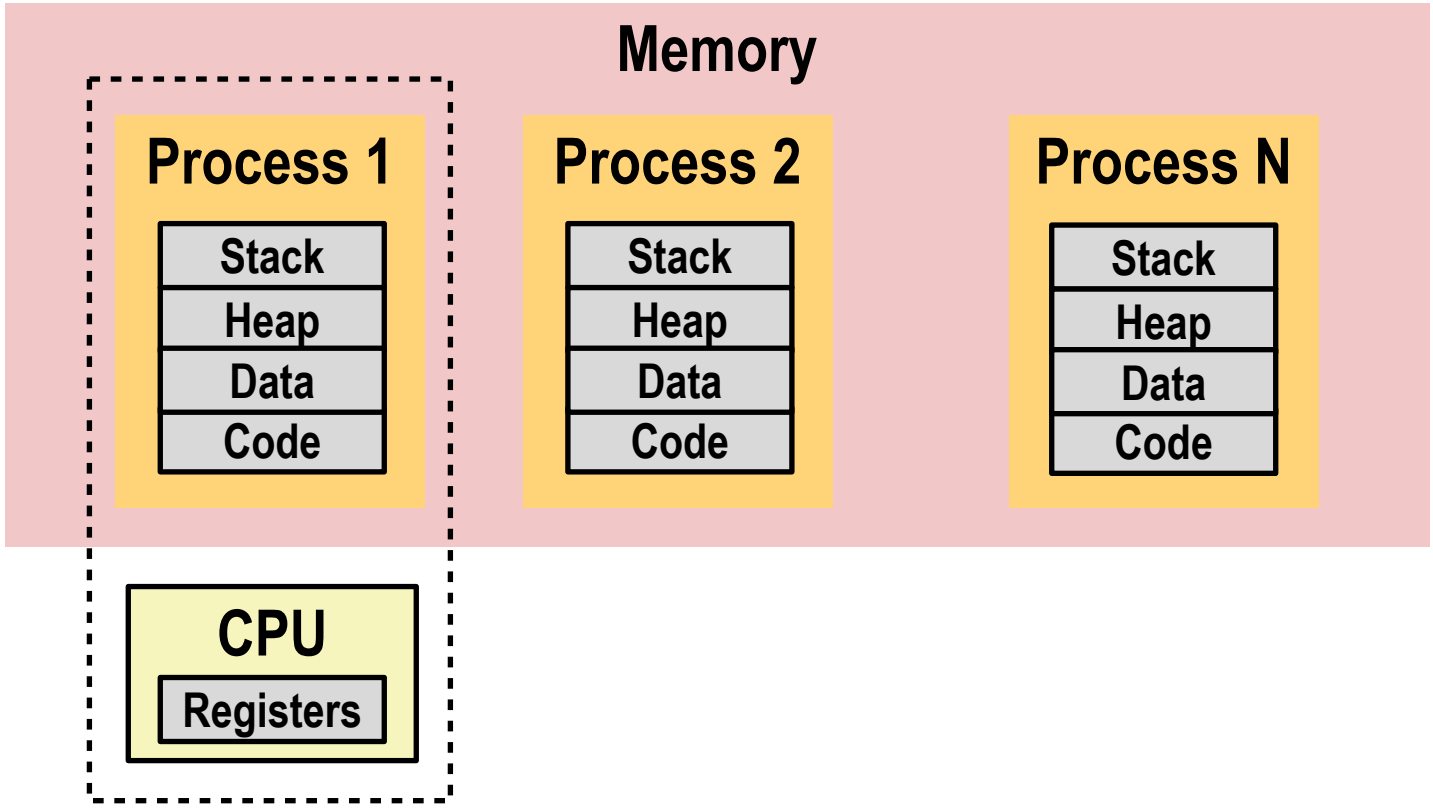
CSC 252/452: Computer Organization
Fall 2025: Lecture 20

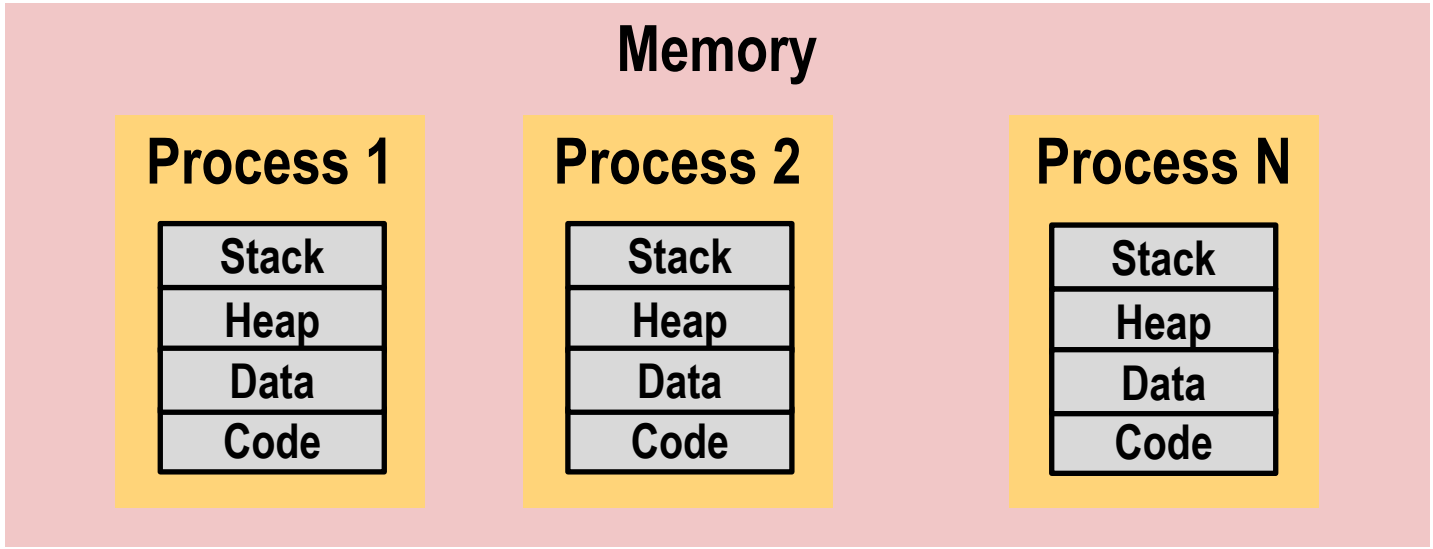
Instructor: Yanan Guo

Department of Computer Science
University of Rochester

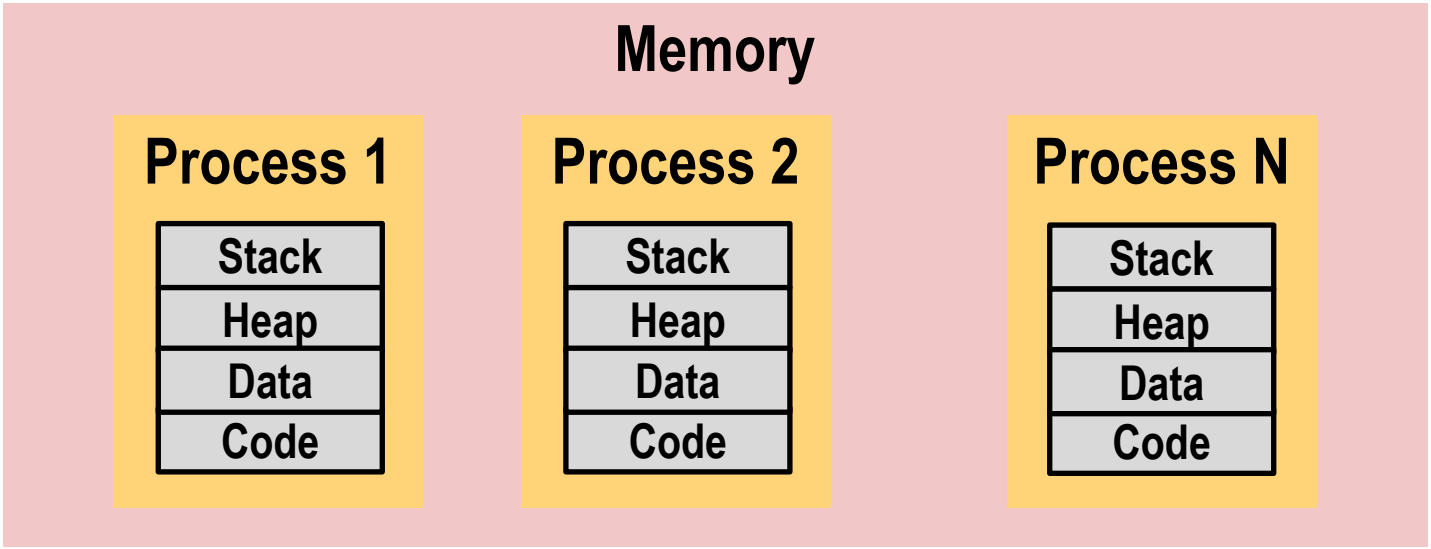
Multiprocessing Illustration



Problem 1: Space



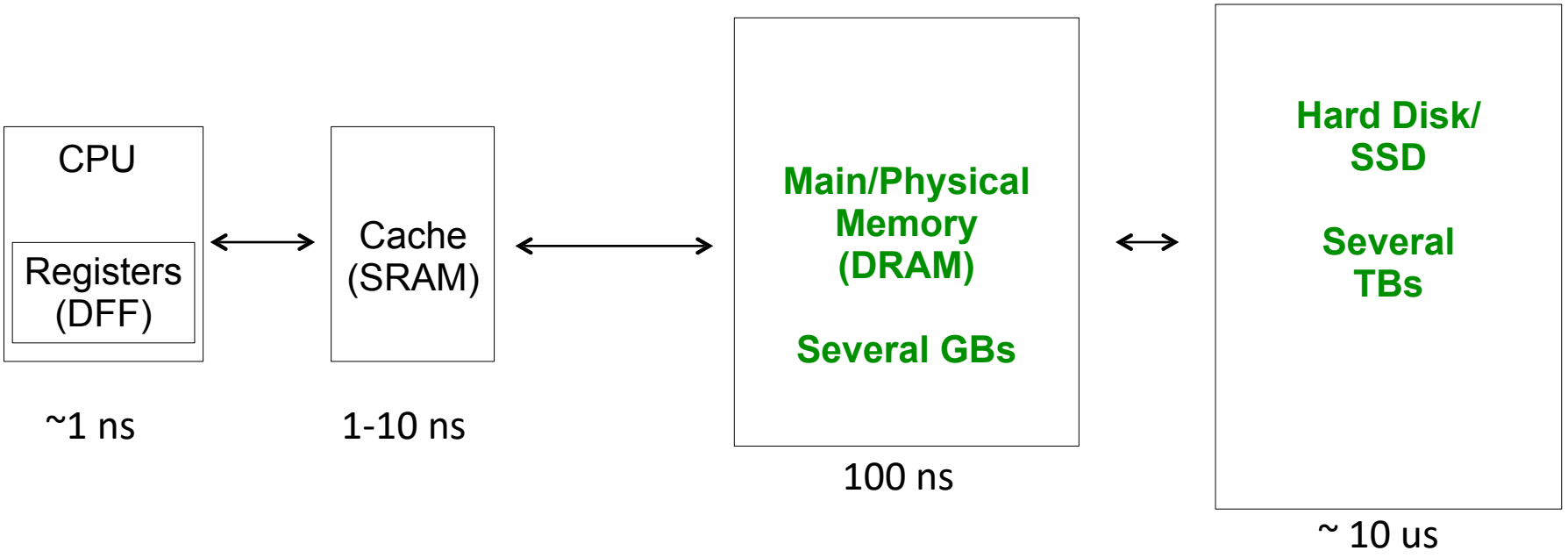
Problem 1: Space



- Space:
  - Each process's address space is huge (64-bit): can memory hold it (16GB is just 34-bit)?
  - 2^48 bytes is 256 TB
  - There are multiple processes, increasing the storage requirement further

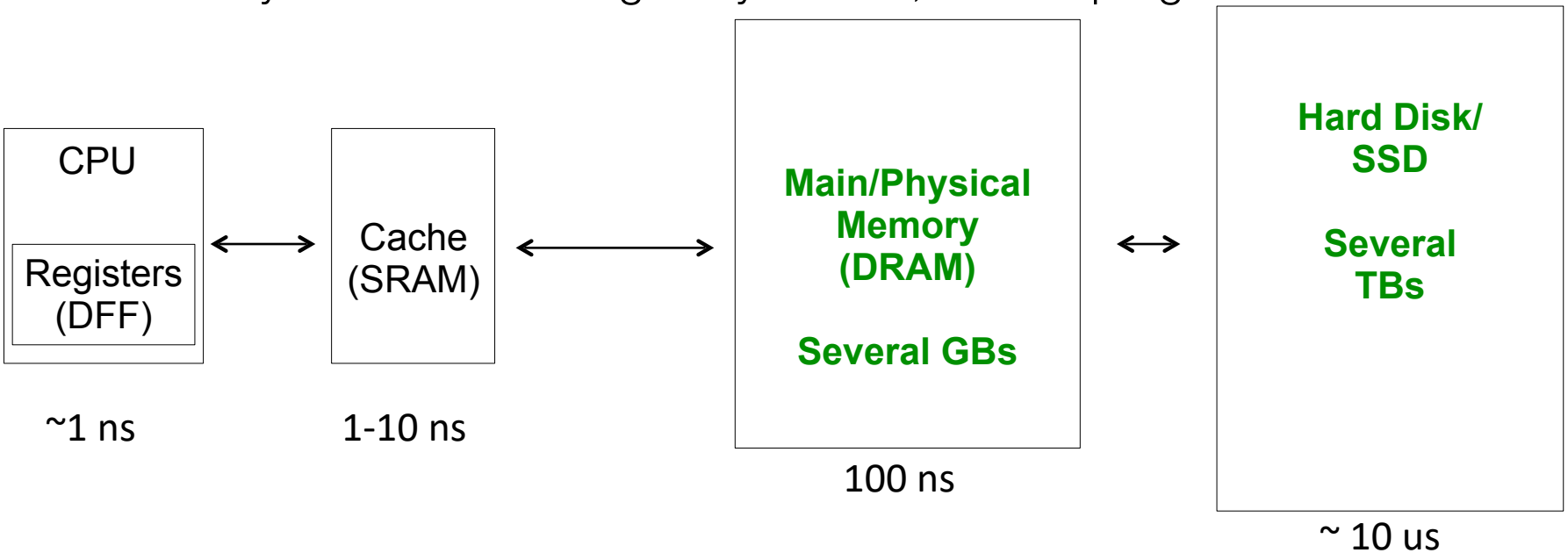
Recall: Memory Hierarchy

- Solution: store all the data in disk (several TBs typically), and use memory only for most recently used data
  - Of course if a process uses all its address space that won't be enough, but usually a process won't use all 64 bits. So it's OK.



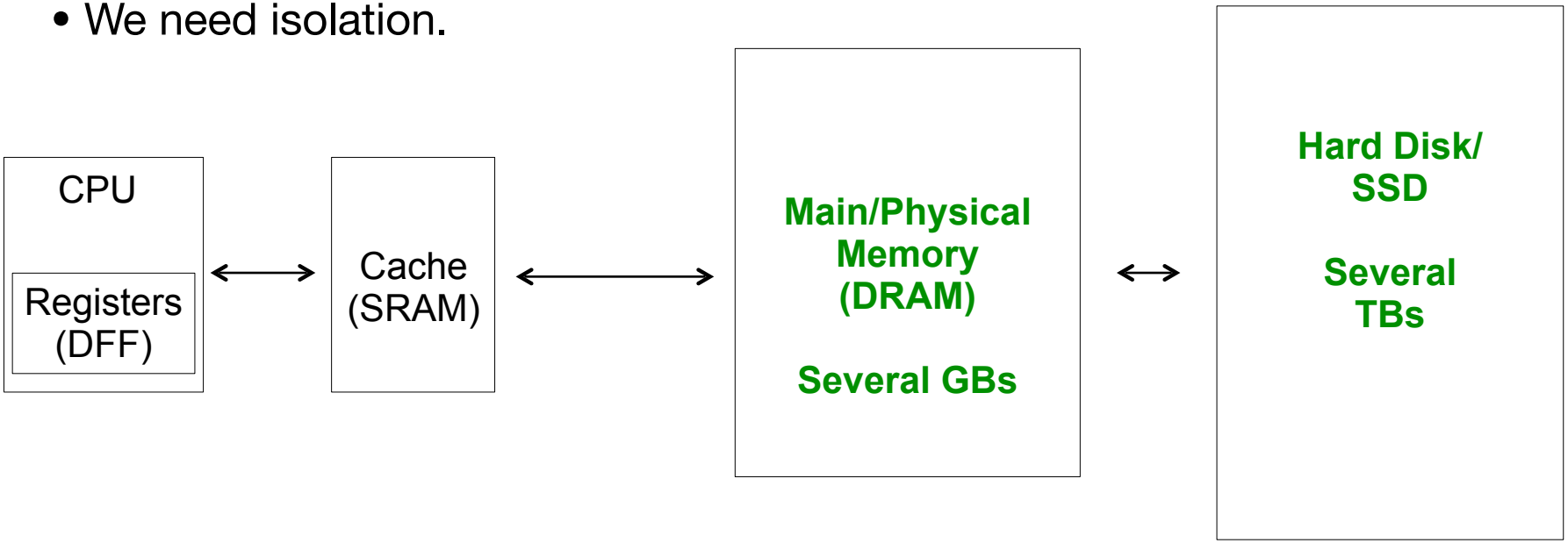
Recall: Memory Hierarchy

- Solution: store all the data in disk (several TBs typically), and use memory only for most recently used data
  - Of course if a process uses all its address space that won't be enough, but usually a process won't use all 64 bits. So it's OK.
- Challange: who is moving data back and forth between the DRAM/main memory/physical memory and the disk?
  - Ideally should be managed by the OS, not the programmer.



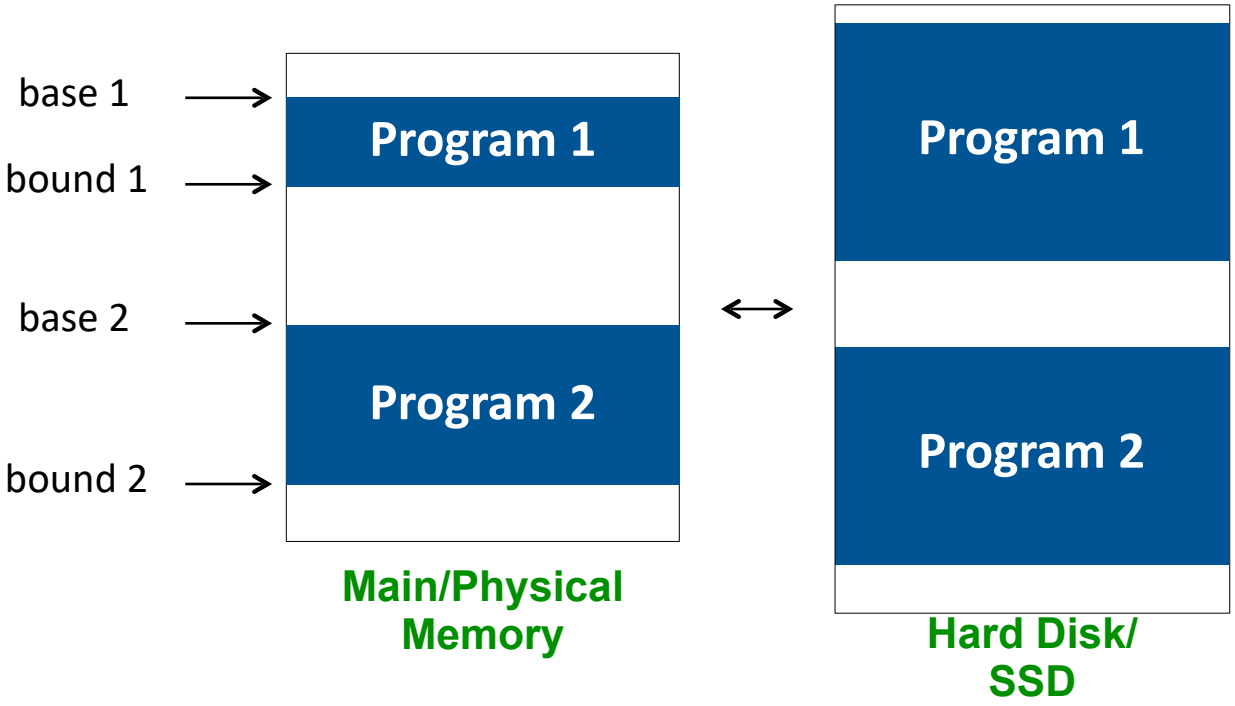
Problem 2: Security

- Different programs/processes will share the same physical memory
  - Or even different uses. A CSUG machine is accessed by all students, but there is one single physical memory!
- What if a malicious program steals/modifies data from your program?
  - If the malicious program get the address of the memory that stores your password, should it be able to access it? If not, how to prevent it?
- We need isolation.



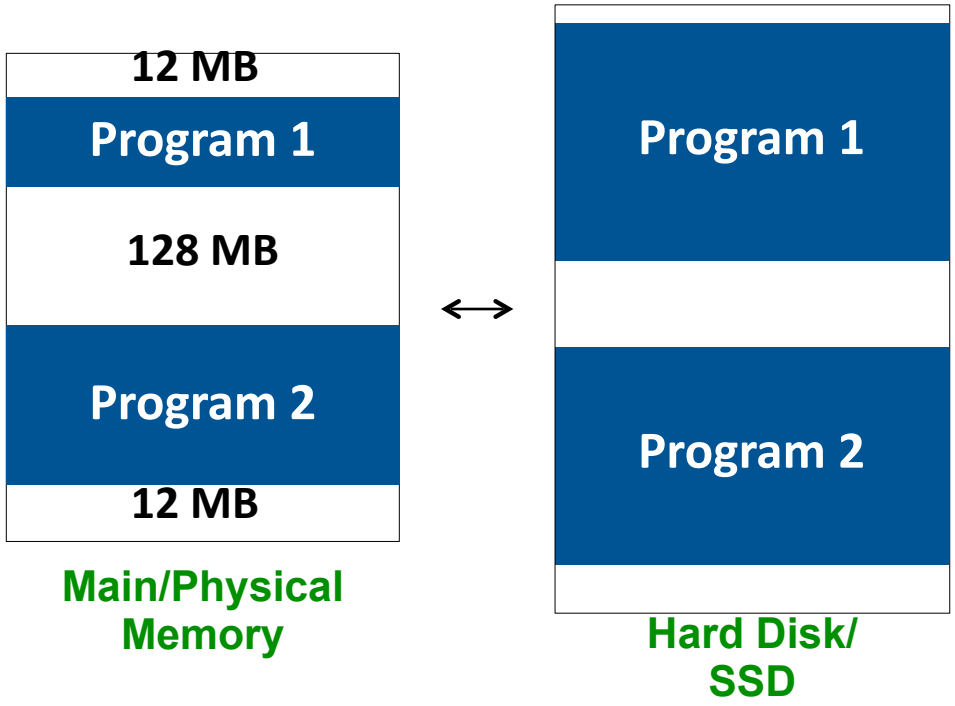
One Way to Isolate: Segments

- Different processes will have exclusive access to just one part of the physical memory.
- This is called ***Segments***.
  - Need a base register and a bound register for each process. Not widely used today. x86 still supports it (backward compatibility!)
  - Fast but not inflexible. Makes benign sharing hard.



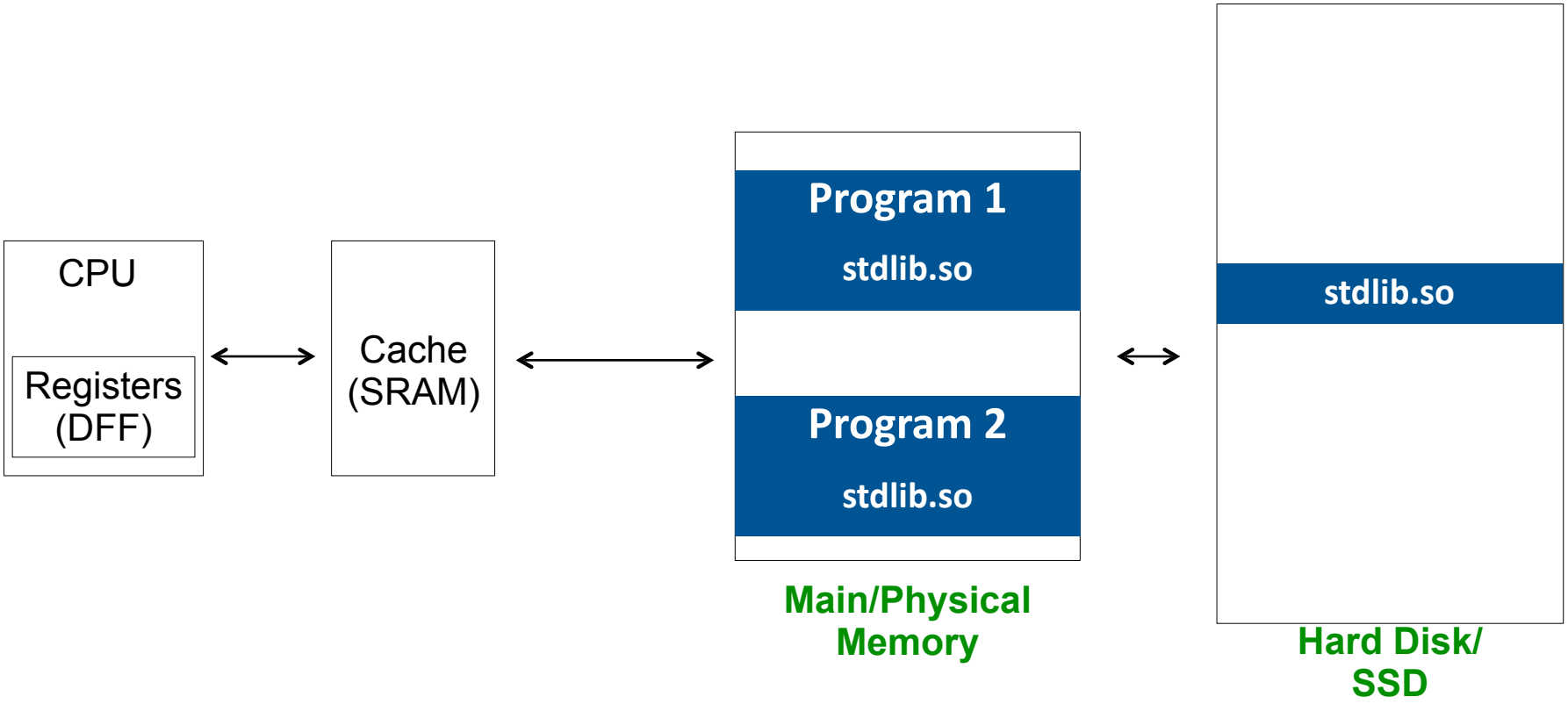
Problem 3: Fragmentation (with Segments)

- Each process gets a continuous chunk of memory. Inflexible.
- What if a process requests more space than any continuous chunk in memory but smaller than the total free memory?
  - This is called “fragmentation”; will talk about this more later.
- Need to allow assigning discontinuous chunks of memory to processes.

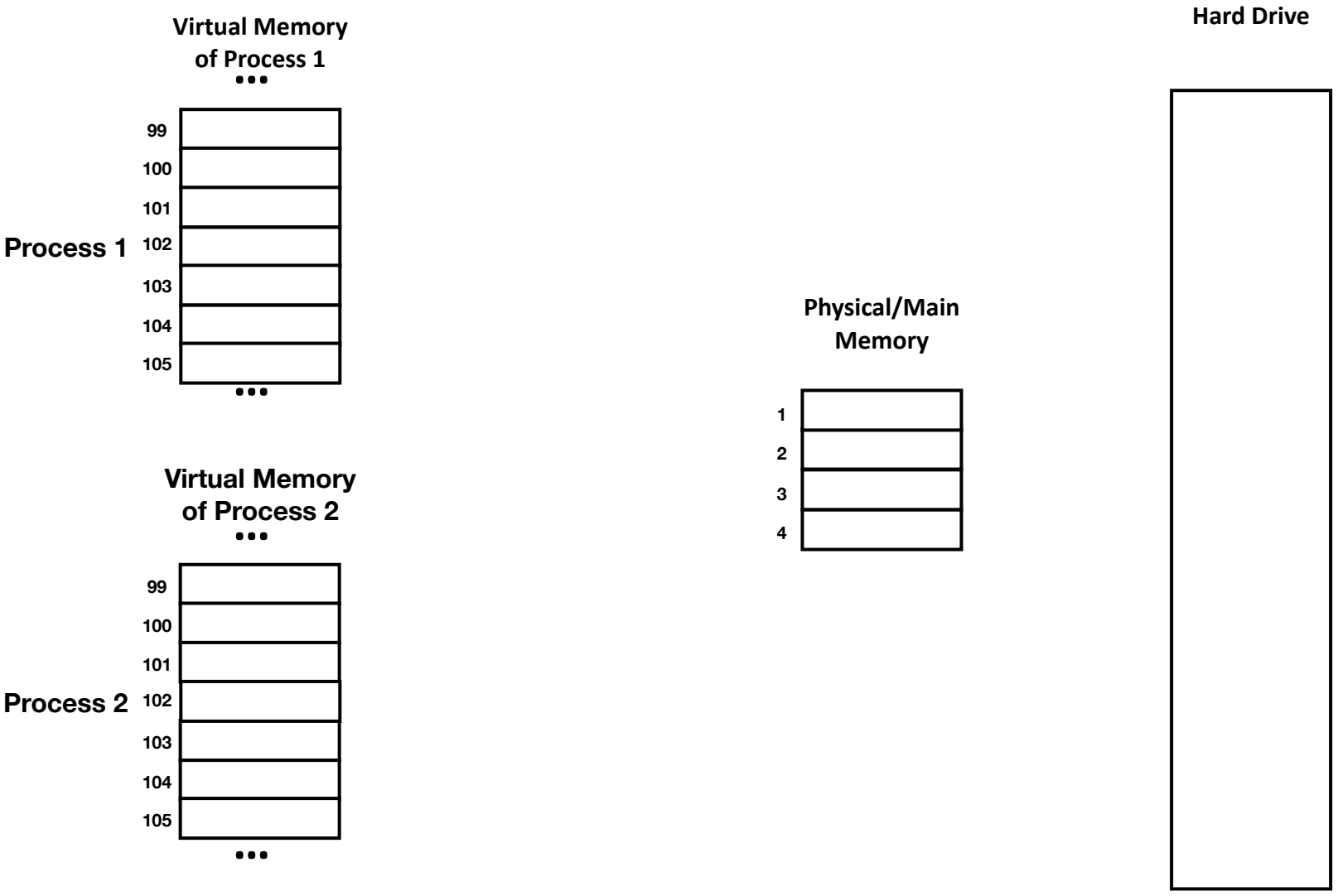


Problem 4: Benign Sharing (with Segments)

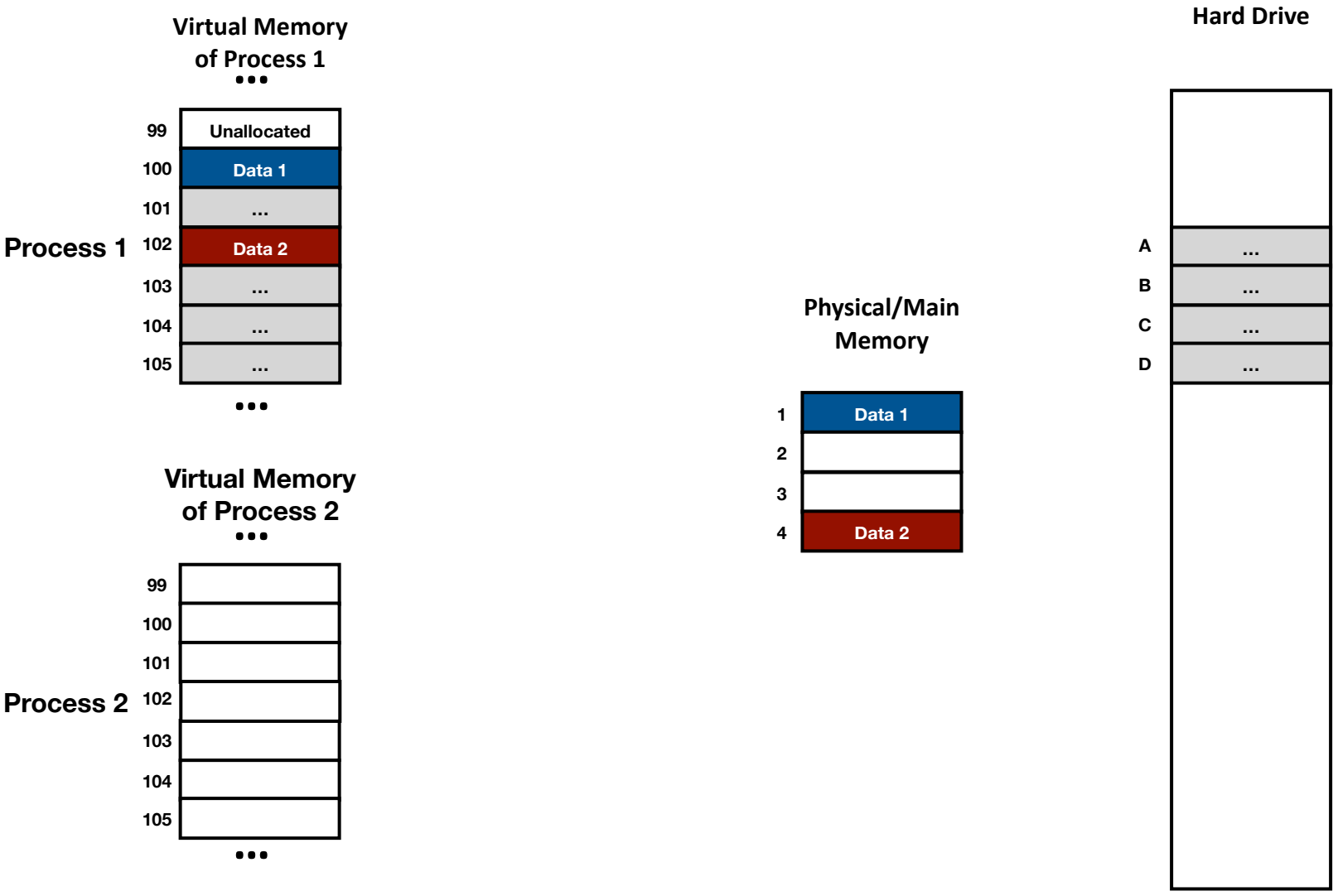
- Different programs/processes will share same data: files, libraries, etc.
- No need to have separate copies in the physical memory.
- Would be good to let other processes access part of the current's process' memory based on the "permission".



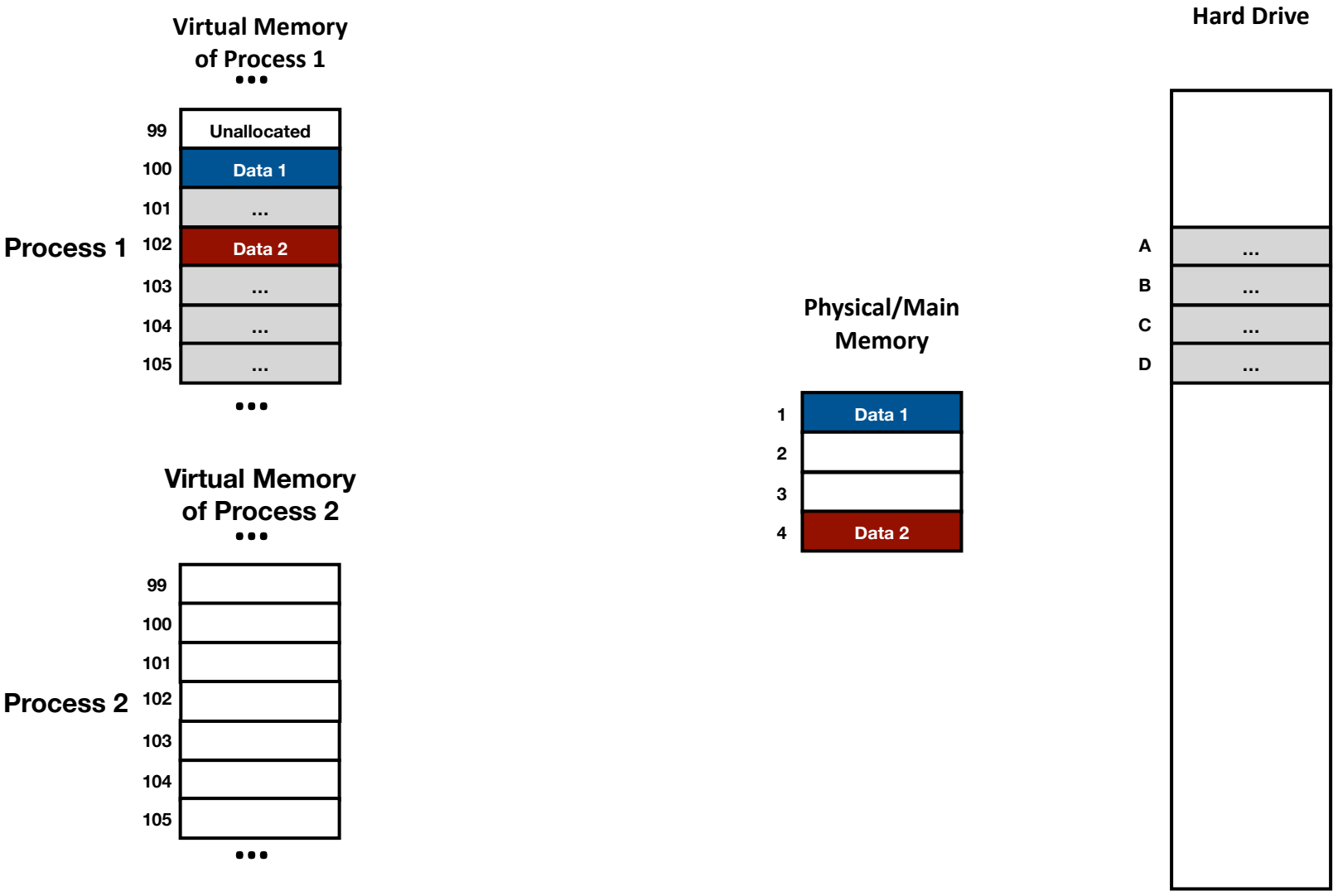
The Big Idea: Virtual Memory



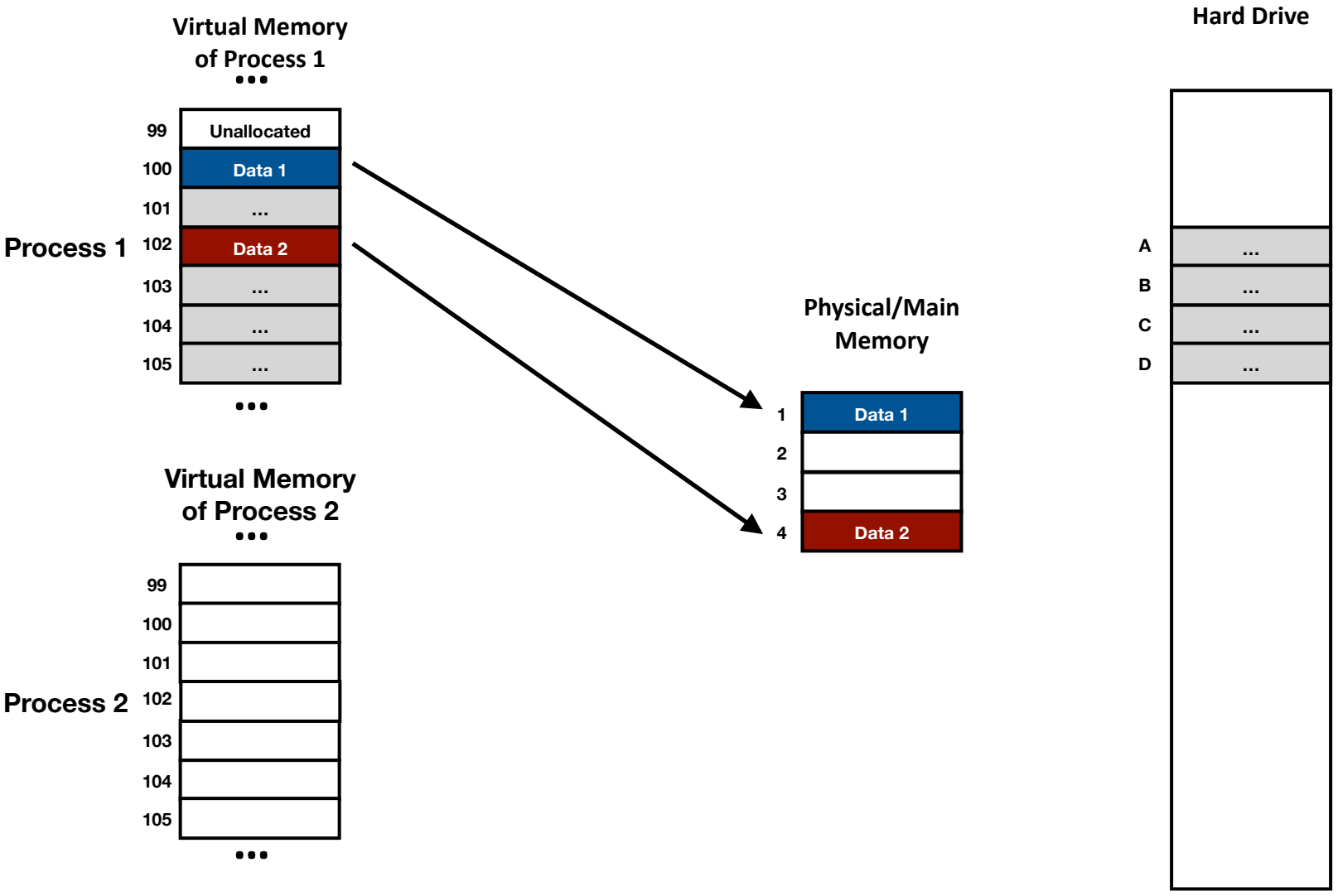
“Cache” Data in Physical Memory



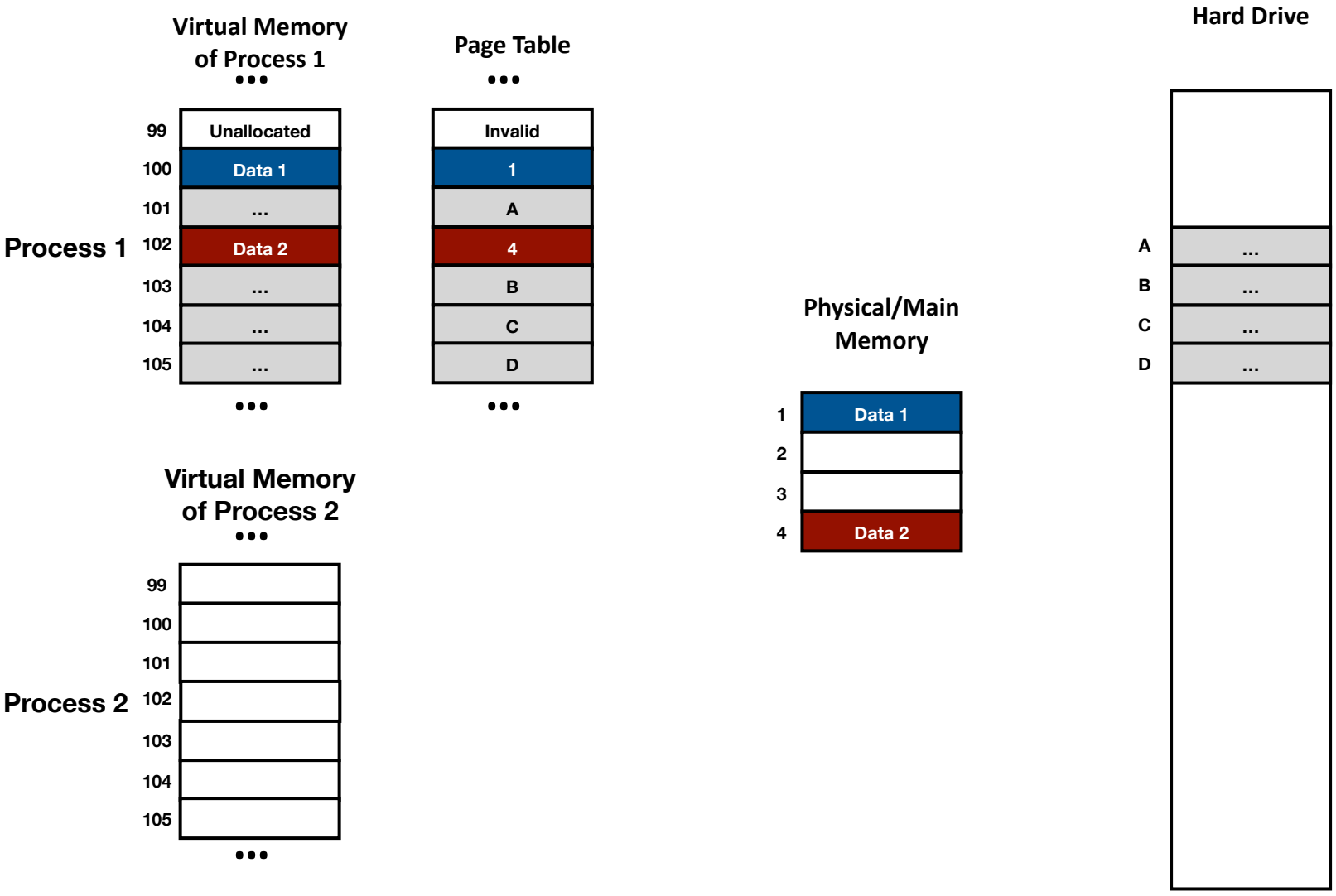
Allow Using Discontinuous Allocation



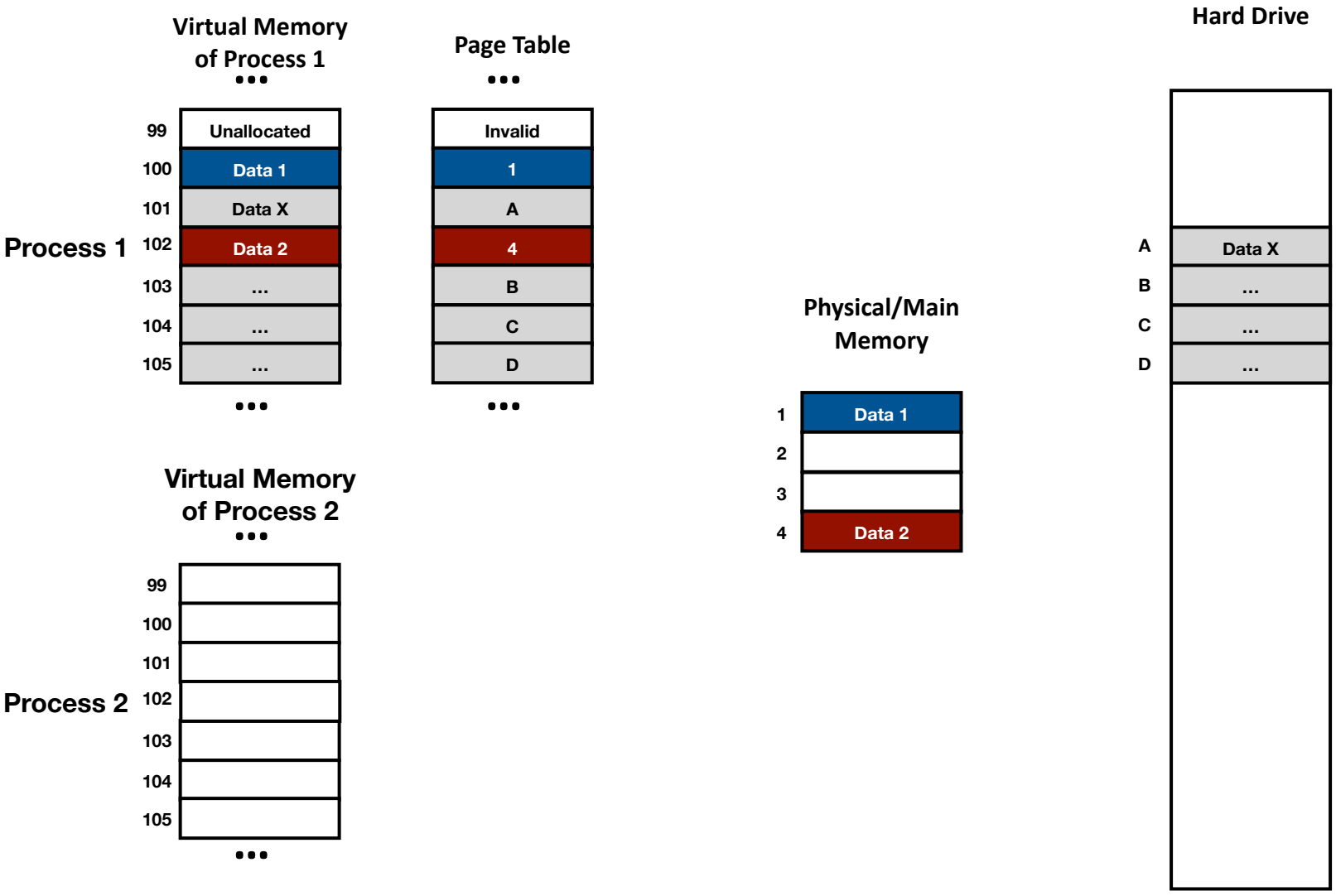
Allow Using Discontinuous Allocation



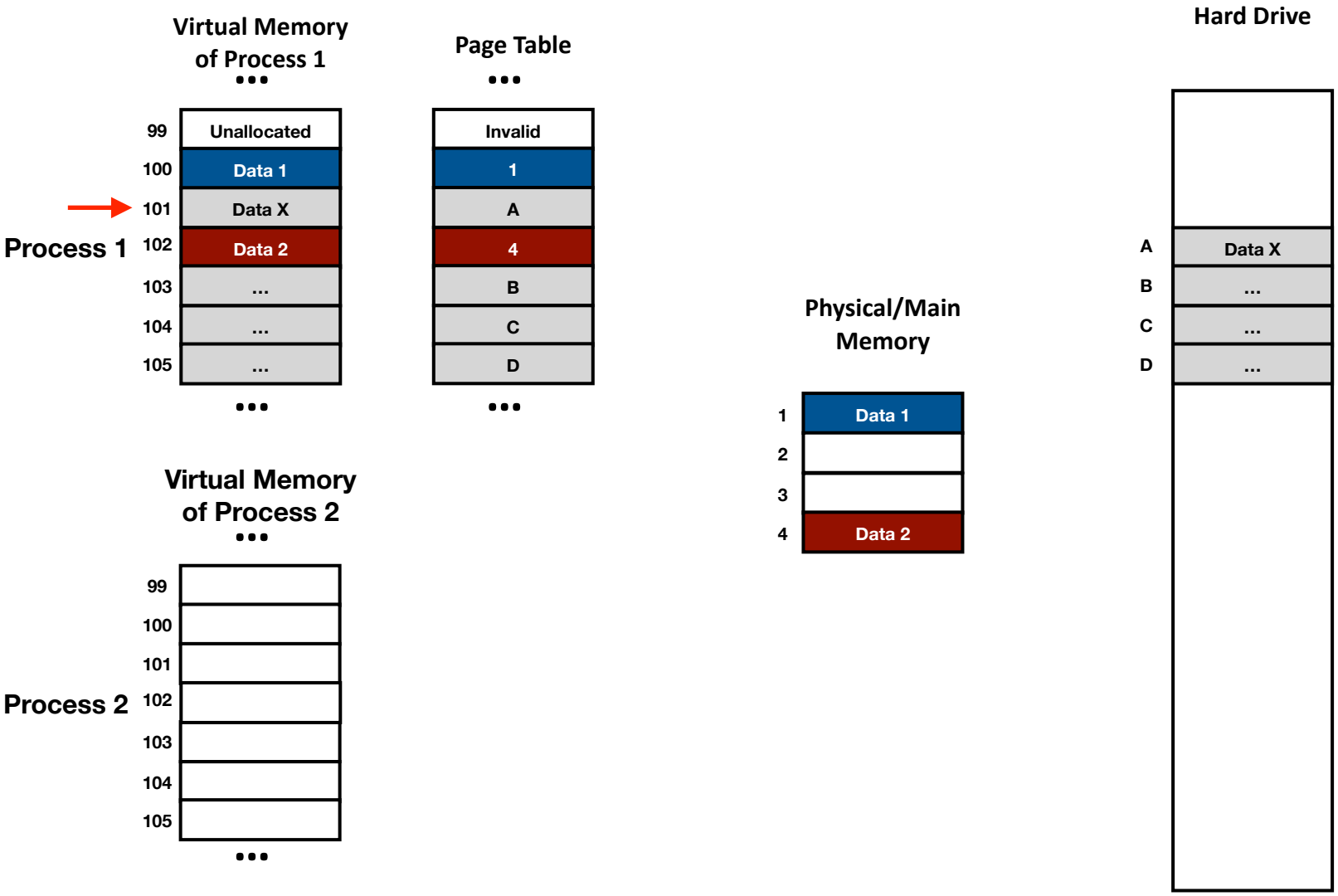
Page Table


Virtual Memory of Process 1
Page Table
Process 1
99
100
101
102
103
104
105
Unallocated
Data 1
...
Data 2
...
...
...
Invalid
1
A
4
B
C
D
Physical/Main Memory
1
2
3
4
Data 1
Data 2
Hard Drive
A
B
C
D
...
...
...
...
Virtual Memory of Process 2
Process 2
99
100
101
102
103
104
105

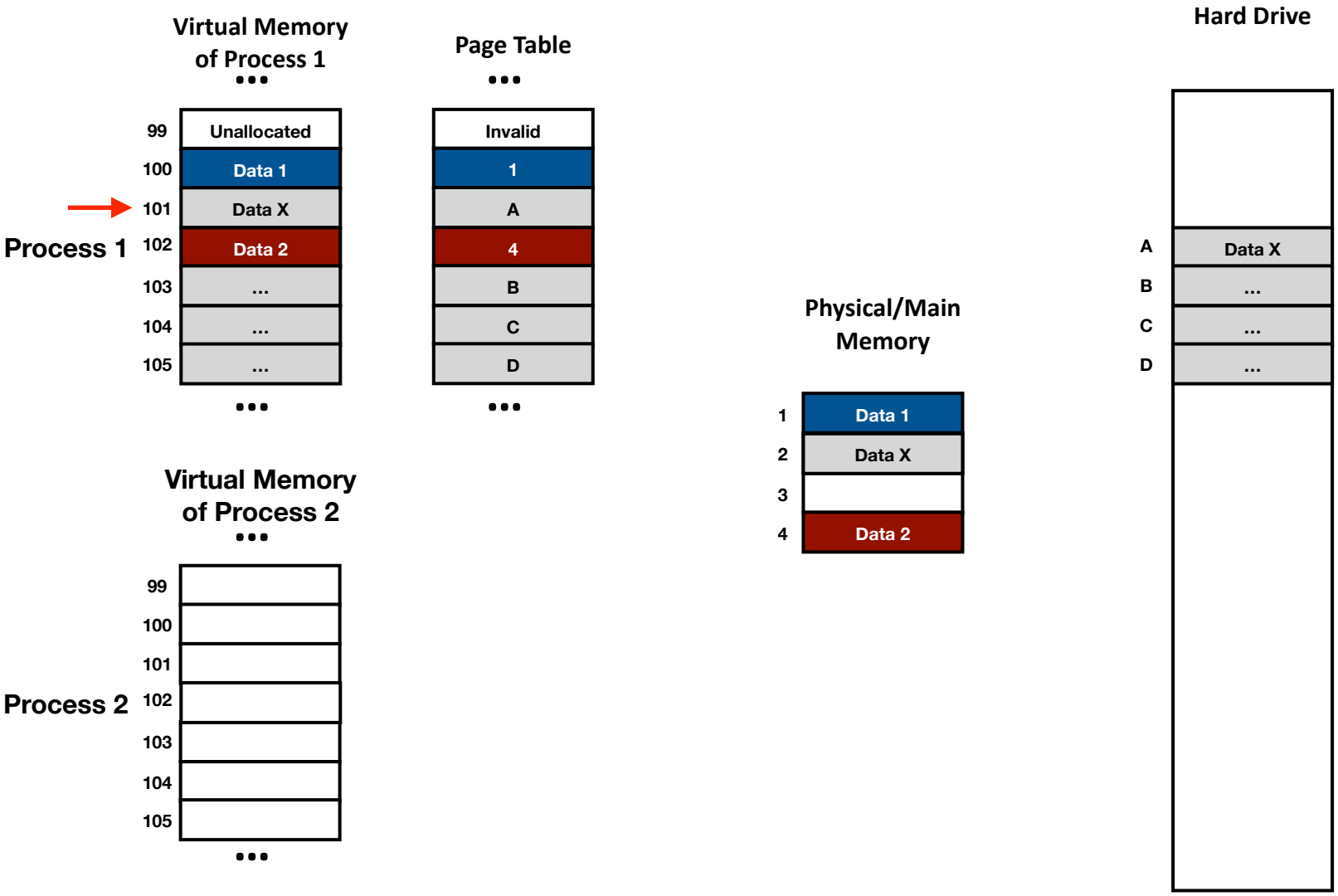
Demand Paging



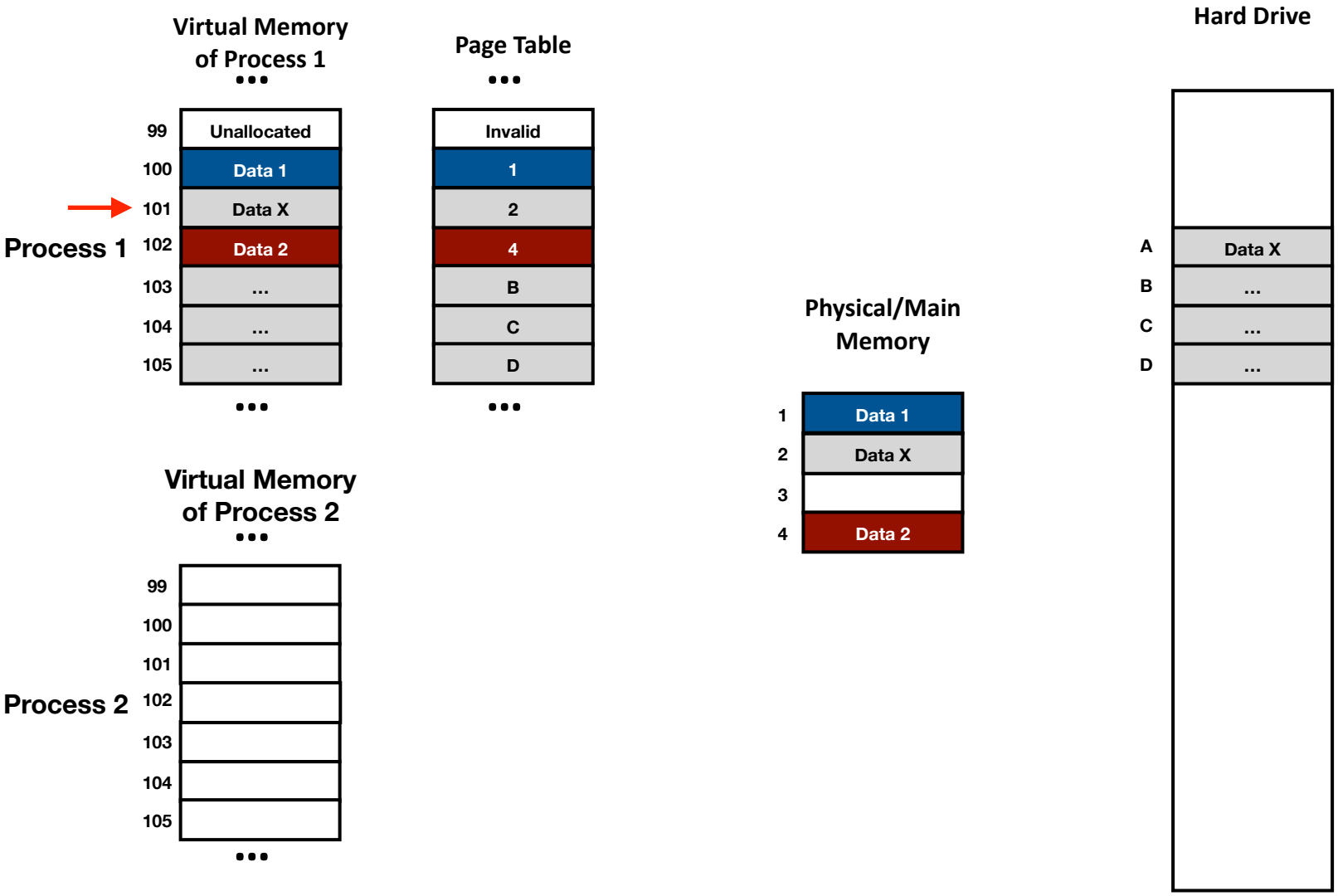
Demand Paging



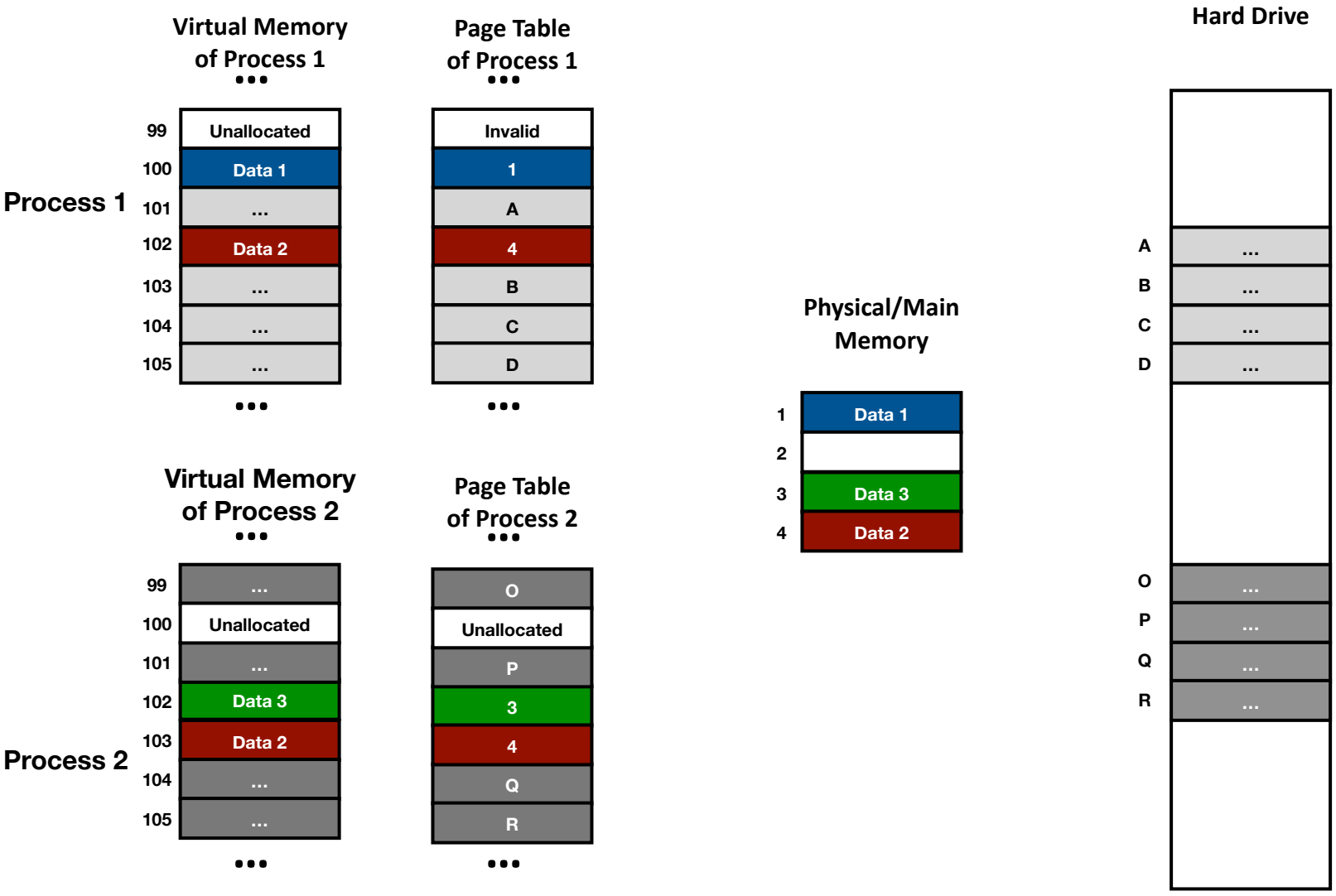
Demand Paging



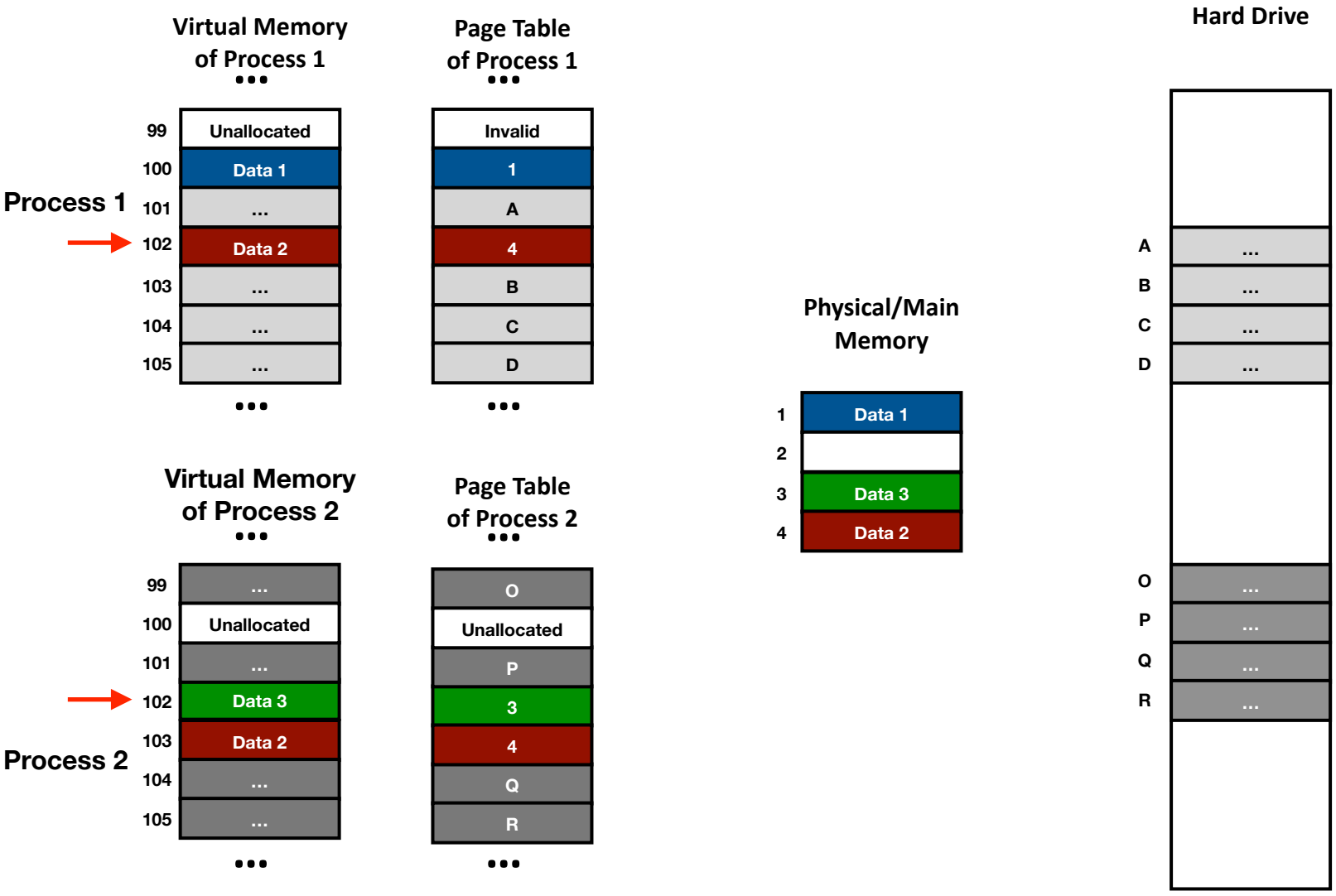
Demand Paging



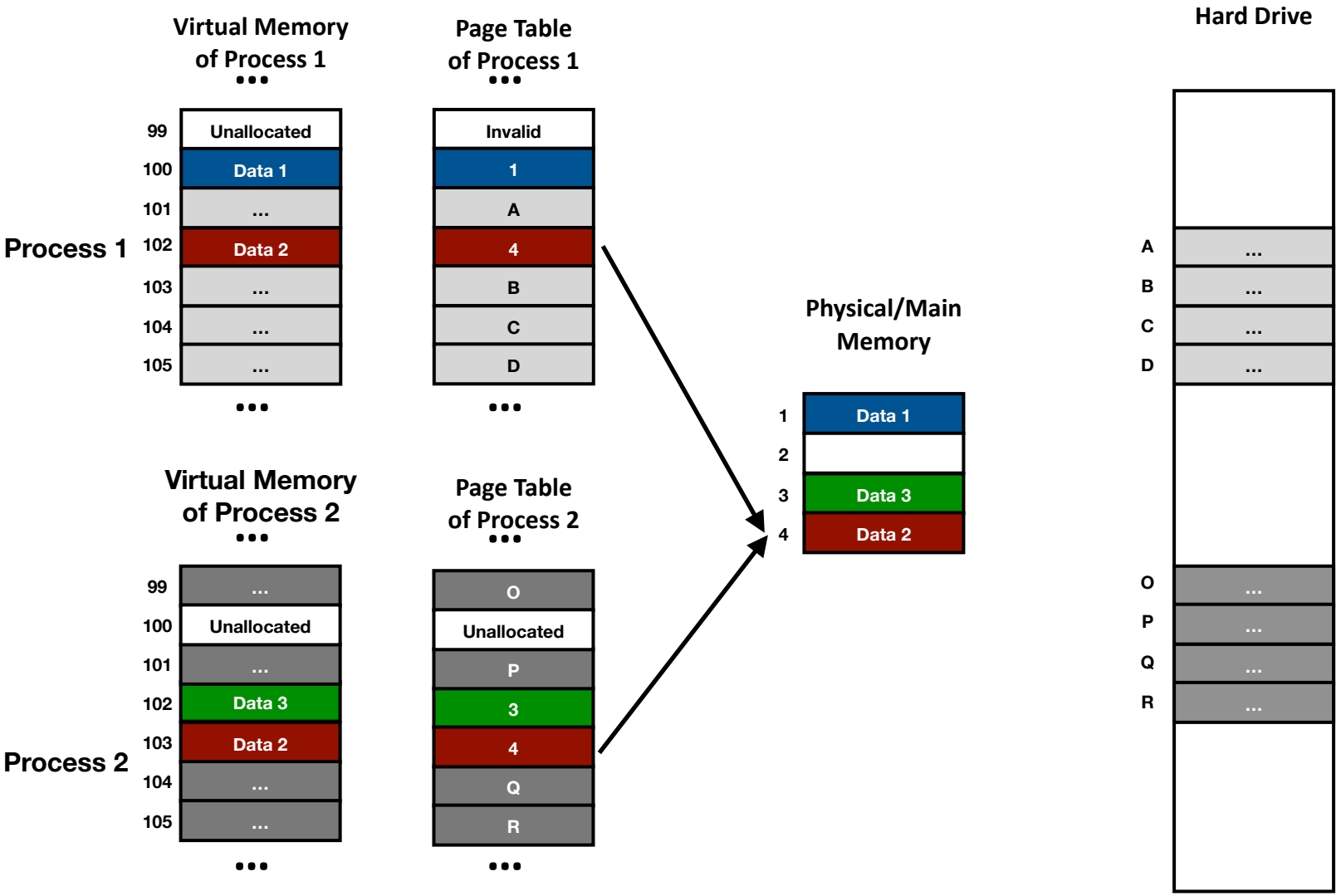
Prevent Unwanted Sharing



Prevent Unwanted Sharing



Enable Benign Sharing



Analogy for Virtual Memory: A Secure Hotel

Analogy for Virtual Memory: A Secure Hotel

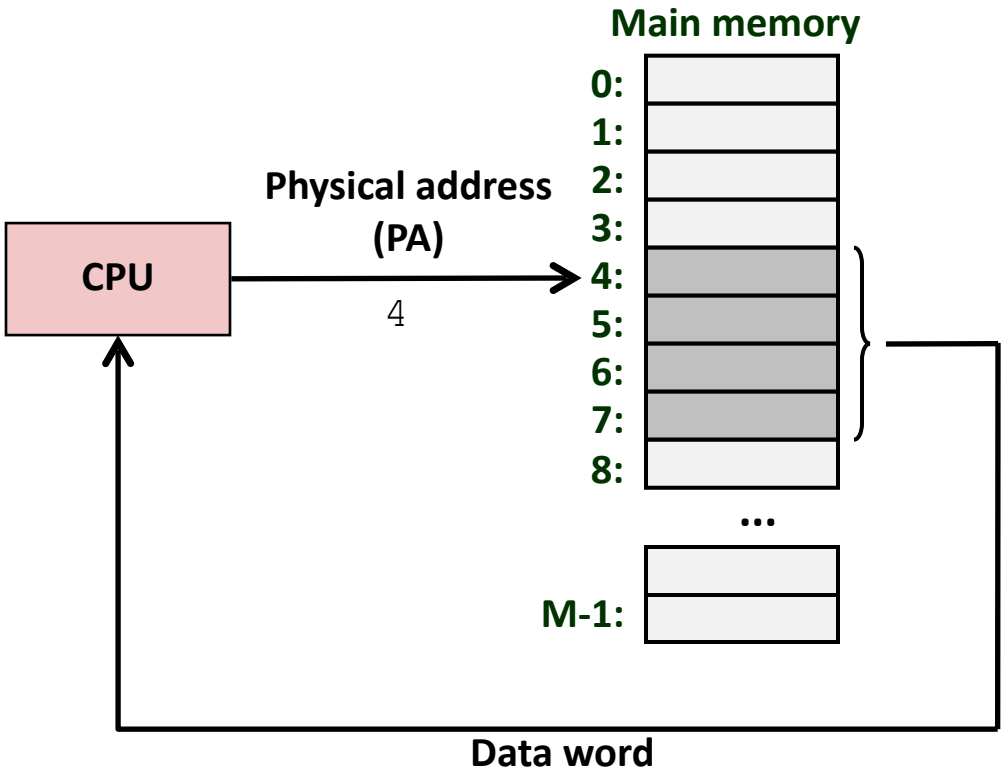
- Call a hotel looking for a guest; what happens?
  - Front desk routes call to room, does not give out room number
  - Guest's name is a virtual address
  - Room number is physical address
  - Front desk is doing address translation!

Analogy for Virtual Memory: 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!

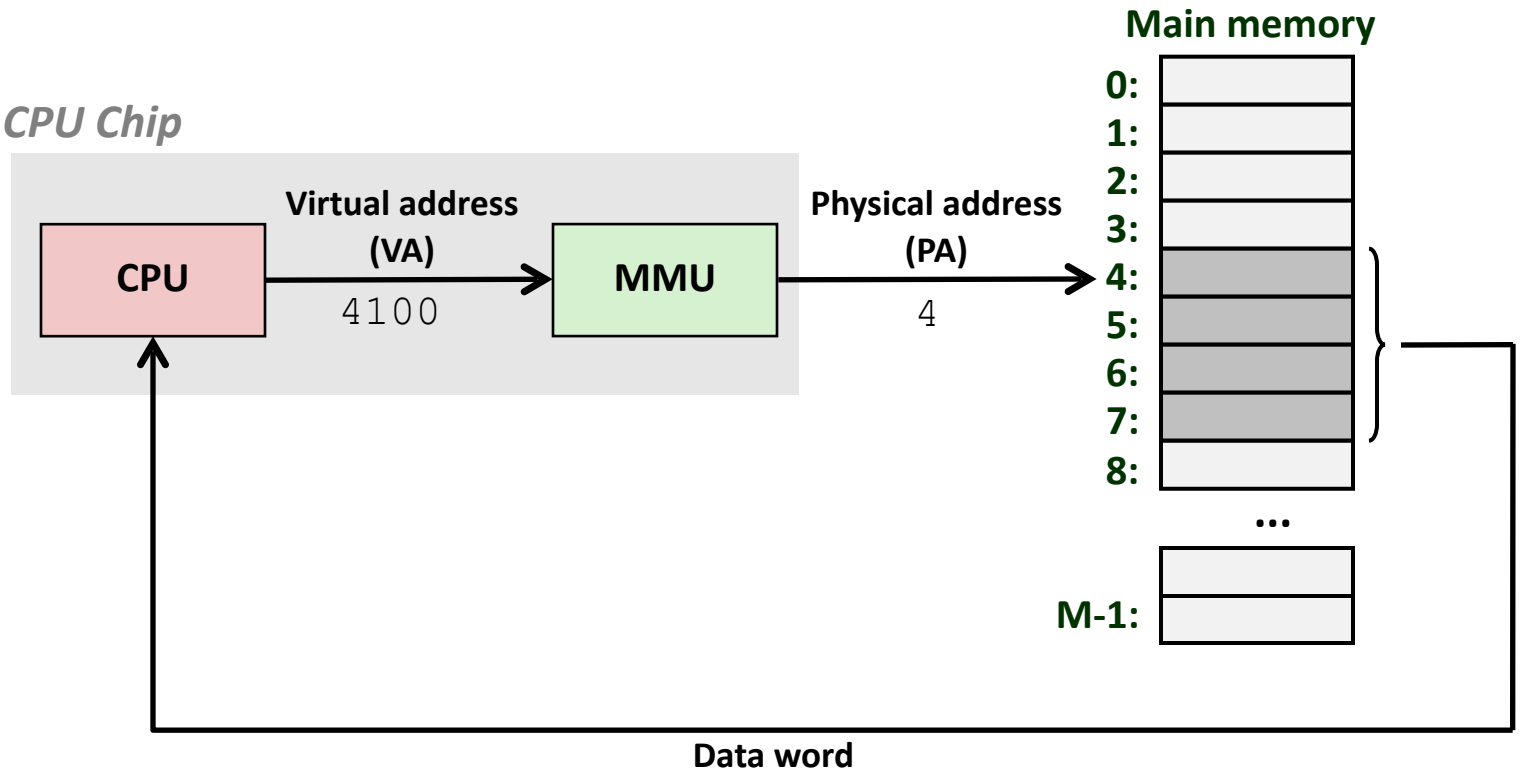
- Benefits
  - **Ease of management**: Guest could change rooms (physical address). You can still find her without knowing it
  - **Protection**: Guest could have block on calls, block on calls from specific callers (permissions)
  - **Sharing**: Multiple guests (virtual addresses) can share the same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

A System Using Physical Memory Only



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

A System Using Virtual Memory



- Used in all modern servers, laptops, and smart phones
- One of the great ideas in computer science (back in the 60s)
- MMU: Memory Management Unit; part of the OS

The Big Idea: Virtual Memory

The Big Idea: Virtual Memory

- What Does a Programmer Want?
  - Infinitely large, infinitely fast memory
  - Strong isolation between processes to prevent unwanted sharing
  - Enable wanted sharing

The Big Idea: Virtual Memory

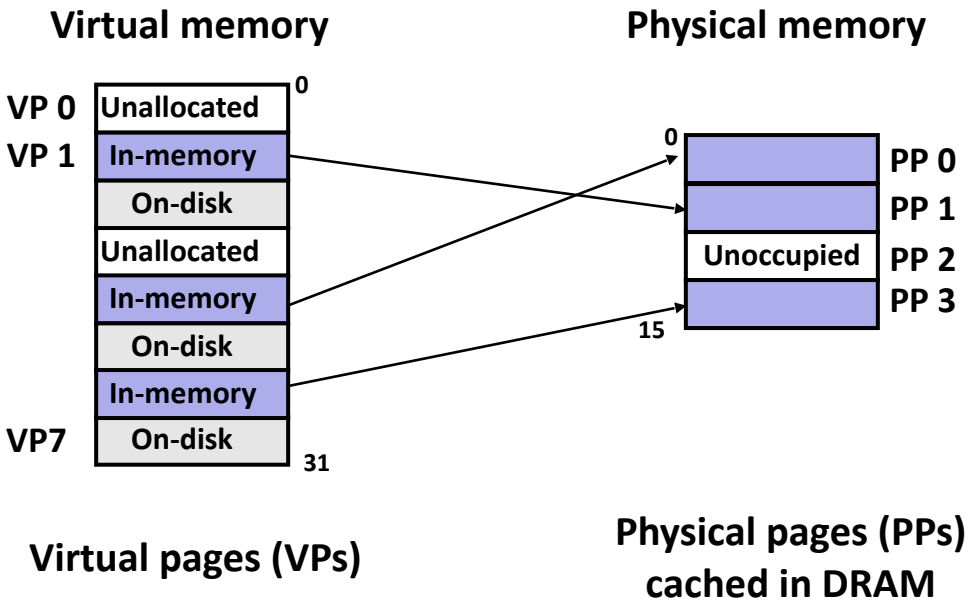
- What Does a Programmer Want?
  - Infinitely large, infinitely fast memory
  - Strong isolation between processes to prevent unwanted sharing
  - Enable wanted sharing
- Virtual memory to the rescue
  - Present a large, uniform memory to programmers
  - Data in virtual memory by default stays in disk
  - 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

- VM basic concepts and operation
- Other critical benefits of VM
- Address translation

VM Concepts

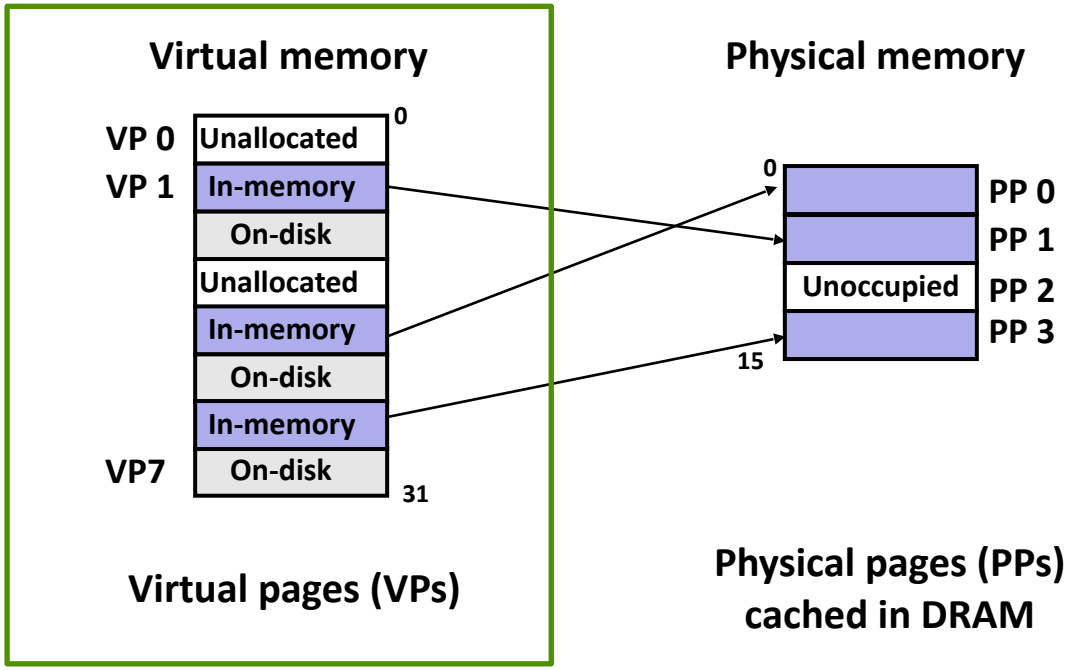
- *Virtual memory* is an array of N contiguous *pages* (each page has a certain amount of continuous bytes).
- Physical memory is also divided into pages. Each physical page (sometimes called *frames*) has the same size as a virtual page. Physical memory has way fewer pages.
- A page can either be on the ("uncached") disk or in the physical memory ("cached").



VM Concepts

- *Virtual memory* is an array of N contiguous *pages* (each page has a certain amount of continuous bytes).
- Physical memory is also divided into pages. Each physical page (sometimes called *frames*) has the same size as a virtual page. Physical memory has way fewer pages.
- A page can either be on the ("uncached") disk or in the physical memory ("cached").

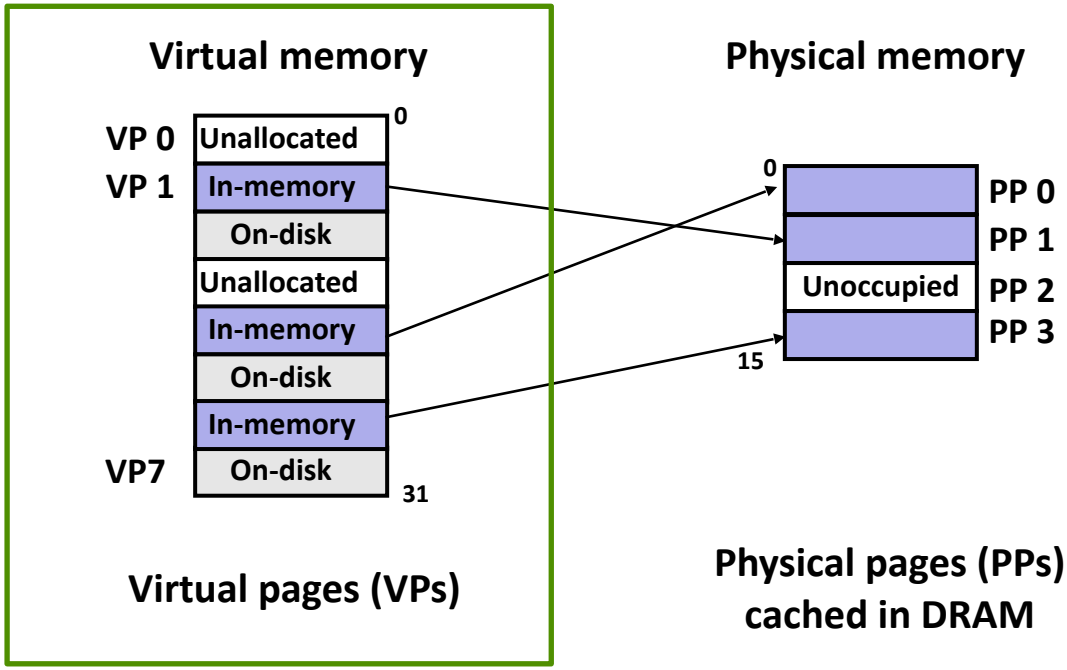
What programmers see



VM Concepts

- *Virtual memory* is an array of N contiguous *pages* (each page has a certain amount of continuous bytes).
- Physical memory is also divided into pages. Each physical page (sometimes called *frames*) has the same size as a virtual page. Physical memory has way fewer pages.
- A page can either be on the ("uncached") disk or in the physical memory ("cached").

What programmers see

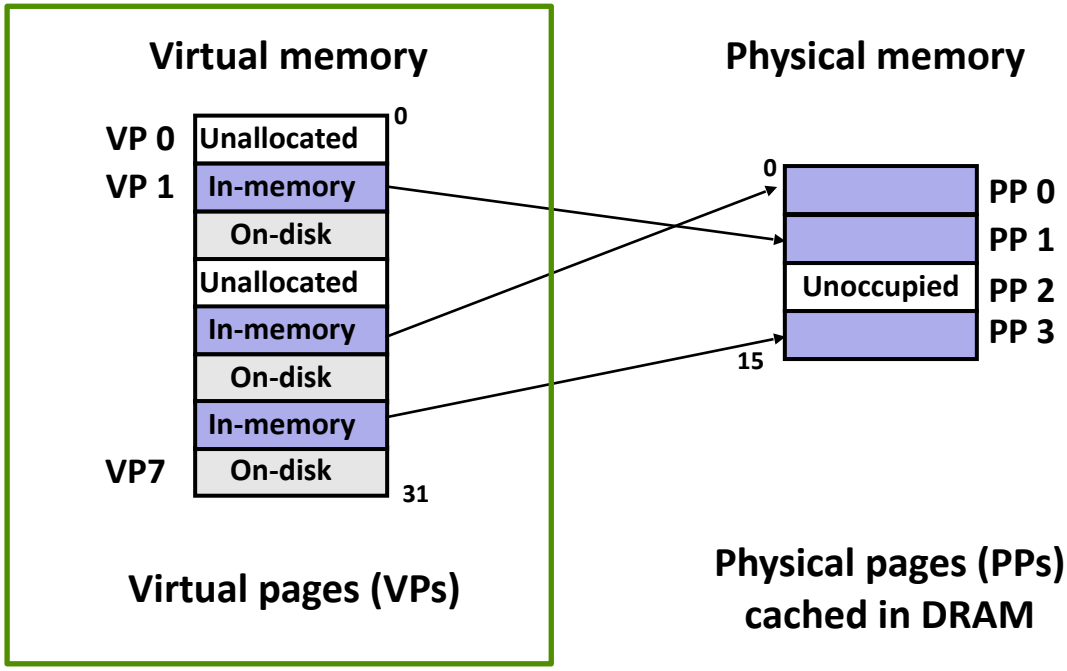


Assuming page size is 4B
Virtual memory size is 32B
Physical memory size is 16B

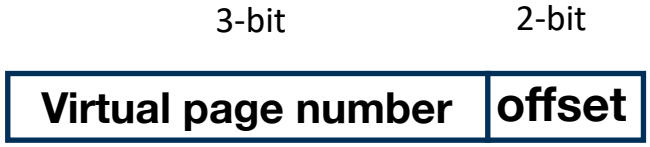
VM Concepts

- *Virtual memory* is an array of N contiguous *pages* (each page has a certain amount of continuous bytes).
- Physical memory is also divided into pages. Each physical page (sometimes called *frames*) has the same size as a virtual page. Physical memory has way fewer pages.
- A page can either be on the ("uncached") disk or in the physical memory ("cached").

What programmers see



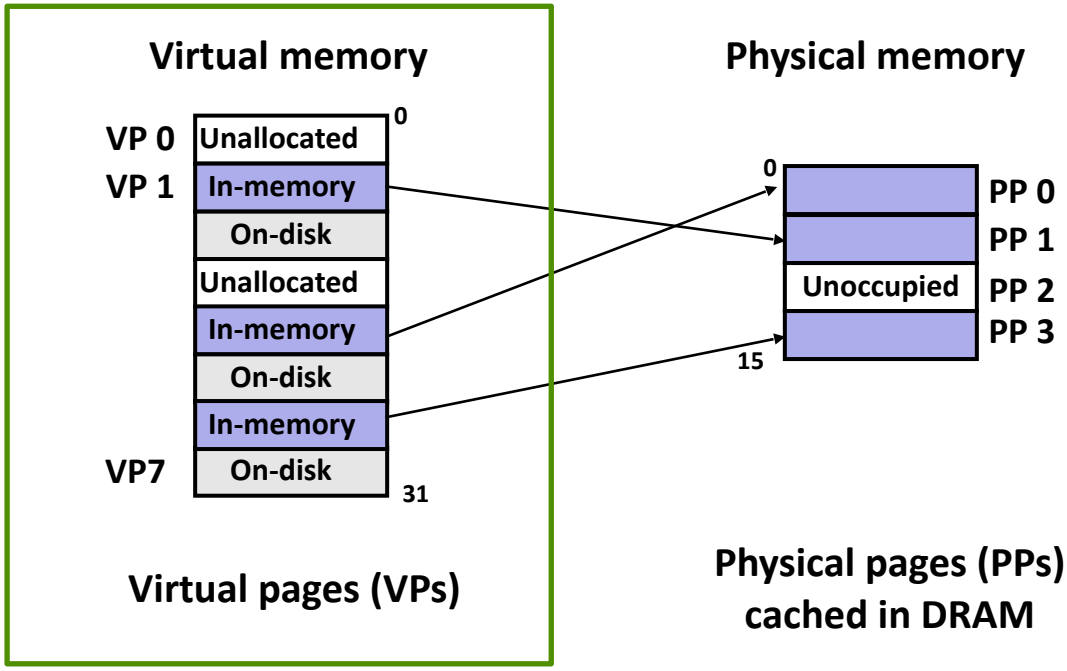
Assuming page size is 4B
Virtual memory size is 32B
Physical memory size is 16B



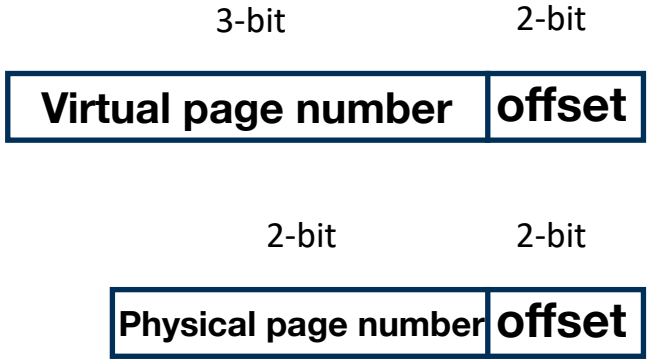
VM Concepts

- *Virtual memory* is an array of N contiguous *pages* (each page has a certain amount of continuous bytes).
- Physical memory is also divided into pages. Each physical page (sometimes called *frames*) has the same size as a virtual page. Physical memory has way fewer pages.
- A page can either be on the ("uncached") disk or in the physical memory ("cached").

What programmers see



Assuming page size is 4B
Virtual memory size is 32B
Physical memory size is 16B



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 corresponding 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 corresponding 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 corresponding 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 corresponding 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

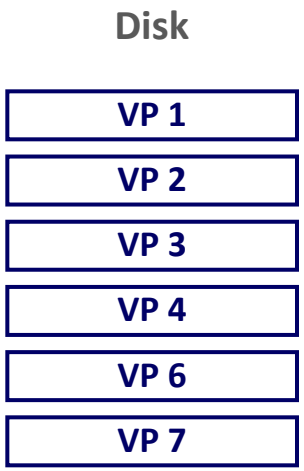
- How do we track which virtual pages are mapped to physical pages, and where they are mapped?
- Use a table to track this. The table is called page table, in which each virtual page has an entry
- Each entry records whether the corresponding virtual page is mapped to the physical memory
  - If mapped, 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, i.e., virtual to physical address translation.
- One PTE for each virtual page.

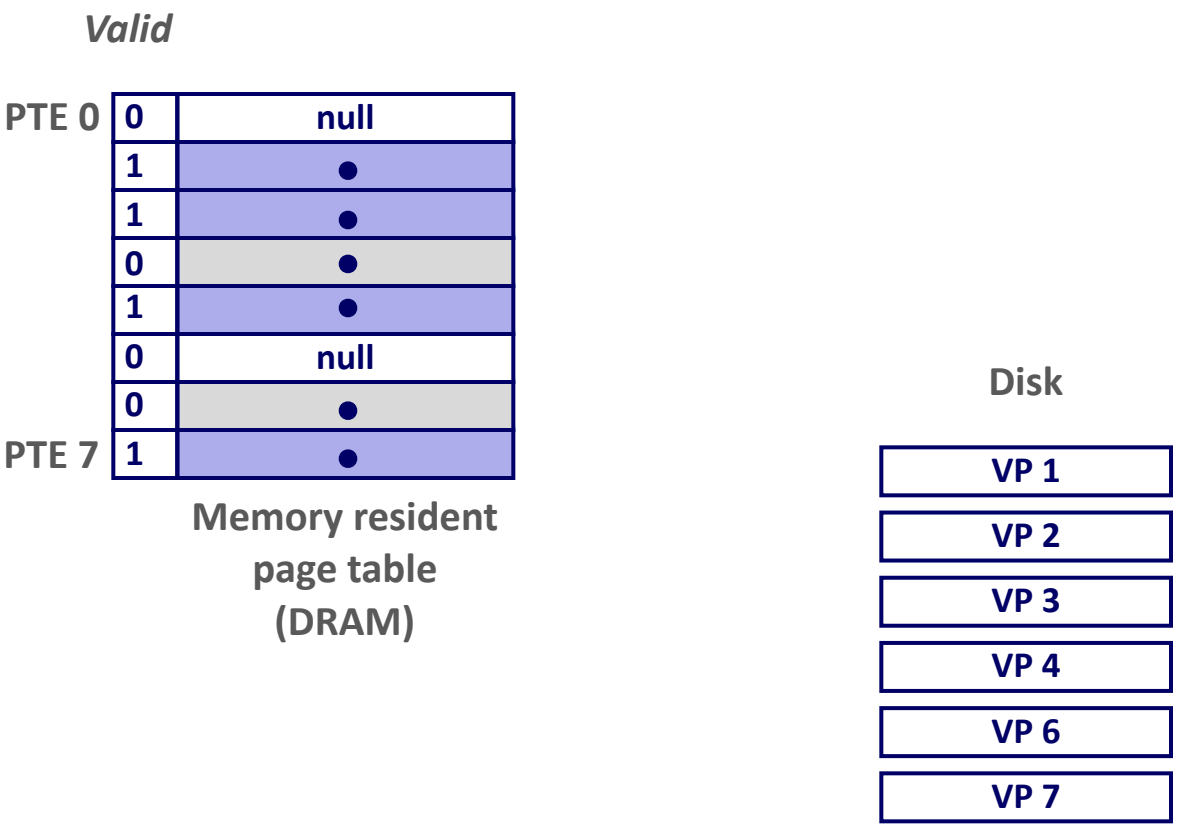
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page, i.e., virtual to physical address translation.
- One PTE for each virtual page.



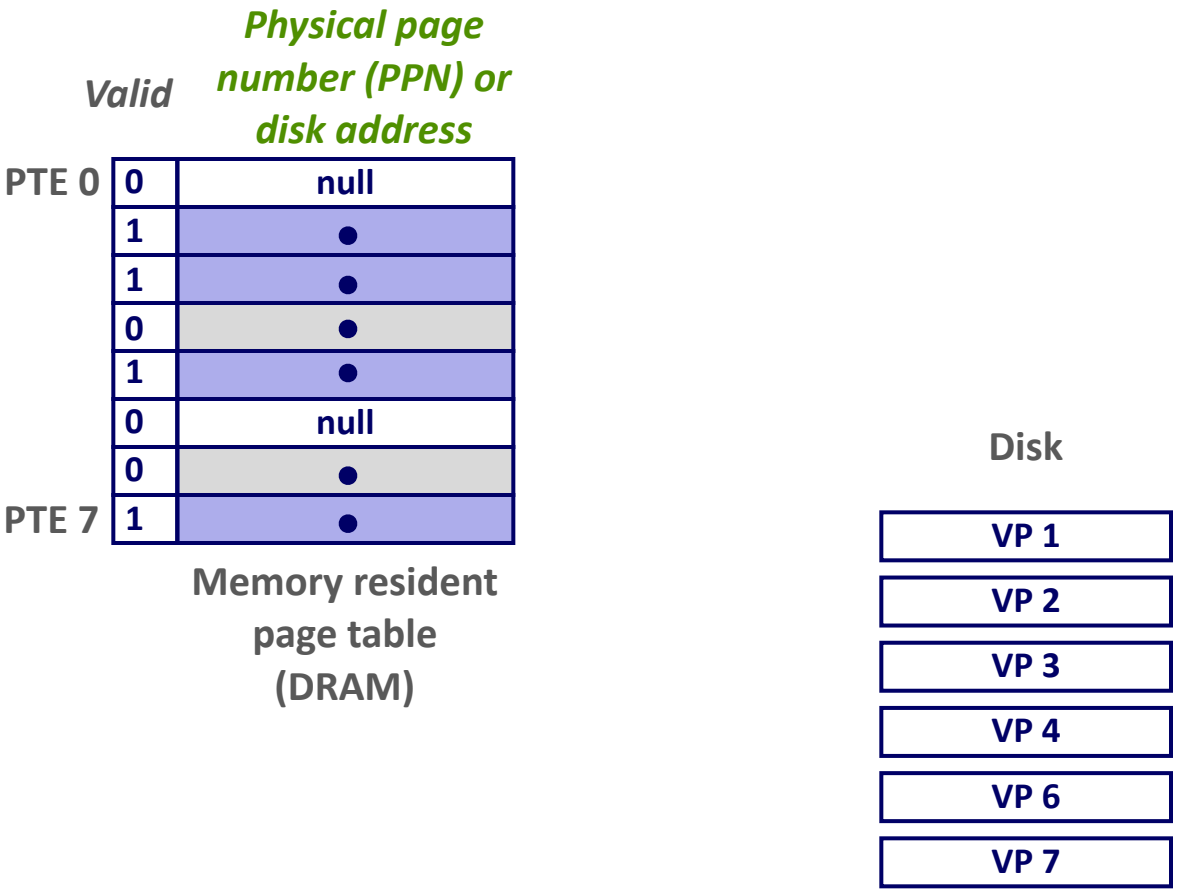
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page, i.e., virtual to physical address translation.
- One PTE for each virtual page.



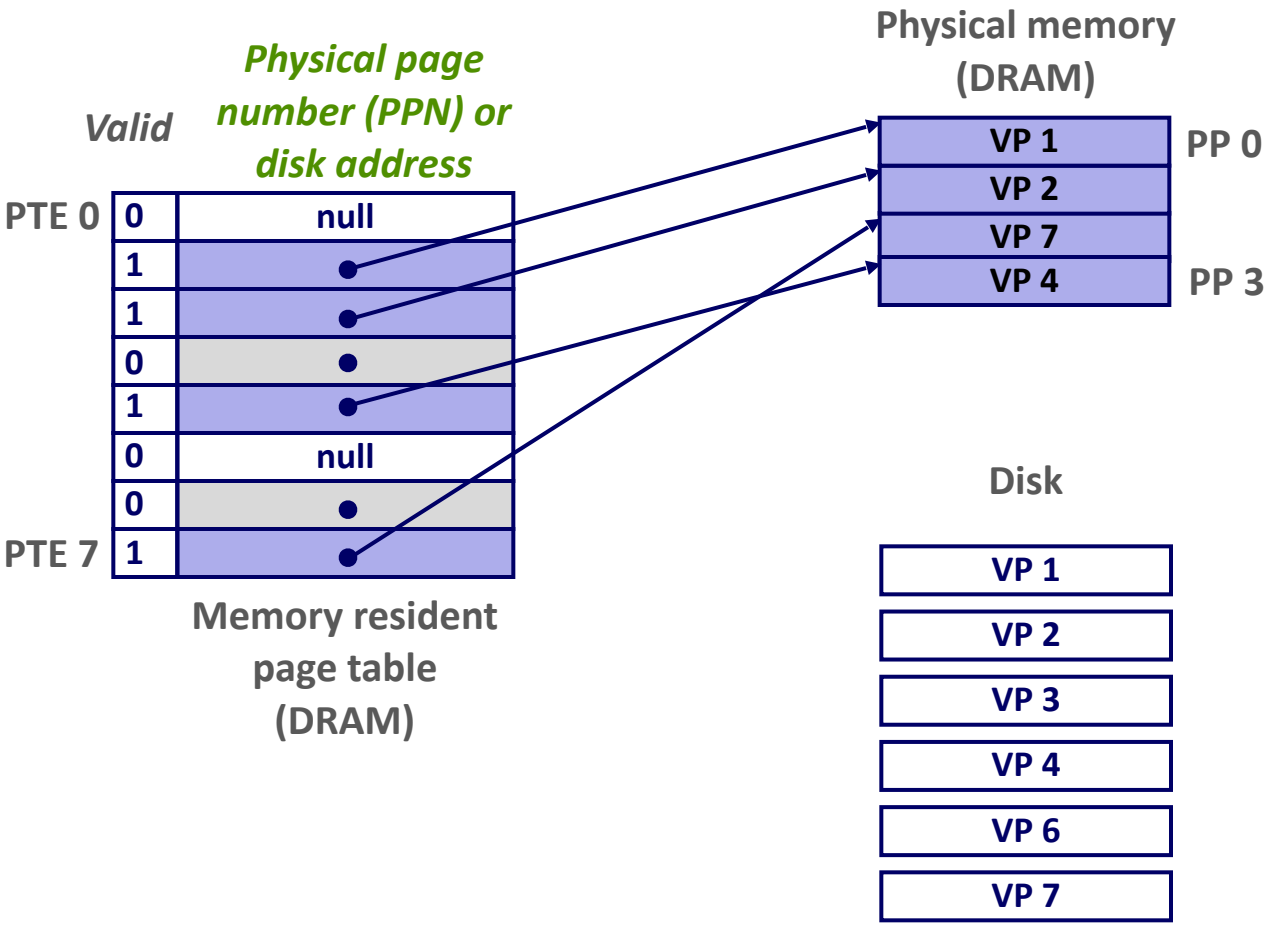
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page, i.e., virtual to physical address translation.
- One PTE for each virtual page.



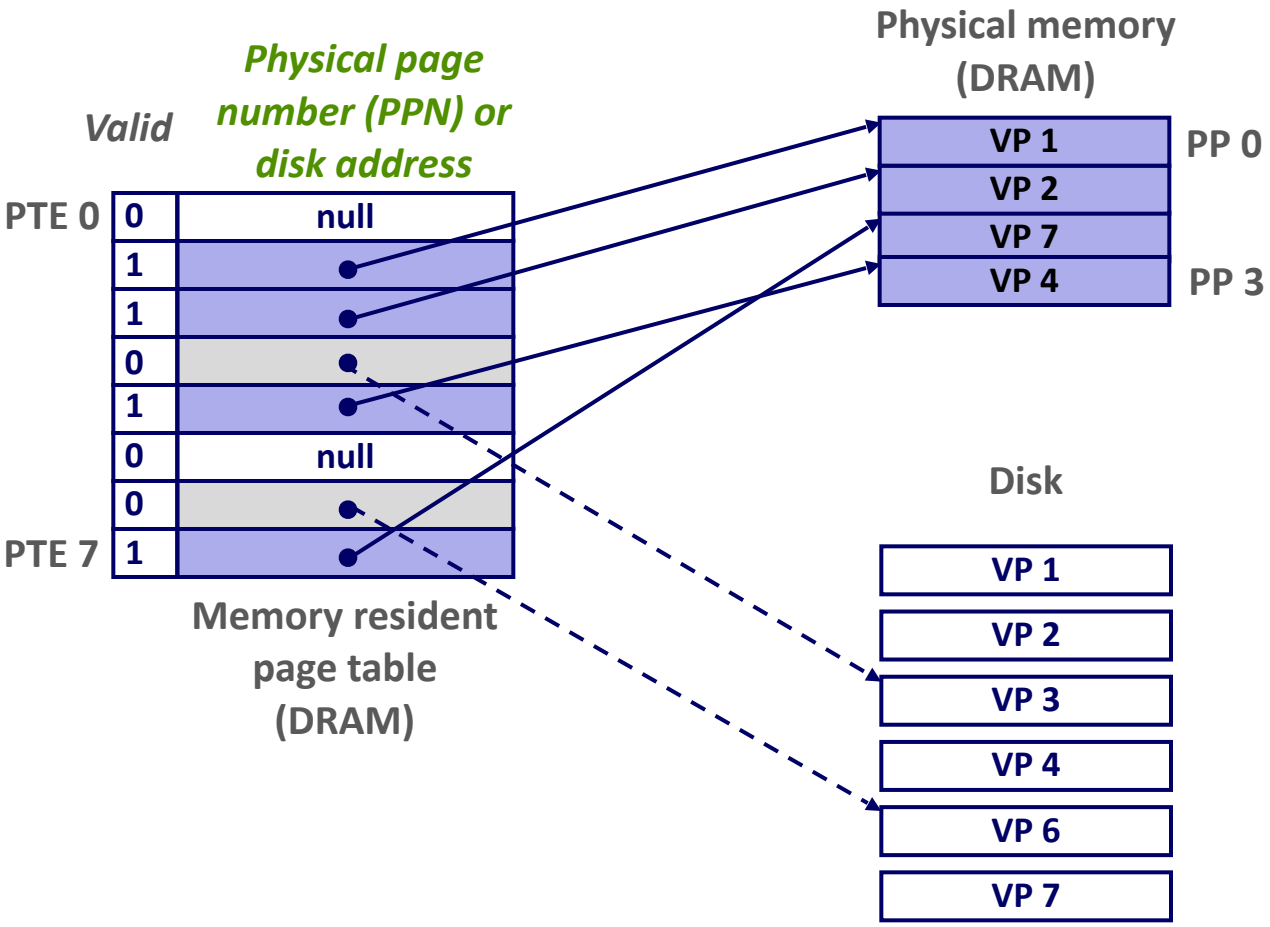
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page, i.e., virtual to physical address translation.
- One PTE for each virtual page.



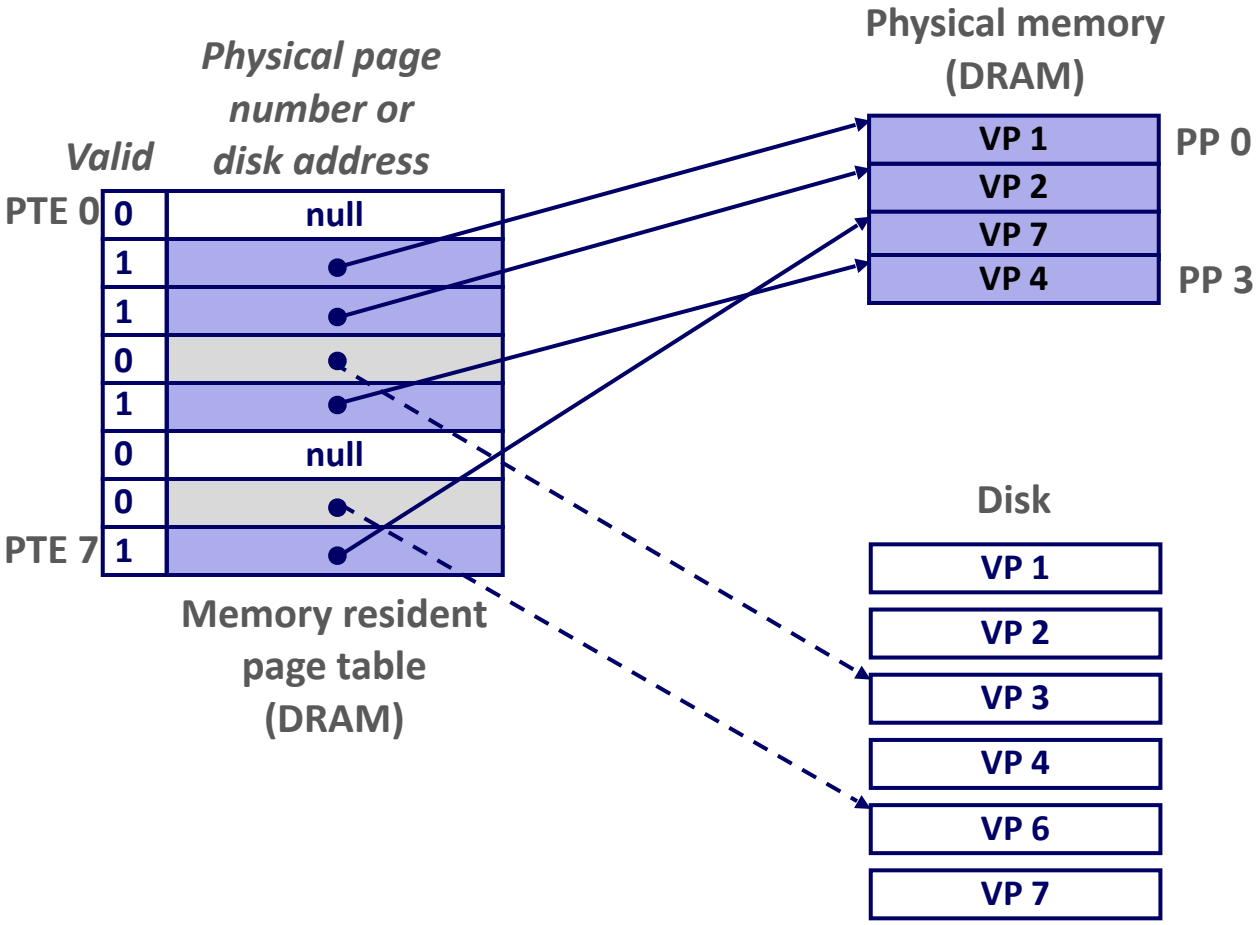
Enabling Data Structure: Page Table

- A page table is an array of page table entries (PTEs) that maps every virtual page to its physical page, i.e., virtual to physical address translation.
- One PTE for each virtual page.



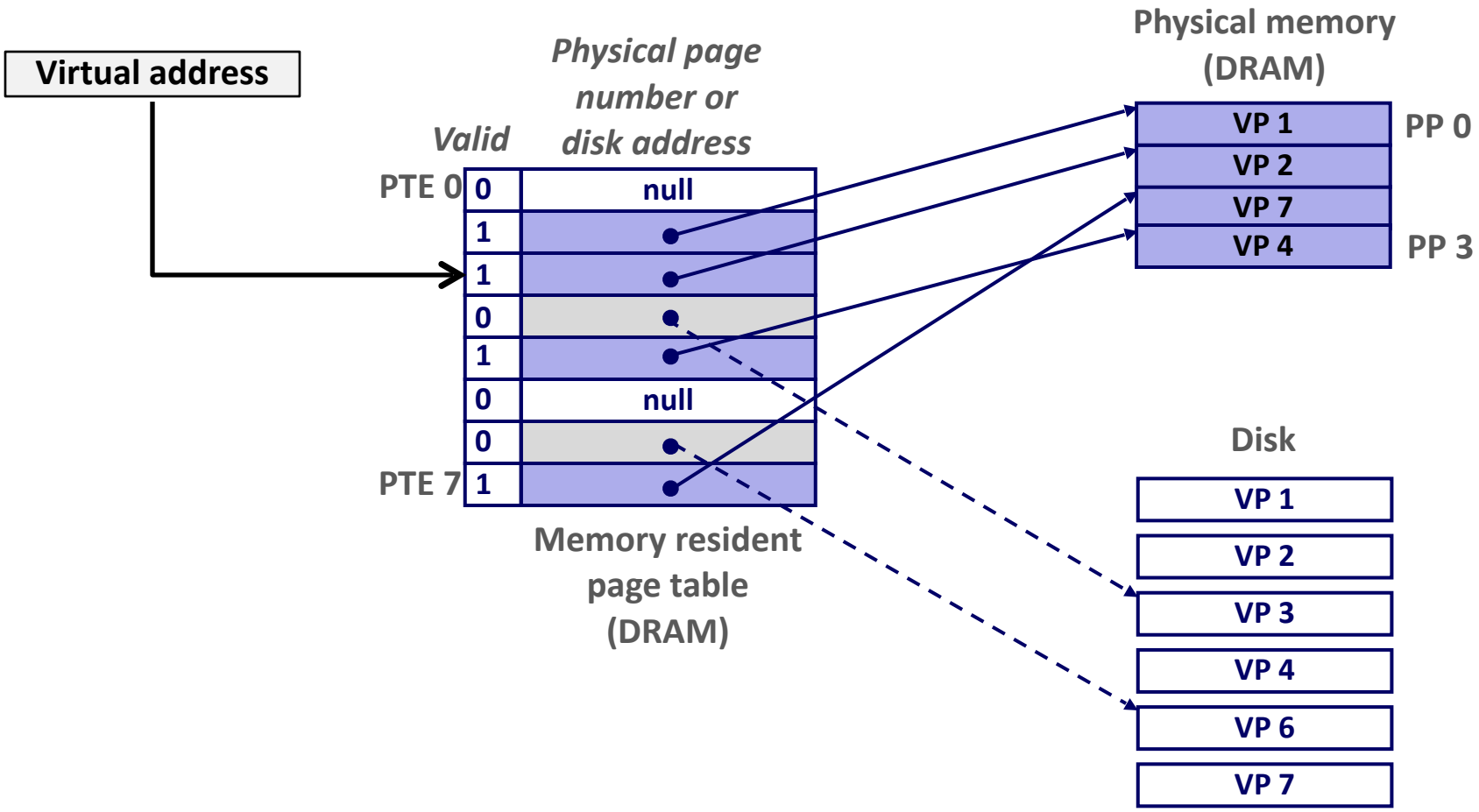
Page Hit

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



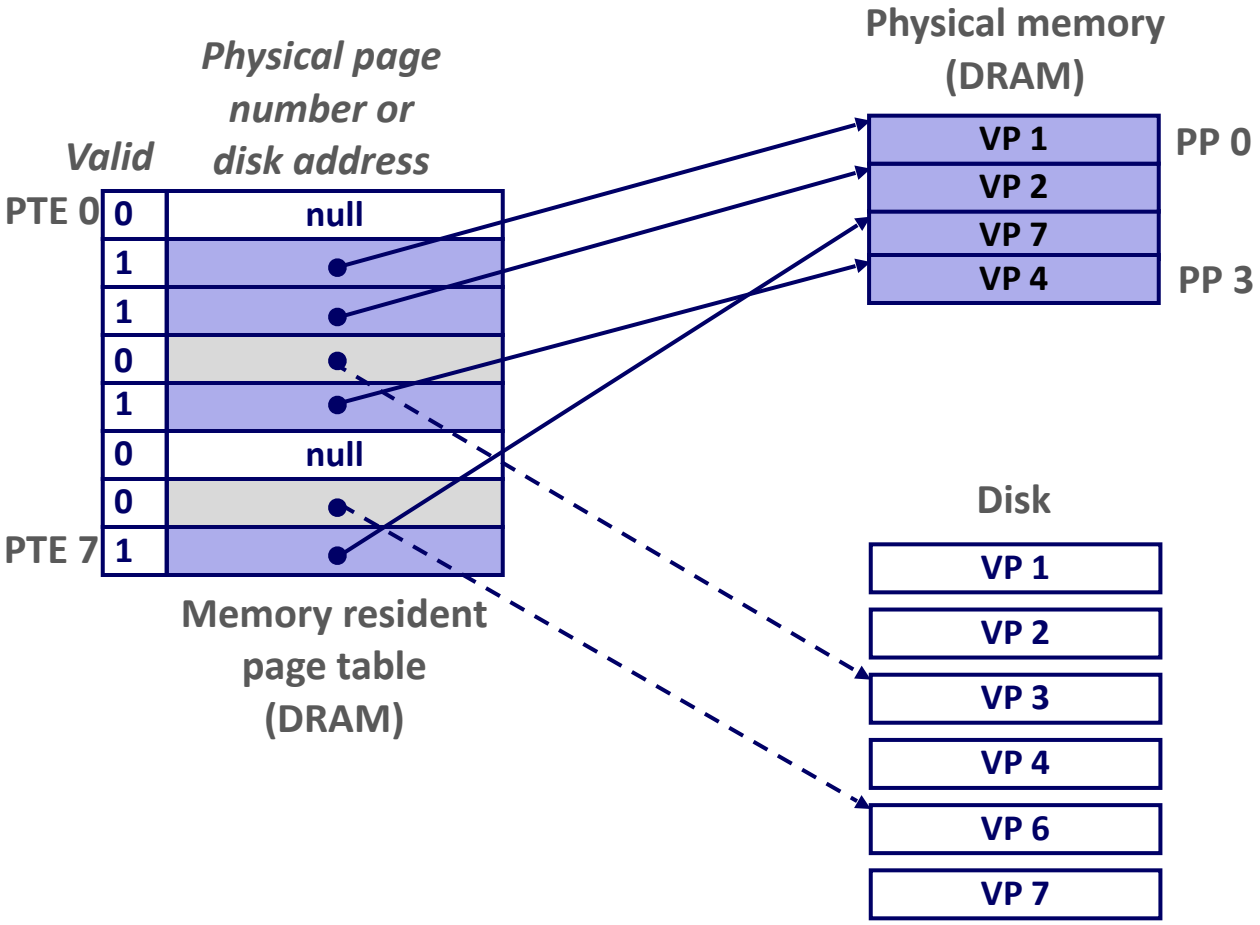
Page Hit

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



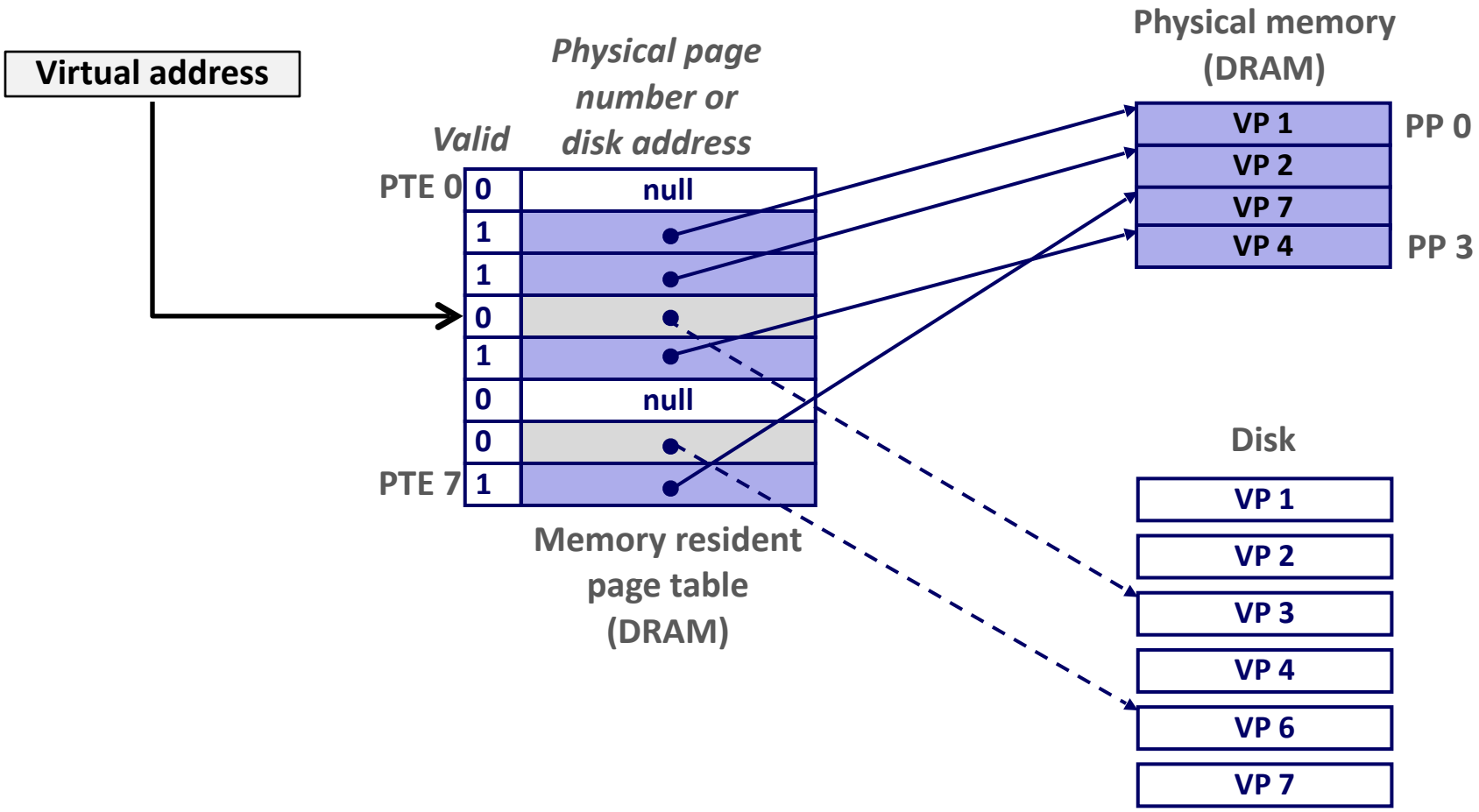
Page Fault

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



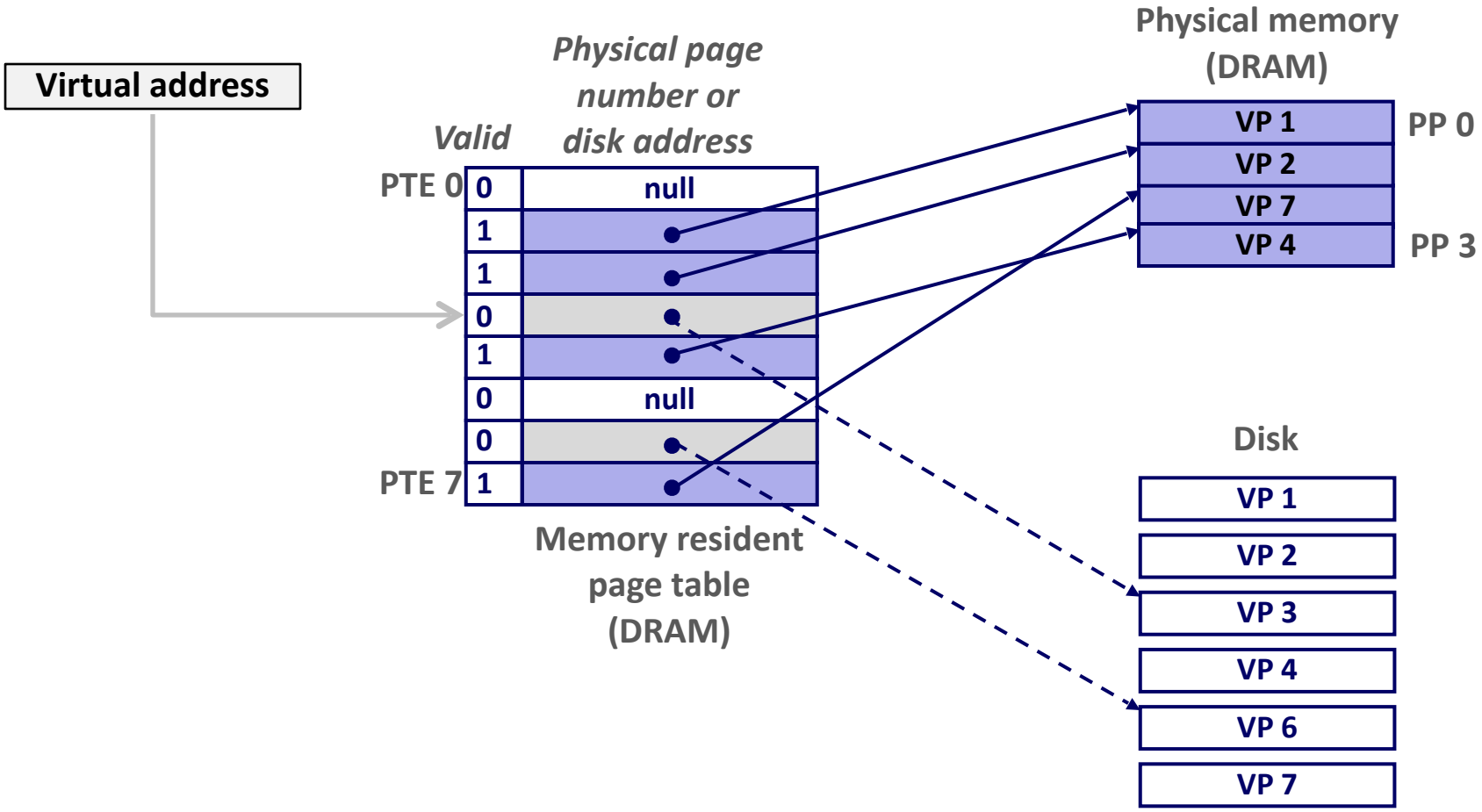
Page Fault

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



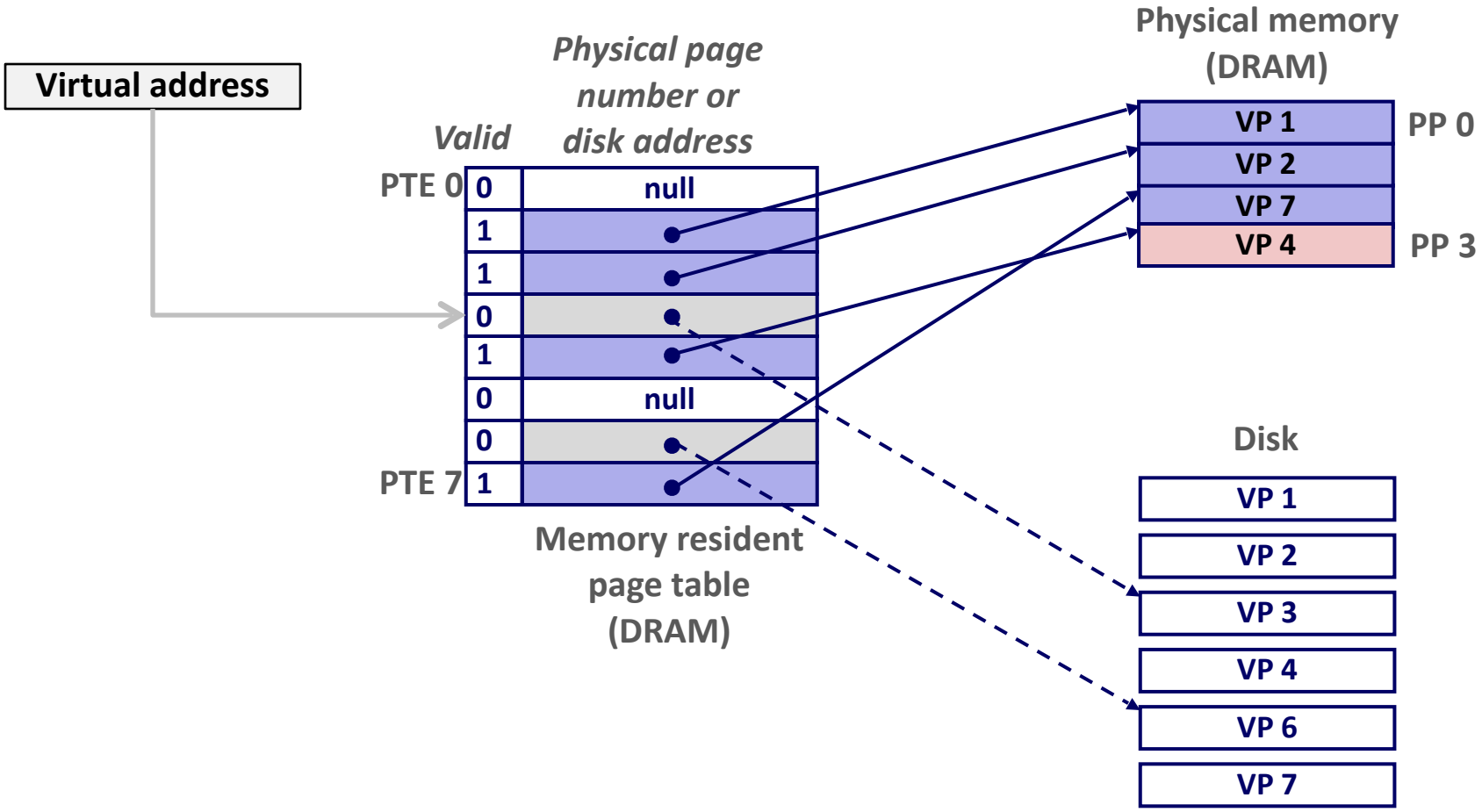
Handling Page Fault

- Page miss causes ***page fault (an exception)***



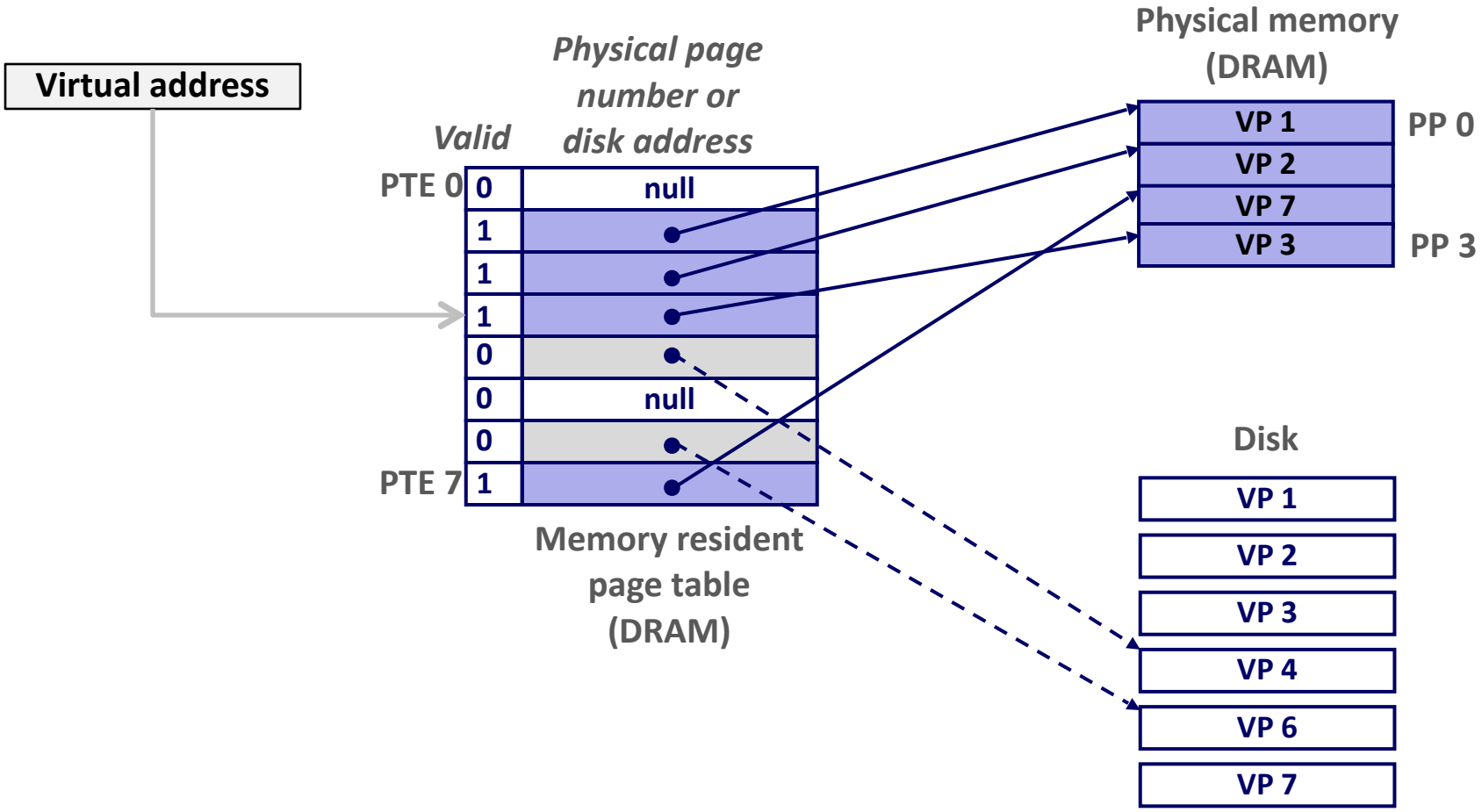
Handling Page Fault

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



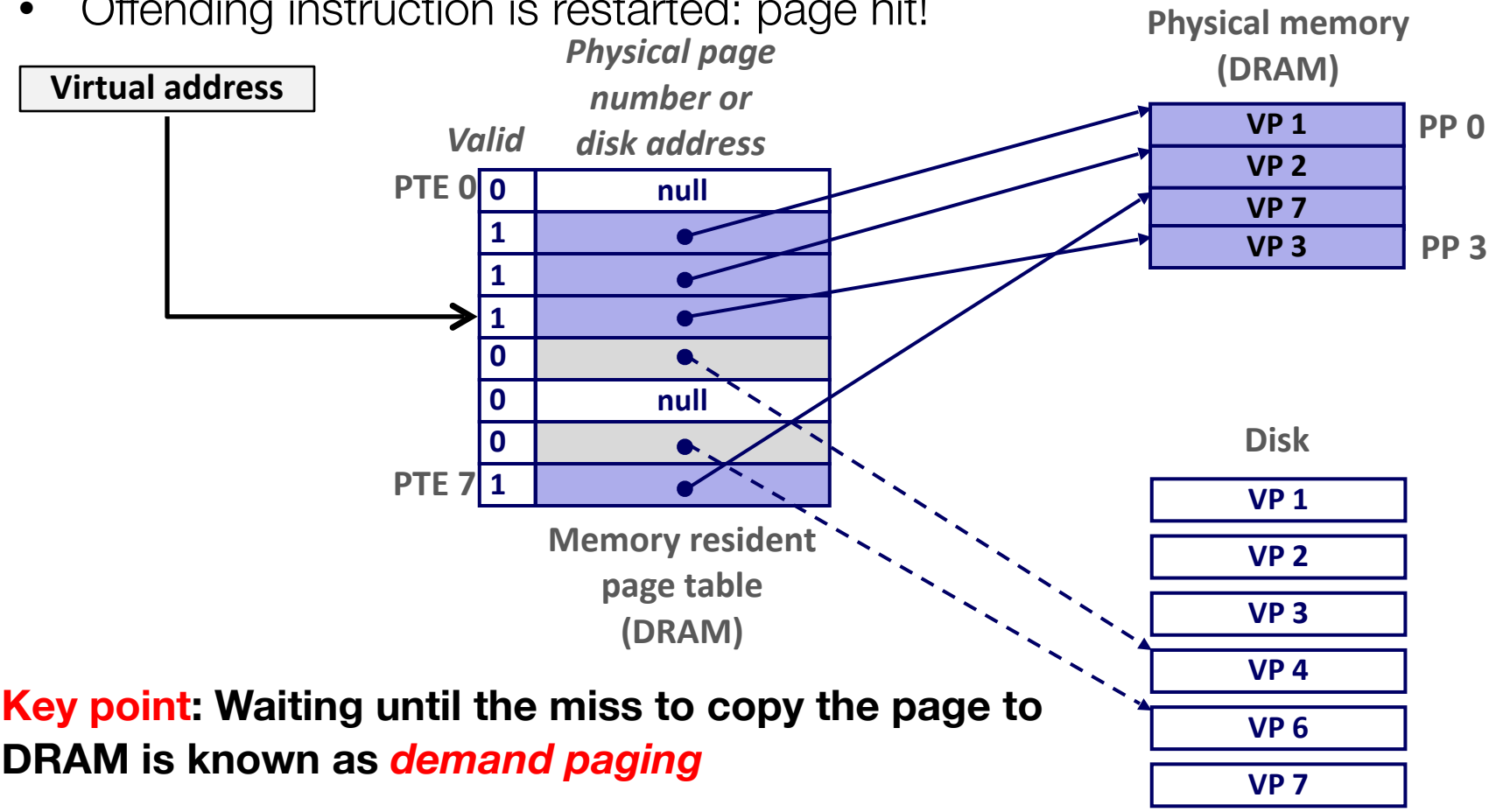
Handling Page Fault

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



Handling Page Fault

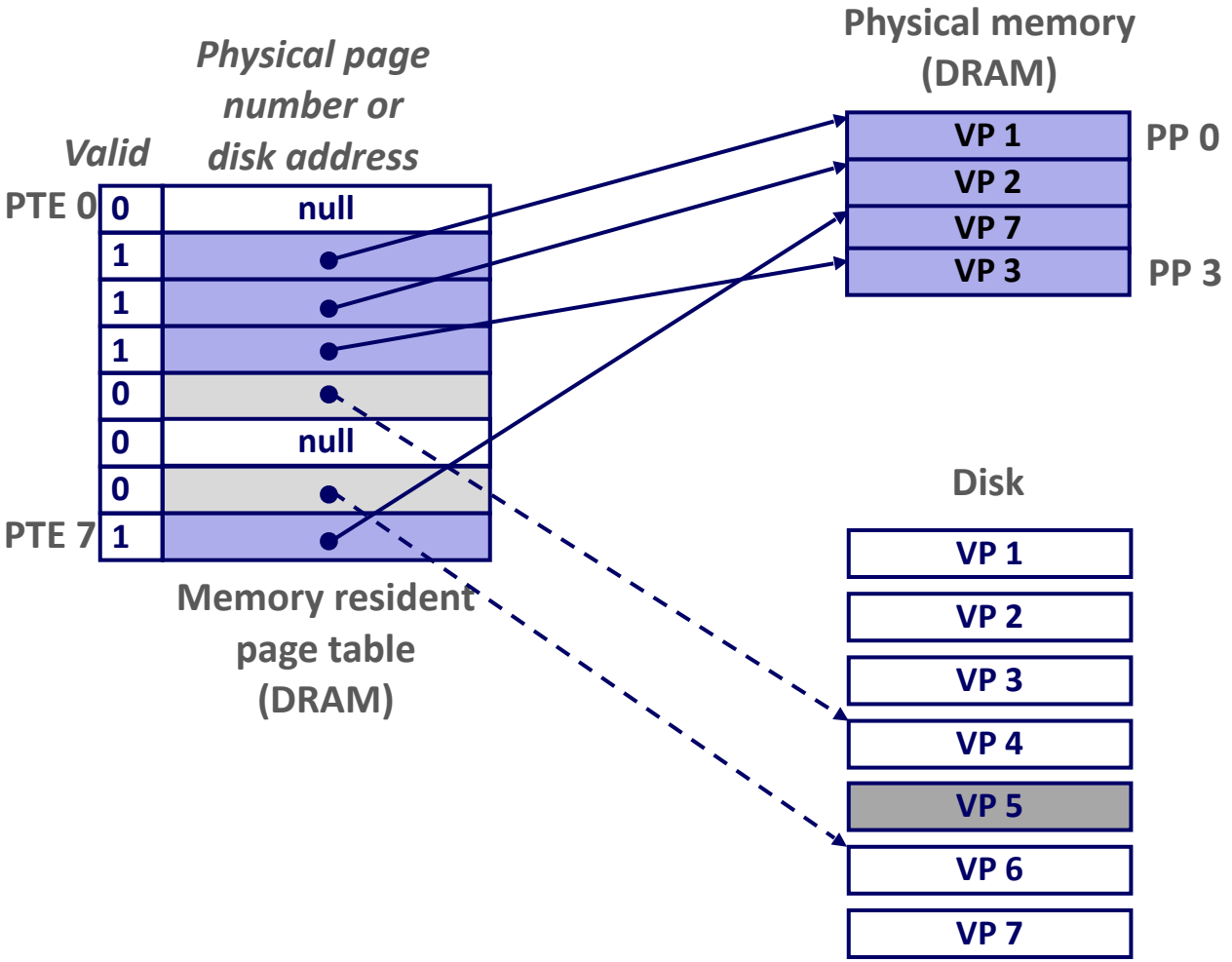
- Page miss causes ***page fault (an exception)***
- Page fault ***handler*** selects a victim to be evicted (here VP 4)
- Offending instruction is restarted: page hit!



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

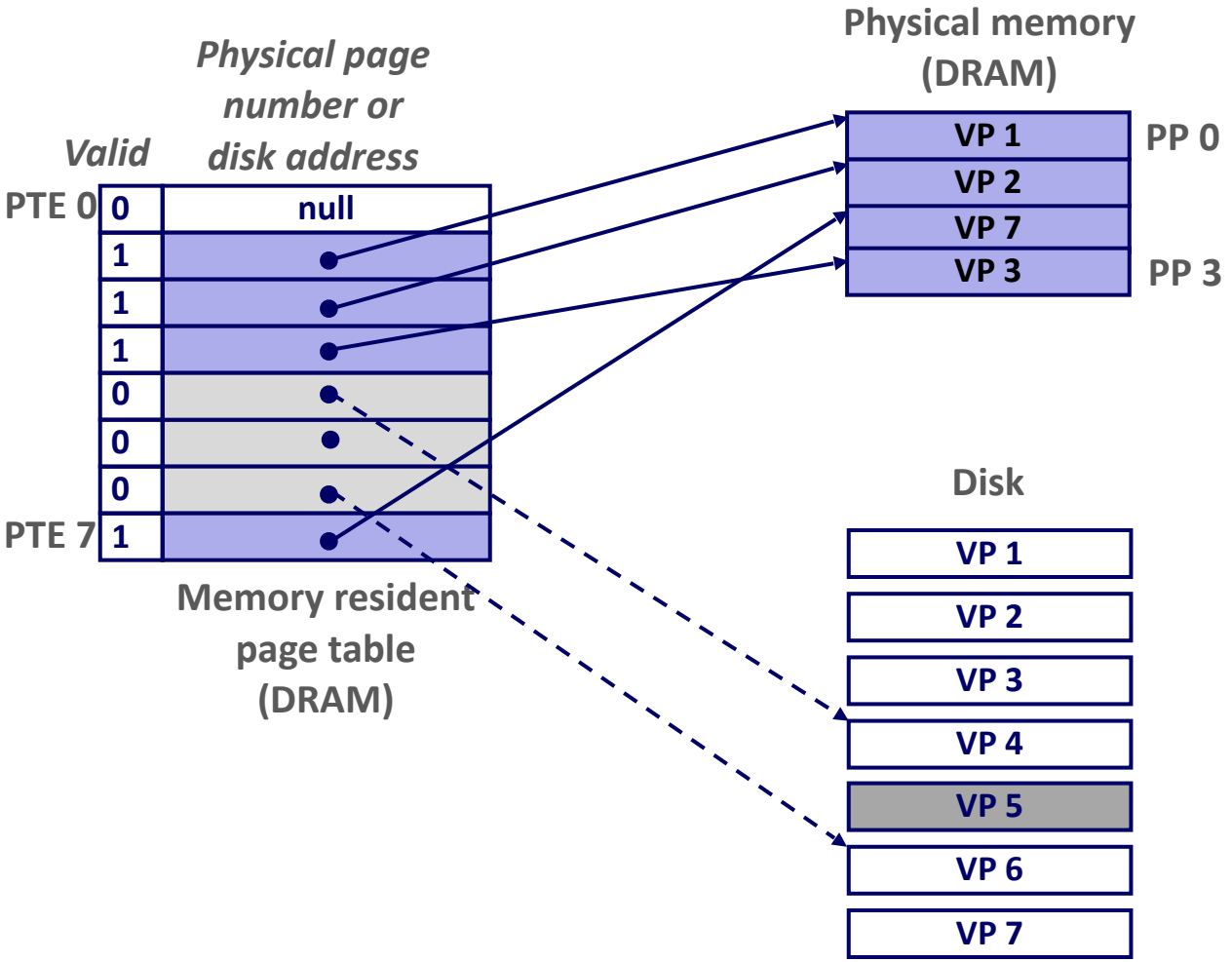
Allocating Pages

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



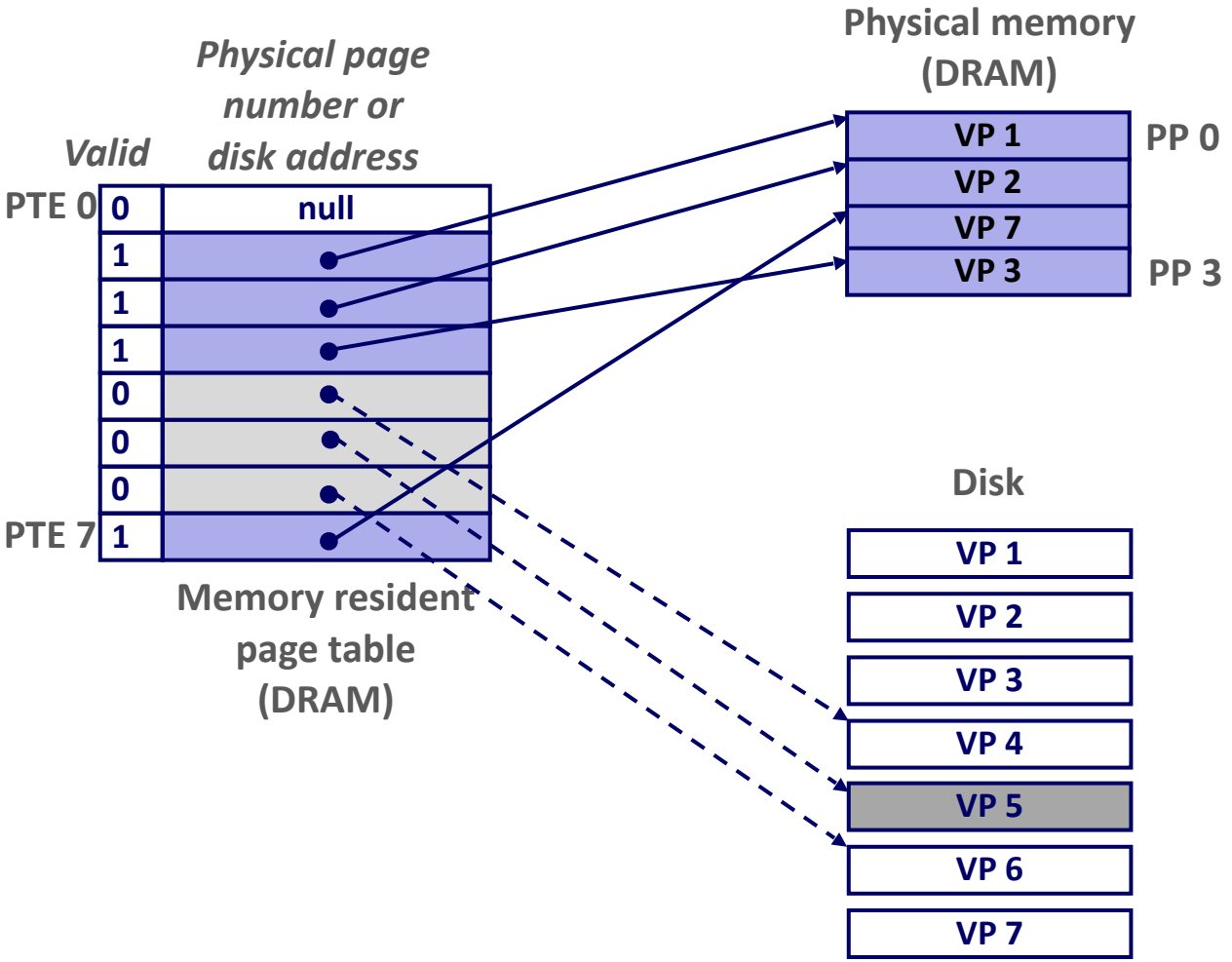
Allocating Pages

- 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

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated
- The physical memory (PM) is an array of M *pages* stored in DRAM.

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated
- The physical memory (PM) is an array of M *pages* stored in DRAM.
- Page size is the same for VM and PM

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated
- The physical memory (PM) is an array of M *pages* stored in DRAM.
- Page size is the same for VM and PM
- M << N

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated
- The physical memory (PM) is an array of M *pages* stored in DRAM.
- Page size is the same for VM and PM
- M << N
- On a 64-bit machine, virtual memory size = 2^{64}

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated
- The physical memory (PM) is an array of M *pages* stored in DRAM.
- Page size is the same for VM and PM
- M << N
- On a 64-bit machine, virtual memory size = 2^{64}
- Physical memory size is much much smaller:
  - iPhone 8: 2 GB (2^{31})
  - 15-inch Macbook Pro 2017: 16 GB (2^{34})

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated
- The physical memory (PM) is an array of M *pages* stored in DRAM.
- Page size is the same for VM and PM
- M << N
- On a 64-bit machine, virtual memory size = 2^{64}
- Physical memory size is much much smaller:
  - iPhone 8: 2 GB (2^{31})
  - 15-inch Macbook Pro 2017: 16 GB (2^{34})
- Store only the most frequently used pages in the physical memory

VM Concepts Summary

- Conceptually, *virtual memory* is an array of N *pages*
  - Each virtual page is either in physical memory, or on disk, or unallocated
- The physical memory (PM) is an array of M *pages* stored in DRAM.
- Page size is the same for VM and PM
- M << N
- On a 64-bit machine, virtual memory size = 2^{64}
- Physical memory size is much much smaller:
  - iPhone 8: 2 GB (2^{31})
  - 15-inch Macbook Pro 2017: 16 GB (2^{34})
- Store only the most frequently used pages in the physical memory
- If a page is not on the physical memory, have to first swap it from the disk to the DRAM.

Calculate the Page Table Size

- Assume 4KB page, 4GB virtual memory, each PTE is 8 Bytes

Calculate the Page Table Size

- Assume 4KB page, 4GB virtual memory, each PTE is 8 Bytes
  - 4GB/4KB = 1M virtual pages

Calculate the Page Table Size

- Assume 4KB page, 4GB virtual memory, each PTE is 8 Bytes
  - 4GB/4KB = 1M virtual pages
  - 1M PTEs in a page table

Calculate the Page Table Size

- Assume 4KB page, 4GB virtual memory, each PTE is 8 Bytes
  - 4GB/4KB = 1M virtual pages
  - 1M PTEs in a page table
  - 8MB total size per page table

Calculate the Page Table Size

- Assume 4KB page, 4GB virtual memory, each PTE is 8 Bytes
  - 4GB/4KB = 1M virtual pages
  - 1M PTEs in a page table
  - 8MB total size per page table
- Do you need a page table for each process?

Calculate the Page Table Size

- Assume 4KB page, 4GB virtual memory, each PTE is 8 Bytes
  - 4GB/4KB = 1M virtual pages
  - 1M PTEs in a page table
  - 8MB total size per page table
- Do you need a page table for each process?
  - Yes

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?

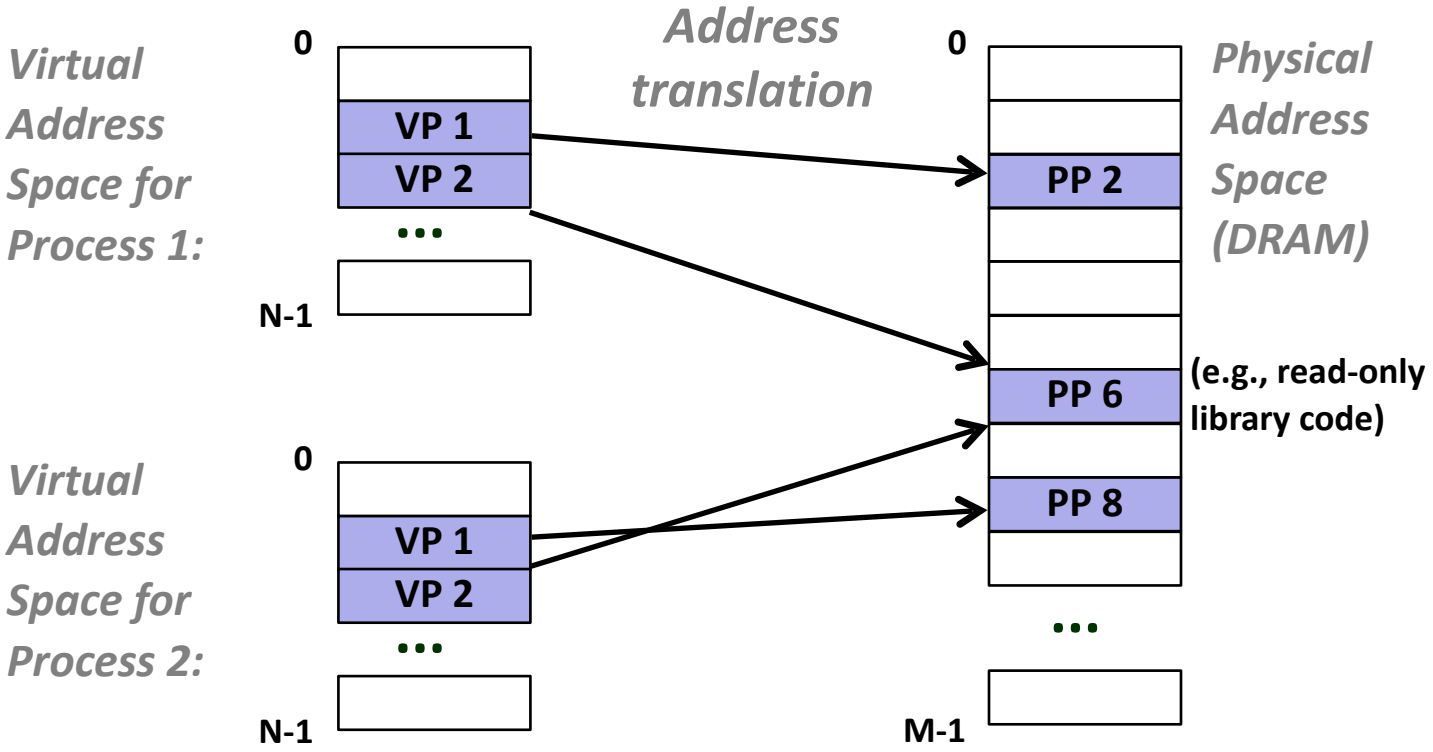
- It needs to be at a specific location where we can find it
  - Some special SRAM?
  - In main memory?
  - On disk?
- ~MBs of a page table *per process*
  - Too big for on-chip SRAM (c.f., a L1 cache is ~32 KB)
  - Too slow to access in disk
  - Put the page table in DRAM, with its start address stored in a special register (Page Table Base Register). More on this later.

Today

- VM basic concepts and operation
- Other critical benefits of VM
- Address translation

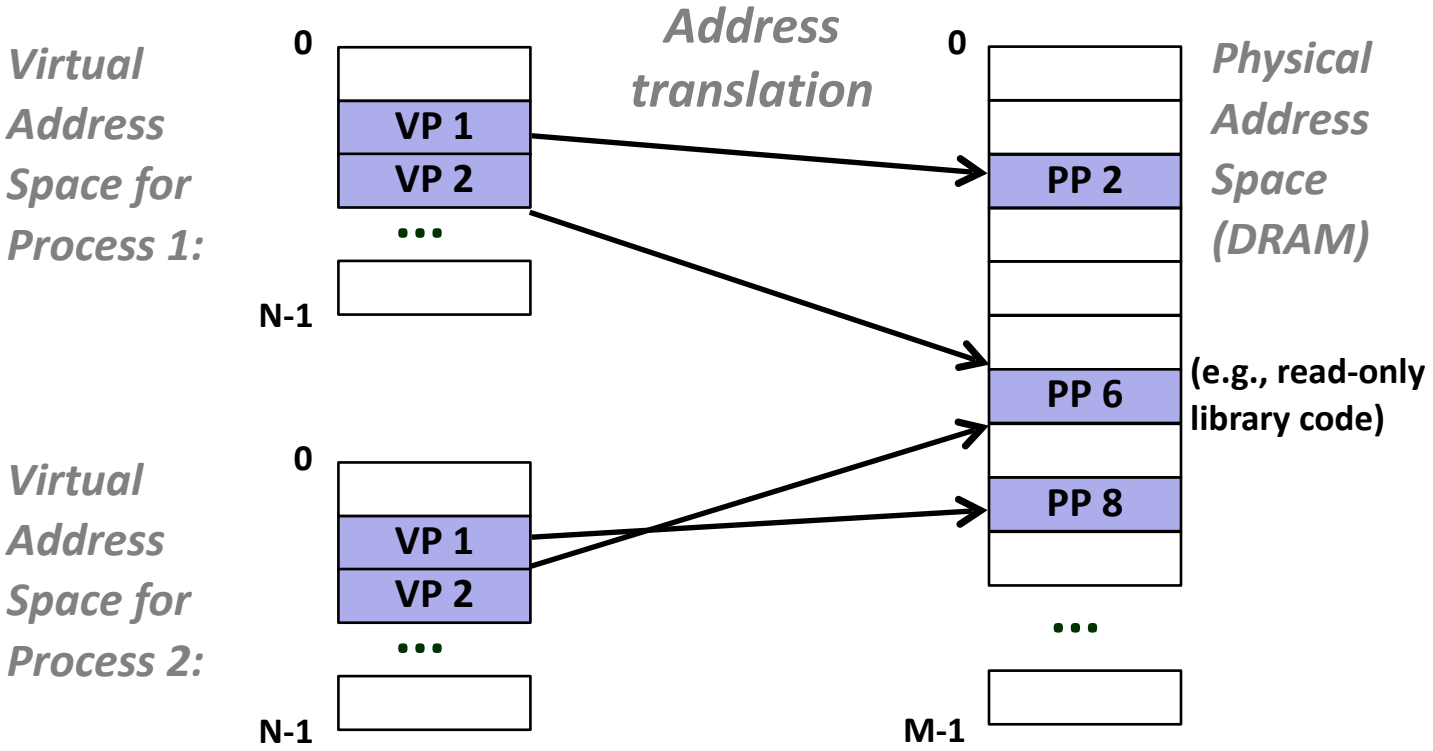
VM as a Tool for Memory Management

- 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 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 (read/write/executable/ accessible only in supervisor mode?)
- Remember buffer overflow attack?