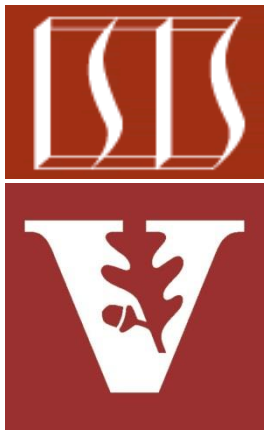
The AsyncTask Framework: Structure & Functionality



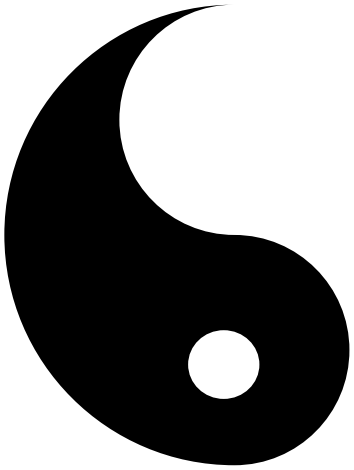
Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

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

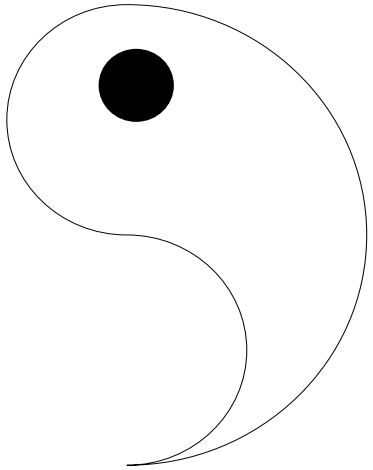
Learning Objectives in this Part of the Lesson

- 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…

Black-box



White-box

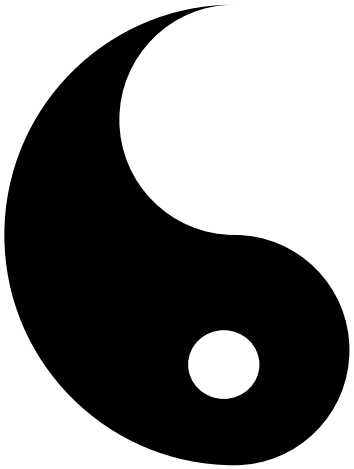


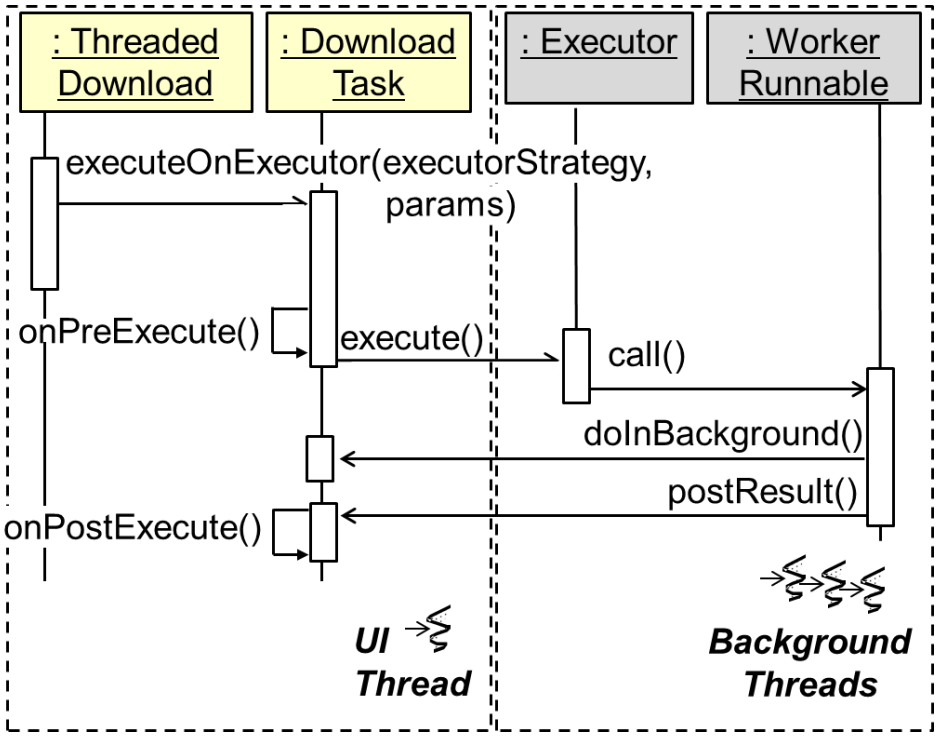
See www.dre.vanderbilt.edu/~schmidt/PDF/DRC.pdf

Learning Objectives in this Part of the Lesson

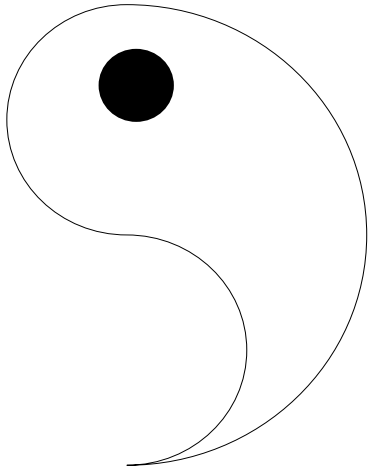
- Recognize the capabilities provided by the Android AsyncTask framework
- Know which methods are provided by AsyncTask class
- Understand what black-box & white-box framework are… & how AsyncTask implements both types of frameworks

Black-box





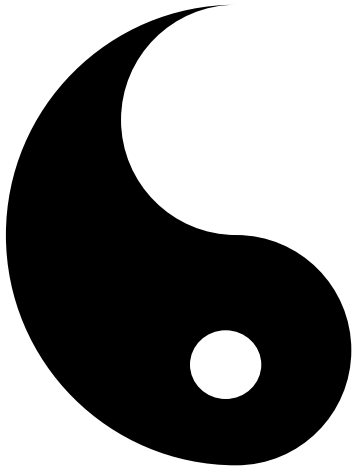
White-box



Common Types of Frameworks

Common Types of Frameworks

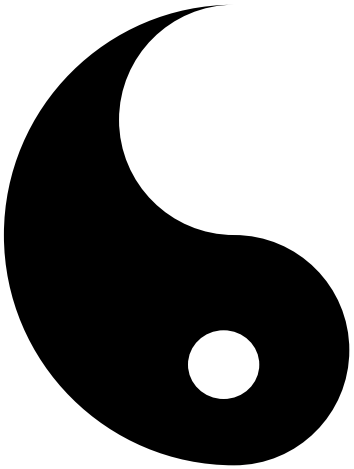
- **Black-box frameworks** only require understanding external interfaces of objects



See wiki.c2.com/?BlackBoxFramework

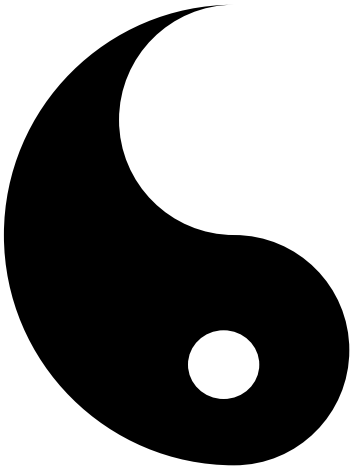
Common Types of Frameworks

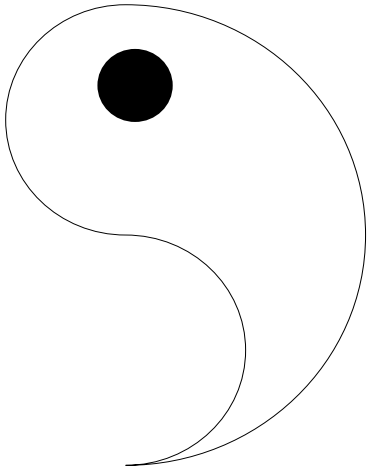
- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects



Common Types of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects
- **White-box frameworks** require understanding some parts of the framework implementation



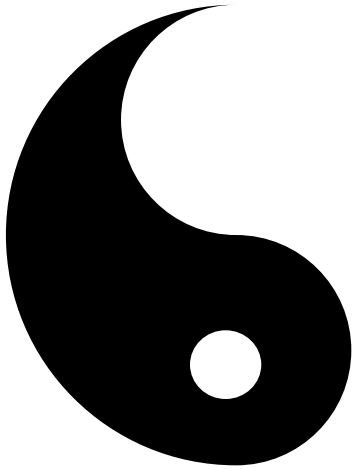


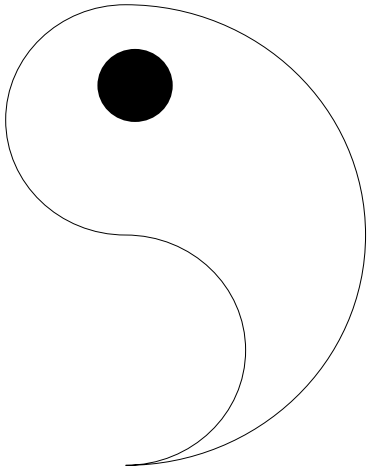
See wiki.c2.com/?WhiteBoxFramework

Common Types of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
  - Framework elements typically reused by parameterizing & assembling objects

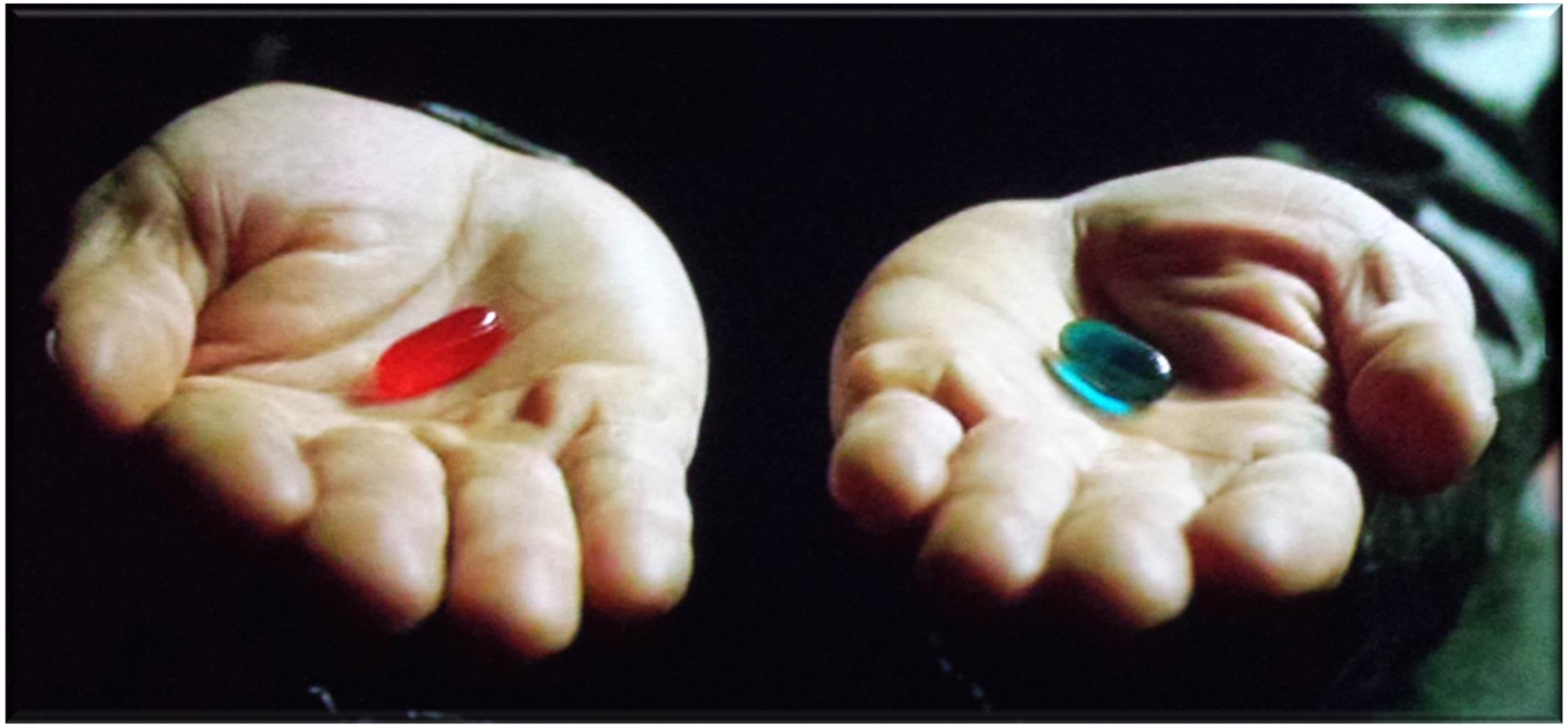
- **White-box frameworks** require understanding some parts of the framework implementation
  - Framework elements typically reused by subclassing & overridding





Common Types of Frameworks

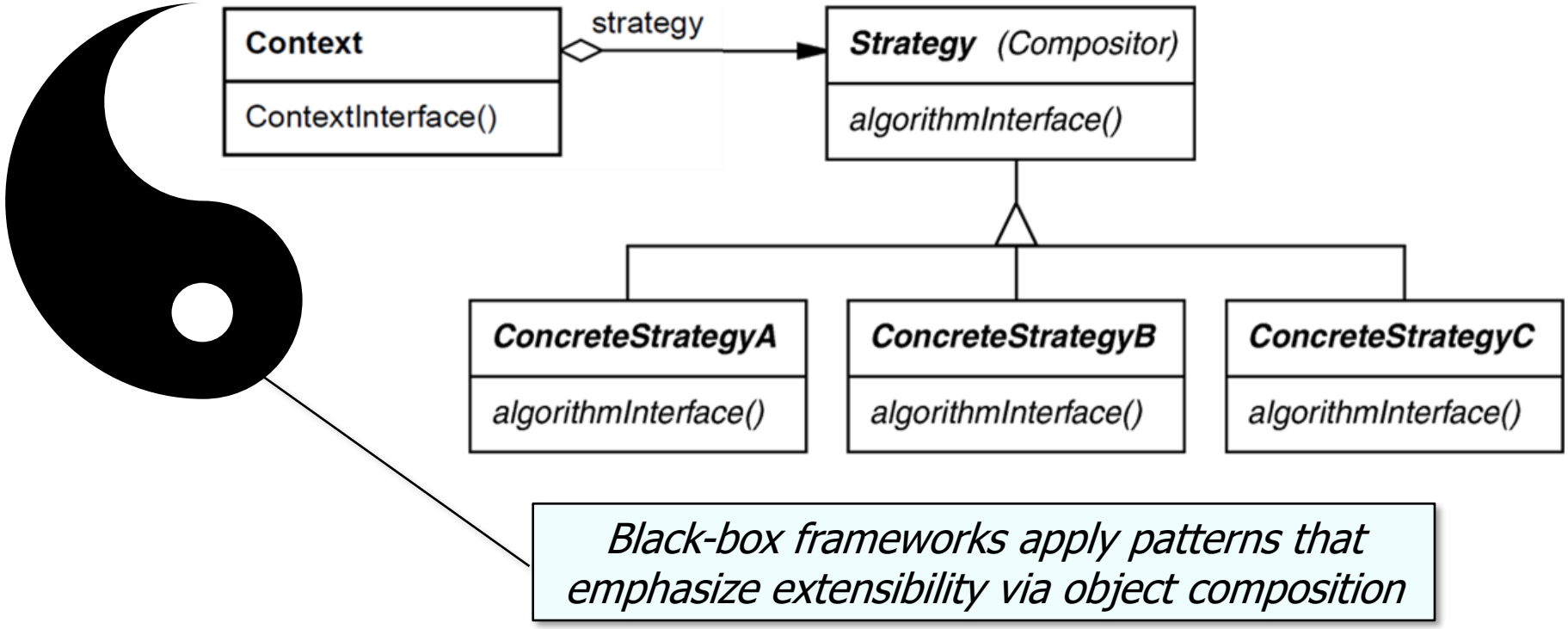
- **Black-box frameworks** only require understanding external interfaces of objects
- **White-box frameworks** require understanding some parts of the framework implementation
- Each category of OO framework uses different sets of patterns



See www.dre.vanderbilt.edu/~schmidt/PDF/DRC.pdf

Common Types of Frameworks

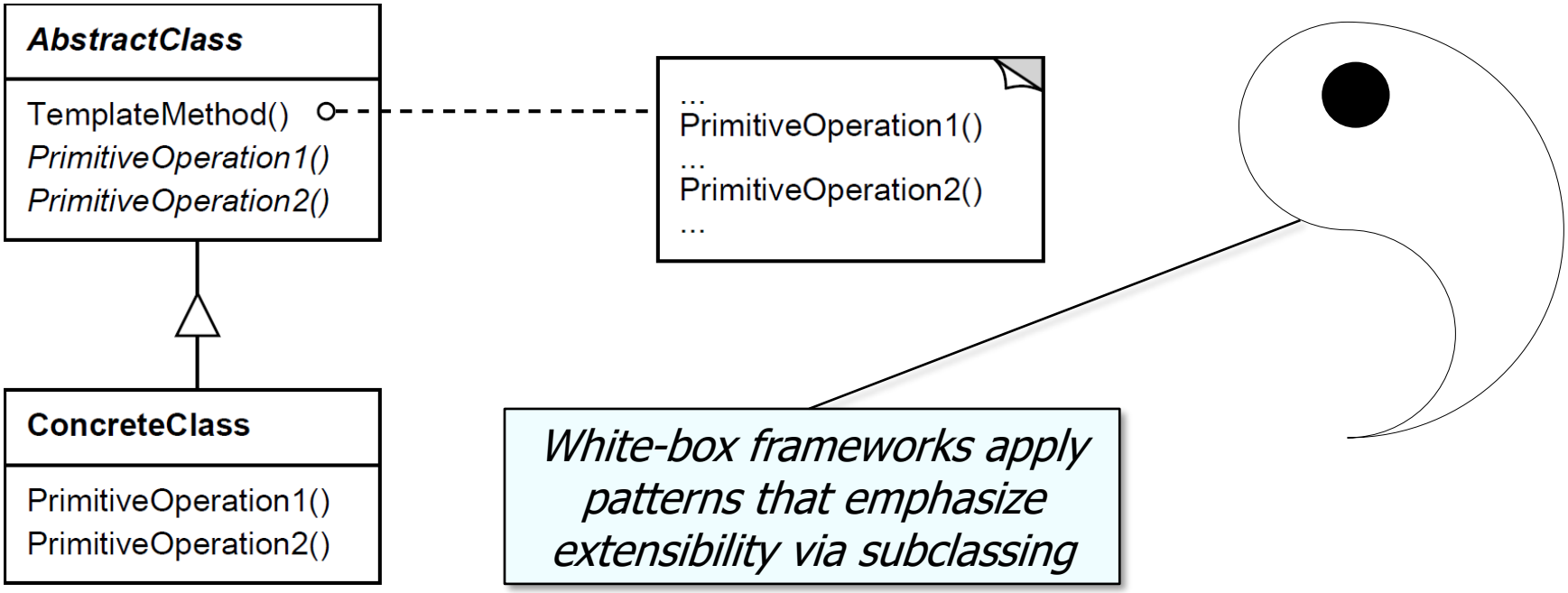
- **Black-box frameworks** only require understanding external interfaces of objects
- **White-box frameworks** require understanding some parts of the framework implementation
- Each category of OO framework uses different sets of patterns



See en.wikipedia.org/wiki/Strategy_pattern & en.wikipedia.org/wiki/Decorator_pattern

Common Types of Frameworks

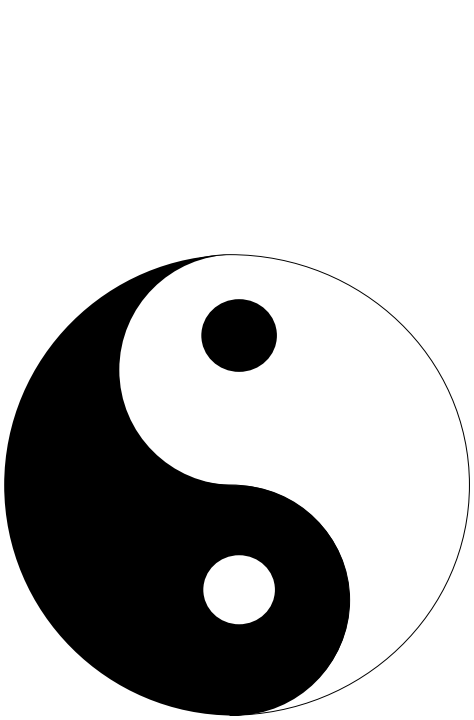
- **Black-box frameworks** only require understanding external interfaces of objects
- **White-box frameworks** require understanding some parts of the framework implementation
- Each category of OO framework uses different sets of patterns

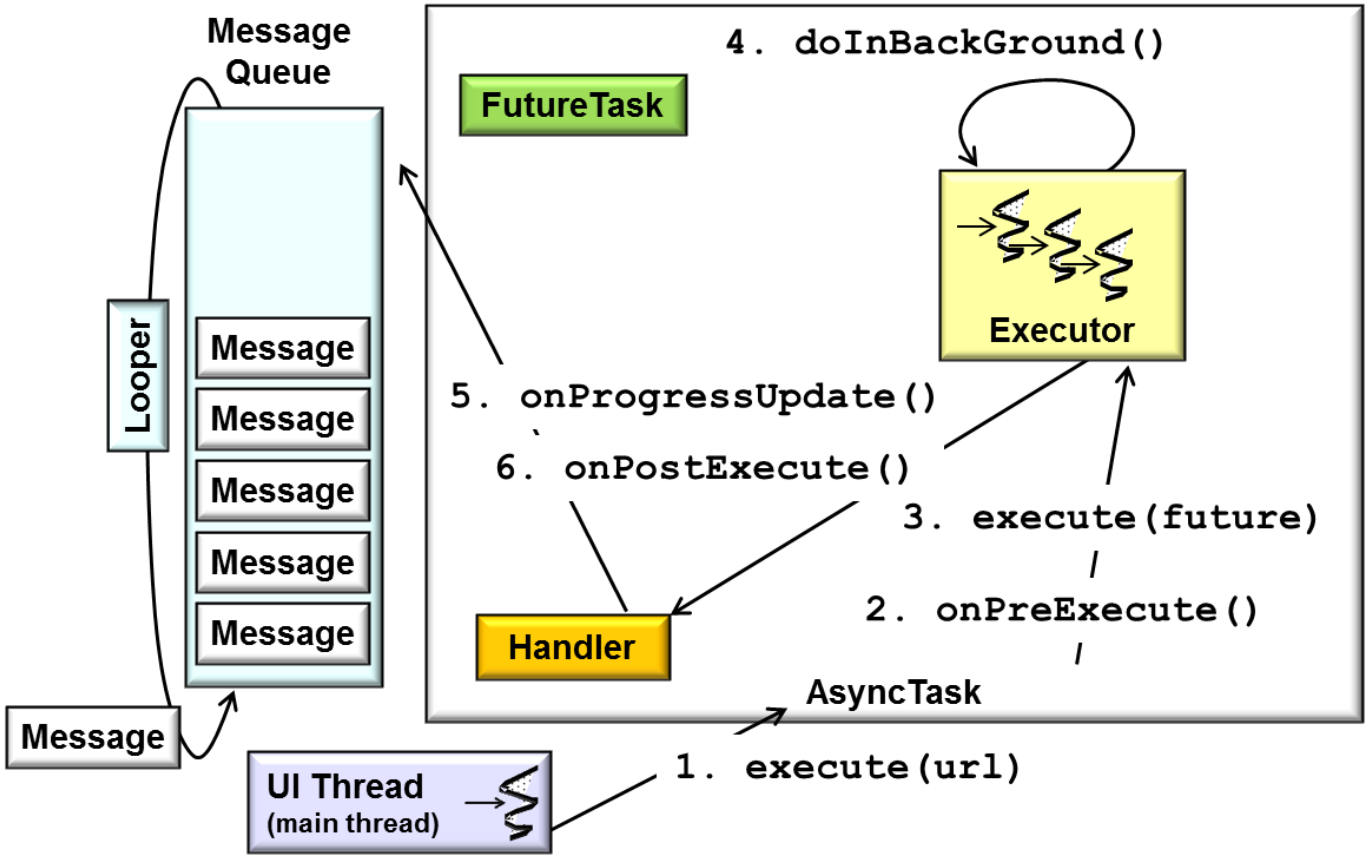


See en.wikipedia.org/wiki/Template_method & en.wikipedia.org/wiki/State_pattern

Common Types of Frameworks

- **Black-box frameworks** only require understanding external interfaces of objects
- **White-box frameworks** require understanding some parts of the framework implementation
- Each category of OO framework uses different sets of patterns



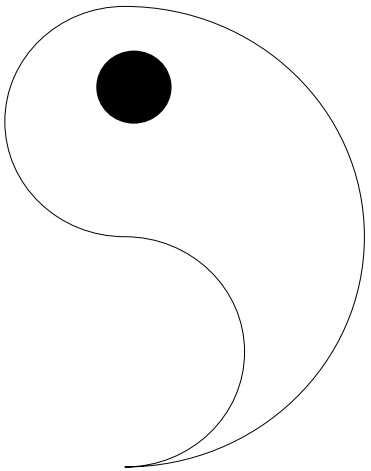


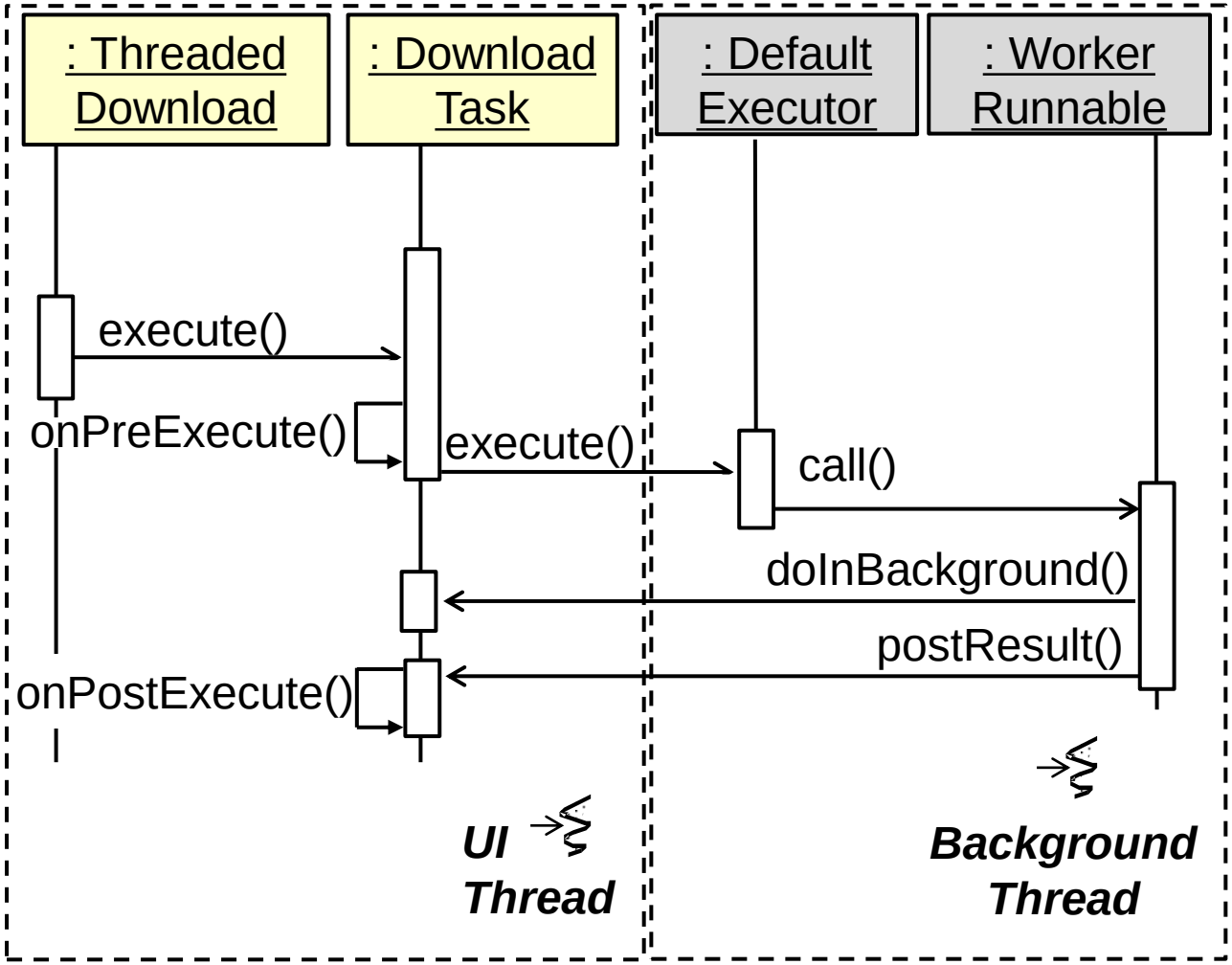
Android's AsyncTask framework embodies both black-box & white-box frameworks

White-box Elements of the AsyncTask Framework

White-box Elements of the AsyncTask Framework

- White-box framework elements enable long duration operations to interact with UI thread

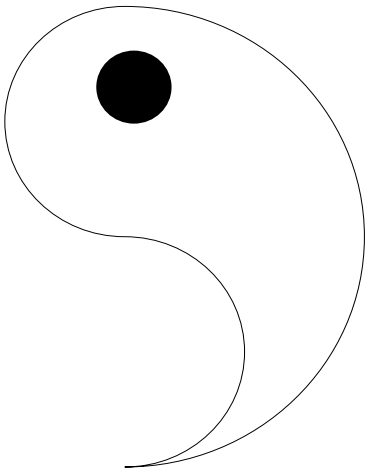


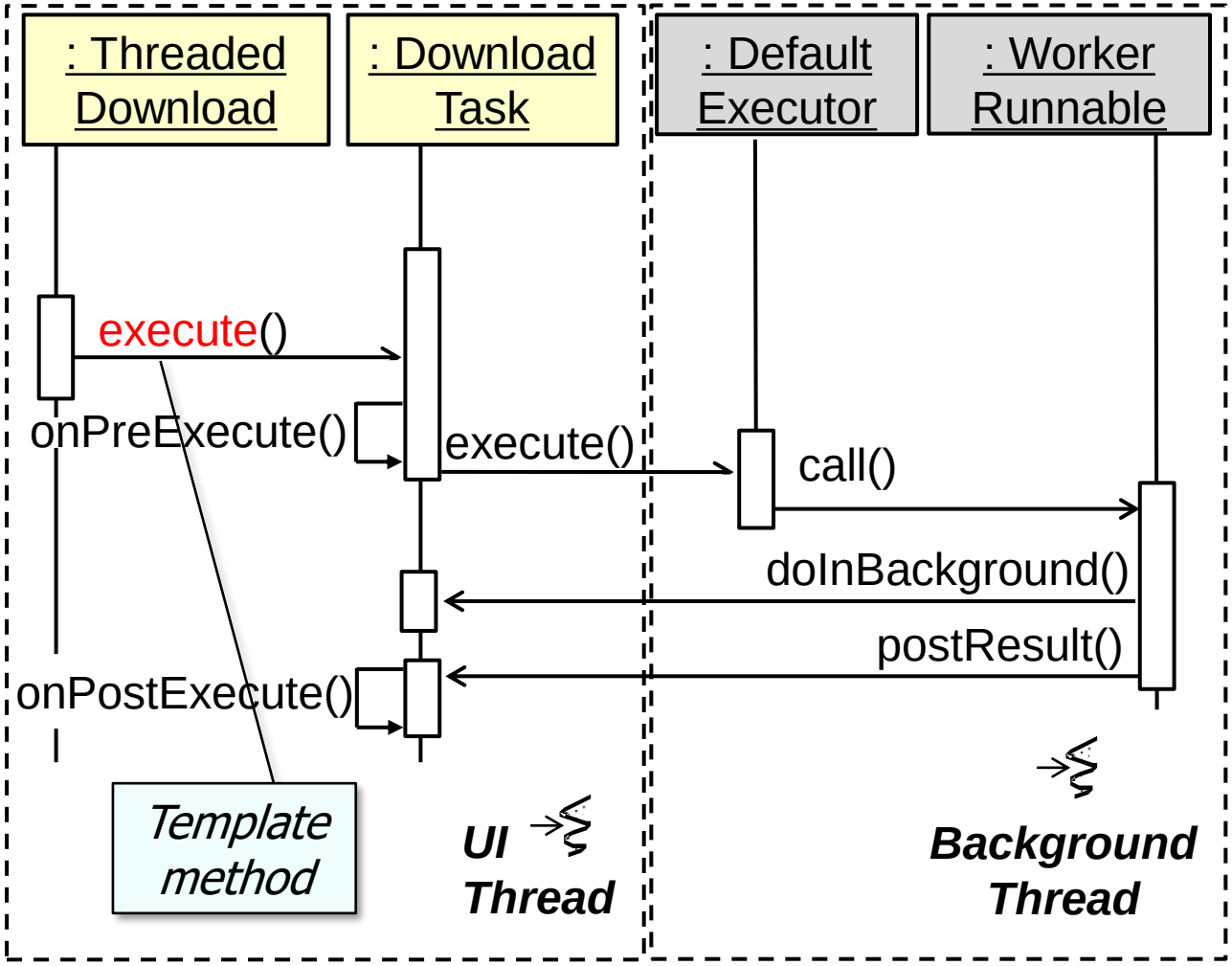


See en.wikipedia.org/wiki/Template_method

White-box Elements of the AsyncTask Framework

- White-box framework elements enable long duration operations to interact with UI thread

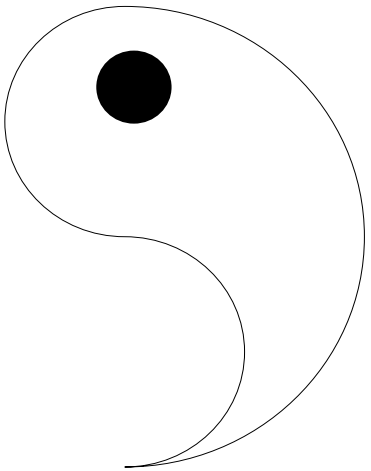


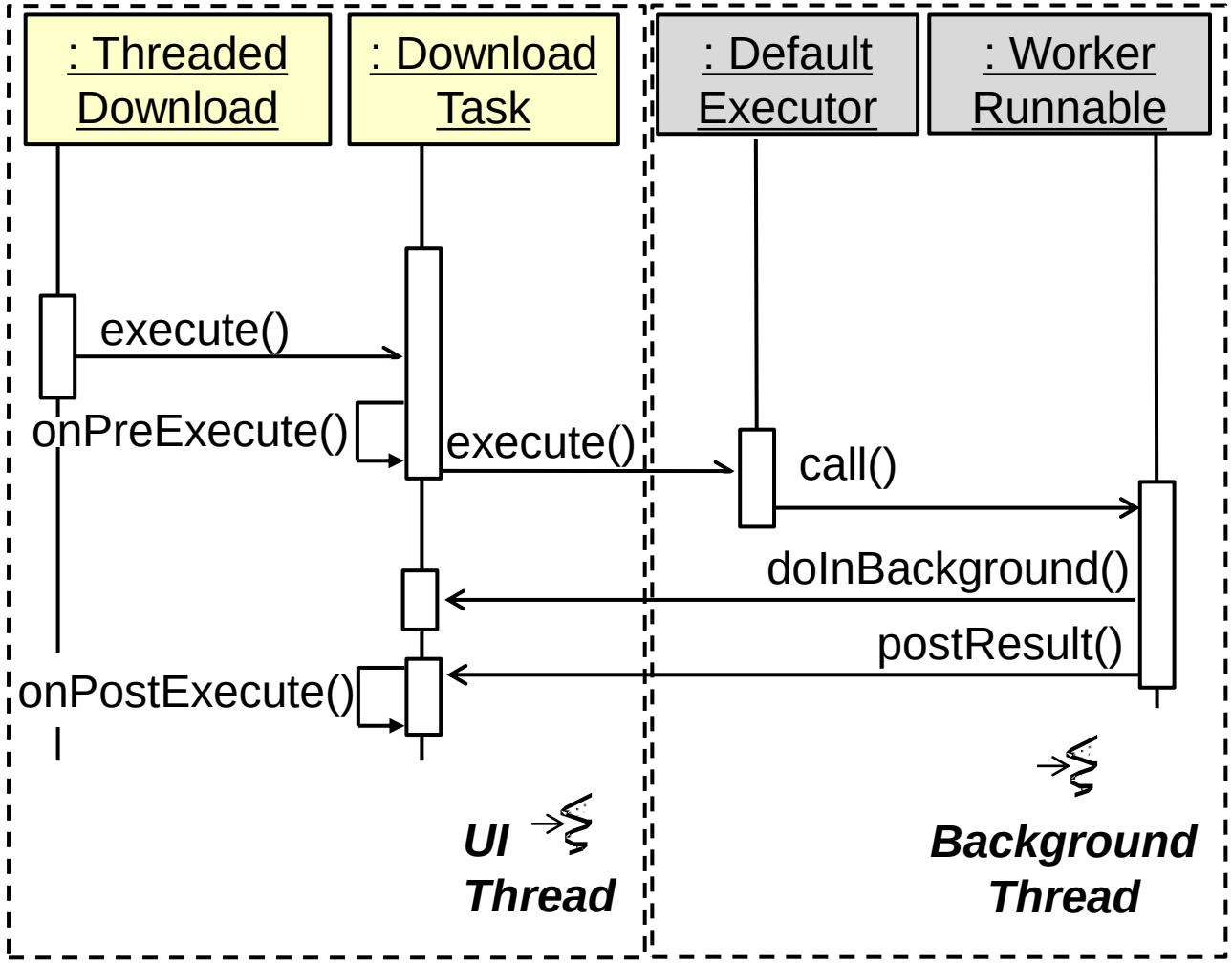


See xp123.com/wwake/fw/ch12-bb.htm for more on framework design

White-box Elements of the AsyncTask Framework

- Framework dictates control flow via hook method callbacks

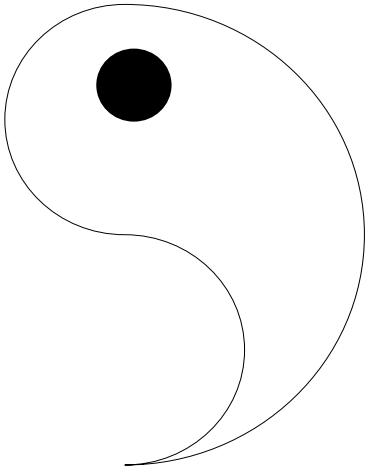


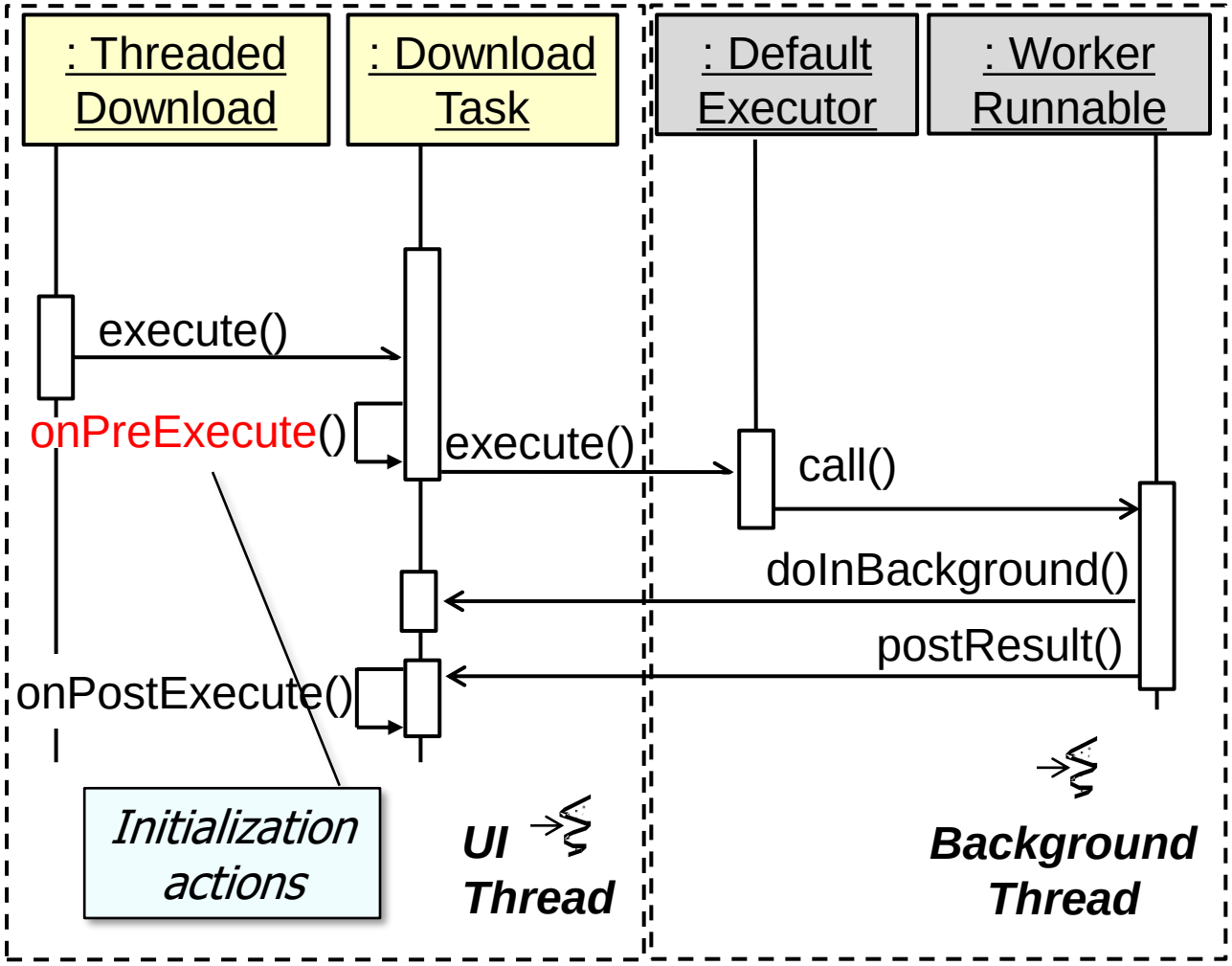


See en.wikipedia.org/wiki/Callback_(computer_programming)

White-box Elements of the AsyncTask Framework

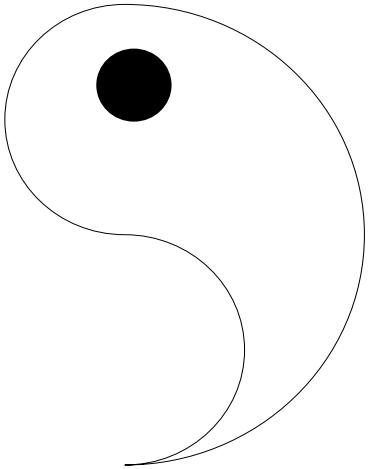
- Framework dictates control flow via hook method callbacks

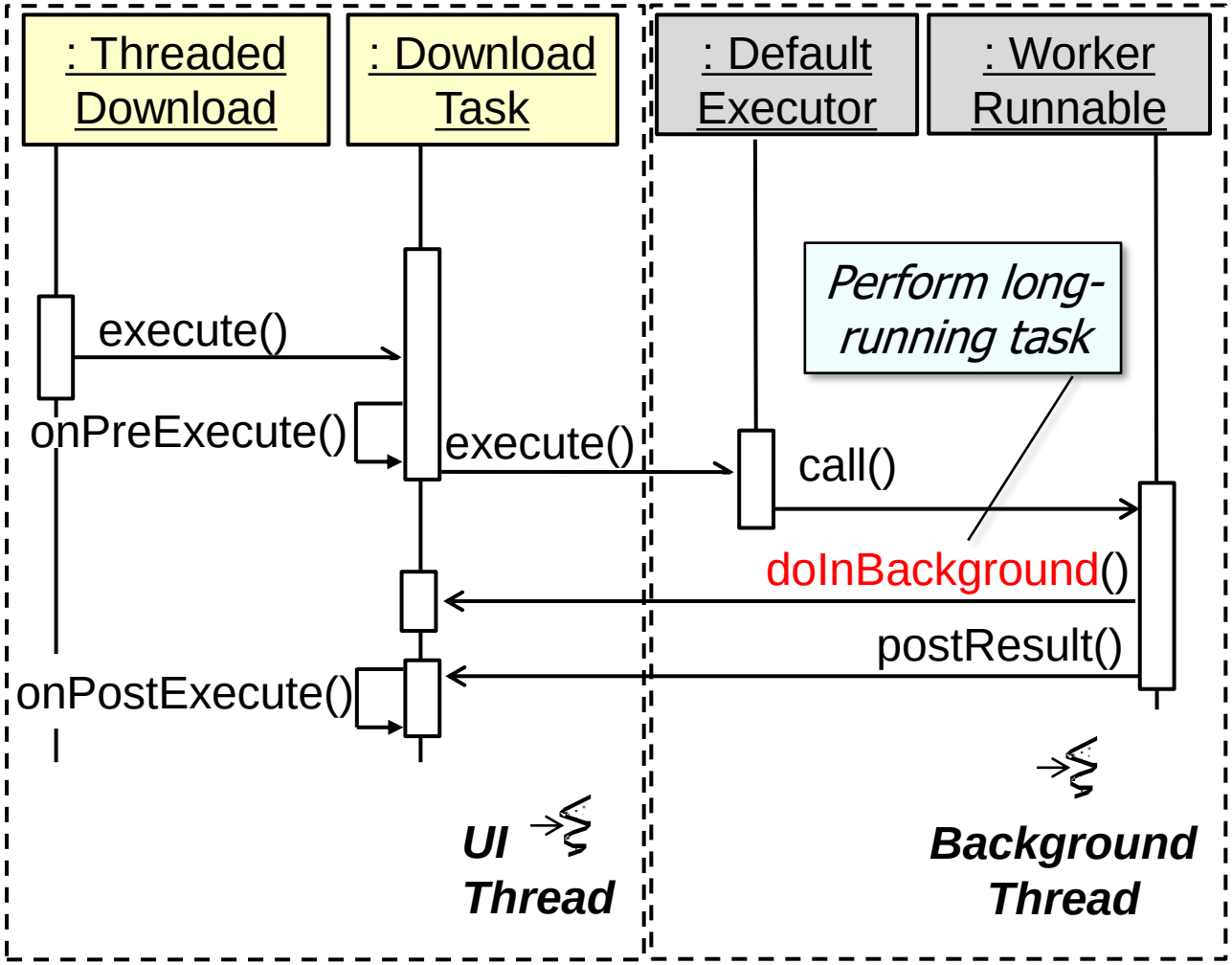




White-box Elements of the AsyncTask Framework

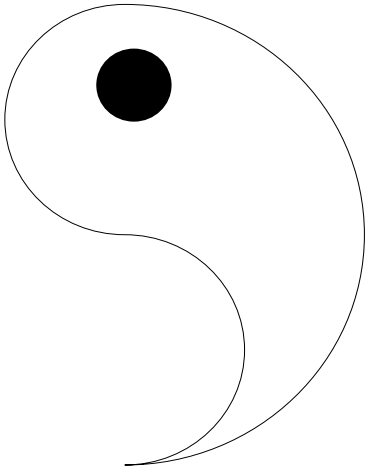
- Framework dictates control flow via hook method callbacks

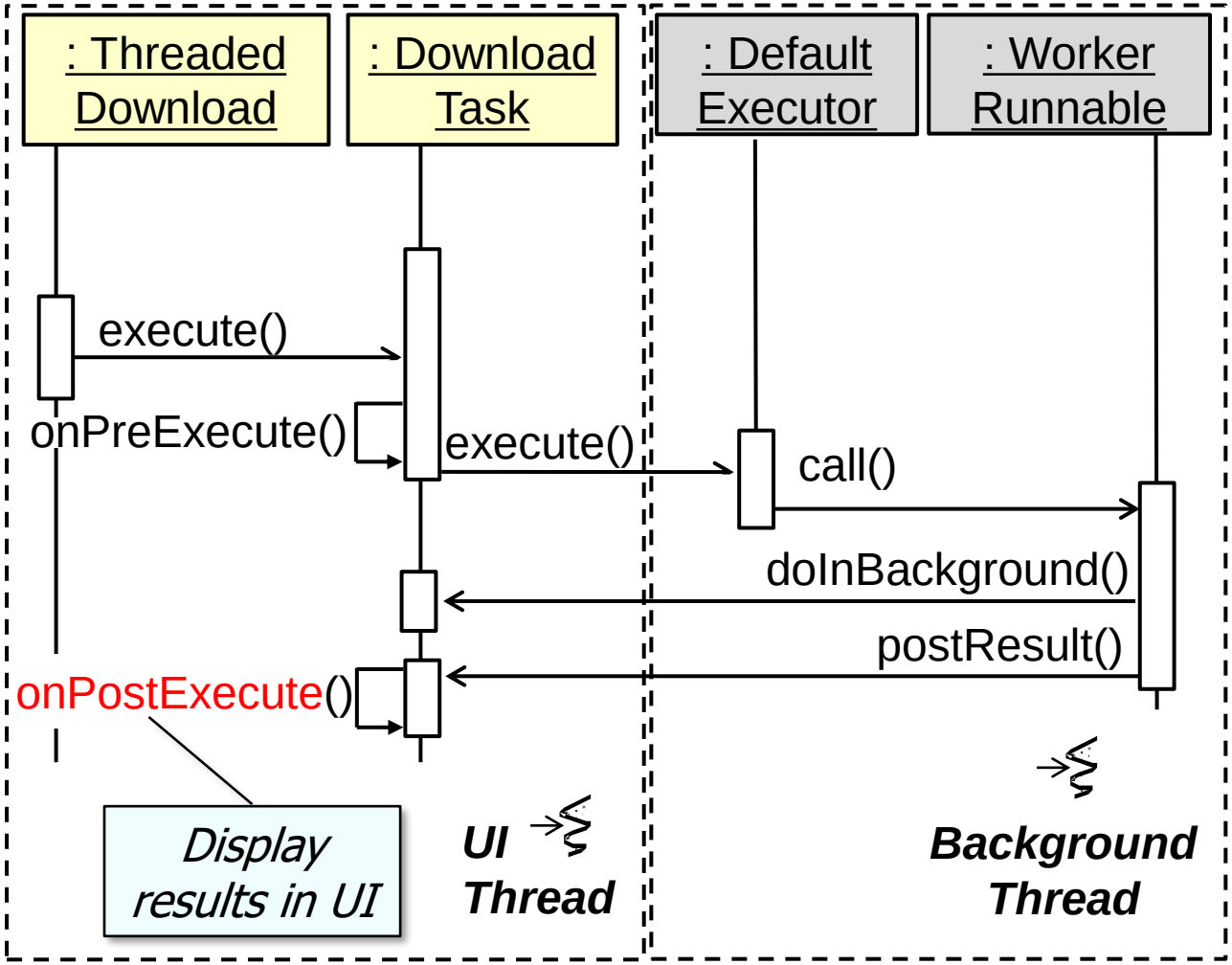




White-box Elements of the AsyncTask Framework

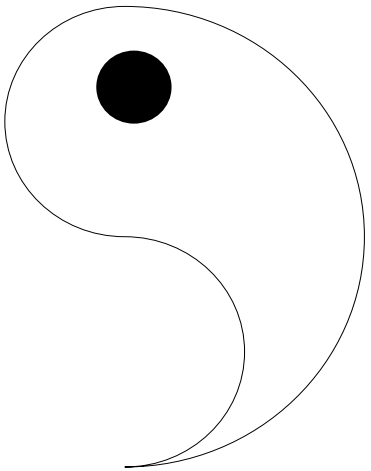
- Framework dictates control flow via hook method callbacks

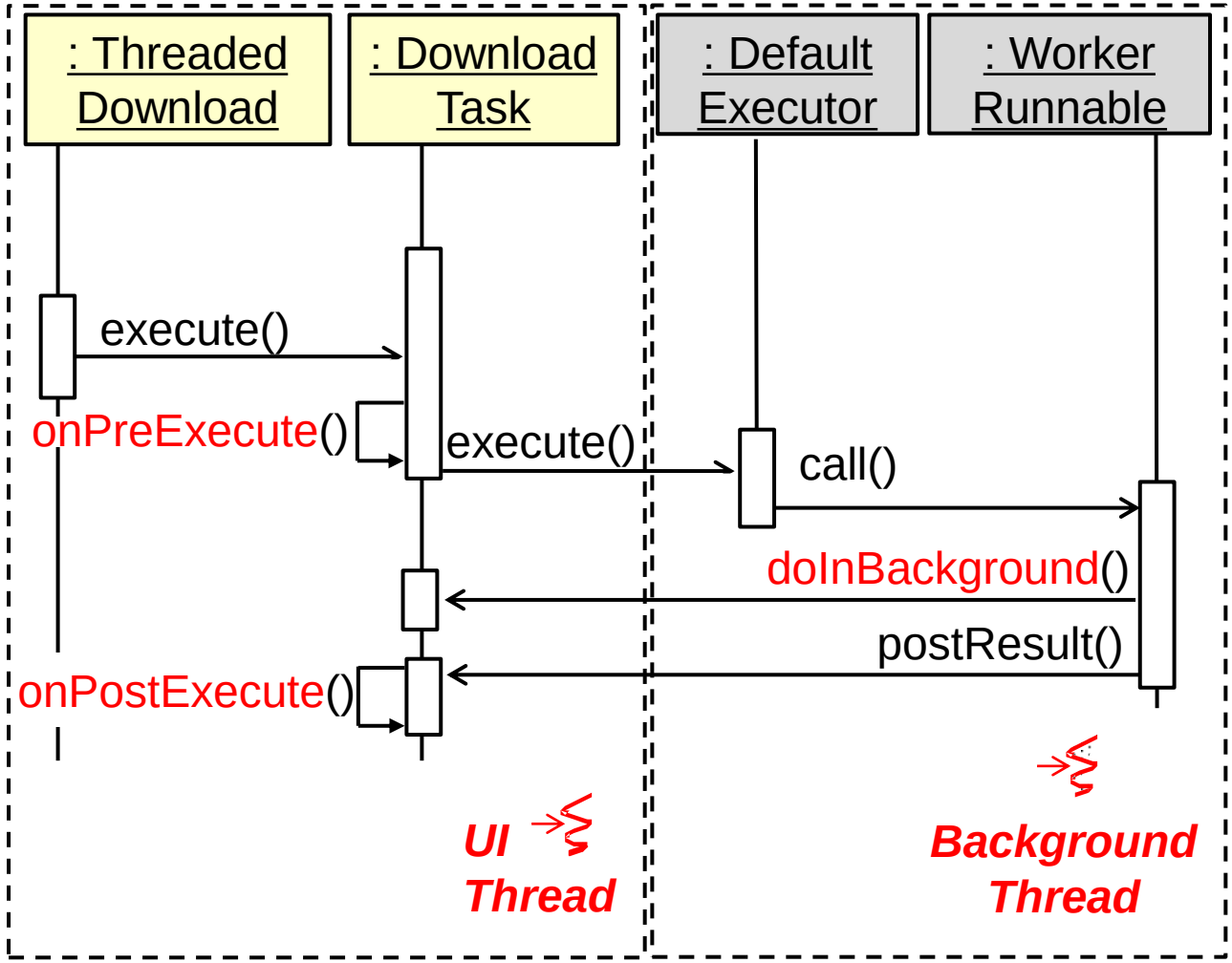




White-box Elements of the AsyncTask Framework

- Framework dictates control flow via hook method callbacks



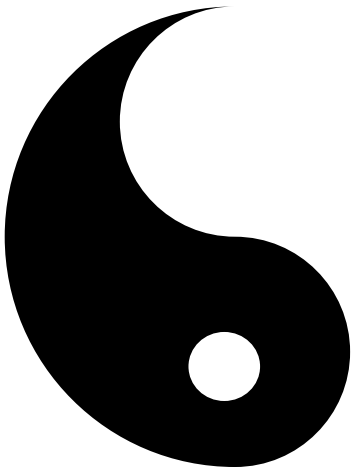


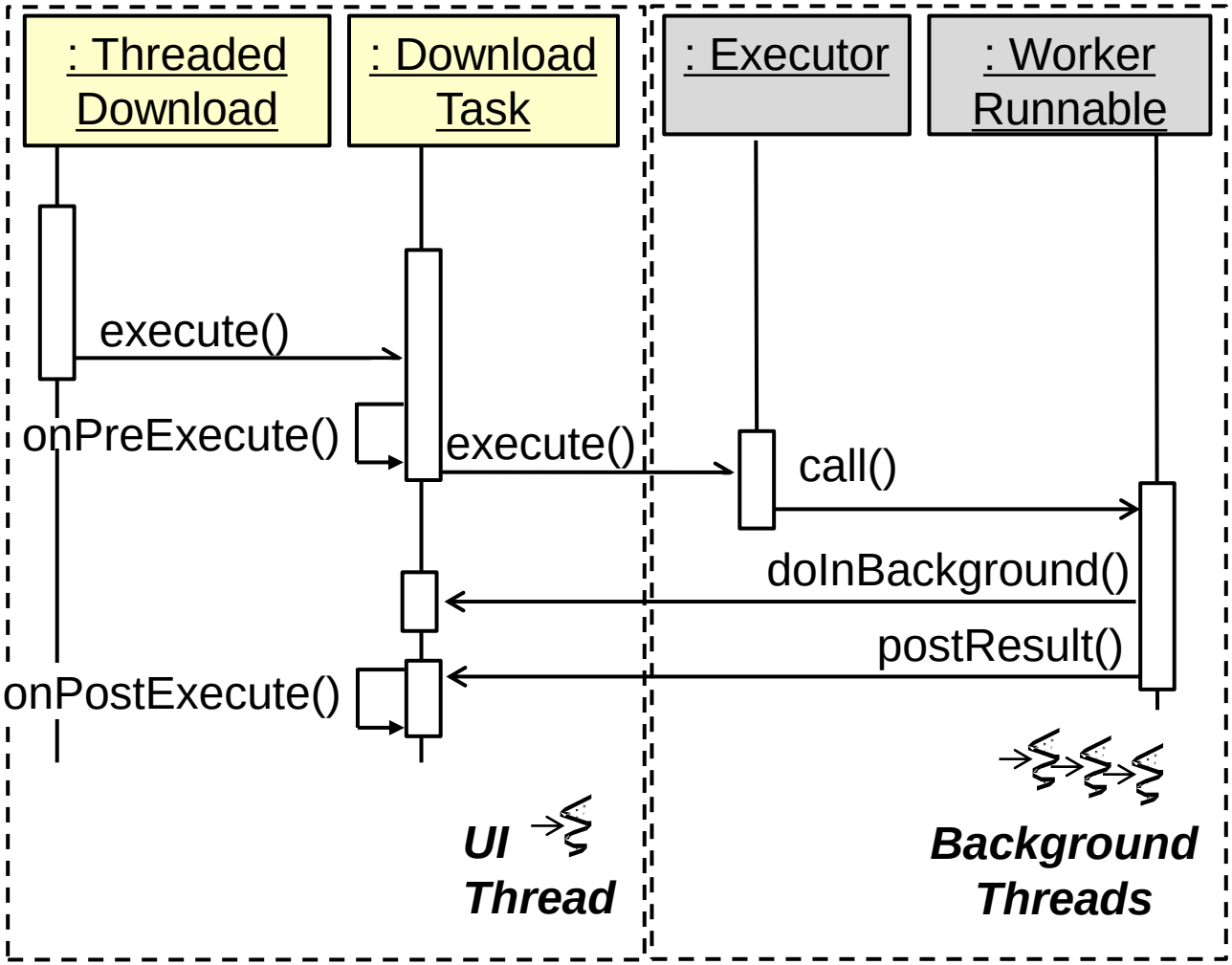
This *Template Method* variant allows hook methods to run in different threads

Black-box Elements of the AsyncTask Framework

Black-box Elements of the AsyncTask Framework

- Black-box framework elements control the background thread(s)

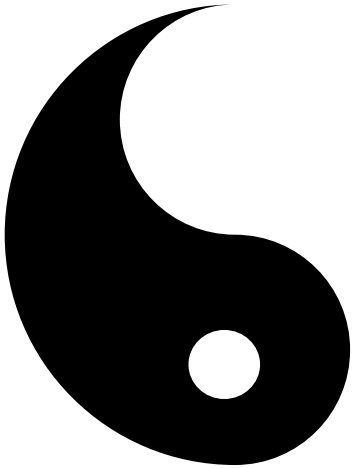


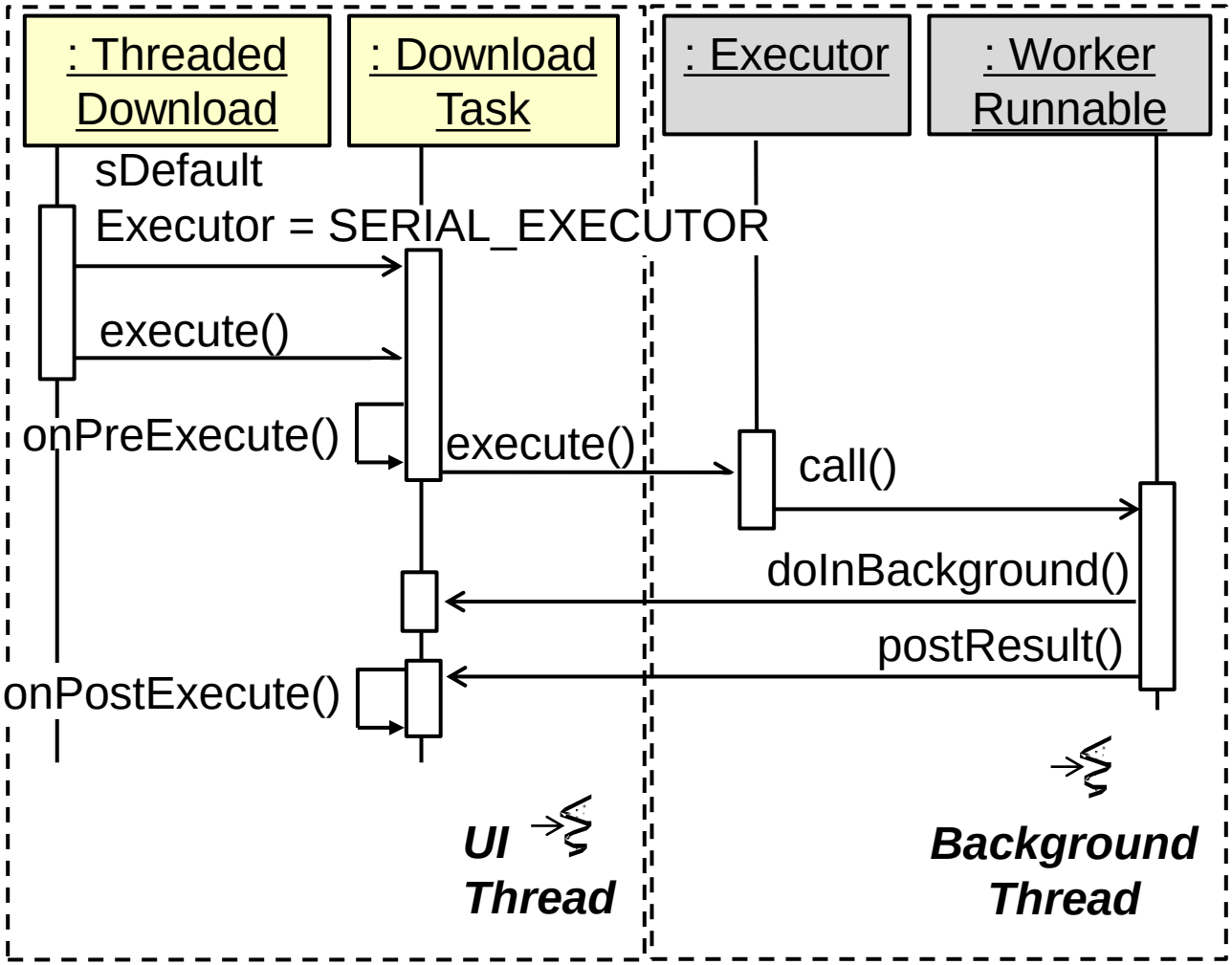


See en.wikipedia.org/wiki/Strategy_pattern

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

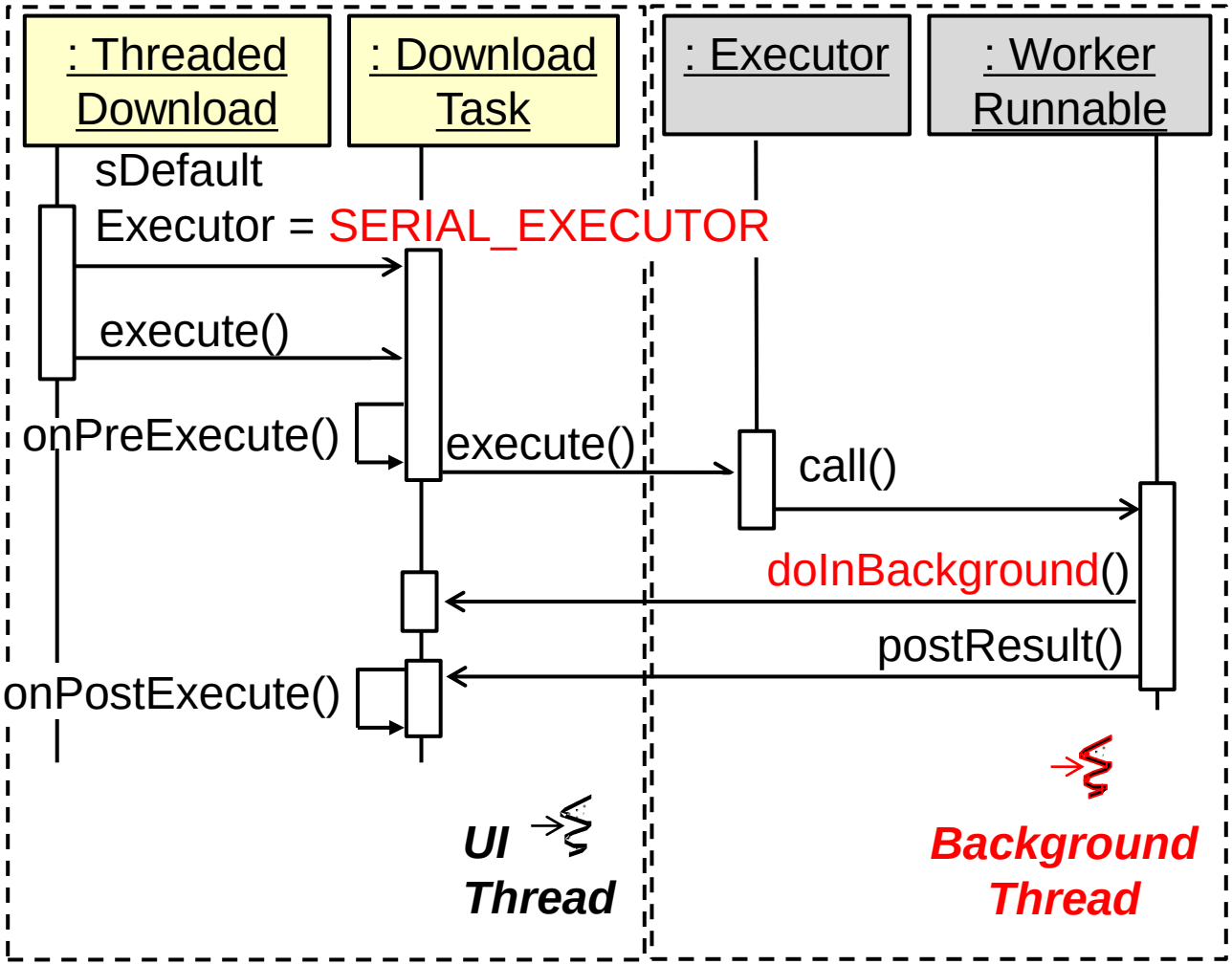




See developer.android.com/reference/java/util/concurrent/Executor.html

Black-box Elements of the AsyncTask Framework

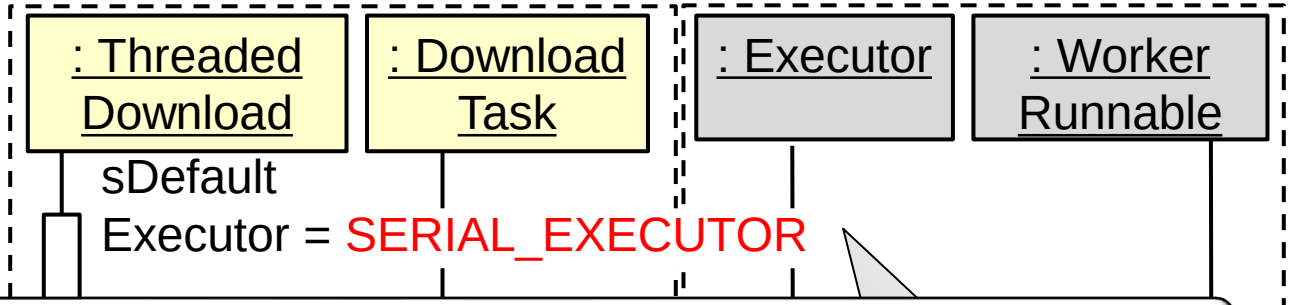
- AsyncTask can be configured via a # of Executor strategies



SERIAL_EXECUTOR executes tasks one at a time in serial order

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

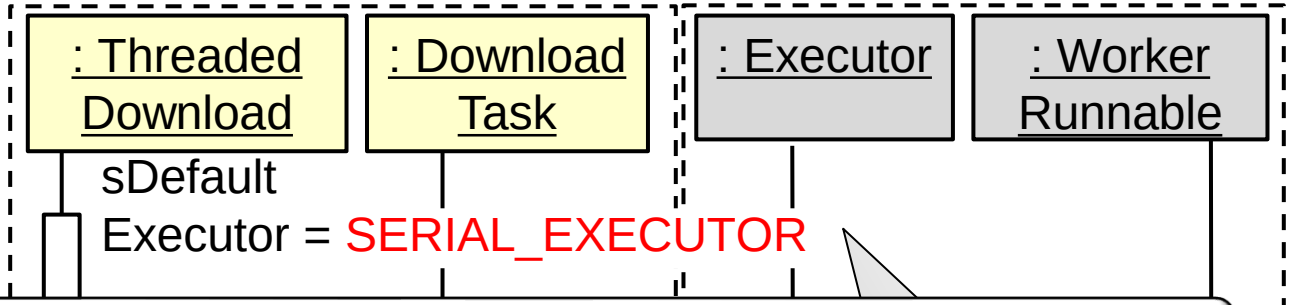



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

See docs.oracle.com/javase/8/docs/api/java/util/concurrent/Executor.html

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

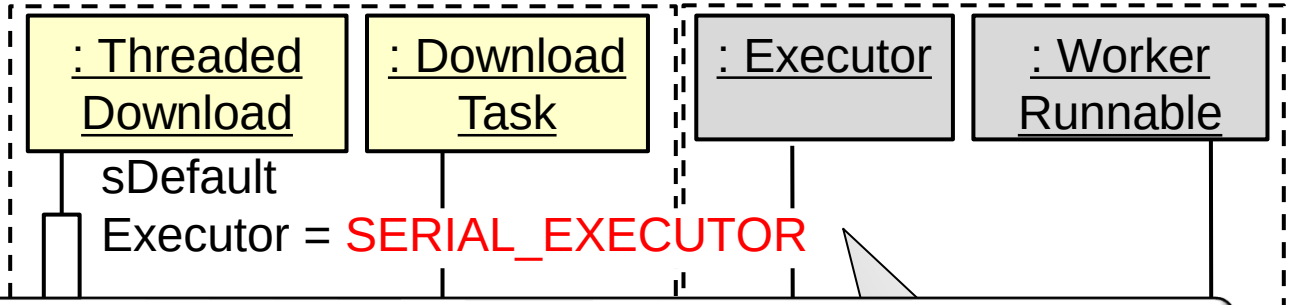



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

Queue up waiting tasks

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

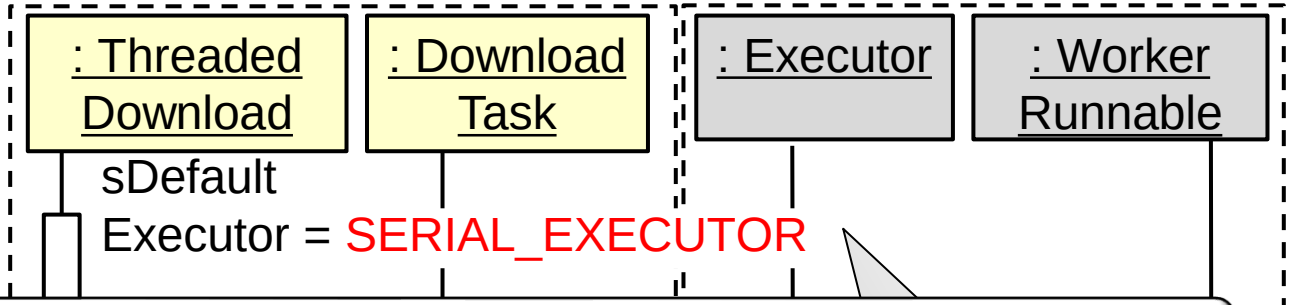



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

Currently active task

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

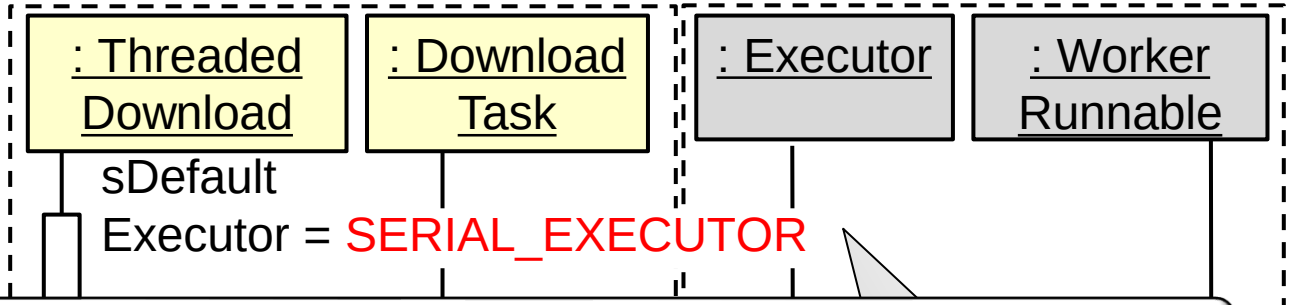



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

Execute a task

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

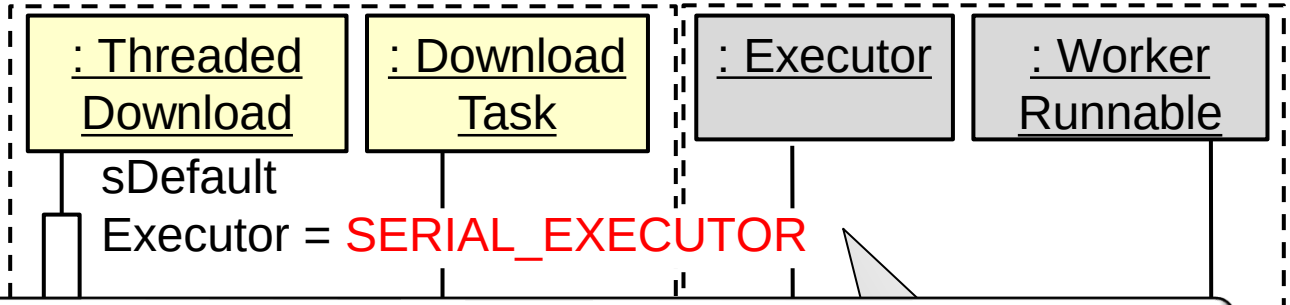



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

Enqueue a lambda that runs the task

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

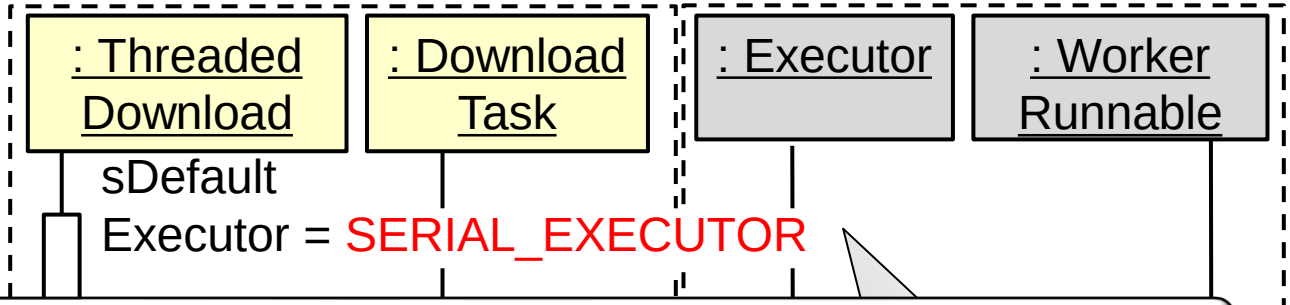



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

Schedule a new task if there isn't one currently running

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

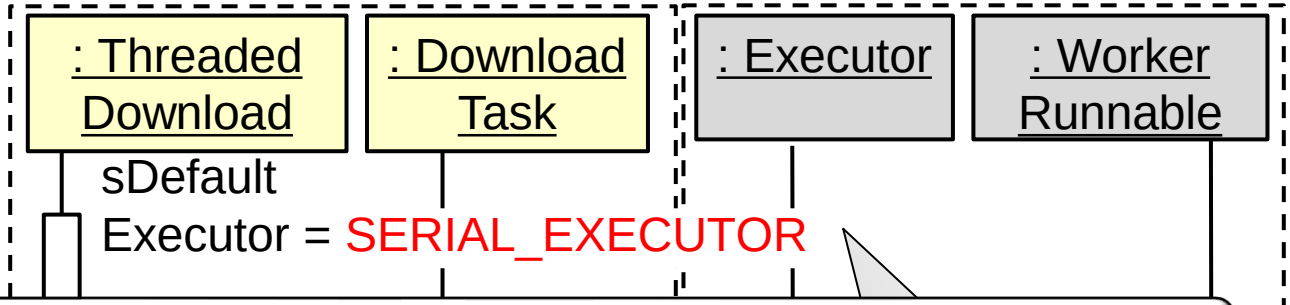



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

Schedule a new task for execution if one's available

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

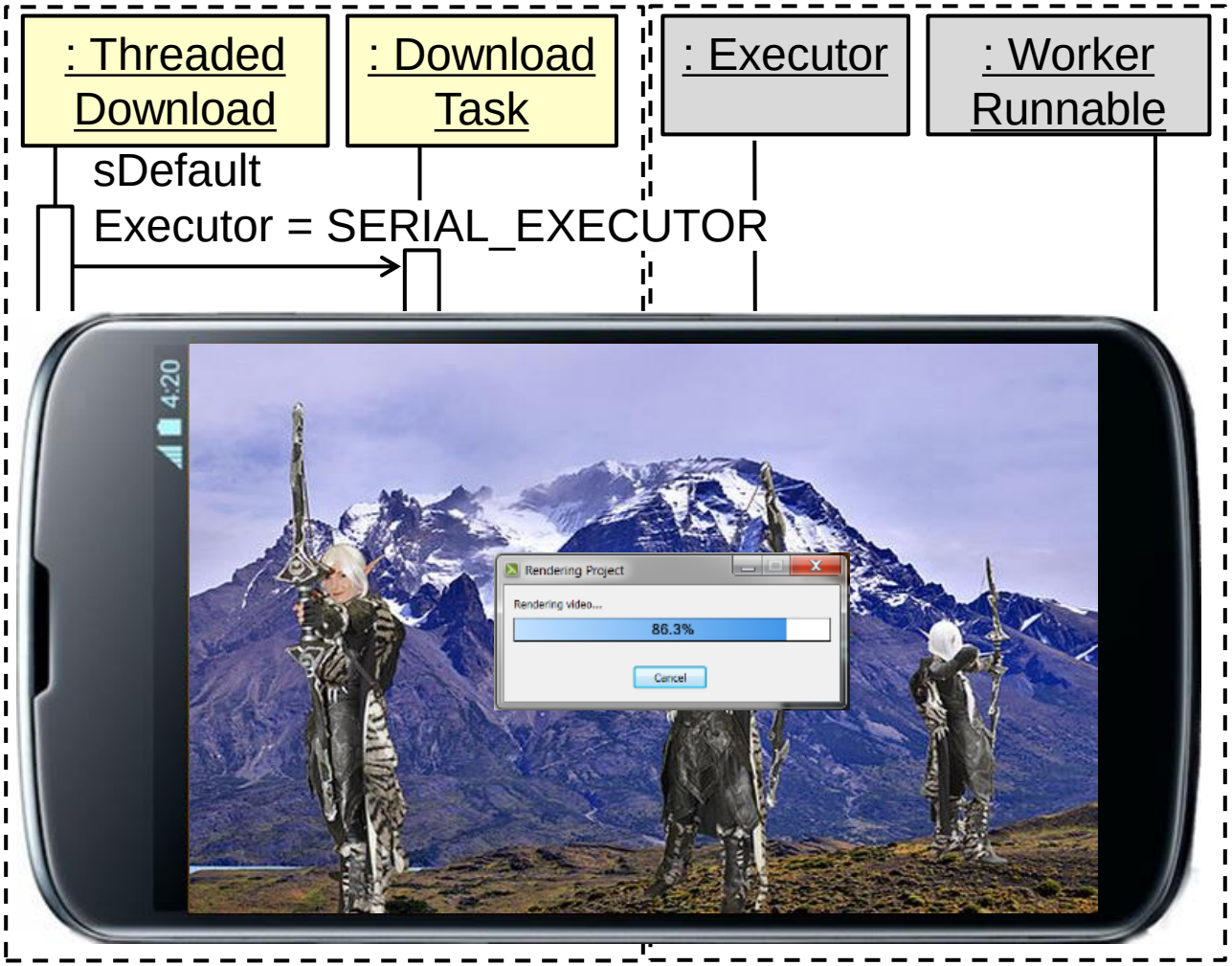



```
class SerialExecutor implements Executor {
  final ArrayDeque<Runnable> mTasks = new ArrayDeque<>();
  Runnable mActive;

  public synchronized void execute(final Runnable r) {
    mTasks.offer(() -> { try { r.run(); }
                         finally { scheduleNext(); }});
    if (mActive == null) scheduleNext();
  }

  protected synchronized void scheduleNext() {
    if ((mActive = mTasks.poll()) != null)
      THREAD_POOL_EXECUTOR.execute(mActive);
  }
}
```

Schedule a new task for execution if one's available

Black-box Elements of the AsyncTask Framework

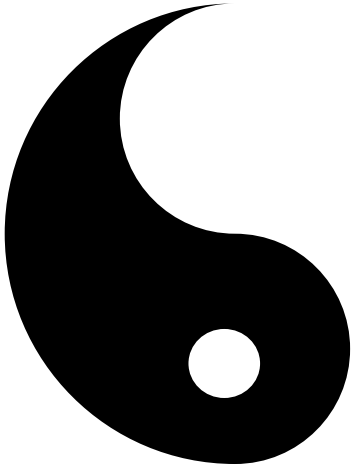
- AsyncTask can be configured via a # of Executor strategies

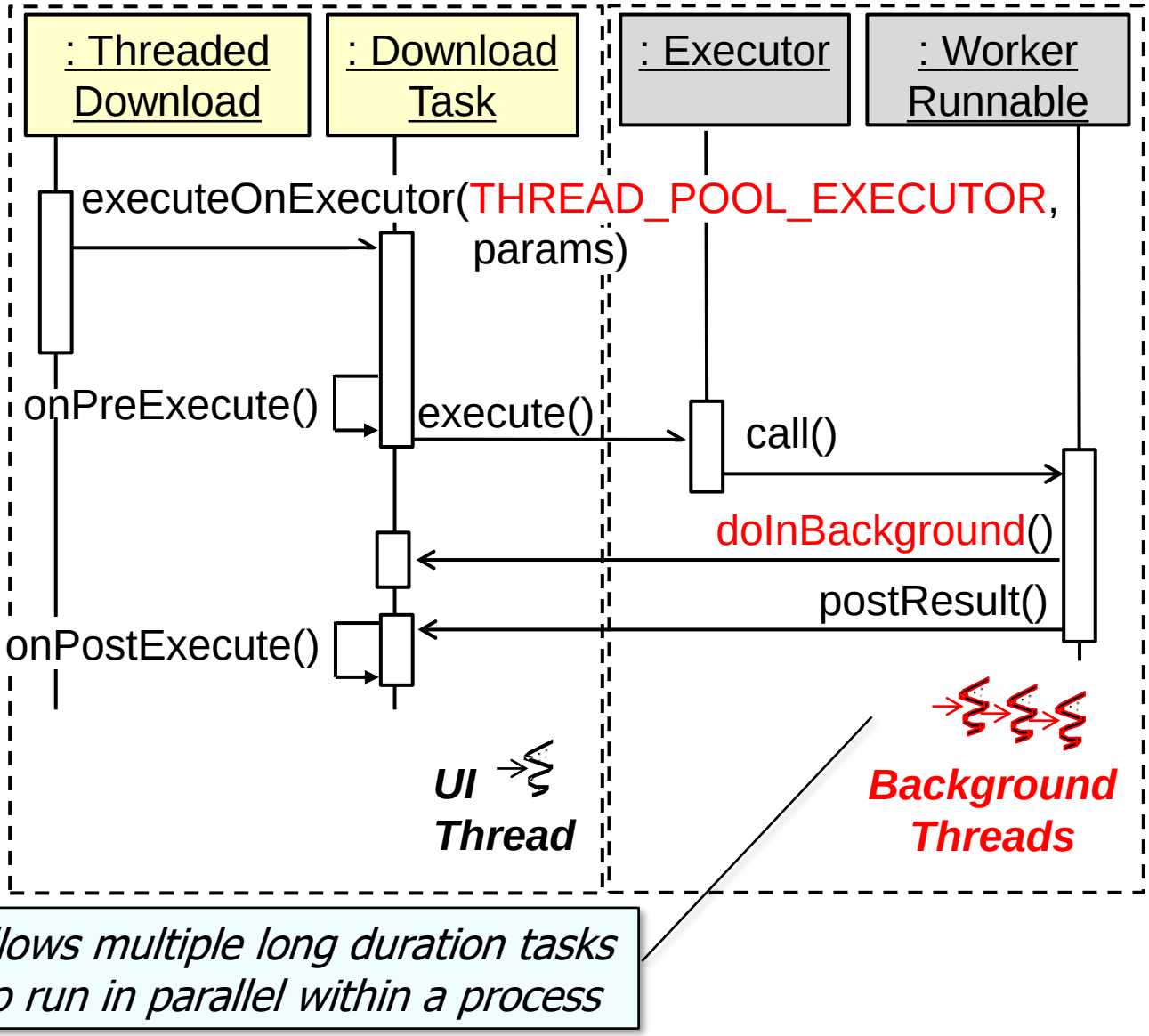


Some apps need to run AsyncTask objects in parallel instead of serially

Black-box Elements of the AsyncTask Framework

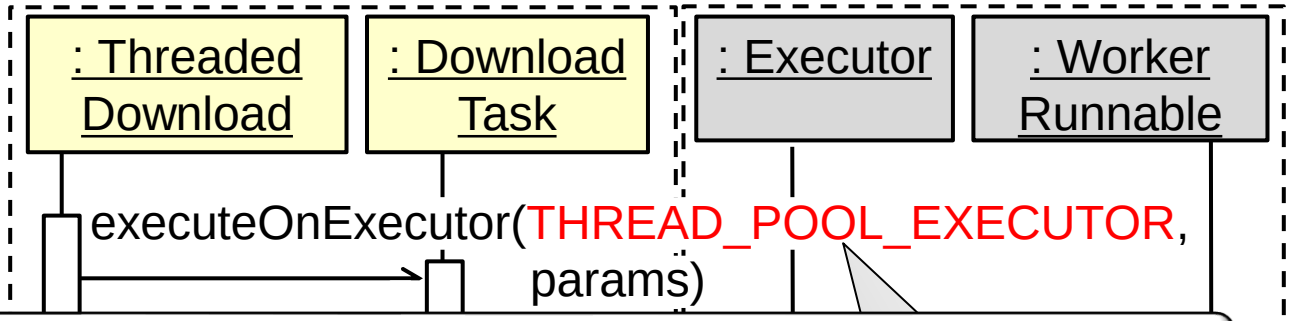
- AsyncTask can be configured via a # of Executor strategies





Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

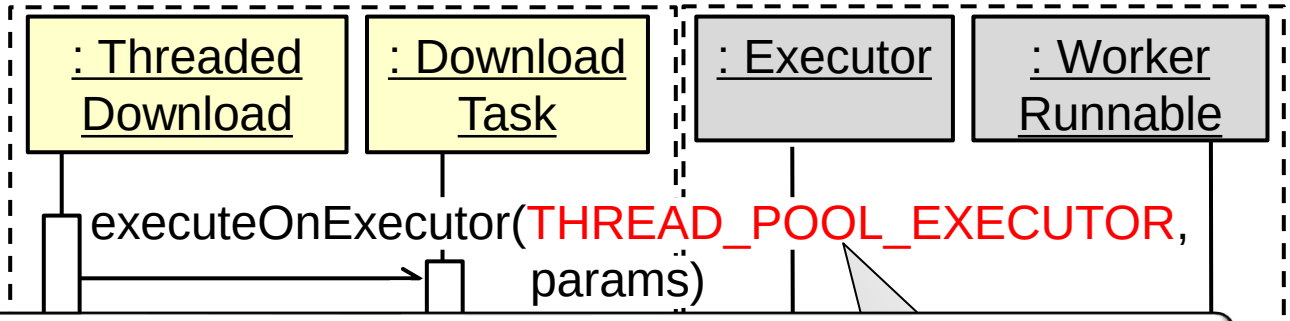



```
static final int CPU_COUNT =
  Runtime.getRuntime().availableProcessors();
static final int CORE_POOL_SIZE = CPU_COUNT + 1;
static final int MAXIMUM_POOL_SIZE = CPU_COUNT * 2 + 1;
static final int KEEP_ALIVE = 1;


static final BlockingQueue<Runnable> sPoolWorkQueue =
  new LinkedBlockingQueue<Runnable>(128);

static Executor THREAD_POOL_EXECUTOR = new
  ThreadPoolExecutor(CORE_POOL_SIZE, MAXIMUM_POOL_SIZE,
                     KEEP_ALIVE, TimeUnit.SECONDS,
                     sPoolWorkQueue, sThreadFactory);
```

THREAD_POOL_EXECUTOR can be used to execute tasks in parallel

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies



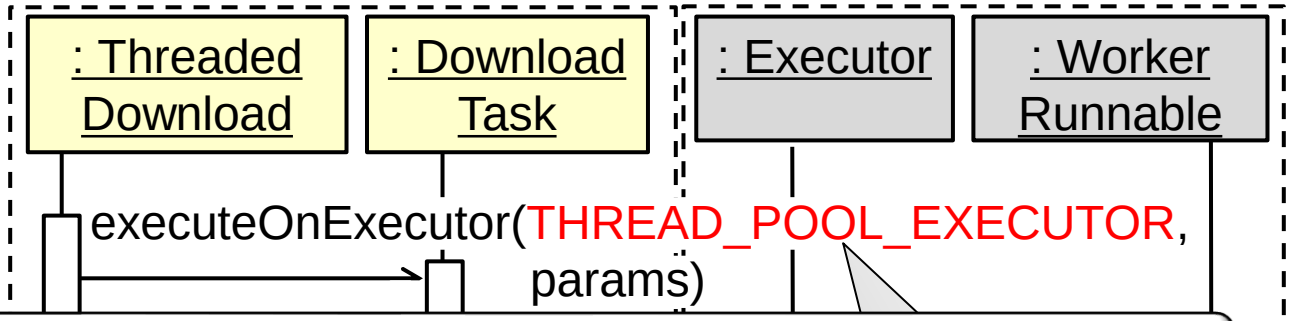

```
static final int CPU_COUNT =
  Runtime.getRuntime().availableProcessors();
static final int CORE_POOL_SIZE = CPU_COUNT + 1;
static final int MAXIMUM_POOL_SIZE = CPU_COUNT * 2 + 1;
static final int KEEP_ALIVE = 1;
```

Determine the "core" & "max" pool sizes

```
static final BlockingQueue<Runnable> sPoolWorkQueue =
  new LinkedBlockingQueue<Runnable>(128);

static Executor THREAD_POOL_EXECUTOR = new
  ThreadPoolExecutor(CORE_POOL_SIZE, MAXIMUM_POOL_SIZE,
                     KEEP_ALIVE, TimeUnit.SECONDS,
                     sPoolWorkQueue, sThreadFactory);
```

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies



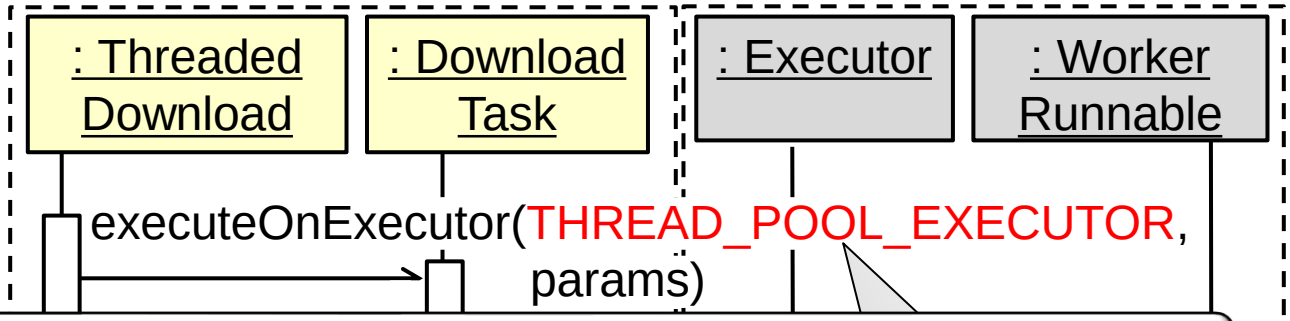

```
static final int CPU_COUNT =
  Runtime.getRuntime().availableProcessors();
static final int CORE_POOL_SIZE = CPU_COUNT + 1;
static final int MAXIMUM_POOL_SIZE = CPU_COUNT * 2 + 1;
static final int KEEP_ALIVE = 1;
```

Create a queue of up to 128 tasks

```
static final BlockingQueue<Runnable> sPoolWorkQueue =
  new LinkedBlockingQueue<Runnable>(128);

static Executor THREAD_POOL_EXECUTOR = new
  ThreadPoolExecutor(CORE_POOL_SIZE, MAXIMUM_POOL_SIZE,
                     KEEP_ALIVE, TimeUnit.SECONDS,
                     sPoolWorkQueue, sThreadFactory);
```

Black-box Elements of the AsyncTask Framework

- AsyncTask can be configured via a # of Executor strategies

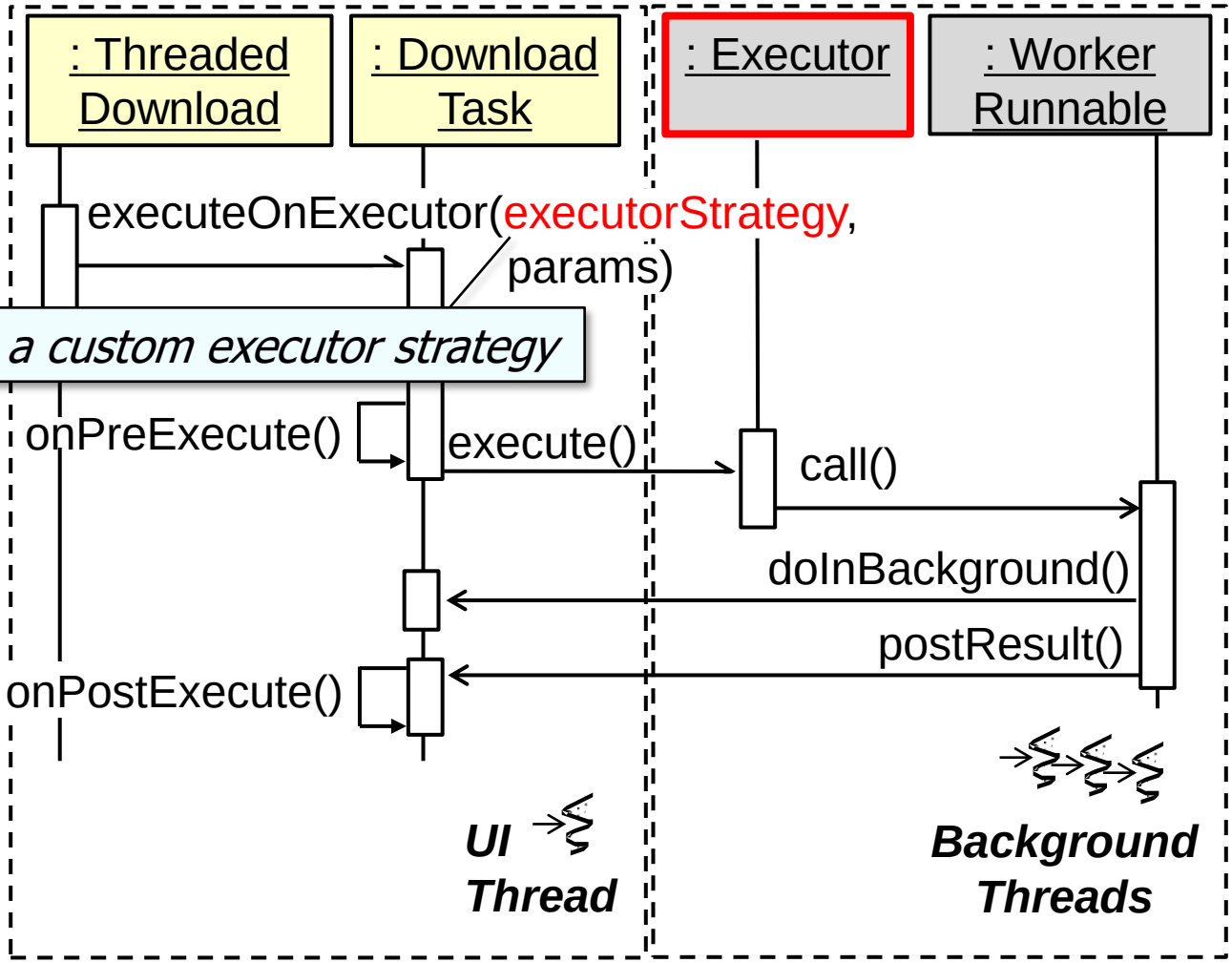



```
static final int CPU_COUNT =
  Runtime.getRuntime().availableProcessors();
static final int CORE_POOL_SIZE = CPU_COUNT + 1;
static final int MAXIMUM_POOL_SIZE = CPU_COUNT * 2 + 1;
static final int KEEP_ALIVE = 1;


static final BlockingQueue<Runnable> sPoolWorkQueue =
  new LinkedBlockingQueue<Runnable>(128);

static Executor THREAD_POOL_EXECUTOR = new
  ThreadPoolExecutor(CORE_POOL_SIZE, MAXIMUM_POOL_SIZE,
                     KEEP_ALIVE, TimeUnit.SECONDS,
                     sPoolWorkQueue, sThreadFactory);
```

Create the custom thread pool

Black-box Elements of the AsyncTask Framework

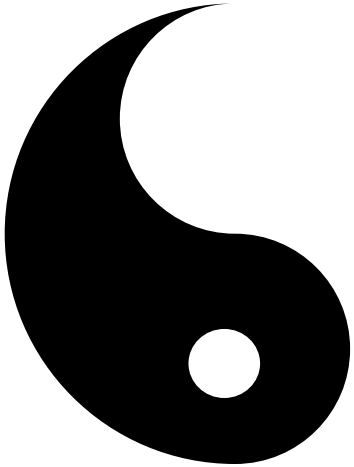
- AsyncTask can be configured via a # of Executor strategies

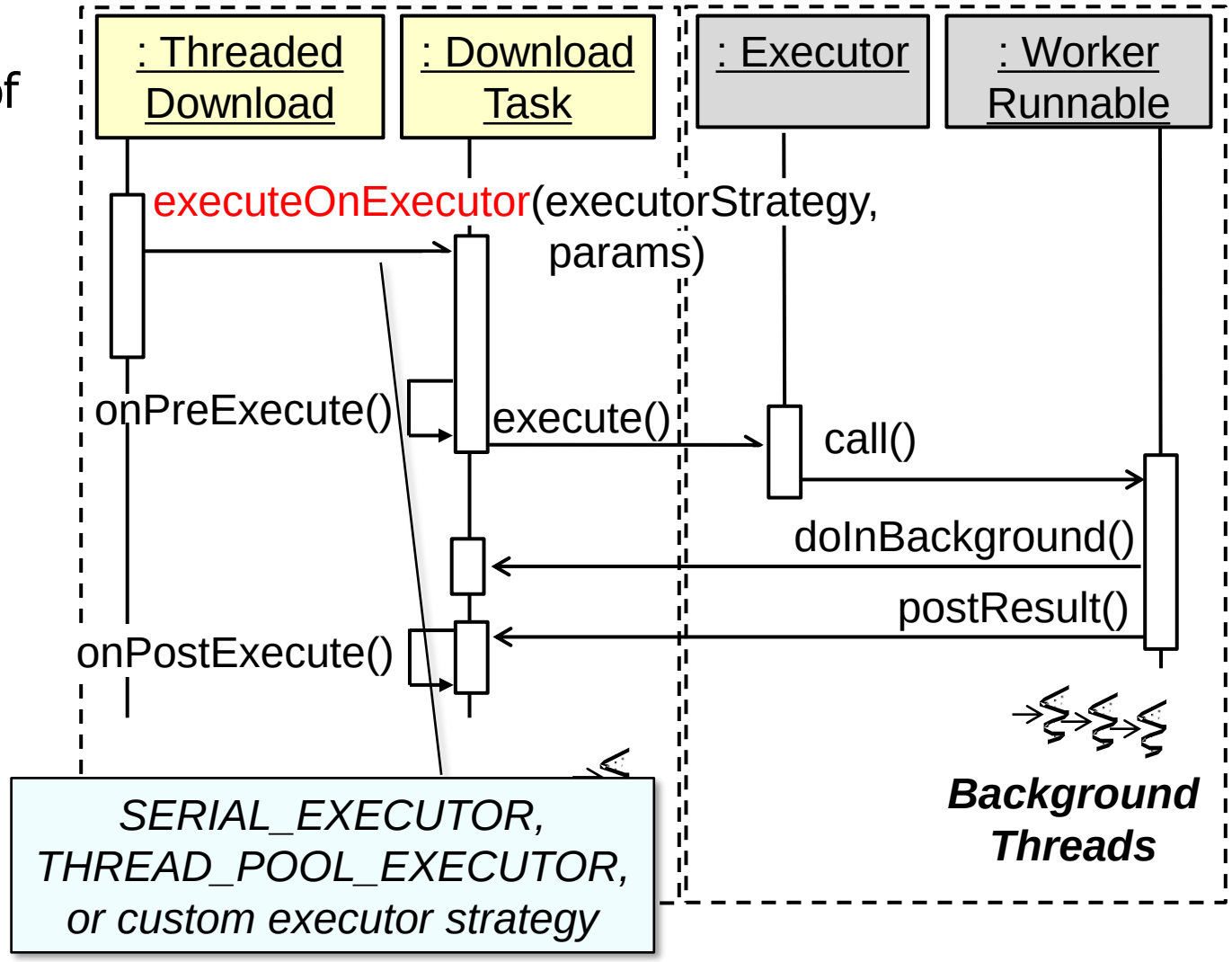


See developer.android.com/reference/java/util/concurrent/ExecutorService.html

Black-box Elements of the AsyncTask Framework

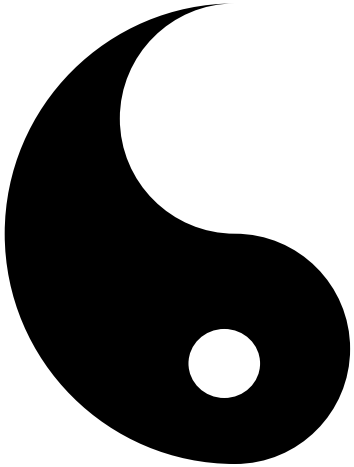
- AsyncTask can be configured via a # of Executor strategies

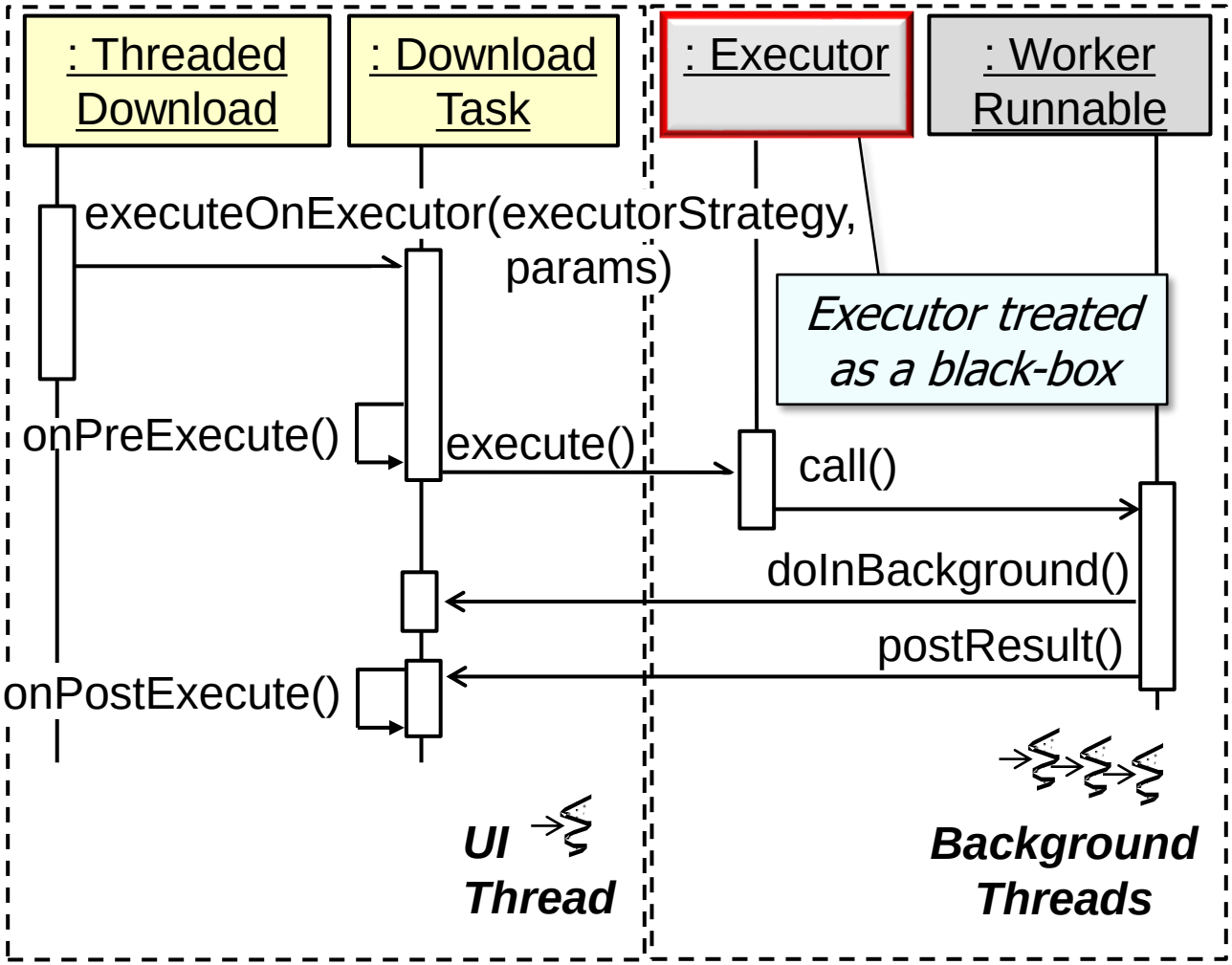




Black-box Elements of the AsyncTask Framework

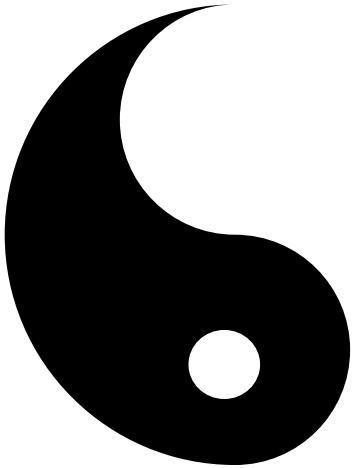
- AsyncTask can be configured via a # of Executor strategies

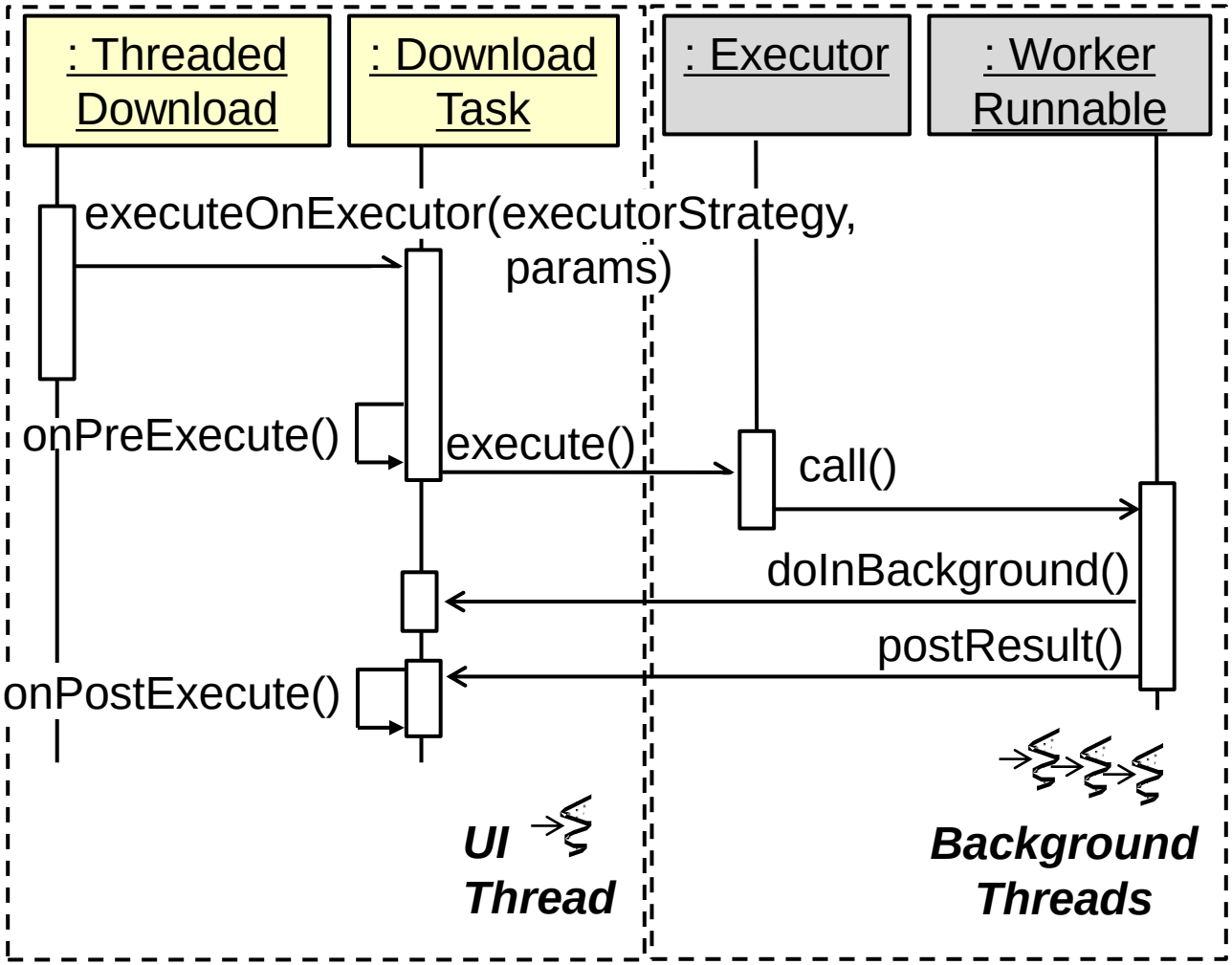




Black-box Elements of the AsyncTask Framework

- AsyncTask ensures certain operations are safe without explicit synchronizations

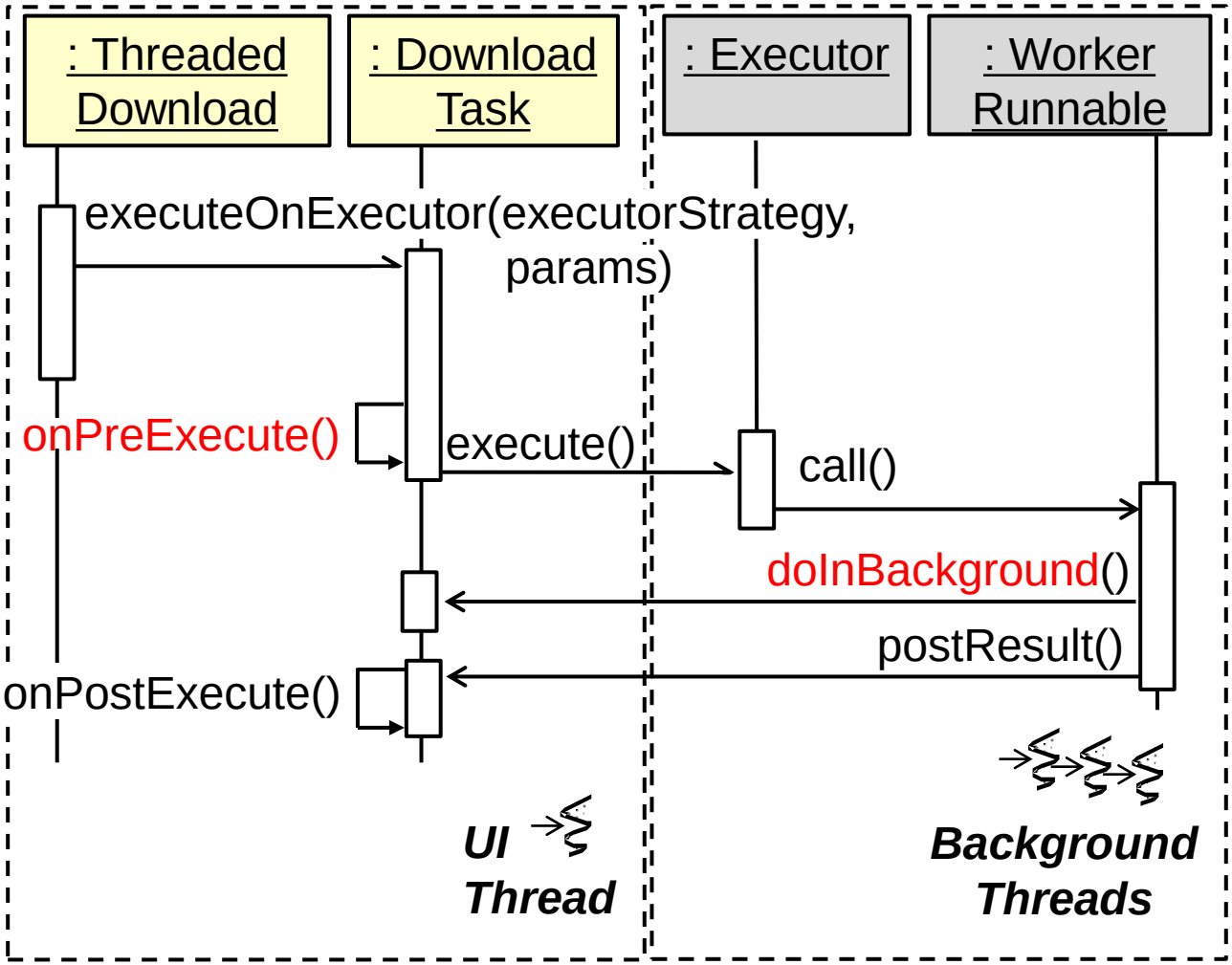




See en.wikipedia.org/wiki/Happened-before

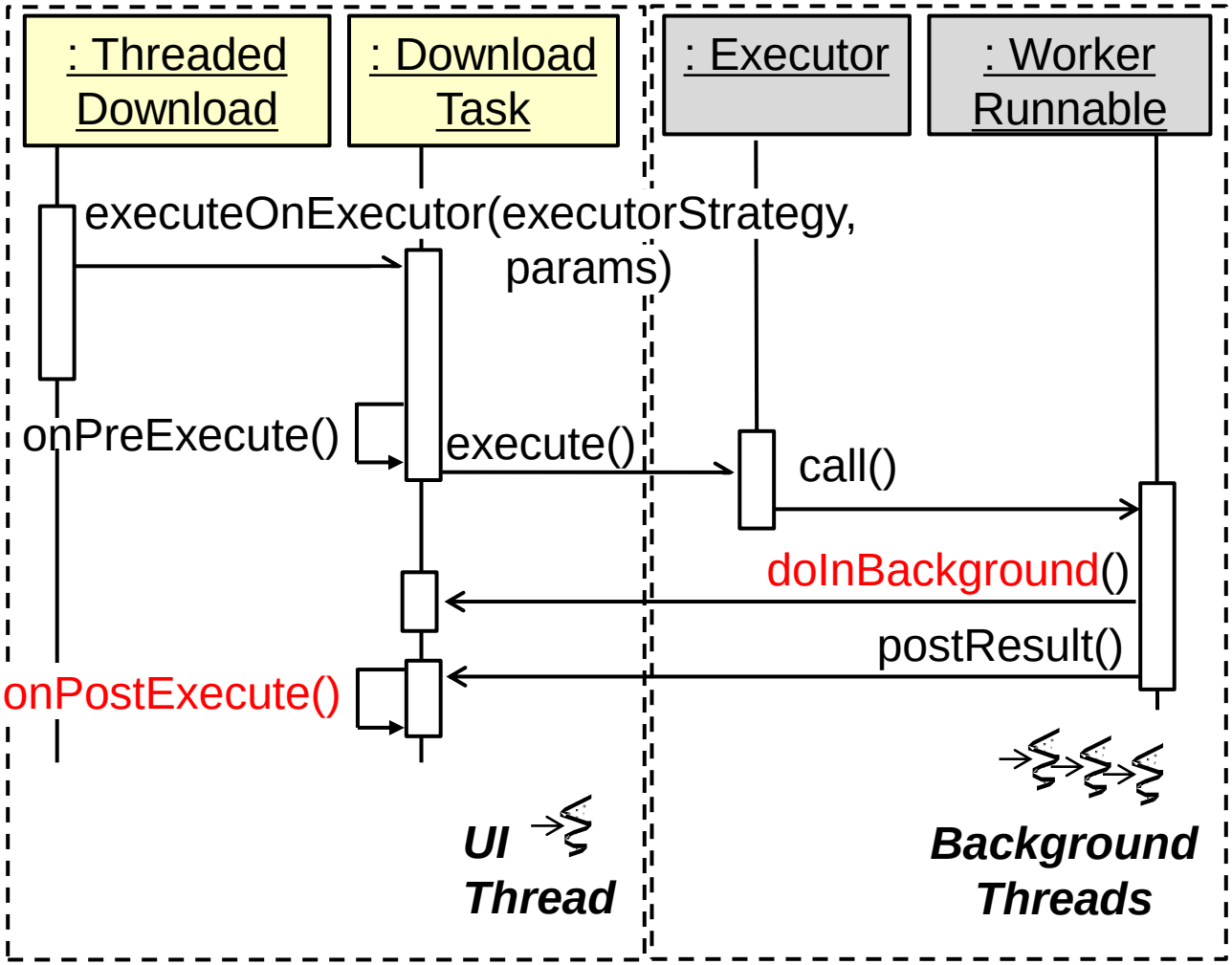
Black-box Elements of the AsyncTask Framework

- AsyncTask ensures certain operations are safe without explicit synchronizations, e.g.
  - Set fields in the AsyncTask ctor or onPreExecute(), & refer to them in doInBackground()



Black-box Elements of the AsyncTask Framework

- AsyncTask ensures certain operations are safe without explicit synchronizations, e.g.
  - Set fields in the AsyncTask ctor or onPreExecute(), & refer to them in doInBackground()
  - Set member fields in doInBackground() & refer to them in onProgressUpdate() & onPostExecute()



End of The AsyncTask Framework: Structure & Functionality