Today

- Learning Objectives
  - Define
    - Fully Associative Cache
    - Set-Associative Cache
  - Evaluate tradeoffs between different cache organizations
- How we'll get there
  - Motivate caches that are more complex than direct mapped
  - Show what information you need to have a set-associative cache
- Reading
  - 6.4 (We will get to write-caching later.)

Recall:

- In a direct mapped cache, every bit of data has one and only one place it can be in the cache.
- Consider the following:
  - Let's assume we have an 8 KB cache with a line size of 64 bytes.
  - Assume a long is 8 bytes
  - Consider the code to the right.

```
long a[1024];
long b[1024];
long c[1024];

for (int i = 0; i < 1024; i++) {
      c[i] = a[i] + b[i];
}
```

How large is each array?

8192 bytes

Assuming they are allocated contiguously, what is going to happen when we run this program?

Every single array access is going to cause a miss!

The problem with a direct mapped cache

- If array A starts at address 0x2000; where does array B start?
  - 0x4000
- Array C?
  - 0x6000
- What are the implications in the cache?
  - If addresses are 32 bits, how large are the offset, index, and tag?

	0x2000	0x4000	0x6000
a[0], b[0], c[0]	0010 0000 0000 000	0100 0000 0000 0000	0110 0000 0000 0000

Tag = 19	Index = 7 bits (8192/64 = 128 = 2^7)	Offset = 6 bits (2^6 = 64)

The problem with a direct mapped cache

- If array A starts at address 0x2000; where does array B start?
  - 0x4000
- Array C?
  - 0x6000
- What are the implications in the cache?
  - If addresses are 32 bits, how large are the offset, index, and tag?

	0x2000	0x4000	0x6000
a[1], b[1], c[1]	0010 0000 0000 1000	0100 0000 0000 1000	0110 0000 0000 1000

Tag = 19	Index = 7 bits (8192/64 = 128 = 2^7)	Offset = 6 bits (2^6 = 64)

The problem with a direct mapped cache

- If array A starts at address 0x2000; where does array B start?
  - 0x4000
- Array C?
  - 0x6000
- What are the implications in the cache?
  - If addresses are 32 bits, how large are the offset, index, and tag?

	0x2000	0x4000	0x6000
a[8], b[8], c[8]	0010 0000 0100 0000	0100 0000 0100 0000	0110 0000 0100 0000

Tag = 19	Index = 7 bits (8192/64 = 128 = 2^7)	Offset = 6 bits (2^6 = 64)

The problem with a direct mapped cache

- If array A starts at address 0x2000; where does array B start?
  - 0x4000
- Array C?
  - 0x6000
- What are the implications in the cache?
  - If addresses are 32 bits, how large are the offset, index, and tag?

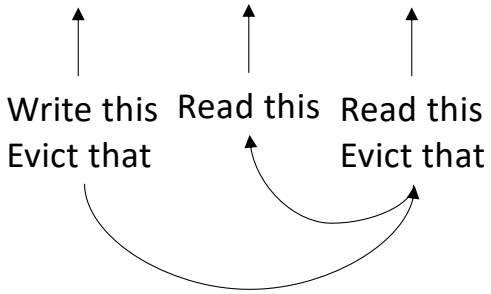
	0x2000	0x4000	0x6000
a[1023], b[1023], c[1023]	0011 1111 1111 1000	0101 1111 1111 1000	0111 1111 1111 1000

Tag = 19	Index = 7 bits (8192/64 = 128 = 2^7)	Offset = 6 bits (2^6 = 64)

Counting misses:

```
long a[1024];
long b[1024];
long c[1024];

for (int i = 0; i < 1024; i++) {
      c[i] = a[i] + b[i];
}
```

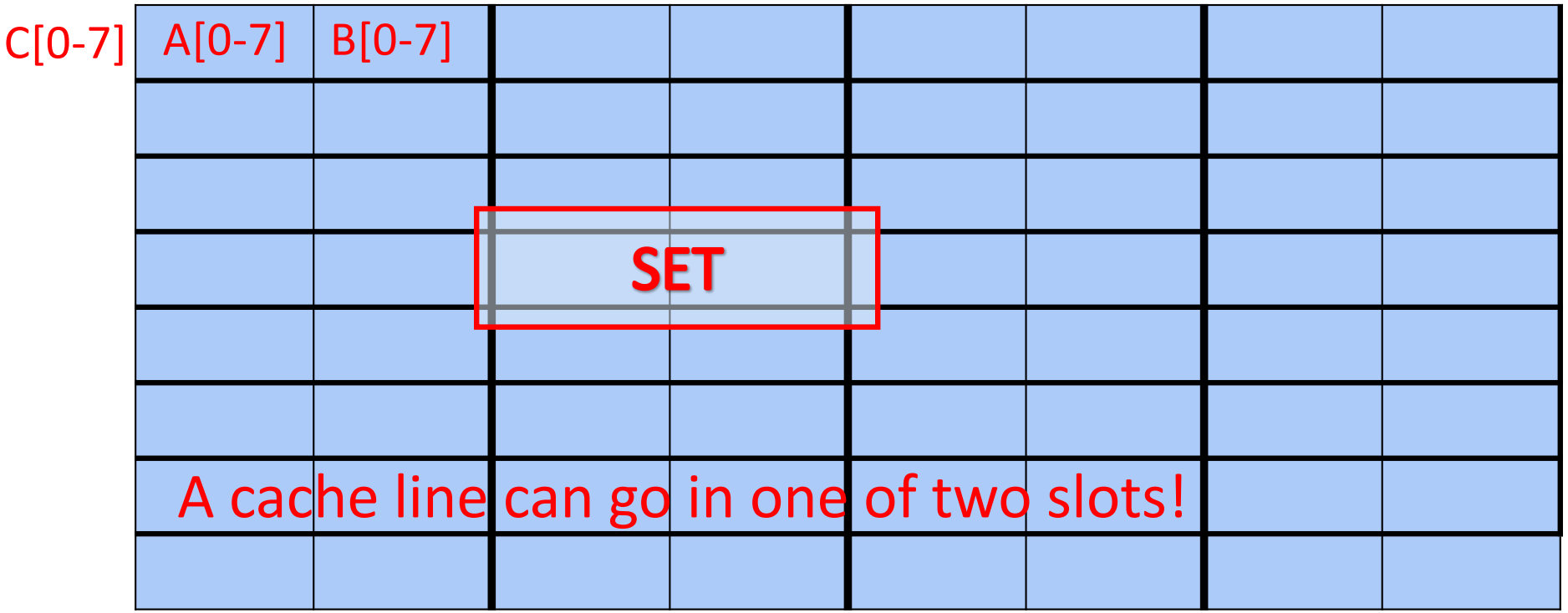


How do we fix this?

Cache Arrangement: Fully Associative

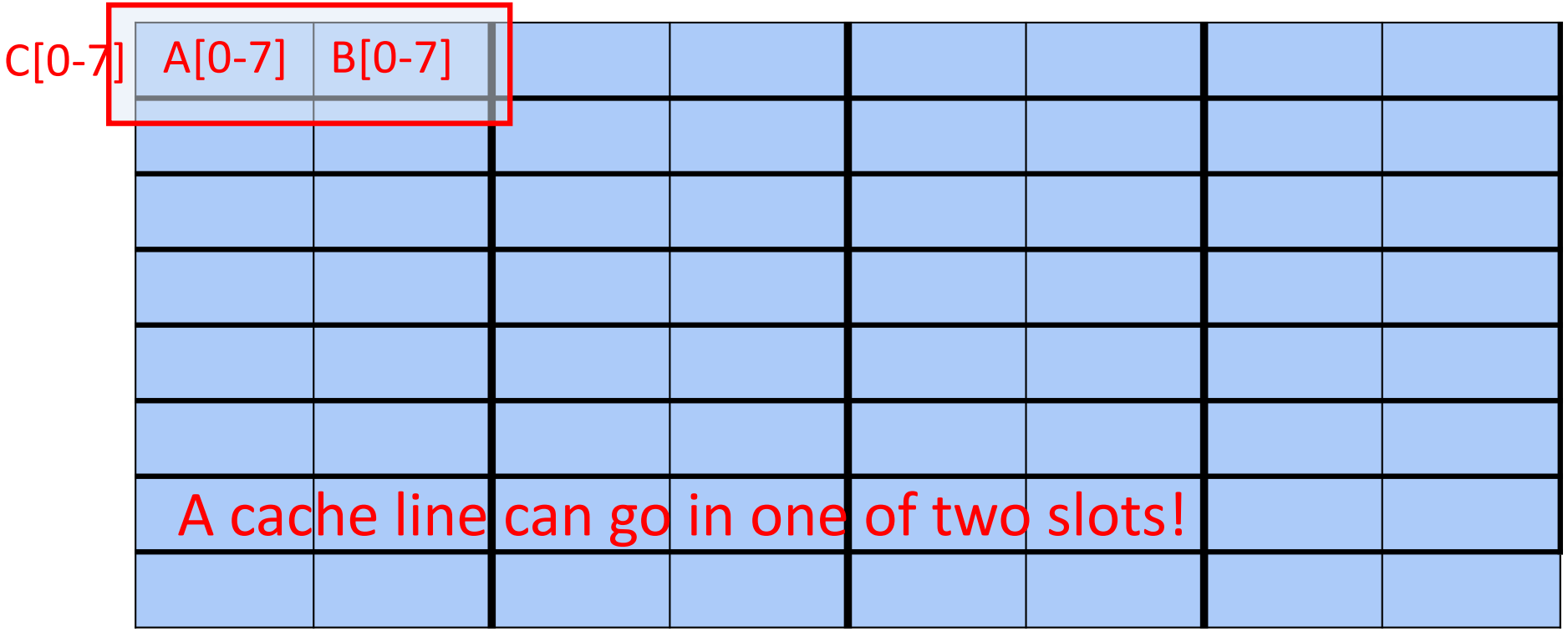
Each box is a slot

Cache Arrangement: 2-way set associative



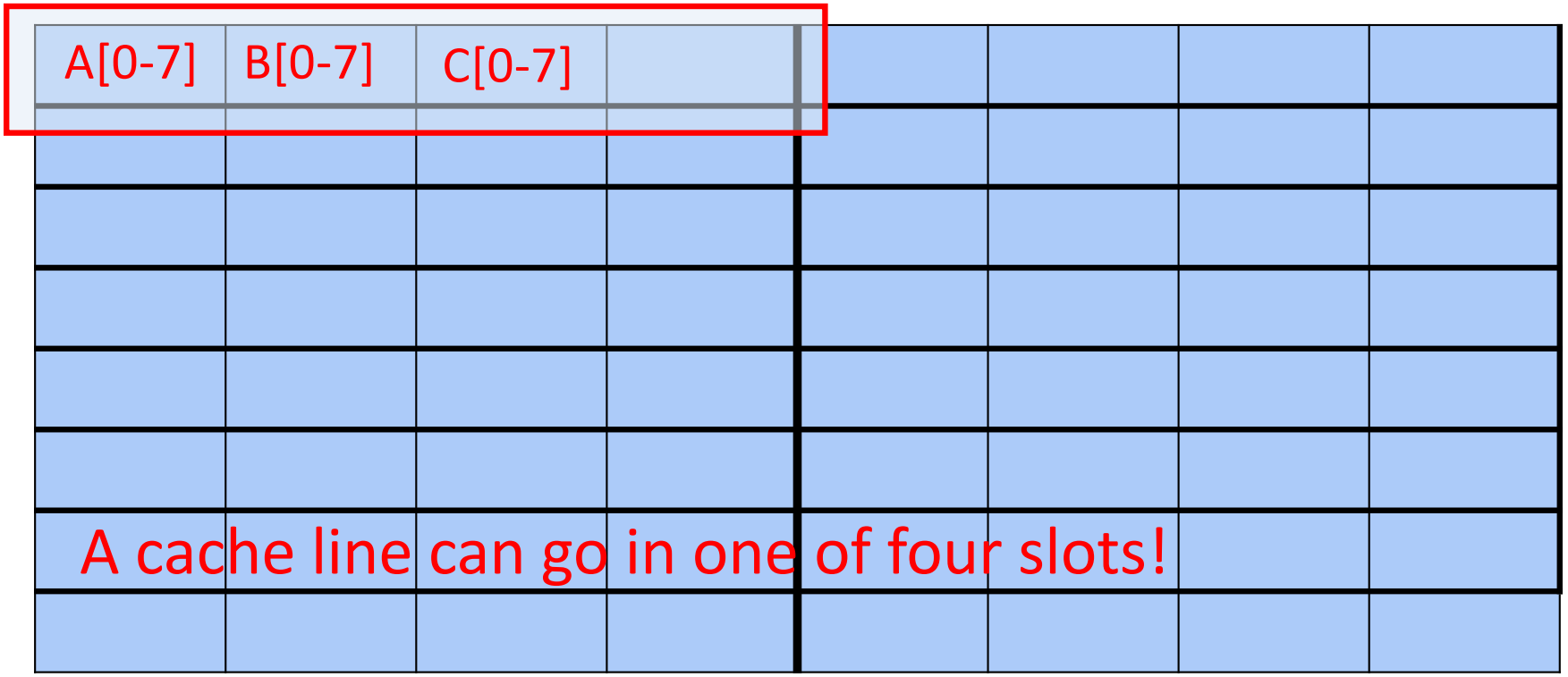
A slot holds a cache line

Set Associativity: Eviction Policy



A slot holds a cache line

Set Associativity: 4-way set associative



A slot holds a cache line

Associativity and Mapping Addresses to Cache Lines

- Let's continue to assume an 8 KB cache with 64 bit cache lines and look at what happens to how we pick index bits depending on set associativity.
- Direct mapped cache:

Tag = 19	Index = 7 bits (8192/64 = 128 = 2^7)	Offset = 6 bits (2^6 = 64)

  - Why?
    - 8192 bytes / 64 bytes-per-line = 128 lines => we need 7 bits to pick a line

Associativity and Mapping Addresses to Cache Lines

- Let's continue to assume an 8 KB cache with 64 byte cache lines and look at what happens to how we pick index bits depending on set associativity.
- What happens when we make this 2-way set associative?

Tag = 20 bits	Index = 6 bits	**Offset = 6 bits (2^6 = 64)**

  - Our cache is still large enough to hold 128 lines, but
    - each line can go in two places
    - the address has only half as many places from which to choose (because each 'place' holds two cache lines).
    - So, when we INCREASE associativity, we DECREASE the number of index bits and INCREASE the number of tag bits.