

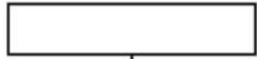
CSE251: System Programming

16. Virtual Memory (2)

Seongil Wi

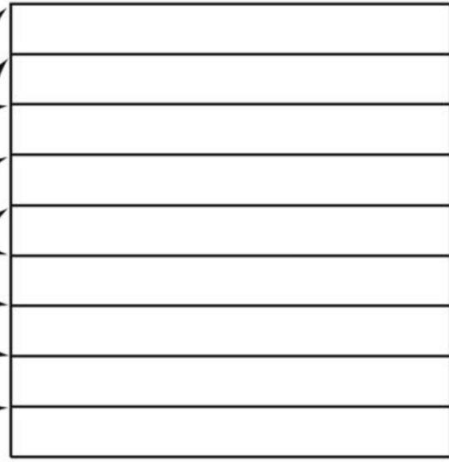
Place of the Page Table?

Virtual page number

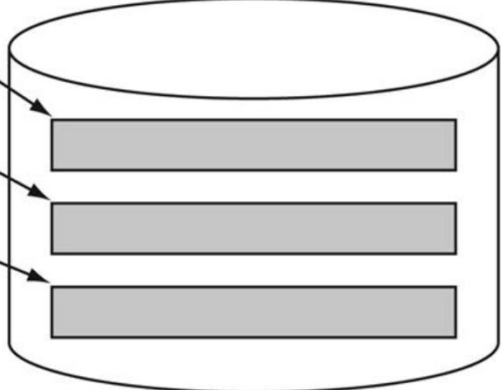


Valid	Physical page or disk address
1	•
1	•
1	•
1	•
0	•
1	•
1	•
0	•
1	•
1	•
0	•
1	•

Physical memory



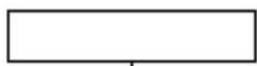
Disk storage



Where does the page table exist?
Physical memory!

Place of the Page Table?

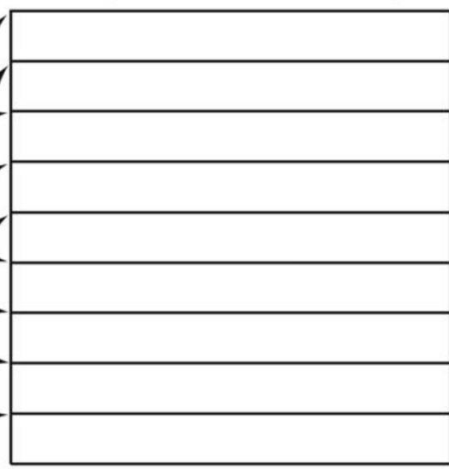
Virtual page number



Page table
Valid Physical page or disk address

Valid	Physical page or disk address
1	
1	
1	
1	
0	
1	
1	
0	

Physical memory



*Where does the page table exist?
Physical memory!*



Any problems?

Multiple memory access: one memory access to obtain the physical address and a second access to get the data

Problem: Multiple Memory Access

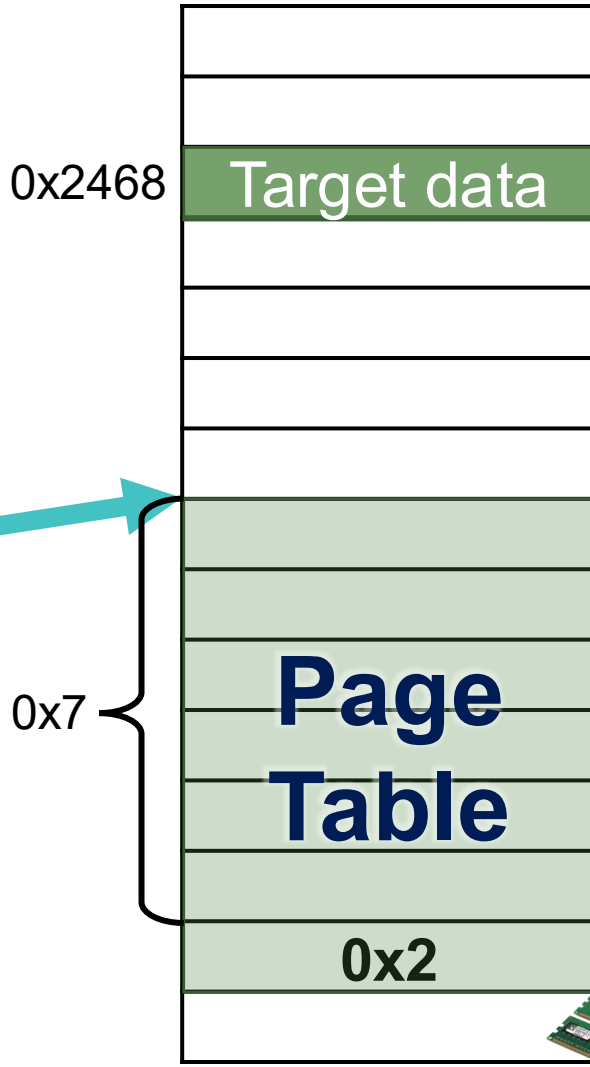
Access to 0x7468
(Assumption: virtual page number is 0x7, page offset is 0x468)



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

Page table register

Processor



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

0x2468

Target data

0x7

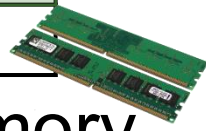
Page Table

0x2

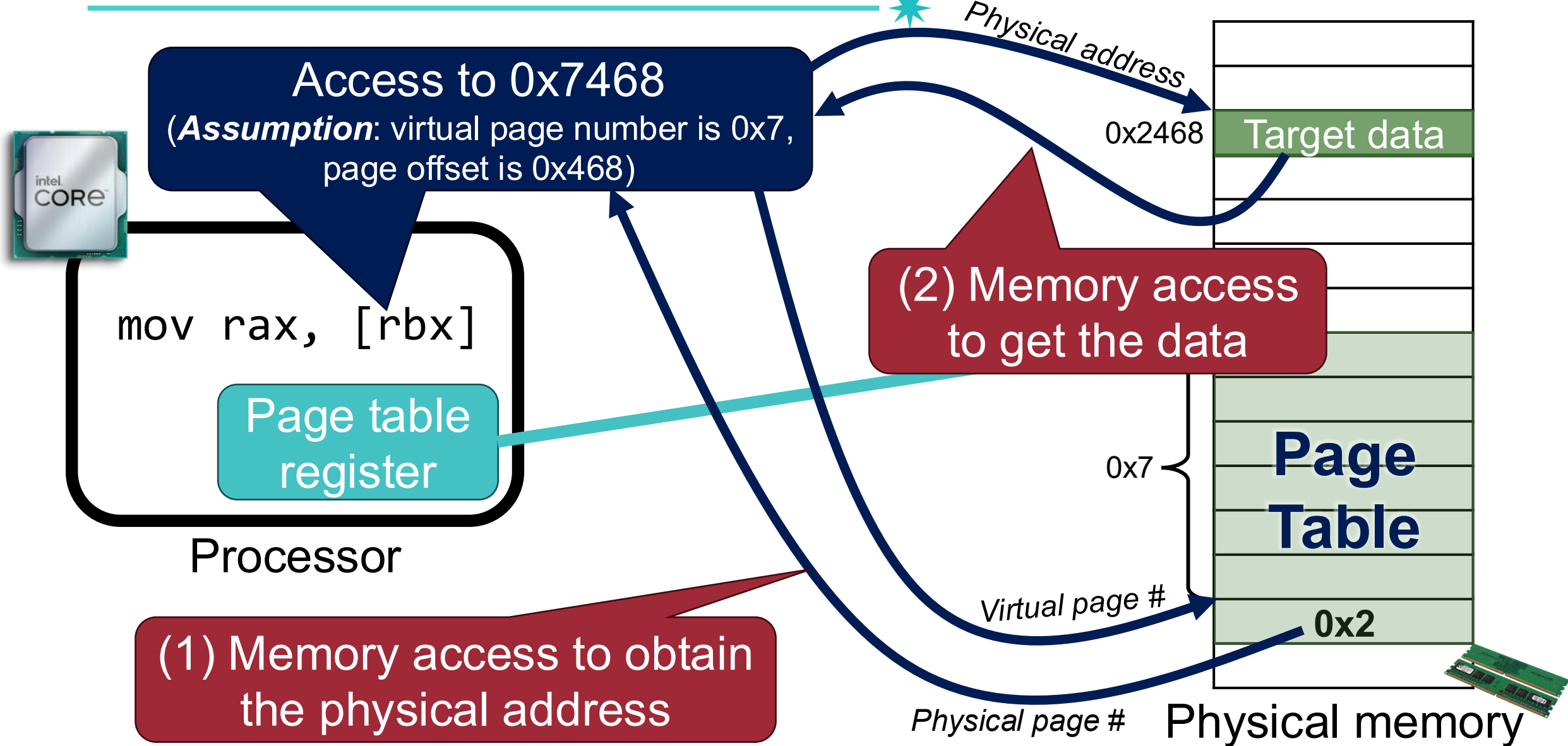
Virtual page #

Physical page #

Physical memory

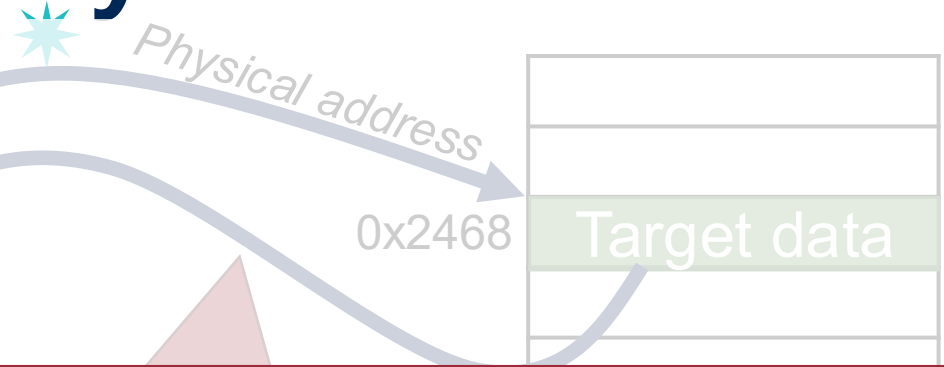


Problem: Multiple Memory Access



Problem: Multiple Memory Access

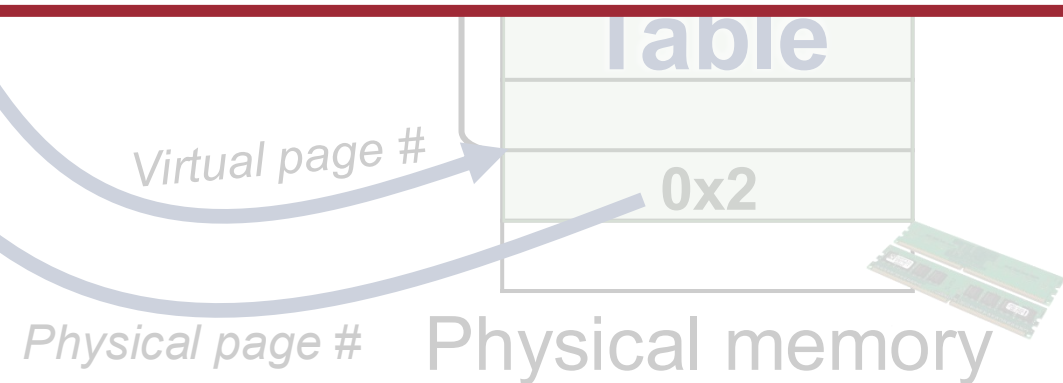
Access to 0x7468
(Assumption: virtual page number is 0x7,
page offset is 0x468)



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

Processor

(1) Memory access to obtain the physical address



Solution: A Cache for Address Translation ⁸



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

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

Page table register

Processor

Translation cache

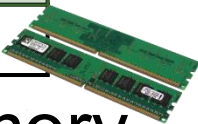
0x2468

Target data

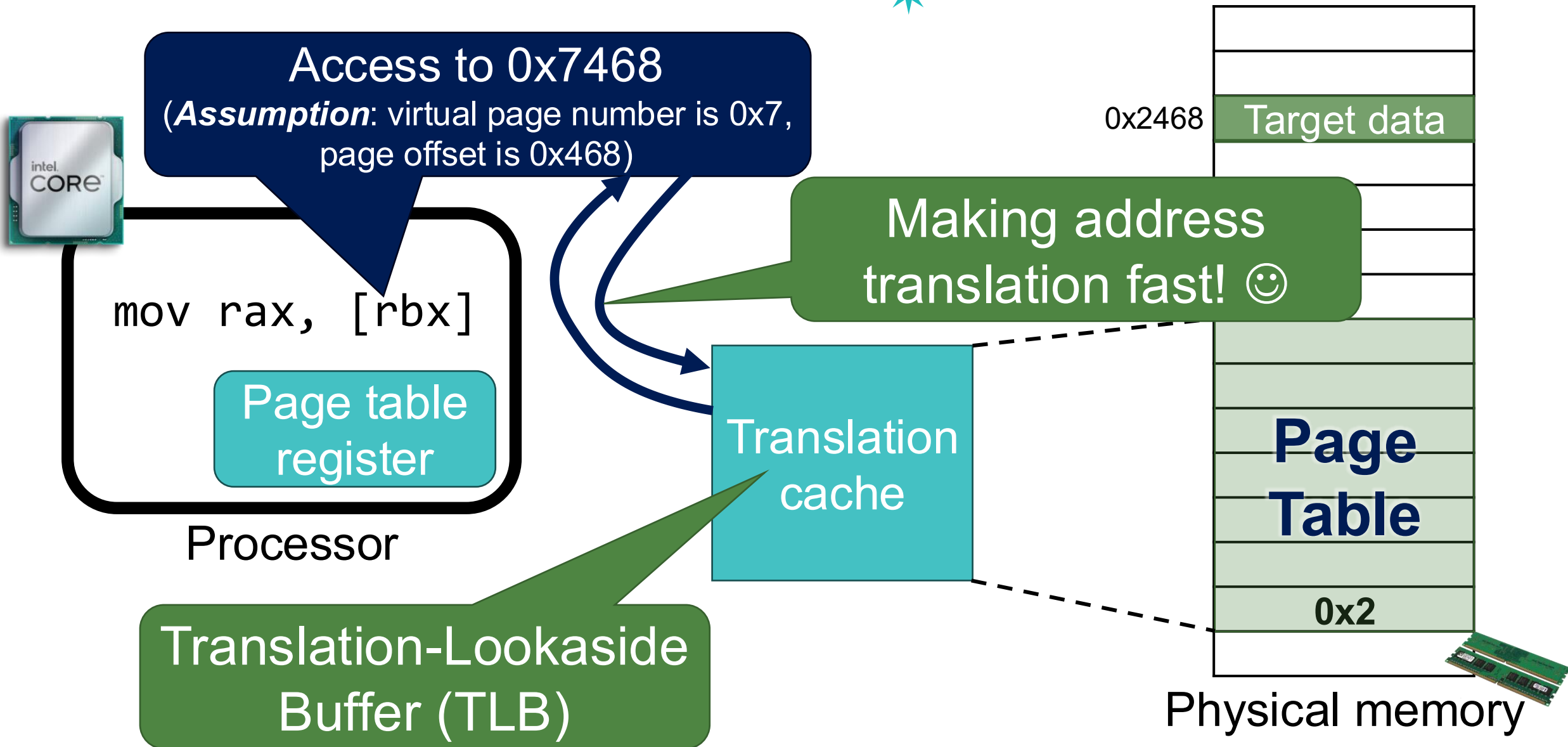
Page Table

0x2

Physical memory



Solution: A Cache for Address Translation ⁹



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)

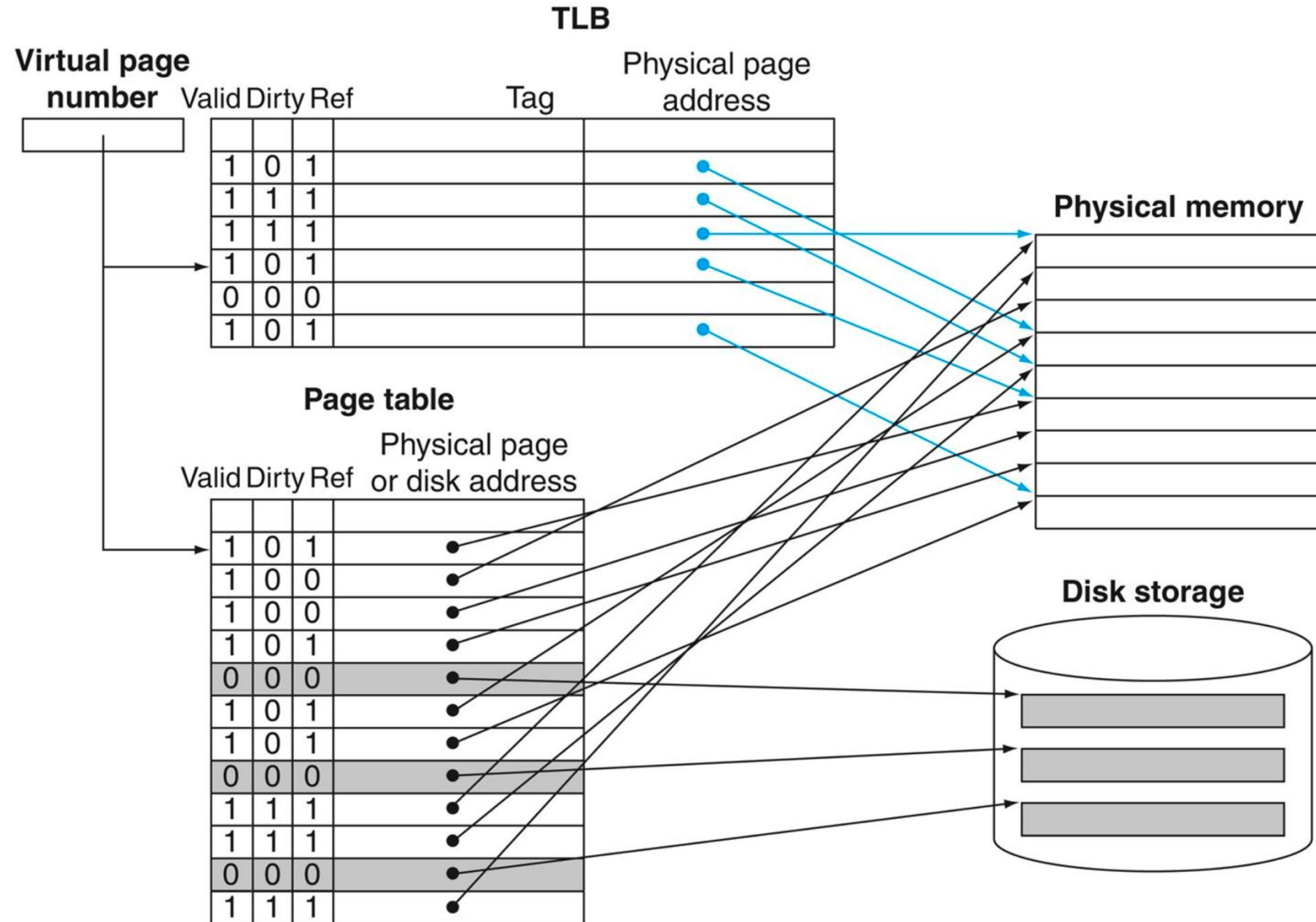


A **cache** that keeps track of recently used address mappings to try to avoid an access to the page table

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

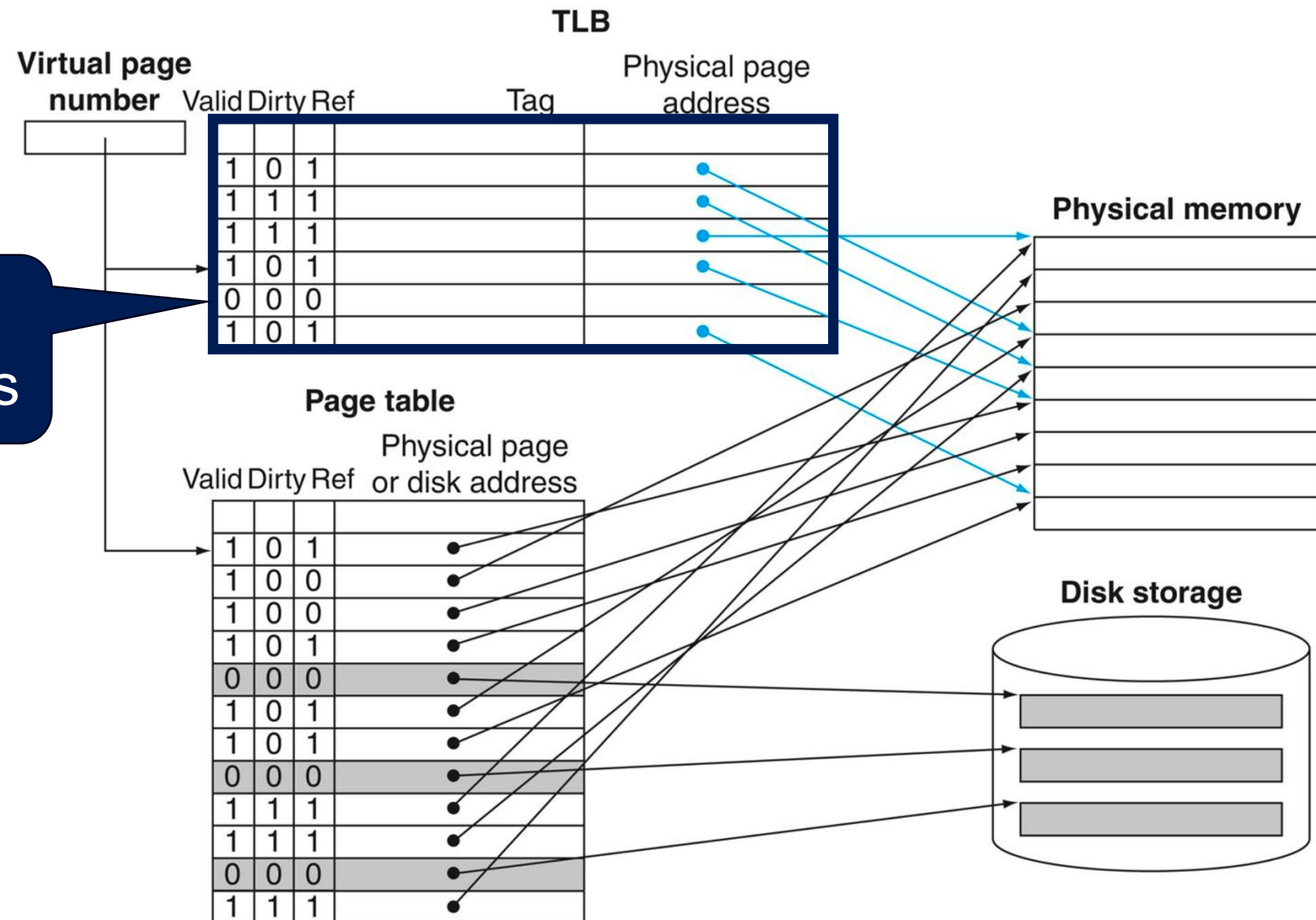
“When a translation for a virtual page number is used, it will probably be needed again in the near future”

Translation-Lookaside Buffer (TLB)



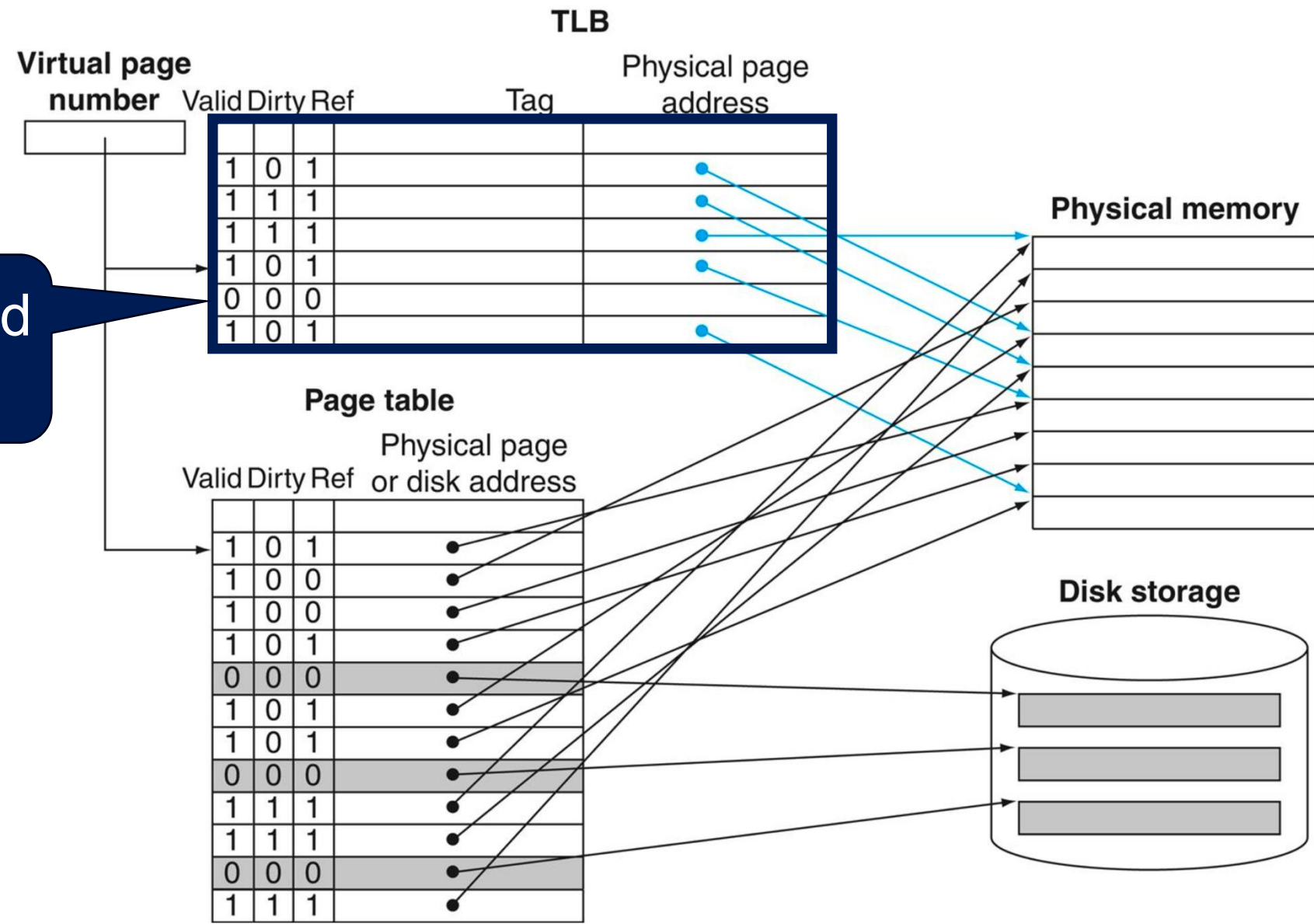
Translation-Lookaside Buffer (TLB)

TLB: a special cache for address translations



Translation-Lookaside Buffer (TLB)

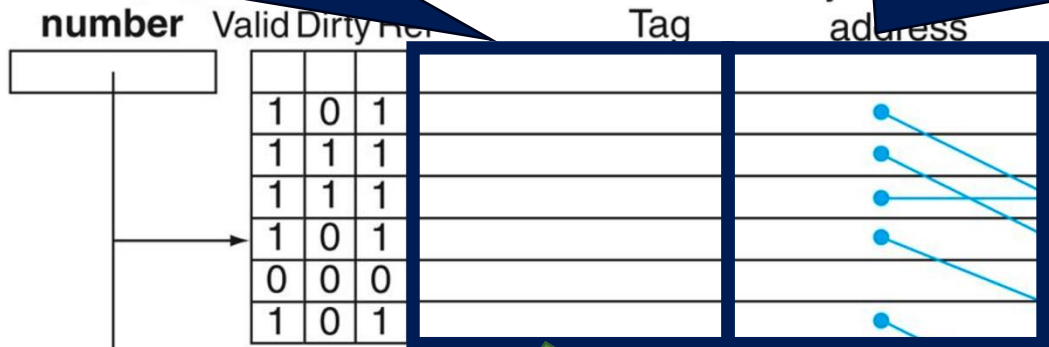
Stores the recently used address mappings



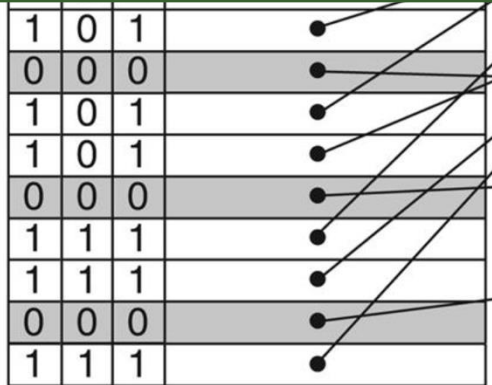
Translation-Lookaside Buffer (TLB)

Virtual page number

Physical page number

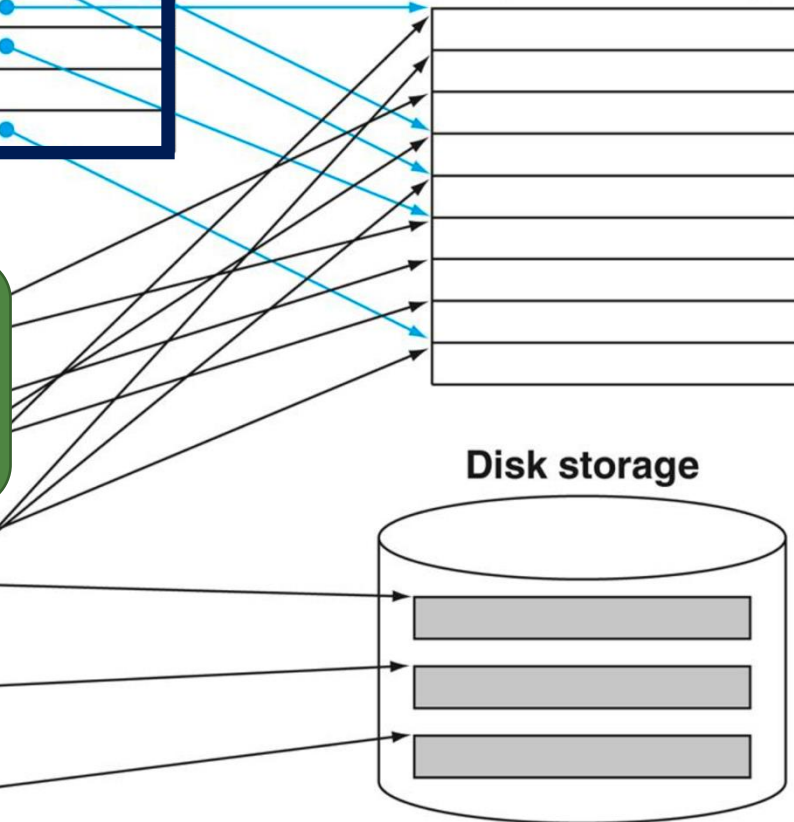


Tag = virtual page #
(→ fully-associative)



Physical memory

Disk storage



Translation-Lookaside Buffer (TLB)

Virtual page number

Physical page number

Same with the cache "valid" bit

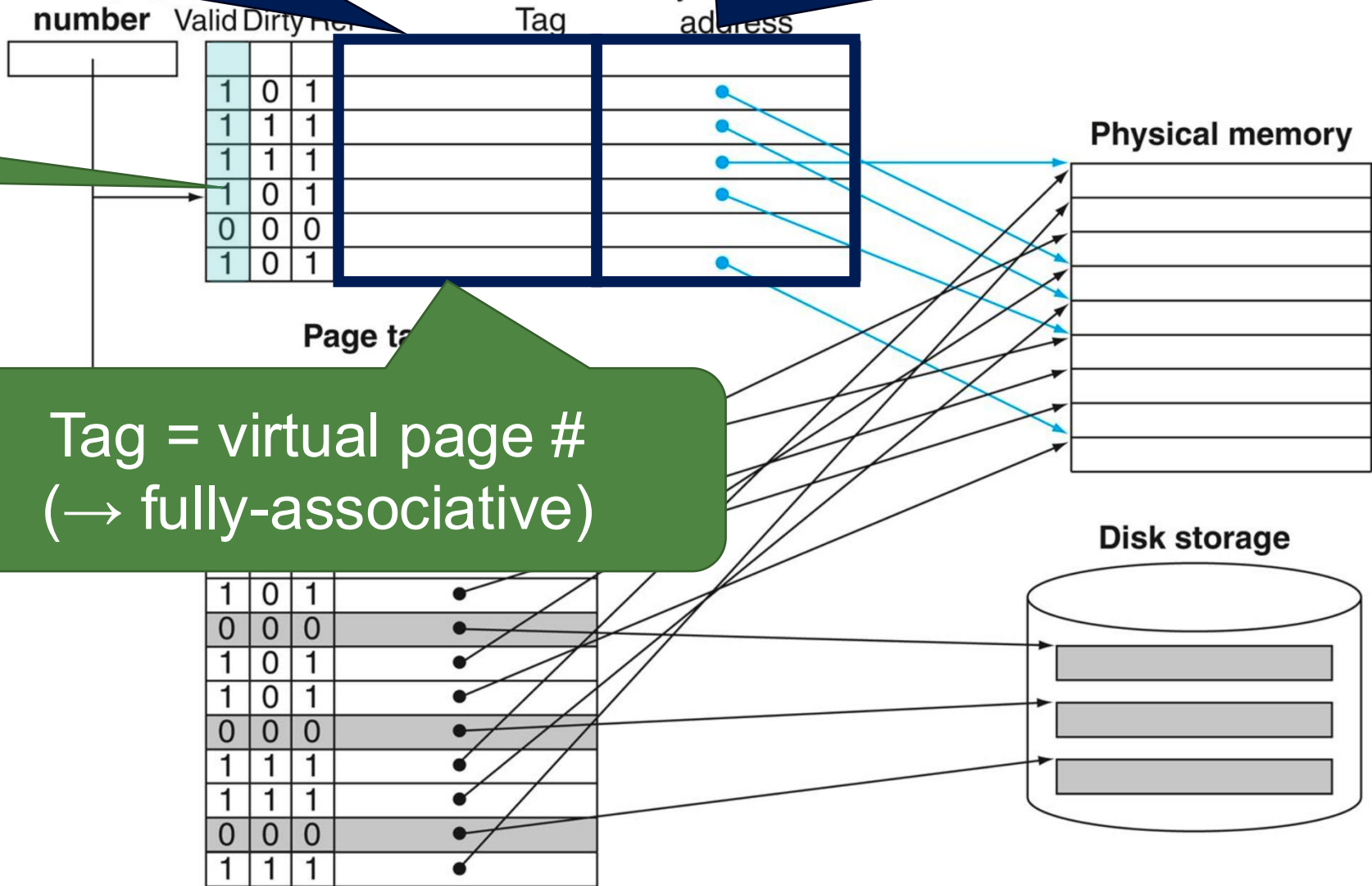
Tag = virtual page #
(→ fully-associative)

number	Valid	Dirty	Physical page number	Tag	address
	1	0			
	1	1			
	1	1			
	1	0			
	0	0			
	1	0			

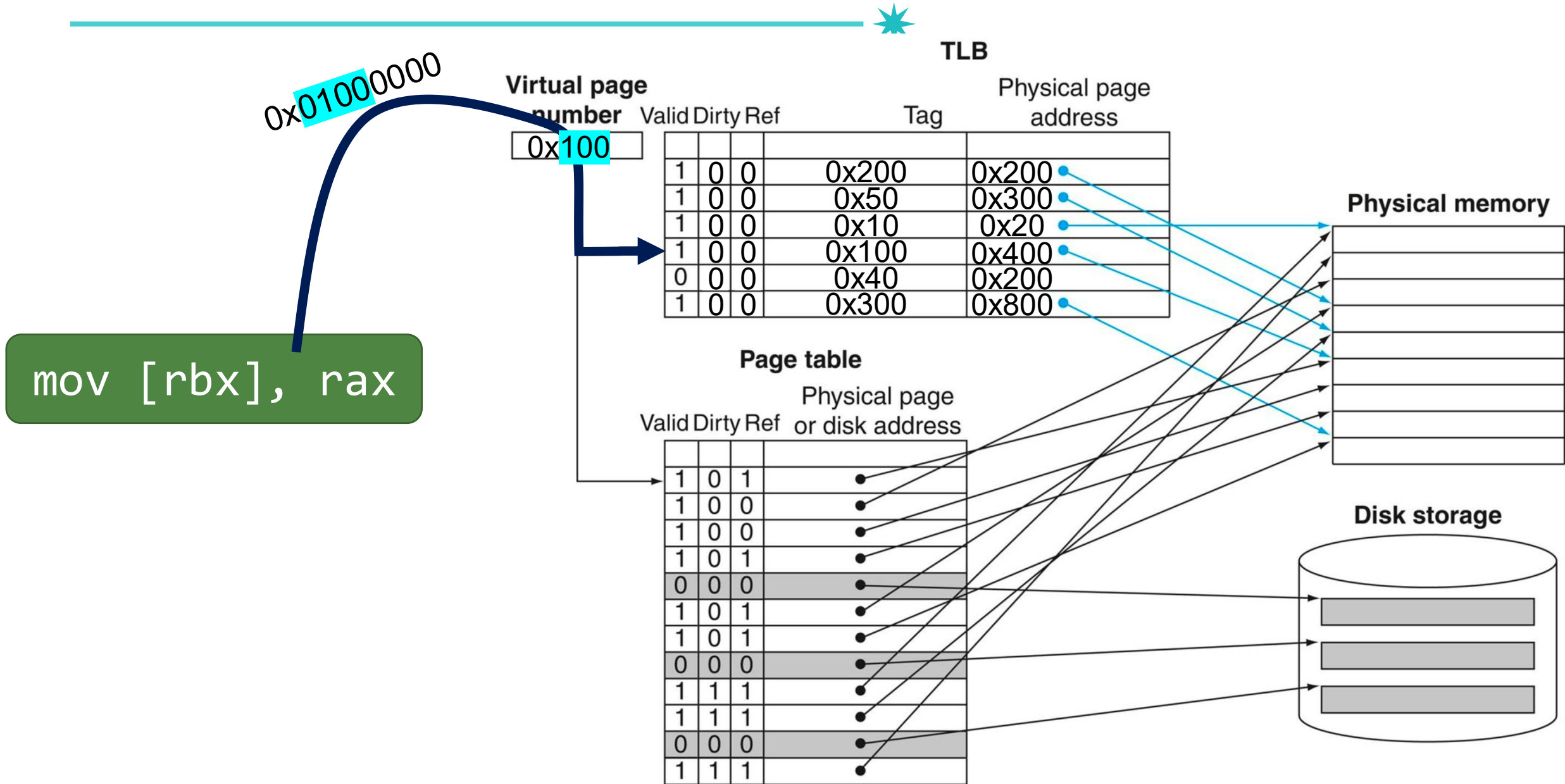
1	0	1		
0	0	0		
1	0	1		
1	0	1		
0	0	0		
1	1	1		
1	1	1		
0	0	0		
1	1	1		

Physical memory

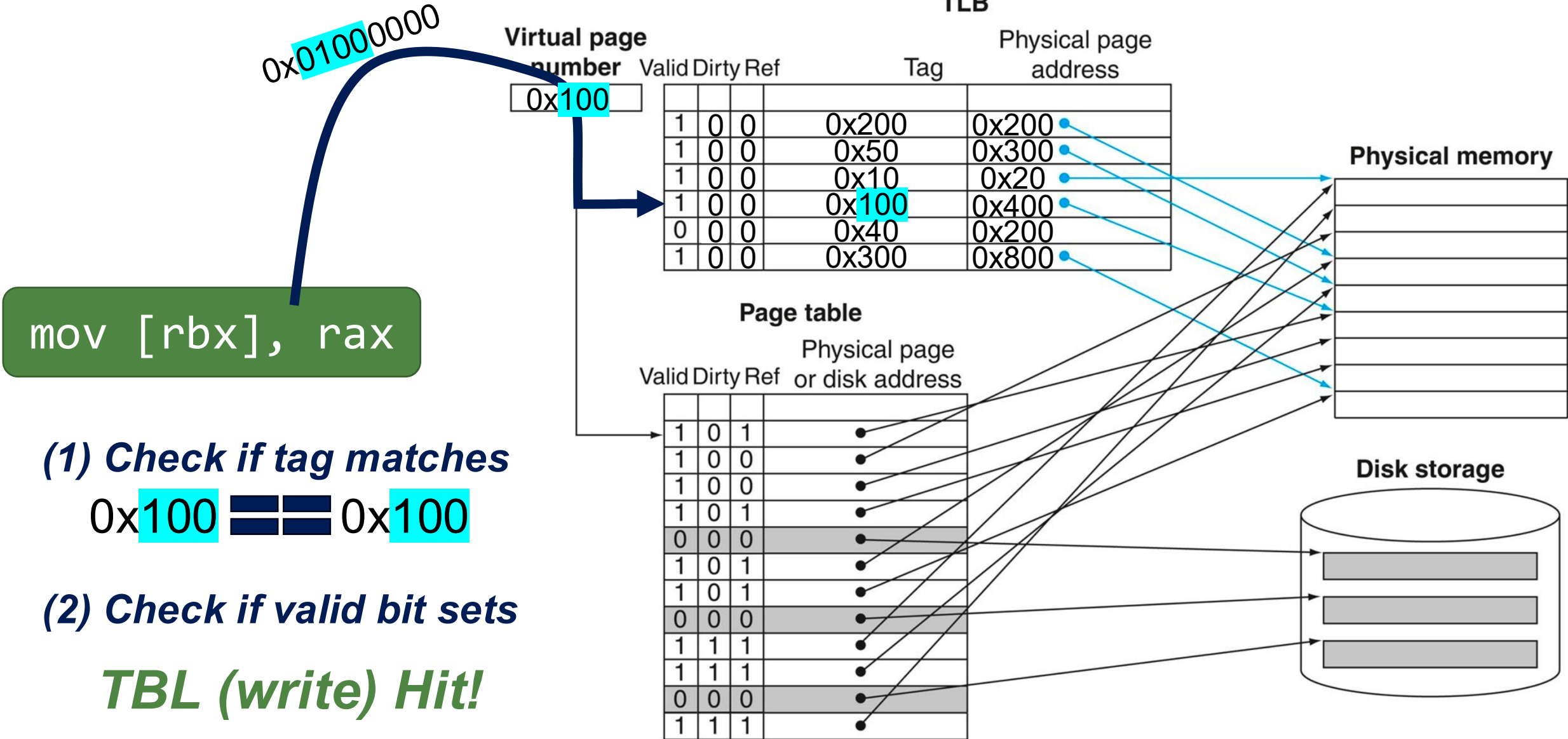
Disk storage



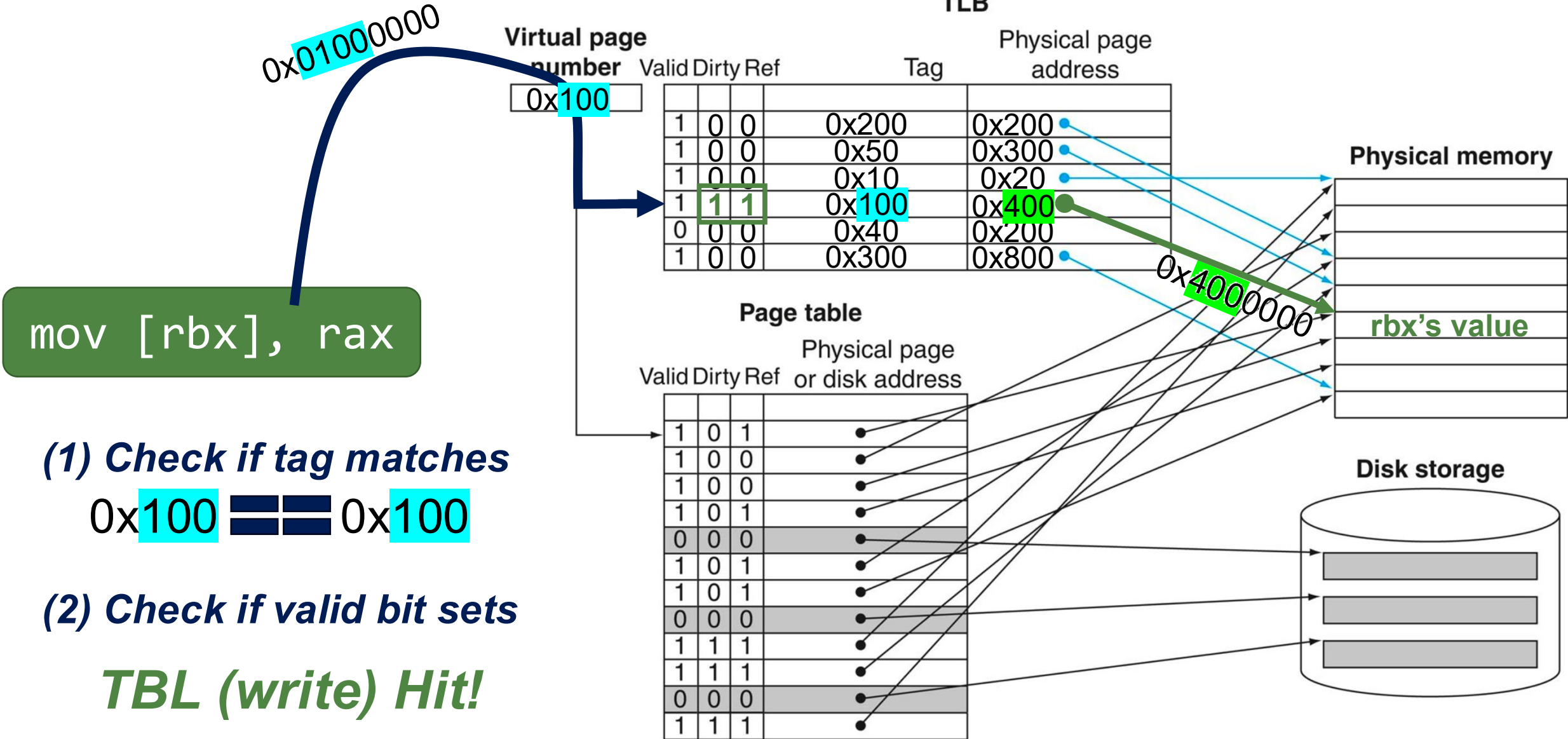
Case #1: TLB Hits



Case #1: TLB Hits



Case #1: TLB Hits



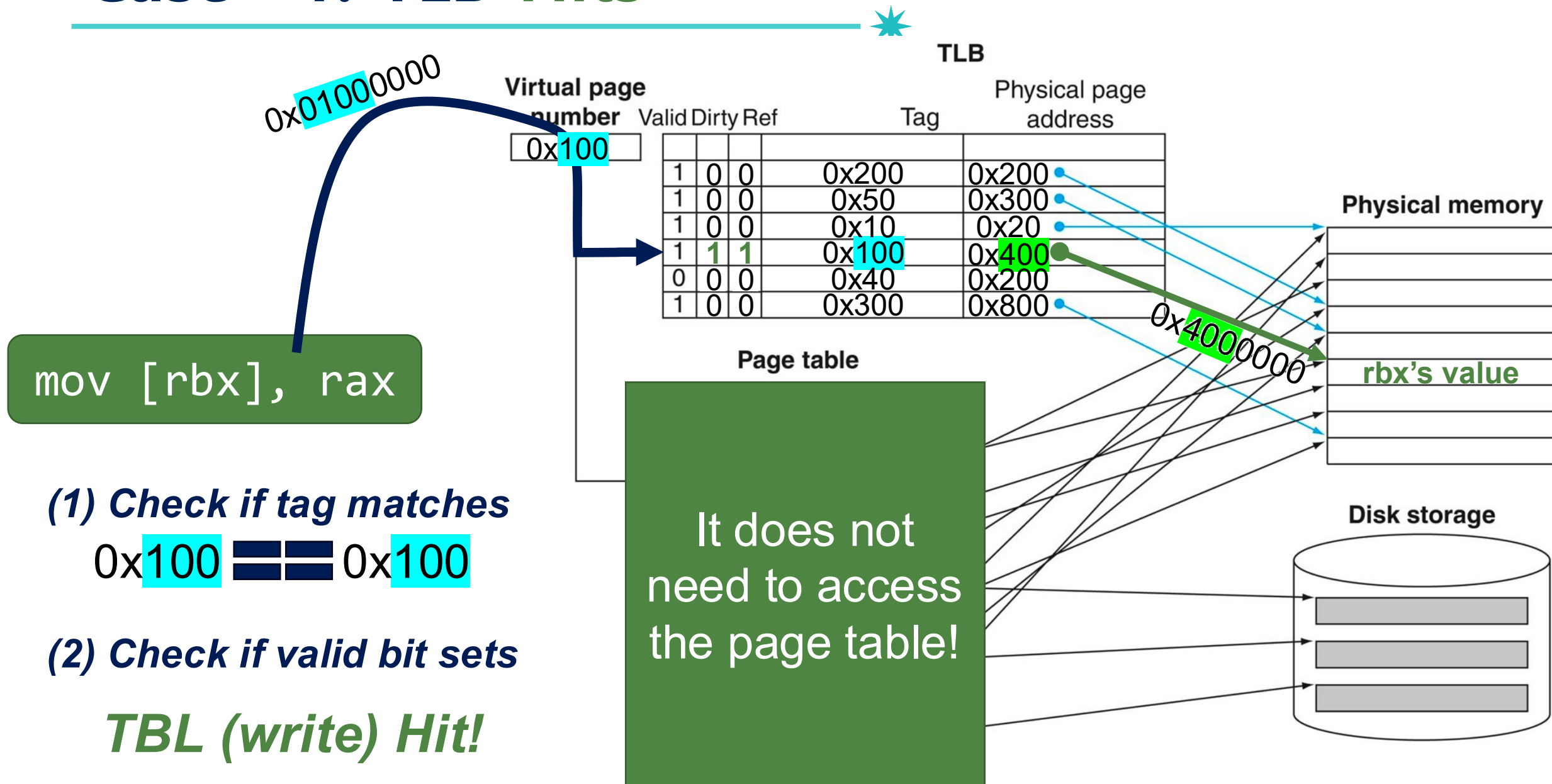
(1) Check if tag matches

`0x100` == `0x100`

(2) Check if valid bit sets

TLB (write) Hit!

Case #1: TLB Hits



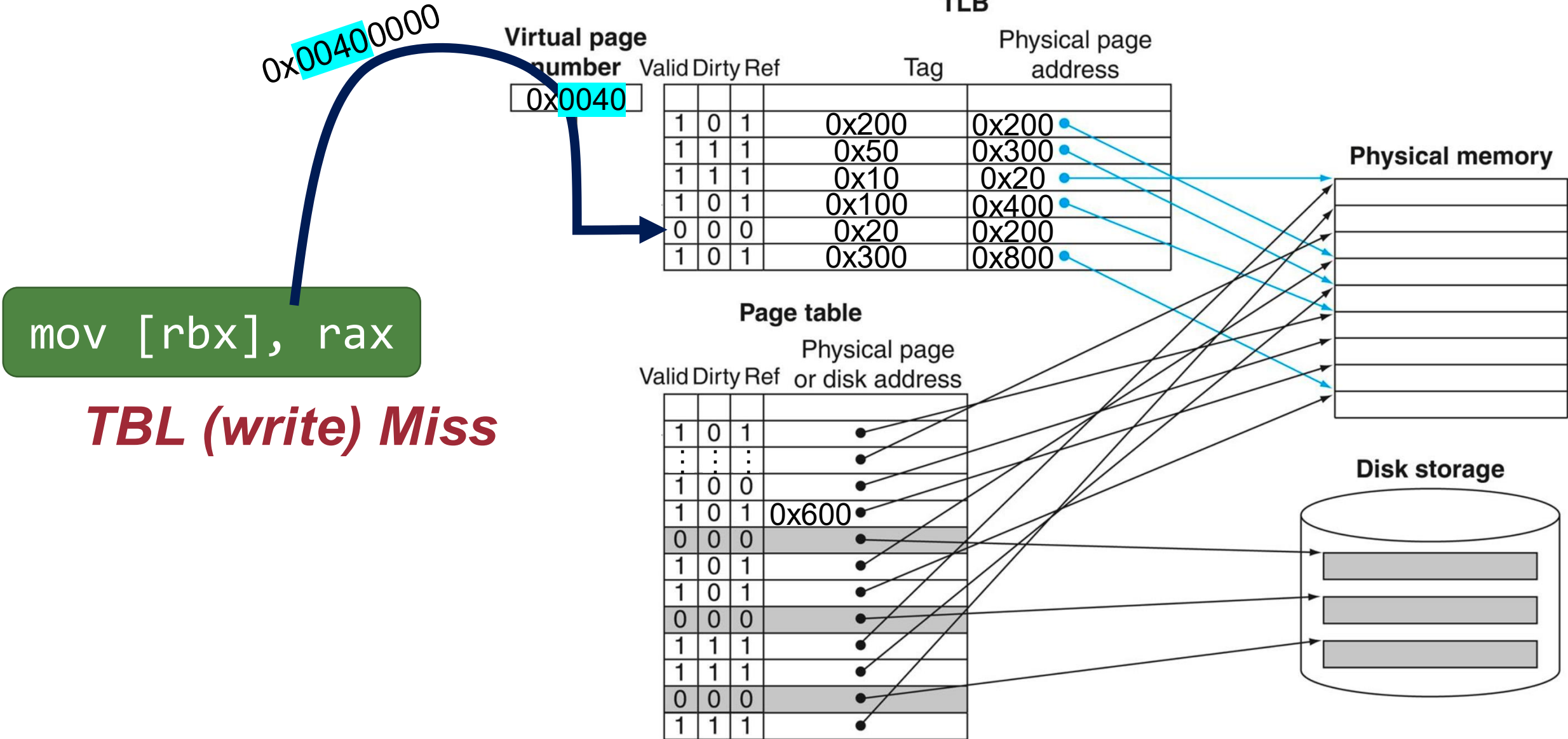
(1) Check if tag matches

`0x100` == `0x100`

(2) Check if valid bit sets

TLB (write) Hit!

Case #2: TLB Misses



Case #2: TLB Misses

0x00400000

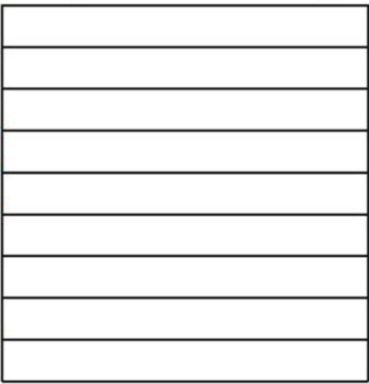
Virtual page number

0x0040

TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
0	0	0	0x20	0x200
1	0	1	0x300	0x800

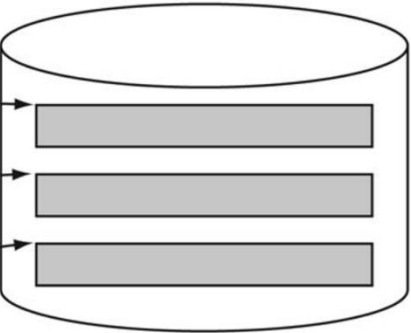
Physical memory



Page table

Valid	Dirty	Ref	Physical page or disk address
1	0	1	•
⋮	⋮	⋮	•
1	0	0	•
1	0	1	0x600
0	0	0	•
1	0	1	•
1	0	1	•
0	0	0	•
1	1	1	•
1	1	1	•
0	0	0	•
1	1	1	•

Disk storage



```
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 #2-1: TLB Misses – Memory Hit

0x00400000

Virtual page number

0x0040

```
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)

TLB

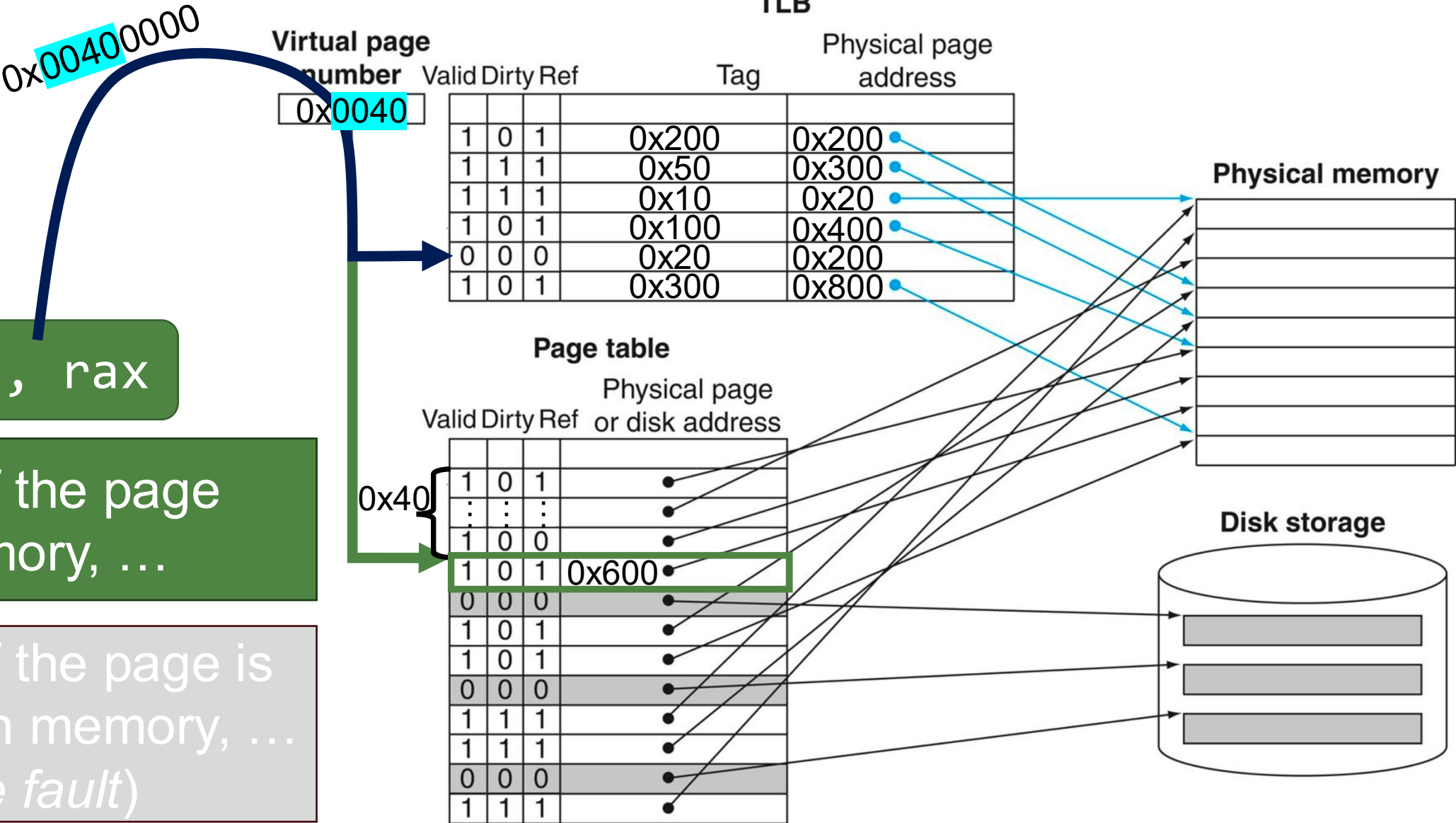
Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
0	0	0	0x20	0x200
1	0	1	0x300	0x800

Page table

Valid	Dirty	Ref	Physical page or disk address
1	0	1	•
•	•	•	•
1	0	0	•
1	0	1	0x600
0	0	0	•
1	0	1	•
1	0	1	•
0	0	0	•
1	1	1	•
1	1	1	•
0	0	0	•
1	1	1	•

Physical memory

Disk storage



Case #2-1: TLB Misses – Memory Hit

0x00400000

Virtual page number

0x0040

```
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)

TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
1	0	1	0x40	0x600
1	0	1	0x300	0x800

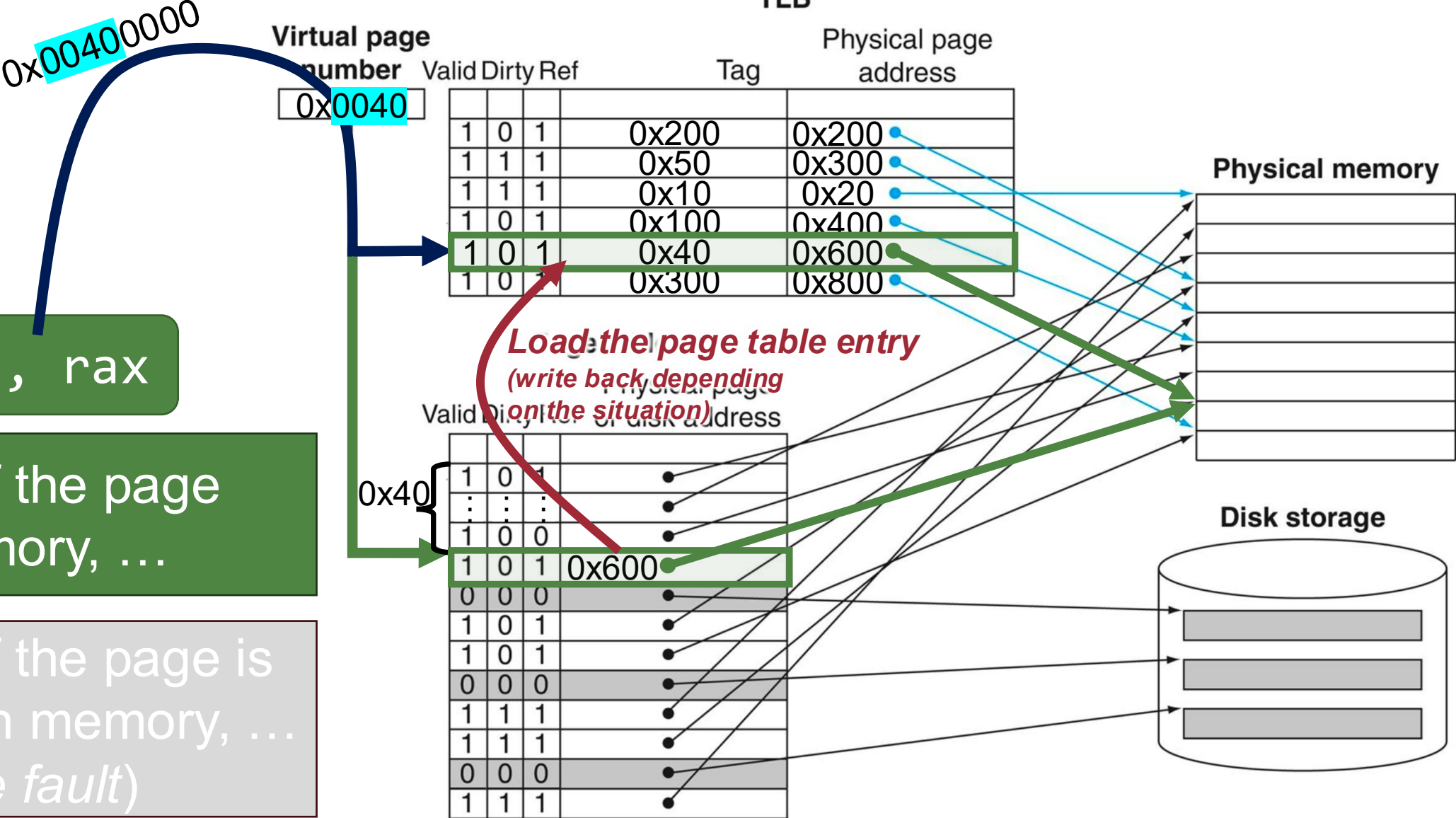
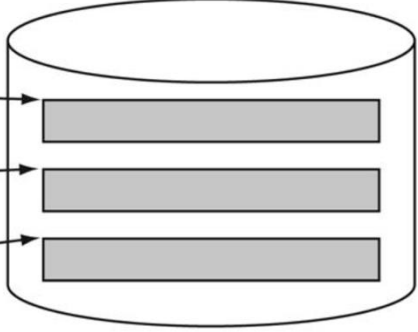
Physical memory

Load the page table entry (write back depending on the situation)

Valid Dirty Ref of disk address

1	0	1	•
•	•	•	•
•	•	•	•
1	0	1	0x600
0	0	0	•
1	0	1	•
1	0	1	•
0	0	0	•
1	1	1	•
1	1	1	•
0	0	0	•
1	1	1	•

Disk storage



Case #2-1: TLB Misses – Memory Hit

0x00400000

Virtual page number

0x0040

```
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)

TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
1	0	1	0x40	0x600
1	0	1	0x300	0x800

Physical memory

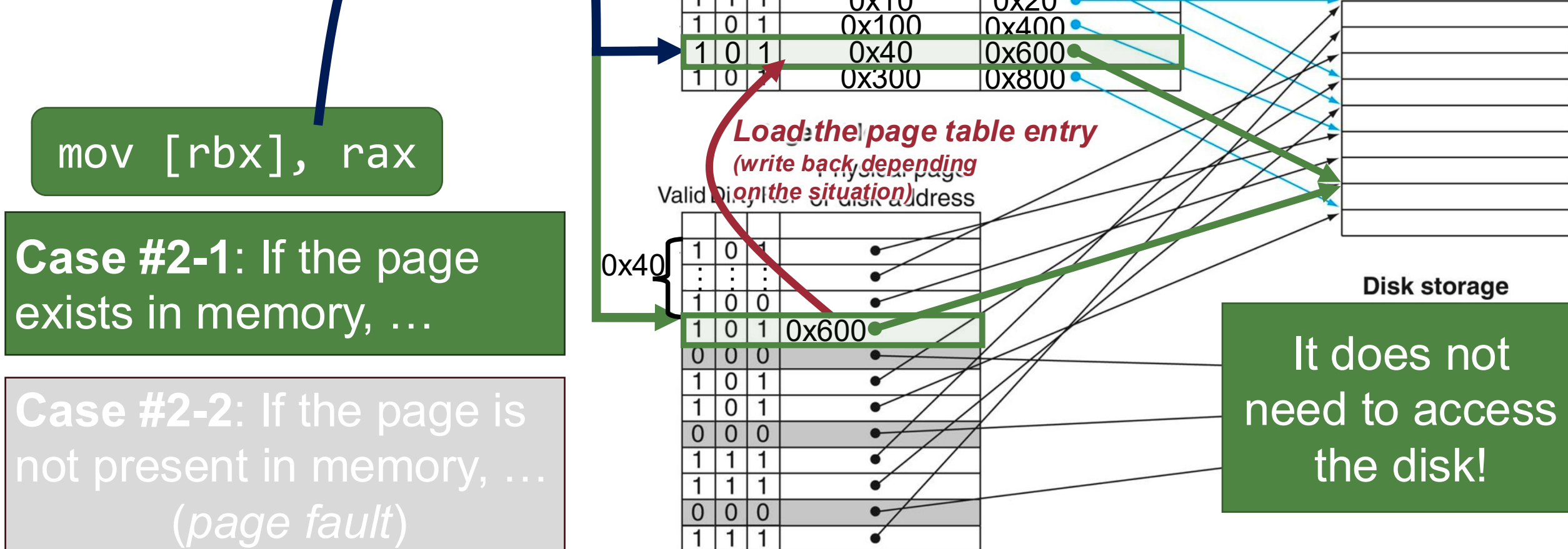
Load the page table entry (write back depending on the situation)

Valid Dirty Ref of disk address

Valid	Dirty	Ref	of disk address
1	0	1	•
•	•	•	•
•	•	•	•
1	0	1	0x600
0	0	0	•
1	0	1	•
1	0	1	•
0	0	0	•
1	1	1	•
1	1	1	•
0	0	0	•
1	1	1	•

Disk storage

It does not need to access the disk!



Case #2-2: TLB Misses – Page Fault

0x00700000

Virtual page number

0x0070

TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
0	0	0	0x20	0x600
1	0	1	0x300	0x800

Physical memory

Disk storage

Page table

Valid Dirty Ref Physical page or disk address

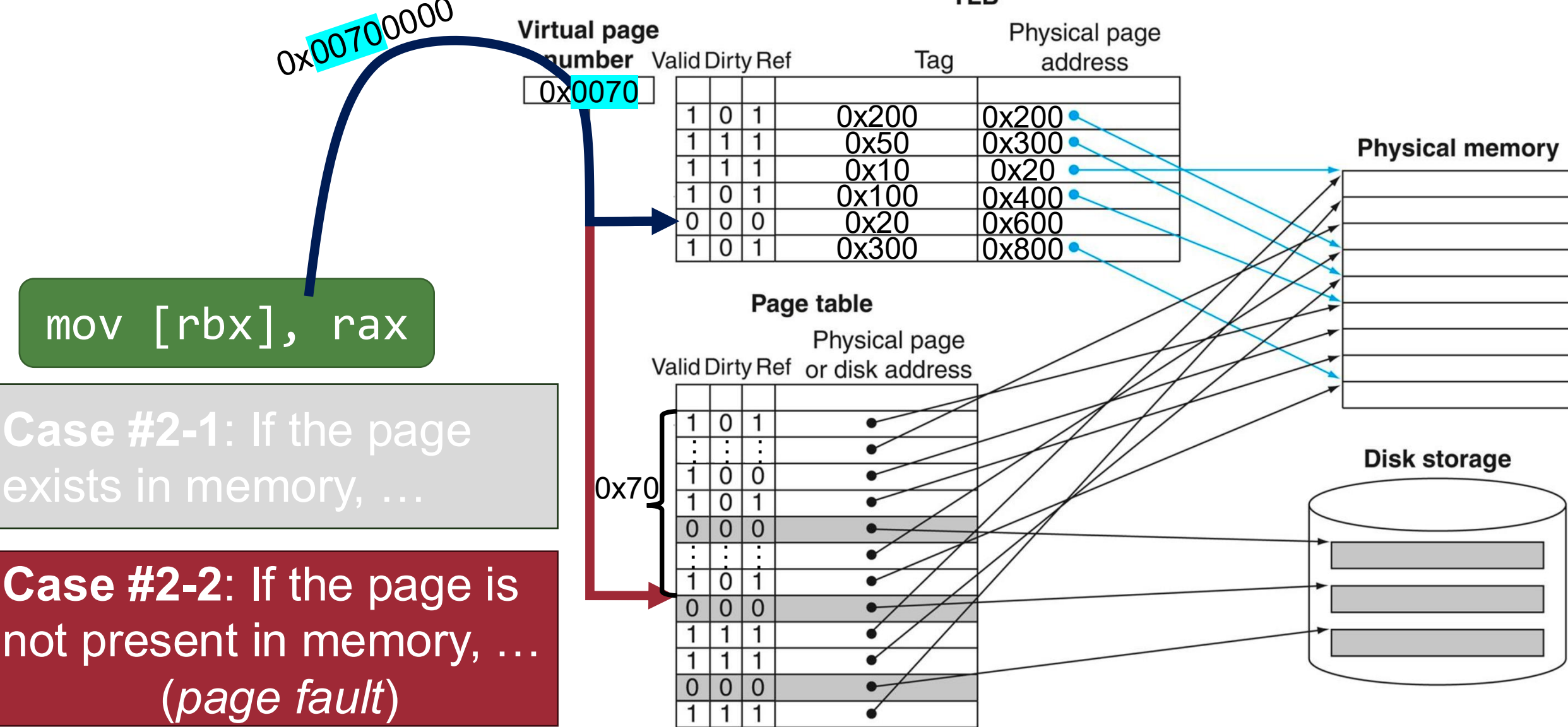
1	0	1	•
⋮	⋮	⋮	•
1	0	0	•
1	0	1	•
0	0	0	•
⋮	⋮	⋮	•
1	0	1	•
0	0	0	•
1	1	1	•
1	1	1	•
0	0	0	•
1	1	1	•

0x70

```
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

0x00700000

Virtual page number

0x0070

Valid Dirty Ref

TLB

Tag

Physical page address

Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400

Physical memory

```
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)

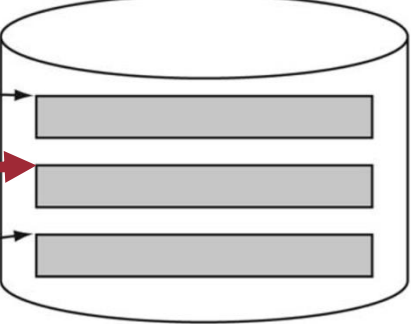
Miss (valid bit 0): access on memory space not in physical memory (but in disk)

Valid Dirty Ref

0x70

Valid	Dirty	Ref
1	0	1
...
1	0	1
1	0	1
0	0	0
...
1	0	1
0	0	0
1	1	1
1	1	1
0	0	0
1	1	1

Disk storage



Case #2-2: TLB Misses – Page Fault

0x00700000

Virtual page number

0x0070

Valid Dirty Ref			Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
0	0	0	0x20	0x600
1	0	1	0x300	0x800

Page table

Valid Dirty Ref Physical page or disk address

1	0	1	•
⋮	⋮	⋮	•
1	0	0	•
1	0	1	•
0	0	0	•
⋮	⋮	⋮	•
1	0	1	•
0	0	0	•
1	1	1	•
1	1	1	•
0	0	0	•
1	1	1	•

0x70

Physical memory

Disk storage

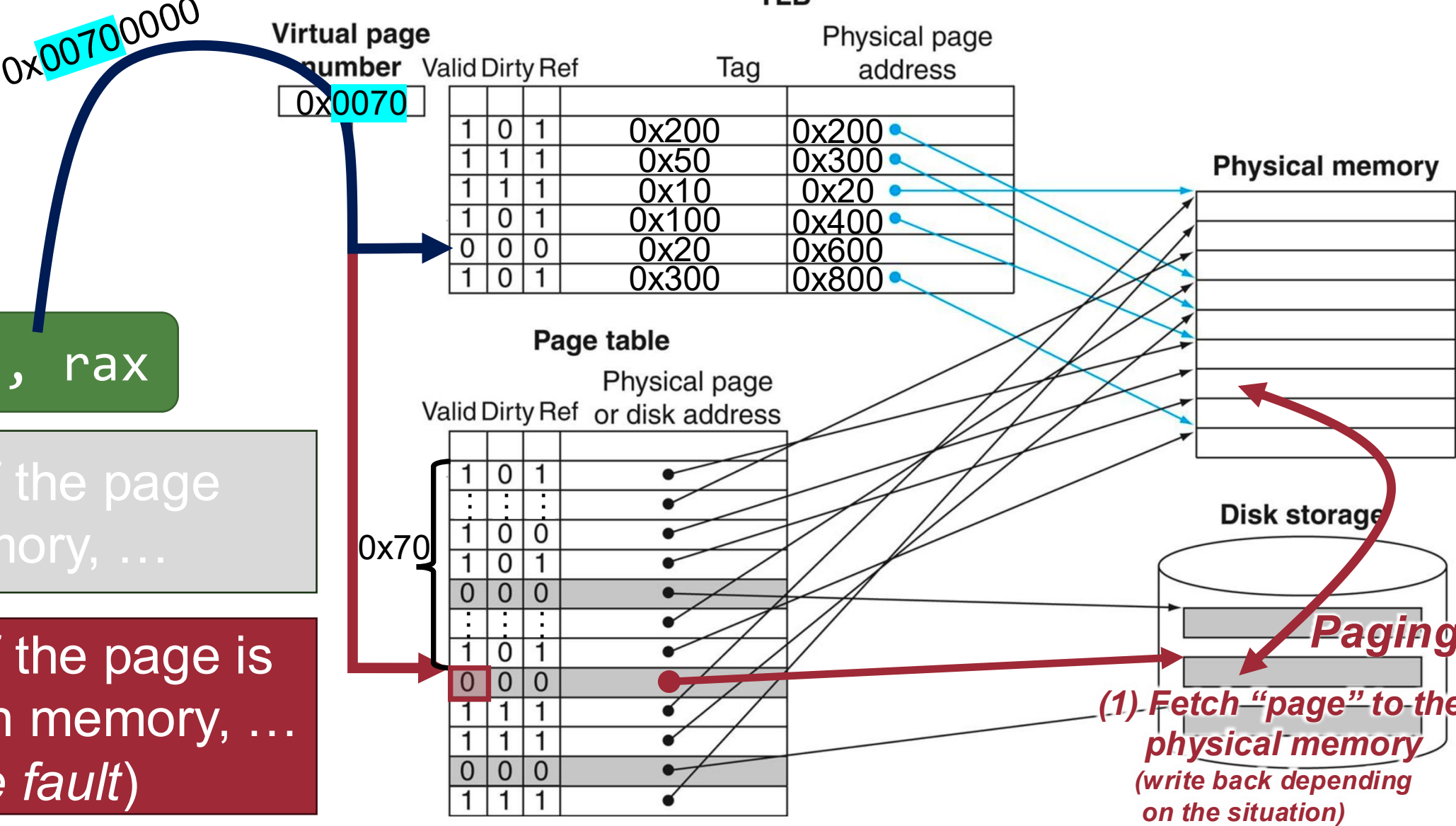
Paging

(1) Fetch "page" to the physical memory
(write back depending on the situation)

```
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

0x00700000

Virtual page number

0x0070

```
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)

TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
0	0	0	0x20	0x600
1	0	1	0x300	0x800

Page table

Valid	Dirty	Ref	Physical page or disk address
1	0	1	•
•	•	•	•
0	0	0	•
1	0	1	•
0	0	0	•
•	•	•	•
1	0	1	•
1	0	0	•
1	1	1	•
1	1	1	•
0	0	0	•
1	1	1	•

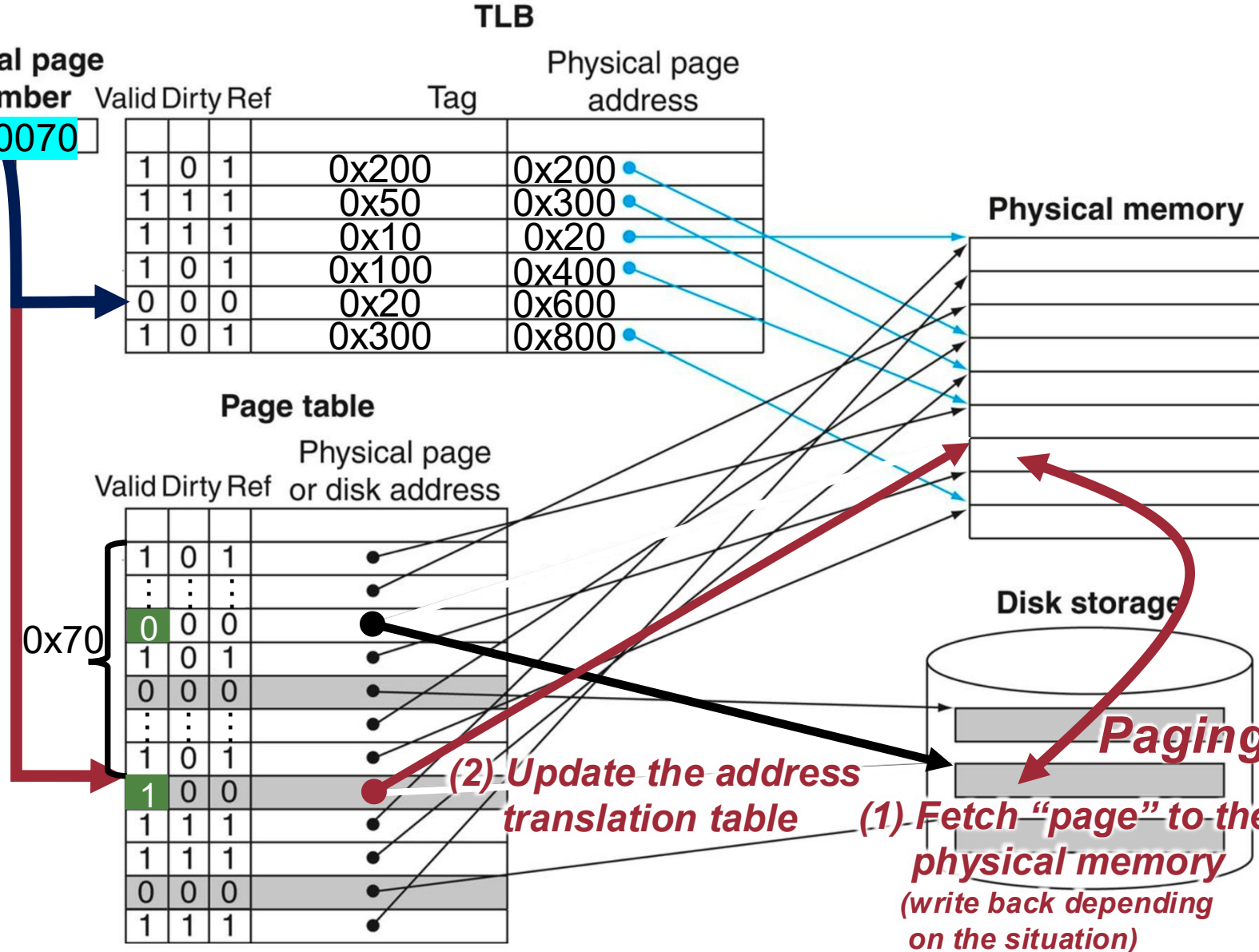
Physical memory

Disk storage

Paging

(2) Update the address translation table

(1) Fetch "page" to the physical memory (write back depending on the situation)



Case #2-2: TLB Misses – Page Fault

0x00700000

Virtual page number

0x0070

```
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)

TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1	0x200	0x200
1	1	1	0x50	0x300
1	1	1	0x10	0x20
1	0	1	0x100	0x400
1	0	0	0x70	0x500
1	0	1	0x300	0x800

Physical memory

Valid Dirty Ref address

Valid	Dirty	Ref	address
1	0	1	
...	
0	0	0	0x70
1	0	1	
0	0	0	
...	
1	0	1	
1	0	0	
1	1	1	
1	1	1	
0	0	0	
1	1	1	

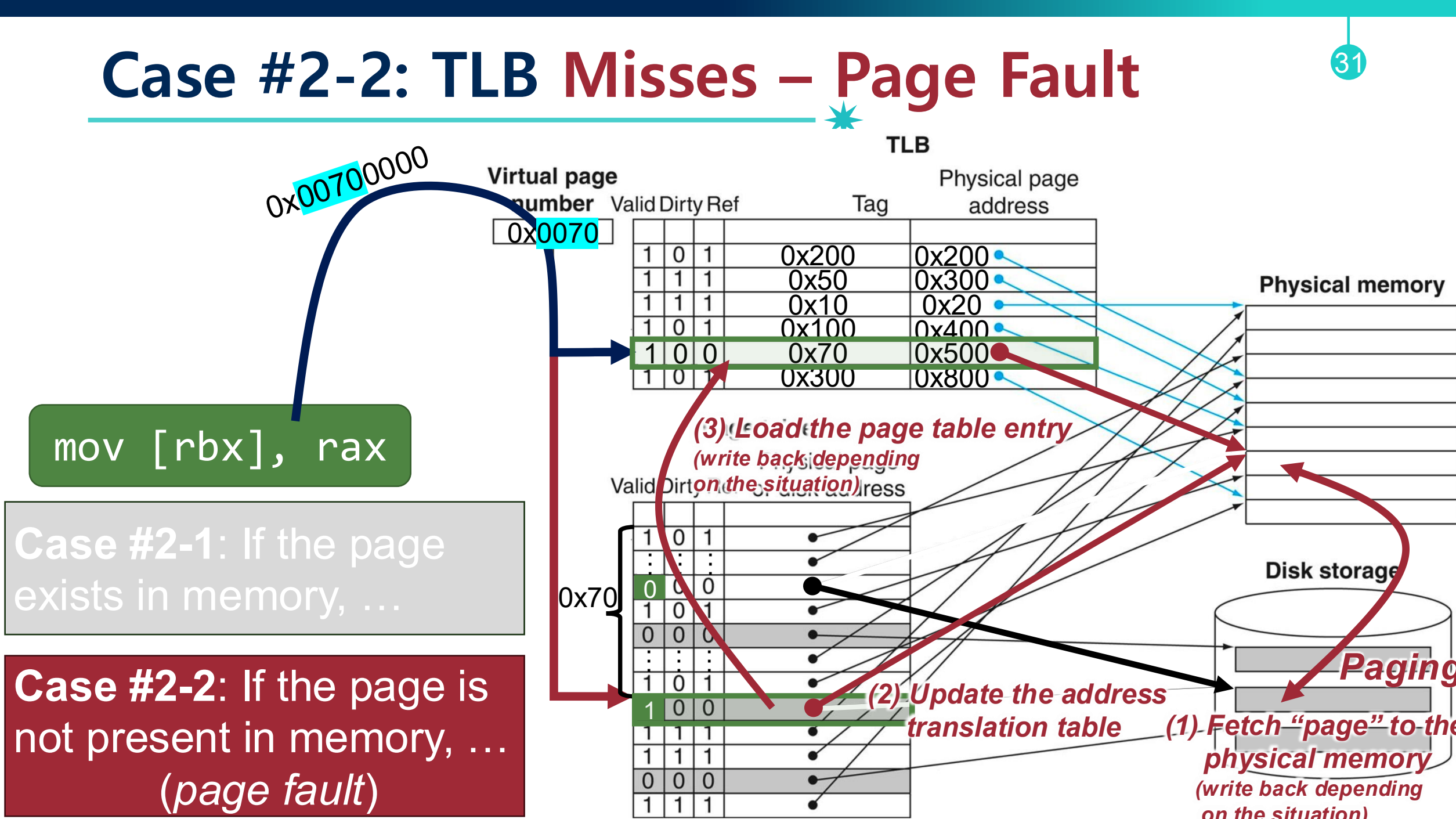
Disk storage

(3) Load the page table entry (write back depending on the situation)

(2) Update the address translation table

(1) Fetch "page" to the physical memory (write back depending on the situation)

Paging



Case #2: TLB Misses



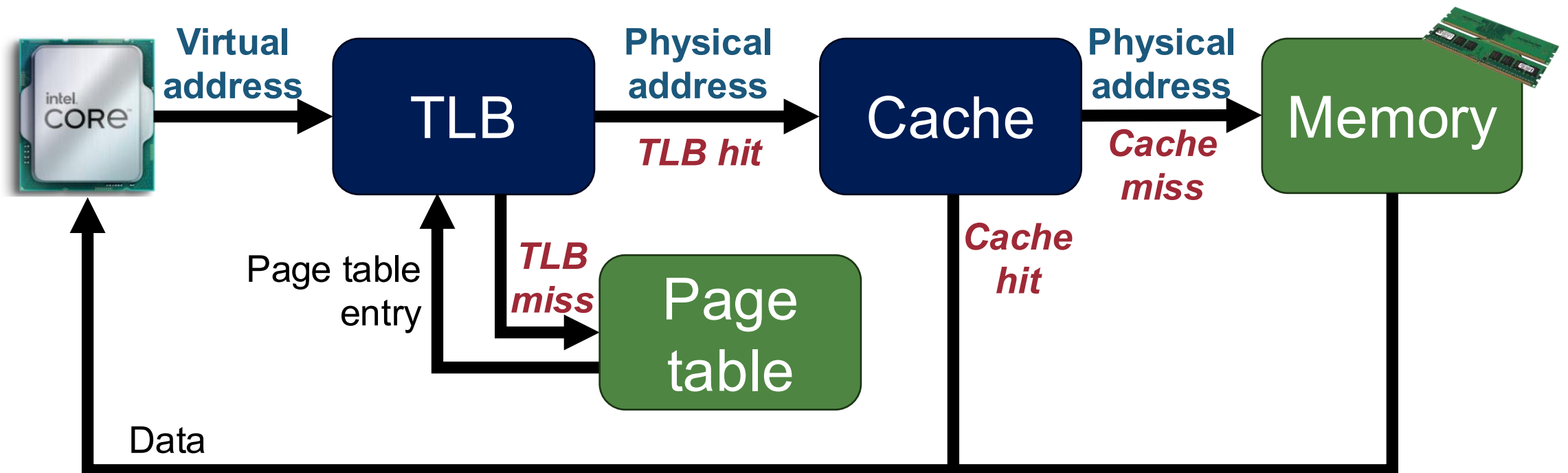
Case #2-1: If page is **in memory**

- 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)

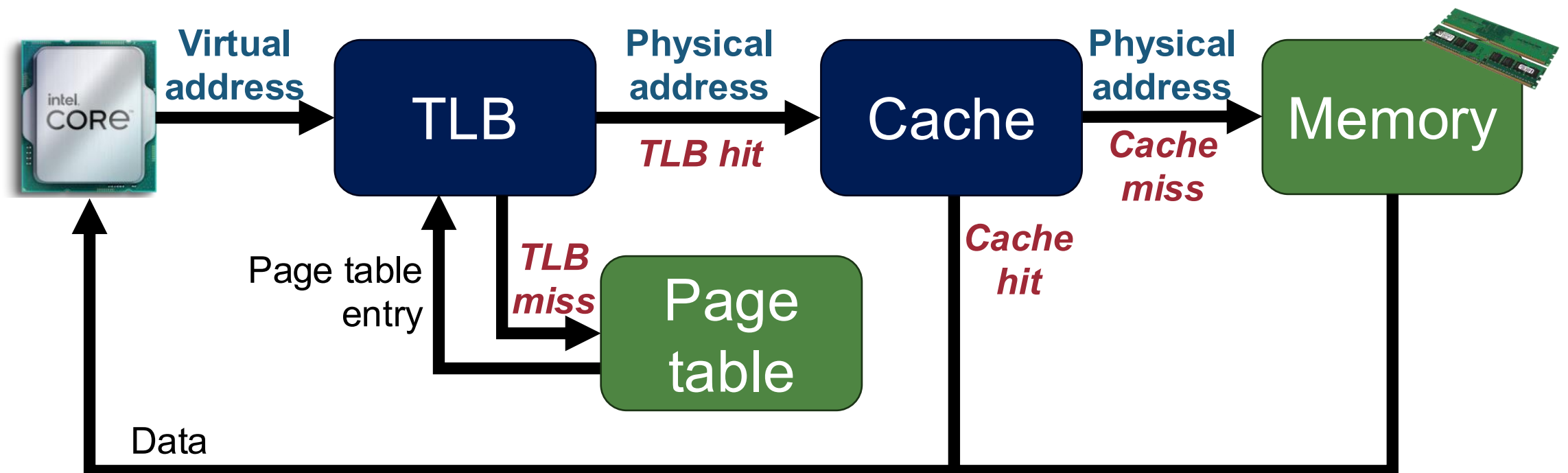
- OS handles fetching the page and updating the page table
- Then restart the faulting instruction

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

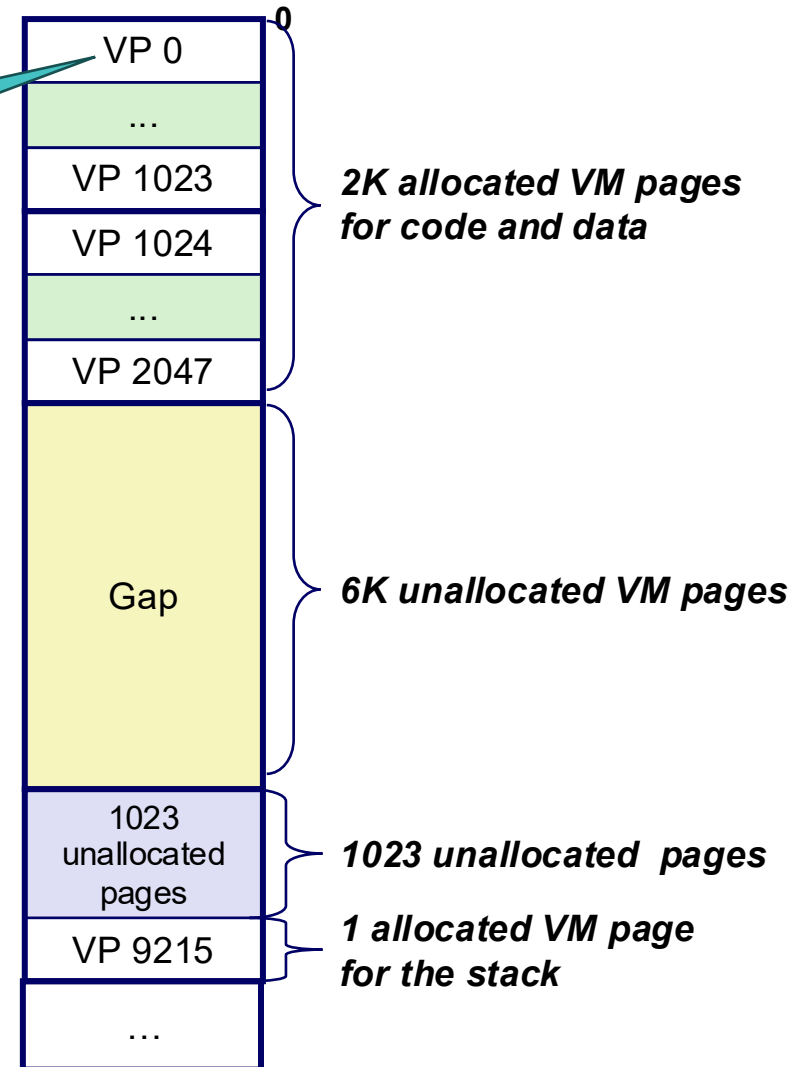


Multi-level Page Tables

We Have a Problem!

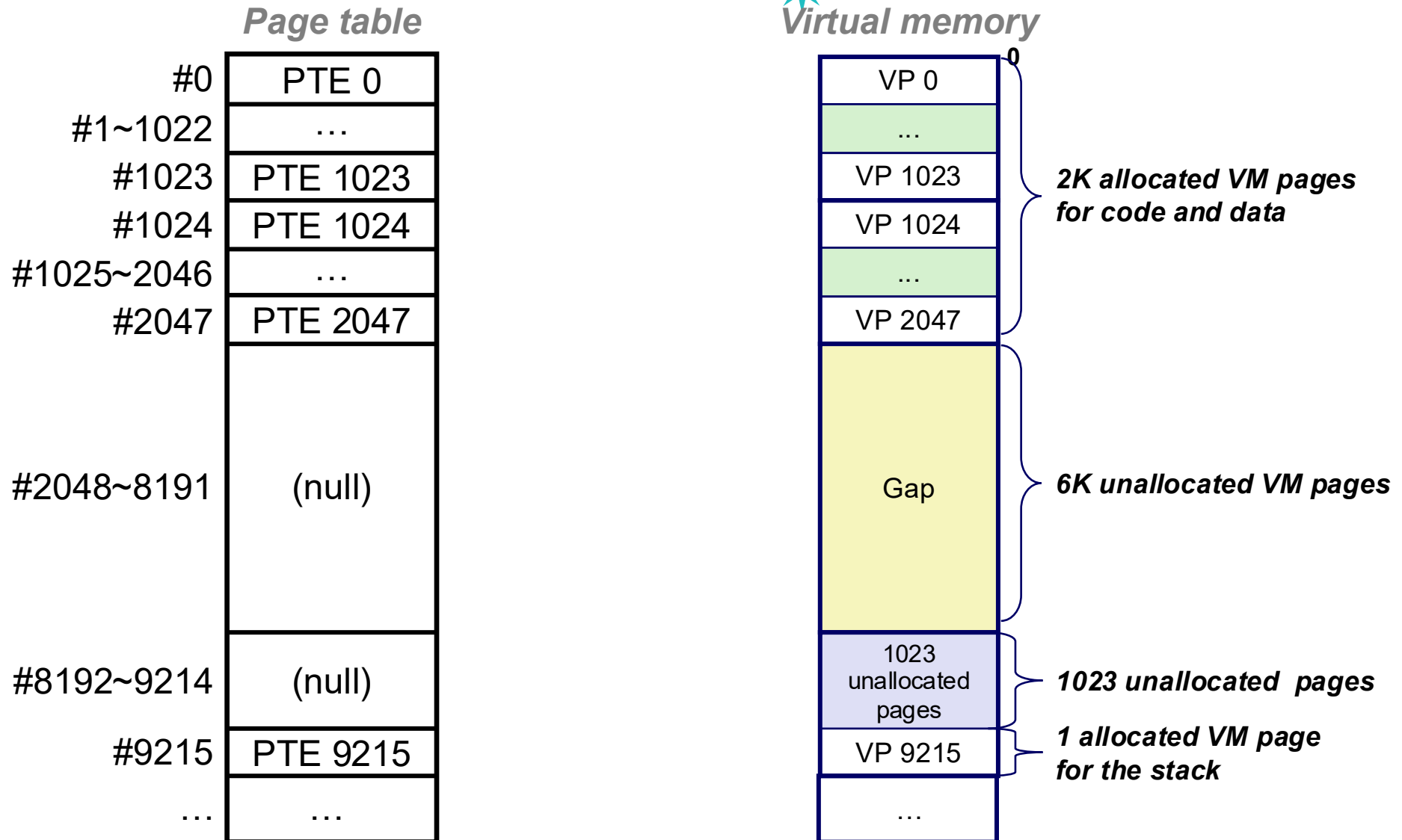
Virtual page (VP) number

Virtual memory

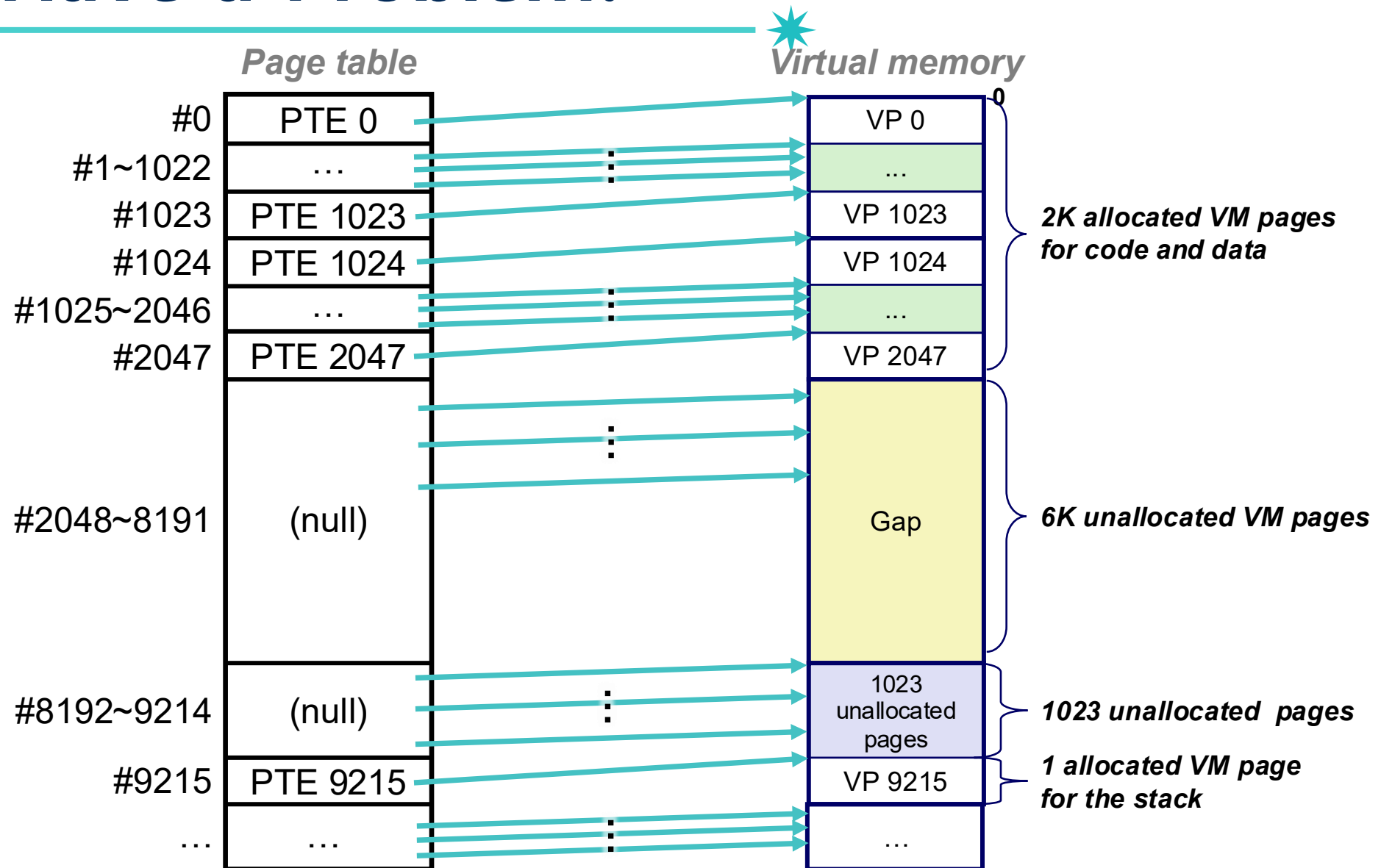


What is the problem?

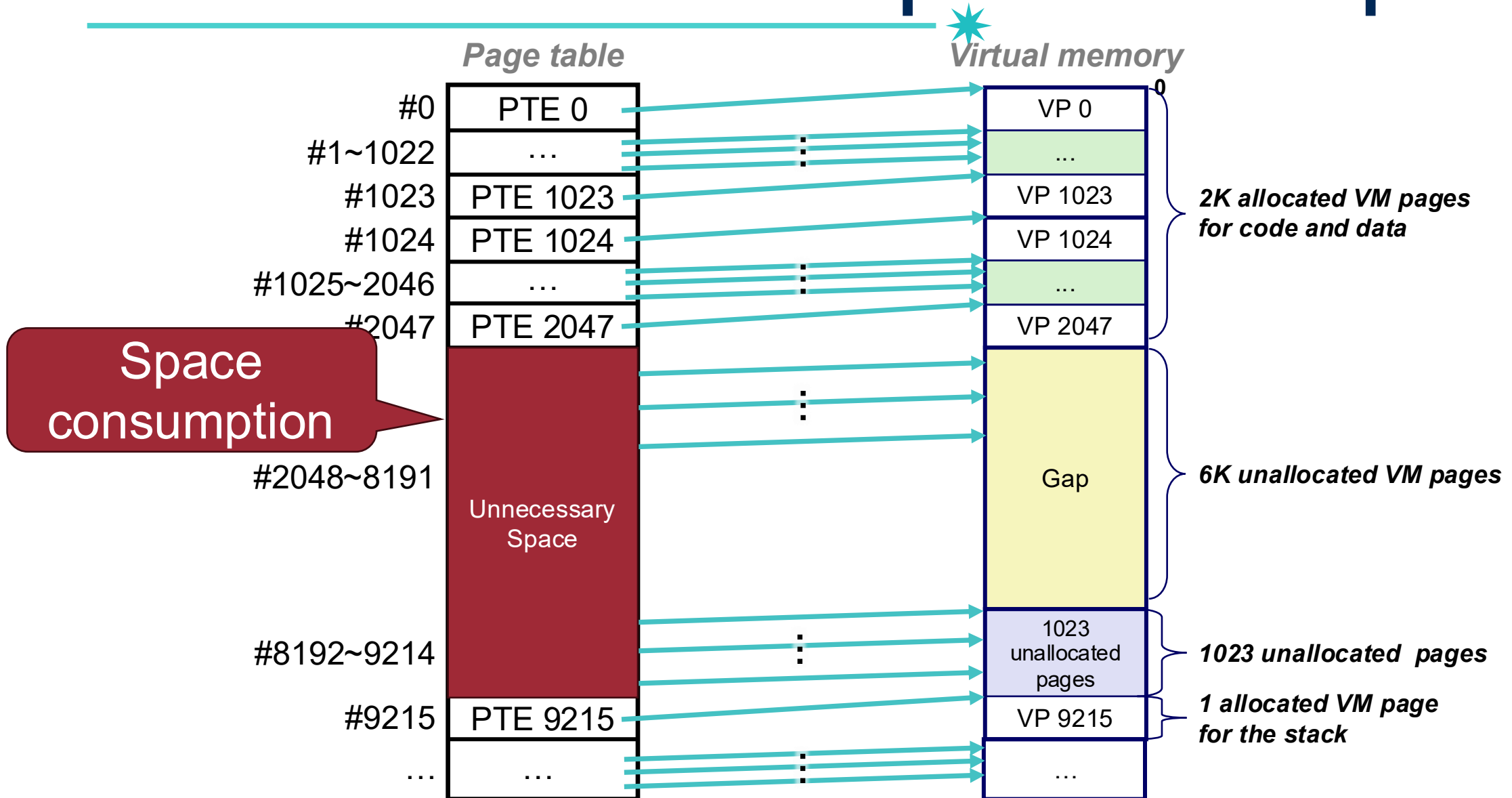
We Have a Problem!



We Have a Problem!

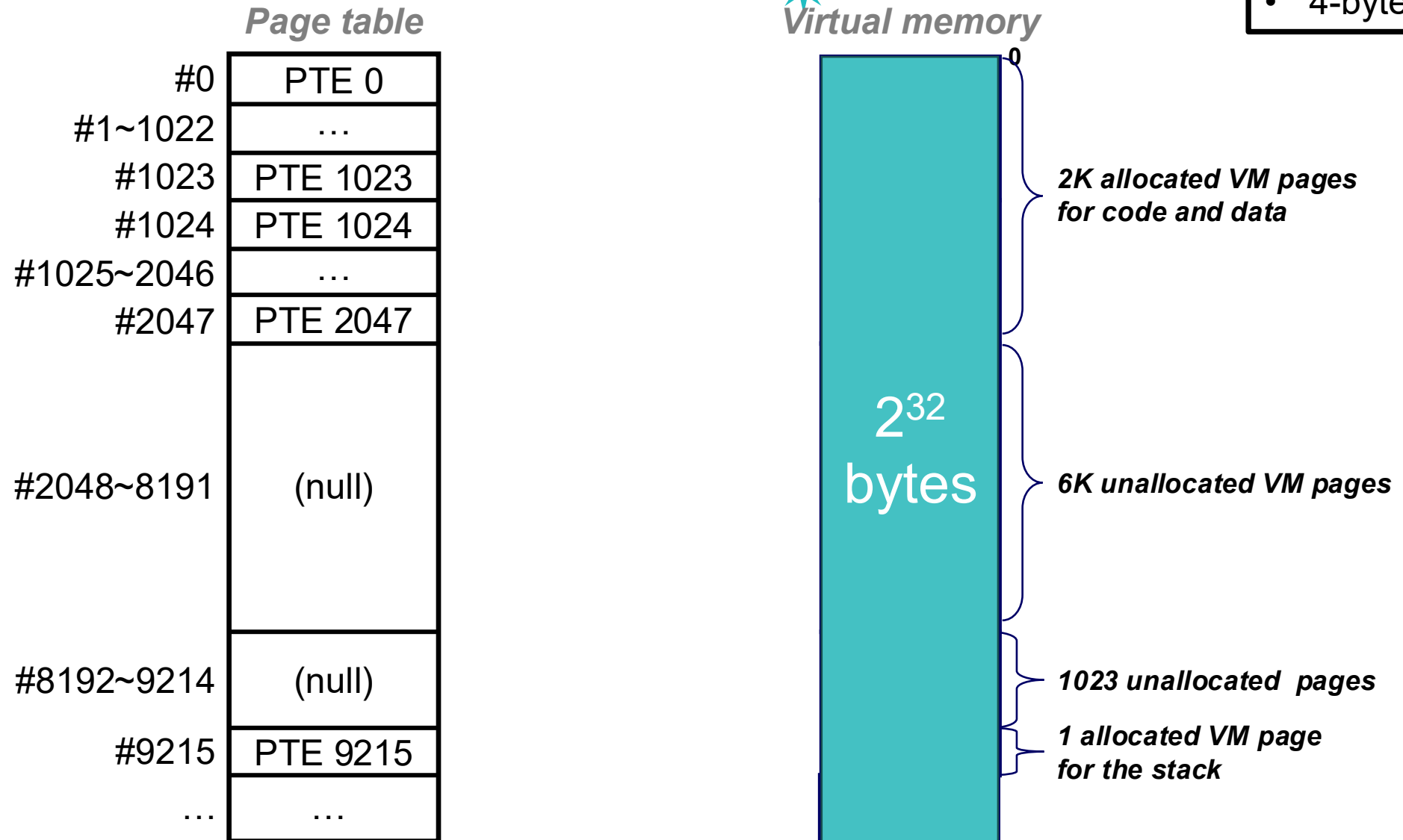


We Have a Problem!: Space Consumption 39



What Is the # of Page Table Entries?

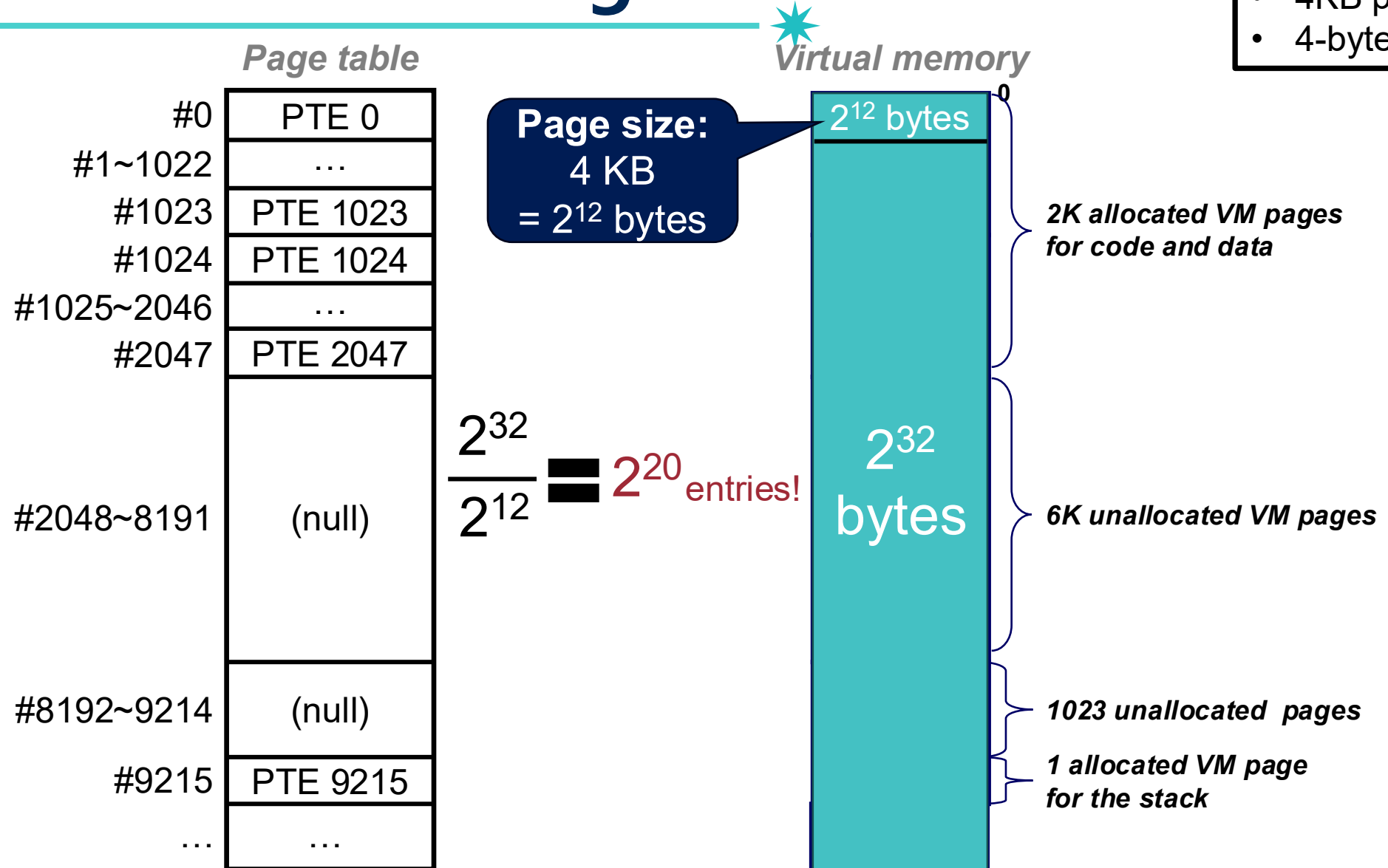
- Assumption:**
- 32 bit addresses
 - 4KB pages
 - 4-byte PTEs



What Is the # of Page Table Entries?

Assumption:

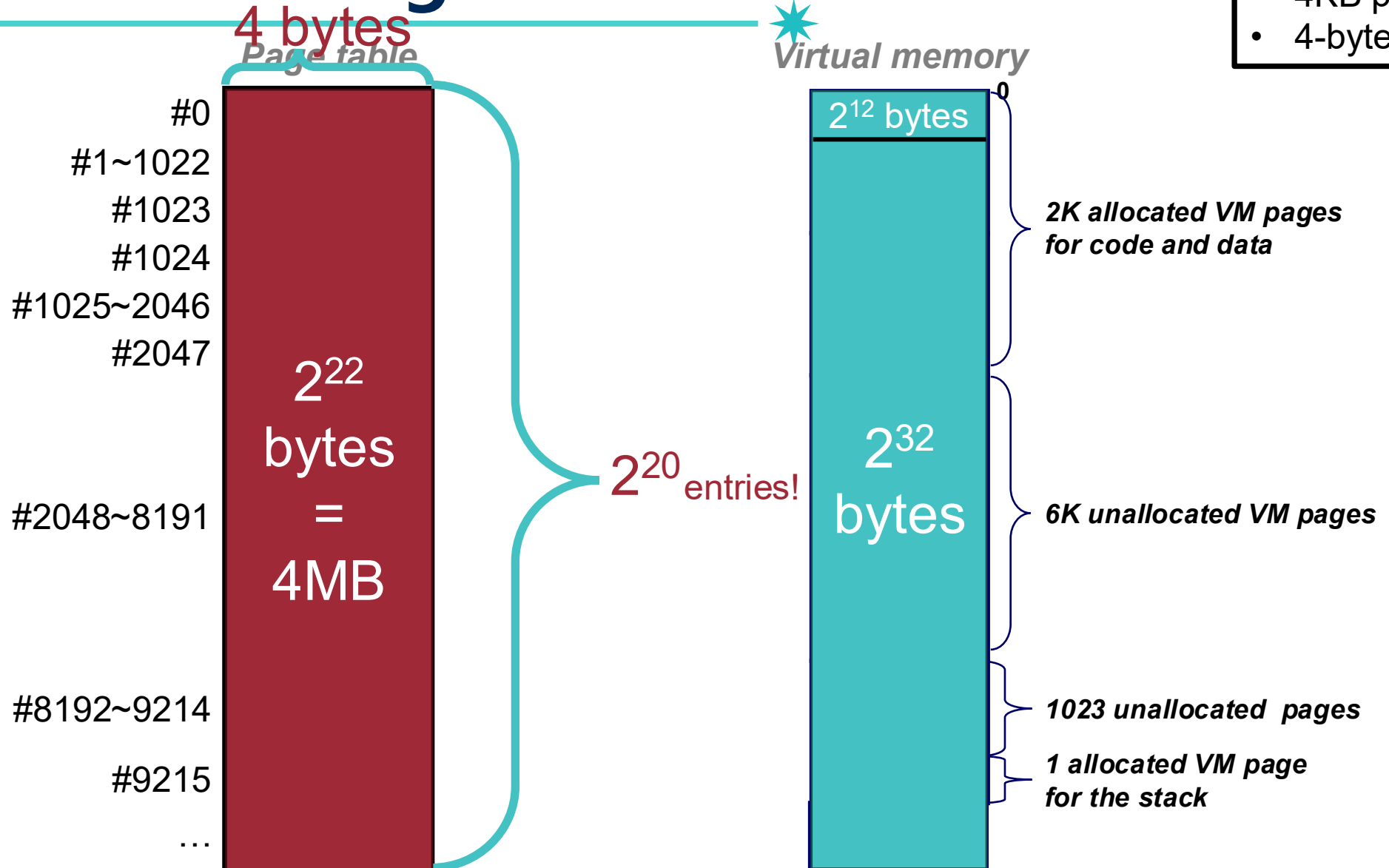
- 32 bit addresses
- 4KB pages
- 4-byte PTEs



What Is the Page Table Size?

Assumption:

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



Limitation of a Single-level Page Table

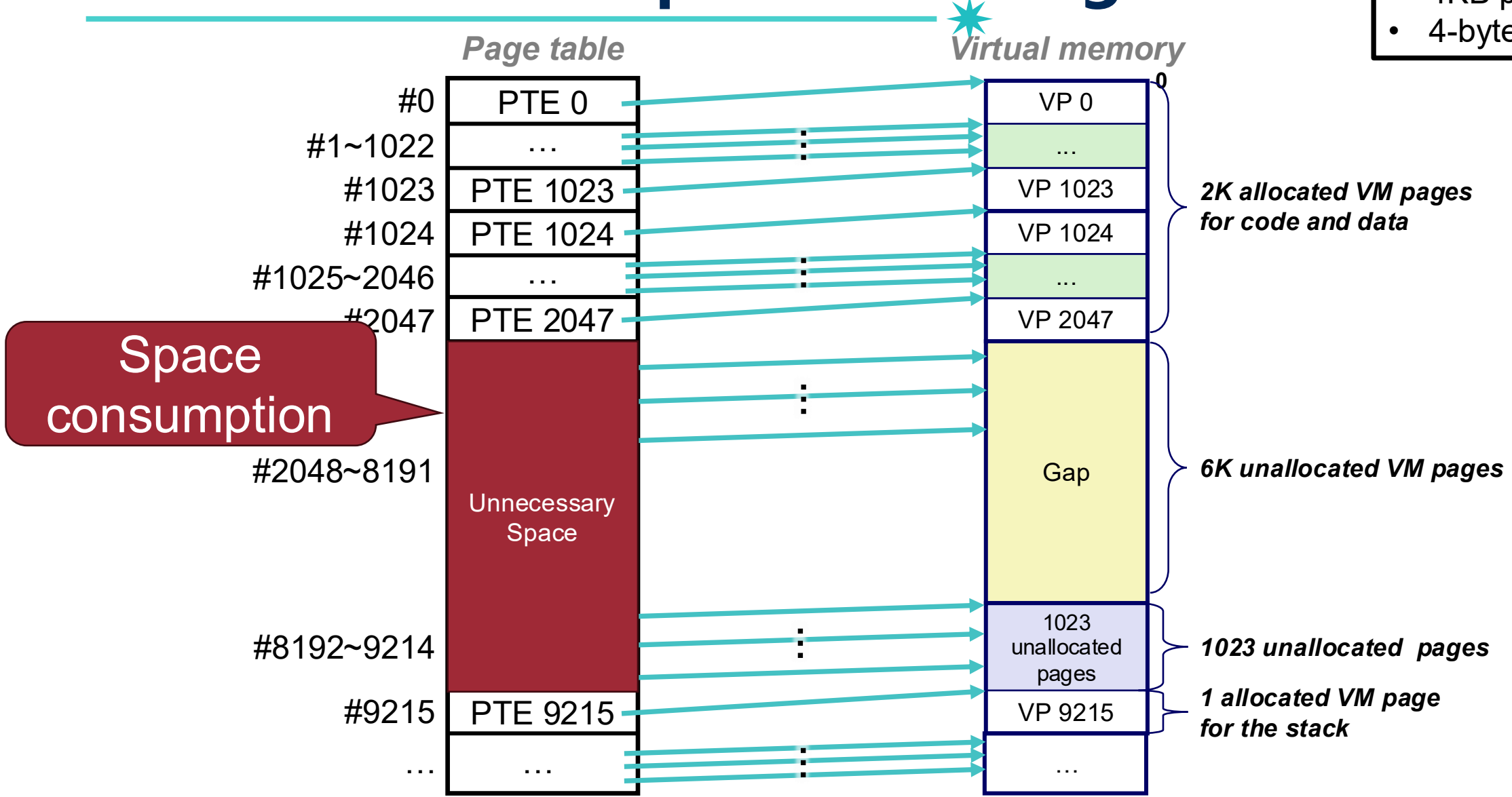


- Suppose #1:
 - 32 bit addresses, 4KB pages, 4-byte PTEs
 - For each program, we need a 4MB page table 😞
- Suppose #2:
 - 42 bit addresses, 4KB pages, 8-byte PTEs
 - For each program, we need a 512GB page table 😞

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

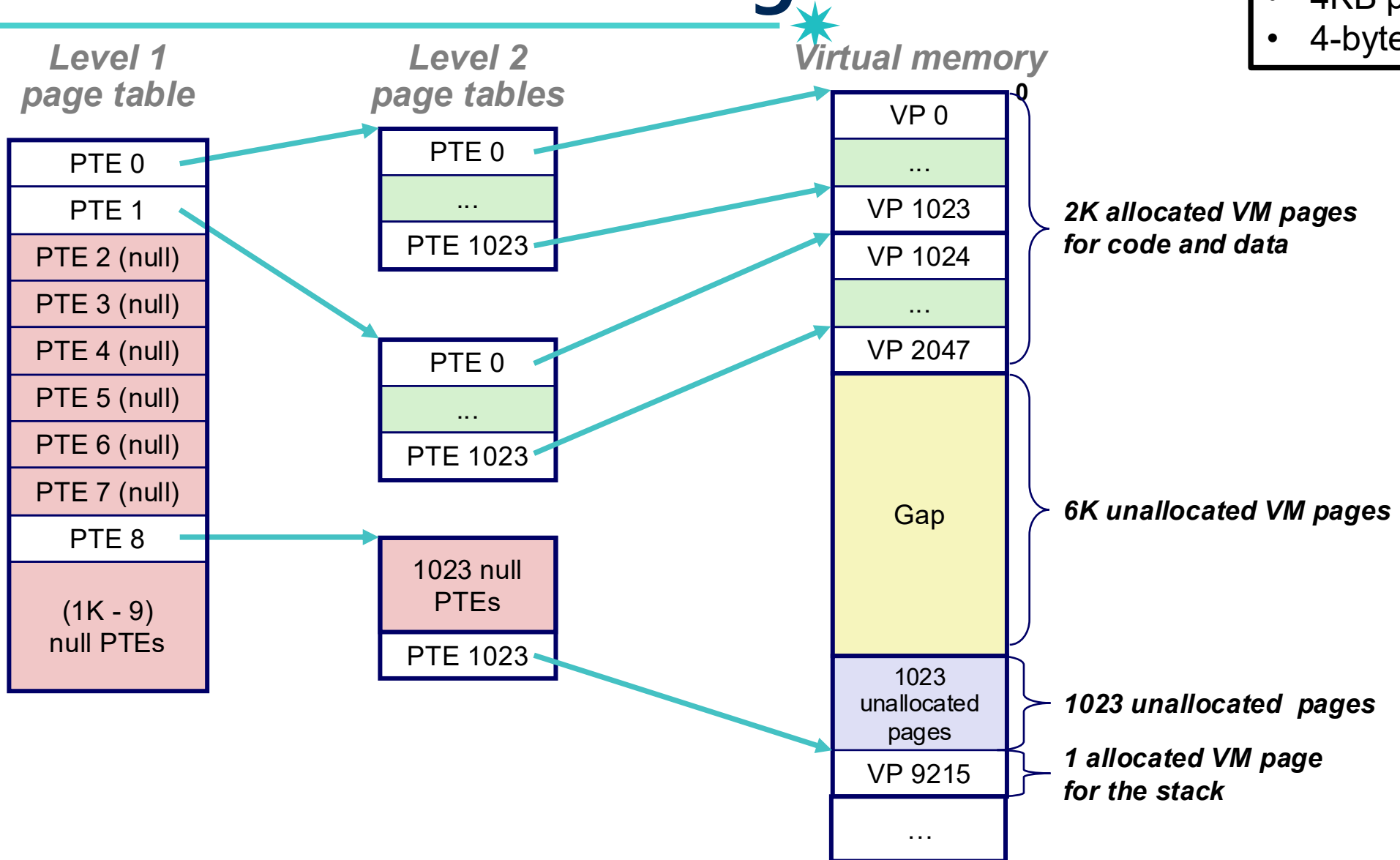
Root Cause of Space Bloating

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



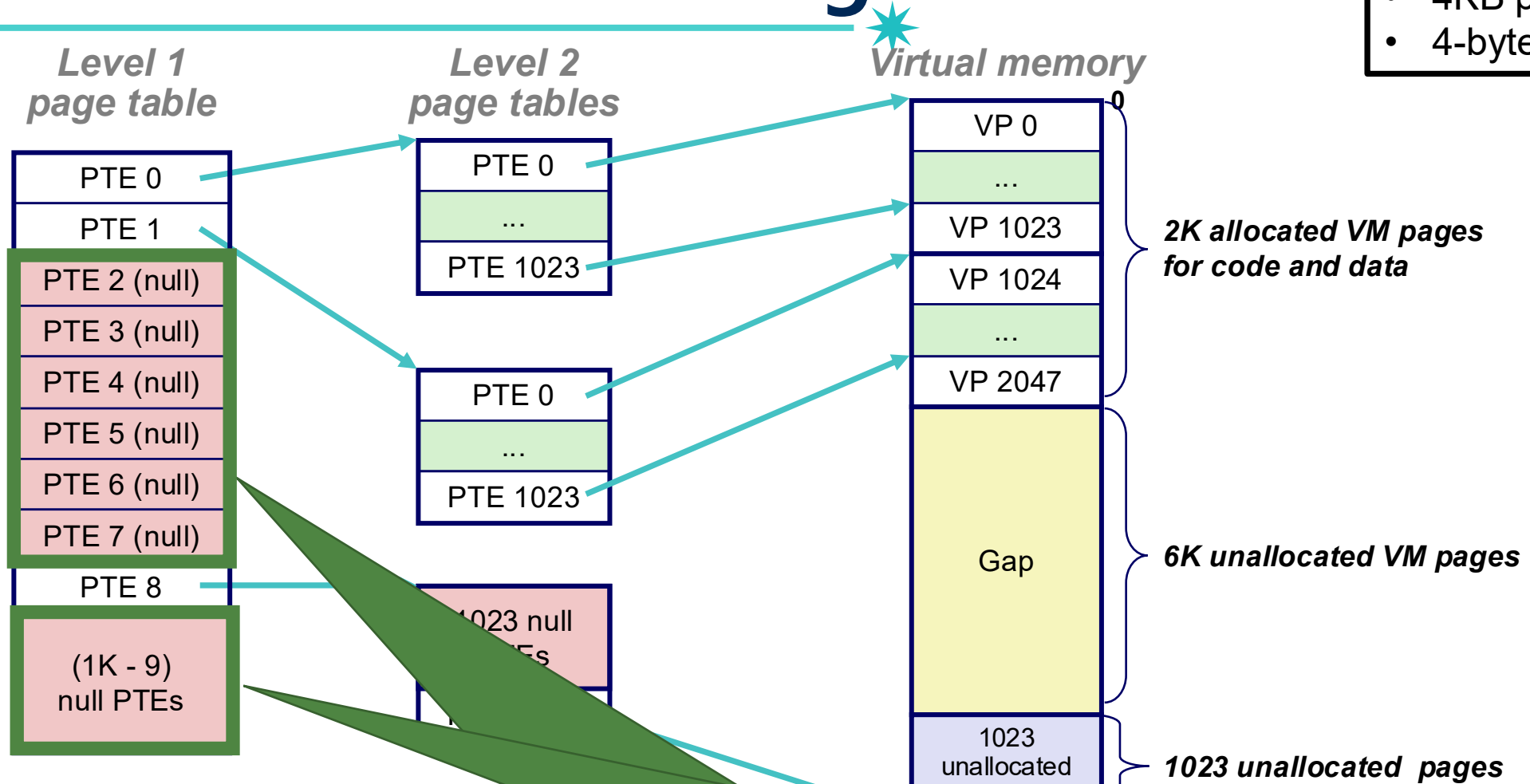
Solution: Multi-level Page Tables

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



Solution: Multi-level Page Tables

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

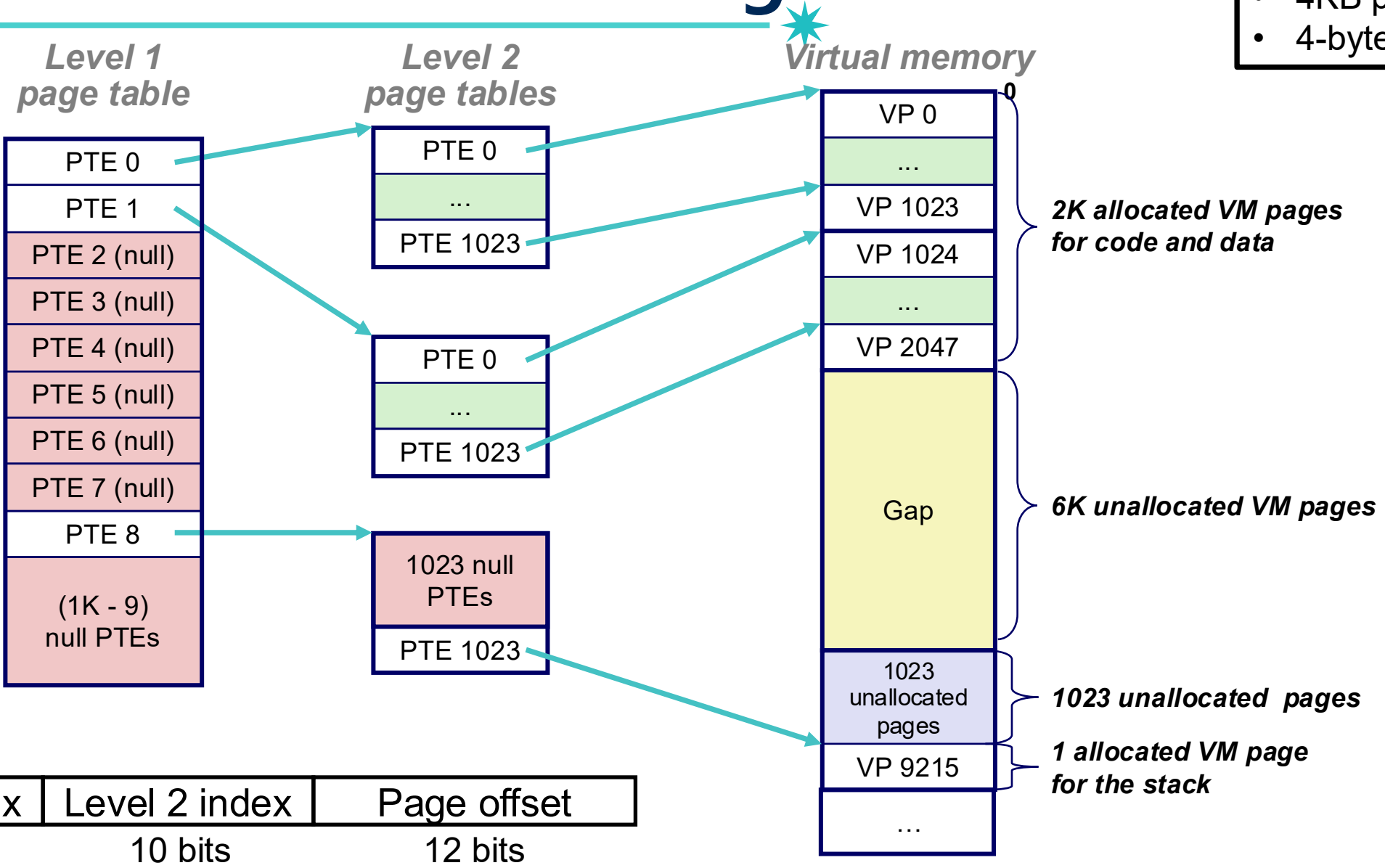


For unallocated virtual pages, no second-level page table is allocated

...ted VM page stack

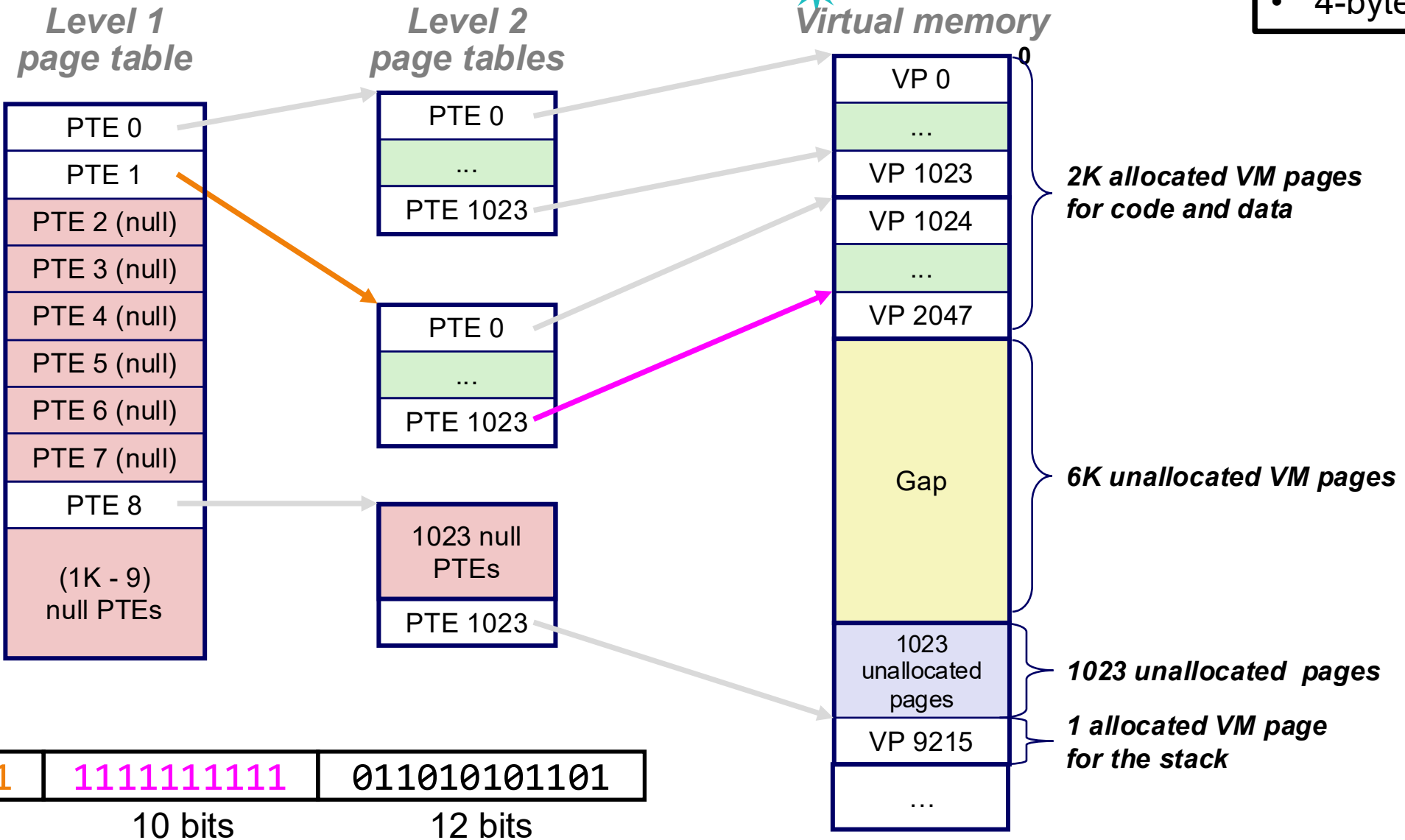
Solution: Multi-level Page Tables

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



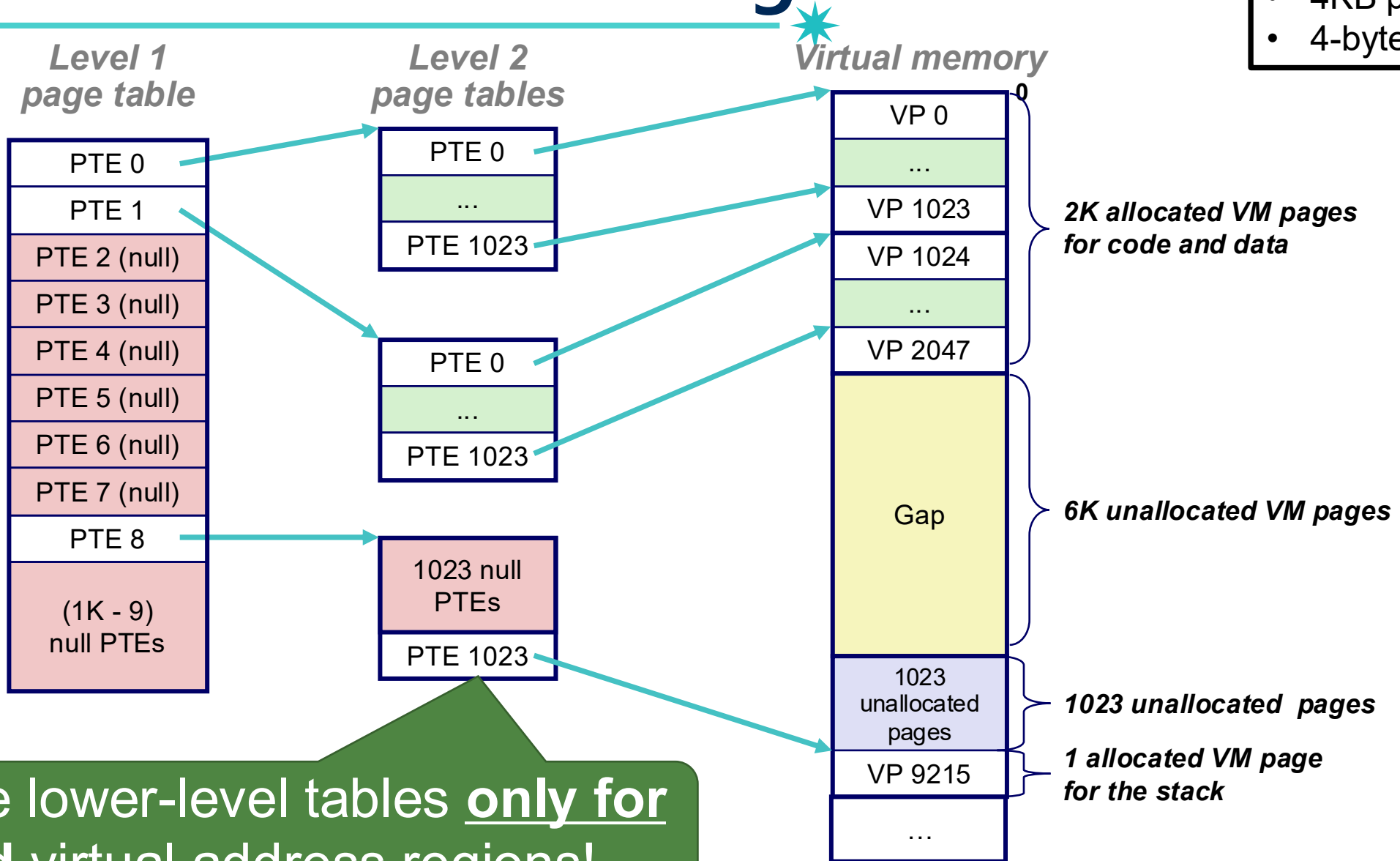
Solution: Multi-level Page Tables

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



Solution: Multi-level Page Tables

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

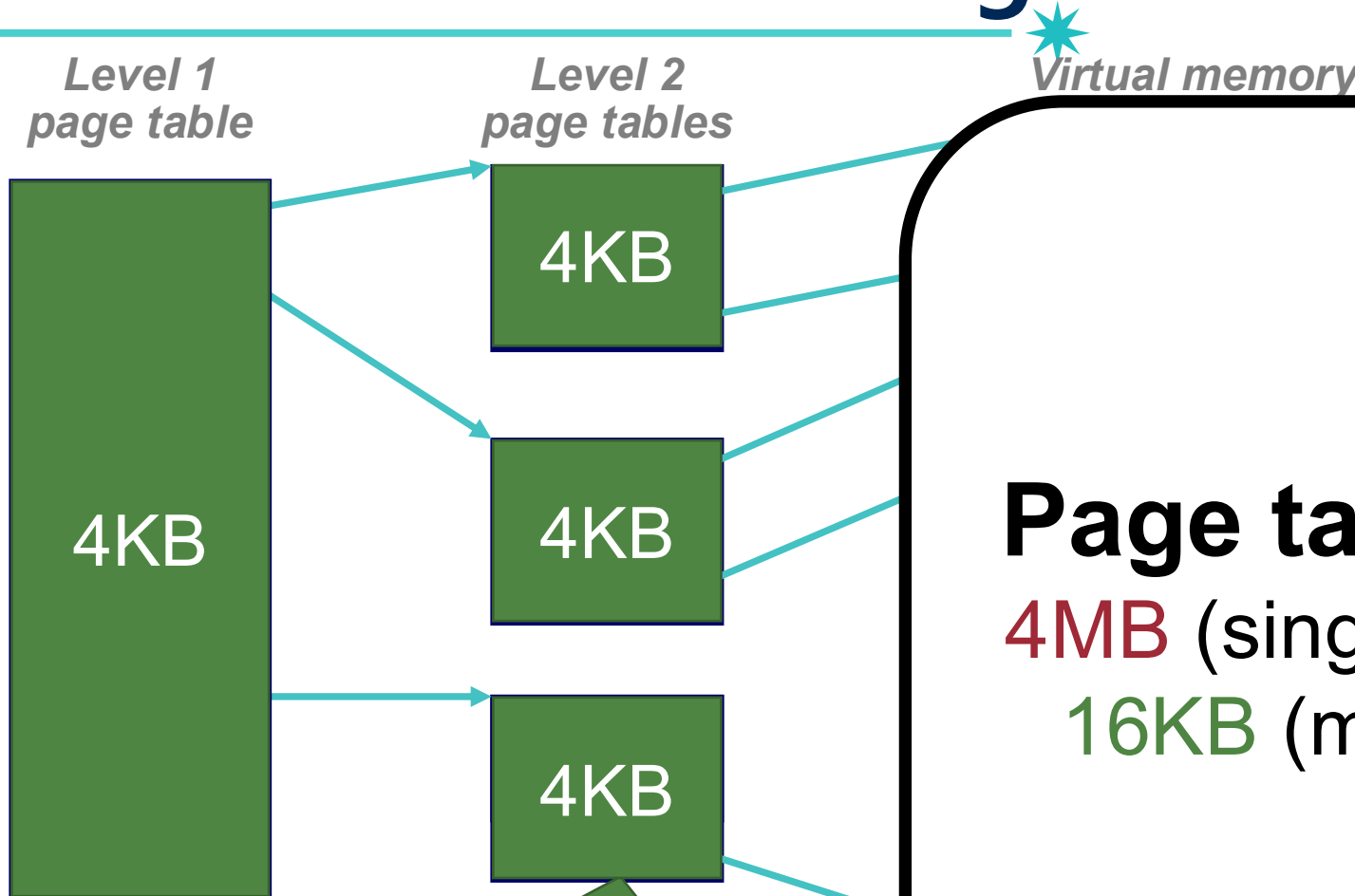


Allocate lower-level tables only for used virtual address regions!

Solution: Multi-level Page Tables

Assumption:

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



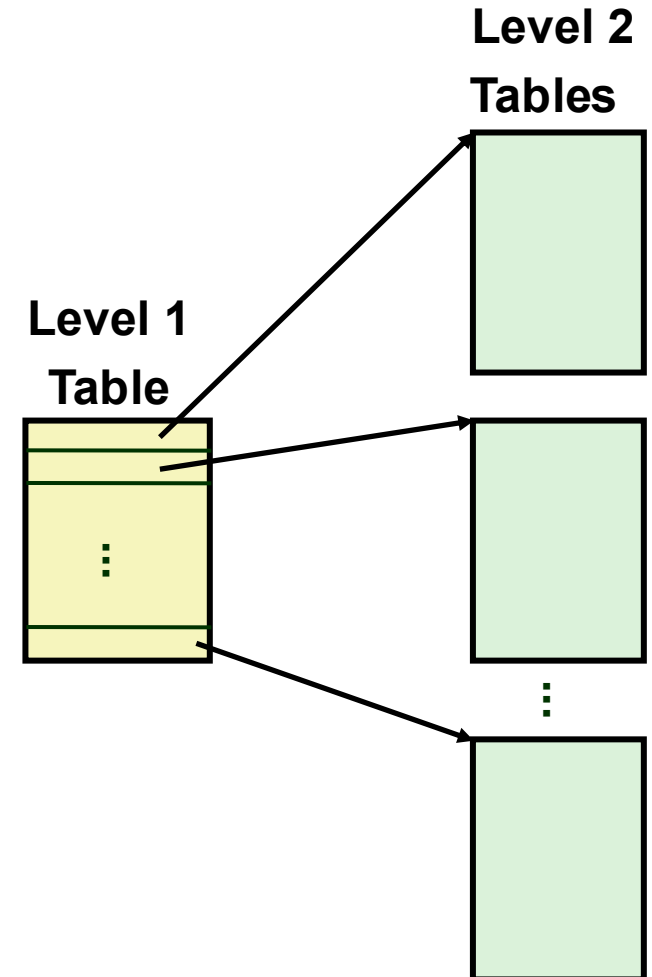
Page table size:
4MB (single-level) →
16KB (multi-level)

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

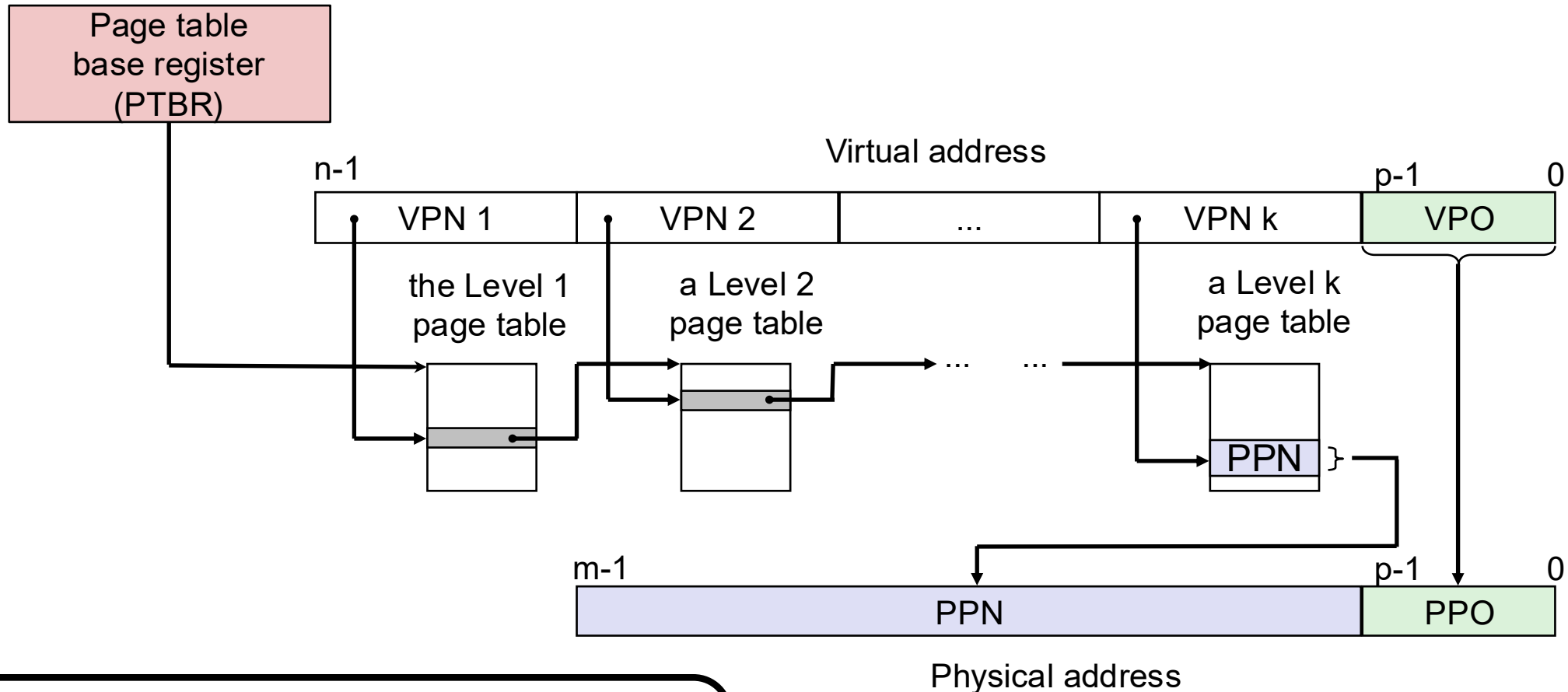
Multi-level Page Tables



- 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: Physical 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?