Lecture 15: Caches (2) - Design

(CPEG323: Intro. to Computer System Engineering)

Cache design

Single-core
Two-level cache

Dual-core
Three-level cache

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.

Block index	8-bit data
000	
001	
010	
011	
100	
101	
110	
111	

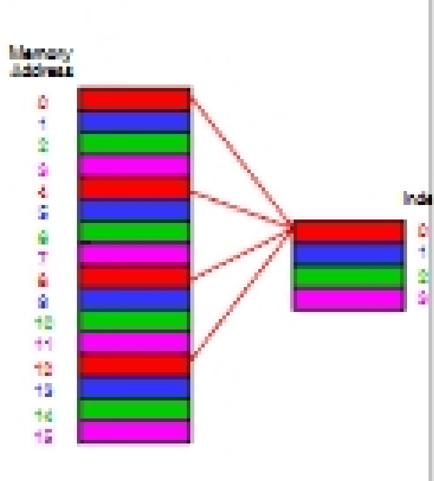
Four important questions

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!

Direct-mapped cache

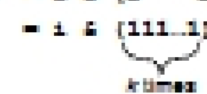
- Each main memory block is mapped to exactly one block in the cache.
  - For example, on the right is a 16-byte main memory and a 4-byte cache (four 1-byte blocks).
- Many lower level blocks share a given cache block.
  - E.g., Memory locations 0, 4, 8 and 12 all map to cache block 0.
- How can we compute this mapping?



Direct-mapped cache

- Think of memory as a long array `M[]` of bytes, and the cache as a short array `C[]` of bytes
- Where should we place `M[i]` in the cache?
  - A natural choice: `C[i % sizeof(C)]`
- If `sizeof(C)` `(i.e.,bytes)` = 2^k (a power of two):

```
i % 2^k = i & (2^k - 1)
        = i & (111..1)
```



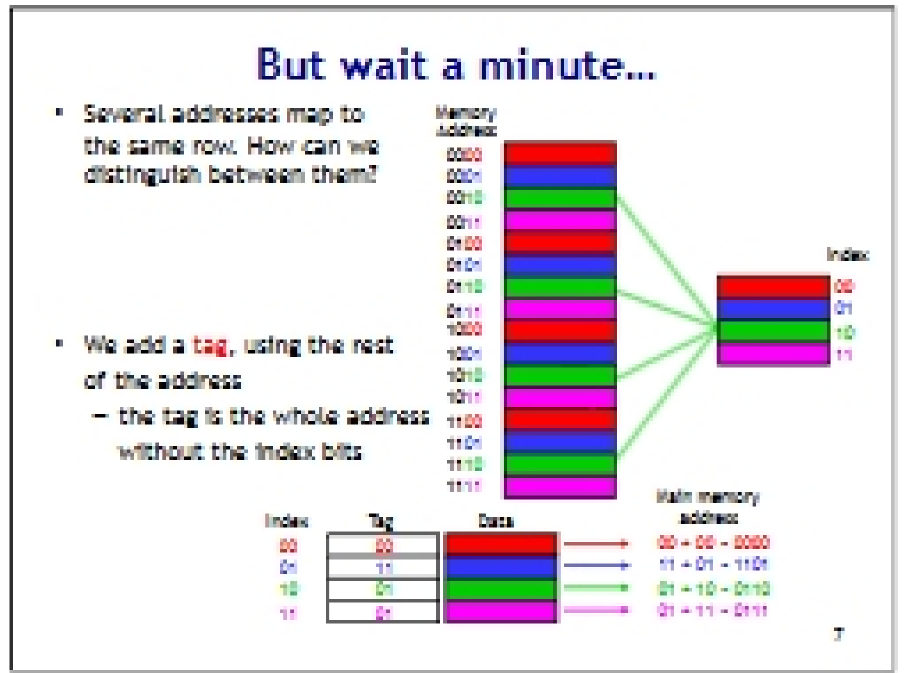
M C

- Thus, C-index = k least significant bits of M-index
  = k least significant bits of address
- In a direct-mapped cache, the address directly tells us the location of the data in the cache

6

But wait a minute…

- Several addresses map to the same row. How can we distinguish between them?



- We add a tag, using the rest of the address
  - the tag is the whole address without the index bits

7

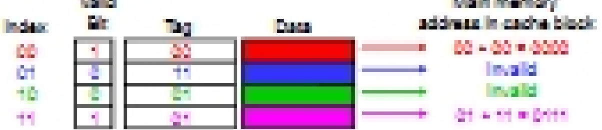
One more detail: the valid bit

- 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.

One more detail: the valid bit

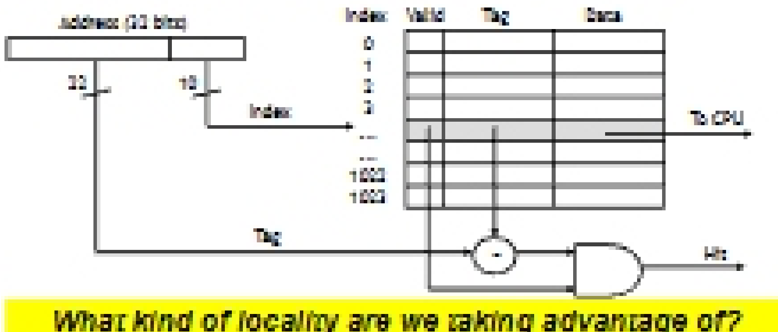
- When started, the cache is empty and does not contain valid data.
- We should account for this by adding a valid bit for each cache block.
  - When the system is initialized, all the valid bits are set to 0.
  - When data is loaded into a particular cache block, the corresponding valid bit is set to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



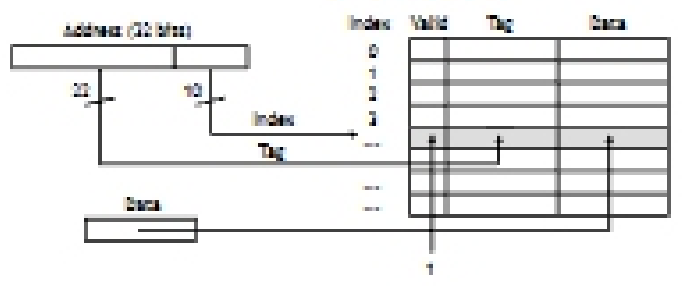
What kind of locality are we taking advantage of?

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 byte into 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 the cache index
  - The upper $(m - k)$ address bits are stored in the tag field
  - The data from main memory is stored in the data field
  - The valid bit is set to 1 ← at boot-up, the tag and data will be garbage and we don't want an "accidental" hit



12

Direct Mapped Cache

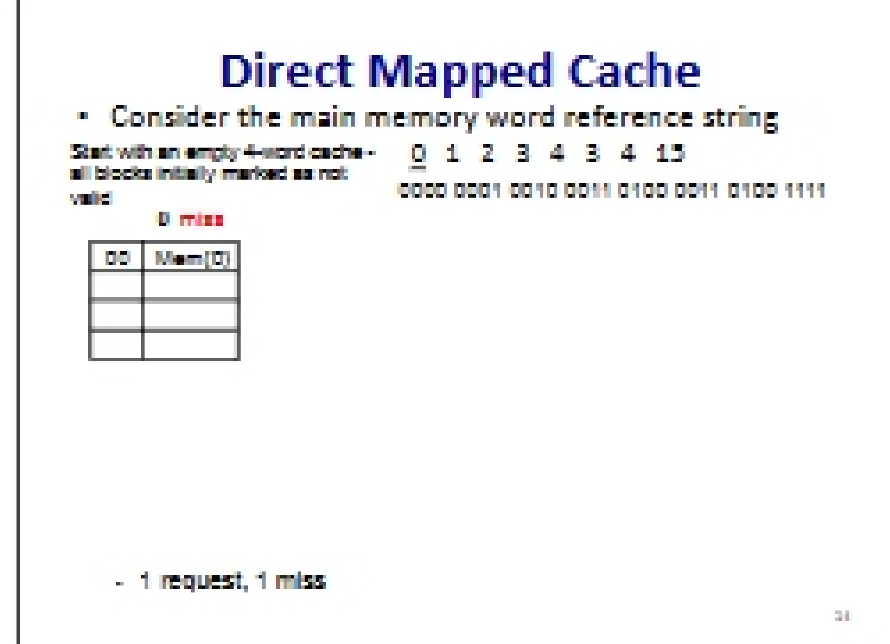
- Consider the main memory word reference string

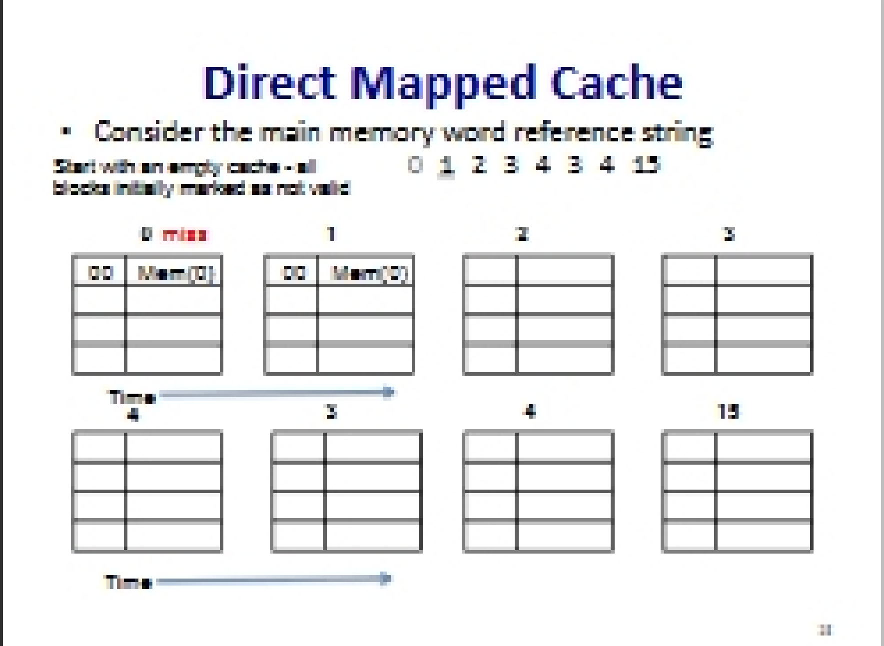
Start with an empty cache - all blocks initially marked as not valid

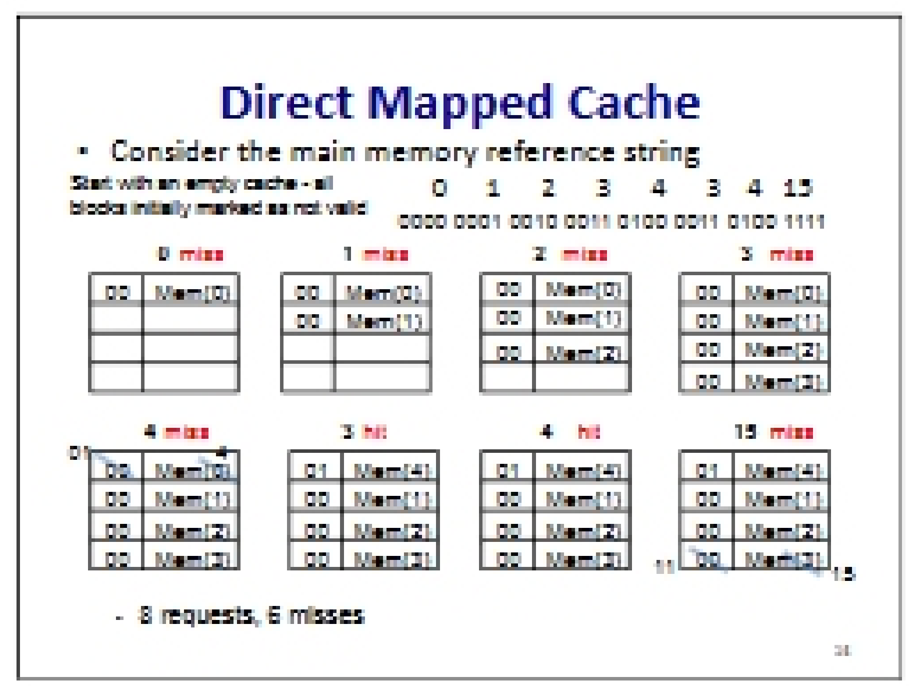
0 1 2 3 4 3 4 15

0000 0001 0010 0011 0100 0011 0100 1111

Address 0







How big is the cache?

Byte-addressable machine, 16-bit addresses, cache details:

- direct-mapped
- block size = one byte
- index = 5 least significant bits

Two questions:

- How many blocks does the cache hold?
- How many bits of storage are required to build the cache (data plus *all overhead* including tags, etc.)?

17

How big is the cache?

For a byte-addressable machine with 16-bit addresses with a cache with the following characteristics:

- It is direct-mapped (as discussed last time)
- Each block holds one byte
- The cache index is the four least significant bits

Two questions:

- How many blocks does the cache hold?
- How many bits of storage are required to build the cache (*e.g.*, for the data array, tags, etc.)?