



**LA TROBE**  
UNIVERSITY

All kinds of clever

# CSE2MAD

## Mobile Application Development Lecture 4



# Outline

- Recap last week's lecture + lab
  - Android App Core Components – Intents / Broadcast Receiver
  - Android Process and Thread Handling
-

# Recap

## Android Application Anatomy

### Activities

1. Provides **User Interface**
2. Usually represents a **Single Screen**
3. Can contain one/more **Views**
4. **Extends** the **Activity** Base class

### Services

1. **No User Interface**
2. Runs in **Background**
3. **Extends** the **Service** Base Class

Application= Set of Android Components

### Intent/Broadcast Receiver


1. Receives and Reacts to broadcast **Intents**
2. No UI but **can start** an Activity
3. **Extends** the **BroadcastReceiver** Base Class

### Content Provider

1. Makes application data available to other apps
2. Data stored in SQLite database
3. **Extends** the **ContentProvider** Base class

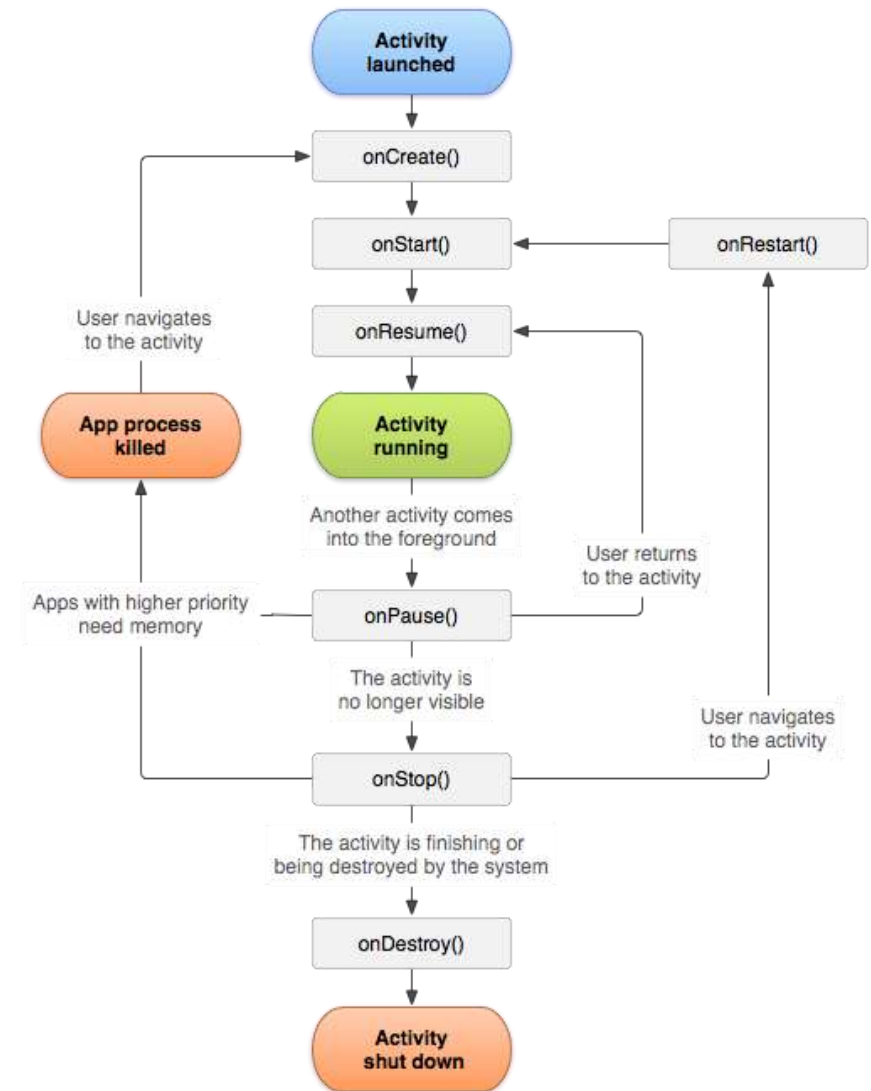
# Activities

A thick red horizontal bar spans the width of the slide, with a vertical red bar extending downwards from its right end.

- Screens of an Android Application
  - Have both a XML layout and a Java Class
  - Create class members to UI Objects
  - Initialise them in onCreate
  - Set listeners to events raised by UI Objects
  - Implement additional methods
- 
- A thin grey horizontal bar spans the width of the slide at the bottom.

# android.app.Activity

```
public class Activity extends ApplicationContext {  
    protected void onCreate(Bundle savedInstanceState);  
    protected void onStart();  
    protected void onRestart();  
    protected void onResume();  
    protected void onPause();  
    protected void onStop();  
    protected void onDestroy();  
}
```



# Activity Lifecycle Code Demonstration

<https://github.com/latrobe-cs-educator/CSE2MAD> Lecture 4 ActivityLifeCycleDemo



# Intents / Broadcast Receivers

Intent: Objects that describe the task of connecting components of Android applications

E.g. use an intent to start one Activity from another

```
public void goToSwitchActivity(View view) {  
    //EXPLICIT intent example  
    Intent intent = new Intent( packageContext: this, SwitchButtonExample.class);  
    startActivity(intent);  
}
```

- Explicit : The intent has a specified to / from
- Implicit : The intent has no directly specified destination

# Broadcasts

A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging.

You can deliver a broadcast to other apps by passing an Intent to `sendBroadcast()`, `sendOrderedBroadcast()`, or `sendStickyBroadcast()`.





# Using Intents to Broadcast Events

- To send structured messages at system-level, across process boundaries.
- Each application can have its own Broadcast Receivers to listen for, and respond to, these Broadcast Intents.

## **Example of a broadcast intent < Android 3.0:**

```
Intent intent = new Intent();  
intent.setAction("com.example.Broadcast");  
intent.putExtra("MyData", 1000);  
sendBroadcast(intent);
```

# Using Intents to Broadcast Events

- To send structured messages at system-level, across process boundaries.
- Each application can have its own Broadcast Receivers to listen for, and respond to, these Broadcast Intents.

## Example of a broadcast intent >= Android 3.0

```
Intent intent = new Intent();  
intent.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);  
intent.setAction("com.example.Broadcast");  
intent.putExtra("MyData", 1000);  
sendBroadcast(intent);
```

# Create Intent Filters and Broadcast Receivers

**Intent filters** used by app components to declare actions and data they support.

To register an activity or service as an Intent handler, **add intent-filter tag to its manifest** using:

- **Action:** name of action being serviced, at least one for each intent filter.
- **Category:** specify which circumstances the action should be serviced.
  - Can have more than one in a intent filter,
  - Can be self defined or provided by Android such as: ALTERNATIVE, SELECTED\_ALTERNATIVE, BROWSABLE, DEFAULT, HOME, LAUNCHER.

# Create Intent Filters and Broadcast Receivers

- **Data:** which data types your component can act on. Can have multiple, each can use any combination of the following attributes:
  - Android:host - host name (e.g., **google.com**)
  - Android:mimetype – type of data (<**type**  
**android:value="vnd.android.cursor.dir/\*" />**)
  - Android:path – path for URI (e.g., **/transport/boats/**)
  - Android:port – ports of host
  - Android:scheme – scheme such as **content** or **http**

# Broadcast and Listening

- Broadcast:

```
sendBroadcast(intent);
```

- Self built Receiver: need to be created and registered in code or in manifest (manifest Receiver).
- Activities having manifest Receiver will be started automatically when a matching intent is broadcast -> save resources.
- Use Intent Filter to choose actions and data your Receiver is listening for.

# Broadcast and Listening

Create Receiver for listening:

- Extend BroadcastReceiver class,
- Override onReceive event handler.

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //TODO: React to the Intent received.
    }
}
```

# Register Receivers

In code:

```
private IntentFilter filter = new IntentFilter(LifeformDetectedReceiver.NEW_LIFEFORM);
private LifeformDetectedReceiver receiver = new LifeformDetectedReceiver();
@Override
public void onResume() {
    super.onResume();
    // Register the broadcast receiver. registerReceiver(receiver, filter);
}

@Override
public void onPause() {
    // Unregister the receiver
    unregisterReceiver(receiver); super.onPause();
}
```



# Register Receivers (cont)

In manifest (alternate to the previous way):

```
<receiver android:name=".LifeformDetectedReceiver">  
  <intent-filter>  
    <action android:name="com.paad.alien.action.NEW_LIFEFORM"/>  
  </intent-filter>  
</receiver>
```

Note: If your app targets API level 26 or higher, you cannot use the manifest to declare a receiver for implicit broadcasts (broadcasts that do not target your app specifically), except for a few implicit broadcasts that are exempted from that restriction. In most cases, you can use scheduled jobs instead.

# Intent Filters and Broadcast Receivers Code Demonstration

<https://github.com/latrobe-cs-educator/CSE2MAD> Lecture 4 SendBroadcast

# Intent filter example

Intent Filter for activity that can perform the SHOW\_DAMAGE action as either a primary or an alternative action based on its mime(media) type:

```
<intent-filter>  
  <action android:name="com.paad.earthquake.intent.action.SHOW_DAMAGE"/>  
  <category android:name="android.intent.category.DEFAULT"/>  
  <category android:name="android.intent.category.SELECTED_ALTERNATIVE"/>  
  <data android:mimeType="vnd.earthquake.cursor.item/*"/>  
</intent-filter>
```

# Broadcast Ordered Intents

When the order of receiving intent is important to receiver -> use ordered intents.

- To broadcast: Set permission and send.

```
String requiredPermission = "com.paad.MY_BROADCAST_PERMISSION";  
sendOrderedBroadcast(intent, requiredPermission);
```

- Receiver: only ones having permission can get the intents, higher permission value, higher priority.

```
<receiver  
    android:name=".MyOrderedReceiver"  
    android:permission="com.paad.MY_BROADCAST_PERMISSION">  
    <intent-filter android:priority="100">  
        <action android:name="com.paad.action.ORDERED_BROADCAST" />  
    </intent-filter>  
</receiver>
```

Highest priority

# Broadcast Sticky Intents

Is Broadcast intents that:

- **Persist the values of last broadcast.**
- Return the values when a new receiver is registered to receive the broadcast.
- Normally used for device state broadcasts (battery, docking state .etc).

```
IntentFilter battery = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);  
Intent currentBatteryCharge = registerReceiver(null, battery);
```

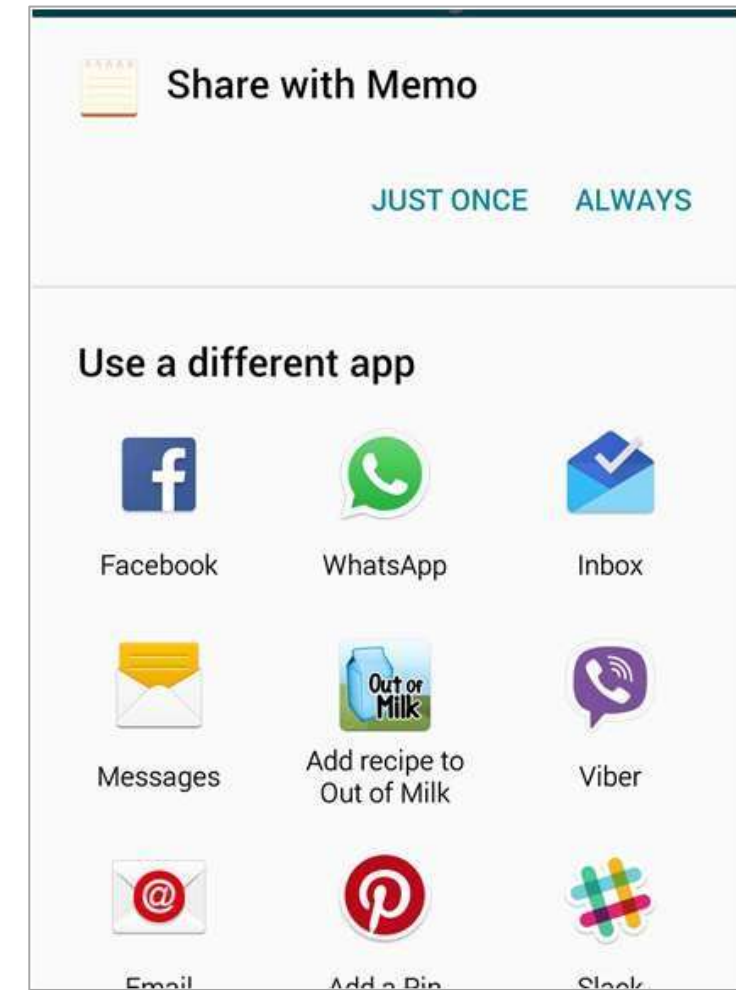
- To broadcast: Need BROADCAST\_STICKY uses permission and call

```
sendStickyBroadcast(intent);  
removeStickyBroadcast(intent);
```

# Intent Resolution

Which activity to start:

- If more than one intent handlers are matched for one implicit intent with the same priority, all matching possibilities are offered to the user
- For Broadcast Receivers, each matching Receiver will receive the broadcast Intent.
- Pass responsibility for the next best activity to handle:



# Monitoring Device State Using Broadcast Intents, Example

- Using Broadcast Intents to listen for device state such as Battery Change.
- Using sticky Intent -> retrieve status any time and don't need to implement a Broadcast Receiver.

```
IntentFilter batIntentFilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);  
Intent battery = context.registerReceiver(null, batIntentFilter); int status =  
battery.getIntExtra(BatteryManager.EXTRA_STATUS, -1);  
boolean isCharging =  
status == BatteryManager.BATTERY_STATUS_CHARGING || status ==  
BatteryManager.BATTERY_STATUS_FULL;
```

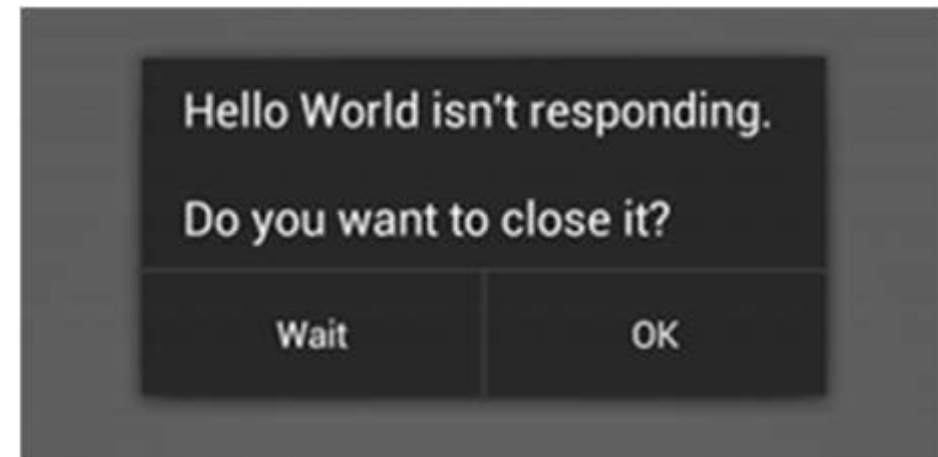


# Working in the Background

- Avoid ANR situations (Application Not Responding)

Use

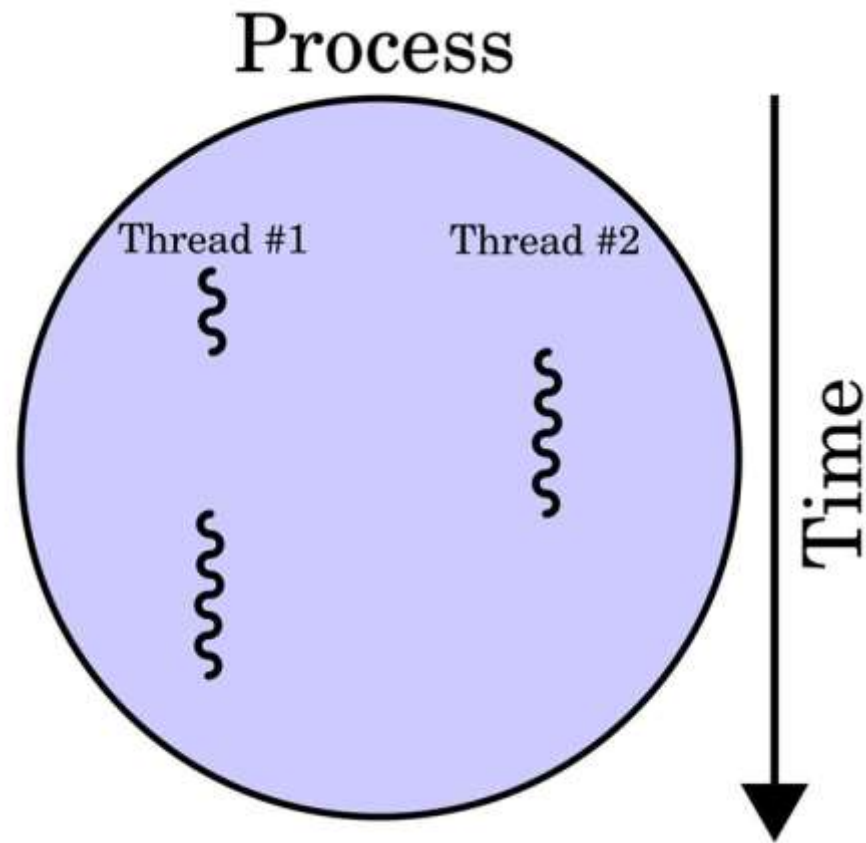
- Handlers
- WorkManager
- Services
- Intent Service



**The worst thing that can happen to your app's responsiveness is an "Application Not Responding" (ANR) dialog.**

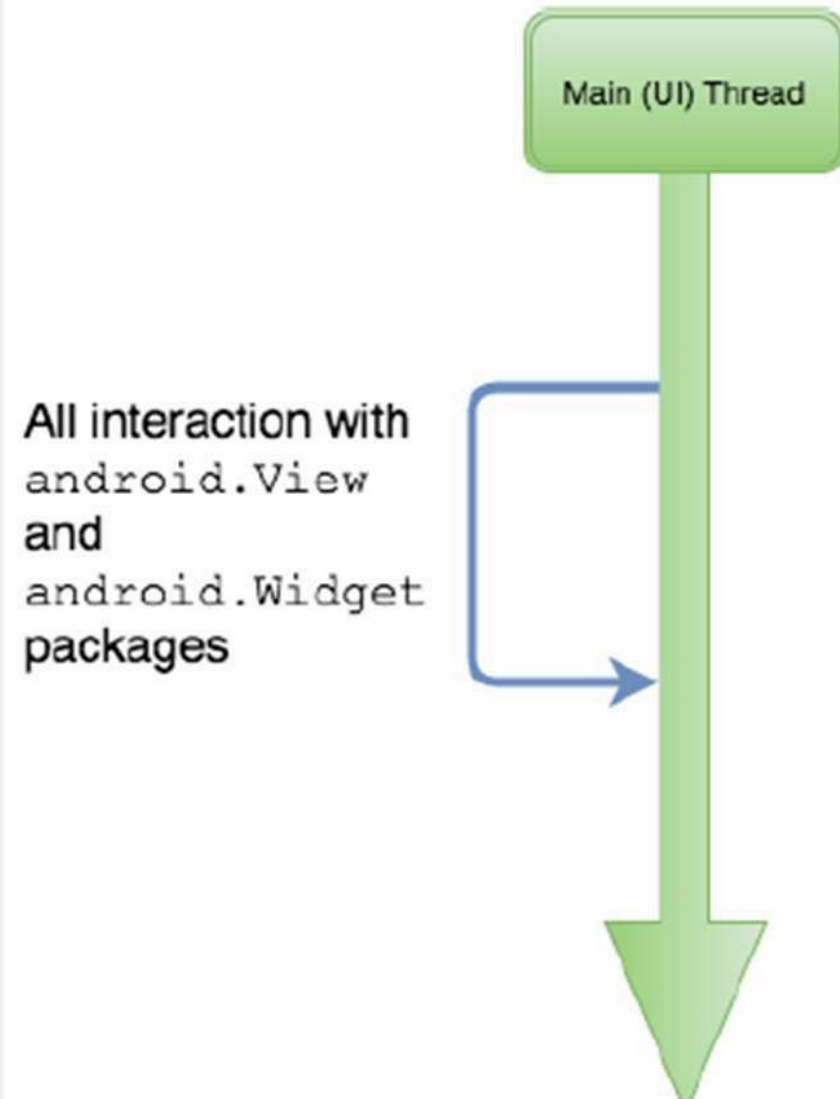
**Generally, 100 to 200ms is the threshold beyond which users will perceive slowness in an application**

# Working in the Background



- **Process:** A process has a self-contained execution environment. A process generally has a complete, private set of basic run-time resources; in particular, each process has its own memory space.
- **Thread:** Lightweight version of a process similar attributes as above but less resource intensive. Threads exist within a process. Share a process's resources.

# Working in the Background



Android creates a thread called “main” (often referred to as the UI thread) for each application when it starts.

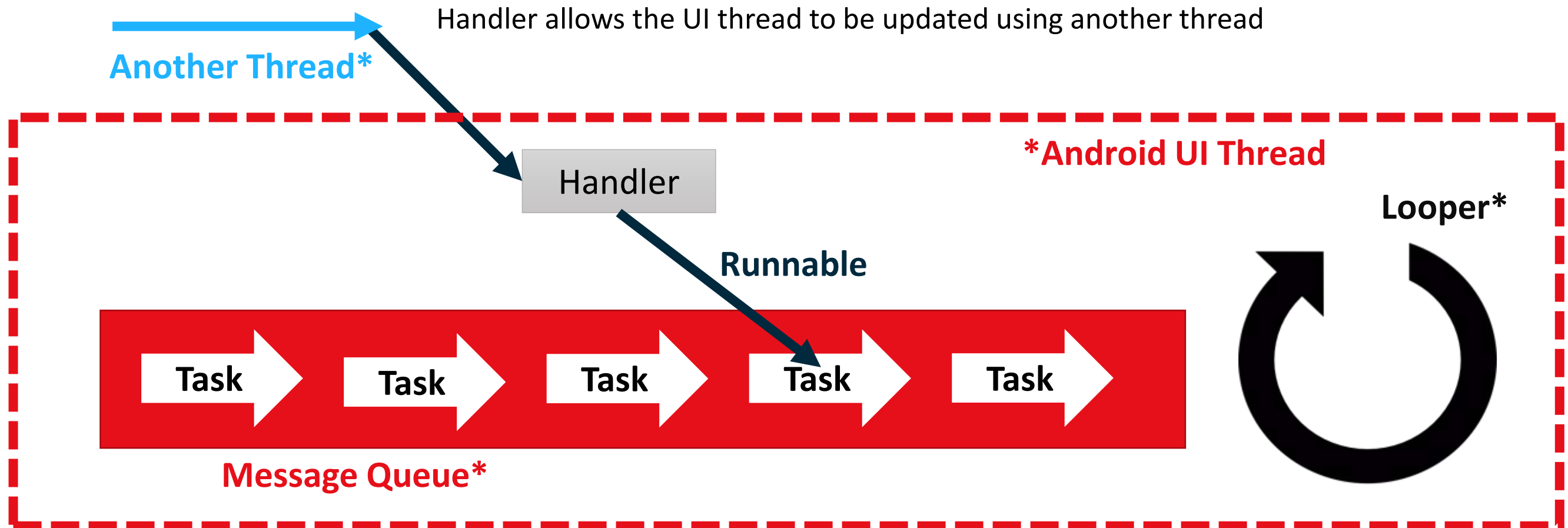
- The Android operating system does not create a separate thread for each instance of a component
- Methods that respond to system callbacks (e.g. key and touch events) always run on the UI thread

Source: <http://developer.android.com/guide/components/processes-and-threads.html#Threads>

# Handler

- Handlers are used for managing threads in Android.
- A Handler object receives messages and runs code to handle the messages.
- It provides a channel to send data to this thread.
- Normally, you create a Handler for a new thread, but you can also create a Handler that's connected to an existing thread.
- When you connect a Handler to your UI thread, the code that handles messages runs on the UI thread.

# Main UI Thread and Handler



# Creating Instance of Handler

```
Handler handler = new  
Handler(Looper.getMainLooper()) {  
    @Override  
    public void handleMessage(Message msg) {  
        myTextView.setText("Button Pressed");  
    }  
};
```

```
.....  
Message msg = handler.obtainMessage();  
Bundle bundle = new Bundle();  
bundle.putString("myKey", "Hello handler");  
msg.setData(bundle);  
handlerMsg.sendMessage(msg);  
.....
```

Effect a UI change via a message

```
Handler handlerMsg = new  
Handler(Looper.getMainLooper()) {  
    @Override  
    public void handleMessage(Message msg) {  
        Bundle bundle = msg.getData(); String string =  
        bundle.getString("myKey");  
        myTextView.setText(string);  
    }  
};
```

Send a message using the handler



# Handler & Thread Code Demonstration

<https://github.com/latrobe-cs-educator/CSE2MAD> Lecture 4 HandlerDemo



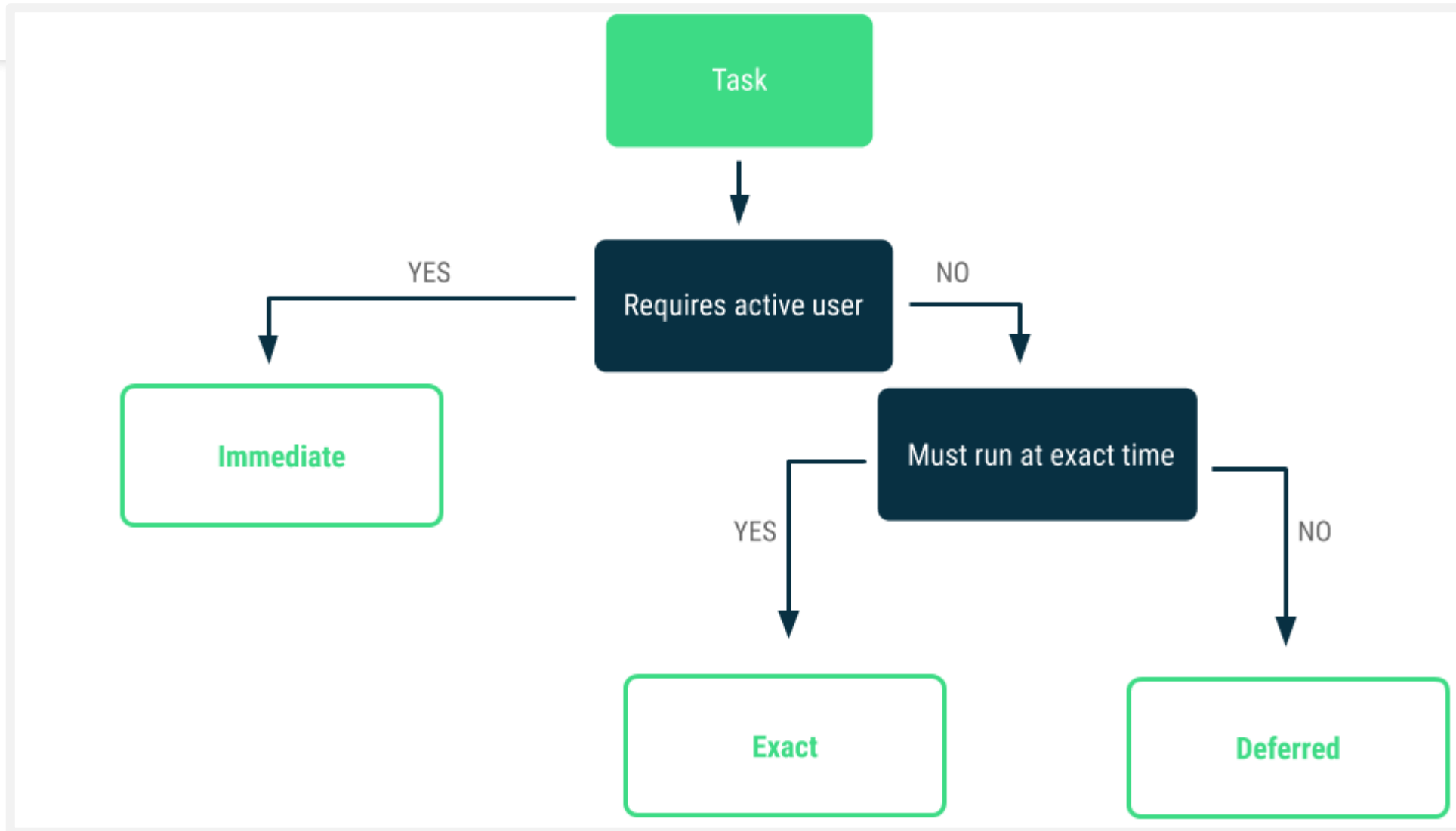
# What is Background processing?

In general, any task that takes more than a few milliseconds should be delegated to a background thread. Common long-running tasks include things like decoding a bitmap, accessing storage, working on a machine learning (ML) model, or performing network requests.

Background tasks fall into one of the following main categories:

- Immediate
- Deferred
- Exact

# Categories of background tasks



# Immediate tasks

- You can create additional background threads to handle long-running operations while the main thread continues to handle UI updates.
- For tasks that should be executed immediately and need continued processing, even if the user puts the application in background or the device restarts, it is recommend to use WorkManager and its support for long-running tasks.
- In specific cases, such as with media playback or active navigation, you might want to use foreground Services directly.

# Deferred tasks

- Every task that is not directly connected to a user interaction and can run at any time in the future can be deferred. The recommended solution for deferred tasks is WorkManager.
- WorkManager makes it easy to schedule deferrable, asynchronous tasks that are expected to run even if the app exits or the device restarts.

# Exact tasks

- A task that needs to be executed at an exact point in time can use `AlarmManager`.
- Alarms (based on the `AlarmManager` class) give you a way to perform time-based operations outside the lifetime of your application. For example, you could use an alarm to initiate a long-running operation, such as starting a service once a day to download a weather forecast.

# Whats next?

- View the Agile Part 2 video in the Lecture Tab on LMS