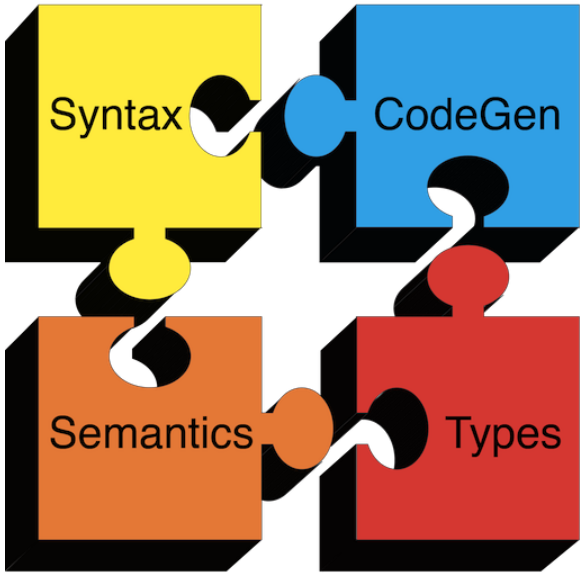


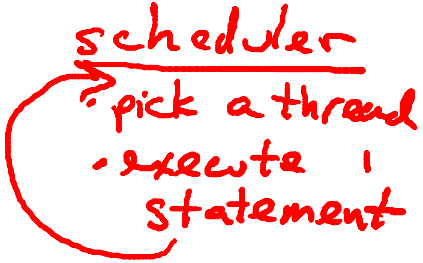
Compilers

Java Threads

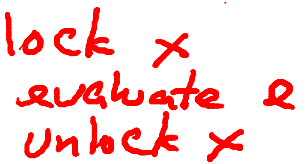
Java Threads

- Java has concurrency built in through *threads*
  - Each thread has its own program counter & stack

- Thread objects have class Thread
  - Start and stop methods



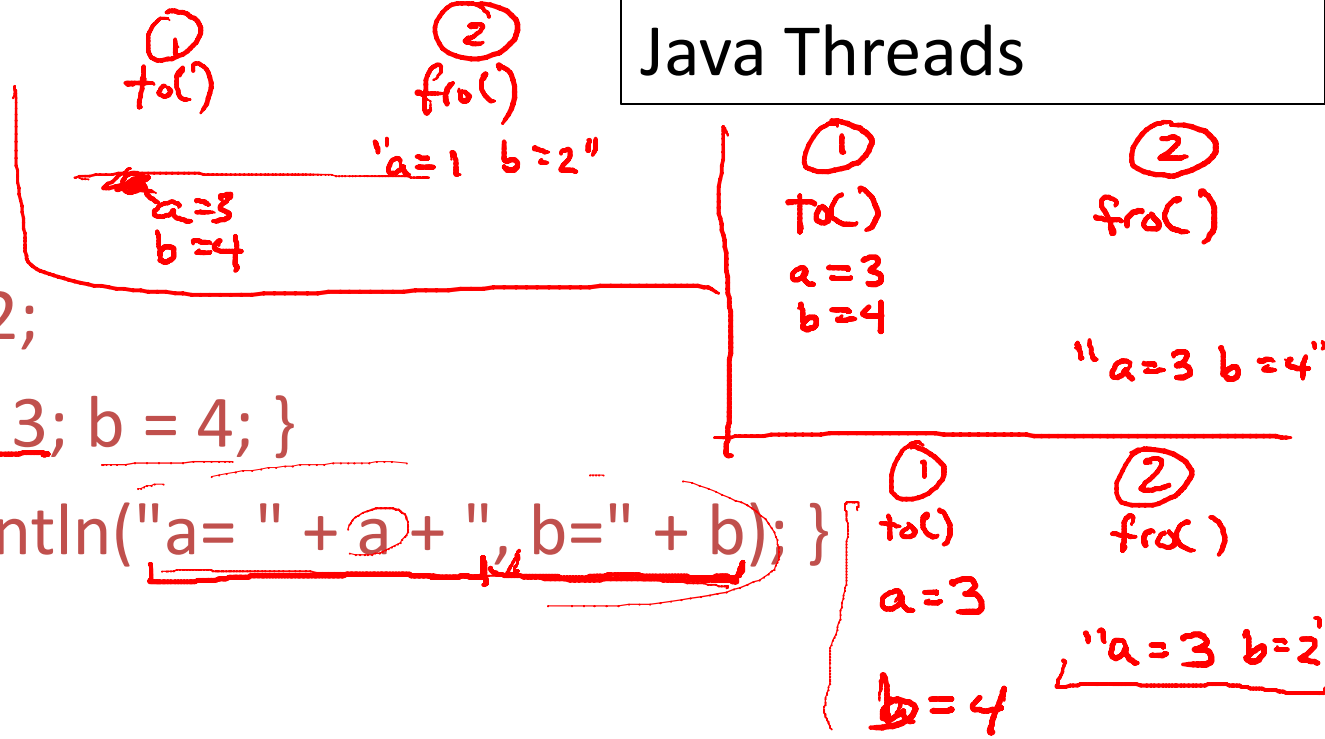
- Synchronization obtains a lock on the object:
  - → synchronized (x) { e }
- In synchronized methods, this is locked



synchronized f( ) { }

Java Threads

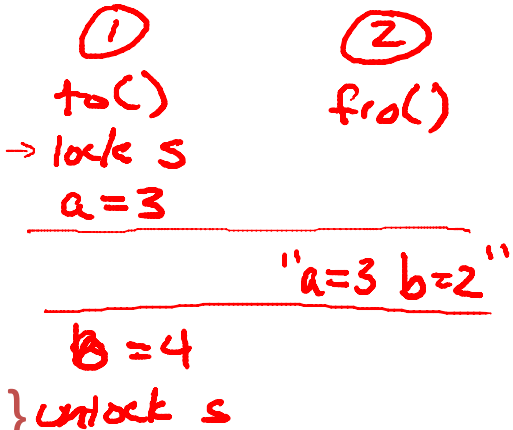
```
class Simple {
    int a = 1, b = 2;
    void to() { a = 3; b = 4; }
    void fro() {println("a= " + a + ", b=" + b); }
}
```



Two threads call to() and fro(). What is printed?

Java Threads


```
class Simple {
    int a = 1, b = 2;
    void synchronized to() { a = 3; b = 4; }
    void fro() {println("a= " + a + ", b=" + b); }
}
```



Two threads call to() and fro(). What is printed?

Java Threads

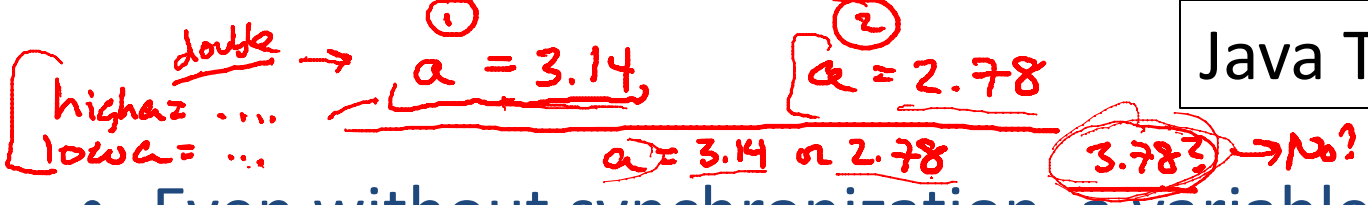
```
class Simple {
    int a = 1, b = 2;
→   void synchronized to() { a = 3; b = 4; }
→   void synchronized fro() {println("a= " + a + ", b=" + b); }
}
```

"a = 1 b = 2"
fro() before to()

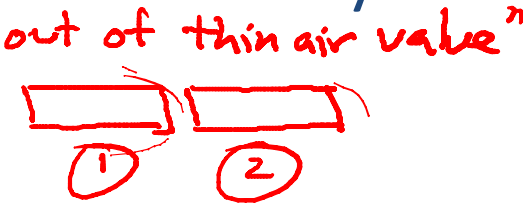
"a=3 b = 4"
to() before fro()

Two threads call to() and fro(). What is printed?

Java Threads



- Even without synchronization, a variable should only hold values written by some thread
  - Writes of values are atomic
  - Violated for doubles, though



volatile

- Java concurrency semantics are difficult to understand in detail, particularly as to how they might be implemented on certain machines