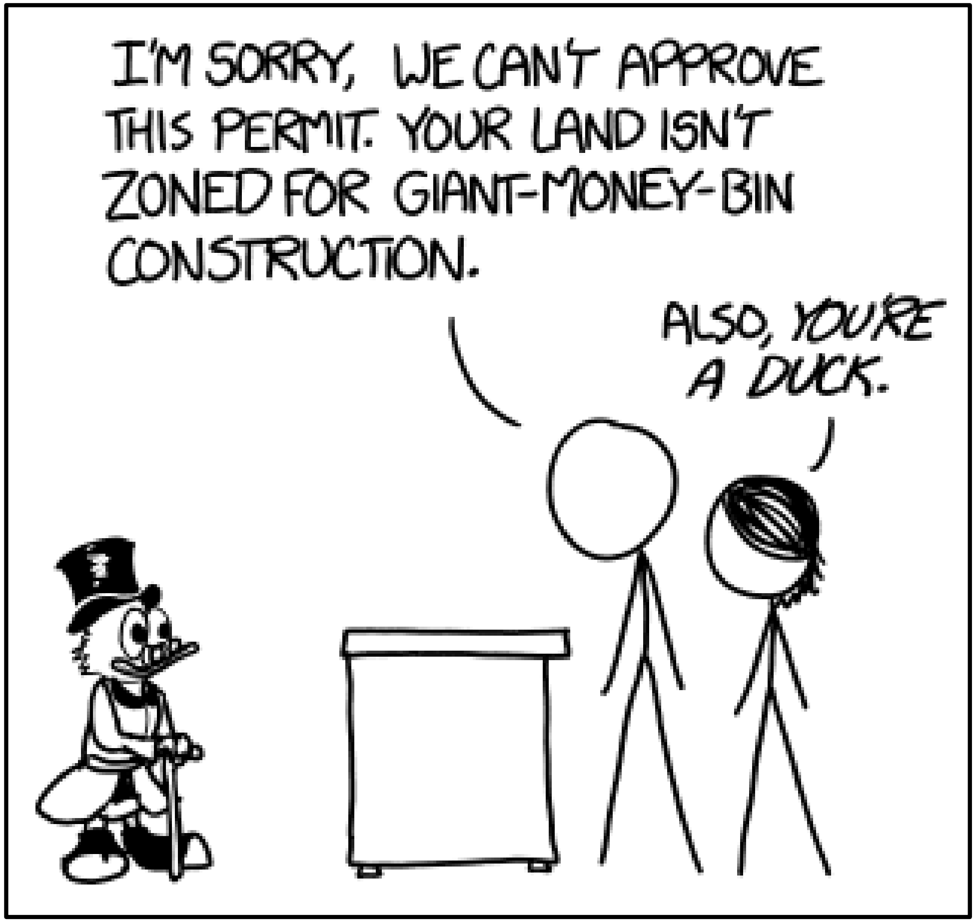
Caches III

CSE 351 Spring 2018



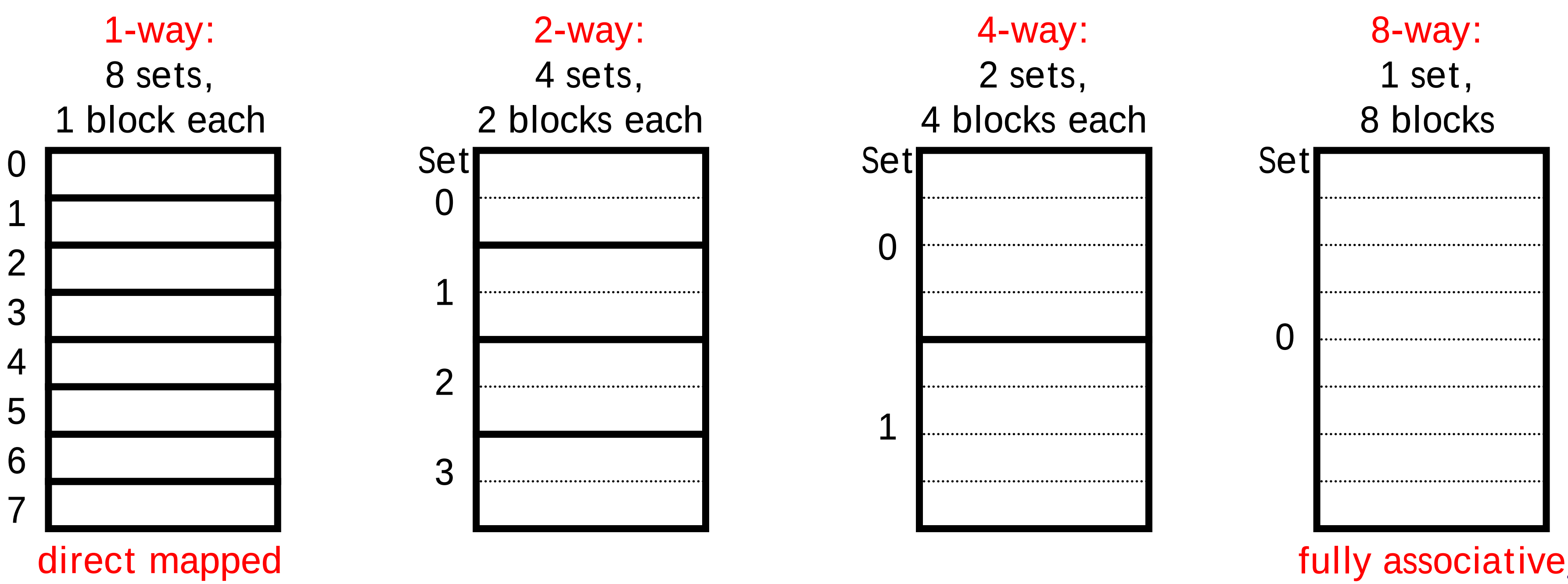
https://what-if.xkcd.com/111/

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

Associativity

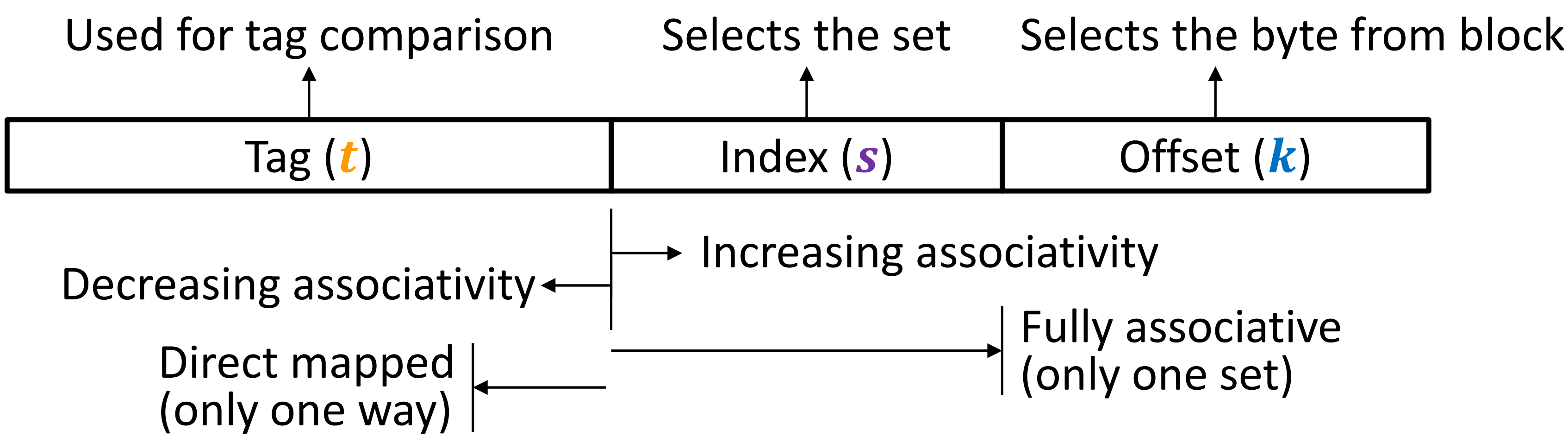
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**



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 $C/K/E$
  - 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`

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

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*")

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?

  A. 2

  B. 4

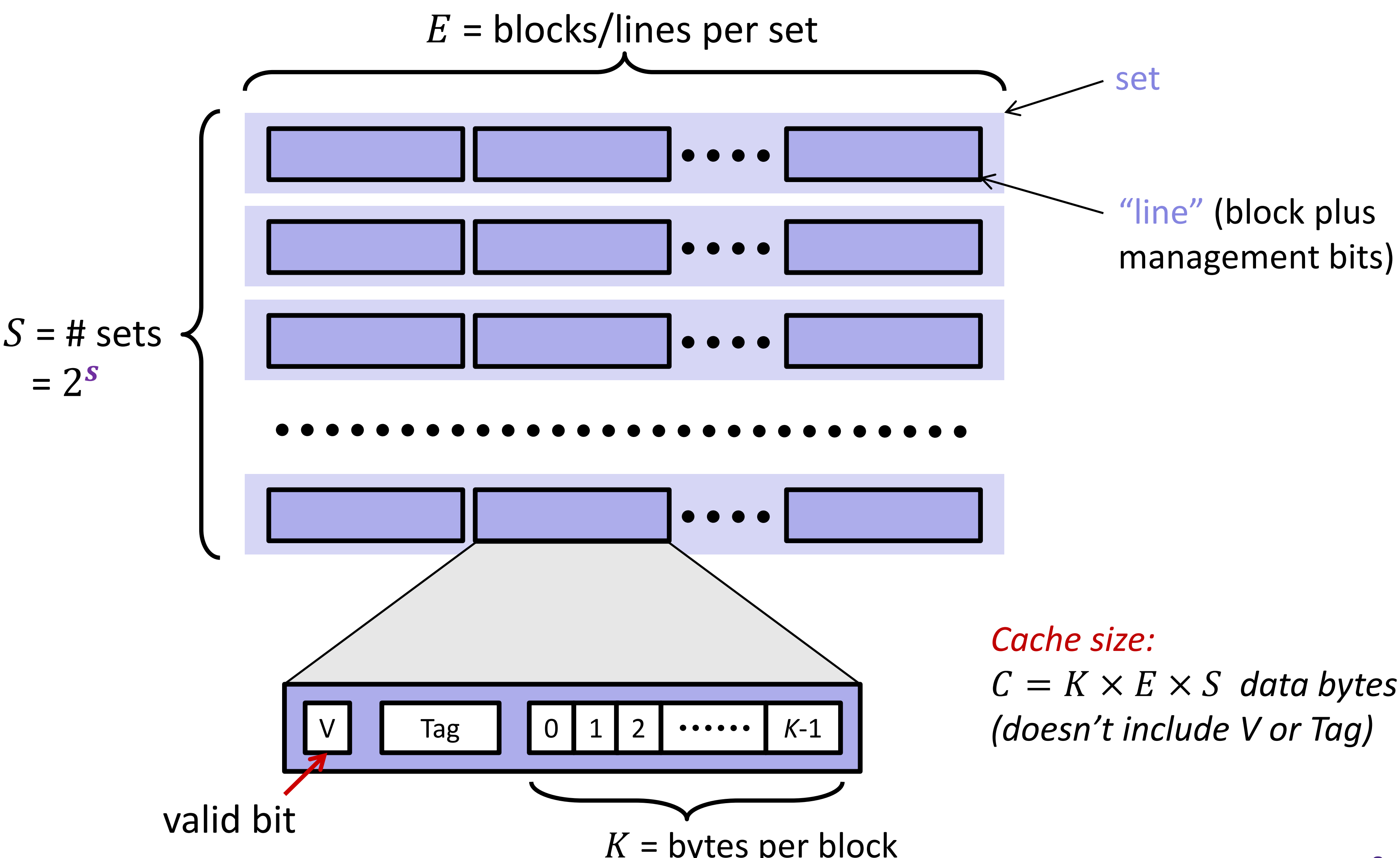
  C. 8

  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)

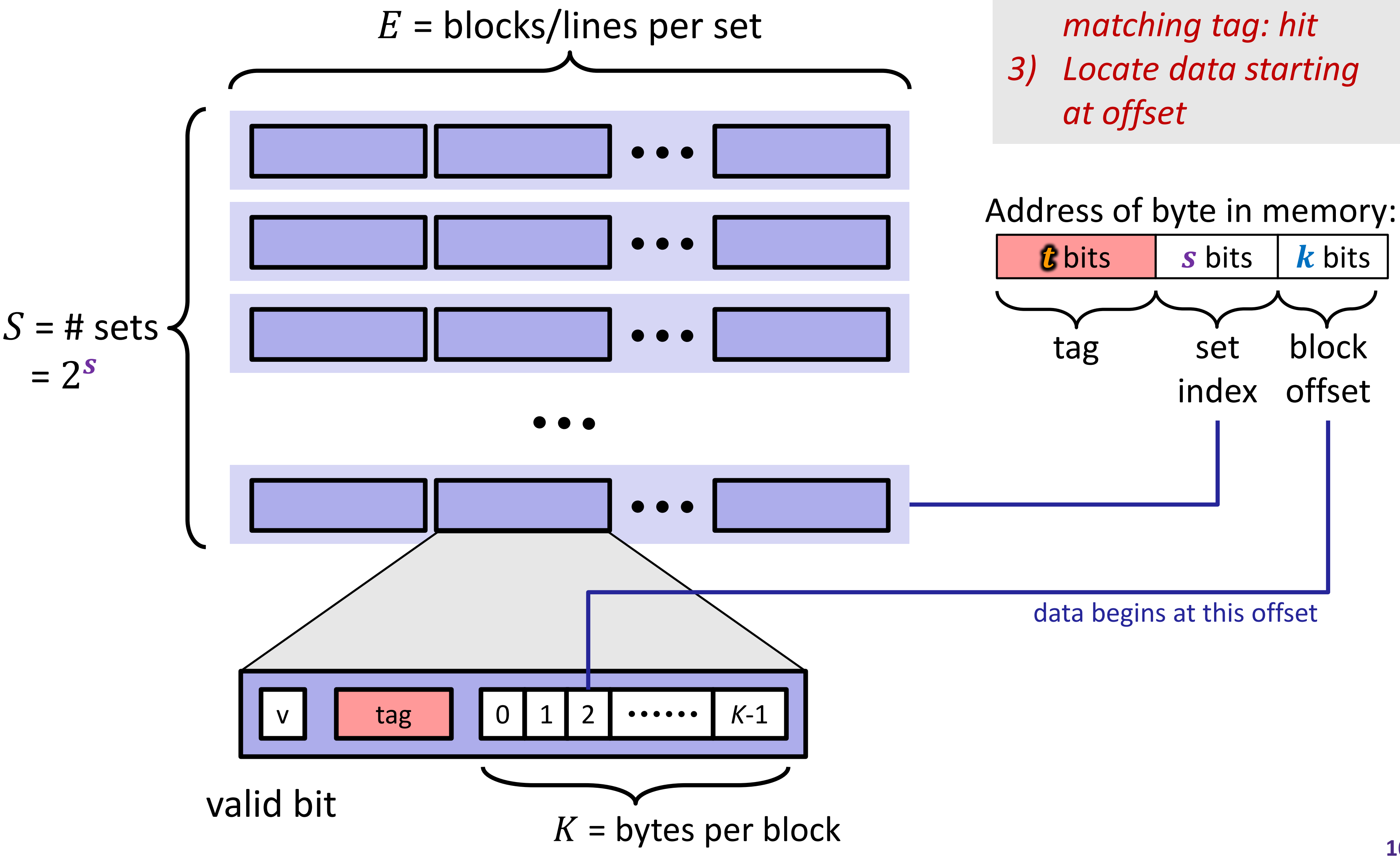
Notation Review

- 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

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)	

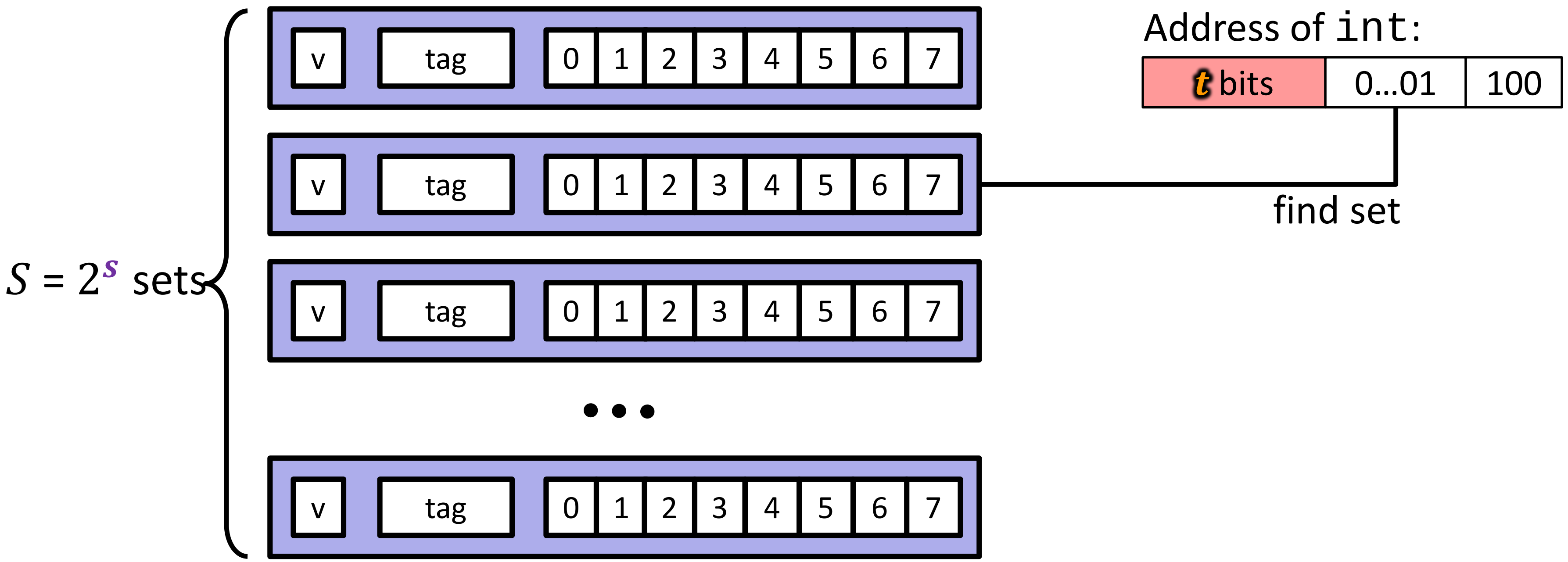
Cache Read

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



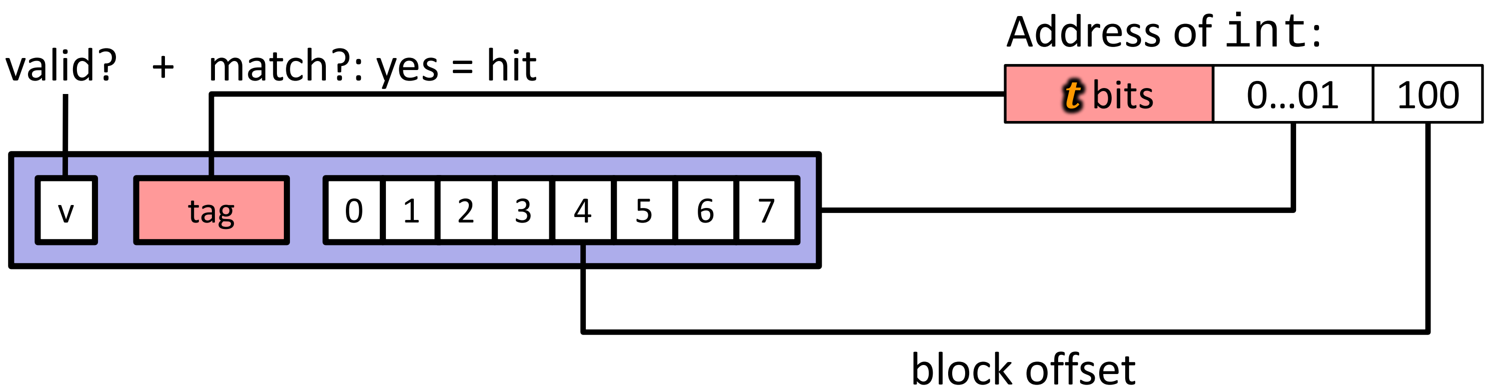
Example: Direct-Mapped Cache (E = 1)

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



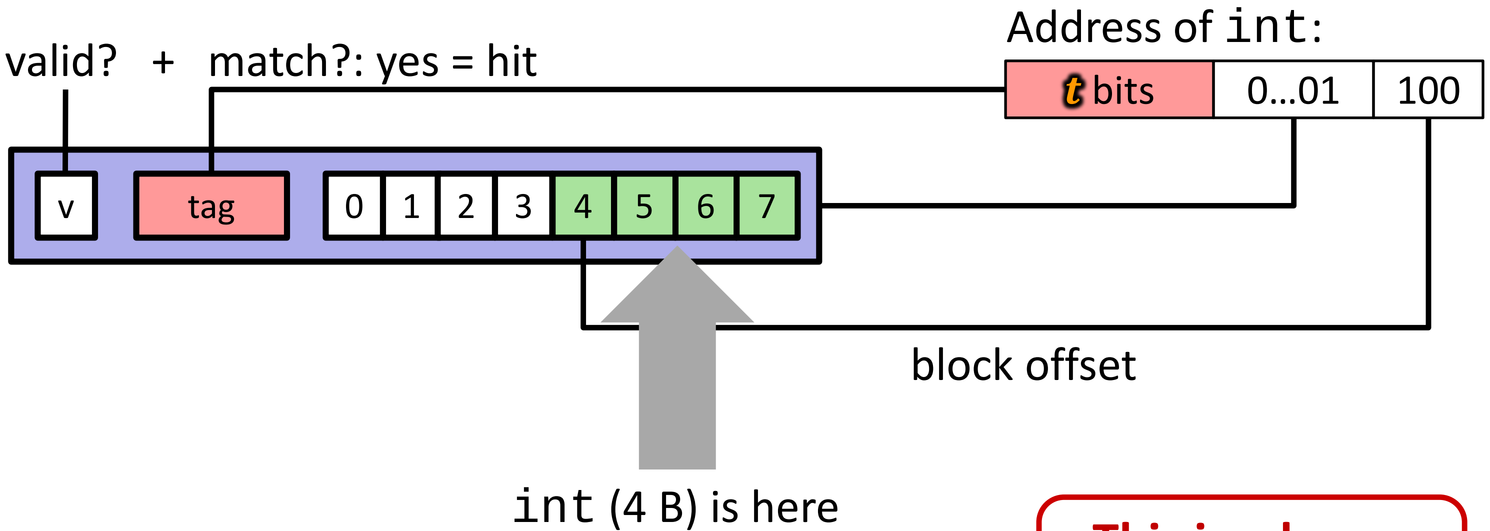
Example: Direct-Mapped Cache (E = 1)

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

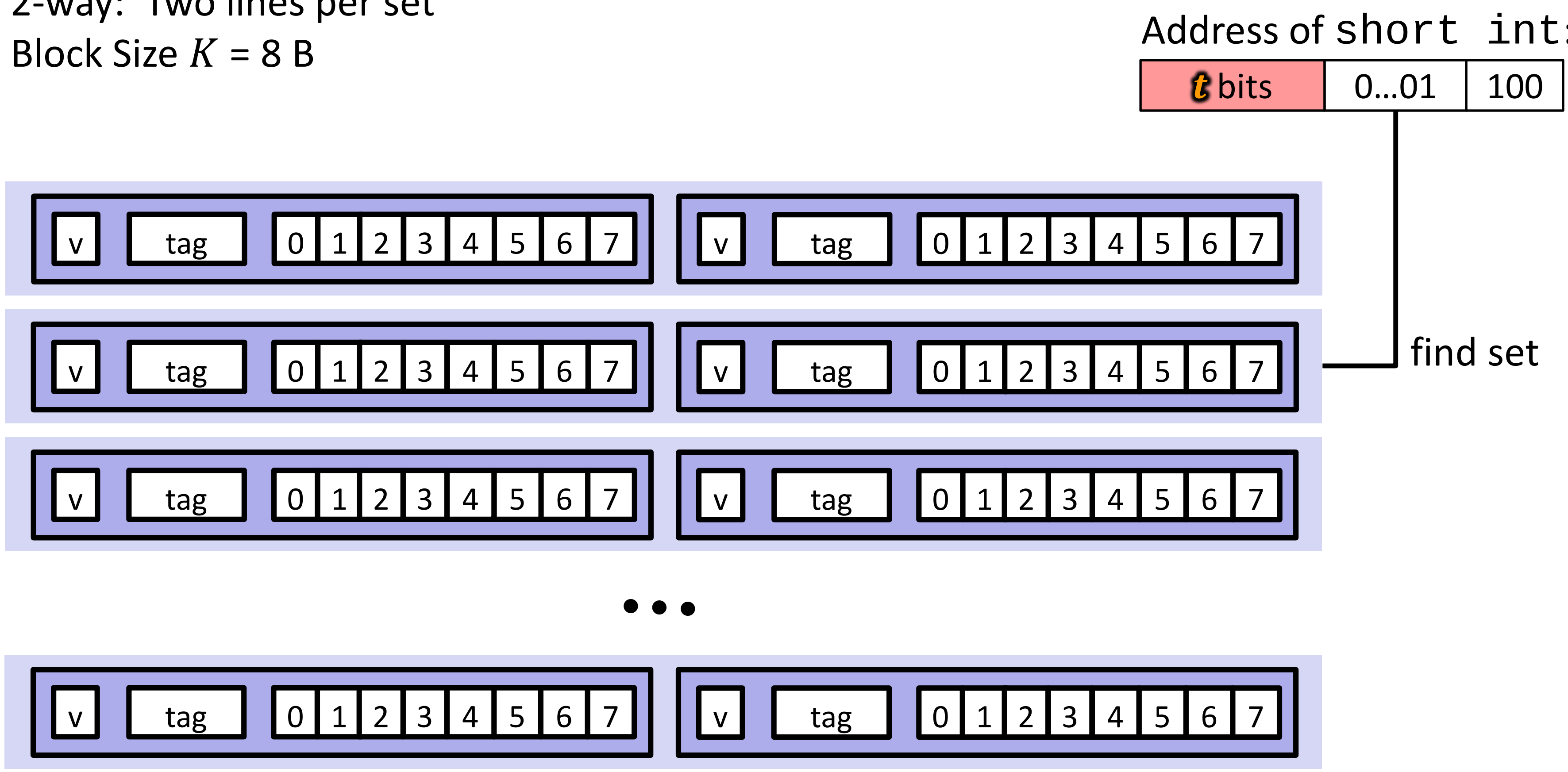


This is why we want alignment!

No match? Then old line gets evicted and replaced

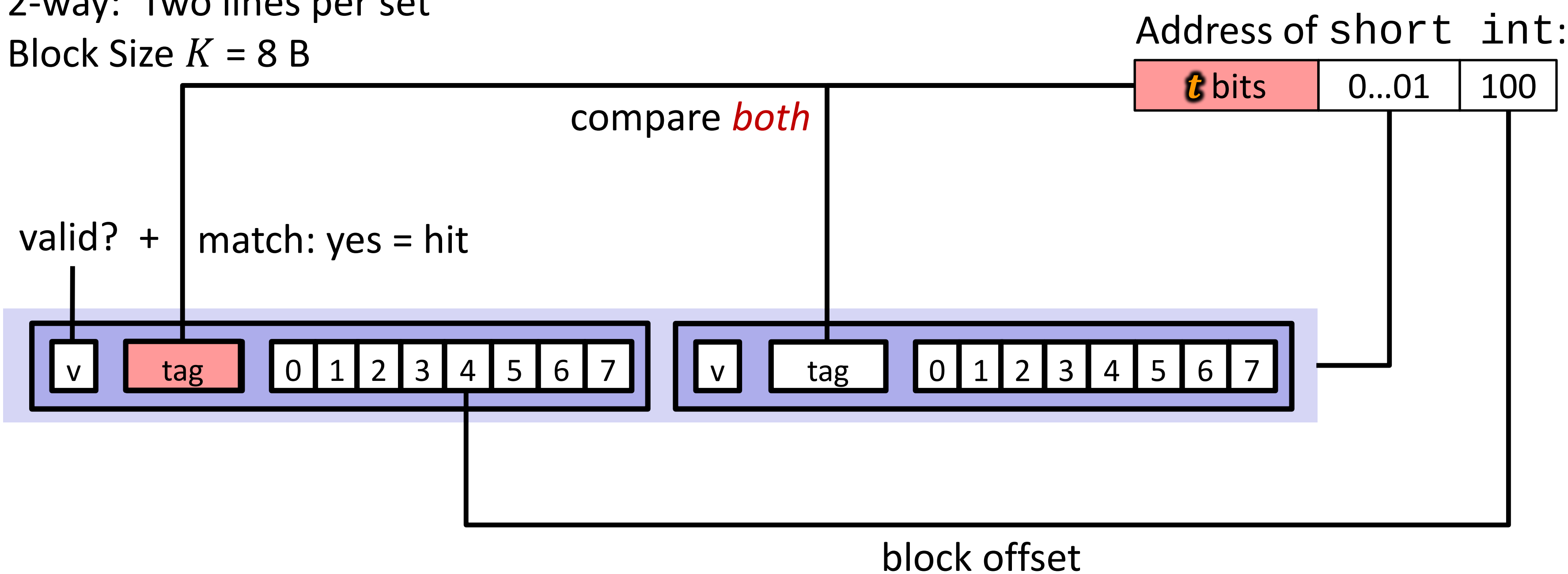
Example: Set-Associative Cache (E = 2)

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



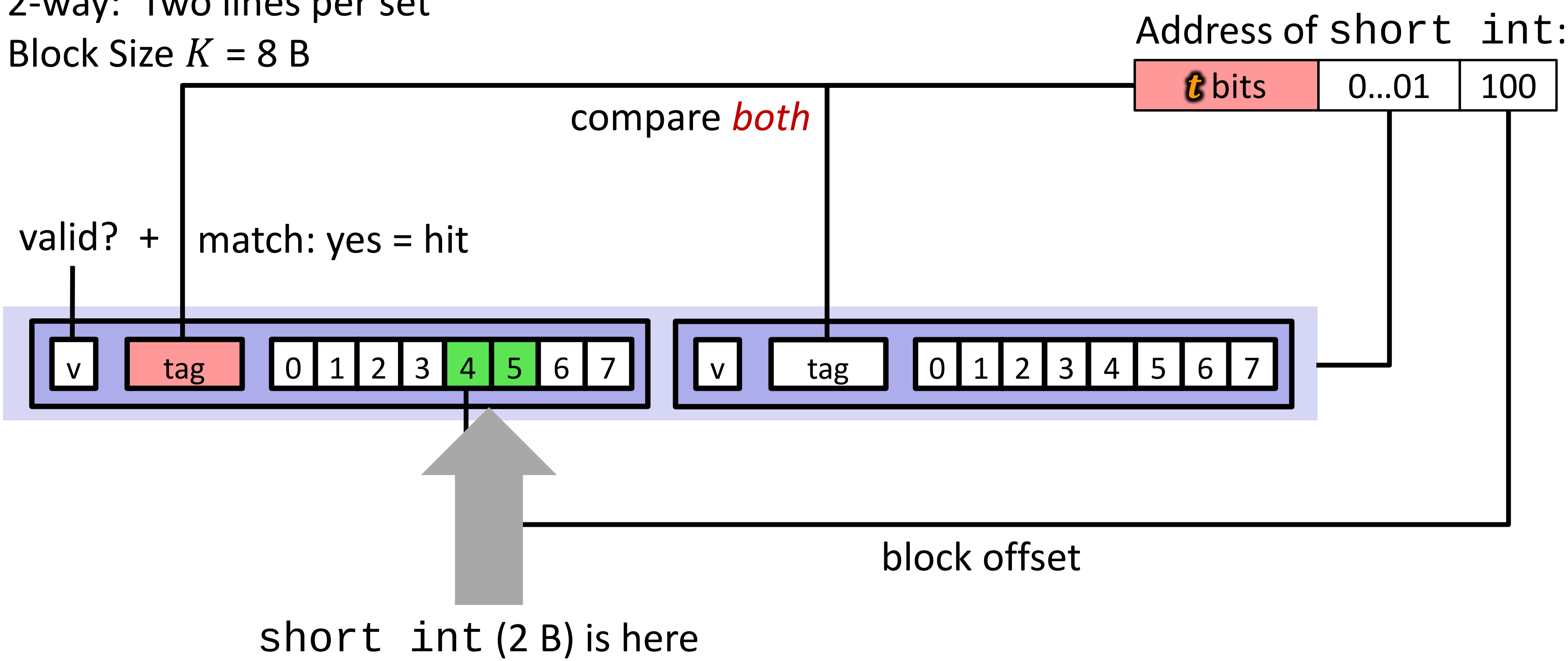
Example: Set-Associative Cache (E = 2)

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



Example: Set-Associative Cache (E = 2)

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), …

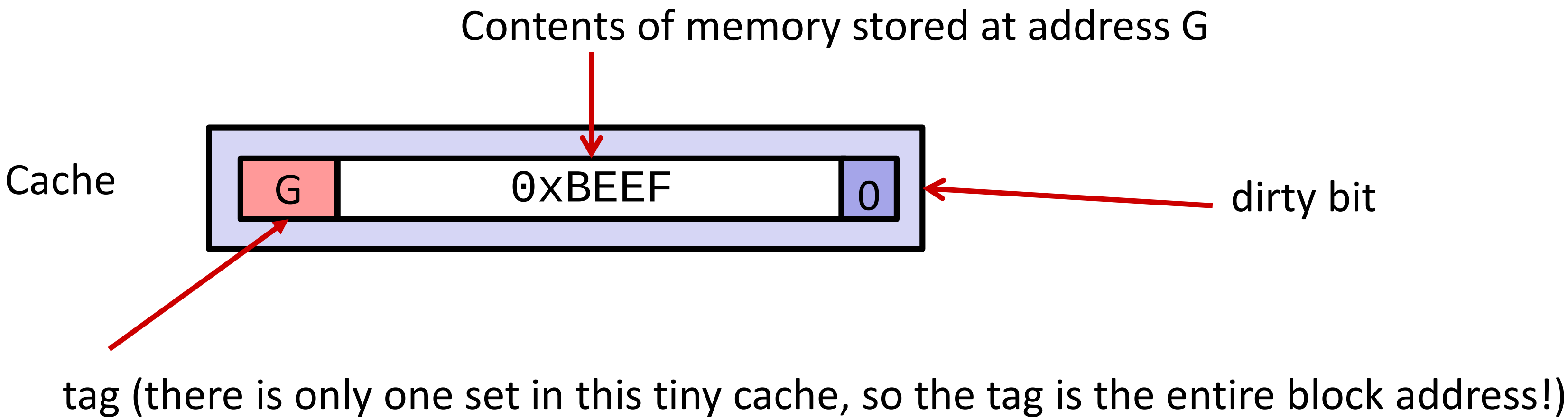
Types of Cache Misses: 3 C's!

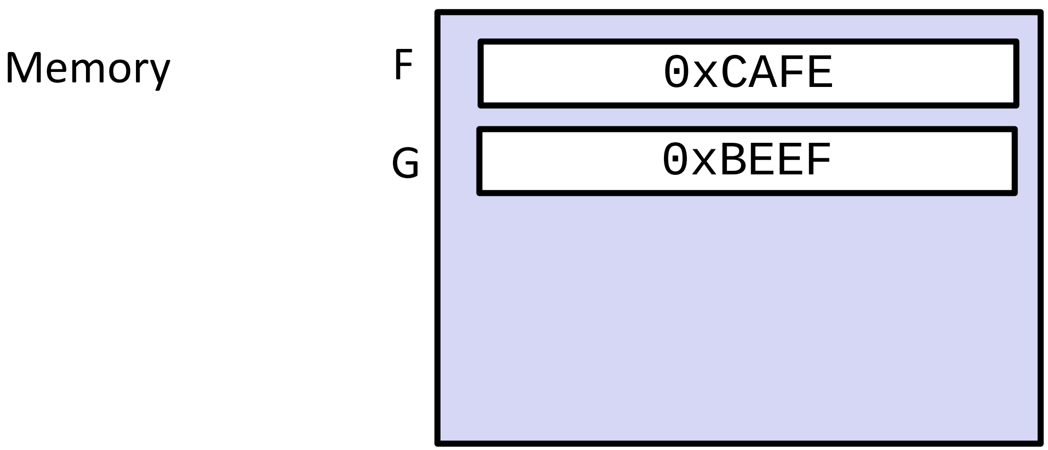
- Compulsory (cold) miss
  - Occurs on first access to a block
- Conflict miss
  - Conflict misses occur when the cache is large enough, but multiple data objects all map to the same slot
    - *e.g.* referencing blocks 0, 8, 0, 8, ... could miss every time
  - Direct-mapped caches have more conflict 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

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



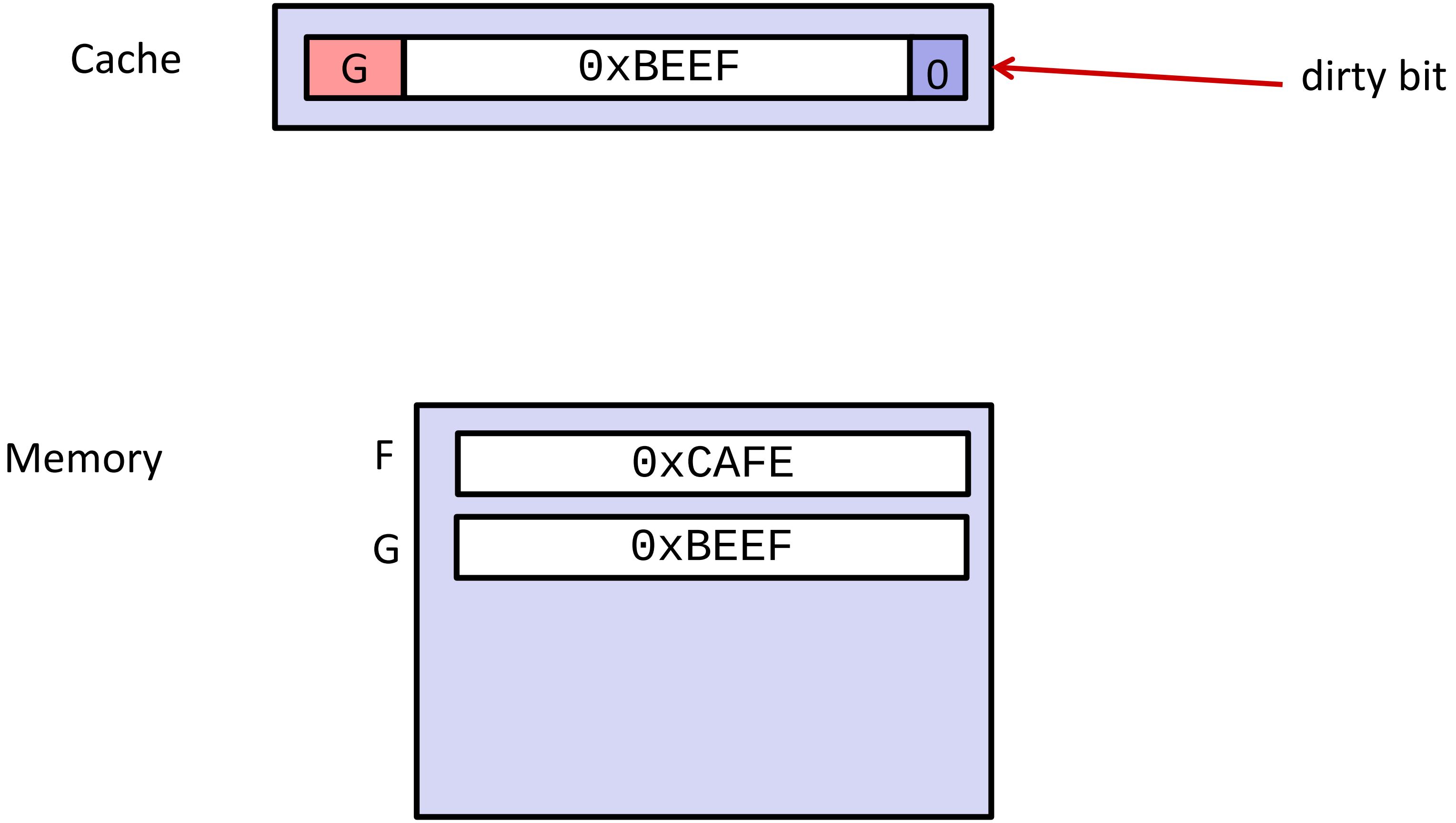


In this example we are sort of ignoring block offsets. Here a block holds 2 bytes (16 bits, 4 hex digits).

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.

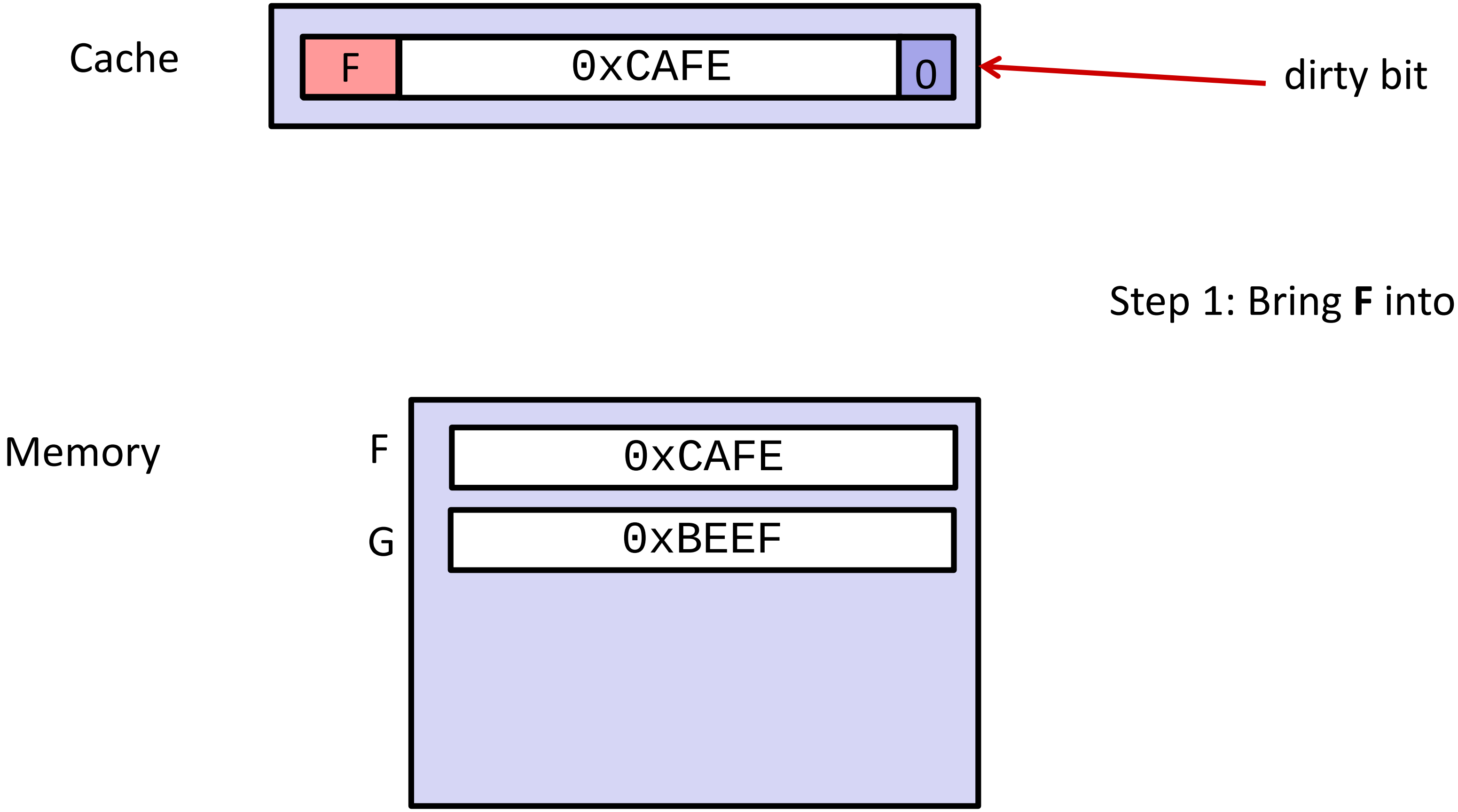
Write-back, write-allocate example

```
mov 0xFACE, F
```



Write-back, write-allocate example

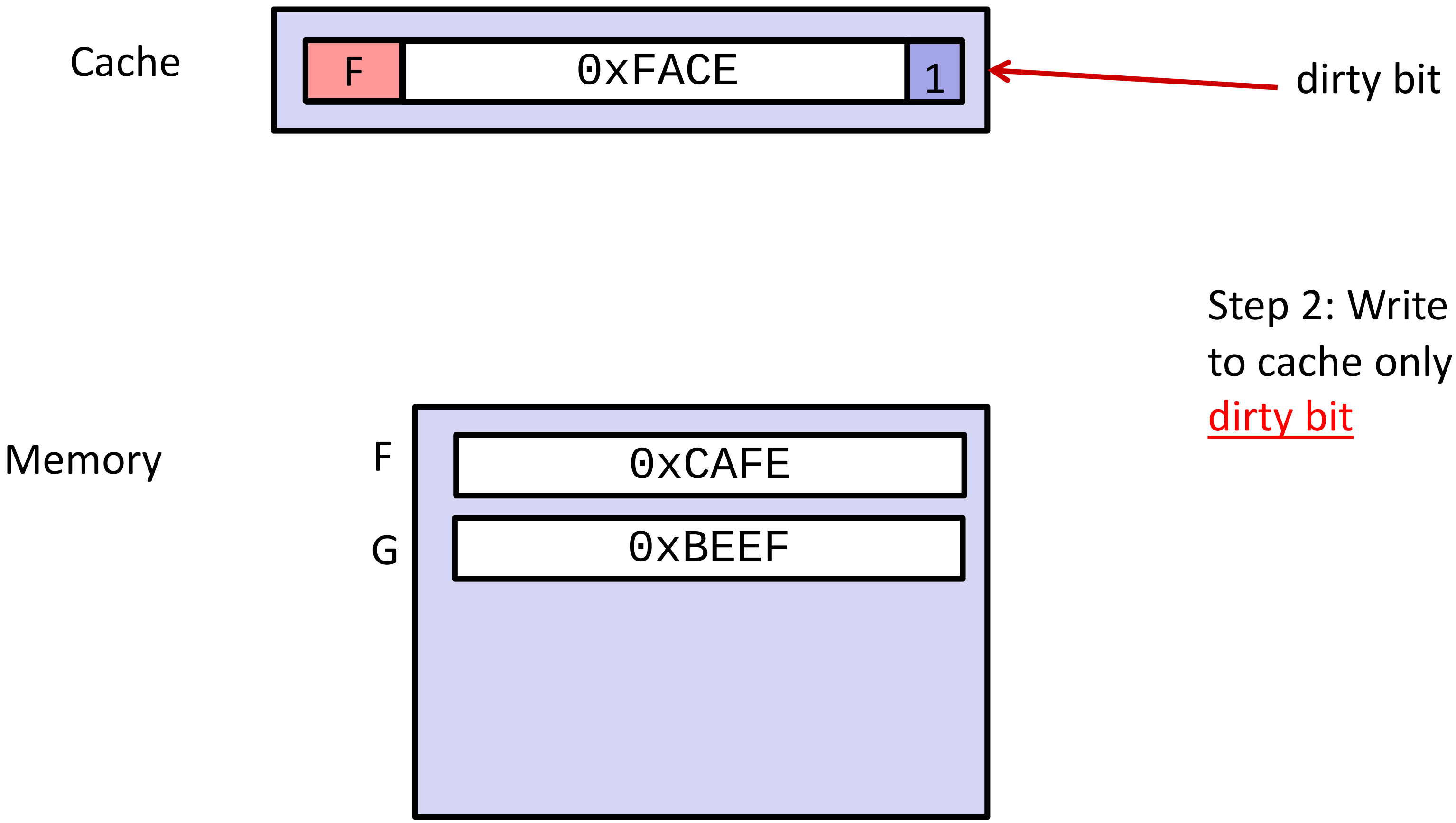
`mov 0xFACE, F`



Step 1: Bring **F** into cache

Write-back, write-allocate example

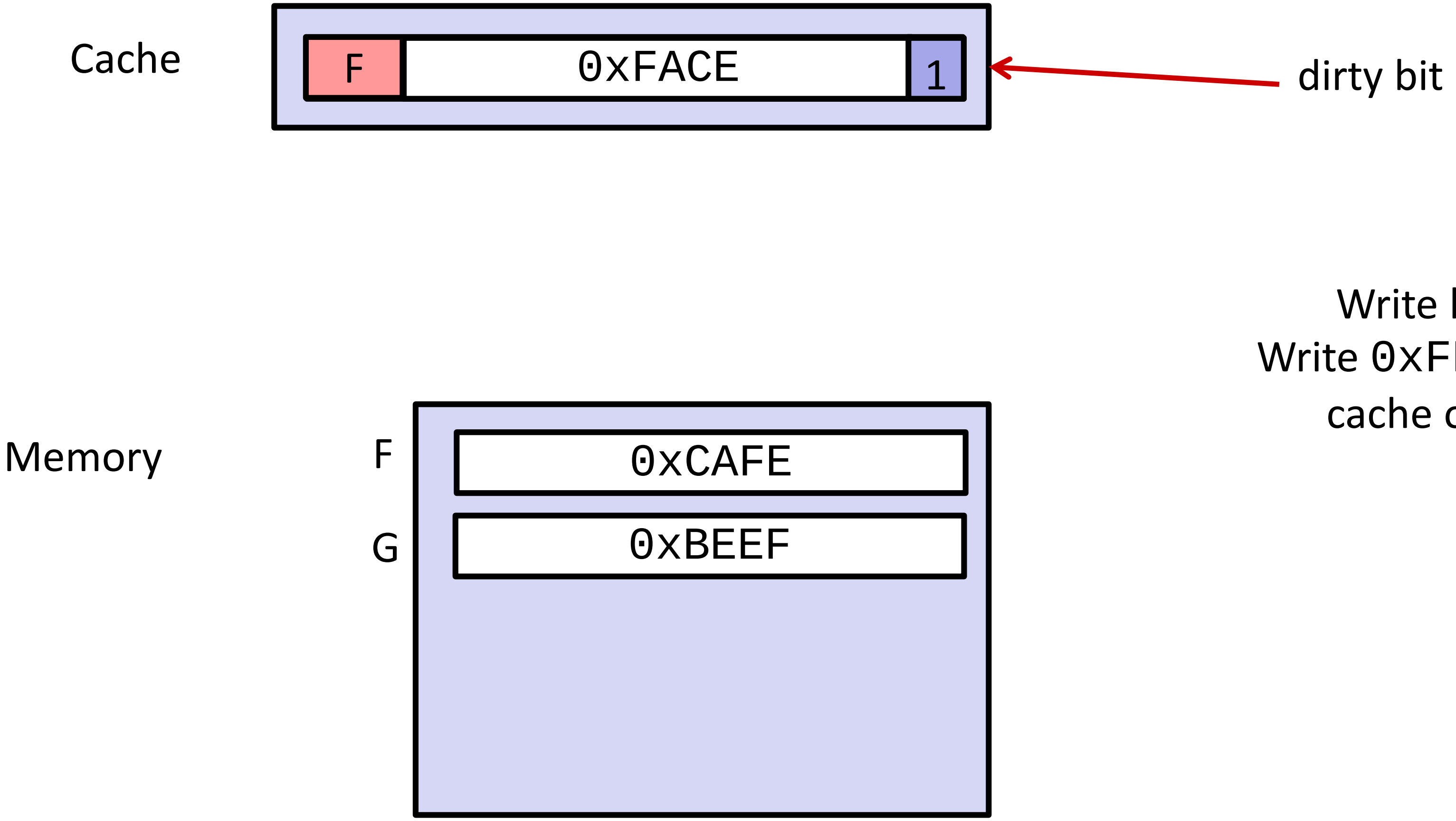
`mov 0xFACE, F`



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

Write-back, write-allocate example

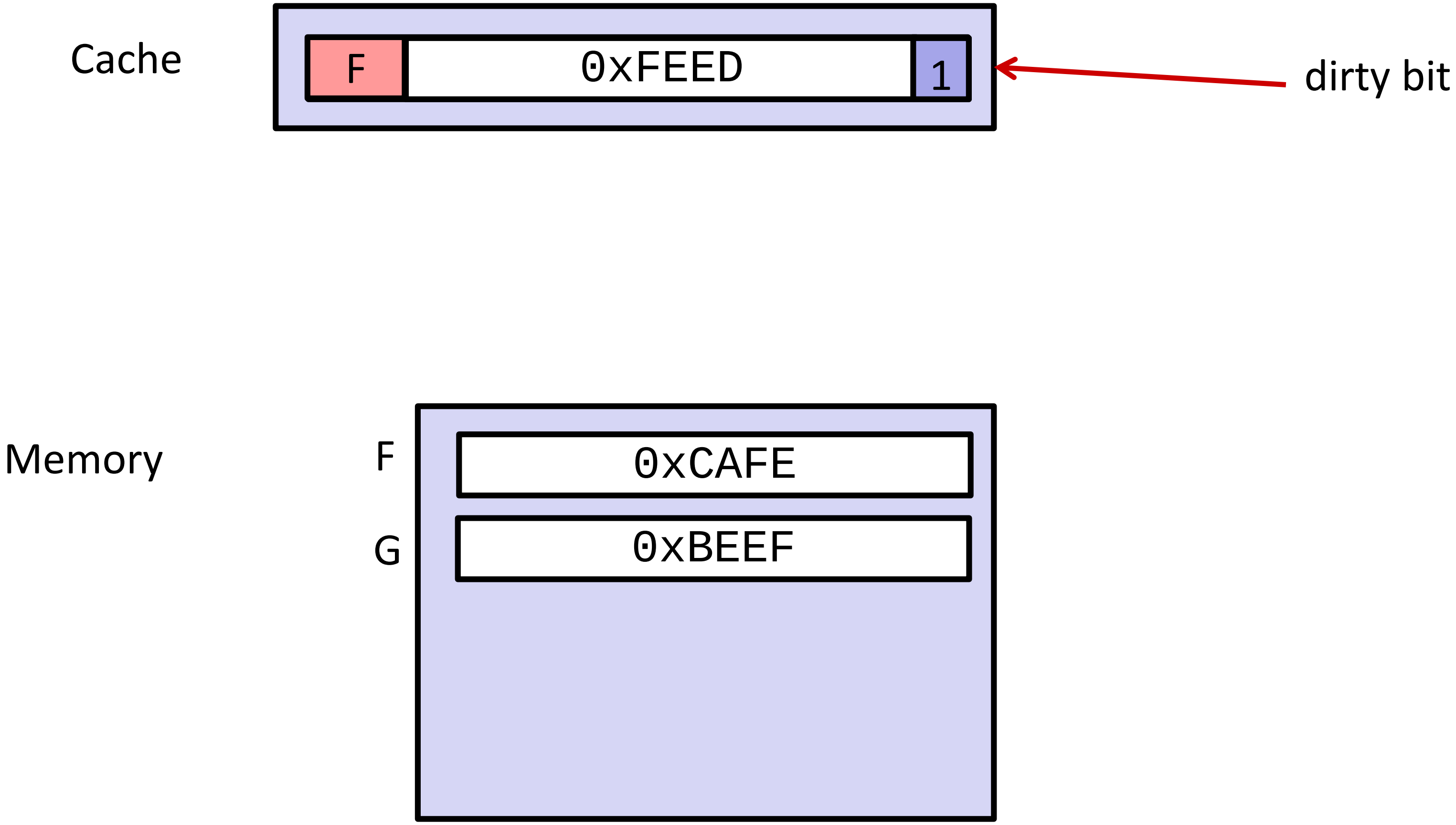
`mov 0xFACE, F`   `mov 0xFEED, F`



Write hit!
Write `0xFEED` to cache only

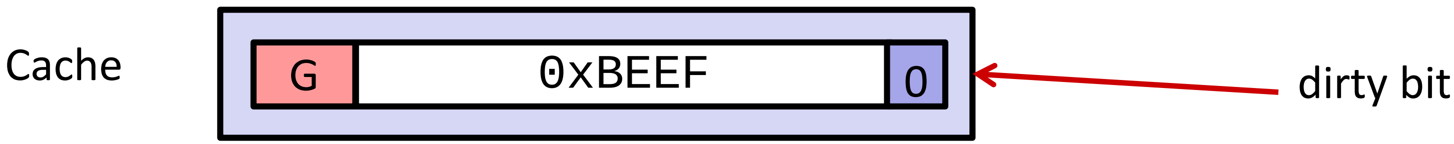
Write-back, write-allocate example

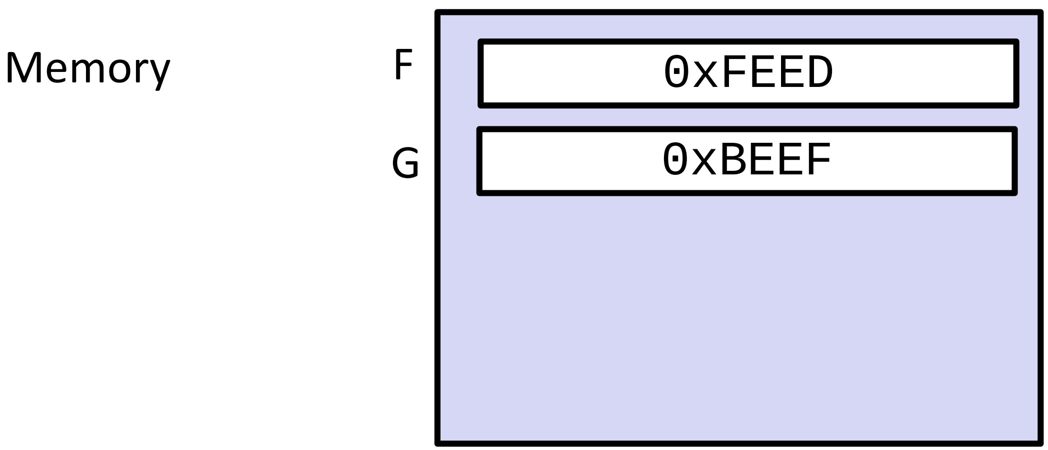
```
mov 0xFACE, F      mov 0xFEED, F      mov G, %rax
```



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…

Example Cache Parameters Problem

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

Cache Size	4 KiB
Block Size	16 B
Associativity	4-way
Hit Time	3 cycles
Miss Rate	20%
Write Policy	Write-through
Replacement Policy	LRU
Tag Bits	10
Index Bits	6
Offset Bits	4
AMAT	AMAT = 3 + 0.2 * 125 = 28

Example Code Analysis Problem

- Assuming the cache starts cold (all blocks invalid), calculate the **miss rate** for the following loop:
  - m = 20 bits, C = 4 KiB, K = 16 B, E = 4

```
#define AR_SIZE 2048
int int_ar[AR_SIZE], sum=0;     // &int_ar=0x80000
for (int i=0; i<AR_SIZE; i++)
   sum += int_ar[i];
for (int j=AR_SIZE-1; j>=0; j--)
   sum += int_ar[i];
```

Cache-Friendly Code

- Programmer can optimize for cache performance
  - How data structures are organized
  - How data are accessed
    - Nested loop structure
    - Blocking is a general technique
- All systems favor "cache-friendly code"
  - Getting absolute optimum performance is very platform specific
    - Cache sizes, line sizes, associativities, etc.
  - Can get most of the advantage with generic code
    - Keep working set reasonably small (temporal locality)
    - Use small strides (spatial locality)
    - Focus on inner loop code

Where else is caching used?

- Caching is one of the biggest ideas in CSE, far beyond the hardware memory hierarchy

- Use a faster set of a subset of the data to exploit temporal and spatial locality
  - At the end of Summer, grab a few jackets from the bedroom closet and leave them near the door

Software Caches are More Flexible

- Examples
  - File system buffer caches, web browser caches, etc.

- Some design differences
  - Almost always fully-associative
    - so, no placement restrictions
    - index structures like hash tables are common (for placement)
  - Often use complex replacement policies
    - misses are very expensive when disk or network involved
    - worth thousands of cycles to avoid them
  - Not necessarily constrained to single “block” transfers
    - may fetch or write-back in larger units, opportunistically