Memory & Caches III

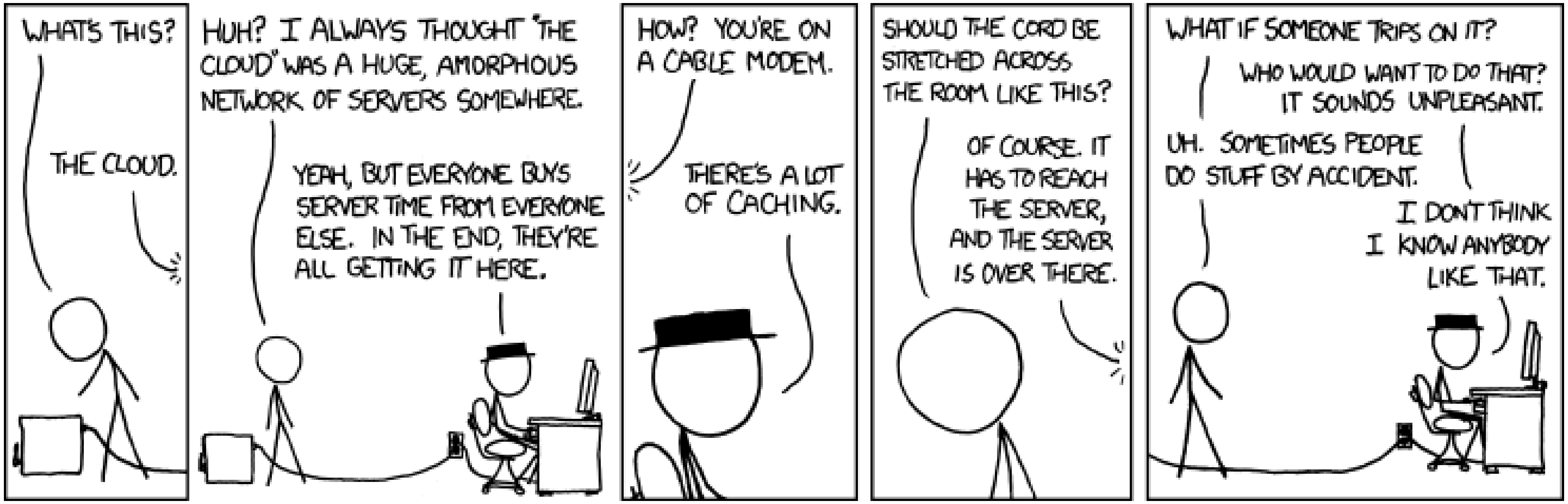
CSE 351 Winter 2021

Instructor:

Mark Wyse

Teaching Assistants:

Kyrie Dowling   Catherine Guevara   Ian Hsiao
Jim Limprasert   Armin Magness   Allie Pfleger
Cosmo Wang   Ronald Widjaja


WHAT'S THIS?
THE CLOUD.
HUH? I ALWAYS THOUGHT "THE CLOUD" WAS A HUGE, AMORPHOUS NETWORK OF SERVERS SOMEWHERE.
YEAH, BUT EVERYONE BUYS SERVER TIME FROM EVERYONE ELSE. IN THE END, THEY'RE ALL GETTING IT HERE.
HOW? YOU'RE ON A CABLE MODEM.
THERE'S A LOT OF CACHING.
SHOULD THE CORD BE STRETCHED ACROSS THE ROOM LIKE THIS?
OF COURSE. IT HAS TO REACH THE SERVER, AND THE SERVER IS OVER THERE.
WHAT IF SOMEONE TRIPS ON IT?
WHO WOULD WANT TO DO THAT? IT SOUNDS UNPLEASANT.
UH. SOMETIMES PEOPLE DO STUFF BY ACCIDENT.
I DON'T THINK I KNOW ANYBODY LIKE THAT.

http://xkcd.com/908/

Administrivia

- Lab 3 due Monday 2/22

- hw16 due Monday (2/22)
  - Covers the major cache mechanics
- hw17 due Wednesday (2/24)
  - Preparation for Lab 4

- Cache simulator:
  - https://courses.cs.washington.edu/courses/cse351/cachesim/

Making memory accesses fast!

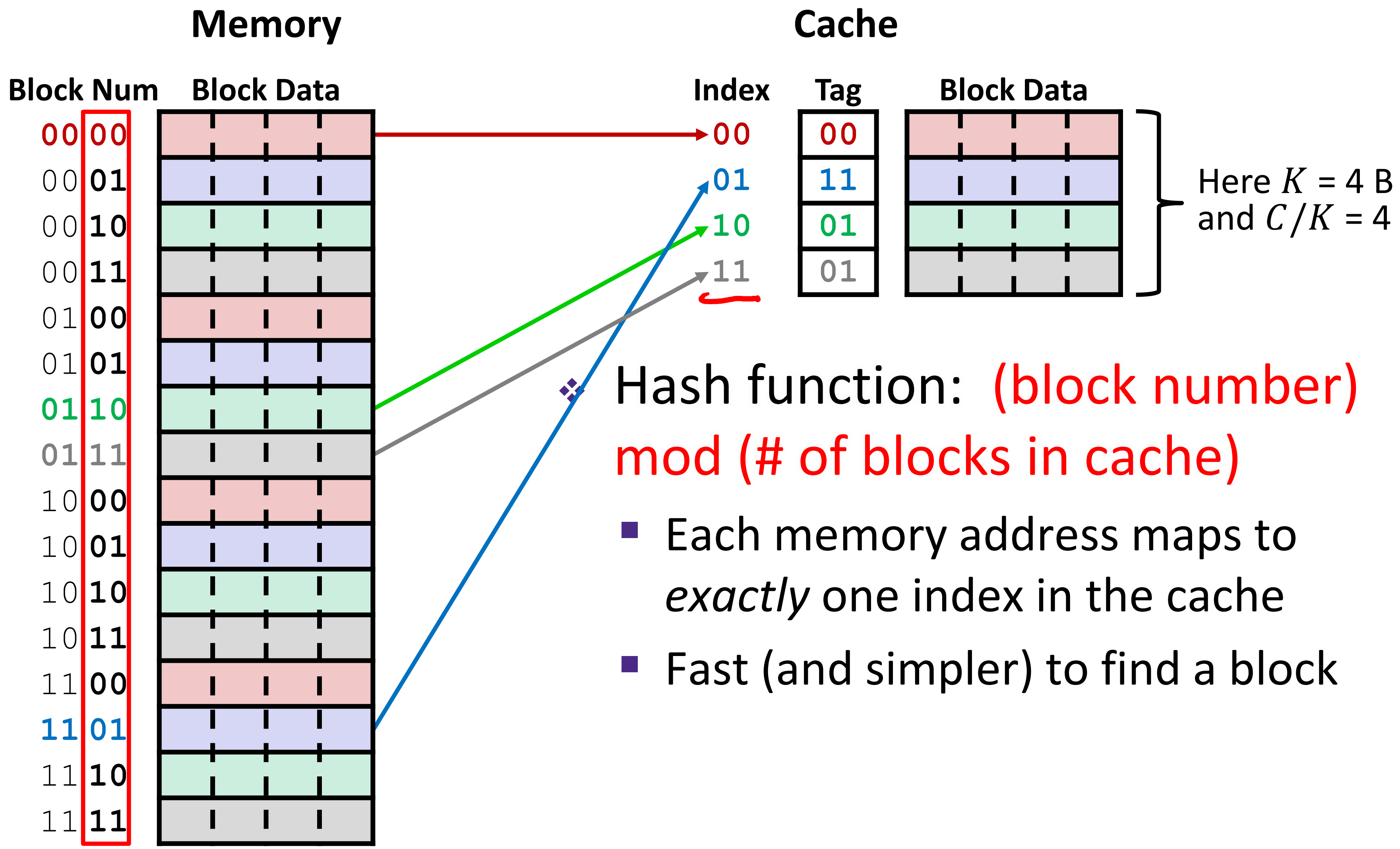
- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
  - Direct-mapped (*sets*; index + tag)
  - **Associativity (*ways*)**
  - **Replacement policy**
  - Handling writes
- Program optimizations that consider caches

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?

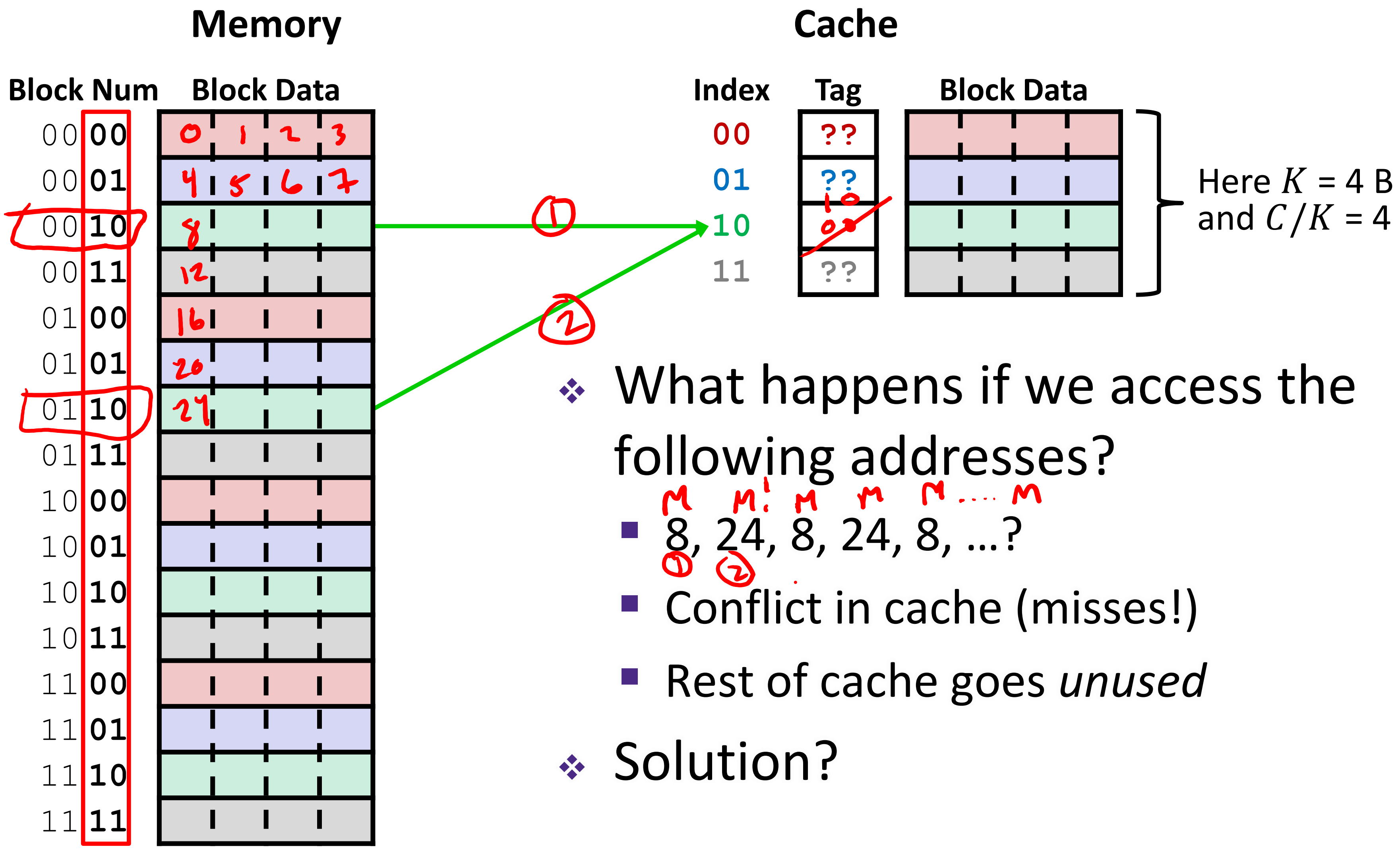
Review: Direct-Mapped Cache



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

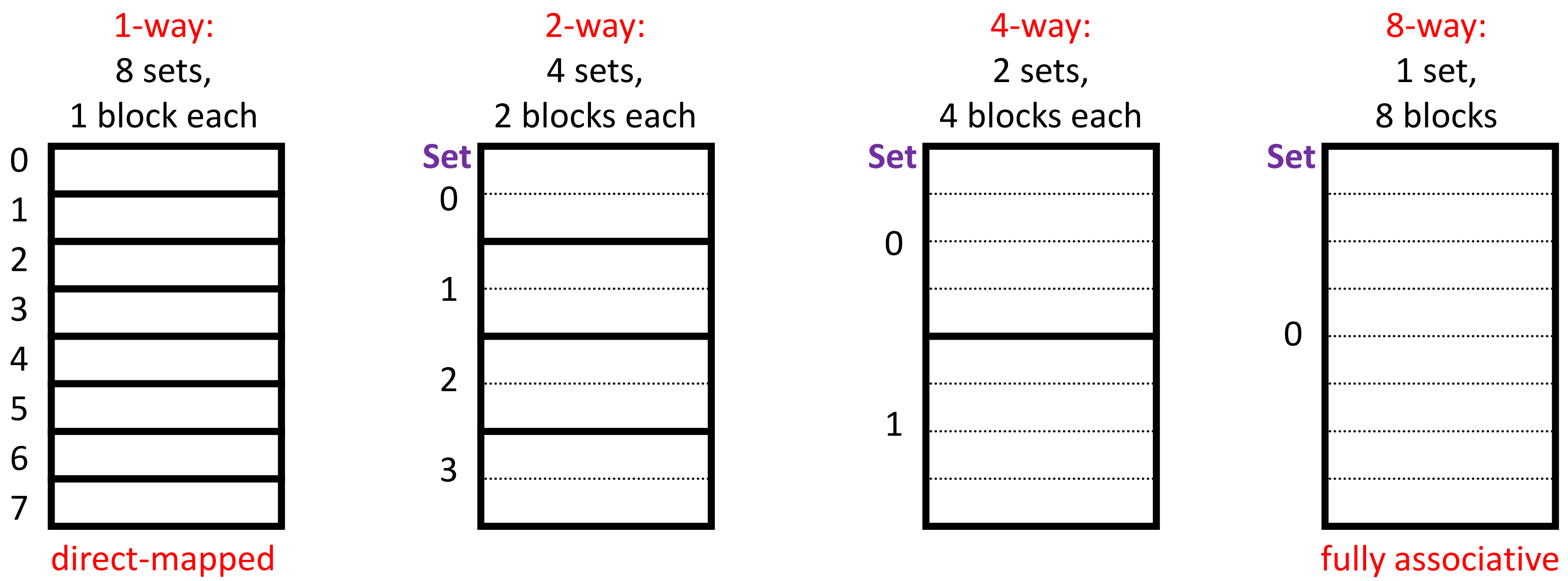
Direct-Mapped Cache Problem



- What happens if we access the following addresses?
  - $\overset{M}{8}$, $\overset{M!}{24}$, $\overset{M}{8}$, $\overset{M}{24}$, $\overset{M}{8}$, …? M
  - 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**



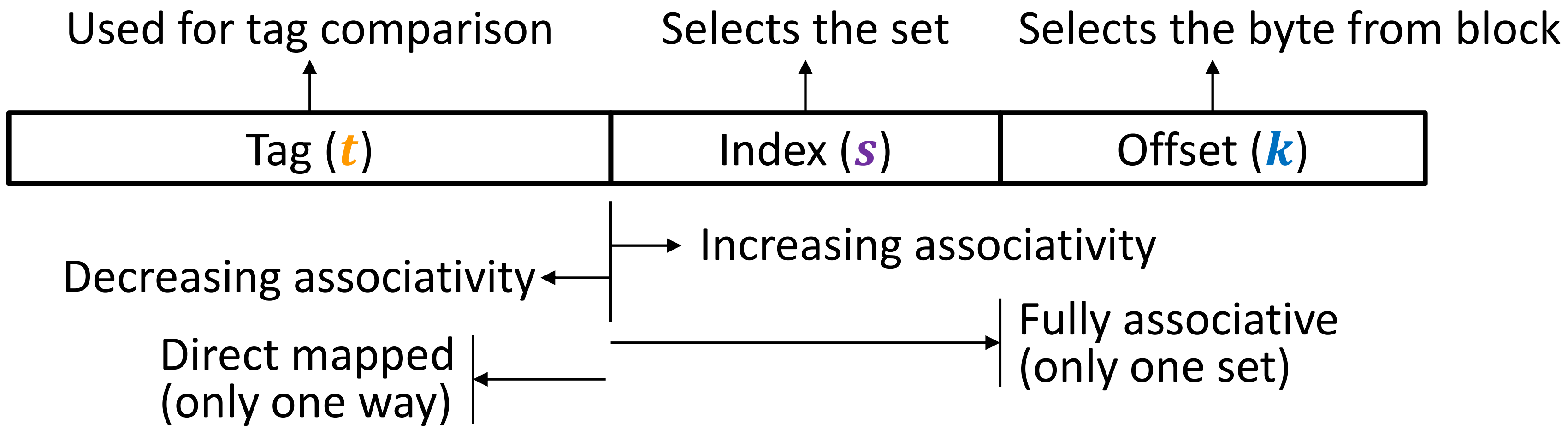
Cache Organization (3)

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

$C = S*E*K$

$s = \log_2\left(\frac{C}{EK}\right)$



Example Placement

block size:	16 B = K
capacity:	8 blocks
address:	16 bits = m

- Where would data from address `0x1833` be placed? A
  - Binary: `0b 0001 1000 0011 0011`

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

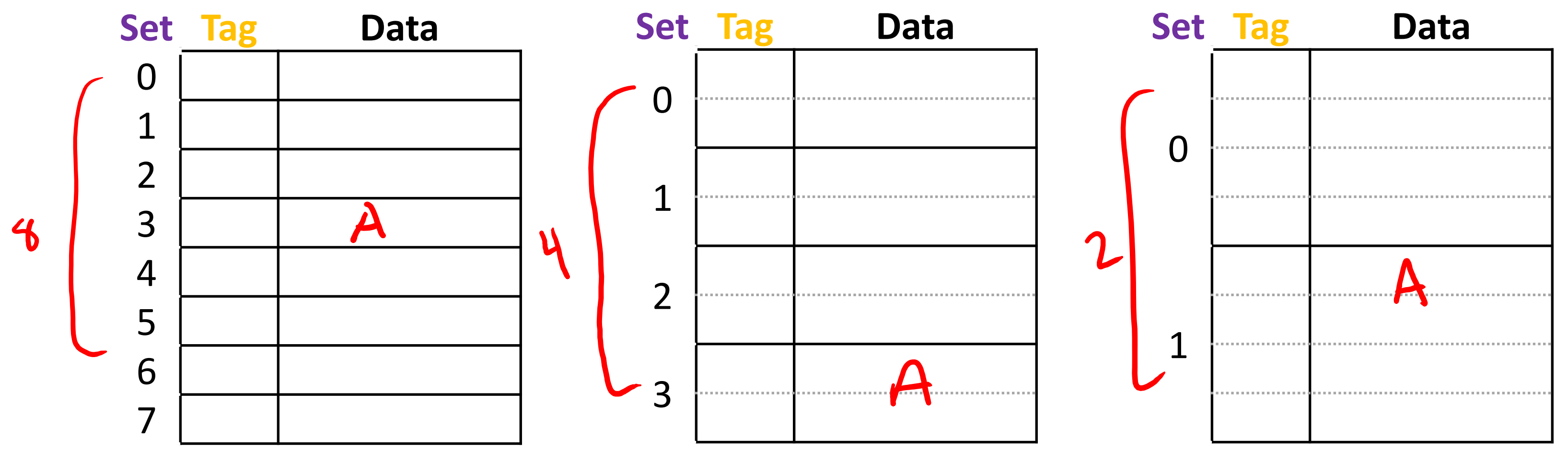
m-bit address: 16

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

$s = ?$ 3 — Direct-mapped

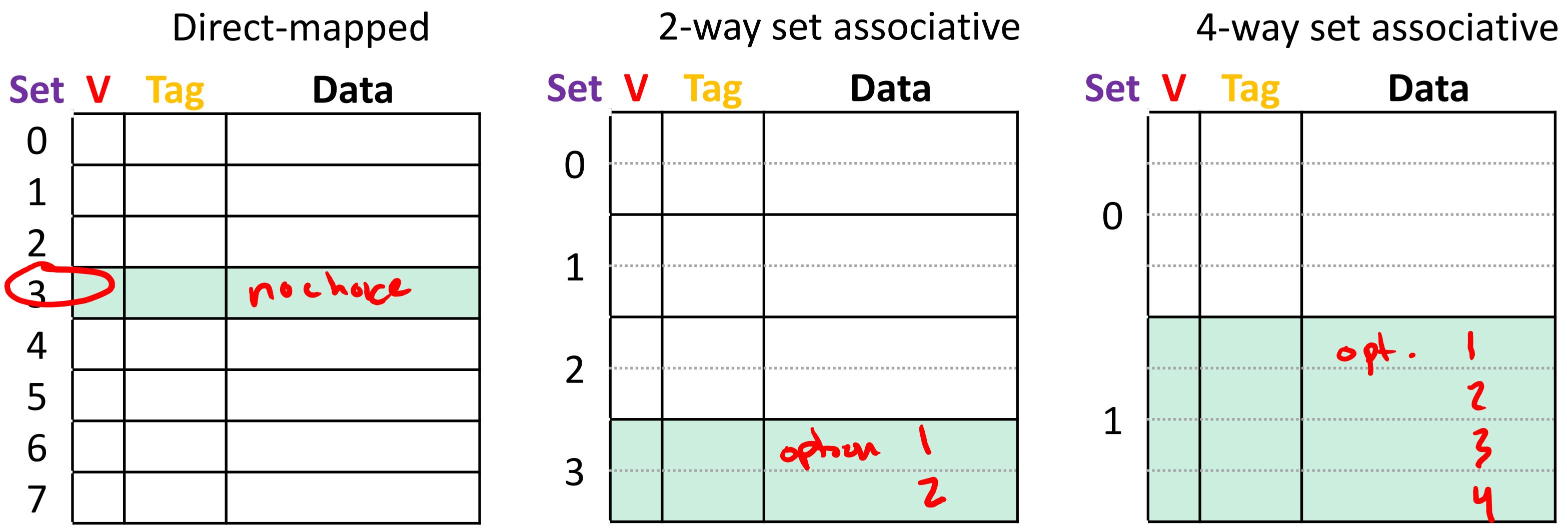
$s = ?$ 2 — 2-way set associative

$s = ?$ 1 — 4-way set associative



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*")



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

$C = S * E * K$

$E = \frac{C}{S K} = \frac{2^{11}}{2 * 2^7} = 2^3 = 8$

A. 2

B. 4

[✓ C. 8]

D. 16

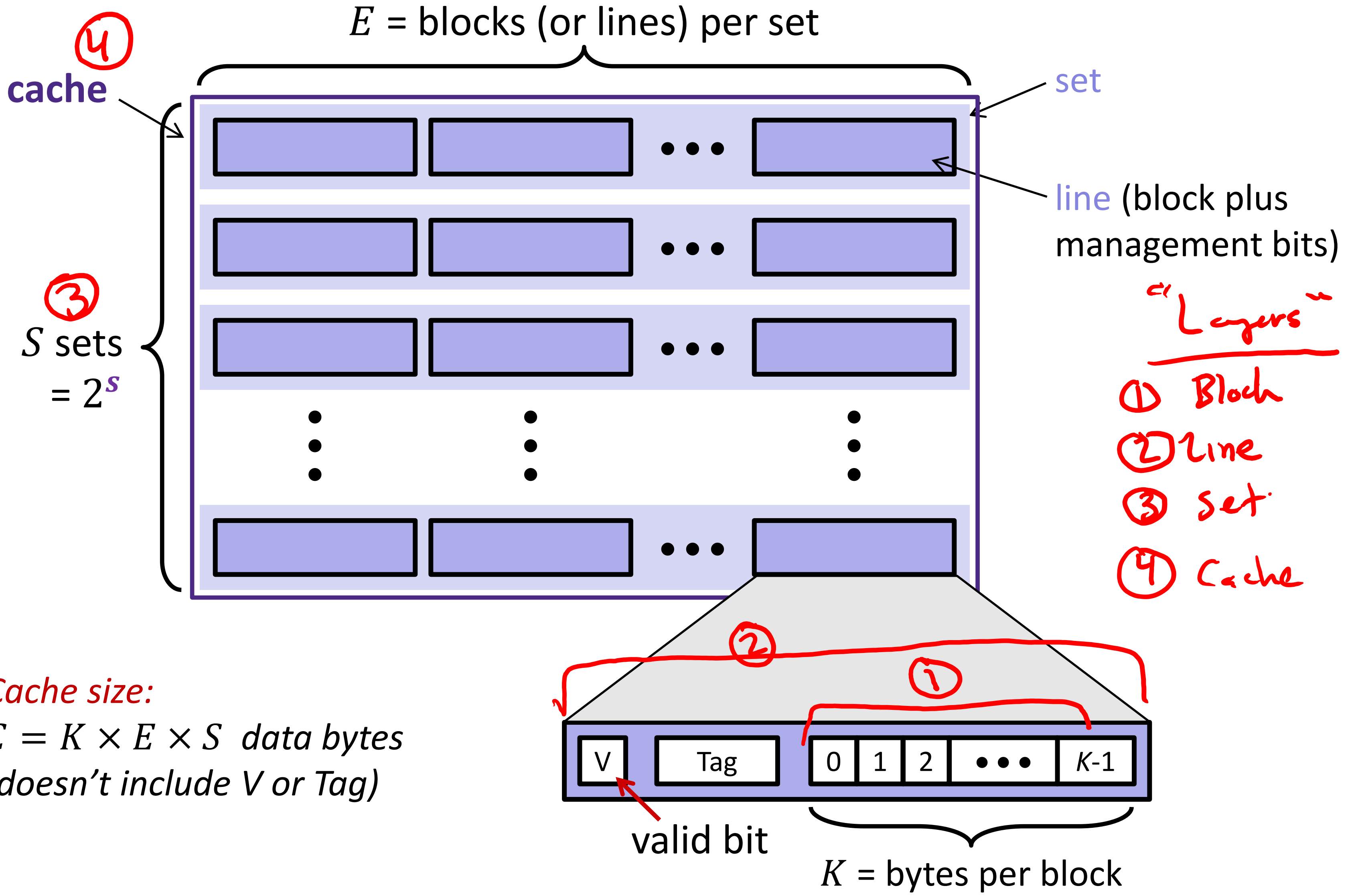
E. We're lost…

$s = \log_2(2) = 1$

$t = m - s - k = 16 - 1 - 7 = \boxed{8}$

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

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

MP

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

$M = 4\text{ KiB} \rightarrow 2^2 * 2^{10} = 2^{12}$ B

$m = \log_2 M = 12$

$m = t + s + k$

$S = \frac{C}{EK} = \frac{2^8}{2^1 2^5}$

$S = 2^2 = 4$

Cache Size	256 B
Block Size	32 B
Associativity	2-way
Hit Time	3 cycles
Miss Rate	20%
Tag Bits	5
Index Bits	$s = \log_2 4 = 2$
Offset Bits	5
AMAT	28 cycles

$C = S * E * K$

$K \Rightarrow k = \log_2 32 = 5$

E

HT

MR

$t = 12 - \overset{2}{\cancel{8}} - 5 = 5$

$s = 12 - t - 5$

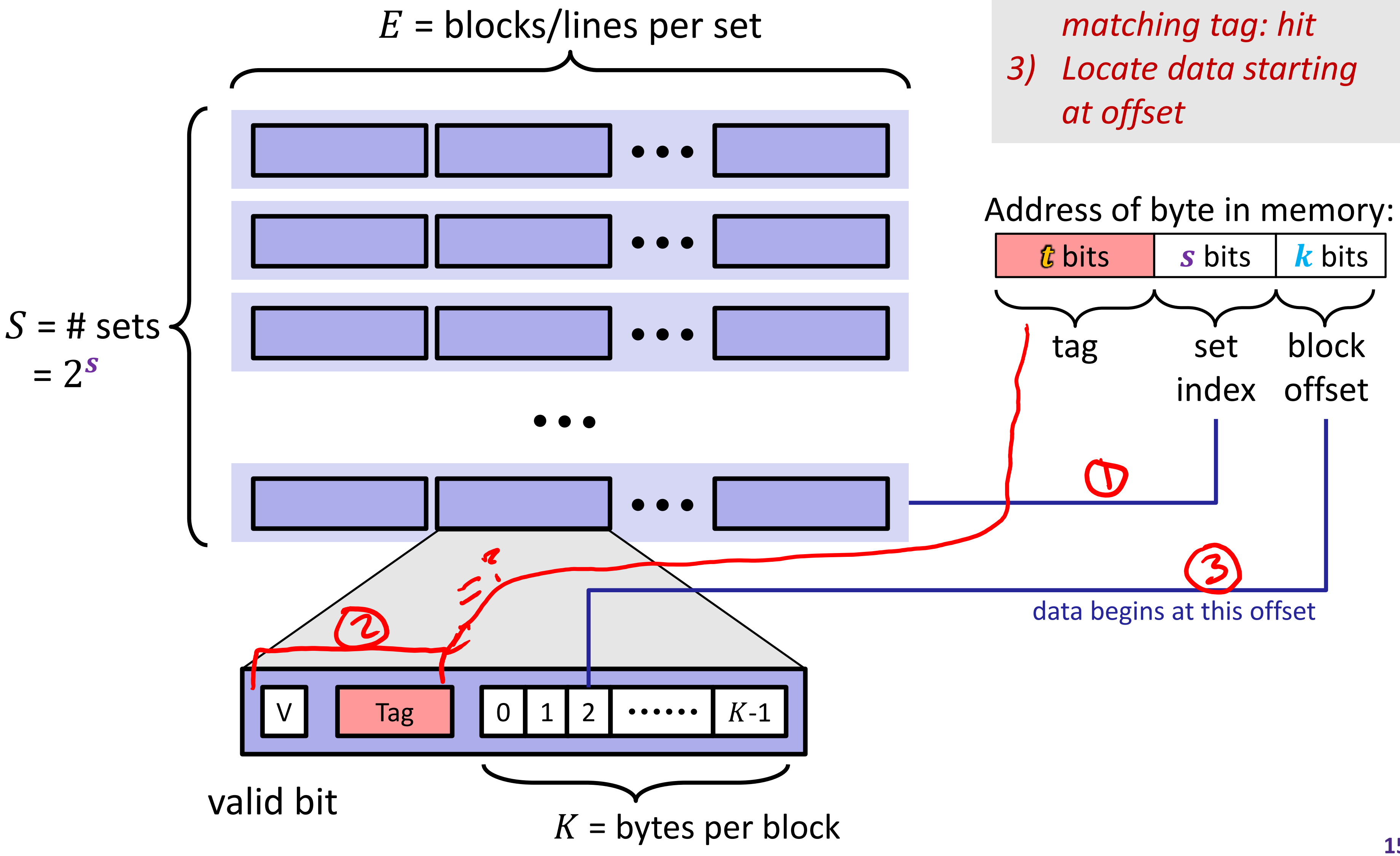
$= HT + MR * MP$

$= 3 + (.2)(125)$

$= 3 + 25$

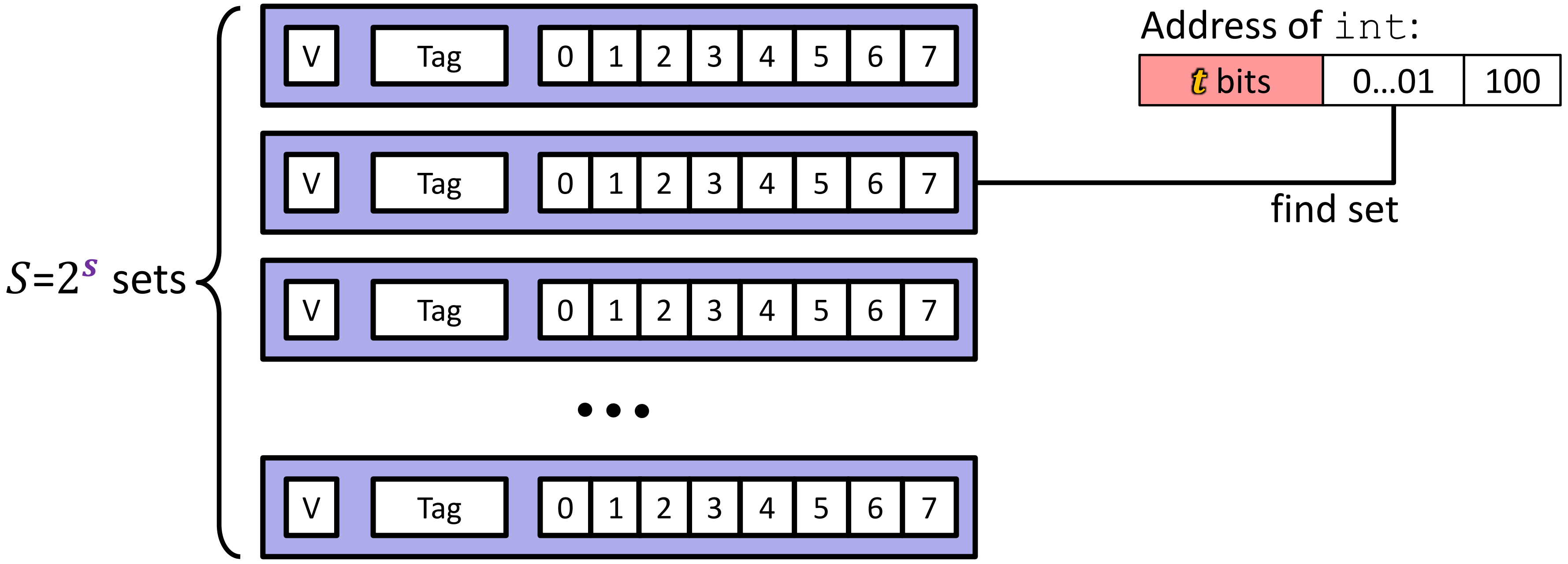
Cache Read

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



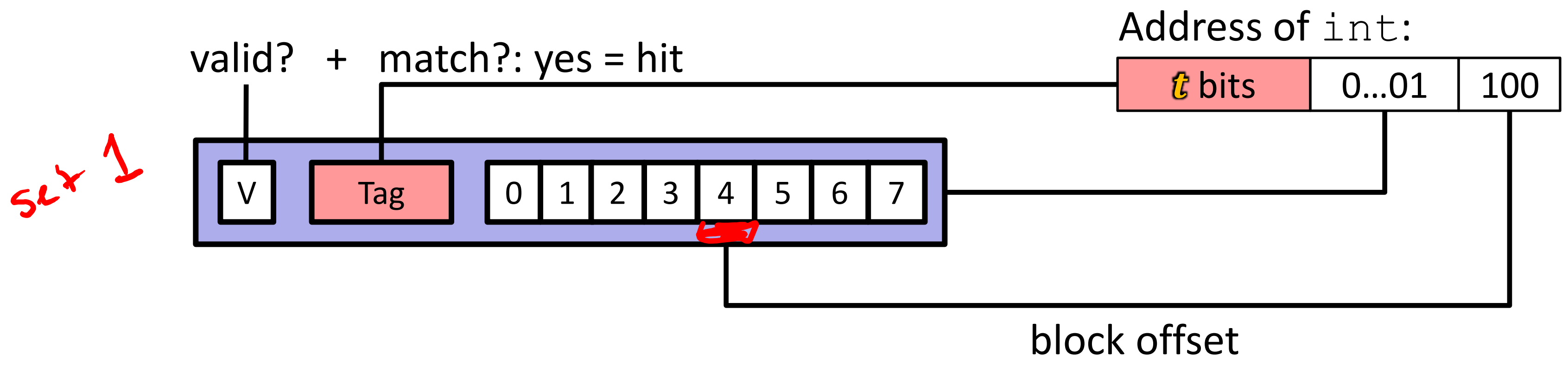
Example: Direct-Mapped Cache (E = 1)

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



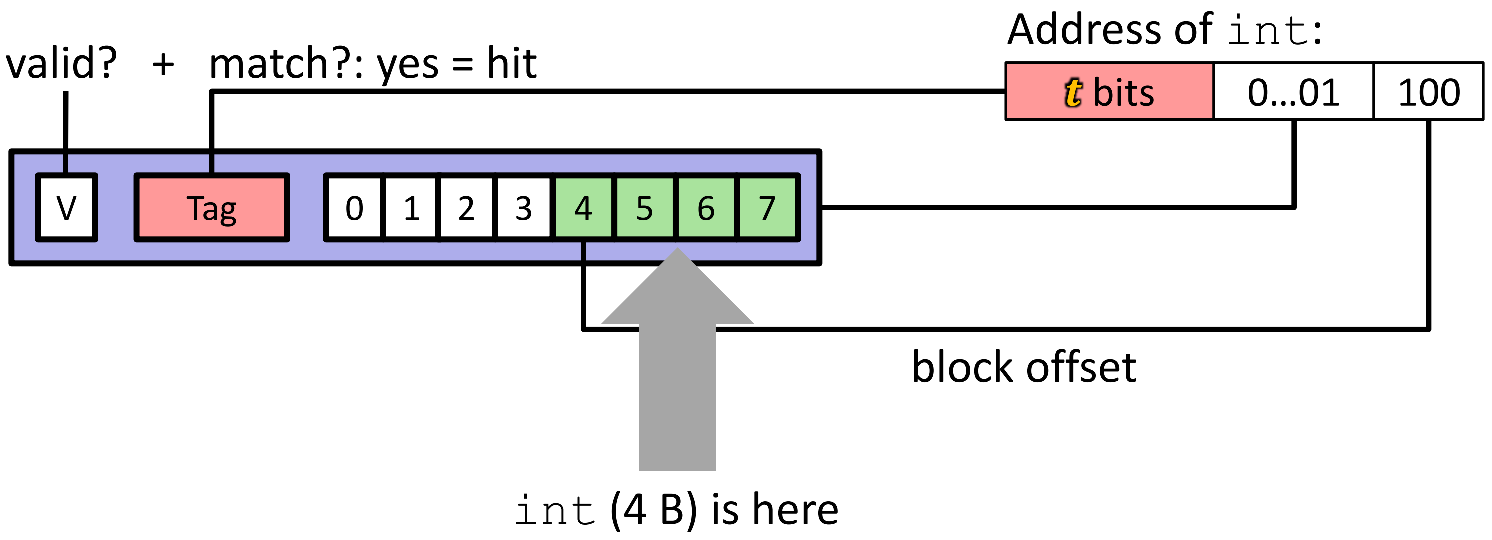
Example: Direct-Mapped Cache (E = 1)

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



Example: Direct-Mapped Cache (E = 1)

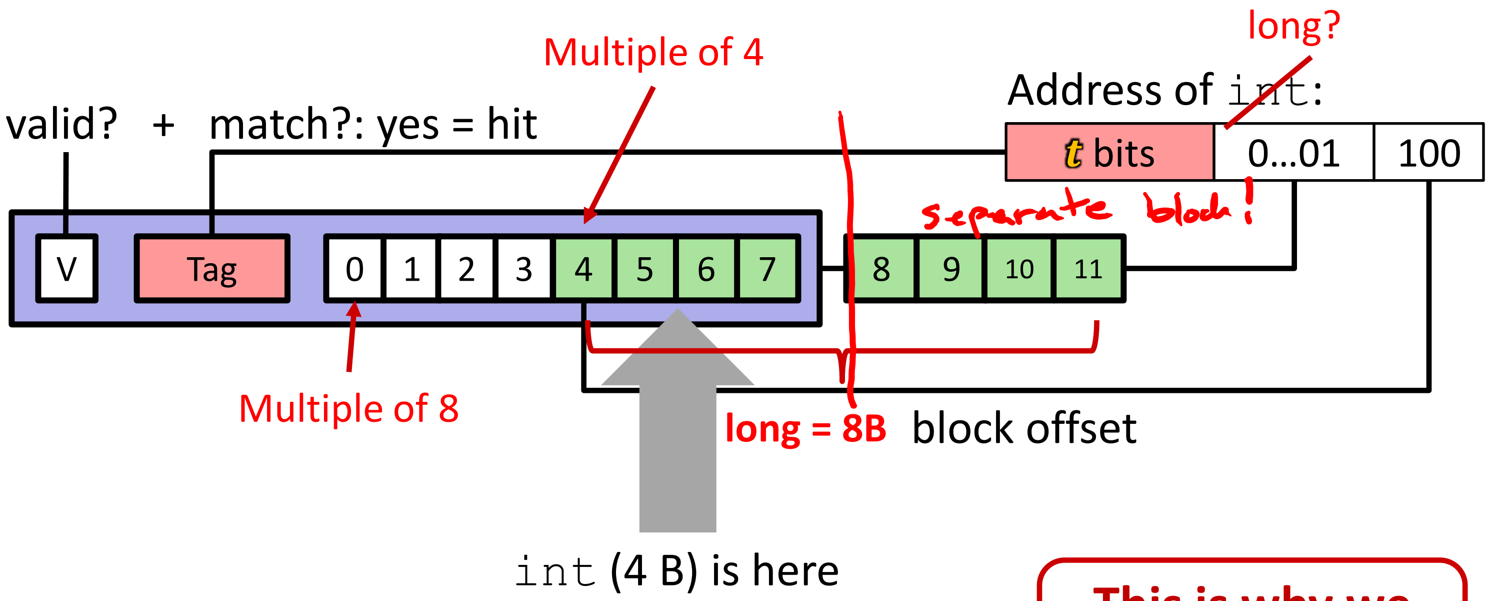
Direct-mapped: One line per set
Block Size K = 8 B



No match? Then old line gets evicted and replaced

Example: Direct-Mapped Cache (E = 1)

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

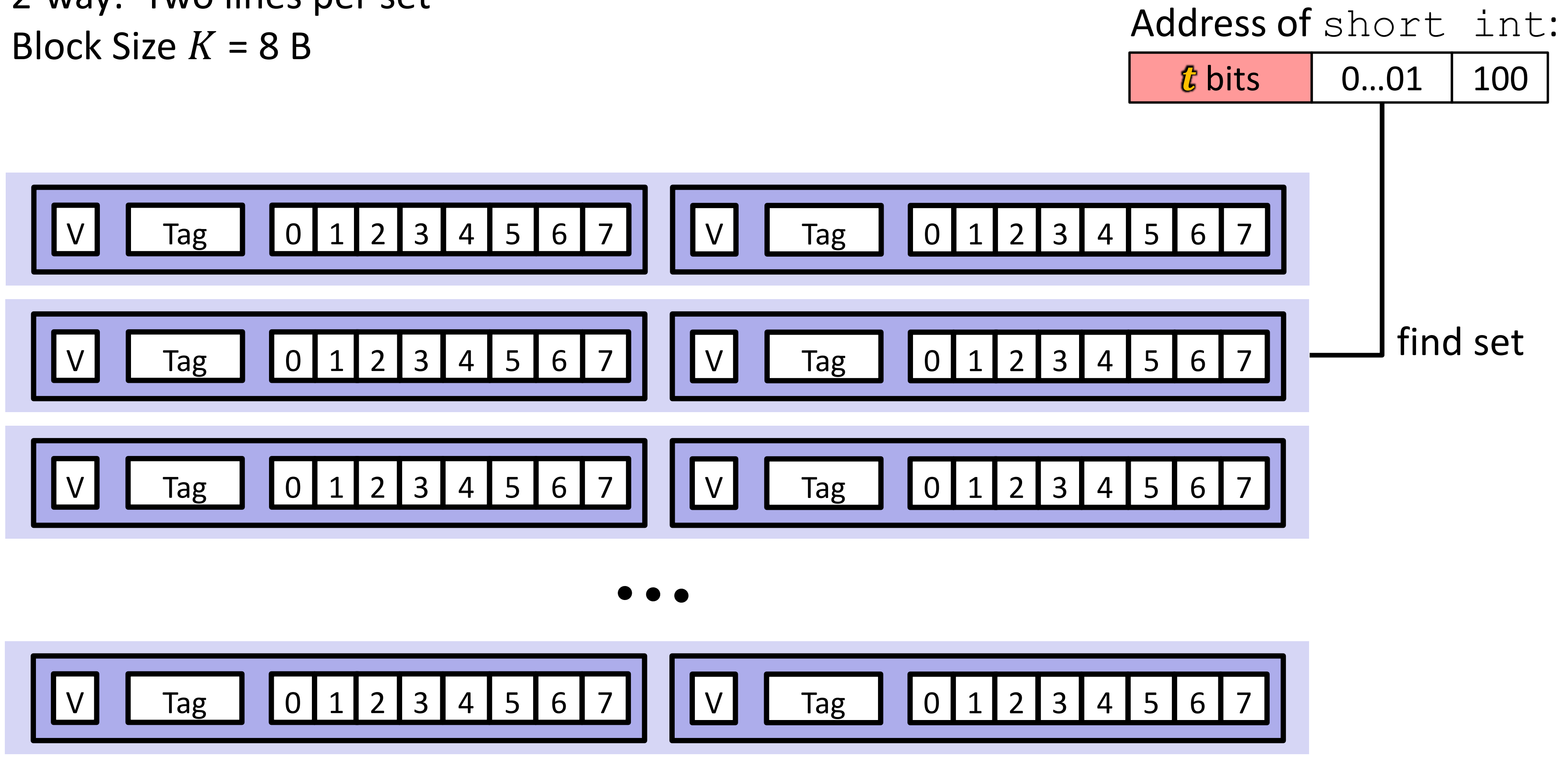


This is why we want alignment!

No match? Then old line gets evicted and replaced

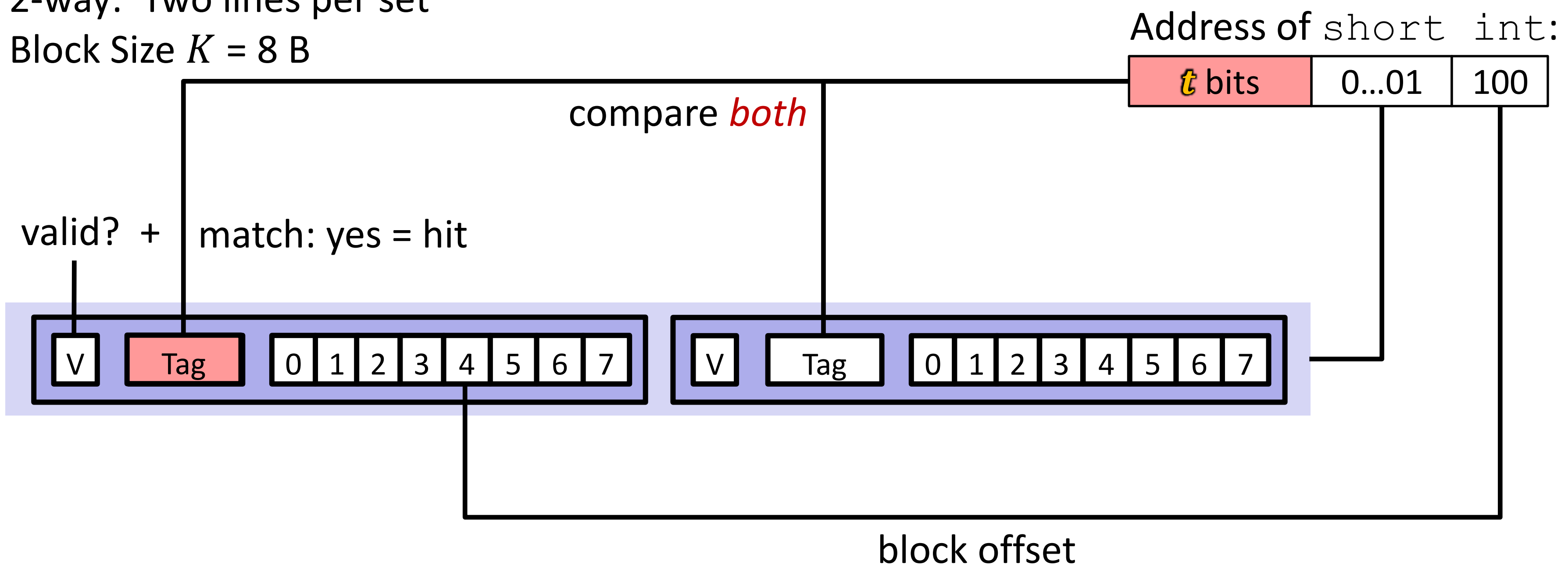
Example: Set-Associative Cache (E = 2)

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



Example: Set-Associative Cache (E = 2)

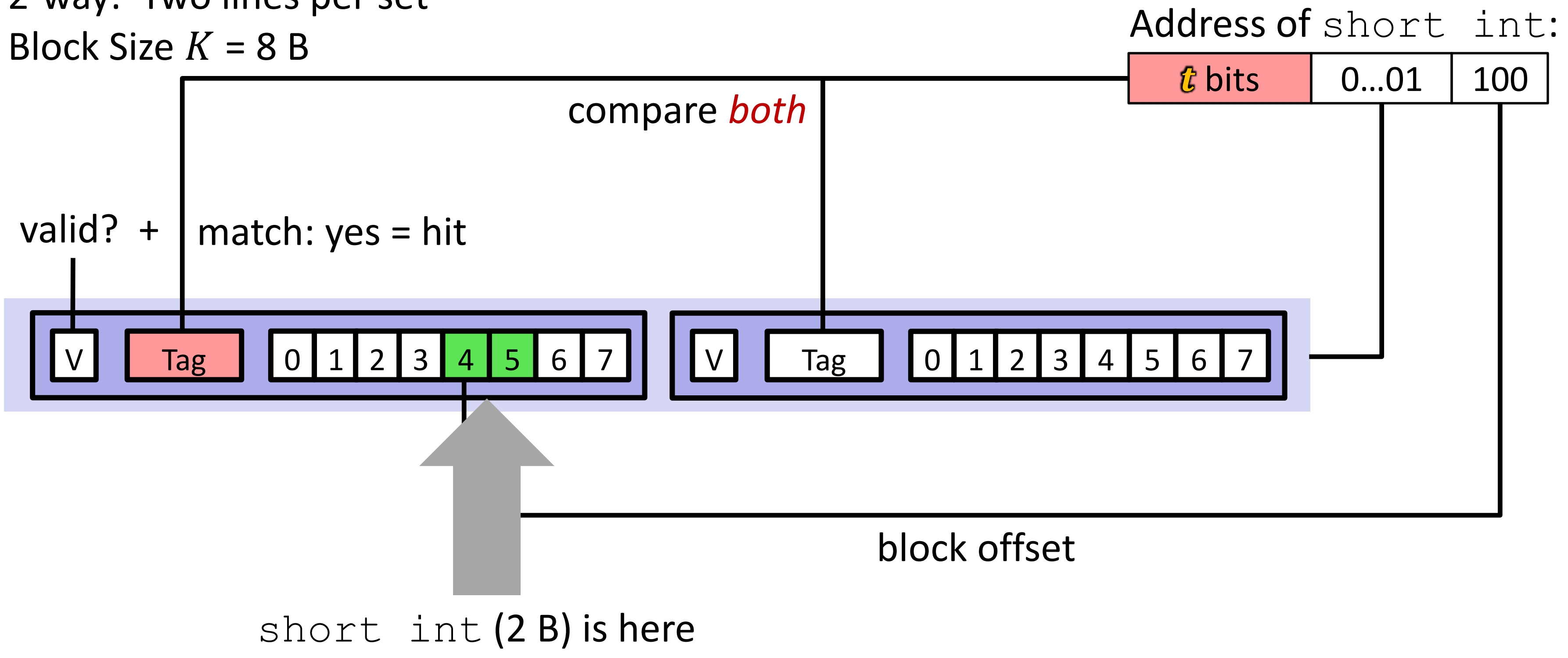
2-way: Two lines per set
Block Size K = 8 B



Example: Set-Associative Cache (E = 2)

Generalizes to any E!

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), …

Round - Robin
Not Most Rec. Used
Pseudo - 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

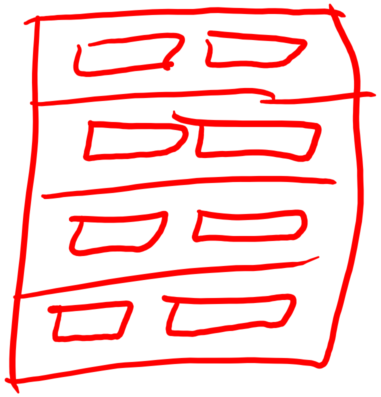
assume: only consider accesses to ar

- Assuming the cache starts cold (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

$S = \frac{256}{2 * 32} = 4$

```
#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];
```

t = 5 bits, s = 2 bits, k = 5 bits
ar is array of 64 elements arranged in 8 rows and 8 columns, stored in row-major order
Miss rate = ¼
Explanation: Each cache block (32B) holds 4 array entries (long = 8B). Thus, access to ar[0][0] misses, but the loaded block will hold ar[0][0], ar[0][1], ar[0][2], and ar[0][3]. Since the loop ordering iterates columns in the inner-loop, the next three accesses to ar[0][1-3] hit. The access to ar[0][4] misses, but loads the next three elements, too. The pattern repeats with a miss followed by three hits, so the miss rate is one miss per four accesses.



Note: For this specific code, because no element is accessed a second time, it doesn't matter where the blocks get placed in the cache.

Challenge Question: What is the miss rate if we switch the ordering of the loops?
(hint: where the blocks are placed in the cache matters now!)