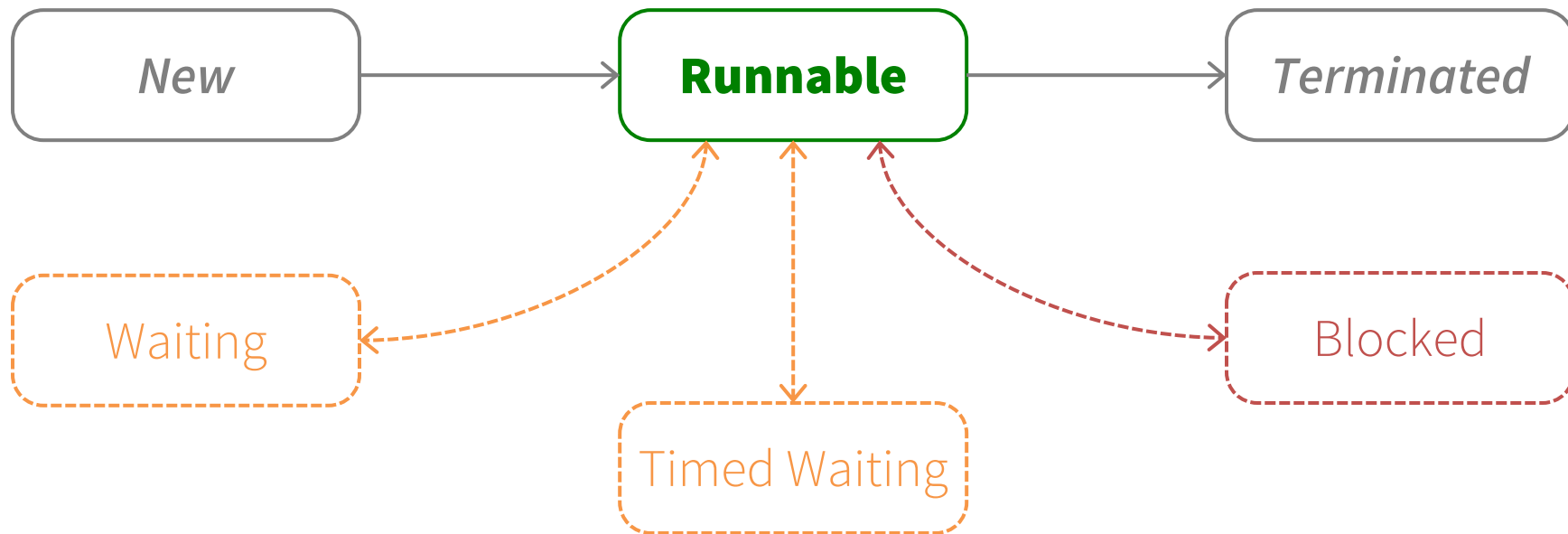


Multithreading

Custom Locks



Thread States



<http://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html>

Motivation

- Need **multithreading** to speedup calculation for large, complex problems
- Need **synchronization** to protect data (**memory consistency**) and operations (**atomicity**)
- The **synchronized** keyword causes **blocking**, reducing the speedup needed in the first place



Motivation

- **Assume have a large shared data structure**
 - When is it okay to read from this data structure?
 - When is it okay to write to this data structure?
- **What operations may occur concurrently?**
 - Thread 1 reads A, Thread 2 reads A
 - Thread 1 reads A, Thread 2 writes A
 - Thread 1 writes A, Thread 2 writes A



Motivation

- **Assume have a large shared data structure**
 - When is it okay to read from this data structure?
 - When is it okay to write to this data structure?
- **What operations may occur concurrently?**
 - Thread 1 reads A, Thread 2 reads A
 - ~~– Thread 1 reads A, Thread 2 writes A~~
 - ~~– Thread 1 writes A, Thread 2 writes A~~



Concurrent Operations

- **Mutual Exclusion**

- Only one thread may enter synchronized code at a time (blocking other threads)
- Lots of blocking defeats purpose of multithreading

- **Conditional Synchronization**

- Only block if certain conditions are true
- Uses a combination of `wait()` and `notify()`



Using Wait and Notify

- A thread will keep its locks during **sleep()**
 - Rarely appropriate, except for debugging
- A thread releases its lock during **wait()**
 - Hence why **wait()** must be called on the lock object in the synchronized block
 - Only return from **wait()** if able to reacquire lock



Using Wait and Notify

- Both **wait()** and **notify()** require lock objects
 - Can have threads waiting on different locks
- Only one waiting thread woken up by **notify()**
- All threads waiting on lock woken up by **notifyAll()**
 - Usually used, despite sometimes being slower
 - Place **wait()** in while loop to make sure thread woke up for the correct reason



Custom Lock Class

- May read to shared data structure if...
 - No other threads are writing to it
- May write to shared data structure if...
 - No other threads are reading from it
 - No other threads are writing to it
- Must track...
 - Number of active readers and writers

<http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/ReadWriteLock.html>



Creating a Custom Lock Class

- **Lock Methods**

- Wait until safe to acquire lock
- Use a loop to avoid **spurious wakeups**
- Use wait() and notifyAll() to avoid **busy-wait**
- Increase number of threads with lock

- **Unlock Methods**

- Decrease number of threads with lock
- Wake up threads if necessary using notifyAll()

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Condition.html>



Using a Custom Lock Class

```
1 CustomLock lock = new CustomLock();
2 SharedData data = new SharedData();
3
4 lock.lockReadOnly();           // protects read-only operations
5 data.read();
6 lock.unlockReadOnly();
7
8 lock.lockReadWrite();          // protects write operations
9 data.read();                  // or read/write operations
10 data.write();
11 lock.unlockReadWrite();
```



Creating a Custom Lock Class

```
1 public synchronized void lockReadOnly() {  
2     while (writers > 0) {  
3         try {  
4             this.wait();  
5         }  
6         catch (InterruptedException e) {  
7             // log the exception  
8         }  
9     }  
10    readers++;  
11 }
```



Built-in Lock Objects

- See **java.util.concurrent.locks**
 - May not actually use any of these in class, but might be useful for debugging and testing
- Closest to **ReentrantReadWriteLock**
 - Our version prone to starvation
 - Their version provides a fairness policy

<http://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/package-summary.html>





CHANGE THE WORLD FROM HERE