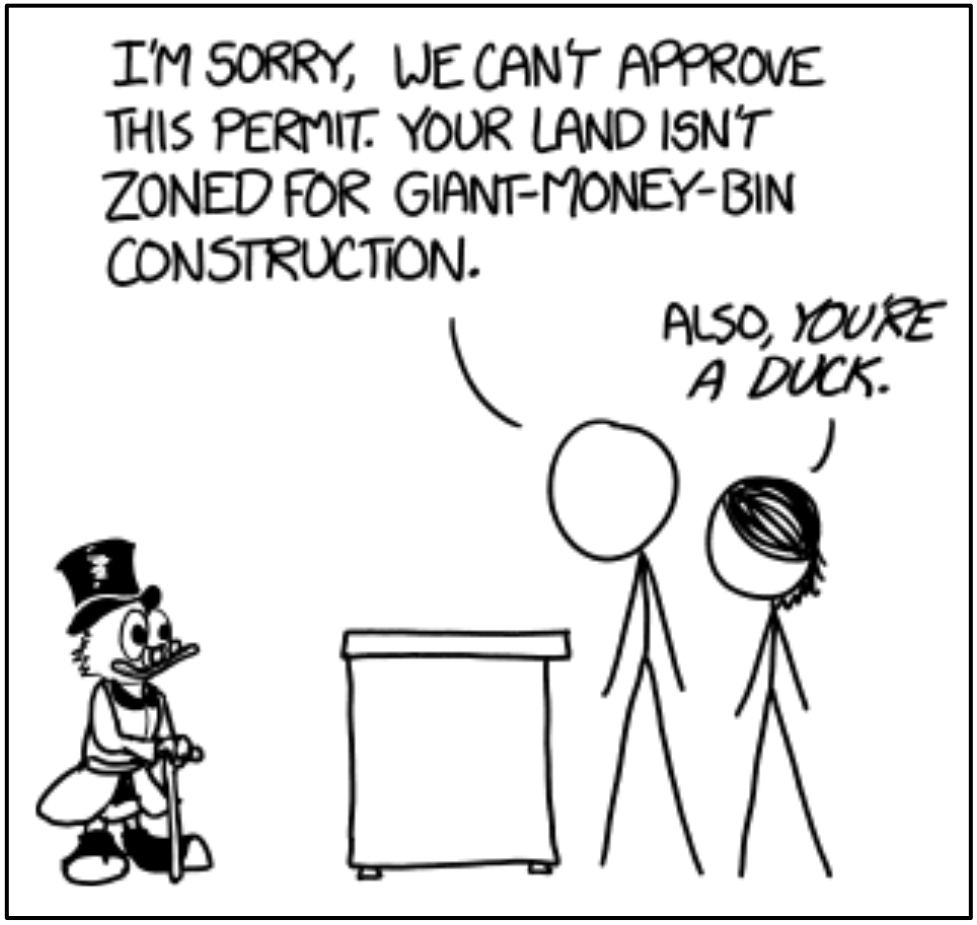
Caches III

CSE 351 Summer 2021

Closer to Fine (Indigo Girls, 1989)

I went to see the doctor of philosophy
With a poster of Rasputin and a beard down to his knee
He never did marry or see a B-grade movie
He graded my performance, he said he could see through me
I spent four years prostrate to the higher mind
Got my paper and I was free



Gentle, Loving Reminders

- hw15 due Monday (8/2)
- Lab 3 due Tonight (7/30)
  - You get to write some buffer overflow exploits!
- Lab 4 released later today
  - All about caches!
- Unit Summary 2 Due Monday! (8/2)
  - Let me know if you'd like to chat!

Learning Objectives

Understanding this lecture means you can:

- Explain associativity to someone that just learned about caches
- Compute cache parameters
- Name the 3 types of cache misses
- Determine the miss rate for associative caches
- Explain how computing uses averages and give a few examples of harm caused

Making memory accesses fast!

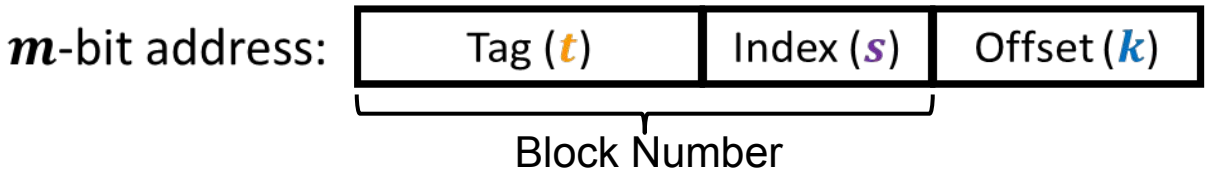
- Cache basics
- Principle of locality
- Memory hierarchies
- Cache organization
  - Direct-mapped (*sets*; index + tag)
  - **Associativity (*ways*)**
  - **Replacement policy**
  - Handling writes
- Program optimizations that consider caches

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:



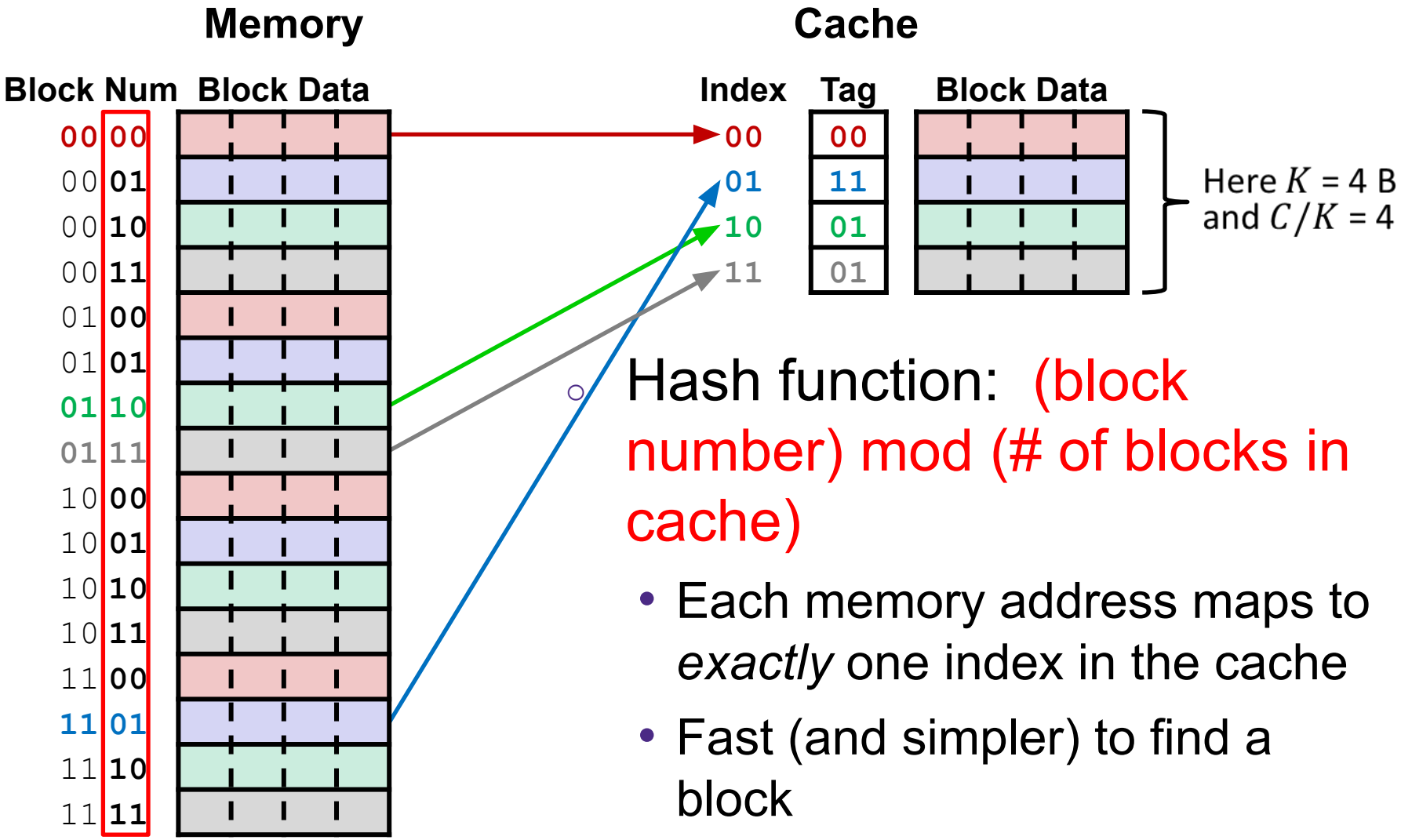
- **Index (s)** field tells you where to look in cache
  - Number of bits is determined by number of sets ($\log_2(C/K/E)$)
  - Need enough bits to reference every set in the cache
- **Tag (t)** field lets you check that data is the block you want
  - Rest of the bits not used for index and offset ($m - s - k$)
- **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**
  - Data is returned to CPU starting at byte offset
- If block is not in the cache, it's a **cache miss**
  - 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

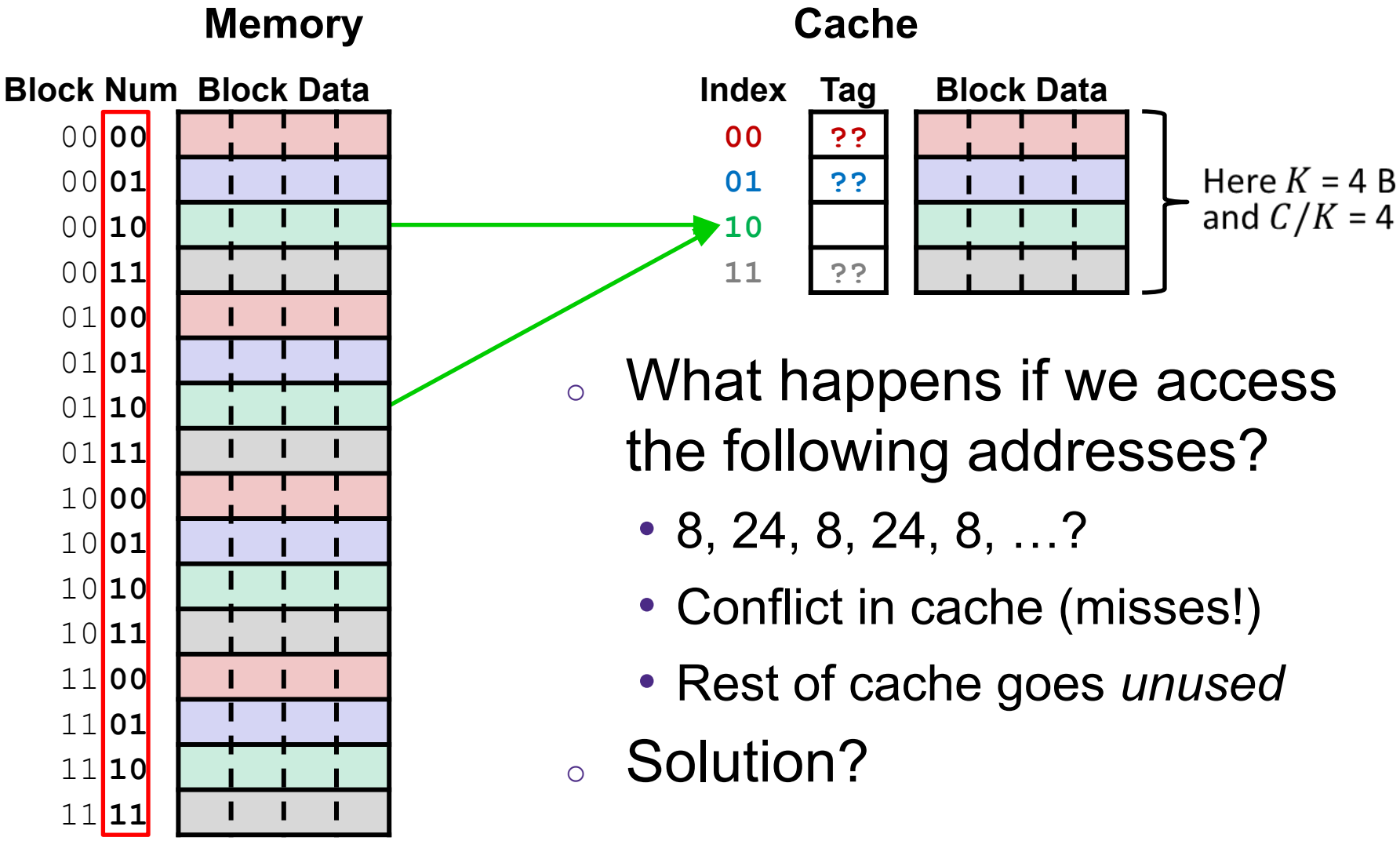
Review: Direct-Mapped Cache



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

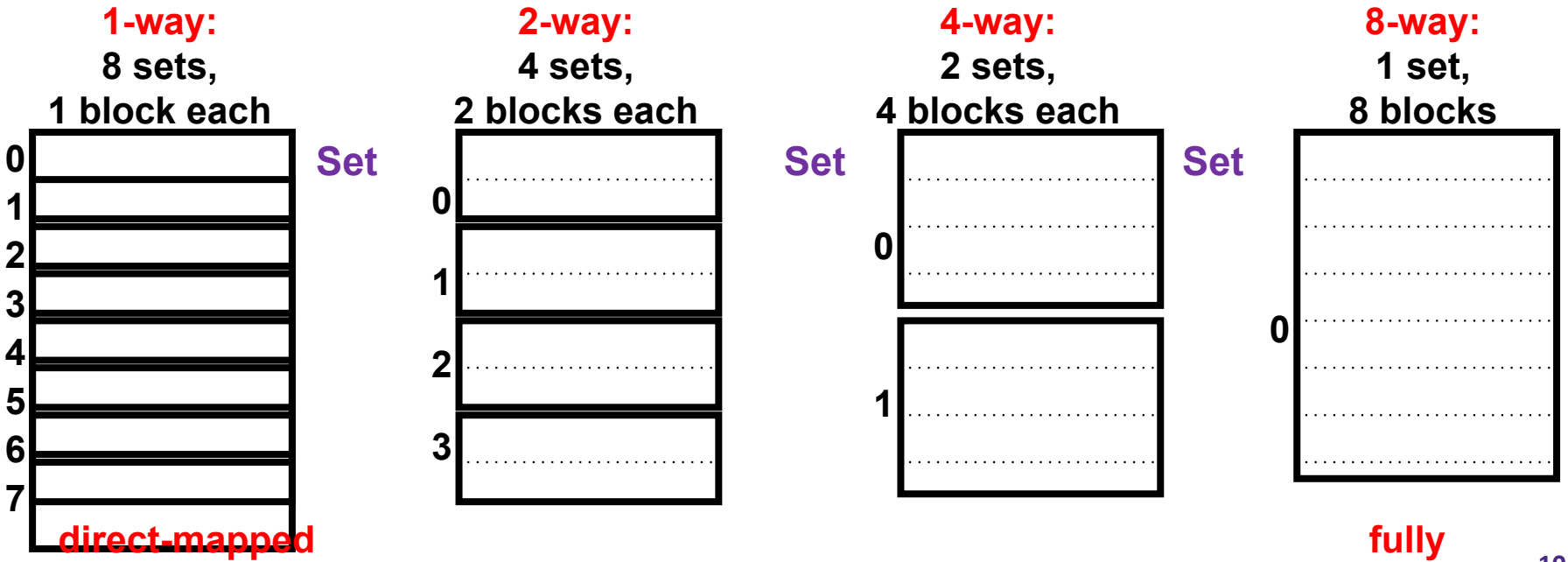
Direct-Mapped Cache Problem



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

Associativity

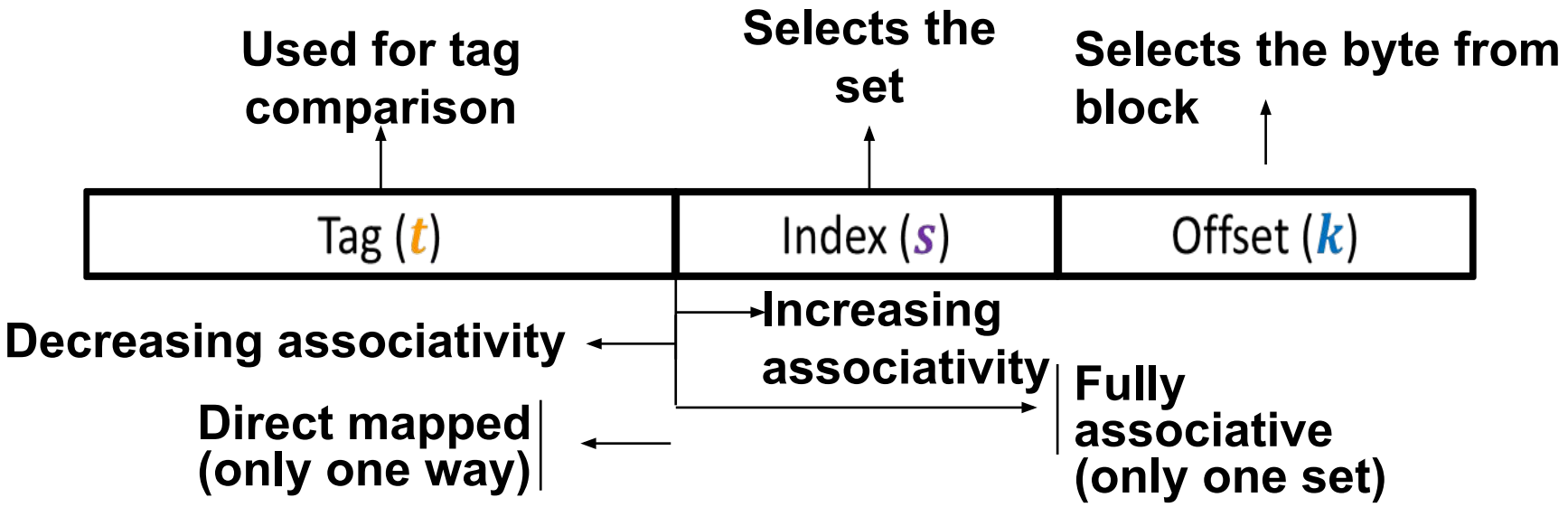
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**



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 $S = C/K/E$
  - 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



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
  - Typically use something close to ***least recently used (LRU)***

Direct-mapped

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

2-way set associative

Set	Tag	Data
0		
1		
2		
3		

4-way set associative

Set	Tag	Data
0		
1		

Checking in!

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

 2

 4

 8

 16

 Help!

Checking in!

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

  🐶 2

  🐈 4

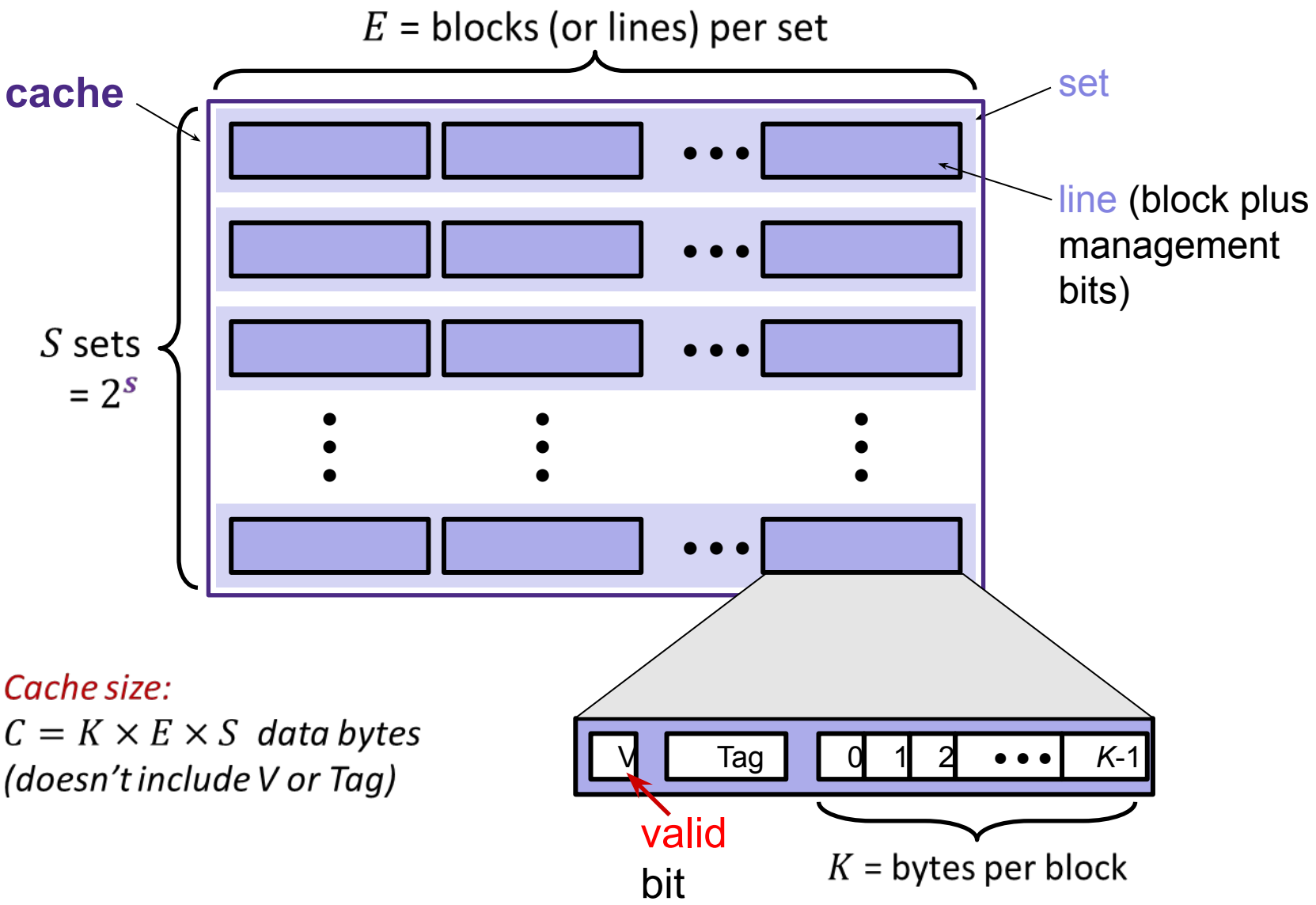
  🐑 8

  🦄 16

  🥶 Help!

- 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)

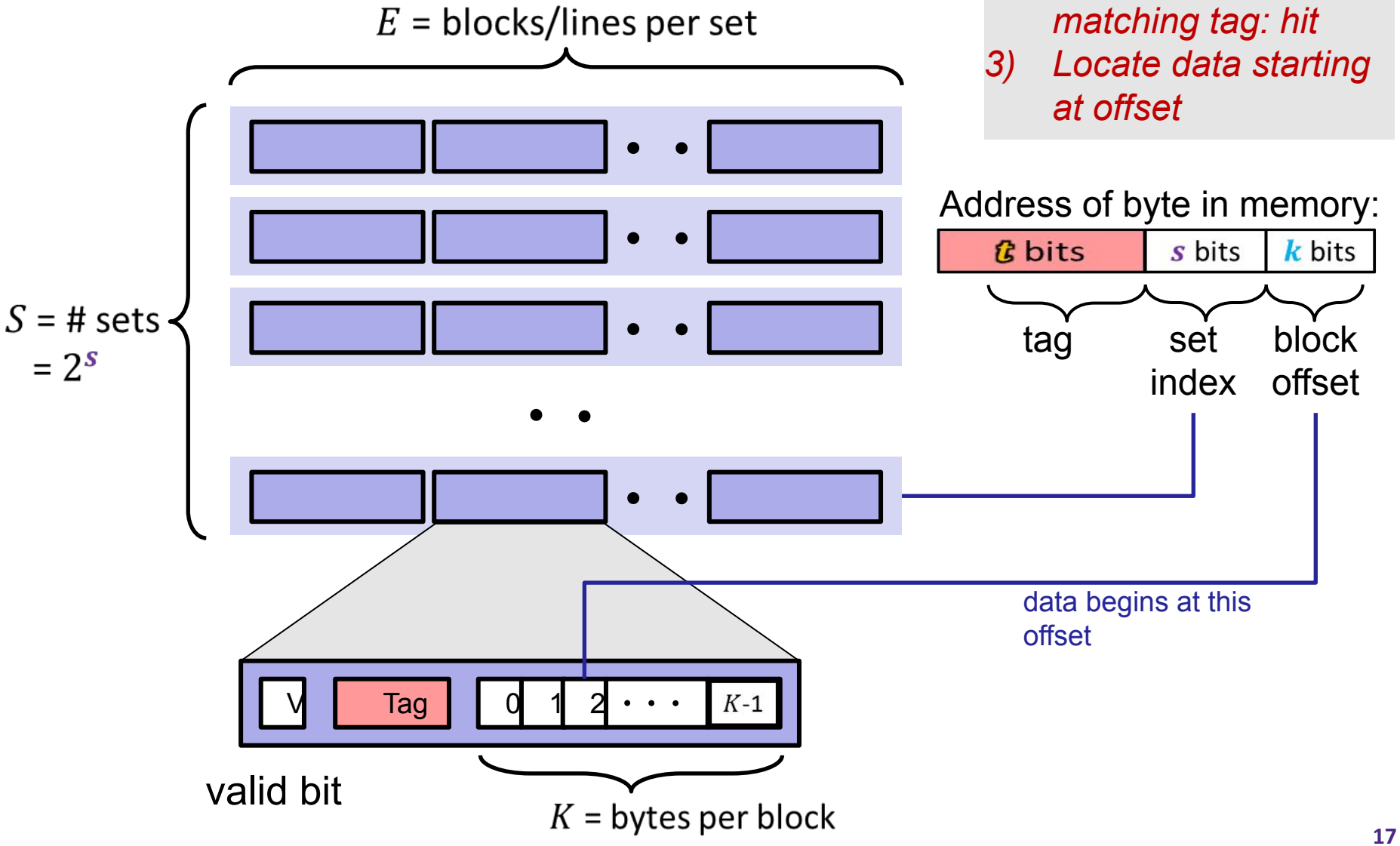
Notation Review

- 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

Parameter	Variable	Formulas
Block size	*K*	*Given*
Cache size	*C (capacity)*	*Given*
Associativity	*E*	*C/K/S*
Number of Sets	*S*	*C/K/E*
Address space	*M*	*Given*
Address width	*m*	$log_2(M)$
Tag field width	*t*	*m-s-k*
Index field width	*s*	$log_2(S)$
Offset field width	*k*	$log_2(K)$

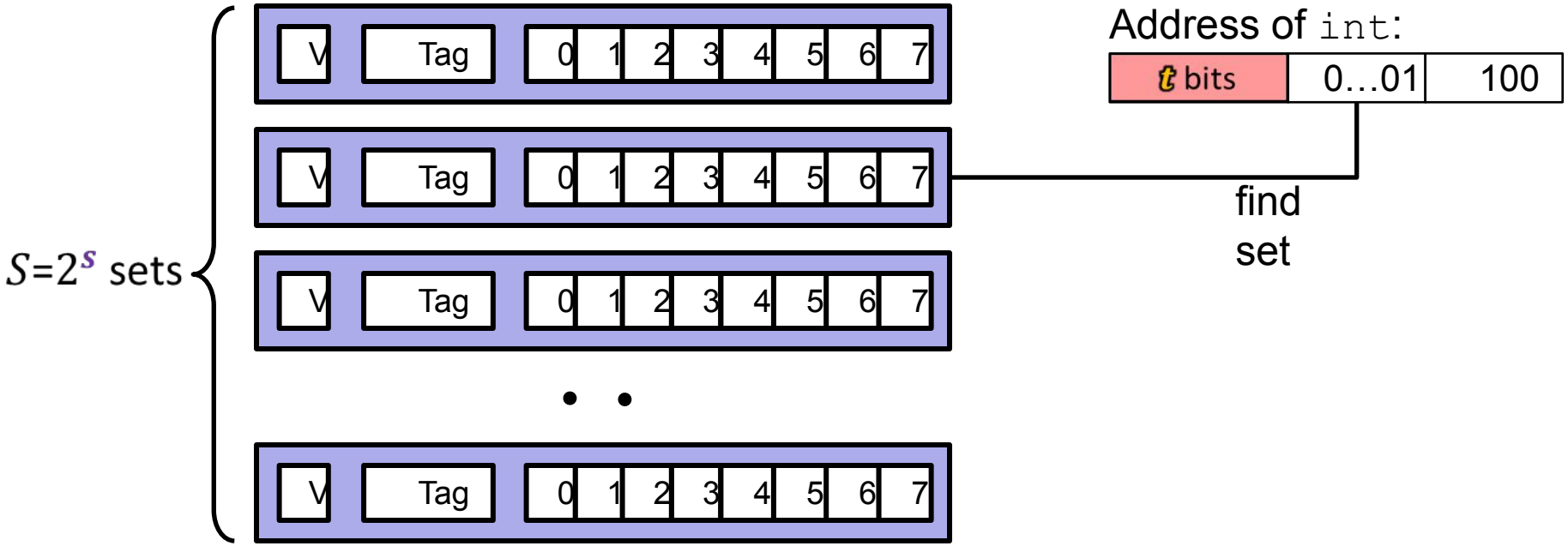
Cache Read

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



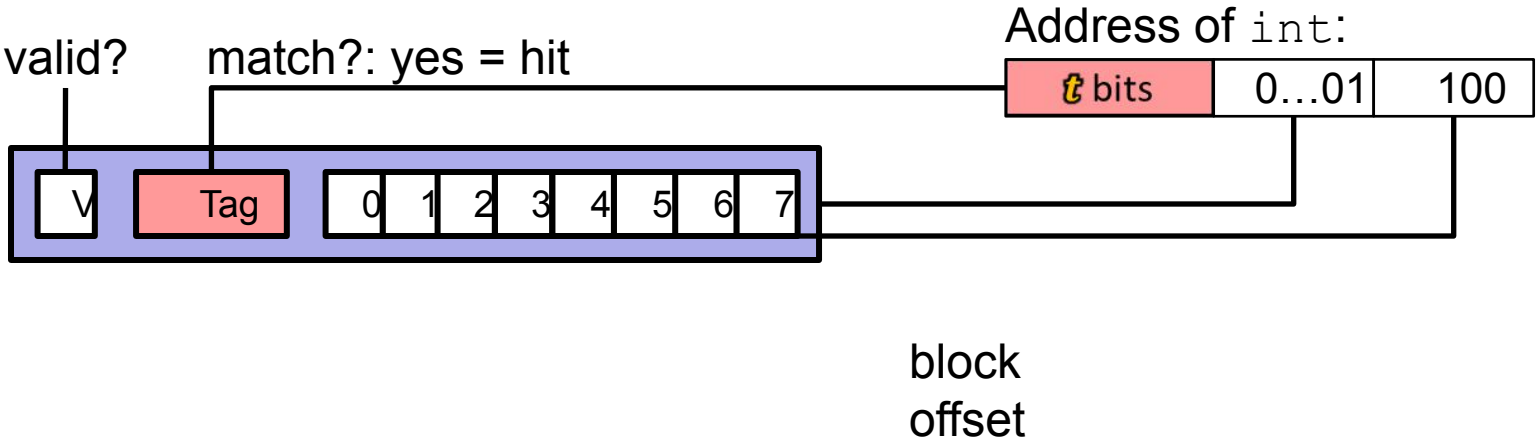
Example: Direct-Mapped Cache (E = 1)

Here K = 4 B
and C/K = 4



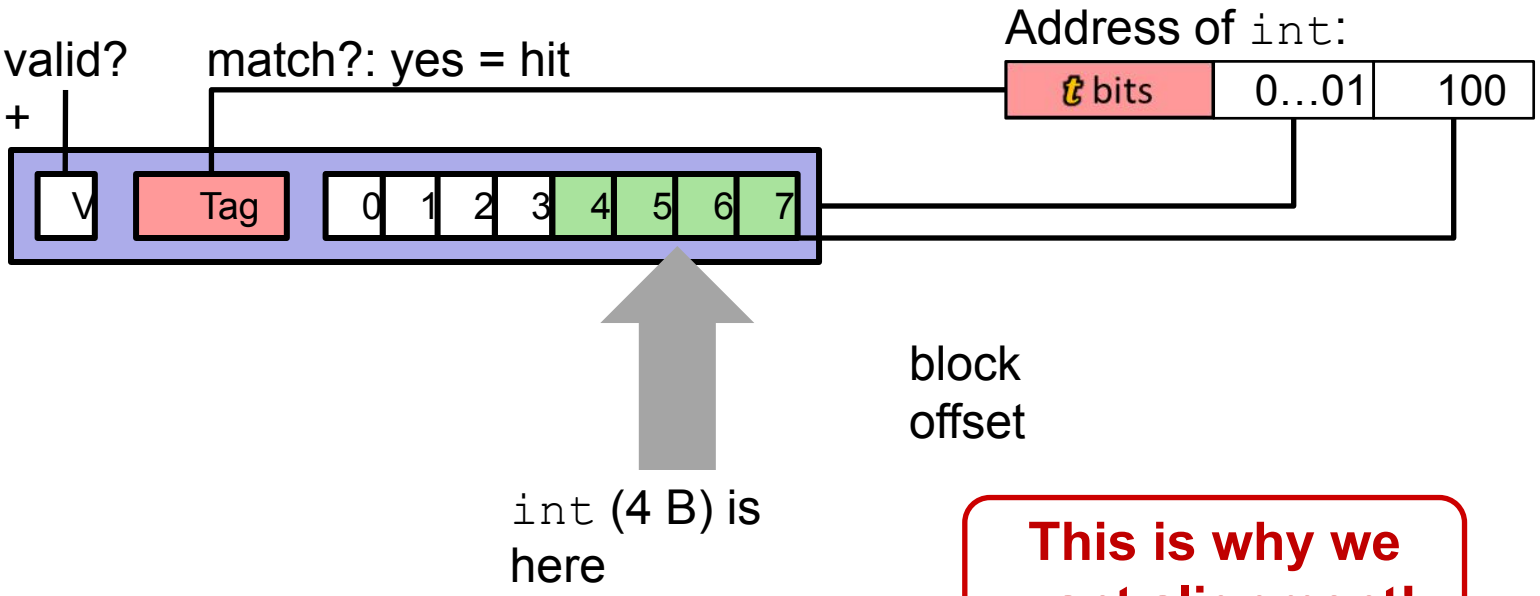
Example: Direct-Mapped Cache (E = 1)

Here K = 4 B
and C/K = 4



Example: Direct-Mapped Cache (E = 1)

Here K = 4 B
and C/K = 4



This is why we want alignment!

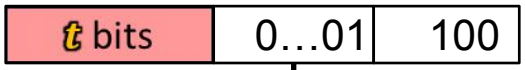
No match? Then old line gets evicted and replaced

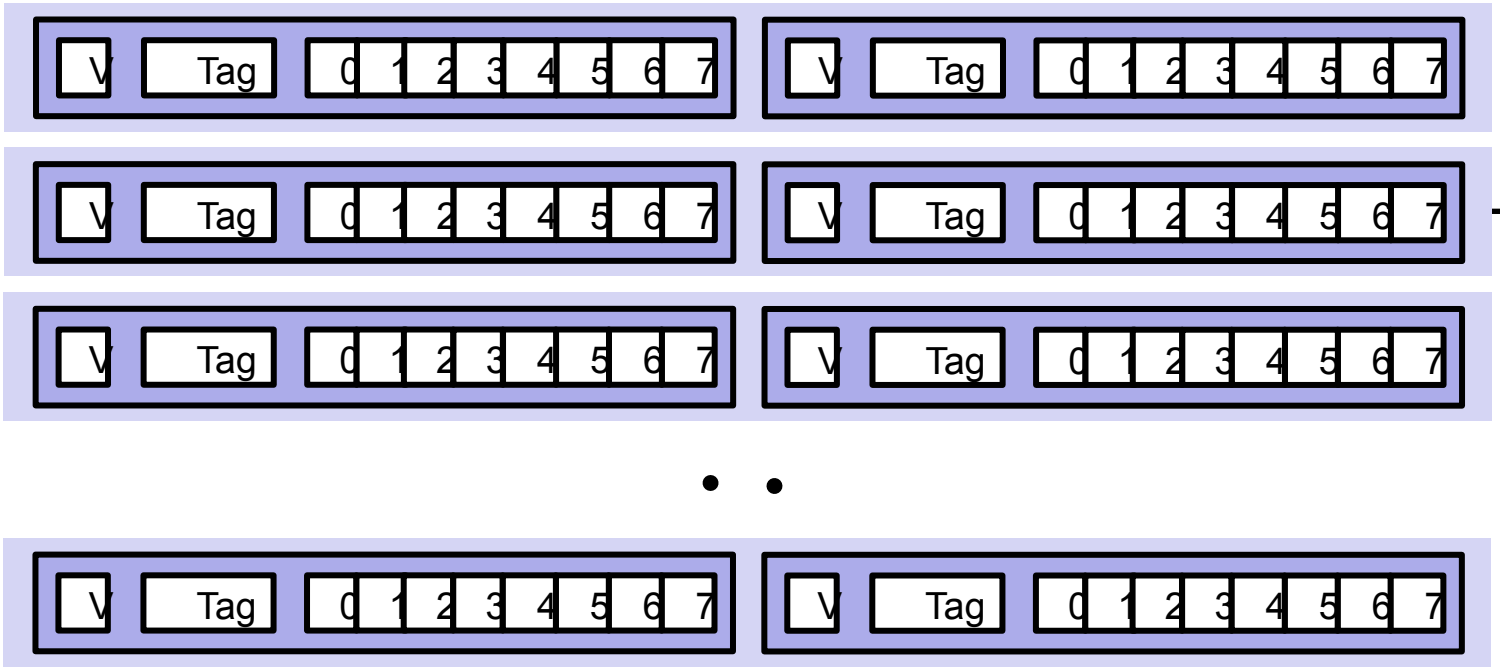
Checking in: Direct Mapped Caches?

Example: Set-Associative Cache (E = 2)

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

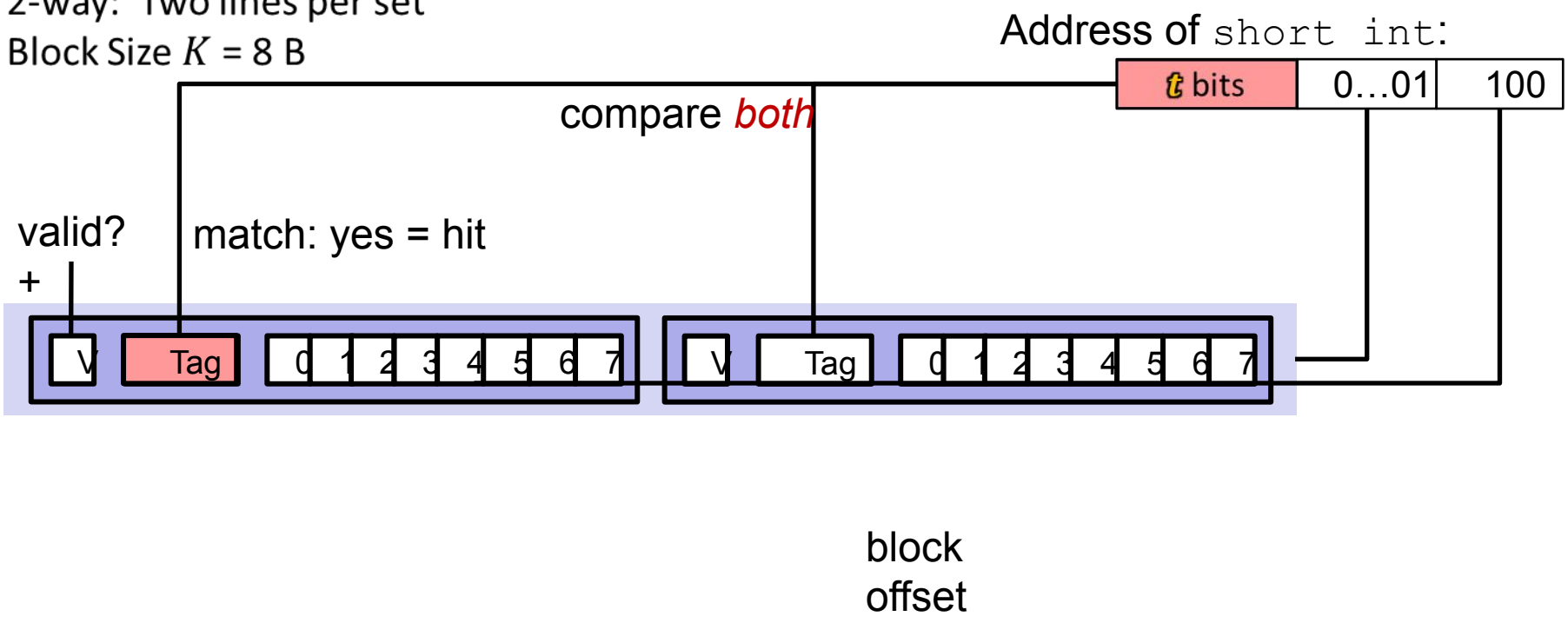
Address of `short int`:



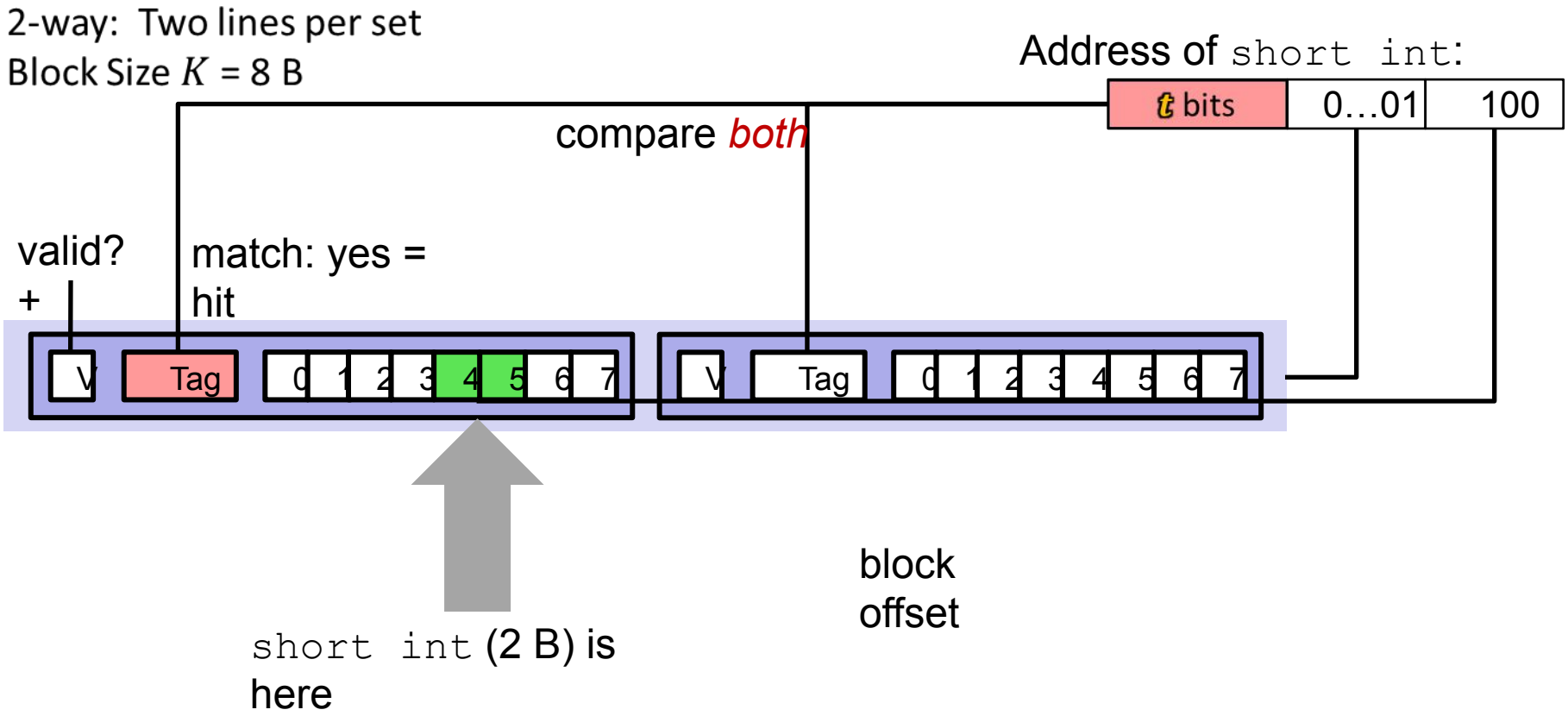


Example: Set-Associative Cache (E = 2)

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



Example: Set-Associative Cache (E = 2)



No match?

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

Checking in: Associative Caches!

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`, `i`, and `j` are stored in registers, calculate the **miss rate**:
  - m = 12 bits, C = 256 B, K = 32 B, E = 2

```
#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];
```

 6.25%

 12.5%

 25%

 33%

 Help!

End of caches for today!

- Direct Mapped, Associative Caches
  - Reads, hits, alignment
- 3 Types of Cache misses, Code Analysis
  - Both super helpful for Lab 4!

- Any questions?

Computing & Averages

Cache Metrics

- Two things hurt the performance of a cache:
  - Miss rate and miss penalty
- *Average Memory Access Time* (AMAT): average time to access memory considering both hits and misses

  AMAT = Hit time + Miss rate × Miss penalty

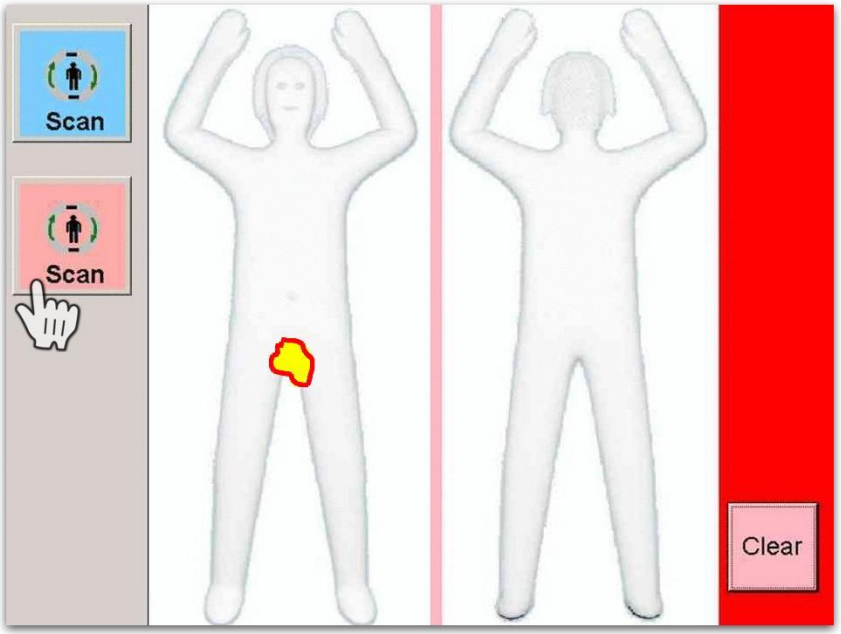
  (abbreviated AMAT = HT + MR × MP)

- 99% hit rate twice as good as 97% hit rate!
  - Assume HT of 1 clock cycle and MP of 100 clock cycles
  - 97%: AMAT = 1 + .03(100) = 4
  - 99%: AMAT = 1 + .01(100) = 2

I want to talk about averages in computing

Storytime!

"I have to scan the way that you're presenting"



Best part: I knew this would happen!

- Lots of other trans folks went through it
  - “Scanning based on presentation” is the guideline
  - Largely a result of trans* advocacy, backlash

Best part: I knew this would happen!

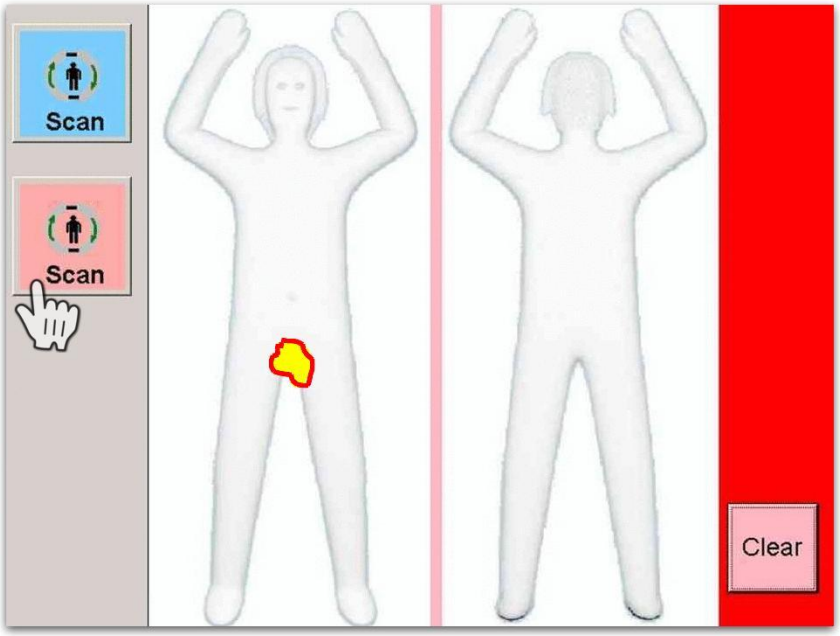
- Lots of other trans folks went through it
  - "Scanning based on presentation" is the guideline
  - Largely a result of trans* advocacy, backlash

- I'll take this over being detained!
  - But goodness, this still hurts

What's going on here?

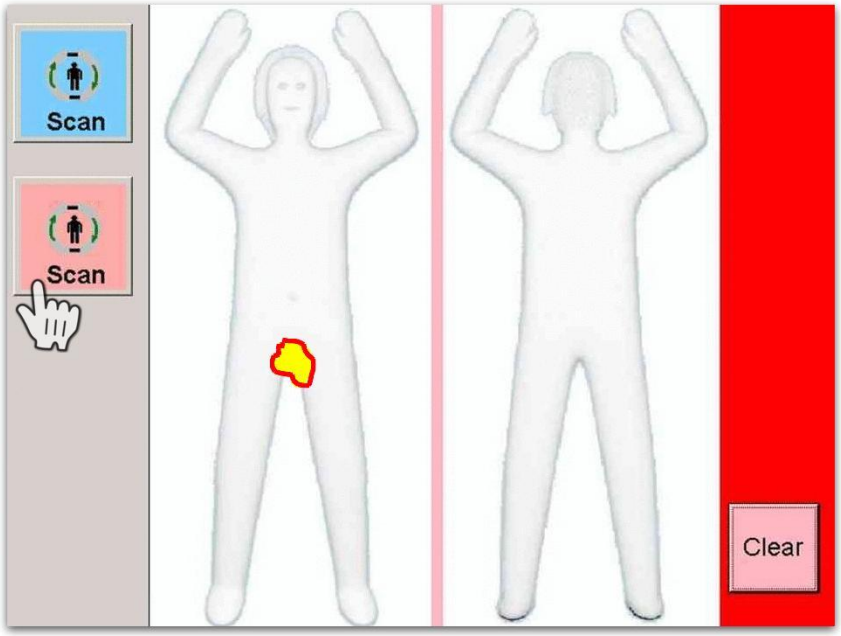
TSA Body Scanners

- Machine Learning!
  - TSA scanned lots of bodies
  - Some predictable variation among “average” bodies
  - Build 2 models, one for “men”, one for “women”
- Agents clock travelers, press a button, and clear alarms as necessary



TSA Body Scanners

- I happened to present more femme that day
  - i.e. bras
- I got "pink button'd"
  - My body doesn't match
- The backs of some TSA hands got *very* intimate



Optimizing for the common case

Common Case Optimizations

- This gets thrown around a lot in algorithms:
  - i.e. worse case perf, average case perf in Big-O

- This is why caches work!
  - The common case is an average program that exhibits locality, assuming that allows for optimizations
- Caches make assumptions about programs in an effort to optimize for “most programs”

Optimization

- Optimizing for the common case is a *classic* CS technique!
  - *One might even say "foundational"*

- Generally, bigger performance impact with common case than edge case optimizations
  - This assumes that performance is most valued…
  - But, generally CS would say to make the common case, as performant as possible, along whatever metric, even at the expense of edge-case performance

Back to our story

ML, CS, and Me

- My body isn't "common case"
  - I'm painfully aware, usually daily, yay dysphoria
- The TSA followed CS values!
  - Really, folks building scanners followed CS values
  - TSA awarded contracts despite hurting trans folks
- "Optimizing the common case at the expense of edge case performance"
  - Really, no one considered trans folks
  - Or, apparently, 1% consistently flagging is fine, **sigh**

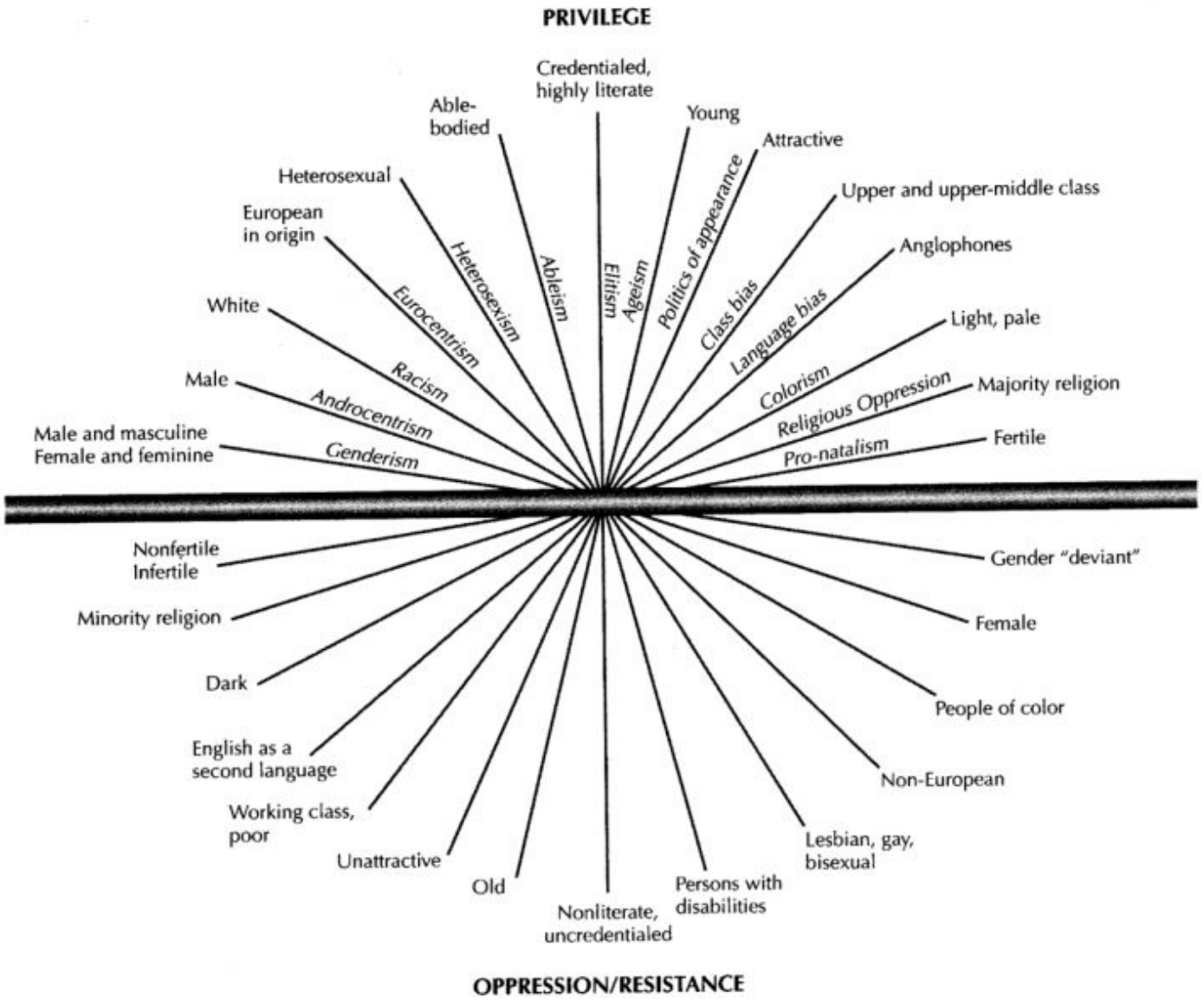
CS interfacing with people

Human Complexity

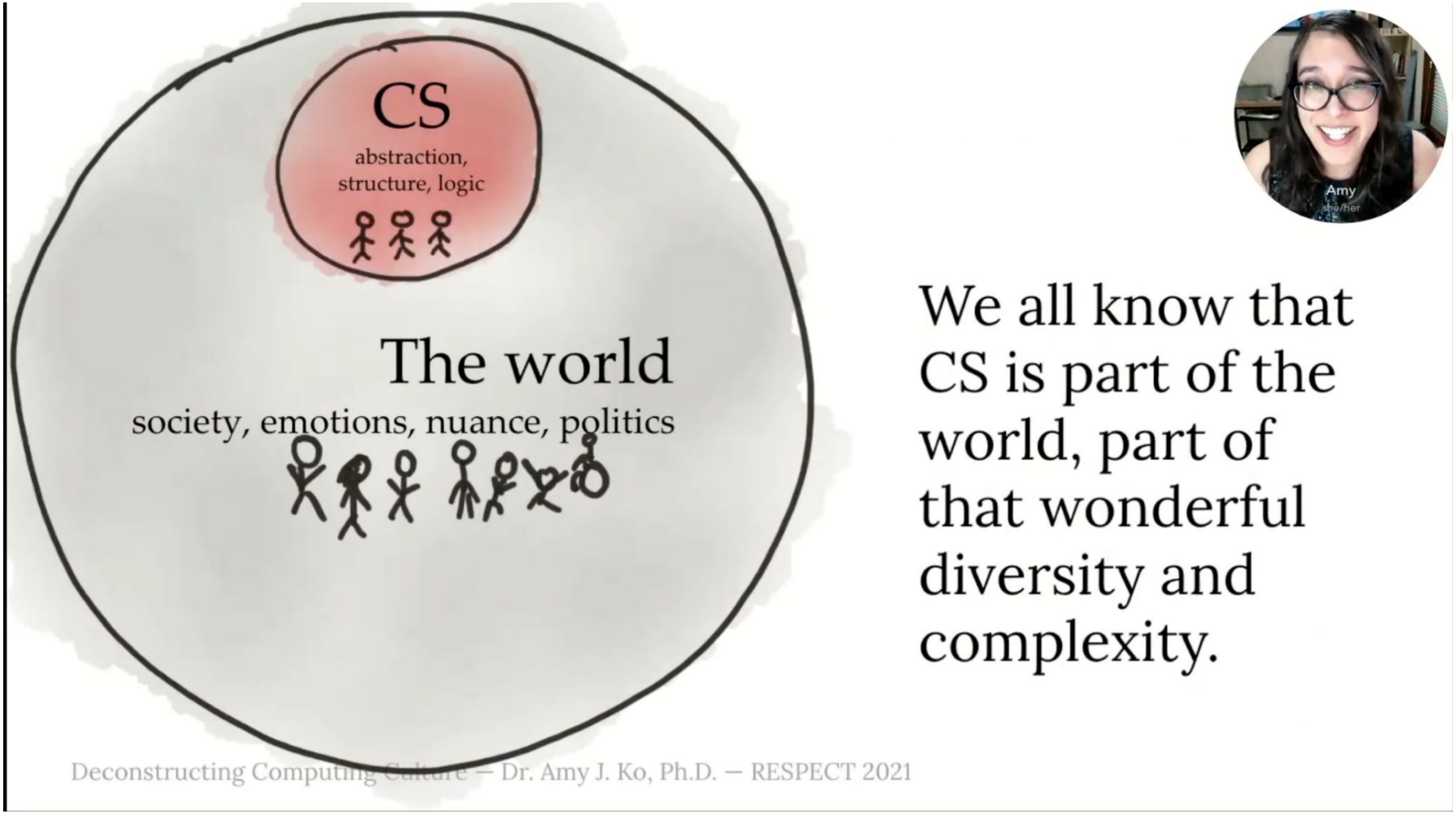
- People are incredibly complex, and diverse!
  - 5% identify as LGBT (Gallup, doesn't include Q!)
  - 16% among Gen Z, y'all are *killing it*
  - **80%** have some kind of non-normative sexuality/gender/relationship!
    - Everyone that doesn't fit gender/sexuality boxes, non-normative sexual preferences (e.g. kink)

- The intersection of normativity doesn't leave space for anyone!

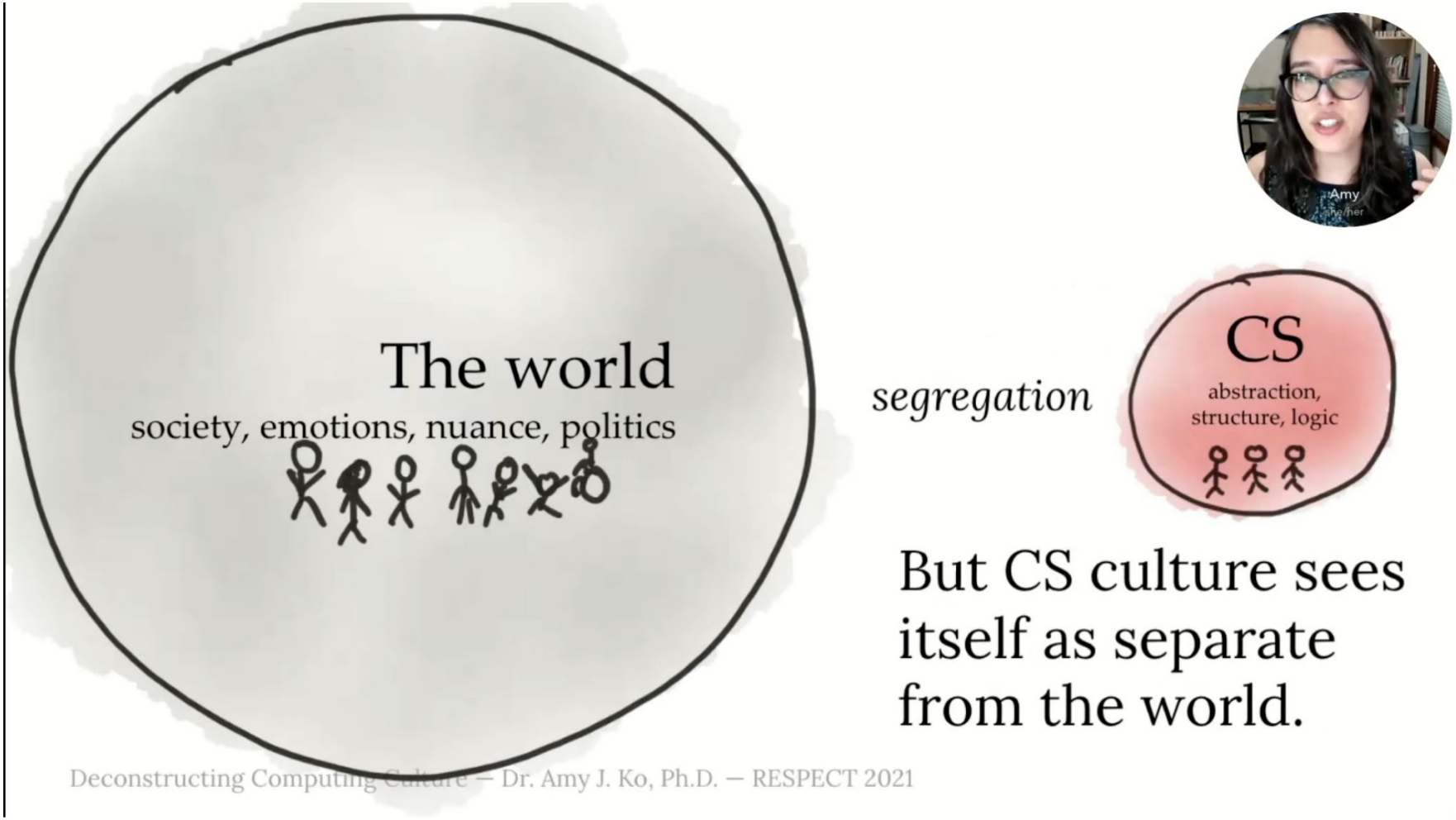
Intersections of complexity (1996)



CS is part of the world...



...but sees itself as separate...



Separate, and segregated

And it's not just social segregation, but intellectual segregation.

- CS **abstracts**, removing *messy social context*
- CS **neutralizes**, removing *nuanced values and politics*
- CS **normalizes**, erasing *diversity and exceptions*
- CS **automates**, removing *people and their unpredictable decisions*

These foundational CS concepts and values benefit the dominant groups in CS, reducing the burden of understand the complex social world, isolating them from its complexities.

Deconstructing Computing Culture — Dr. Amy J. Ko, Ph.D. — RESPECT 2021

So, what happened?

- Transgender bodies are “messy social context”

- My body is non-normative!
  - I don’t automate well, I don’t fit heuristics
  - Normalizing TSA scanners erases my diversity!
  - Even more so, I’m categorized as a threat!

Ask “is this ok?”

Humans are beautifully complex!

But caches are neutral!

The values underpinning them definitely aren't!

But caches are neutral!

The values underpinning them definitely aren't!

Also, they're not! Caching breaks on google drive when anyone changes their name!

But caches are neutral!

The values underpinning them definitely aren't!

Also, they're not! Caching breaks on google drive when anyone changes their name!

Questions?

Discuss!

Where else do you see normative assumptions made in tech or CS?

Notes Diagrams