

Caches II

CSE 351 Spring 2019

Instructor:

Ruth Anderson

Teaching Assistants:

Gavin Cai

Jack Eggleston

John Feltrup

Britt Henderson

Richard Jiang

Jack Skalitzy

Sophie Tian

Connie Wang

Sam Wolfson

Casey Xing

Chin Yeoh



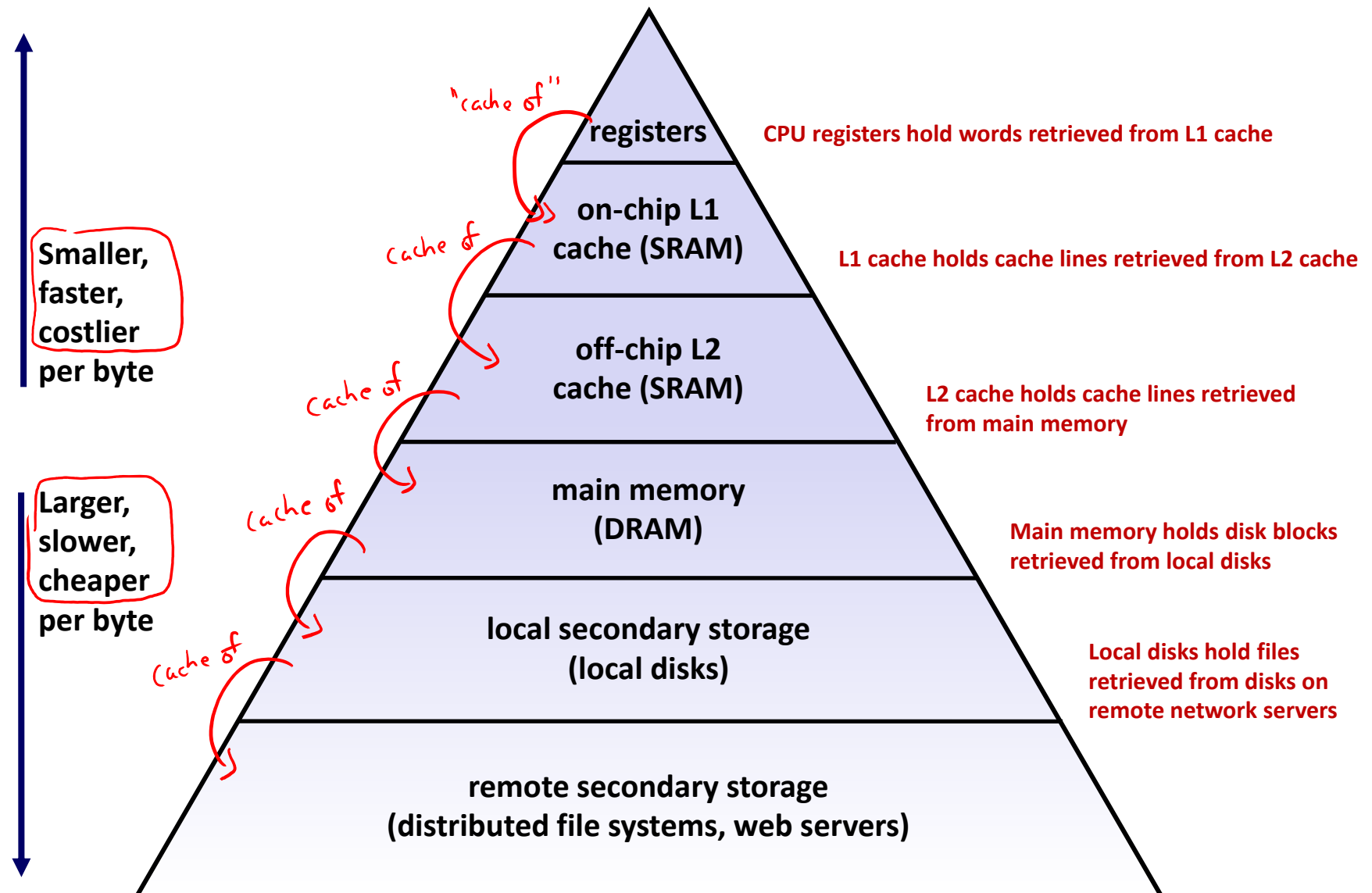
Administrivia

- ❖ Lab 3, due Wednesday (5/15)
- ❖ Homework 4 , due Wed (5/22) (Structs, Caches)
- ❖ Midterm Grading completed
 - You should have received an email from Gradescope
 - Solutions posted on website
 - Rubric and grades will be found on Gradescope
 - Regrade requests will be open for a short time after grade release via Gradescope

Memory Hierarchies

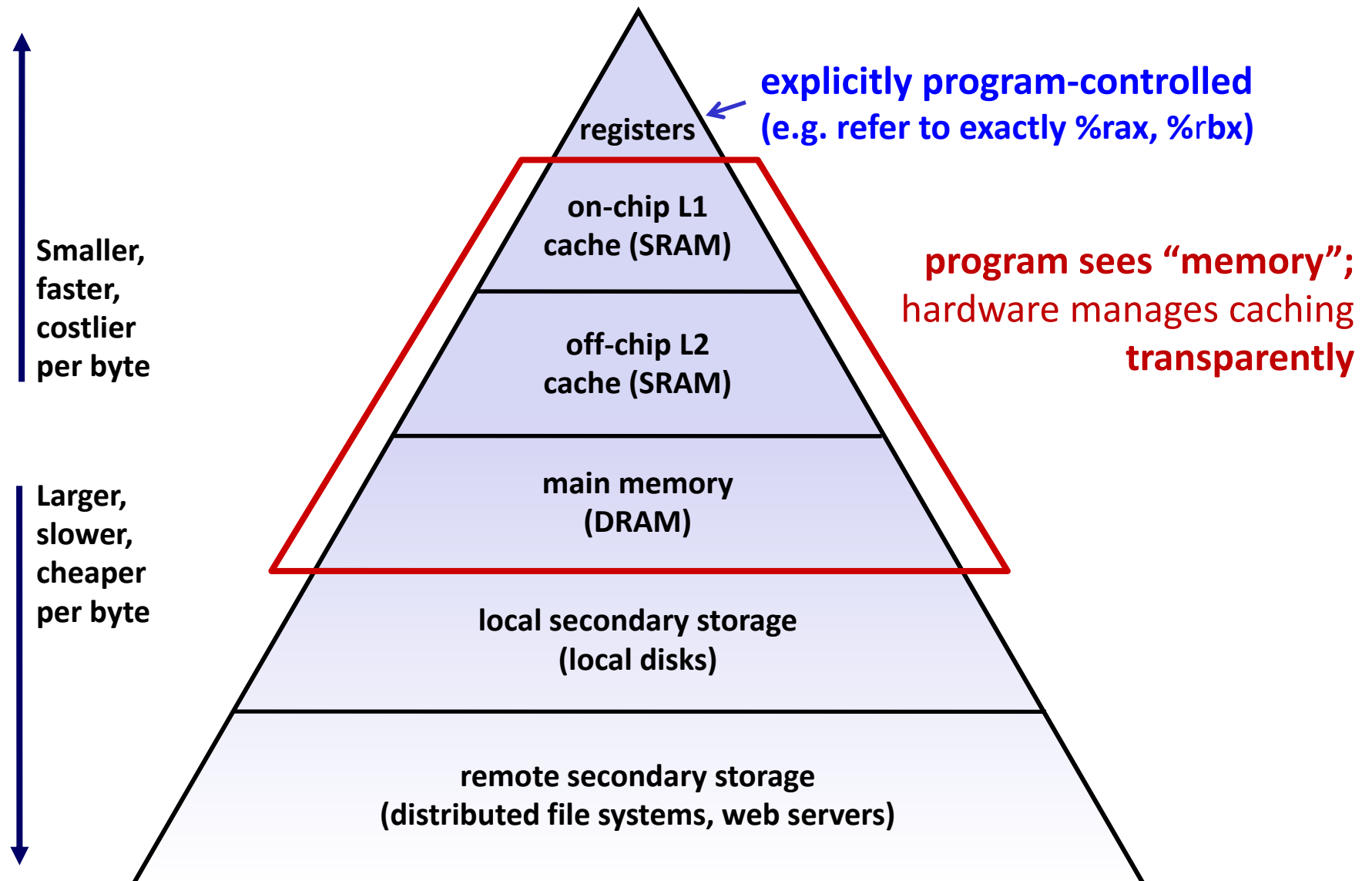
- ❖ Some fundamental and enduring properties of hardware and software systems:
 - Faster storage technologies almost always cost more per byte and have lower capacity
 - The gaps between memory technology speeds are widening
 - True for: registers \leftrightarrow cache, cache \leftrightarrow DRAM, DRAM \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



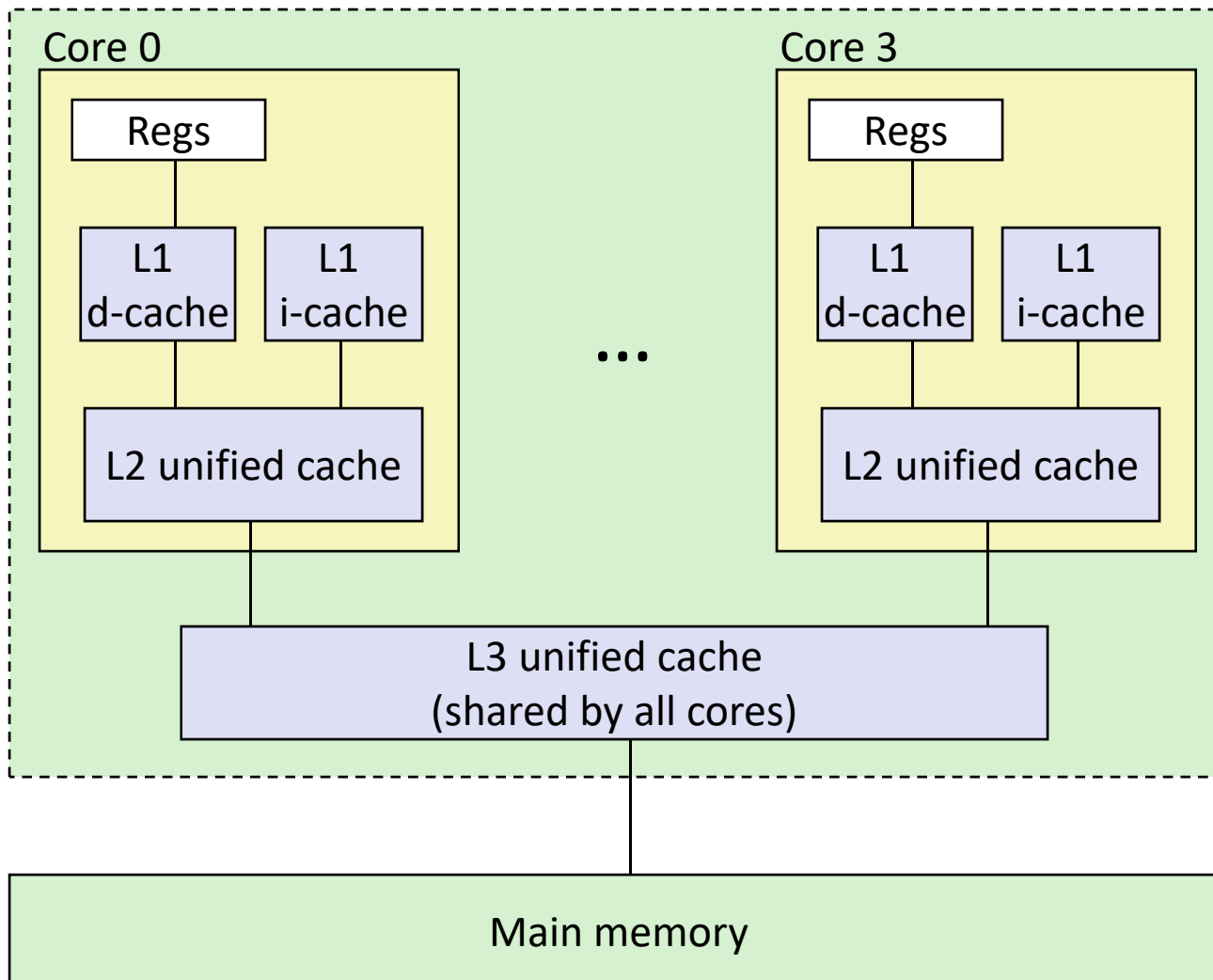
An Example Memory Hierarchy

add %rax(%rbx)



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

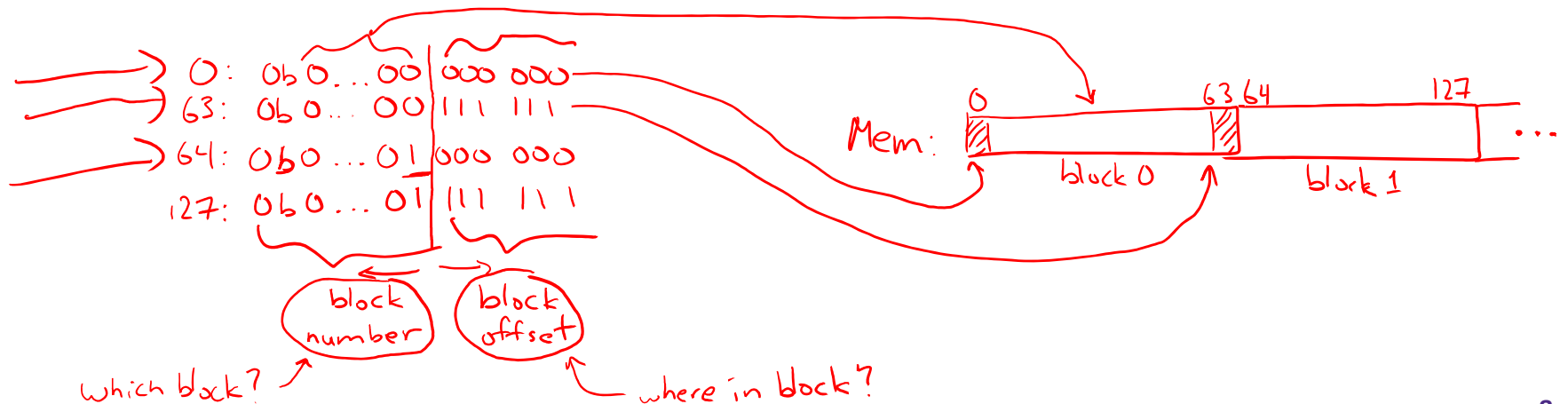
Cache Organization (1)

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!

Lab 1a: within Same Block



Cache Organization (1)

Note: The textbook uses “b” for offset bits

❖ Block Size (K): unit of transfer between \$ and Mem

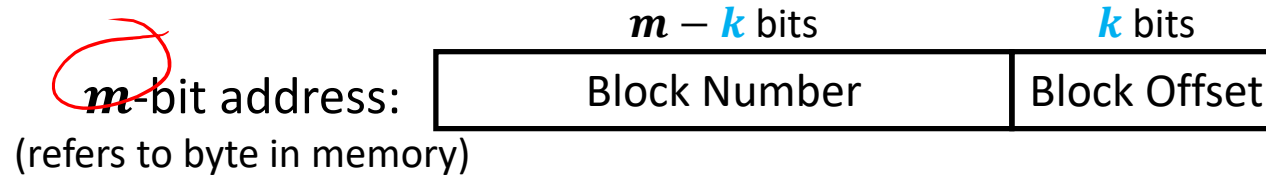
- Given in bytes and always a power of 2 (e.g. 64 B)
- Blocks consist of adjacent bytes (differ in address by 1)
 - Spatial locality!

❖ 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)

64 bytes
↓
6

How many bits do I need to specify every byte in a block?



Peer Instruction 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/rea>

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

2 bits

address: 0x 1 5

0b 0 1 0 1 / 0 1

block num (value 5) offset (value 1)

offset width = $\log_2(K) = \log_2(4) = 2 \text{ bits}$

0x15

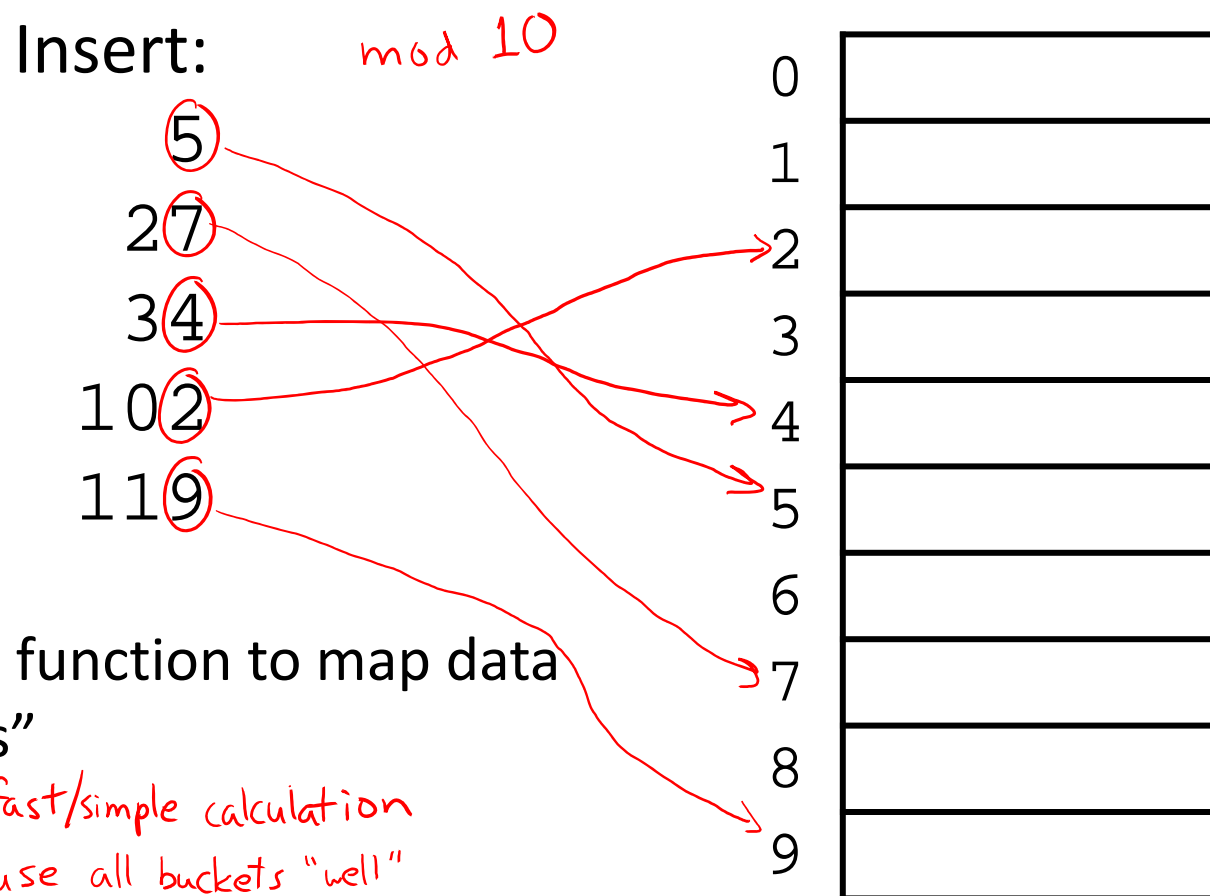
block number 5:

offset: 00 01 10 11

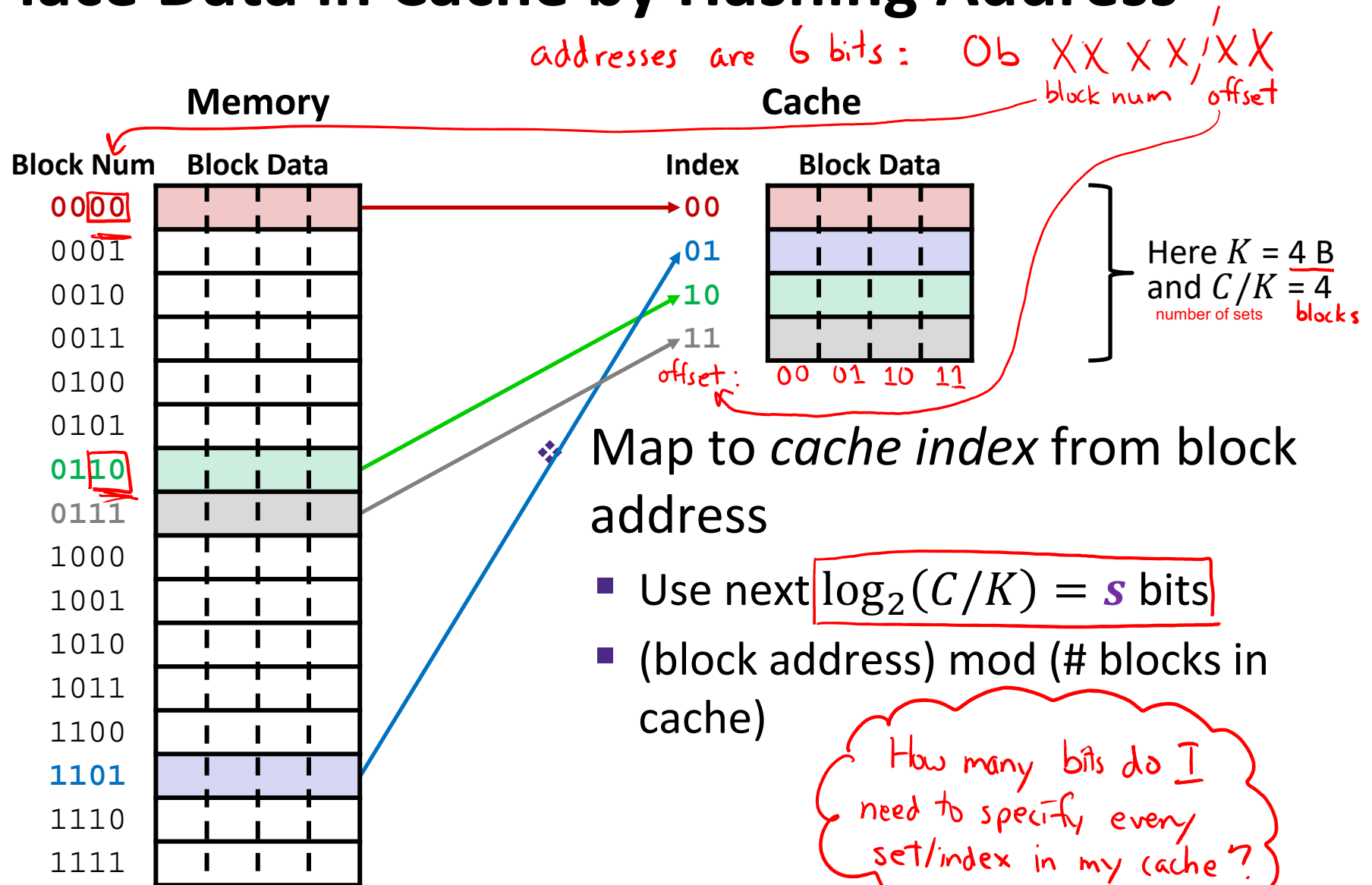
Cache Organization (2)

- ❖ **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: $C = 32 \text{ KiB} = 512 \text{ blocks}$ if using 64-B blocks
$$2^5 \times 2^{10} = 2^{15} \text{ B} \times \frac{1 \text{ block}}{2^6 \text{ B}} = 2^9 \text{ blocks}$$
- ❖ 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!

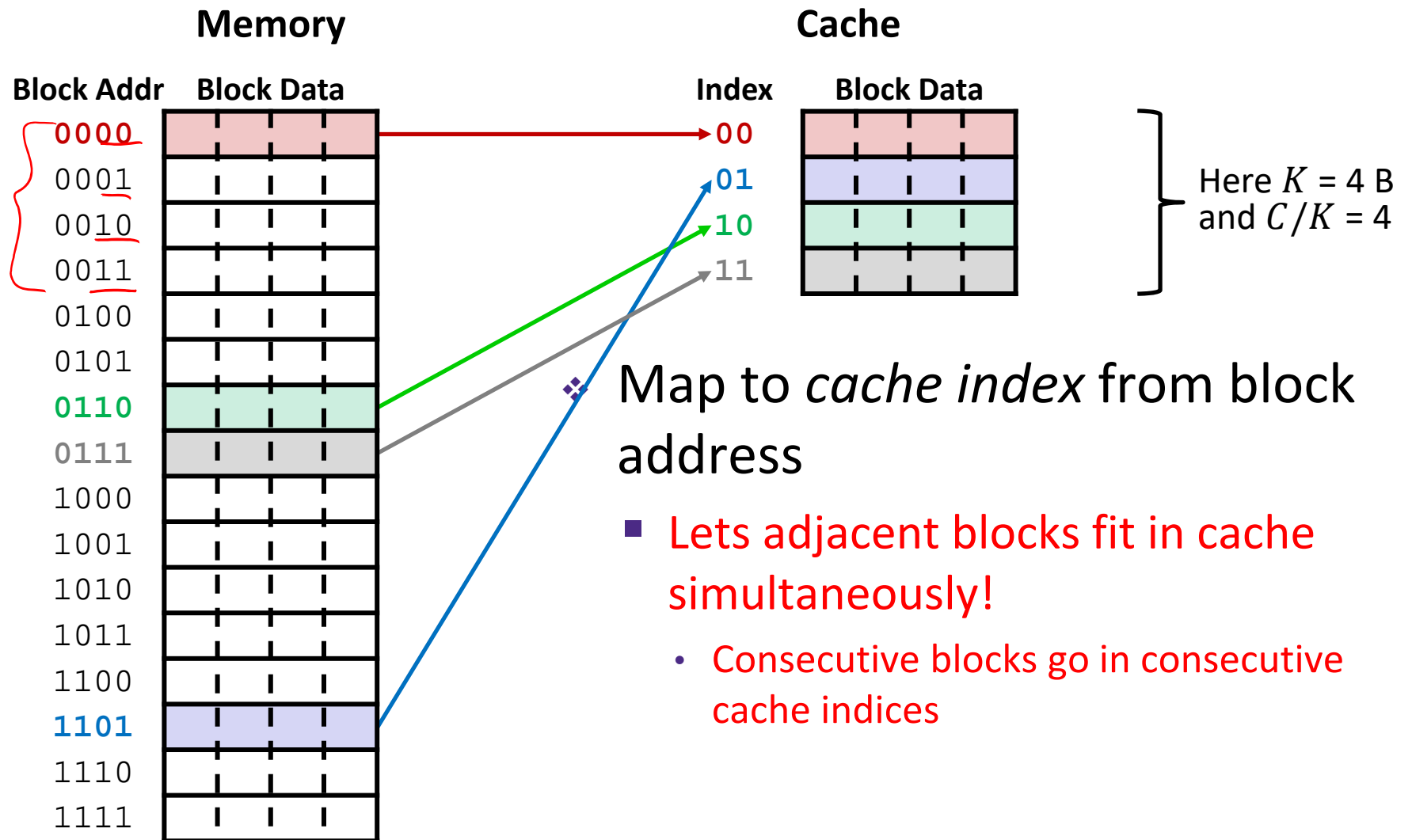
Review: Hash Tables for Fast Lookup



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

2 bits for block offset, 2 blocks for set

10 1010 --> has offset 2 which means that there are two entries before it in the block and one after.

address: 0x 0b ²10 / ^A10 / 10
 index offset
 (value 2) (value 2)
 offset

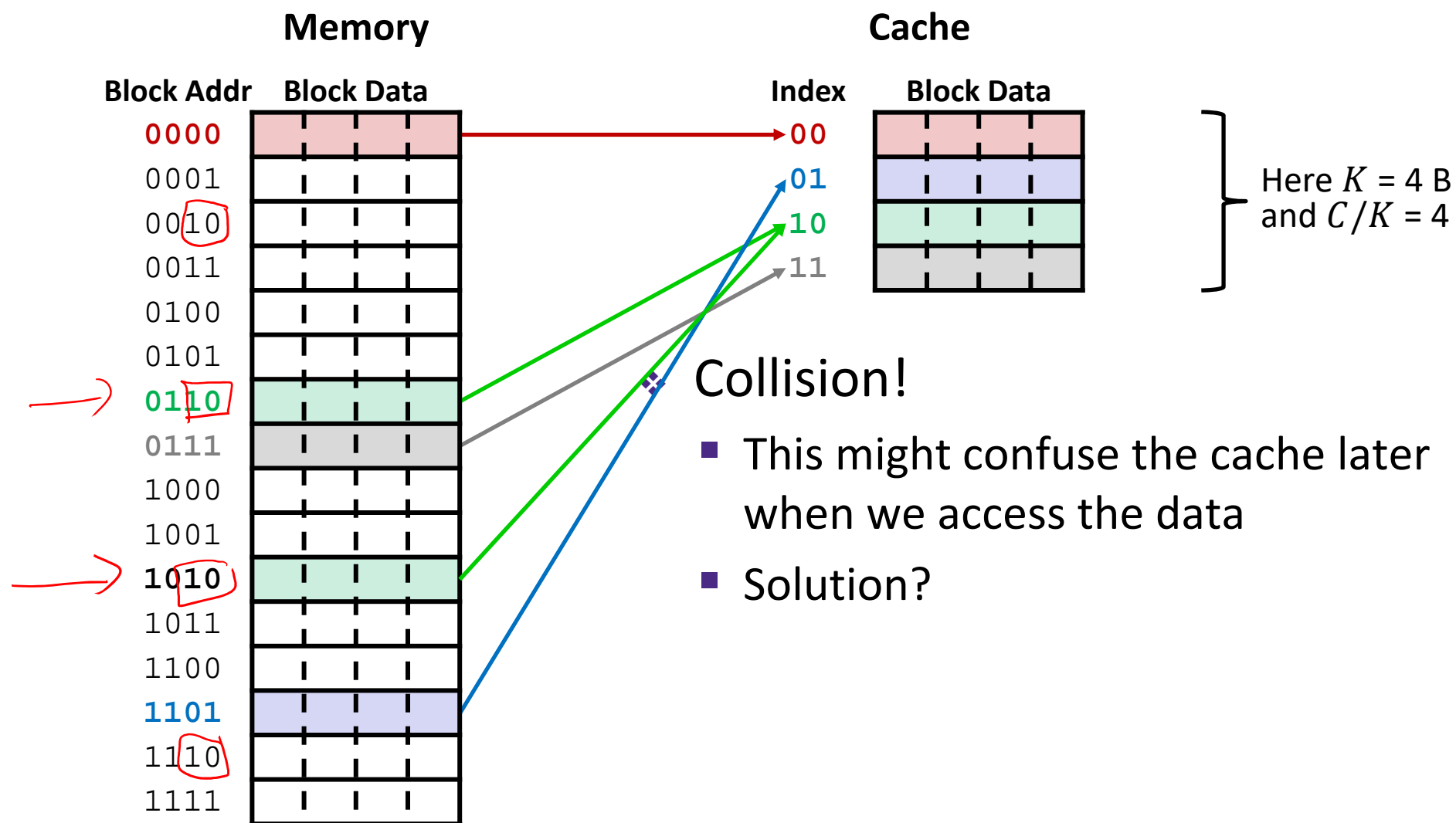
cache: Index 00 01 10 11
 00
 01
 → 10
 11

28	29	2A	2B
↓	↓	↓	↓

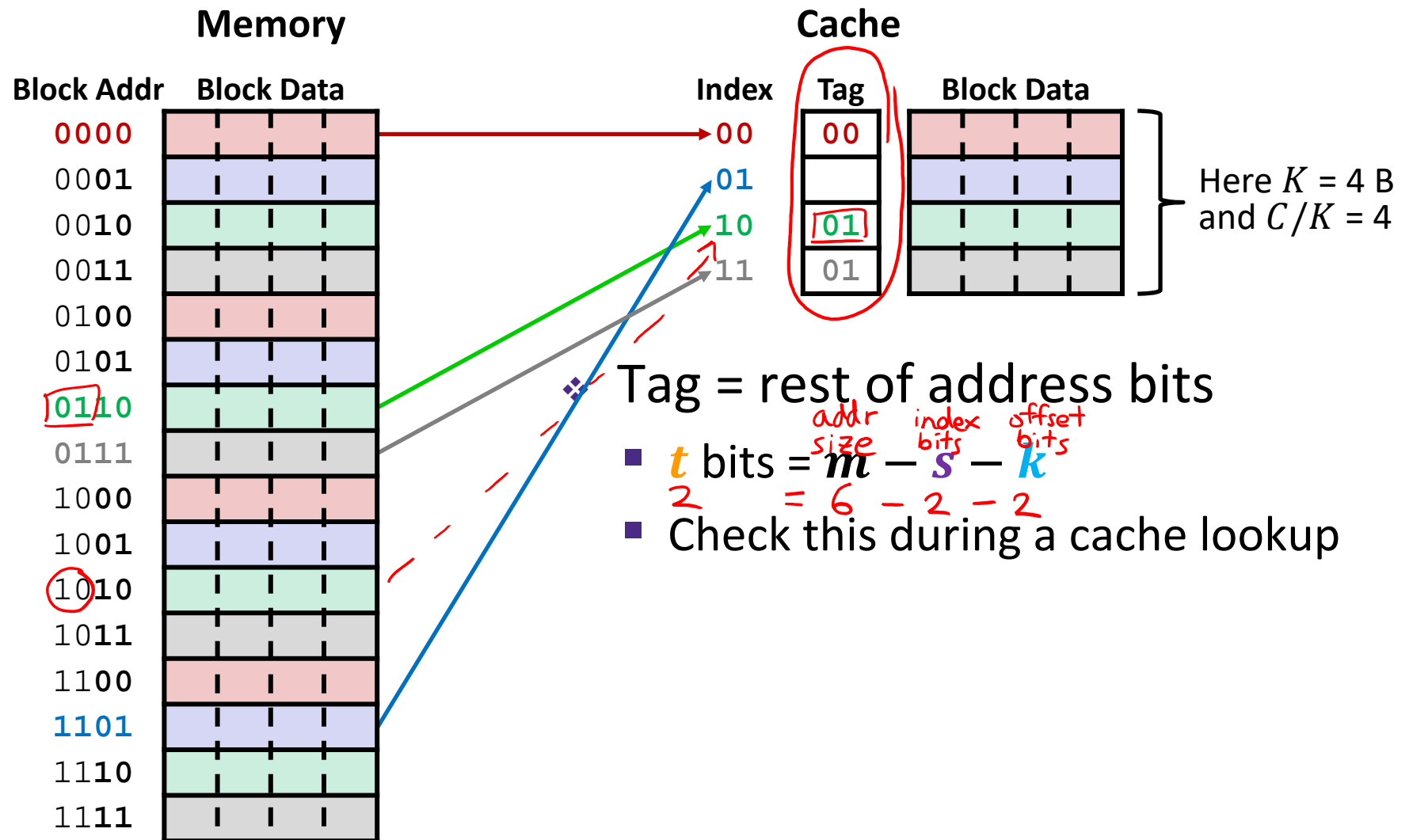
Index 2 (0b10)

along with: 0b1010 00 = 0x28
 01 = 0x29
 11 = 0x2B

Place Data in Cache by Hashing Address



Tags Differentiate Blocks in Same Index

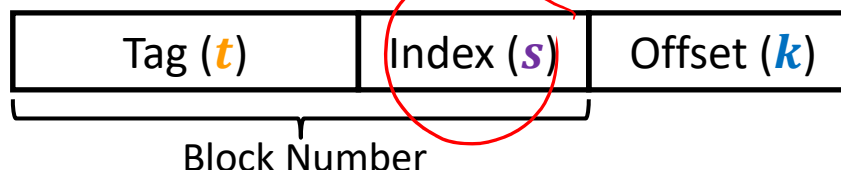


Checking for a Requested Address

- ❖ 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:

m -bit address:



- 1 ■ **Index** field tells you where to look in cache which set the data WOULD be stored in
- 2 ■ **Tag** field lets you check that data is the block you want whether the data you are looking for is currently stored in that set
- 3 ■ **Offset** field selects specified start byte within block where in a given block the data is stored in *or line*

- **Note:** t and s sizes will change based on hash function

Cache Puzzle

Vote at <http://pollev.com/rea>

- ❖ 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:

hit: block with data already in \$
miss: data not in \$, pulls block containing data from Mem

Addresses
in
Decimal

- (14: miss), (15: hit), (16: miss)

A. **4 bytes** ✓

B. **8 bytes** ✓

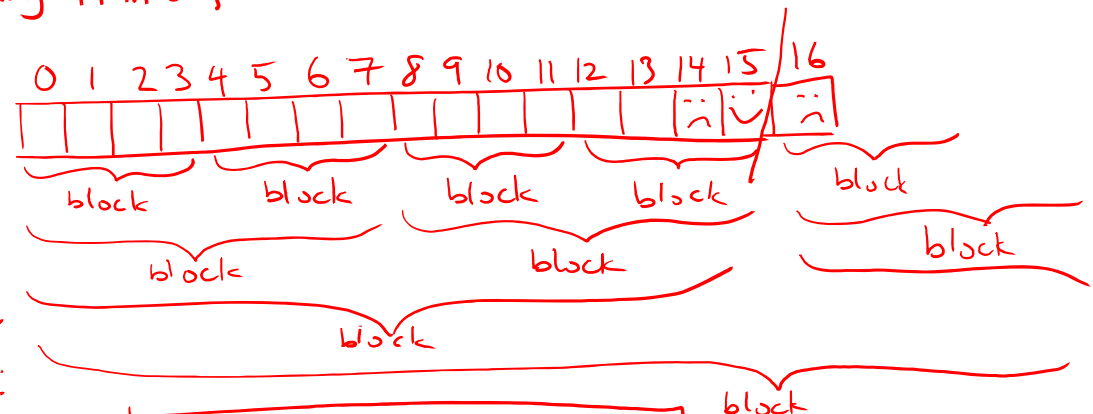
C. **16 bytes** ✓

D. **32 bytes**

E. We're lost...

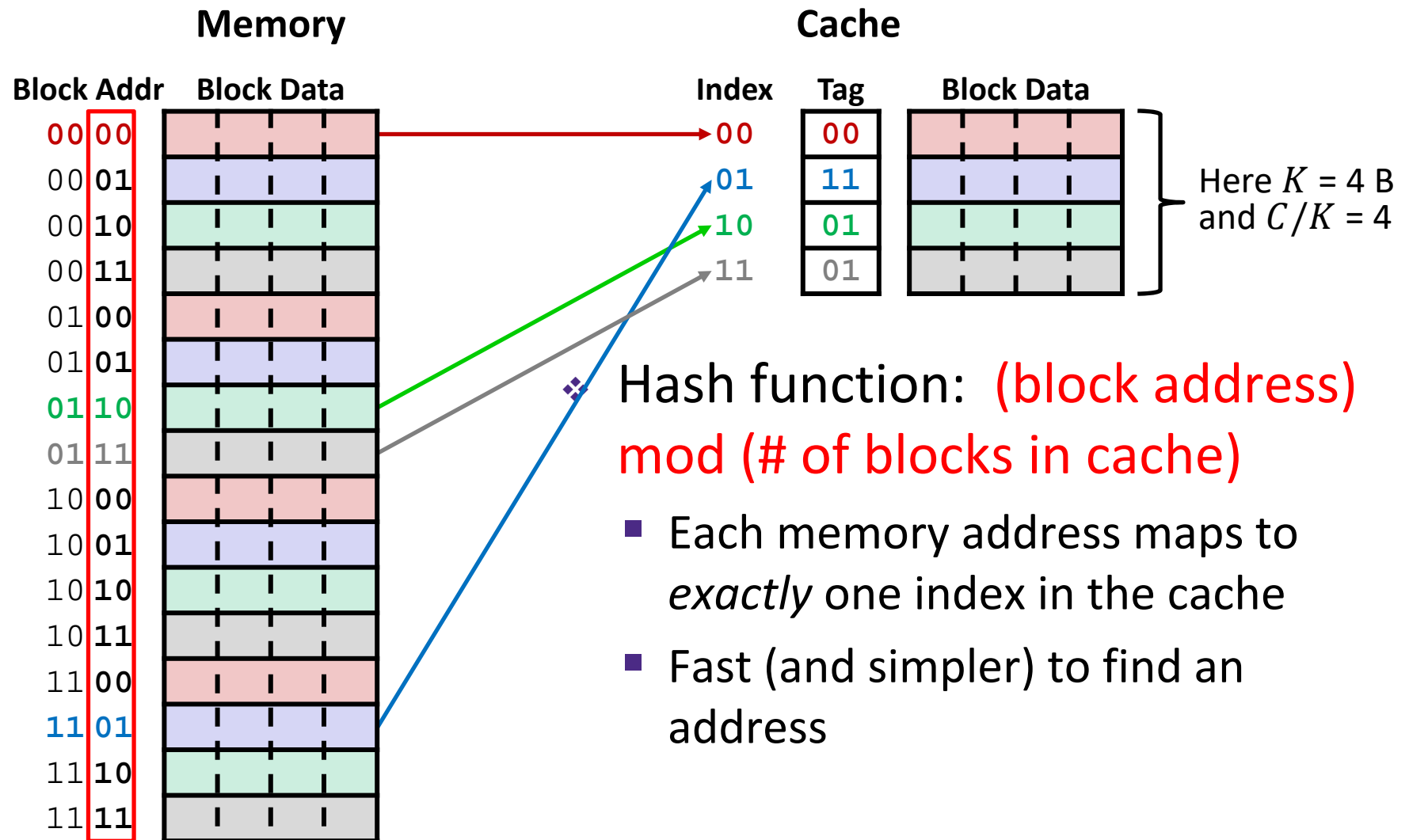
① pulls block containing 14 into \$
 ② 14 & 15 are in the same block
 ③ 16 is in a different block

byte addr:

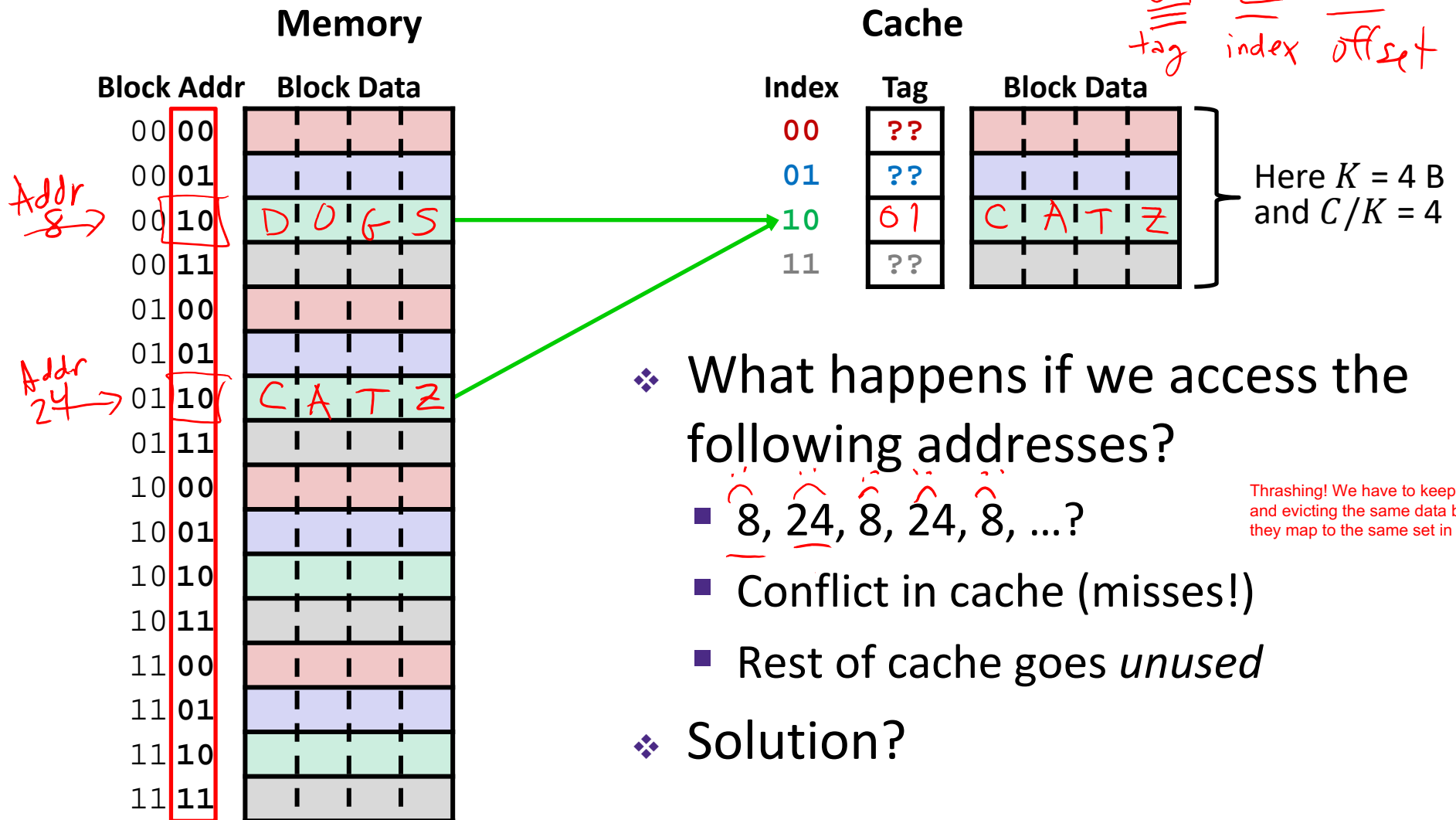


$K_{\min} = 2 \text{ B}, K_{\max} = 16 \text{ B}$

Direct-Mapped Cache



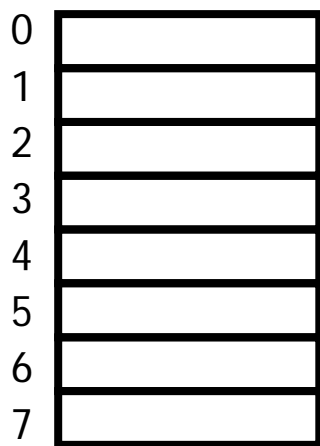
Direct-Mapped Cache Problem



Associativity

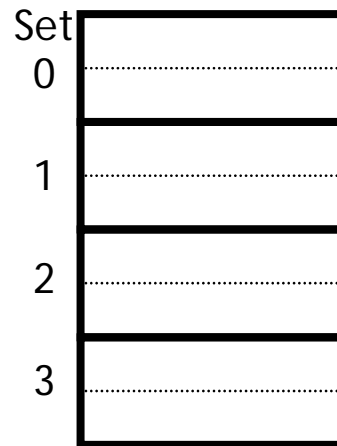
- ❖ What if we could store data in any place in the cache?
 - More complicated hardware = more power consumed, slower
- ❖ So we *combine* the two ideas:
 - Each address maps to exactly one **set**
 - Each set can store block in more than one **way**

3 bit index
1-way:
8 sets,
1 block each

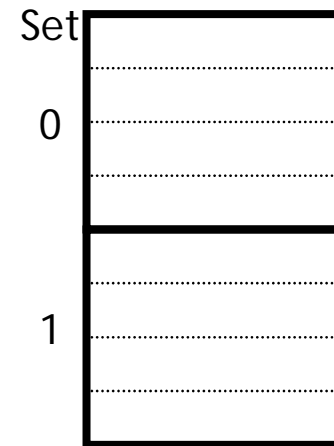


direct mapped

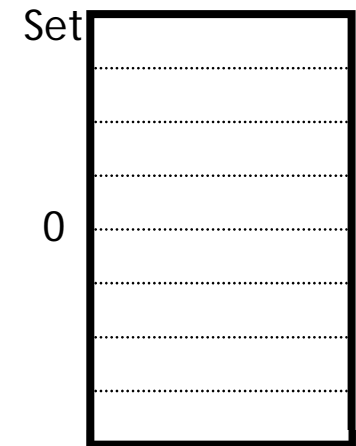
2 bit index
2-way:
4 sets,
2 blocks each



1 bit
4-way:
2 sets,
4 blocks each



0 bits
8-way:
1 set,
8 blocks



fully associative