# G54MDP
# Mobile Device Programming

Lecture 5 – Android Application Components - Activities

# Android Components

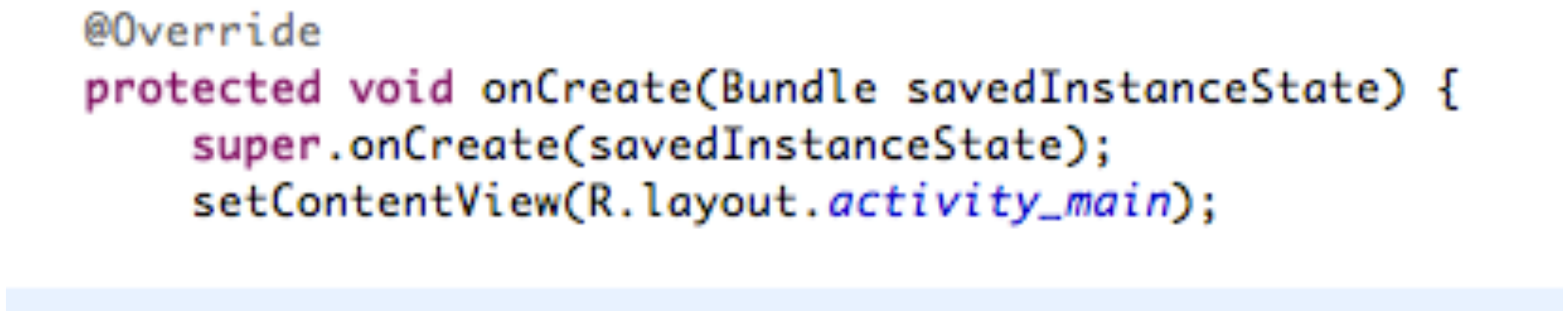
- Activities
  - UI components
- Services
  - Mechanism for doing something long-running in the background
- Broadcast Receivers
  - Respond to broadcast messages from the OS / other apps
- Content Providers
  - Make data available to / make use of data from other apps
  - No access to the file system
    - SD Card

# Activities

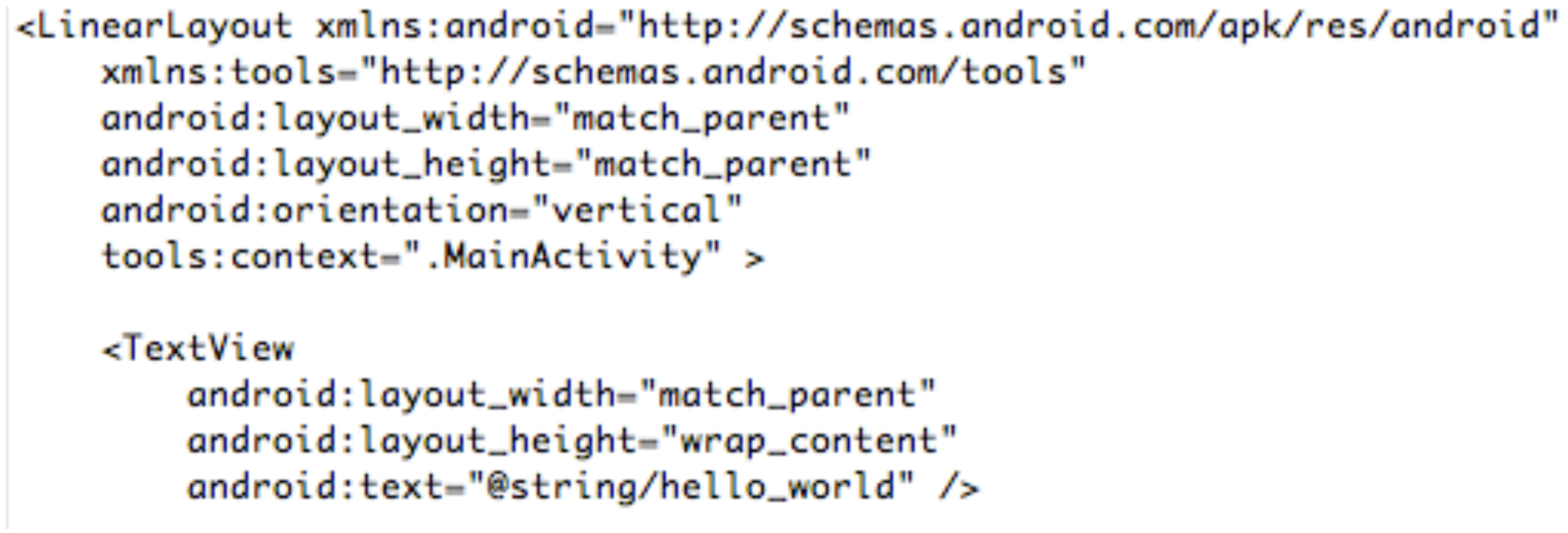
- Sub-classes of **android.app.Activity**
- Presents a visual UI
- Each Activity has its own “window”
  – Only one “window” on screen at once
- UI layout – a “View”
  – Specified in a separate XML file
  – Constructed programmatically
  – Call setContentView() to display it
- Apps usually have several Activities
  – **Context**
    - An abstract class representing the current application environment

# Android UI

- An Activity has a window associated with it
- Usually full screen
  - Can hover over another activity
  - Can be transparent
- Within the window there is a hierarchy of View objects
- Set with setContentView()
- Usually specified via an XML resource
  - /res/layout/mylayout.xml



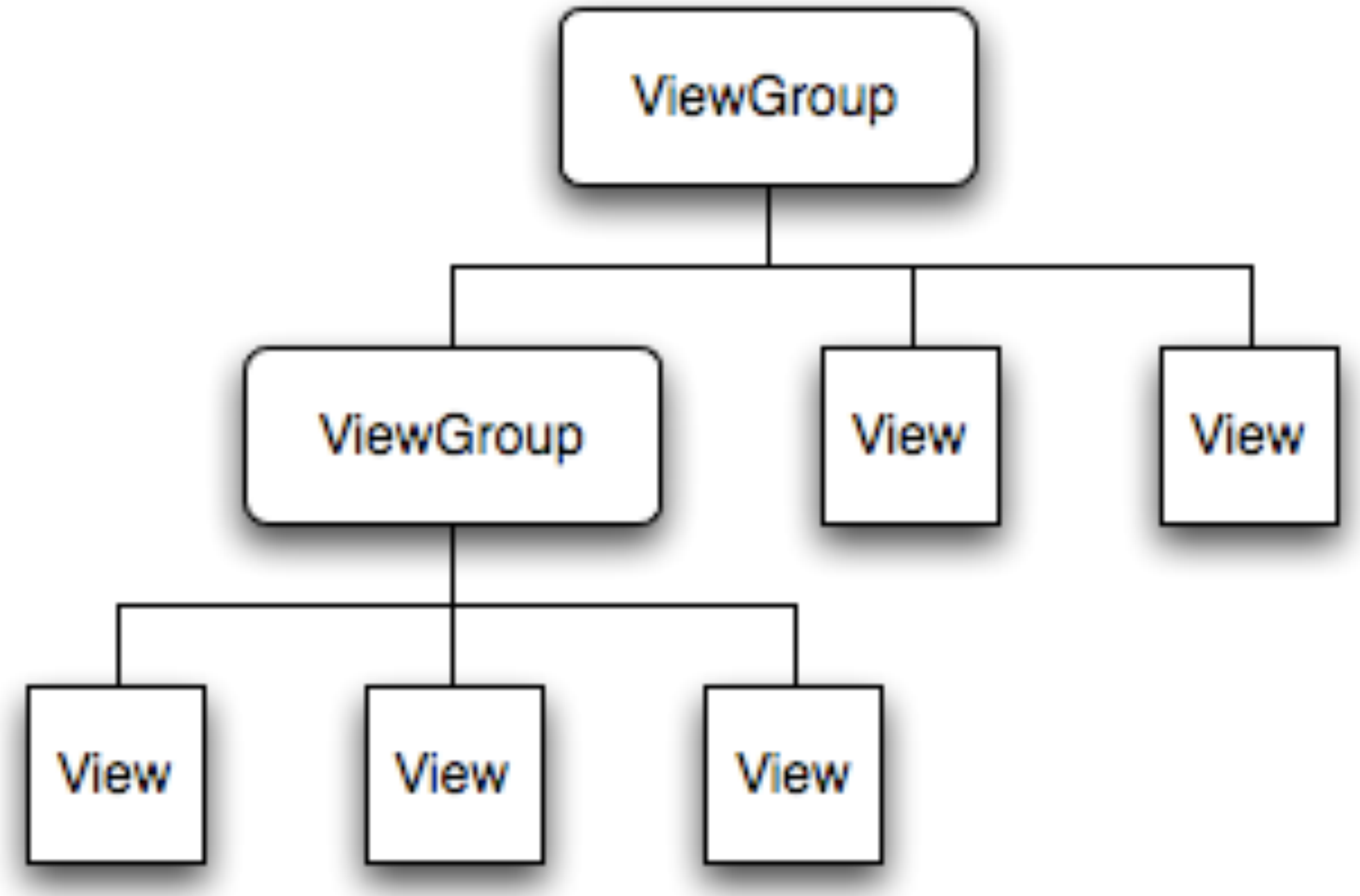
```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity" >

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello_world" />
```

## View Hierarchy

- Types of View subclasses
  - Those that display something
  - Those that do something (Widgets)
  - And ViewGroups which layout a collection of subviews
- Various layout types available — can specify in the XML resource


ViewGroup
ViewGroup
View
View
View
View
View

# ViewGroups - Layouts

- FrameLayout
  - Simplest, contains a single object
- LinearLayout
  - Aligns all children in a single direction, based on the orientation attribute
- TableLayout
  - Positions children into rows and columns
- RelativeLayout
  - Lets the child views specify their position relative to the parent view or to each other
- ScrollView
  - A vertically scrolling view, like FrameLayout only contains a single element (e.g. a LinearLayout)

# Views - Widgets

- A child View that the user can (optionally) interact with
  - Button (a button)
  - EditText (text entry)
  - CalendarViewer (a calendar widget)
  - ImageView (displays an image)
  - …
- Handle appropriate UI events
  - In code, register setOnClickListener()
  - In XML layout, set android:onClick parameter
- Properties / parameters
  - Set via XML at build-time
  - Equivalent set / get methods for modifying at run-time

```
android:text="@string/hello_world" />
.getText(...), .setText(…)
```

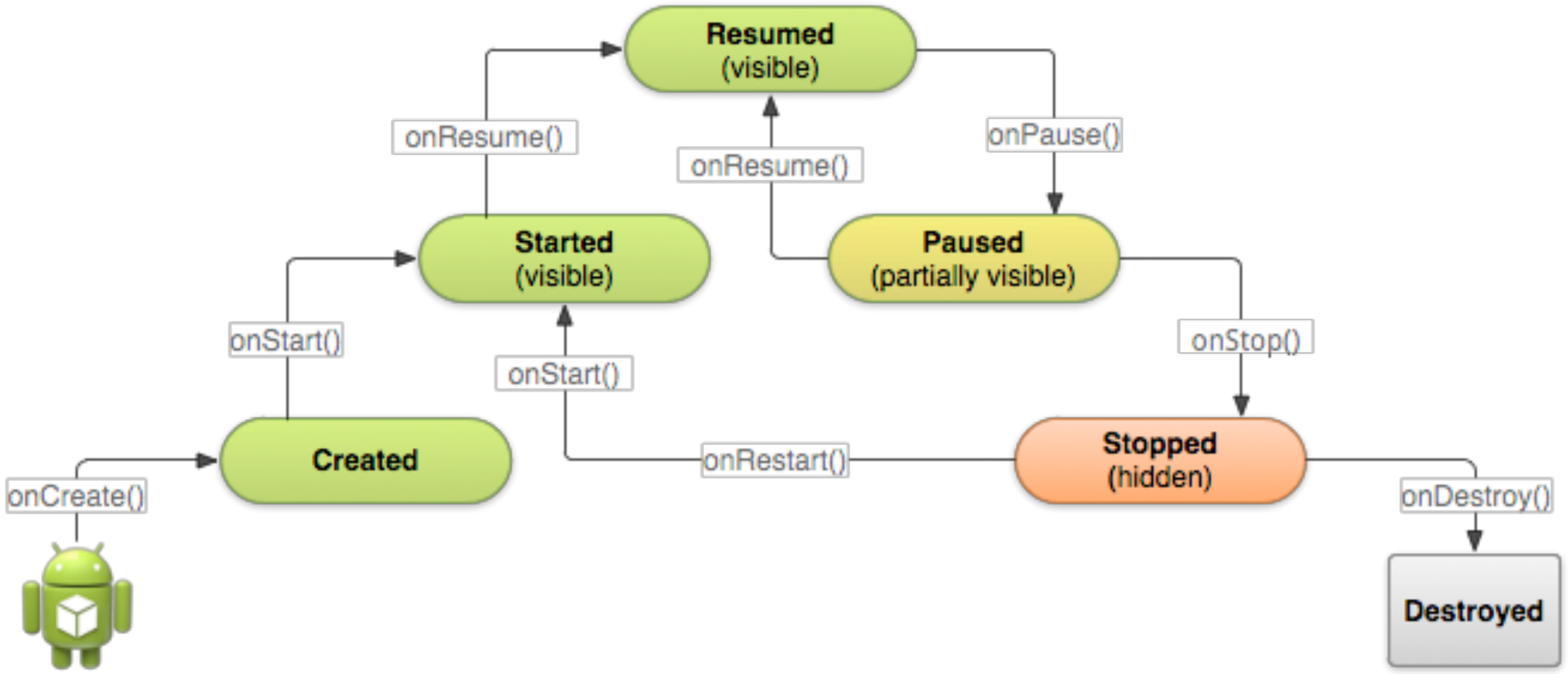
## Views - Parameters

- Parameters specify the details of particular Views
  - Width, height
    - Generally **not** in terms of absolute pixels
    - Relative to parent, percentages, offsets, margins
      - android:layout_width="match_parent”
      - android:layout_height="wrap_content"
  - ID
    - Used to generate a Java member variable we can refer to programmatically
      - android:id="@+id/my_button"
  - Methods
    - Used to automatically bind UI events to code
      - android:onClick=“myMethod”

# Manipulating Views

- We can alter the view hierarchy programmatically as the application runs
- ViewGroup provides methods
  - To add addView()
    - Need to either keep a reference to it or call setId() on the view so we can find it later
    - Or use references generated from layout XML for existing views
  - Or to remove removeView() children
    - Can add new buttons, layouts as required

## Let’s have a look…


Resumed
(visible)
onResume()
onResume()
onPause()
Started
(visible)
Paused
(partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped
(hidden)
onCreate()
onDestroy()
Destroyed

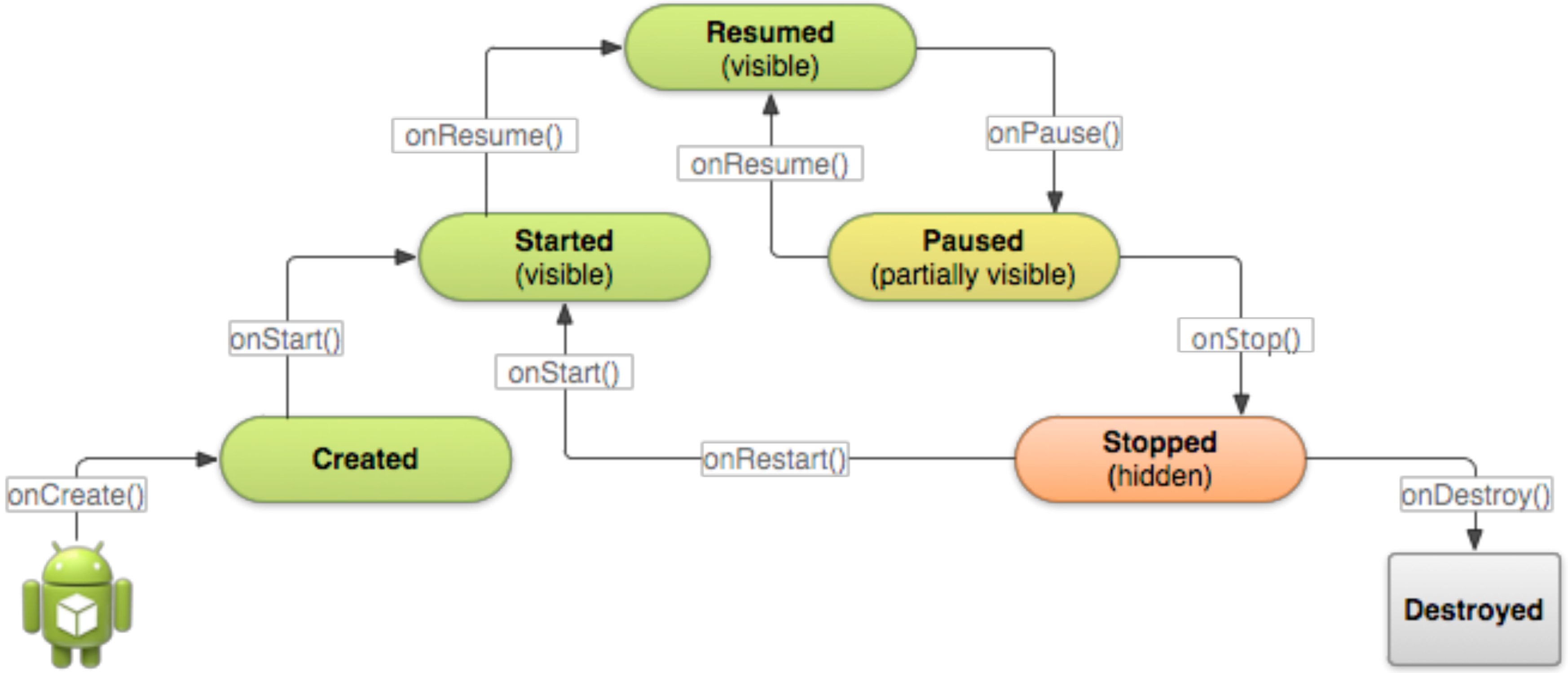
## Saving State

- Shouldn't rely on an Activity storing UI state
  - E.g. rotating the device
    - Destroys and recreates the activity
  - If you need to keep it, save it
- Before onStop() is called,  Android will call onSaveInstanceState()
  - To restore the **UI** to its previous state on restore
  - This allows you to save any **UI state** into a Bundle object
    - When the Activity is recreated, the Bundle is passed to onCreate() and onRestoreInstanceState()
    - Giving the Activity chance to restore its state
  - Save **other state** to more persistent storage
    - SQLite database / user preferences
    - More on this later

# Saving UI State

- Bundle
  - A collection of key/value pairs
  - Key
    - Unique String identifier
  - Value
    - A primitive value
    - A Serializable / Parcelable object
      - Writing and reading a complex class
      - More on this later on (IPC)
  - i.e. myBundle.putInt(“myInteger”, 5);
  - … int i = myBundle.getInt(“myInteger”);

## Let’s have a look…


Resumed (visible)
onResume()
onResume()
onPause()
Started (visible)
Paused (partially visible)
onStart()
onStart()
onStop()
Created
onRestart()
Stopped (hidden)
onCreate()
onDestroy()
Destroyed

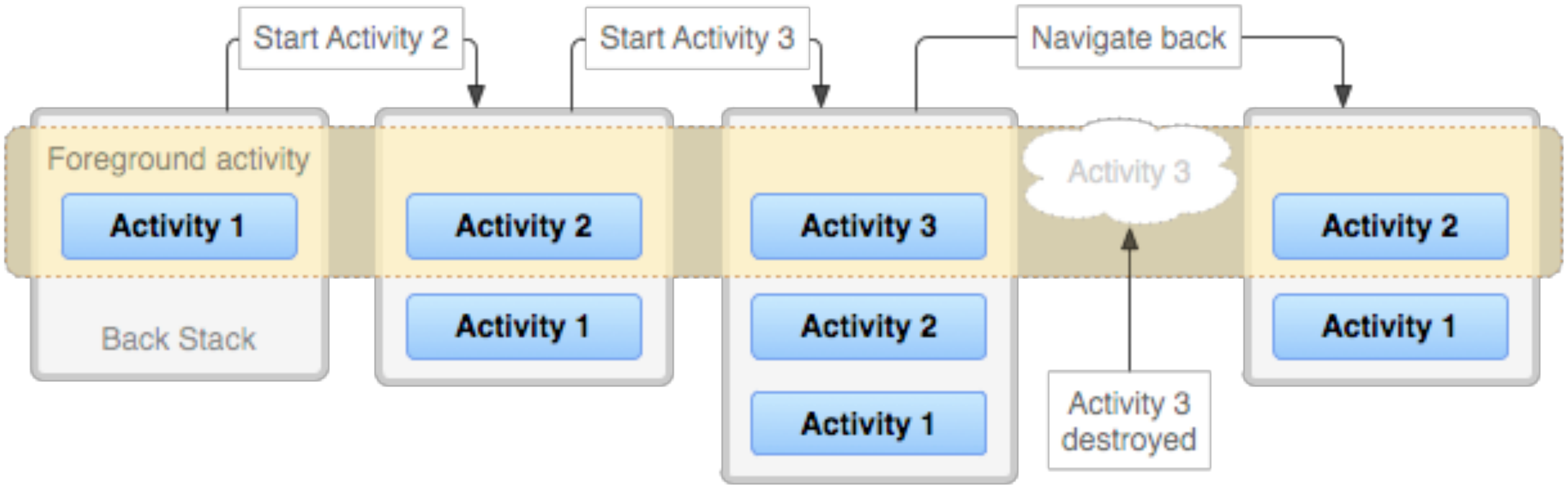
# Intent

- Activities are started by sending an **Intent**
- Represented by an **Intent** object
  - Contains the name of the action requested
  - And the URI of the data to act on

```
Uri webpage = Uri.parse("http://www.cs.nott.ac.uk");
Intent myIntent = new Intent(Intent.ACTION_VIEW, webpage);

Uri number = Uri.parse("tel:01151234567");
Intent myIntent = new Intent(Intent.ACTION_DIAL, number);

Intent myIntent = new Intent(this, otherActivity.class);

startActivity(myIntent);
```

# Intents

- Don't just instantiate the Activity sub-class
  - Already noted that Android works by passing Intent objects about
    - Intent is used to describe an operation
    - Action and the data to operate on (as a URI)
    - Allows for runtime binding
- Starting an Activity
  - Create a new Intent object
  - Specify what you want to send it to
    - Either implicitly, or explicitly
  - Pass the Intent object to **startActivity()**
    - New Activity then started
- Stopping an Activity
  - The called Activity can return to the original one by destroying itself
    - By calling the method **finish()**
  - Or when the user presses the back button


Start Activity 2
Start Activity 3
Navigate back
Foreground activity
Activity 1
Back Stack
Activity 2
Activity 1
Activity 3
Activity 2
Activity 1
Activity 3
Activity 3 destroyed
Activity 2
Activity 1

# Inter-Activity Communication

- We've decomposed a task into multiple activities
  - How do Activities communicate?
  - String otherClass.doStuff(String arg2, MyObject b);
  - Potentially cross-process (IPC)
    - Across memory boundaries enforced by the kernel
  - NB! **Classes** vs **Activities**
- startActivity()
- startActivity(send some data to the new activity)
- startActivityForResult()
- startActivityForResult(send some data)
  - …expect some data back

# Inter-Activity Communication

- startActivity() doesn't allow the Activity to return a result
  - Applications usually want to maintain state
    - Remember what the user has done across all activities
    - We could store state in the broader Application context
      - But activities may be communicating between processes (IPC)
      - Entry point for other applications
- startActivityForResult()
  - Still takes an Intent object, but also a numerical request code
    - Returns an integer result code, set with setResult()
- onActivityResult() then called on the calling Activity
  - Data can be packaged up in an Intent / Bundle
    - Activity creates an Intent object containing the result
    - Use a Bundle to "bundle" complicated objects
  - Calls setResult() to return the Intent
  - Intent object then passed to onActivityResult() on finish

## Let’s have a look…