The Hardware/Software Interface

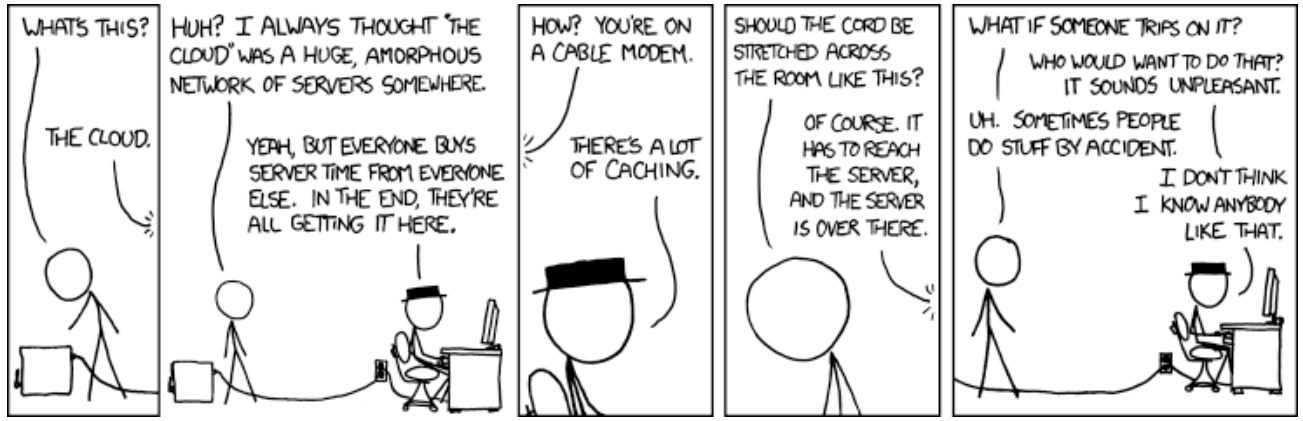
Memory & Caches III

Guest Instructor:
Naama Amiel

Instructors:
Justin Hsia, Amber Hu

Teaching Assistants:

Anthony Mangus	Divya Ramu
Grace Zhou	Jessie Sun
Jiuyang Lyu	Kanishka Singh
Kurt Gu	Liander Rainbolt
Mendel Carroll	Ming Yan
Naama Amiel	Pollux Chen
Rose Maresh	Soham Bhosale
Violet Monserate	



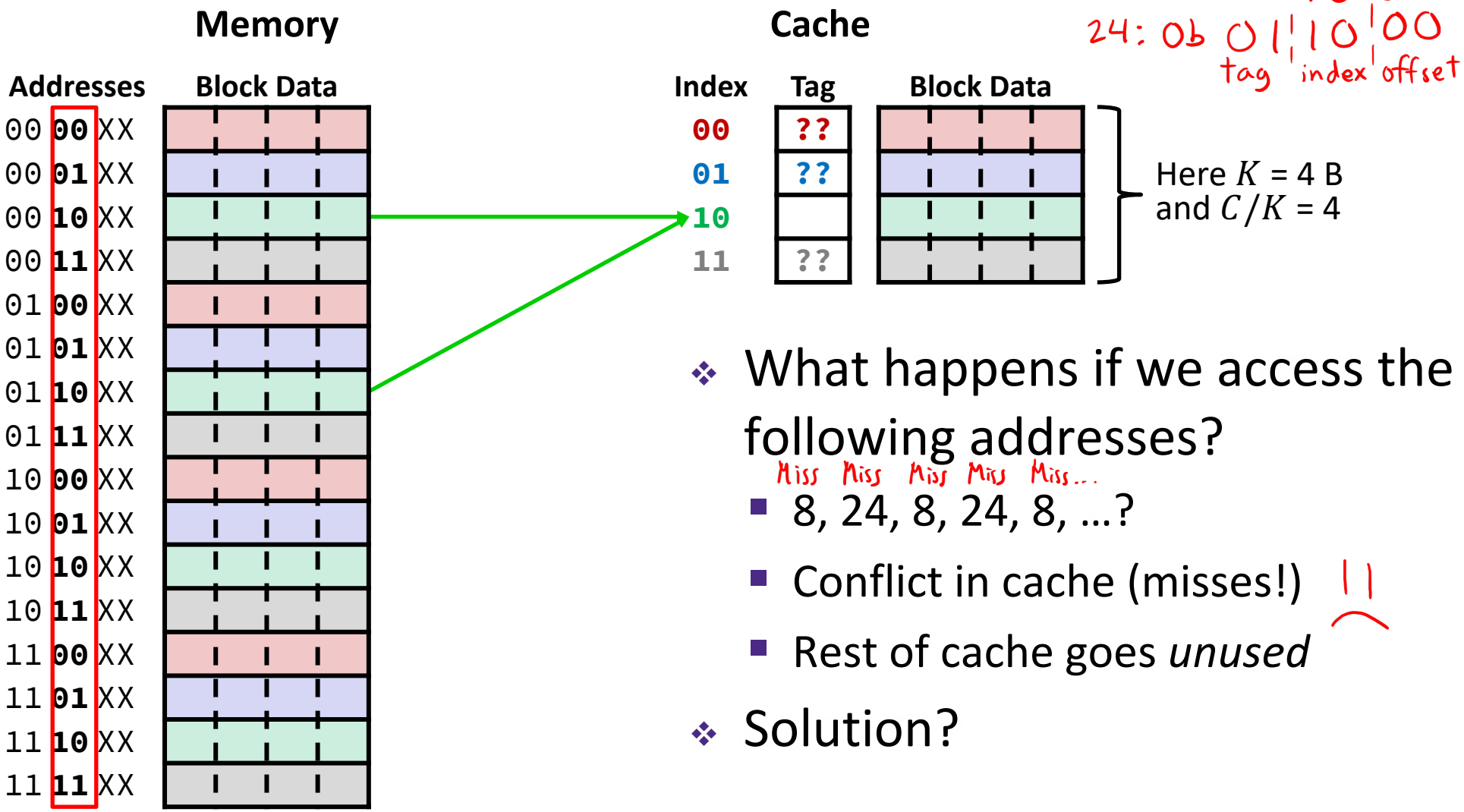
http://xkcd.com/908/

Lecture Outline (1/3)

- **Associativity and Replacement**
- Cache Organization
- Example Cache Problems

Review: Direct-Mapped Cache Problem

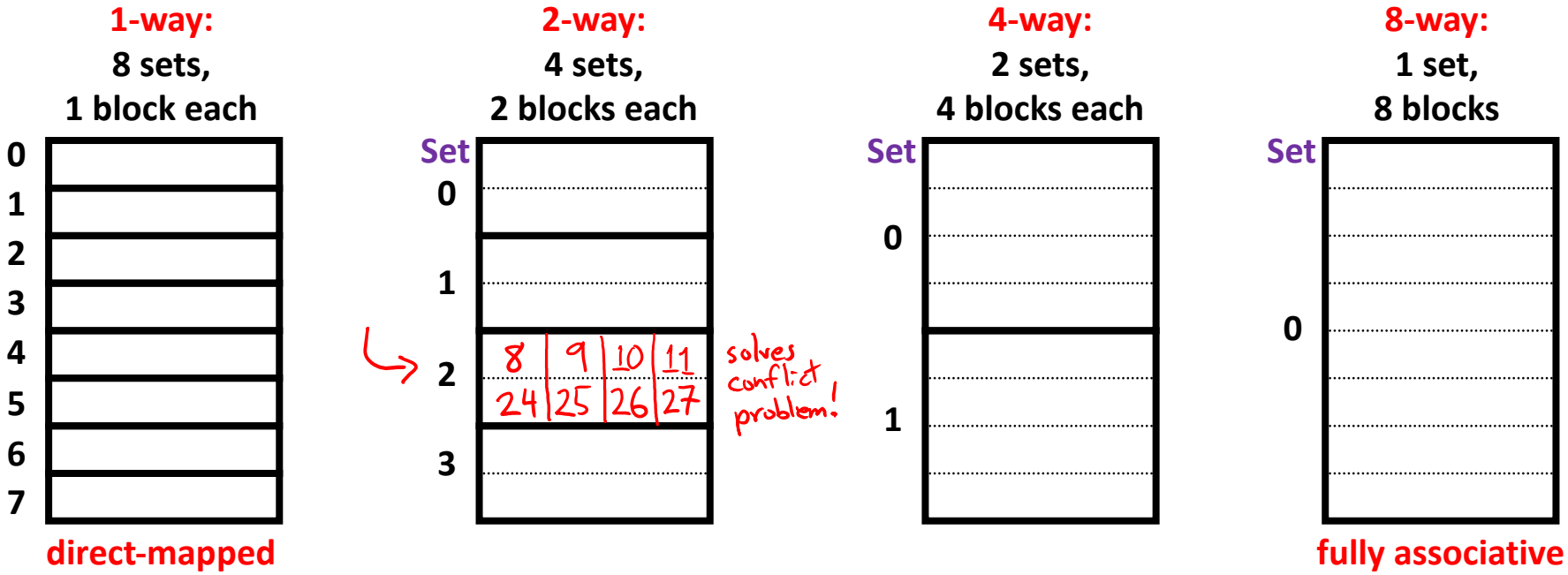
8: 0b 00 | 10 | 00
24: 0b 01 | 10 | 00
tag index offset



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

Associativity Motivation

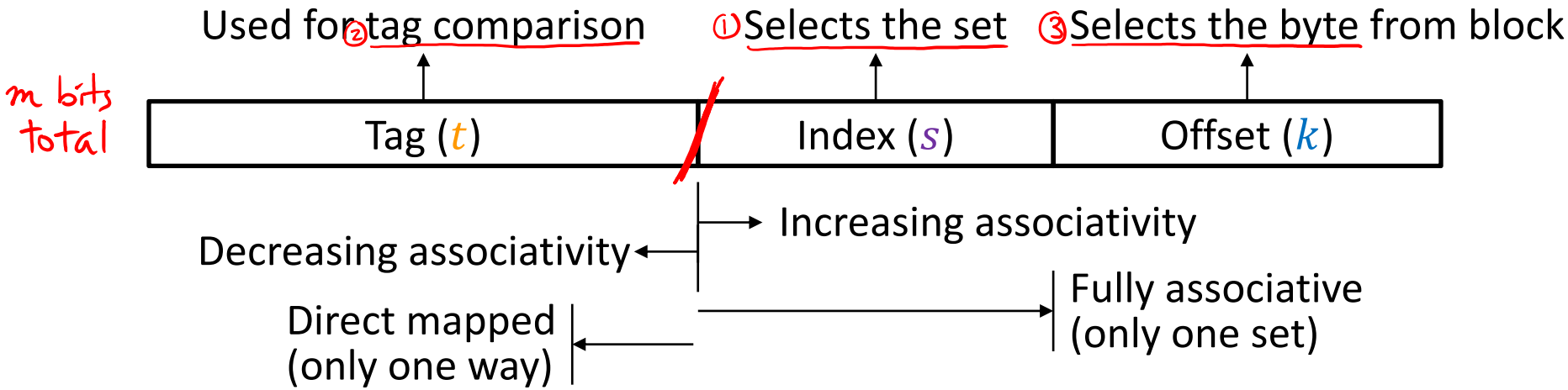
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**, each set can store block in some # of **ways**



Associativity (Review)

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 index into cache *sets*, of which there are $S = (C/K)/E$
  - Use lowest $\log_2(S)$ = 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



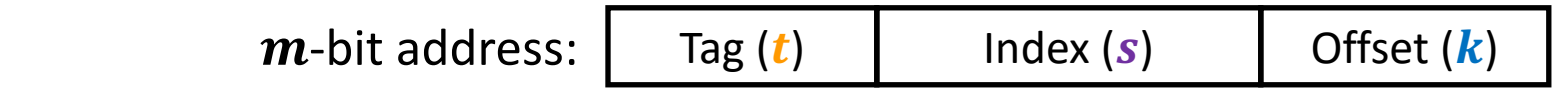
Example Placement

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

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

$s_{E=1}$ $s_{E=4}$ $s_{E=2}$ offset

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



s = ?3 bits
Direct-mapped

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

s = ?2 bits
2-way set associative

Set	Tag	Data
(00) 0		
(01) 1		
(10) 2		
(11) 3		✓
		✓

s = ? 1 bit
4-way set associative

Set	Tag	Data
(0) 0		
(1) 1		✓
		✓
		✓
		✓

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?

  A. 2

  B. 4

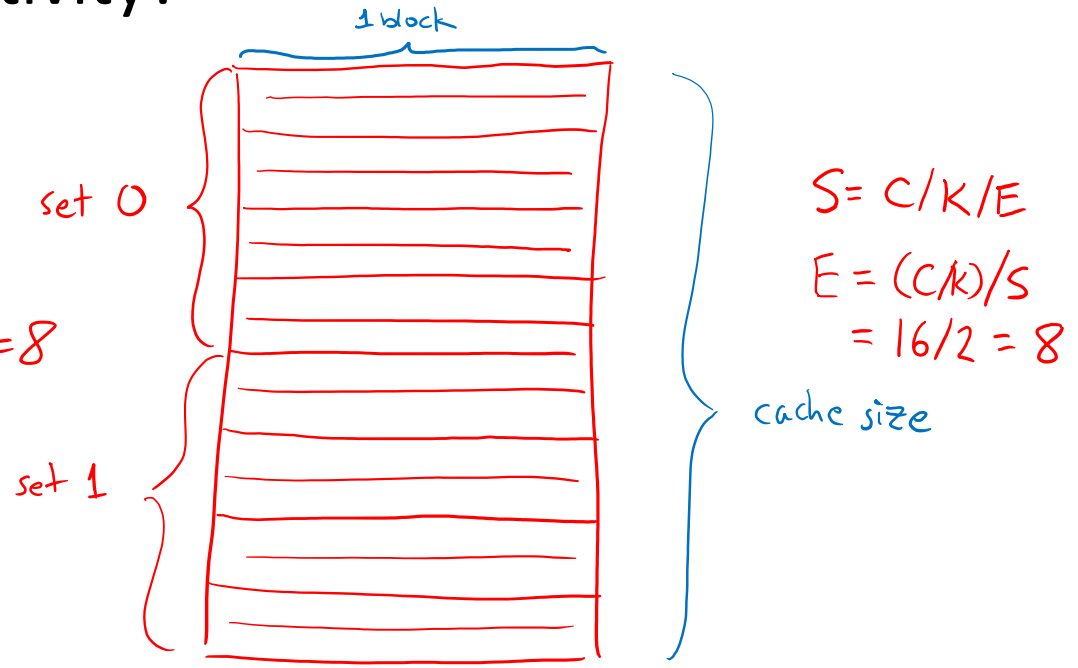
  C. 8

  D. 16

$C = 2^{11}$ B

$K = 2^7$ B

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



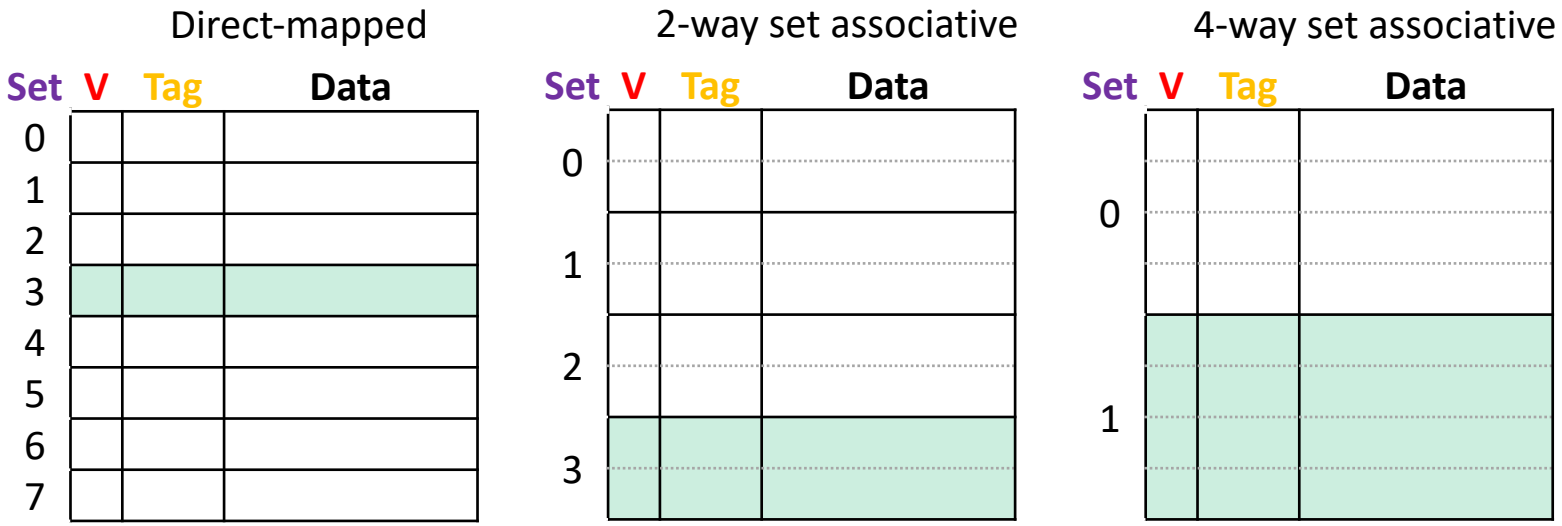
$m = 16$

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

$k = \log_2(K) = 7$ bits, $s = \log_2(S) = 1$ bit, $t = m - s - k =$ 8 bits

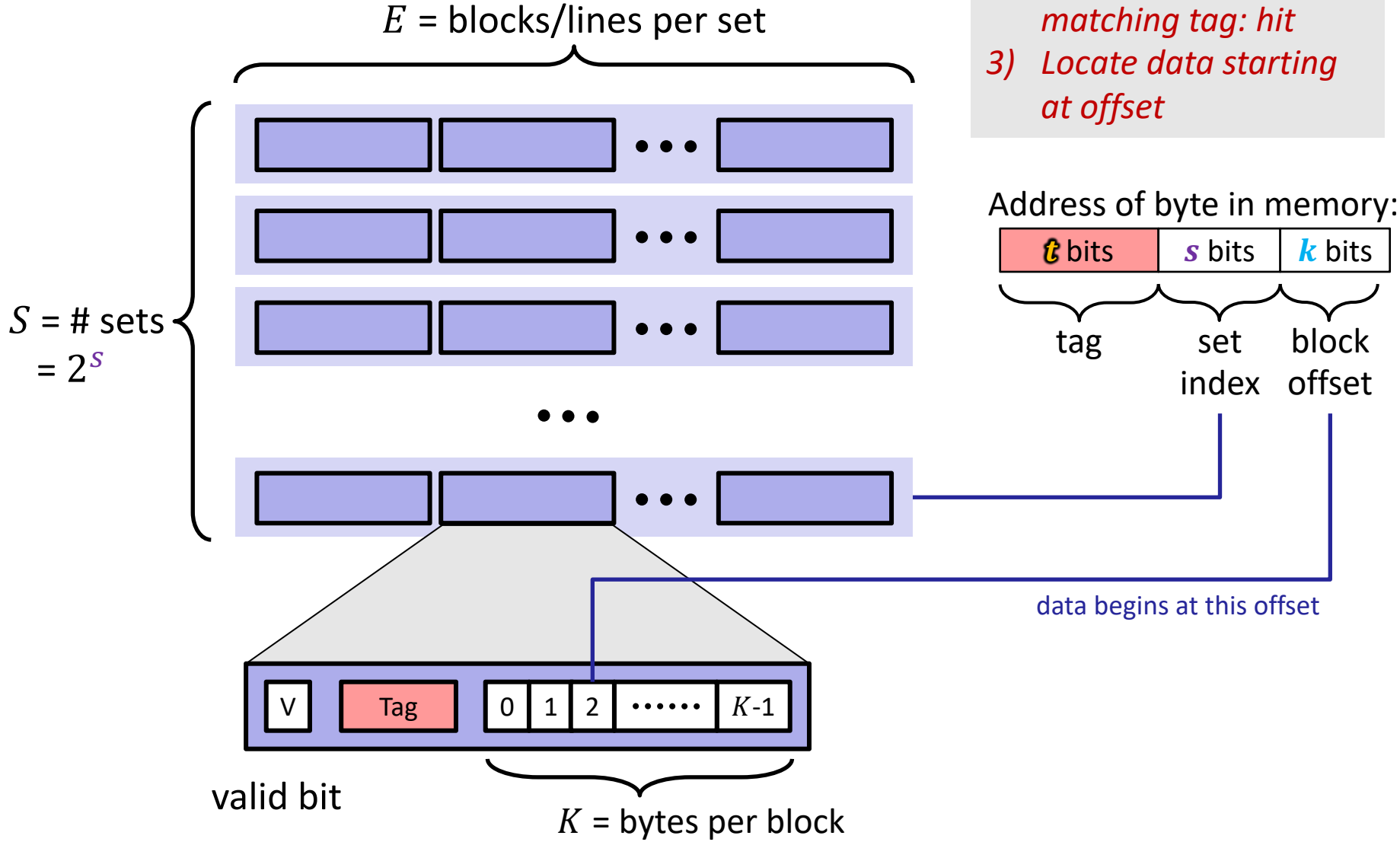
Block Placement and Replacement (Review)

- *Any* empty block in the correct set may be used to store block
  - **Valid bit** for each cache block indicates if valid (1) or mystery (0) data
- 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*")



Cache Read Revisited

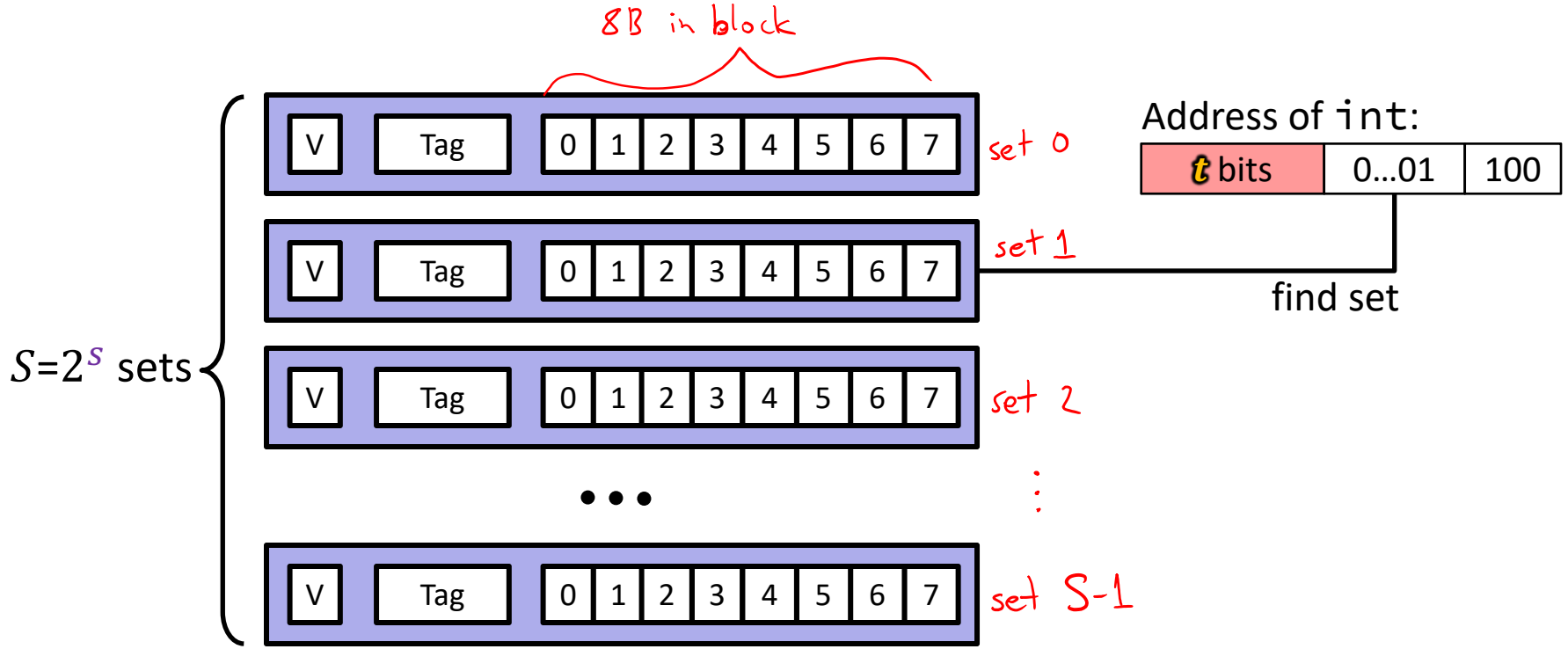
1) *Locate set*
2) *Check if any line in set is valid and has matching tag: hit*
3) *Locate data starting at offset*



Example: Direct-Mapped (E = 1) Cache (1/3)

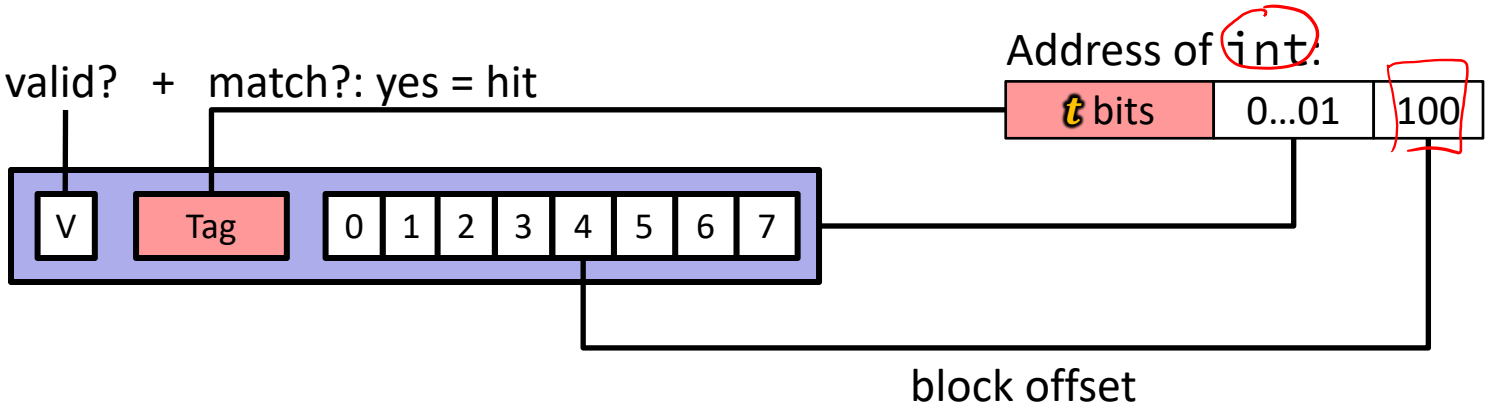
Direct-mapped: One line per set

Block Size K = 8 B



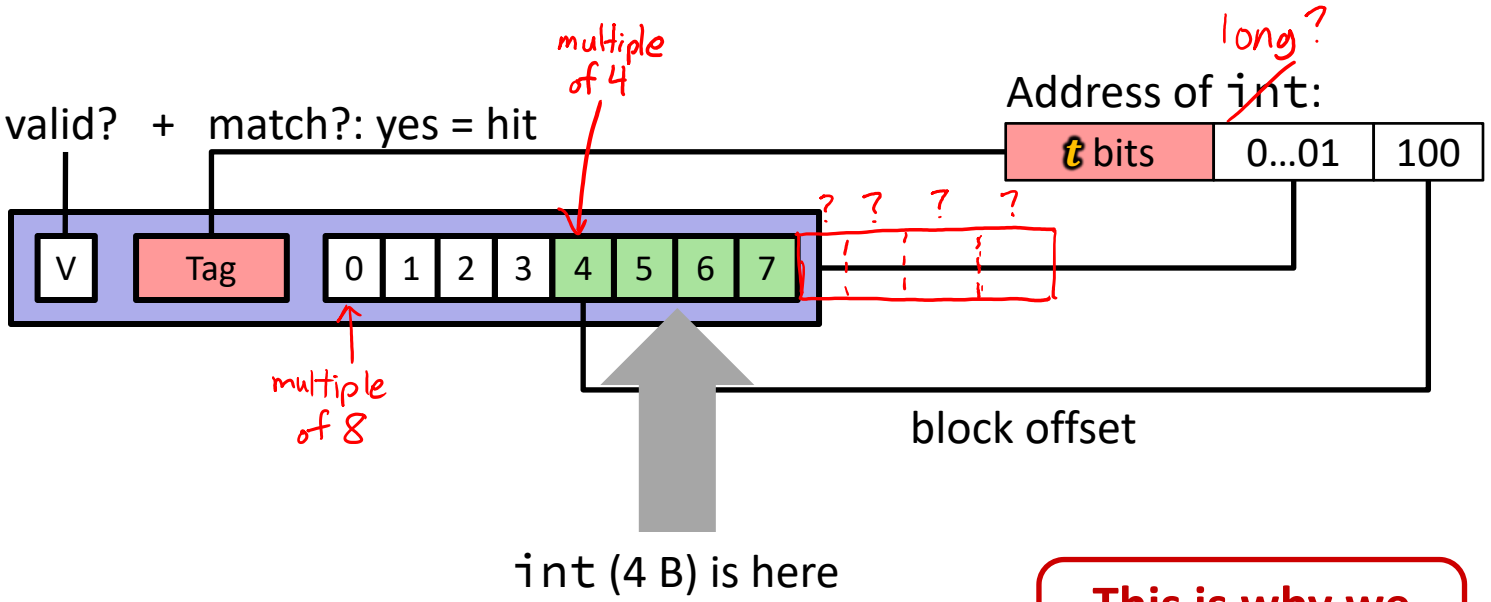
Example: Direct-Mapped (E = 1) Cache (2/3)

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



Example: Direct-Mapped (E = 1) Cache (3/3)

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



No match? Then old line gets evicted and replaced

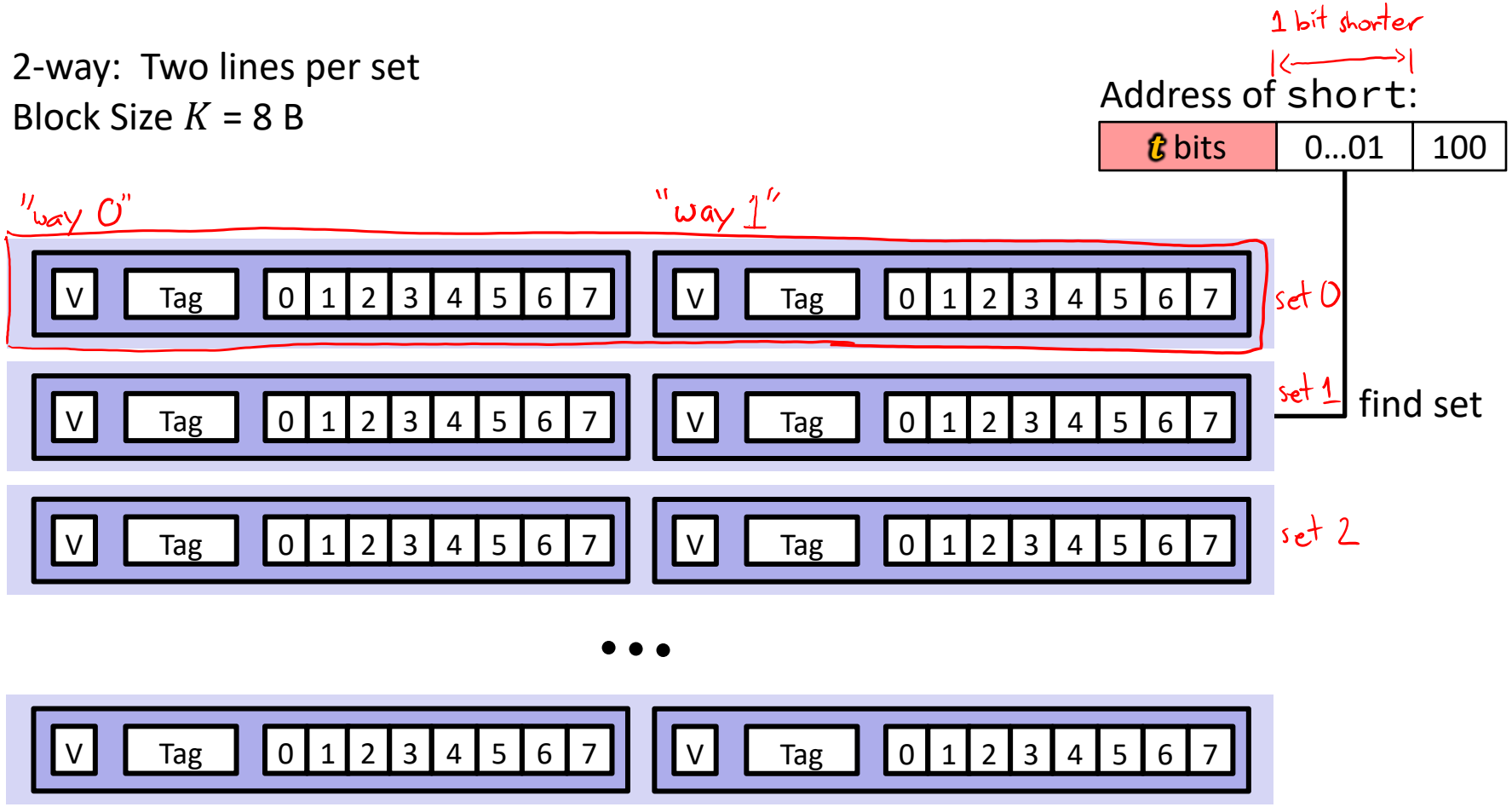
This is why we want alignment!

no unnecessary extra cache accesses across block boundaries

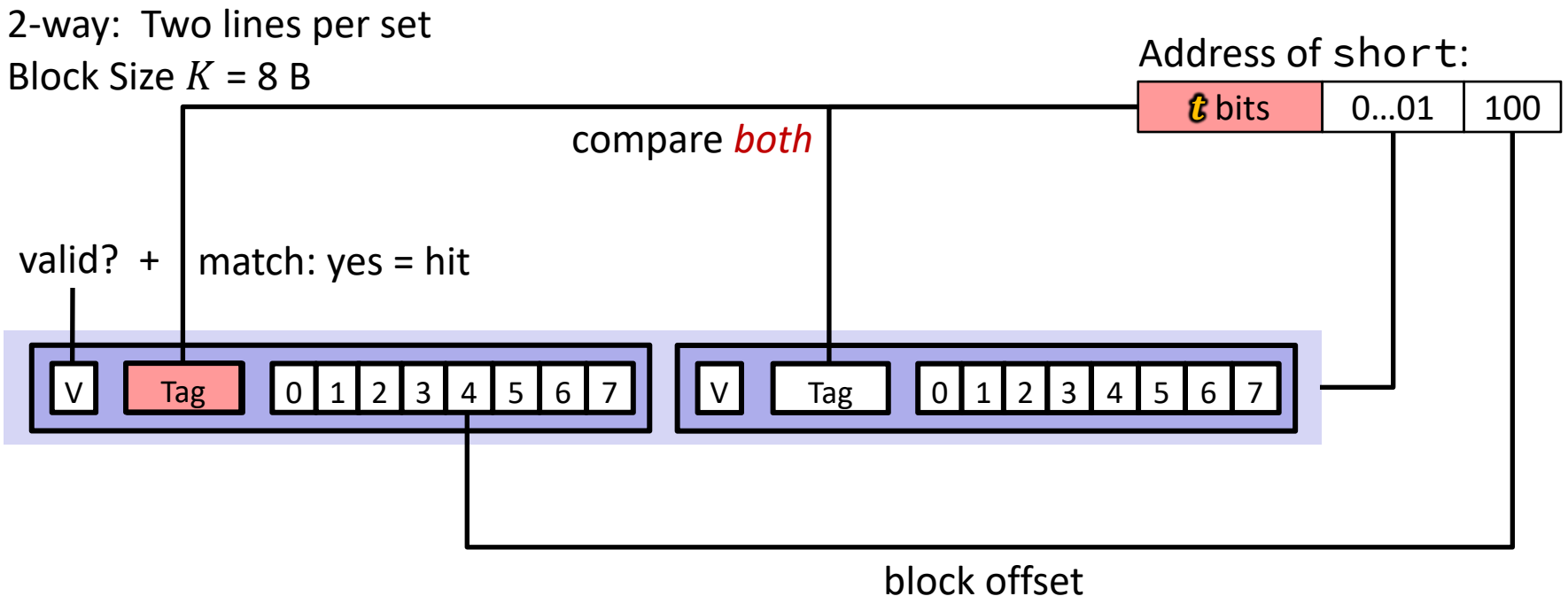
Example: Set-Associative (E = 2) Cache (1/?)

2-way: Two lines per set

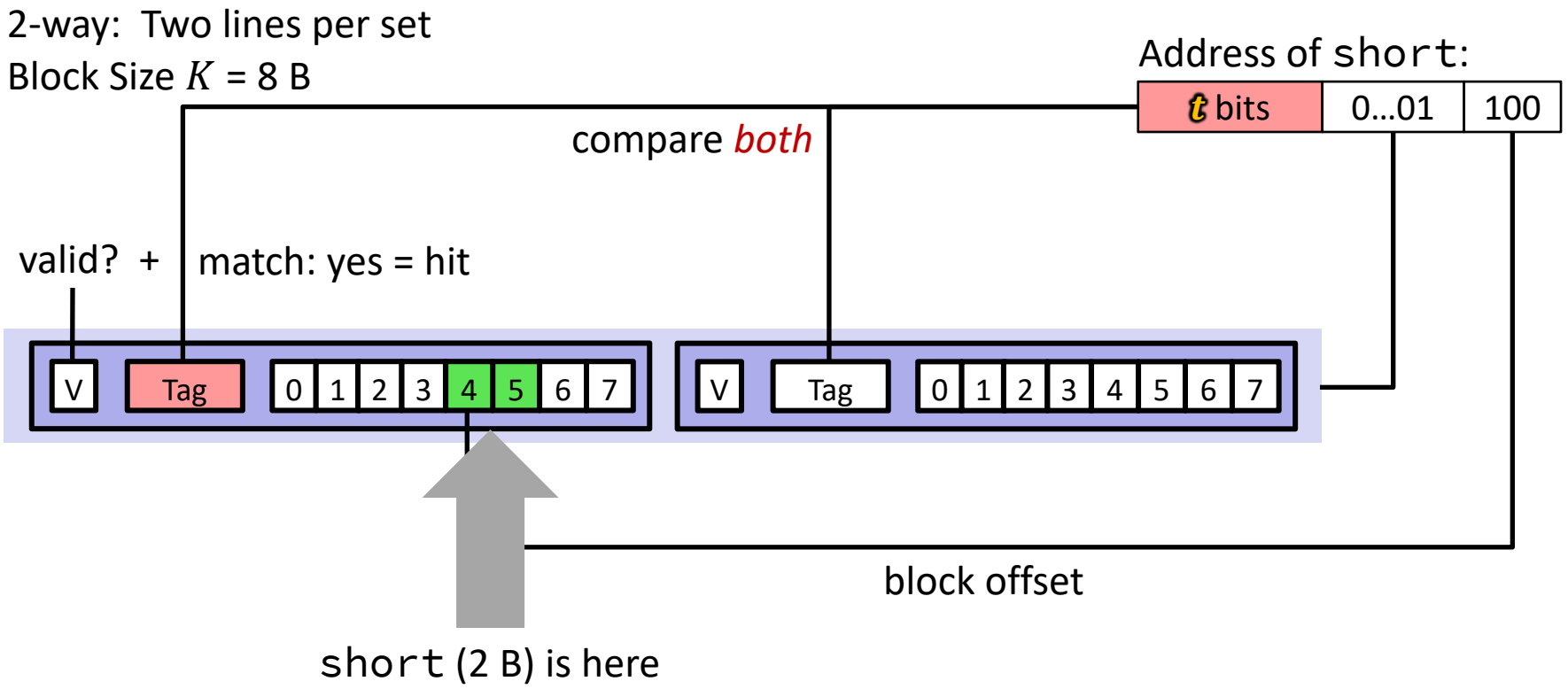
Block Size K = 8 B



Example: Set-Associative (E = 2) Cache (2/3)



Example: Set-Associative (E = 2) Cache (3/3)



No match?

- One line in set is selected for eviction and replacement
- Replacement policies: random, least recently used (LRU), …

Lecture Outline (2/3)

- Associativity and Replacement
- **Cache Organization**
- Example Cache Problems

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

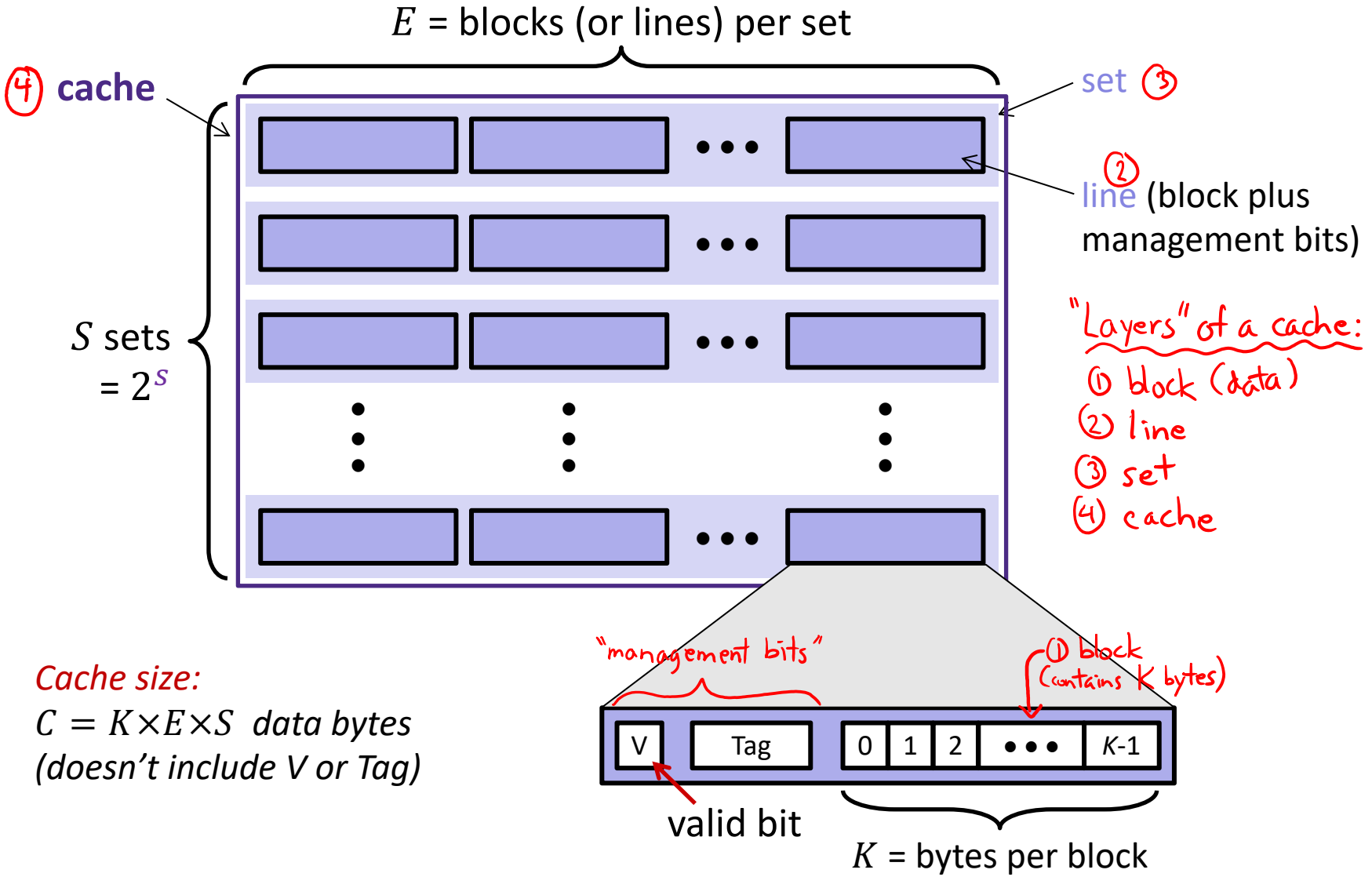
Parameter	Variable	Formulas
Block size	K (B in book)	$M = 2^{\boldsymbol{m}} \leftrightarrow \boldsymbol{m} = \log_2 M$ $S = 2^{\boldsymbol{s}} \leftrightarrow \boldsymbol{s} = \log_2 S$ $K = 2^{\boldsymbol{k}} \leftrightarrow \boldsymbol{k} = \log_2 K$ $C = K \times E \times S$ $\boldsymbol{s} = \log_2(C/K/E)$ $\boldsymbol{m} = \boldsymbol{t} + \boldsymbol{s} + \boldsymbol{k}$
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)	

General Cache Organization (S, E, K)

associativity

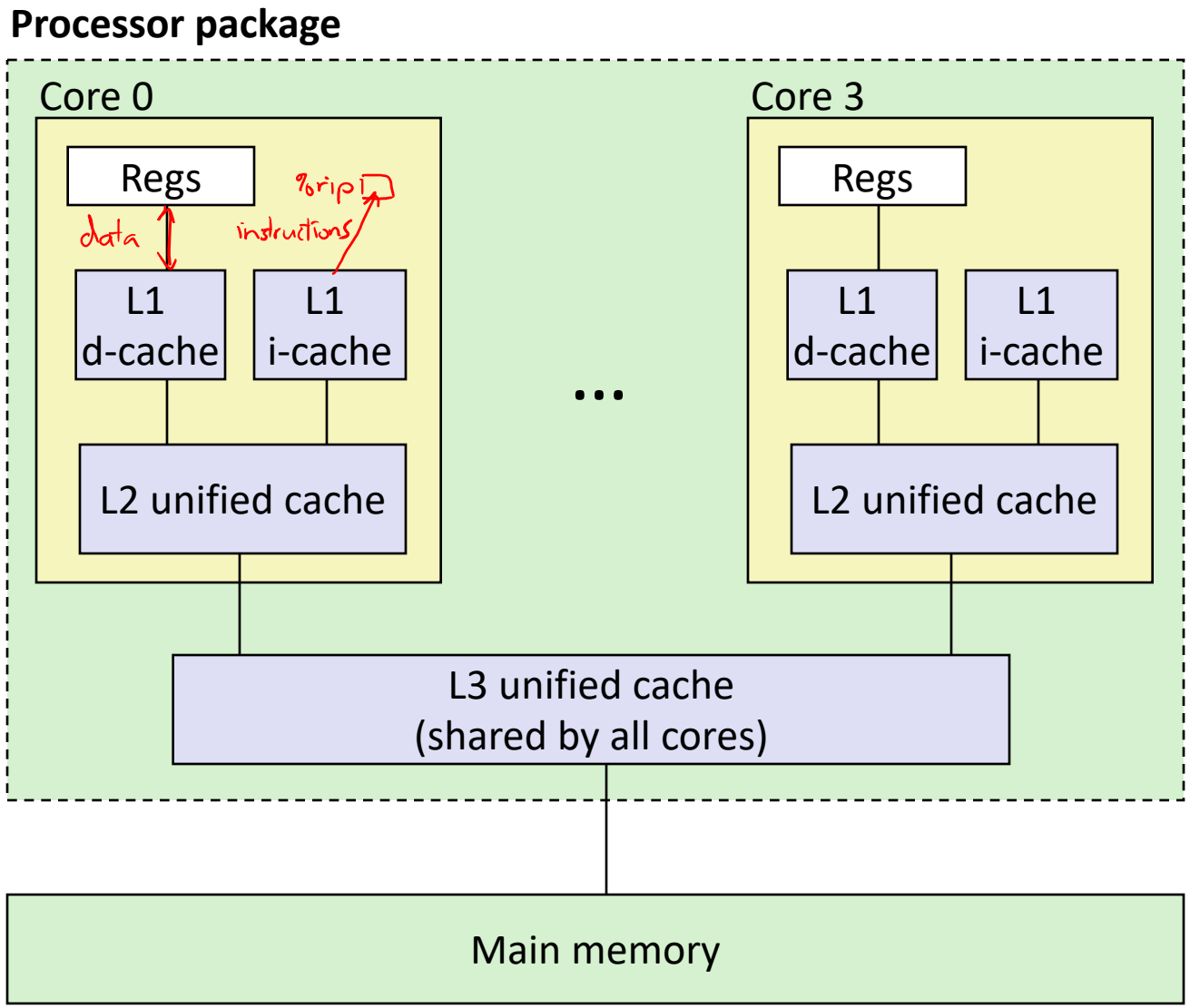
sets

block size



Cache size:
$C = K \times E \times S$ *data bytes*
(doesn't include V or Tag)

Intel Core i7 Cache Hierarchy



Block size:
64 bytes for all caches

L1 i-cache and d-cache:
32 KiB, 8-way,
Access: 4 cycles

L2 unified cache:
256 KiB, 8-way,
Access: 11 cycles

L3 unified cache:
8 MiB, 16-way,
Access: 30-40 cycles

Learning About Your Machine

- **Linux:**
  - `lscpu`
  - ls /sys/devices/system/cpu/cpu0/cache/index0/
    - Example: cat /sys/devices/system/cpu/cpu0/cache/index*/size
- **Windows:**
  - `wmic memcache get <query>` (all values in KB)
  - Example: `wmic memcache get MaxCacheSize`

- Modern processor specs: http://www.7-cpu.com/

Lecture Outline (3/3)

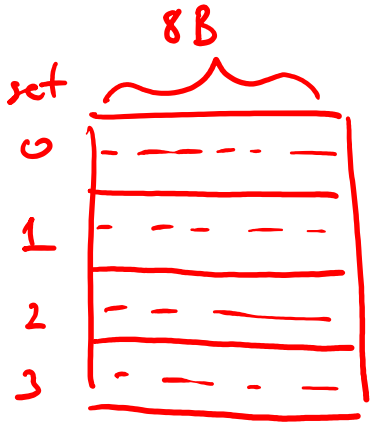
- Associativity and Replacement
- Cache Organization
- **Example Cache Problems**

Example Cache Parameters Problem

$\rightarrow 2^{10}$ B $\Longleftrightarrow$ m = 10 bits

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

MP



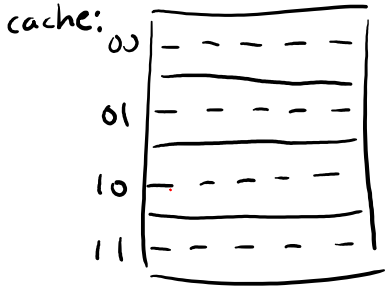
C	**Cache Size**	64 B	2^6
K	**Block Size**	8 B	2^3
E	**Associativity**	2-way	2^1
HT	**Hit Time**	3 cycles	
MR	**Miss Rate**	20%	
$t = m-s-k$	**Tag Bits**	5	
$s = \log_2(C/K/E)$	**Index Bits**	2	$2^6/2^3/2^1$
$k = \log_2(K)$	**Offset Bits**	3	
AMAT = HT + MR * MP	**AMAT**	3 + 0.2(125) = 28 clock cycles	

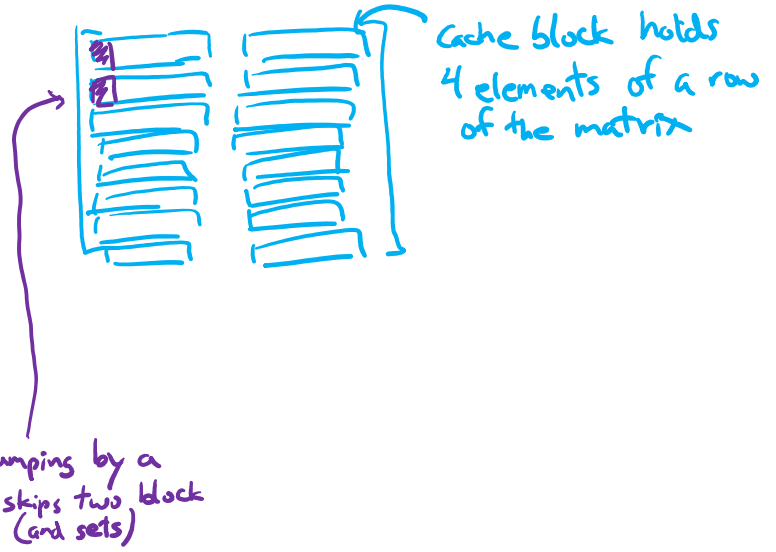
Example Code Analysis Problem

- Assuming the cache starts cold (all blocks invalid) and `sum`, `i`, and `j` are stored in registers, calculate the **miss rate**: 100%
  - m = 10 bits, C = 64 B, K = 8 B, E = 2 t = 5 bits, s = 2 bits, k = 3 bits

2 bytes per element

```
#define SIZE 8
short ar[SIZE][SIZE], sum = 0;   // &ar=0x200
for (int i = 0; i < SIZE; i++)
   for (int j = 0; j < SIZE; j++)
      sum += ar[j][i];
```





Cache Simulator

- https://courses.cs.washington.edu/courses/cse351/cachesim/
  - From course website: Simulators → Cache Simulator
  - Allows you to play around with the effects of cache parameters and policies
  - Lots of neat features like highlighting, hover text, ability to rewind and replay accesses, and copy-and-paste access patterns

- Ways to use:
  - Take advantage of "explain mode" and navigable history to test your own hypotheses and answer your own questions
  - A tutorial Ed lesson is available
  - Will be used in HW19 – Lab 4 Preparation

Cache Simulator Demo

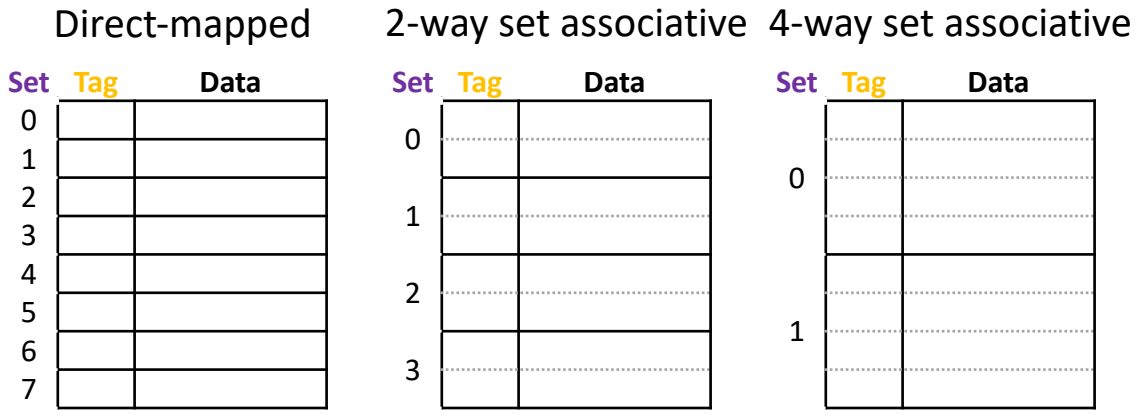
- https://courses.cs.washington.edu/courses/cse351/cachesim/
  - From course website: Simulators → Cache Simulator
  - Allows you to play around with the effects of cache parameters and policies
  - Lots of neat features like highlighting, hover text, ability to rewind and replay accesses, and copy-and-paste access patterns

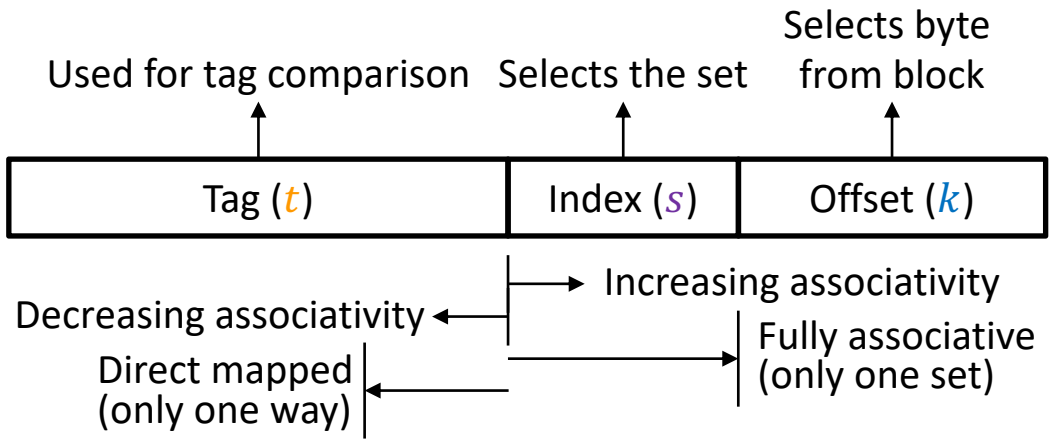
- Let's simulate the example code analysis problem:

```
#define SIZE 8
short ar[SIZE][SIZE], sum = 0;  // &ar=0x200
for (int i = 0; i < SIZE; i++)
   for (int j = 0; j < SIZE; j++)
      sum += ar[j][i];
```

Summary (1/2)

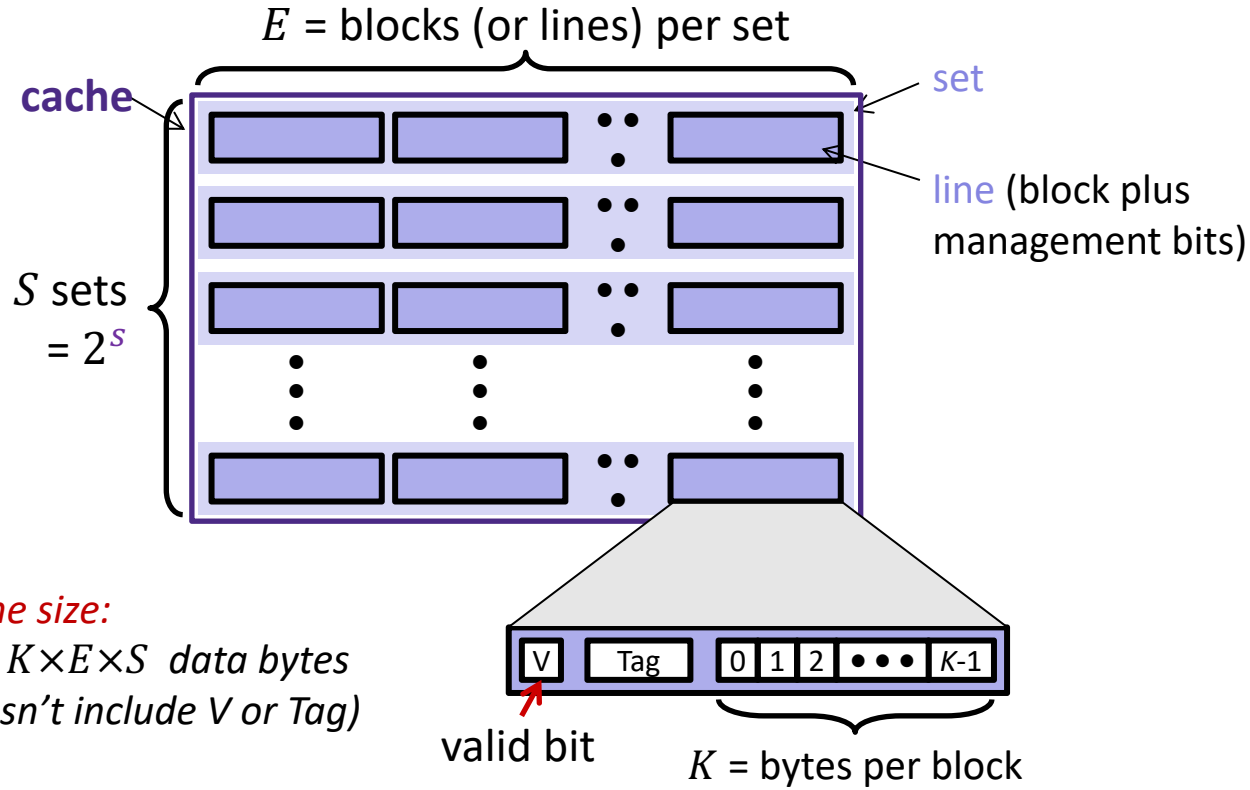
- **Associativity** gives us flexibility in where to place blocks in the cache
  - Group E slots into **sets**, means there are E **ways** to place block within each set
    - Direct-mapped is E = 1, **fully associative** is E = # of slots in cache (*i.e.*, S = 1)
    - Helps avoid conflicts in each set at the expense of slightly longer and more complex searching & placing in the cache
    - By default, we will replace the **least-recently used** block in a set
  - Index → Set, $S = (C/K)/E$, still $s = \log_2(S)$





Summary (2/2)

- Management bits
  - Information needed for proper management of the cache & its data, but not counted in the cache size
  - **Valid bit** for validity of data
  - **Tag bits** for identifying which block



Cache size:
$C = K \times E \times S$ *data bytes*
(doesn't include V or Tag)