Caches II
CSE 351 Winter 2020

Instructor:

Ruth Anderson

Teaching Assistants:

Jonathan Chen

Justin Johnson

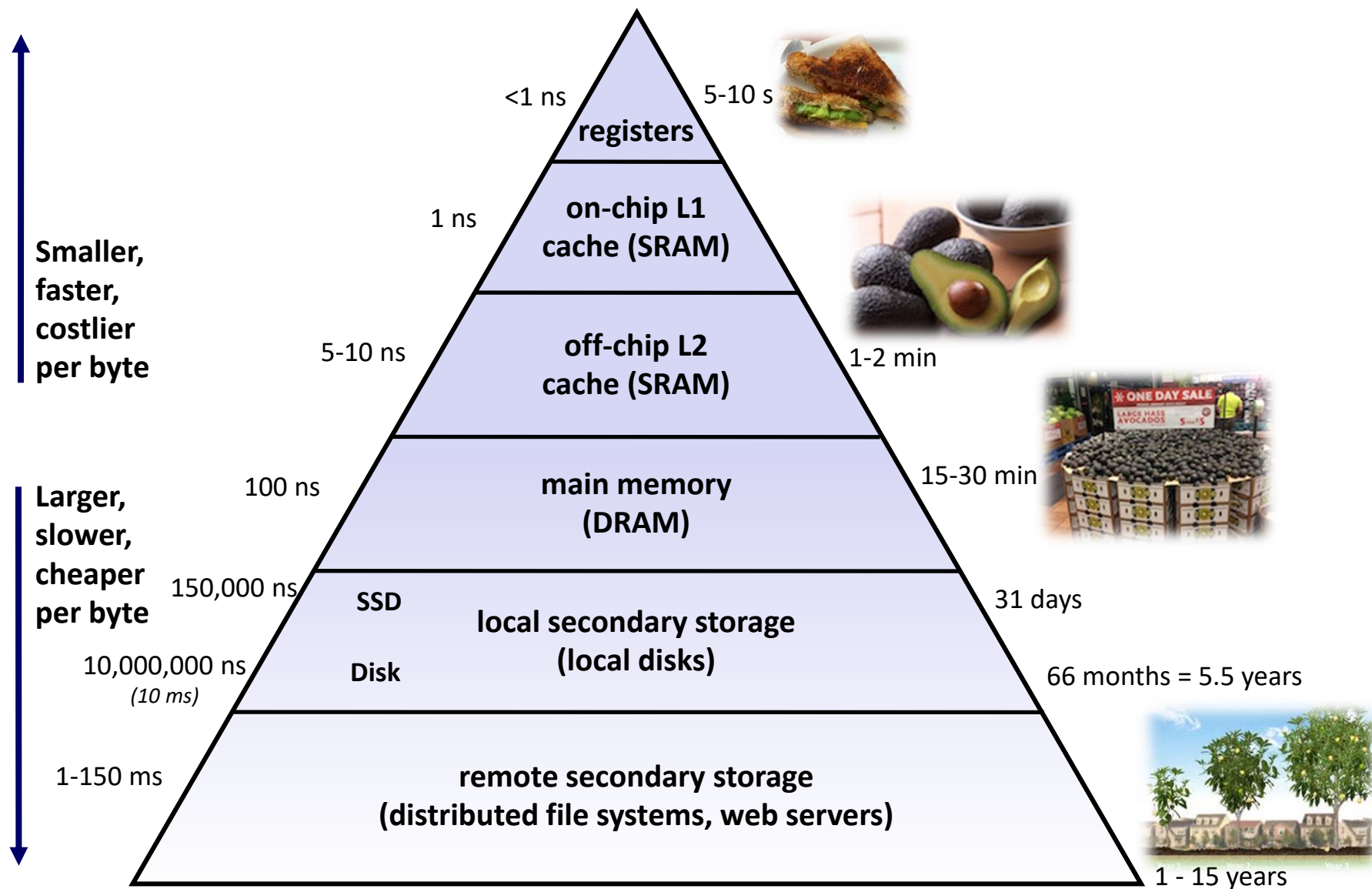Porter Jones

Josie Lee

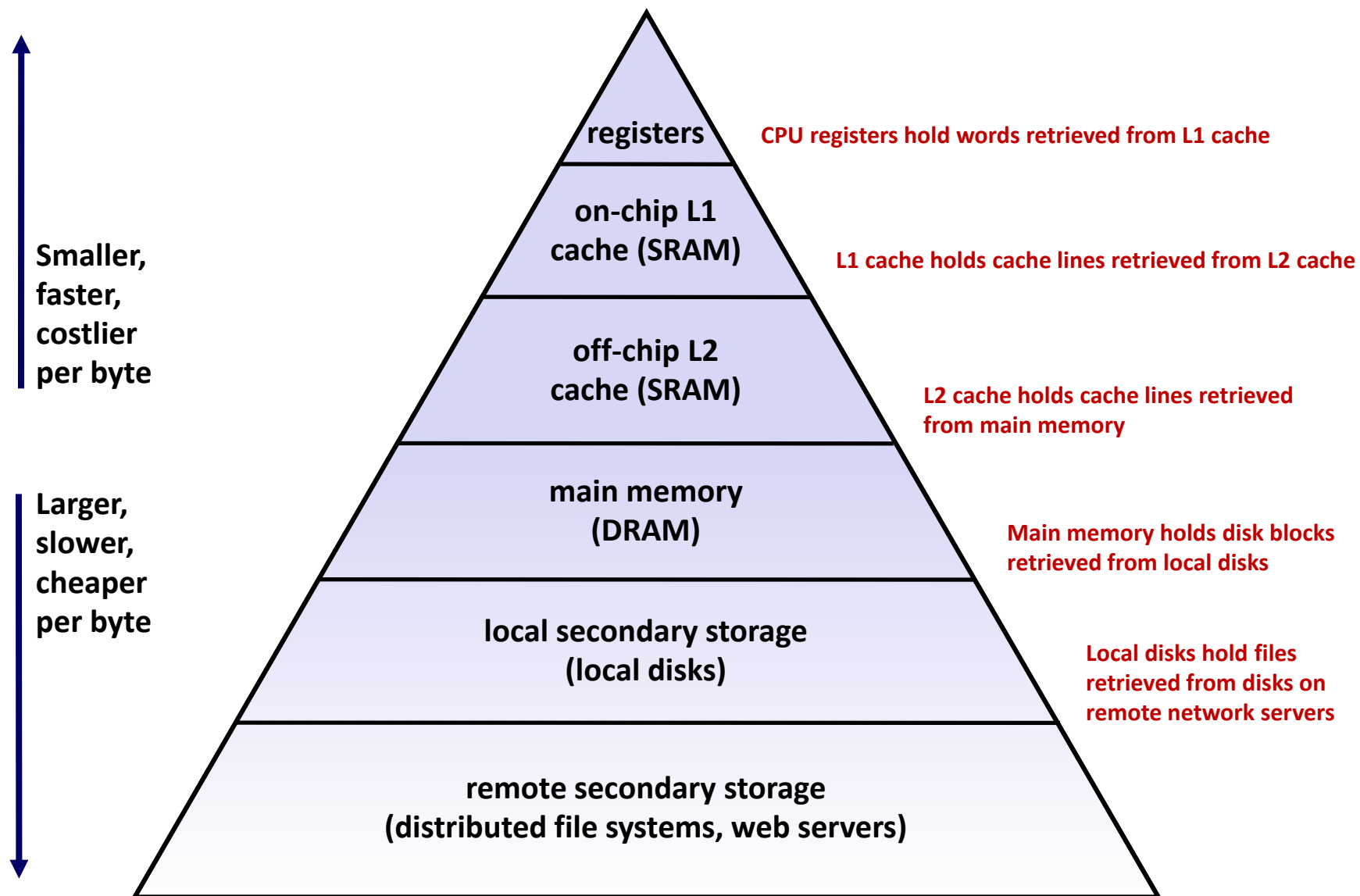Jeffery Tian

Callum Walker

Eddy (Tianyi) Zhou

An Example Memory Hierarchy

**Smaller,
faster,
costlier
per byte**

**Larger,
slower,
cheaper
per byte**

<1 ns — registers — 5-10 s

1 ns — on-chip L1 cache (SRAM)

5-10 ns — off-chip L2 cache (SRAM) — 1-2 min

100 ns — main memory (DRAM) — 15-30 min

150,000 ns — SSD — local secondary storage (local disks) — 31 days

10,000,000 ns (10 ms) — Disk — 66 months = 5.5 years

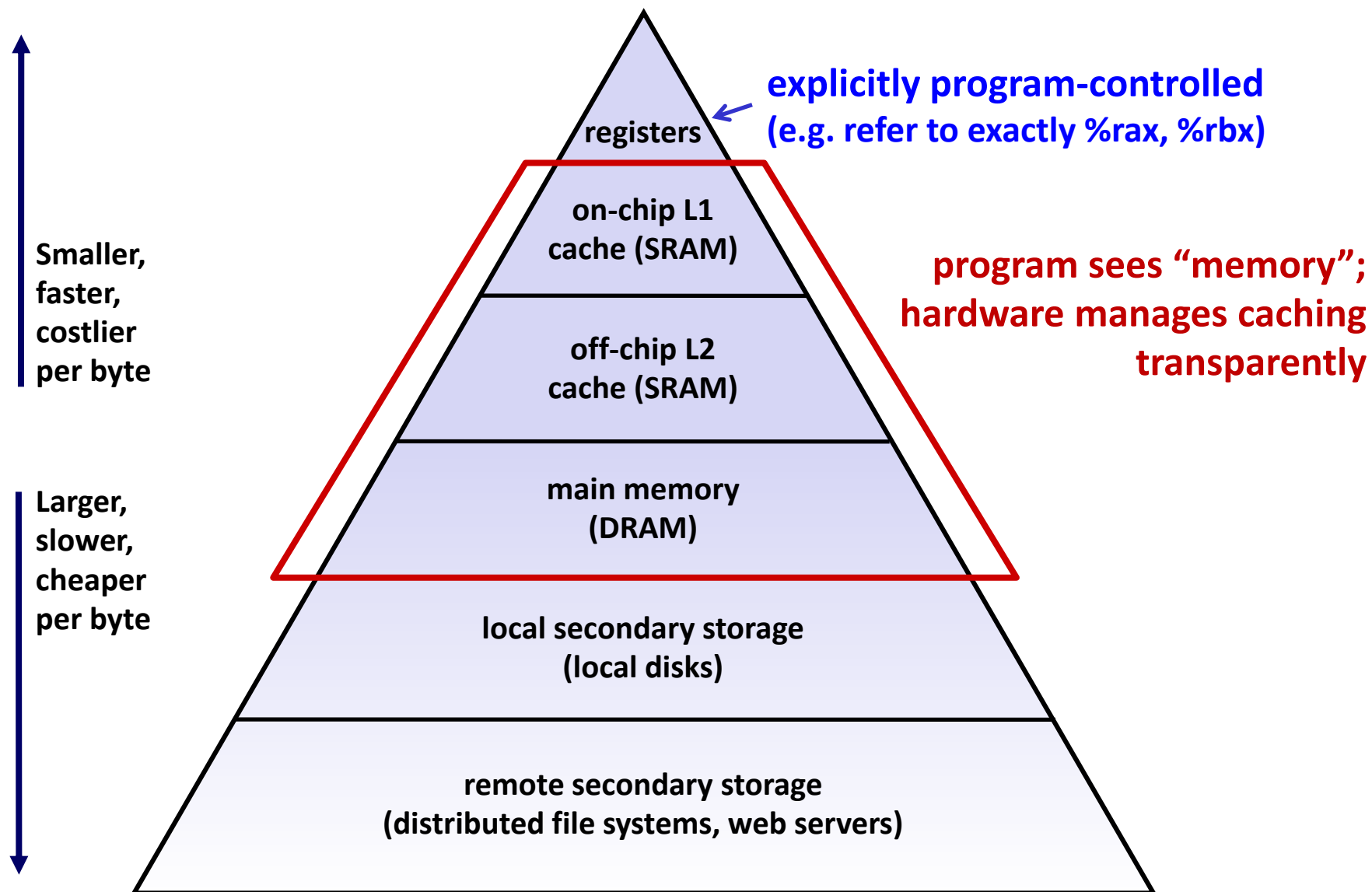1-150 ms — remote secondary storage (distributed file systems, web servers) — 1 - 15 years

2

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 ⟷ cache, cache ⟷ DRAM, DRAM ⟷ 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

Smaller, faster, costlier per byte

Larger, slower, cheaper per byte

registers

CPU registers hold words retrieved from L1 cache

on-chip L1 cache (SRAM)

L1 cache holds cache lines retrieved from L2 cache

off-chip L2 cache (SRAM)

L2 cache holds cache lines retrieved from main memory

main memory (DRAM)

Main memory holds disk blocks retrieved from local disks

local secondary storage (local disks)

Local disks hold files retrieved from disks on remote network servers

remote secondary storage (distributed file systems, web servers)

An Example Memory Hierarchy

**Smaller,
faster,
costlier
per byte**

registers

**on-chip L1
cache (SRAM)**

**off-chip L2
cache (SRAM)**

**main memory
(DRAM)**

**local secondary storage
(local disks)**

**Larger,
slower,
cheaper
per byte**

**remote secondary storage
(distributed file systems, web servers)**

**explicitly program-controlled
(e.g. refer to exactly %rax, %rbx)**

**program sees "memory";
hardware manages caching
transparently**

Intel Core i7 Cache Hierarchy

Processor package

Core 0

Regs

L1 d-cache   L1 i-cache

L2 unified cache

...

Core 3

Regs

L1 d-cache   L1 i-cache

L2 unified cache

L3 unified cache
(shared by all cores)

Main memory
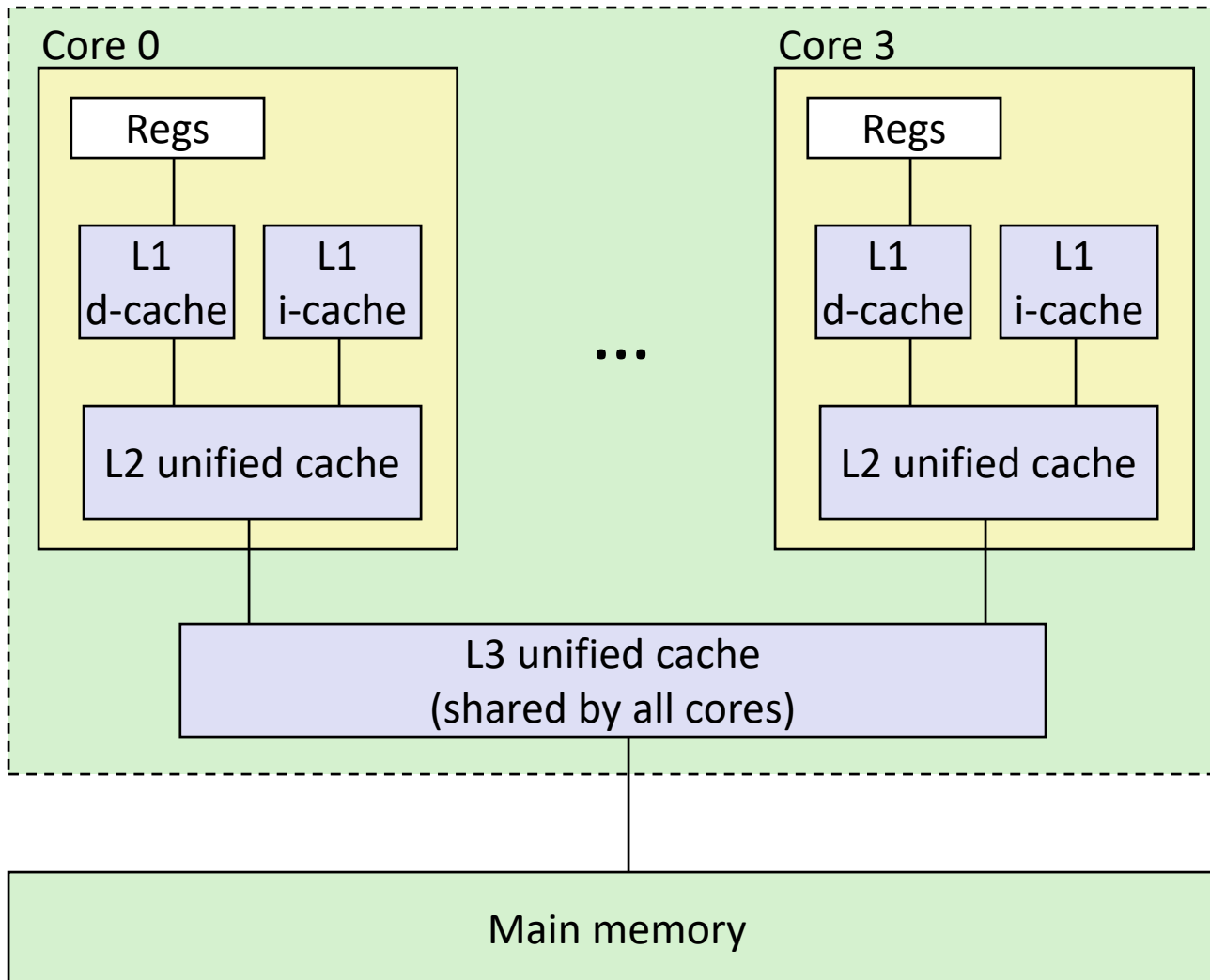
Block size:
64 bytes for all caches

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

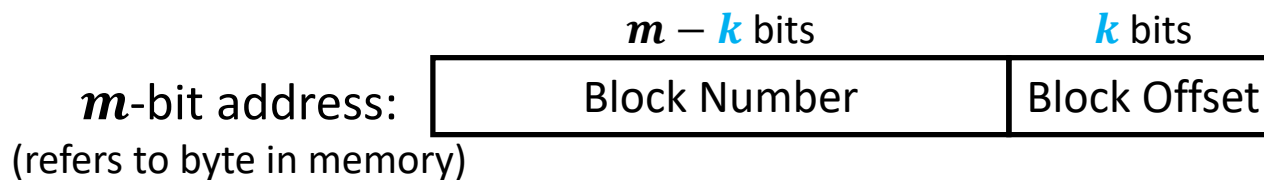❖ Program optimizations that consider caches

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 $\log_2(K) = \boldsymbol{k}$ 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)

	$\boldsymbol{m} - \boldsymbol{k}$ bits	$\boldsymbol{k}$ bits
$\boldsymbol{m}$-bit address: (refers to byte in memory)	Block Number	Block Offset

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/rea

	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)
- <u>Example</u>: 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!

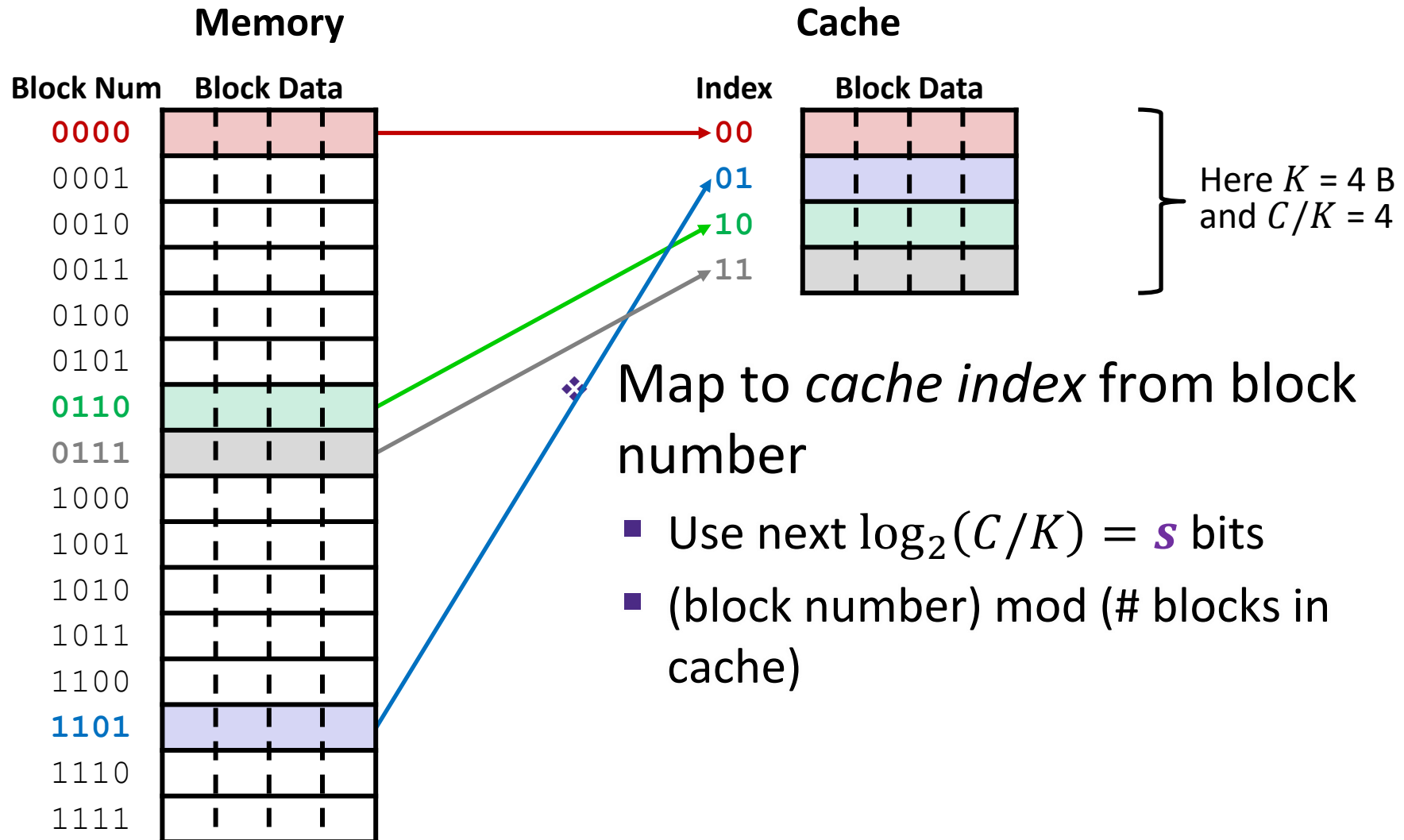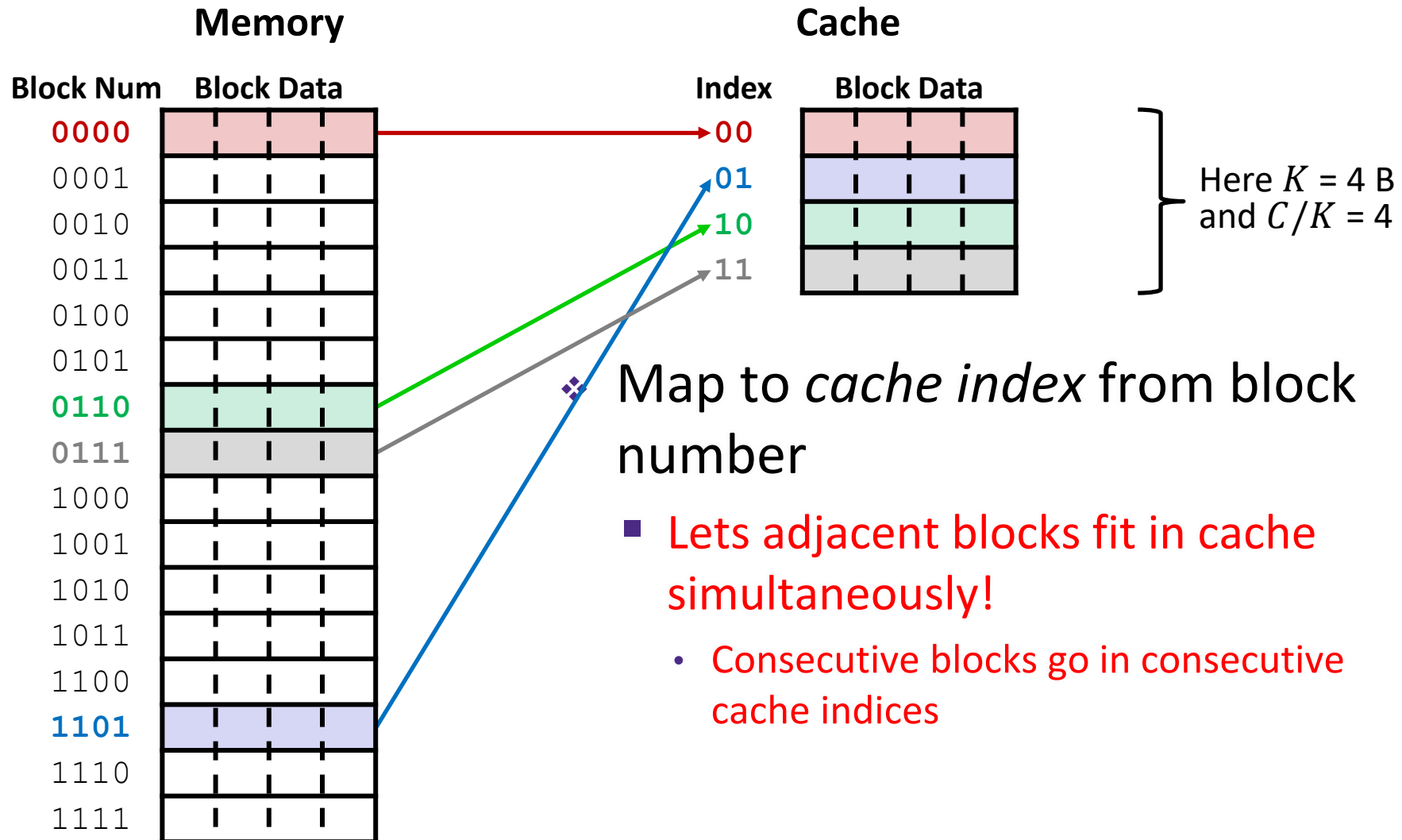# Review: Hash Tables for Fast Lookup

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

Memory   **Cache**

Block Num   **Block Data**   **Index**   **Block Data**

0000		
0001		
0010		
0011		
0100		
0101		
0110		
0111		
1000		
1001		
1010		
1011		
1100		
1101		
1110		
1111		

00
01
10
11

Here K = 4 B and C/K = 4

❖ Map to *cache index* from block number

- Use next $\log_2(C/K) = s$ bits
- (block number) mod (# blocks in cache)

13

Place Data in Cache by Hashing Address

Memory   **Cache**

Block Num   **Block Data**   **Index**   **Block Data**

Here K = 4 B
and C/K = 4

❖ Map to *cache index* from block number

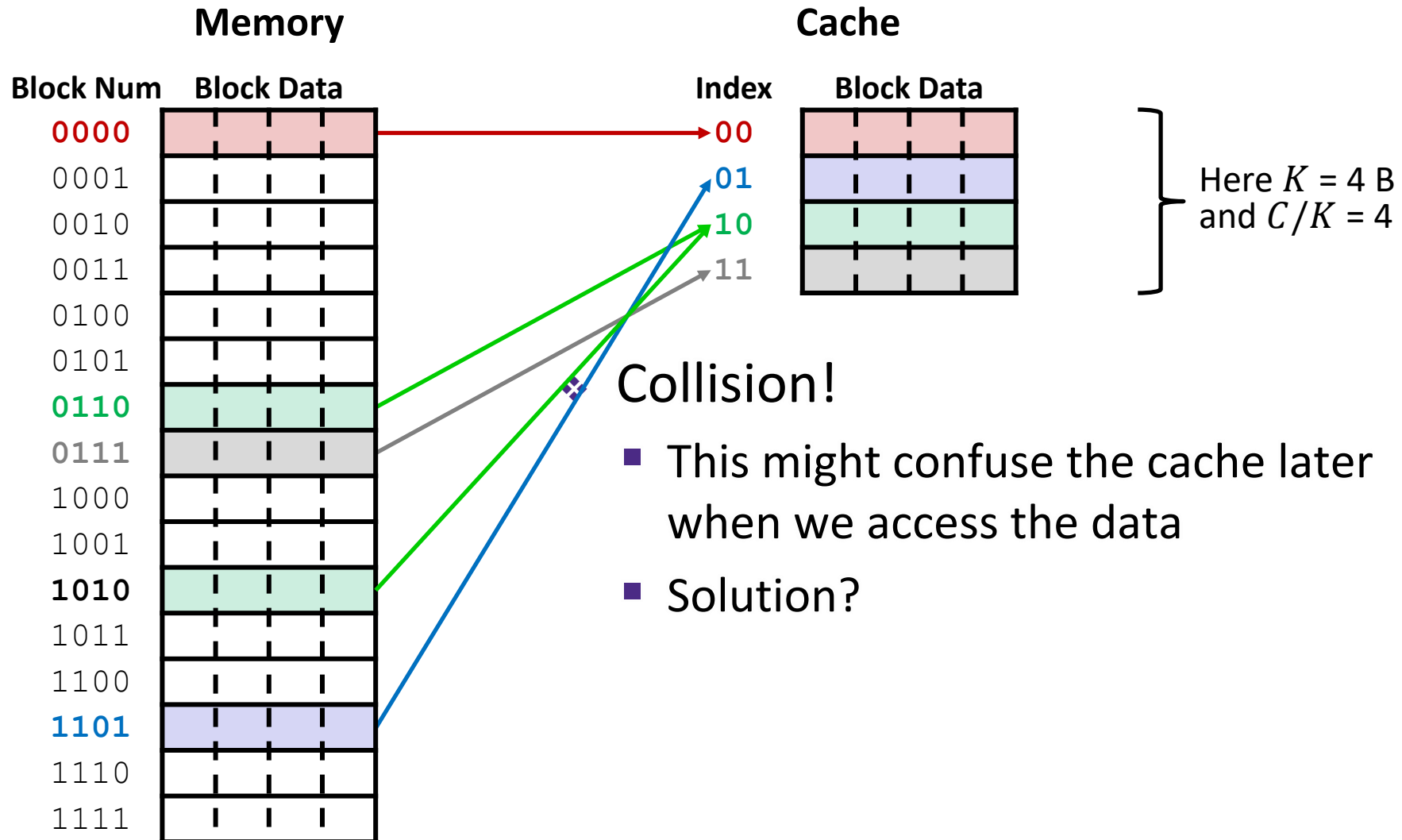■ **Lets adjacent blocks fit in cache simultaneously!**

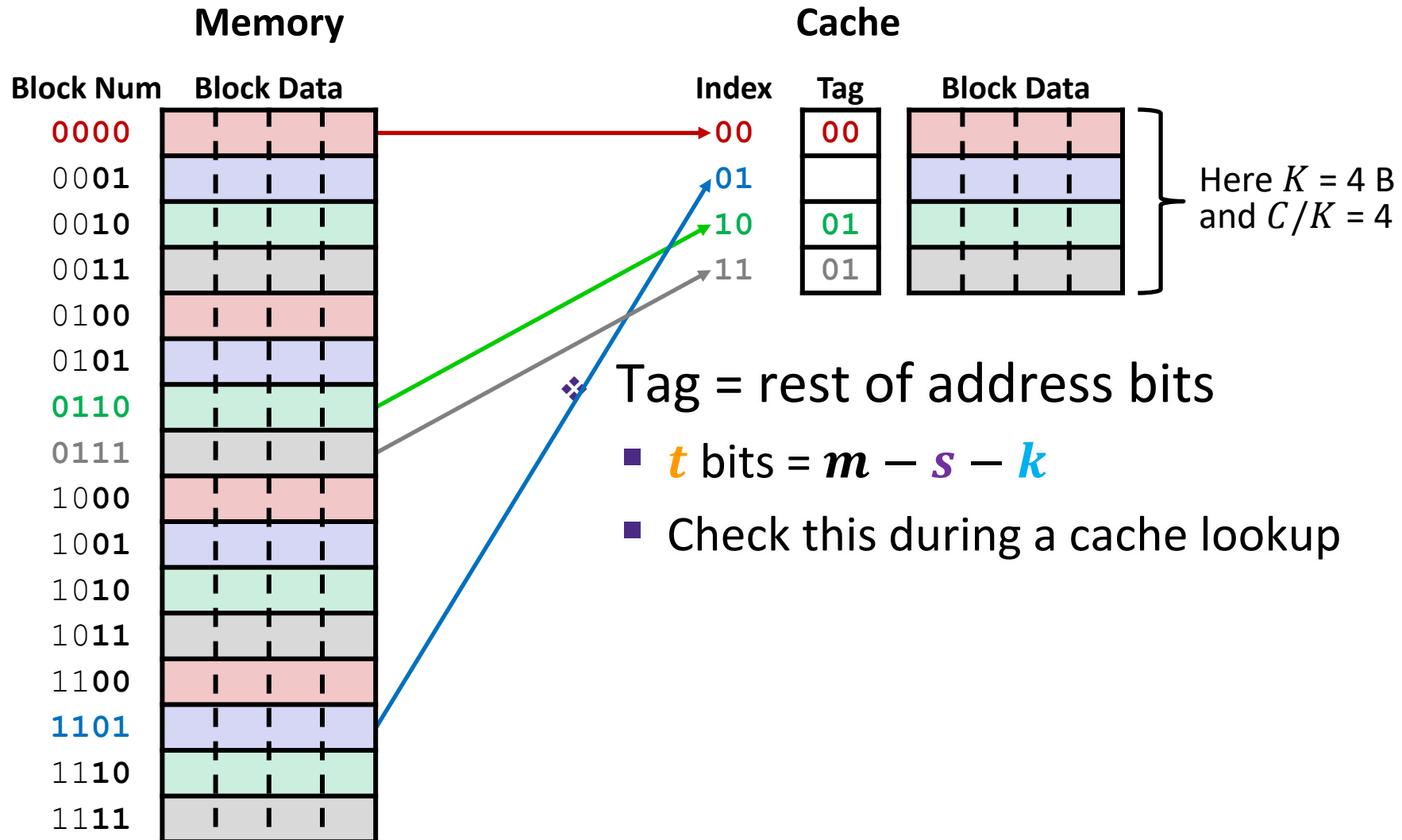 • Consecutive blocks go in consecutive cache indices

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

Memory   **Cache**

Block Num **Block Data**   **Index** **Block Data**

0000   00

0001   01   Here K = 4 B
                                                                                       and C/K = 4
0010   10

0011   11

0100

0101

0110

0111

1000

1001

1010

1011

1100

1101

1110

1111

Collision!

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

- Solution?

Tags Differentiate Blocks in Same Index

Memory

Cache

Block Num **Block Data**

Index **Tag** **Block Data**

Block Num	
0000	
0001	
0010	
0011	
0100	
0101	
0110	
0111	
1000	
1001	
1010	
1011	
1100	
1101	
1110	
1111	

Index: 00, 01, 10, 11

Tag: 00, 01, 01

Here K = 4 B and C/K = 4

❖ Tag = rest of address bits

- t bits = $m - s - k$

- 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 ≠ their phone number

❖ TIO address breakdown:

m-bit address:

Tag (t)	Index (s)	Offset (k)

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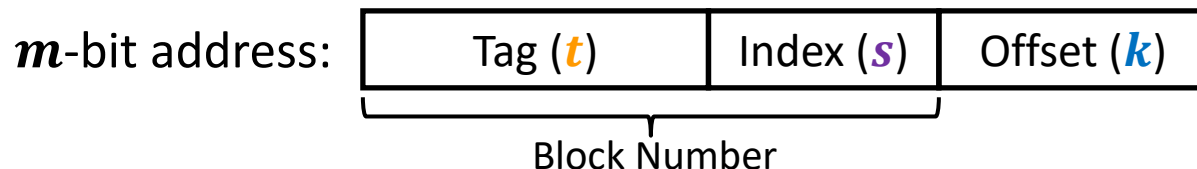
  ▪ **Index** field tells you where to look in cache
  ▪ **Tag** field lets you check that data is the block you want
  ▪ **Offset** field selects specified start byte within block

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

18

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

A. **4 bytes**

B. **8 bytes**

C. **16 bytes**

D. **32 bytes**

E. **We're lost…**