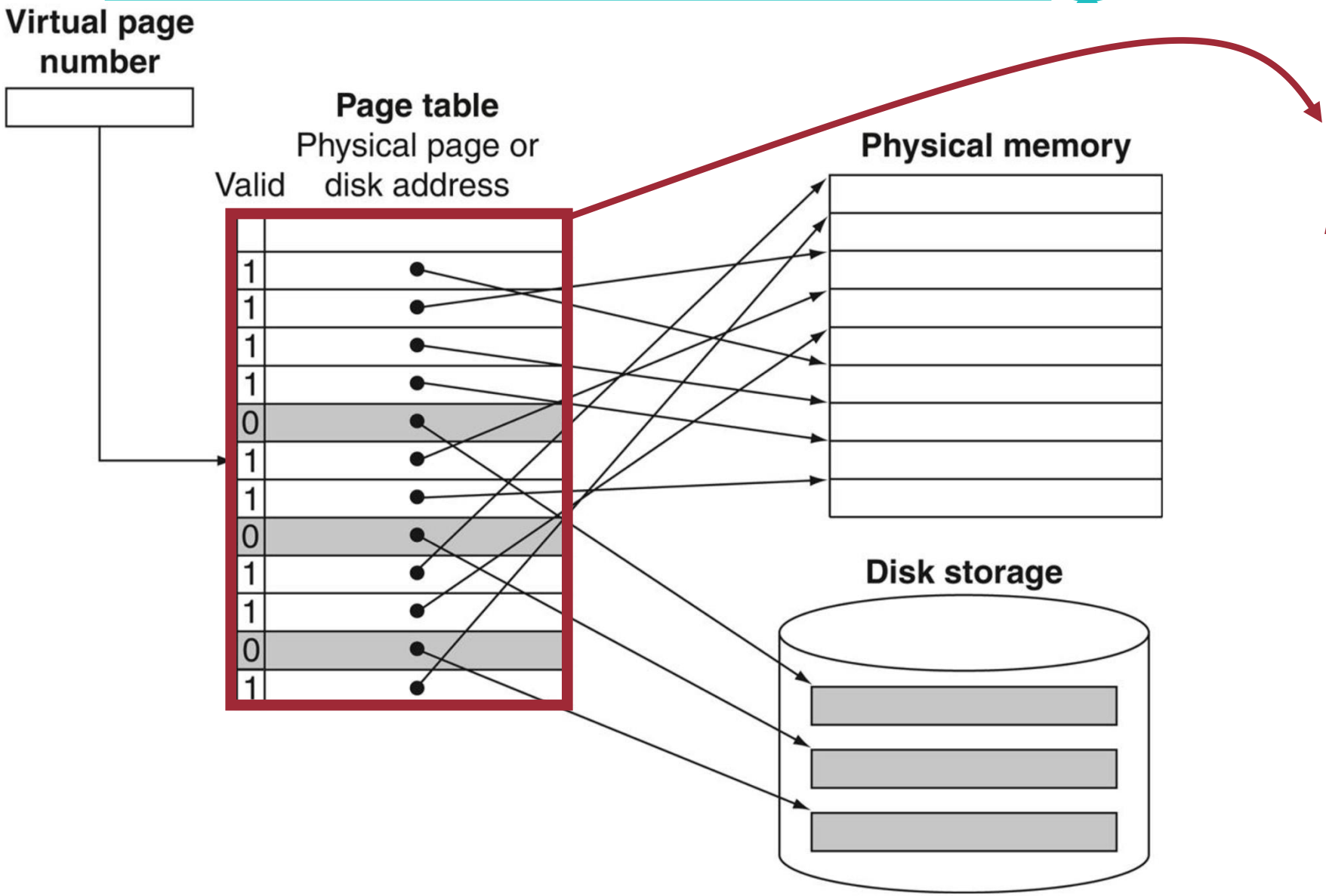
CSE251: System Programming

16. Virtual Memory (2)

Seongil Wi

Department of Computer Science and Engineering

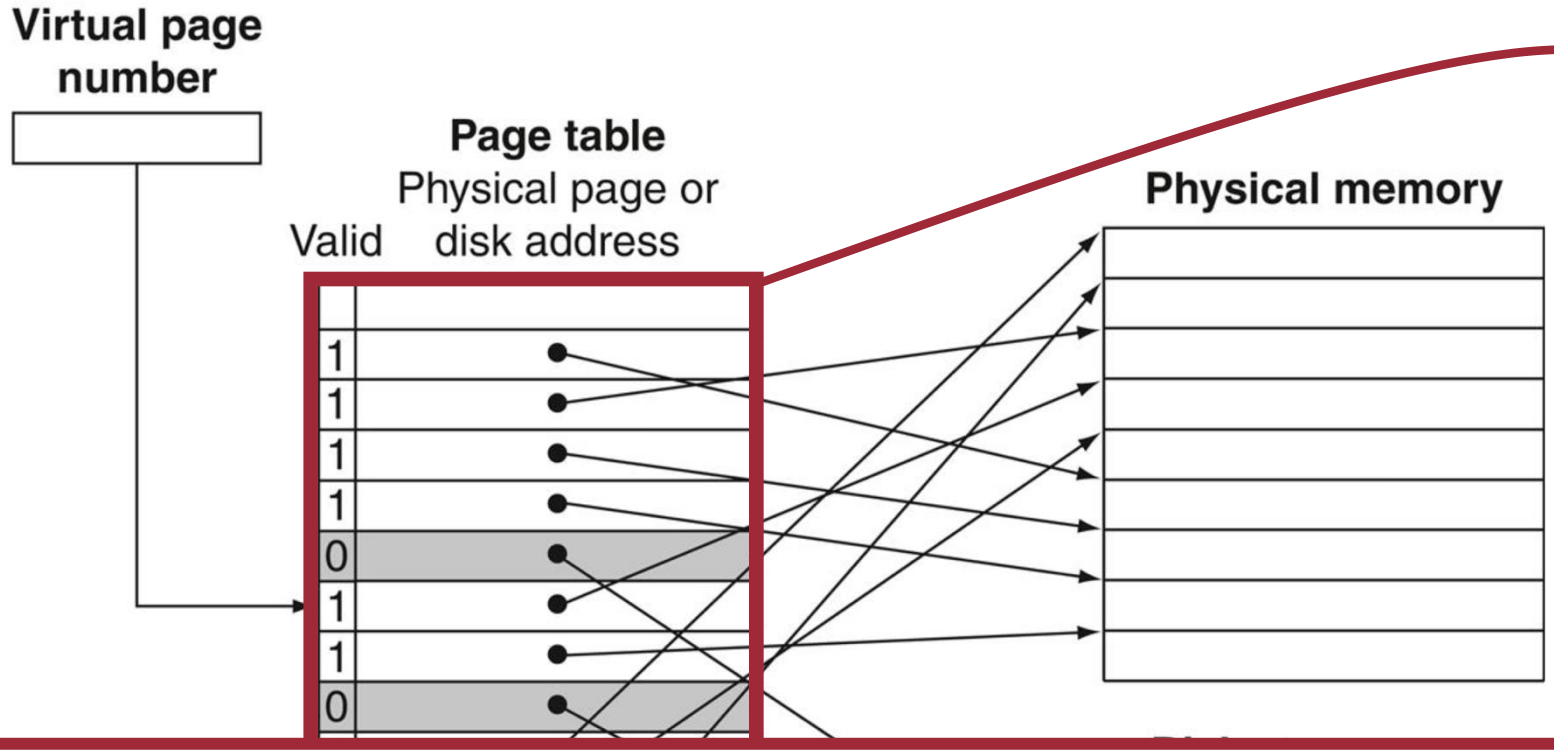
Place of the Page Table?



Where does the page table exist?

Physical memory!

Place of the Page Table?



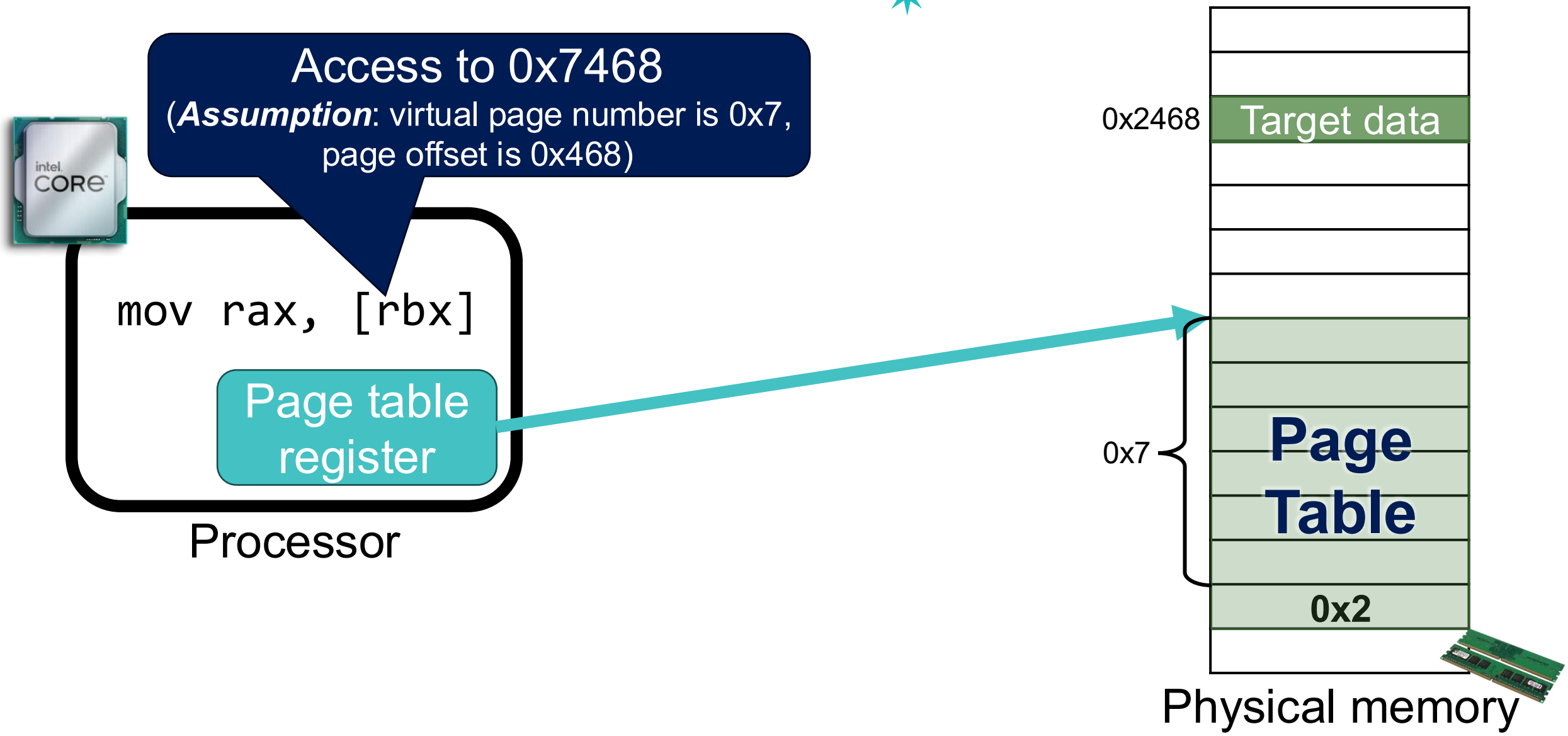
Where does the page table exist?

Physical memory!

Any problems?

Multiple memory access: ***<u>one memory access</u>*** *to obtain the physical address and* ***<u>a second access</u>*** *to get the data*

Problem: Multiple Memory Access



Problem: Multiple Memory Access

Access to 0x7468
(***Assumption***: virtual page number is 0x7, page offset is 0x468)

```
mov rax, [rbx]
```

Page table register

Processor

(1) Memory access to obtain the physical address

0x2468 Target data

0x7

Page Table

0x2

Virtual page #

Physical page #

Physical memory

Problem: Multiple Memory Access

Access to 0x7468
(***Assumption***: virtual page number is 0x7, page offset is 0x468)

```
mov rax, [rbx]
```

Page table register

Processor

(1) Memory access to obtain the physical address

(2) Memory access to get the data

Physical address

0x2468

Target data

0x7

Page Table

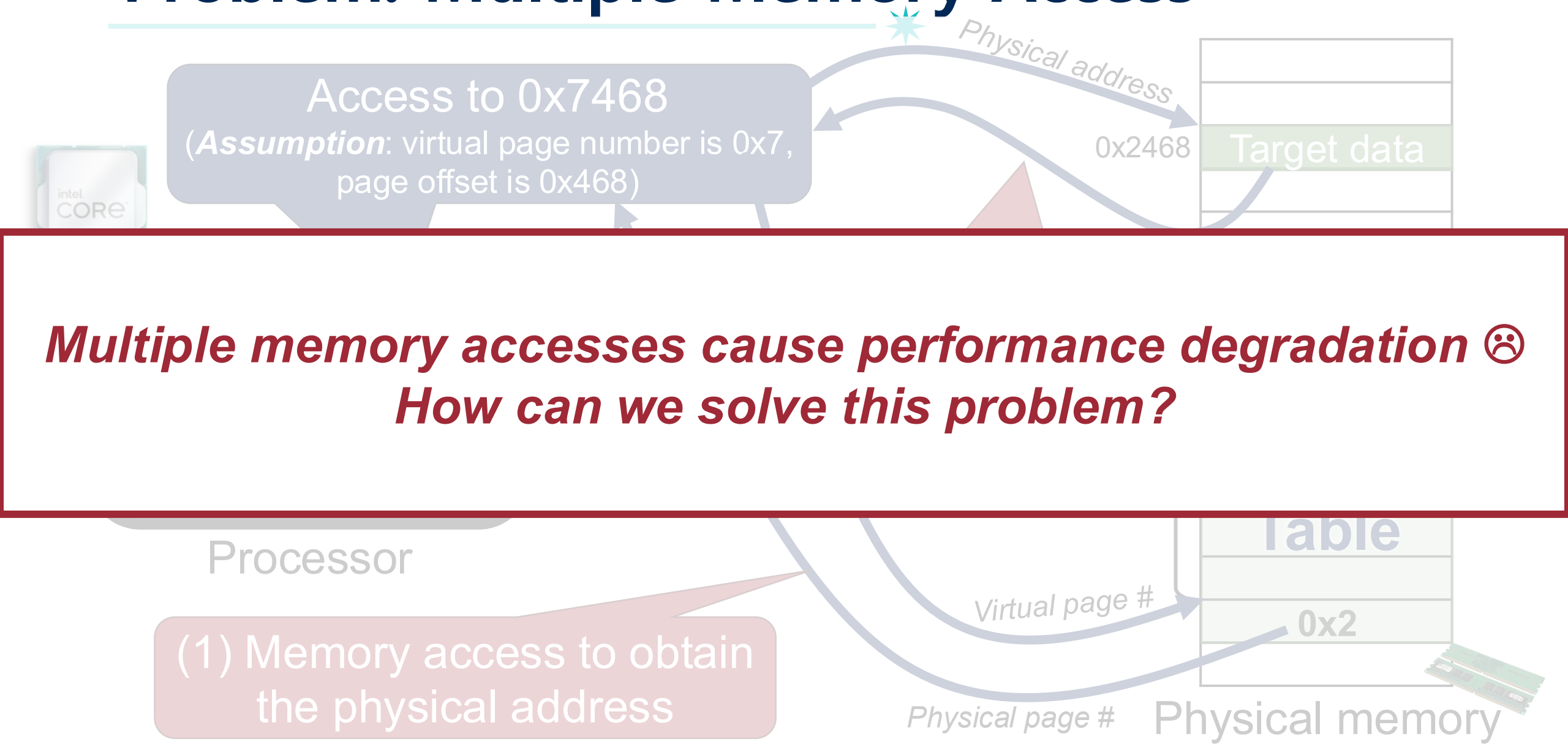
Virtual page #

0x2

Physical page #

Physical memory

Problem: Multiple Memory Access



Multiple memory accesses cause performance degradation ☹
How can we solve this problem?

Solution: A Cache for Address Translation

Access to 0x7468
(***Assumption***: virtual page number is 0x7, page offset is 0x468)

```
mov rax, [rbx]
```

Page table register

Processor

Translation cache

0x2468 Target data

Page Table

0x2

Physical memory

Solution: A Cache for Address Translation

Access to 0x7468
(***Assumption***: virtual page number is 0x7, page offset is 0x468)

```
mov rax, [rbx]
```

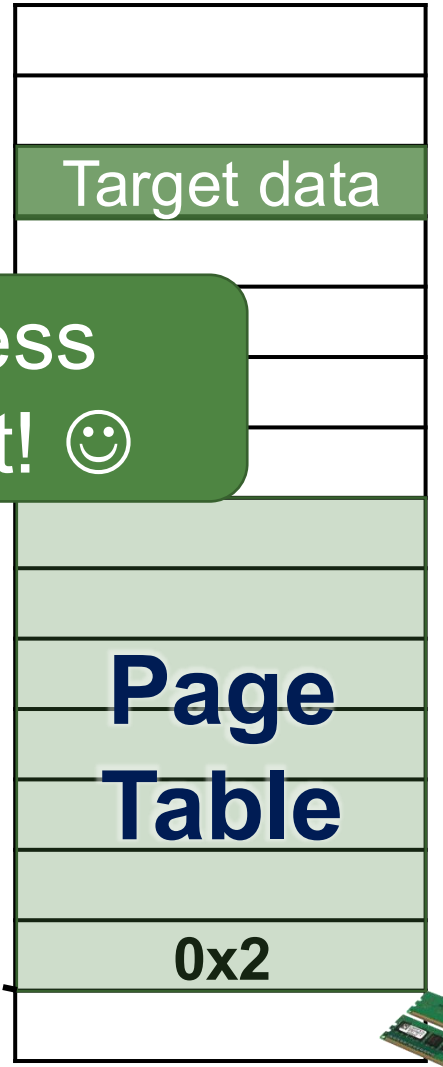
Page table register

Processor

Translation cache

Making address translation fast! ☺

Translation-Lookaside Buffer (TLB)



Physical memory

Making Address Translation Fast: the TLB

Translation-Lookaside Buffer (TLB)

A ***cache*** that keeps track of recently used address mappings to try to avoid an access to the page table

- Small set-associative hardware cache in MMU

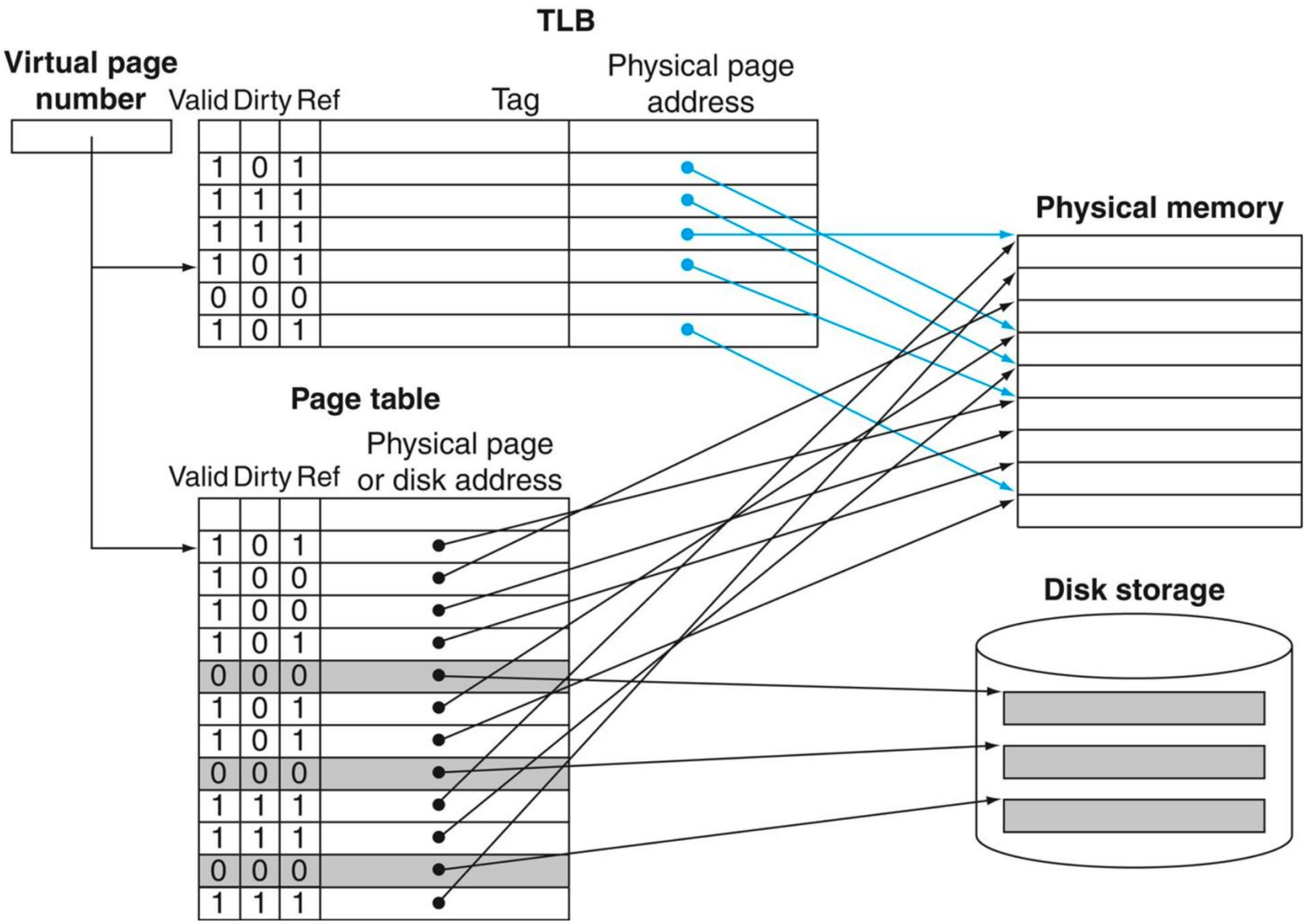
Translation-Lookaside Buffer (TLB)

A ***cache*** that keeps track of recently used address mappings to try to avoid an access to the page table

- Small set-associative hardware cache in MMU
- Rely on ***locality*** of reference to the page table

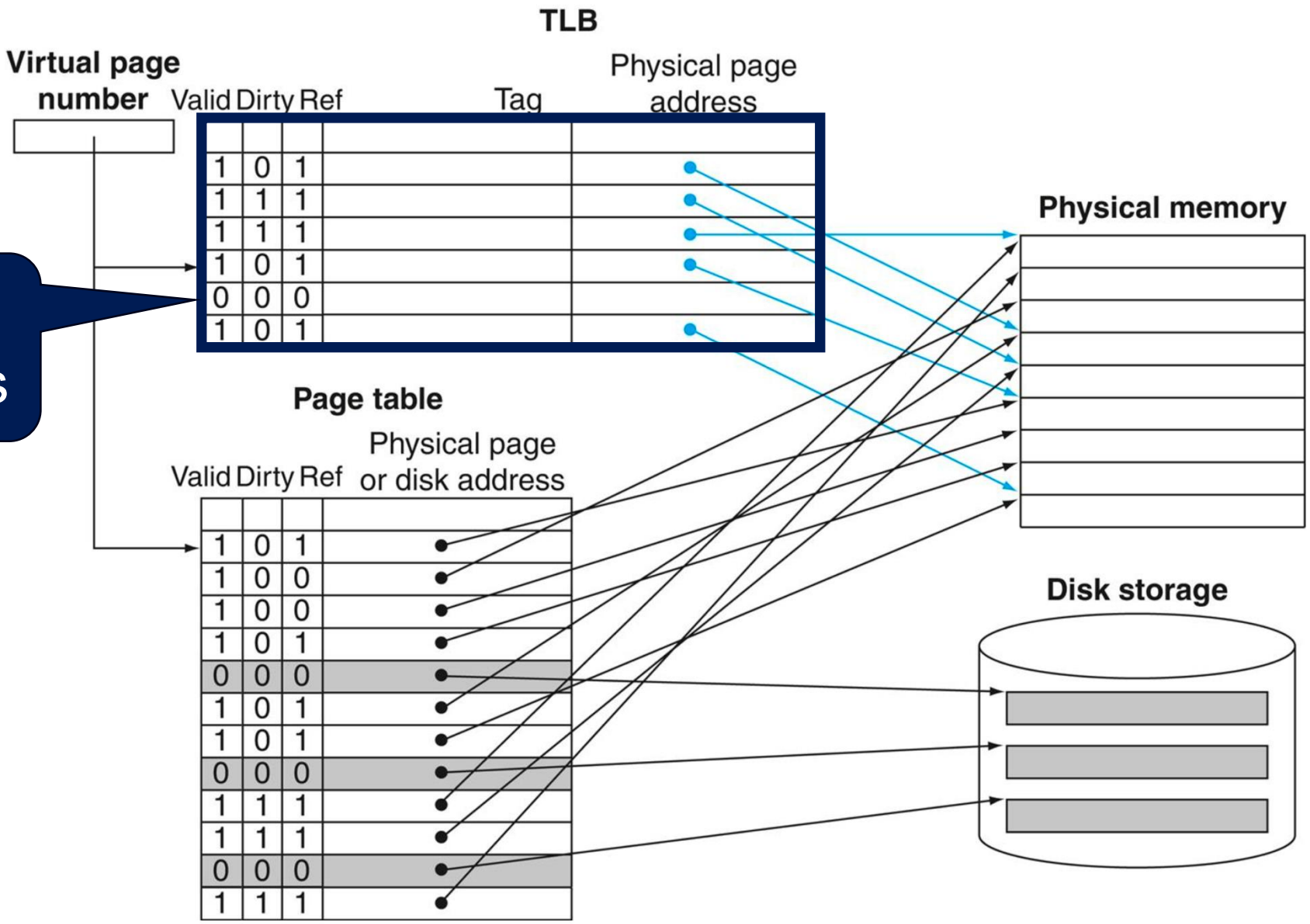
*"When a translation for a virtual page number is used, it will probably be needed again **in the near future**"*

Translation-Lookaside Buffer (TLB)



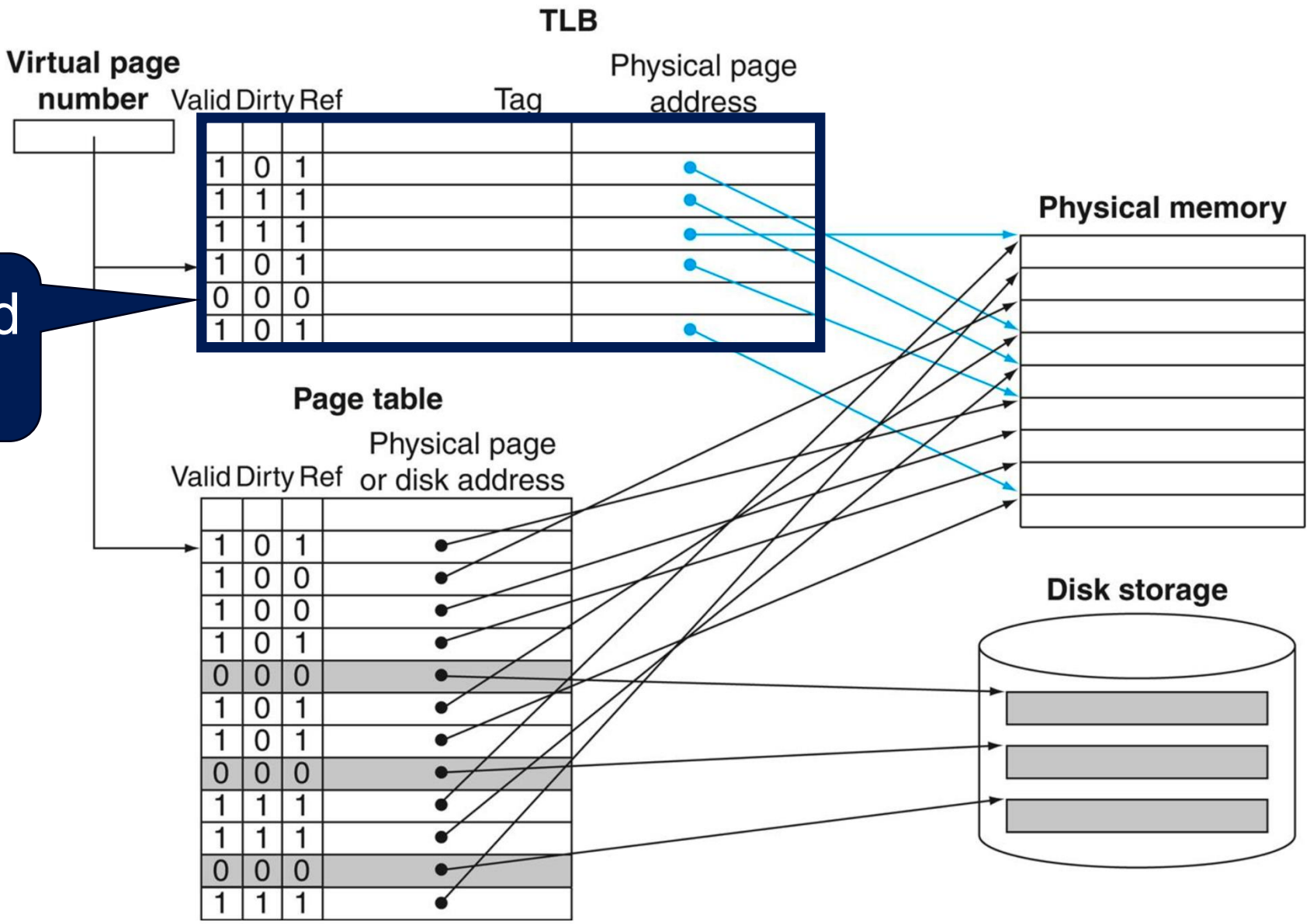
Translation-Lookaside Buffer (TLB)

TLB: a special cache for address translations

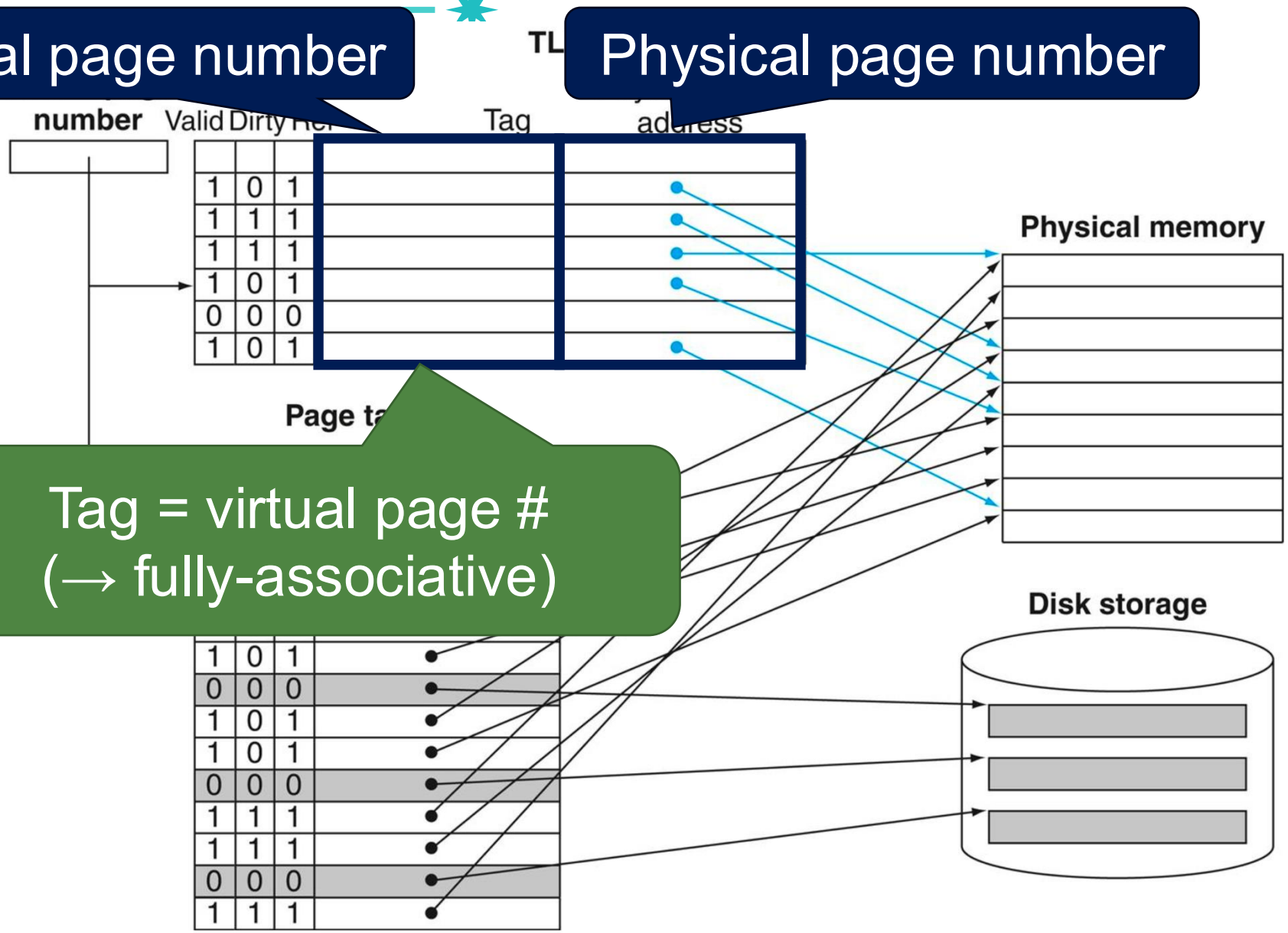


Translation-Lookaside Buffer (TLB)

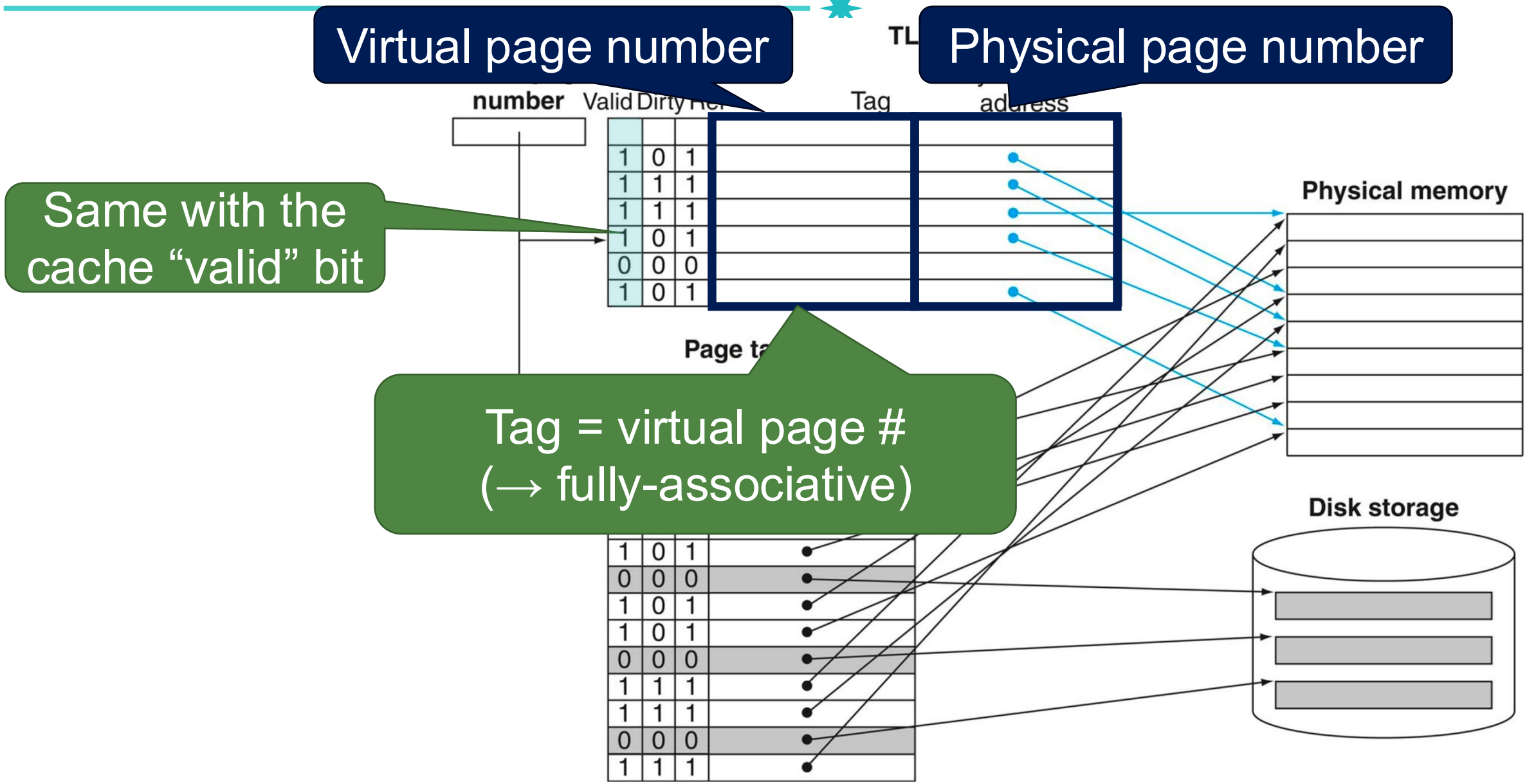
Stores the recently used address mappings



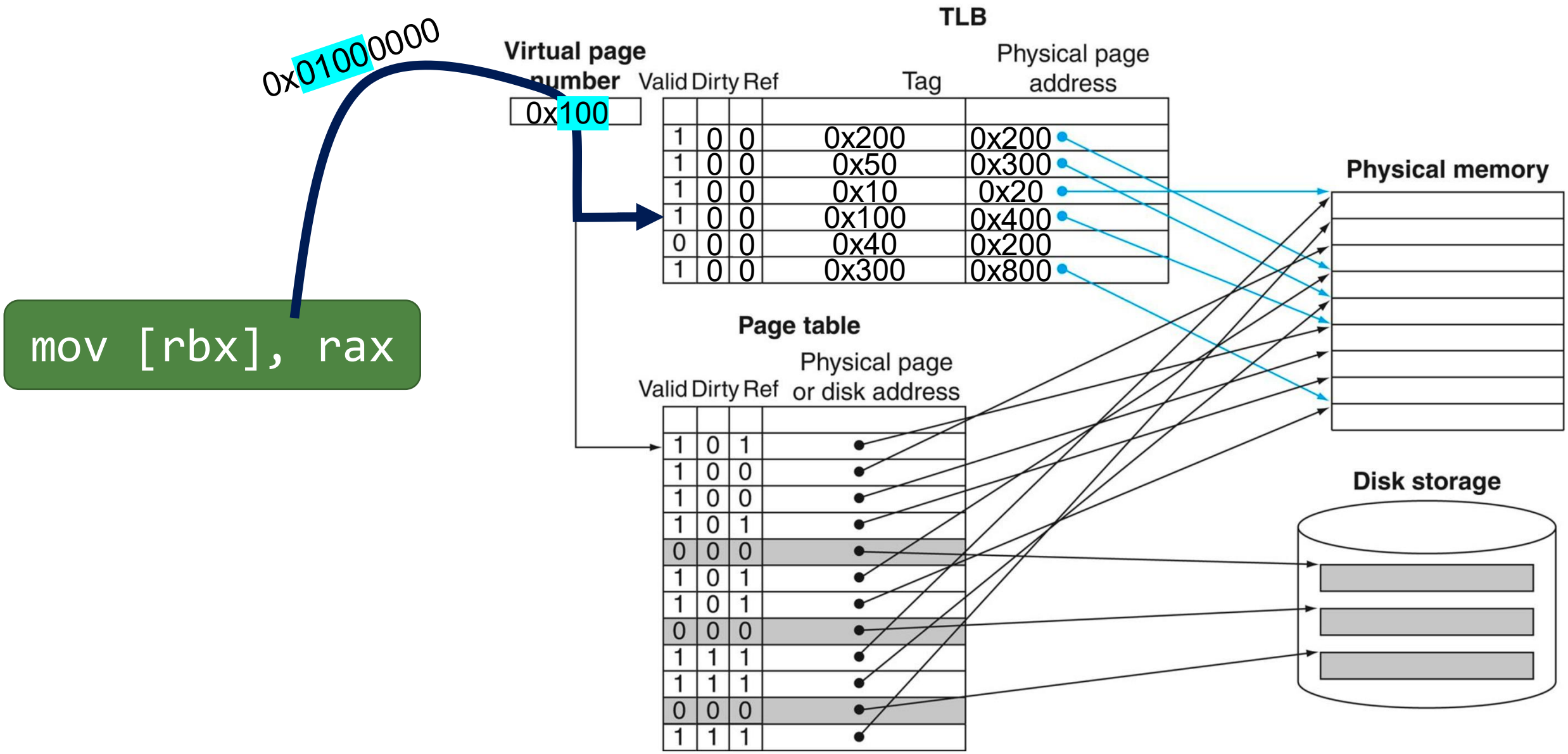
Translation-Lookaside Buffer (TLB)



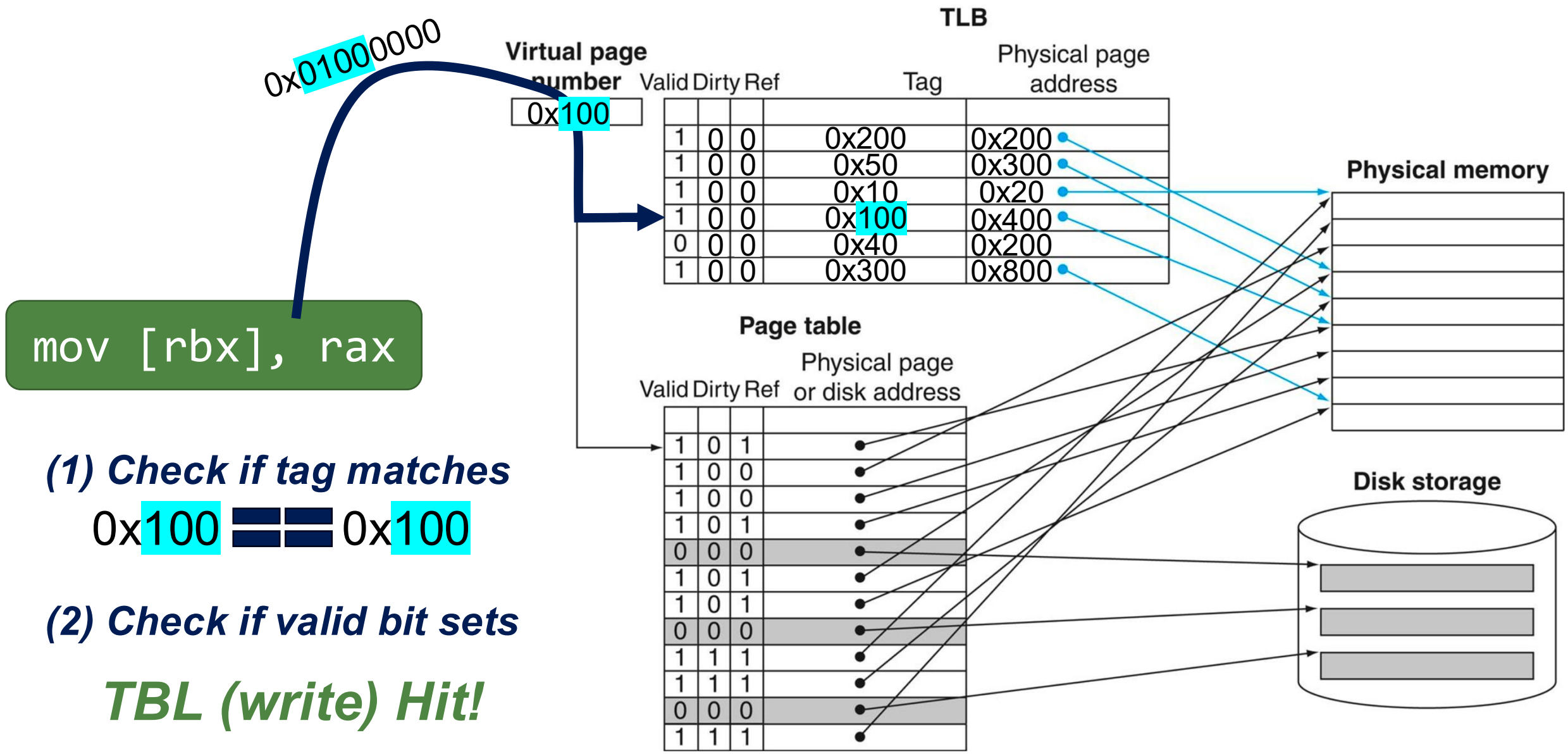
Translation-Lookaside Buffer (TLB)



Case #1: TLB Hits



Case #1: TLB Hits



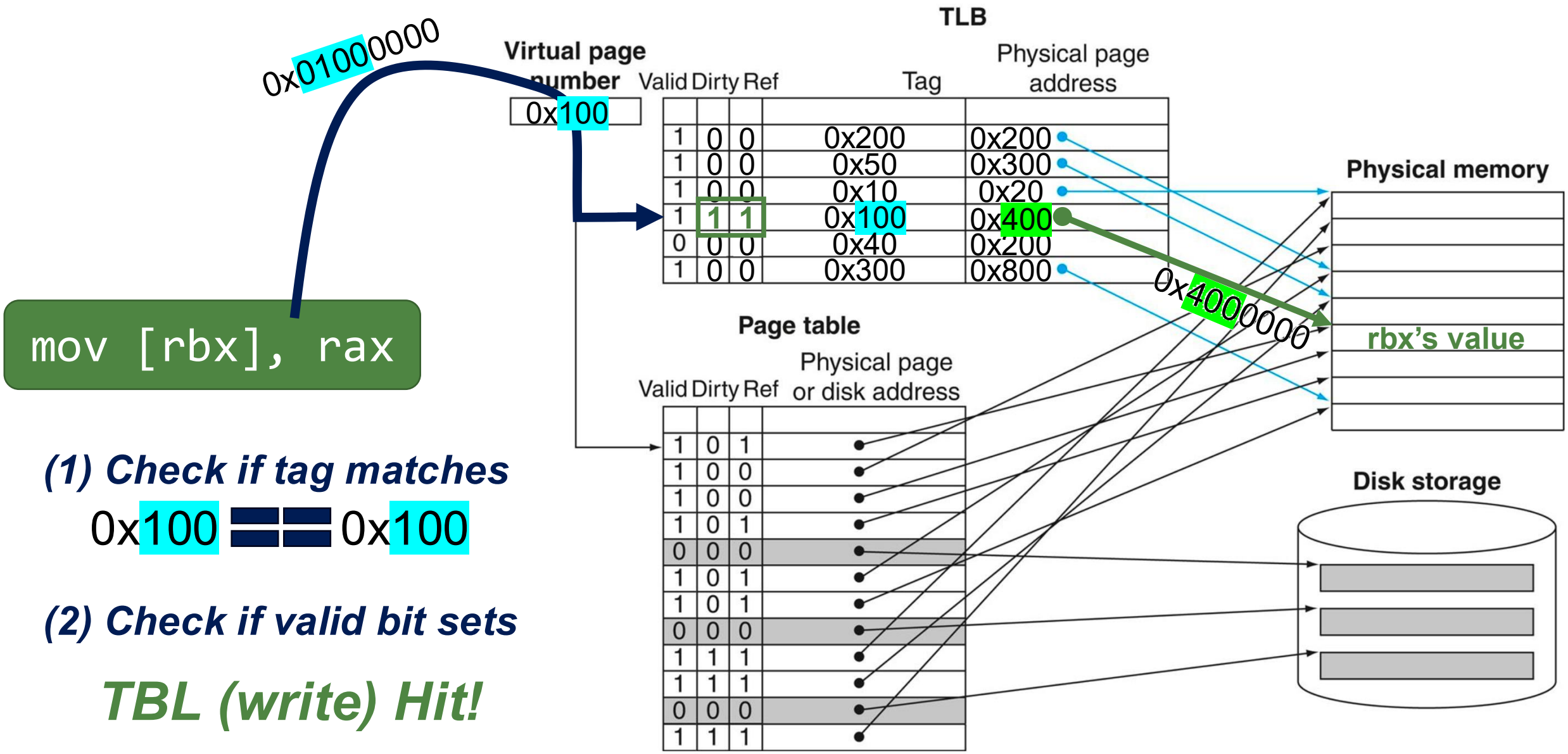
(1) Check if tag matches

0x100 == 0x100

(2) Check if valid bit sets

TBL (write) Hit!

Case #1: TLB Hits



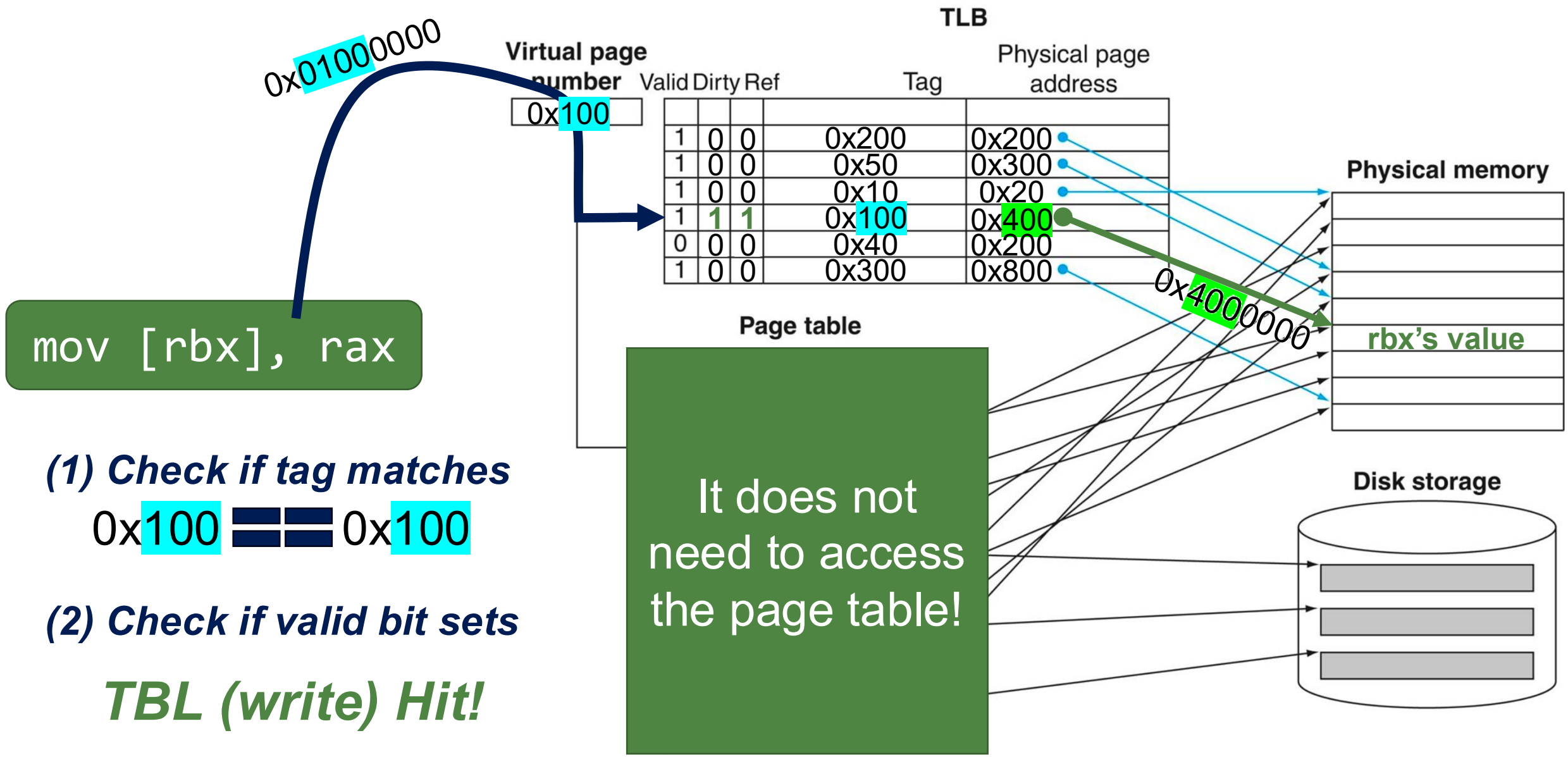
(1) Check if tag matches

0x100 == 0x100

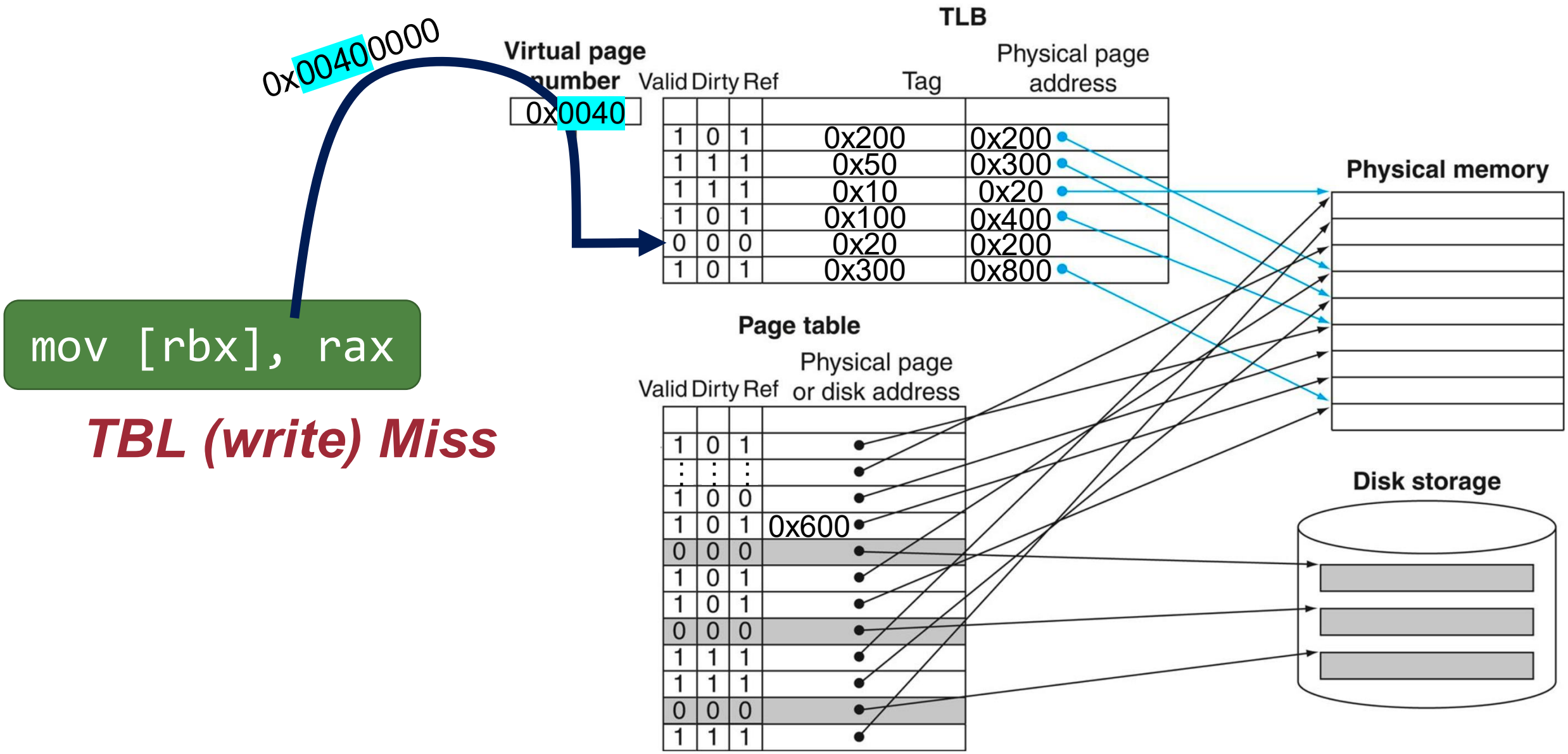
(2) Check if valid bit sets

TBL (write) Hit!

Case #1: TLB Hits

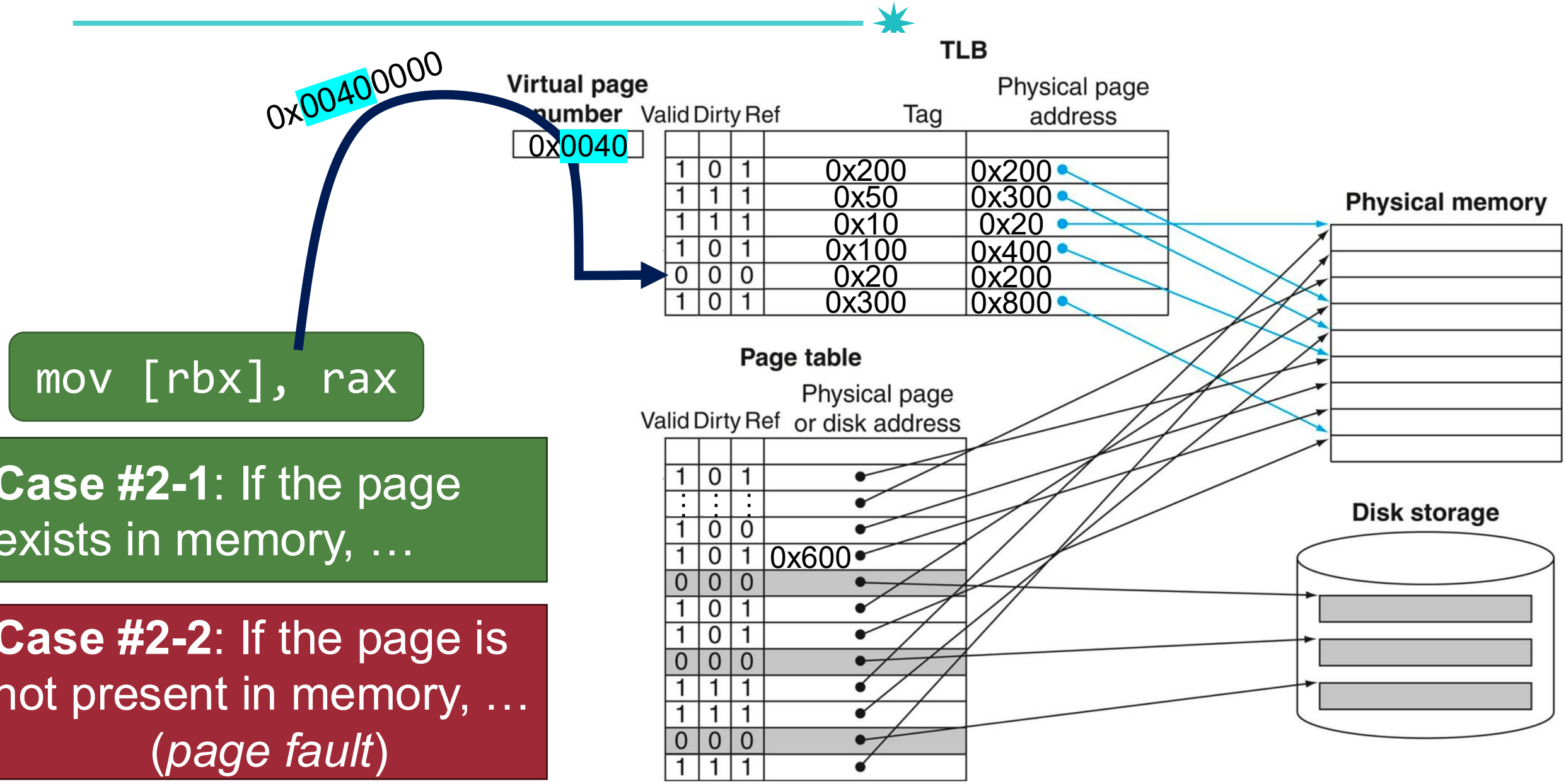


Case #2: TLB Misses



TBL (write) Miss

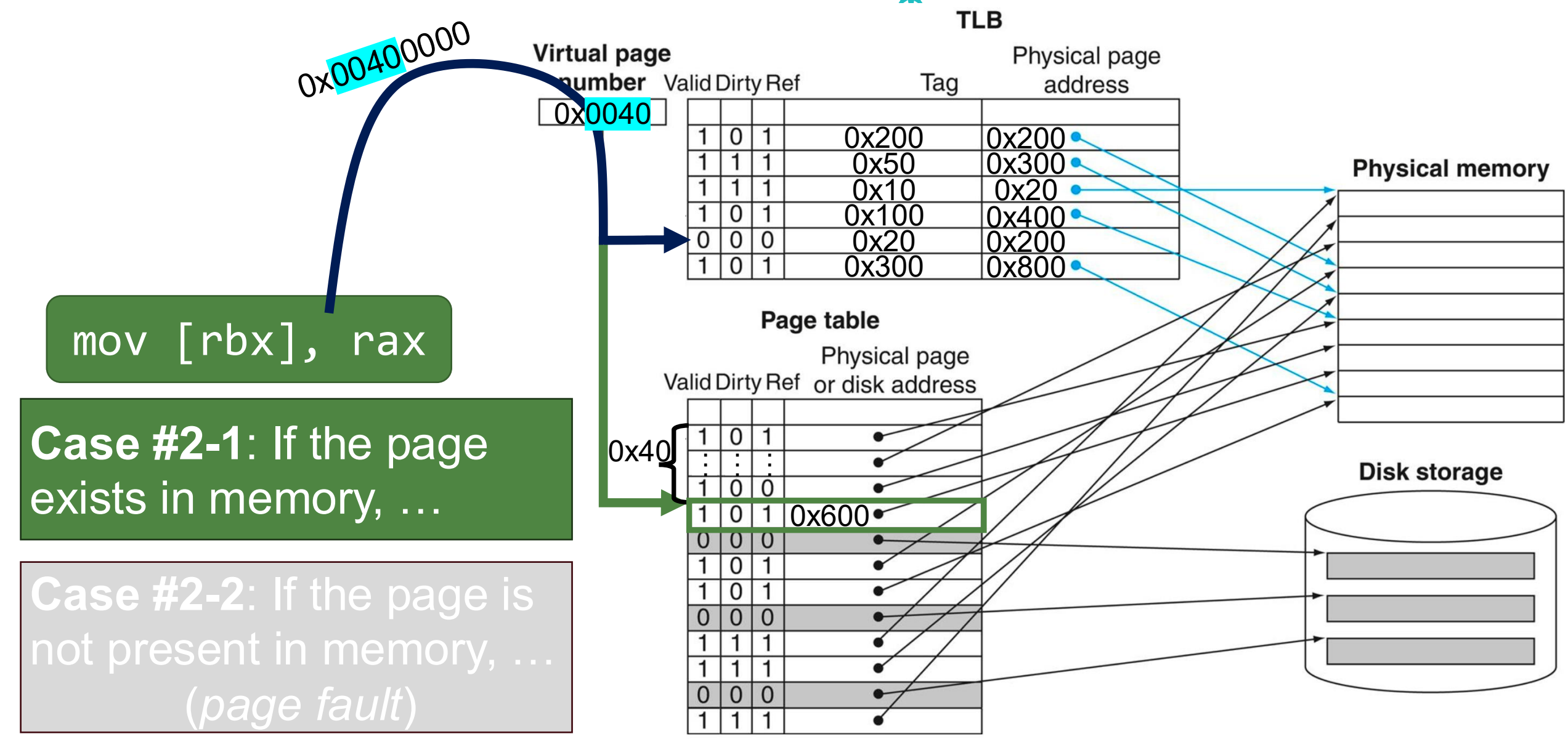
Case #2: TLB Misses



Case #2-1: If the page exists in memory, …

Case #2-2: If the page is not present in memory, … (*page fault*)

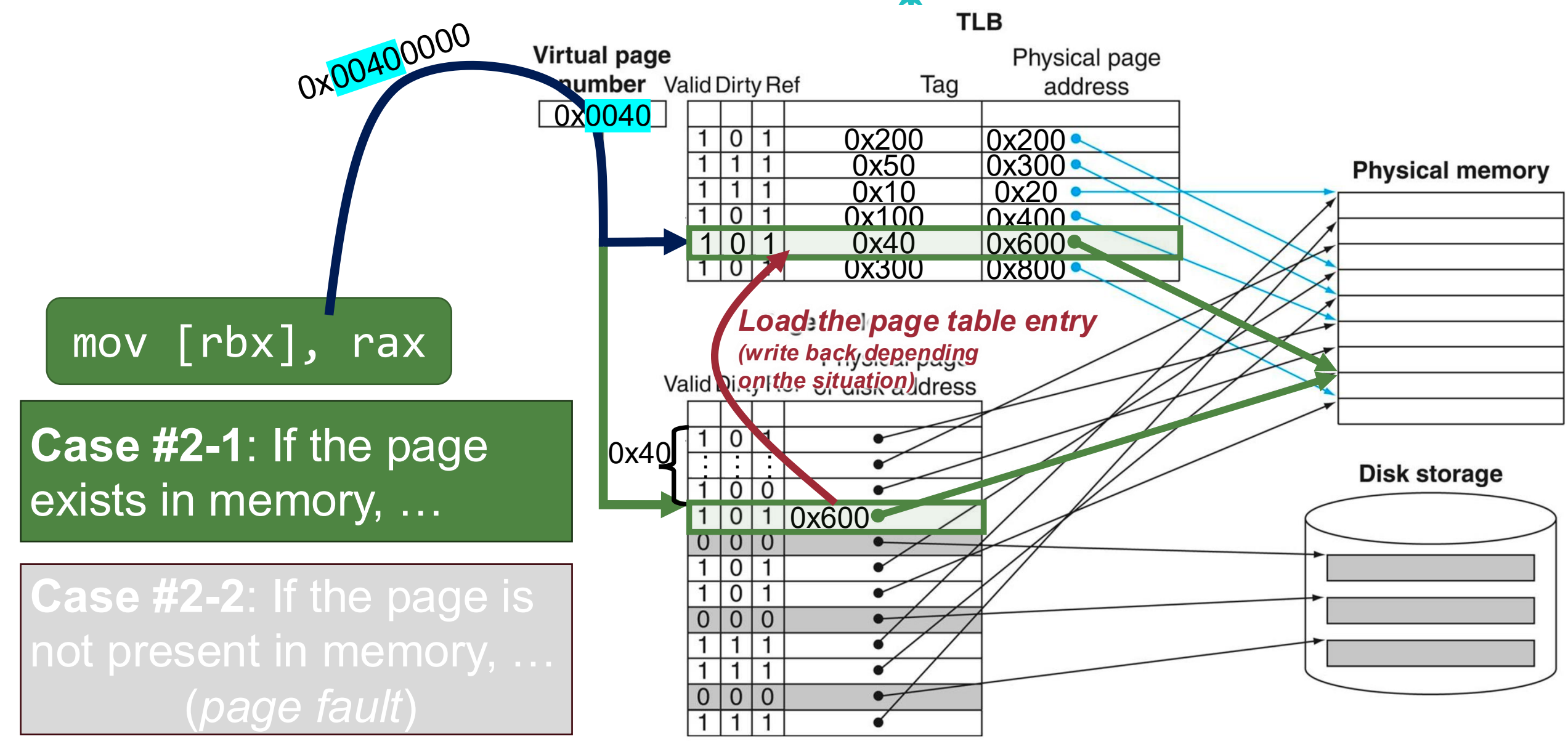
Case #2-1: TLB Misses – Memory Hit



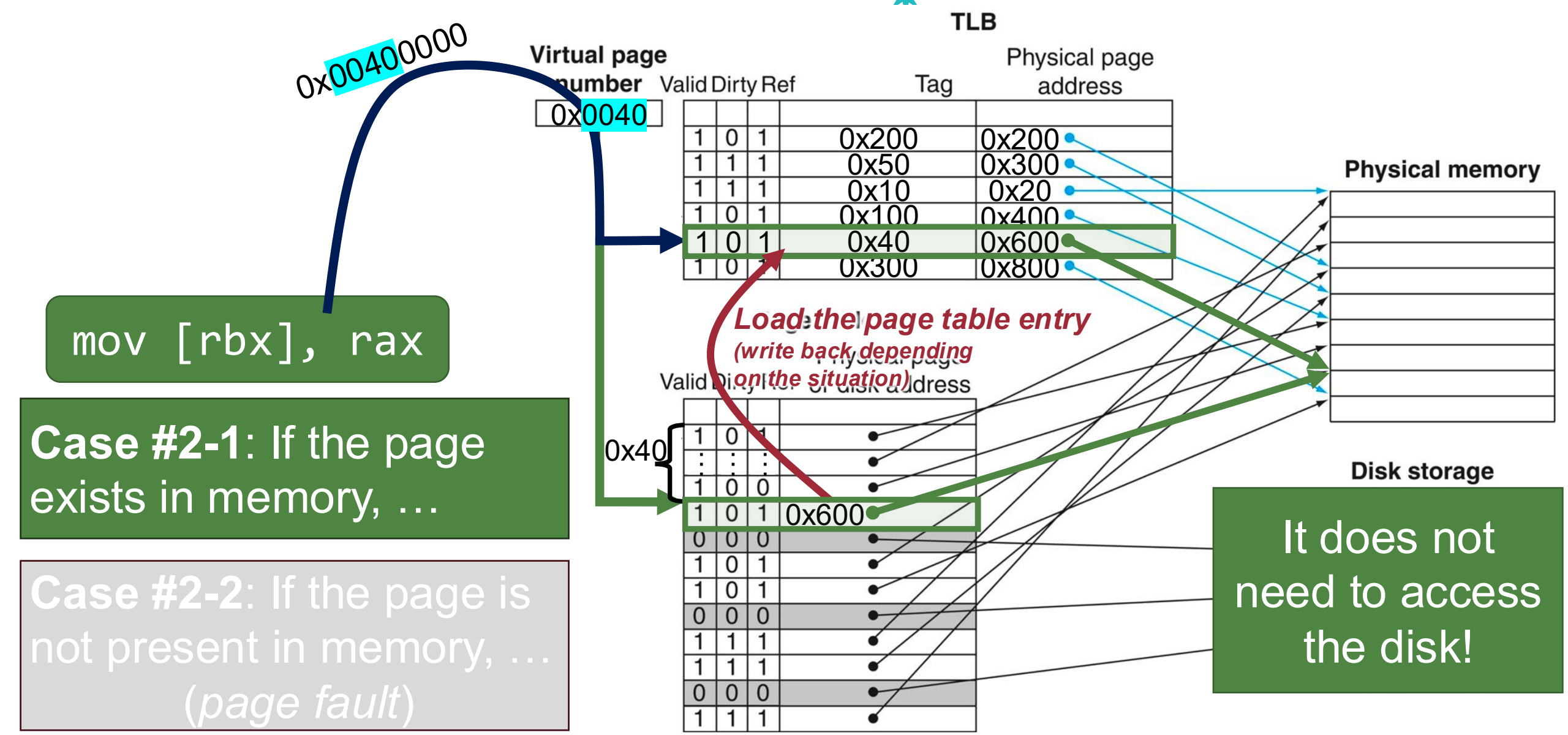
Case #2-1: If the page exists in memory, …

Case #2-2: If the page is not present in memory, … (*page fault*)

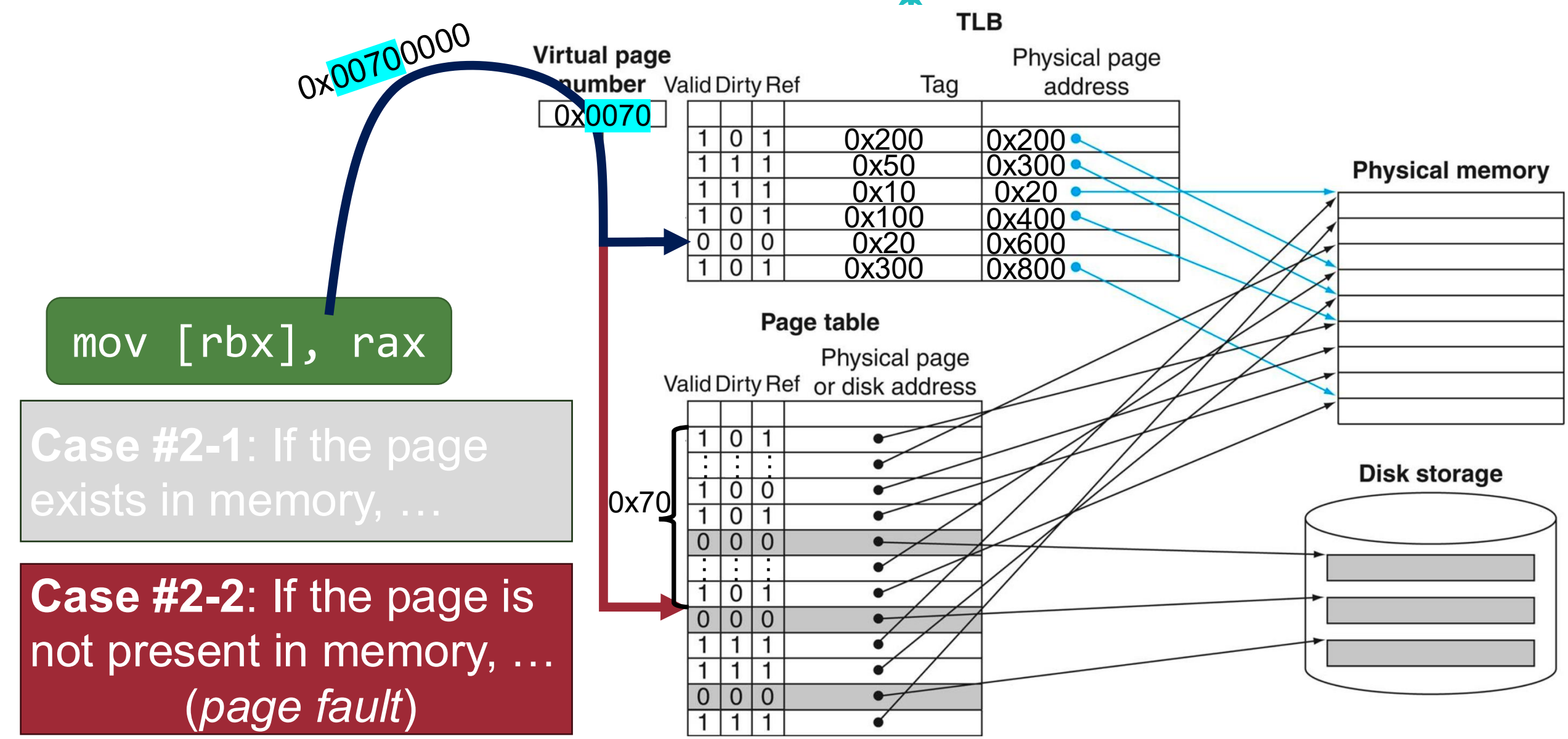
Case #2-1: TLB Misses – Memory Hit



Case #2-1: TLB Misses – Memory Hit



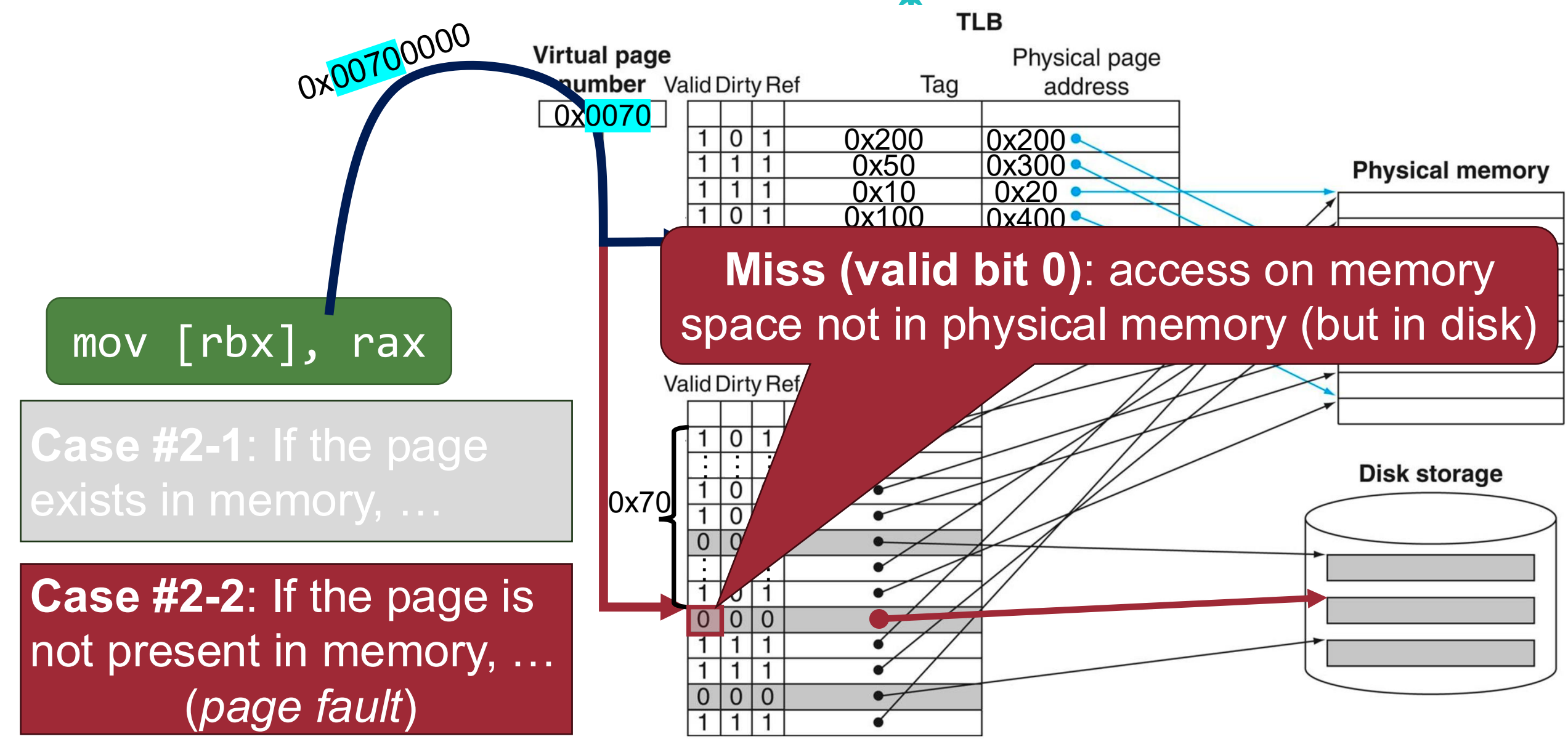
Case #2-2: TLB Misses – Page Fault



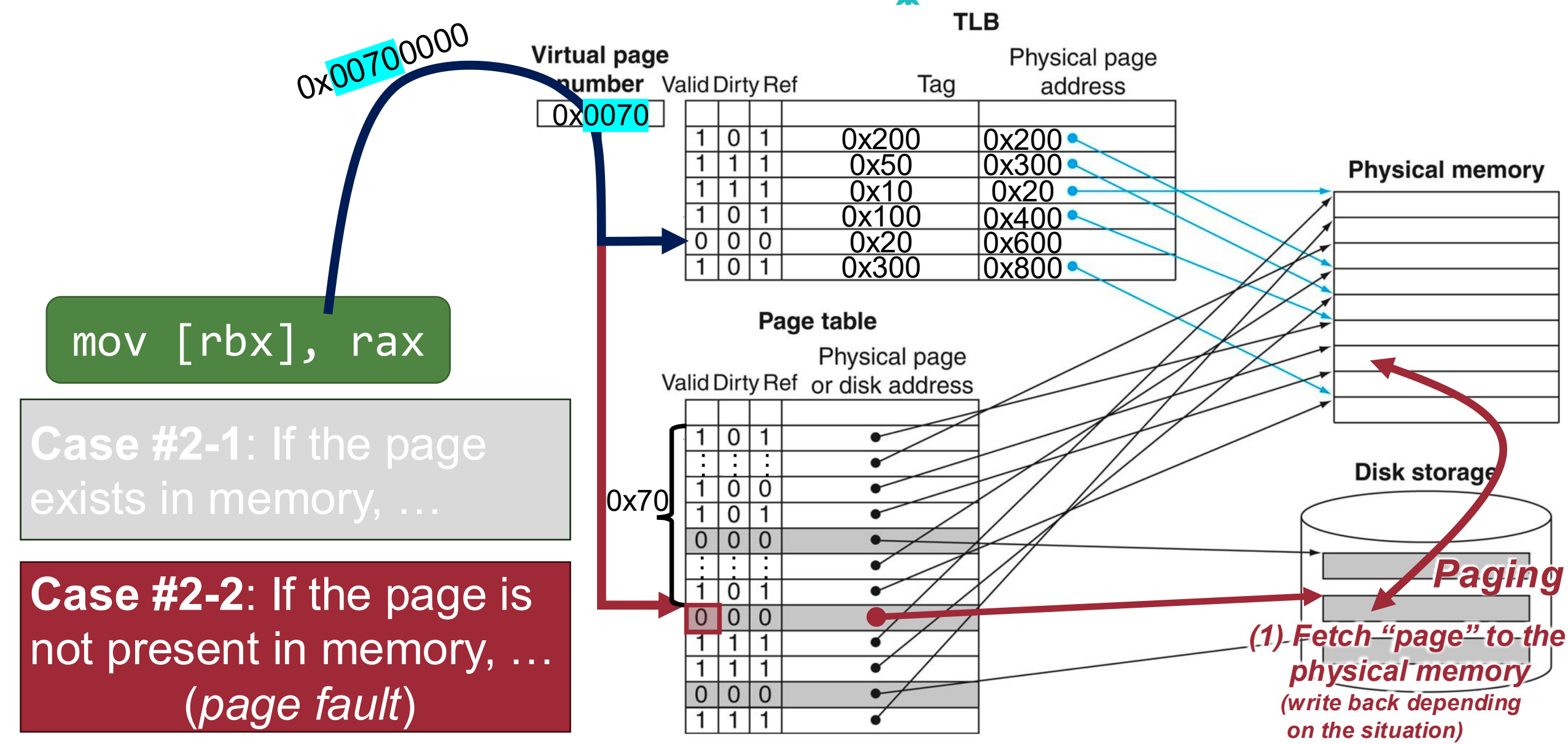
Case #2-1: If the page exists in memory, …

Case #2-2: If the page is not present in memory, … (*page fault*)

Case #2-2: TLB Misses – Page Fault



Case #2-2: TLB Misses – Page Fault

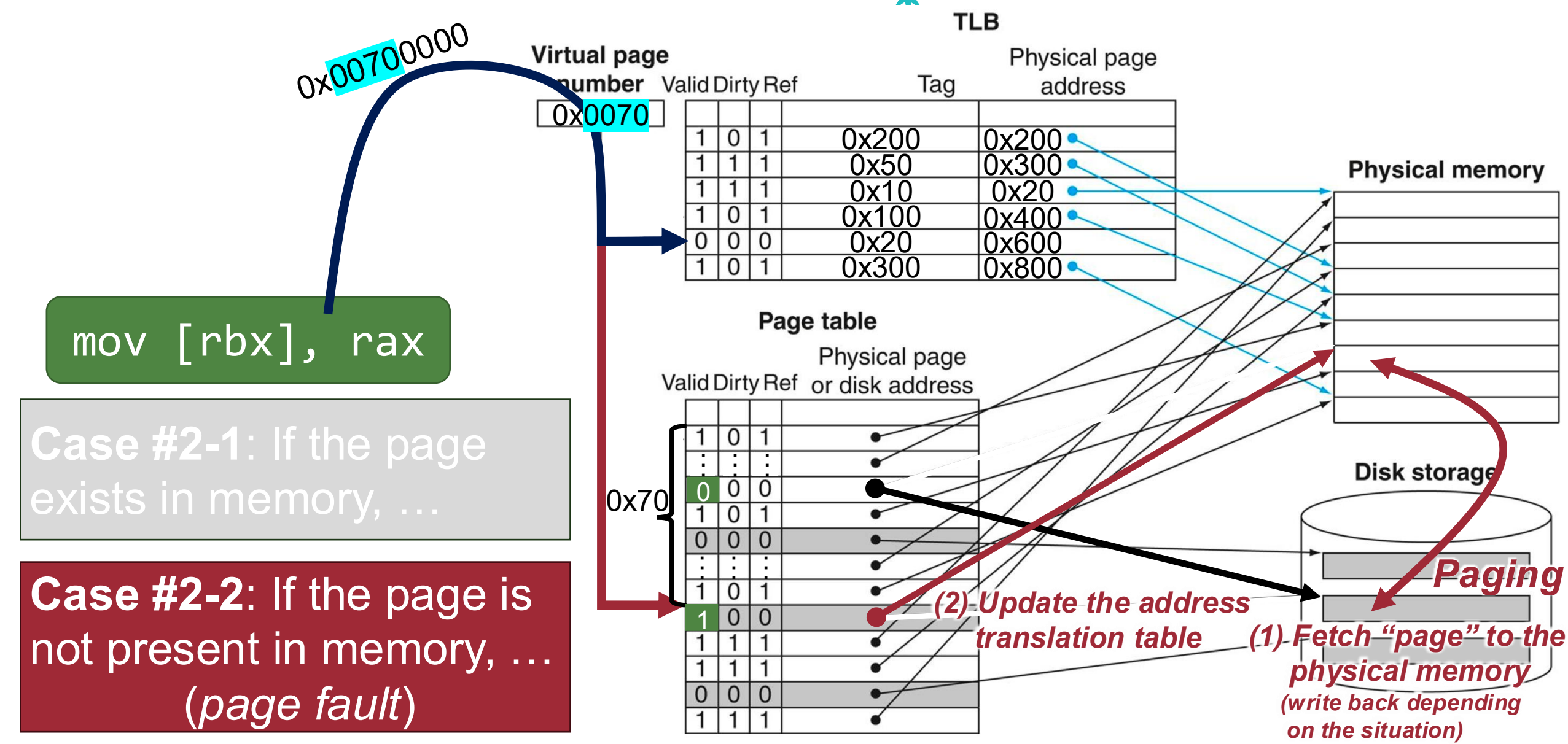



```
mov [rbx], rax
```

Case #2-1: If the page exists in memory, …

Case #2-2: If the page is not present in memory, … (*page fault*)

Case #2-2: TLB Misses – Page Fault

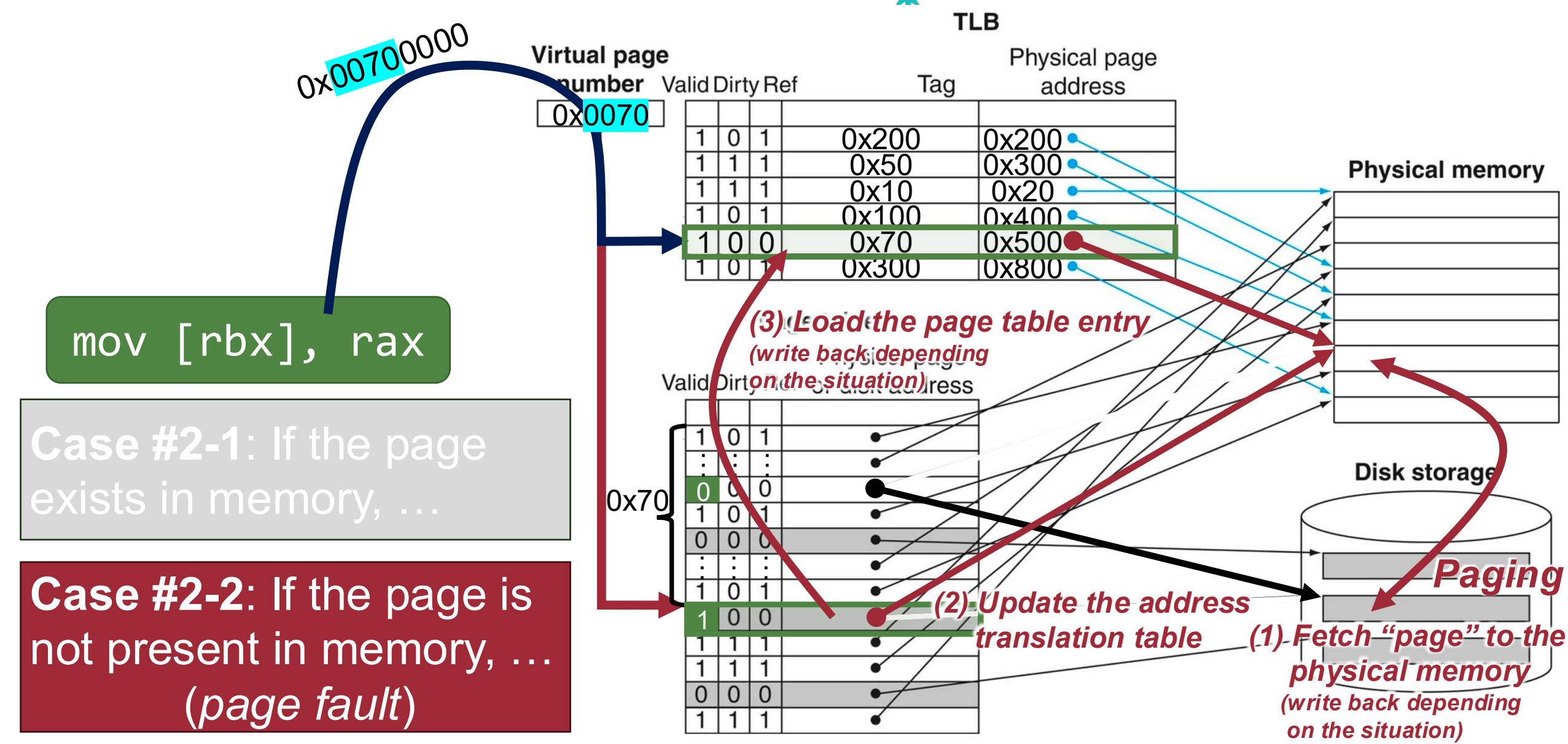



```
mov [rbx], rax
```

Case #2-1: If the page exists in memory, …

Case #2-2: If the page is not present in memory, … (*page fault*)

Case #2-2: TLB Misses – Page Fault



Case #2: TLB Misses

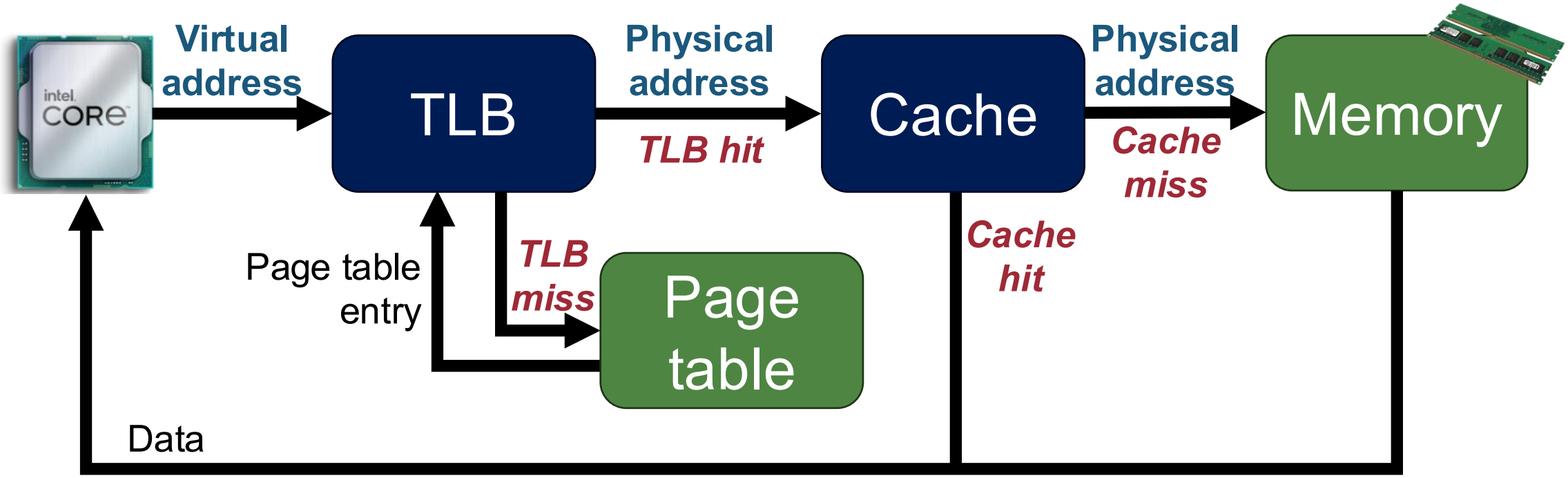
Case #2-1: If page is in memory

- Load the page table entry from main memory and retry
- Could be handled in hardware or in software
  - Raise a special exception, with optimized handler

Case #2-2: If page is not in memory (page fault)

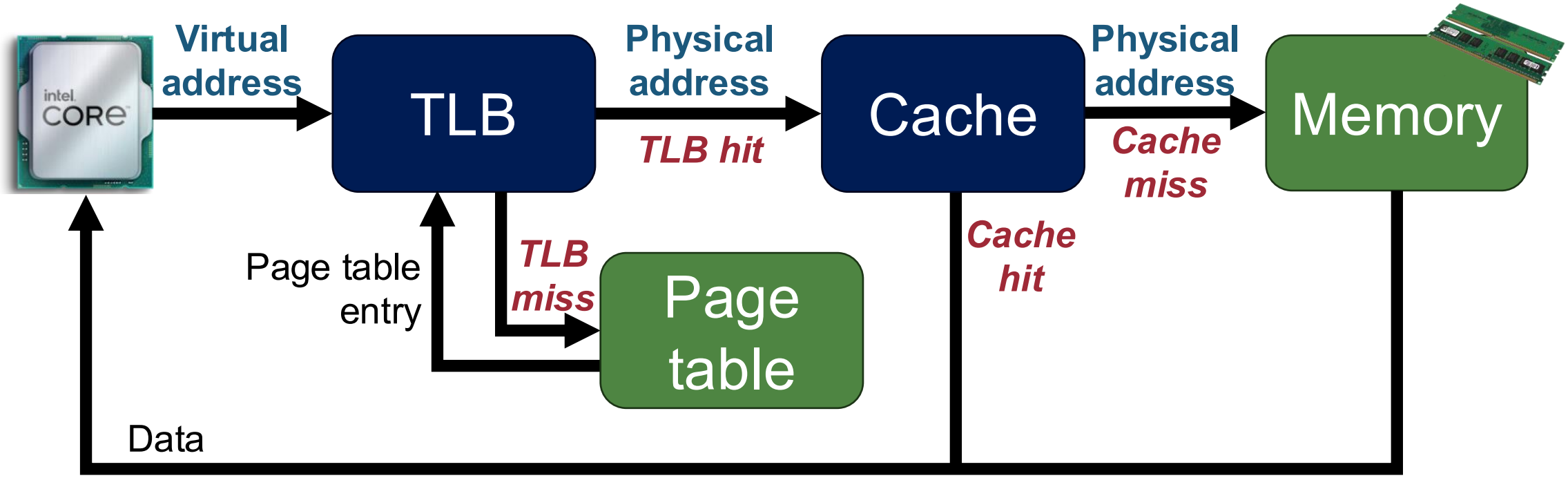
- OS handles fetching the page and updating the page table
- Then restart the faulting instruction

TLB and Cache Interaction



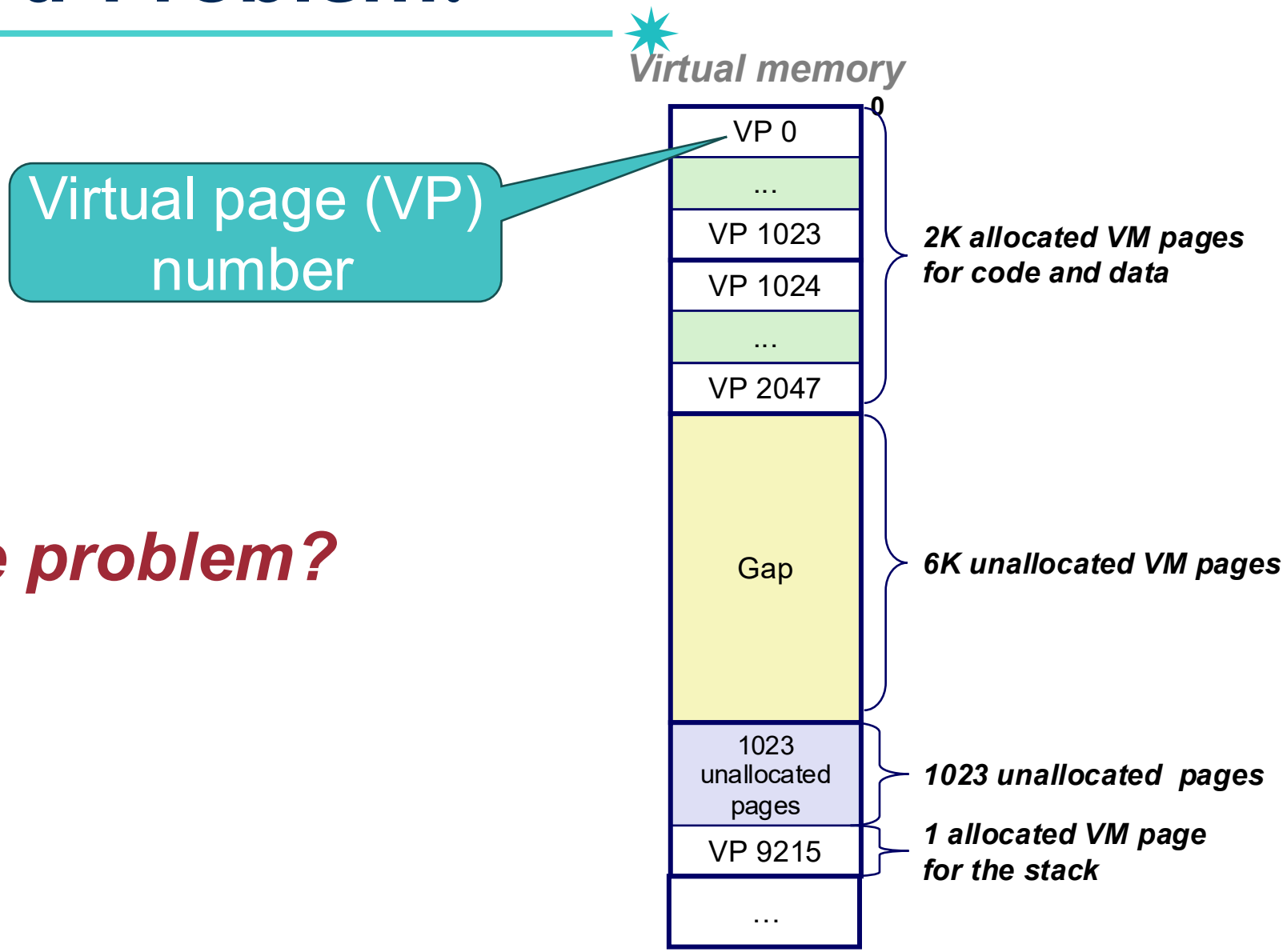
TLB and Cache Interaction

- Physically addressed caches
  - Cache tag uses physical address
    - Always translate before cache lookup
  - Allows multiple processes to have blocks in cache at the same time
  - Allows multiple processes to share pages



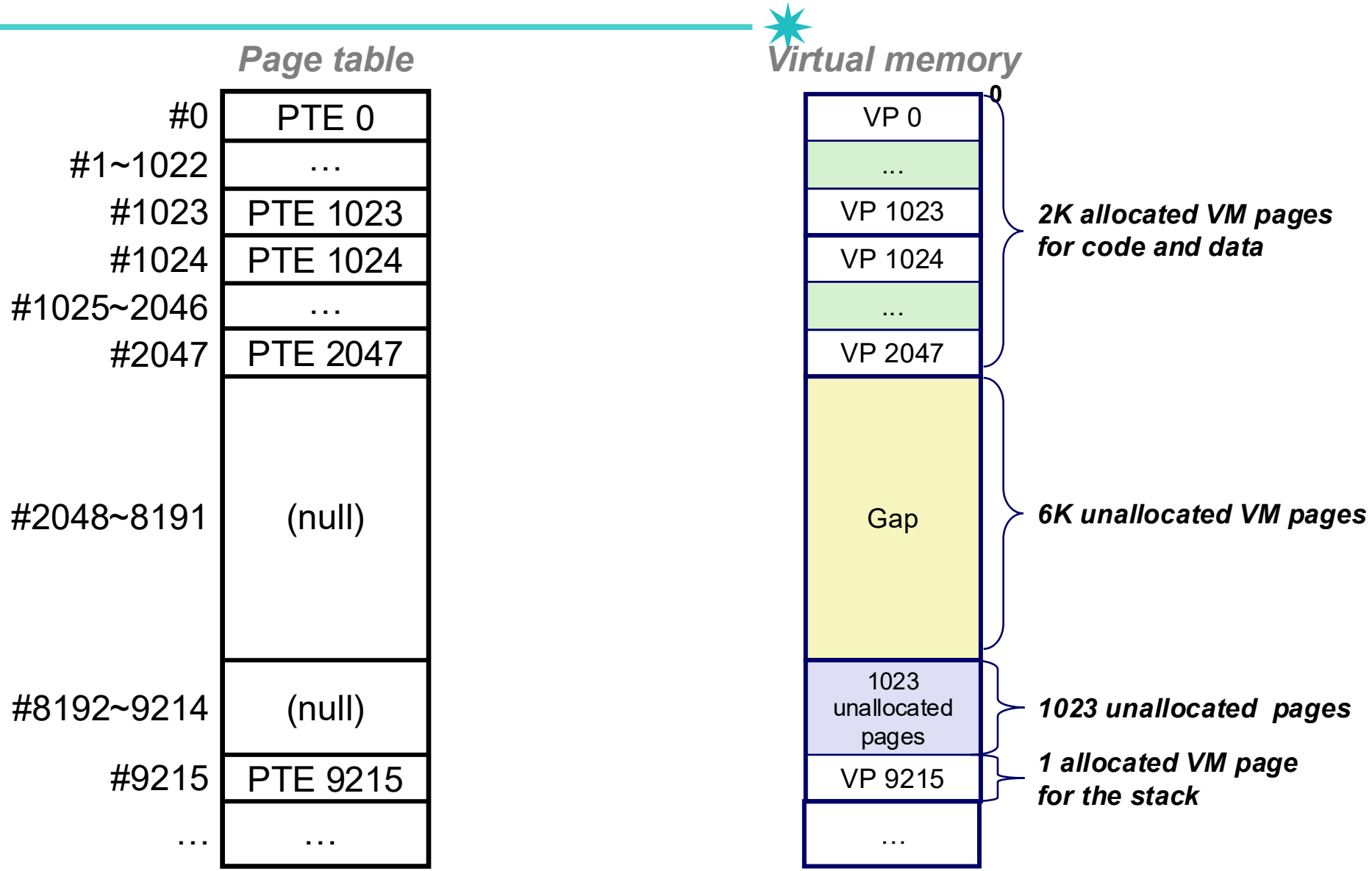
Multi-level Page Tables

We Have a Problem!

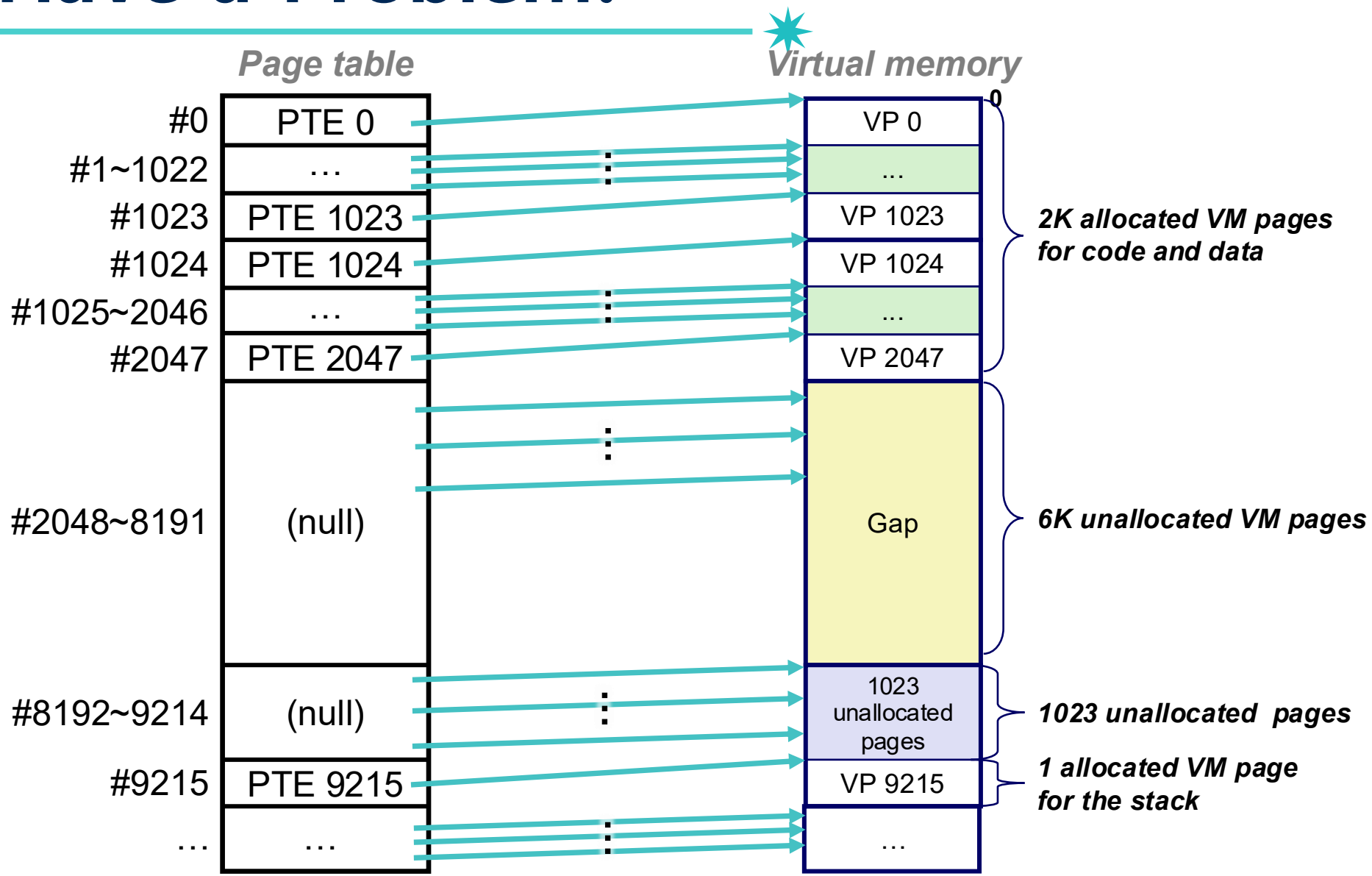


What is the problem?

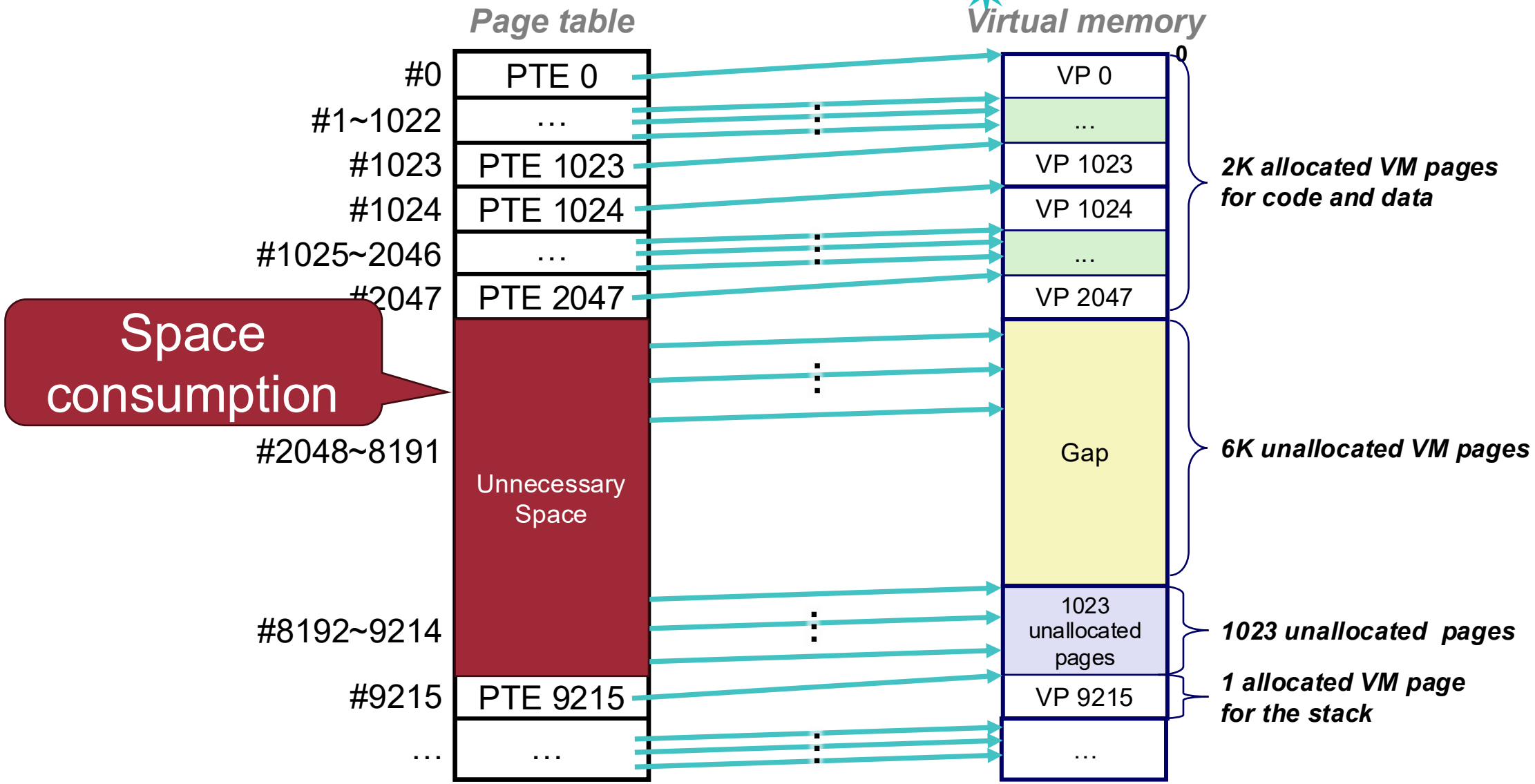
We Have a Problem!



We Have a Problem!

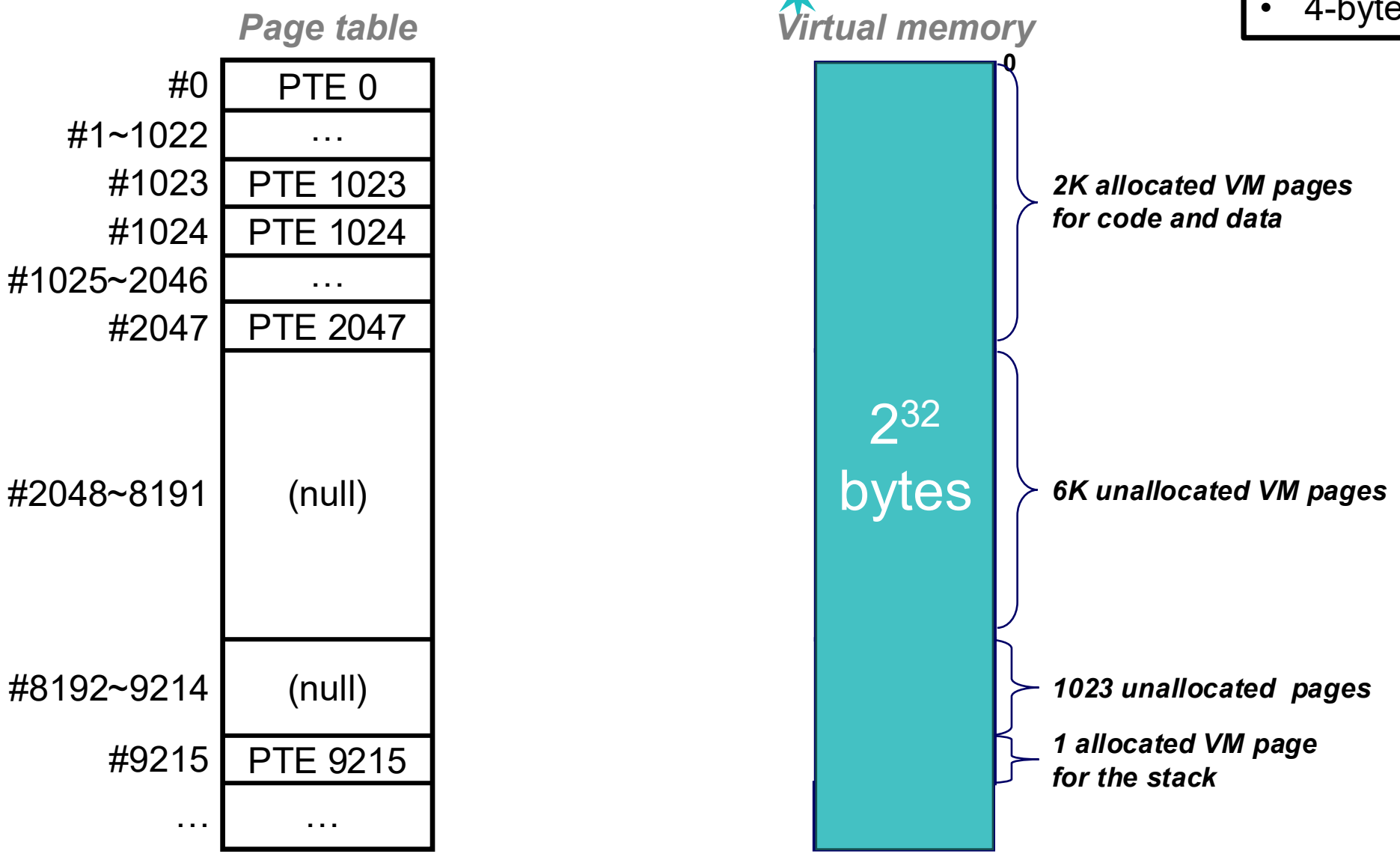


We Have a Problem!: Space Consumption



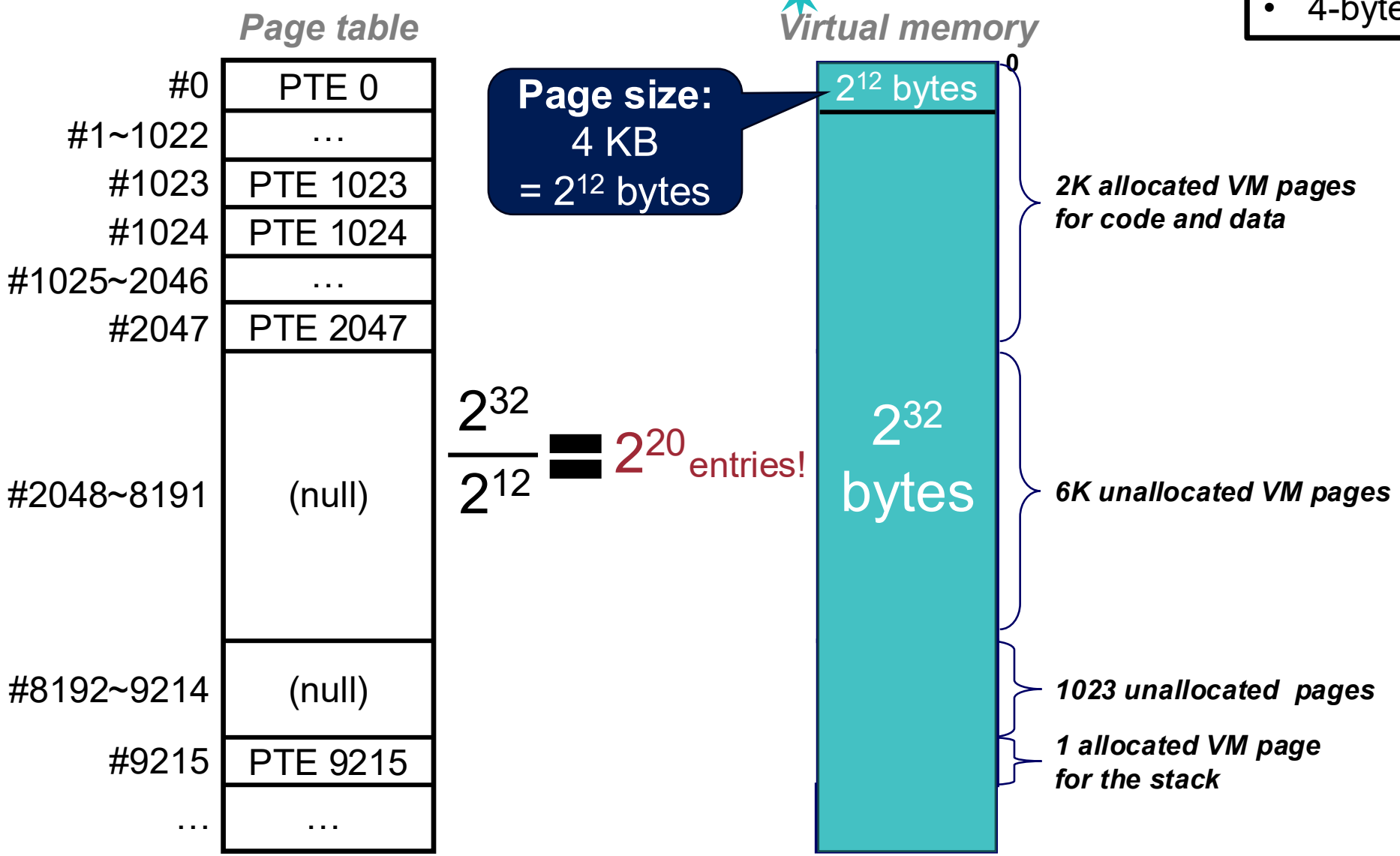
What Is the # of Page Table Entries?

Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



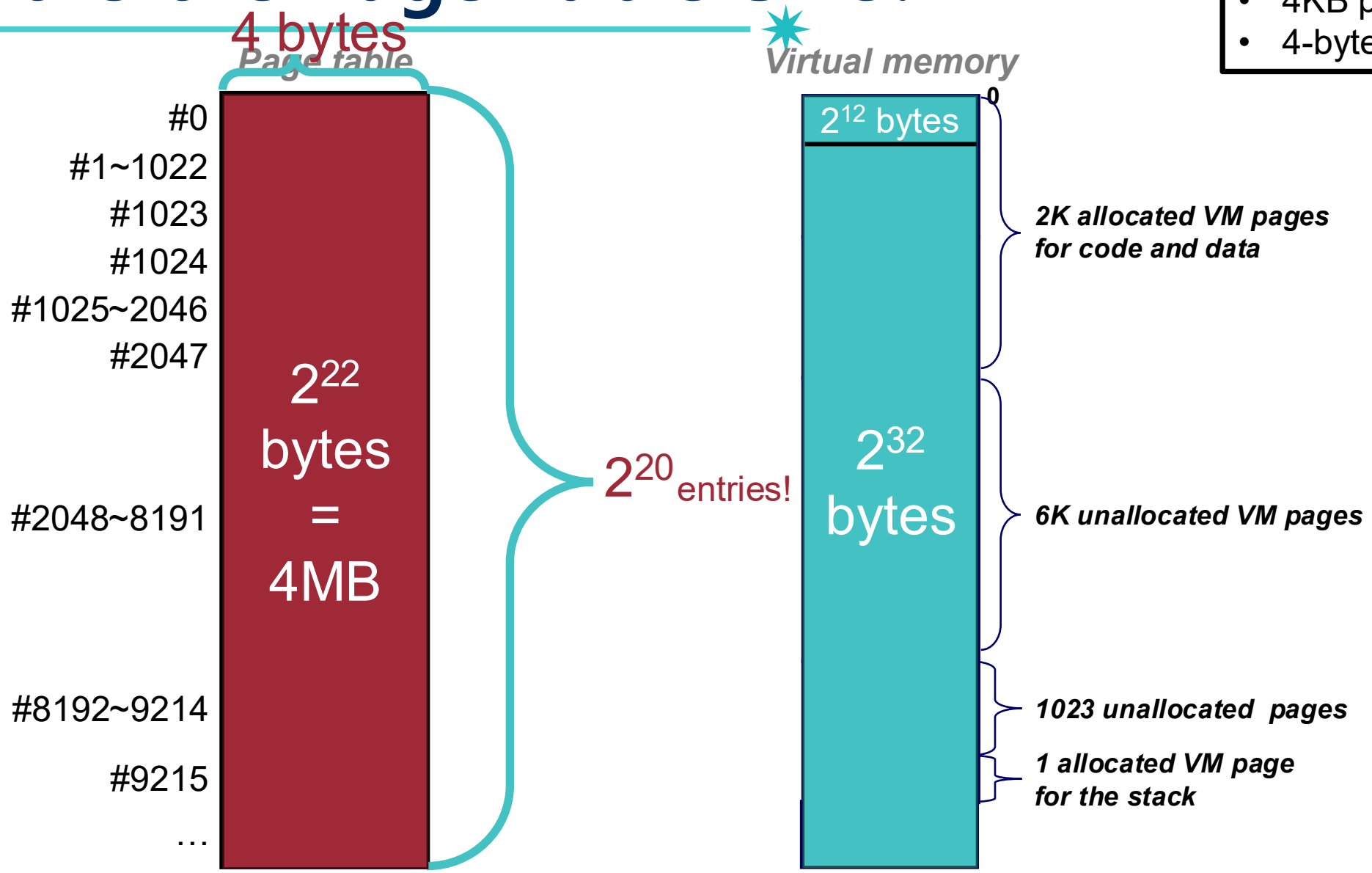
What Is the # of Page Table Entries?

Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



What Is the Page Table Size?

Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



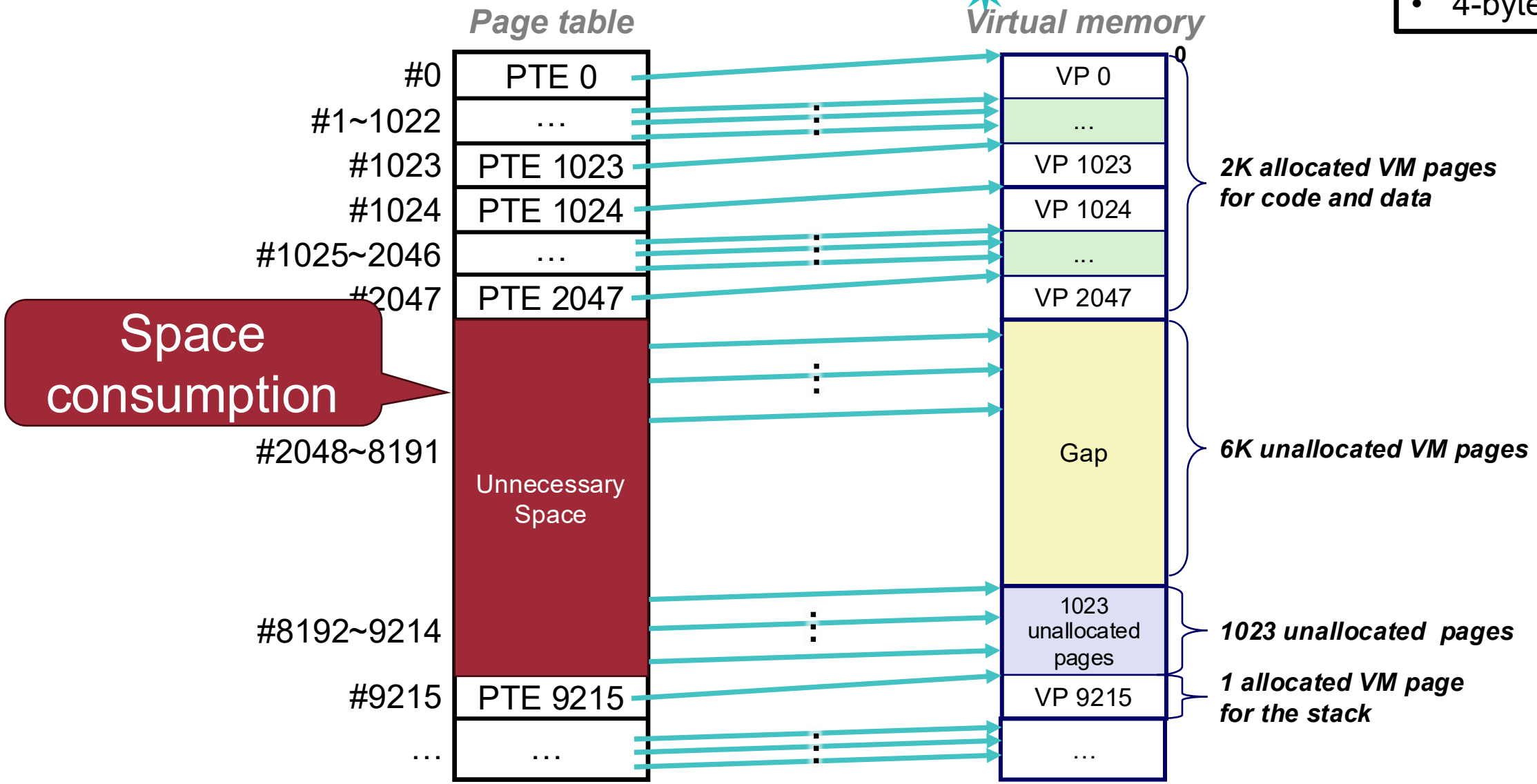
Limitation of a Single-level Page Table

- Suppose #1:
  - 32 bit addresses, 4KB pages, 4-byte PTEs
  - For each program, we need a 4MB page table ☹
- Suppose #2:
  - 42 bit addresses, 4KB pages, 8-byte PTEs
  - For each program, we need a 512GB page table ☹

Waste substantial memory because it must allocate entries for the entire virtual address space, even when most pages are unused

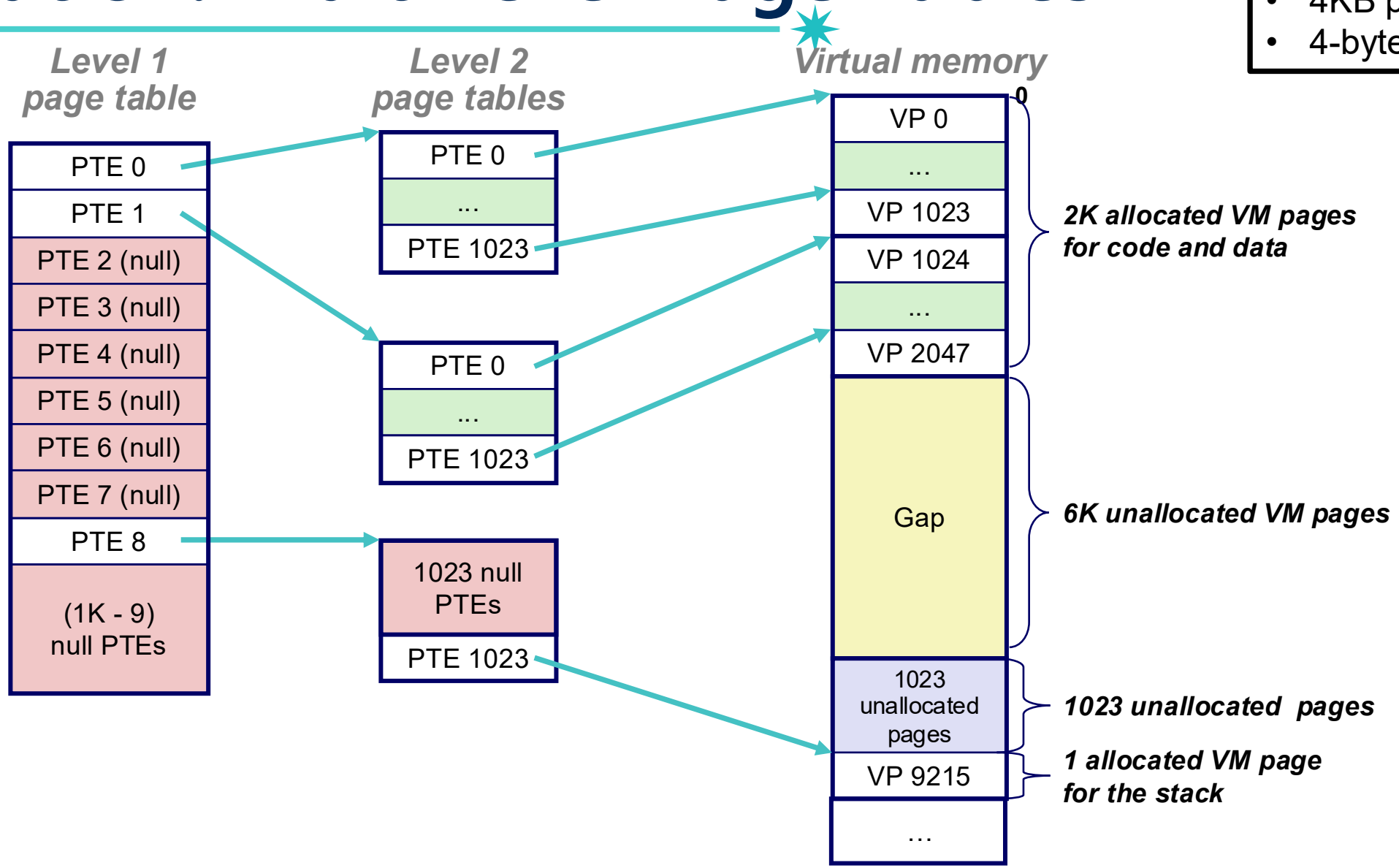
Root Cause of Space Bloating

Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



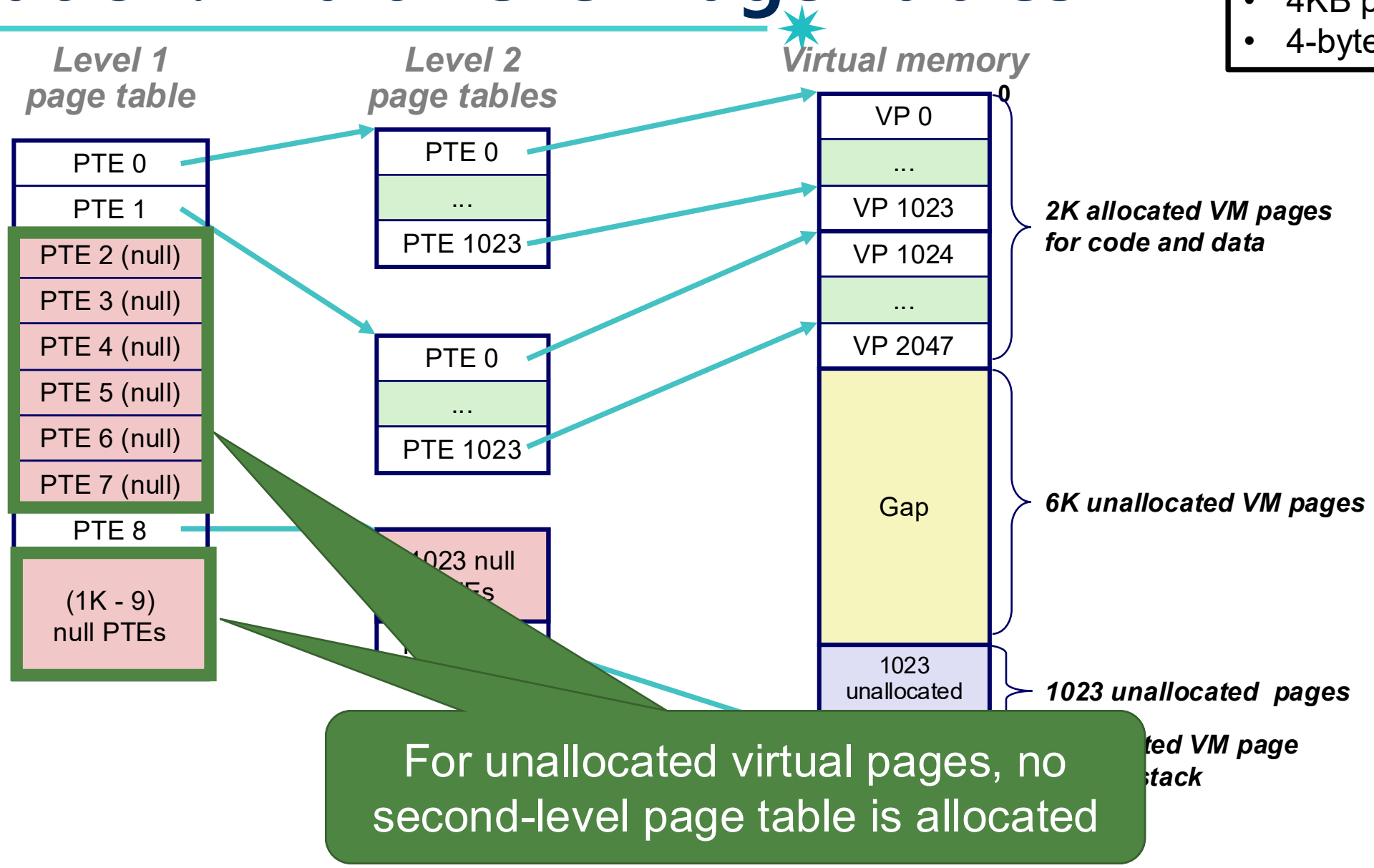
Solution: Multi-level Page Tables

Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



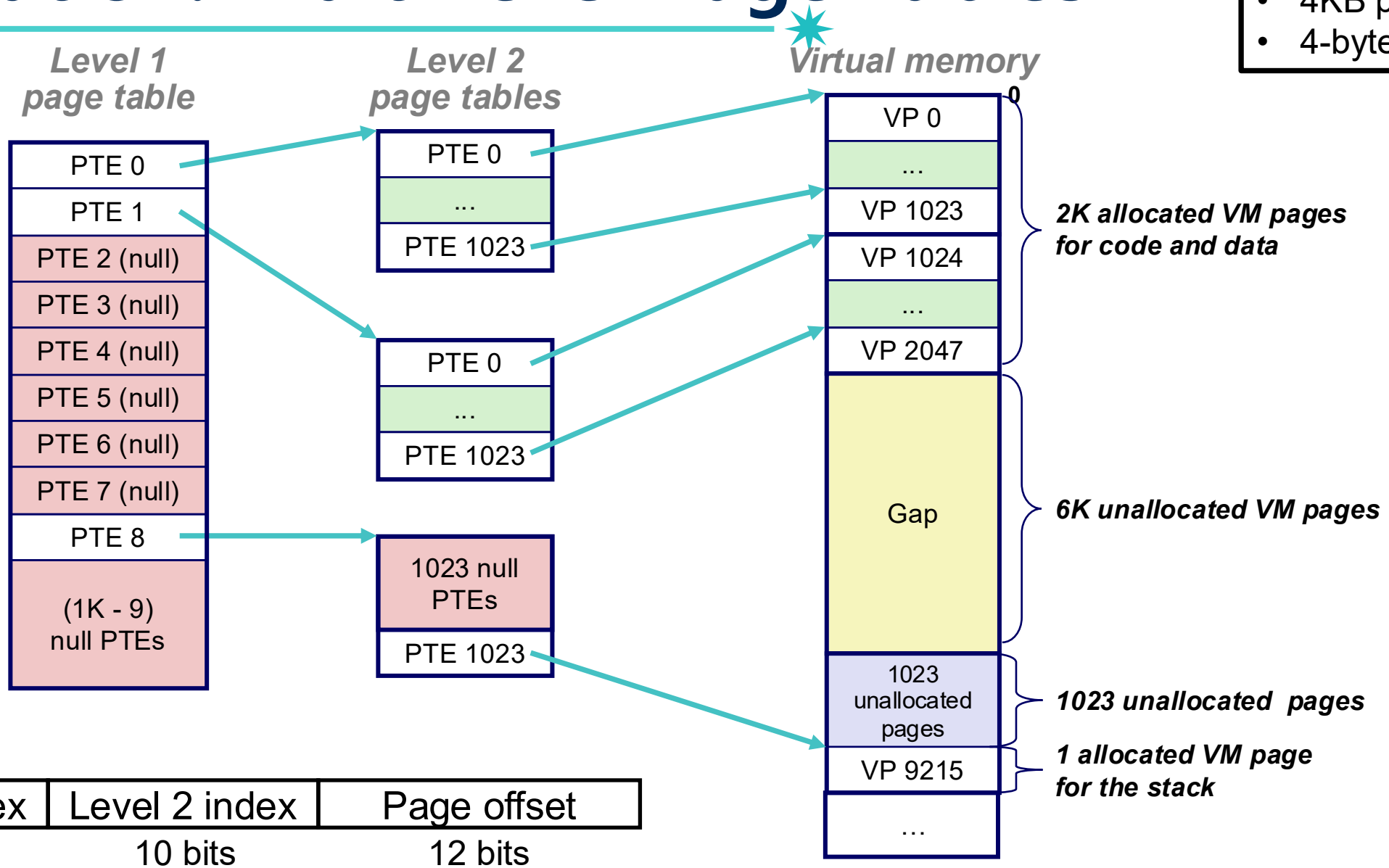
Solution: Multi-level Page Tables

Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



Solution: Multi-level Page Tables

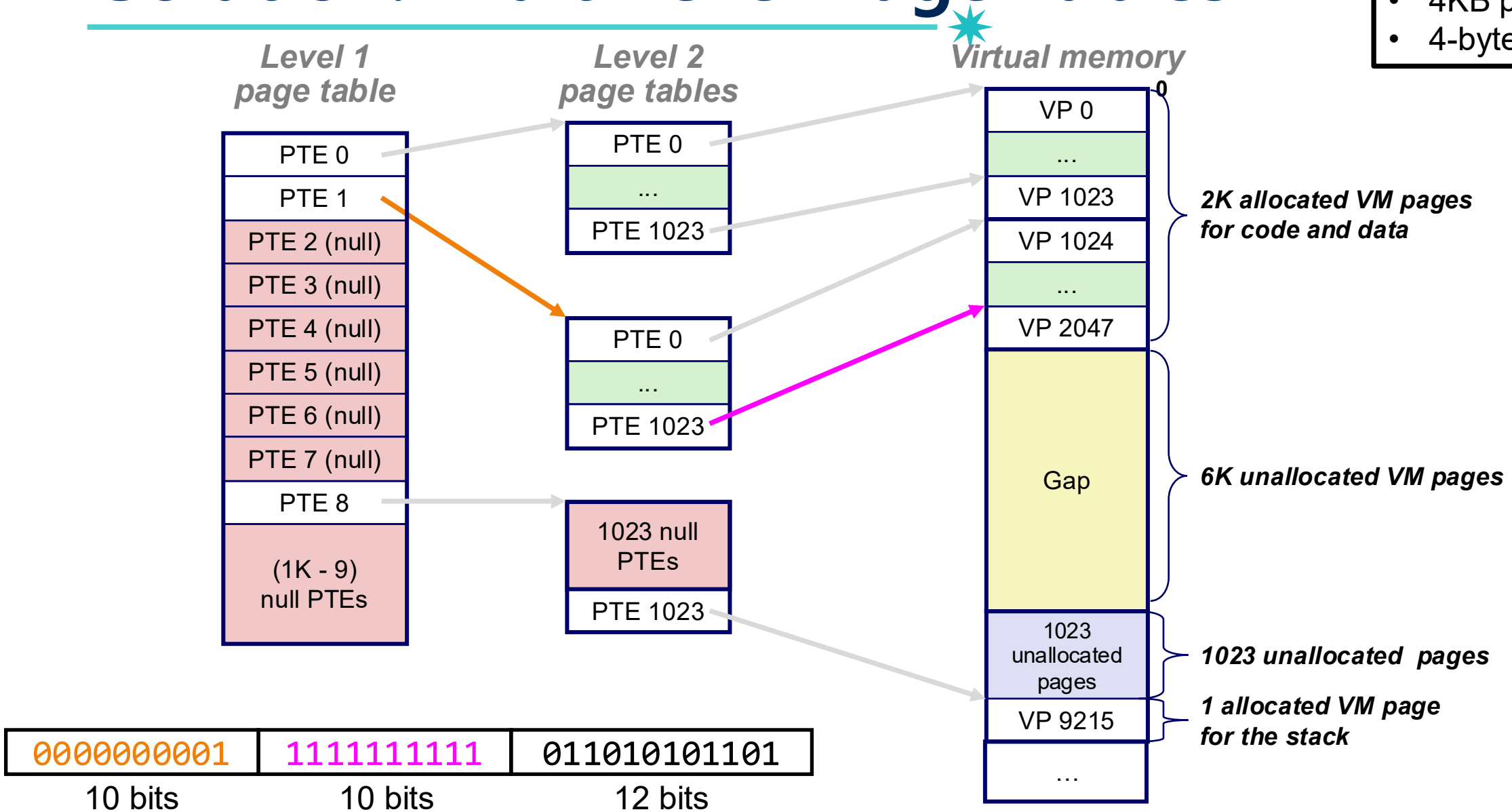
Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



Level 1 index	Level 2 index	Page offset
10 bits	10 bits	12 bits

Solution: Multi-level Page Tables

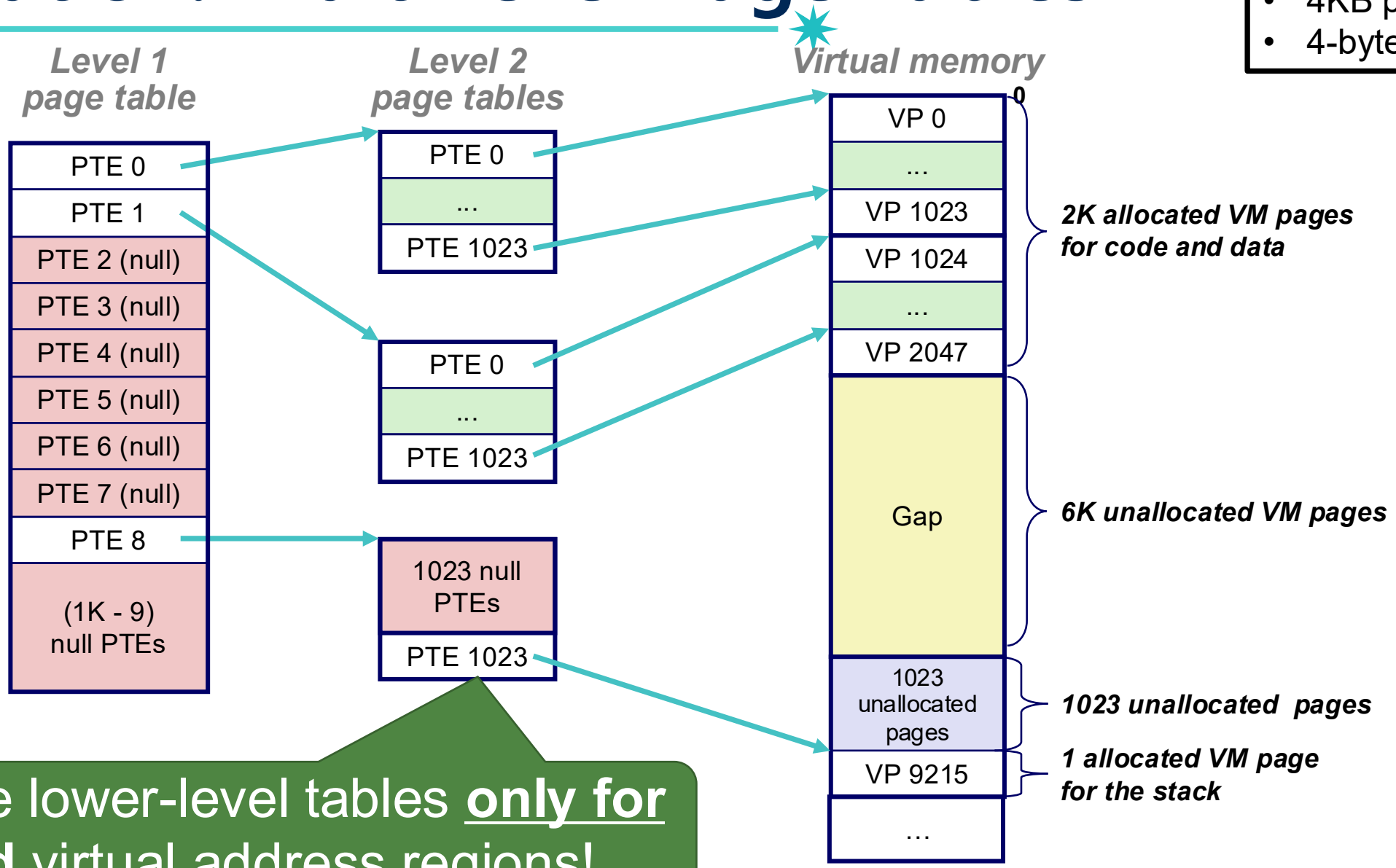
Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



0000000001	1111111111	011010101101
10 bits	10 bits	12 bits

Solution: Multi-level Page Tables

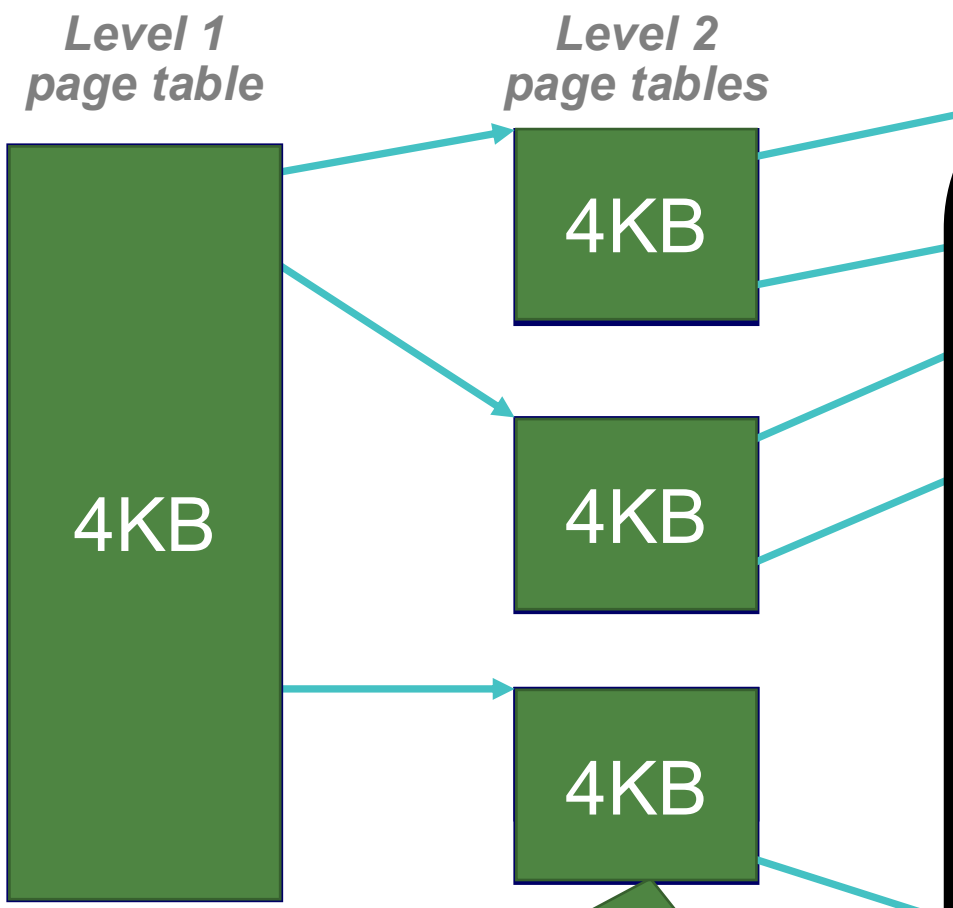
Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



Allocate lower-level tables **only for used** virtual address regions!

Solution: Multi-level Page Tables

Assumption:
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



Page table size:

4MB (single-level) →

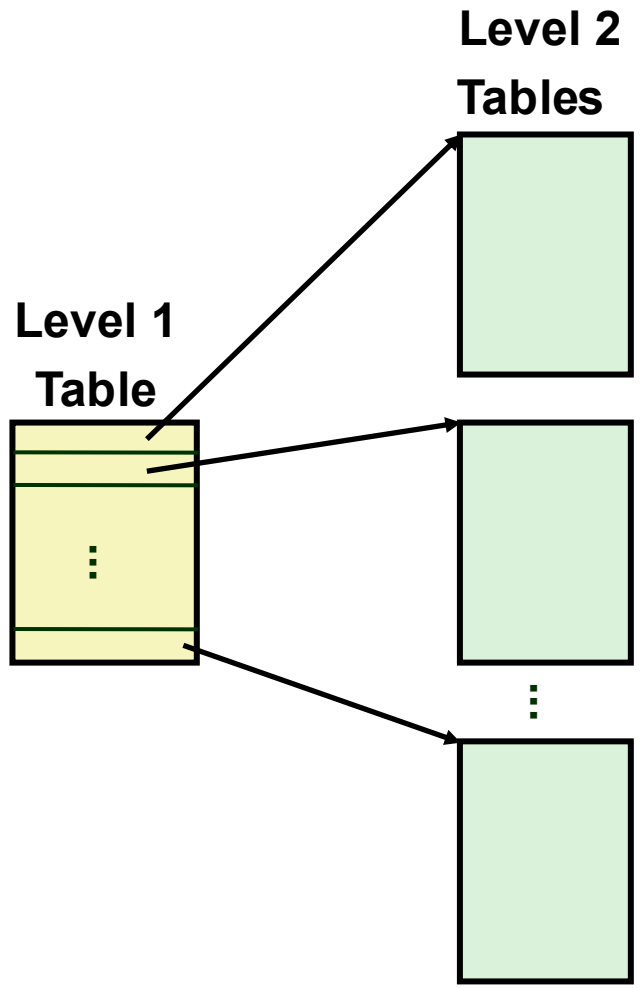
16KB (multi-level)

Allocate lower-level tables **only for used** virtual address regions!

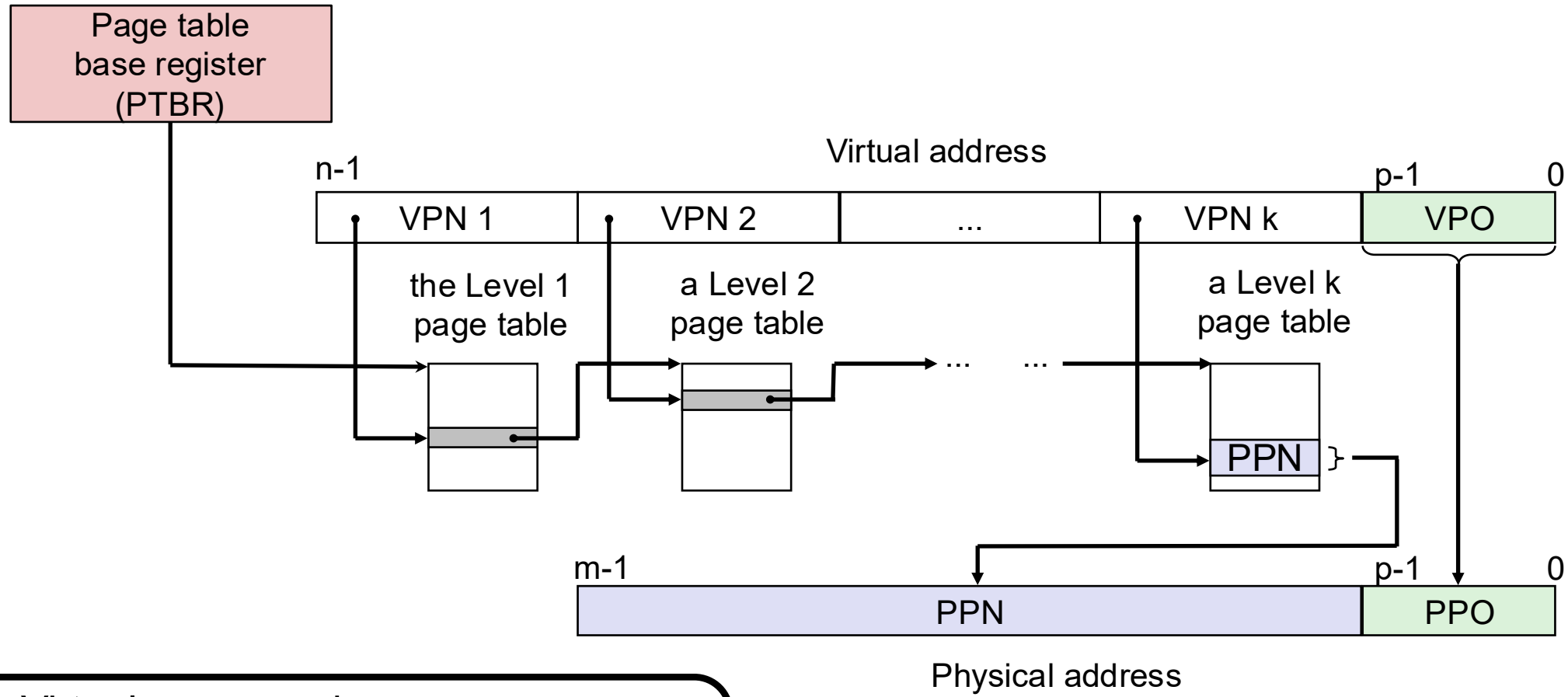
Multi-level Page Tables

- Organize page-table entries hierarchically
- Allocate lower-level tables only for used virtual address regions
  - ***So that we can save the space of page tables!***

- Example: 2-level page table
  - Level-1 table: each PTE points to a level-2 page table
  - Level-2 table: each PTE points to a page



Translating with a K-level Page Table



- VPN: Virtual page number
- VPO: Virtual page offset
- PPN: Phisical page number
- PPO: Physical page offset (same as VPO)

TLBs and Multi-level Page Tables

- TLBs cache the complete virtual to physical mapping
  - Regardless of the levels of page tables, the TLB stores the VPN → PPN

Virtual Memory Summary

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

- System view of virtual memory
  - Uses memory efficiently by caching virtual memory pages
    - Efficient only because of locality
  - Simplifies memory management and programming
  - Simplifies protection

Question?