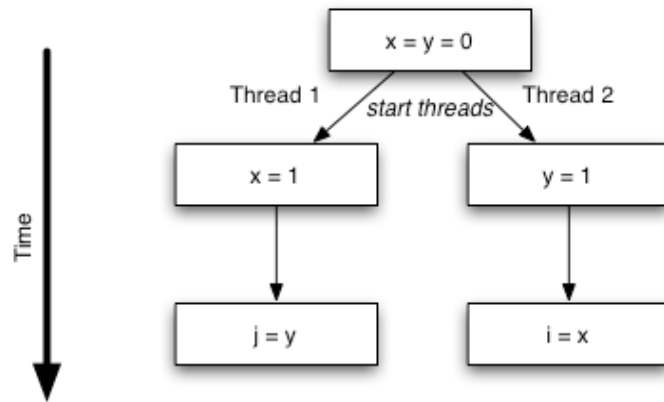


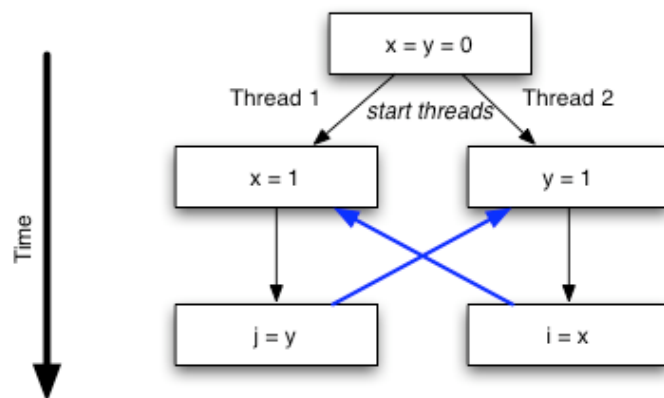
Quiz Time



- Can this result in $i=0$ and $j=0$?

82

Doesn't Seem Possible...



- But this can happen!

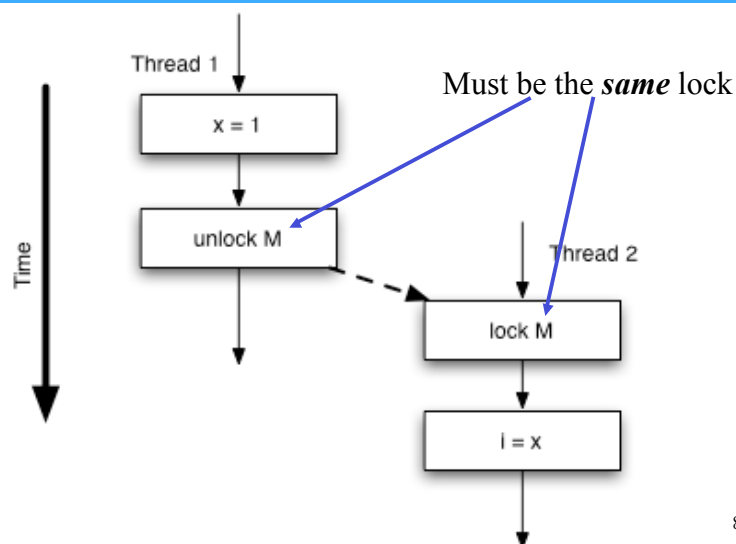
83

How Can This Happen?

- Compiler can reorder statements
 - Or keep values in registers
- Processor can reorder them
- On multi-processor, values not synchronized in global memory

84

When Are Actions Visible?



85

Forcing Visibility of Actions

- All writes from thread that holds lock M are visible to next thread that acquires lock M
 - Must be the same lock
- Use synchronization to enforce **visibility** and **ordering**
 - As well as mutual exclusion

86

Volatile Fields

- If you are going to access a shared field without using synchronization
 - It needs to be **volatile**
- Semantics for **volatile** have been strengthened in JSR-133
 - Many VM's already compliant
- If you don't try to be too clever
 - Declaring it **volatile** just works

87

Using Volatile

- A one-writer/many-reader value
 - Simple control flags:
 - `volatile boolean done = false;`
- Keeping track of a “recent value” of something

88

Misusing Volatile

- Incrementing a volatile field doesn't work
 - In general, writes to a volatile field that depend on the previous value of that field don't work
 - Use `AtomicInteger`, etc.
- A volatile reference to an object isn't the same as having the fields of that object be volatile
 - No way to make elements of an array volatile
- Can't keep two volatile fields in sync

89

Selected Guidelines for Programming with Threads

- Synchronize access to shared data
- Don't hold multiple locks at a time
 - Could cause deadlock
- Hold a lock for as little time as possible
 - Reduces blocking waiting for locks
- While holding a lock, don't call a method you don't understand
 - E.g., a method provided by someone else, especially if you can't be sure what it locks
 - Corollary: document which locks a method acquires

90