Memory & Caches III

CSE 351 Summer 2022

Instructor:
Kyrie Dowling

Teaching Assistants:
Aakash Srazali
Allie Pfleger
Ellis Haker

Sometimes I wonder what kind of bugs they will have in 100 or 200 years time…
Technology will be so advanced… I'm sure they'll have bugs we can't even imagine now…
Yeah…!
It'll be like unexpected bugs caused by corrupt strands of DNA or because of desynchronised quantum calculators!
Haha yeah it would be crazy!
Somewhere, in the year 2162
THE CACHE!
Just. Clear. Your. Cache!
CommitStrip.com

Relevant Course Information

- Unit Portfolio 2 due Wednesday (8/3)
  - UP1 feedback released on Canvas
- Lab 4 released, due (8/10)
  - Can do Part 1 after today; will need Lecture 18 to do Part 2
- hw16 due Friday (8/5)
  - Covers the major cache mechanics – BIG homework
- hw18 due Monday (8/8)
  - Released after Wednesday, Preparation for Lab 4
- Office Hours update

Making memory accesses fast!

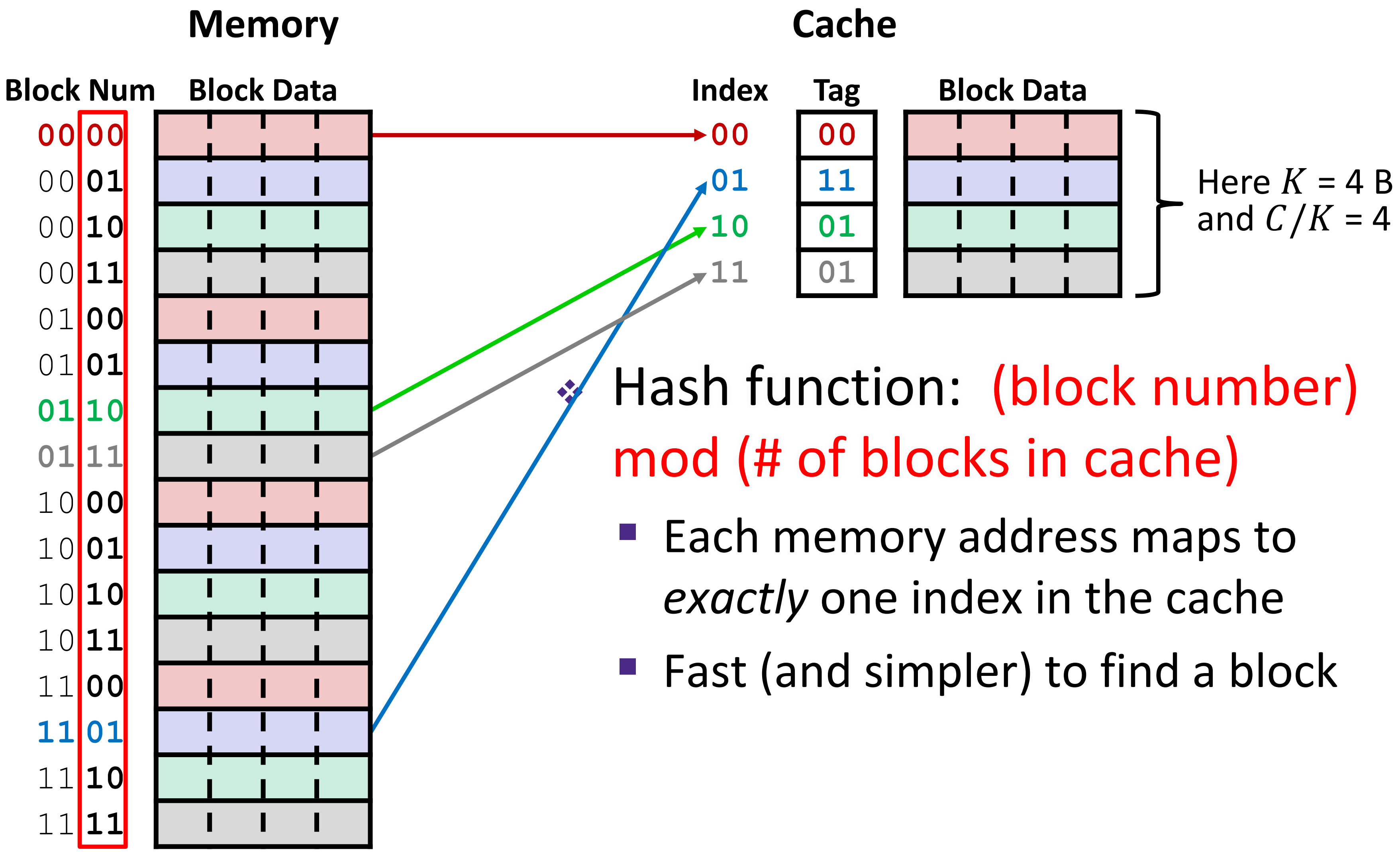
- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
  - Direct-mapped (*sets*; index + tag)
  - **Associativity (*ways*)**
  - **Replacement policy**
  - Handling writes
- Program optimizations that consider caches

Reading Review

- Terminology:
  - Associativity: sets, fully-associative cache
  - Replacement policies: least recently used (LRU)
  - Cache line: cache block + management bits (valid, tag)
  - Cache misses: compulsory, conflict, capacity

- Questions from the Reading?

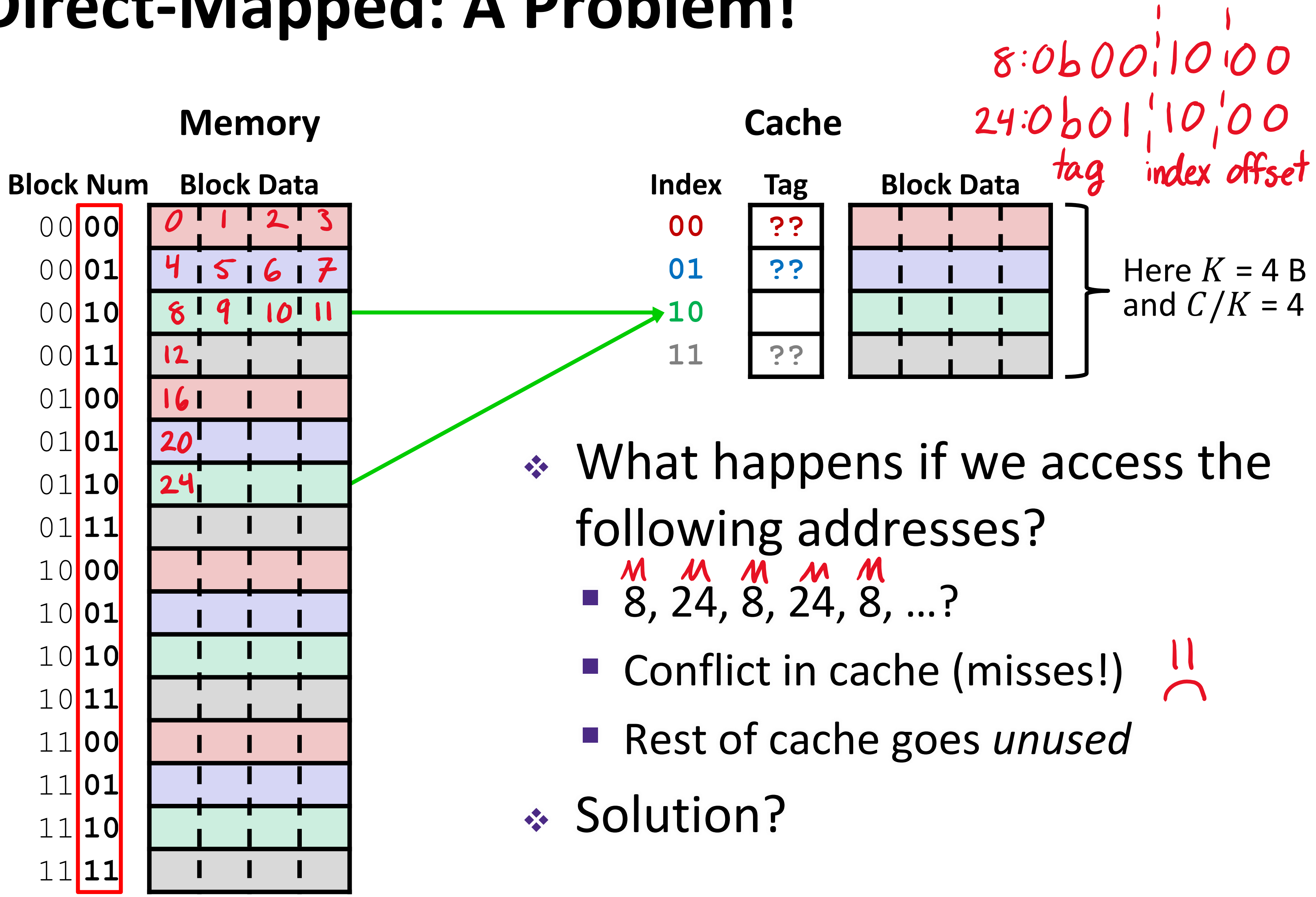
Review: Direct-Mapped Cache



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

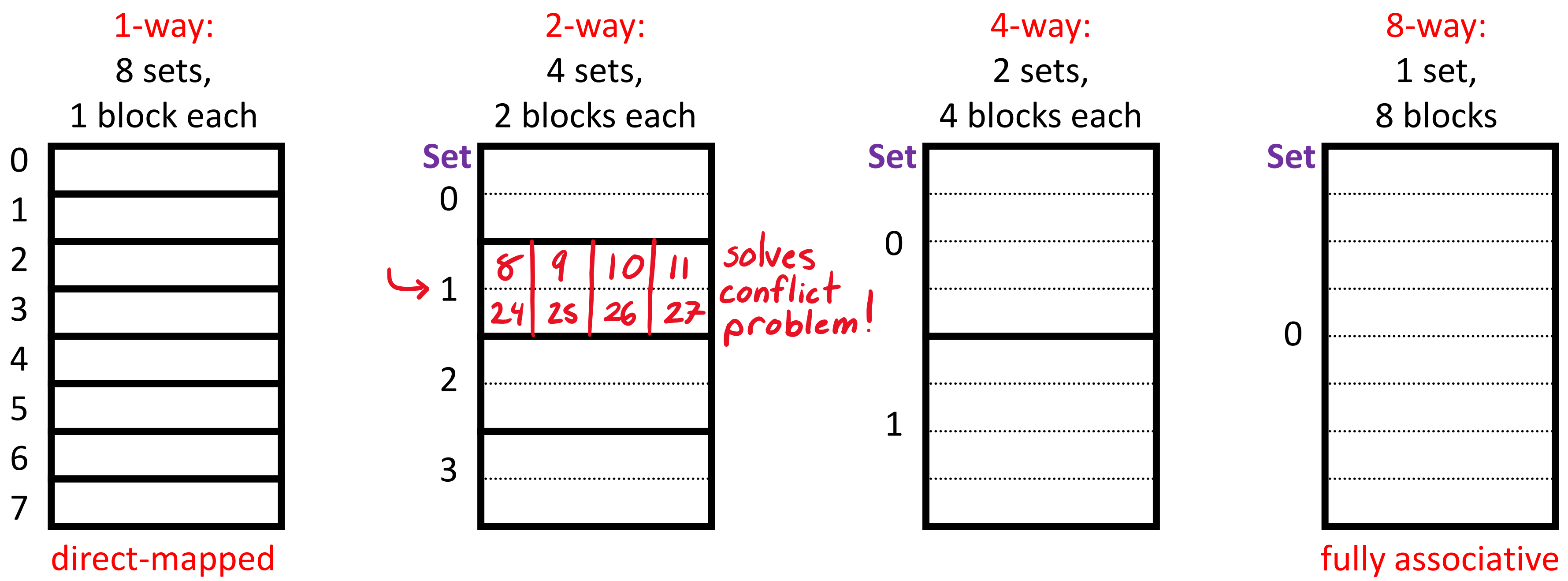
Direct-Mapped: A Problem!



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

Associativity: A Solution!

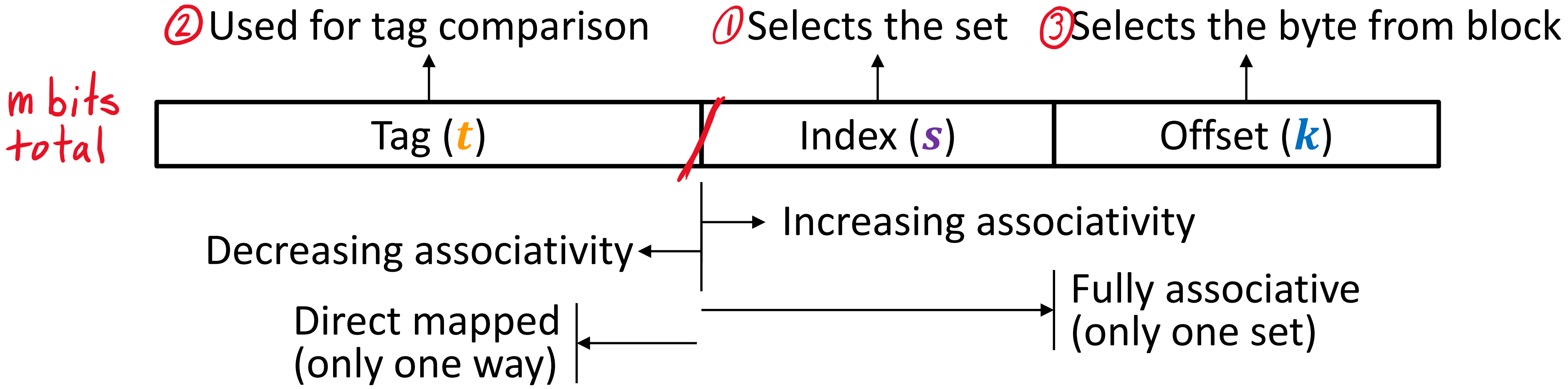
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**



Cache Organization (3)

Note: The textbook uses "b" for offset bits

- Associativity (E): number of ways to store in each set
  - Such a cache is called an "*E-way set associative cache*"
  - We now index into cache *sets*, of which there are $\boxed{S = C/K/E}$
  - Use lowest $\boxed{\log_2(C/K/E) = s}$ 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



② Used for tag comparison
① Selects the set
③ Selects the byte from block
m bits total
Tag (t)
Index (s)
Offset (k)
Increasing associativity
Decreasing associativity
Fully associative (only one set)
Direct mapped (only one way)

Example Placement

block size K:	16 B
Capacity C/K:	8 blocks
Address m:	16 bits

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

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

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

m-bit address:

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

$\log_2(8/1) = 3 \text{ bits}$ $\log_2(8/2) = 2 \text{ bits}$ $\log_2(8/4) = 1 \text{ bit}$

Direct-mapped

$s = ?$ 011

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

2-way set associative

$s = ?$ 11

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

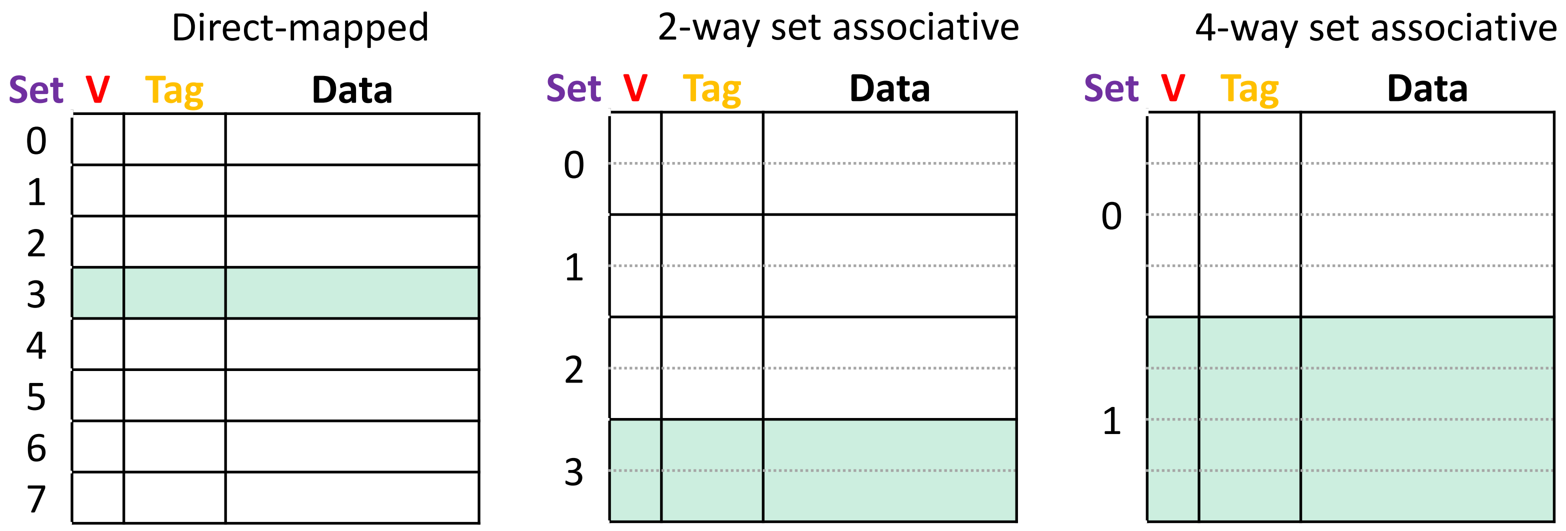
4-way set associative

$s = ?$ 1

Set	Tag	Data
0		
1		✓
		✓
		✓
		✓

Block Placement and Replacement

- *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 (easy!)
  - Caches typically use something close to ***least recently used (LRU)*** (hardware usually implements "*not most recently used*")



Polling Questions

$C = 2^{11}$ B $\quad K = 2^7$ 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?

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

$S = C/K/E$

$E = (C/K)/S$

$= 16/2 = 8$

A. 2

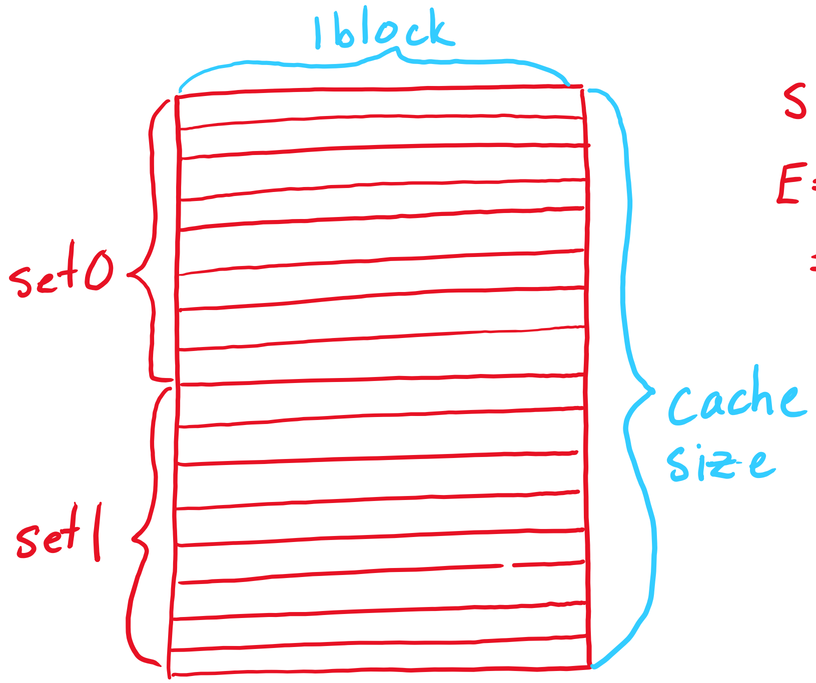
B. 4

[✓ C. 8]

D. 16

E. We're lost...

each set has 8 blocks so $E = 8$

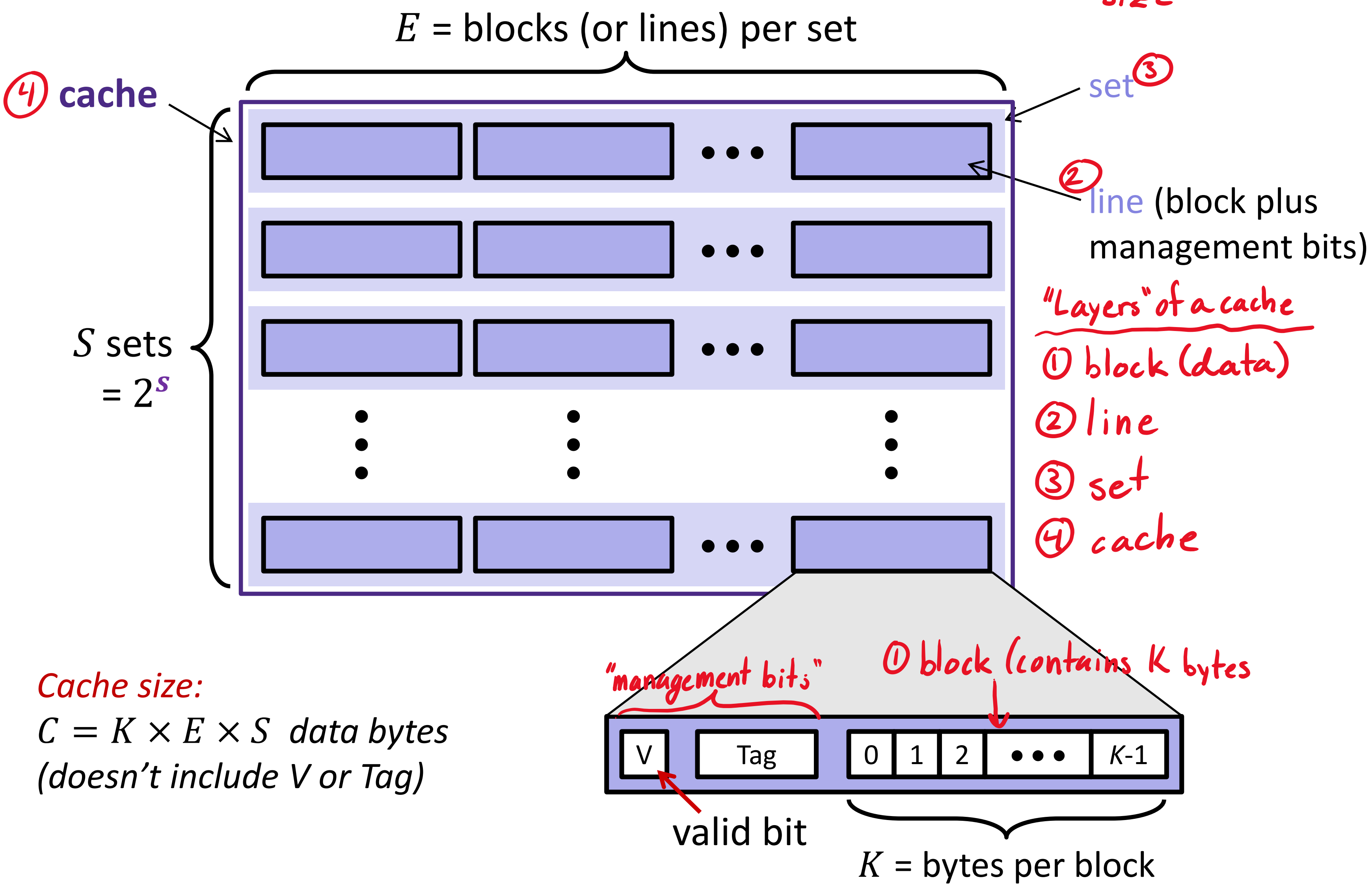


- If addresses are 16 bits wide, how wide is the Tag field? $m=16$

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

General Cache Organization (S, E, K)

sets associativity block size



"Layers" of a cache
① block (data)
② line
③ set
④ cache

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

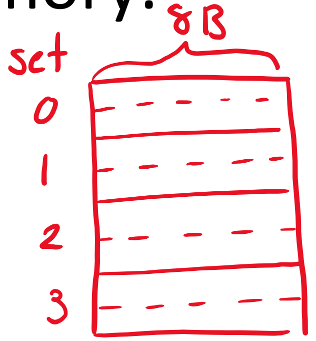
Parameter	Variable	Formulas
Block size	K (B in book)	$M = 2^{\boldsymbol{m}} \leftrightarrow \boldsymbol{m} = \log_2 M$ $S = 2^{\boldsymbol{s}} \leftrightarrow \boldsymbol{s} = \log_2 S$ $K = 2^{\boldsymbol{k}} \leftrightarrow \boldsymbol{k} = \log_2 K$ $C = K \times E \times S$ $\boldsymbol{s} = \log_2(C/K/E)$ $\boldsymbol{m} = \boldsymbol{t} + \boldsymbol{s} + \boldsymbol{k}$
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^{10} B $\Leftrightarrow$ m=10 bits

MP

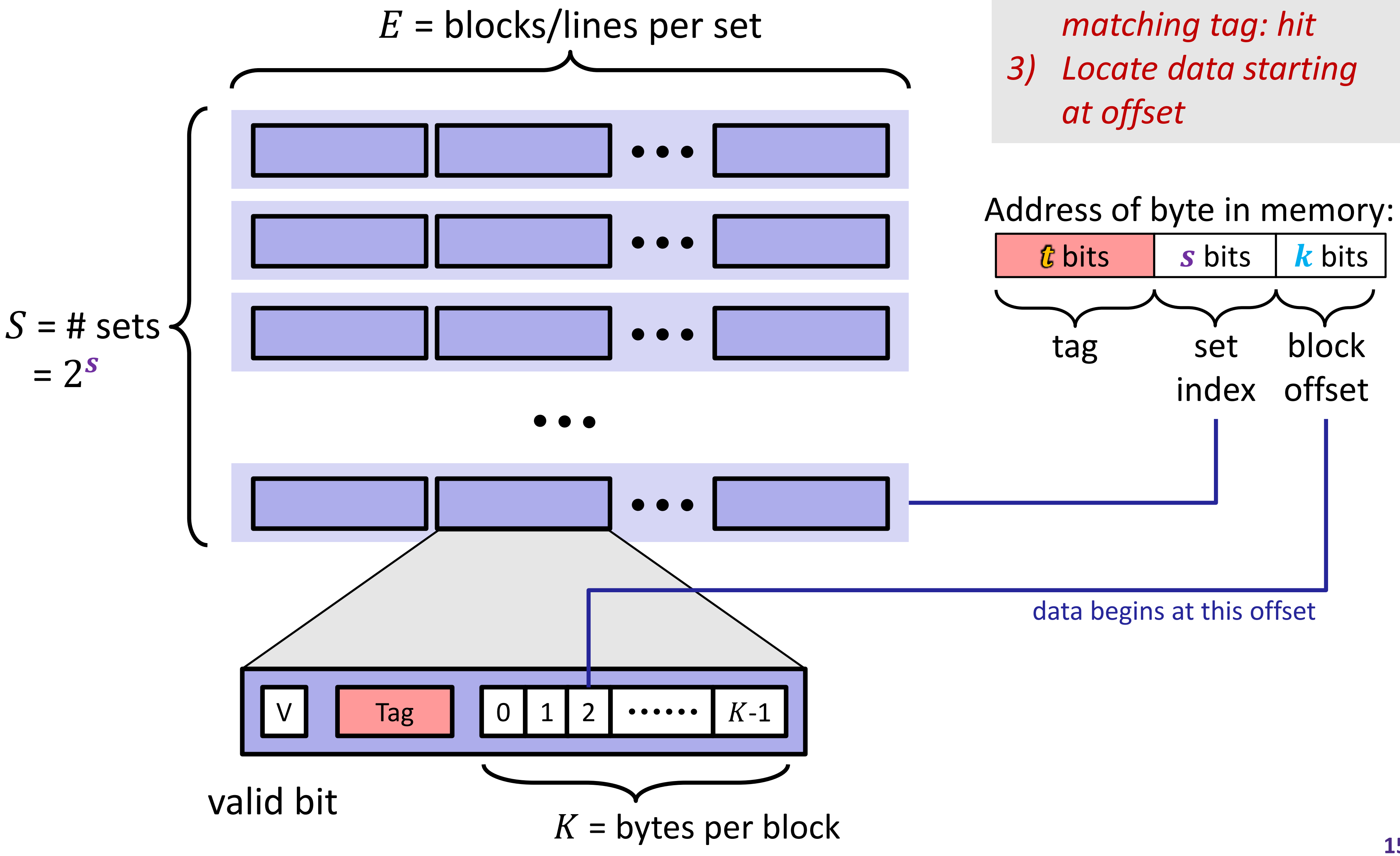
- 1 KiB address space, 125 cycles to go to memory. Fill in the following table:



C	**Cache Size *C***	64 B	2^6
K	**Block Size *K***	8 B	2^3
E	**Associativity *E***	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	

Cache Read

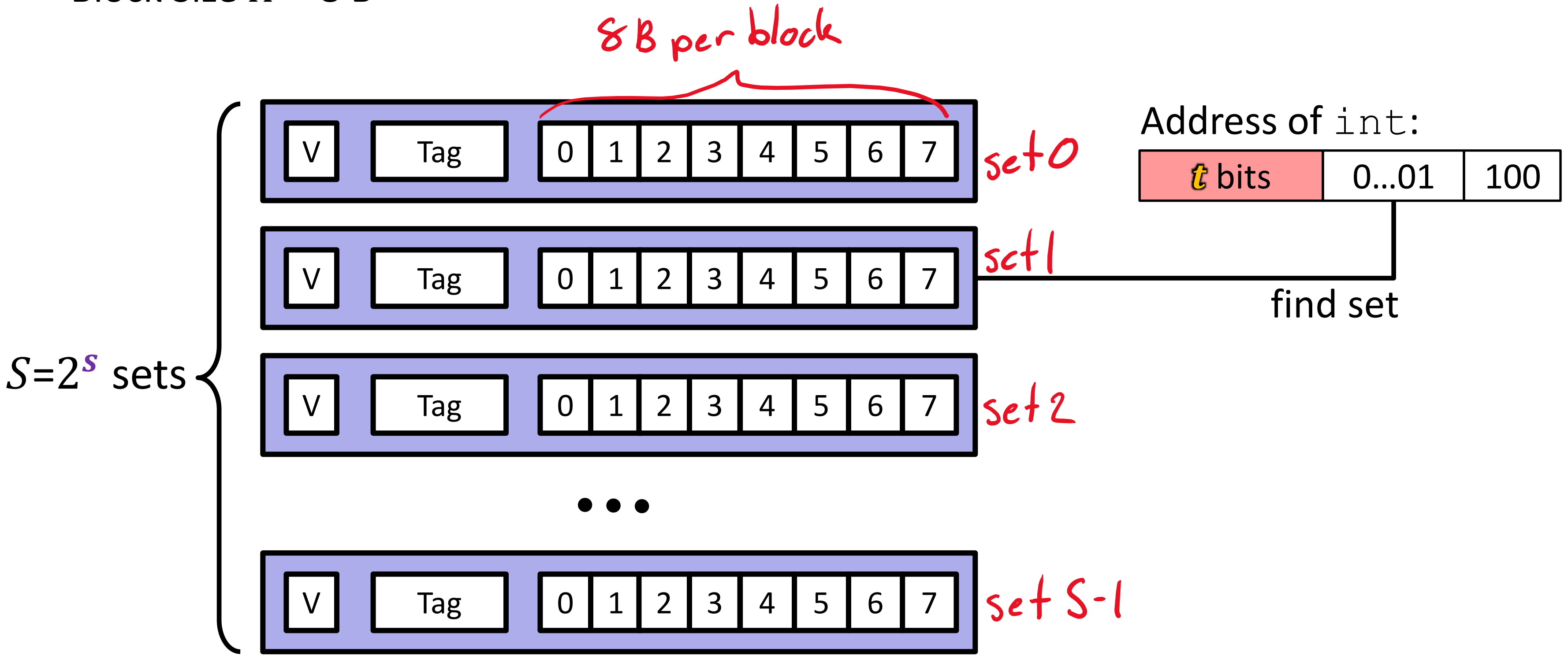
1) Locate set
2) Check if any line in set is valid and has matching tag: hit
3) Locate data starting at offset



Example: Direct-Mapped Cache (E = 1)

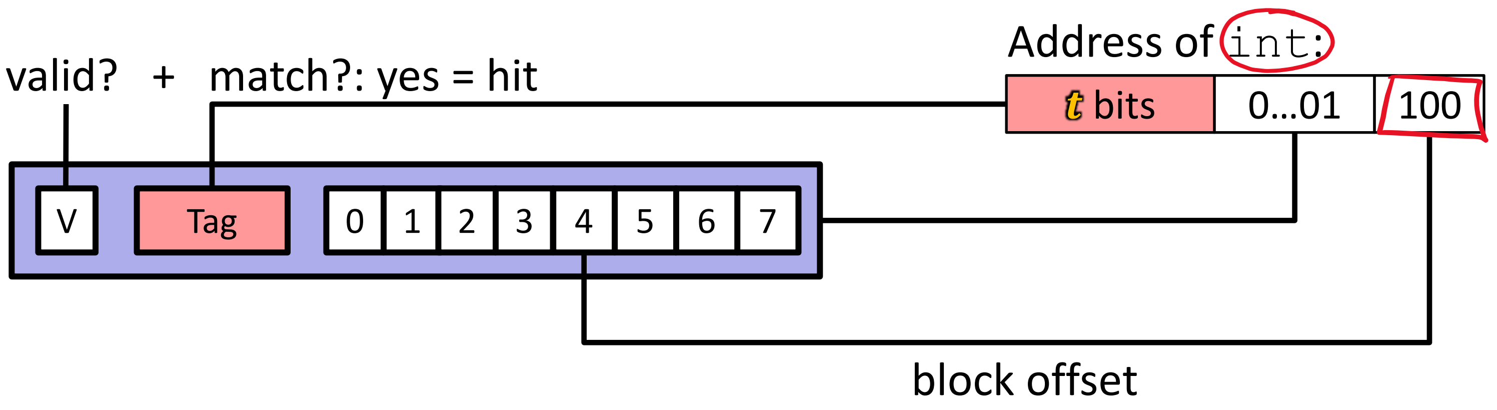
Direct-mapped: One line per set

Block Size K = 8 B



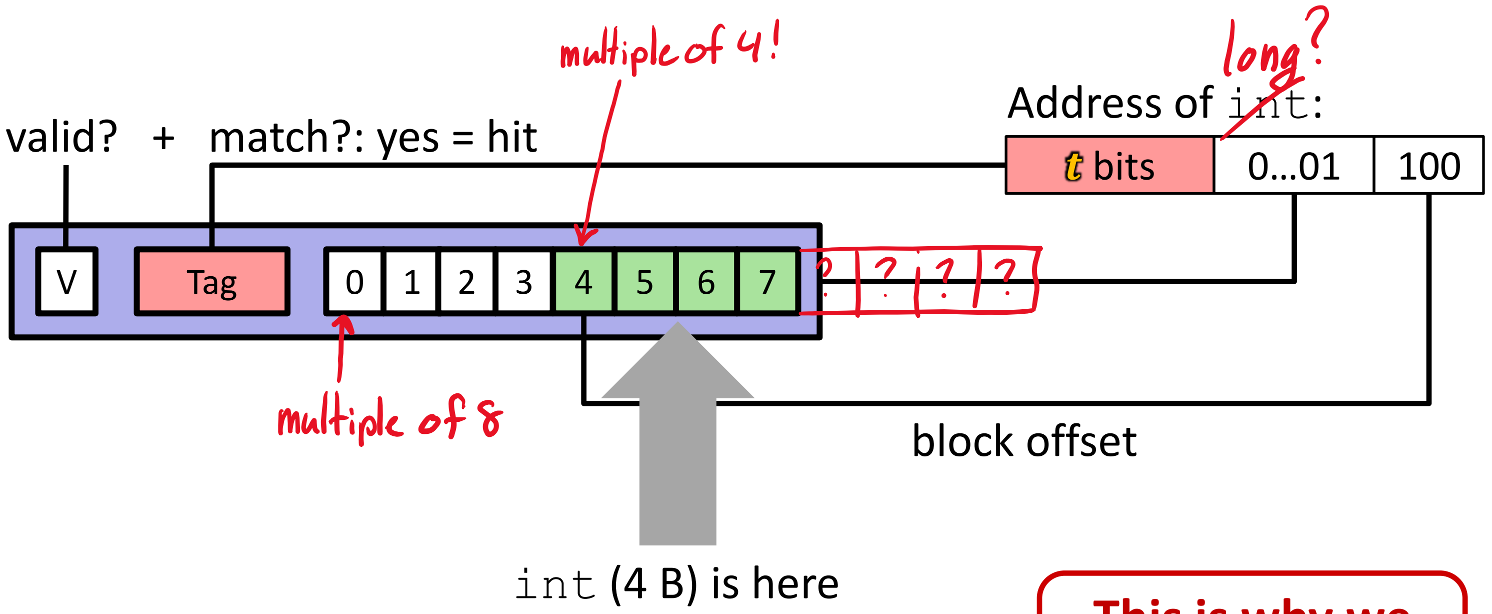
Example: Direct-Mapped Cache (E = 1)

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



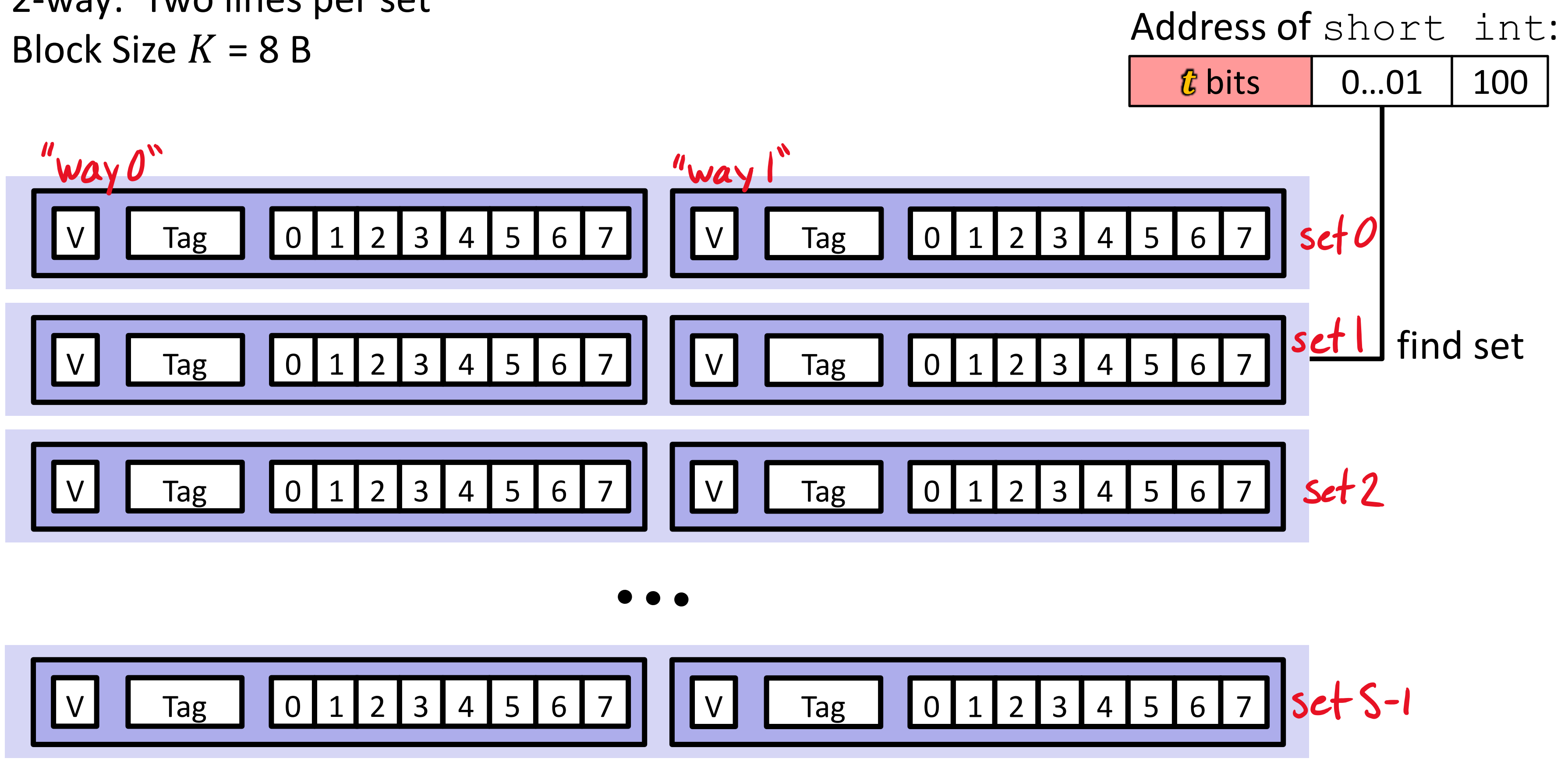
This is why we want alignment!

no unnecessary extra cache accesses across block boundaries

No match? Then old line gets evicted and replaced

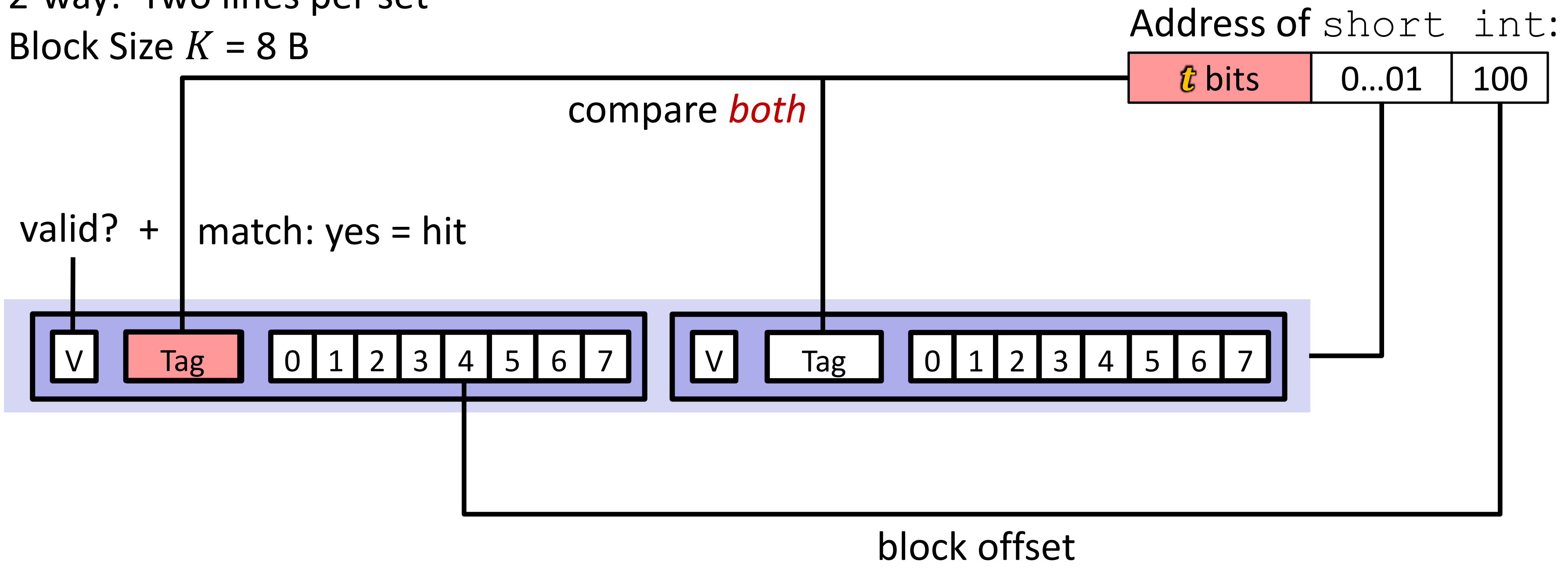
Example: Set-Associative Cache (E = 2)

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



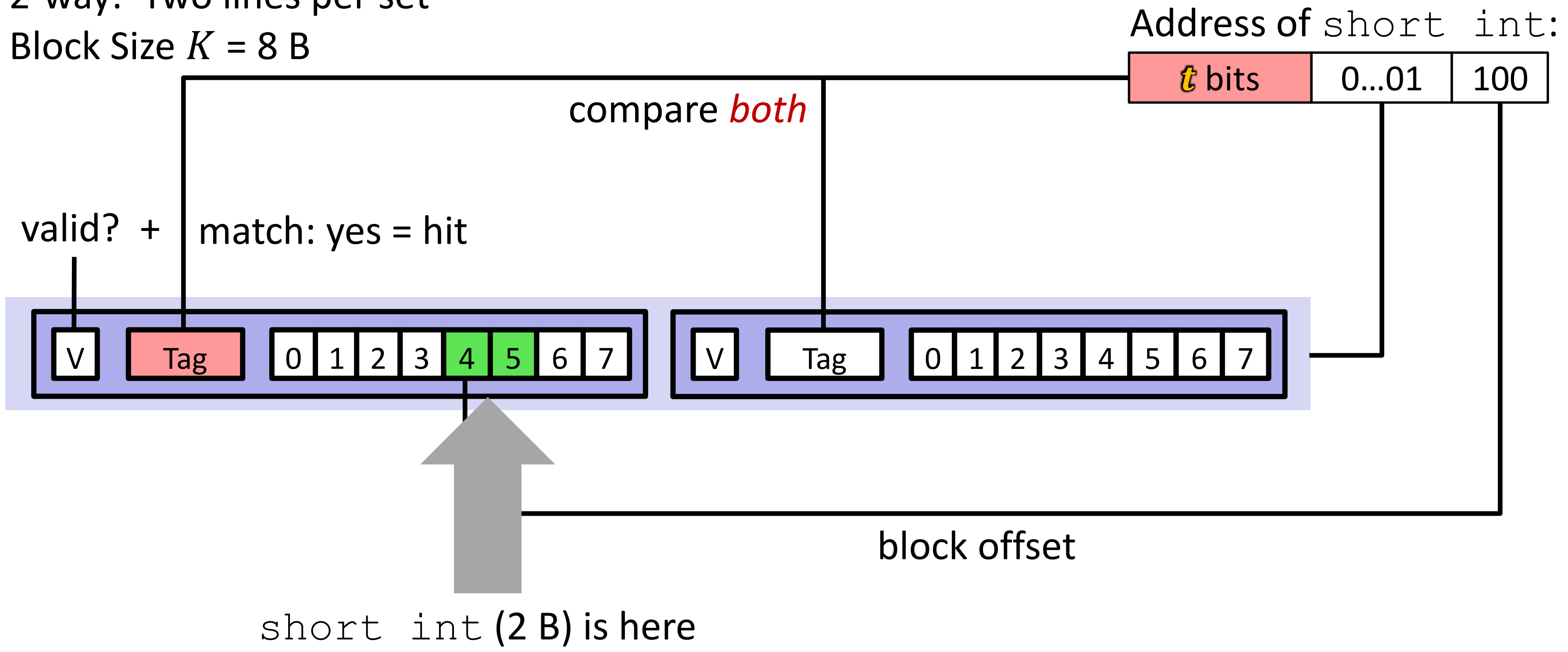
Example: Set-Associative Cache (E = 2)

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 <u>cold</u> (all blocks invalid) and `sum`, `i`, and `j` are stored in registers, calculate the **miss rate**: $\boxed{100\%}$
  - k=3bits, s=2 bits, t=5 bits
  - m = 10 bits, C = 64 B, K = 8 B, E = 2

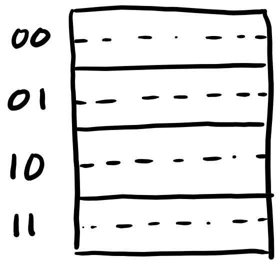
2 bytes per element

```
#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];
```

→ stride 8 → addr + 16 each time

block:

cache:



tag | index | offset

ar[0][0] → 0b 10 000|0 0|000 → M (1st way)

ar[1][0] → 0b 10 000|1 0|000 → M (1st way)

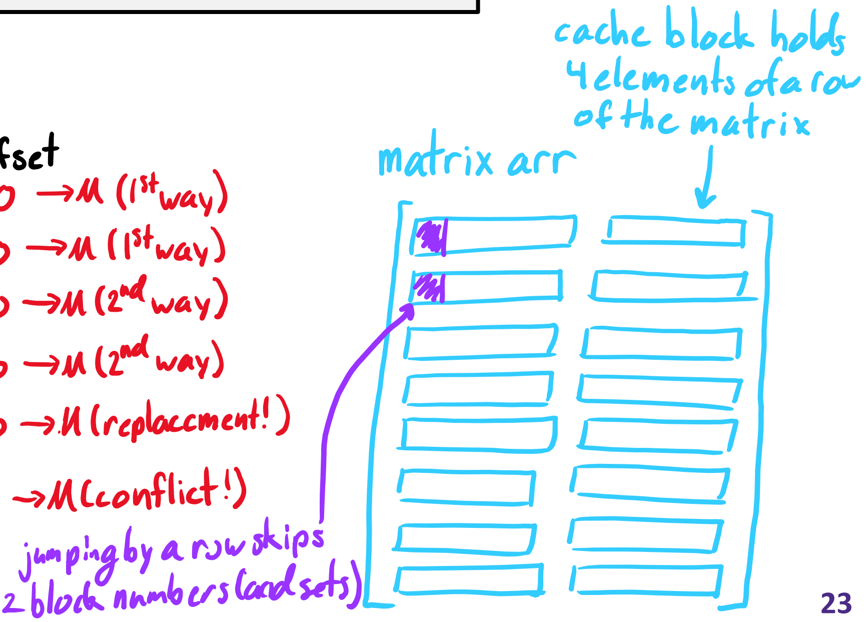
ar[2][0] → 0b 10 001|0 0|000 → M (2nd way)

ar[3][0] → 0b 10 001|1 0|000 → M (2nd way)

ar[4][0] → 0b 10 010|0 0|000 → M (replacement!)

⋮

ar[0][1] → 0b 10 000|0 00|10 → M (conflict!)


cache block holds 4 elements of a row of the matrix
matrix arr
jumping by a row skips 2 block numbers (and sets)