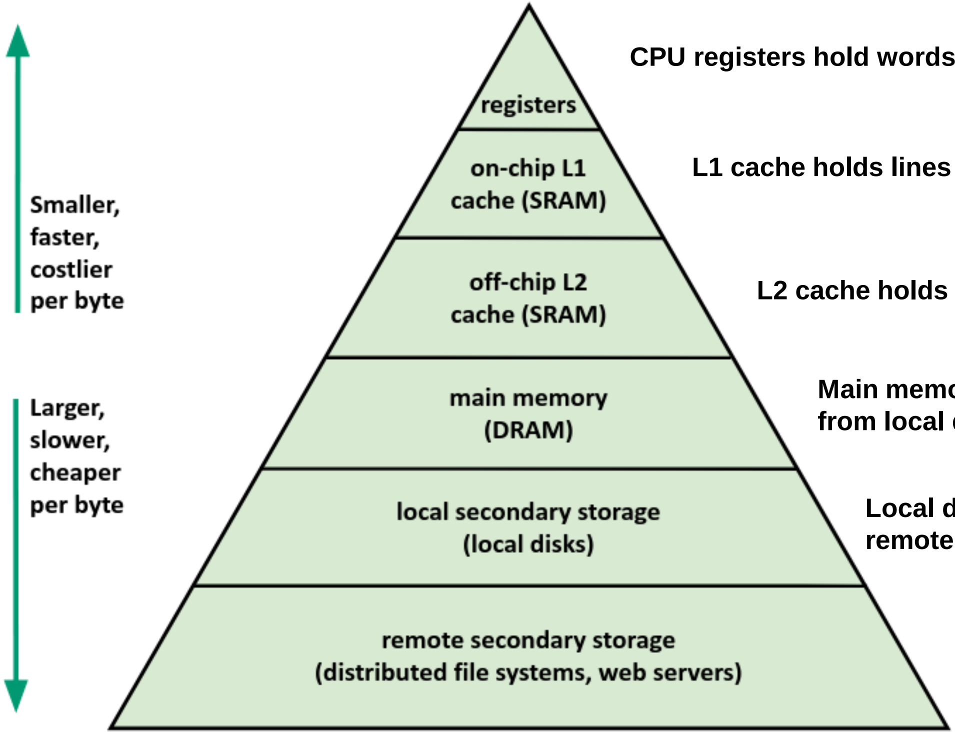
Caches II

Memory Hierarchy Review



CPU registers hold words retrieved from L1 cache

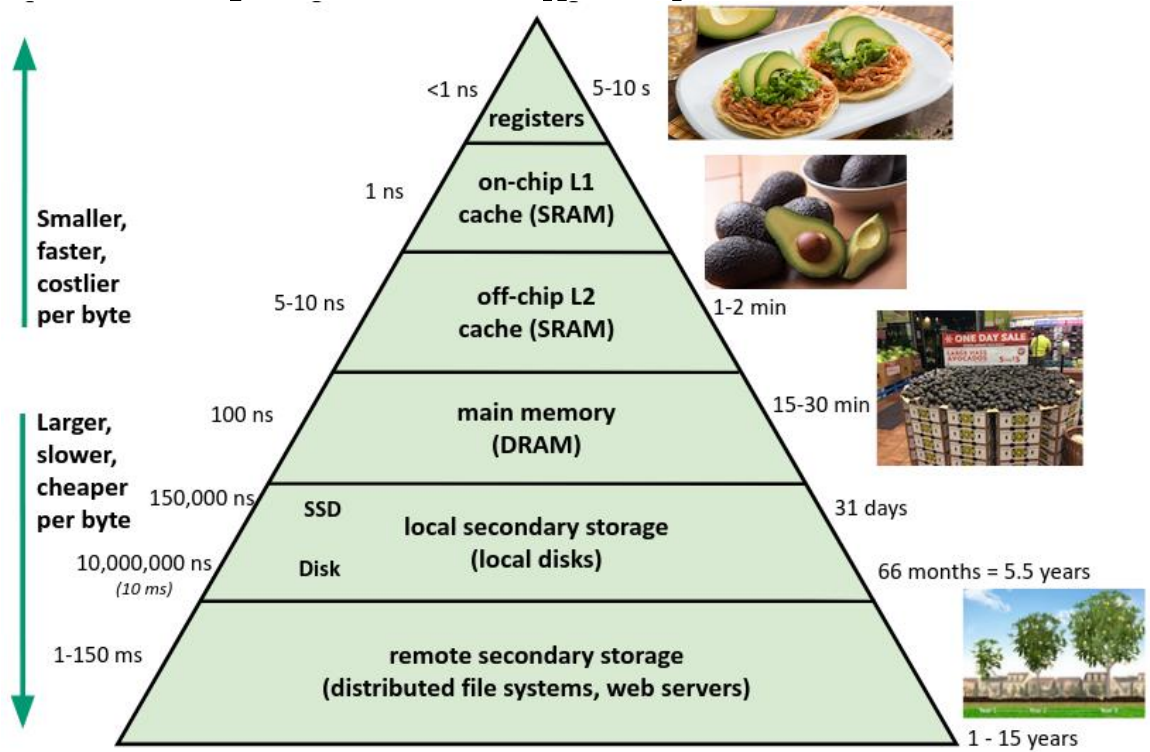
L1 cache holds lines (i.e. blocks) retrieved from L2 cache

L2 cache holds lines retrieved from main memory

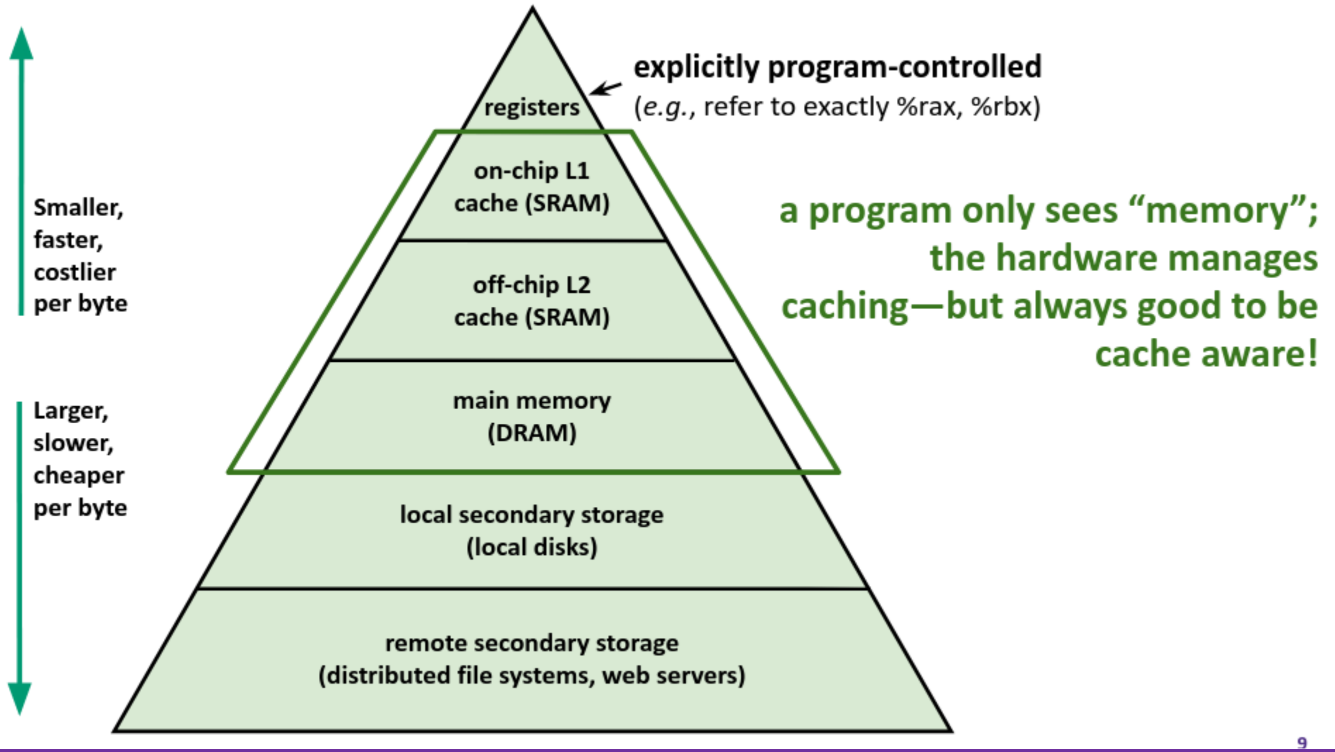
Main memory holds disk blocks retrieved from local disks

Local disks hold files retrieved from disks on remote network servers

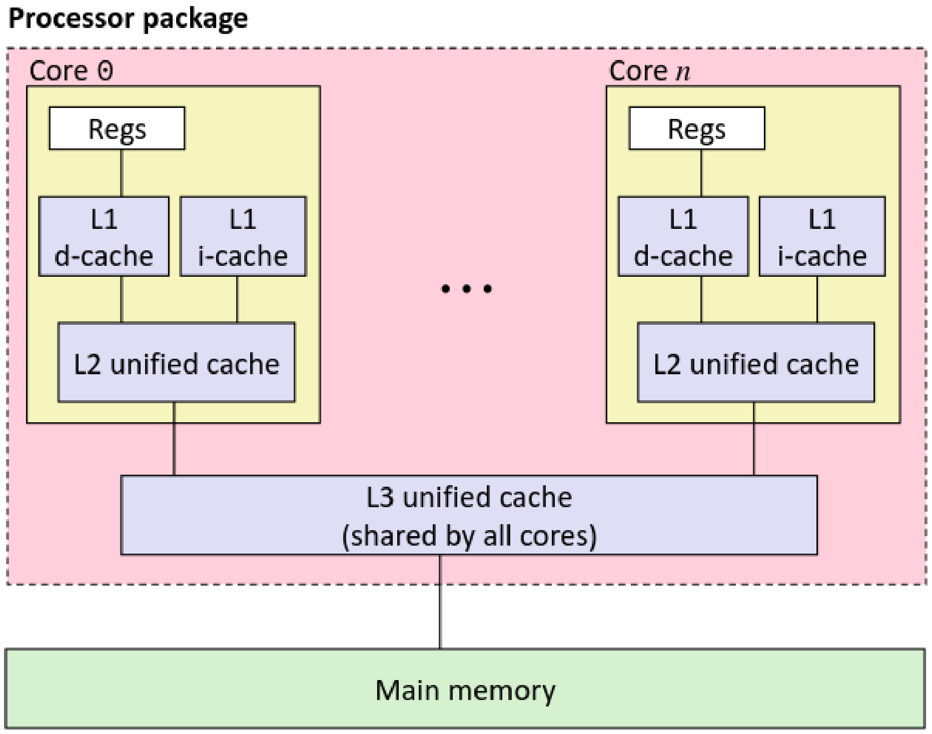
Memory Hierarchy Review (pt 2)



Memory Hierarchy Review (pt 3)



Example Microarchitecture: Intel Core i7



- **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

Caches

- 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

Review Question

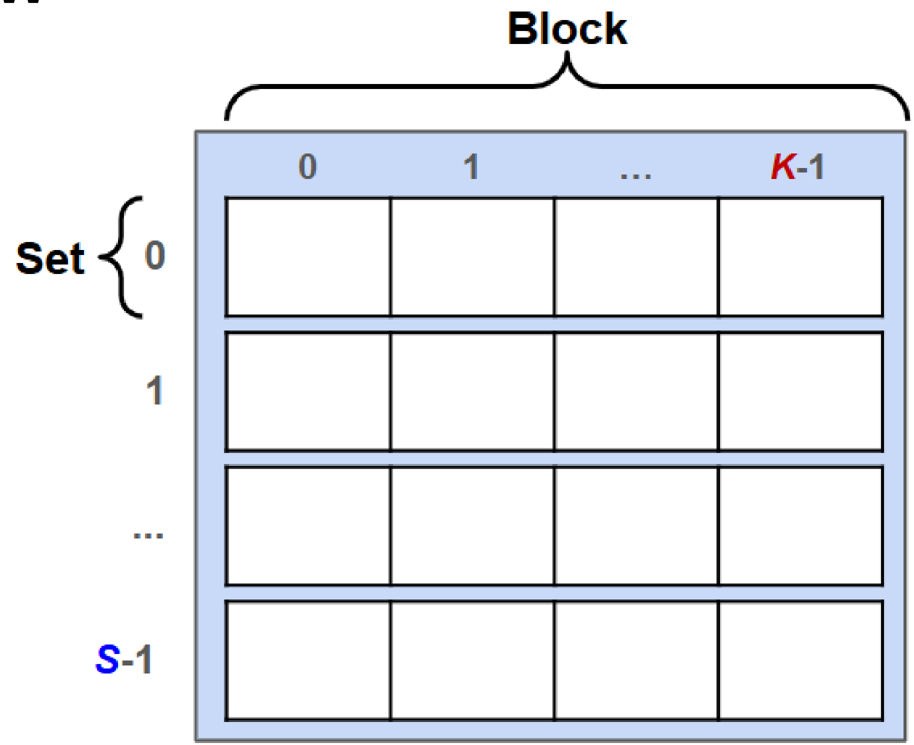
We have a **direct-mapped cache** with the following parameters:

- Block size of 8 bytes
- Cache size of 4 KiB

1. How many blocks can the cache hold?
2. How many bits wide is the block offset field?
3. Which of the following addresses (could be multiple) would fall under block 3?
   A) 0x3
   B) 0x1F
   C) 0x30
   D) 0x38

Reading Terminology Review

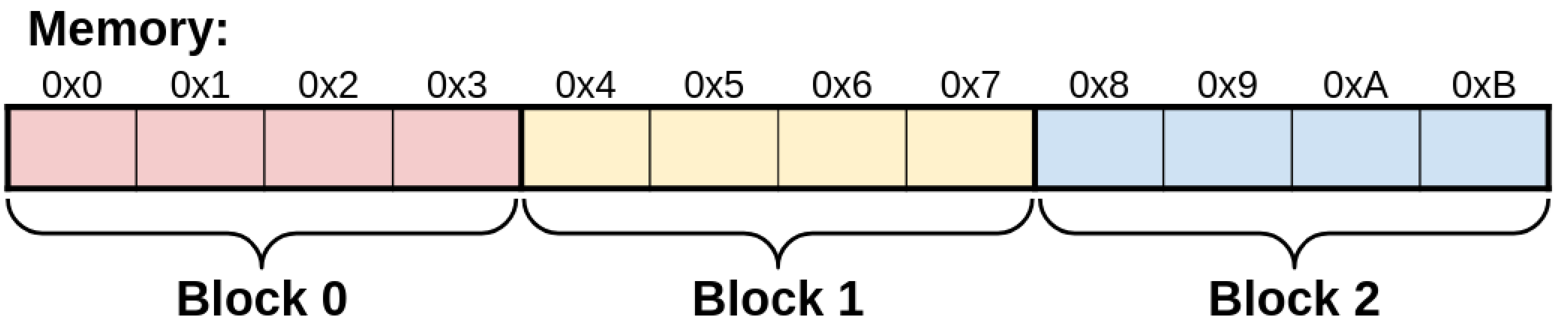
- Cache Parameters
    - Block size (***K***)
    - Cache size (***C*** bytes, or ***S*** sets)
- Address fields
    - Block offset (***k*** bits wide)
    - Block number (also called “block address”)
        - Index field (***s*** bits wide)
        - Tag (***t*** bits wide)



Cache Organization: Block Size

- **Block Size** (***K***): unit of transfer between cache and memory
  - Given in bytes and <u>always</u> a power of 2
  - Blocks are aligned and consist of adjacent bytes
    - Spatial locality!

<u>Example</u>: ***K*** = 4B



Cache Organization: Block Size (pt 2)

- Given block size ***K***:
  - Address ÷ ***K*** = **block number** (i.e. which block this address belongs to)
  - Address % ***K*** = **block offset** (i.e. where in the block this address is located)
- Define ***k*** = $\log_2(K)$
  - Lowest ***k*** bits of address tell us the block offset

	m - ***k*** bits	***k*** bits
***m*-bit address**	block number	block offset

<u>Example</u>: If we have 6-bit addresses and ***K*** = 4B, which block does address 0x15 belong to? What is its offset within that block?

Cache Organization: Cache Size

- **Cache size** (***C***) = how much data the cache can hold
  - Does not include any metadata
  - If size is (***C***) bytes, then the cache can hold ***C***/***K*** blocks
    - Ex: if ***C*** = 32KiB and ***K*** = 64B, then the cache can hold 512 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 data structure provides fast lookup?

Hash Tables for Fast Lookup

- Divide cache into “buckets” (**sets**)
  - Apply hash function to map each block to a set
  - What’s a simple hash function we can use?

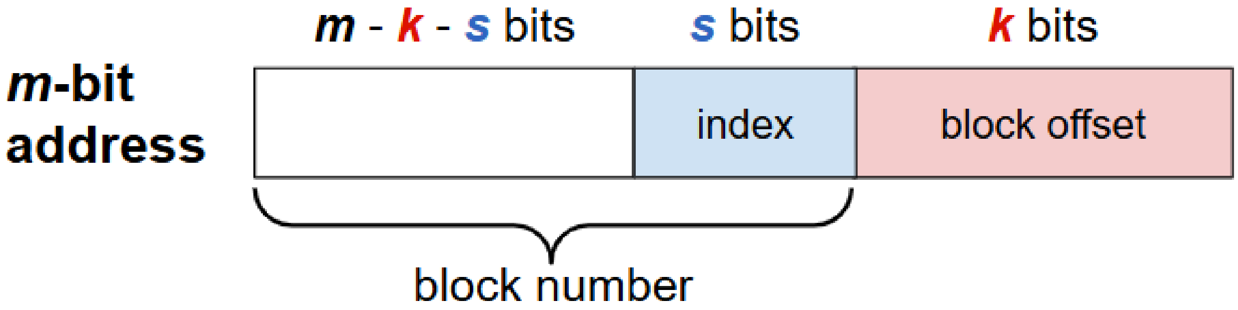
<u>Example</u>: If we have 10 sets, what indices should each of these blocks go into?

- 5
- 27
- 34
- 102

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	

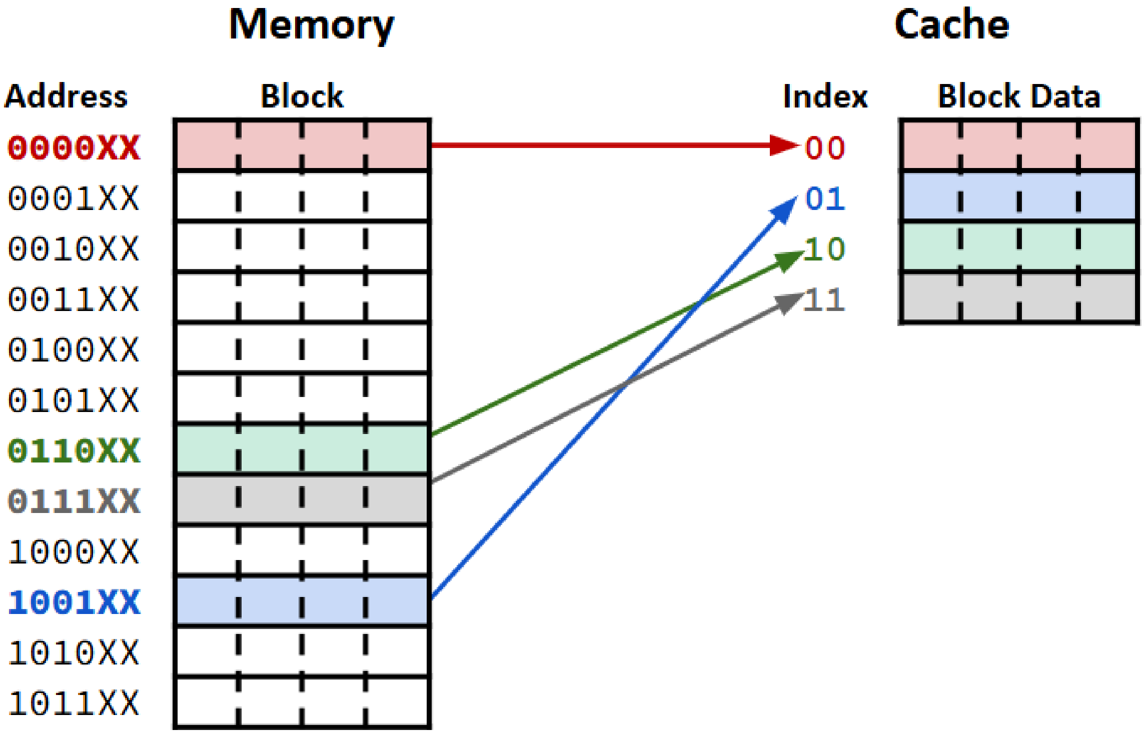
Cache Organization: Sets

- **Number of sets** (***S***) = cache size (***C***) ÷ block size (***K***)
  - <u>Always</u> a power of 2
  - Block number % ***S*** = **set index** (i.e. where in the cache this block goes
- Define ***s*** = $\log_2$(***S***)
  - Lowest ***s*** bits of the **block number** tell us the index



Memory and Cache Example

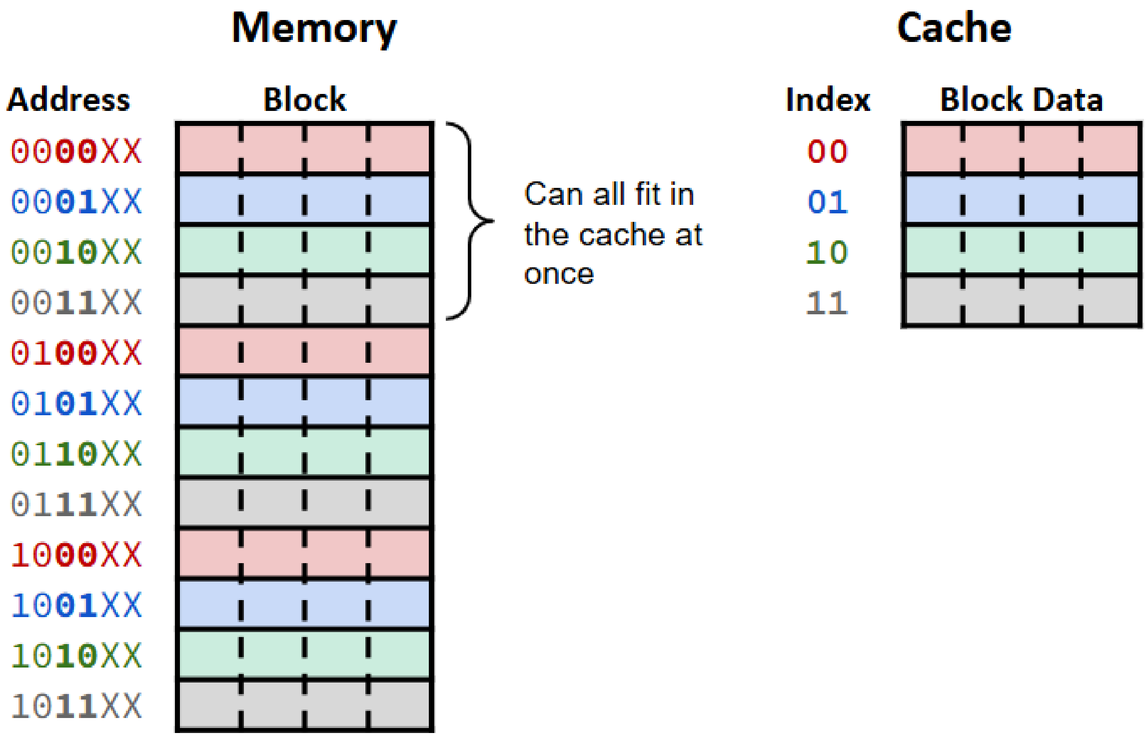
In this example:
K = 4B
S = 4



- Map blocks to cache sets
  - Block# mod ***S*** = index

Memory and Cache Example (pt 2)

In this example:
K = 4B
S = 4



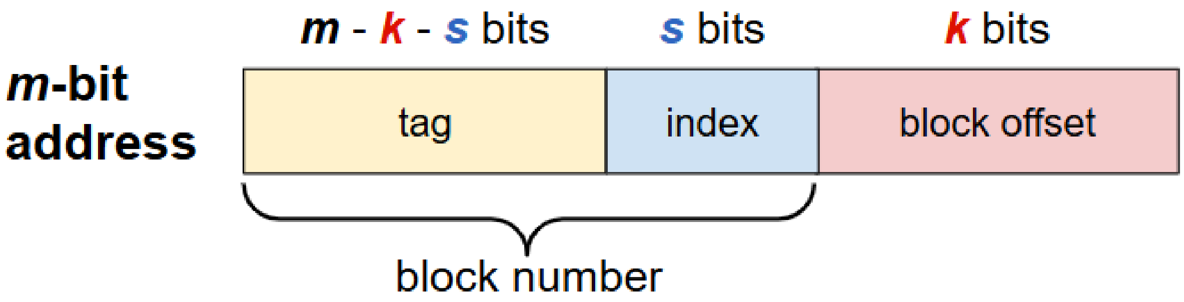
- Map blocks to cache sets
  - Block# mod ***S*** = index
- **Adjacent blocks can fit into the cache at the same time!**
  - Map to consecutive sets

Polling Question

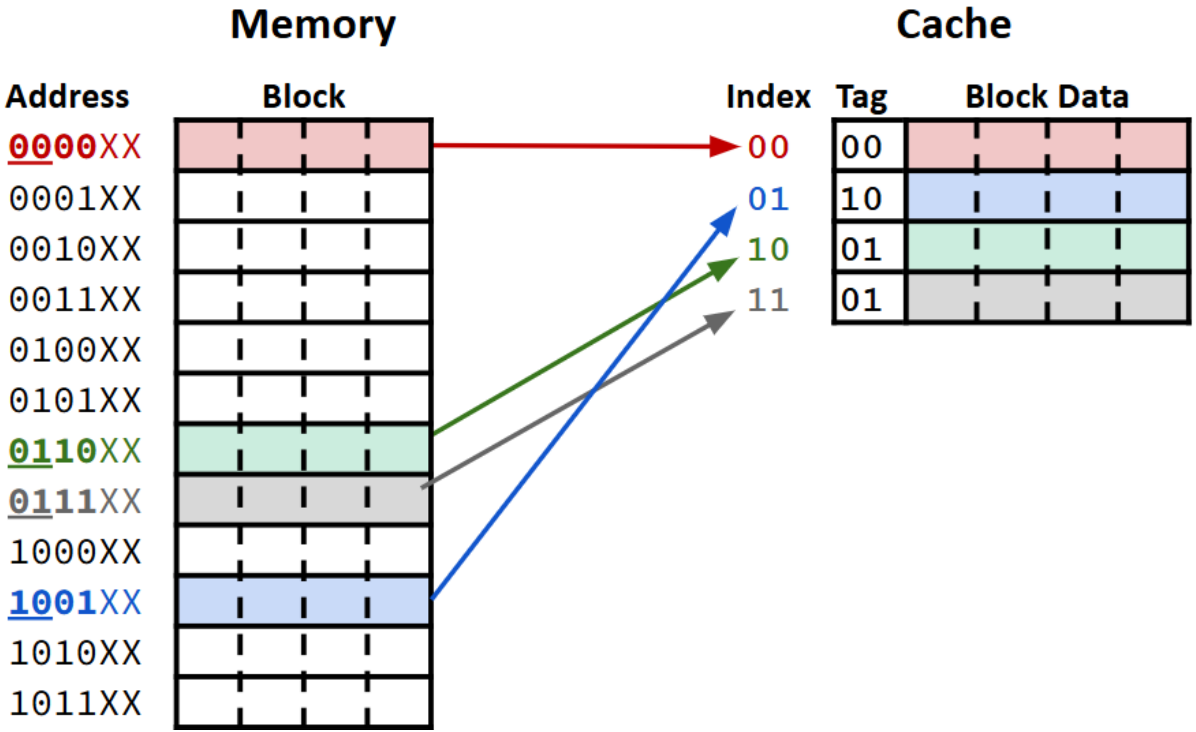
- 6-bit addresses, block size ***K*** = 4 B, and our cache holds ***S*** = 4 blocks
- The CPU requests data at address `0x2A`.
  - Which index can this address be found in?
  - Which 3 other addresses can be found in the same block? (No Ed poll for this one)

Cache Organization: Sets and Tags

- **Problem**: multiple blocks in memory will map to the same set
  - There will always be more blocks than sets because cache is smaller than memory
  - If we look in a set in the cache, how can we tell which block in memory it has?
- **Solution**: store the remaining bits of the block number as a **tag**



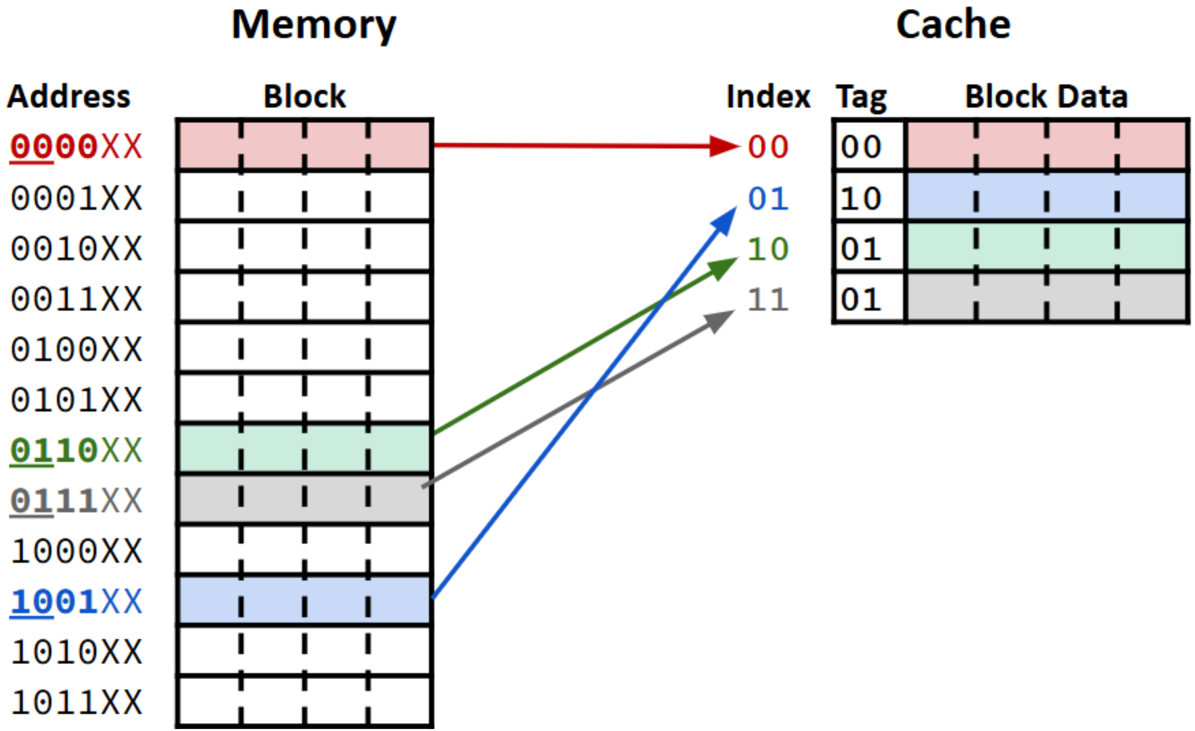
Memory and Cache Example (pt 3)



In this example:
K = 4B
S = 4

- Save the tag in the cache along with the data block
    - All bits of the block# not used for the index

Memory and Cache Example (pt 4)



In this example:
K = 4B
S = 4

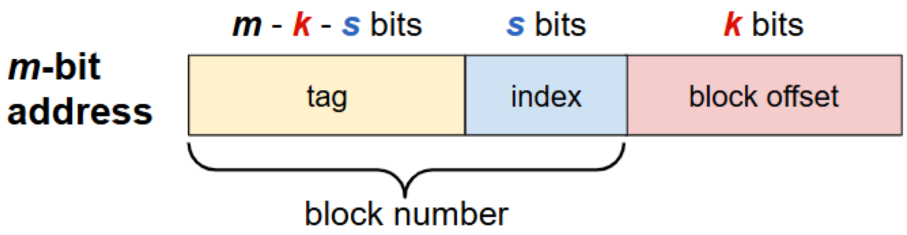
- Save the tag in the cache along with the data block
  - All bits of the block# not used for the index

- On lookup, check the tag to make sure we have the right block

Accessing Data

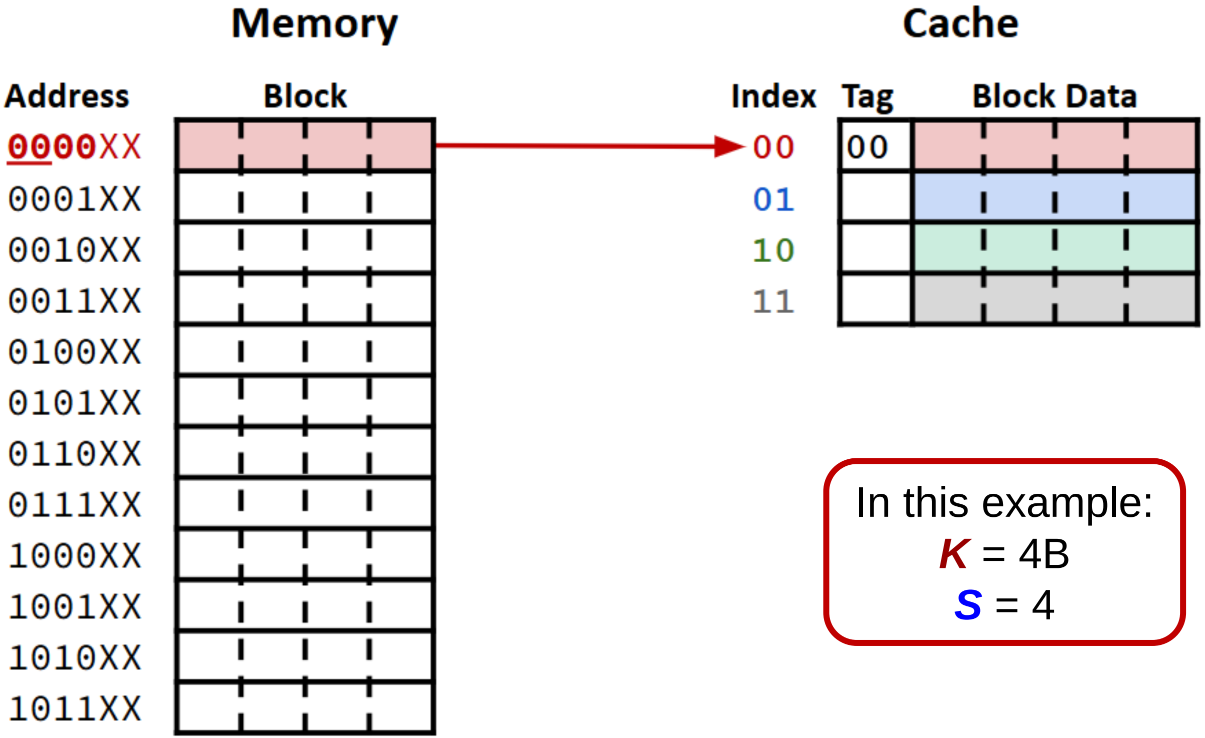
1. CPU requests a chunk of data at some address
2. Break address up into **Tag**, **Index**, and **Offset**
   a. **O** = lowest ***k*** bits, **I** = next ***s*** bits, **T** = remaining bits

   b. Check set **I** in the cache
   c. If the tag matches **T**, return the data starting at offset **O**
   d. Otherwise, load block from memory
      i. Goes into set **I**, update tag to match
      ii. Then return the data at offset **O**



Accessing Data Example: Before

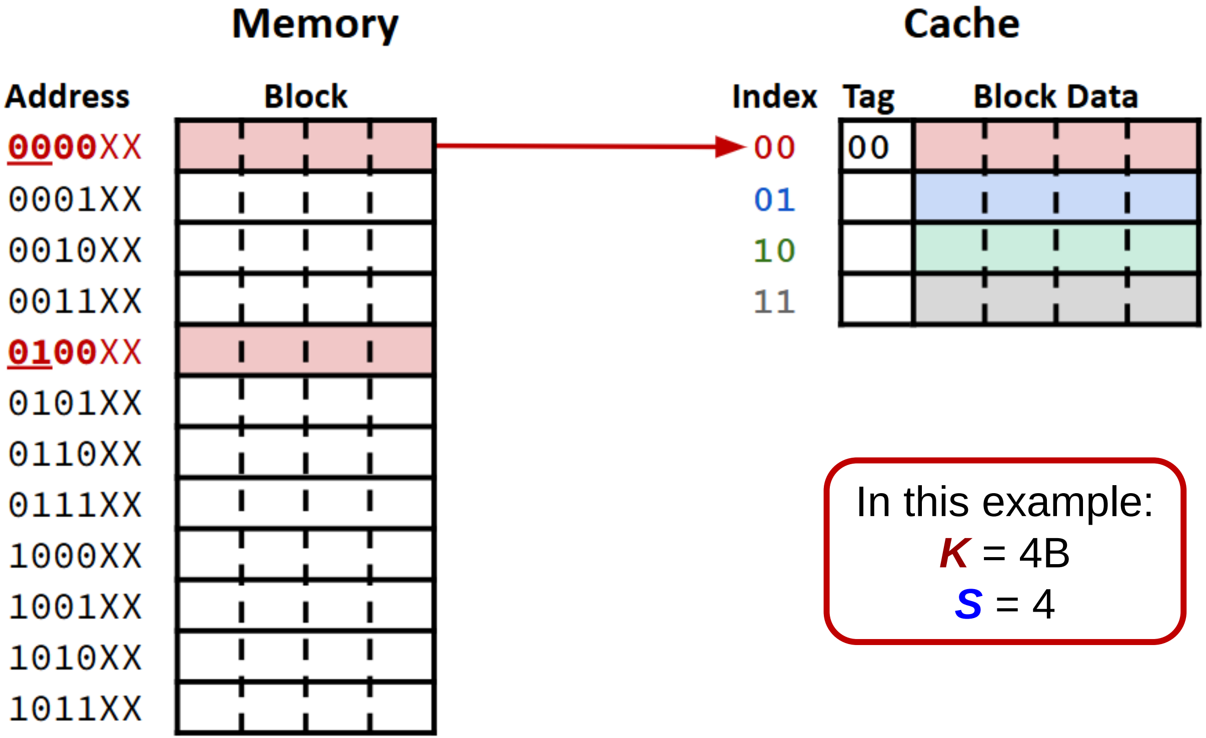
- Block 0 already loaded into the cache
- CPU requests 2B of data at address `0b010001`



In this example:
K = 4B
S = 4

Accessing Data Example: T/I/O breakdown

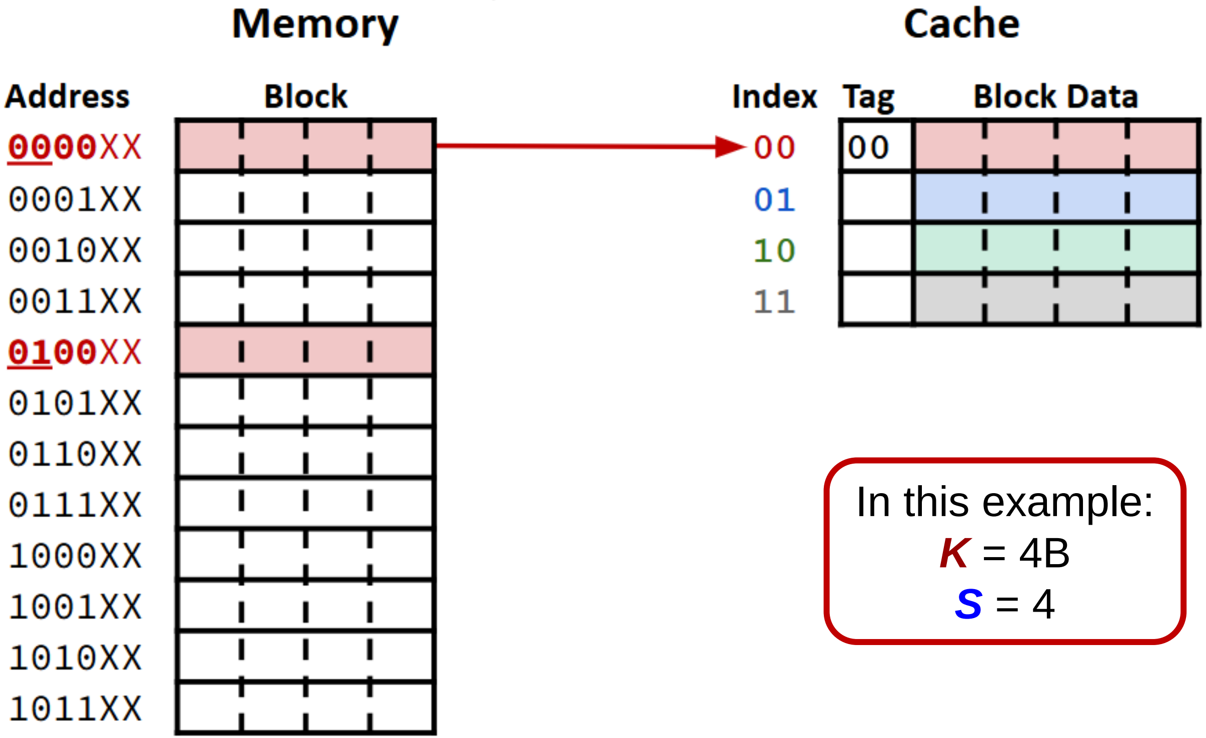
- *k* = 2, *s* = 2
- CPU requests data at address `0b010001`
    - T = `0b01`
    - I = `0b00`
    - O = `0b01`



In this example:
K = 4B
S = 4

Accessing Data Example: Checking Set

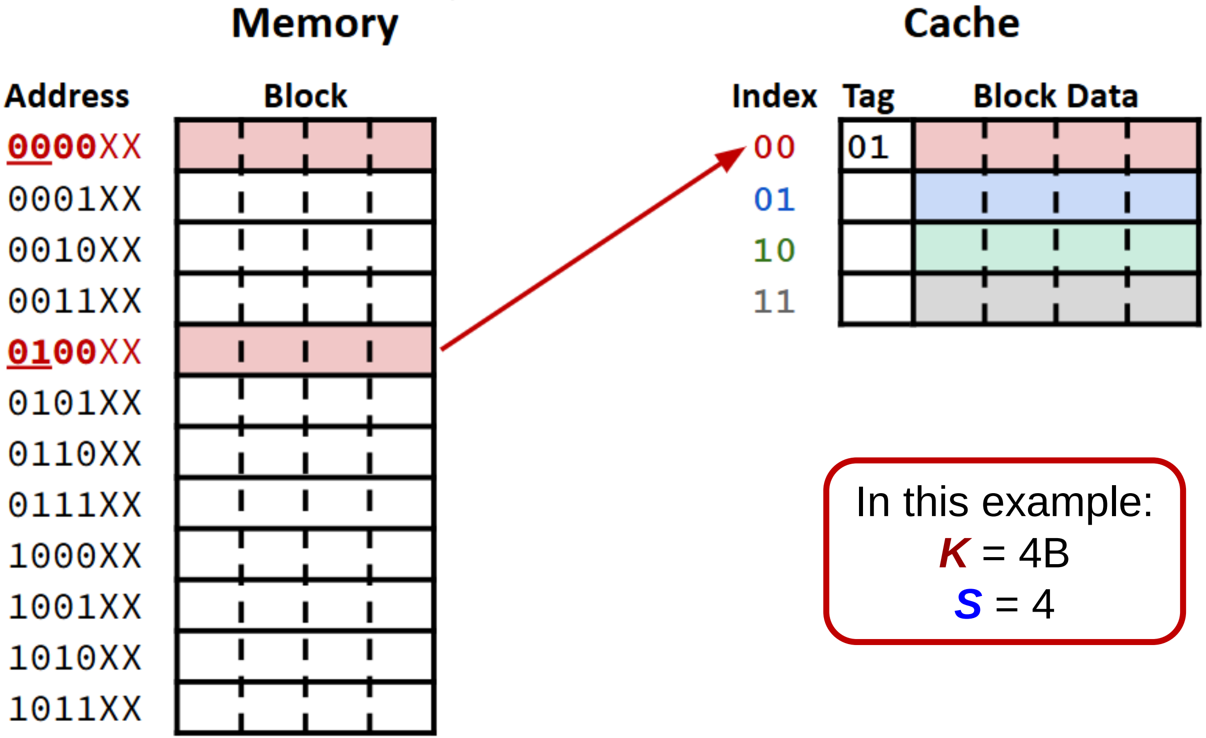
- T = `0b01`, I = `0b00`, O = `0b01`
- Set 0 has tag 00, doesn't match
  - *Cache miss!*



In this example:
K = 4B
S = 4

Accessing Data Example: Loading from Memory

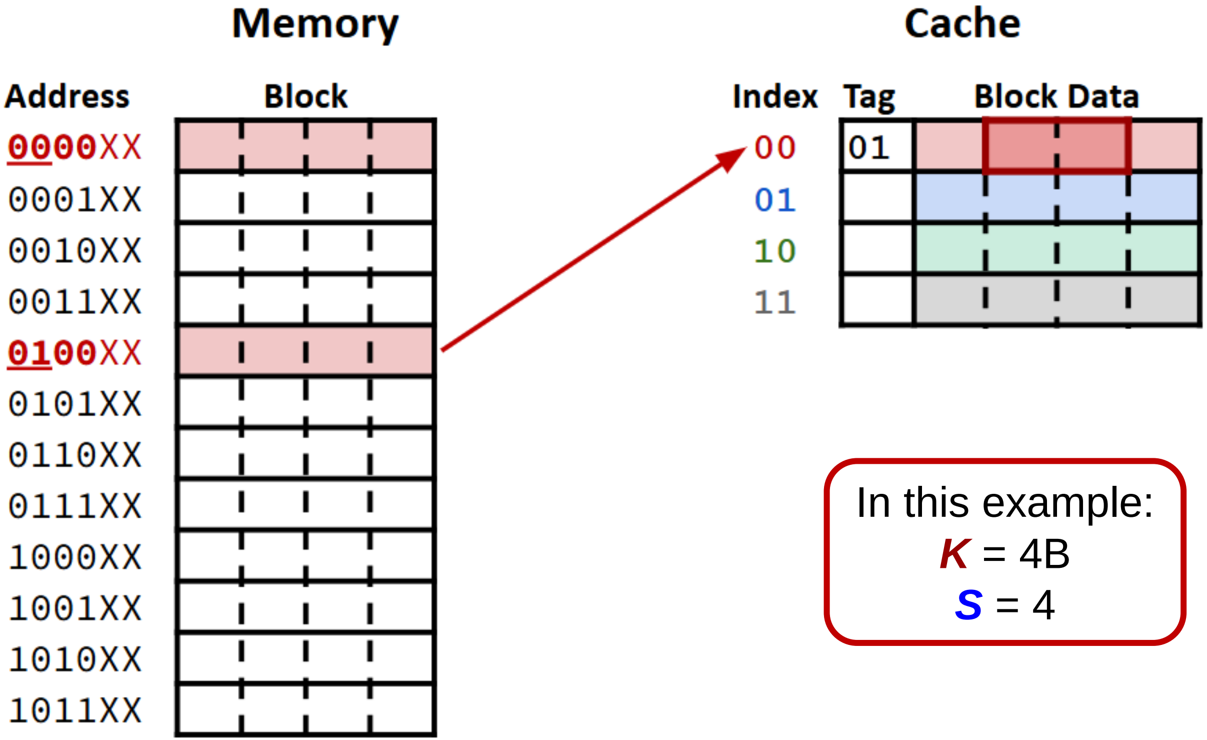
- **T** = `0b01`, **I** = `0b00`, **O** = `0b01`
- Store block 4 (0b0100) into the cache in set 0
  - Update Tag



In this example:
K = 4B
S = 4

Accessing Data Example: Returning Data

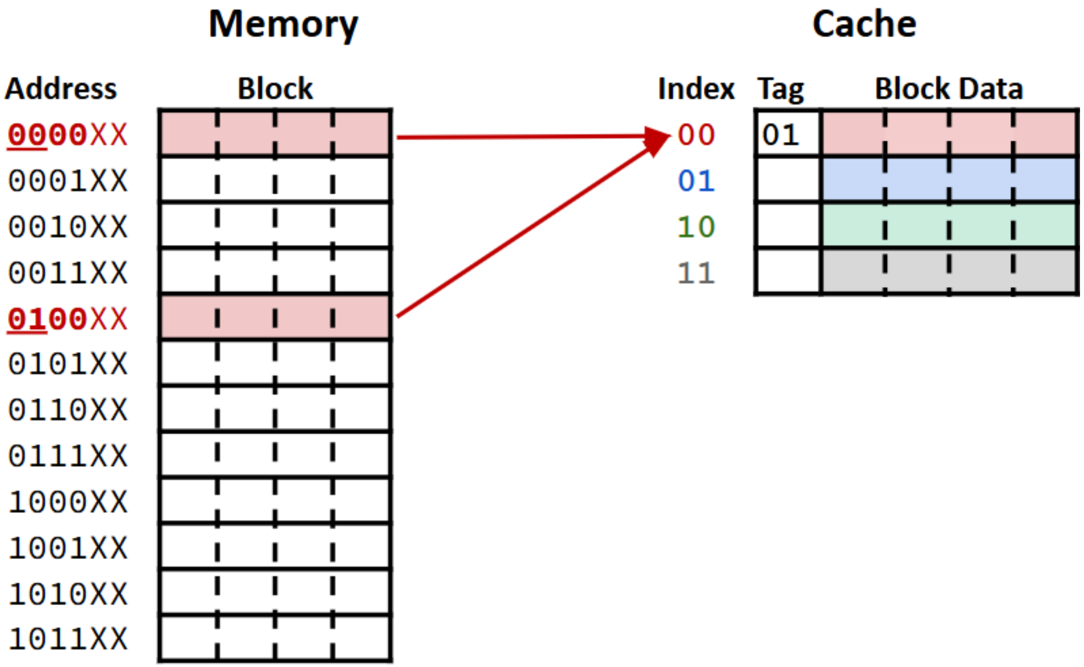
- **T** = `0b01`, **I** = `0b00`, **O** = `0b01`
- Return data starting at offset 1



In this example:
K = 4B
S = 4

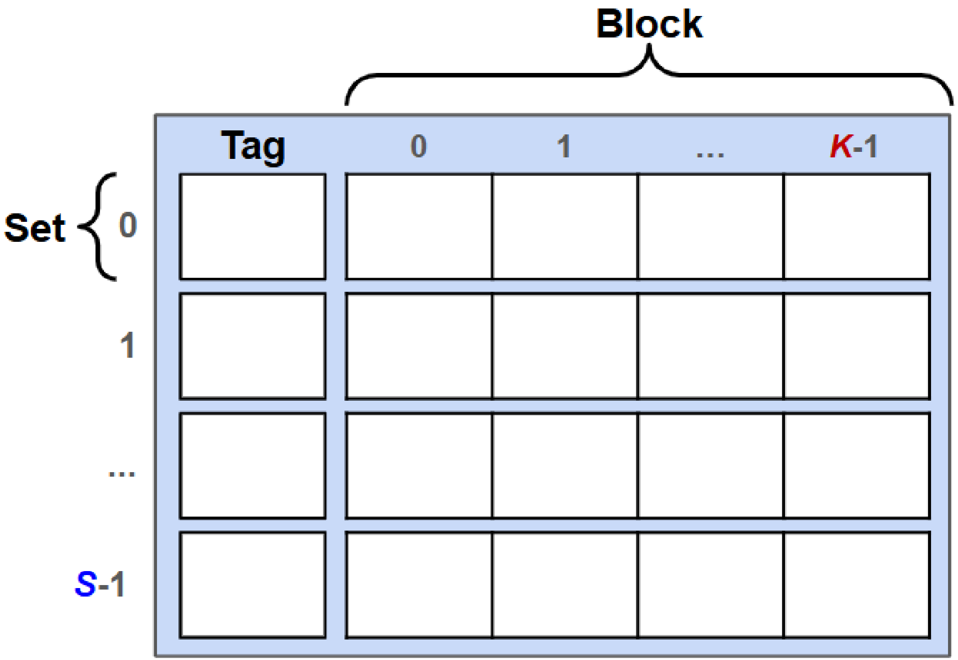
Collisions

- **Problem**: multiple blocks map to the same set
  - Collision occurs when we try to load a block into a set that already has data
    - Evict the old block to make room
  - How can we fix this?
    - *Next lecture!*



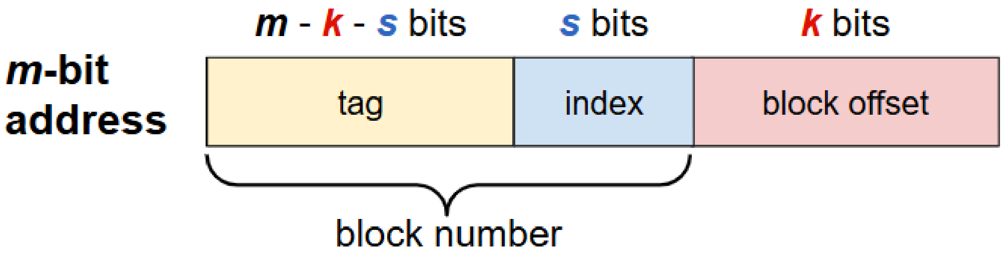
Summary: Cache Terminology

- Memory is broken up into aligned **blocks**
- Cache is broken up into **sets**
  - Each set holds one block (*for now*)
  - Store **tag** along with data block
  - Sets referenced by their **index**
- **Cache size** = number of bytes of data the cache can hold
  - Number of sets * block size



Summary: Address Translation

- Block size = K
  - $k = \log_2(K)$
- Cache size = C
- Number of sets = $S = C \div K$
  - $s = \log_2(S)$



- Divide addresses (a) into fields
  - **Offset** = lowest k bits = a % K
    - Starting location within a block
  - **Index** = next s bits = $(a \div K)$ % S
    - Which set the block is in
  - **Tag** = Remaining bits = $(a \div K) \div S$
    - Used to distinguish different blocks with the same index