Caches III

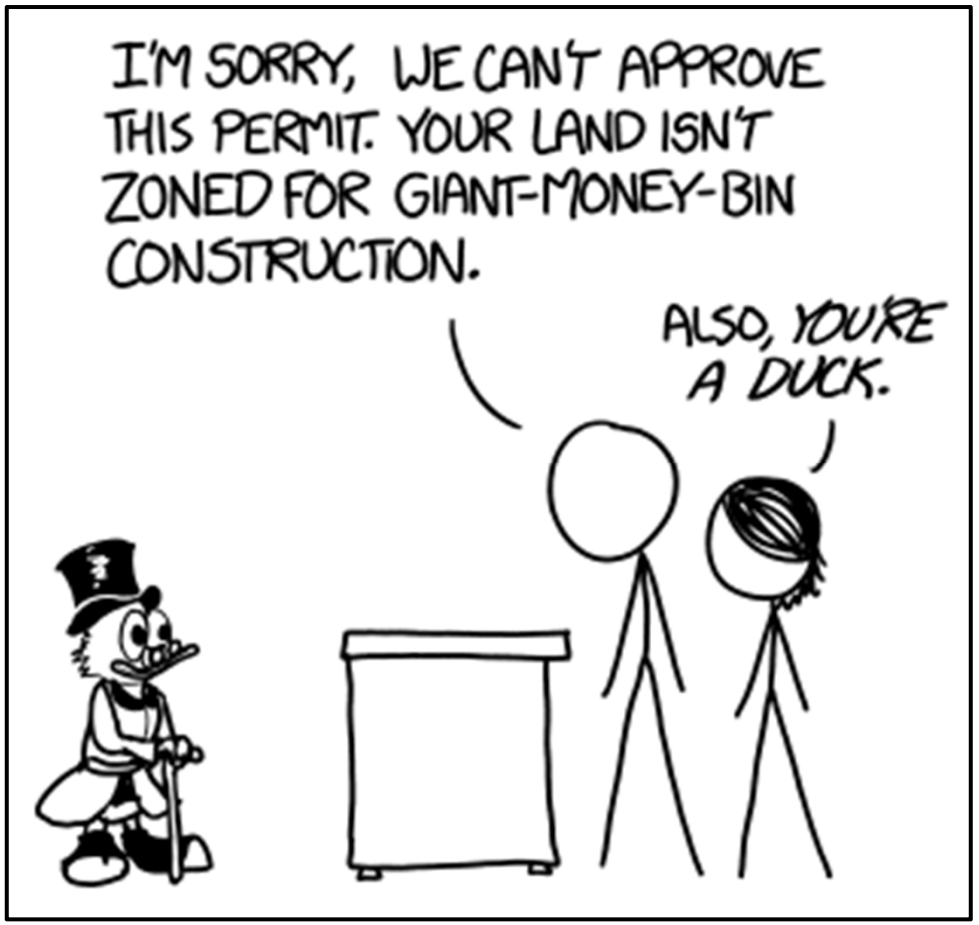
CSE 351 Spring 2019

Instructor:

Ruth Anderson

Teaching Assistants:

Gavin Cai
Jack Eggleston
John Feltrup
Britt Henderson
Richard Jiang
Jack Skalitzky
Sophie Tian
Connie Wang
Sam Wolfson
Casey Xing
Chin Yeoh



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

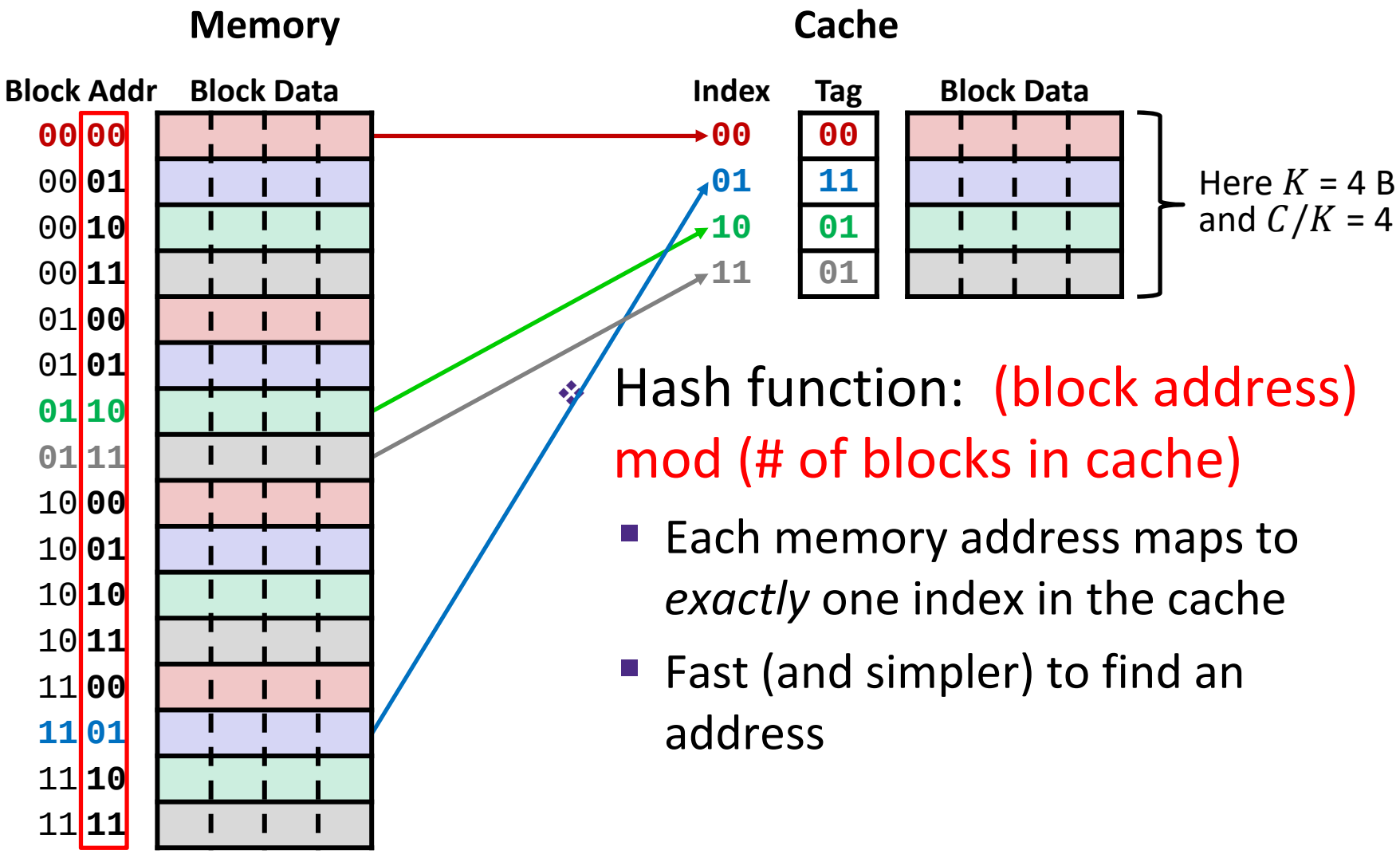
Administrivia

- Lab 3, due Wednesday (5/15)
- Homework 4 , due Wed (5/22) (Structs, Caches)

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

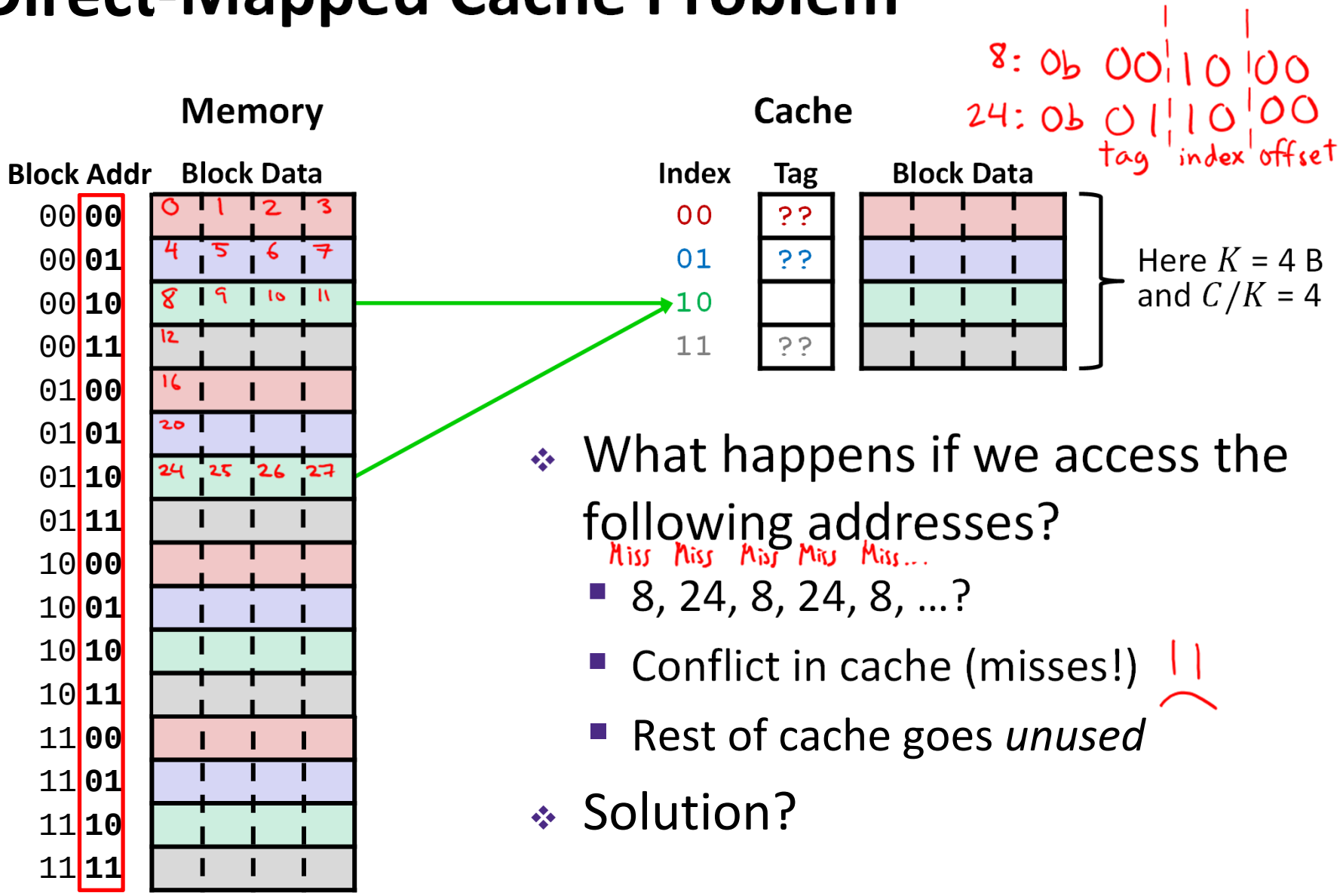
Direct-Mapped Cache



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

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

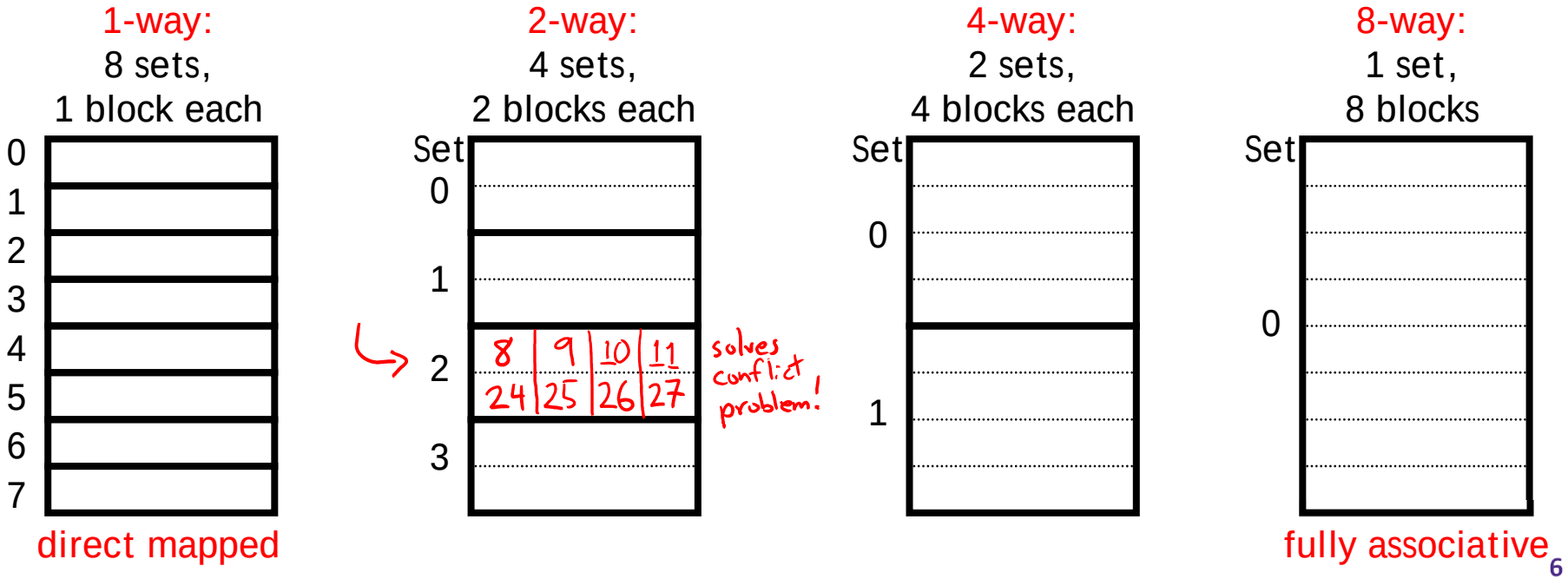
Direct-Mapped Cache Problem



- What happens if we access the following addresses?
  - 8, 24, 8, 24, 8, …?
  - Conflict in cache (misses!)
  - Rest of cache goes *unused*
- Solution?

Associativity

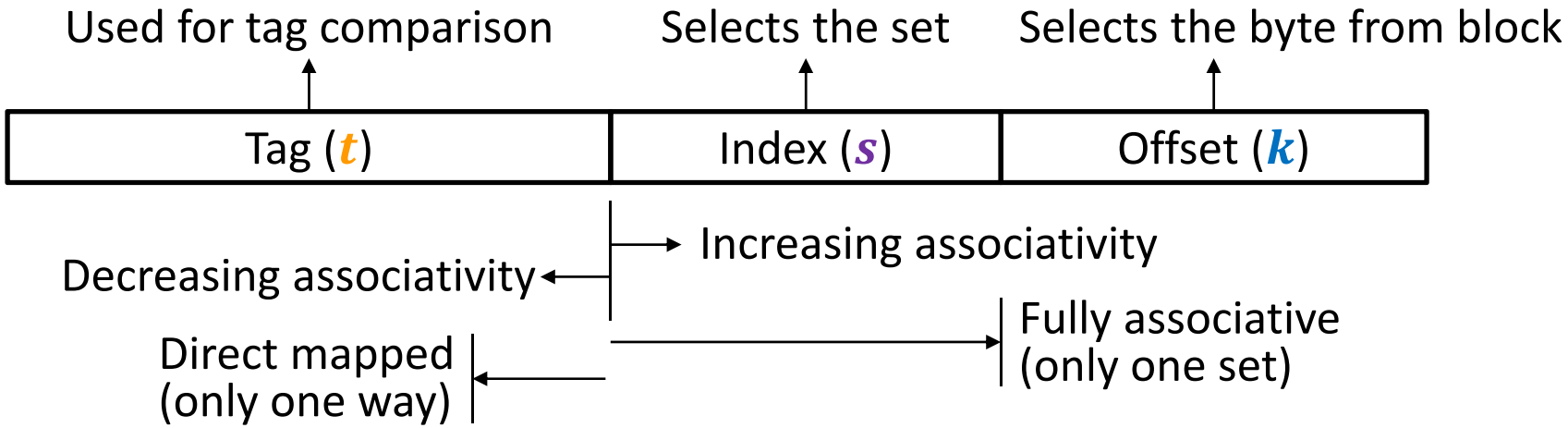
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**



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



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

$t = m - s - k$ $s = \log_2(C/K/E)$ $k = \log_2(K) = 4$ bits

m-bit address:

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

s = ? $\log_2(8/1) = 3$ bits — Direct-mapped (E=1)

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

s = ? $\log_2(8/2) = 2$ bits — 2-way set associative (E=2)

Set	Tag	Data
00 0		
01 1		
10 2		
11 3		

s = ? $\log_2(8/4) = 1$ bit — 4-way set associative (E=4)

Set	Tag	Data
0		
1	—	
	—	
	—	

Block Replacement

- *Any* empty block in the correct set may be used to store block
- 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*")

Peer Instruction Question

- 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 at http://pollev.com/rea

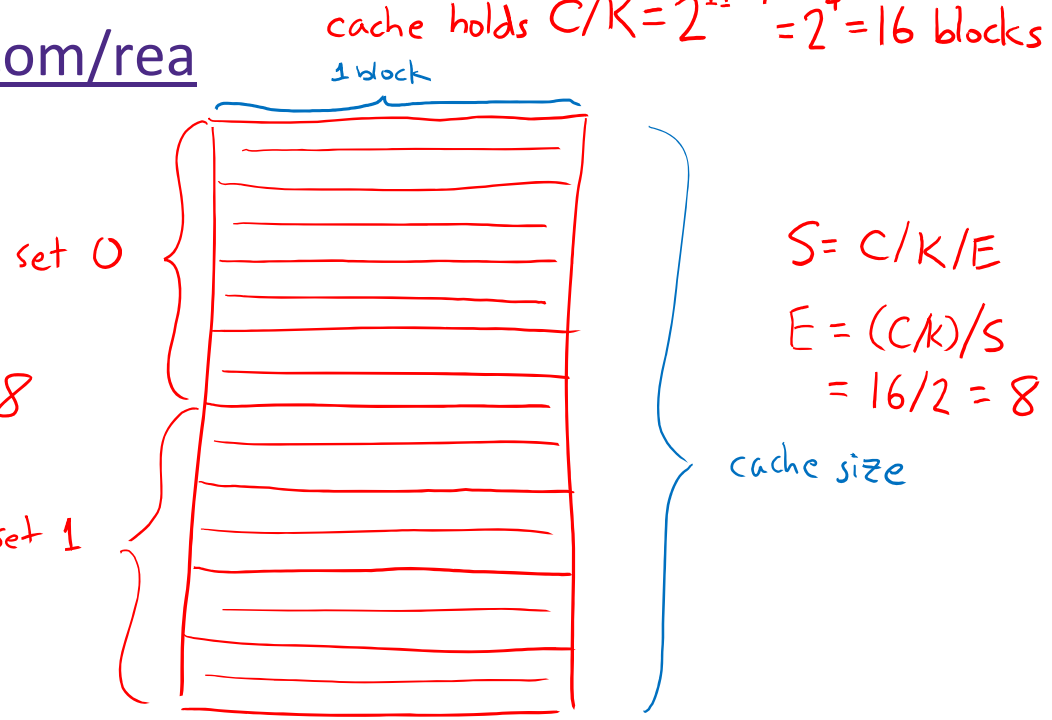
A. 2

B. 4

C. 8

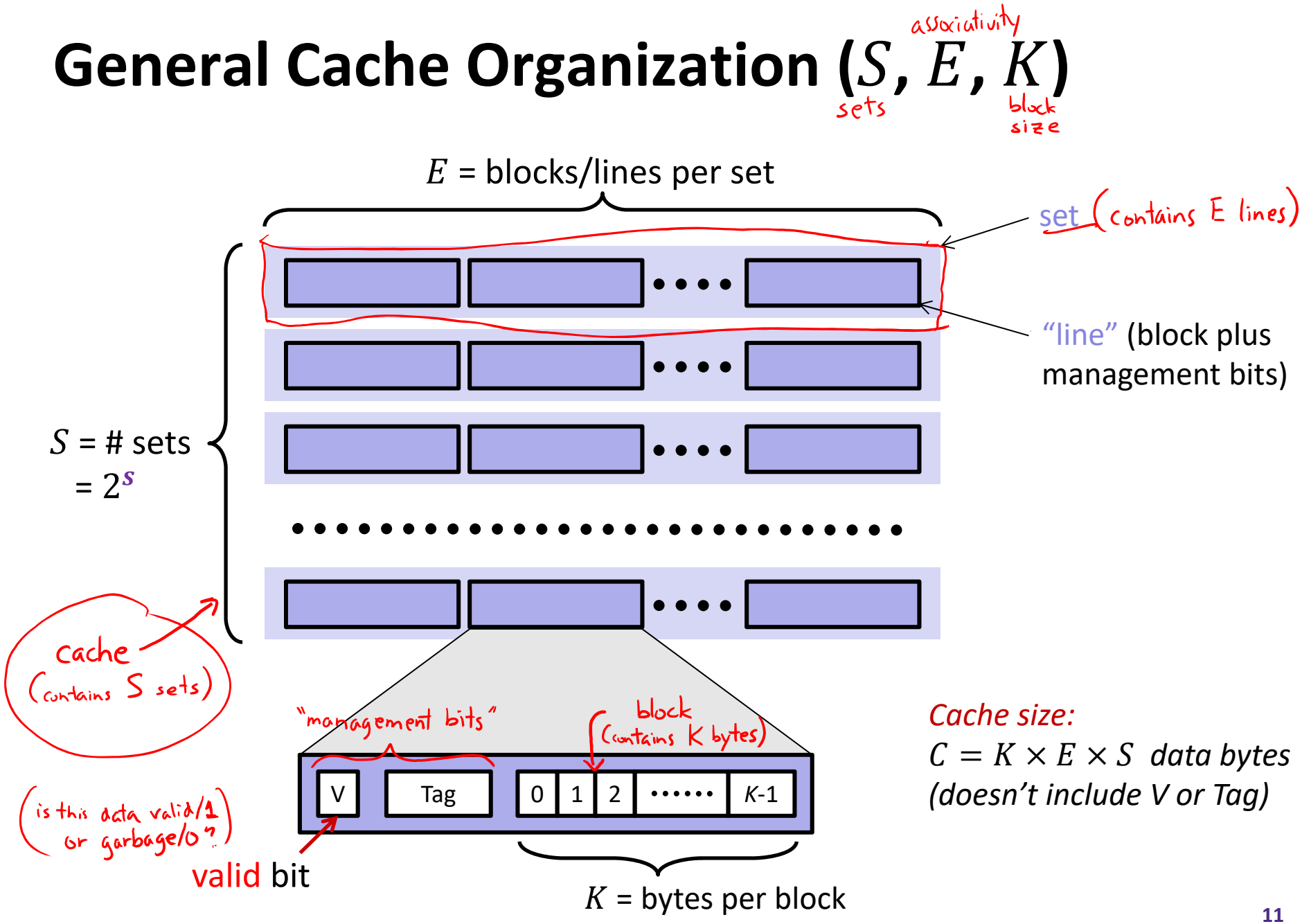
D. 16

E. We're lost…



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

General Cache Organization (S, E, K)



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

Variable	This Quarter	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	$\boldsymbol{m}$	
Tag field width	$\boldsymbol{t}$	
Index field width	$\boldsymbol{s}$	
Offset field width	$\boldsymbol{k}$ ($\boldsymbol{b}$ in book)	

Example Cache Parameters Problem

$\rightarrow 2^{12}$ B $\Longleftrightarrow$ m = 12 bits

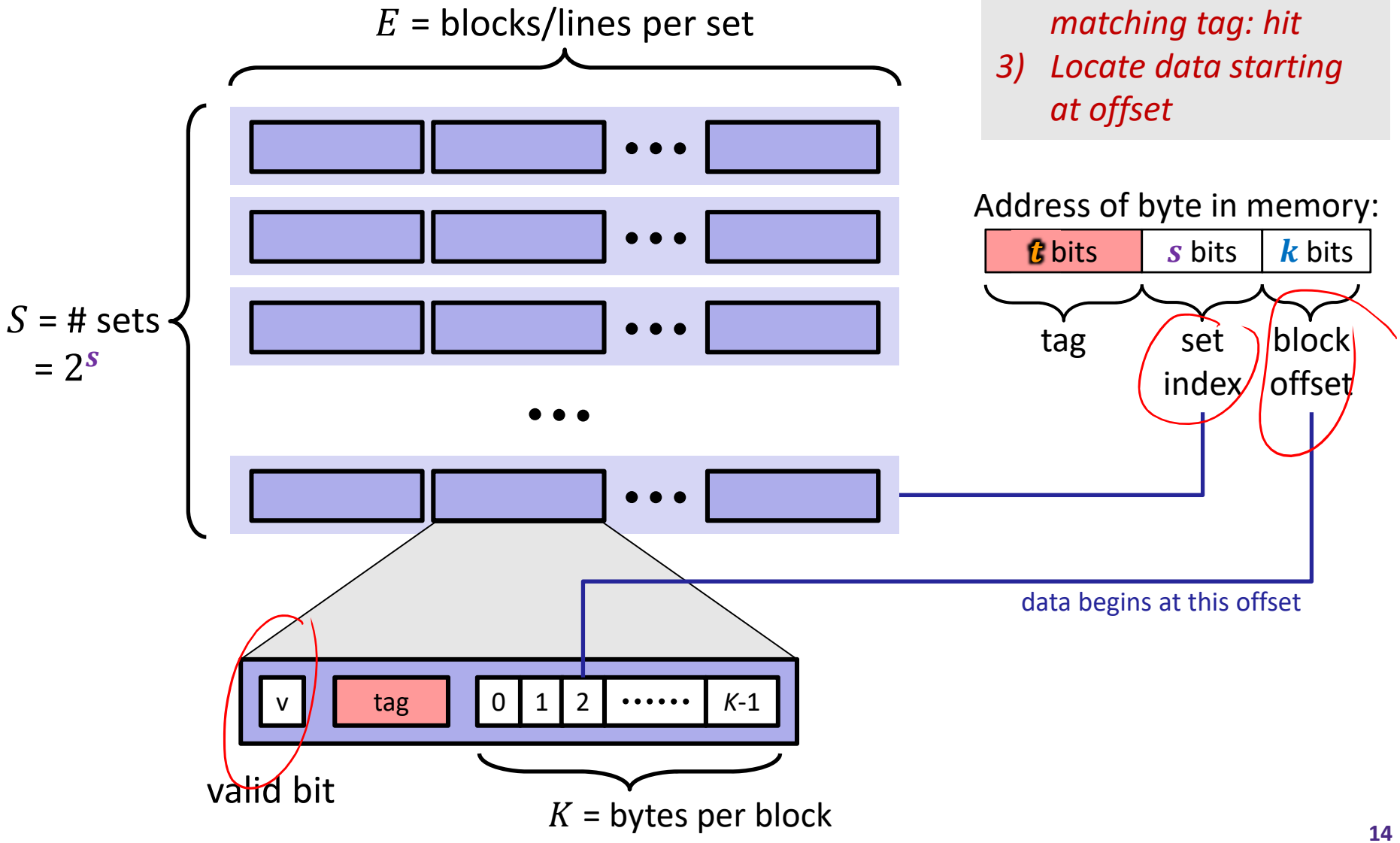
MP

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

C	**Cache Size**	256 B	2^8
K	**Block Size**	32 B	2^5
E	**Associativity**	2-way	2^1
HT	**Hit Time**	3 cycles	
MR	**Miss Rate**	20%	
t = m - s - k	**Tag Bits**	5	
s = $\log_2$(C/K/E)	**Index Bits**	2	$2^8/2^5/2^1$
k = $\log_2$(K)	**Offset Bits**	5	
AMAT = HT + MR * MP	**AMAT**	3 + 0.2(125) = 28 clock cycles	

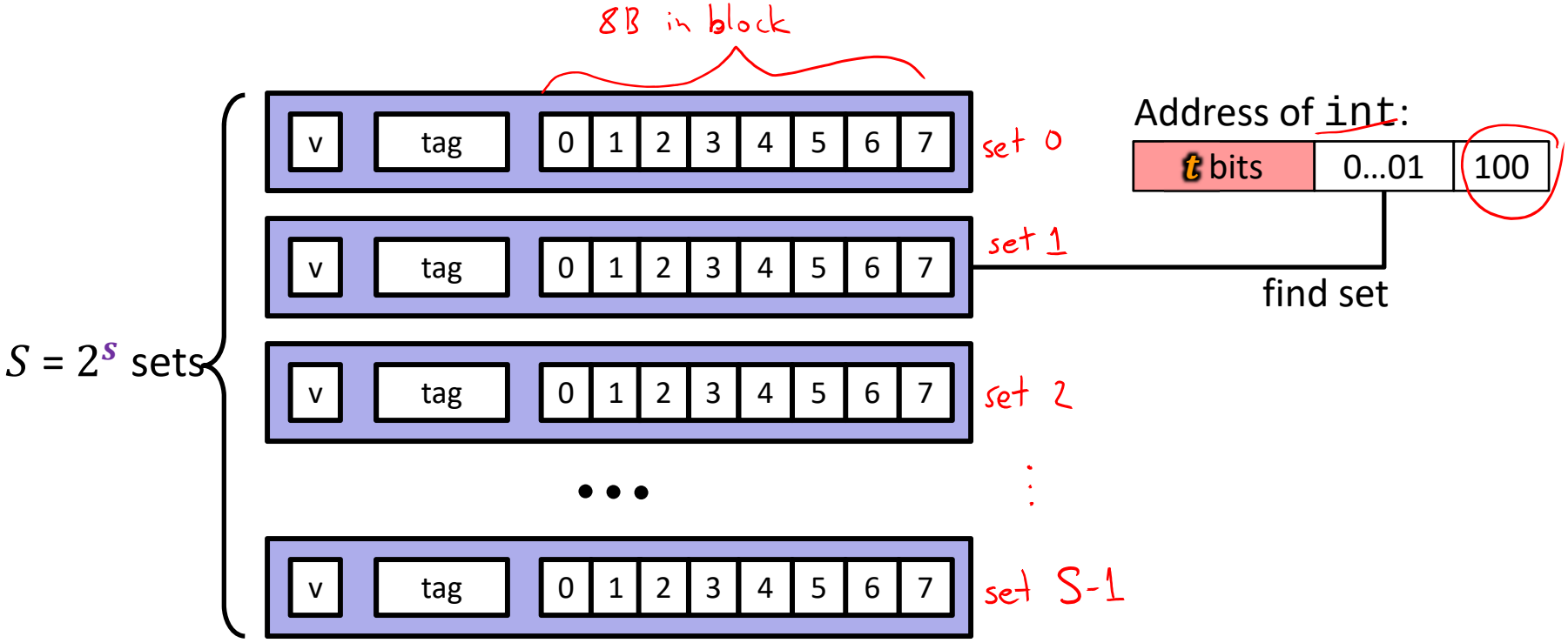
Cache Read

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



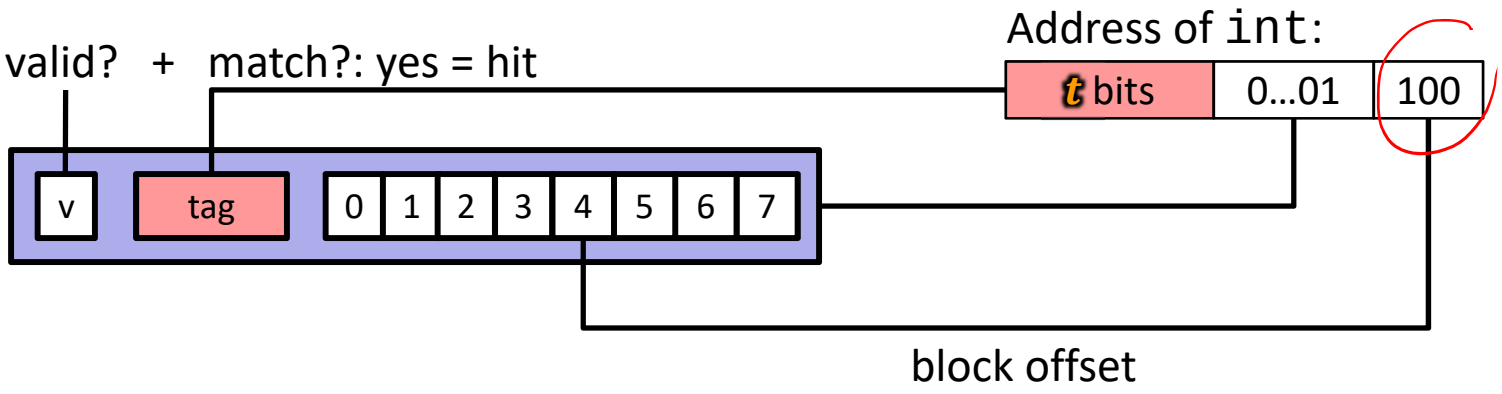
Example: Direct-Mapped Cache (E = 1)

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



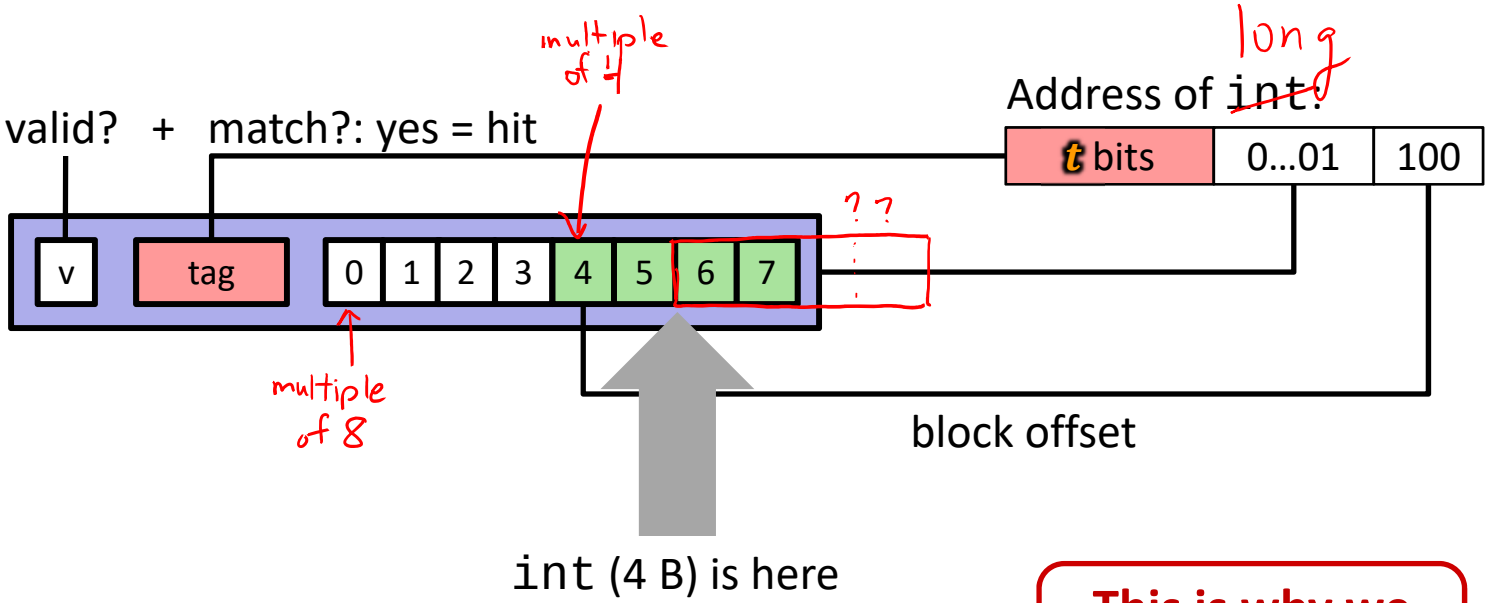
Example: Direct-Mapped Cache (E = 1)

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



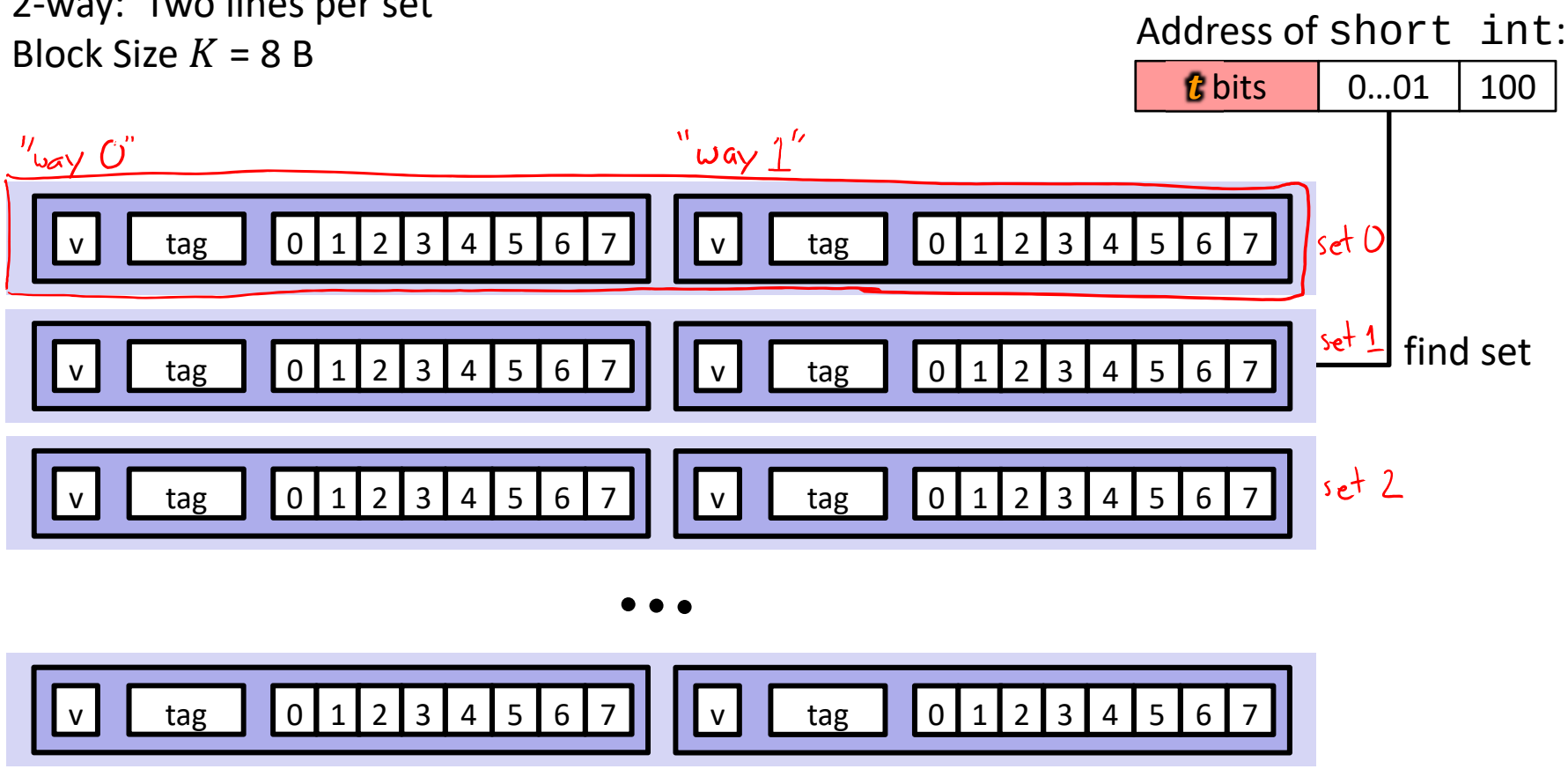
This is why we want alignment!

No match? Then old line gets evicted and replaced

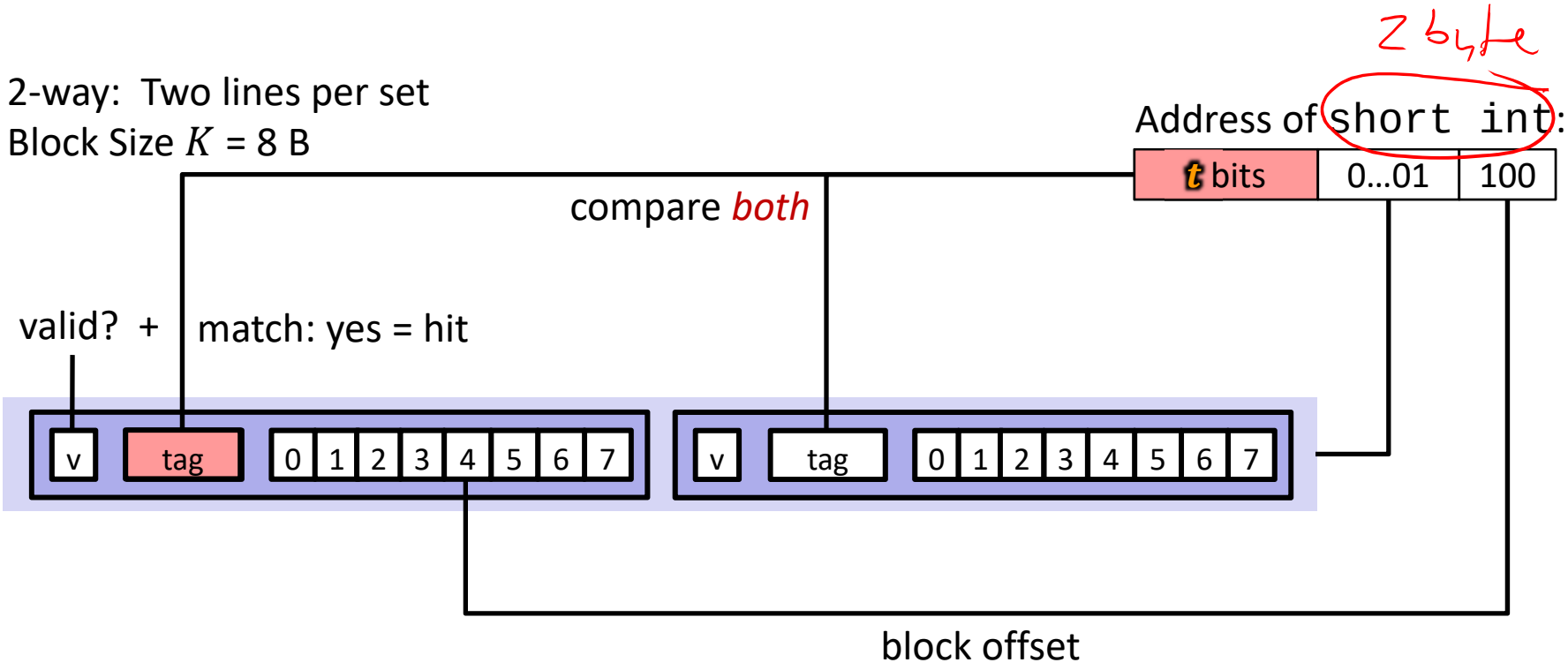
no unnecessary extra cache accesses across block boundaries

Example: Set-Associative Cache (E = 2)

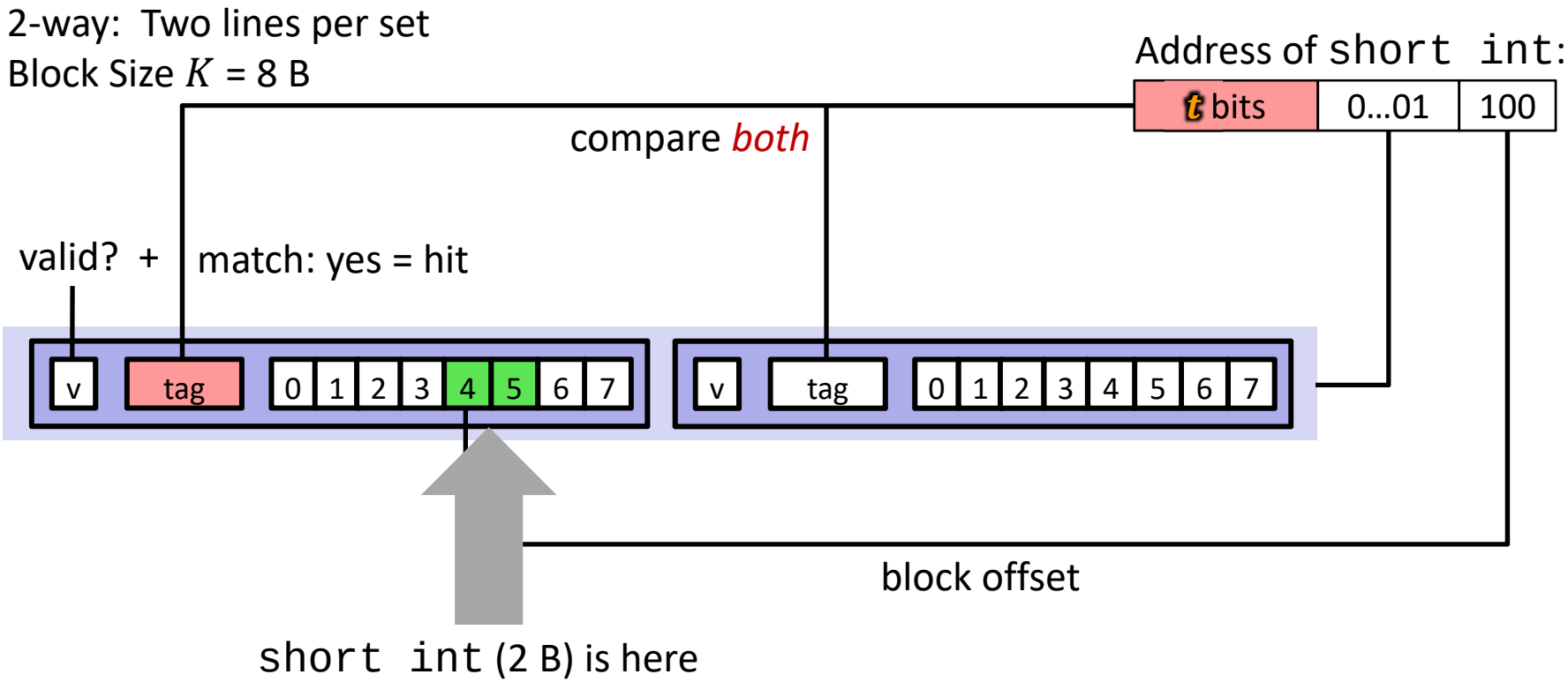
2-way: Two lines per set
Block Size K = 8 B



Example: Set-Associative Cache (E = 2)



Example: Set-Associative Cache (E = 2)



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

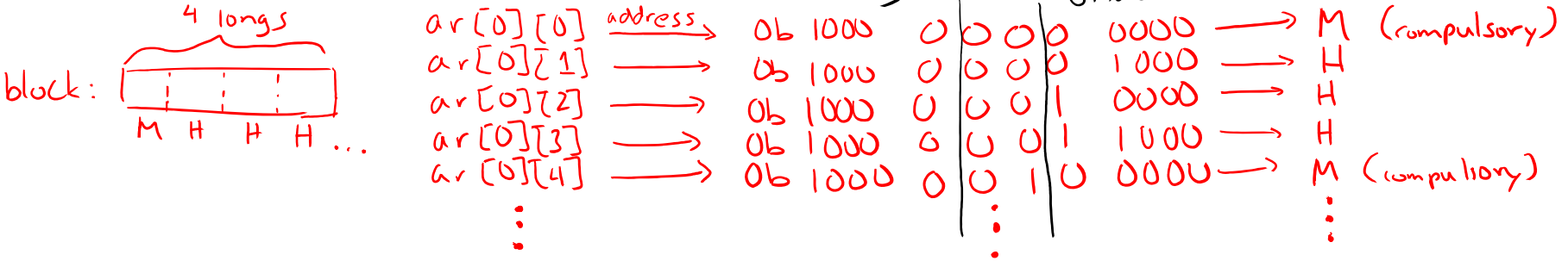
Example Code Analysis Problem

- Assuming the cache starts cold (all blocks invalid) and sum is stored in a register, calculate the **miss rate**: 1/4
  - m = 12 bits, C = 256 B, K = 32 B, E = 2

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

8 bytes per element

```
#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];
```



Challenge: what is the miss rate if we switch the ordering of the for-loops?

What about writes?

- Multiple copies of data exist:
  - L1, L2, possibly L3, main memory
- What to do on a write-hit? (block/data already in $)
  - Write-through: write immediately to next level
  - Write-back: defer write to next level until line is evicted (replaced)
    - Must track which cache lines have been modified ("*dirty bit*") ← extra management bit only for write-back cache
- What to do on a write-miss? (block/data not currently in $)
  - Write-allocate: ("fetch on write") load into cache, update line in cache
    - Good if more writes or reads to the location follow
  - No-write-allocate: ("write around") just write immediately to ~~memory~~ next level
- Typical caches:
  - Write-back + Write-allocate, usually ☆
  - Write-through + No-write-allocate, occasionally