

Memory & Caches II

CSE 351 Summer 2022

Instructor:

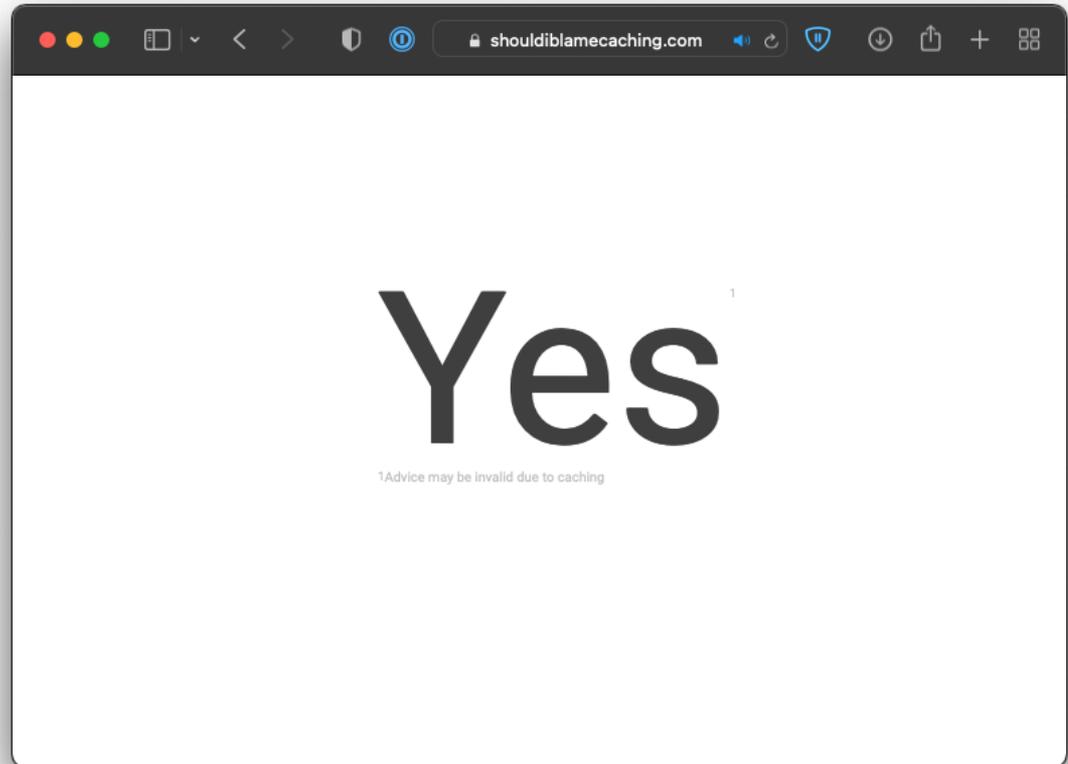
Kyrie Dowling

Teaching Assistants:

Aakash Srazali

Allie Pflieger

Ellis Haker



Relevant Course Information

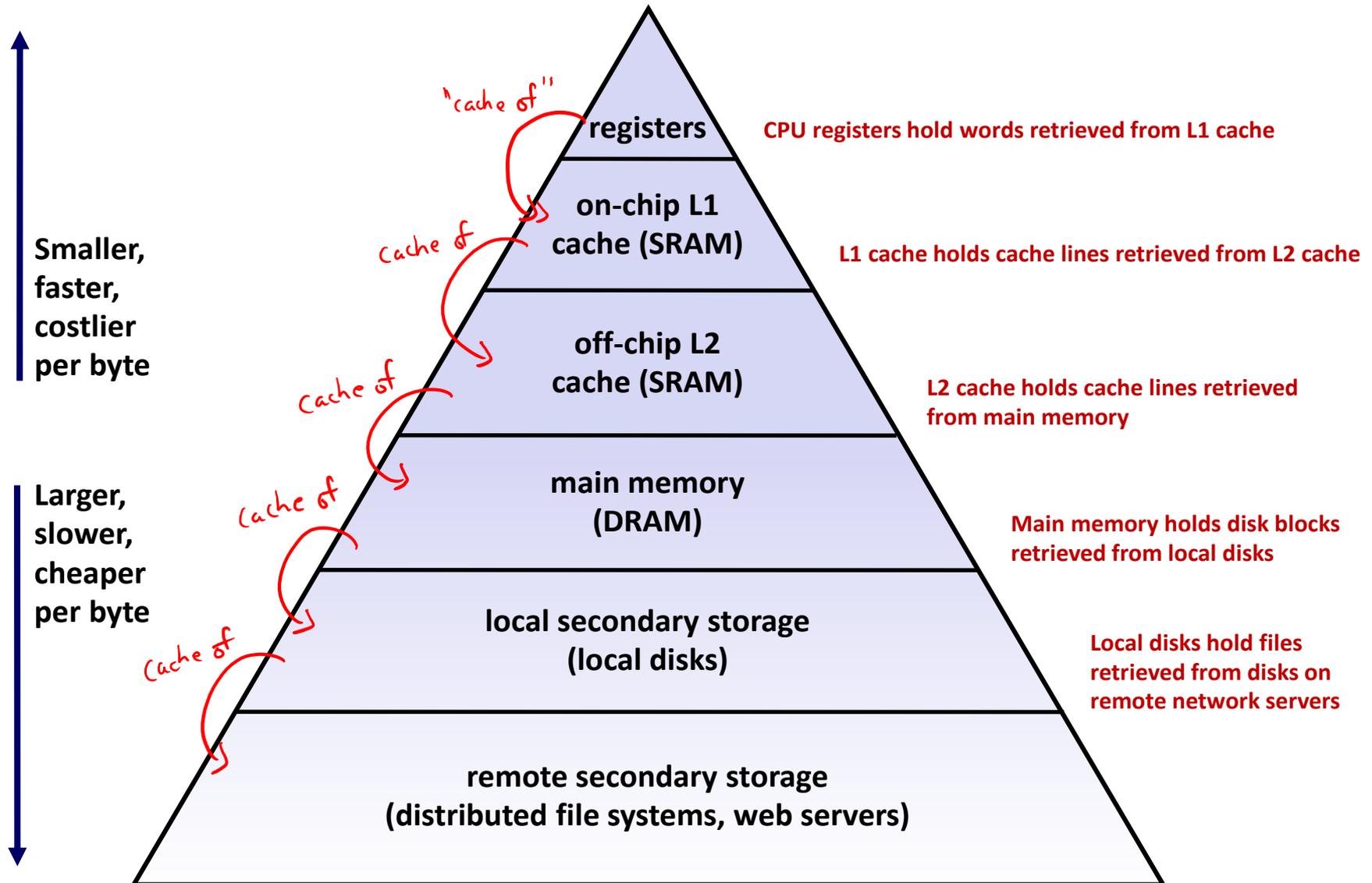
- ❖ Lab 3 due tonight
- ❖ hw 14 due tonight, hw15 due Monday
- ❖ hw16 due *next* Friday (8/5)
 - Don't wait too long, this is a BIG hw (includes this lecture)
- ❖ Unit Portfolio 2 due Wednesday (8/3)
 - Submit on Canvas, see Ed post with some tips and clarifications

Memory Hierarchies (Review)

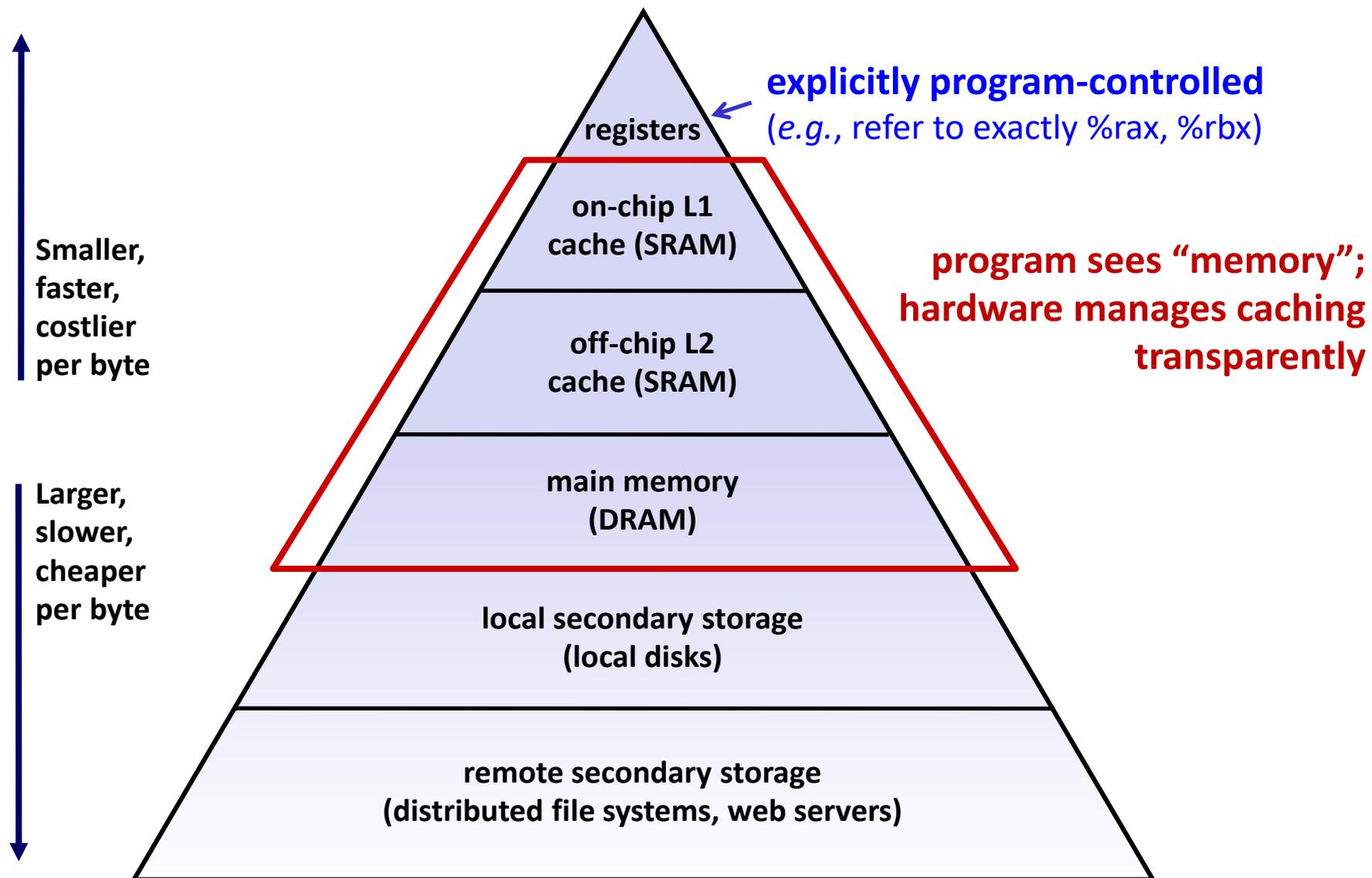
- ❖ Some fundamental and enduring properties of hardware and software systems:
 - Faster storage technologies almost always cost more per byte and have lower capacity
 - The gaps between memory technology speeds are widening
 - True for: registers \leftrightarrow cache, cache \leftrightarrow DRAM, DRAM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

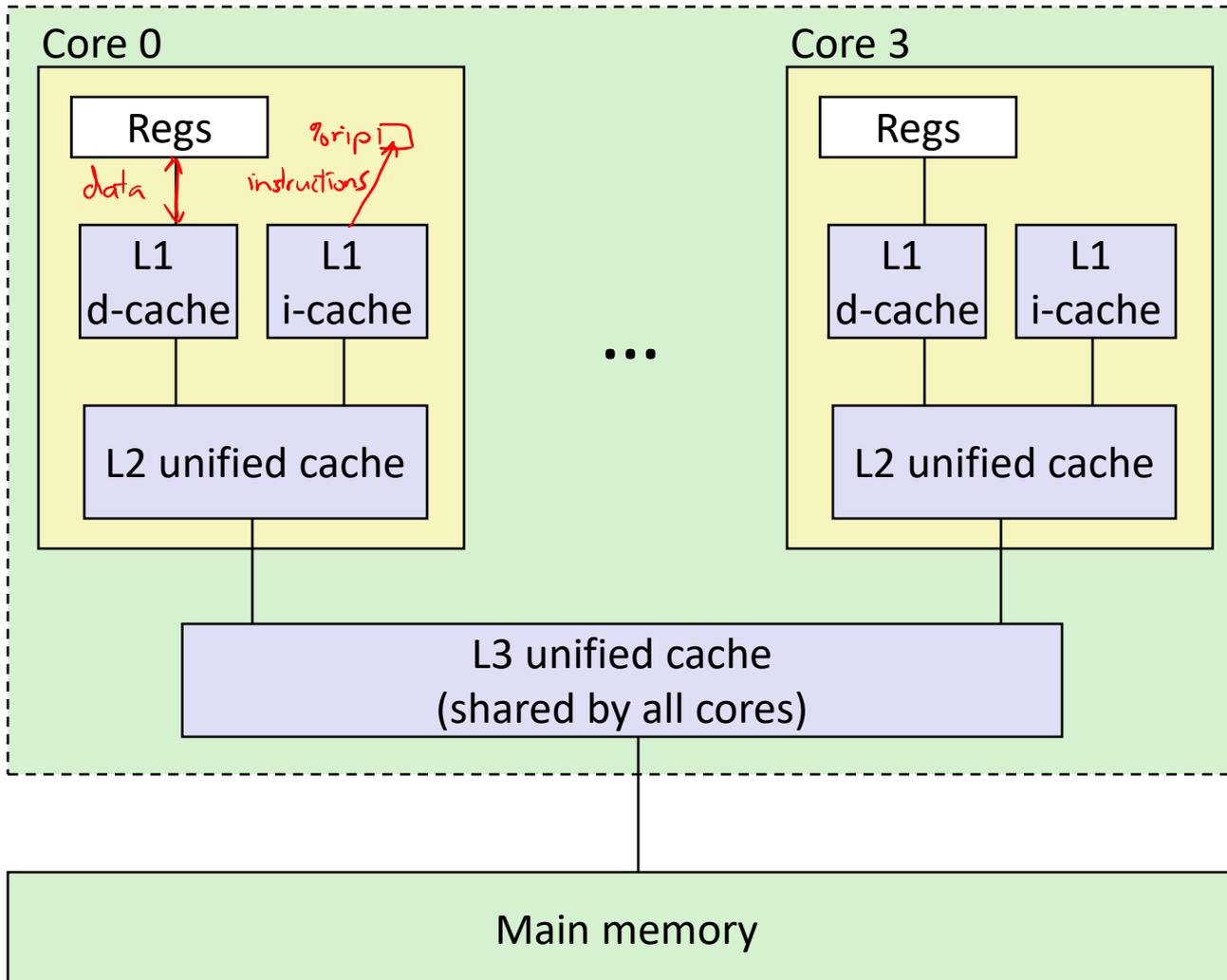


An Example Memory Hierarchy



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

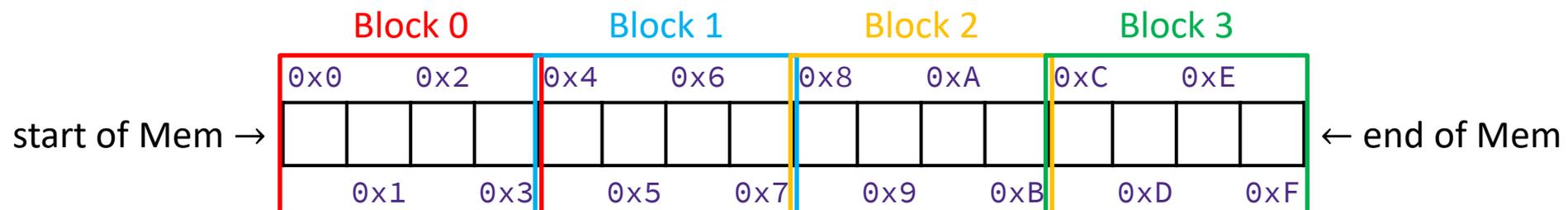
- ❖ We have a direct-mapped cache with the following parameters:
 - Block size of 8 bytes $K = 2^3 B$
 - Cache size of 4 KiB $C = 2^{12} B$
- ❖ How many blocks can the cache hold? $C/K = 2^{12-3} = 2^9 = 512 \text{ blocks}$
- ❖ How many bits wide is the block offset field? $k = \log_2(K) = 3 \text{ bits}$
- ❖ Which of the following addresses would fall under block number 3?

$\lfloor 3/8 \rfloor = 0$ A. 0x3 0b <u>000</u> /001 block num 0	$\lfloor 31/8 \rfloor = 3$ B. 0x1F 0b <u>011</u> /111 block num 3	$\lfloor 48/8 \rfloor = 6$ C. 0x30 0b <u>110</u> /000 block num 6	$\lfloor 56/8 \rfloor = 7$ D. 0x38 0b <u>111</u> /000 block num 7
---	---	---	---

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):

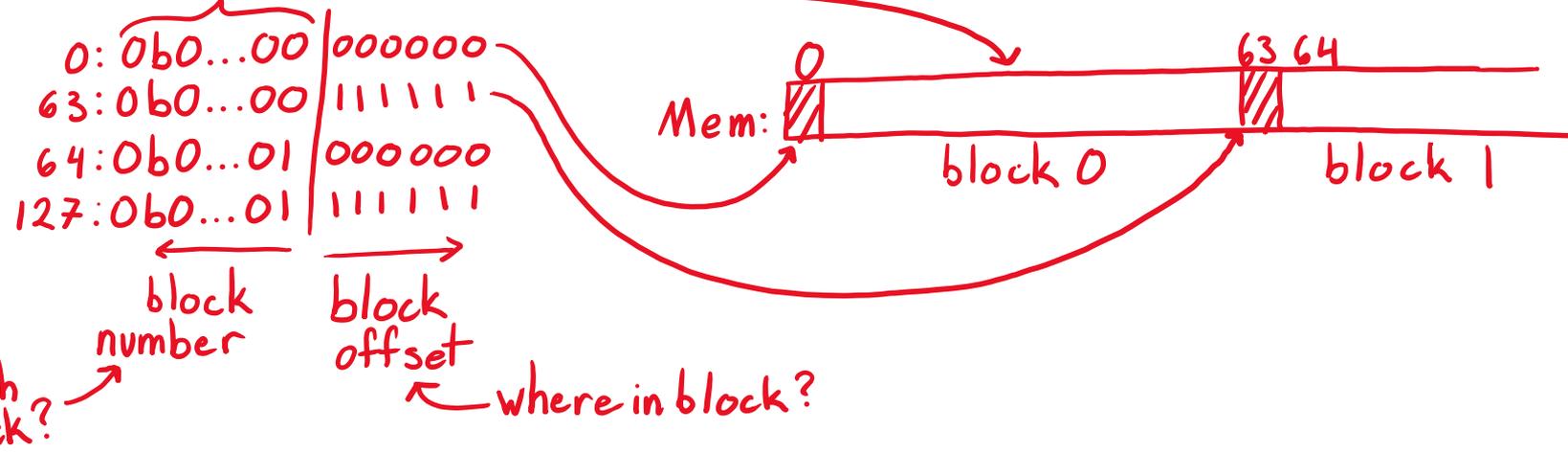
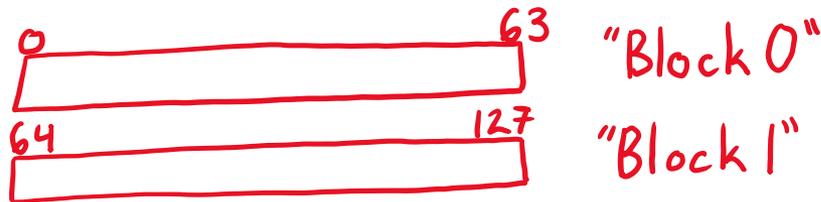


Cache Organization (1)

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!

Lab 1a: within Same Block



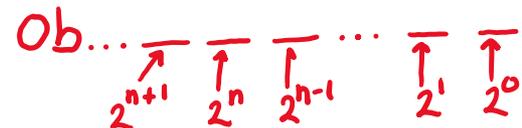
Cache Organization (1)

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!

$x \% 2^n = \text{value of lowest } n \text{ bits}$

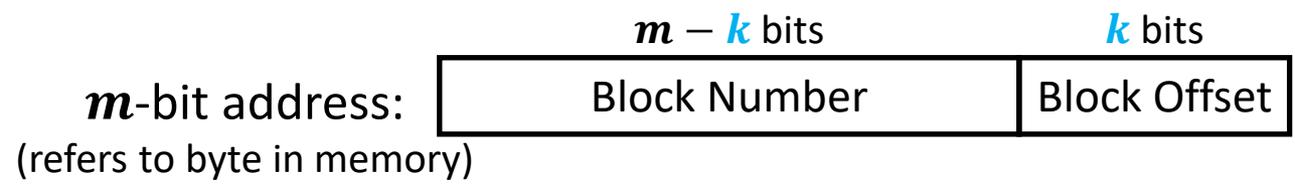
- ❖ **Offset field**
 - Low-order $\log_2(K) = k$ bits of address tell you which byte within a block



• (address) mod $2^n = n$ lowest bits of address

- (address) modulo (# of bytes in a block)

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



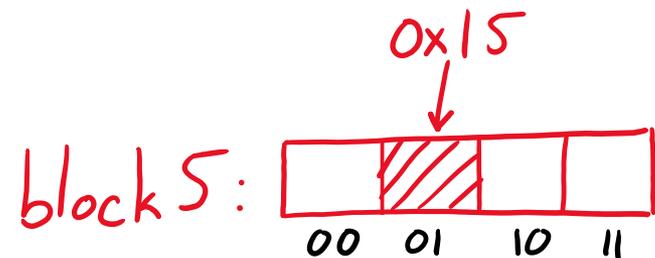
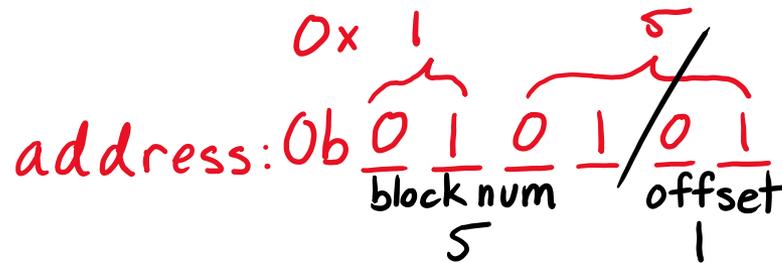
Cache Organization (1)

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 ^{m} 6-bit addresses and block size $K = 4$ B, which block and byte does 0x15 refer to?



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

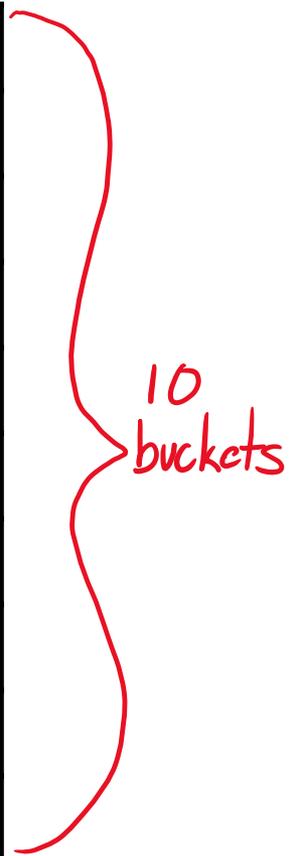
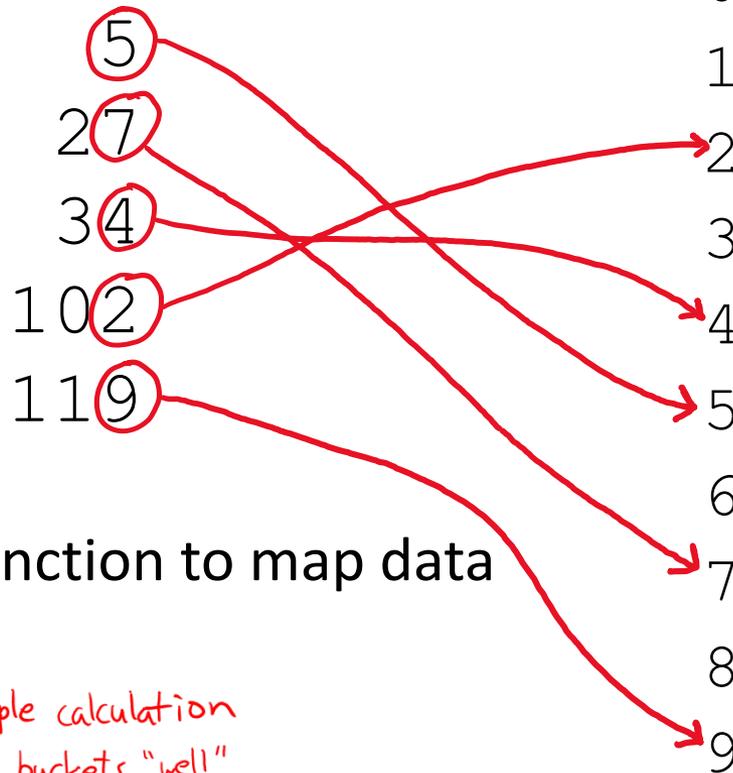
Cache Organization (2)

- ❖ **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

$$2^5 \times 2^{10} = 2^{15} \text{ B} \times \frac{1 \text{ block}}{2^6 \text{ B}} = 2^9 \text{ blocks}^{2^6}$$
- ❖ 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:

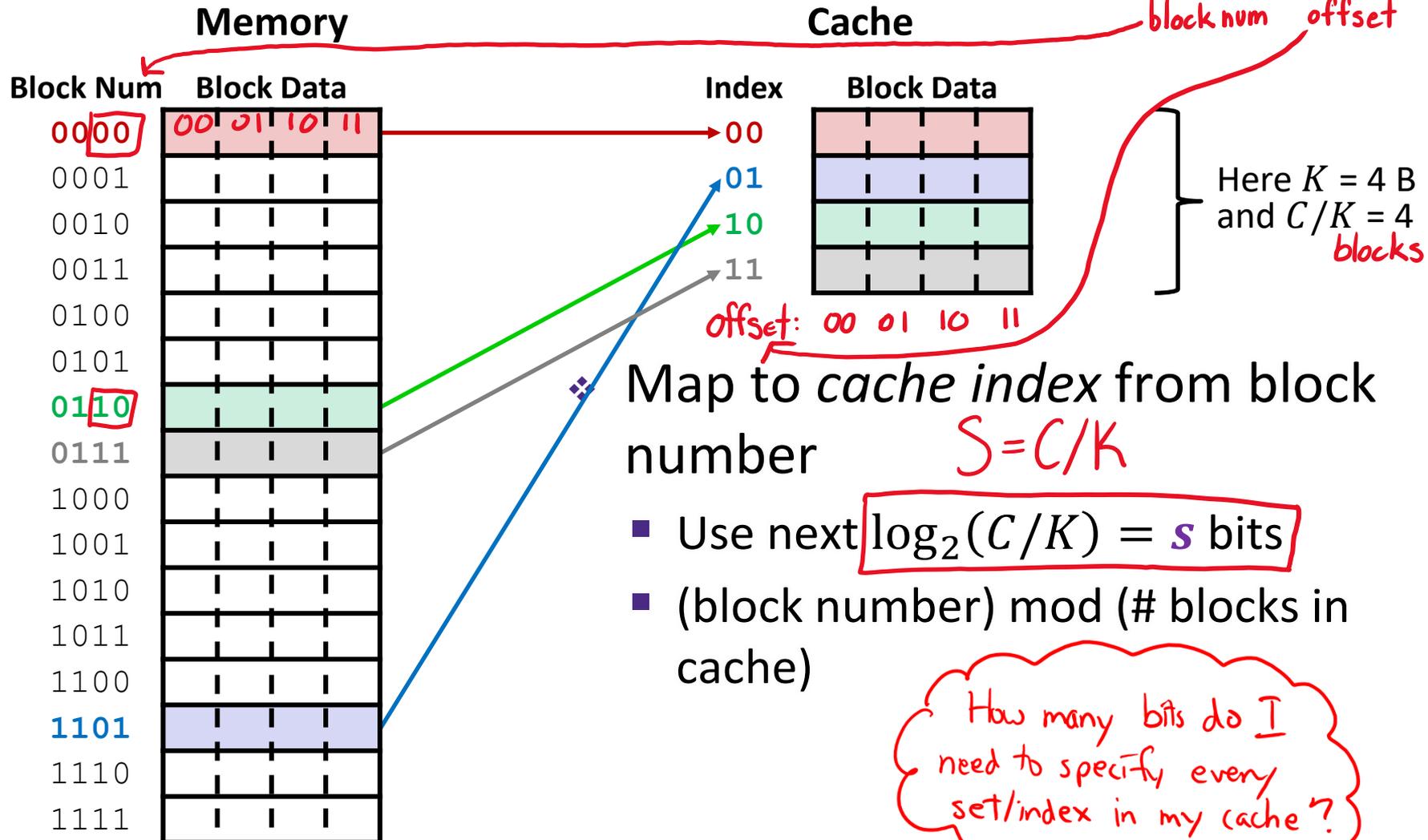


Apply hash function to map data to "buckets"

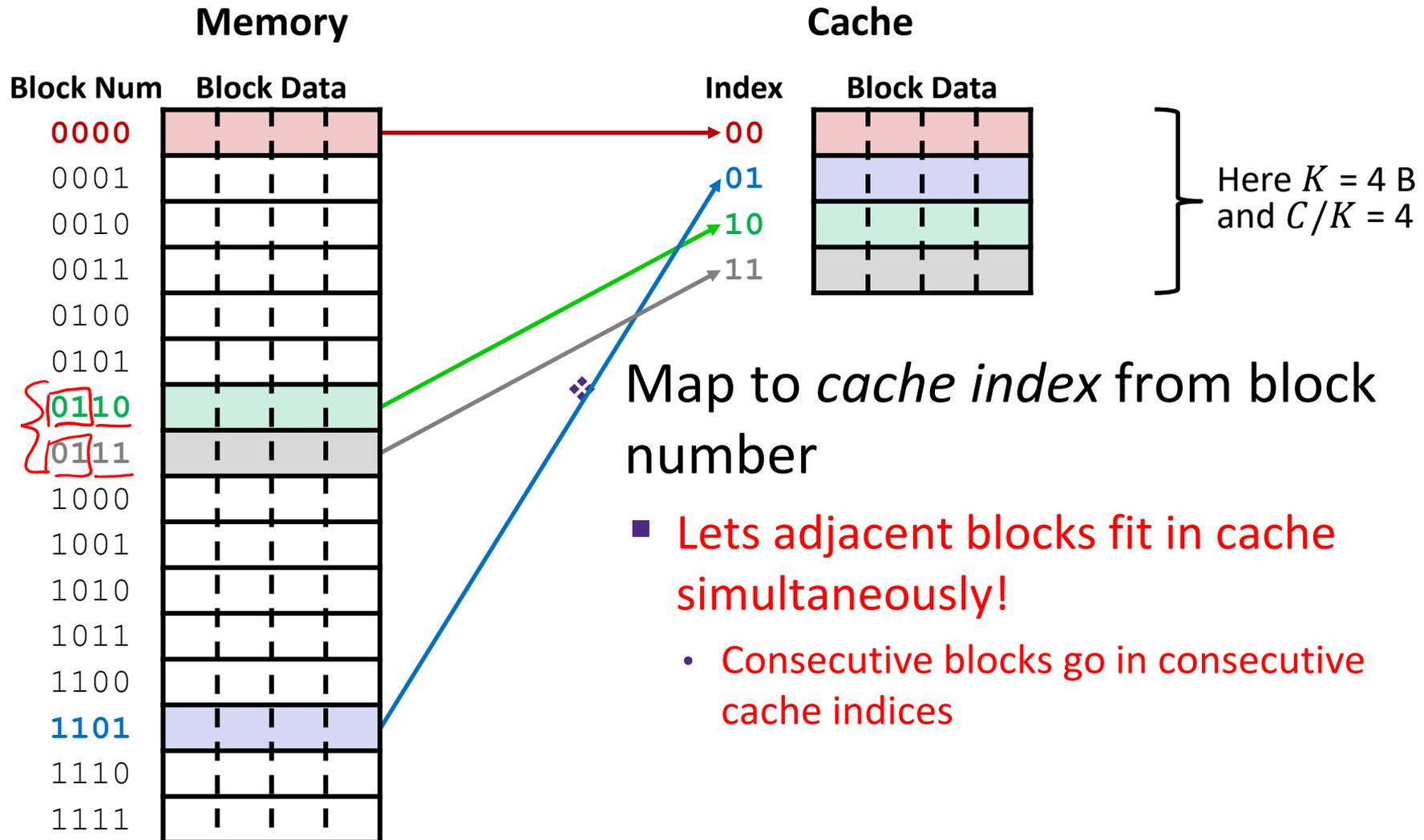
- Goals:
- ① fast/simple calculation
 - ② use all buckets "well"

Place Data in Cache by Hashing Address

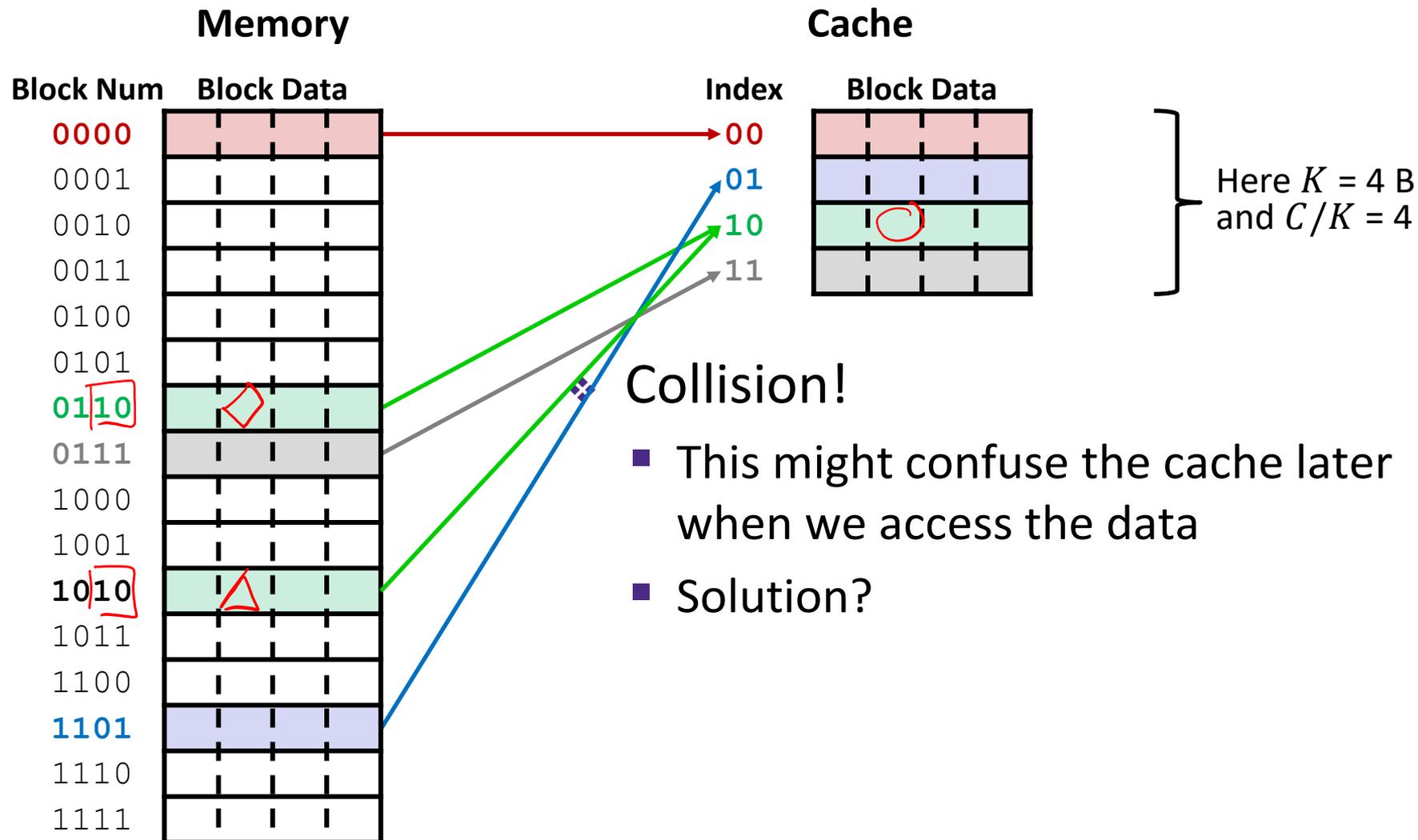
addresses are 6 bits: 0b XXXX/XX
 block num offset



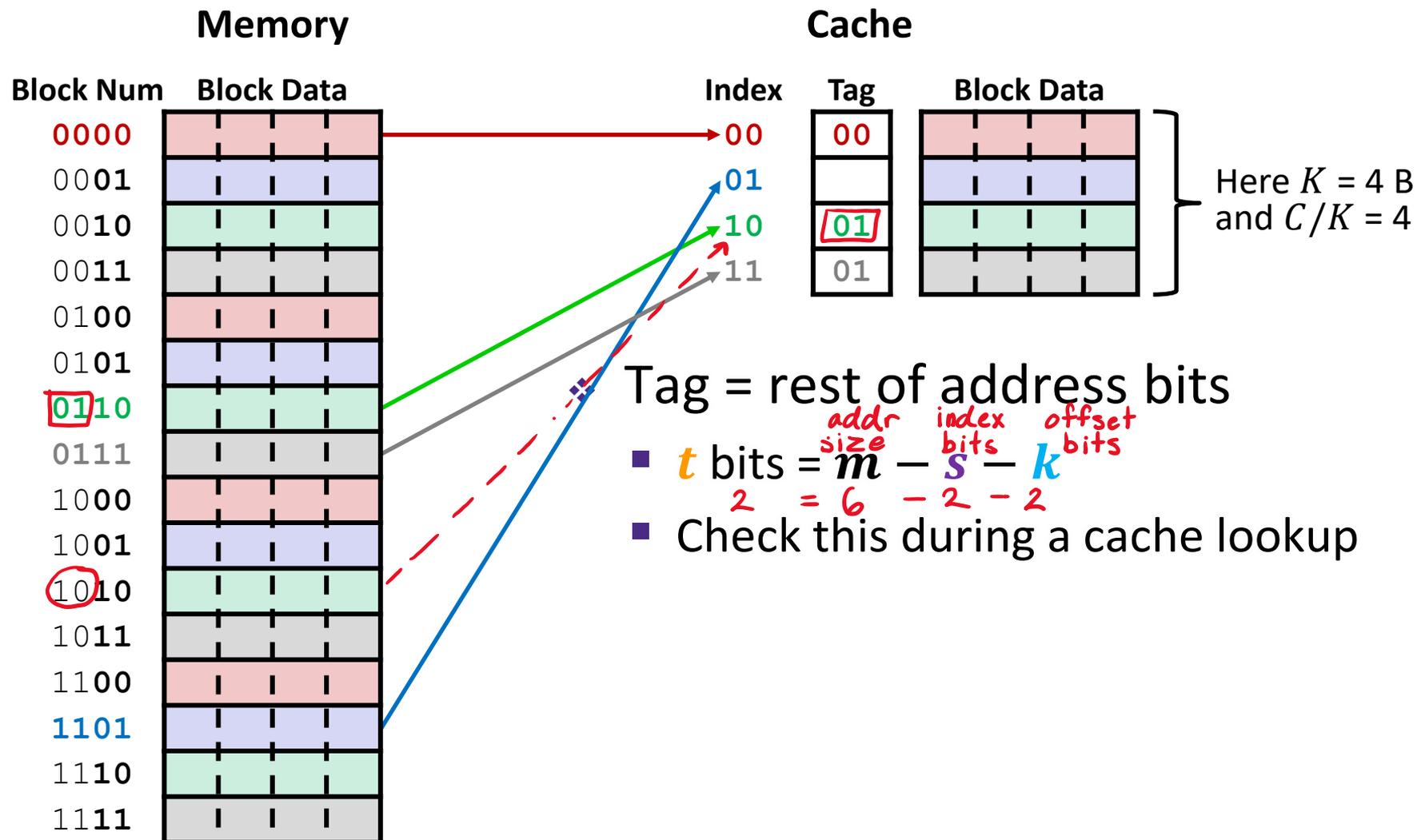
Place Data in Cache by Hashing Address



Place Data in Cache by Hashing Address



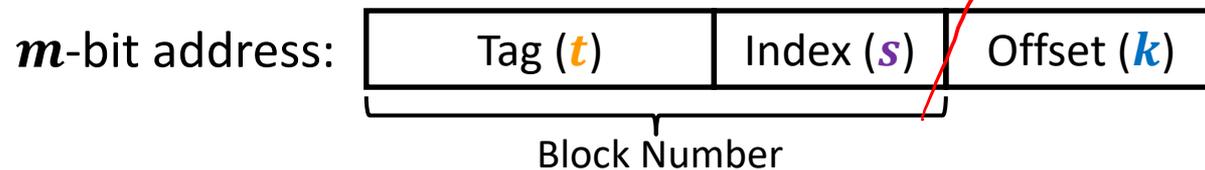
Tags Differentiate Blocks in Same Index



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:



- ① ■ **Index** field tells you where to look in cache
 - ② ■ **Tag** field lets you check that data is the block you want
 - ③ ■ **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)

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

A. 4 bytes

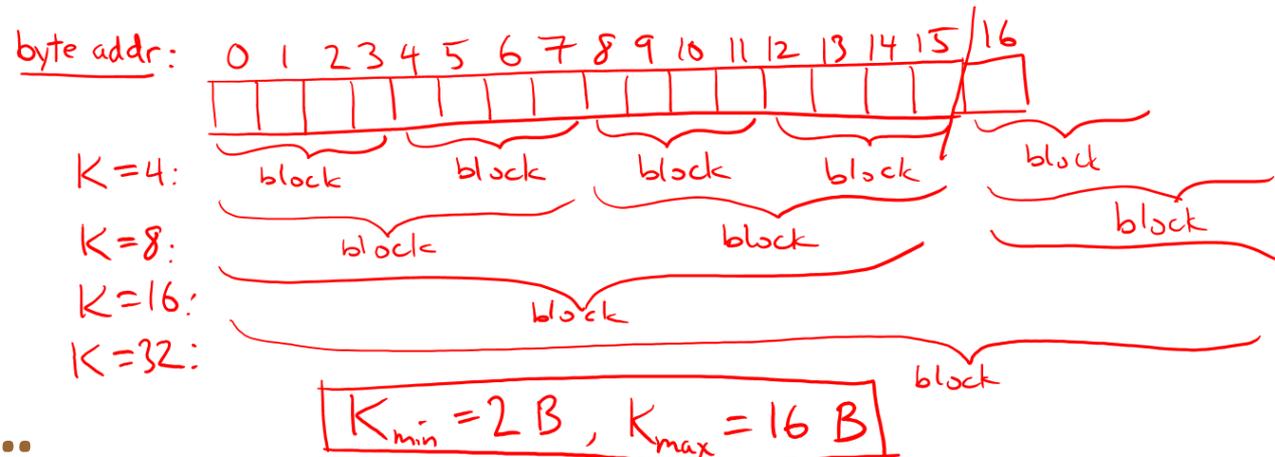
B. 8 bytes

C. 16 bytes

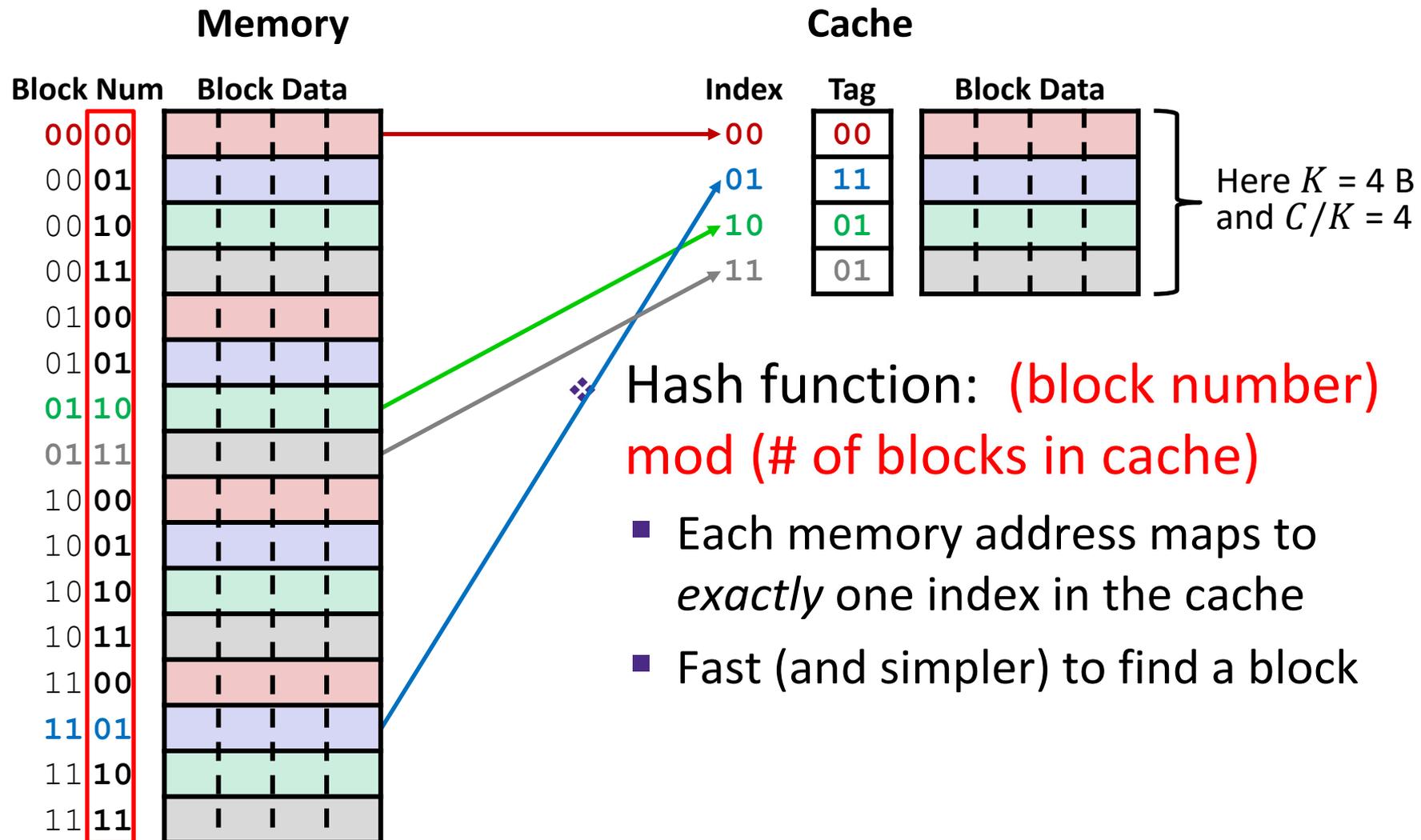
D. 32 bytes

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



Summary: Direct-Mapped Cache



Direct-Mapped Cache: A Problem!

