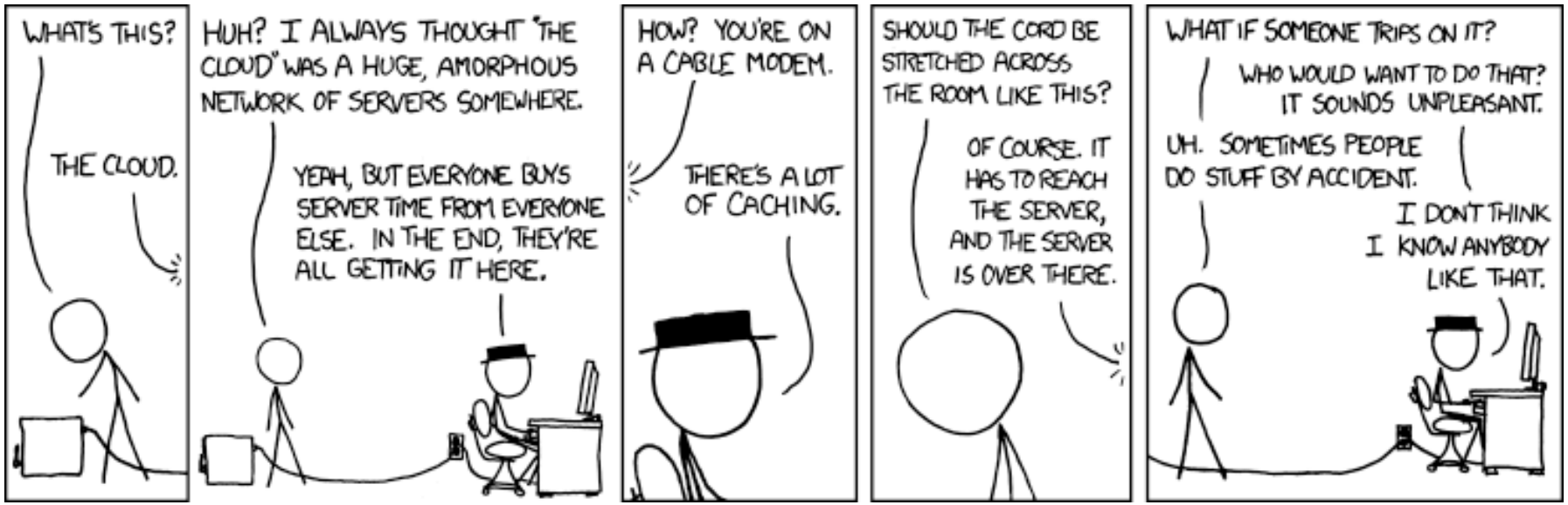
Caches II


WHAT'S THIS?
THE CLOUD.
HUH? I ALWAYS THOUGHT "THE CLOUD" WAS A HUGE, AMORPHOUS NETWORK OF SERVERS SOMEWHERE.
YEAH, BUT EVERYONE BUYS SERVER TIME FROM EVERYONE ELSE. IN THE END, THEY'RE ALL GETTING IT HERE.
HOW? YOU'RE ON A CABLE MODEM.
THERE'S A LOT OF CACHING.
SHOULD THE CORD BE STRETCHED ACROSS THE ROOM LIKE THIS?
OF COURSE. IT HAS TO REACH THE SERVER, AND THE SERVER IS OVER THERE.
WHAT IF SOMEONE TRIPS ON IT?
WHO WOULD WANT TO DO THAT? IT SOUNDS UNPLEASANT.
UH. SOMETIMES PEOPLE DO STUFF BY ACCIDENT.
I DON'T THINK I KNOW ANYBODY LIKE THAT.

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)

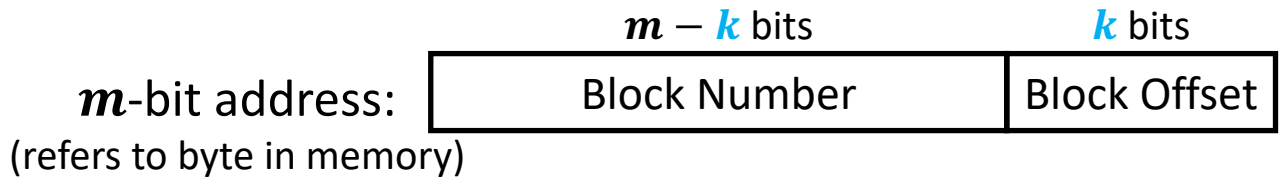
Note: The textbook uses "B" for block size

- 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)



Peer Instruction Question

- If we have 6-bit addresses and block size K = 4 B, which block and byte does 0x15 refer to?

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

Cache Organization (2)

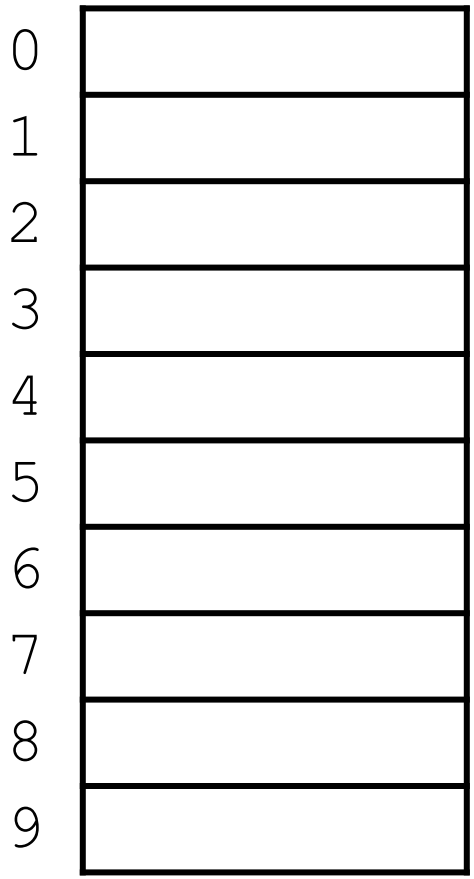
- Cache Size (C): amount of *data* the $ can store
  - Cache can only hold so much data (subset of next level)
  - Given in bytes (C) or number of blocks (C/K)
  - Example: C = 32 KiB = 512 blocks if using 64-B blocks
- Where should data go in the cache?
  - We need a mapping from memory addresses to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

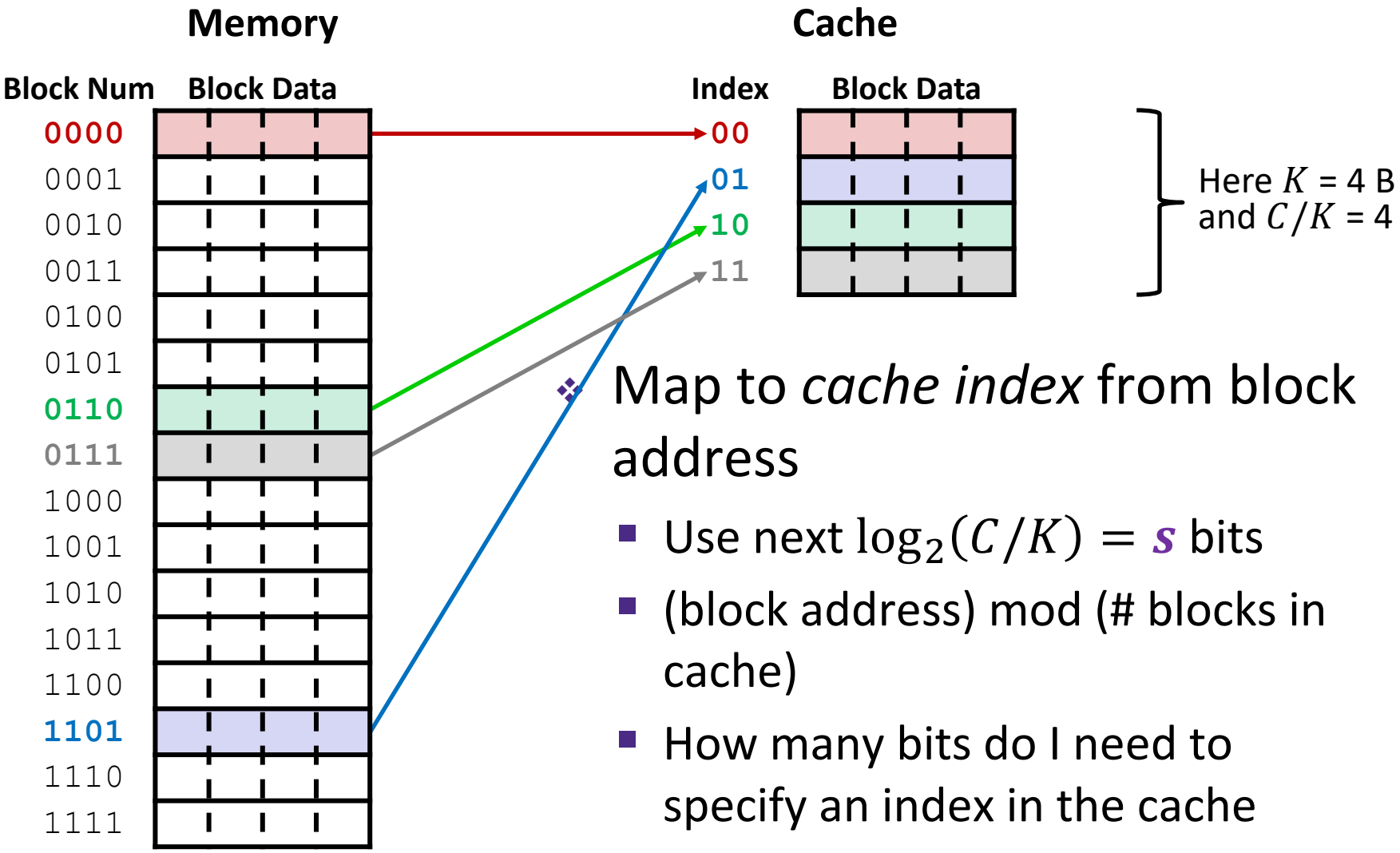
Insert:

5
27
34
102
119

Apply hash function to map data to “buckets”



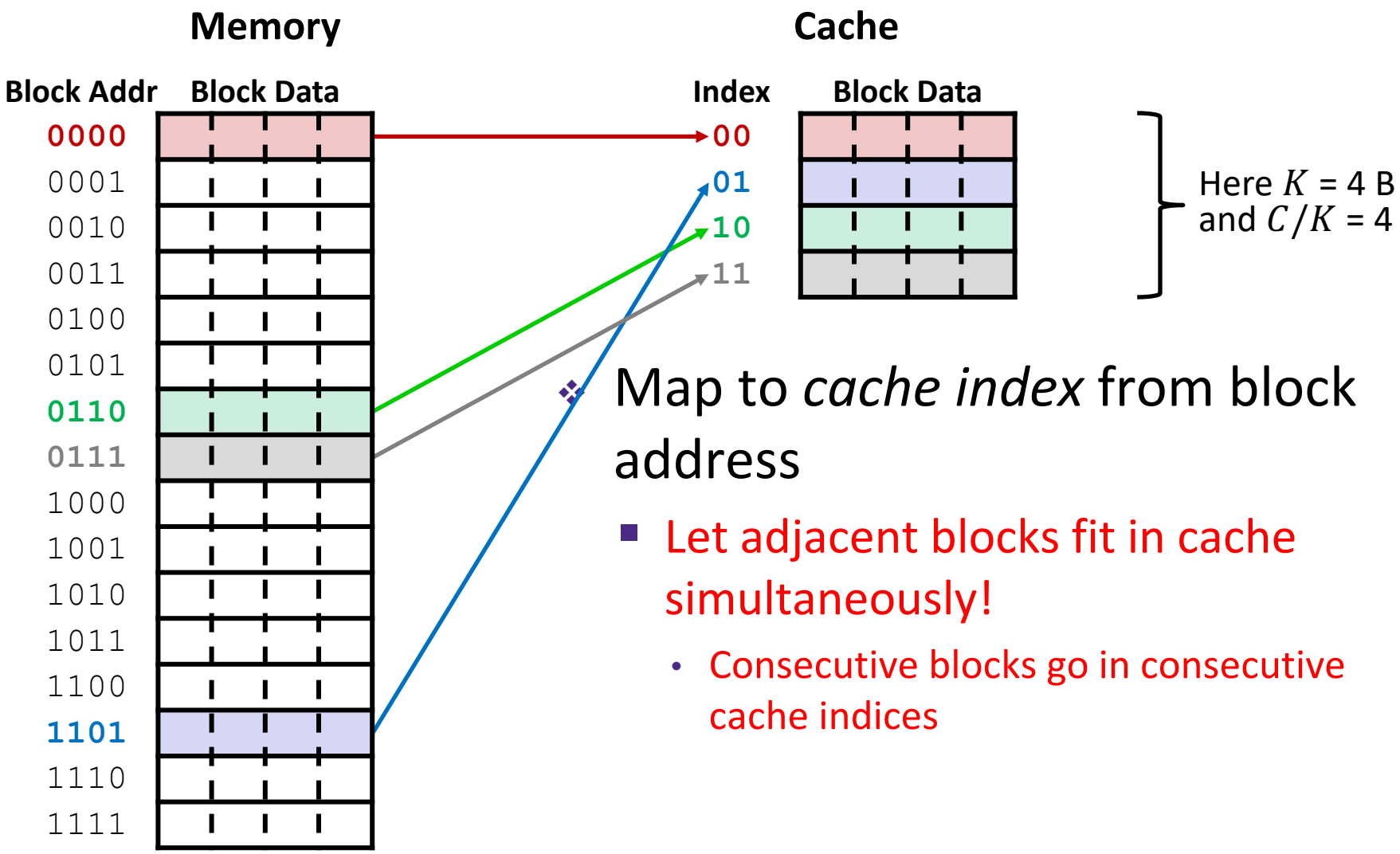
Place Data in Cache by Hashing Address



Map to *cache index* from block address

- Use next $\log_2(C/K) = \boldsymbol{s}$ bits
- (block address) mod (# blocks in cache)
- How many bits do I need to specify an index in the cache

Place Data in Cache by Hashing Address



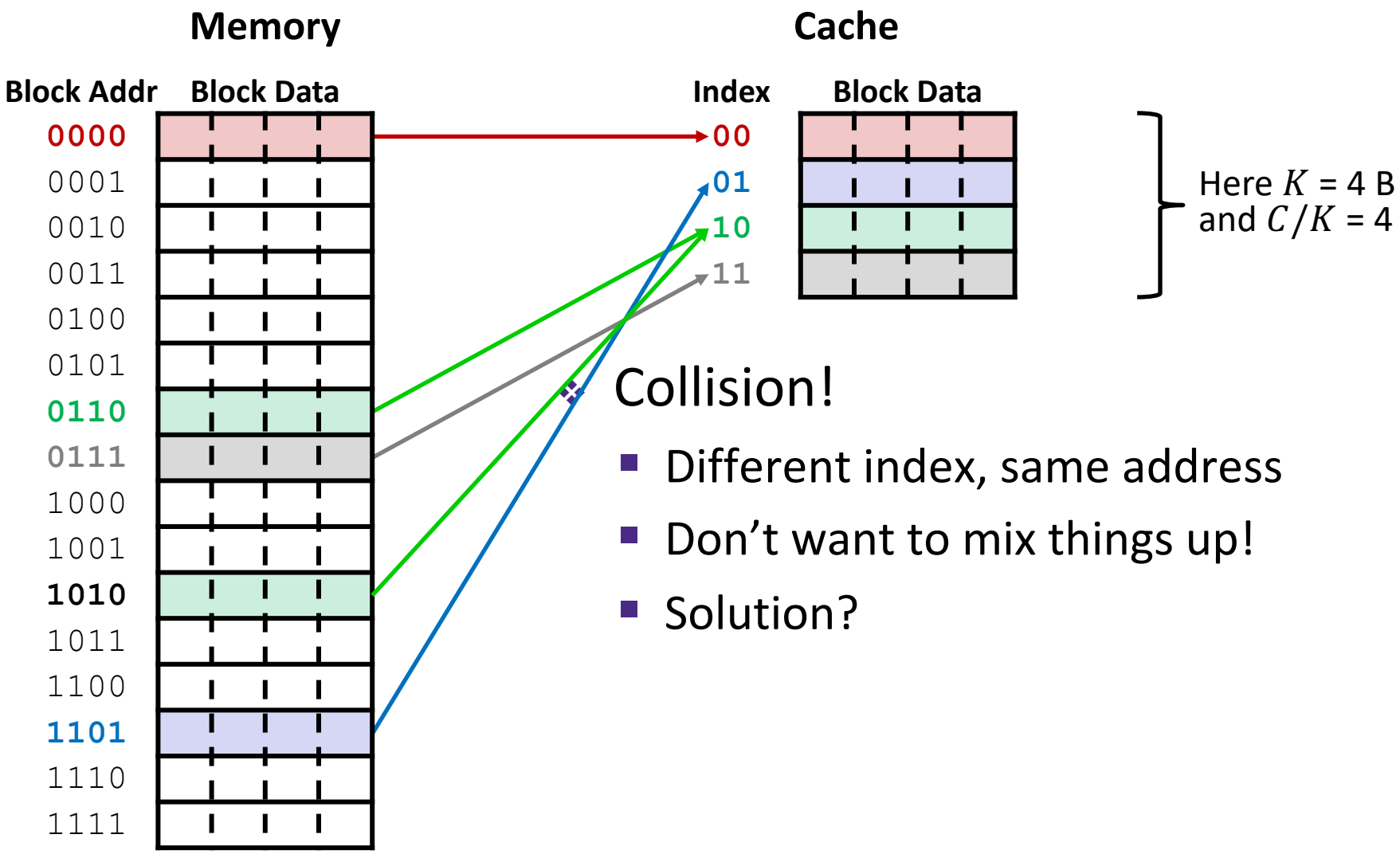
Map to *cache index* from block address

- Let 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?

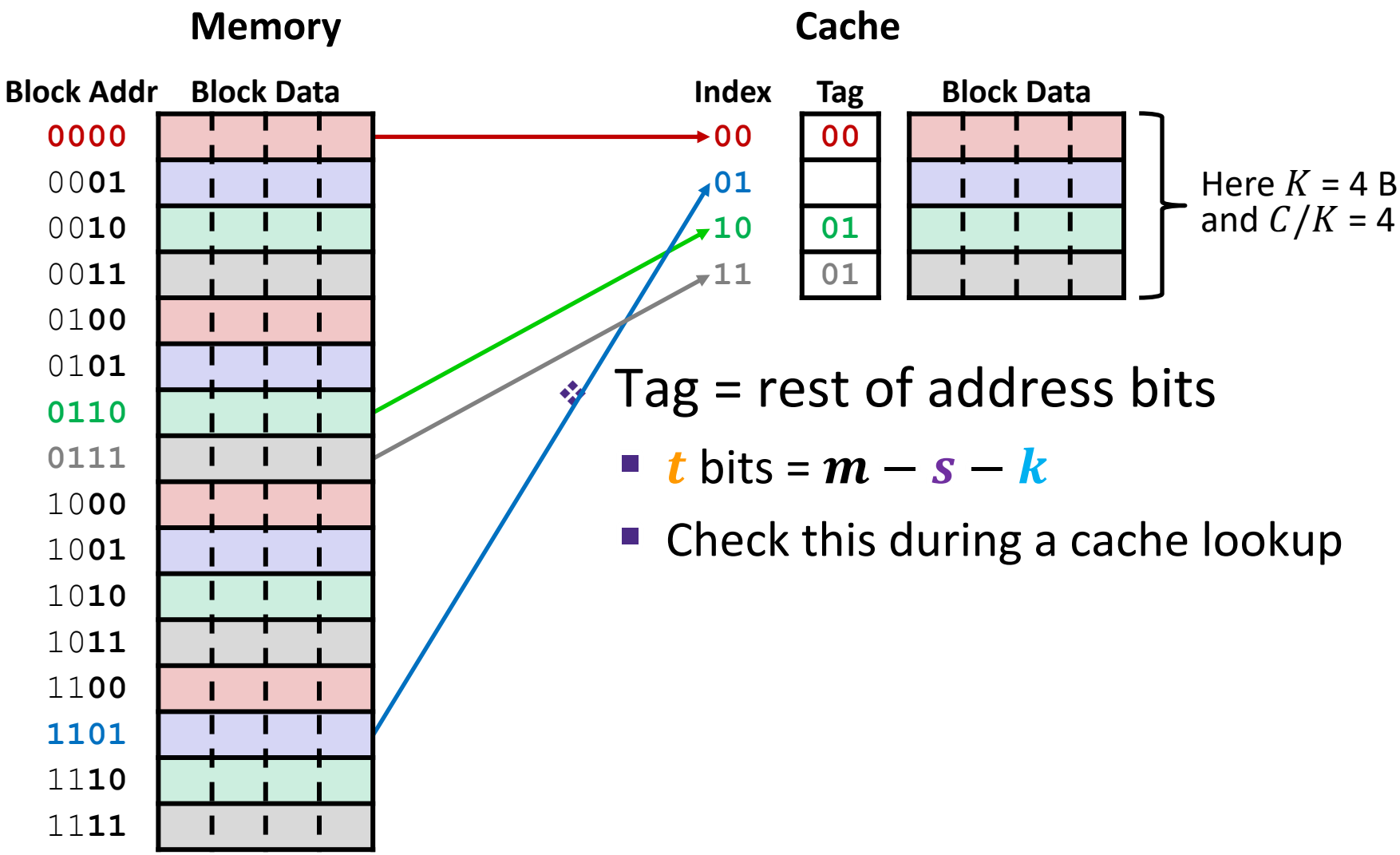
Place Data in Cache by Hashing Address



Collision!

- Different index, same address
- Don't want to mix things up!
- Solution?

Tags Differentiate Blocks in Same Index

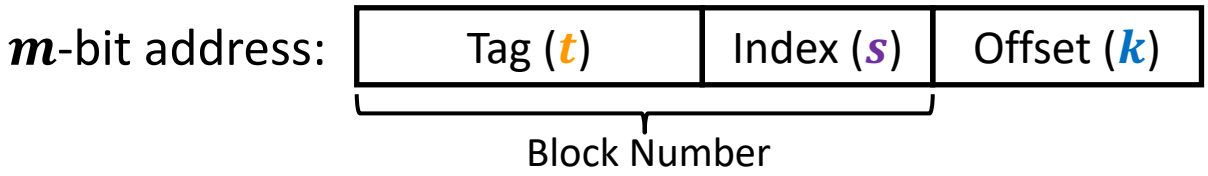


- Tag = rest of address bits
  - t bits = $m - s - k$
  - Check this during a cache lookup

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

- 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

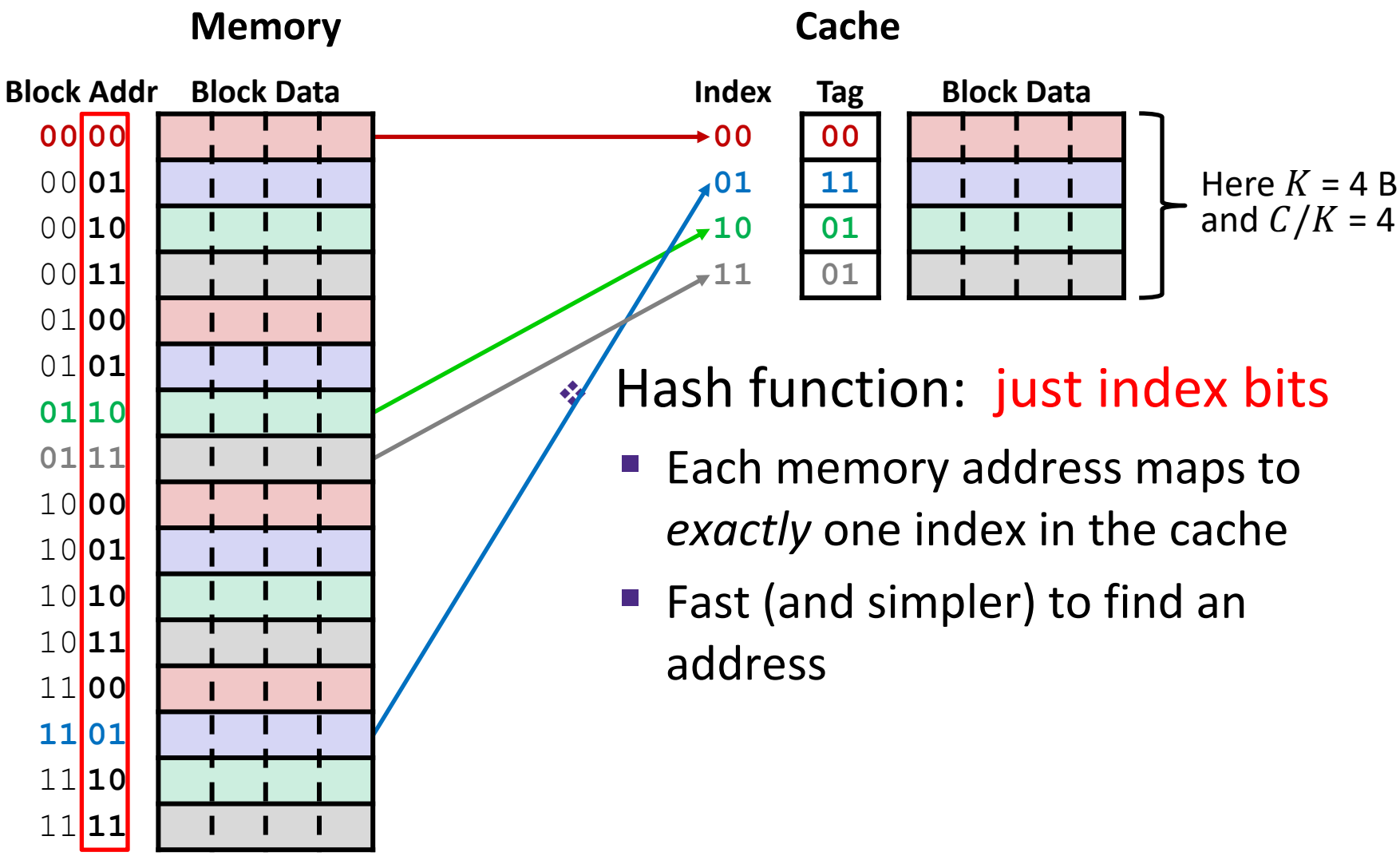
B. 8 bytes

C. 16 bytes

D. 32 bytes

E. We're lost…

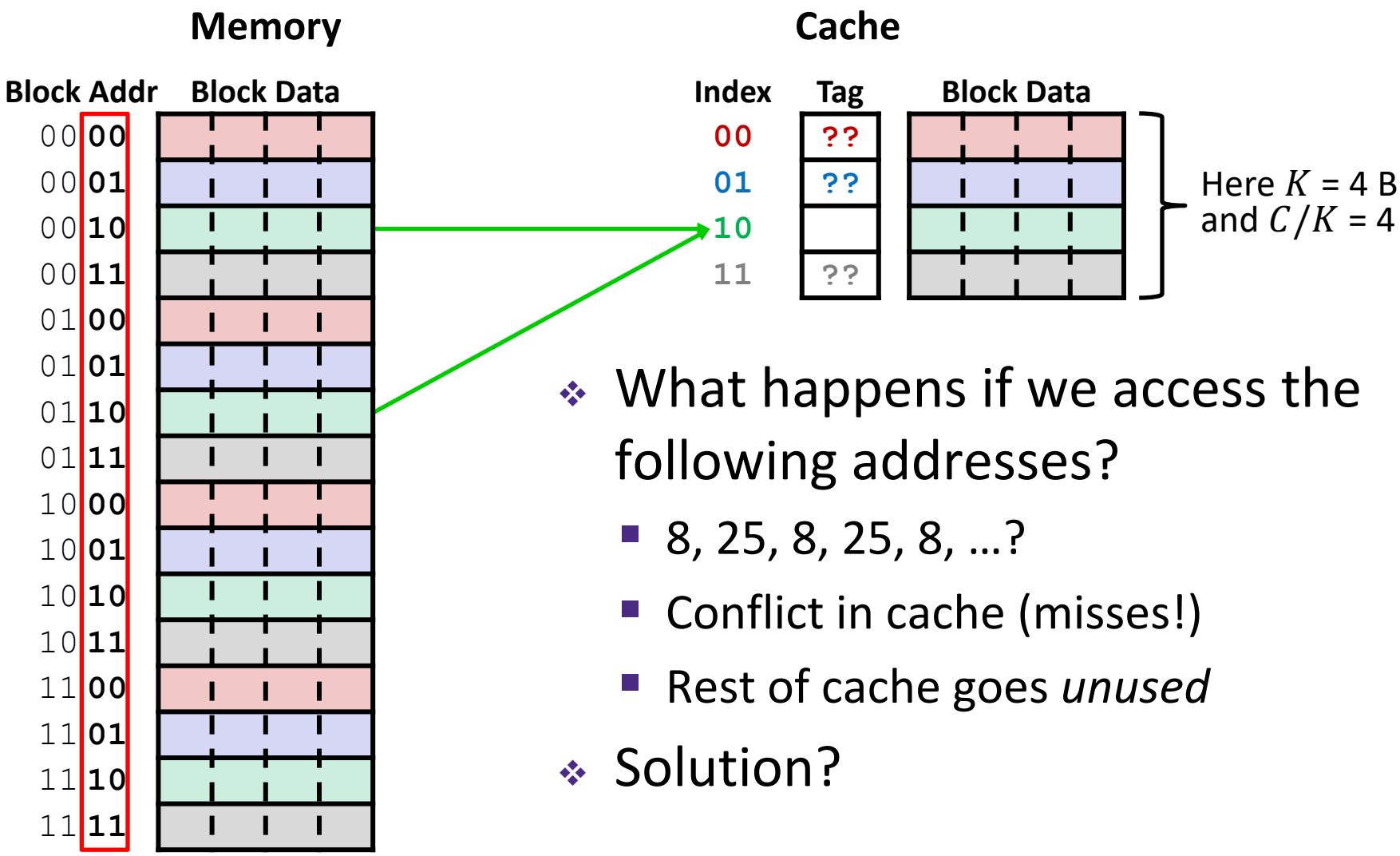
Direct-Mapped Cache



Hash function: just index bits

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

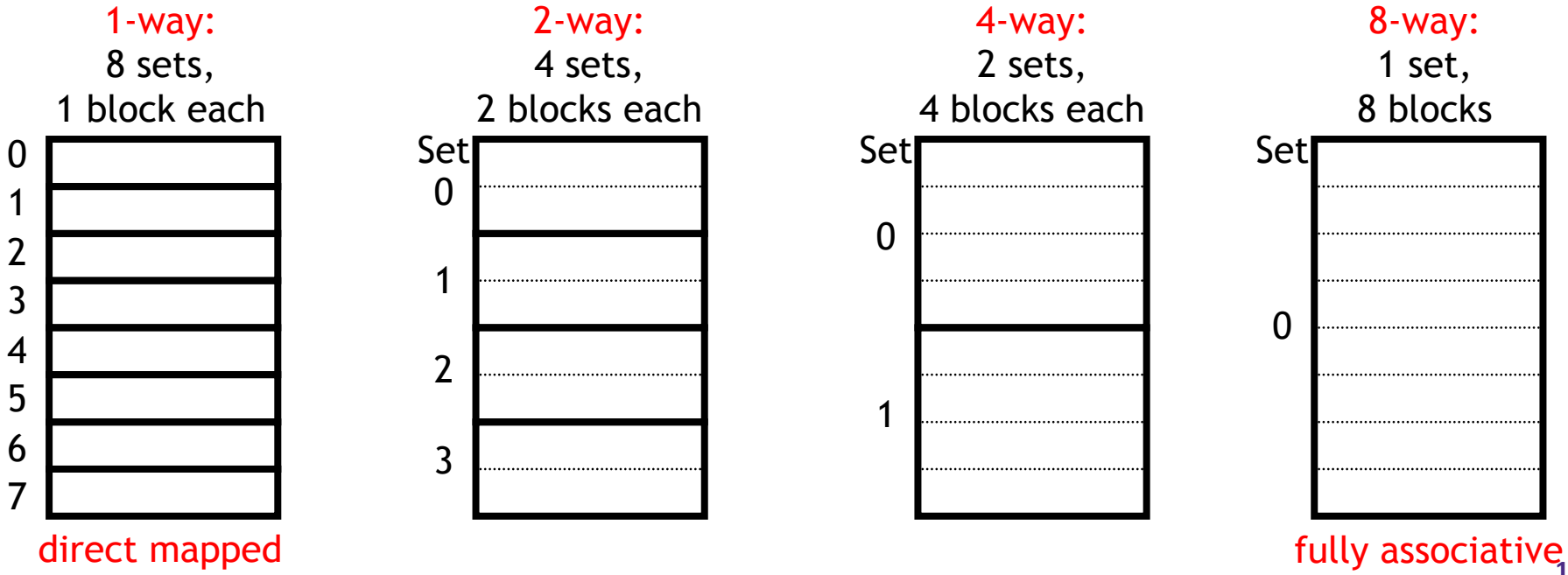
Direct-Mapped Cache Problem



- What happens if we access the following addresses?
  - 8, 25, 8, 25, 8, …?
  - Conflict in cache (misses!)
  - Rest of cache goes *unused*
- Solution?

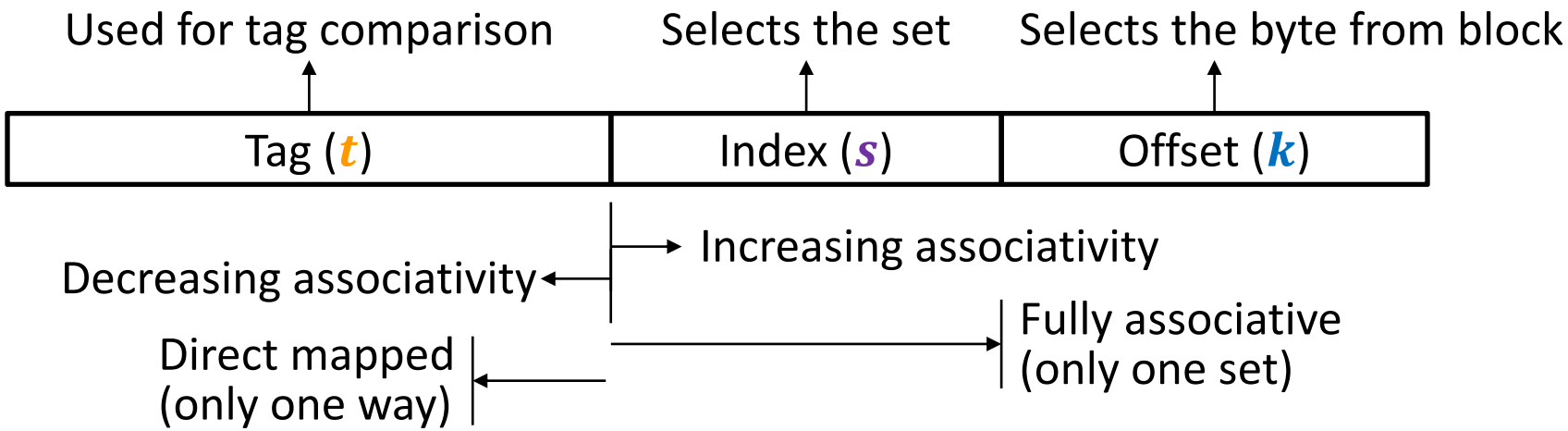
Associativity

- 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**



Cache Organization (3)

- Associativity (E): # of ways for each set
  - Such a cache is called an "*E-way set associative cache*"
  - We now index into cache *sets*, of which there are $S = C/K/E$
  - Use lowest $\log_2(C/K/E)$ = $\boldsymbol{s}$ bits of block address
    - Direct-mapped: E = 1, so $\boldsymbol{s}$ = $\log_2(C/K)$ as we saw previously
    - Fully associative: $E = C/K$, so $\boldsymbol{s}$ = 0 bits



Example Placement

block size:	16 B
capacity:	8 blocks
address:	16 bits

- Where would data from address `0x1833` be placed?
  - Binary: `0b 0001 1000 0011 0011`

$t = m-s-k$ $s = \log_2(C/K/E)$ $k = \log_2(K)$

m-bit address:

s = ?
Direct-mapped

Set	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

s = ?
2-way set associative

Set	Tag	Data
0		
1		
2		
3		

s = ?
4-way set associative

Set	Tag	Data
0		
1		

Block Replacement

- *Any* empty block in the correct set may be used to store block
- If there are no empty blocks, which one should we replace?
  - No choice for direct-mapped caches
  - Caches typically use something close to ***least recently used (LRU)*** (hardware usually implements "*not most recently used*")

Peer Instruction Question

- We have a cache of size 2 KiB with block size of 128 B. If our cache has 2 sets, what is its associativity?

  A. 2

  B. 4

  C. 8

  D. 16

  E. We're lost…

- If addresses are 16 bits wide, how wide is the Tag field?