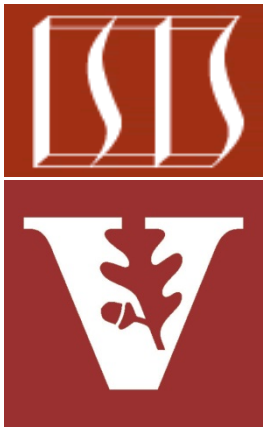
Android Concurrency: Overview of Java Synchronization & Scheduling Classes

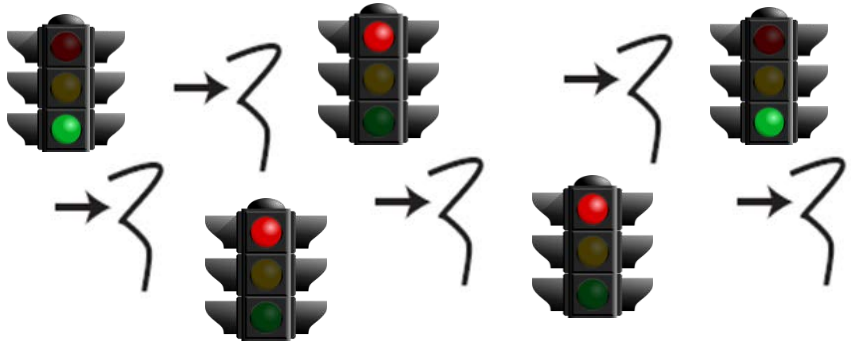


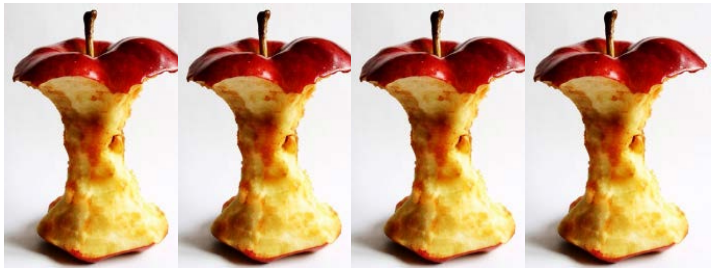
Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Recognize the key Java classes in Android that synchronize & schedule access to critical sections & interactions among threads in concurrent programs





We don’t show any Java code yet – that’s covered in subsequent parts

Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
  - e.g., java.util.concurrent & java.util.concurrent.lock

package **Added in API level 1**

java.util.concurrent.locks

Interfaces and classes providing a framework for locking and waiting for conditions that is distinct from built-in synchronization and monitors. The framework permits much greater flexibility in the use of locks and conditions, at the expense of more awkward syntax.
The `Lock` interface supports locking disciplines that differ in semantics (reentrant, fair, etc), and that can be used in non-block-structured contexts including hand-over-hand and lock reordering algorithms. The main implementation is `ReentrantLock`.

package **Added in API level 1**

java.util.concurrent

Utility classes commonly useful in concurrent programming. This package includes a few small standardized extensible frameworks, as well as some classes that provide useful functionality and are otherwise tedious or difficult to implement. Here are brief descriptions of the main components. See also the `java.util.concurrent.locks` and `java.util.concurrent.atomic` packages.

Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
Semaphore	A non-negative integer that controls the access of multiple threads to a limited number of shared resources
ConditionObject	Block thread(s) until some condition(s) becomes true
CountDownLatch	Allows one or more threads to wait until a set of operations being performed in other threads complete

Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
Semaphore	A non-negative integer that controls the access of multiple threads to a limited number of shared resources
ConditionObject	Block thread(s) until some condition(s) becomes true
CountDownLatch	Allows one or more threads to wait until a set of operations being performed in other threads complete

Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
Semaphore	A non-negative integer that controls the access of multiple threads to a limited number of shared resources
ConditionObject	Block thread(s) until some condition(s) becomes true
CountDownLatch	Allows one or more threads to wait until a set of operations being performed in other threads complete

Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
Semaphore	A non-negative integer that controls the access of multiple threads to a limited number of shared resources
ConditionObject	Block thread(s) until some condition(s) becomes true
CountDownLatch	Allows one or more threads to wait until a set of operations being performed in other threads complete

Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
Semaphore	A non-negative integer that controls the access of multiple threads to a limited number of shared resources
ConditionObject	Block thread(s) until some condition(s) becomes true
CountDownLatch	Allows one or more threads to wait until a set of operations being performed in other threads complete

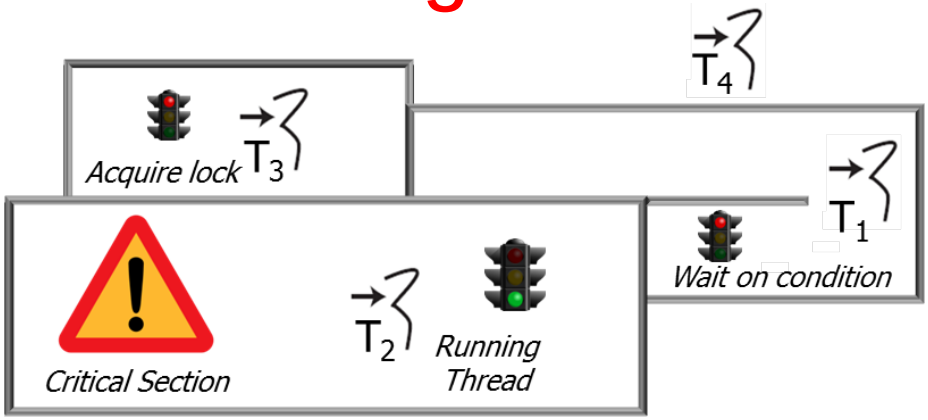
Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes

Java Class	Purpose
ReentrantLock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
ReentrantReadWriteLock	Improves performance when resources are read much more often than written
Semaphore	A non-negative integer that controls the access of multiple threads to a limited number of shared resources
ConditionObject	Block thread(s) until some condition(s) becomes true
CountDownLatch	Allows one or more threads to wait until a set of operations being performed in other threads complete

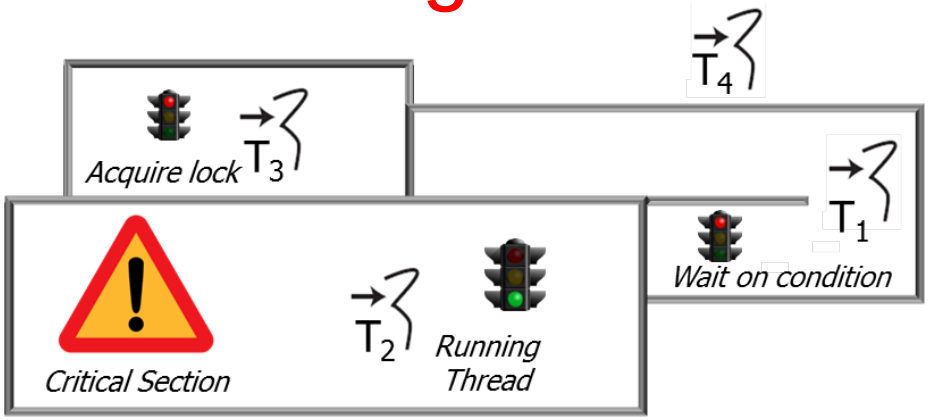
Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes
- These classes are distinct from Java's built-in monitor object mechanisms



Java Synchronization & Scheduling Classes

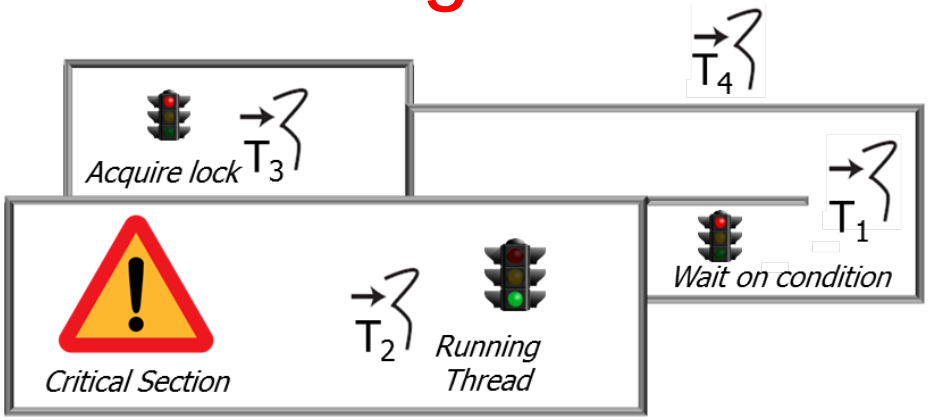
- Java provides many synchronization & scheduling classes
- We cover a subset of these classes
- These classes are distinct from Java's built-in monitor object mechanisms
  - e.g., the synchronized keyword & the wait(), notify(), & notifyAll() methods

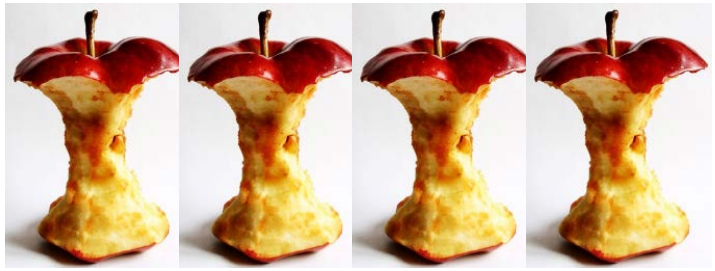


See docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html

Java Synchronization & Scheduling Classes

- Java provides many synchronization & scheduling classes
- We cover a subset of these classes
- These classes are distinct from Java's built-in monitor object mechanisms
- They are more fundamental & flexible, so we present them first

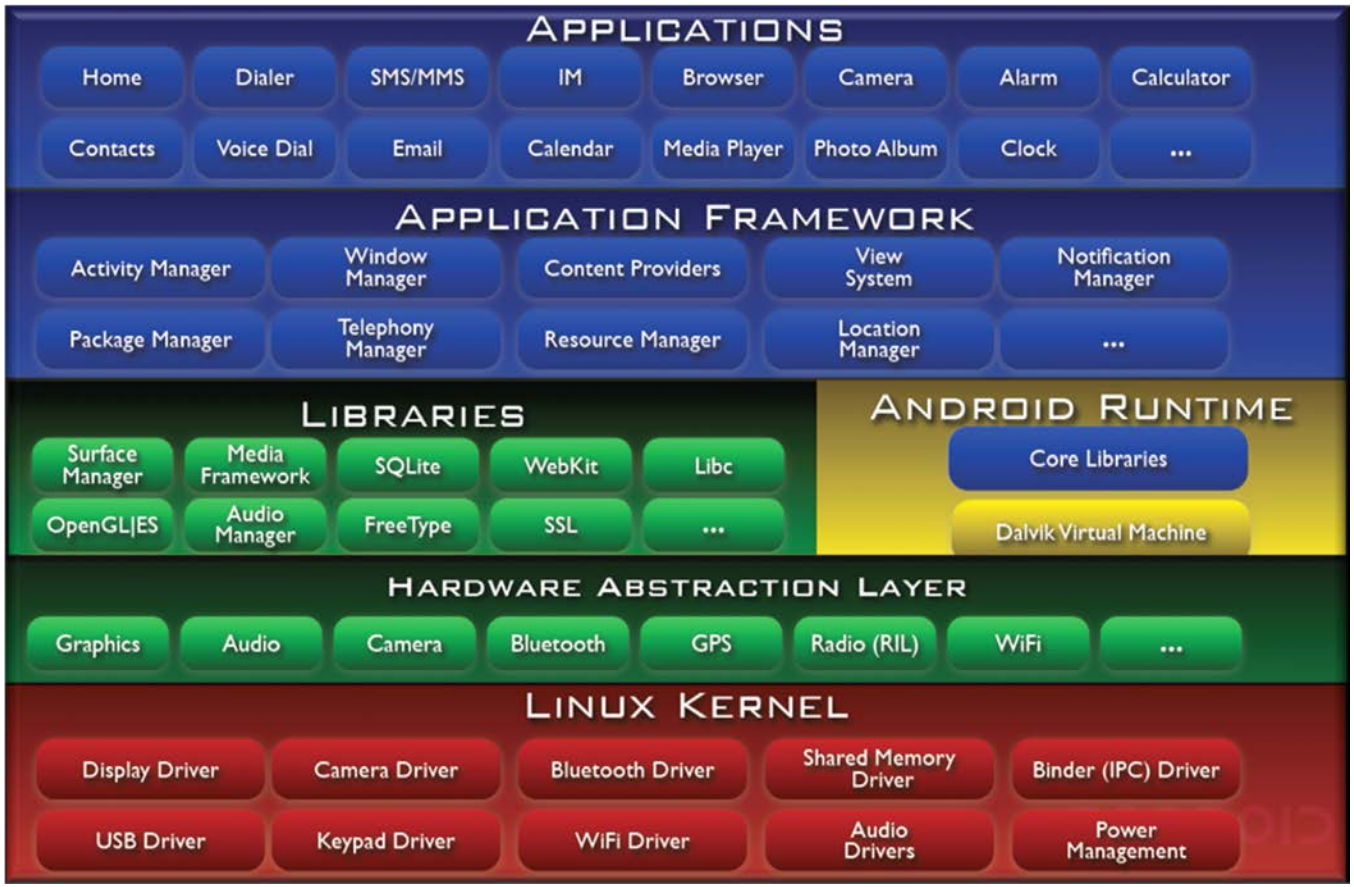




Performance & Usability Considerations

Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases



Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases

en.wikipedia.org/wiki/Spinlock has more on spin locks

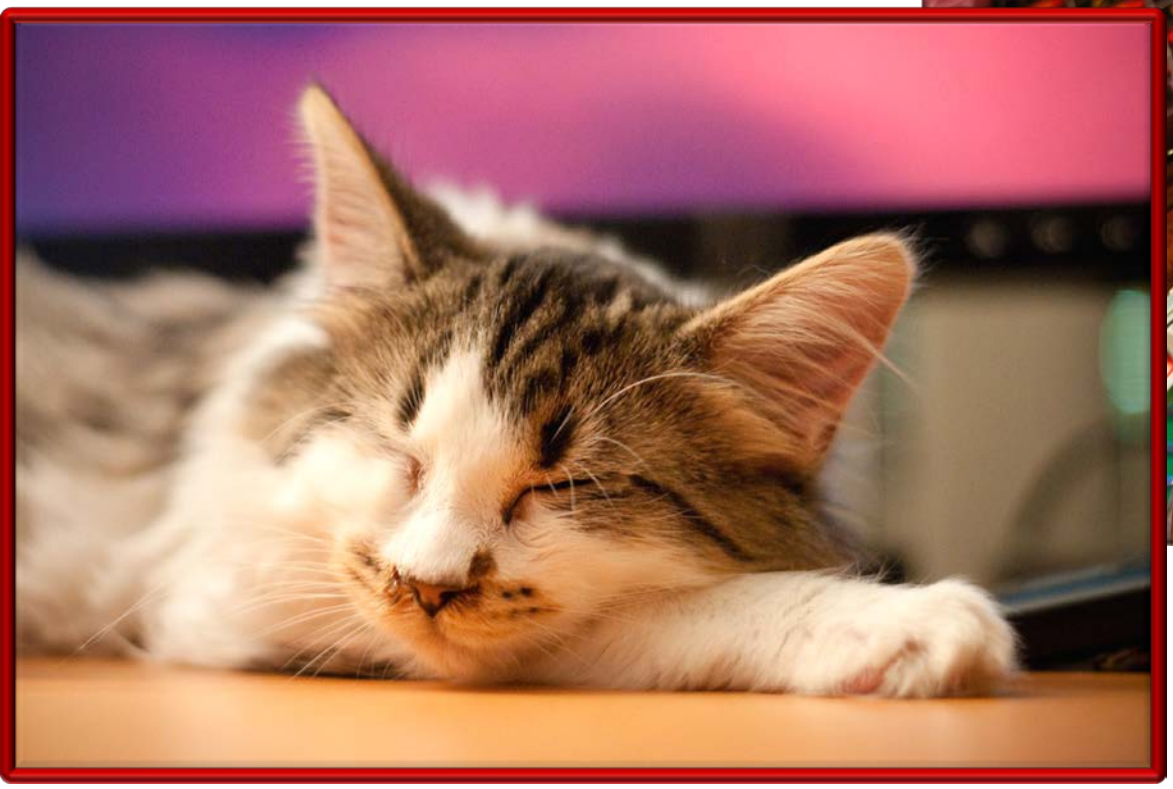
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases

en.wikipedia.org/wiki/Busy_waiting has more on busy waiting

Performance & Usability Considerations

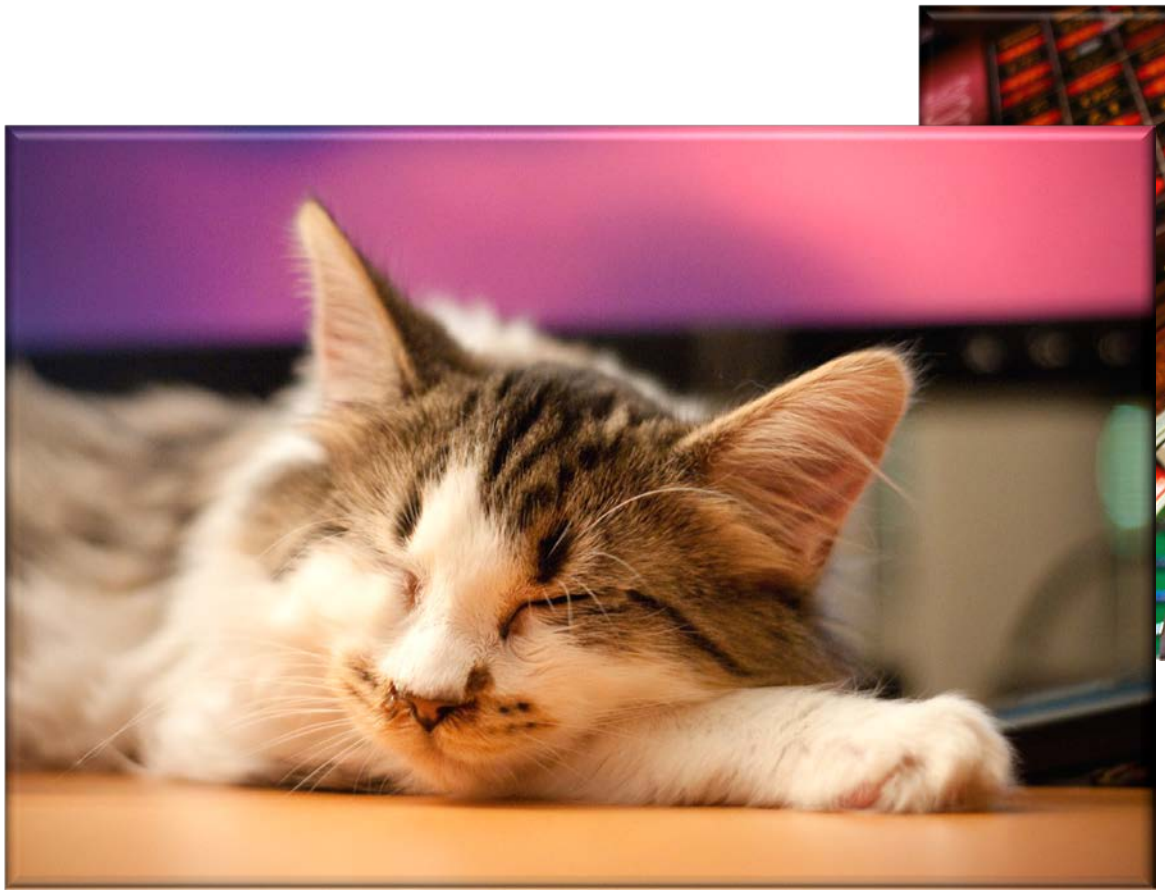
- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases



See docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html

Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases



www.dre.vanderbilt.edu/~schmidt/PDF/beyond_mp.pdf covers adaptive locks

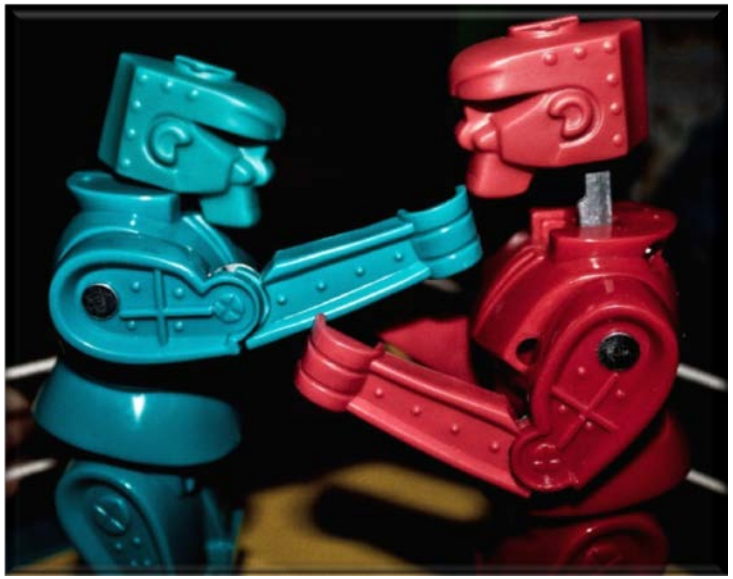
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases

www.youtube.com/watch?v=sq0MX3fHkro has more on java.util.concurrent

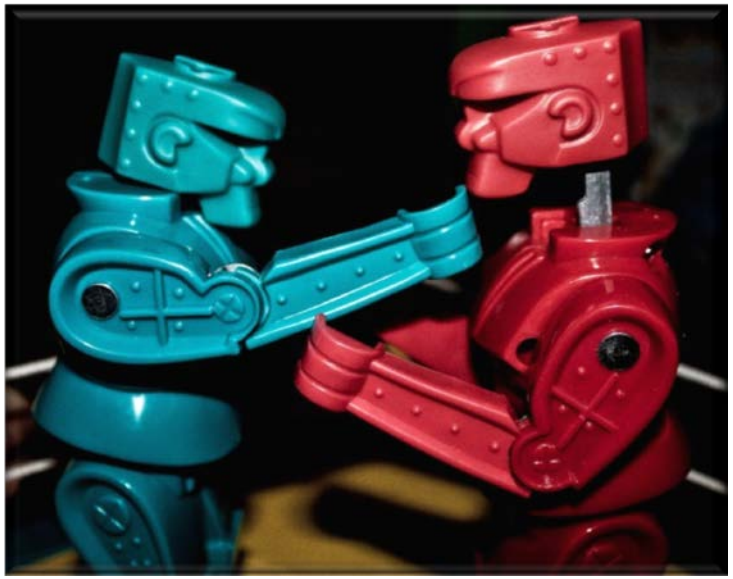
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:



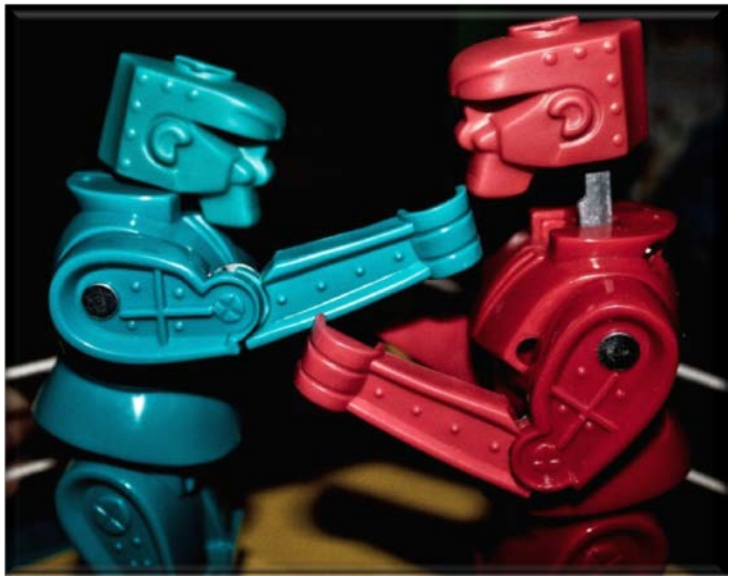
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks



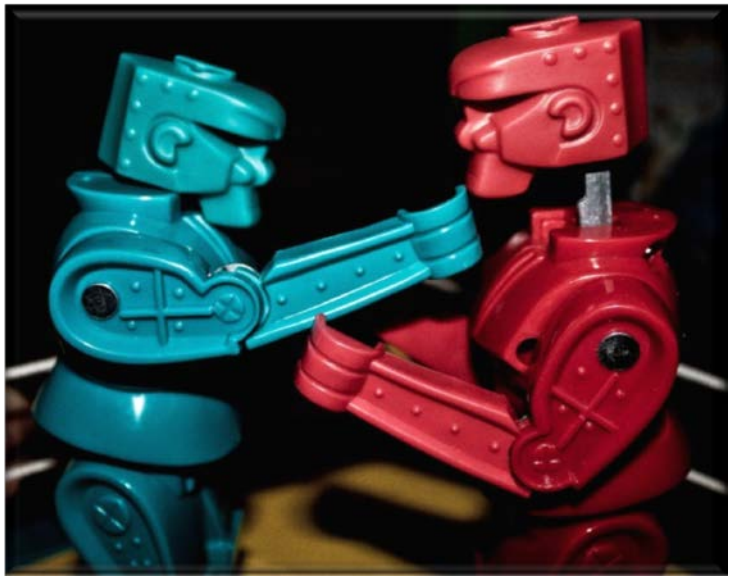
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks
    - But ReentrantReadWriteLocks may enable more concurrency on multi-core or multi-processor hardware



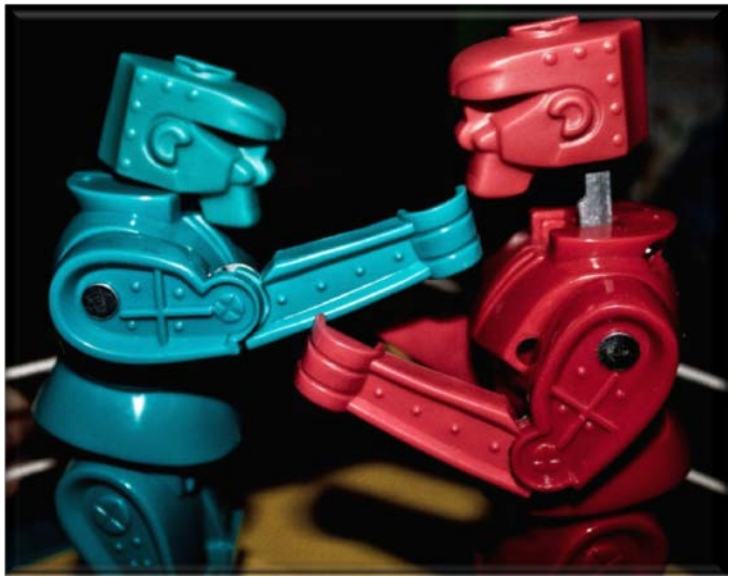
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks
  - ConditionObjects & Semaphores have higher overhead than ReentrantLocks & Reentrant ReadWriteLocks



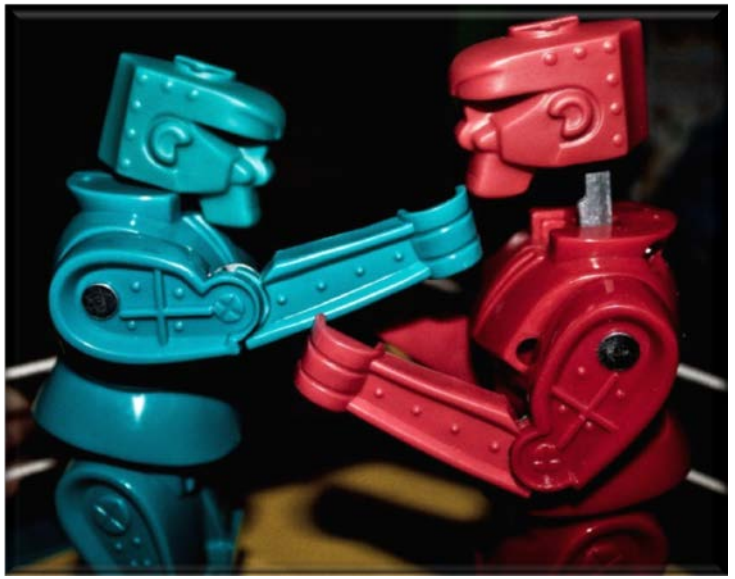
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks
  - ConditionObjects & Semaphores have higher overhead than ReentrantLocks & Reentrant ReadWriteLocks
    - But they are also more expressive & more flexible



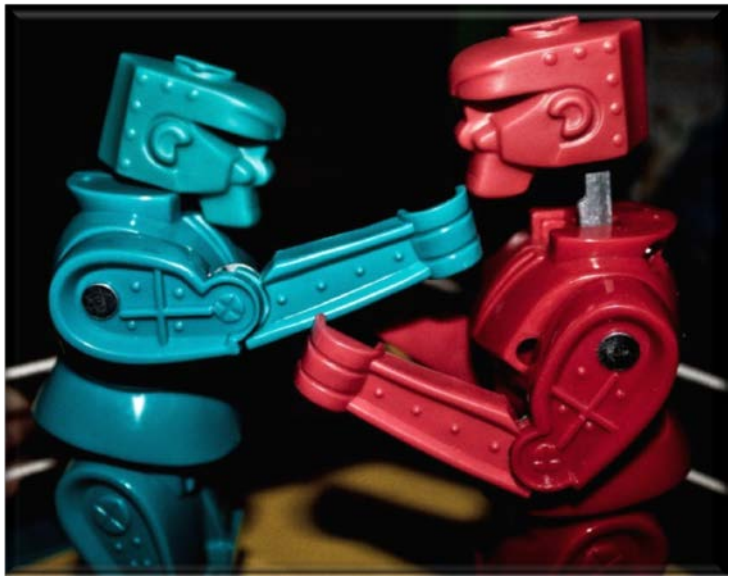
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks
  - ConditionObjects & Semaphores have higher overhead than ReentrantLocks & Reentrant ReadWriteLocks
    - But they are also more expressive & more flexible
    - e.g., they allow threads to coordinate their interactions



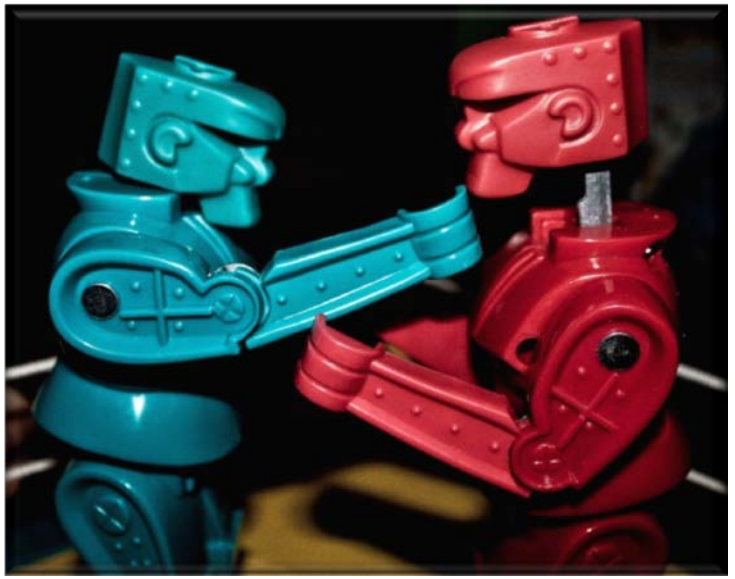
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks
  - ConditionObjects & Semaphores have higher overhead than ReentrantLocks & Reentrant ReadWriteLocks
  - ConditionObjects have a different purpose than other locks



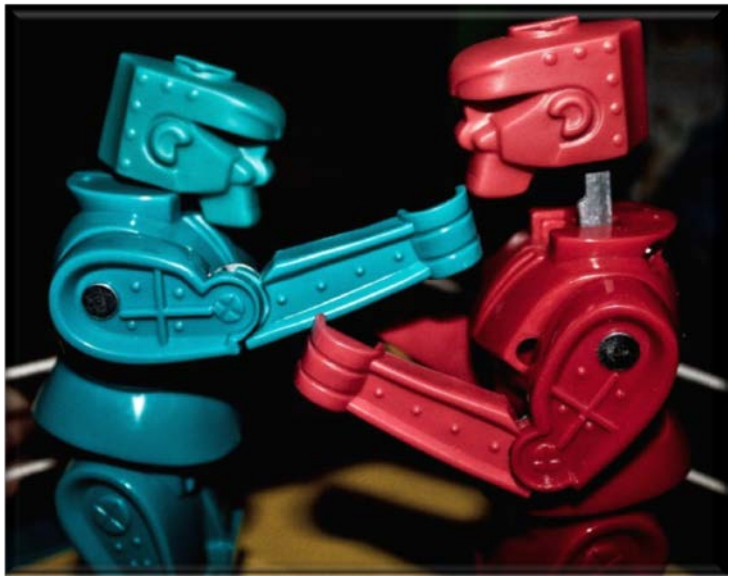
Performance & Usability Considerations

- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks
  - ConditionObjects & Semaphores have higher overhead than ReentrantLocks & Reentrant ReadWriteLocks
  - ConditionObjects have a different purpose than other locks
    - A thread uses a lock to keep other threads out of a critical section



Performance & Usability Considerations

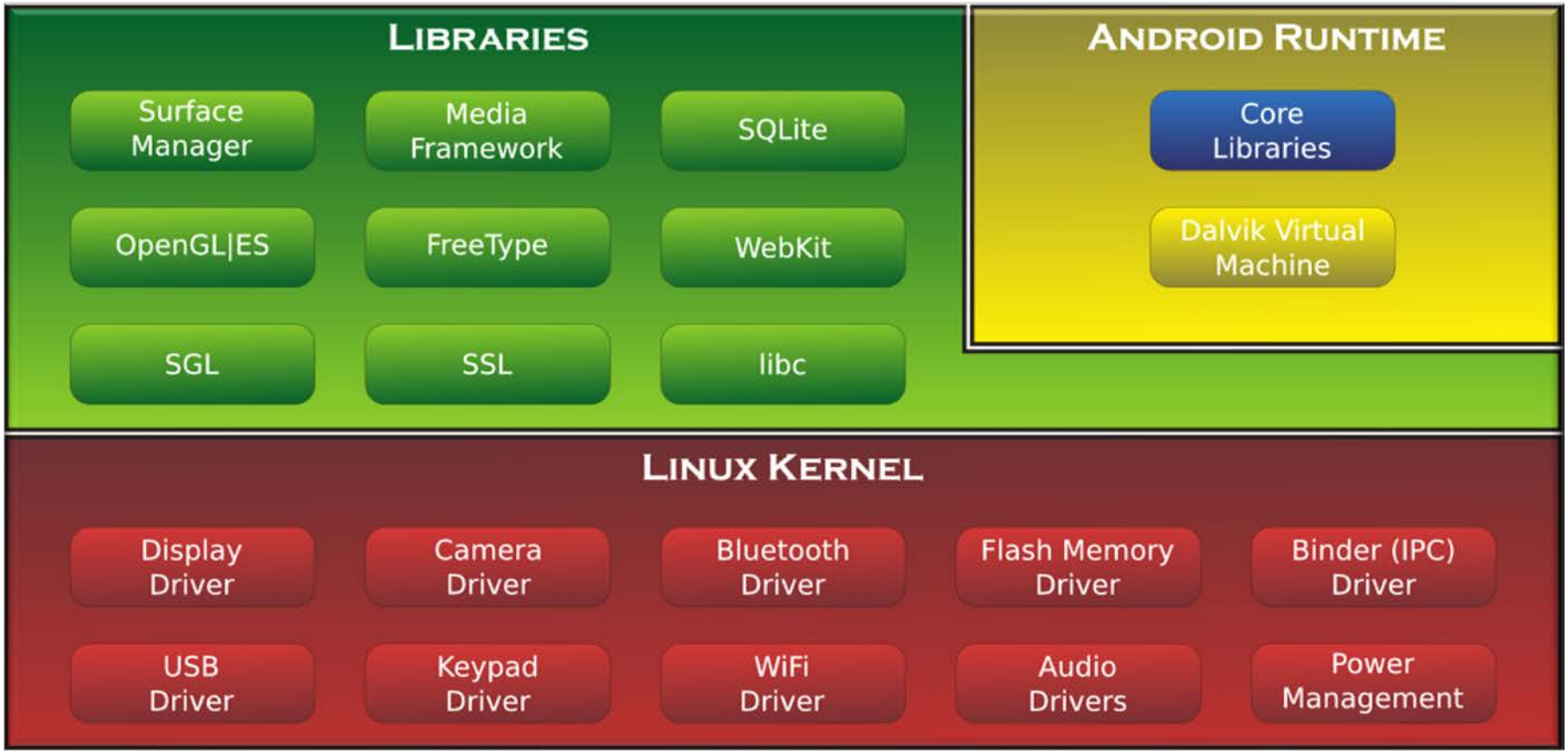
- The performance of these classes depends on details of the Java virtual machine, operating system, hardware, & program use cases
- Some issues to consider when choosing between classes:
  - ReentrantLocks have lower overhead than ReentrantReadWriteLocks
  - ConditionObjects & Semaphores have higher overhead than ReentrantLocks & Reentrant ReadWriteLocks
  - ConditionObjects have a different purpose than other locks
    - A thread uses a lock to keep other threads out of a critical section
    - A thread uses a ConditionObject to keep itself out of a critical section until it can make forward progress



Summary

Summary

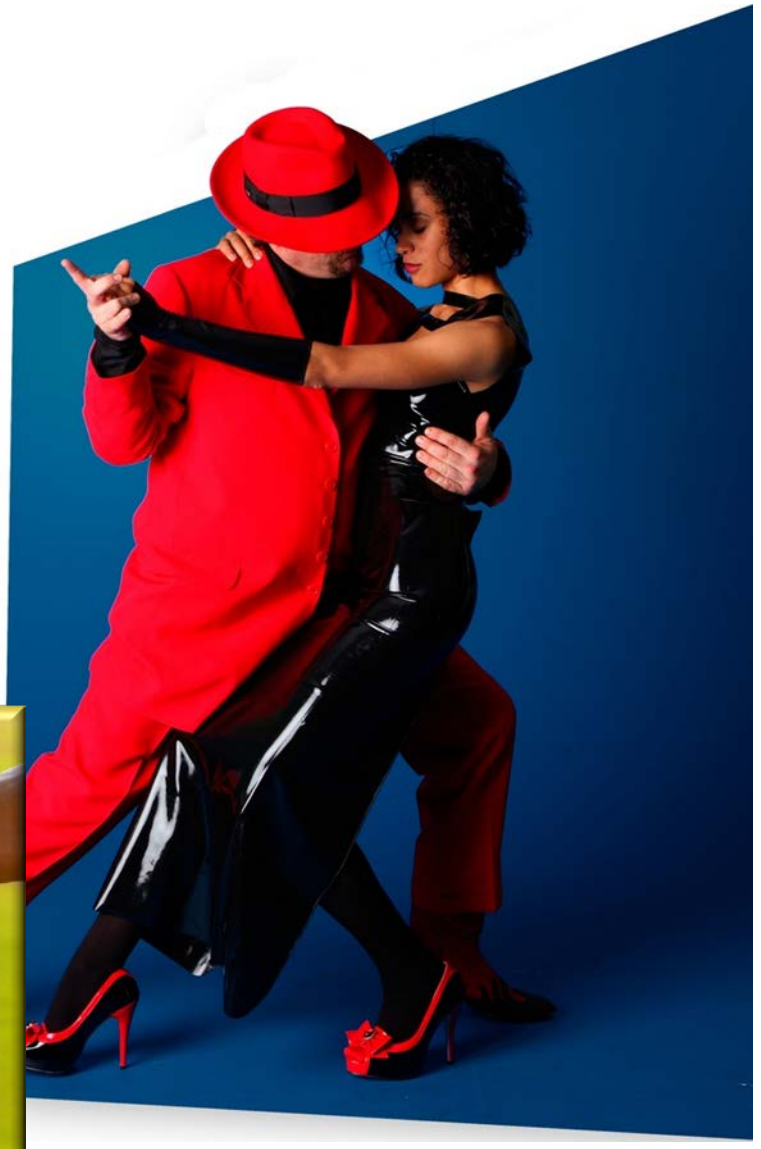
- All these Java synchronization & scheduling classes are used throughout Android



Download the Android source code from source.android.com

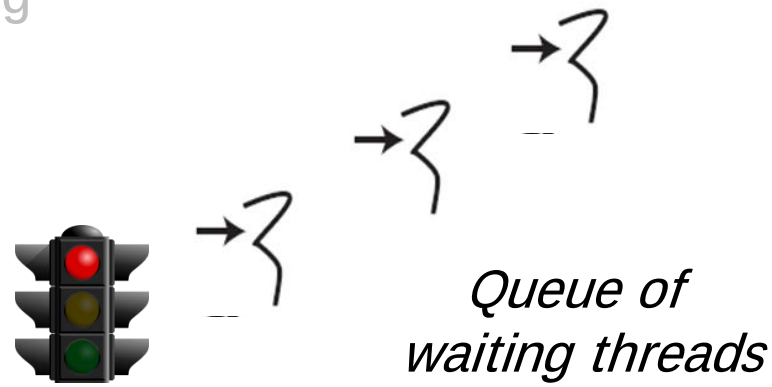
Summary

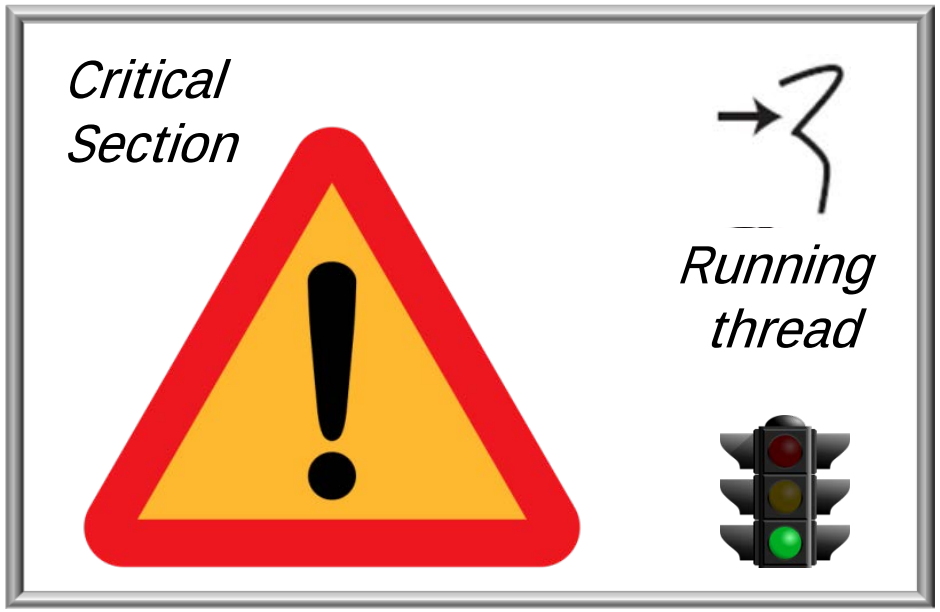
- All these Java synchronization & scheduling classes are used throughout Android
- Concurrent programs use these classes for several reasons



Summary

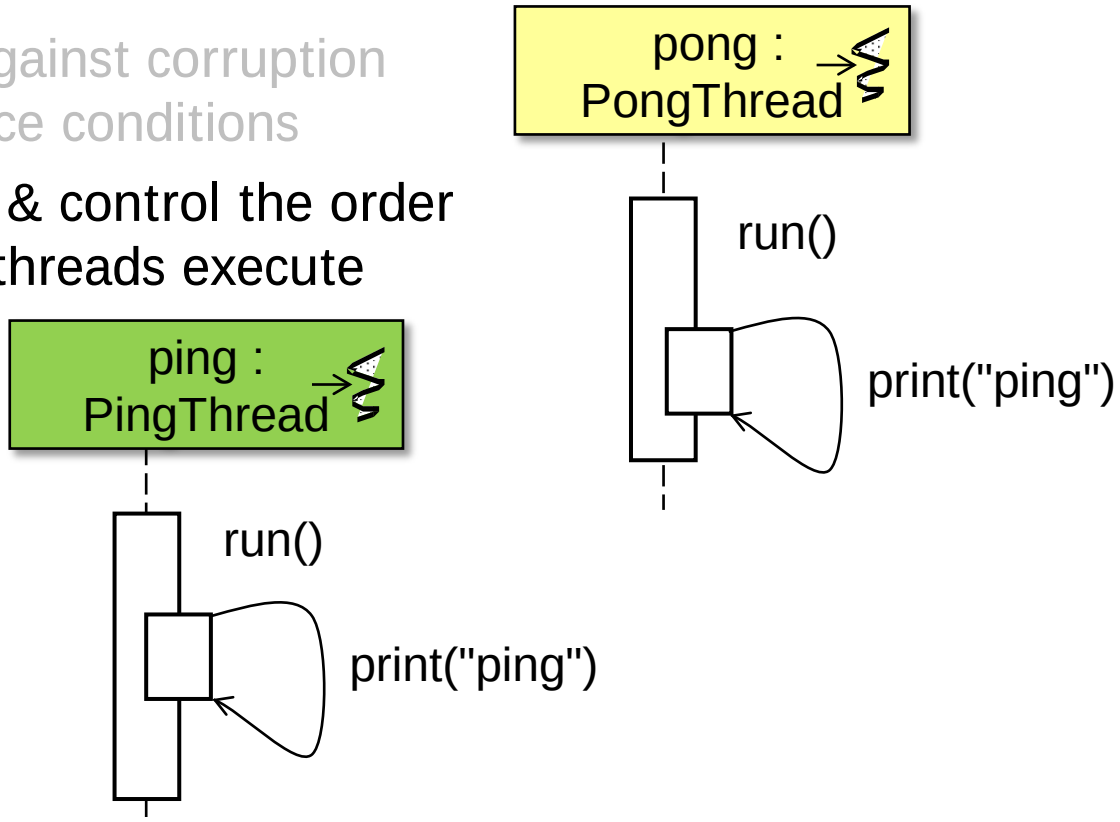
- All these Java synchronization & scheduling classes are used throughout Android
- Concurrent programs use these classes for several reasons
  - Protect against corruption due to race conditions





Summary

- All these Java synchronization & scheduling classes are used throughout Android
- Concurrent programs use these classes for several reasons
  - Protect against corruption due to race conditions
  - Schedule & control the order in which threads execute




```
% java PingPong
Ready...Set...Go!
Ping!(1)
Pong!(1)
Ping!(2)
Pong!(2)
Ping!(3)
Pong!(3)
Ping!(4)
Pong!(4)
Ping!(5)
Pong!(5)
Ping!(6)
Pong!(6)
Ping!(7)
Pong!(7)
Ping!(8)
Pong!(8)
Ping!(9)
Pong!(9)
Ping!(10)
Pong!(10)
Done!
```

Summary

- All these Java synchronization & scheduling classes are used throughout Android
- Concurrent programs use these classes for several reasons
- We devote more time to Java synchronization & scheduling mechanisms than to Java threading mechanisms

Thread coverage

Synchronization & scheduling coverage

Summary

- All these Java synchronization & scheduling classes are used throughout Android
- Concurrent programs use these classes for several reasons
- We devote more time to Java synchronization & scheduling mechanisms than to Java threading mechanisms
  - Complexity arises largely from coordinating interactions of multiple entities that run concurrently

Summary

- All these Java synchronization & scheduling classes are used throughout Android
- Concurrent programs use these classes for several reasons
- We devote more time to Java synchronization & scheduling mechanisms than to Java threading mechanisms
- Each Java synchronization & scheduling class is covered systematically

Java Class	Purpose
Reentrant Lock	A reentrant mutual exclusion lock that extends the built-in monitor lock capabilities
Reentrant Read WriteLock	Improves performance when resources are read much more often than written
Semaphore	A non-negative integer that controls the access of multiple threads to a limited number of shared resources
Condition Object	Block thread(s) until some condition(s) becomes true
CountDown Latch	Allows one or more threads to wait until a set of operations being performed in other threads complete

Summary

- All these Java synchronization & scheduling classes are used throughout Android
- Concurrent programs use these classes for several reasons
- We devote more time to Java synchronization & scheduling mechanisms than to Java threading mechanisms
- Each Java synchronization & scheduling class is covered systematically

We'll show how these classes are implemented & used in Android