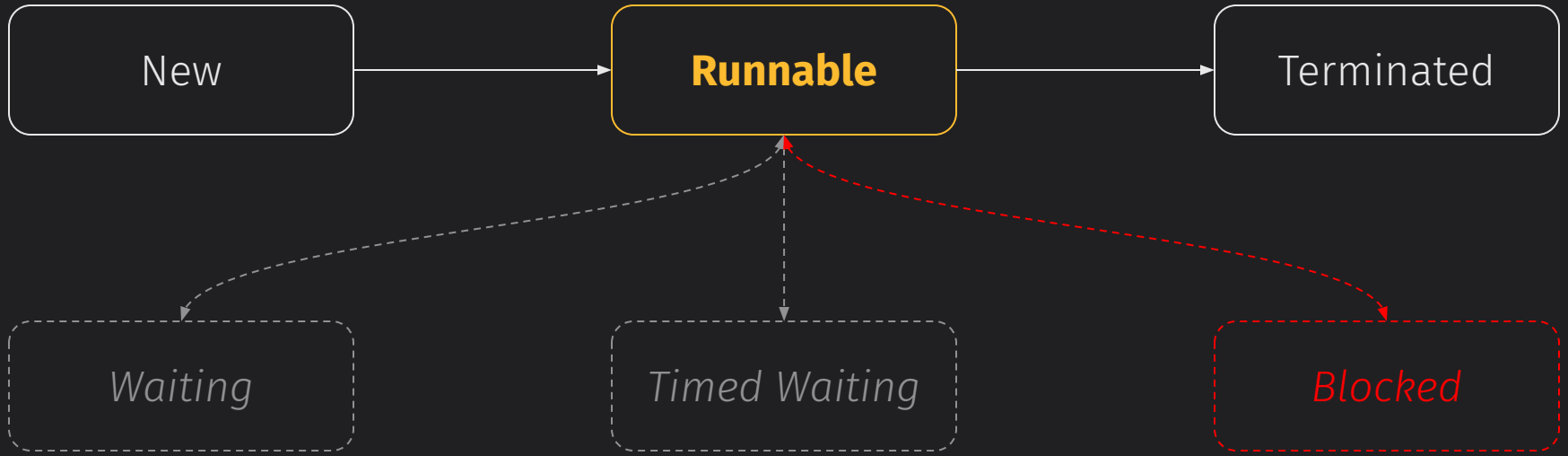
USF UNIVERSITY OF SAN FRANCISCO
CHANGE THE WORLD FROM HERE

Conditional Locks

CS 272 Software Development

Professor Sophie Engle
Department of Computer Science

Thread States



https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/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**
  - One thread may run 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()`

Simple Read/Write Lock

- 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 or writing the data
- Must track...
  - Number of active readers and writers

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/concurrent/locks/ReadWriteLock.html

Simple Read/Write Lock

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

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/concurrent/locks/Condition.html

Simple Read/Write Lock

- Unlock methods
  - Decrease number of threads with lock
  - Wake up threads if necessary using `notifyAll()`
- Separate lock methods for read and read/write

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/concurrent/locks/Condition.html

```
1. ReadWriteLock lock = new ReadWriteLock();
2. SharedData data = new SharedData();
3.
4. lock.readLock().lock();   // protects read-only ops
5. data.read();
6. lock.readLock().unlock();
7.
8. lock.writeLock().lock();  // protects write operations
9. data.read();              // or read/write operations
10. data.write();
11. lock.writeLock().unlock();
```

Using a Simple Read/Write Lock

```
1. while (writers > 0) {
2.     try {
3.         this.wait(); // assumes synchronized method
4.     }
5.     catch (InterruptedException e) {
6.         // log and re-interrupt
7.     }
8. }
9.
10. readers++;
```

Example Read Lock Implementation

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`
  - Ours prone to starvation, theirs has fairness policy
  - Supports reentrant locks (re-acquiring locks)

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/concurrent/locks/package-summary.html

Improved Read/Write Lock

- Must also track...
  - Active writer thread
- May read to shared data structure if...
  - Active writer -or- no other threads are writing to it
- May write to shared data structure if...
  - Active writer -or- no threads reading or writing

https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/util/concurrent/locks/ReentrantReadWriteLock.html

CHANGE THE WORLD FROM HERE

Software Development
Department of Computer Science

Professor Sophie Engle
sjengle.cs.usfca.edu