

The First Look At Android Programming with Kotlin

Veljko Pejovic (veljko.pejovic@fri.uni-lj.si)

University of Ljubljana, Slovenia

The World of Android

- The Android Platform
 - A mobile operating system + libraries + application frameworks + key apps
 - Based on Linux and open source
- Runs on a range of devices, market share ~ 70%
- Android SDK for creating apps
 - Lots of documentation
 - Huge community



Android Versions



API 30

API 31

API 32

API 33

API 34

Key Android Features

- Process management tailored for battery-powered and low-memory devices
 - When an app is not used, it gets suspended by Android
 - When the memory is low, suspended apps are terminated
- Support for direct manipulation interfaces
 - Touchscreen gestures, sensors, notifications
- Open ecosystem of applications
 - Support for developing and distributing Android apps

**So, how do I start programming for
Android?**

Android Development Environment Tools

- Android Software Development Kit (SDK)
 - Libraries
 - Debugger
 - Android device emulator via Android Debug Bridge (adb)
- Android Studio (IDE)
 - Code editor
 - Compilation, running, emulation control

Anatomy of an Android App

Conceptual parts

User Interface (UI)

- What the user sees and can interact with
- Written in XML or using Jetpack Compose
- Standard elements: Layouts, Buttons, TextViews, etc.
- In “res” folder (in case of XML)

Interaction and navigation

- What happens when a user interacts with the app
- Written in Java or Kotlin
- In “kotlin+java” folder

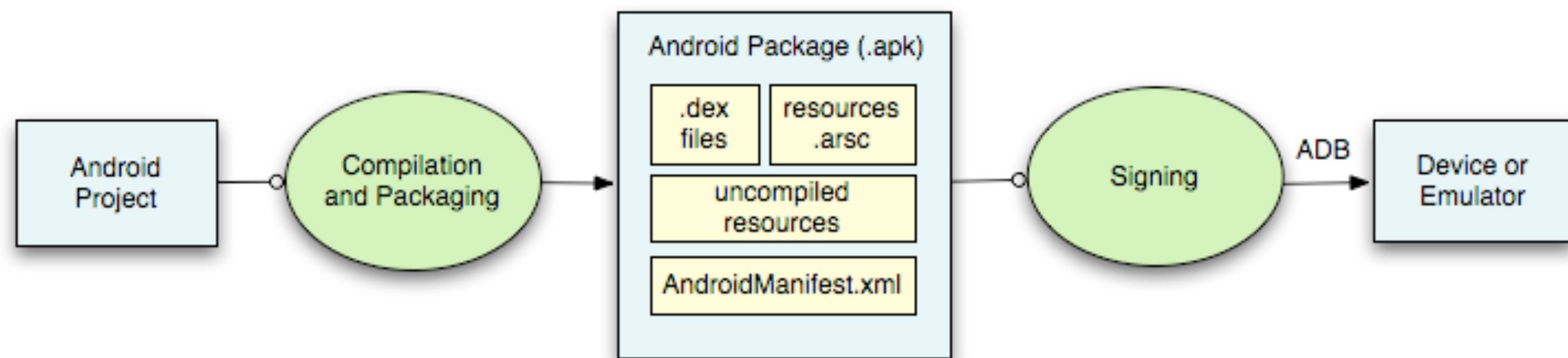
Background processing

- Long-running heavy computation
- Data storage and network communication
- Written in Java or Kotlin
- In “kotlin+java” folder

Anatomy of an Android App

Structure and compilation

- Application
 - A collection of components that are packaged together, can be instantiated and ran as needed
- Non-compiled resources that the application needs
 - Images, Strings, Media files
- Building the application:



Anatomy of an Android App

Basic components

- **Activity**
 - Has a graphical user interface (GUI)
- **Service**
 - Performs background processing
- **BroadcastReceiver**
 - Subscribes to events of interest
- **Intent**
 - Communicates an intention to perform an action

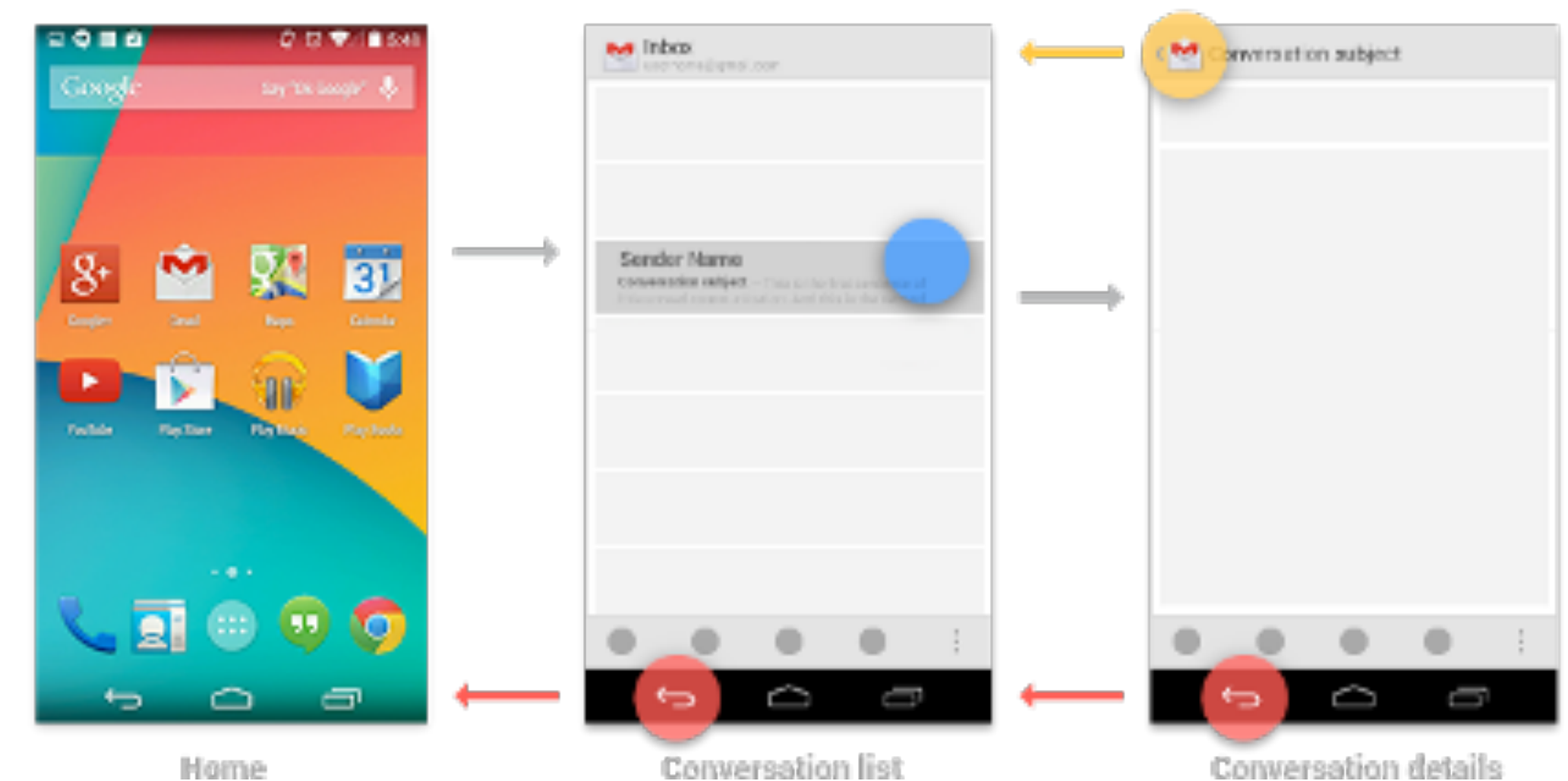
Activity

Purpose

- The primary class for managing user interaction
- One Activity usually implements a single focused task a user can do:
 - Log-in screen
 - Select a contact to write a message to
 - “Compose message” window
- Usually more than one Activity per application
- Activity UI defined in a separate XML layout file in “res” folder

Activity Functioning

- A user's interaction influences the activity that is going to be shown
- Activity launching/parking via **Intents** in the code
- Using “Up”, “Back”, “Home”, “Menu/Recent apps” buttons, swipes



Activity Lifecycle

- Mobile devices have limited resources
 - Battery charge
 - Memory and computing power
 - Screen real estate
- Activities are kept active only when a user can interact with them
- Activities are stopped in the background when not used
- Activities may be destroyed when the OS needs resources

Activity Lifecycle

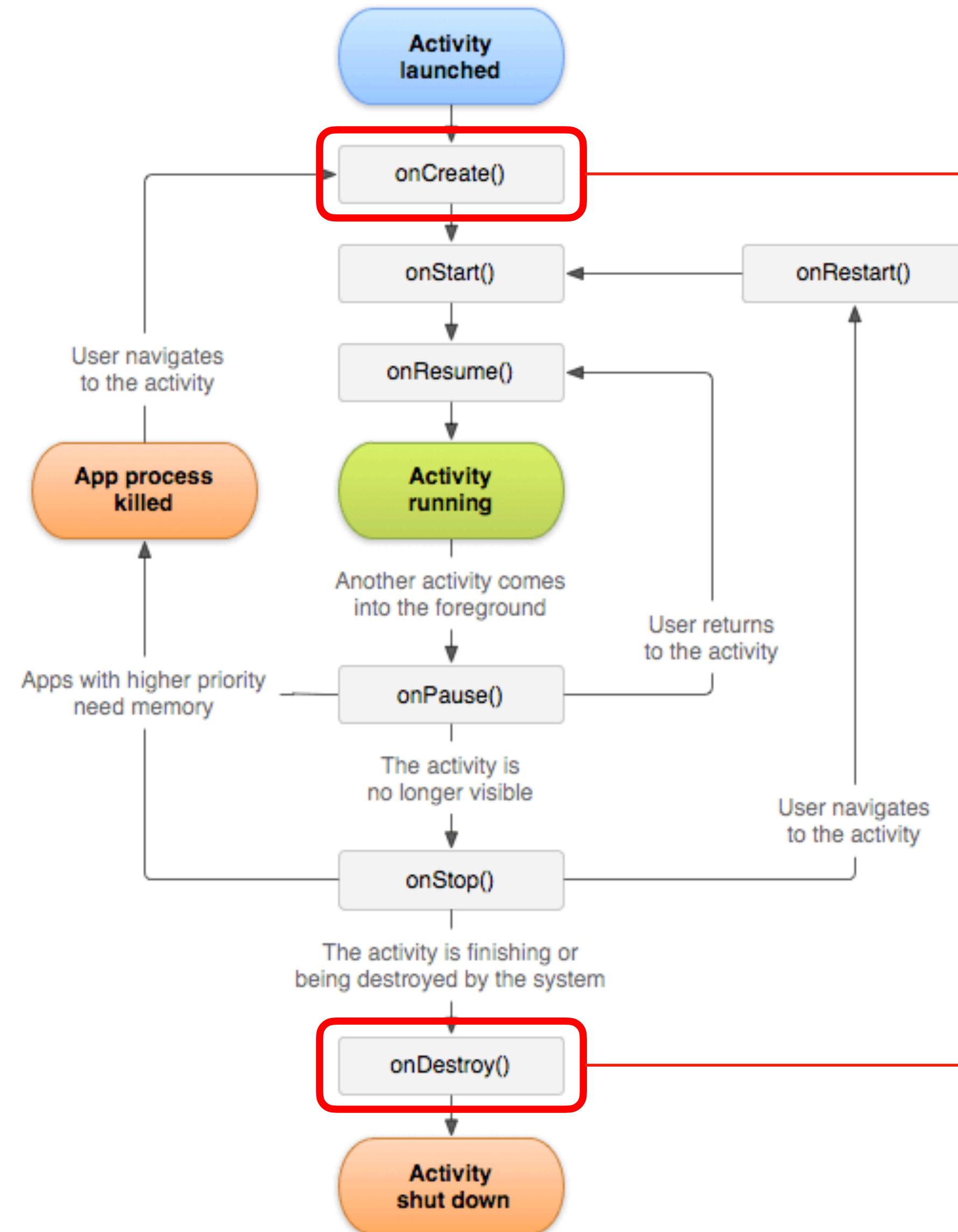
- Activity state:
 - **Active/Running** – in the foreground, visible, user interacting
 - **Paused** – lost focus but still visible, maintains state and member information
 - **Stopped** – completely obscured by another activity, retains state and member information, however, no longer visible; can be terminated by the OS when needed

Activity Lifecycle

- An Activity moves through lifecycle state changes, usually as dictated by the user interaction
- Activity lifecycle state changes trigger the following activity methods:

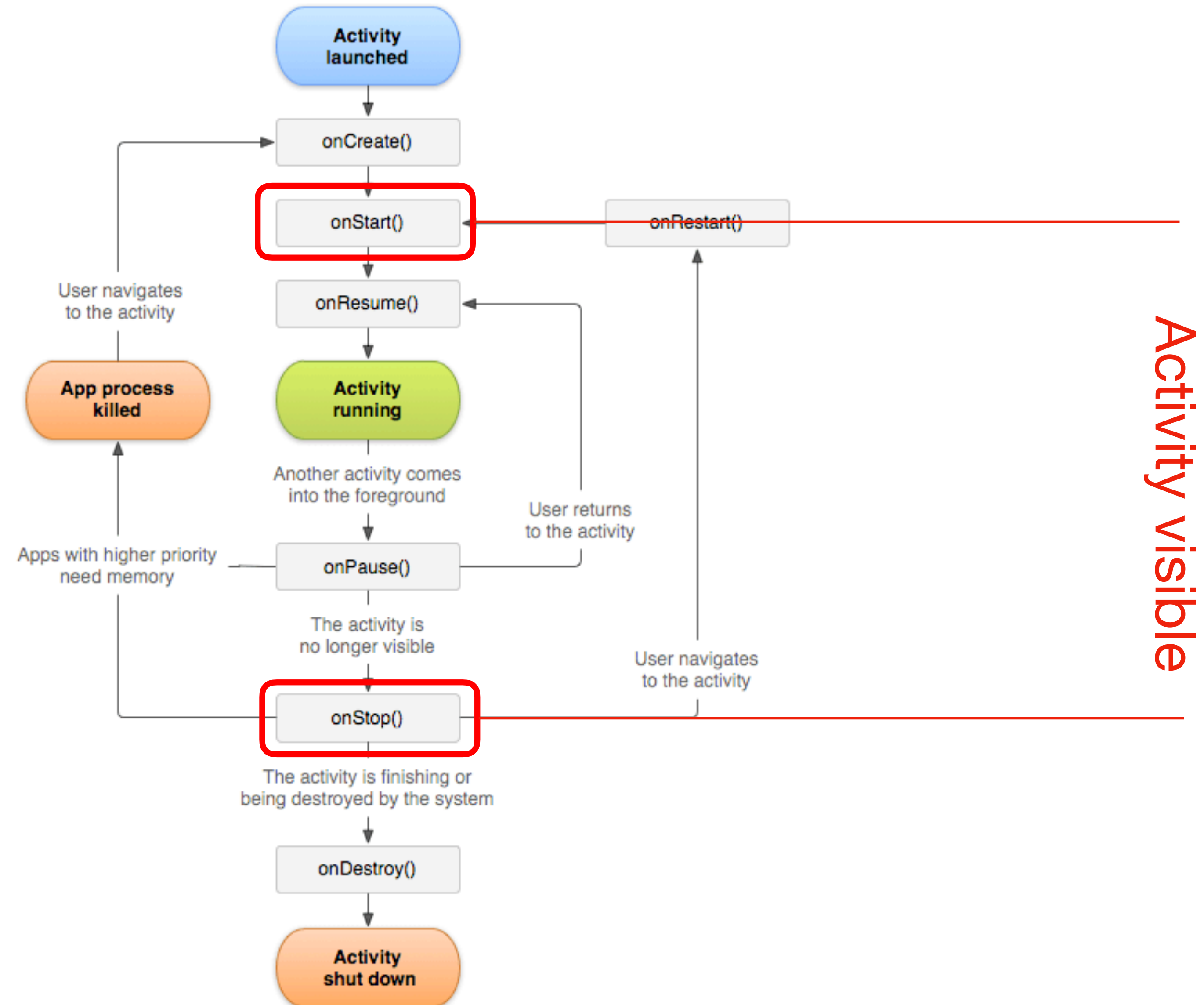
```
protected open fun onCreate(savedInstanceState: Bundle?)
protected open fun onStart()
protected open fun onResume()
protected open fun onPause()
protected open fun onRestart()
protected open fun onStop()
protected open fun onDestroy()
```

Activity Lifecycle

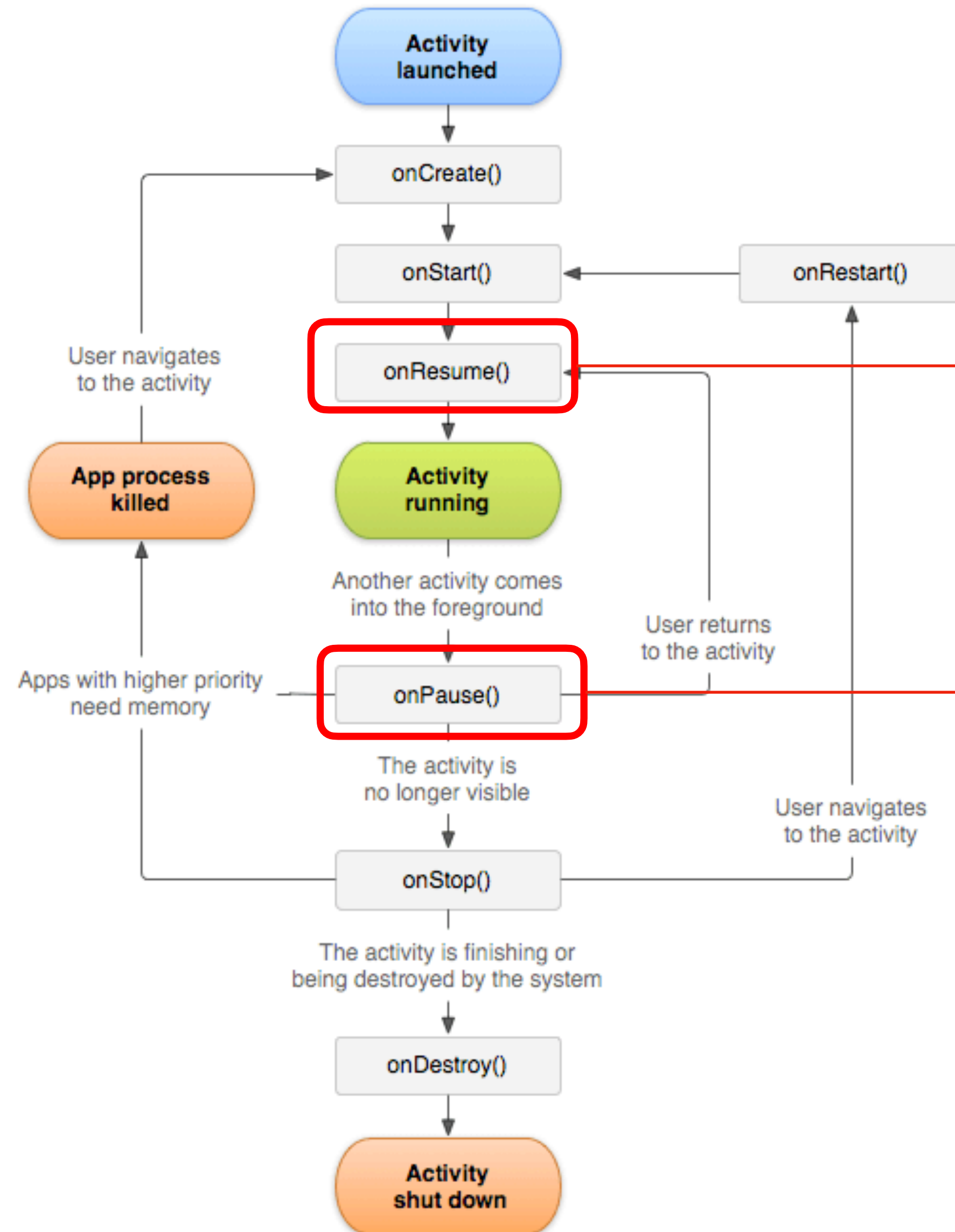


Activity exists

Activity Lifecycle



Activity Lifecycle



Activity
visible and
in foreground

Activity onCreate()

- Called when the activity is first created
- Sets up the initial state:
 - Set the Activity's content view, i.e. instruct the Activity to show something to the user
 - `super.onCreate()` – hides some complex code that must be called to instantiate the Activity properly
 - Initialise other relevant parts (e.g. bind data to lists)
- `onCreate()` also gets a Bundle with the Activity's previous state

Activity

onStart()

- Called when the activity is becoming visible
- Setup state relevant for visible-only behaviour, for example:
 - Register certain BroadcastReceivers
- Load persistent application state

Activity onResume()

- Called when the activity is **visible and is about to start interacting** with the user
- Start foreground-only activities
 - For example, get user location and show it on the map

Activity onPause()

- Called when the Activity loses focus, and another activity is about to start
- Use it to commit unsaved changes to persistent data, stop animations, CPU-intensive processing
- Processing in this method should be done quickly, because the next activity will not start until this method returns
 - Alternatively, run a parallel thread from onPause()

Activity

onStop()

- Called when the Activity is no longer visible
 - Another Activity is being started, an existing one is being brought in front of this one, or this one is being destroyed
- Release resources that are not needed while the activity is not visible
- Perform CPU-heavy shutdown operations
- Your activity still exists, but does not have UI

Activity onDestroy()

- Called when the Activity is about to get destroyed
 - Happens when finish() is called
 - Happens when the OS calls it
- Use it to release resources such as Threads that are associated with the Activity
- Note: **may be skipped if the OS kills your application**

Activity

Starting activities

- Create an Intent specifying the Activity to start
- Pass the **Intent** to one of the following methods:
 - **startActivity()**
 - launches the Activity described by the Intent
 - **startActivityForResult()**
 - launches the Activity described by the Intent and expects a result that will be returned via **onActivityResult**
 - the called activity can set result via **setResult()** method

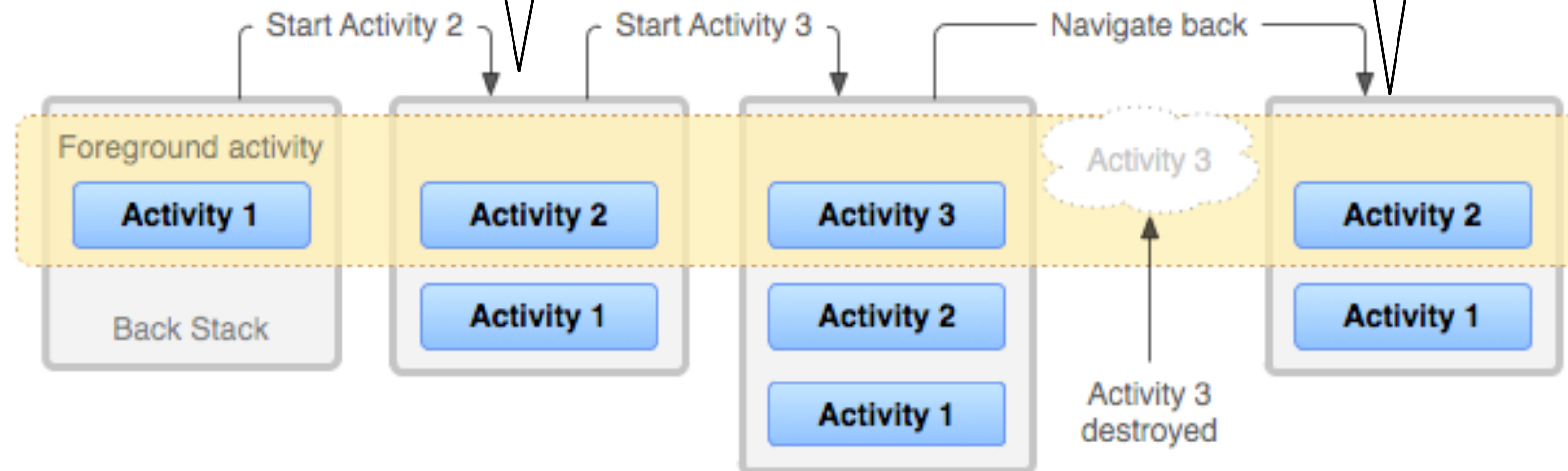
Task

- A task is a collection of Activities that users interact with when performing a certain job
- The Activities need not be from the same application (although usually they are)
- Backstack: the activities are arranged in a stack in the order in which each activity is opened
 - When launched the activity goes on top of the backstack
 - When destroyed it is popped of the **backstack**

Backstack

A new activity (Activity 2) is created and started, the old one (Activity 1) is stopped

Activity 3 destroyed when the user clicked BACK, Activity 2 is started



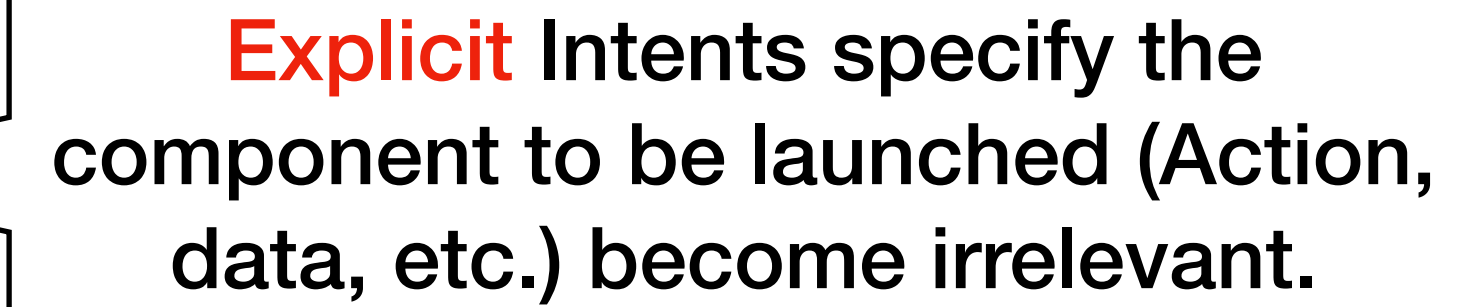
Intent

Purpose

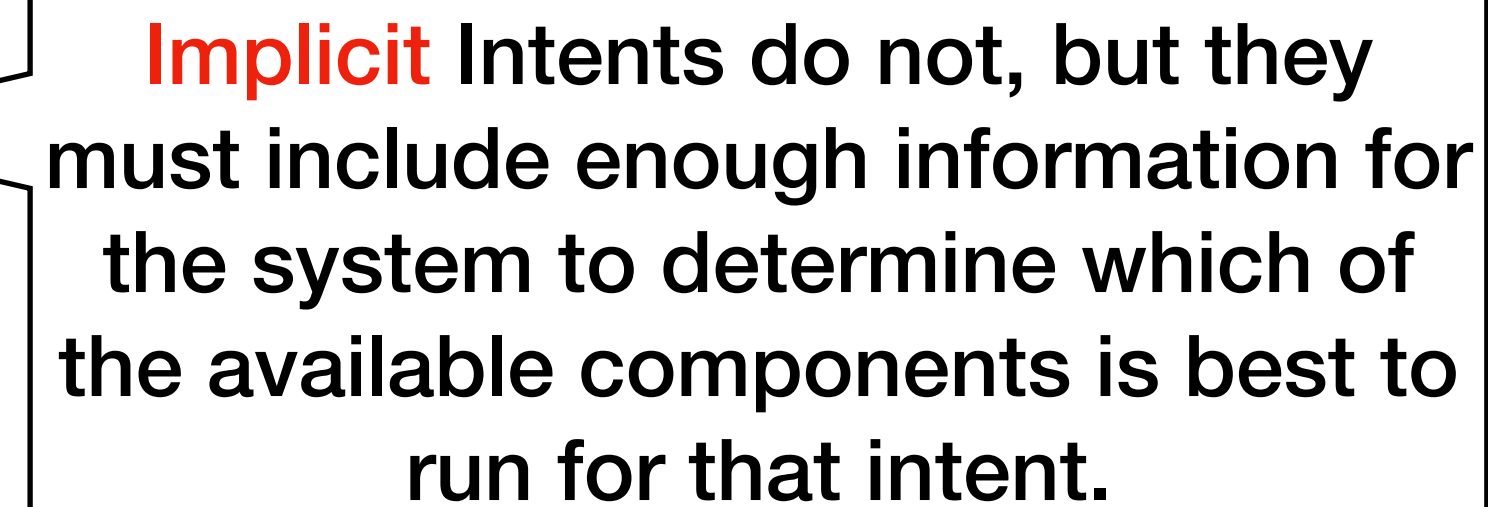
- A data structure representing:
 - An operation to be performed or
 - An event that has occurred
- Intents serve as a glue between components
 - Constructed by a component that wants some work to be done
 - Received by a component that can perform that work
- Hold an abstract description of an action to be performed

Intent Fields

- Component
- Action
- Data
- Category
- Type
- Extras



Explicit Intents specify the component to be launched (Action, data, etc.) become irrelevant.



Implicit Intents do not, but they must include enough information for the system to determine which of the available components is best to run for that intent.

Service

Purpose

- Activities run on the UI (main) thread and have a UI attached (layout)
 - Processing-heavy functions on the main thread impact the responsiveness
- **Services** can run on either the Main or separate threads and **do not have a UI** attached
 - Run outside UI, for long-running operations
- Services are often more convenient than custom Threads for tasks that need to be “independent” and **run even when the Activity is destroyed**

Service

Background and foreground

- Background Service
 - For actions that do not have to be noticed by the user (e.g. sensing a user's physical activity)
- Foreground Service
 - For actions that the user needs be aware of and should the control of (e.g. a music player app)
 - A foreground service must show a notification in the **notification** bar

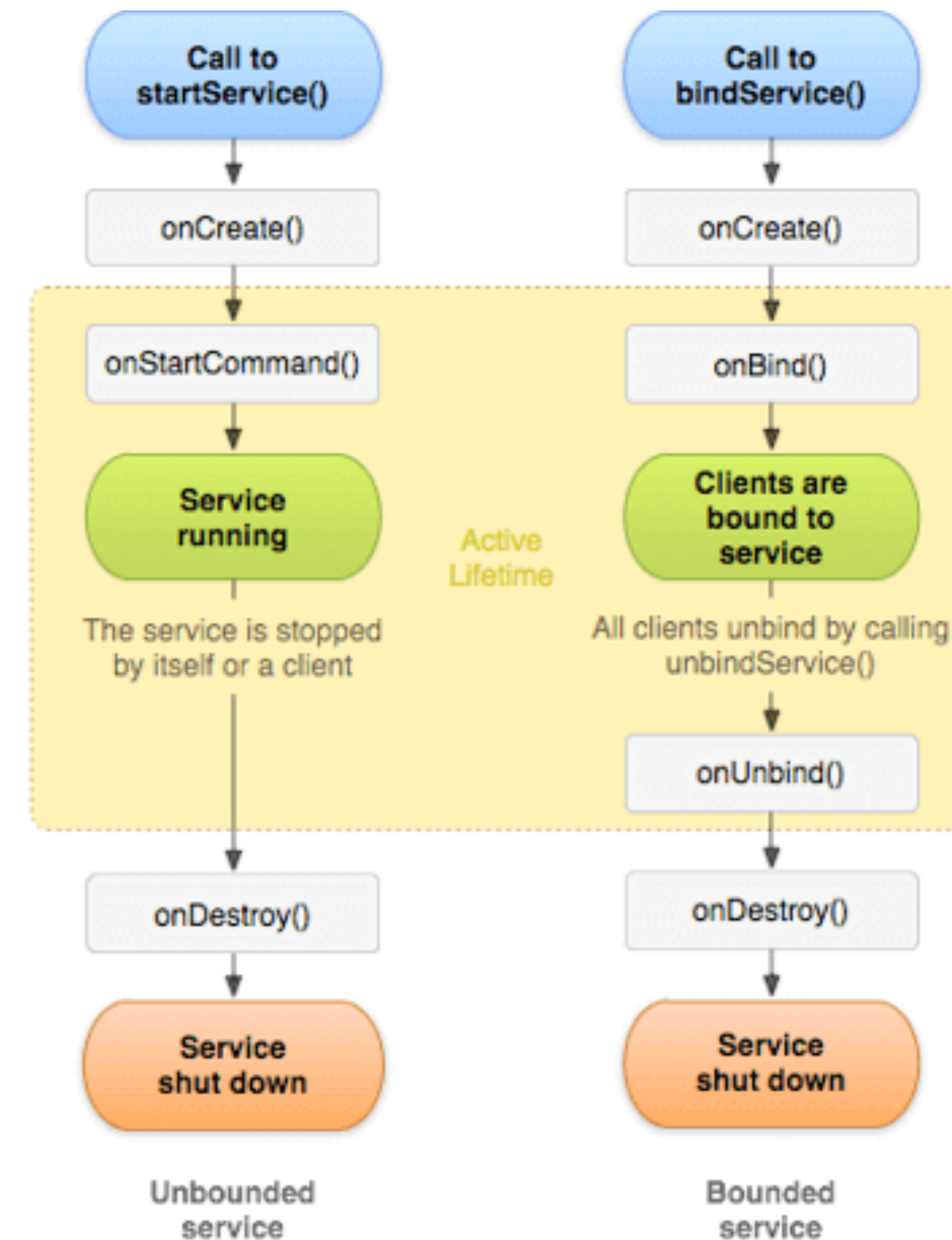
Service

Starting and stopping a Service

- Services can be created:
 - Explicitly using `Context.startService()`
 - Implicitly, if not already running, when a client requests connection to a Service via `Context.bindService()`
- Services can be stopped:
 - From within the Service with `stopSelf()`
 - From another component with `Context.stopService()`

Service Lifecycle

- Multiple startService calls do not nest – you only have one service; however, **onStartCommand()** will be called repeatedly
- Service will be stopped only once with Context.stopService() or stopSelf()



Service

Bound Service

- Like servers in a client-server paradigm
- Services started through binding, do not call onStartCommand()
- Return IBinder object from **onBind(Intent)** so that connected clients can call the Service
- The service remains running as long as the connection is established

Broadcast

Purpose

- Messages sent from components of your app, other apps or from the Android system

- Messages are wrapped in Intents

```
val intent = Intent()  
intent.setAction(ACTION)  
intent.putExtra(STOP_SERVICE_BROADCAST_KEY,  
                RQS_STOP_SERVICE)  
sendBroadcast(intent)
```

- Send Broadcast

- Systems sends certain broadcasts when an event happens, e.g. ACTION_BOOT_COMPLETED
- You can send custom broadcast via **sendBroadcast()**

Broadcast Receiving

- Broadcasts are captured in an app/component if a BroadcastReceiver is registered in the code:
 - Create a **BroadcastReceiver** and impl. **onReceive()**

```
class NotifyServiceReceiver: BroadcastReceiver(){  
  
    override fun onReceive(context: Context, intent: Intent) {  
        ...  
    }  
}
```

- **Register** for receiving certain kinds of Intents

```
val intentFilter = IntentFilter()  
intentFilter.addAction(ACTION)  
registerReceiver(notifyServiceReceiver, intentFilter)
```

Broadcast

Receiving

- Some Broadcasts can be captured if a BroadcastReceiver is registered in AndroidManifest.XML and onReceive() is implemented in the code:

```
<receiver android:name=".MyBroadcastReceiver" android:exported="true">
  <intent-filter>
    <action android:name="android.intent.action.BOOT_COMPLETED"/>
    <action android:name="android.intent.action.INPUT_METHOD_CHANGED"/>
  </intent-filter>
</receiver>
```

```
class MyBroadcastReceiver: BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
    ...
}
```

A few words about Kotlin

Kotlin

Benefits for Android programmers

- Coroutines
 - For background processing, without the need to spawn custom Threads
- Data class
 - Creates fields, getters/setters, equals, toString, hashCode methods for you
- Extension functions
 - You can add a function to any class, without formally placing the function in a class
- Functions are the first-class citizens
 - Stored in variables, passed as arguments

Kotlin

Benefits for Android programmers

- Null safety
 - Distinction between nullable and non-nullable objects
 - Safe navigation operator “?” – ensures that you don’t try to access a property/method of a null object
 - Smart cast

Practical lab: Mobile Music Player