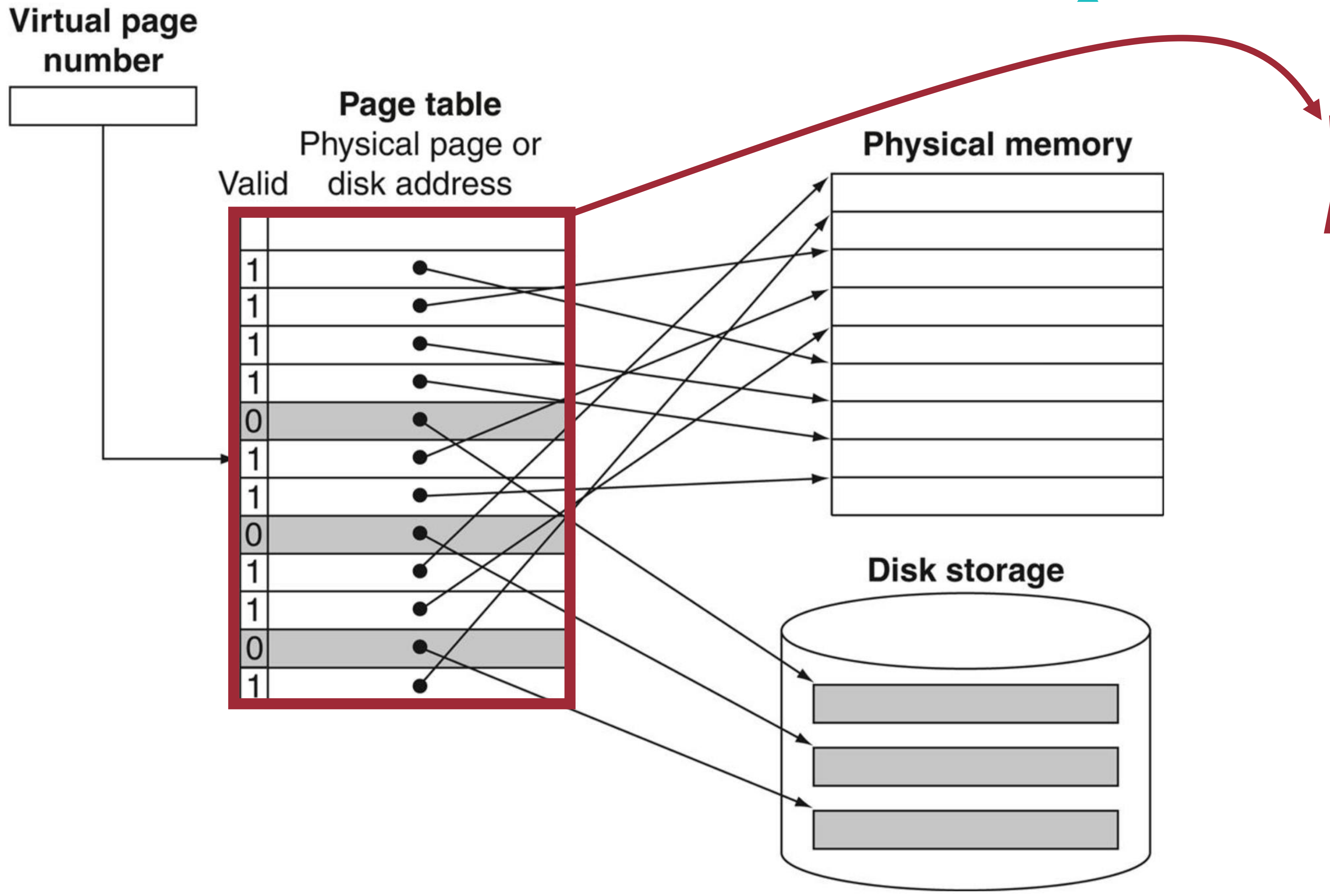
CSE261: Computer Architecture

19. Memory Hierarchy (5): Virtual Memory #2

Seongil Wi

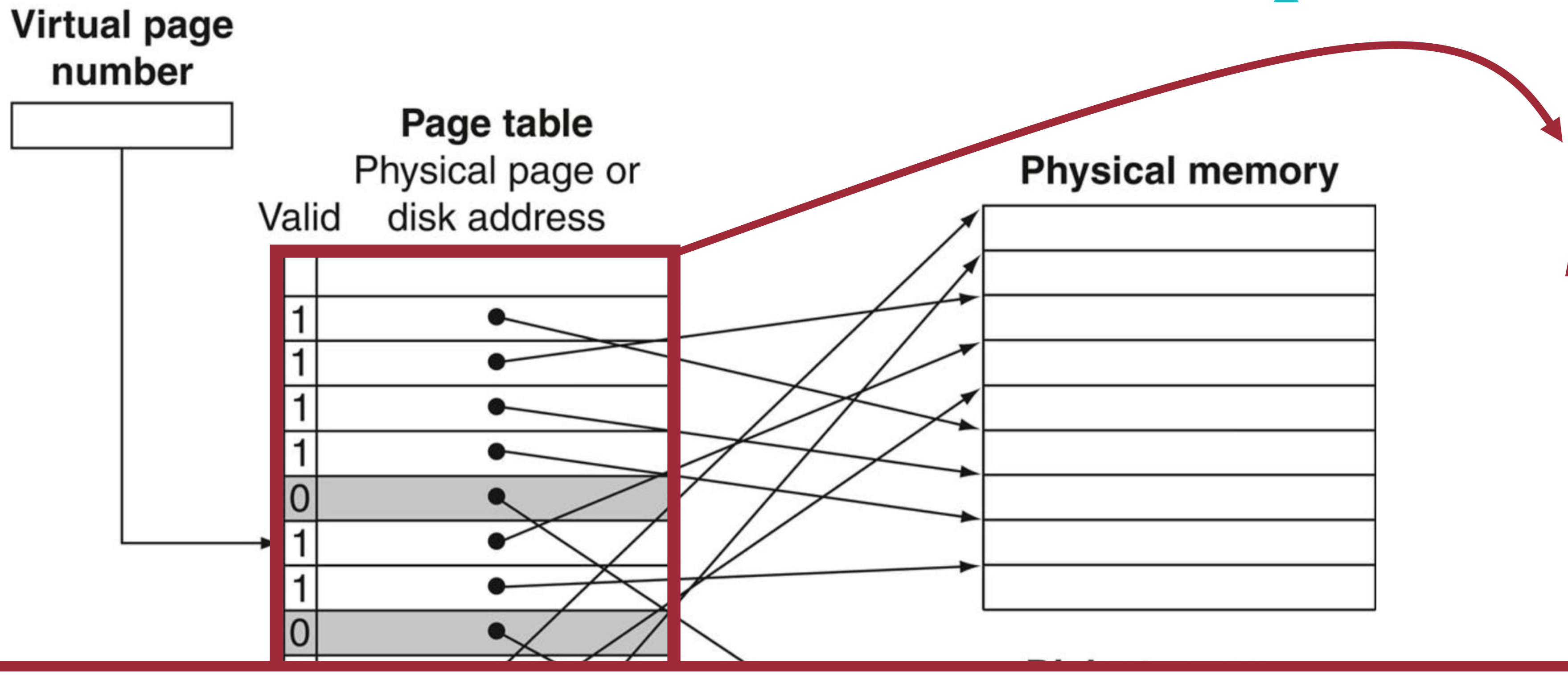
Place of the Page Table?



Where does the page table exist?

Physical memory!

Place of the Page Table?



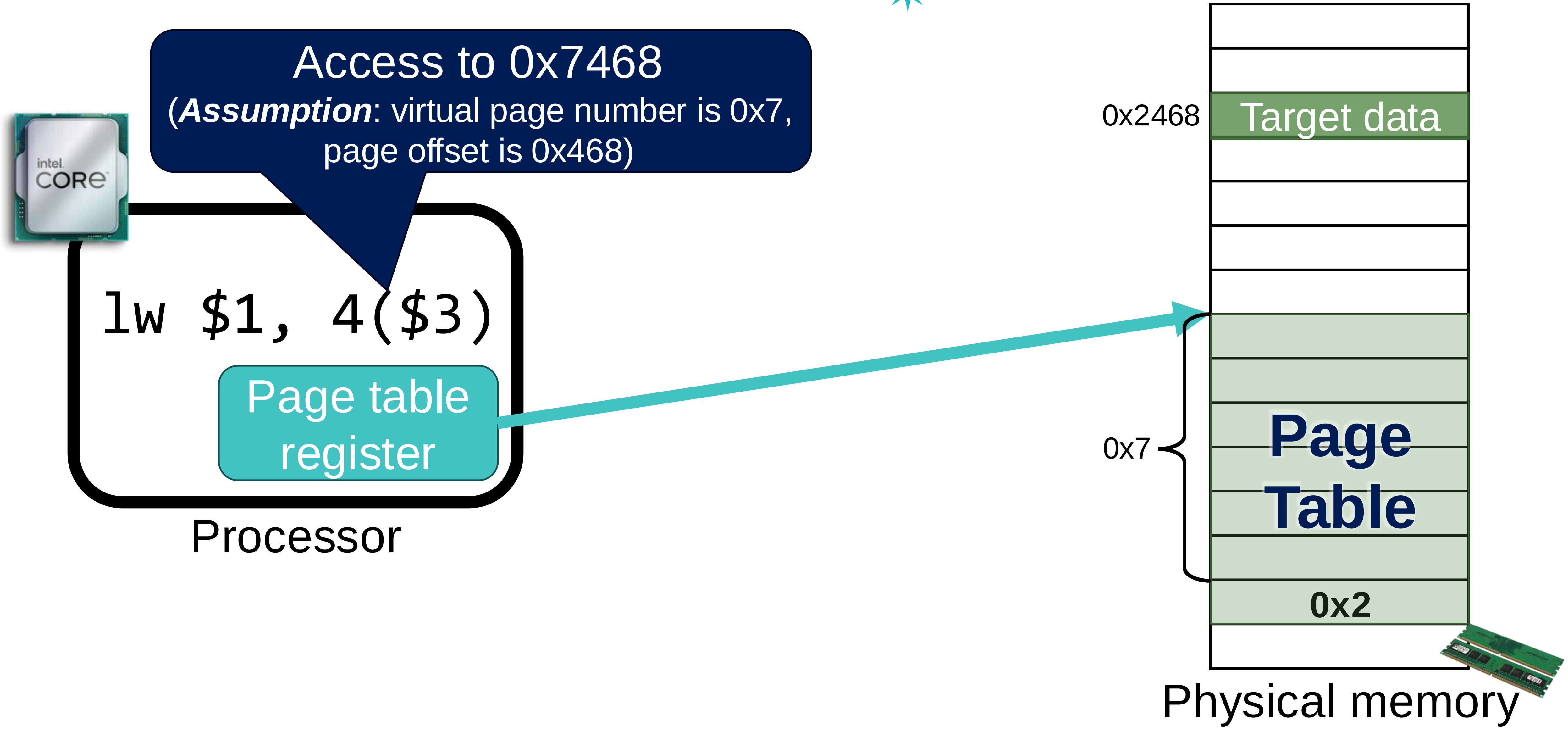
Where does the page table exist?

Physical memory!

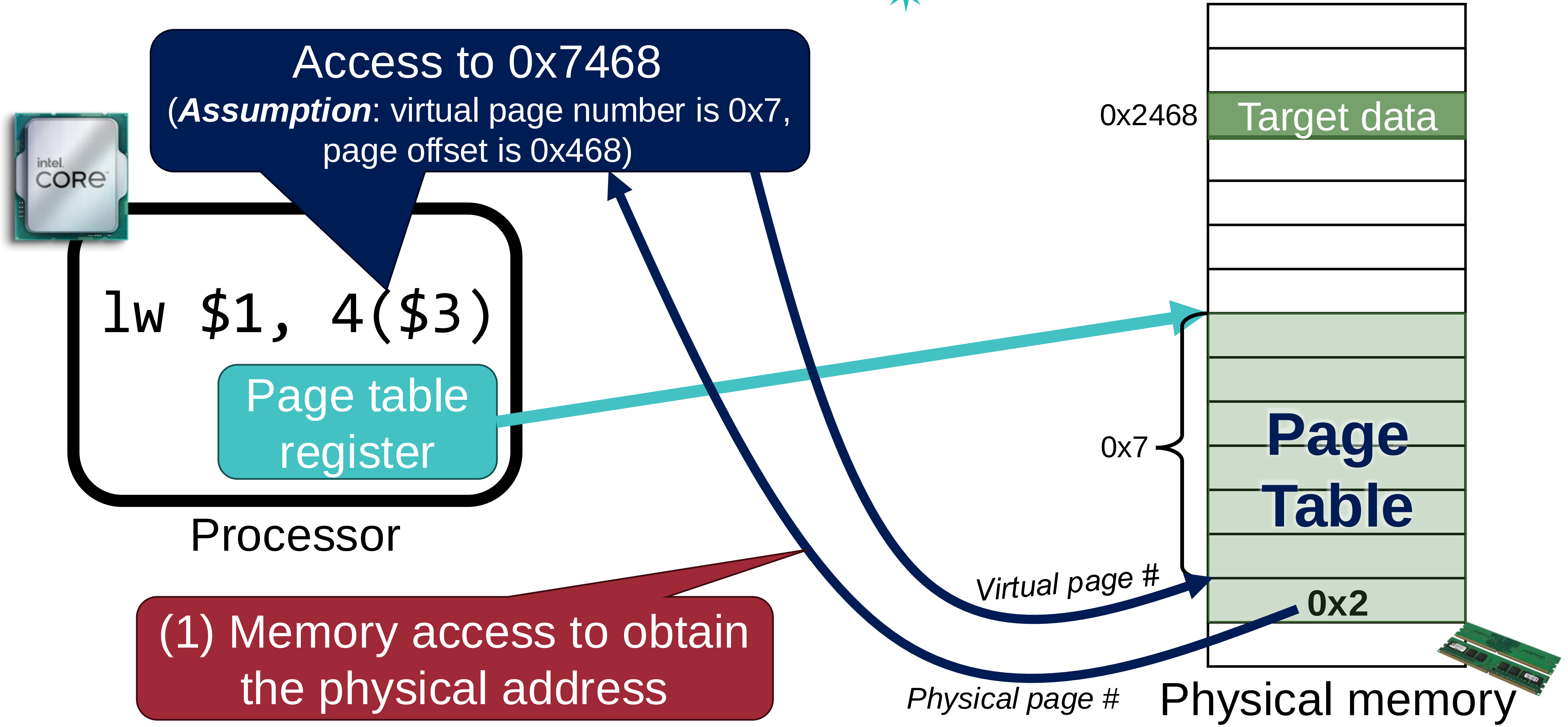
Any problems?

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

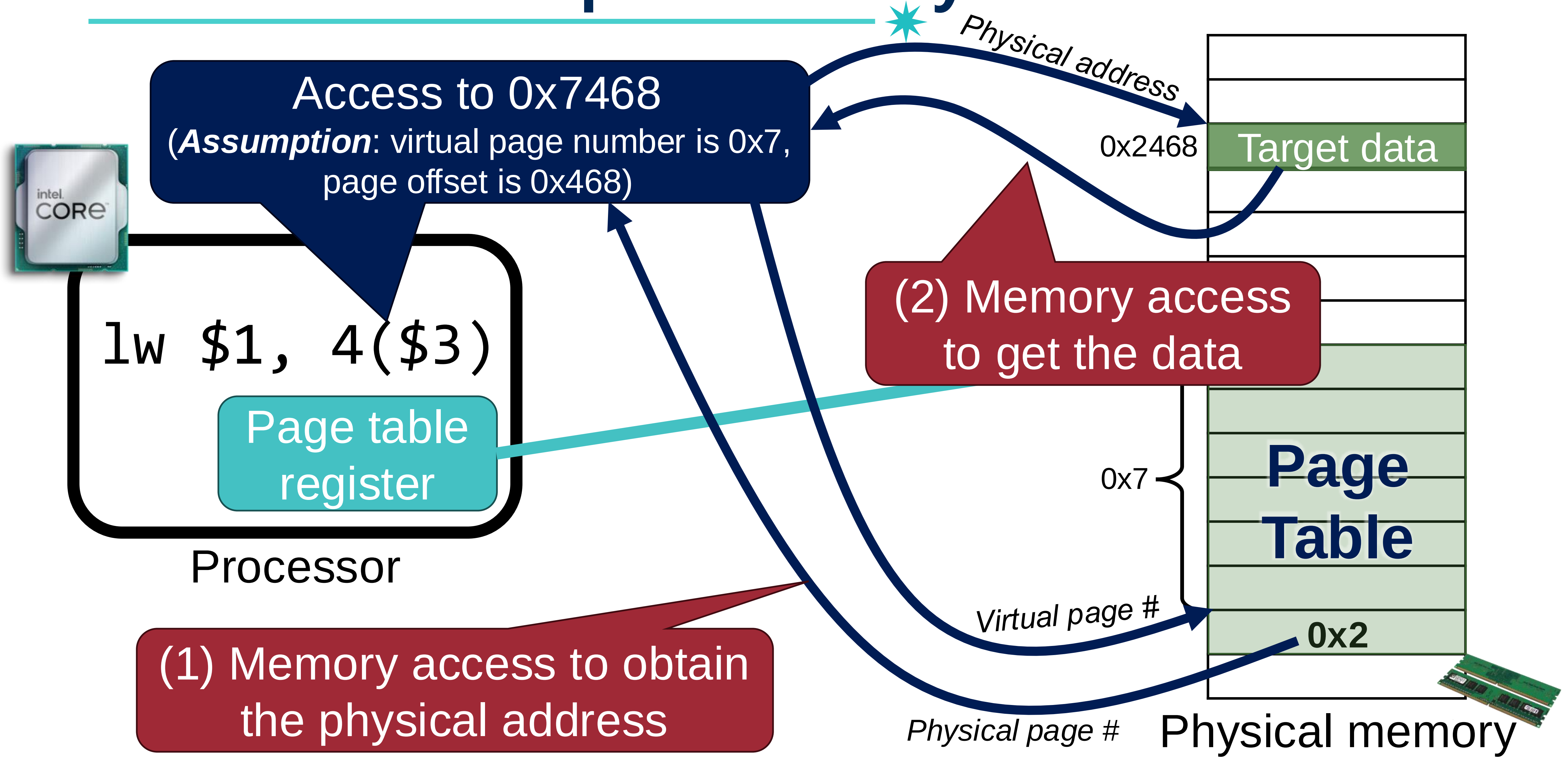
Problem: Multiple Memory Access



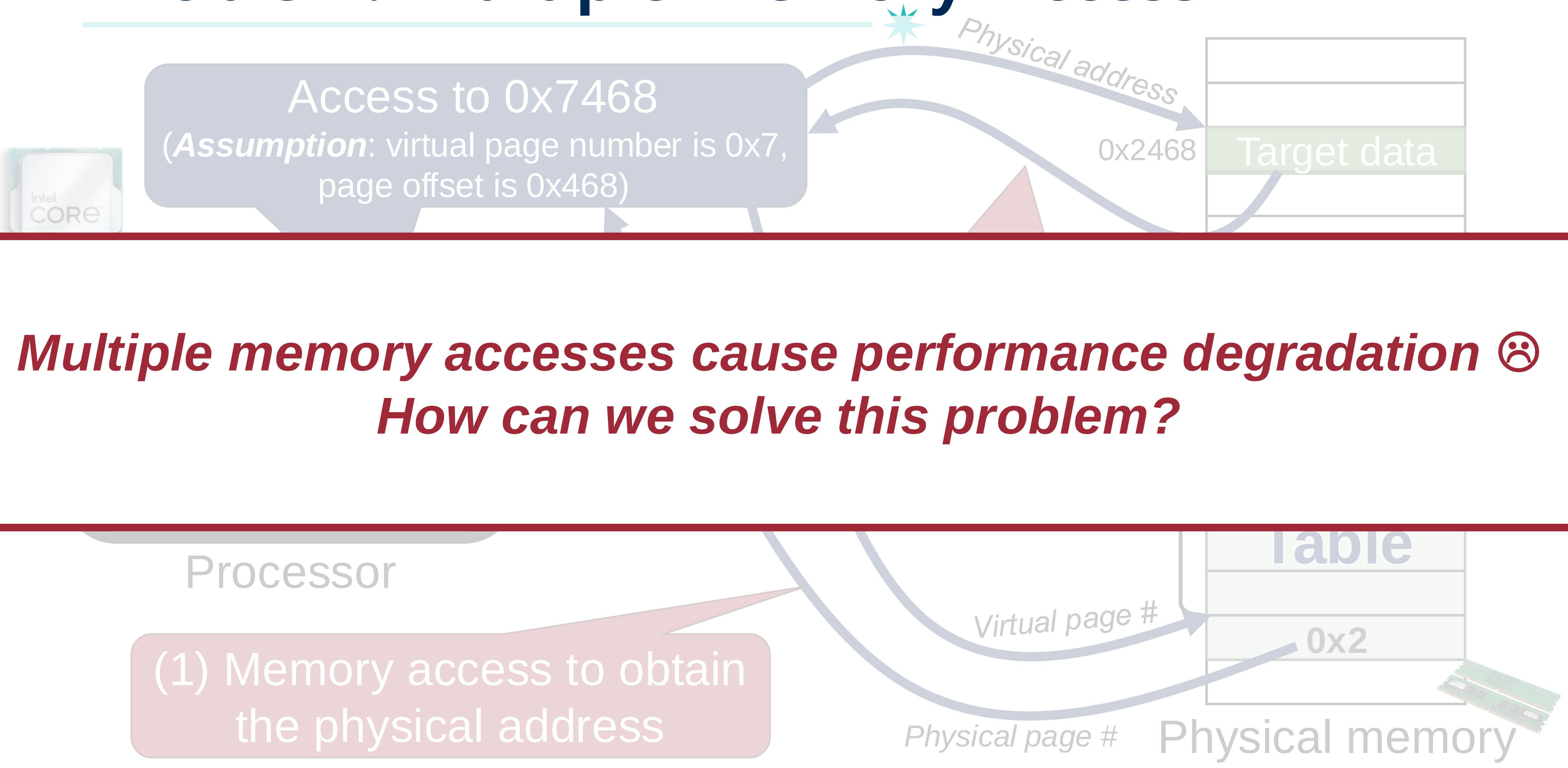
Problem: Multiple Memory Access



Problem: Multiple Memory Access

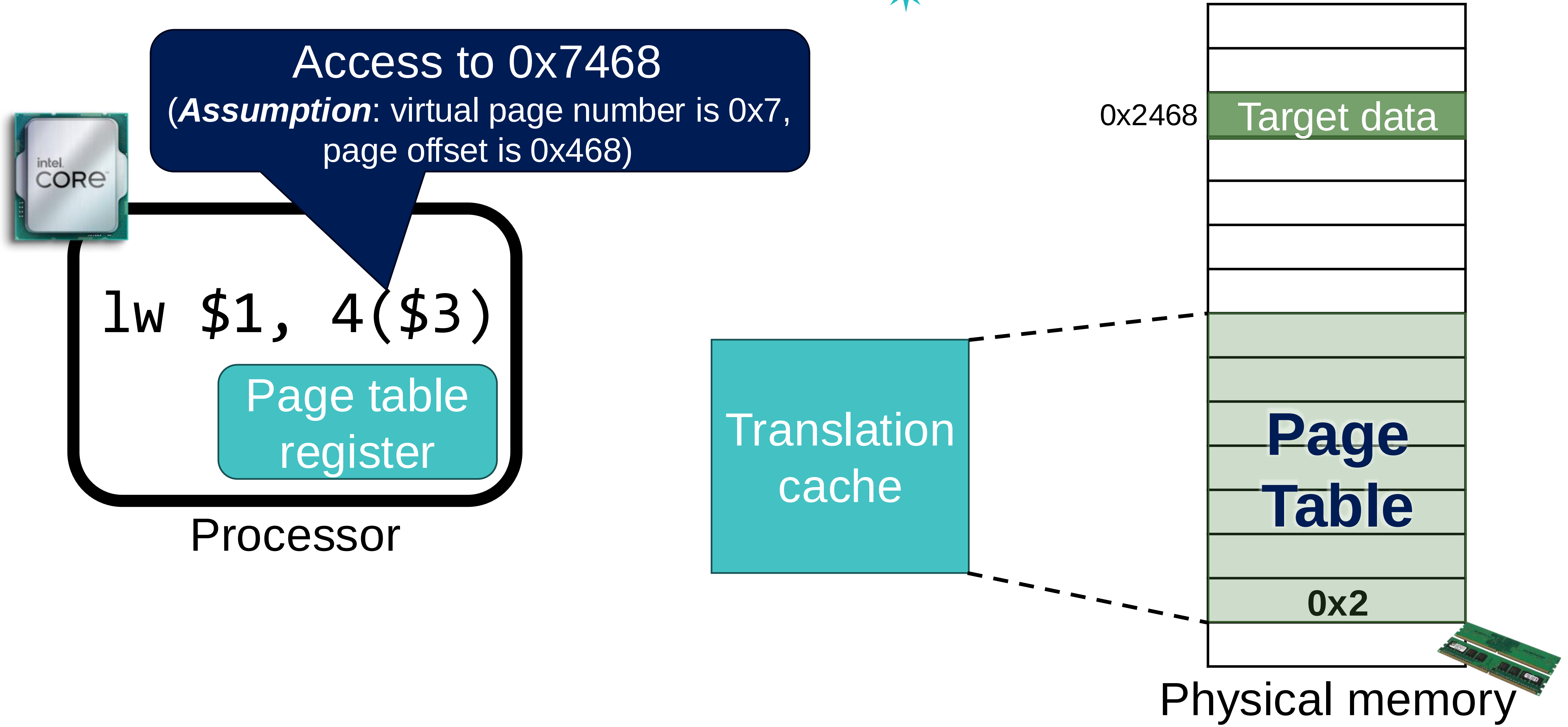


Problem: Multiple Memory Access

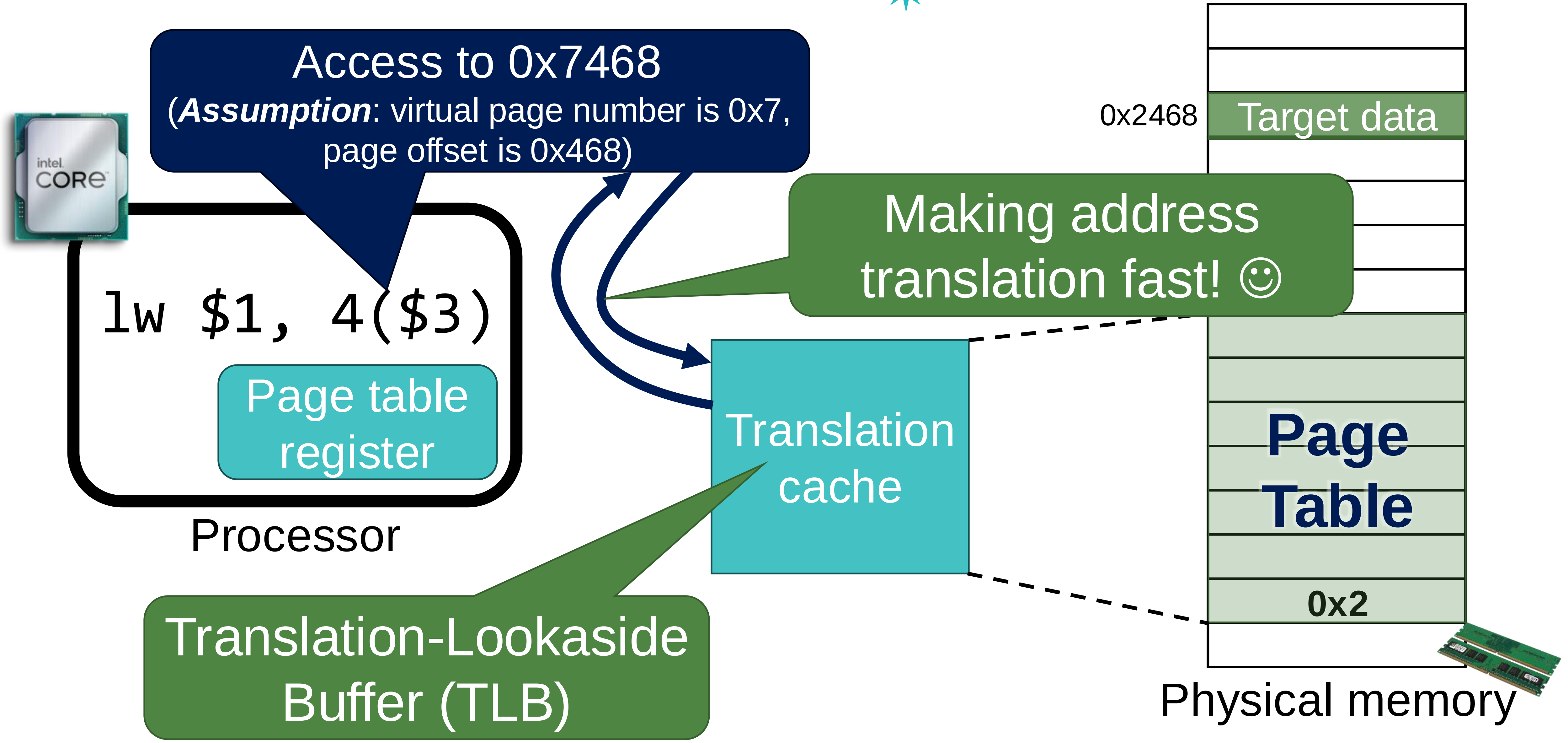

Access to 0x7468
(*Assumption*: virtual page number is 0x7, page offset is 0x468)
Physical address
0x2468
Target data
Table
Processor
Virtual page #
0x2
Physical page #
Physical memory
(1) Memory access to obtain the physical address

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

Solution: A Cache for Address Translation



Solution: A Cache for Address Translation


Access to 0x7468
(*Assumption*: virtual page number is 0x7, page offset is 0x468)
lw $1, 4($3)
Page table register
Processor
Making address translation fast! ☺
Translation cache
Translation-Lookaside Buffer (TLB)
0x2468
Target data
Page Table
0x2
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

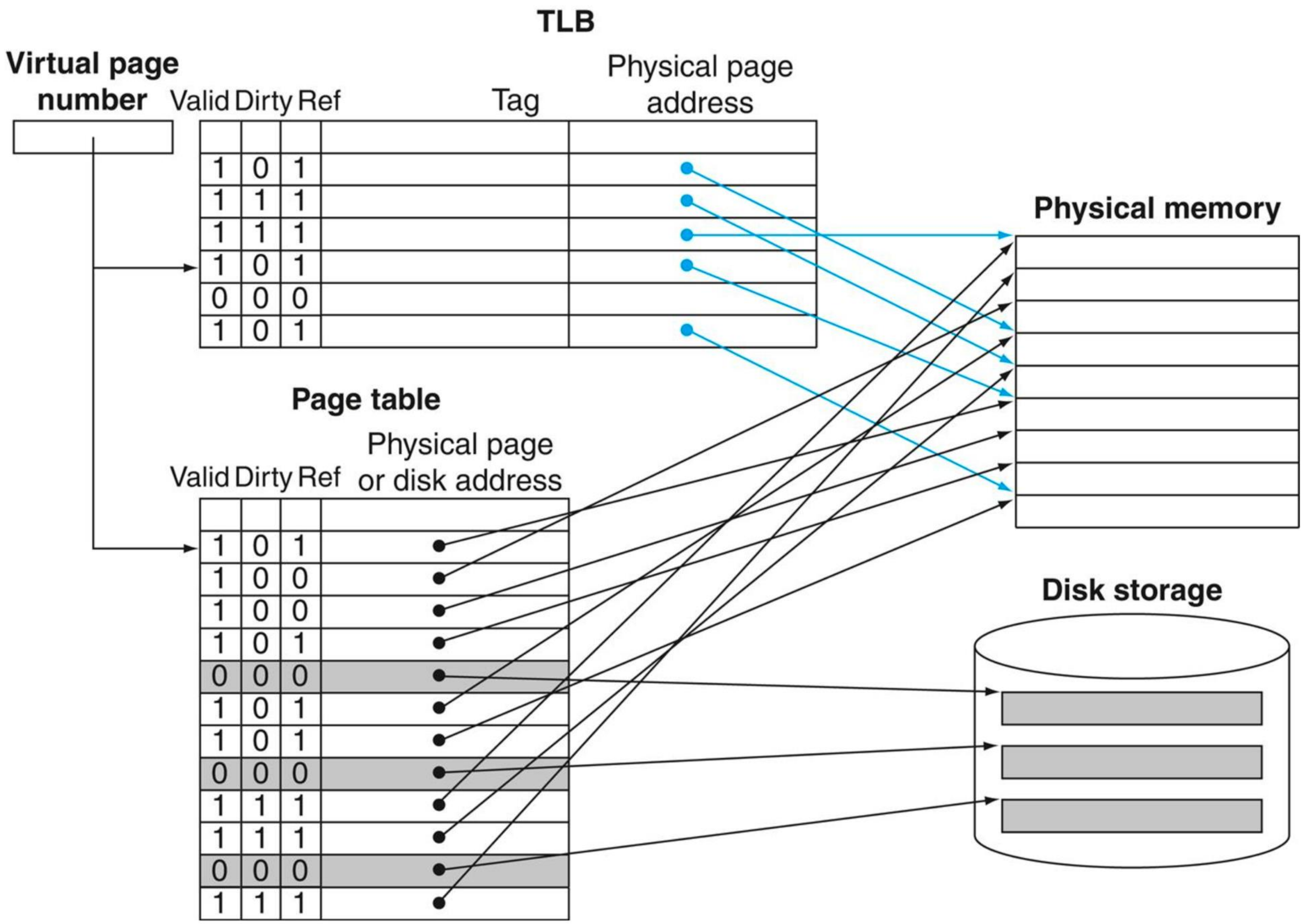
Translation-Lookaside Buffer (TLB)

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

- 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 **<u>in the near future</u>**"*

Translation-Lookaside Buffer (TLB)



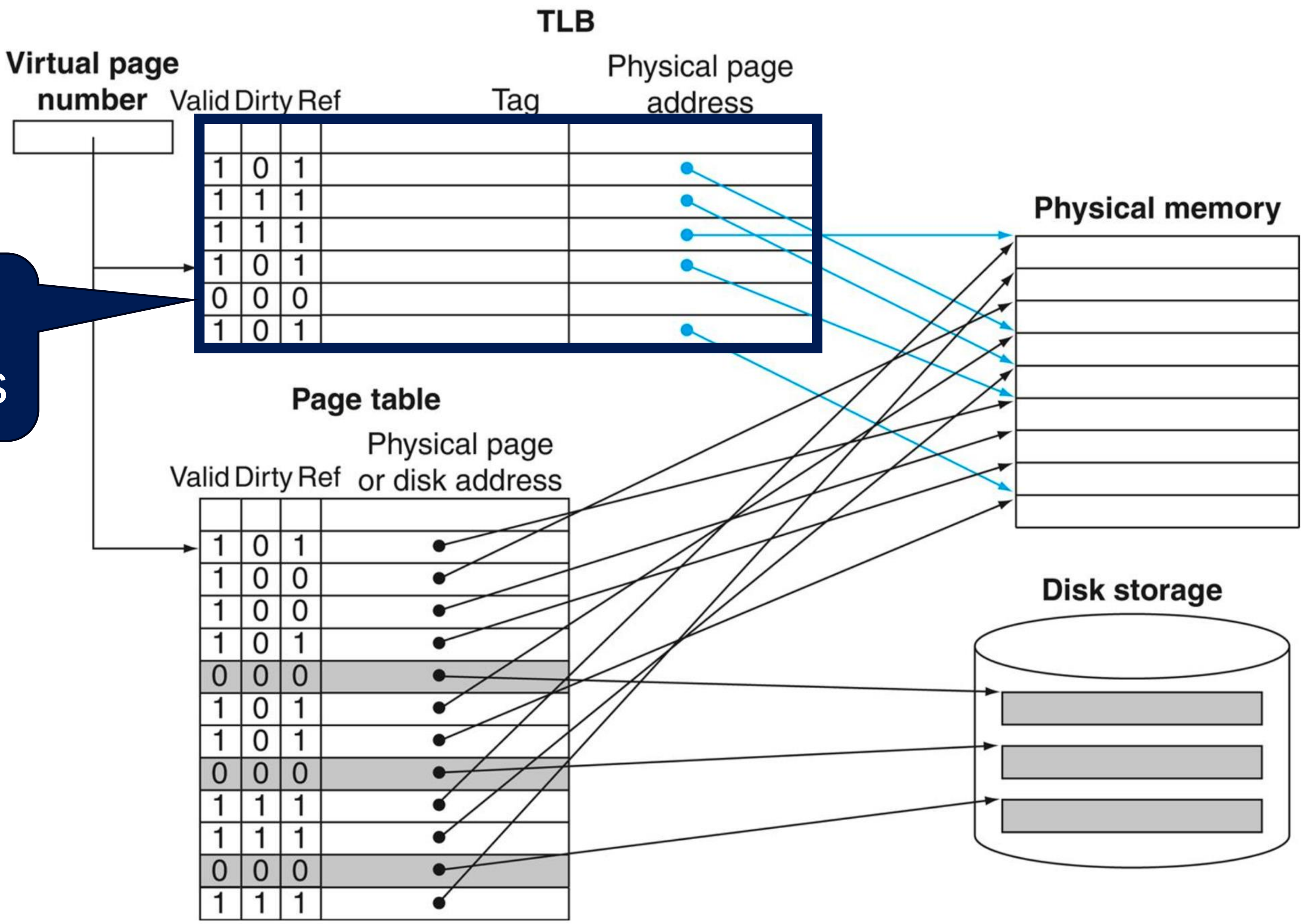
TLB

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

Page table

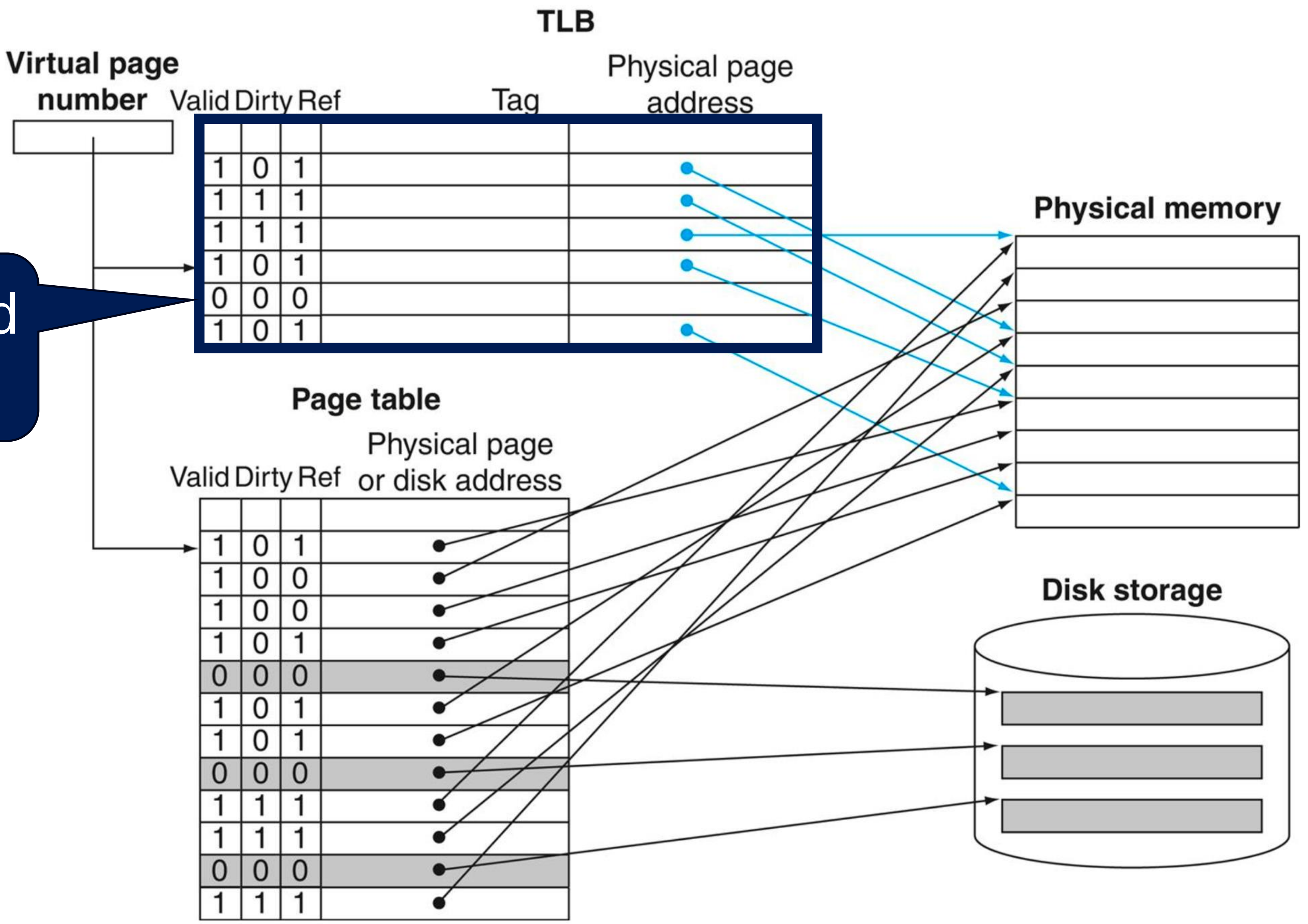
Valid	Dirty	Ref
1	0	1
1	0	0
1	0	0
1	0	1
0	0	0
1	0	1
1	0	1
0	0	0
1	1	1
1	1	1
0	0	0
1	1	1

Translation-Lookaside Buffer (TLB)



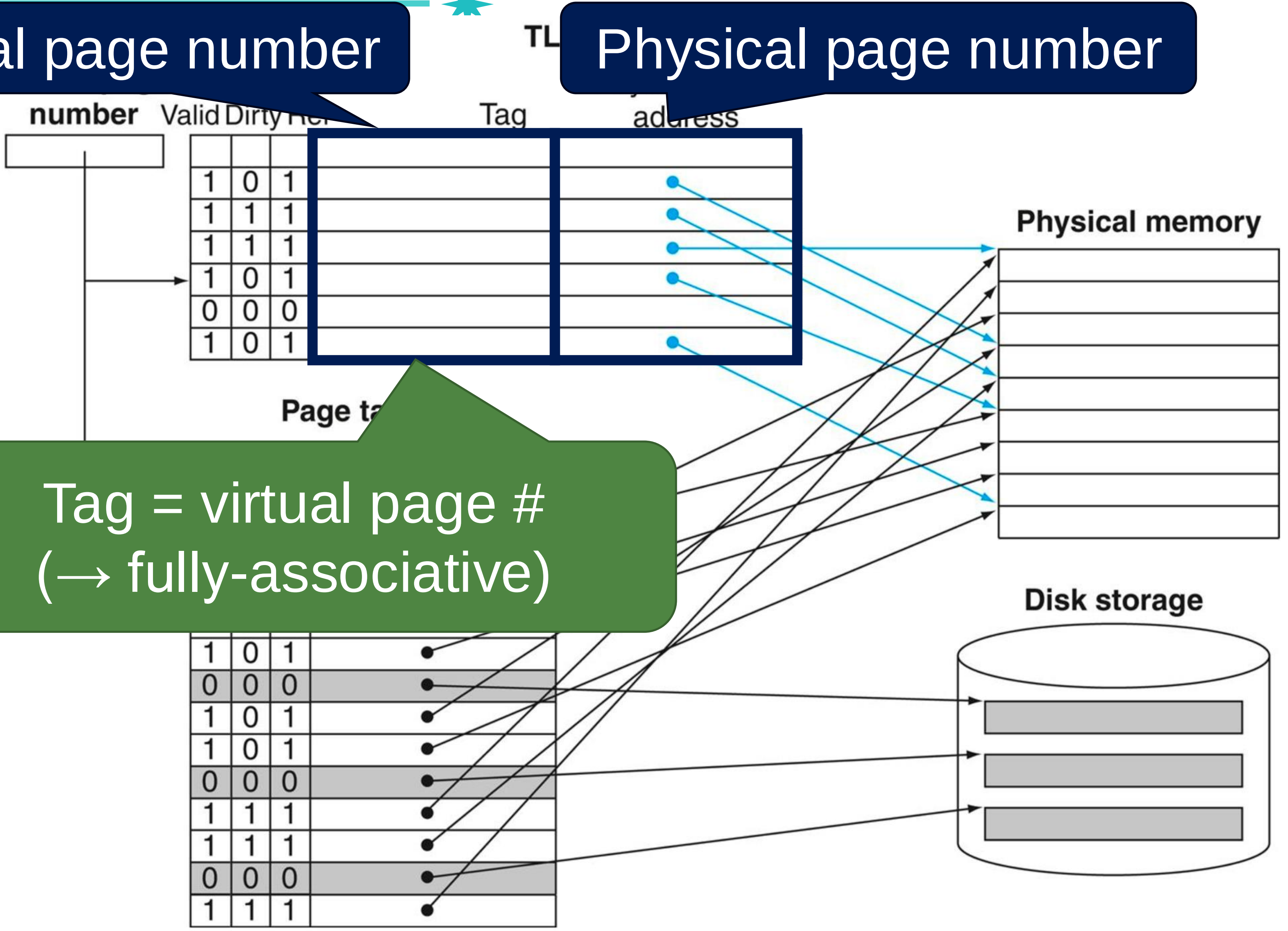
TLB: a special cache for address translations

Translation-Lookaside Buffer (TLB)



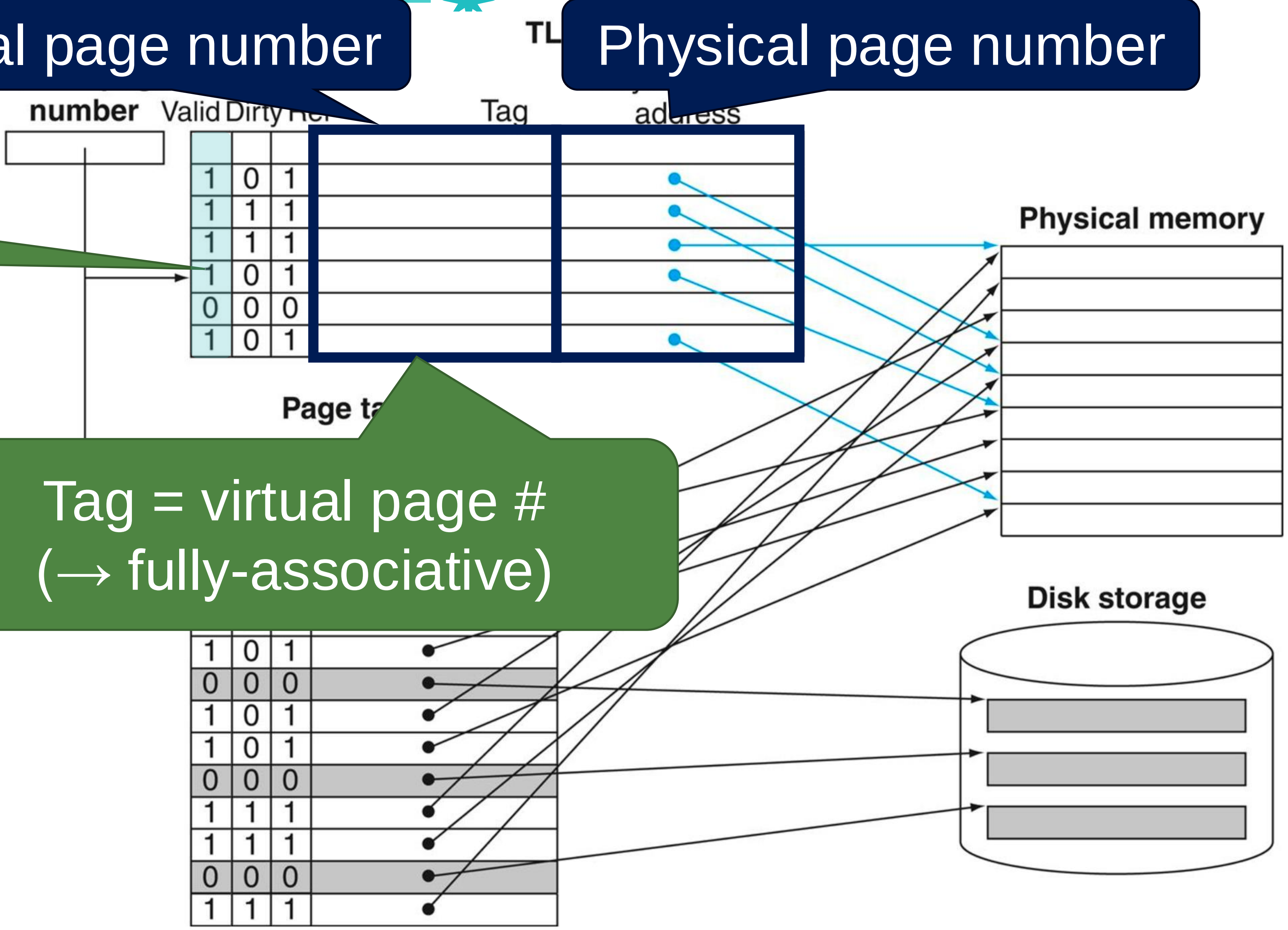
Stores the recently used address mappings

Translation-Lookaside Buffer (TLB)

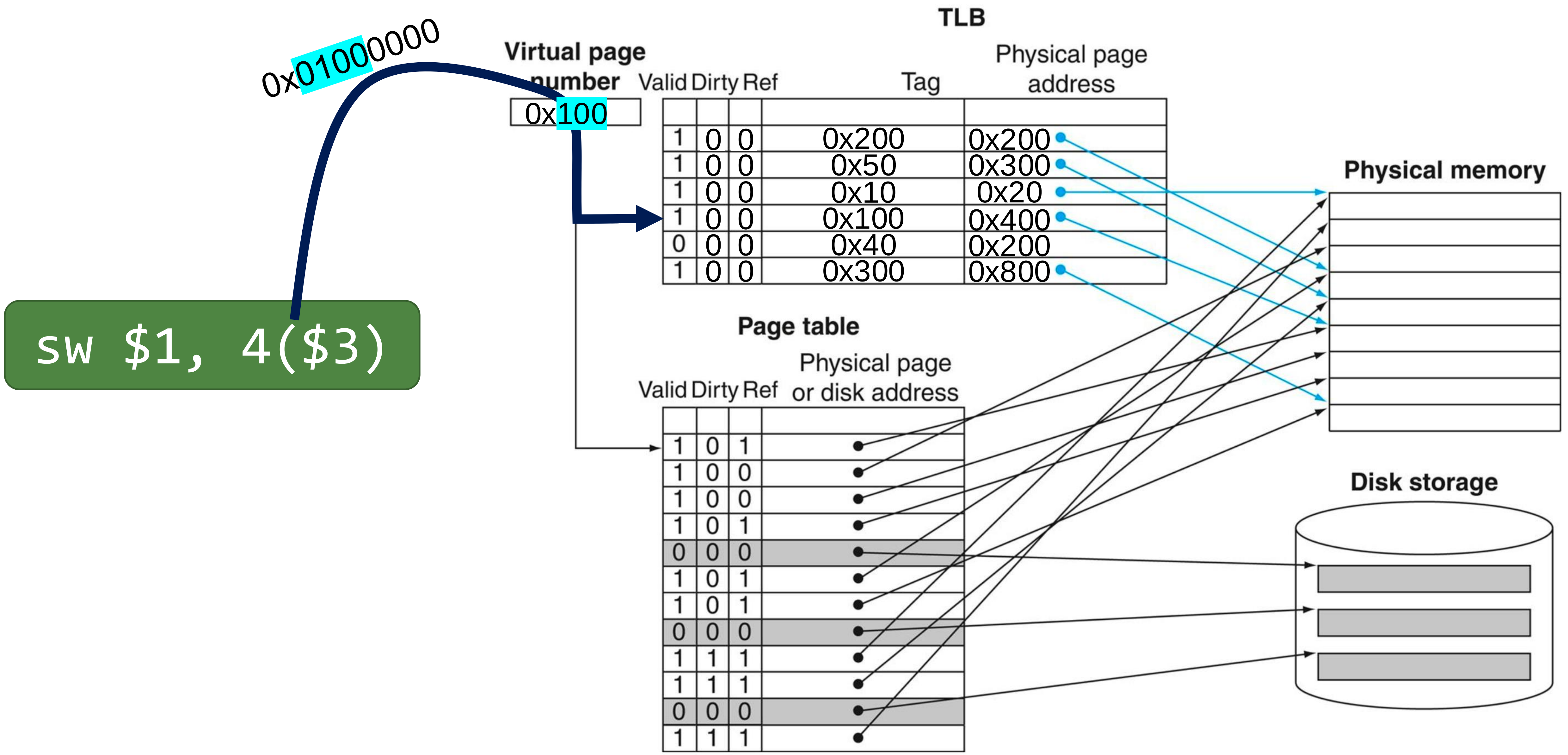


Tag = virtual page #
(→ fully-associative)

Translation-Lookaside Buffer (TLB)



Case #1: TLB Hits



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

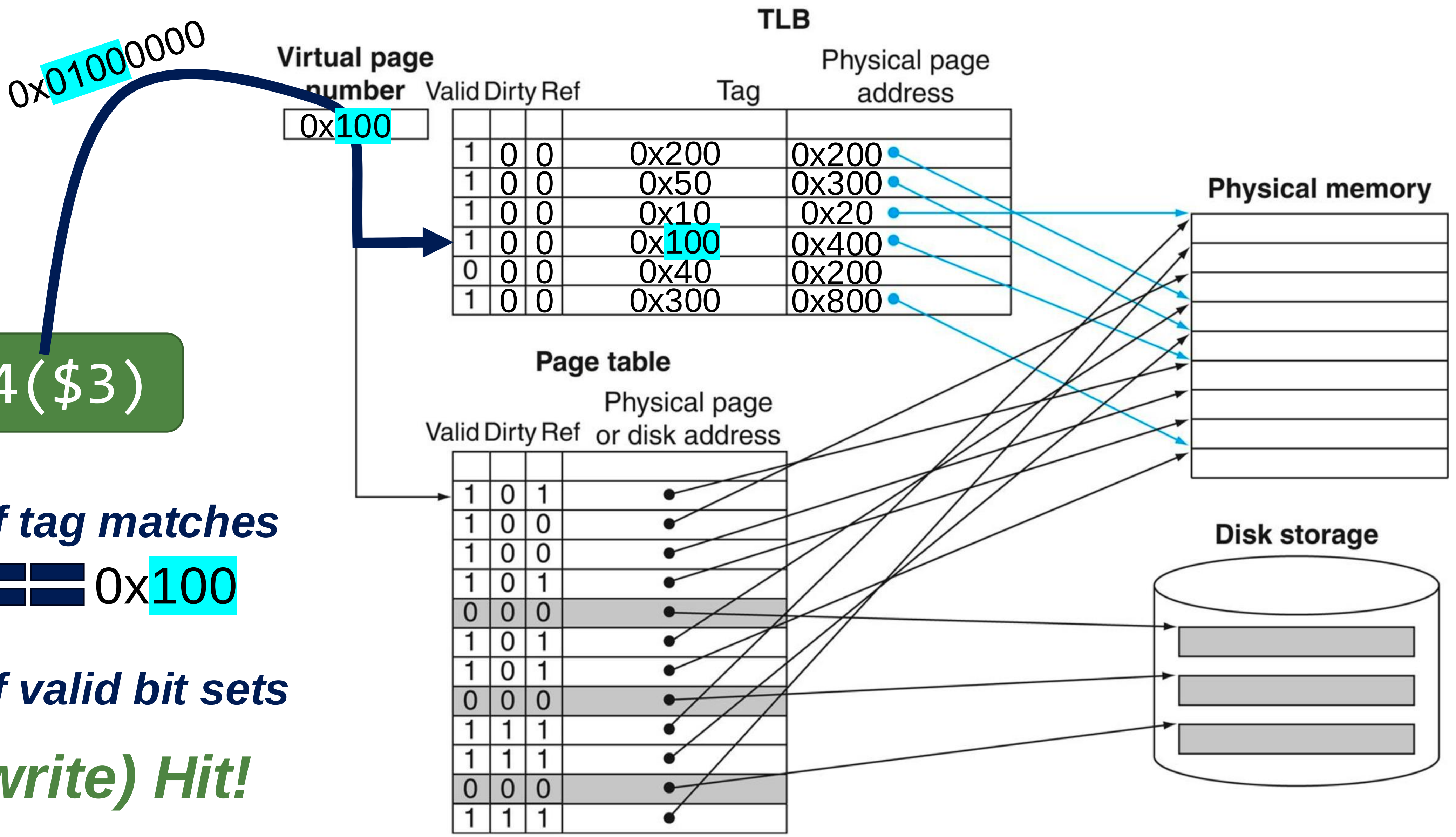
TLB

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

Page table

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

Case #1: TLB Hits



TLB

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

Page table

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

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

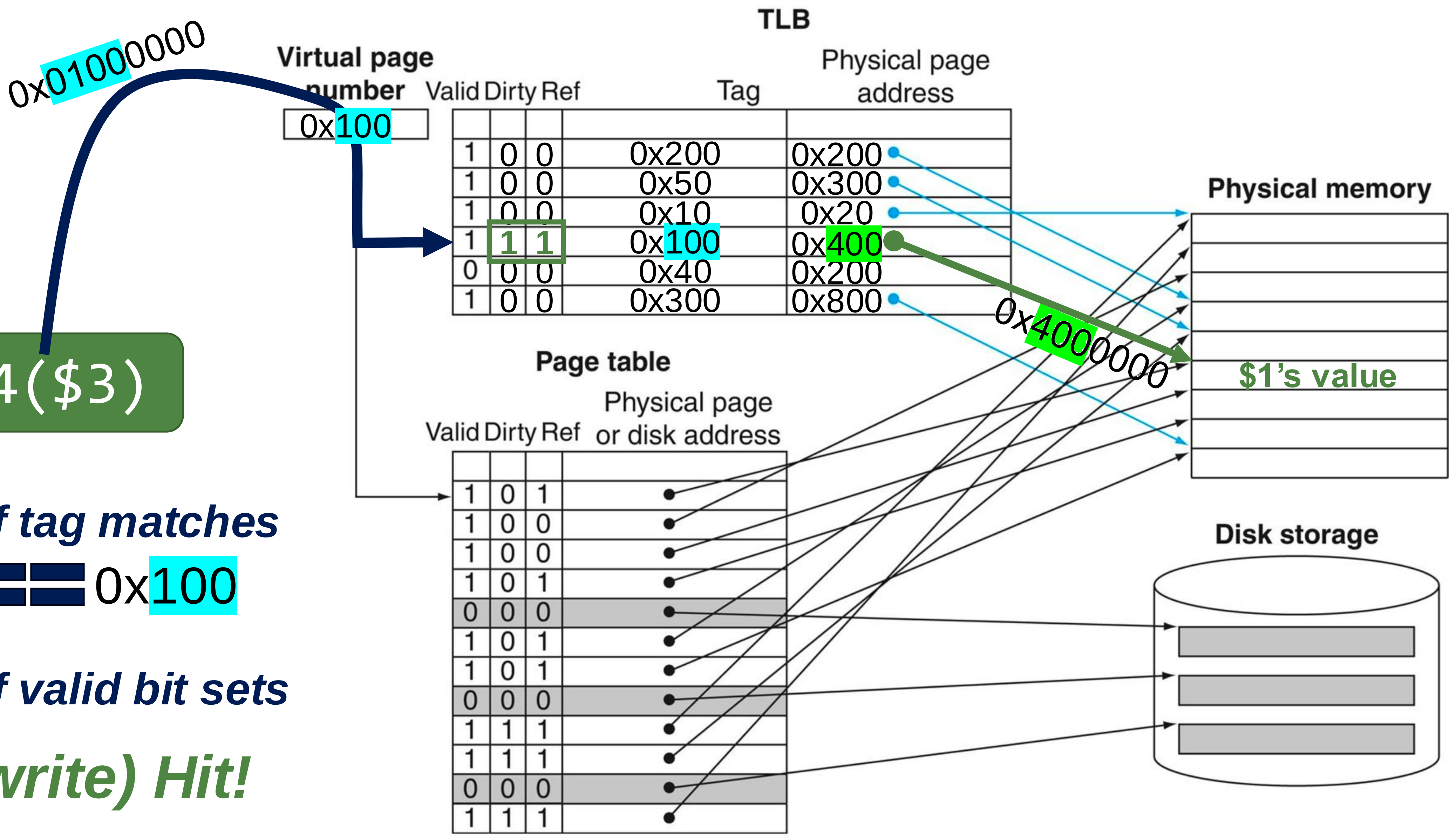
(1) Check if tag matches

0x100 == 0x100

(2) Check if valid bit sets

TBL (write) Hit!

Case #1: TLB Hits



TLB

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

Page table

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

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

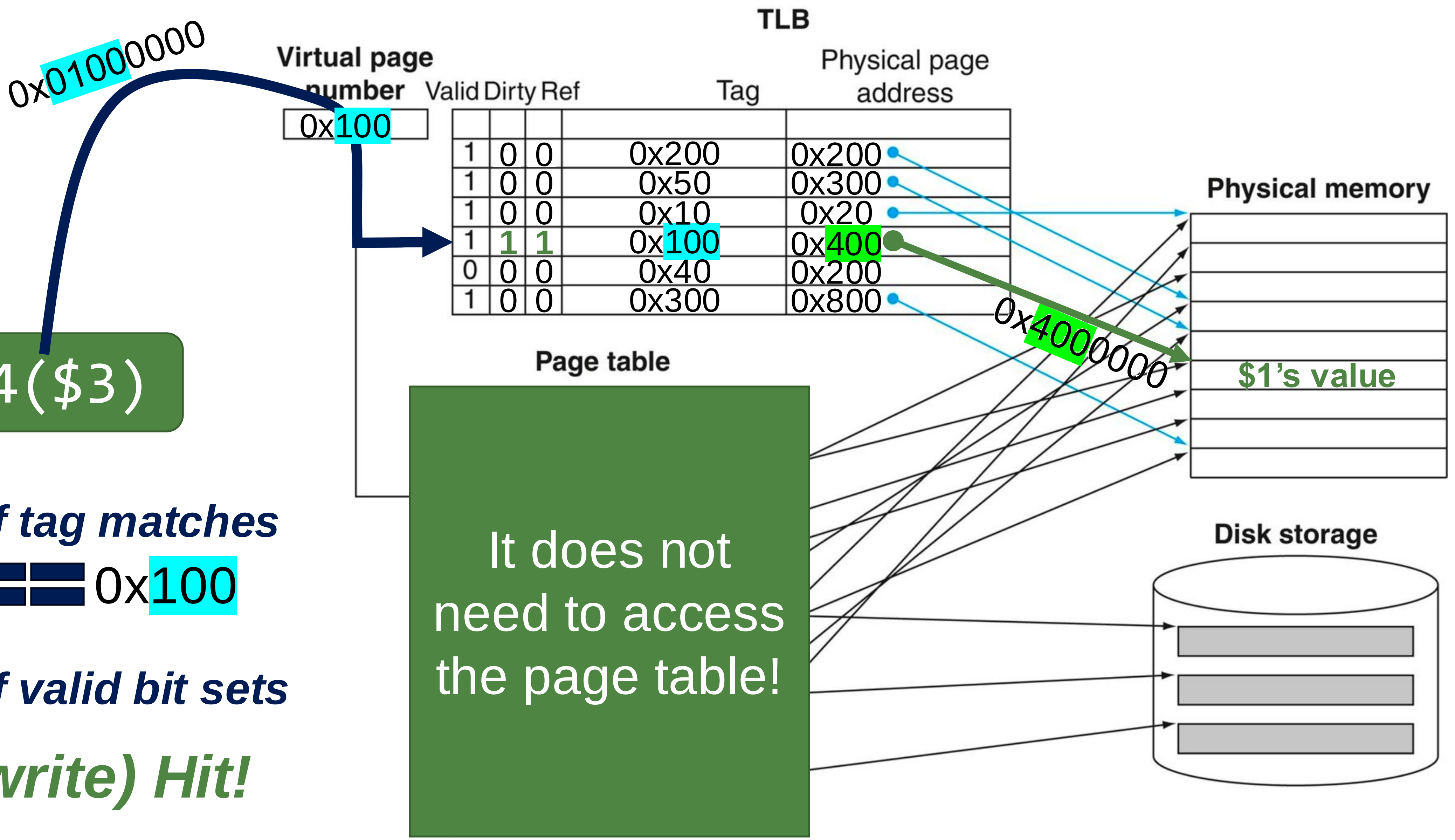
(1) Check if tag matches

0x100 == 0x100

(2) Check if valid bit sets

TBL (write) Hit!

Case #1: TLB Hits



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

It does not need to access the page table!

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

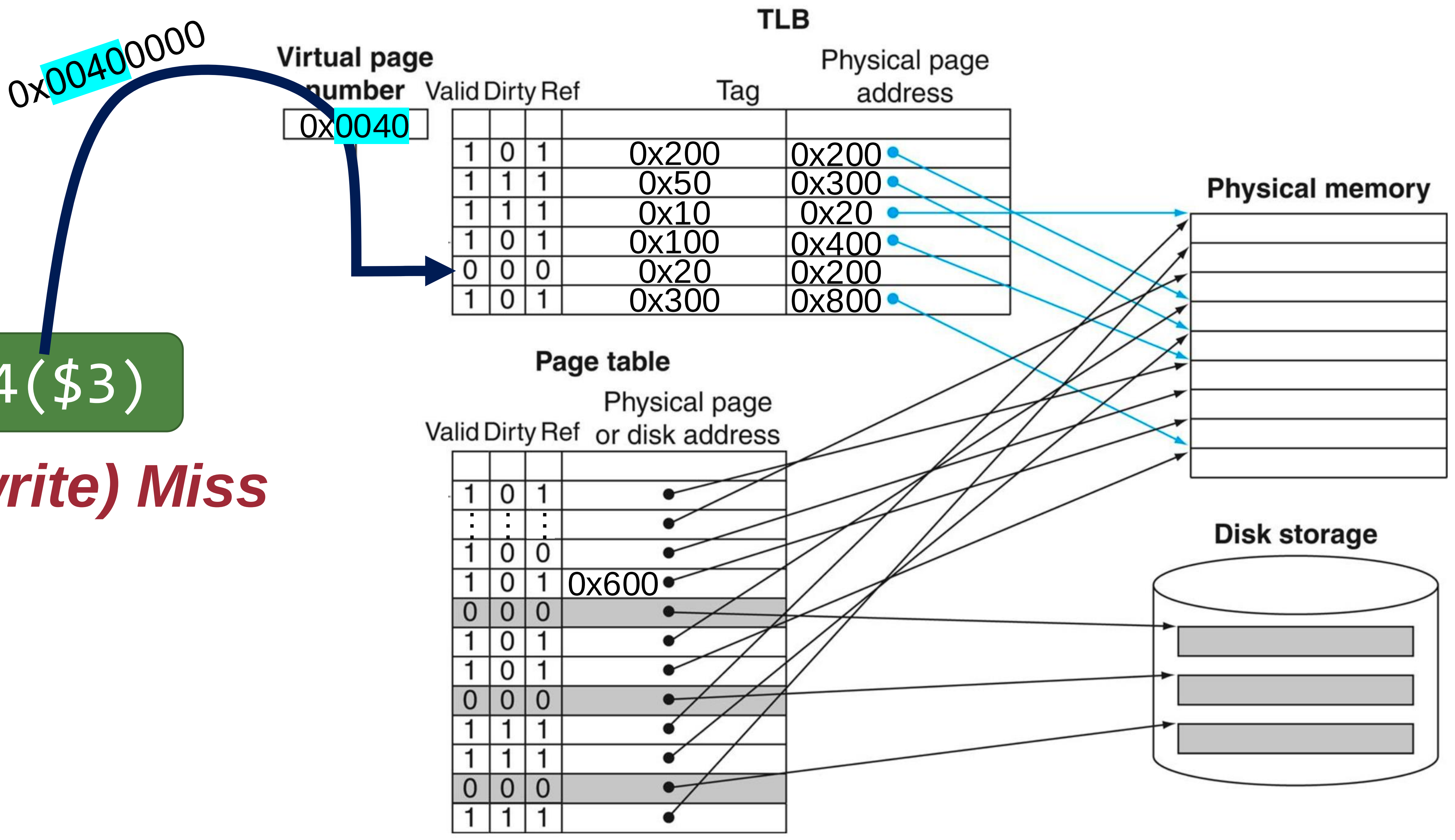
(1) Check if tag matches

0x100 == 0x100

(2) Check if valid bit sets

TBL (write) Hit!

Case #2: TLB Misses



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

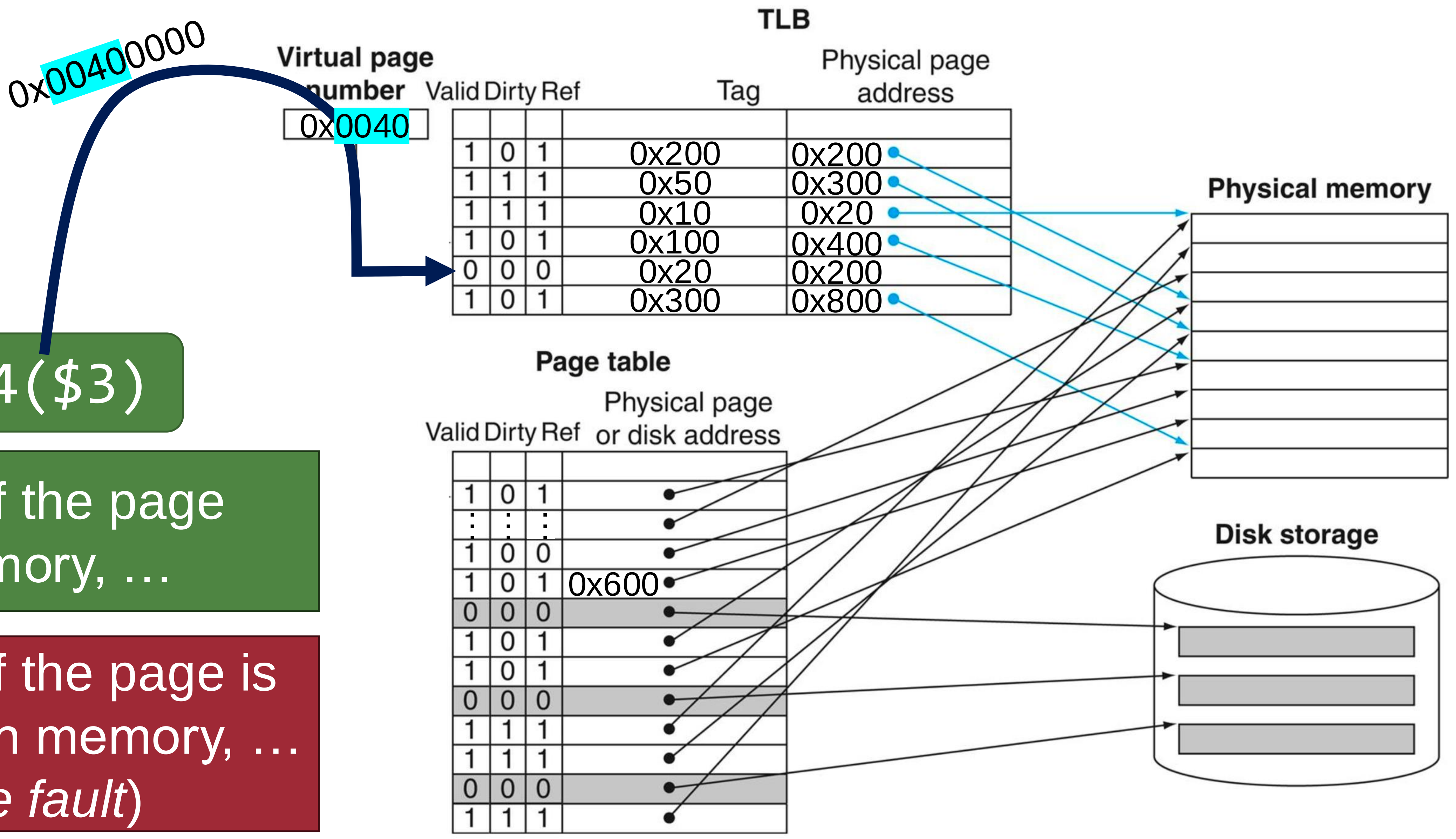
Page table

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	●

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

TBL (write) Miss

Case #2: TLB Misses



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

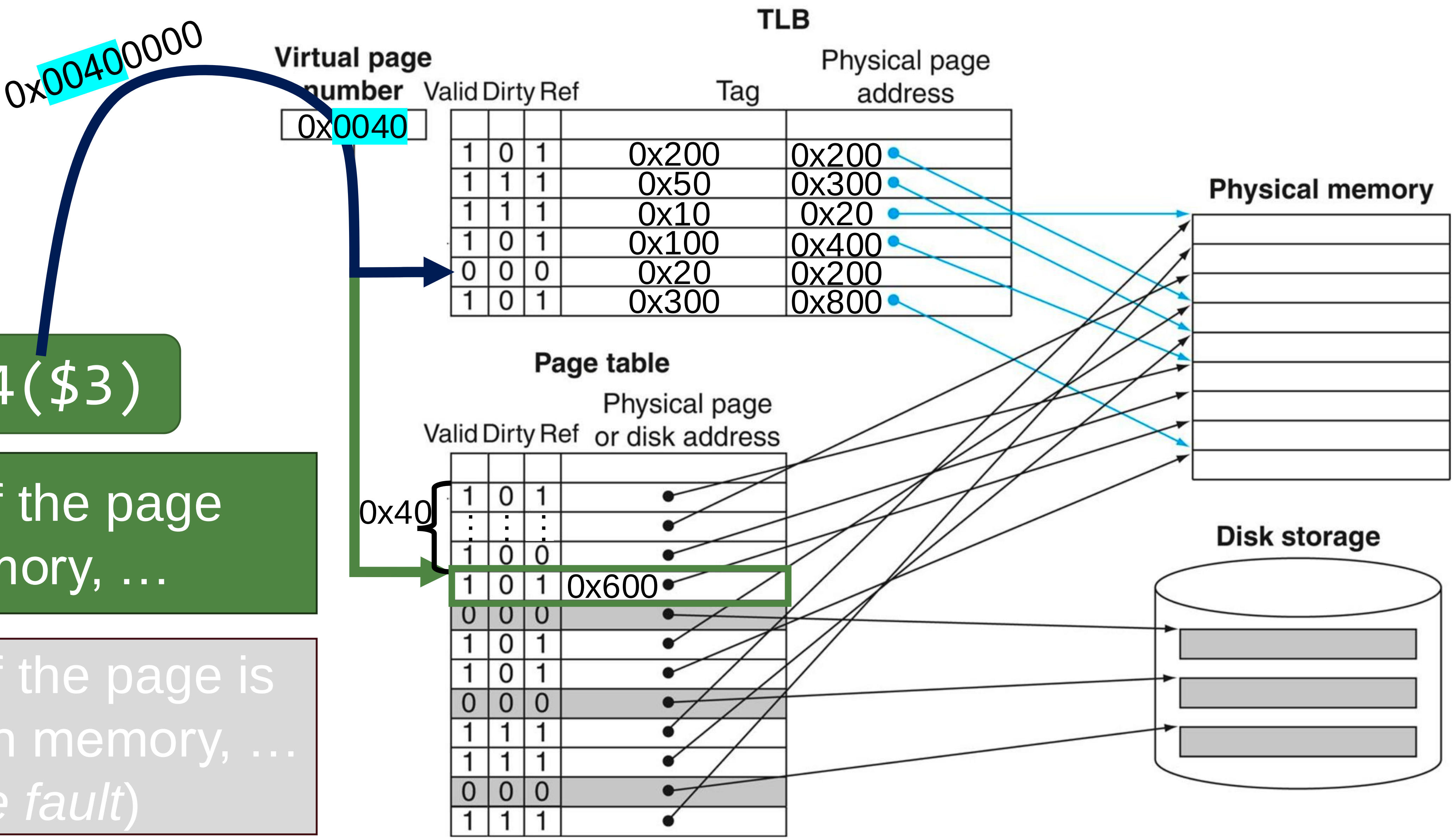
Valid	Dirty	Ref	Physical page or disk address
1	0	1	
:	:	:	
1	0	0	
1	0	1	0x600
0	0	0	
1	0	1	
1	0	1	
0	0	0	
1	1	1	
1	1	1	
0	0	0	
1	1	1	

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

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

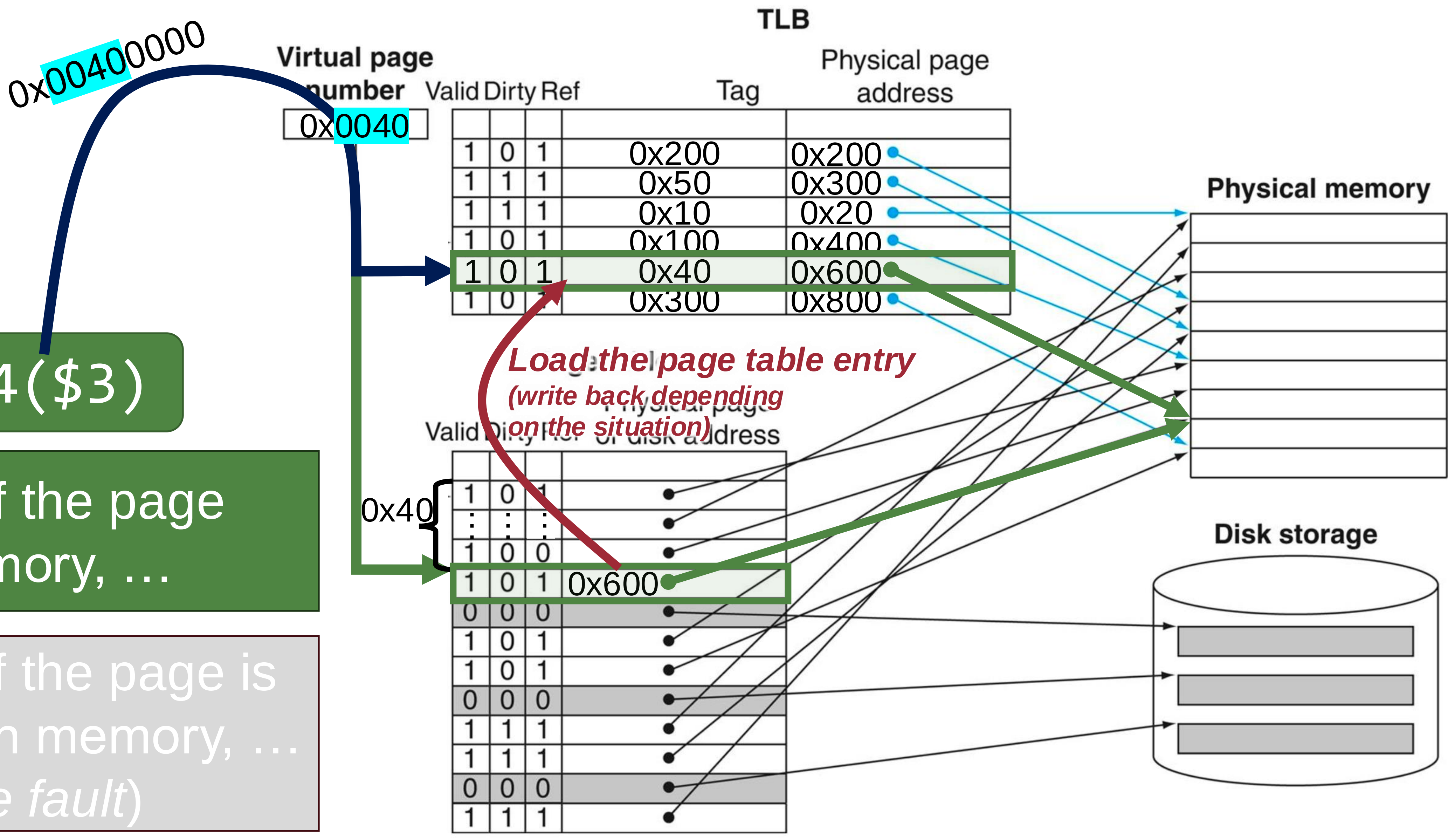


```
sw $1, 4($3)
```

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
TLB
Physical memory
Disk storage
Load the page table entry
(write back depending on the situation)
Physical page or disk address
0x40

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

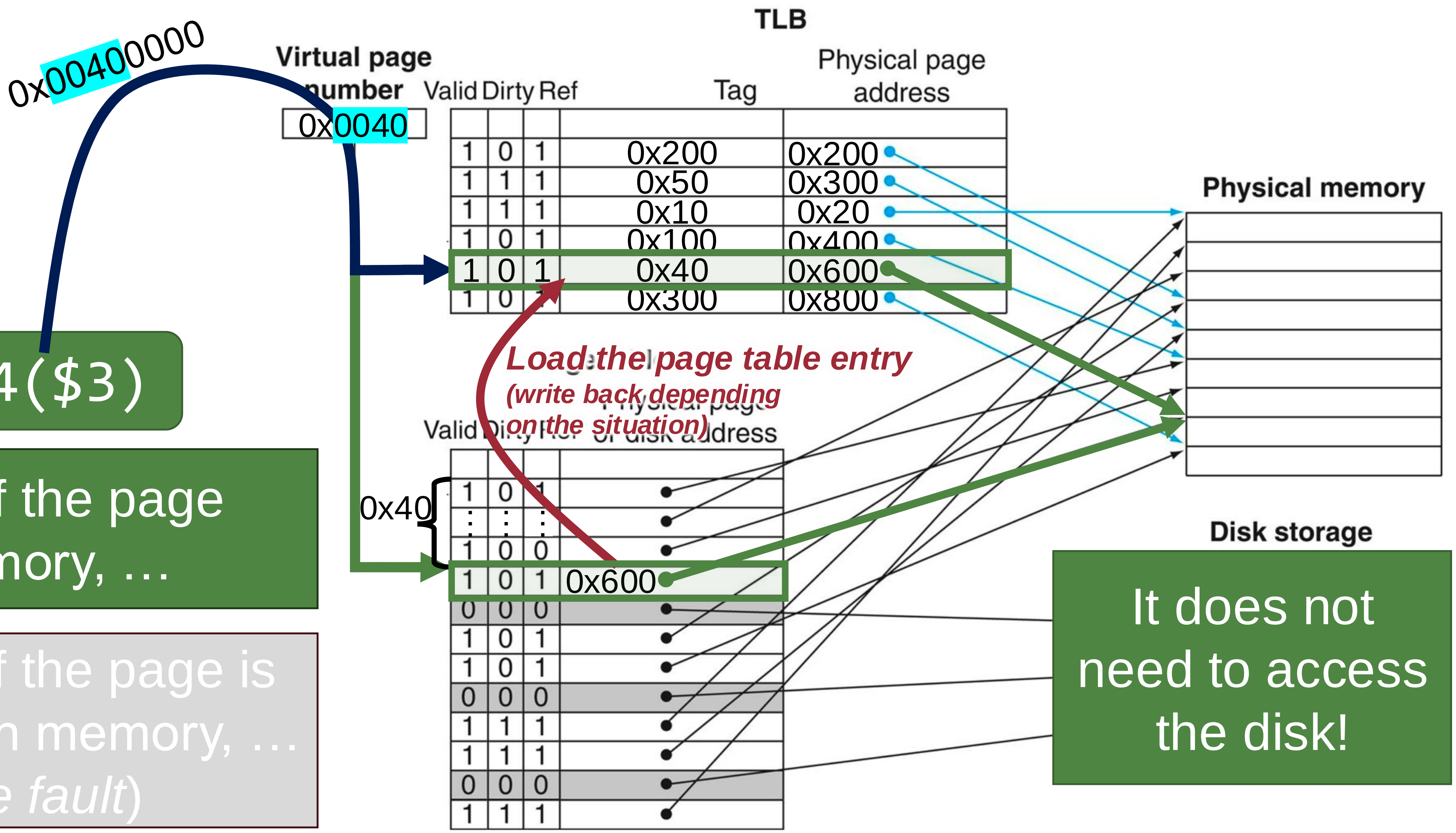
Valid	Dirty	Ref	Physical page or disk address
1	0	1	
:	:	:	
1	0	0	
1	0	1	0x600
0	0	0	
1	0	1	
1	0	1	
0	0	0	
1	1	1	
1	1	1	
0	0	0	
1	1	1	

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

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



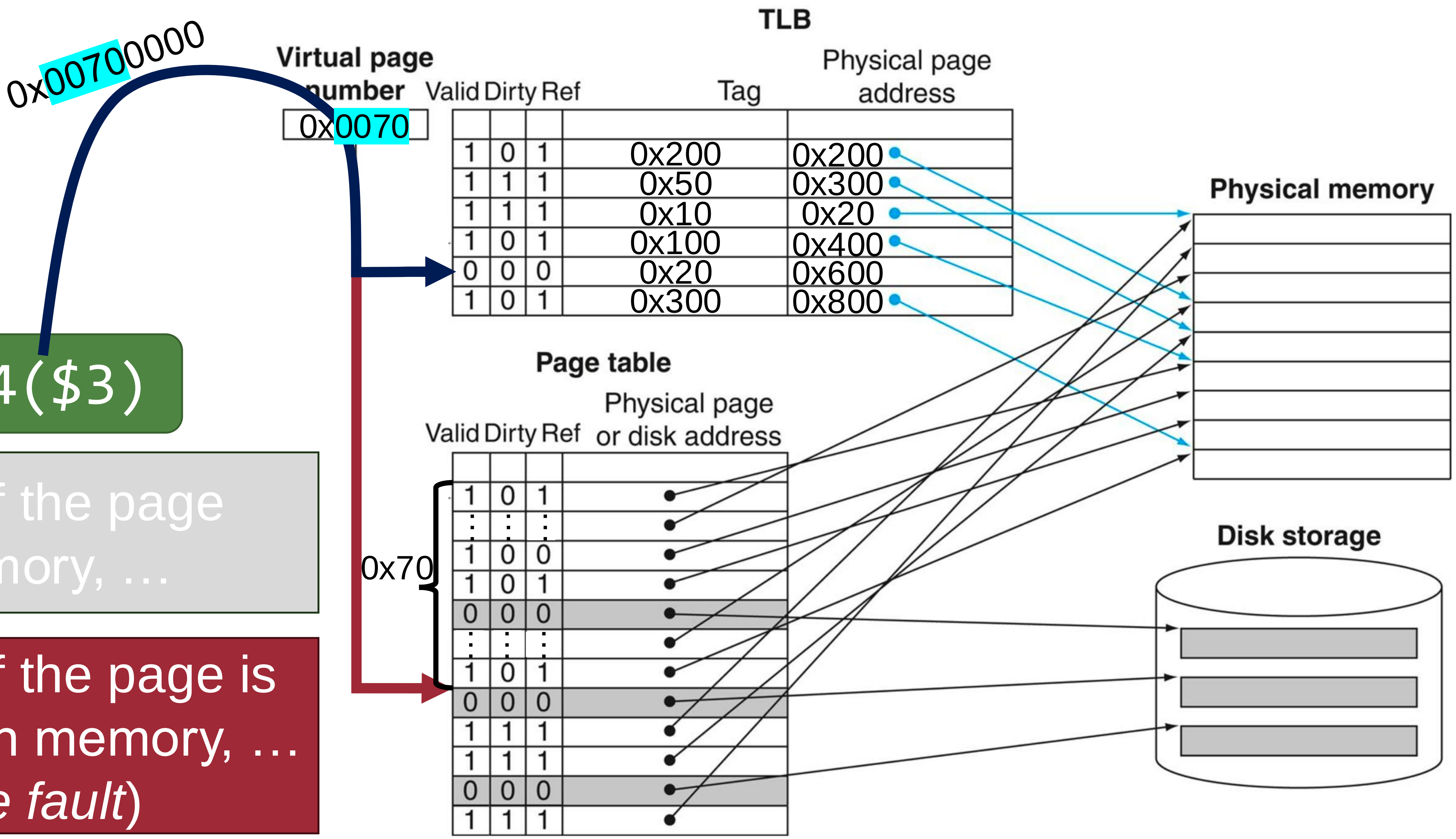
```
sw $1, 4($3)
```

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

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

It does not need to access the disk!

Case #2-2: TLB Misses – Page Fault



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

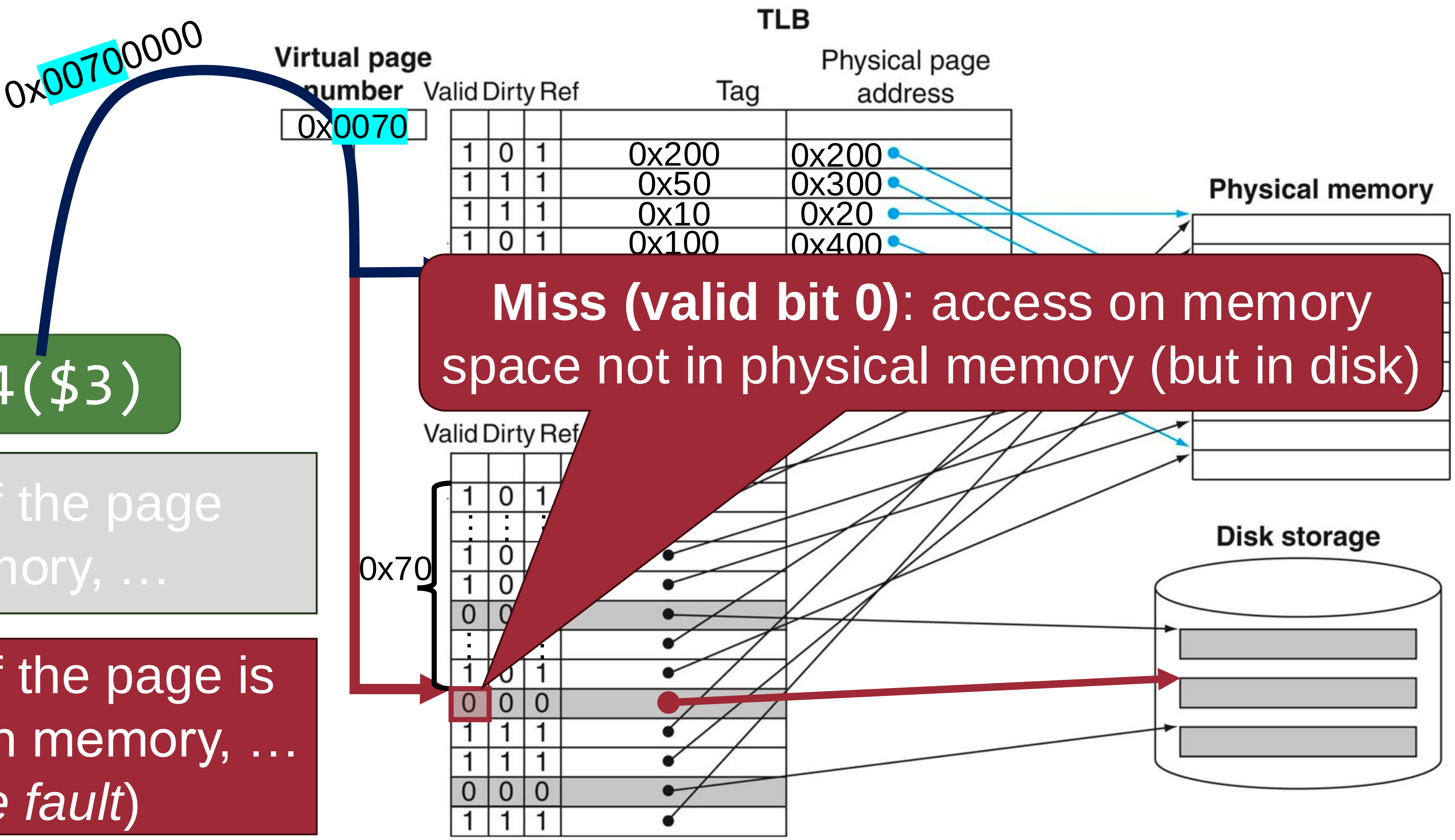
Valid	Dirty	Ref	Physical page or disk address
1	0	1	●
⋮	⋮	⋮	●
1	0	0	●
1	0	1	●
0	0	0	●
⋮	⋮	⋮	●
1	0	1	●
0	0	0	●
1	1	1	●
1	1	1	●
0	0	0	●
1	1	1	●

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

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



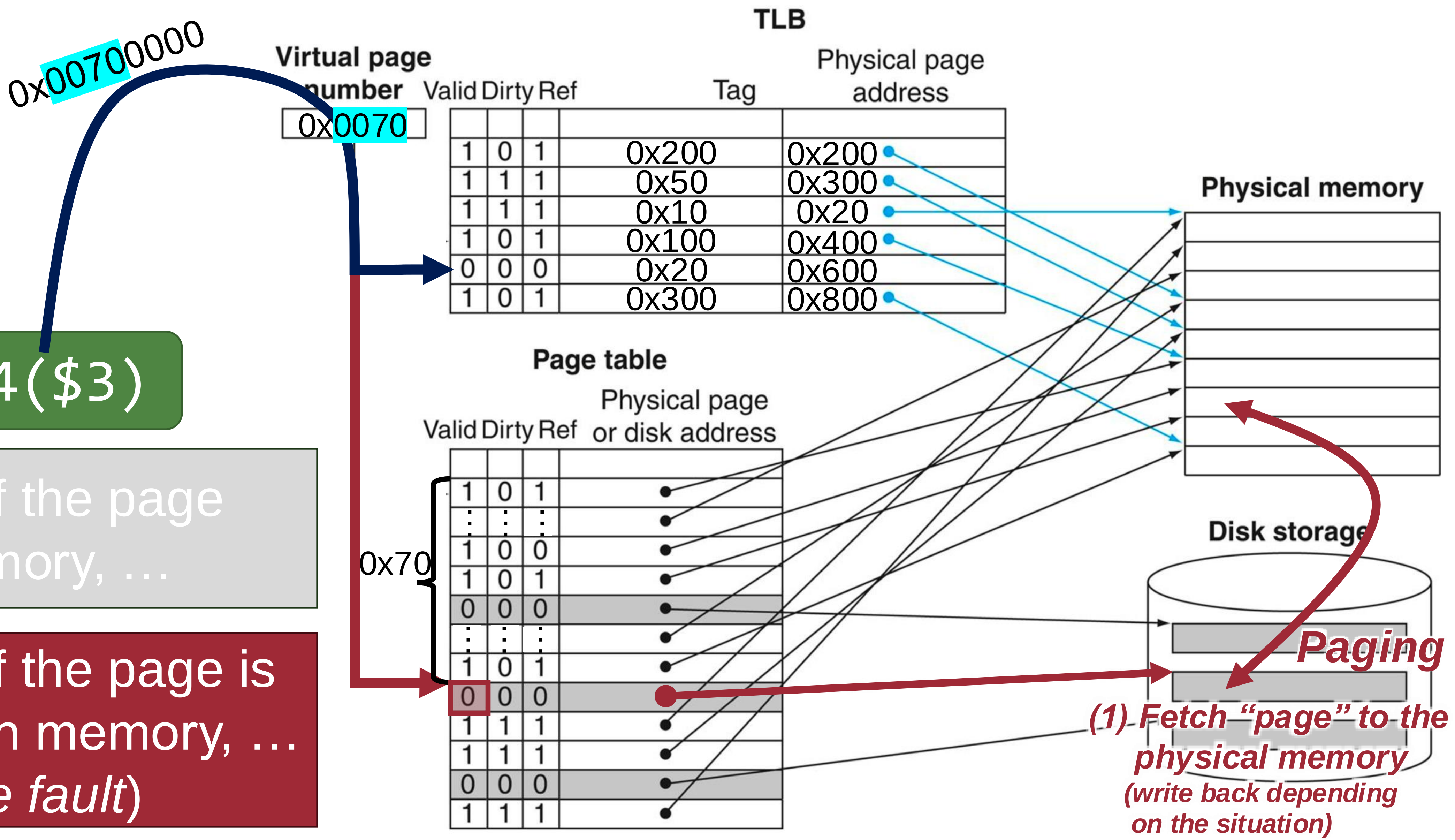
Miss (valid bit 0): access on memory space not in physical memory (but in disk)

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

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
TLB
0x70
Physical memory
Disk storage
Paging
(1) Fetch "page" to the physical memory
(write back depending on the situation)

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

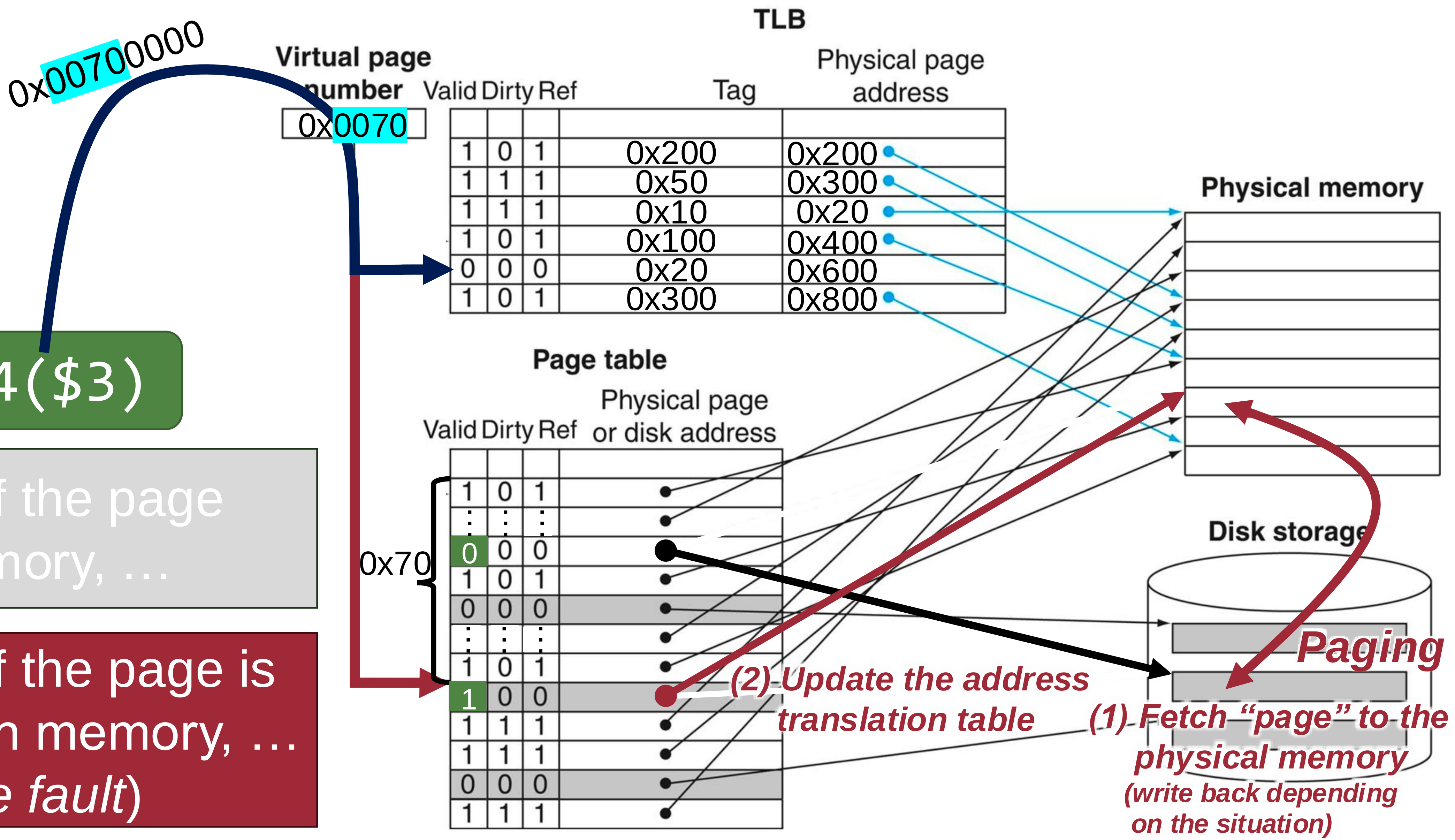
Valid	Dirty	Ref	Physical page or disk address
1	0	1	●
⋮	⋮	⋮	●
1	0	0	●
1	0	1	●
0	0	0	●
⋮	⋮	⋮	●
1	0	1	●
0	0	0	●
1	1	1	●
1	1	1	●
0	0	0	●
1	1	1	●

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

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
TLB
Physical memory
Page table
0x70
Disk storage
Paging
(2) Update the address translation table
(1) Fetch "page" to the physical memory
(write back depending on the situation)

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

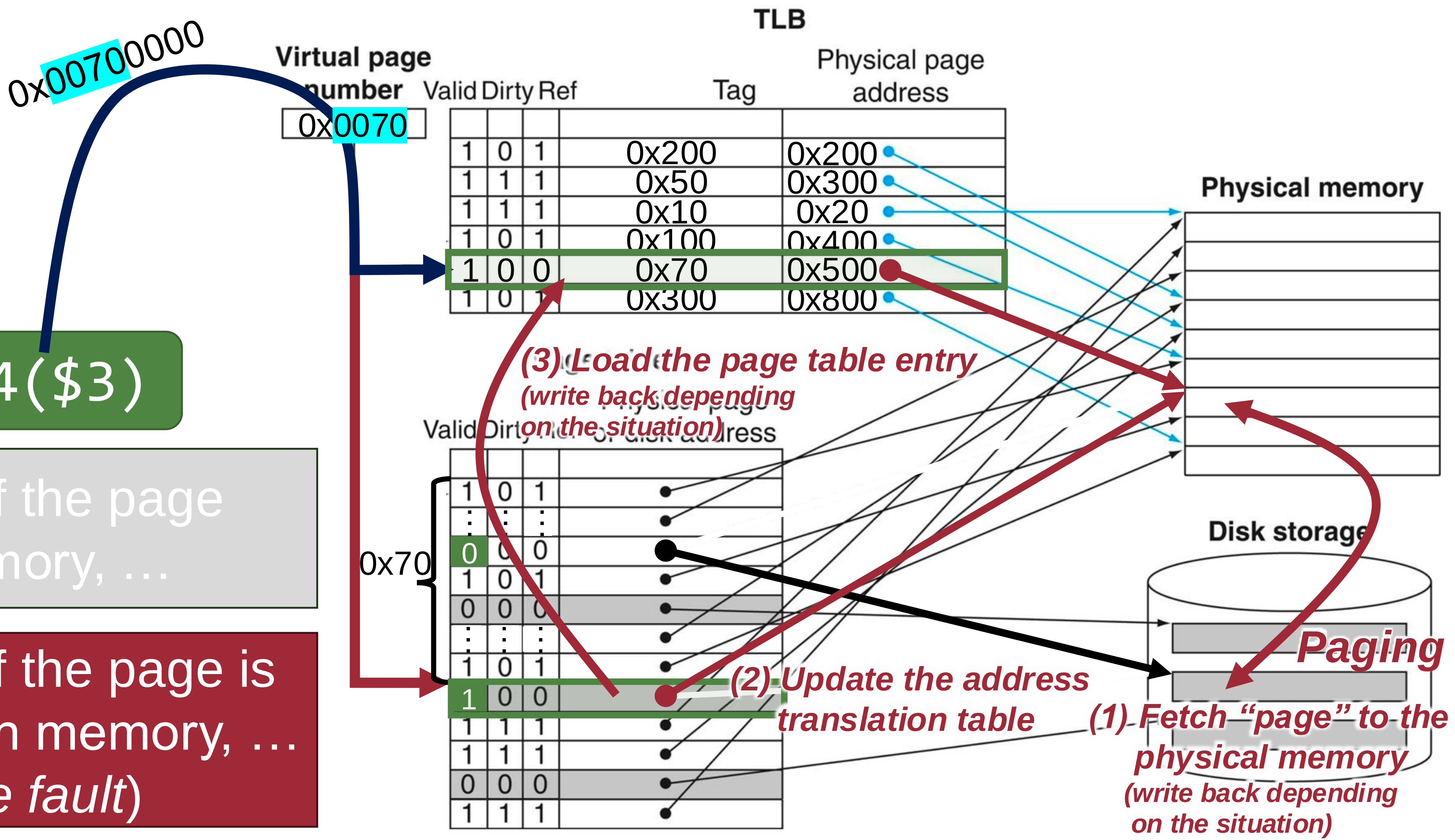
Valid	Dirty	Ref	Physical page or disk address
1	0	1	●
⋮	⋮	⋮	●
0	0	0	●
1	0	1	●
0	0	0	●
⋮	⋮	⋮	●
1	0	1	●
1	0	0	●
1	1	1	●
1	1	1	●
0	0	0	●
1	1	1	●

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

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



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

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

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

Case #2: TLB Misses

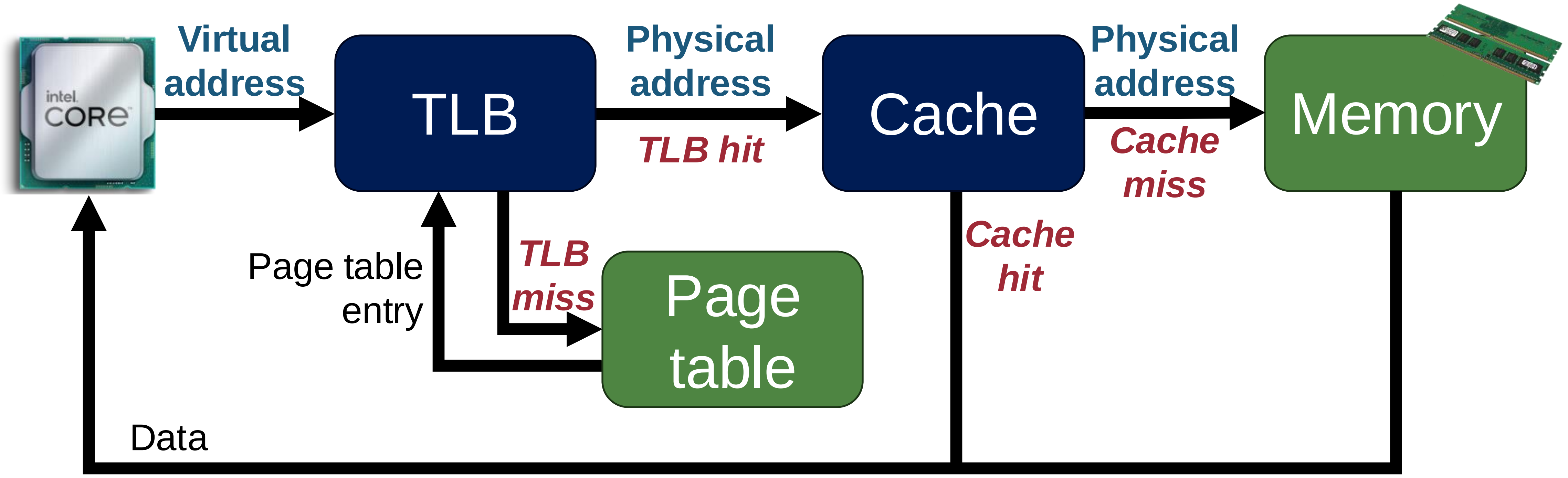
Case #2-1: If page is in memory

- 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)

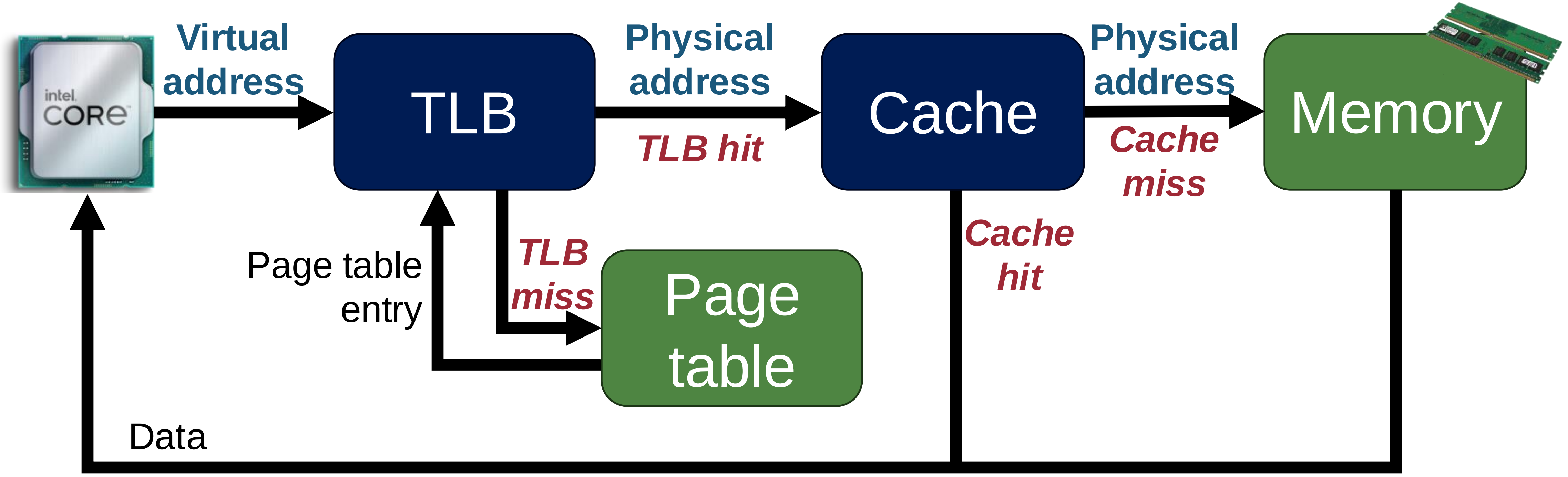
- OS handles fetching the page and updating the page table
- Then restart the faulting instruction

TLB and Cache Interaction

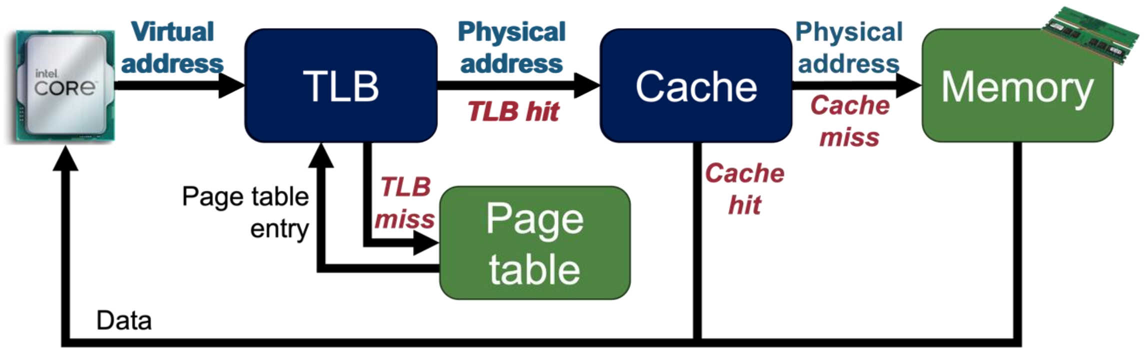


TLB and Cache Interaction

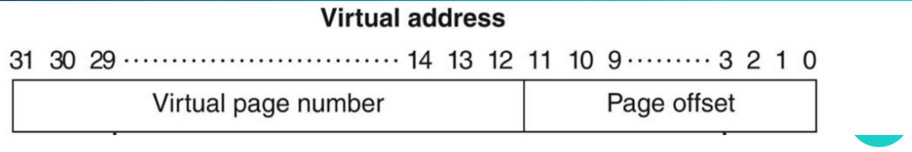
- Physically addressed caches
  - Cache tag uses physical address
    - Always translate before cache lookup
  - Allows multiple processes to have blocks in cache at the same time
  - Allows multiple processes to share pages

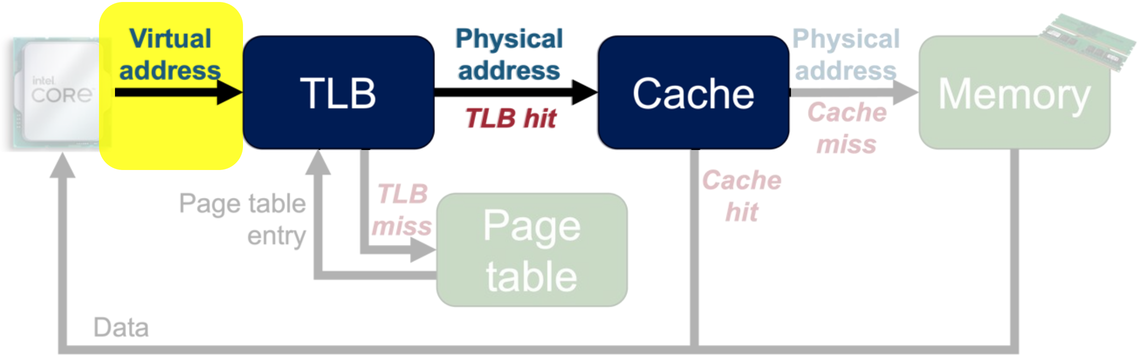


TLB and Cache

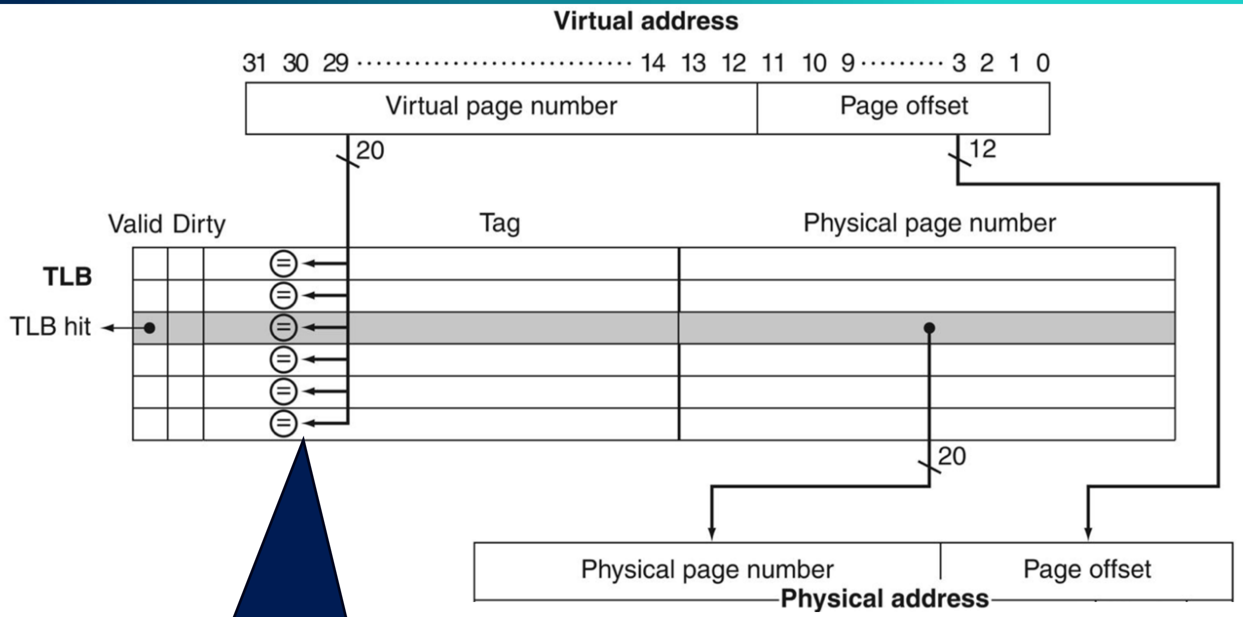


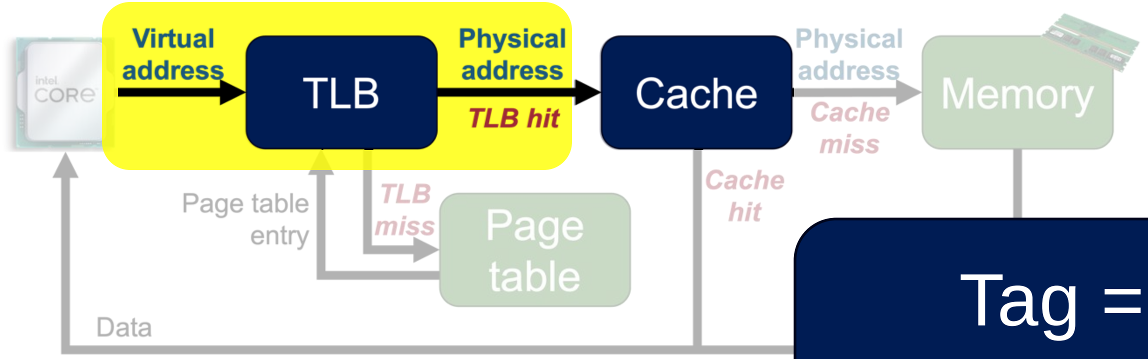
TLB and Cache





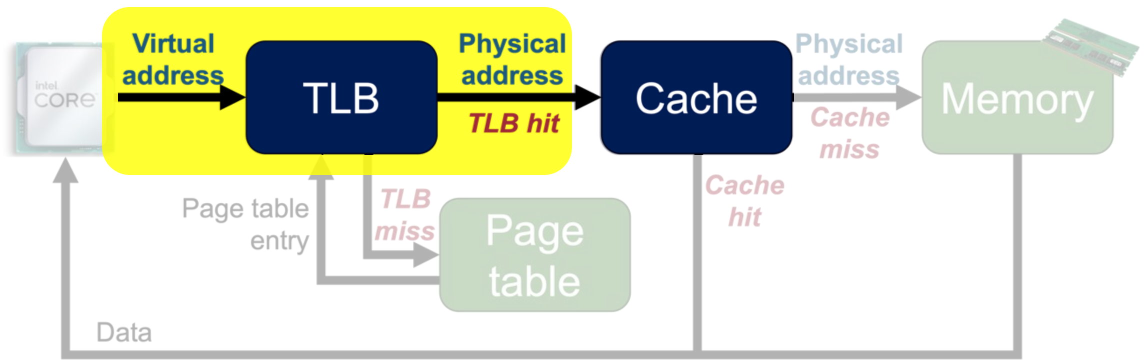
TLB and Cache

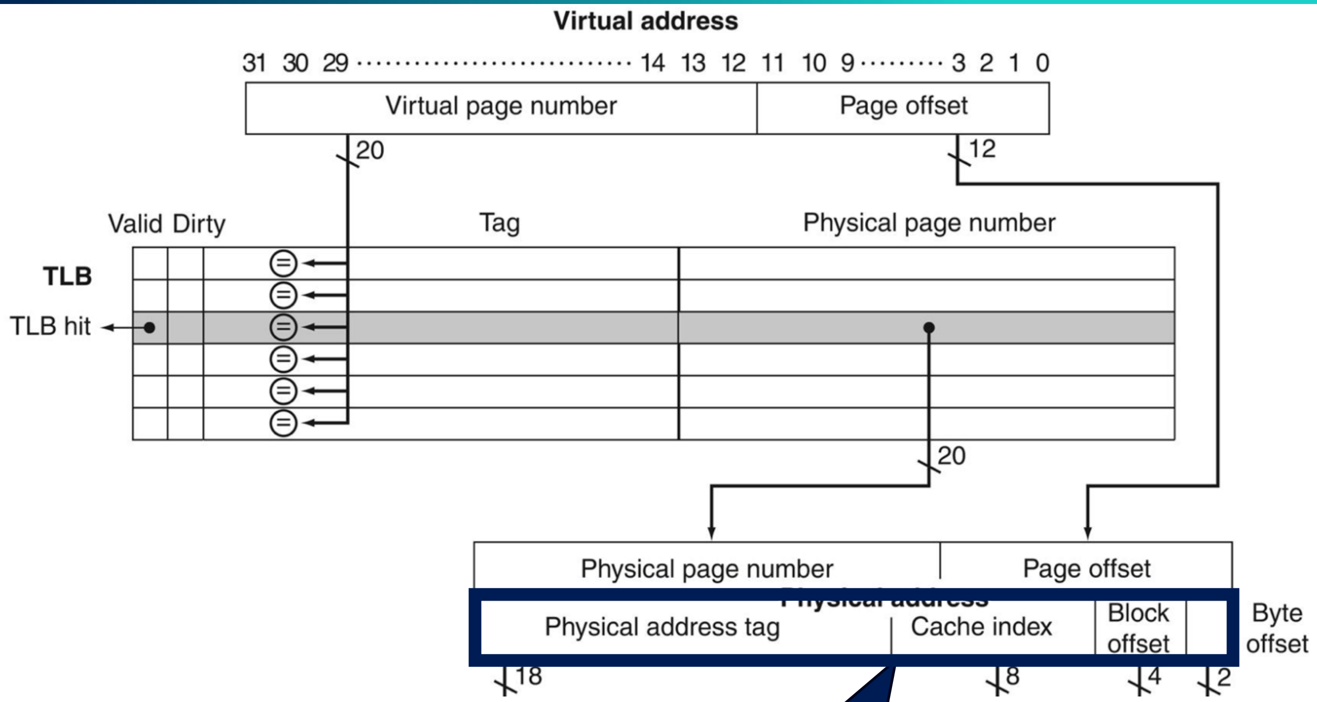




Tag = Virtual page #
(→ fully-associative)

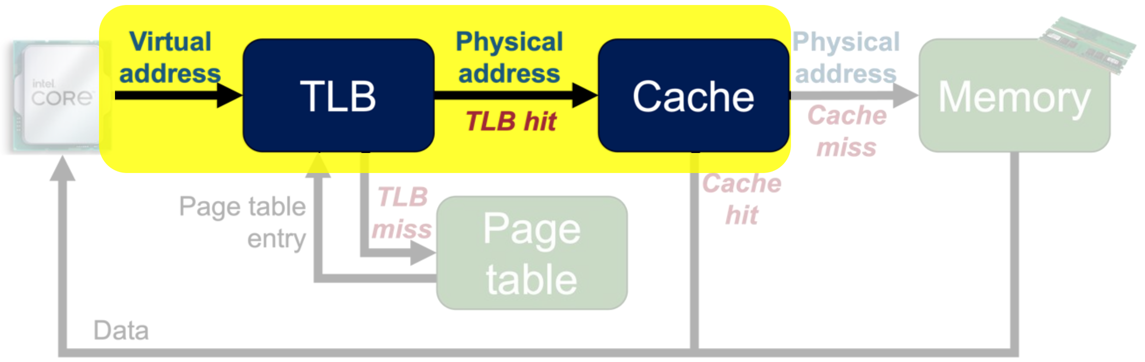
TLB and Cache

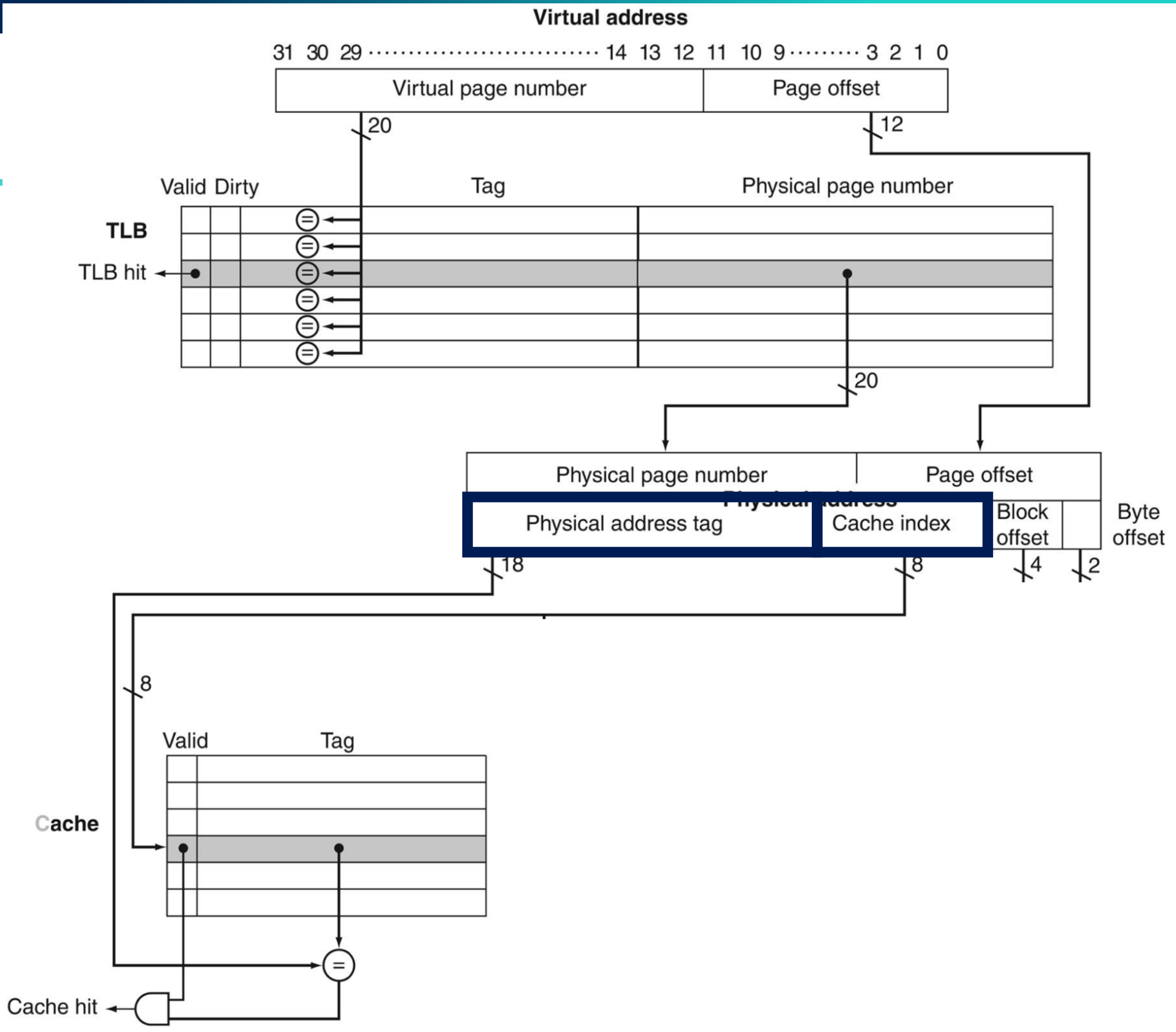




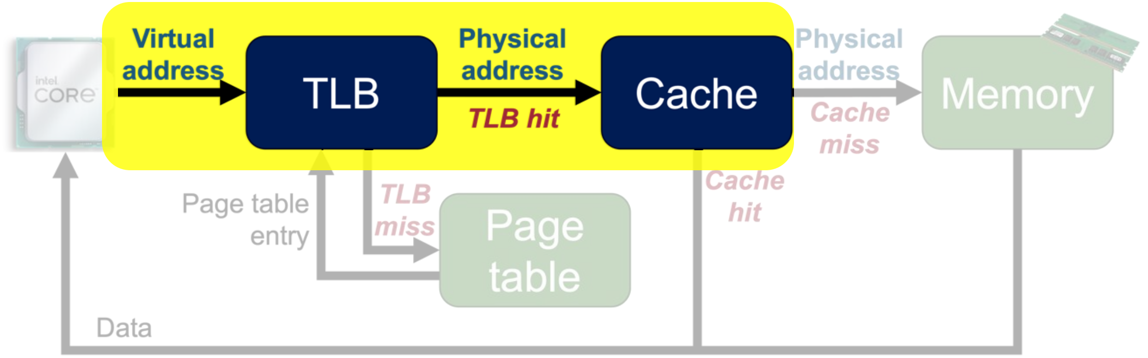
Fields for caching

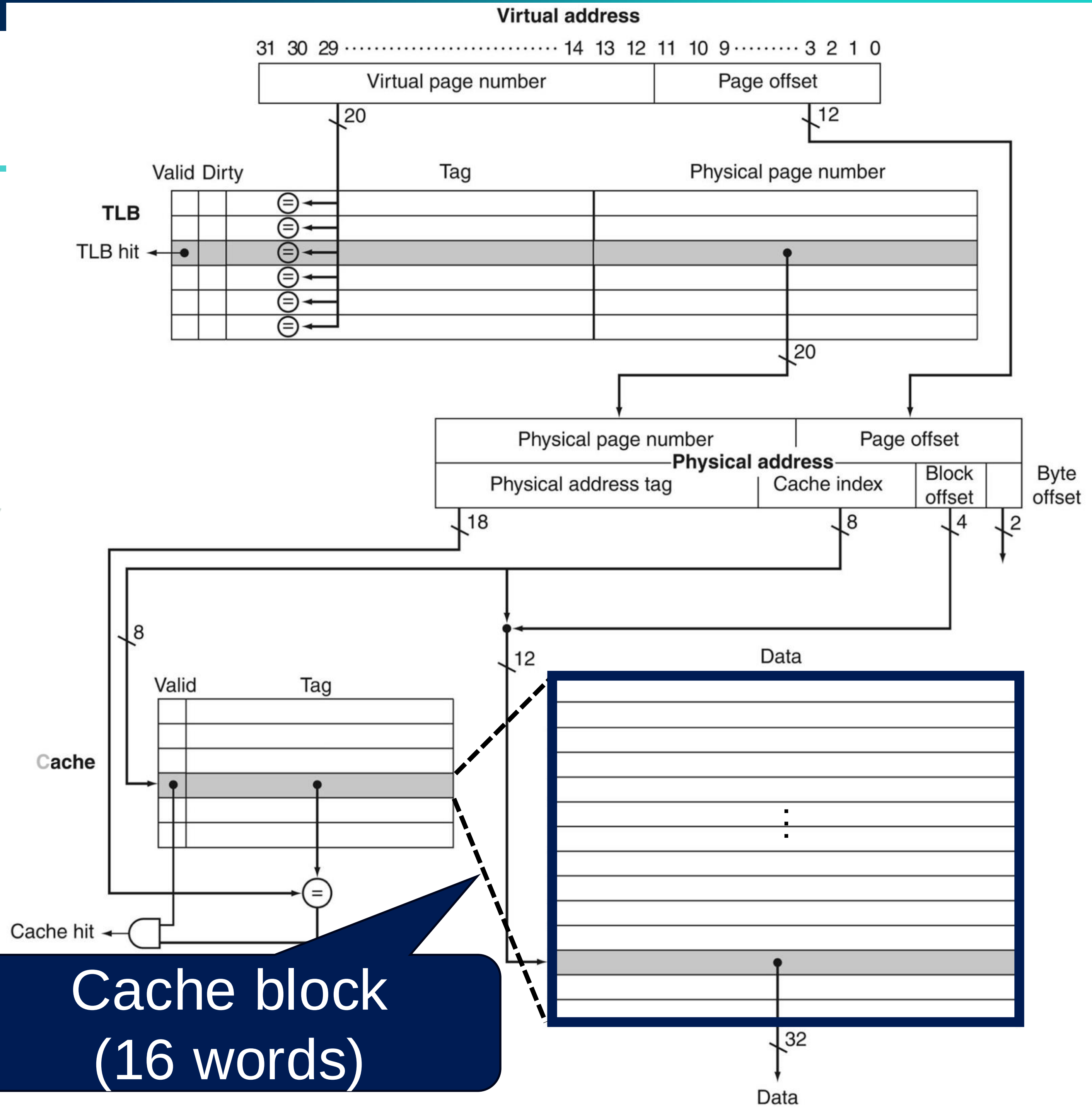
TLB and Cache



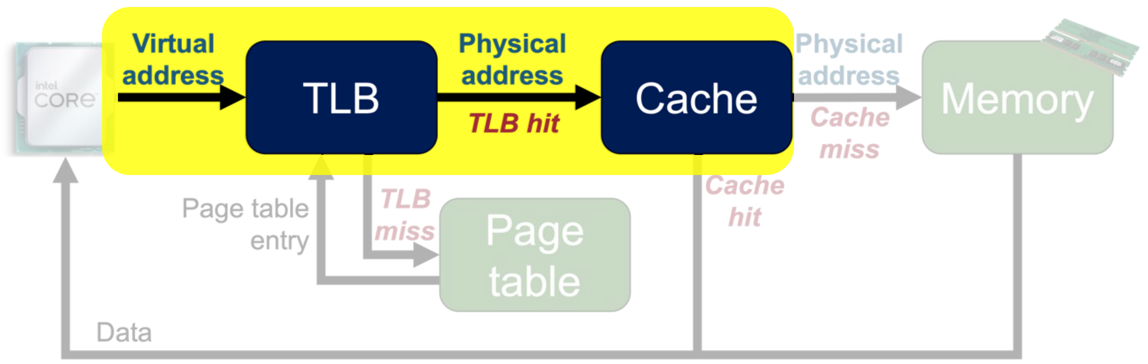


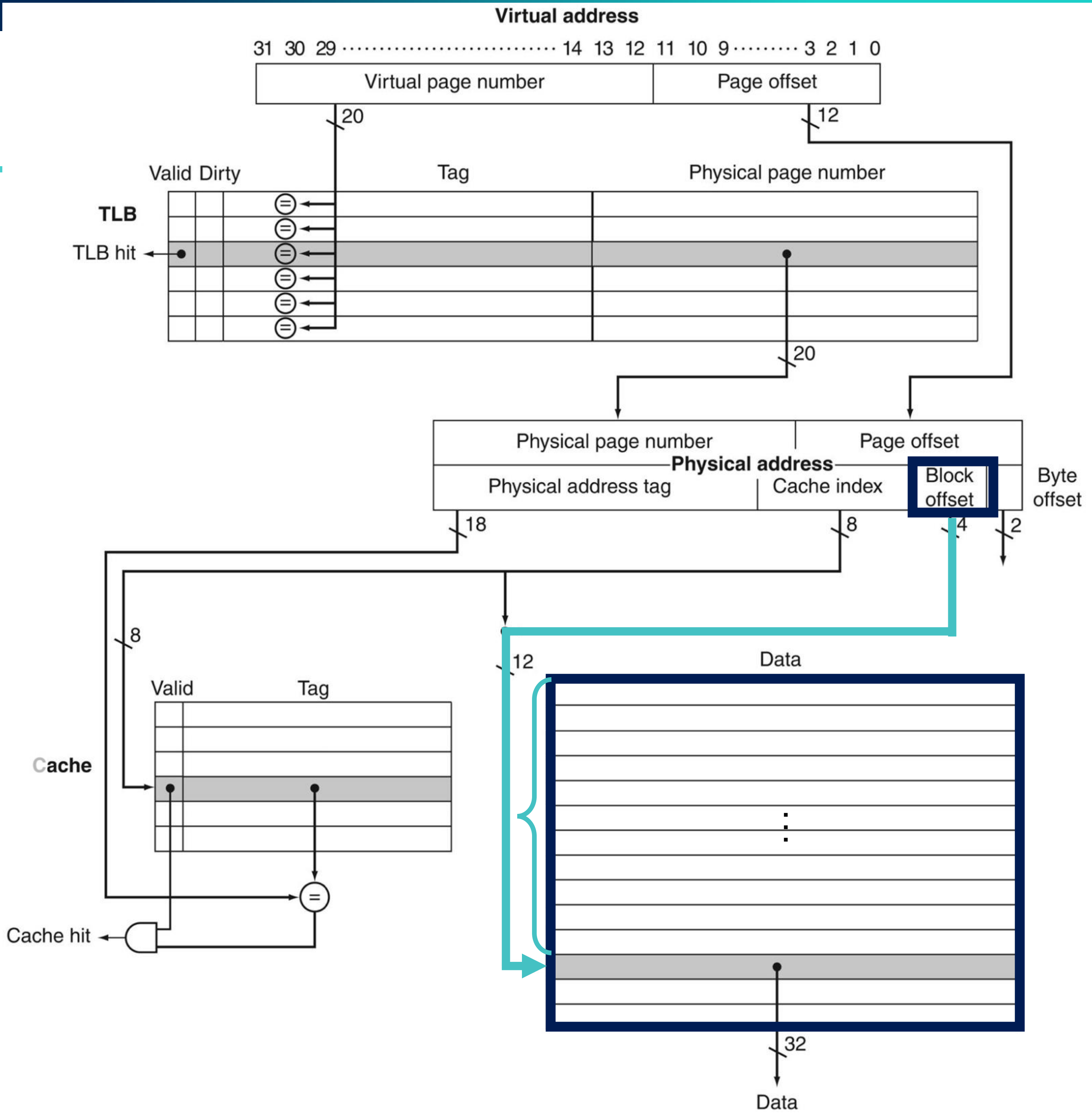
TLB and Cache



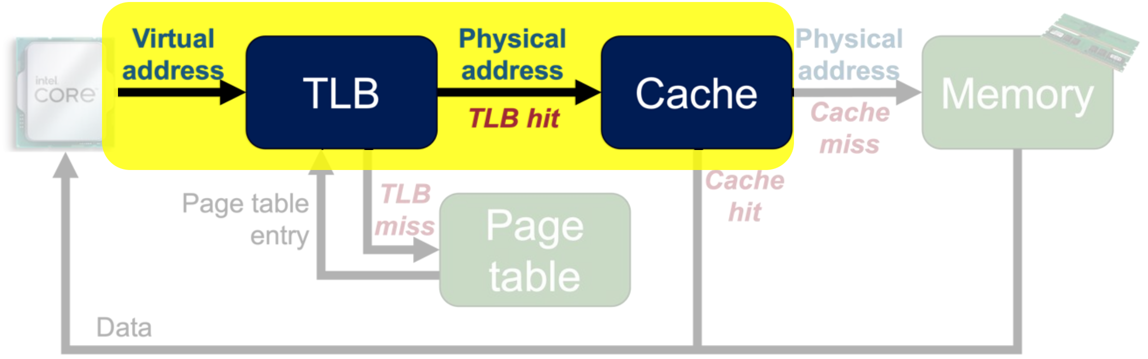


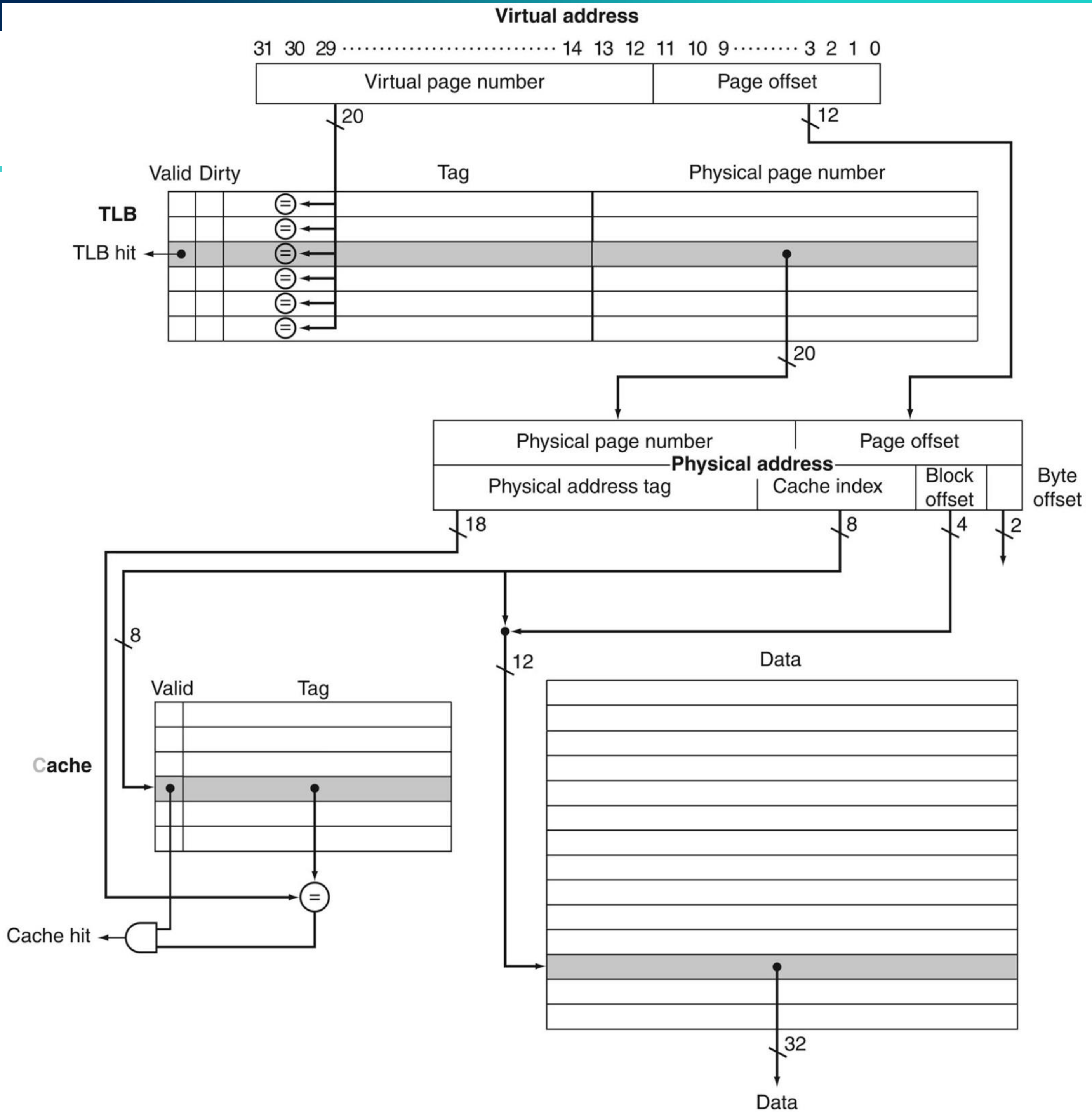
TLB and Cache





TLB and Cache





Question?