

Memory & Caches III

CSE 351 Autumn 2021

Instructor:

Justin Hsia

Teaching Assistants:

Allie Pflieger

Atharva Deodhar

Francesca Wang

Joy Dang

Monty Nitschke

Anirudh Kumar

Celeste Zeng

Hamsa Shankar

Julia Wang

Morel Fotsing

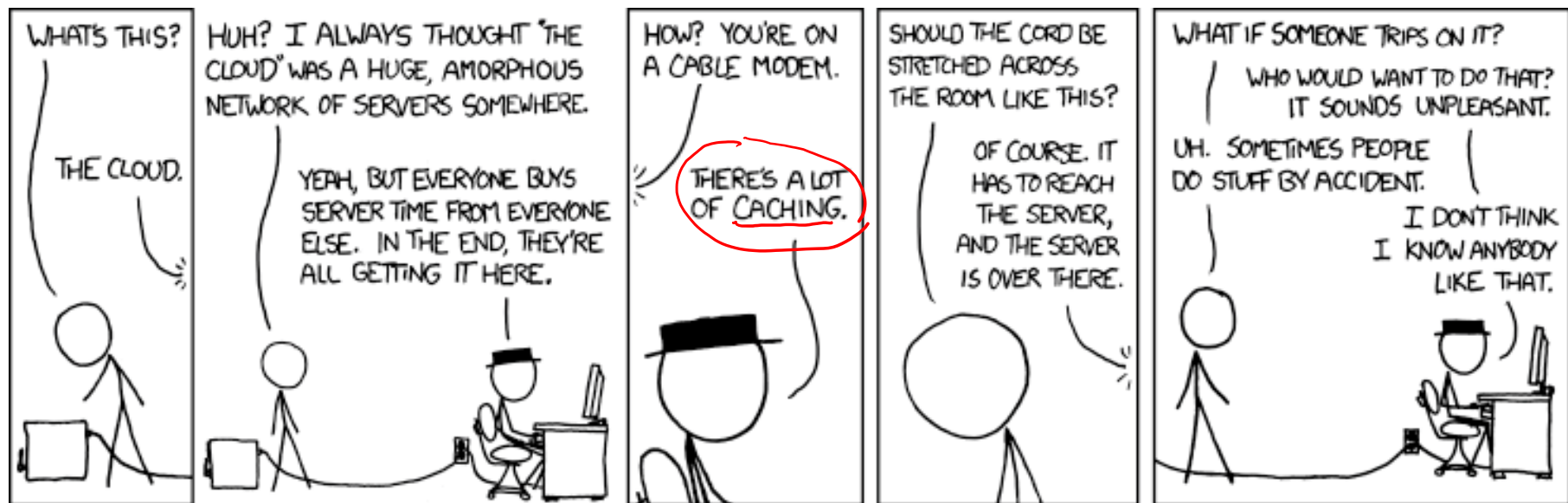
Assaf Vayner

Dominick Ta

Isabella Nguyen

Maggie Jiang

Sanjana Chintalapati



Relevant Course Information

- ❖ Lab 3 due tonight
- ❖ Lab 4 released Monday, due after Thanksgiving
 - Can do Part 1 after today; will need Lecture 20 to do Part 2
- ❖ hw19 due Wednesday (11/17)
 - Covers the major cache mechanics – BIG homework
- ❖ hw20 due Friday (11/19)
 - Preparation for Lab 4

Mid-quarter Survey Debrief

- ❖ Pace is a little fast (lecture pace, lots of assignments)
- ❖ Some readings are too dense/confusing
 - Ask on the Ed lecture post!
- ❖ We have e-flashcards? (Ed post #131)
 - #.2 lecture lesson → Terminology slide → .apkg file
 - <https://courses.cs.washington.edu/courses/cse351/21au/flashcards/>
- ❖ HW: formatting is frustrating, could use explanations
- ❖ Midterm:
 - 72-hour window was good (maybe include a weekend day?)
 - GDB/Stack question was long/difficult
 - Love and hate for the design/explanation questions

Making memory accesses fast!

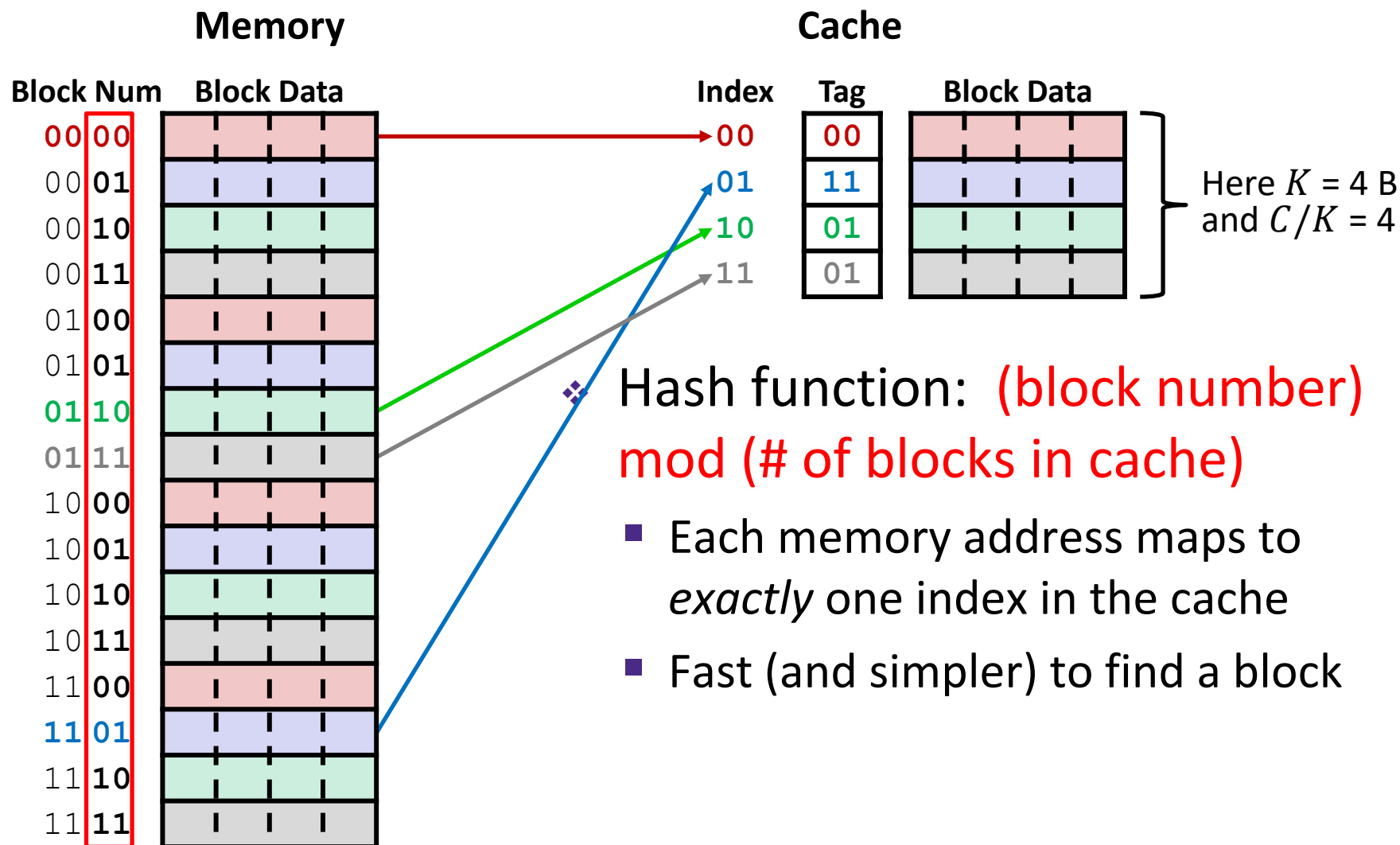
- ❖ Cache basics
- ❖ Principle of locality
- ❖ Memory hierarchies
- ❖ Cache organization
 - Direct-mapped (*sets*; index + tag)
 - **Associativity (*ways*)**
 - **Replacement policy**
 - Handling writes
- ❖ Program optimizations that consider caches

Reading Review

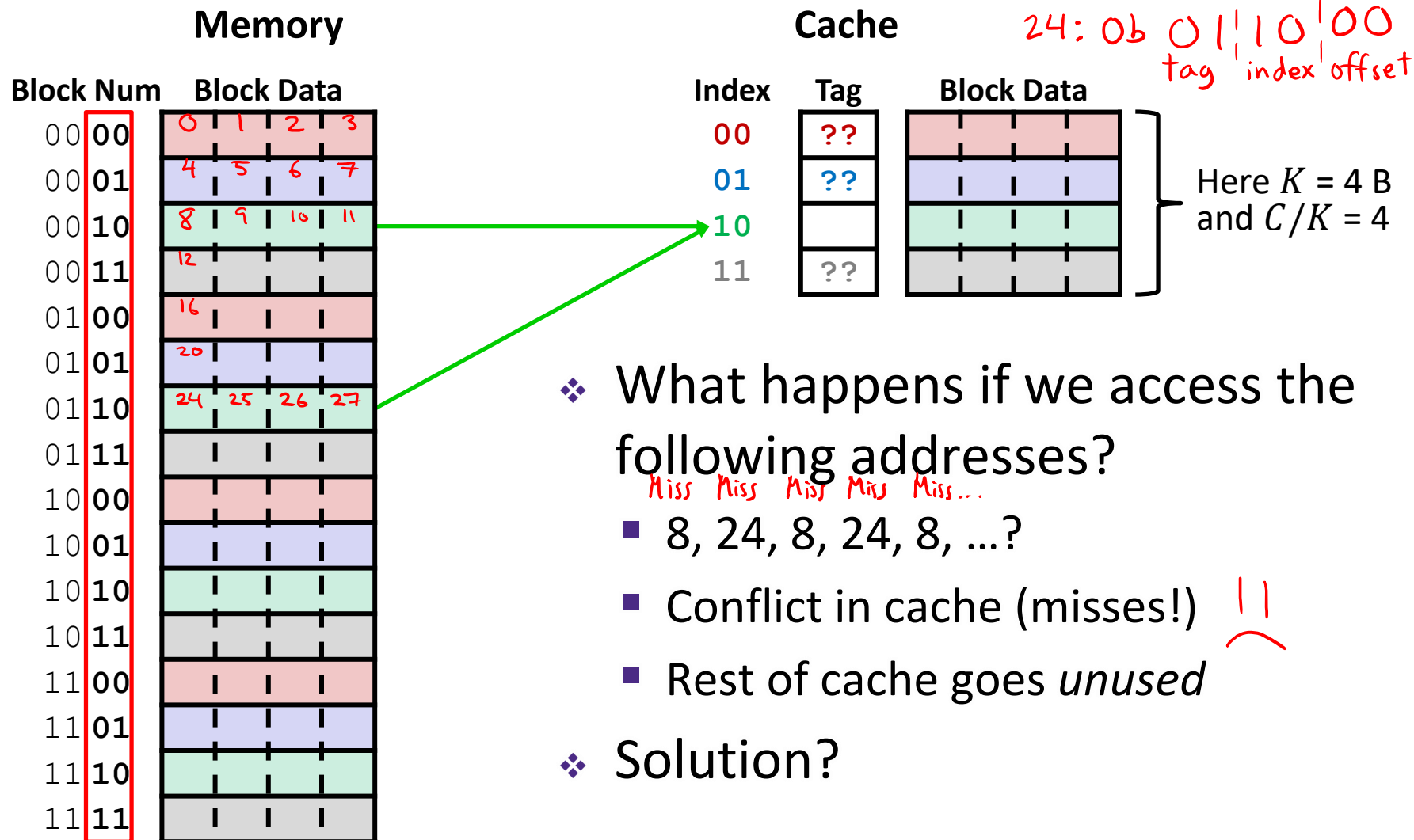
- ❖ Terminology:
 - Associativity: sets, fully-associative cache
 - Replacement policies: least recently used (LRU)
 - Cache line: cache block + management bits (valid, tag)
 - Cache misses: compulsory, conflict, capacity

- ❖ Questions from the Reading?

Review: Direct-Mapped Cache

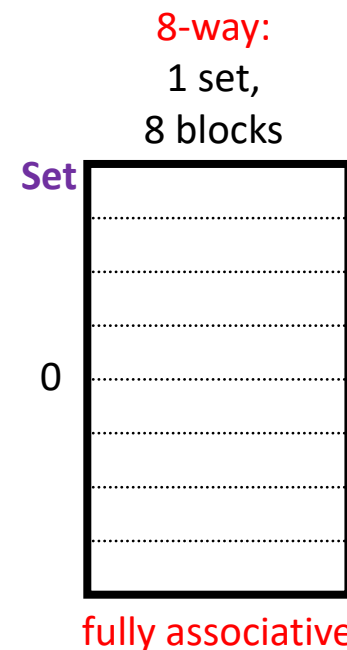
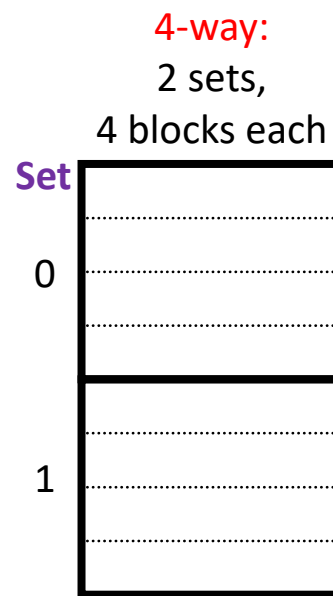
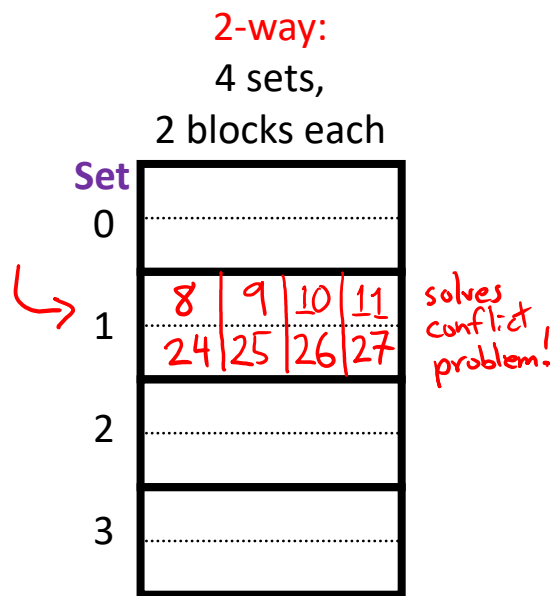
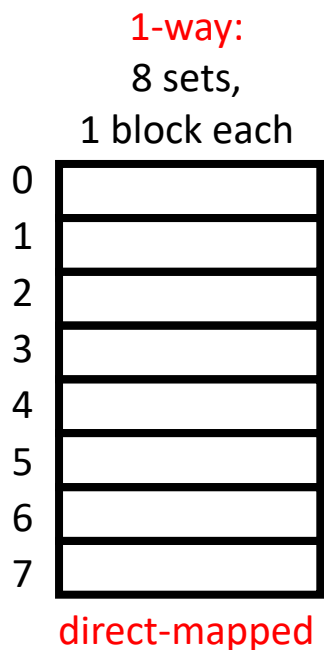


Direct-Mapped Cache Problem



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**

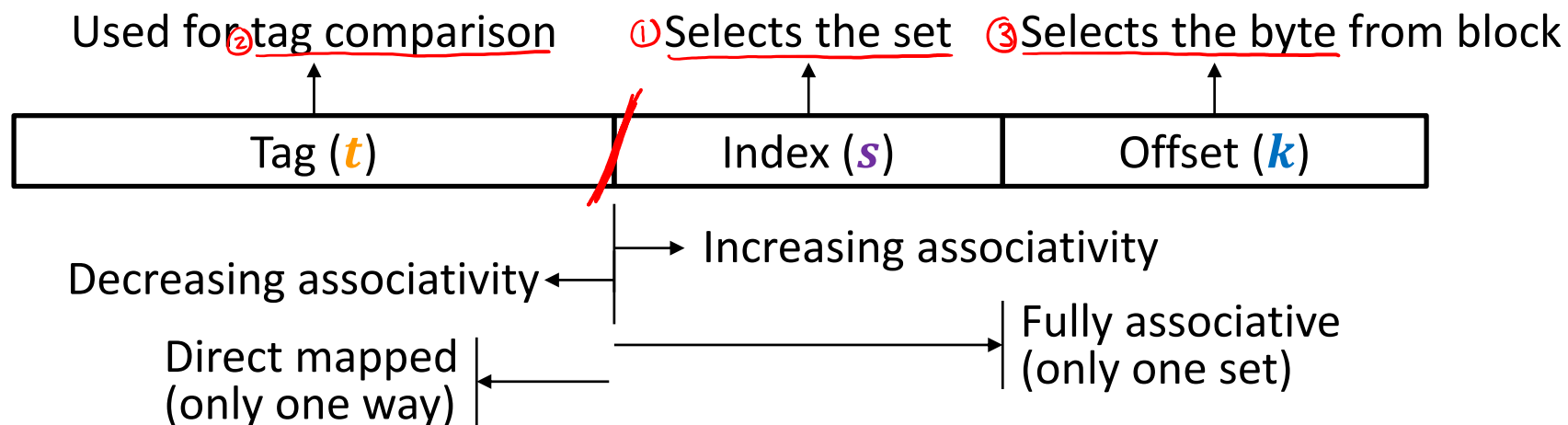


Cache Organization (3)

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
 - Direct-mapped: $E = 1$, so $s = \log_2(C/K)$ as we saw previously
 - Fully associative: $E = C/K$, so $s = 0$ bits

m bits total

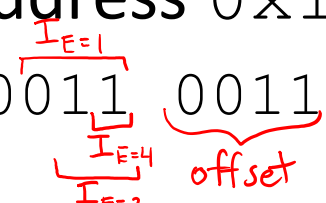


Example Placement

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

❖ Where would data from address 0x1833 be placed?

■ Binary: 0b 0001 1000 0011 0011



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

m-bit address:

Tag (<i>t</i>)	Index (<i>s</i>)	Offset (<i>k</i>)
------------------	--------------------	---------------------

s = ?

Direct-mapped

Set	Tag	Data
(000) 0		
(001) 1		
(010) 2		
(011) 3		✓
(100) 4		
(101) 5		
(110) 6		
(111) 7		

s = ?

2-way set associative

Set	Tag	Data
(00)0		
(01)1		
(10)2		
(11)3	✓	✓

s = ?

4-way set associative

Set	Tag	Data
(0)0		
(1)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 valid (1) or mystery (0) data
- ❖ 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

Set	V	Tag	Data
0			
1			
2			
3			
4			
5			
6			
7			

2-way set associative

Set	V	Tag	Data
0			
1			
2			
3			

4-way set associative

Set	V	Tag	Data
0			
1			

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?

$C = 2^{11} B$

$K = 2^7 B$

cache holds $C/K = 2^{11-7} = 2^4 = 16$ blocks

- Vote in Ed Lessons

A. 2

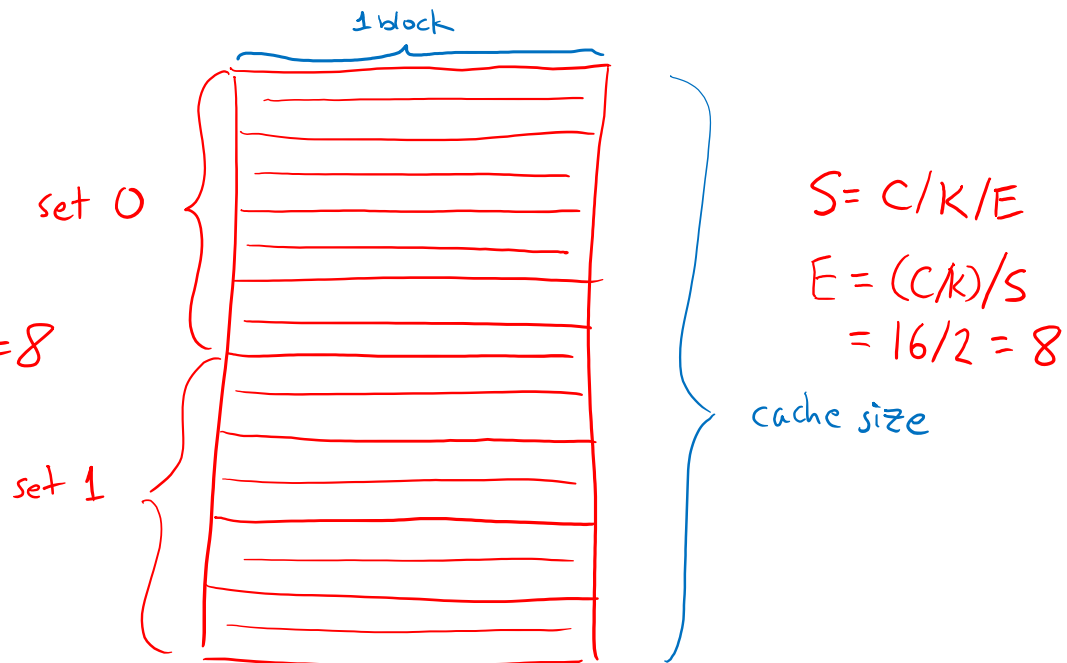
B. 4

C. 8

D. 16

E. We're lost...

each set has 8 blocks, so $E = 8$



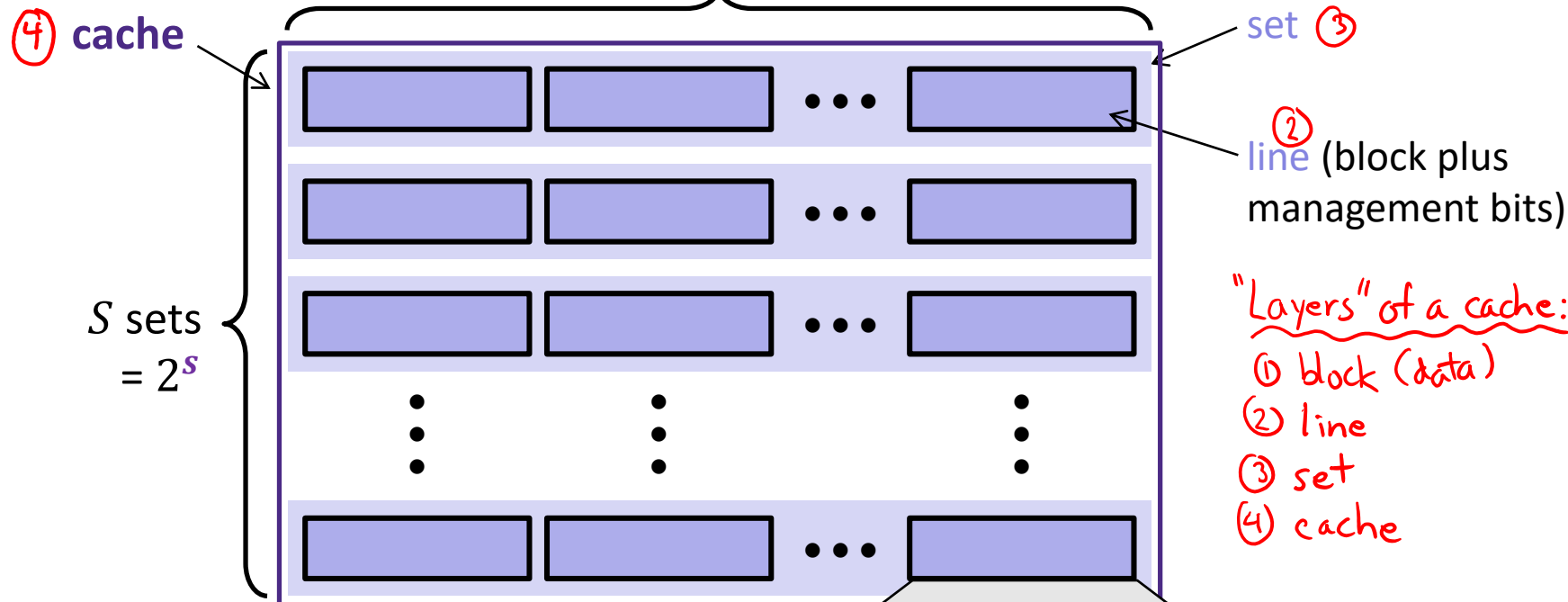
$m = 16$

- ❖ If addresses are 16 bits wide, how wide is the Tag field? $k = \log_2(K) = 7$ bits, $s = \log_2(S) = 1$ bit, $t = m - s - k = \boxed{8 \text{ bits}}$

General Cache Organization (S, E, K)

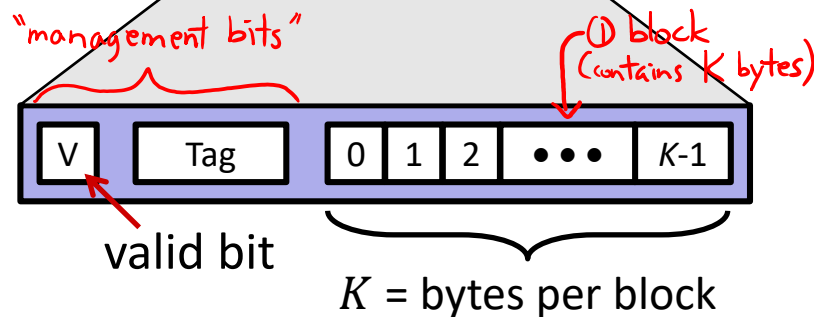
associativity
sets *block size*

$E =$ blocks (or lines) per set



Cache size:

$C = K \times E \times S$ data bytes
(doesn't include V or Tag)



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

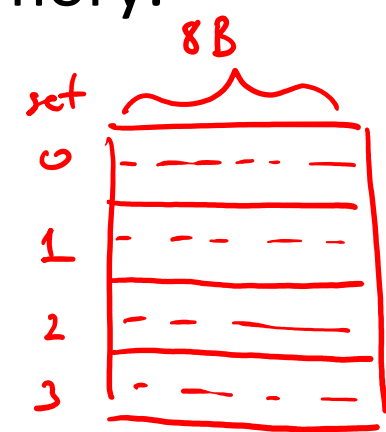
Parameter	Variable	Formulas
Block size	K (B in book)	$M = 2^m \leftrightarrow m = \log_2 M$ $S = 2^s \leftrightarrow s = \log_2 S$ $K = 2^k \leftrightarrow k = \log_2 K$ $C = K \times E \times S$ $s = \log_2(C/K/E)$ $m = t + s + k$
Cache size	C	
Associativity	E	
Number of Sets	S	
Address space	M	
Address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

Example Cache Parameters Problem

$\rightarrow 2^{10} \text{ B} \iff m = 10 \text{ bits}$ MP

- ❖ 1 KiB address space, 125 cycles to go to memory.

Fill in the following table:



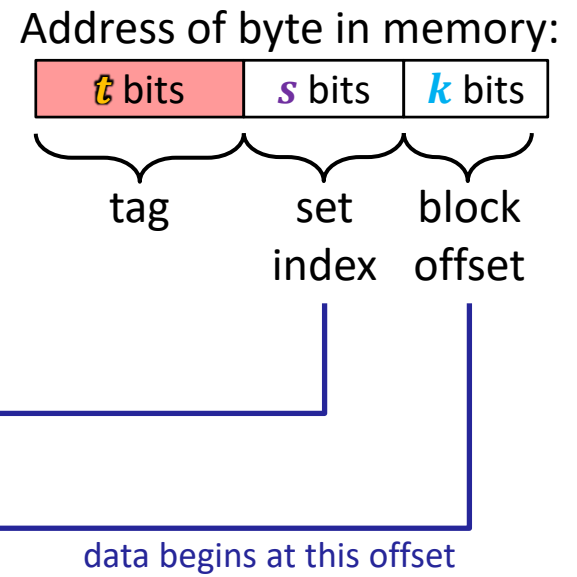
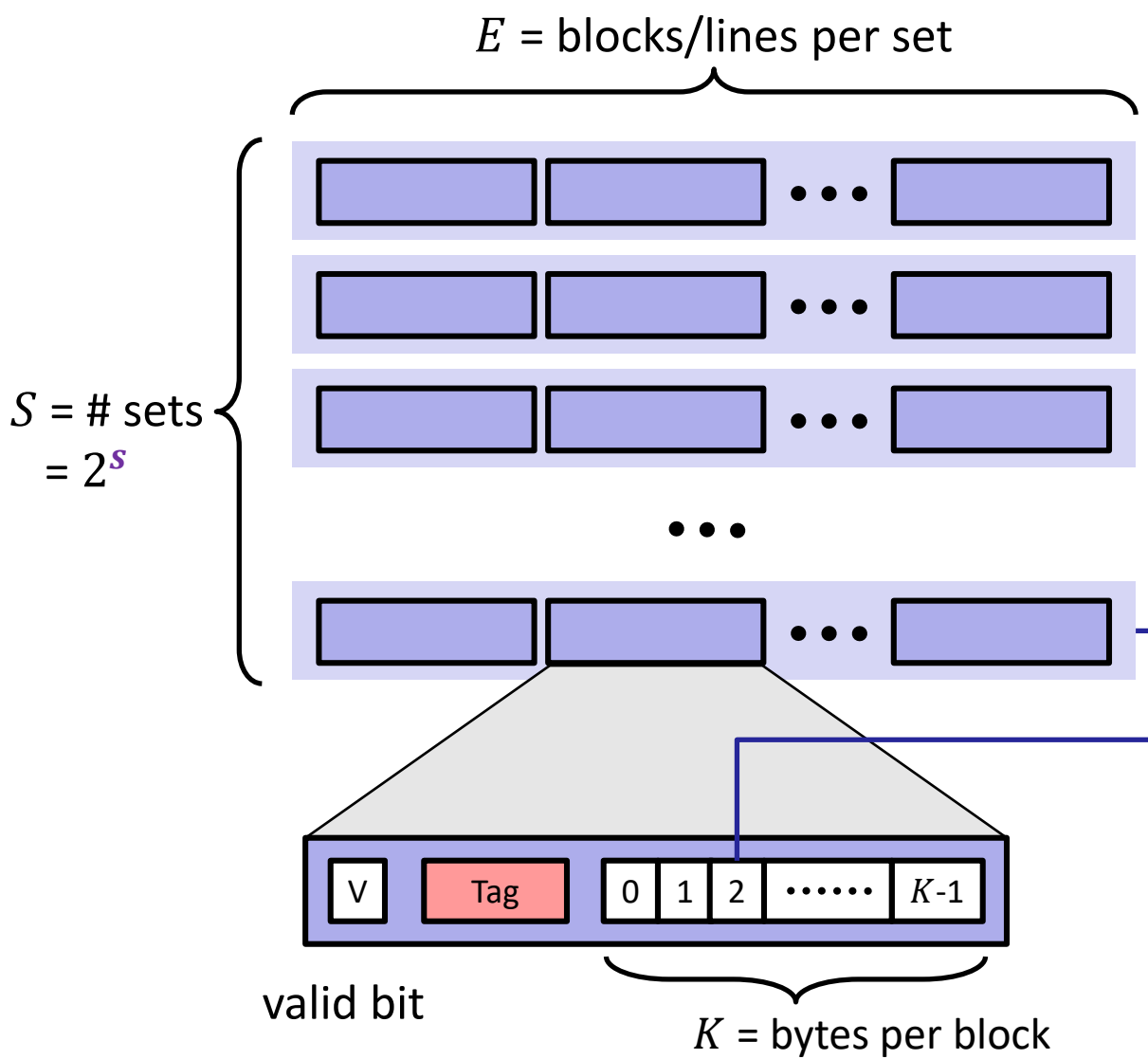
C	Cache Size	64 B	2^6
K	Block Size	8 B	2^3
E	Associativity	2-way	2^1
HT	Hit Time	3 cycles	
MR	Miss Rate	20%	
	Tag Bits	5	
	Index Bits	2	$2^6 / 2^3 / 2^1$
	Offset Bits	3	
	AMAT	$3 + 0.2(125) = 28 \text{ clock cycles}$	

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

$AMAT = HT + MR * MP$

Cache Read

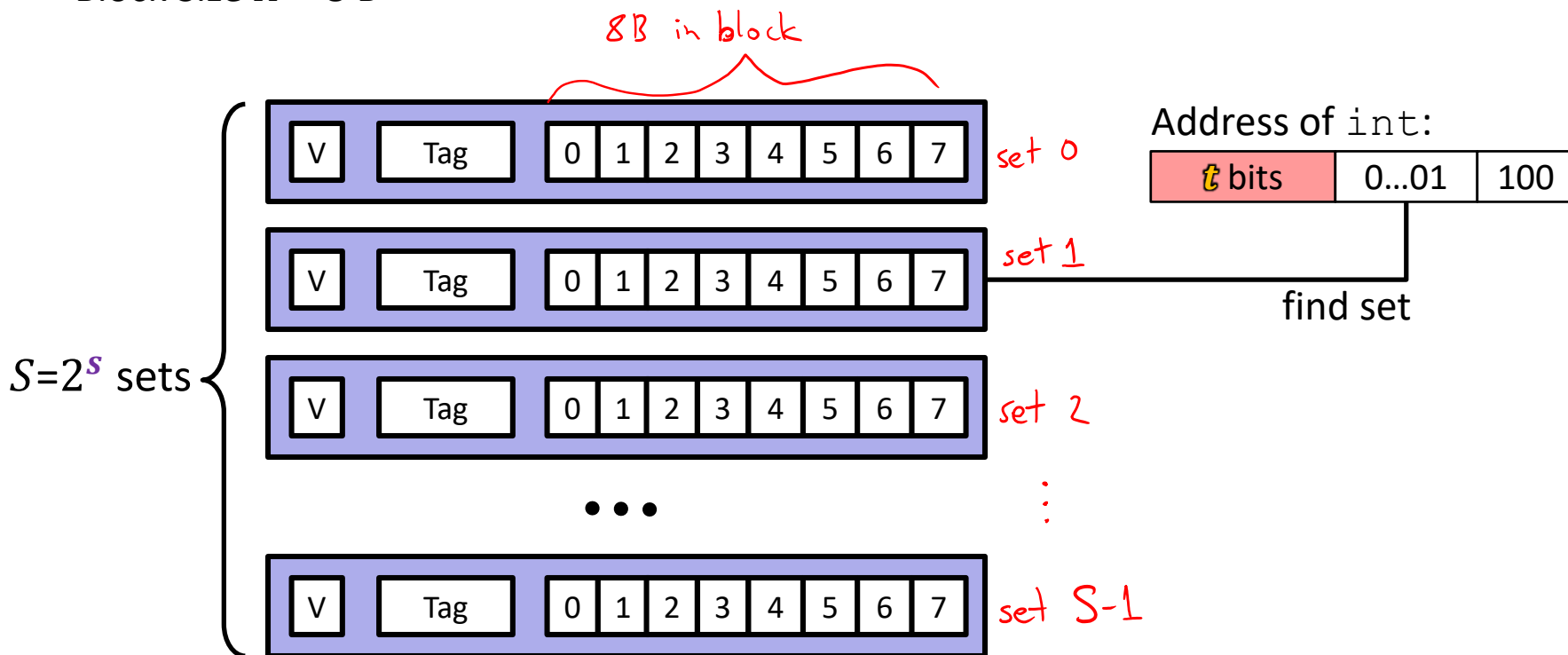
- 1) *Locate set*
- 2) *Check if any line in set is valid and has matching tag: hit*
- 3) *Locate data starting at offset*



Example: Direct-Mapped Cache ($E = 1$)

Direct-mapped: One line per set

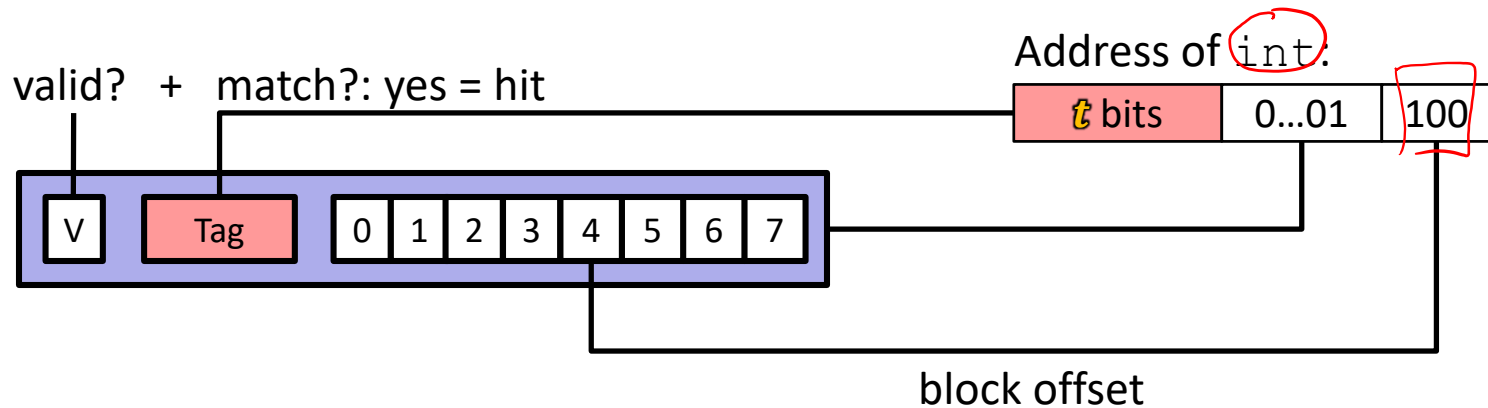
Block Size $K = 8$ B



Example: Direct-Mapped Cache ($E = 1$)

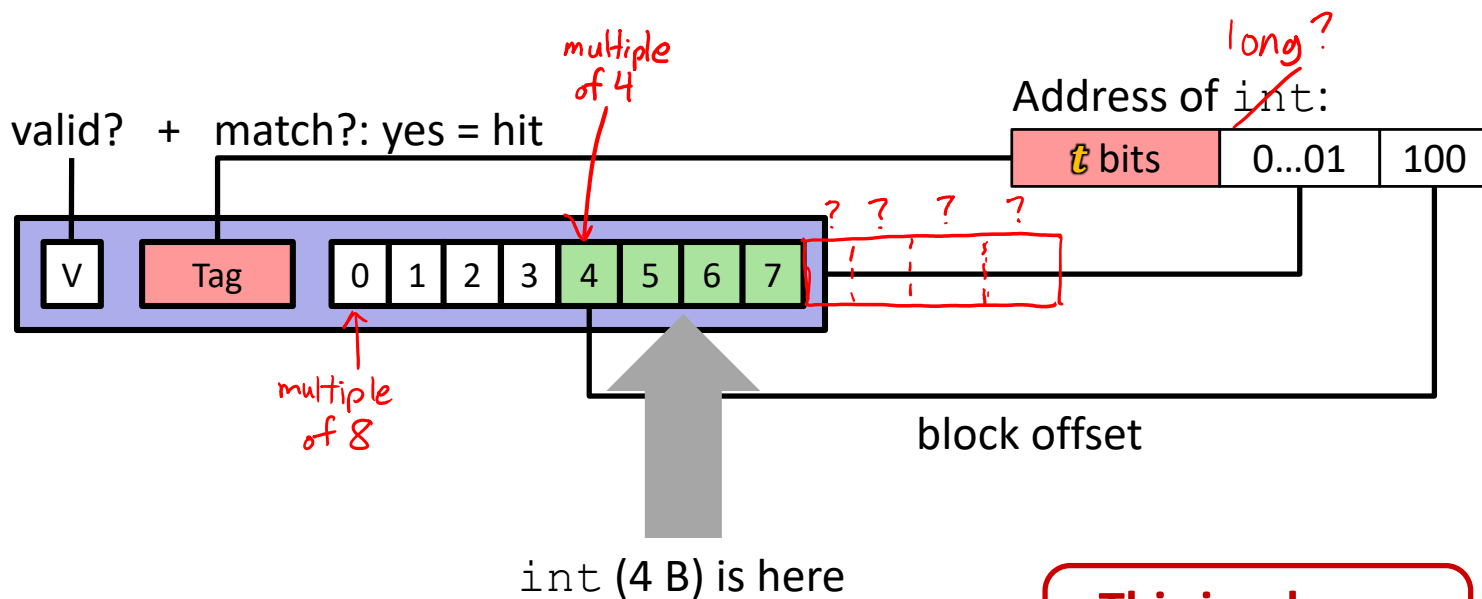
Direct-mapped: One line per set

Block Size $K = 8$ B



Example: Direct-Mapped Cache ($E = 1$)

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



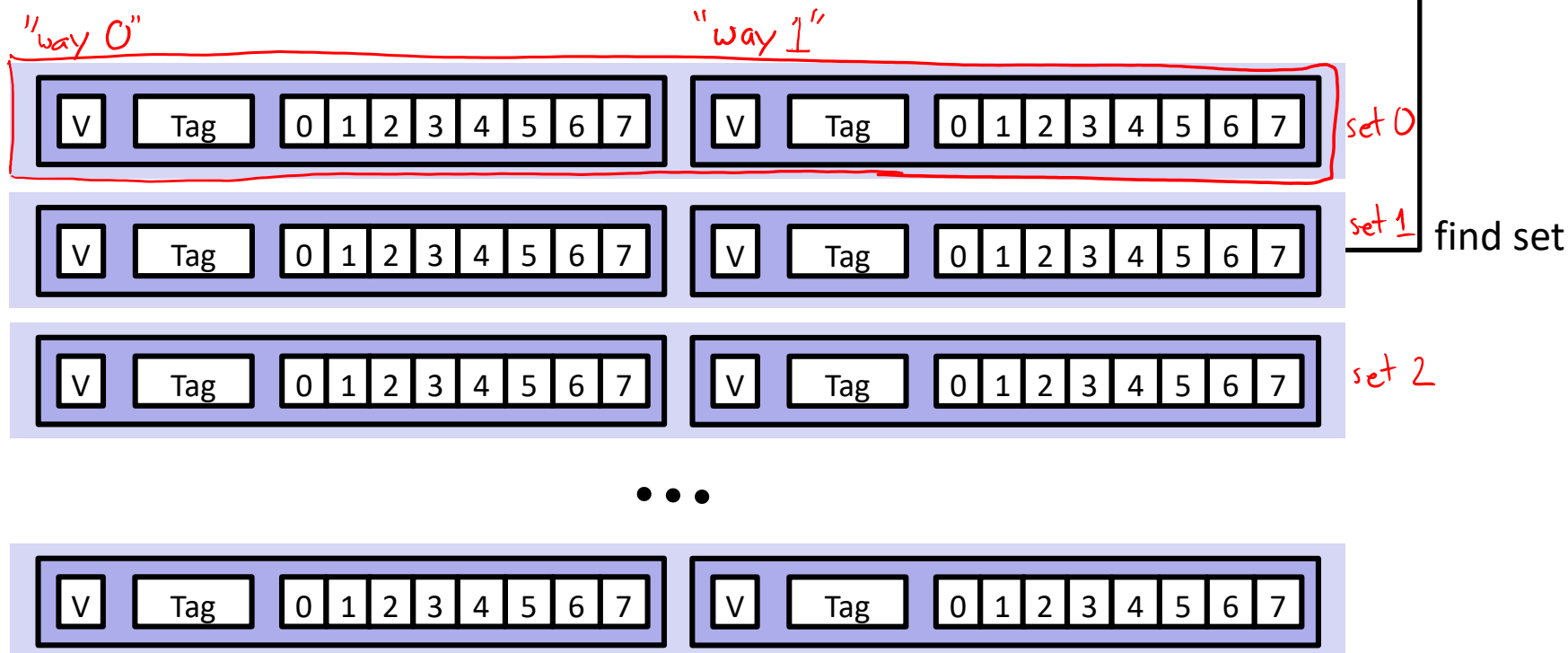
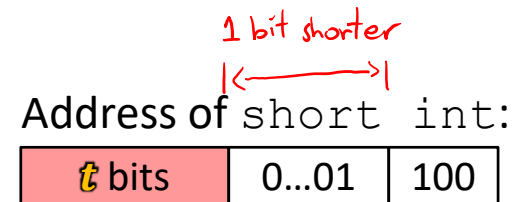
This is why we want alignment!

No match? Then old line gets evicted and replaced

no unnecessary extra cache accesses across block boundaries

Example: Set-Associative Cache ($E = 2$)

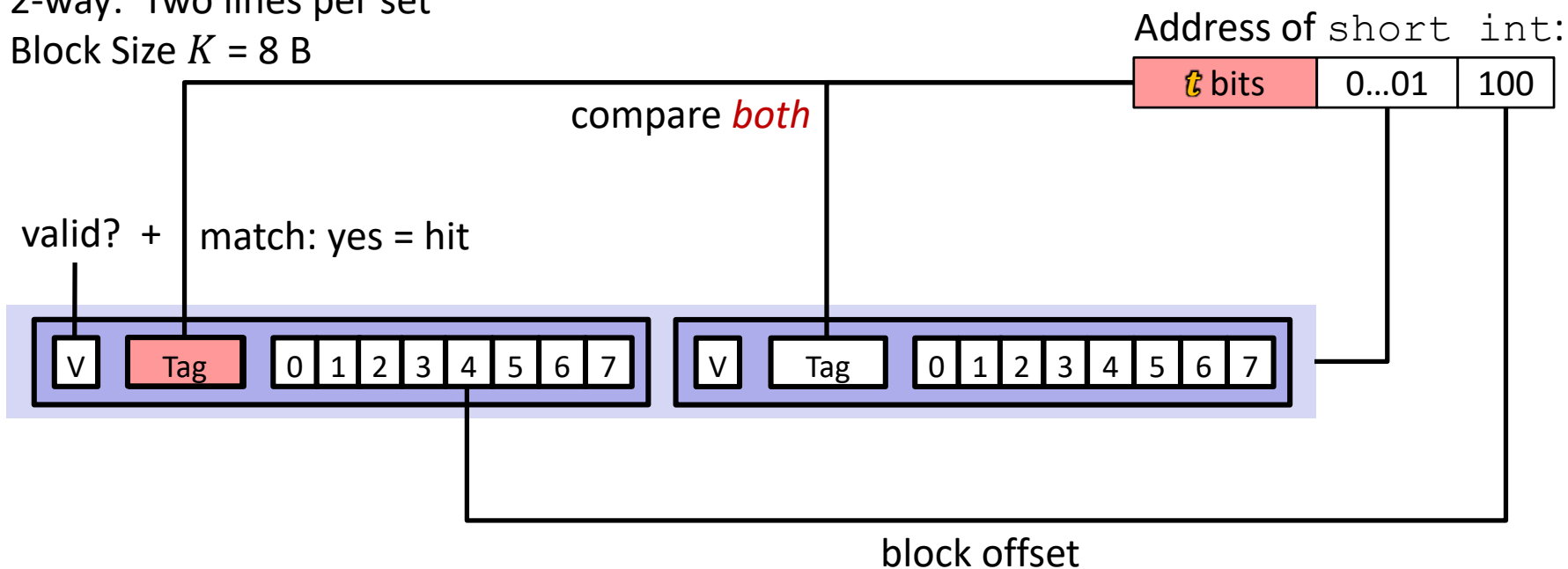
2-way: Two lines per set
 Block Size $K = 8$ B



Example: Set-Associative Cache ($E = 2$)

2-way: Two lines per set

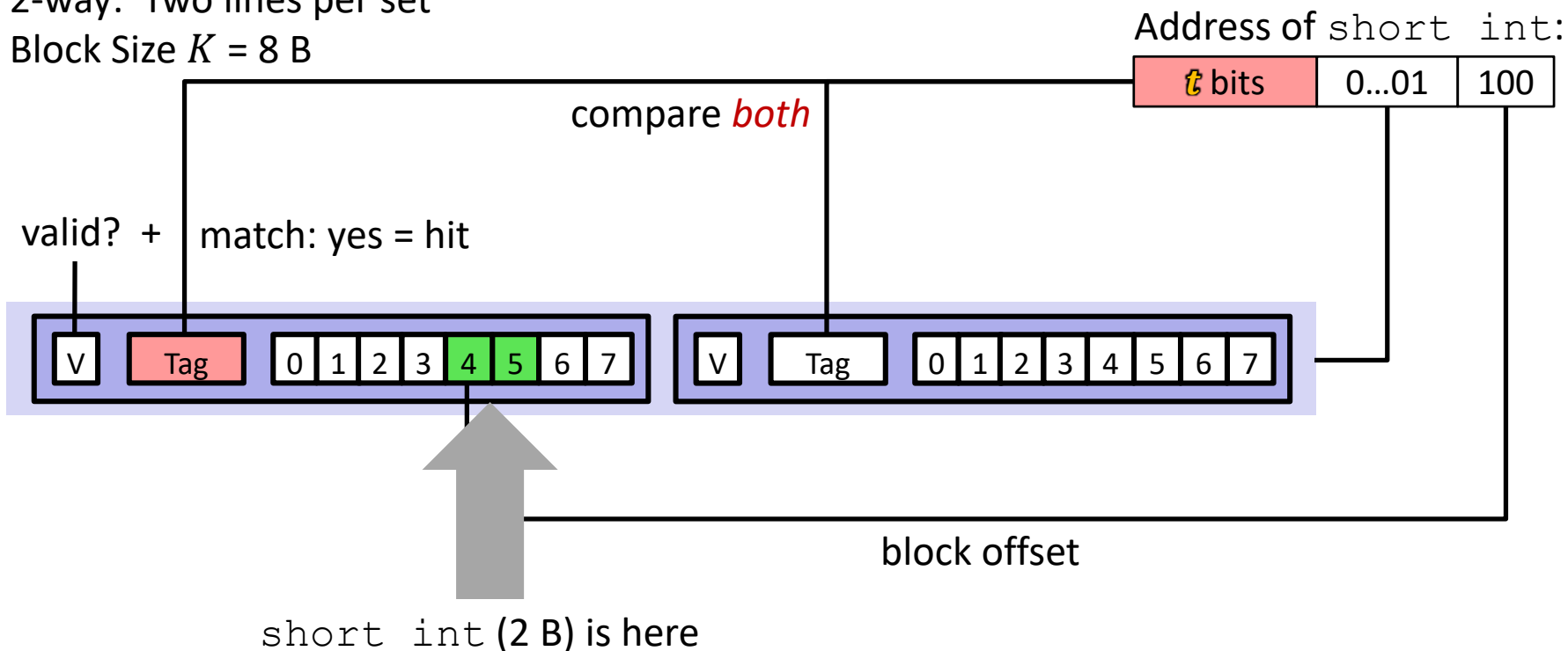
Block Size $K = 8$ B



Example: Set-Associative Cache ($E = 2$)

2-way: Two lines per set

Block Size $K = 8$ B



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

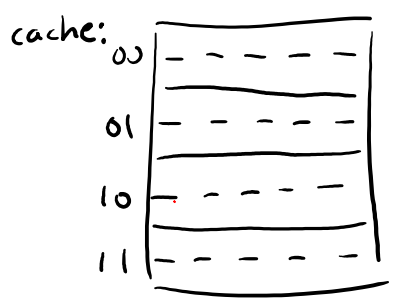
- Assuming the cache starts cold (all blocks invalid) and `sum`, `i`, and `j` are stored in registers, calculate the **miss rate**: 100%

t = 5 bits, s = 2 bits, k = 3 bits

- $m = 10$ bits, $C = 64$ B, $K = 8$ B, $E = 2$

2 bytes per element

```
#define SIZE 8
short ar[SIZE][SIZE], sum = 0; // &ar=0x200
for (int i = 0; i < SIZE; i++)
    for (int j = 0; j < SIZE; j++)
        sum += ar[j][i];
```



	tag	index	offset	
<code>ar[0][0]</code>	0b 10	0000	0000	→ M (1 st way)
<code>ar[1][0]</code>	0b 10	0001	0000	→ M (1 st way)
<code>ar[2][0]</code>	0b 10	0010	0000	→ M (2 nd way)
<code>ar[3][0]</code>	0b 10	0011	0000	→ M (2 nd way)
<code>ar[4][0]</code>	0b 10	0100	0000	→ M (replacement!)
⋮		⋮		
<code>ar[0][1]</code>	0b 10	0000	0010	→ M (conflict!)

matrix ar:

jumping by a row skips two block numbers (and sets)

A diagram showing a vertical stack of boxes representing a matrix. A blue box highlights one element, and an arrow points to the next element in the next row, illustrating the row-major traversal.

Cache block holds 4 elements of a row of the matrix