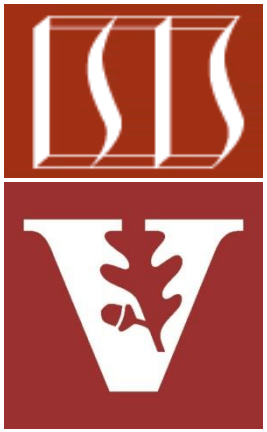
Android Concurrency: Overview of Java Threads (Part 2)

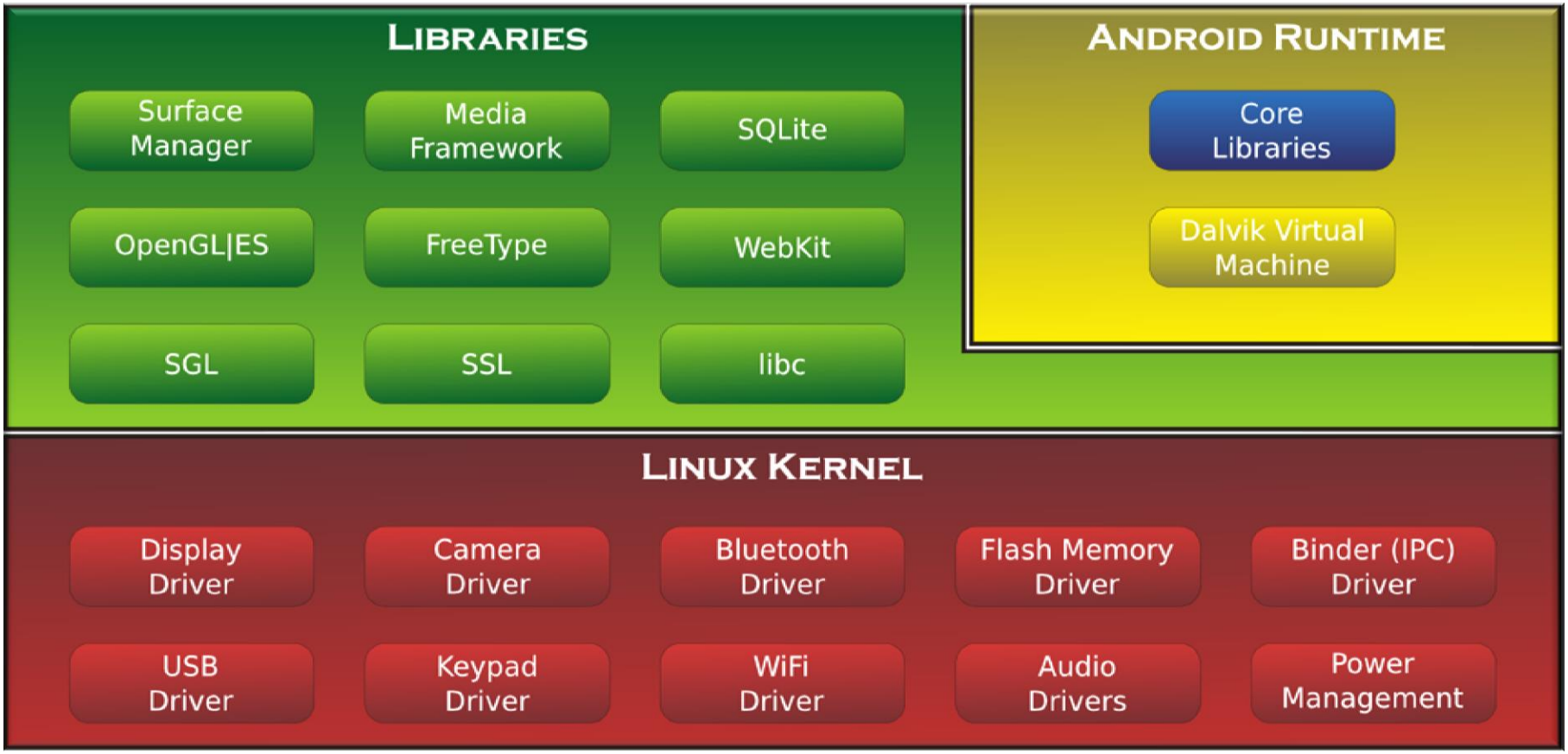


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

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

Learning Objectives in this Part of the Module

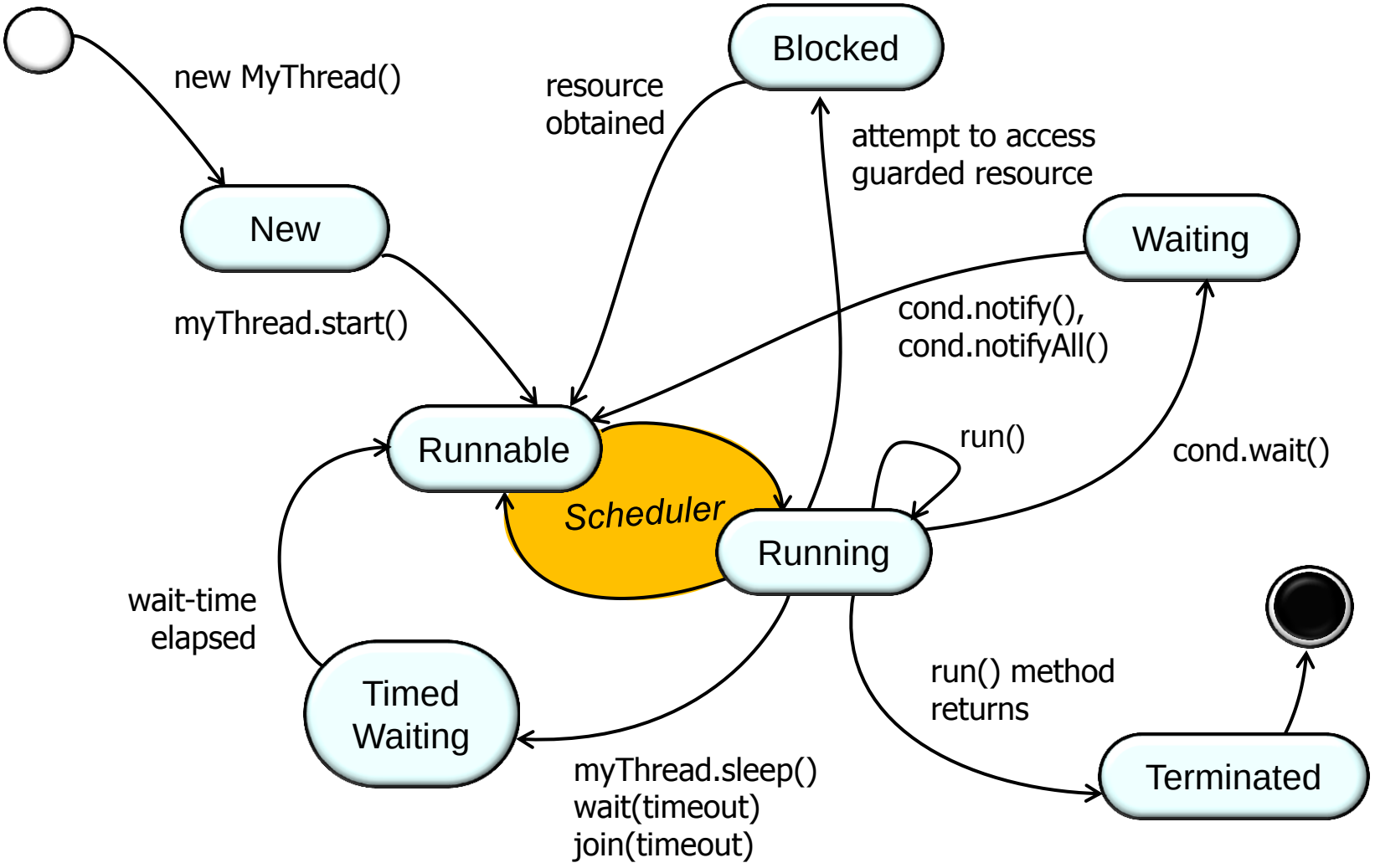
- Understand more about how Java concurrency mechanisms available in Android are implemented



Learning Objectives in this Part of the Module

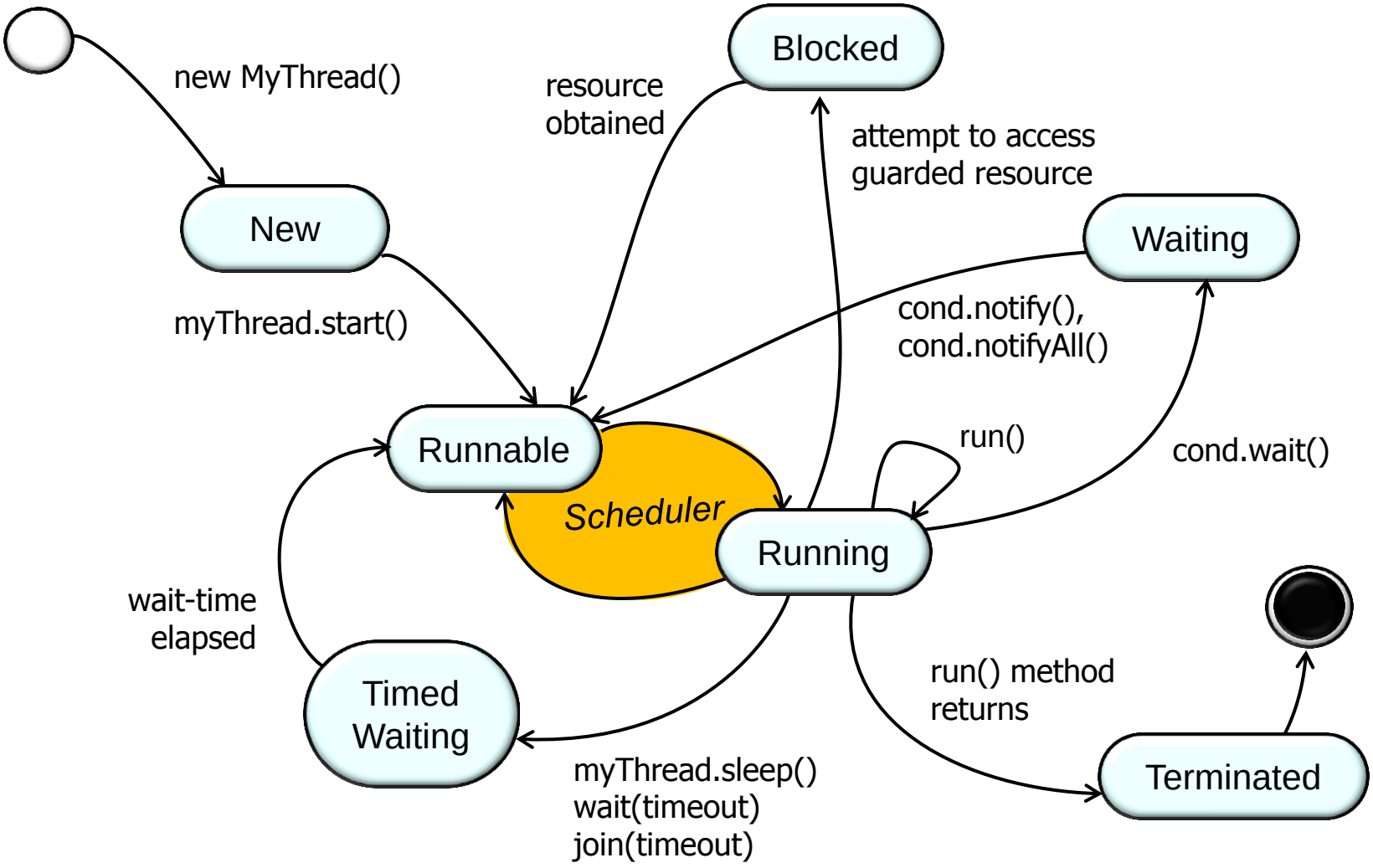
- Understand more about how Java concurrency mechanisms available in Android are implemented

State Machine for Java Threads in Android



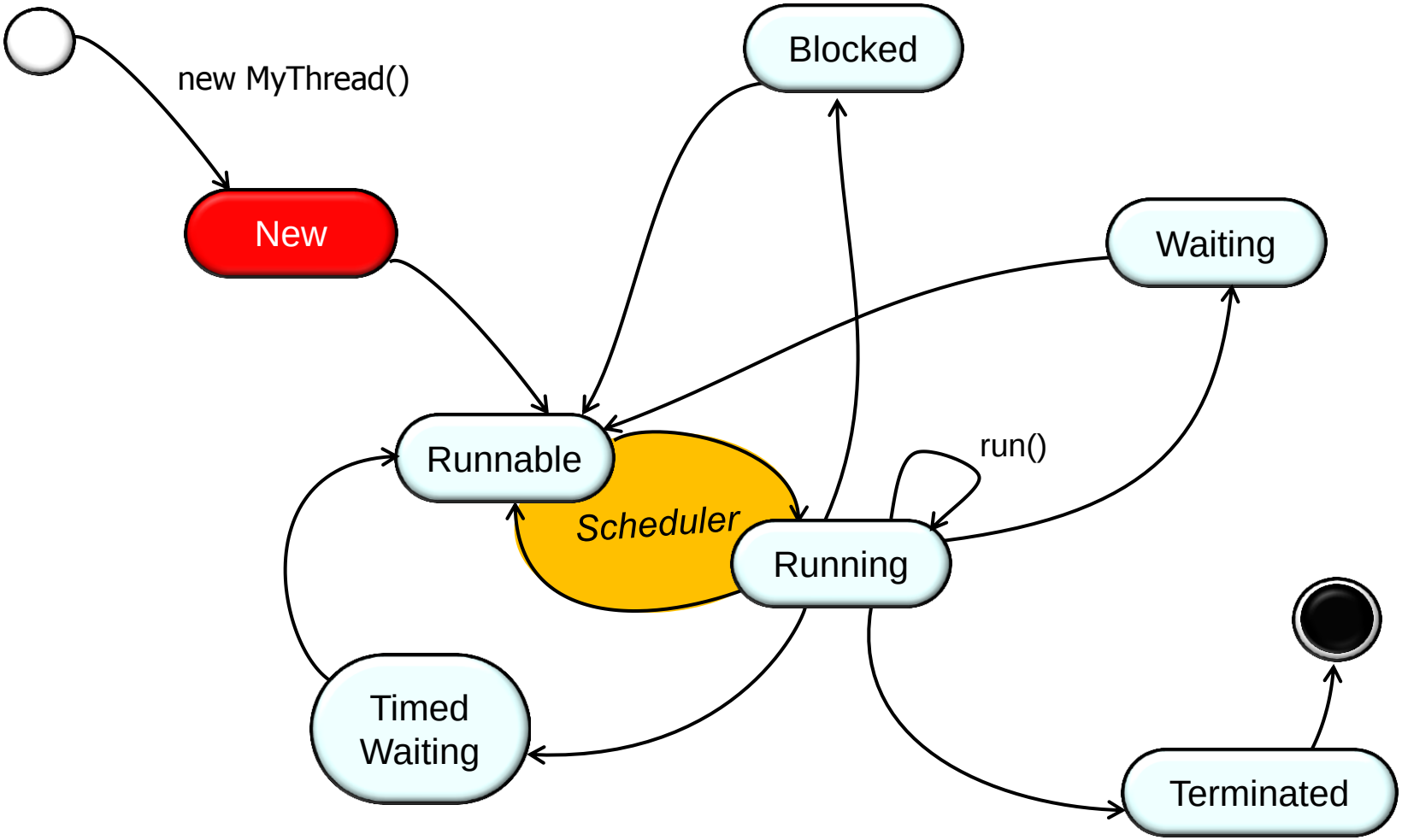
docs.oracle.com/javase/7/docs/api/java/lang/Thread.State.html has more

State Machine for Java Threads in Android

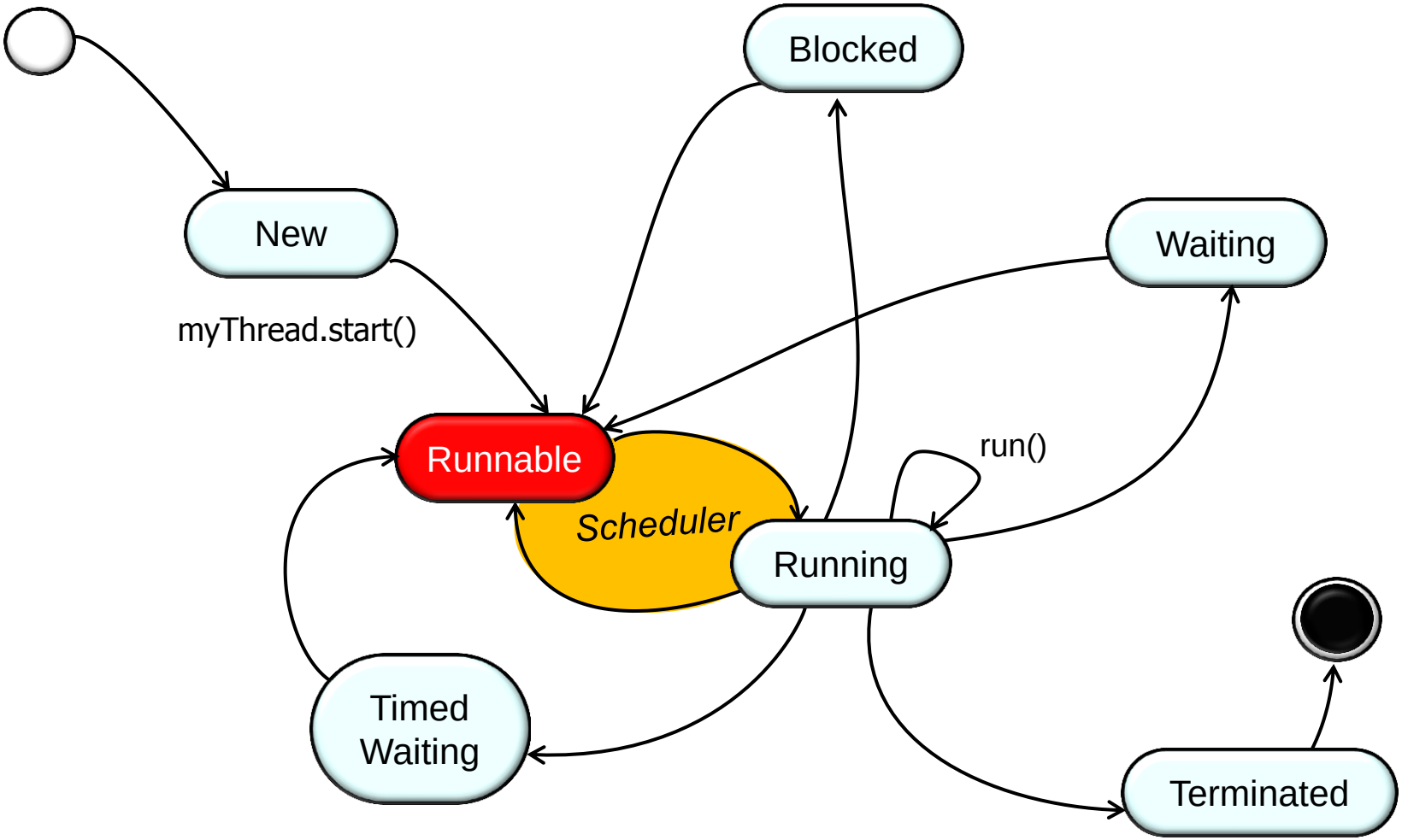


You don't need to understand all these details to program Java threads!!

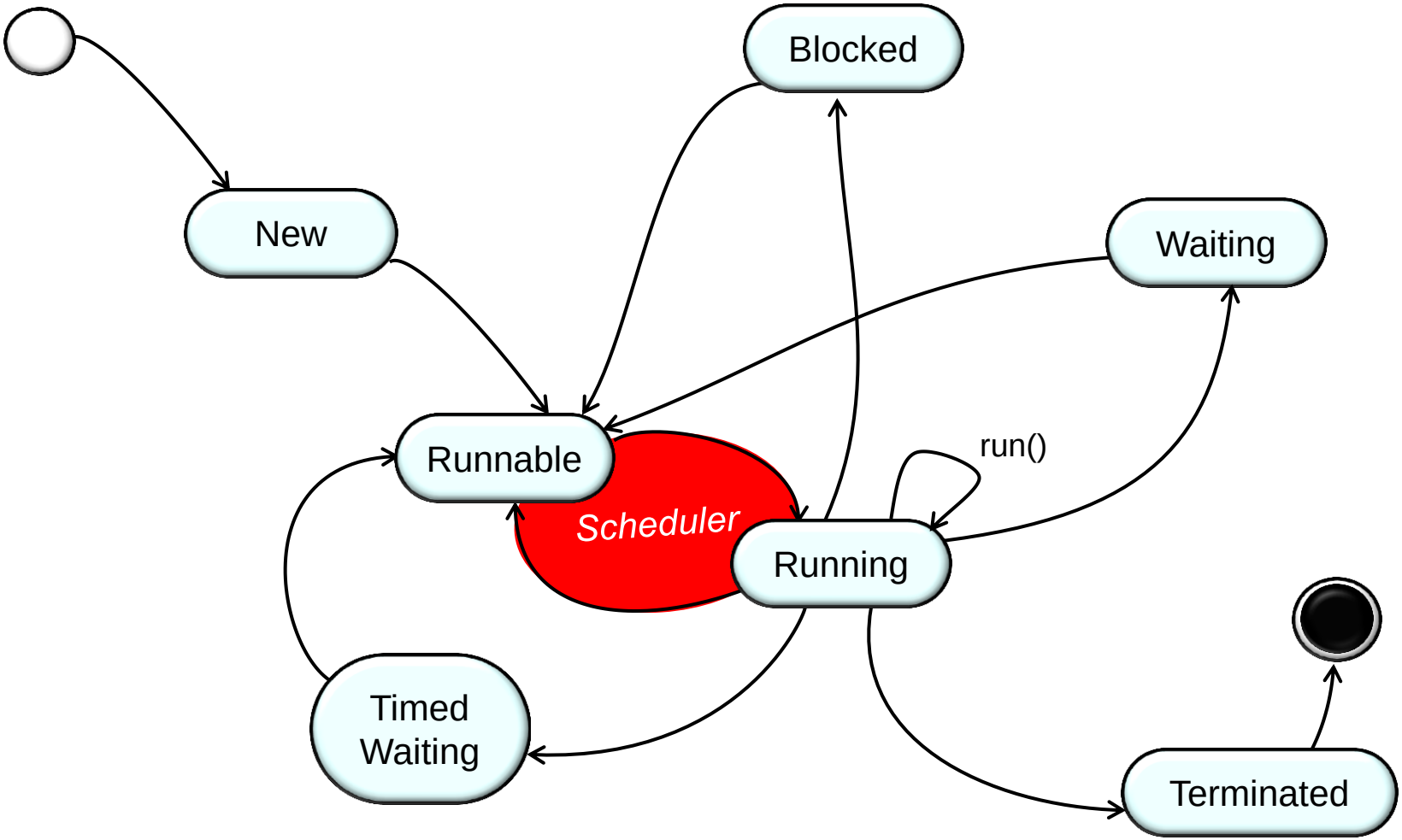
State Machine for Java Threads in Android



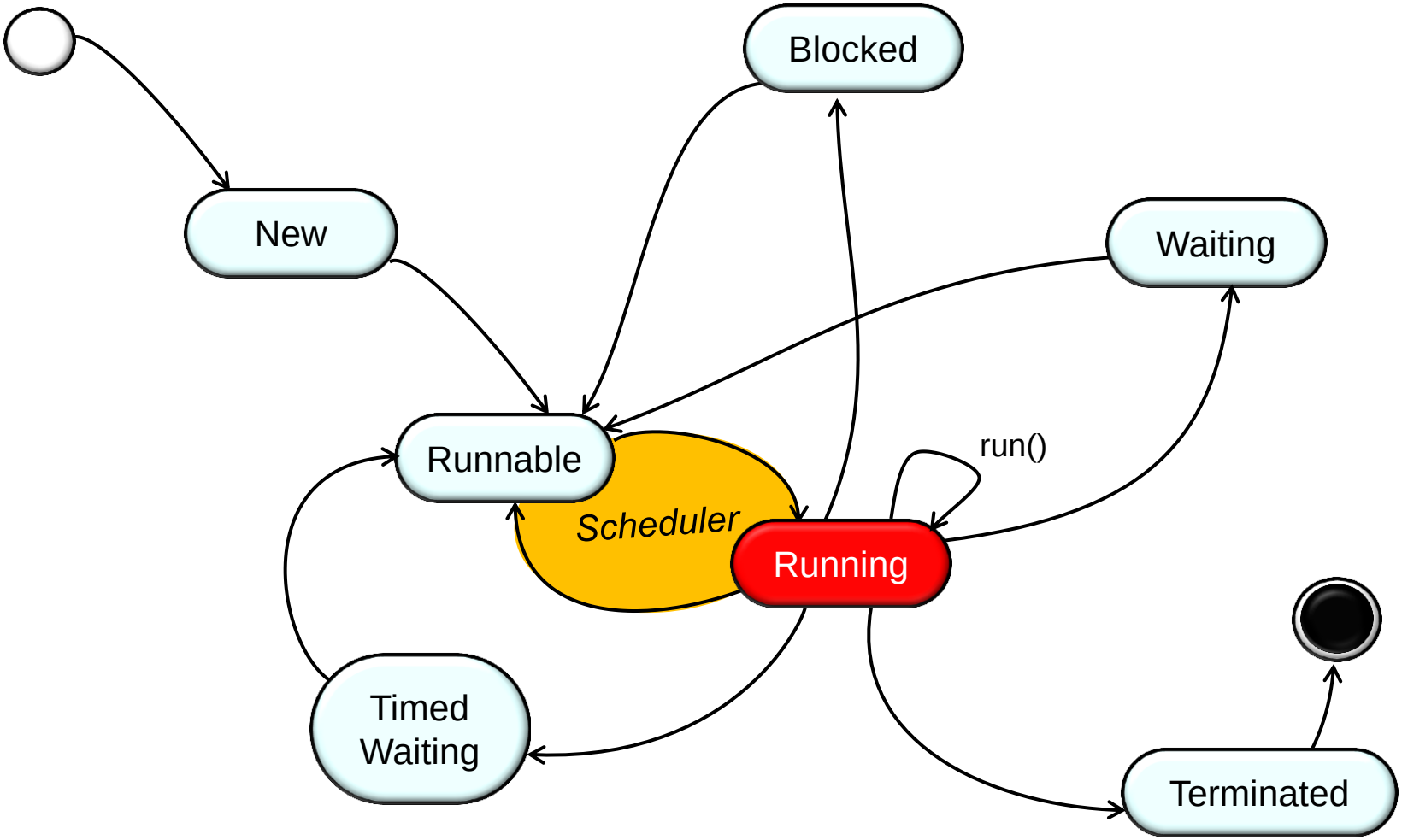
State Machine for Java Threads in Android



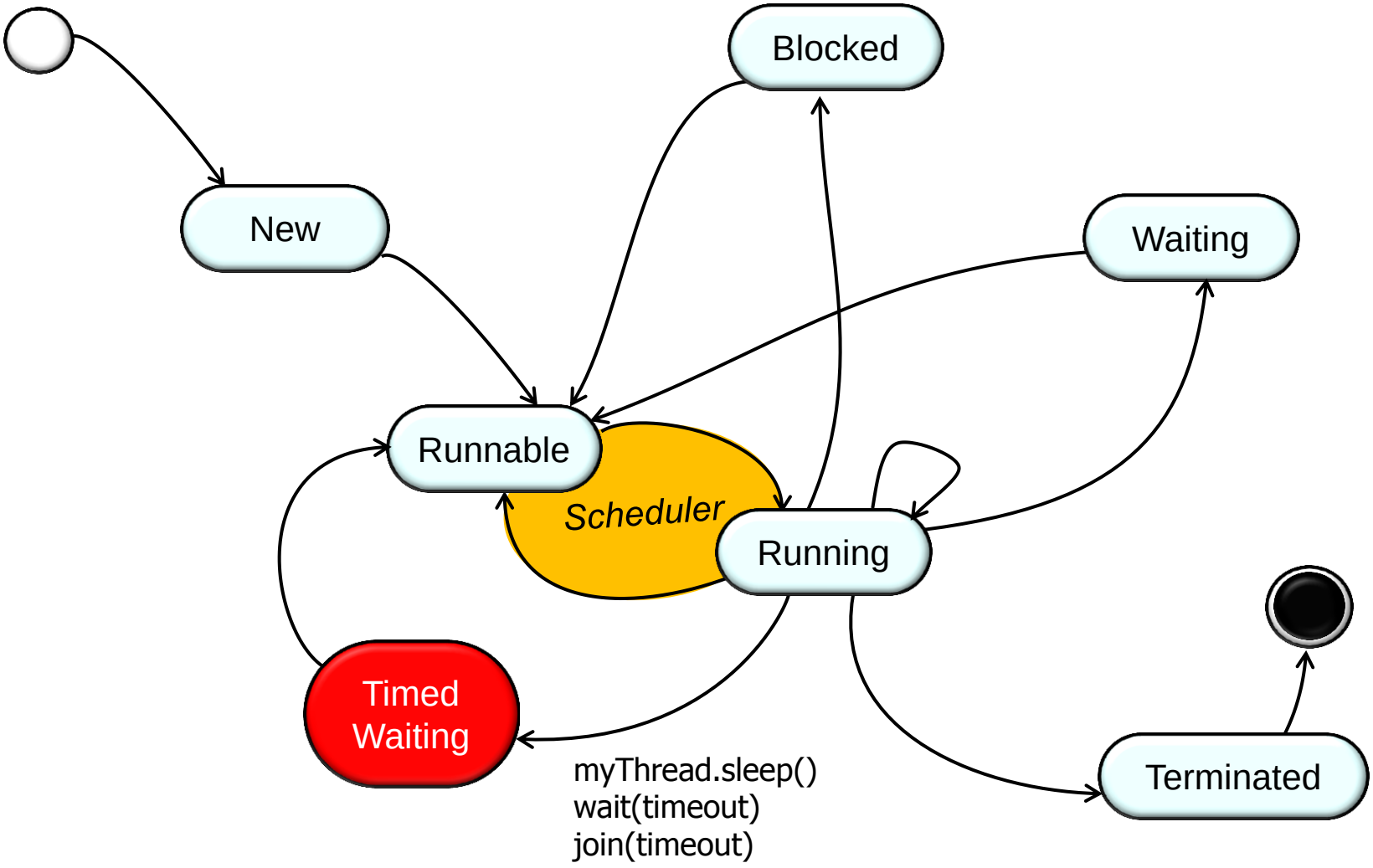
State Machine for Java Threads in Android



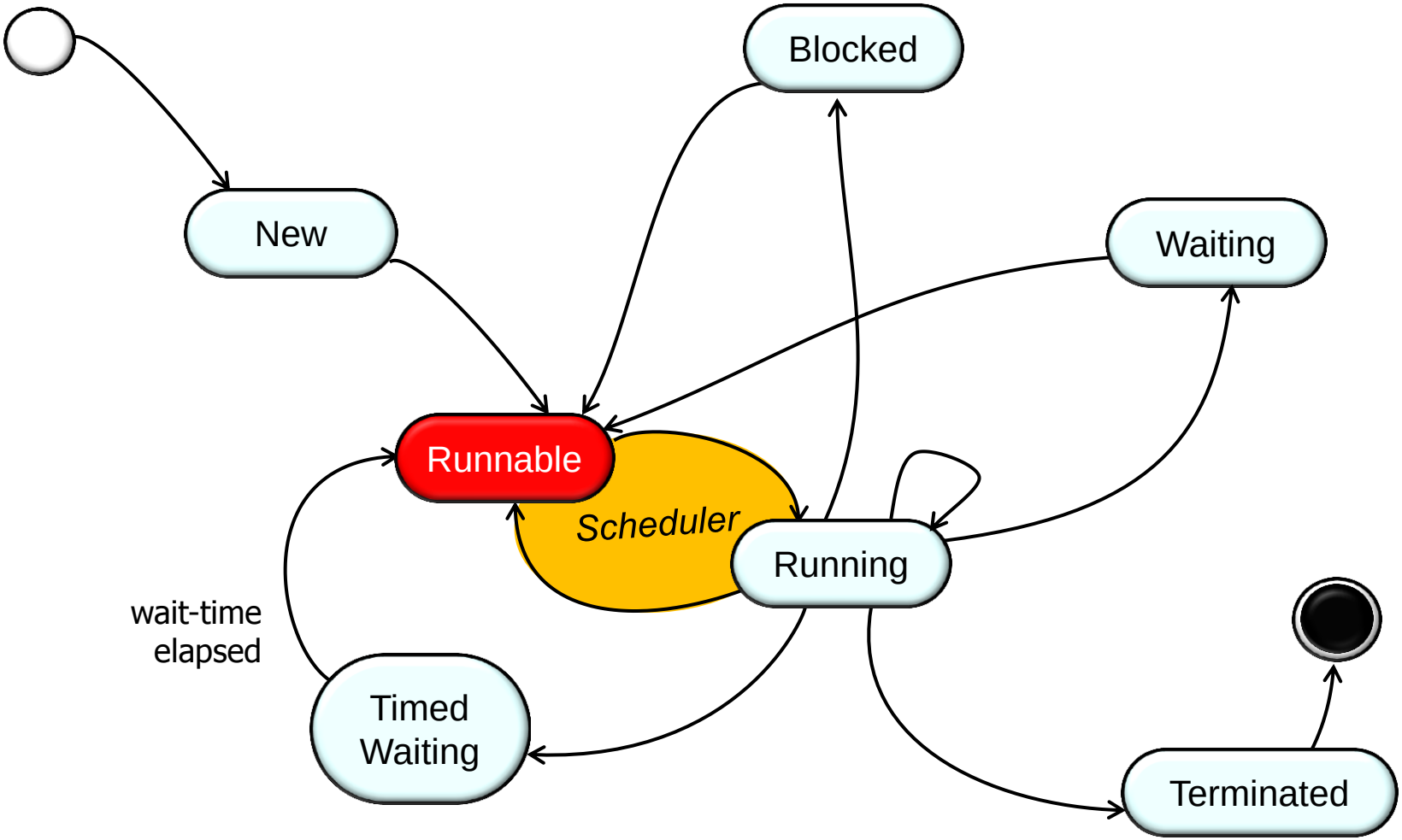
State Machine for Java Threads in Android



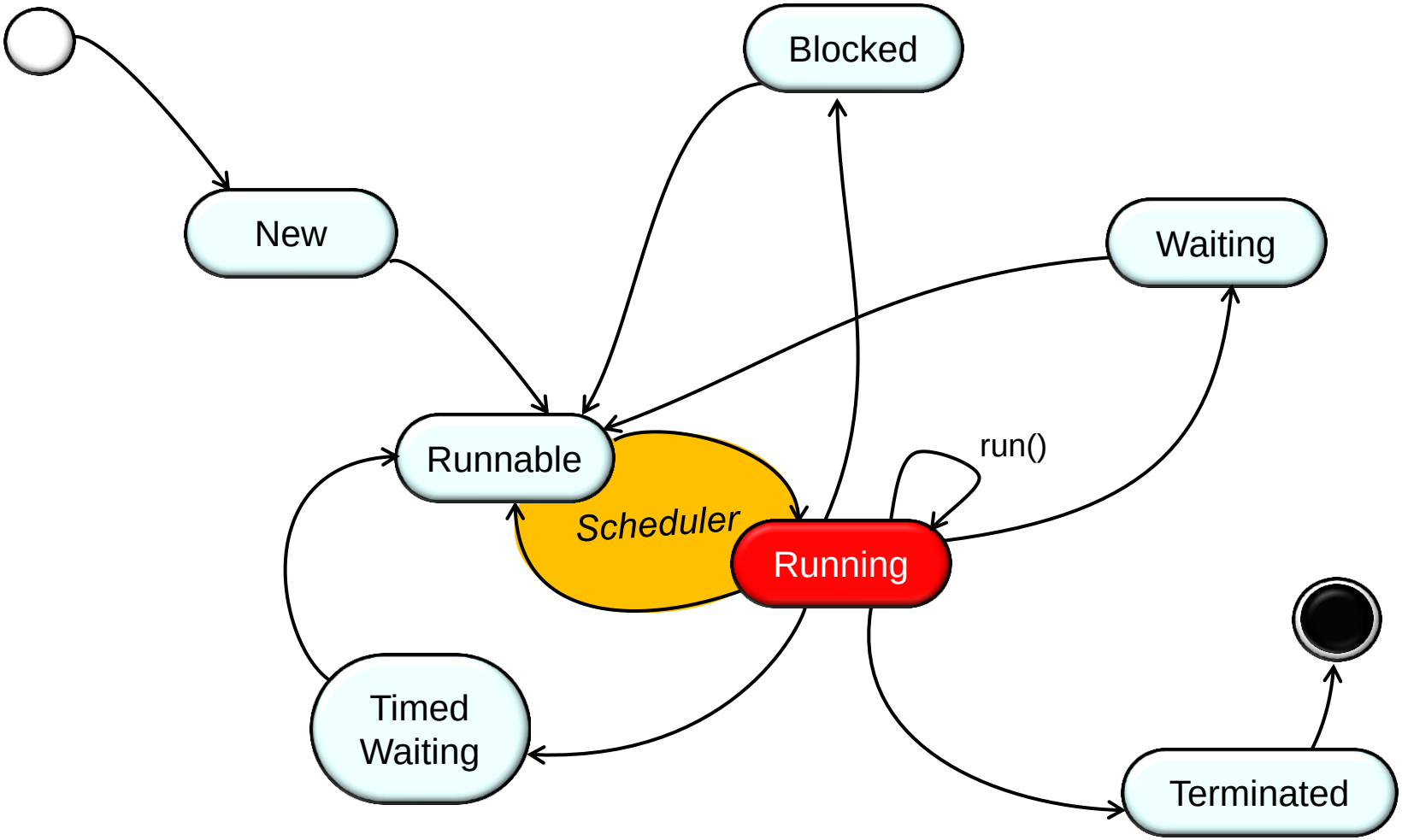
State Machine for Java Threads in Android



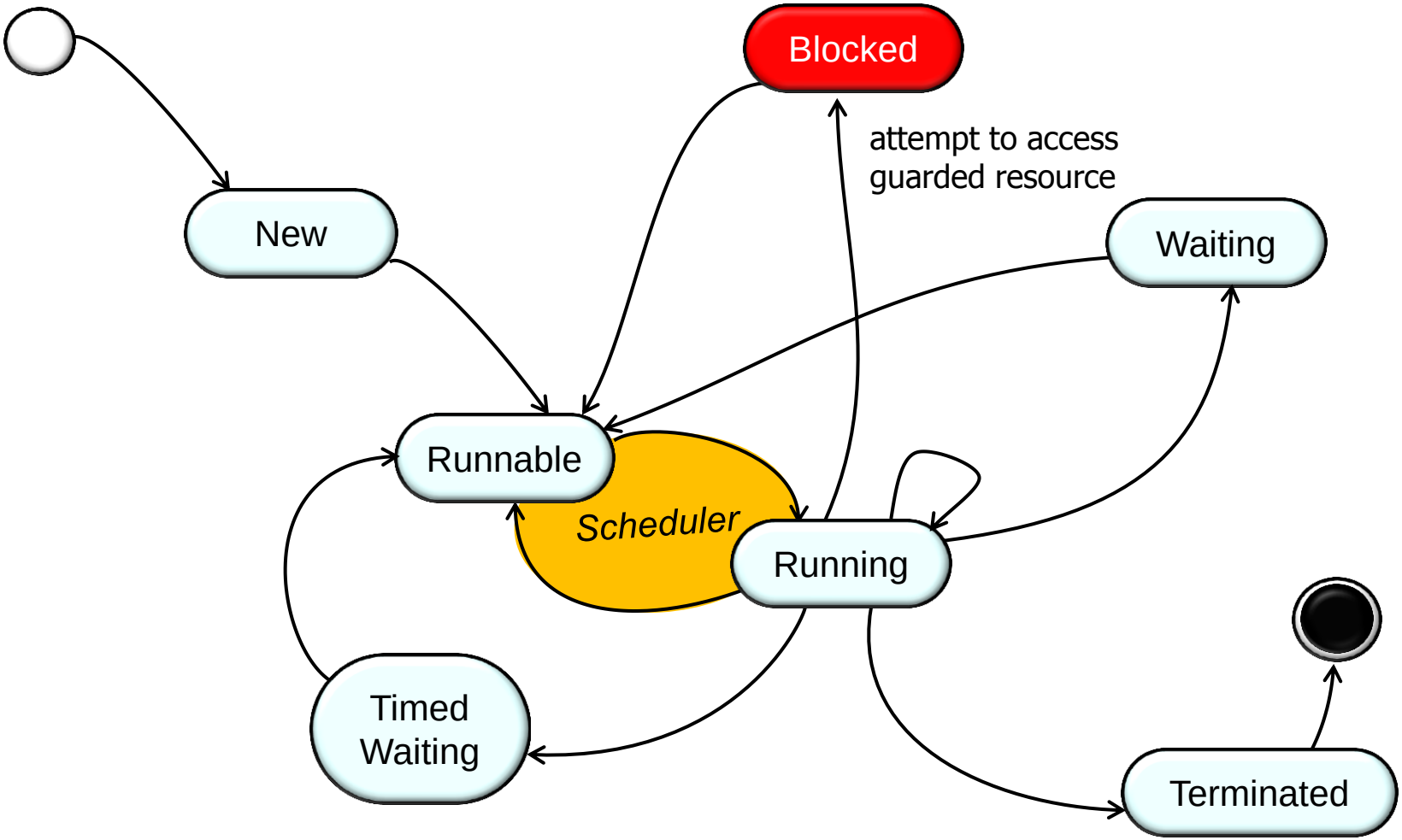
State Machine for Java Threads in Android



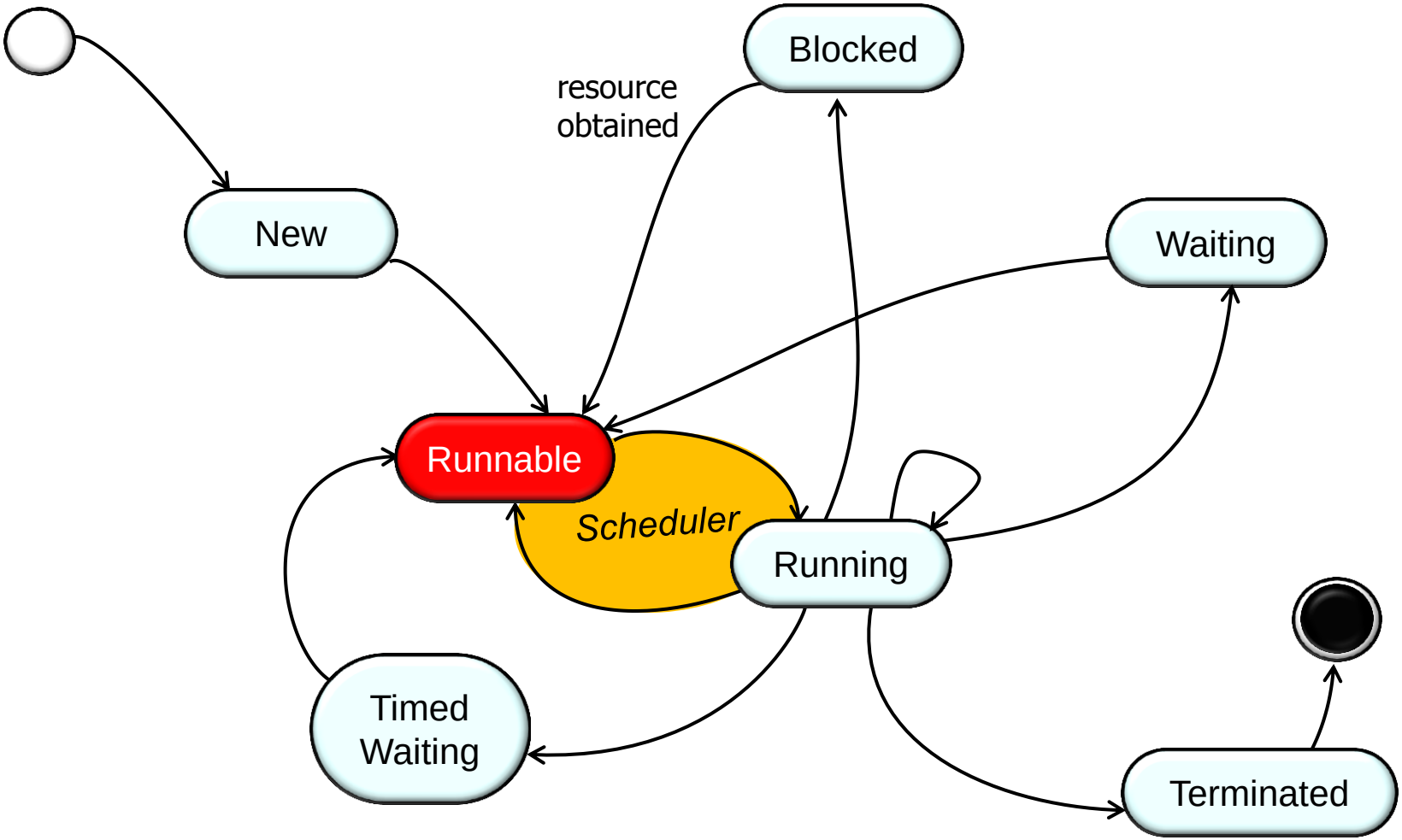
State Machine for Java Threads in Android



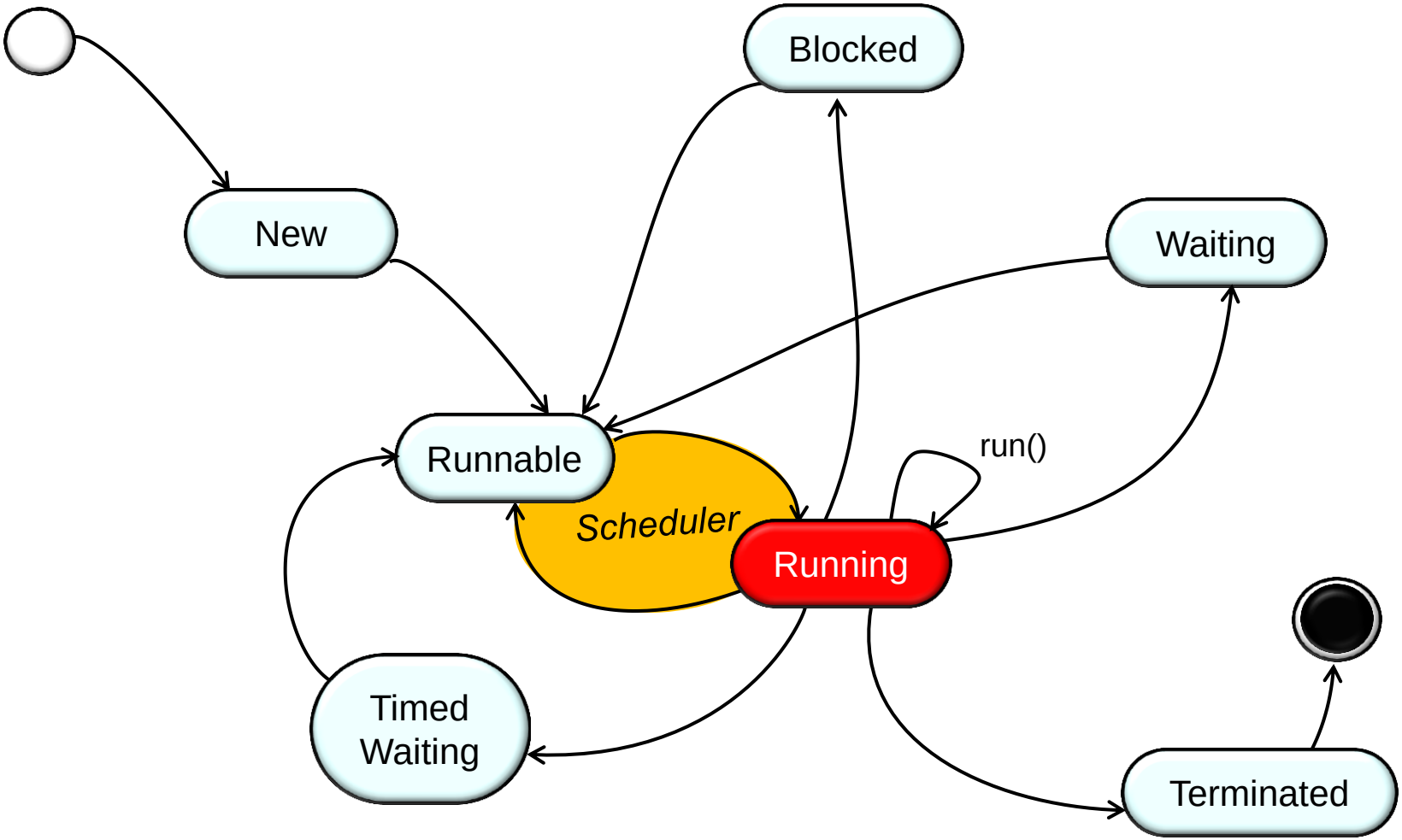
State Machine for Java Threads in Android



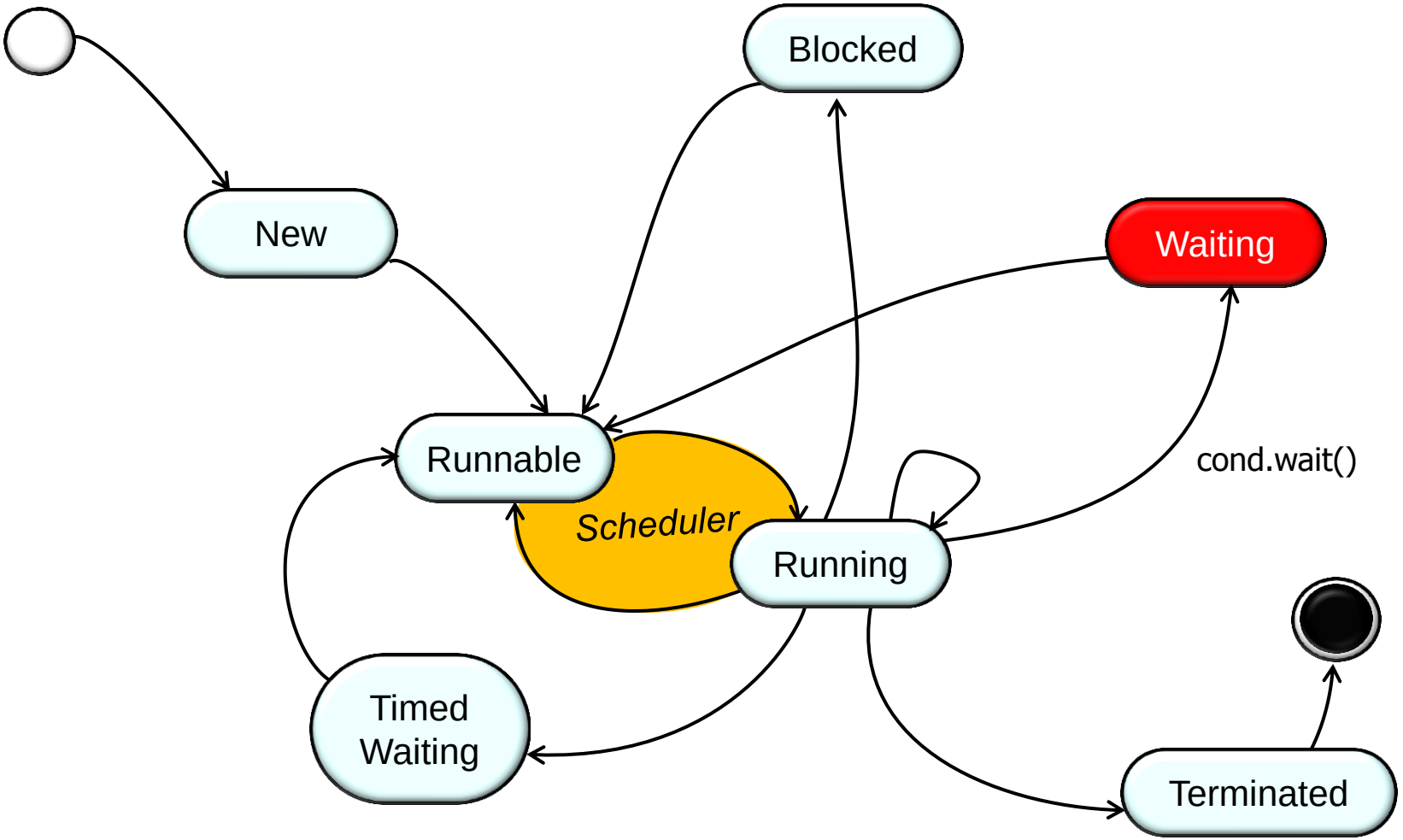
State Machine for Java Threads in Android



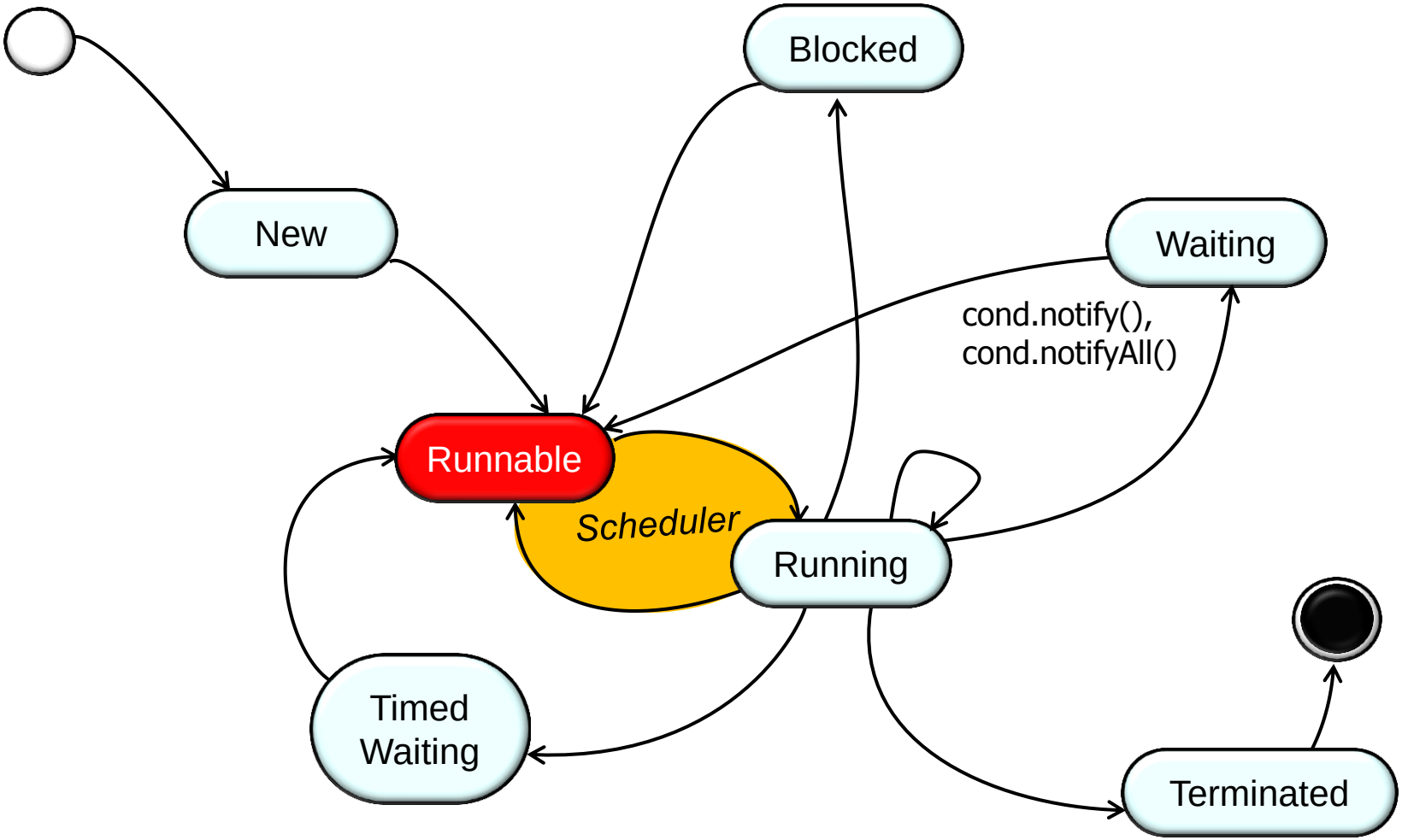
State Machine for Java Threads in Android



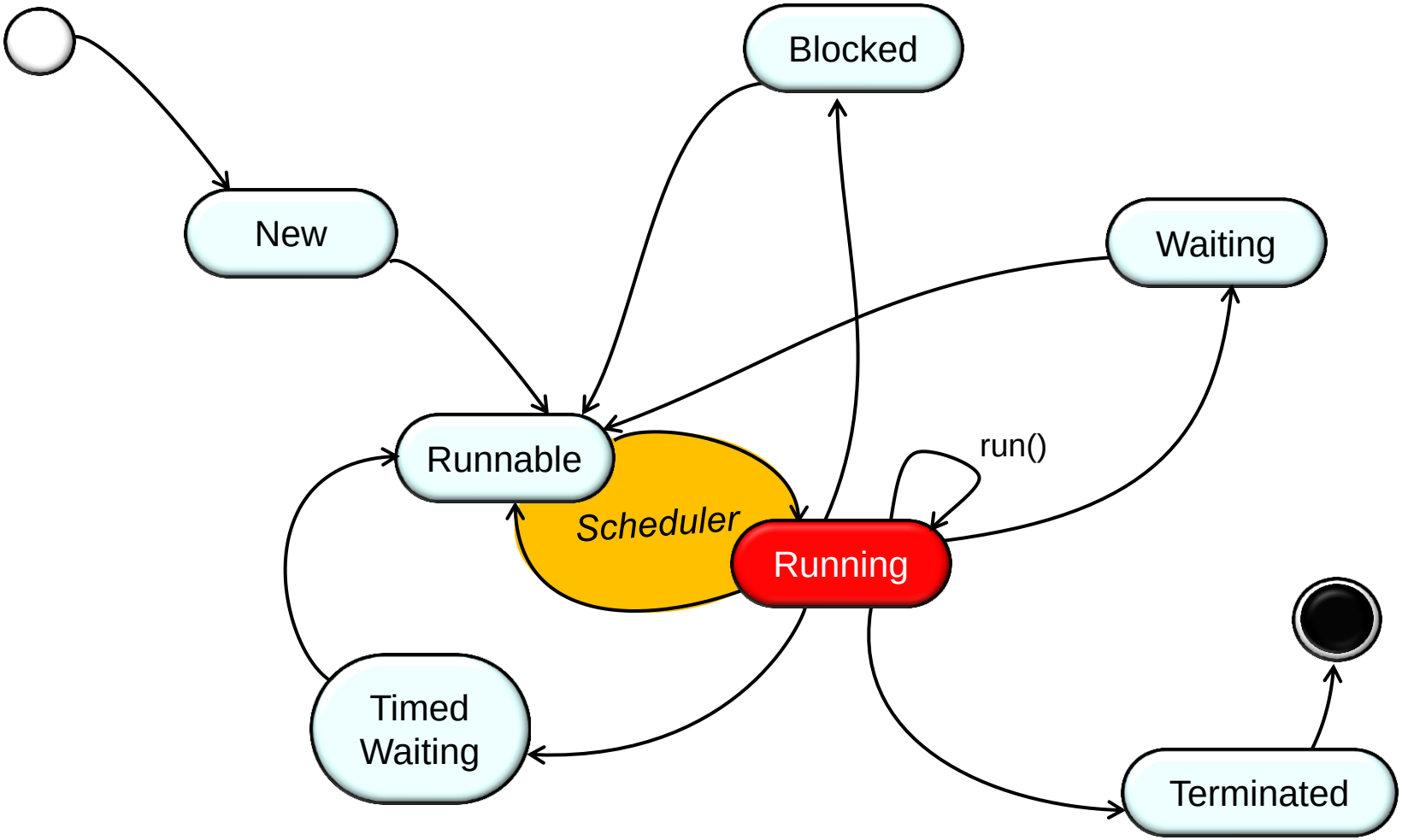
State Machine for Java Threads in Android



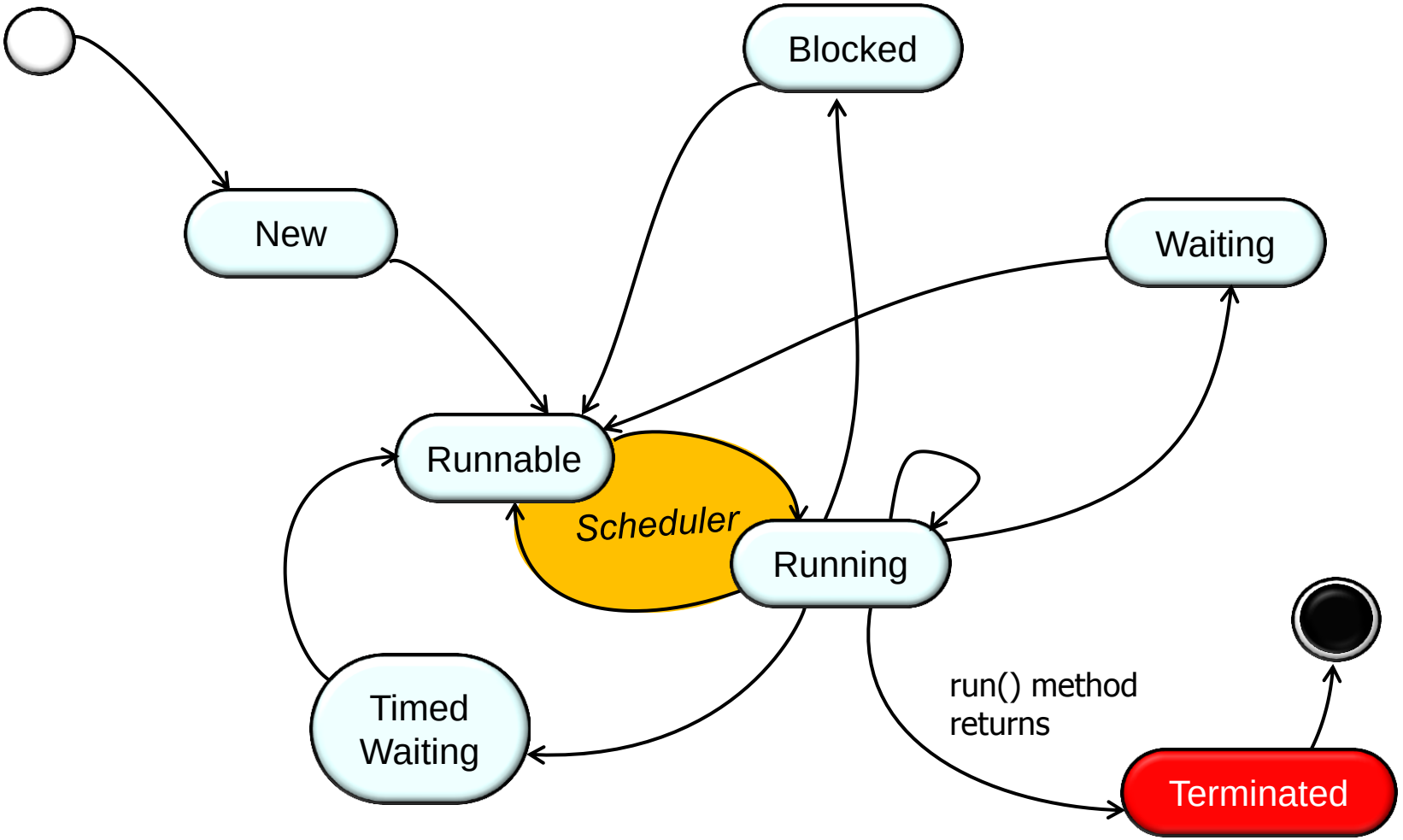
State Machine for Java Threads in Android



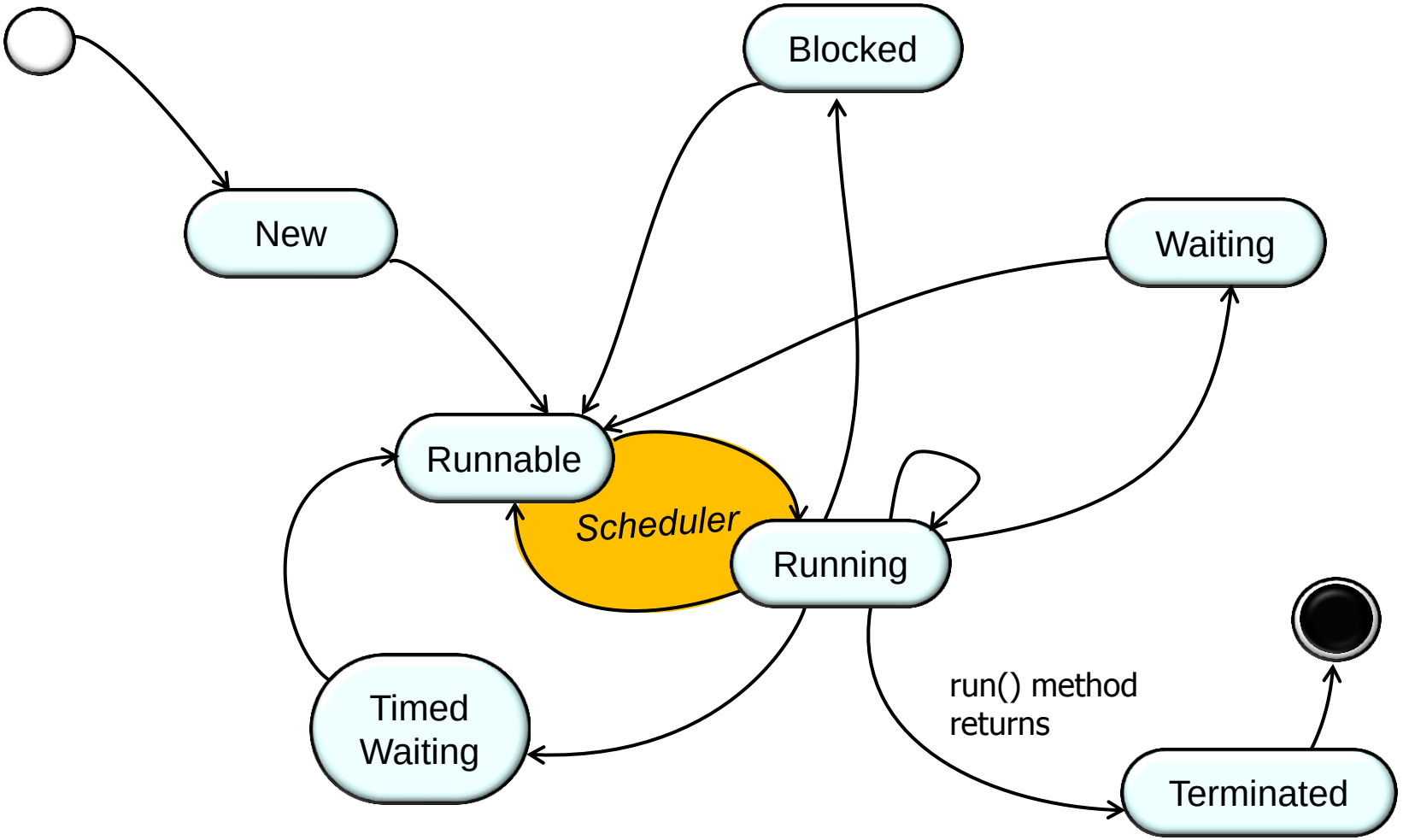
State Machine for Java Threads in Android



State Machine for Java Threads in Android



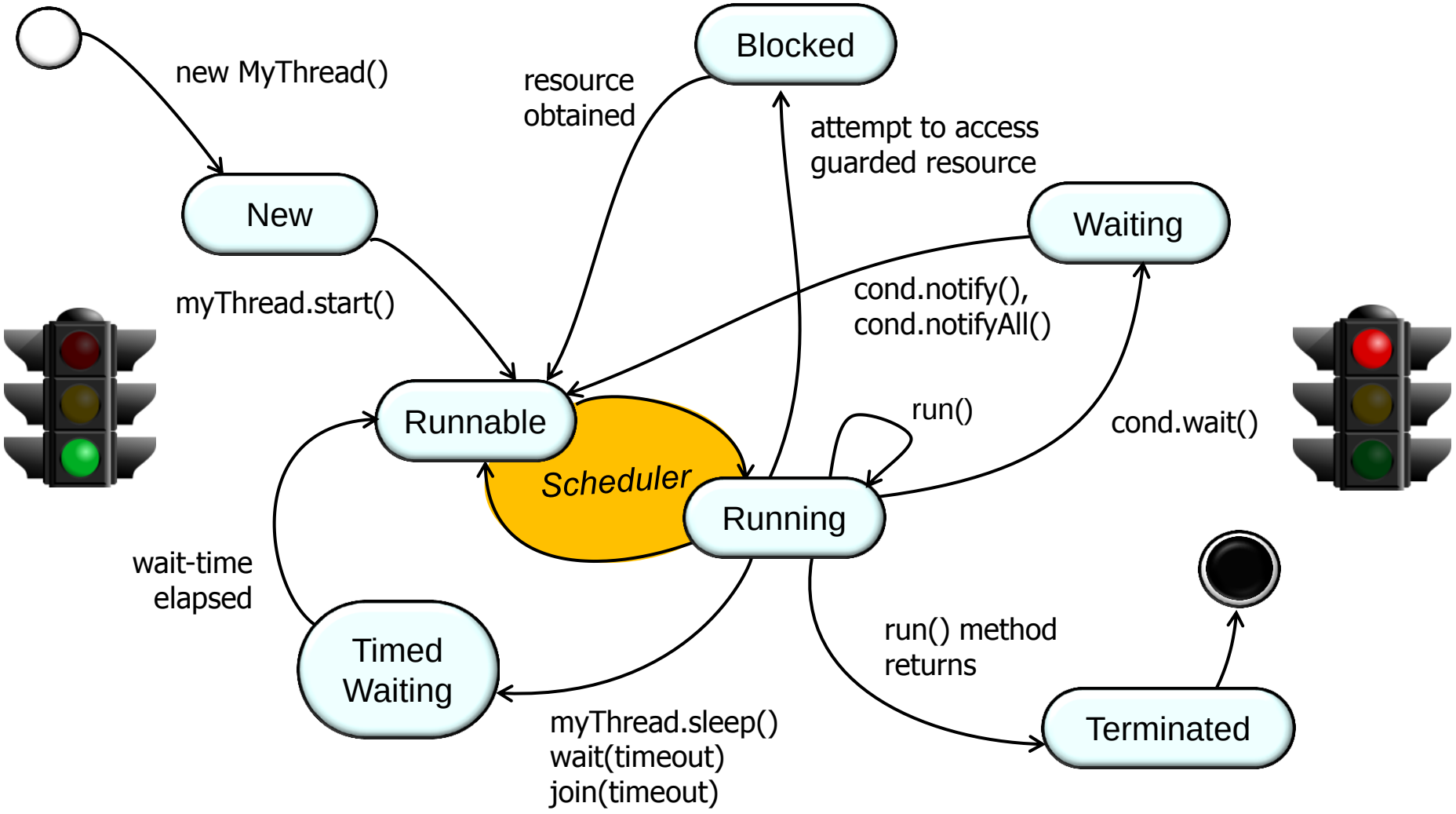
State Machine for Java Threads in Android



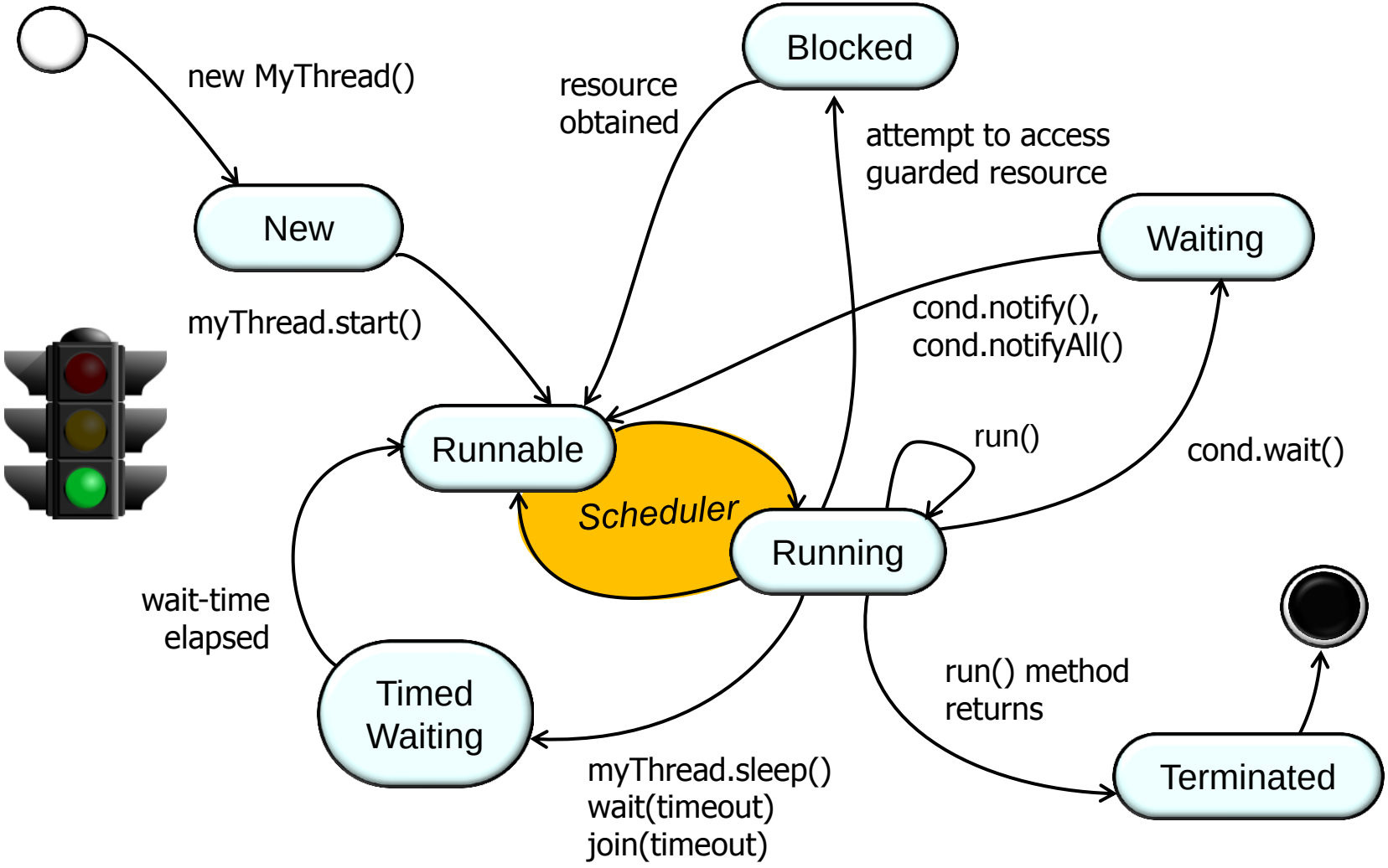
www.uml-diagrams.org/examples/java-6-thread-state-machine-diagram-example.html

Starting Java Threads on Android

Starting & Stopping Java Threads on Android



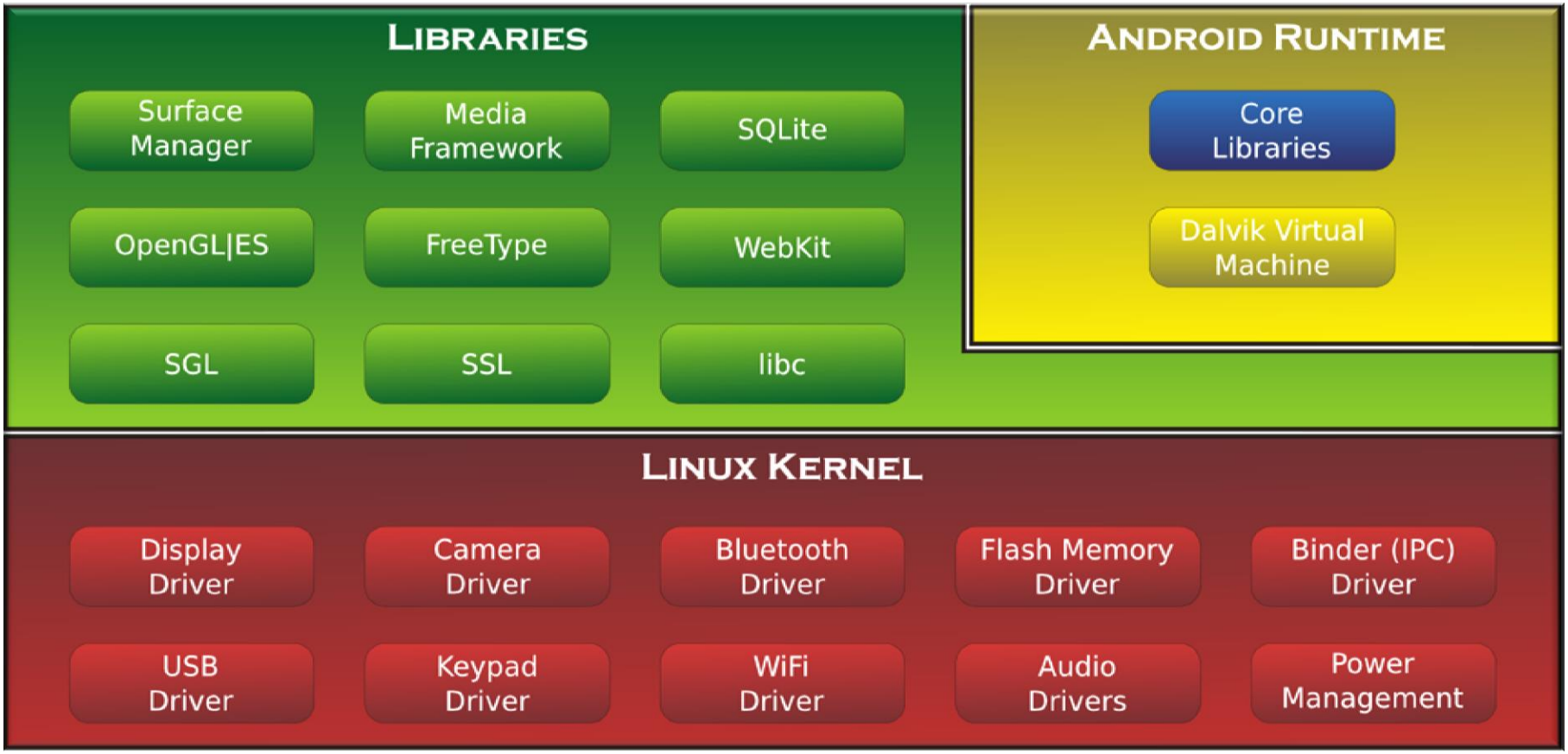
Starting Java Threads on Android



www.songlyrics.com/sound-of-music/do-re-mi-lyrics

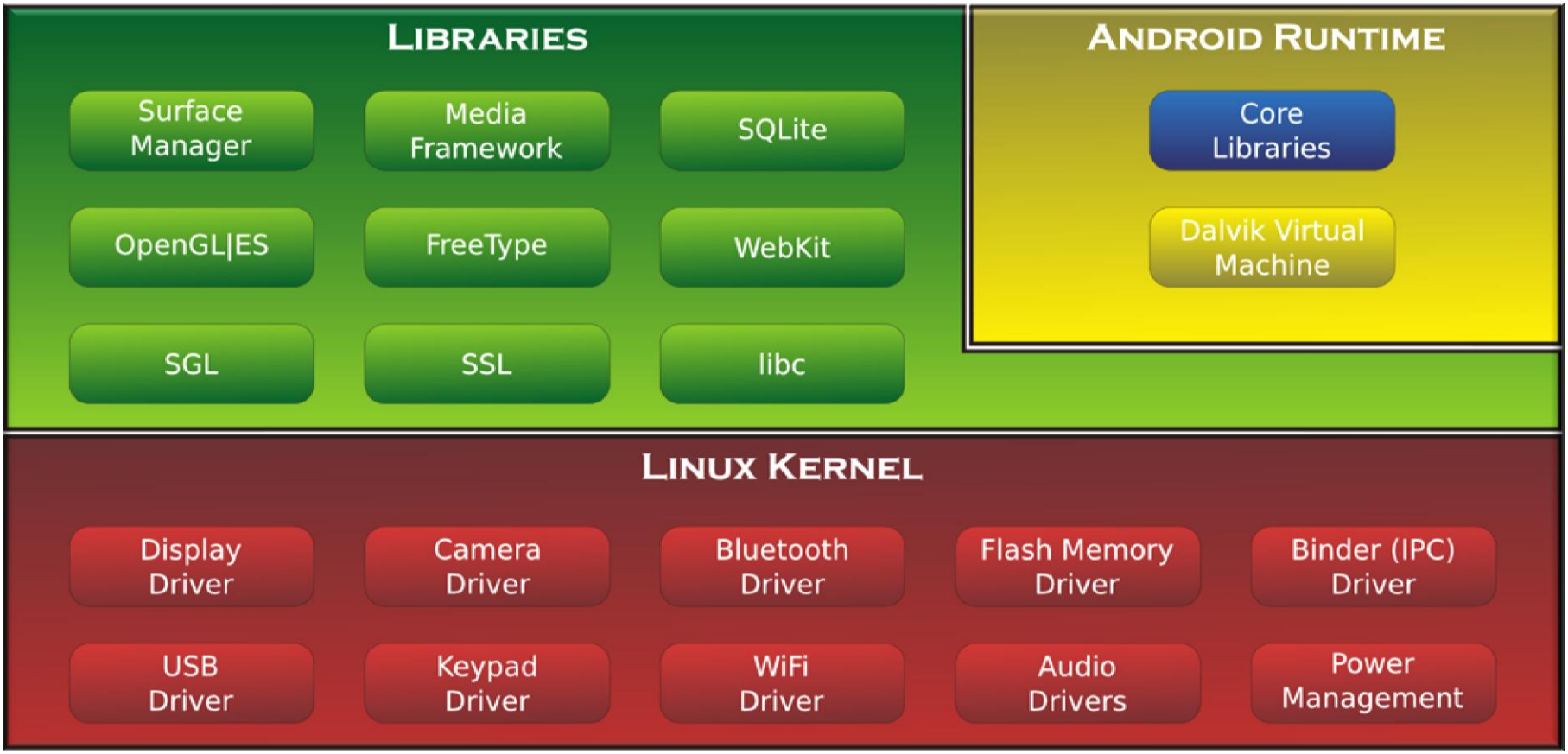
Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers



Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers



Again, you don't need to understand all these details to program Java threads!

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
```

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
2. Thread.start() // Java method
```

See libcore/luni/src/main/java/java/lang/Thread.java

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
```

See libcore/luni/src/main/java/java/lang/VMThread.java

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
```

See dalvik/vm/native/java_lang_VMThread.cpp

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

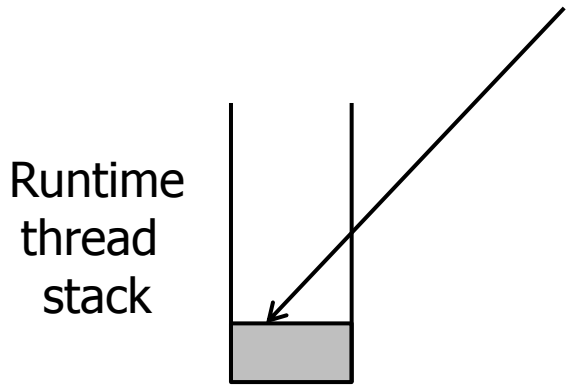
```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
5. dvmCreateInterpThread(Object* threadObj,
                         int reqStackSize) // Dalvik method
```

See dalvik/vm/Thread.cpp

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
5. dvmCreateInterpThread(Object* threadObj,
                         int reqStackSize) // Dalvik method
6. pthread_create(&threadHandle, &threadAttr,
                  interpThreadStart, newThread) // Pthreads method
```

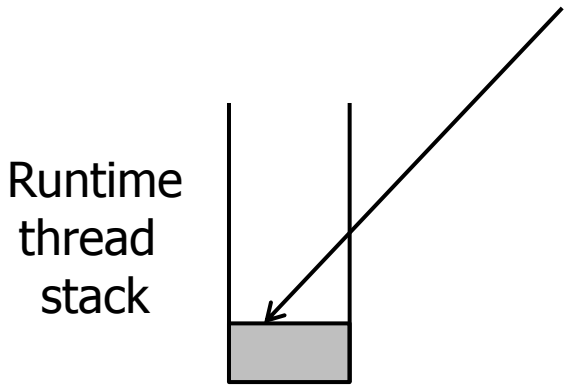


See bionic/libc/bionic/pthread.c

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
5. dvmCreateInterpThread(Object* threadObj,
                         int reqStackSize) // Dalvik method
6. pthread_create(&threadHandle, &threadAttr,
                  interpThreadStart, newThread) // Pthreads method
```

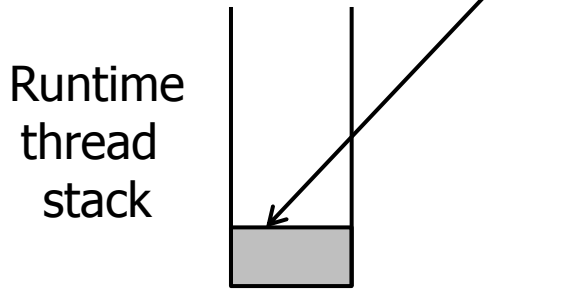


See bionic/libc/bionic/pthread.c

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
5. dvmCreateInterpThread(Object* threadObj,
                         int reqStackSize) // Dalvik method
6. pthread_create(&threadHandle, &threadAttr,
                  interpThreadStart, newThread) // Pthreads method
7. interpThreadStart(void* arg) // Adapter
```

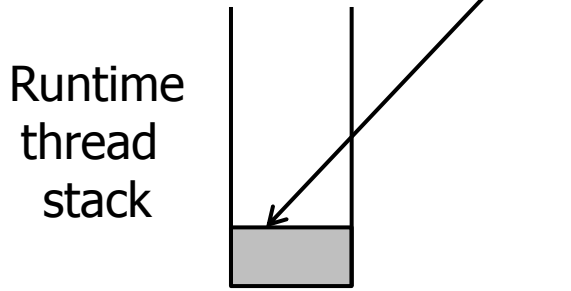


See dalvik/vm/Thread.cpp

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
5. dvmCreateInterpThread(Object* threadObj,
                         int reqStackSize) // Dalvik method
6. pthread_create(&threadHandle, &threadAttr,
                  interpThreadStart, newThread) // Pthreads method
7. interpThreadStart(void* arg) // Adapter
8. dvmCallMethod(self, run,
                 self->threadObj,
                 &unused) // Dalvik method
```

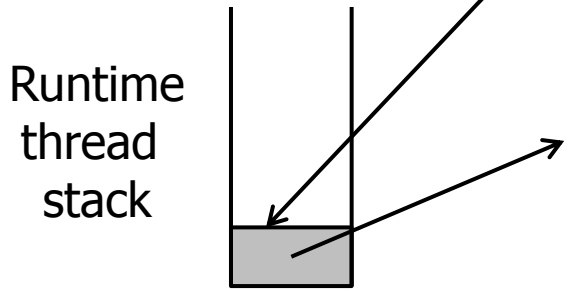


See dalvik/vm/interp/Stack.cpp

Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

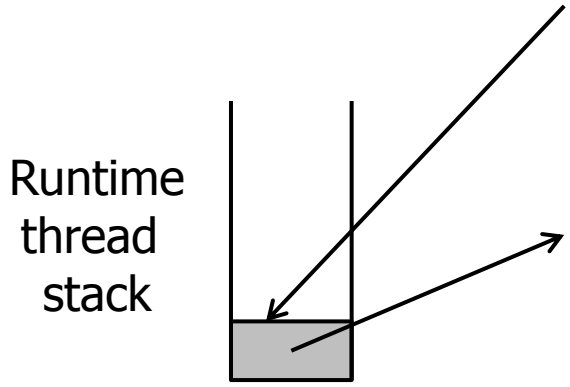
```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
5. dvmCreateInterpThread(Object* threadObj,
                         int reqStackSize) // Dalvik method
6. pthread_create(&threadHandle, &threadAttr,
                  interpThreadStart, newThread) // Pthreads method
7. interpThreadStart(void* arg) // Adapter
8. dvmCallMethod(self, run,
                 self->threadObj,
                 &unused) // Dalvik method
9. MyThread.run() // User-defined hook
```



Starting Java Threads on Android

- Calling start() on a Thread causes it to begin executing its run() hook method
  - Many steps occur at the Java middleware, virtual machine, & OS layers

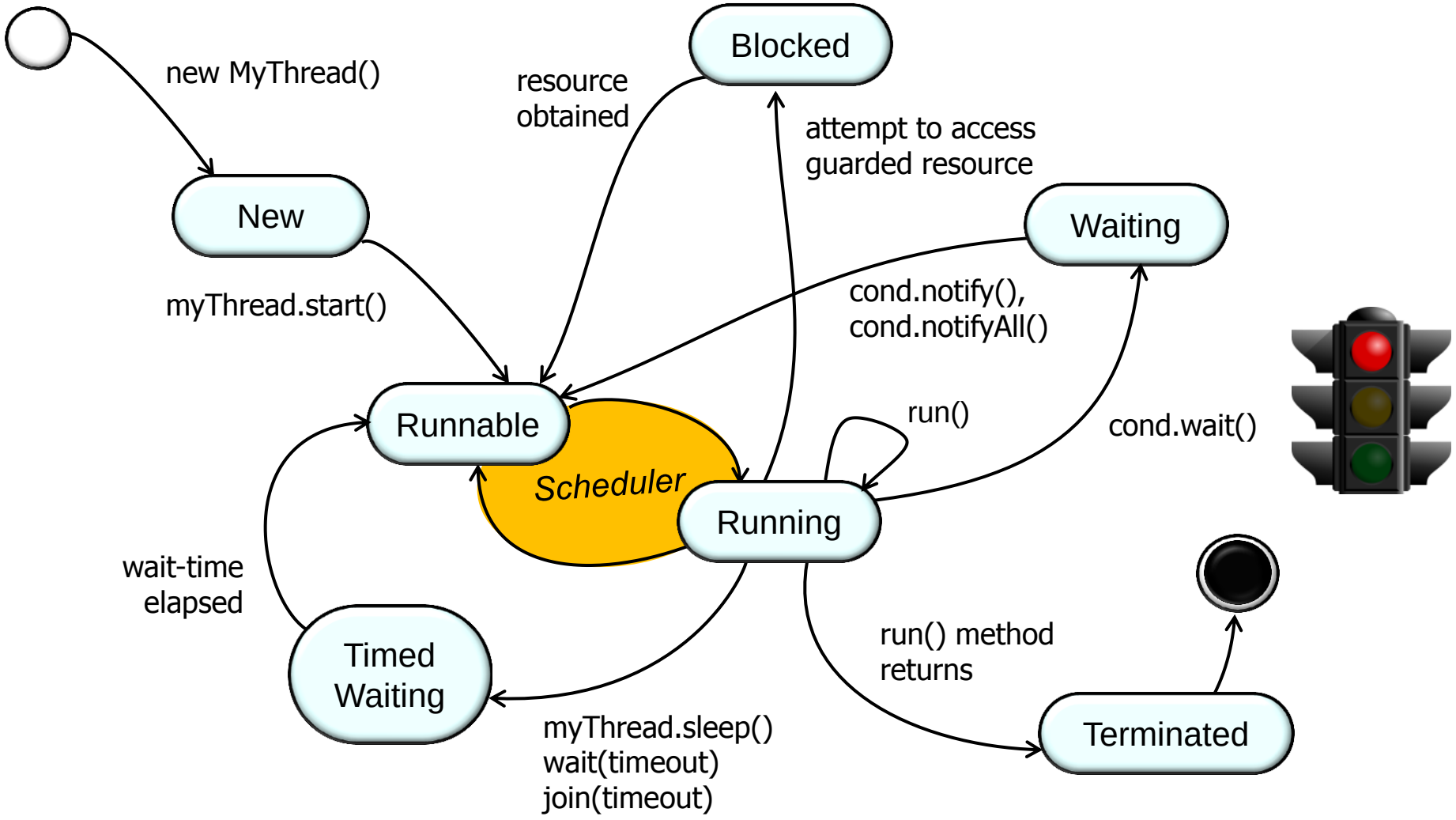
```
1. MyThread.start()
2. Thread.start() // Java method
3. VMThread.create() // Native method
4. Dalvik_java_lang_VMThread_create(const u4* args,
                                    JValue* pResult) // JNI method
5. dvmCreateInterpThread(Object* threadObj,
                         int reqStackSize) // Dalvik method
6. pthread_create(&threadHandle, &threadAttr,
                  interpThreadStart, newThread) // Pthreads method
7. interpThreadStart(void* arg) // Adapter
8. dvmCallMethod(self, run,
                 self->threadObj,
                 &unused) // Dalvik method
9. MyThread.run() // User-defined hook
```



source.android.com/devices/tech/dalvik/art.html has more info on ART

Stopping Java Threads on Android (Part 1)

Stopping Java Threads



www.songlyrics.com/sound-of-music/so-long-farewell-lyrics

Stopping Java Threads

- Stopping Threads is surprisingly hard

www.youtube.com/watch?v=5rzyuY8-Ao8 has more examples ;-)

Stopping Java Threads

- Stopping Threads is surprisingly hard

www.youtube.com/watch?v=5rzyuY8-Ao8 has more examples ;-)

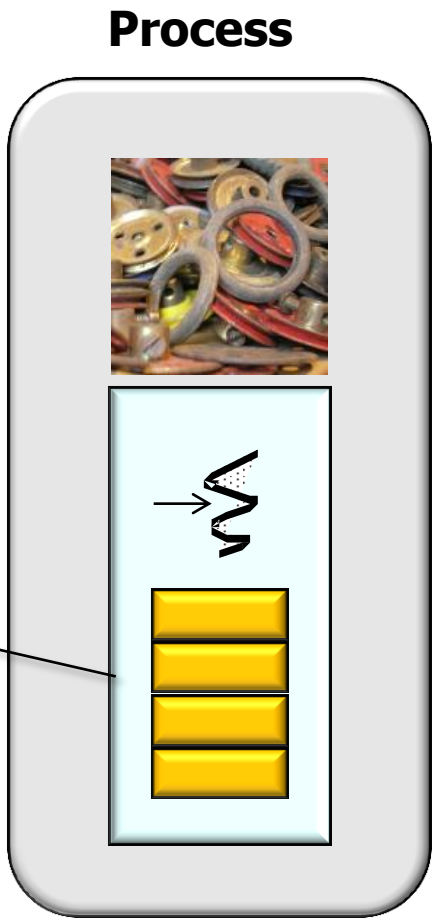
Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
  - Long running operations in a Thread must be coded to stop voluntarily!

```
public void run(){
  while (true) {
    // Check to see
    // if the thread
    // should stop
  }
}
```



Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag

```
public class MyRunnable
             implements Runnable
{

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                  isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

en.wikipedia.org/wiki/Volatile_variable#In_Java has more on volatile

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable
  - Add a stopMe() method that sets "isStopped" to true

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                  isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable
  - Add a stopMe() method that sets "isStopped" to true

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable
  - Add a stopMe() method that sets "isStopped" to true
  - Check "isStopped" periodically to see if thread's been stopped

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                  isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable
  - Add a stopMe() method that sets "isStopped" to true
  - Check "isStopped" periodically to see if thread's been stopped

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    ...
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
  - Add a volatile boolean flag "isStopped" to a class that implements Runnable
  - Add a stopMe() method that sets "isStopped" to true
  - Check "isStopped" periodically to see if thread's been stopped

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                  isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    return;
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
- This solution is lightweight, but isn't integrated into the JVM

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    return;
```

Stopping Java Threads

- Stopping Threads is surprisingly hard
- There's no safe way to stop a Java thread involuntarily
- One way to stop a thread is to use a "stop" flag
- This solution is lightweight, but isn't integrated into the JVM

```
public class MyRunnable
             implements Runnable
{
  private volatile boolean
                isStopped = false;

  public void stopMe() {
    isStopped = true;
  }

  public void run() {
    while(isStopped != true) {
      // a long-running operation
    }
    return;
```

Blocking operations won't be awakened, which impedes shutdown processing

Stopping Java Threads on Android (Part 2)

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

Interrupts

An *interrupt* is an indication to a thread that it should stop what it is doing and do something else. It's up to the programmer to decide exactly how a thread responds to an interrupt, but it is very common for the thread to terminate. This is the usage emphasized in this lesson.

A thread sends an interrupt by invoking `interrupt` on the `Thread` object for the thread to be interrupted. For the interrupt mechanism to work correctly, the interrupted thread must support its own interruption.

See developer.android.com/reference/java/lang/Thread.html#interrupt()

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

  t1.start();
  ... // Run concurrently
  t1.interrupt();
  ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped

```
static int main(String args[]) {
  Thread t1 =
    new Thread(new Runnable() {
      public void run(){
        for (int i = 0;
             i < args.length; i++) {
          processBlocking(args[i]);
          processNonBlocking(args[i]);
        }
      }
    });

t1.start();
... // Run concurrently
t1.interrupt();
...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped
  - Certain blocking operations will return automatically
    - e.g., wait(), join(), sleep() & blocking I/O calls

```
void processBlocking(String input) {
  ...
  while (true) {
    try {
      Thread.currentThread().
        sleep(interval);
      synchronized(this) {
        while (someConditionFalse)
          wait();
      }
    }
    catch (InterruptedException e)
    { ... }
    ...
  }
}
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped
- Certain blocking operations will return automatically
  - e.g., wait(), join(), sleep() & blocking I/O calls

```
void processBlocking(String input) {
  ...
  while (true) {
    try {
      Thread.currentThread().
        sleep(interval);
      synchronized(this) {
        while (someConditionFalse)
          wait();
      }
    }
    catch (InterruptedException e)
    { ... }
    ...
  }
}
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped
- Certain blocking operations will return automatically
  - e.g., wait(), join(), sleep() & blocking I/O calls

```
void processBlocking(String input) {
  ...
  while (true) {
    try {
      Thread.currentThread().
        sleep(interval);
      synchronized(this) {
        while (someConditionFalse)
          wait();
      }
    }
    catch (InterruptedException e)
    { ... }
    ...
  }
}
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped
  - Certain blocking operations will return automatically
  - Non-blocking operations must check periodically to see if Thread.interrupted() has been called

```
void processNonBlocking(String input) {
  ...
  while (true) {
    ... // Do long-running computation
    if (Thread.interrupted())
      throw new InterruptedException();
    ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped
  - Certain blocking operations will return automatically
  - Non-blocking operations must check periodically to see if Thread.interrupted() has been called

```
void processNonBlocking(String input) {
  ...
  while (true) {
    ... // Do long-running computation
    if (Thread.interrupted())
      throw new InterruptedException();
    ...
```

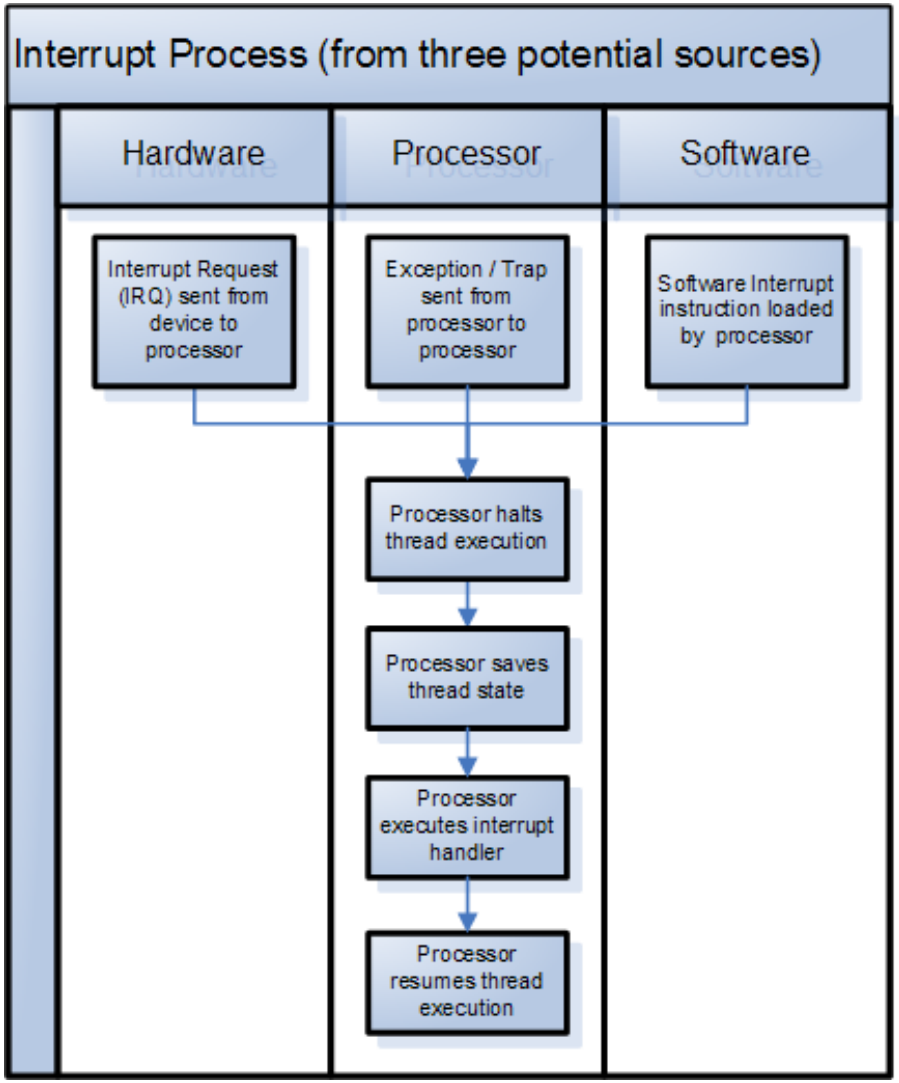
Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
  - Posts an interrupt request to a Thread
  - Check periodically to see if Thread's been stopped
  - Certain blocking operations will return automatically
  - Non-blocking operations must check periodically to see if Thread.interrupted() has been called

```
void processNonBlocking(String input) {
  ...
  while (true) {
    ... // Do long-running computation
    if (Thread.interrupted())
      throw new InterruptedException();
    ...
```

Stopping Java Threads

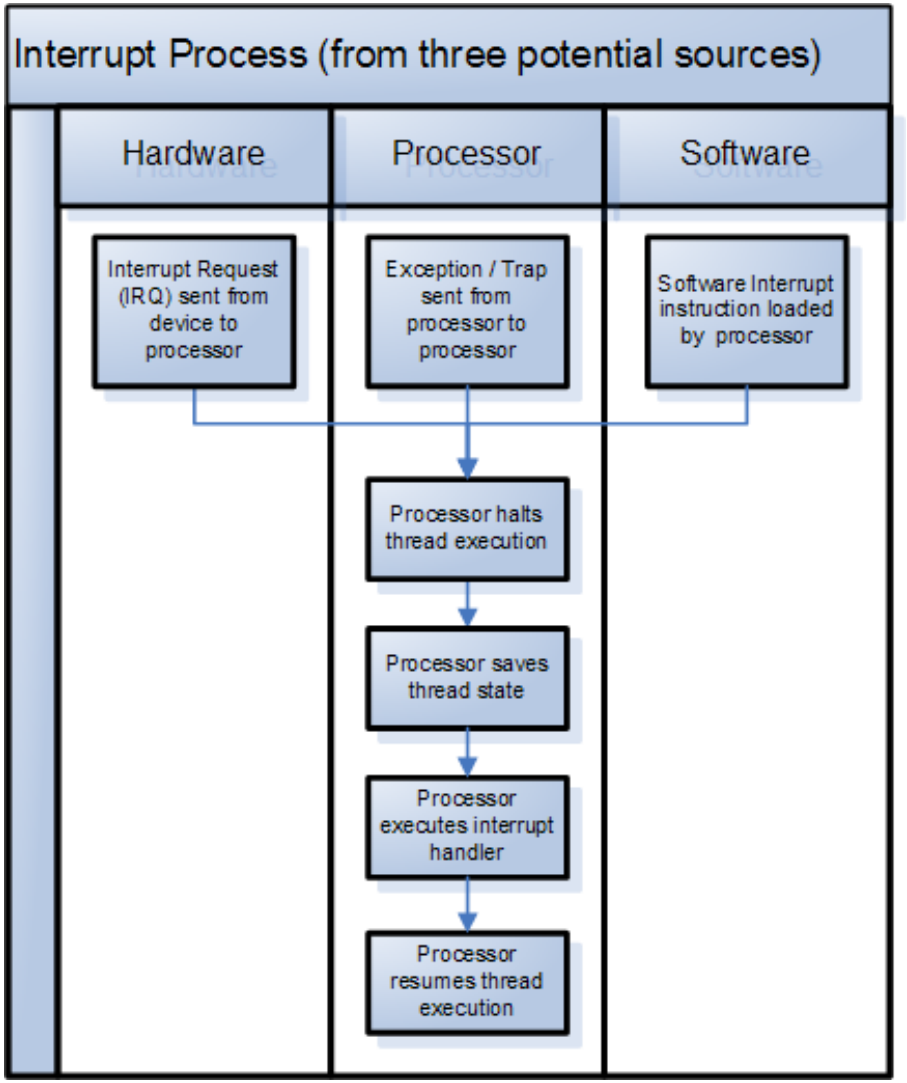
- Another way to stop a Thread is to use interrupt()
- Thread interrupts don't behave like traditional hardware or operating system interrupts



en.wikipedia.org/wiki/Interrupt has more on interrupts

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
- Thread interrupts don't behave like traditional hardware or operating system interrupts
  - They aren't delivered asynchronously or preemptively



en.wikipedia.org/wiki/Interrupt has more on interrupts

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
- Thread interrupts don't behave like traditional hardware or operating system interrupts
  - They aren't delivered asynchronously or preemptively
  - They must be tested for explicitly

```
void processNonBlocking(String input) {
  ...
  while (true) {
    ... // Do long-running computation
    if (Thread.interrupted())
      throw new InterruptedException();
    ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
- Thread interrupts don't behave like traditional hardware or operating system interrupts
  - They aren't delivered asynchronously or preemptively
  - They must be tested for explicitly

```
void processNonBlocking(String input) {
  ...
  while (true) {
    ... // Do long-running computation
    if (Thread.interrupted())
      throw new InterruptedException();
    ...
```

Stopping Java Threads

- Another way to stop a Thread is to use interrupt()
- Thread interrupts don't behave like traditional hardware or operating system interrupts
- Both solutions require Threads to cooperate when stopping them

Stopping Java Threads

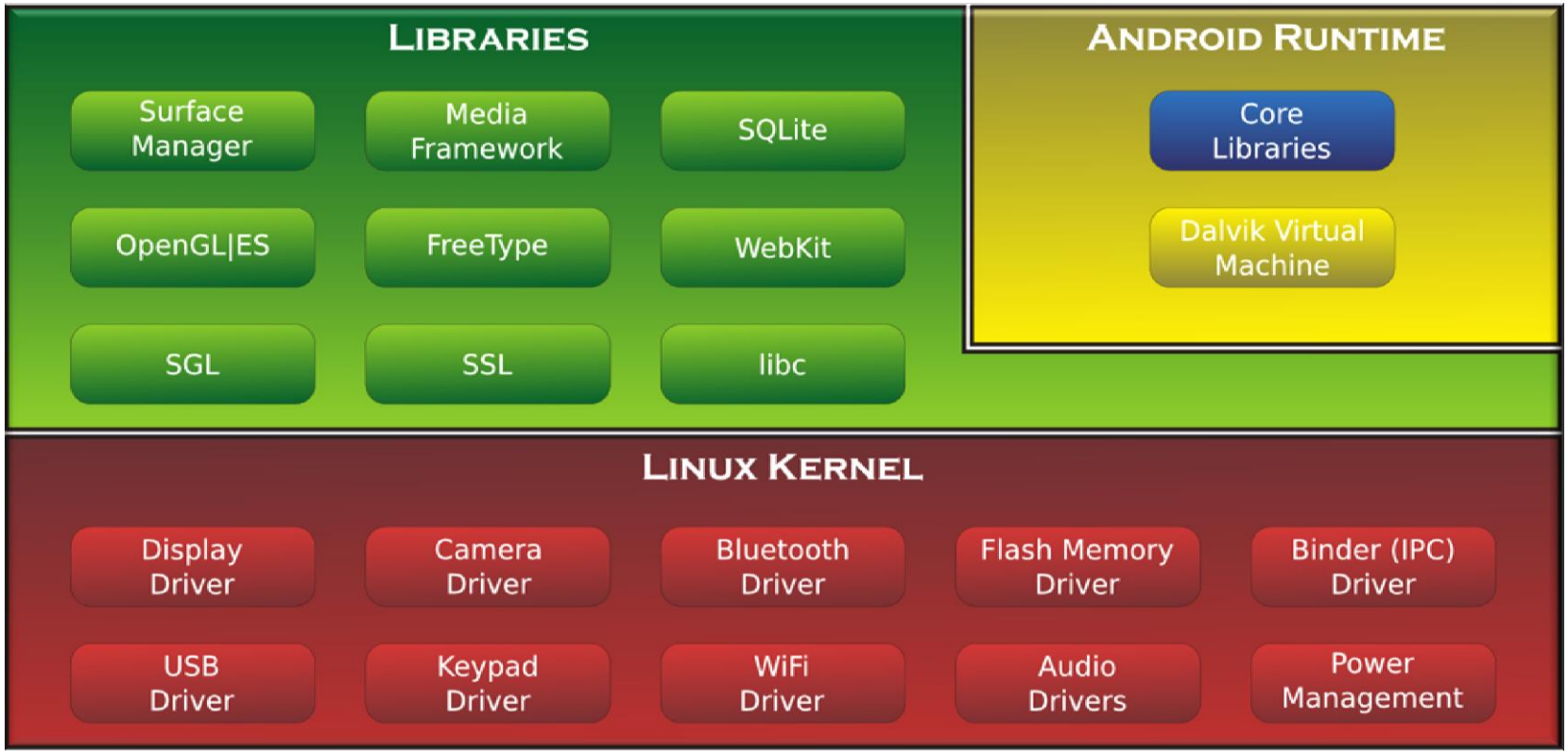
- Another way to stop a Thread is to use interrupt()
- Thread interrupts don't behave like traditional hardware or operating system interrupts
- Both solutions require Threads to cooperate when stopping them
  - This approach can be tedious, but it's the recommended way of stopping Java & Android Threads

stackoverflow.com/questions/8505707/android-best-and-safe-way-to-stop-thread

Summary

Summary

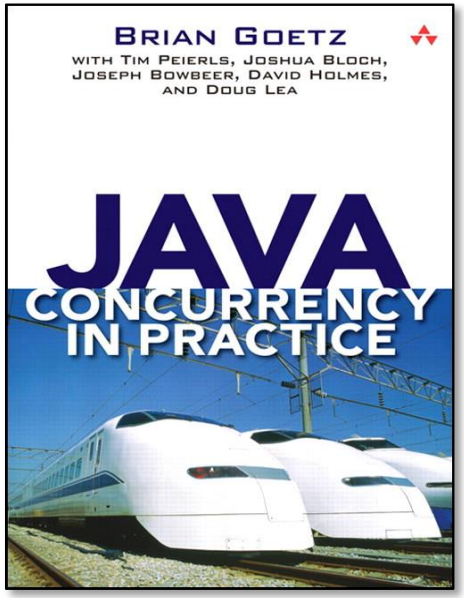
- Java Threads are implemented using various mechanisms defined by lower layers of the Android software stack

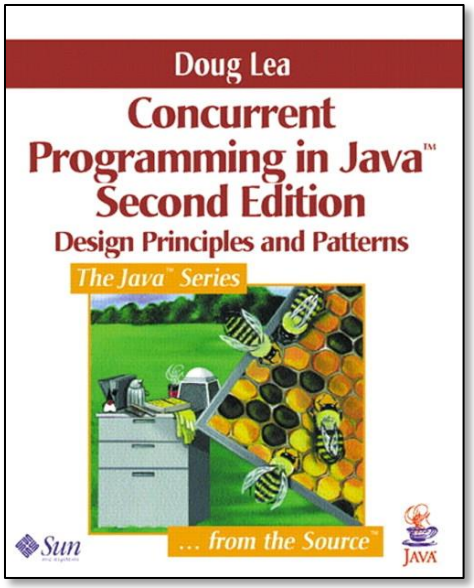


en.wikipedia.org/wiki/Dalvik_(software) has more info on the Dalvik VM

Summary

- Java Threads are implemented using various methods & functions defined by lower layers of the Android software stack
- There are good books on Java concurrency mechanisms that also cover key patterns & best practices of programming multi-threaded software in Java





See online supplement at gee.cs.oswego.edu/dl/cpj