Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Master IAO
Master II–Semestre 3
Cours

Mobile & Cloud Computing

Pr. REDA Oussama Mohammed

2015/2016

Concuruncy in Android

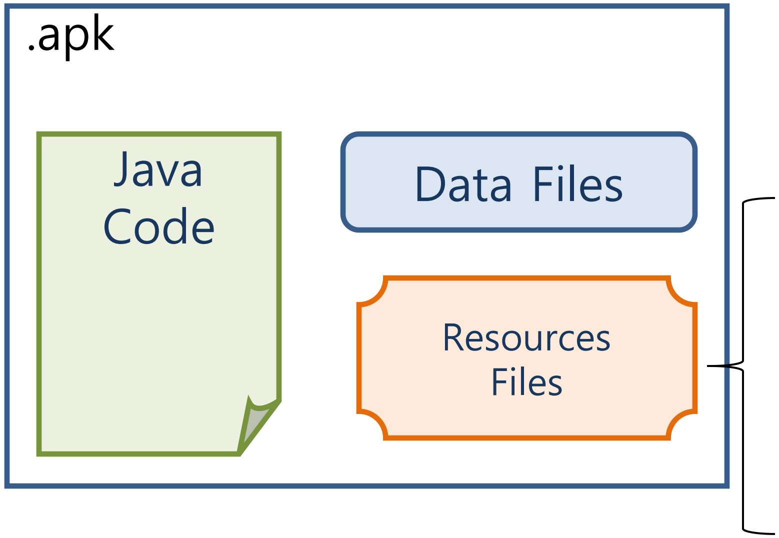
Android Application Model, Processes, UI Thread

and

Handlers

Android Application Package

- Android applications are written in Java.
- An Android application is bundled by the aapt tool into an Android package (.apk)



- res/layout: declaration layout files
- res/drawable: intended for drawing
- res/admin: bitmaps, animations for transitions
- res/values: externalized values
  - **strings**, **colors**, **styles**, etc
- res/xml: general XML files used at runtime
- res/raw: binary files (e.g. sound)

Application Components

- Android applications do not have a single entry point (e.g. no main () function).
- They have essential components that the system can instantiate a nd run as needed.
- Four basic components

Components	Description
Activity	UI component typically corresponding to one screen
Service	Background process without UI
Broadcast Receiver	Component that responds to broadcast Intents
Content Provider	Component that enables applications to share data

Activities	Presents a visual user interface for one focused endeavor the user can undertake. List of menu items a user can choose from or display photographs along with their captions
Services	Doesn't have a visual user interface, instead runs in the background Play background audio as the user attends to other matters
Broadcast Receivers	Receives and reacts to broadcast announcements An application can announce to "whoever is listening" that a picture was taken.
Content Providers	Makes a specific set of the application's data available to other applications. An application uses a contact list component
Intents	A simple message passing framework. Using intents you can broadcast messages system-wide or to a target Activity or Service.

Components - Activity

- An activity is usually a single screen:

  Implemented as a single class extending Activity.

  Displays user interface controls (views).

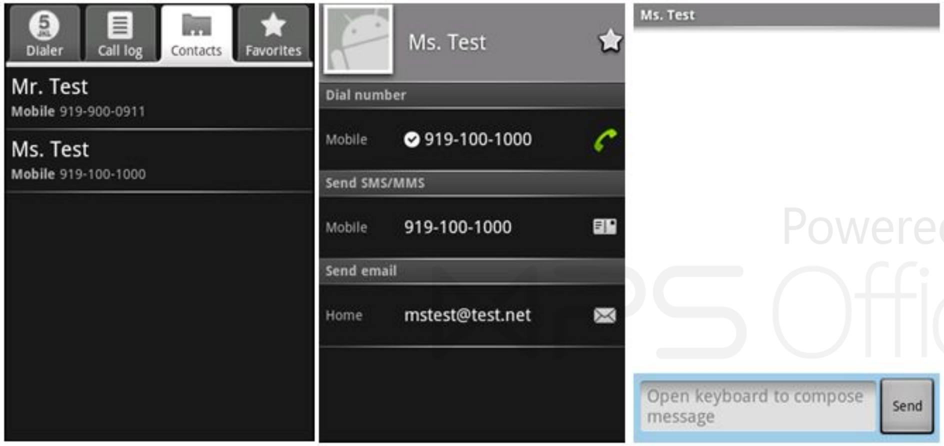
  Reacts on user input/events.

- An application typically consists of several screens:

  Each screen is implemented by one activity.

  Moving to the next screen means starting a new activity.

  An activity may return a result to the previous activity.



Mr. Test
Mobile 919-900-0911

Ms. Test
Mobile 919-100-1000

Ms. Test

Mobile 919-100-1000

Mobile 919-100-1000

Home mstest@test.net

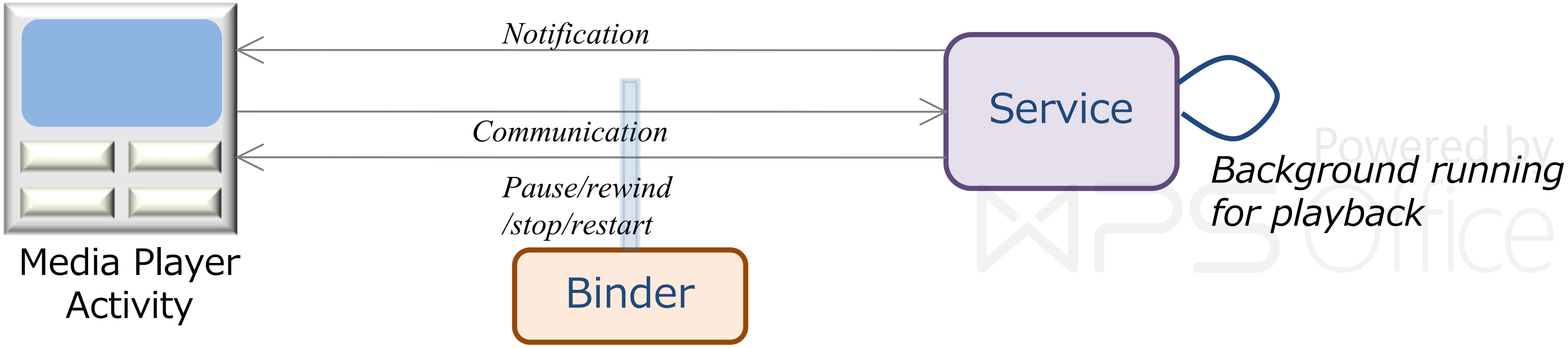
Ms. Test

Components - Service

- A service does not have a visual user interface, but rather runs in the background for an indefinite period time.

  Example: audio player, network download, etc

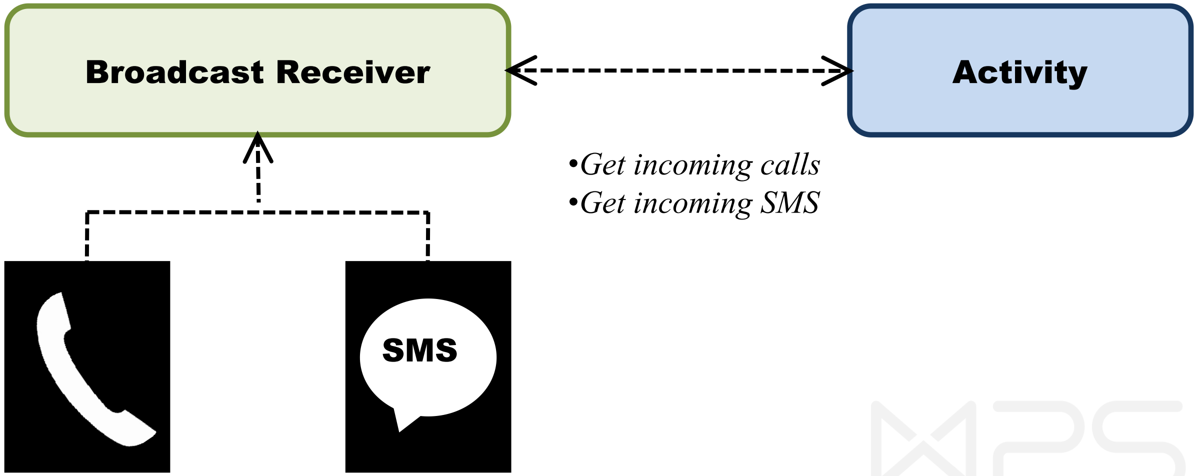
- Each service extends the Service base class.
- It is possible to bind to a running service and start the service if it's not already running.
- While connected, it is possible communicate with the service through an interface defined in an AIDL (Android Interface Definition Language).



Components - Broadcast Receivers

- A broadcast receiver is a component that receives and reacts to broadcast announcements (Intents).
  - ✓ Many broadcasts originate in system code.

    E.g. announcements that the time zone has changed, that th e battery is low, etc.



•*Get incoming calls*
•*Get incoming SMS*

Components - Broadcast Receivers (Cont)

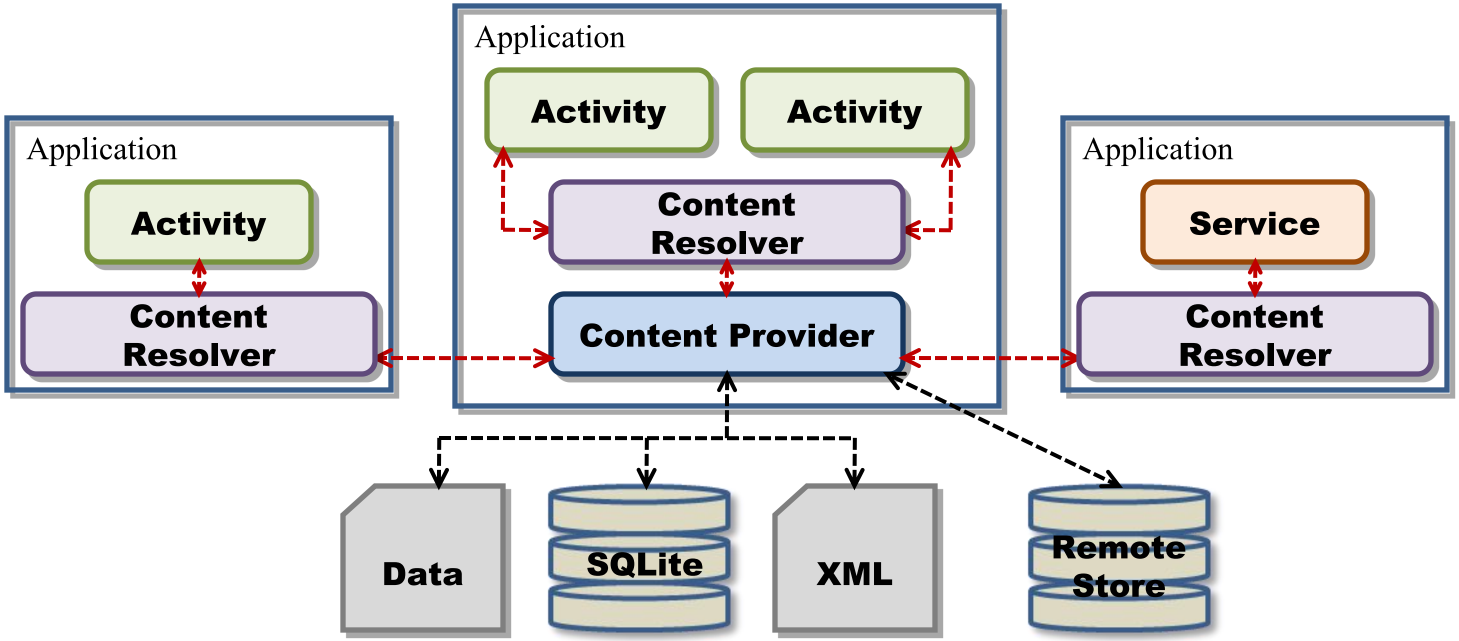
- A broadcast receiver is a component that receives and reacts to broadcast announcements. (Cont)
  - ✓ Applications can also initiate broadcasts.

    E.g. to let other applications know that some data has been d ownloaded to the device and is available for them to use.

  - ✓ Applications can also initiate announcements to let other applications know of some change in state.
  - ✓ May be used to start an activity when a message arrives.

- All receivers extend the ***BroadcastReceiver*** base class.

Components - Content Providers



- A content provider makes a specific set of the application's data available to other applications.
  - ✓ The data can be stored in the file system, in an SQLite, or in any other manner that makes sense.

Components - Content Providers (Cont)

- Using a content provider is the only way to share data between A ndroid applications.
- It extends the ContentProvider bas class and implements a stand ard set of methods to allow access to a data store.
  - ✓ Querying
  - ✓ Delete, update, and insert data
- Applications do not call these methods directly.
  - ✓ They use a ContentResolver object and call its methods instead.
  - ✓ A ContentResolver can talk to any content provider.
- Content is represented by URI and MIME type.

Intents

- Intents are simple message objects each of which consists of
  - Action to be performed (MAIN/VIEW/EDIT/PICK/DELETE/DIAL/etc)
  - Data to operate on (URI)

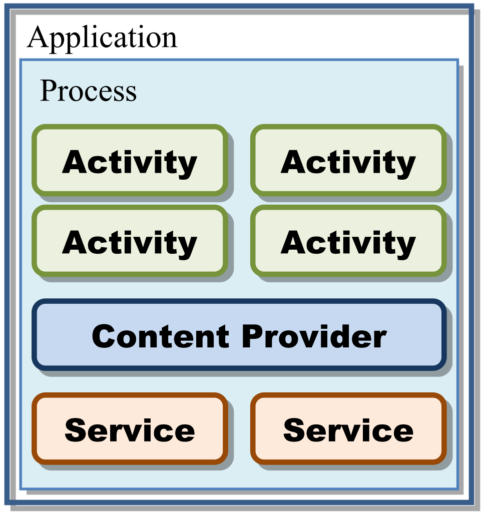
```
startActivity(new Intent(Intent.VIEW_ACTION, Uri.parse("http://www.fhnw.ch"));

startActivity(new Intent(Intent.VIEW_ACTION, Uri.parse("geo:47.480843,8.211293"));

startActivity(new Intent(Intent.EDIT_ACTION,Uri.parse("content://contacts/people/1
"));
```

Android Component Model

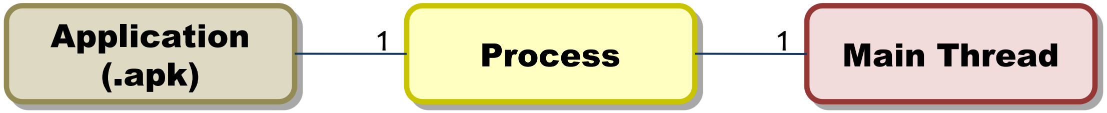
- An Android application is packaged in a .apk file.
  - ✓ A .apk file is a collection of components.



  - ✓ Components share a Linux process: by default, one process per .apk file.
  - ✓ .apk files are isolated and communicate with each other via Intents or AIDL.
  - ✓ Every component has a managed lifecycle.

Processes and Threads

- Processes
  - ✓ When the first of an application's components needs to be run, Android starts a Linux process for it with a single thread of execution (Main Thread). Additional threads can be spawned for any process.



  - ➢ Each component can run in its own process.
  - ➢ You can arrange for components to run in other processes.
  - ➢ Some components share a process while others do not.
  - ➢ Components of different applications also can run in the same process.

  - ✓ Android may decide to kill a process to reclaim resources.

Component Lifecycles

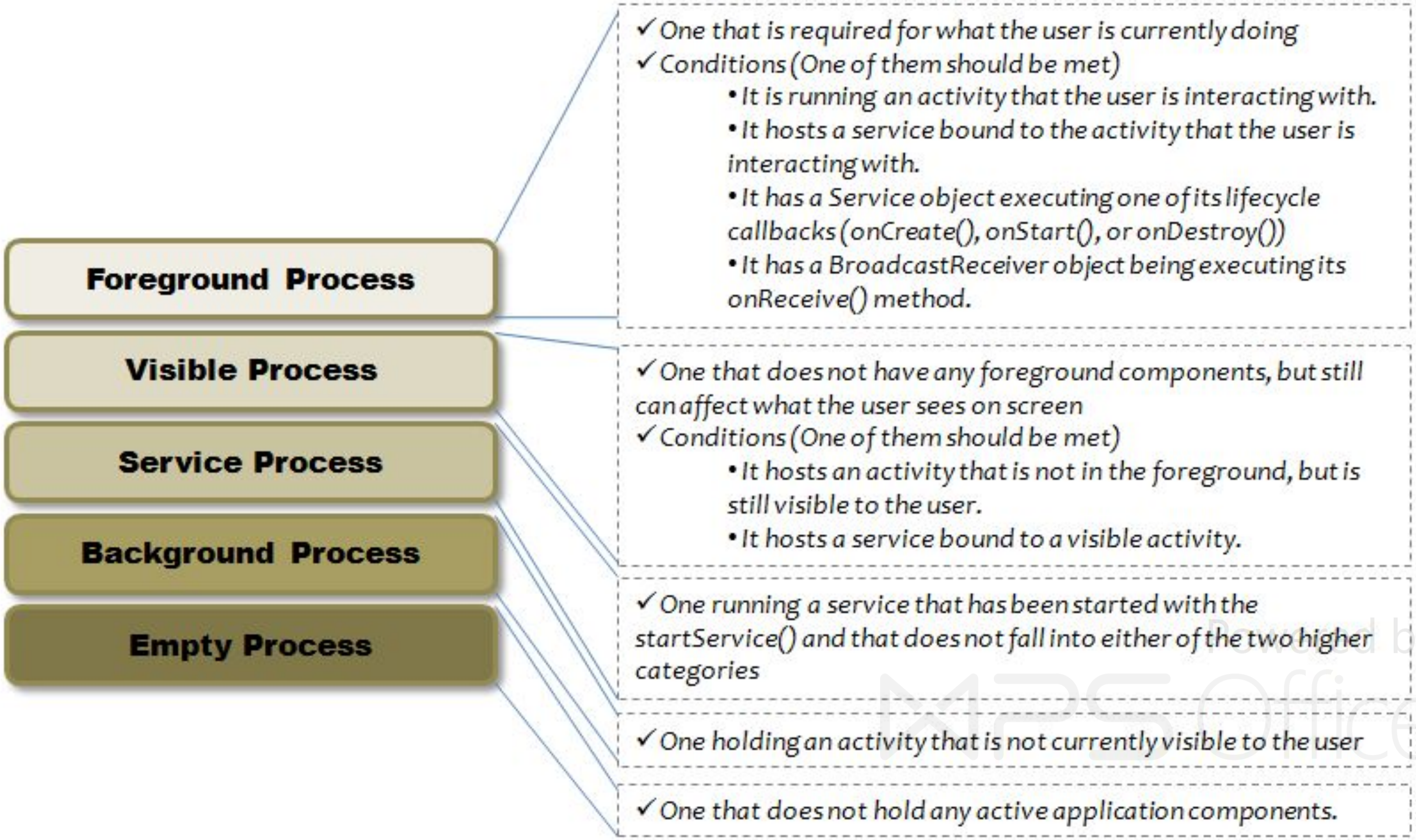
- Application components have a lifecycle — a beginning when Android instantiates them to respond to intents through to an end when the instances are destroyed.
- In between, they may sometimes be active or inactive,or, in the case of activities, visible to the user or invisible.

Component Lifecycles (Cont)

- Processes and Lifecycles
  - ✓ Android tries to maintain a process for as long as possible, but eventually it will need to remove old processes when memory runs low.
    - ➢ To determine candidates to be killed, Android places each process into an "importance hierarchy" based on the components running in it and the state of those components.
    - ➢ Processes with the lowest importance are eliminated first, then those with the next lowest, and so on.

Component Lifecycles (Cont)

- Processes and Lifecycles (Cont)
  - ✓ Five levels in the Importance Hierarchy



Foreground Process

✓ One that is required for what the user is currently doing

✓ Conditions (One of them should be met)

- It is running an activity that the user is interacting with.
- It hosts a service bound to the activity that the user is interacting with.
- It has a Service object executing one of its lifecycle callbacks (onCreate(), onStart(), or onDestroy())
- It has a BroadcastReceiver object being executing its onReceive() method.

Visible Process

✓ One that does not have any foreground components, but still can affect what the user sees on screen

✓ Conditions (One of them should be met)

- It hosts an activity that is not in the foreground, but is still visible to the user.
- It hosts a service bound to a visible activity.

Service Process

✓ One running a service that has been started with the startService() and that does not fall into either of the two higher categories

Background Process

✓ One holding an activity that is not currently visible to the user

Empty Process

✓ One that does not hold any active application components.

Processes

- Android may decide to shut down a process at some point, when memory is low and required by other processes that are more immediately serving the user.
- Application components running in the process are consequently destroyed.
- A process is restarted for those components when there's again work for them to do.
- When deciding which processes to terminate, Android weighs their relative importance to the user.
- For example, it more readily shuts down a process with activities that are no longer visible on screen than a process with visible activities.
- The decision whether to terminate a process, therefore, depends on the state of the components running in that process.

Processes and Threads

- Threads
  - ✓ Main Thread (UI Thread)
    - ➢ It is in charge of dispatching events to the appropriate user interface widgets, including drawing events.
    - ➢ It is also the thread in which your application interacts with components from the Android UI toolkit (components from the android.widget and android.view packages).
      - ■ As such, the main thread is also sometimes called the UI thread.

Processes and Threads (Cont)

- Threads
  - ✓ Main Thread
    - ➢ All components are instantiated in the main thread (UI Thread) of the specified process.
    - ➢ System calls to the components are dispatched from the main thread (UI widgets and views).
      - ■ Methods that respond to those calls always run in the main thread of the process (such as onKeyDown() to report user actions or a lifecycle callback method).

Processes and Threads (Cont)

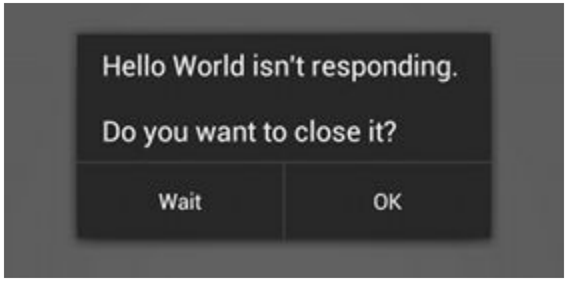
- Threads
  - ✓ Main/UI Thread (Exemple)
    - The user touches a button on the screen
      - The application's UI thread dispatches the touch event to the widget.
      - The widget sets its pressed state and posts an invalidate request to the event queue.
      - The UI thread dequeues the request and notifies the widget that it should redraw itself.

Processes and Threads (Cont)

- Threads
  - ✓ Main Thread (UI Thread)
    - ➢ When an app performs intensive work in response to user interaction, this single thread model can yield poor performance unless the application is implemented properly.
    - ➢ Specifically, if everything is happening in the UI thread, p erforming long operations such as network access or datab ase queries will block the whole UI.
      - ■ When the thread is blocked, no events can be dispatched, including drawing events.
      - ■ From the user's perspective, the application appears to hang.

Processes and Threads (Cont)

- Threads
  - ✓ If the UI thread is blocked for more than a few seconds (about 5 seconds currently) the user is presented with the infamous "application not responding" (ANR) dialog.



Hello World isn't responding.

Do you want to close it?

- ➢ The user might then decide to quit your application and uninstall it if they are unhappy. So,

No component should perform long or blocking operations

(e.g. I/O operations, network access, computation loops)

Processes and Threads (Cont)

- Threads (Cont)
  - ✓ Solution
- Use a background thread to do the task (e.g. I/O operations, net work access, computation loops)

  - ✓ Consequence

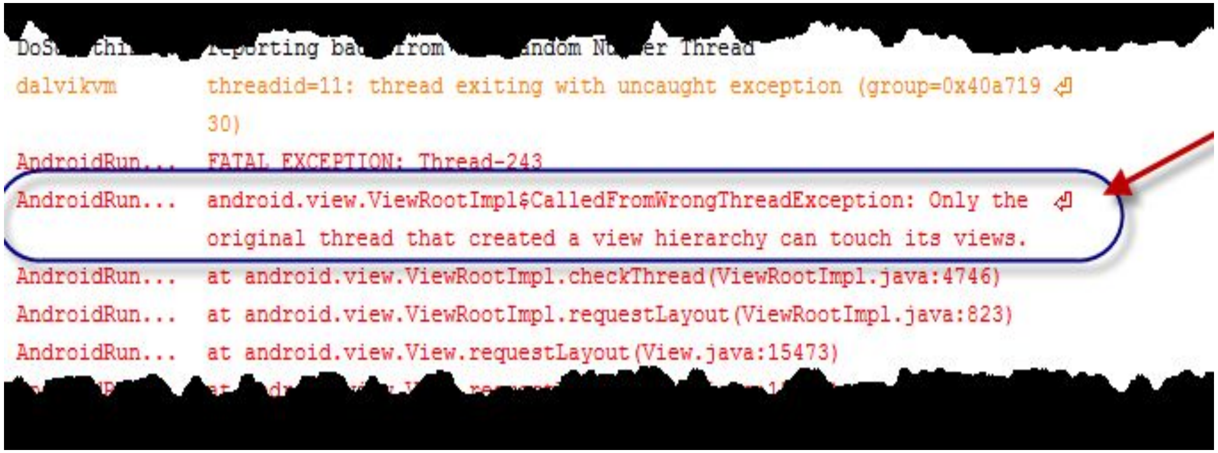
    Background thread and UI thread are running concurrently and may have race conditions if they modify UI simultaneously (e.g., UI switches to a different orientation)

Problem:

The Andoid UI toolkit is **not thread-safe**

Processes and Threads (Cont)

- Threads
  - ✓ The Andoid UI toolkit is **not thread-safe**. So, you must not manipulate your UI from a worker thread—you must do all manipulation to your user interface from the UI thread.



```
DoS   thi     reporting ba   from       andom Nu   er Thread
dalvikvm        threadid=11: thread exiting with uncaught exception (group=0x40a719 ⏎
                30)
AndroidRun...   FATAL EXCEPTION: Thread-243
AndroidRun...   android.view.ViewRootImpl$CalledFromWrongThreadException: Only the ⏎
                original thread that created a view hierarchy can touch its views.
AndroidRun...   at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:4746)
AndroidRun...   at android.view.ViewRootImpl.requestLayout(ViewRootImpl.java:823)
AndroidRun...   at android.view.View.requestLayout(View.java:15473)
```

Do not access the Android UI toolkit from outside the UI thread

Processes and Threads (Cont)

No component should perform long or blocking operations (such as networking operations or computation loops) when called by the system, since this will block any other components also in the process.

- Since the user interface must always be quick to respond to user actions, the thread that hosts an activity should not also host time-consuming operations like network downloads.
- Anything that may not be completed quickly should be assigned to a different thread (Spawn separate threads for long operations (background work)).

That's Multithreading in Android

Processes and Threads (Cont)

- Threads (Cont)
  - ✓ Anything that may not be completed quickly should be assigned to a different thread.
    - ➢ Threads are created in code using standard Java Thread objects.
  - ✓ Some convenience classes Android provides for managing threads:
    - ➢ Looper for running a message loop within a thread
    - ➢ Handler for processing messages
    - ➢ HandlerThread for providing a handy way for starting a new thread that has a looper

Event-driven programming

- Worker threads should communicate with the UI thread. Which communication Model will they use.
- Some of the goals of threads can be met by using an event-driven programming model.
- An event-driven program executes a sequence of events. The program consists of a set of handlers for those events.

  e.g., Unix signals
- The program executes sequentially (no concurrency). But the int erleaving of handler executions is determined by the event order.
- Pure event-driven programming can simplify management of inherently concurrent activities.

  E.g., I/O, user interaction, children, client requests

Android app: main event loop

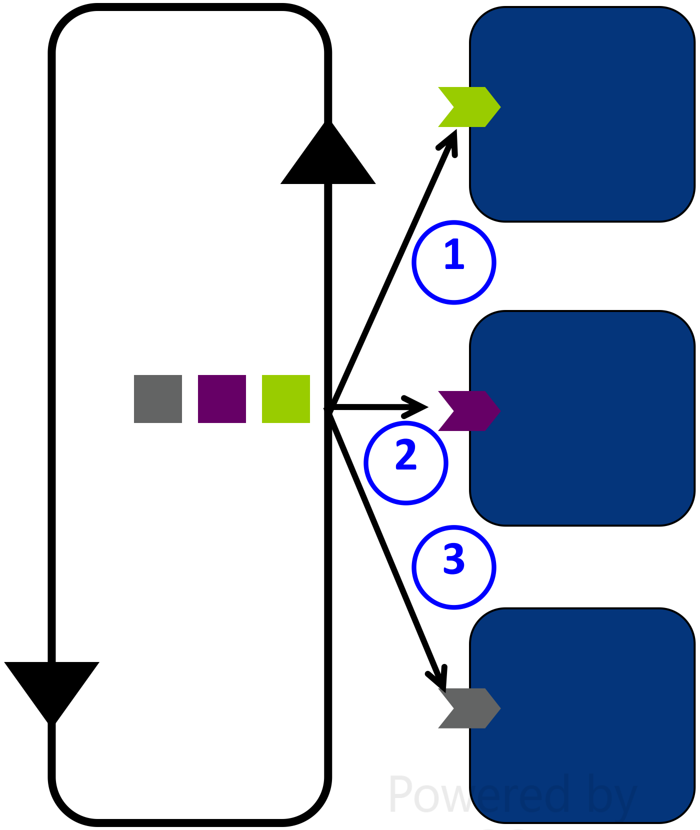
- The main thread of an Android app is called the Activity Thread.
- It receives a sequence of events and invokes their handlers.
- Also called the "UI thread" because it receives all User Interface events.

  screen taps, clicks, swipes, etc.

  All UI calls must be made by the UI thread: the UI lib is not thread-safe.

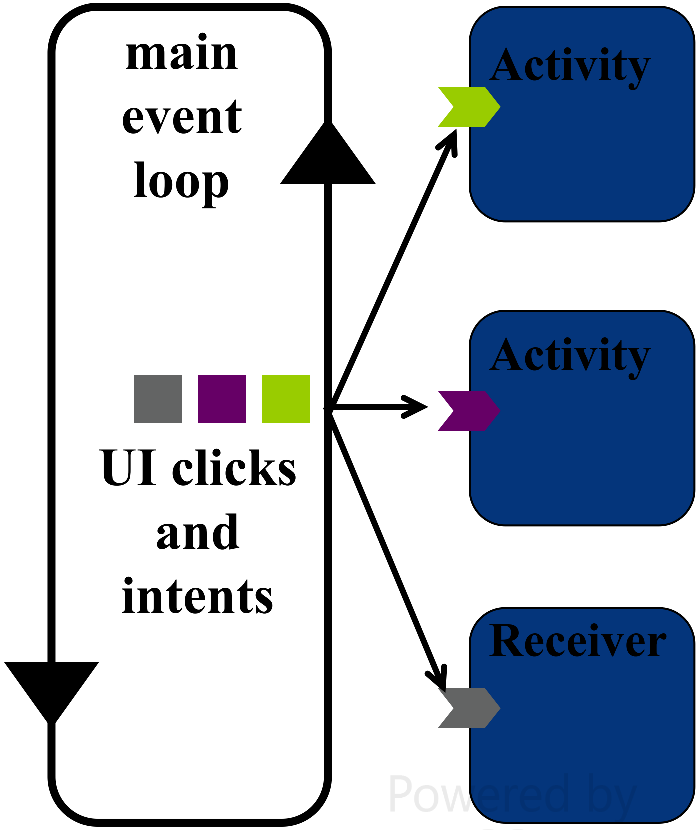
  MS-Windows apps are similar.
- The UI thread must not block!

  If it blocks, then the app becomes u nresponsive to user input: bad.



Android event loop: a closer look

- The main thread delivers UI events and intents to Activity components.
- It also delivers events (broadcast intents) to Receiver components.
- Handlers defined for these components must not block.
- The handlers execute serially in event arrival order.
- Note: Service and ContentProvider components receive invocations from other apps (i.e., they are servers).
- These invocations run on different threads.



Dispatch events by invoking component-defined handlers.

Event-driven programming

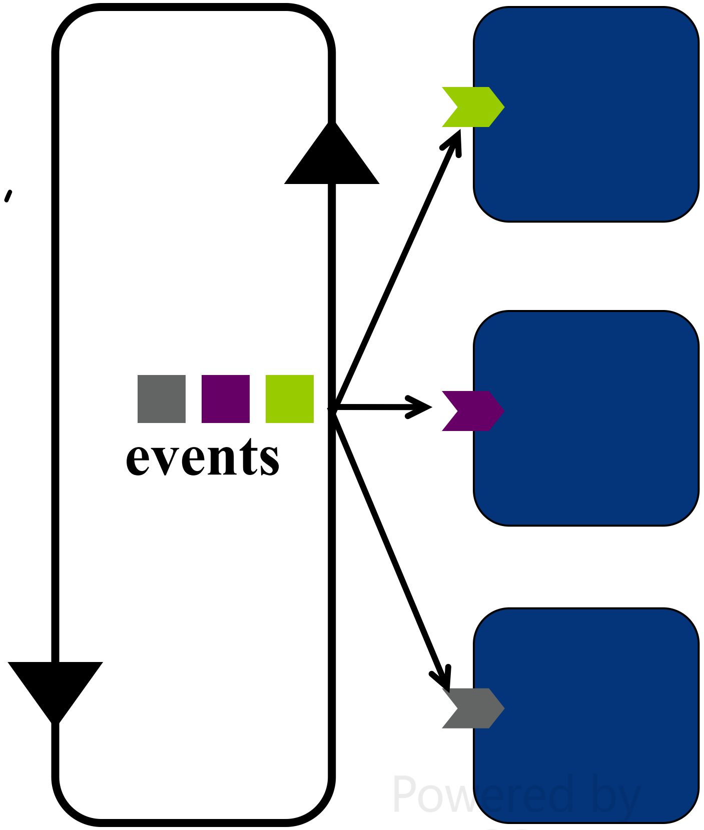
- This "design pattern" is called event-driven (event-based) programming.
- In its pure form the thread never blocks, except to wait for the next event, whatever it is.
- We can think of the program as a set of handlers: the system upcalls a handler to dispatch each event.
- Note: here we are using the term "event" to refer to any notification:

  arriving input

  asynchronous I/O completion

  subscribed events

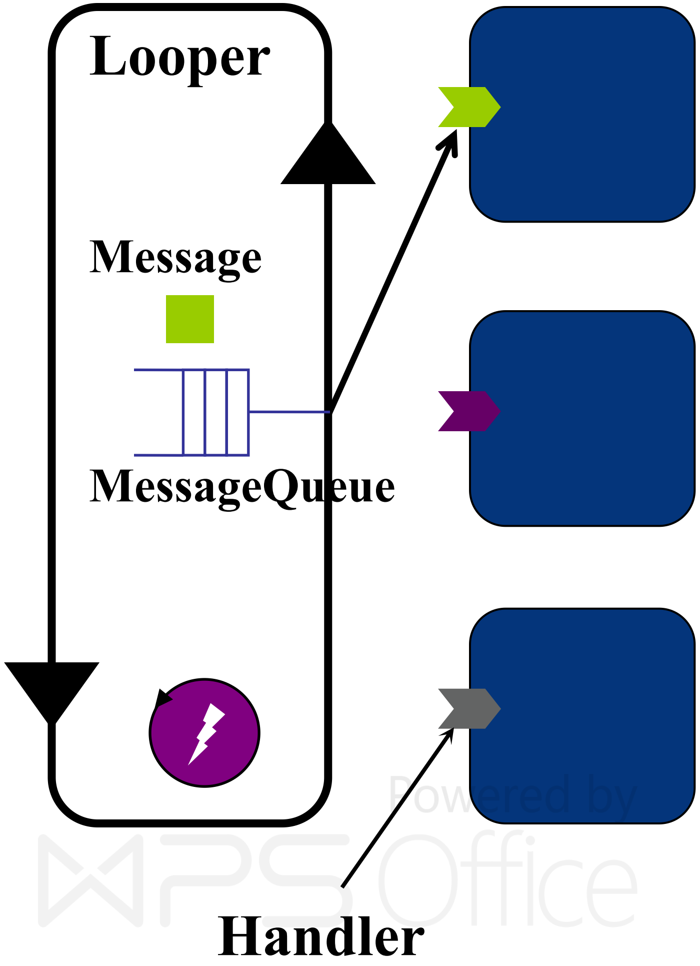
  child stop/exit, "signals", etc.



Dispatch events by invoking handlers (upcalls).

Android event classes: some details

- Android defines a set of classes for event-driven programming in conjunction with threads.
- A thread may have at most one Looper bound to a MessageQueue.
- Each Looper has exactly one thread and exactly one MessageQueue.
- The Looper has an interface to register Handlers.
- There may be any number of Handlers registered per Looper.
- These classes are used for the UI thread, but have other uses as well.



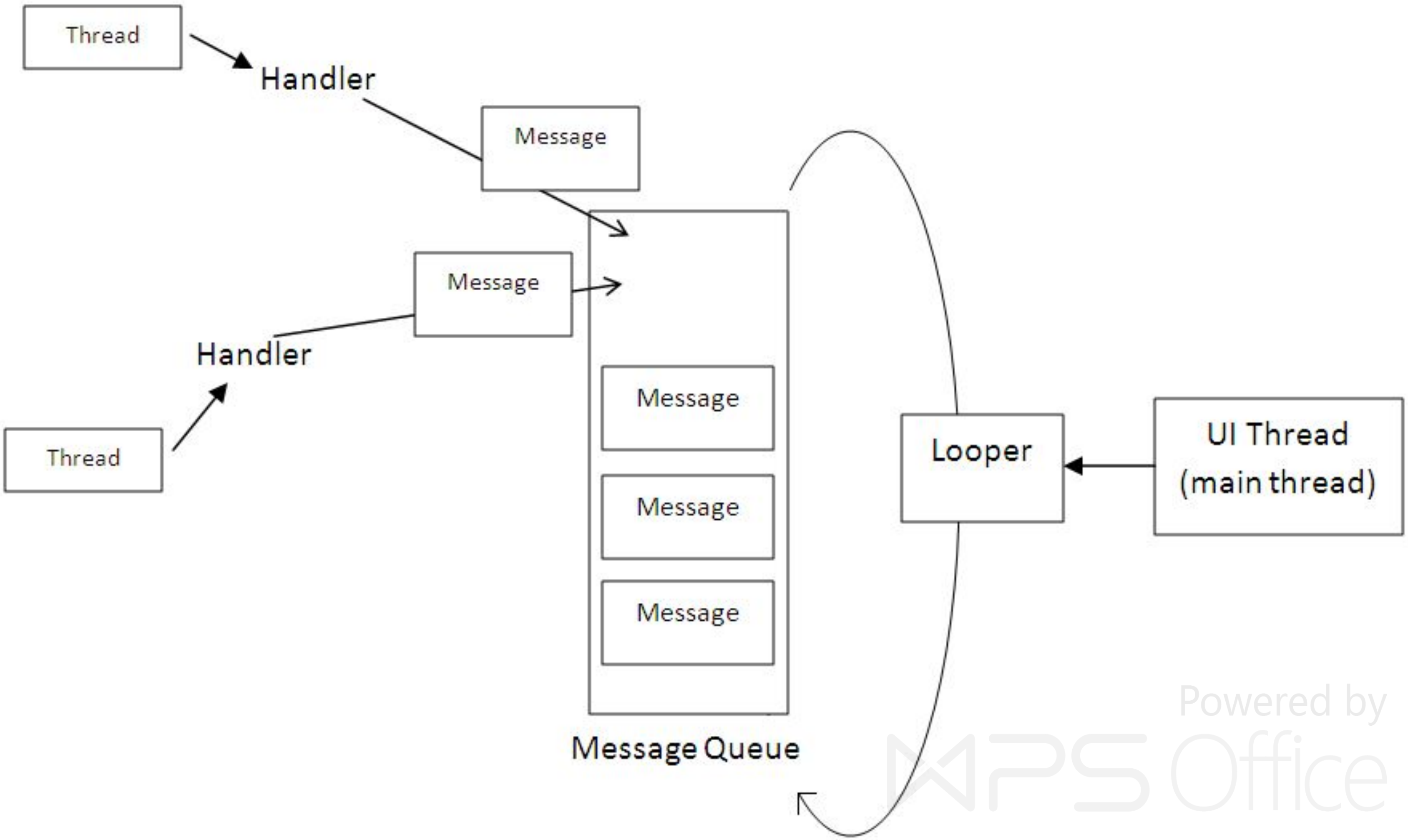
[These Android details are provided for completeness.]

Android Handler

- Android’s mechanism to send and process Message and Runnable objects associated with a thread's MessageQueue.

- Each Handler instance is associated with a single thread and that thread's message queue
  - A handler is bound to the thread / message queue of the thread that creates it
  - from that point on, it will deliver messages and runnables to that message queue
  - That thread processes msgs

Android Handler



Using Handler: Examples

- There are two main uses for a Handler

  to schedule messages and ***runnables*** to be executed as some point in the future

  postDelayed(*Runnable*, delayMillis)

  to enqueue an action to be performed on a different thread than your own.

  post(*Runnable*)

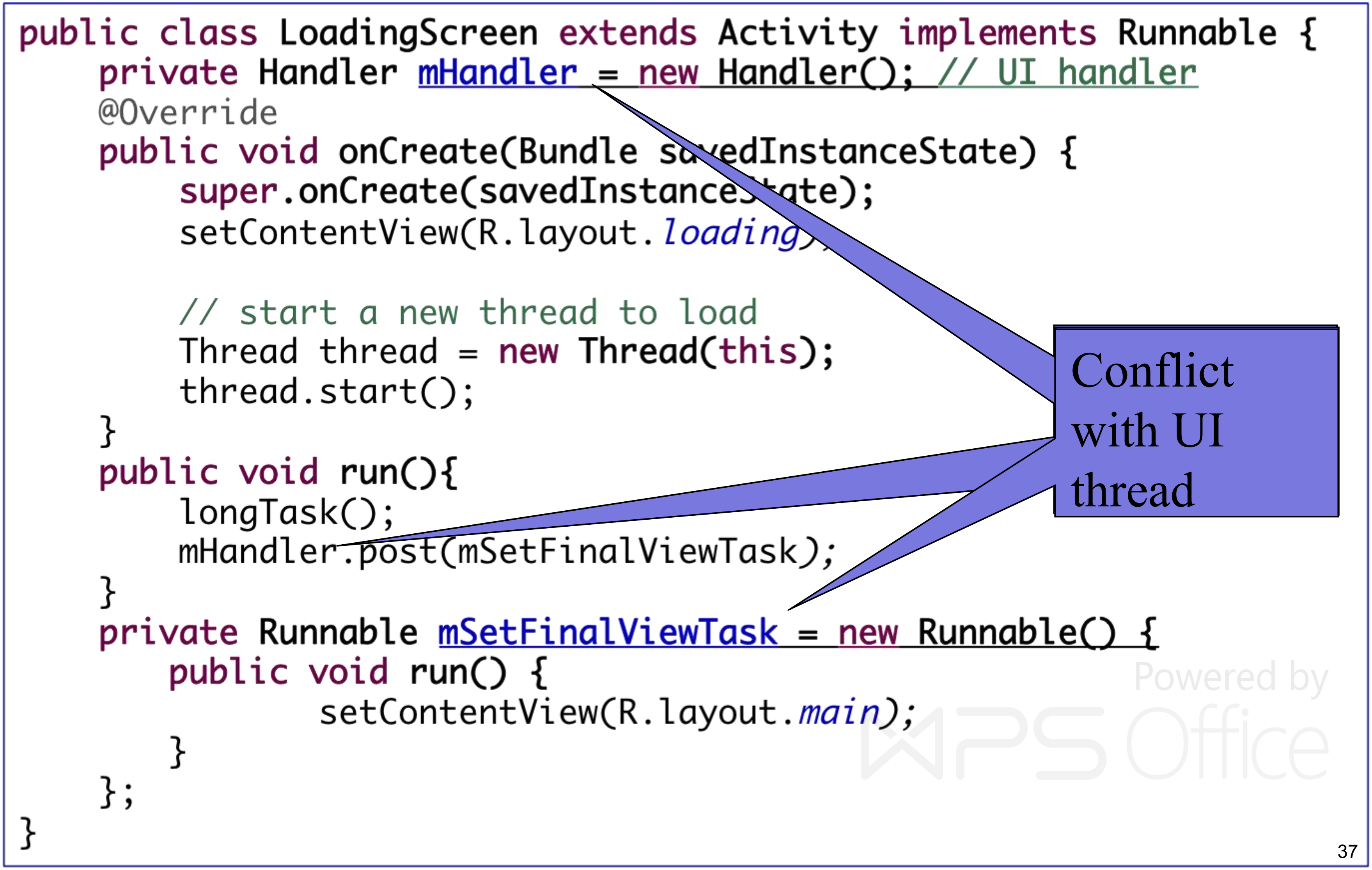
Android Handler

```
public class MyActivity extends Activity {
    [ . . . ]
    // Need handler for callbacks to the UI thread
    final Handler mHandler = new Handler();
    // Create runnable task to give to UI thread
    final Runnable mUpdateResultsTask = new Runnable() {
        public void run() {
            updateResultsInUi();
        }
    };
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        [ . . . ]
    }
```

Android Handler

```
protected void startLongRunningOperation() {
    // Fire off a thread to do some work that we shouldn't do directly in the UI thread
    Thread t = new Thread() {
      public void run() {
        mResults = doSomethingExpensive();
        mHandler.post(mUpdateResultsTask);
      }
    };
    t.start();
  }
  private void updateResultsInUi() {
    // Back in the UI thread -- update our UI elements based on the data in mResults
    [ . . . ]
  }
}
```

Example: Fixing LoadingScreen

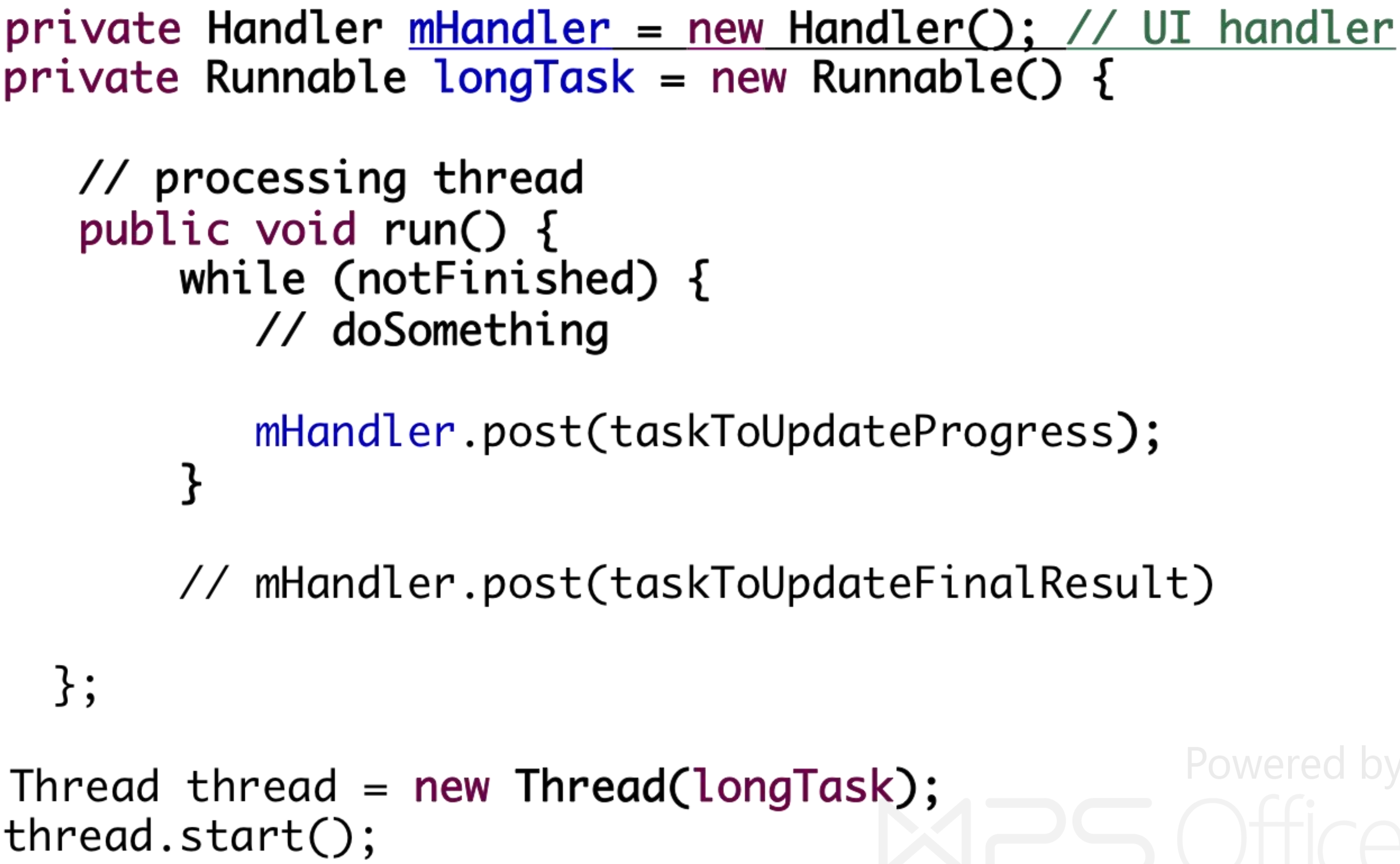


```
public class LoadingScreen extends Activity implements Runnable {
    private Handler mHandler = new Handler(); // UI handler
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.loading);

        // start a new thread to load
        Thread thread = new Thread(this);
        thread.start();
    }
    public void run(){
        longTask();
        mHandler.post(mSetFinalViewTask);
    }
    private Runnable mSetFinalViewTask = new Runnable() {
        public void run() {
            setContentView(R.layout.main);
        }
    };
}
```

Conflict with UI thread

Common Pattern



```
private Handler mHandler = new Handler(); // UI handler
private Runnable longTask = new Runnable() {

   // processing thread
   public void run() {
       while (notFinished) {
          // doSomething

          mHandler.post(taskToUpdateProgress);
       }

       // mHandler.post(taskToUpdateFinalResult)

  };

Thread thread = new Thread(longTask);
thread.start();
```

Concuruncy in Android

Android Application Model, Processes, UI Thread

and

Handlers

Android Application Model, Processes, UI Thread and Handlers

Supports de présentation

http://moss.csc.ncsu.edu/~mueller/g1/
http://db.cs.duke.edu/courses/cps110/fall12/slides/
http://zoo.cs.yale.edu/classes/cs434/cs434-2012-fall/lectures/

Université Mohammed V
FACULTE DES SCIENCES
RABAT / FSR
Département informatique

Master IAO
Master II–Semestre 3
Cours

Mobile & Cloud Computing

Pr. REDA Oussama Mohammed

2015/2016