Caches II

CSE 351 Autumn 2016

Instructor:

Justin Hsia

Teaching Assistants:

Chris Ma

Hunter Zahn

John Kaltenbach

Kevin Bi

Sachin Mehta

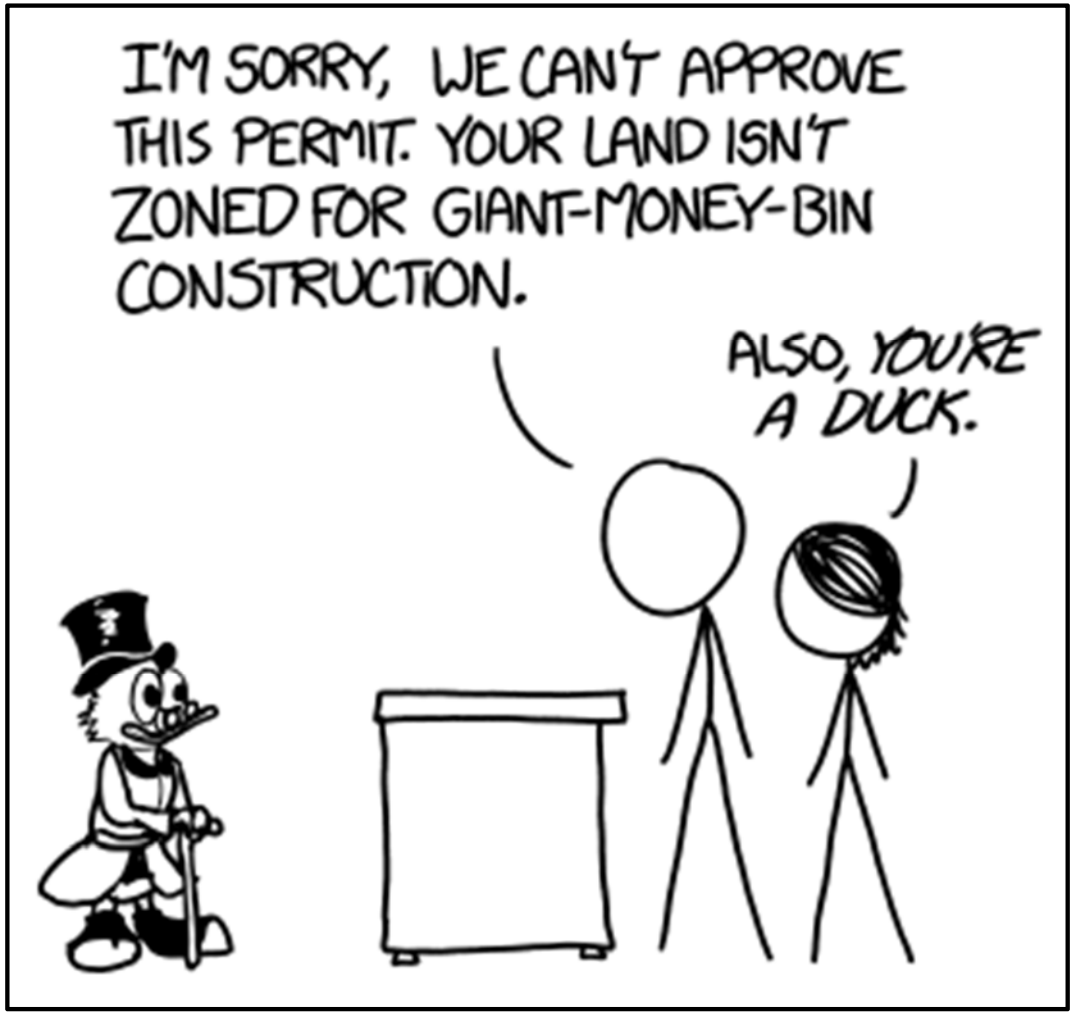
Suraj Bhat

Thomas Neuman

Waylon Huang

Xi Liu

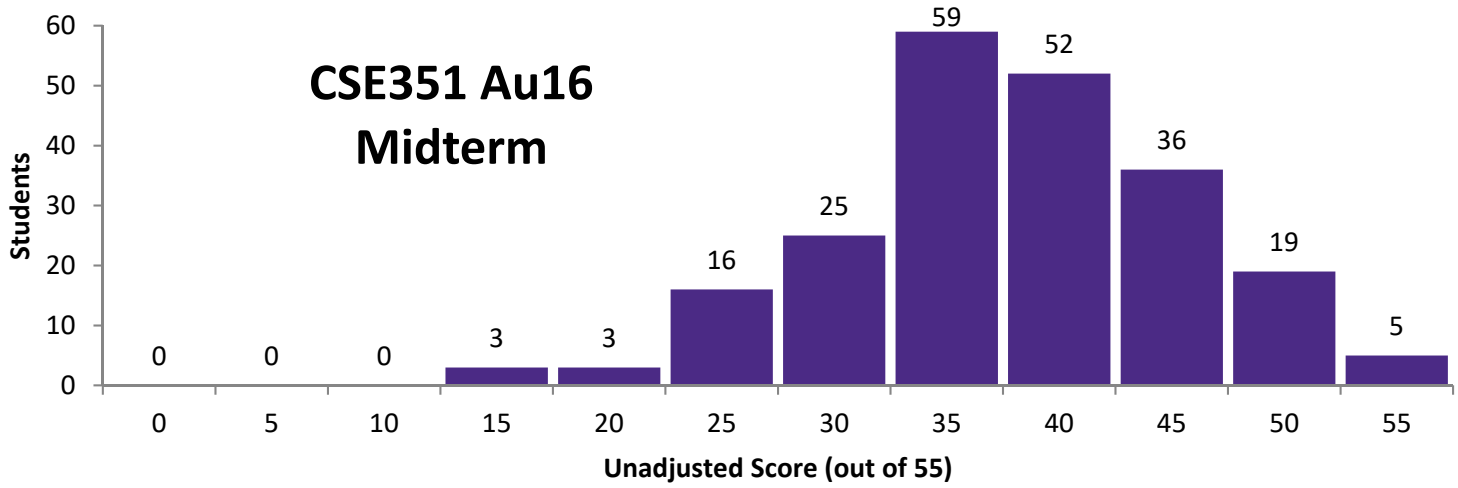
Yufang Sun



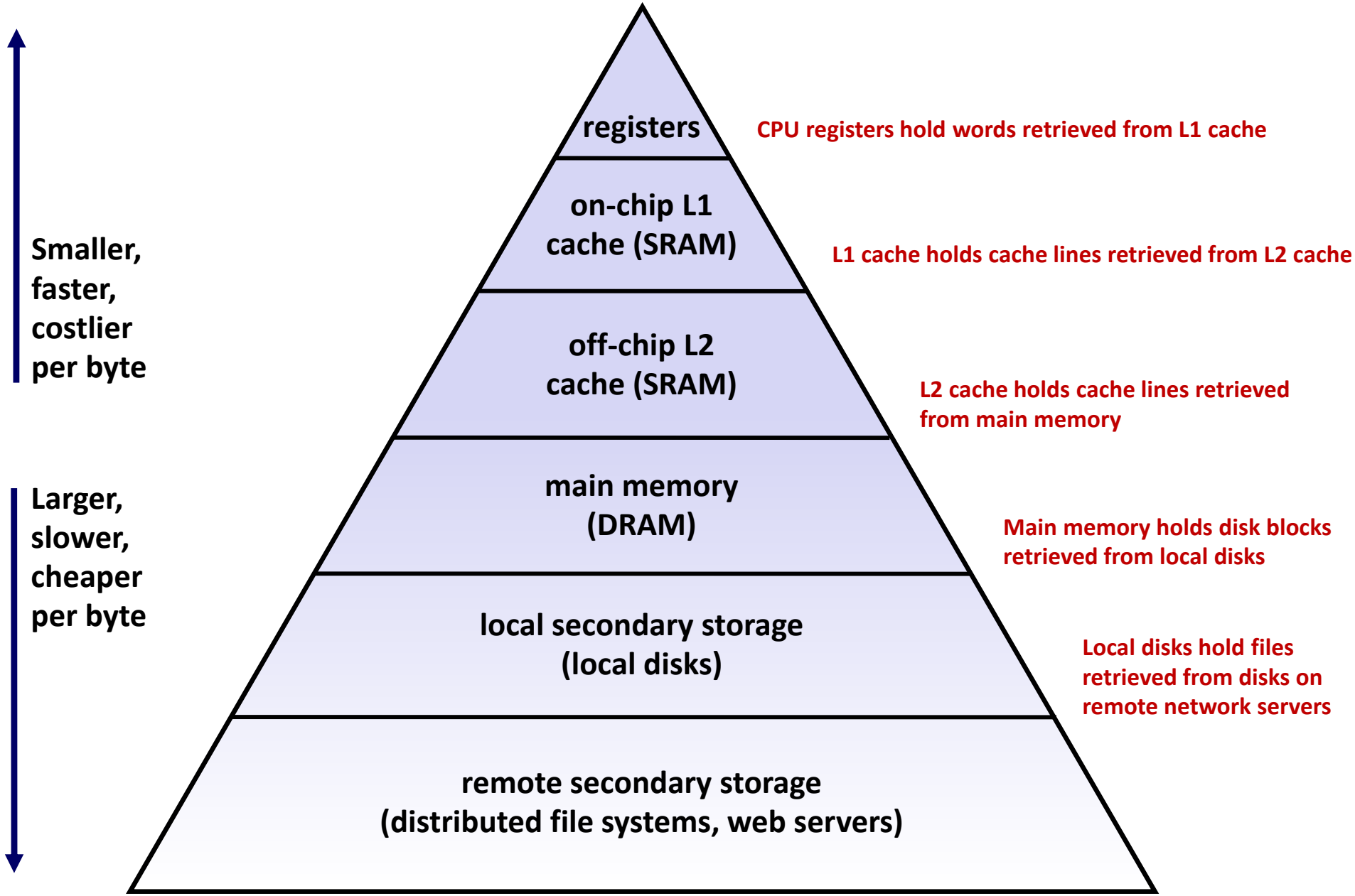
https://what-if.xkcd.com/111/

Administrivia

- Lab 3 due Thursday
- **Midterm grades**
  - Unadjusted average currently right around 65%
    - All scores will be adjusted up by 6 points in Catalyst
  - Regrade requests open on Gradescope until end of Thursday
    - It is possible for your grade to go *down*
    - Make sure you submit separate requests for each portion of a question (e.g. Q5A and Q5B) – these may go to different graders!



An Example Memory Hierarchy



Making memory accesses fast!

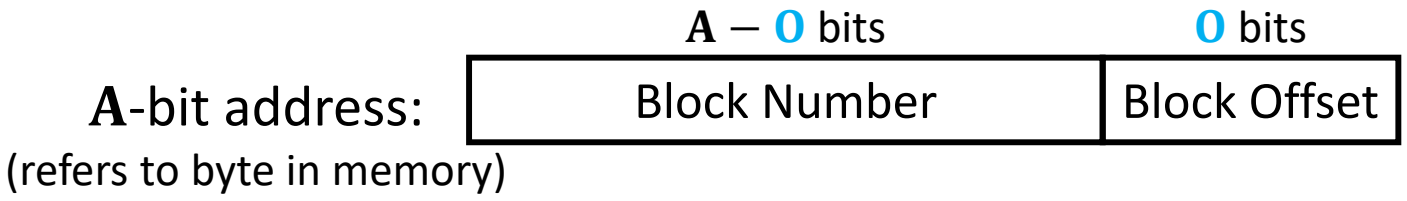
- Cache basics
- Principle of locality
- Memory hierarchies
- **Cache organization**
  - Direct-mapped (*sets*; index + tag)
  - Associativity (*ways*)
  - Replacement policy
  - Handling writes
- Program optimizations that consider caches

Cache Organization (1)

- Block Size (K): unit of transfer between $ and Mem
  - Given in bytes and always a power of 2 (e.g. 64 B)
  - Blocks consist of adjacent bytes (differ in address by 1)
    - Spatial locality!

Cache Organization (1)

- Block Size (K): unit of transfer between \$ and Mem
  - Given in bytes and always a power of 2 (e.g. 64 B)
  - Blocks consist of adjacent bytes (differ in address by 1)
    - Spatial locality!
- Offset field
  - Low-order $\log_2(\mathrm{K}) = \mathbf{O}$ bits of address tell you which byte within a block
    - (address) mod 2^n = n lowest bits of address
  - (address) modulo (# of bytes in a block)



Cache Organization (2)

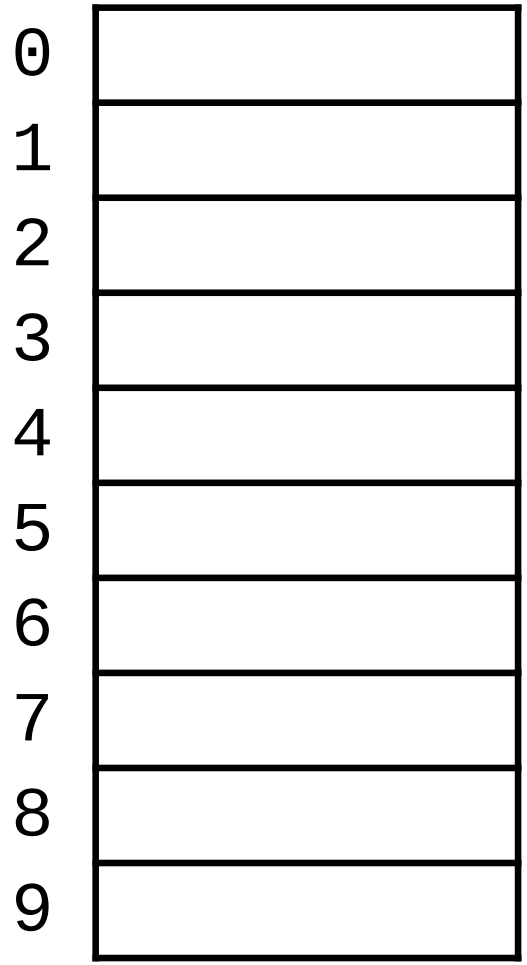
- Cache Size (C): amount of *data* the $ can store
  - Cache can only hold so much data (subset of next level)
  - Given in bytes (C) or number of blocks (C/K)
  - Example: C = 32 KiB = 512 blocks if using 64-B blocks
- Where should data go in the cache?
  - We need a mapping from memory addresses to specific locations in the cache to make checking the cache for an address **fast**
- What is a data structure that provides fast lookup?
  - Hash table!

Aside: Hash Tables for Fast Lookup

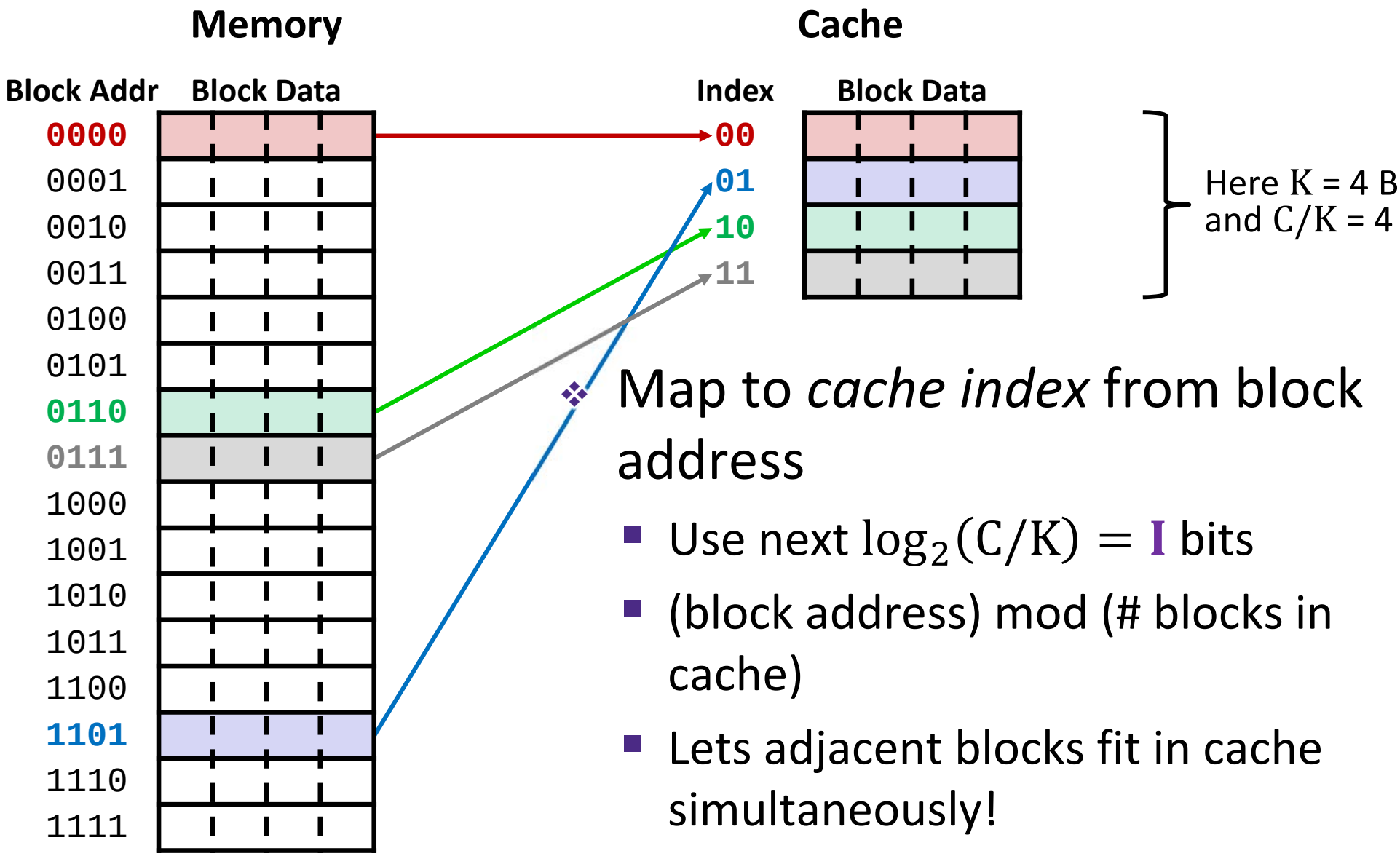
Insert:

5
27
34
102
119

Apply hash function to map data to "buckets"



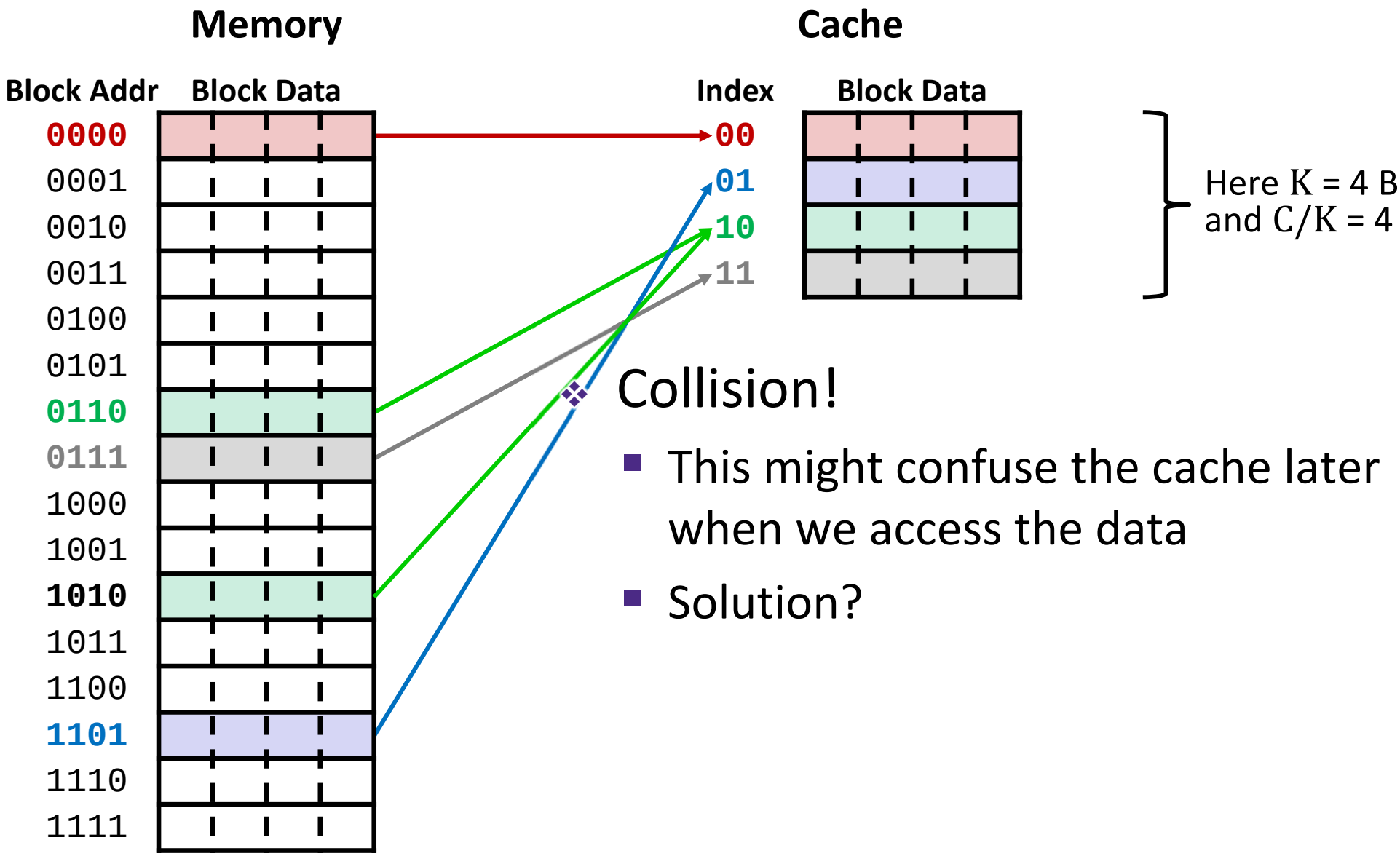
Place Data in Cache by Hashing Address



Map to *cache index* from block address

- Use next $\log_2(C/K) = I$ bits
- (block address) mod (# blocks in cache)
- Lets adjacent blocks fit in cache simultaneously!

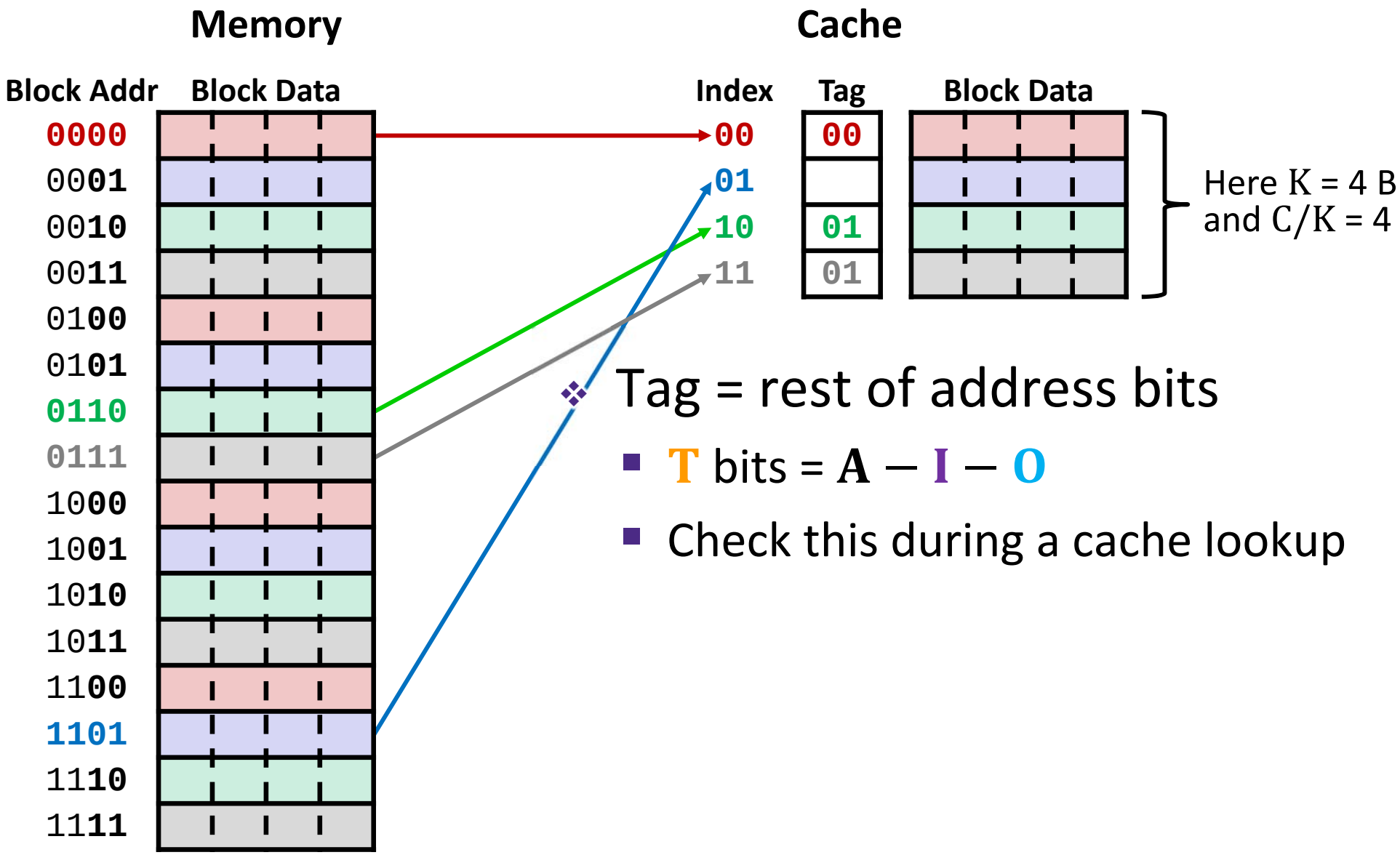
Place Data in Cache by Hashing Address



Collision!

- This might confuse the cache later when we access the data
- Solution?

Tags Differentiate Blocks in Same Index

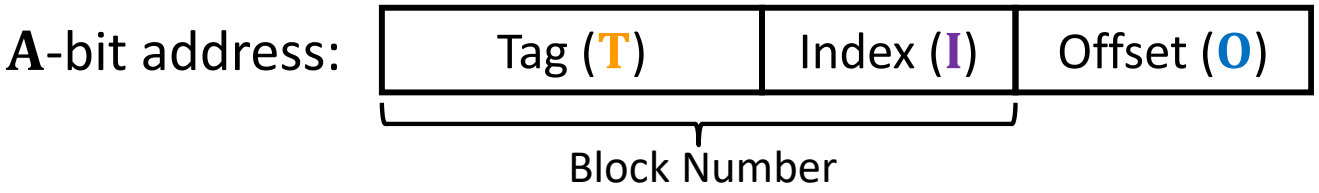


- Tag = rest of address bits
  - T bits = A – I – O
  - Check this during a cache lookup

Checking for a Requested Address

- CPU sends address request for chunk of data
  - Address and requested data are not the same thing!
    - Analogy: your friend ≠ his or her phone number

- TIO address breakdown:



  - **Index** field tells you where to look in cache
  - **Tag** field lets you check that data is the block you want
  - **Offset** field selects specified start byte within block

  - **Note:** T and I sizes will change based on hash function

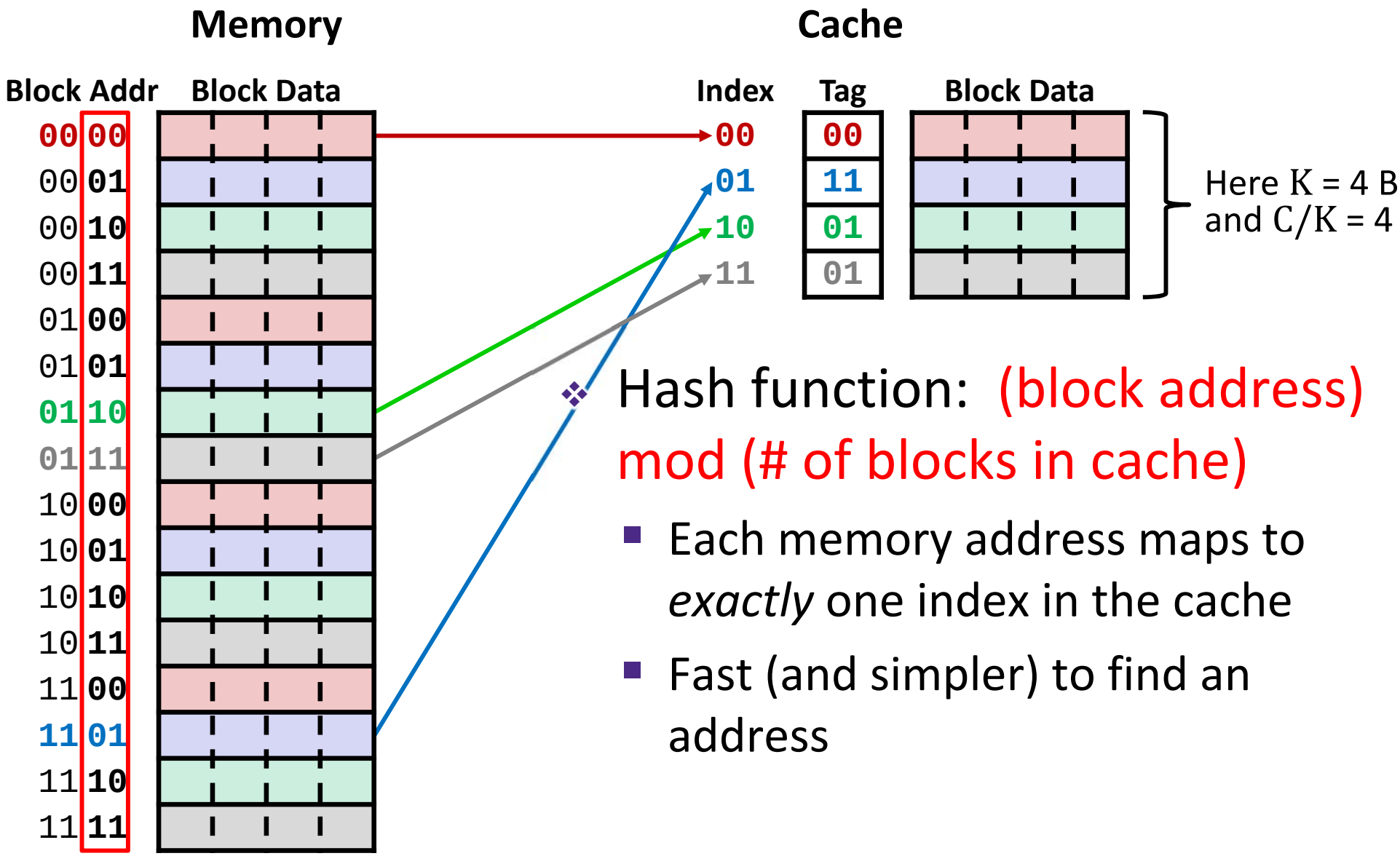
Cache Puzzle #1

- What can you infer from the following behavior?
  - Cache starts *empty*, also known as a *cold cache*
  - Access (addr: hit/miss) stream:
    - (14: miss), (15: hit), (16: miss)

- Minimum block size?

- Maximum block size?

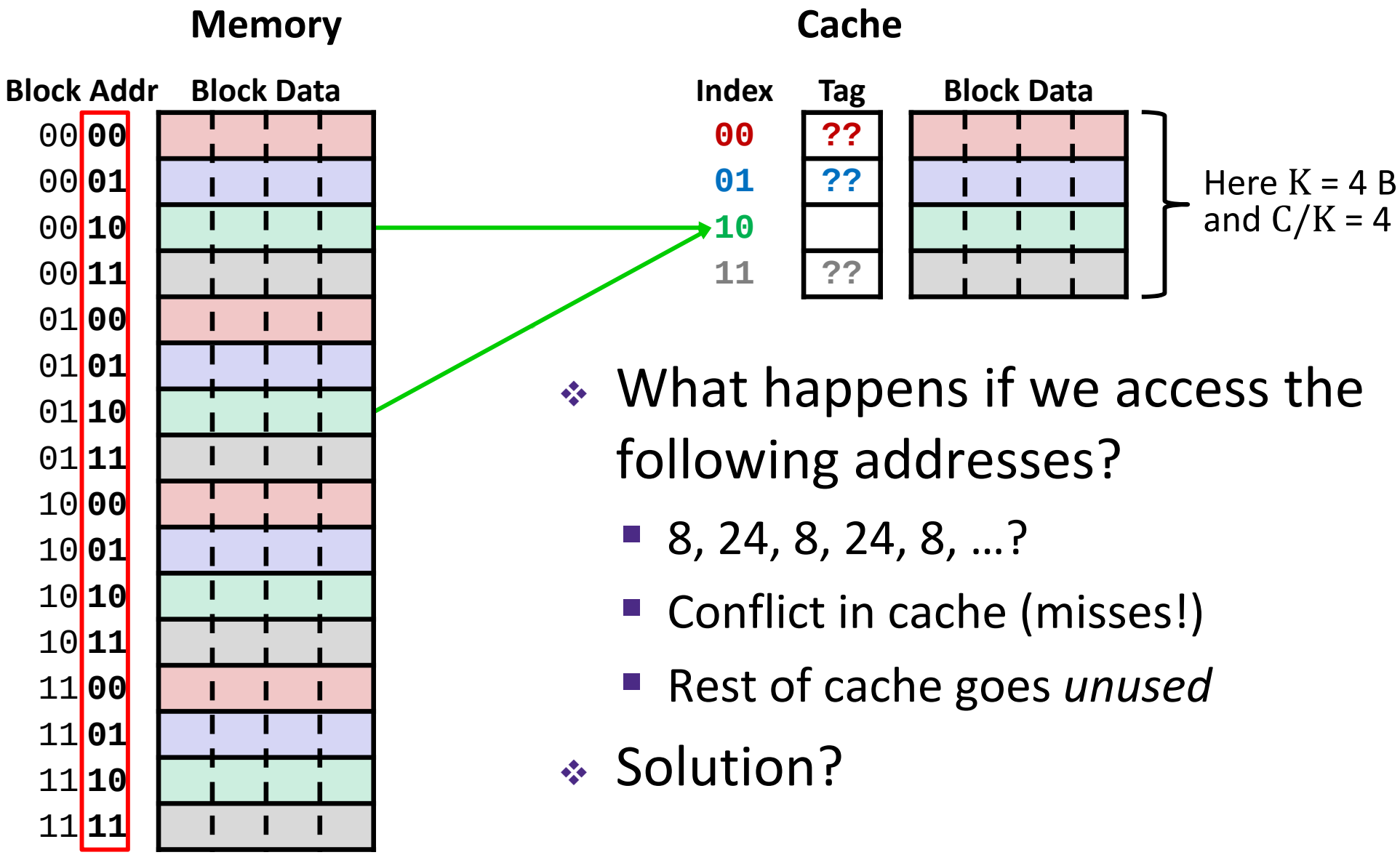
Direct-Mapped Cache



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

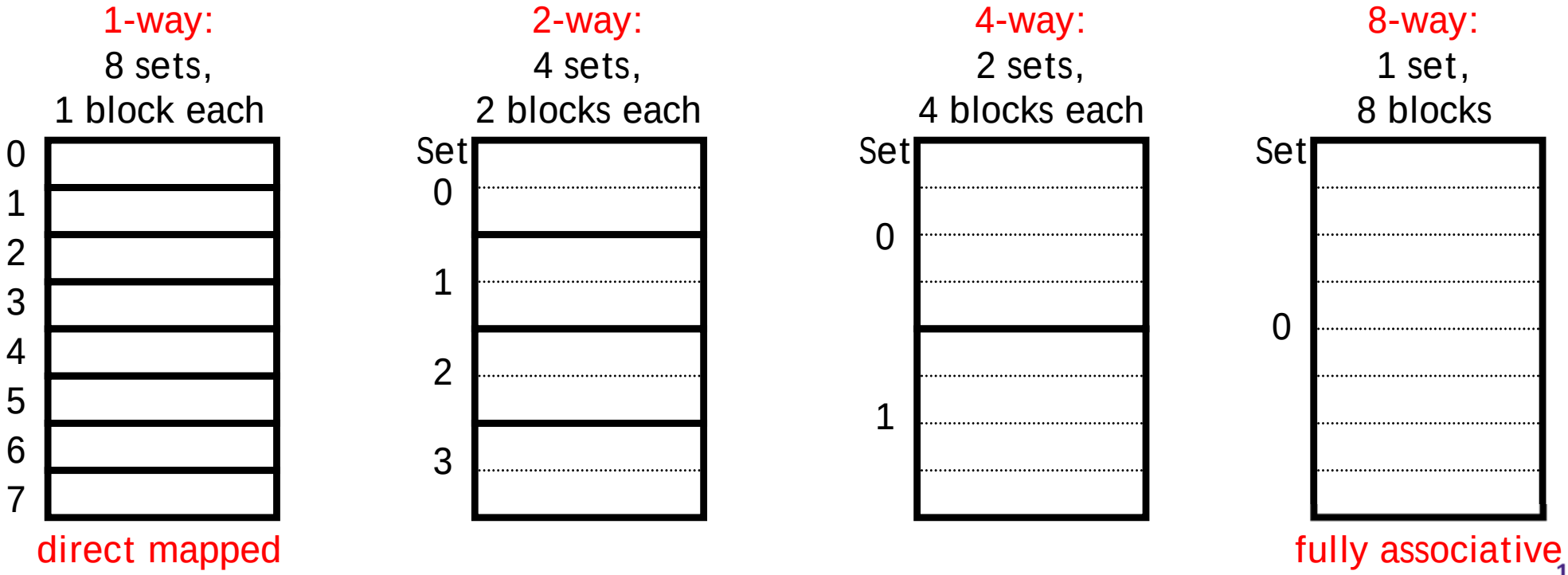
Direct-Mapped Cache Problem



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

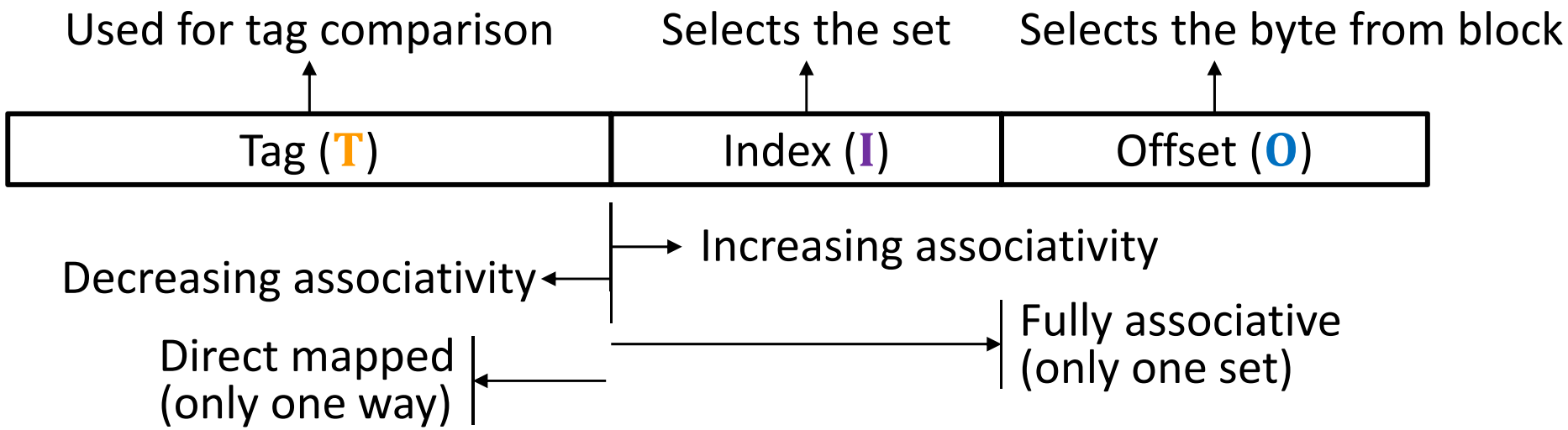
Associativity

- 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)

- Associativity (N): # of ways for each set
  - Such a cache is called an "*N-way set associative cache*"
  - We now index into cache *sets*, of which there are C/K/N
  - Use lowest $\log_2(C/K/N)$ = **I** bits of block address
    - Direct-mapped: N = 1, so **I** = $\log_2(C/K)$ as we saw previously
    - Fully associative: N = C/K, so **I** = 0 bits

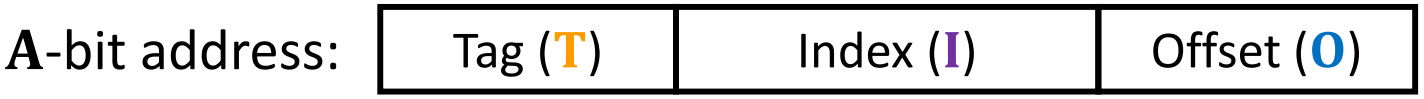


Example Placement

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

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

$\mathbf{T} = \mathbf{A}–\mathbf{I}–\mathbf{O}$ $\quad \mathbf{I} = \log_2(\mathrm{C/K/N})$ $\quad \mathbf{O} = \log_2(\mathrm{K})$



I = ?
Direct-mapped

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

I = ?
2-way set associative

Set	Tag	Data
0		
1		
2		
3		

I = ?
4-way set associative

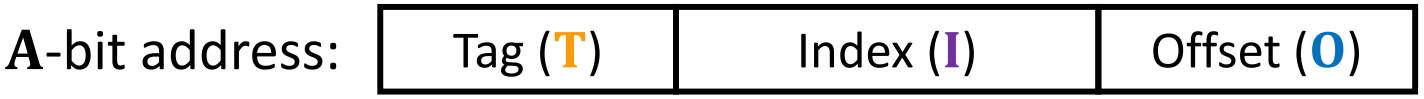
Set	Tag	Data
0		
1		

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`

$\mathbf{T} = \mathbf{A}–\mathbf{I}–\mathbf{O}$ $\quad \mathbf{I} = \log_2(\mathrm{C/K/N})$ $\quad \mathbf{O} = \log_2(\mathrm{K})$



I = 3
Direct-mapped

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

I = 2
2-way set associative

Set	Tag	Data
0		
1		
2		
3		

I = 1
4-way set associative

Set	Tag	Data
0		
1		

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

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		

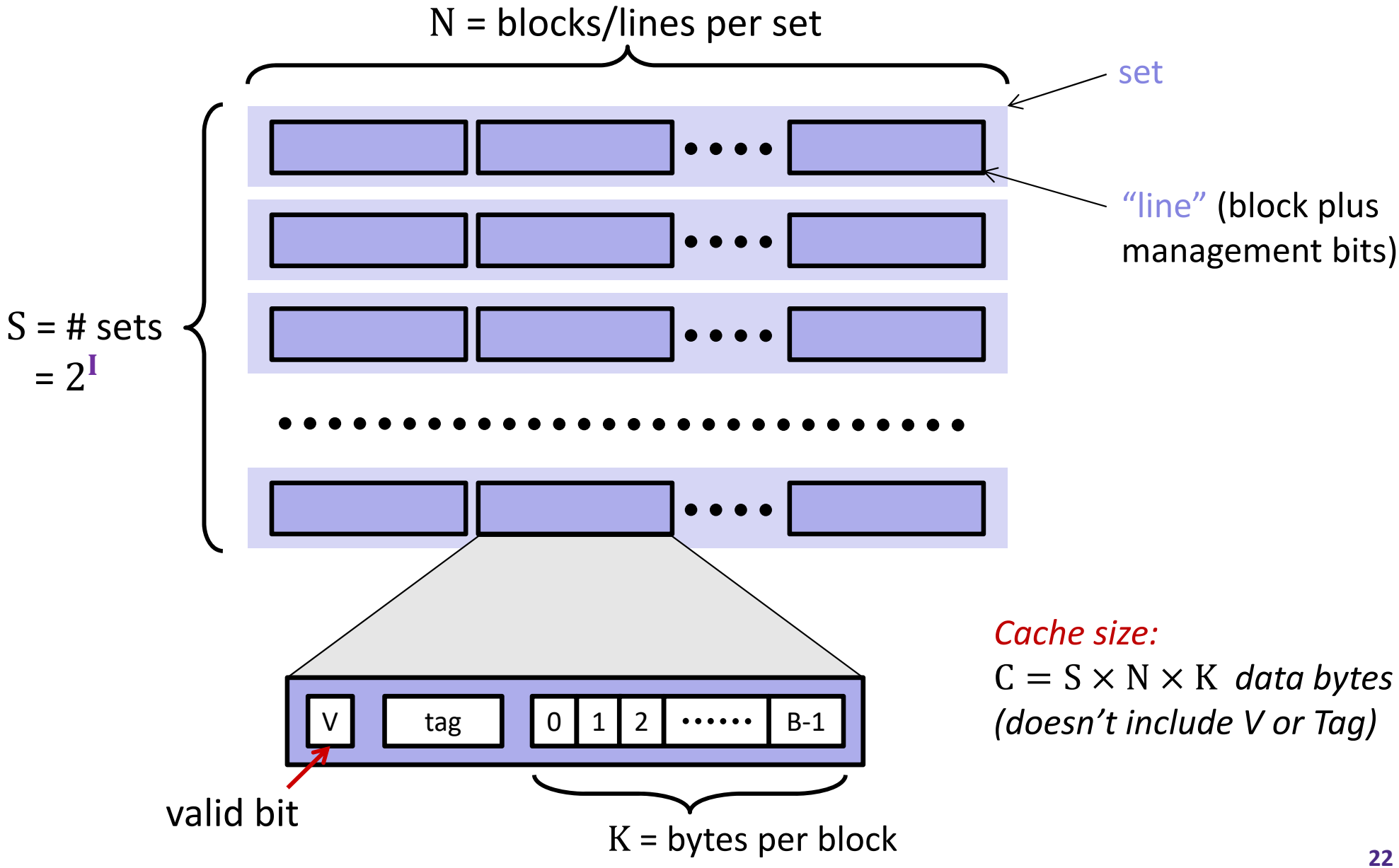
Cache Puzzle #2

- What can you infer from the following behavior?
  - Cache starts *empty*, also known as a *cold cache*
  - Access (addr: hit/miss) stream:
    - (10: miss), (12: miss), (10: miss)

- Associativity?

- Number of sets?

General Cache Organization (S, N, K)



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

Notation Changes

- We are using different variable names from previous quarters
  - Please be aware of this when you look at past exam questions or are watching videos

Variable	This Quarter	Previous Quarters
Block size	K	B
Cache size	C	---
Associativity	N	E
Address width	A	m
Tag field width	T	t
Index field width	I	k, s
Offset field width	O	n, b
Number of Sets	S	S