

# Caches II

CSE 351 Autumn 2019

## Instructor:

Justin Hsia

## Teaching Assistants:

Andrew Hu

Antonio Castelli

Cosmo Wang

Diya Joy

Ivy Yu

Kaelin Laundry

Maurice Montag

Melissa Birchfield

Millicent Li

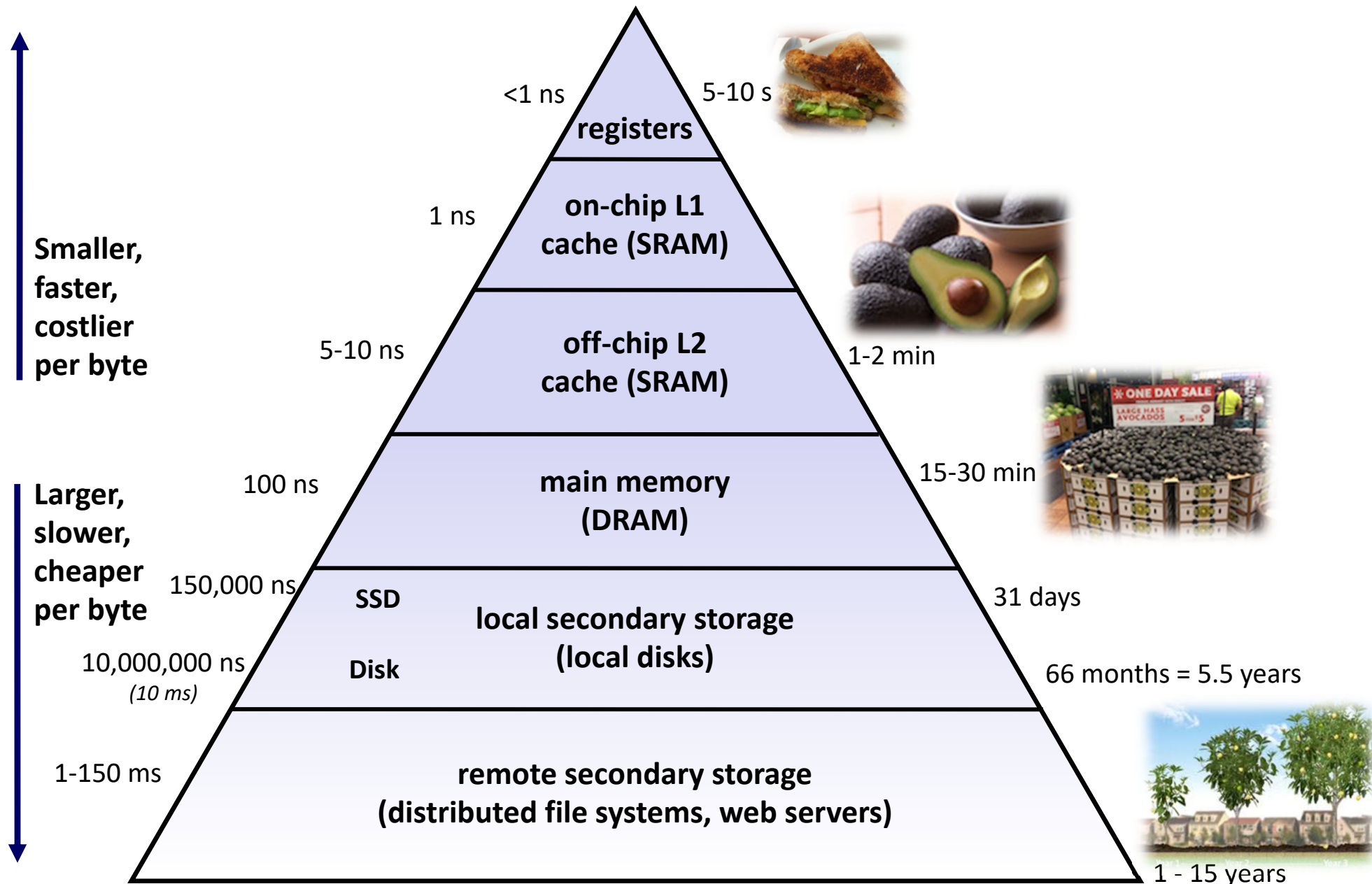
Suraj Jagadeesh



# Administrivia

- ❖ hw15 due Wednesday, hw16 due *next* Wed (11/20)
  - Don't wait too long, this is a BIG hw (11.5 points)
- ❖ Lab 3 due Friday (11/8)
  
- ❖ **Mid-Quarter Survey Feedback**
  - Pace is “moderate” to “a bit too fast”
  - Lecture sometimes moves too quickly, especially at the end
  - Midterm was difficult and under significant time pressure
  - String matching on Gradescope quizzes is super frustrating

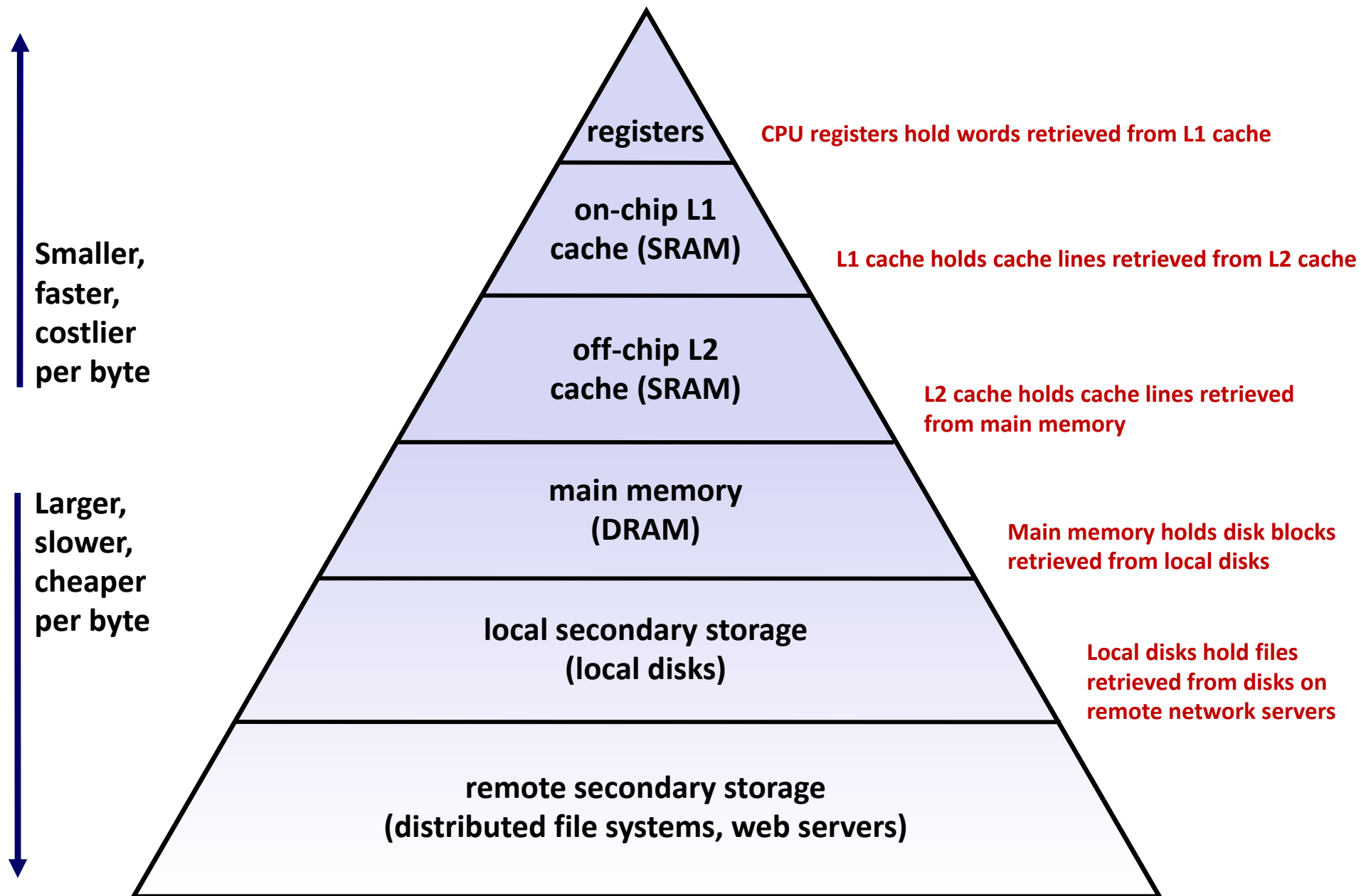
# An Example Memory Hierarchy



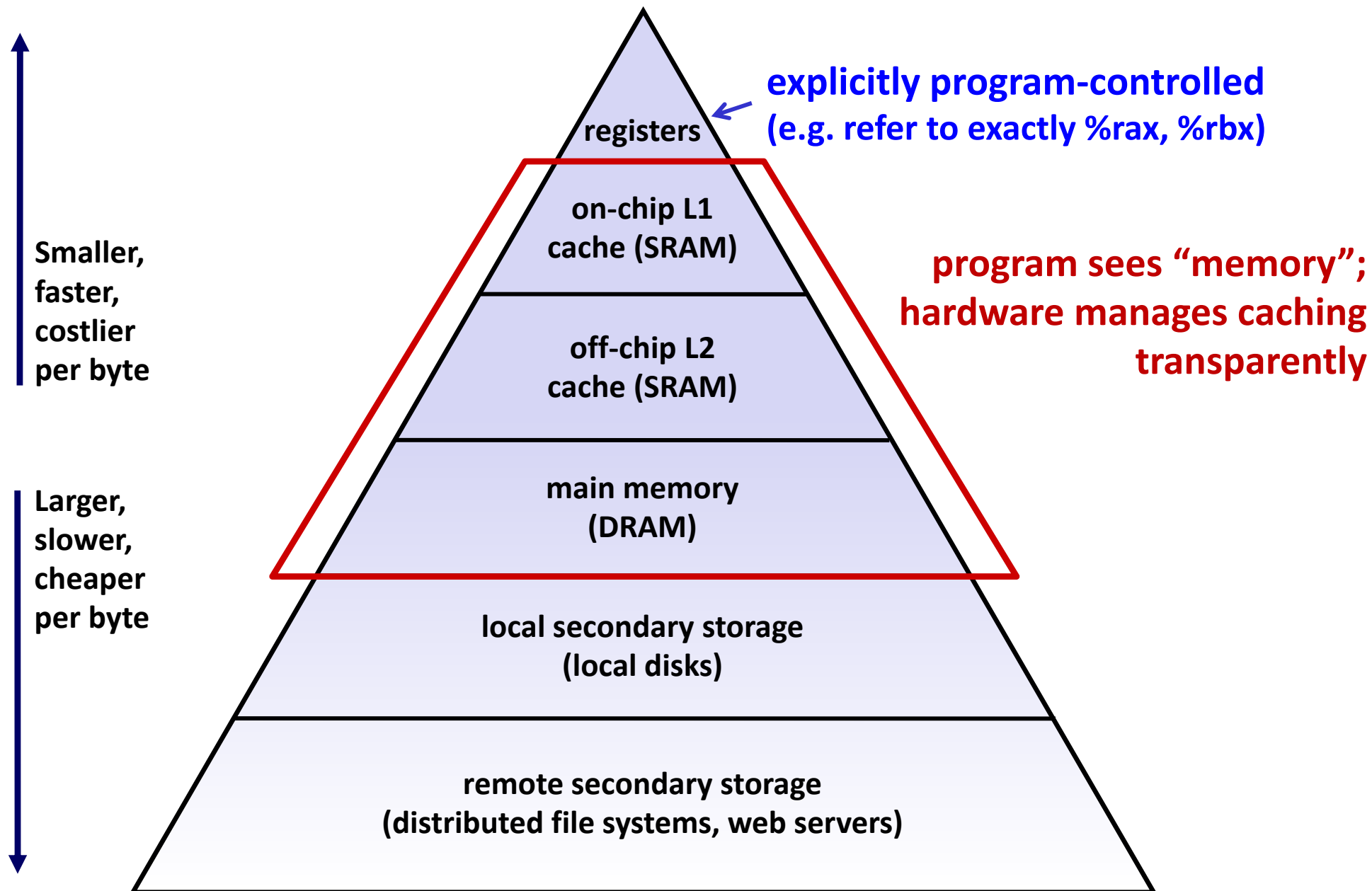
# Memory Hierarchies

- ❖ Some fundamental and enduring properties of hardware and software systems:
  - Faster storage technologies almost always cost more per byte and have lower capacity
  - The gaps between memory technology speeds are widening
    - True for: registers  $\leftrightarrow$  cache, cache  $\leftrightarrow$  DRAM, DRAM  $\leftrightarrow$  disk, etc.
  - Well-written programs tend to exhibit good locality
  
- ❖ These properties complement each other beautifully
  - They suggest an approach for organizing memory and storage systems known as a memory hierarchy
    - For each level  $k$ , the faster, smaller device at level  $k$  serves as a cache for the larger, slower device at level  $k+1$

# An Example Memory Hierarchy

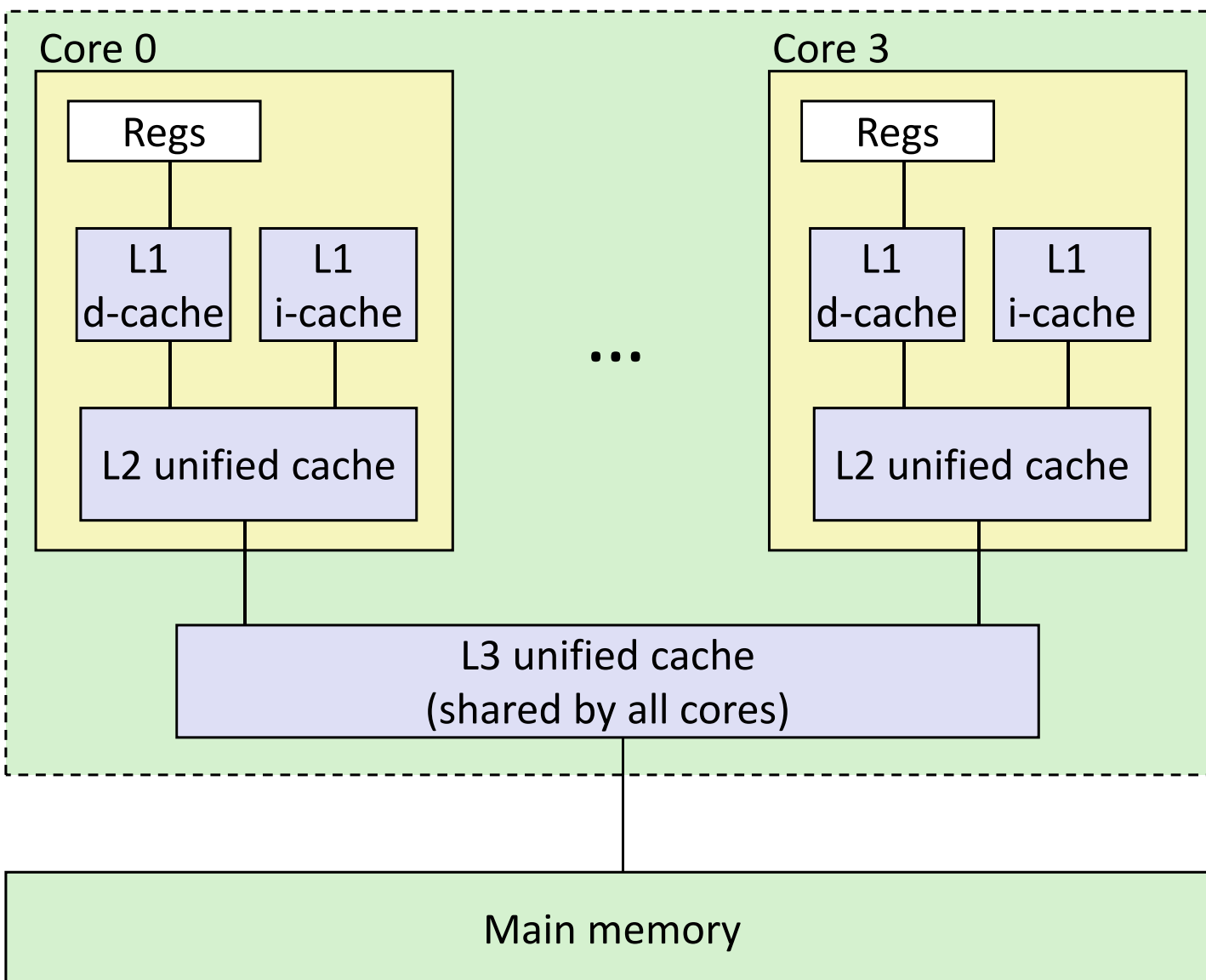


# An Example Memory Hierarchy



# Intel Core i7 Cache Hierarchy

## Processor package



### 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

# 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)

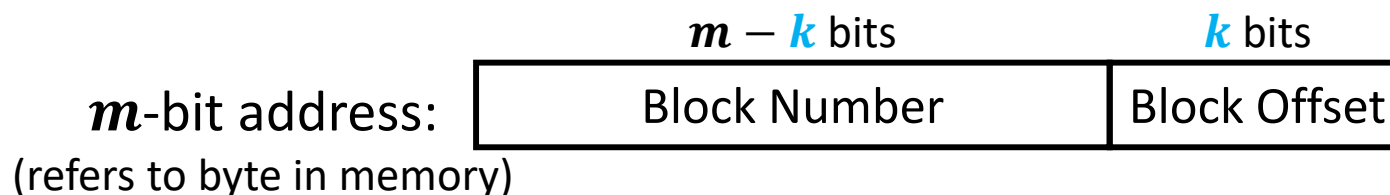
**Note:** The textbook uses “B” for block size

- ❖ **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)

**Note:** The textbook uses “b” for offset bits

- ❖ **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  $\log_2(K) = k$  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)



# Polling Question

- ❖ If we have 6-bit addresses and block size  $K = 4$  B, which block and byte does 0x15 refer to?
  - Vote at: <http://PollEv.com/justinh>

	Block Num	Block Offset
A.	1	1
B.	1	5
C.	5	1
D.	5	5
E.	We're lost...	

# 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:  $C = 32 \text{ KiB} = 512$  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!

# Review: Hash Tables for Fast Lookup

**Insert:**

5

27

34

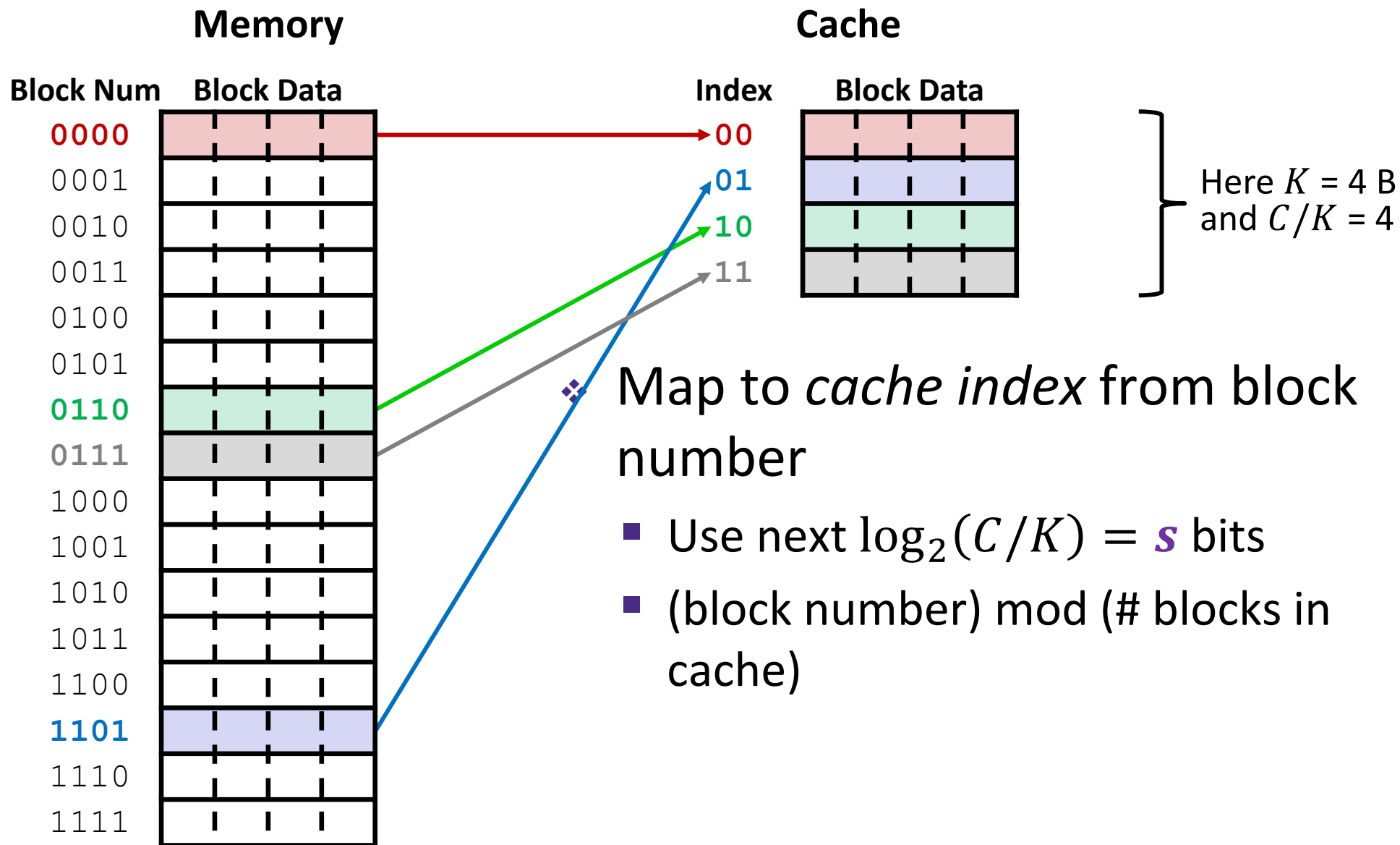
102

119

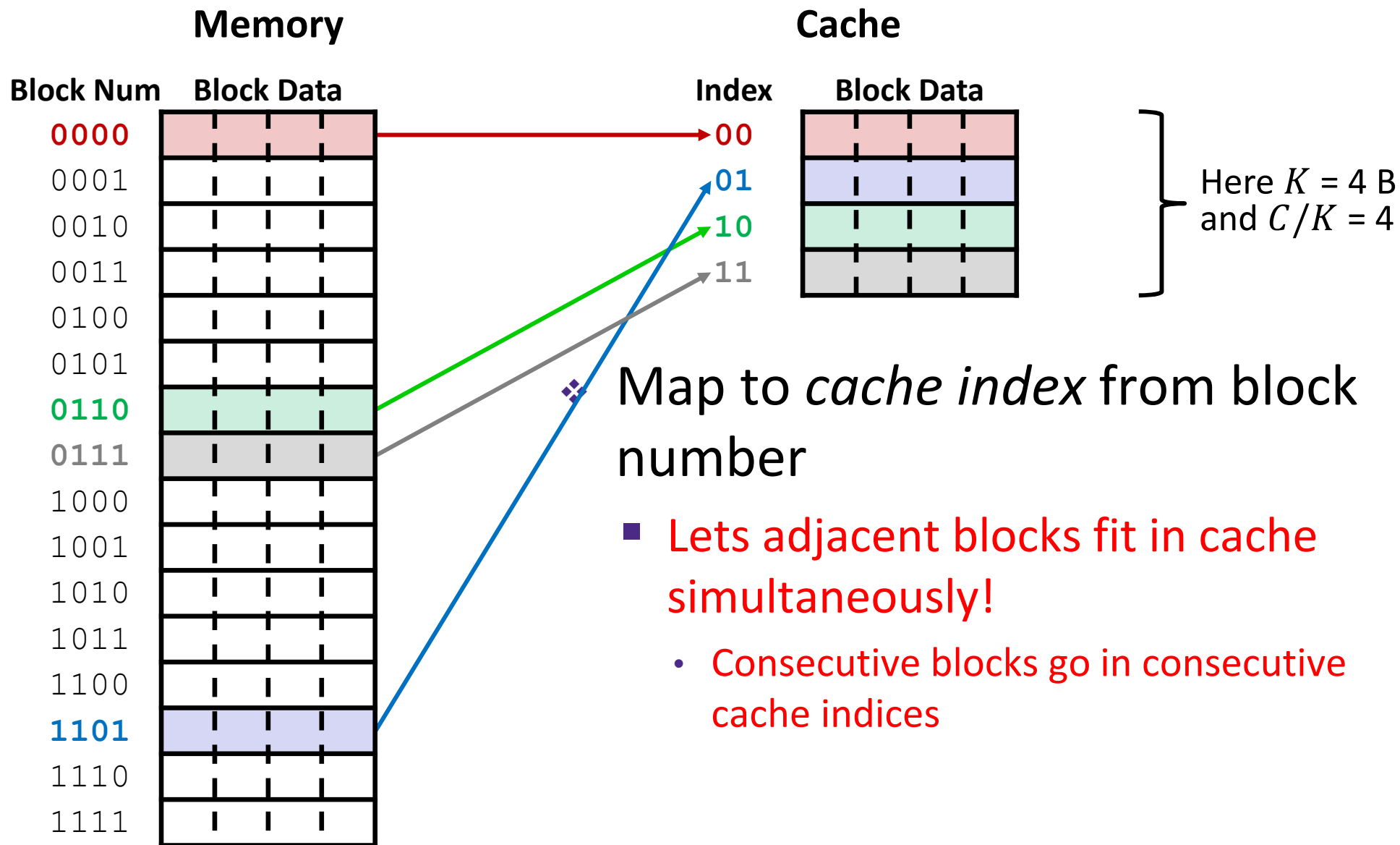
Apply hash function to map data  
to “buckets”

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

# Place Data in Cache by Hashing Address



# Place Data in Cache by Hashing Address

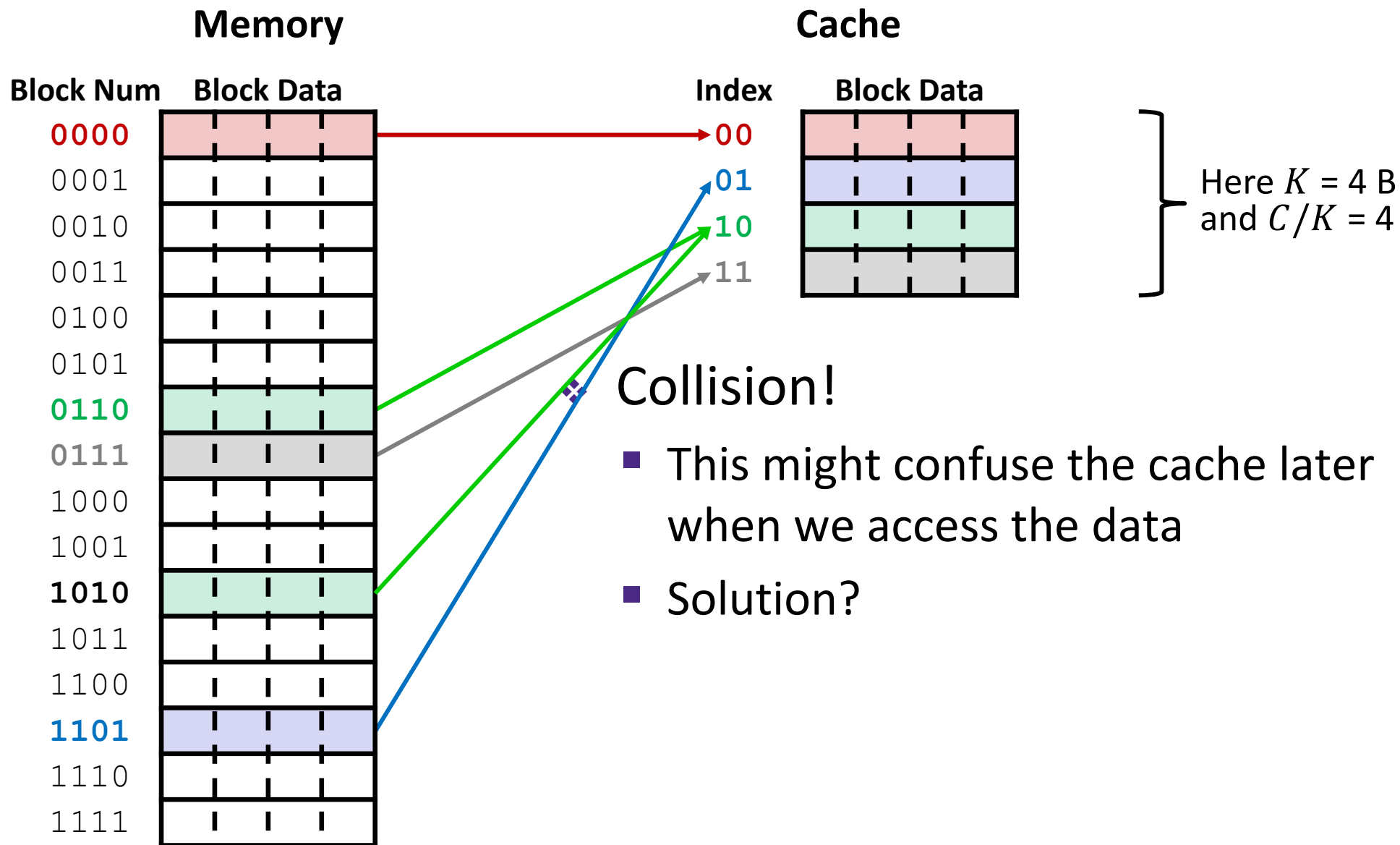


# Practice Question

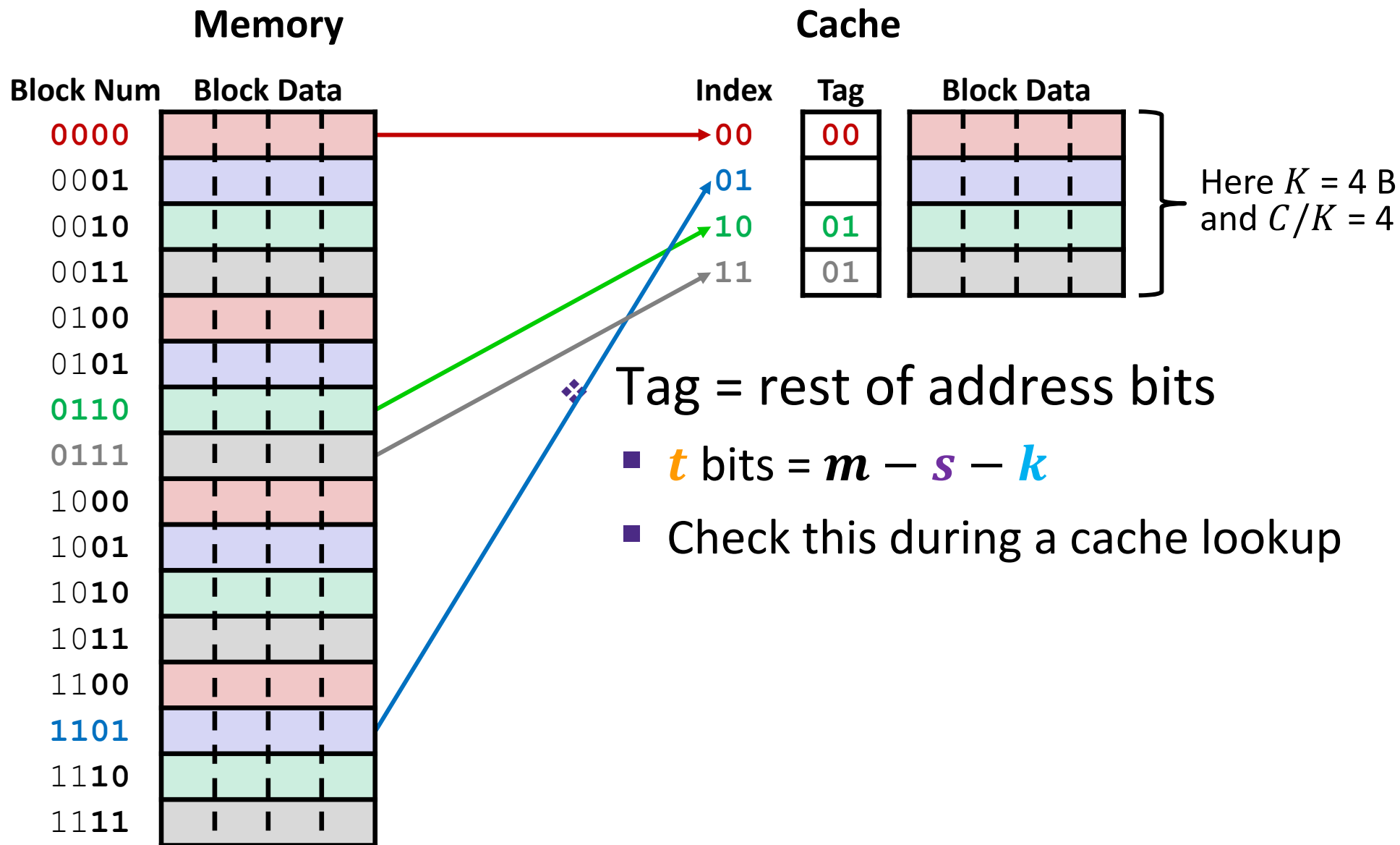
- ❖ 6-bit addresses, block size  $K = 4$  B, and our cache holds  $S = 4$  blocks.
- ❖ A request for address **0x2A** results in a cache miss. Which index does this block get loaded into and which 3 other addresses are loaded along with it?
  - No voting for this question



# Place Data in Cache by Hashing Address



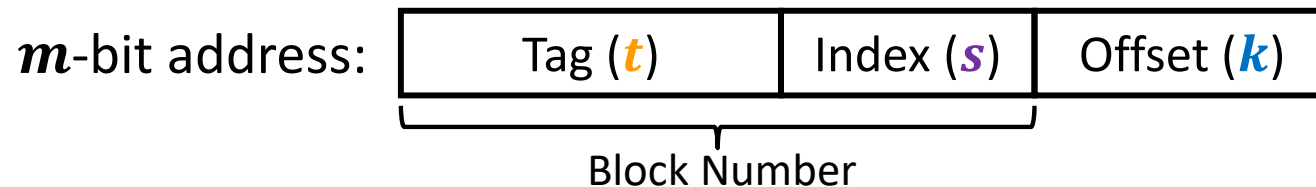
# Tags Differentiate Blocks in Same Index



# 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  $\neq$  his or her phone number

- ❖ TIO address breakdown:



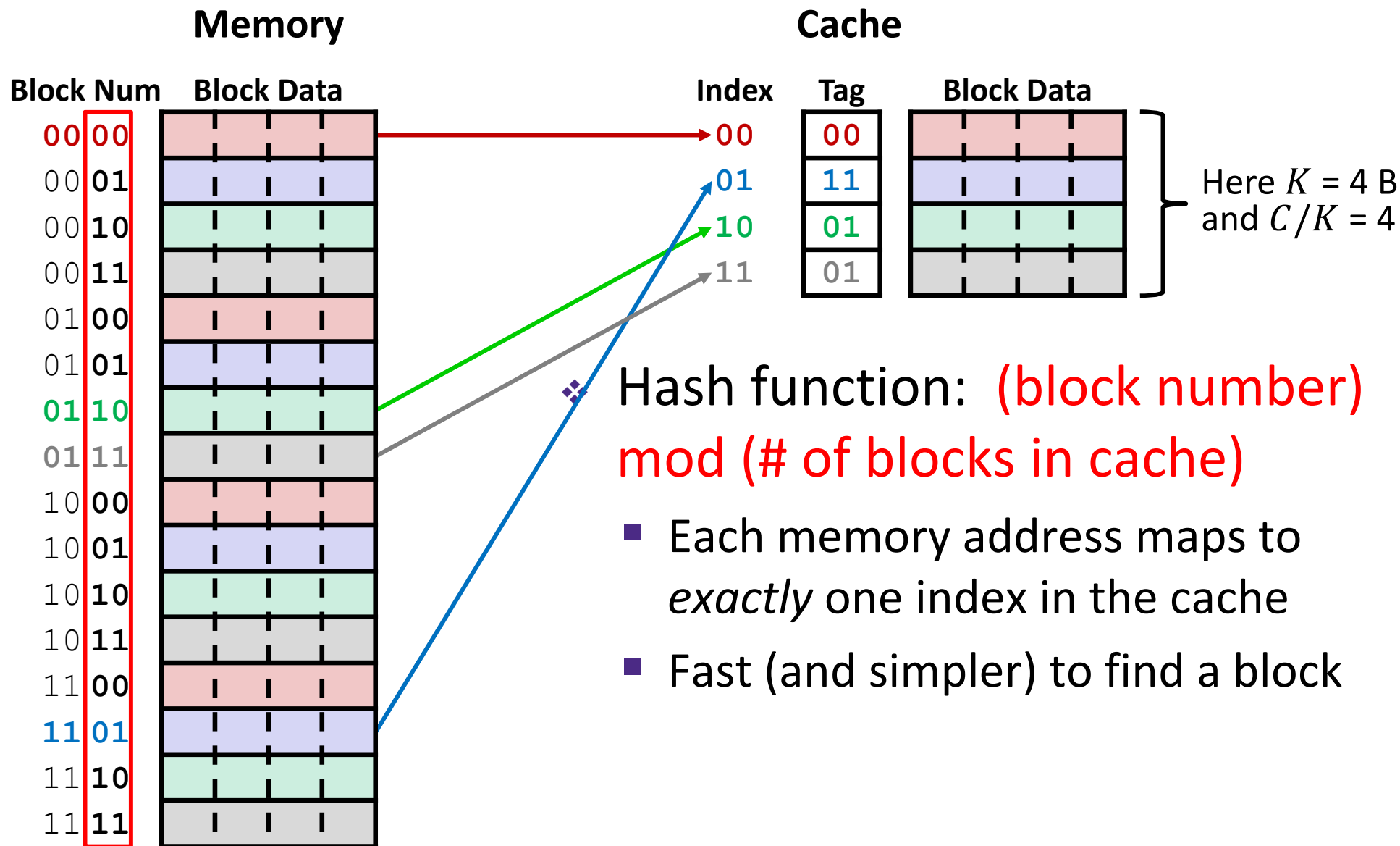
- **Index** field tells you where to look in cache
- **Tag** field lets you check that data is the block you want
- **Offset** field selects specified start byte within block
- **Note:** *t* and *s* sizes will change based on hash function

# Cache Puzzle

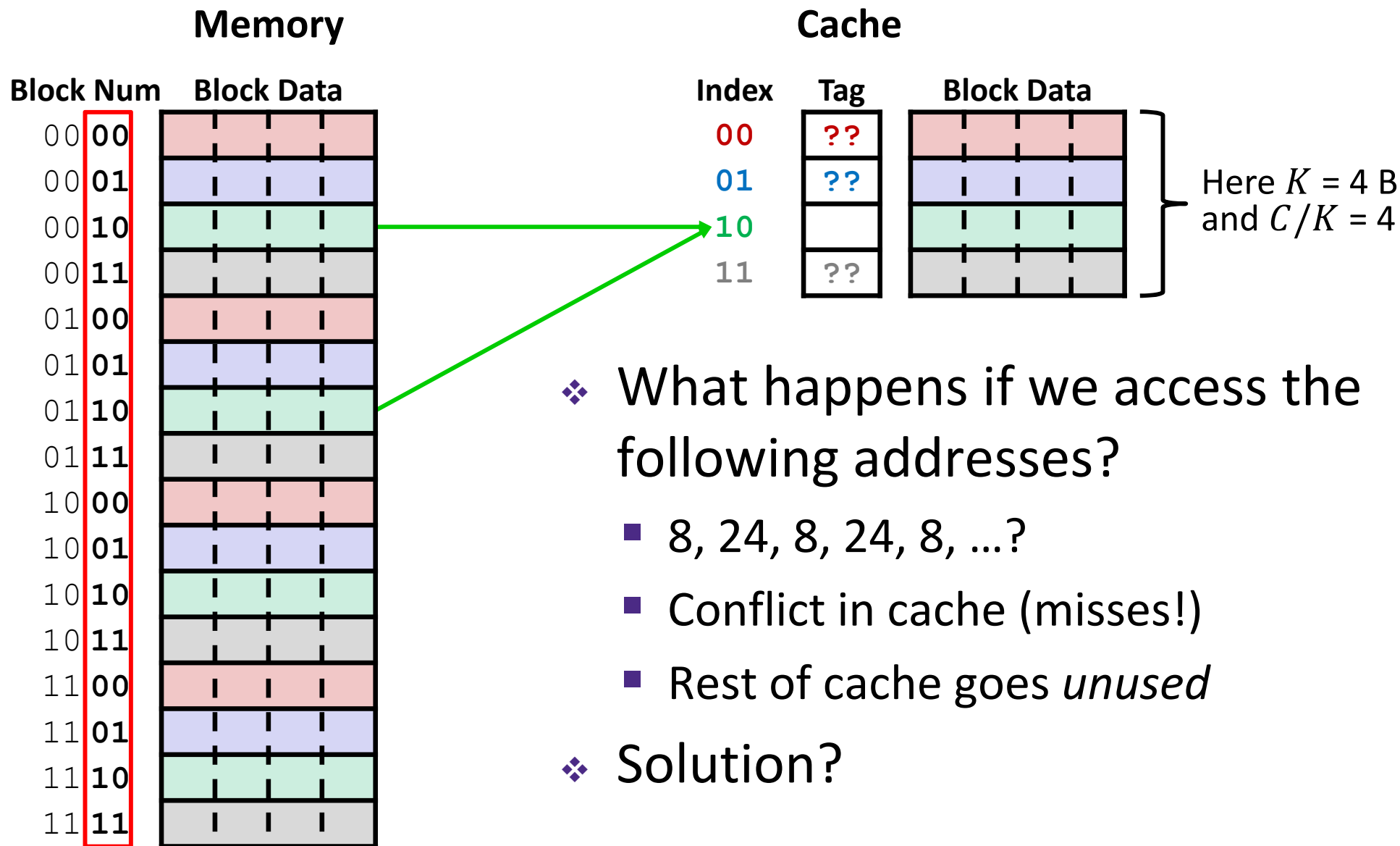
Vote at <http://PollEv.com/justinh>

- ❖ Based on the following behavior, which of the following block sizes is NOT possible for our cache?
  - Cache starts *empty*, also known as a *cold cache*
  - Access (addr: hit/miss) stream:
    - (14: miss), (15: hit), (16: miss)
  
- A. 4 bytes
- B. 8 bytes
- C. 16 bytes
- D. 32 bytes
- E. We're lost...

# Direct-Mapped Cache



# 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?