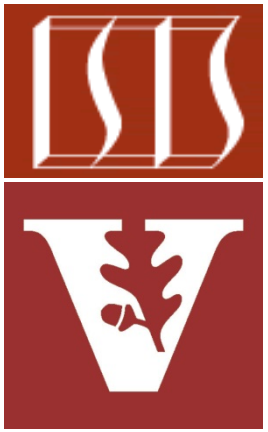
Android Services & Local IPC: Overview of Started & Bound Services

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt



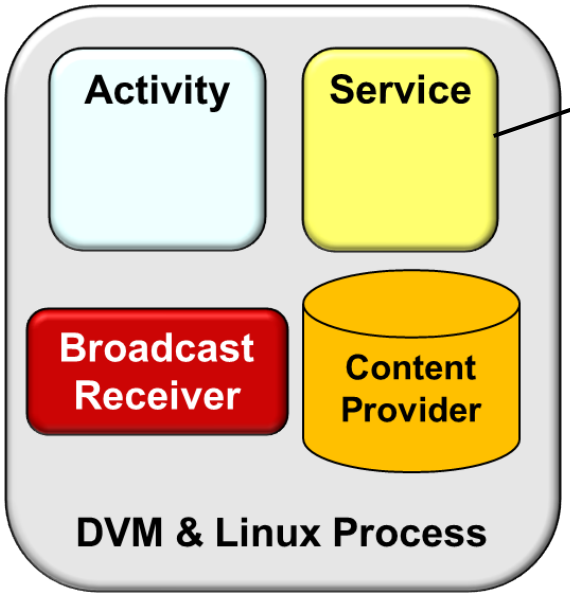
Professor of Computer Science

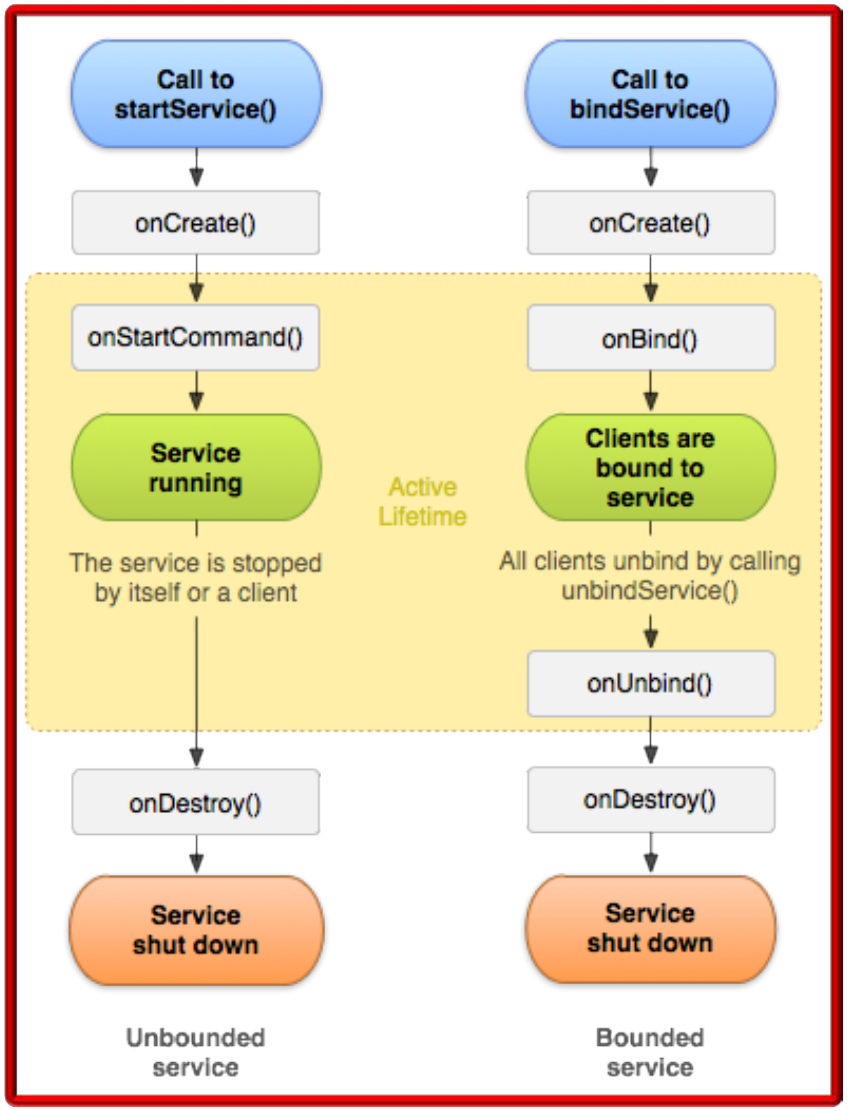
Institute for Software Integrated Systems

Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Module

- Understand the motivations for—& capabilities of—Android Services & the steps involved in implementing & configuring these Services





Motivating the Need for Android Services

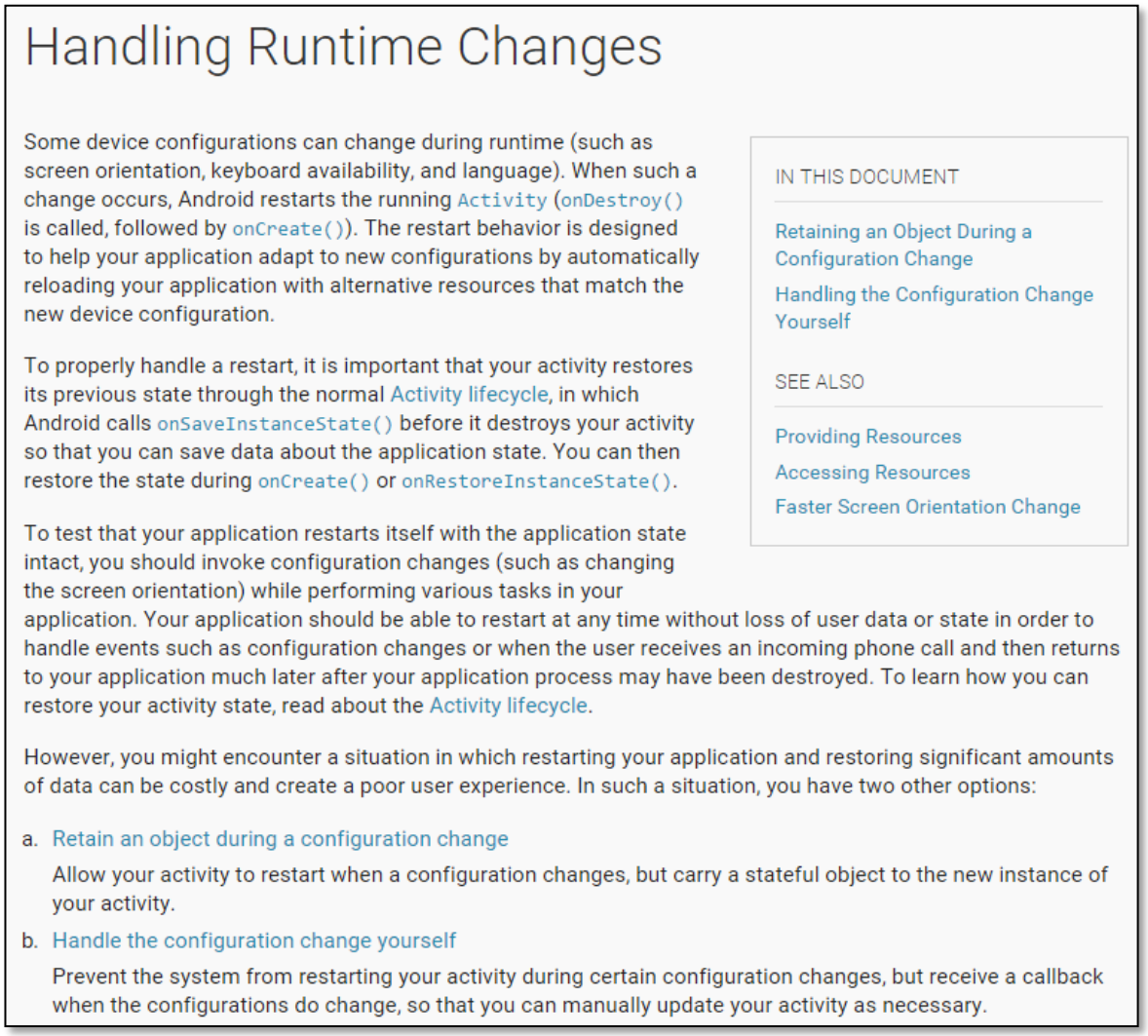
Motivating the Need for Android Services

- Activities have limitations when used w/long-duration concurrent operations

See android-developers.blogspot.com/2010/04/multitasking-android-way.html

Motivating the Need for Android Services

- Activities have limitations when used w/long-duration concurrent operations
  - Must handle runtime configuration changes



Handling Runtime Changes

Some device configurations can change during runtime (such as screen orientation, keyboard availability, and language). When such a change occurs, Android restarts the running `Activity` (`onDestroy()` is called, followed by `onCreate()`). The restart behavior is designed to help your application adapt to new configurations by automatically reloading your application with alternative resources that match the new device configuration.

IN THIS DOCUMENT

Retaining an Object During a Configuration Change

Handling the Configuration Change Yourself

SEE ALSO

Providing Resources

Accessing Resources

Faster Screen Orientation Change

To properly handle a restart, it is important that your activity restores its previous state through the normal Activity lifecycle, in which Android calls `onSaveInstanceState()` before it destroys your activity so that you can save data about the application state. You can then restore the state during `onCreate()` or `onRestoreInstanceState()`.

To test that your application restarts itself with the application state intact, you should invoke configuration changes (such as changing the screen orientation) while performing various tasks in your application. Your application should be able to restart at any time without loss of user data or state in order to handle events such as configuration changes or when the user receives an incoming phone call and then returns to your application much later after your application process may have been destroyed. To learn how you can restore your activity state, read about the Activity lifecycle.

However, you might encounter a situation in which restarting your application and restoring significant amounts of data can be costly and create a poor user experience. In such a situation, you have two other options:

a. Retain an object during a configuration change

   Allow your activity to restart when a configuration changes, but carry a stateful object to the new instance of your activity.

b. Handle the configuration change yourself

   Prevent the system from restarting your activity during certain configuration changes, but receive a callback when the configurations do change, so that you can manually update your activity as necessary.

See developer.android.com/guide/topics/resources/runtime-changes.html

Motivating the Need for Android Services

- Activities have limitations when used w/long-duration concurrent operations
  - Must handle runtime configuration changes
  - Must be destroyed (& thus must be recreated later) when back button pressed

Recreating an Activity

There are a few scenarios in which your activity is destroyed due to normal app behavior, such as when the user presses the *Back* button or your activity signals its own destruction by calling `finish()`. The system may also destroy your activity if it's currently stopped and hasn't been used in a long time or the foreground activity requires more resources so the system must shut down background processes to recover memory.

‹ PREVIOUS NEXT ›

THIS LESSON TEACHES YOU TO

1. Save Your Activity State
2. Restore Your Activity State

YOU SHOULD ALSO READ

- Supporting Different Screens
- Handling Runtime Changes
- Activities

When your activity is destroyed because the user presses *Back* or the activity finishes itself, the system's concept of that `Activity` instance is gone forever because the behavior indicates the activity is no longer needed. However, if the system destroys the activity due to system constraints (rather than normal app behavior), then although the actual `Activity` instance is gone, the system remembers that it existed such that if the user navigates back to it, the system creates a new instance of the activity using a set of saved data that describes the state of the activity when it was destroyed. The saved data that the system uses to restore the previous state is called the "instance state" and is a collection of key-value pairs stored in a `Bundle` object.

> **Caution:** Your activity will be destroyed and recreated each time the user rotates the screen. When the screen changes orientation, the system destroys and recreates the foreground activity because the screen configuration has changed and your activity might need to load alternative resources (such as the layout).

By default, the system uses the `Bundle` instance state to save information about each `View` object in your activity layout (such as the text value entered into an `EditText` object). So, if your activity instance is destroyed and recreated, the state of the layout is restored to its previous state with no code required by you. However, your activity might have more state information that you'd like to restore, such as member variables that track the user's progress in the activity.

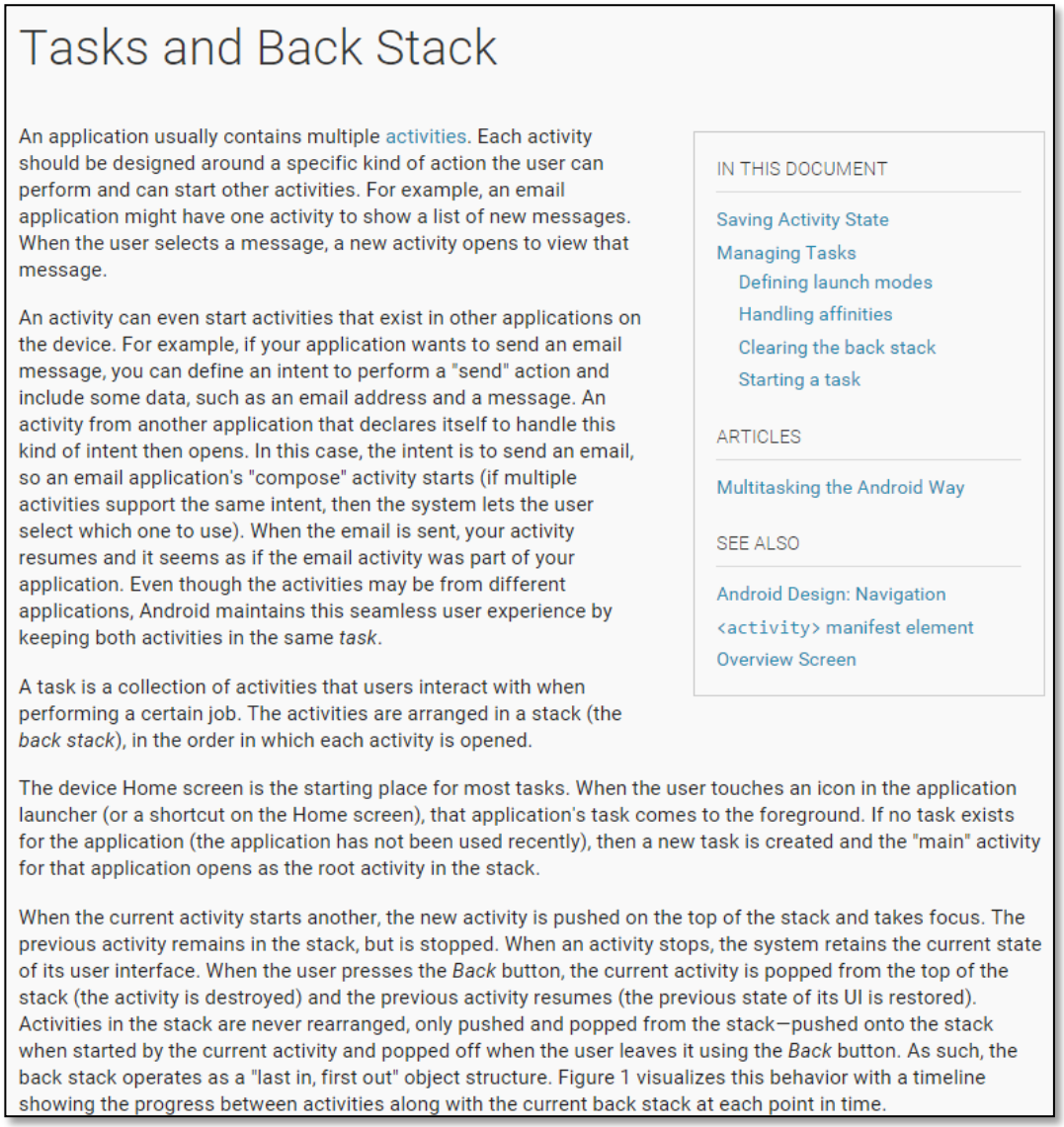
> **Note:** In order for the Android system to restore the state of the views in your activity, **each view must have a unique ID**, supplied by the `android:id` attribute.

To save additional data about the activity state, you must override the `onSaveInstanceState()` callback method. The system calls this method when the user is leaving your activity and passes it the `Bundle` object that will be saved in the event that your activity is destroyed unexpectedly. If the system must recreate the activity instance later, it passes the same `Bundle` object to both the `onRestoreInstanceState()` and `onCreate()` methods.

See developer.android.com/training/basics/activity-lifecycle/recreating.html

Motivating the Need for Android Services

- Activities have limitations when used w/long-duration concurrent operations
  - Must handle runtime configuration changes
  - Must be destroyed (& thus must be recreated later) when back button pressed
  - An existing Activity is paused when a new Activity is started



Tasks and Back Stack

An application usually contains multiple activities. Each activity should be designed around a specific kind of action the user can perform and can start other activities. For example, an email application might have one activity to show a list of new messages. When the user selects a message, a new activity opens to view that message.

An activity can even start activities that exist in other applications on the device. For example, if your application wants to send an email message, you can define an intent to perform a "send" action and include some data, such as an email address and a message. An activity from another application that declares itself to handle this kind of intent then opens. In this case, the intent is to send an email, so an email application's "compose" activity starts (if multiple activities support the same intent, then the system lets the user select which one to use). When the email is sent, your activity resumes and it seems as if the email activity was part of your application. Even though the activities may be from different applications, Android maintains this seamless user experience by keeping both activities in the same *task*.

A task is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the *back stack*), in the order in which each activity is opened.

IN THIS DOCUMENT

Saving Activity State
Managing Tasks
Defining launch modes
Handling affinities
Clearing the back stack
Starting a task

ARTICLES

Multitasking the Android Way

SEE ALSO

Android Design: Navigation
<activity> manifest element
Overview Screen

The device Home screen is the starting place for most tasks. When the user touches an icon in the application launcher (or a shortcut on the Home screen), that application's task comes to the foreground. If no task exists for the application (the application has not been used recently), then a new task is created and the "main" activity for that application opens as the root activity in the stack.

When the current activity starts another, the new activity is pushed on the top of the stack and takes focus. The previous activity remains in the stack, but is stopped. When an activity stops, the system retains the current state of its user interface. When the user presses the *Back* button, the current activity is popped from the top of the stack (the activity is destroyed) and the previous activity resumes (the previous state of its UI is restored). Activities in the stack are never rearranged, only pushed and popped from the stack—pushed onto the stack when started by the current activity and popped off when the user leaves it using the *Back* button. As such, the back stack operates as a "last in, first out" object structure. Figure 1 visualizes this behavior with a timeline showing the progress between activities along with the current back stack at each point in time.

See developer.android.com/guide/components/tasks-and-back-stack.html

Motivating the Need for Android Services

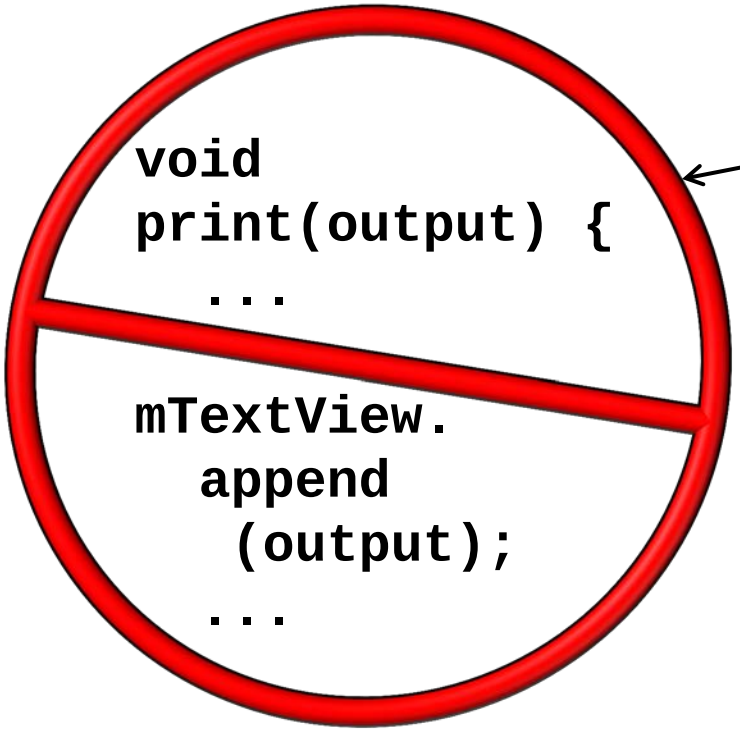
- Activities have limitations when used w/long-duration concurrent operations
  - Must handle runtime configuration changes
  - Must be destroyed (& thus must be recreated later) when back button pressed
  - An existing Activity is paused when a new Activity is started

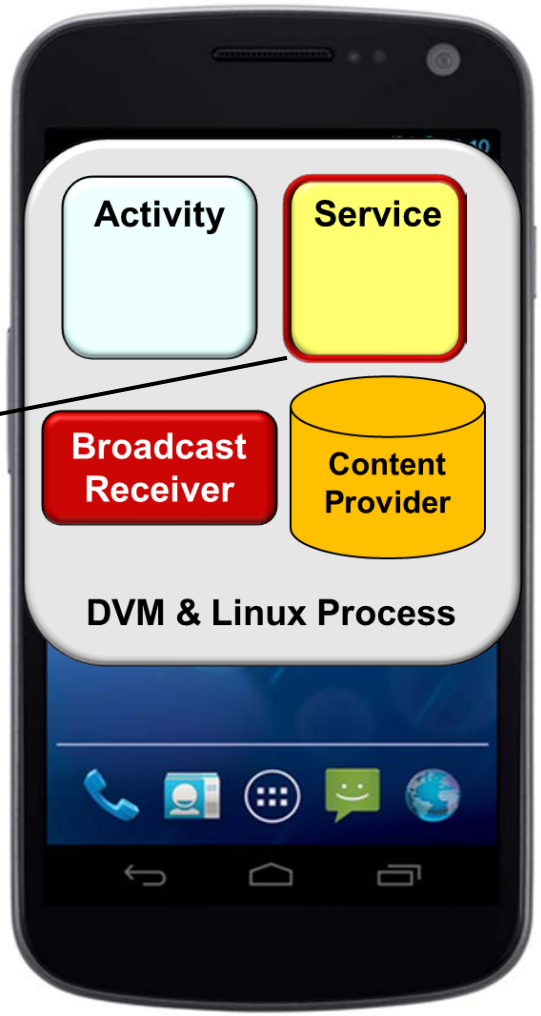
Android therefore needs some other mechanism to perform long-duration operations concurrently in the background

Overview of Android Services

Overview of Android Services

- Unlike Activities, Services don't interact with the user directly

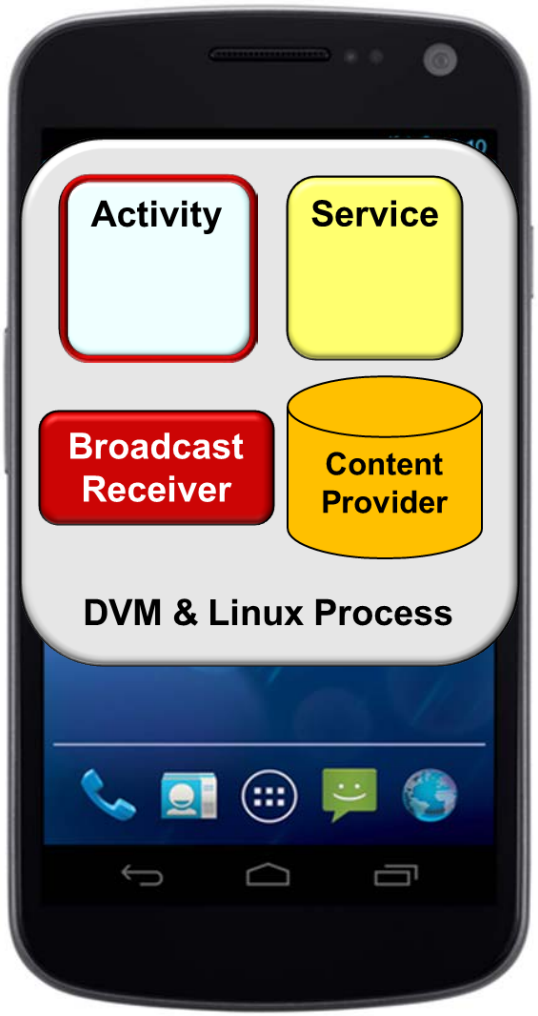




See developer.android.com/guide/components/services.html

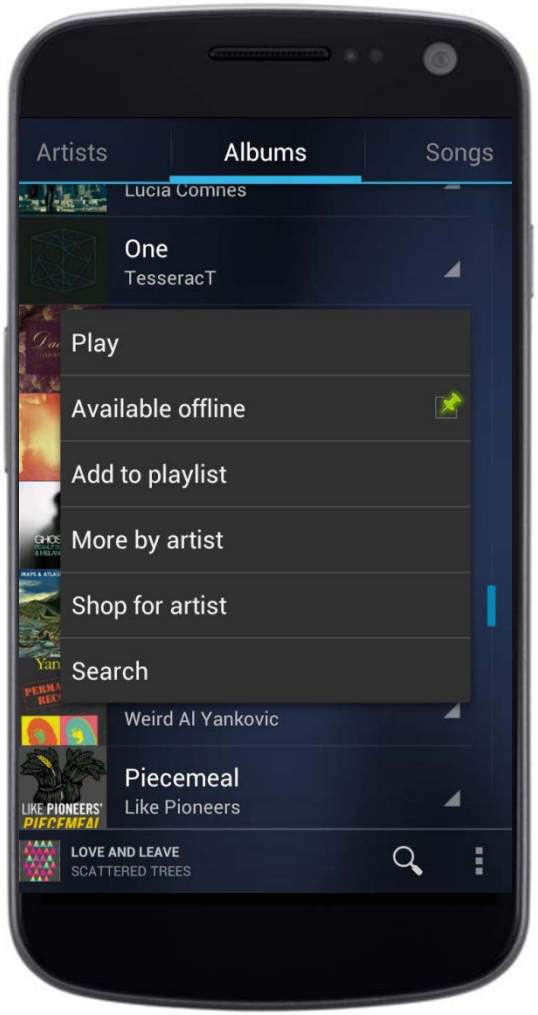
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
  - Activities can use Services to perform long-duration operations or access remote resources in the background



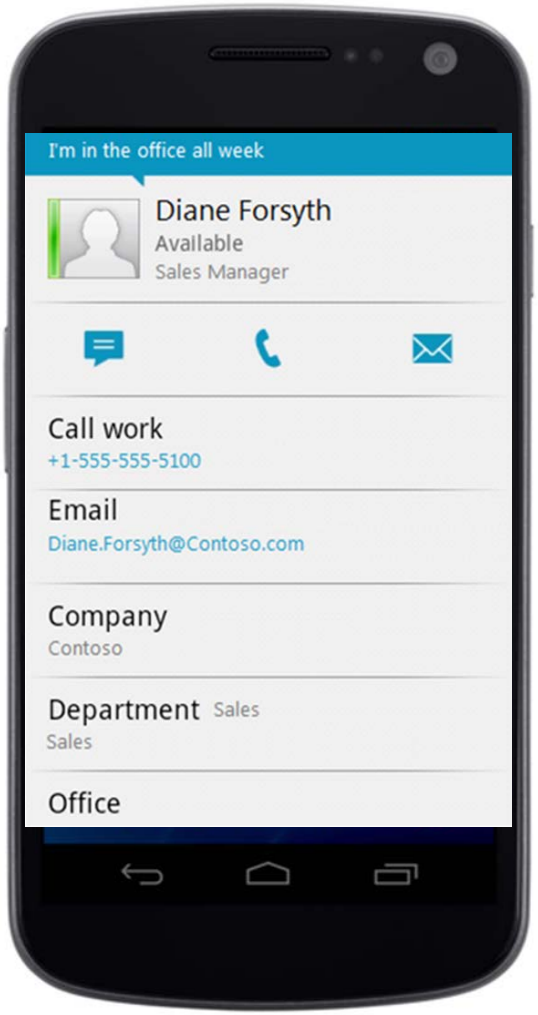
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
  - Activities can use Services to perform long-duration operations or access remote resources in the background, e.g.
    - Play music or videos on a device



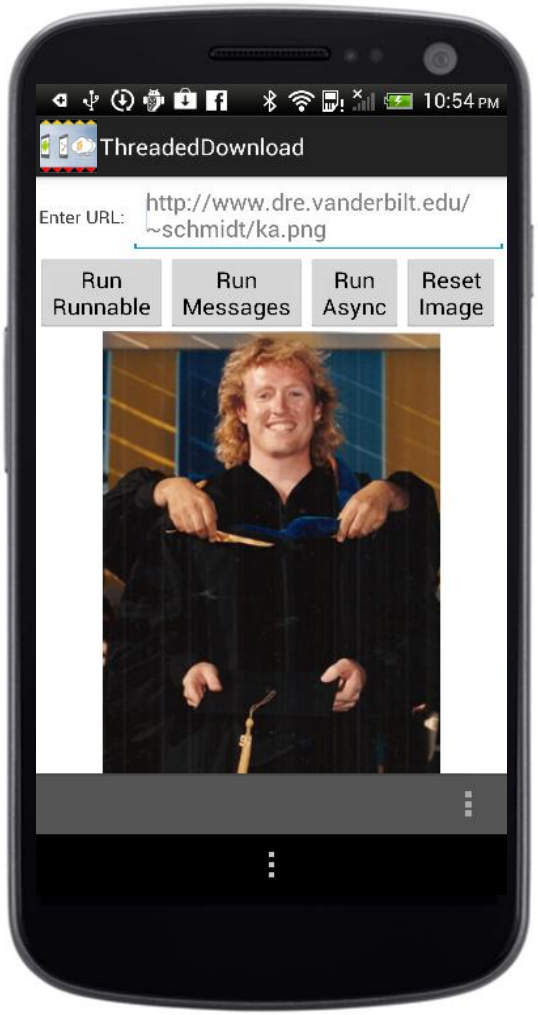
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
  - Activities can use Services to perform long-duration operations or access remote resources in the background, e.g.
    - Play music or videos on a device
    - Synchronize contents of an Email, Contacts, Calendar, or MMS/SMS databases with cloud servers



Overview of Android Services

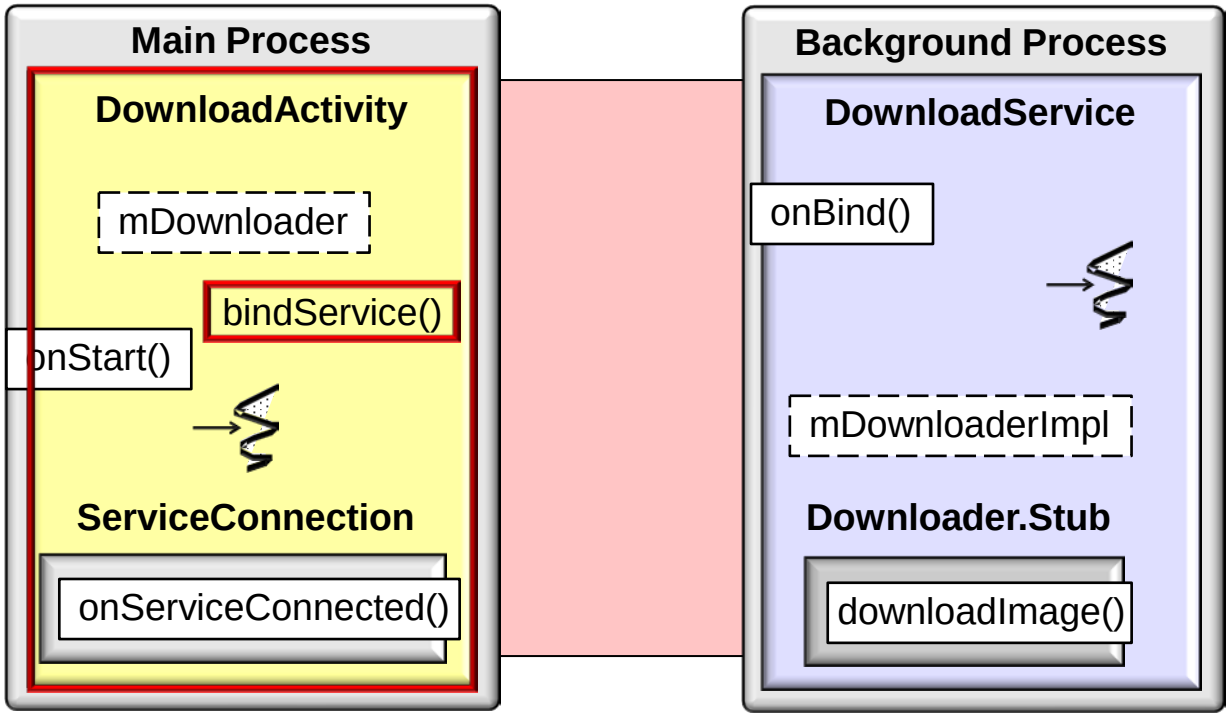
- Unlike Activities, Services don't interact with the user directly
  - Activities can use Services to perform long-duration operations or access remote resources in the background, e.g.
    - Play music or videos on a device
    - Synchronize contents of an Email, Contacts, Calendar, or MMS/SMS databases with cloud servers
    - Download & store images

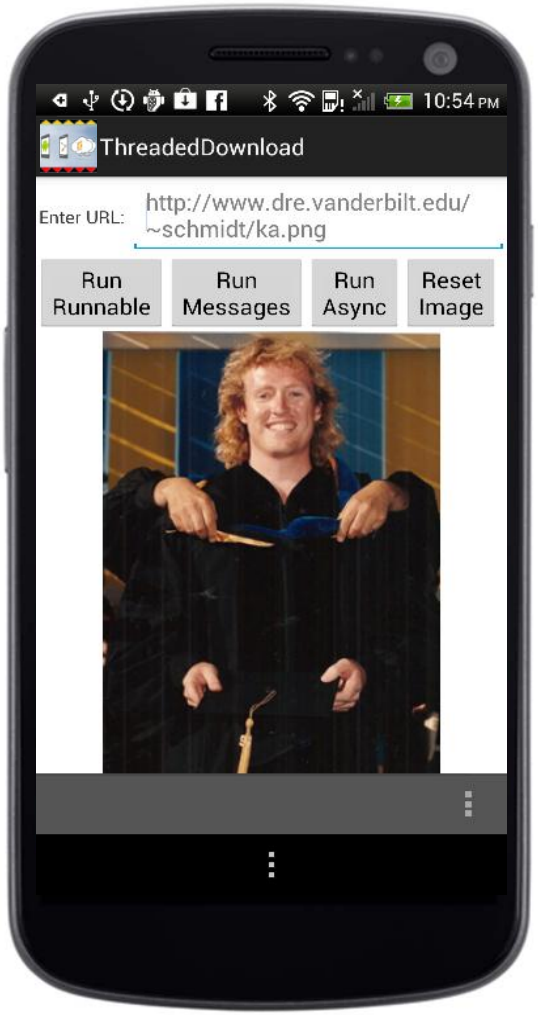


See github.com/douglascraigschmidt/CS282/tree/master/ex/DownloadApplication

Overview of Android Services

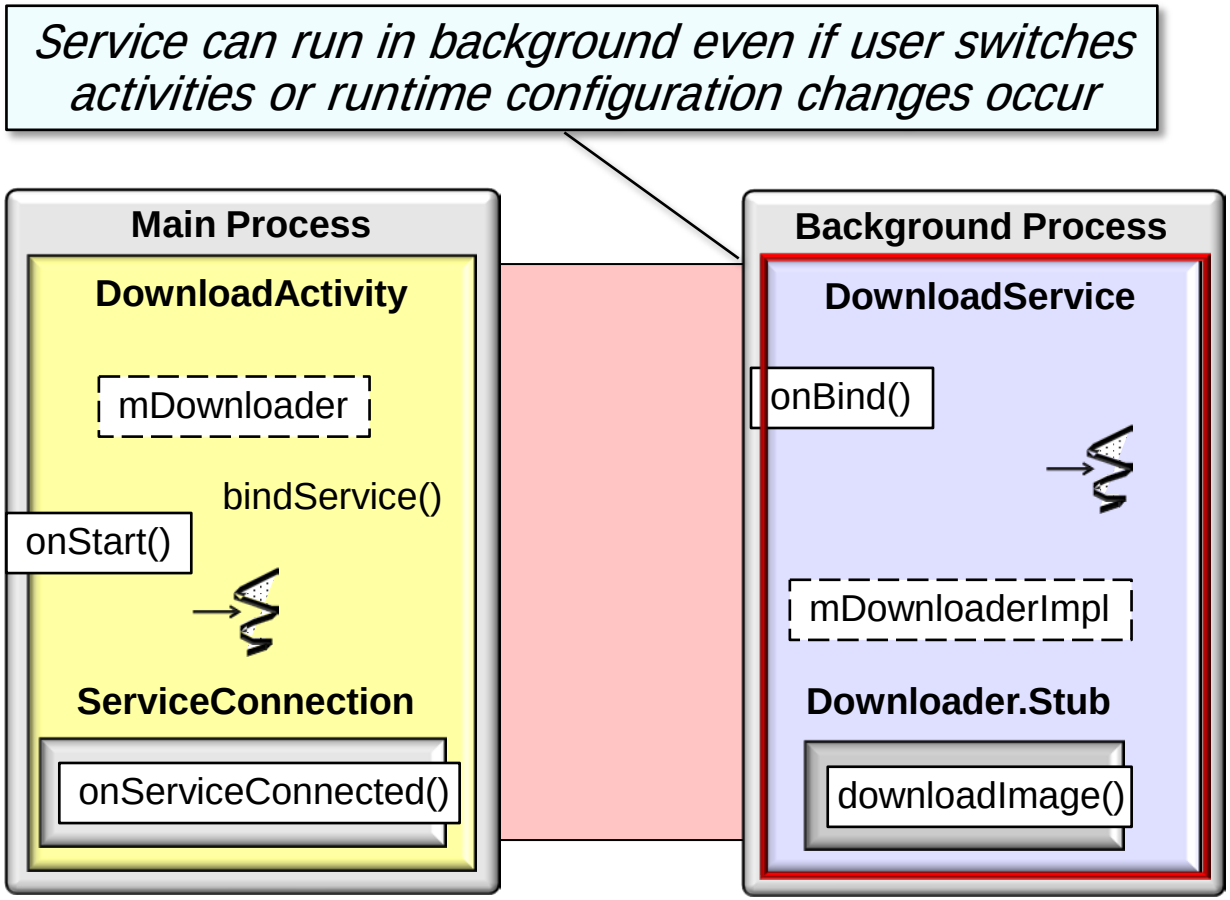
- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services

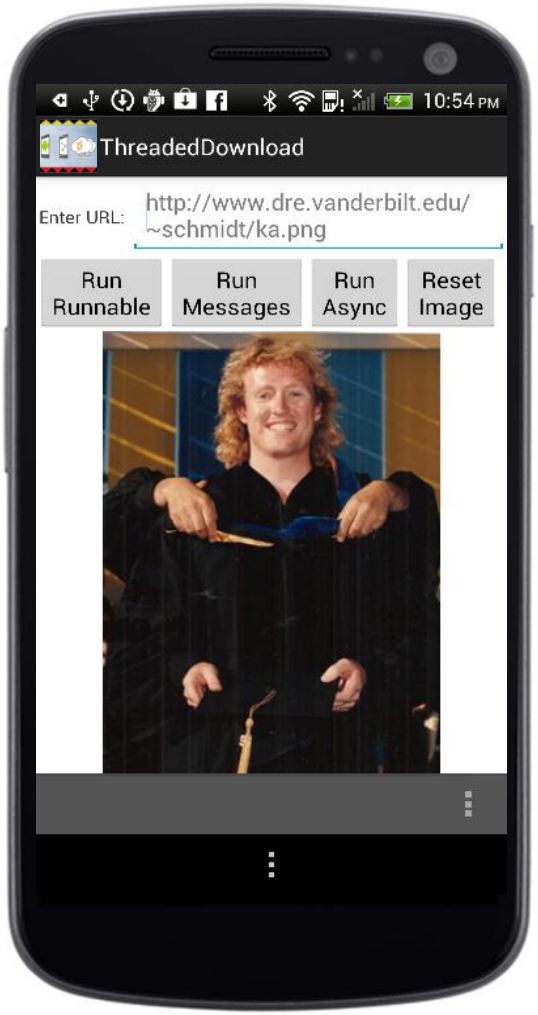




Overview of Android Services

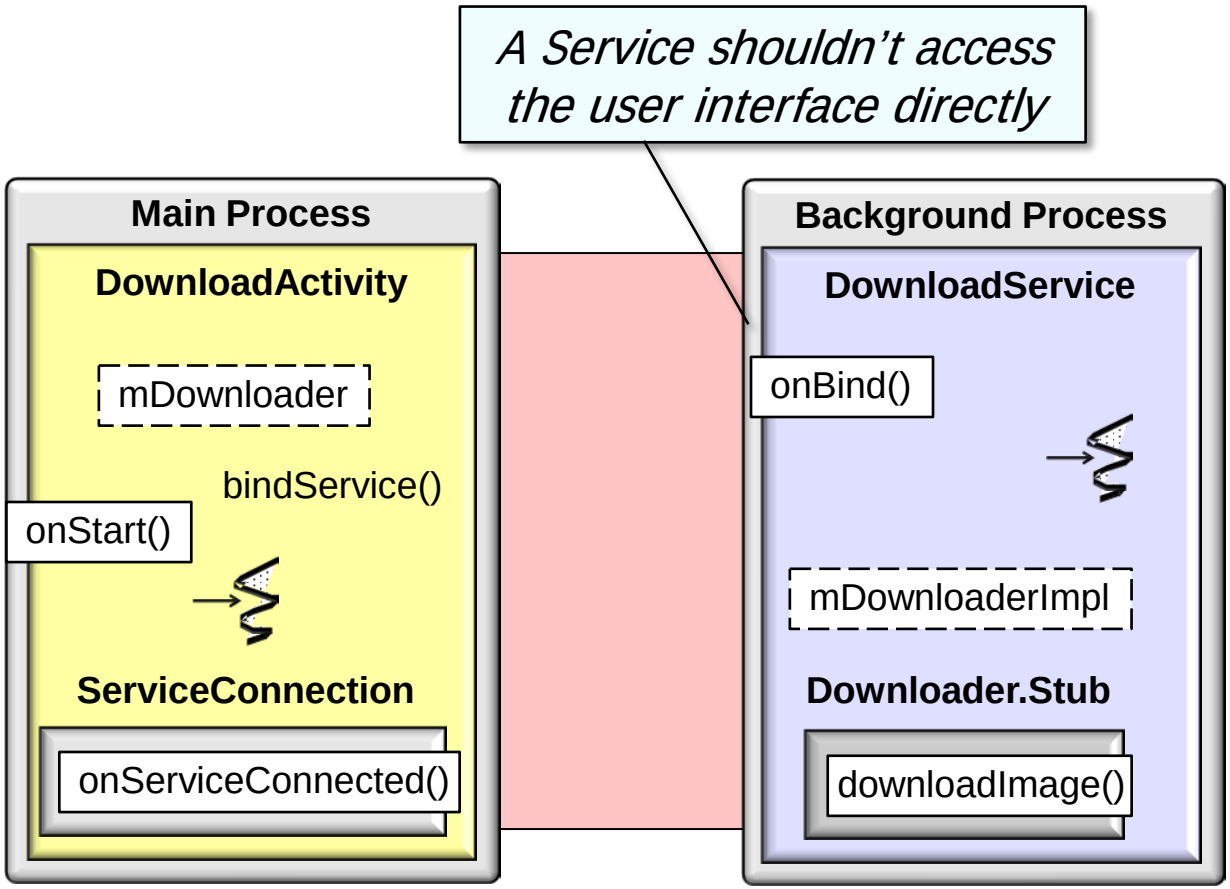
- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services

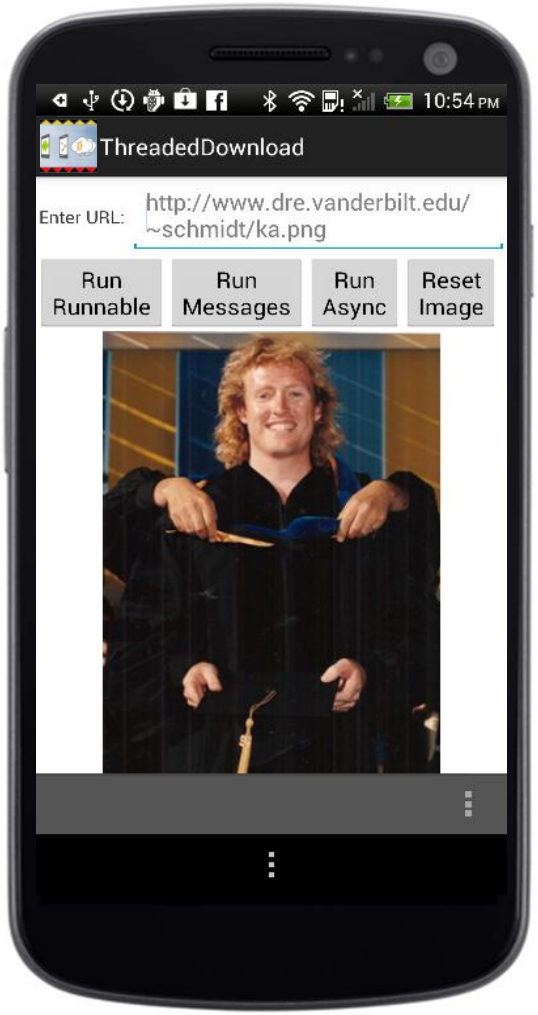




Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services

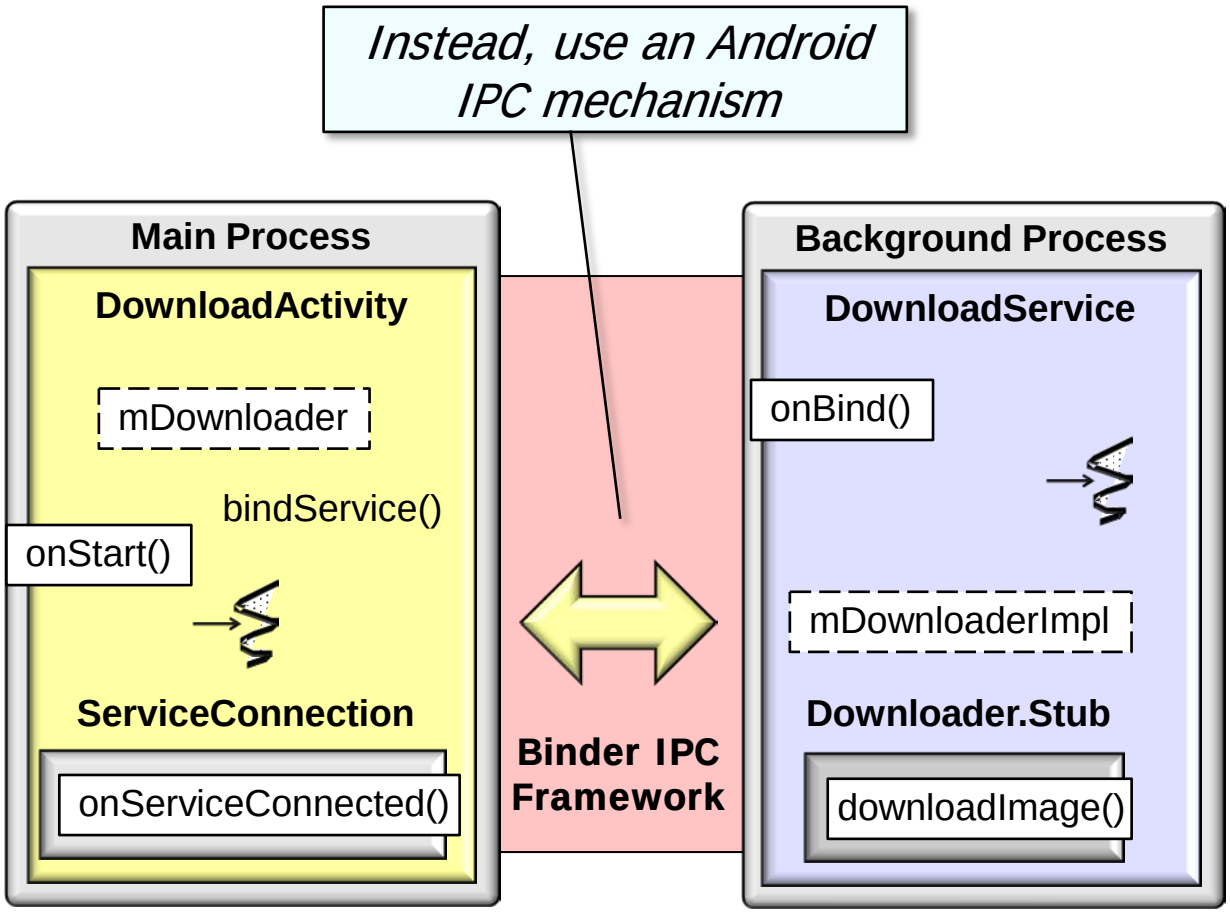


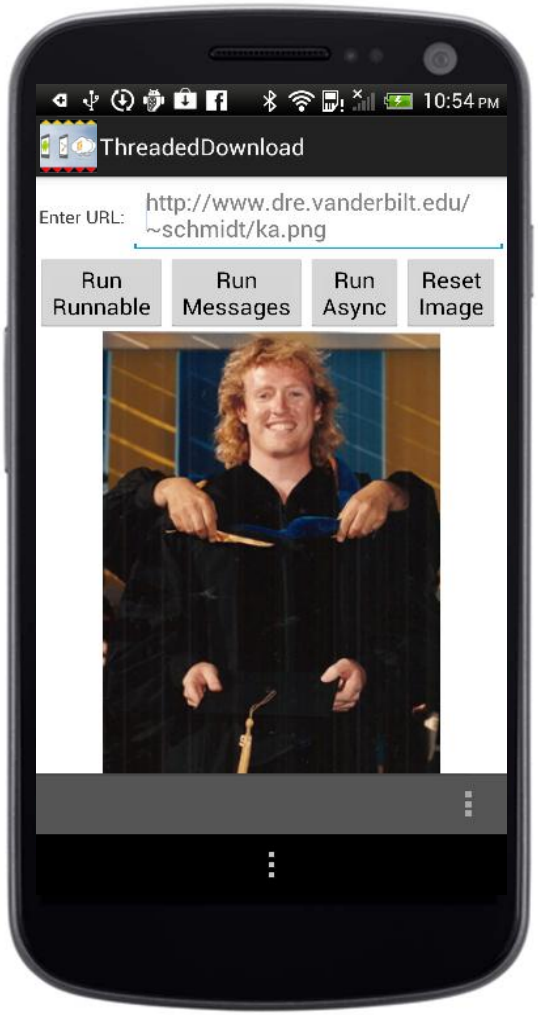


See developer.android.com/training/run-background-service/create-service.html

Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services

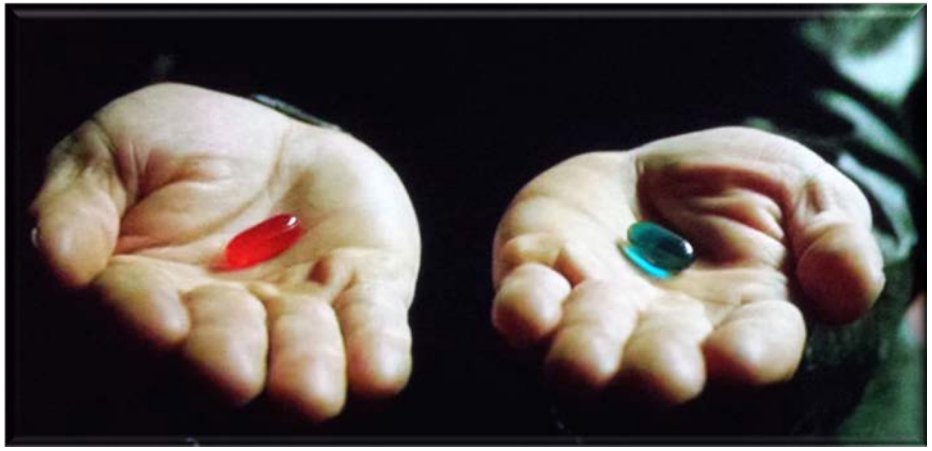


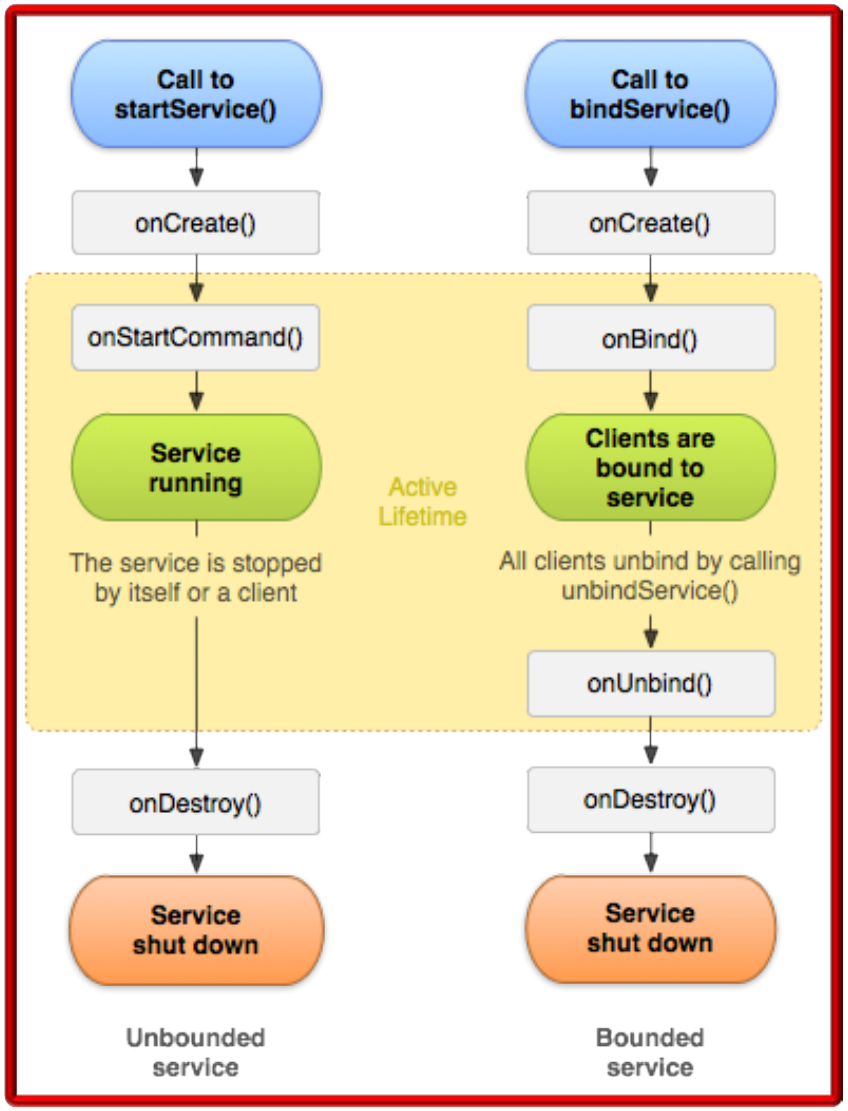


A Service may or may not actually run in a background thread or process

Overview of Android Services

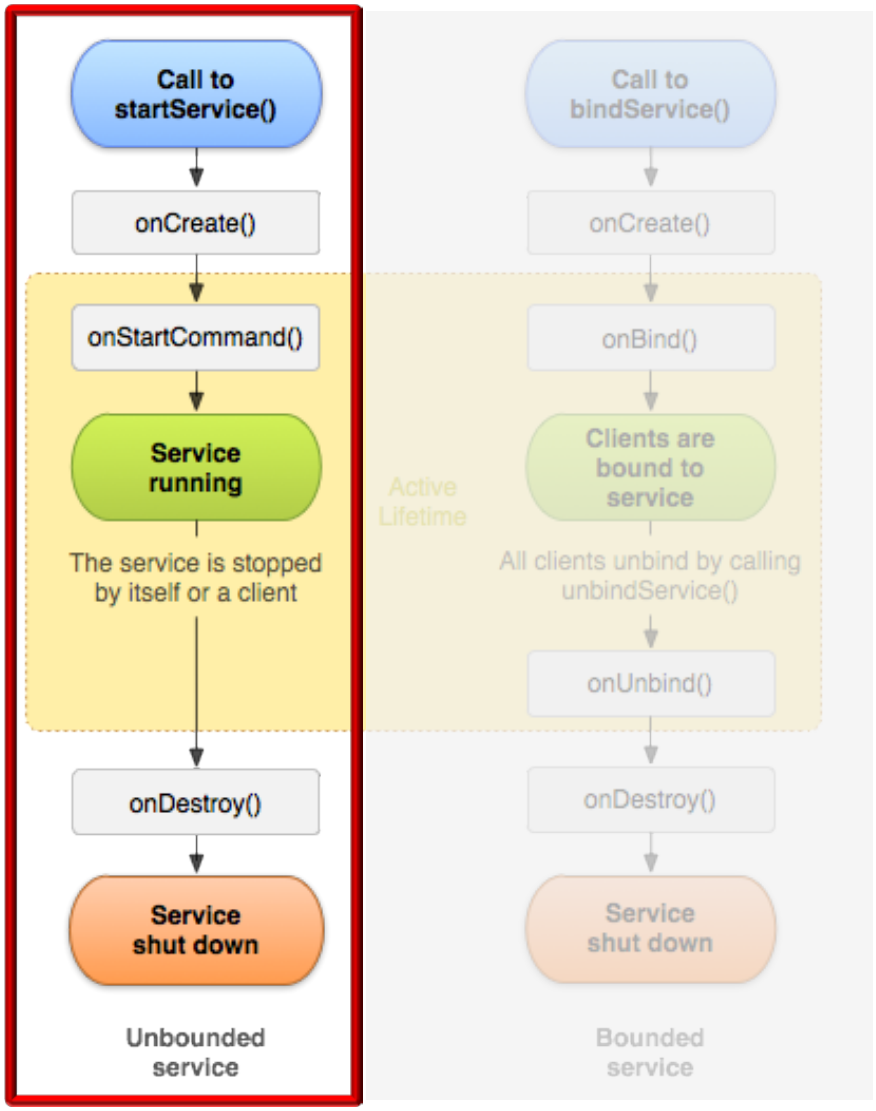
- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services





Overview of Android Services

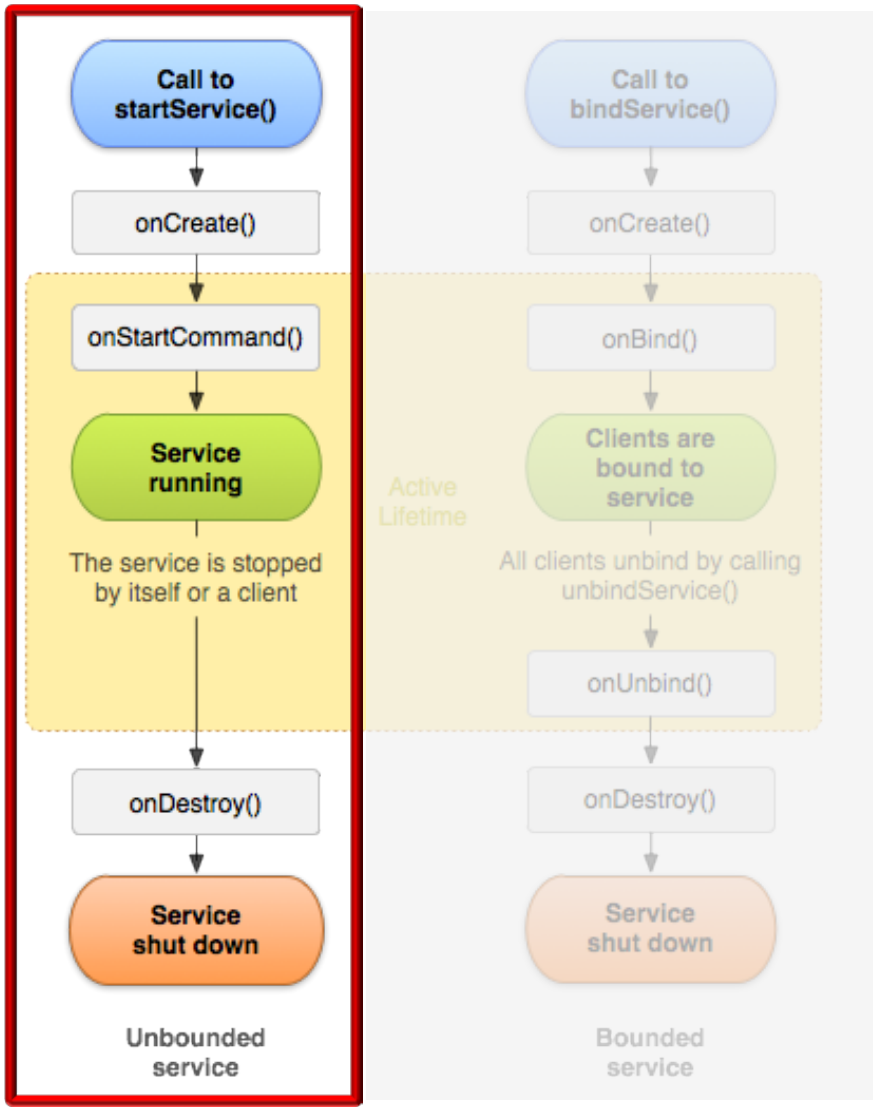
- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*



See developer.android.com/guide/components/services.html#CreatingStartedService

Overview of Android Services

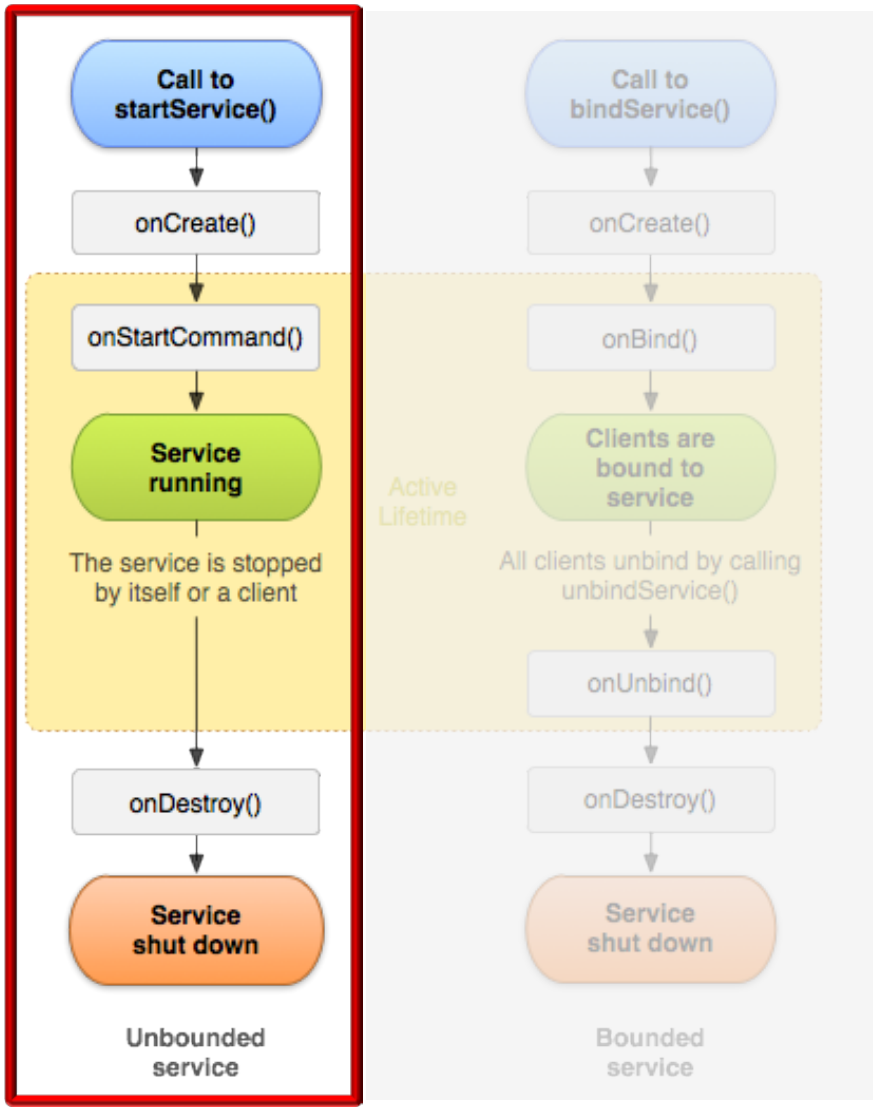
- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
    - Launched via startService()



Parameters can be passed as "extras" to the Intent used to start the Service

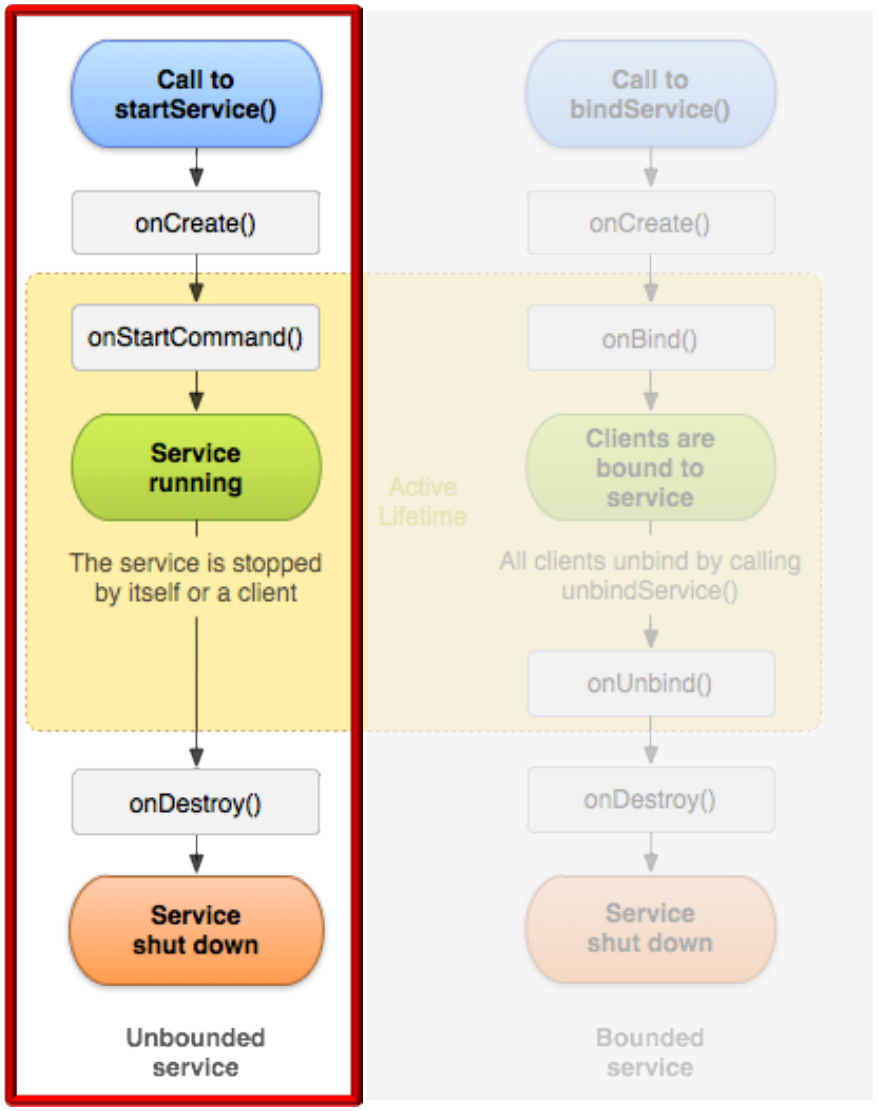
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
    - Launched via startService()
    - Often performs one operation



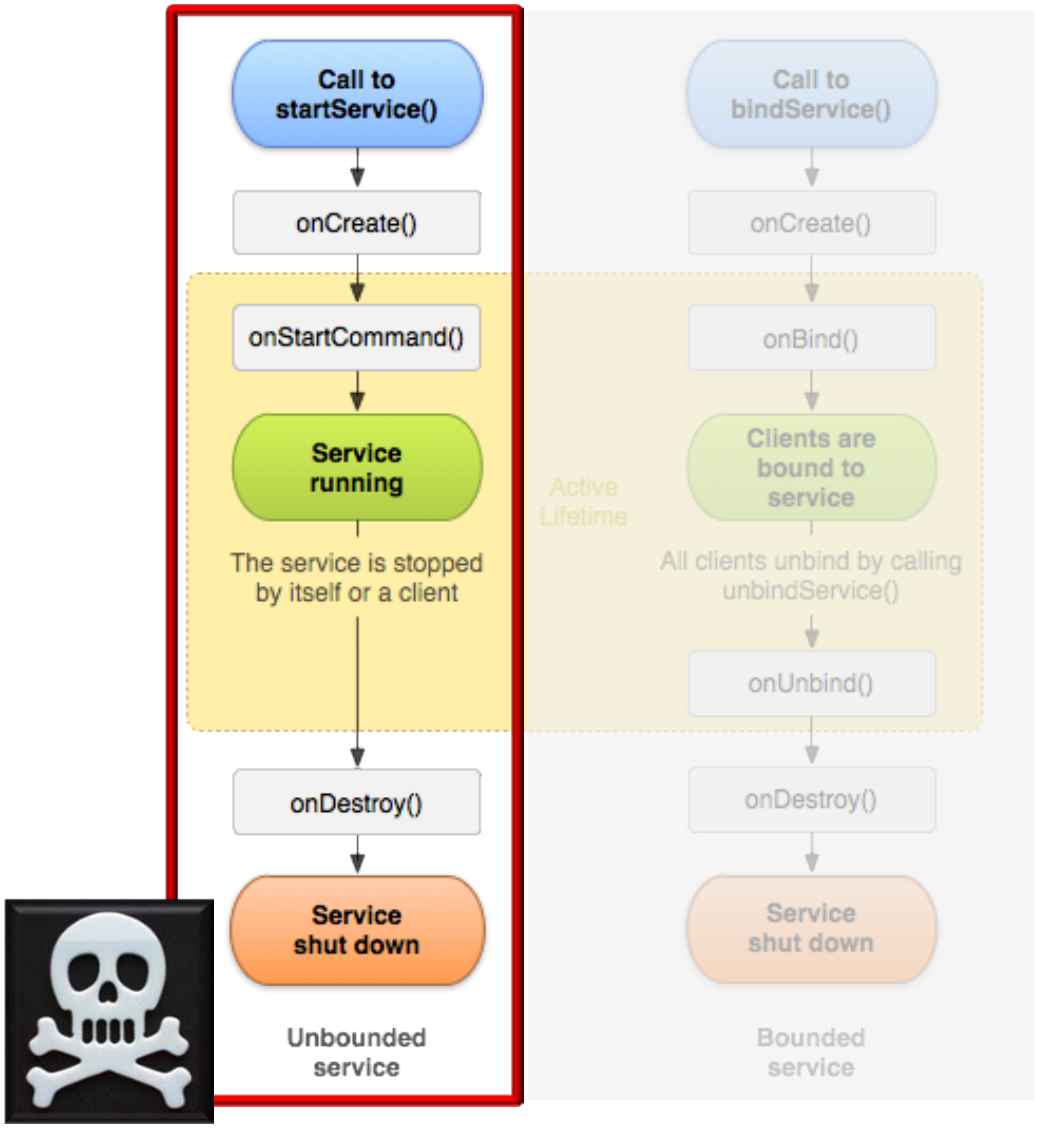
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
    - Launched via startService()
    - Often performs one operation
    - Needn't return a result to the client



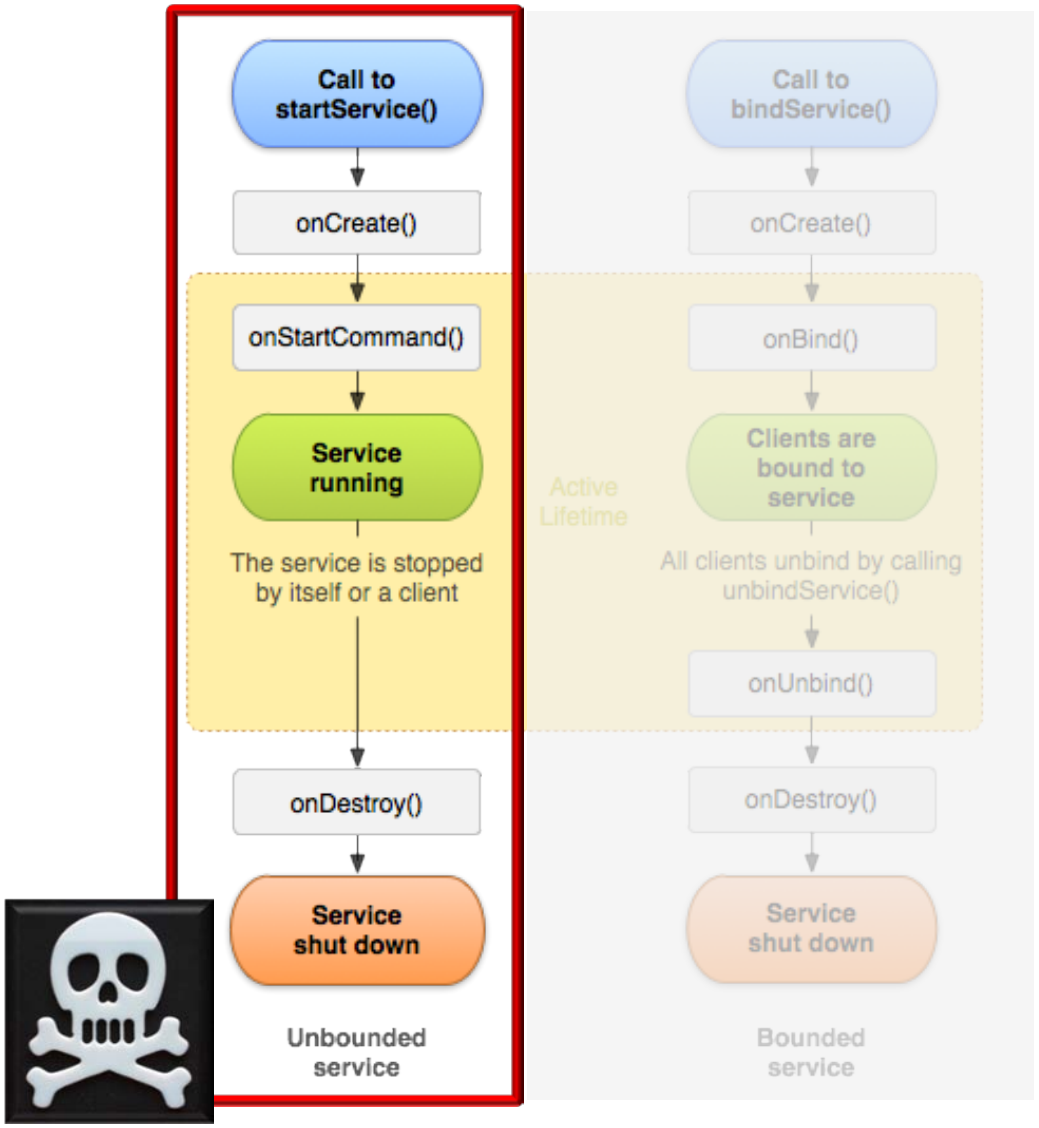
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
    - Launched via startService()
    - Often performs one operation
    - Needn't return a result to the client
    - Typically shuts itself down when it's done



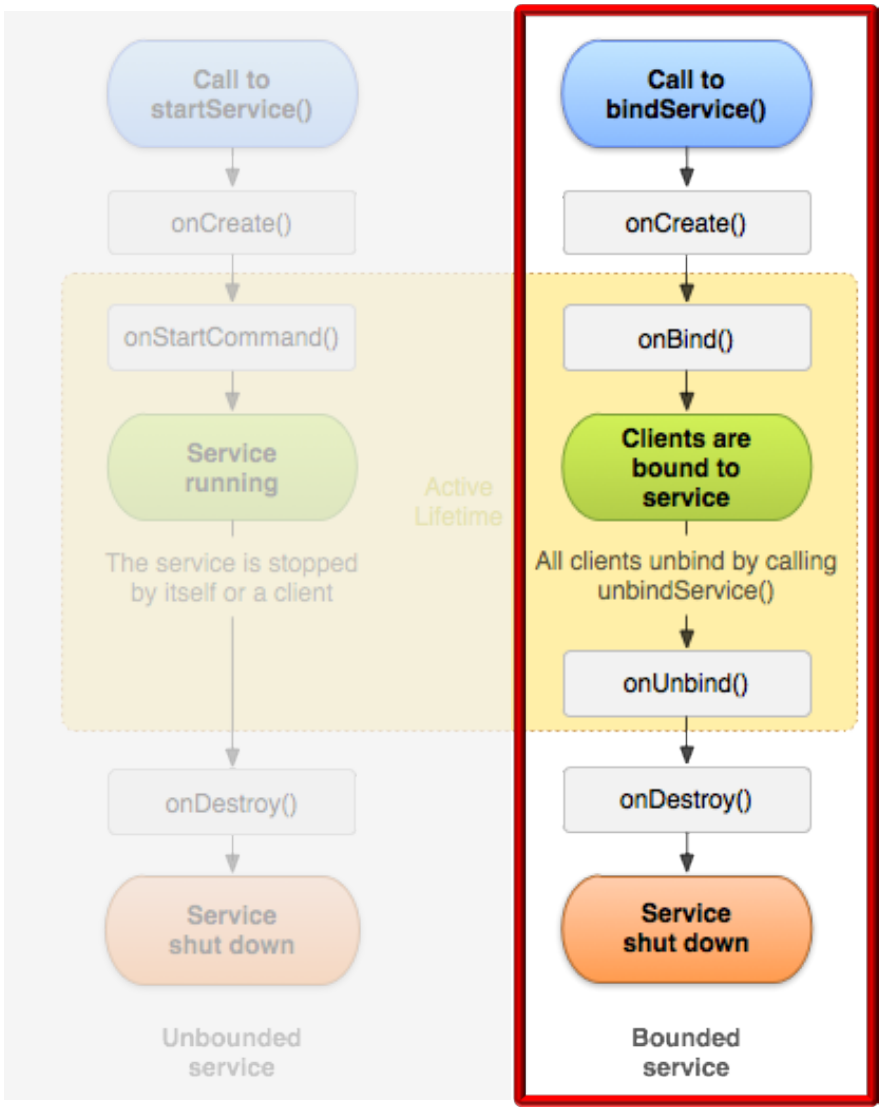
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
    - Launched via startService()
    - Often performs one operation
    - Needn't return a result to the client
    - Typically shuts itself down when it's done
      - Can also be stopped when Activity calls stopService()



Overview of Android Services

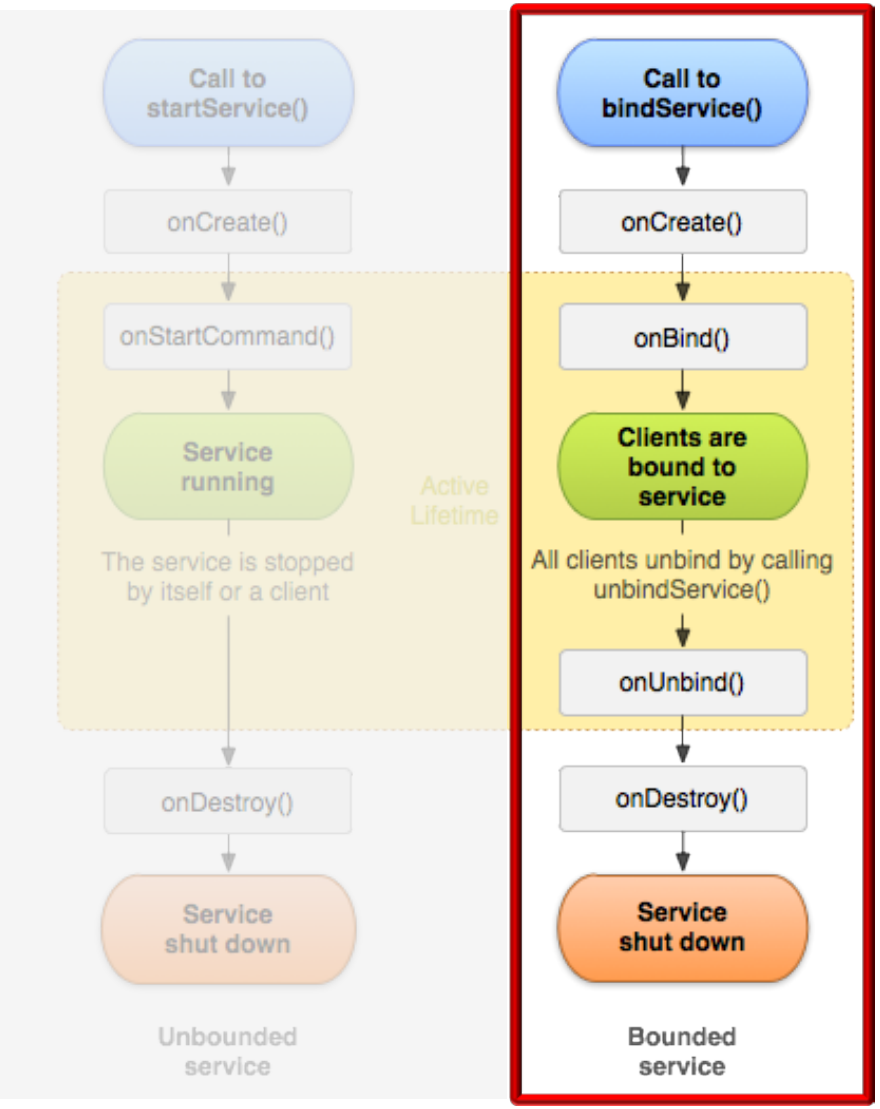
- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
  - *Bound Service*



See developer.android.com/guide/components/bound-services.html

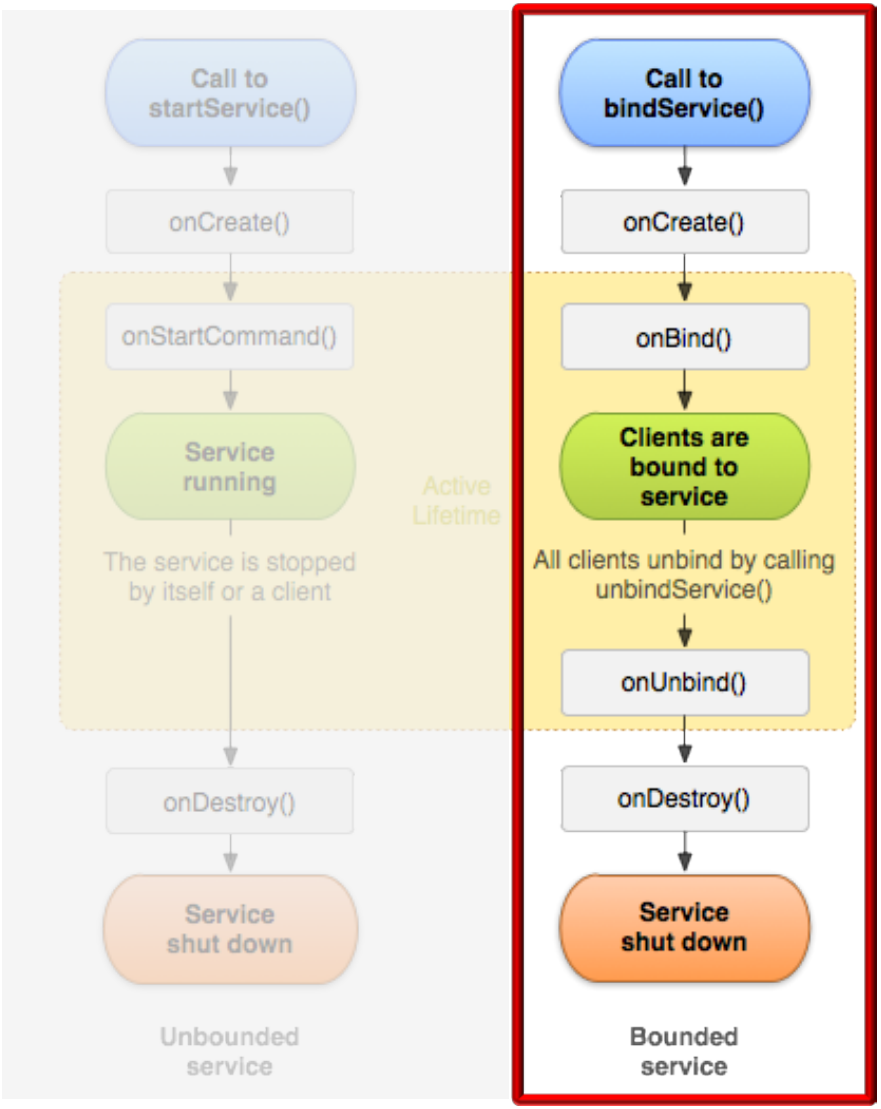
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
  - *Bound Service*
    - Launched via bindService()



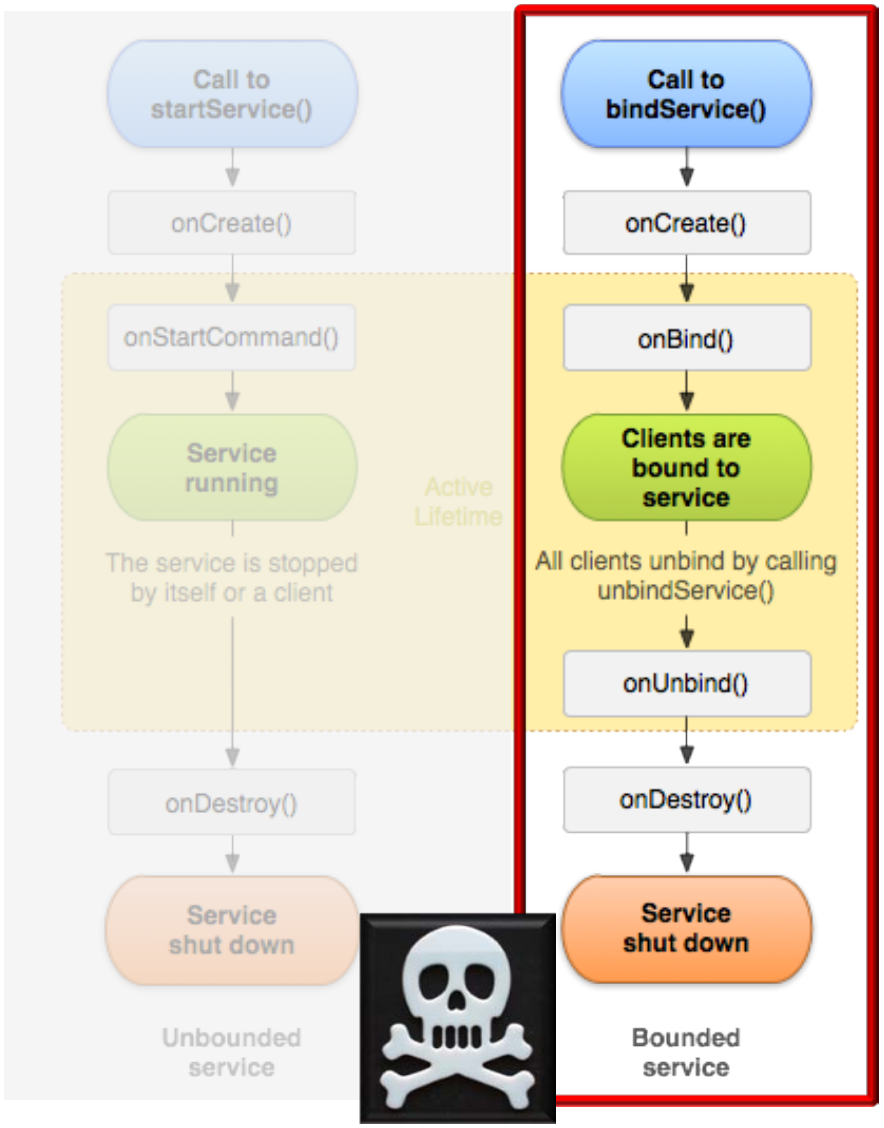
Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
  - *Bound Service*
    - Launched via bindService()
    - Can allow extended two-way conversations between client(s) & the Service



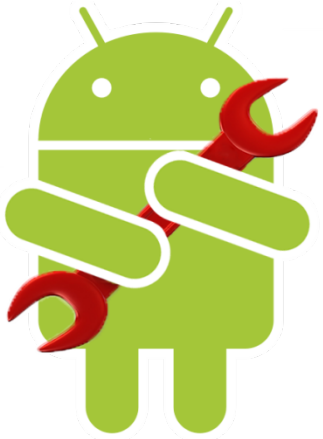
Overview of Android Services

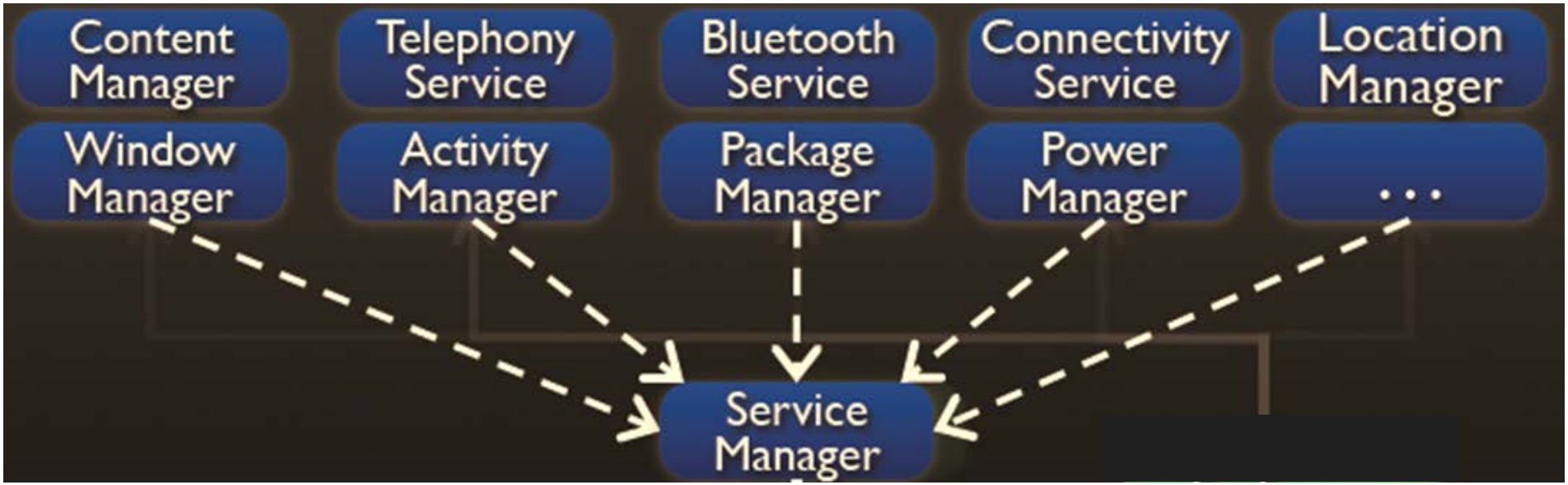
- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
  - *Started Service*
  - *Bound Service*
    - Launched via bindService()
    - Can allow extended two-way conversations between client(s) & the Service
    - Automatically destroyed when all clients unbind



Overview of Android Services

- Unlike Activities, Services don't interact with the user directly
- Activities typically launch Services
- We cover two types of Services
- Android also supports "system services"
  - i.e., expose low-level functions of the hardware & the Linux OS kernel to high-level applications





See www.opensourceforu.com/2013/12/birds-eye-view-android-system-services