

Coordination

CS 272 Software Development

Providing Consistency

- If **multithreading**...
 - If **sharing data** between threads...
 - If shared data not already **thread safe**...
 - must **synchronize** access to that data



Synchronization

- Using the **synchronized** keyword and intrinsic (or monitor) lock objects to protect blocks of code
- Using the **volatile** keyword to protect* variables
- Using **wait()** and **notifyAll()** to coordinate threads
- Using **conditional synchronization** via lock objects



Synchronization

- Using the **synchronized** keyword and intrinsic (or monitor) lock objects to protect blocks of code
- Using the **volatile** keyword to protect* variables
- Using **wait()** and **notifyAll()** to coordinate threads
- Using **conditional synchronization** via lock objects



Motivation

- Synchronization helps coordinate threads with shared resources and provide thread safety
- Sometimes need coordination for other reasons
 - Scanner needs to wait for console input...
 - Server needs to wait for incoming requests...
 - Main thread needs to wait for work to complete...



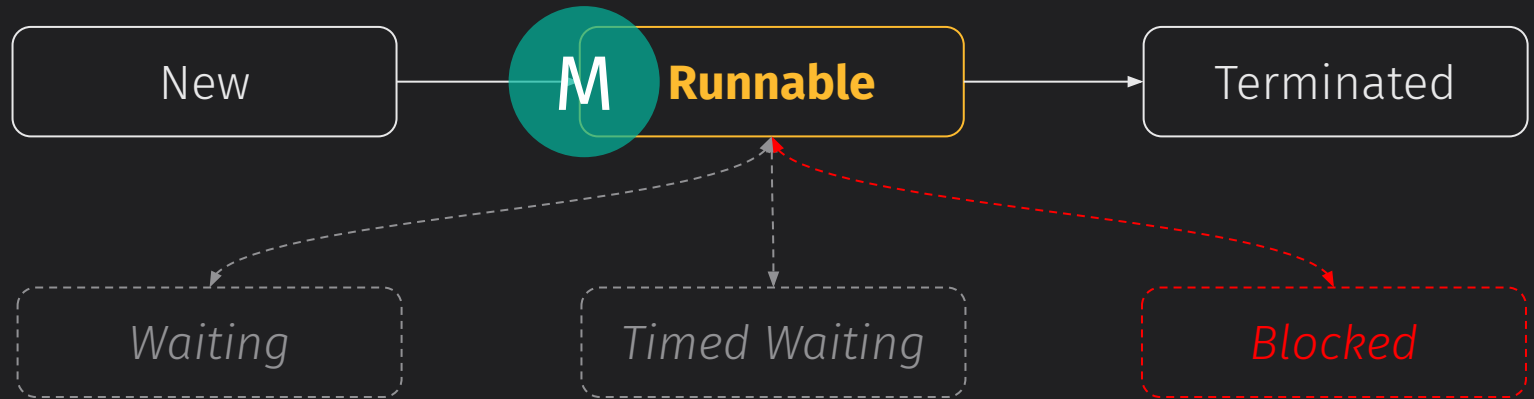
Example: Thread.join()

```
1. public static void main( ... ) {  
2.     Thread worker = new Thread();  
3.     worker.start(); // assume long-running  
4.     worker.join();  
5. }
```

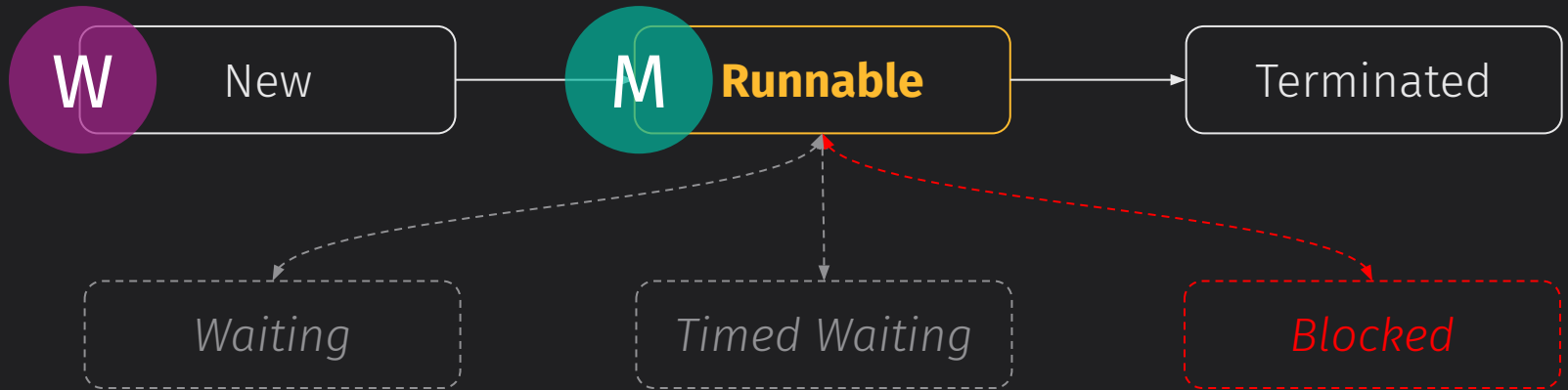
[https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Thread.html#join\(\)](https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Thread.html#join())



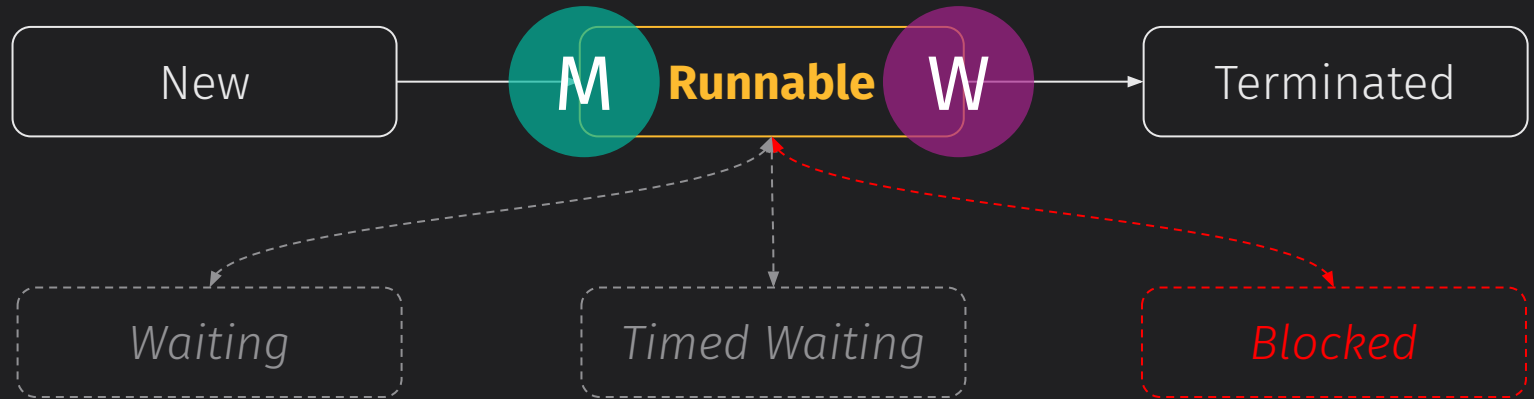
```
1. public static void main( ... ) {  
2.     Thread worker = new Thread();  
3.     worker.start();  
4.     worker.join();  
5. }
```



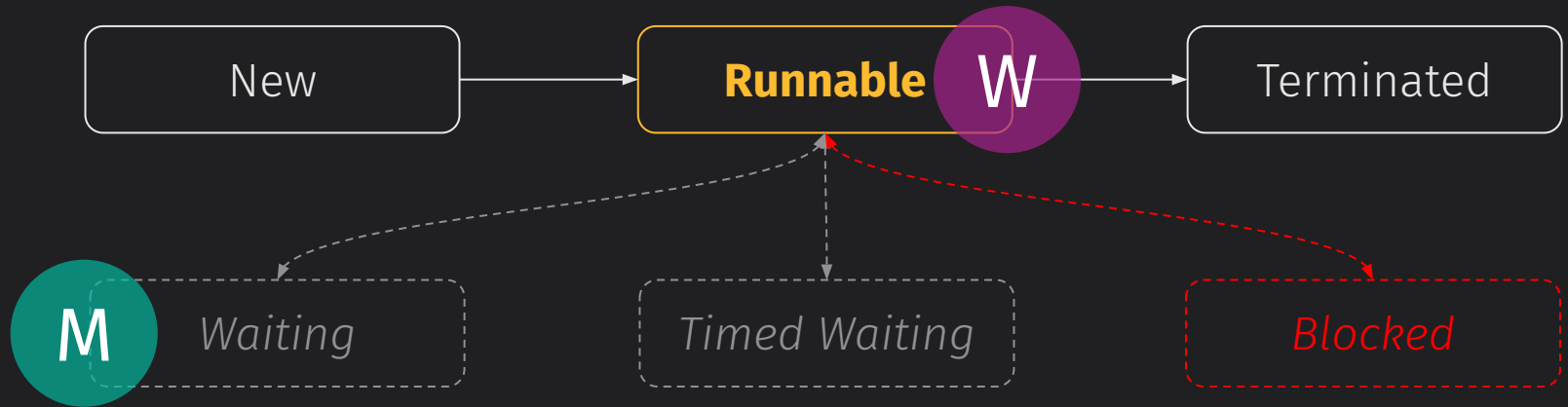
```
1. public static void main( ... ) {  
2.     Thread worker = new Thread();  
3.     worker.start();  
4.     worker.join();  
5. }
```



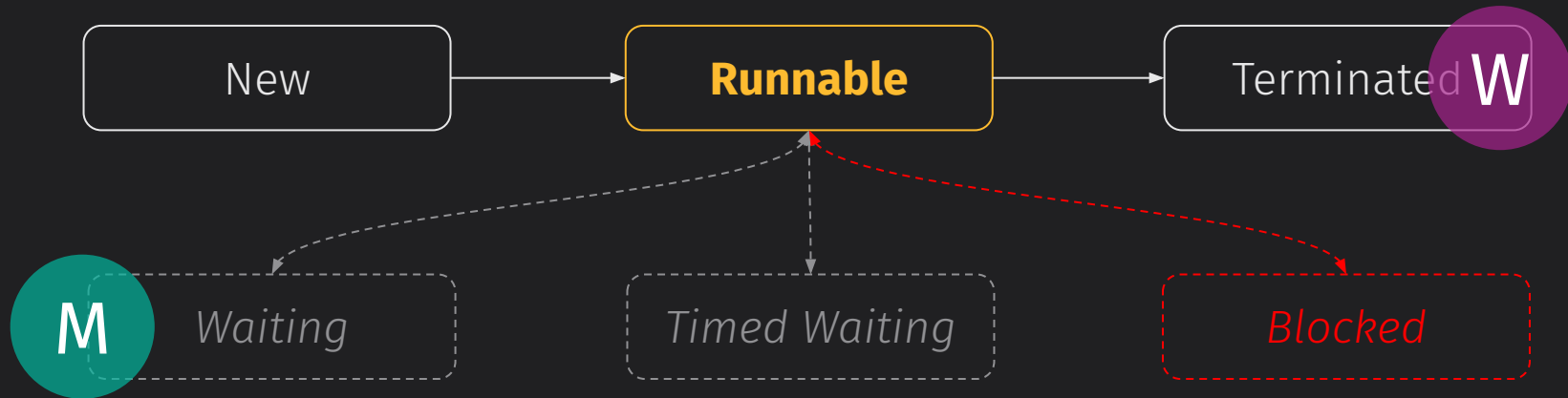

```
1. public static void main( ... ) {  
2.     Thread worker = new Thread();  
3.     worker.start();  
4.     worker.join();  
5. }
```



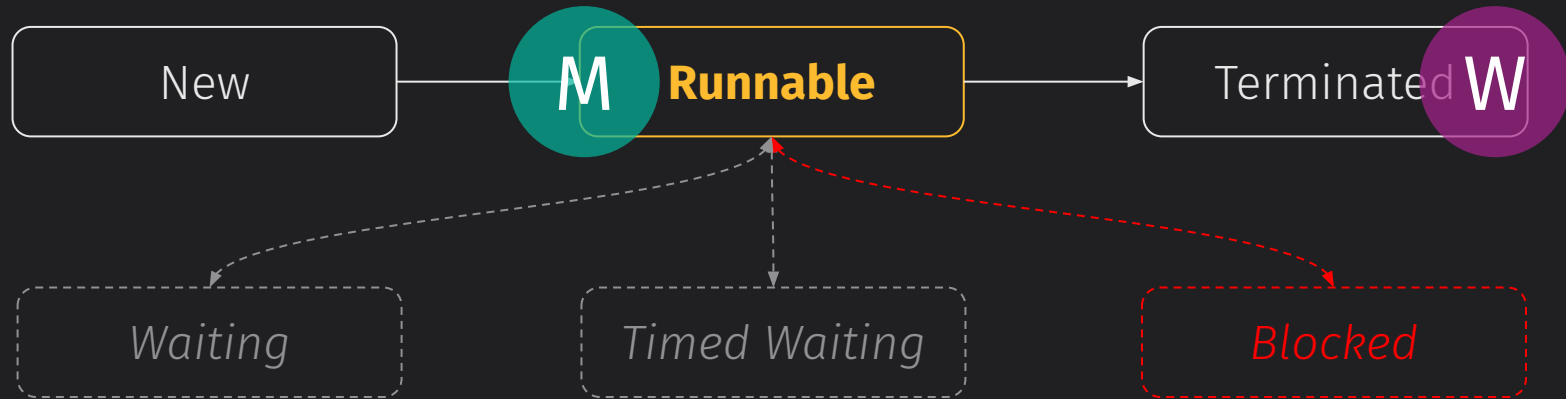
```
1. public static void main( ... ) {  
2.     Thread worker = new Thread();  
3.     worker.start();  
4.     worker.join();  
5. }
```



```
1. public static void main( ... ) {  
2.     Thread worker = new Thread();  
3.     worker.start();  
4.     worker.join();  
5. }
```



```
1. public static void main( ... ) {  
2.     Thread worker = new Thread();  
3.     worker.start();  
4.     worker.join();  
5. }
```



```
1. public static void main( ... ) {
2.     Thread worker = new Thread();
3.     worker.start();
4.     worker.join();
5. }
```



Example: Thread.join()

- The calling thread **main** calls `wait()` to transition from **RUNNABLE** to the **WAITING** state
- The target thread **worker** calls `notifyAll()` when it transitions to **TERMINATED** state
- The calling thread **main** wakes up and transitions from **WAITING** back into its **RUNNABLE** state



Using Wait and Notify

- Must be called within a synchronized block of code on the intrinsic lock object
 - `synchronized (lock) { lock.wait(); }`
 - `synchronized (this) { this.notify(); }`
- The intrinsic lock object determines which `wait()` calls are woken up by `notify()` and `notifyAll()` calls



Using Intrinsic Locks

```
synchronized (hello) { hello.wait(); }  
synchronized (hello) { hello.notify(); }  
synchronized (hello) { hello.notifyAll(); }
```

```
synchronized (world) { world.wait(); }  
synchronized (world) { world.notify(); }  
synchronized (world) { world.notifyAll(); }
```



Using wait(), wait(long), ...

- Current thread transitions from RUNNABLE to WAITING or TIMED WAITING state
- Releases intrinsic lock while waiting
- Waits until notified, timed out, interrupted, or... ??????
 - A **spurious wakeup** can occur (rarely)
 - Must wait in a while loop instead of if as a result!

[https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Object.html#wait\(\)](https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Object.html#wait())



Using notify(), notifyAll()

- Wakes up one or all threads waiting on lock
 - Rarely use notify() since unable to choose thread
- Awoken thread(s) attempt to acquire lock and transition back into RUNNABLE state
 - If unable to acquire lock, will be BLOCKED** until able to acquire lock object

[https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Object.html#notify\(\)](https://www.cs.usfca.edu/~cs272/javadoc/api/java.base/java/lang/Object.html#notify())



Using Thread.sleep()

- Pauses execution temporarily
- Does not release locks (i.e. blocking)
- Often used to test code if attempting to cause blocking
- Most cases should use `wait(...)` with a time instead

<https://docs.oracle.com/javase/tutorial/essential/concurrency/sleep.html>



Use Cases

- Bounded Buffer
 - Data structure for producer/consumer problems
- Work Queue
 - Manages worker threads and work
- Conditional Synchronization
 - Only block when certain conditions hold





CHANGE THE WORLD FROM HERE