Consistency

Consistency

Harsh Reality

Atomicity

Java utility concurrent atomic

constituents

Sequential consistency

Volatile keyword

Atomicity

volatile int x = 0;

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
```

**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!