Consistency

```java
x = 2;
y = 3;
a = y > 0 ? x : 0;
System.out.println(a);
```

What is printed?

0, 1, and 2 can be printed!

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

```java
volatile int x = 0;
```

What is the value of x?

Can be both 1 and 2!
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!

```
AtomicInteger n = new AtomicInteger(5);
n.compareAndSet(3, 6); // return false – no change
n.compareAndSet(5, 7); // returns true – now is 7
```

Incrementing with CAS

```java
/** 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!