Caches III

CSE 351 Summer 2020

Instructor:

Porter Jones

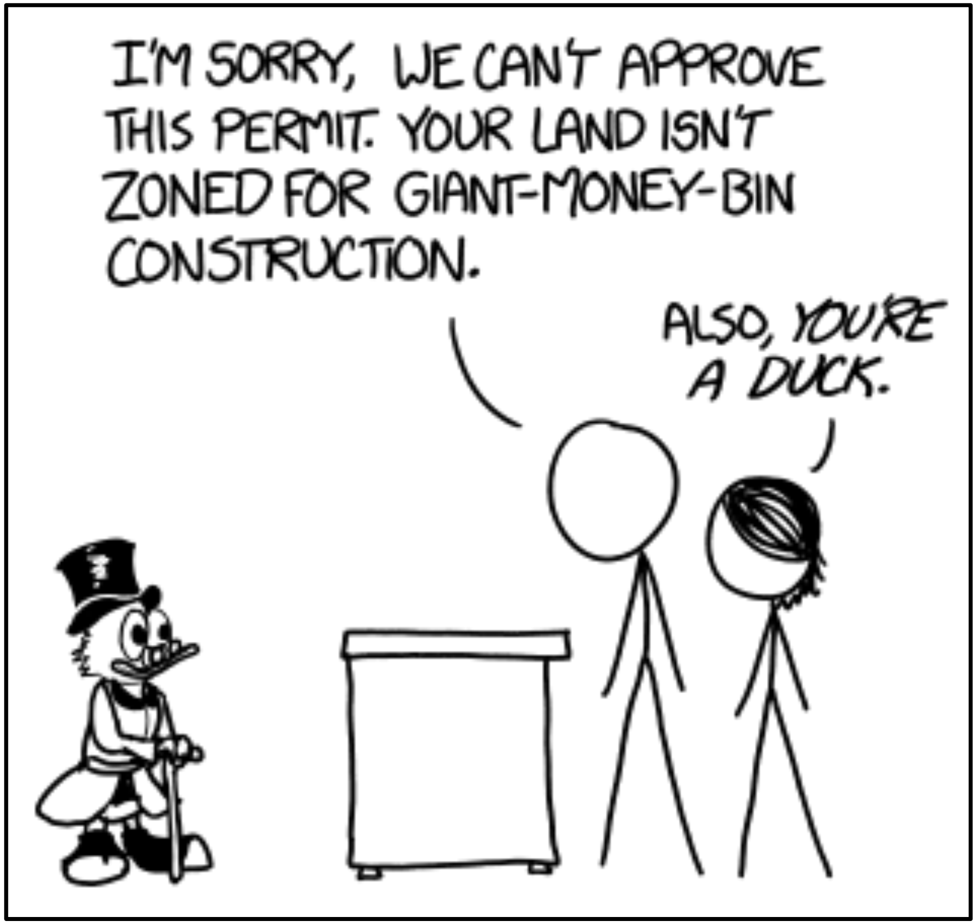
Teaching Assistants:

Amy Xu

Callum Walker

Sam Wolfson

Tim Mandzyuk


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

- Questions doc: https://tinyurl.com/CSE351-7-31
- hw16 due Wednesday (8/5) – 10:30am

  long, start early!
- 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

Course Staff 1-on-1s – see Piazza

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

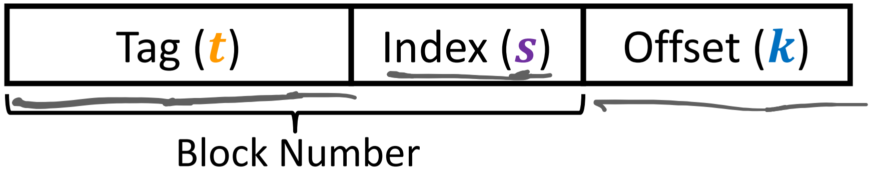
Review: Cache Parameters

- **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:

m-bit address:


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

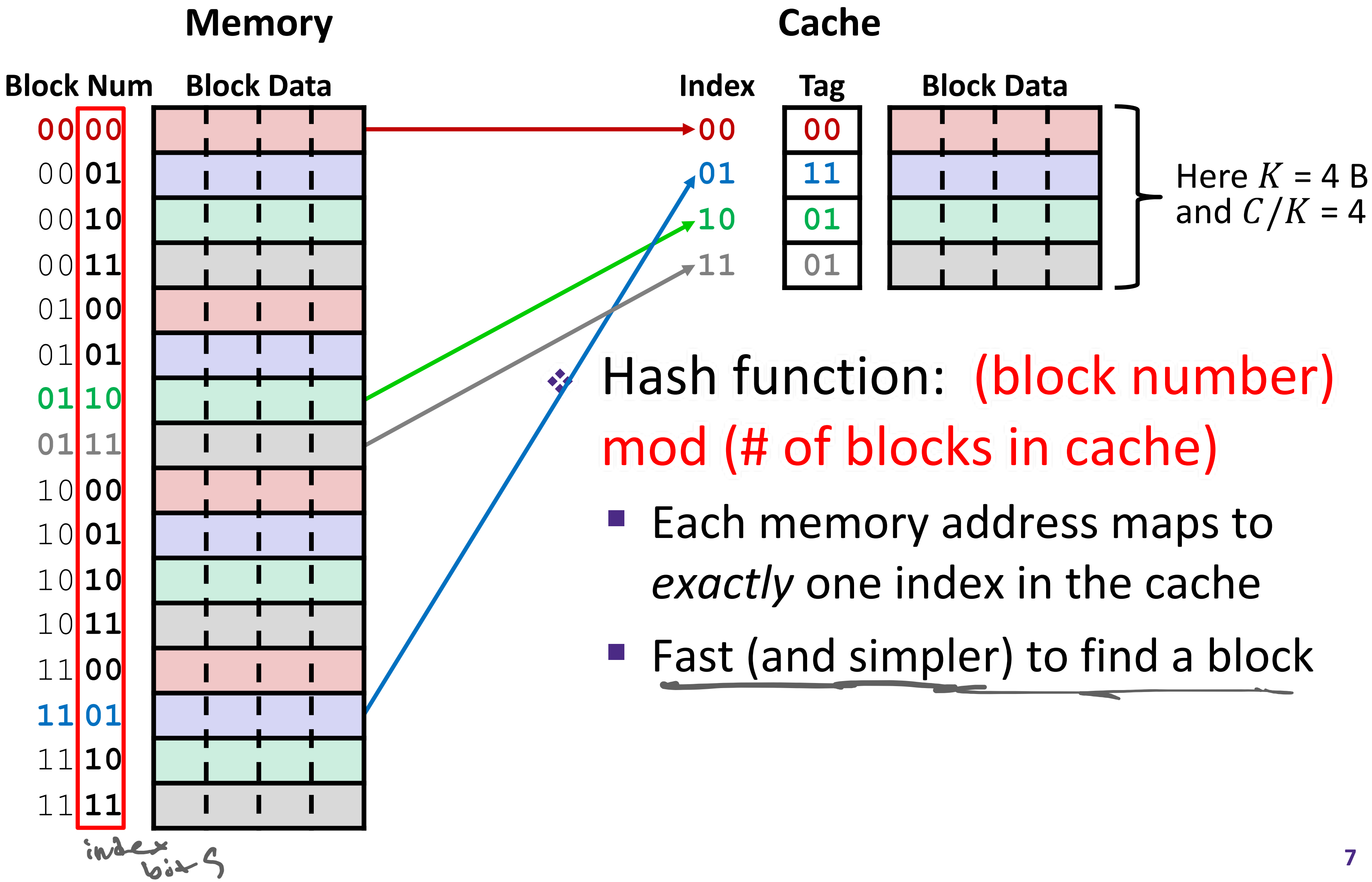
- **Index (s)** field tells you where to look in cache which set
  - Number of bits is determined by number of sets ($\log_2(\underline{C/K/E})$) $== \log_2(S)$
  - Need enough bits to reference every set in the cache
- **Tag (t)** field lets you check that data is the block you want
  - Rest of the 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

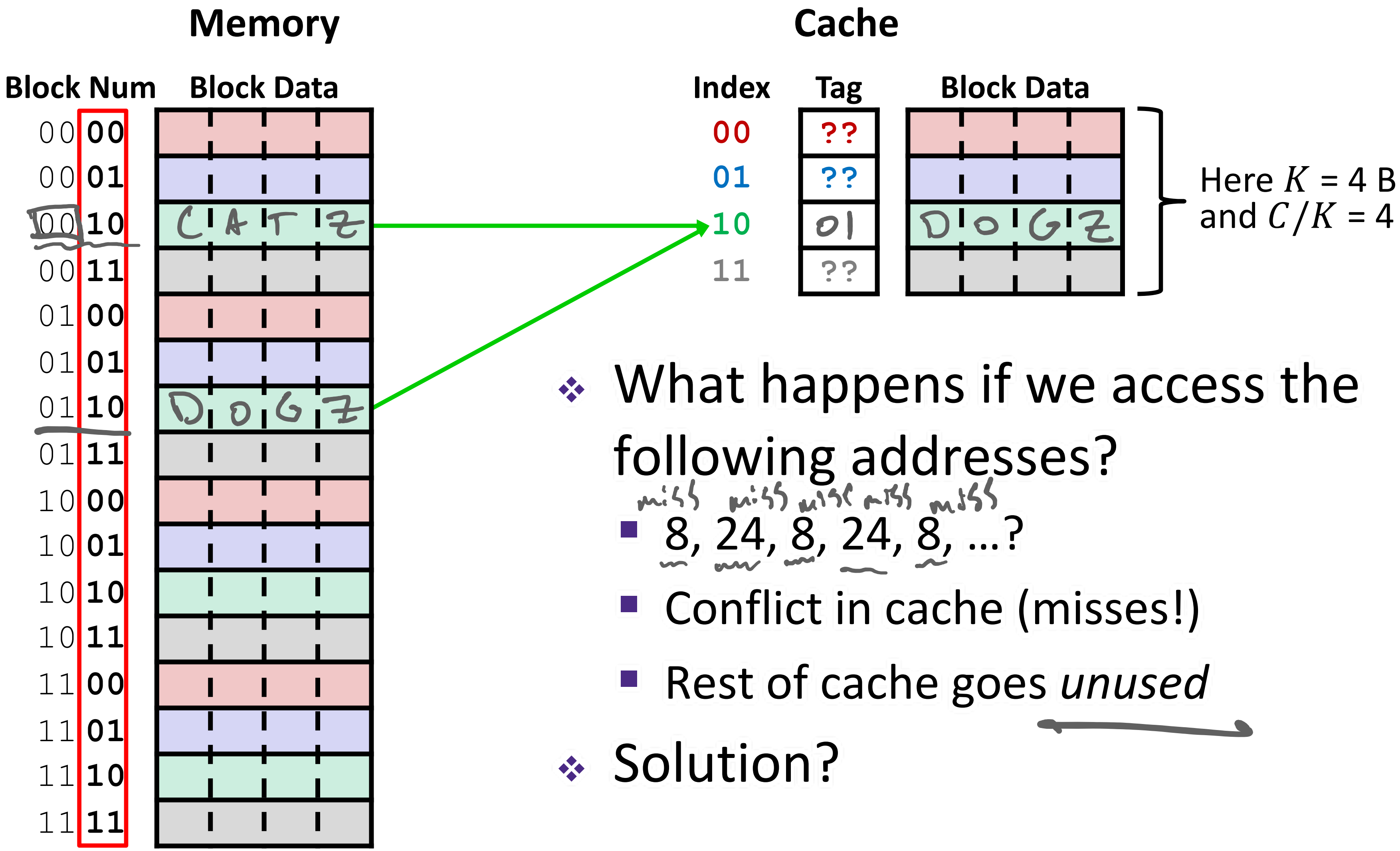
- If block is in the cache, it's a **cache hit** fast!
  - Data is returned to CPU starting at byte offset
- If block is not in the cache, it's a **cache miss** slow!
  - Block is loaded from memory into the cache, evicting other blocks from the cache if necessary
  - Data is returned to CPU starting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



Memory

Block Num	Block Data
0000	
0001	
0010	C A T Z
0011	
0100	
0101	
0110	D O G Z
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

Cache

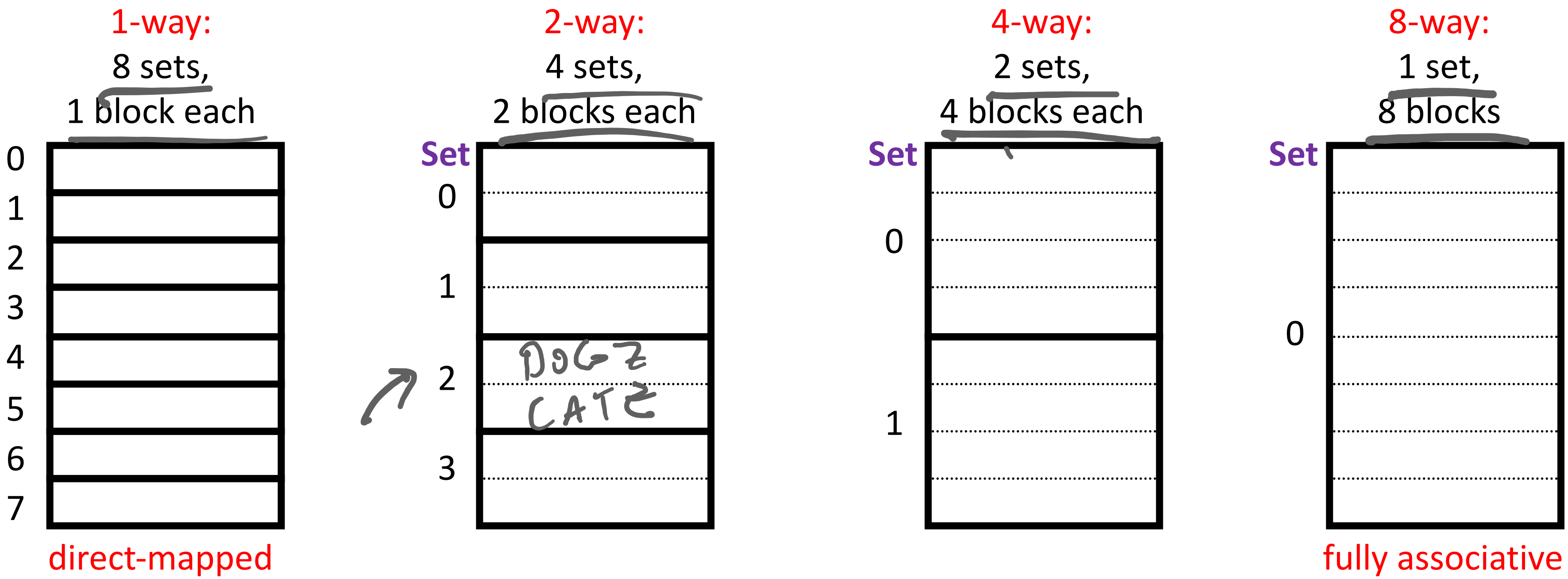
Index	Tag	Block Data
00	??	
01	??	
10	01	D O G Z
11	??	

Here K = 4 B and C/K = 4

- 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

- 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**



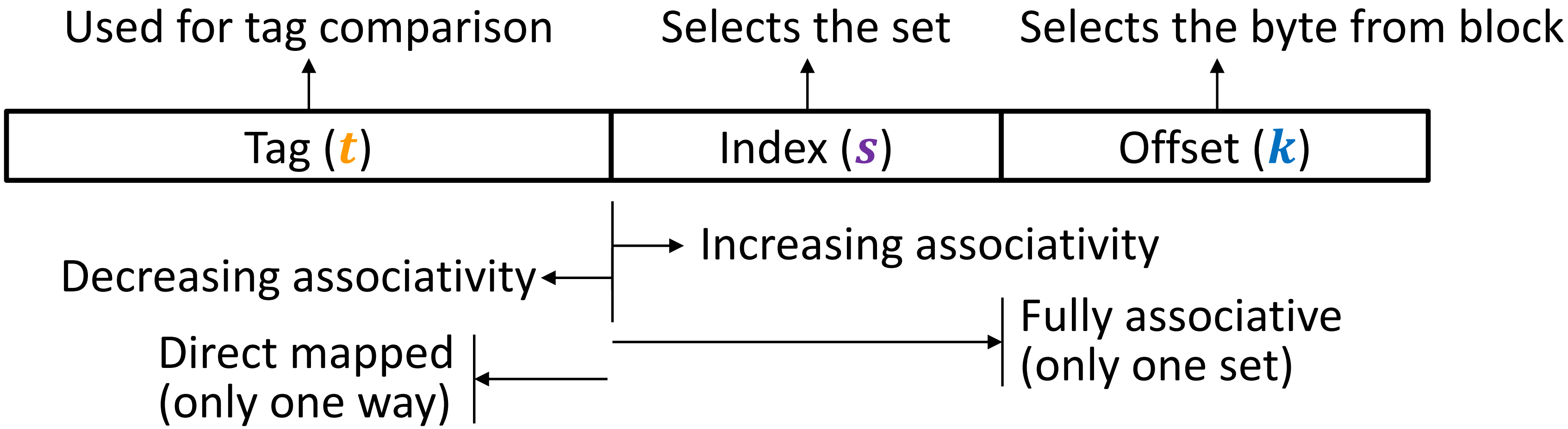
Cache Organization (3)

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 $\underline{S} = \underline{C/K}/E$
  - Use lowest $\log_2(C/K/E)$ = s bits of block address
    - Direct-mapped: $\underline{E = 1}$, so $\underline{s} = \log_2(\underline{C/K})$ as we saw previously
    - Fully associative: $\underline{E = C/K}$, so s = $\underline{0 \text{ bits}}$ one set!

num blocks

blocks per set


Used for tag comparison

Selects the set

Selects the byte from block

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

Decreasing associativity

Increasing associativity

Direct mapped (only one way)

Fully associative (only one set)

Example Placement

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

- Where would data from address `0x1833` be placed?
  - Binary: $\texttt{0b 0001 1000 }\overbrace{\texttt{0011}}^{E=1}\ \underbrace{\texttt{0011}}_{\text{offset}}$

E=4

E=2

$t = m{-}s{-}k$ $\quad s = \log_2(C/K/E)$ $\quad k = \log_2(\overset{16}{K}) = 4$

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

E=1, S= C/K/E = 8

s = ? $\log_2(8)=3$

Direct-mapped

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

E=2 S= C/K/E=4

s = ? $\log_2(4)=2$

2-way set associative

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

E=4 S= C/K/E=2

s = ? $\log_2(2)=1$

4-way set associative

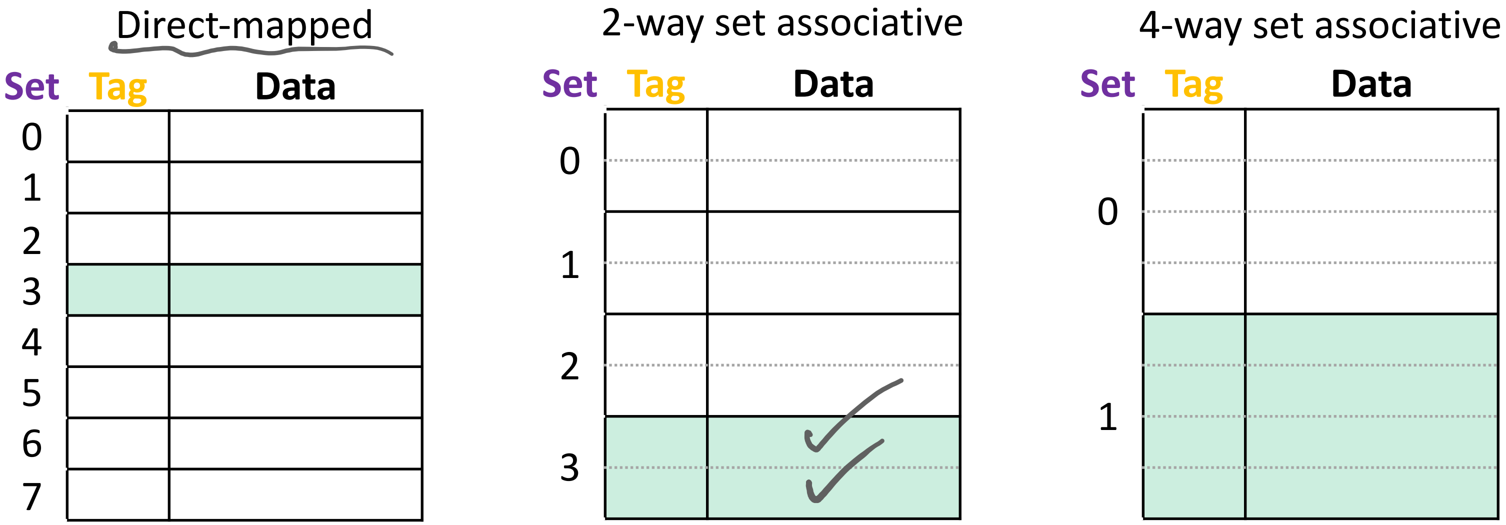
Set	Tag	Data
0 0		
1 1		✓
		✓
		✓
		✓

Block Replacement

valid = 0

- *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*")

hard to implement in hardware



Polling Question [Cache III]

$C = 2 * 2^{10} B$ $K = 2^7 B$

- We have a cache of size 2 KiB with block size of 128 B. If our cache has 2 sets, what is its associativity?

  $S = 2$

  - Vote at http://pollev.com/pbjones

A. 2

B. 4

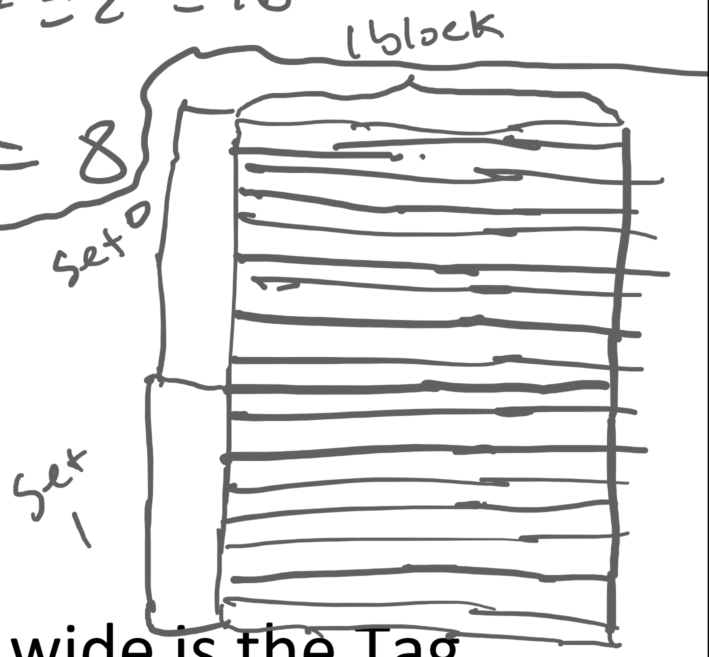
[✓ C. 8]

D. 16

E. We're lost…

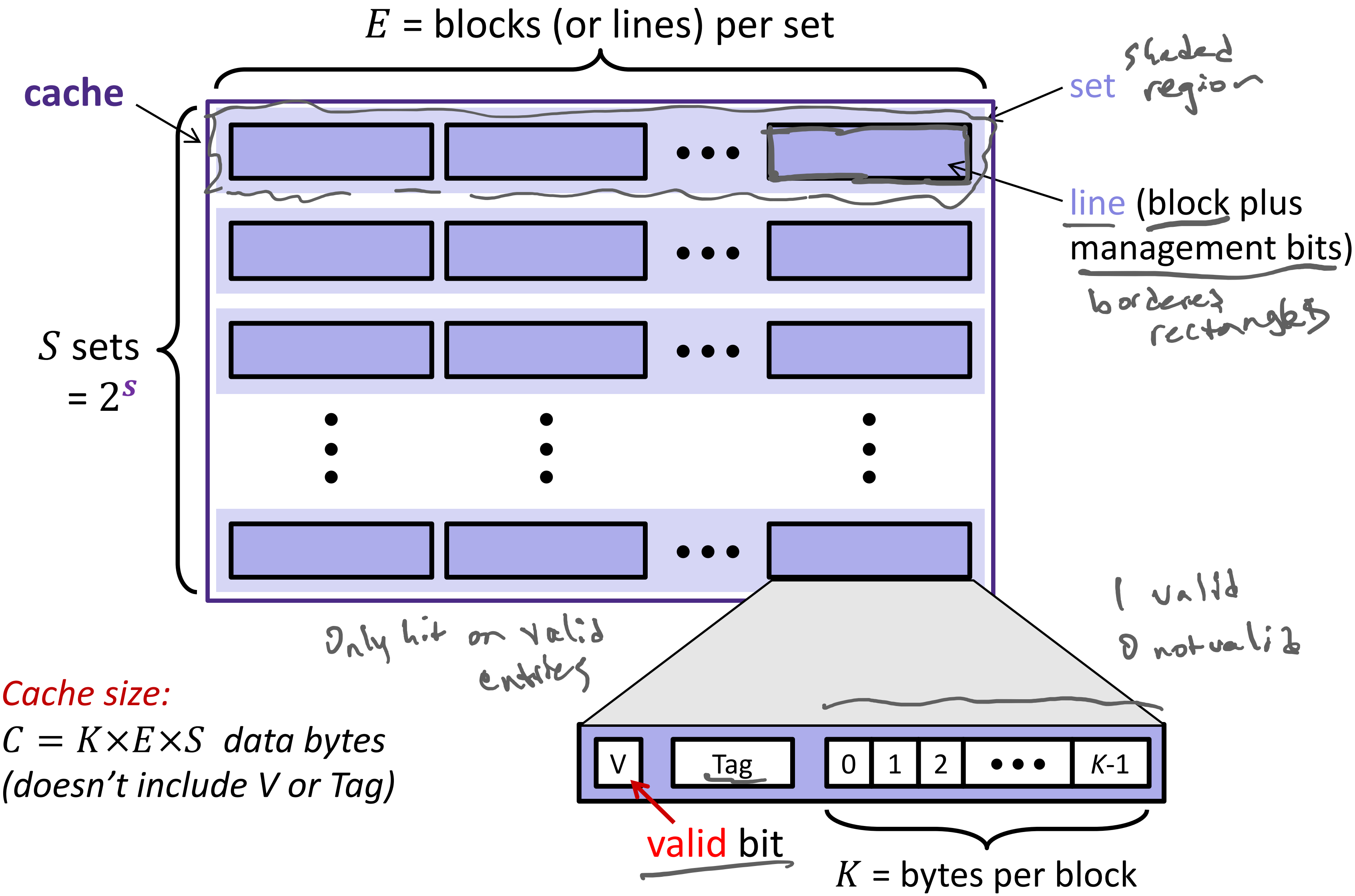
num blocks $= C/K = 2^{11}/2^7 = 2^4 = 16$ blocks

blocks per set $= E = 16/2 = 8$



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

General Cache Organization (S, E, K)



E = blocks (or lines) per set

cache

set

shaded region

line (block plus management bits)

bordered rectangles

S sets $= 2^s$

Only hit on valid entries

1 valid

0 not valid

V

Tag

valid bit

K = bytes per block

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

$2^2 * 2^{10} = 2^{12}$

MP

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

$C/K = 2^8/2^5 = 2^3 = 8$

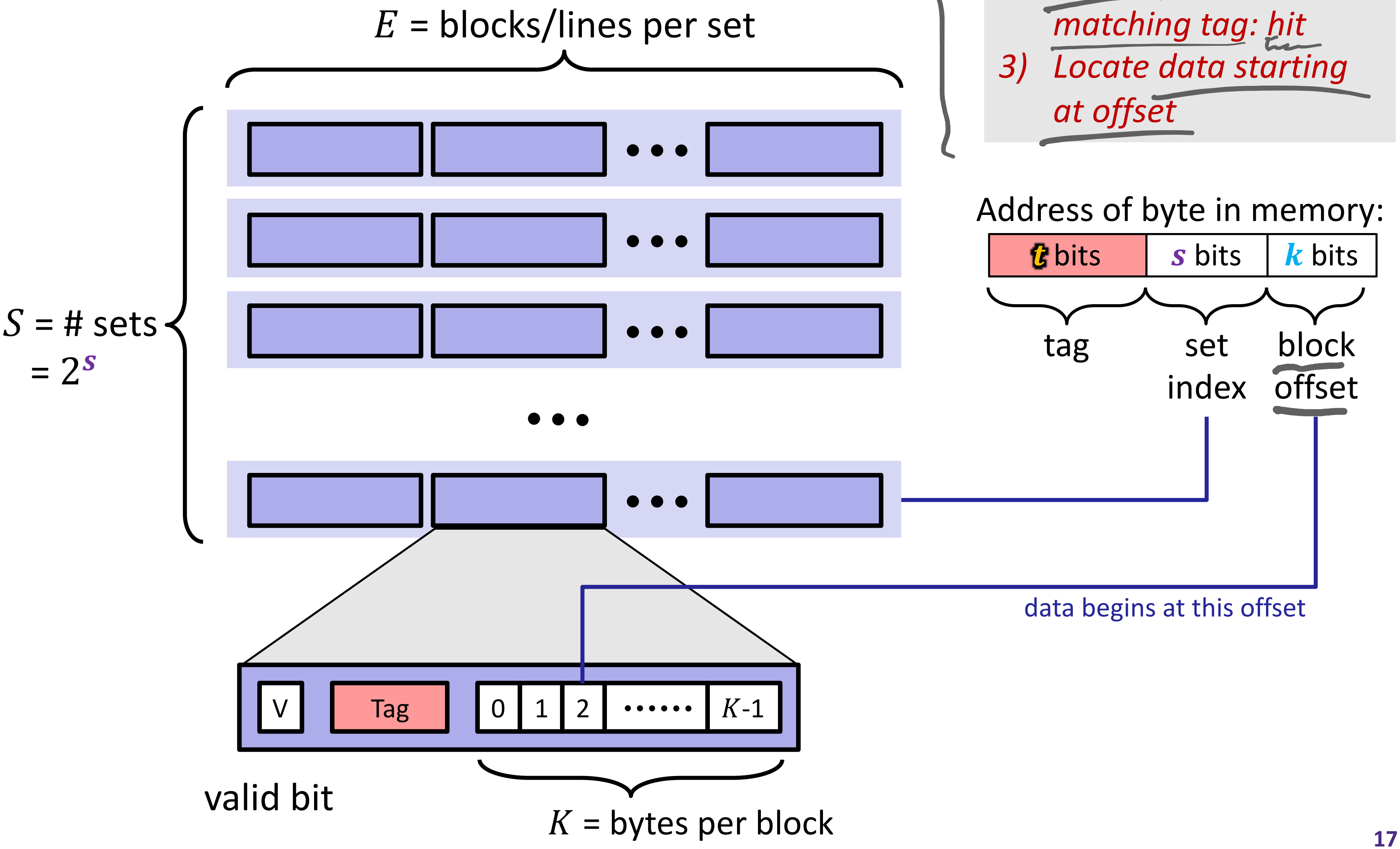
$C/K/E = 8/2 = 4$

$m = \log_2(M)$
$= \log_2(2^2 * 2^{10})$
$= \log_2(2^{12})$
$= 12$ bits

C	Cache Size	256 B	2^8
K	Block Size	32 B	2^5
E	Associativity	2-way	2
	Hit Time	3 cycles	HT
	Miss Rate	20%	MR
$m-s-k$	Tag Bits	$= 12 - 2 - 5 = 5$ bits	
$\log_2(C/K/E)$	Index Bits s	$= \log_2(4) = 2$ bits	
$\log_2(K)$	Offset Bits k	$= \log_2(32) = 5$ bits	
HT + MR*MP	AMAT	$= 3 + .2(125) = 28$ 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*



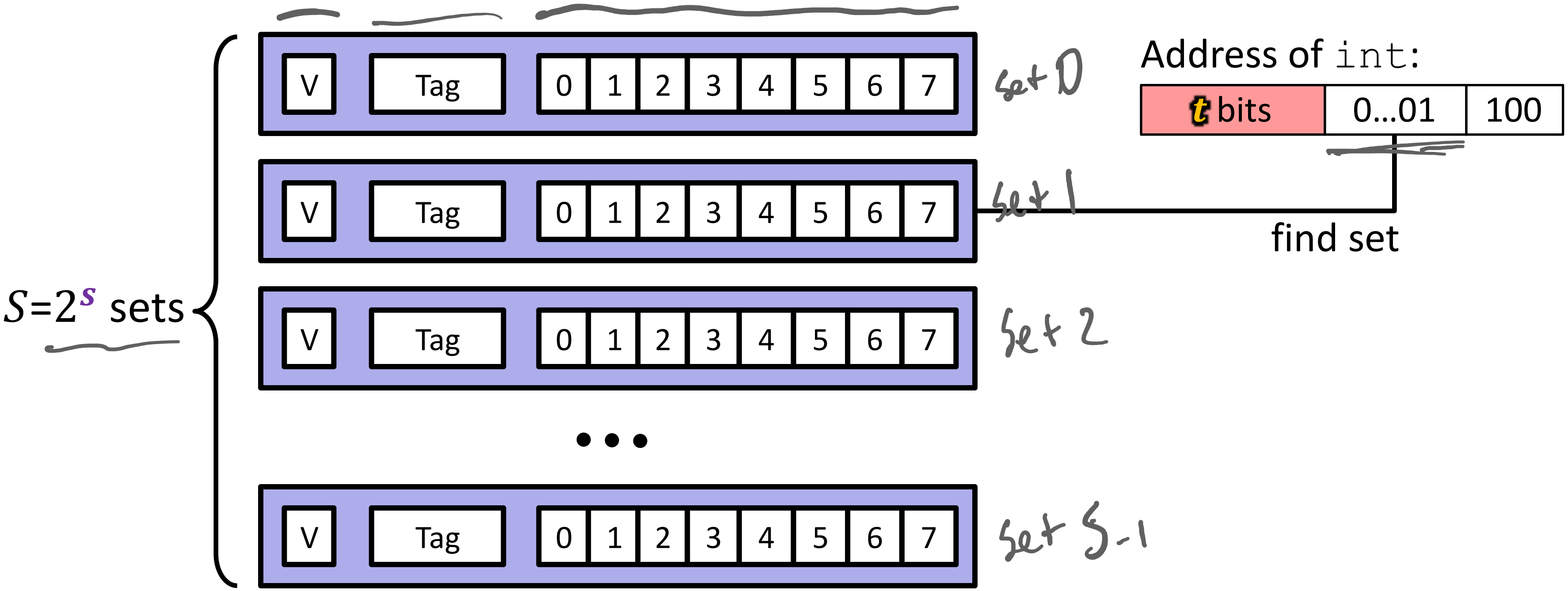
Example: Direct-Mapped Cache (E = 1)

one block per set

Direct-mapped: One line per set

Block Size K = 8 B

8 bytes per block



$S=2^s$ sets

Address of `int`:

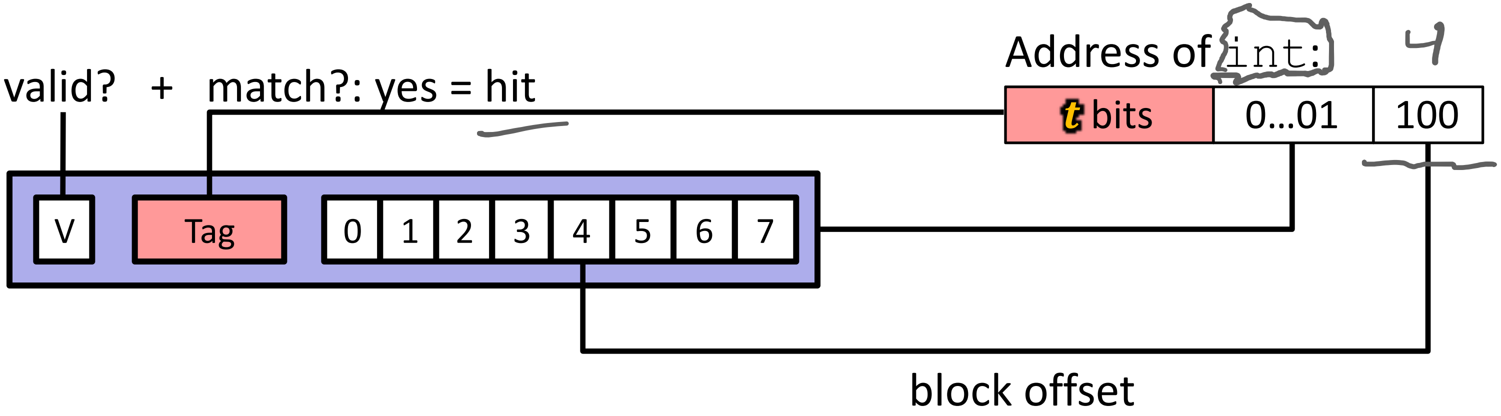
t bits	0…01	100

find set

Example: Direct-Mapped Cache (E = 1)

start reading at offset
read sizeof(int) bytes
4

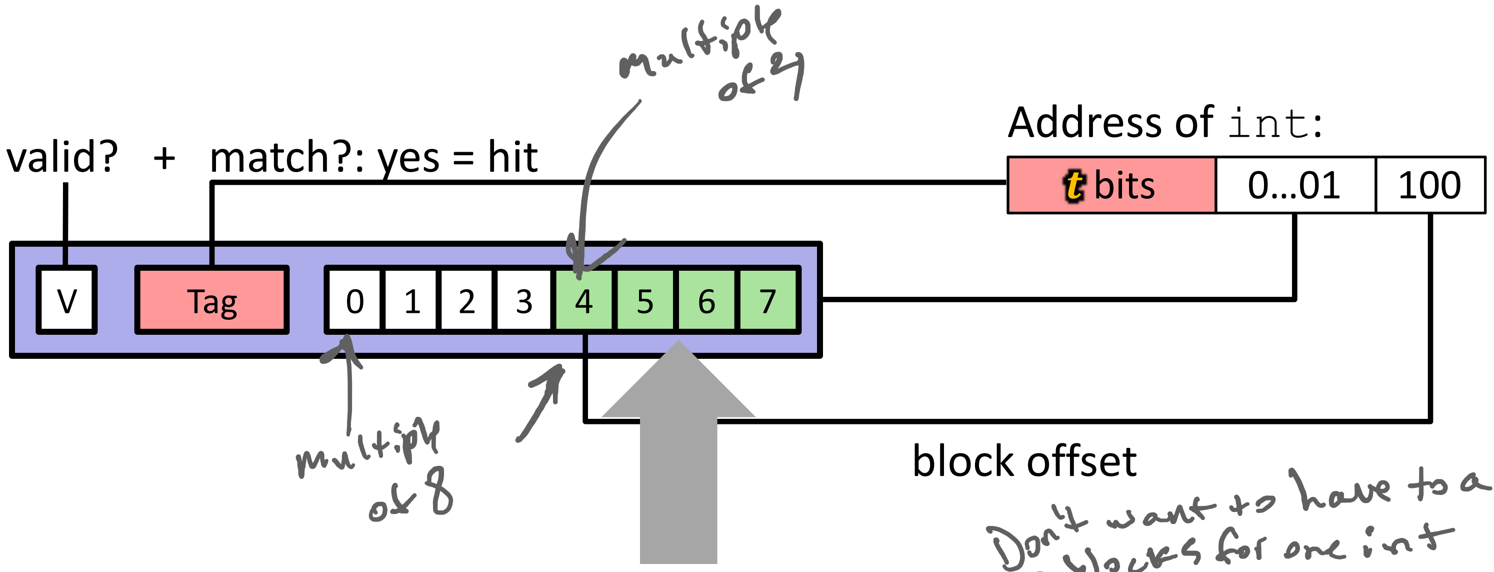
Direct-mapped: One line per set
Block Size K = 8 B



Example: Direct-Mapped Cache (E = 1)

Direct-mapped: One line per set

Block Size K = 8 B



int (4 B) is here

Don't want to have to access 2 blocks for one int

This is why we want alignment!

No match? Then old line gets evicted and replaced by the new block from memory

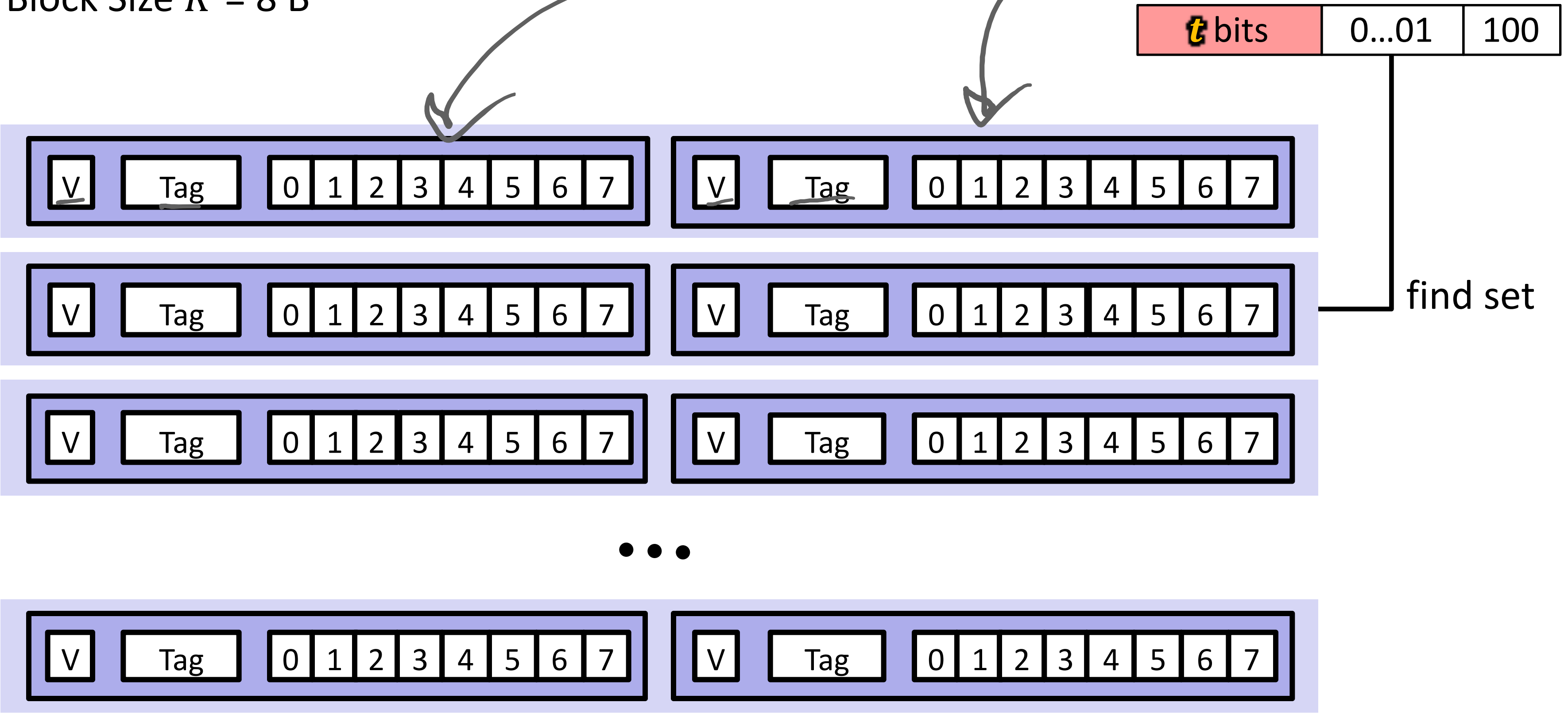
Example: Set-Associative Cache (E = 2)

2 blocks per set

2-way: Two lines per set

Block Size K = 8 B

Address of `short int`:



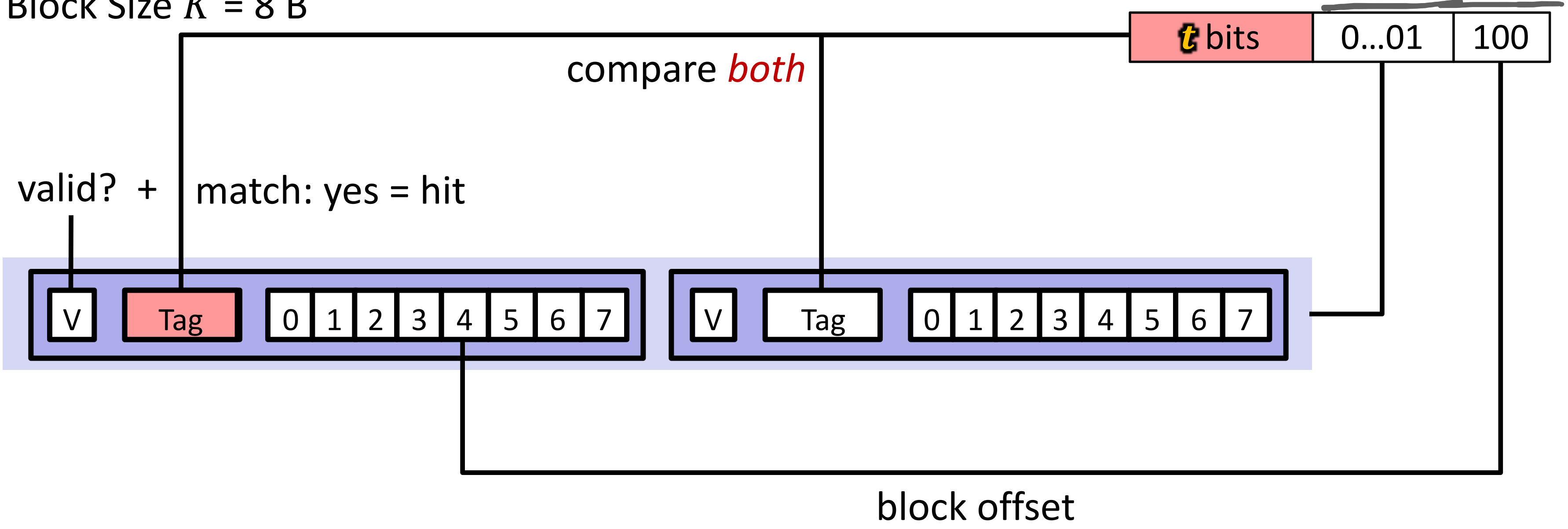
Example: Set-Associative Cache (E = 2)

2-way: Two lines per set

Block Size K = 8 B

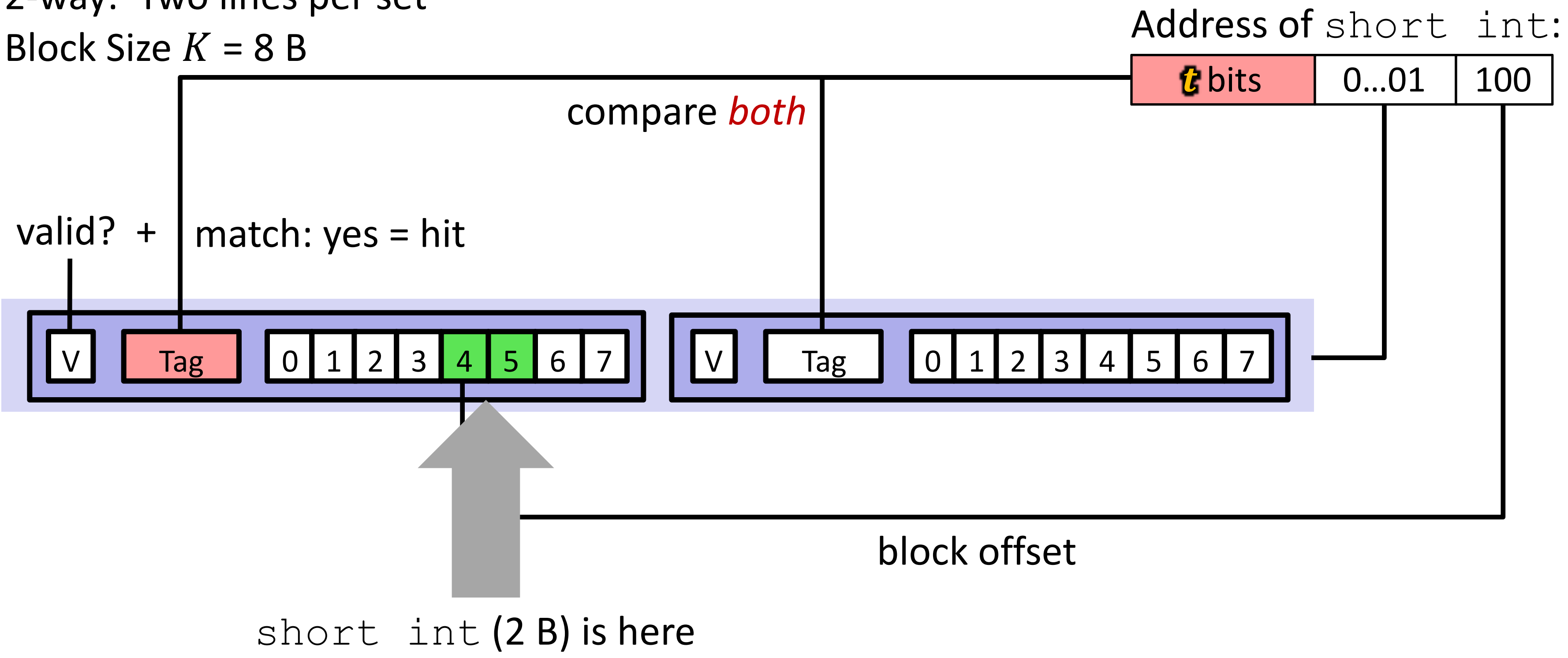
2 bytes

Address of `short int`:



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), …

not most recently used

Types of Cache Misses: 3 C's!

- Compulsory (cold) miss
  - Occurs on first access to a block
- Conflict miss

  to store data we are working with

  - 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*)

    one set

  - **Note:** *Fully-associative* only has Compulsory and Capacity misses

Example Code Analysis Problem

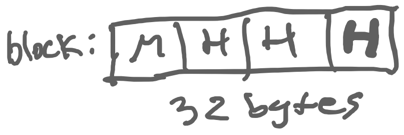
$k = \log_2(K) = 5$ bits
$s = \log_2(S) = 2$ bits
$t = m - k - s = 5$ bits

- Assuming the cache starts cold (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 $S = C/K/E = 4$

8 byte elements

```
#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];
```

4 longs



$\text{ar[0][0]} \xrightarrow{\text{addr}} 0\text{x}800 = 0\text{b}\underbrace{1000\ 0}_{\text{tag}}\underbrace{00}_{\text{index}}\underbrace{0\ 0000}_{\text{offset}}$ miss! (compulsory)

$\text{ar[0][1]} \rightarrow 0\text{x}808 = 0\text{b}\overbrace{1000\ 0}^{\text{tag}}\overbrace{00}\overbrace{0\ 1000}$ hit!

ar[0][2] → 0x810 = same block, hit!

ar[0][3] → 0x818 = same block, hit!

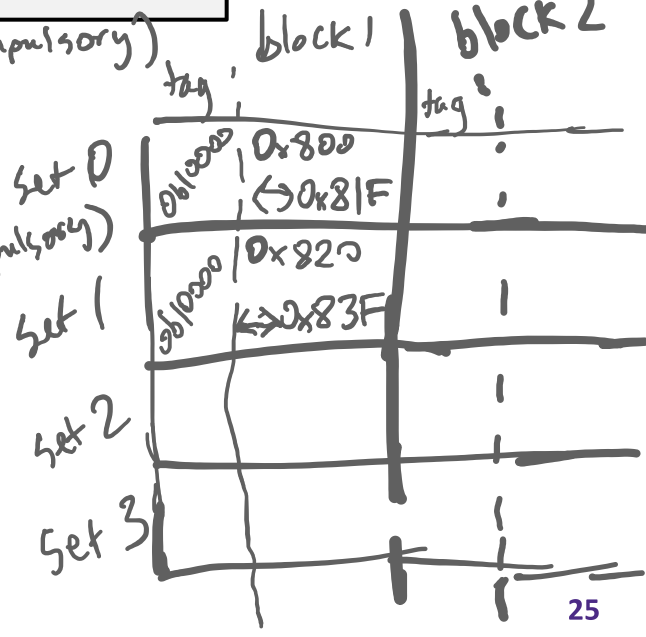
$\text{ar[0][4]} \rightarrow 0\text{x}820 = 0\text{b}\underbrace{1000\ 0}_{\text{tag}}\underbrace{01}_{\text{set}}\underbrace{0\ 0000}_{\text{offset}}$ miss! (compulsory)

ar[0][5] → 0x828 hit!

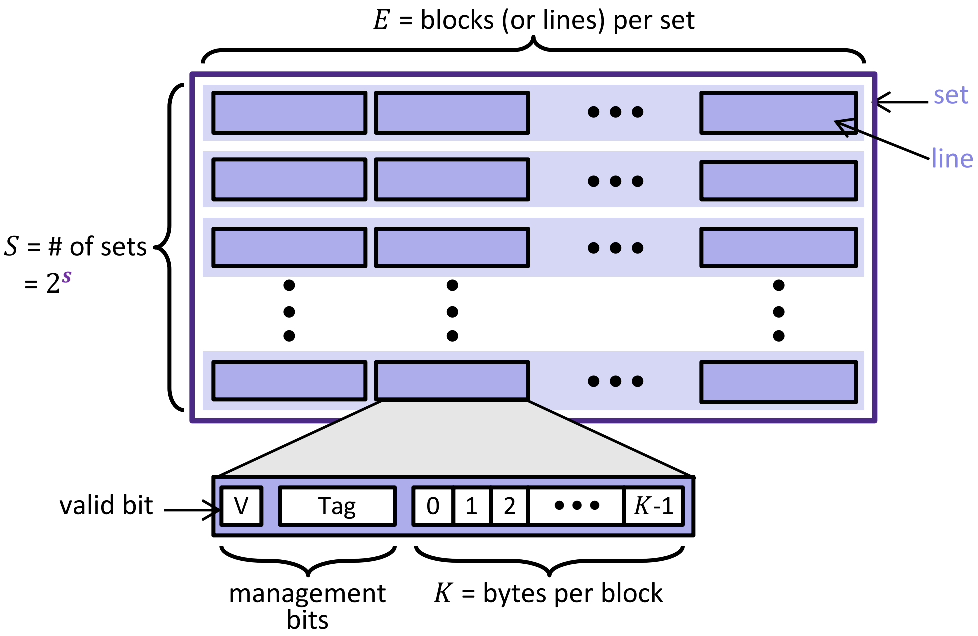
Pattern continues...

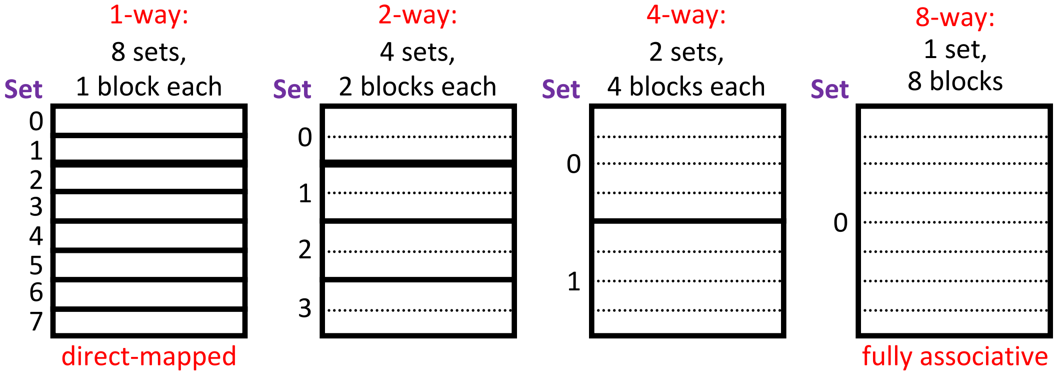
4 longs/accesses per block, one compulsory miss, then 3 hits

miss rate = 1/4



Notes Diagrams


E = blocks (or lines) per set
set
line
S = # of sets
= 2^s
valid bit
V
Tag
0
1
2
• • •
K-1
management bits
K = bytes per block


1-way:
8 sets,
1 block each
Set
0
1
2
3
4
5
6
7
direct-mapped
2-way:
4 sets,
2 blocks each
Set
0
1
2
3
4-way:
2 sets,
4 blocks each
Set
0
1
8-way:
1 set,
8 blocks
Set
0
fully associative