Caches III

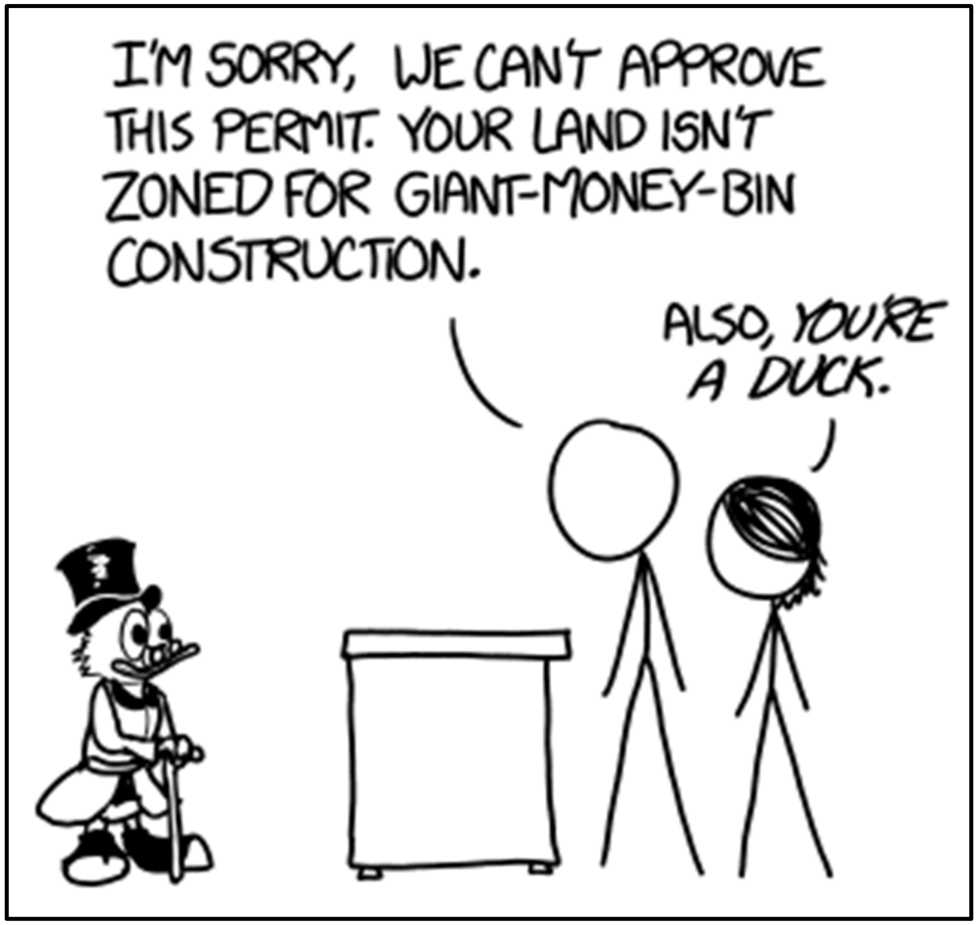
CSE 351 Spring 2019

Instructor:

Ruth Anderson

Teaching Assistants:

Gavin Cai
Jack Eggleston
John Feltrup
Britt Henderson
Richard Jiang
Jack Skalitzky
Sophie Tian
Connie Wang
Sam Wolfson
Casey Xing
Chin Yeoh


I'M SORRY, WE CAN'T APPROVE THIS PERMIT. YOUR LAND ISN'T ZONED FOR GIANT-MONEY-BIN CONSTRUCTION.
ALSO, YOU'RE A DUCK.

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

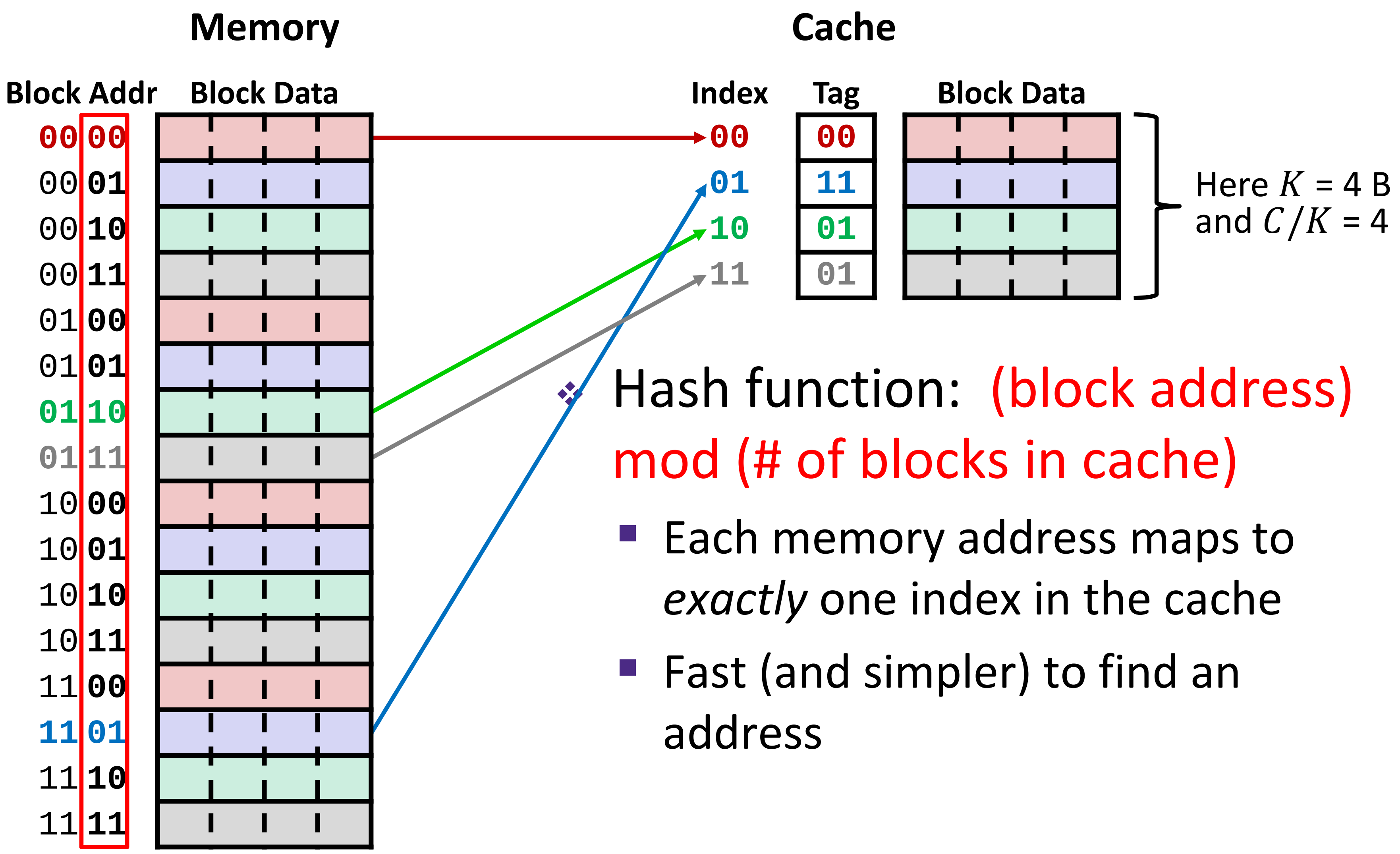
Administrivia

- Lab 3, due Wednesday (5/15)
- Homework 4 , due Wed (5/22) (Structs, Caches)

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

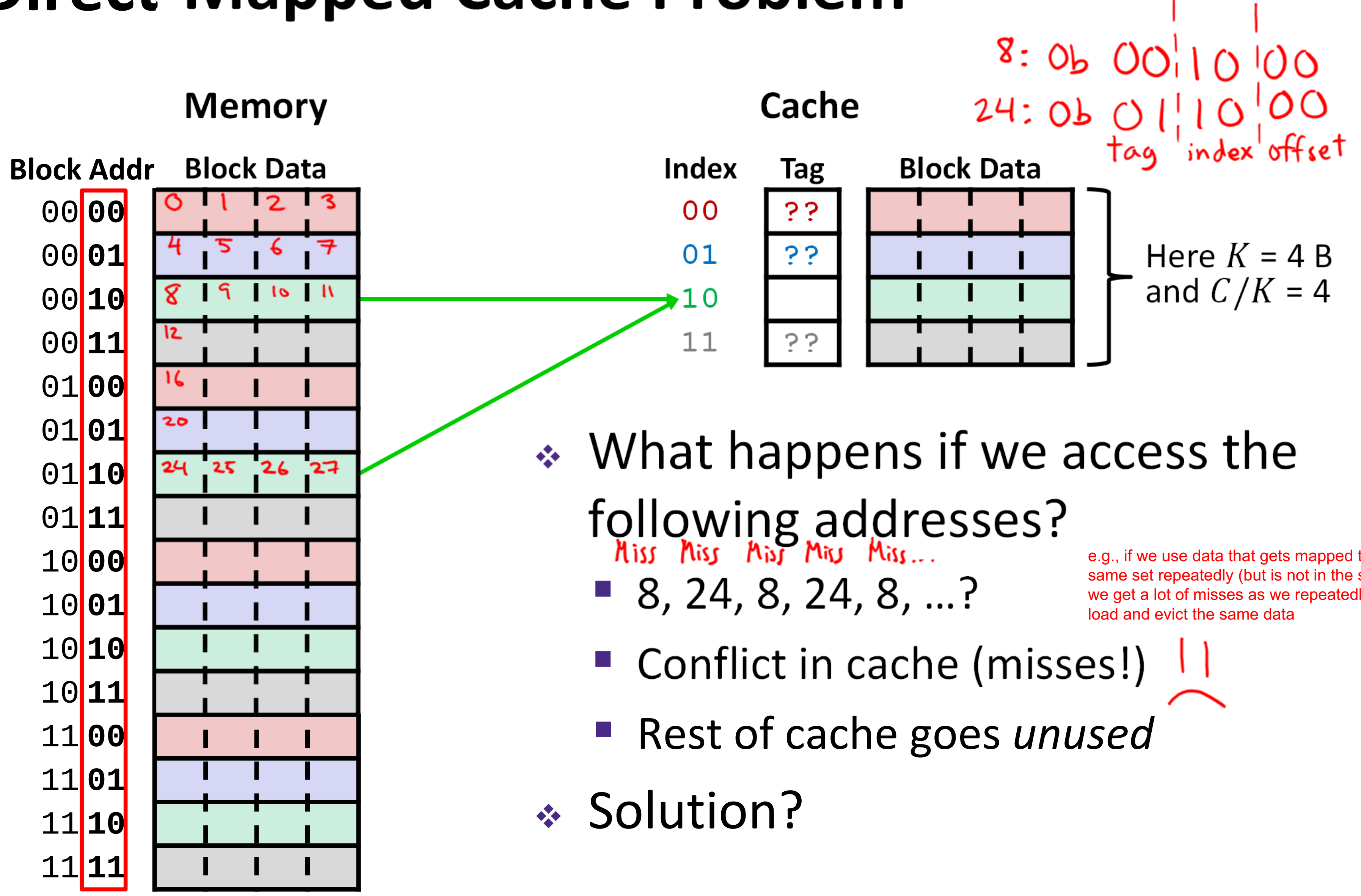
Direct-Mapped Cache



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

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

Direct-Mapped Cache Problem



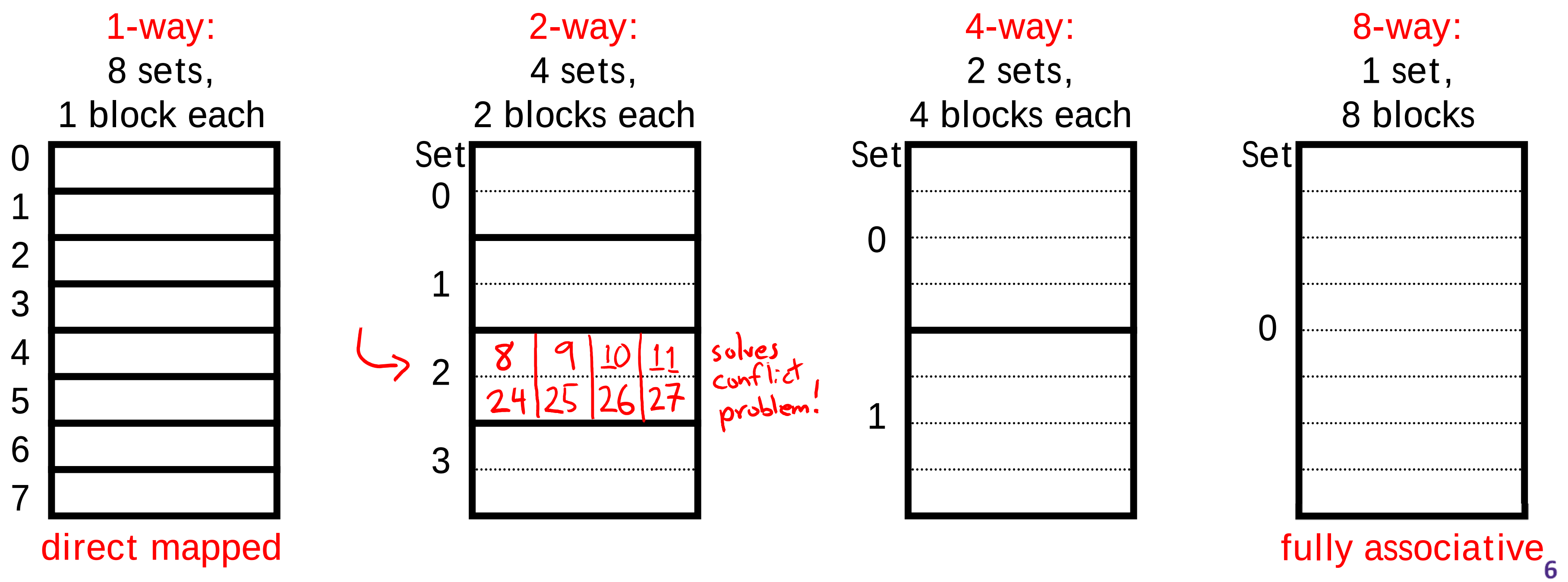
- What happens if we access the following addresses?
  - 8, 24, 8, 24, 8, …?

    Miss Miss Miss Miss Miss…

    e.g., if we use data that gets mapped to the same set repeatedly (but is not in the same block we get a lot of misses as we repeatedly load and evict the same data
  - 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**

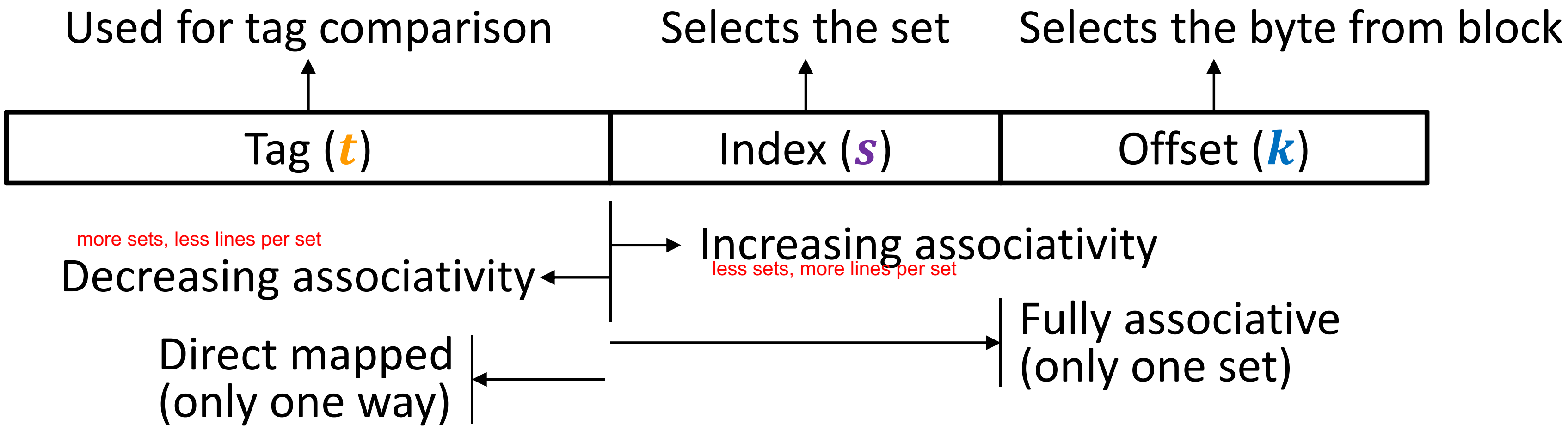


Cache Organization (3)

Note: The textbook uses "b" for offset bits

number of different blocks each set can store

- 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


Used for tag comparison
Selects the set
Selects the byte from block
Tag (t)
Index (s)
Offset (k)
more sets, less lines per set
Decreasing associativity
Increasing associativity
less sets, more lines per set
Direct mapped (only one way)
Fully associative (only one set)

Example Placement

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

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

$\underbrace{\texttt{0011}}_{\text{offset}}$ $I_{E=1}$ $I_{E=4}$ $I_{E=2}$

log_2(block size) = log_2(16) = 4

$\boldsymbol{t} = \boldsymbol{m}-\boldsymbol{s}-\boldsymbol{k}$ $\quad \boldsymbol{s} = \log_2(C/K/E)$ $\quad \boldsymbol{k} = \log_2(K) = 4$ bits

***m*-bit address:**

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

$\boldsymbol{s}$ = ? $\log_2(8/1) = 3$ bits

Direct-mapped (E=1)

	Set	Tag	Data
000	0		
001	1		
010	2		
011	3	—	
100	4		
101	5		
110	6		
111	7		

$\boldsymbol{s}$ = ? $\log_2(8/2) = 2$ bits

2-way set associative (E=2)

	Set	Tag	Data
00	0		
01	1		
10	2		
11	3	—	
		—	

$\boldsymbol{s}$ = ? $\log_2(8/4) = 1$ bit

4-way set associative (E=4)

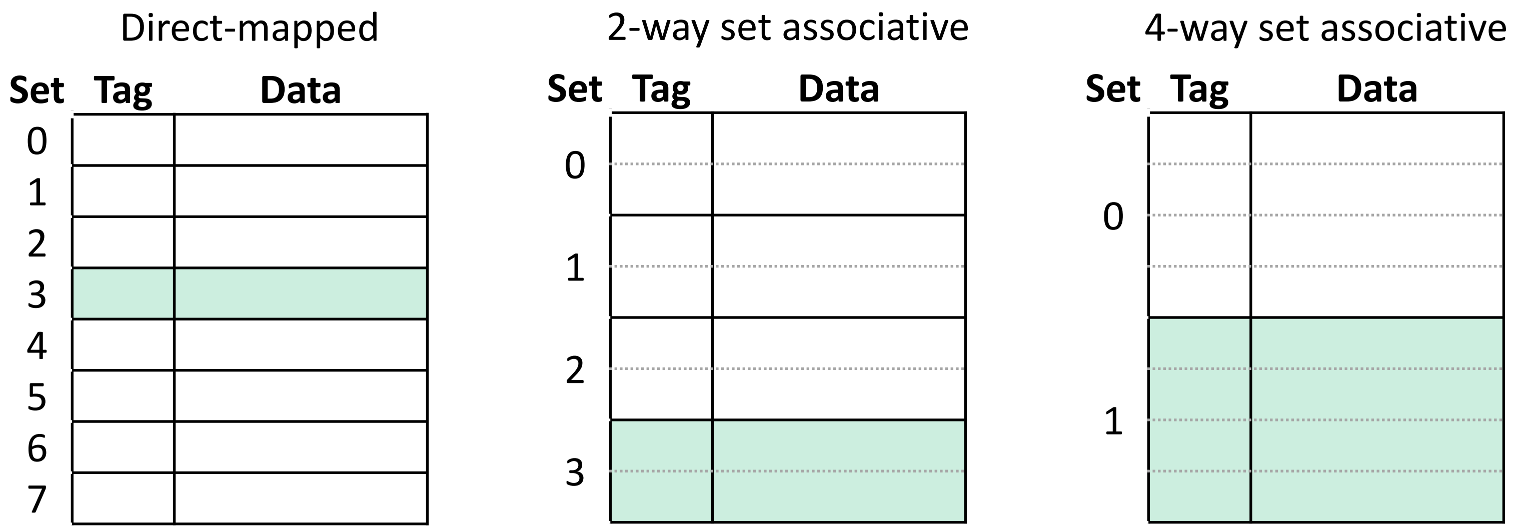
Set	Tag	Data
0		
1	—	
	—	
	—	
	—	

Block Replacement

Suppose there is a block miss and we have to load data into the cache. Once we determine which set it goes in, how do we choose which line in the set to overwrite? If there are unused lines, its an easy choice. If not, we have to overwrite something. What's a good criteria for choosing what to overwrite? We want to avoid the trashing issue discussed on page 604 of textbook, so maybe a policy of removing data that hasn't been used in a while?

- *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*")

Replace the line that has been used the fewest number of times in some given time frame,.



Peer Instruction Question

$C = 2^{11}$ B $K = 2^{7}$ 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 at http://pollev.com/rea

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

(number sets) * (number lines per set) = number of blocks

A. 2

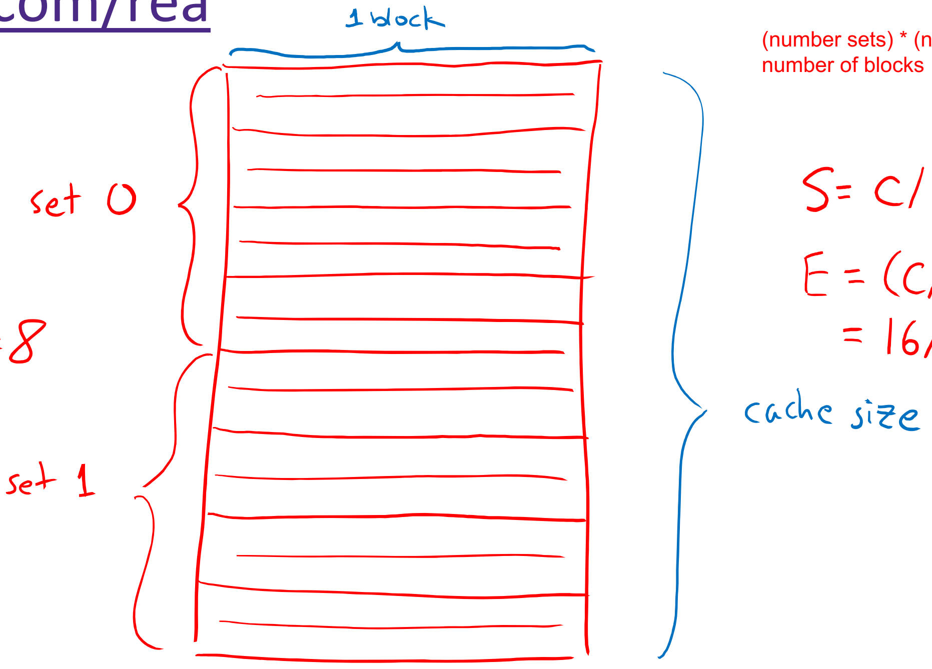
B. 4

[✓ C. 8]

D. 16

E. We're lost…

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



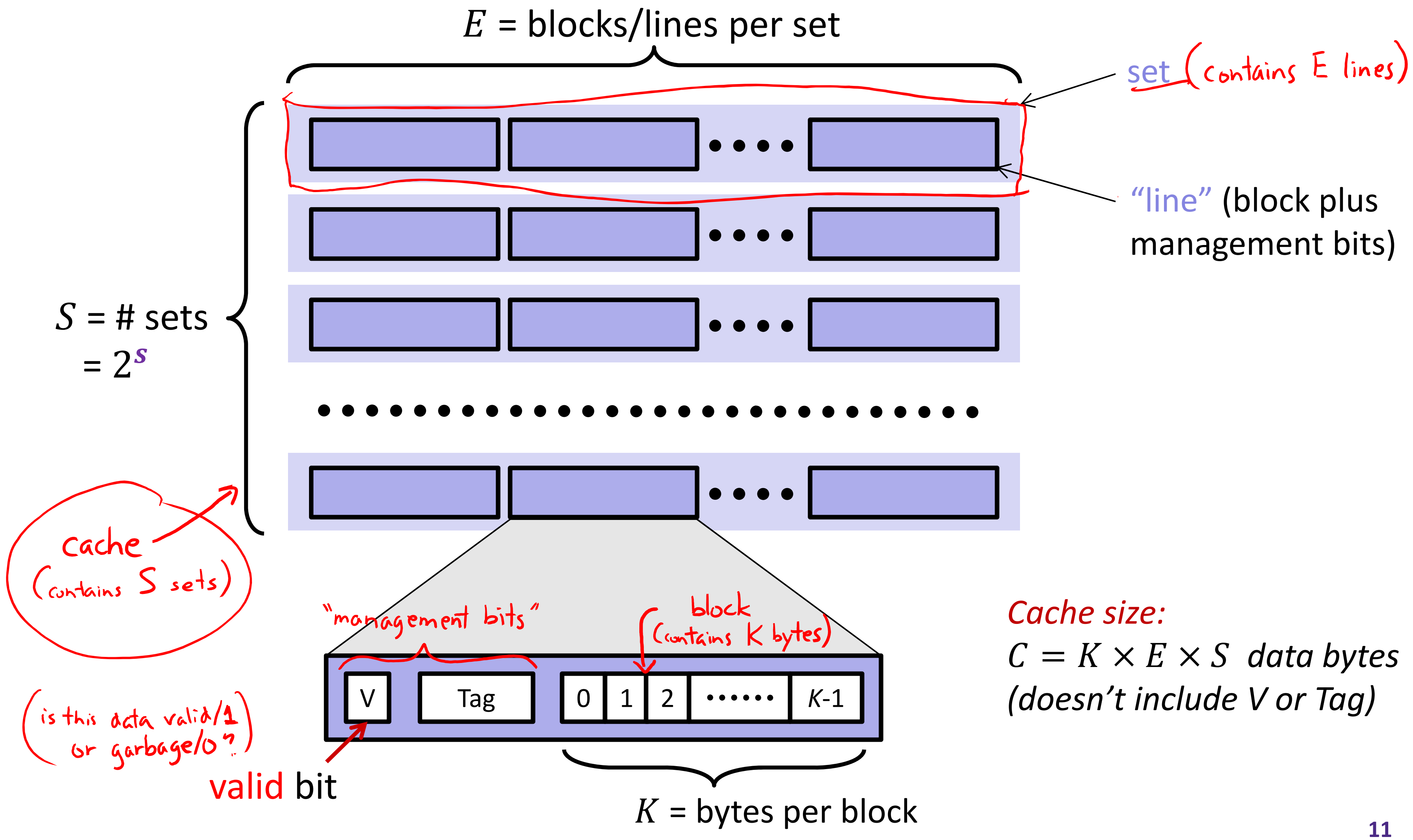
$S = C/K/E$

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

$m=16$

- 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 $(\underset{\text{sets}}{S}, \overset{\text{associativity}}{E}, \underset{\text{block size}}{K})$



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

blocks/lines per set

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

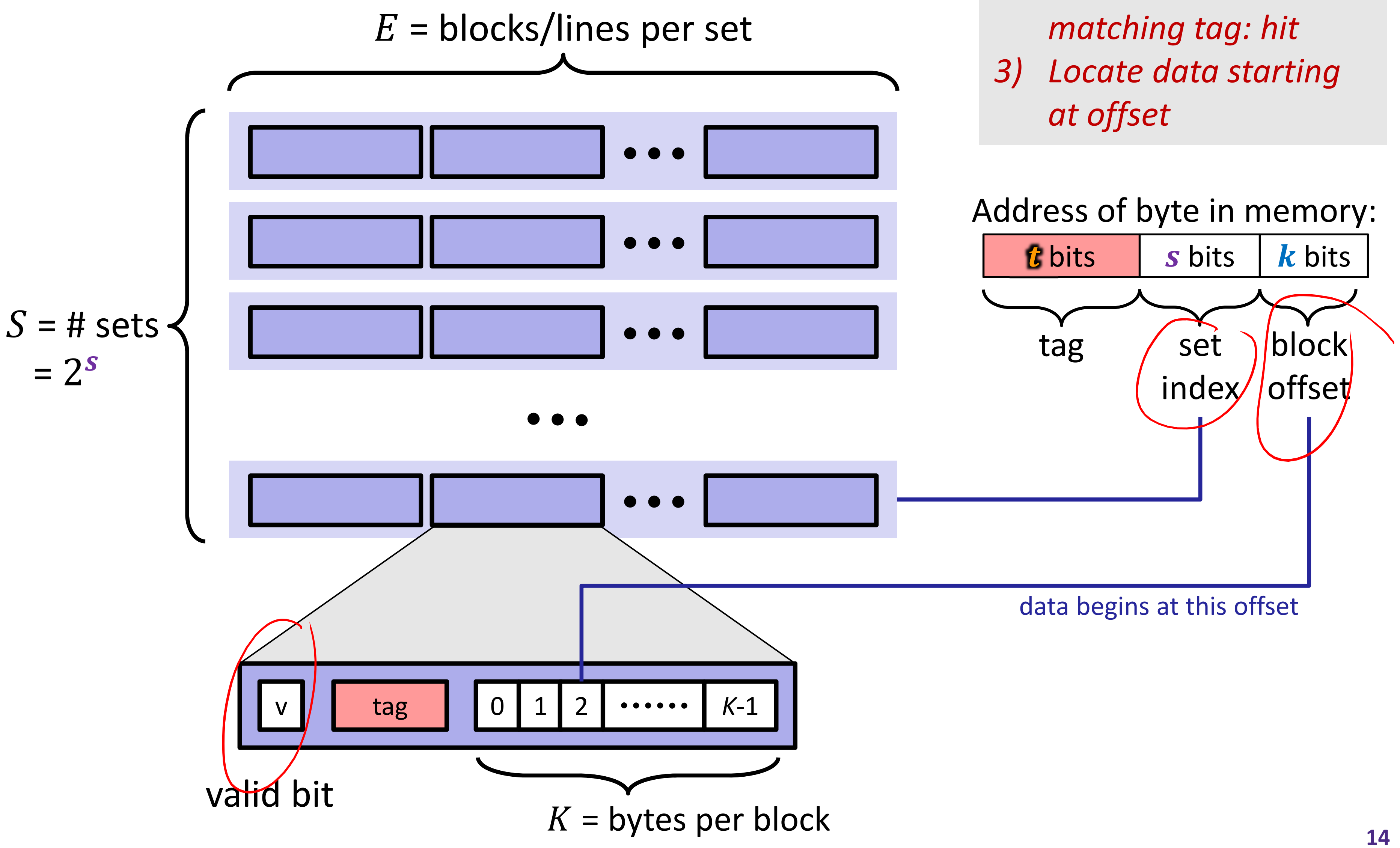
$\to 2^{12}$ B $\Longleftrightarrow$ $m = 12$ bits MP

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

C	**Cache Size**	256 B	2^8
K	**Block Size**	32 B	2^5
E	**Associativity**	2-way	2^1
HT	**Hit Time**	3 cycles	
MR	**Miss Rate**	20%	
$t = m - s - k$	**Tag Bits**	5	
$s = \log_2(C/K/E)$	**Index Bits**	2	$2^8/2^5/2^1$
$k = \log_2(K)$	**Offset Bits**	5	
AMAT = HT + MR * MP	**AMAT**	$3 + 0.2(125) = 28$ clock cycles	

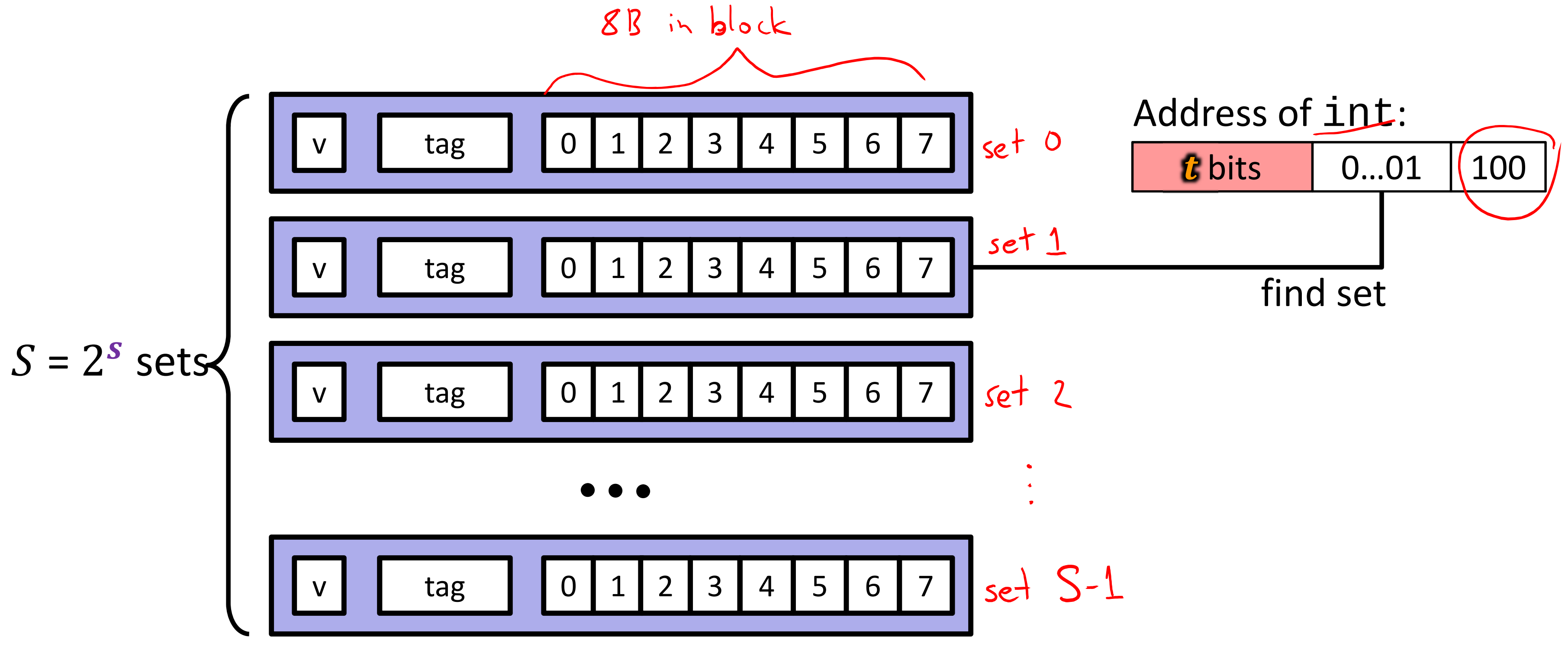
Cache Read

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



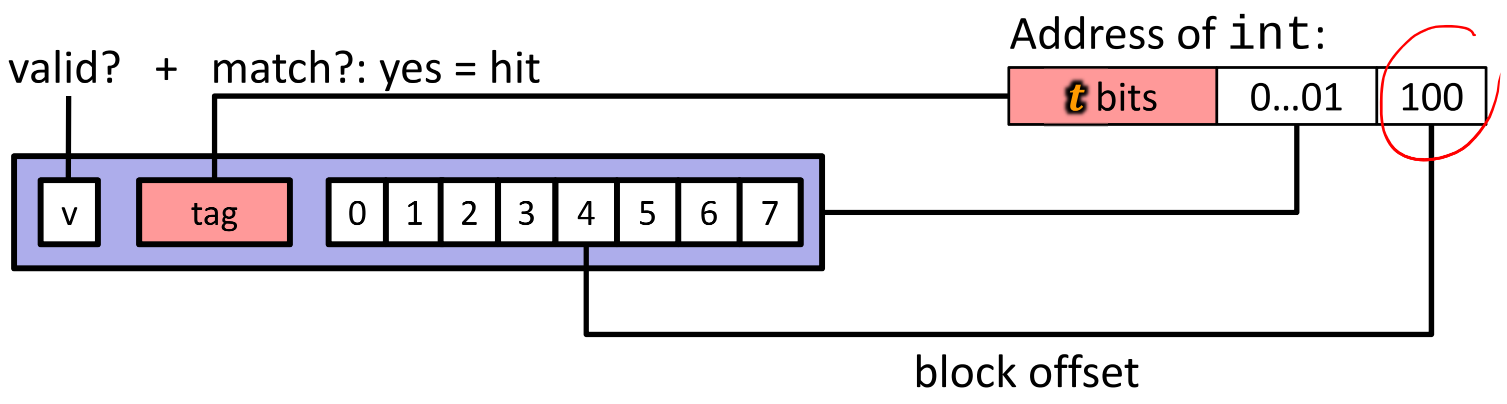
Example: Direct-Mapped Cache (E = 1)

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



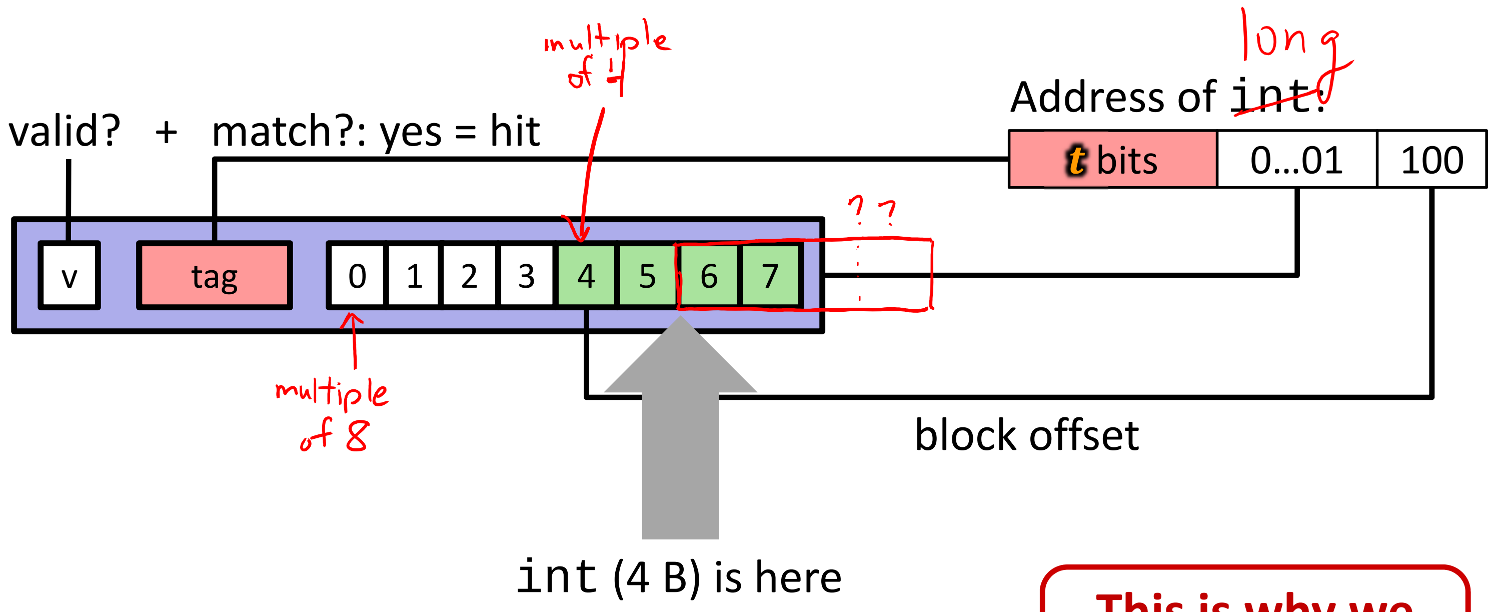
Example: Direct-Mapped Cache (E = 1)

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!

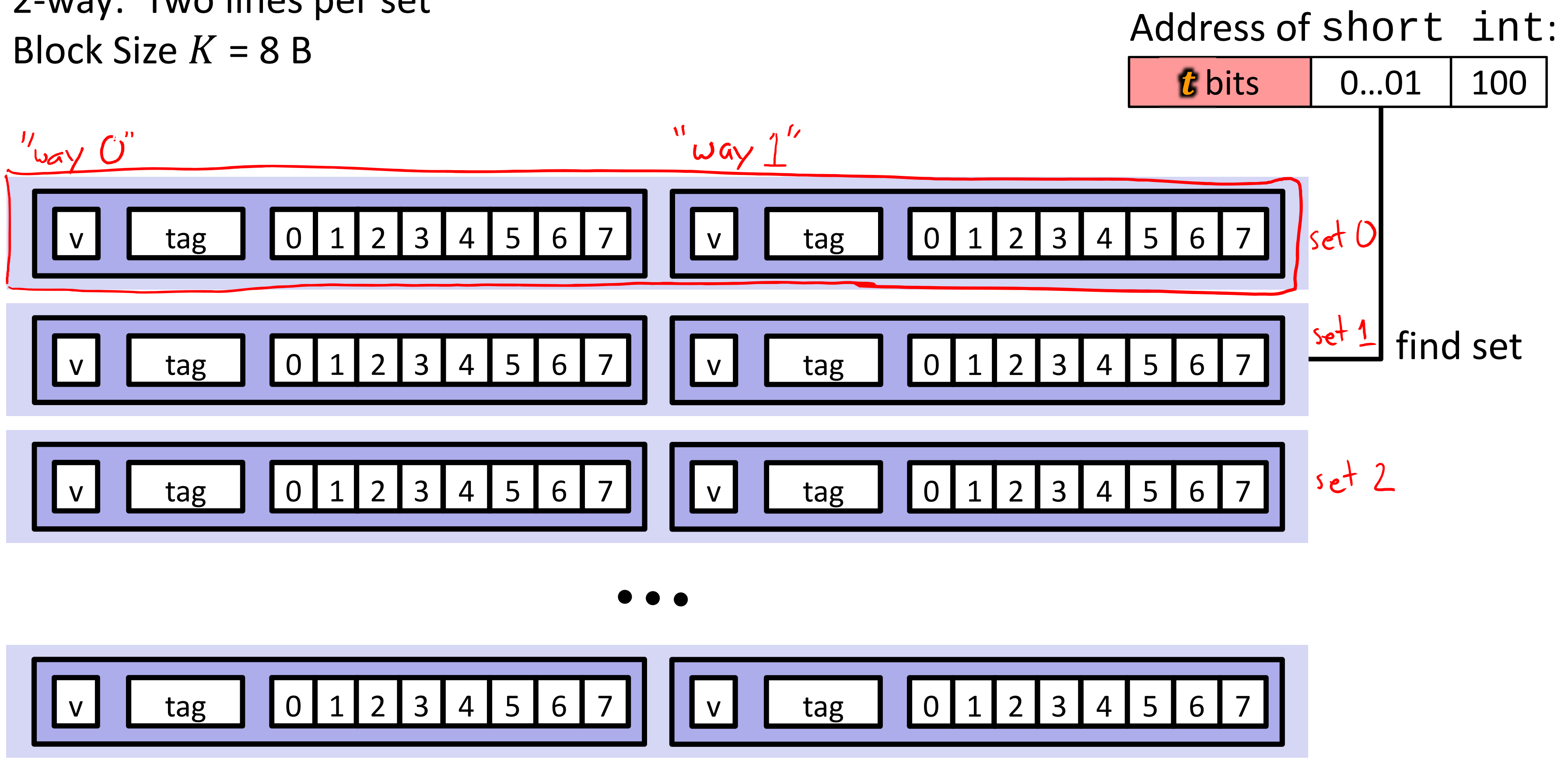
no unnecessary extra cache accesses across block boundaries

e.g. data is always stored in one block?

No match? Then old line gets evicted and replaced

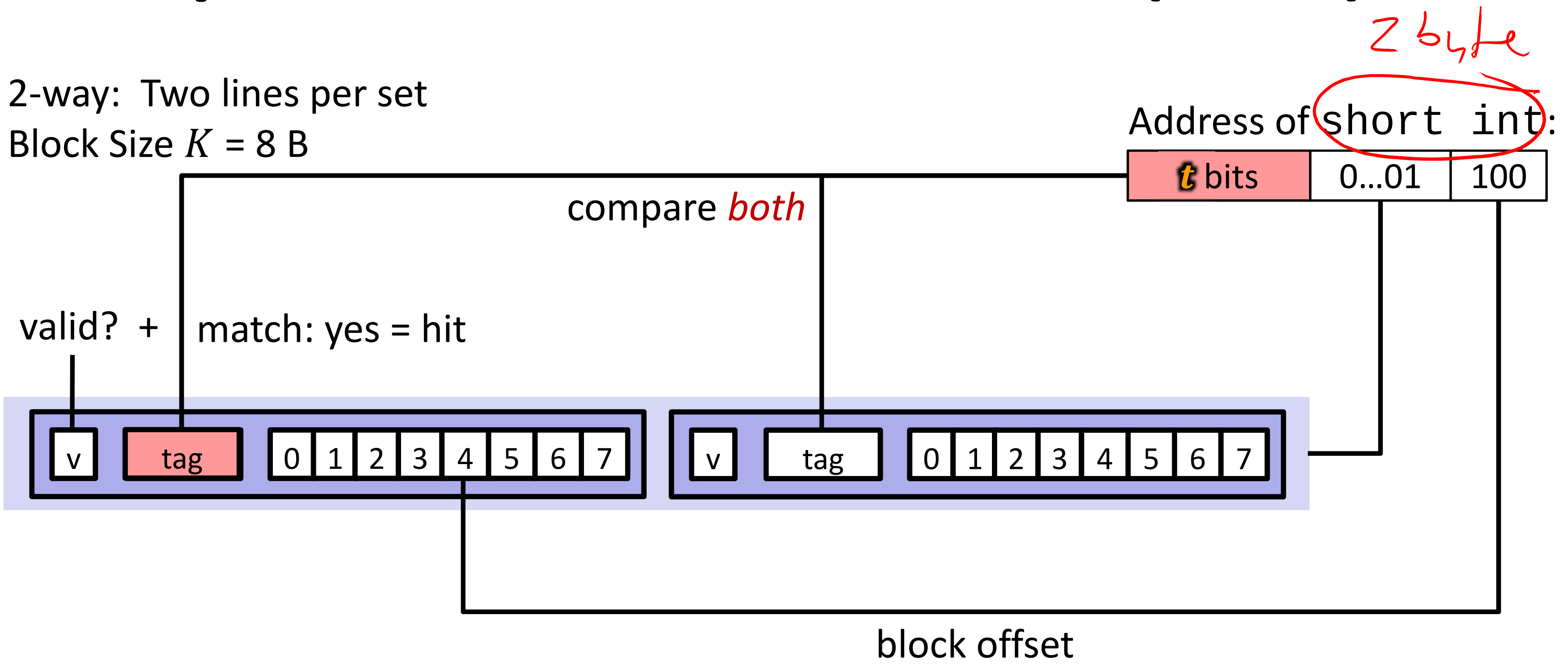
Example: Set-Associative Cache (E = 2)

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



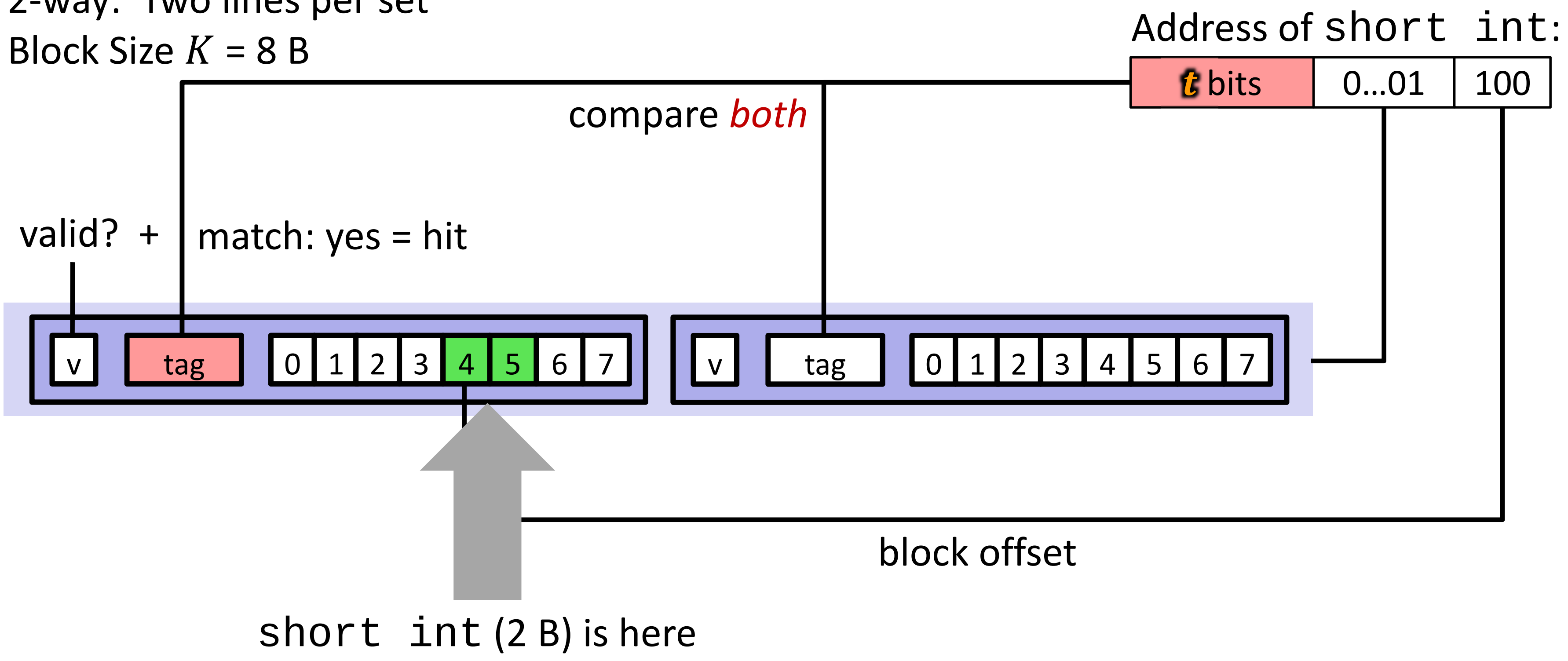
Example: Set-Associative Cache (E = 2)

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

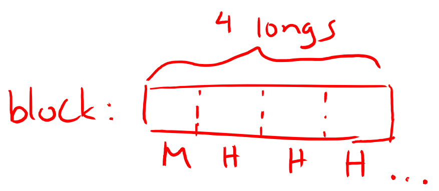
Example Code Analysis Problem

- Assuming the cache starts cold (all blocks invalid) and `sum` is stored in a register, calculate the **miss rate**: $\boxed{1/4}$
  - m = 12 bits, C = 256 B, K = 32 B, E = 2

    $t = 5$ bits, $s = 2$ bits, $k = 5$ bits

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];
```



	address	tag	index	offset	
ar[0][0]	$\rightarrow$	0b 1000 0	0 0	0 0000	$\rightarrow$ M (compulsory)
ar[0][1]	$\rightarrow$	0b 1000 0	0 0	0 1000	$\rightarrow$ H
ar[0][2]	$\rightarrow$	0b 1000 0	0 0	1 0000	$\rightarrow$ H
ar[0][3]	$\rightarrow$	0b 1000 0	0 0	1 1000	$\rightarrow$ H
ar[0][4]	$\rightarrow$	0b 1000 0	0 1	0 0000	$\rightarrow$ M (compulsory)
⋮			⋮		⋮

Challenge: what is the miss rate if we switch the ordering of the for-loops?

What about writes?

- Multiple copies of data exist:
  - L1, L2, possibly L3, main memory
- What to do on a write-hit? (block/data already in $)
  - 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"*) ← extra management bit only for write-back cache
- What to do on a write-miss? (block/data not currently in $)
  - 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~~ next level
- Typical caches:
  - Write-back + Write-allocate, usually ☆
  - Write-through + No-write-allocate, occasionally