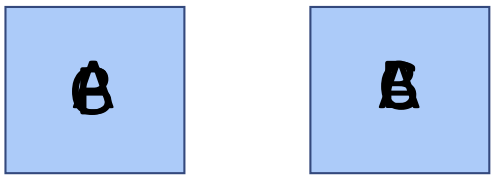
Today

- Learning Objectives
  - Analyze different cache eviction policies
    - When LRU fails: MRU
    - First-in-first-out: FIFO
- How we'll get there
  - Review in-class problem
  - Look at one last replacement algorithm
- Reading
  - 6.4

When LRU is terrible

- We left you in class asking you to derive a sequence of accesses that made LRU perform terribly.
- It's a common and simple access pattern:

A, B, C, A, B, C



Compulsory misses:

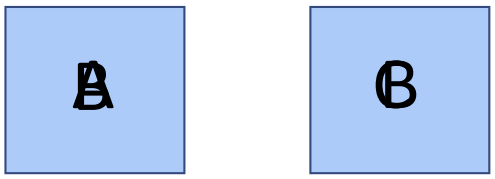
Capacity misses:

Hits:

LRU is always doing the wrong thing!

- Iterating over something is pretty common, and if that something is too big, LRU is awful.
- What would be better?

A, B, C, A, B, C



Compulsory misses: I I I
Capacity misses: I
Hits: I I

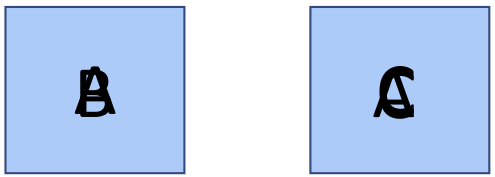
Most-Recently-Used: MRU

- Ideal when iterating over an array that is too large.
  - In other workloads, not necessarily particularly good.

Let's see what FIFO does.

- Evict the item that was loaded into the cache earliest.
  - Assume we have 2 slots in our cache.
  - Assume fully associative
  - We access cache blocks in the following order:

A, A, C, A, C, B, B, B, A, A, B, C



Compulsory misses: I I I

Capacity misses: I I

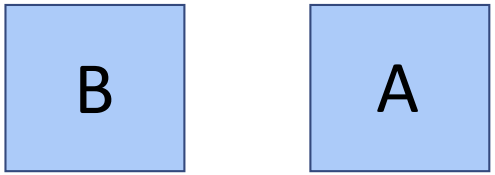
Hits: I I I I I I I

Let's see what FIFO does.

- Evict the item that was loaded into the cache earliest.
  - Assume we have 2 slots in our cache.
  - Assume fully associative
  - We access cache blocks in the following order:

A, A, C, A, C, B, B, B, A, A, B, C

Hit Rate: 7/12



Compulsory misses: I I I

Capacity misses: I I

Hits: I I I I I I I

There is no Perfect Policy

- Most commonly used:
  - LRU (frequently just fine)
  - LFU (sometimes better)
  - MRU (when you detect the right pattern)
- For anything but Belady, we can construct an adversarial workload that can be pretty devastating.