UC Berkeley
Teaching Professor
Dan Garcia

CS61C

Great Ideas in Computer Architecture
(a.k.a. Machine Structures)

Caches II

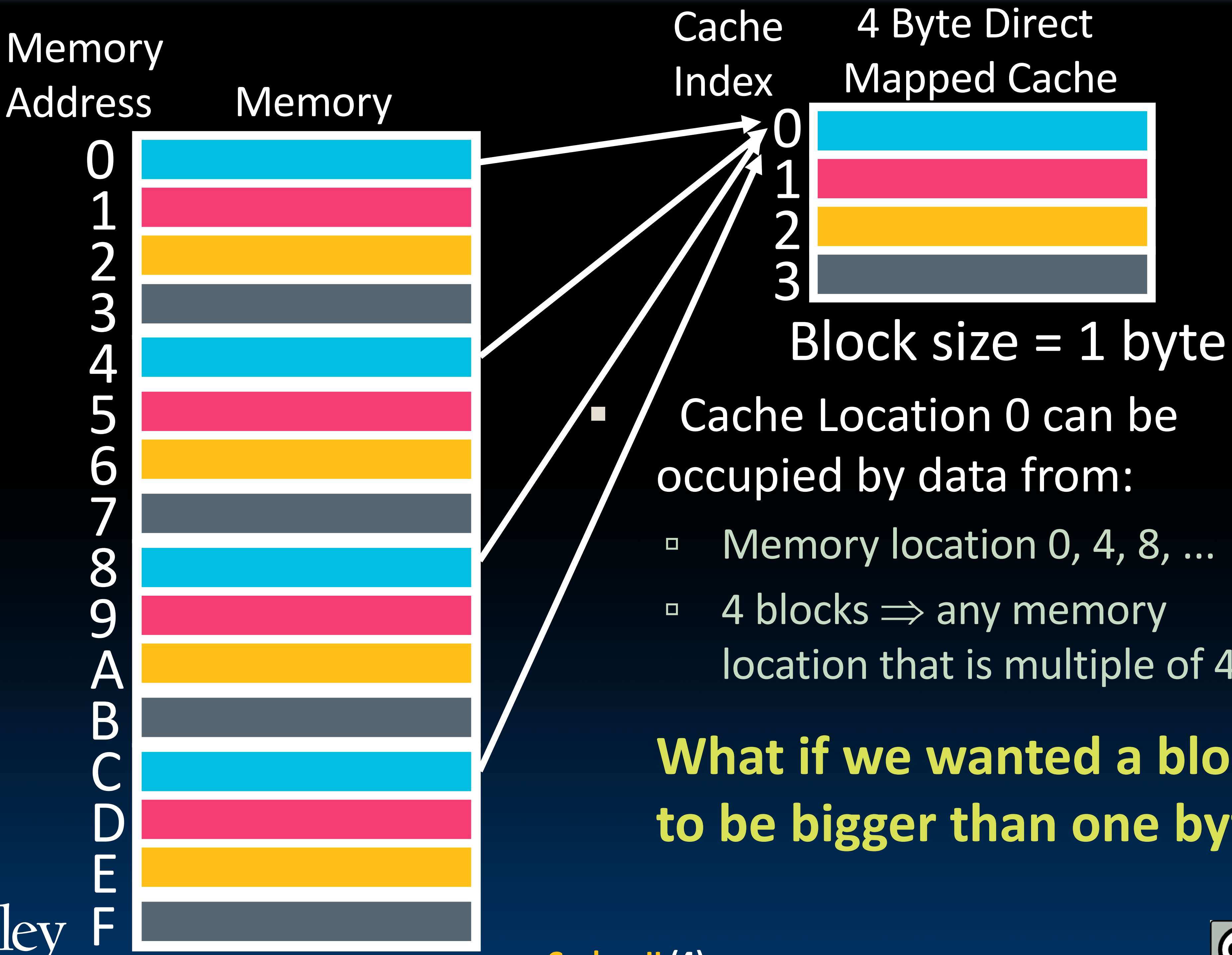
cs61c.org

Direct Mapped Caches

Direct-Mapped Cache (1/4)

- In a direct-mapped cache, each memory address is associated with one possible block within the cache
  - Therefore, we only need to look in a single location in the cache for the data if it exists in the cache
  - Block is the unit of transfer between cache and memory

Direct-Mapped Cache (2/4)



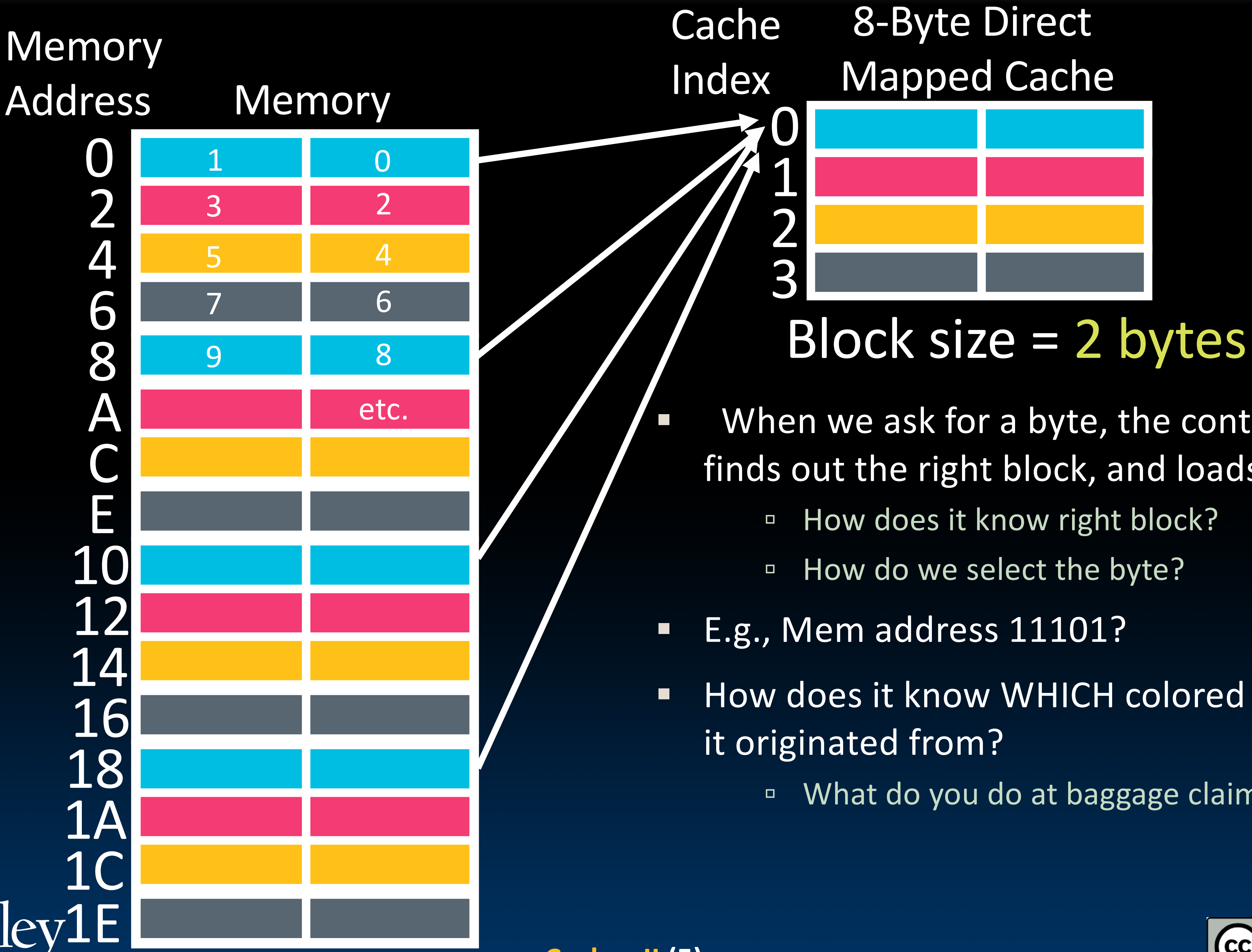
Block size = 1 byte

- Cache Location 0 can be occupied by data from:
  - Memory location 0, 4, 8, ...
  - 4 blocks ⇒ any memory location that is multiple of 4

What if we wanted a block to be bigger than one byte?

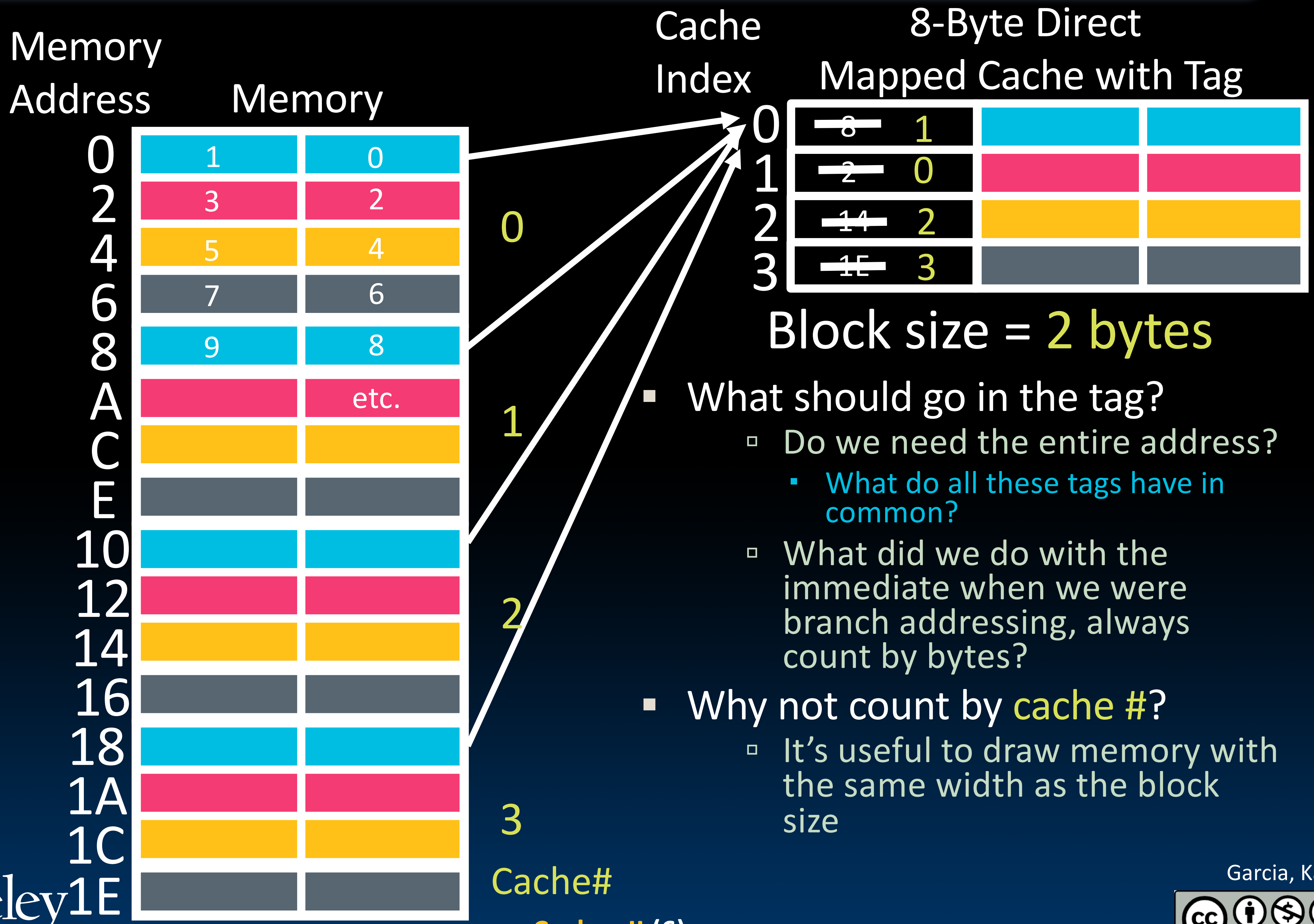
Garcia, Kao

Direct-Mapped Cache (3/4)



- When we ask for a byte, the controller finds out the right block, and loads it all!
  - How does it know right block?
  - How do we select the byte?
- E.g., Mem address 11101?
- How does it know WHICH colored block it originated from?
  - What do you do at baggage claim?

Direct-Mapped Cache (4/4)

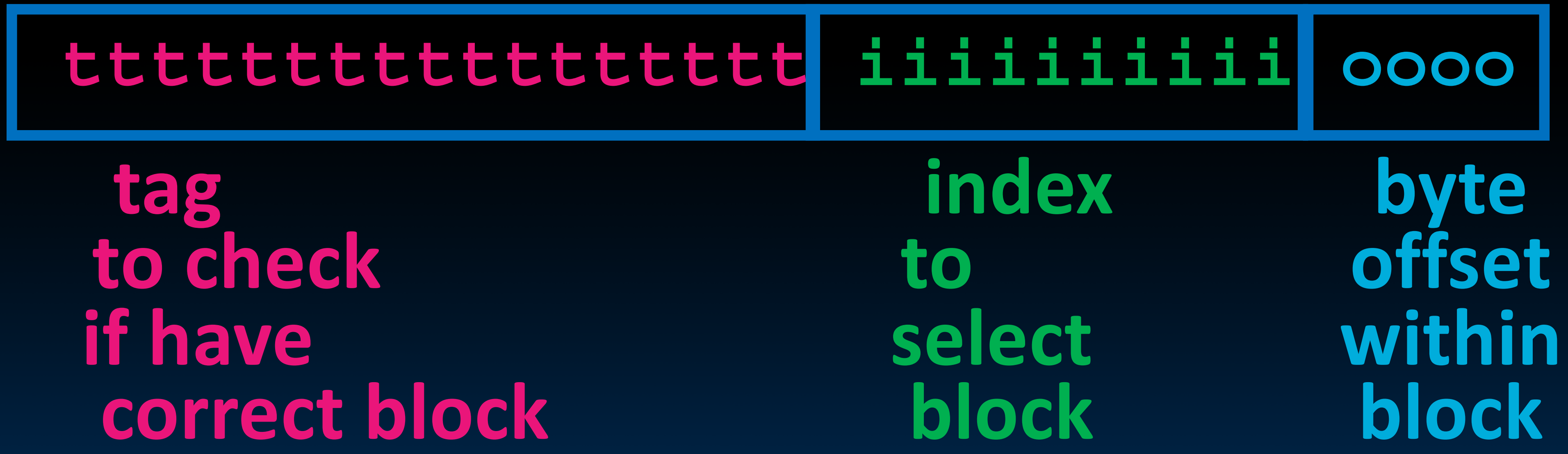


Block size = 2 bytes

- What should go in the tag?
  - Do we need the entire address?
    - What do all these tags have in common?
  - What did we do with the immediate when we were branch addressing, always count by bytes?
- Why not count by cache #?
  - It's useful to draw memory with the same width as the block size

Issues with Direct-Mapped

- Since multiple memory addresses map to same cache index, how do we tell which one is in there?
- What if we have a block size > 1 byte?
- Answer: divide memory address into three fields



Direct-Mapped Cache Terminology

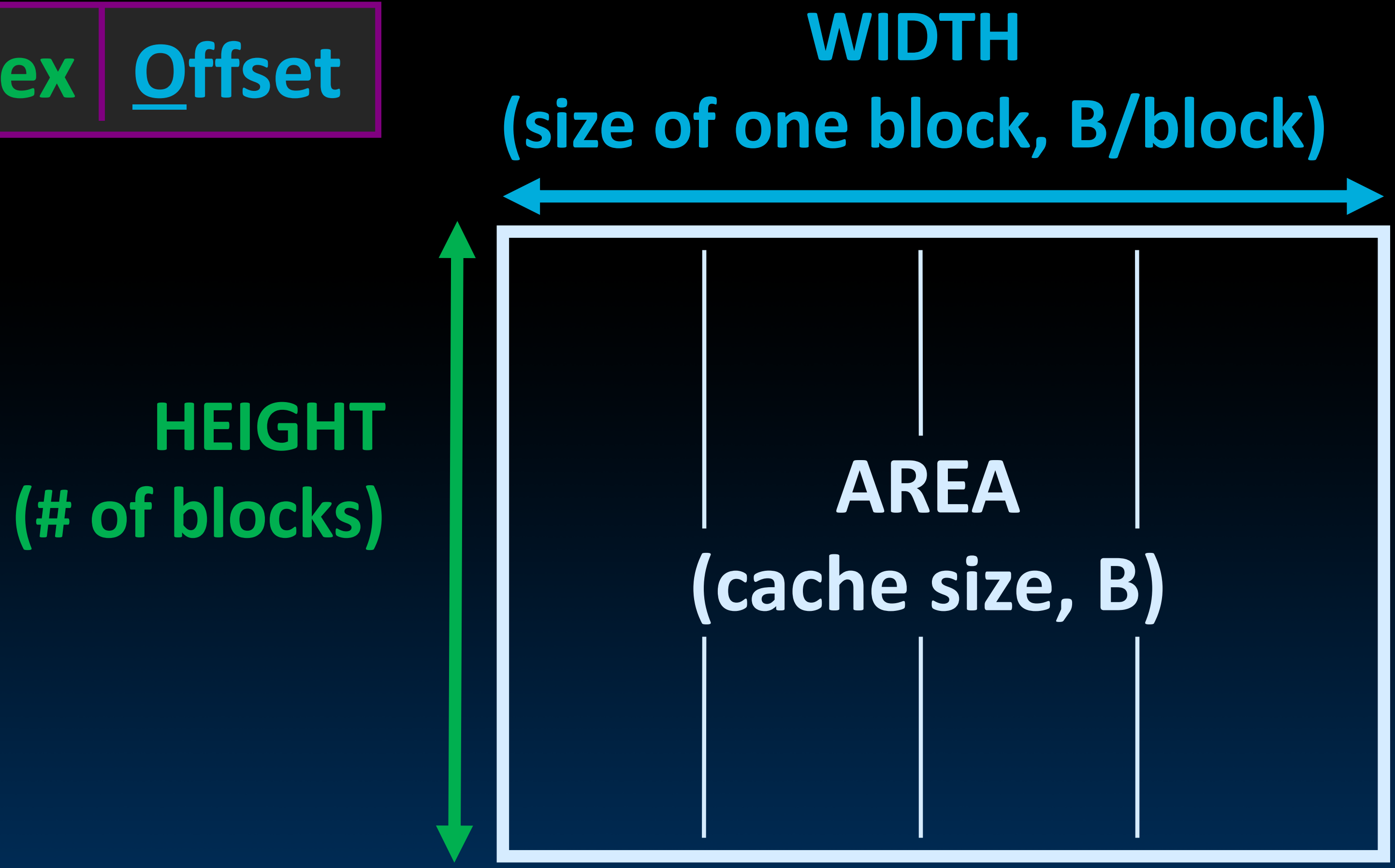
- All fields are read as unsigned integers.
- Index
  - specifies the cache index (which “row”/block of the cache we should look in)
- Offset
  - once we’ve found correct block, specifies which byte within the block we want
- Tag
  - the remaining bits after offset and index are determined; these are used to distinguish between all the memory addresses that map to the same location

TIO Cache Mnemonic (Thanks Uncle Dan!)

AREA (cache size, B)
= HEIGHT (# of blocks)
* WIDTH (size of one block, B/block)

$$2^{(H+W)} = 2^{H} * 2^{W}$$

Tag	Index	Offset



Direct Mapped Example

Direct-Mapped Cache Example (1/3)

- Suppose we have a 8B of data in a direct-mapped cache with 2-byte blocks
  - Sound familiar?
- Determine the size of the tag, index and offset fields if using a 32-bit arch (RV32)
- Offset
  - need to specify correct byte within a block
  - block contains 2 bytes

    = 2^1 bytes
  - need 1 bit to specify correct byte

Direct-Mapped Cache Example (2/3)

- Index: (~index into an "array of blocks")
  - need to specify correct block in cache
  - cache contains 8 B = 2^3 bytes
  - block contains 2 B = 2^1 bytes
  - # blocks/cache

$$= \frac{\text{bytes/cache}}{\text{bytes/block}}$$

$$= \frac{2^3 \text{ bytes/cache}}{2^1 \text{ bytes/block}}$$

$$= 2^2 \text{ blocks/cache}$$

  - need 2 bits to specify this many blocks

Direct-Mapped Cache Example (3/3)

- Tag: use remaining bits as tag
  - tag length = addr length – offset - index
    = 32 - 1 - 2 bits
    = 29 bits
  - so tag is leftmost 29 bits of memory address
  - Tag can be thought of as “cache number”
- Why not full 32-bit address as tag?
  - All bytes within block need same address
  - Index must be same for every address within a block, so it’s redundant in tag check, thus can leave off to save memory

Memory Access without Cache

- Load word instruction: `lw t0, 0(t1)`
- `t1` contains `1022`$_{\text{ten}}$, `Memory[1022] = 99`

1. Processor issues address `1022`$_{\text{ten}}$ to **`Memory`**
2. Memory reads word at address `1022`$_{\text{ten}}$ `(99)`
3. Memory sends `99` to Processor
4. Processor loads `99` into register `t0`

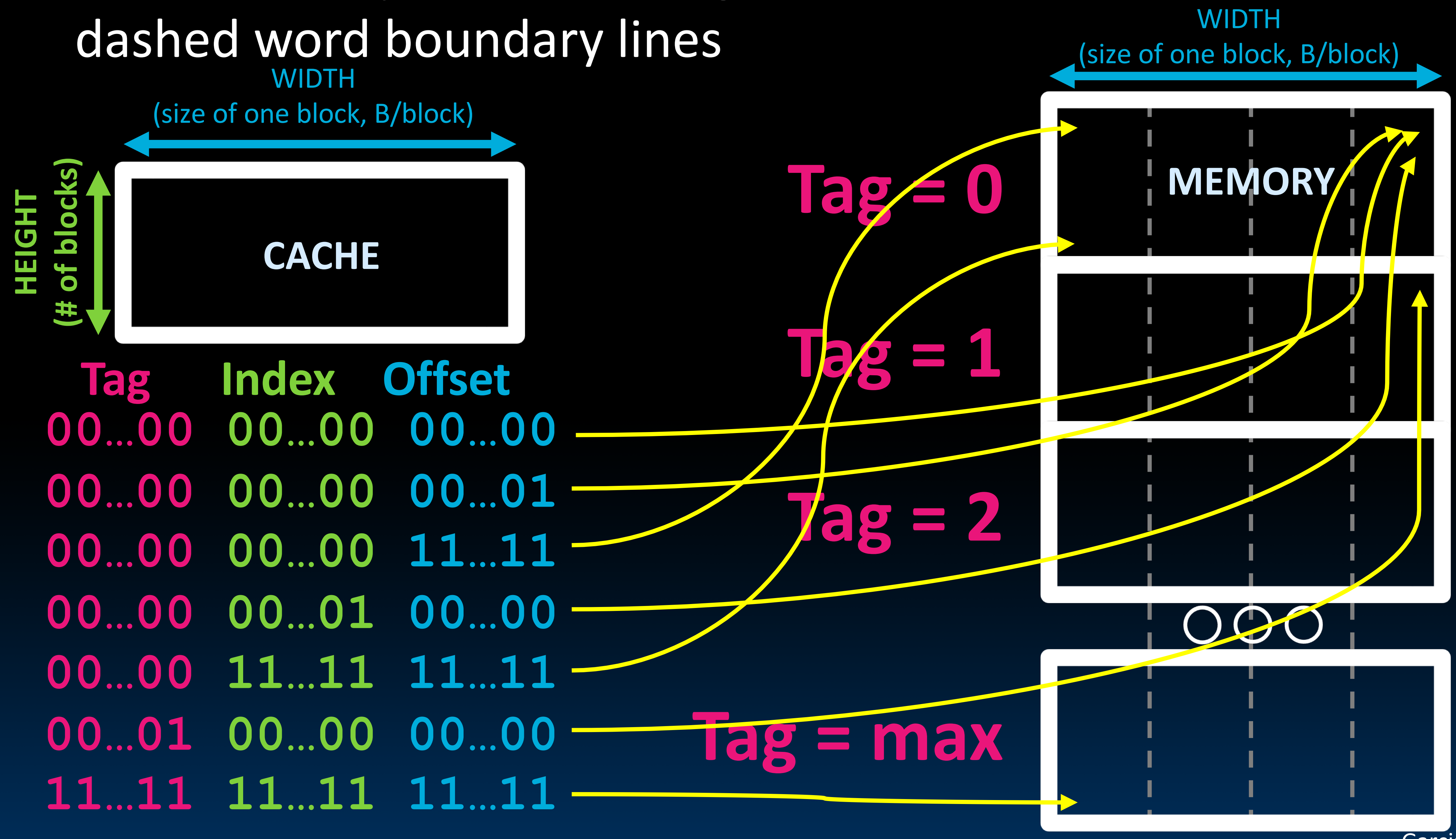
Memory Access with Cache

- Load word instruction: `lw t0, 0(t1)`
- `t1` contains `1022`$_{ten}$, `Memory[1022] = 99`
- With cache (similar to a hash)
  1. Processor issues address `1022`$_{ten}$ to Cache
  2. Cache checks to see if has copy of data at address `1022`$_{ten}$
     - 2a. If finds a match (Hit): cache reads `99`, sends to processor
     - 2b. No match (Miss): cache sends address `1022` to Memory
       - I. Memory reads `99` at address `1022`$_{ten}$
       - II. Memory sends `99` to Cache
       - III. Cache replaces word with new `99`
       - IV. Cache sends `99` to processor
  3. Processor loads `99` into register `t0`

Solving Cache problems

Tag Index Offset

- Draw memory a block wide given T I O bits, dashed word boundary lines



Cache Terminology

Caching Terminology

- When reading memory, 3 things can happen:
  - cache hit:
    cache block is valid and contains proper address, so read desired word
  - cache miss:
    nothing in cache in appropriate block, so fetch from memory
  - cache miss, block replacement:
    wrong data is in cache at appropriate block, so discard it and fetch desired data from memory (cache always copy)

Cache Temperatures

- Cold
  - Cache empty
- Warming
  - Cache filling with values you'll hopefully be accessing again soon
- Warm
  - Cache is doing its job, fair % of hits
- Hot
  - Cache is doing very well, high % of hits

Cache Terms

- Hit rate: fraction of access that hit in the cache
- Miss rate: 1 – Hit rate
- Miss penalty: time to replace a block from lower level in memory hierarchy to cache
- Hit time: time to access cache memory (including tag comparison)

One More Detail: Valid Bit

- When start a new program, cache does not have valid information for this program
- Need an indicator whether this tag entry is valid for this program
- Add a “valid bit” to the cache tag entry
  - 0 → cache miss, even if by chance, address = tag
  - 1 → cache hit, if processor address = tag

Example: 16 KB Direct-Mapped Cache, 16B blocks

- Valid bit: determines whether anything is stored in that row (when computer initially powered up, all entries invalid)

Index	Valid	Tag	0xc-f	0x8-b	0x4-7	0x0-3
0	0					
1	0					
2	0					
3	0					
4	0					
5	0					
6	0					
7	0					
...				...		
1022	0					
1023	0					

Looks like a real cache, will investigate it some more!

"And in Conclusion…"

- We have learned the operation of a direct-mapped cache
- Mechanism for transparent movement of data among levels of a memory hierarchy
  - set of address/value bindings
  - address → index to set of candidates
  - compare desired address with tag
  - service hit or miss
    - load new block and binding on miss

☑