



LA TROBE
UNIVERSITY

All kinds of clever

CSE2MAD

Mobile Application Development Lecture 3 Part 2



Outline for second session

Consolidating your Android experience thus far

- Event-Driven Programming
- Android App lifecycle
- Android Activities



Consolidating your knowledge

XML vs
Code

Debugging

Activities

Events

Recap from lecture:

- Activity
- Broadcast Receiver
- Service
- Content Provider

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

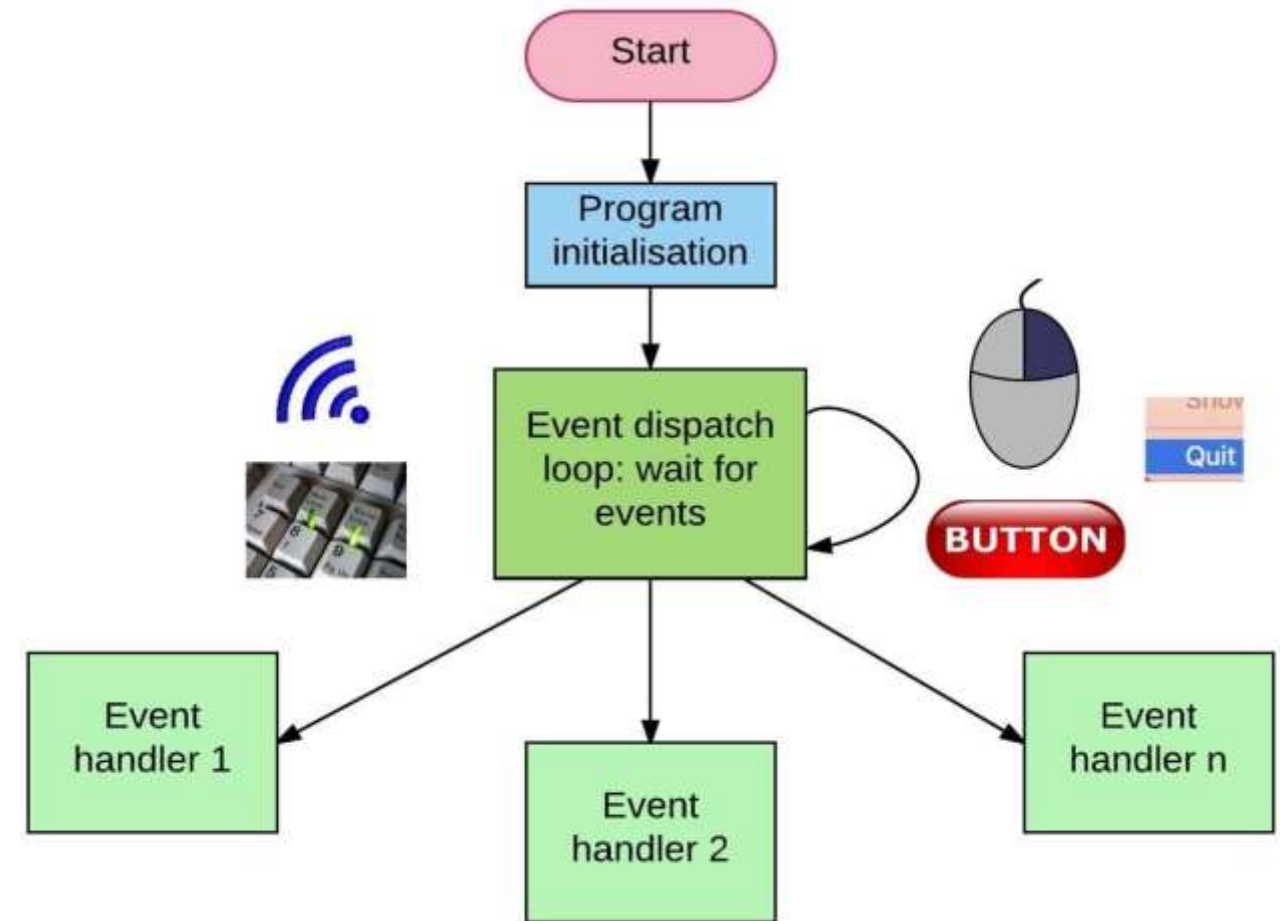
1. Receives and Reacts to broadcast **Intents**
2. No UI but **can start** an Activity
3. **Extends** the **BroadcastReceiver** Base Class

Content Provider

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


Image source: <http://www.slideshare.net/androidstream/android-services>

3.2 Event-Driven Programming



Event-Driven Programming in Android

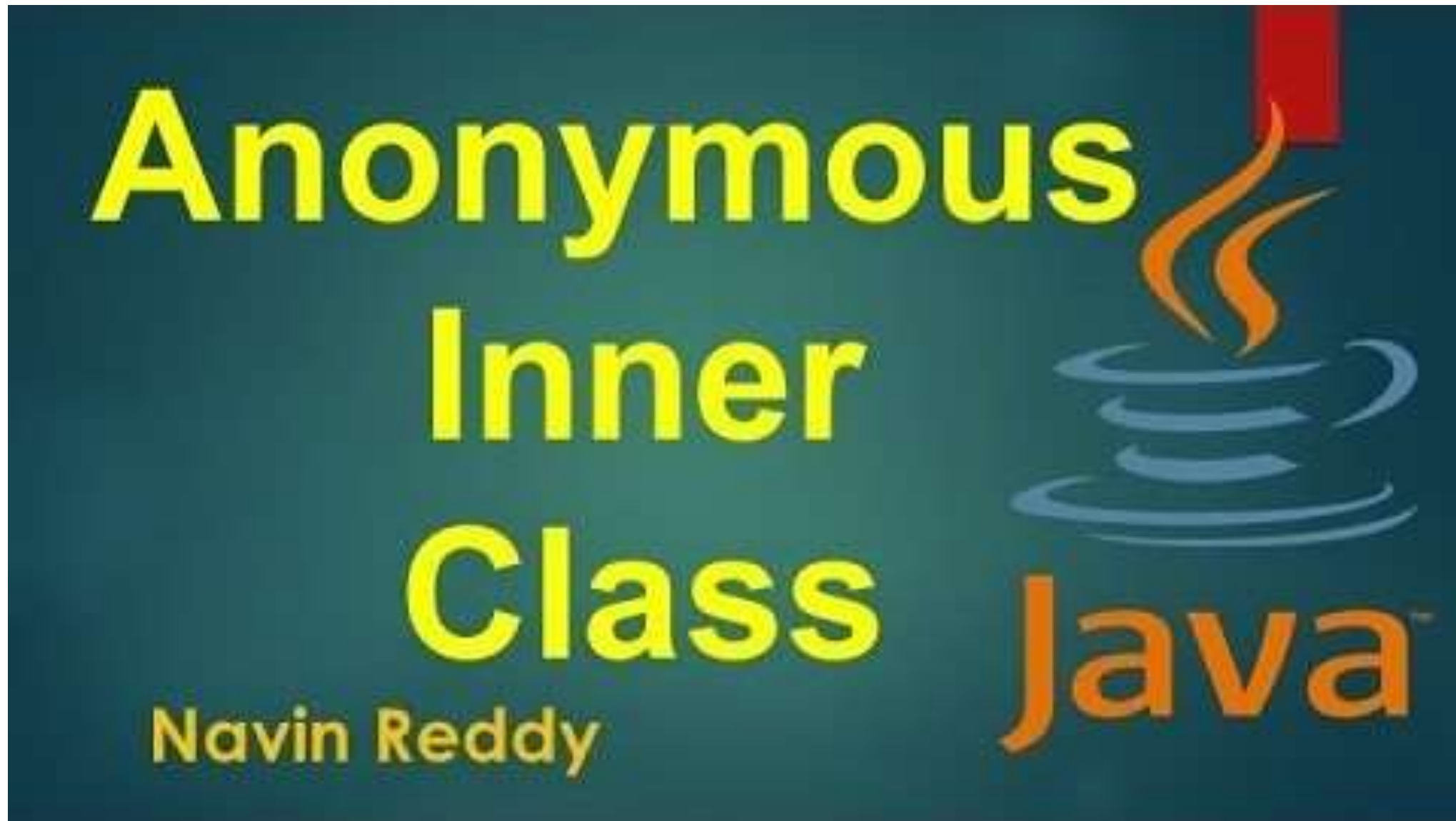
A thick red horizontal bar spans the width of the slide, with a vertical red bar extending downwards from its right end.

- Programming model, roughly: things happen to an app
 - Android app “sits” waiting for events to happen and then responds to them
 - User presses buttons, swipes, etc
- 
- A thick gray horizontal bar spans the width of the slide at the bottom.

3.2.1 Asynchronous Callbacks

- To handle asynchronous events
 - synchronous vs asynchronous?
 - Can you think of some asynchronous events that happen in a computer program?

3.2.2 Anonymous Inner Classes



3.2.2 Anonymous Inner Classes

```
addWindowListener(  
    new WindowAdapter() {  
        public void windowClosing( WindowEvent e ) {  
            System.exit(0);  
        }  
    });
```

This is the same as the following non-anonymous inner:

```
public class WindowClosingAdapter extends WindowAdapter {  
    public void windowClosing( WindowEvent e ) {  
        System.exit(0);  
    }  
}  
  
...  
  
addWindowListener( new WindowClosingAdapter() );
```

3.2.3 Android - Event Handling

Event Listeners – An event listener is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

Event Listeners Registration – Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

Event Handlers – When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.

3.2.3 Event Listeners & Event Handlers

An **event listener** is an interface in the View class that contains a single callback method. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

Included in the event listener interfaces are the following **callback** methods:

`onClick()`

From `View.OnClickListener`. This is called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball.

`onLongClick()`

From `View.OnLongClickListener`. This is called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second).

`onFocusChange()`

From `View.OnFocusChangeListener`. This is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

3.2.4 Event Listeners Registration

... put everything together

```
// Create an anonymous implementation of OnClickListener
private OnClickListener corkyListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    ...
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(corkyListener);
    ...
}
```

3.2.4 Event Listeners Registration

Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

There are many ways to register an event listener these would be the 3 most common

- Using an Anonymous Inner Class
- Activity class implements the Listener interface.
- Using Layout file activity_main.xml to specify event handler directly.

See

https://github.com/latrobe-cs-educator/CSE2MAD_Lecture3_Event_Listener_Demo
for examples

3.2 Android lifecycles

Android has lifecycles for its components;



activity lifecycle



service lifecycle

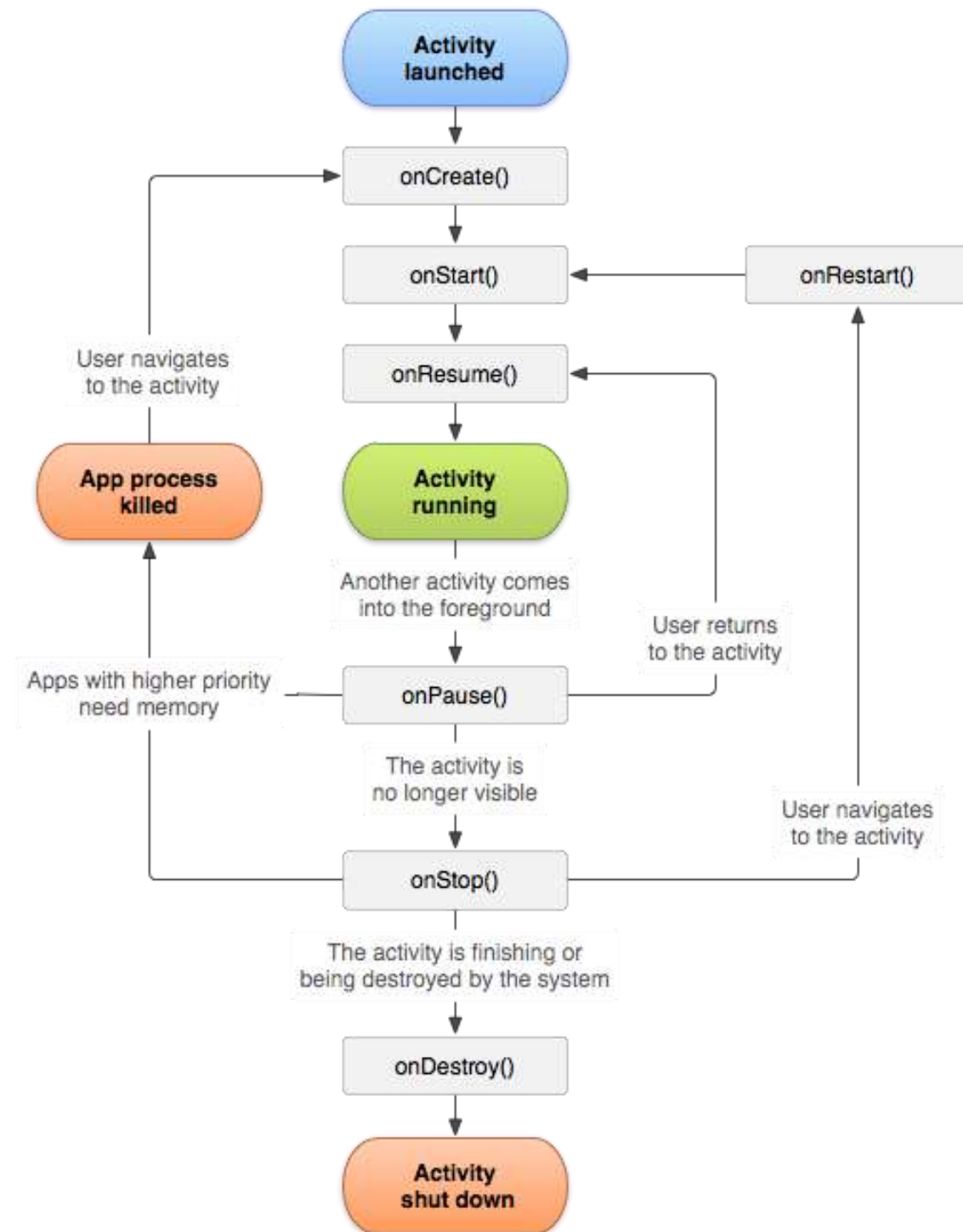


broadcast receiver
lifecycle



content provider
lifecycle

3.2.1 Android Activity lifecycle



onCreate()

onStart()

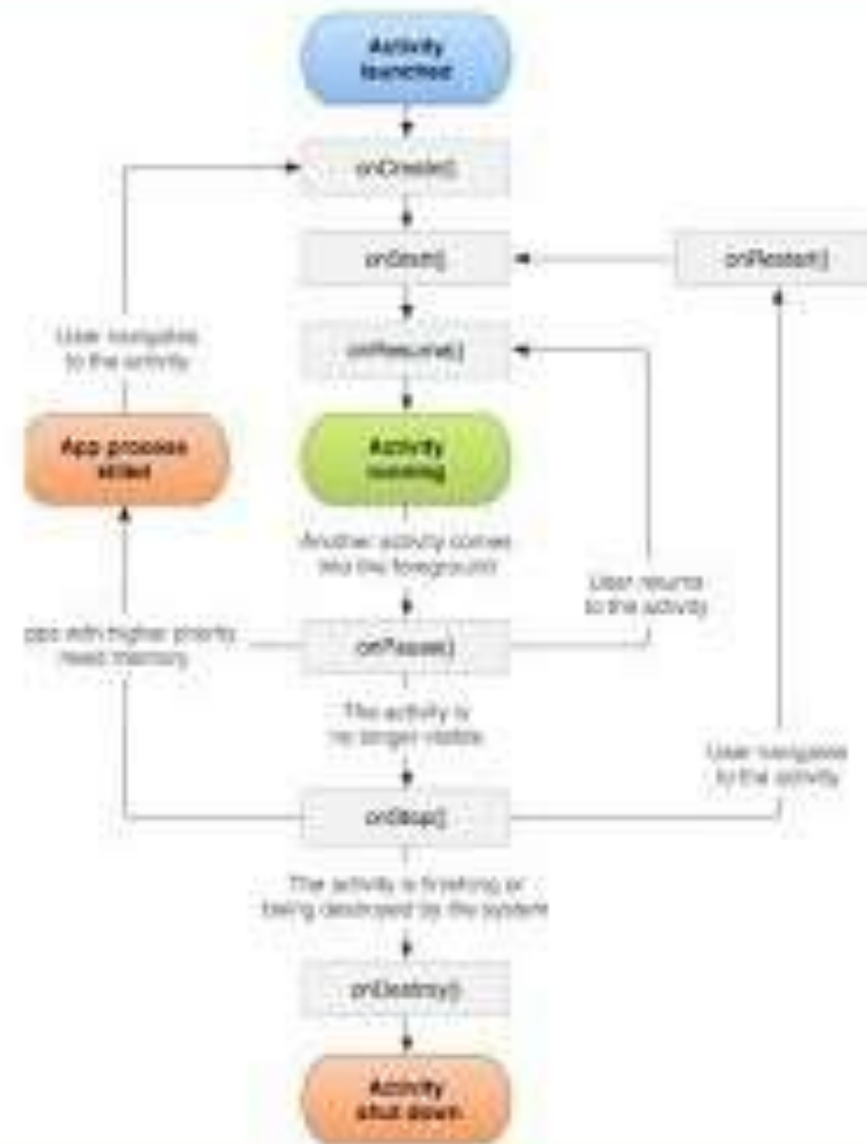
onRestart()

onResume()

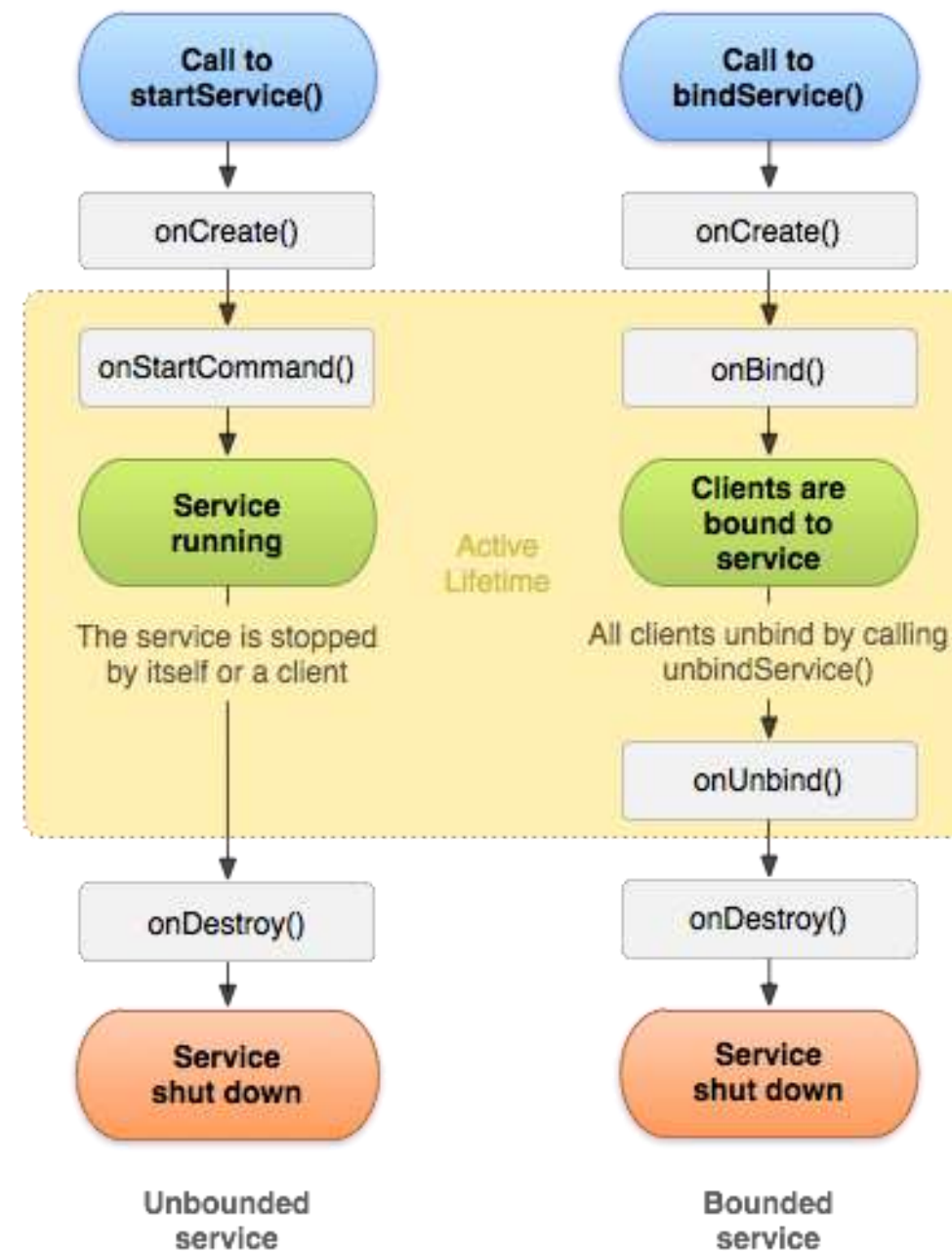
onPause()

onStop()

onDestroy()

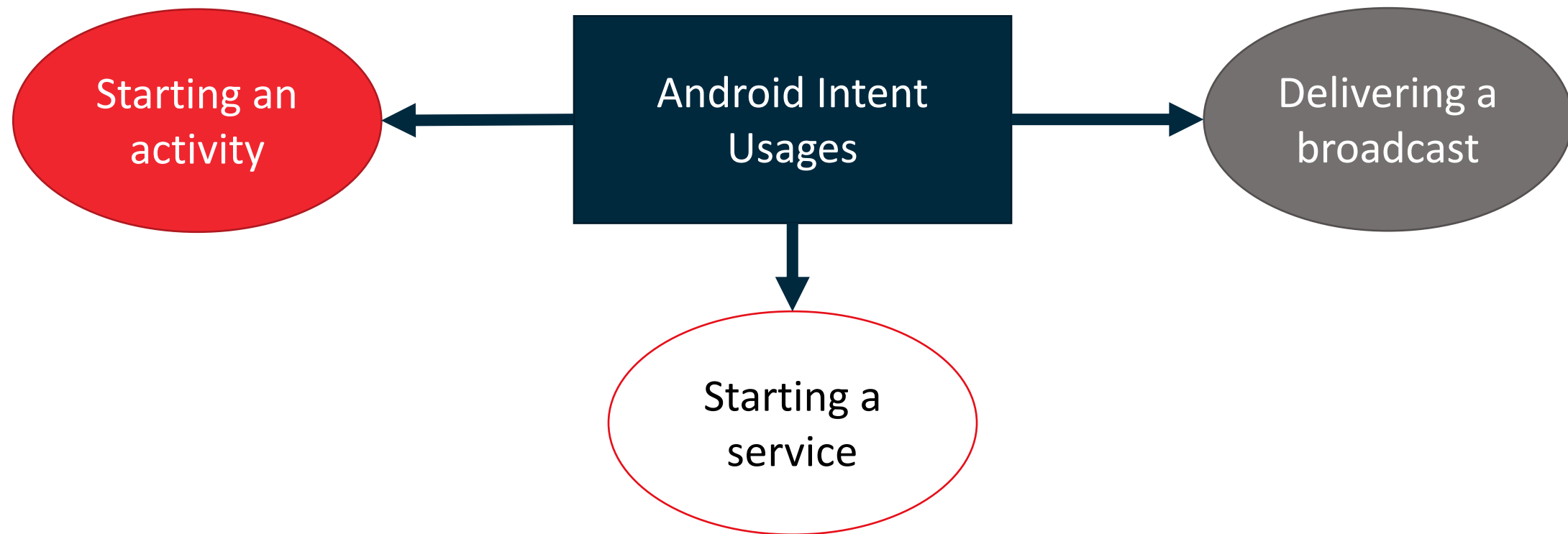


3.2.2 Android Service lifecycle



The service lifecycle. The diagram on the left shows the lifecycle when the service is created with `startService()` and the diagram on the right shows the lifecycle when the service is created with `bindService()`.

3.2.3 Android Intents



An Intent is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways, there are three fundamental use cases:

3.2.3 Explicit Intents

Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action or start a service to download a file in the background.

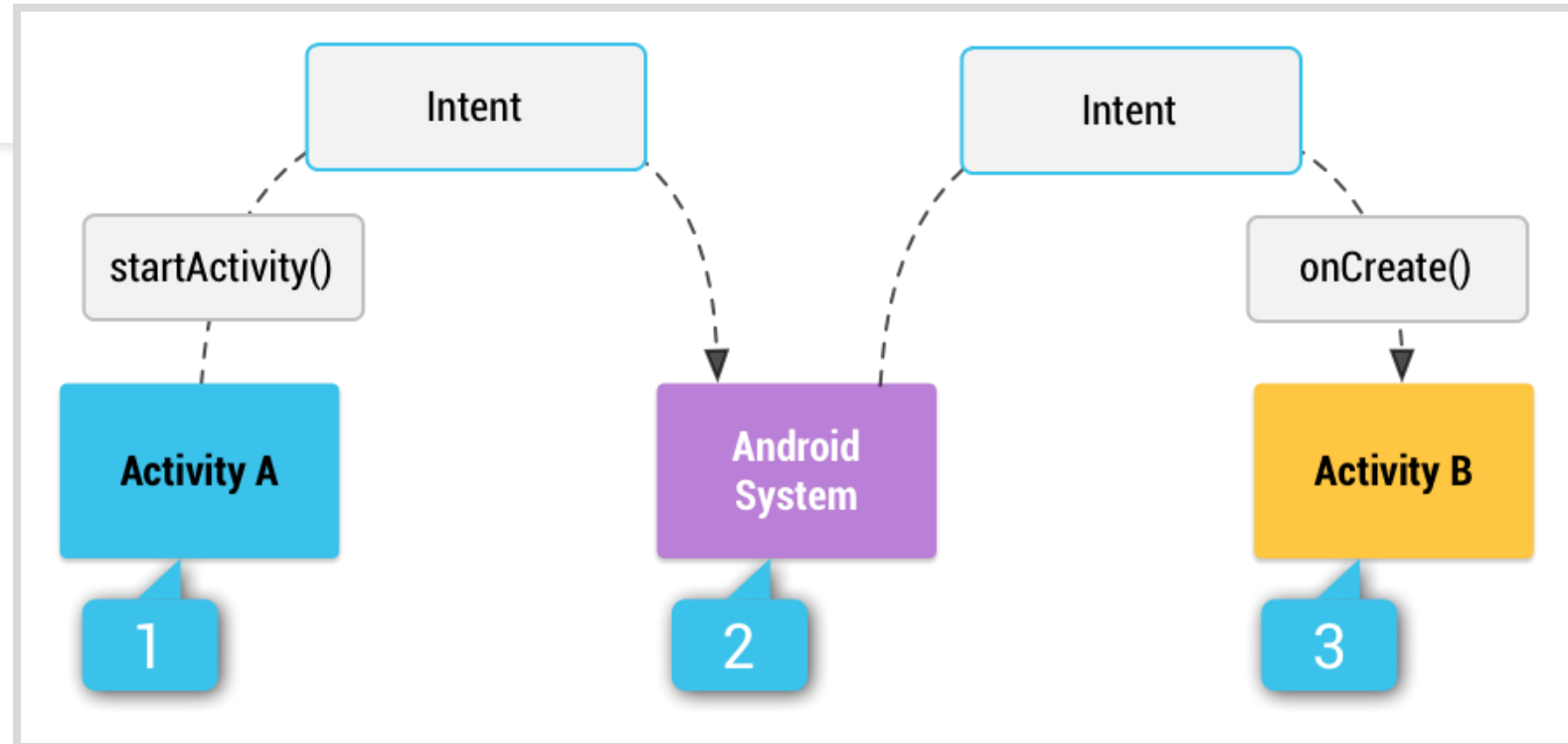
```
public void goToSwitchActivity(View view) {  
    //EXPLICIT intent example  
    Intent intent = new Intent( packageContext: this, SwitchButtonExample.class);  
    startActivity(intent);  
}
```

3.2.3 Implicit Intents

Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

```
String url = "https://www.latrobe.edu.au/";  
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));  
startActivity(intent);
```

3.2.4 Intent Filters



When you use an implicit intent, the Android system finds the appropriate component to start by comparing the contents of the intent to the intent filters declared in the manifest file of other apps on the device. If the intent matches an intent filter, the system starts that component and delivers it the Intent object. If multiple intent filters are compatible, the system displays a dialog so the user can pick which app to use.

3.2.4 Intent Filters

```
<activity ...>
  <intent-filter>
    <action android:name="android.intent.action.VIEW" />
    <!-- Include the host attribute if you want your app to respond
         only to URLs with your app's domain. -->
    <data android:scheme="http" android:host="www.example.com" />
    <category android:name="android.intent.category.DEFAULT" />
    <!-- The BROWSABLE category is required to get links from web pages. -->
    <category android:name="android.intent.category.BROWSABLE" />
  </intent-filter>
</activity>
```

3.2.4 Intents... More Examples

Send an Email

```
public void composeEmail(String[] addresses, String subject) {  
    Intent intent = new Intent(Intent.ACTION_SENDTO);  
    intent.setData(Uri.parse("mailto:")); // only email apps should handle this  
    intent.putExtra(Intent.EXTRA_EMAIL, addresses);  
    intent.putExtra(Intent.EXTRA_SUBJECT, subject);  
    if (intent.resolveActivity(getPackageManager()) != null) {  
        startActivity(intent);  
    }  
}
```

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

Starts a new activity

3.2.5 Starting an Activity for a Result

```
Intent i = new Intent(this, ActivityB.class);

final EditText editText1 = findViewById(R.id.editText1);
String myString = editText1.getText().toString();
i.putExtra("qString", myString);
startActivity(i);
startActivityForResult(i, request_code);
```

ActivityB is started by ActivityA

Data is returned via intent
to the onActivityResult()
method in ActivityA

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    if ((requestCode == request_code) &&
        (resultCode == RESULT_OK)) {

        TextView textView1 = findViewById(R.id.textView1);
        String returnString =
            data.getExtras().getString("returnData");

        textView1.setText(returnString);

    }
}
```