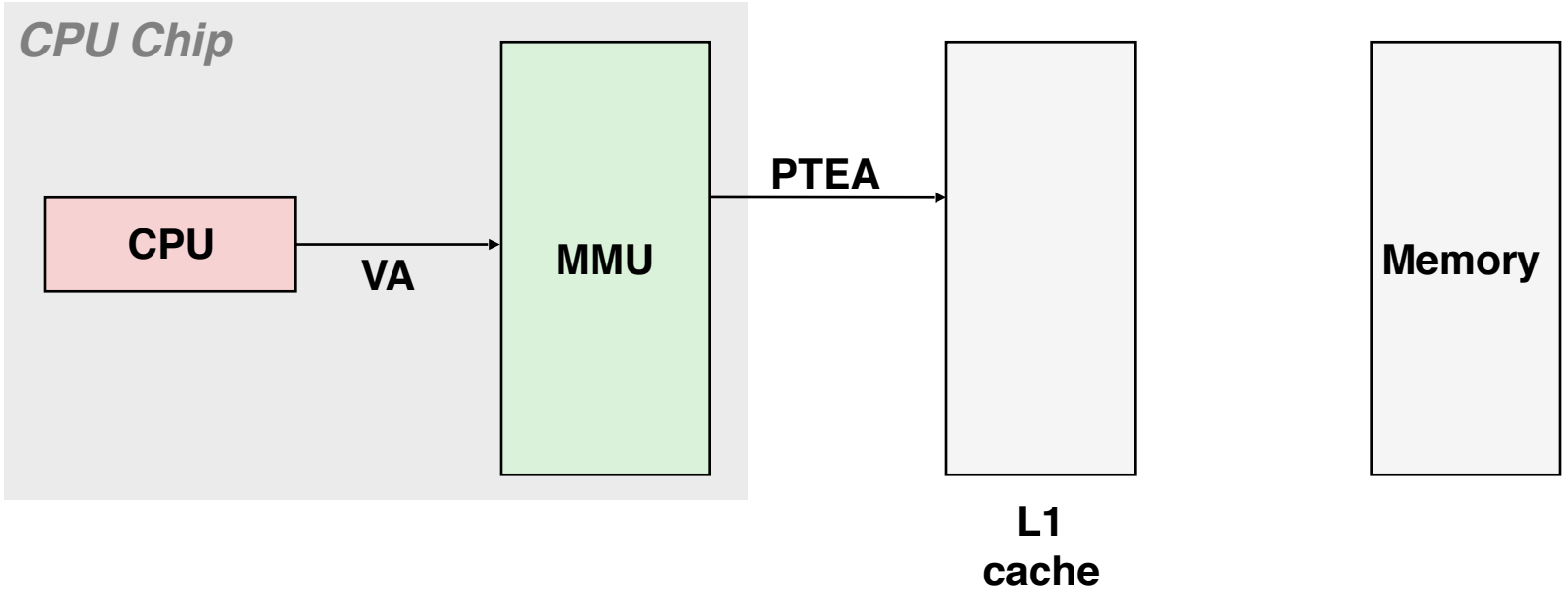
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



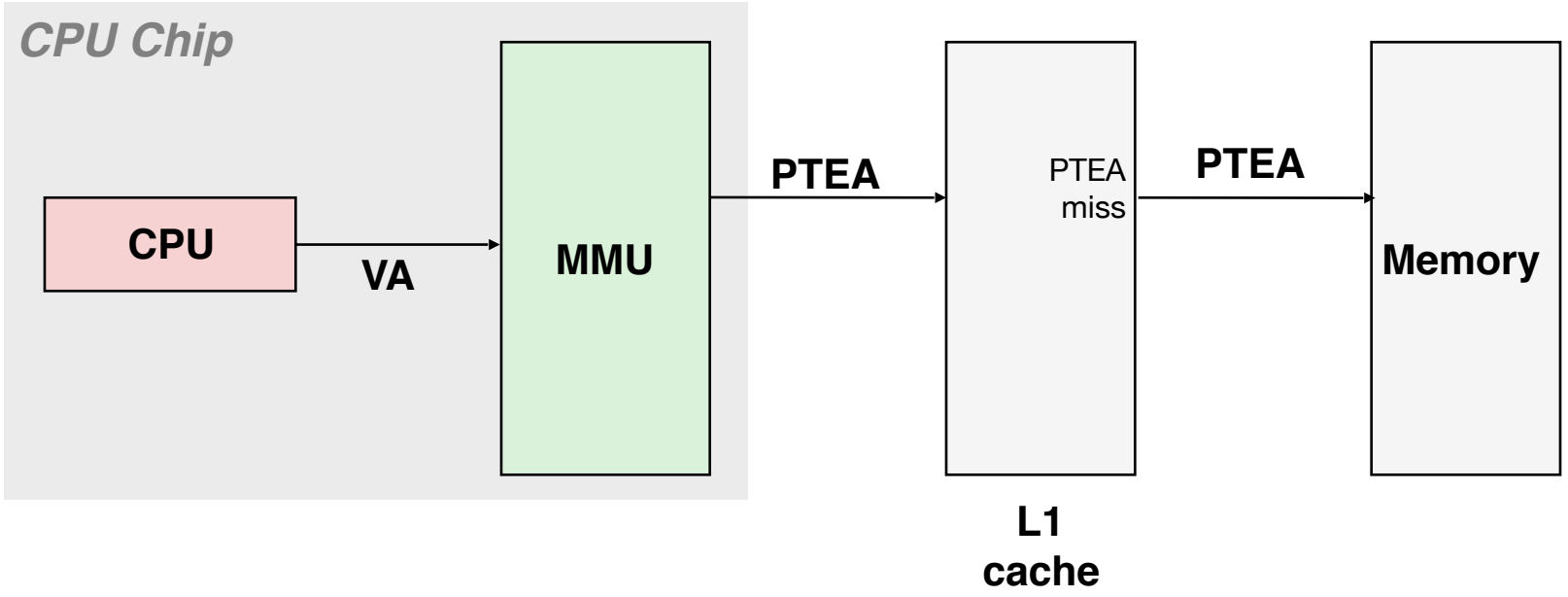
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



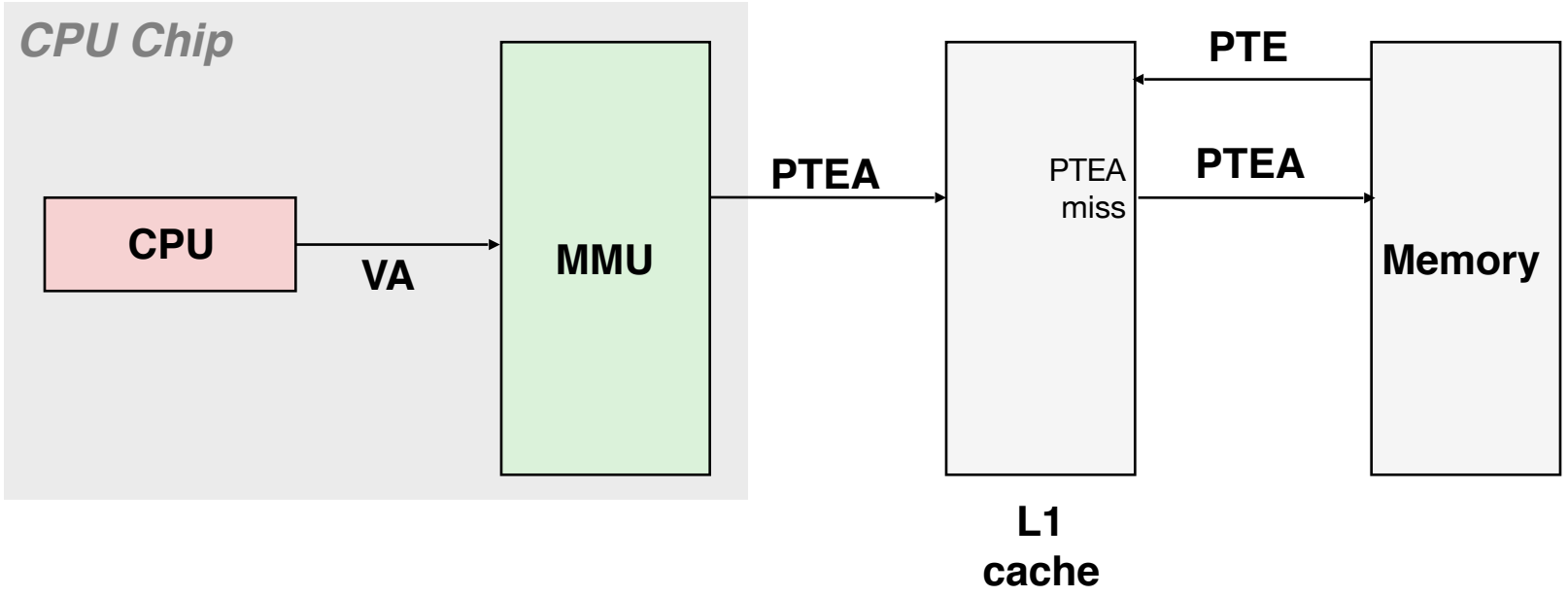
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



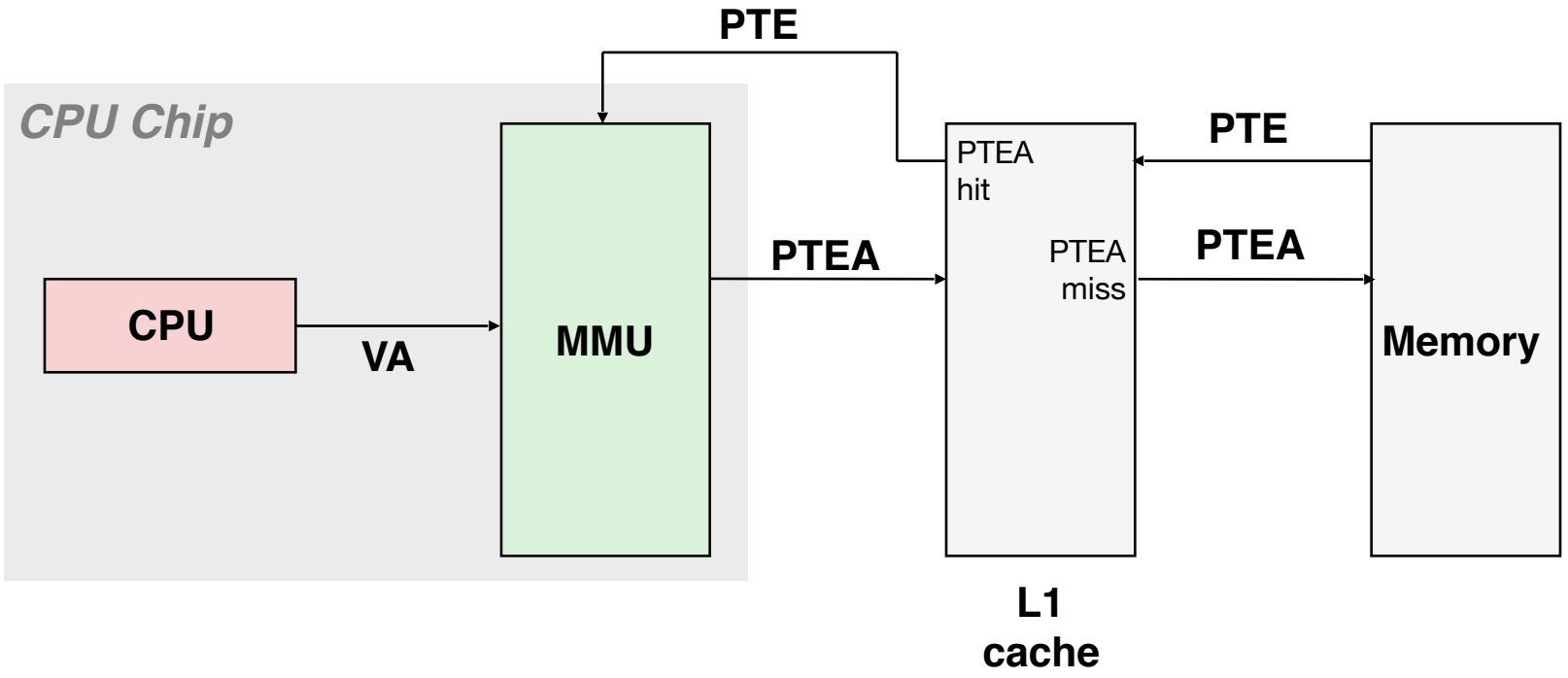
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



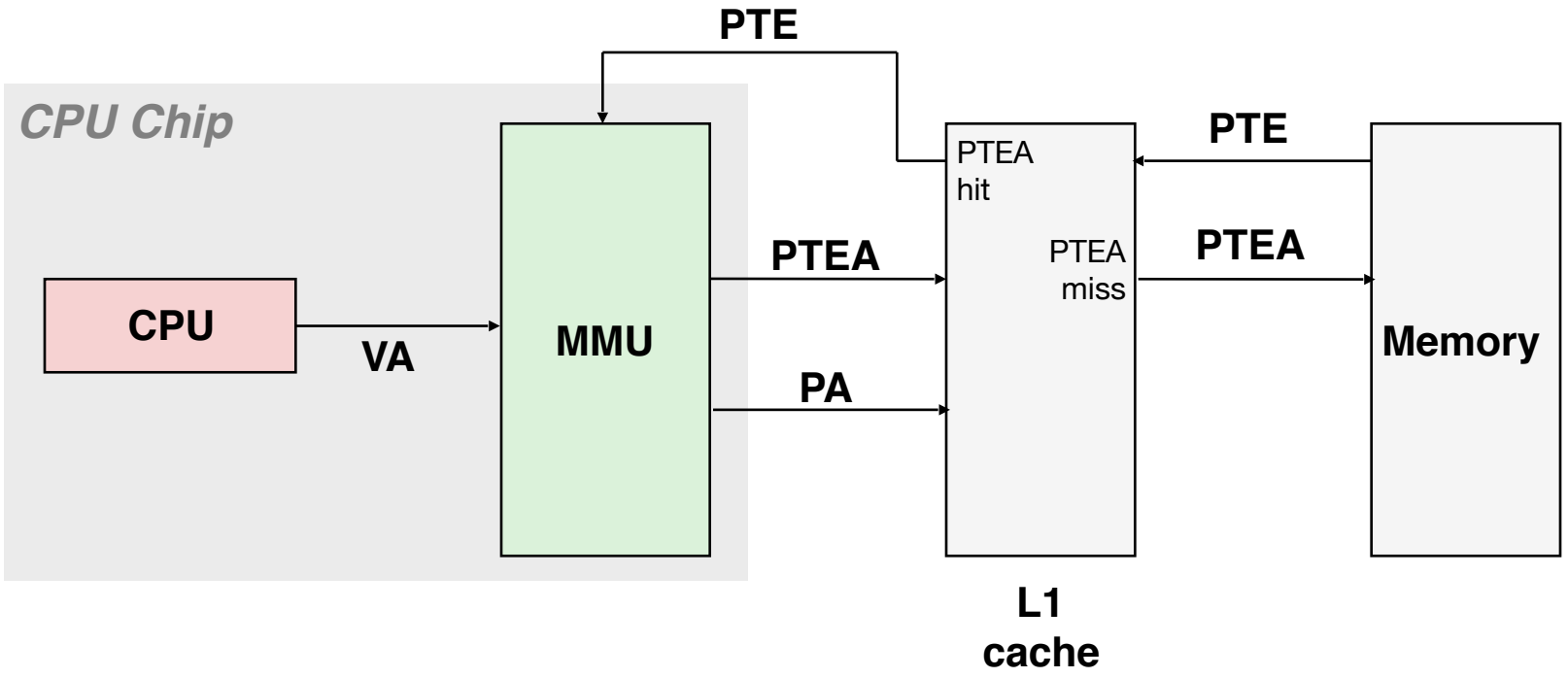
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



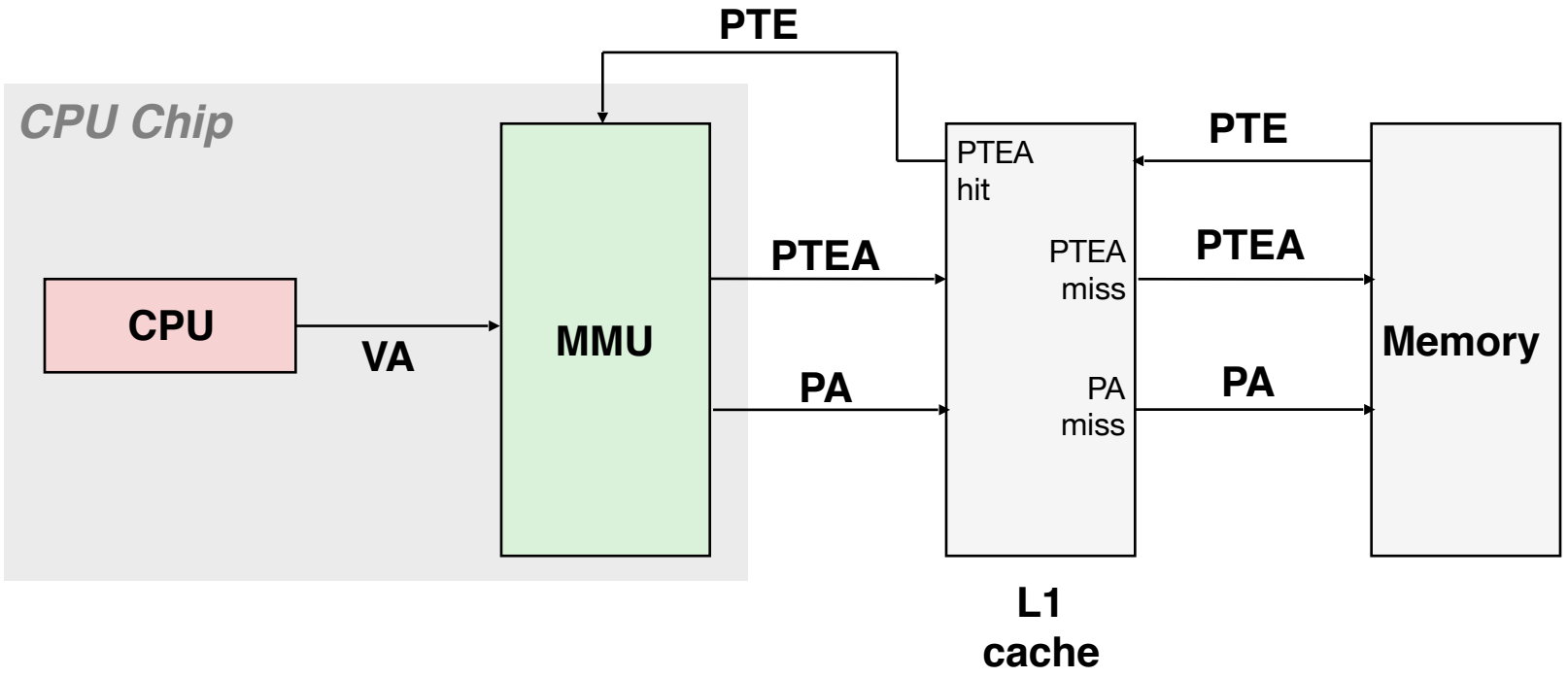
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



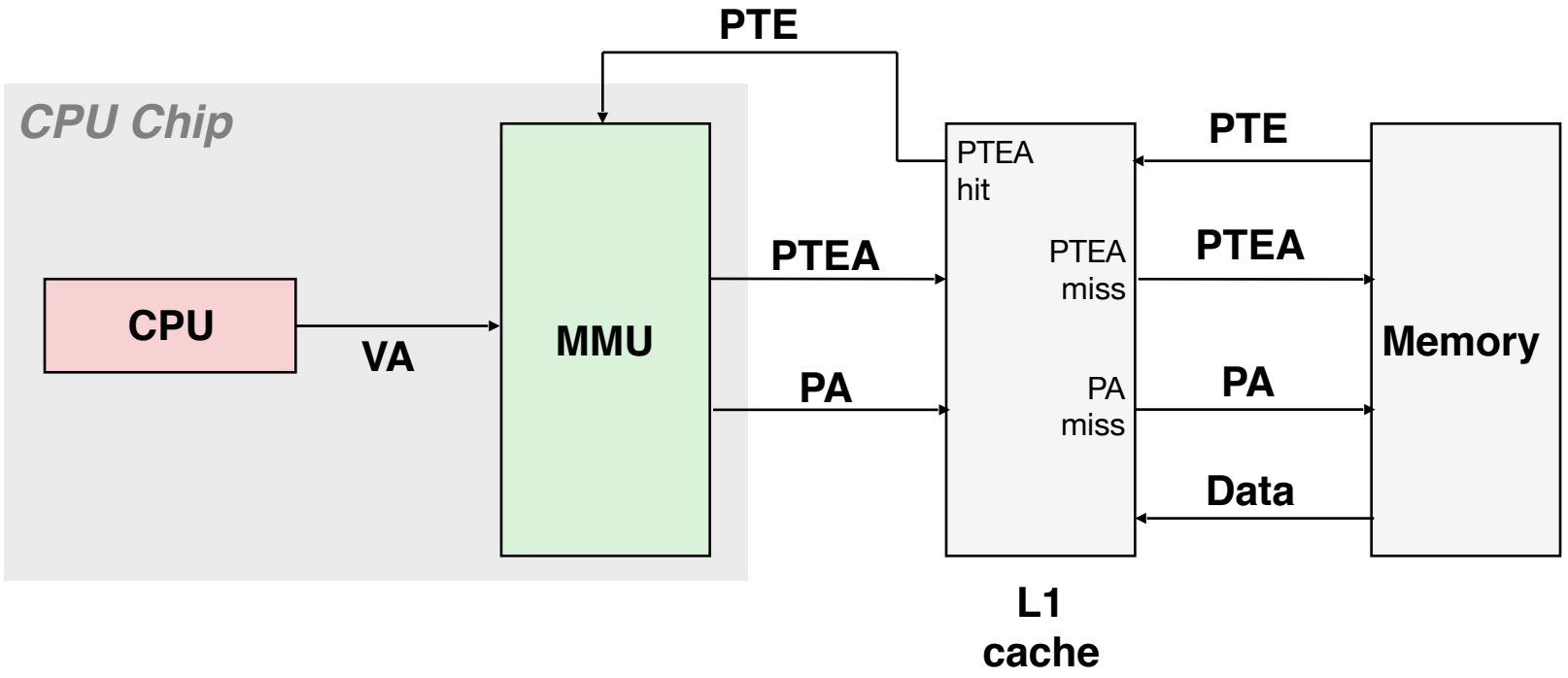
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



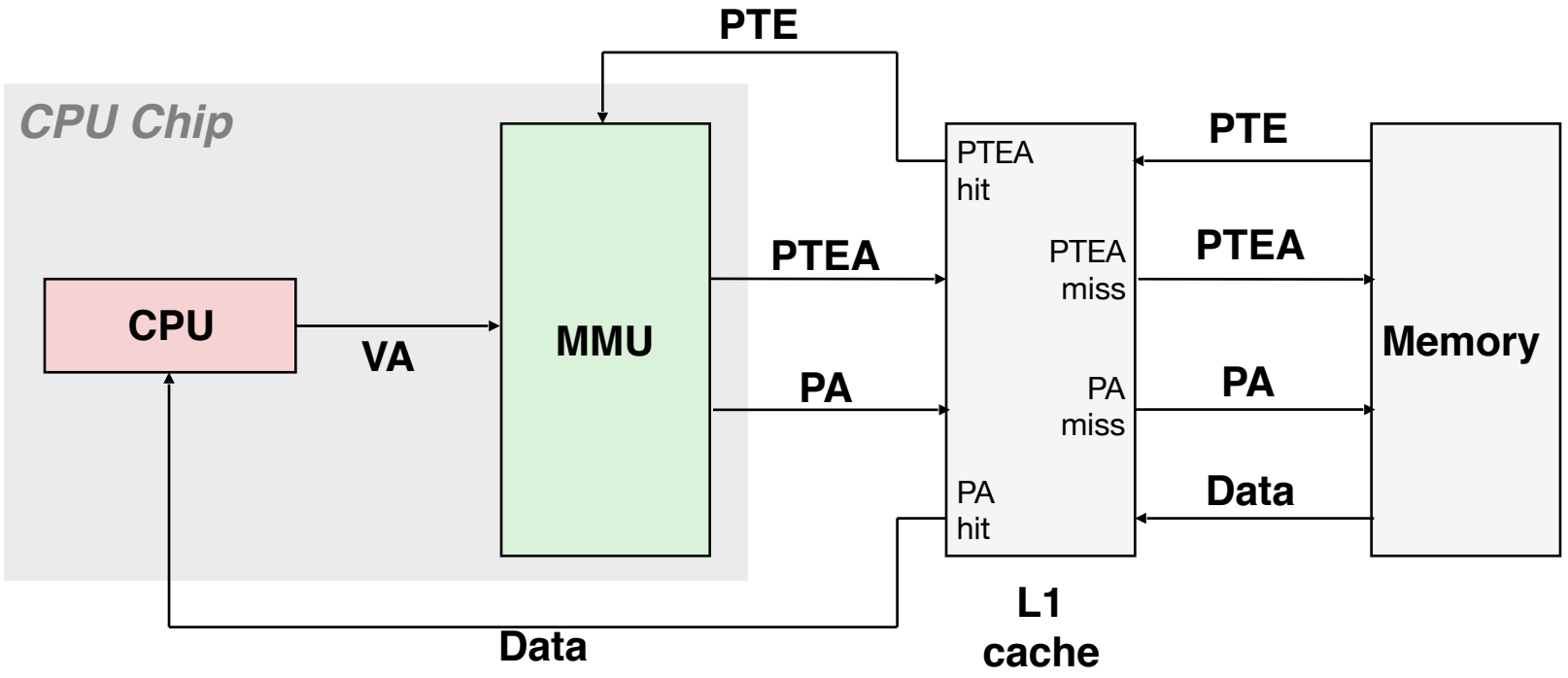
VA: virtual address, PA: physical address, PTE: page table entry, PTEA = PTE address

Integrating VM and Cache



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

Integrating VM and Cache



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