Caches III

CSE 351 Winter 2019



Instructors:

Max Willsey

Luis Ceze

Teaching Assistants:

Britt Henderson

Lukas Joswiak

Josie Lee

Wei Lin

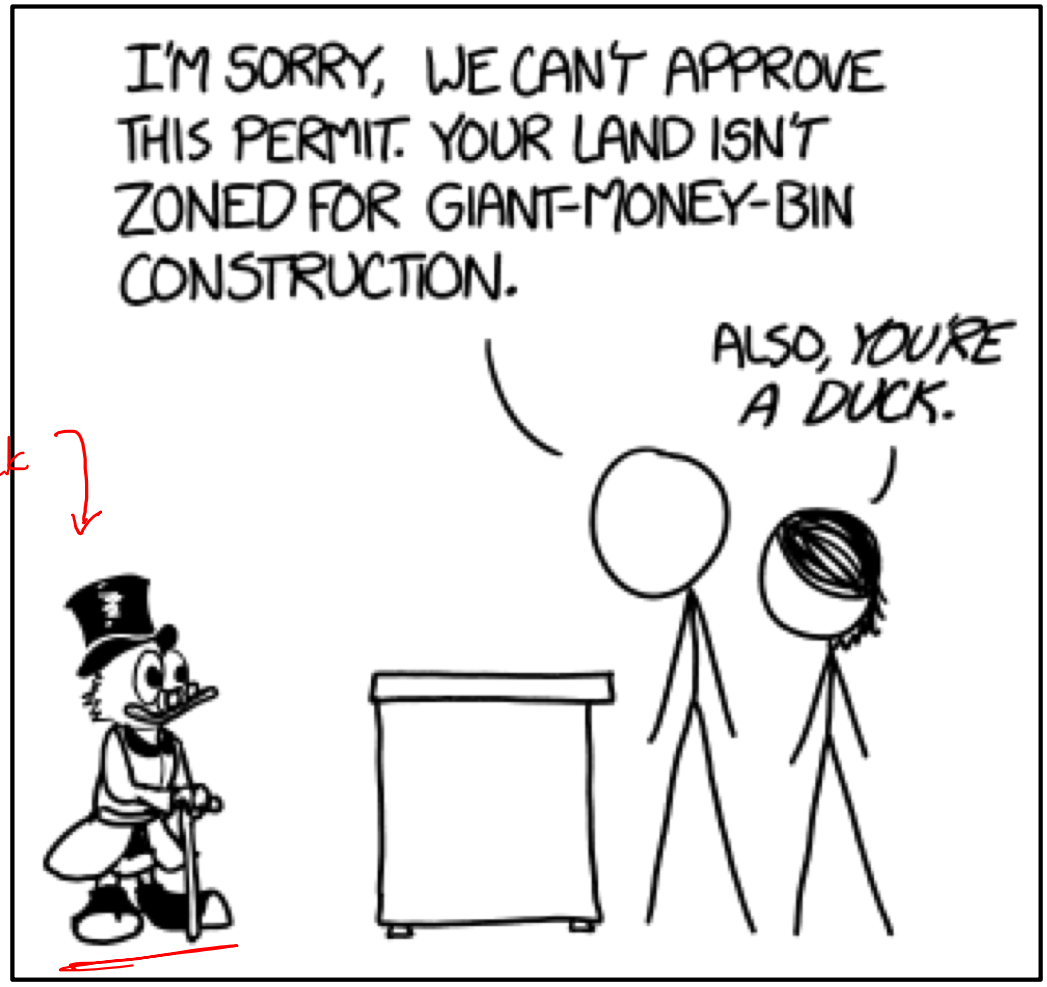
Daniel Snitkovsky

Luis Vega

Kory Watson

Ivy Yu

Scrooge McDuck



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

Administrivia

- Lab 3 due today (Friday)
- Lab 4 out this weekend

- HW 4 is released, due next Friday (03/01)
- Last day for regrade requests!

- Extra OH today (Ivy @ 9:30)
  - Me – 12:00-1:30pm – CSE 280
  - Lukas – 2:30pm-close – 2nd floor breakout

- Cache sim!

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

Checking for a Requested Address

- CPU sends *address* request for chunk of data
  - Address and requested data are not the same thing!
    - Analogy: your friend ≠ his or her phone number

index tag s

- TIO address breakdown:

address

m-bit address:

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

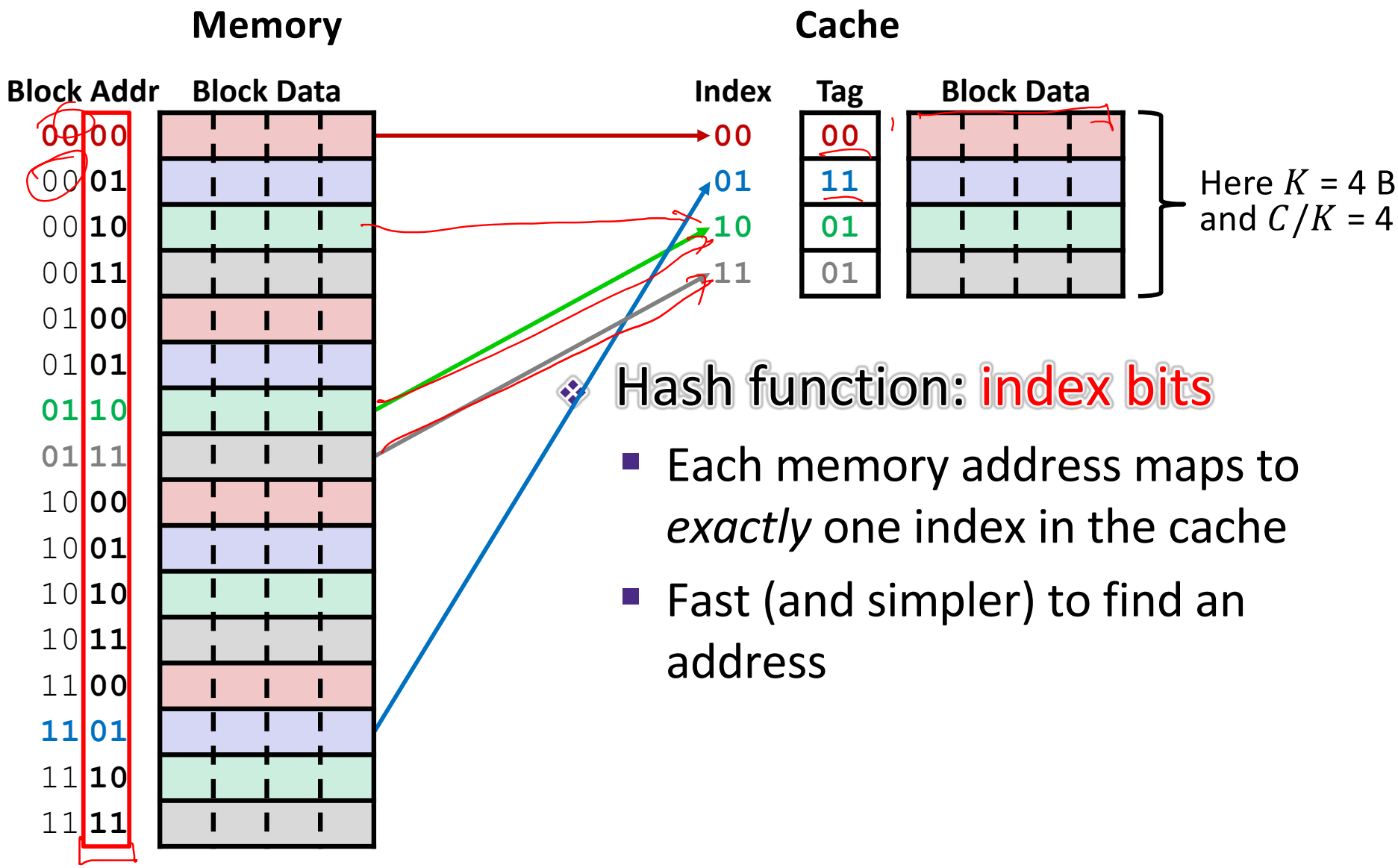
Block Number

K

  - **Index** field tells you where to look in cache
  - **Tag** field lets you check that data is the block you want
  - **Offset** field selects specified start byte within block

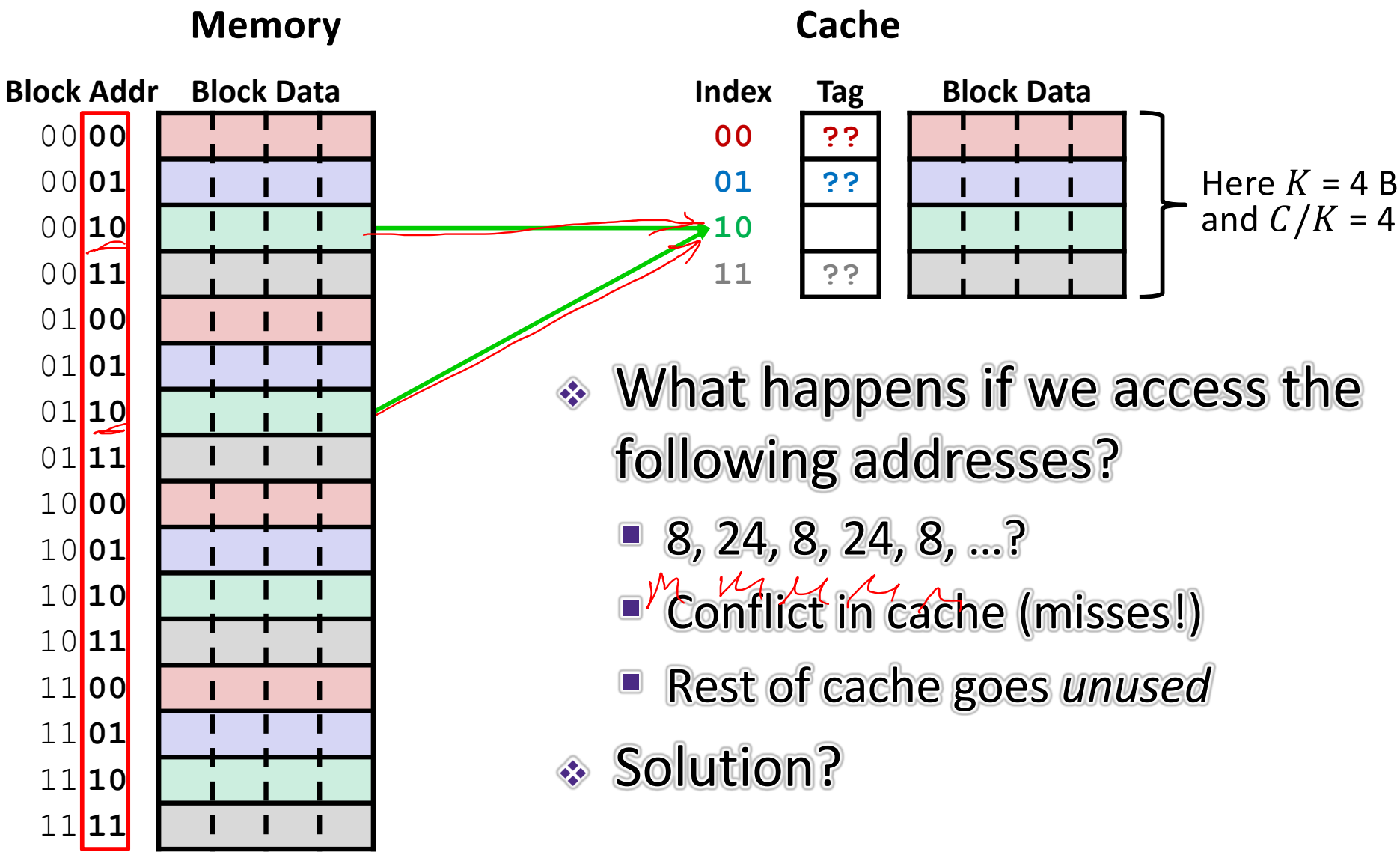
  - **Note:** t and s sizes will change based on hash function

Direct-Mapped Cache



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

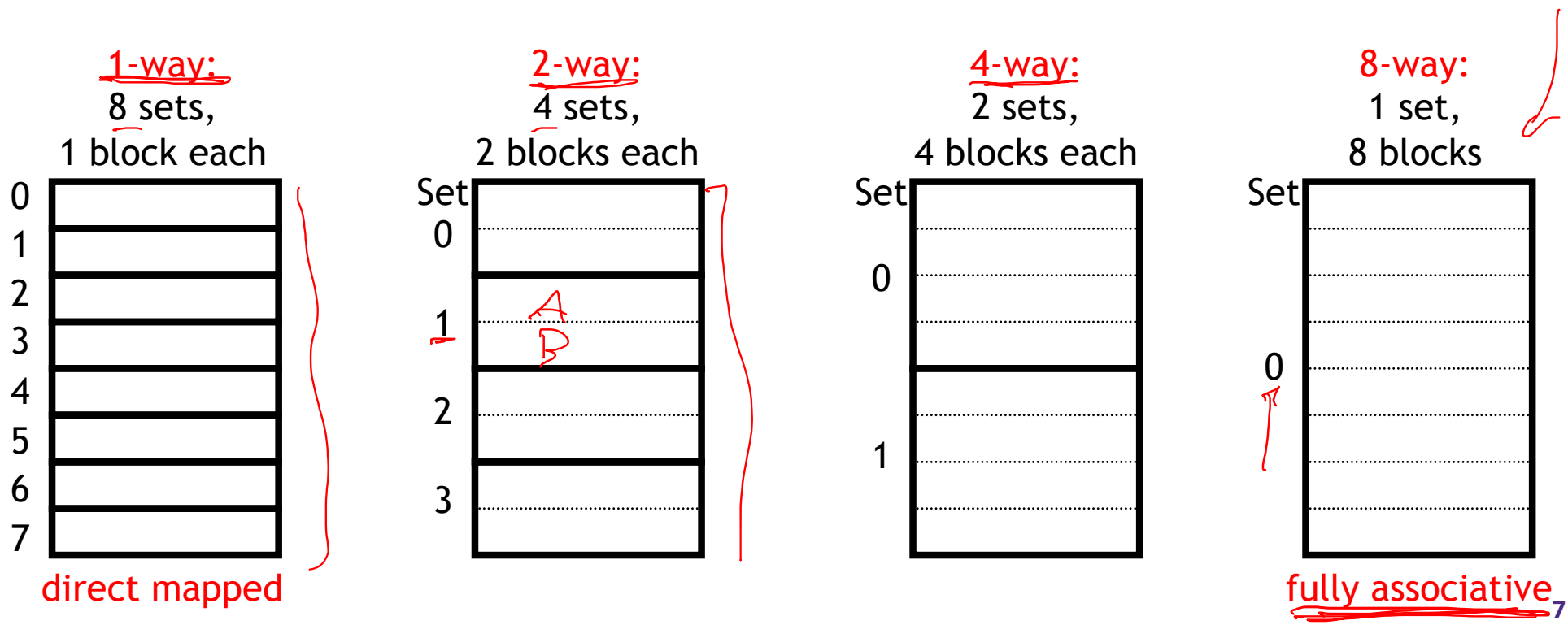
Direct-Mapped Cache Problem



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

Associativity

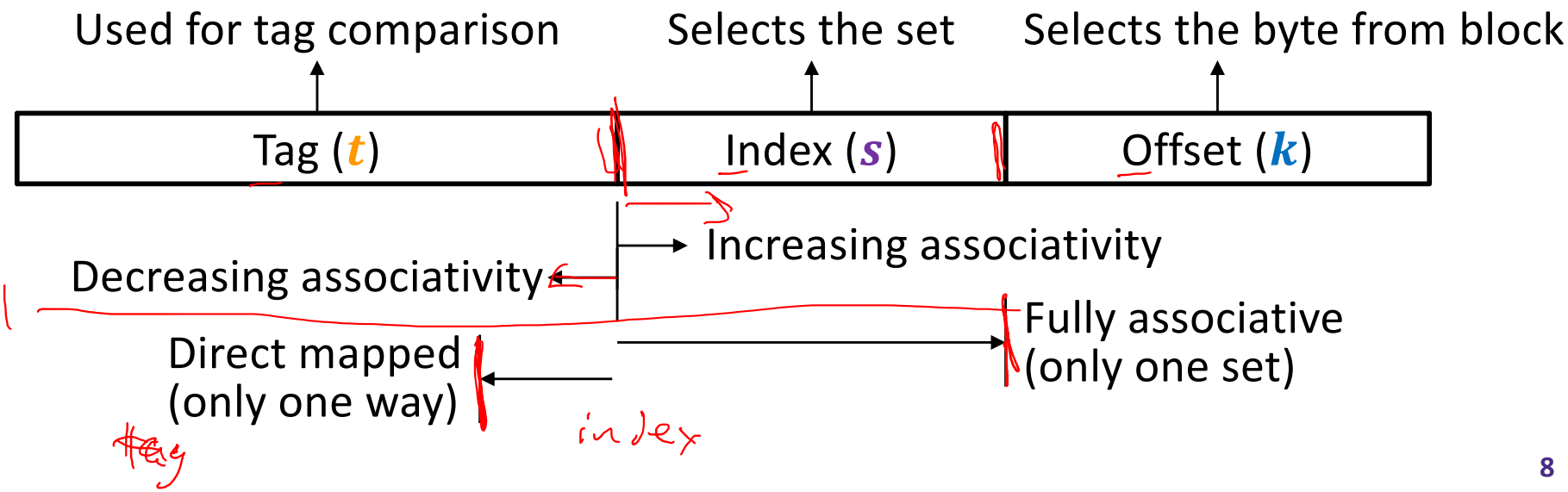
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**



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

k=4

K

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

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

offset

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

m-bit address:	Tag (t)	Index (s)	Offset (k)

E=1 s=3

s = ?

Direct-mapped

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

E=2 s=2

s = ?

2-way set associative

Set	Tag	Data
0		
1		
2		
3		✓ ✓

E=4

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*")

Direct-mapped

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

2-way set associative

Set	Tag	Data
0		
1		
2		
3		

4-way set associative

Set	Tag	Data
0		
1		

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?

  $C = 2^{11}$ $K = 2^{7}$

  $2^{4} = 16$ blocks

  A. 2

  B. 4

  C. 8 -way

  D. 16

  E. We're lost…

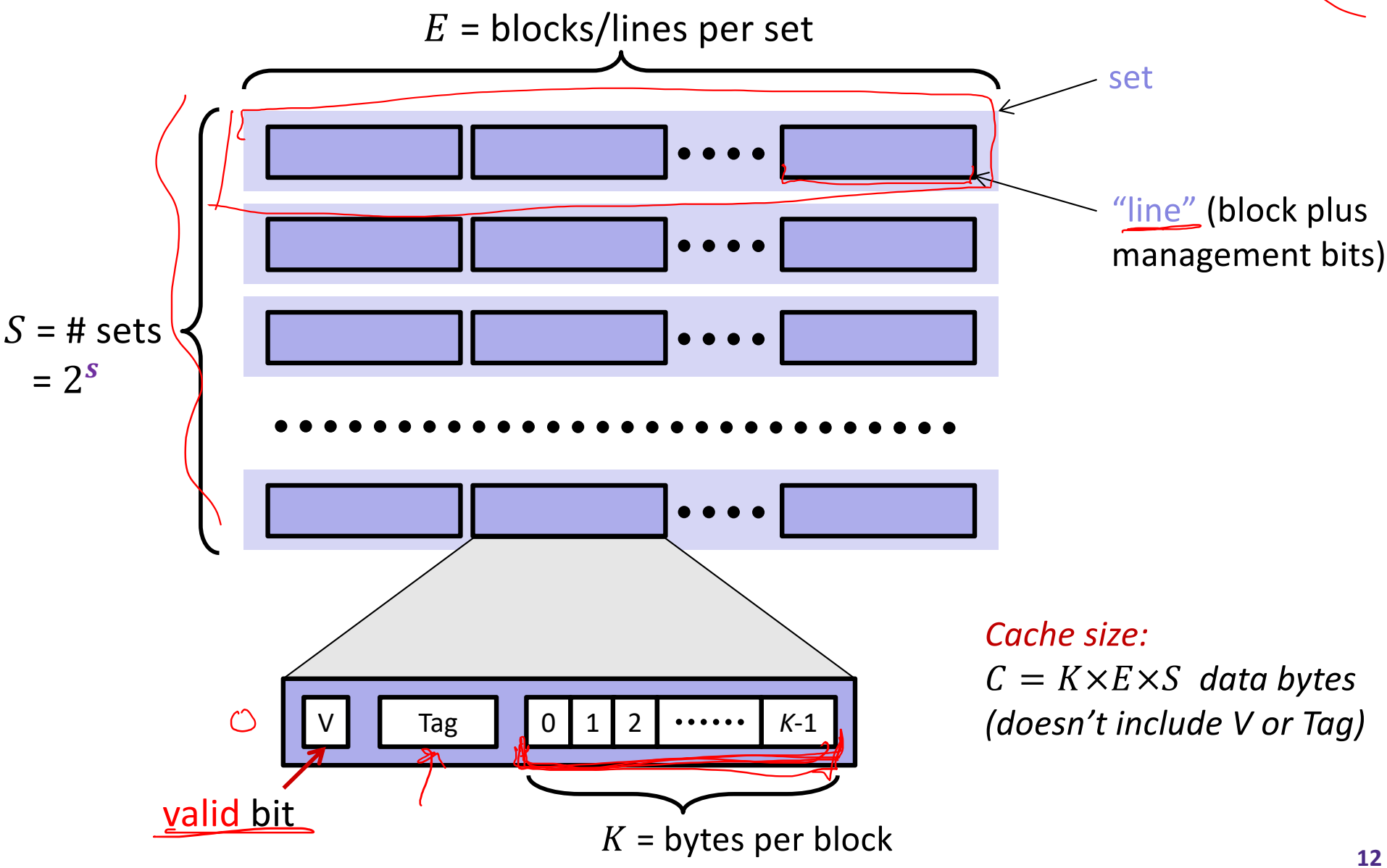
tag	index	offset
8	1	7

Set

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

General Cache Organization (S, E, K)

= C



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	m	
Tag field width	t	
Index field width	s	
Offset field width	k (b in book)	

Example Cache Parameters Problem

2^{12} 12 bit

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

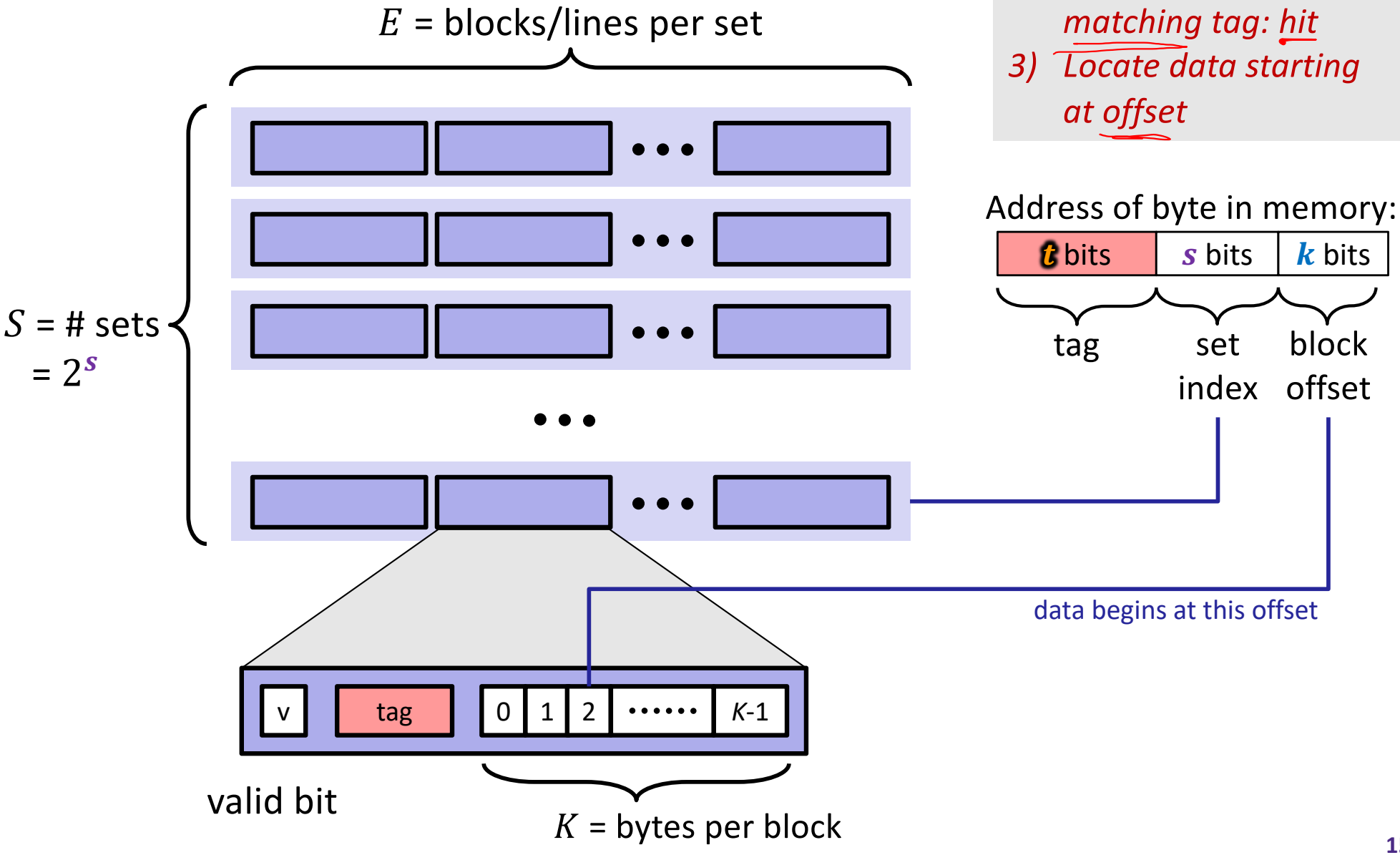
Cache Size	256 B	= C
Block Size	32 B	= K
Associativity	2-way	= E
Hit Time	3 cycles	
Miss Rate	20%	
Tag Bits	5	
Index Bits	2	
Offset Bits	5	
AMAT	28	

8 blocks

$3 + .2(125)$

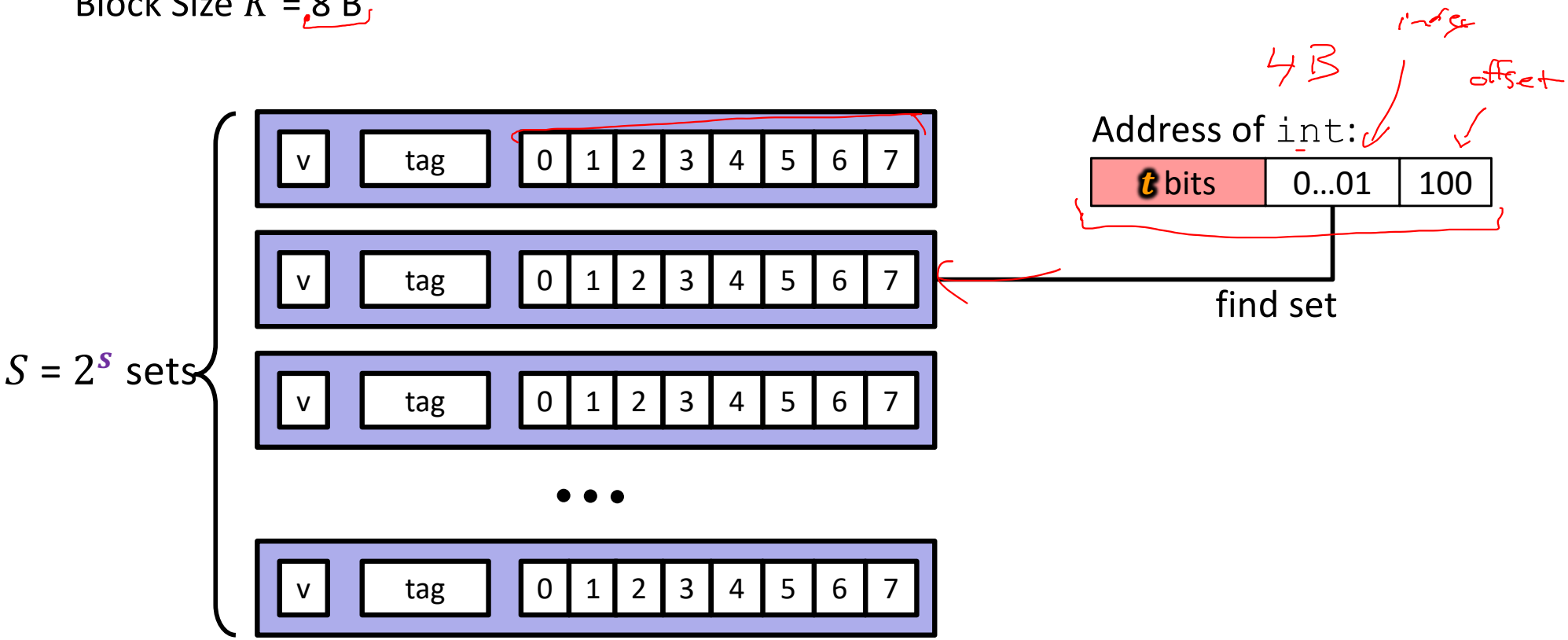
Cache Read

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



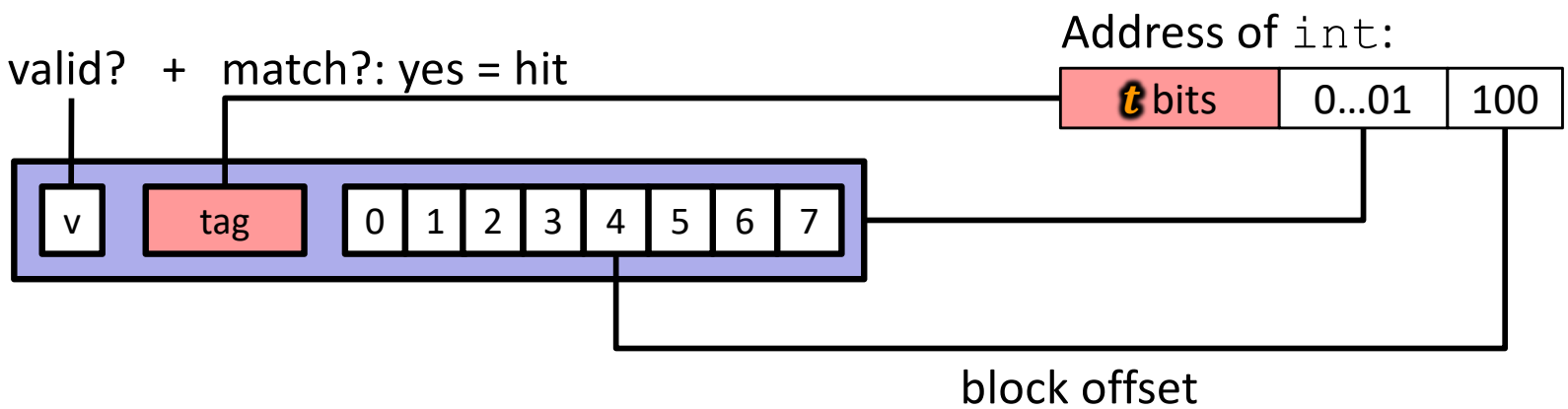
Example: Direct-Mapped Cache (E = 1)

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



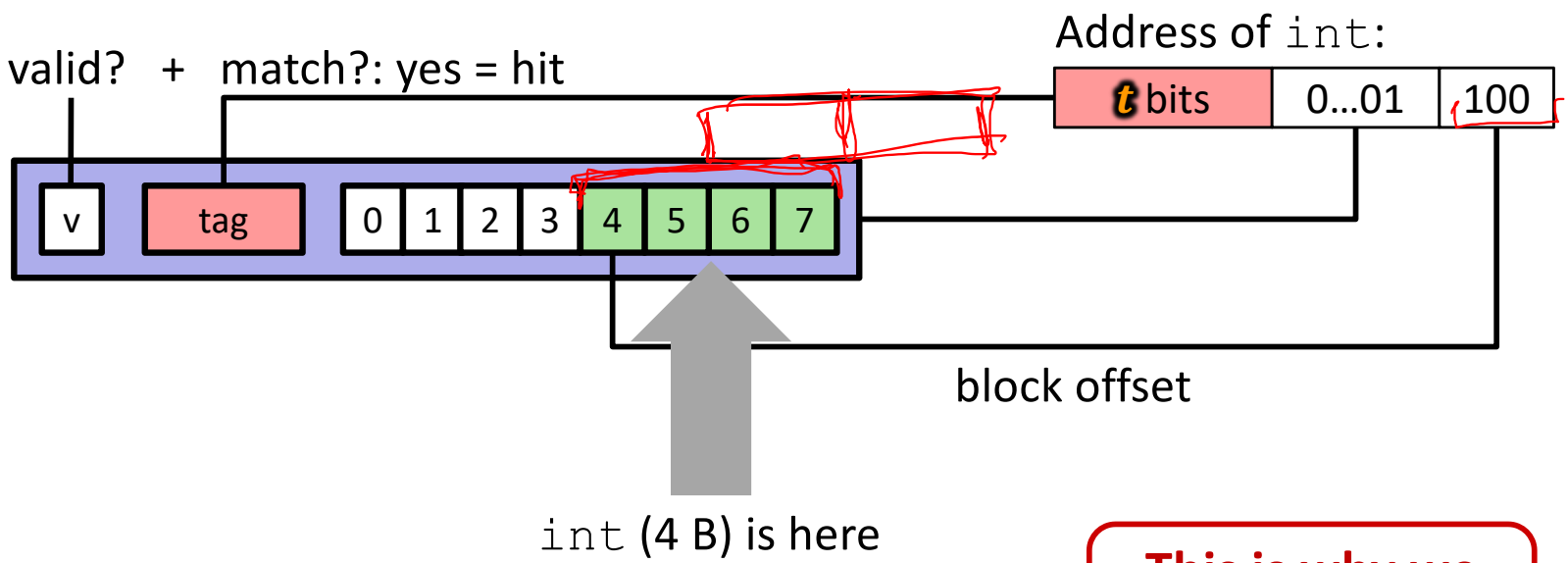
Example: Direct-Mapped Cache (E = 1)

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)

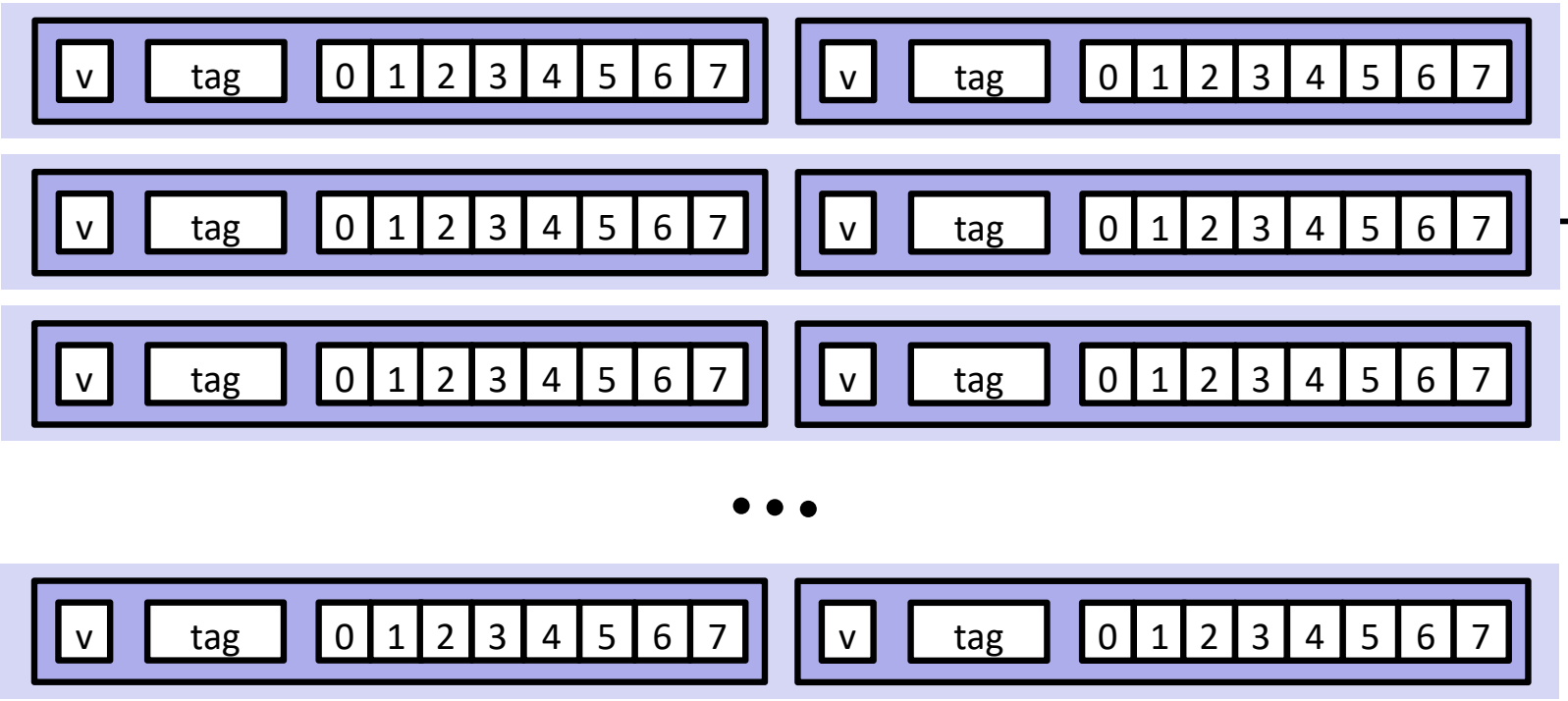
2-way: Two lines per set
Block Size K = 8 B

2 B

Address of `short int`:

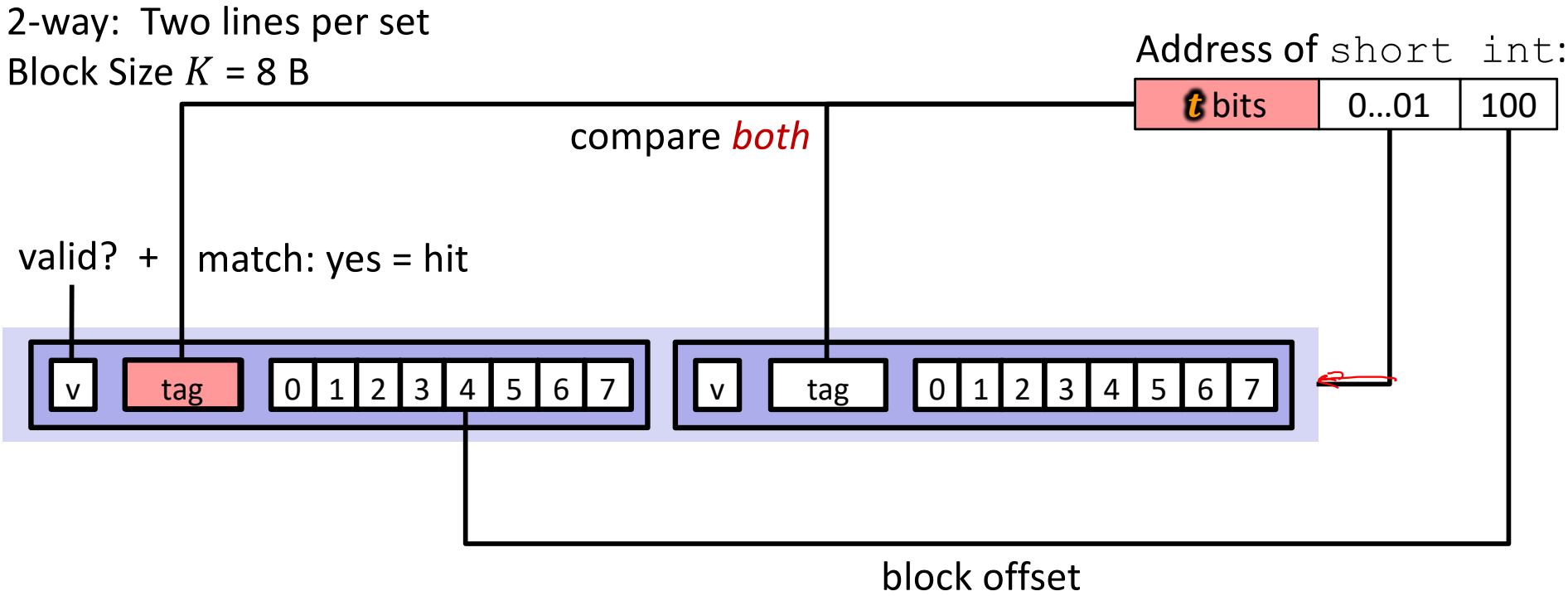
t bits	0...01	100

find set



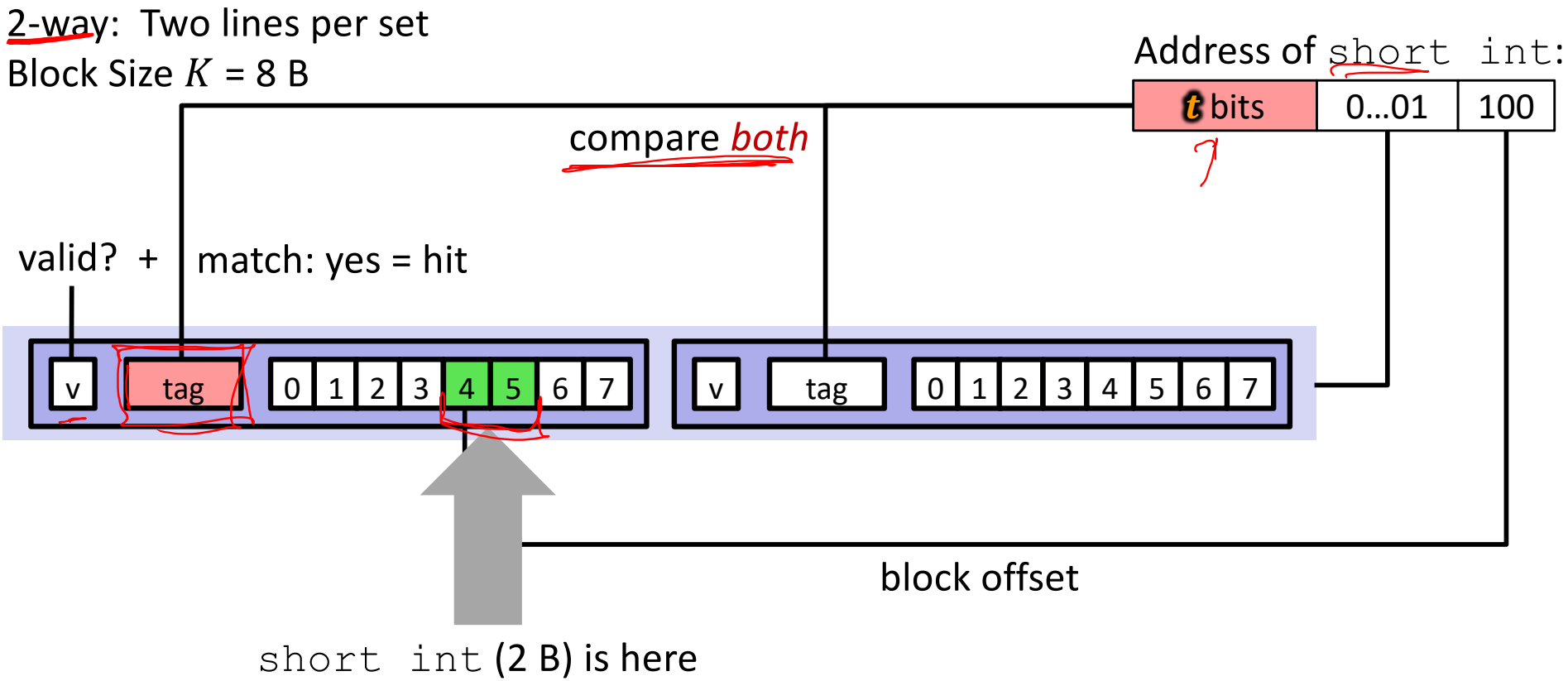
Example: Set-Associative Cache (E = 2)

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

      8, 24, 8, 24
  - Direct-mapped caches have more conflict misses than E-way set-associative (where $E > 1$)
  - **Note:** *Fully-associative* only has Compulsory and Capacity misses
- 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*)

Example Code Analysis Problem

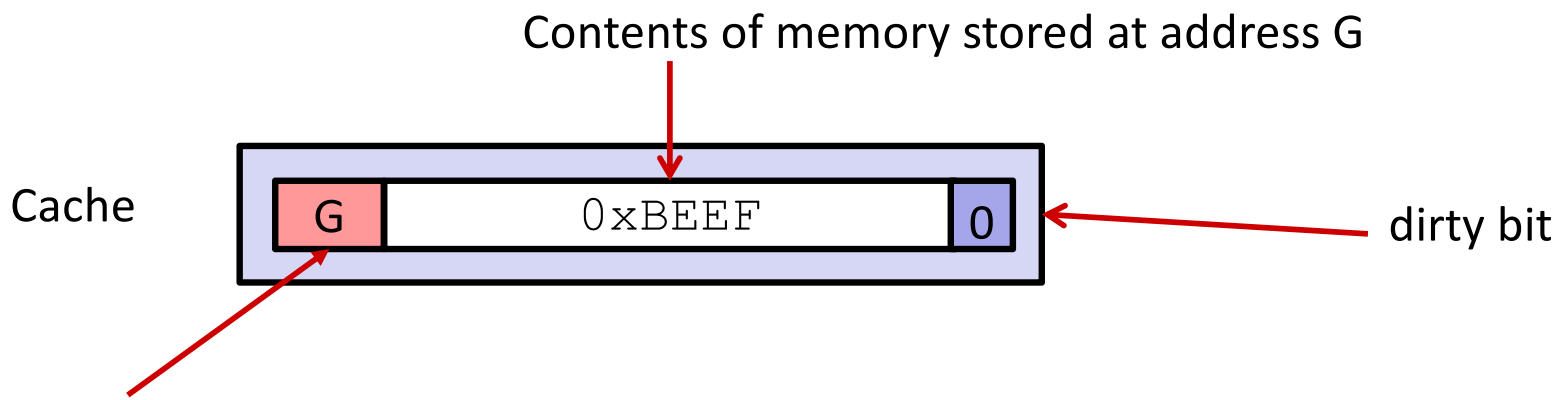
- Assuming the cache starts cold (all blocks invalid) and `sum` is stored in a register, calculate the **miss rate**:
  - m = 12 bits, C = 256 B, K = 32 B, E = 2

```
#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];
```

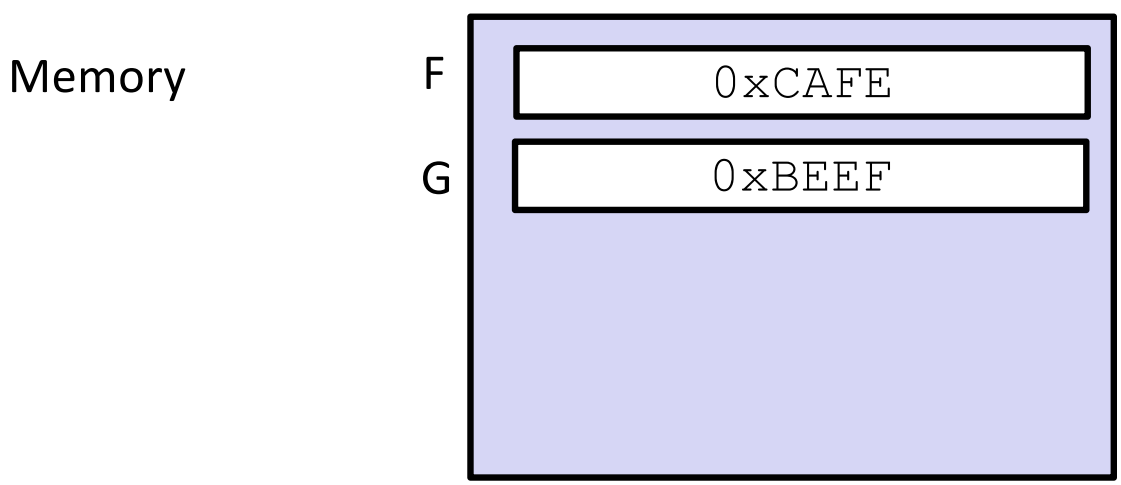
What about writes?

- Multiple copies of data exist:
  - L1, L2, possibly L3, main memory
- What to do on a write-hit?
  - 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*")
- What to do on a write-miss?
  - 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
- Typical caches:
  - Write-back + Write-allocate, usually
  - Write-through + No-write-allocate, occasionally

Write-back, write-allocate example



tag (there is only one set in this tiny cache, so the tag is the entire block address!)

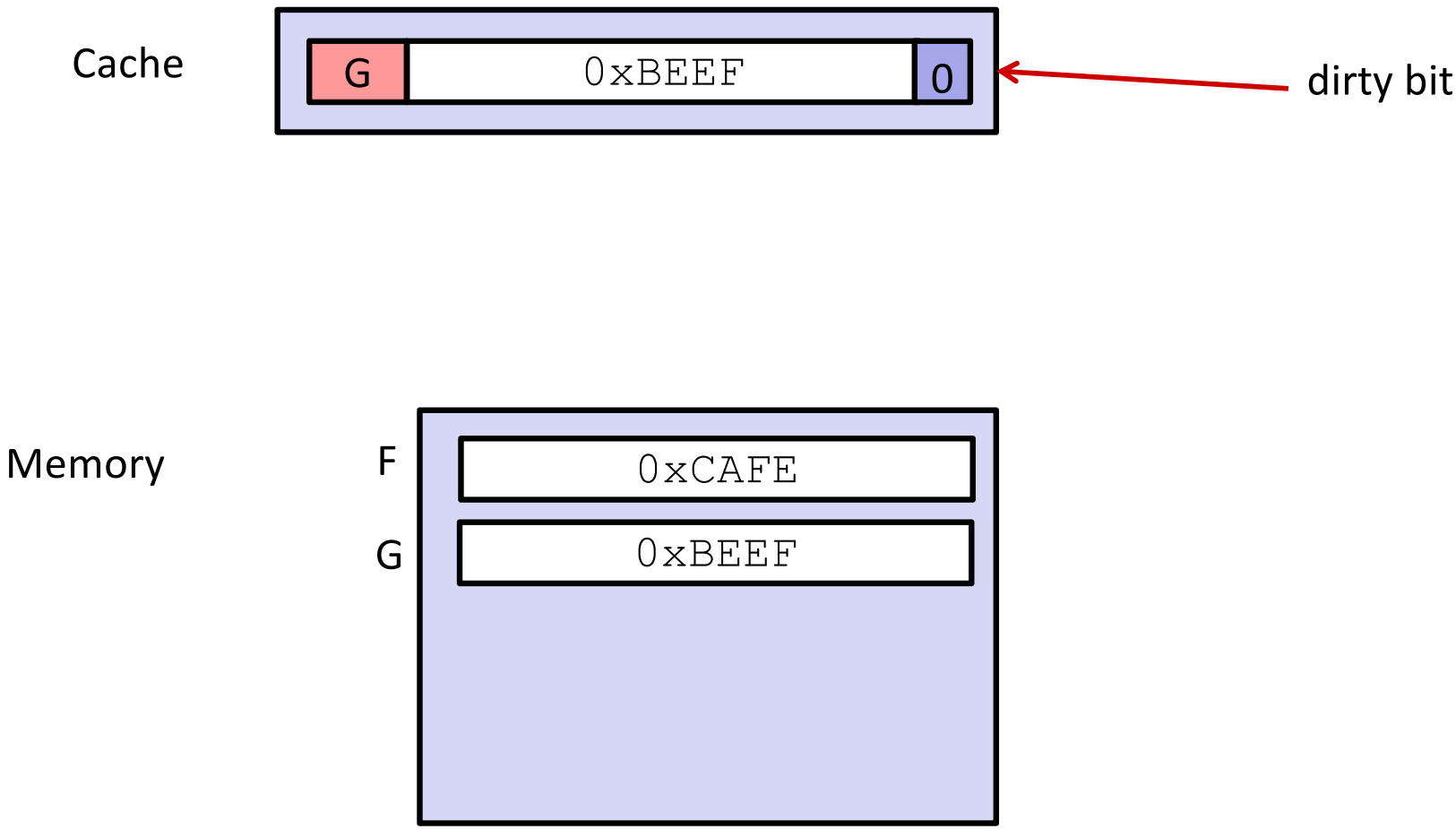


In this example we are sort of ignoring block offsets. Here a block holds 2 bytes (16 bits, 4 hex digits).

Normally a block would be much bigger and thus there would be multiple items per block. While only one item in that block would be written at a time, the entire line would be brought into cache.

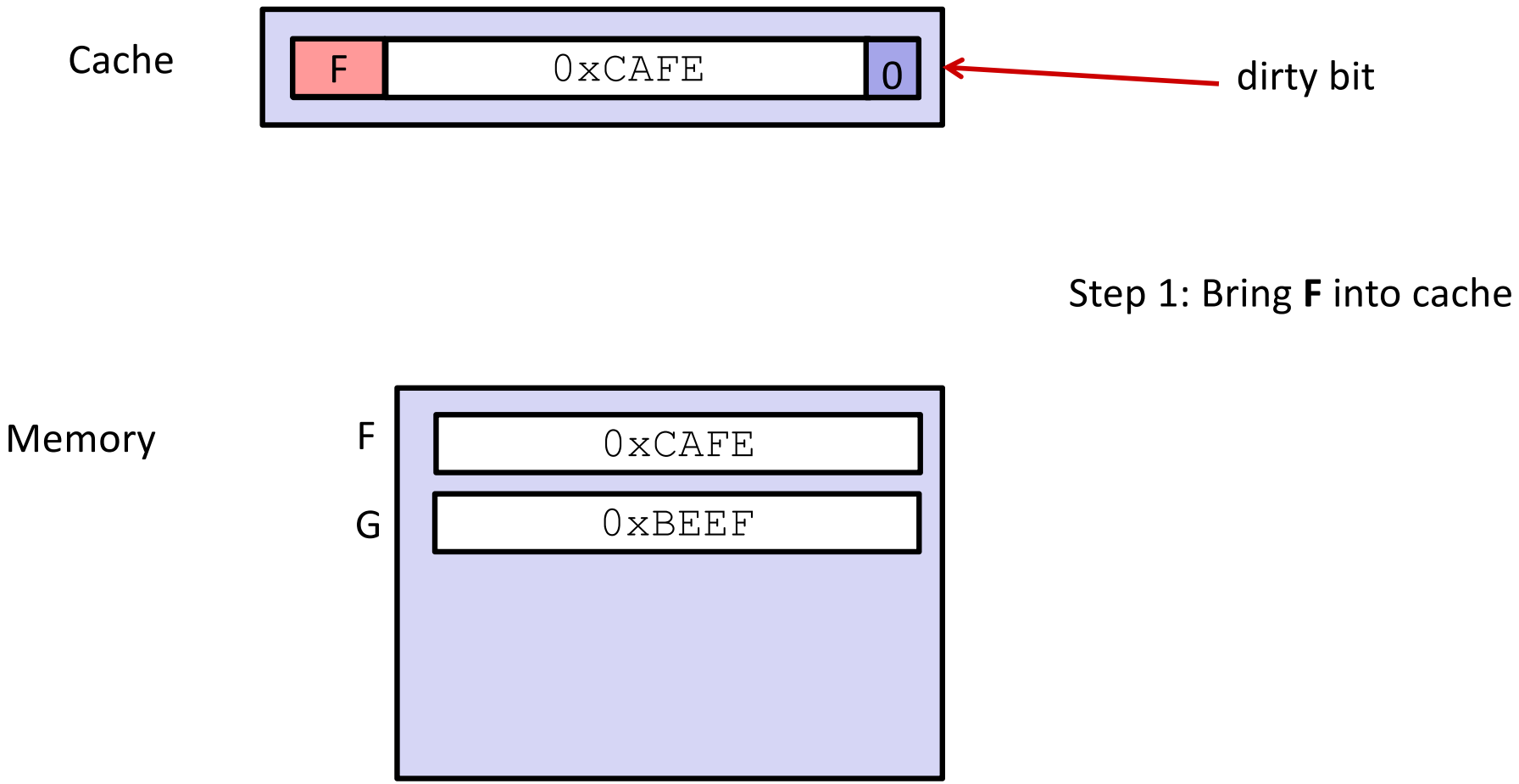
Write-back, write-allocate example

```
mov 0xFACE, F
```



Write-back, write-allocate example

```
mov 0xFACE, F
```

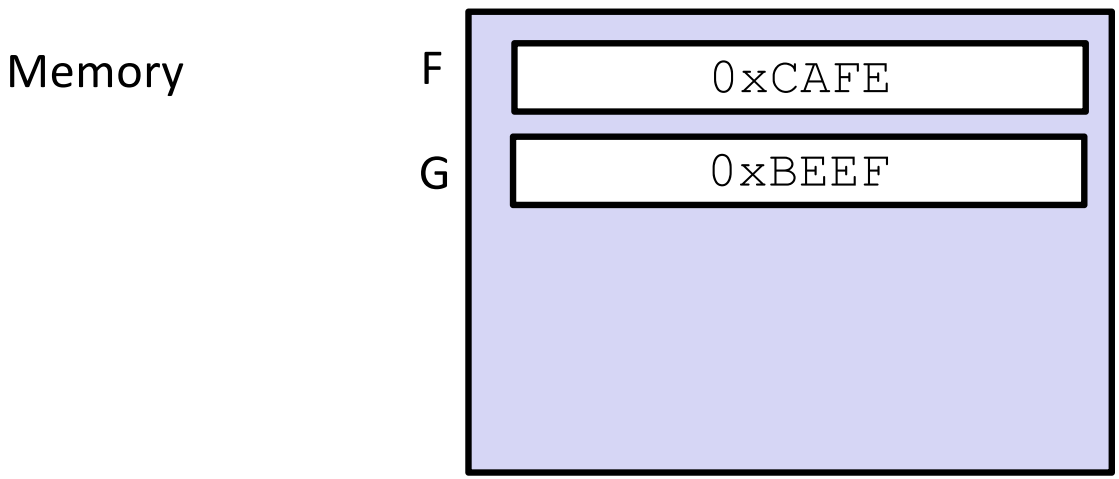


Step 1: Bring **F** into cache

Write-back, write-allocate example

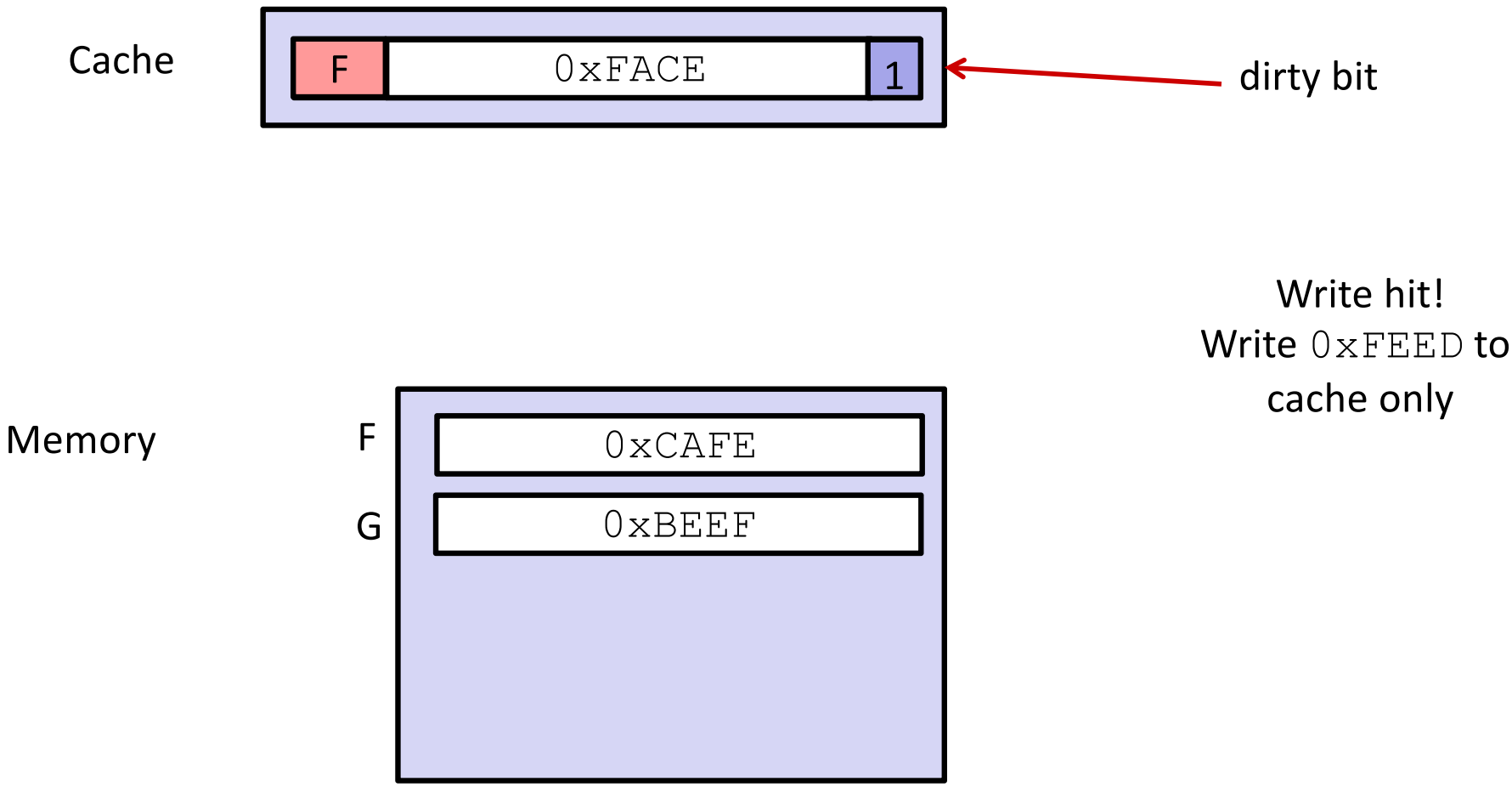
```
mov 0xFACE, F
```

Step 2: Write `0xFACE` to cache only and set dirty bit



Write-back, write-allocate example

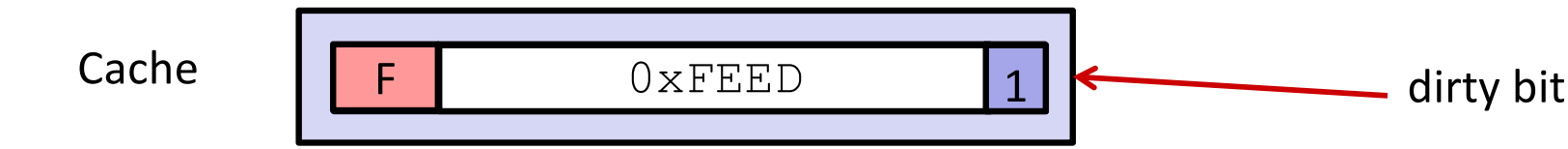
`mov 0xFACE, F` `mov 0xFEED, F`

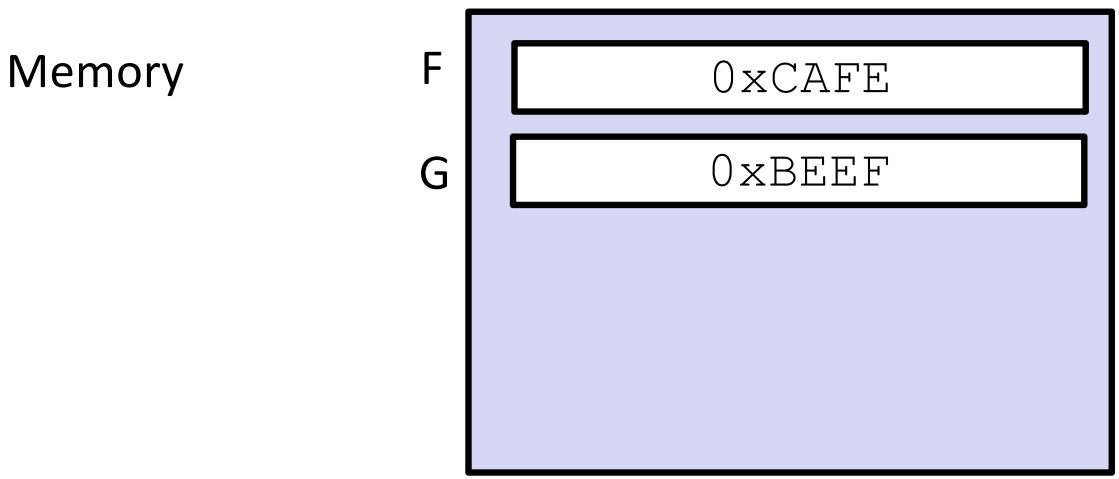


Write hit!
Write `0xFEED` to cache only

Write-back, write-allocate example

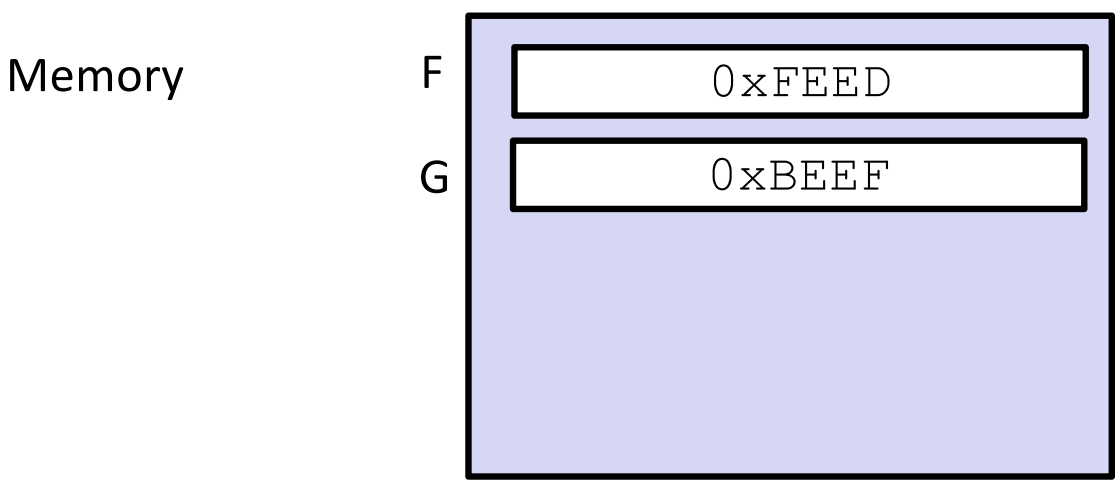
```
mov 0xFACE, F      mov 0xFEED, F          mov G, %rax
```





Write-back, write-allocate example

`mov 0xFACE, F` `mov 0xFEED, F` `mov G, %rax`



1. Write **F** back to memory since it is dirty
2. Bring **G** into the cache so we can copy it into `%rax`

Peer Instruction Question

- Which of the following cache statements is FALSE?
  - A. We can reduce compulsory misses by decreasing our block size
  - B. We can reduce conflict misses by increasing associativity
  - C. A write-back cache will save time for code with good temporal locality on writes
  - D. A write-through cache will always match data with the memory hierarchy level below it
  - E. We're lost…