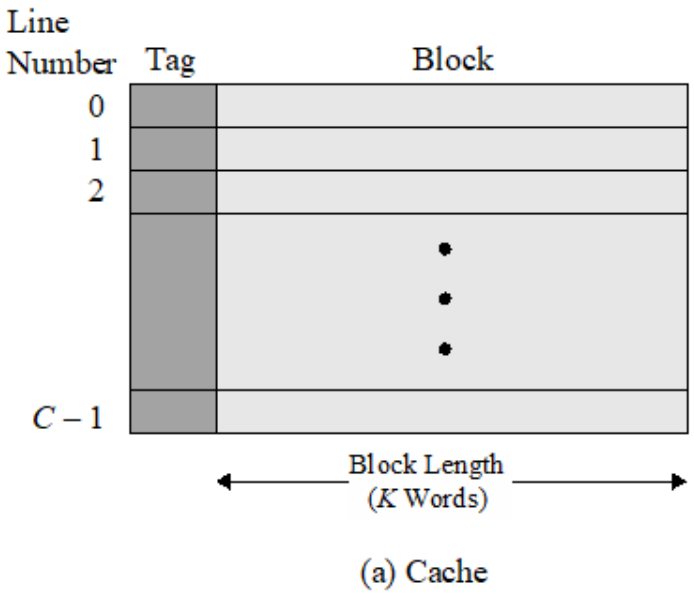
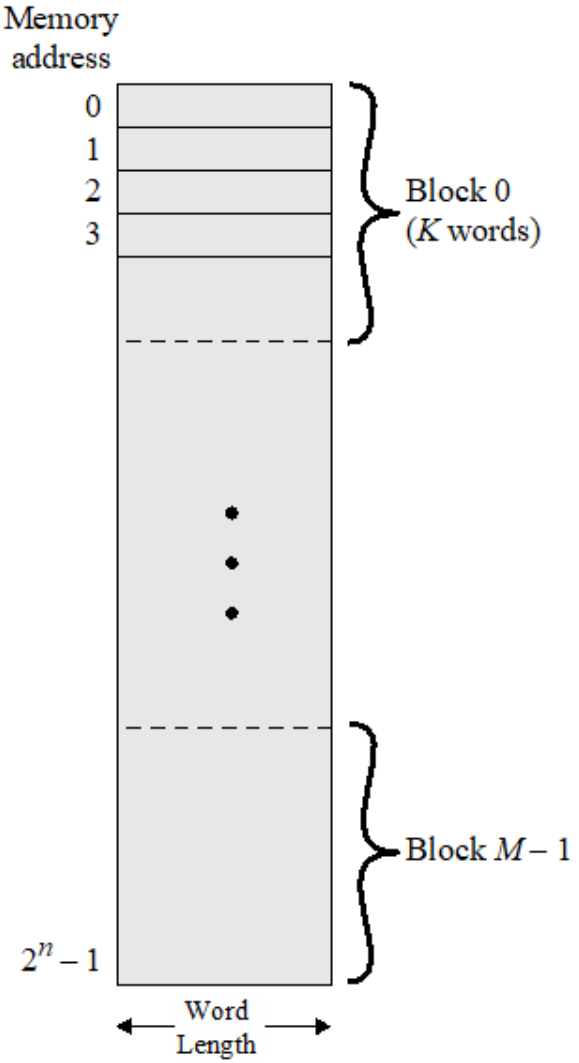
Cache and Main Memory Structure

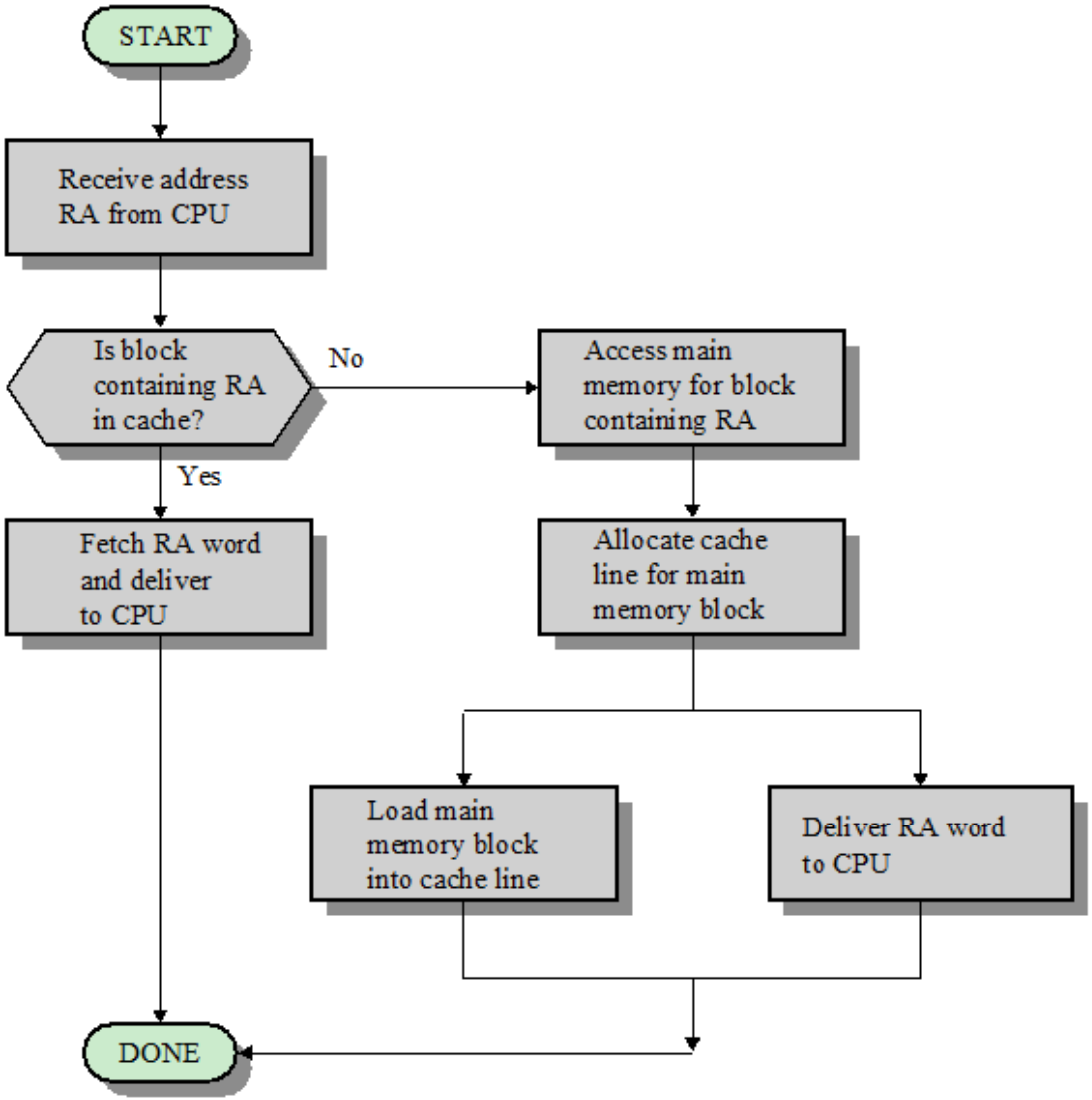


(a) Cache

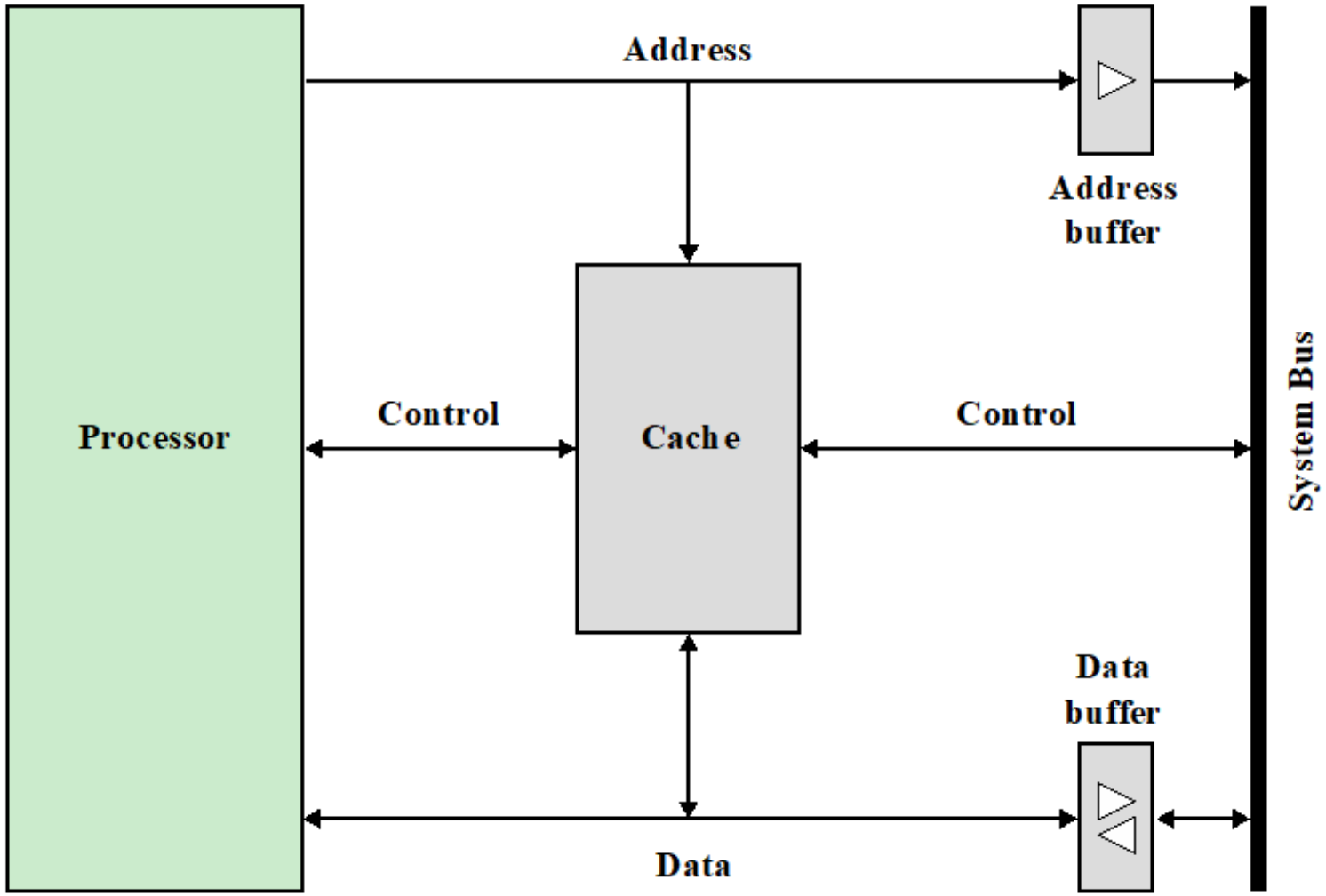


(b) Main memory

Cache Read Operation



Typical Cache Organization



Elements of Cache Design

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set Associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back

Line Size

Number of caches
- Single or two level
- Unified or split

What is a Cache?

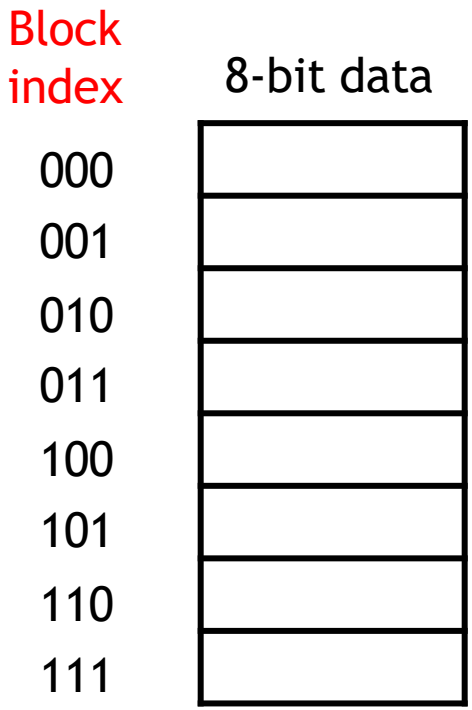
- A cache allows for fast accesses to a subset of a larger data store
- Your web browser's cache gives you fast access to pages you visited recently
  - faster because it's stored locally
  - subset because the web won't fit on your disk
- The memory cache gives the processor fast access to memory that it used recently
  - faster because it's usually located on the CPU chip
  - subset because the cache is smaller than main memory

Cache Contents?

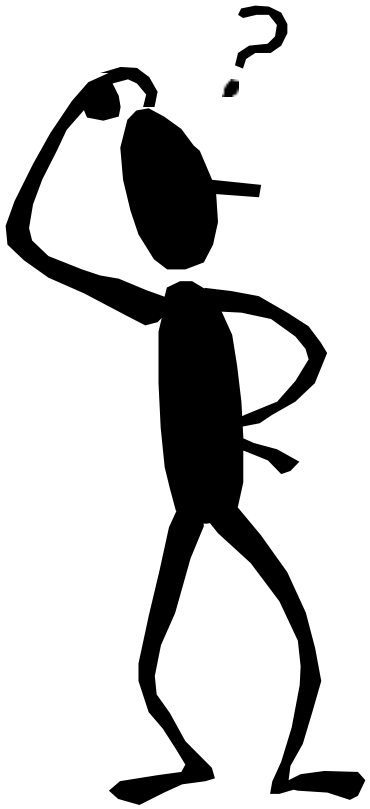
- When do we put something in the cache?
  - When it is used for the first time
- When do we overwrite something in the cache?
  - When we need the space in the cache for some other entry
  - All of memory won't fit on the CPU chip so not every location in memory can be cached

A Simple Cache Design

- Caches are divided into blocks, which may be of various sizes.
  - The number of blocks in a cache is usually a power of 2.
  - For now we'll say that each block contains one byte. This won't take advantage of spatial locality, but we'll do that next time.
- Here is an example cache with eight blocks, each holding one byte.



Important Questions

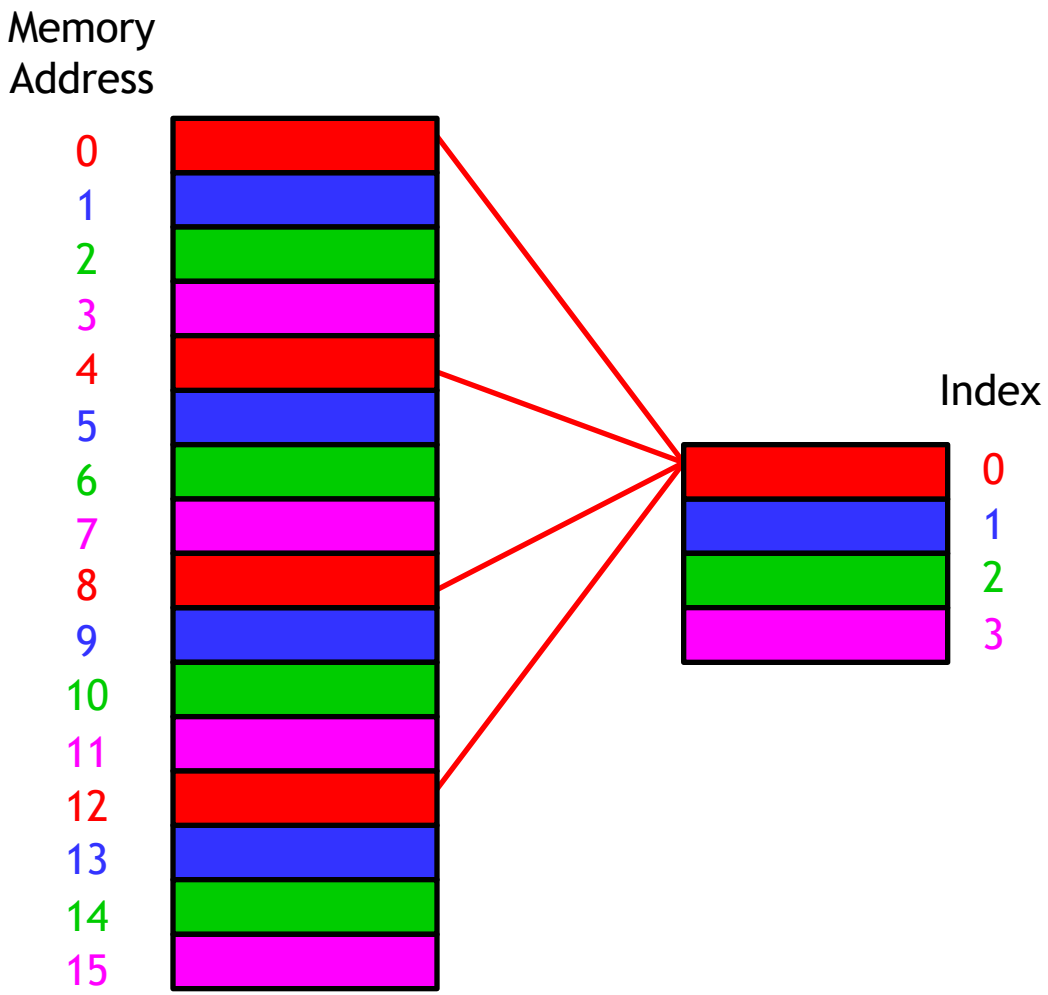


1. When we copy a block of data from main memory to the cache, where exactly should we put it?
2. How can we tell if a word is already in the cache, or if it has to be fetched from main memory first?
3. Eventually, the small cache memory might fill up. To load a new block from main RAM, we'd have to replace one of the existing blocks in the cache... which one?
4. How can *write* operations be handled by the memory system?

- Questions 1 and 2 are related—we have to know where the data is placed if we ever hope to find it again later!

Where Should we put Data in Cache

- A direct-mapped cache is the simplest approach: each main memory address maps to exactly one cache block.
- For example, on the right is a 16-byte main memory and a 4-byte cache (four 1-byte blocks).
- Memory locations 0, 4, 8 and 12 all map to cache block 0.
- Addresses 1, 5, 9 and 13 map to cache block 1, etc.
- How can we compute this mapping?



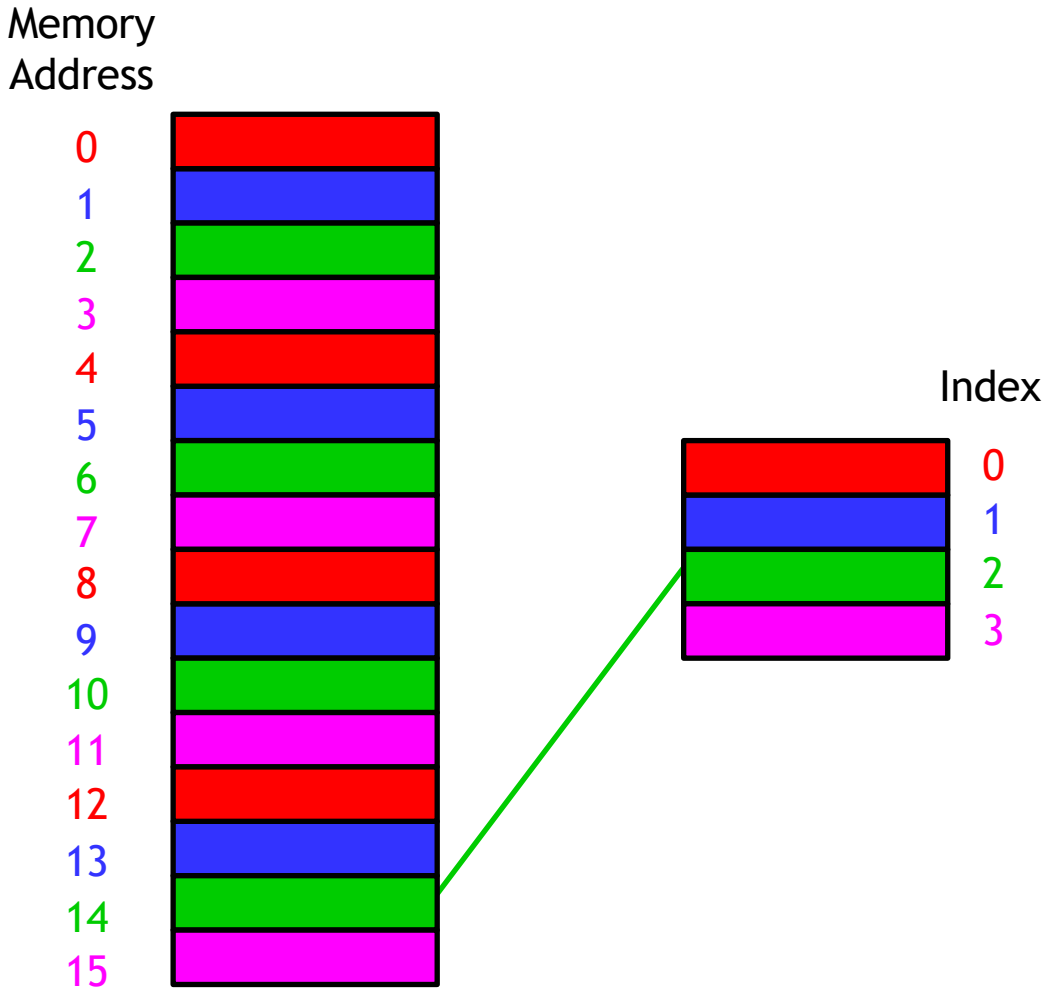
It's all Divisions

- One way to figure out which cache block a particular memory address should go to is to use the mod (remainder) operator.
- If the cache contains 2^k blocks, then the data at memory address *i* would go to cache block index

  $i \bmod 2^k$

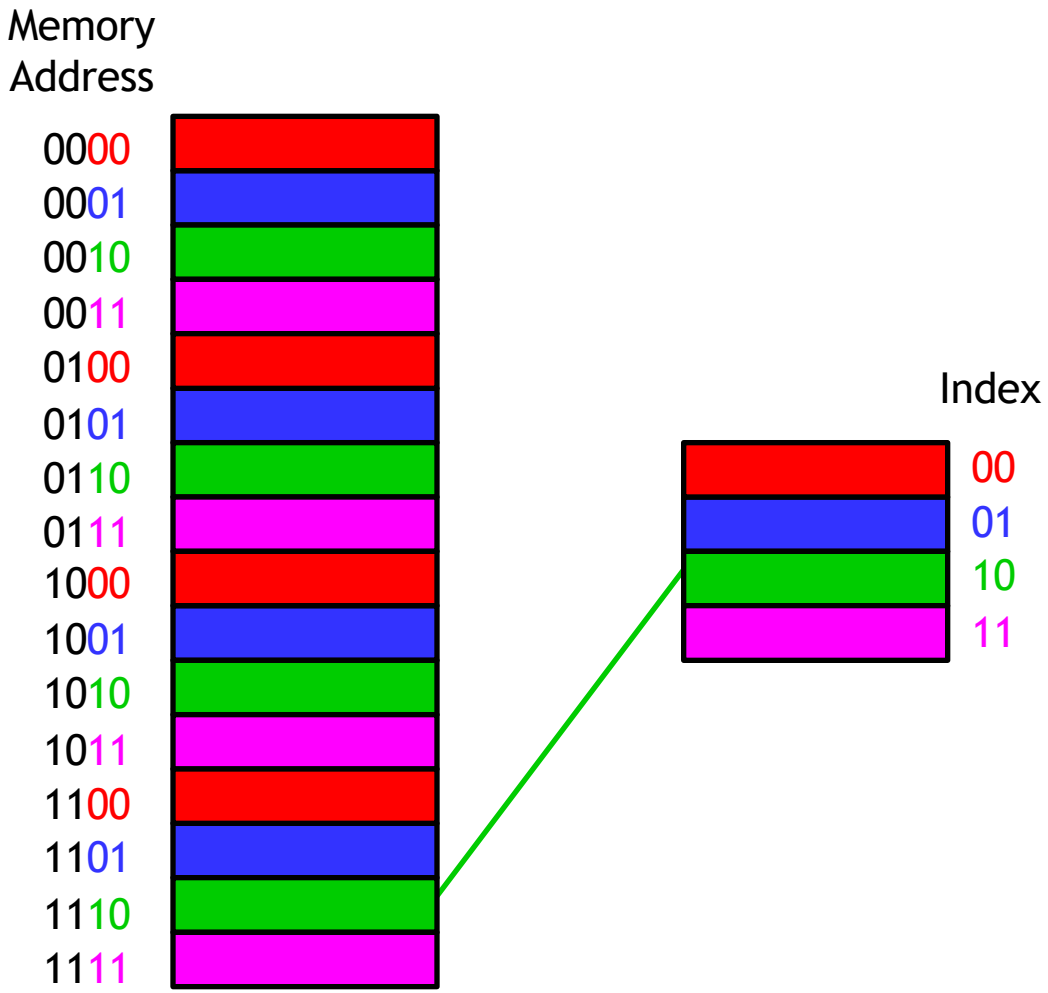
- For instance, with the four-block cache here, address 14 would map to cache block 2.

  $14 \bmod 4 = 2$



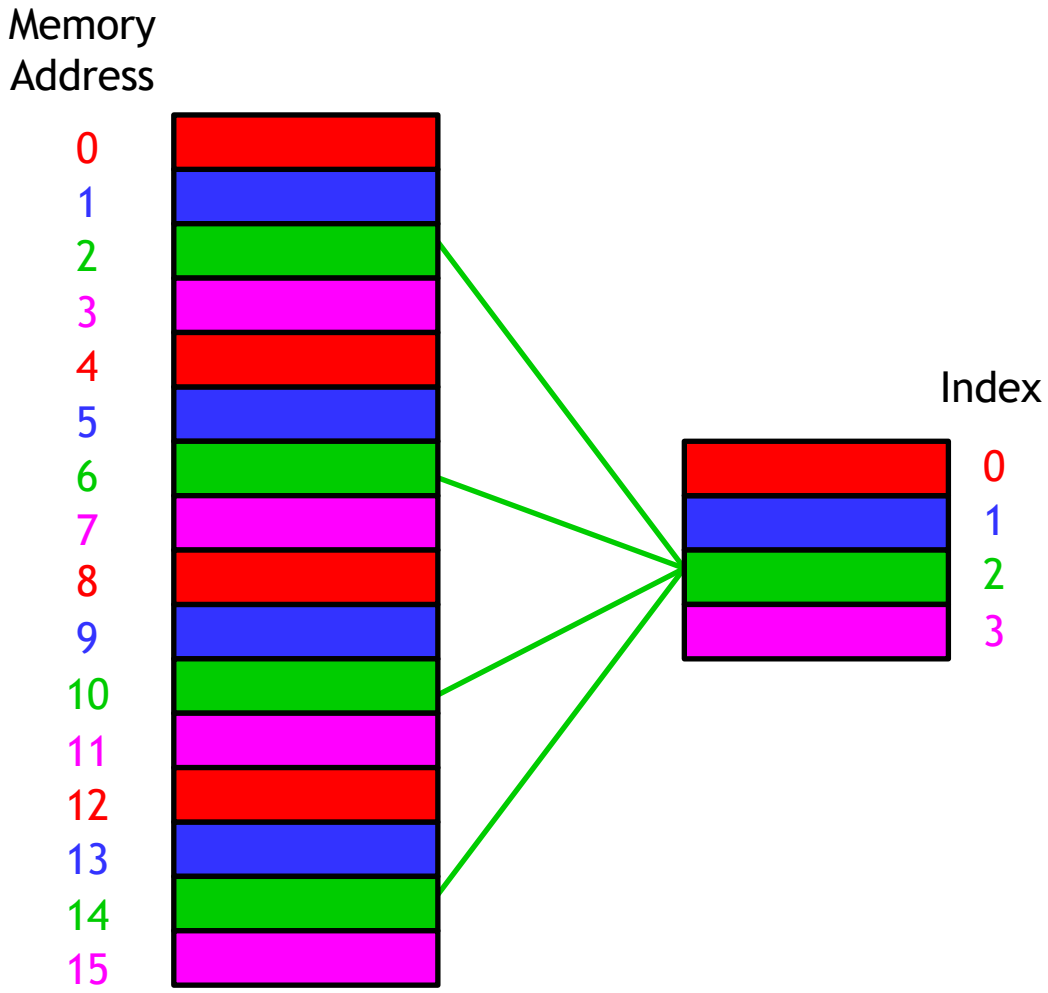
……or Least Significant Bits

- An equivalent way to find the placement of a memory address in the cache is to look at the least significant *k* bits of the address.
- With our four-byte cache we would inspect the two least significant bits of our memory addresses.
- Again, you can see that address 14 (1110 in binary) maps to cache block 2 (10 in binary).
- Taking the least *k* bits of a binary value is the same as computing that value mod 2^k.



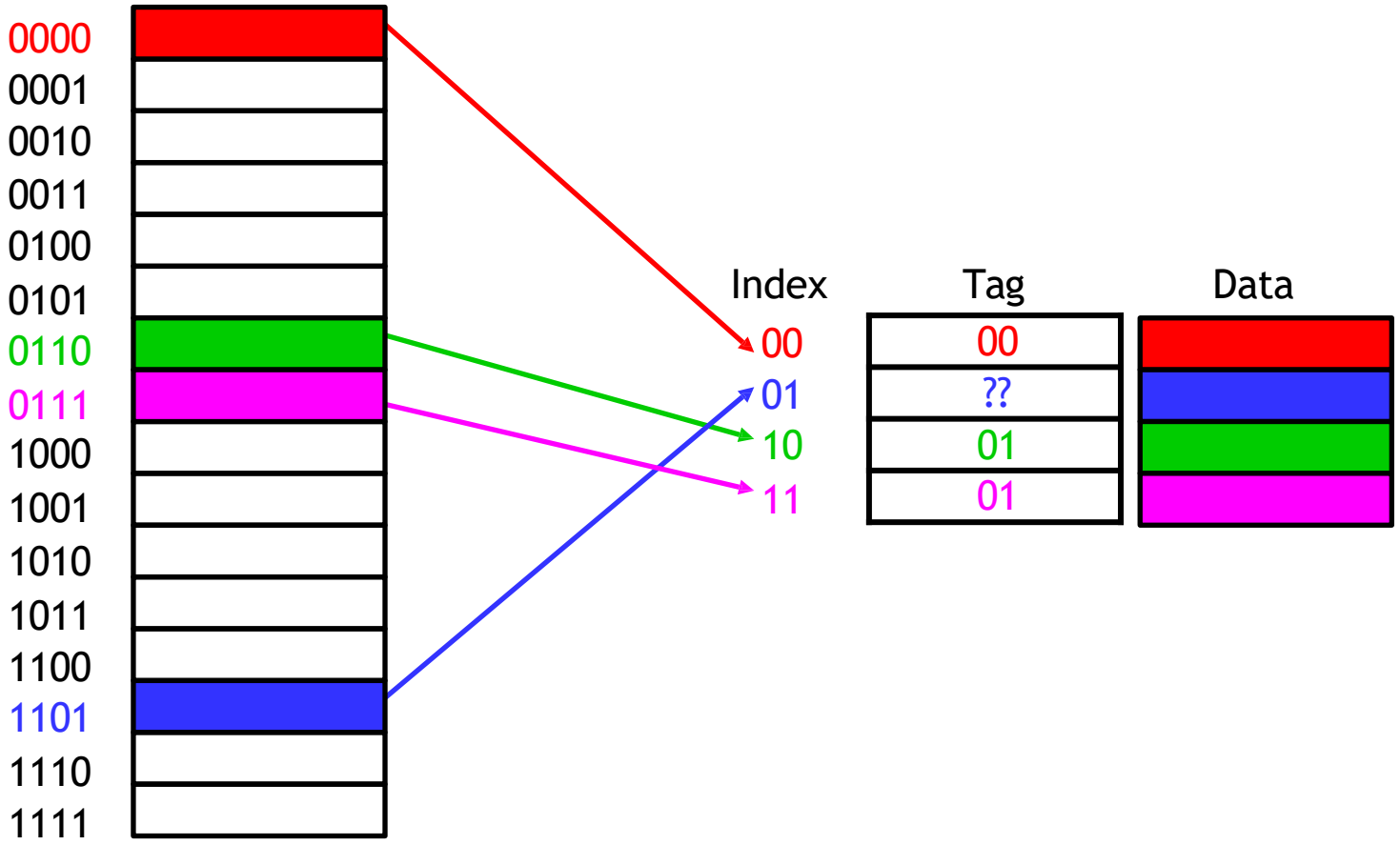
How can we find Data in Cache?

- The second question was how to determine whether or not the data we're interested in is already stored in the cache.
- If we want to read memory address *i*, we can use the mod trick to determine which cache block would contain *i*.
- But other addresses might *also* map to the same cache block. How can we distinguish between them?
- For instance, cache block 2 could contain data from addresses 2, 6, 10 *or* 14.



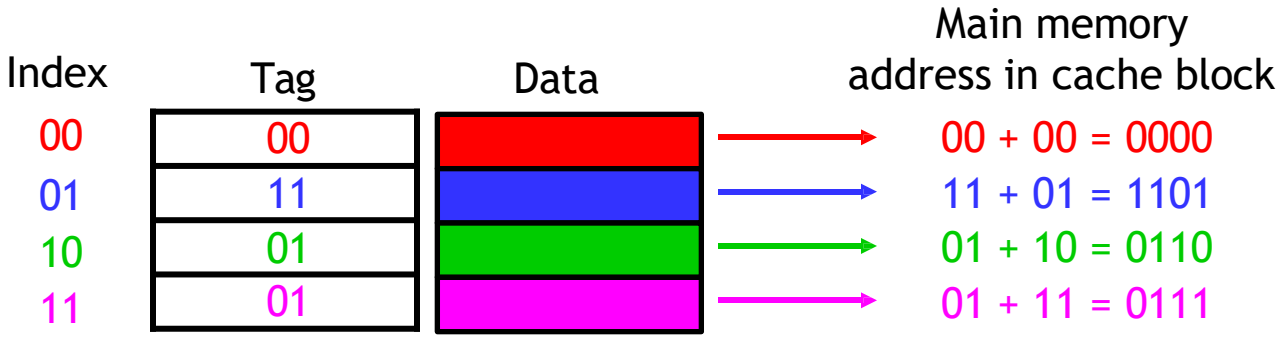
Adding Tags

- We need to add tags to the cache, which supply the rest of the address bits to let us distinguish between different memory locations that map to the same cache block.



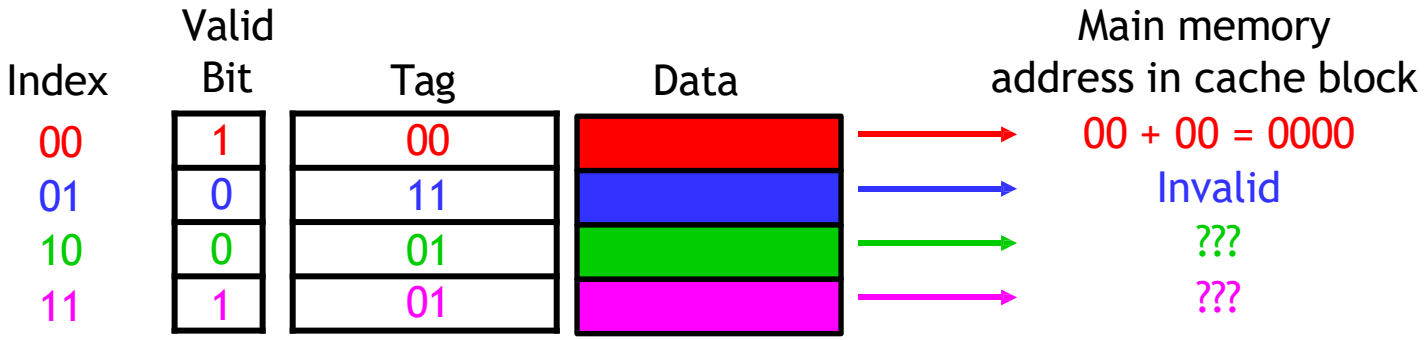
Figuring out what's Inside the Cache

- Now we can tell exactly which addresses of main memory are stored in the cache, by concatenating the cache block tags with the block indices.



One more detail: the Valid Bit

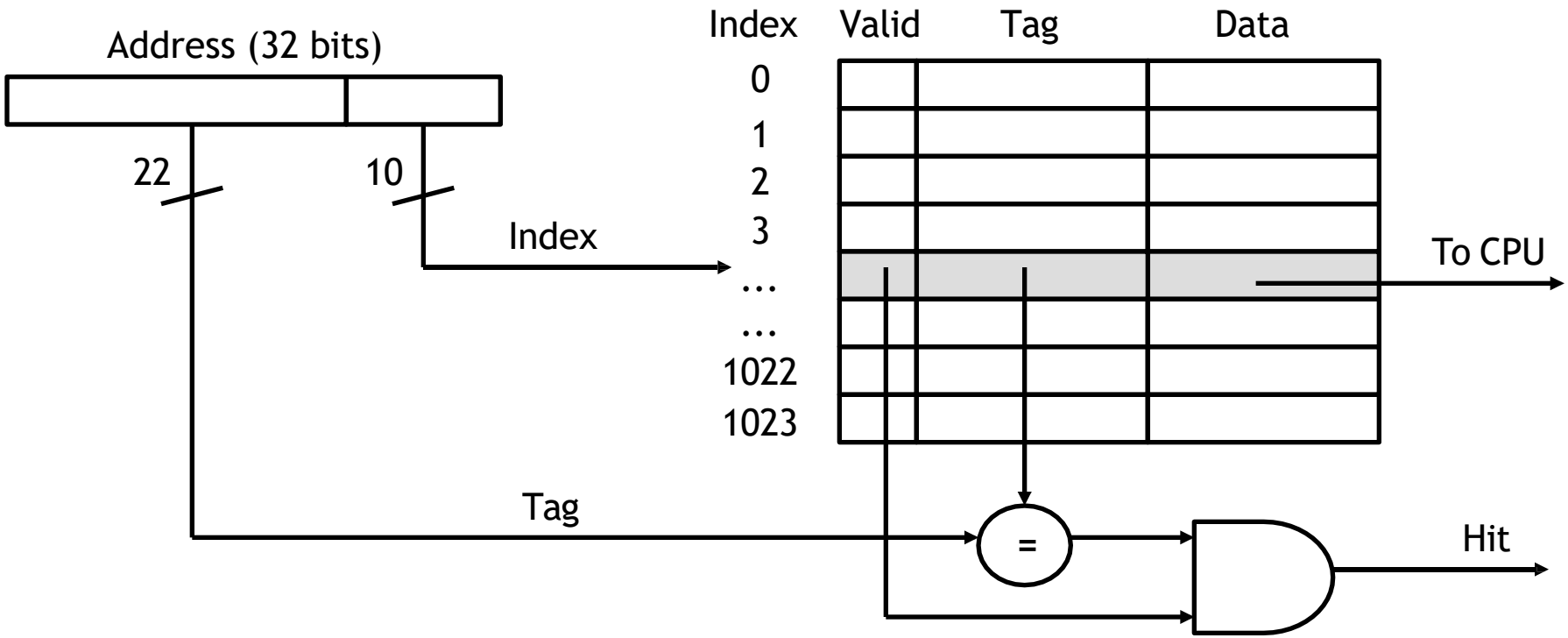
- When started, the cache is empty and does not contain valid data.
- We should account for this by adding a valid bit for each cache block.
  - When the system is initialized, all the valid bits are set to 0.
  - When data is loaded into a particular cache block, the corresponding valid bit is set to 1.



- So, the cache contains more than just copies of the data in memory; it also has bits to help us find data within the cache and verify its validity.

What Happens on a Cache Hit

- When the CPU tries to read from memory, the address will be sent to a cache controller.
  - The lowest *k* bits of the address will index a block in the cache.
  - If the block is valid and the tag matches the upper (*m* – *k*) bits of the *m*-bit address, then that data will be sent to the CPU.
- Here is a diagram of a 32-bit memory address and a 2^{10}-byte cache.

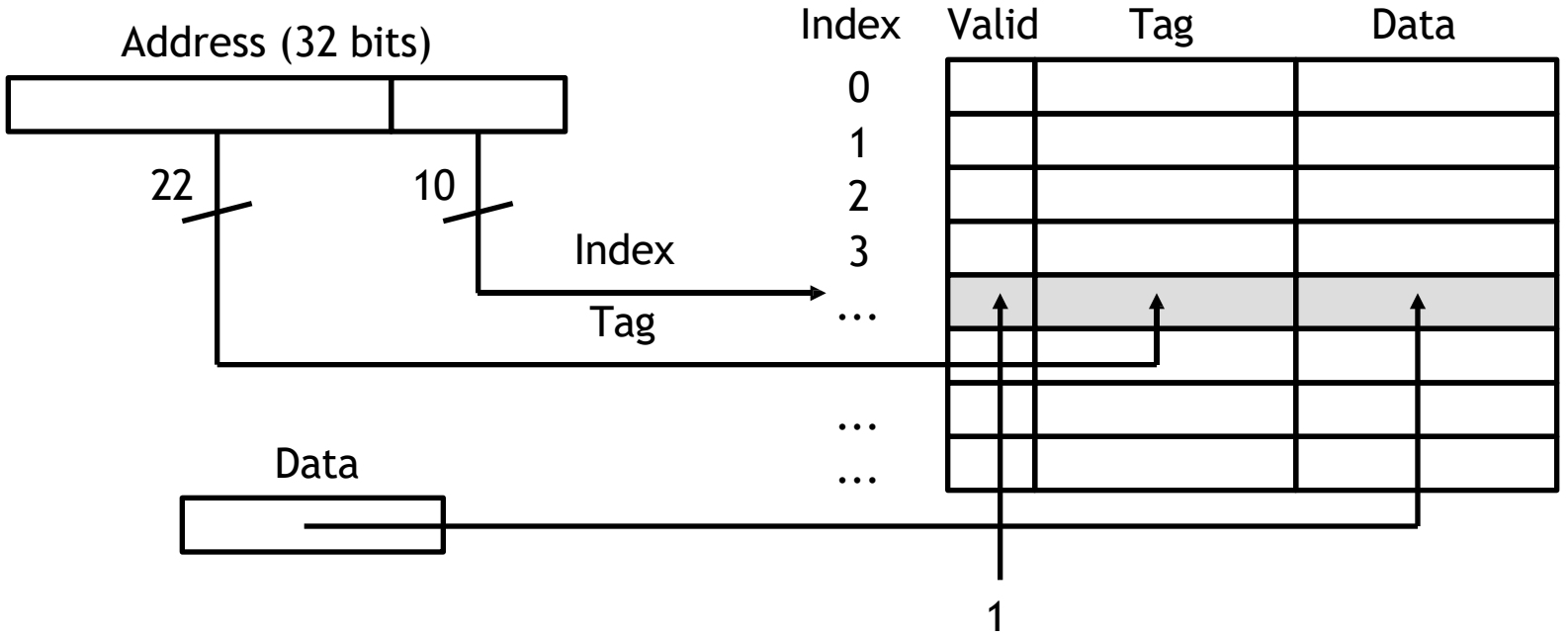


What Happens on a Cache Miss

- The delays that we've been assuming for memories (e.g., 2ns) are really assuming cache hits.
  - If our CPU implementations accessed main memory directly, their cycle times would have to be much larger.
  - Instead we assume that most memory accesses will be cache hits, which allows us to use a shorter cycle time.
- However, a much slower main memory access is needed on a cache miss. The simplest thing to do is to stall the pipeline until the data from main memory can be fetched (and also copied into the cache).

Loading a Block in the Cache

- After data is read from main memory, putting a copy of that data into the cache is straightforward.
  - The lowest *k* bits of the address specify a cache block.
  - The upper (*m* – *k*) address bits are stored in the block's tag field.
  - The data from main memory is stored in the block's data field.
  - The valid bit is set to 1.



What if the Cache Fills-up

- Our third question was what to do if we run out of space in our cache, or if we need to reuse a block for a different memory address.
- We answered this question implicitly on the last page!
  - A miss causes a new block to be loaded into the cache, automatically overwriting any previously stored data.
  - This is a least recently used replacement policy, which assumes that older data is less likely to be requested than newer data.
- We'll see a few other policies next.

More Cache Organizations?

Now we'll explore some alternate cache organizations.

- How can we take advantage of spatial locality too?
- How can we take advantage of temporal locality?
- How can we reduce the number of potential conflicts?

Principle of Locality of Reference

- Temporal locality – nearness in time
  - Data being accessed now will probably be accessed again soon
  - Useful data tends to continue to be useful

- Spatial locality – nearness in address
  - Data near the data being accessed now will probably be needed soon
  - Useful data is often accessed sequentially
  - Memory accesses speed up by 9% annually
  - It's becoming harder and harder to keep these processors fed