Consistency

What is printed?

0, 1, and 2 can be printed!
Consistency

Thread 1 on Core 1
Write 2 to x in local cache
Write 3 to y in local cache
3 gets pushed to y in memory

Thread 2 on Core 2
2 gets pushed to x in memory

Not sequentially consistent!
Harsh Reality

- **Sequential Consistency**
  - There is an interleaving of the parallel operations that explains the observations and events
  - Currently unknown how to implement efficiently

- **Volatile keyword**
  - Java fields can be declared volatile
  - Writing to a volatile variable ensures all local changes are made visible to other threads
  - $x$ and $y$ would have to be made volatile to fix code
Atomicity

volatile int x = 0;

Thread 1
x++;  

Thread 2

What is the value of x?

Can be both 1 and 2!
class AtomicInteger, AtomicReference<T>, ...

- Represents a value

method set(newValue)
- has the effect of writing to a volatile variable

method get()
- returns the current value

effectively an extension of volatile

but what about atomicity???
Compare and Set (CAS)

- boolean compareAndSet(expectedValue, newValue)
  - If value doesn’t equal expectedValue, return false
  - if equal, store newValue in value and return true
  - executes as a single atomic action!
  - supported by many processors
  - without requiring locks!

```java
AtomicInteger n = new AtomicInteger(5);
n.compareAndSet(3, 6); // return false – no change
n.compareAndSet(5, 7); // returns true – now is 7
```
/** Increment n by one. Other threads use n too. */

public static void increment(AtomicInteger n) {
    int i = n.get();
    while (n.compareAndSet(i, i+1))
        i = n.get();
}

// AtomicInteger has increment methods doing this
Lock-Free Data Structures

- Usable by many concurrent threads
- using only atomic actions – no locks!
- compare and swap is god here
- but it only atomically updates one variable at a time!

Let’s implement one!