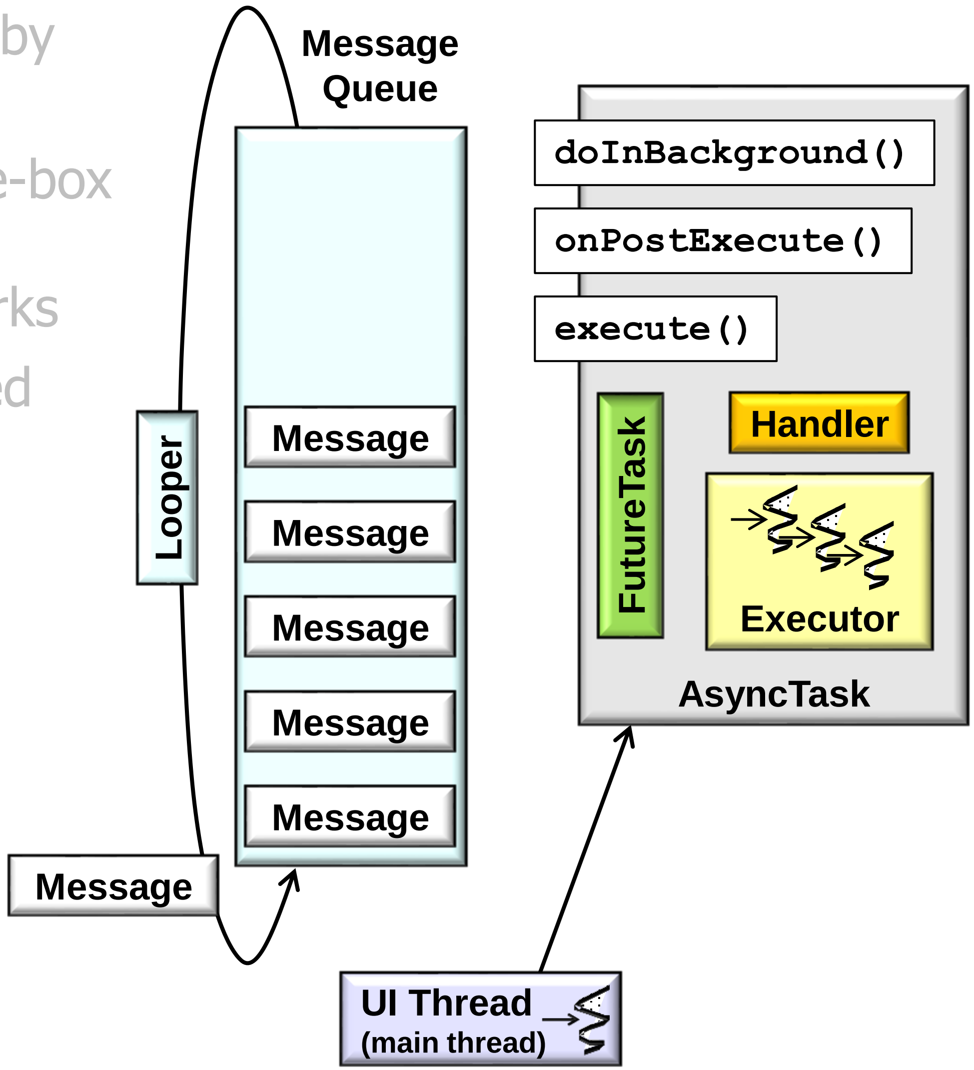
The AsyncTask Framework: Usage Considerations

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 Lesson

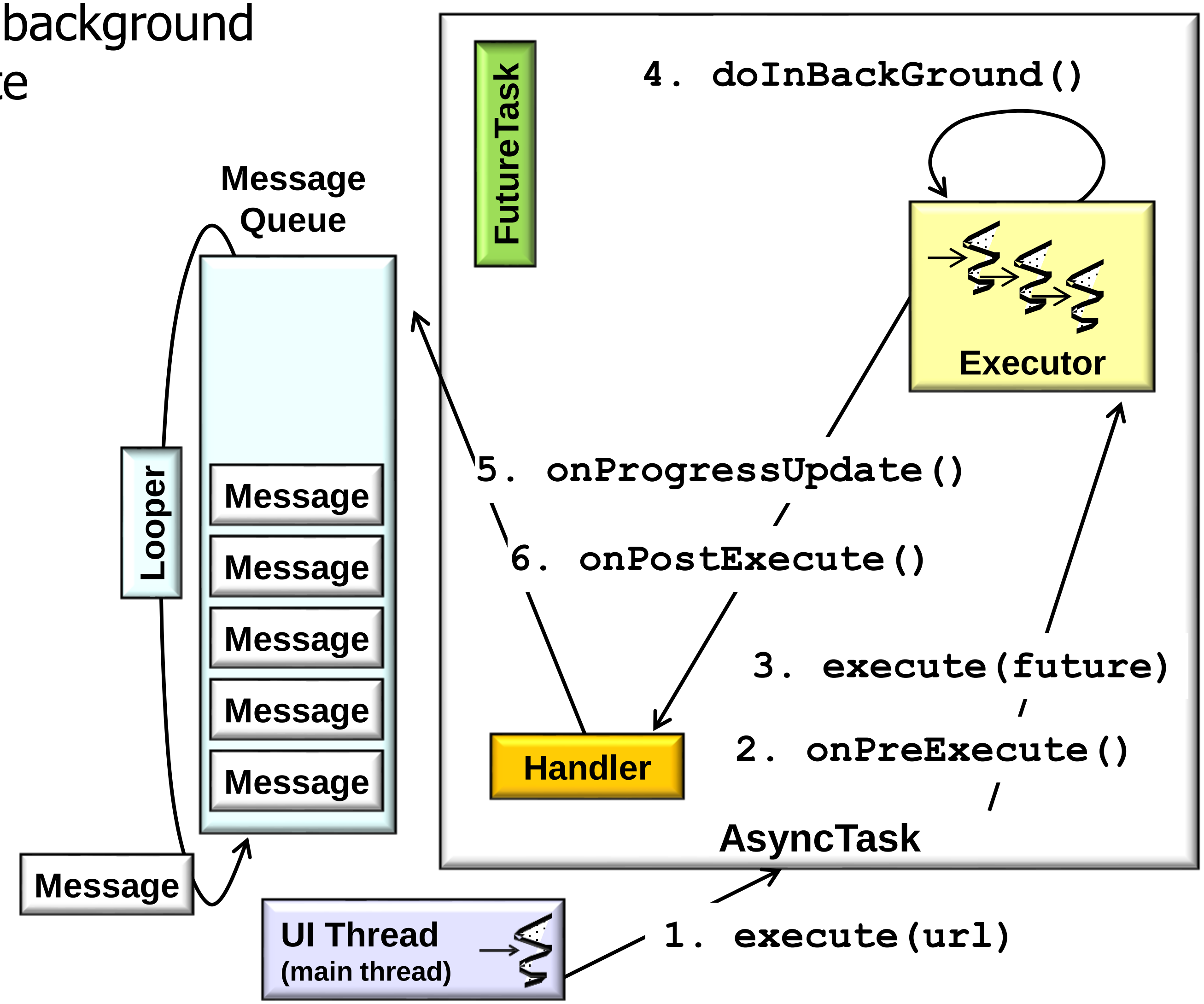
- Recognize the capabilities provided by the Android AsyncTask framework
- Know which methods are provided by AsyncTask class
- Understand what black-box & white-box framework are… & how AsyncTask implements both types of frameworks
- Learn how the AsyncTaskInterrupted program works
- Appreciate AsyncTask usage considerations



AsyncTask Usage Considerations

AsyncTask Usage Considerations

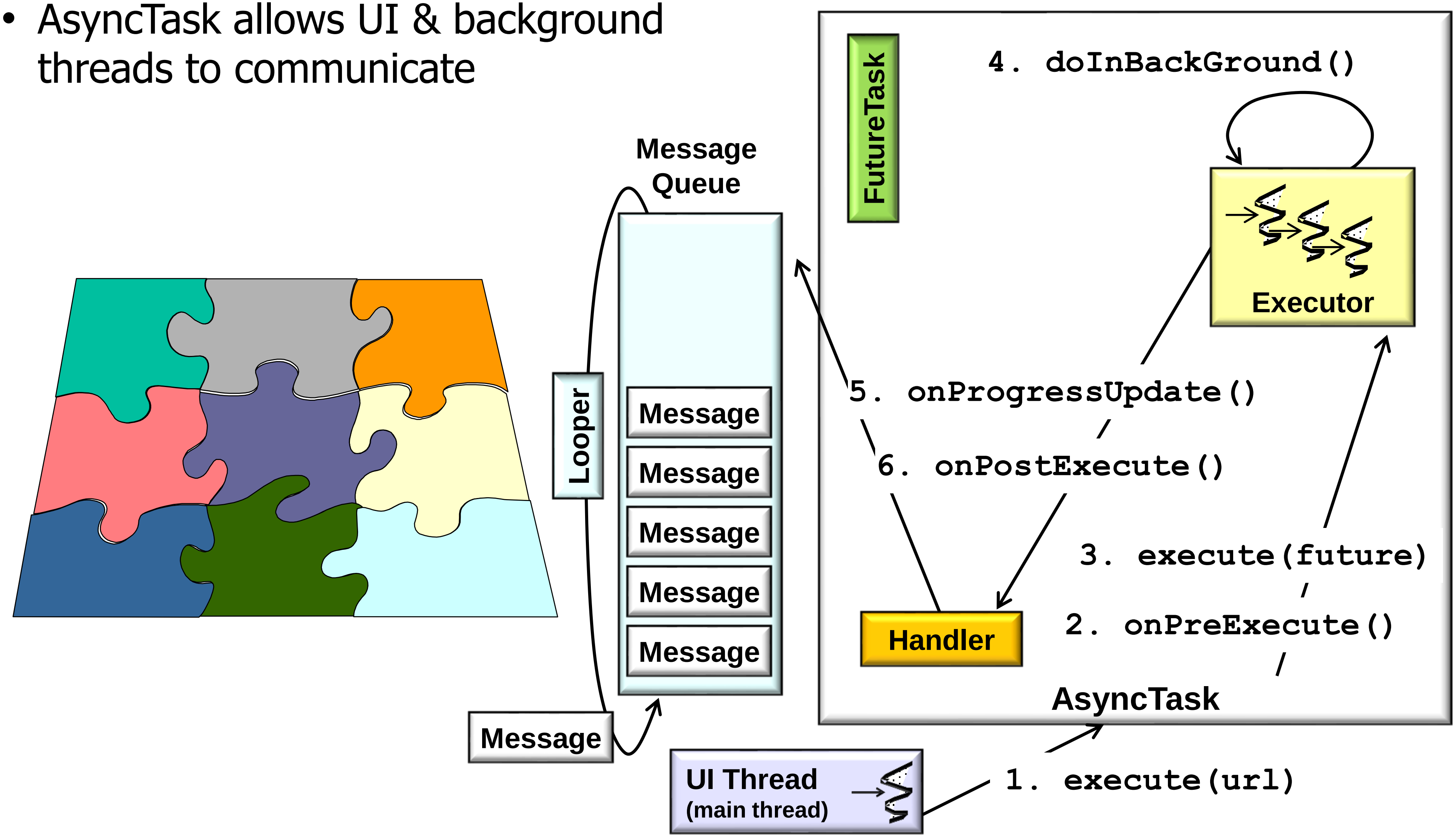
- AsyncTask allows UI & background threads to communicate



Its onPreExecute(), onProgressUpdate(), & onPostExecute() methods *always* run in the context of the UI thread!

AsyncTask Usage Considerations

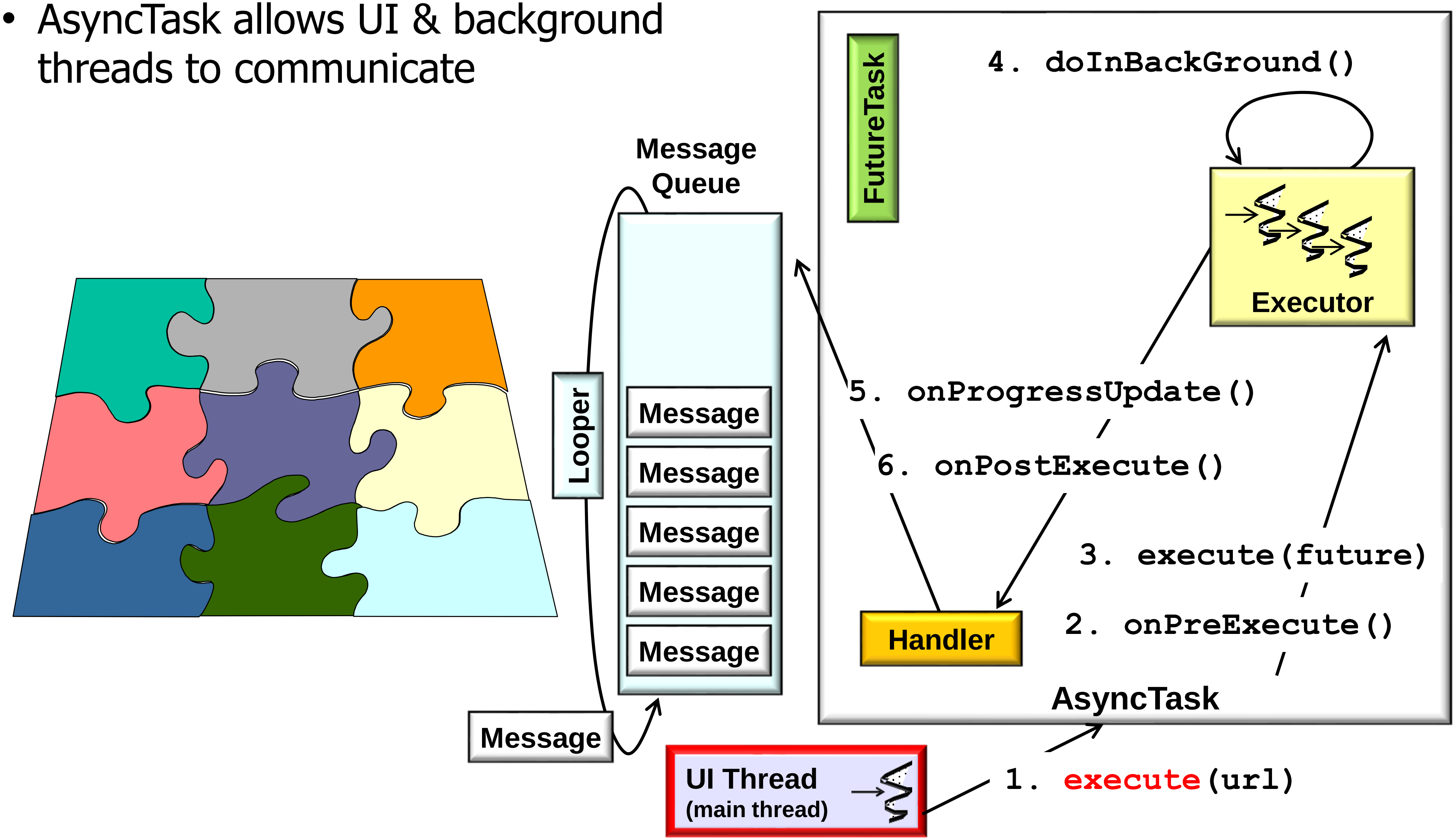
- AsyncTask allows UI & background threads to communicate



These methods are strongly connected via AsyncTask framework classes

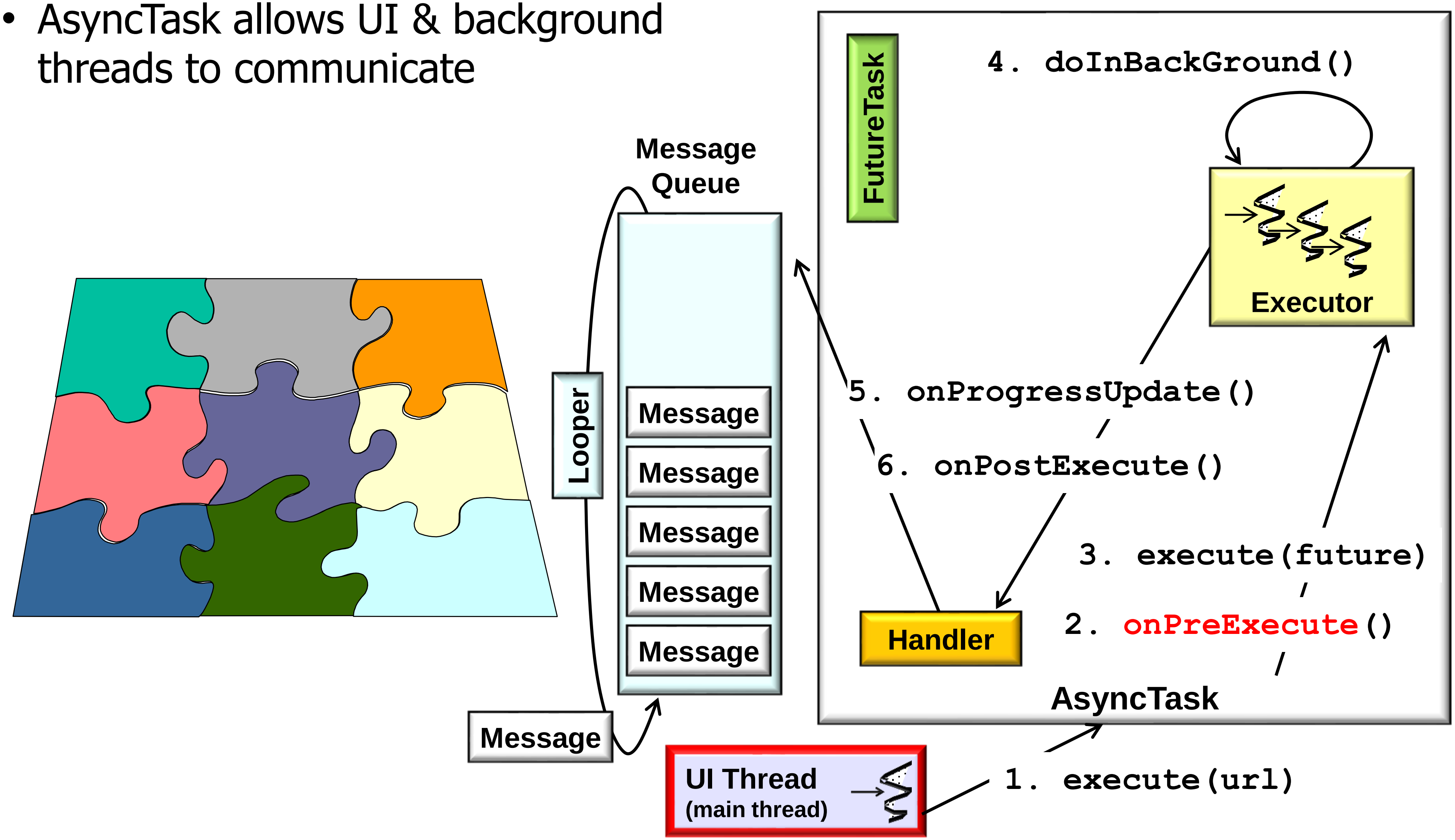
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate



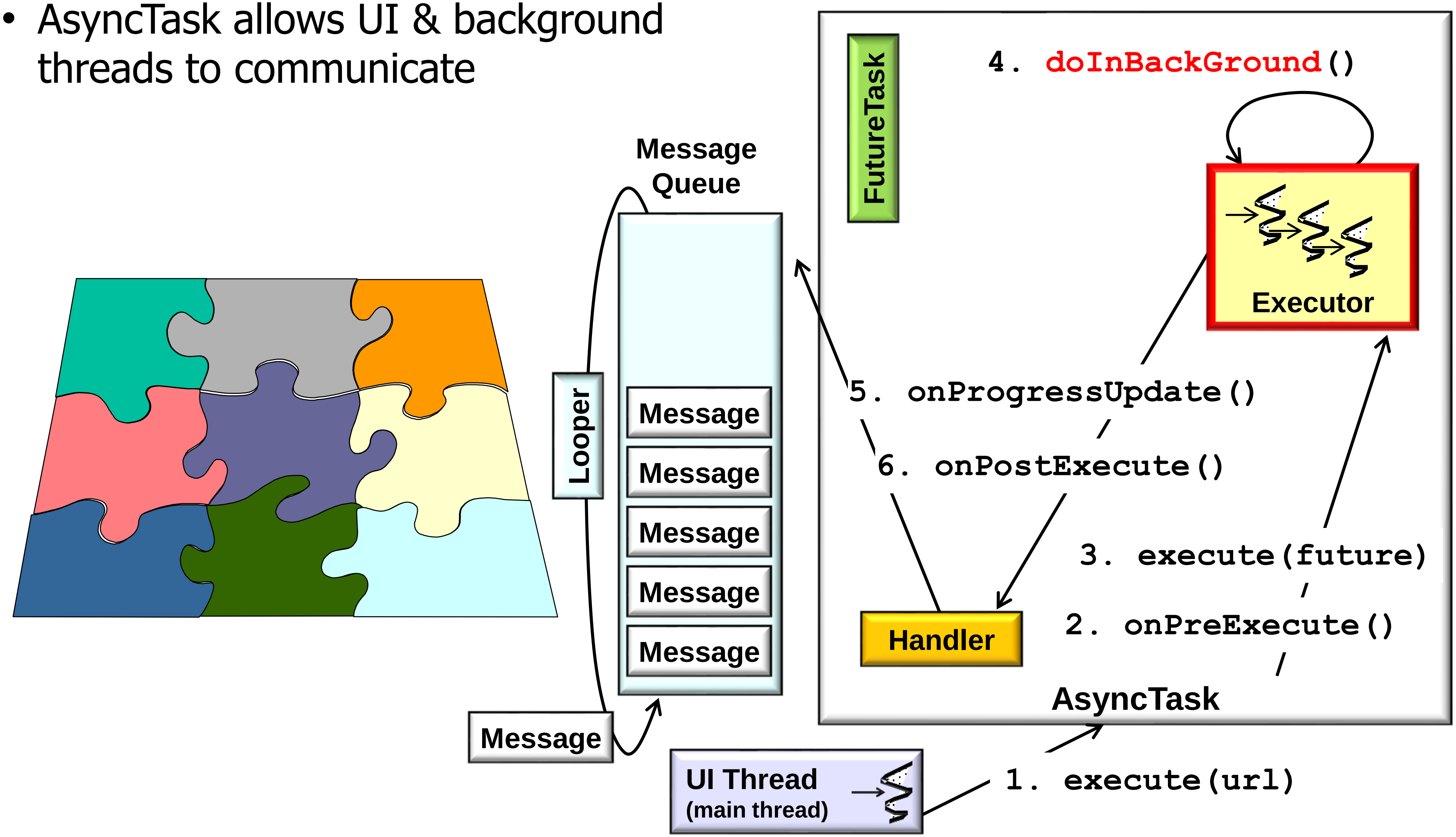
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate



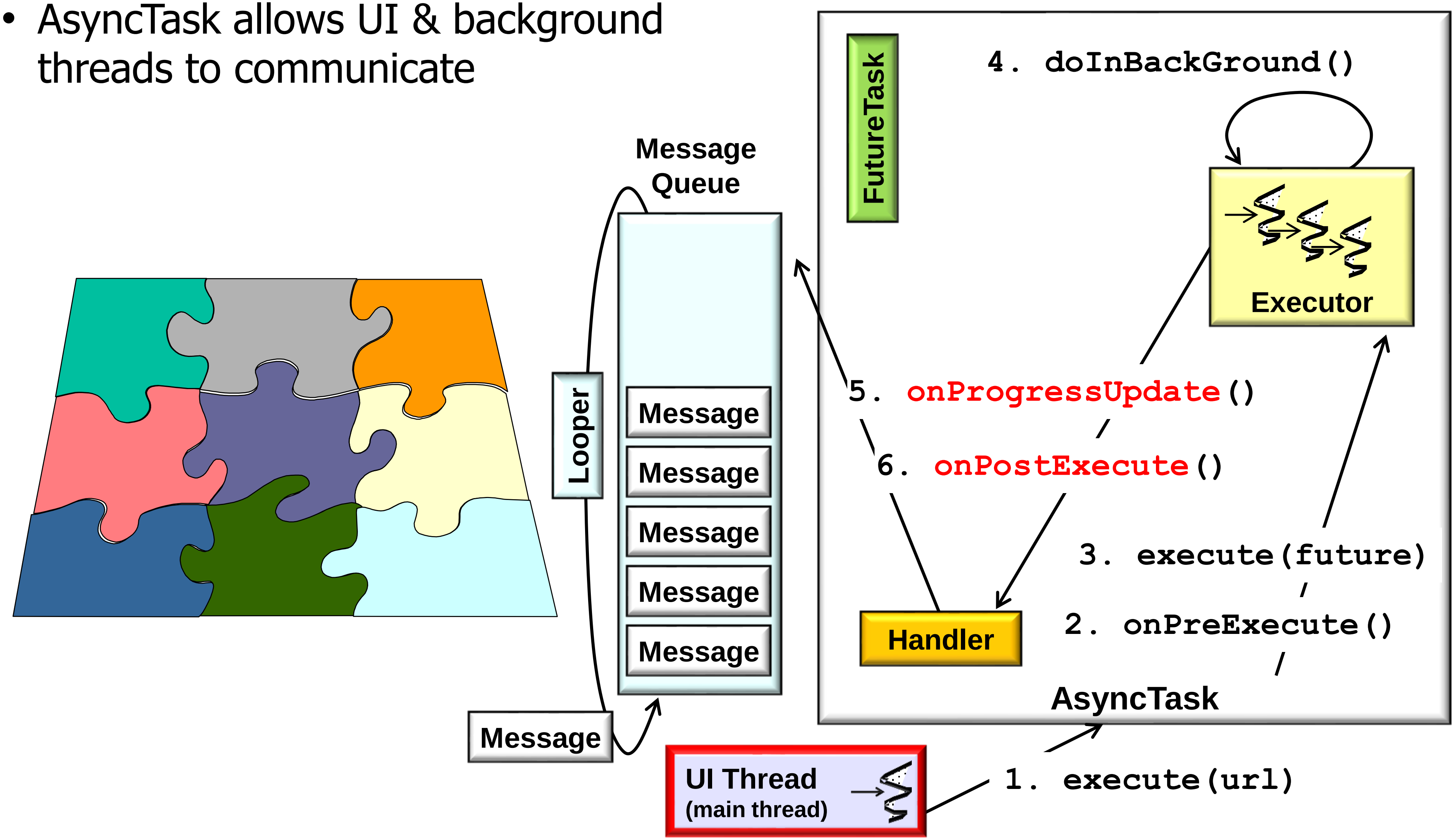
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate



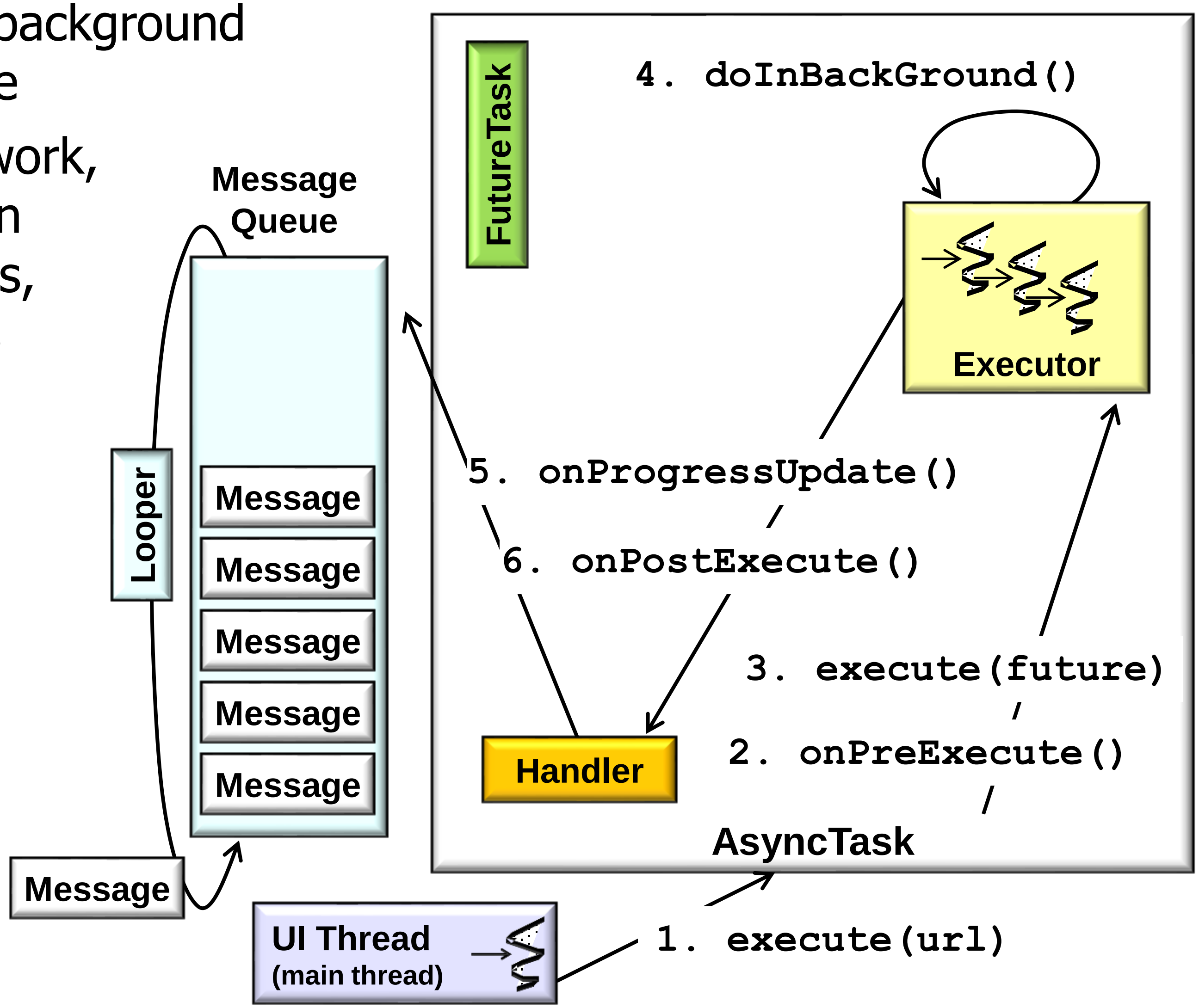
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate



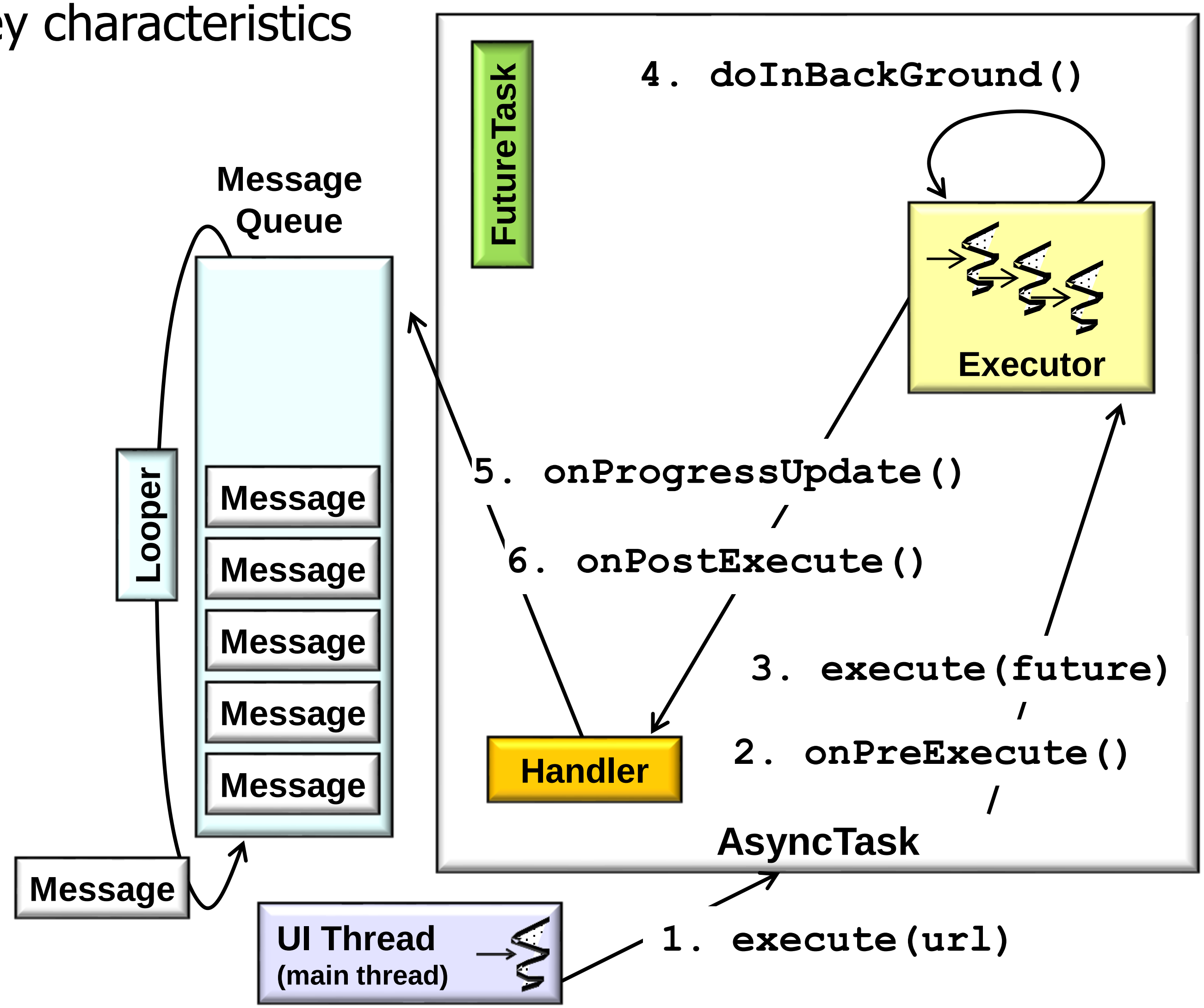
AsyncTask Usage Considerations

- AsyncTask allows UI & background threads to communicate
  - Unlike HaMeR framework, no direct manipulation of handlers, messages, runnables, or threads



AsyncTask Usage Considerations

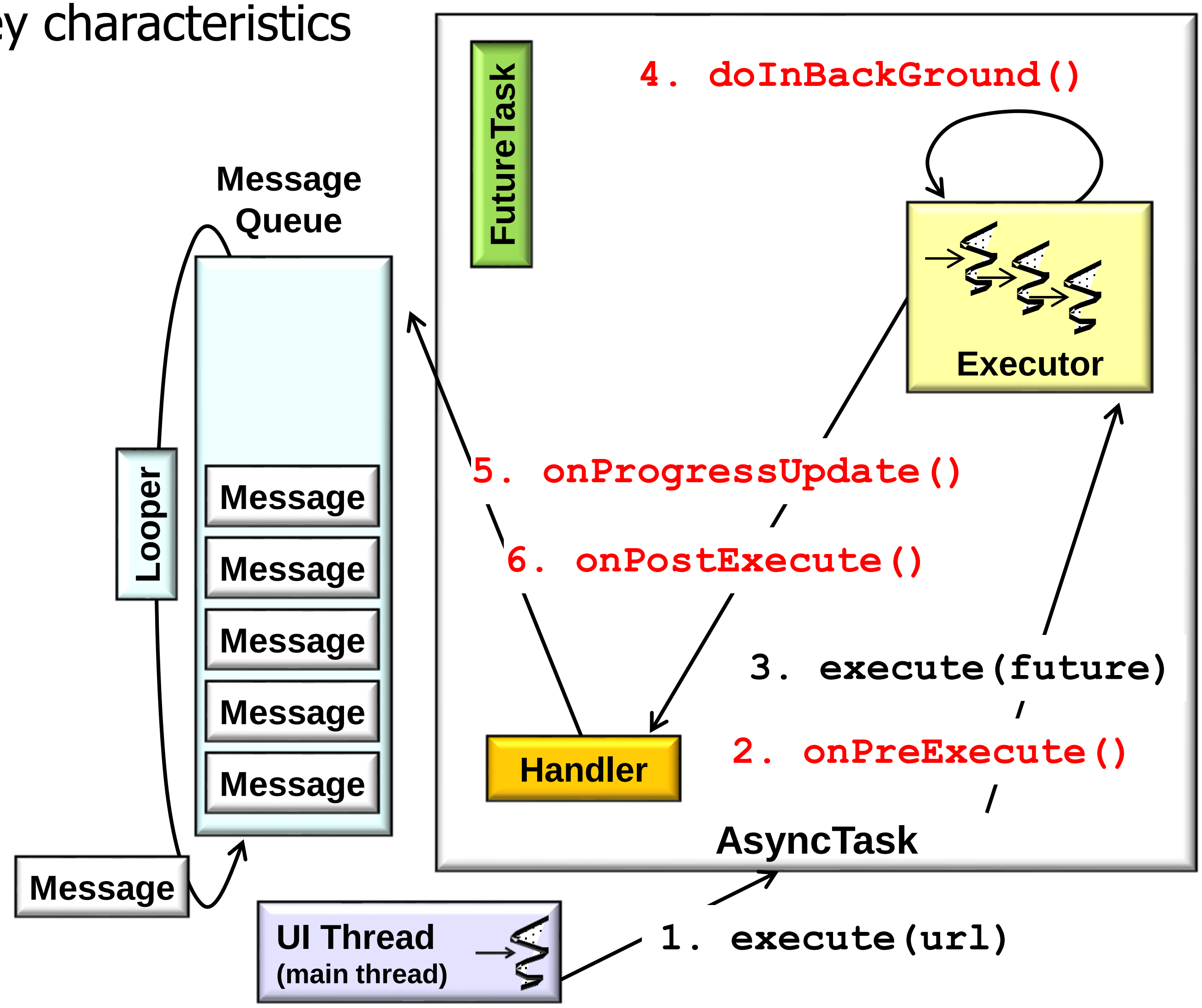
- AsyncTask embodies key characteristics of a framework



See www.dre.vanderbilt.edu/~schmidt/PDF/Queue-04.pdf

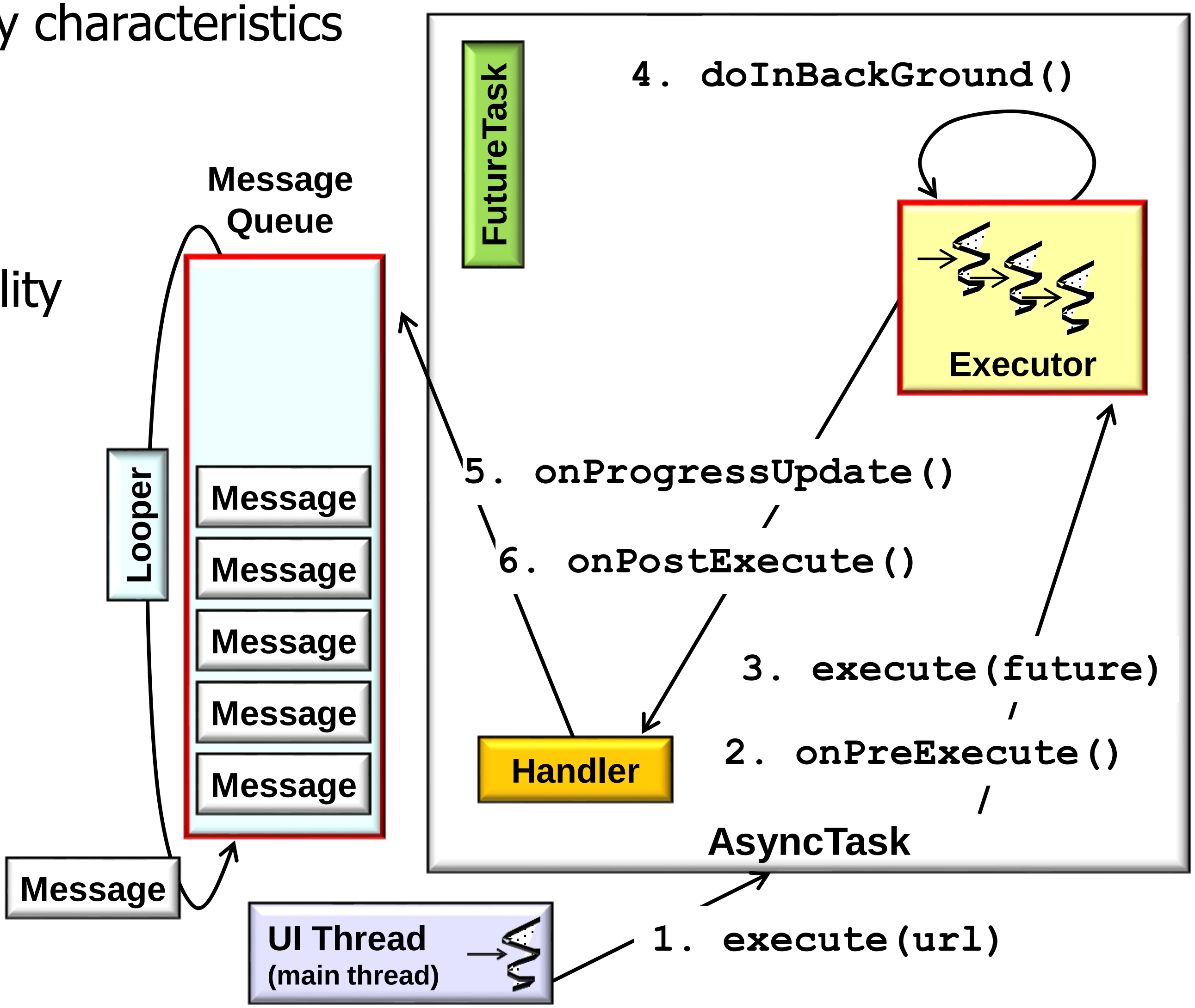
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control



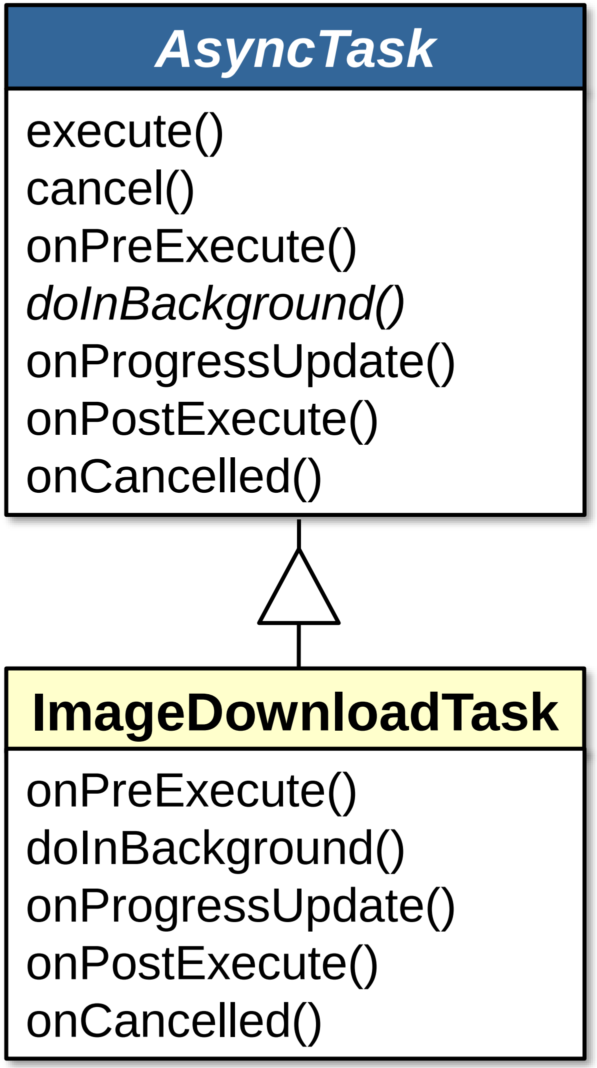
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control
  - Domain-specific structure & functionality



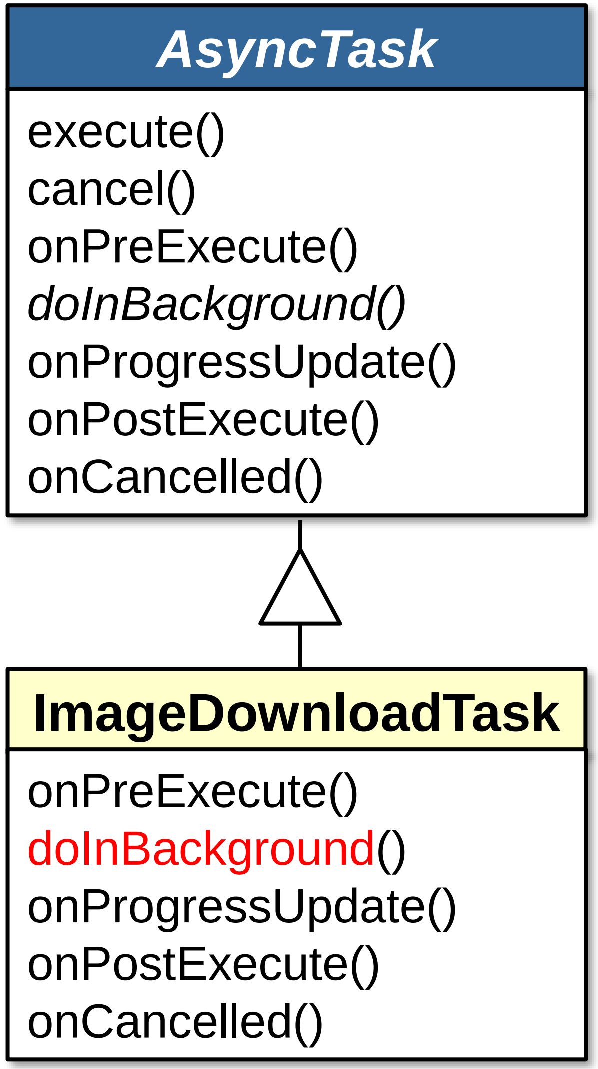
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps



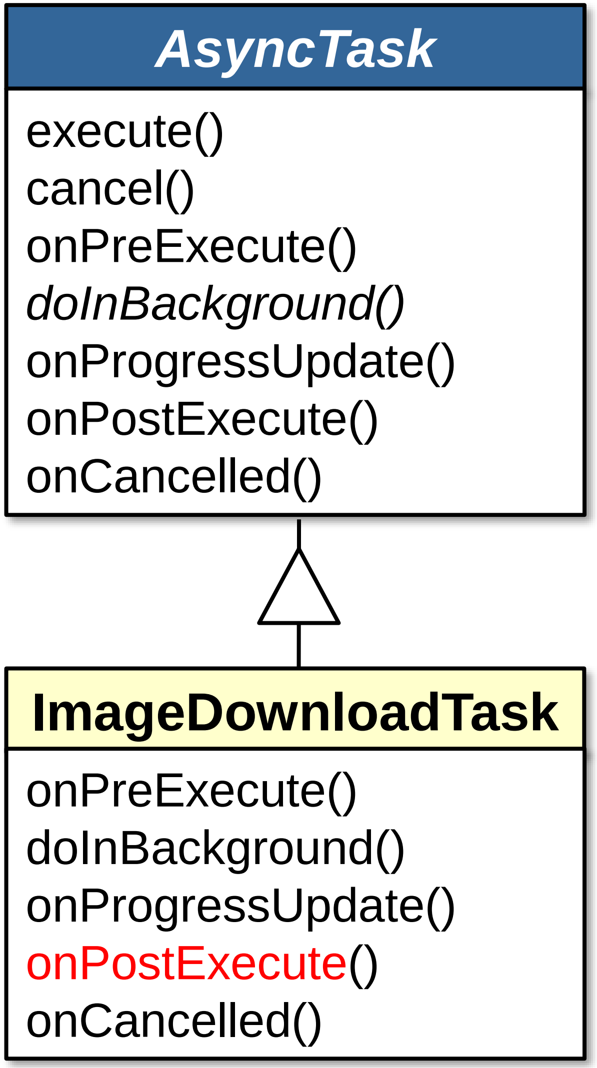
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps



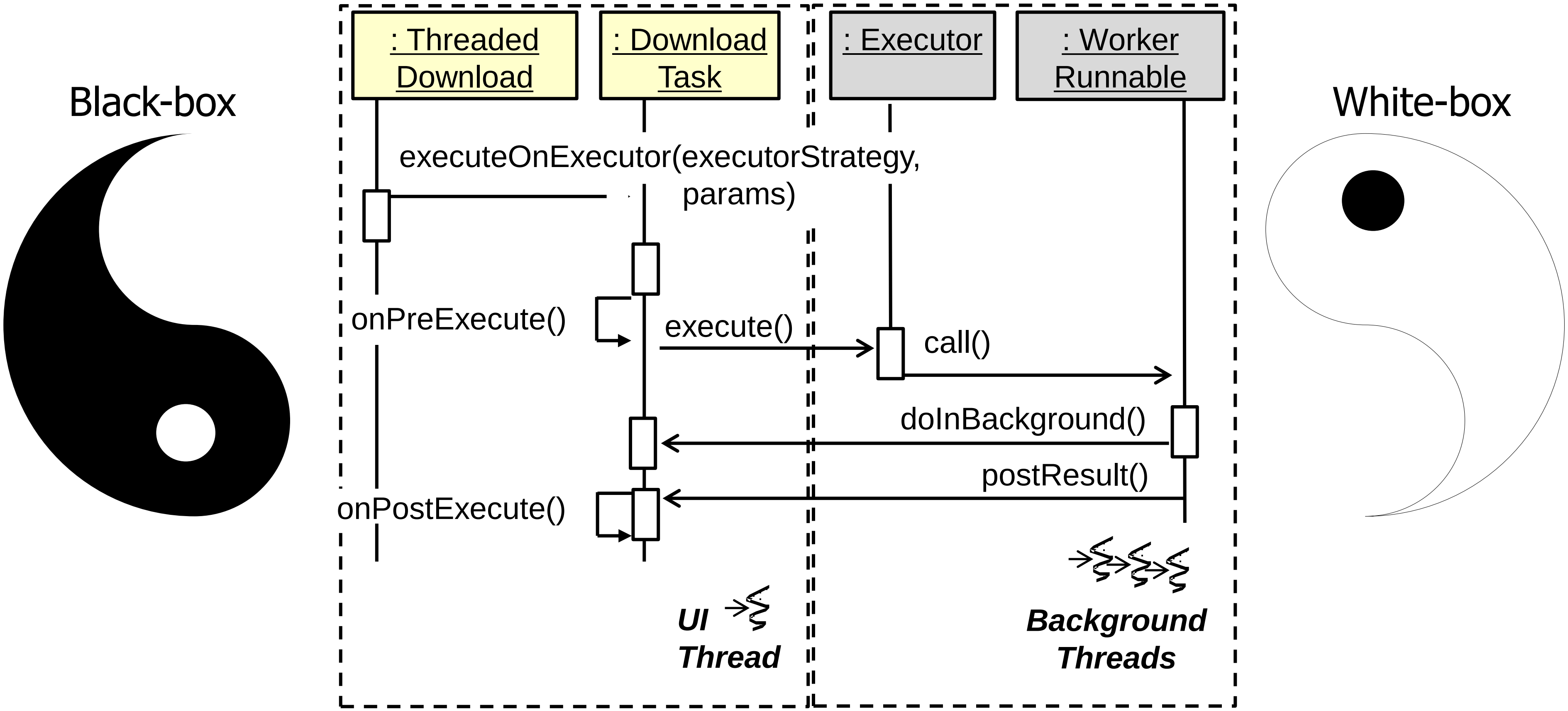
AsyncTask Usage Considerations

- AsyncTask embodies key characteristics of a framework, e.g.
  - Inversion of control
  - Domain-specific structure & functionality
  - Semi-complete portions of apps



AsyncTask Usage Considerations

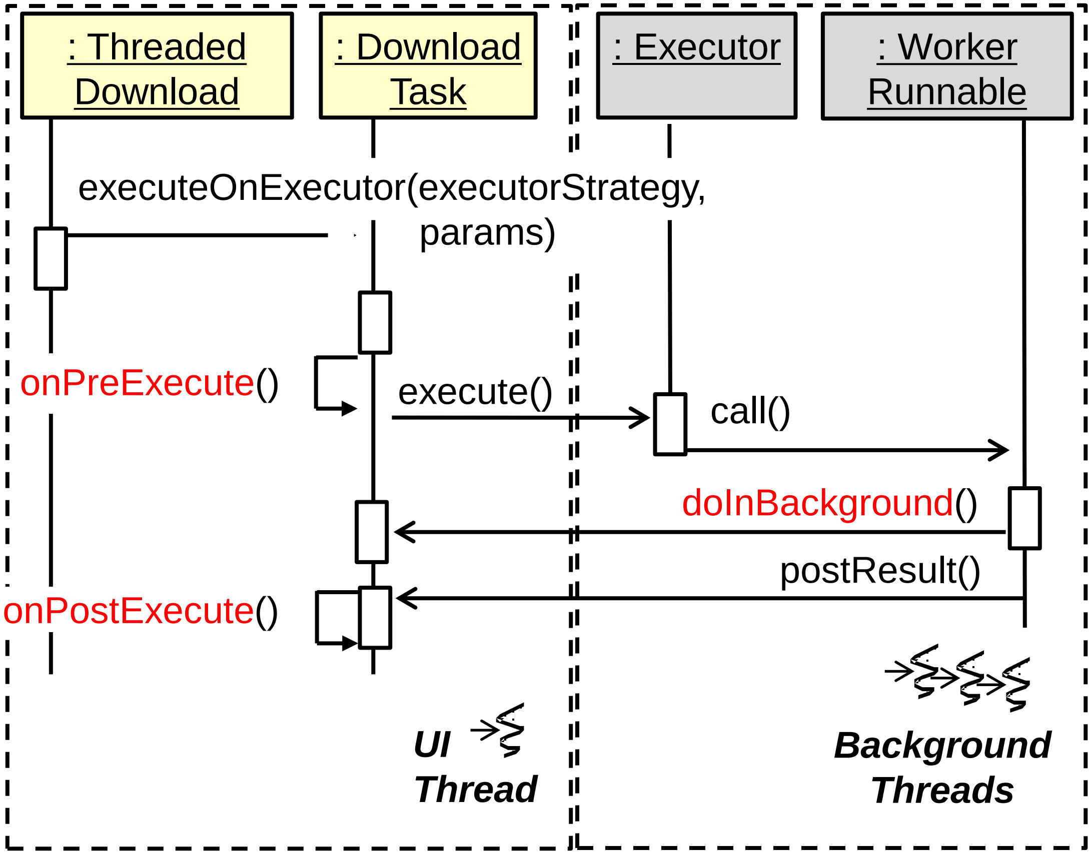
- AsyncTask has elements of both black-box & white-box frameworks



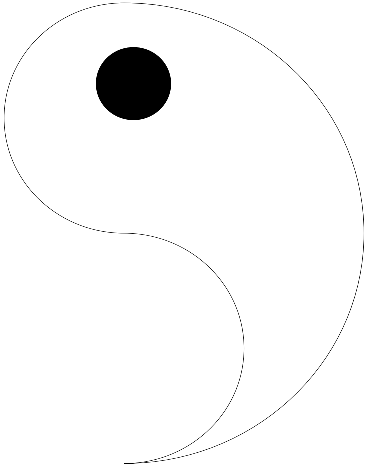
AsyncTask Usage Considerations

- AsyncTask has elements of both black-box & white-box frameworks, e.g.
  - Its hook methods are elements of a white-box framework

Black-box



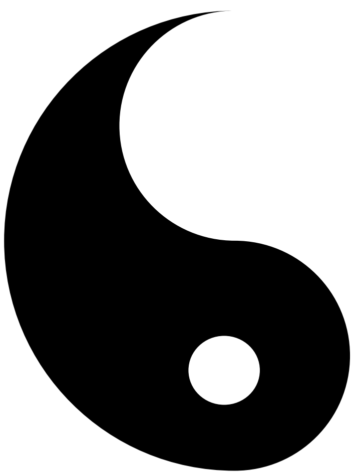
White-box

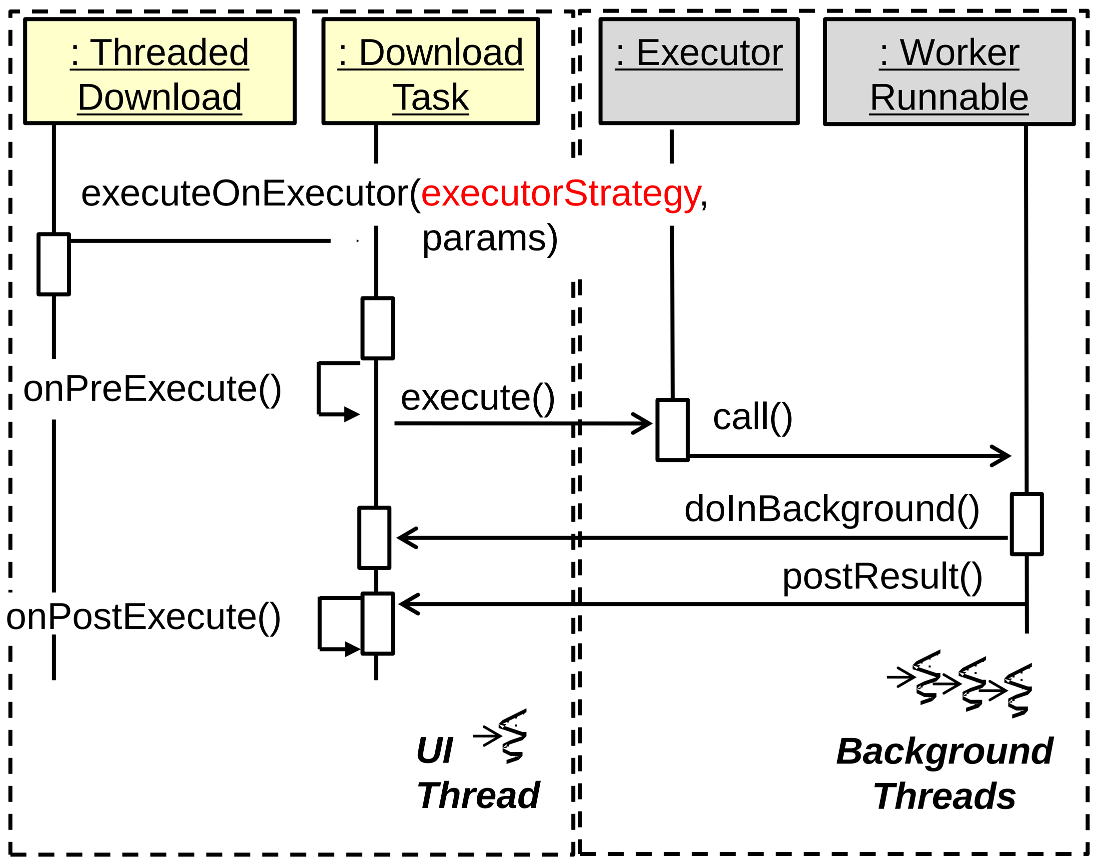


AsyncTask Usage Considerations

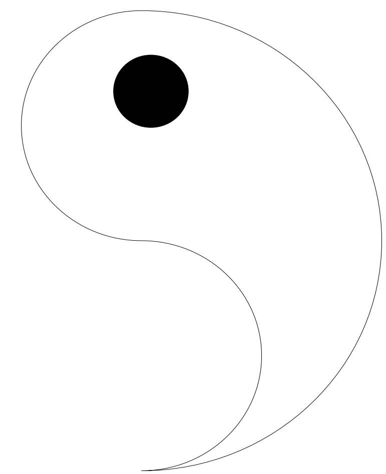
- AsyncTask has elements of both black-box & white-box frameworks, e.g.
  - Its hook methods are elements of a white-box framework
  - Its executor strategy is an element of a black-box framework

Black-box





White-box



AsyncTask Usage Considerations

- There are trade-offs between each approach
  - White-box frameworks are generally easier to develop…

AsyncTask Usage Considerations

- There are trade-offs between each approach
  - White-box frameworks are generally easier to develop…
    - … but harder to use

AsyncTask Usage Considerations

- There are trade-offs between each approach
  - White-box frameworks are generally easier to develop…
    - … but harder to use
  - Black-box frameworks are generally harder to develop…

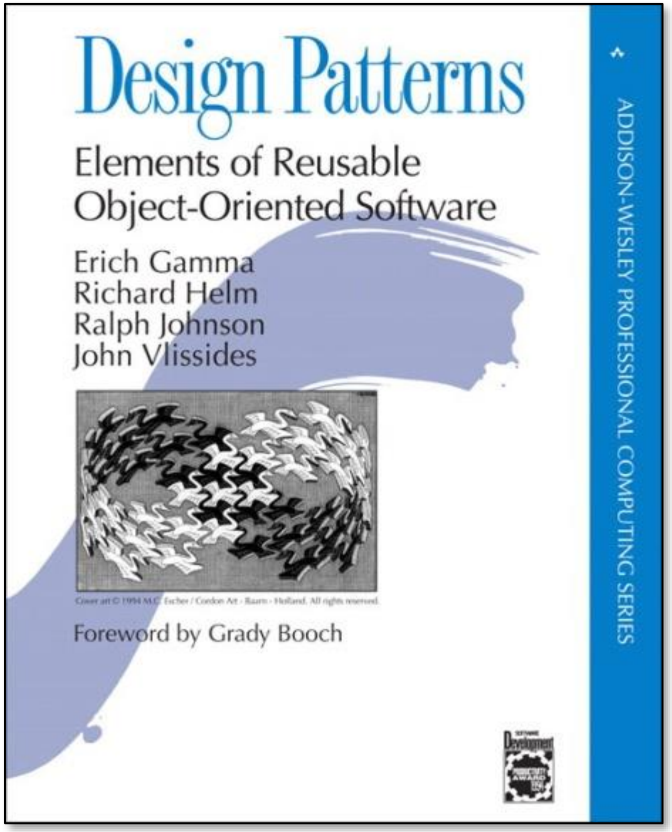
AsyncTask Usage Considerations

- There are trade-offs between each approach
  - White-box frameworks are generally easier to develop…
    - … but harder to use
  - Black-box frameworks are generally harder to develop…
    - … but easier to use

See en.wikipedia.org/wiki/Plug-in_(computing)

AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns



AsyncTask Usage Considerations

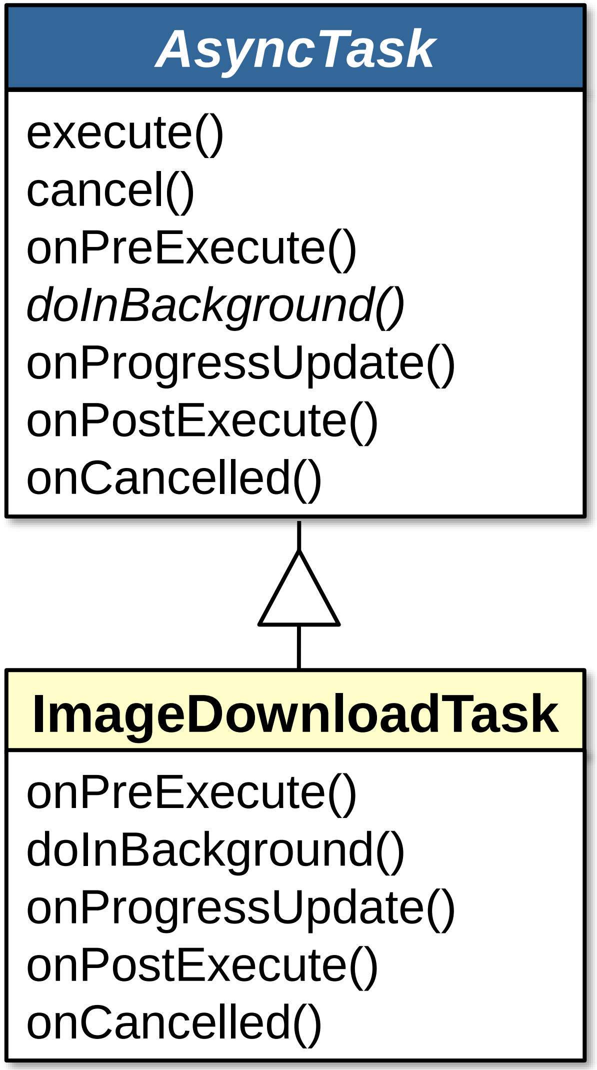
- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities

AsyncTask
execute()
cancel()
onPreExecute()
doInBackground()
onProgressUpdate()
onPostExecute()
onCancelled()

See en.wikipedia.org/wiki/Template_method_pattern

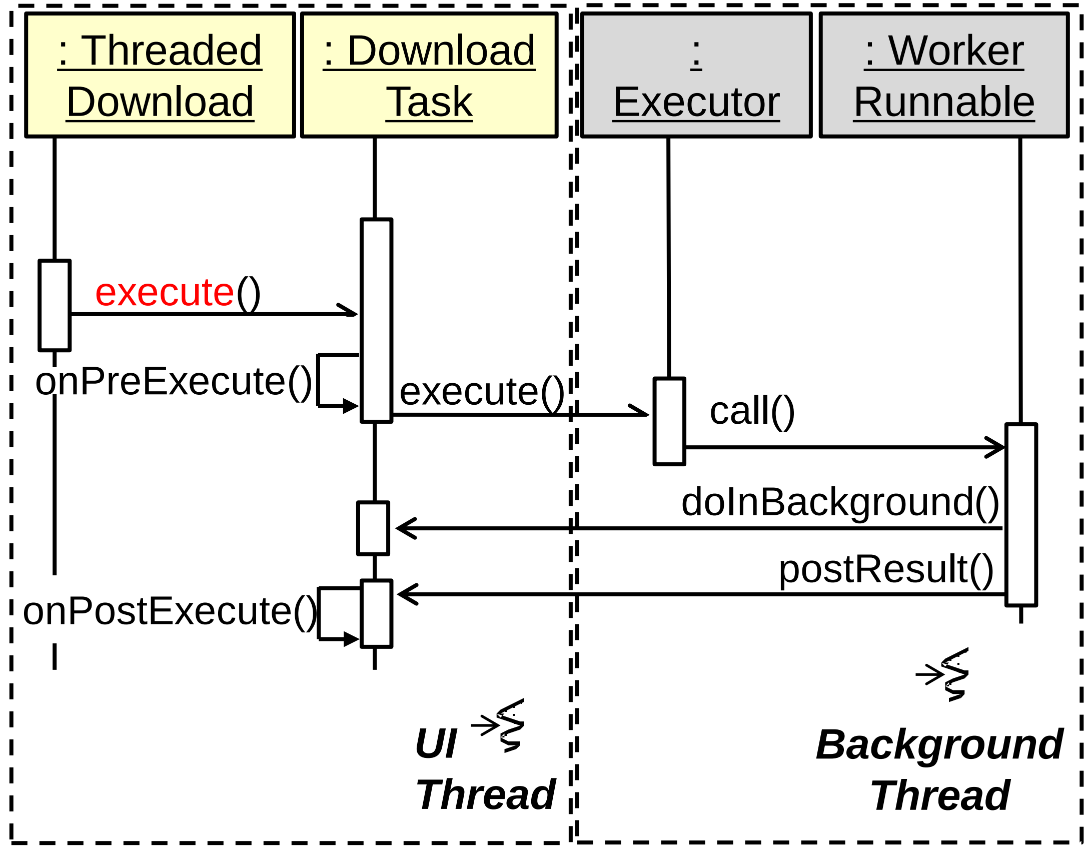
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities



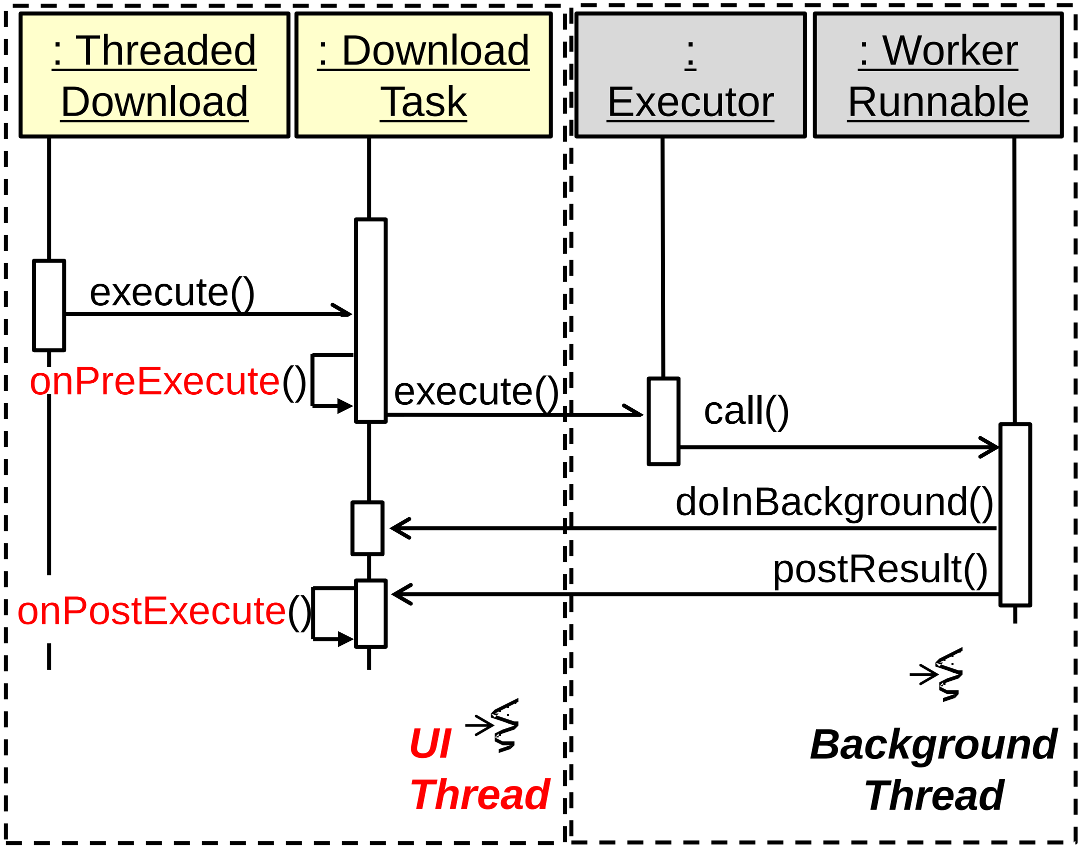
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities



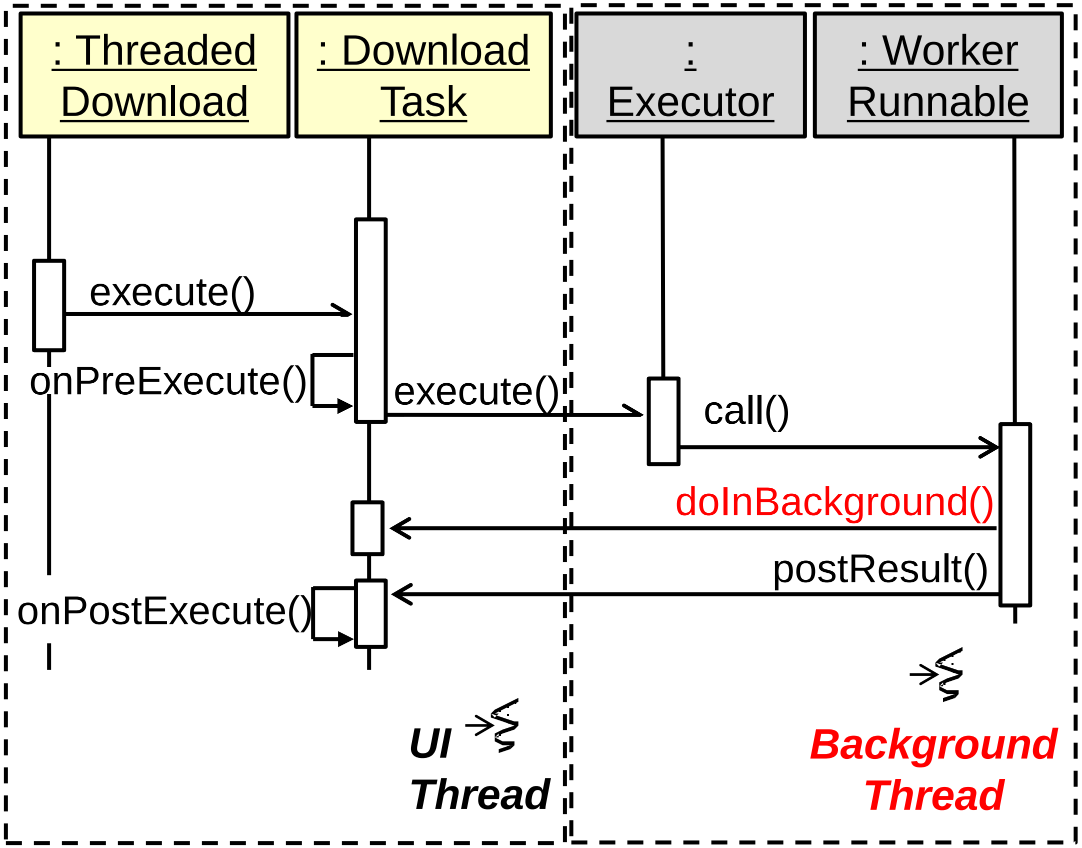
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities



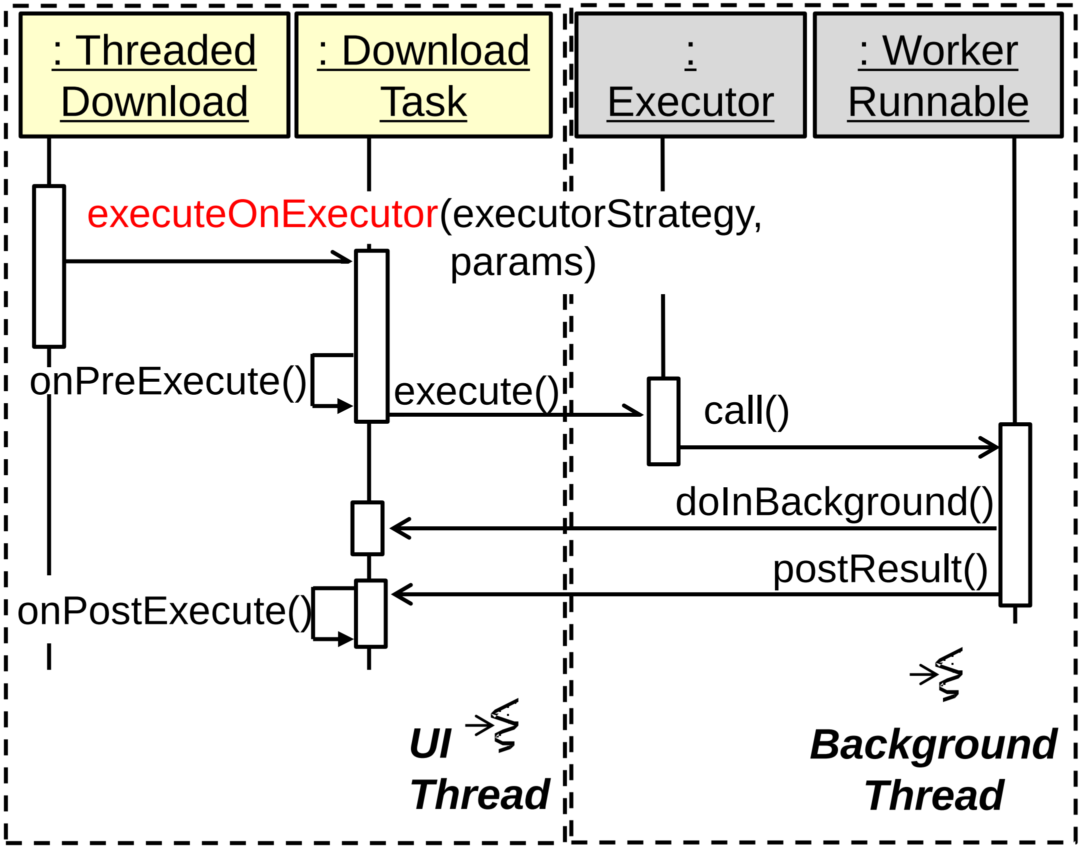
AsyncTask Usage Considerations

- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities



AsyncTask Usage Considerations

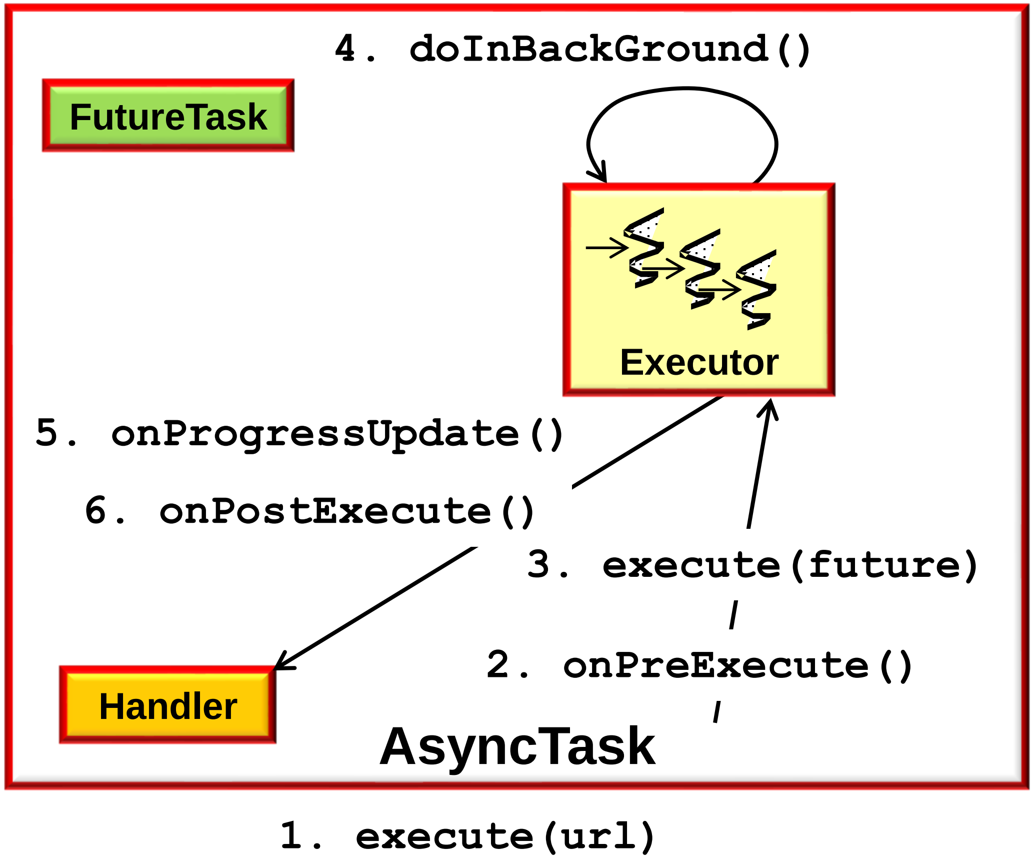
- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities
  - *Strategy* is used for its black-box capabilities



See en.wikipedia.org/wiki/Strategy_pattern

AsyncTask Usage Considerations

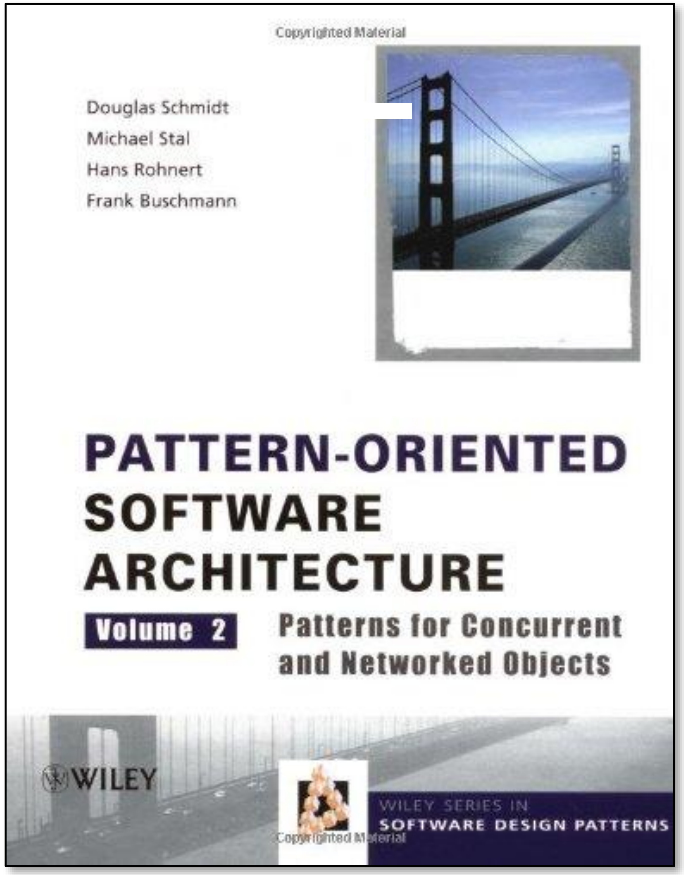
- AsyncTask applies several GoF patterns
  - *Template Method* is used for its white-box capabilities
  - *Strategy* is used for its black-box capabilities
  - *Façade* is used to simplify access to the Java Executor framework



See en.wikipedia.org/wiki/Facade_pattern

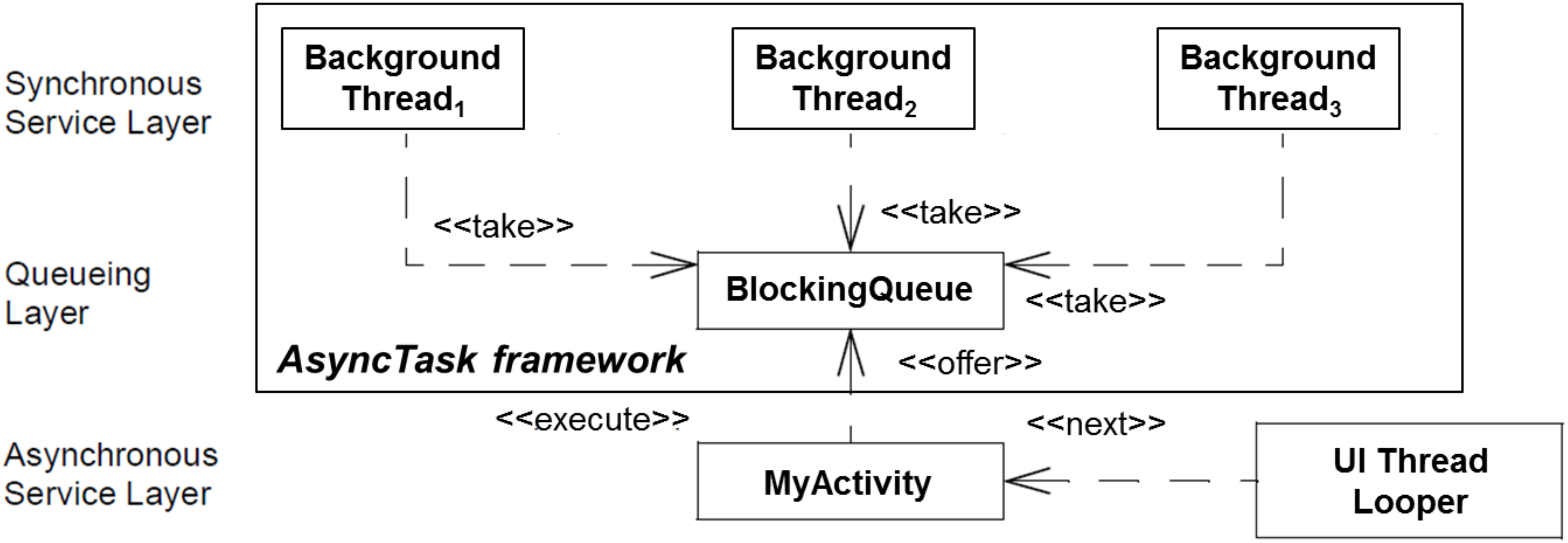
AsyncTask Usage Considerations

- AsyncTask also applies several POSA patterns



AsyncTask Usage Considerations

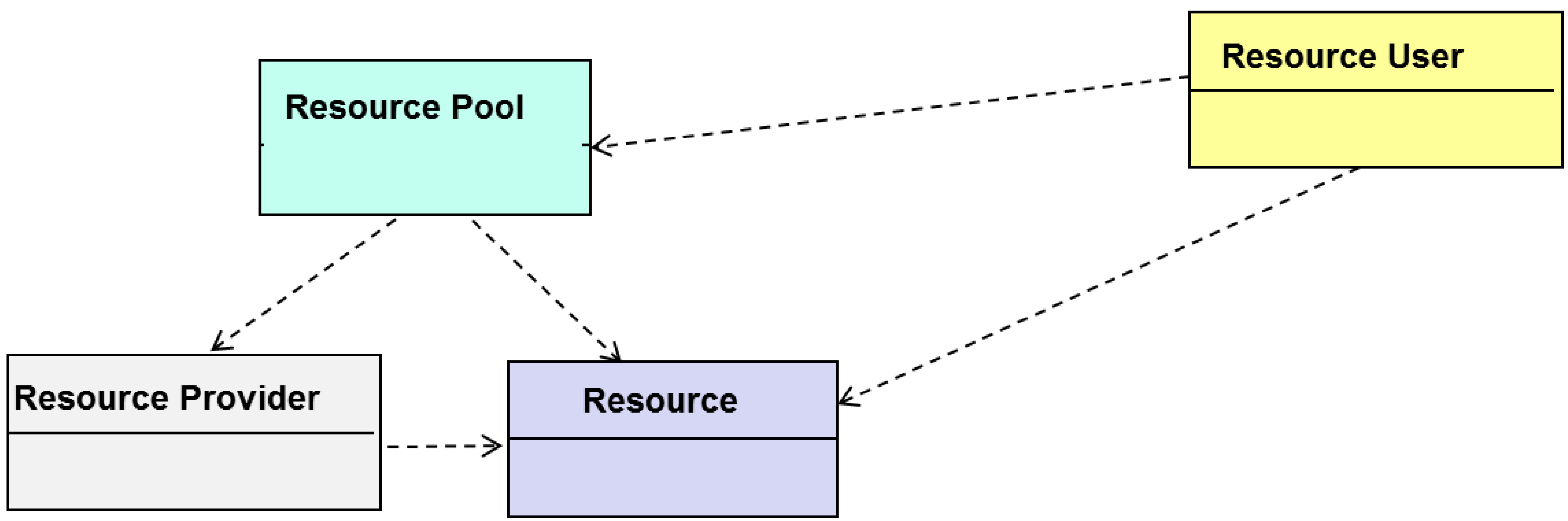
- AsyncTask also applies several POSA patterns
  - *Half-Sync/Half-Async* is used to coordinate between the UI thread & background thread(s)



See www.dre.vanderbilt.edu/~schmidt/PDF/HS-HA.pdf

AsyncTask Usage Considerations

- AsyncTask also applies several POSA patterns
  - *Half-Sync/Half-Async* is used to coordinate between the UI thread & background thread(s)
  - *Pooling* is used to manage multiple instances of threads, which allows for reuse when AsyncTasks release threads they no longer need



See www.kircher-schwanninger.de/michael/publications/Pooling.pdf

AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls

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

AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
  - Cancellation
    - Cancellation is voluntary, just like Thread.interrupt()

See www.technotalkative.com/cancel-asynctask-in-android

AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
    - Memory leaks occur if there’s a strong references to enclosing Activity

See medium.com/@zhangqichuan/memory-leak-in-android-4a6a7e8d7780

AsyncTask Usage Considerations

- 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.

See commonsware.com/blog/2010/09/10/asynctask-screen-rotation.html

AsyncTask Usage Considerations

- 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):

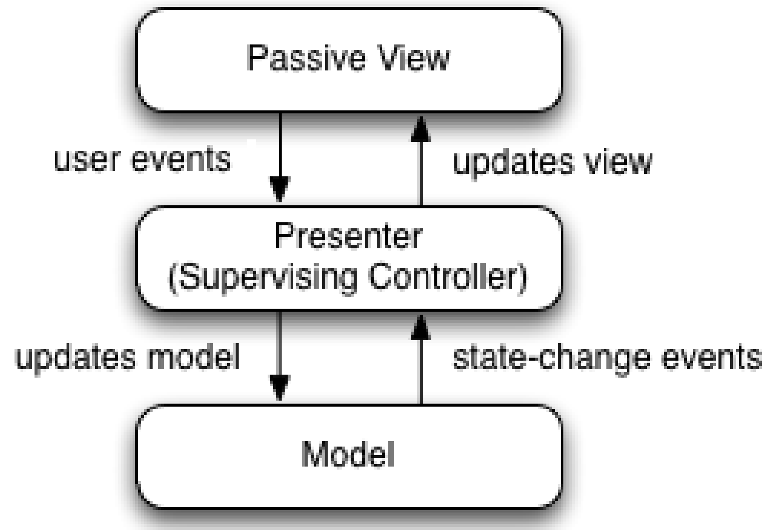
- The Android developers team decided to change this so that AsyncTasks could run parallel on a separate worker thread. There was one problem. Many developers relied on the sequential behavior and suddenly they were having a lot of concurrency issues.

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 has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
  - Portability



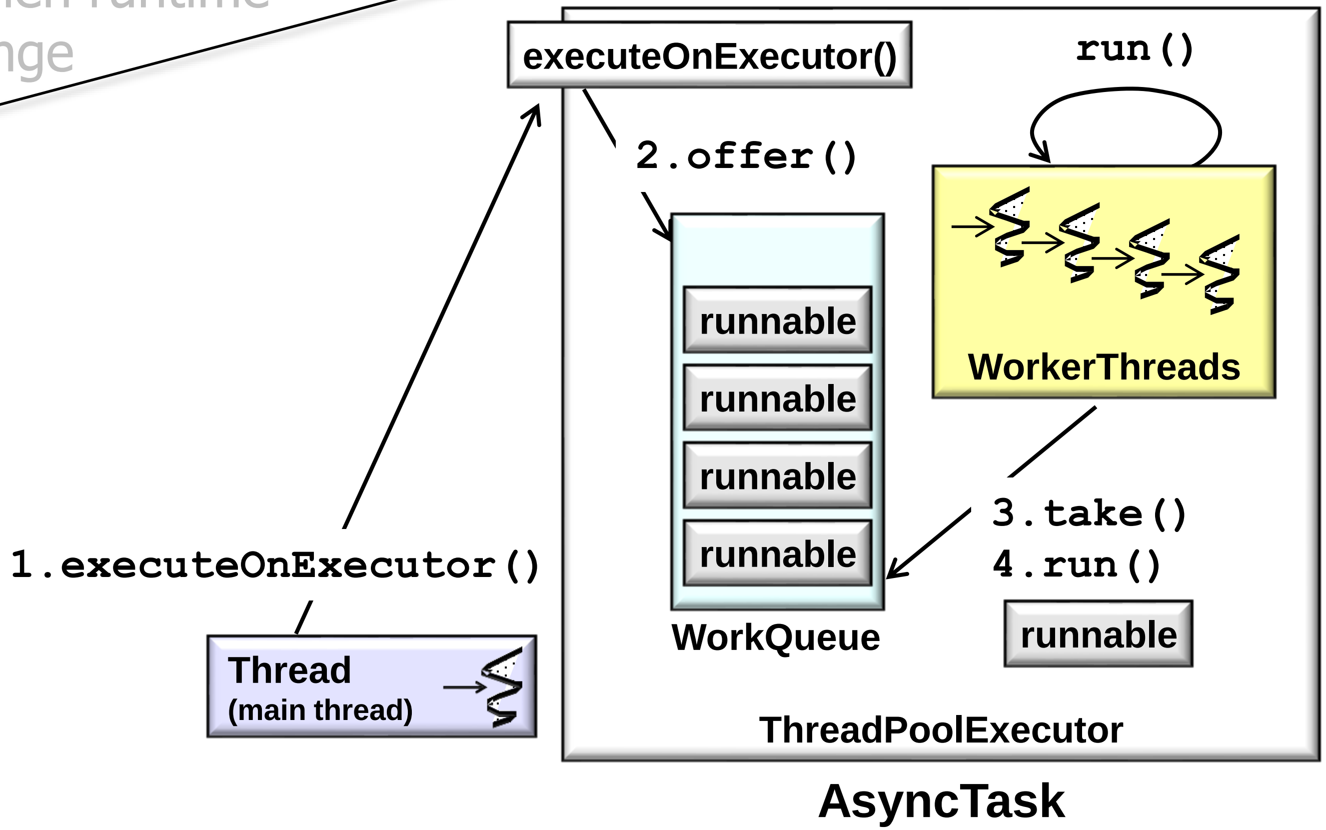
The Model-View Presenter (MVP) pattern addresses some of these issues

See en.wikipedia.org/wiki/Model-view-presenter

AsyncTask Usage Considerations

- AsyncTask has traps & pitfalls
  - Cancellation
  - Dependency on Activity
  - Losing results if/when runtime configurations change
  - Portability

Other issues can be addressed only by understanding Android patterns & APIs



See developer.android.com/training/articles/perf-anr.html#Avoiding

End of the AsyncTask Framework: Usage Considerations