Memory & Caches II

CSE 351 Spring 2024

Instructor:

Elba Garza

Teaching Assistants:

Ellis Haker
Adithi Raghavan
Aman Mohammed
Brenden Page
Celestine Buendia
Chloe Fong
Claire Wang
Hamsa Shankar
Maggie Jiang
Malak Zaki
Naama Amiel
Nikolas McNamee
Shananda Dokka
Stephen Ying
Will Robertson

Playlist: CSE 351 24Sp Lecture Tunes!

Relevant Course Information

- HW13/14 due tonight
  - HW15 due Friday (03 May)
  - HW16 due Monday (06 May)
- Take-home Midterm, May 6th to May 7th
- Lab 3 due May 8th after the midterm!
- Mid-Quarter Canvas Survey out; closes on Monday, May 6th
- HW17/18 out today & due following Friday (10 May)
  - Don't wait too long, this is a **big** homework & includes topics in this lecture!

Mid-Quarter Evaluation w/ ET&L

- Things going well:
  - Pre-lecture readings, labs & video tutorials — very useful!
  - Section & office hours
  - TAs (Thanks so much for all you do, TAs!)
- Things to improve on:
  - Lecture Polls — usefulness & placement
  - Course pace & workload — Homeworks out at start of week? Weekly homework?
  - Absences

Current Events & CSE 351

- There may be interruptions to course resources:
  - Office Hours
  - Section
  - Grading

- Please bear with us as information comes in and the situation develops…

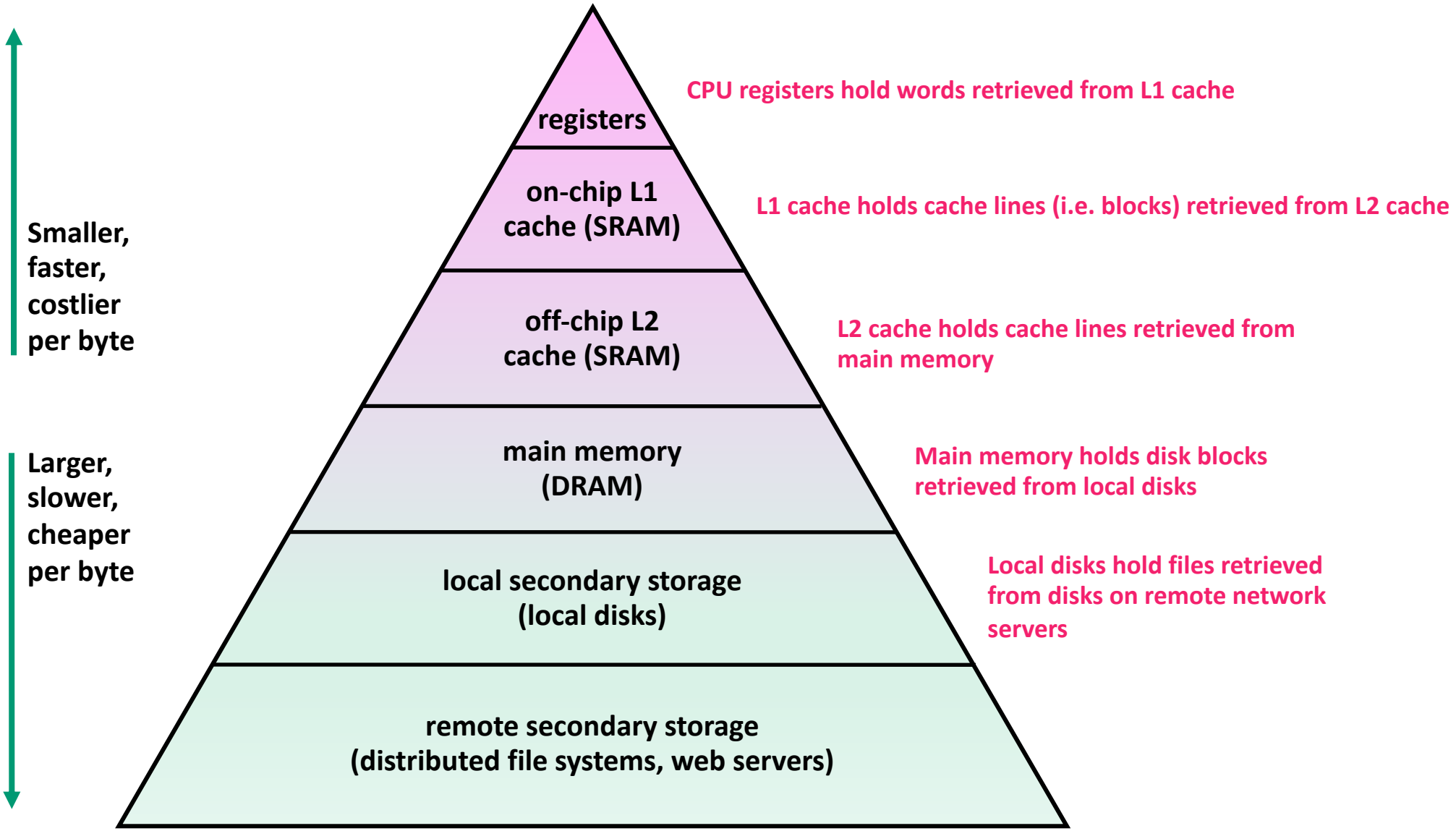
Memory Hierarchies (Review)

- Some fundamental and enduring properties of hardware and software systems:
  - Faster storage technologies generally cost more per byte & have lower capacity
  - The gaps between memory technology speeds are widening:
    registers ↔ cache, cache ↔ DRAM, DRAM ↔ disk, etc.
  - Well-written programs tend to exhibit good locality

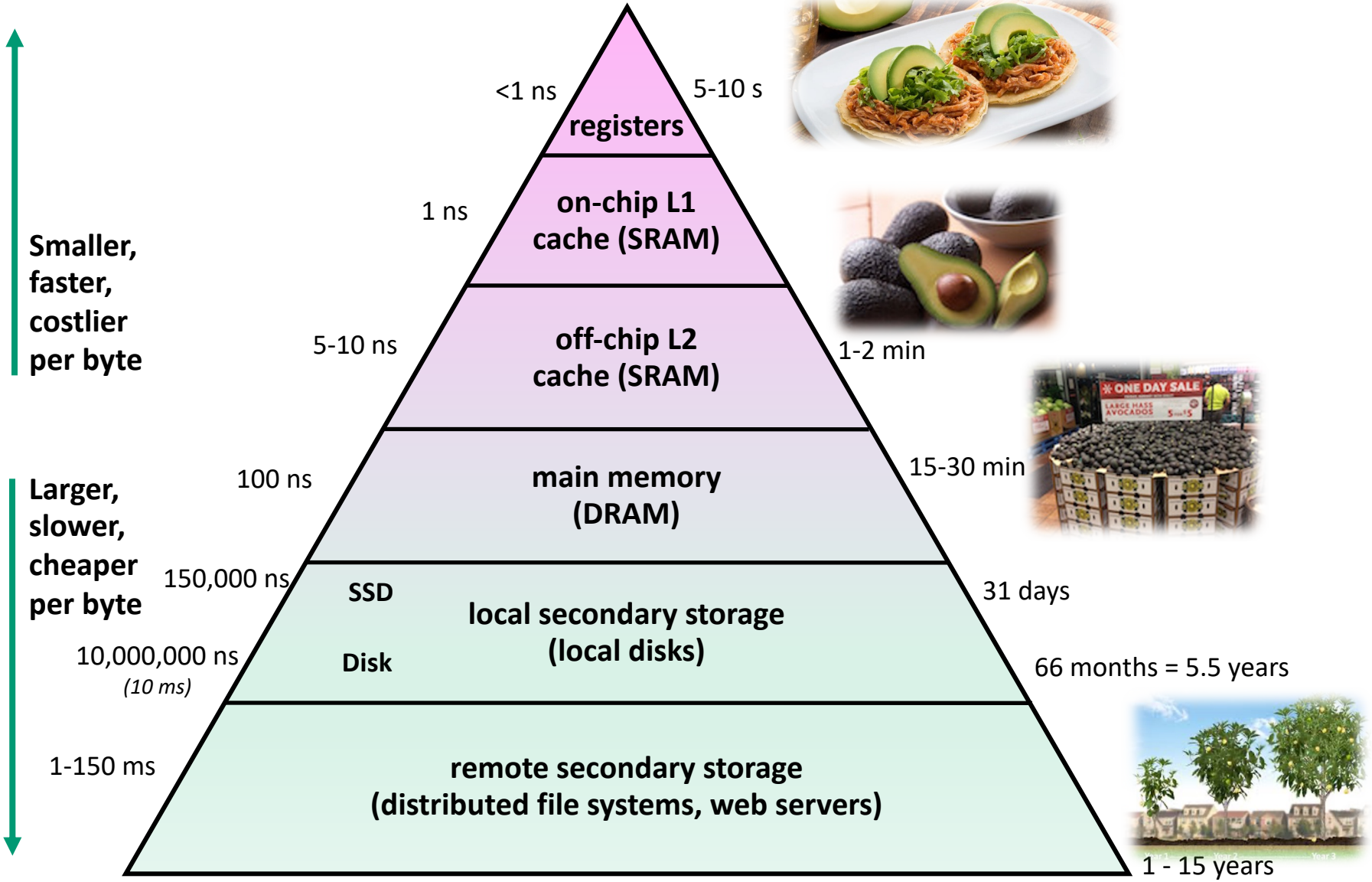
- These properties complement each other & suggest approach for organizing memory/storage systems known as a

  Memory Hierarchy: For each level l, the faster, smaller device at level l serves as a cache for the larger, slower device at level $l+1$

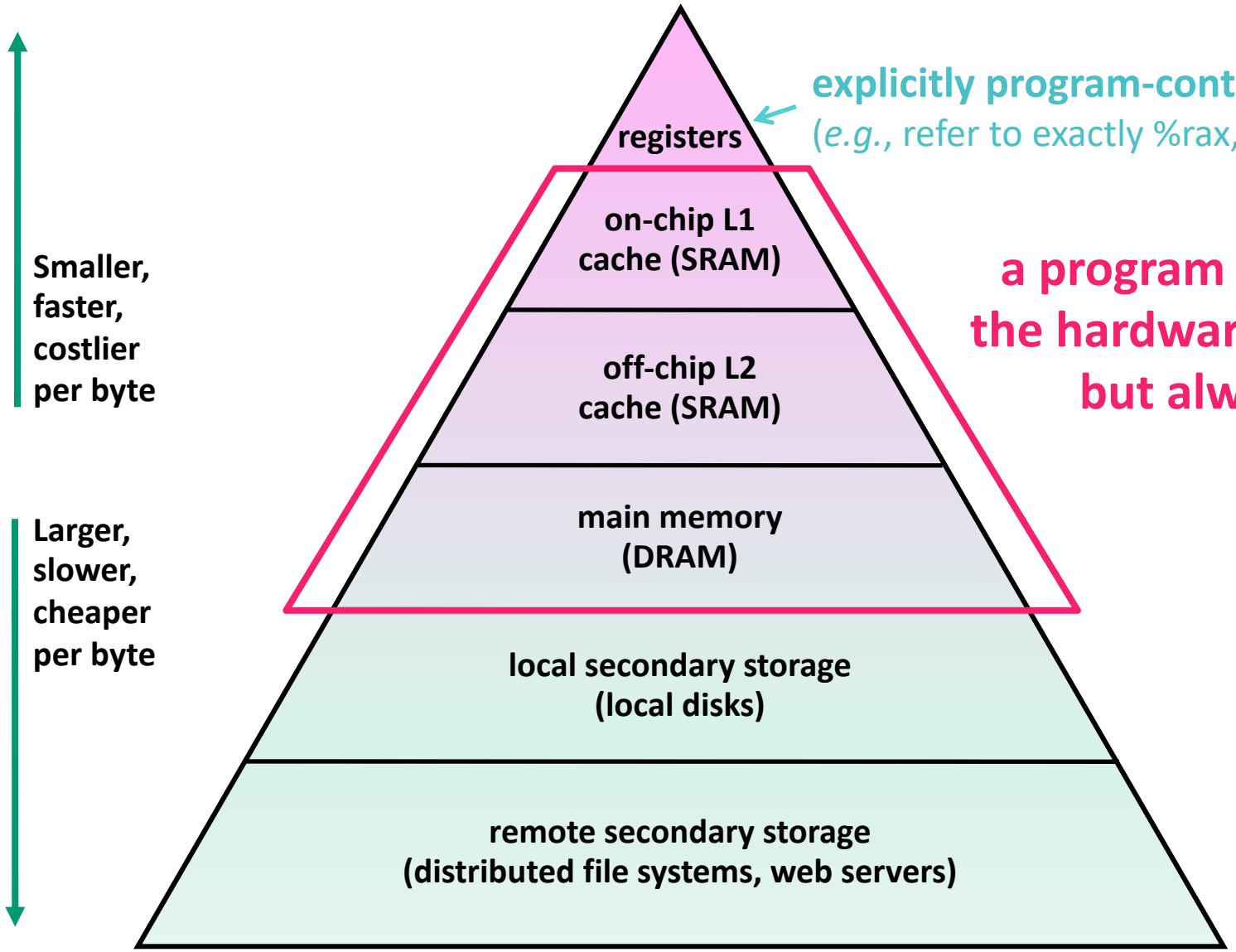
An Example Memory Hierarchy



An Example Memory Hierarchy



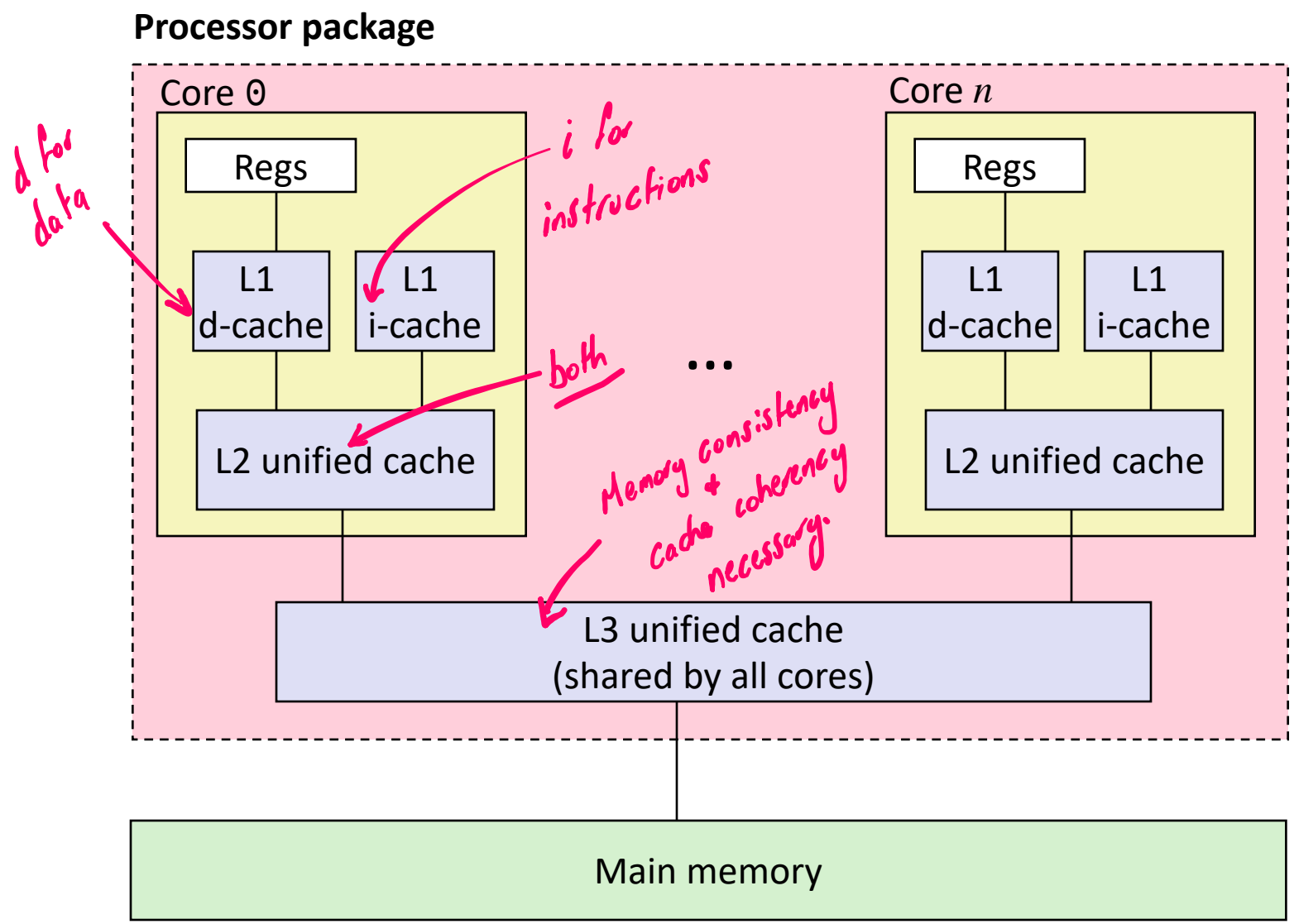
Why hadn't we talked about this before?!



explicitly program-controlled
(*e.g.*, refer to exactly %rax, %rbx)

a program only sees "memory"; the hardware manages caching—but always good to be cache aware!

Example Microarchitecture



Block size:
$K =$ [64 bytes] for all caches

L1 i-cache and d-cache:
$C =$ [32 KiB,] 8-way,
Access: 4 cycles

x 2 time

L2 unified cache:
$C =$ [256 KiB,] 8-way,
Access: 11 cycles

x 3 time

L3 unified cache:
$C =$ [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 for total size in B(ytes), or S for number of blocks)
  - Addresses:
    - block offset field (k bits wide)
    - block address:
      - index field (s bits wide)
      - tag field (t bits wide)
  - Cache organization: direct-mapped cache

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 $C = 4\ KiB = 2^{12} B$
- How many blocks can the cache hold? $S = C/K = 2^{12}/2^3 = 2^9$ blocks
- How many bits wide is the block offset field? $k = \log_2(K) = 3$ bits
- Which of the following addresses (possibly multiple) would fall under block number 3?

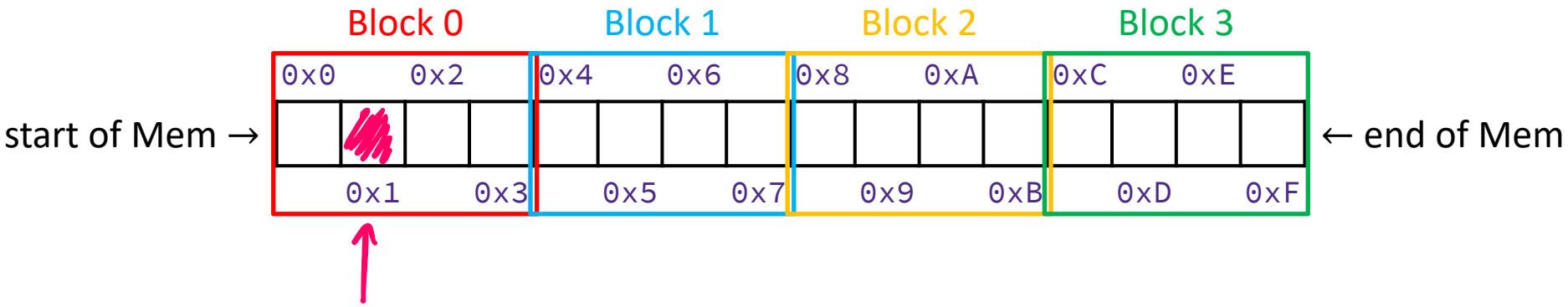
A. 0x3 **B. 0x1F** **C. 0x30** **D. 0x38**

0b0...00011 0b0..011|111 (k) 0b0..0110|000 (k) 0b0..0111|000 (k)

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!
  - Small example ($K = 4$ B):



Cache Organization (1)

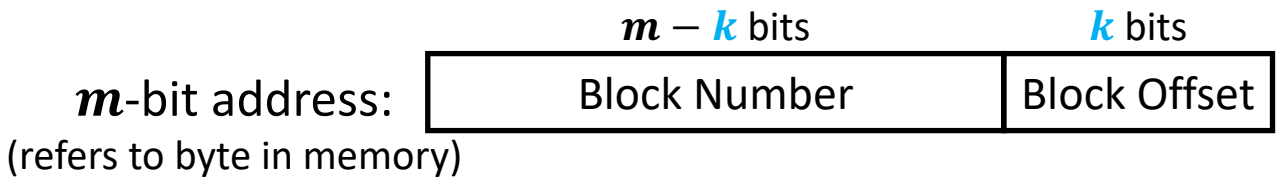
Note: The textbook uses "b" for offset bits

- 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!

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)



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?

$K = 4\,B$

$\Rightarrow k = \log_2(K) = 2$

$0x15 = 0b\,0\,1\,0\,1\,0\,1$

Block number | k offset

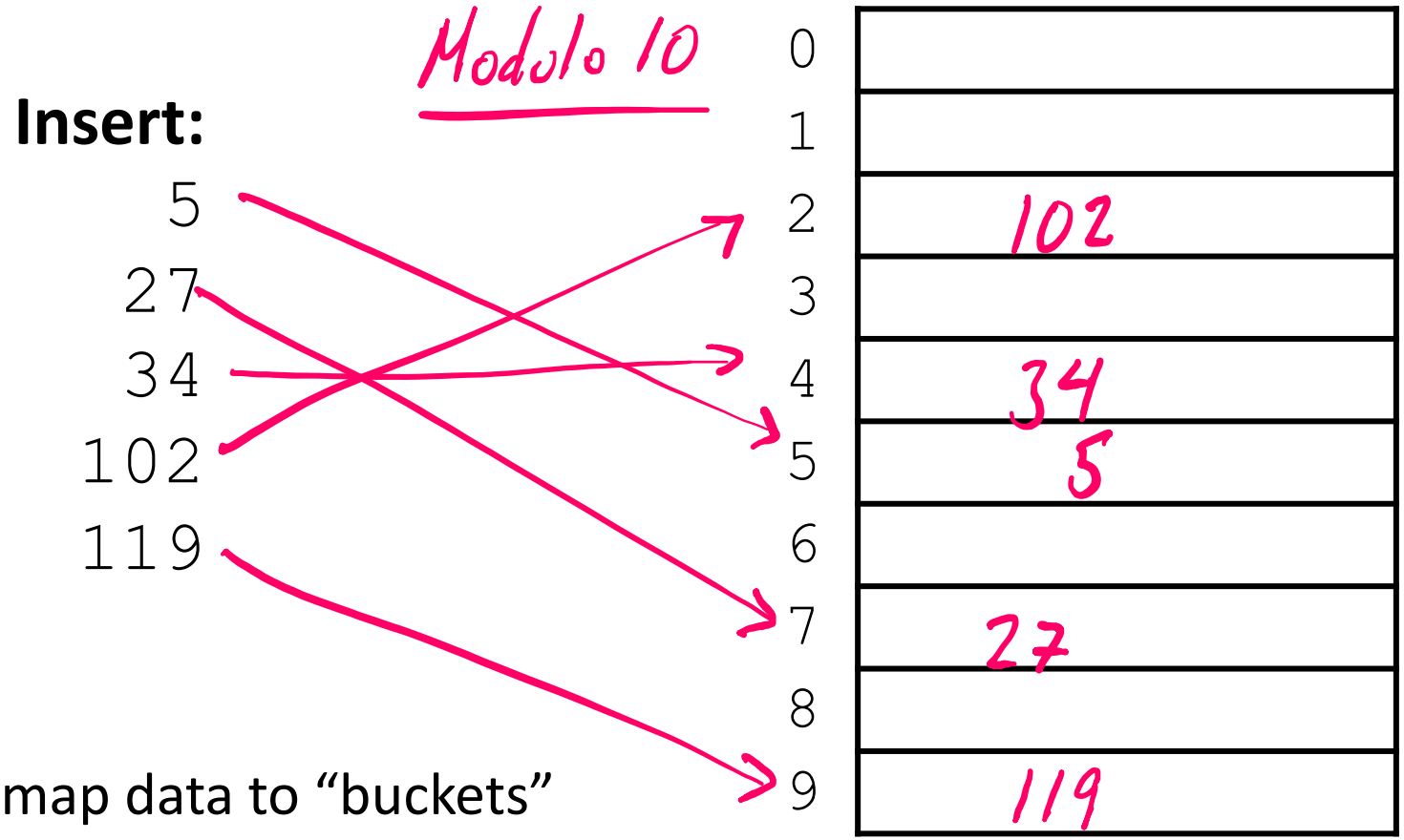
Cache Organization (2)

- 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 ($S = C/K$)

  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 you've learned that provides **fast lookup**?
  - Hash table!

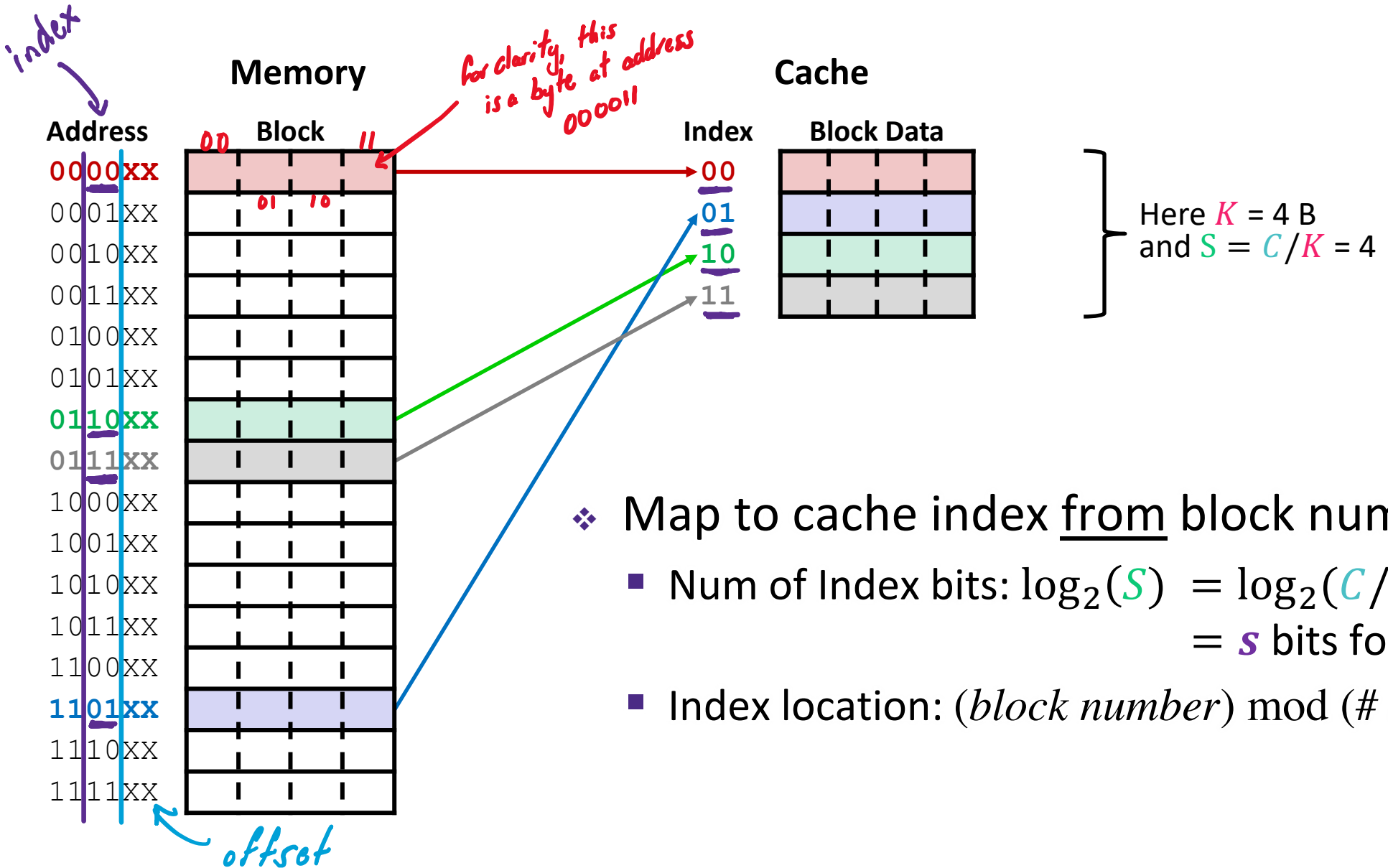
Hash Tables for Fast Lookup



Apply hash function to map data to “buckets”

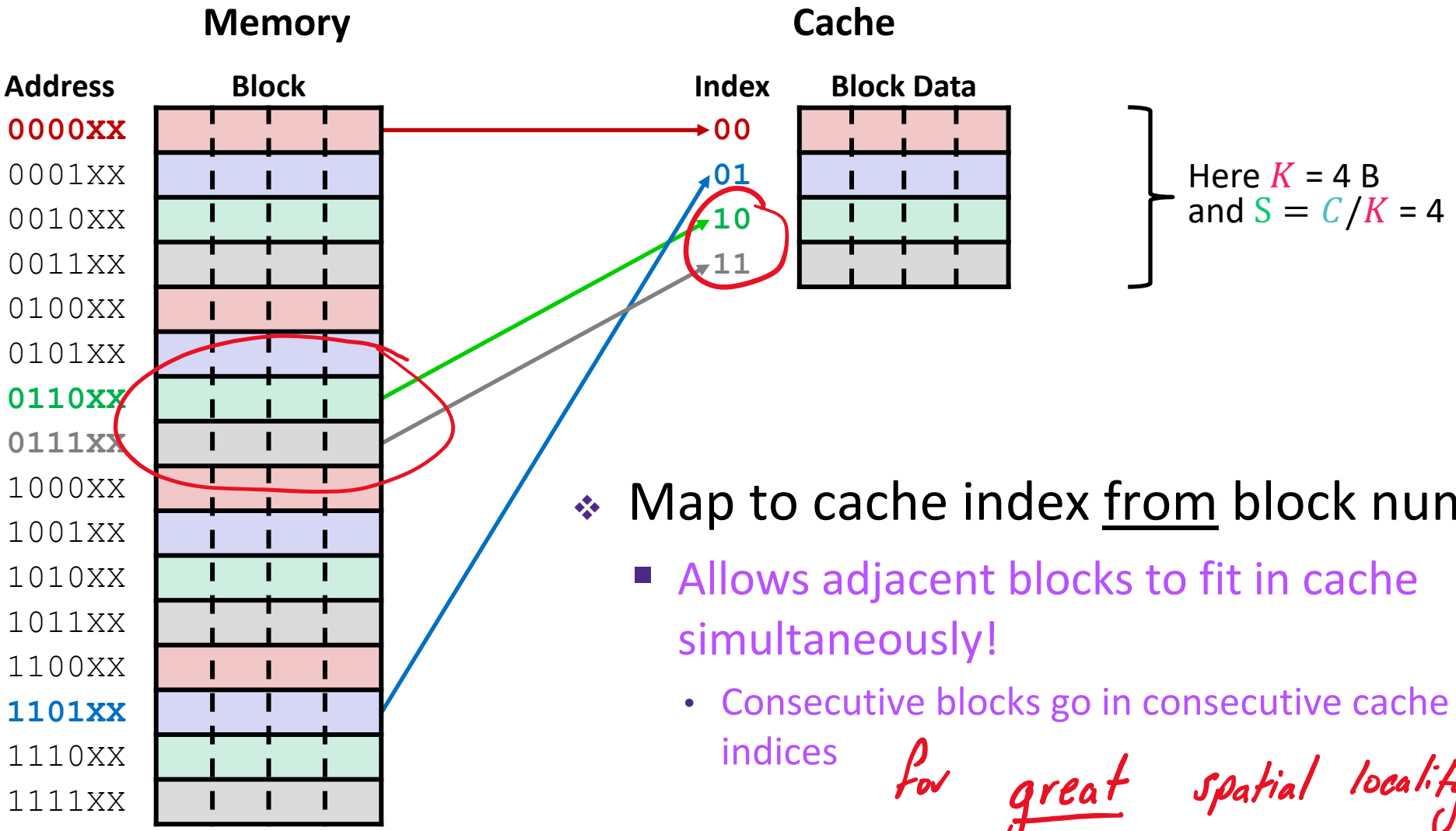
e.g. hash function = $N \bmod 10$

Place Data in Cache by Hashing Address



- Map to cache index from block number (using s bits from it)
  - Num of Index bits: $\log_2(S) = \log_2(C/K)$
    $= s$ bits for index
  - Index location: $(block\ number)$ mod $(\#\ blocks\ in\ cache)$

Place Data in Cache by Hashing Address



- Map to cache index from block number
  - Allows adjacent blocks to fit in cache simultaneously!
    - Consecutive blocks go in consecutive cache indices

for great spatial locality

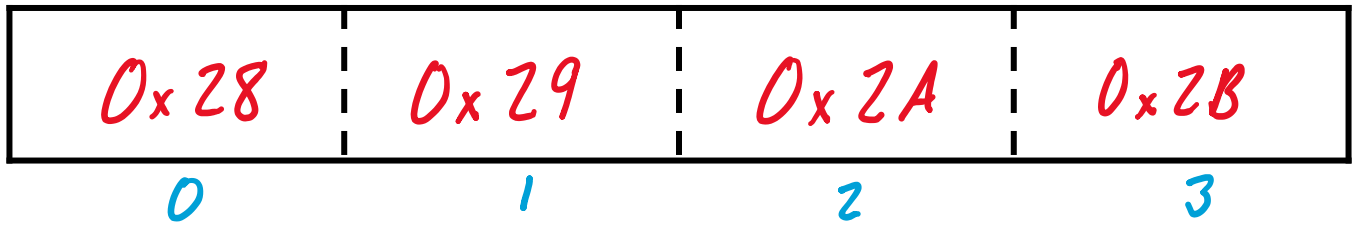
Polling Question

$\log_2(K) = 2$

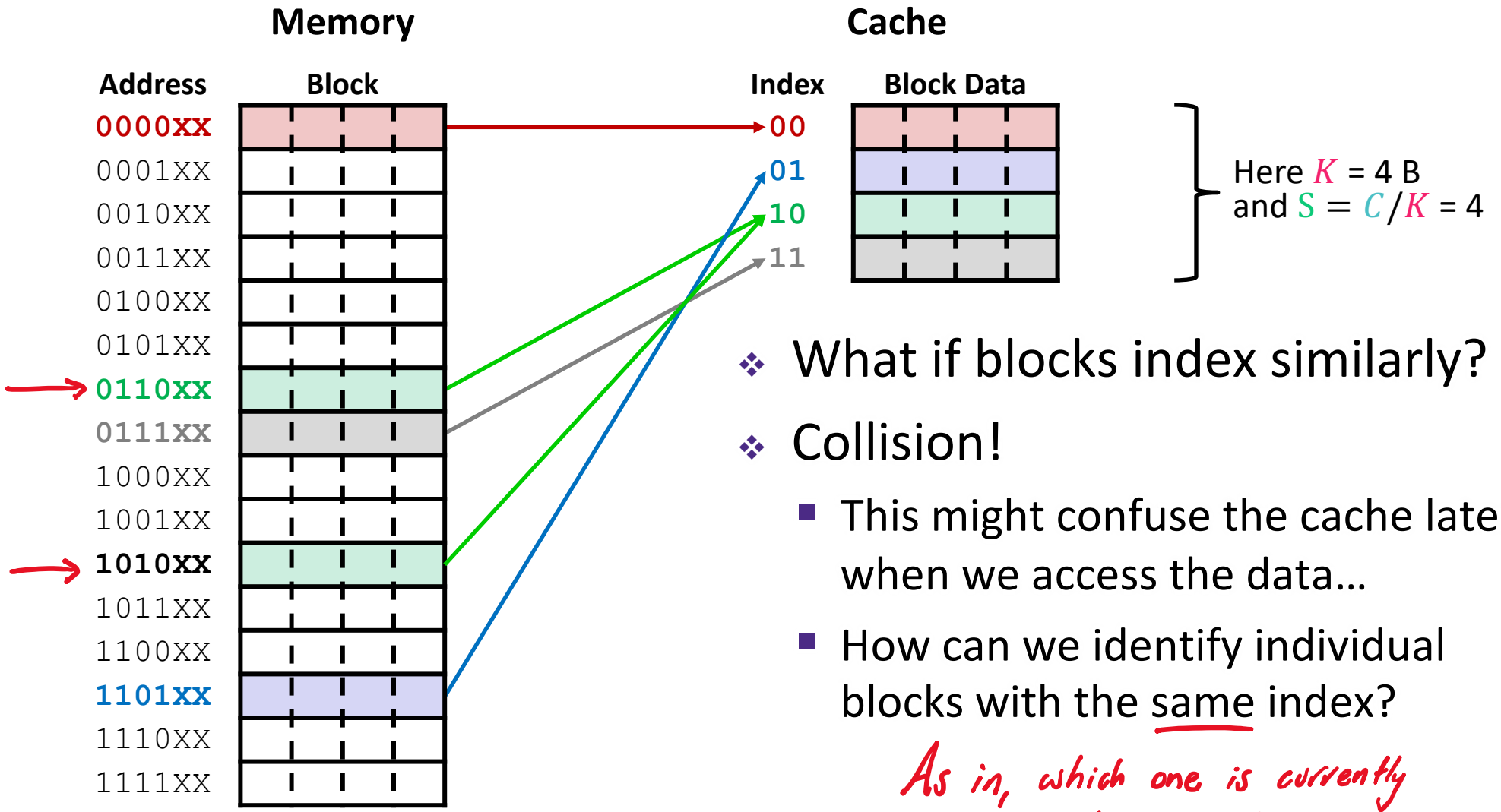
- 6-bit addresses, block size K = 4 B, and our cache holds S = 4 blocks.
- A request for address **0x2A** results in a cache miss. Which index does this block get loaded into and which 3 other addresses are loaded along with it?

0x2A = 0b101010 (10 = k) ← 2-bit offset bits have a value of 2₁₀. Therefore, 0x2A's data will be at index 2.

Block:



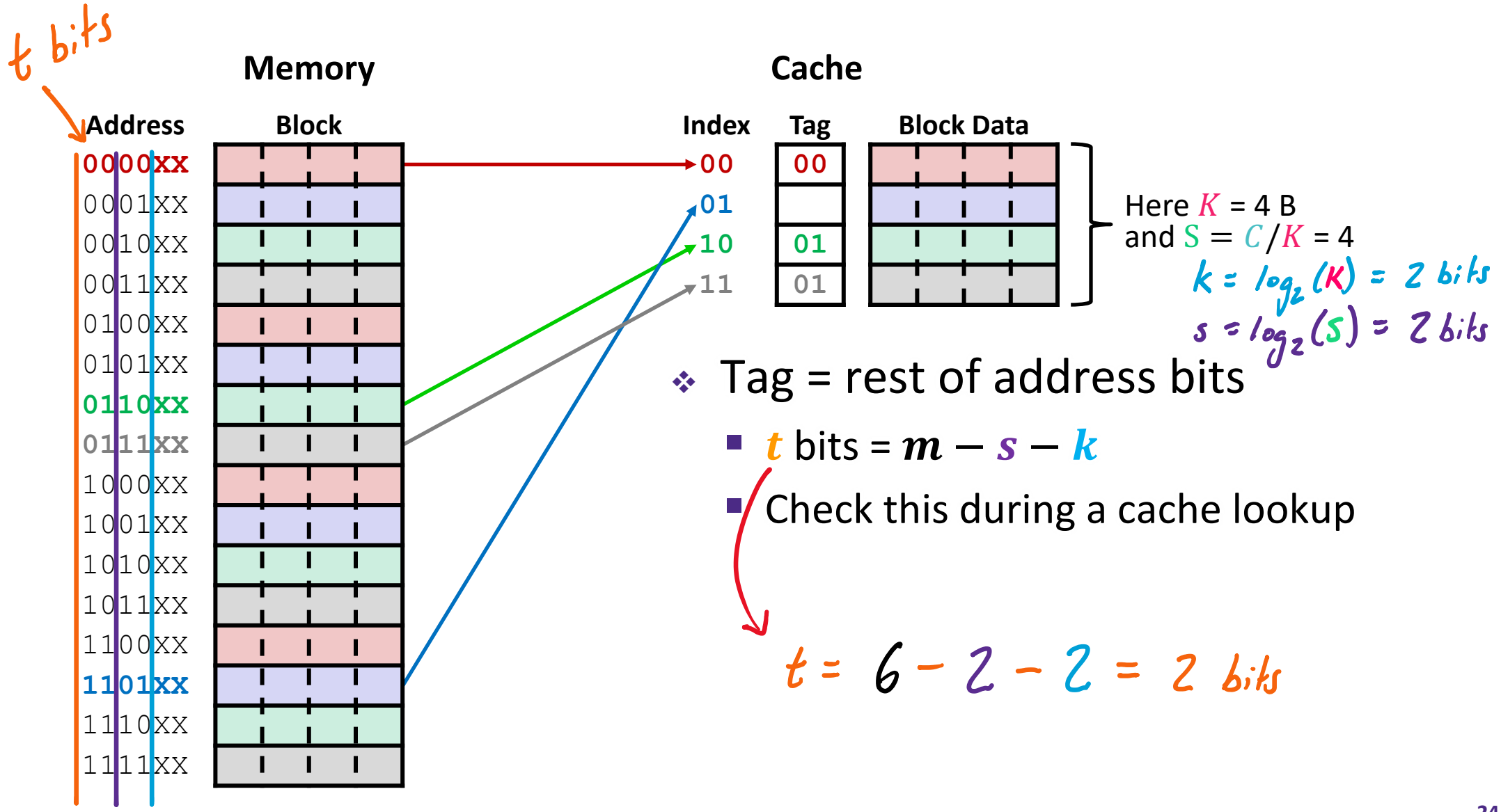
Place Data in Cache by Hashing Address



- What if blocks index similarly?
- Collision!
  - This might confuse the cache later when we access the data…
  - How can we identify individual blocks with the same index?

As in, which one is currently in the cache?!

Tags Differentiate Blocks in Same Index



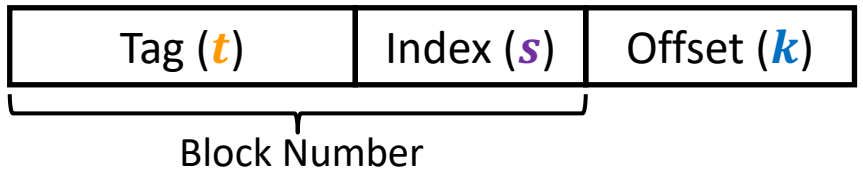
- Tag = rest of address bits
  - t bits = $m - s - k$
  - Check this during a cache lookup

$t = 6 - 2 - 2 = 2$ bits

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:



  - **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

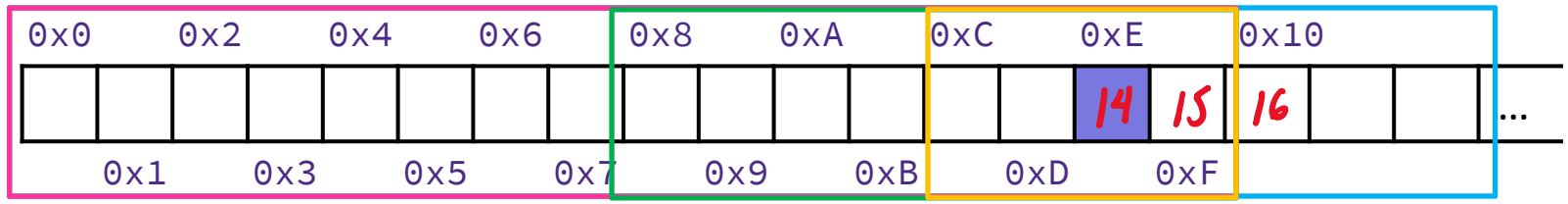
  - **Note:** t and s sizes will change based on hash function

Cache Puzzle Example

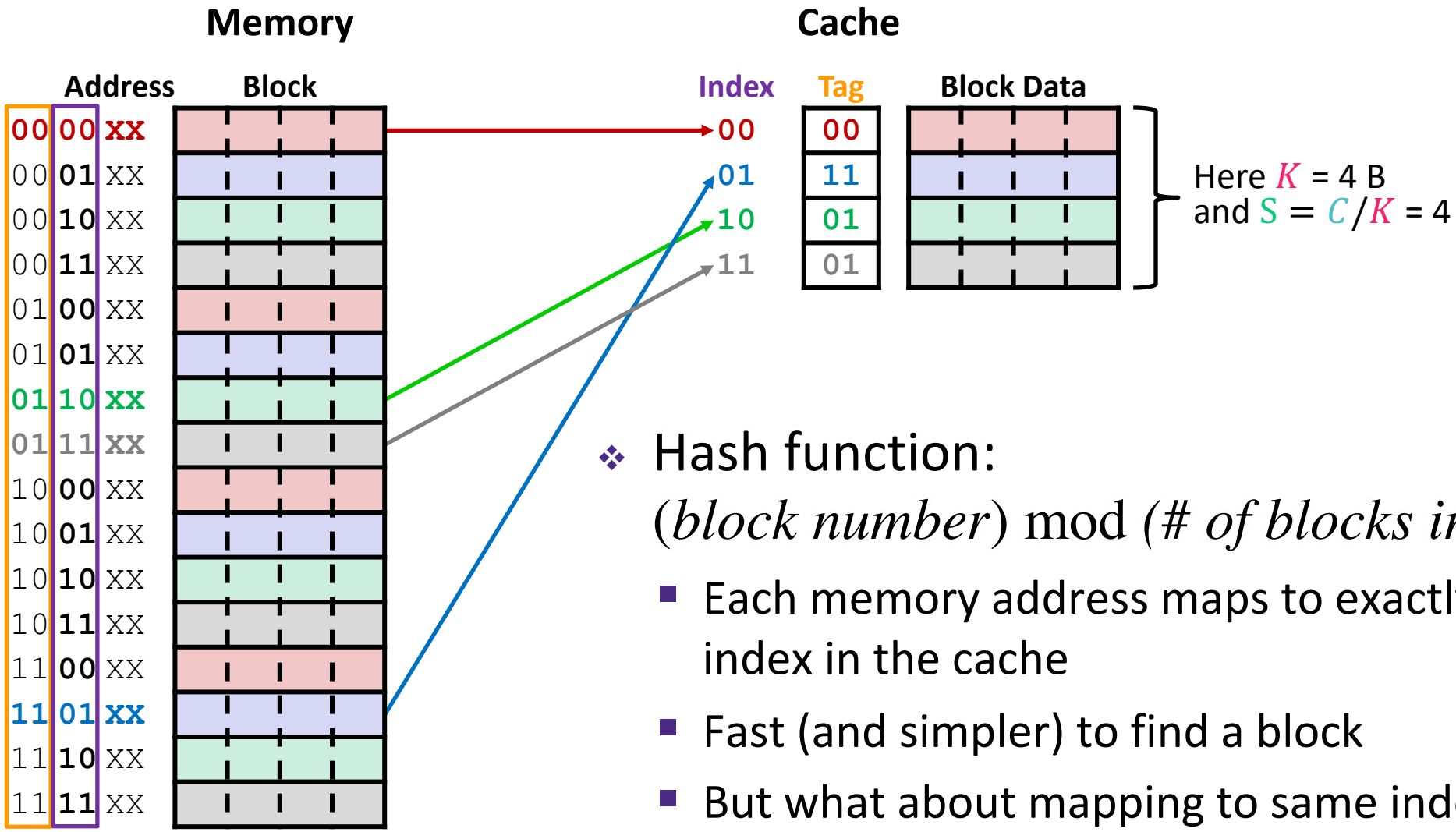
- Based on the following behavior, which of the following block sizes is NOT possible for our cache?
  - Cache starts *empty*, also known as a **cold cache**
  - Access (addr: hit/miss) stream:
    - (14: **miss**), (15: hit!), (16: **miss**)

0b000…01110

A. 4 bytes
B. 8 bytes
C. 16 bytes
D. 32 bytes
E. We're lost…

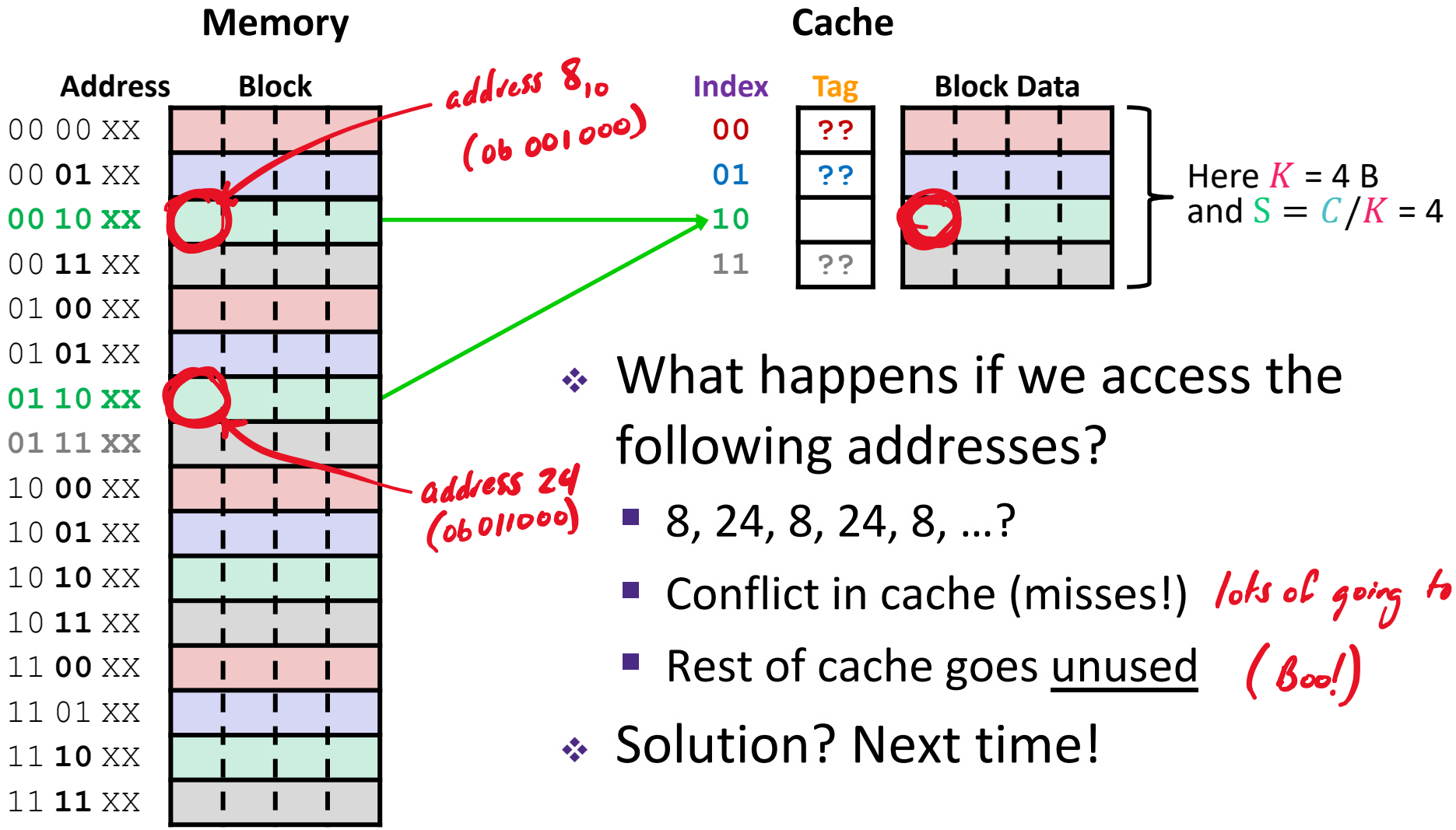


Summary: Direct-Mapped Cache



- 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
  - But what about mapping to same index?...

Direct-Mapped Cache: A Problem!



- What happens if we access the following addresses?
  - 8, 24, 8, 24, 8, …?
  - Conflict in cache (misses!) lots of going to lower level in memory hierarchy. (Boo!) → bad performance!
  - Rest of cache goes unused
- Solution? Next time!