The Hardware/Software Interface

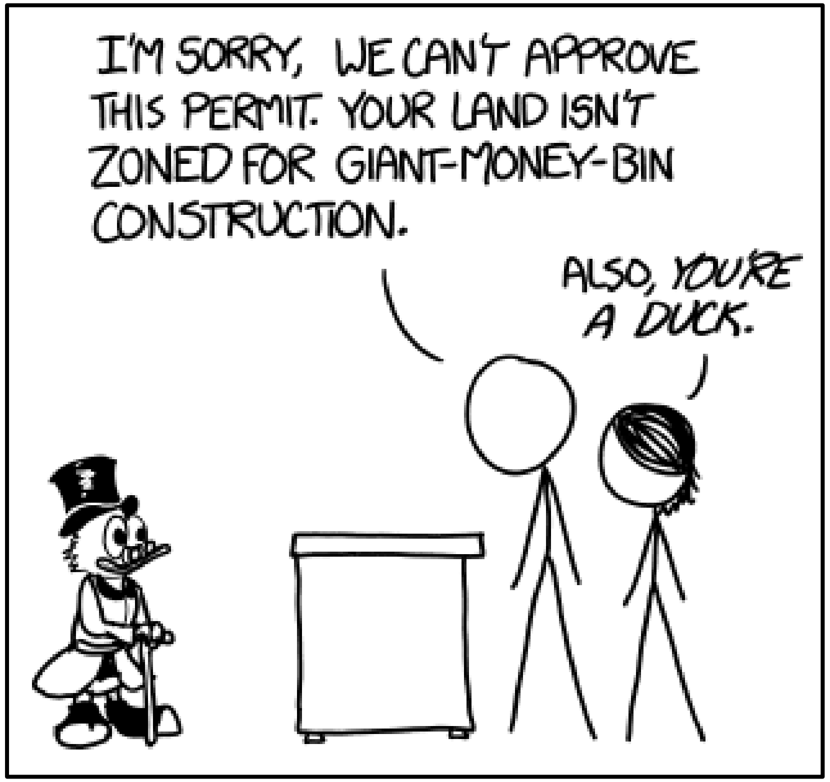
Memory & Caches II

Instructors:

Amber Hu, Justin Hsia

Teaching Assistants:

Anthony Mangus
Grace Zhou
Jiuyang Lyu
Kurt Gu
Mendel Carroll
Naama Amiel
Rose Maresh
Violet Monserate
Divya Ramu
Jessie Sun
Kanishka Singh
Liander Rainbolt
Ming Yan
Pollux Chen
Soham Bhosale



https://what-if.xkcd.com/111/

Relevant Course Information

- HW 16 due Wednesday (11/5)
- HW 17 due Friday (11/7)
  - Don’t wait too long, this is a ***big*** HW (includes this lecture)

- Lab 3 due Friday (11/7)
  - Late days open until Monday (11/10)
  - Monday is Veteran’s Day
  - No lecture, but some office hours (see Ed)

- Midterm grades will be released when we can
  - Regrade requests will be available afterward

Lecture Outline (1/2)

- **Blocks and Addresses**
- Direct-Mapped Caches

Cache Sizes (Review)

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!

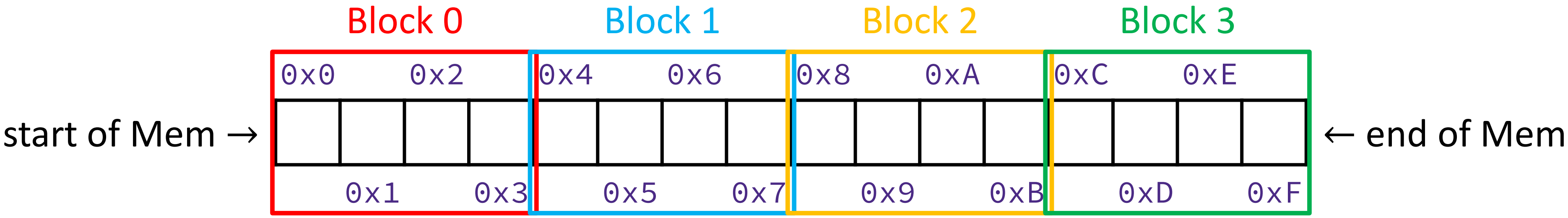
- Cache Size (C): amount of *data* the \$ can store
  - Cache can only hold so much data (subset of next level)
  - Given in bytes (C) or number of blocks (C/K)
  - Example: C = 32 KiB using 64-B blocks means that it can hold ______ blocks

Memory and Blocks (Review)

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!

- The address space is divided into adjacent, numbered blocks
  - For address A of width m bits, can determine its block number via $\lfloor A/K \rfloor$
  - Example: Let m = 4 bits, $K = 4$ B



Addresses and Blocks (Review)

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!

- Remaining address bits tell you ordered position of byte within the block
  - (address) modulo (# of bytes in a block) = A % K
  - Offset field is the low-order $\log_2(K) = \boldsymbol{k}$ bits of address
    - mod 2^n = n lowest bits

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

Addresses and Blocks Example

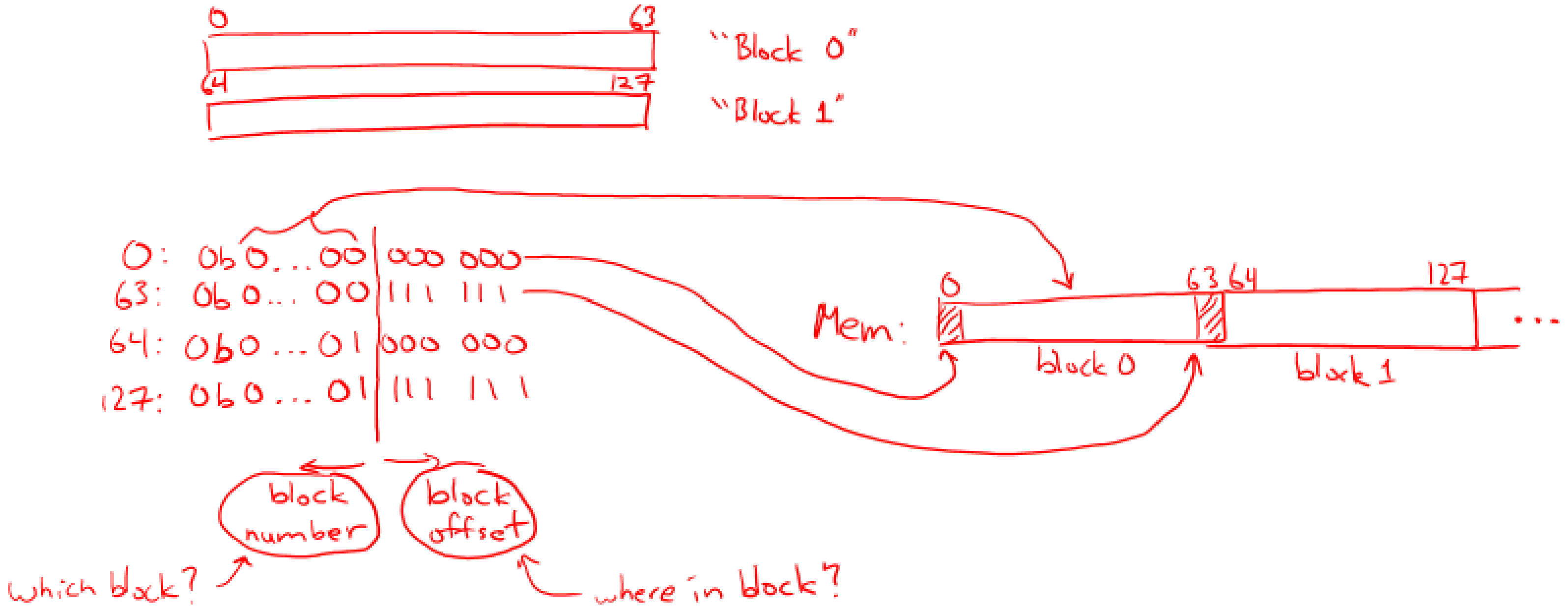
Note: The textbook uses "b" for offset bits

- 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 **`0x19`** refer to?

Lab 1a Callback

Note: The textbook uses "B" for block size

- The `within_same_block` function asks you to determine if the addresses stored by two pointers lie within the *same block of 64-byte aligned memory*
  - Solution: Right shift by 6 and compare



Polling Questions (1/2)

- We have a 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

Lecture Outline (2/2)

- Blocks and Addresses
- **Direct-Mapped Caches**

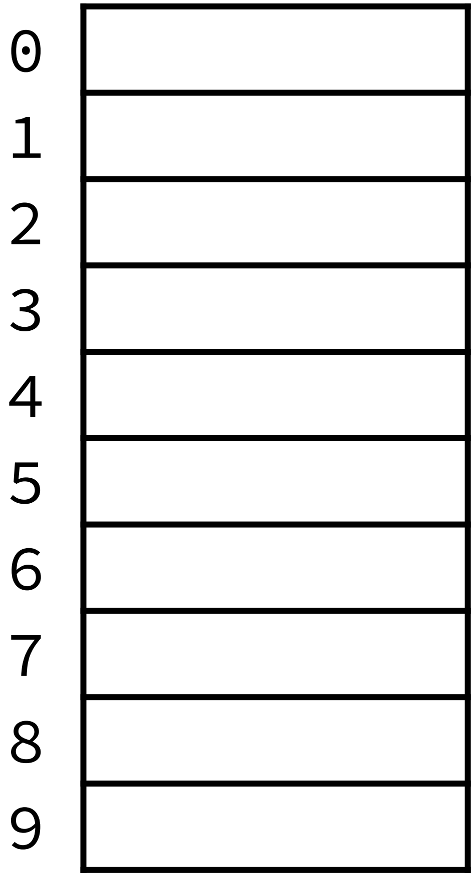
Block Placement (Review)

- 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!

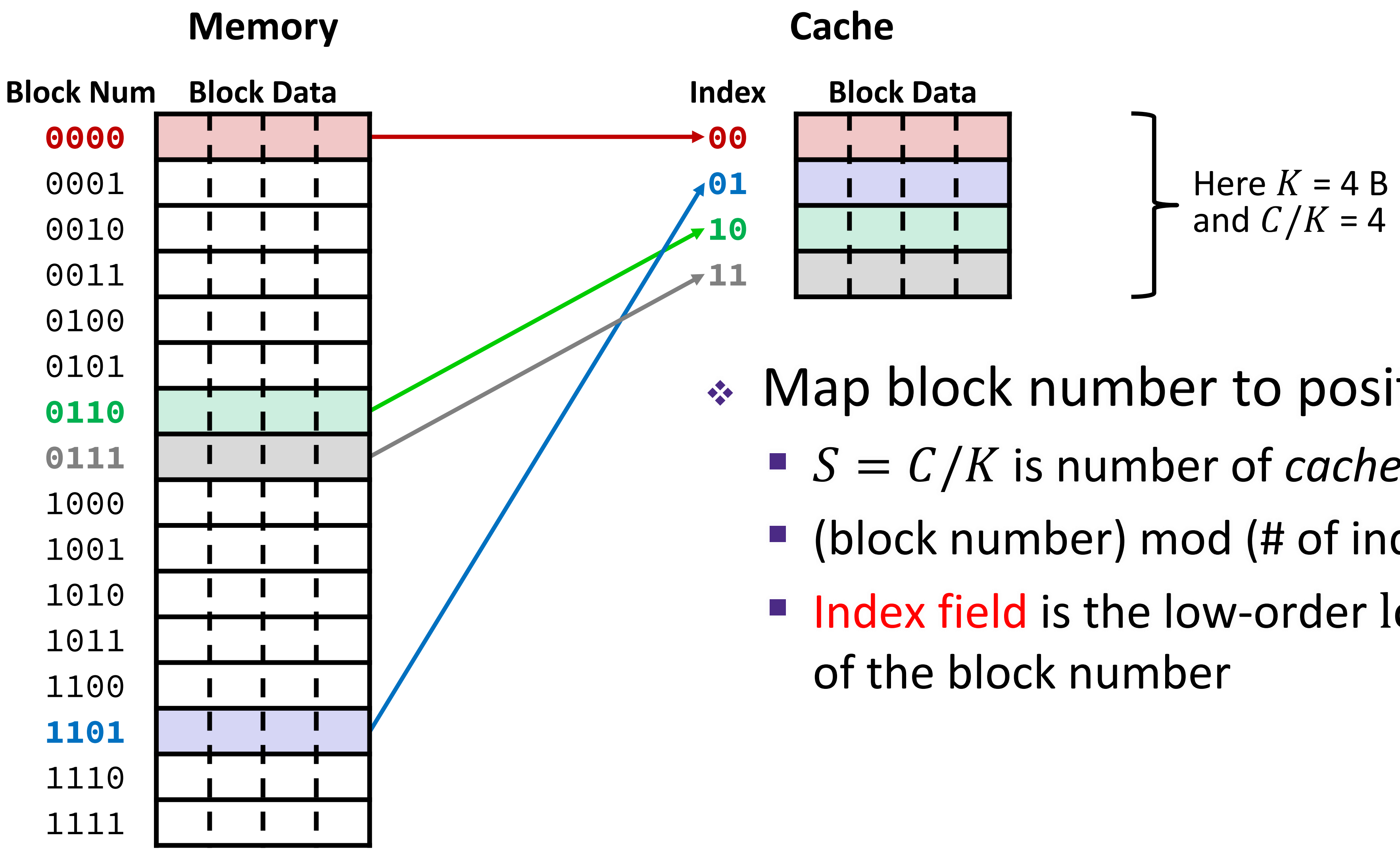
Hash Tables for Fast Lookup

- Apply hash function to map data to “buckets”
  - **Goals:** (1) fast/simple calculation
    (2) use all buckets “well”

Insert:
```
  5
 27
 34
102
119
```

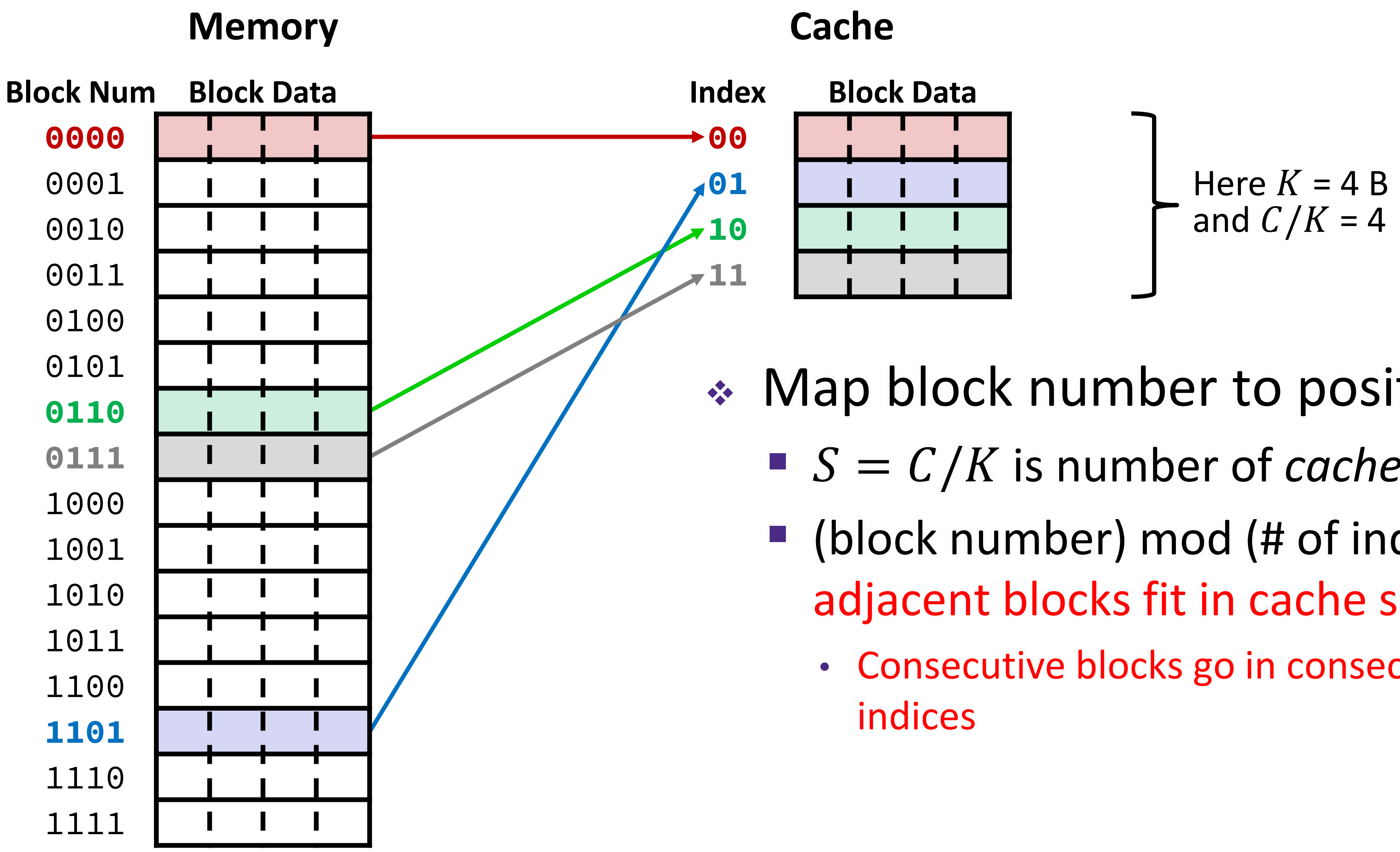


Direct-Mapped Address Hashing (Review)



- Map block number to position in cache
  - $S = C/K$ is number of *cache indices*
  - (block number) mod (# of indices)
  - Index field is the low-order $\log_2(S) = \boldsymbol{s}$ bits of the block number

Direct-Mapped Address Hashing Consequence



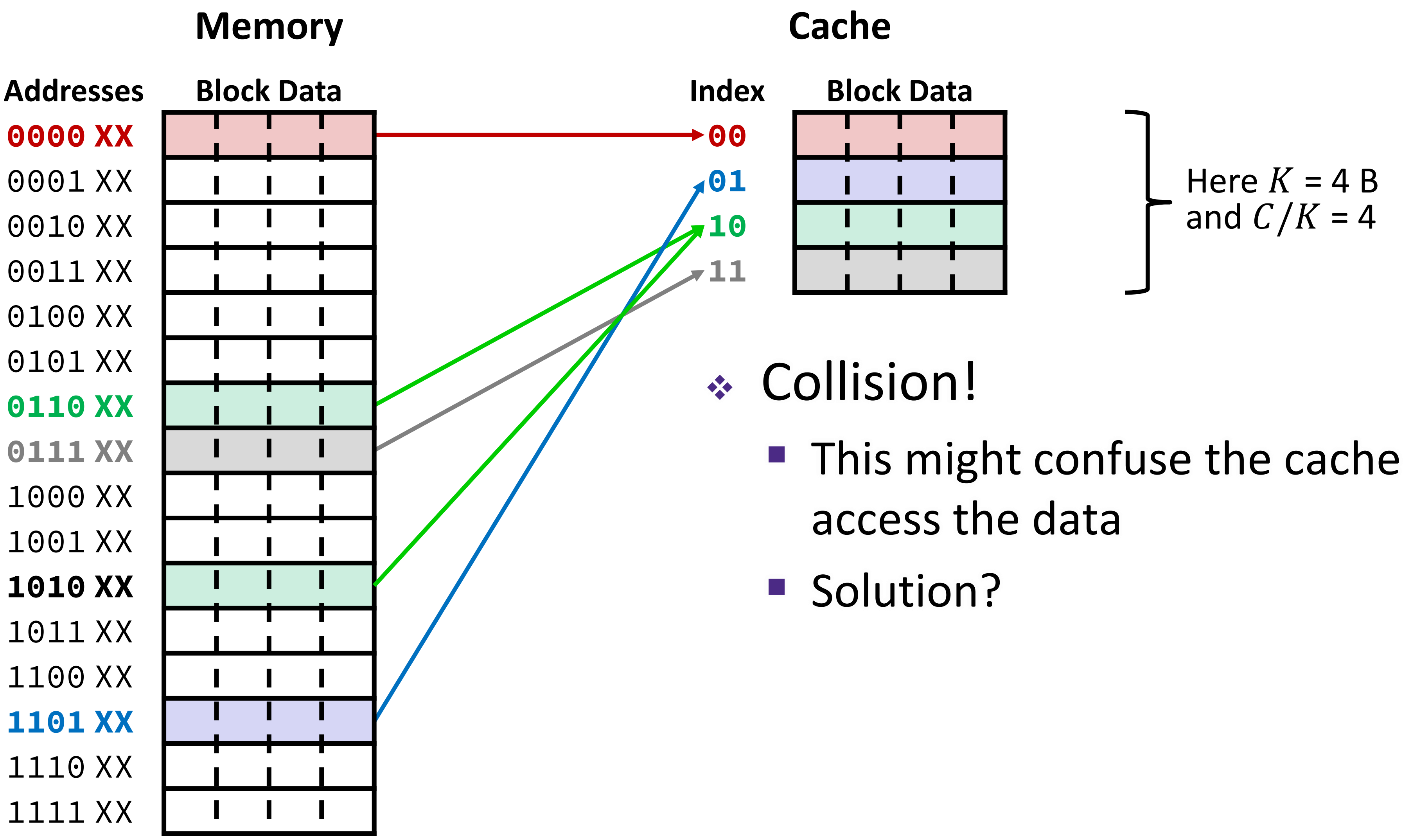
- Map block number to position in cache
  - $S = C/K$ is number of *cache indices*
  - (block number) mod (# of indices) lets adjacent blocks fit in cache simultaneously!
    - Consecutive blocks go in consecutive cache indices

Block Placement Example

- 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?

Cache:

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

Memory

Addresses	Block Data
00 00 XX	
00 01 XX	
00 10 XX	
00 11 XX	
01 00 XX	
01 01 XX	
01 10 XX	
01 11 XX	
10 00 XX	
10 01 XX	
10 10 XX	
10 11 XX	
11 00 XX	
11 01 XX	
11 10 XX	
11 11 XX	

Cache

Index	Tag	Block Data
00	00	
01		
10	01	
11	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

Analogy: Looking for your friend in a crowded room, so you yell their first name (index)

Several people respond "yeah?", so you get more specific: first name + last name (tag)

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)

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

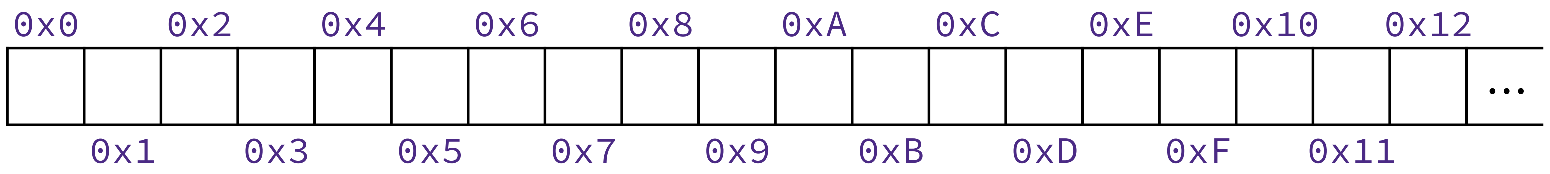
1) **Index** field tells you where to look in cache
2) **Tag** field lets you check that data is the block you want
3) **Offset** field selects specified start byte within block

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

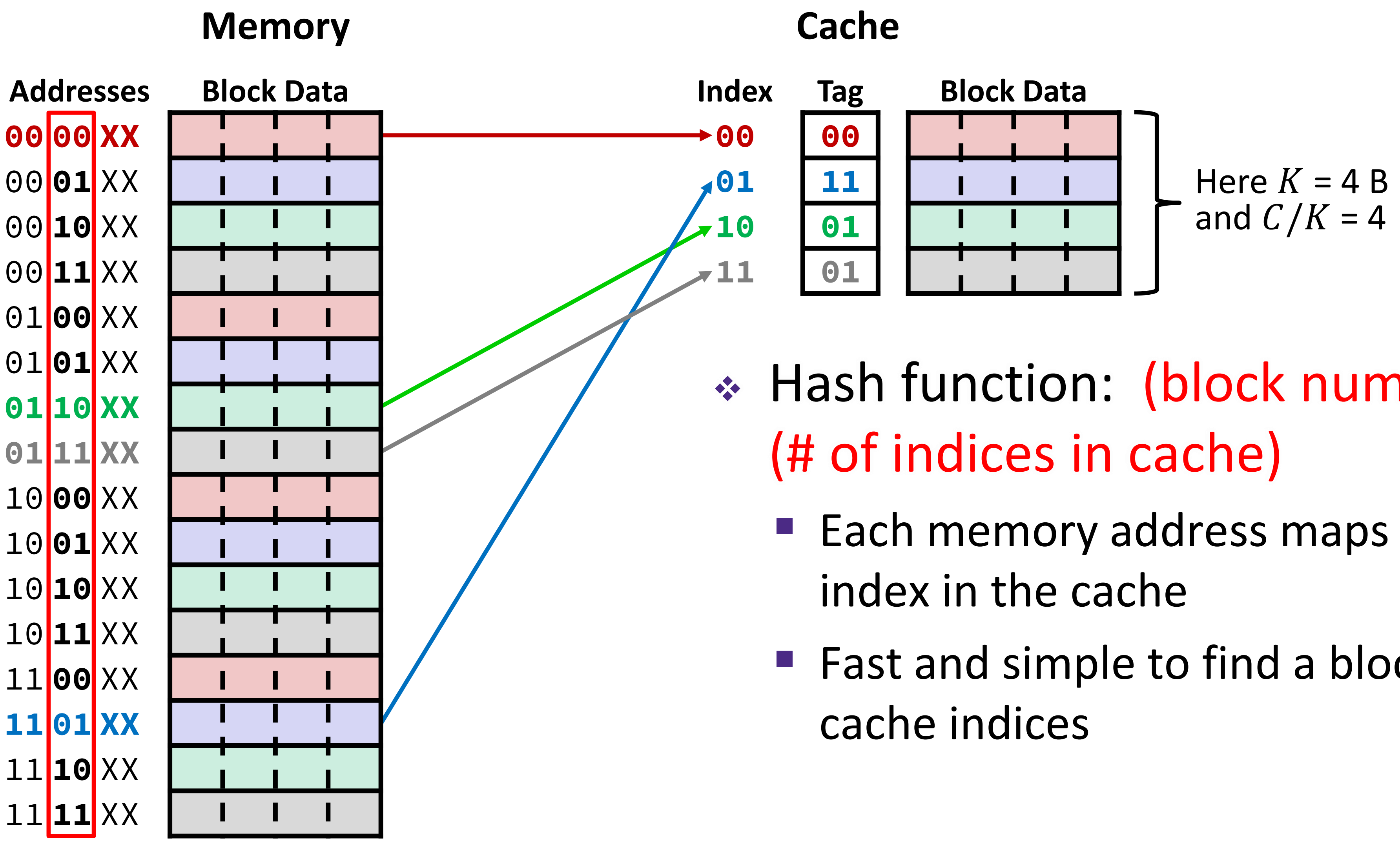
Polling Questions (2/2)

- 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:
    - (`0xE`: miss), (`0xF`: hit), (`0x10`: miss)

A. 4 bytes
B. 8 bytes
C. 16 bytes
D. 32 bytes

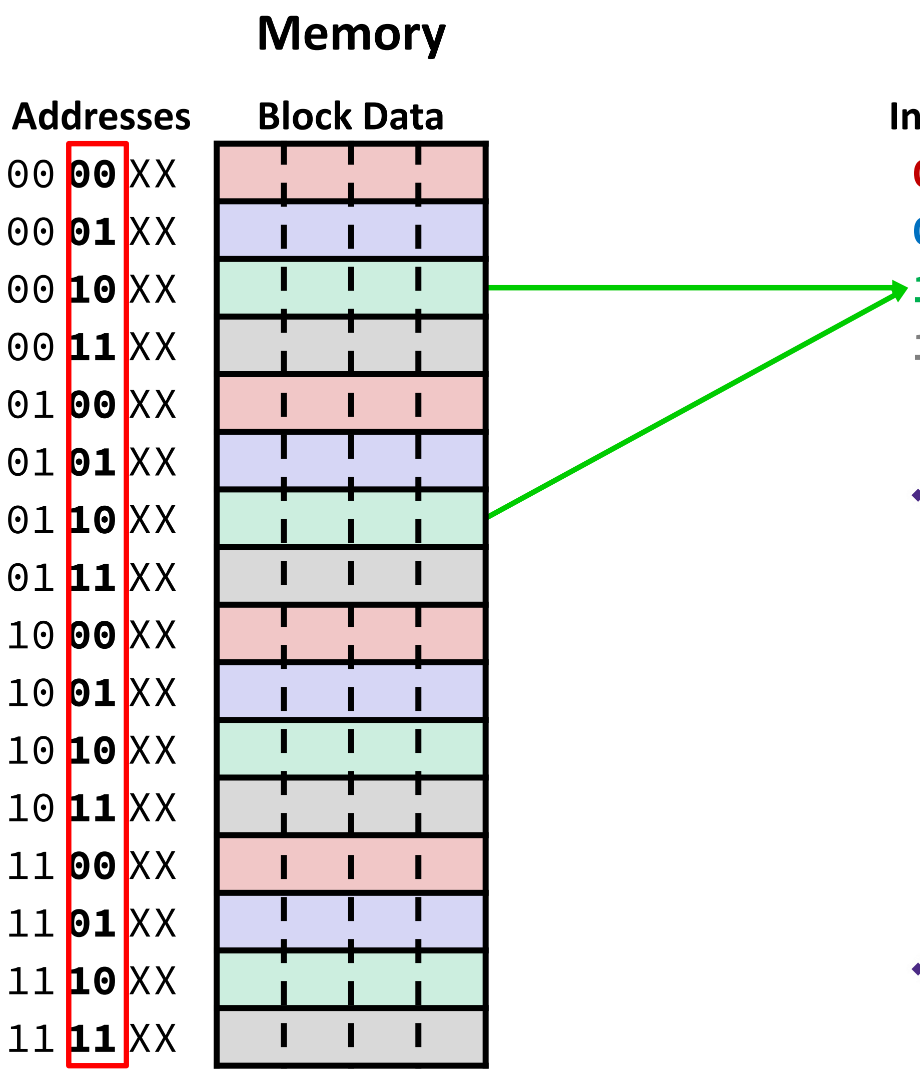


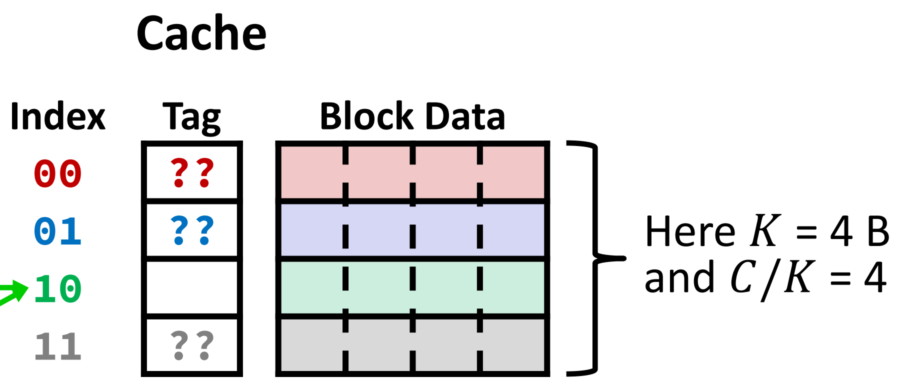
Direct-Mapped Cache Summary



- Hash function: (block number) mod (# of indices in cache)
  - Each memory address maps to *exactly* one index in the cache
  - Fast and simple to find a block, uses all cache indices

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? **Next lecture!**

Homework Setup (1/2)

```
struct WolfPos {
   float x;
   float y;
   float z;
   int id;
};
struct WolfPos grid[16][16];
```

- Assume `&grid = 0`

- What are the addresses of the following pieces of data?
  - `&(grid[0][0].id) =`
  - `&(grid[1][0].y) =`
  - `&(grid[3][4].x) =`

Homework Setup (2/2)

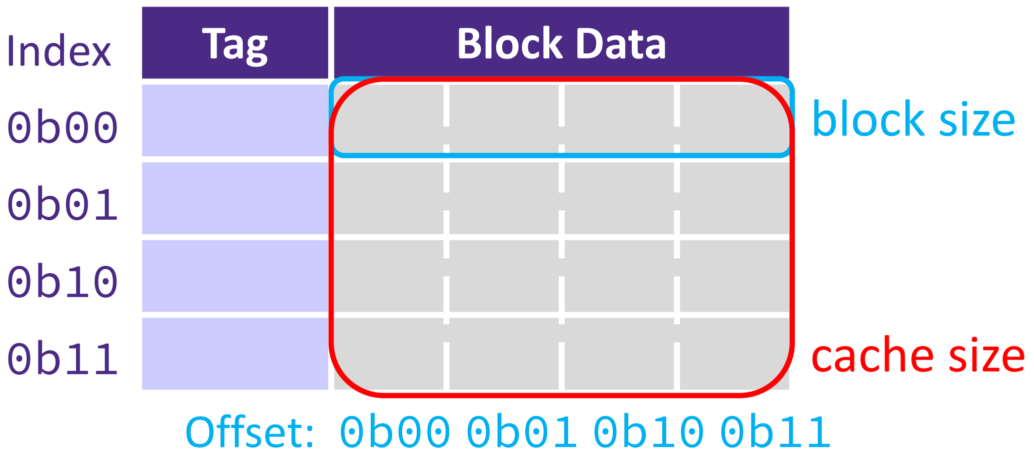
```
struct WolfPos {
   float x;
   float y;
   float z;
   int id;
};
struct WolfPos grid[16][16];
```

- Assume `&grid = 0`

- Cold direct-mapped cache with $C = 1024\text{ B}$ and $K = 16\text{ B}$
  - What happens if we access `grid[0][0].x` and then `grid[4][0].x`?

Summary (1/2)

- Cache parameters define the cache geometry:
  - **Block size** is number of bytes per block
  - **Cache size** is number of bytes (or blocks) of data the cache can hold
- Finding a byte in the cache:
  - **Offset** refer to which byte in block
  - **Index** refers to which block in cache
- Example:
  - K = 4 B, C = 16 B = 4 blocks



Summary (2/2)

- **Direct-mapped cache:** each block in cache is assigned a unique index
  - Uses hash function of (block number) mod (# of cache indices)
    - Deterministic placement of each block, with many blocks mapping into the same index
    - Tag bits stored in cache and used to distinguish between blocks that map to same index

- Accessing the cache:
  (TIO address breakdown)

$\boldsymbol{m}$-bit address:

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

Block Number (Tag + Index)

1) **Index** field tells you where to look in cache (width $s = \log_2 S$)
2) **Tag** field lets you check that data is the block you want (width $t = m - s - k$)
3) **Offset** field selects specified start byte within block (width $k = \log_2 K$)