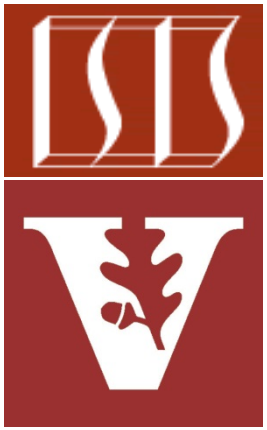
Android Concurrency: Java CountDownLatch



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 how CountDownLatch allows one or more threads to wait until a set of operations being performed in other threads completes

CountDownLatch

Added in API level 1

extends Object

java.lang.Object
↳java.util.concurrent.CountDownLatch

Class Overview

A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes.

A `CountDownLatch` is initialized with a given *count*. The `await` methods block until the current count reaches zero due to invocations of the `countDown()` method, after which all waiting threads are released and any subsequent invocations of `await` return immediately. This is a one-shot phenomenon -- the count cannot be reset. If you need a version that resets the count, consider using a `CyclicBarrier`.

A `CountDownLatch` is a versatile synchronization tool and can be used for a number of purposes. A `CountDownLatch` initialized with a count of one serves as a simple on/off latch, or gate: all threads invoking `await` wait at the gate until it is opened by a thread invoking `countDown()`. A `CountDownLatch` initialized to *N* can be used to make one thread wait until *N* threads have completed some action, or some action has been completed N times.

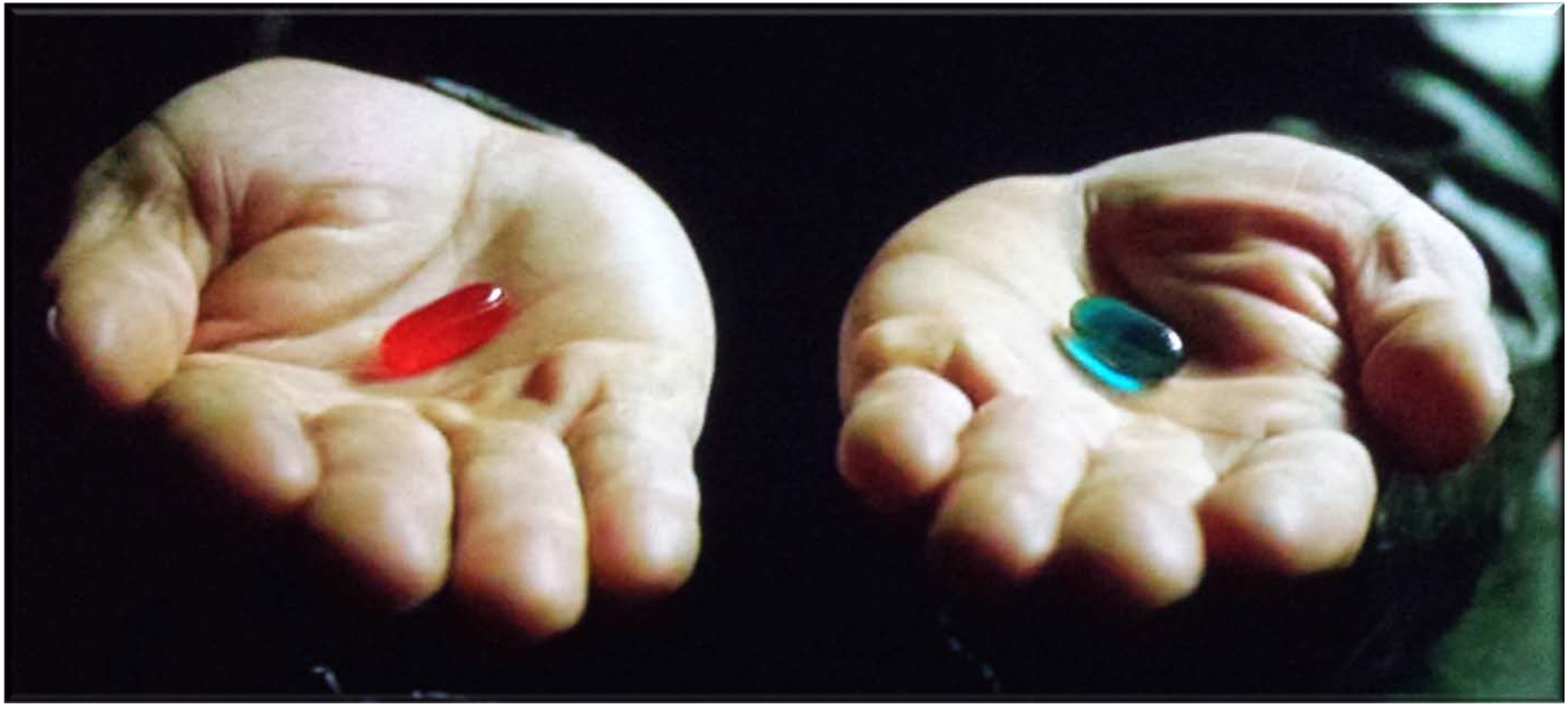
Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point

en.wikipedia.org/wiki/Barrier_(computer_science) has more information

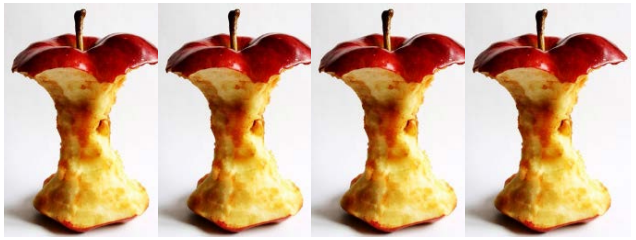
Overview of Barriers

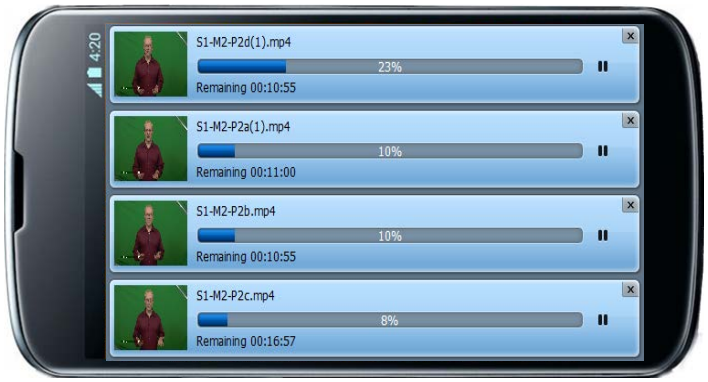
- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways



Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized

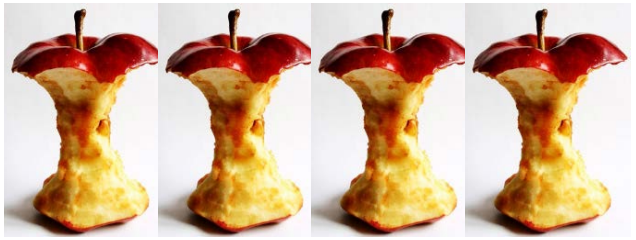


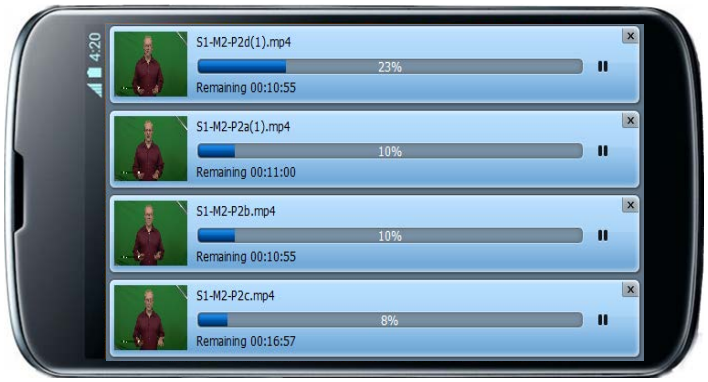


Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized

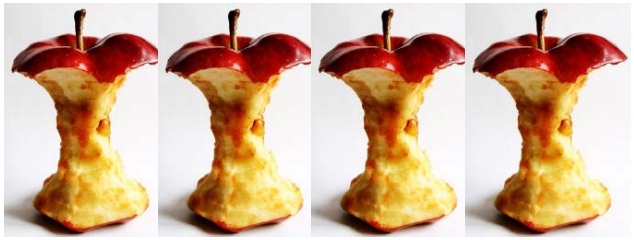
Main thread

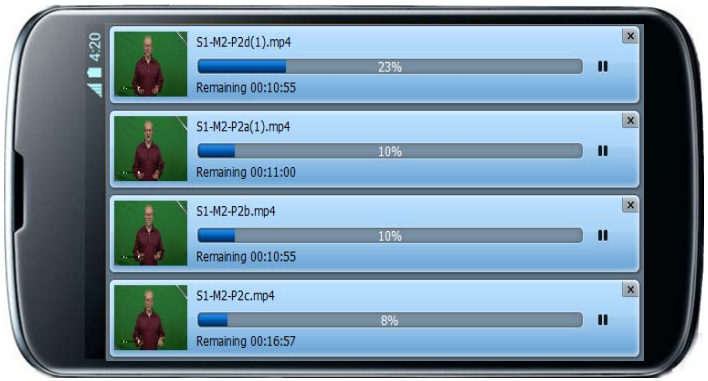




Overview of Barriers

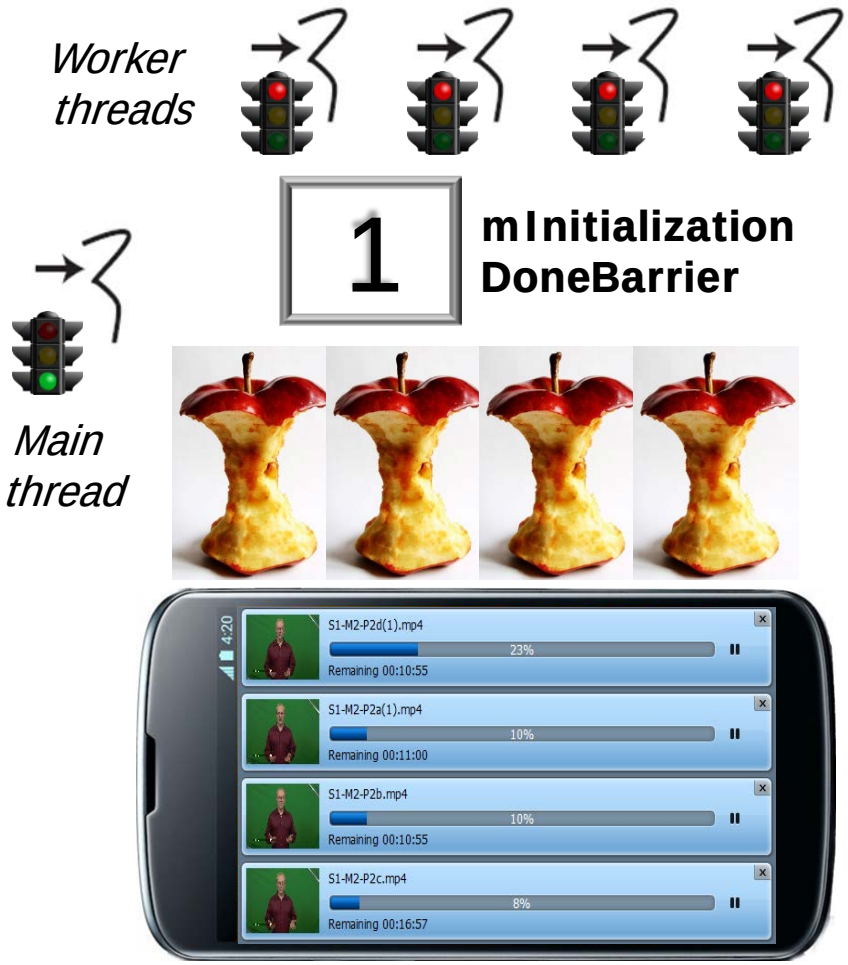
- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized





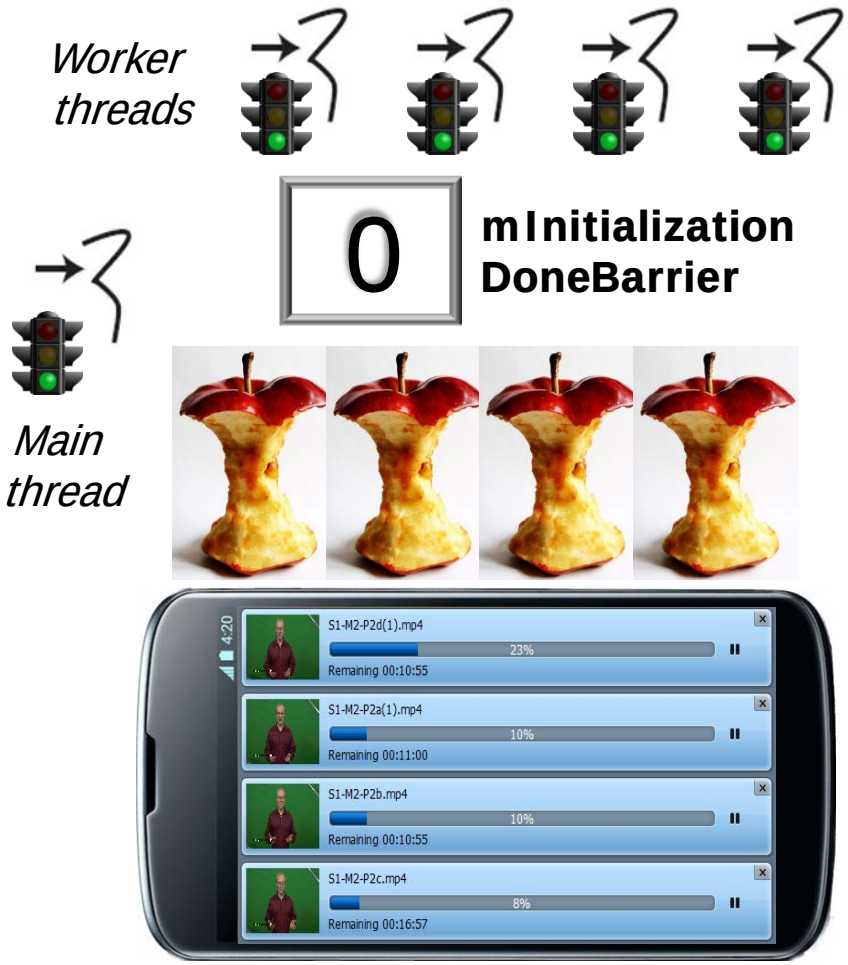
Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized



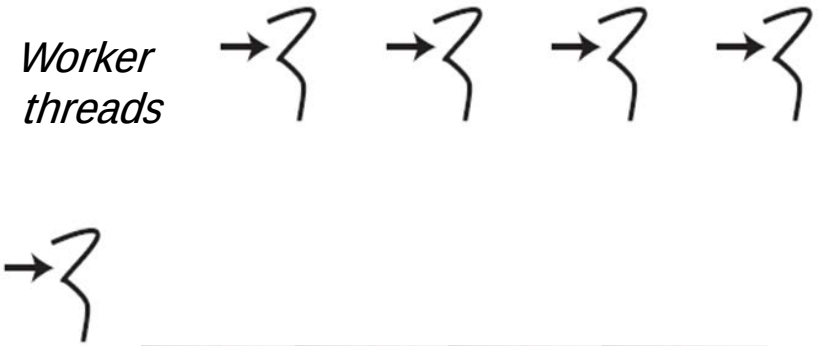
Overview of Barriers

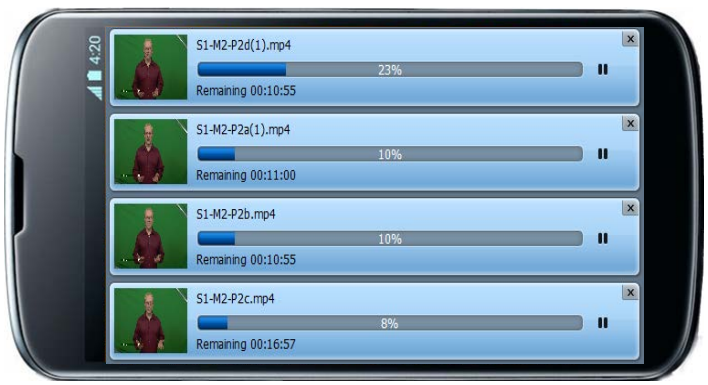
- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized



Overview of Barriers

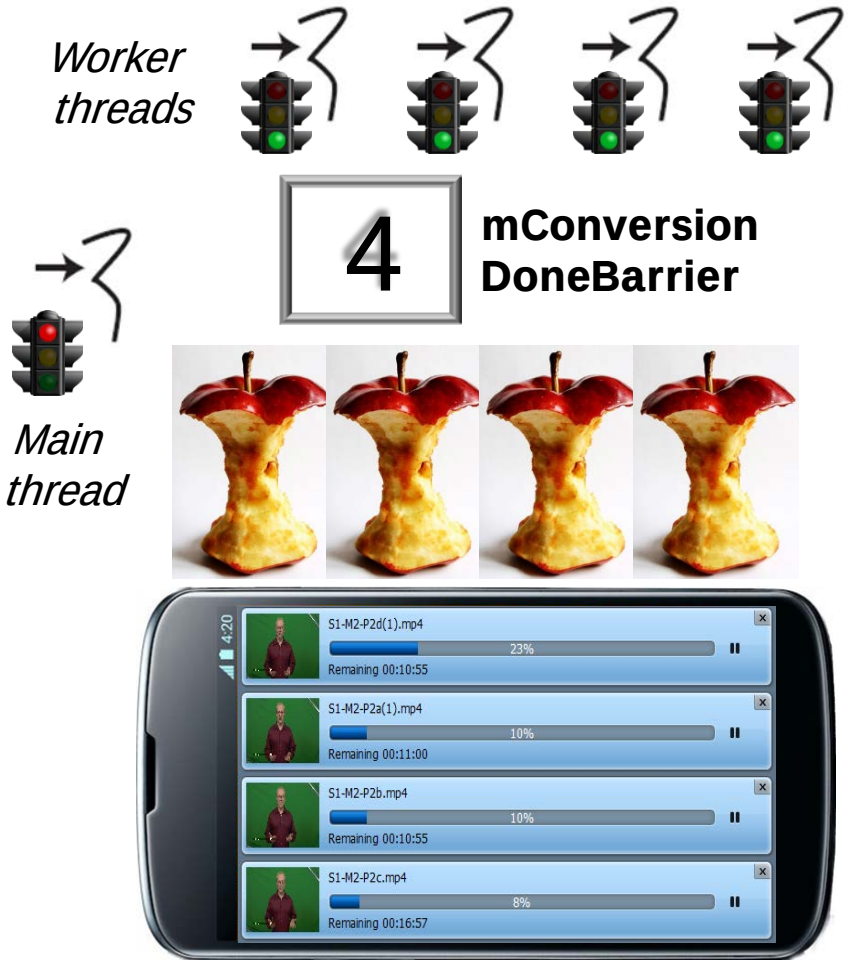
- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized
  - Wait until all concurrent threads are done with their processing before continuing





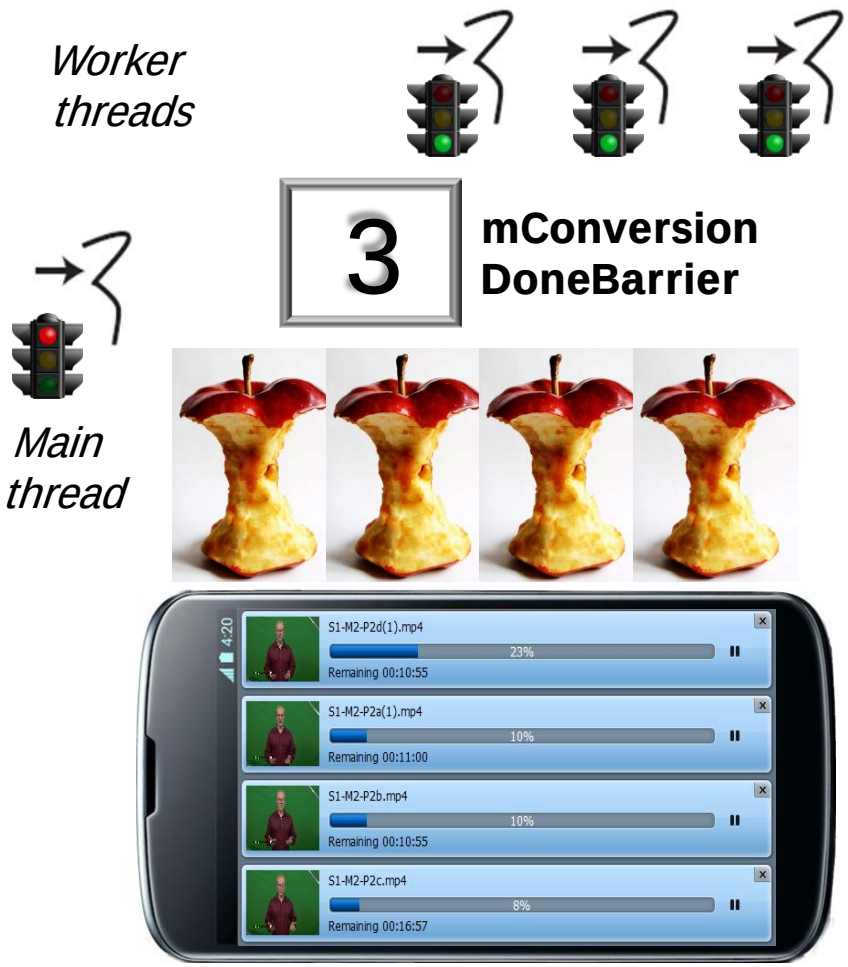
Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized
  - Wait until all concurrent threads are done with their processing before continuing



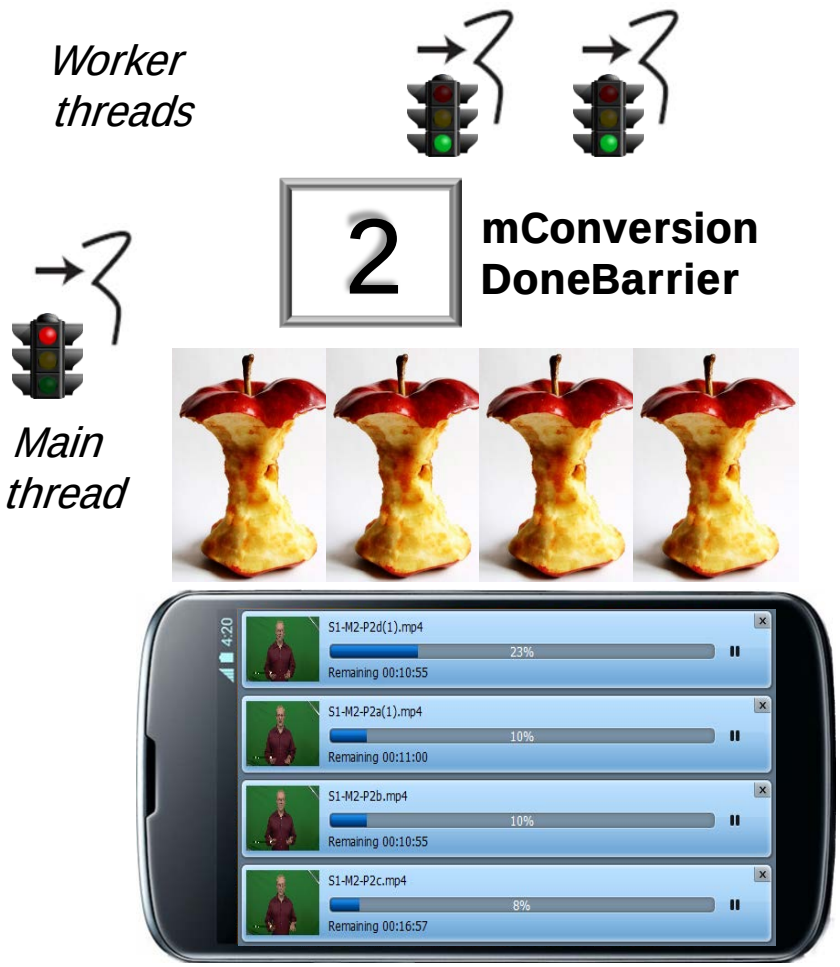
Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized
  - Wait until all concurrent threads are done with their processing before continuing



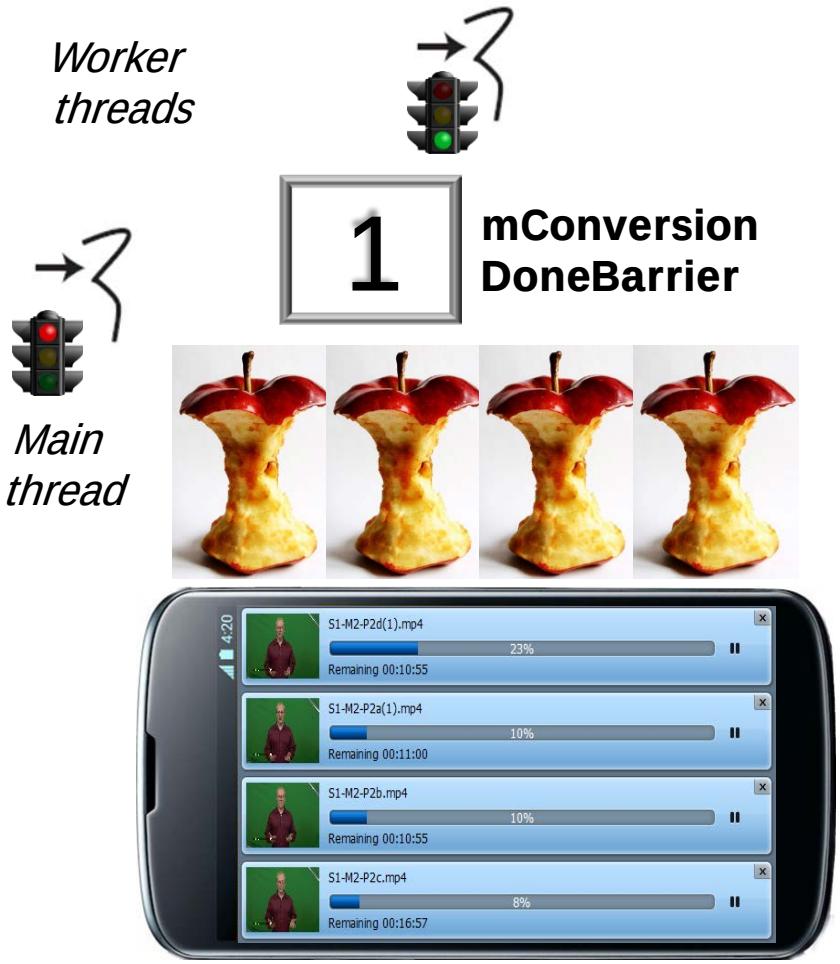
Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized
  - Wait until all concurrent threads are done with their processing before continuing



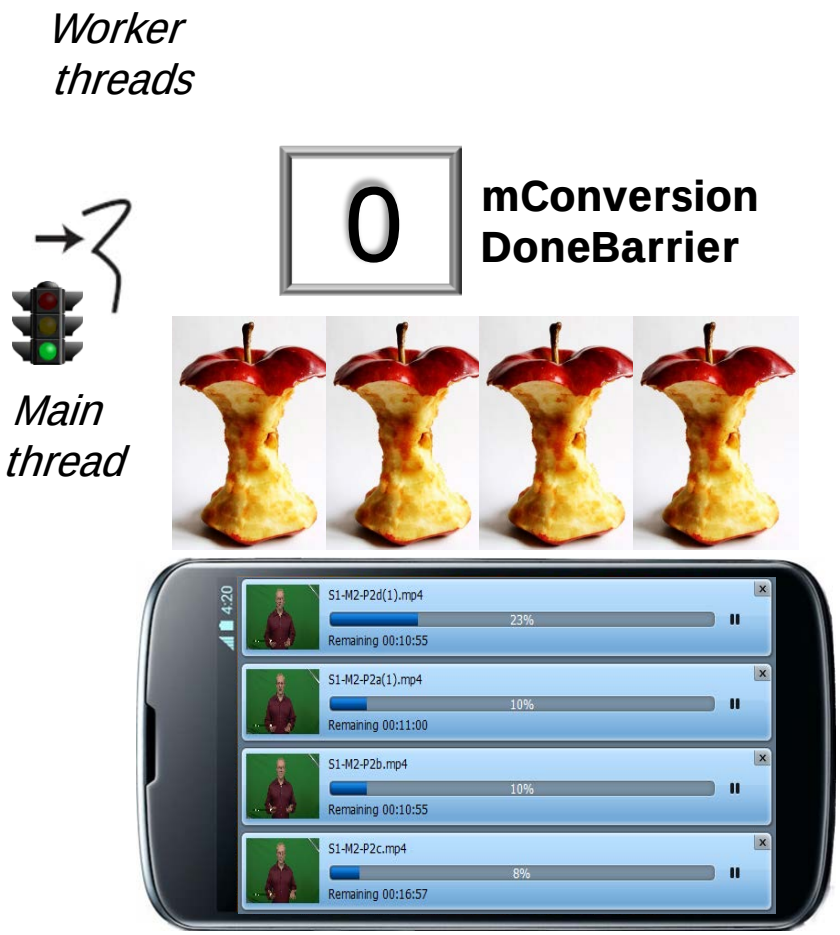
Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized
  - Wait until all concurrent threads are done with their processing before continuing



Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
  - Defer the start of concurrent computations until an object is initialized
  - Wait until all concurrent threads are done with their processing before continuing



Overview of Barriers

- A barrier is a synchronization method that halts the progress of one or more threads at a particular point
- Barriers can be used in several ways
- A human known use is the protocol used by a tour guide

en.wikipedia.org/wiki/Tour_guide has more info on tour guides

Overview of Java CountDownLatch

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization

CountDownLatch

Added in API level 1

extends Object

java.lang.Object
↳java.util.concurrent.CountDownLatch

Class Overview

A synchronization aid that allows one or more threads to wait until a set of operations being performed in other threads completes.

A `CountDownLatch` is initialized with a given *count*. The `await` methods block until the current count reaches zero due to invocations of the `countDown()` method, after which all waiting threads are released and any subsequent invocations of `await` return immediately. This is a one-shot phenomenon -- the count cannot be reset. If you need a version that resets the count, consider using a `CyclicBarrier`.

A `CountDownLatch` is a versatile synchronization tool and can be used for a number of purposes. A `CountDownLatch` initialized with a count of one serves as a simple on/off latch, or gate: all threads invoking `await` wait at the gate until it is opened by a thread invoking `countDown()`. A `CountDownLatch` initialized to *N* can be used to make one thread wait until *N* threads have completed some action, or some action has been completed N times.

developer.android.com/reference/java/util/concurrent/CountDownLatch.html

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
  - Most of it is written in Java

```
public class CountDownLatch {
   ...
```

See libcore/luni/src/main/java/java/util/concurrent/CountDownLatch.java

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern

```
public class CountDownLatch {
  ...
  /**
   * Synchronization control or
   * CountDownLatch. Uses AQS
   * state to represent count.
   */
  private static final class
    Sync extends
      AbstractQueuedSynchronizer {
    ...
  }

  private final Sync sync;
  ...
```

en.wikipedia.org/wiki/Bridge_pattern contains more on the *Bridge* pattern

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
  - Inherits functionality from the AbstractQueuedSynchronizer class

```
public class CountDownLatch {
  ...
  /**
   * Synchronization control or
   * CountDownLatch. Uses AQS
   * state to represent count.
   */
  private static final class
    Sync extends
      AbstractQueuedSynchronizer {
    ...
  }

  private final Sync sync;
  ...
```

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
  - Inherits functionality from the AbstractQueuedSynchronizer class
  - The constructor initializes CountDownLatch with count

```
public class CountDownLatch {
  ...
  public CountDownLatch
    (int count) {
    ...
    this.sync = new Sync(count);
  }
  ...
```

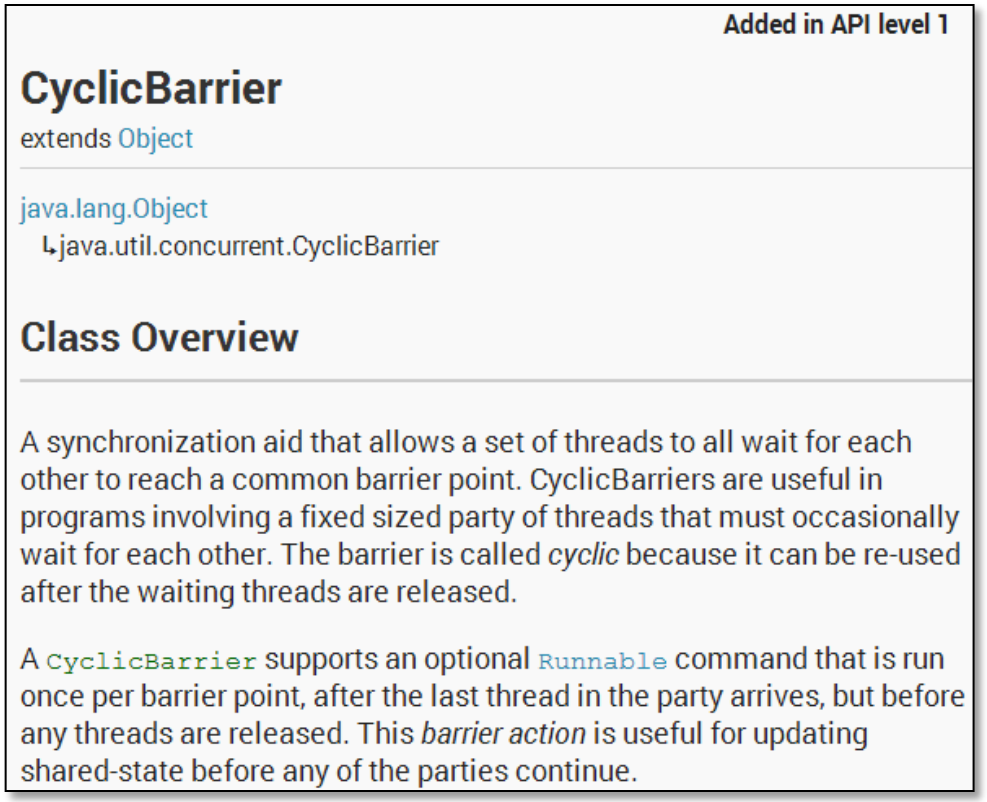
Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
  - Inherits functionality from the AbstractQueuedSynchronizer class
  - The constructor initializes CountDownLatch with count
    - The count cannot be reset

```
public class CountDownLatch {
  ...
  public CountDownLatch
    (int count) {
    ...
    this.sync = new Sync(count);
  }
  ...
```

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
  - Inherits functionality from the AbstractQueuedSynchronizer class
  - The constructor initializes CountDownLatch with count
    - The count cannot be reset
    - CyclicBarrier enables count to be reset



Added in API level 1

CyclicBarrier

extends Object

java.lang.Object
↳java.util.concurrent.CyclicBarrier

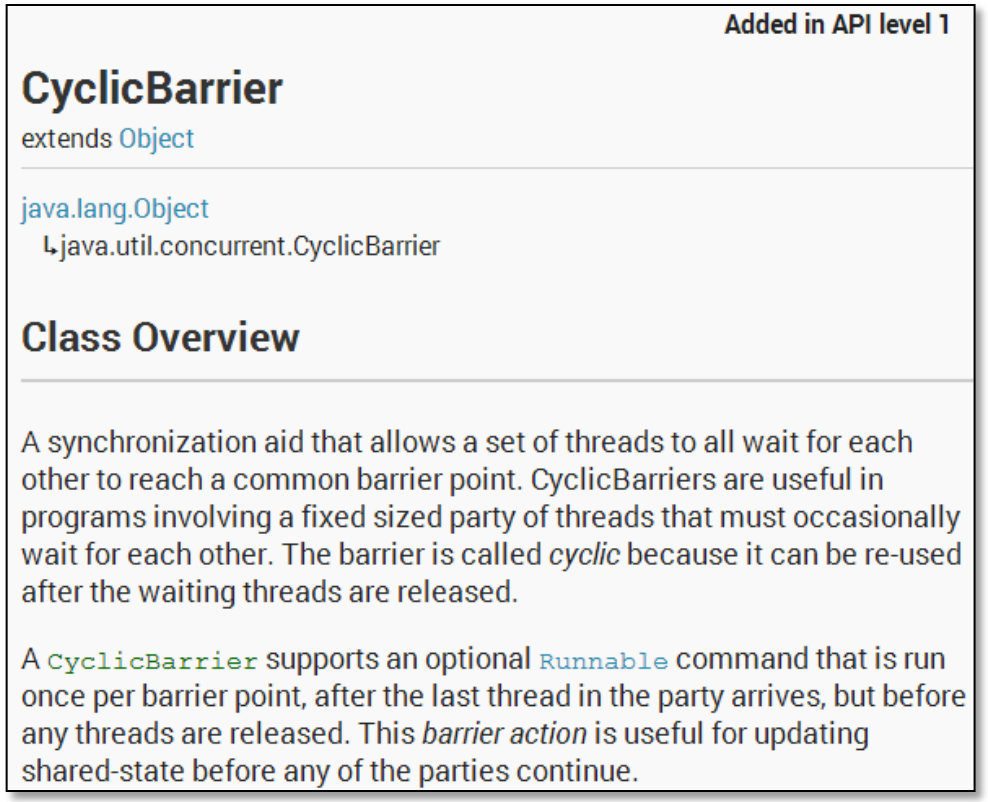
Class Overview

A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. CyclicBarriers are useful in programs involving a fixed sized party of threads that must occasionally wait for each other. The barrier is called *cyclic* because it can be re-used after the waiting threads are released.

A `CyclicBarrier` supports an optional `Runnable` command that is run once per barrier point, after the last thread in the party arrives, but before any threads are released. This *barrier action* is useful for updating shared-state before any of the parties continue.

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
  - Inherits functionality from the AbstractQueuedSynchronizer class
  - The constructor initializes CountDownLatch with count
    - The count cannot be reset
    - CyclicBarrier enables count to be reset
      - Useful for fixed-sized number of threads that wait for each other



Added in API level 1

CyclicBarrier

extends Object

java.lang.Object
↳java.util.concurrent.CyclicBarrier

Class Overview

A synchronization aid that allows a set of threads to all wait for each other to reach a common barrier point. CyclicBarriers are useful in programs involving a fixed sized party of threads that must occasionally wait for each other. The barrier is called *cyclic* because it can be re-used after the waiting threads are released.

A `CyclicBarrier` supports an optional `Runnable` command that is run once per barrier point, after the last thread in the party arrives, but before any threads are released. This *barrier action* is useful for updating shared-state before any of the parties continue.

See developer.android.com/reference/java/util/concurrent/CyclicBarrier.html

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
- Its key methods are await() & countDown()

```
public class CountDownLatch {
  ...
  public void await() ... {
    sync.acquire...(1);
  }

  public void countDown() {
    sync.releaseShared(1);
  }
  ...
```

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
- Its key methods are await() & countDown()
  - await() causes calling thread to wait until the latch's count reaches zero

```
public class CountDownLatch {
  ...
  public void await() ... {
    sync.acquire...(1);
  }

  public void countDown() {
    sync.releaseShared(1);
  }
  ...
```

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
- Its key methods are await() & countDown()
  - await() causes calling thread to wait until the latch's count reaches zero
  - countDown() decrements latch count, releasing all waiting threads if count reaches zero

```
public class CountDownLatch {
  ...
  public void await() ... {
    sync.acquire...(1);
  }

  public void countDown() {
    sync.releaseShared(1);
  }
  ...
```

Overview of Java CountDownLatch

- The CountDownLatch class implements barrier synchronization
- CountDownLatch uses the *Bridge* pattern
- Its key methods are await() & countDown()
  - await() causes calling thread to wait until the latch's count reaches zero
  - countDown() decrements latch count, releasing all waiting threads if count reaches zero

```
public class CountDownLatch {
  ...
  public void await() ... {
    sync.acquire...(1);
  }

  public void countDown() {
    sync.releaseShared(1);
  }
  ...
```

Threads calling countDown() needn't wait for count to reach 0 before proceeding

Using CountDownLatch in Android

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app

Contacts Provider

The Contacts Provider is a powerful and flexible Android component that manages the device's central repository of data about people. The Contacts Provider is the source of data you see in the device's contacts application, and you can also access its data in your own application and transfer data between the device and online services. The provider accommodates a wide range of data sources and tries to manage as much data as possible for each person, with the result that its organization is complex. Because of this, the provider's API includes an extensive set of contract classes and interfaces that facilitate both data retrieval and modification.

This guide describes the following:

- The basic provider structure.
- How to retrieve data from the provider.
- How to modify data in the provider.
- How to write a sync adapter for synchronizing data from your server to the Contacts Provider.

This guide assumes that you know the basics of Android content providers. To learn more about Android content providers, read the Content Provider Basics guide. The Sample Sync Adapter sample app

See developer.android.com/guide/topics/providers/contacts-provider.html

Using CountDownLatch in Android

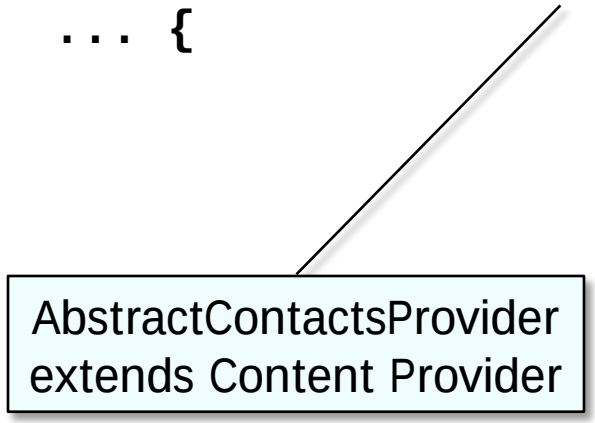
- Contacts Provider is the source of data for a device's Contacts app

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
```



packages/providers/ContactsProvider/src/com/android/providers/contacts/ContactsProvider2.java

Using CountDownLatch in Android

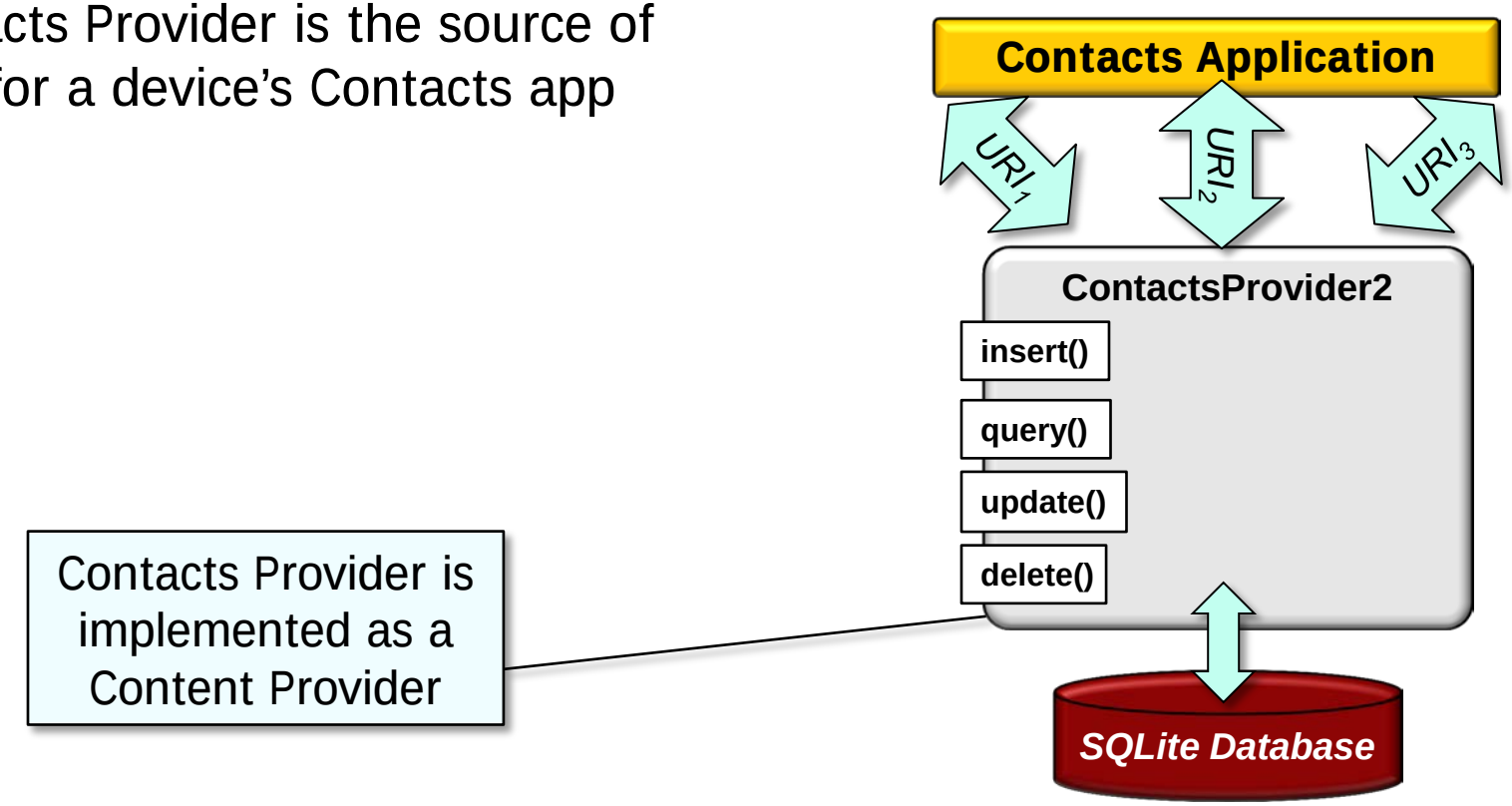
- Contacts Provider is the source of data for a device's Contacts app

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
```

packages/providers/ContactsProvider/src/com/android/providers/contacts/ContactsProvider2.java

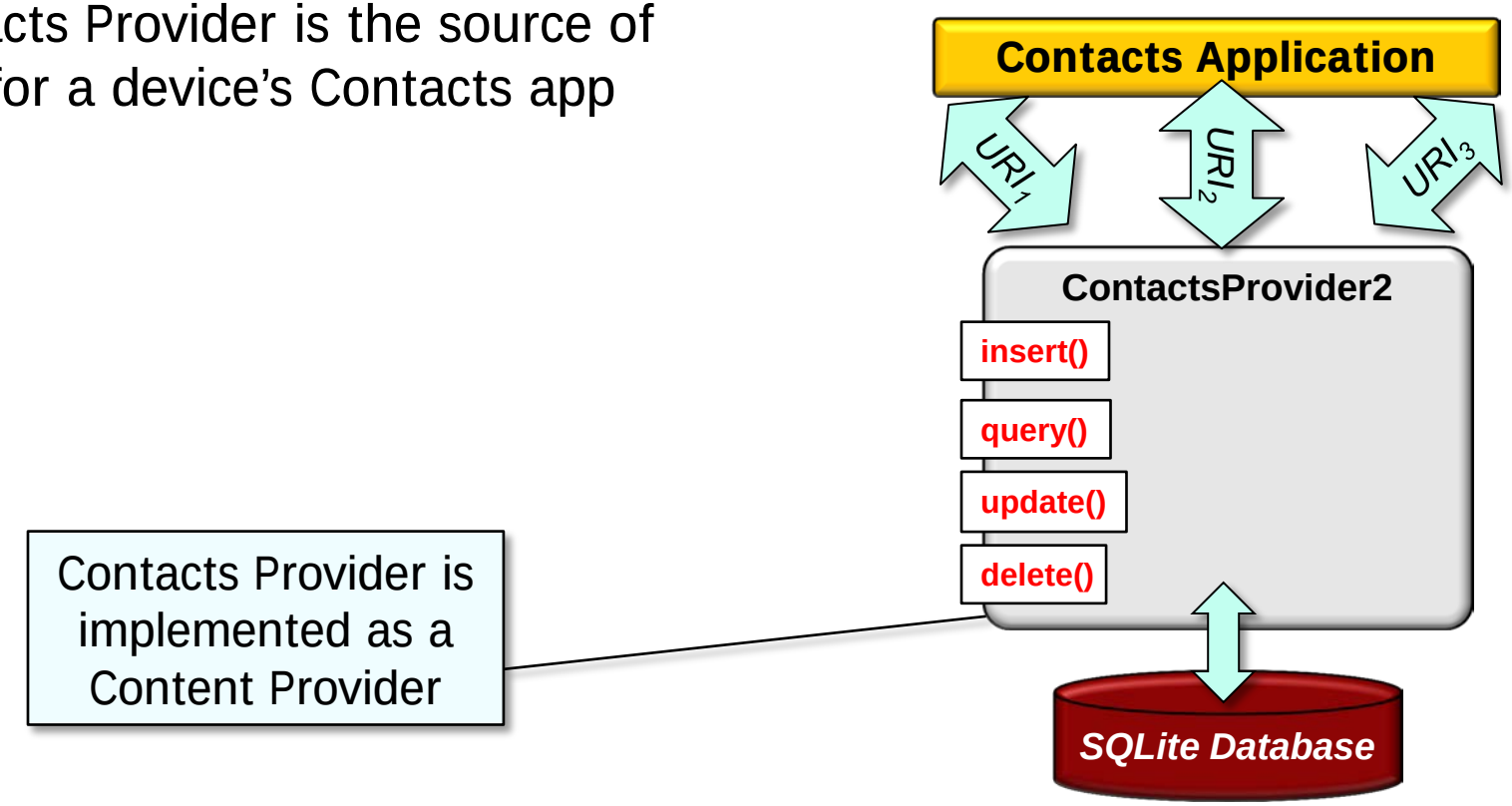
Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app



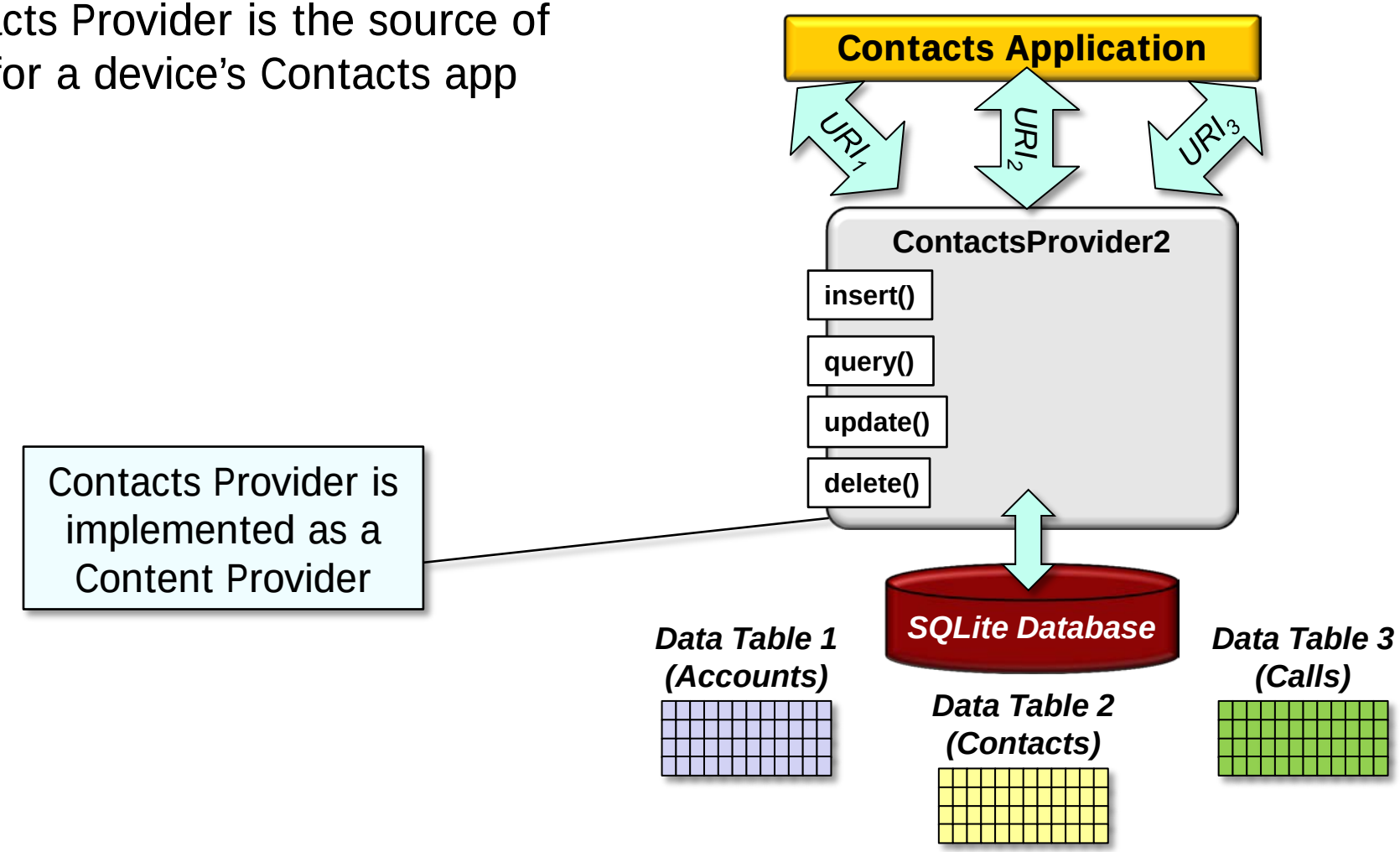
Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app



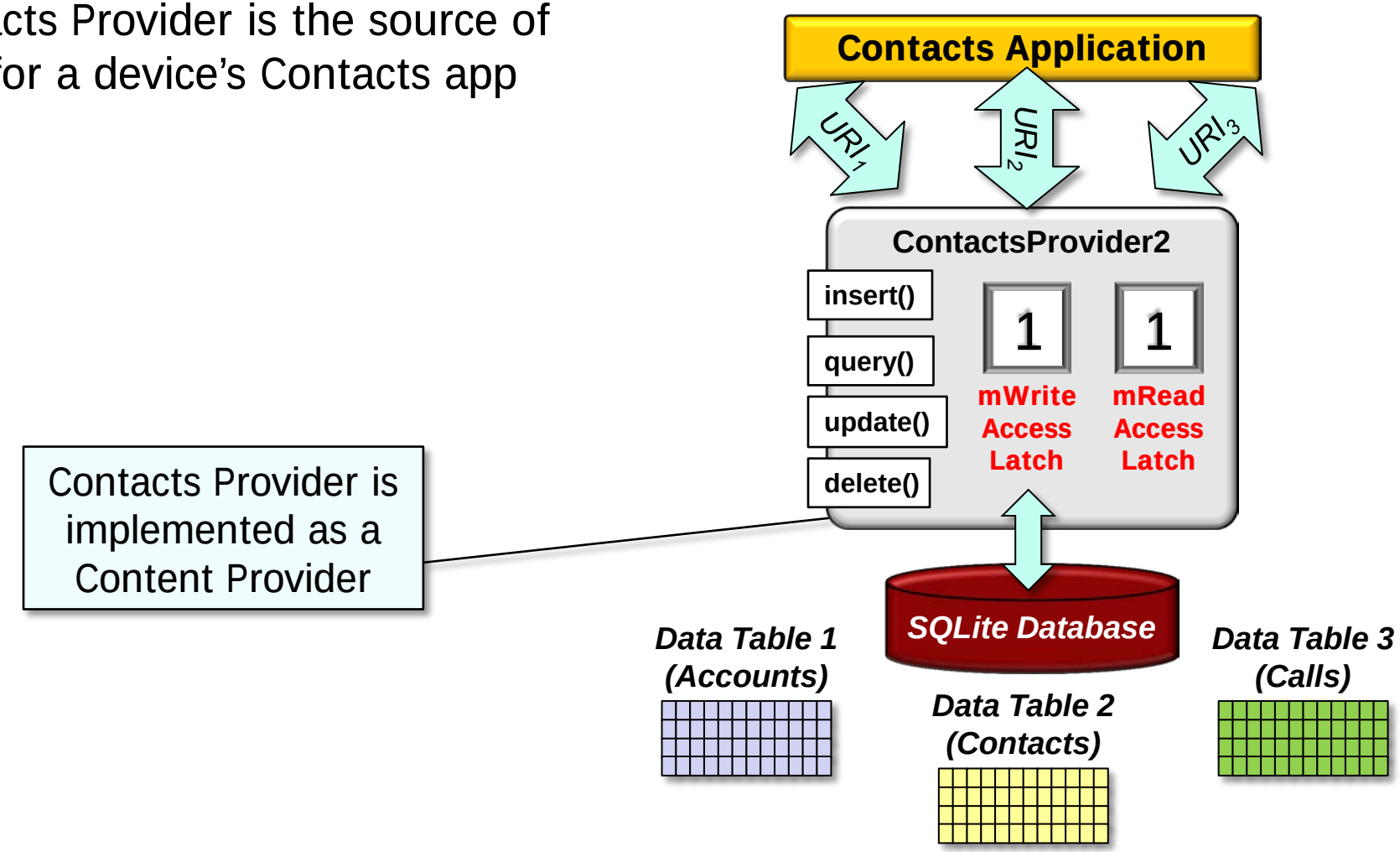
Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app



Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app



Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  private boolean onCreate() {
    ...
    return initialize();
    ...
```

ContactsProvider2

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
    - initialize() starts a worker thread

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  private boolean initialize() {
    ...
```

```
  mBackgroundThread =
    new HandlerThread
     ("ContactsProviderWorker");
  mBackgroundThread.start();
  ...
```

ContactsProvider2

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
    - initialize() starts a worker thread
      - Performs SQLite database tasks that continue to run after initialize() has returned

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  private boolean initialize() {
    ...
```

ContactsProvider2

```
  mBackgroundThread =
    new HandlerThread
     ("ContactsProviderWorker");
  mBackgroundThread.start();
  ...
  scheduleBackgroundTask(...);
  scheduleBackgroundTask(...);
  ...
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
    - initialize() starts a worker thread
    - Methods must not read from or write to the database until it's completely initialized

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  private boolean initialize() {
    ...
```

ContactsProvider2

```
  mBackgroundThread =
    new HandlerThread
     ("ContactsProviderWorker");
  mBackgroundThread.start();
  ...
  scheduleBackgroundTask(...);
  scheduleBackgroundTask(...);
  ...
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization

ContactsProvider2

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  private boolean initialize() {
    ...
    mReadAccessLatch =
      new CountDownLatch(1);
    mWriteAccessLatch =
      new CountDownLatch(1);
    ...
    mBackgroundThread =
      new HandlerThread
       ("ContactsProviderWorker");
    mBackgroundThread.start();
    ...
    scheduleBackgroundTask(...);
    scheduleBackgroundTask(...);
    ...
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
    - Read & write access is split to optimize performance

ContactsProvider2

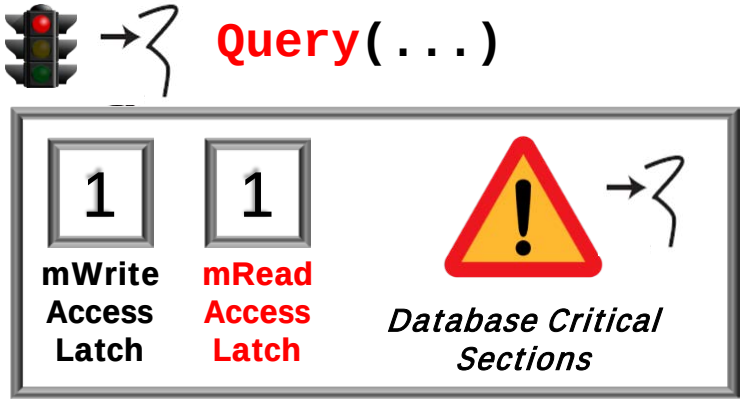
```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  public Cursor query(...) {
   waitForAccess
     (mReadAccessLatch);
     ...
  }

  public Uri insert(Uri uri,
            ContentValues values) {
    waitForAccess
      (mWriteAccessLatch);
    ...
  }

  ...
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
    - Read & write access is split to optimize performance



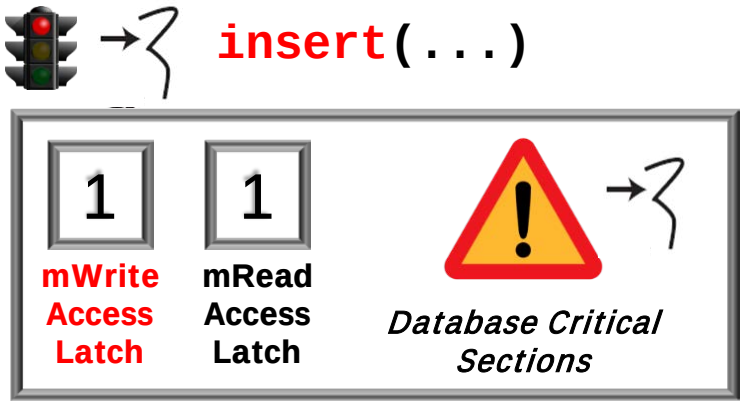

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  public Cursor query(...) {
   waitForAccess
     (mReadAccessLatch);
     ...
  }

  public Uri insert(Uri uri,
            ContentValues values) {
    waitForAccess
      (mWriteAccessLatch);
    ...
  }

  ...
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
    - Read & write access is split to optimize performance




```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  public Cursor query(...) {
   waitForAccess
     (mReadAccessLatch);
     ...
  }

  public Uri insert(Uri uri,
            ContentValues values) {
    waitForAccess
      (mWriteAccessLatch);
    ...
  }

  ...
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
  - waitForAccess() blocks operations until initialization is complete

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  private void waitForAccess
    (CountDownLatch latch) {
    while (true) {
      ...
      latch.await();
      return;
    ...
```

ContactsProvider2

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
  - waitForAccess() blocks operations until initialization is complete

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  private void waitForAccess
    (CountDownLatch latch) {
    while (true) {
      ...
      latch.await();
      return;
    ...
```

ContactsProvider2

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
  - waitForAccess() blocks operations until initialization is complete
  - performBackgroundTask() calls countDown() on CountDownLatches

ContactsProvider2

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  protected void
    performBackgroundTask
      (int task, ...) {
    switch (task) {
    case ..._INITIALIZE:
      ...
      initForDefaultLocale();
      mReadAccessLatch.
        countDown();
      ...
    case ..._OPEN_WRITE_ACCESS:
      ...
      mWriteAccessLatch.
        countDown();
      ...
```

Using CountDownLatch in Android

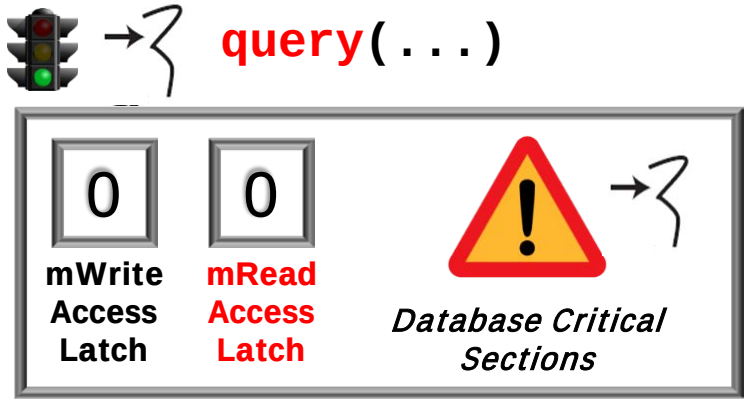
- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
  - waitForAccess() blocks operations until initialization is complete
  - performBackgroundTask() calls countDown() on CountDownLatches

ContactsProvider2

```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  protected void
    performBackgroundTask
      (int task, ...) {
    switch (task) {
    case ..._INITIALIZE:
      ...
      initForDefaultLocale();
      mReadAccessLatch.
        countDown();
      ...
    case ..._OPEN_WRITE_ACCESS:
      ...
      mWriteAccessLatch.
        countDown();
      ...
```

Using CountDownLatch in Android

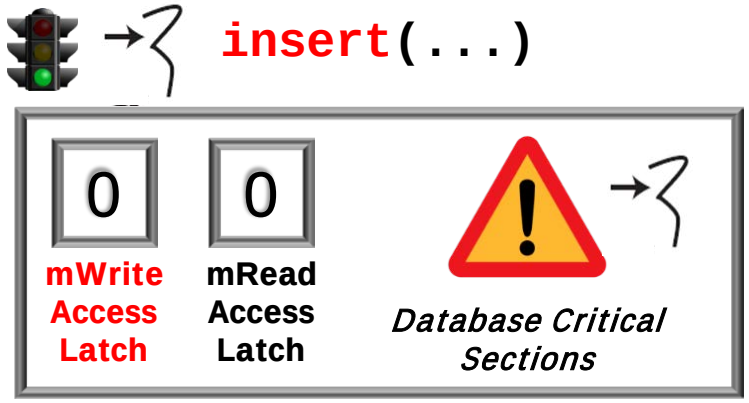
- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
  - waitForAccess() blocks operations until initialization is complete
  - performBackgroundTask() calls countDown() on CountDownLatches




```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  protected void
    performBackgroundTask
      (int task, ...) {
    switch (task) {
    case ..._INITIALIZE:
      ...
      initForDefaultLocale();
      mReadAccessLatch.
        countDown();
      ...
    case ..._OPEN_WRITE_ACCESS:
      ...
      mWriteAccessLatch.
        countDown();
      ...
```

Using CountDownLatch in Android

- Contacts Provider is the source of data for a device's Contacts app
  - onCreate() calls initialize()
  - CountDownLatches prevent access to database during initialization
  - waitForAccess() blocks operations until initialization is complete
  - performBackgroundTask() calls countDown() on CountDownLatches




```
public class ContactsProvider2
  extends AbstractContactsProvider
  ... {
  ...
  protected void
    performBackgroundTask
      (int task, ...) {
    switch (task) {
    case ..._INITIALIZE:
      ...
      initForDefaultLocale();
      mReadAccessLatch.
        countDown();
      ...
    case ..._OPEN_WRITE_ACCESS:
      ...
      mWriteAccessLatch.
        countDown();
      ...
```

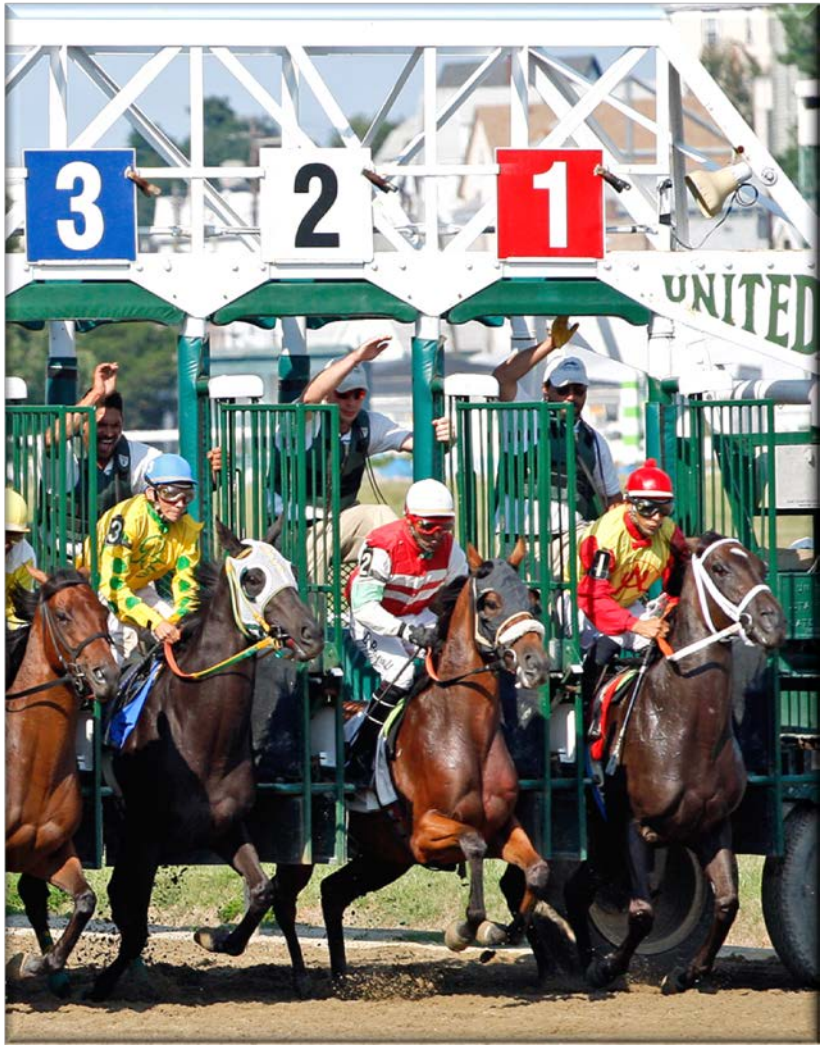
Summary

Summary

- A CountDownLatch is a versatile barrier synchronization tool
  - It can be used for several purposes

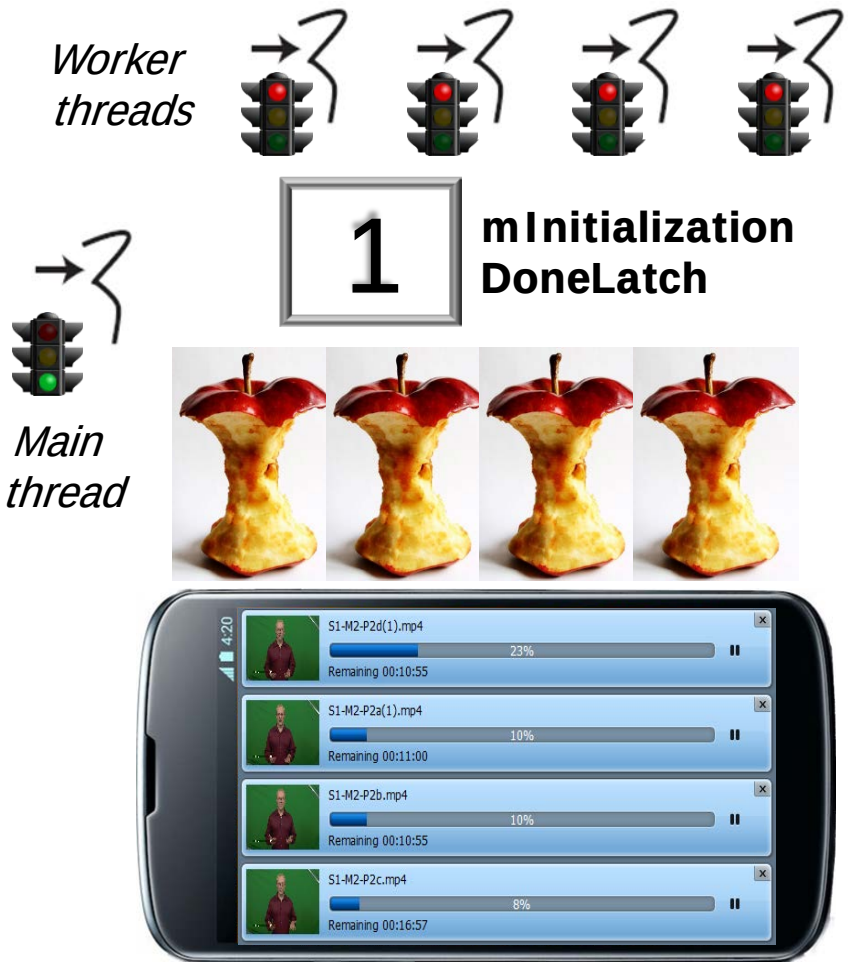
Summary

- A CountDownLatch is a versatile barrier synchronization tool
  - It can be used for several purposes
    - It can be a simple on/off latch



Summary

- A CountDownLatch is a versatile barrier synchronization tool
  - It can be used for several purposes
    - It can be a simple on/off latch
      - e.g., all video conversion threads invoking await() block at the latch until the main thread invokes countDown()

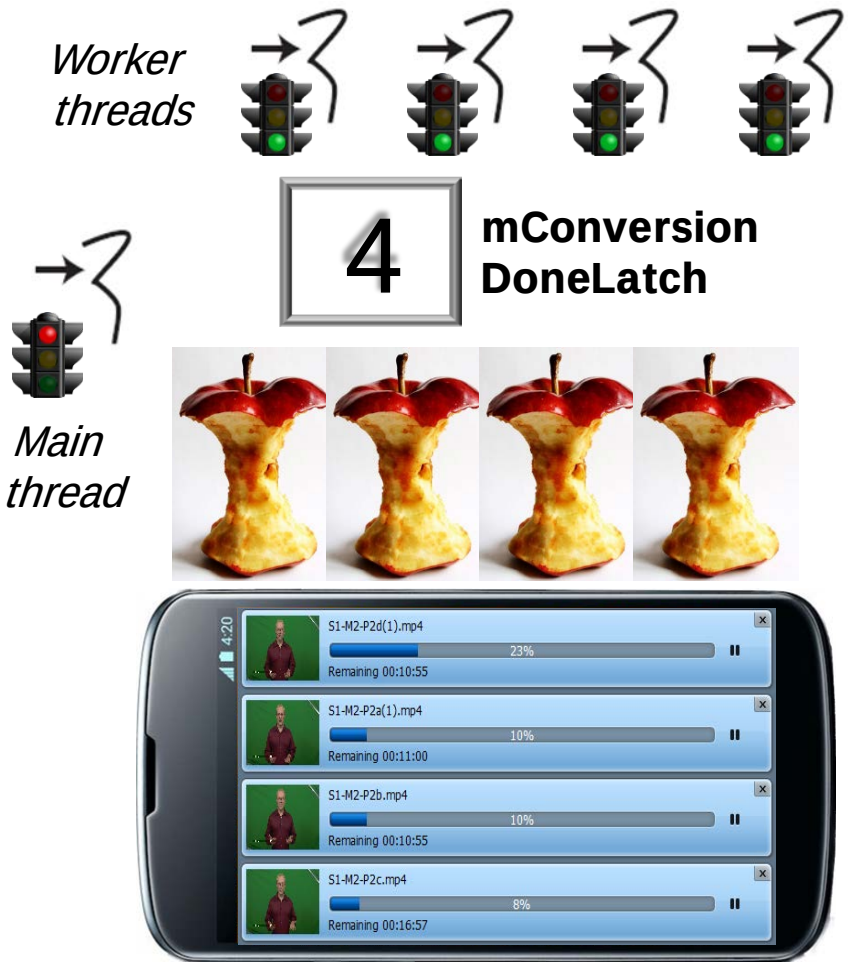


Summary

- A CountDownLatch is a versatile barrier synchronization tool
  - It can be used for several purposes
    - It can be a simple on/off latch
    - It can make one thread wait until *N* threads have completed some action or some action has been completed N times

Summary

- A CountDownLatch is a versatile barrier synchronization tool
  - It can be used for several purposes
    - It can be a simple on/off latch
    - It can make one thread wait until *N* threads have completed some action or some action has been completed N times
      - e.g., the main thread waits until the worker threads are finished converting the video



Summary

- A CountDownLatch is a versatile barrier synchronization tool
  - It can be used for several purposes
  - It supports several types of await operations
    - e.g., interruptible & timed operations

await ()

Causes the current thread to wait until the latch has counted down to zero, unless the thread is `interrupted`.

await (long timeout, TimeUnit unit)

Causes the current thread to wait until the latch has counted down to zero, unless the thread is `interrupted`, or the specified waiting time elapses.

Summary

- A CountDownLatch is a versatile barrier synchronization tool
- CountDownLatch is used throughout Android

frameworks/base/core/java/android/app/SharedPreferencesImpl.java
frameworks/base/core/java/android/content/AsyncTaskLoader.java
frameworks/base/core/java/android/content/SyncManager.java
frameworks/base/core/java/android/inputmethodservice/IInputMethodWrapper.java
frameworks/base/core/java/android/view/inputmethod/InputMethodManager.java
frameworks/base/core/java/android/view/ViewDebug.java
frameworks/base/services/java/com/android/server/location/GpsLocationProvider.java
frameworks/base/services/java/com/android/server/NetworkManagementService.java
frameworks/base/tests/DumpRenderTree/src/com/android/dumprendertree/LoadTestsAutoTest.java
frameworks/ex/variablespeed/src/com/android/ex/variablespeed/VariableSpeed.java
packages/apps/Browser/src/com/android/browser/AutofillHandler.java
packages/apps/Browser/src/com/android/browser/NfcHandler.java
packages/apps/Browser/tests/src/com/android/browser/PopularUrlsTest.java
packages/apps/Contacts/src/com/android/contacts/model/AccountTypeManager.java
packages/apps/Contacts/tests/src/com/android/contacts/util/FakeAsyncTaskExecutor.java
packages/apps/Gallery/src/com/android/camera/CropImage.java
packages/apps/Gallery2/tests/src/com/android/gallery3d/data/LocalDataTest.java
packages/providers/ContactsProvider/src/com/android/providers/contacts/ContactsProvider2.java

Summary

- A CountDownLatch is a versatile barrier synchronization tool
- CountDownLatch is used throughout Android

frameworks/base/core/java/android/app/SharedPreferencesImpl.java
frameworks/base/core/java/android/content/AsyncTaskLoader.java
frameworks/base/core/java/android/content/SyncManager.java
frameworks/base/core/java/android/inputmethodservice/IInputMethodWrapper.java
frameworks/base/core/java/android/view/inputmethod/InputMethodManager.java
frameworks/base/core/java/android/view/ViewDebug.java
frameworks/base/services/java/com/android/server/location/GpsLocationProvider.java
frameworks/base/services/java/com/android/server/NetworkManagementService.java
frameworks/base/tests/DumpRenderTree/src/com/android/dumprendertree/LoadTestsAutoTest.java
frameworks/ex/variablespeed/src/com/android/ex/variablespeed/VariableSpeed.java
packages/apps/Browser/src/com/android/browser/AutofillHandler.java
packages/apps/Browser/src/com/android/browser/NfcHandler.java
packages/apps/Browser/tests/src/com/android/browser/PopularUrlsTest.java
packages/apps/Contacts/src/com/android/contacts/model/AccountTypeManager.java
packages/apps/Contacts/tests/src/com/android/contacts/util/FakeAsyncTaskExecutor.java
packages/apps/Gallery/src/com/android/camera/CropImage.java
packages/apps/Gallery2/tests/src/com/android/gallery3d/data/LocalDataTest.java
packages/providers/ContactsProvider/src/com/android/providers/contacts/ContactsProvider2.java