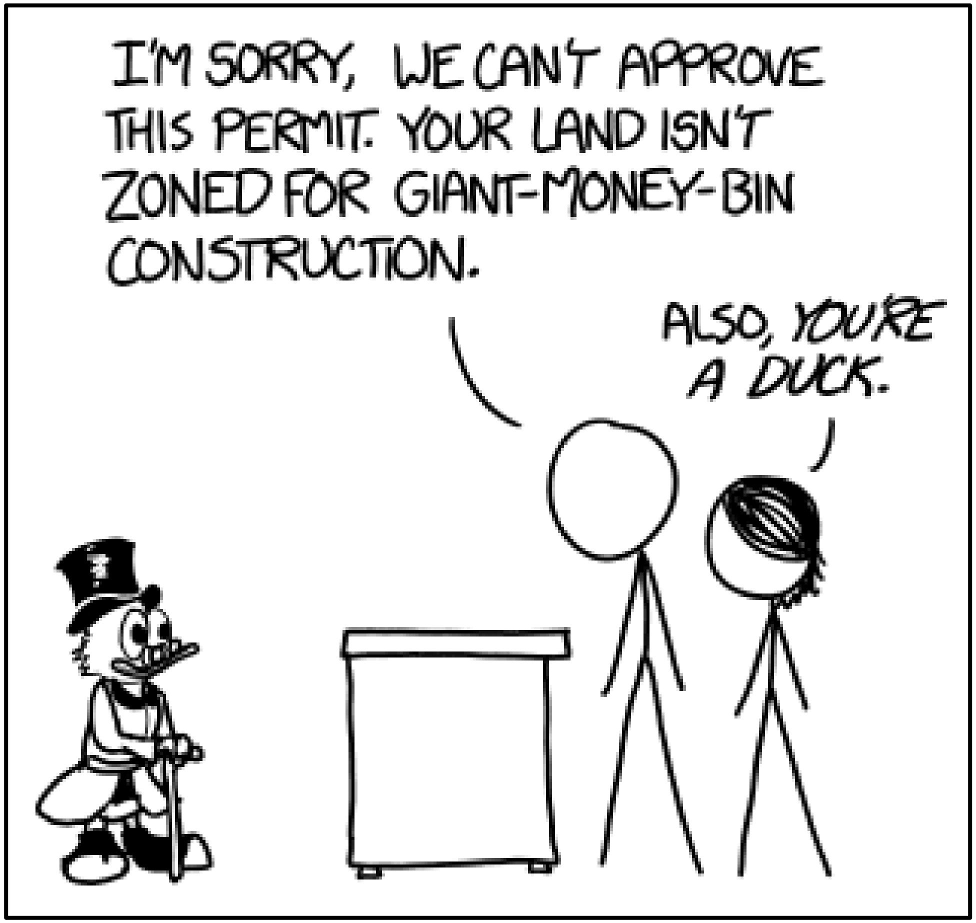
Caches III

CSE 351 Autumn 2024

Instructor:
Ruth Anderson

Teaching Assistants:
Alexandra Michael
Connie Chen
Chloe Fong
Chendur Jayavelu
Joshua Tan
Nikolas McNamee
Nahush Shrivatsa
Naama Amiel
Neela Kausik
Renee Ruan
Rubee Zhao
Samantha Dreussi
Sean Siddens
Waleed Yagoub



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

Relevant Course Information

- HW16 due TONIGHT, Wednesday (11/06) @ 11:59 pm
- Lab 3 due Mon 11/11
  - Encouraged to aim for Fri 11/08, actual deadline Mon 11/11
  - You have everything you need to do the lab as of 10/28
  - Last part of HW15 is useful for Lab 3
- HW17 due Friday (11/08) @ 11:59 pm
- Mid-quarter Survey due Saturday (11/09)
- HW18 due Wednesday (11/13) @ 11:59 pm

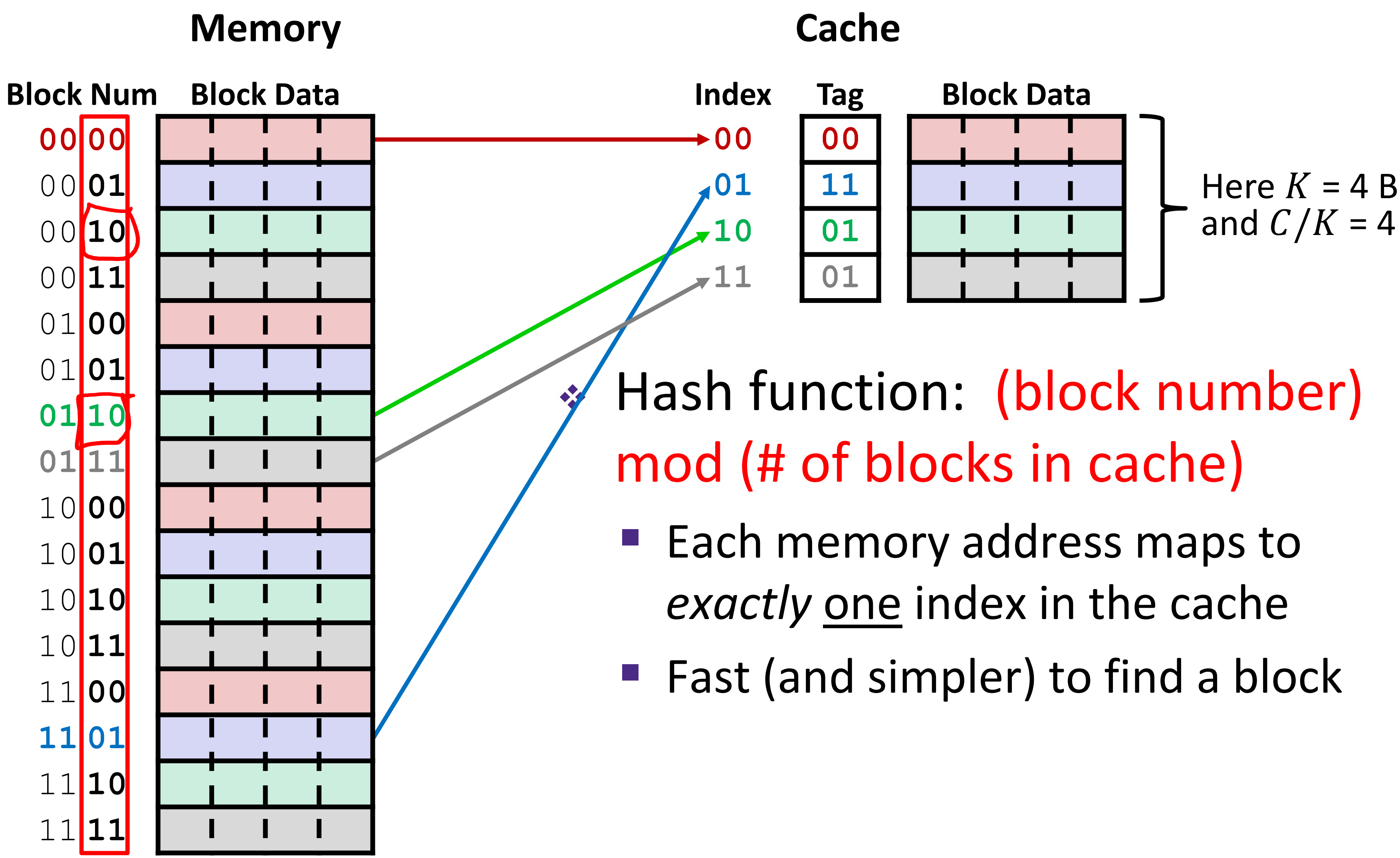
Making memory accesses fast!

- 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

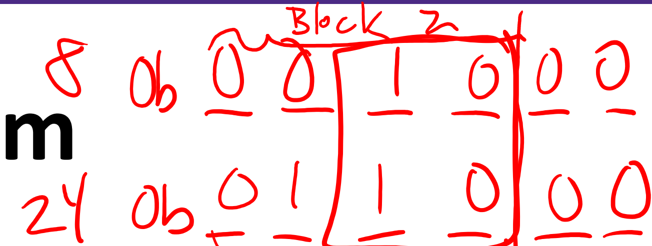
Review: Direct-Mapped Cache

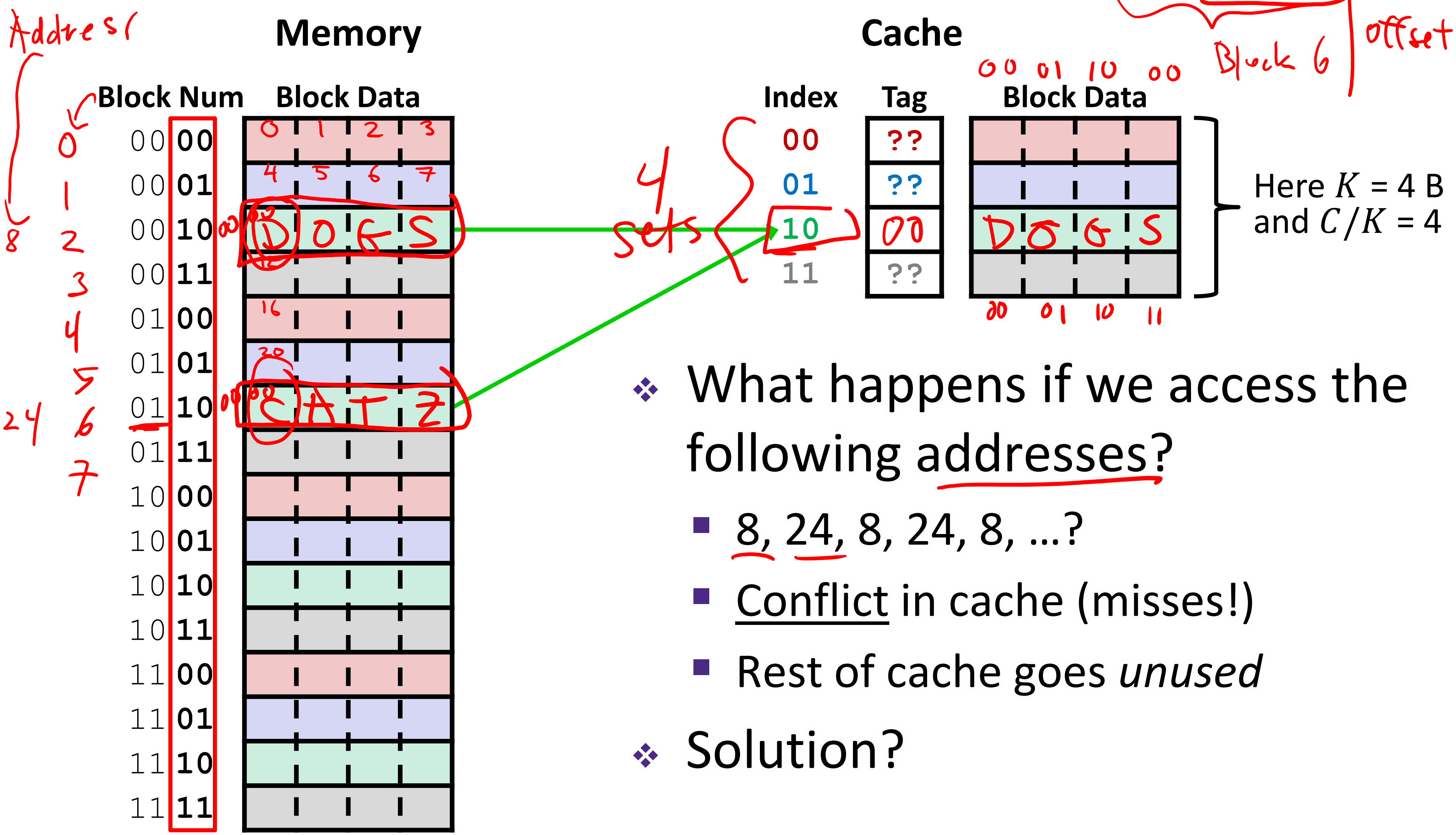


Hash function: (block number) mod (# of blocks in cache)

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

Direct-Mapped Cache Problem

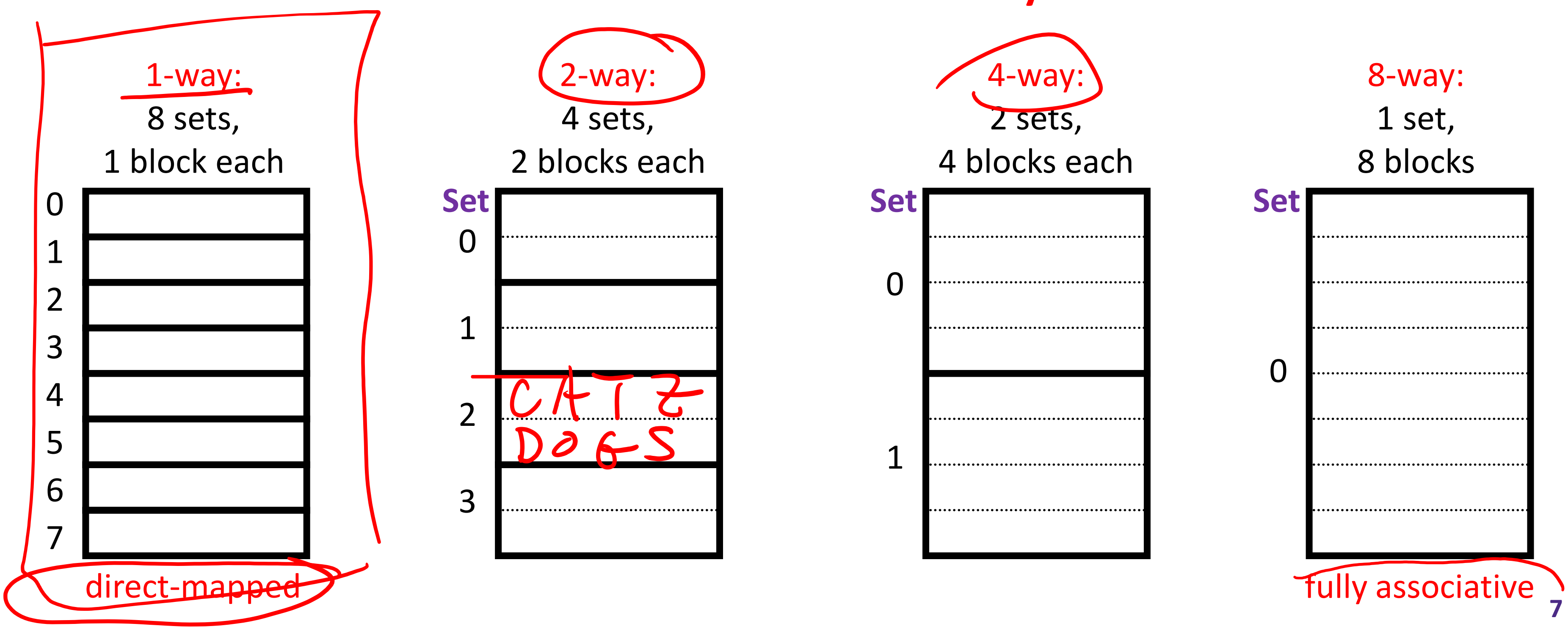




- What happens if we access the following addresses?
  - 8, 24, 8, 24, 8, …?
  - Conflict in cache (misses!)
  - Rest of cache goes *unused*
- Solution?

Associativity: A Solution!

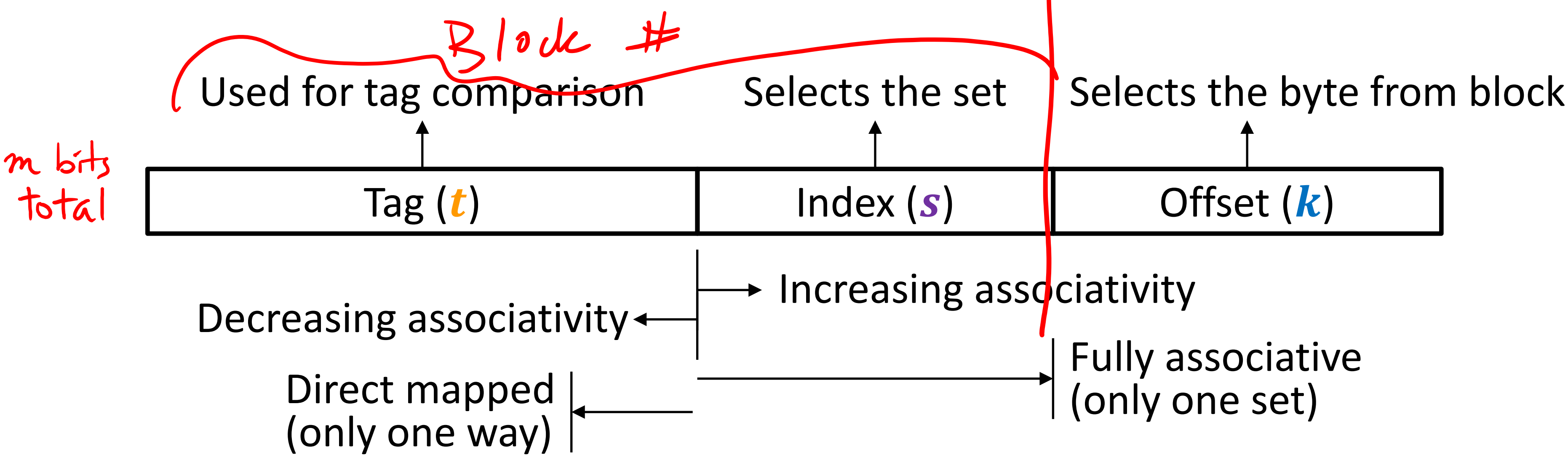
- What if we could store any data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way** within the set



Cache Associativity (E)

Note: The textbook uses "b" for offset bits

- Associativity (E): number of ways to store in 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`

$I_{E=1}$ $I_{E=4}$ $I_{E=2}$ offset

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

m-bit address:

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

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		✓
		✓

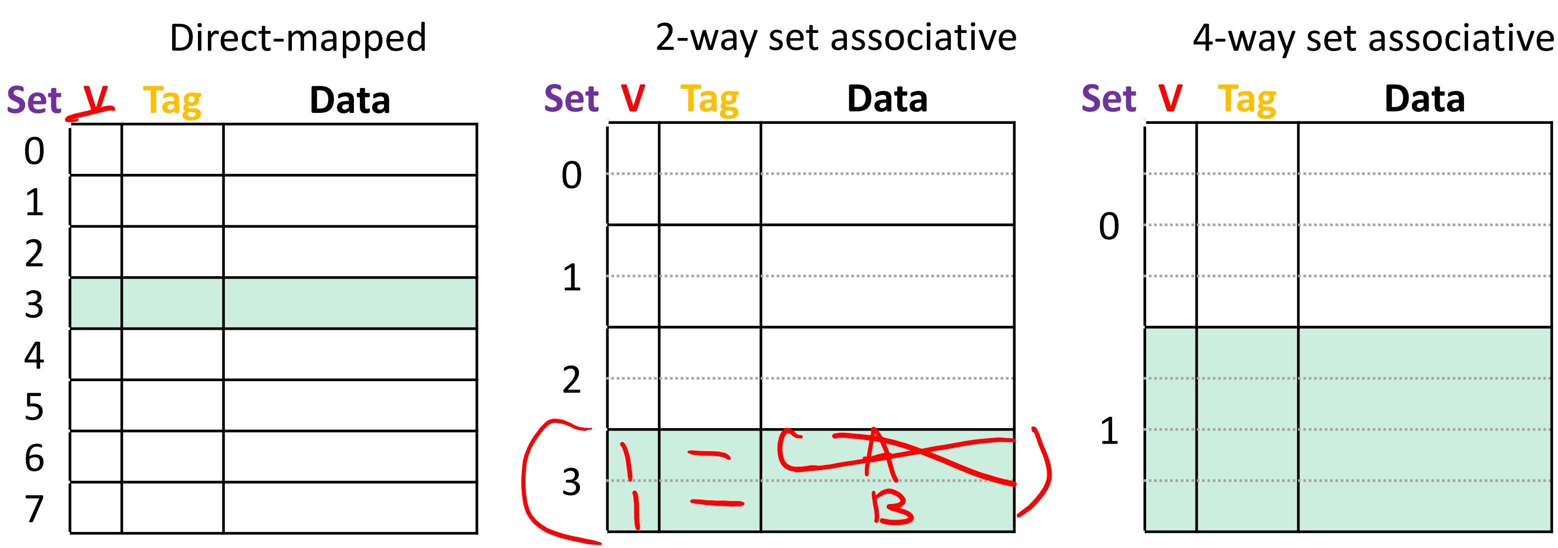
s = ?
4-way set associative

Set	Tag	Data
(0) 0		
(1) 1		✓
		✓
		✓
		✓

Block Placement and Replacement

A, B, C

- *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 mystery (0) data
- If there are no empty blocks, which one should we <u>replace</u>?
  - No choice for direct-mapped caches
  - Caches typically use something close to ***<u>least recently used (LRU)</u>*** (hardware usually implements "*not most recently used*")



Polling Questions

$K = 2^7$ B

$C = 2^{11}$ B

- 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

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

A. 2

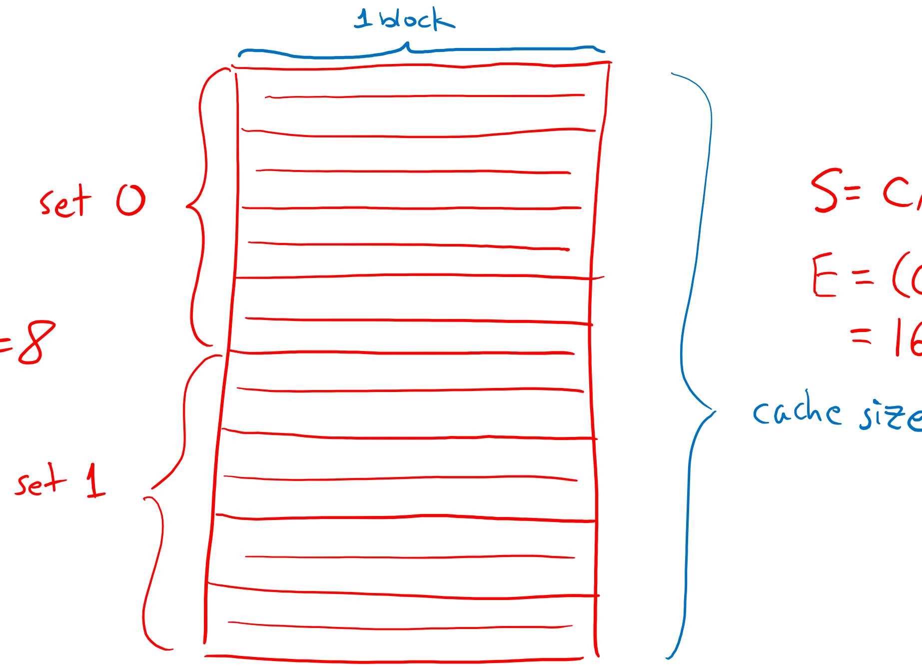
B. 4

C. $\boxed{8}$

D. 16

E. We're lost…

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



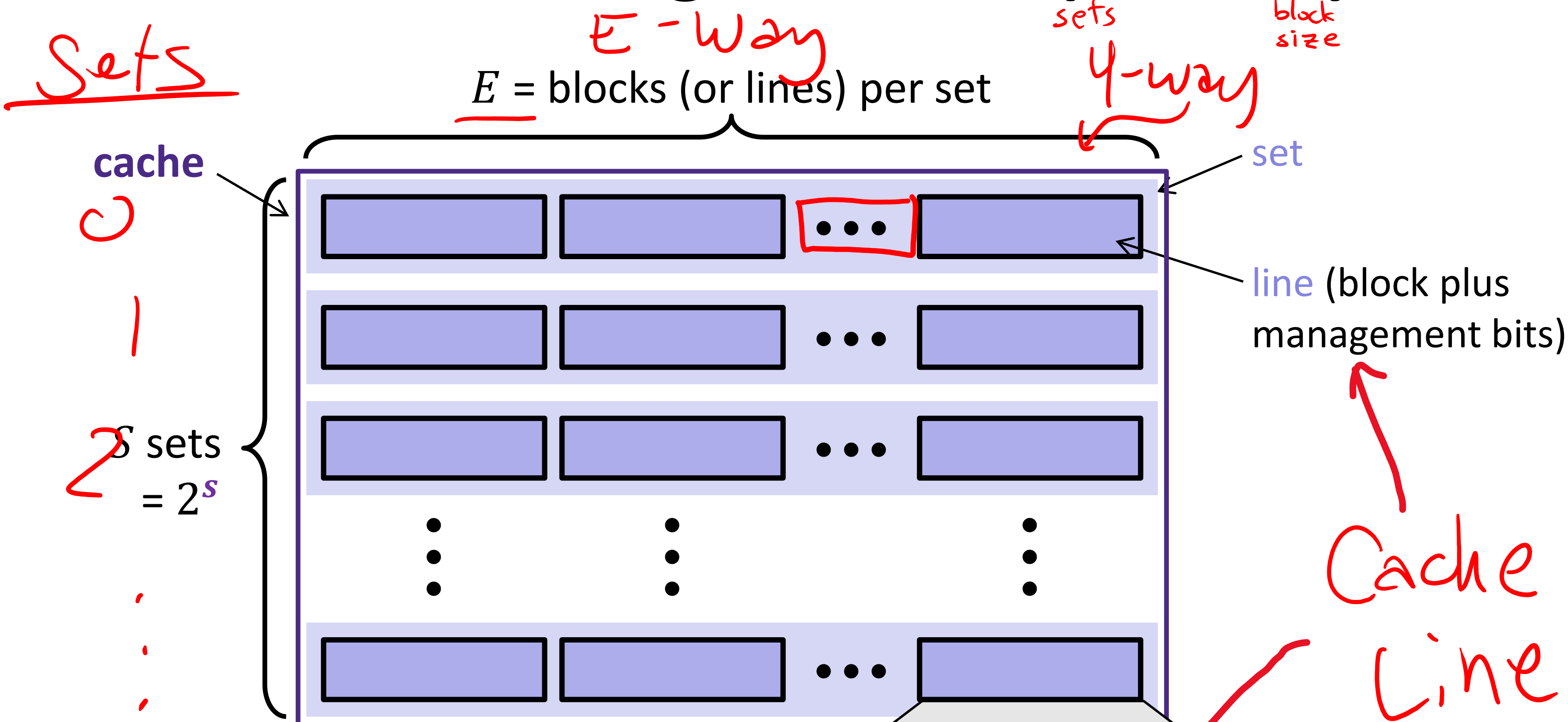
$S = C/K/E$

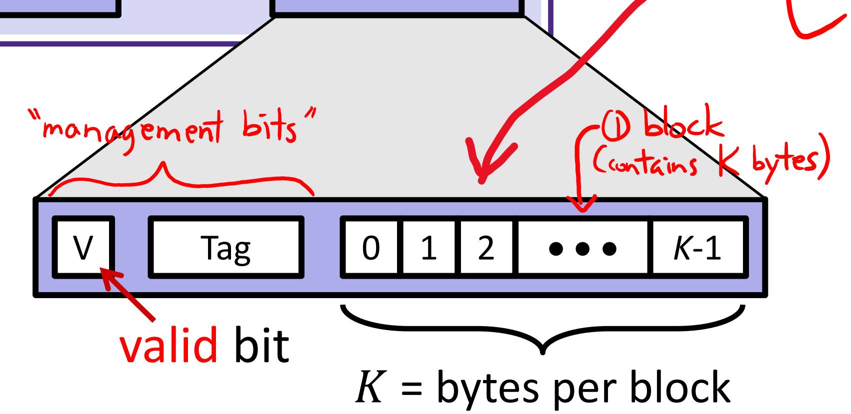
$E = (C/K)/S$
$= 16/2 = 8$

$m = 16$

- 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)





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)	$M = 2^{\boldsymbol{m}} \leftrightarrow \boldsymbol{m} = \log_2 M$ $S = 2^{\boldsymbol{s}} \leftrightarrow \boldsymbol{s} = \log_2 S$ $K = 2^{\boldsymbol{k}} \leftrightarrow \boldsymbol{k} = \log_2 K$ $C = K \times E \times S$ $\boldsymbol{s} = \log_2(C/K/E)$ $\boldsymbol{m} = \boldsymbol{t} + \boldsymbol{s} + \boldsymbol{k}$
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)	

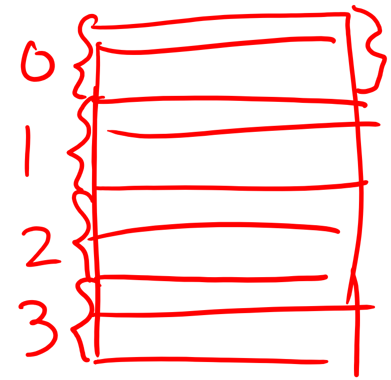
Example Cache Parameters Problem

2^{10} Bytes ⟷ 10 bits Address is 10 bits wide

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

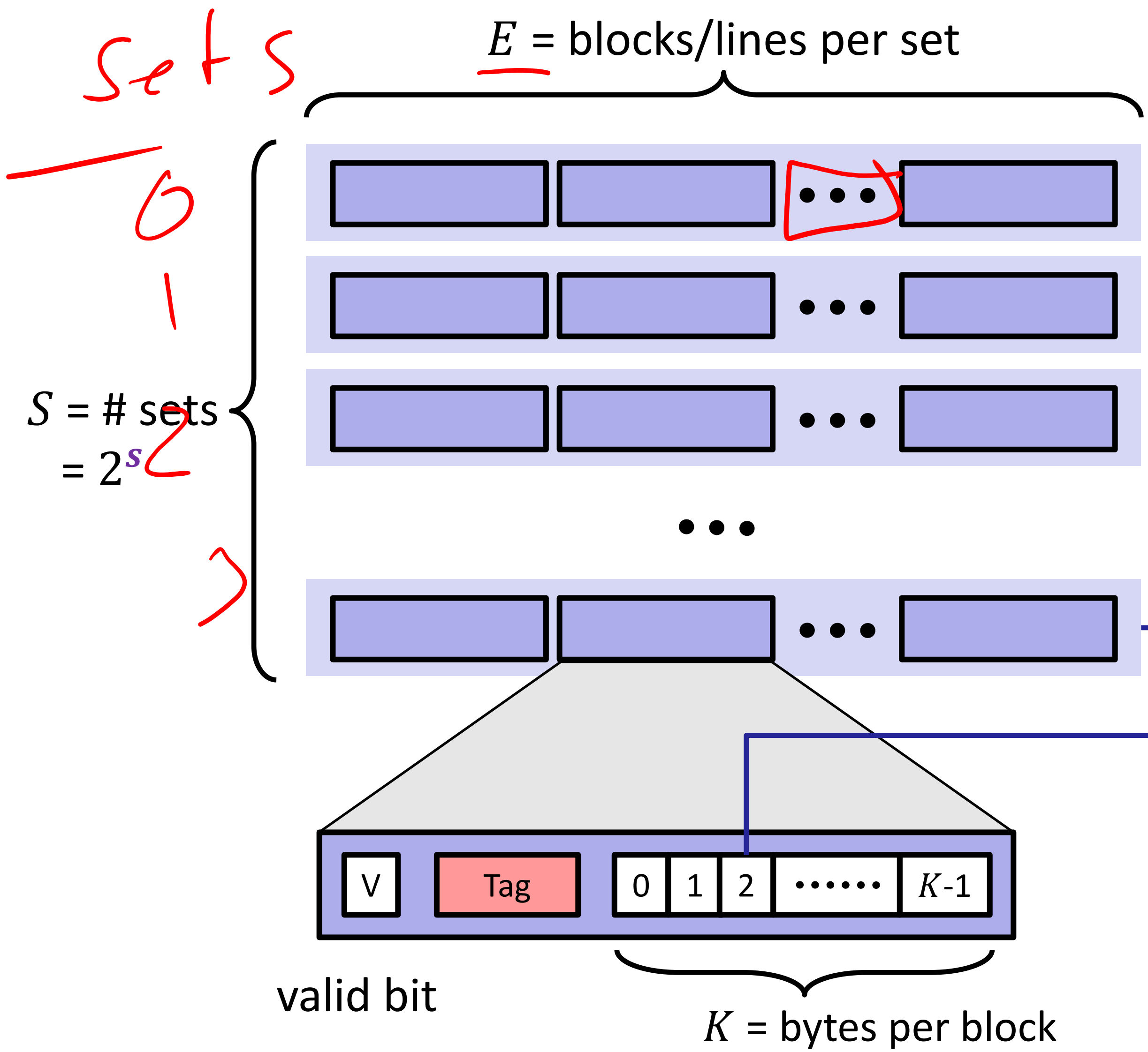
$\frac{2^6}{2^3} = 2^3$ blocks

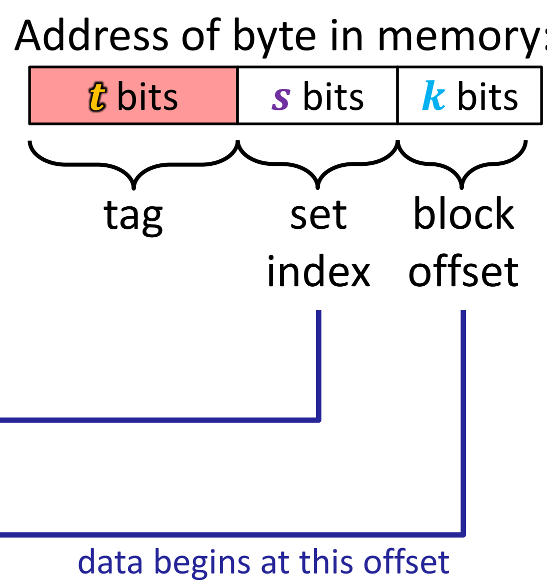
C	**Cache Size**	64 B	2^6 Bytes
K	**Block Size**	8 B	2^3 Bytes
E	**Associativity**	2-way	
HT	**Hit Time**	3 cycles	
MR	**Miss Rate**	20%	
$t = m - s - k$	**Tag Bits**	$10-3-2 = 5$	
$s = \log_2(C/K/E)$	**Index Bits**	2	
$k = \log_2(K)$	**Offset Bits**	3	
$AMAT = HT + MR * MP$	**AMAT**	$3 + 0.2(125) = 28$ clock cycles	



Cache Read

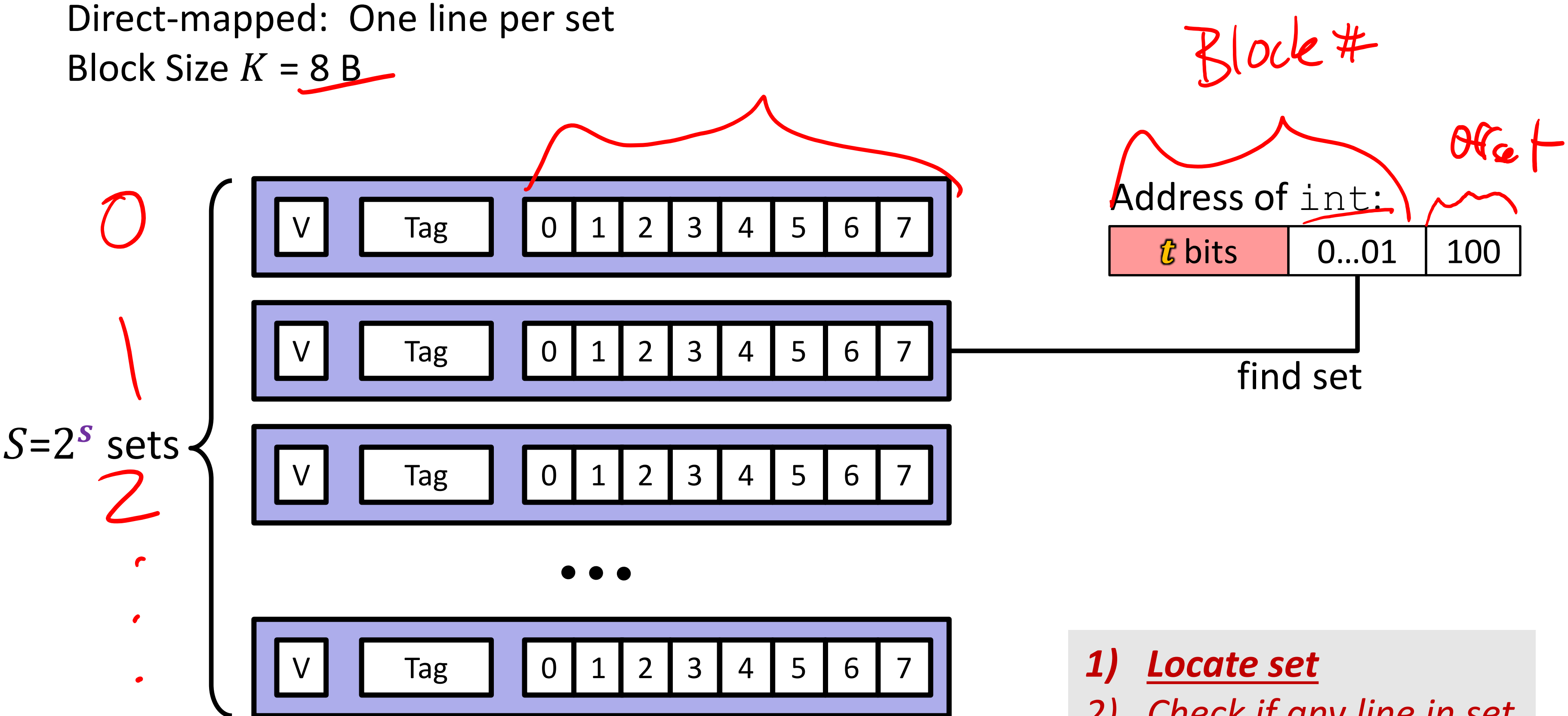
1) *Locate set*
2) *Check if any line in set is valid and has matching tag: hit*
3) *Locate data starting at offset*





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

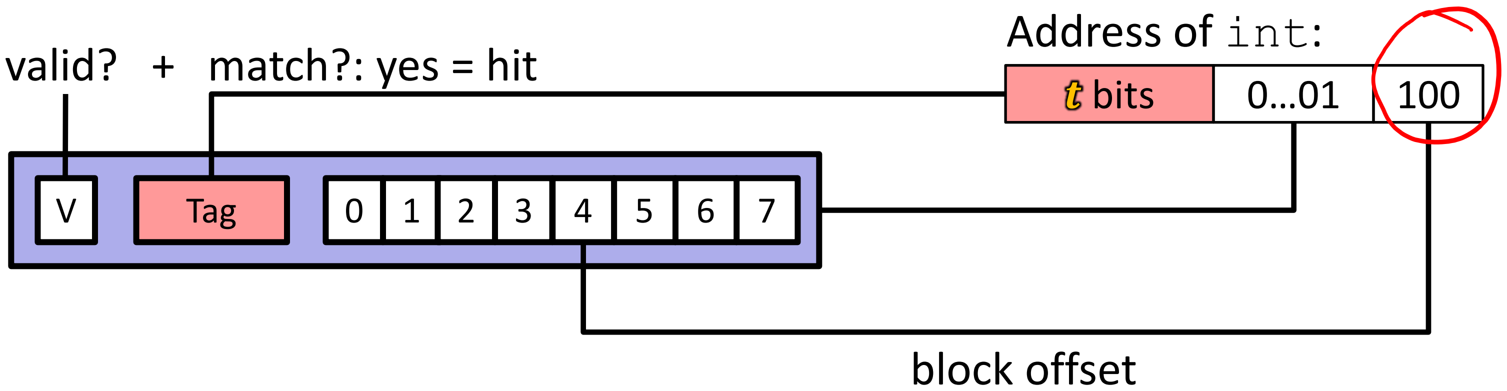
Direct-mapped: One line per set
Block Size K = 8 B



1) ***<u>Locate set</u>***
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) (step 2)

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

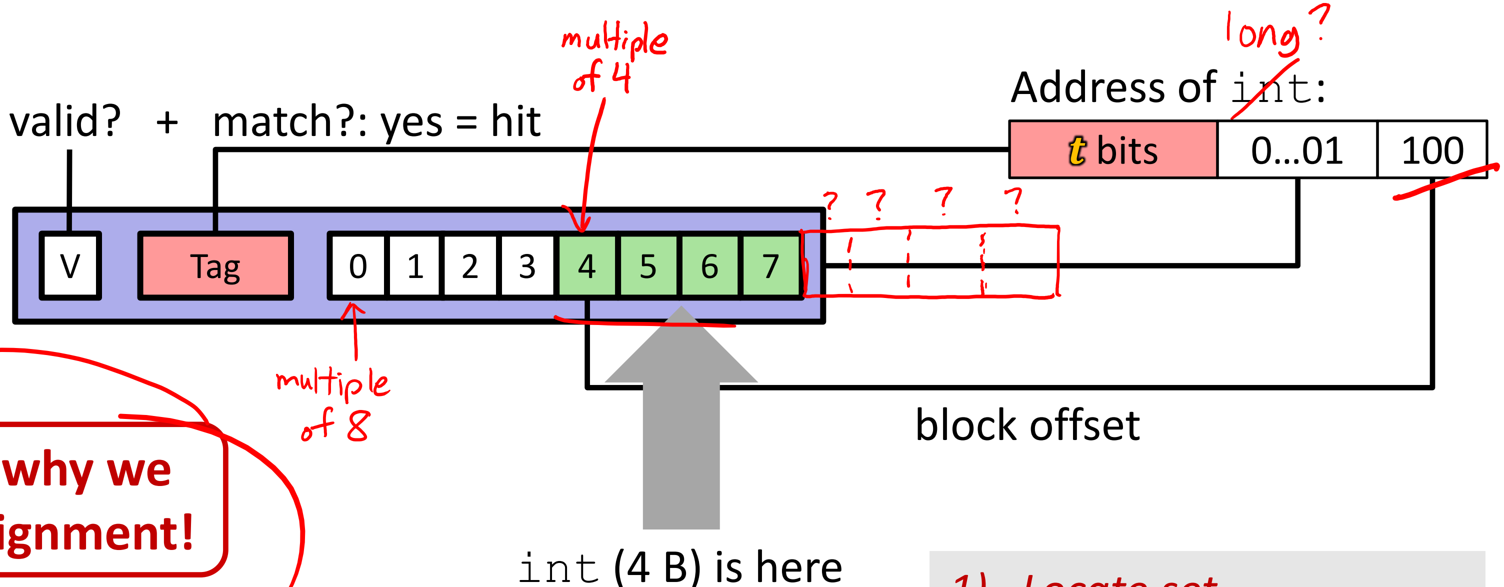


1) *Locate set*
2) ***Check if any line in set is valid and has matching tag: hit***
3) *Locate data starting at offset*

Example: Direct-Mapped Cache (E = 1) (step 3)

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

movl (~), %eax



This is why we want alignment!

No match? Then old line gets evicted and replaced

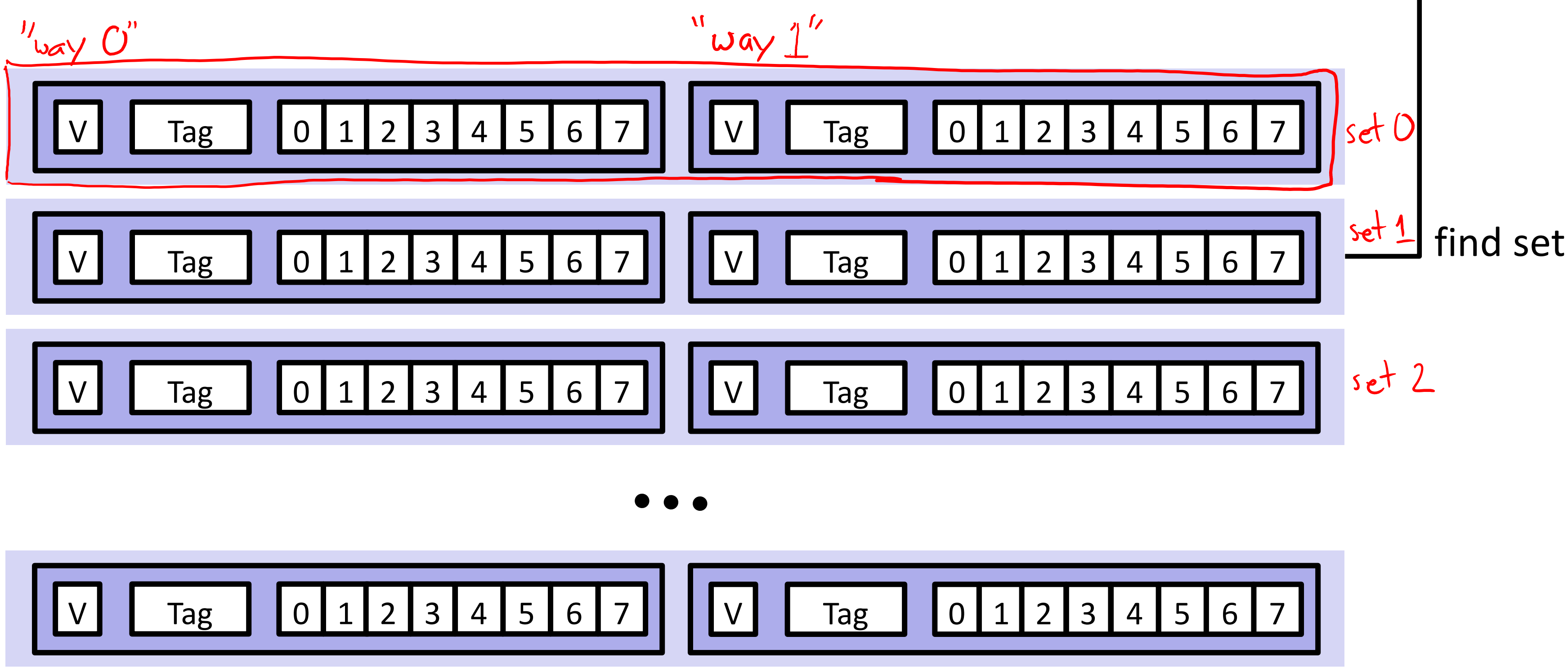
1) *Locate set*
2) *Check if any line in set is valid and has matching tag: hit*
3) ***Locate data starting at offset***

Example: Set-Associative Cache (E = 2) (step 1)

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

Address of `short int`:

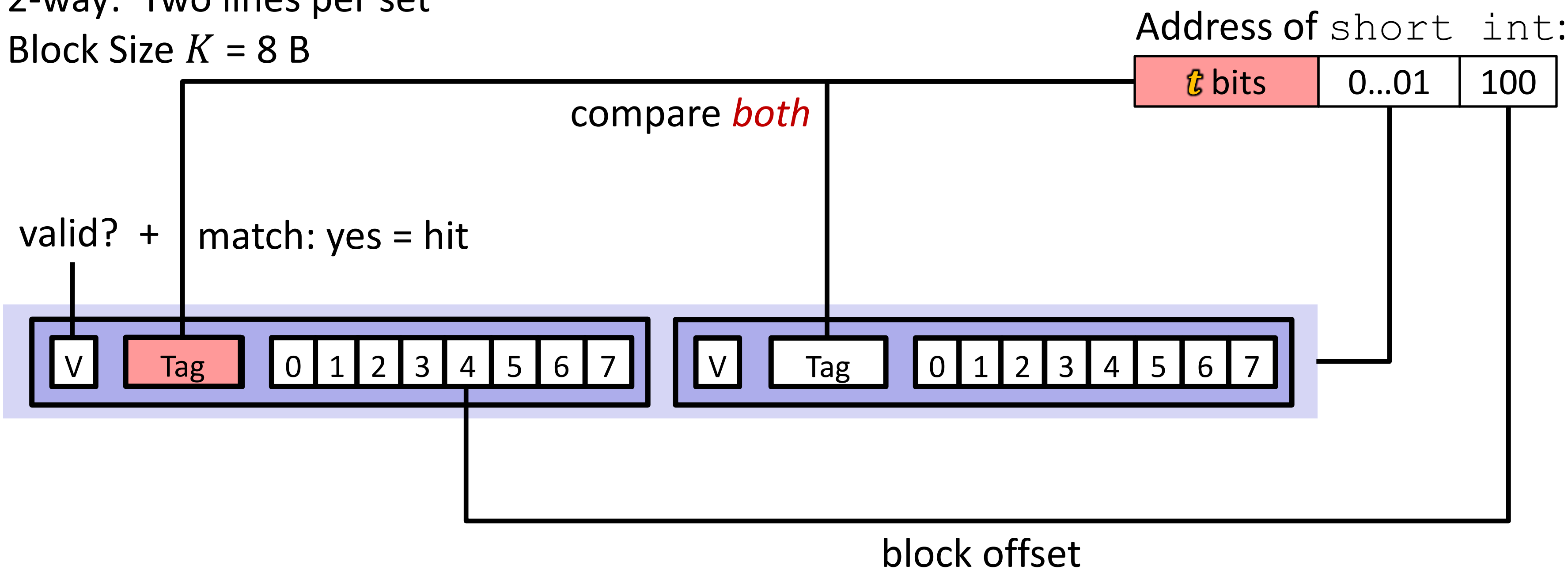
t bits	0…01	100



1) Locate set

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

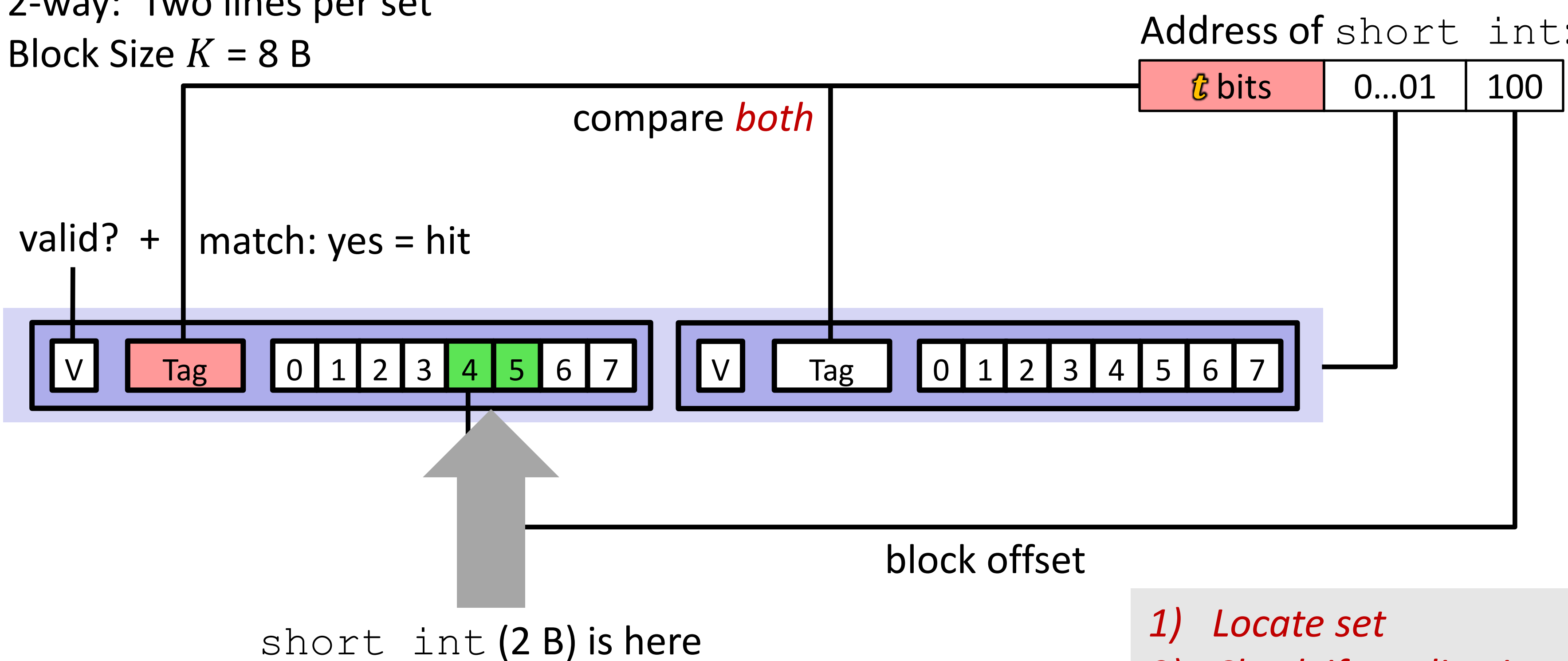
2-way: Two lines per set
Block Size K = 8 B



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

Example: Set-Associative Cache (E = 2) (step 3)

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), …

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

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**: 25%
  - t = 5 bits, s = 2 bits, k = 5 bits
  - m = 12 bits, C = 256 B, K = 32 B, E = 2

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];
```

4 longs

block: M H H H

address	tag	index	offset	
ar[0][0]	0b 1000 0	0 0	0 0000	M
ar[0][1]	0b 1000 0	0 0	0 1000	H
ar[0][2]	0b 1000 0	0 0	1 0000	H
ar[0][3]	0b 1000 0	0 0	1 1000	H
ar[0][4]	0b 1000 0	0 1	0 0000	M
⋮		⋮		H
				H
				H

Slightly more complex example posted in the video link