CSE261: Computer Architecture

18. Memory Hierarchy (4): Virtual Memory #1

Seongil Wi

Department of Computer Science and Engineering

Notification: Final Exam

- Date: Dec. 19 (Thursday)
- Class Time (1h 15m), Closed book

- T/F problems + Computation problems + Descriptive problems

- Scope: All the material learned *after the midterm*

Notification: HW3

- Implementing a cache simulator
- Due: Dec 15, 11:59 PM

- Since the final exam period is approaching, it is recommended to complete this assignment *as soon as possible*

Where Are We?

Register 1

Register 2

On-chip cache (L1 cache)

Processor (CPU) (e.g., Intel I7)

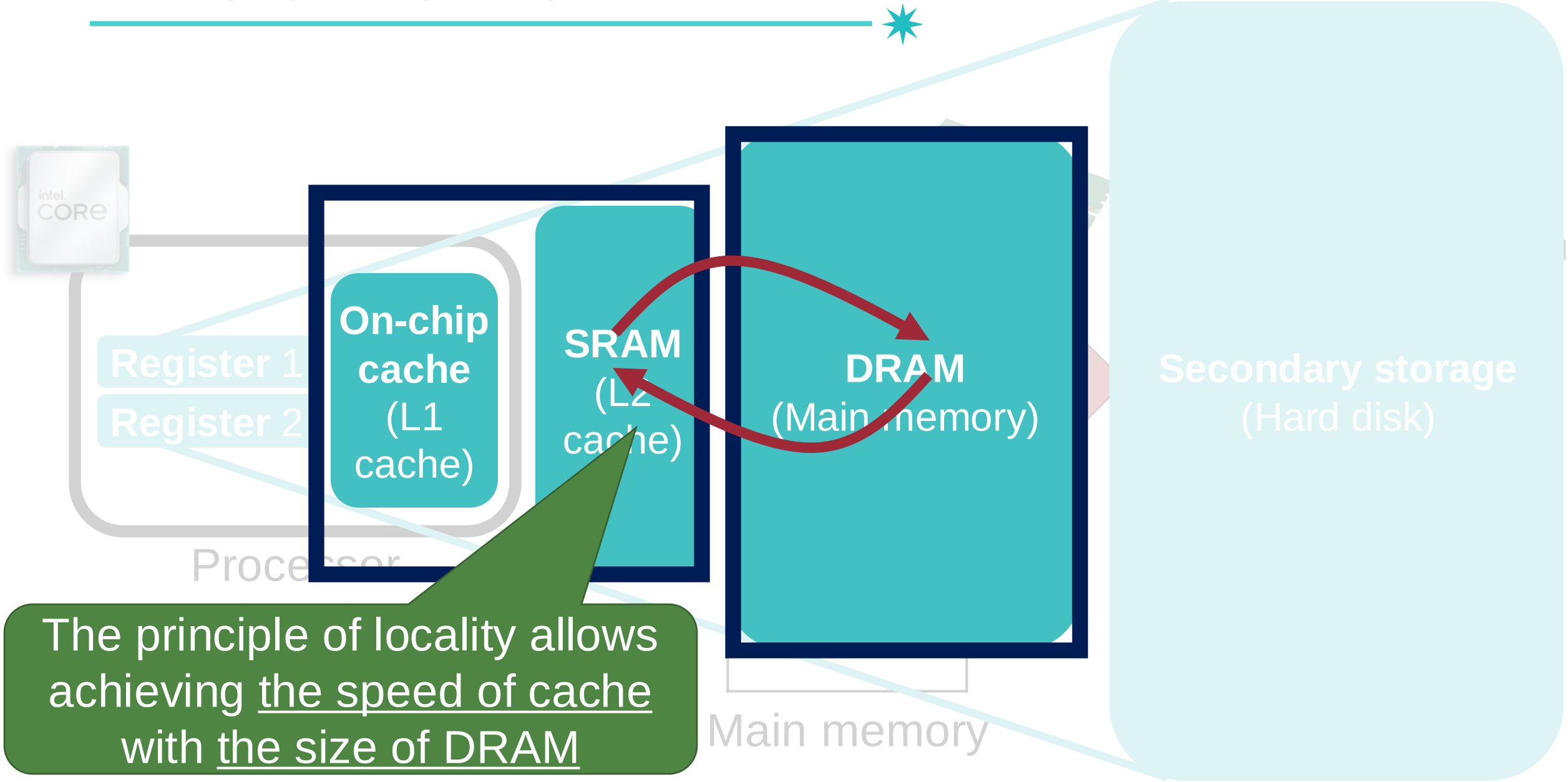
SRAM (L2 cache)

DRAM (Main memory)

Main memory

Secondary storage (Hard disk)

Where Are We?


Register 1
Register 2
Processor
On-chip cache (L1 cache)
SRAM (L2 cache)
DRAM (Main memory)
Main memory
Secondary storage (Hard disk)
The principle of locality allows achieving the speed of cache with the size of DRAM

Today's Topic: DRAM/HDD

We want to give
the speed of DRAM with
the size of hard disk!

Register 1
Register 2

On-chip cache (L1 cache)

SRAM (L2 cache)

Processor (CPU) (e.g., Intel I7)

DRAM (Main memory)

Main memory

Secondary storage (Hard disk)

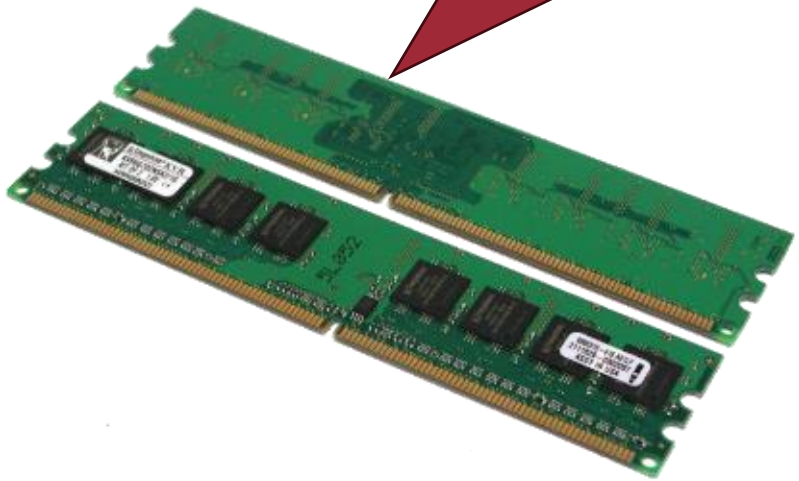
Introduction to Virtual Memory

In the Physical World, We Have Limited-sized Memory


```
lw $1, 4($3)
```

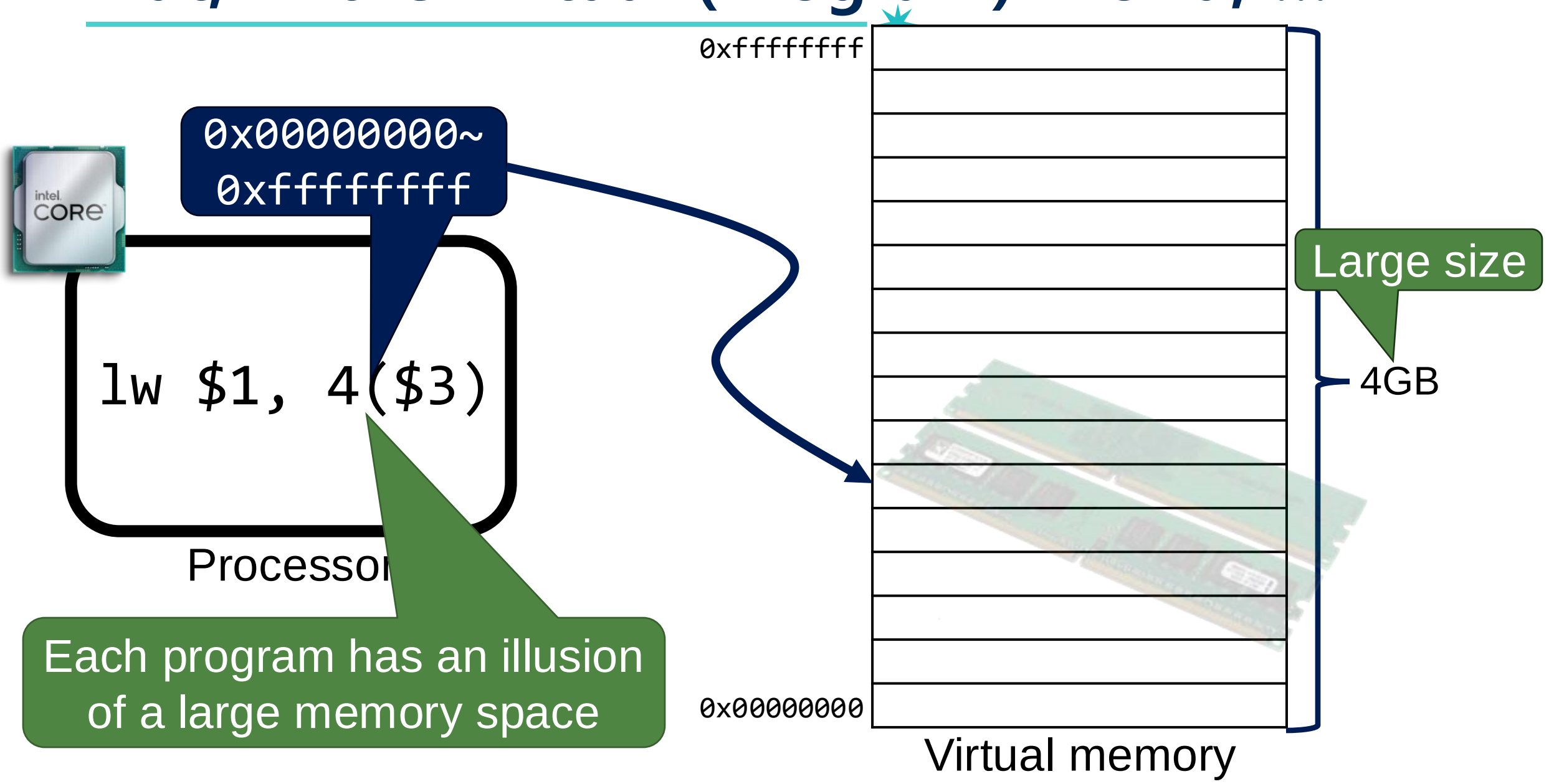
Processor

Very small size
(e.g., 256MB)



Physical memory

But, In the Virtual (Program) World, …



Virtual Memory

- Give programmers **an illusion of a large memory space** irrespective of actual capacity

Virtual Memory

- Give programmers **an illusion of a large memory space** irrespective of actual capacity

How do we give an illusion
despite having small-sized physical memory?

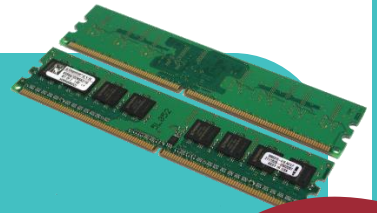
Memory hierarchy between RAM and HDD

Memory Hierarchy: DRAM/HDD

0x00000000~
0xffffffff

```
lw $1, 4($3)
```

Processor

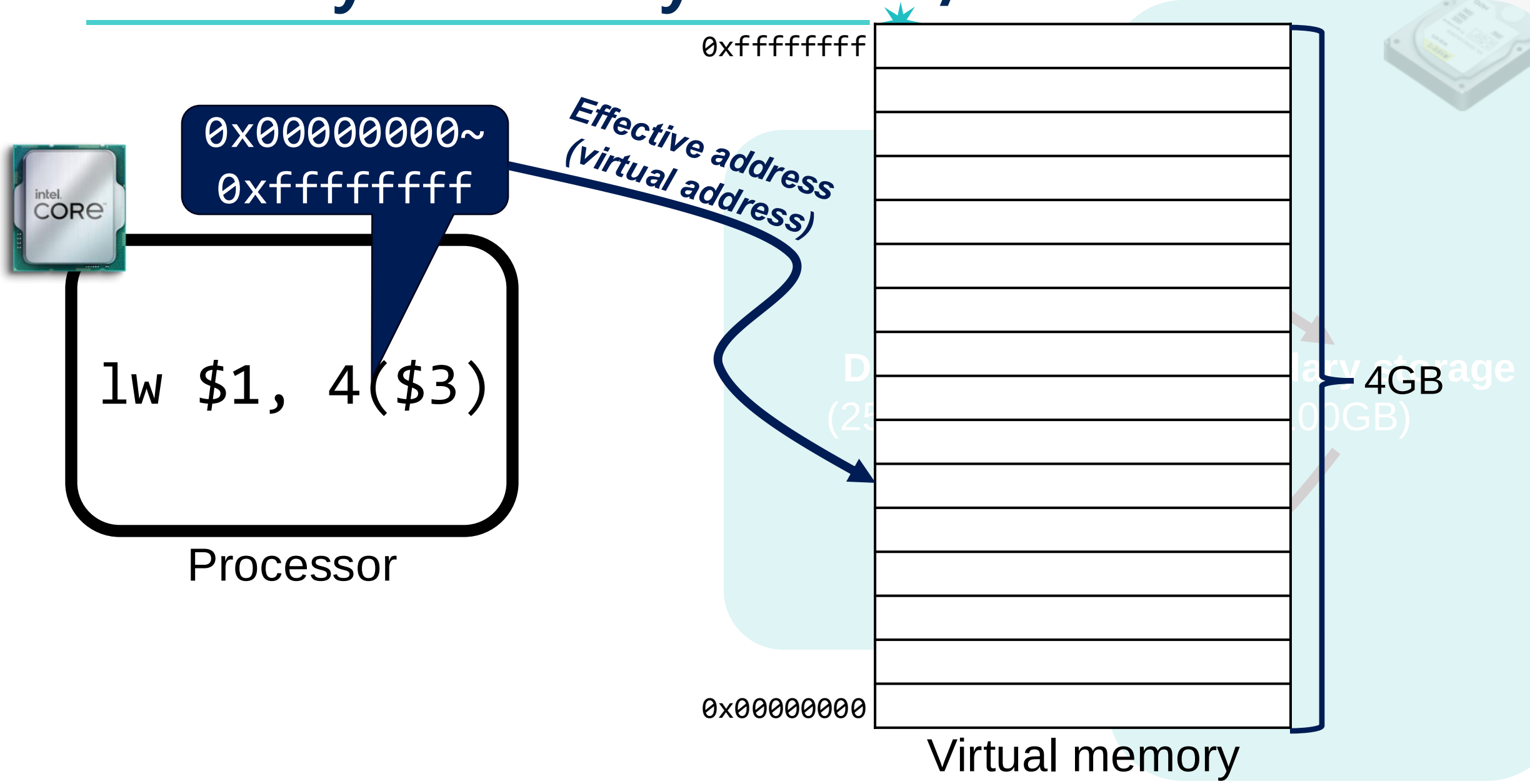


DRAM
(256MB)

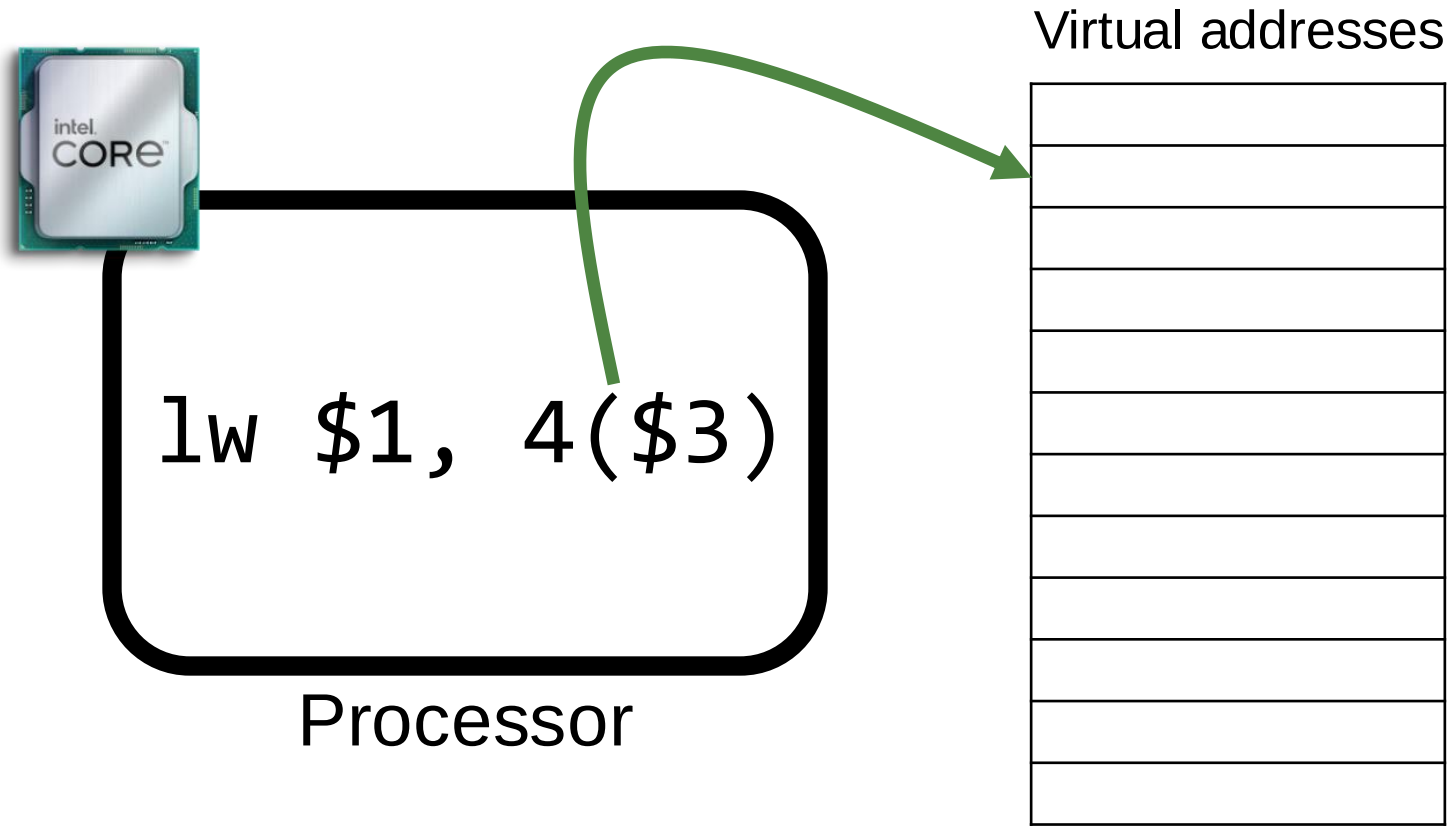
Secondary storage
(100GB)

Give the
speed of the DRAM with
the size of hard disk!

Memory Hierarchy: DRAM/HDD

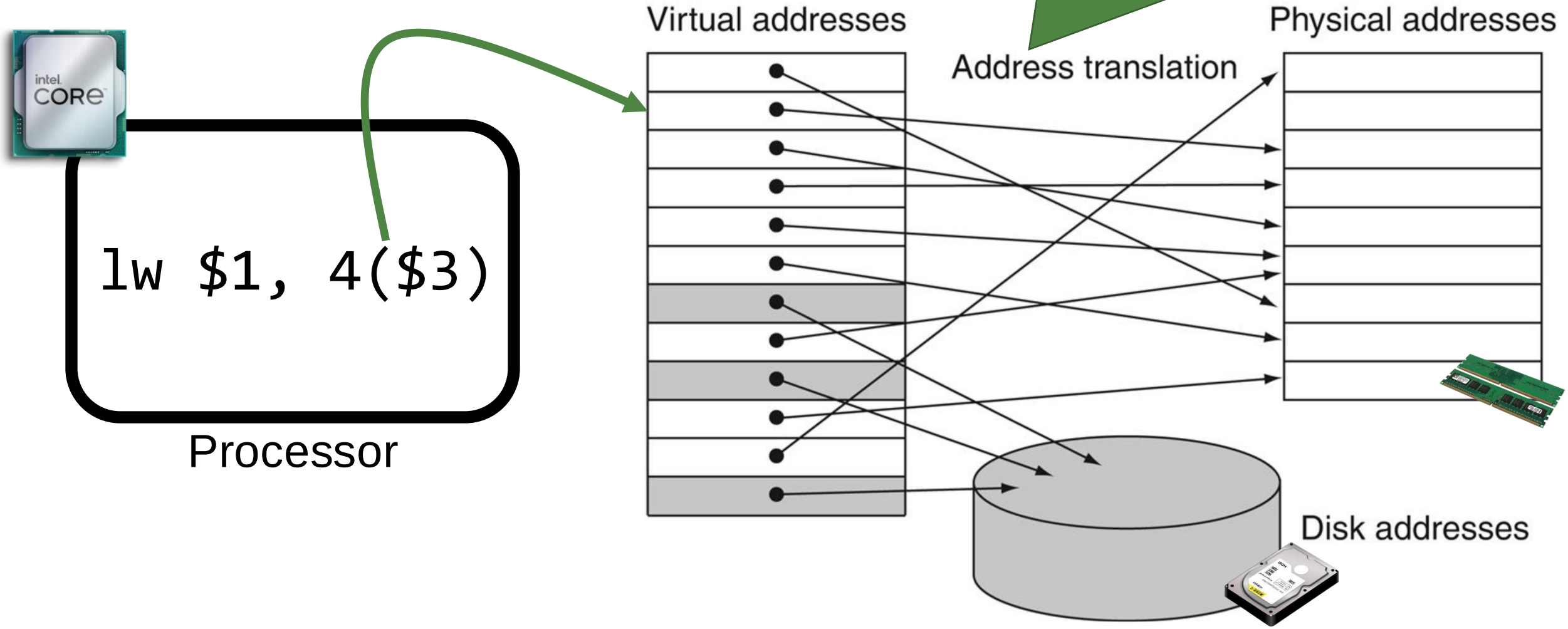


Memory Hierarchy: DRAM/HDD

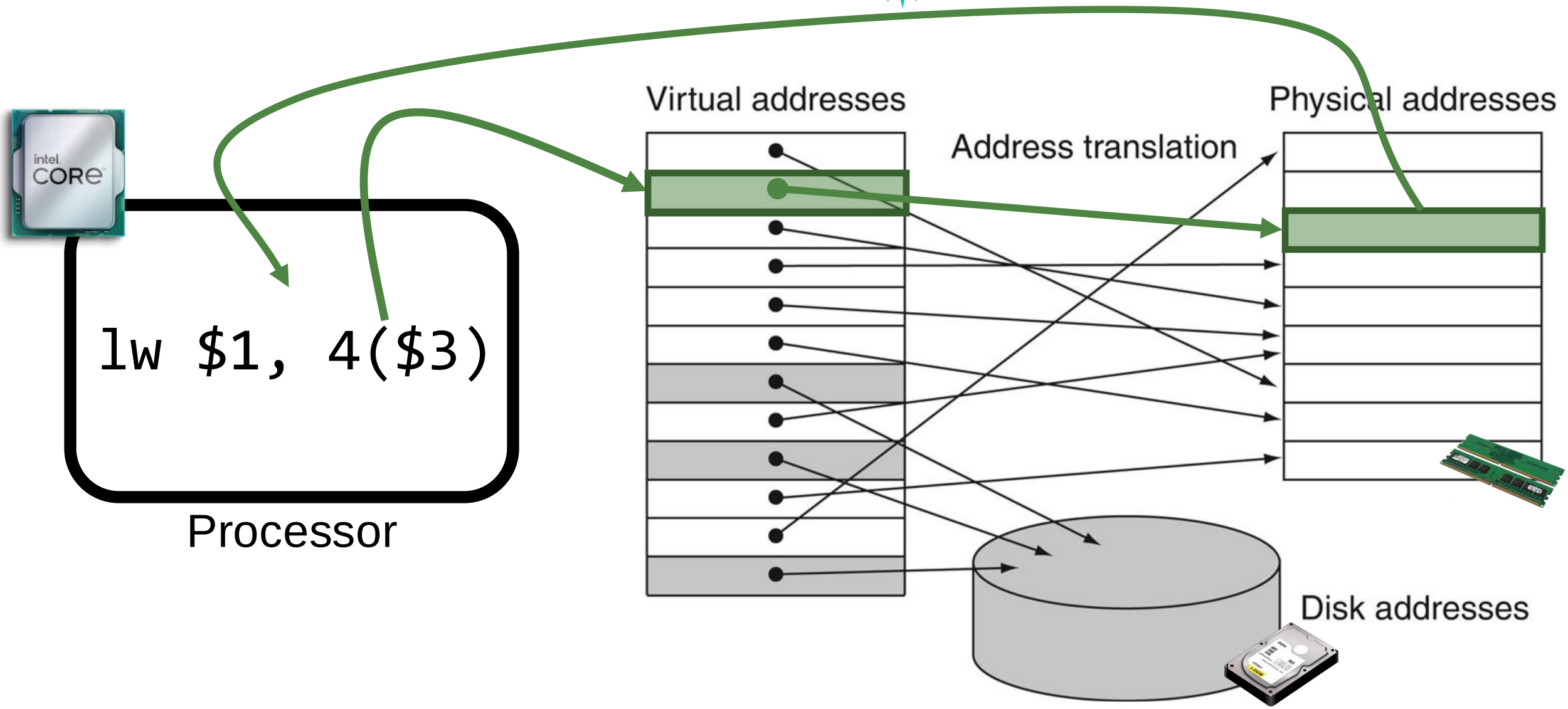


Memory Hi

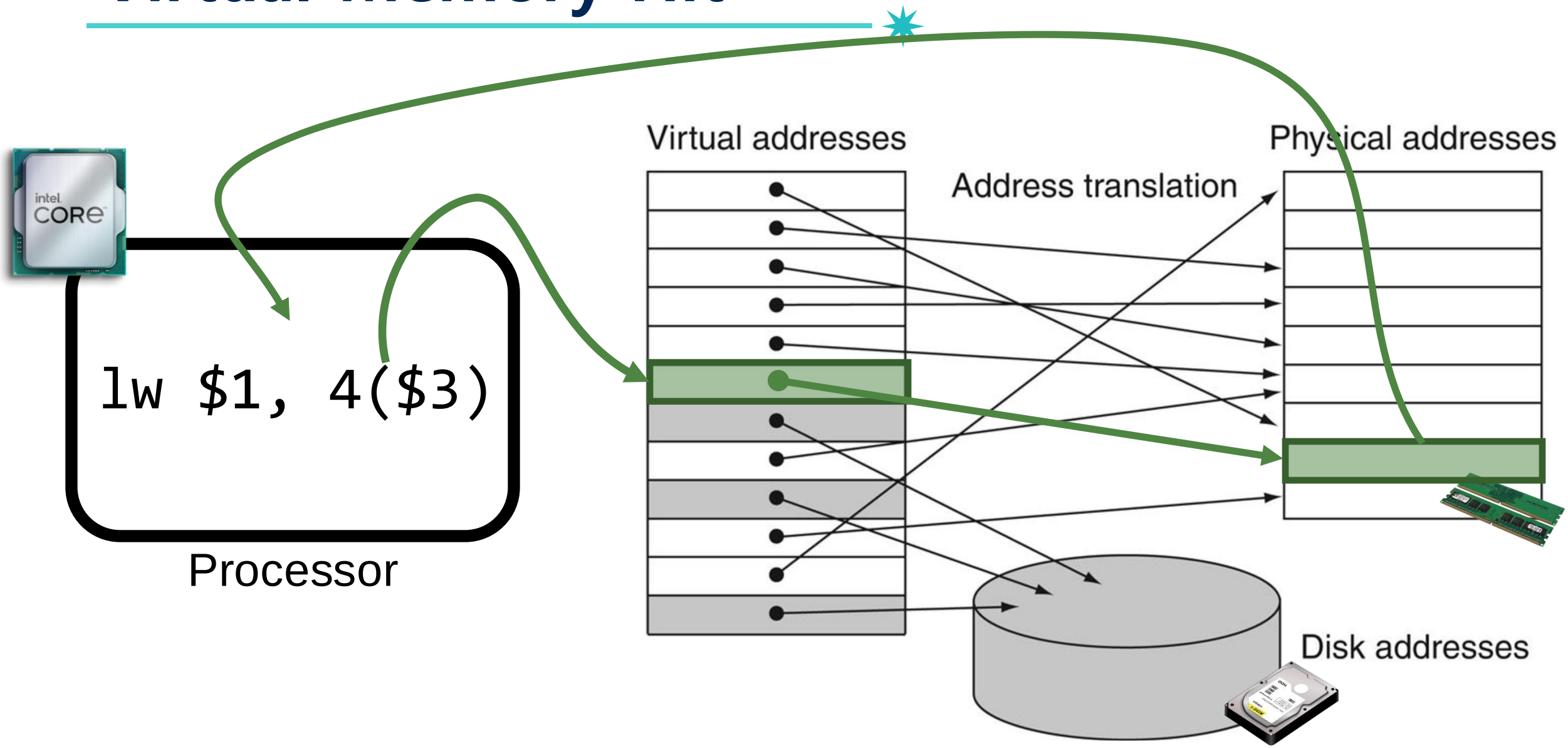
Address translation table (OS-managed table)
Virtual address → Physical address



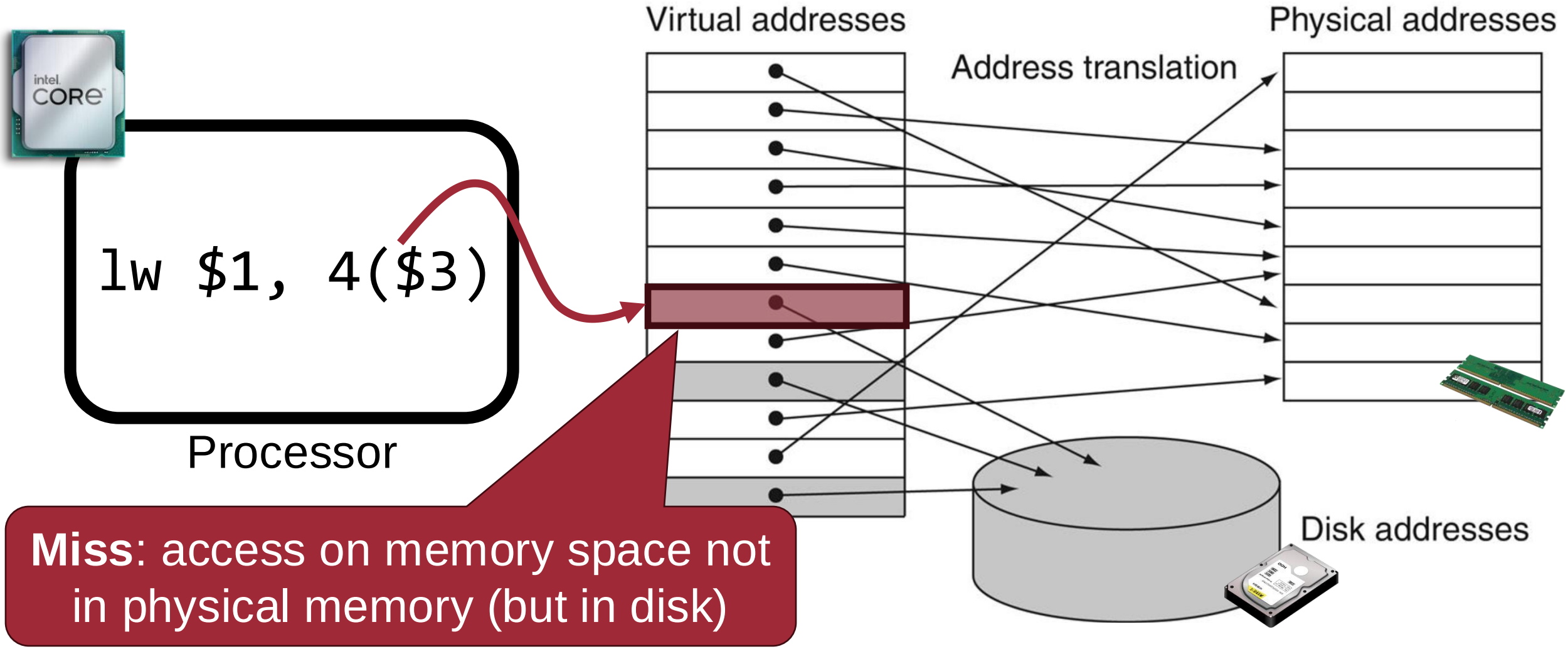
Virtual Memory Hit



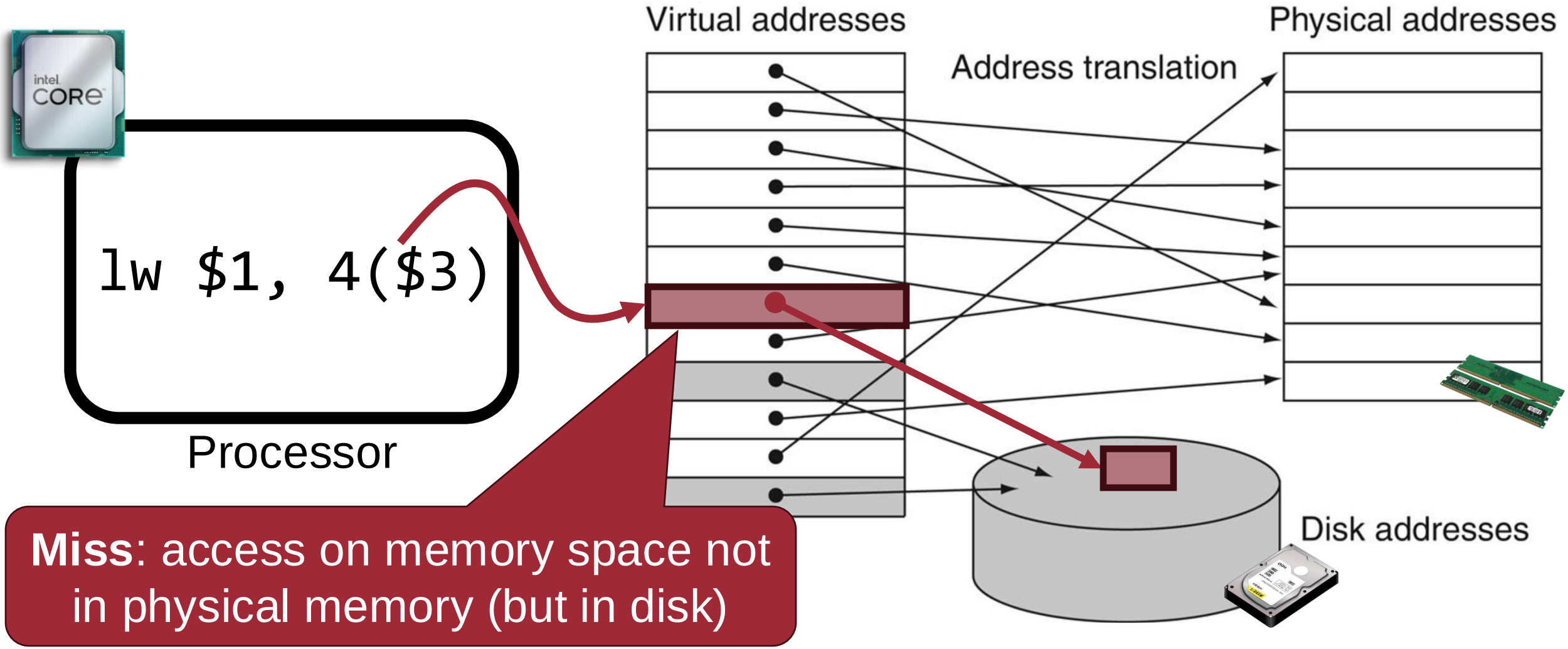
Virtual Memory Hit



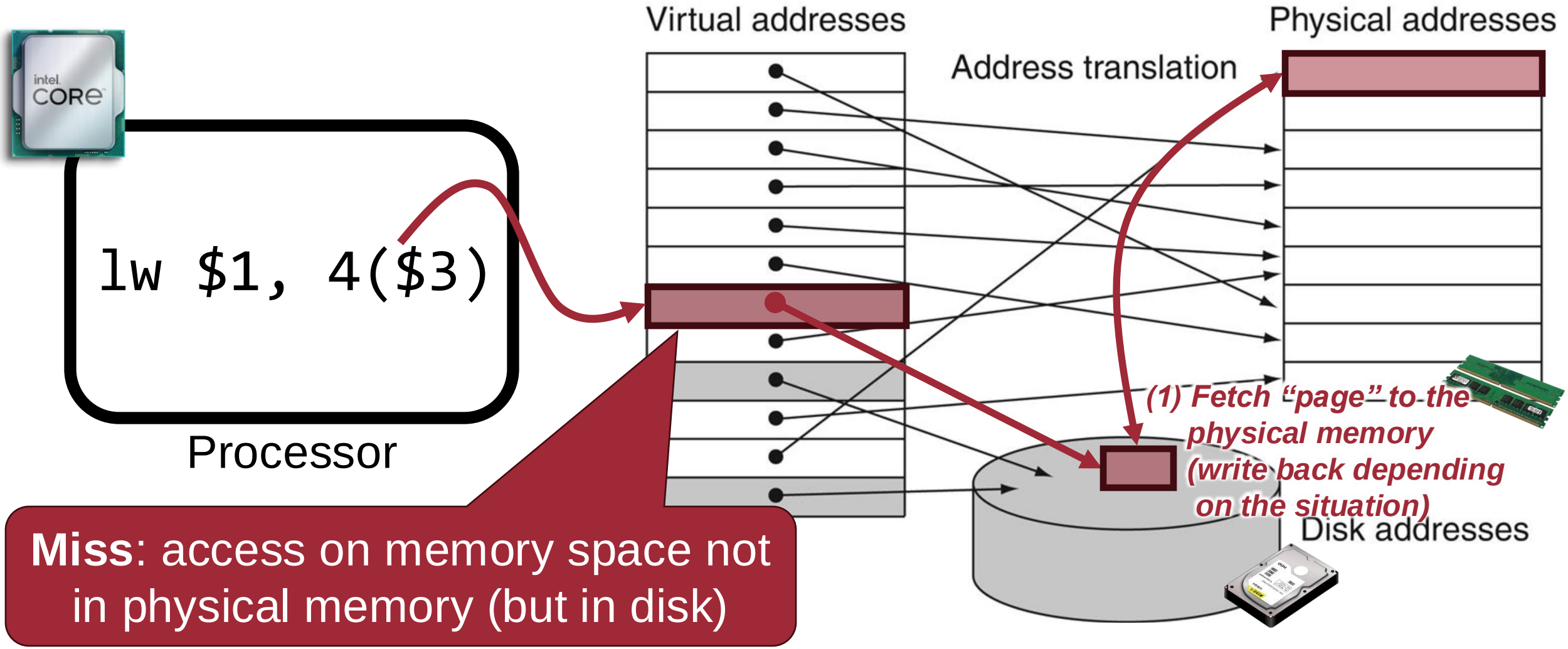
Virtual Memory Miss (a.k.a., Page Fault)



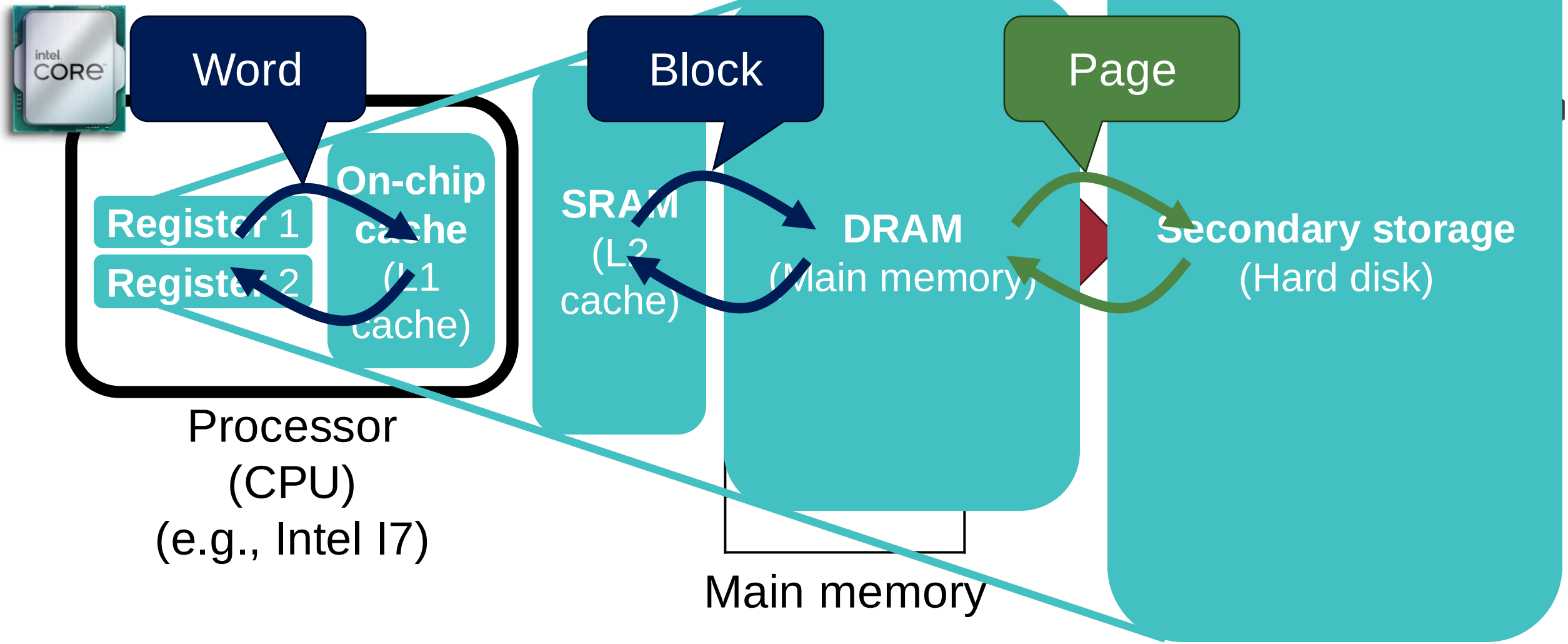
Virtual Memory Miss (a.k.a., Page Fault)



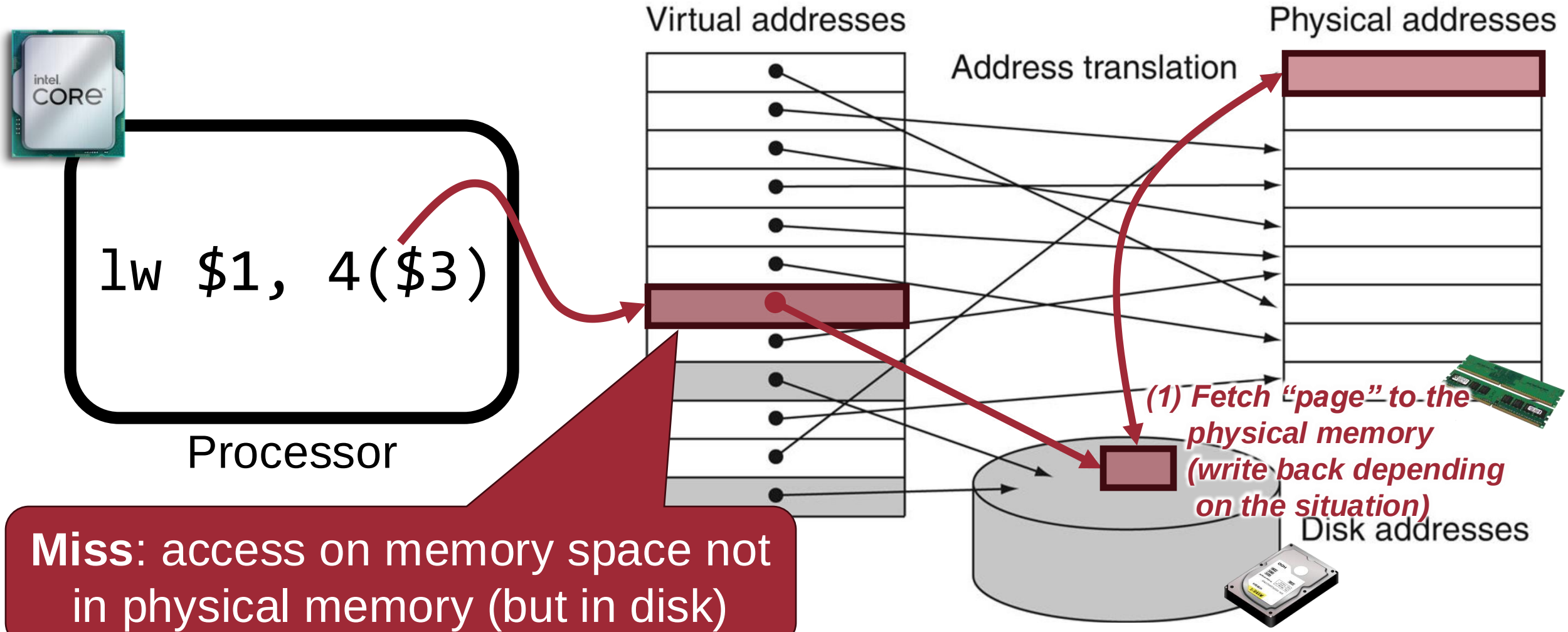
Virtual Memory Miss (a.k.a., Page Fault)



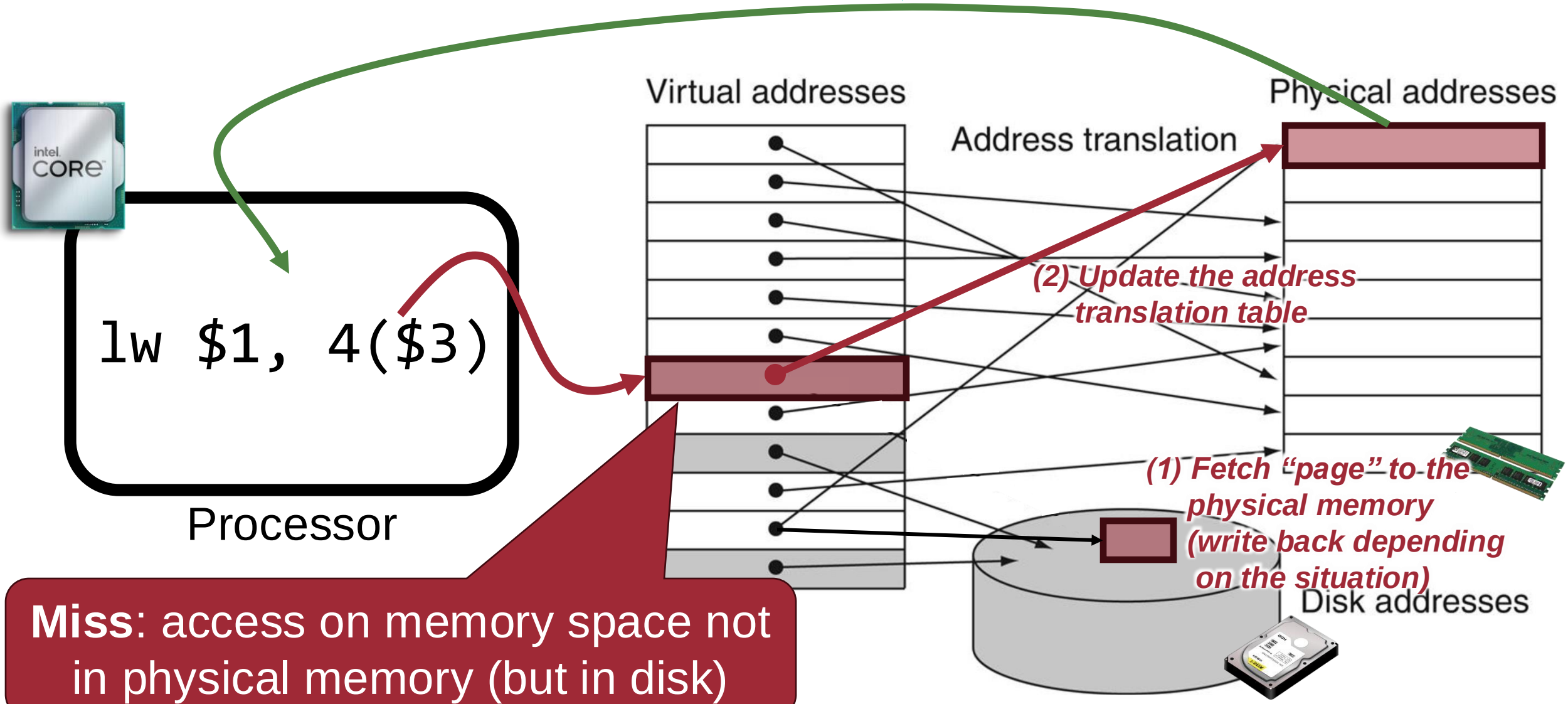
Unit of Copying: Page



Virtual Memory Miss (a.k.a., Page Fault)



Virtual Memory Miss (a.k.a., Page Fault)

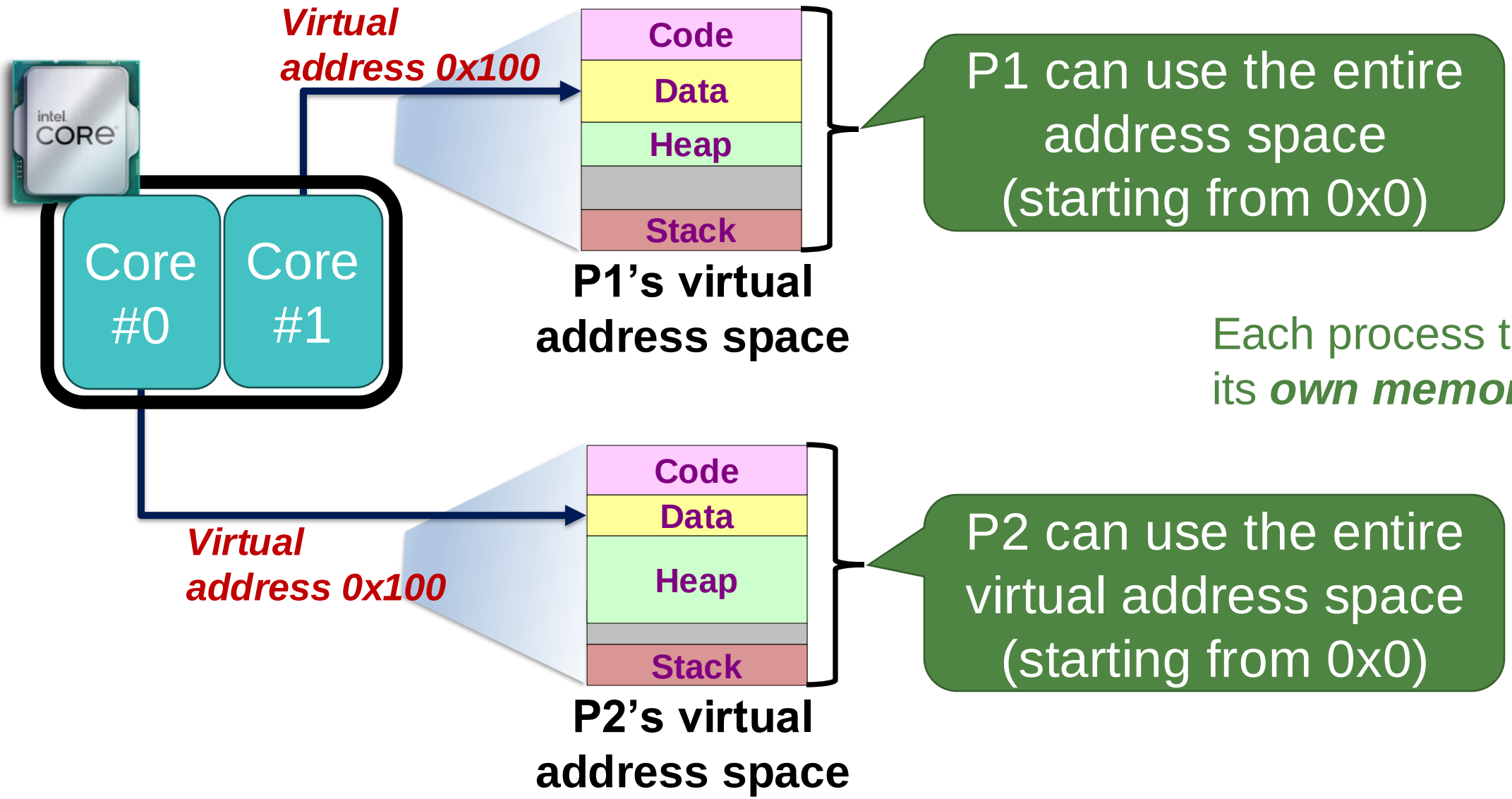


Virtual Memory

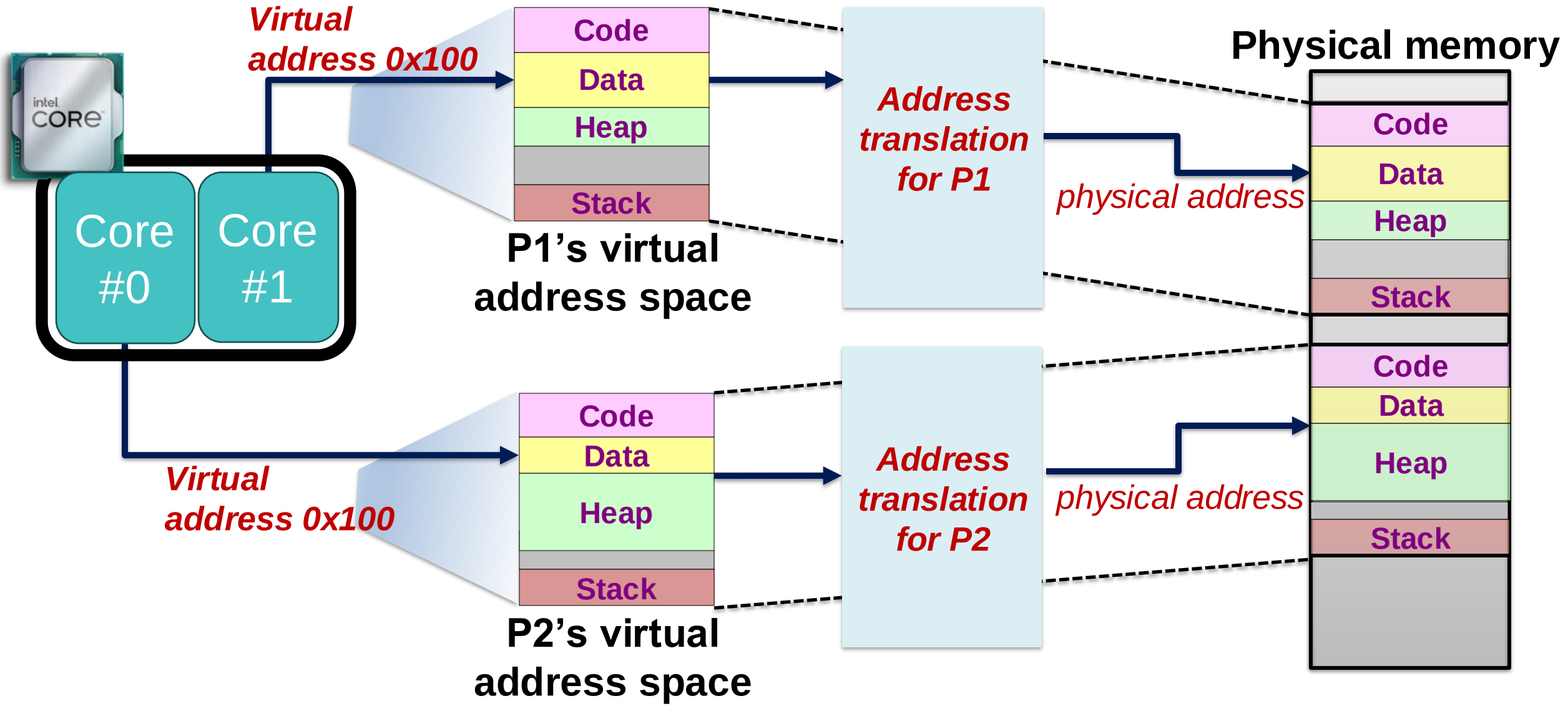
- Give programmers **an illusion of a large memory space** irrespective of actual capacity
- Uses main memory as a “cache” for the hard disk

- Other benefits?
  - **Program simplification**: give each program the illusion that it has its *own private memory* (e.g., `0x00000000~0xffffffff`)

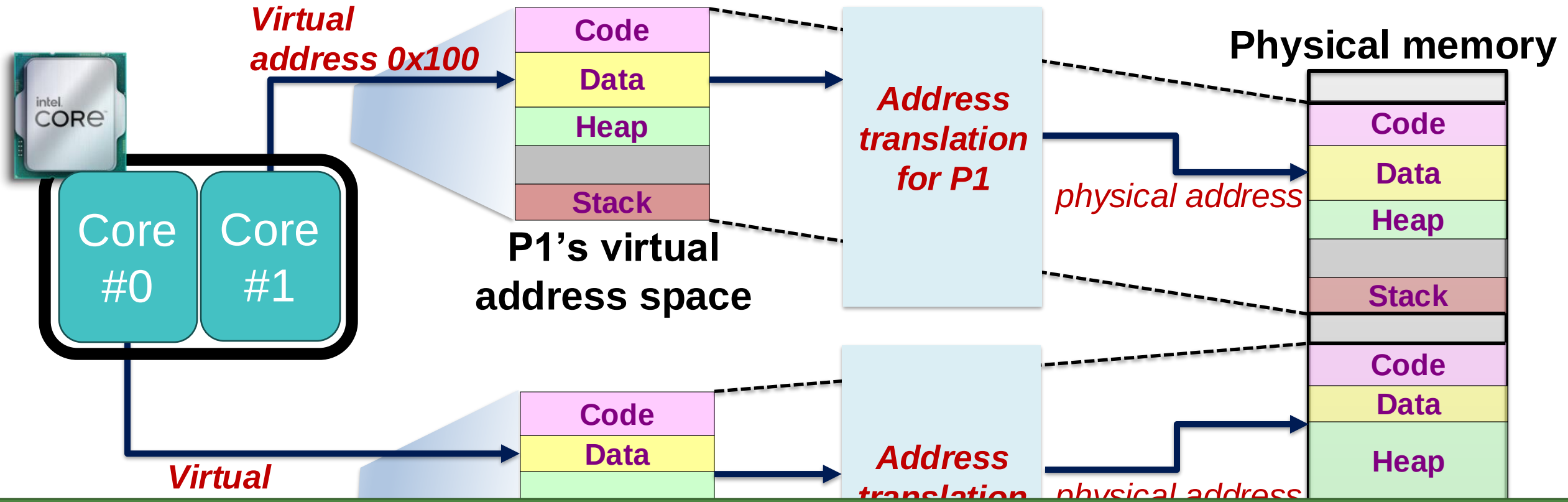
Program Simplification



Program Simplification



Program Simplification



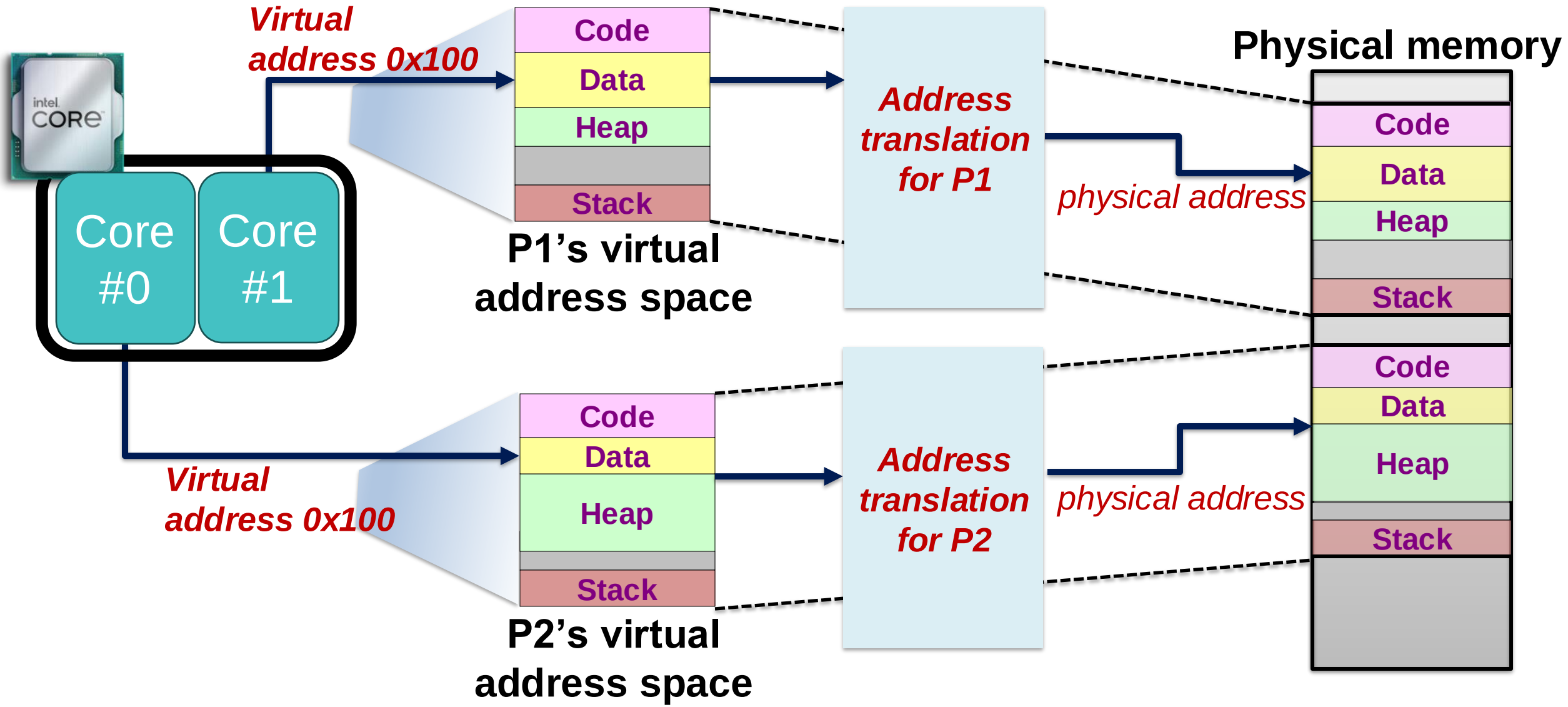
Programming and storage management ease

Virtual Memory

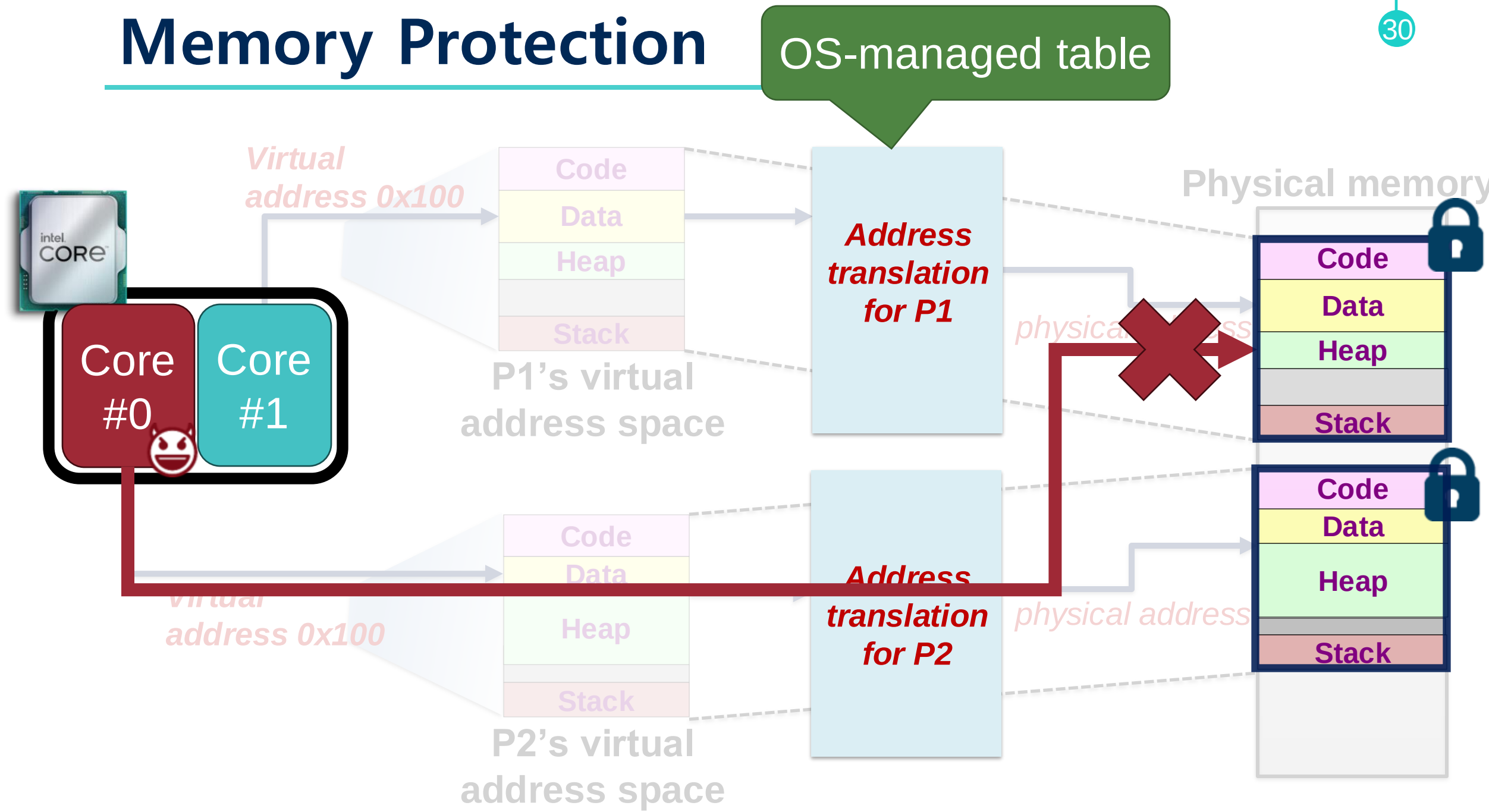
- Give programmers **an illusion of a large memory space** irrespective of actual capacity
- Uses main memory as a “cache” for the hard disk

- Other benefits?
  - **Program simplification**: give each program the illusion that it has its *own private memory* (e.g., `0x00000000~0xffffffff`)
  - **Security**: memory protection between processes

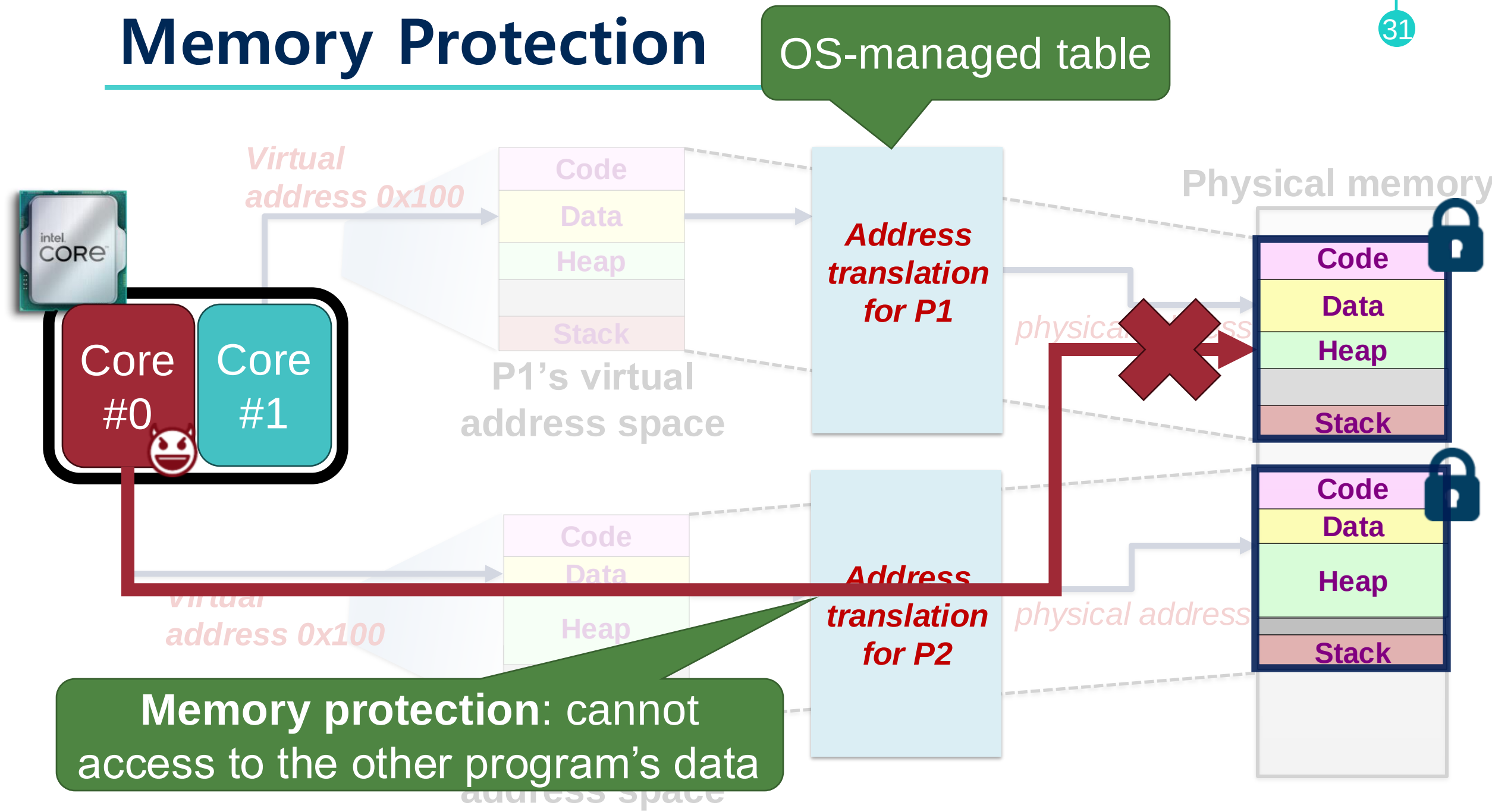
Memory Protection



Memory Protection



Memory Protection



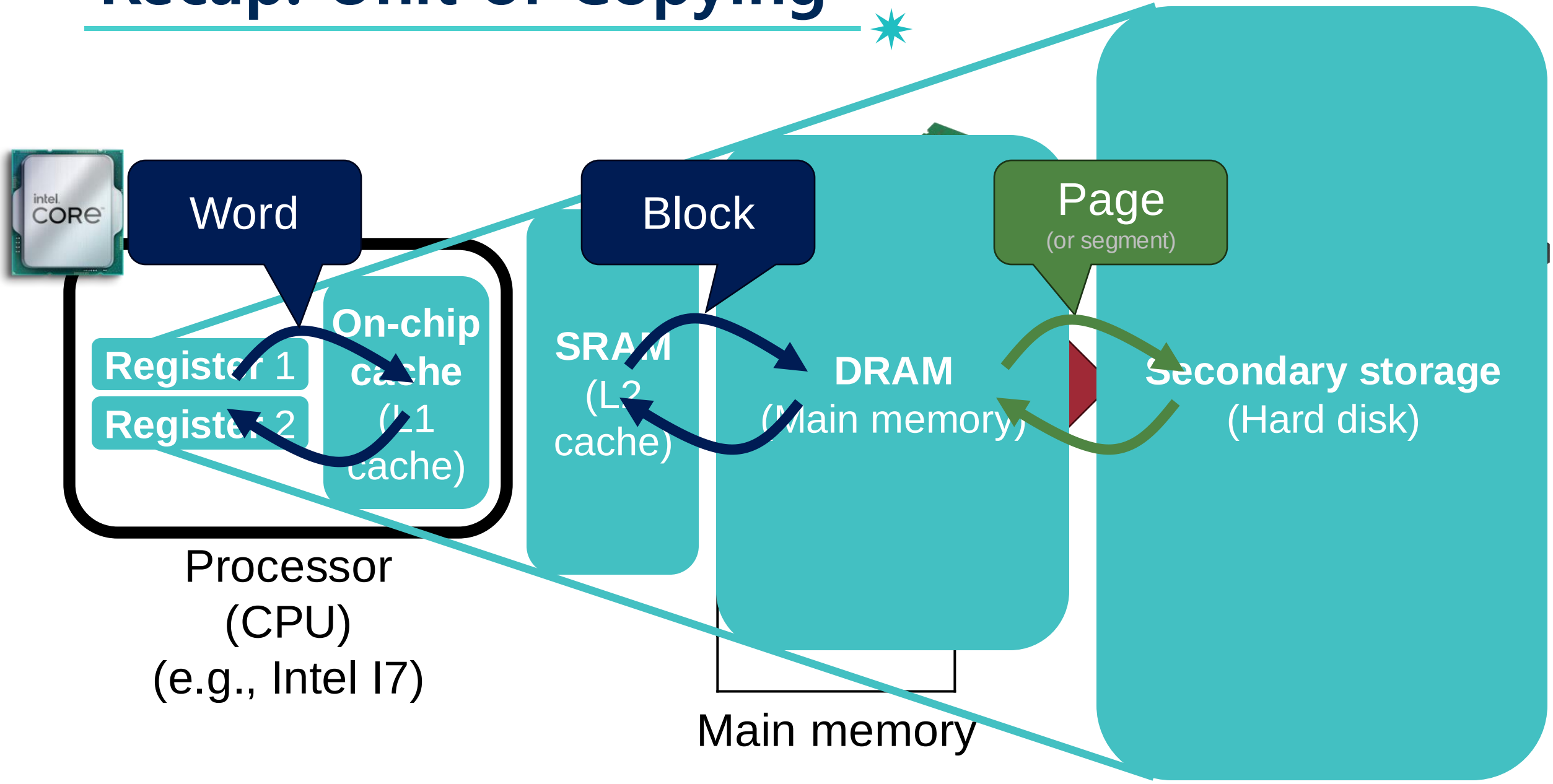
Virtual Memory Summary

- Idea: Give programmers **an illusion of a large memory space** irrespective of actual capacity
  - Programmers assume they have “infinite” amount of physical memory
  - Programmers do not need to worry about how to manage it (simplifies memory management for multi-processing system)

- CPU and OS cooperatively manage physical memory space to provide the illusion on behalf of users
  - Illusion is maintained for each independent process
  - Let OS to share memory, protect programs from each other

Unit of Copying: Page

Recap: Unit of Copying

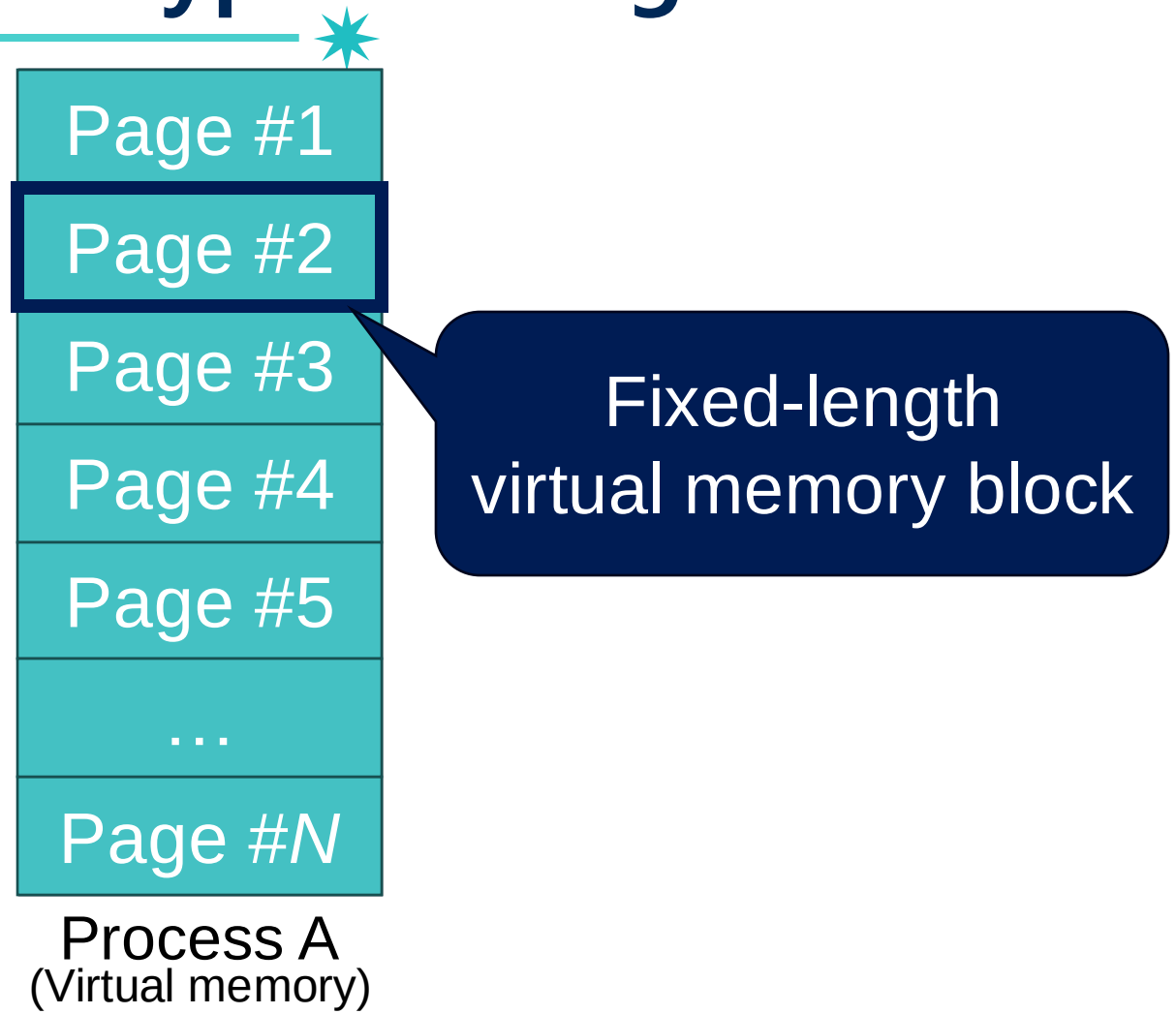


Virtual Memory Block Type #1: Page

- A block of virtual memory
- Unit of copying
  - between DRAM and Disk
- Fixed length
  - e.g., 4KB
  - Generally, 4KB ~ 4MB

Virtual Memory Block Type #1: Page

- A block of virtual memory
- Unit of copying
  - between DRAM and Disk
- Fixed length
  - e.g., 4KB
  - Generally, 4KB ~ 4MB



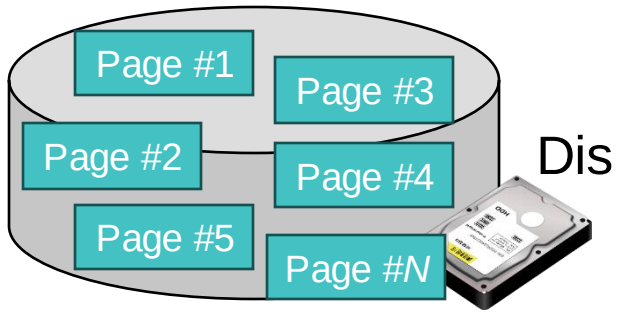
Virtual Memory Block Type #2: Segment

Segment length is variable!

Page

- A block of virtual memory
- Unit of copying
  - between DRAM and Disk
- Fixed length
  - e.g., 4KB
  - Generally, 4KB ~ 4MB

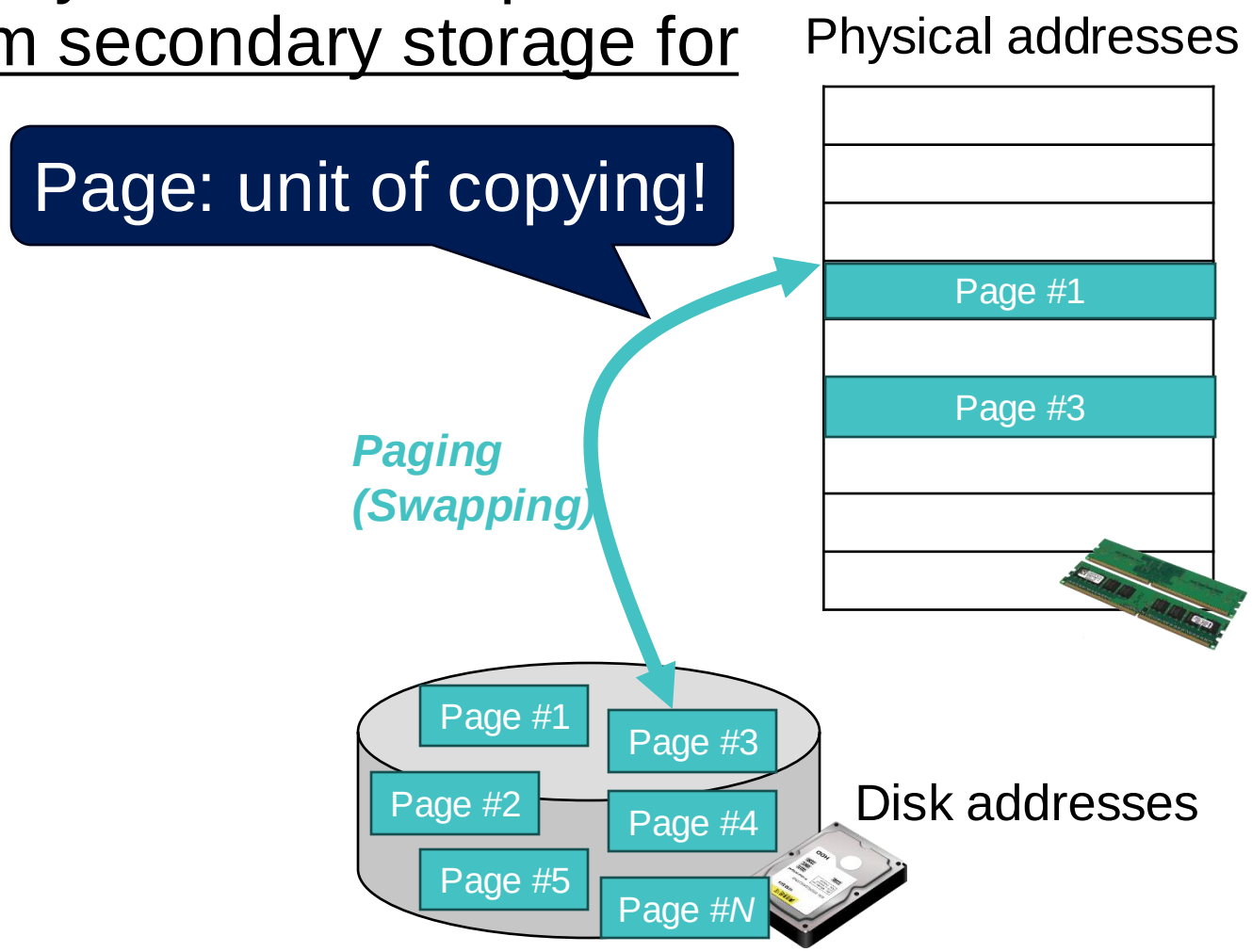
Process A
(Virtual memory)



Disk addresses

Memory Paging (a.k.a., Swapping)

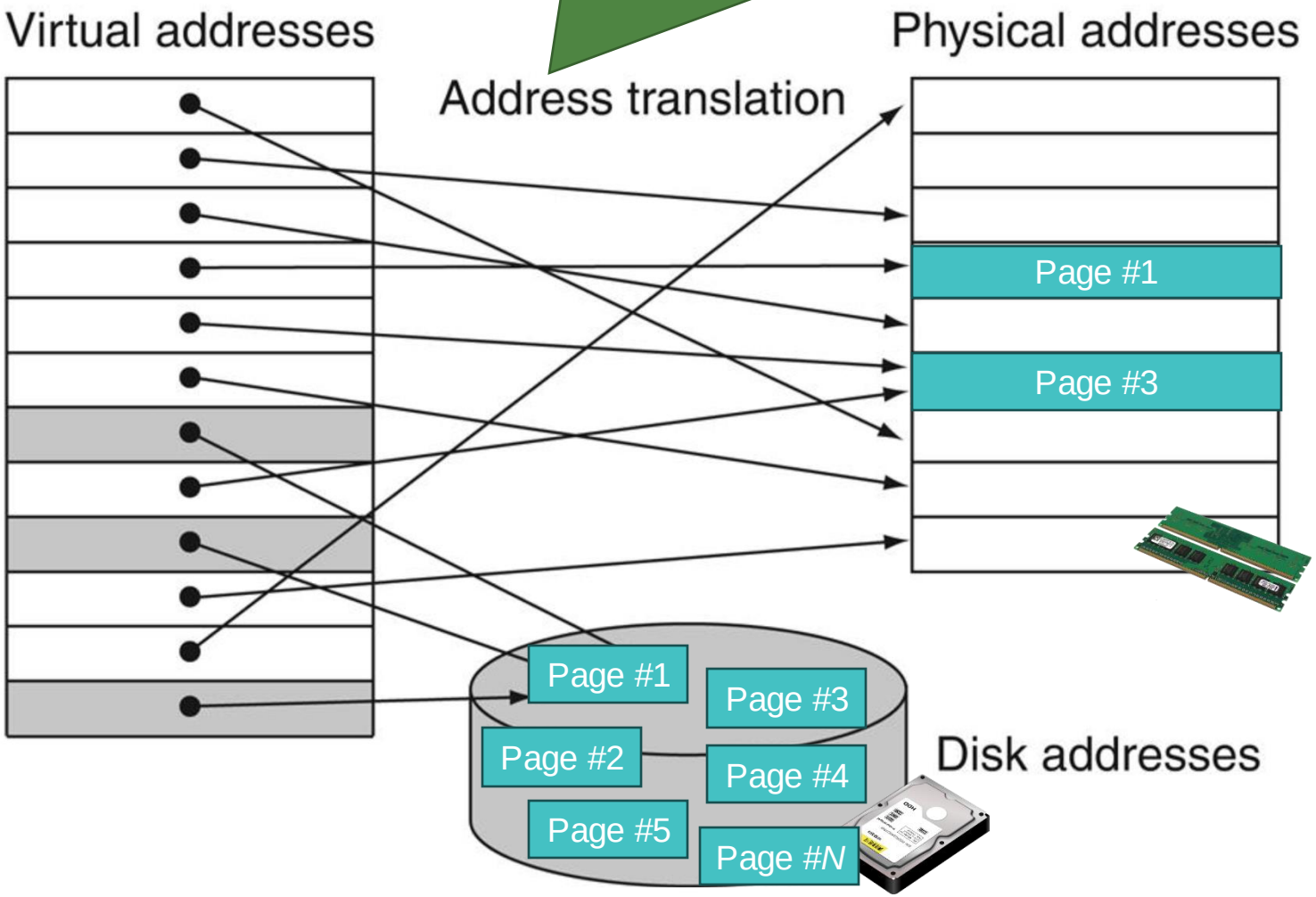
Memory management scheme by which a computer **stores and retrieves data** from secondary storage for use in main memory



Address Translation

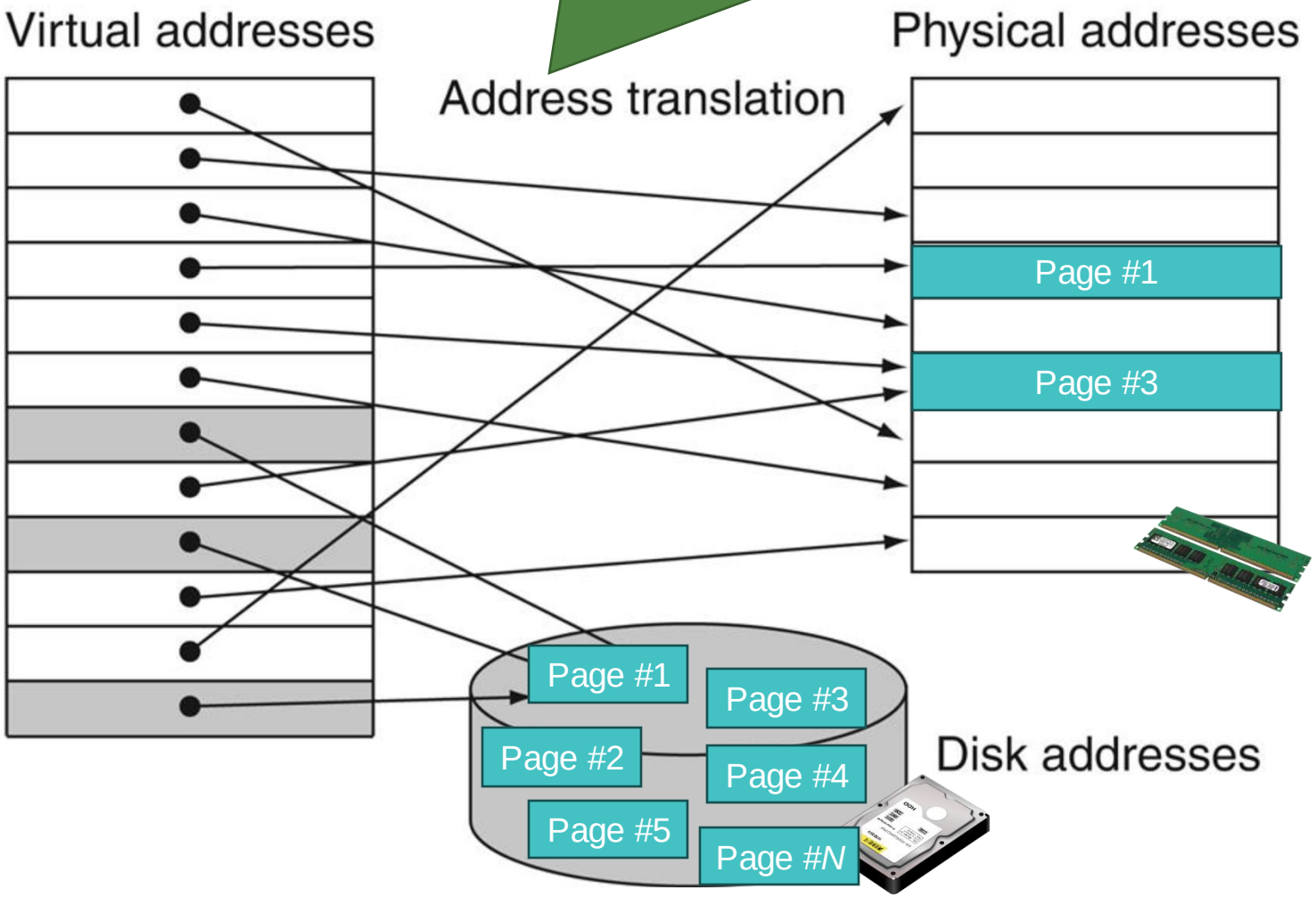
Memory Hi

Address translation table (OS-managed table)
Virtual address → Physical address



Memory Hi

"Page Table"

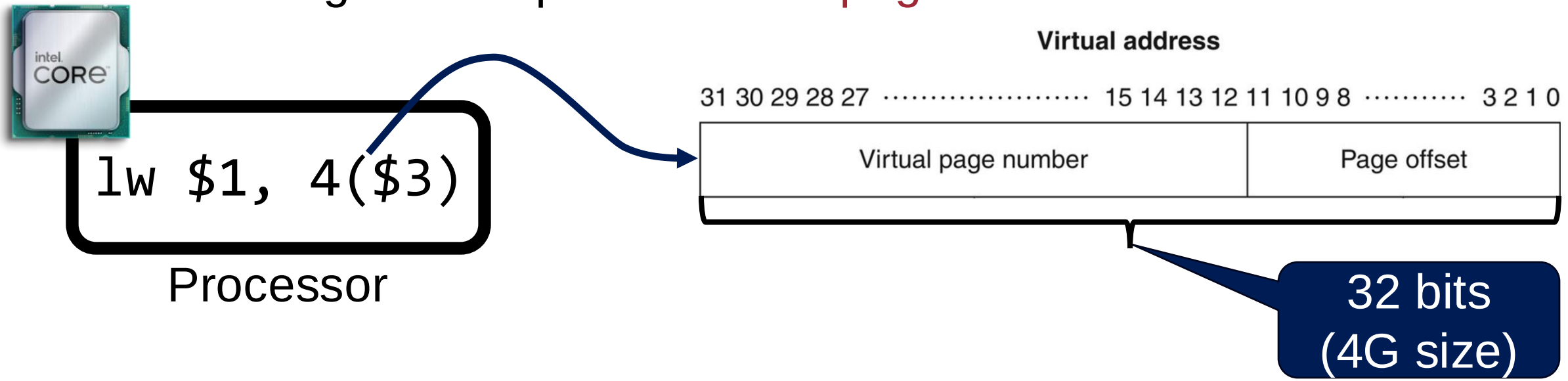


Page Table: Address Translation

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table

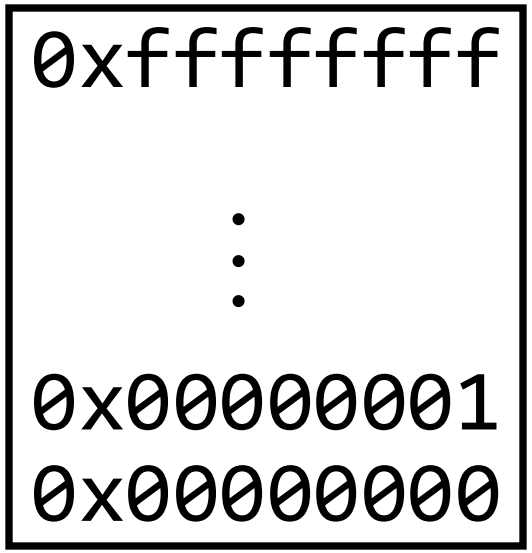
Page Table: Address Translation

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table



Recap: the range of memory addresses a 32-bit machine (e.g., MIPS) can access?

Representable range $= 0 \sim 2^{32}-1$

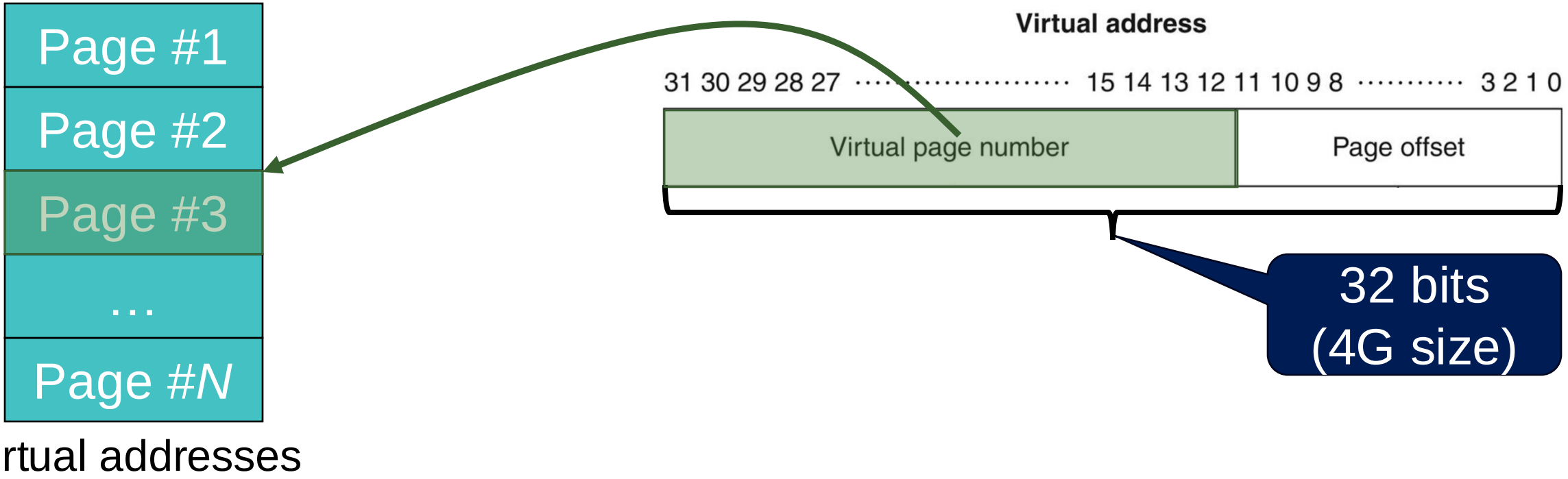


Q. What is the maximum memory capacity a 32-bit machine can handle?

2^{32} bytes = 4GB

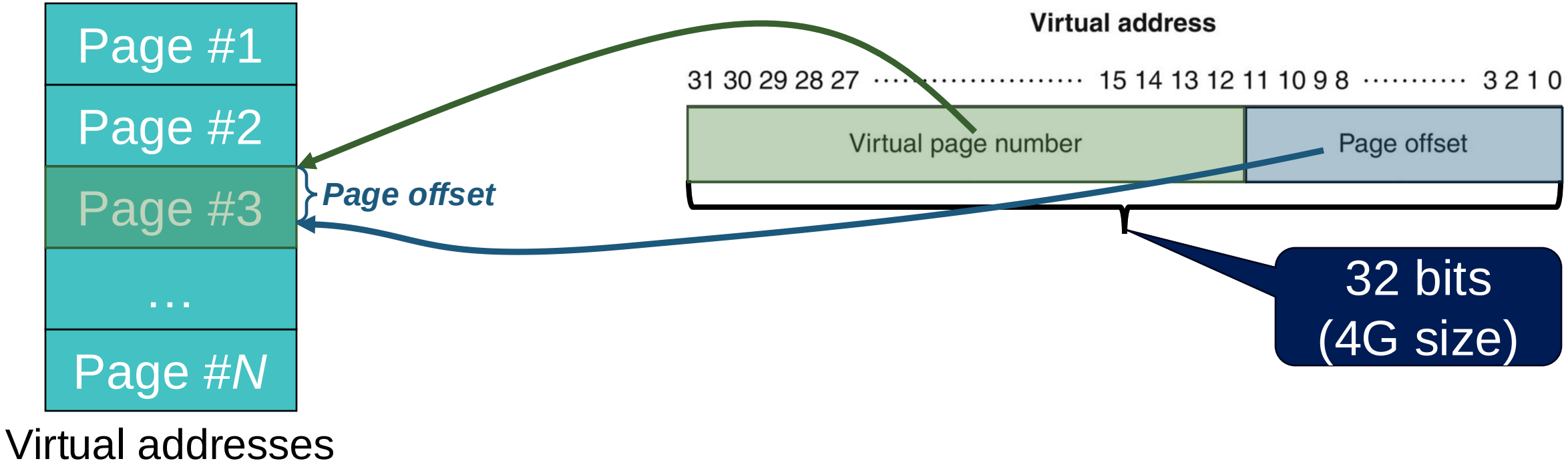
Programmer's Point of View

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table



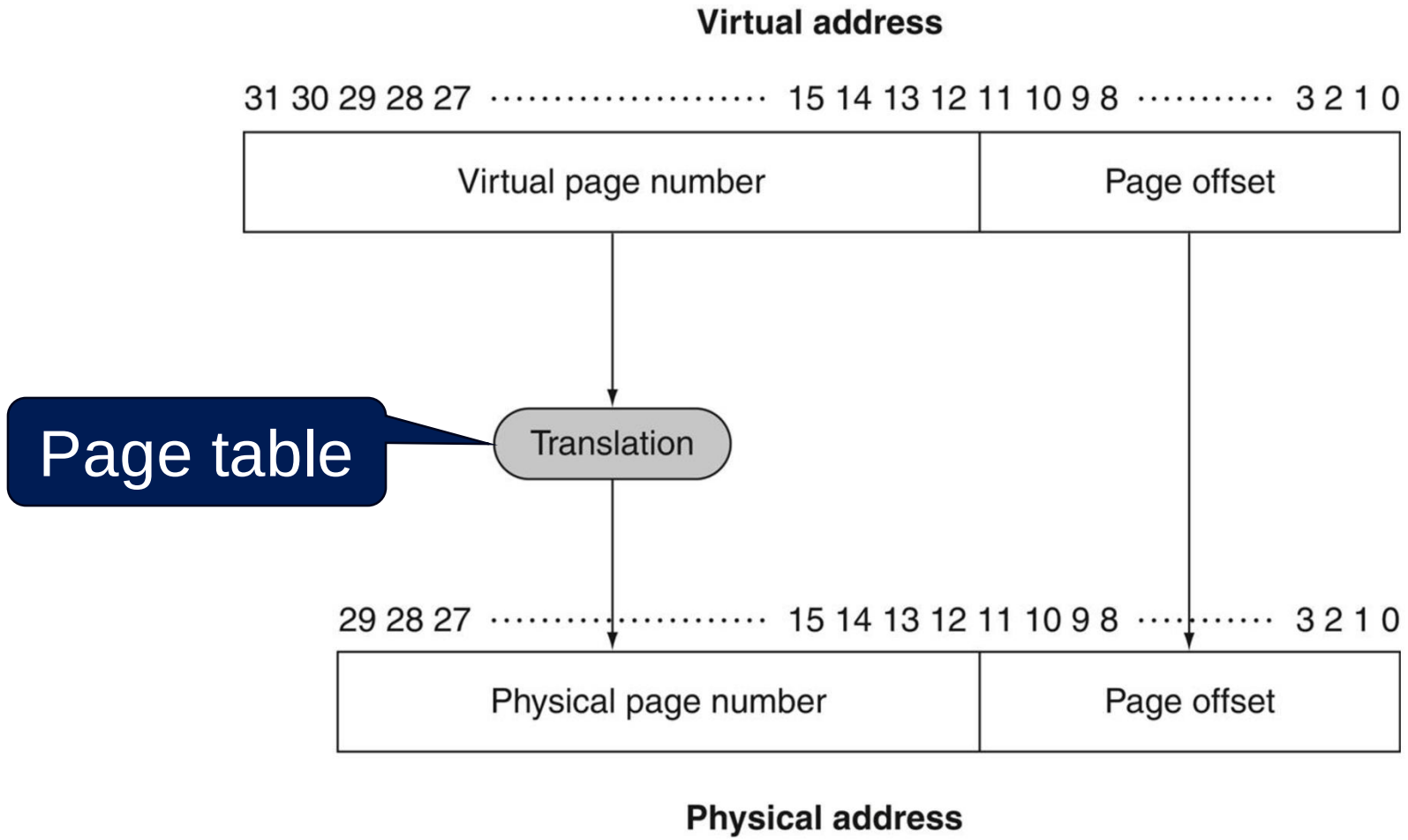
Programmer's Point of View

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table



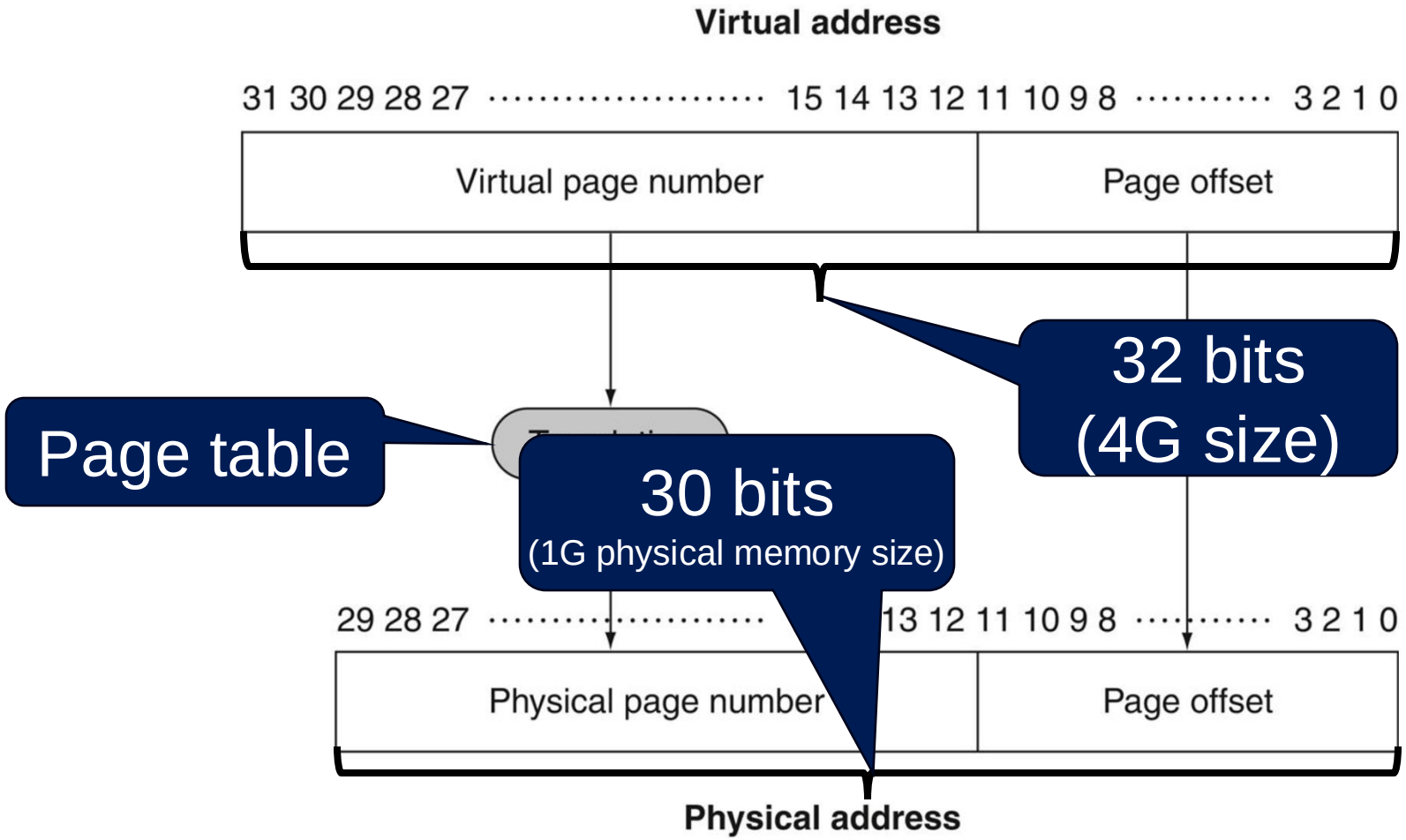
Page Table: Address Translation

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table



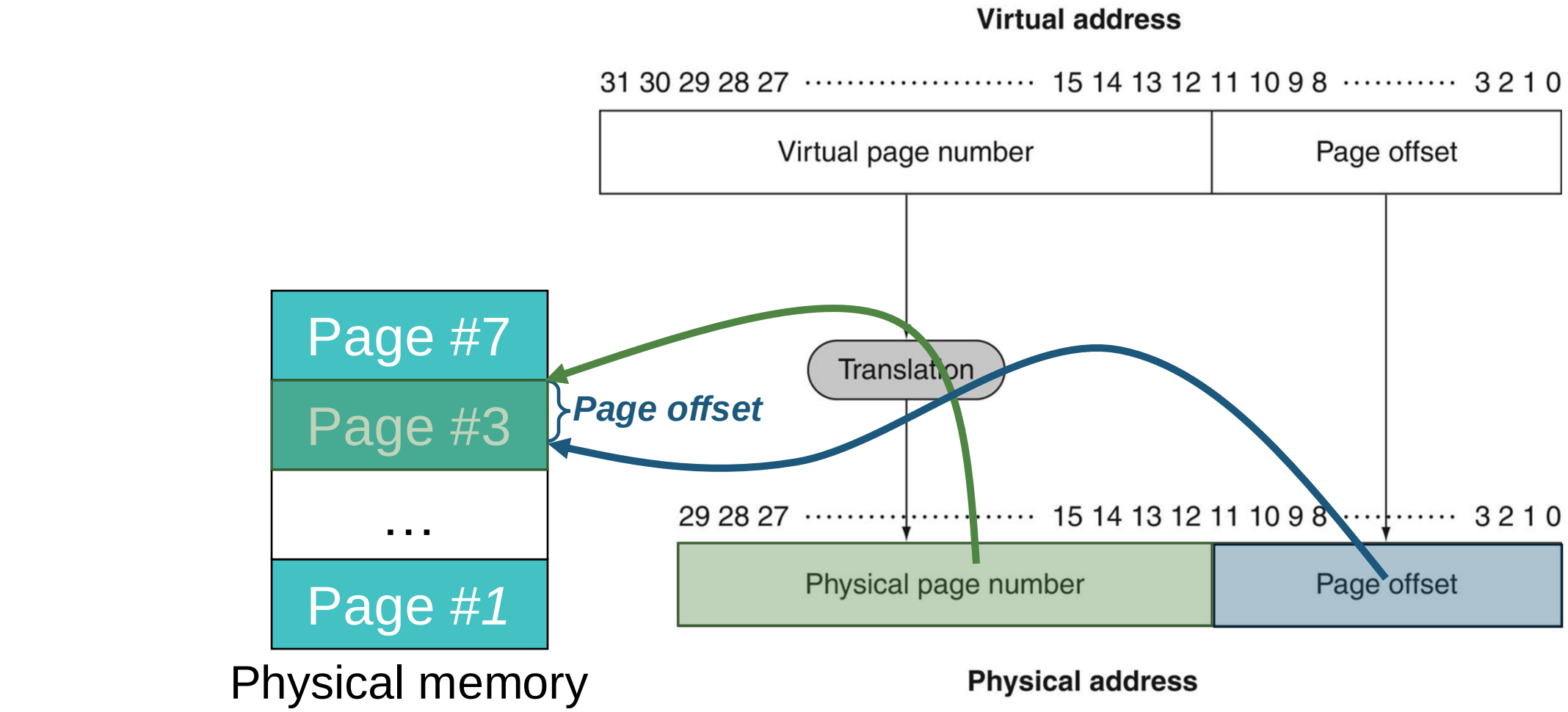
Page Table: Address Translation

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table



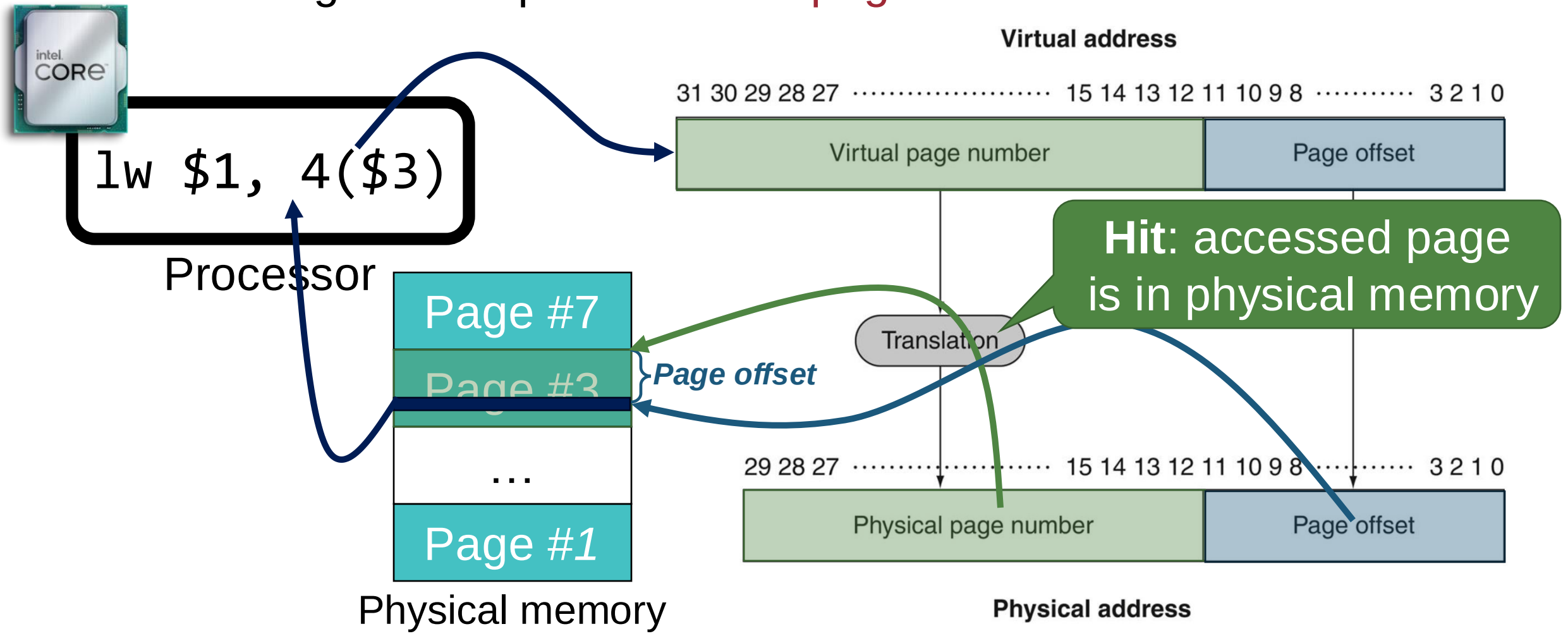
Page Table: Address Translation

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table



Address Translation Flow (Hit Case)

CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table

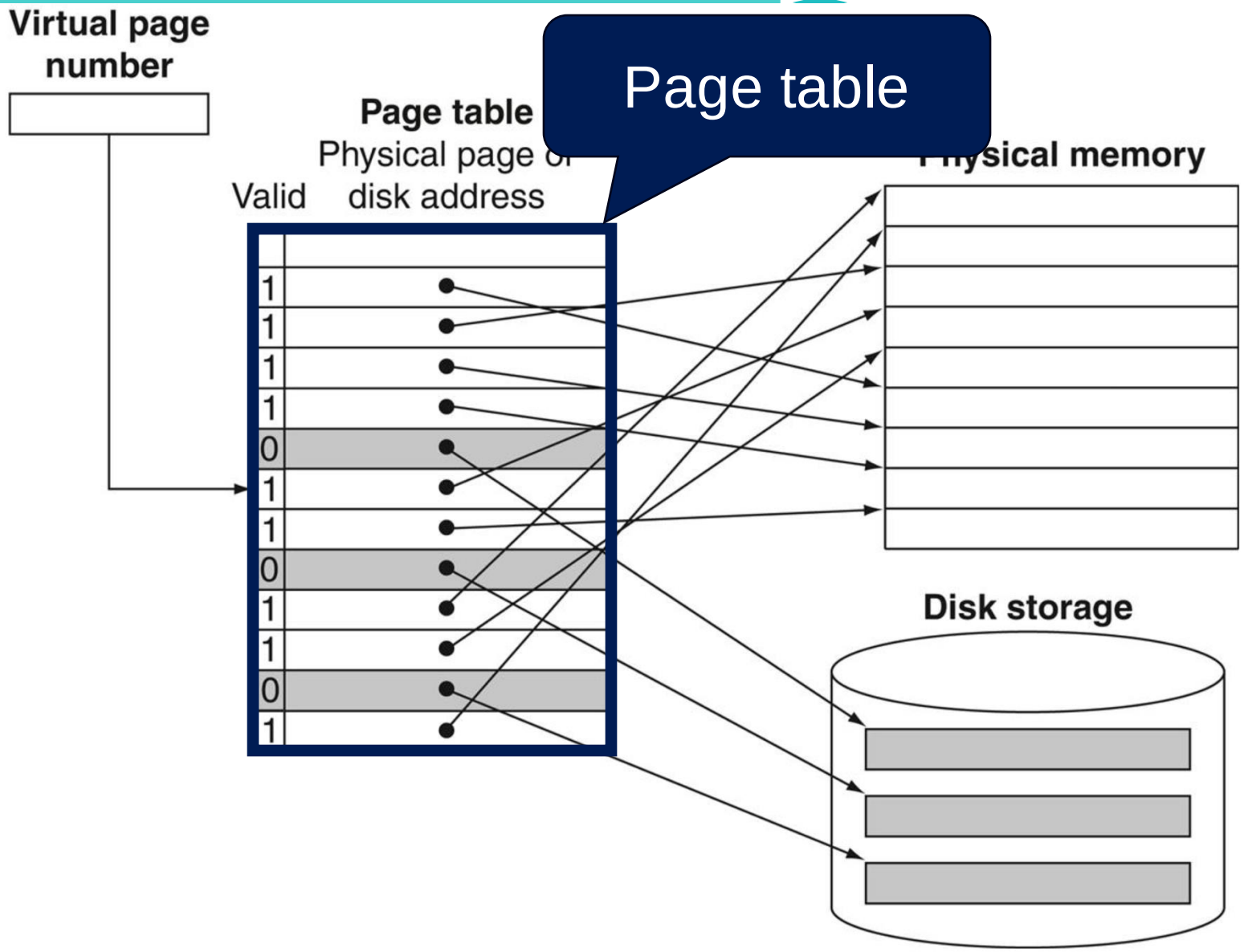


Page Table: Address Translation

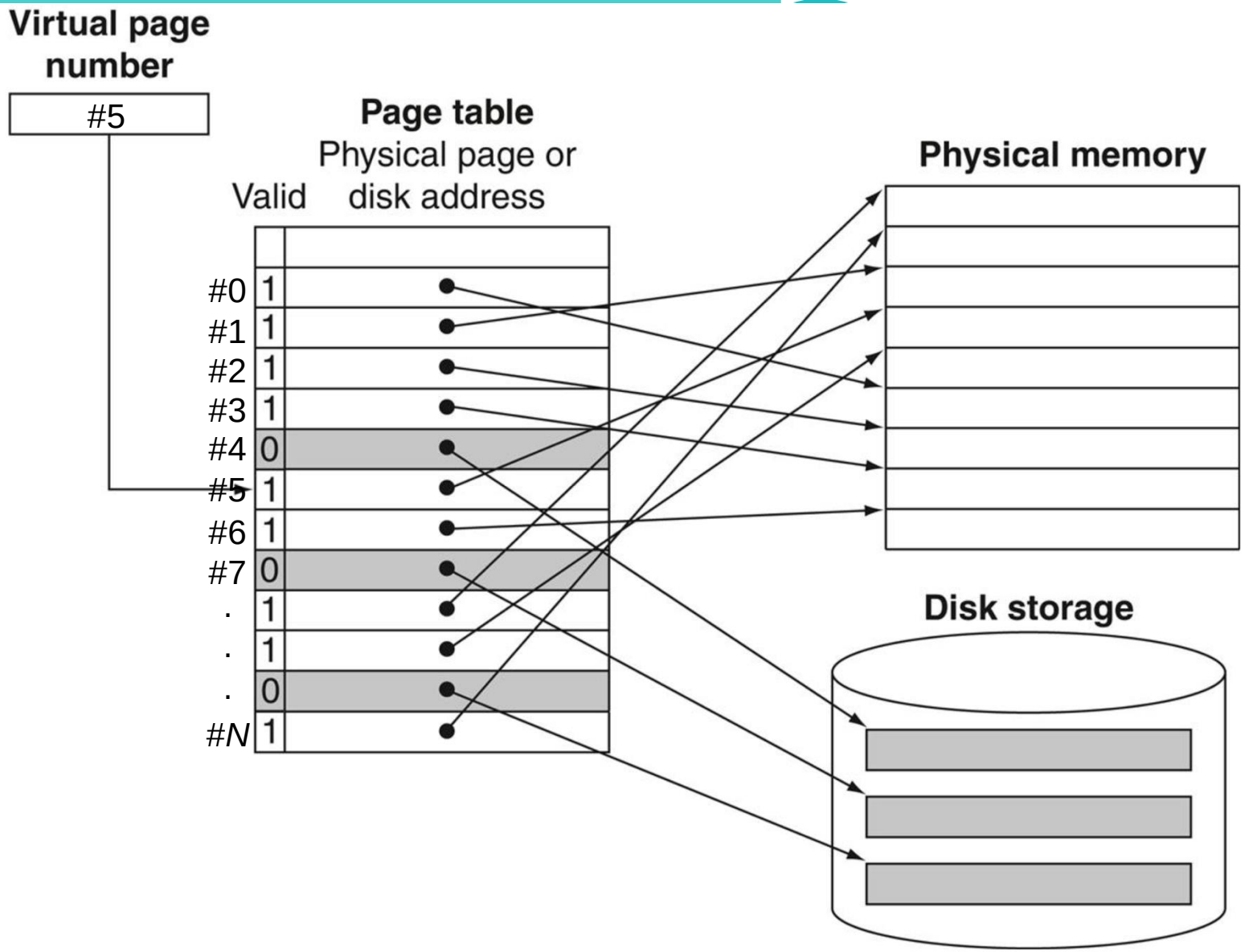
- CPU converts virtual addresses into physical addresses via an OS-managed lookup table called page table
- **Mapping from a virtual to physical address**
  - **Virtual address** = virtual page number + page offset
  - **Physical address** = physical page number + page offset

- Each program has its own page table!

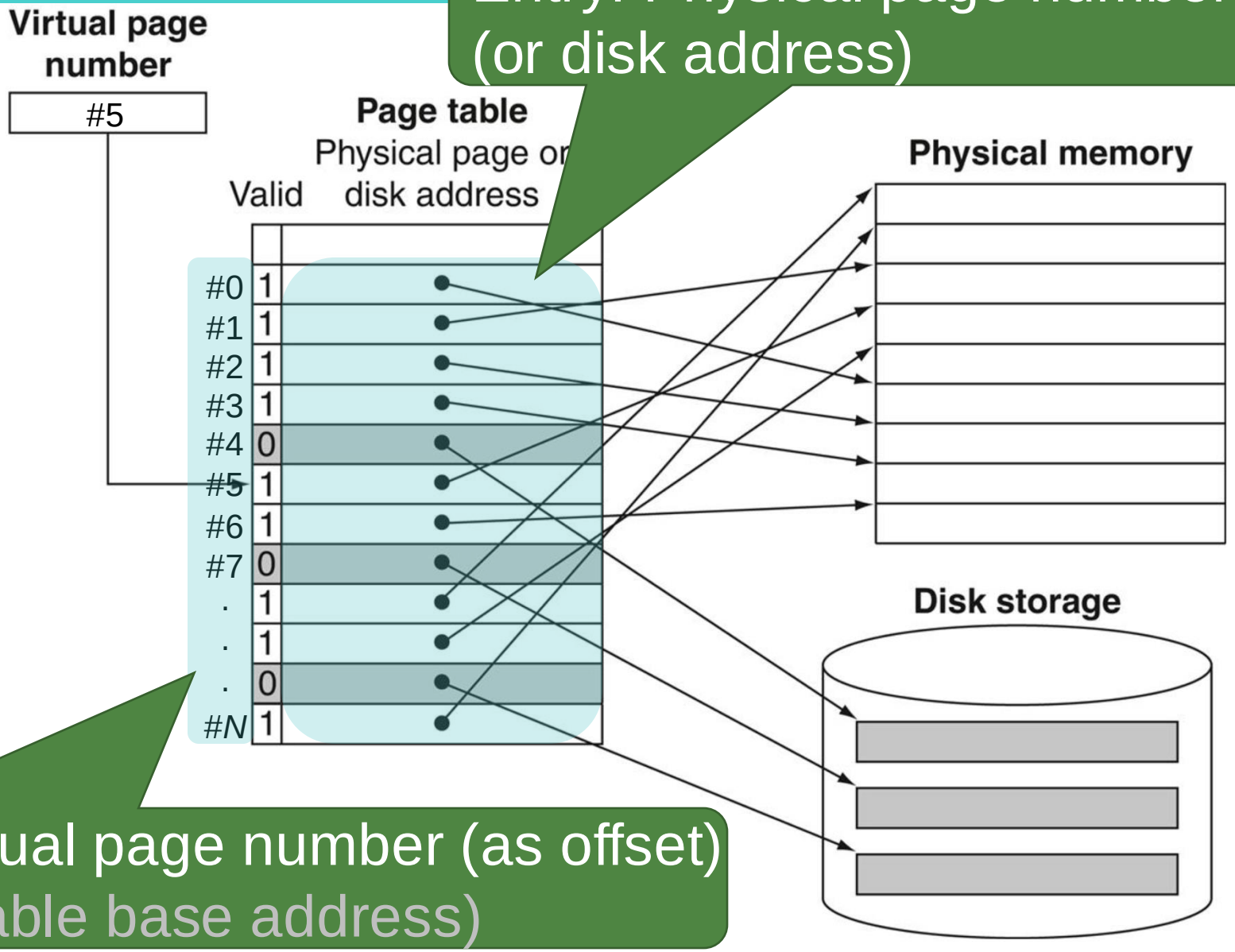
Page Table: Details



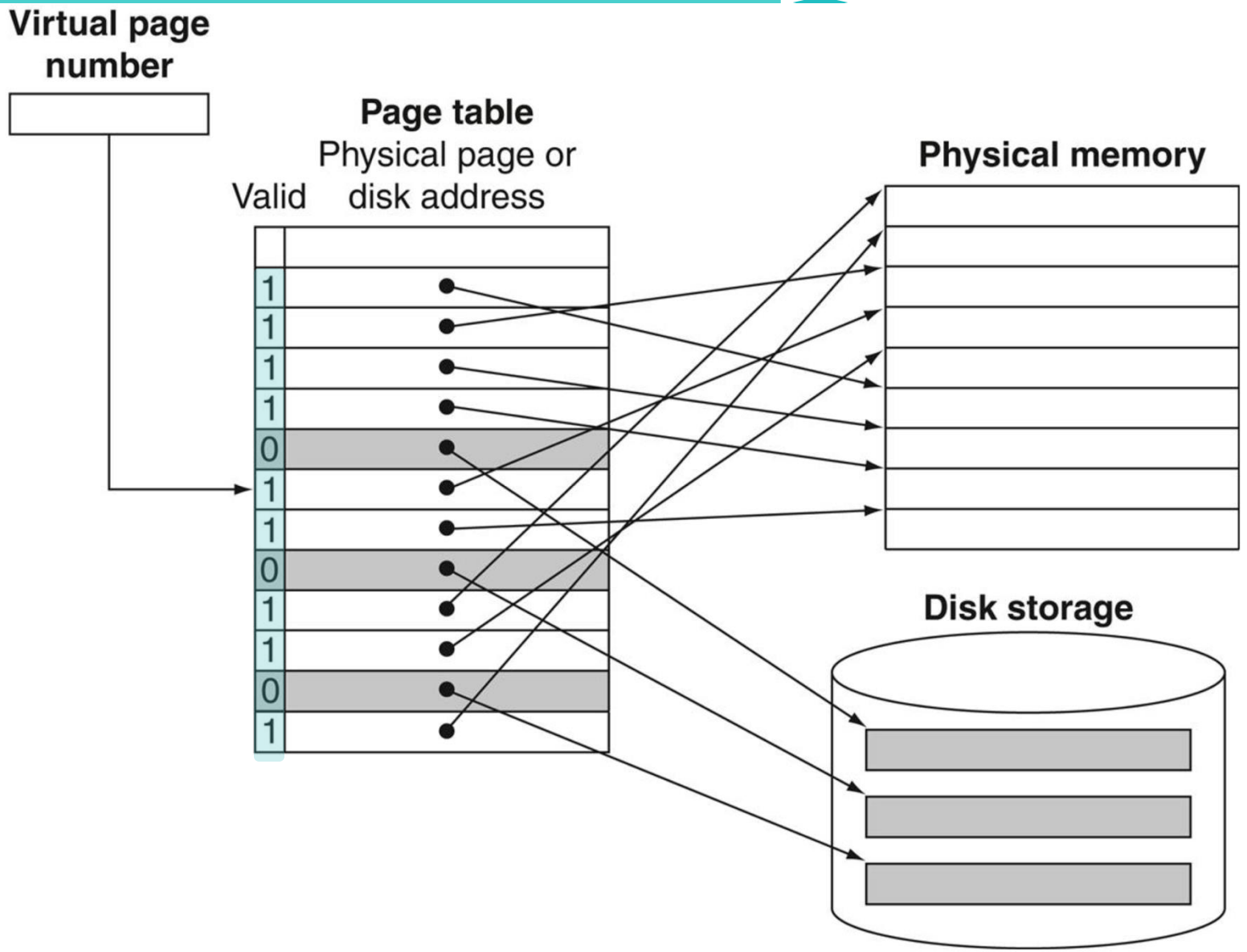
Page Table: Details



Page Table: Details

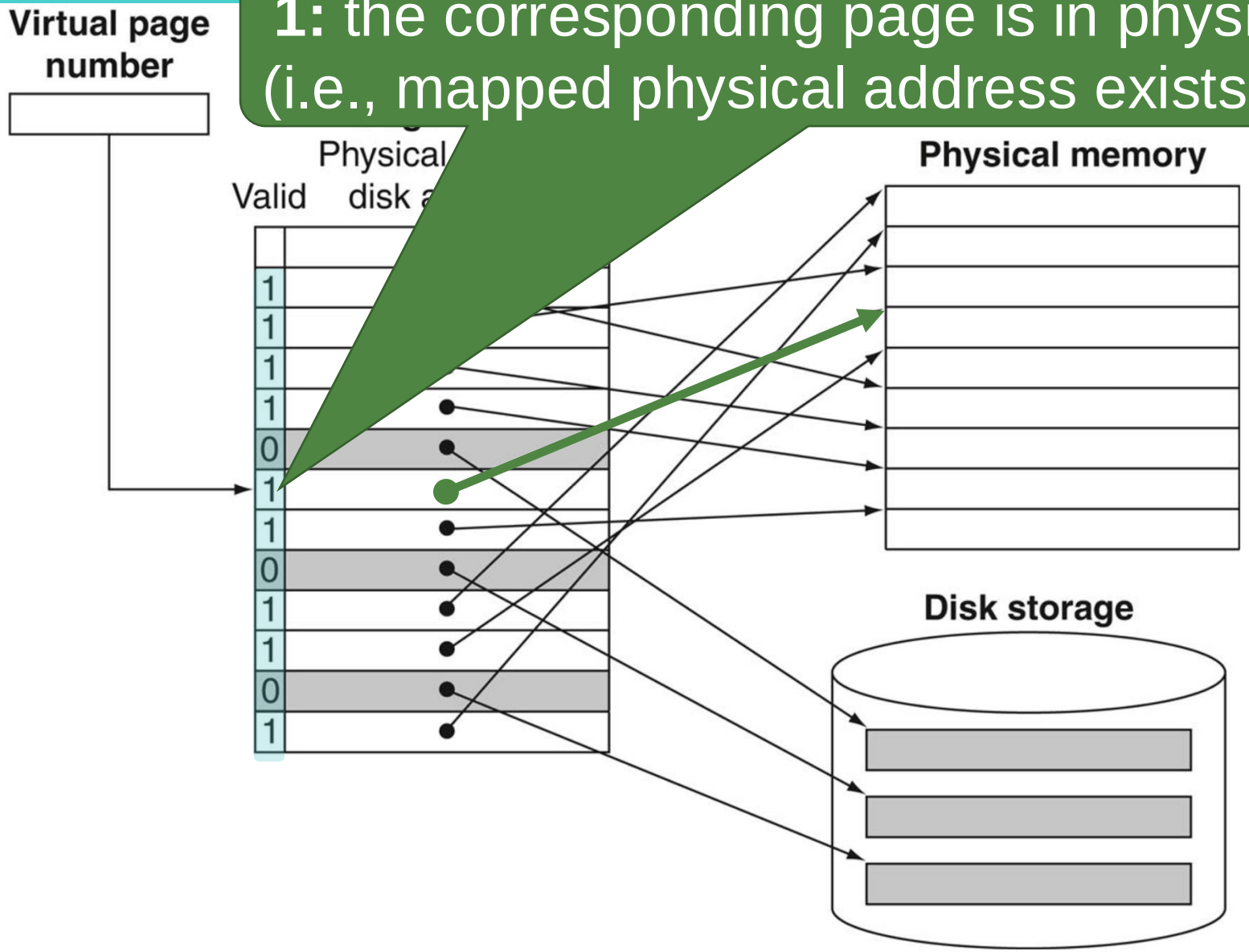


Valid Bit in Page Table



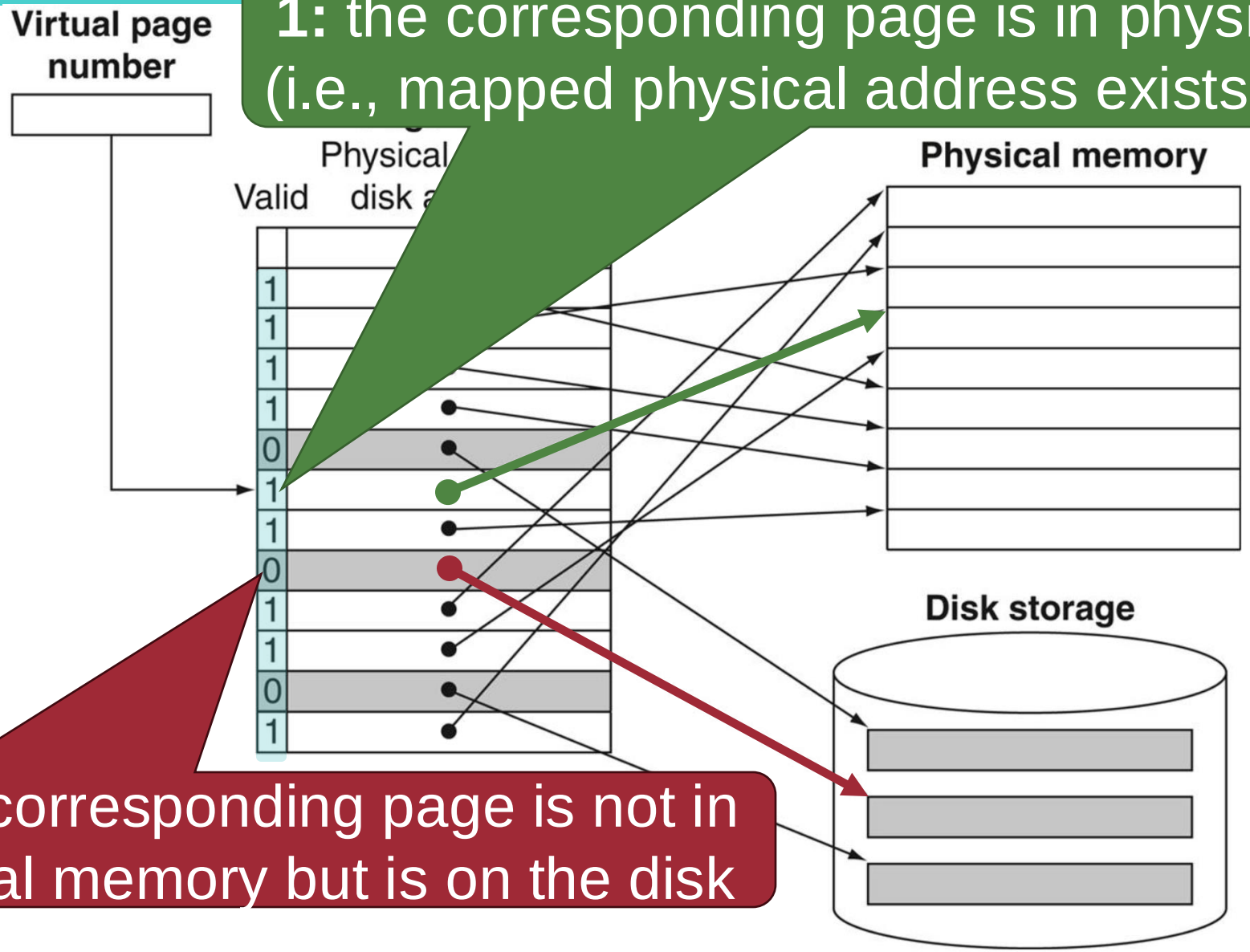
Valid Bit in Page Table

1: the corresponding page is in physical memory (i.e., mapped physical address exists in the entry)



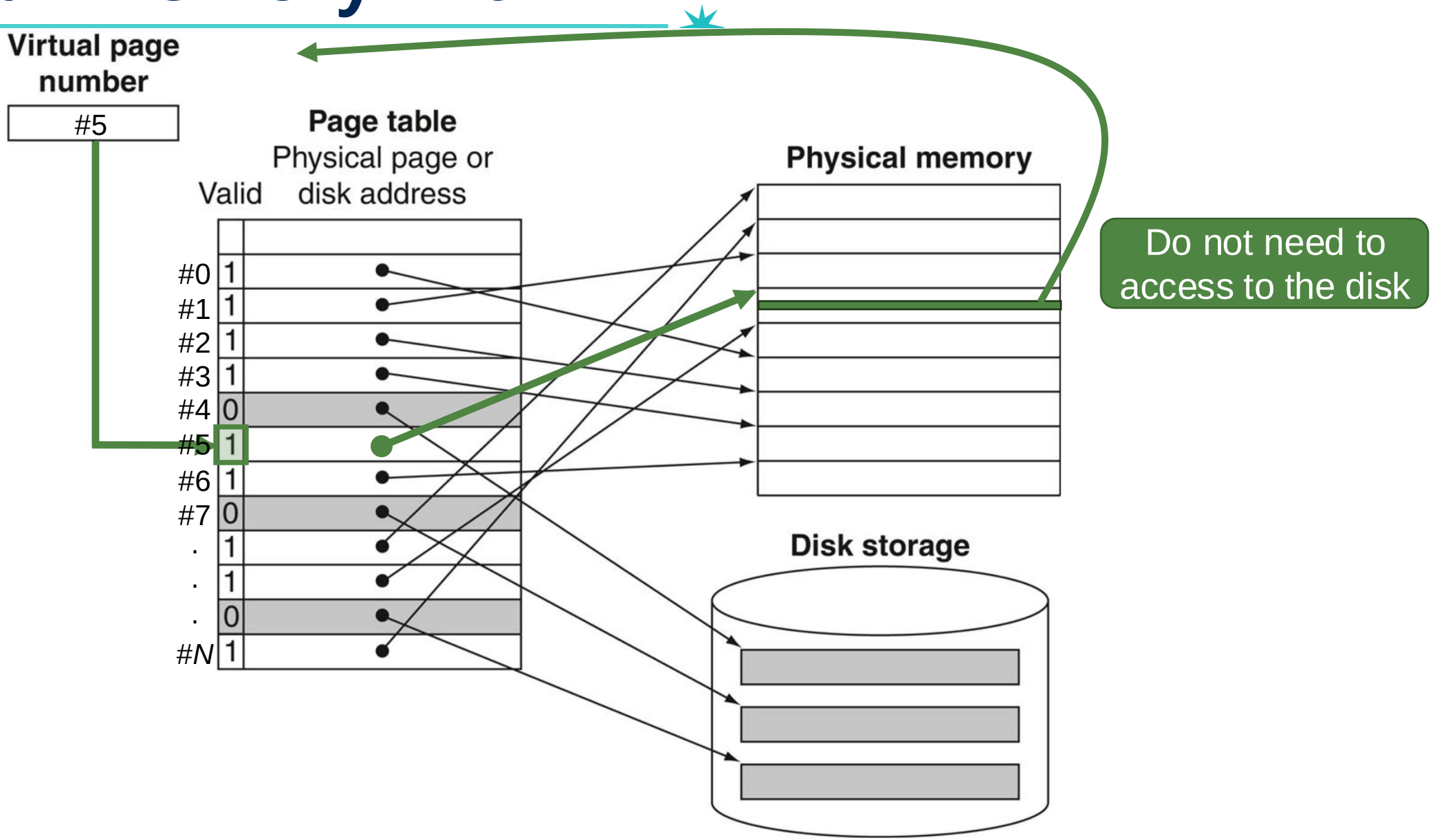
Valid Bit in Page Table

1: the corresponding page is in physical memory (i.e., mapped physical address exists in the entry)

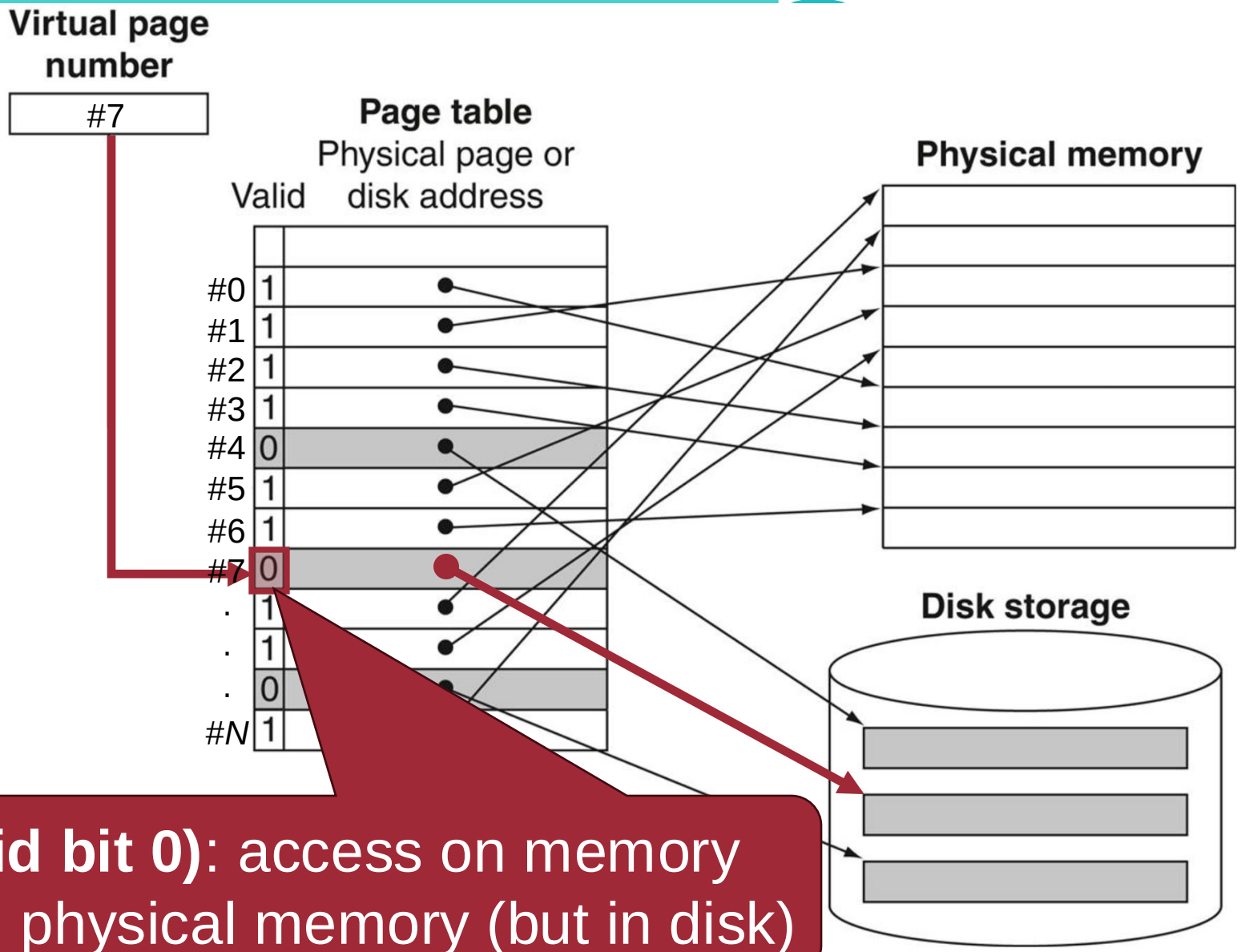


0: the corresponding page is not in physical memory but is on the disk

Virtual Memory Hit

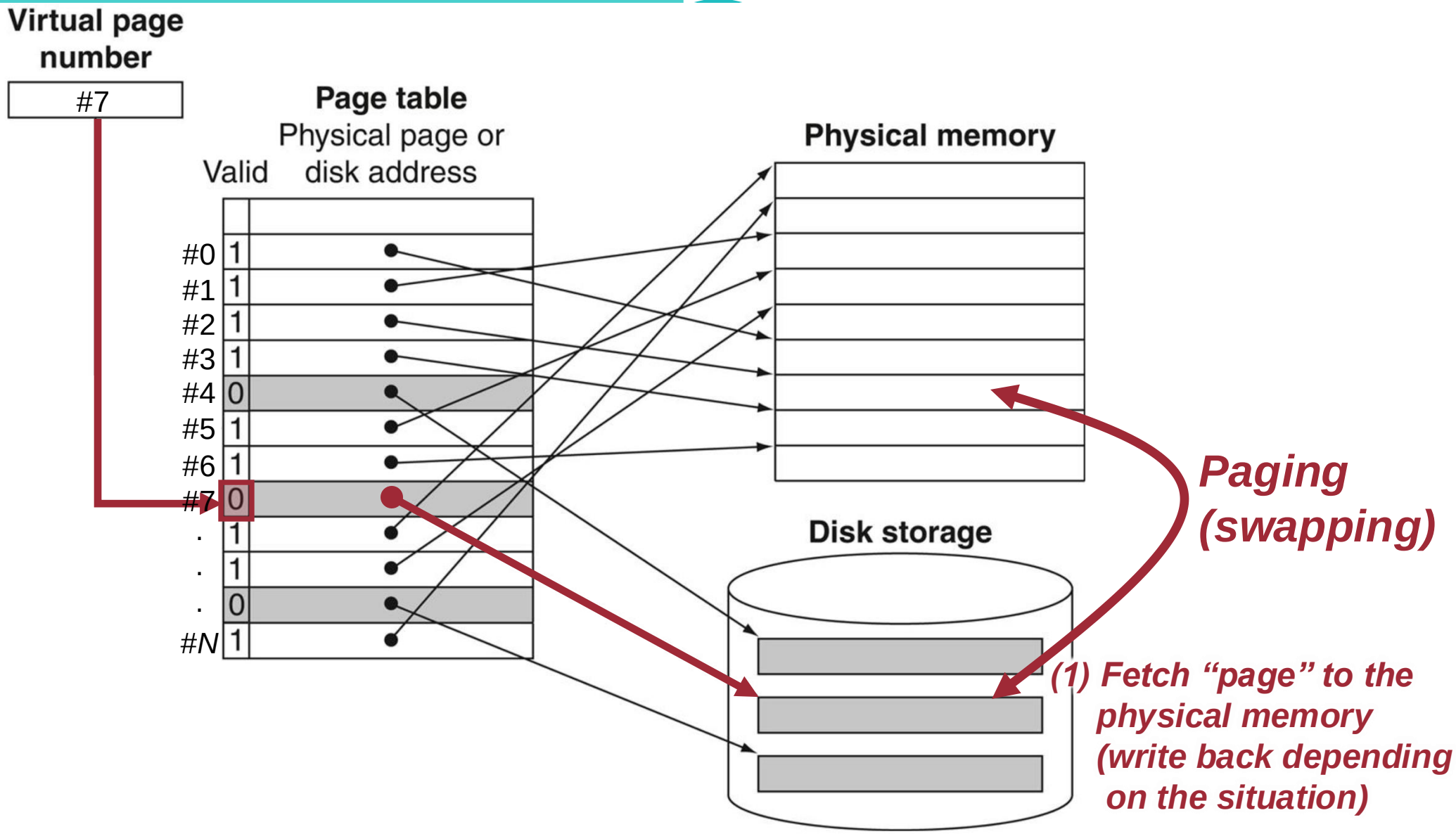


Virtual Memory Miss (Page Fault)

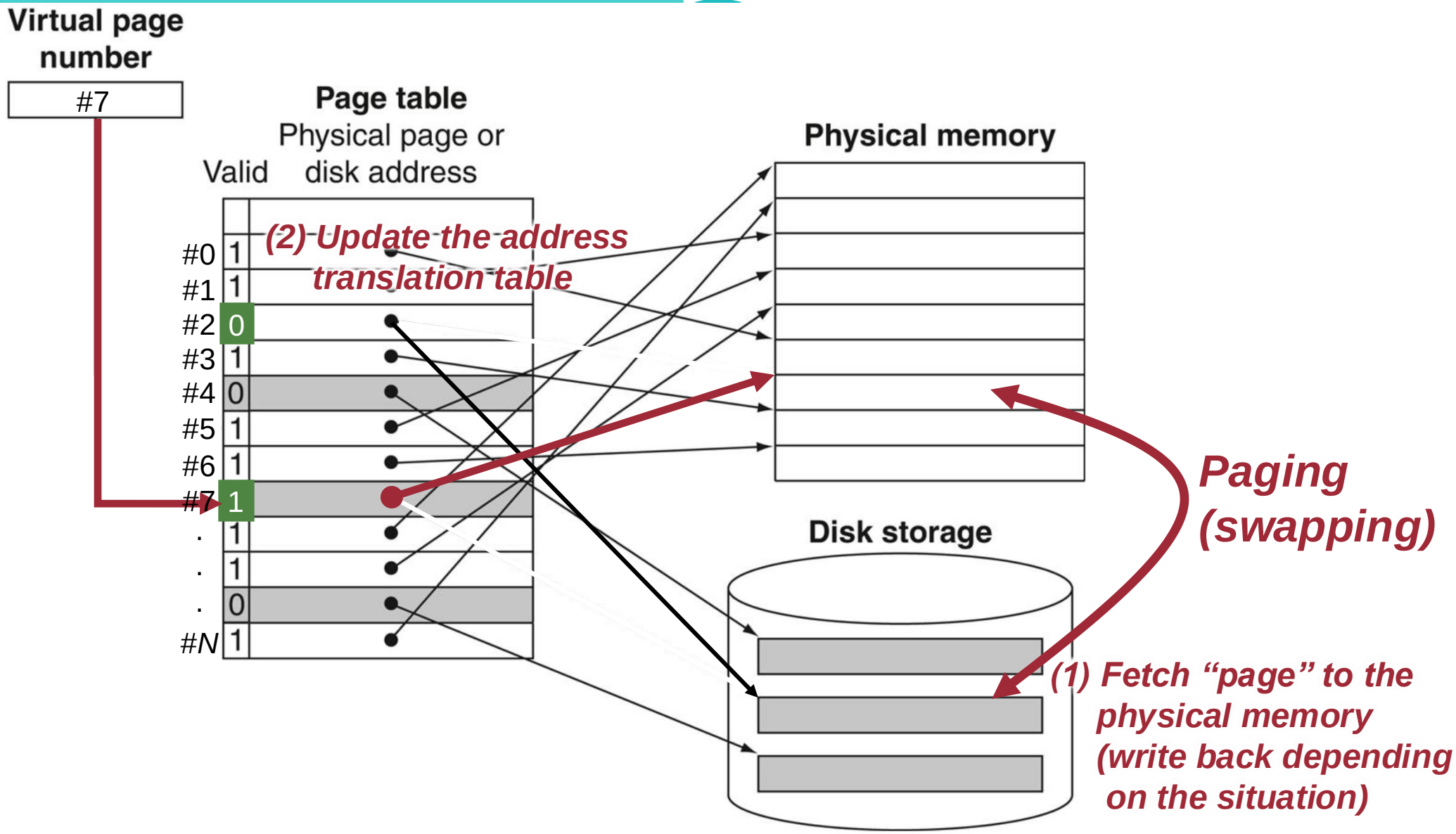


Miss (valid bit 0): access on memory space not in physical memory (but in disk)

Virtual Memory Miss (Page Fault)

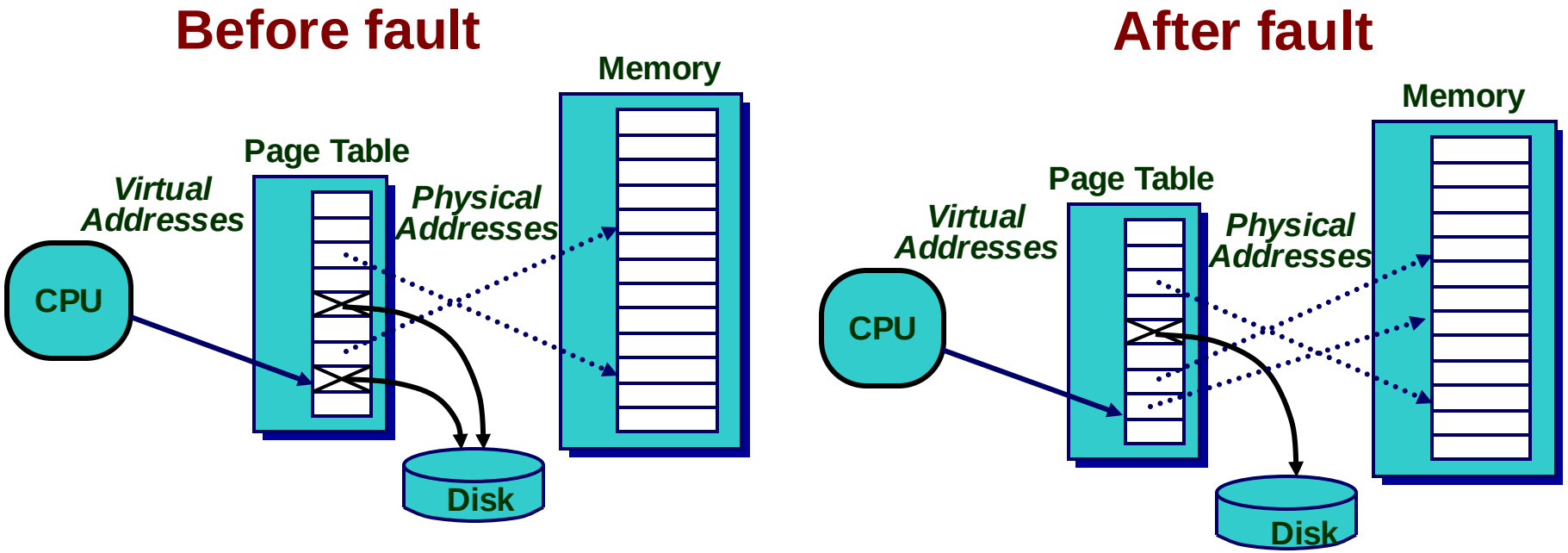


Virtual Memory Miss (Page Fault)



Virtual Memory Miss (Page Fault)

- If a page is not in physical memory but disk (if valid == 0)
  - Page table entry (especially, valid bit) indicates that the page not in memory
  - Page fault exception is occurred! OS trap handler invoked to move data from disk into memory
    - OS has full control over placement!



Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles

- Try to minimize page fault rate and fault penalty
  - Prefer **write back** (write through is impractical)
    - **Dirty** bit in each page table entry
  - Prefer **LRU** replacement
    - Reference bit in each page table set to 1 on access to page
    - Periodically cleared to 0 by OS
    - A page with **reference** bit = 0 has NOT been used recently

Page table entry

Page Fault Penalty

- On page fault, the pa
  - Takes millions of cloc

- Set (1) if the corresponding page is written
- Cleared (0) when the page is replaced

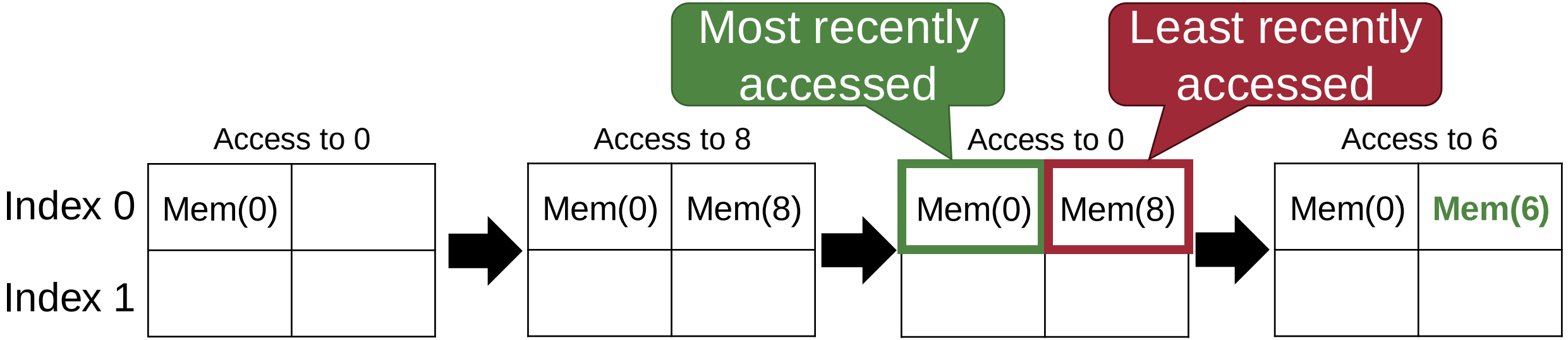
- Try to minimize page fa ate and fault penalty
  - Prefer **write ba** (write through is impractical)
    - **Dirty** bit in each page table entry
  - Prefer **LRU** replacement
    - Reference bit in each page table set to 1 on access to page
    - Periodically cleared to 0 by OS
    - A page with **reference** bit = 0 has NOT been used recently

Reference	Dirty	Valid	Physical page number

Page table entry

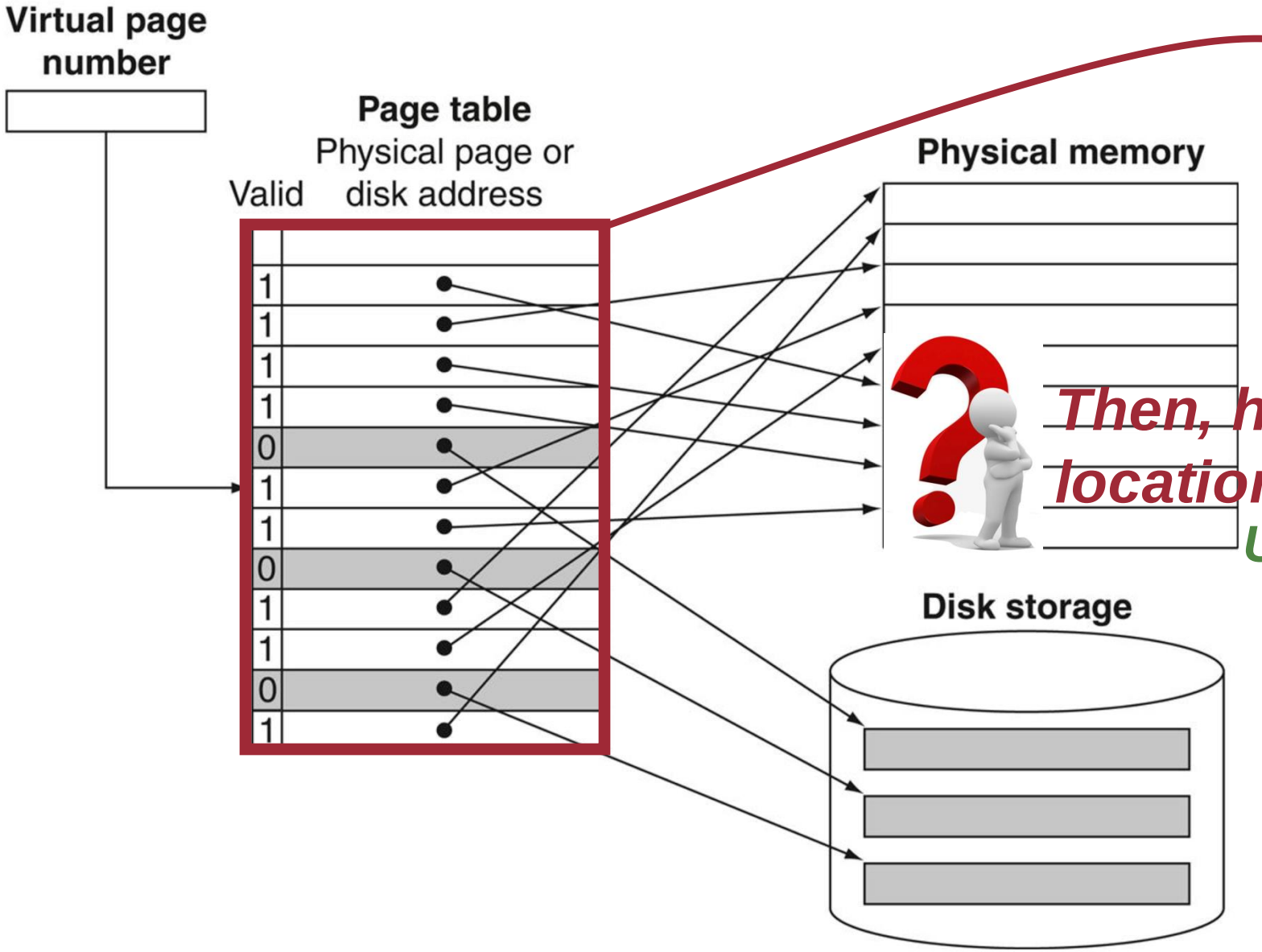
Recap: LRU

- **Least Recently Used (LRU)**: replace the one NOT used (accessed) for the longest time
  - Temporal locality of access is considered
  - Need a reference history information



2-way set associative

Place of the Page Table?



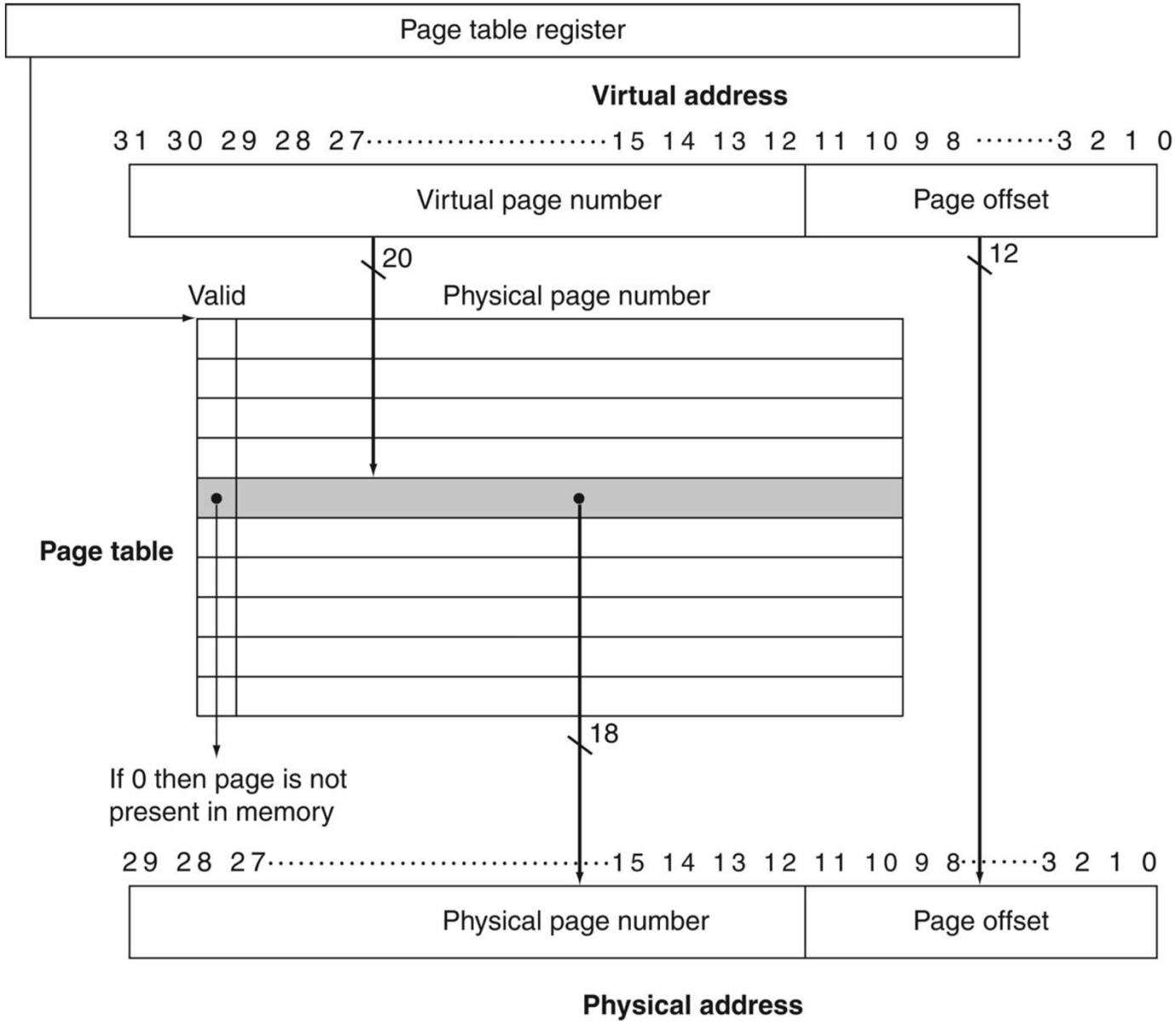
Where does the page table exist?

Physical memory!

Then, how can we know the location of the page table itself?

Use a page table (base) register

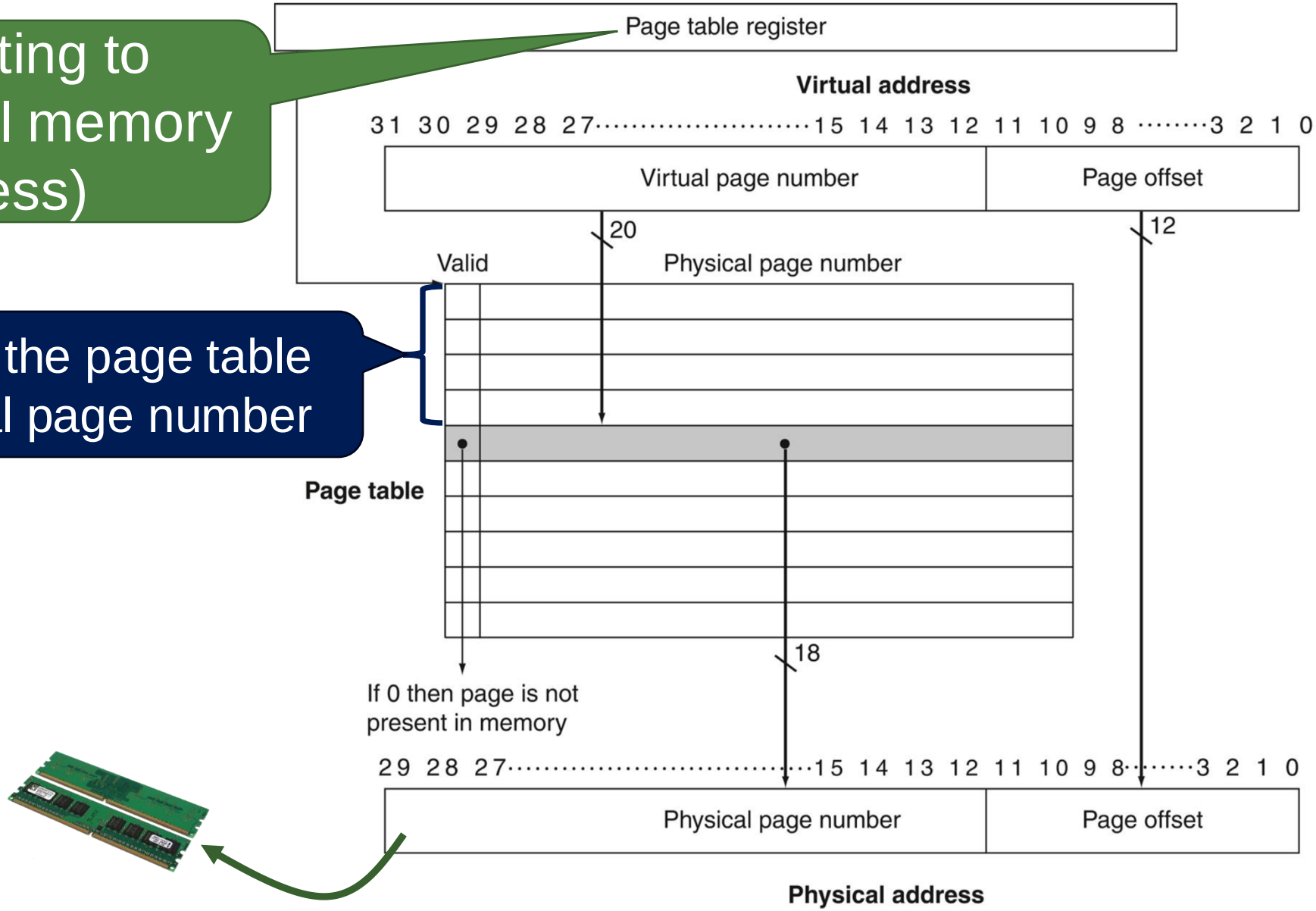
Page Table Register



Page Table Register

CPU register pointing to page table in physical memory (Absolute address)

Index: value of the page table register + virtual page number



Question?