Caches II

CSE 351 Autumn 2018

Instructor:

Justin Hsia

Teaching Assistants:

Akshat Aggarwal
An Wang
Andrew Hu
Brian Dai
Britt Henderson
James Shin
Kevin Bi
Kory Watson
Riley Germundson
Sophie Tian
Teagan Horkan

Administrivia

- Homework 4 released tomorrow (Structs, Caches)
- Lab 3 due Friday (11/9)

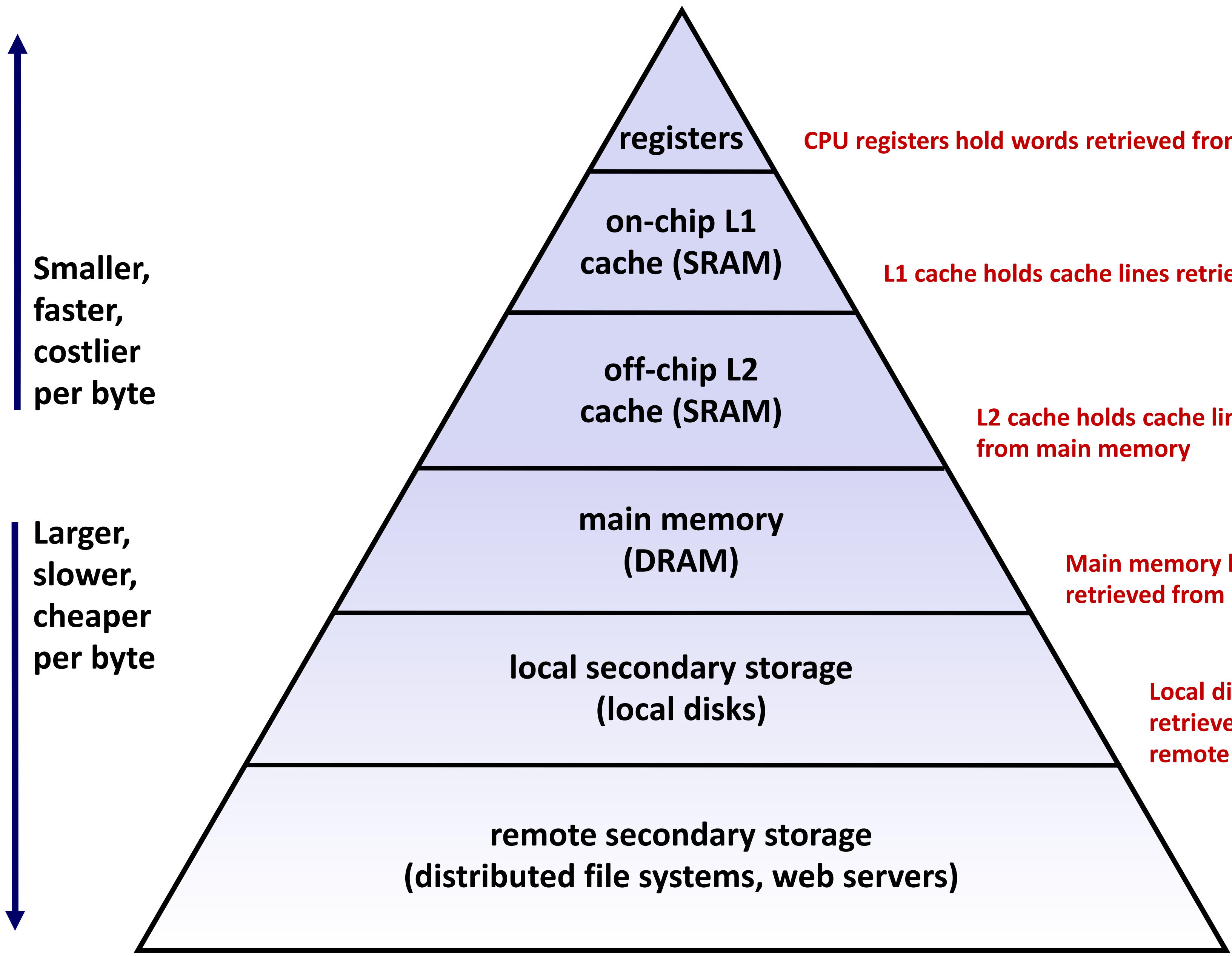
- **Mid-Quarter Survey Feedback**
  - Pace is “moderate” to “a bit too fast”
  - Canvas quiz answer keys are annoying, but instant homework feedback is great

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
on-chip L1 cache (SRAM)
off-chip L2 cache (SRAM)
main memory (DRAM)
local secondary storage (local disks)
remote secondary storage (distributed file systems, web servers)

CPU registers hold words retrieved from L1 cache

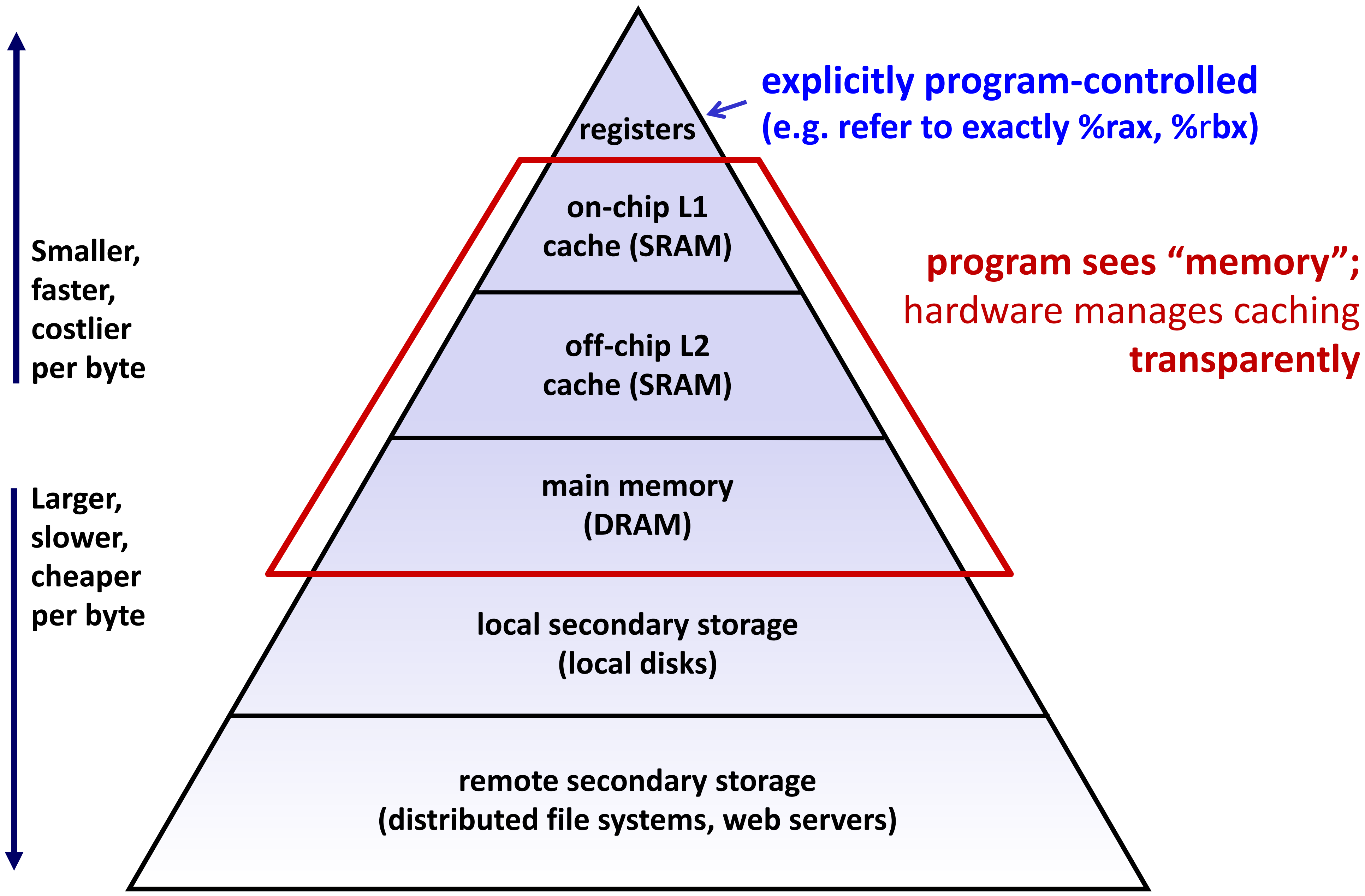
L1 cache holds cache lines retrieved from L2 cache

L2 cache holds cache lines retrieved from main memory

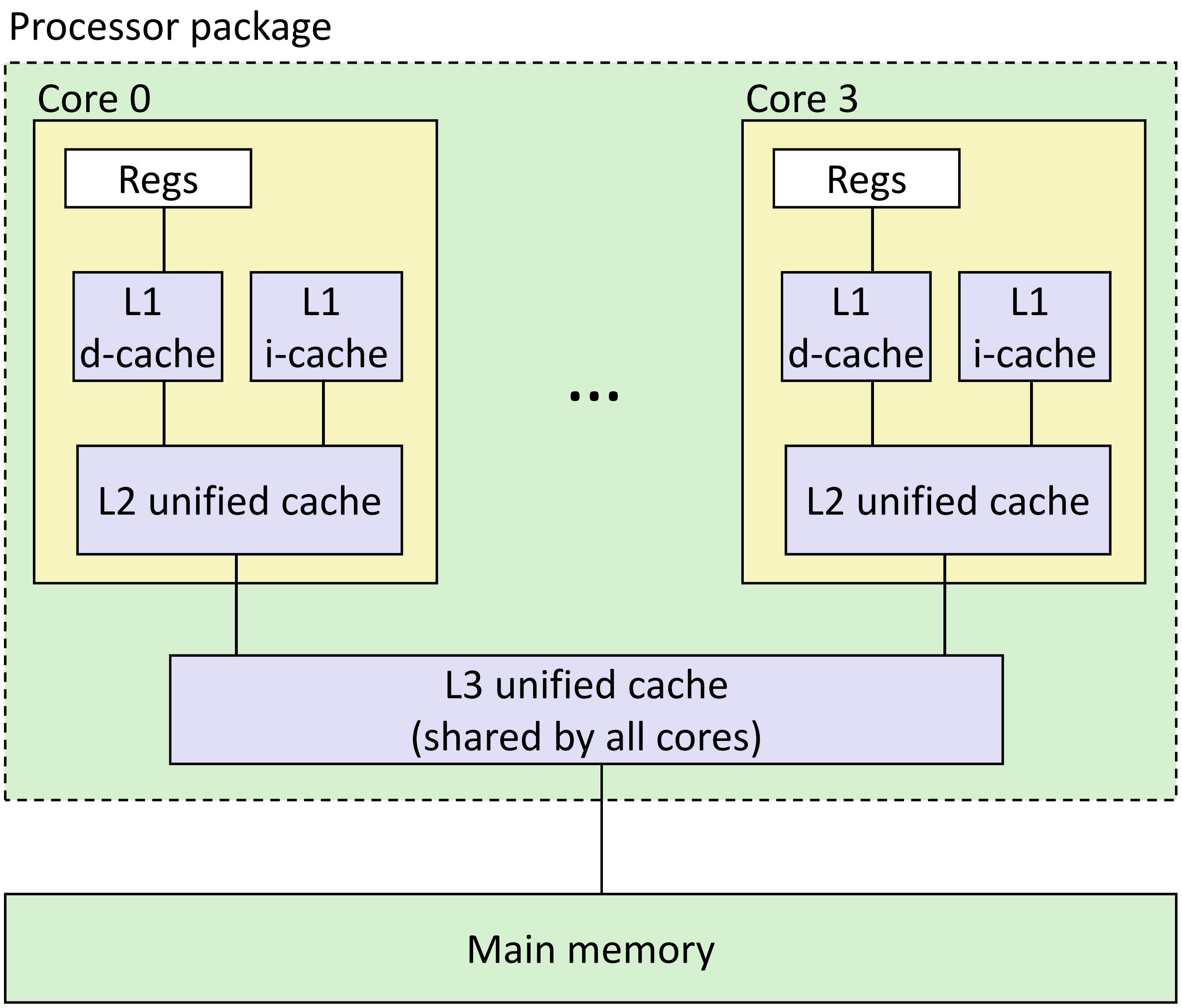
Main memory holds disk blocks retrieved from local disks

Local disks hold files retrieved from disks on remote network servers

An Example Memory Hierarchy


Smaller, faster, costlier per byte
Larger, slower, cheaper per byte
registers
explicitly program-controlled (e.g. refer to exactly %rax, %rbx)
on-chip L1 cache (SRAM)
off-chip L2 cache (SRAM)
main memory (DRAM)
program sees "memory"; hardware manages caching transparently
local secondary storage (local disks)
remote secondary storage (distributed file systems, web servers)

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

Cache Organization (1)

Note: The textbook uses "B" for block size

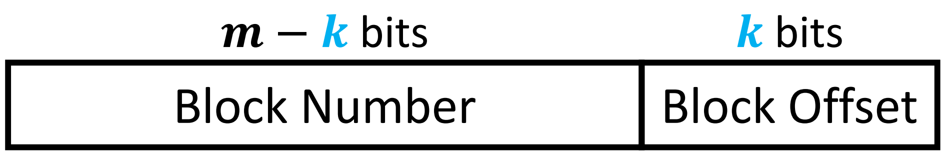
- Block Size (K): unit of transfer between \$ and Mem
  - Given in bytes and always a power of 2 (*e.g.* 64 B)
  - Blocks consist of adjacent bytes (differ in address by 1)
    - Spatial locality!

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)

$\boldsymbol{m}$-bit address:
(refers to byte in memory)



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/justinh

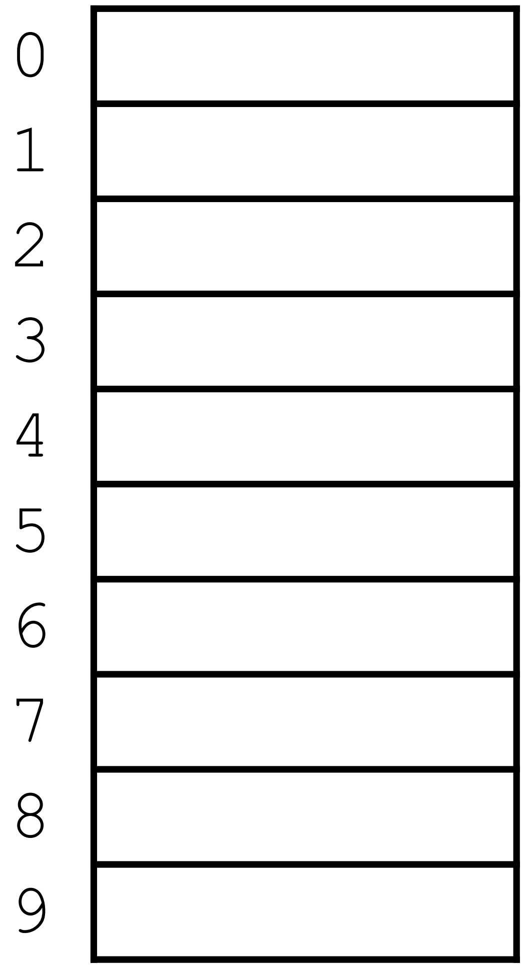
	Block Num	Block Offset
A.	1	1
B.	1	5
C.	5	1
D.	5	5
E.	We're lost…	

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)
  - 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!

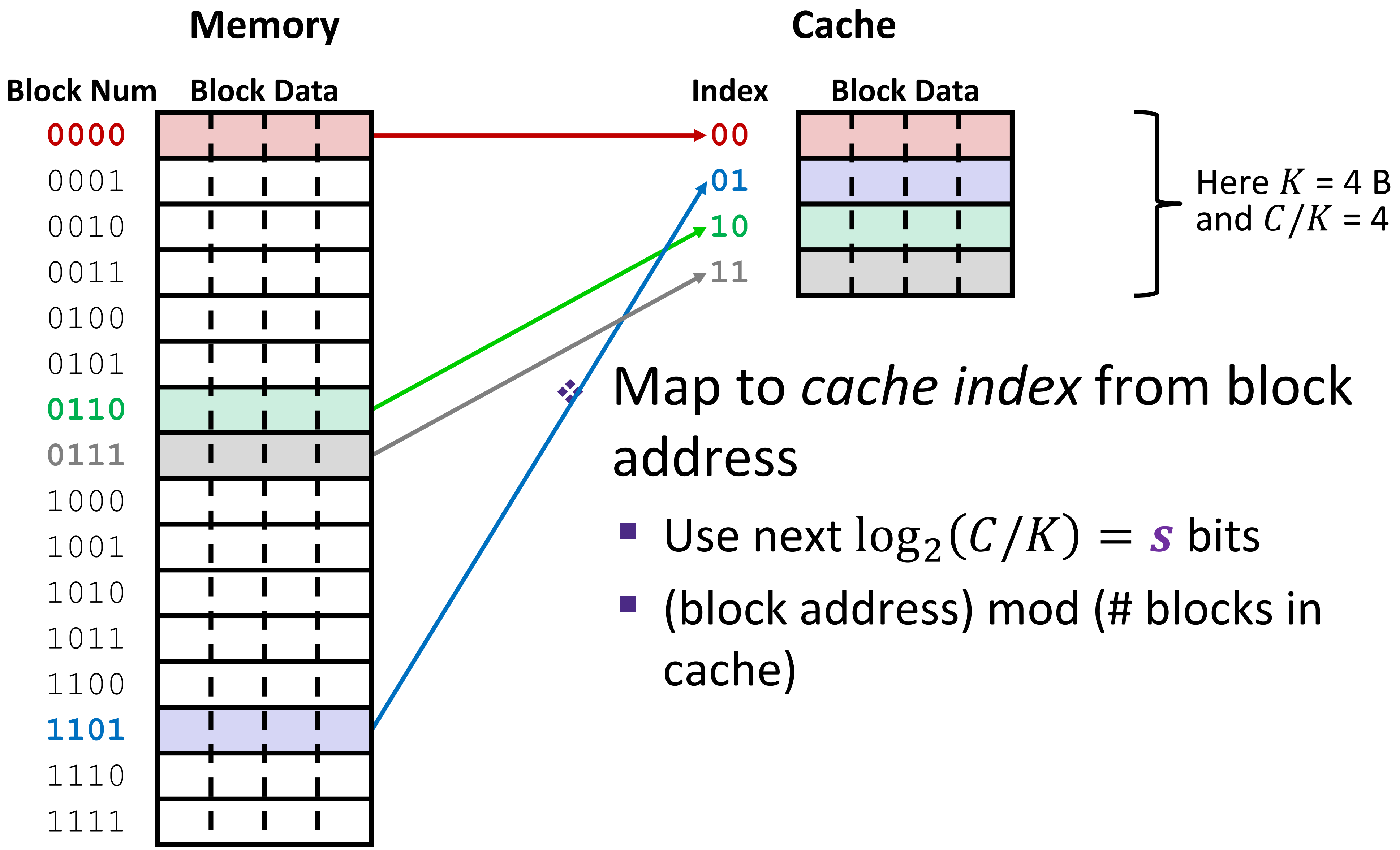
Review: Hash Tables for Fast Lookup

Insert:
5
27
34
102
119



Apply hash function to map data to “buckets”

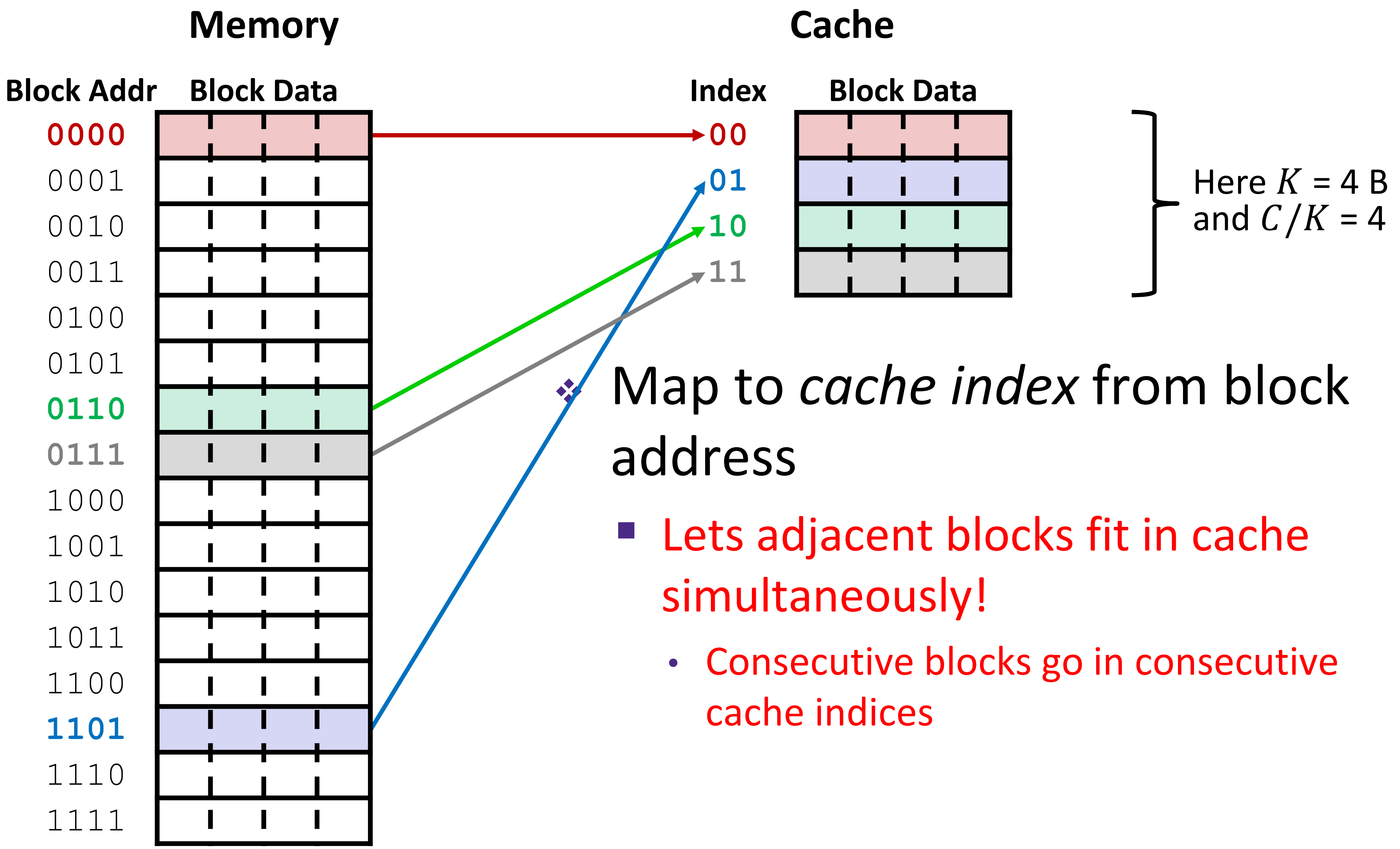
Place Data in Cache by Hashing Address



Map to *cache index* from block address

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

Place Data in Cache by Hashing Address



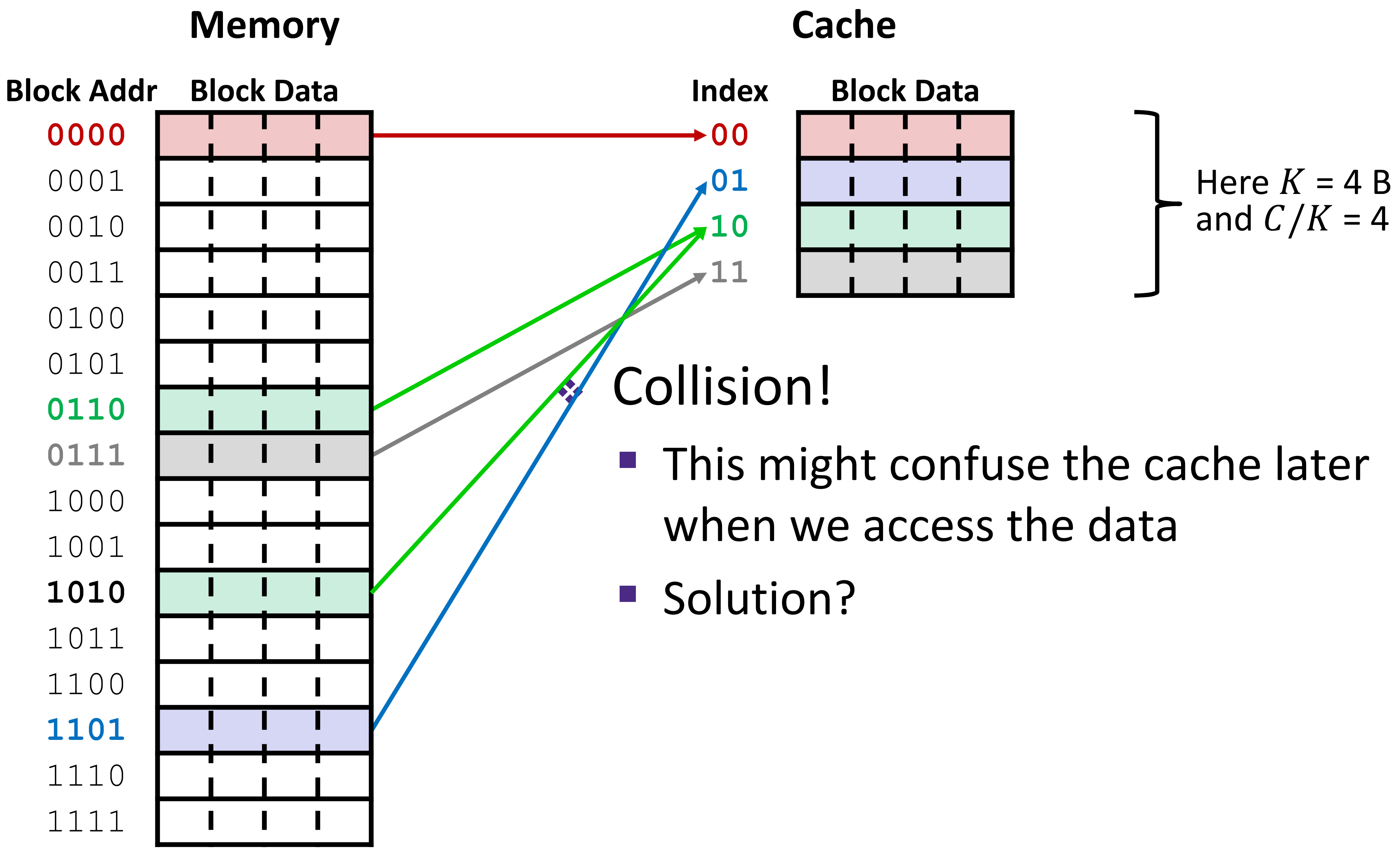
Map to *cache index* from block address

- 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

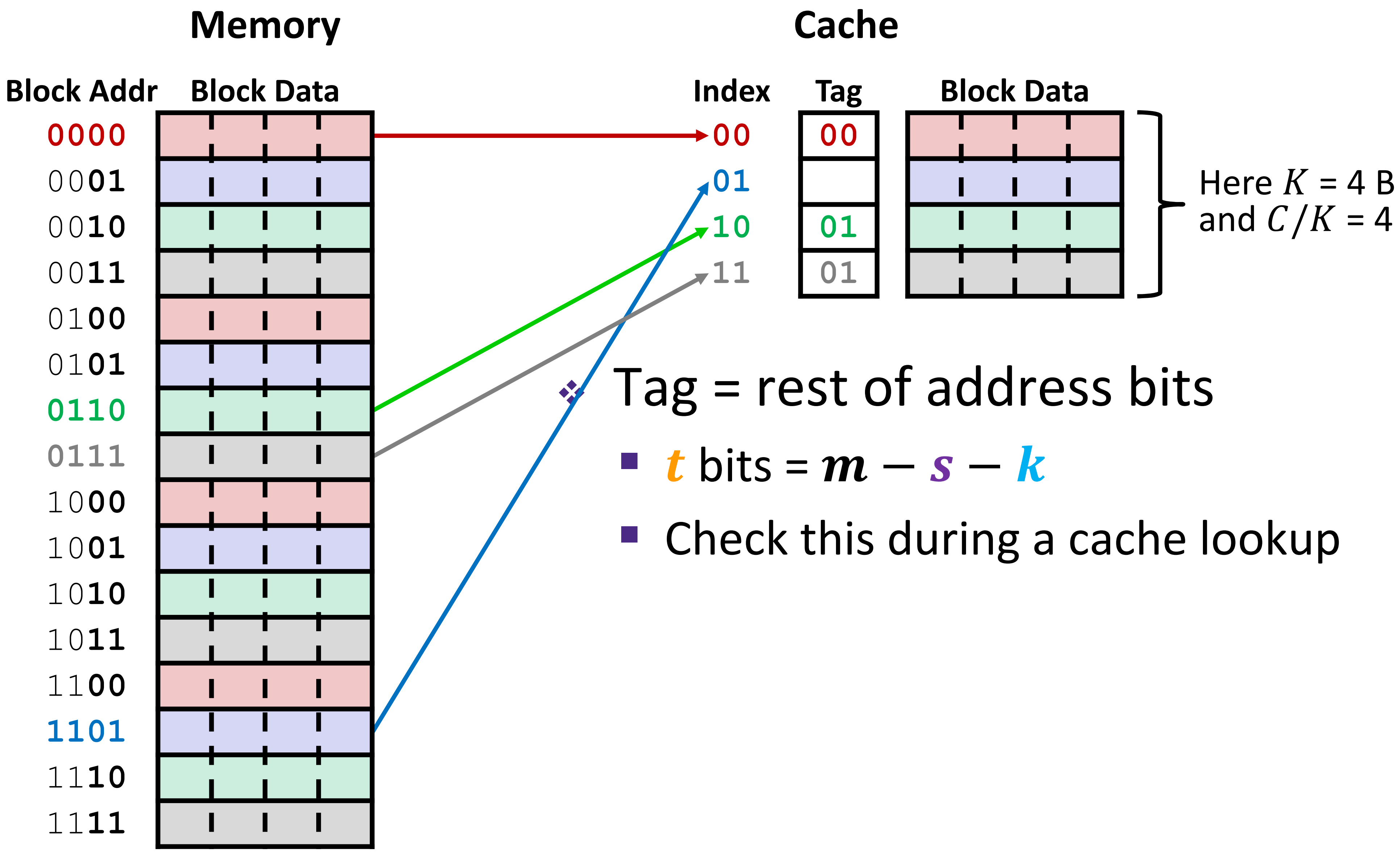
Place Data in Cache by Hashing Address



Collision!

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

Tags Differentiate Blocks in Same Index

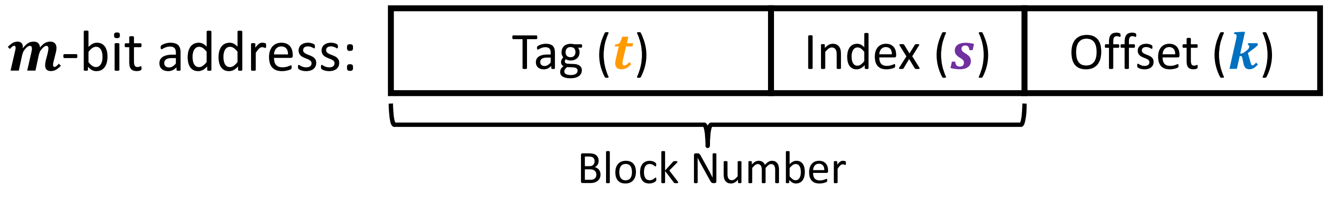


- Tag = rest of address bits
  - t bits = $m - s - k$
  - Check this during a cache lookup

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 ≠ his or her 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

Vote at http://PollEv.com/justinh

- 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

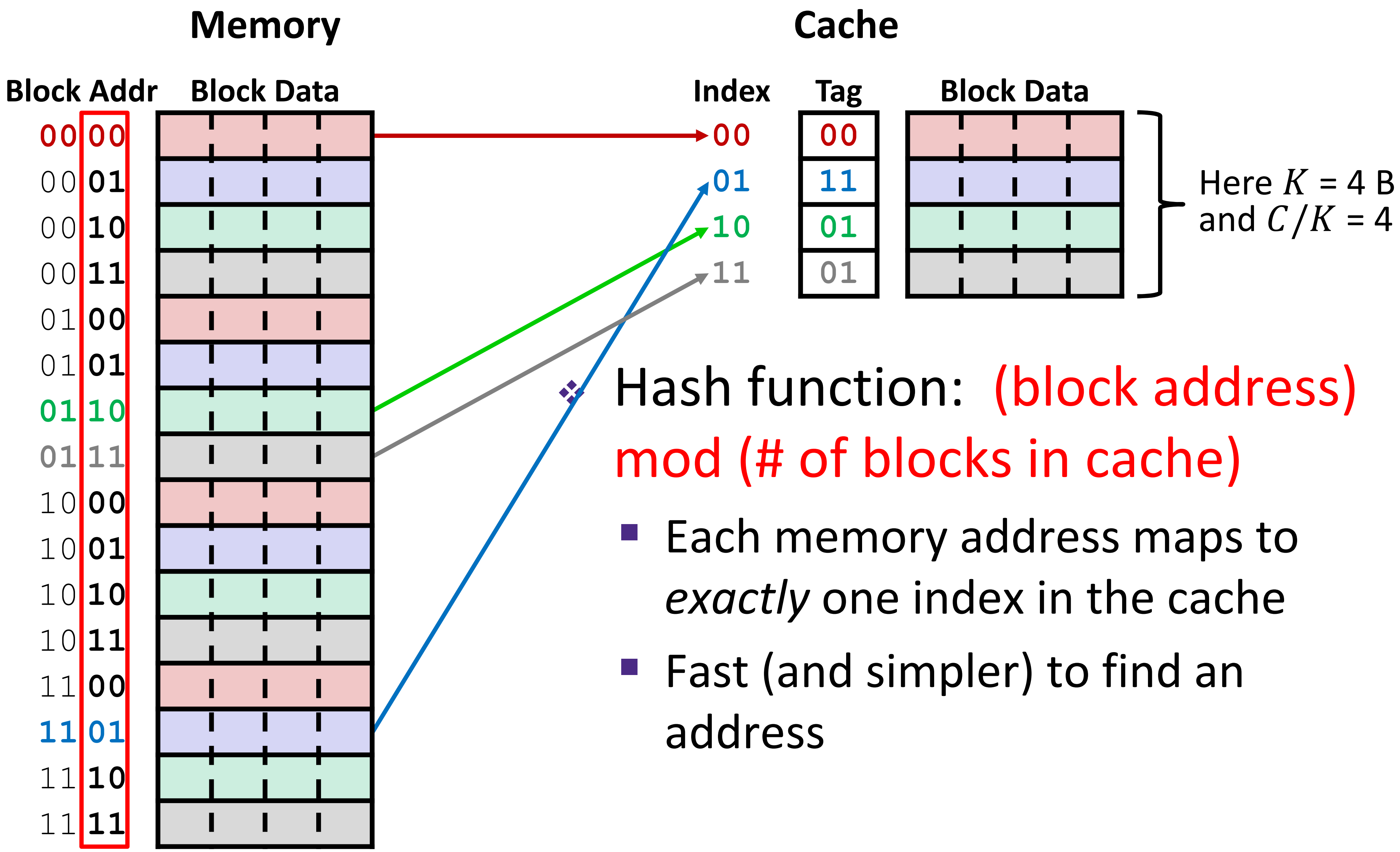
B. 8 bytes

C. 16 bytes

D. 32 bytes

E. We're lost…

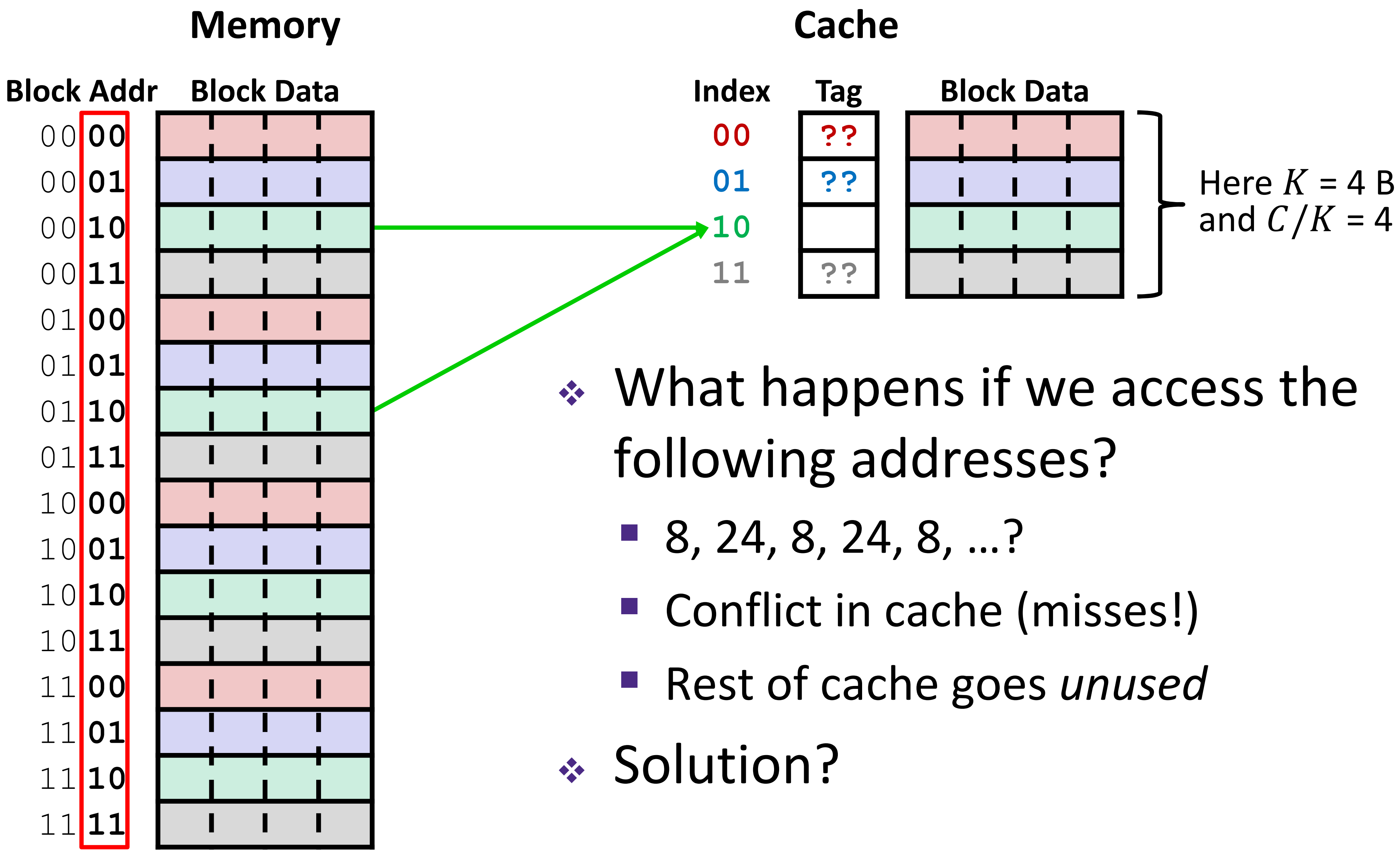
Direct-Mapped Cache



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

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

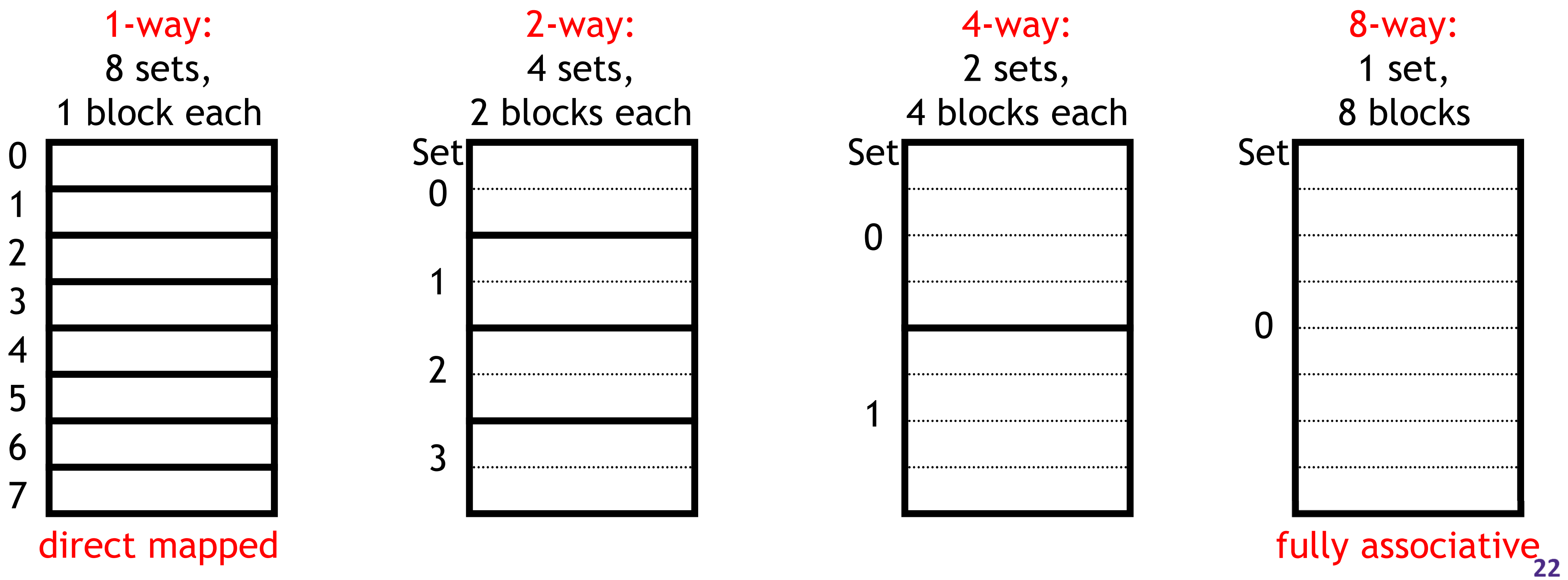
Direct-Mapped Cache Problem



- What happens if we access the following addresses?
  - 8, 24, 8, 24, 8, …?
  - Conflict in cache (misses!)
  - Rest of cache goes *unused*
- Solution?

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**


1-way:
8 sets,
1 block each
0
1
2
3
4
5
6
7
direct mapped
2-way:
4 sets,
2 blocks each
Set
0
1
2
3
4-way:
2 sets,
4 blocks each
Set
0
1
8-way:
1 set,
8 blocks
Set
0
fully associative