Caches II

CSE 351 Summer 2021

Instructor:
Mara Kirdani-Ryan

Teaching Assistants:
Kashish Aggarwal
Nick Durand
Colton Jobs
Tim Mandzyuk

Gentle, Loving Reminders

- hw14 due tonight! hw15 due monday!
  - No homework due Friday!
- Lab 3 due Friday (7/30)
- Unit Summary 2 Due next Monday (8/2)
  - Critique today!
  - Task #3 is out -- you'll look at some assembly

Feedback on Unit Summaries? Come talk!

Learning Objectives

Understanding this lecture means that you can:

- Explain the benefits of a memory hierarchy to someone that hasn’t taken this course
- Define cache terminology:
  - Block size vs Cache size; block number vs offset
  - Sets, associativity, tags
- Differentiate between direct-mapped, associative, and fully-associative caches
- Give & receive unit summary feedback!

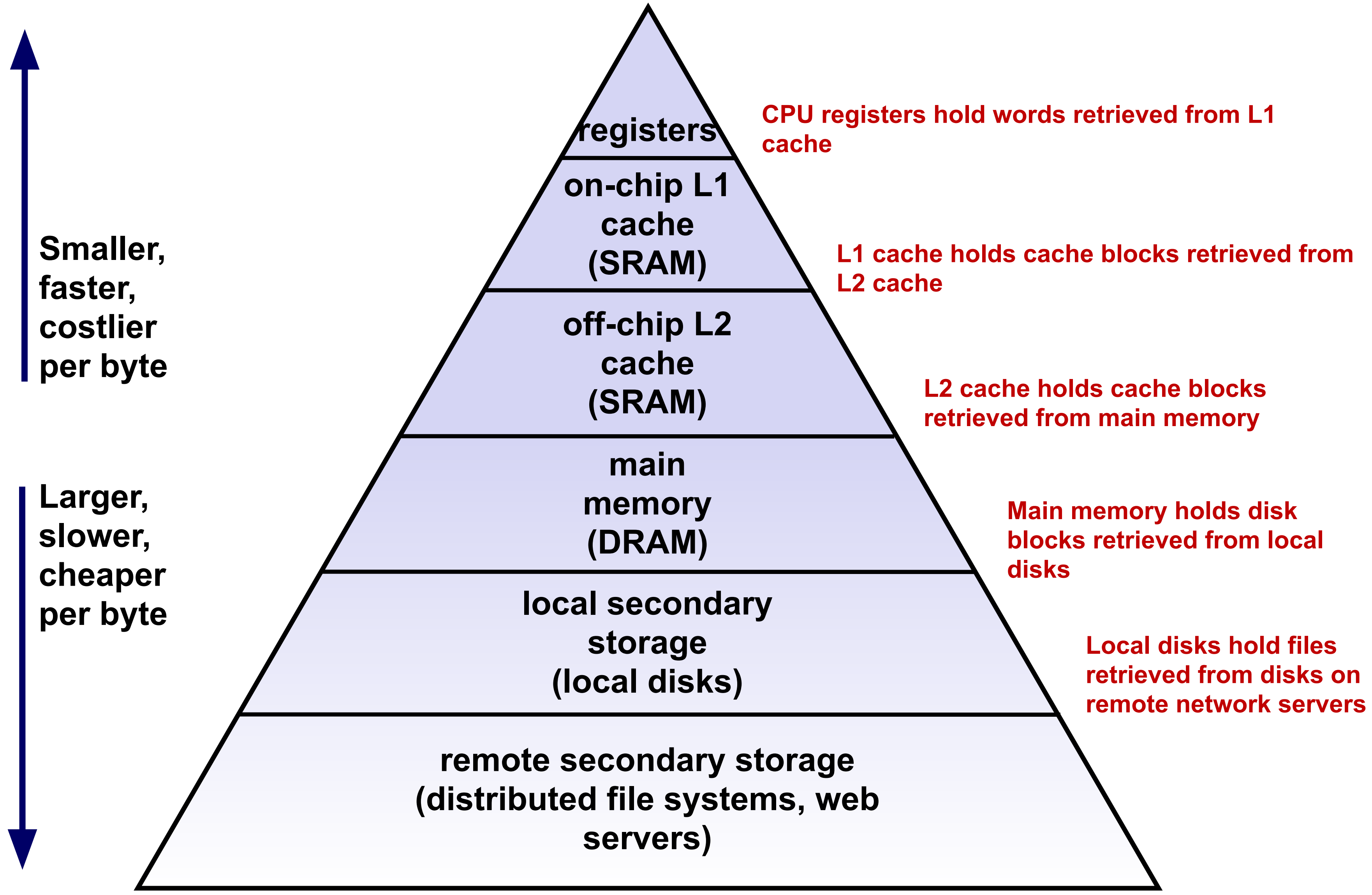
Memory Hierarchies

- 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.
  - “Average” programs tend to exhibit good locality

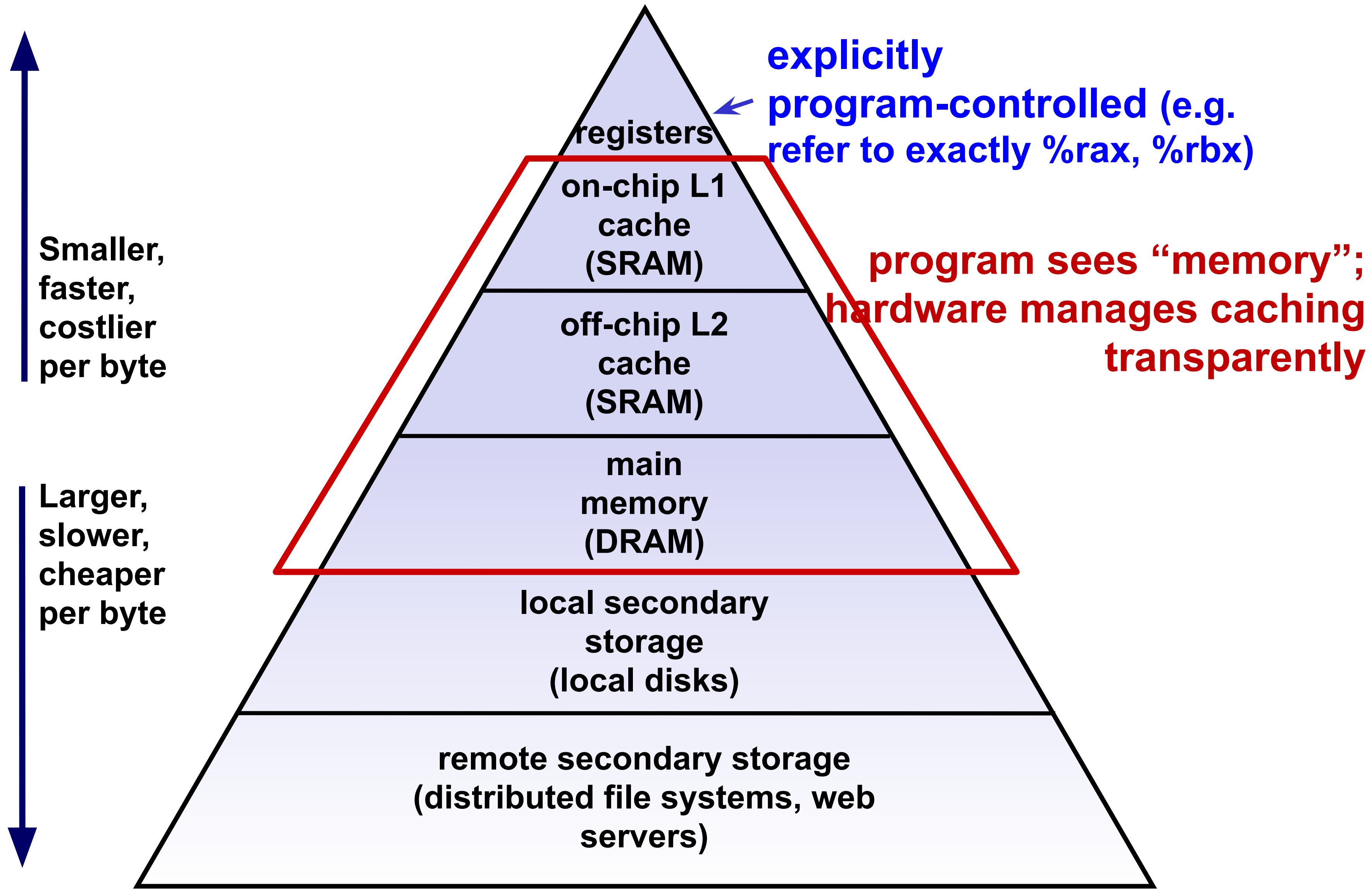
Memory Hierarchies

- If you're trying to make things faster, you might end up with a <u>memory hierarchy</u>
  - For each level k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1

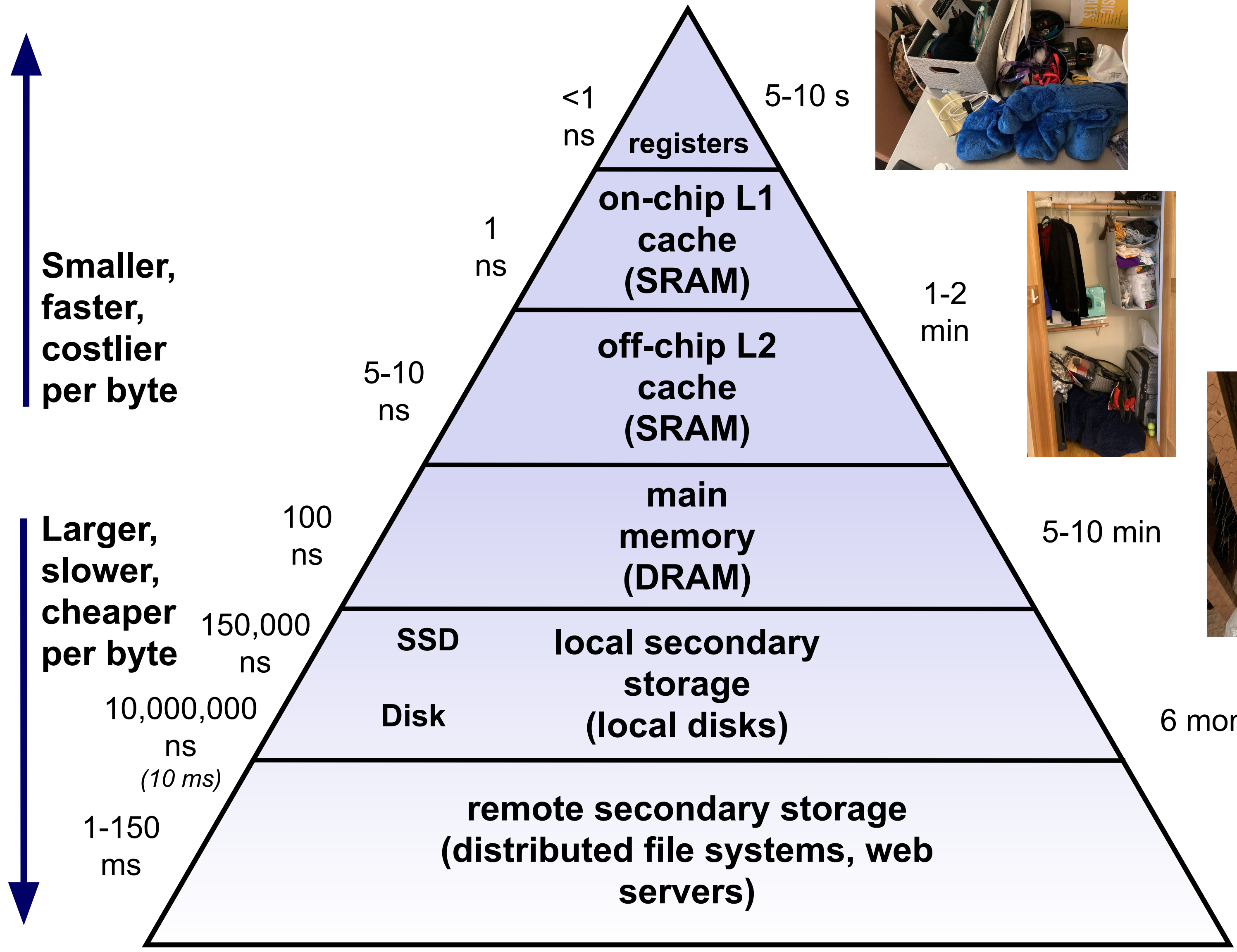
An Example Memory Hierarchy

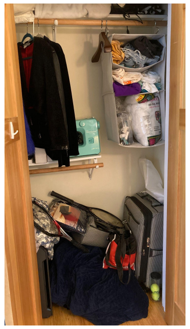

Smaller, faster, costlier per byte
Larger, slower, cheaper per byte
registers
on-chip L1 cache (SRAM)
off-chip L2 cache (SRAM)
main memory (DRAM)
local secondary storage (local disks)
remote secondary storage (distributed file systems, web servers)
CPU registers hold words retrieved from L1 cache
L1 cache holds cache blocks retrieved from L2 cache
L2 cache holds cache blocks 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

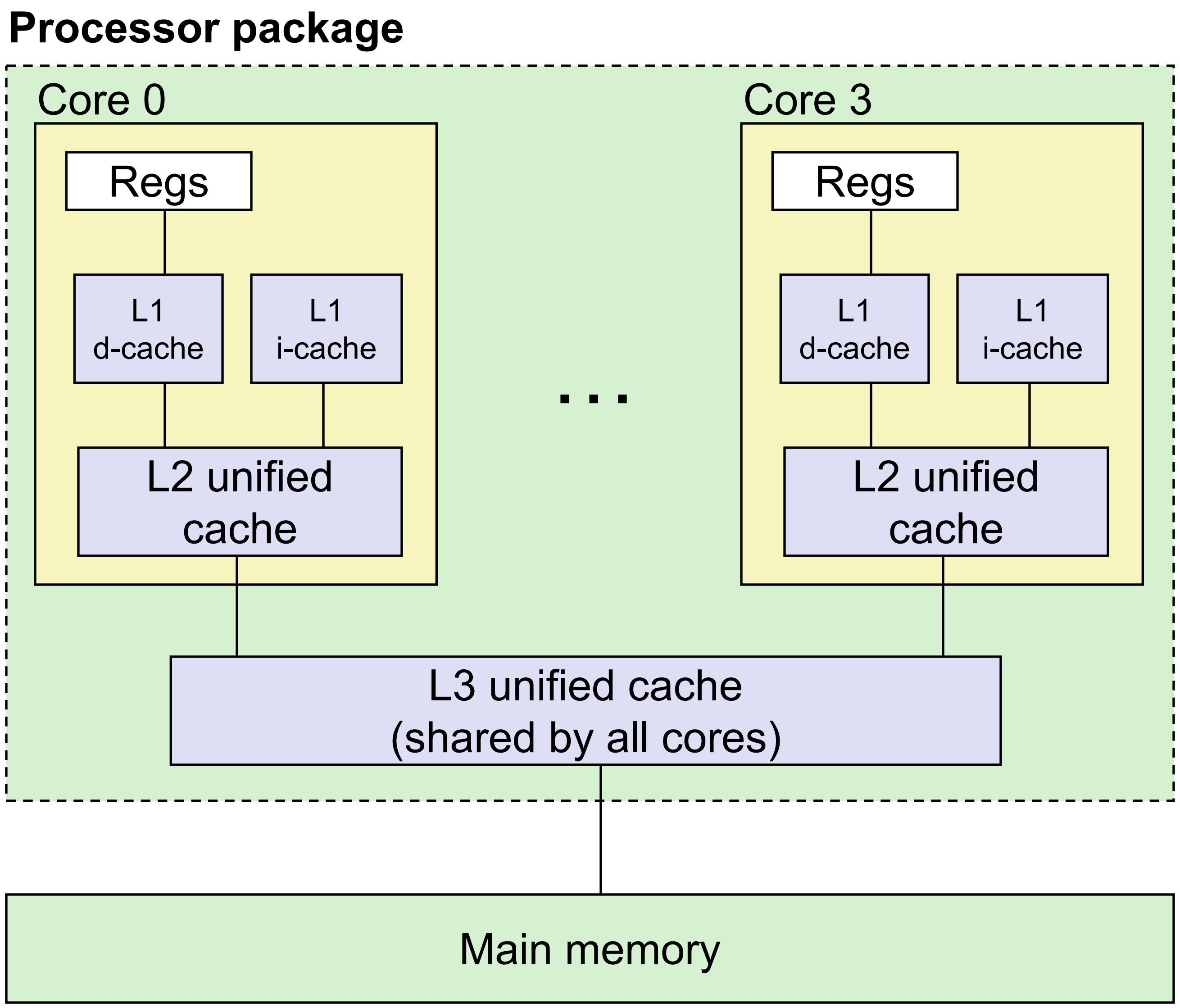

Smaller,
faster,
costlier
per byte
Larger,
slower,
cheaper
per byte
registers
on-chip L1 cache (SRAM)
off-chip L2 cache (SRAM)
main memory (DRAM)
local secondary storage (local disks)
remote secondary storage (distributed file systems, web servers)
explicitly program-controlled (e.g. refer to exactly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

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

That’s the memory hierarchy!
Feeling ok?

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)

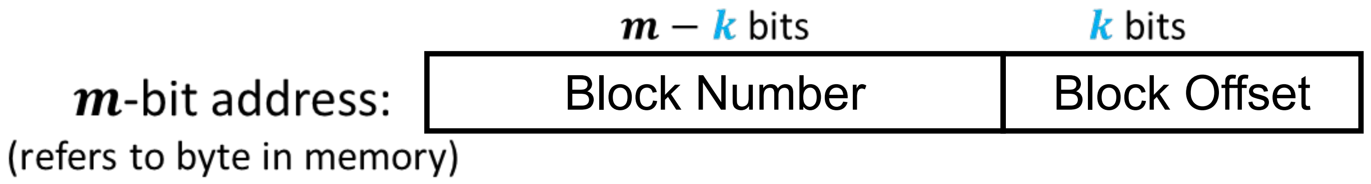
Note: The textbook uses “B” for block size

- 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)


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

Checking in!

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

	Block Num	Block Offset
🐶	1	1
🐈	1	5
🐑	5	1
🦄	5	5
🥶	Help!	

Checking in!

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

0x15 = 0001 0101

	Block Num	Block Offset
🐶	1	1
🐈	1	5
🐑	5	1
🦄	5	5
🥶 Help!		

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

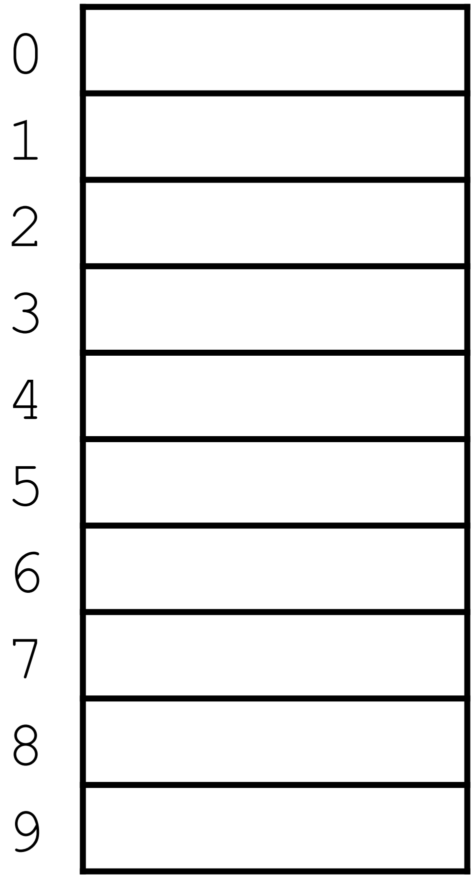
- Where should data go in the cache?
  - We need a mapping from memory addresses to specific locations in the cache to make checking the cache for an address **fast**

- What is a data structure that provides fast lookup?
  - Hash table!

Review: Hash Tables for Fast Lookup

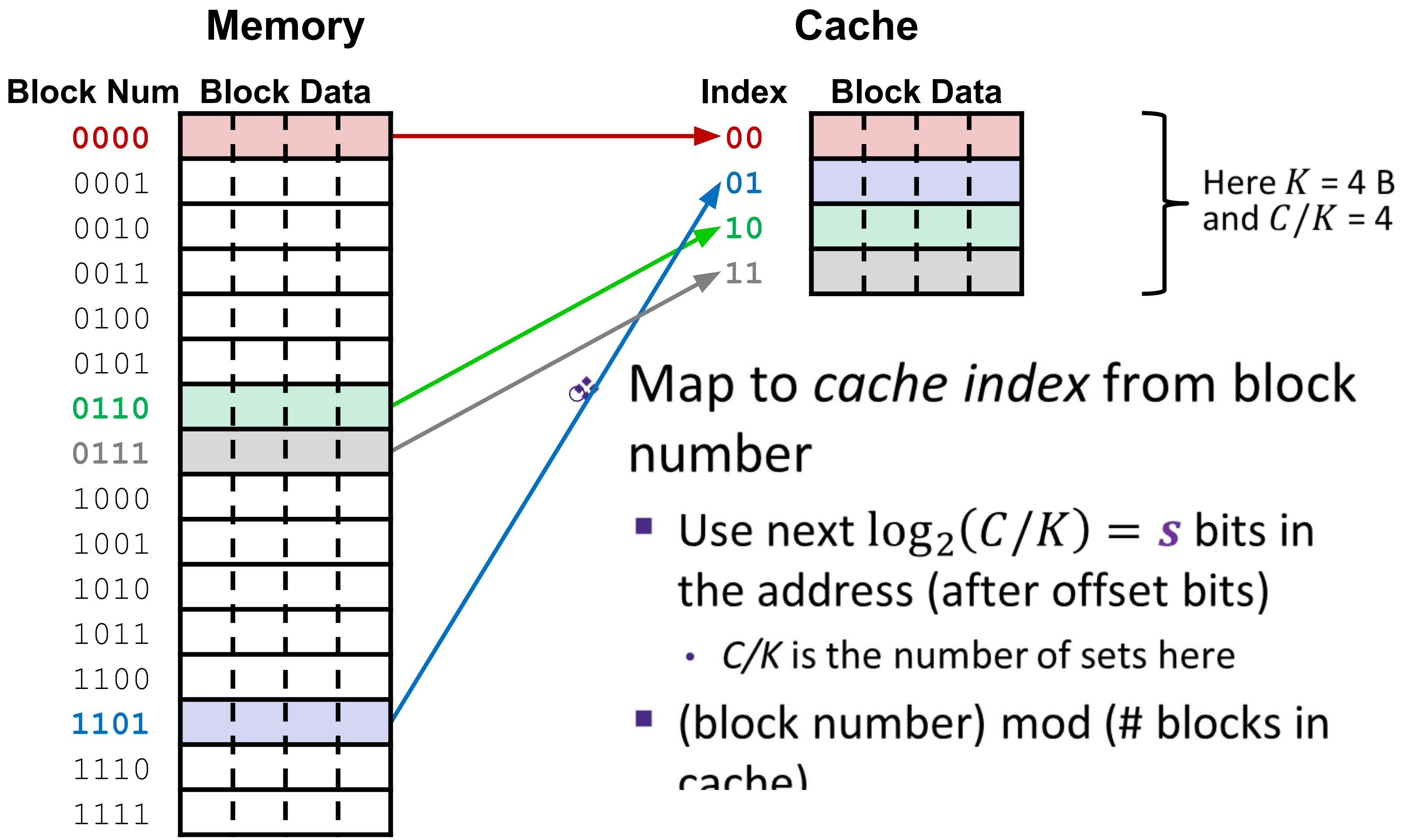
Insert:

```
  5
 27
 34
102
119
```



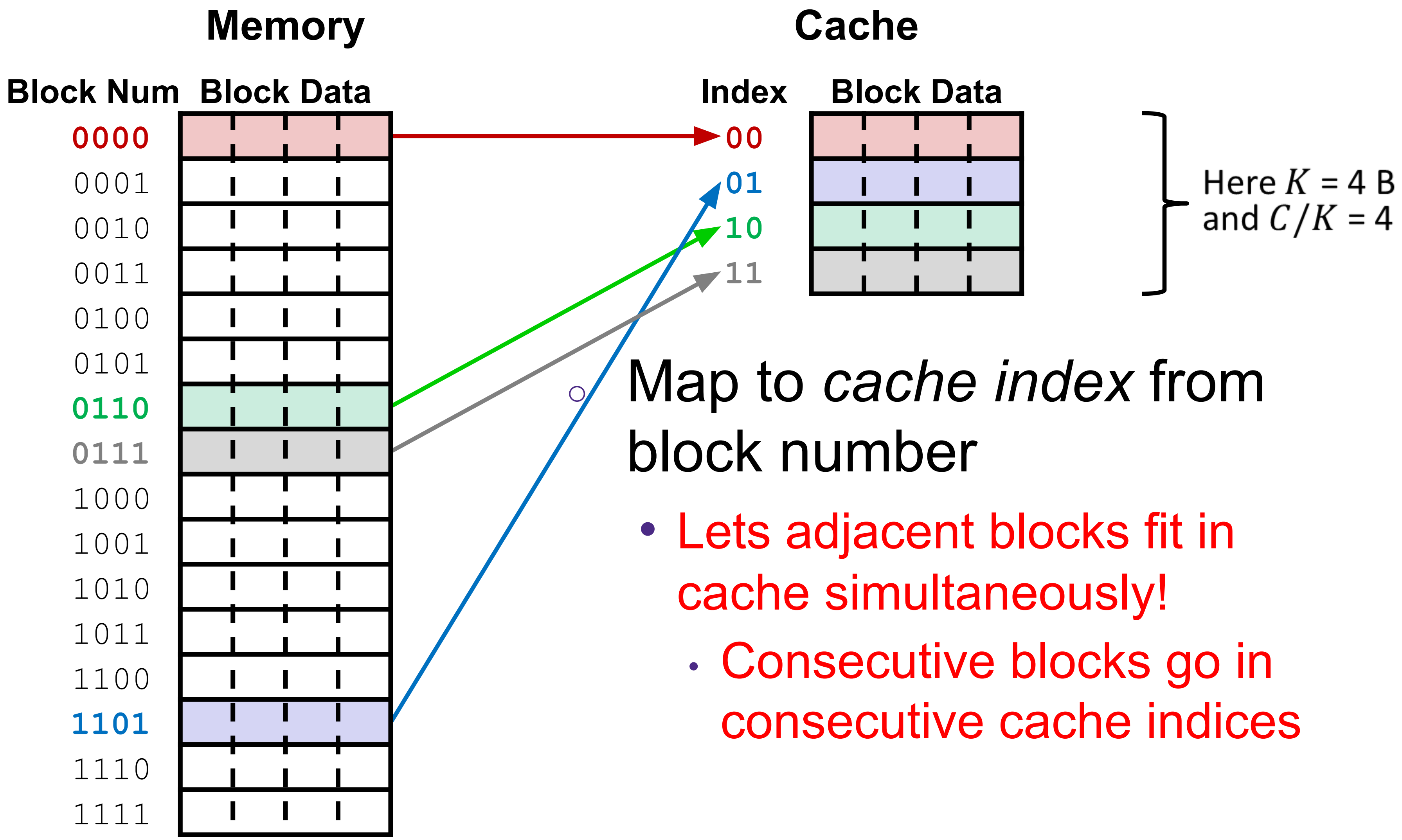
Apply hash function to map data to "buckets"

Place Data in Cache by Hashing Address



- Map to *cache index* from block number
  - Use next $\log_2(C/K) = s$ bits in the address (after offset bits)
    - *C/K* is the number of sets here
  - (block number) mod (# blocks in cache)

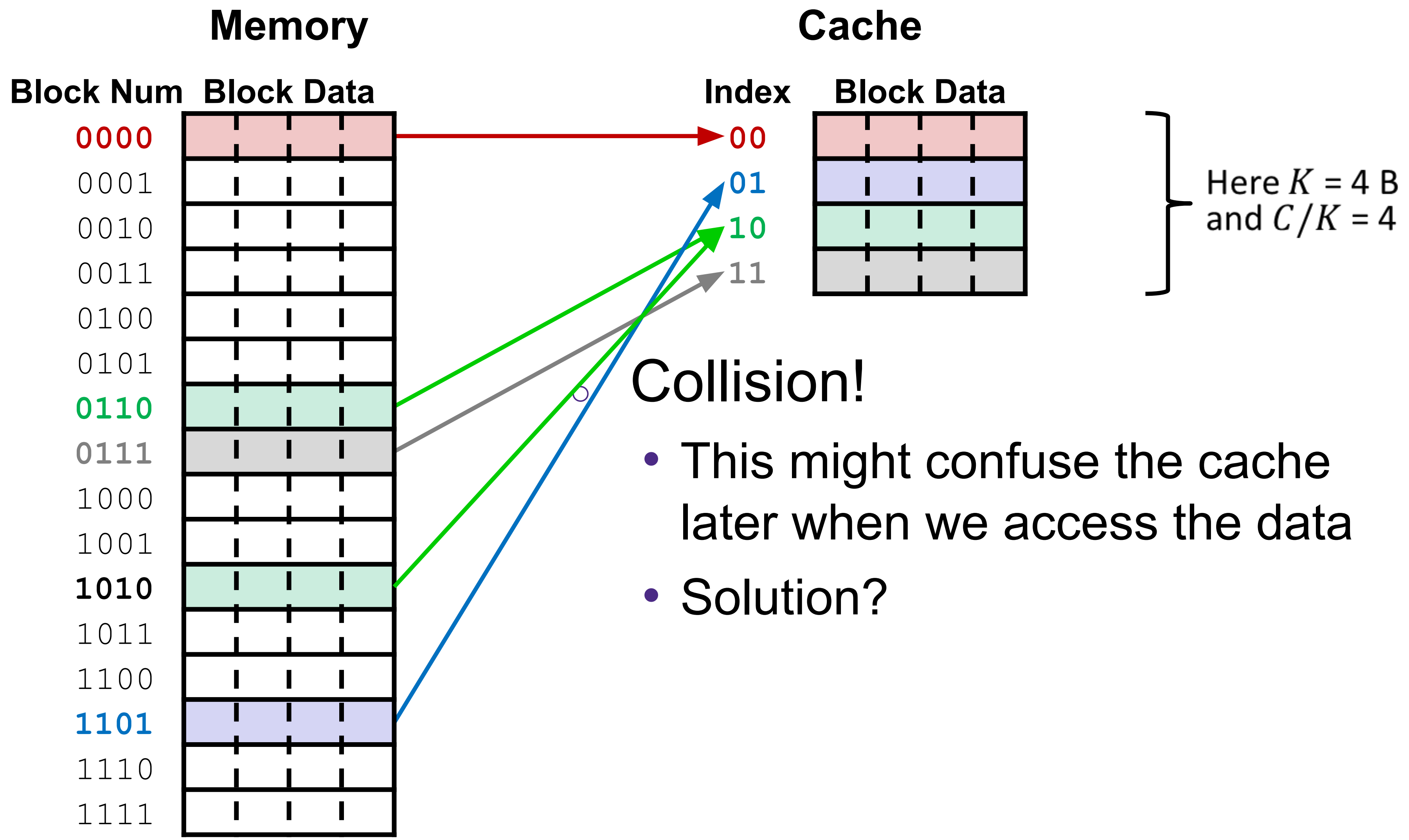
Place Data in Cache by Hashing Address



Map to *cache index* from block number

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

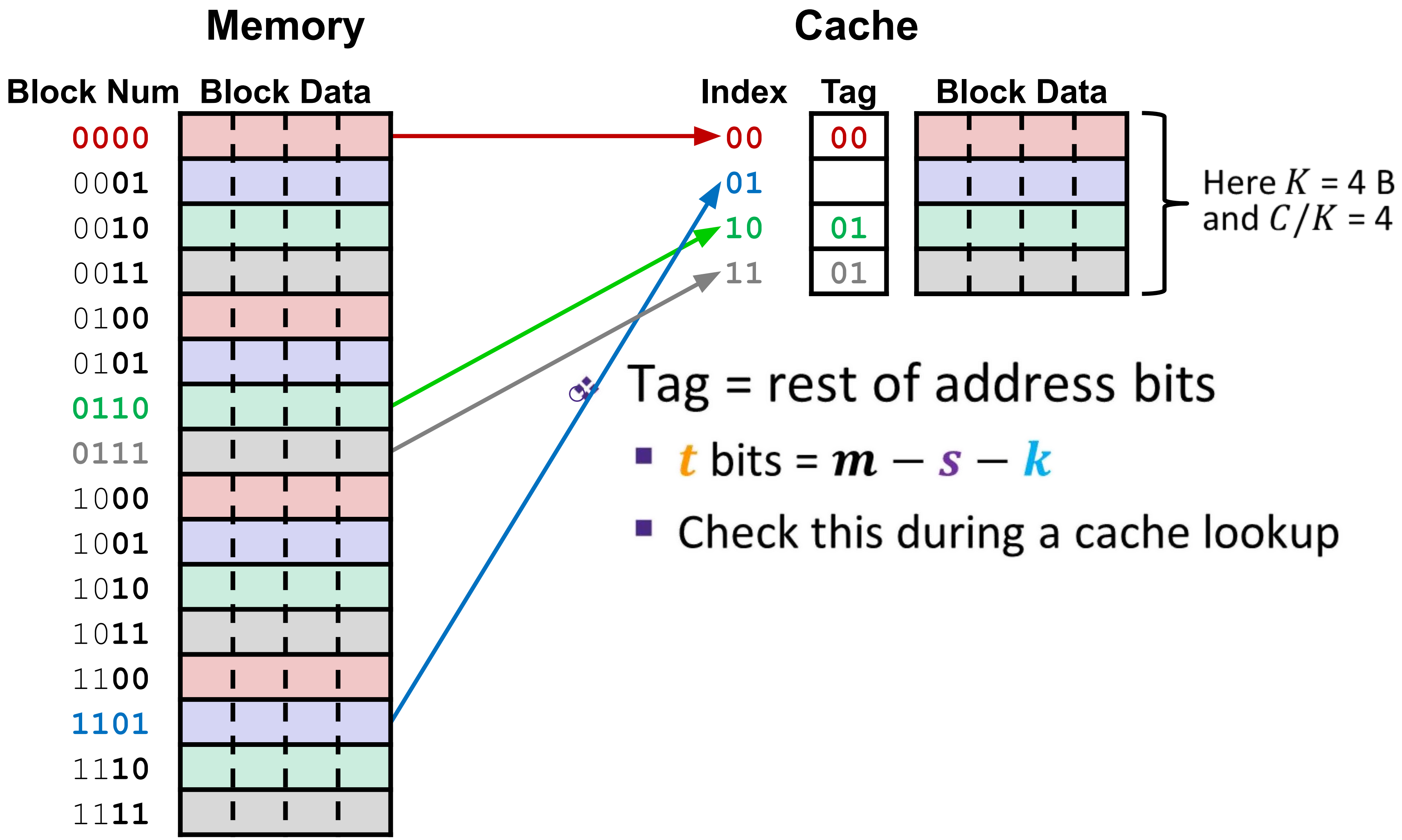
Place Data in Cache by Hashing Address



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:

$$m\text{-bit address: } \underbrace{\boxed{\text{Tag } (t)}\boxed{\text{Index } (s)}}_{\text{Block Number}}\boxed{\text{Offset } (k)}$$

  - **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

Checking for a Requested Address Example

- Using 8-bit addresses.
- Cache Params: block size (K) = 4 B, cache size (C) = 32 B (which means number of sets is C/K = 8 sets).
  - Offset bits (k) = $\log_2(K)$ = **2**
  - Index bits (s) = $\log_2(num\ sets)$ = **3**
  - Tag bits (t) = Rest of the bits in the address = **3**

$\boldsymbol{m}$-bit address: $\underbrace{\boxed{\text{Tag } (t)}\boxed{\text{Index } (s)}}_{\text{Block Number}}\boxed{\text{Offset } (k)}$

- What are the fields for address 0xBA?
  - Tag bits (unique id for block): **0xBA = 1011 1010; Tag = 101**
  - Index bits (cache set block maps to): **Index = 110**

Checking in, caches!

- 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_{10}: miss), (15_{10}: hit), (16_{10}: miss)

🐶 **4 bytes**

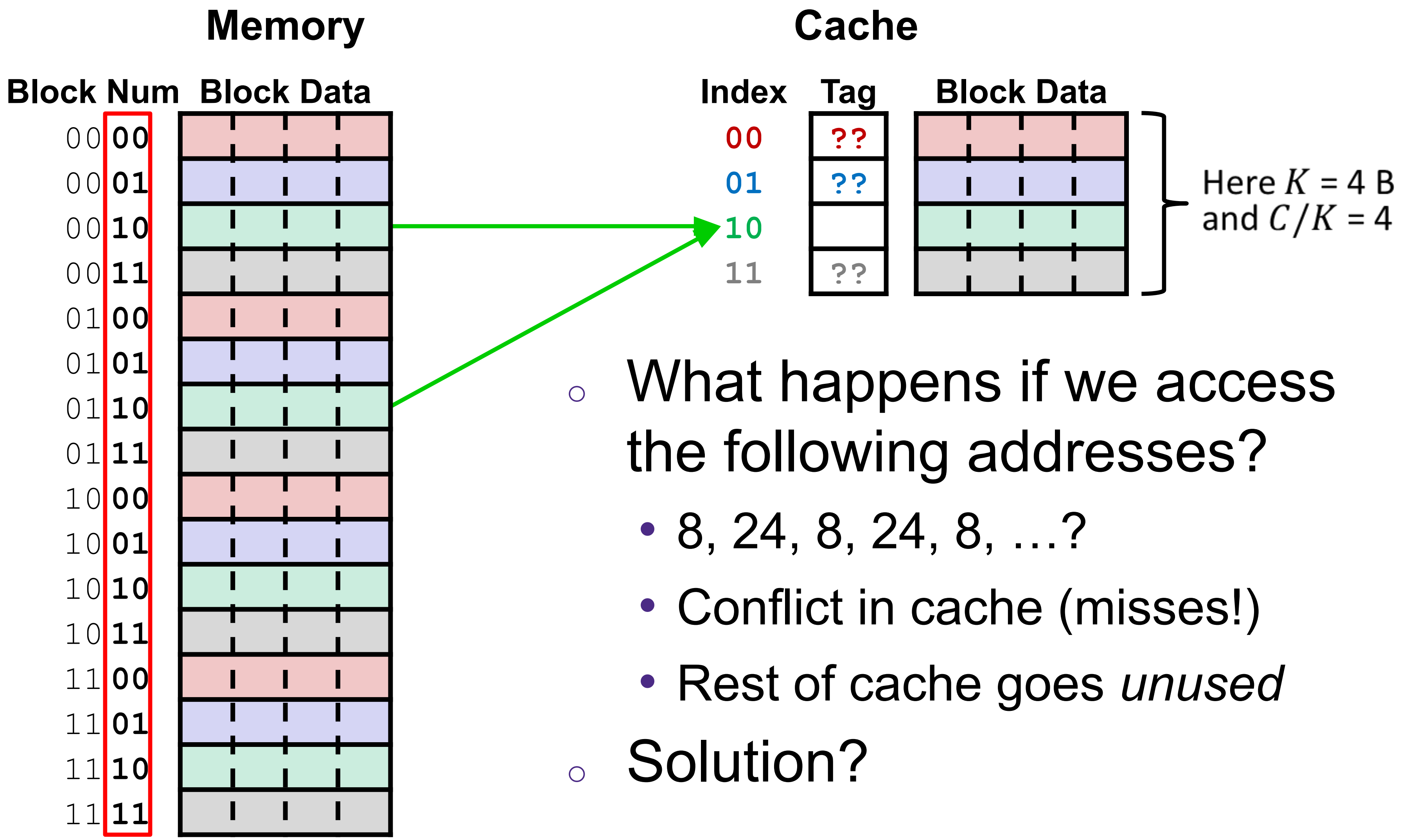
🐈 **8 bytes**

🐑 **16 bytes**

🦄 **32 bytes**

🥶 **Help!**

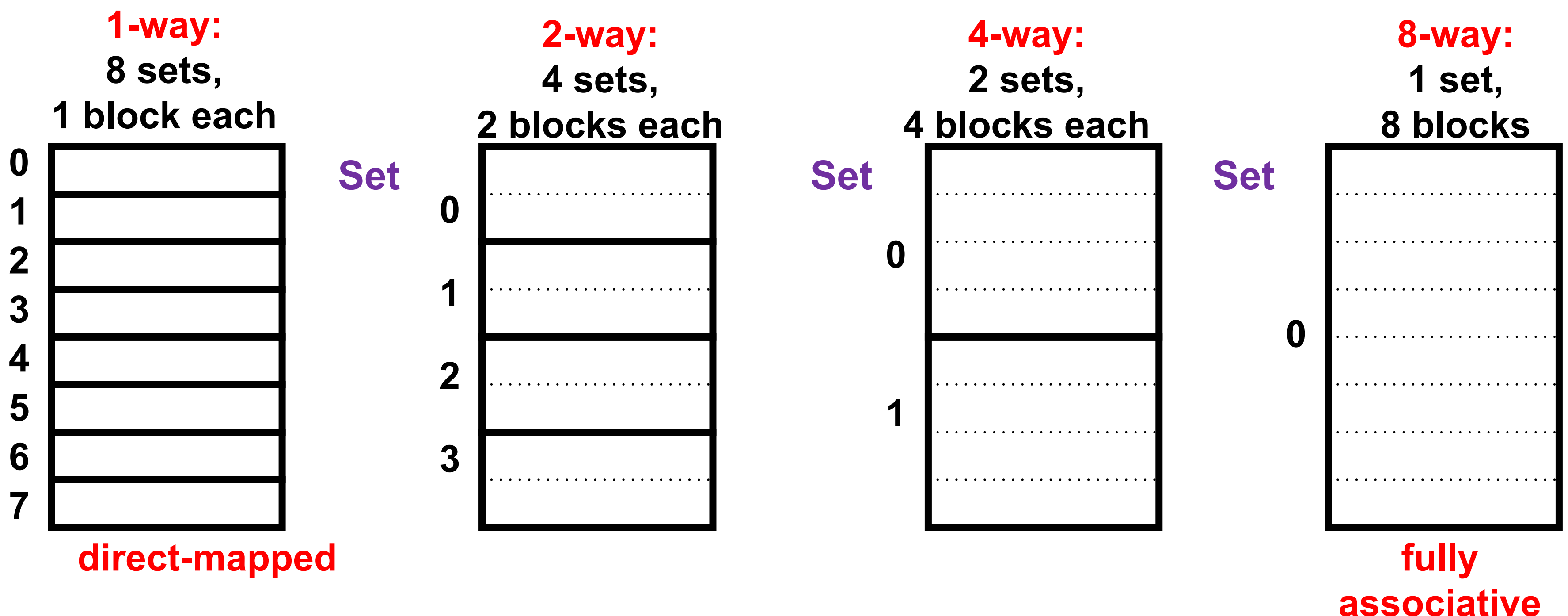
Direct-Mapped Cache Problem



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

Associativity

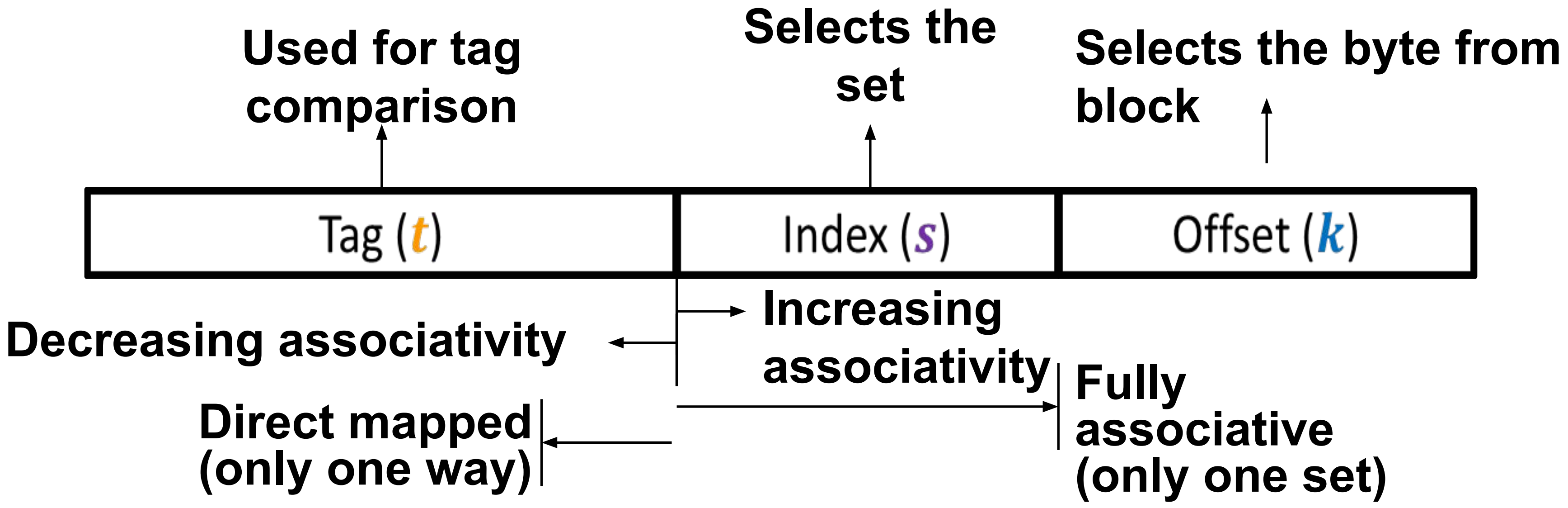
- What if we could store data in any place in the cache?
  - More complicated hardware = more power consumed, slower
- So we *combine* the two ideas:
  - Each address maps to exactly one **set**
  - Each set can store block in more than one **way**



Cache Organization (3)

Note: The textbook uses "b" for offset bits

- 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 $\log_2(C/K/E)$ = s bits of block address
    - Direct-mapped: E = 1, so s = $\log_2(C/K)$ as we saw previously
    - Fully associative: E = C/K, so s = 0 bits



Example Placement

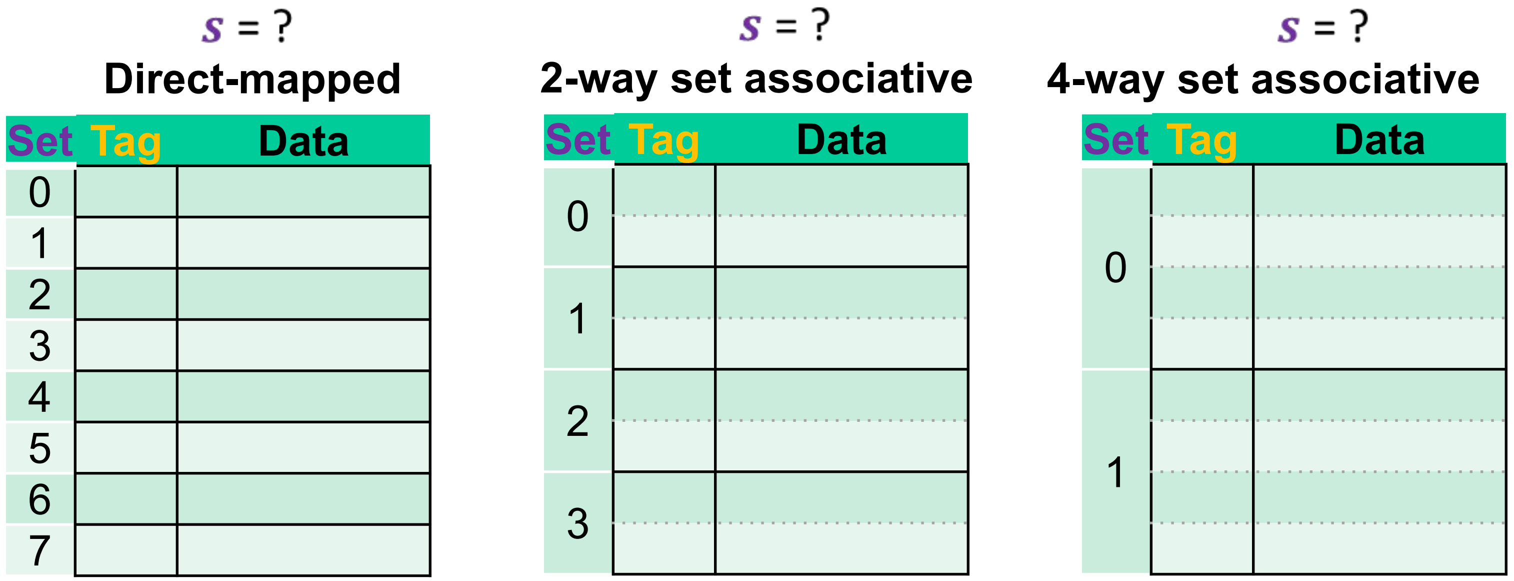
block size:	16 B
capacity:	8 blocks
address:	16 bits

- 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:

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



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

s=3

Direct-mapped

Set	Tag	Data
0		
1		
2		
3	0x30	
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		

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:

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

s=3
Direct-mapped

Set	Tag	Data
0		
1		
2		
3	0x30	
4		
5		
6		
7		

s=2
2-way set associative

Set	Tag	Data
0		
1		
2		
3	0x30	

s = ?
4-way set associative

Set	Tag	Data
0		
1		

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:

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

s=3
Direct-mapped

Set	Tag	Data
0		
1		
2		
3	0x30	
4		
5		
6		
7		

s=2
2-way set associative

Set	Tag	Data
0		
1		
2		
3	0x30	

s=1
4-way set associative

Set	Tag	Data
0		
1	0x30	

Summary

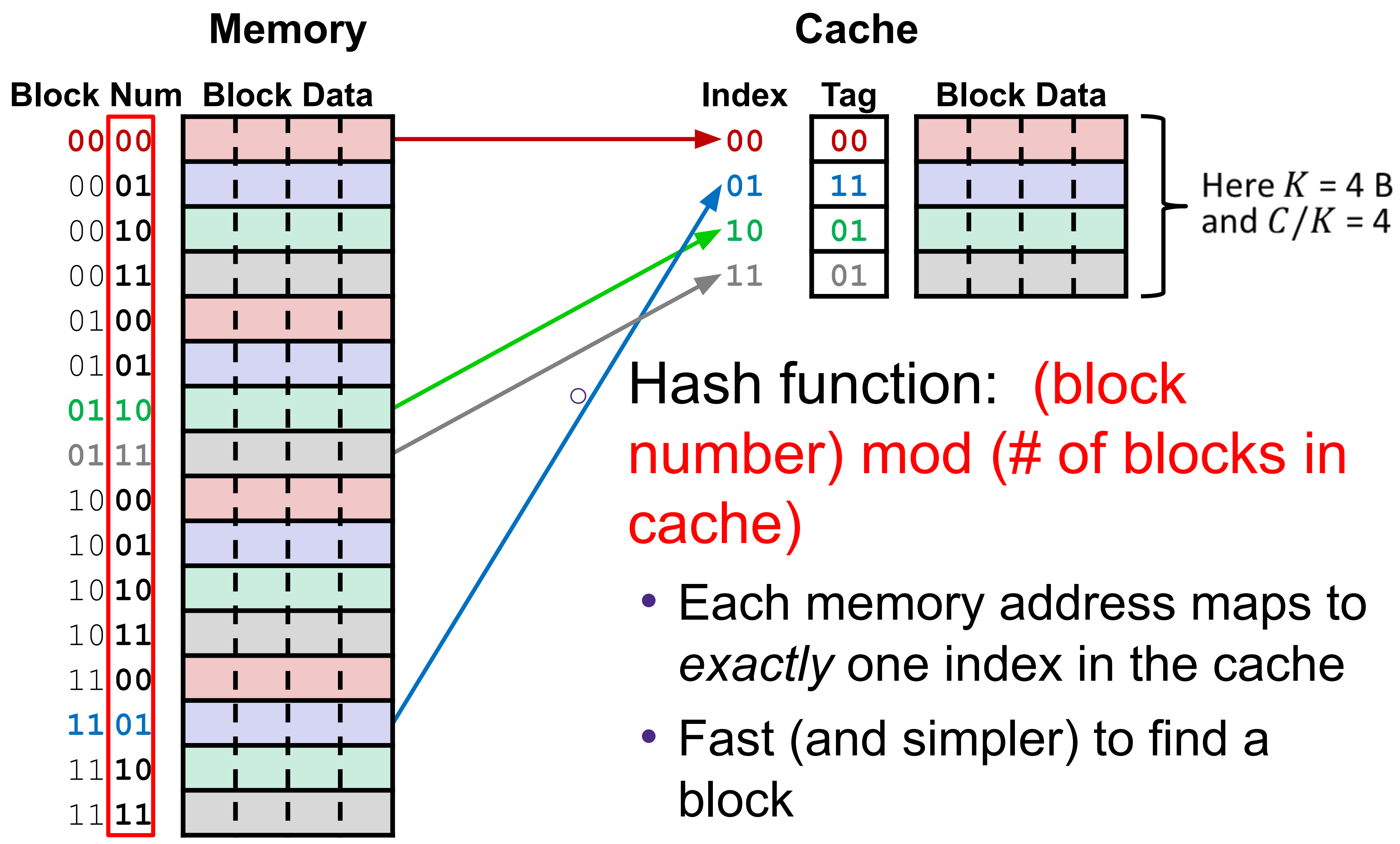
- Memory hierarchy between caches (multi-level), memory, disk
  - Like any other storage, short-term/long-term
- Caches are organized into blocks by hashing addresses
  - Store tag to avoid confusing data

Unit Summary Critique

Critique

- 3 people/breakout!
  - Try to make sure everyone has time to share
  - Helpful feedback includes things that you like and don't like!
  - No need to be *critical*
- These are personal representations of knowledge!
  - You're allowed to ignore the feedback that you get
  - But, do make sure to listen!

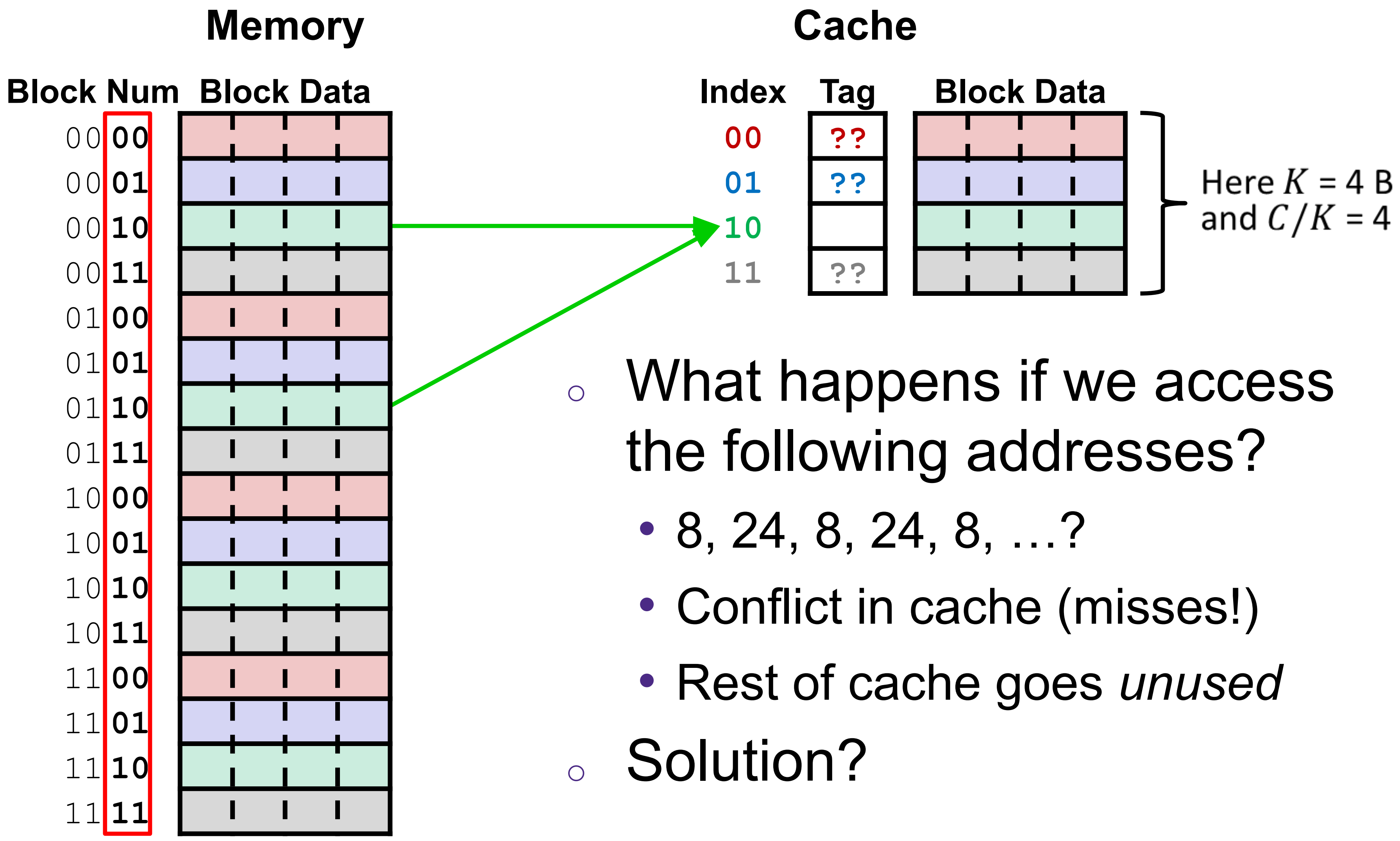
Direct-Mapped Cache



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

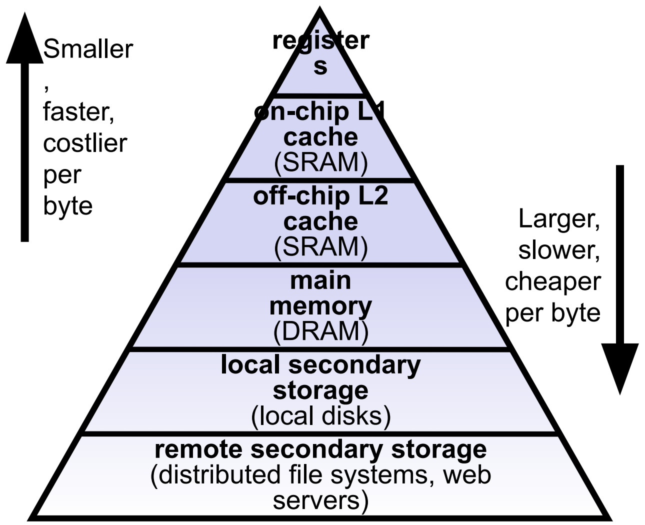
- Each memory address maps to *exactly* one index in the cache
- Fast (and simpler) to find a block

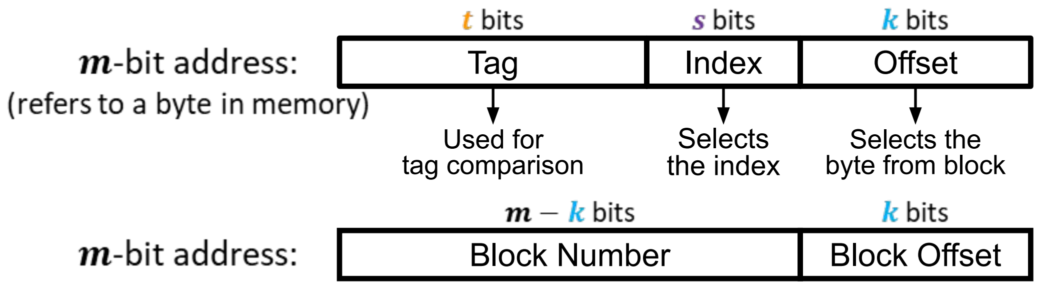
Direct-Mapped Cache Problem



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

Notes Diagrams


Smaller
,
faster,
costlier
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)
Larger,
slower,
cheaper
per byte


m-bit address:
(refers to a byte in memory)
t bits
s bits
k bits
Tag
Index
Offset
Used for tag comparison
Selects the index
Selects the byte from block
$m-k$ bits
k bits
m-bit address:
Block Number
Block Offset