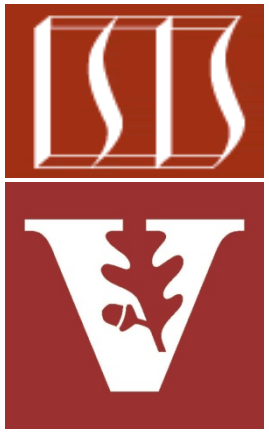
Android Concurrency: The AsyncTask Framework

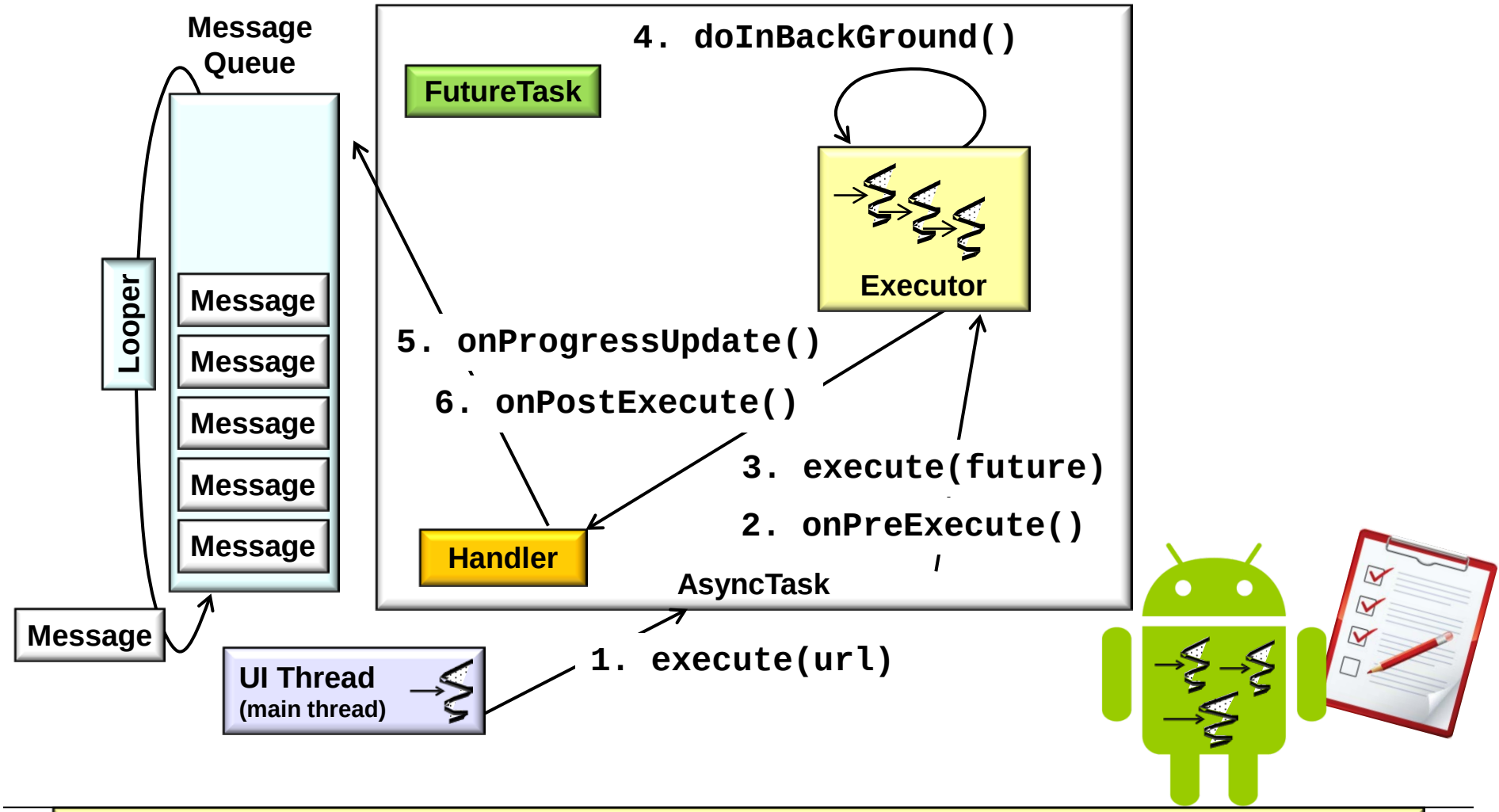


Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

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

Learning Objectives in this Part of the Module

- Recognize the concurrency idioms & mechanisms associated with programming the Android AsyncTask framework

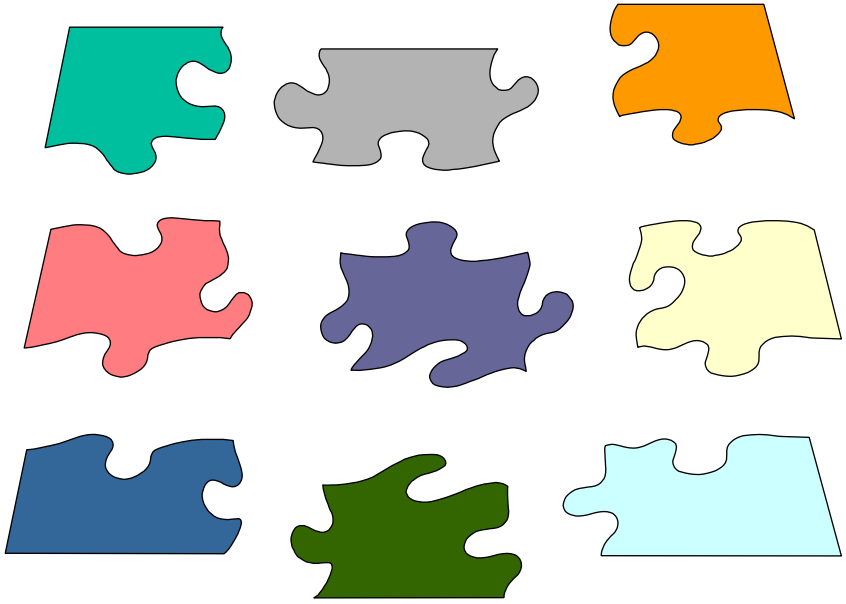


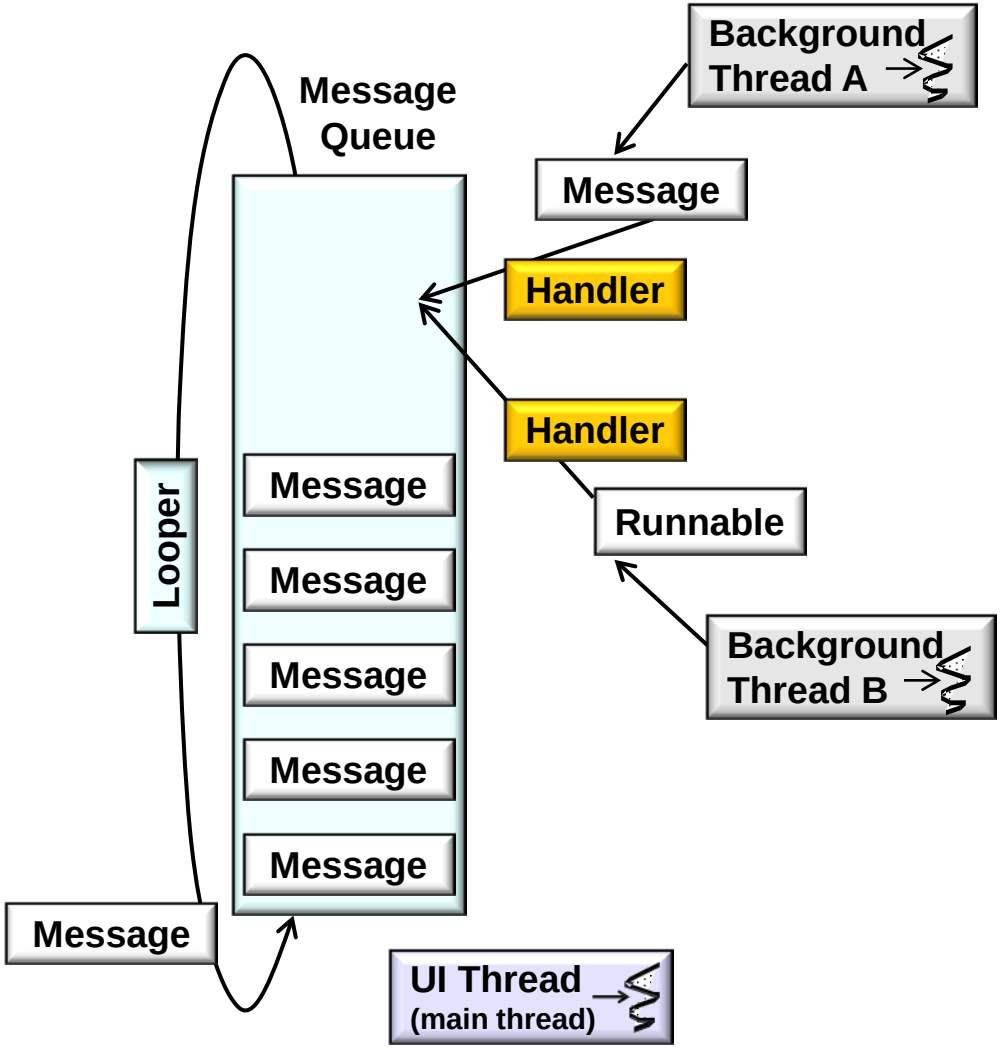
Allows apps to perform background operations & publish results on UI thread *without* manipulating threads, handlers, messages, or runnables

Overview of the AsyncTask Framework

Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected

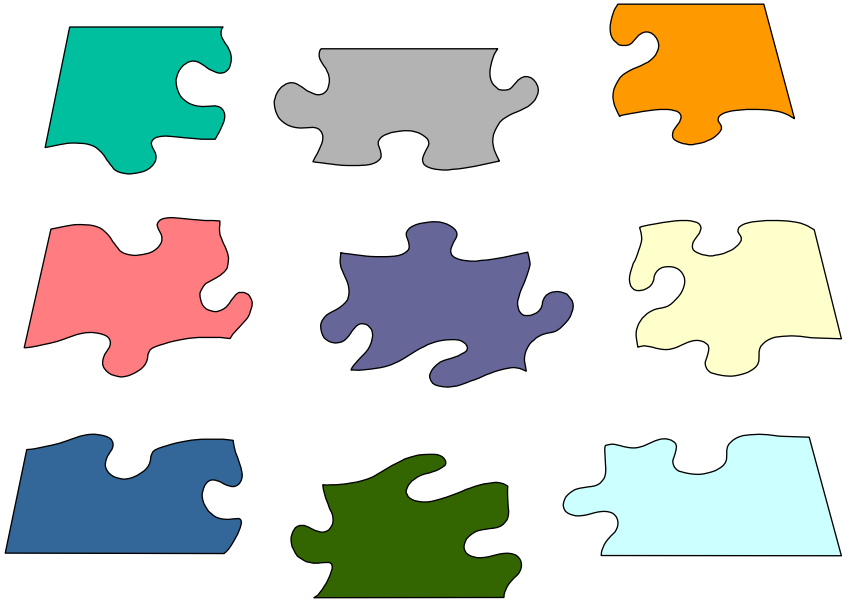


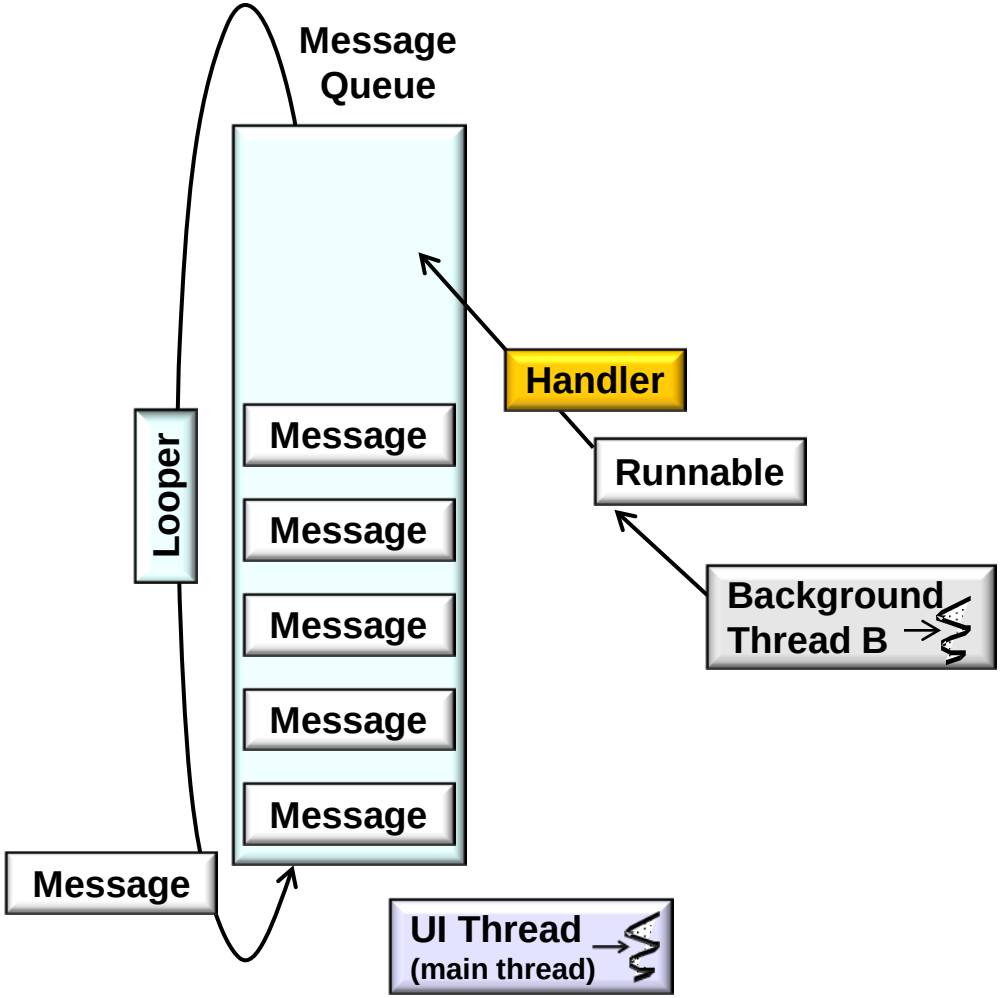


See previous part on the HaMeR framework

Overview of the AsyncTask Framework

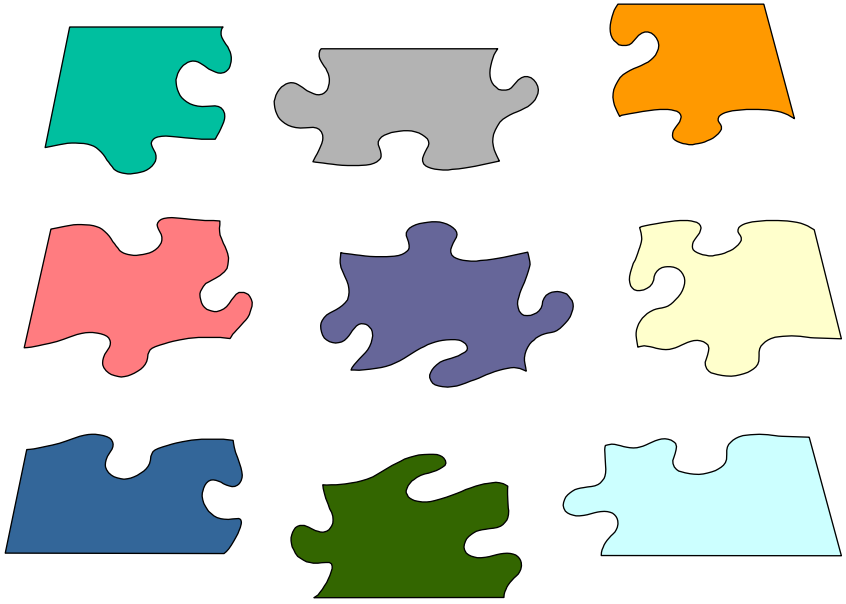
- Classes in HaMeR framework are loosely connected
  - This flexibility works well for simple concurrency use cases

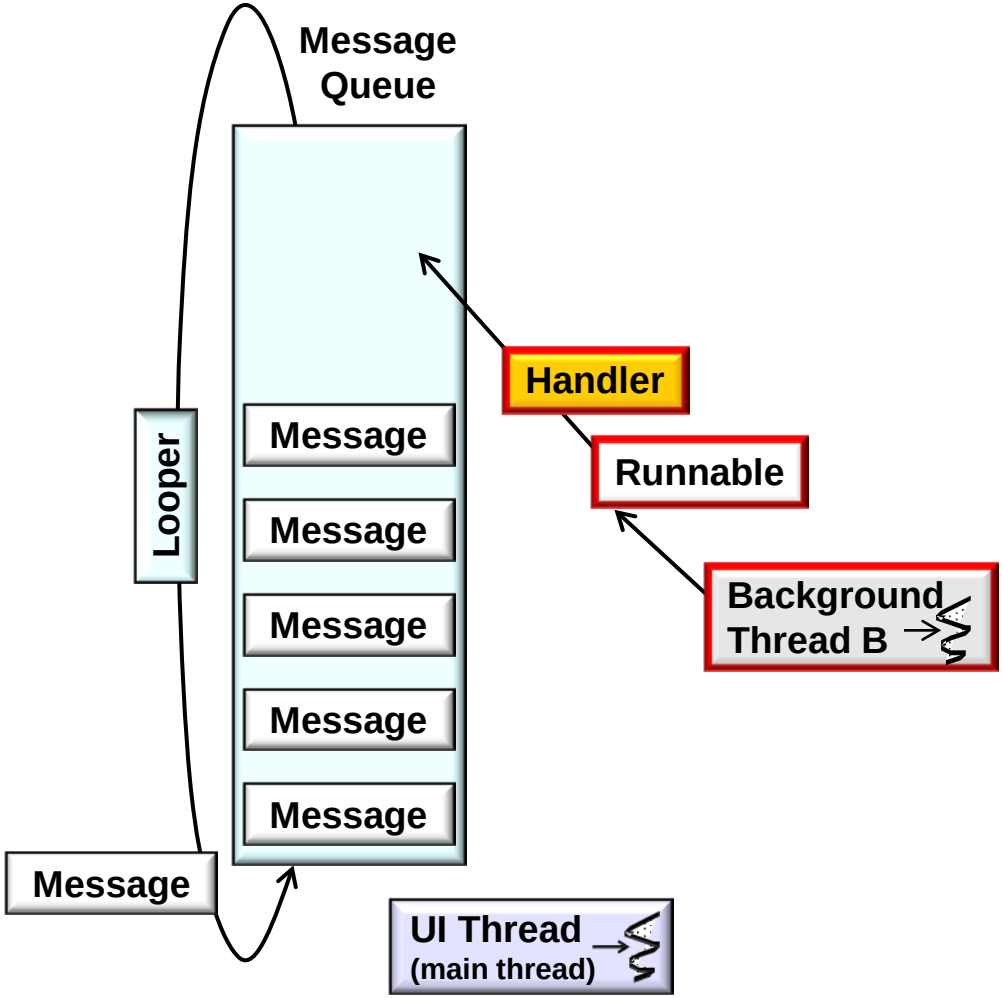




Overview of the AsyncTask Framework

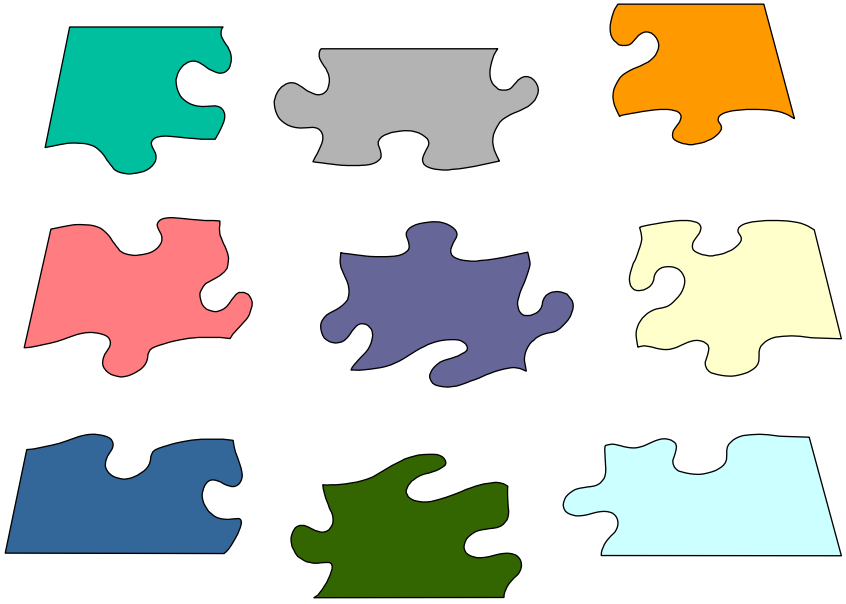
- Classes in HaMeR framework are loosely connected
  - This flexibility works well for simple concurrency use cases

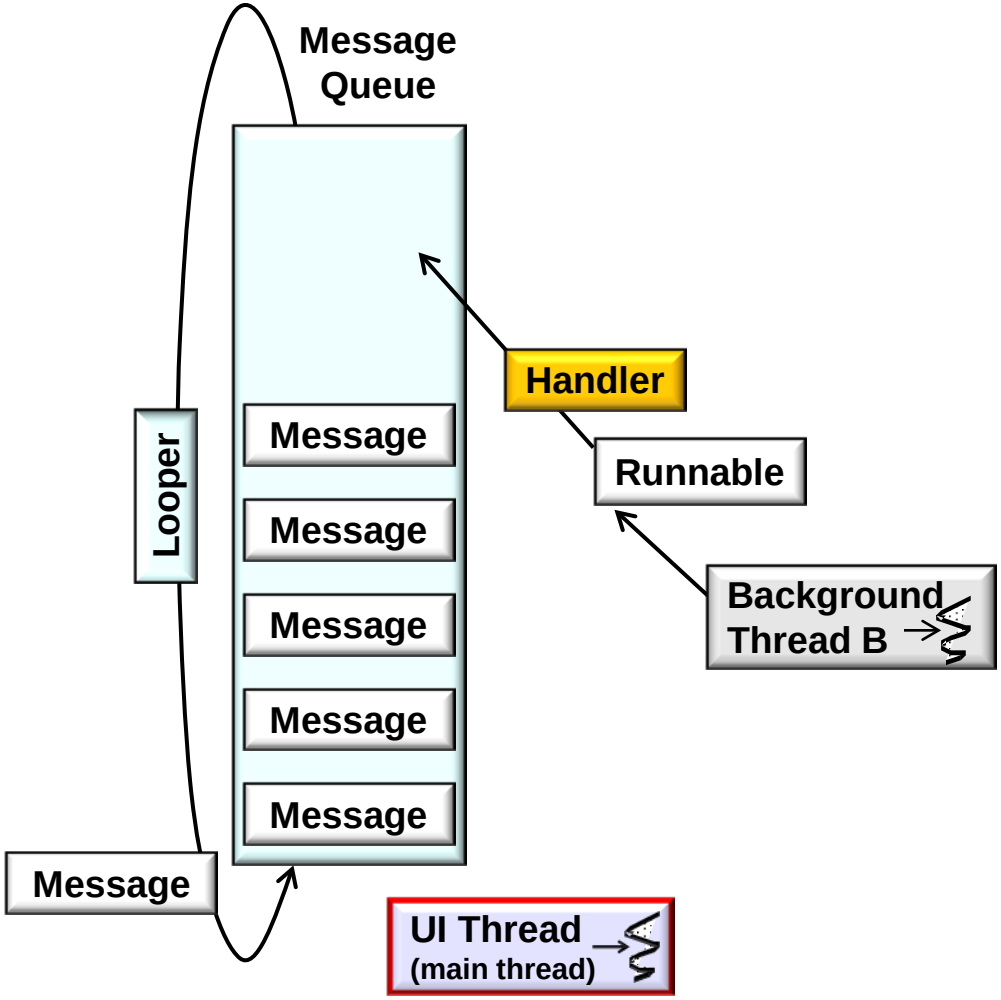




Overview of the AsyncTask Framework

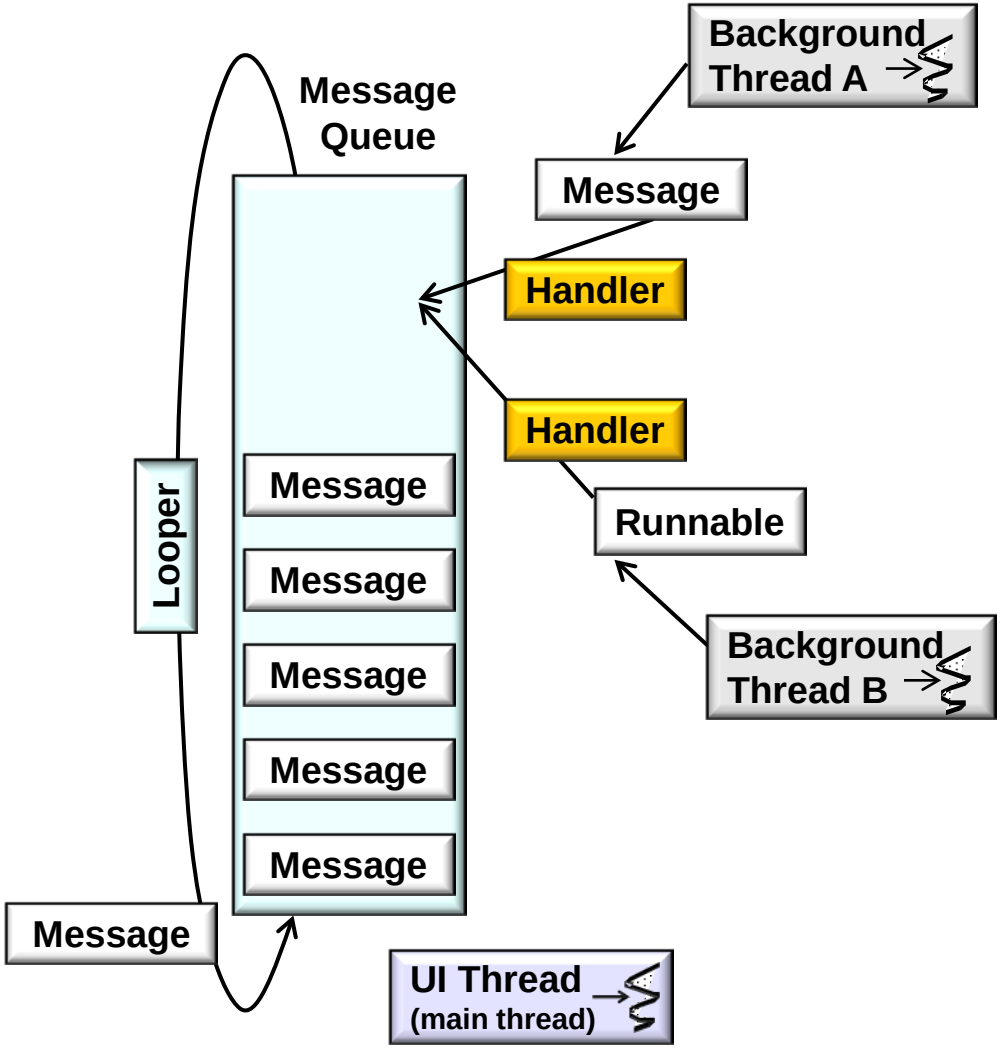
- Classes in HaMeR framework are loosely connected
  - This flexibility works well for simple concurrency use cases





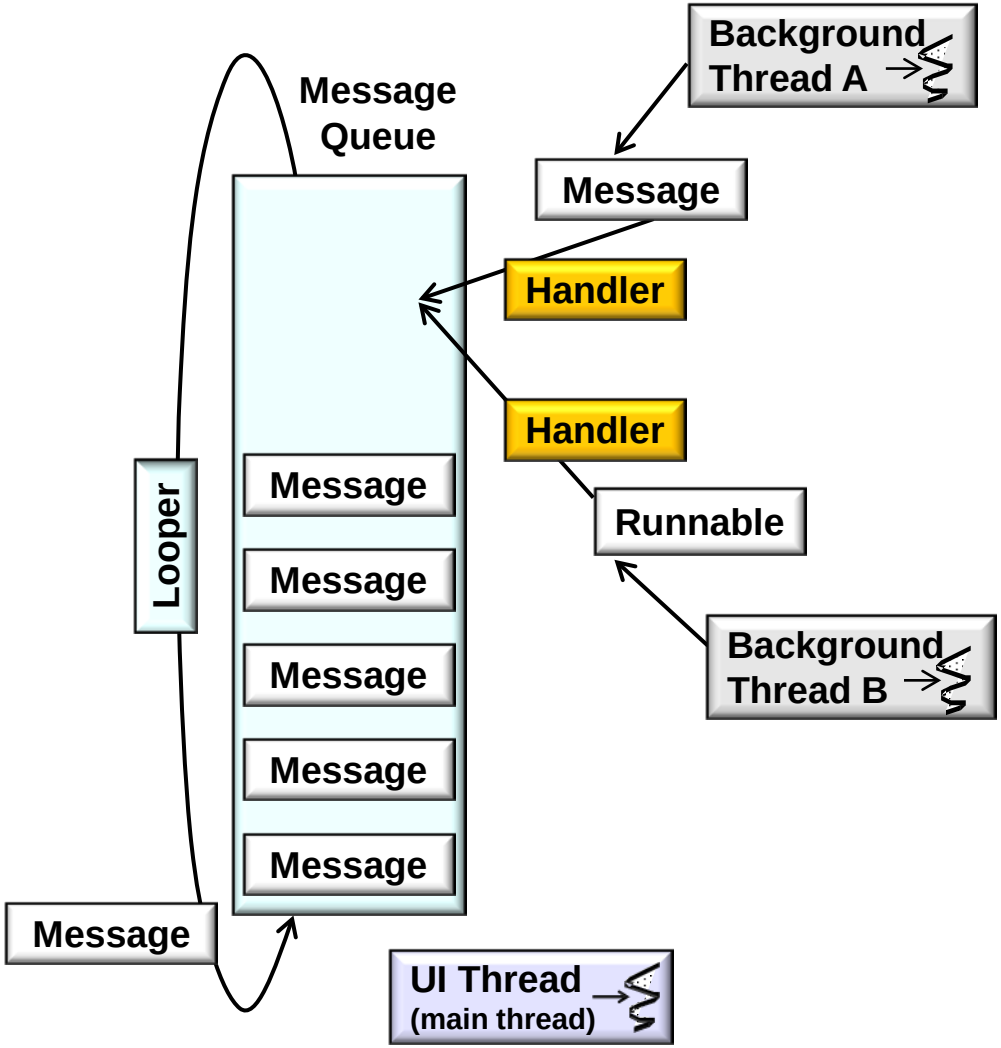
Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected
  - This flexibility works well for simple concurrency use cases
  - However, there are drawbacks



Overview of the AsyncTask Framework

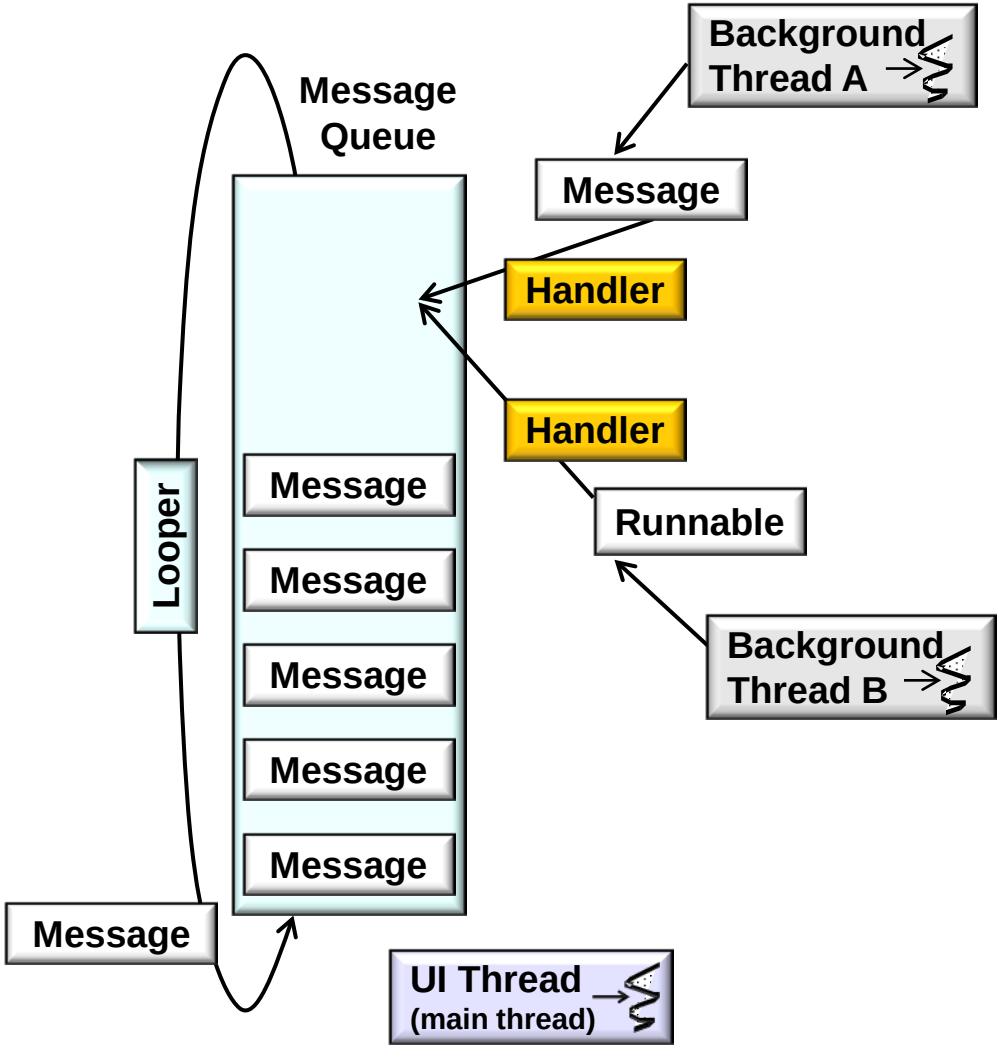
- Classes in HaMeR framework are loosely connected
  - This flexibility works well for simple concurrency use cases
  - However, there are drawbacks
    - Must understand patterns



See en.wikipedia.org/wiki/Active_object & www.dre.vanderbilt.edu/~schmidt/CommandProcessor.pdf

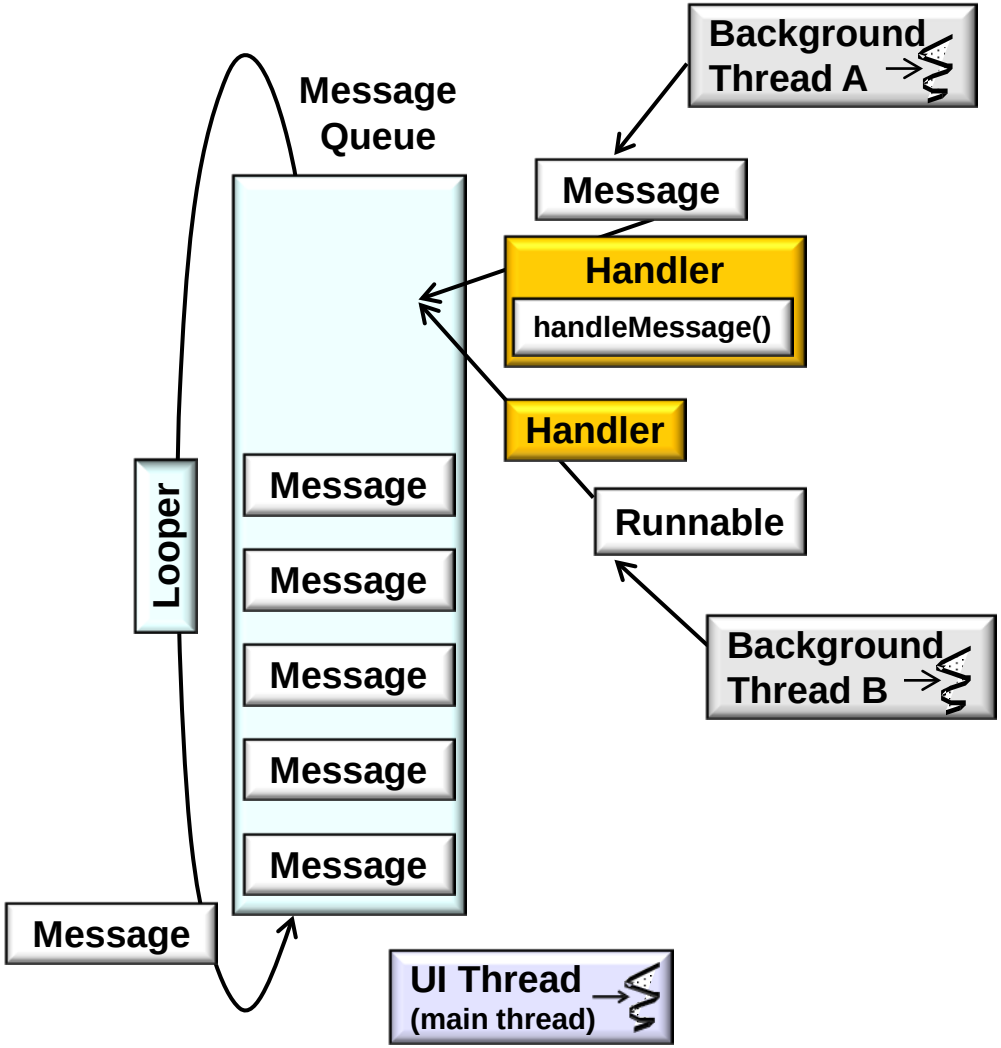
Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected
  - This flexibility works well for simple concurrency use cases
  - However, there are drawbacks
    - Must understand patterns
    - Tedious & error-prone



Overview of the AsyncTask Framework

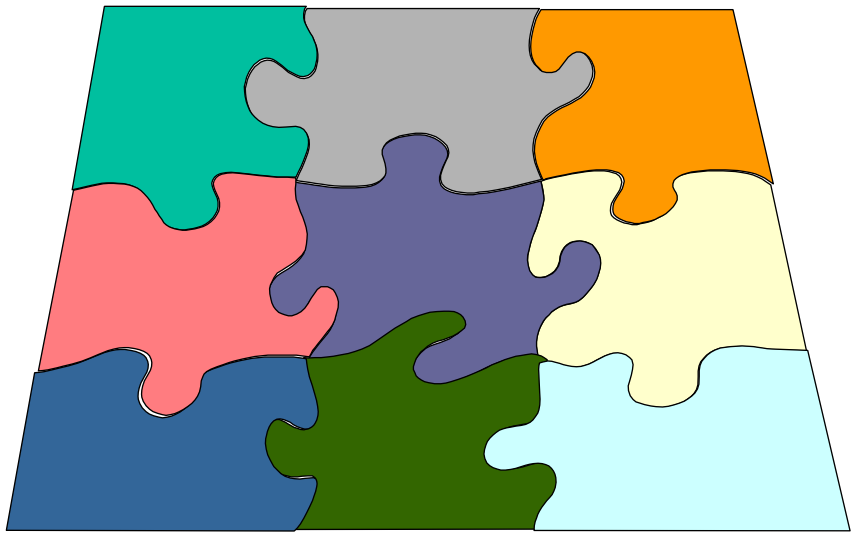
- Classes in HaMeR framework are loosely connected
  - This flexibility works well for simple concurrency use cases
  - However, there are drawbacks
    - Must understand patterns
    - Tedious & error-prone

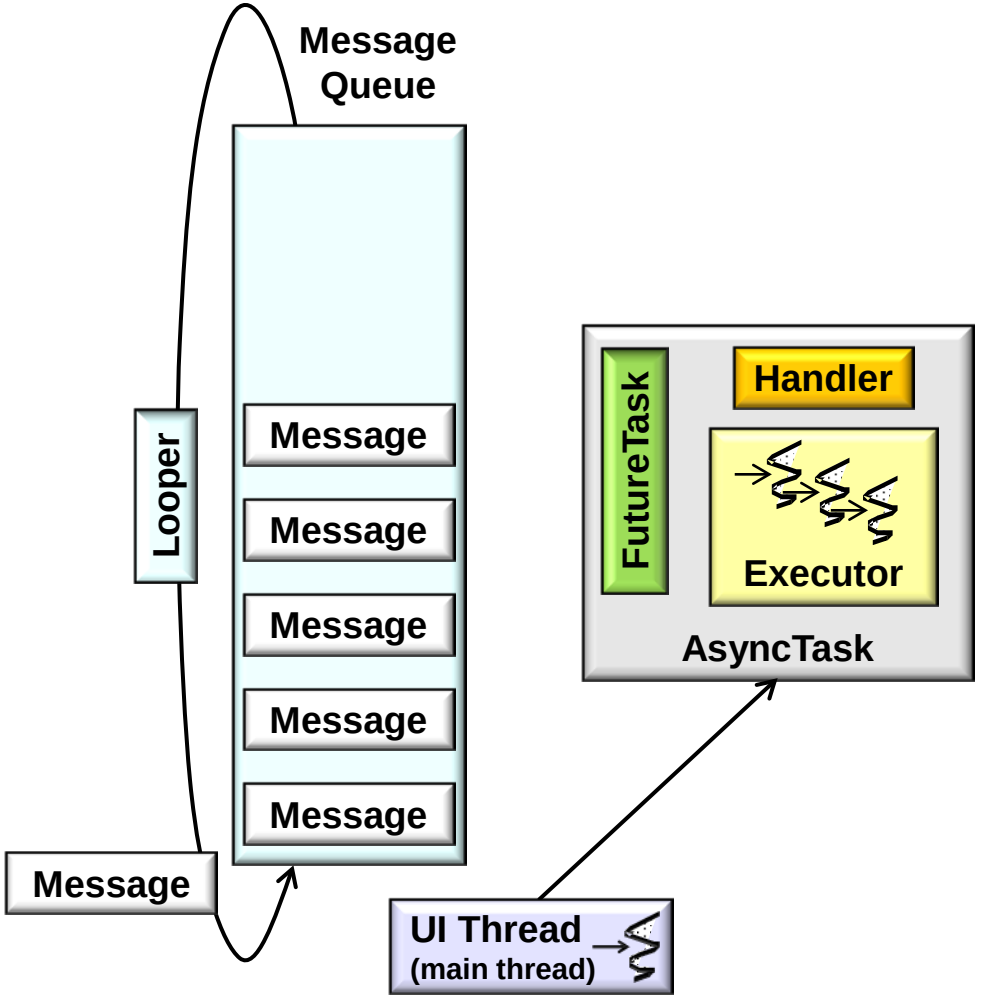


e.g., apps must understand how to manage the lifecycle of Messages

Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected
- Classes in AsyncTask framework are strongly connected

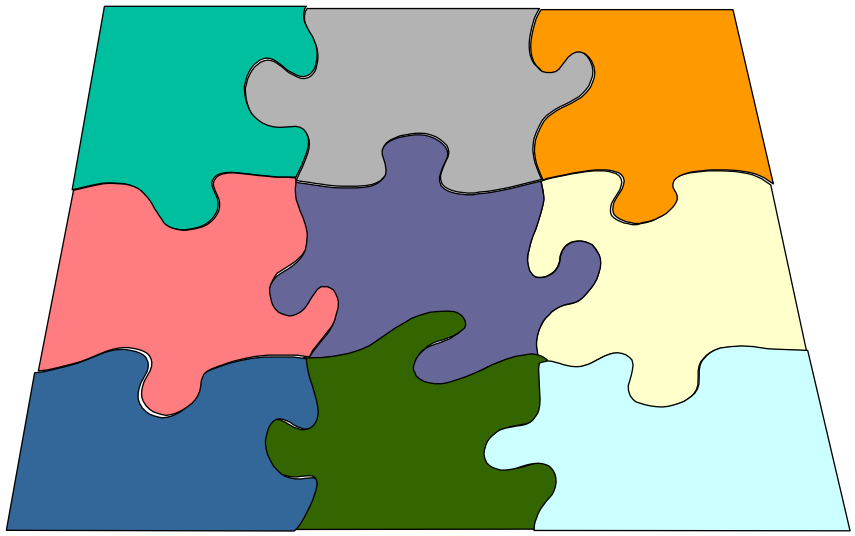


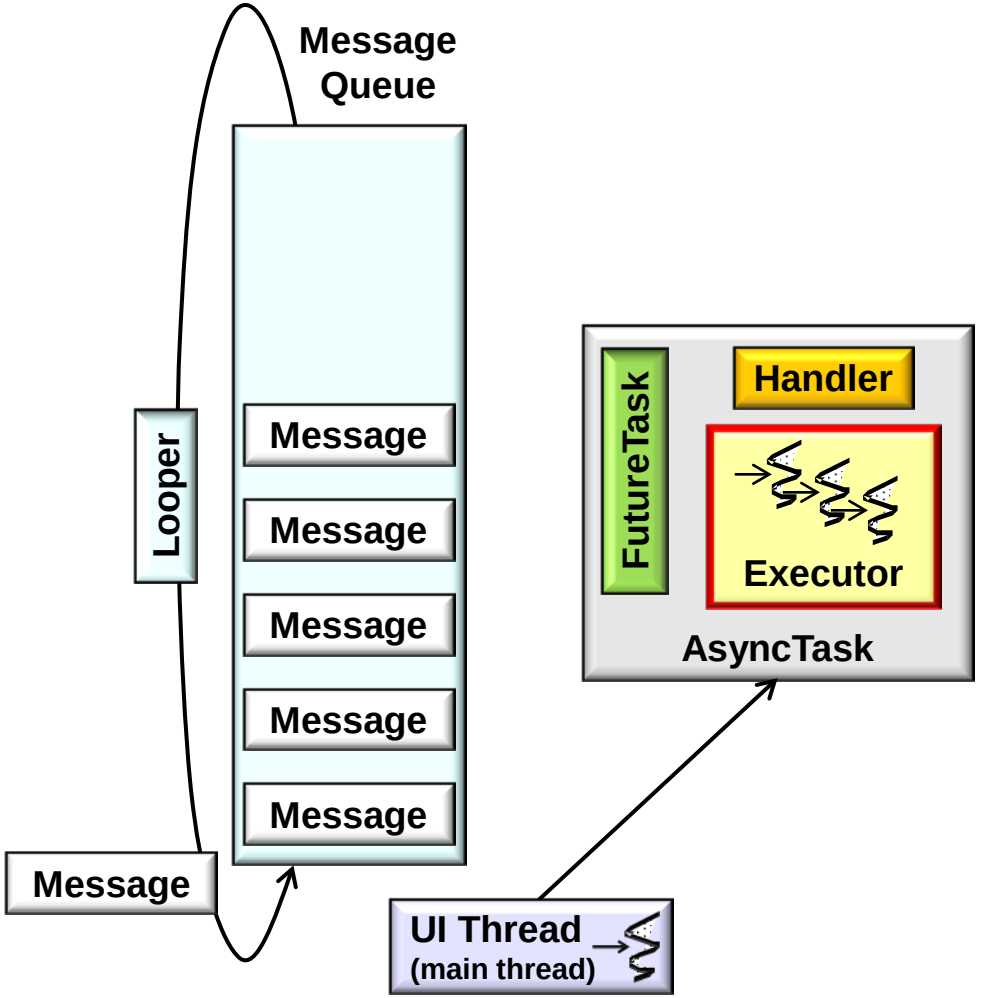


See en.wikipedia.org/wiki/Template_Method_pattern

Overview of the AsyncTask Framework

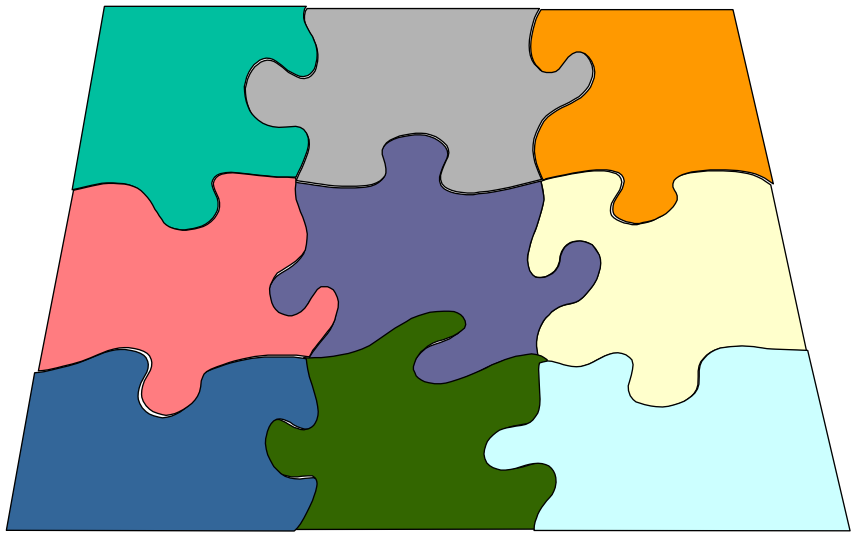
- Classes in HaMeR framework are loosely connected
- Classes in AsyncTask framework are strongly connected

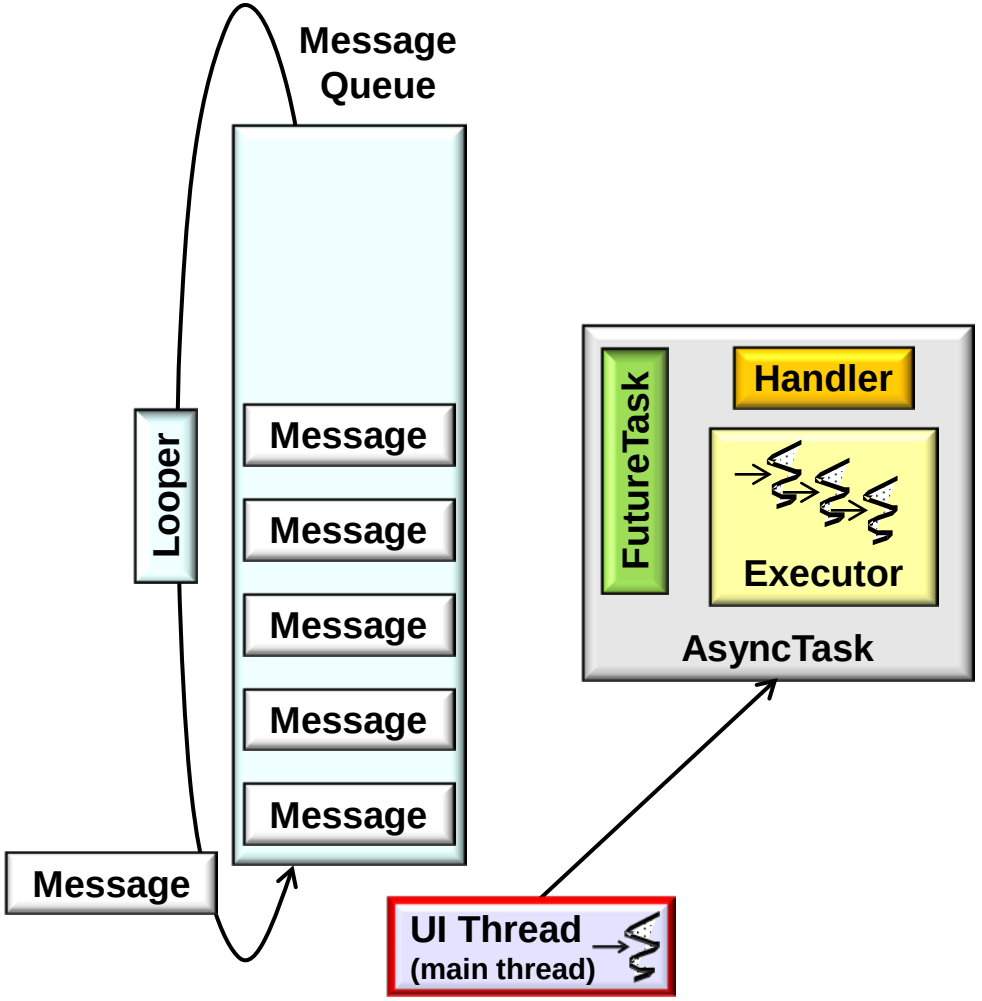




Overview of the AsyncTask Framework

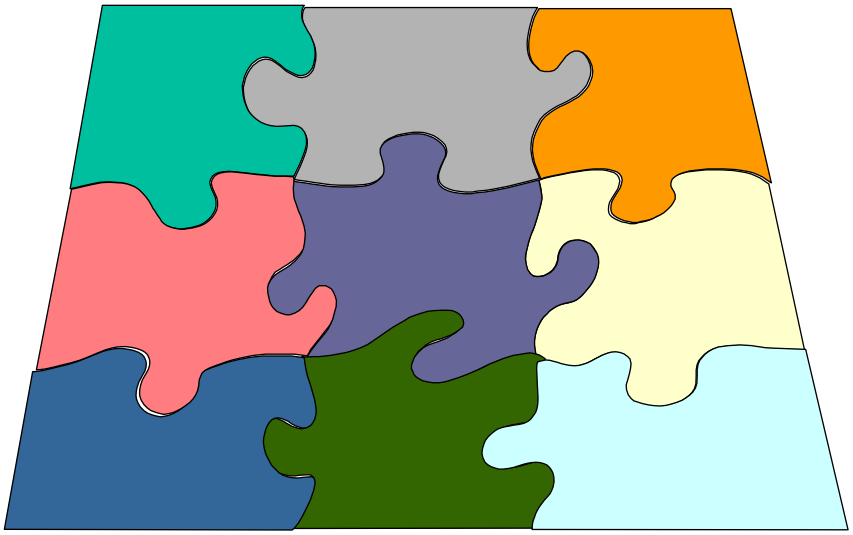
- Classes in HaMeR framework are loosely connected
- Classes in AsyncTask framework are strongly connected

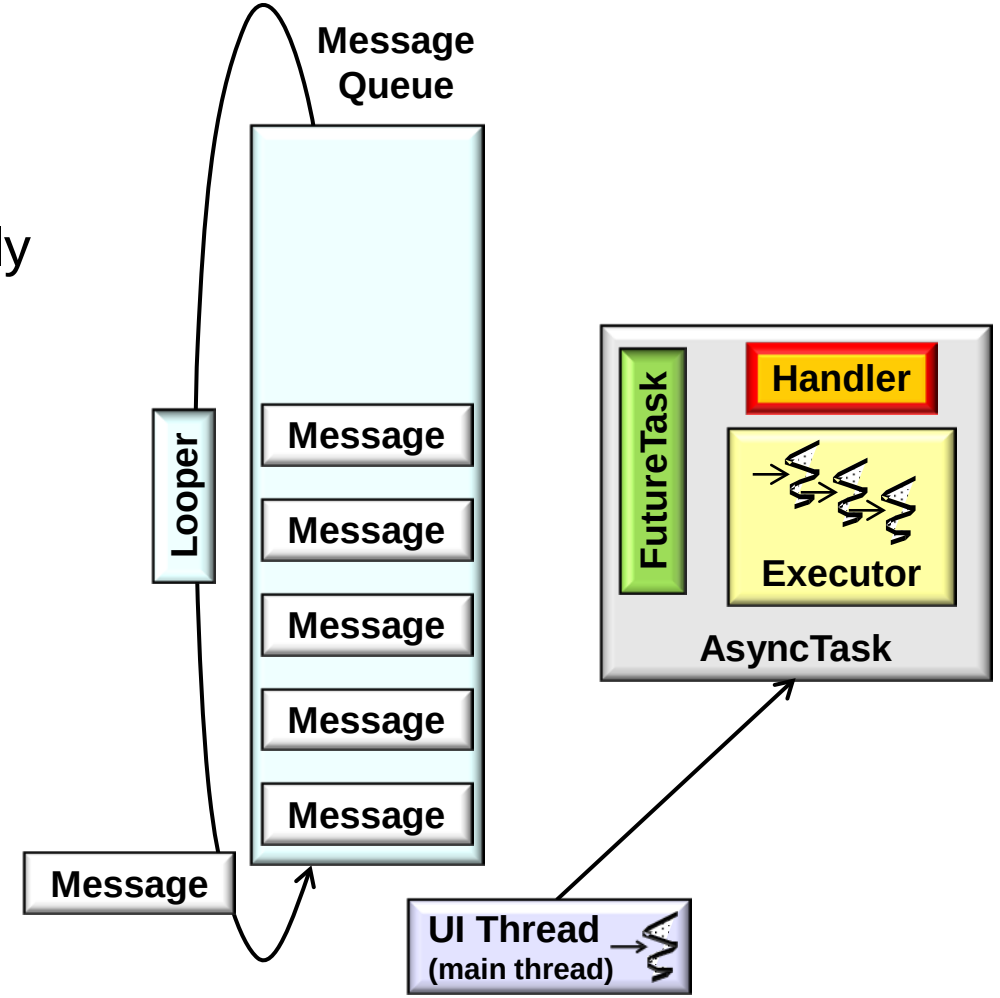




Overview of the AsyncTask Framework

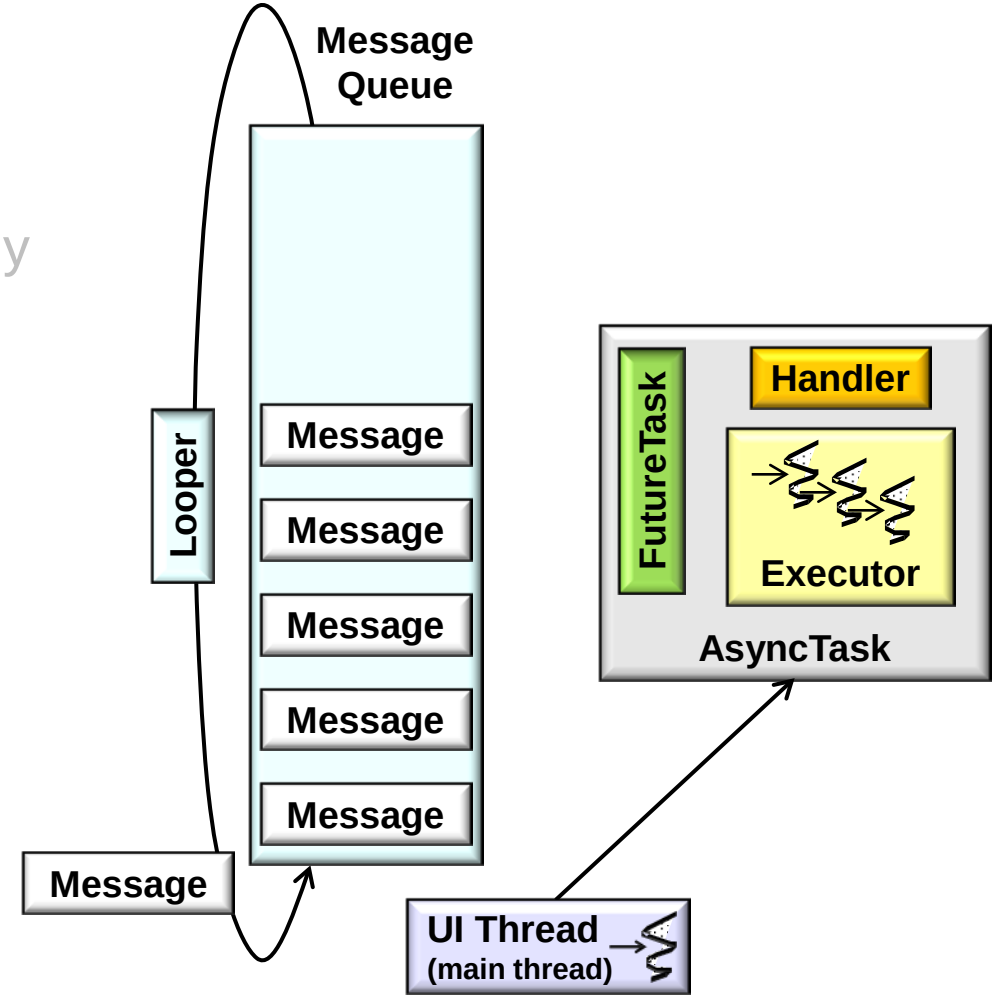
- Classes in HaMeR framework are loosely connected
- Classes in AsyncTask framework are strongly connected
  - Run concurrently, *without* directly manipulating Threads, Handlers, Messages, or Runnables





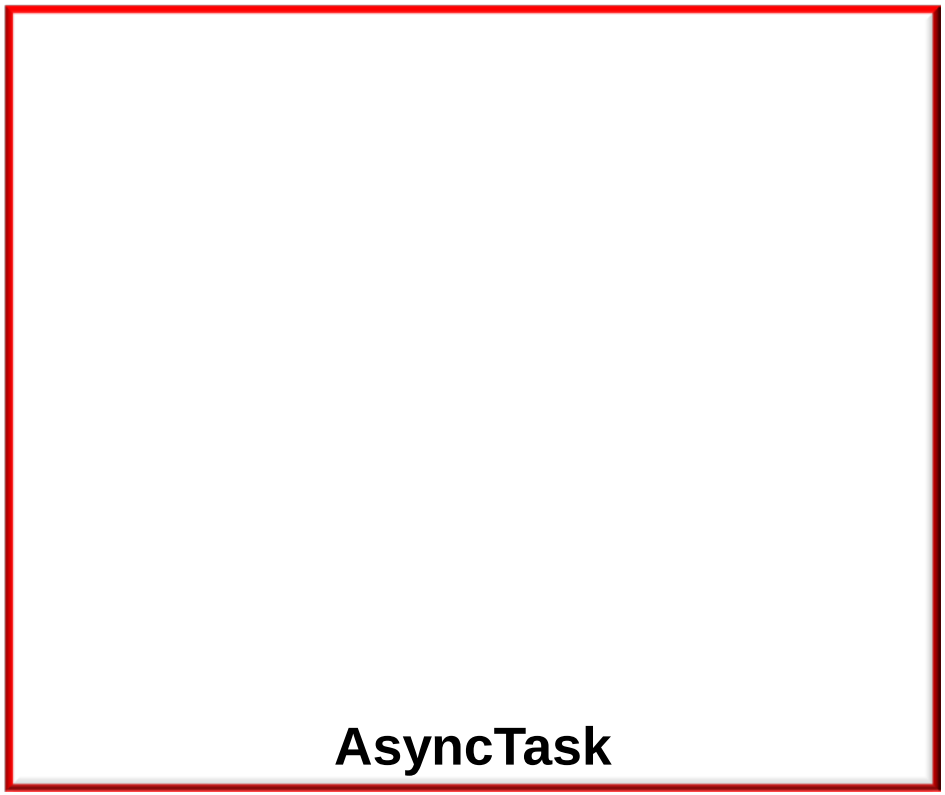
Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected
- Classes in AsyncTask framework are strongly connected
  - Run concurrently, *without* directly manipulating Threads, Handlers, Messages, or Runnables
  - Smaller “surface area”



Overview of the AsyncTask Framework

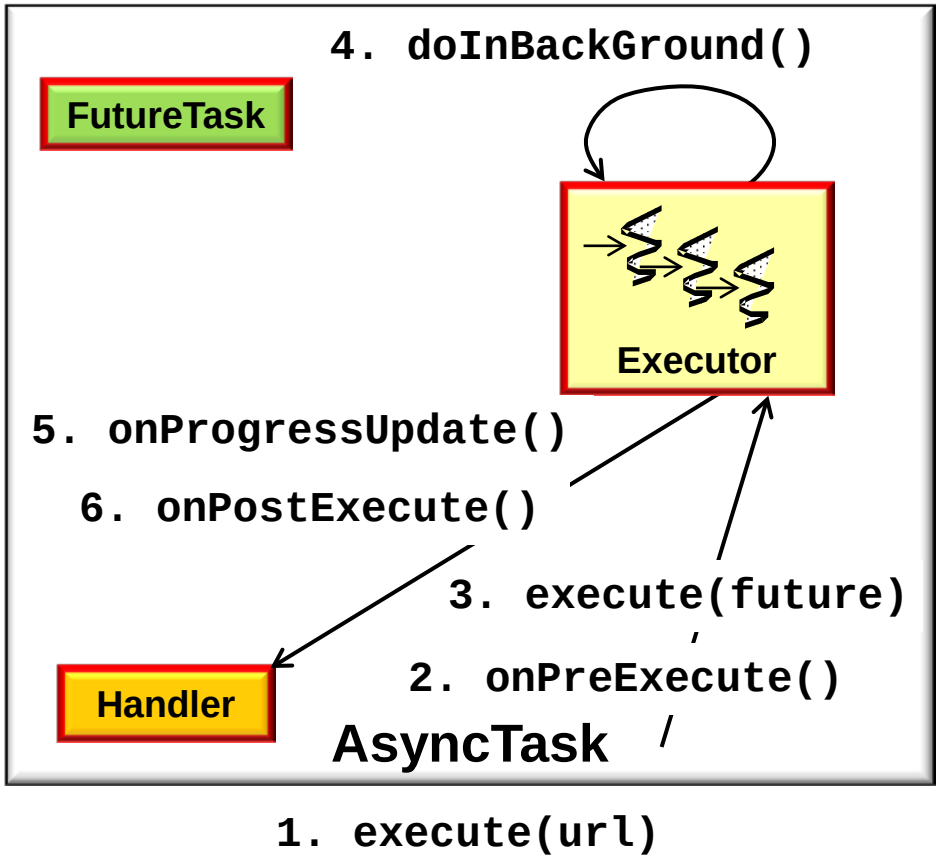
- Classes in HaMeR framework are loosely connected
- Classes in AsyncTask framework are strongly connected
  - Run concurrently, *without* directly manipulating Threads, Handlers, Messages, or Runnables
  - Smaller "surface area"
  - Complex framework details accessed via *Façade* pattern



See en.wikipedia.org/wiki/Facade_pattern

Overview of the AsyncTask Framework

- Classes in HaMeR framework are loosely connected
- Classes in AsyncTask framework are strongly connected
  - Run concurrently, *without* directly manipulating Threads, Handlers, Messages, or Runnables
  - Smaller "surface area"
  - Complex framework details accessed via *Façade* pattern
    - Wraps complicated subsystem or framework with simpler interface



Categories of Methods in AsyncTask

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods

AsyncTask

Added in API level 3

extends Object

java.lang.Object
↳android.os.AsyncTask<Params, Progress, Result>

Class Overview

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a helper class around `Thread` and `Handler` and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the `java.util.concurrent` pacakge such as `Executor`, `ThreadPoolExecutor` and `FutureTask`.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called `Params`, `Progress` and `Result`, and 4 steps, called `onPreExecute`, `doInBackground`, `onProgressUpdate` and `onPostExecute`.

See developer.android.com/reference/android/os/AsyncTask.html

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
    - Invoked by apps

`AsyncTask<Params, Progress, Result> execute(Params... params)`

- Executes the task with the specified parameters

`AsyncTask<Params, Progress, Result> executeOnExecutor(Executor exec, Params... params)`

- Executes the task with the specified parameters on the specified Executor

`static void execute(Runnable runnable)`

- Convenience version of execute(Object) for use with a simple Runnable object

`boolean cancel (boolean mayInterruptIfRunning)`

- Attempts to cancel execution of this task

`...`

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
    - Invoked by apps

`AsyncTask<Params, Progress, Result> execute(Params... params)`

- Executes the task with the specified parameters

`AsyncTask<Params, Progress, Result> executeOnExecutor(Executor exec, Params... params)`

- Executes the task with the specified parameters on the specified Executor

`static void execute(Runnable runnable)`

- Convenience version of execute(Object) for use with a simple Runnable object

`boolean cancel (boolean mayInterruptIfRunning)`

- Attempts to cancel execution of this task

`...`

execute() runs each AsyncTask one-at-a-time (serially) in a background thread within a process

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
    - Invoked by apps

`AsyncTask<Params, Progress, Result> execute(Params... params)`

- Executes the task with the specified parameters

`AsyncTask<Params, Progress, Result> executeOnExecutor(Executor exec, Params... params)`

- Executes the task with the specified parameters on the specified Executor

`static void execute(Runnable runnable)`

- Convenience version of execute(Object) for use with a simple Runnable object

`boolean cancel (boolean mayInterruptIfRunning)`

- Attempts to cancel execution of this task

`...`

executeOnExecutor() can run multiple AsyncTasks concurrently in a pool of threads within a process

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
    - Invoked by apps

`AsyncTask<Params, Progress, Result> execute(Params... params)`

- Executes the task with the specified parameters

`AsyncTask<Params, Progress, Result> executeOnExecutor(Executor exec, Params... params)`

- Executes the task with the specified parameters on the specified Executor

`static void execute(Runnable runnable)`

- Convenience version of execute(Object) for use with a simple Runnable object

`boolean cancel (boolean mayInterruptIfRunning)`

- Attempts to cancel execution of this task

`...`

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
    - Invoked by apps

`AsyncTask<Params, Progress, Result> execute(Params... params)`

- Executes the task with the specified parameters

`AsyncTask<Params, Progress, Result> executeOnExecutor(Executor exec, Params... params)`

- Executes the task with the specified parameters on the specified Executor

`static void execute(Runnable runnable)`

- Convenience version of execute(Object) for use with a simple Runnable object

`boolean cancel (boolean mayInterruptIfRunning)`

- Attempts to cancel execution of this task

`...`

cancel() requires cooperation by the AsyncTask, i.e., it's voluntary

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
  - Protected hook methods

`void onPreExecute()`
- Runs on UI thread before doInBackground()

`abstract Result doInBackground (Params... params)`
- Override this method to perform a computation on a background thread

`void onPostExecute(Result result)`
- Runs on UI thread after doInBackground()

`void onProgressUpdate(Progress... values)`
- Runs on UI thread after publishProgress() called

`void onCancelled()`
- Runs on UI thread after cancel() is invoked & doInBackground() has finished

`...`

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
  - Protected hook methods
    - Overridden by apps

`void onPreExecute()`
- Runs on UI thread before doInBackground()

`abstract Result doInBackground (Params... params)`
- Override this method to perform a computation on a background thread

`void onPostExecute(Result result)`
- Runs on UI thread after doInBackground()

`void onProgressUpdate(Progress... values)`
- Runs on UI thread after publishProgress() called

`void onCancelled()`
- Runs on UI thread after cancel() is invoked & doInBackground() has finished

`...`

Categories of Methods in the AsyncTask Class

- The AsyncTask class has two types of methods
  - Public methods
  - Protected hook methods
    - Overridden by apps
    - Invoked by framework
      - At different points of time &
      - In different threading contexts

`void onPreExecute()`

- Runs on UI thread before doInBackground()

`abstract Result doInBackground (Params... params)`

- Override this method to perform a computation on a background thread

`void onPostExecute(Result result)`

- Runs on UI thread after doInBackground()

`void onProgressUpdate(Progress... values)`

- Runs on UI thread after publishProgress() called

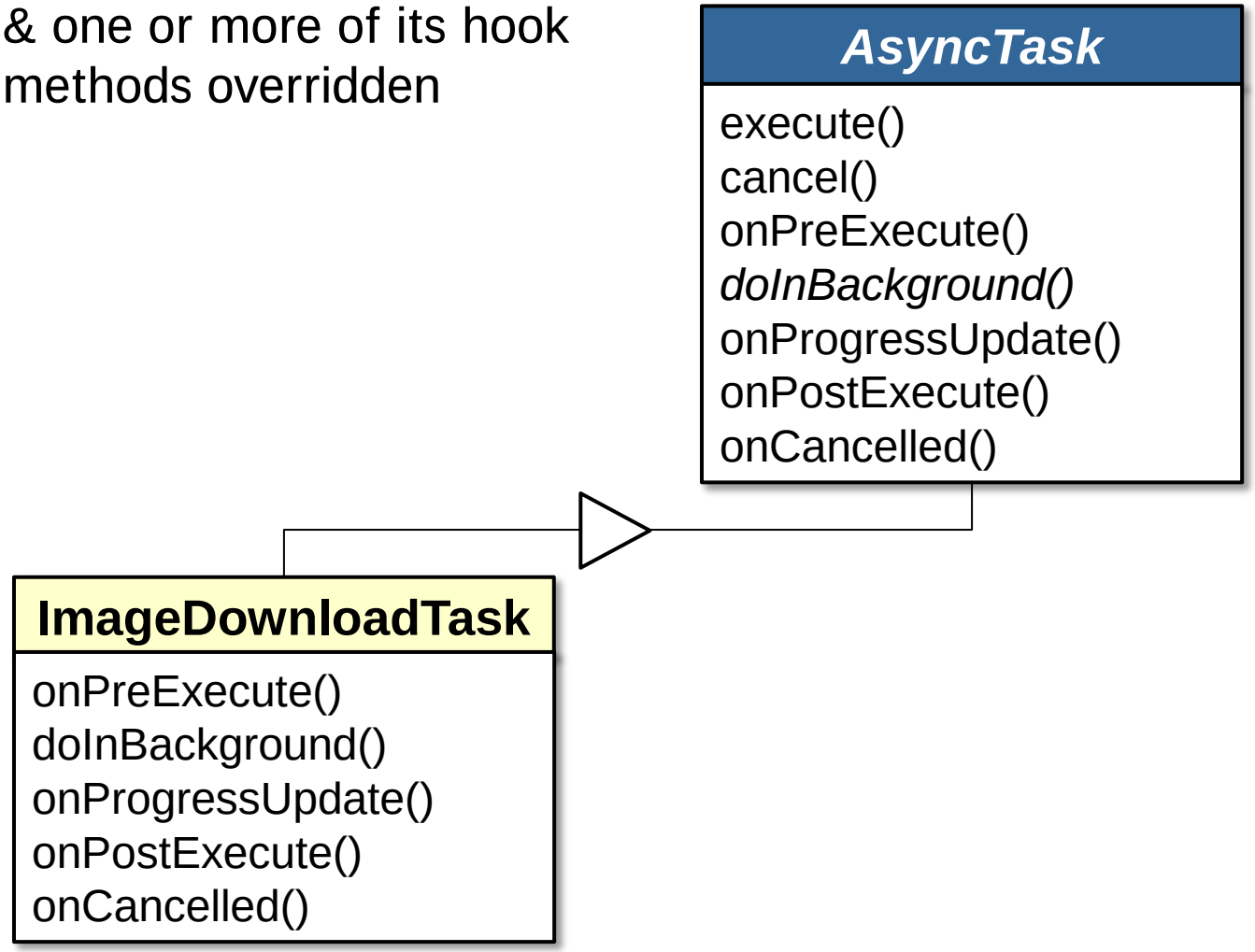
`void onCancelled(Result result)`

- Runs on UI thread after cancel() is invoked & doInBackground() has finished

...

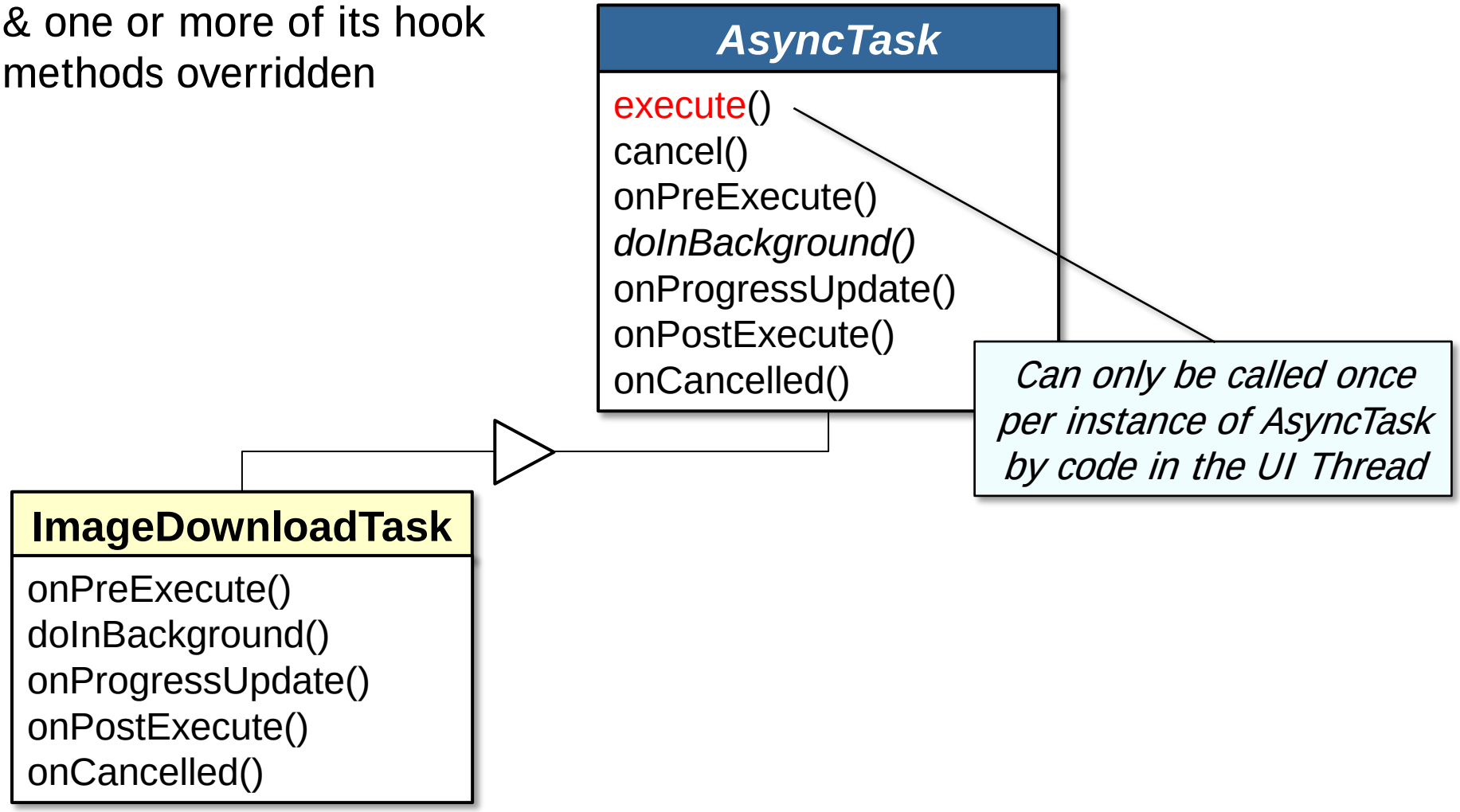
Categories of Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden



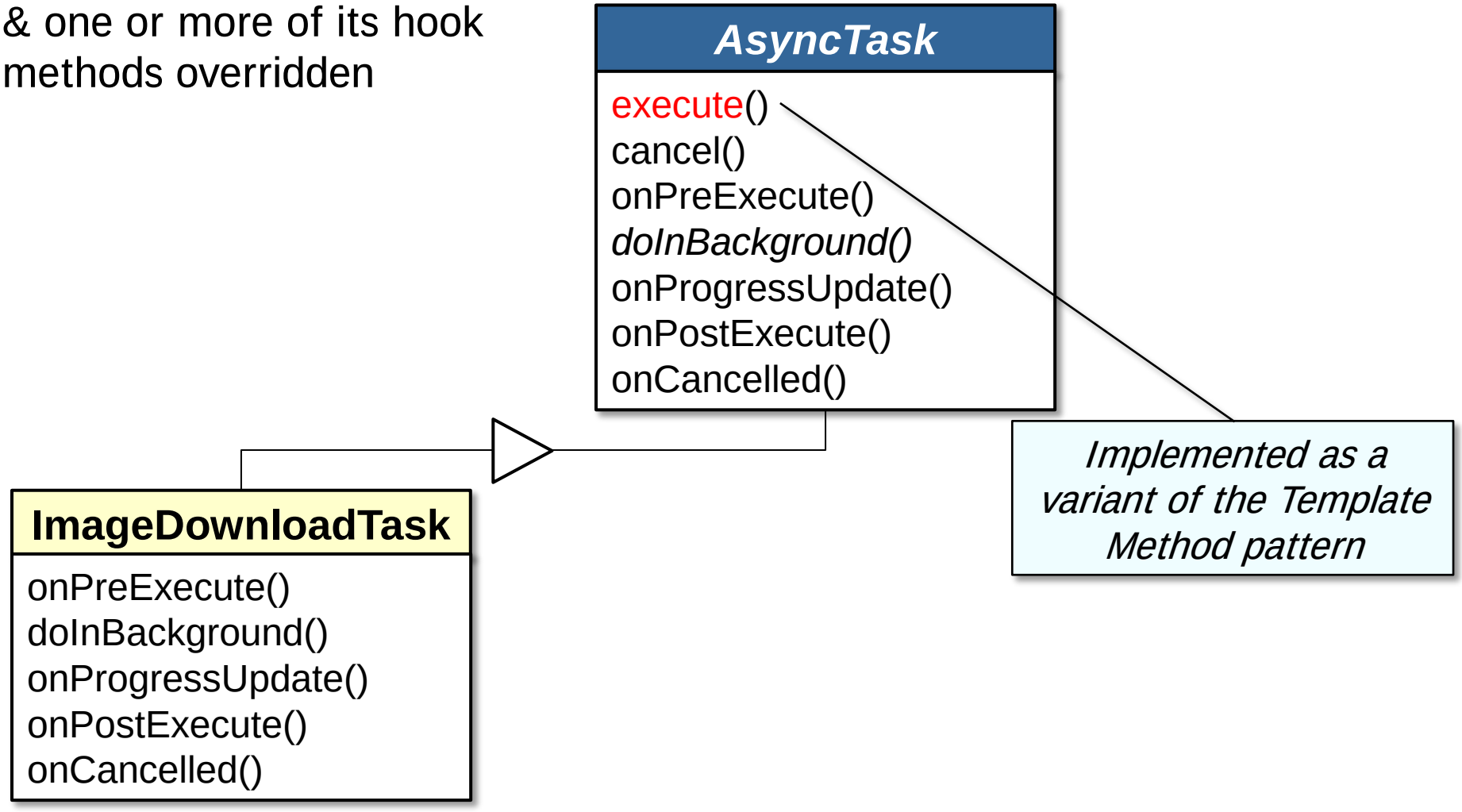
Categories of Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden



Categories of Methods in the AsyncTask Class

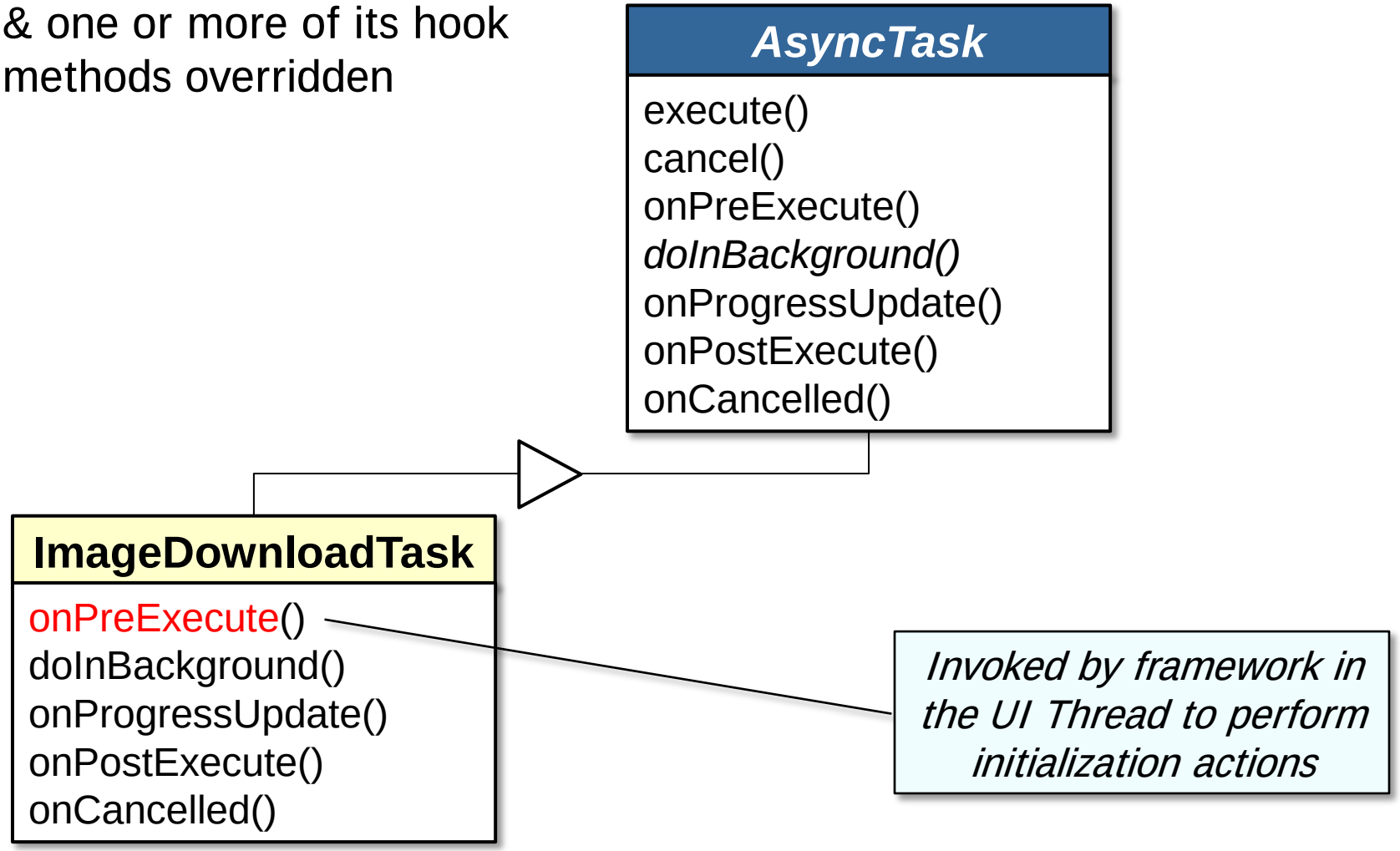
- AsyncTask must be extended & one or more of its hook methods overridden



See en.wikipedia.org/wiki/Template_method_pattern

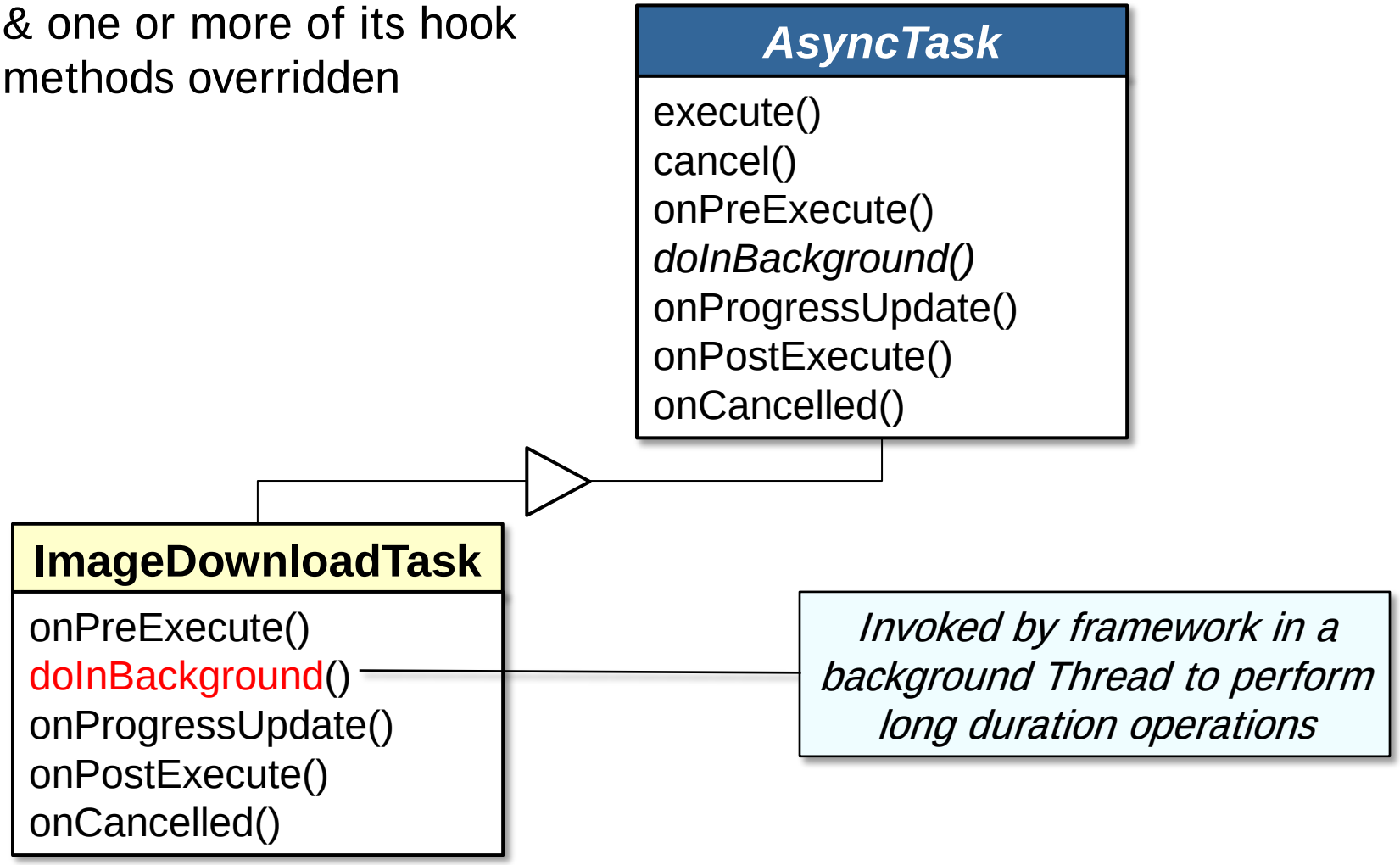
Categories of Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden



Categories of Methods in the AsyncTask Class

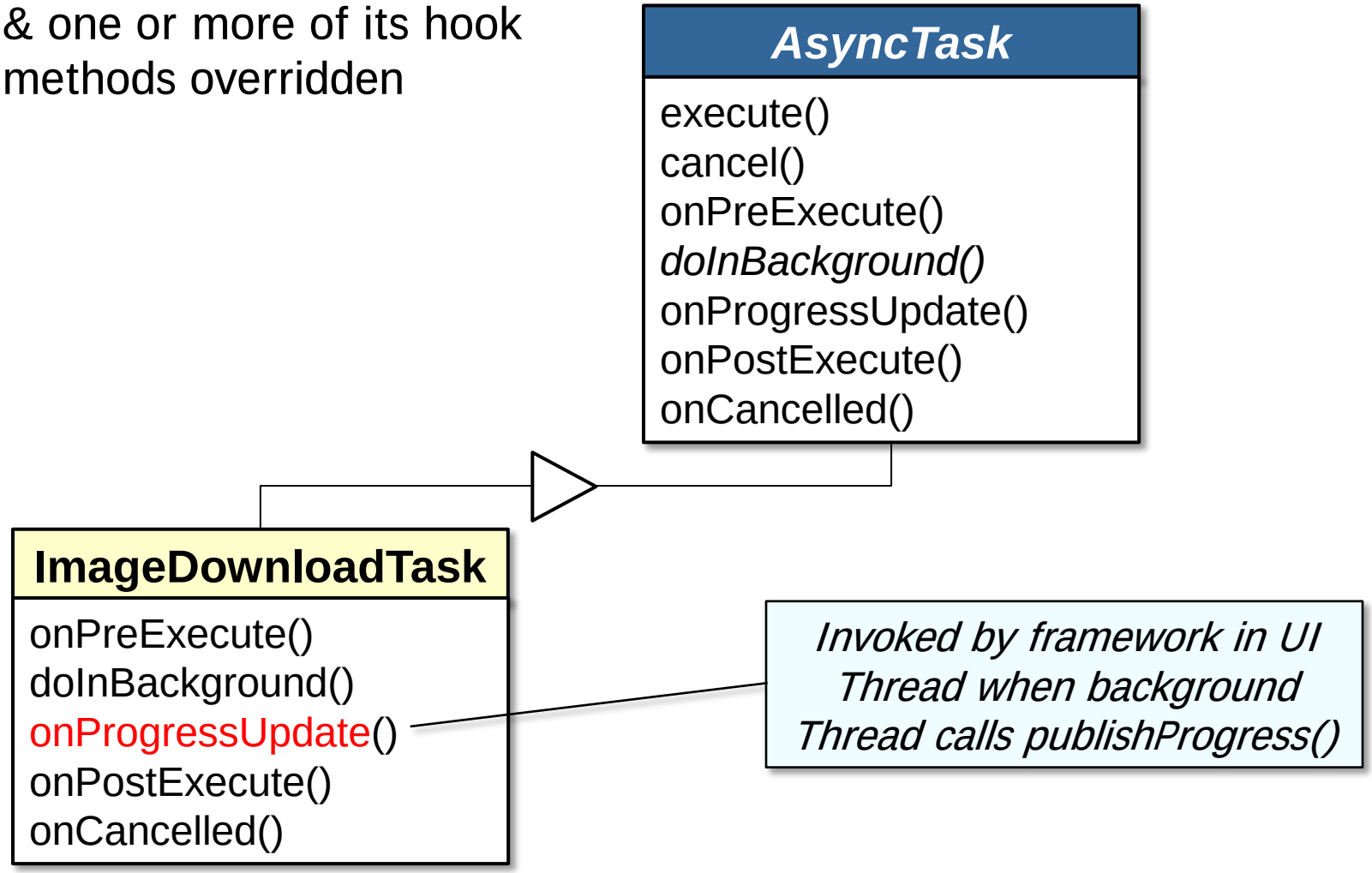
- AsyncTask must be extended & one or more of its hook methods overridden



See www.androiddesignpatterns.com/2014/01/thread-scheduling-in-android.html

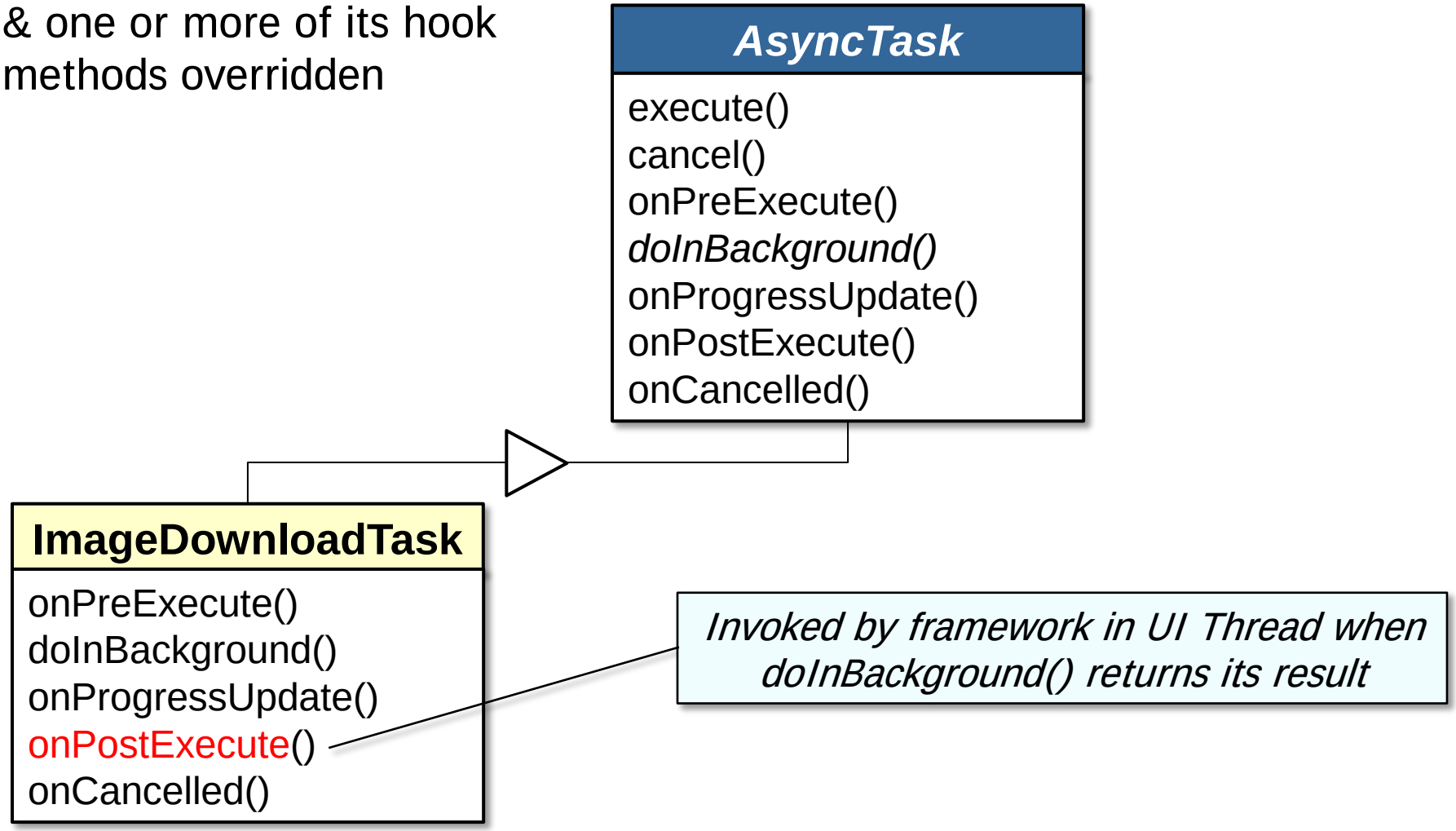
Categories of Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden



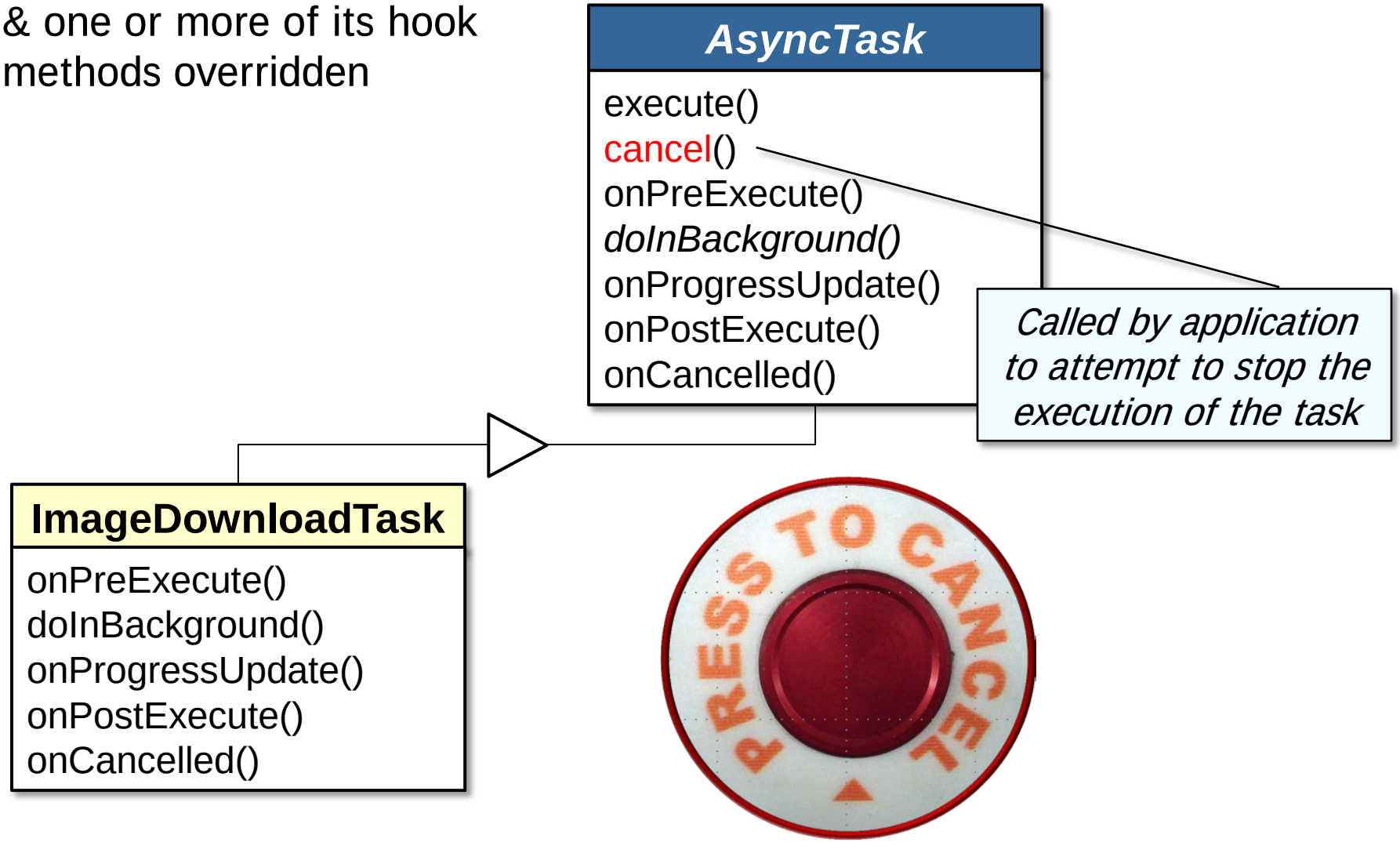
Categories of Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden



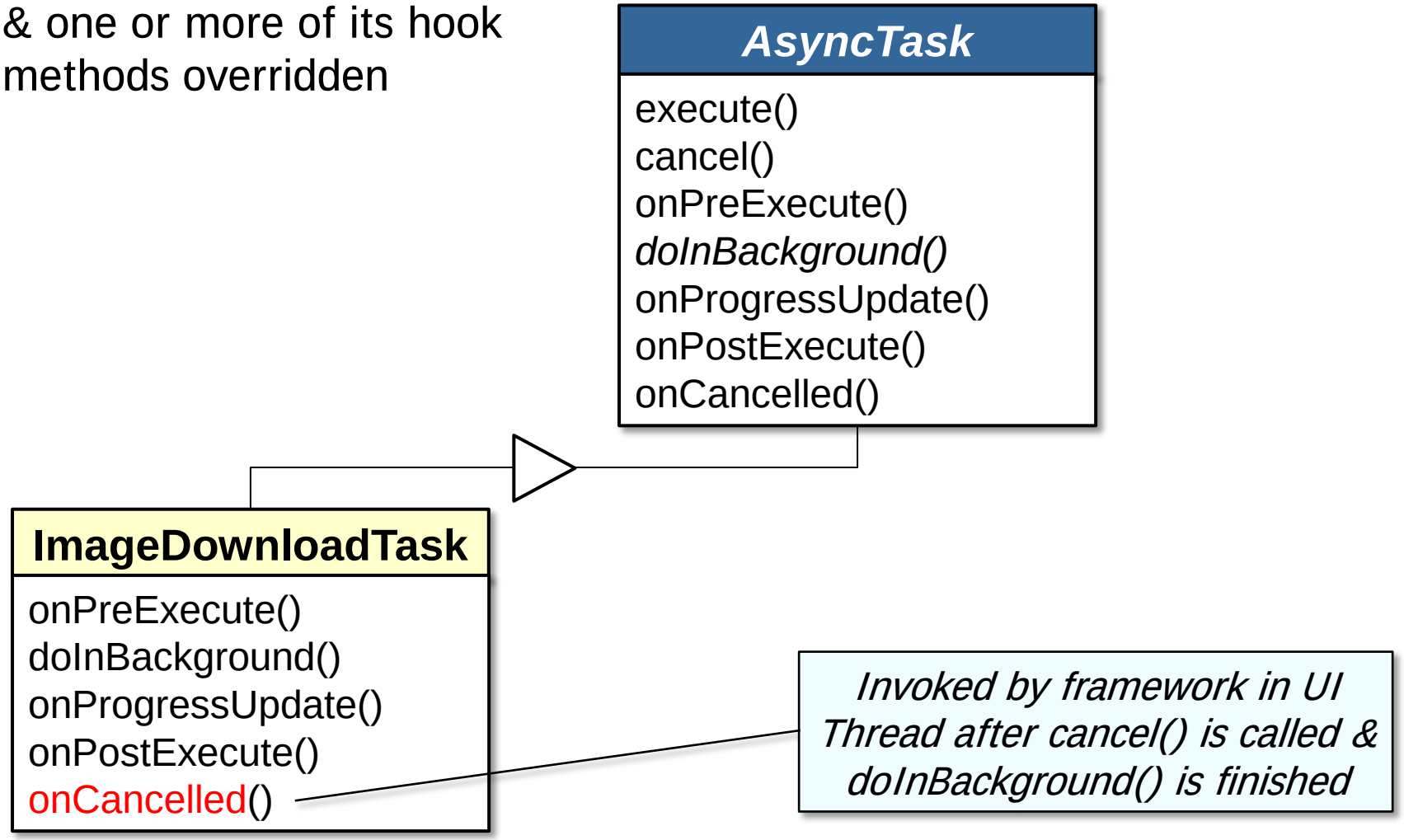
Categories of Methods in the AsyncTask Class

- AsyncTask must be extended & one or more of its hook methods overridden



Categories of Methods in the AsyncTask Class

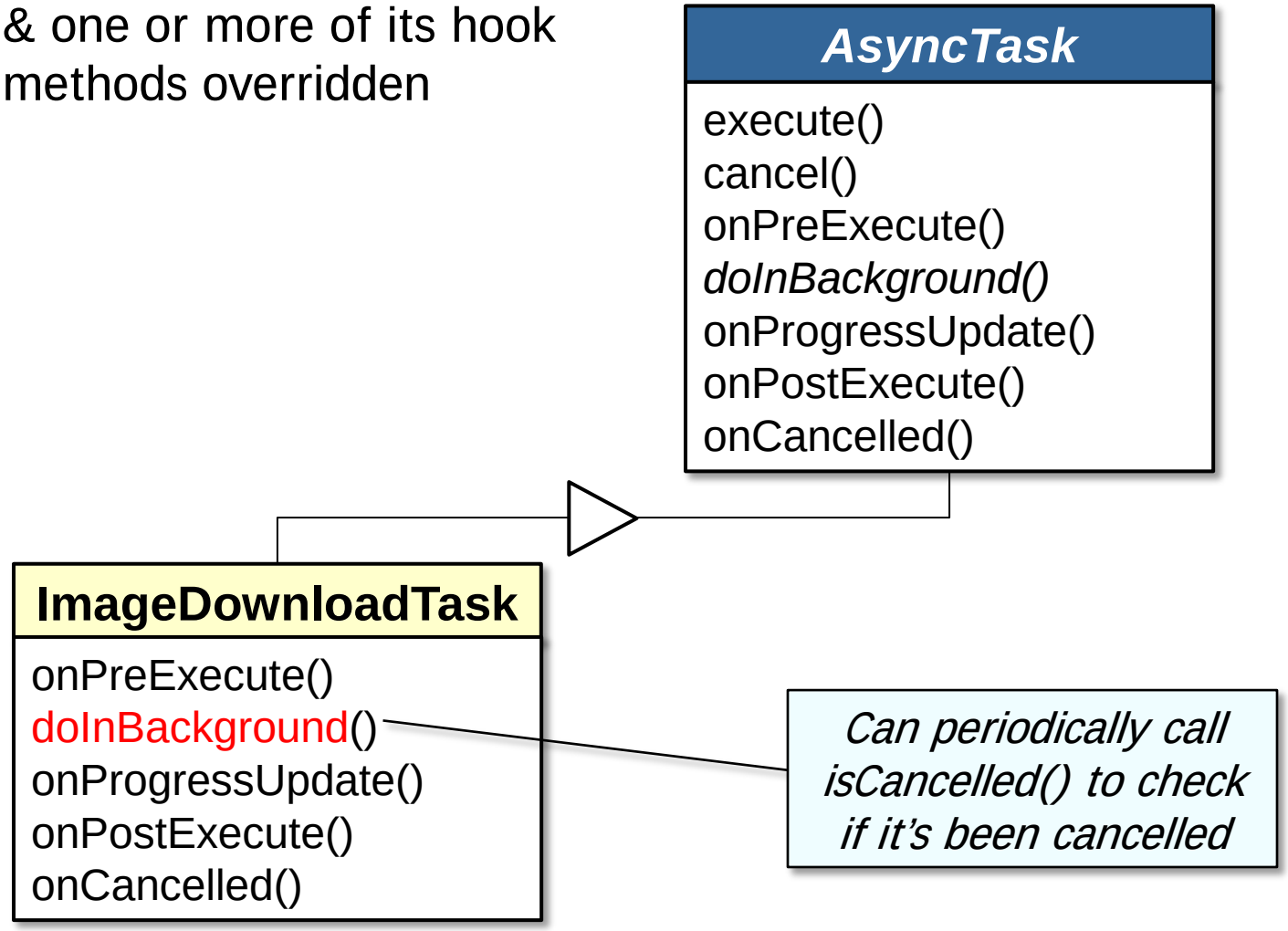
- AsyncTask must be extended & one or more of its hook methods overridden



If onCancelled() is called then onPostExecute() is *not* called

Categories of Methods in the AsyncTask Class

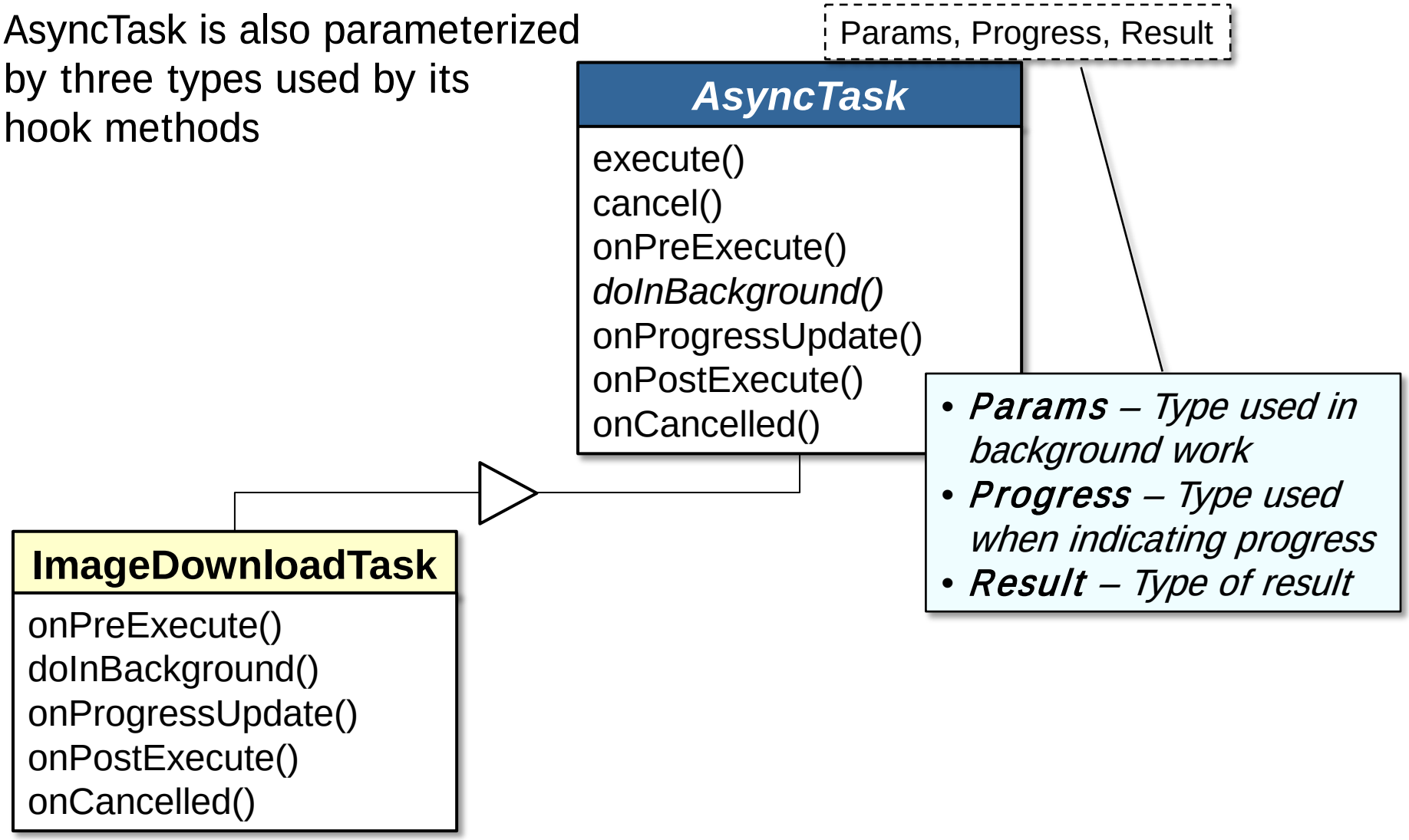
- AsyncTask must be extended & one or more of its hook methods overridden



Similar to using Java Thread interrupt requests to voluntarily shutdown Threads

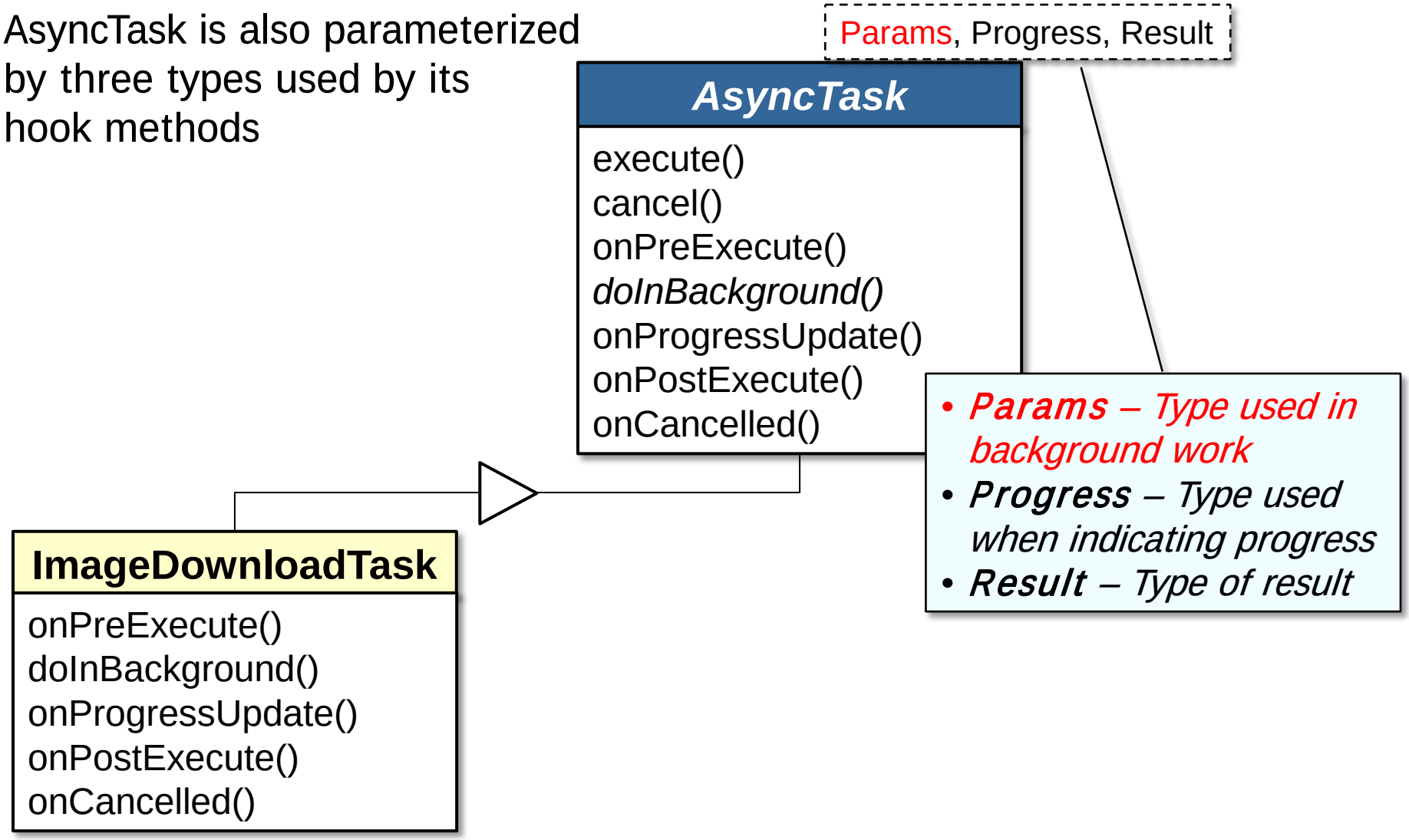
Categories of Methods in the AsyncTask Class

- AsyncTask is also parameterized by three types used by its hook methods



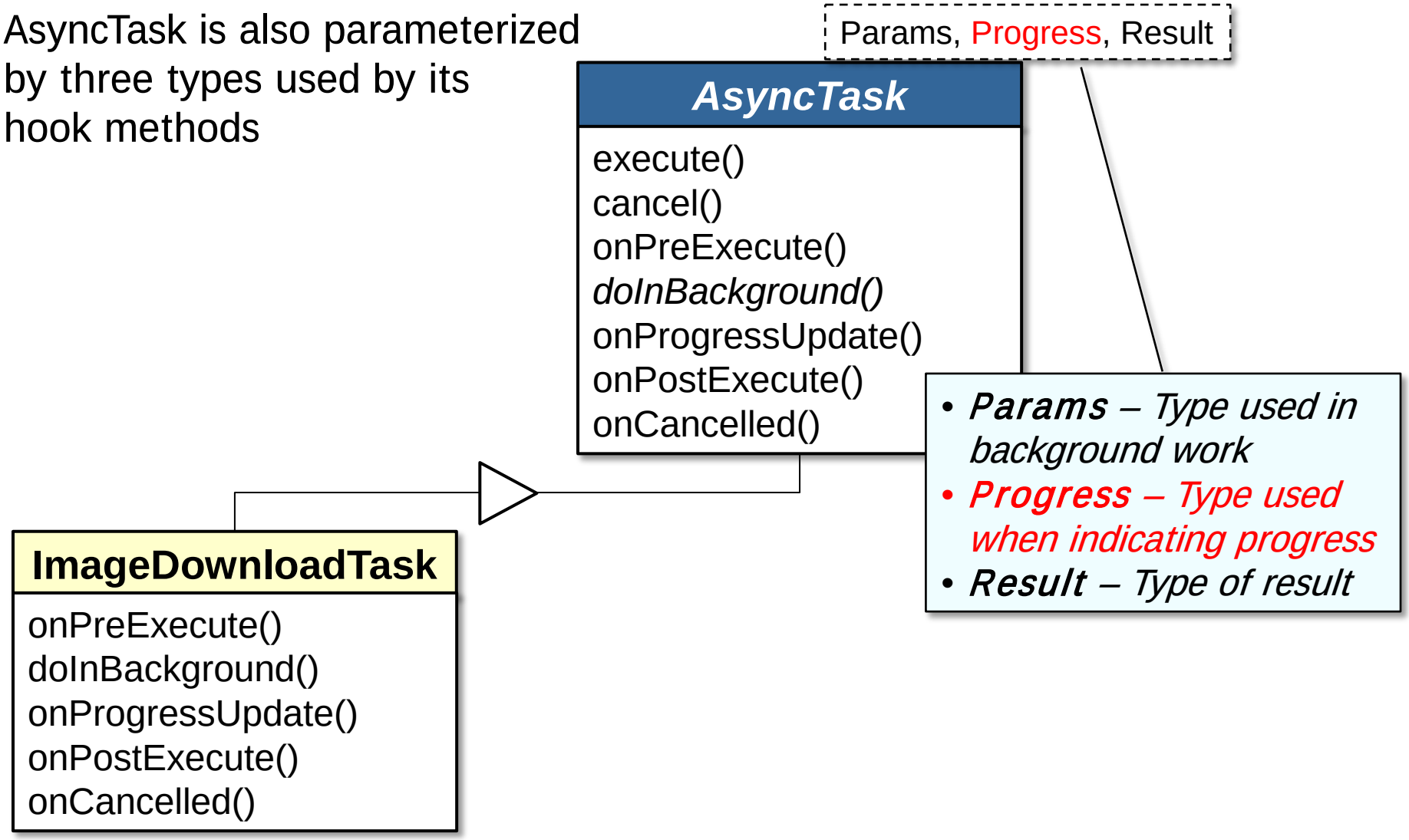
Categories of Methods in the AsyncTask Class

- AsyncTask is also parameterized by three types used by its hook methods



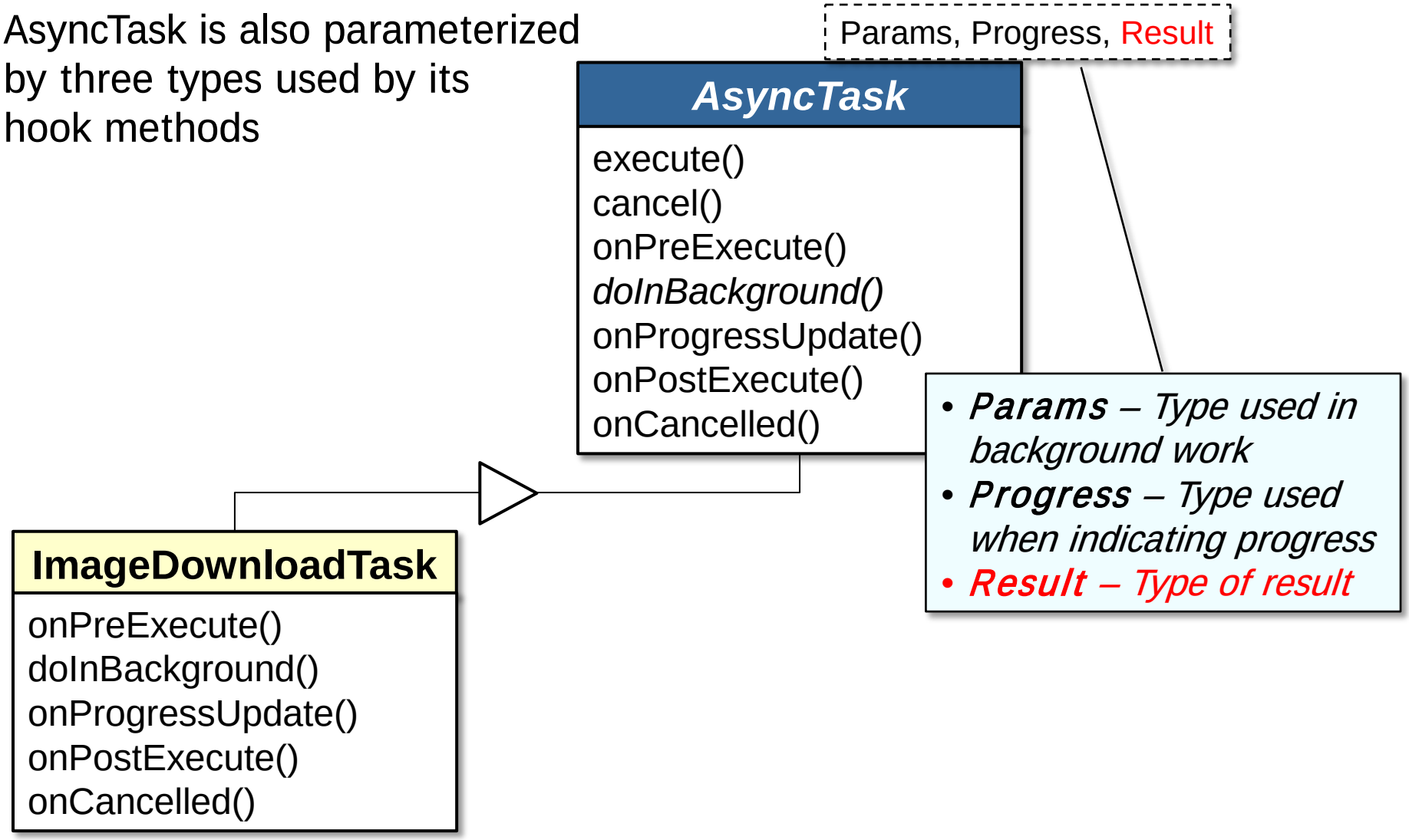
Categories of Methods in the AsyncTask Class

- AsyncTask is also parameterized by three types used by its hook methods



Categories of Methods in the AsyncTask Class

- AsyncTask is also parameterized by three types used by its hook methods



Categories of Methods in the AsyncTask Class

- Apps must customize the AsyncTask class to meet their concurrency needs

```
class DownloadTask extends
      AsyncTask<Uri, Integer, Long> {
  protected Long doInBackground
                          (Uri... urls)
  { /* Download files */ }

  protected void onProgressUpdate
                (Integer... progress)
  { setProgressPercent(progress[0]); }

  protected void onPostExecute
                           (Long result)
  { showDialog("Downloaded "
                + result
                + " bytes"); }
}

new DownloadTask().execute(downloadURL);
```

Categories of Methods in the AsyncTask Class

- Apps must customize the AsyncTask class to meet their concurrency needs

```
class DownloadTask extends
      AsyncTask<Uri, Integer, Long> {
  protected Long doInBackground
                          (Uri... urls)
  { /* Download files */ }

  protected void onProgressUpdate
                (Integer... progress)
  { setProgressPercent(progress[0]); }

  protected void onPostExecute
                           (Long result)
  { showDialog("Downloaded "
                + result
                + " bytes"); }
}

new DownloadTask().execute(downloadURL);
```

Categories of Methods in the AsyncTask Class

- Apps must customize the AsyncTask class to meet their concurrency needs

```
class DownloadTask extends
      AsyncTask<Uri, Integer, Long> {
  protected Long doInBackground
                          (Uri... urls)
  { /* Download files */ }

  protected void onProgressUpdate
                (Integer... progress)
  { setProgressPercent(progress[0]); }

  protected void onPostExecute
                           (Long result)
  { showDialog("Downloaded "
                + result
                + " bytes"); }
}

new DownloadTask().execute(downloadURL);
```

Categories of Methods in the AsyncTask Class

- Apps must customize the AsyncTask class to meet their concurrency needs

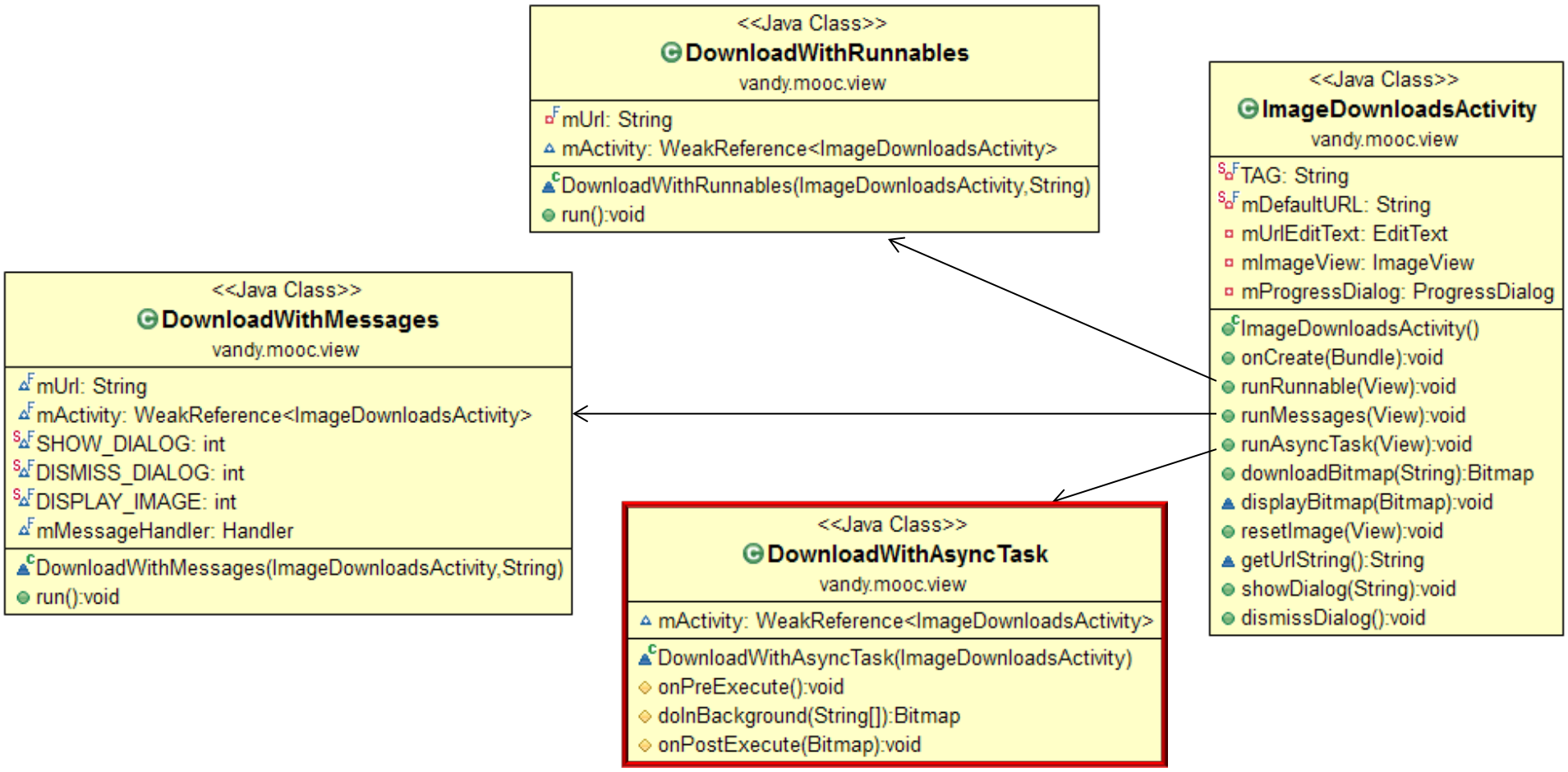
```
class DownloadTask extends
      AsyncTask<Uri, Integer, Long> {
  protected Long doInBackground
                          (Uri... urls)
  { /* Download files */ }

  protected void onProgressUpdate
                (Integer... progress)
  { setProgressPercent(progress[0]); }

  protected void onPostExecute
                           (Long result)
  { showDialog("Downloaded "
                + result
                + " bytes"); }
}

new DownloadTask().execute(downloadURL);
```

See developer.android.com/reference/android/os/AsyncTask.html

Programming with AsyncTask

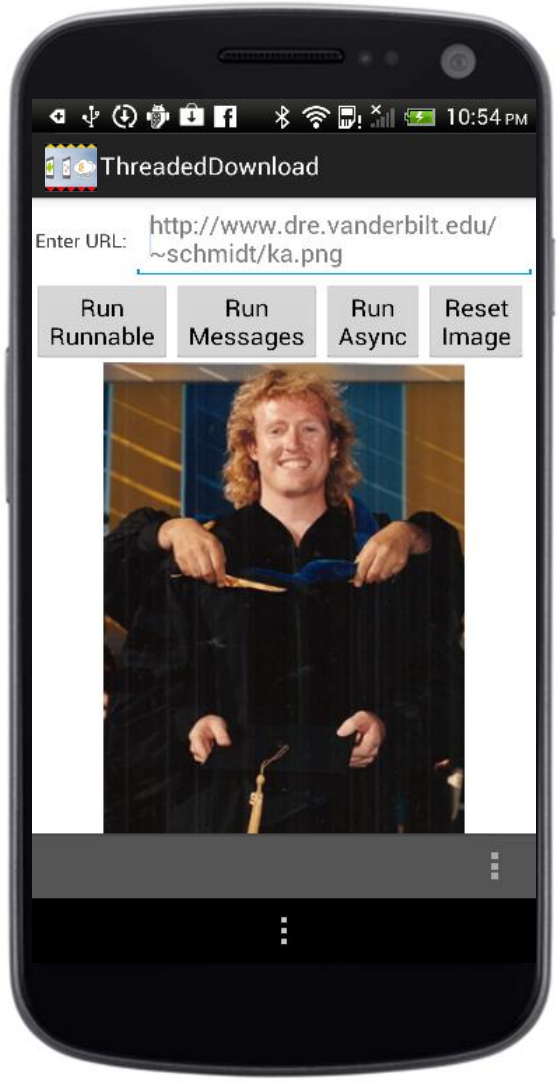
“Run Async” Behavior for SimpleImageDownloads

- Downloads & Displays image via AsyncTask

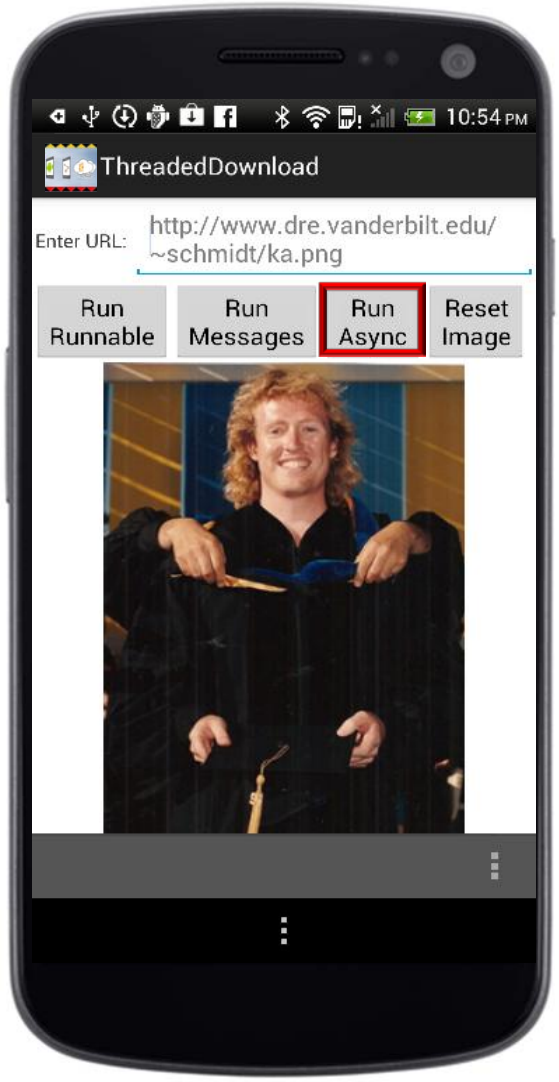


See github.com/douglascraigschmidt/POSA-15/tree/master/ex/SimpleImageDownloads

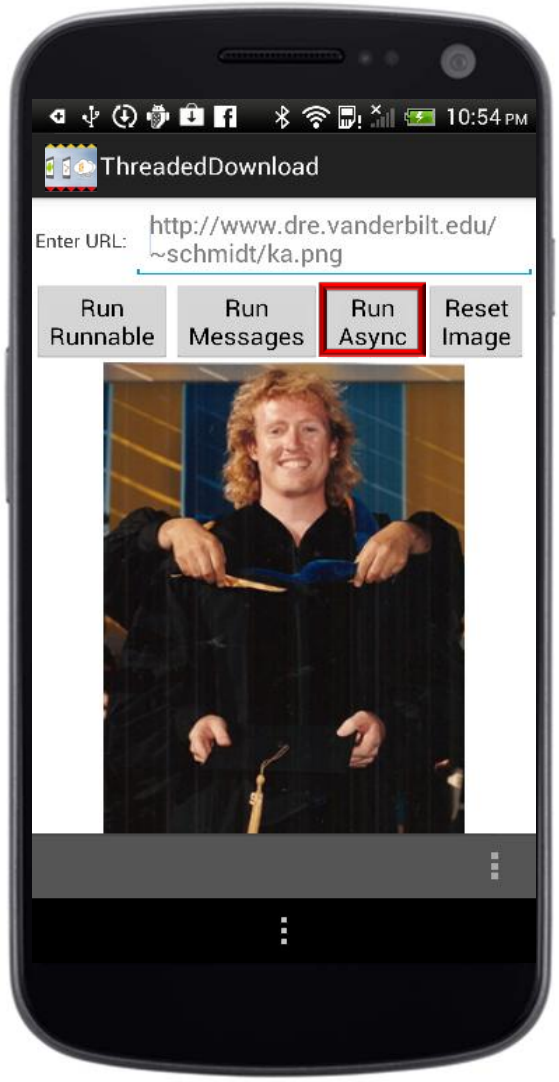
“Run Async” Behavior for SimpleImageDownloads



“Run Async” Behavior for SimpleImageDownloads

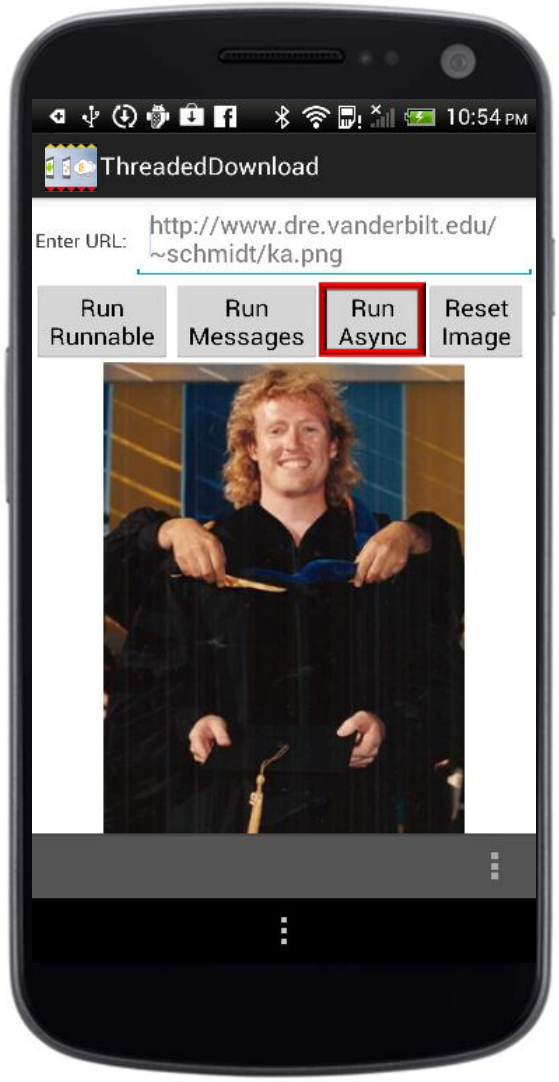


“Run Async” Behavior for SimpleImageDownloads




```
<Button
   ...
   android:onClick="runAsyncTask"
   android:text="@string/runAsync" />
```

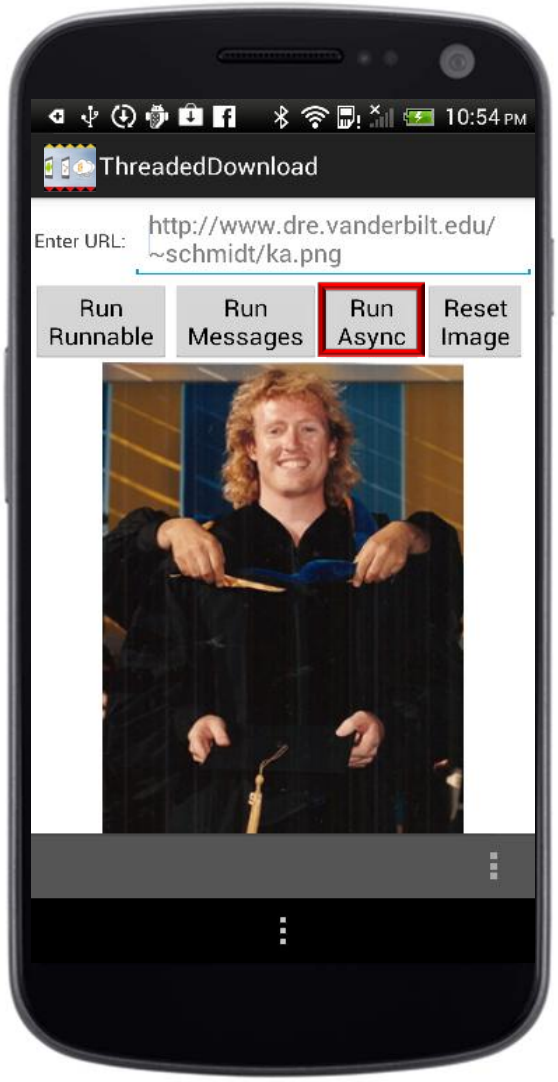
“Run Async” Behavior for SimpleImageDownloads




```
<Button
   ...
   android:onClick="runAsyncTask"
   android:text="@string/runAsync" />


public class ImageDownloadsActivity
       extends Activity {
   ...
   public void runAsyncTask(View view) {
     /* all magic happens here! */
   }
```

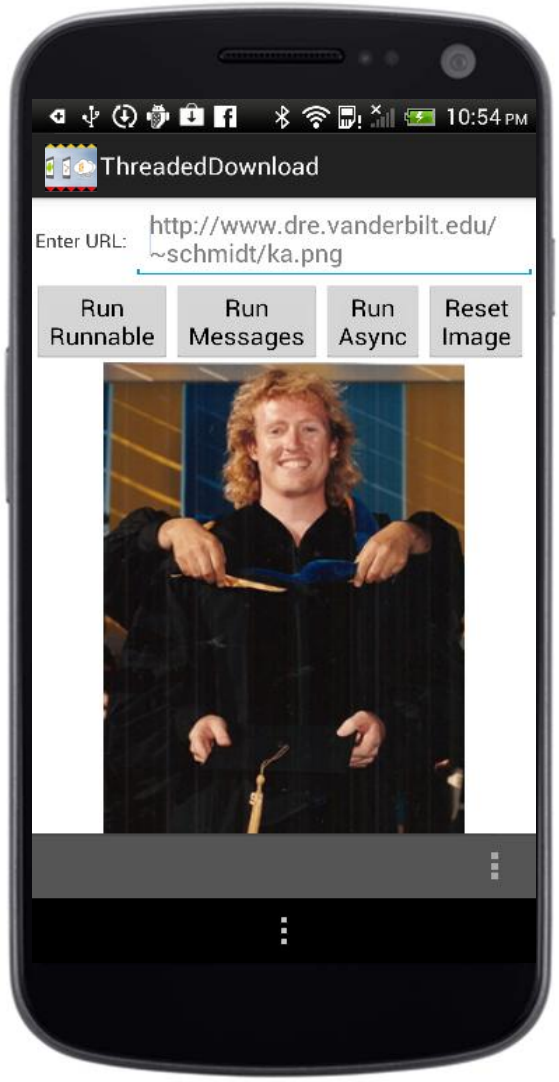
See ThreadedDownloads/src/edu/vuum/mocca/ImageDownloadsActivity.java

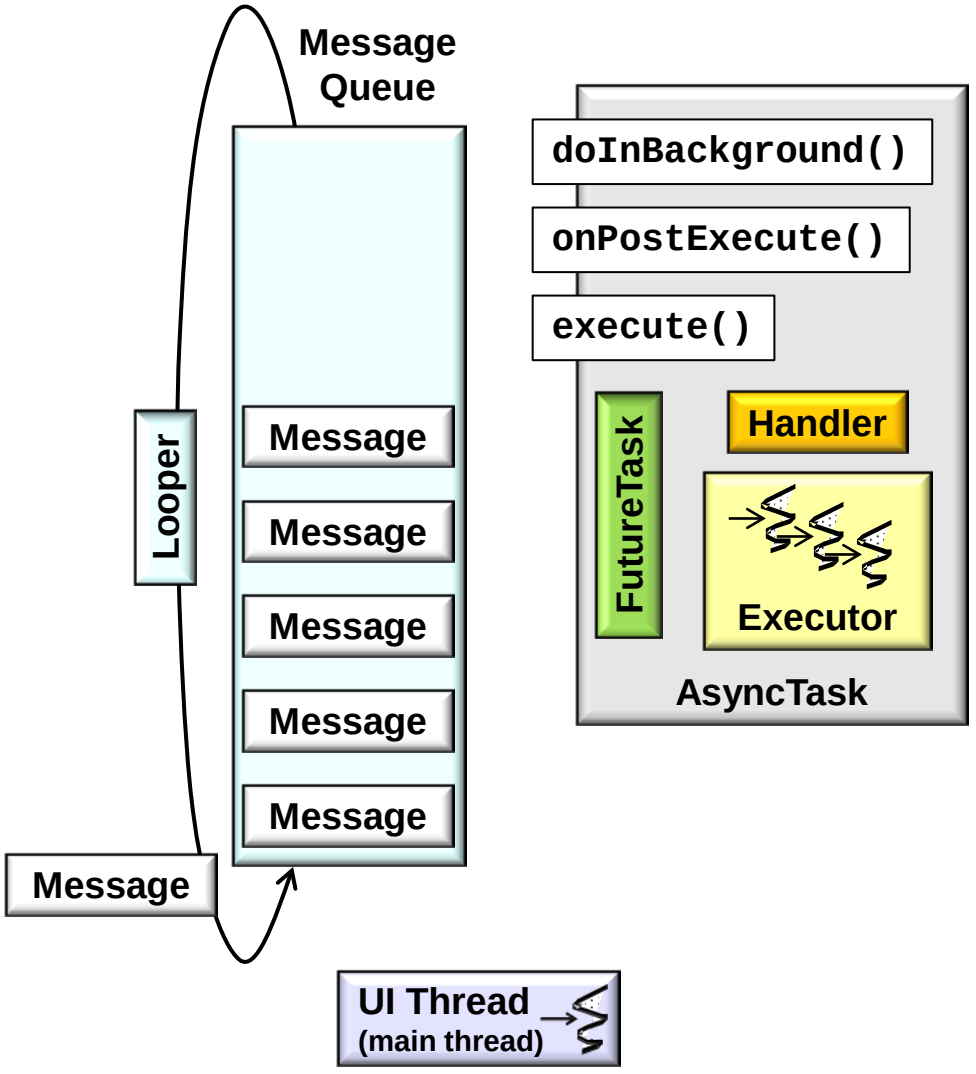
“Run Async” Behavior for SimpleImageDownloads



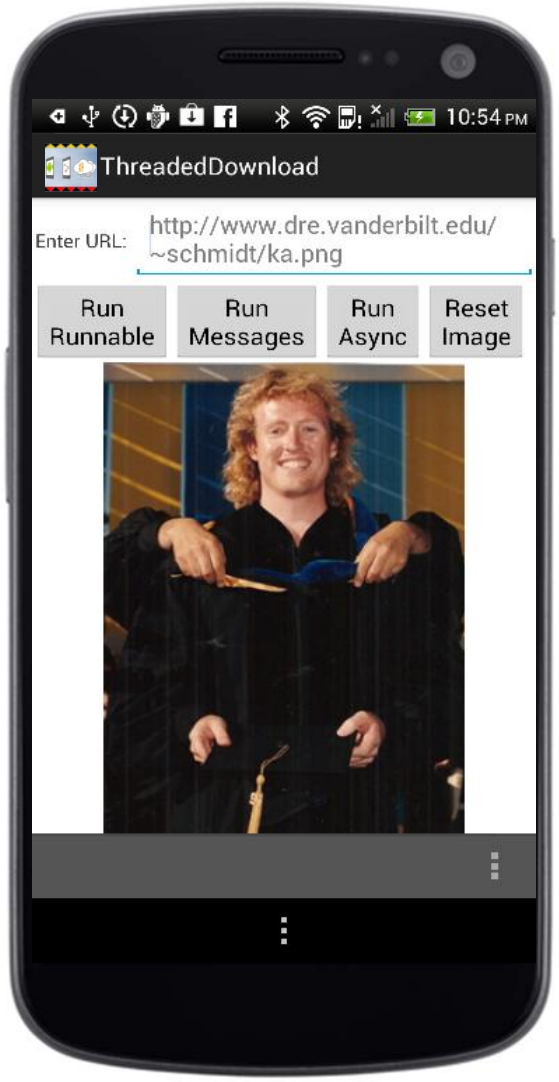

```
<Button
   ...
   android:onClick="runAsyncTask"
   android:text="@string/runAsync" />


public class ImageDownloadsActivity
       extends Activity {
   ...
   public void runAsyncTask(View view) {
     /* all magic happens here! */
   }
```

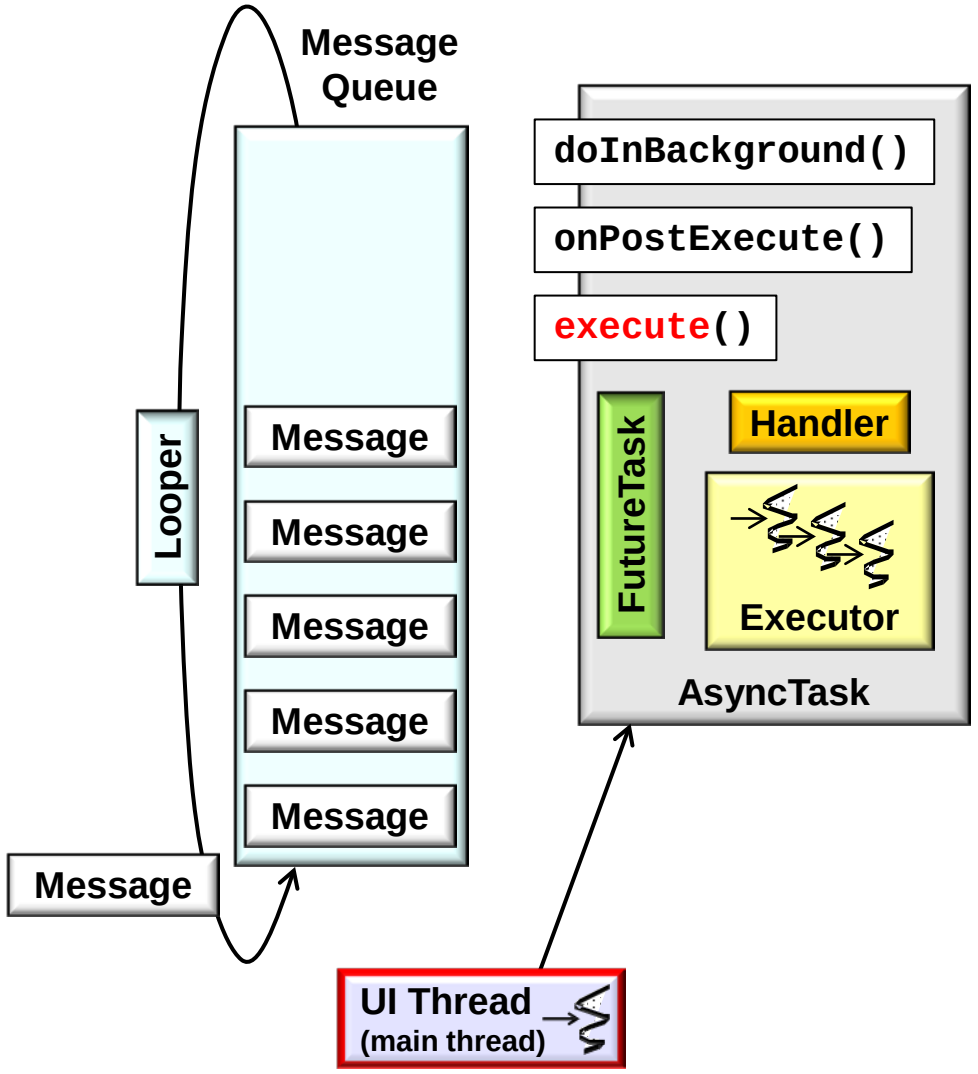
“Run Async” Behavior for SimpleImageDownloads



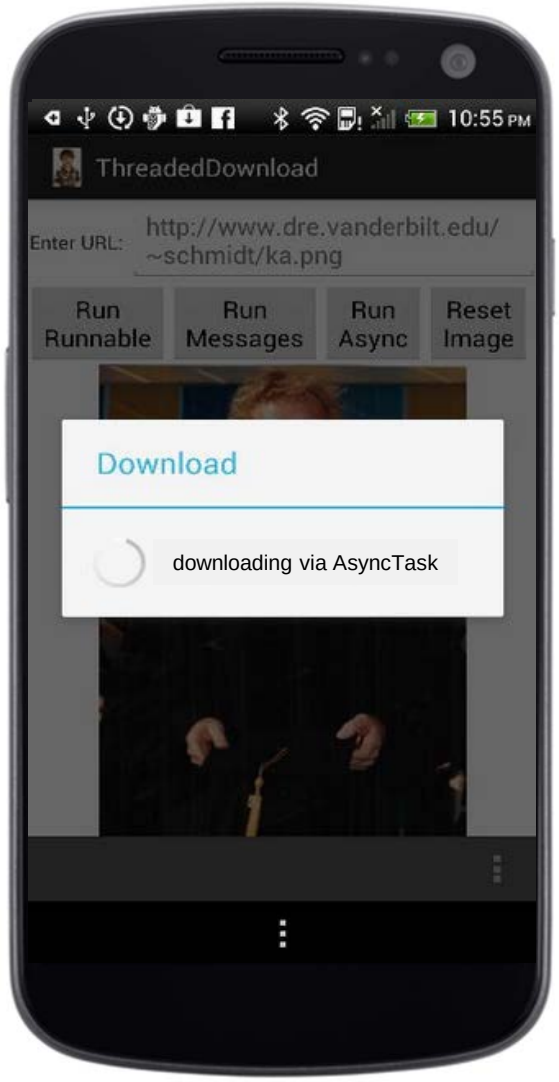


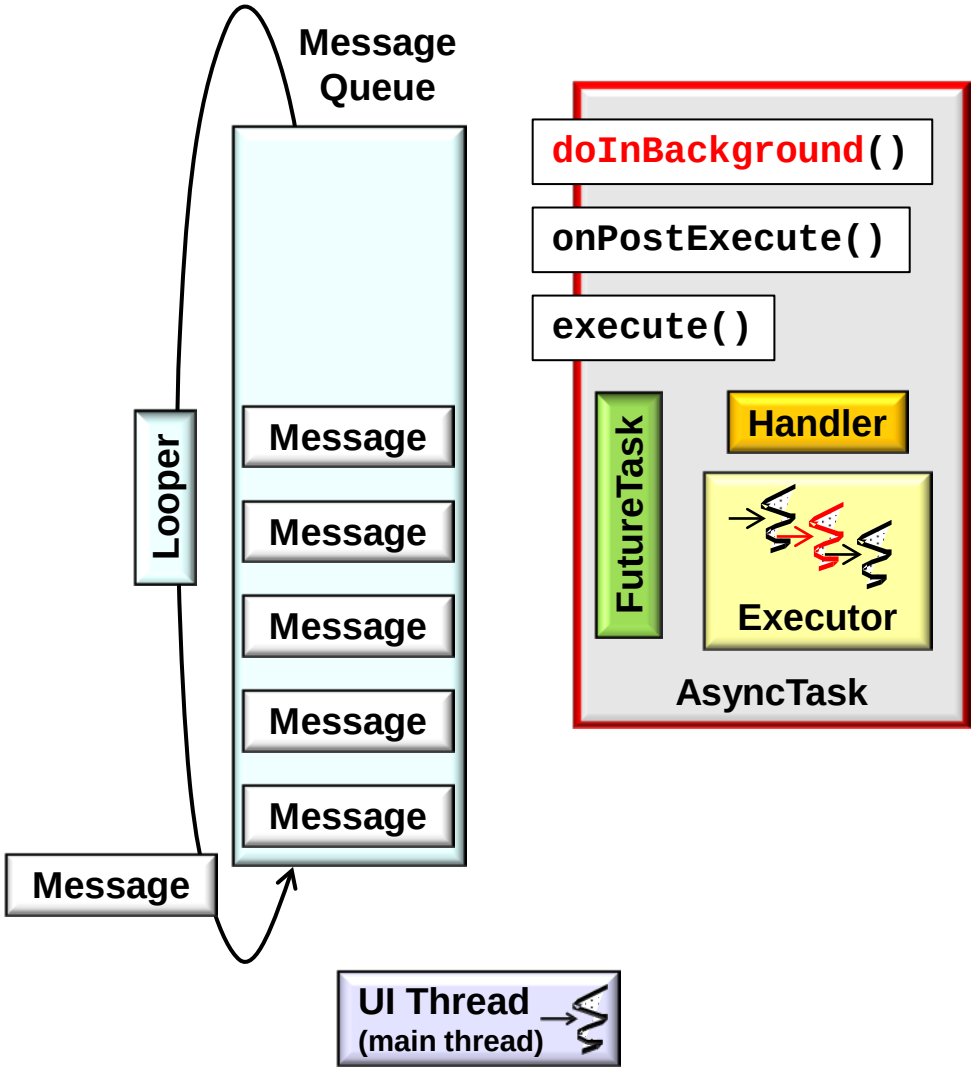
“Run Async” Behavior for SimpleImageDownloads



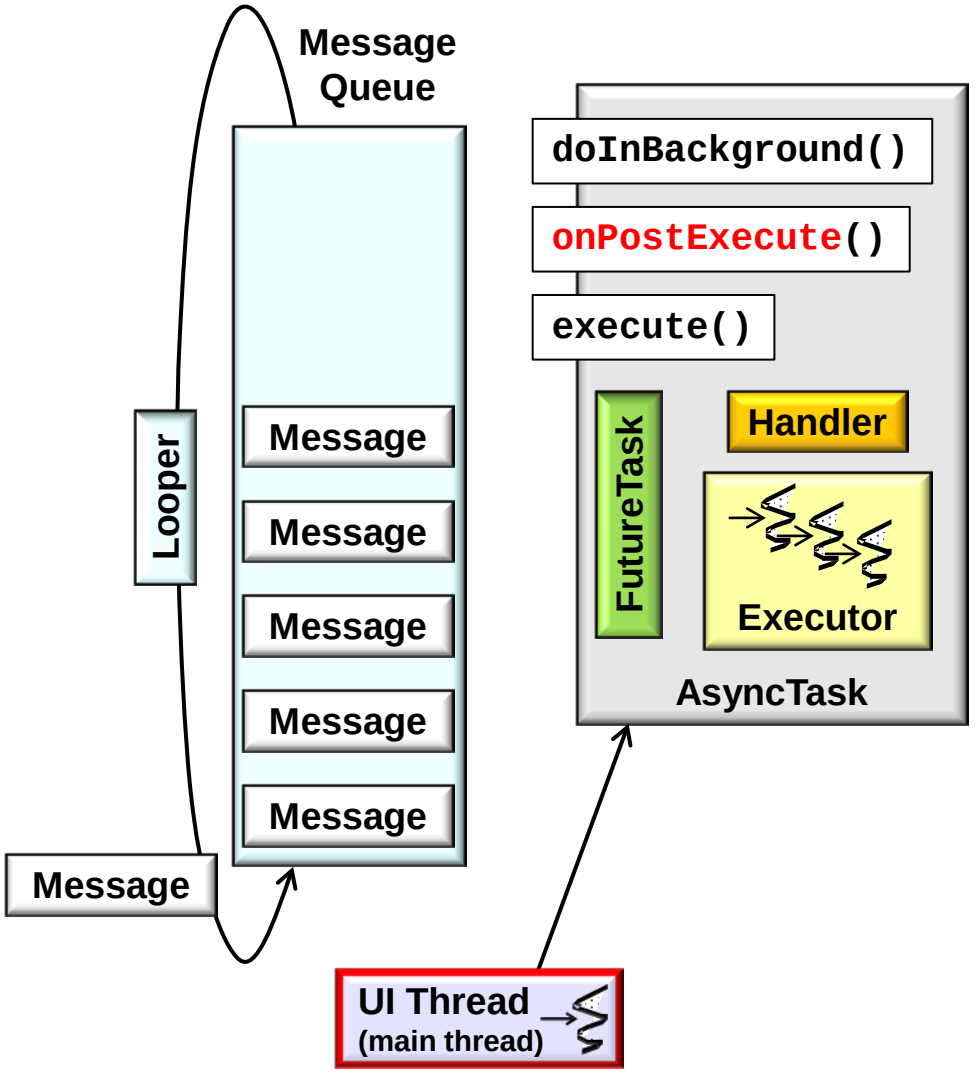


“Run Async” Behavior for SimpleImageDownloads

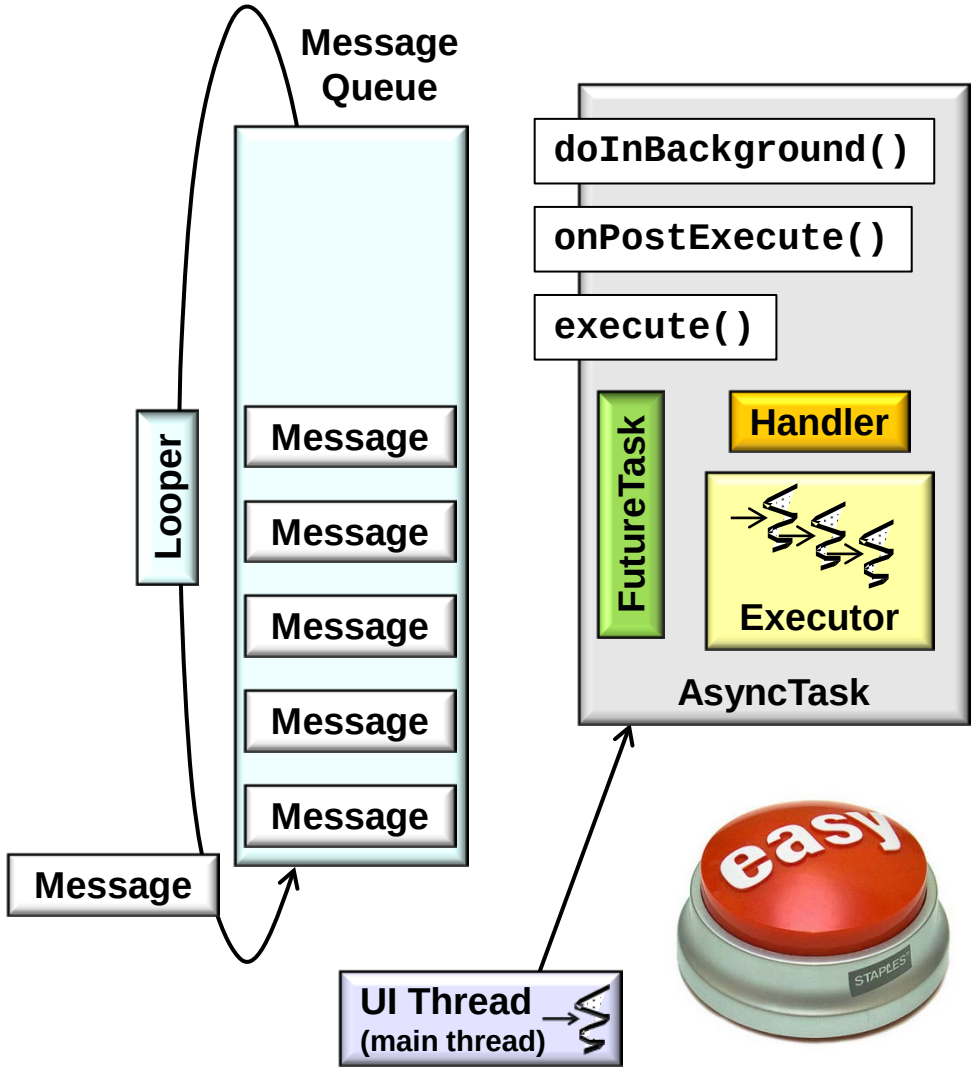




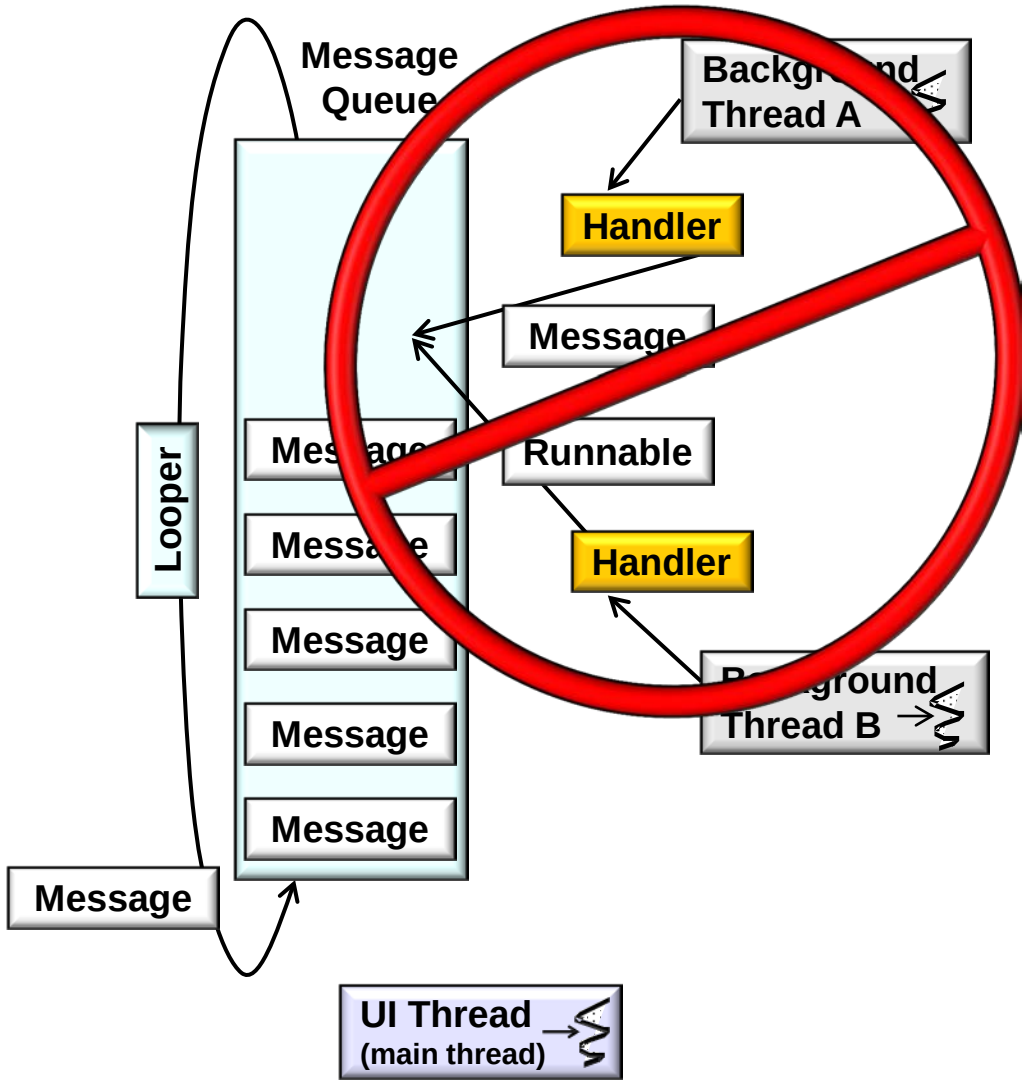
“Run Async” Behavior for SimpleImageDownloads



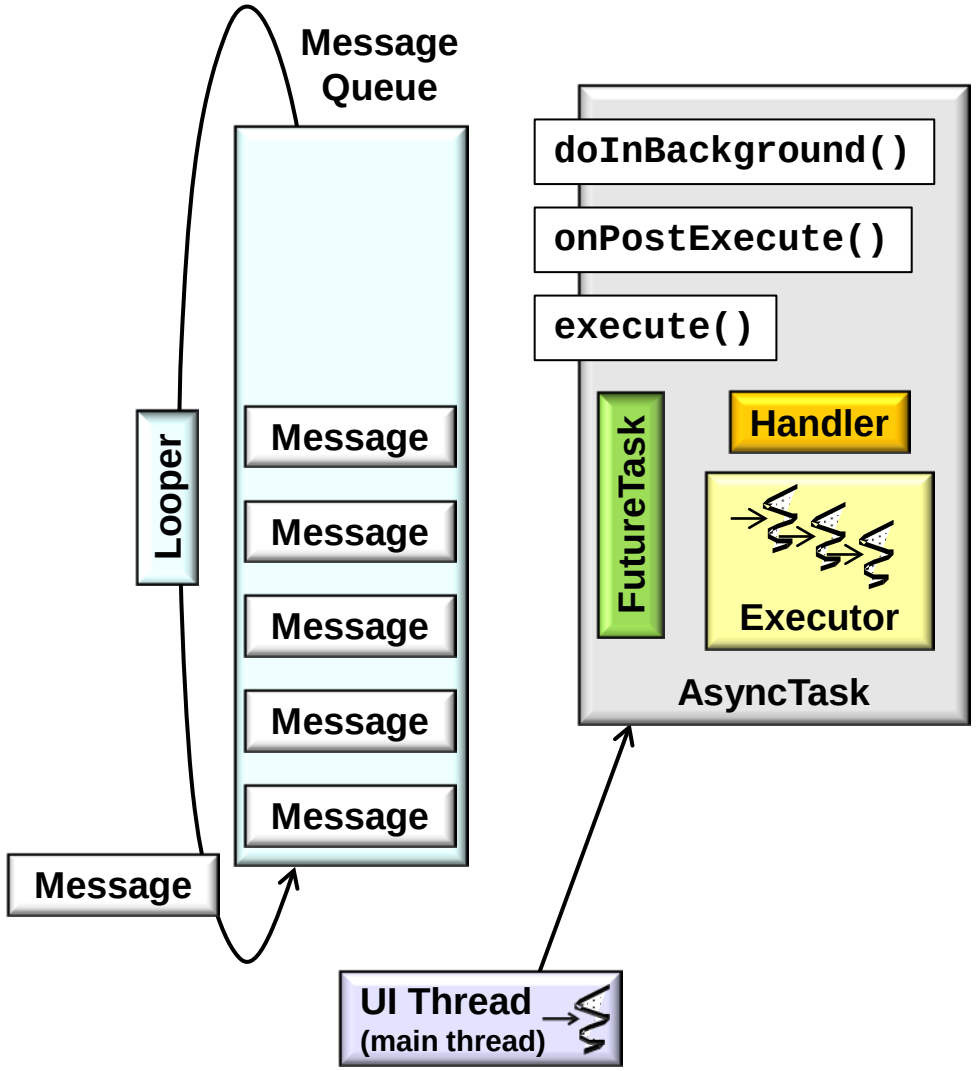
“Run Async” Behavior for SimpleImageDownloads



“Run Async” Behavior for SimpleImageDownloads

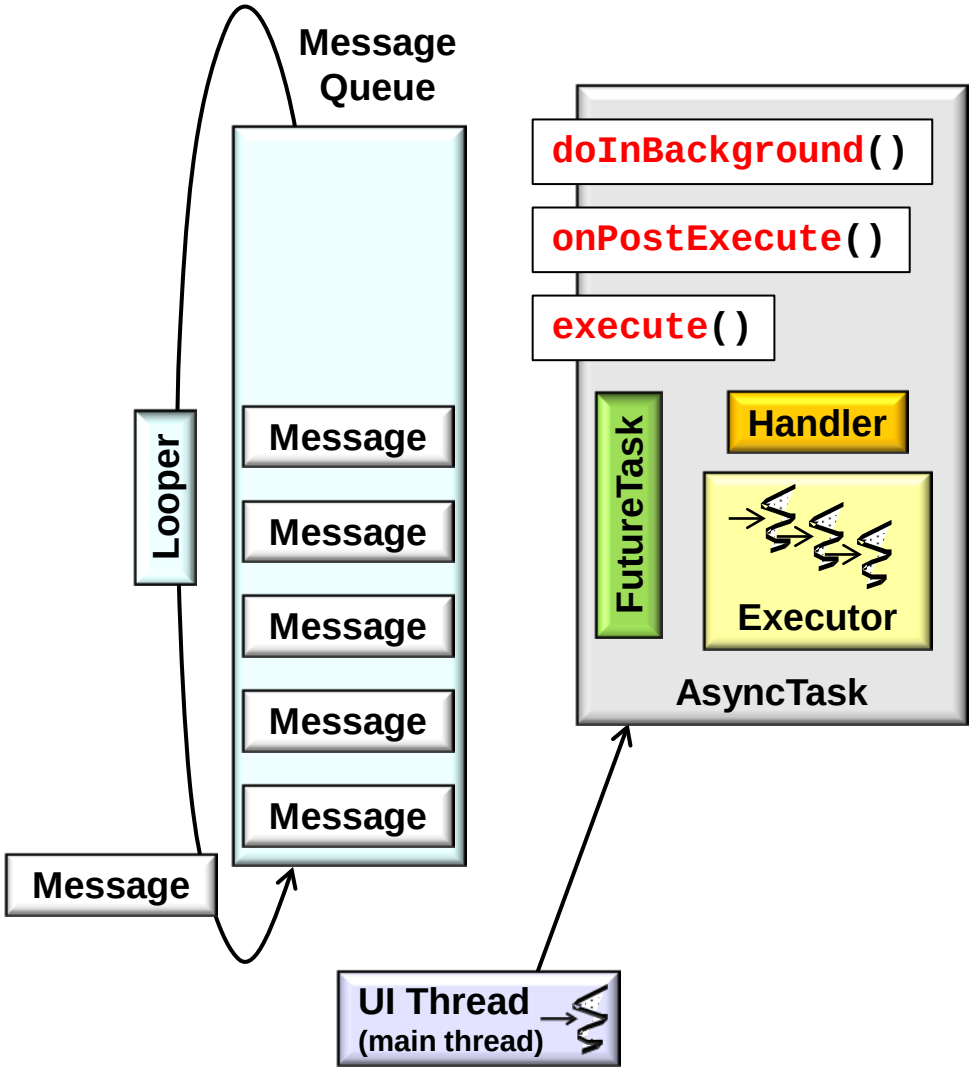


“Run Async” Behavior for SimpleImageDownloads



See en.wikipedia.org/wiki/Template_method_pattern

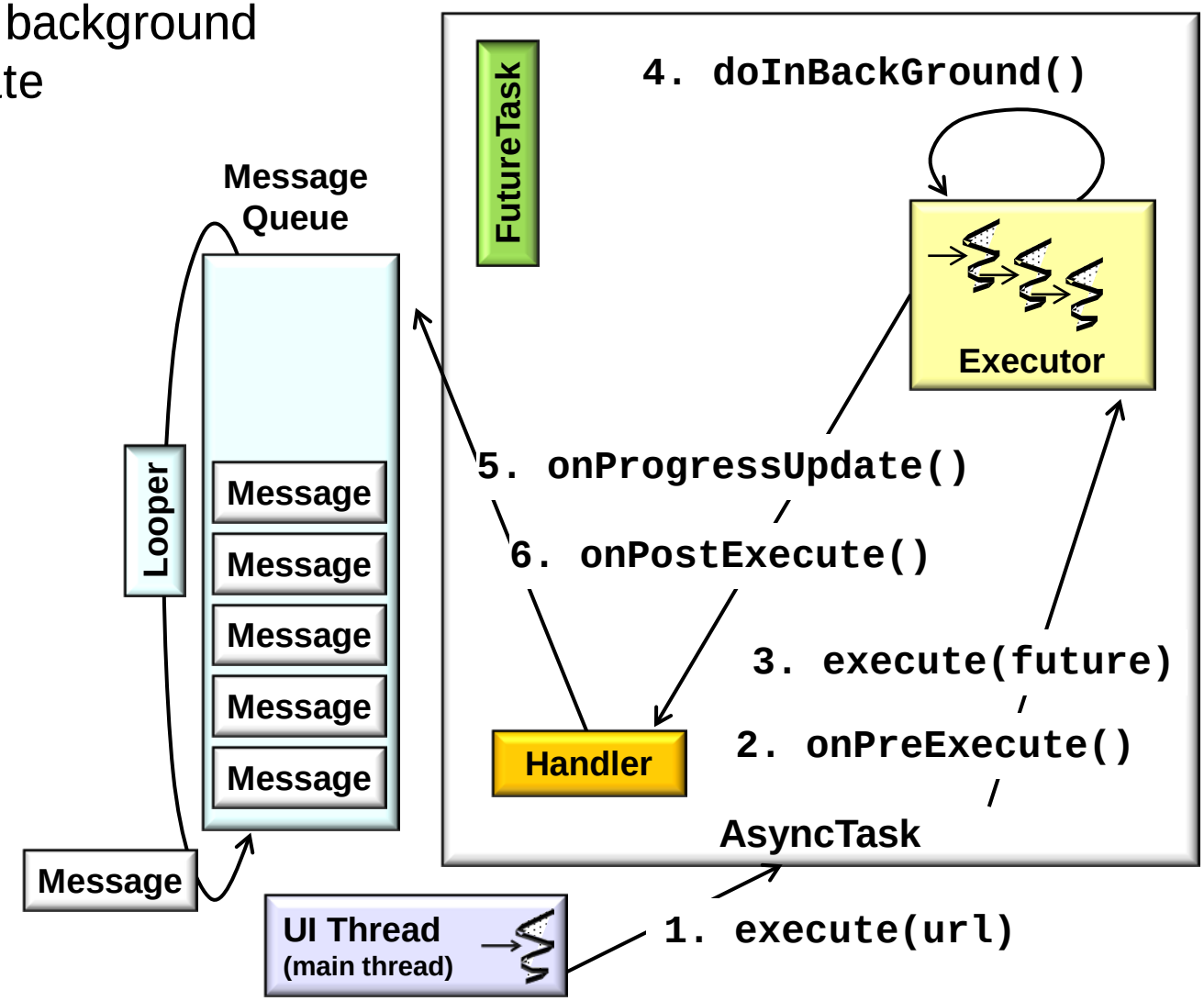
“Run Async” Behavior for SimpleImageDownloads



AsyncTask Usage Considerations

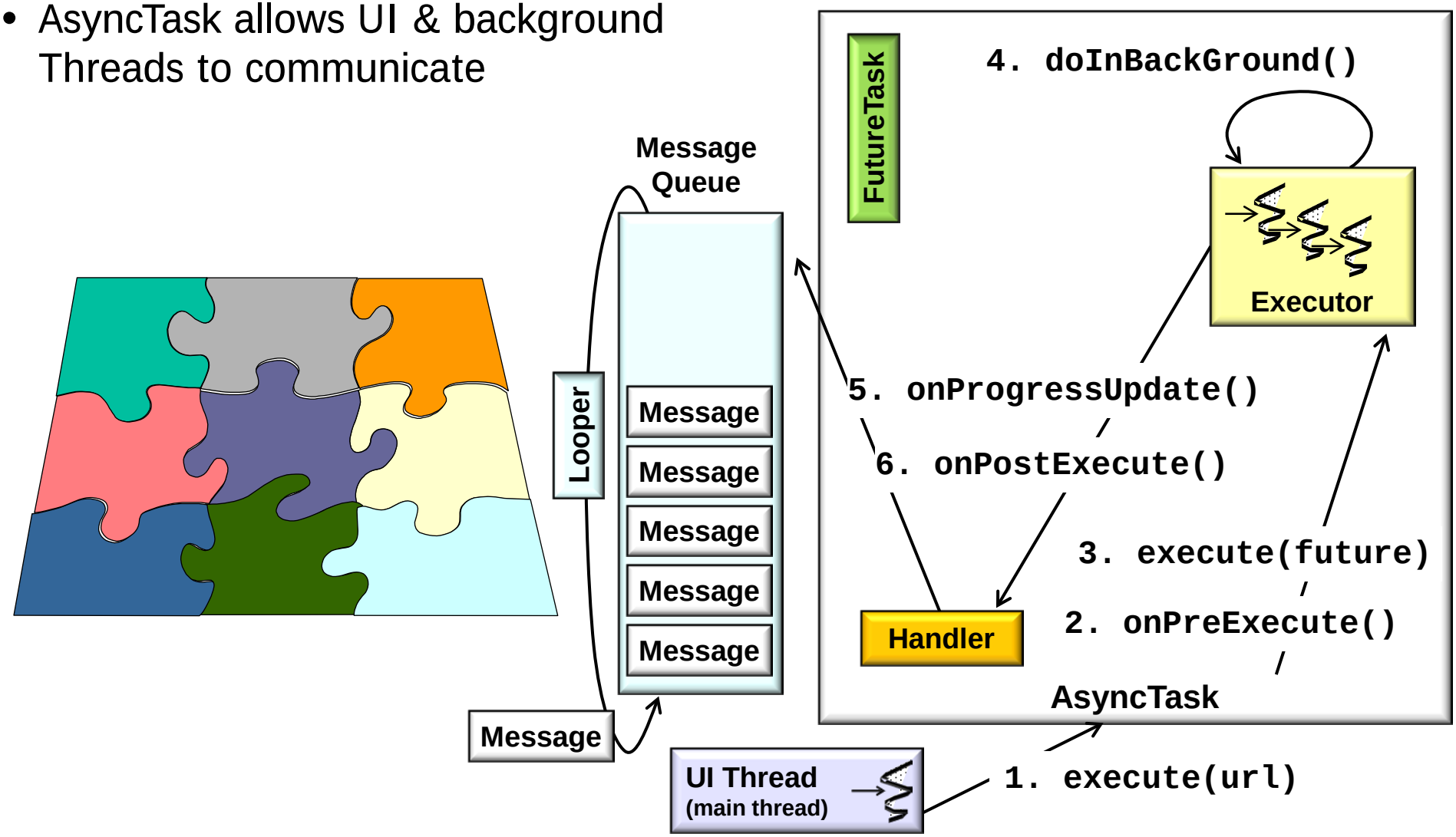
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate



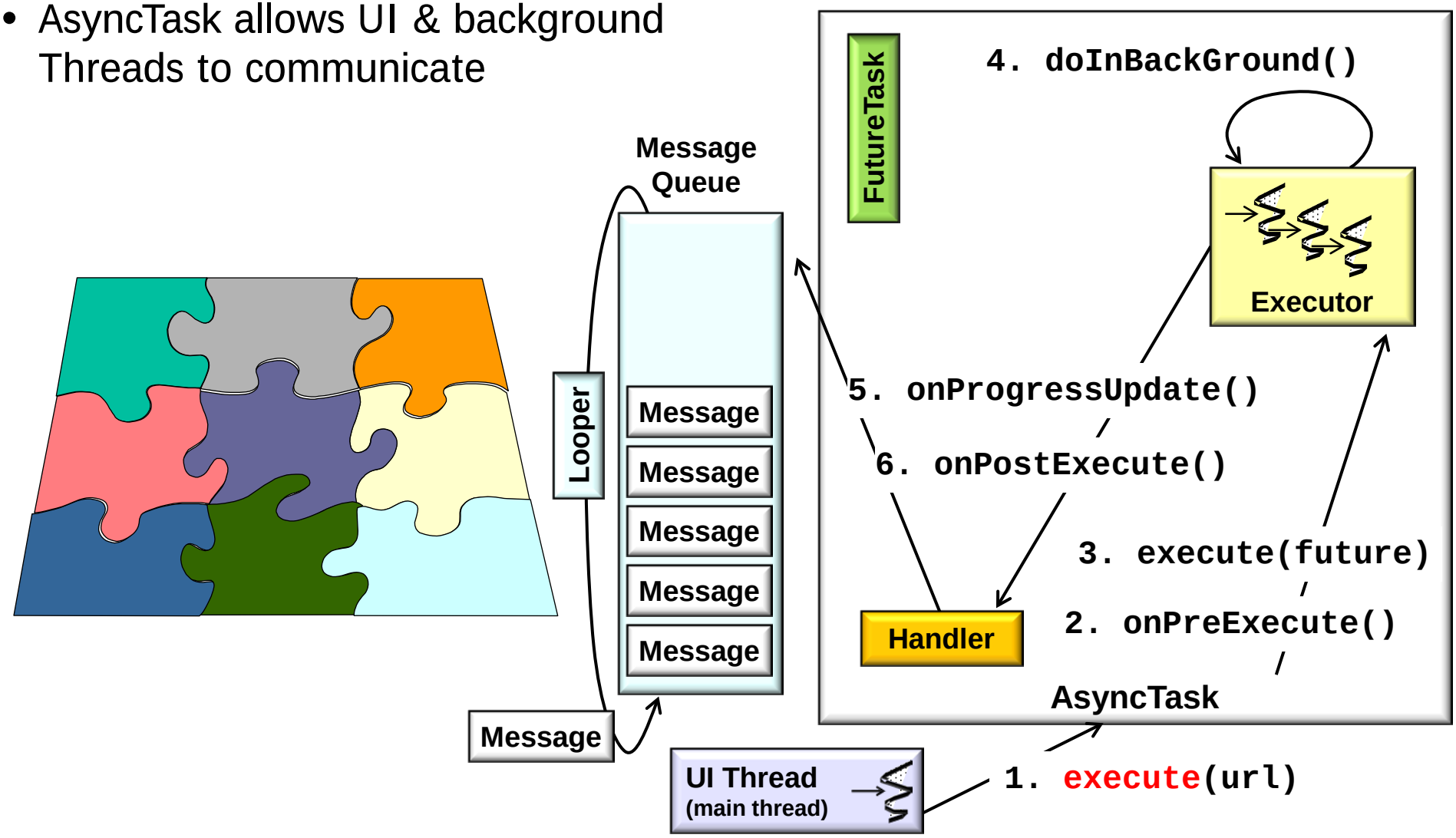
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate



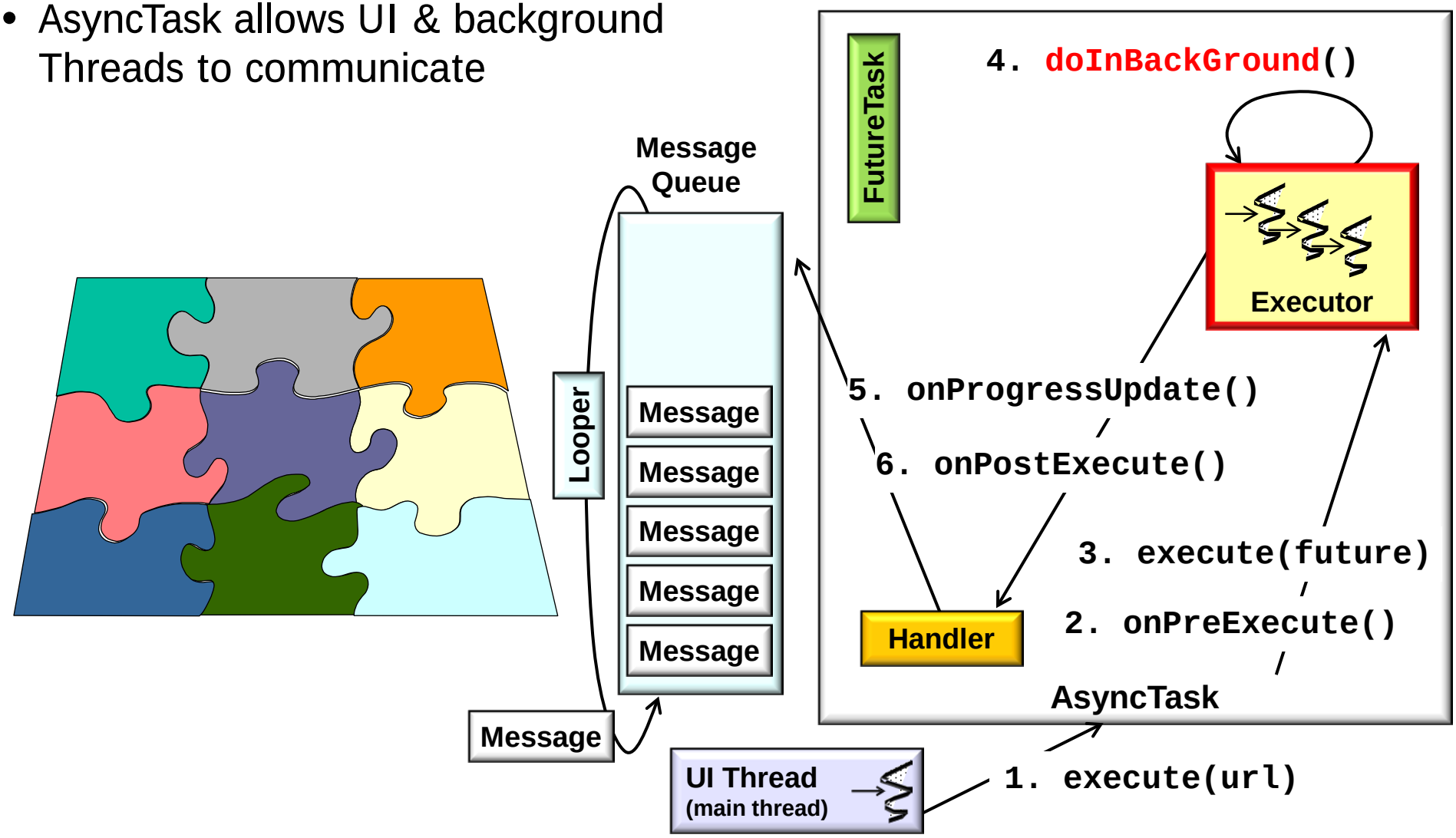
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate



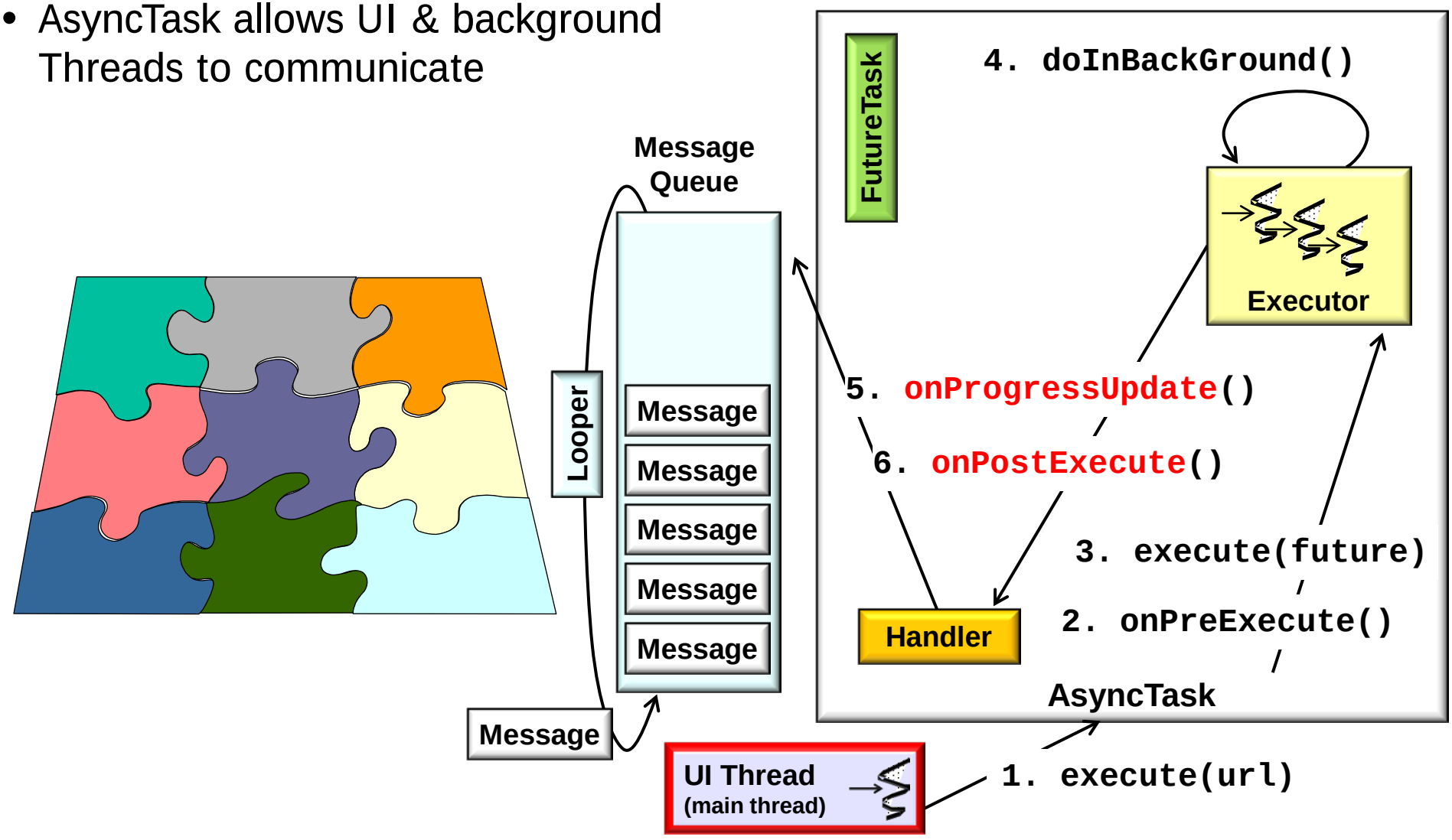
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate



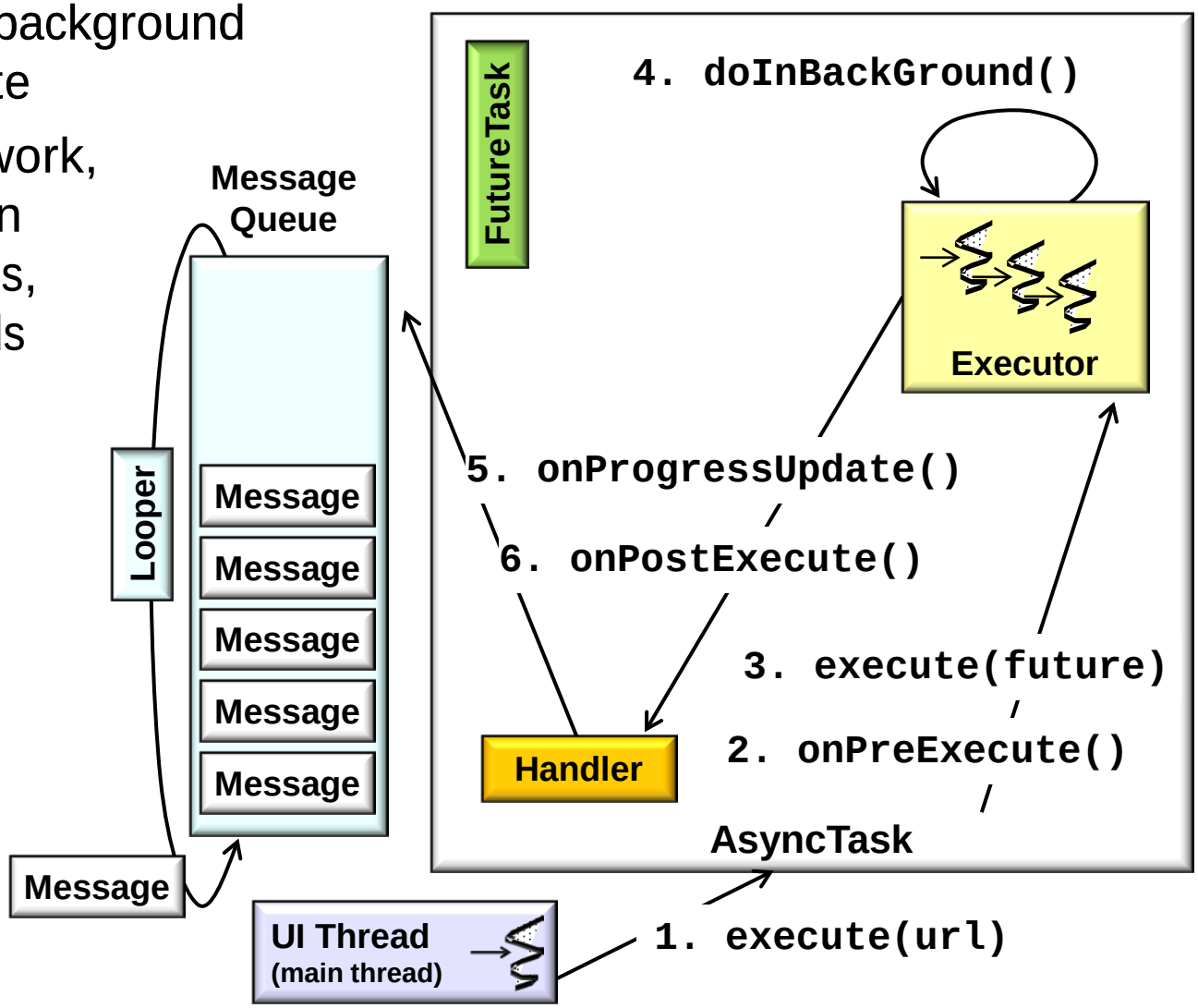
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate



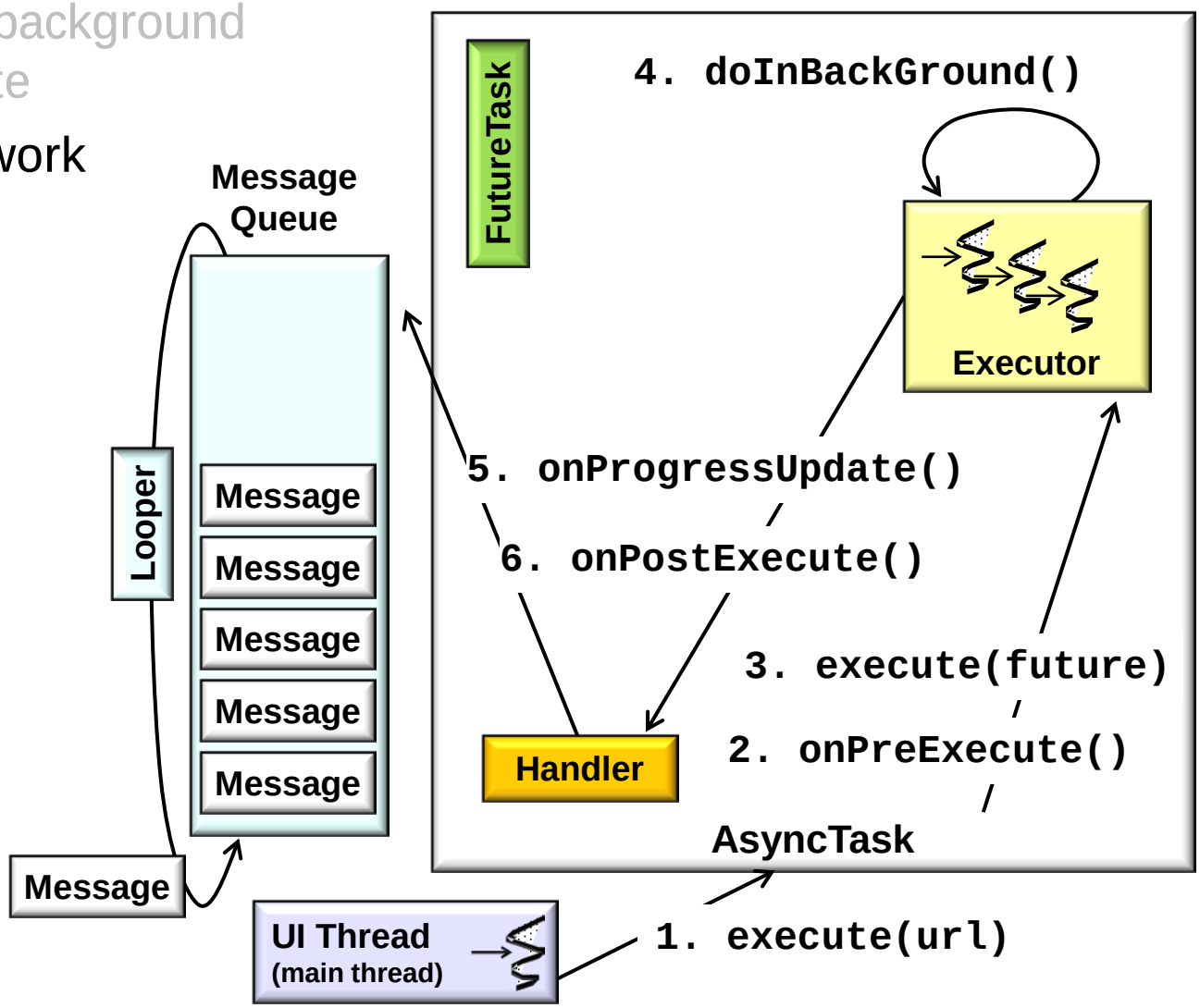
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
  - Unlike HaMeR framework, no direct manipulation of Handlers, Messages, Runnables, or Threads



AsyncTask Usage Considerations

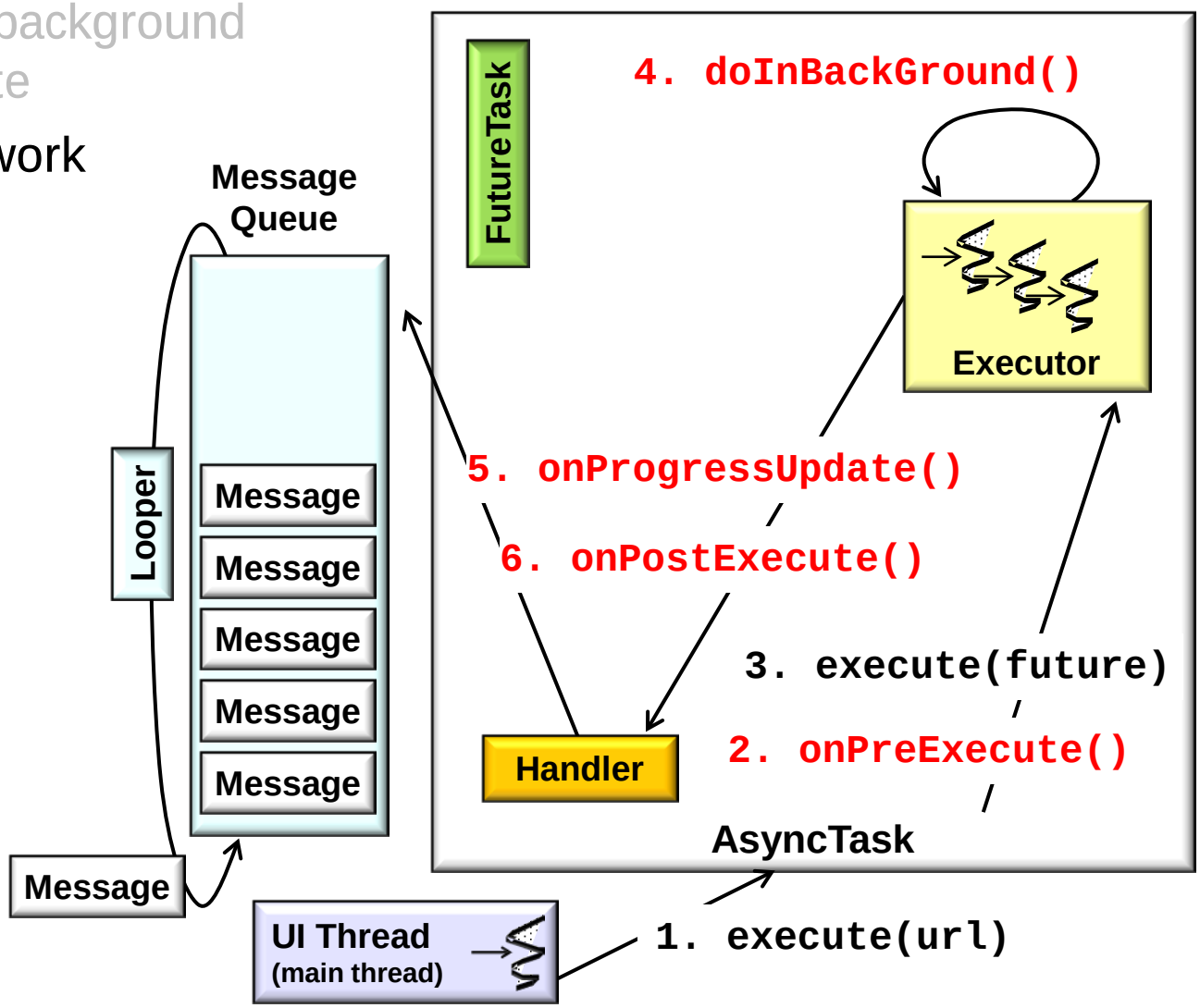
- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics



See earlier discussions on “Android of Concurrency Patterns & Frameworks”

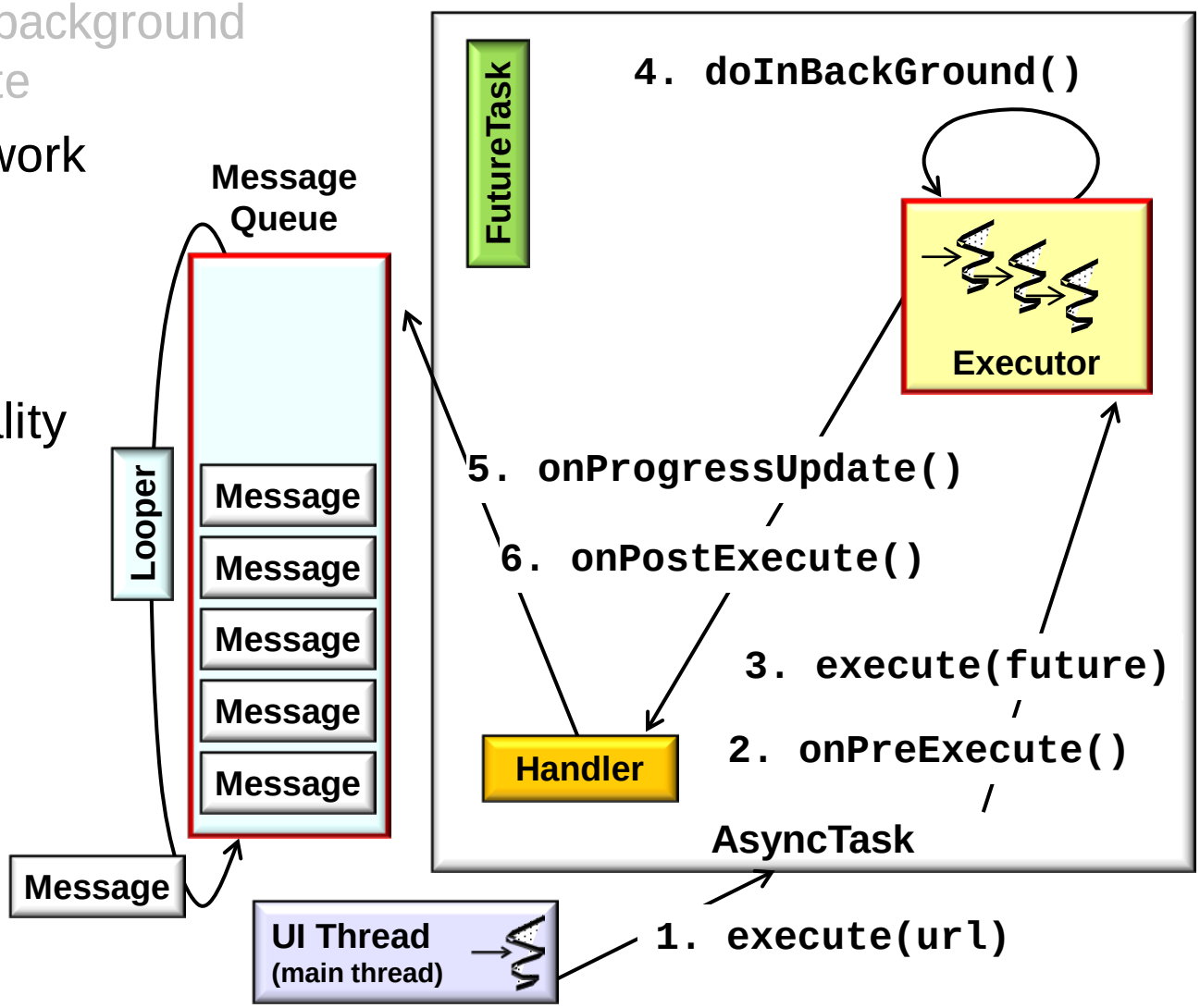
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics, e.g.
  - Inversion of control



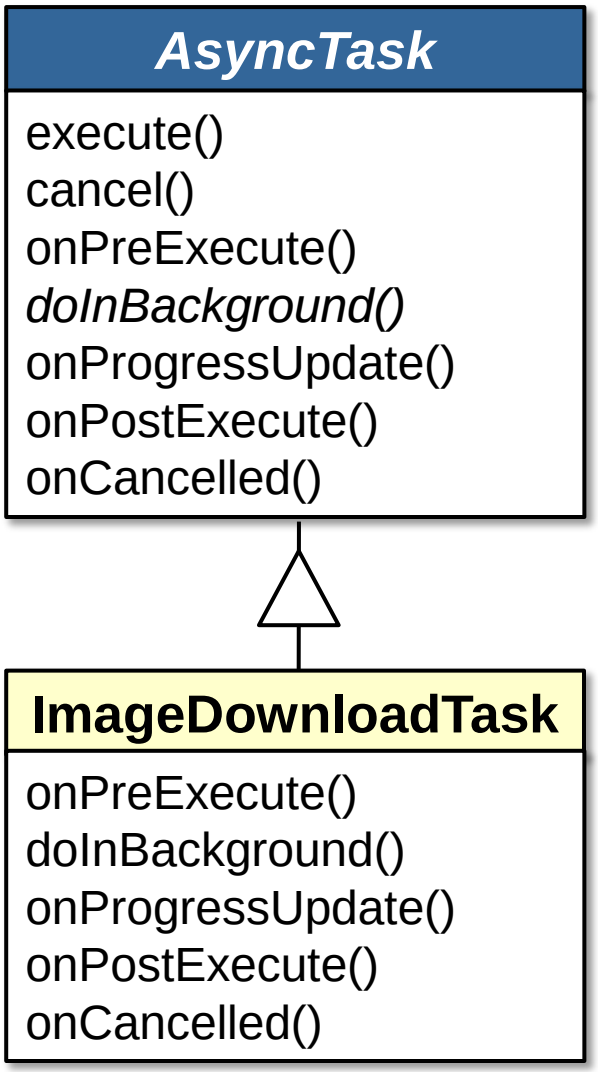
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics, e.g.
  - Inversion of control
  - Domain-specific structure & functionality



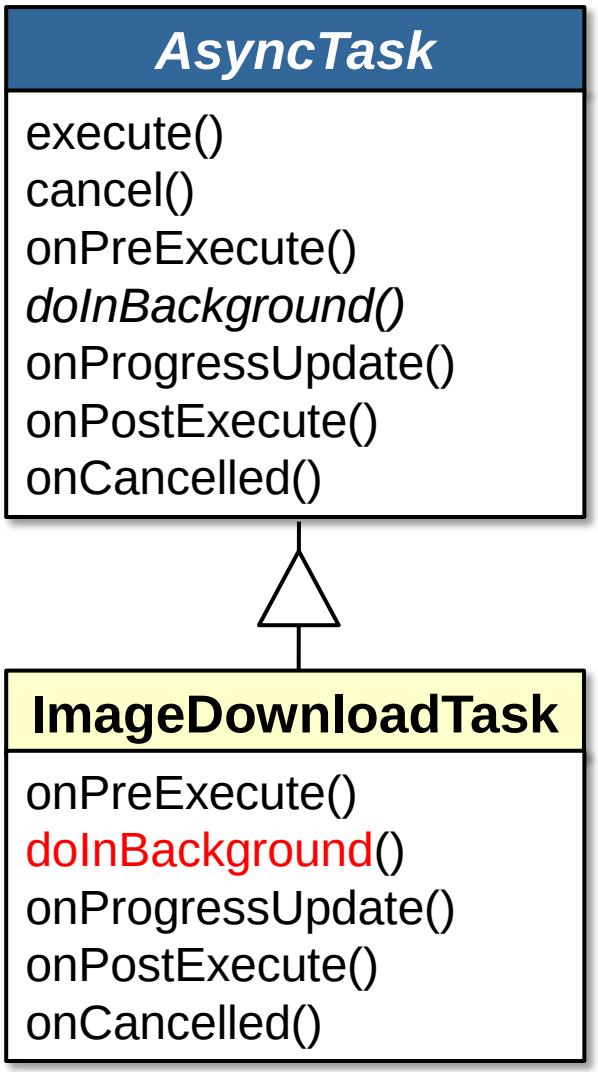
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps



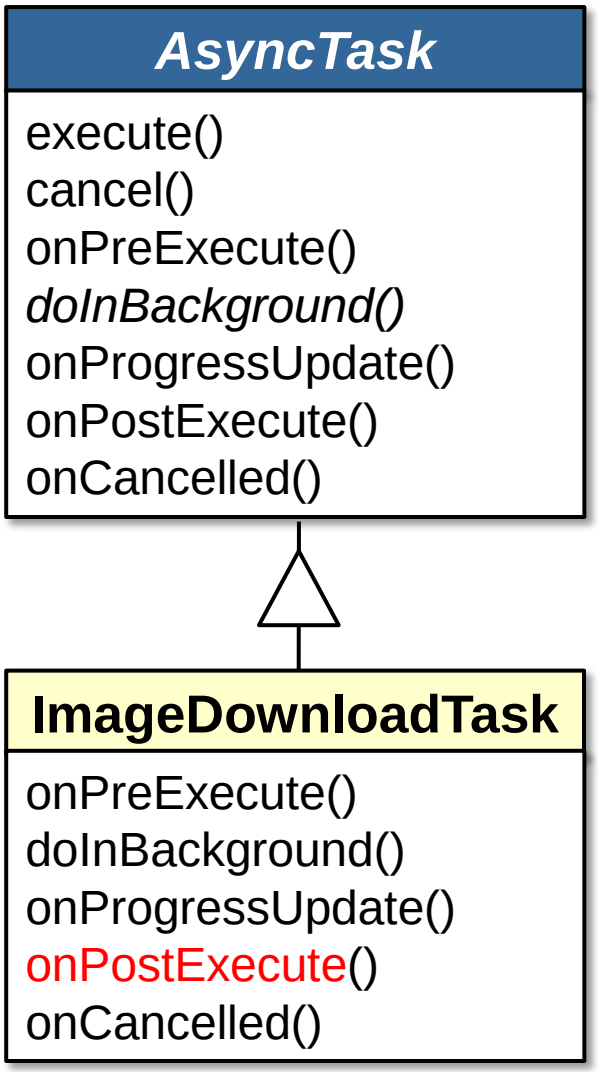
AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps



AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps



AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls

See bon-app-etit.blogspot.com/2013/04/the-dark-side-of-asynctask.html

AsyncTask Usage Considerations

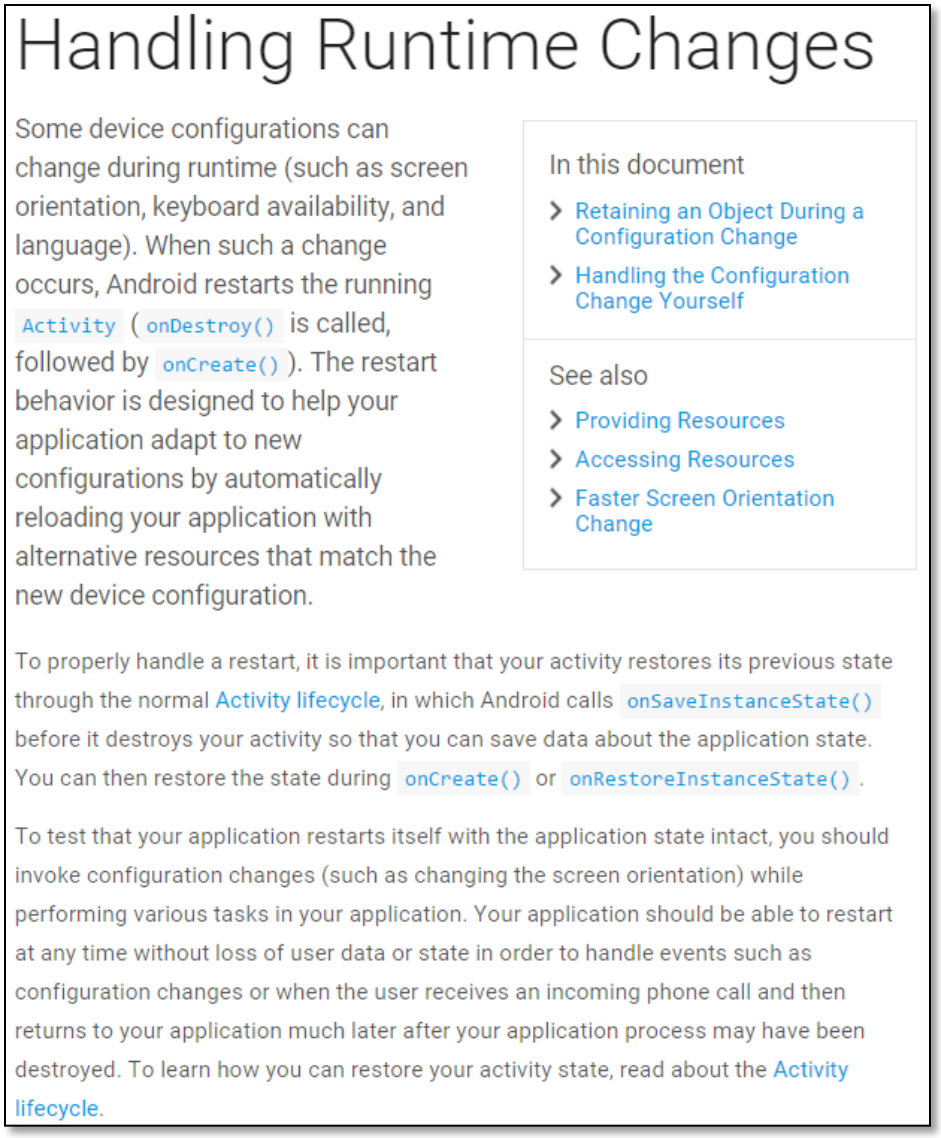
- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls
  - Cancellation
    - Cancellation is voluntary, just like Thread.interrupt()

AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
    - Memory leaks occur if there's a strong references to enclosing Activity

AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
    - e.g., Activity associated with an AsyncTask may be destroyed



Handling Runtime Changes

Some device configurations can change during runtime (such as screen orientation, keyboard availability, and language). When such a change occurs, Android restarts the running `Activity` (`onDestroy()` is called, followed by `onCreate()`). The restart behavior is designed to help your application adapt to new configurations by automatically reloading your application with alternative resources that match the new device configuration.

In this document

- Retaining an Object During a Configuration Change
- Handling the Configuration Change Yourself

See also

- Providing Resources
- Accessing Resources
- Faster Screen Orientation Change

To properly handle a restart, it is important that your activity restores its previous state through the normal Activity lifecycle, in which Android calls `onSaveInstanceState()` before it destroys your activity so that you can save data about the application state. You can then restore the state during `onCreate()` or `onRestoreInstanceState()` .

To test that your application restarts itself with the application state intact, you should invoke configuration changes (such as changing the screen orientation) while performing various tasks in your application. Your application should be able to restart at any time without loss of user data or state in order to handle events such as configuration changes or when the user receives an incoming phone call and then returns to your application much later after your application process may have been destroyed. To learn how you can restore your activity state, read about the Activity lifecycle.

AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
  - Portability
    - Concurrency semantics of AsyncTask execute() have changed over time

Before API 1.6 (Donut):

- In the first version of AsyncTask, the tasks were executed serially, so a task won't start before a previous task is finished. This caused quite some performance problems. One task had to wait on another one to finish.

API 1.6 to API 2.3 (Gingerbread):

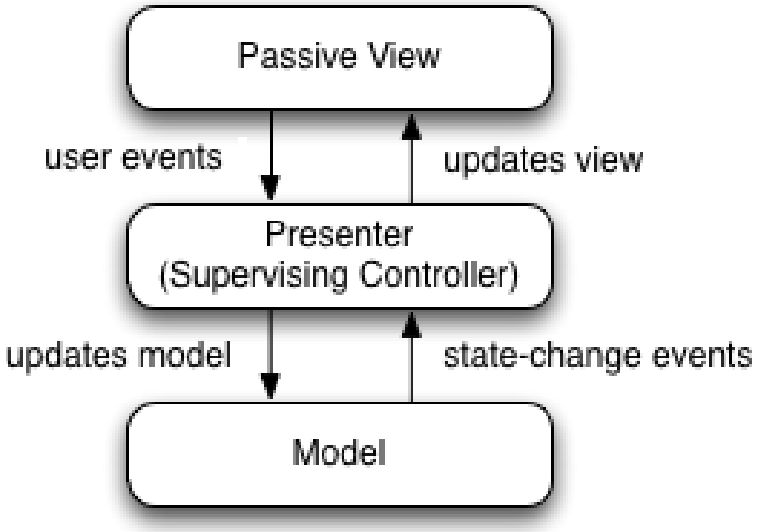
- The Android developers team decided to change this so that AsyncTasks could run parallel on a separate worker thread. There was one problem. Many developers relied on the sequential behavior and suddenly they were having a lot of concurrency issues.

API 3.0 (Honeycomb) until now

- "Hmmm, developers don't seem to get it? Let's just switch it back." The AsyncTasks where executed serially again. However, they can run parallel via executeOnExecutor(Executor).

AsyncTask Usage Considerations

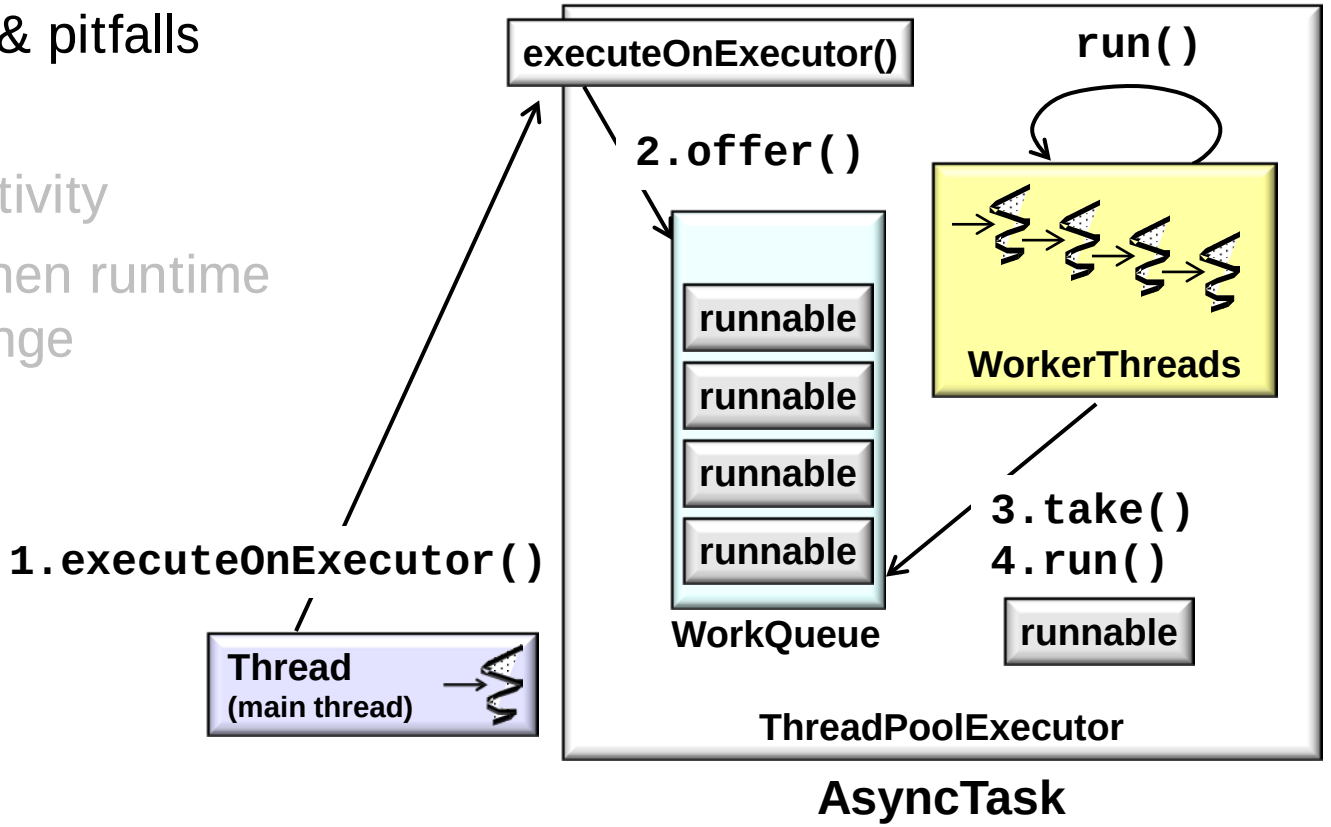
- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
  - Portability



The *Model-View Presenter* (MVP) pattern addresses some of these issues

AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
  - Portability



Other issues can be addressed by understanding Android patterns & APIs

AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls
- AsyncTask used throughout Android

frameworks/base/core/java/android/content/AsyncTaskLoader.java
frameworks/base/core/java/android/content/CursorLoader.java
frameworks/base/core/java/android/os/AsyncTask.java
packages/apps/Browser/src/com/android/browser/UrlHandler.java
packages/apps/Calendar/src/com/android/calendar/CalendarController.java
packages/apps/Gallery/src/com/android/camera/ReverseGeocoderTask.java
packages/apps/Nfc/src/com/android/nfc/NfcService.java
packages/apps/Mms/src/com/android/mms/transaction/PushReceiver.java
packages/apps/Phone/src/com/android/phone/CallLogAsync.java
packages/apps/VideoEditor/src/com/android/videoeditor/BaseAdapterWithImages.java
...

AsyncTask Usage Considerations

- AsyncTask allows UI & background Threads to communicate
- It embodies key framework characteristics
- AsyncTask has traps & pitfalls

- AsyncTask used throughout Android
- onProgressUpdate() is not widely used

frameworks/base/media/java/android/media/videoeditor/MediaArtistNativeHelper.java
frameworks/base/packages/SystemUI/src/com/android/systemui/recent/
    RecentTasksLoader.java
packages/apps/Email/emailcommon/src/com/android/emailcommon/utility/
    EmailAsyncTask.java
packages/apps/Email/src/com/android/email/activity/setup/
    AccountCheckSettingsFragment.java
packages/apps/Gallery2/src/com/android/gallery3d/app/ManageCachePage.java
packages/apps/Gallery2/src/com/android/gallery3d/ui/ImportCompleteListener.java
packages/apps/Gallery2/src/com/android/gallery3d/ui/MenuExecutor.java
packages/apps/Settings/src/com/android/settings/TrustedCredentialsSettings.java