# Caches II

CSE 351 Spring 2022

**Instructor:**

Ruth Anderson

**Teaching Assistants:**

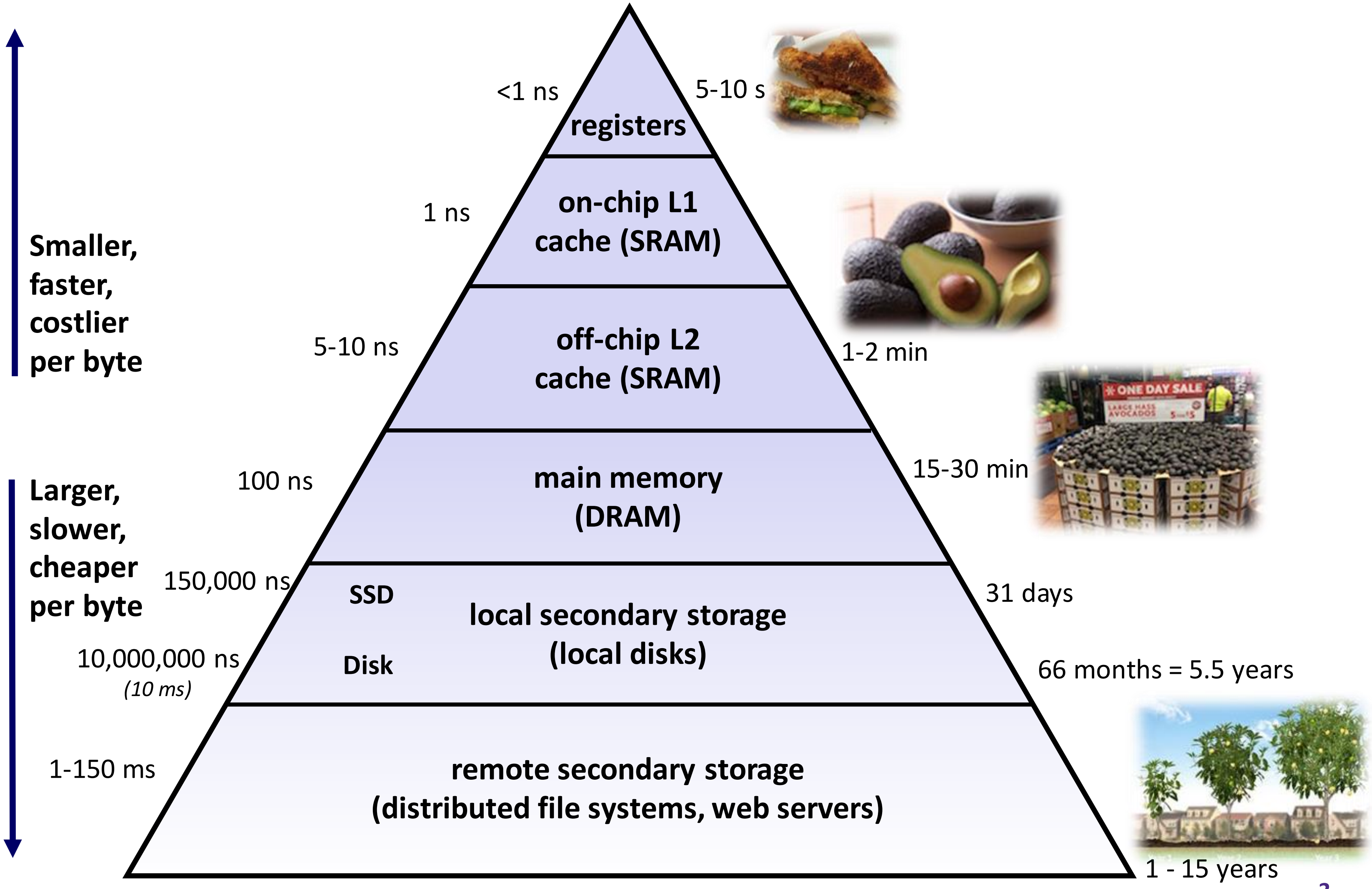
Melissa Birchfield
Jacob Christy
Alena Dickmann
Kyrie Dowling
Ellis Haker
Maggie Jiang
Diya Joy
Anirudh Kumar
Jim Limprasert
Armin Magness
Hamsa Shankar
Dara Stotland
Jeffery Tian
Assaf Vayner
Tom Wu
Angela Xu
Effie Zheng

# Relevant Course Information

- **Midterm due TONIGHT** Wednesday 5/04 11:59pm
- hw15 due Friday (5/06)
- Mid-quarter Survey due Saturday (5/07)

- hw16 due Monday (5/09)
- Lab 3 due Wednesday (5/11)
  - You will have everything you need for this now!
  - Some discussion in section this week
  - Last part of hw15 (due Fri 5/06) is useful for Lab 3
- hw17 due *next* Friday (5/13)
  - Don’t wait too long, this is a BIG hw

# An Example Memory Hierarchy



Smaller, faster, costlier per byte

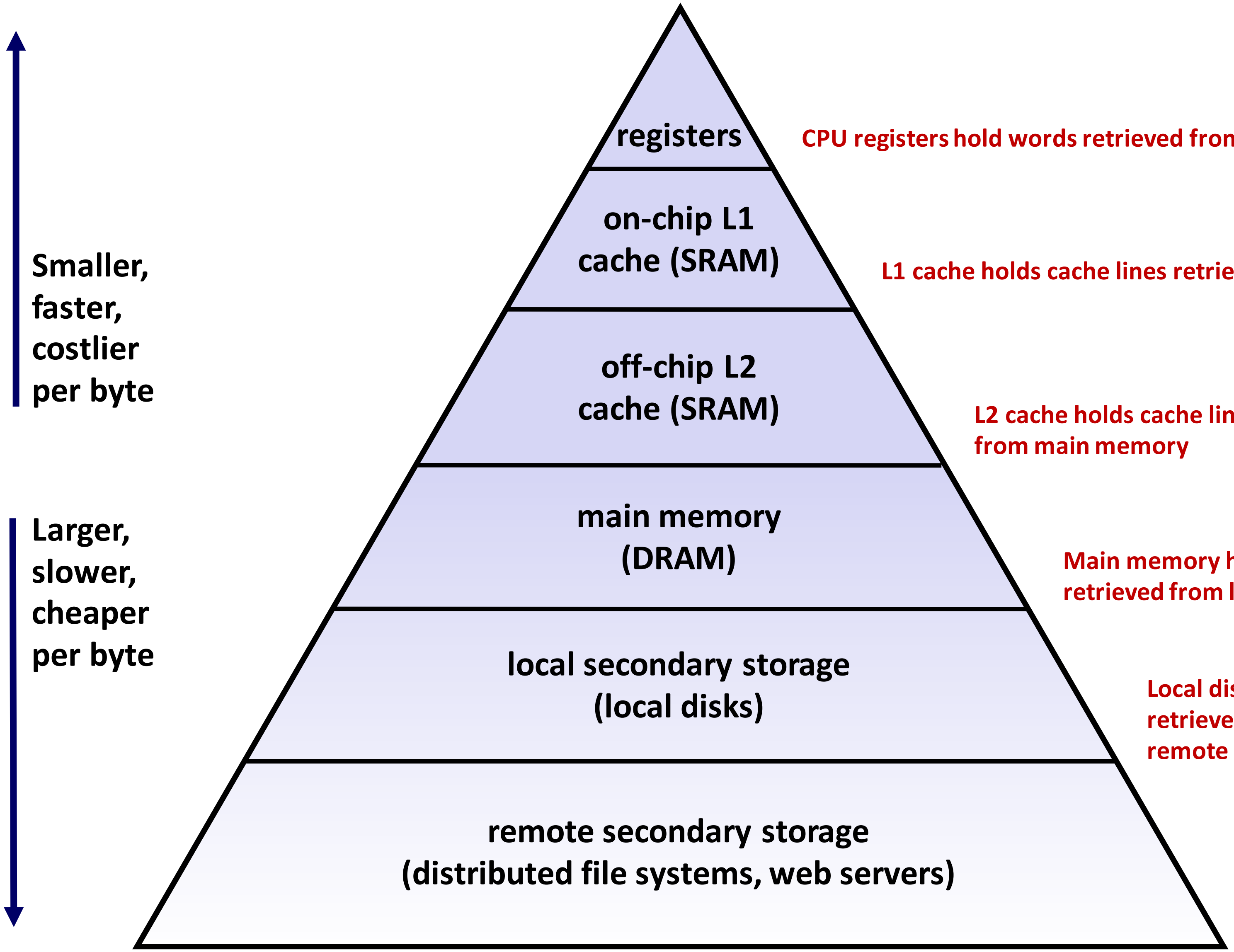
Larger, slower, cheaper per byte

# 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 ↔ 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



CPU registers hold words retrieved from L1 cache

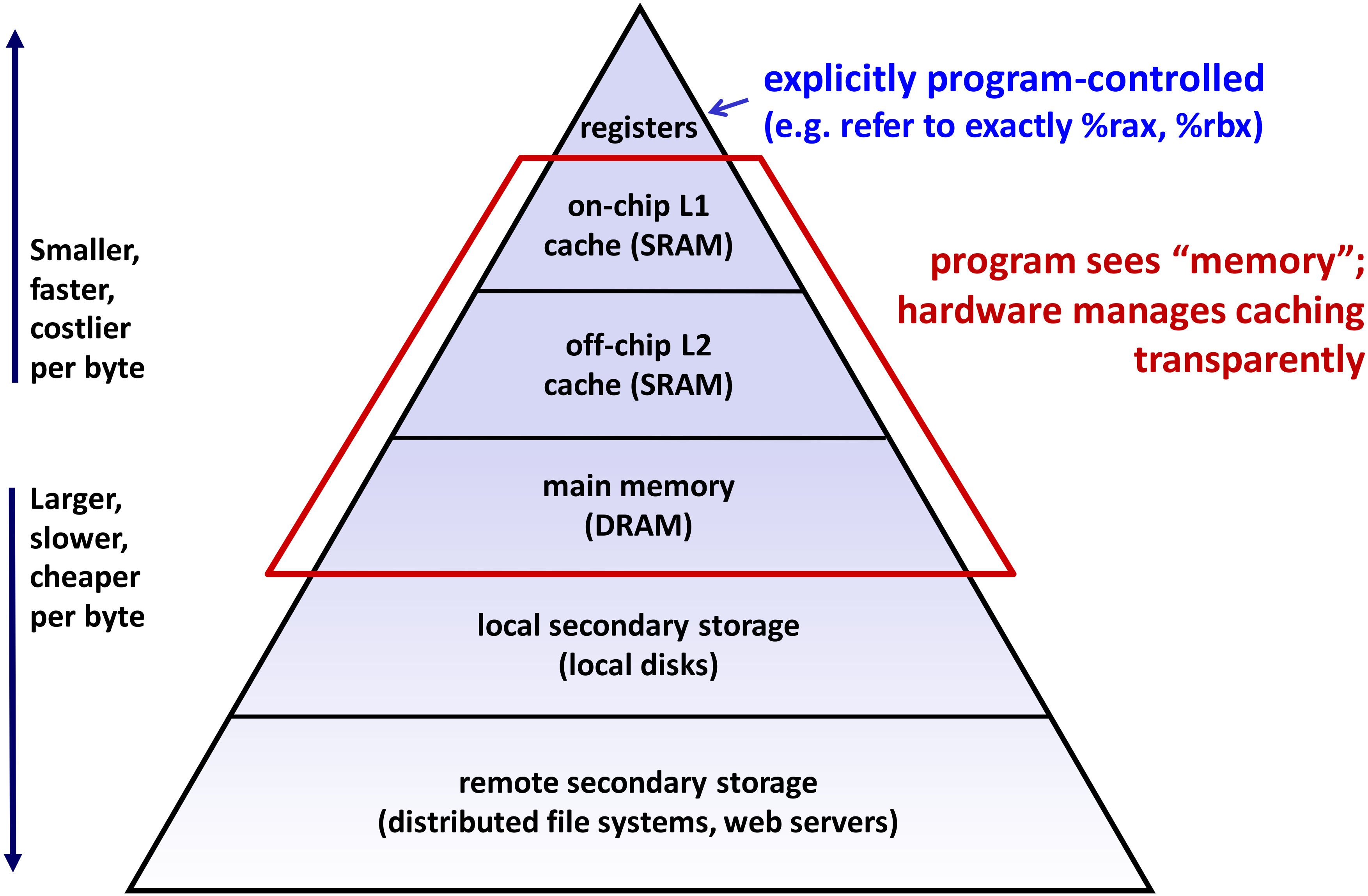
L1 cache holds cache lines retrieved from L2 cache

L2 cache holds cache lines retrieved from main memory

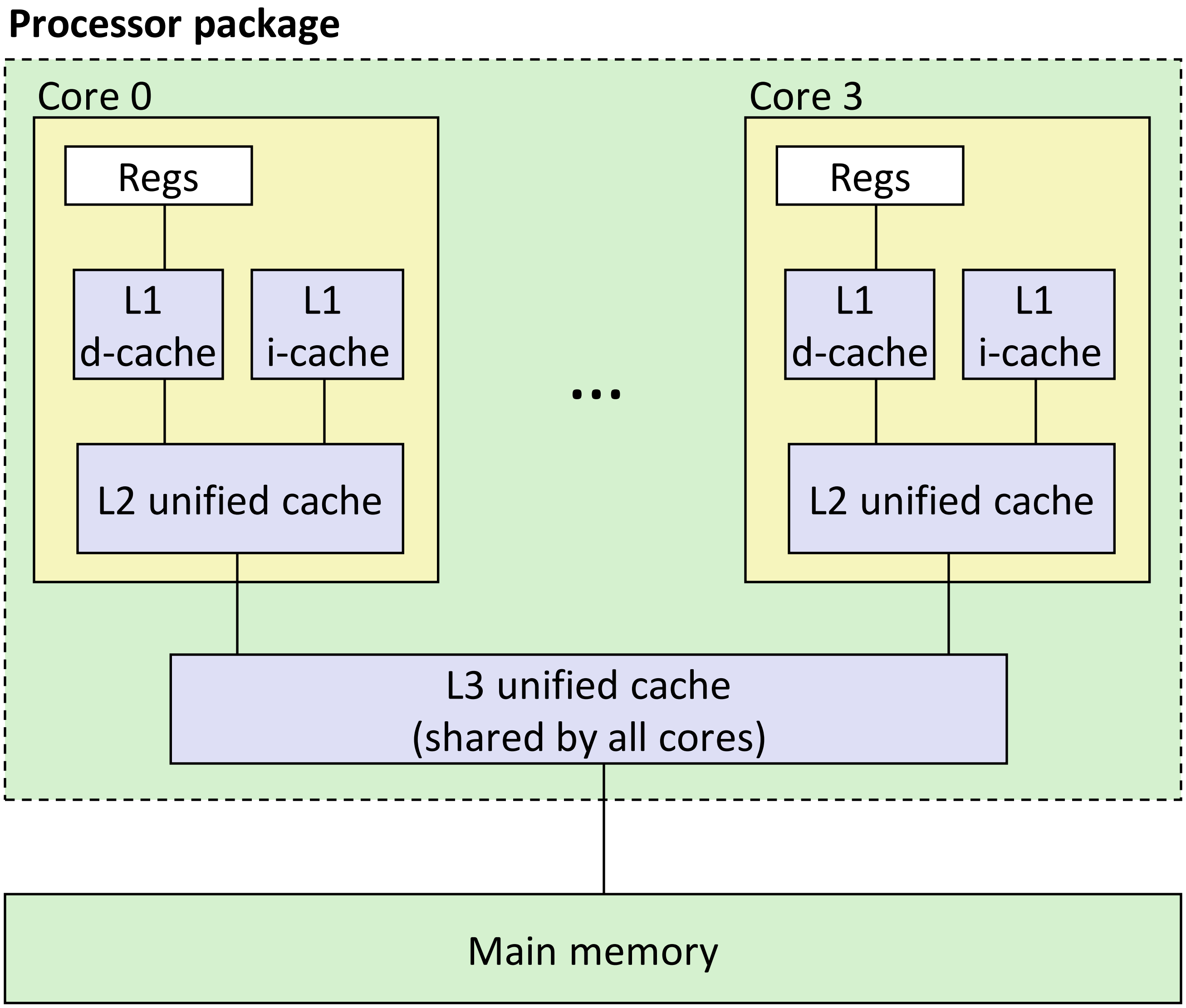
Main memory holds disk blocks retrieved from local disks

Local disks hold files retrieved from disks on remote network servers

# An Example Memory Hierarchy



# Intel Core i7 Cache Hierarchy



**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

# Review Questions

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

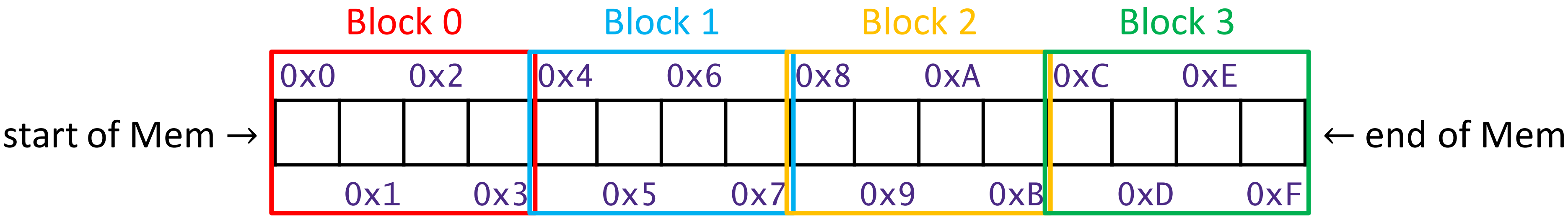
- How many blocks can the cache hold?
- How many bits wide is the block offset field?
- Which of the following addresses would fall under block number 3?

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)

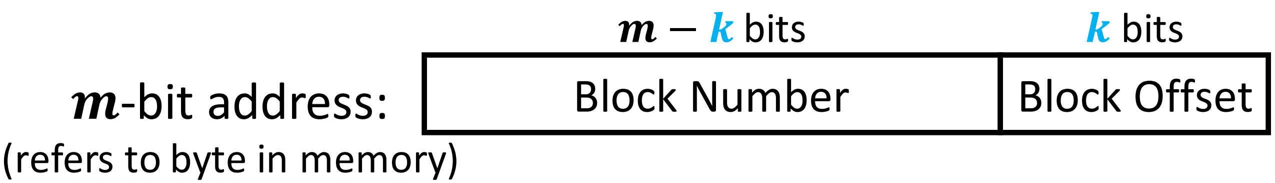
**Note:** The textbook uses “B” for block size

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

# 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)

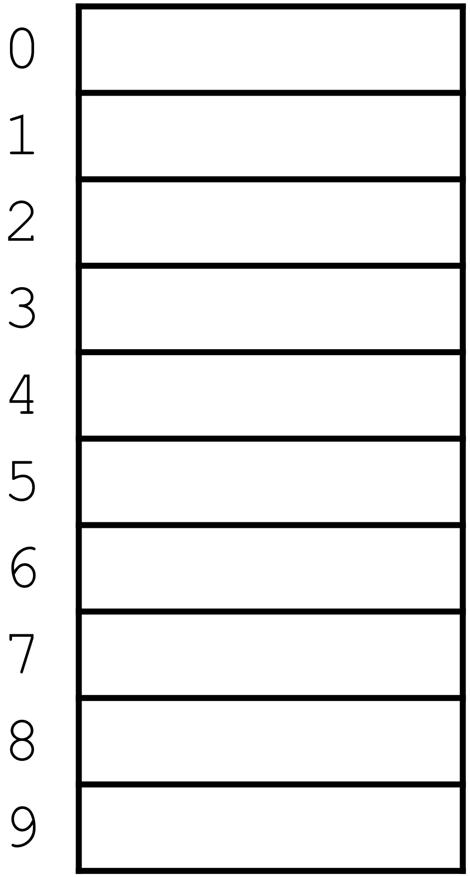
- Cache Size ($C$): amount of *data* the $ can store
  - Cache can only hold so much data (subset of next level)
  - Given in bytes ($C$) or number of blocks ($C/K$)
  - Example: $C$ = 32 KiB = 512 blocks if using 64-B blocks
- Where should data go in the cache?
  - We need a mapping from memory addresses 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

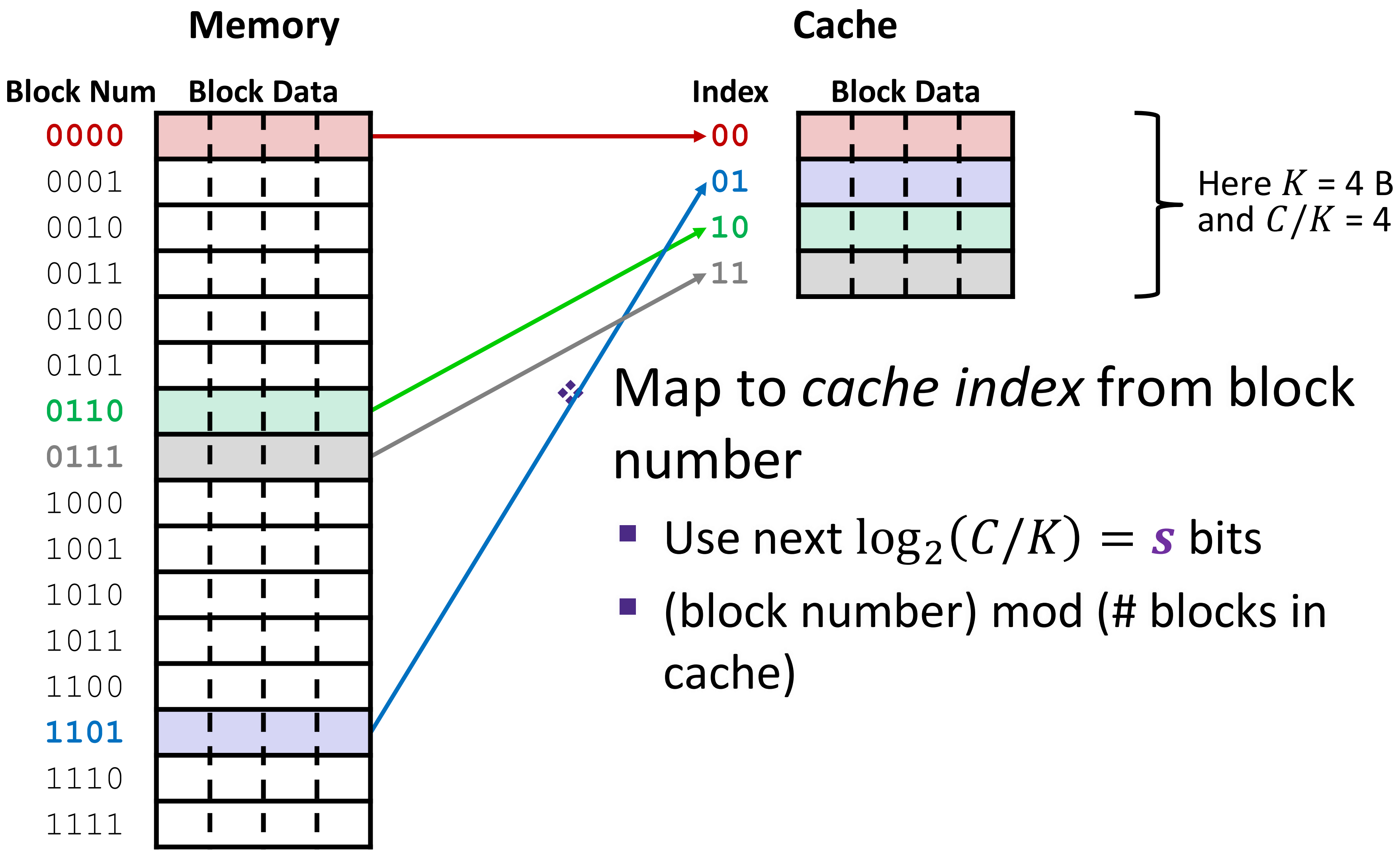
**Insert:**

```
  5
 27
 34
102
119
```

Apply hash function to map data to “buckets”



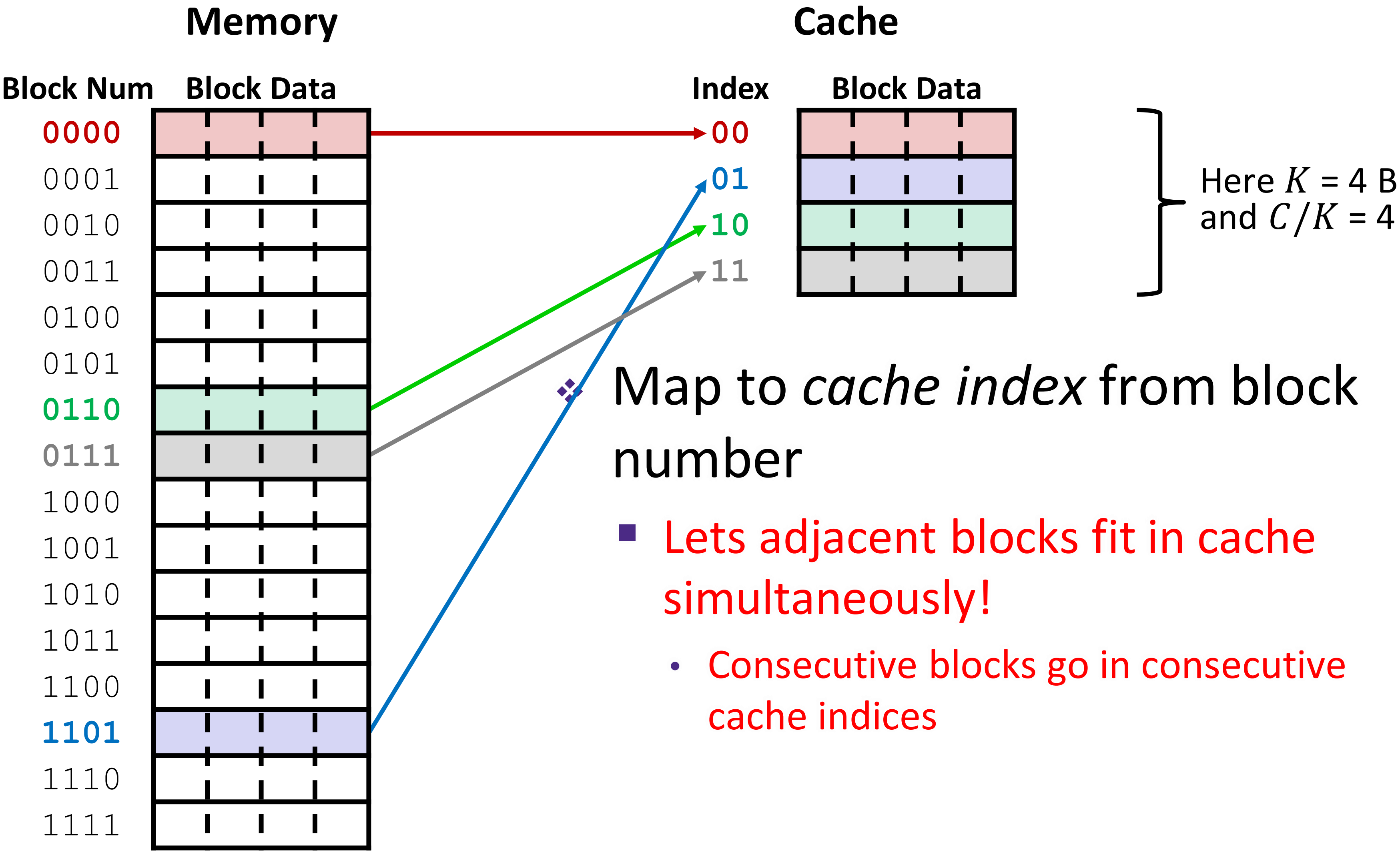
# Place Data in Cache by Hashing Address



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)

# Place Data in Cache by Hashing Address



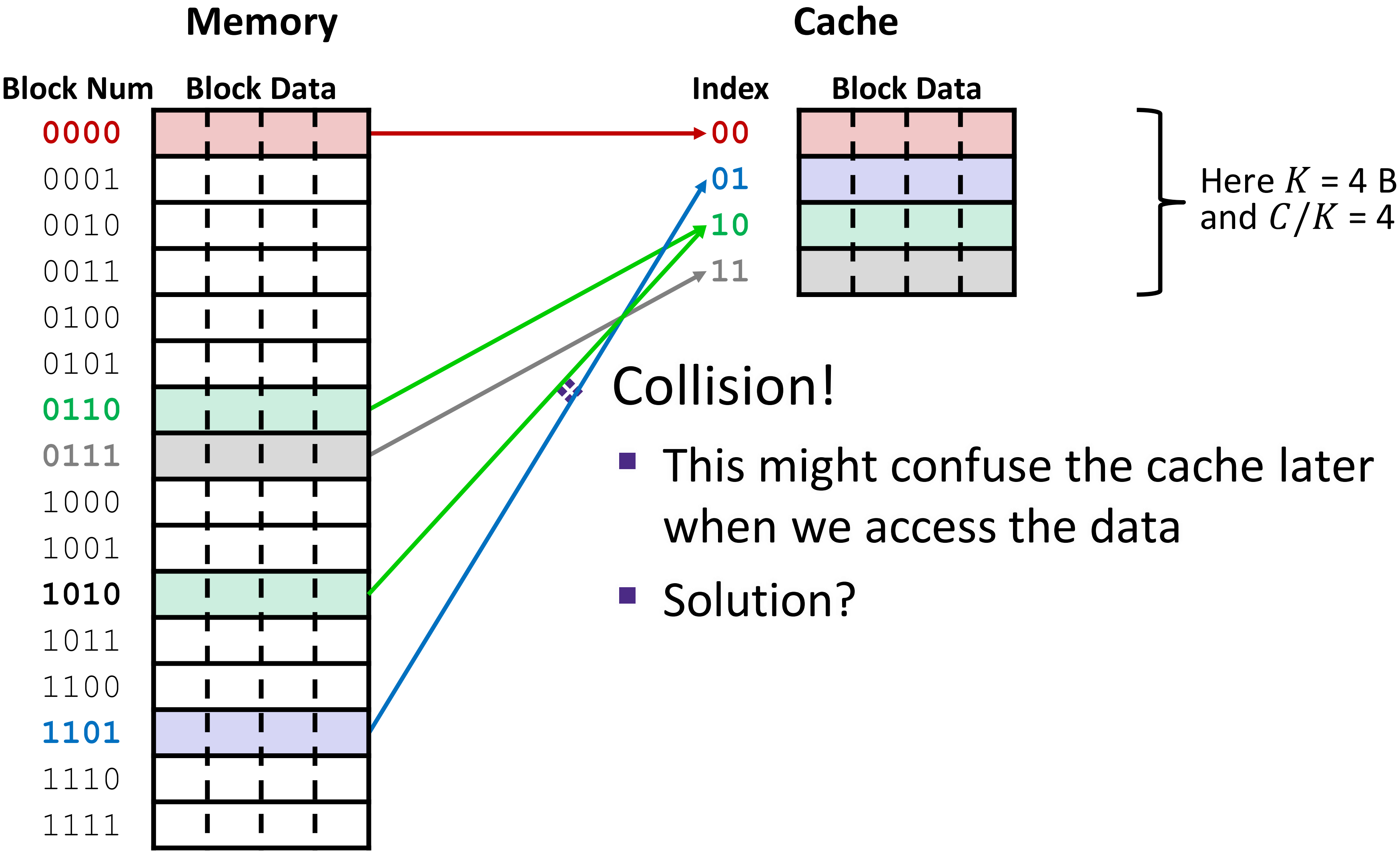
Map to *cache index* from block number

- Lets adjacent blocks fit in cache simultaneously!
    - Consecutive blocks go in consecutive cache indices

# 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

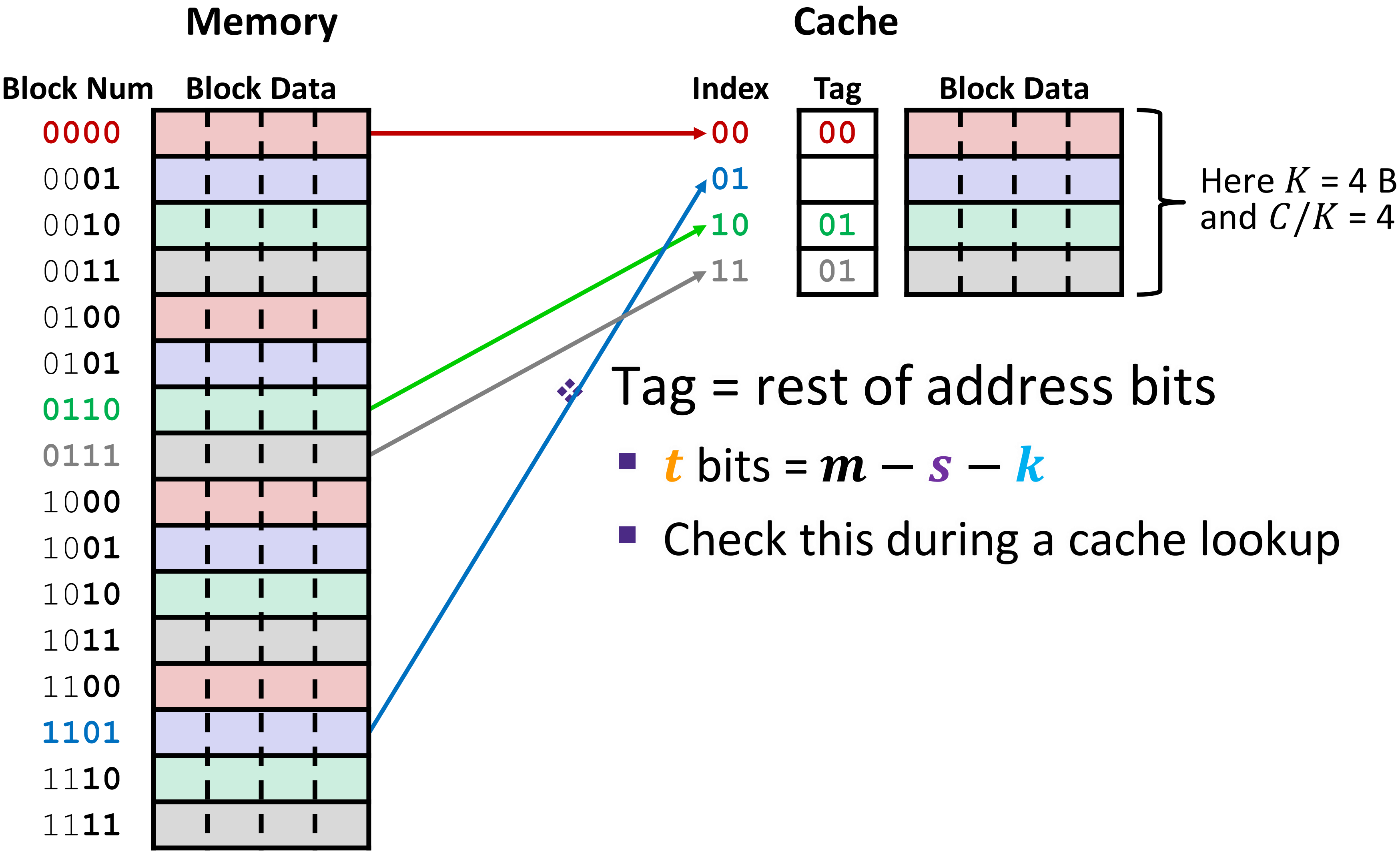
# Place Data in Cache by Hashing Address



## Collision!

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

# Tags Differentiate Blocks in Same Index

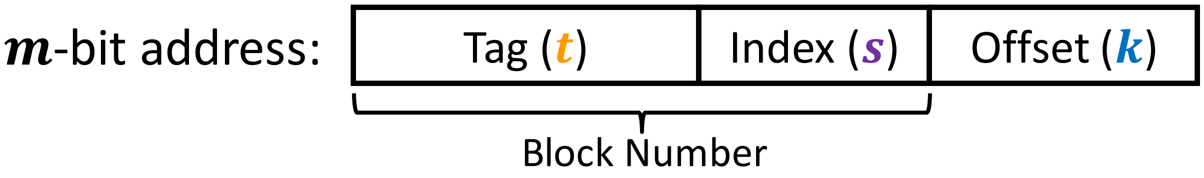


- Tag = rest of address bits
  - $t$ bits = $m - s - k$
  - Check this during a cache lookup

# 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 ($k$) |
|---|---|---|

$\underbrace{\text{Tag } (t) \quad \text{Index } (s)}_{\text{Block Number}}$

  - **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

# 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)
- [Not in Ed Lessons]

**A. 4 bytes**

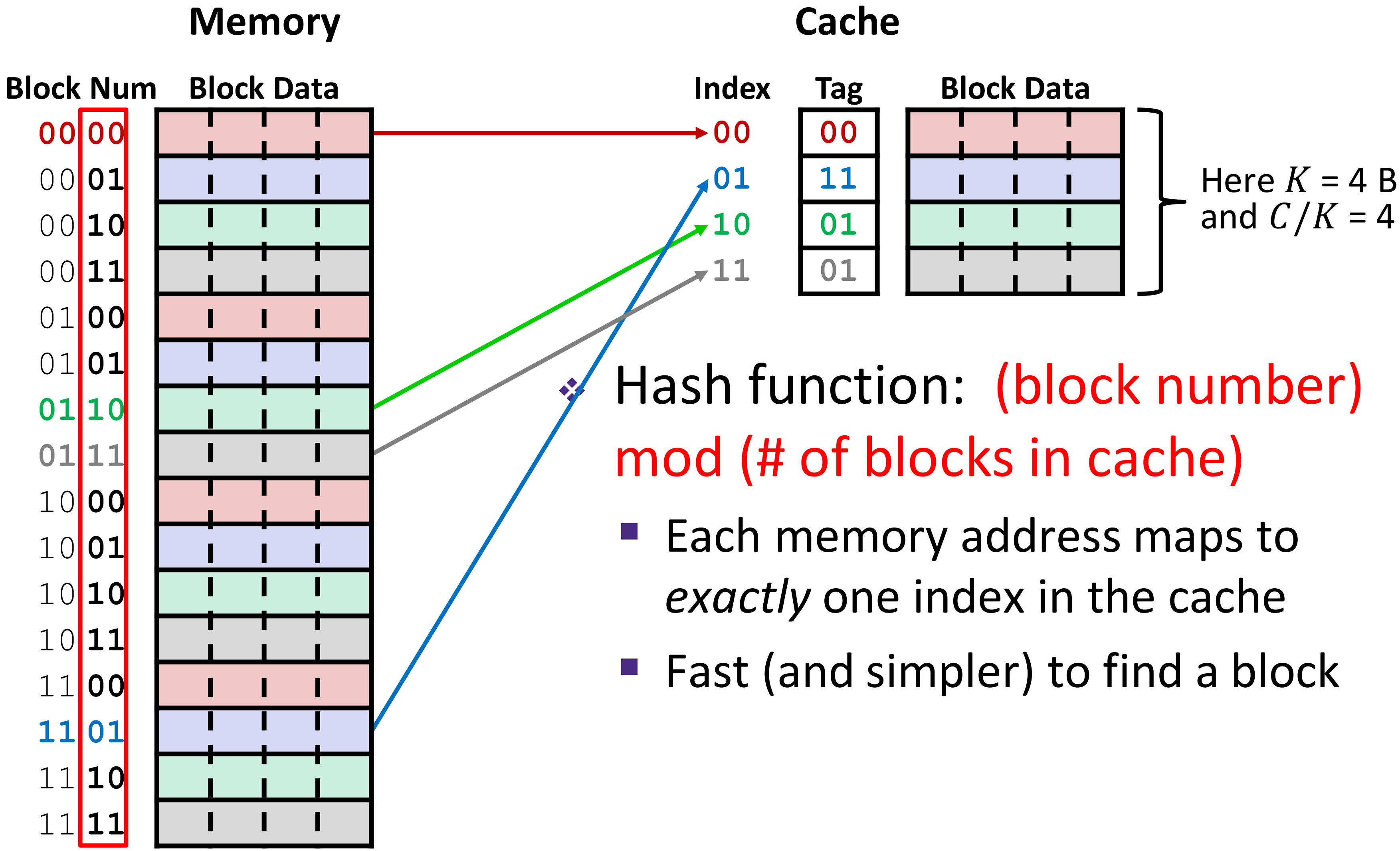
**B. 8 bytes**

**C. 16 bytes**

**D. 32 bytes**

**E. We’re lost…**

# Summary: Direct-Mapped Cache

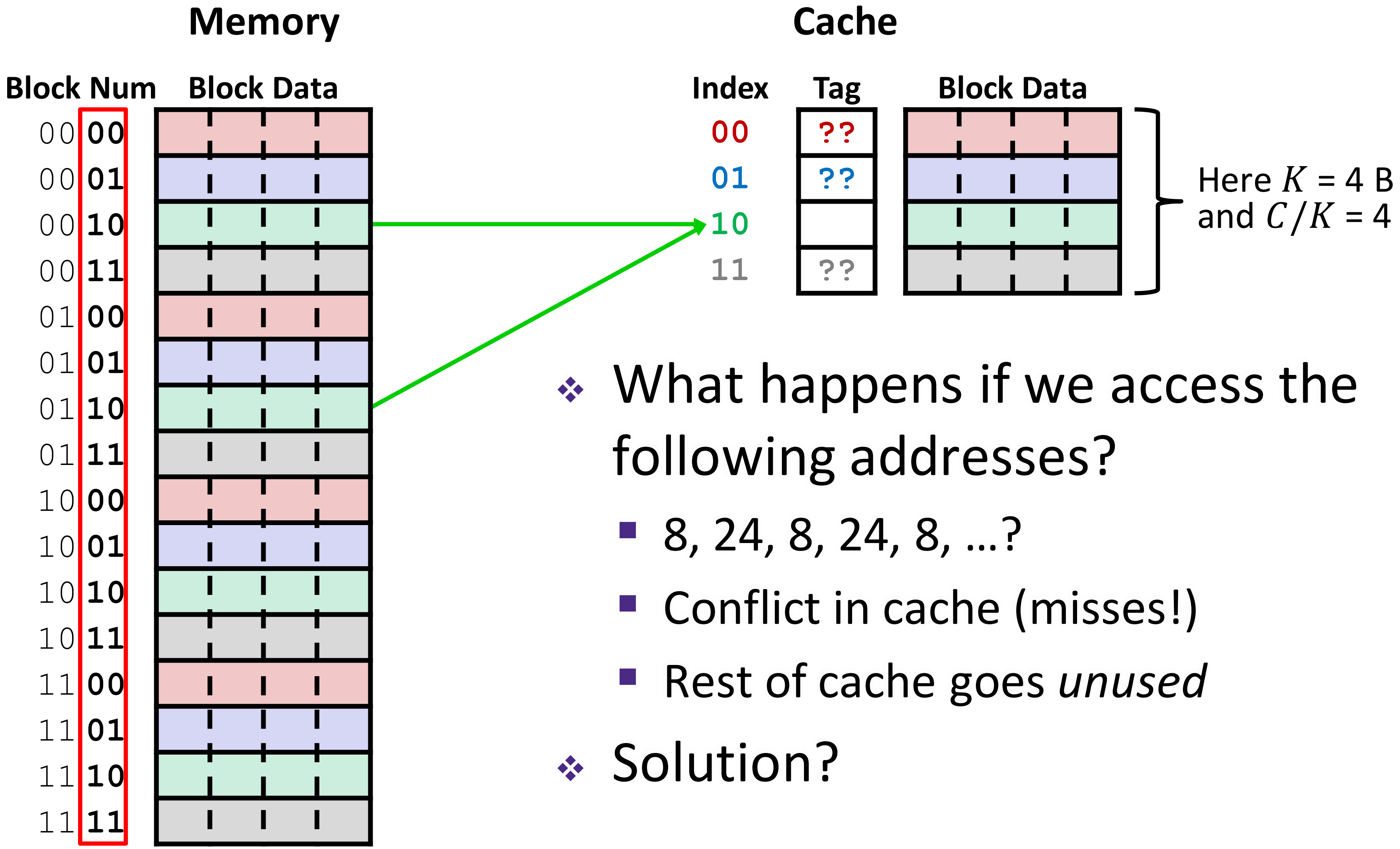


Here $K$ = 4 B and $C/K$ = 4

Hash function: (block number) mod (# of blocks in cache)

- Each memory address maps to *exactly* one index in the cache
- Fast (and simpler) to find a block

# 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?