Caches III

CSE 351 Spring 2022

Instructor:

Ruth Anderson

Teaching Assistants:

Melissa Birchfield

Jacob Christy

Alena Dickmann

Kyrie Dowling

Ellis Haker

Maggie Jiang

Diya Joy

Anirudh Kumar

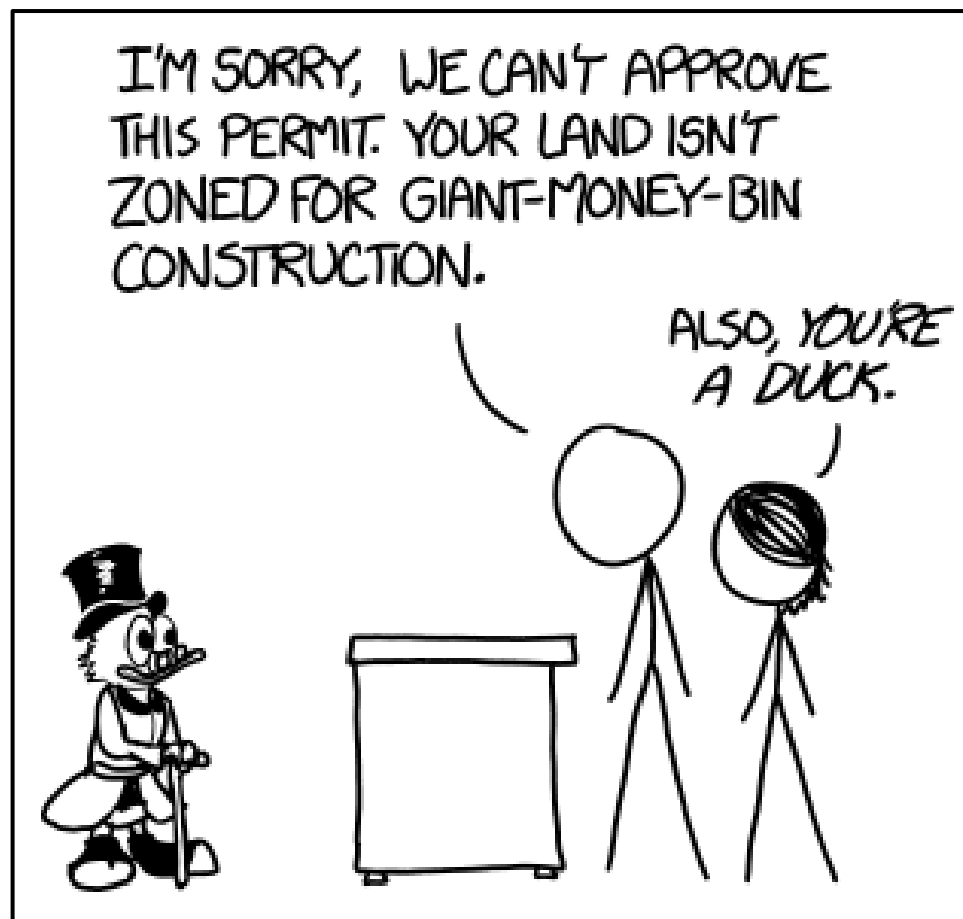Jim Limprasert

Armin Magness

Hamsa Shankar

Dara Stotland

Jeffery Tian

Assaf Vayner

Tom Wu

Angela Xu

Effie Zheng

https://what-if.xkcd.com/111/

Relevant Course Information

❖ hw15 due Friday (5/06)

❖ Mid-quarter Survey due Saturday (5/07)

❖ hw16 due Monday (5/09)

❖ Lab 3 due Wednesday (5/11)

- You will have everything you need for this now!

- Some discussion in section this week

- Last part of hw15 (due Fri 5/06) is useful for Lab 3

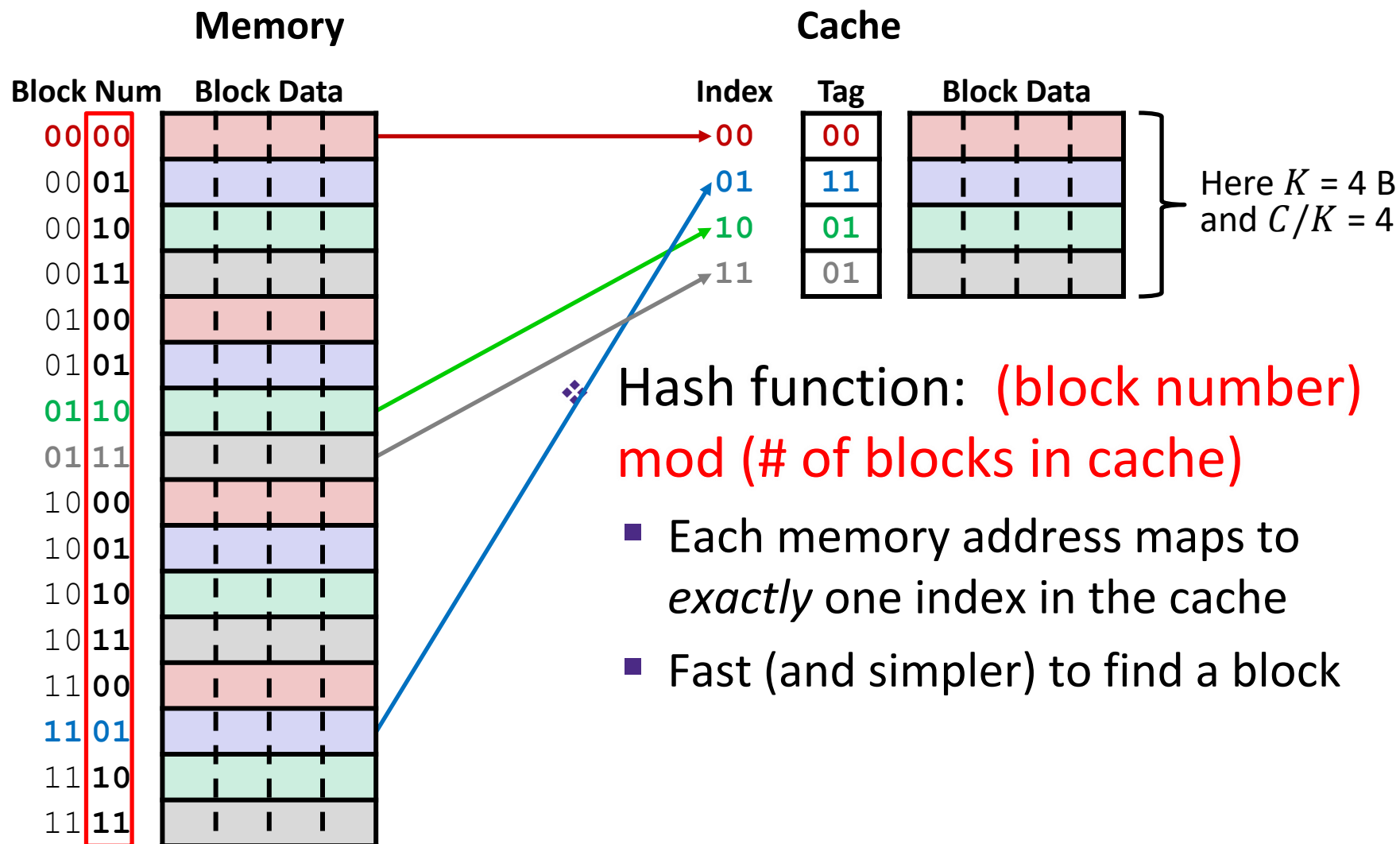❖ hw17 due *next* Friday (5/13)

- Don't wait too long, this is a BIG hw

Making memory accesses fast!

❖ Cache basics

❖ Principle of locality

❖ Memory hierarchies

❖ Cache organization
  ▪ Direct-mapped (*sets*; index + tag)
  ▪ **Associativity (*ways*)**
  ▪ **Replacement policy**
  ▪ Handling writes
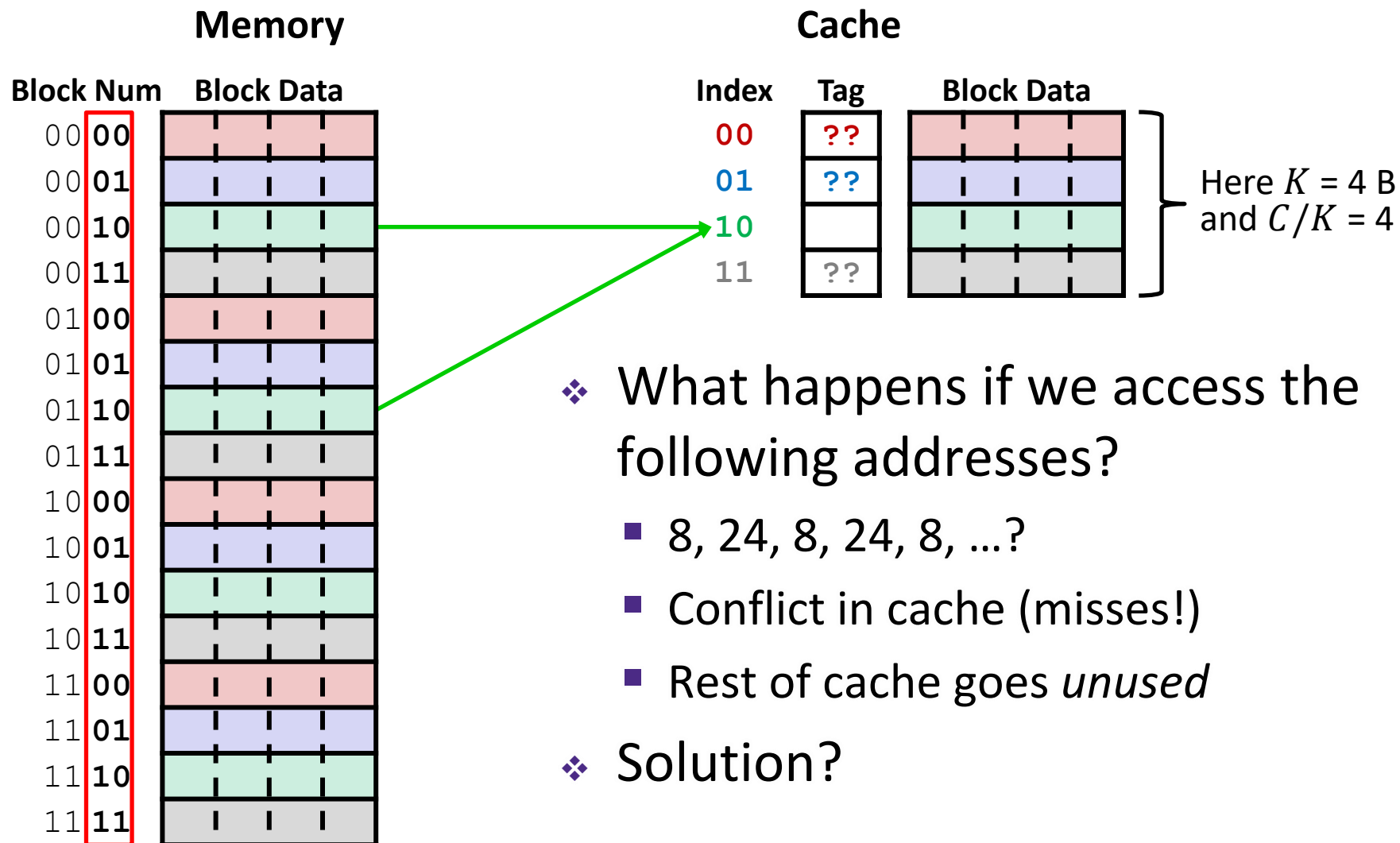
❖ Program optimizations that consider caches

Reading Review

❖ Terminology:
- Associativity: sets, fully-associative cache
- Replacement policies: least recently used (LRU)
- Cache line: cache block + management bits (valid, tag)
- Cache misses: compulsory, conflict, capacity

Review: Direct-Mapped Cache

Memory

Cache

Block Num   **Block Data**

Index   **Tag**   **Block Data**

Here K = 4 B
and C/K = 4

❖ Hash function: (block number) mod (# of blocks in cache)

- Each memory address maps to *exactly* one index in the cache
- Fast (and simpler) to find a block

Direct-Mapped Cache Problem

Memory

Cache

Block Num **Block Data**

Index **Tag** **Block Data**

00 **00**	
00 **01**	
00 **10**	
00 **11**	
01 **00**	
01 **01**	
01 **10**	
01 **11**	
10 **00**	
10 **01**	
10 **10**	
10 **11**	
11 **00**	
11 **01**	
11 **10**	
11 **11**	

Index	Tag
00	??
01	??
10	
11	??

Here K = 4 B
and C/K = 4

❖ What happens if we access the following addresses?

  ▪ 8, 24, 8, 24, 8, …?

  ▪ Conflict in cache (misses!)

  ▪ Rest of cache goes *unused*

❖ Solution?
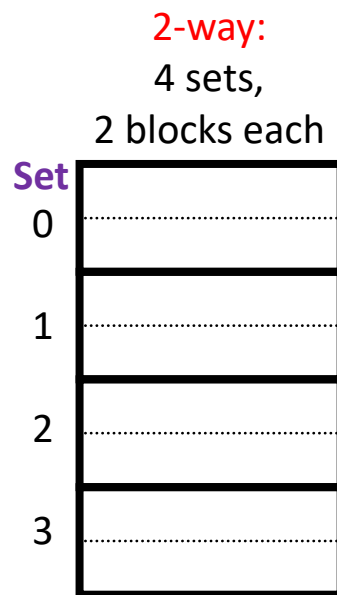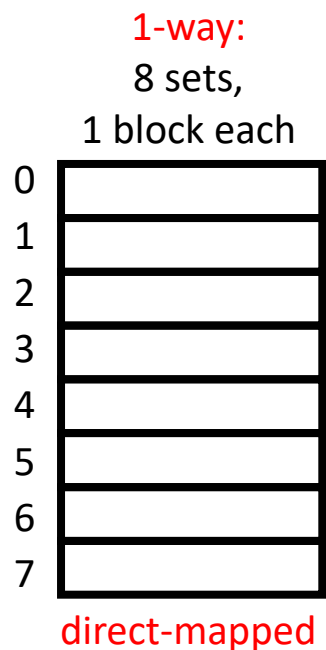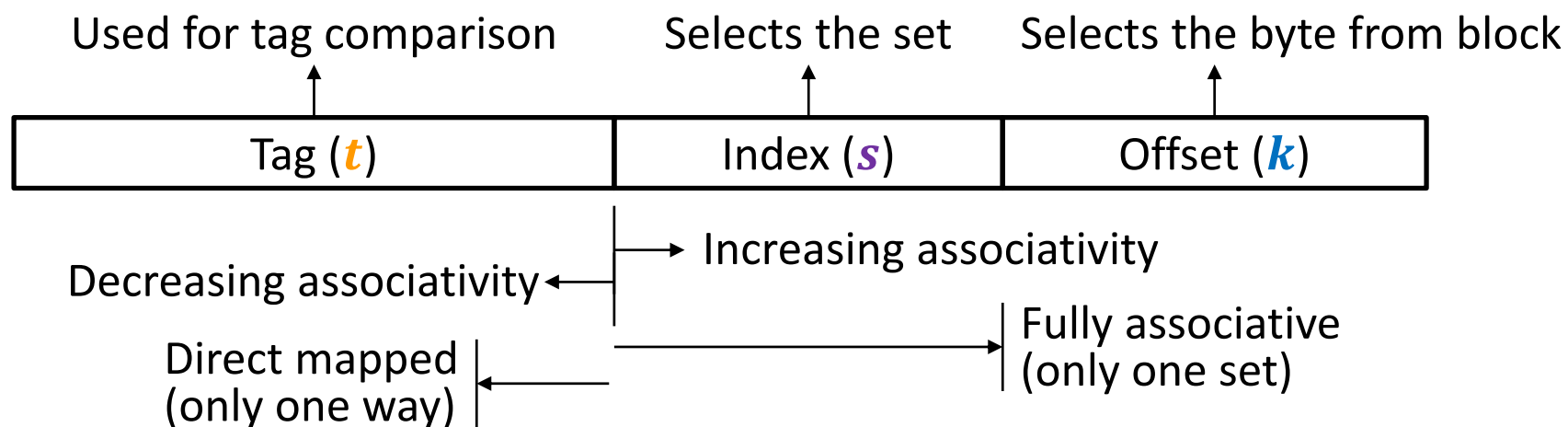
Associativity

- ❖ What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- ❖ So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**

1-way:
8 sets,
1 block each

2-way:
4 sets,
2 blocks each

4-way:
2 sets,
4 blocks each

8-way:
1 set,
8 blocks

direct-mapped
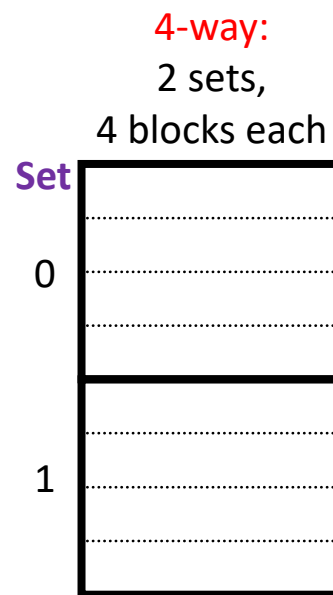
fully associative

Cache Organization (3)

Note: The textbook uses "b" for offset bits

❖ Associativity (E): # of ways for each set

  ▪ Such a cache is called an "E-*way set associative cache*"

  ▪ We now index into cache *sets*, of which there are $S = C/K/E$

  ▪ Use lowest $\log_2(C/K/E)$ = s bits of block address

    • <u>Direct-mapped</u>: E = 1, so s = $\log_2(C/K)$ as we saw previously

    • <u>Fully associative</u>: $E = C/K$, so s = 0 bits

Used for tag comparison   Selects the set   Selects the byte from block

Tag (t)	Index (s)	Offset (k)

Increasing associativity

Decreasing associativity

Direct mapped (only one way)

Fully associative (only one set)

Example Placement

block size:	16 B
capacity:	8 blocks
address:	16 bits

❖ Where would data from address `0x1833` be placed?

- Binary: `0b 0001 1000 0011 0011`

$$t = m-s-k \qquad s = \log_2(C/K/E) \qquad k = \log_2(K)$$

m-bit address:

Tag (t)	Index (s)	Offset (k)

s = ?
Direct-mapped

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

s = ?
2-way set associative

Set	Tag	Data
0		
1		
2		
3		

s = ?
4-way set associative

| Set	Tag	Data
0		
1		

Block Placement and Replacement

❖ *Any* empty block in the correct set may be used to store block

  ▪ **Valid bit** for each cache block indicates if data is valid (1) or garbage (0)

❖ If there are no empty blocks, which one should we replace?

  ▪ No choice for direct-mapped caches

  ▪ Caches typically use something close to *least recently used (LRU)* (hardware usually implements "*not most recently used*")

Direct-mapped   2-way set associative   4-way set associative

Polling Questions

❖ We have a cache of size 2 KiB with block size of 128 B. If our cache has 2 sets, what is its associativity?

   ▪ Vote in Ed Lessons

   A. 2
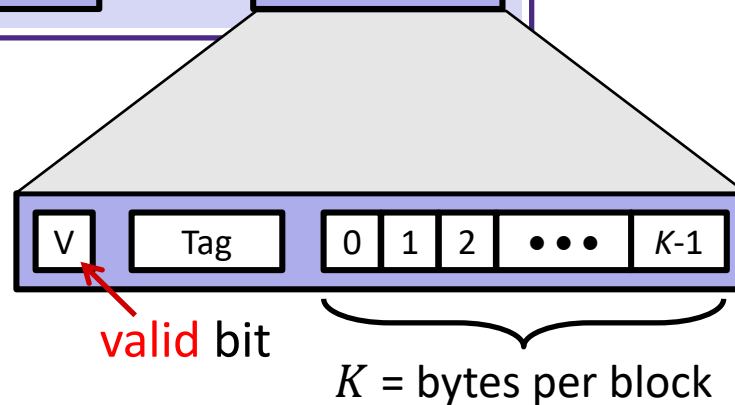
   B. 4

   C. 8

   D. 16

   E. We're lost…

❖ If addresses are 16 bits wide, how wide is the Tag field?

General Cache Organization (S, E, K)

E = blocks (or lines) per set

cache

set

line (block plus management bits)

S sets
= 2^s
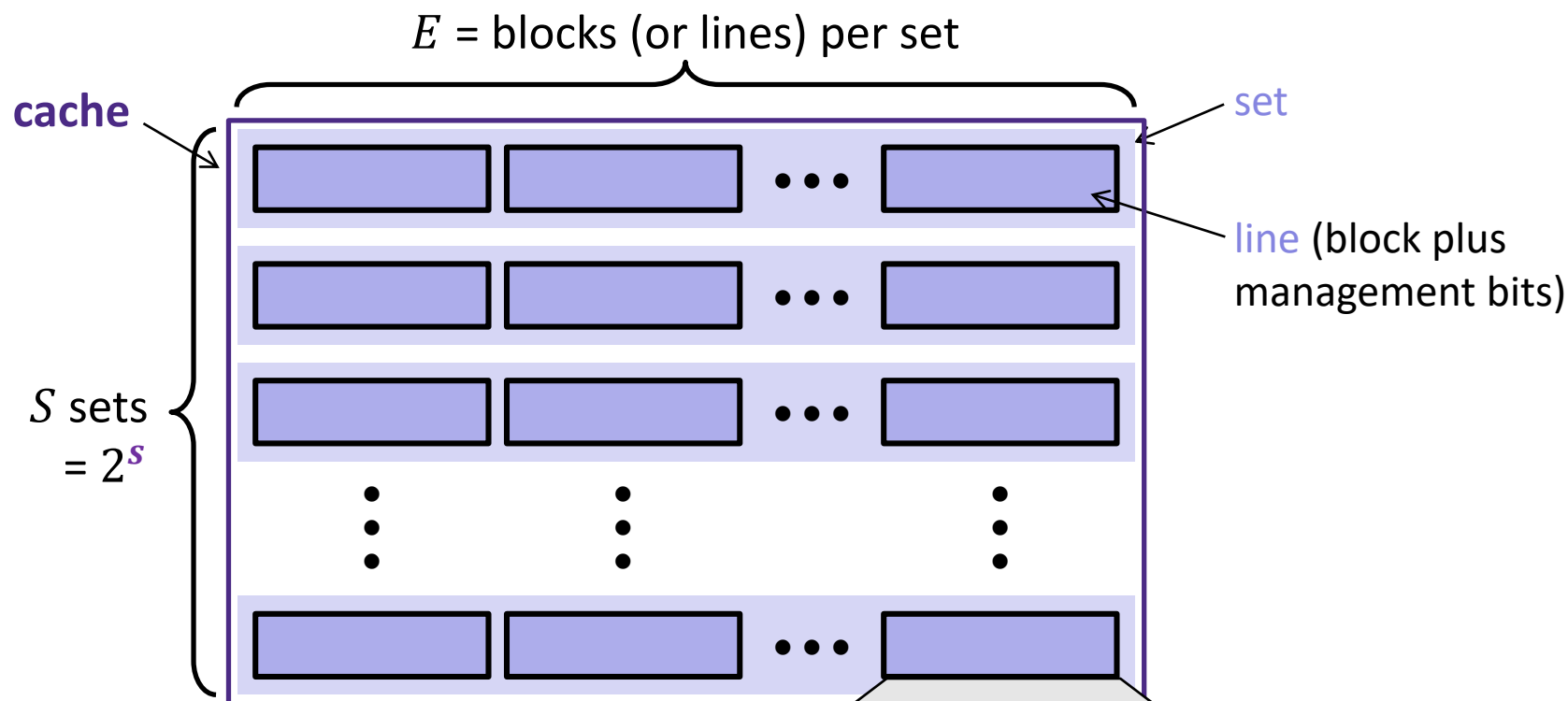
Cache size:
$C = K \times E \times S$ *data bytes*
(doesn't include V or Tag)

| V | Tag | 0 | 1 | 2 | • • • | K-1 |

valid bit

K = bytes per block

Notation Review

❖ We just introduced a lot of new variable names!

- Please be mindful of block size notation when you look at past exam questions or are watching videos

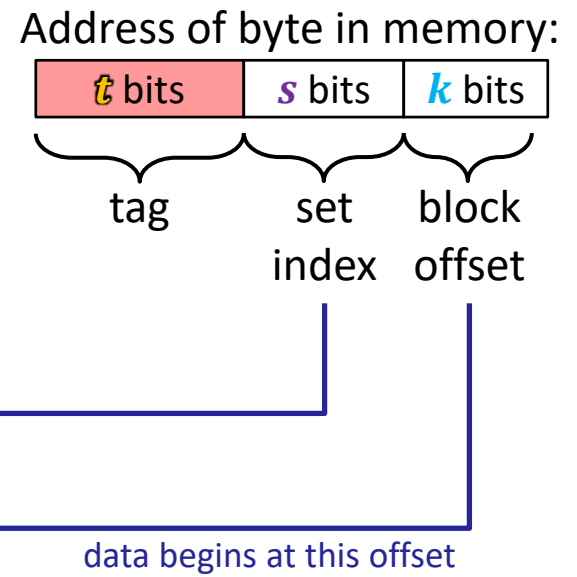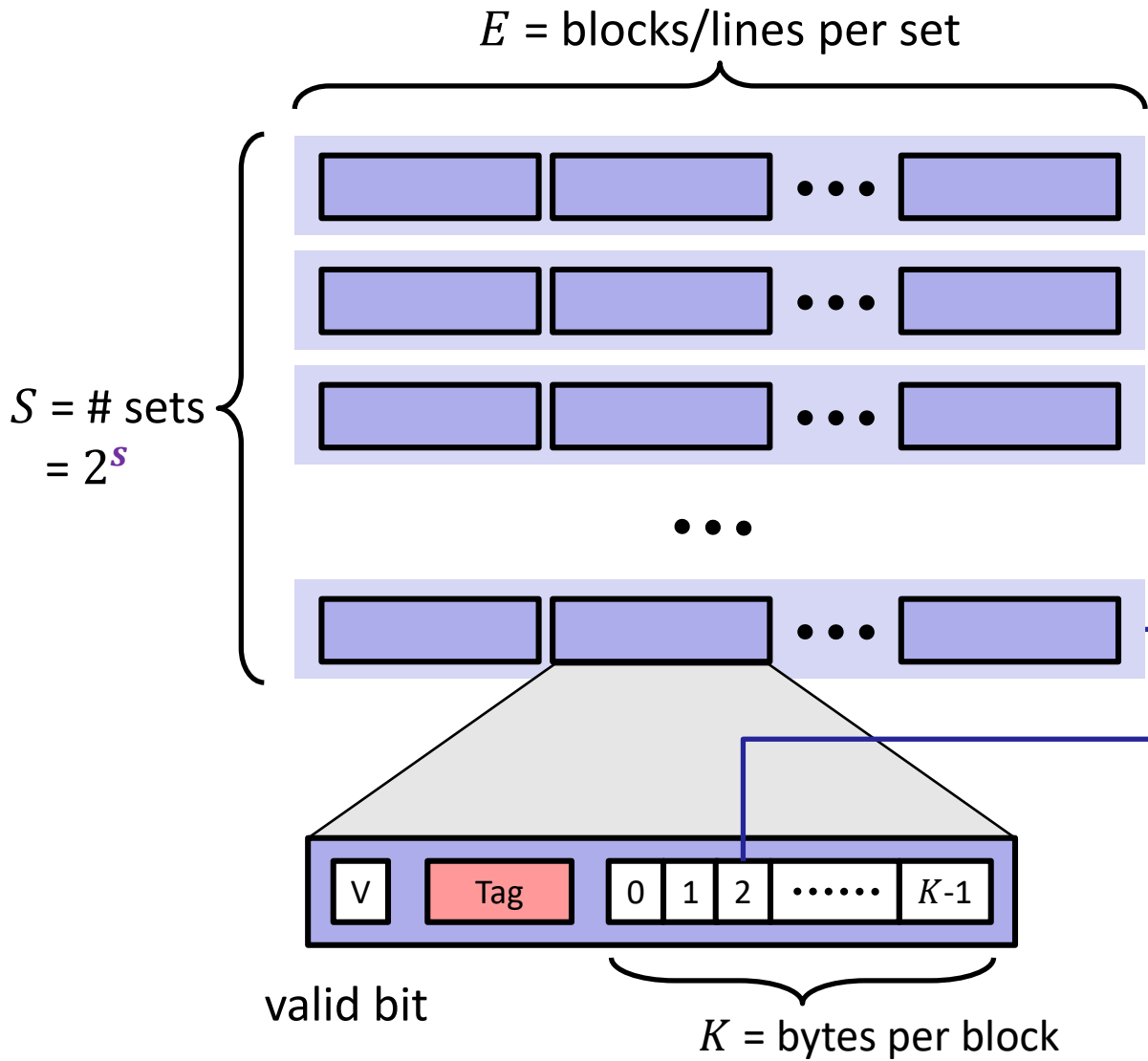| Parameter	Variable	Formulas
Block size	K (B in book)	
Cache size	C	$M = 2^m \leftrightarrow m = \log_2 M$
Associativity	E	$S = 2^s \leftrightarrow s = \log_2 S$
Number of Sets	S	$K = 2^k \leftrightarrow k = \log_2 K$
Address space	M	
Address width	m	$C = K \times E \times S$
Tag field width	t	$s = \log_2(C/K/E)$
Index field width	s	$m = t + s + k$
Offset field width	k (b in book)	

Example Cache Parameters Problem

❖ 1 KiB address space, 125 cycles to go to memory. Fill in the following table:

Cache Size	64 B
Block Size	8 B
Associativity	2-way
Hit Time	3 cycles
Miss Rate	20%
Tag Bits	
Index Bits	
Offset Bits	
AMAT	

Cache Read

E = blocks/lines per set

S = # sets $= 2^s$

Address of byte in memory:

t bits	s bits	k bits

tag   set index   block offset

data begins at this offset

| V | Tag | 0 | 1 | 2 | ······ | K-1 |

valid bit

K = bytes per block

15

Example: Direct-Mapped Cache (E = 1)

Direct-mapped: One line per set
Block Size K = 8 B

$S=2^s$ sets

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

• • •

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Address of `int`:

| t bits | 0…01 | 100 |

find set

Example: Direct-Mapped Cache (E = 1)

Direct-mapped: One line per set
Block Size K = 8 B

Address of int:

valid? + match?: yes = hit

| | t bits | 0…01 | 100 |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

block offset

Example: Direct-Mapped Cache (E = 1)

Direct-mapped: One line per set
Block Size K = 8 B

Address of `int`:

valid? + match?: yes = hit

| | t bits | 0...01 | 100 |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

block offset

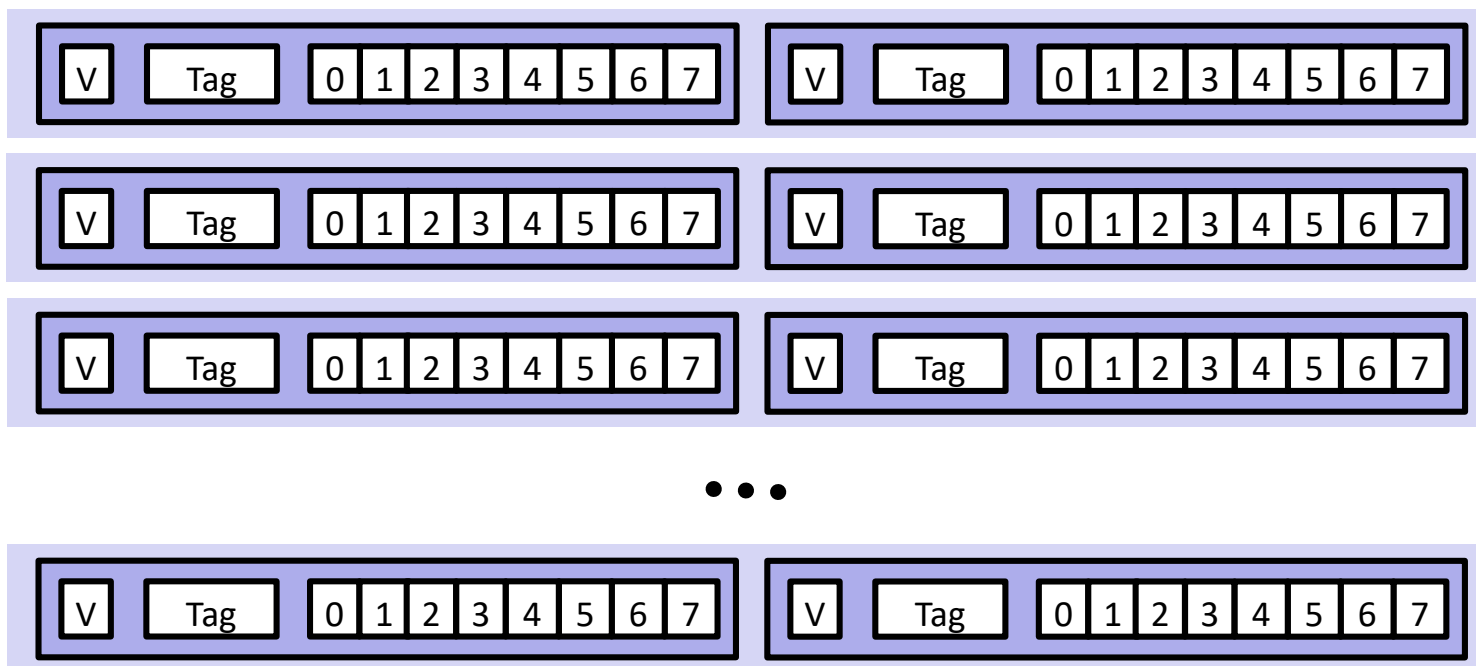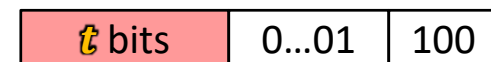`int` (4 B) is here
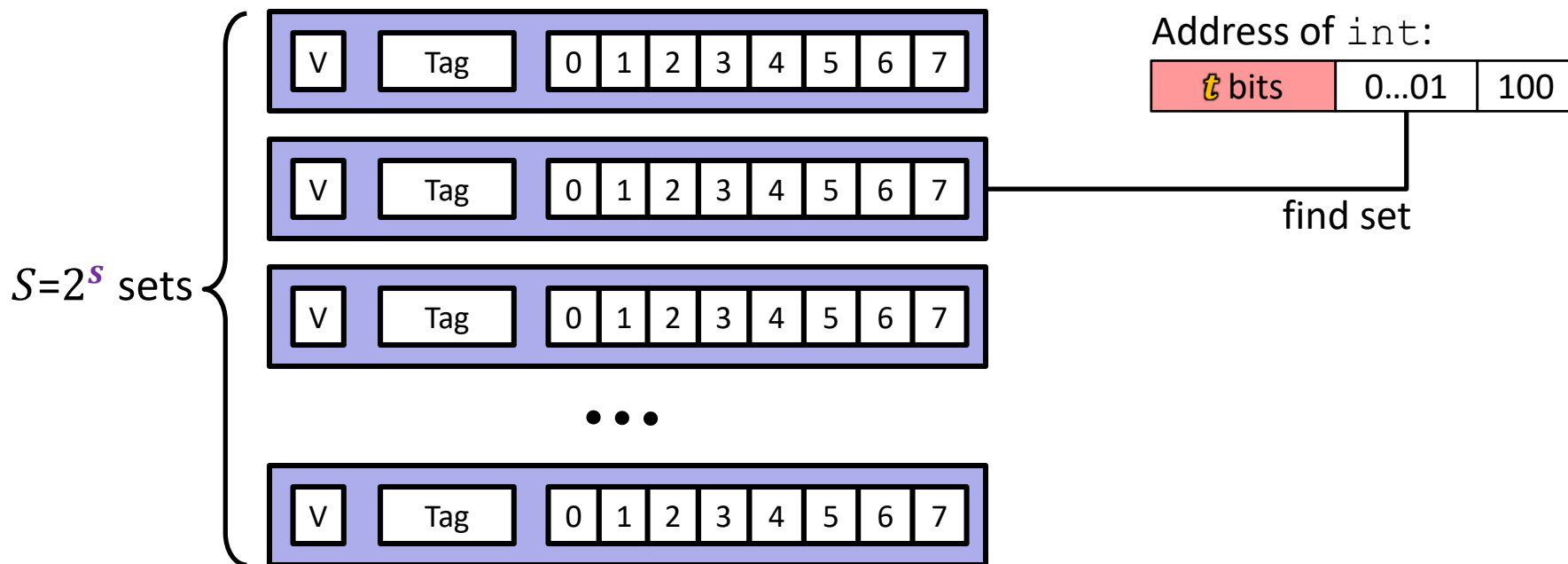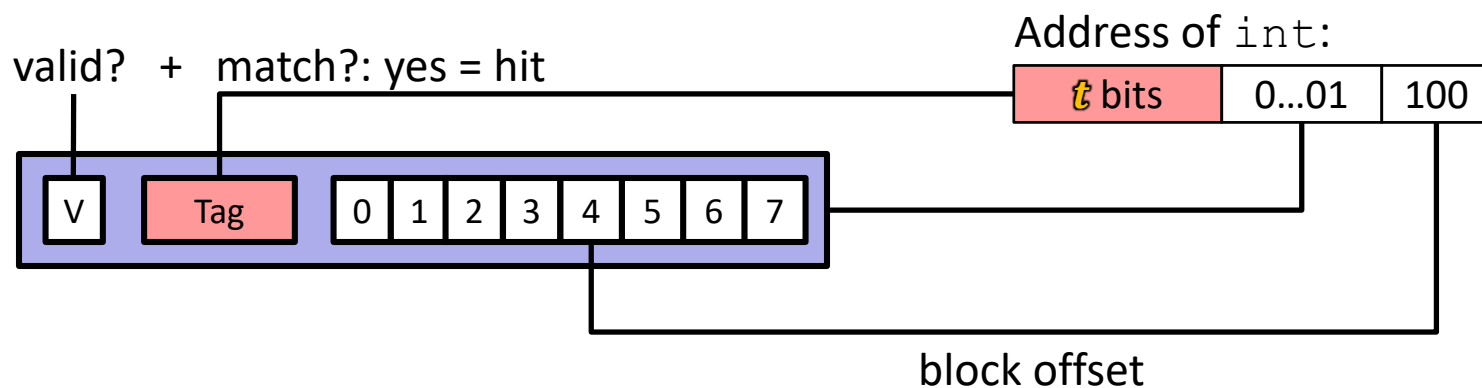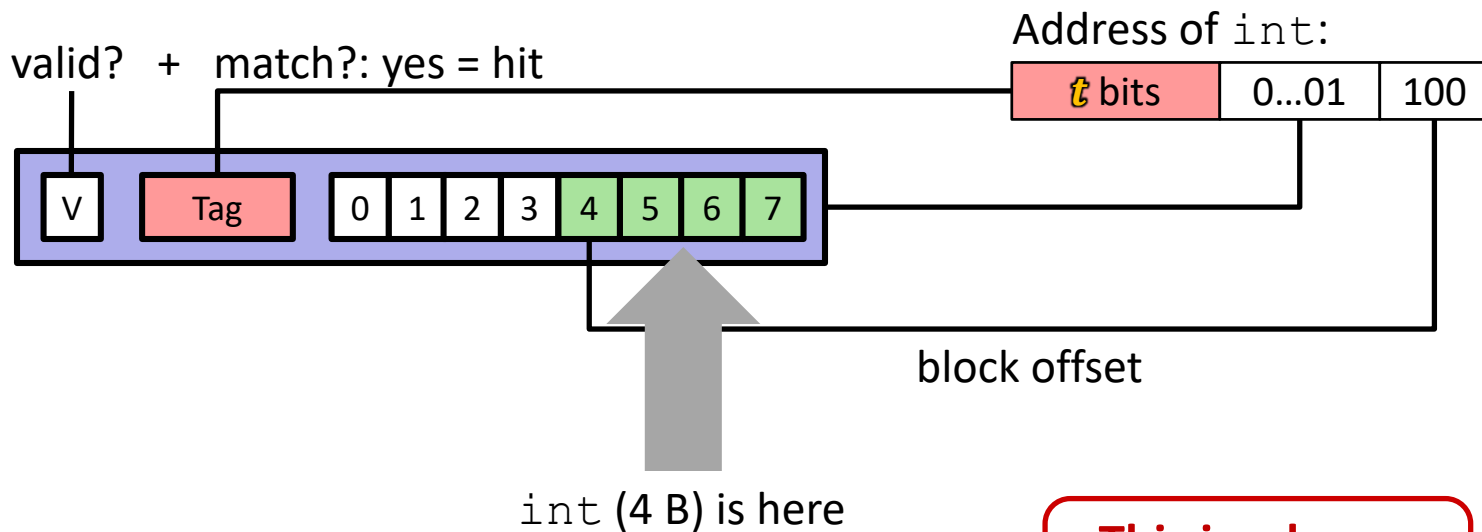
This is why we want alignment!

No match? Then old line gets evicted and replaced

Example: Set-Associative Cache (E = 2)

2-way: Two lines per set
Block Size K = 8 B

Address of `short int`:

t bits	0…01	100

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | find set |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

• • •

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

Example: Set-Associative Cache (E = 2)

2-way: Two lines per set
Block Size K = 8 B

Address of `short int`:

| t bits | 0...01 | 100 |

compare *both*

valid? + match: yes = hit

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

block offset

Example: Set-Associative Cache (E = 2)

2-way: Two lines per set
Block Size K = 8 B

Address of short int:

| t bits | 0...01 | 100 |

compare *both*

valid? + | match: yes = hit

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| V | Tag | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

block offset

short int (2 B) is here
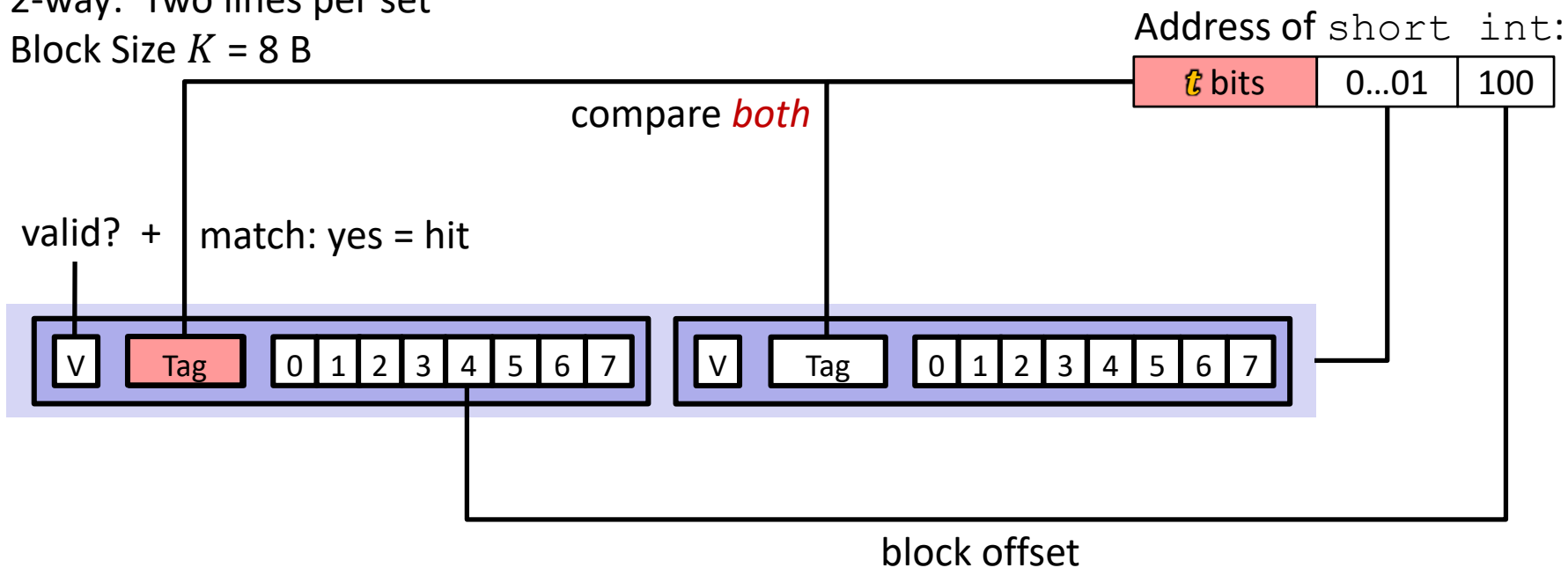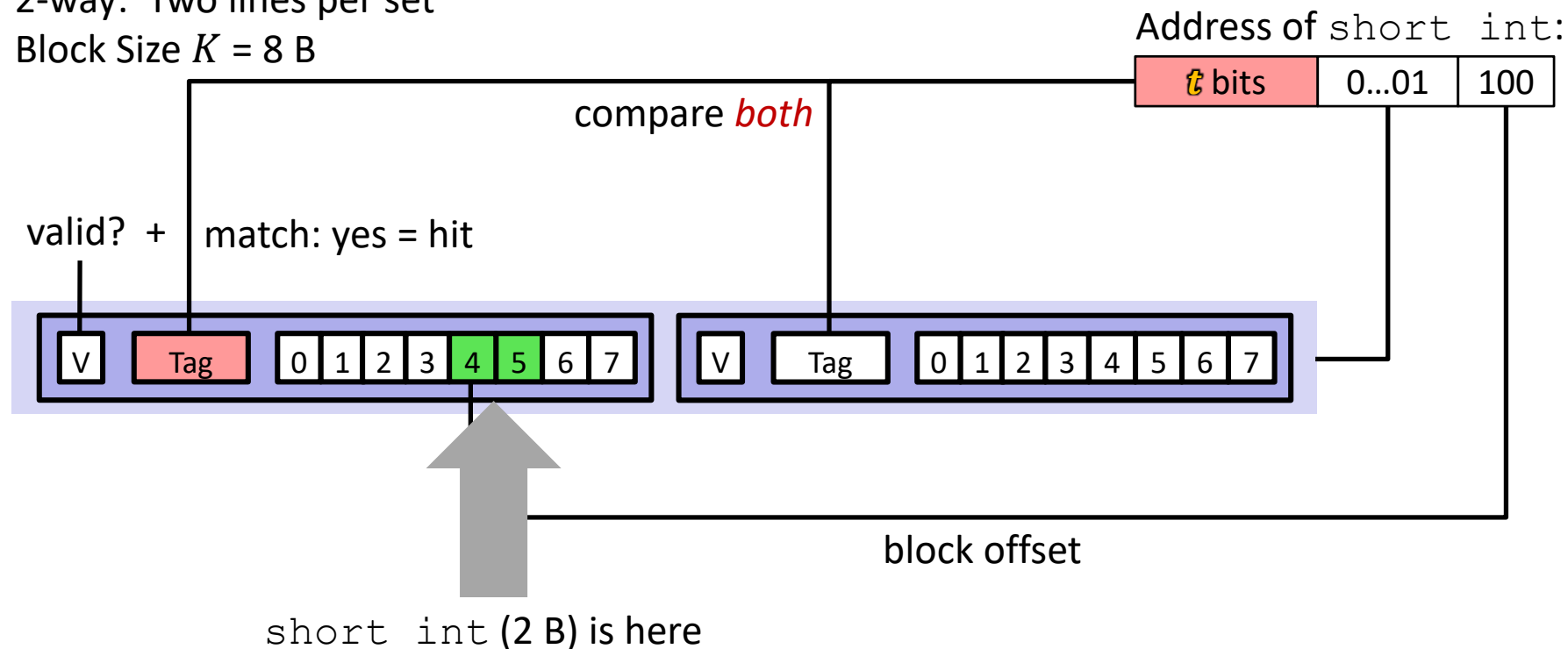
No match?

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), …

Types of Cache Misses: 3 C's!

❖ **Compulsory** (cold) miss
  ▪ Occurs on first access to a block

❖ **Conflict** miss
  ▪ Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
    • *e.g.* referencing blocks 0, 8, 0, 8, ... could miss every time
  ▪ Direct-mapped caches have more conflict misses than E-way set-associative (where $E > 1$)

❖ **Capacity** miss
  ▪ Occurs when the set of active cache blocks (the *working set*) is larger than the cache (just won't fit, even if cache was *fully-associative*)
  ▪ **Note:** *Fully-associative* only has Compulsory and Capacity misses

Example Code Analysis Problem

❖ Assuming the cache starts <u>cold</u> (all blocks invalid) and `sum`, `i`, and `j` are stored in registers, calculate the **miss rate**:

- m = 12 bits, C = 256 B, K = 32 B, E = 2

```
#define SIZE 8
long ar[SIZE][SIZE], sum = 0;   // &ar=0x800
for (int i = 0; i < SIZE; i++)
    for (int j = 0; j < SIZE; j++)
        sum += ar[i][j];
```