Lecture 20: Android Operating System

Instructor: Mitch Neilsen

Office: N219D

* Most slides are from android.developer.com

Quote of the Day

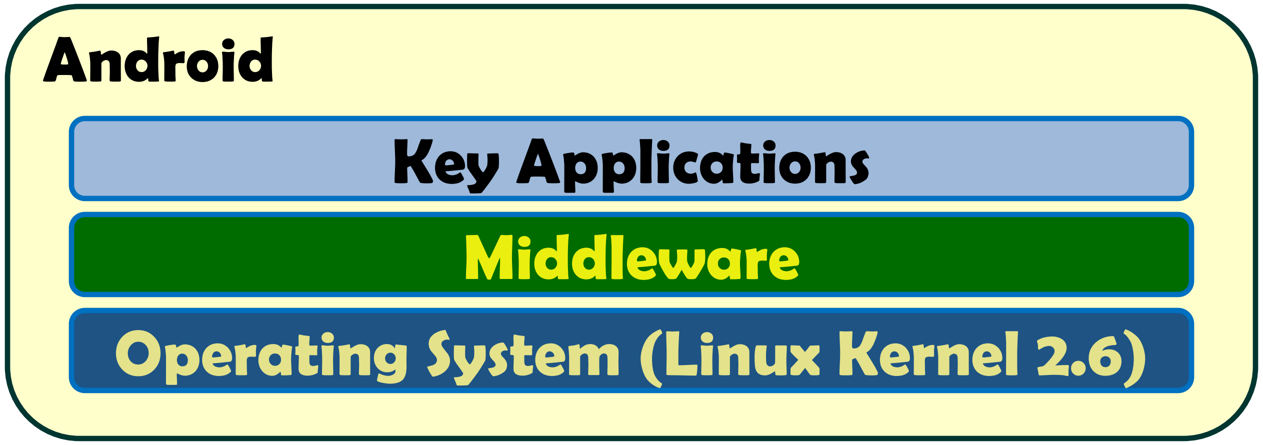
"Android is very different from the GNU/Linux operating system because it contains very little of GNU. Indeed, just about the only component in common between Android and GNU/Linux is Linux, the kernel."

-- Richard Stallman, founder of the GNU Project and the Free Software Foundation

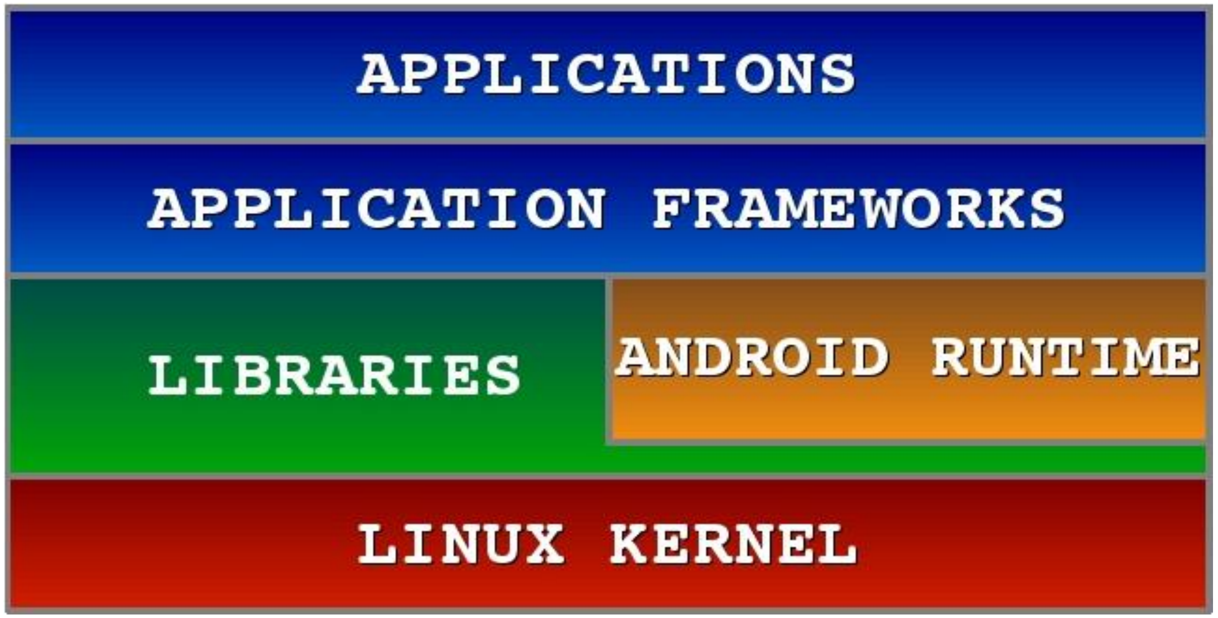
Android

A software stack for mobile devices developed and managed by the Open Handset Alliance (OHA).

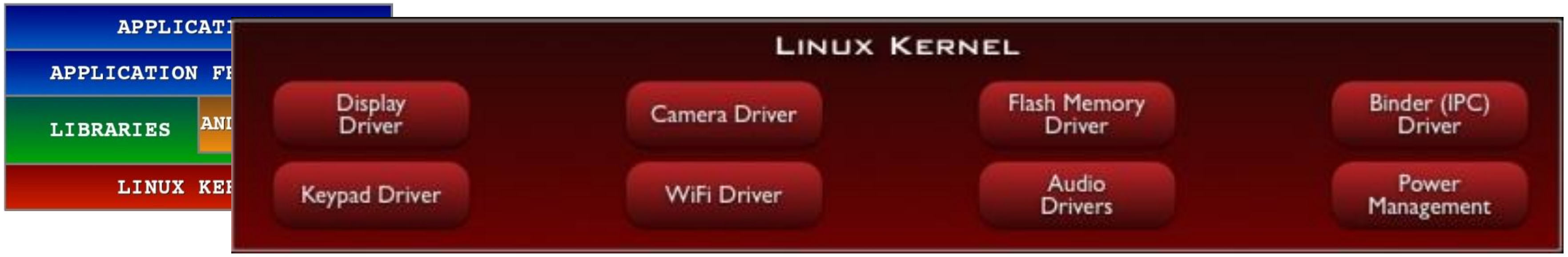
Open source software under an Apache License.



Android Software Stack



Android S/W Stack – Linux Kernel



- Relies on Linux Kernel 2.6 for core system services
  - ✓ Memory and Process Management
  - ✓ Network Protocol Stack
  - ✓ Device Drivers
  - ✓ Security
- Provides an abstraction layer between the H/W and the rest of the S/W stack

Android S/W Stack – Linux Kernel (Cont)

Kernel Enhancements

- ✓ Alarm
- ✓ **Binder**
- ✓ **Power Management**
- ✓ Low Memory Killer
- ✓ Kernel Debugger
- ✓ Logger

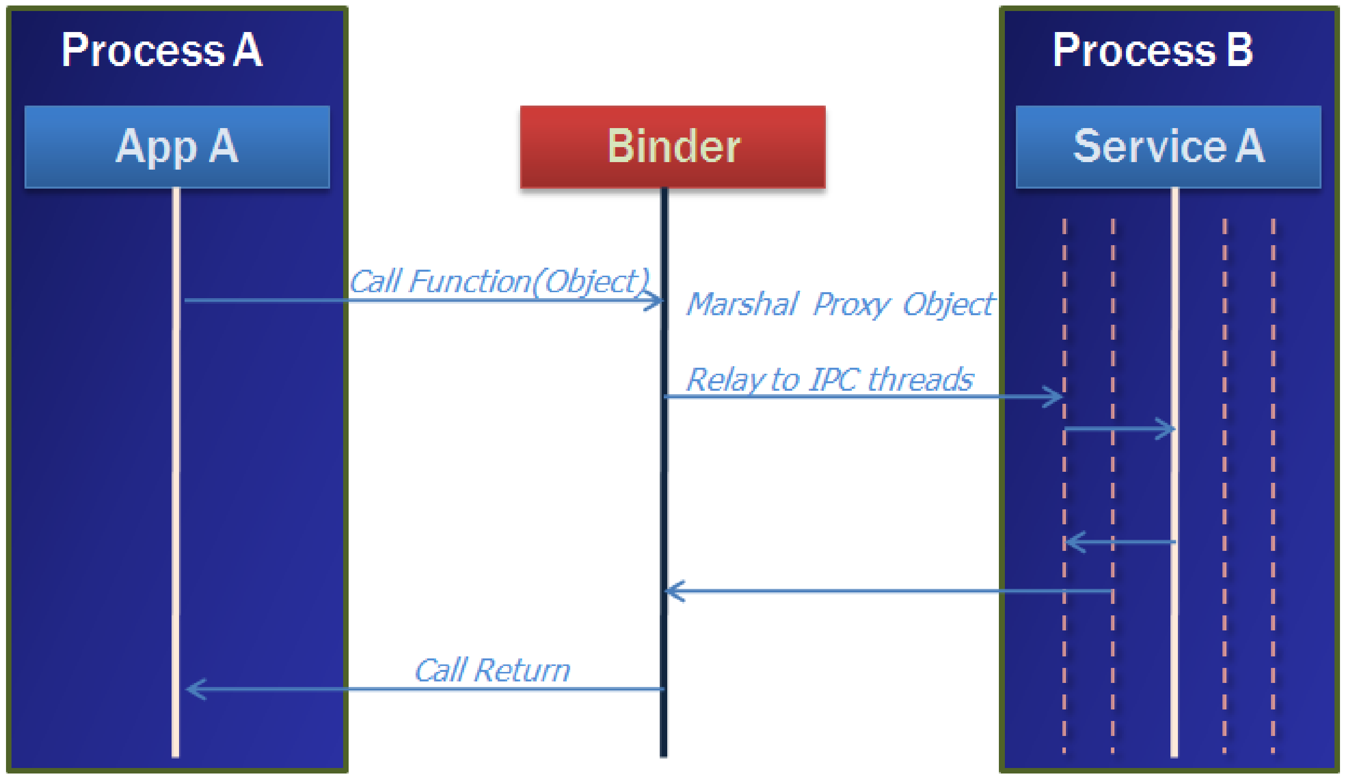
Android S/W Stack – Linux Kernel (Cont)

Binder

- ✓Facilitates IPC between applications and services
- ✓Problems of Linux IPC
  - Applications and Services may run in separate processes but must communicate and share data
  - IPC can introduce significant processing overhead and security issues
- ✓Properties of Binder
  - High performance through shared memory
  - Per-process thread pool for processing requests
  - Reference counting and mapping of object references across processes
  - Synchronous calls between processes

Android S/W Stack – Linux Kernel (Cont)

Binder in Action



- ✓ A pool of threads is associated with each service application to process incoming IPC messages.
- ✓ Binder performs the mapping of an object between two processes.
- ✓ Binder uses an object reference as an address in a process’s memory space.

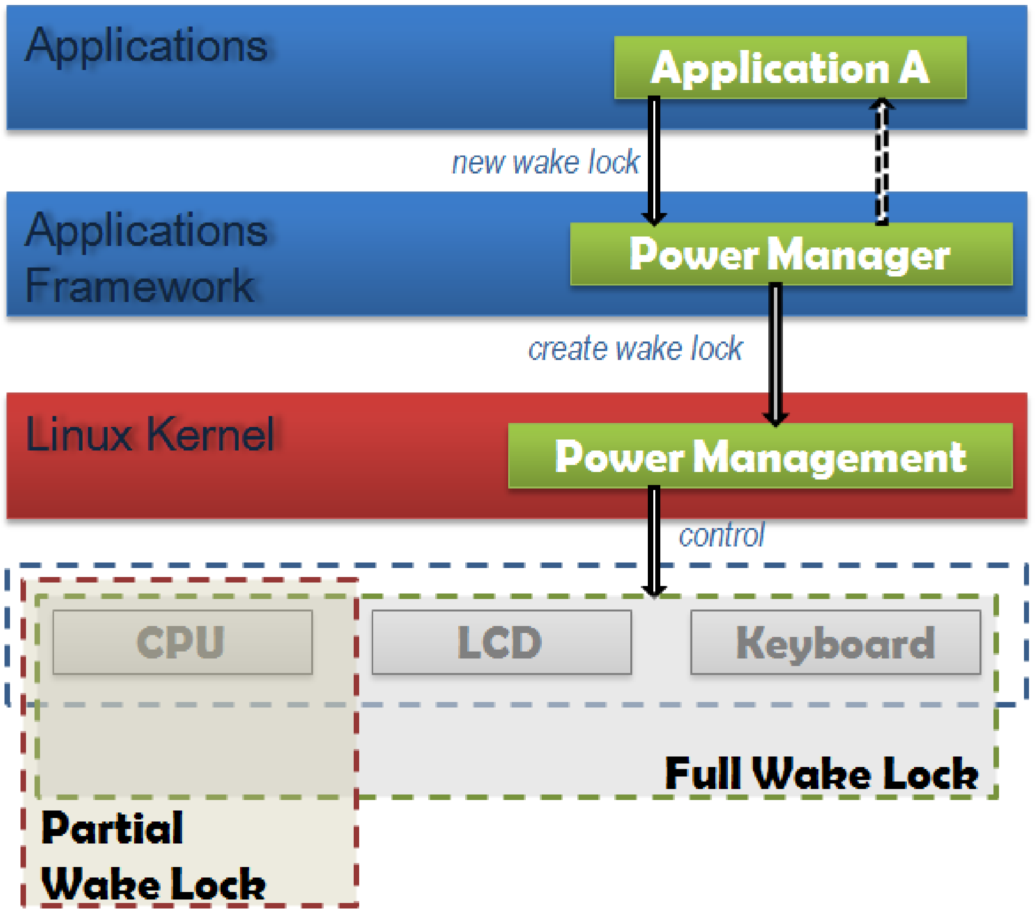
Android S/W Stack – Linux Kernel (Cont)

Power Management

- ✓Problem
  - Mobile devices depend on battery power and batteries have limited capacity.
- ✓Properties of Power Management
  - PM is built on top of standard Linux Power Management.
  - PM supports more aggressive power management policy.
  - Components make requests to keep the power on using "**Wake Locks**".
  - PM supports several different types of wake "Wake Locks".

Android S/W Stack – Linux Kernel (Cont)

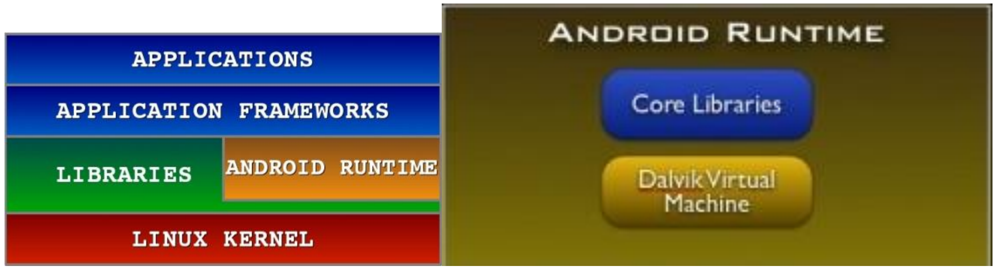
Power Management in Action



✓If there are no active wake locks, CPU will be turned off.

✓If there are only partial wake locks, screen and keyboard will be turned off.

Android S/W Stack - Runtime



Core Libraries

- ✓ Providing most of the functionality available in the core libraries of the Java language
- ✓ APIs
  - Data Structures
  - Utilities
  - File Access
  - Network Access
  - Graphics

Android S/W Stack – Runtime (Cont)

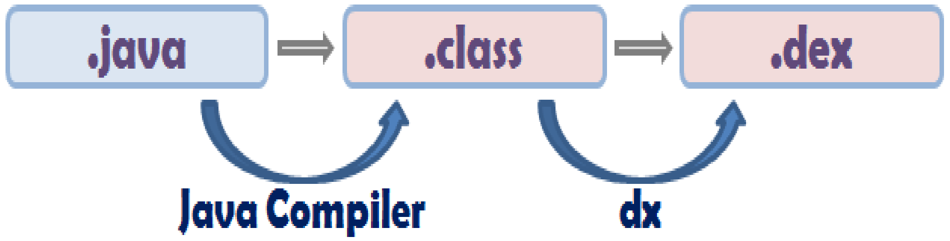
Dalvik Virtual Machine

- ✓**Provides the environment on which every Android application runs**
  - Each Android application runs in its own process, with its own instance of the Dalvik VM.
  - Dalvik has been written so that a device can run multiple VMs efficiently.
- ✓**Register-based virtual machine**

Android S/W Stack – Runtime (Cont)

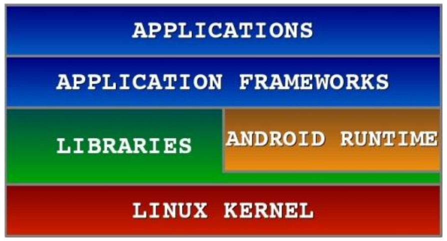
Dalvik Virtual Machine (Cont)

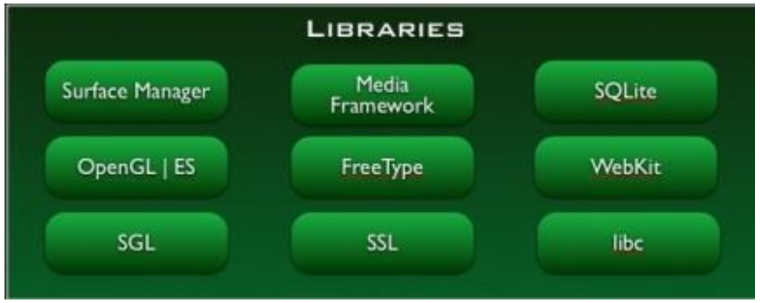
- ✓Executing the Dalvik Executable (.dex) format
  - .dex format is optimized for minimal memory footprint.
  - Compilation



- ✓Relying on the Linux Kernel for:
  - Threading
  - Low-level memory management

Android S/W Stack - Libraries





Libraries

- Includes a set of C/C++ libraries used by components of the Android system
- Exposed to developers through the Android application framework

Android S/W Stack – Libraries (Cont)

Features

- ✓ **System C Library (Bionic)**
- ✓ Media Libraries
- ✓ Surface Manager (Surface Flinger)
- ✓ Audio Manager (Audio Flinger)
- ✓ LibWebCore (WebKit)
- ✓ SGL
- ✓ 3D Libraries
- ✓ FreeType
- ✓ SQLite

Android S/W Stack – Libraries (Cont)

Bionic

- ✓ Custom libc implementation optimized for embedded use
- ✓ Problem with GNU libc

License	The authors want to keep GPL out of user-space.
Size	Libc will load in each process, so it needs to be small.
Speed	Limited CPU power means it needs to be fast.

Android S/W Stack – Libraries (Cont)

Bionic (Cont)

✓Properties

- BSD license
- Small size and fast code paths
- Very fast and small custom pthread implementation
- No support for certain POSIX features
- No compatibility with GNU libc
- Constraint that all native code must be compiled against bionic

Android S/W Stack – Libraries (Cont)

WebKit

- ✓ An application framework that provides foundation for building a web browser based on open source WebKit browser
- ✓ Properties
  - Ability to render pages in full (desktop) view
  - Full CSS, JavaScript, DOM, AJAX support
  - Support for single-column and adaptive view rendering

Android S/W Stack – Libraries (Cont)

Media Framework

- ✓A media framework based on PacketVideo OpenCore platform
- ✓Properties
  - Support for standard video, audio, still-frame formats
  - Support for hardware/software codec plug-ins

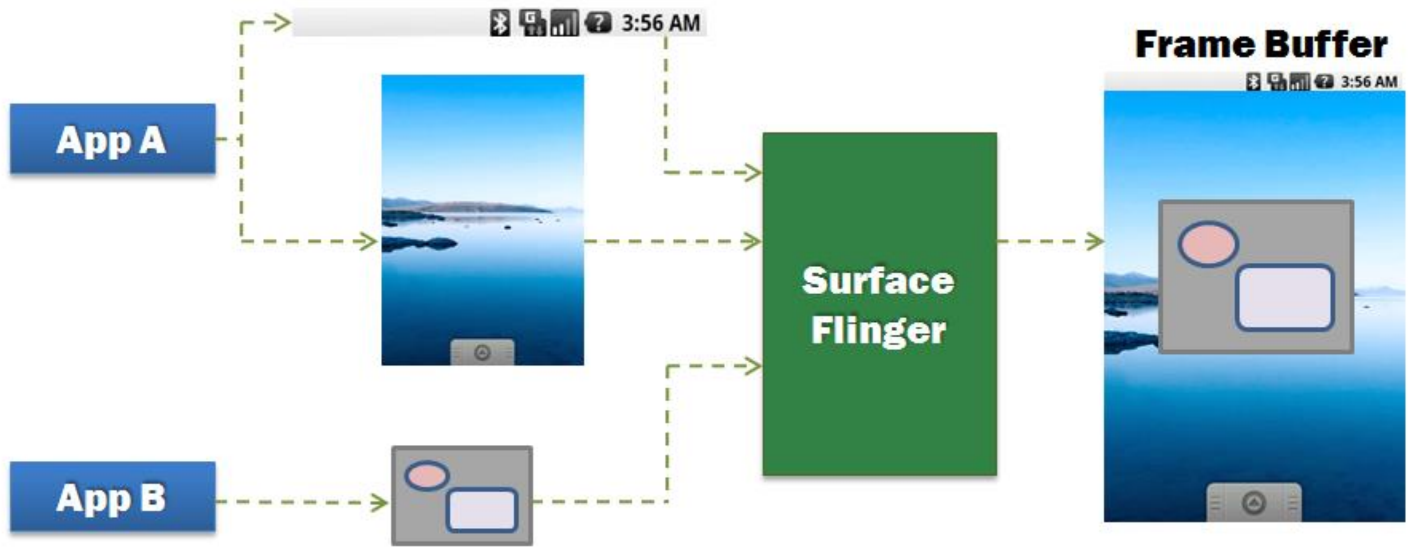
SQLite

- ✓Light-weight relational database management system
- ✓Back end for most platform data storgae

Android S/W Stack – Libraries (Cont)

Surface Manager (Surface Flinger)

- ✓ Provides system-wide surface composer, handling all surface rendering to frame buffer device
- ✓ Operation



Android S/W Stack – Libraries (Cont)

Surface Manager (Cont)

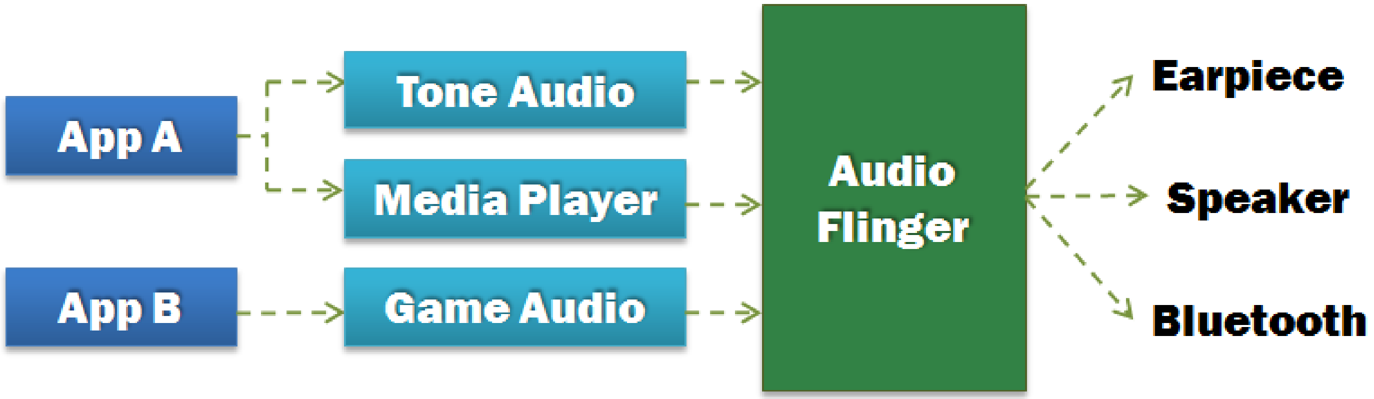
✓Properties

- Can combine 2D and 3D surfaces and surfaces from multiple applications
- Surfaces passed as buffers via Binder IPC calls
- Can use OpenGL ES and 2D hardware accelerator for its compositions
- Double-buffering using page-flip

Android S/W Stack – Libraries (Cont)

Audio Manager (Audio Flinger)

- ✓ Processing multiple audio streams into PCM audio out paths
- ✓ Operation



Android S/W Stack – Libraries (Cont)

SGL

- ✓ The underlying 2D graphics engine

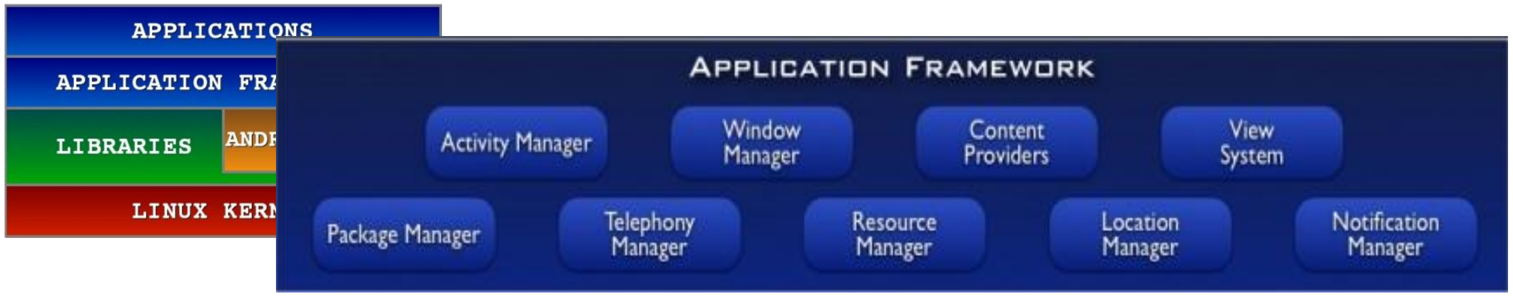
3D Libraries

- ✓ An implementation based on OpenGL ES 1.0 APIs
- ✓ Using either H/W 3D acceleration (if available) or the included optimized 3D S/W rasterizer

FreeType

- ✓ Rendering bitmap and vector font

Android S/W Stack – App Framework



Enabling and simplifying the reuse of components

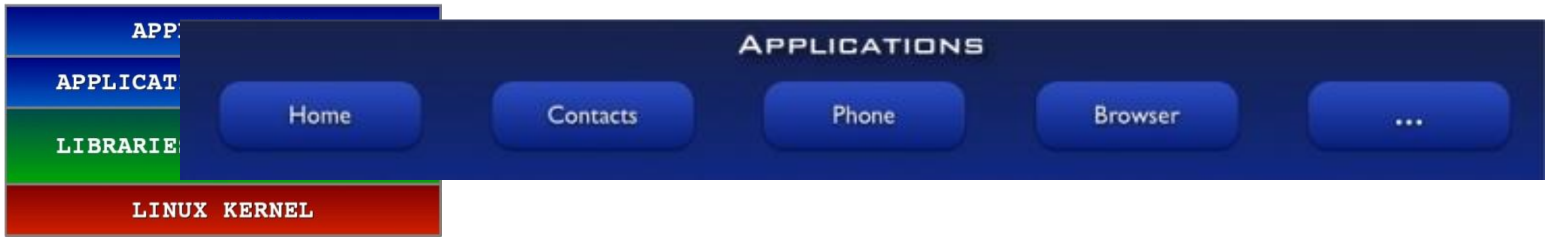
- ✓ Developers have full access to the same framework APIs used by the core applications.
- ✓ Users are allowed to replace components.

Android S/W Stack – App Framework (Cont)

Features

Feature	Role
View System	Used to build an application, including lists, grids, text boxes, buttons, and embedded web browser
Content Provider	Enabling applications to access data from other applications or to share their own data
Resource Manager	Providing access to non-code resources (localized string, graphics, and layout files)
Notification Manager	Enabling all applications to display customer alerts in the status bar
Activity Manager	Managing the lifecycle of applications and providing a common navigation back stack

Android S/W Stack - Application



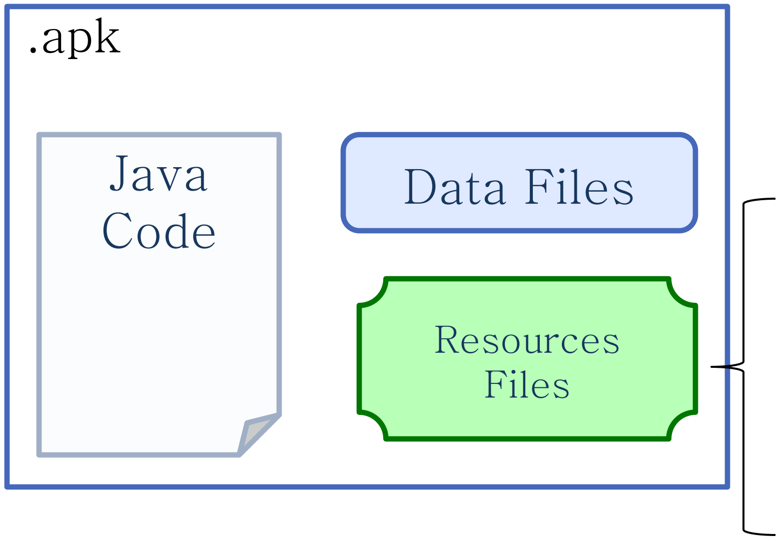
Android provides a set of **Core Applications**:

- ✓ Email Client
- ✓ SMS Program
- ✓ Calendar
- ✓ Maps
- ✓ Browser
- ✓ Contacts
- ✓ Etc

All applications are written using the Java language.

Android Application Package

- Android applications are written in Java.
- An Android application is bundled by the **aapt** tool into an Android package (.apk)



- res/layout: declaration layout files
- res/drawable: intended for drawing
- res/admin: bitmaps, animations for transitions
- res/values: externalized values
  - **strings**, **colors**, **styles**, etc
- res/xml: general XML files used at runtime
- res/raw: binary files (e.g. sound)

Application Components

- Android applications do not have a single entry point (e.g. no main() function).
- They have essential components that the system can instantiate and run as needed.
- **Four basic components**

Components	**Description**
Activity	**UI component typically corresponding to one screen**
Service	**Background process without UI**
Broadcast Receiver	**Component that responds to broadcast Intents**
Content Provider	**Component that enables applications to share data**

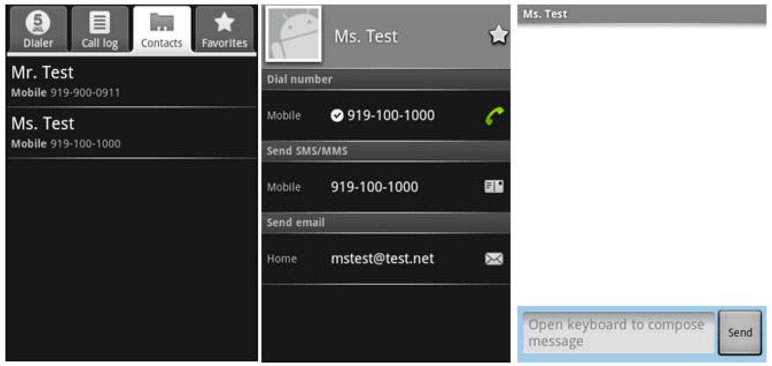
Components - Activity

An **activity** is usually displayed on a single screen:

- Implemented as a single class extending Activity.
- Displays user interface controls (views).
- Reacts to user inputs/events.

An application typically consists of several screens:

- **Each screen is implemented by one activity.**
- Moving to the next screen means starting a new activity.
- An activity may return a result to the previous activity.



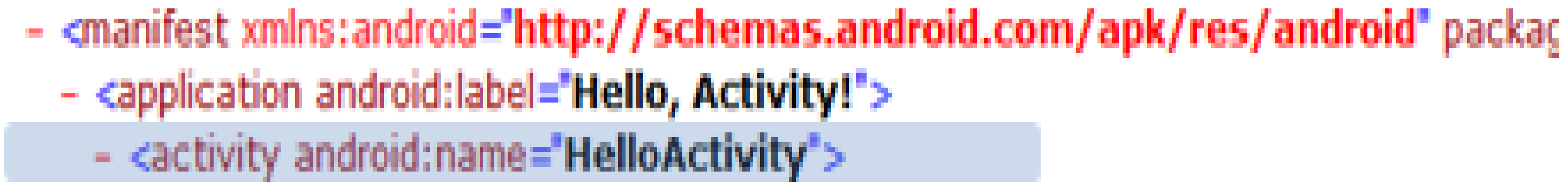
Mr. Test
Mobile 919-900-0911
Ms. Test
Mobile 919-100-1000

Ms. Test
Dial number
Mobile 919-100-1000
Send SMS/MMS
Mobile 919-100-1000
Send email
Home mstest@test.net

Ms. Test
Open keyboard to compose message

Components - Activity (Cont)

- Typically, one of the activities is marked as the first one that should be presented to the user when the application is launched.
- Created "Activity" must be defined in the application's manifest.



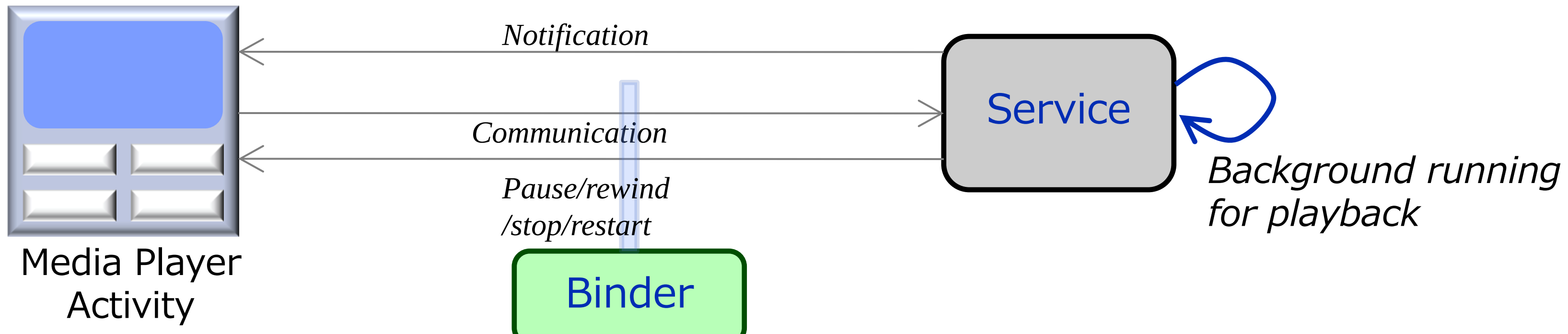
```
- <manifest xmlns:android="http://schemas.android.com/apk/res/android" packag
  - <application android:label="Hello, Activity!">
    - <activity android:name="HelloActivity">
```

Components - Service

A service does not have a visual user interface, but rather runs in the background for an indefinite period time.

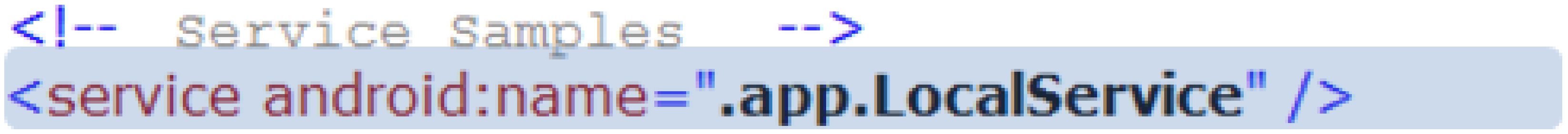
Example: music player, network download, etc.

- Each service extends the Service base class.
- It is possible to bind to a running service and start the service if it's not already running.
- While connected, it is possible communicate with the service through an interface defined in an AIDL (Android Interface Definition Language).



Components - Service (Cont)

Adding a "Service" with Android is quite similar to adding an "Activity".



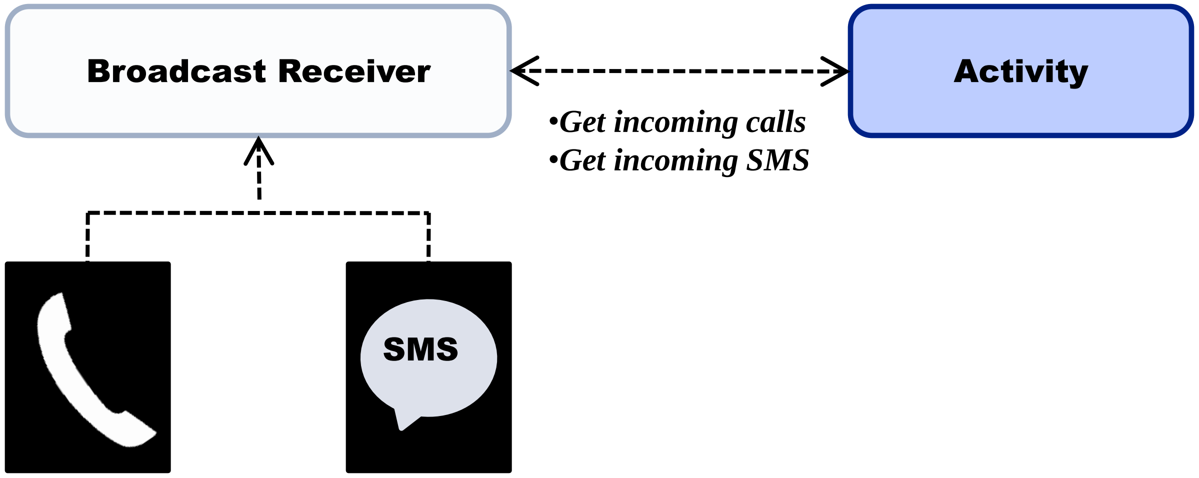
```
<!--  Service Samples   -->
<service android:name=".app.LocalService" />
```

Components - Broadcast Receivers

A **broadcast receiver** is a component that receives and reacts to broadcast announcements (Intents).

- ✓ Many broadcasts originate in system code.

  E.g. announcements that the time zone has changed, that the battery is low, etc.



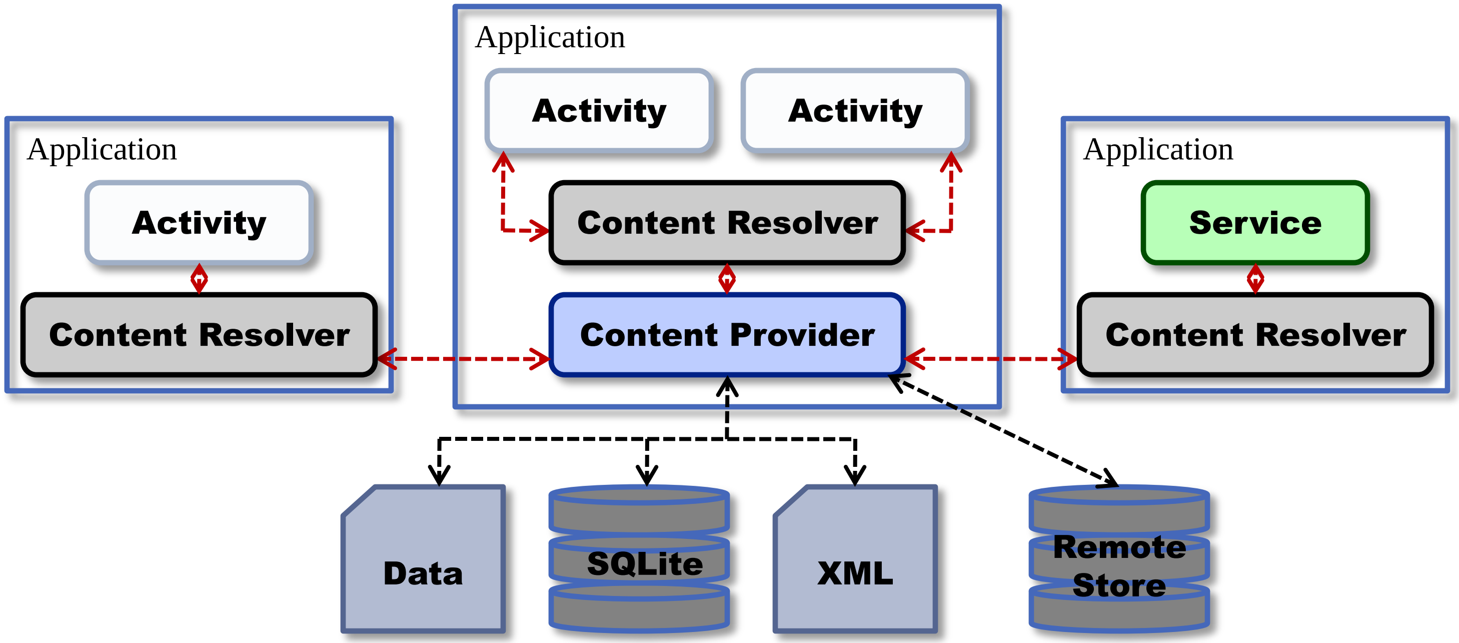
Components - Broadcast Receivers (Cont)

A broadcast receiver is a component that receives and reacts to broadcast announcements.

- ✓Applications can also initiate broadcasts.
  - *E.g. to let other applications know that some data has been downloaded to the device and is available for them to use.*

All receivers extend the ***BroadcastReceiver*** base class.

Components - Content Providers



A content provider makes a specific set of the application's data available to other applications.

- ✓ The data can be stored in the file system, in an SQLite, or in any other manner that makes sense.

Components - Content Providers (Cont)

Using a content provider is the only way to share data between Android applications.

It extends the ContentProvider base class and implements a standard set of methods to allow access to a data store.

- ✓ Querying
- ✓ Delete, update, and insert data

Applications do not call these methods directly.

- ✓ They use a ContentResolver object and call its methods instead.
- ✓ A ContentResolver can talk to any content provider.

Content is represented by URI and MIME type.

Intents

Intents are simple message objects each of which consists of

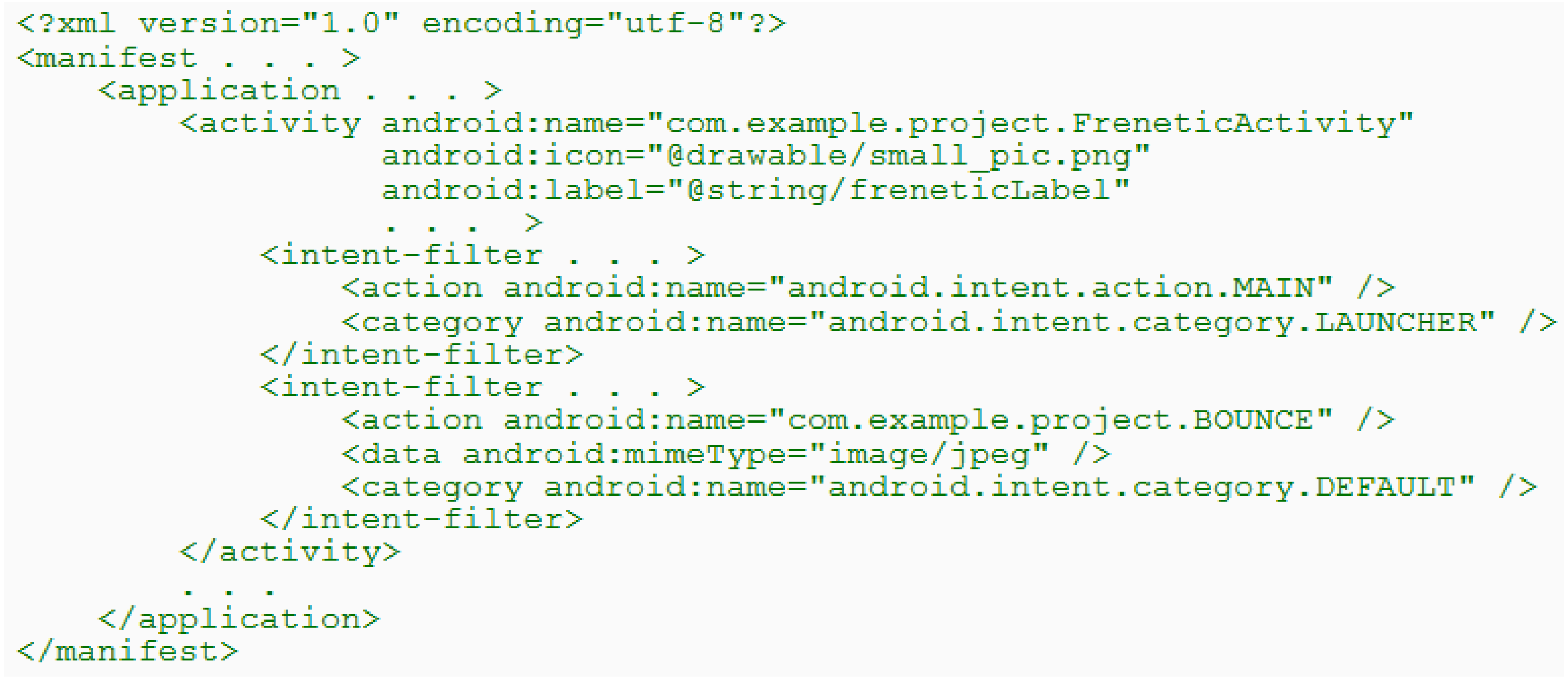
- ✓ Action to be performed (MAIN/VIEW/EDIT/PICK/DELETE/DIAL/etc)
- ✓ Data to operate on (URI)

```
startActivity(new Intent(Intent.VIEW_ACTION, Uri.parse("http://www.fhnw.ch"));

startActivity(new Intent(Intent.VIEW_ACTION, Uri.parse("geo:47.480843,8.211293"));

startActivity(new Intent(Intent.EDIT_ACTION,Uri.parse("content://contacts/people/1"));
```

Intents (Cont)

Intent Filters

- ✓ A component's intent filters in the manifest file inform Android of the kinds of intents the component is able to handle.
- ✓ An example



```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT" />
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

Intents (Cont)

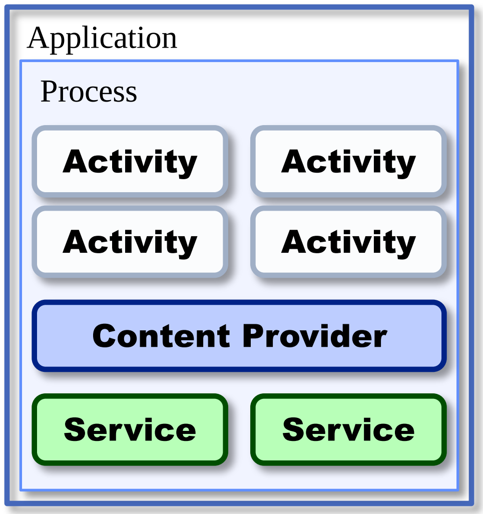
Intent Filters (Cont)

- ✓ An example (Cont)
  - A component can have any number of intent filters, each one declaring a different set of capabilities.
  - The first filter in the example indicates that the activity is the entry point for the application.
  - The second filter declares an action that the activity can perform on a particular type of data.

Android Component Model

An Android application is packaged in an .apk file.

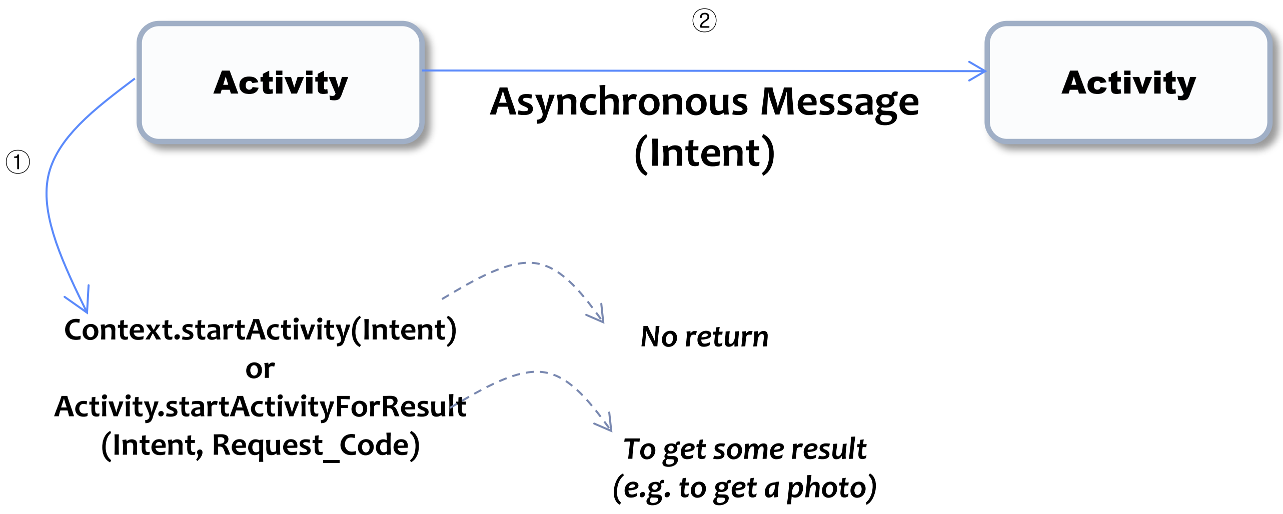
- ✓ An .apk file is a collection of components.



- ✓ Components share a Linux process: by default, one process per .apk file.
- ✓ .apk files are isolated and communicate with each other via Intents.
- ✓ Every component has a managed lifecycle.

Activities and Tasks

One activity can start another, including one defined in a different application.

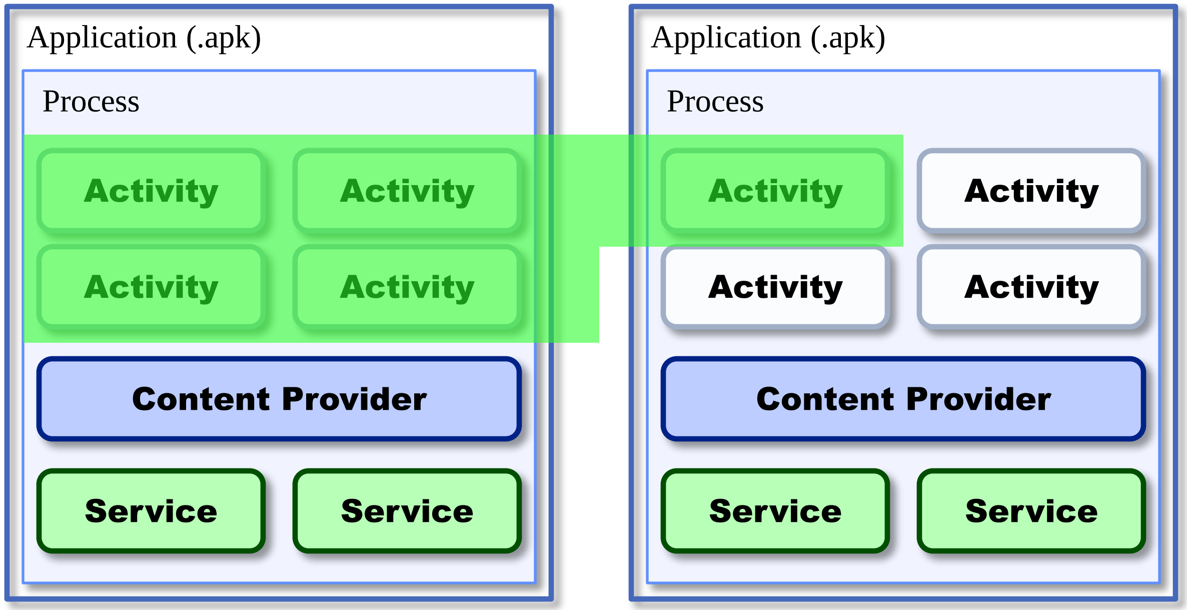


②
Activity
Asynchronous Message (Intent)
Activity
①
Context.startActivity(Intent)
or
Activity.startActivityForResult (Intent, Request_Code)
No return
To get some result (e.g. to get a photo)

Activities and Tasks (Cont)

Tasks

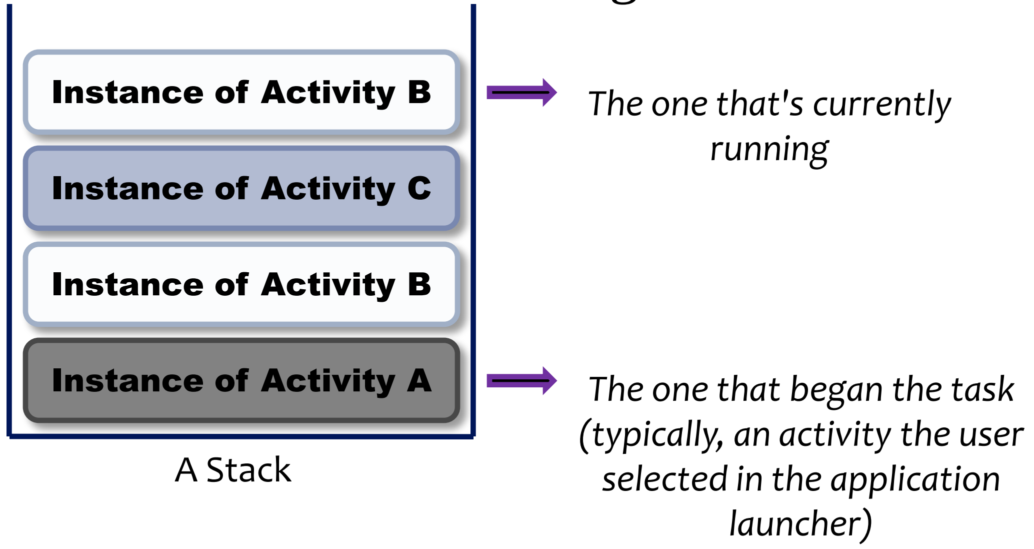
- ✓ A task is a collection of related Activities.
- ✓ It is capable of spanning multiple processes.



Activities and Tasks (Cont)

Tasks (Cont)

✓ All activities in a task are arranged in a stack.



Instance of Activity B
Instance of Activity C
Instance of Activity B
Instance of Activity A
A Stack
The one that's currently running
The one that began the task (typically, an activity the user selected in the application launcher)

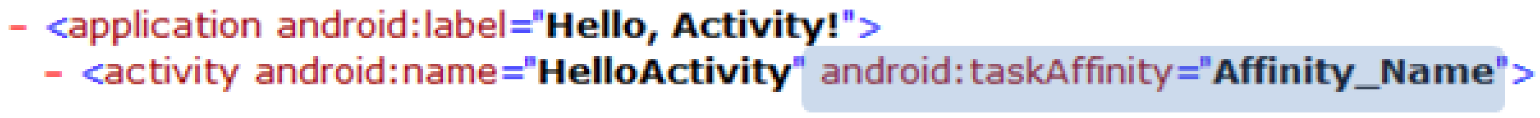
① If one activity starts another, the new activity is pushed on the stack and it becomes the running activity.

② When the user presses the BACK key, the current activity is popped from the stack and the previous one resumes.

Activities and Tasks (Cont)

Affinities

- ✓ An **affinity** means a preference for each activity to belong to a certain task.
- ✓ An individual affinity can be set for each activity:



```
- <application android:label="Hello, Activity!">
  - <activity android:name="HelloActivity" android:taskAffinity="Affinity_Name">
```

- ✓ By default, a new activity is launched into the task of the activity that called startActivity().

Activities and Tasks (Cont)

Affinities (Cont)

✓ Two circumstances where the affinity comes into play:

FLAG_ACTIVITY_NEW_TASK flag

If the Intent object passed to startActivity() contains the FLAG_ACTIVITY_NEW_TASK flag, the system looks for a different task to house the new activity.

If there's already an existing task with the same affinity as the new activity, the activity is launched into that task.

If not, it begins a new task.

② **allowTaskReparenting** attribute

If an activity has its allowTaskReparenting attribute set to "true", it can move from the task it starts in to the task it has an affinity for when that task comes to the fore.

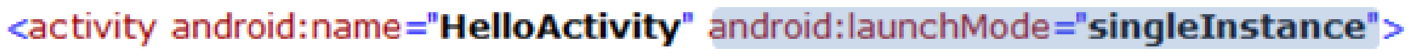
Activities and Tasks (Cont)

Launch Modes

- ✓ There are four launch modes:

  standard (default) / singleTop / singleTask / singleInstance

- ✓ A launch mode can be set for each activity:



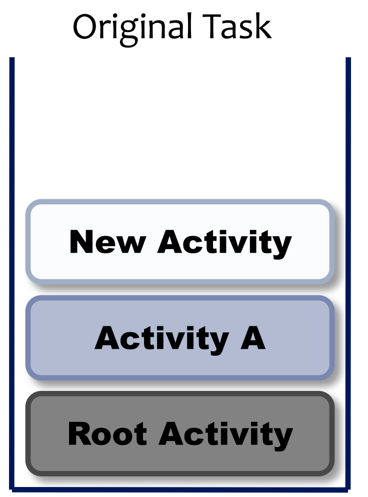
```
<activity android:name="HelloActivity" android:launchMode="singleInstance">
```

Activities and Tasks (Cont)

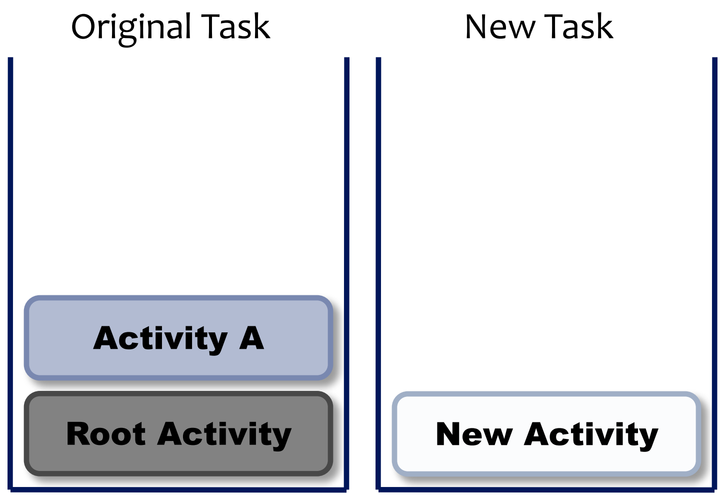
Launch Modes (Cont)

- ✓ The modes differ from each other on four points:

  Which task will hold the activity that responds to the intent



standard/singleTop without FLAG_ACTIVITY_NEW_TASK



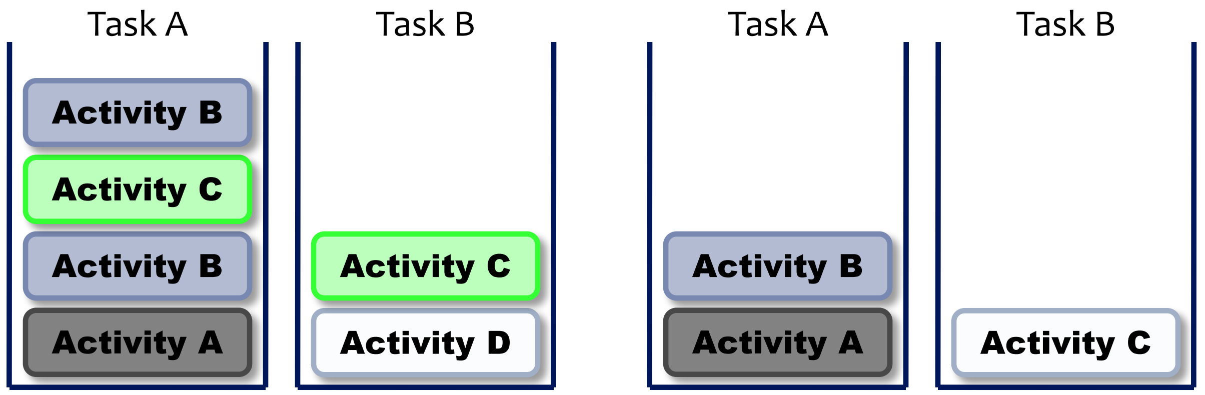
singleTask/singleInstance

Activities and Tasks (Cont)

Launch Modes (Cont)

- ✓ The modes differ from each other on four points: (Cont)

Whether there can be multiple instances of the activity



Activity B and Activity C are standard/singleTop

Activity C is singleTask or singleInstance

A "standard" or "singleTop" activity can be instantiated many times.

A "singleTask" or "singleInstance" activity is limited to just one instance.

Activities and Tasks (Cont)

Launch Modes (Cont)

✓ The modes differ from each other on four points: (Cont)

Whether the instance can have other activities in its task

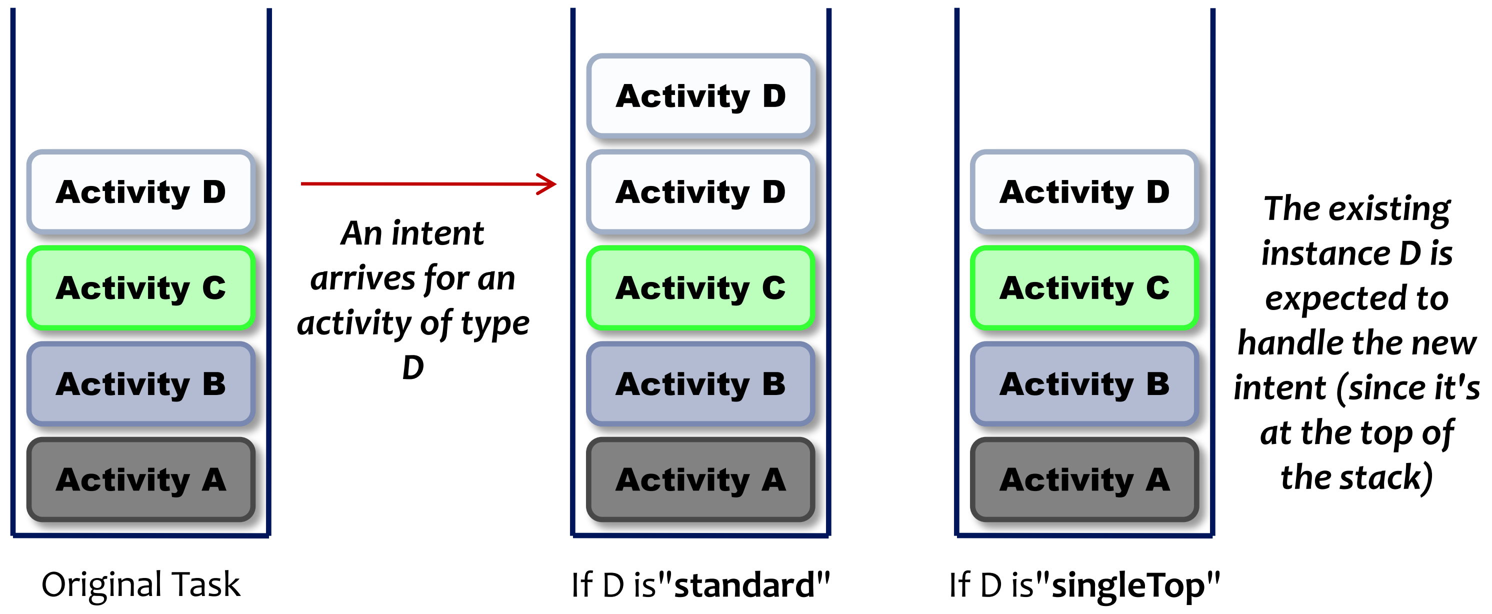
"standard"	• These modes permit multiple activities to belong to the task. • A "singleTask" activity will always be the root activity of the task.
"singleTop"	
"singleTask"	
"singleInstance"	• An activity stands alone as the only activity in its task.

Activities and Tasks (Cont)

Launch Modes (Cont)

- ✓ The modes differ from each other on four points: (Cont)

Whether a new instance of the class will be launched to handle a new intent

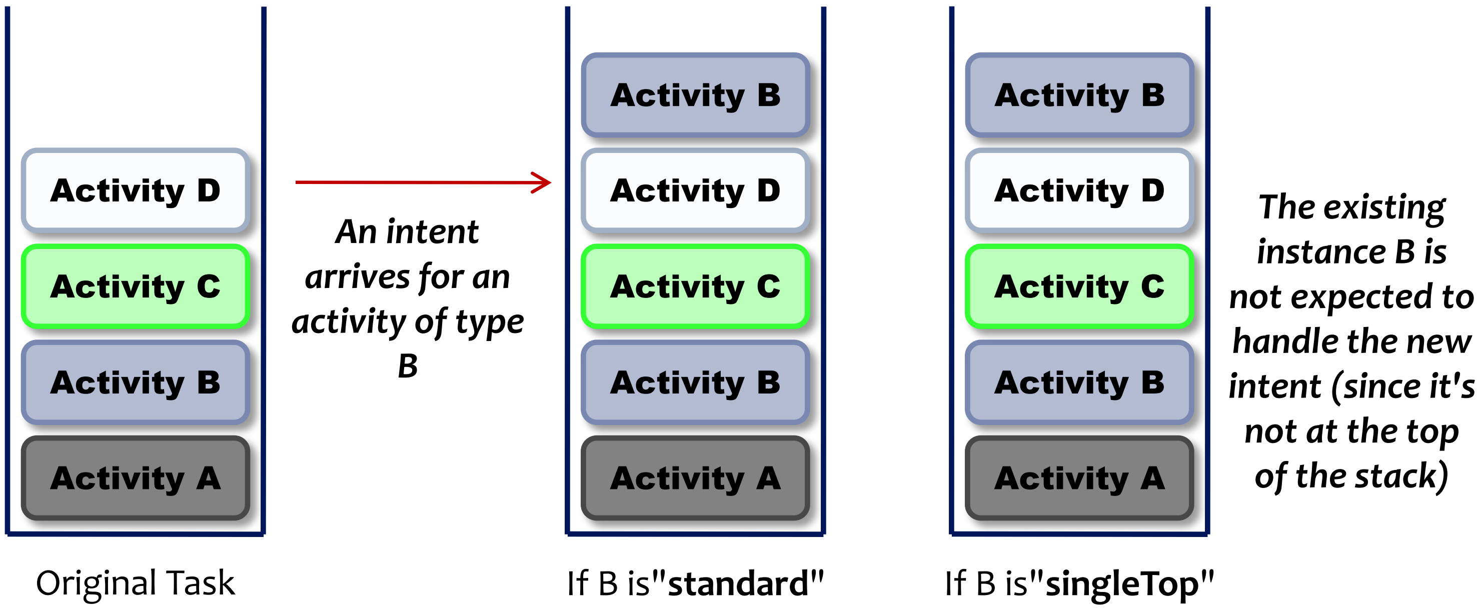


Activities and Tasks (Cont)

Launch Modes (Cont)

✓ The modes differ from each other on four points: (Cont)

Whether a new instance of the class will be launched to handle a new intent

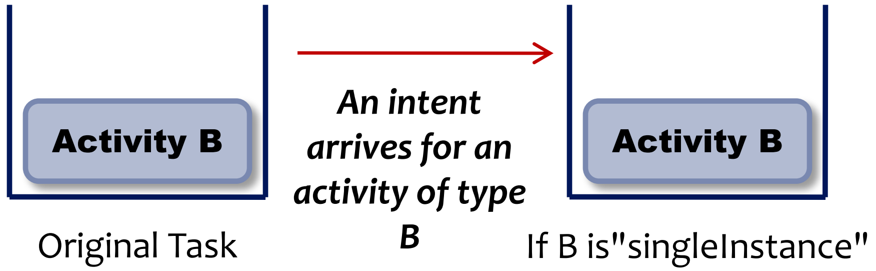


Activities and Tasks (Cont)

Launch Modes (Cont)

- ✓ The modes differ from each other on four points: (Cont)

  Whether a new instance of the class will be launched to handle a new intent



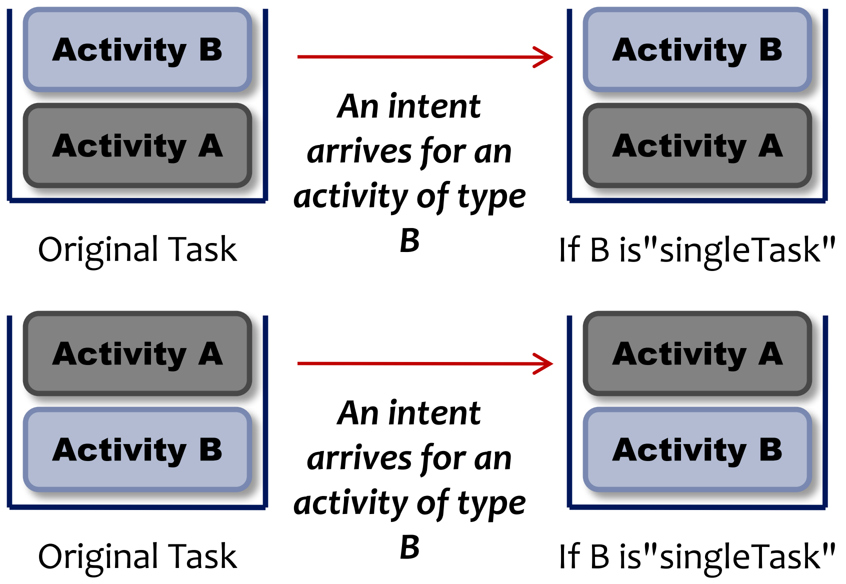
A "singleInstance" activity is always at the top of the stack, so it is always in position to handle the intent.

Activities and Tasks (Cont)

Launch Modes (Cont)

✓ The modes differ from each other on four points: (Cont)

Whether a new instance of the class will be launched to handle a new intent



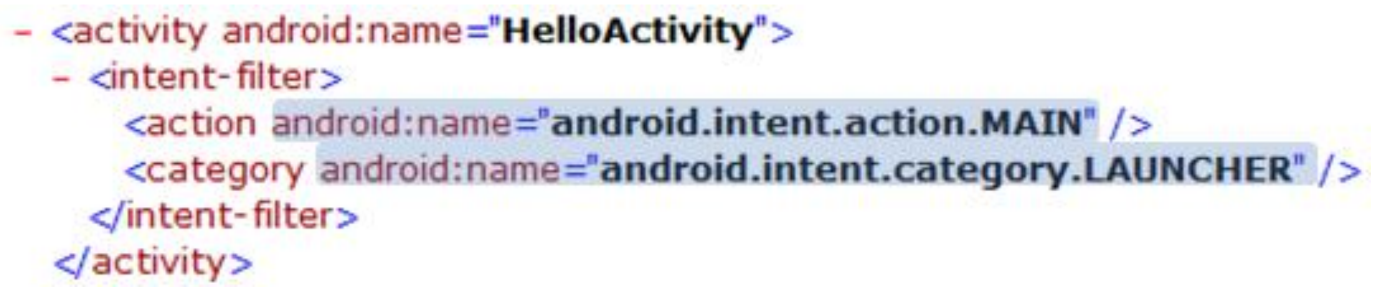
Activity B can handle the intent since it is in position.

Activity B cannot handle the intent since it is not in position and the intent is dropped.

Activities and Tasks (Cont)

Starting Tasks

- ✓ How to set up an activity as the entry point of a task

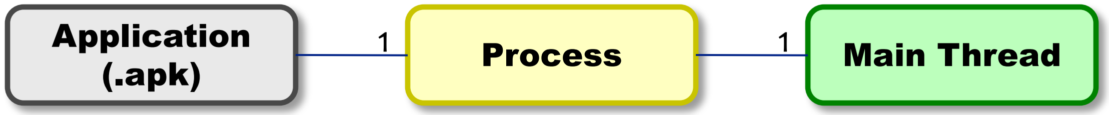


```
- <activity android:name="HelloActivity">
  - <intent-filter>
      <action android:name="android.intent.action.MAIN" />
      <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
  </activity>
```

Processes and Threads

Processes

✓ When the first of an application's components needs to be run, Android starts a Linux process for it with a single thread of execution (**Main Thread**).



✓ Android may decide to kill a process to reclaim resources.

Processes and Threads (Cont)

Process (Cont)

- ✓ We can specify a process where an individual component should run by setting a process name to “process” attribute of <activity>, <service>, <receiver>, or <provider>.
  - Each component can run in its own process.
  - Some components share a process while others do not.
  - Components of different applications also can run in the same process.
- ✓ We can set a default value that applies to all components by setting a default process to "process attribute of <application>.

Processes and Threads (Cont)

Threads

- ✓ Main Thread

  All components are instantiated in the main thread of the specified process.

  System calls to the components are dispatched from the main thread.

  - Methods that respond to those calls always run in the main thread of the process.

  - **No component should perform long or blocking operations** (e.g. network downloads, computation loops)

Processes and Threads (Cont)

Threads (Cont)

- ✓ Anything that may not be completed quickly should be assigned to a different thread.

  Threads are created in code using standard Java Thread objects.

- ✓ Some convenience classes Android provides for managing threads:

  Looper for running a message loop within a thread

  Handler for processing messages

  HandlerThread for providing a handy way for starting a new thread that has a looper

Component Lifecycles

Activity Lifecycle

- ✓ Three states

State	Description
Running	• An activity is in the foreground of the screen (at the top of the activity stack for the current task).
Paused	• An activity has lost focus but is still visible to the user.
Stopped	• An activity is completely obscured by another activity. • It still retains all state and member information.

- ✓ If an activity is **paused** or **stopped**, the system can drop it from memory either by:

  asking it to finish (calling its *finish()* method)

  simply killing its process.

Component Lifecycles

Activity Lifecycle (Cont)

✓An activity's overall lifecycle

onCreate()

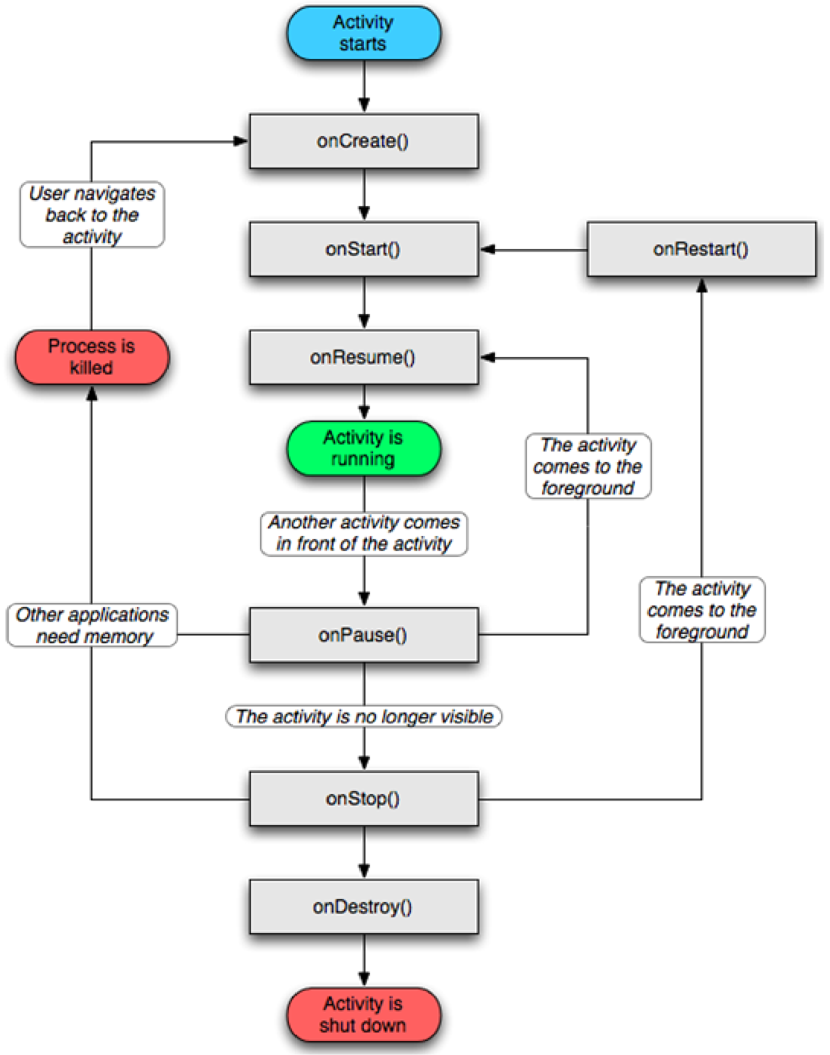
Called when the activity is first created or when the activity was killed

onStart()

Called just before the activity becomes visible to user

onRestart()

Called after the activity has been stopped, just prior to it being started again


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

Component Lifecycles (Cont)

Activity Lifecycle (Cont)

- ✓An activity's overall lifecycle (Cont)
  - onResume()
    - Called just before the activity starts interacting with the user
    - At this point, the activity is at the top of the activity stack, with user input going to it.
  - onPause()
    - Called when the system is about to start resuming another activity
    - This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on.

Component Lifecycles (Cont)

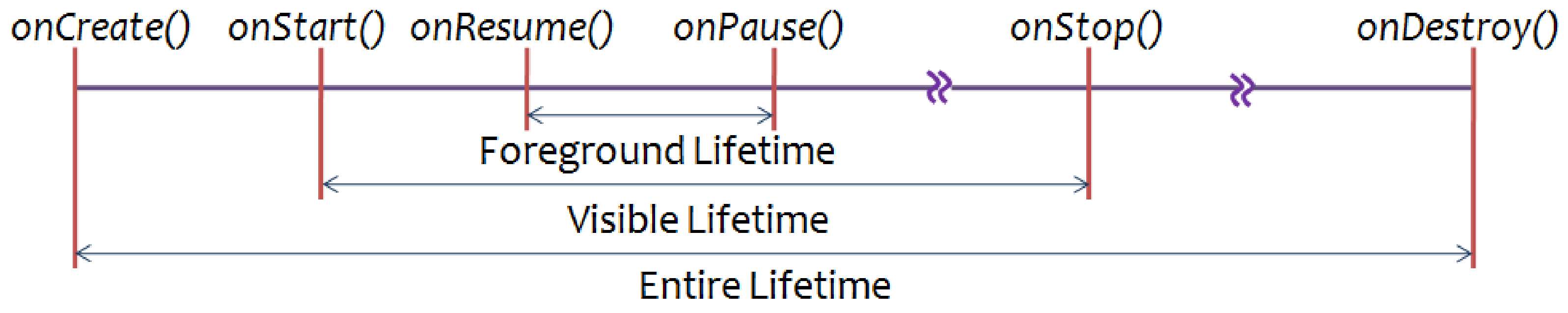
Activity Lifecycle (Cont)

- ✓An activity's overall lifecycle (Cont)
  - onStop()
    - Called when the activity is no longer visible to the user
    - This may happen because it is being destroyed, or because another activity has been resumed and is covering it.
  - onDestroy()
    - Called before the activity is destroyed

Component Lifecycles (Cont)

Activity Lifecycle (Cont)

✓Three nested loops for the entire lifecycle



Visible Lifetime

During this time, the user can see the activity on-screen, though it may be in the foreground and interacting with the user.

onStart() and onStop() can be called multiple times, as the activity alternates between being visible and hidden to the user.

Foreground Lifetime

During this time, the activity is in front of all other activities on screen and is interacting with the user.

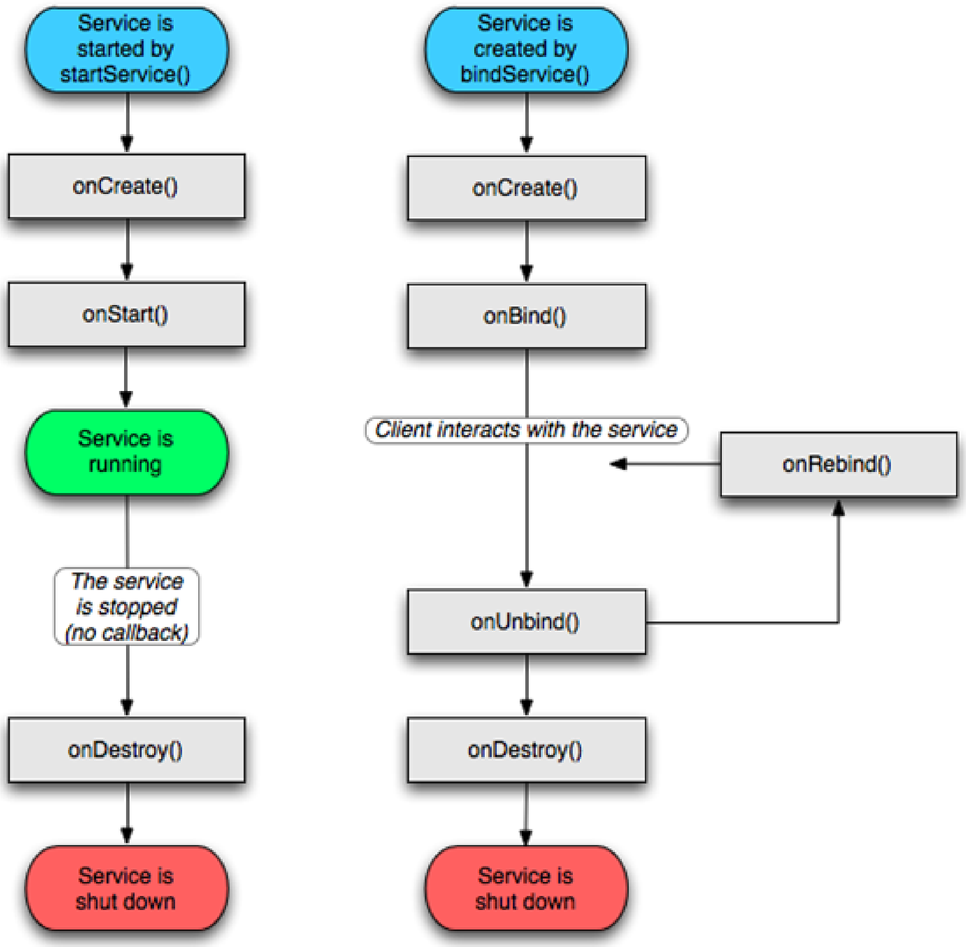
Component Lifecycles (Cont)

Service Lifecycle

- ✓ Two ways that a service can be used
  1. The service can be started and allowed to run until someone stops it or it stops itself.
     - **started by calling Context.startService() and stopped by calling Context.stopService()**
  2. The service can be operated programmatically using an interface that it defines and exports.
     - **Clients establish a connection to the Service object and use that connection to call into the service.**
     - **established by calling Context.bindService() and closed by calling Context.unbindService()**

Component Lifecycles (Cont)

Service Lifecycle (Cont)


Service is started by startService()
onCreate()
onStart()
Service is running
The service is stopped (no callback)
onDestroy()
Service is shut down
Service is created by bindService()
onCreate()
onBind()
Client interacts with the service
onRebind()
onUnbind()
onDestroy()
Service is shut down

Component Lifecycles (Cont)

Broadcast Receiver Lifecycle

- ✓Only single callback method

void onReceive(Context curContext, Intent broadcastMsg)

When a broadcast message arrives for the receiver, Android calls the method and passes it the Intent object containing the message.

- ✓A process with an active broadcast receiver is protected from being killed but a process with only inactive components can be killed by the system at any time.

Component Lifecycles (Cont)

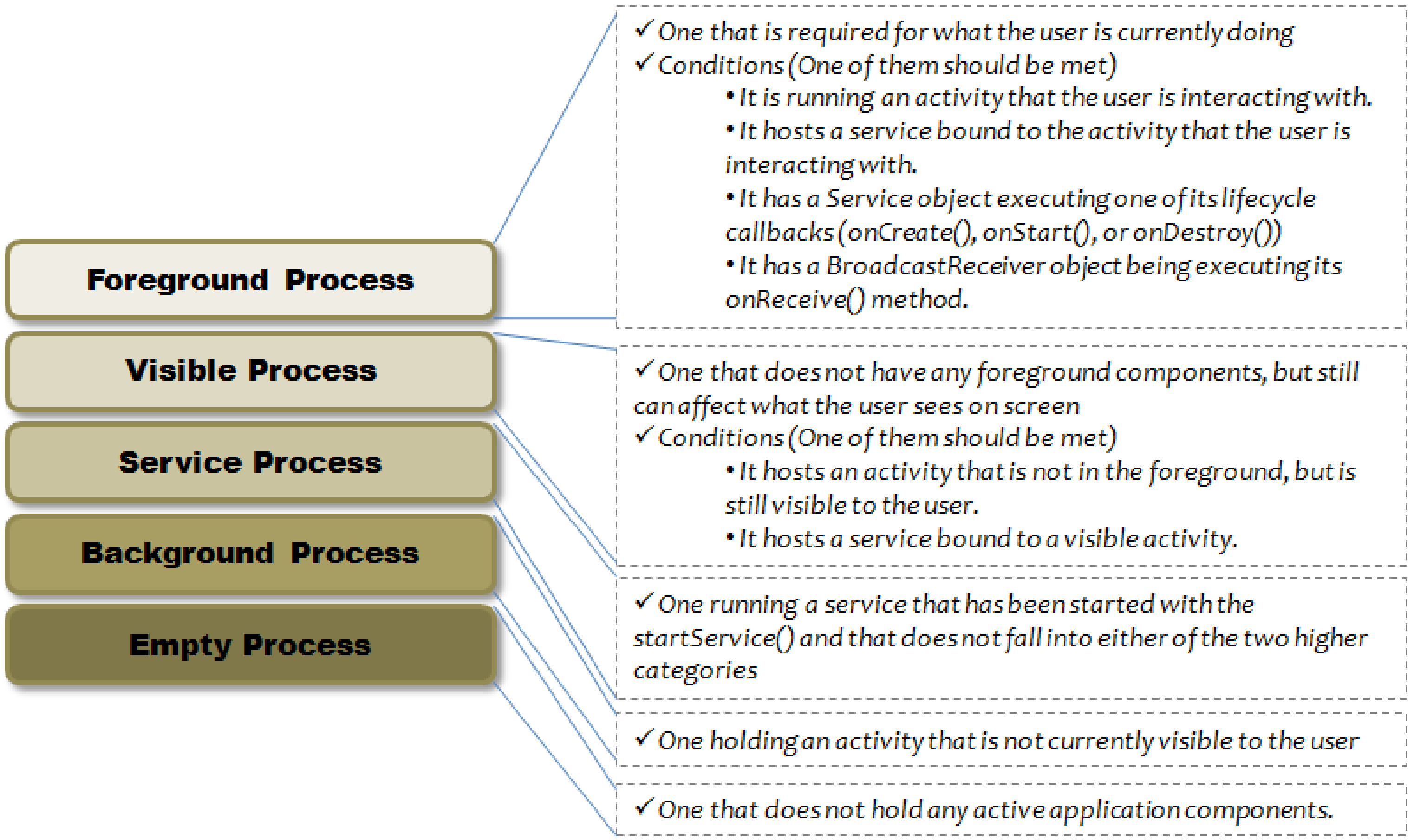
Processes and Lifecycles

- ✓ Android tries to maintain a process for as long as possible, but eventually it will need to remove old processes when memory runs low.
  - To determine candidates to be killed, Android places each process into an "importance hierarchy" based on the components running in it and the state of those components.
  - Sounds like a multi-level feedback queue…

Component Lifecycles (Cont)

Processes and Lifecycles (Cont)

- ✓ Five levels in the Importance Hierarchy



✓ One that is required for what the user is currently doing
✓ Conditions (One of them should be met)
- *It is running an activity that the user is interacting with.*
- *It hosts a service bound to the activity that the user is interacting with.*
- *It has a Service object executing one of its lifecycle callbacks (onCreate(), onStart(), or onDestroy())*
- *It has a BroadcastReceiver object being executing its onReceive() method.*

✓ One that does not have any foreground components, but still can affect what the user sees on screen
✓ Conditions (One of them should be met)
- *It hosts an activity that is not in the foreground, but is still visible to the user.*
- *It hosts a service bound to a visible activity.*

✓ One running a service that has been started with the startService() and that does not fall into either of the two higher categories

✓ One holding an activity that is not currently visible to the user

✓ One that does not hold any active application components.