Leveraging the Power of the Android Platform

UNDERSTANDING THE ANDROID THREADING MODEL

Jim Wilson
MOBILE SOLUTIONS DEVELOPER & ARCHITECT

@hedgehogjim blog.jwhh.com

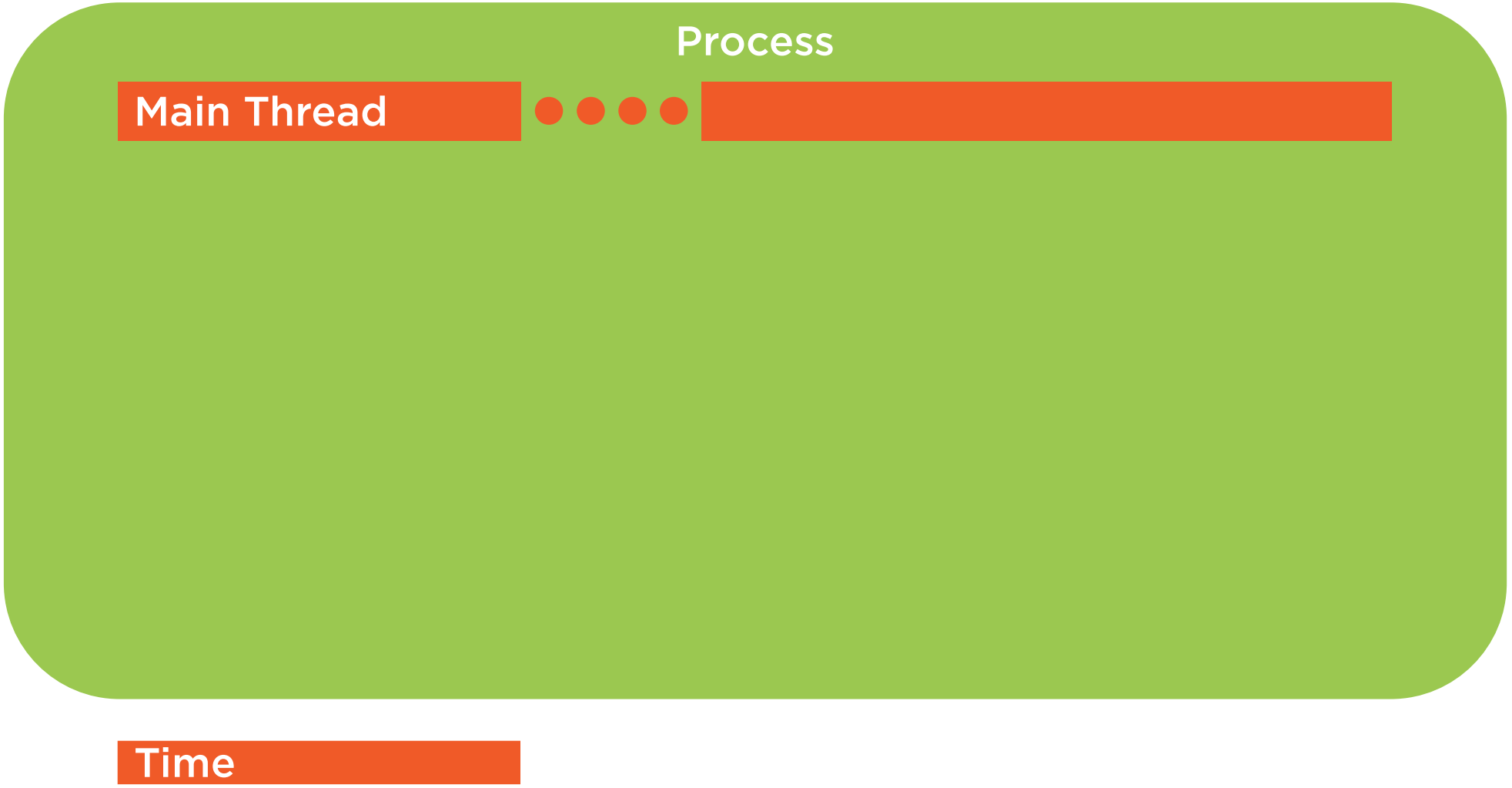
What to Expect from This Course

- Android Threading Model
- Executing Background Work with Services
- Performing Data Syncs with JobScheduler
- Broadcast Receivers
- Scheduling Time-sensitive Work with Alarms

What to Expect from This Module

- Protecting the Main Thread with StrictMode
- Performing Background Work with AsyncTask
- Providing AsyncTask Progress Updates
- Working with Handlers

Protecting the Main Thread



Protecting the Main Thread

Common operations to avoid

- Reading from “disk” storage
- Writing to “disk” storage
- Interacting with network

Being certain can be challenging

- Programs are often complex
- Operations may unexpectedly occur

Protecting the Main Thread

StrictMode class

- Can detect undesirable operations
- Enforces a penalty when detected

Use during debugging/testing

- Build desired thread policy
- Set policy at app/activity start

Protecting the Main Thread

Setting thread policy

- Use StrictMode.setThreadPolicy
- Accepts StrictMode.ThreadPolicy

Creating ThreadPolicy

- Uses builder pattern
- StrictMode.ThreadPolicy.Builder

Protecting the Main Thread

Set what you want to detect

- detectDiskReads
- detectDiskWrites
- detectNetwork
- Commonly use detectAll

Decide what you want the penalty to be

- penaltyLog
- penaltyException
- penaltyDialog
- penaltyDeath

Understanding the Role of the Main Thread

The main thread and app UI

- Most UI work performed on main thread
- Many programmatic UI operations only allowed to occur on main thread

Doing Work on a Background Thread

Perform the long-running work

- Runs on background thread

Present work results

- May require interacting with UI
- Runs on main thread

Doing Work with AsyncTask

AsyncTask class

- Manages threading details
- Methods for each phase of work
- Methods run on appropriate thread

Performing work with AsyncTask

- Define new class that extends AsyncTask
- Override appropriate methods

AsyncTask Methods

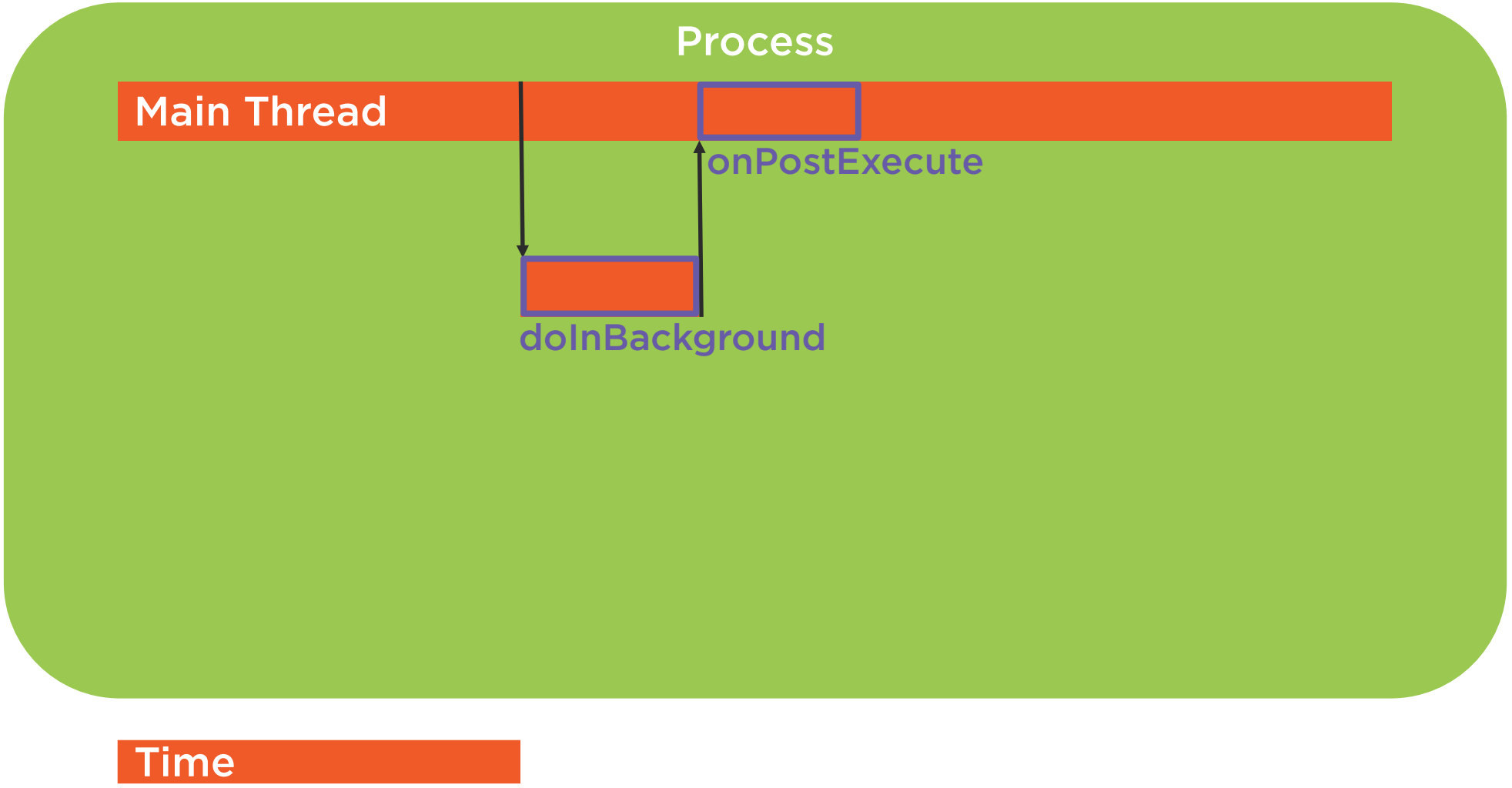
doInBackground method

- Runs on background thread
- Add code to do the work

onPostExecute method

- Runs on main thread
- Add code to present results

AsyncTask Methods



Passing Data Between AsyncTask Methods

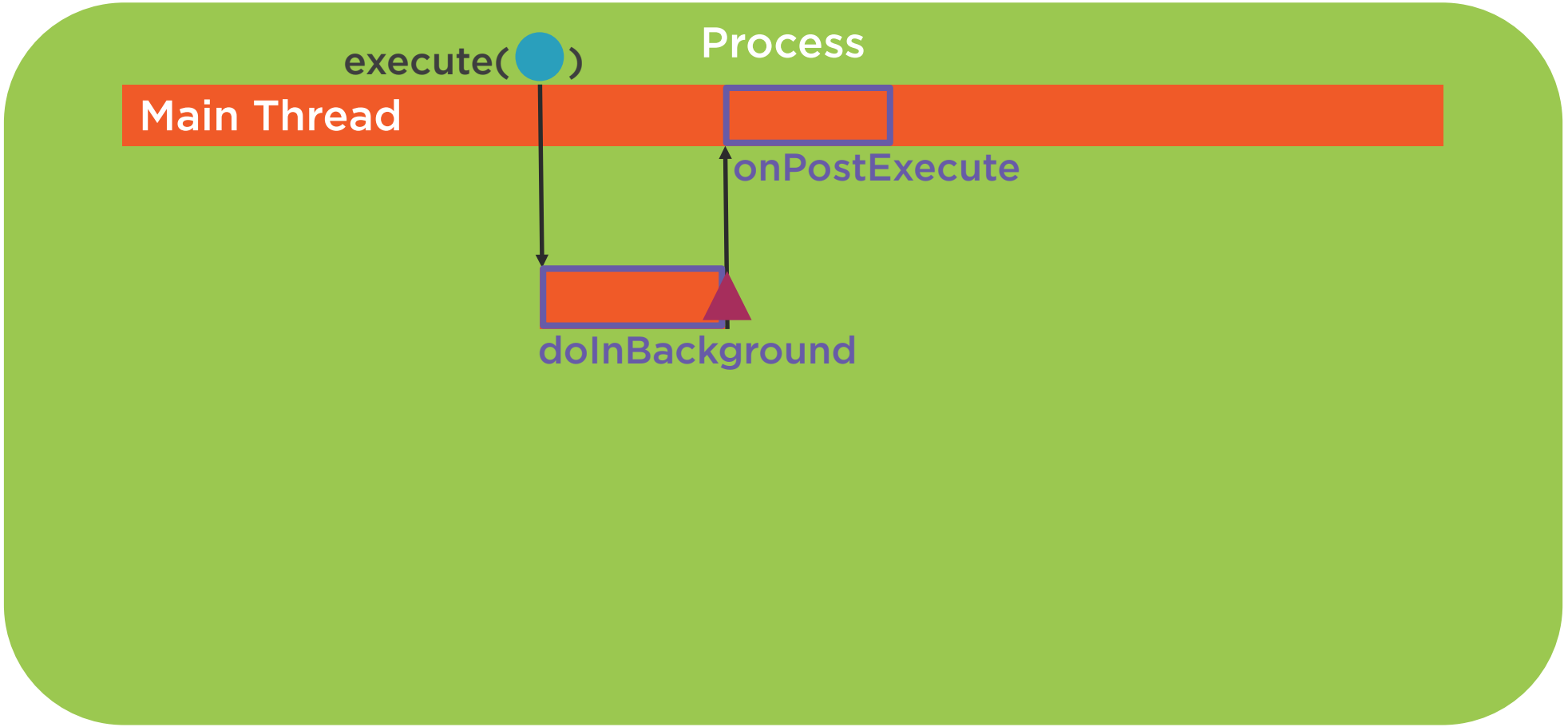
Initiating AsyncTask processing

- Call execute method
- Accepts a variable length parameter list
- Parameters passed to doInBackground

Providing background work results

- Return from doInBackground method
- Value passed to onPostExecute

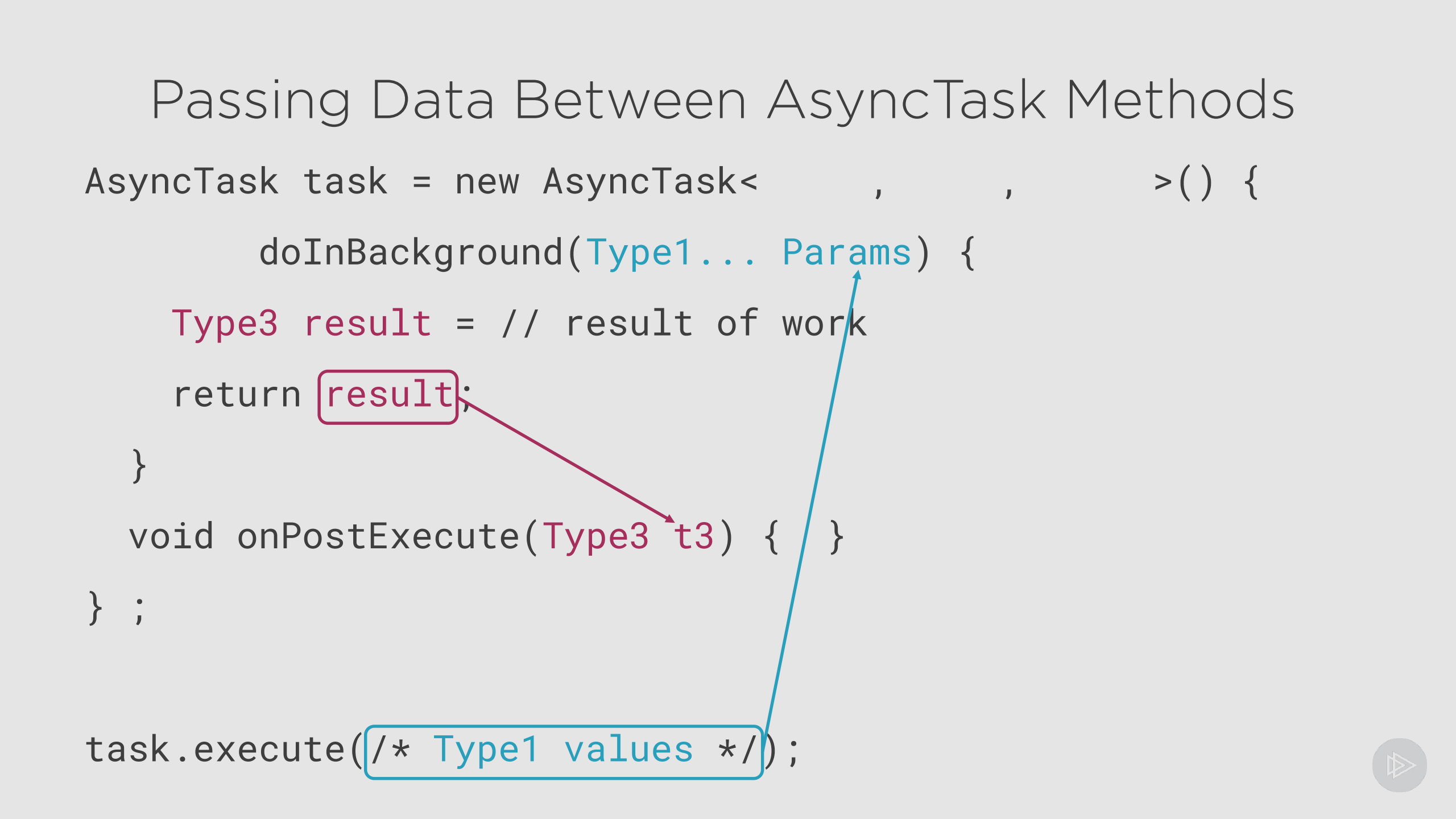
Passing Data Between AsyncTask Methods



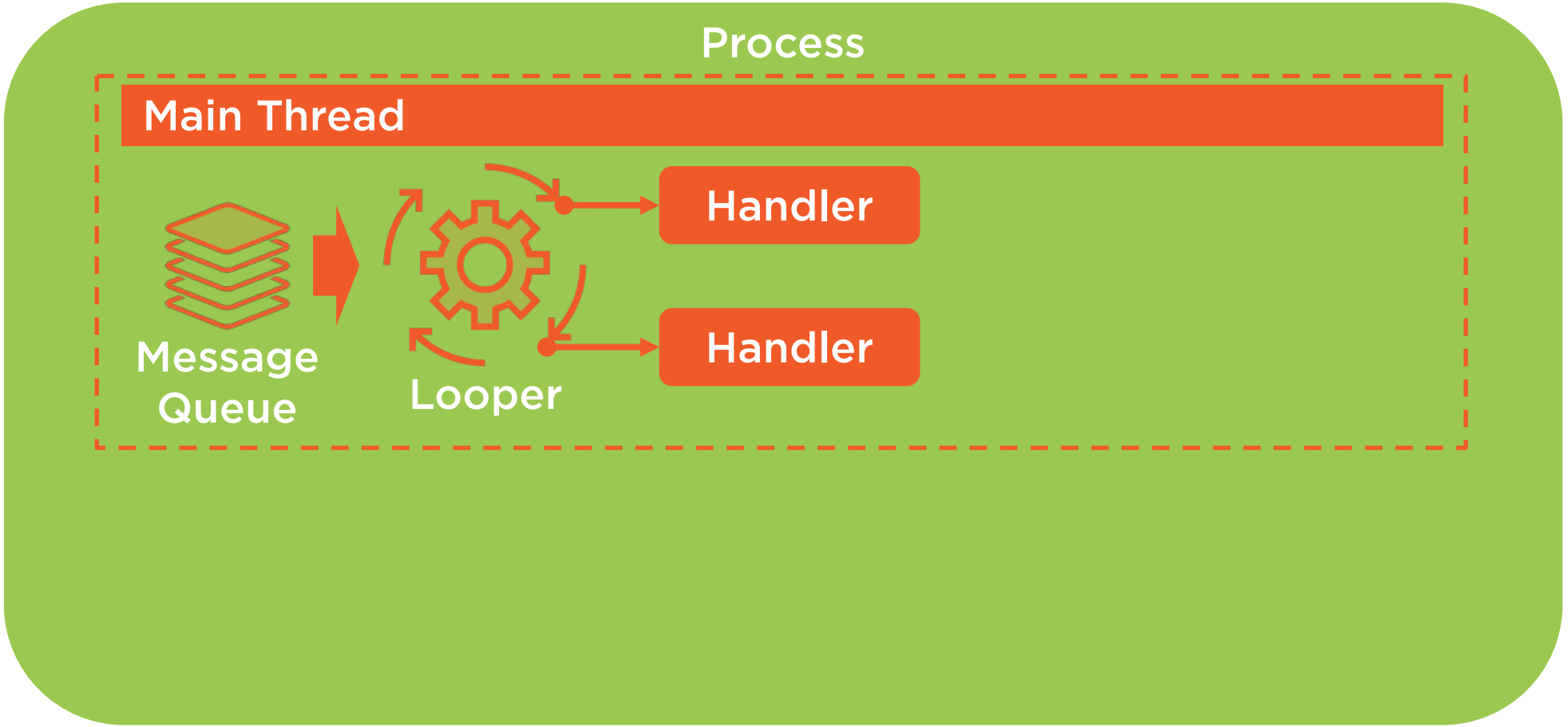
Passing Data Between AsyncTask Methods

```
AsyncTask task = new AsyncTask<      ,      ,      >() {
        doInBackground(Type1... Params) {
    Type3 result = // result of work
    return result;
  }
  void onPostExecute(Type3 t3) {  }
} ;


task.execute(/* Type1 values */);
```



Working with Handlers



Working with Handlers

LooperThread

- Has a Looper and MessageQueue
- Work dispatched to Handler instances
- Main thread is a LooperThread
- Can create others if needed

Handlers are our main point of interaction

- Handler instance enqueues work
- Handler instance receives the work when dispatched by Looper

Working with Handlers

Constructing a Handler

- Must be associated with a Looper
- By default uses current thread's Looper
- Can associate with main thread by constructing with Looper.getMainLooper

Working with Handlers

Handler is bound to Looper

- Can enqueue work from any thread
- Work always performed on thread of associated Looper

Primary uses of Handlers

- Sending work to one thread from another
- Scheduling work for future execution

Summary

Main app thread

- Responsible for most of user's experience
- Many programmatic UI operations only allowed to occur on main thread

StrictMode class

- Detect undesired main thread operations
- Enforce a penalty when detected
- Do not include in app release

Summary

AsyncTask class

- Performs work on background thread
- UI interactions on main thread
- Handles details of moving data between background and main thread

Summary

LooperThread

- Has a Looper and MessageQueue
- Work dispatched to Handler instances
- Main thread is a LooperThread

Handler is bound to a Looper

- All work performed on Looper's thread

Handlers simplify some common tasks

- Scheduling work for future execution
- Dispatching work to Handler's LooperThread from a different thread