

Memory & Caches II

CSE 351 Autumn 2021

Instructor:

Justin Hsia

Teaching Assistants:

Allie Pflieger

Anirudh Kumar

Assaf Vayner

Atharva Deodhar

Celeste Zeng

Dominick Ta

Francesca Wang

Hamsa Shankar

Isabella Nguyen

Joy Dang

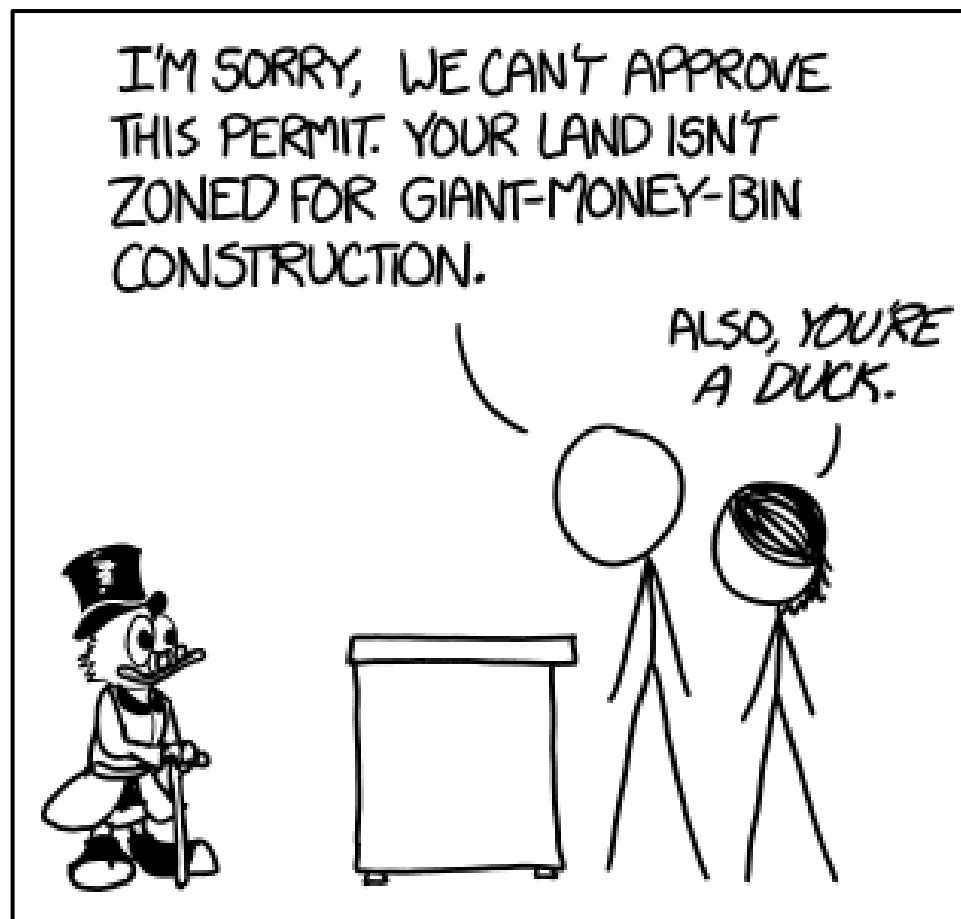
Julia Wang

Maggie Jiang

Monty Nitschke

Morel Fotsing

Sanjana Chintalapati



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

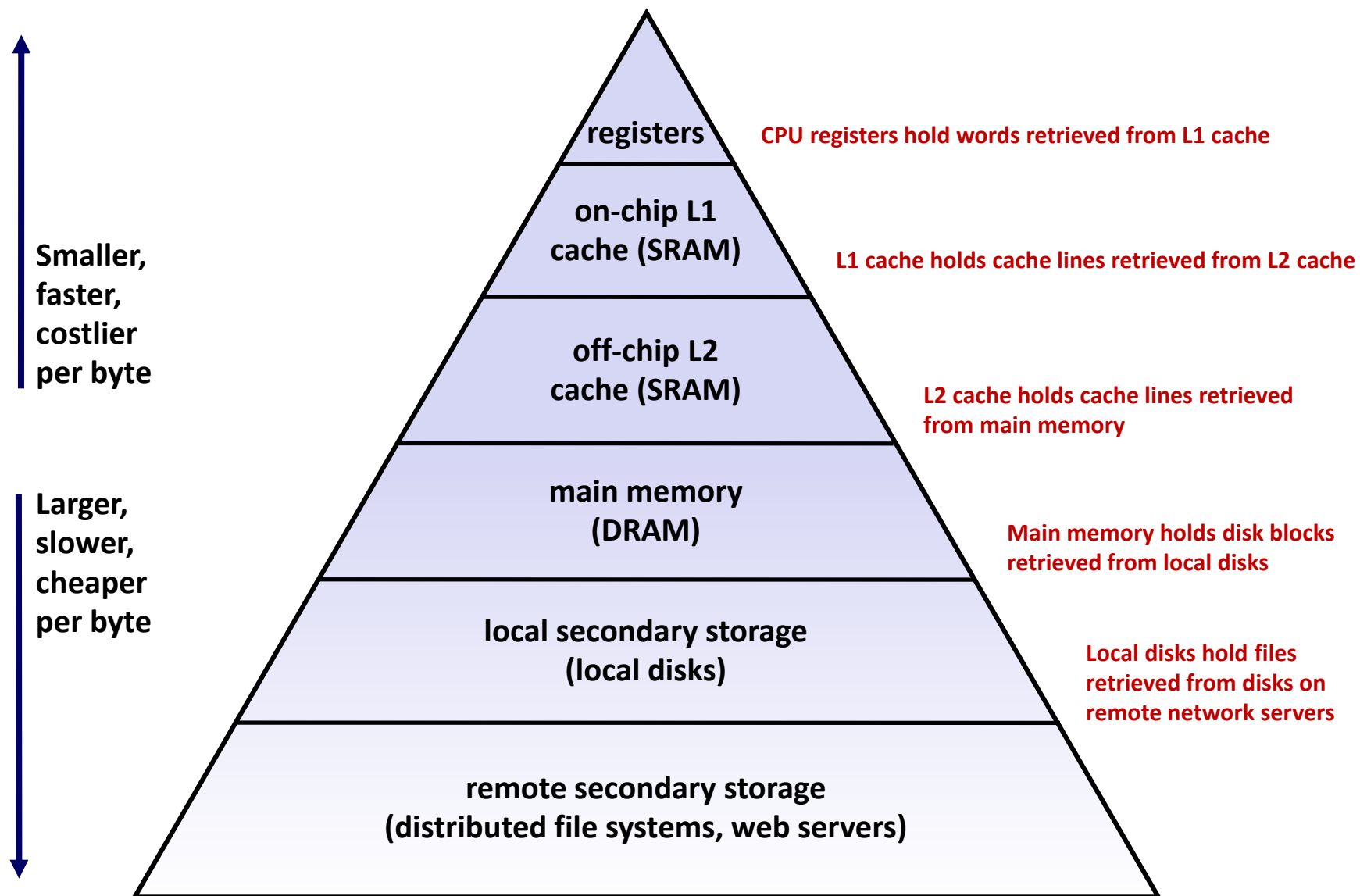
Relevant Course Information

- ❖ Mid-quarter Survey due Wednesday (11/10)
- ❖ hw16 due Friday (11/12)
- ❖ hw19 due *next* Wednesday (11/17)
 - Don't wait too long, this is a BIG hw (includes this lecture)
- ❖ Lab 3 due Friday (11/12)
- ❖ Midterm grades will be released when we can
 - Regrade requests will be available afterward

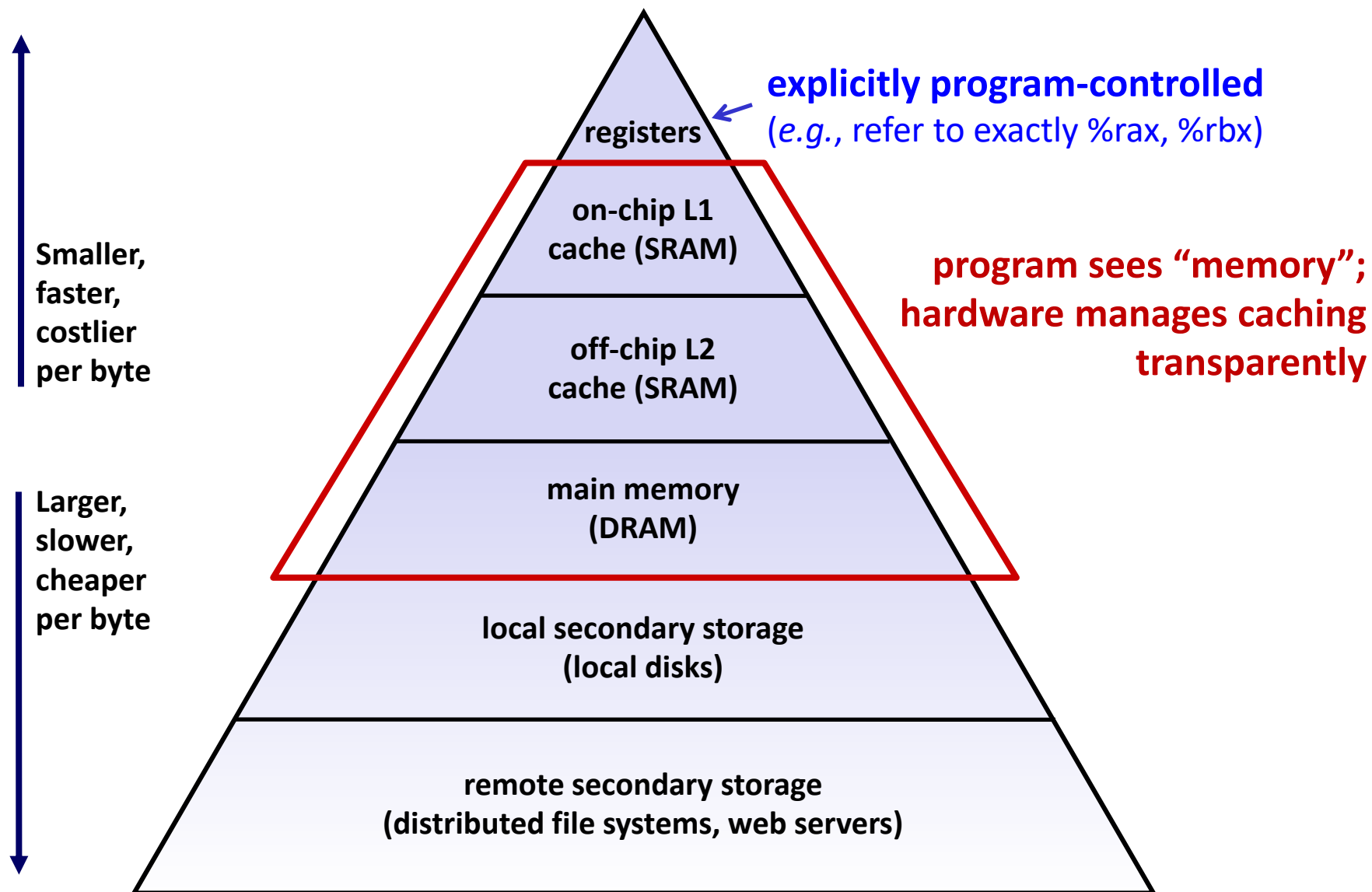
Memory Hierarchies (Review)

- ❖ 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

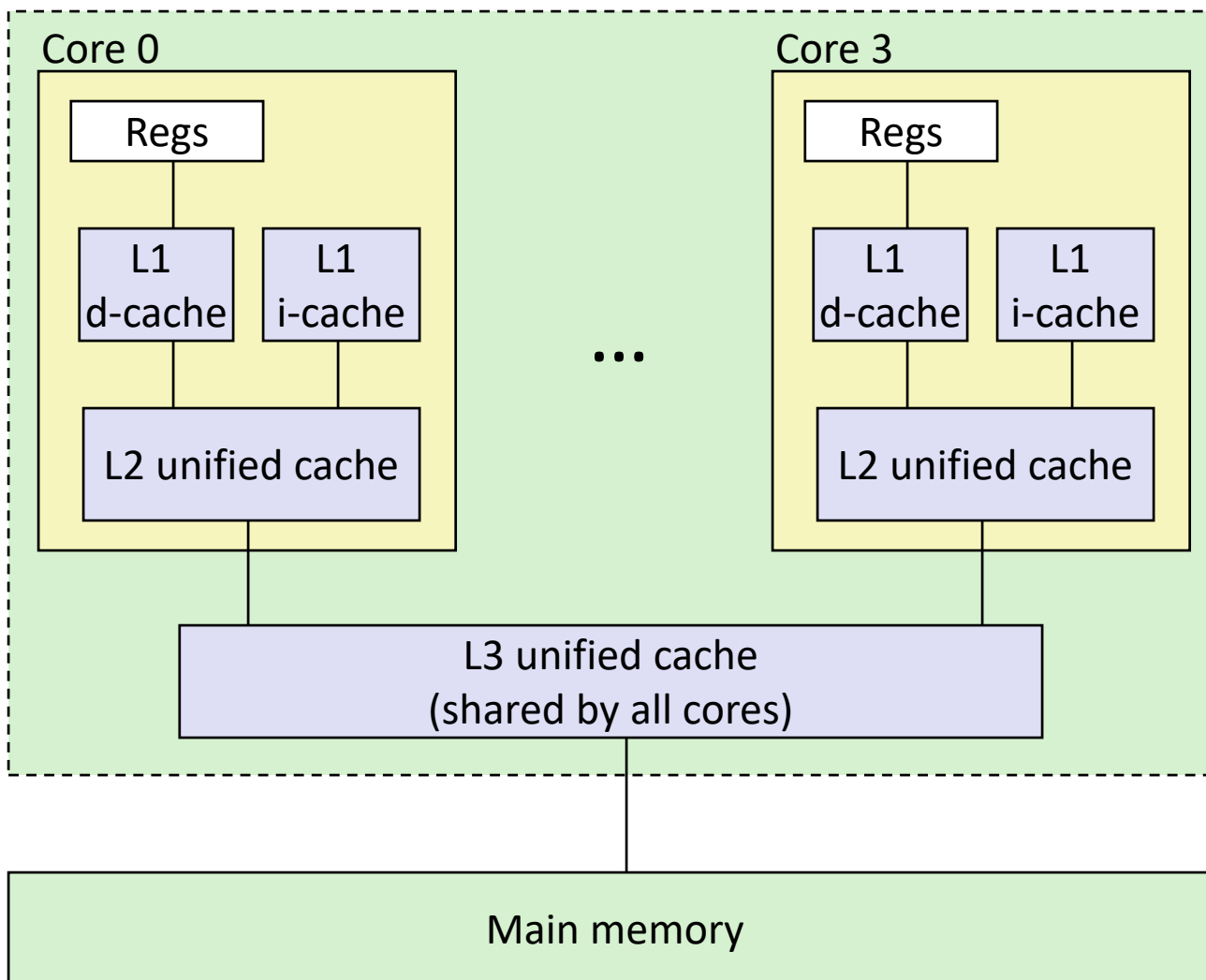


An Example Memory Hierarchy



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

Reading Review

- ❖ Terminology:
 - Memory hierarchy
 - Cache parameters: block size (K), cache size (C)
 - Addresses: block offset field (k bits wide)
 - Cache organization: direct-mapped cache, index field

- ❖ Questions from the Reading?

Review Questions

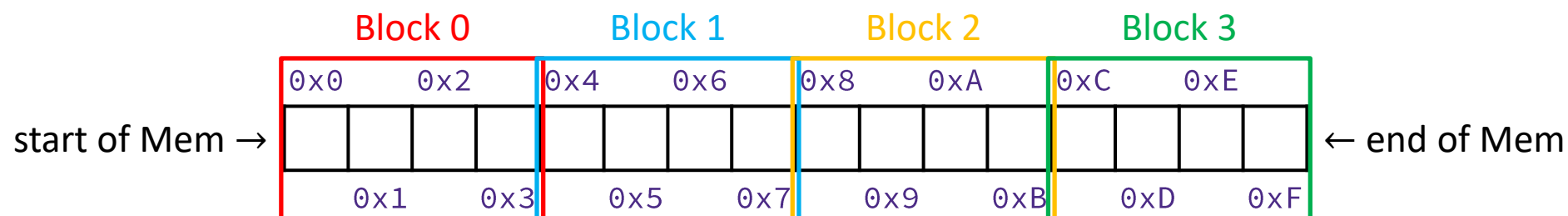
- ❖ We have a direct-mapped cache with the following parameters:
 - Block size of 8 bytes
 - Cache size of 4 KiB

- ❖ How many blocks can the cache hold?
- ❖ How many bits wide is the block offset field?
- ❖ Which of the following addresses would fall under block number 3?
A. 0x3 **B. 0x1F** **C. 0x30** **D. 0x38**

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!
 - Small example ($K = 4$ B):



Cache Organization (1)

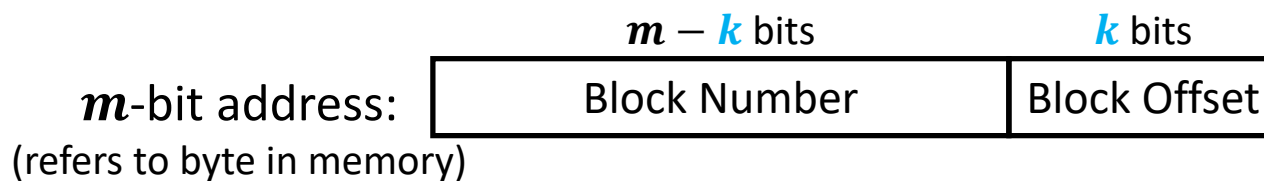
Note: The textbook uses “B” for block size

- ❖ **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)



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!
- ❖ Example:
 - If we have 6-bit addresses and block size $K = 4$ B, which block and byte does 0x15 refer to?

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!

Hash Tables for Fast Lookup

Insert:

5

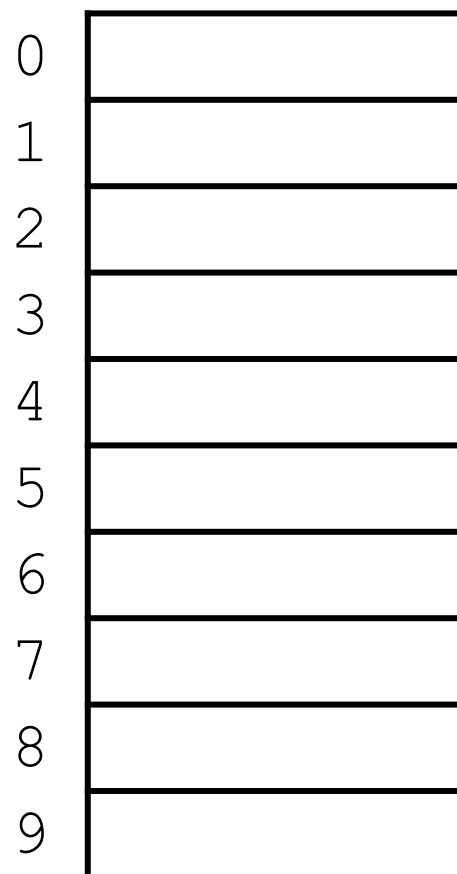
27

34

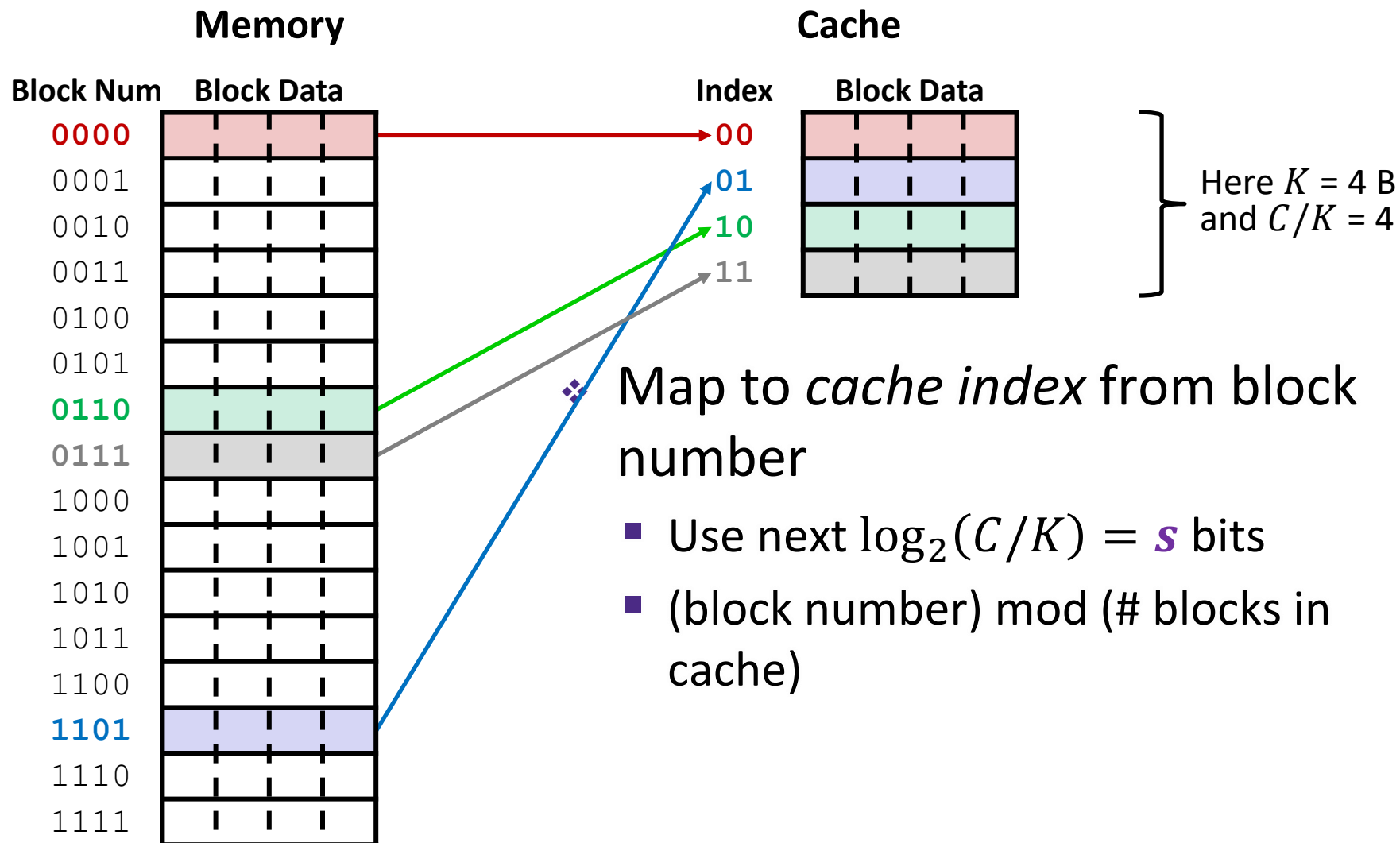
102

119

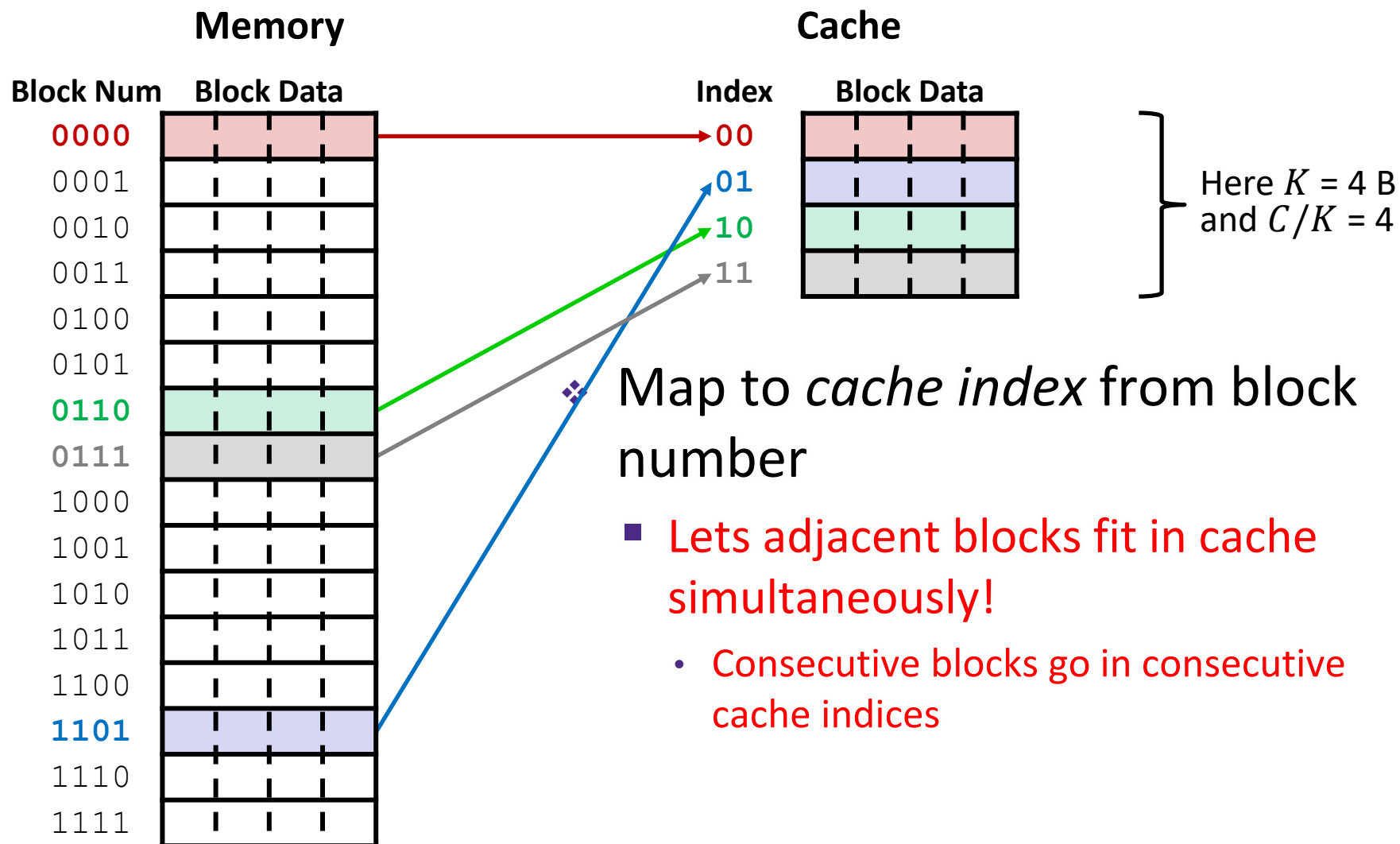
Apply hash function to map data
to “buckets”



Place Data in Cache by Hashing Address



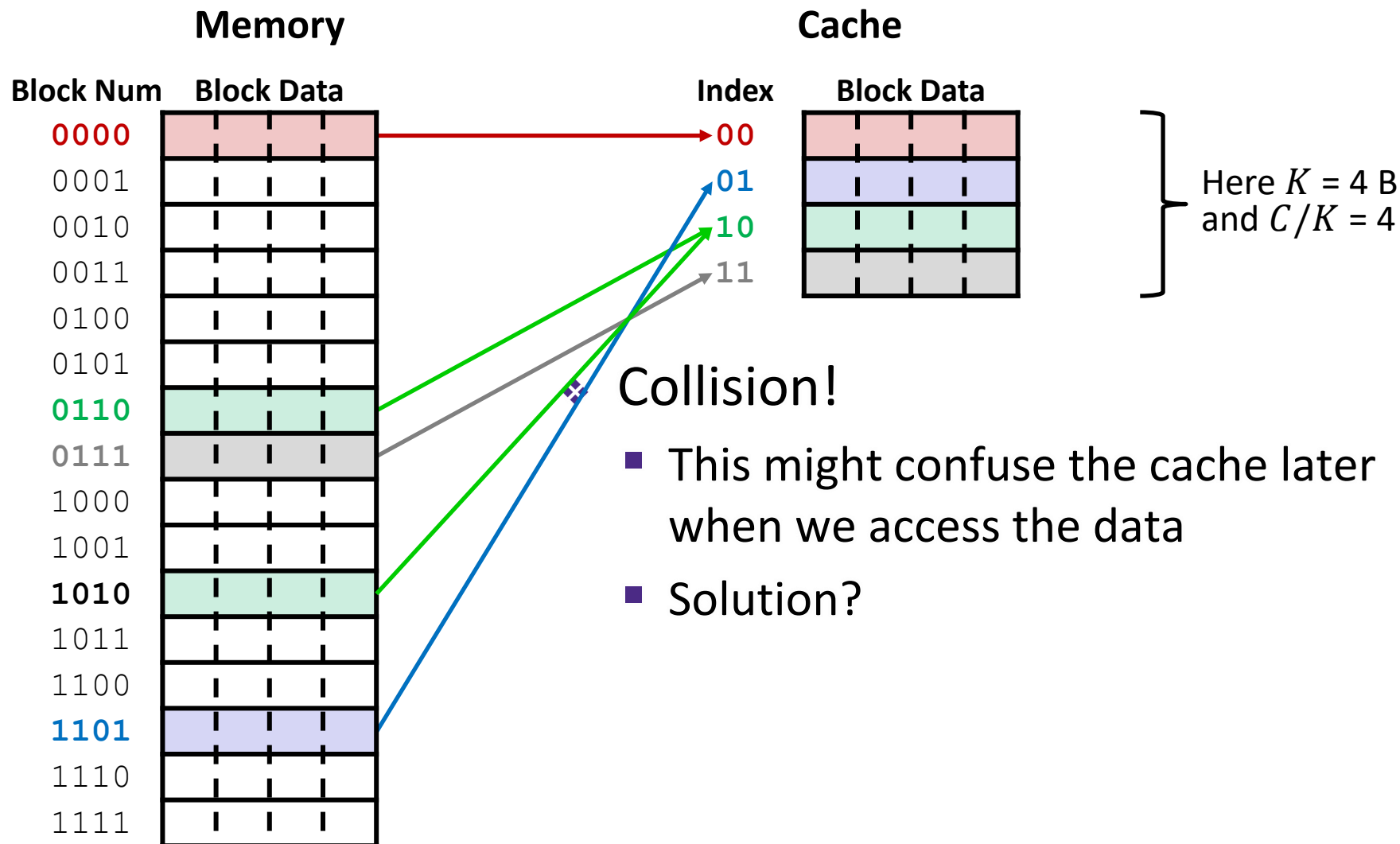
Place Data in Cache by Hashing Address



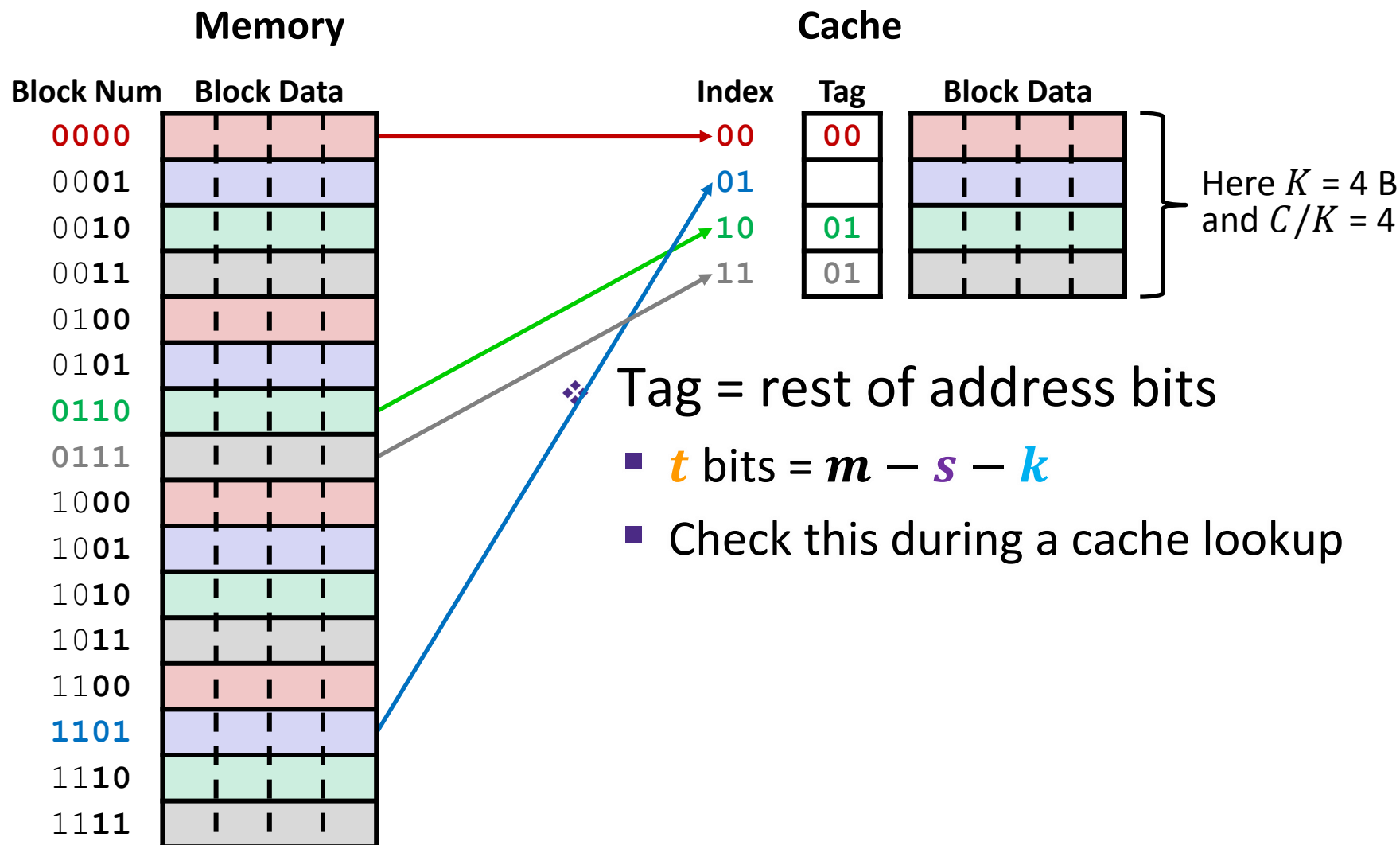
Polling Question

- ❖ 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?
 - Vote on Ed Lessons

Place Data in Cache by Hashing Address



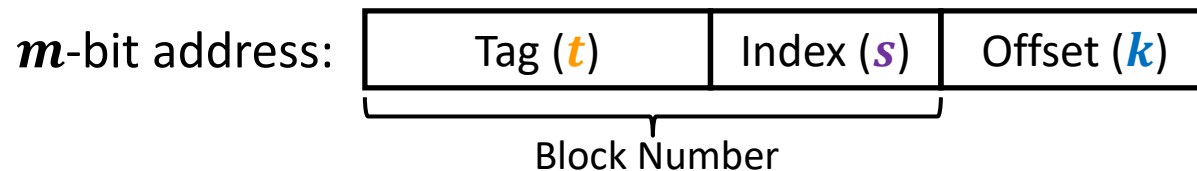
Tags Differentiate Blocks in Same Index



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 their phone number

- ❖ TIO address breakdown:



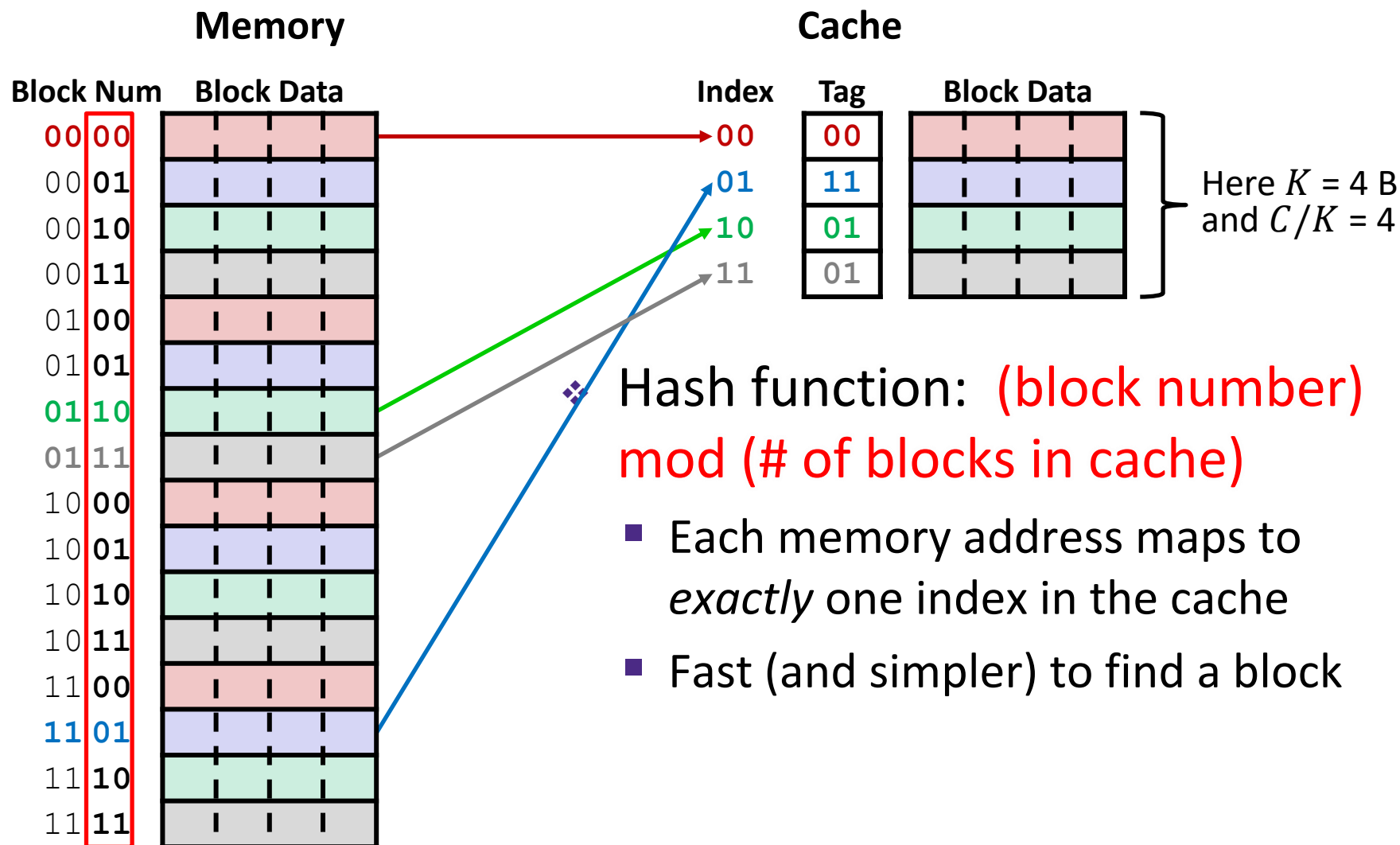
- **Index** field tells you where to look in cache
- **Tag** field lets you check that data is the block you want
- **Offset** field selects specified start byte within block
- **Note:** *t* and *s* sizes will change based on hash function

Cache Puzzle

- ❖ 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...

Summary: Direct-Mapped Cache



Direct-Mapped Cache Problem

