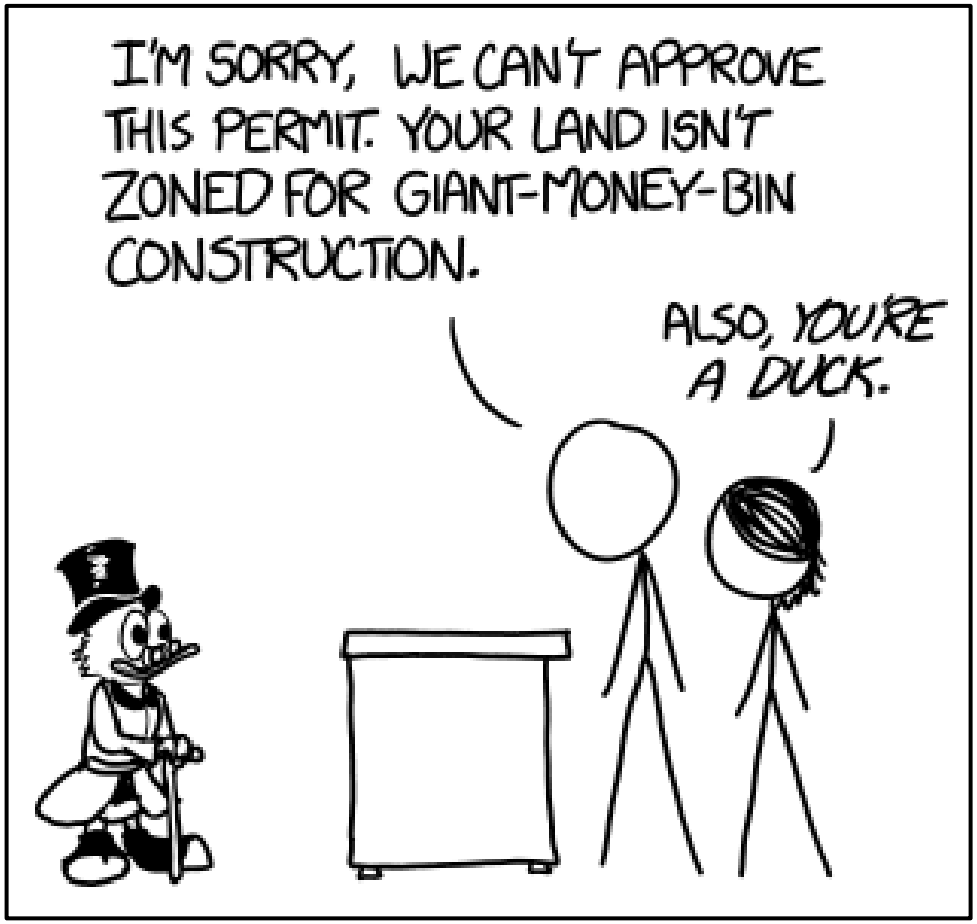
Caches III

CSE 351 Summer 2020

Instructor:
Porter Jones

Teaching Assistants:
Amy Xu
Callum Walker
Sam Wolfson
Tim Mandzyuk



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

Administrivia

- Questions doc: https://tinyurl.com/CSE351-7-31
- hw16 due Wednesday (8/5) – 10:30am
- Lab 3 due Tonight (7/31) – 11:59pm
  - You get to write some buffer overflow exploits!
- Lab 4 released later today
  - All about caches!
- Unit Summary 2 Due next Wednesday (8/5) – 11:59pm

Making memory accesses fast!

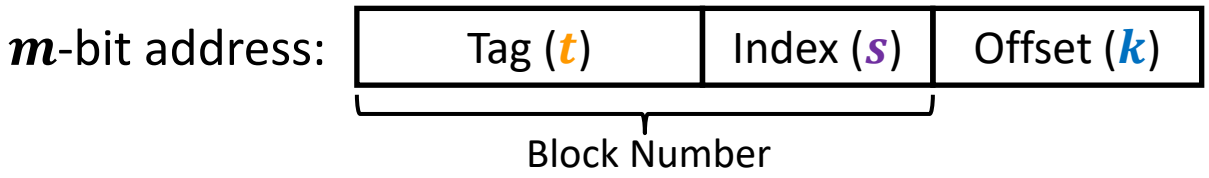
- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
  - Direct-mapped (*sets*; index + tag)
  - **Associativity (*ways*)**
  - **Replacement policy**
  - Handling writes
- Program optimizations that consider caches

Review: Cache Parameters

- **Block size (K):** basic unit of transfer between memory and the cache, given in bytes (e.g. 64 B).
- **Cache size (C):** Total amount of data that can be stored in the cache, given in bytes (e.g. 32 KiB).
  - Must be multiple of block size
  - Number of blocks in cache is calculated by C/K
- **Associativity (E):** Number of ways blocks can be stored in a cache set, or how many blocks in each set
- **Number of sets (S):** Number of unique sets that blocks can be placed into in a cache (calculated as C/K/E).

Review: TIO address breakdown

- TIO address breakdown:



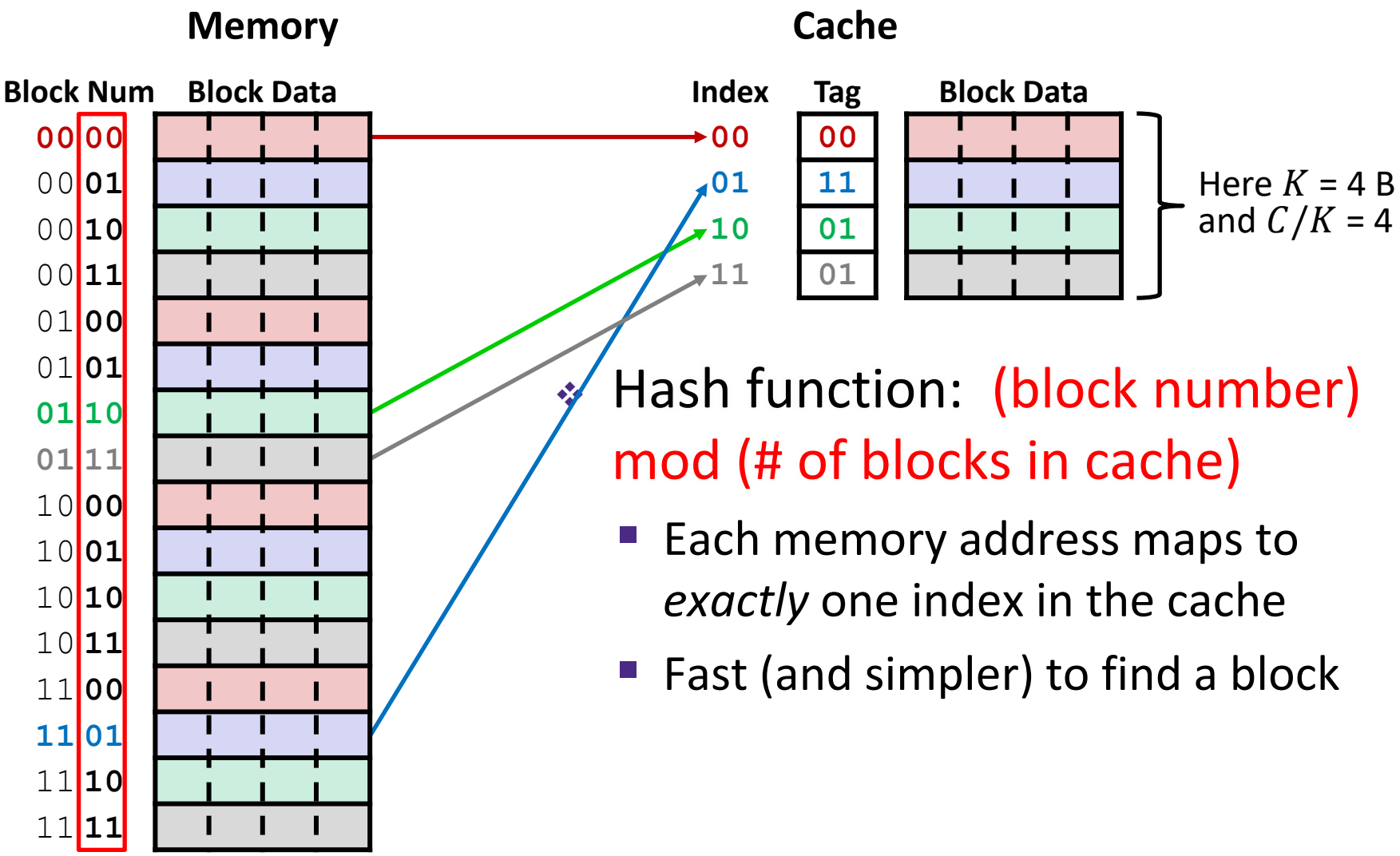
- **Index (s)** field tells you where to look in cache
  - Number of bits is determined by number of sets ($\log_2(C/K/E)$)
  - Need enough bits to reference every set in the cache
- **Tag (t)** field lets you check that data is the block you want
  - Rest of the bits not used for index and offset ($m - s - k$)
- **Offset (k)** field selects specified start byte within block
  - Number of bits is determined by block size ($\log_2(K)$)
  - Need enough bits to reference every byte in a block

Review: Cache Lookup Process

- CPU requests data at a given address
- Cache breaks down address into different bit fields
  - Determines offset, index, and tag bits
- Cache checks to see if block containing address is already in the cache
  - Uses index bits to find which set to look in
  - Uses tag bits to make sure the block in the set matches

- If block is in the cache, it's a **cache hit**
  - Data is returned to CPU starting at byte offset
- If block is not in the cache, it's a **cache miss**
  - Block is loaded from memory into the cache, evicting other blocks from the cache if necessary
  - Data is returned to CPU starting at byte offset

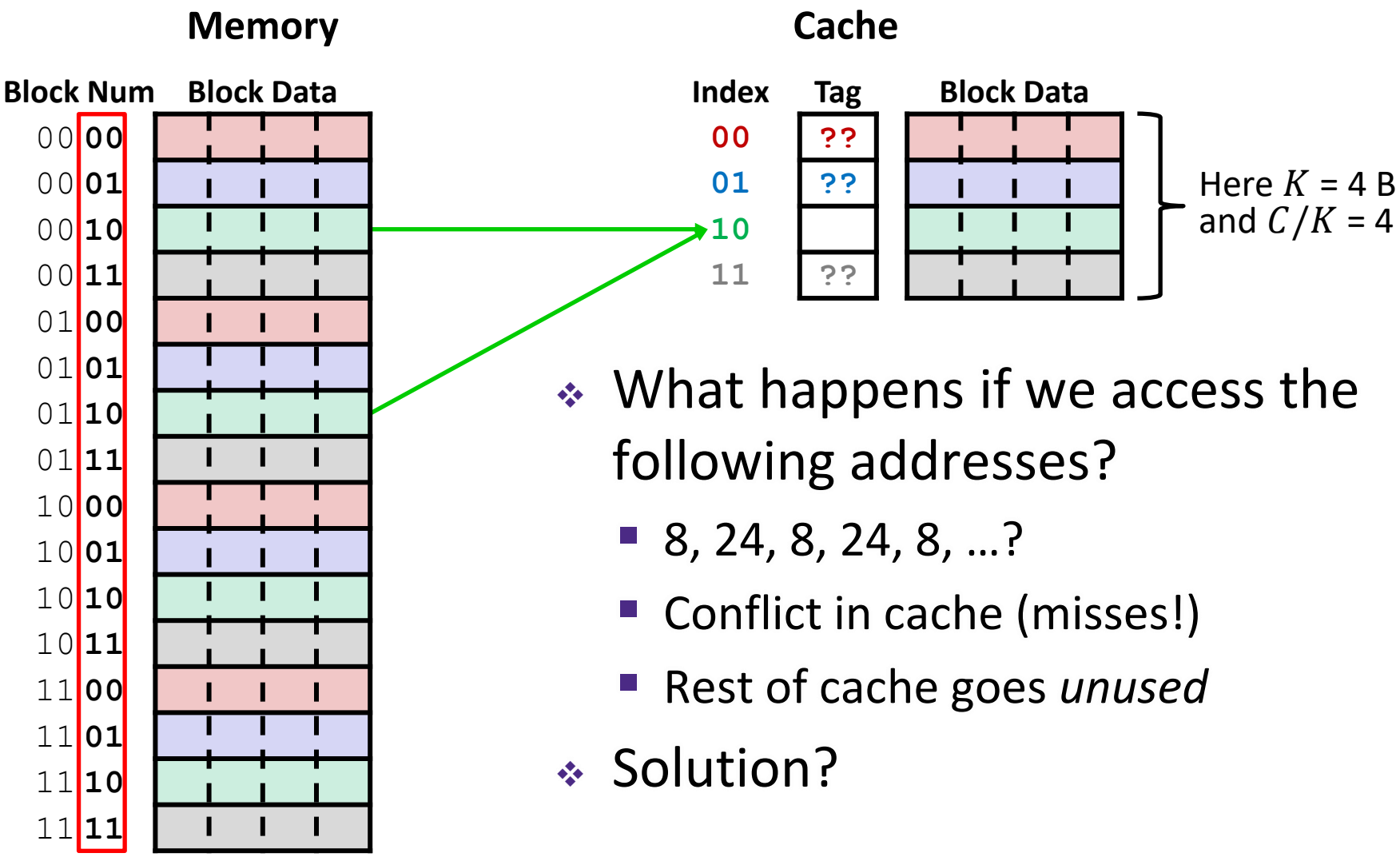
Review: Direct-Mapped Cache



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

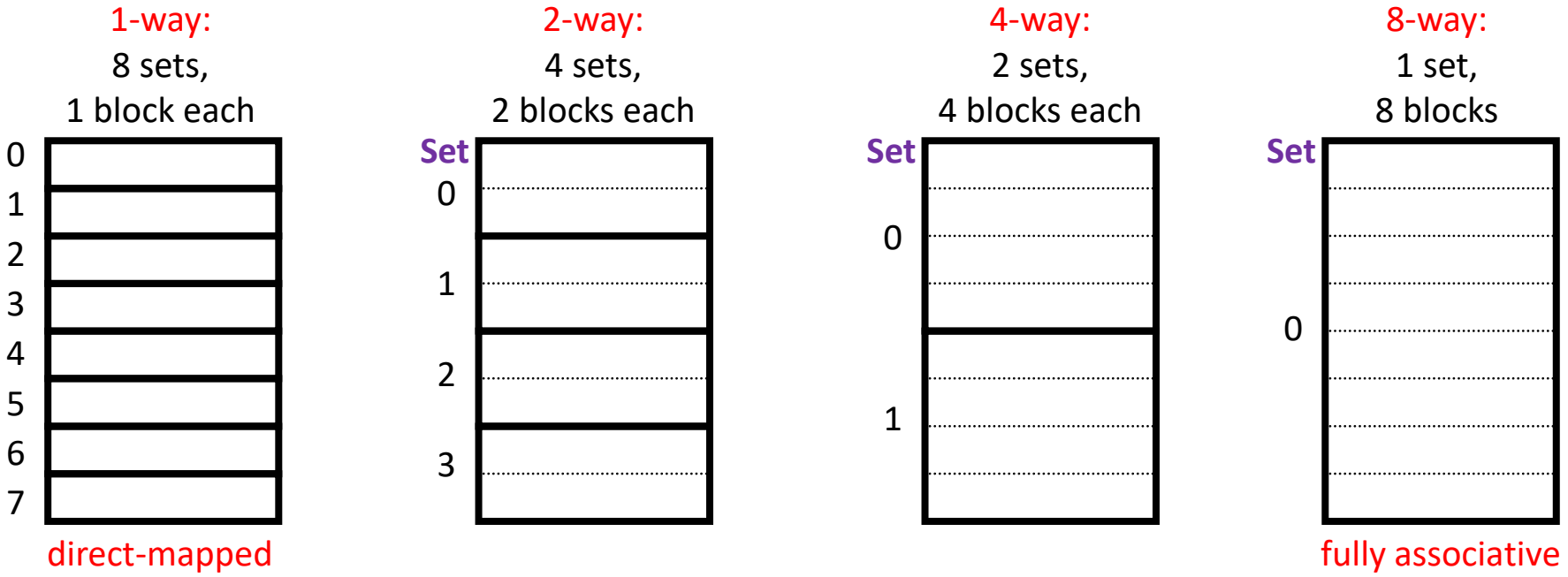
Direct-Mapped Cache Problem



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

Associativity

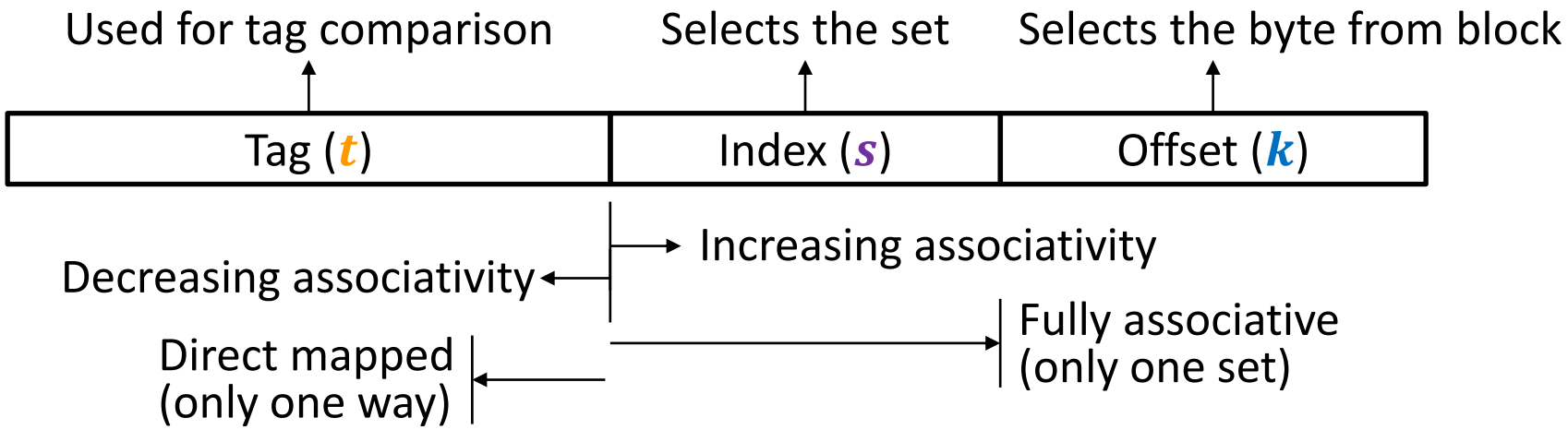
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**



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`

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

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

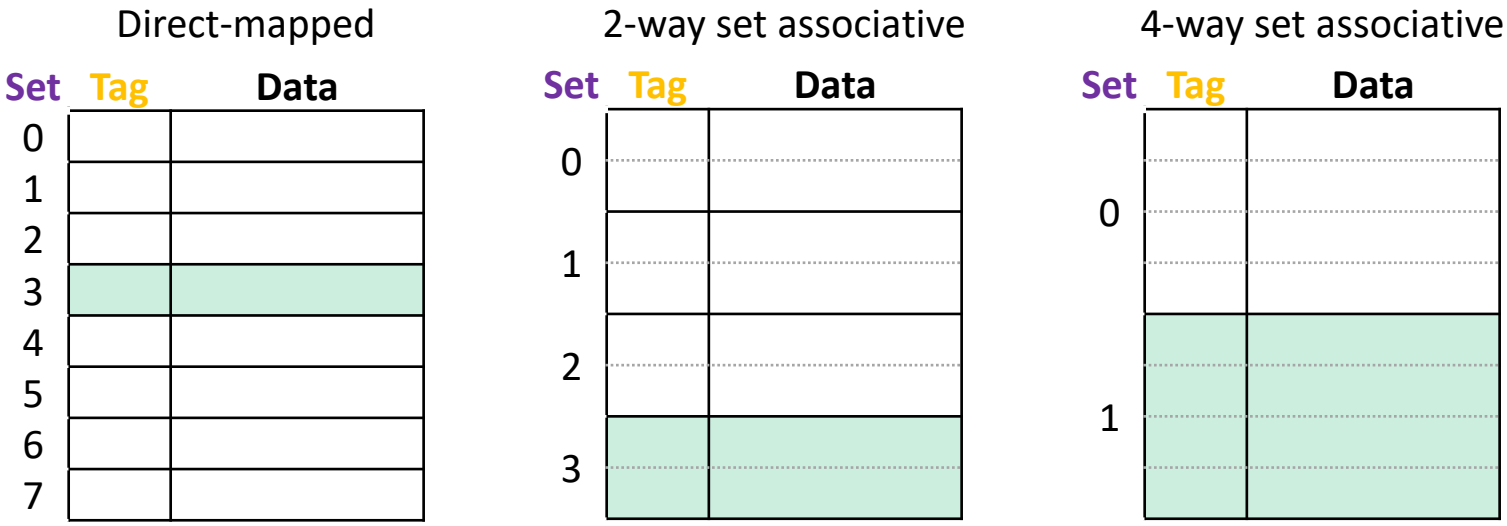
Set	Tag	Data
0		
1		
2		
3		

s = ?
4-way set associative

Set	Tag	Data
0		
1		

Block 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*")



Polling Question [Cache III]

- 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/pbjones

A. 2

B. 4

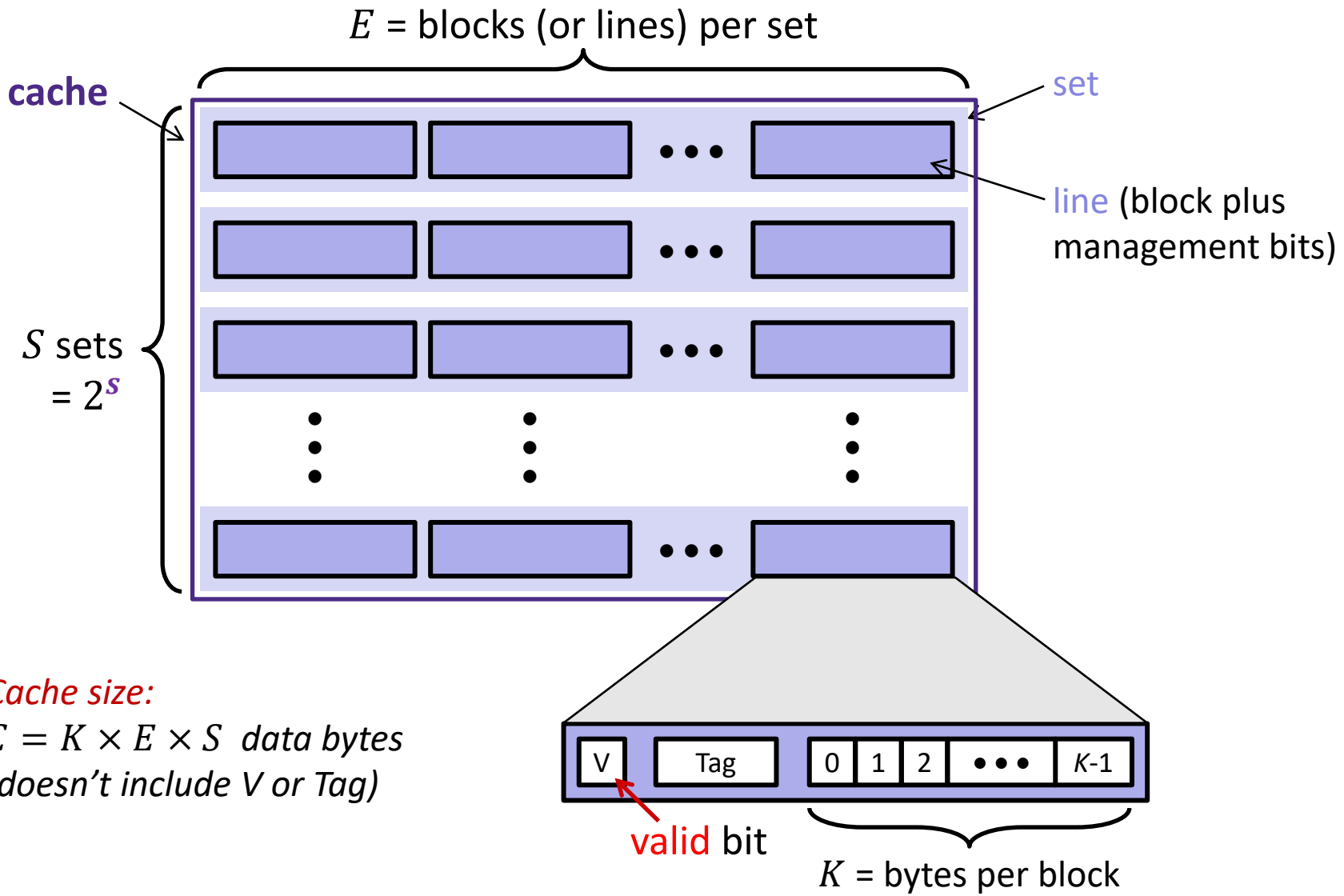
C. 8

D. 16

E. We're lost…

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

General Cache Organization (S, E, K)



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)	

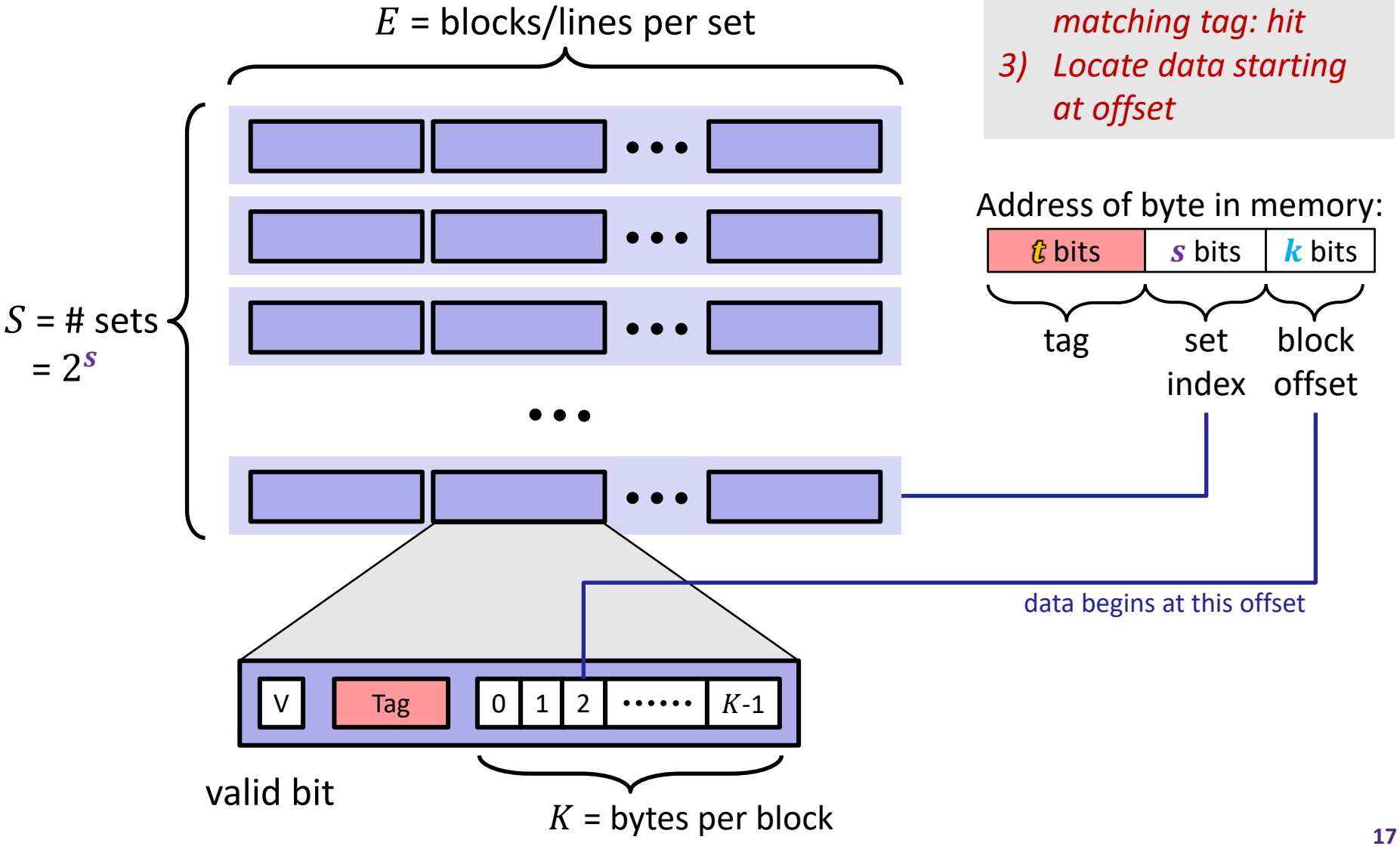
Example Cache Parameters Problem

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

Cache Size	256 B
Block Size	32 B
Associativity	2-way
Hit Time	3 cycles
Miss Rate	20%
Tag Bits	
Index Bits	
Offset Bits	
AMAT	

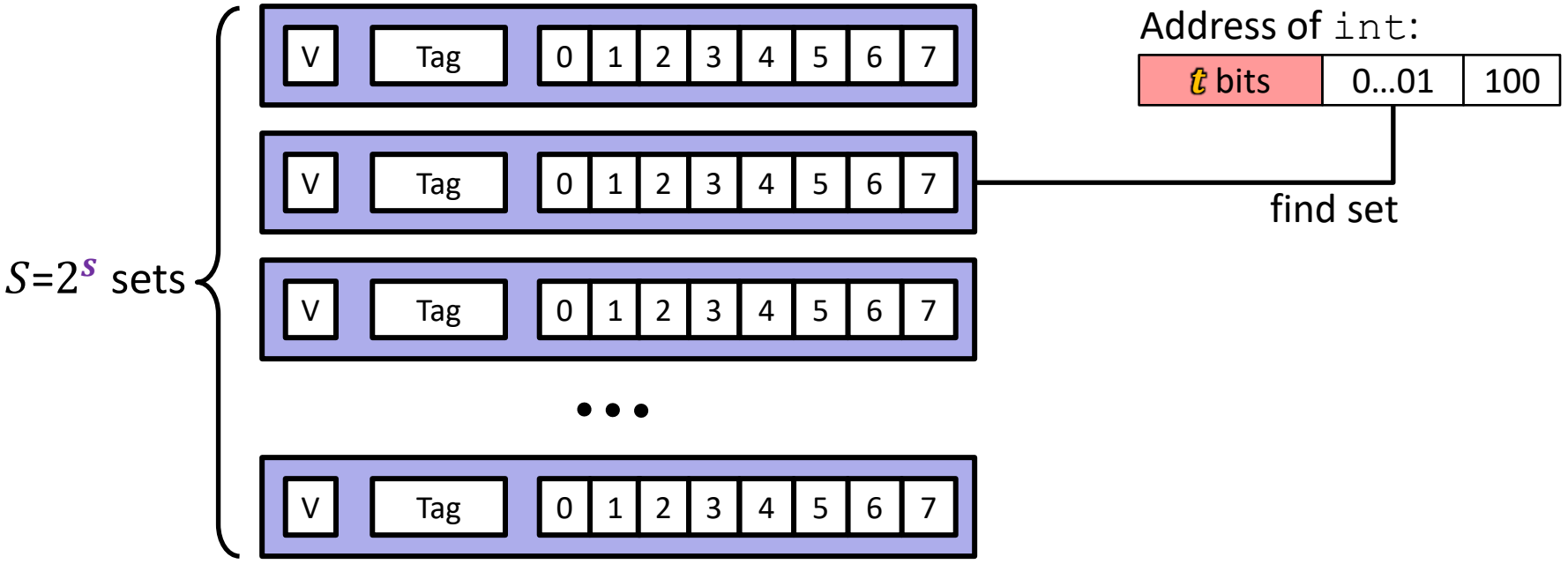
Cache Read

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



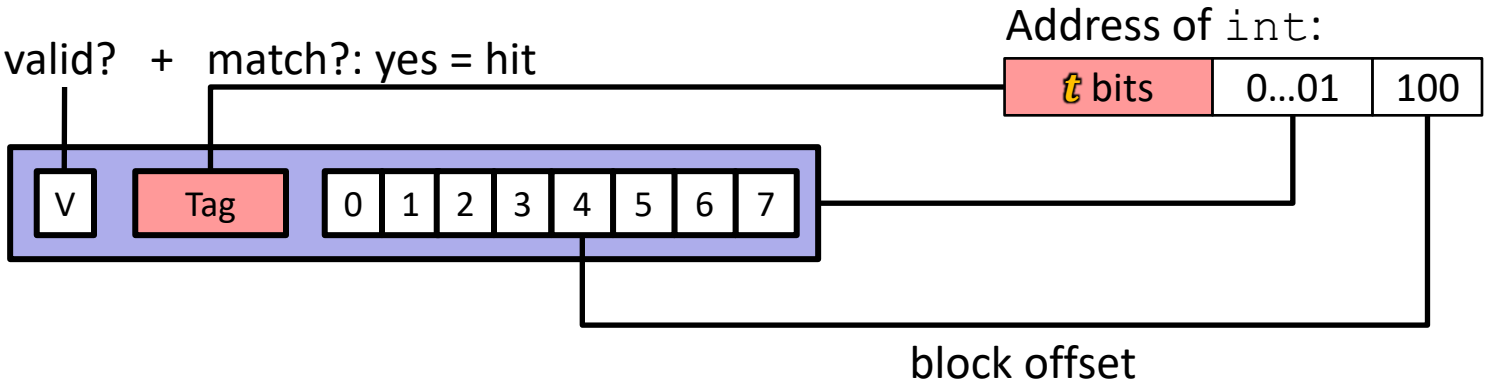
Example: Direct-Mapped Cache (E = 1)

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



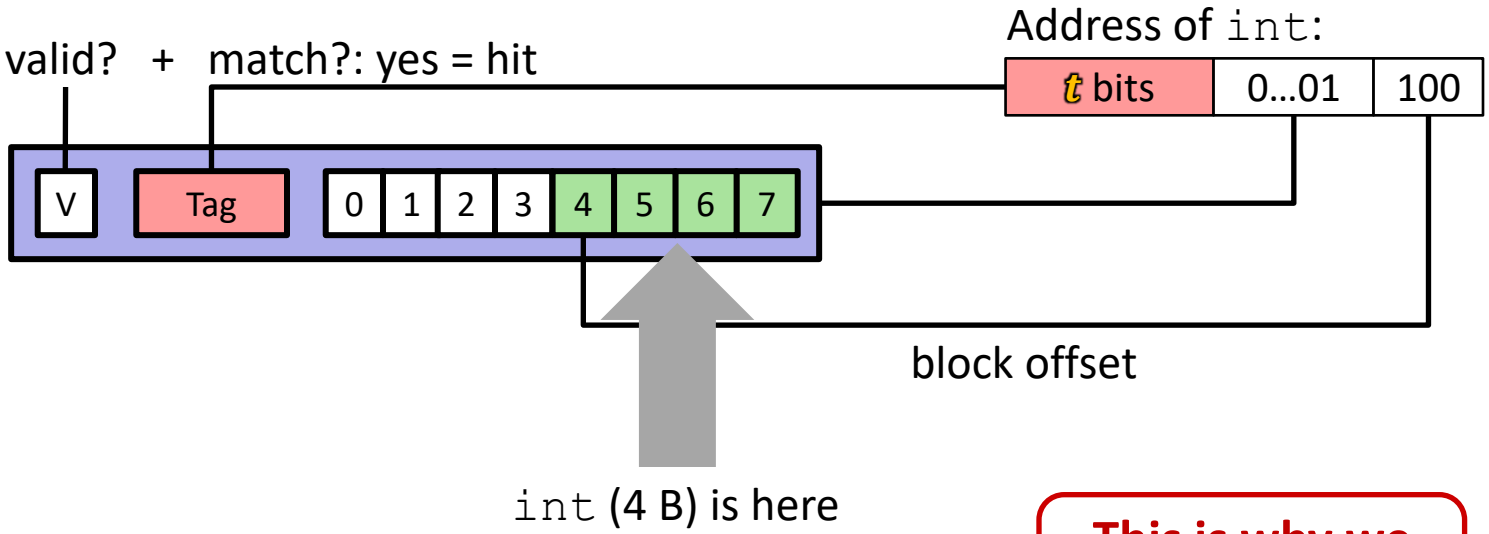
Example: Direct-Mapped Cache (E = 1)

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

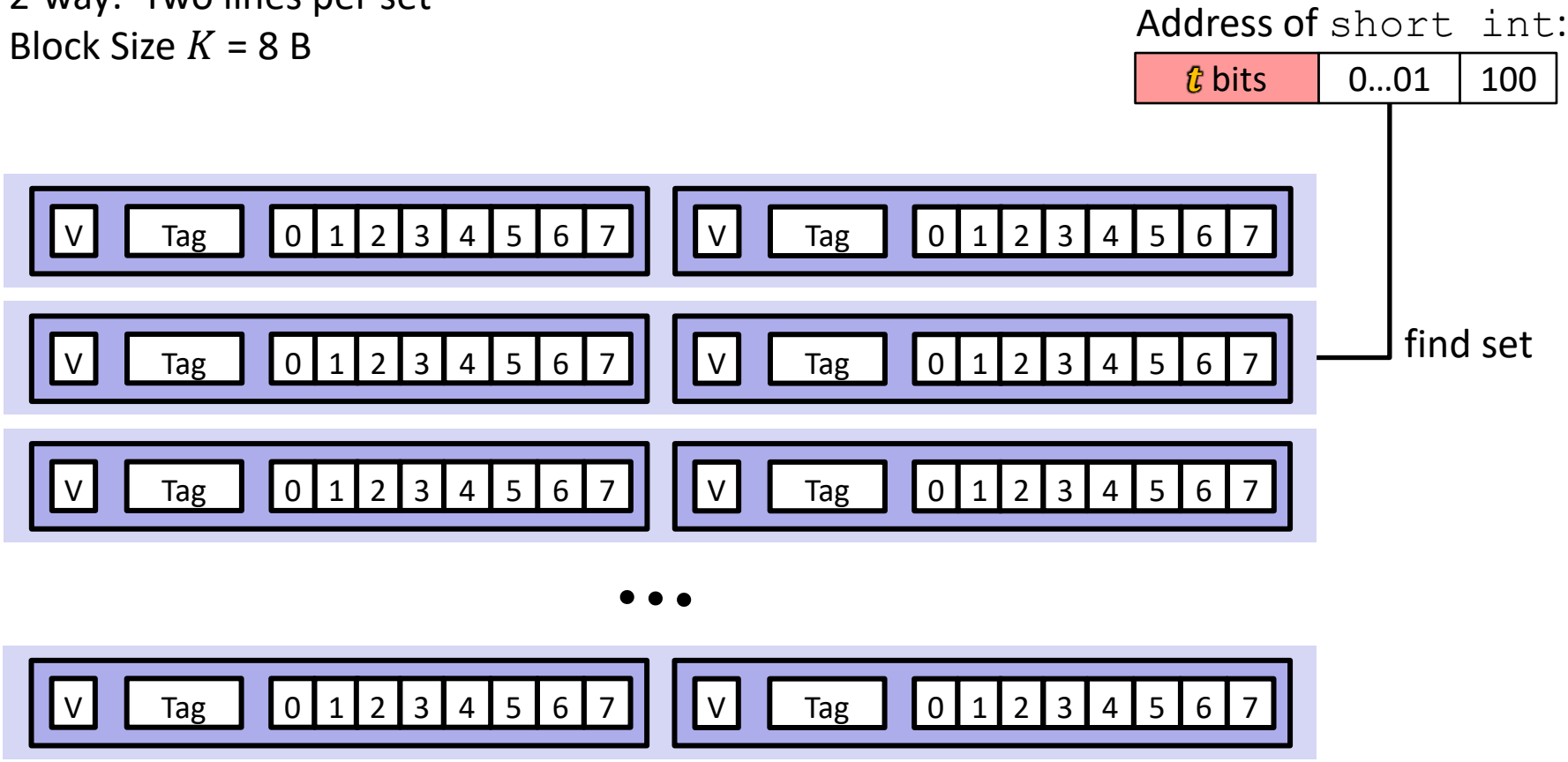


This is why we want alignment!

No match? Then old line gets evicted and replaced

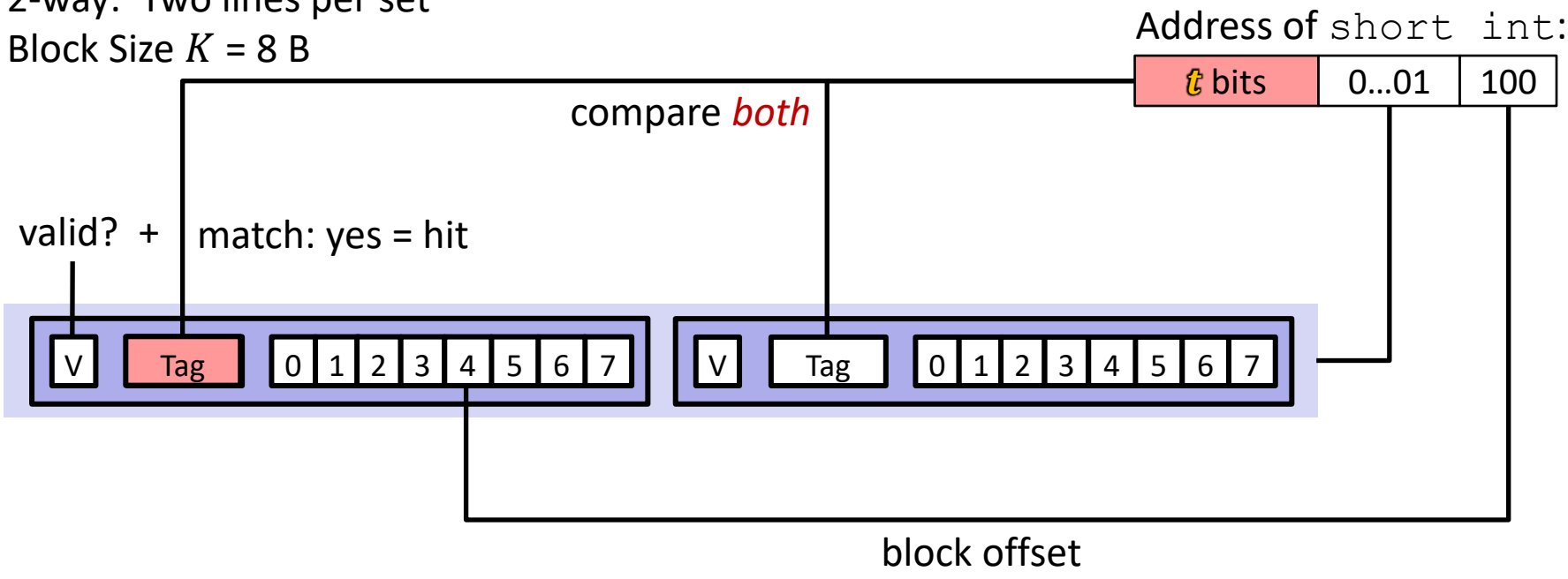
Example: Set-Associative Cache (E = 2)

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

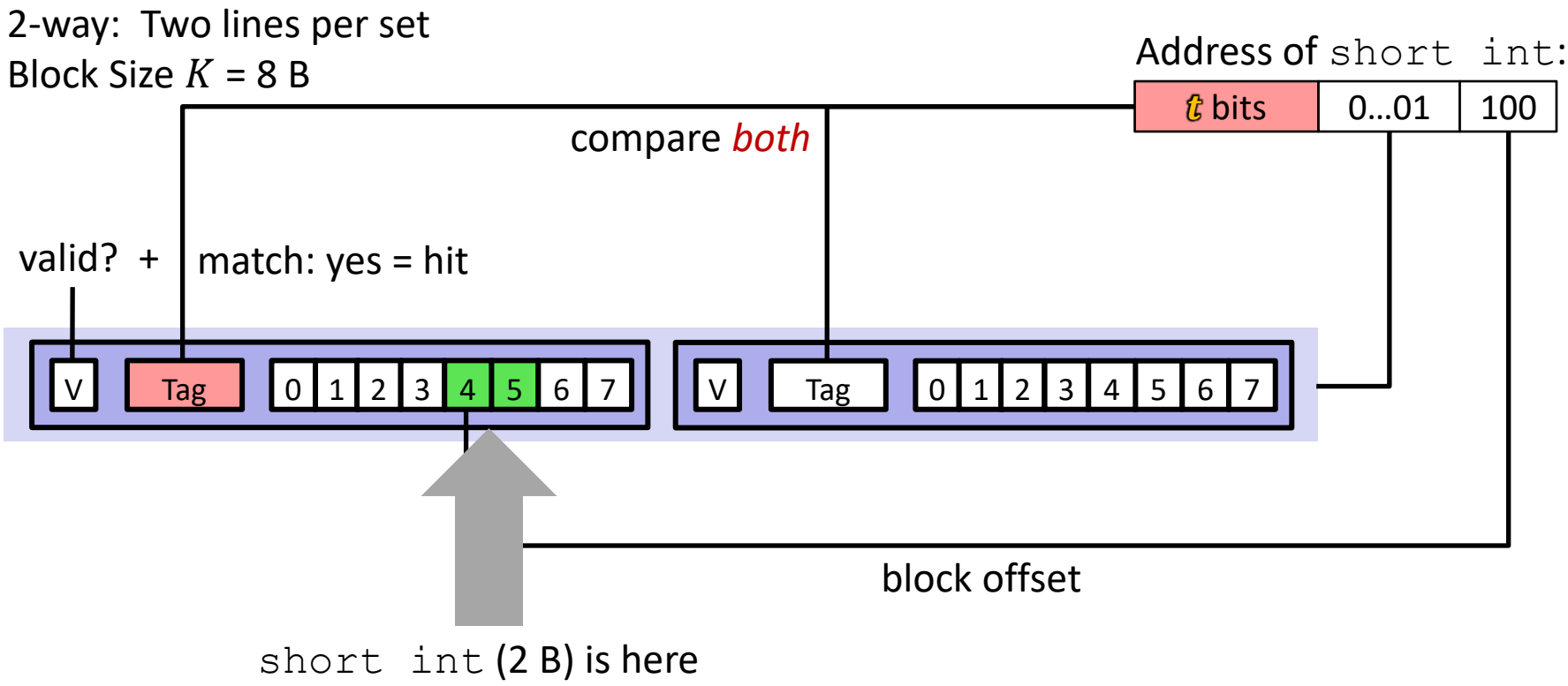


Example: Set-Associative Cache (E = 2)

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



Example: Set-Associative Cache (E = 2)



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];
```