UCD School of Computer Science

# Introduction to Mobile Application Development

*Dr. Abraham(Abey) Campbell*

# Android Operating Systems
## Overview

## Outline

| Title: Android OS | Module: COMP2004L | Number in sequence: 1 |
|---|---|---|

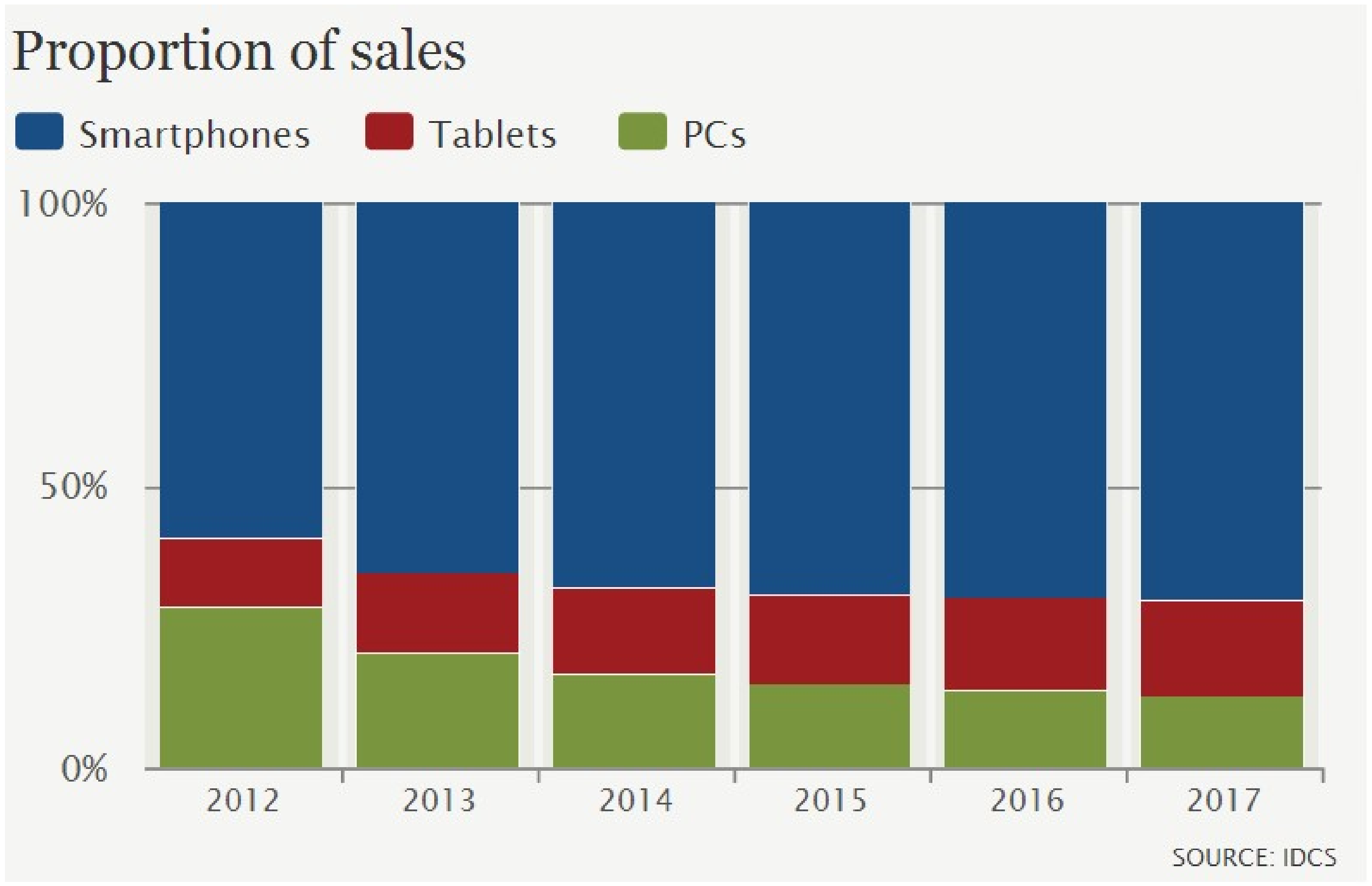
**Aim(s)**: Present basic concepts of Android operating system

**Learning Objectives:**

At the end of this lecture, you will be able to:

- Understand smartphone application
- Identify Android versions and API
- Explain Android stack
- Understand Android application distribution
- Describe Main Building Blocks

## Smartphones, in general



## Smartphones, in general

Smartphone and tablet sales overtaking PCs and Laptops

Roughly 250 000 active patents impact smartphones, or 1/6 of patents overall (not just technology sector).

# Smartphones, vision

The vision continues to accelerate:

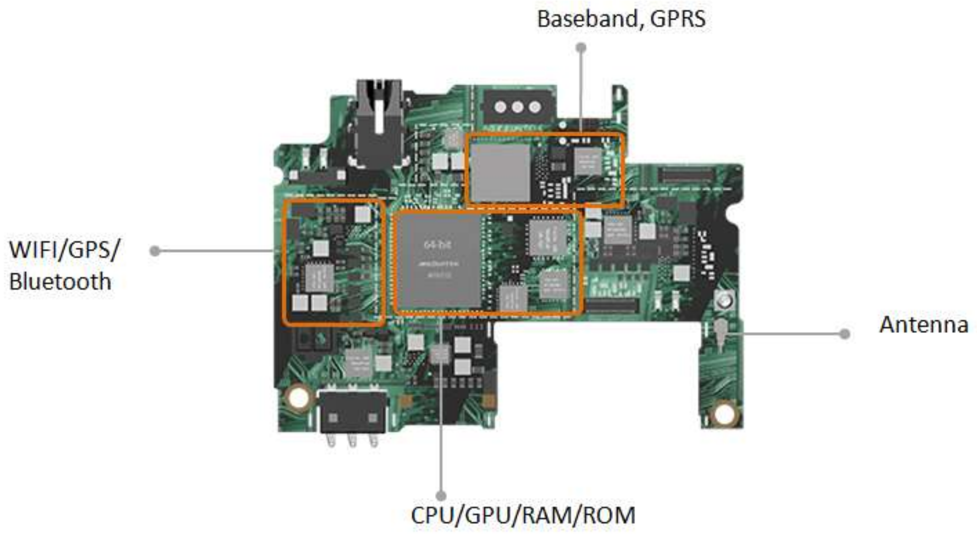
- OLED displays
- Google Glasses
- Smaller, longer-lasting batteries
- … and more to come (any ideas?)

# Smartphones Application Dev.

- Never before so much functionality has been made available to developers in one device.
- Smartphones are extremely personal devices.
- Smartphones are replacing our wallets.
- Smartphones OS are increasingly becoming embedded in everything.

# Smartphones Application Dev.

- Smartphones have limited memory, CPU, and storage capabilities.
- Smartphones have a very limited power source (battery)
- Can be easily stolen or lost.



# Smartphones Application Dev.

- Never before so much functionality has been made available to developers in one device.
- Smartphones are extremely personal devices.
- Smartphones have limited memory, CPU, and storage capabilities.
- Smartphones have a very limited power source (battery)
- Can be easily stolen or lost.
- Smartphones are replacing our wallets.
- Smartphones OS are increasingly becoming embedded in everything.

*Be a developer of a vision and design your apps with all of that in mind!*

# Android - History

- 2003 Android Inc. California.
- 2005 Google bought it.
- 2007, January, iPhone!!
- 2007, November, Google
  gave it to OHA
- Managed by **Open Handset Alliance**
  - Open source
  - Lots of members / votes: Techs & Telecoms
  - Goal: Innovative, richer, cheap and better mobile experience.
  - SW & HW are two different things, finally.

# Android - Facts

- Mainly pushed by: **Google**.
- **Comprehensive**: complete SW stack "no need for anything else to get it running"
- **Better deal** for developers (SDK), users (Team = Services) and manufacturers (Diff. Concerns).

*"Today, there are <u>1.5 billion television sets</u> in use around the world. <u>1 billion people are on the Internet</u>. But nearly <u>3 billion people have a mobile</u> phone, making it one of the world's most successful consumer products."*

open handset alliance

# Android - License

- **Business-friendly licenses (Apache/MIT):**
  - You can freely extend it.
  - You can use it for variety of purposes.
  - Rewrite and include expensive libraries.
  - You have access to the entire source code:
  - Understand the core functionality.
  - Add secret sauces without sharing.

**In brief, there is no need to license Android, you can start using and modifying it today.**

## Android – Assumptions

- Purpose-built platform for mobile devices.
- Android devices are going to always be limited in terms of memory and speed.
- Android is designed to run on all sorts of physical devices.
- Nothing is predefined: screen size, resolution, chipset, … etc.
- Its core is designed to be portable.

## Android – Assumptions

*Google is a media company, and its business model is based on selling advertising.*
*If everyone is using Android, then Google can provide additional services on top*
*of it and compete fairly.*

*Although Google does license some proprietary apps, such as Gmail and Maps, and*
*makes some money off the Android market, its primary motivation is still the advertising*
*revenue that those apps bring in.*

# Android – Assumptions

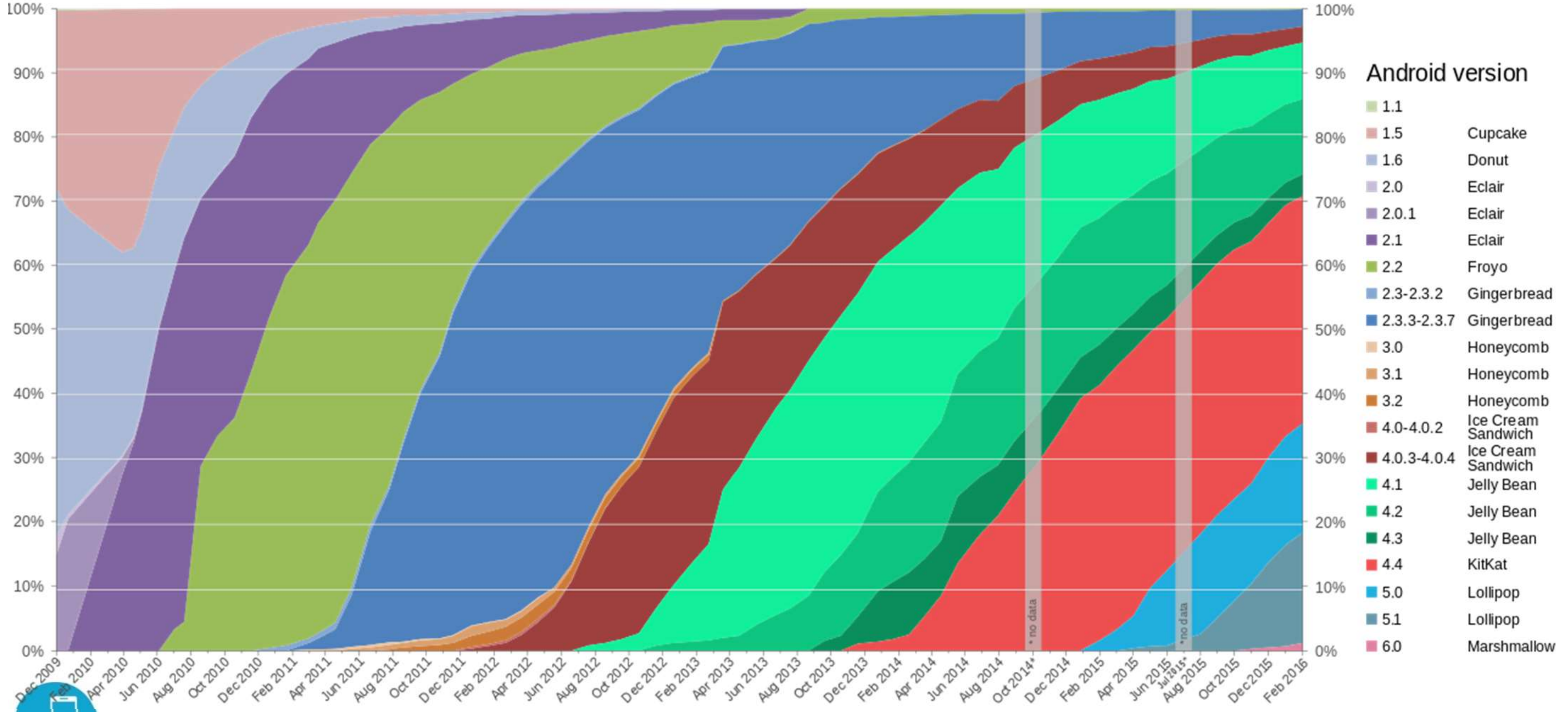
- Purpose-built platform for mobile devices.
- Android devices are going to always be limited in terms of memory and speed.
- Android is designed to run on all sorts of physical devices.
- Nothing is predefined: screen size, resolution, chipset, … etc.
- Its core is designed to be portable.

*Google is a media company, and its business model is based on selling advertising.*
*If everyone is using Android, then Google can provide additional services on top of it and compete fairly.*

*Although Google does license some proprietary apps, such as Gmail and Maps, and makes some money off the Android market, its primary motivation is still the advertising revenue that those apps bring in.*

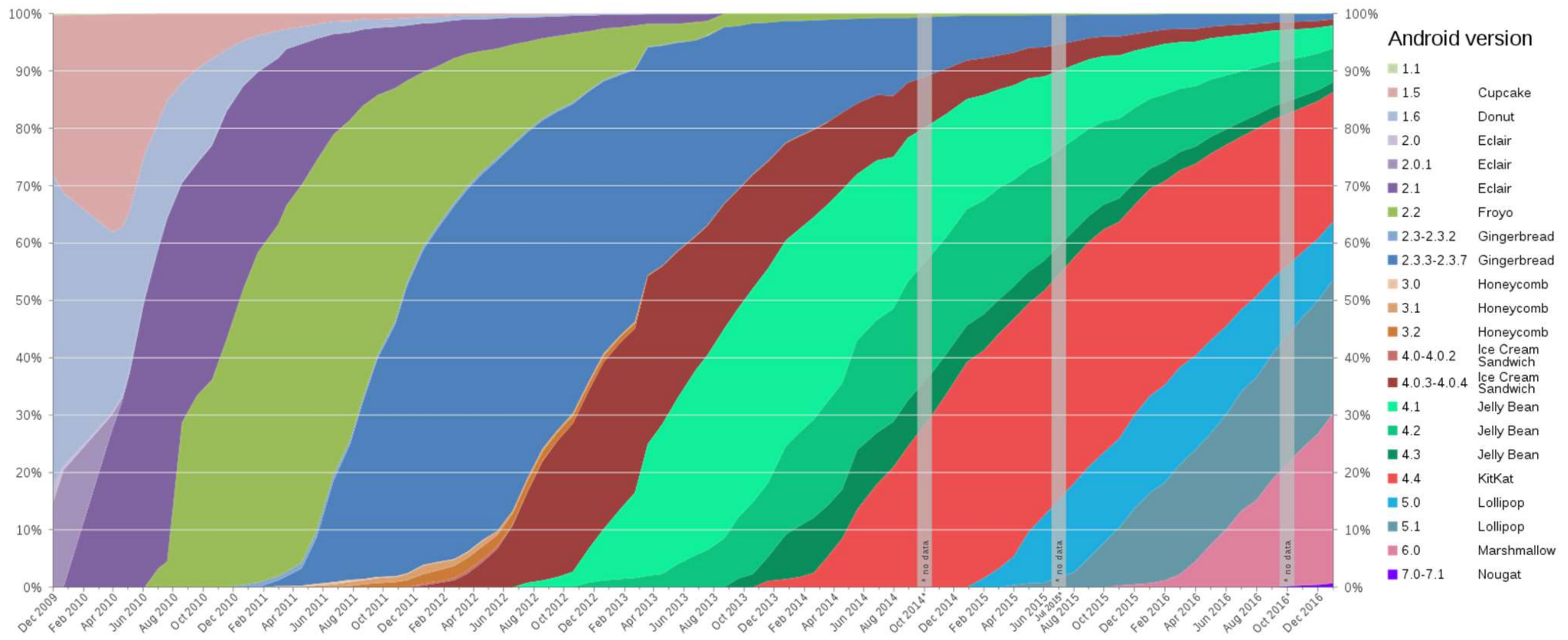
# Android Versions

Global Android version distribution since December 2009. As of February 2016, Android 4.4 "KitKat" is the single most widely used Android version, operating on 35.5% of all Android devices accessing Google Play. The second are different Android "Lollipop" versions (5.0–5.1.1), with a combined share of 34.1%.[1].



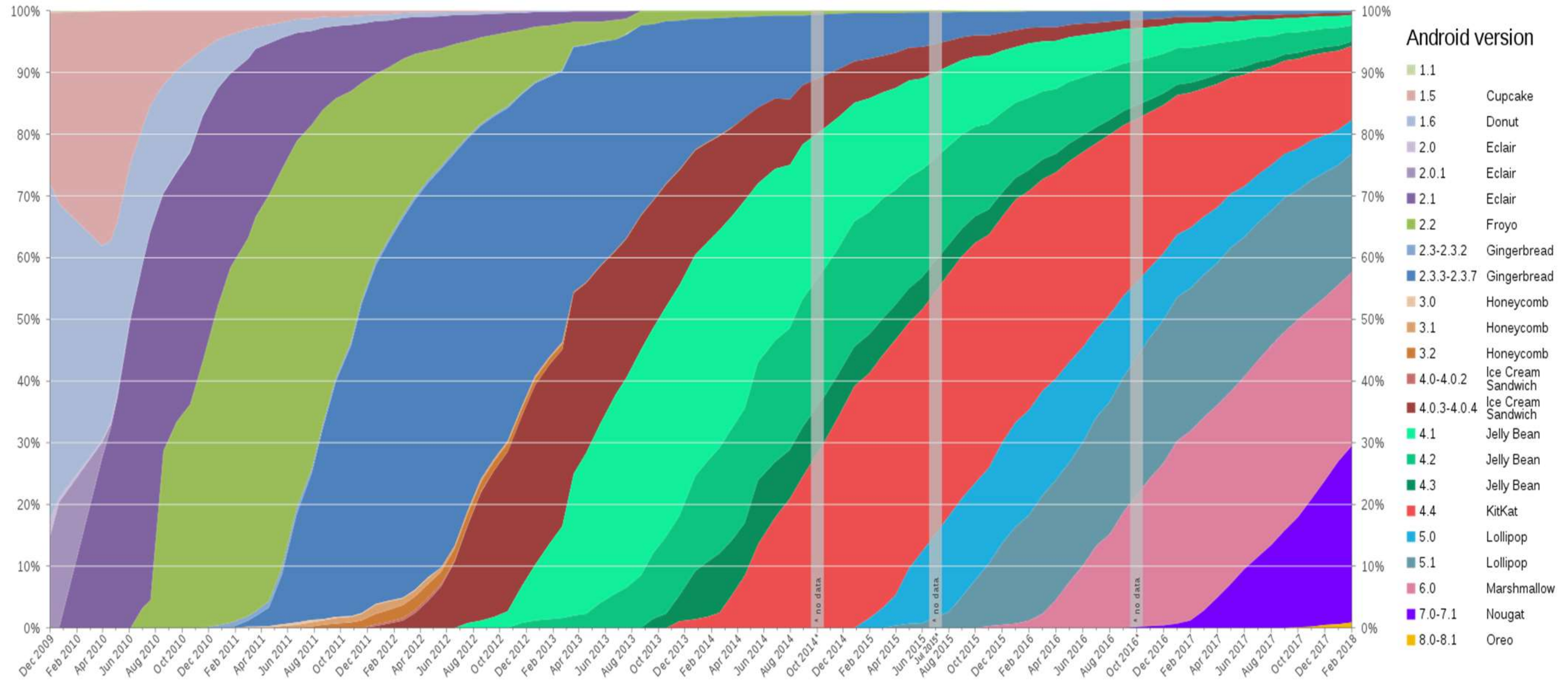
# Android Versions

Global Android version distribution since December 2009. As of February 2017, Android 5.x “Lollipop" is the single most widely used Android version, operating on 33% of all Android devices accessing Google Play. The second are Android “Marshmallow“ with 30%.[1].

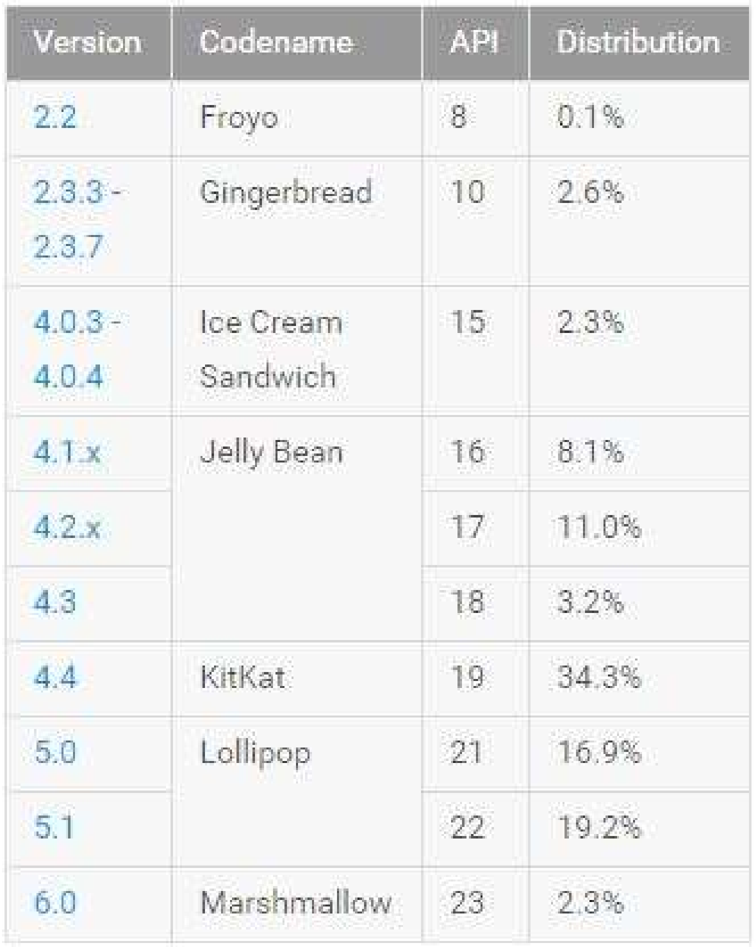


# Android Versions

Global Android version distribution since December 2009. As of February 2018, Android Nougat is the most widely used version of Android, running on 28.5% of all Android devices accessing Google Play, while Android Lollipop 5.1.x, the oldest supported Android version runs on 19.2% of devices. *(wikipedia)*



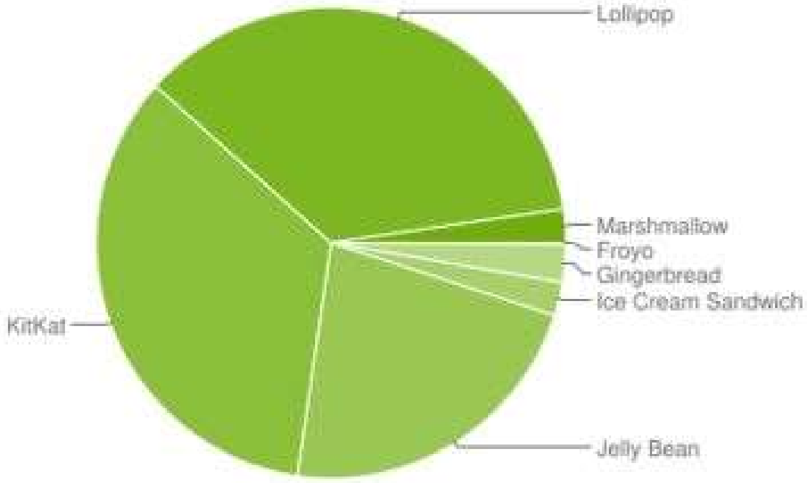
# Android Versions



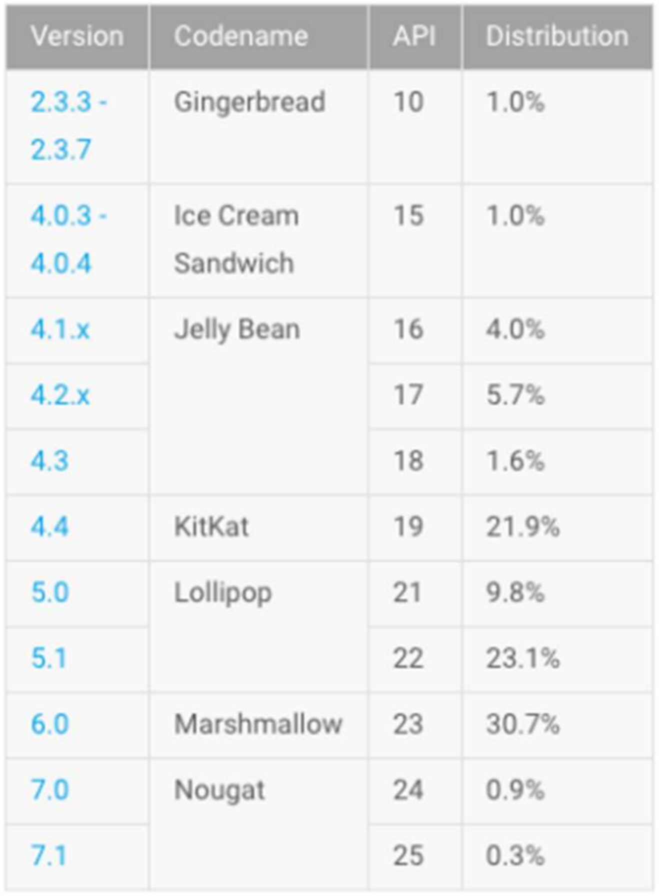
| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.2 | Froyo | 8 | 0.1% |
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 2.6% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 2.3% |
| 4.1.x | Jelly Bean | 16 | 8.1% |
| 4.2.x | | 17 | 11.0% |
| 4.3 | | 18 | 3.2% |
| 4.4 | KitKat | 19 | 34.3% |
| 5.0 | Lollipop | 21 | 16.9% |
| 5.1 | | 22 | 19.2% |
| 6.0 | Marshmallow | 23 | 2.3% |

*Data collected during a 7-day period ending on March 7, 2016.*

*Any versions with less than 0.1% distribution are not shown.*



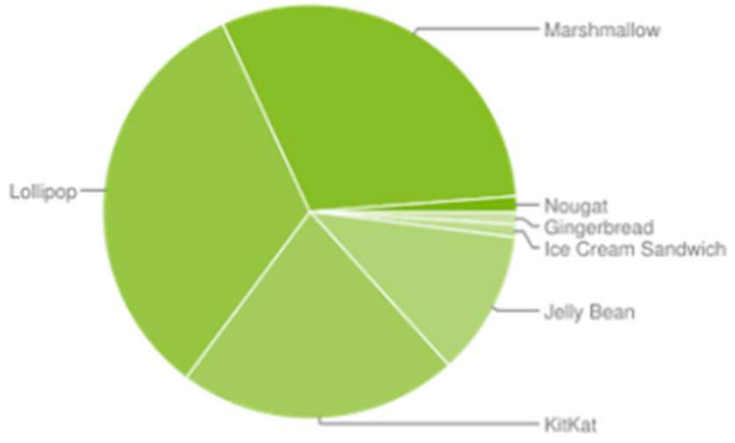
# Android Versions



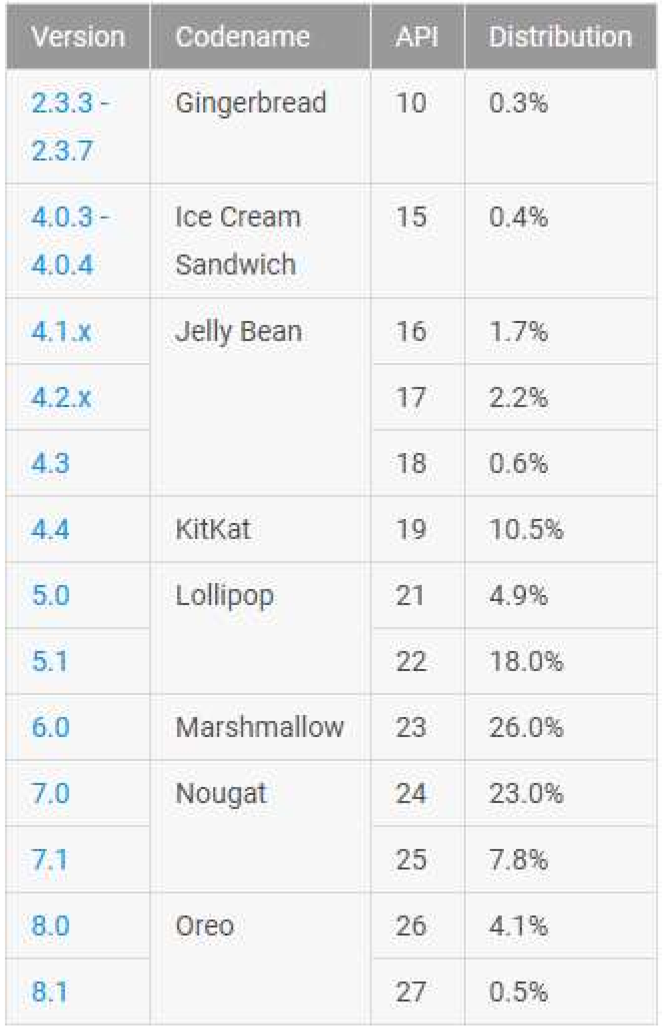
| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 1.0% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 1.0% |
| 4.1.x | Jelly Bean | 16 | 4.0% |
| 4.2.x | | 17 | 5.7% |
| 4.3 | | 18 | 1.6% |
| 4.4 | KitKat | 19 | 21.9% |
| 5.0 | Lollipop | 21 | 9.8% |
| 5.1 | | 22 | 23.1% |
| 6.0 | Marshmallow | 23 | 30.7% |
| 7.0 | Nougat | 24 | 0.9% |
| 7.1 | | 25 | 0.3% |

*Data collected during a 7-day period ending on February 6, 2017.*

*Any versions with less than 0.1% distribution are not shown.*

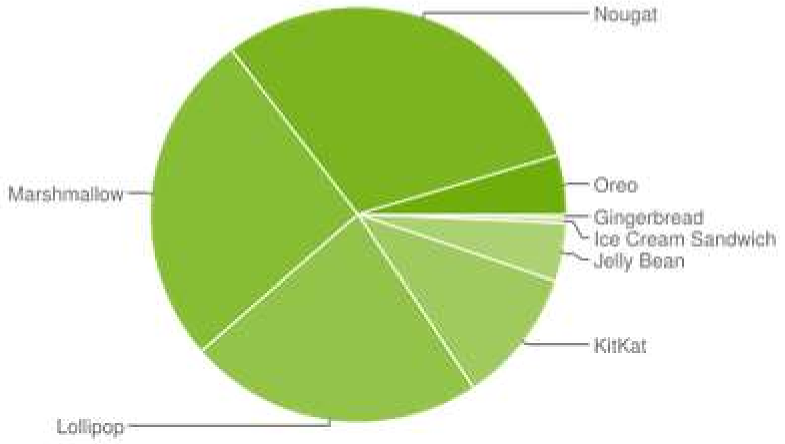


# Android Versions



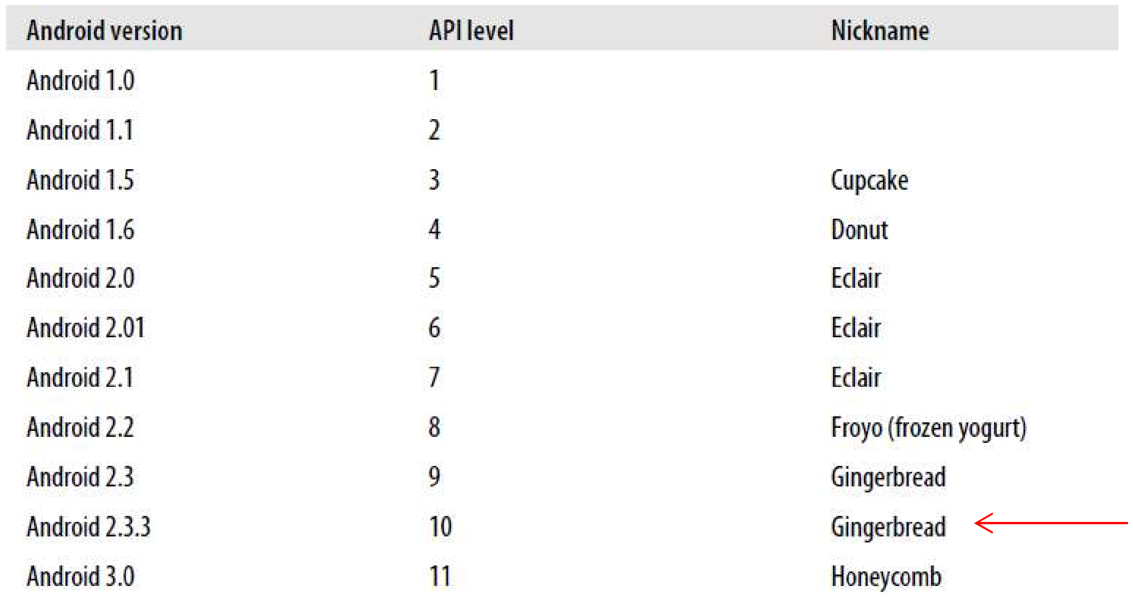
| Version | Codename | API | Distribution |
|---|---|---|---|
| 2.3.3 - 2.3.7 | Gingerbread | 10 | 0.3% |
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 0.4% |
| 4.1.x | Jelly Bean | 16 | 1.7% |
| 4.2.x | | 17 | 2.2% |
| 4.3 | | 18 | 0.6% |
| 4.4 | KitKat | 19 | 10.5% |
| 5.0 | Lollipop | 21 | 4.9% |
| 5.1 | | 22 | 18.0% |
| 6.0 | Marshmallow | 23 | 26.0% |
| 7.0 | Nougat | 24 | 23.0% |
| 7.1 | | 25 | 7.8% |
| 8.0 | Oreo | 26 | 4.1% |
| 8.1 | | 27 | 0.5% |

*Data collected during a 7-day period ending on April 16, 2018.*
*Any versions with less than 0.1% distribution are not shown.*



# Android APIs

If each SDK deals with specific API that targets certain Version, which SDK to use?



| Android version | API level | Nickname |
|---|---|---|
| Android 1.0 | 1 | |
| Android 1.1 | 2 | |
| Android 1.5 | 3 | Cupcake |
| Android 1.6 | 4 | Donut |
| Android 2.0 | 5 | Eclair |
| Android 2.01 | 6 | Eclair |
| Android 2.1 | 7 | Eclair |
| Android 2.2 | 8 | Froyo (frozen yogurt) |
| Android 2.3 | 9 | Gingerbread |
| Android 2.3.3 | 10 | Gingerbread |
| Android 3.0 | 11 | Honeycomb |

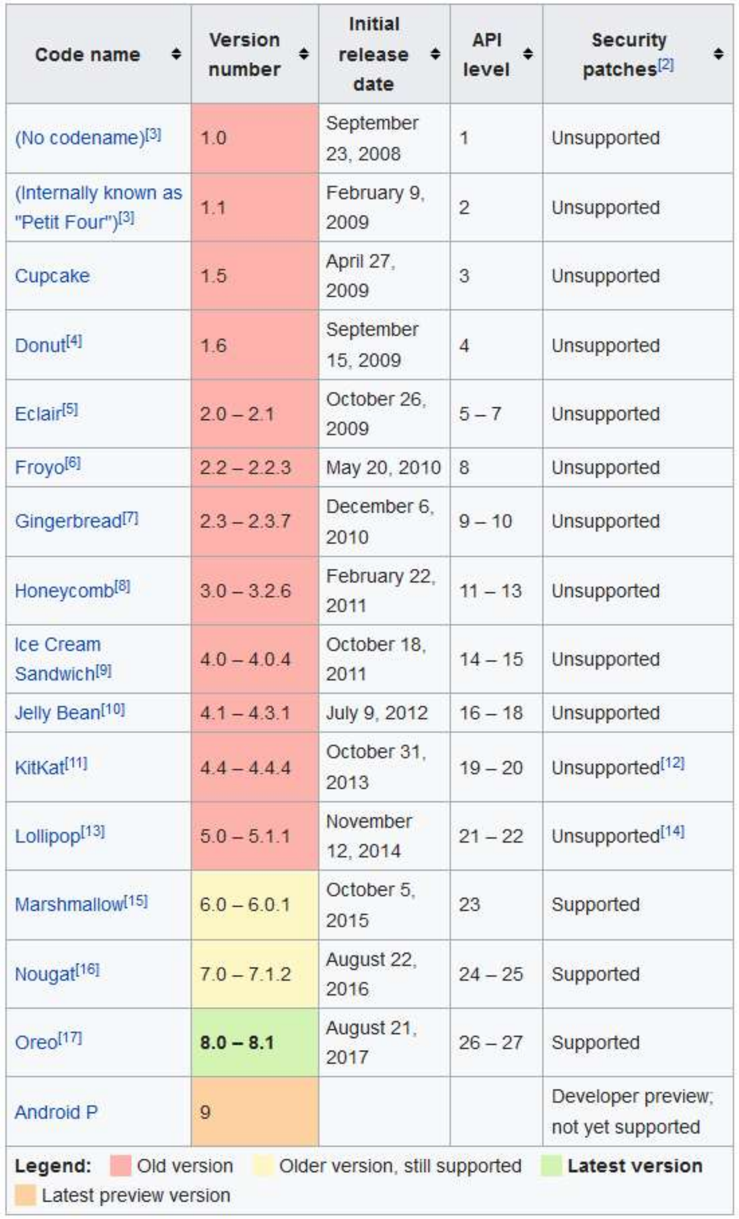
Android 2.3.3, you should set your targetSdkVersion to "10"

*Typically your objective is to have your application run on as many devices as possible.*

*So, with that in mind, try to shoot for an API level that is as low as possible. Keep in mind the distribution of Android versions on real devices out there (earlier slides).*

Please, avoid the usual mistake of directly rushing to the latest SDK version.

# Android APIs



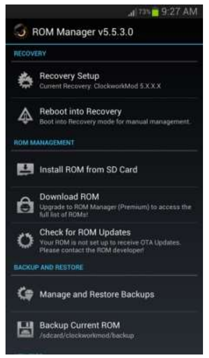
| Code name | Version number | Initial release date | API level | Security patches[2] |
|---|---|---|---|---|
| (No codename)[3] | 1.0 | September 23, 2008 | 1 | Unsupported |
| (Internally known as "Petit Four")[3] | 1.1 | February 9, 2009 | 2 | Unsupported |
| Cupcake | 1.5 | April 27, 2009 | 3 | Unsupported |
| Donut[4] | 1.6 | September 15, 2009 | 4 | Unsupported |
| Eclair[5] | 2.0 – 2.1 | October 26, 2009 | 5 – 7 | Unsupported |
| Froyo[6] | 2.2 – 2.2.3 | May 20, 2010 | 8 | Unsupported |
| Gingerbread[7] | 2.3 – 2.3.7 | December 6, 2010 | 9 – 10 | Unsupported |
| Honeycomb[8] | 3.0 – 3.2.6 | February 22, 2011 | 11 – 13 | Unsupported |
| Ice Cream Sandwich[9] | 4.0 – 4.0.4 | October 18, 2011 | 14 – 15 | Unsupported |
| Jelly Bean[10] | 4.1 – 4.3.1 | July 9, 2012 | 16 – 18 | Unsupported |
| KitKat[11] | 4.4 – 4.4.4 | October 31, 2013 | 19 – 20 | Unsupported[12] |
| Lollipop[13] | 5.0 – 5.1.1 | November 12, 2014 | 21 – 22 | Unsupported[14] |
| Marshmallow[15] | 6.0 – 6.0.1 | October 5, 2015 | 23 | Supported |
| Nougat[16] | 7.0 – 7.1.2 | August 22, 2016 | 24 – 25 | Supported |
| Oreo[17] | **8.0 – 8.1** | August 21, 2017 | 26 – 27 | Supported |
| Android P | 9 | | | Developer preview; not yet supported |

**Legend:** 🟥 Old version 🟨 Older version, still supported 🟩 **Latest version** 🟧 Latest preview version

*s://en.wikipedia.org/wiki/Android_version_history)*

# Android Hardware Platforms

- Core components
- Device types
- ROM and Boot Loaders
- Manufactures



9:27 AM

ROM Manager v5.5.3.0

RECOVERY

Recovery Setup
Current Recovery: ClockworkMod 5.X.X.X

Reboot into Recovery
Boot into Recovery mode for manual management.

ROM MANAGEMENT

Install ROM from SD Card

Download ROM
Upgrade to ROM Manager (Premium) to access the full list of ROMs!

Check for ROM Updates
Your ROM is not set up to receive OTA Updates. Please contact the ROM developer!

BACKUP AND RESTORE

Manage and Restore Backups

Backup Current ROM
/sdcard/clockworkmod/backup

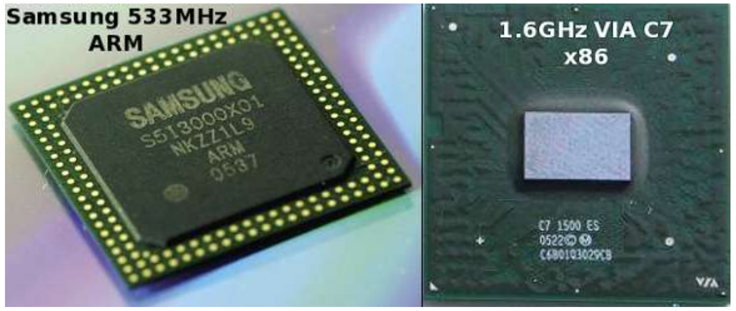
htc
HUAWEI
SAMSUNG
DELL
acer
Haier
MOTOROLA
lenovo

# Core components

- Core components
  - CPU, Modem, RAM, NAND flash, GPS (Global Positioning System), Wireless, SD card
  - Screen, Keyboard, Camera, Battery, USB, …

# CPU

- Responsible for executing operating system (OS) and application code and coordinating or controlling other core components including the network, storage, displays, and input devices
- Android devices utilize ARM processors that are powerful enough for the mobile
- Intel has ported Android to their Atom processors

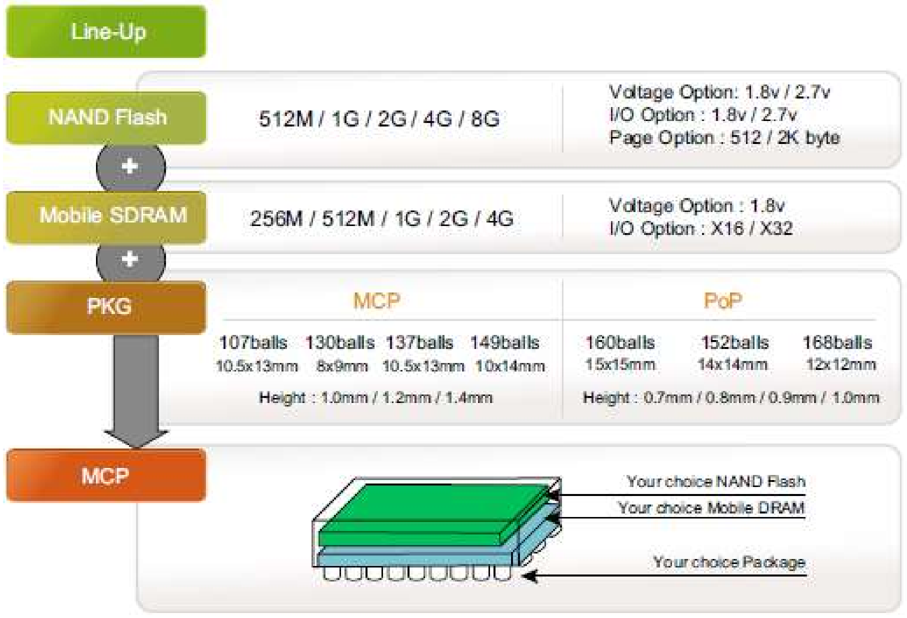


## Modem/Radio

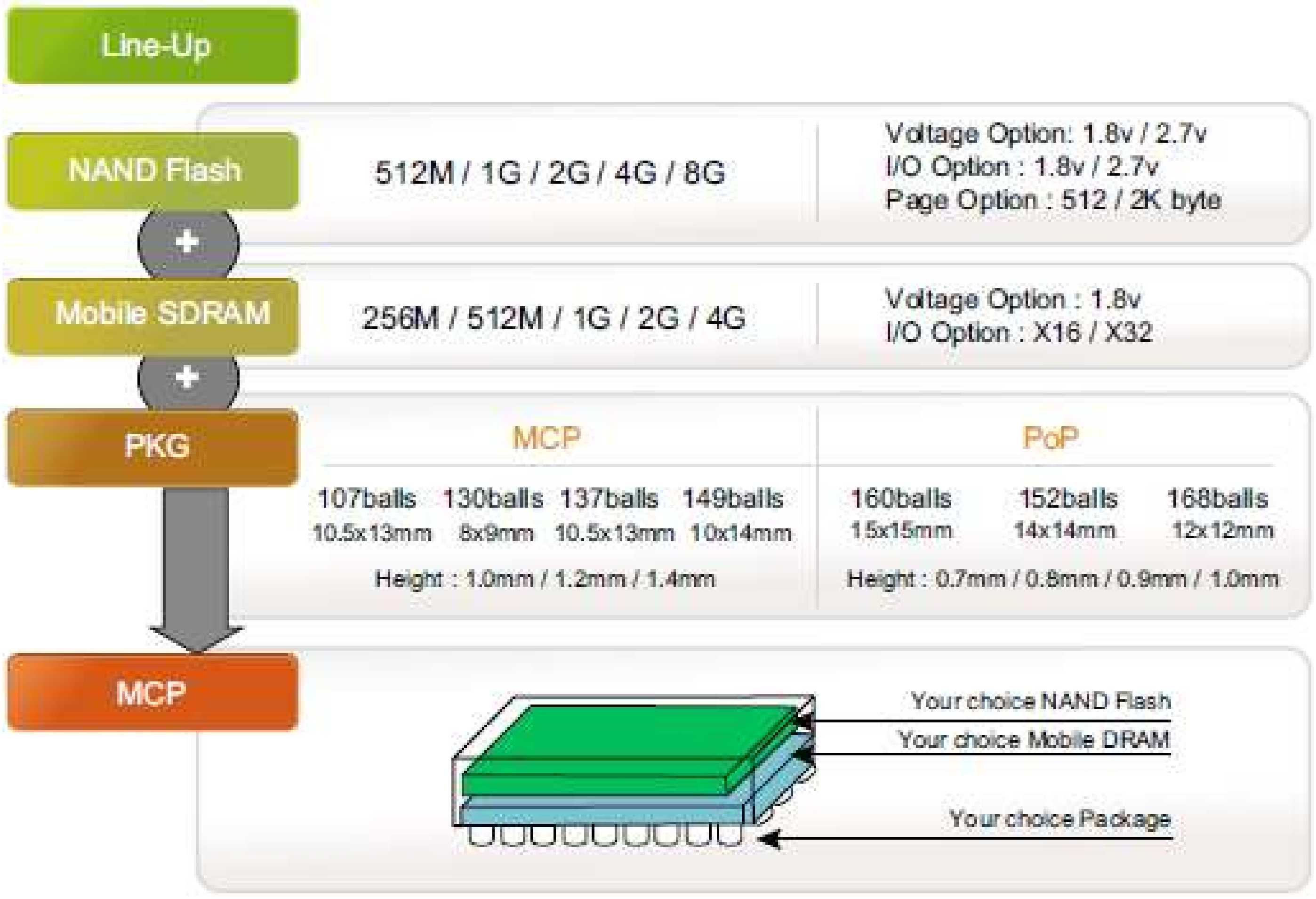
- Hardware and software systems that provide Android devices a connection to the cellular network.
- Allow both voice and data communication from the device.

# RAM & NAND Flash

- RAM is used by the system to load, execute, and manipulate key parts of the OS, applications, or data. RAM is volatile
- NAND flash memory is non-volatile - the data are preserved after the device has been powered off.
  - The NAND flash is used to store the boot loader, OS, and user data



# RAM & NAND Flash



Line-Up

| | | |
|---|---|---|
| NAND Flash | 512M / 1G / 2G / 4G / 8G | Voltage Option: 1.8v / 2.7v<br>I/O Option : 1.8v / 2.7v<br>Page Option : 512 / 2K byte |
| + Mobile SDRAM | 256M / 512M / 1G / 2G / 4G | Voltage Option : 1.8v<br>I/O Option : X16 / X32 |
| + PKG | MCP<br>107balls 10.5x13mm<br>130balls 8x9mm<br>137balls 10.5x13mm<br>149balls 10x14mm<br>Height : 1.0mm / 1.2mm / 1.4mm | PoP<br>160balls 15x15mm<br>152balls 14x14mm<br>168balls 12x12mm<br>Height : 0.7mm / 0.8mm / 0.9mm / 1.0mm |

MCP

## Global positioning system

- Identify the location of the device using the GPS satellite network
- Allow for applications such as point-to-point directions, position-aware applications

## Wireless (WiFi and Bluetooth)

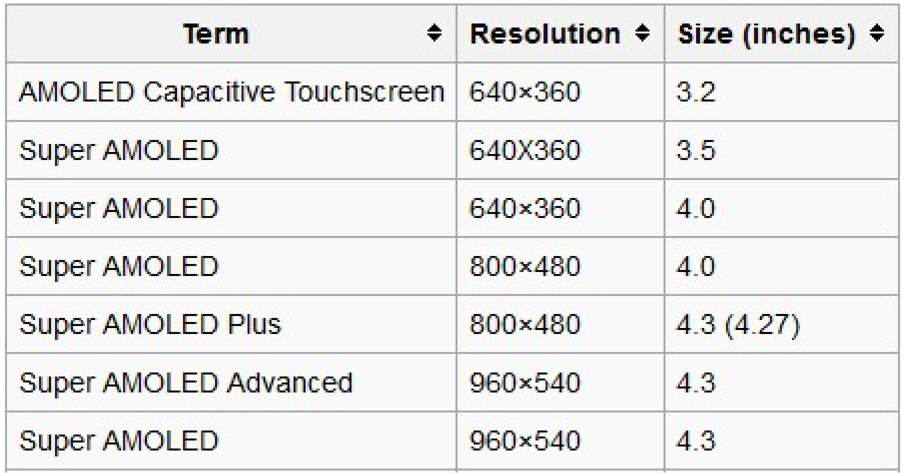
- Wifi: high-speed data connection and
- Bluetooth: Connections to external devices such as headsets, keyboards, printers, and more

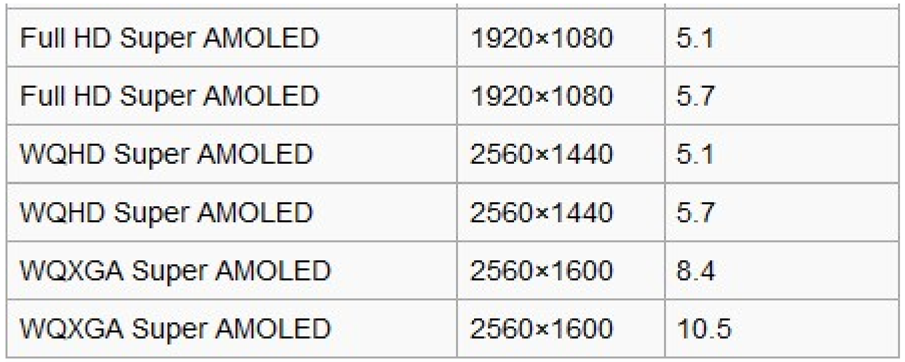
## Secure Digital Card

- Removable memory card
- Non-volatile
- Use NAND flash technology
- Portable: adhere to various physical and communication specifications that allow them to interoperate with most devices
- Larger user files are stored on the SD card

## Screen

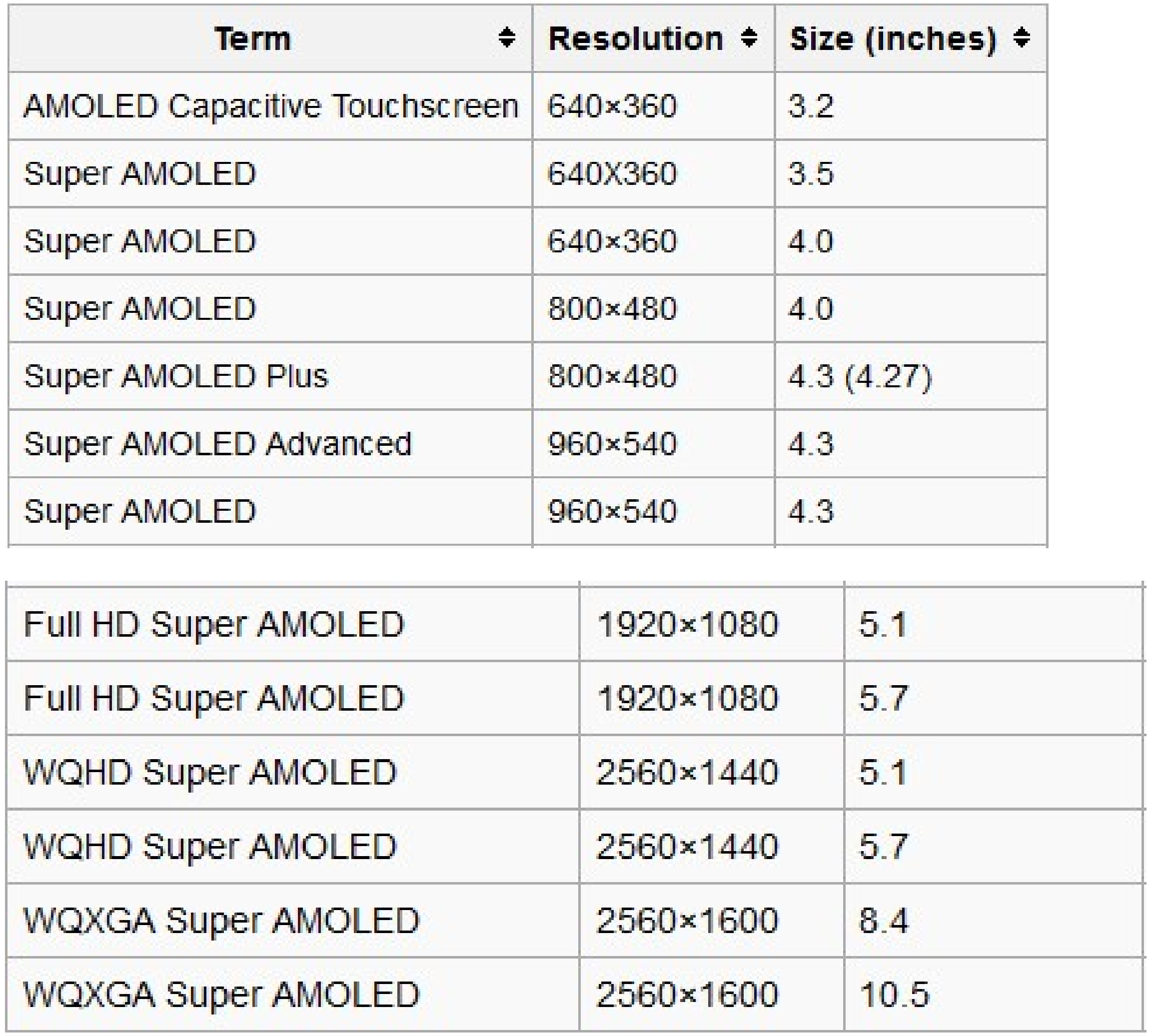
- A critical component: It is the primary interface for user interaction, not only through the visual display but also by responding to the user’s touch.
- The technologies:
  - Liquid crystal display and a second layer that detects user input on the screen.
  - Higher display resolution, brighter screens, more sensitive and complicated user touch interactions, and reduced power consumption.
    - Ex: Samsung’s Super AMOLED





| Term | Resolution | Size (inches) |
|---|---|---|
| AMOLED Capacitive Touchscreen | 640×360 | 3.2 |
| Super AMOLED | 640X360 | 3.5 |
| Super AMOLED | 640×360 | 4.0 |
| Super AMOLED | 800×480 | 4.0 |
| Super AMOLED Plus | 800×480 | 4.3 (4.27) |
| Super AMOLED Advanced | 960×540 | 4.3 |
| Super AMOLED | 960×540 | 4.3 |
| Full HD Super AMOLED | 1920×1080 | 5.1 |
| Full HD Super AMOLED | 1920×1080 | 5.7 |
| WQHD Super AMOLED | 2560×1440 | 5.1 |
| WQHD Super AMOLED | 2560×1440 | 5.7 |
| WQXGA Super AMOLED | 2560×1600 | 8.4 |
| WQXGA Super AMOLED | 2560×1600 | 10.5 |

# Screen



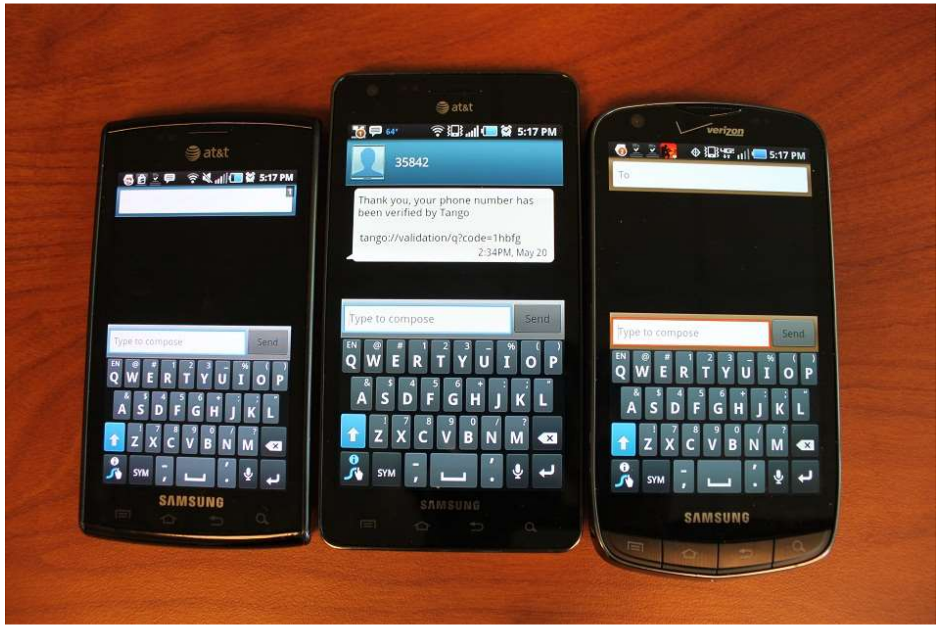
| Term | Resolution | Size (inches) |
|---|---|---|
| AMOLED Capacitive Touchscreen | 640×360 | 3.2 |
| Super AMOLED | 640X360 | 3.5 |
| Super AMOLED | 640×360 | 4.0 |
| Super AMOLED | 800×480 | 4.0 |
| Super AMOLED Plus | 800×480 | 4.3 (4.27) |
| Super AMOLED Advanced | 960×540 | 4.3 |
| Super AMOLED | 960×540 | 4.3 |
| Full HD Super AMOLED | 1920×1080 | 5.1 |
| Full HD Super AMOLED | 1920×1080 | 5.7 |
| WQHD Super AMOLED | 2560×1440 | 5.1 |
| WQHD Super AMOLED | 2560×1440 | 5.7 |
| WQXGA Super AMOLED | 2560×1600 | 8.4 |
| WQXGA Super AMOLED | 2560×1600 | 10.5 |

## Camera

- Two cameras
- Combine the camera functionality with the GPS:
  - Record GPS coordinates.
  - Identify landmarks

# Keyboard

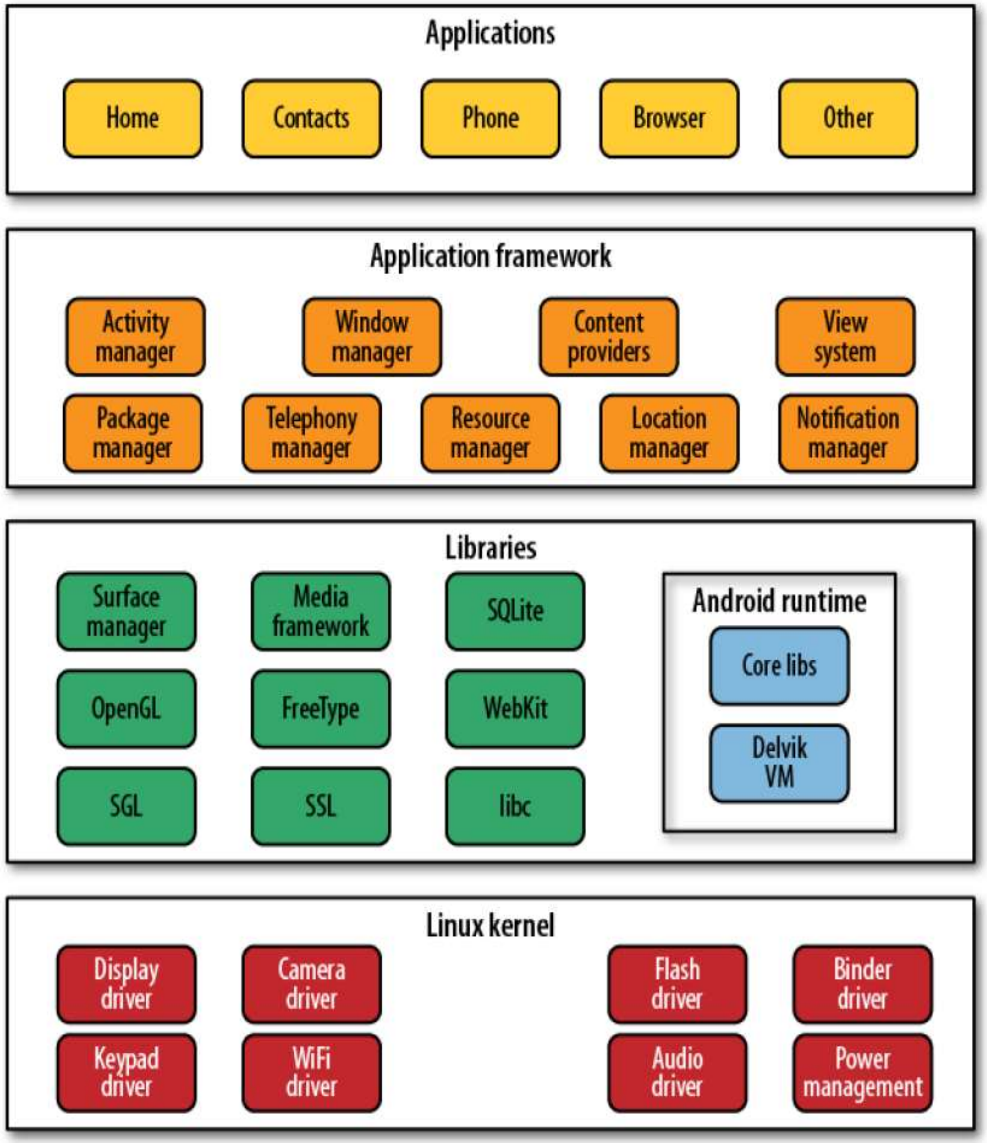
- On-screen keyboard
- Adapt to the screen orientation
- Swype keyboard:
  - Determines the likely word and completes it or offers suggestions

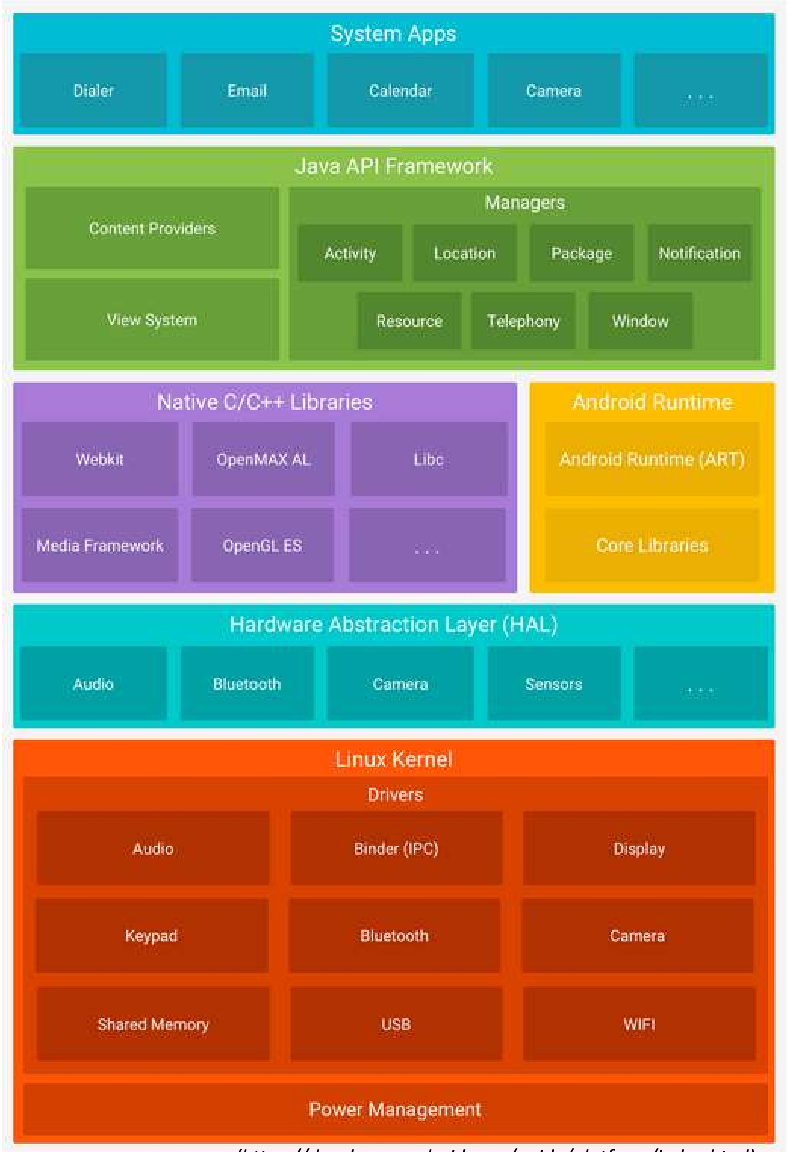


## Universal Serial Bus

- Allows most modern OSs connectivity to the device
- Interfaces exposed by Android devices:
  - Charge only
  - Disk interface: portions of the device, including the SD card, emulated SD card, and other disk interfaces -> Mass Storage Device
  - Vendor-specific interfaces: custom synchronization protocols, emulated CD-read-only memory (ROM) drives for software installs, and specialized connections for sharing the phone’s Internet connection
  - Android Debug Bridge (ADB): an interface that provides the user access to a shell prompt on the device as well as other advanced features

# Android Operating Systems

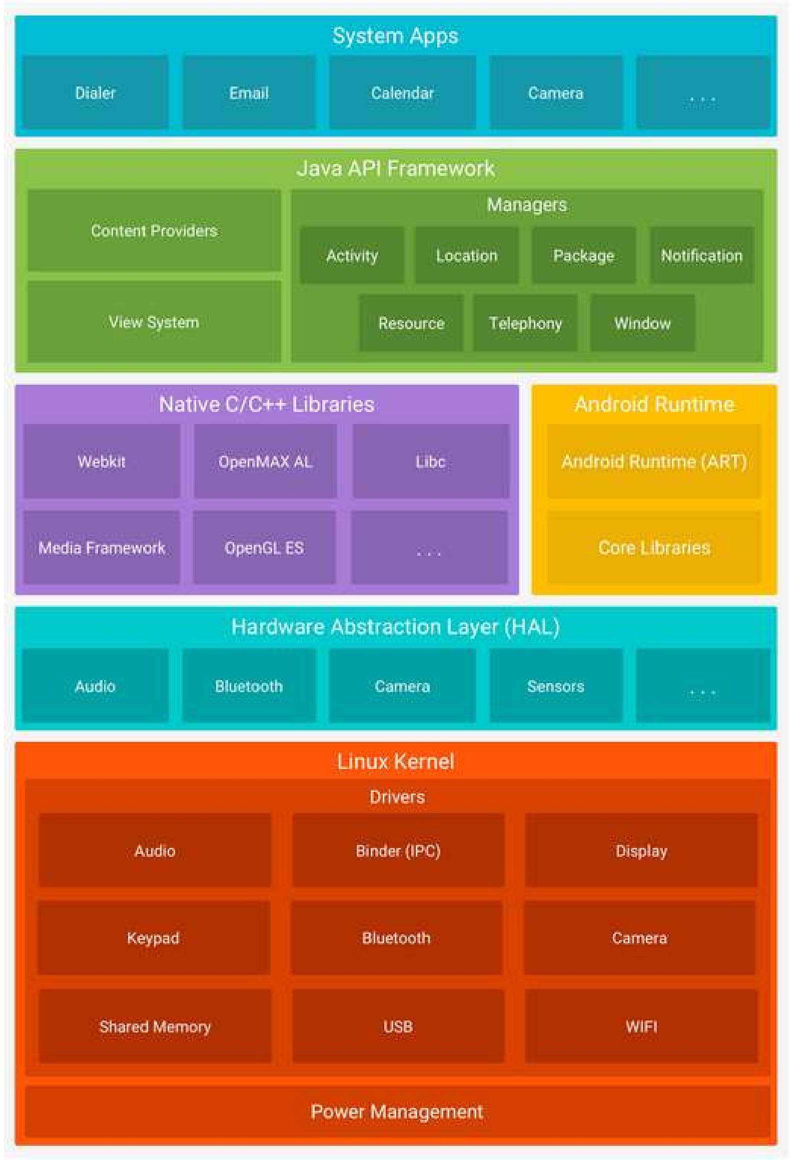




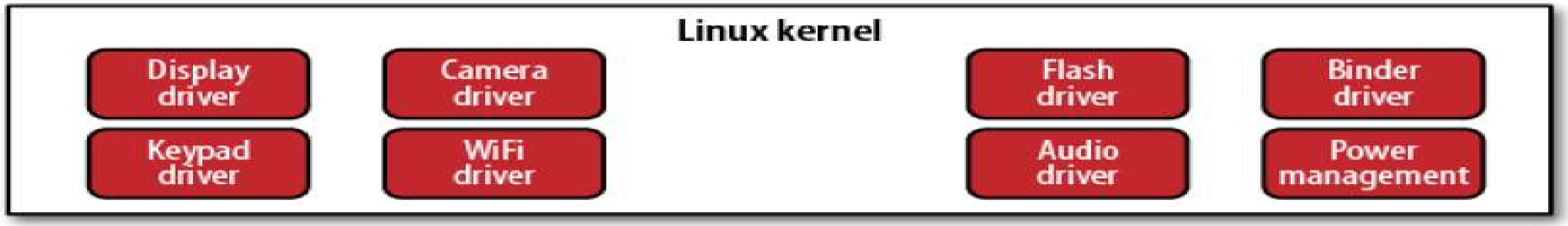
*(https://developer.android.com/guide/platform/index.html)*

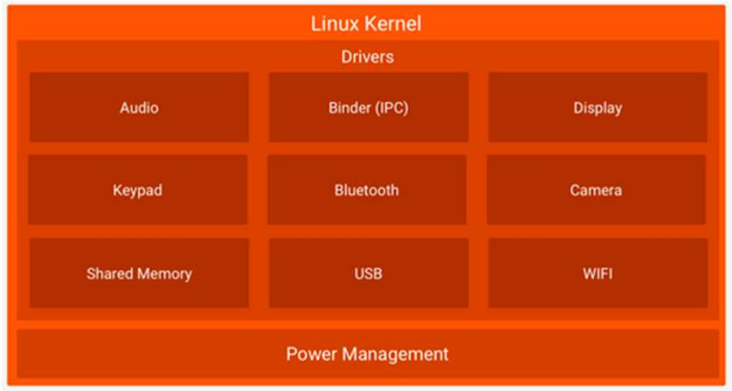
# The Stack

- **What is the Stack?**
  - The system layout / Android Platform
- **Why you need it?**
  - To help you shape your understanding about what you can or cannot do easily with Android.
- **How it looks like?**
  - Layered cake: "*The Android operating system is like a cake consisting of various layers. Each layer has its own characteristics and purpose. The layers are not cleanly separated but often seep into each other.*"



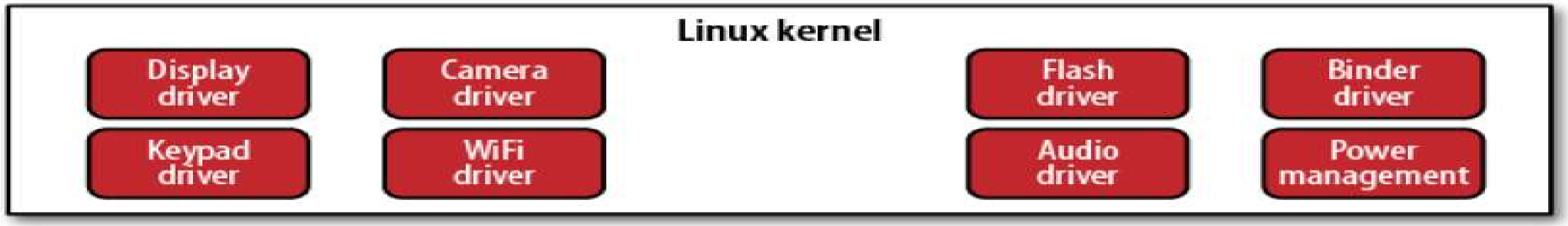
# Layer 1 – Linux Kernel / HW





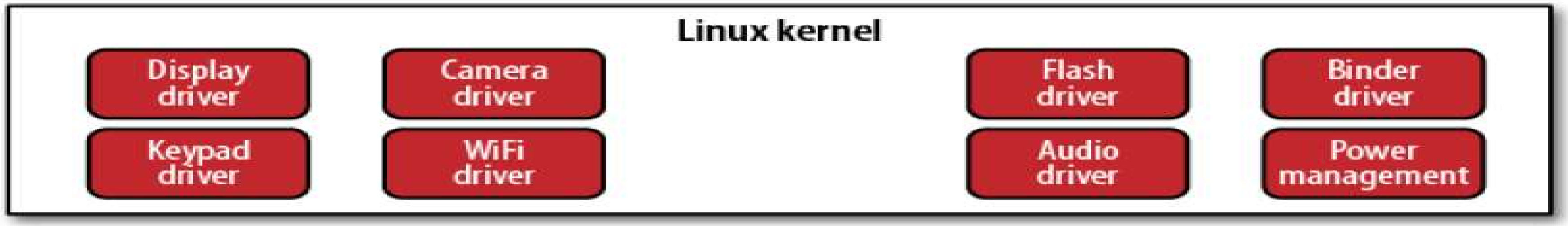
- Android is built on top of Linux.
- Linux is a great operating system and the poster child of open source.
- There are many good reasons for choosing Linux as the base of the Android stack. Some of the main ones are its portability, security, and features.

# Layer 1 – Linux Kernel / HW



| Portability | Security | Features |
|---|---|---|
| ▪ Linux based makes it easy to compile on various hardware architectures.<br>▪ Hardware abstractions: we don't have to worry too much about underlying hardware features.<br>▪ HW friendly because it is written in fairly portable C code. | ▪ Android heavily relies on Linux for security, which is tested through some very harsh environments over the decades.<br>▪ Application signing, and Application-defined and user-granted permissions | ▪ Memory Management<br>▪ Power Management<br>▪ Networking<br>▪ Friendliness<br>▪ Rapid innovation curve |

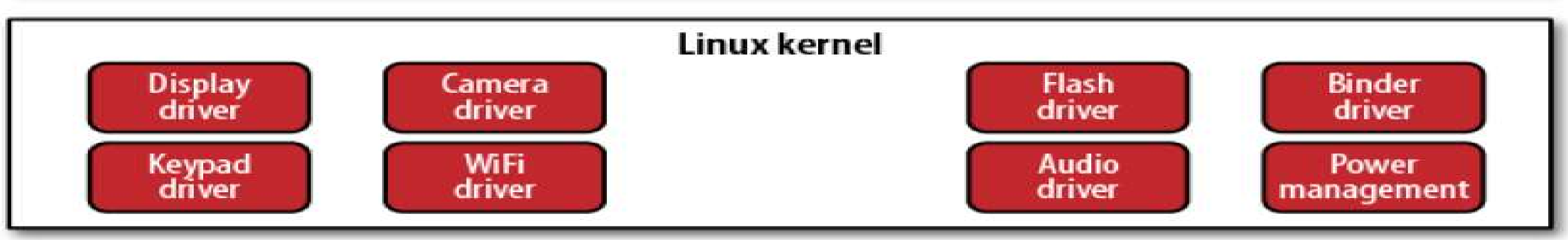
# Layer 1 – Linux Kernel / HW



**Portability**

- Linux based makes it easy to compile on various hardware architectures.
- Hardware abstractions: we don't have to worry too much about underlying hardware features.
- HW friendly because it is written in fairly portable C code.

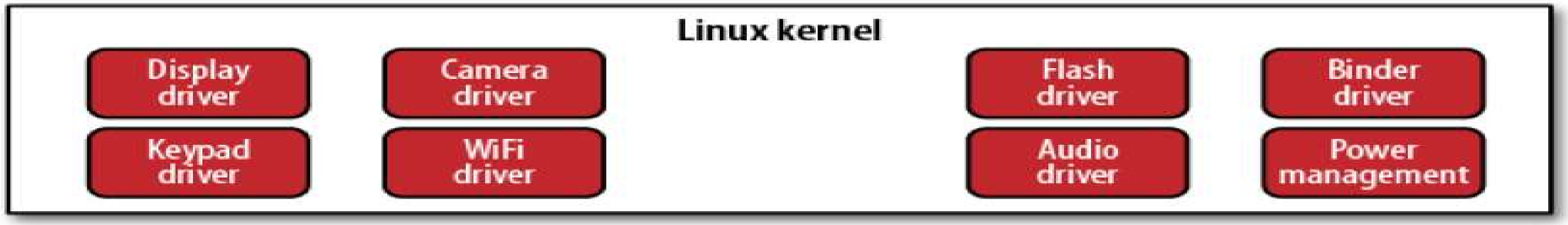
# Layer 1 – Linux Kernel / HW



**Security**

- Android heavily relies on Linux for security, which is tested through some very harsh environments over the decades.
- Application signing, and Application-defined and user-granted permissions

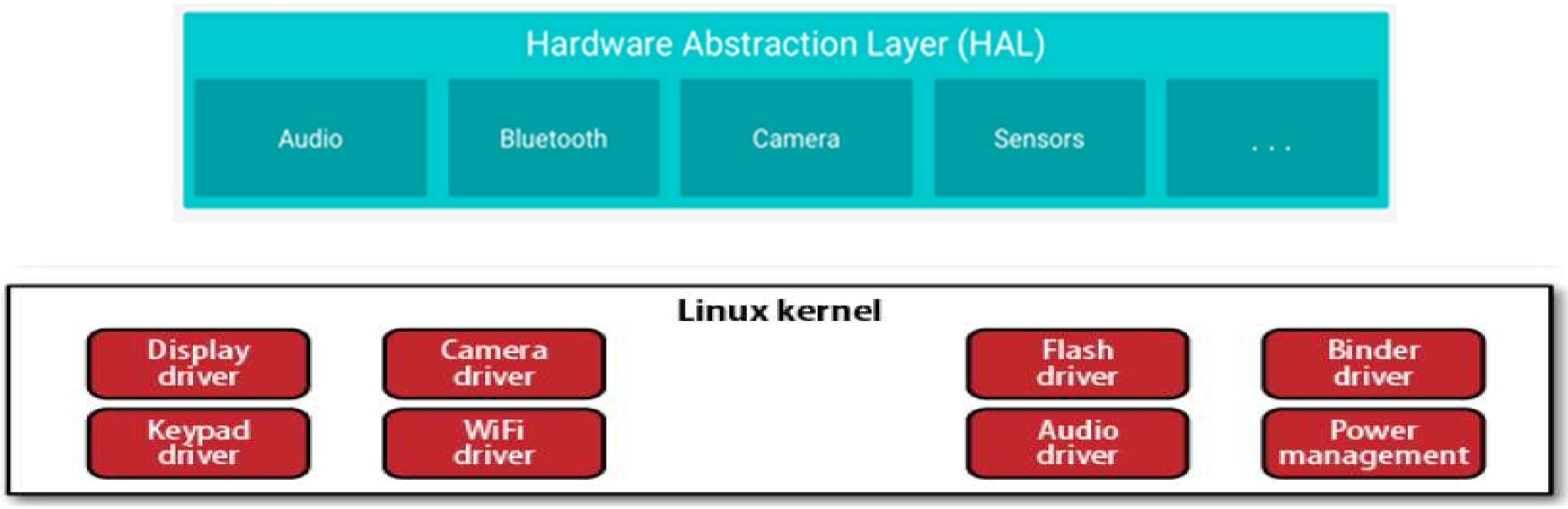
# Layer 1 – Linux Kernel / HW



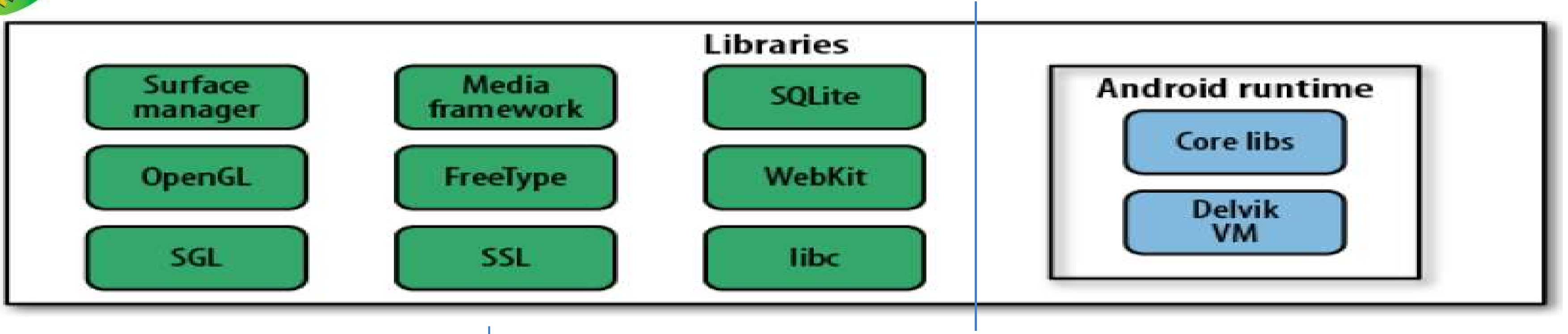
**Features**

- Memory Management
- Power Management
- Networking
- Friendliness
- Rapid innovation curve

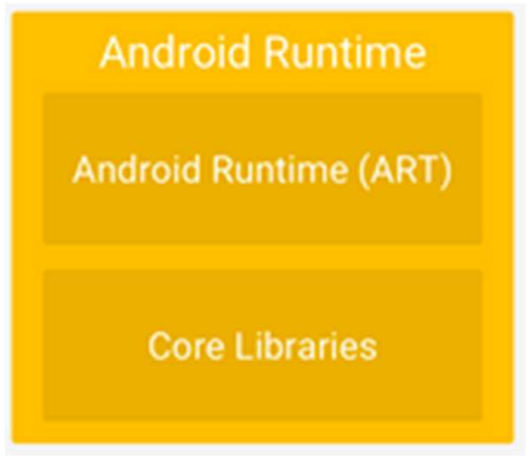
# Layer 1 – Linux Kernel / HW



# Layer 2 – Native Libraries

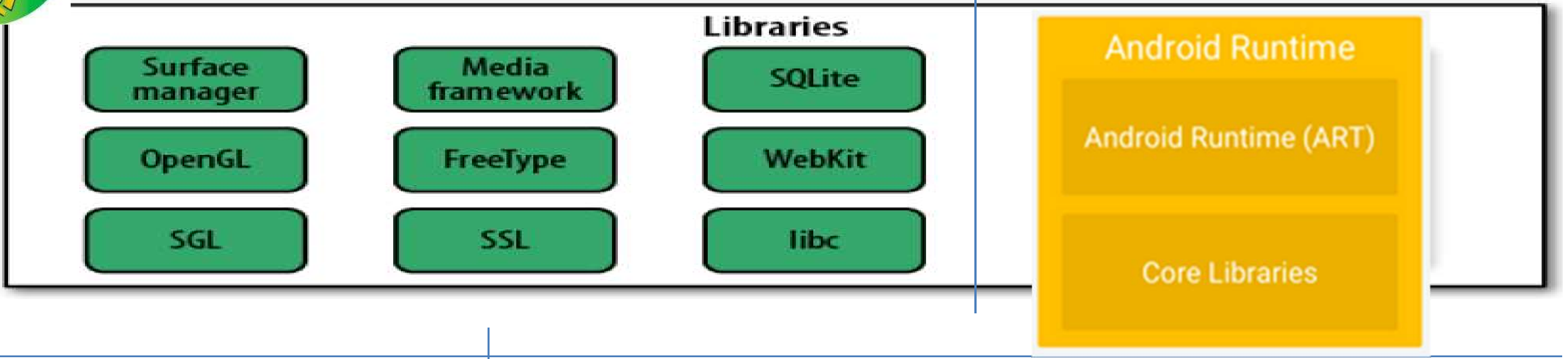


The native libraries are C/C++ libraries, often taken from the open source community in order to provide necessary services to the Android application layer. Among others, they include:



- ***Webkit*** *A fast web-rendering engine used by Safari, Chrome, and other browsers.*
- ***SQLite*** *A full-featured SQL database.*
- ***Apache Harmony*** *An open source implementation of Java.*
- ***OpenGL*** *3D graphics libraries.*
- ***OpenSSL*** *The secure locket layer*

# Layer 2 – Native Libraries



Although many of these libraries are used as-is, one notable exception is Bionic, which is basically a rewritten version of the standard C library.

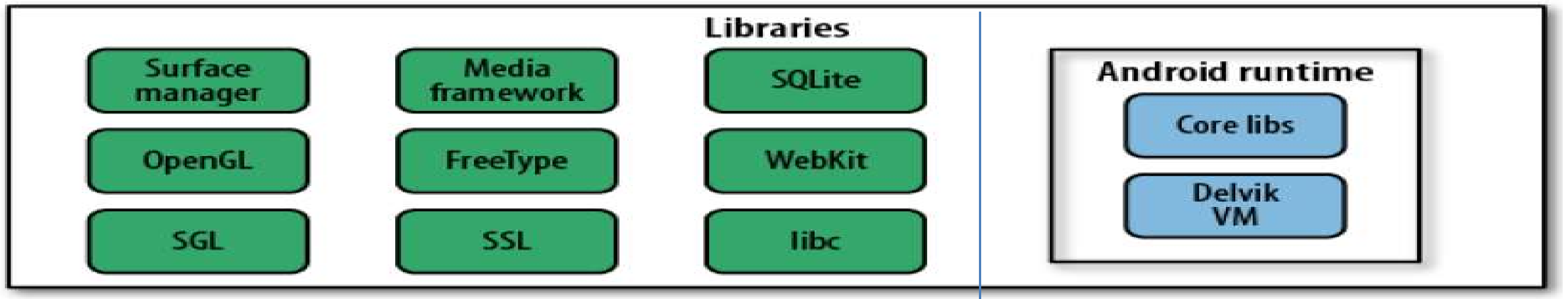
Bionic is used for two reasons:

- **Technology** To make it purpose-built for tiny, battery-powered devices.

BSD

- **License** To make it license-friendly for others who might want to adopt it and change it

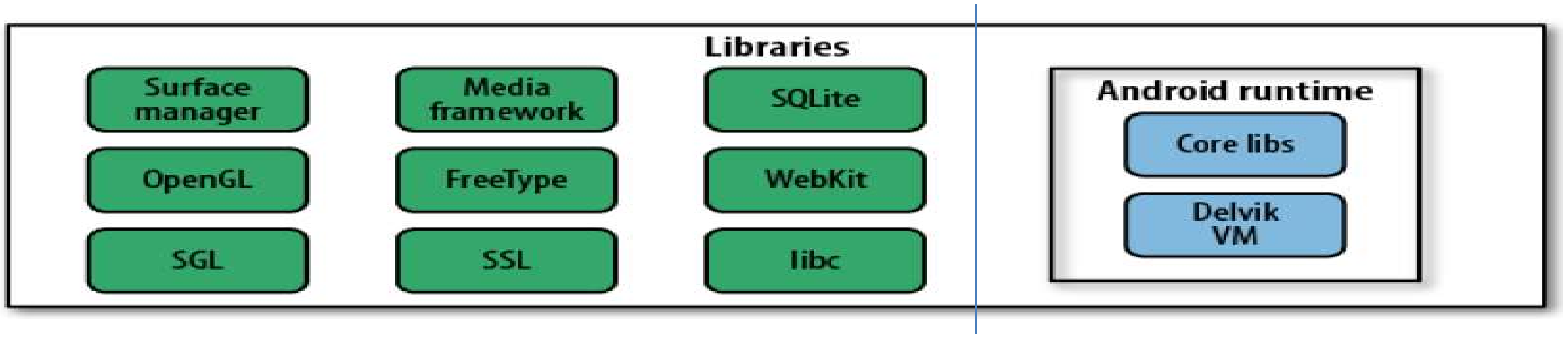
# Layer 2 – Native Libraries



**Dalvik is a purpose-built virtual machine designed specifically for Android, developed by Dan Bornstein and his team at Google**.

The Java virtual machine (VM) was designed to be a one-size-fits-all solution, and the Dalvik team felt they could do a better job by focusing strictly on mobile devices. They looked at which constraints specific to a mobile environment are least likely to change in the near future. One of these is battery life, and the other is processing power. Dalvik was built from the ground up to address those constraints.

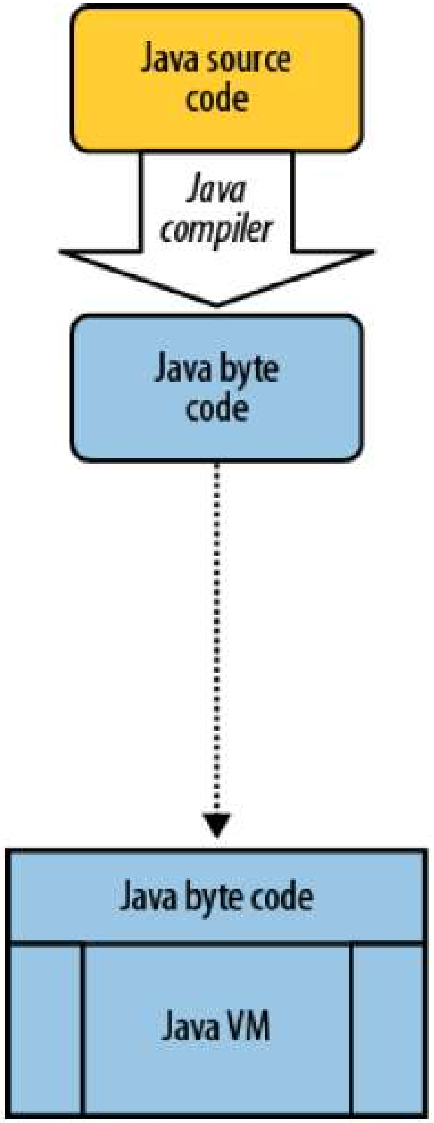
# Layer 2 – Native Libraries

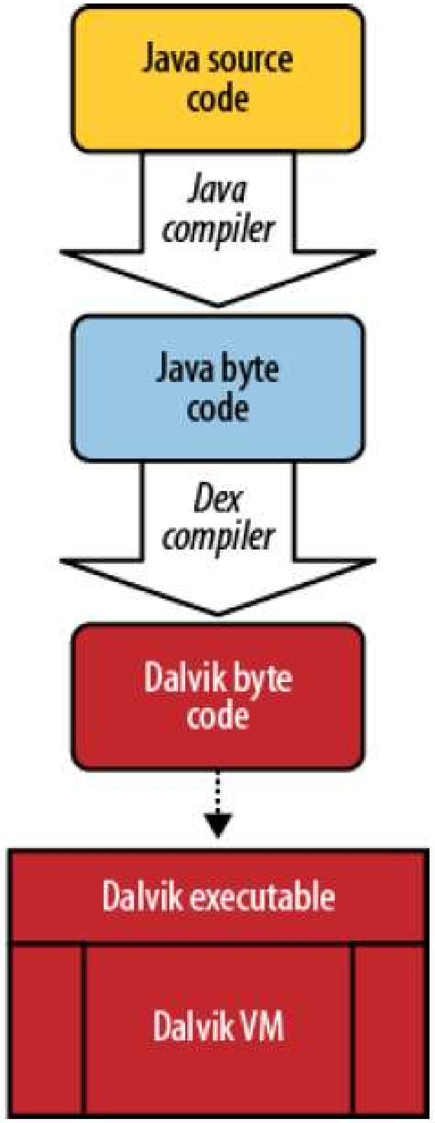


Still on Dalvik: Licensing.

Another side effect of replacing the Java VM with the Dalvik VM is the licensing. Whereas the Java language, Java tools, and Java libraries are free, **the Java virtual machine is not**. This was more of an issue back in 2005 when the work on Dalvik started. Nowadays, there are open source alternatives to Sun's Java VM, namely the OpenJDK and Apache Harmony projects.

## Side note on Dalvik – Part 1





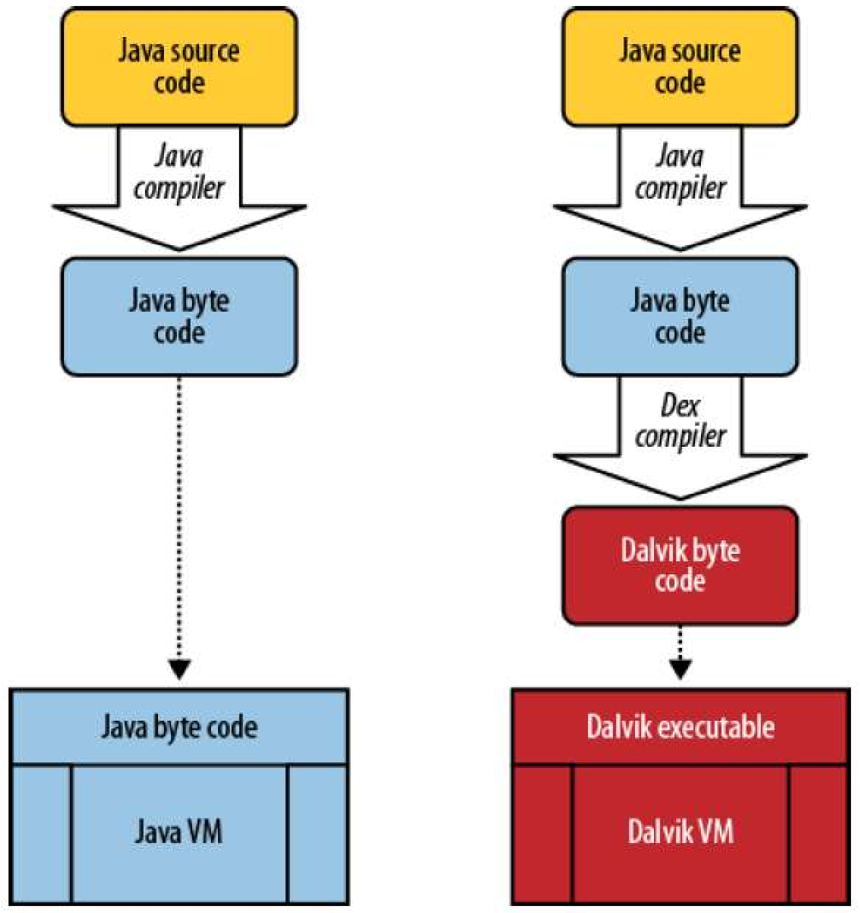
# Side note on Dalvik – Part 1

In Java, you write your Java source file, compile it into a Java byte code using the Java compiler, and then run this byte code on the Java VM.

In Android, things are different.

You still write the Java source file, and you still compile it to Java byte code using the same Java compiler. But at that point, **you recompile it once again using the Dalvik compiler to Dalvik byte code.** It is this Dalvik byte code that is then executed on the Dalvik VM.

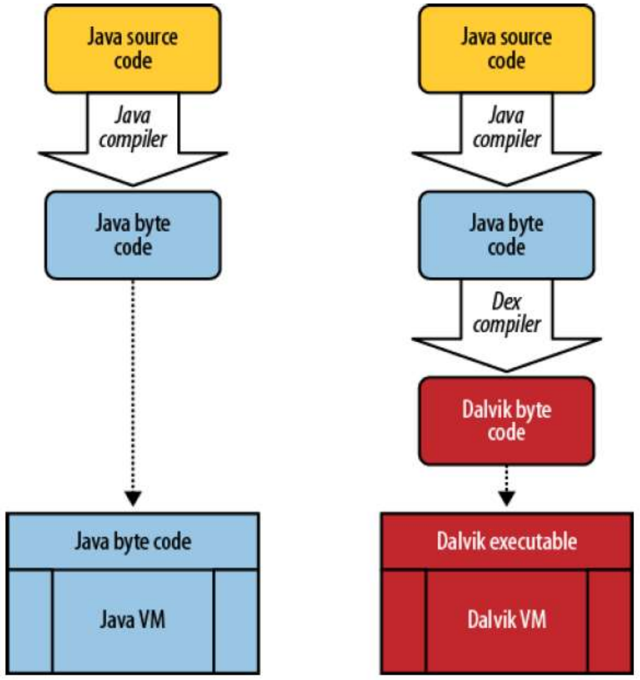
Why?



# Side note on Dalvik – Part 2

**Why not compile straight from Java into the Dalvik byte code?**

Back in 2005, when work on Dalvik started, the Java language was going through frequent changes, but the Java byte code was more or less set in stone. So, the Android team chose to base Dalvik on Java byte code instead of Java source code.



## Side note on Dalvik – Part 3

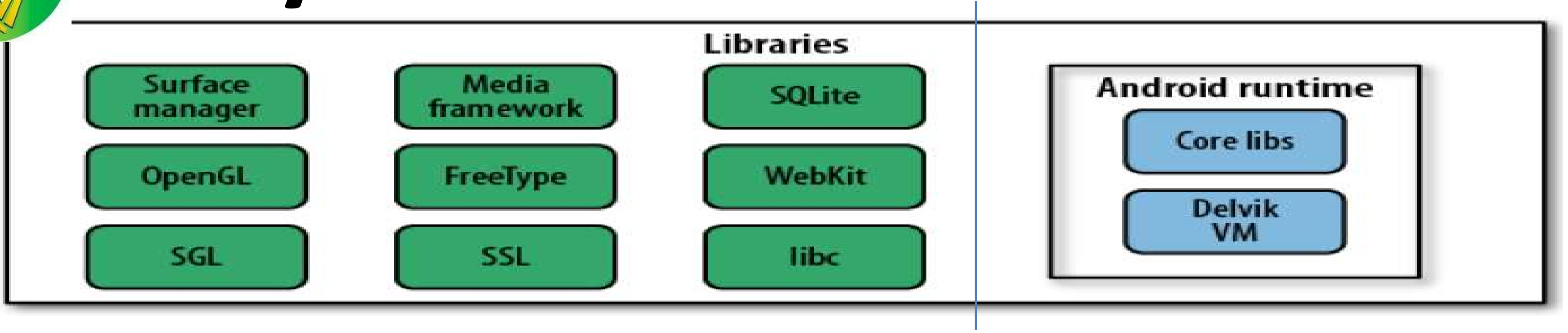
Android Java is a nonstandard collection of Java classes.

Java typically ships in:

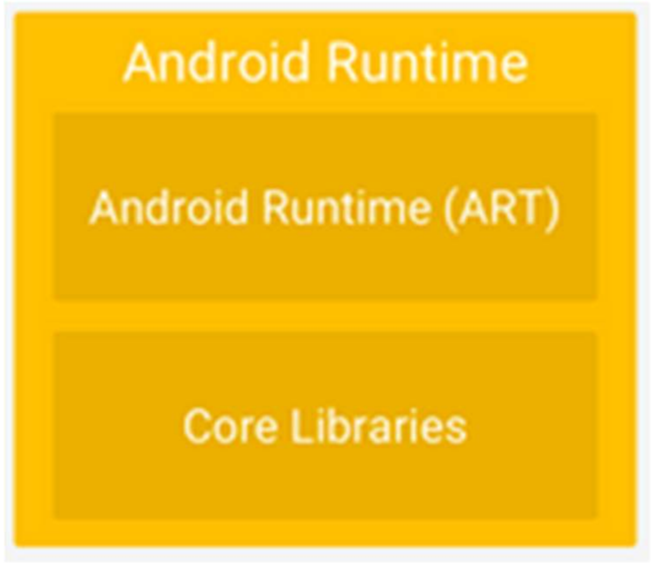
- Java Standard Edition Used for development on basic desktop-type applications.
- Java Enterprise Edition (aka J2EE or JavaEE) Used for development of enterprise applications.
- Java Micro Edition (aka J2ME or JavaME) Java for mobile applications.

Android's Java set of libraries is closest to Java Standard Edition. The major difference is that Java user interface libraries (AWT and Swing) have been **taken out and replaced** with Android-specific user interface libraries.

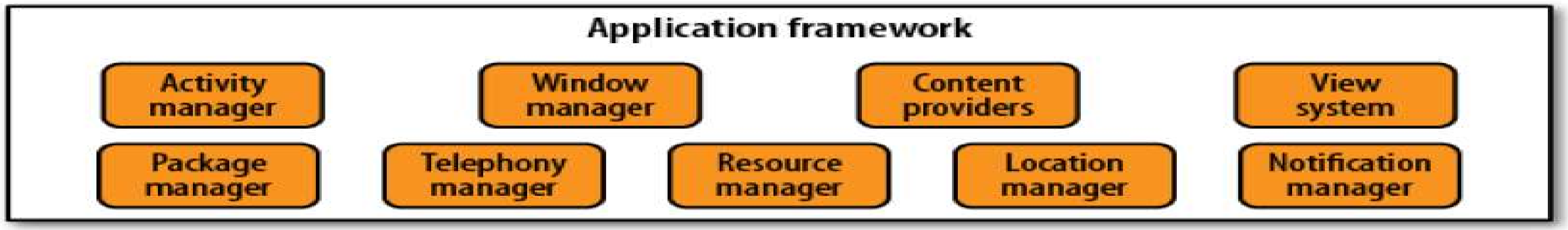
# Layer 2 – Native Libraries



- Devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the ART
- Run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed specially for Android that's optimized for minimal memory footprint.
- If your app runs well on ART, then it should work on Dalvik as well



# Layer 3 – Application Framework

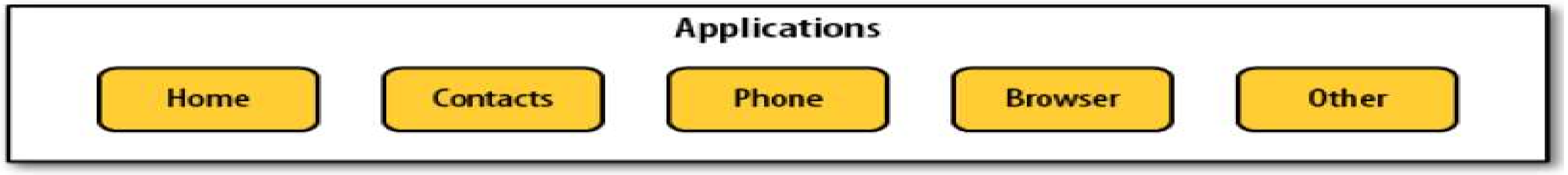


The application framework **is a rich environment that provides numerous services to help you, the app developer, get your job done.**

In the application framework layer, you will find numerous **Java libraries specifically built for Android.**

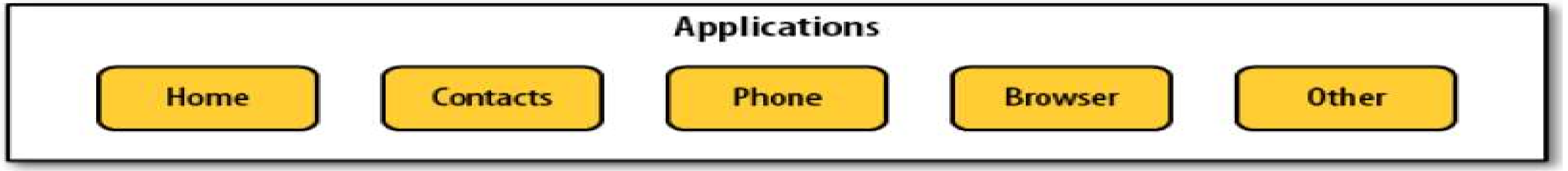
You will also find many services that provide the capabilities your application can link to, such as **location, sensors, WiFi, telephony, and so on.**

# Layer 4 – Applications / APKs



- This is what **you** are going to create.
- This is what **end users** find valuable about Android.
- They can come **preinstalled** on the device or can be **downloaded** from one of the many **Android markets**.

# Layer 4 – Applications / APKs



- An application is a single application package (APK) file.
- An APK file roughly has three main components.
  - **Dalvik executable**: This is all your Java source code compiled down to a Dalvik executable. This is the code that runs your application.
  - **Resources**: are everything that is not code. Your application may contain a number of images and audio/video clips, as well as numerous XML files describing layouts, language packs, and so on. Collectively, these items are the resources.
  - **Native libraries**: Optionally, your application may include some native code, such as C/C++ libraries. These libraries could be packaged together with your APK file.

Additional Functionalities →

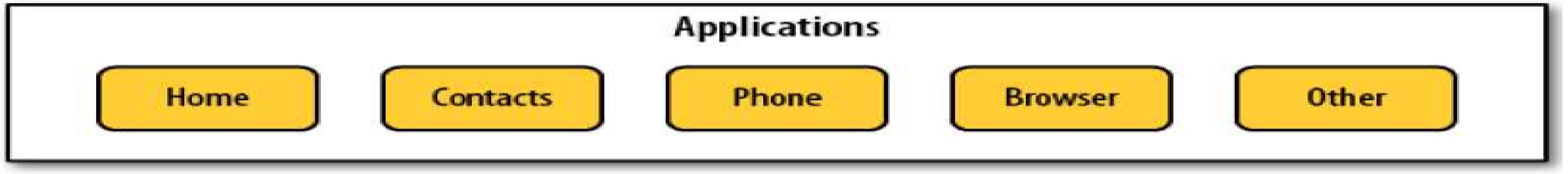
## Layer 4 – Applications / APKs

Applications
Home
Contacts
Phone
Browser
Other

An ***APK*** file roughly has <u>three main components</u>:

- **Dalvik executable**: This is all your Java source code compiled down to a Dalvik executable. This is the code that runs your application.

# Layer 4 – Applications / APKs



- An ***APK*** file roughly has <u>three main components</u>:

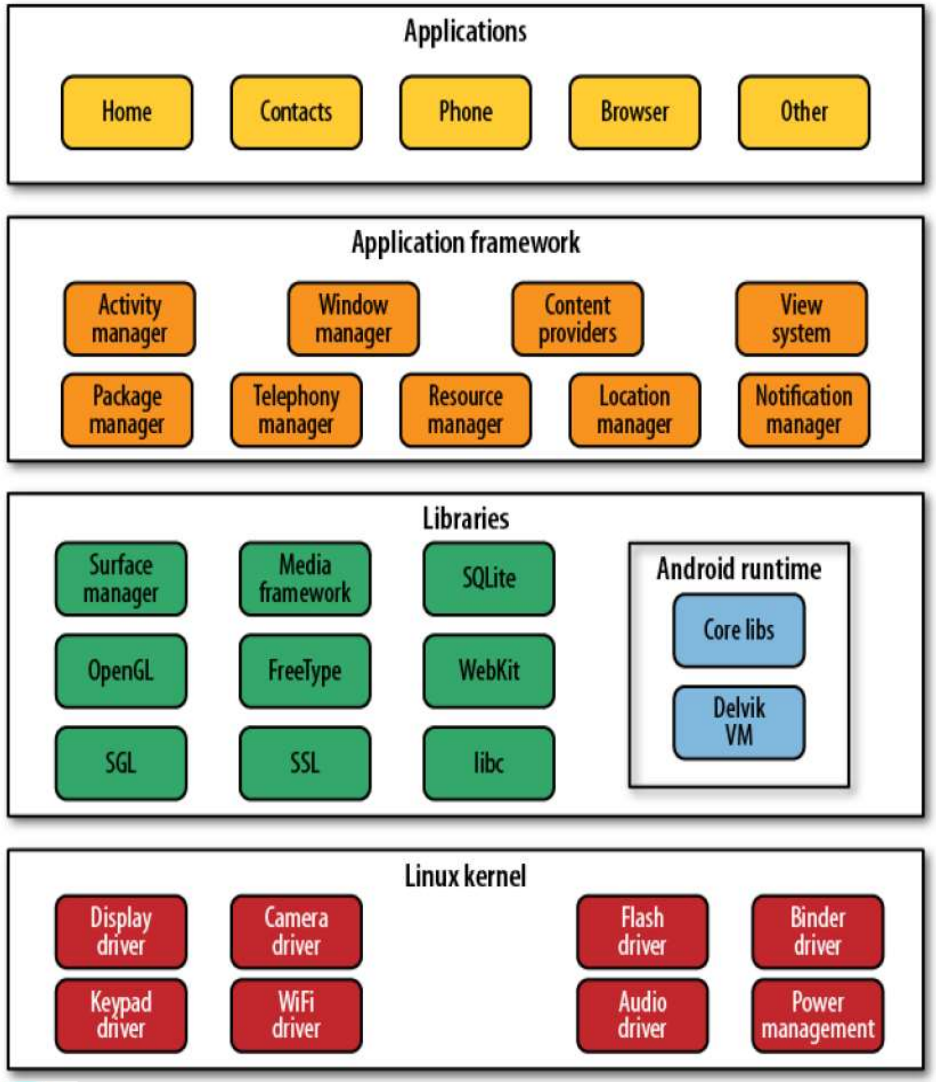
  - **Resources**: are everything that is not code. Your application may contain a number of images and audio/video clips, as well as numerous XML files describing layouts, language packs, and so on. Collectively, these items are the resources.

## Layer 4 – Applications / APKs

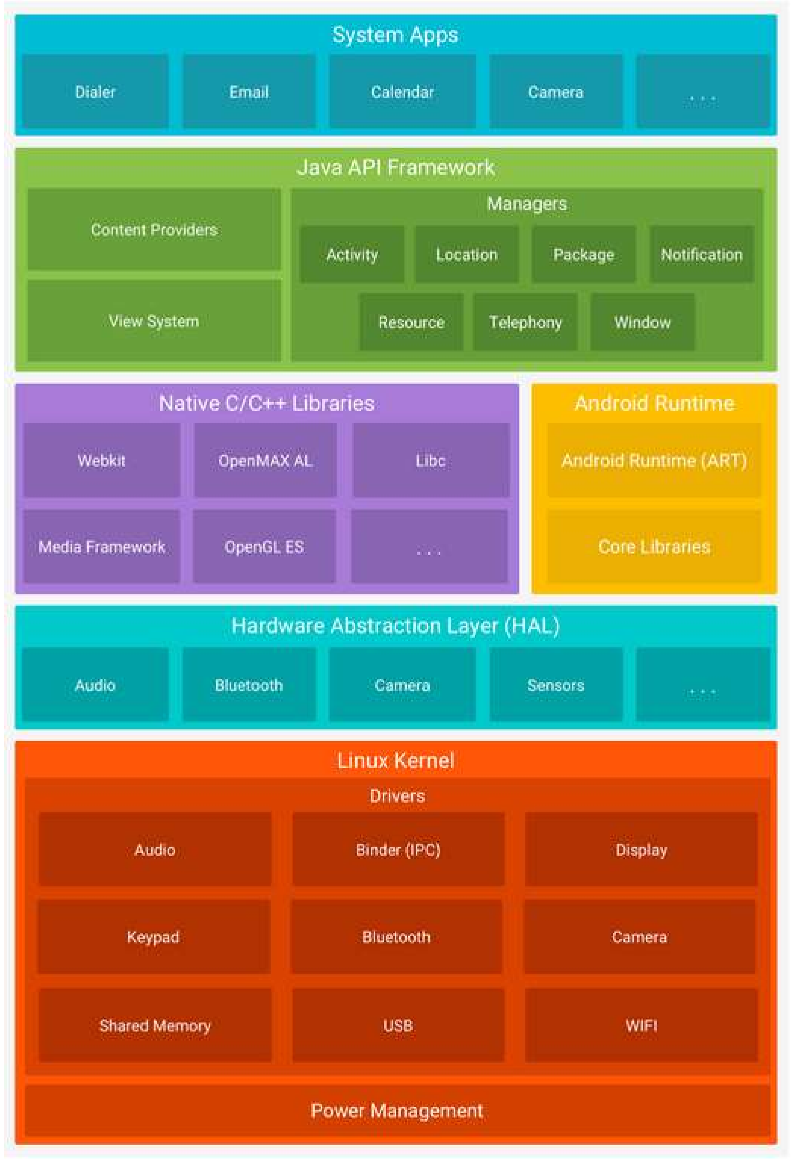
Applications
Home
Contacts
Phone
Browser
Other

- An ***APK*** file roughly has <u>three main components </u>:
    - **Native libraries**: Optionally, your application may include some native code, such as C/C++ libraries. These libraries could be packaged together with your APK file.

# The cake



Applications
Home
Contacts
Phone
Browser
Other
Application framework
Activity manager
Window manager
Content providers
View system
Package manager
Telephony manager
Resource manager
Location manager
Notification manager
Libraries
Surface manager
Media framework
SQLite
OpenGL
FreeType
WebKit
SGL
SSL
libc
Android runtime
Core libs
Delvik VM
Linux kernel
Display driver
Camera driver
Flash driver
Binder driver
Keypad driver
WiFi driver
Audio driver
Power management



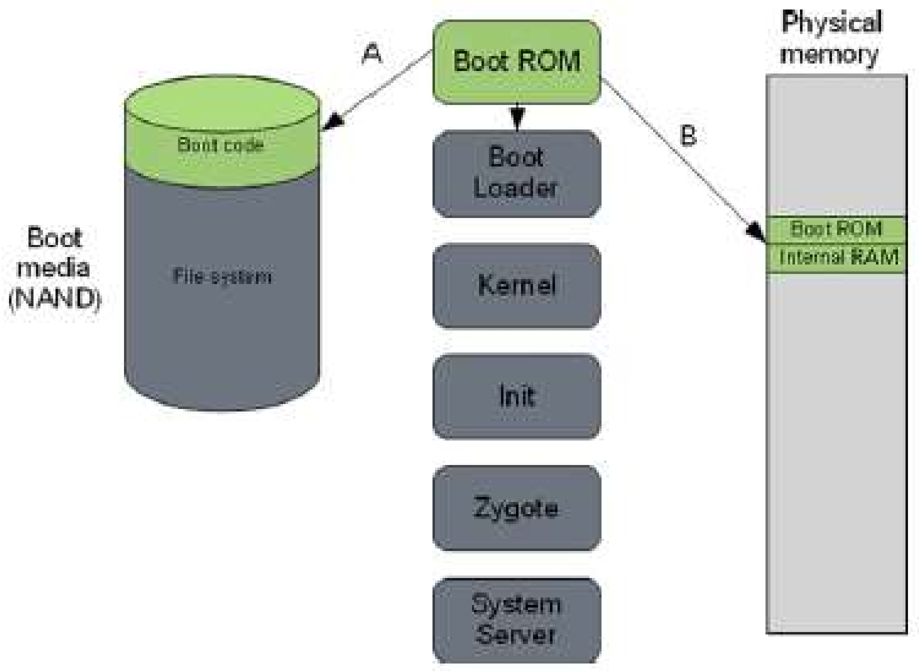
System Apps
Dialer
Email
Calendar
Camera
. . .
Java API Framework
Content Providers
Managers
Activity
Location
Package
Notification
View System
Resource
Telephony
Window
Native C/C++ Libraries
Webkit
OpenMAX AL
Libc
Media Framework
OpenGL ES
. . .
Android Runtime
Android Runtime (ART)
Core Libraries
Hardware Abstraction Layer (HAL)
Audio
Bluetooth
Camera
Sensors
. . .
Linux Kernel
Drivers
Audio
Binder (IPC)
Display
Keypad
Bluetooth
Camera
Shared Memory
USB
WIFI
Power Management

## ROM and Boot Loader

1. Power on and on-chip boot ROM code execution
2. The boot loader
3. The Linux kernel
4. The init process
5. Zygote and Dalvik (ART)
6. The system server
7. Boot complete

# Power On and On-chip Boot ROM Code Execution

- A special boot ROM code paired with the CPU is executed to
  1. initialize the device hardware and
  2. locate the boot media. The ROM code is specific to the CPU the device is using.  This step is similar to the basic input-output system used to boot computers

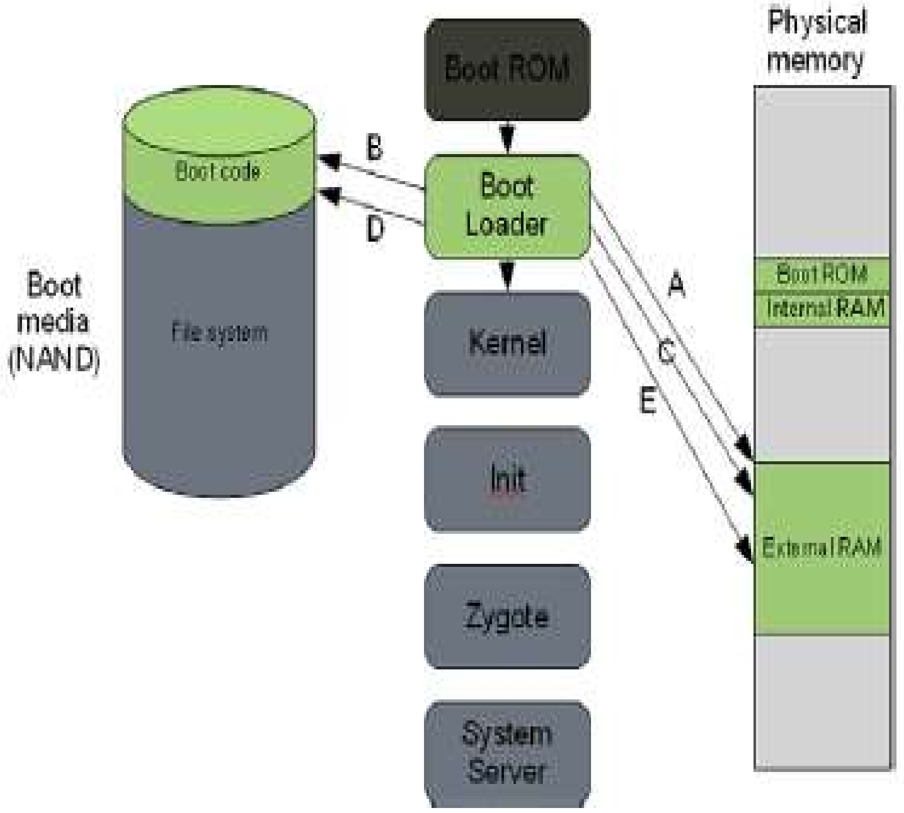


## Boot Loader

- Two distinct stages: the initial program load (IPL) and the second program loader (SPL).
- The IPL is responsible for detecting and setting up external RAM, an essential component needed to boot and operate the device. Once external RAM is prepared, the IPL copies the SPL into RAM and then transfers execution to the SPL
- The SPL is responsible for not only loading the Android OS but also providing access to alternative boot modes: fastboot, recovery, or other modes designed to update and debug or service the device. The SPL is generally provided by the manufacturer.

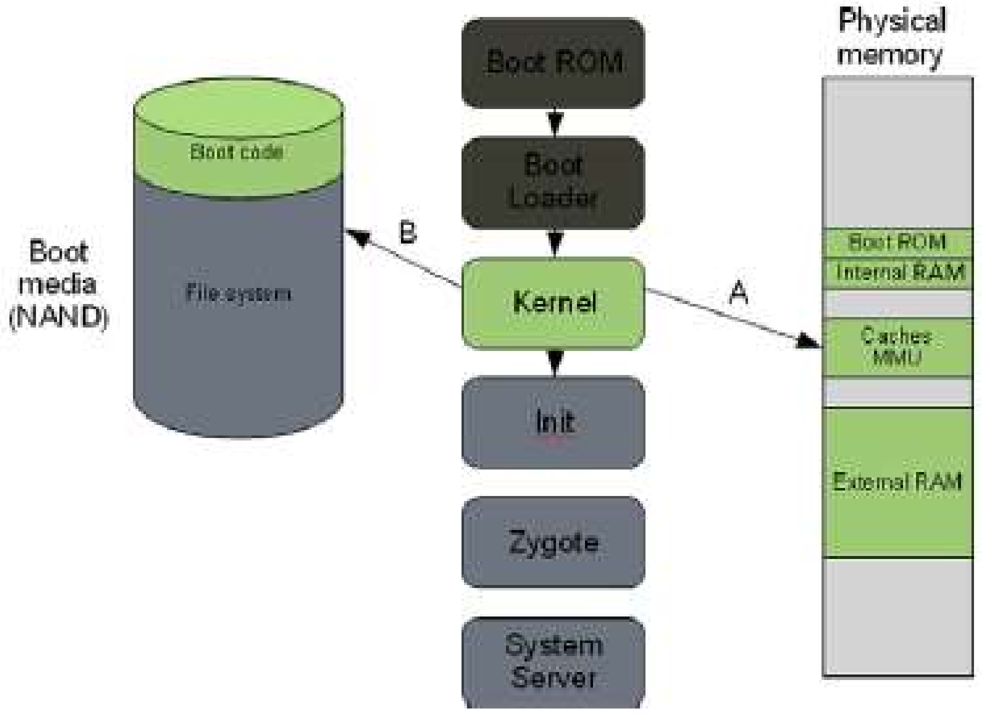
# Boot Loader

- Customise SPL:
  - initialize hardware components such as the clock, console, display, keyboard, and baseband modem as well as file systems, virtual memory, and other features
- The SPL locates the Linux kernel on the boot media, copies it to RAM, loads boot parameters, and then transfers execution to the kernel required to operate the device.



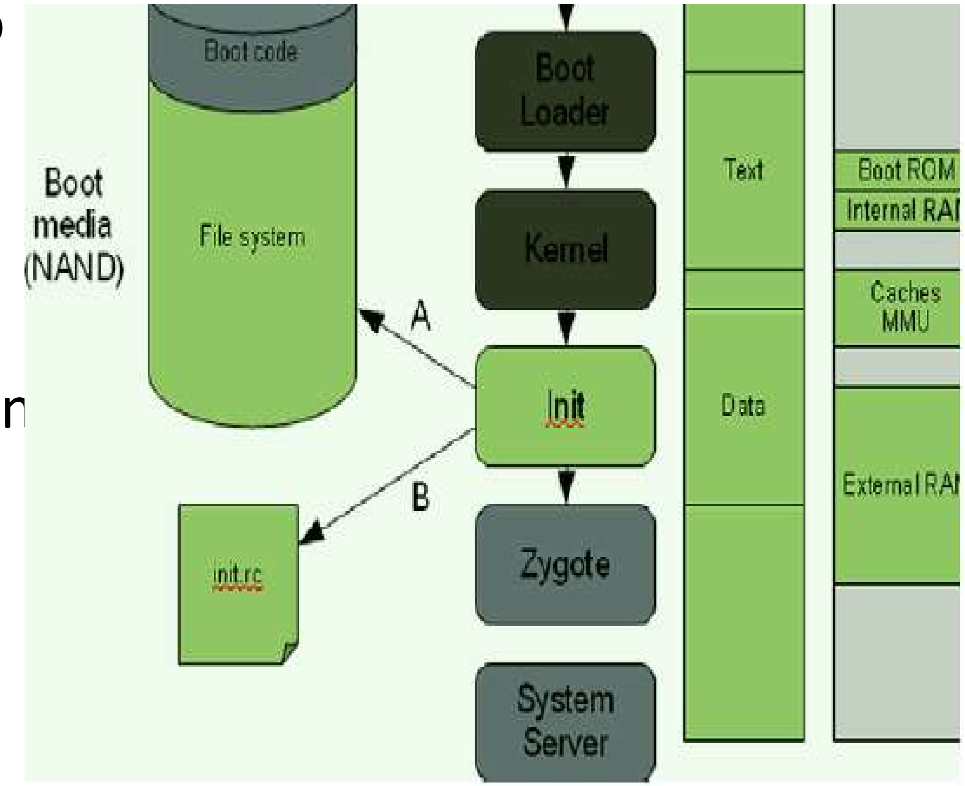
# Linux kernel

- Linux kernel is now controlling the device.
- After setting up additional features on the device, the root file system is read from the NAND flash, which will provide access to system and user data



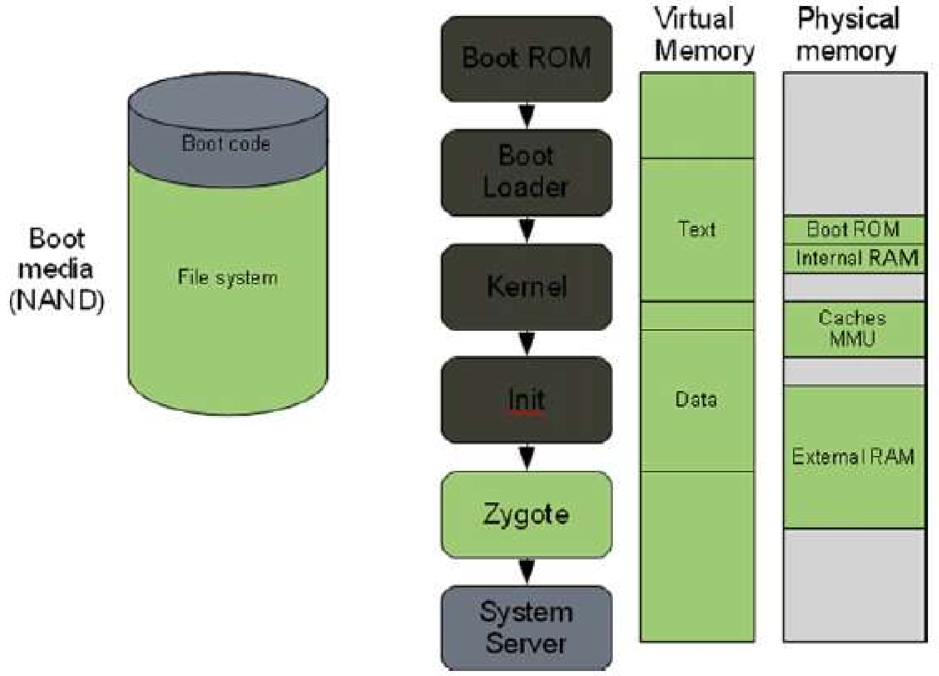
# Init Process

- Once the kernel has access to the system partition, it can process the **init** scripts that start key system and user processes. This is similar to the /etc/init.d scripts found on traditional Linux devices.
- For Android, the init.rc is typically located on the root file system and provides the kernel with the details on how to start core services



# Zygote and Dalvik

- The Dalvik virtual to create this application sandbox.
- The Zygote sequence essentially sets up the JRE and registers a socket with the system -> new applications that need to initialize can request a new Dalvik virtual machine.
- Without the Zygote service: the Android kernel could run. However, no applications would operate including built-in applications such as the phone, browser, and other core features



## System Server

- The core features of the are started by the system server. This runs core features such as telephony, network, and other fundamental components that the device and other applications rely upon
- The system finally sends a standard broadcast action called ACTION_BOOT_COMPLETED, which alerts dependent processes that the boot process is complete. The Android system is now fully operational and is ready to interact with the user.

## Application Signing

- Android applications must be signed before they can be installed on a device.

- When you distribute your application commercially, you'll want to sign it with your own key.

$\left.\begin{matrix}\\ \\ \end{matrix}\right\}$ First step on becoming app developer.

- The Android developer document titled "Signing Your Application" has the details.

# Application Distribution – Part 1

- On most other platforms, such as iPhone, a **single vendor** holds a **monopoly** over the distribution of applications.
- On Android, there are many **different stores, or markets.**
- Each market has its own **set of policies** with respect to what is allowed, how **the revenue is split**, and so on.
- As such, Android is much more of a free market space in which vendors compete for business.

$\}$ Do some research before deciding where to publish your app.

# pplication Distribution – Part 2

- In practice, the **biggest market** currently is **Android Market**, run by **Google**.
- It is unclear whether Google means to just seed the market space while **other stores develop or plans to make it a profitable** venture.
- Applications can also be distributed via the Web. When you download an **APK file** from a website through the browser, the application represented by the APK file is **installed automatically** on your phone.

# Malware ++

- With its **decentralized** application distribution system, it is certainly possible for an unsuspecting user to download a malicious app that consequently does bad things.
- For example, there have been reports of phishing attacks via **fake banking apps**.
- So, Android leaves it to **the marketplace to sort it out**.
- Eventually, there will be stores that are more **reputable** and those that are less so, at least in theory.
- Google relies on user reports for policing its Android Market, but other stores may choose to do more proactive testing and raise the bar on what gets into the store in the first place.

Second step on becoming app developer.

# Main Building Blocks

- **What are they?**
  - The **components** you put together to develop an Android Application.
  - The conceptual items you put together to **draw your bigger picture.**
- **What are they used for?**
  - Starting to **think** about your application.
  - **Top-down** approach.
  - You design your application in terms of **screens, features, and the interactions between them**.
  - This approach to application development helps you see the big picture—how the **components fit together and how it all makes sense.**

# Main Building Blocks - Example

- **Example:**
  - **<u>Twitter App.</u>**

Three main screens

Basic Features

- **User should post updates.**
- **User should see what her friends are posting too.**
- **User should be able to set username and password.**

Background tasks

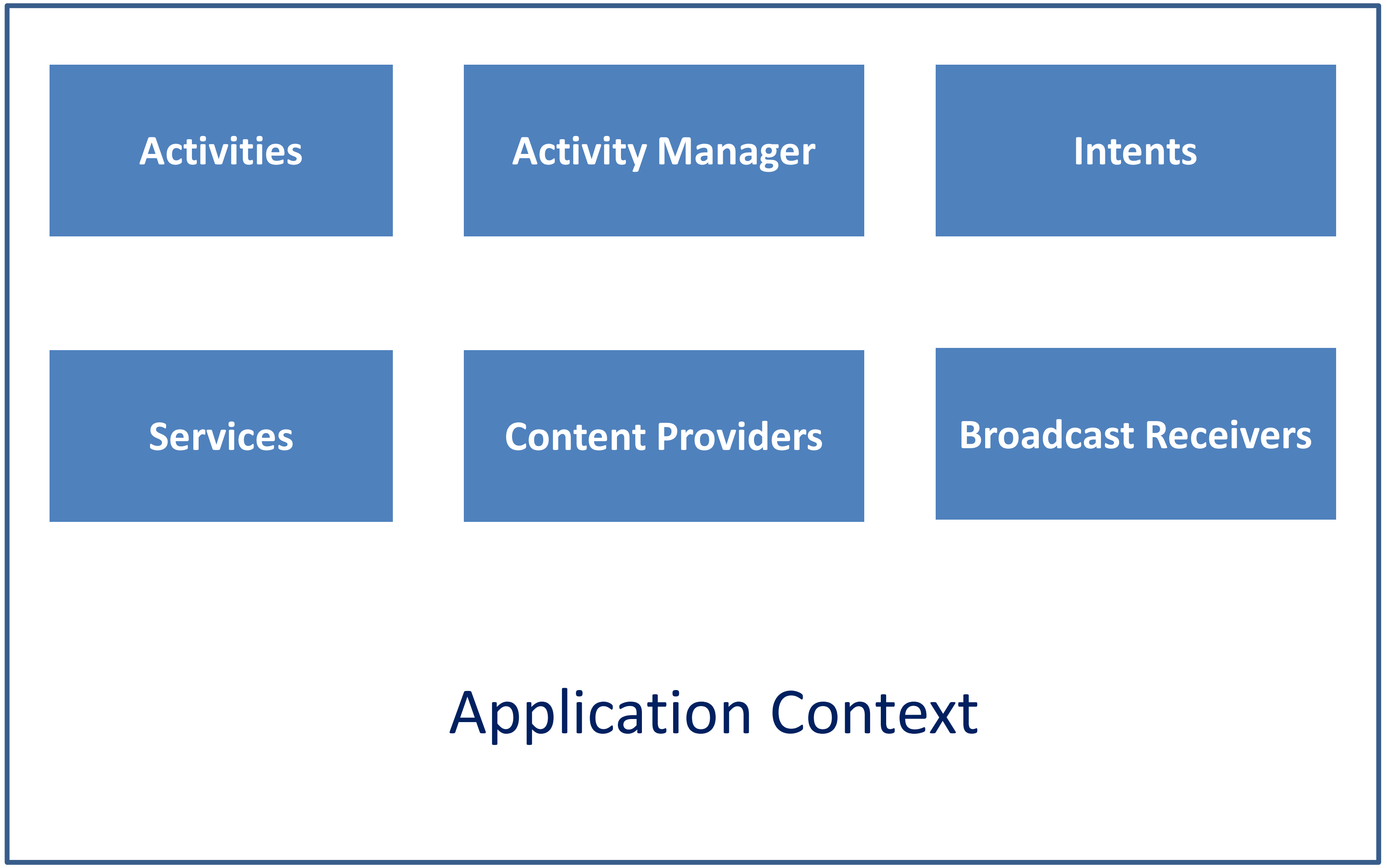
- **The app should work quickly regardless of the network connection.**
  - **The app should cache the data locally.**
    - **Service should run in the background.**
    - **A Database too.**

# Main Building Blocks - Example

Android building blocks make it easy to **break them down** into conceptual units so that you can **work on them independently**, and then easily **put them back together** into a complete package.

- **Example:**
  - **Twitter App.**
    - **User should post updates.**
    - **User should see what her friends are posting too.**
    - **User should be able to set username and password.**
    - **The app should work quickly regardless of the network connection.**
      - **The app should cache the data locally.**
        - **Service should run in the background.**
        - **A Database too.**

## MBB- Who are they?

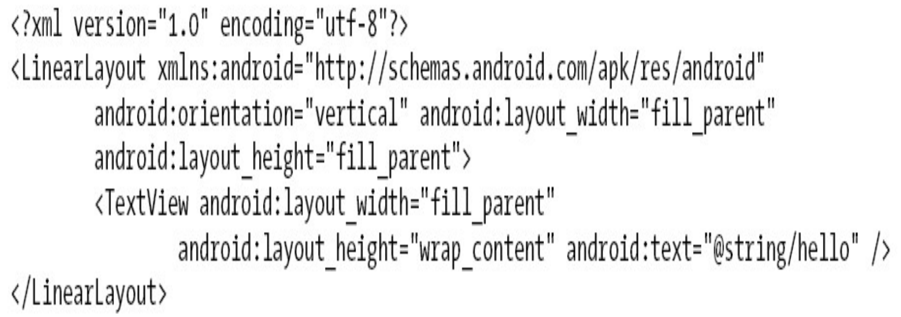


# Activities

- An activity is a **single screen** that the user sees on the device at one time.
- An application typically has **multiple activities**, and the user flips back and forth among them.
- Activities are the **most visible part** of your application.
- Just like a website consists of multiple pages, so does an Android application consist of multiple activities.
- Just like a website has a **"home page,"** an Android app has a **"main" activity**, usually the one that is shown first when you launch the application.
- And just like a website has to provide some sort of **navigation** among various pages, an Android app should do the same.

# Activities are expensive!!

- Launching a single activity in Android involves:
  - Creating new Linux process.
  - Allocating memory for all the UI objects.
  - Retrieving the XML Layouts.
  - Setting up the whole screen.



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView android:layout_width="fill_parent"
                android:layout_height="wrap_content" android:text="@string/hello" />
</LinearLayout>
```

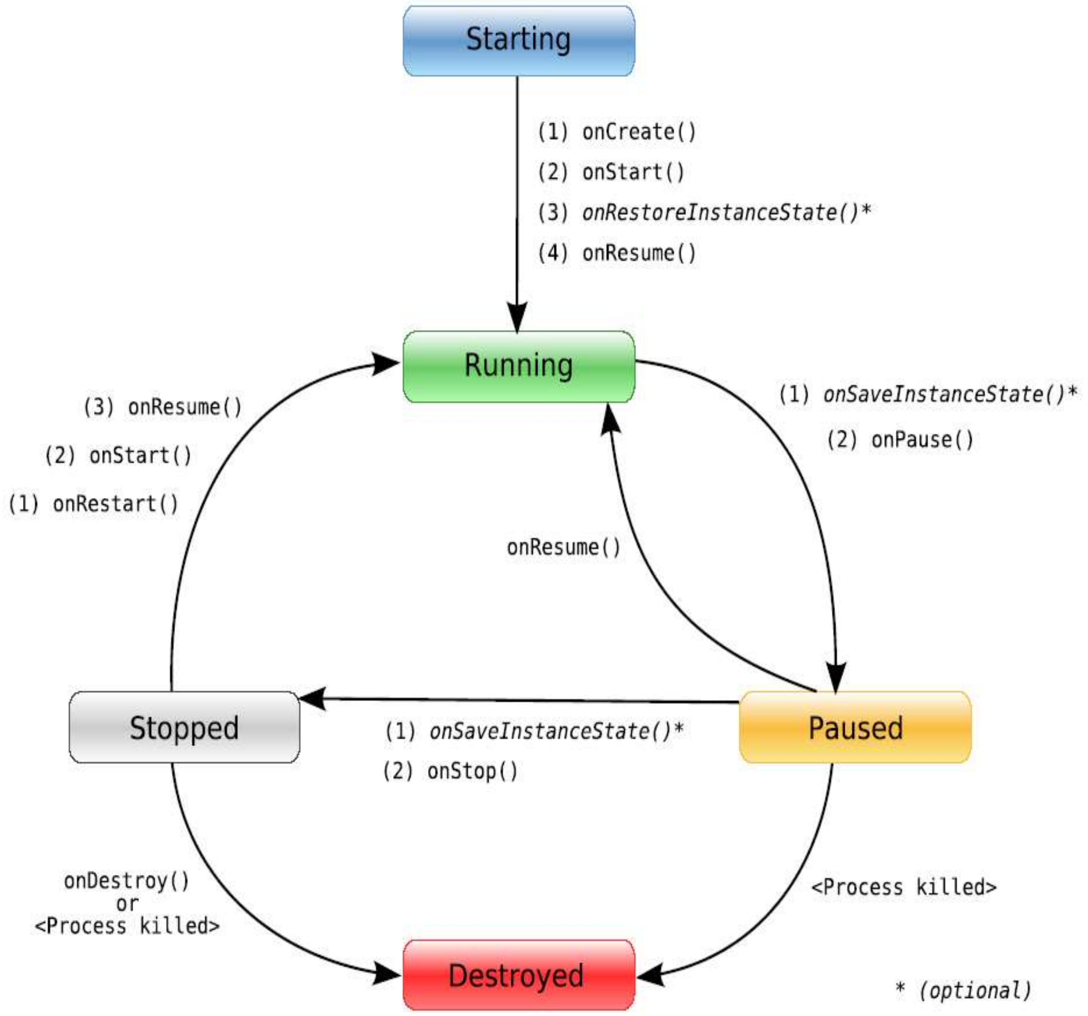
After paying this price, would you actually waste it once the user switches screens?

# Activity Manager

- Activity Manager is responsible for **creating**, **destroying**, and **managing** activities.
  - **For example,** when the user starts an application for the first time, the Activity Manager will create its activity and put it onto the screen.
  - Later, when the user switches screens, the Activity Manager will move that previous activity to a holding place.
  - This way, if the user wants to go back to an older activity, it can be started more quickly.
  - Older activities that the user hasn't used in a while will be destroyed in order to free more space for the currently active one.
  - This mechanism is designed to help improve the speed of the user interface and thus improve the overall user experience.

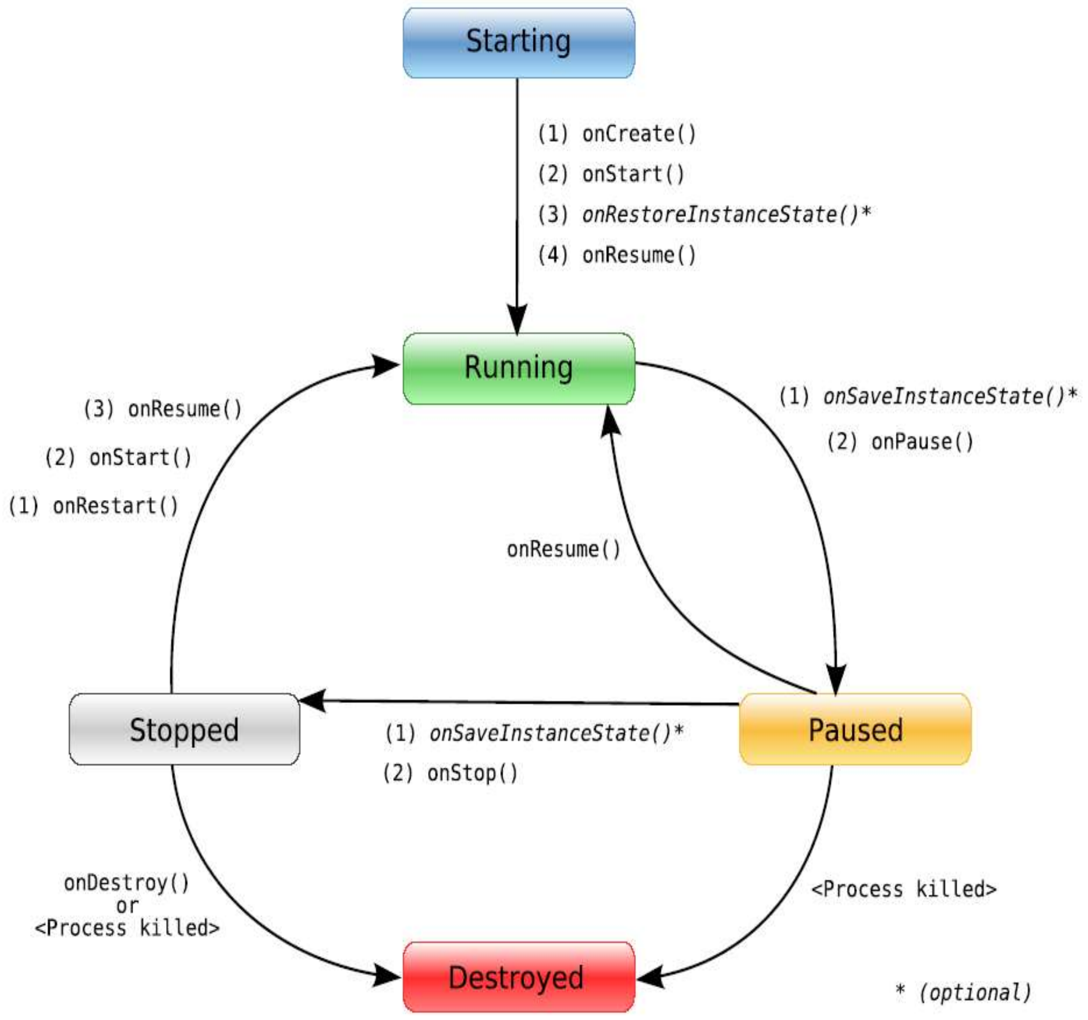
# Activity Life Cycle - STARTING

- When an activity doesn't exist in memory it is then in the starting state.
- Once the activity is started it is then in the running state.
- The transition from starting state to running state is the most expensive part of the overall process in terms of computing time & battery life.
- That's why activities are not directly destroyed if the user switches screens / apps.



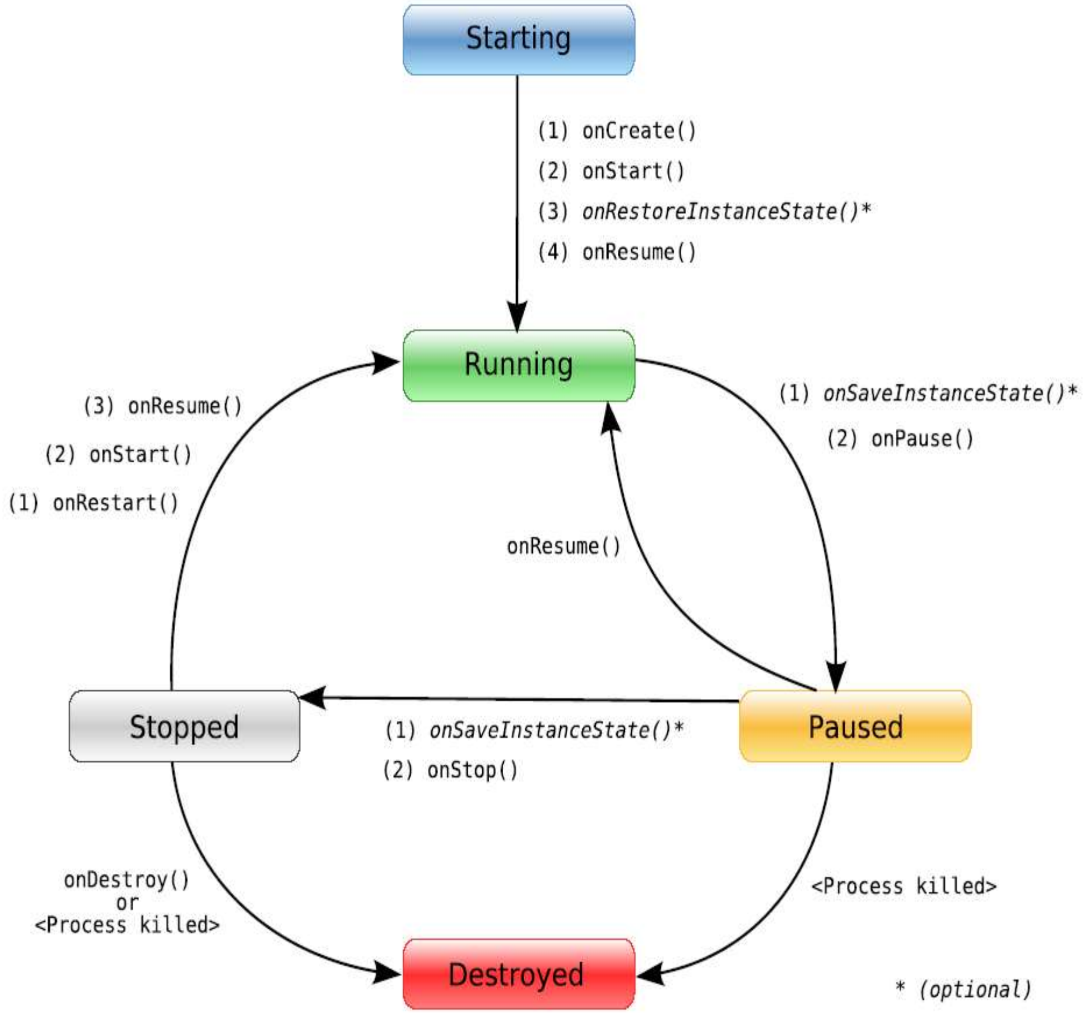
# Activity Life Cycle - RUNNING

- The activity in a running state is the one that is currently on the screen and interacting with the user.
- We also say this activity is in focus, meaning that all user interactions— such as typing, touching the screen, and clicking buttons—are handled by this one activity.
- Therefore there is only one running activity at any given time.



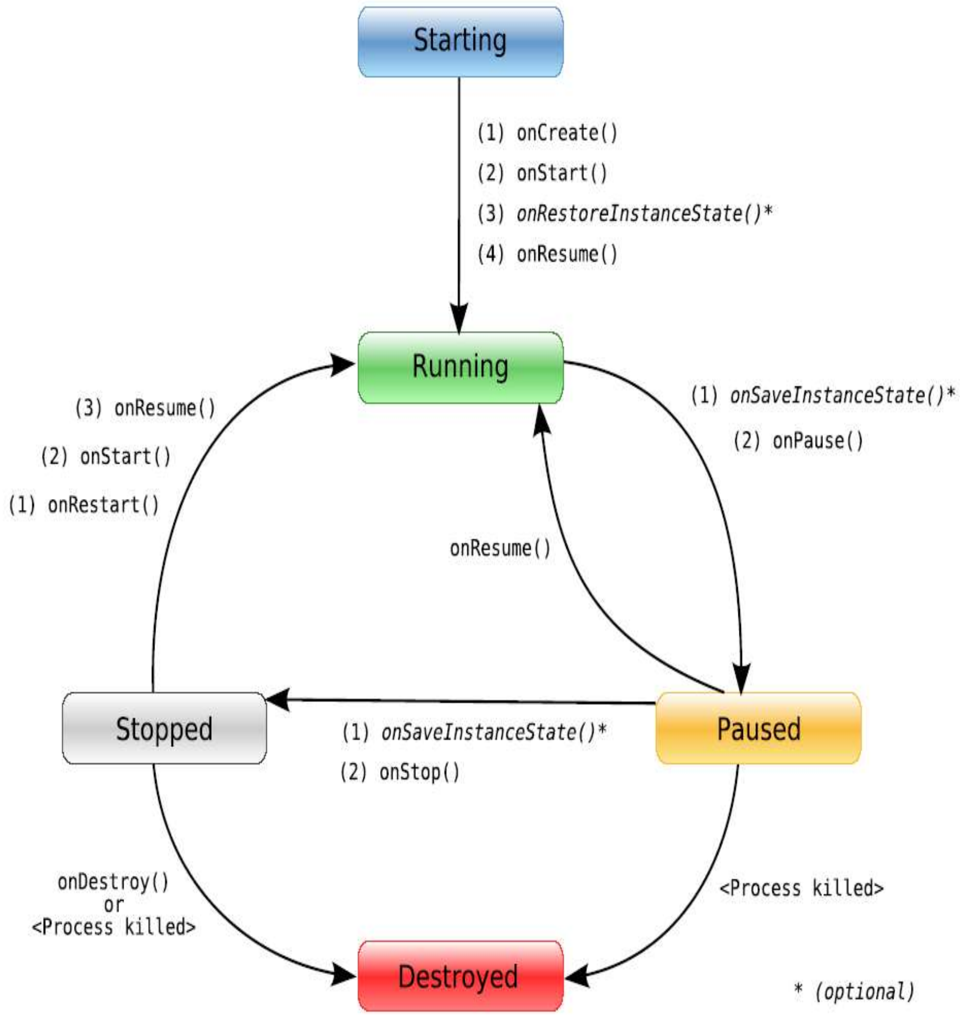
# Activity Life Cycle - PAUSING

- When an activity is not in focus (i.e., not interacting with the user) but still visible on the screen, we say it's in a paused state.
- This is not a typical scenario, because the device's screen is usually small, and an activity is either taking up the whole screen or none at all.
- We often see this case with dialog boxes that come up in front of an activity, causing it to become Paused. All activities go through a paused state en route to being stopped.



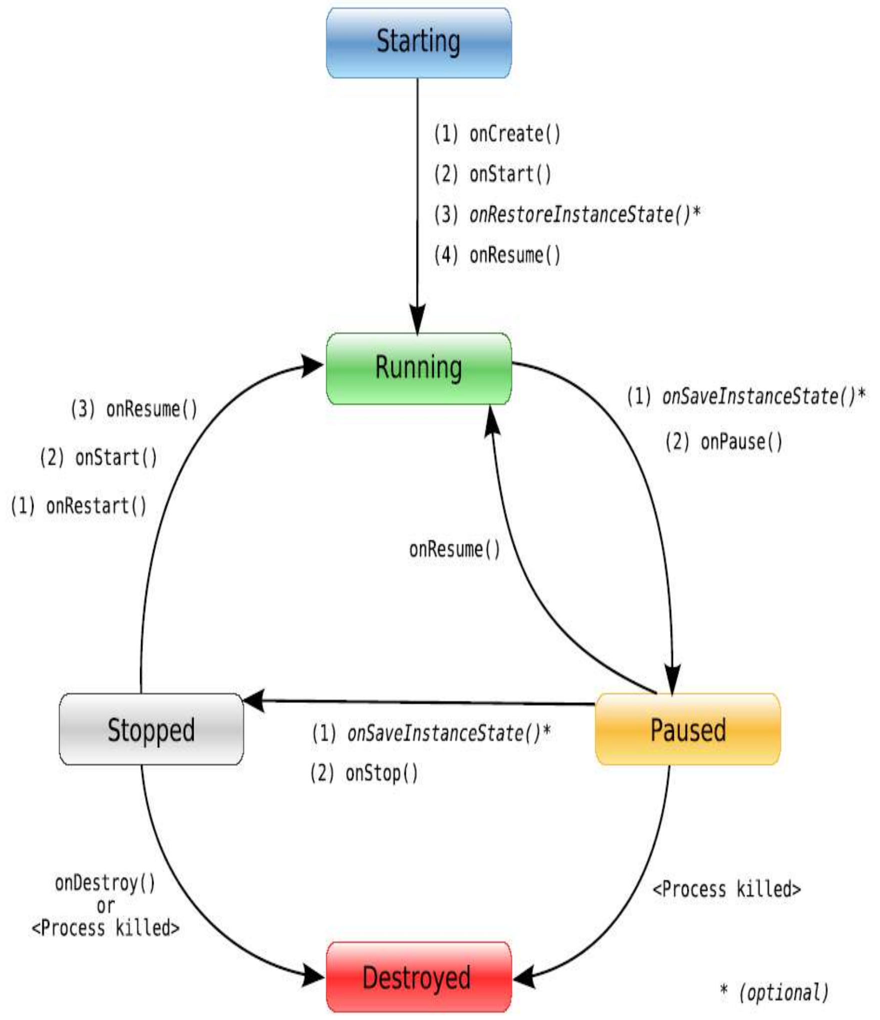
# Activity Life Cycle – Stopped

- When an activity is not visible, but still in memory, we say it's in a stopped state.
- Stopped activity could be brought back to the front to become a Running activity again.
- Or, it could be destroyed and removed from memory.
- The system keeps activities around in a stopped state because it is likely that the user will still want to get back to those activities some time soon, and restarting a stopped activity is far cheaper than starting an activity from scratch.



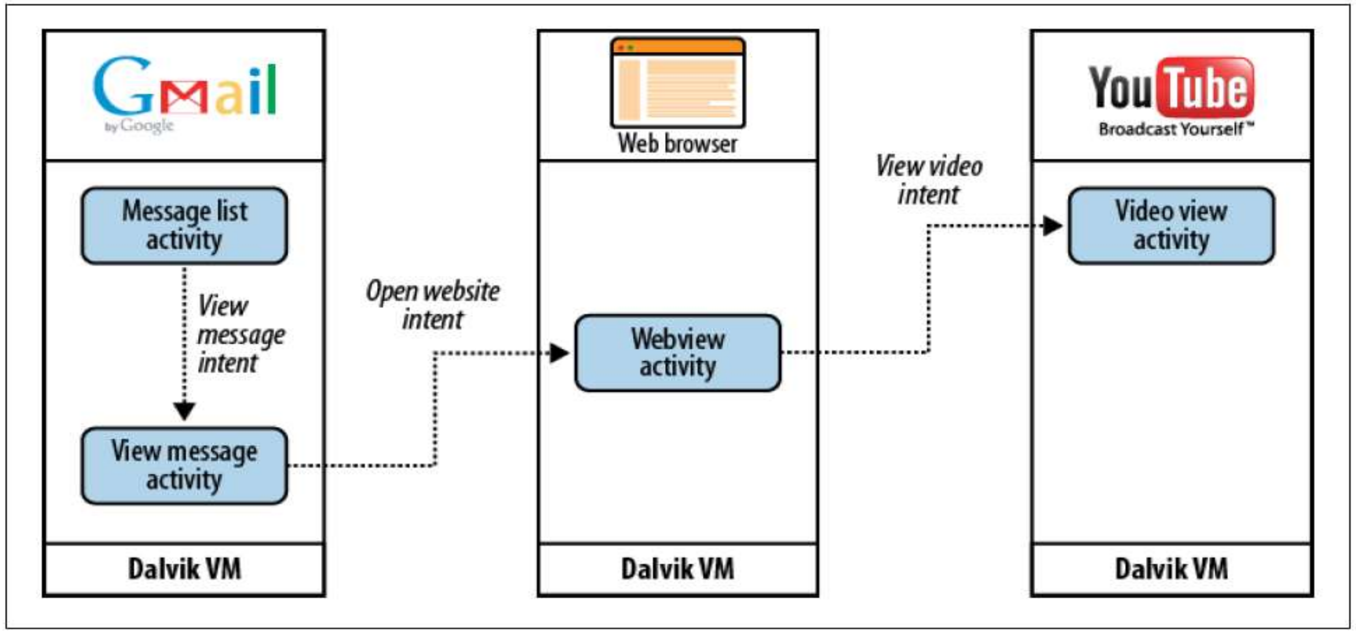
# Activity Life Cycle – Destroyed

- A destroyed activity is no longer in memory.
- The Activity Manager decided that this activity is no longer needed and has removed it.
- Before the activity is destroyed, it can perform certain actions, such as save any unsaved information.
- However, there's no guarantee that your activity will be stopped prior to being destroyed.
- It is possible for a paused activity to be destroyed as well.
- For that reason, it is better to do important work, such as saving unsaved data, en route to a paused state rather than a destroyed state.



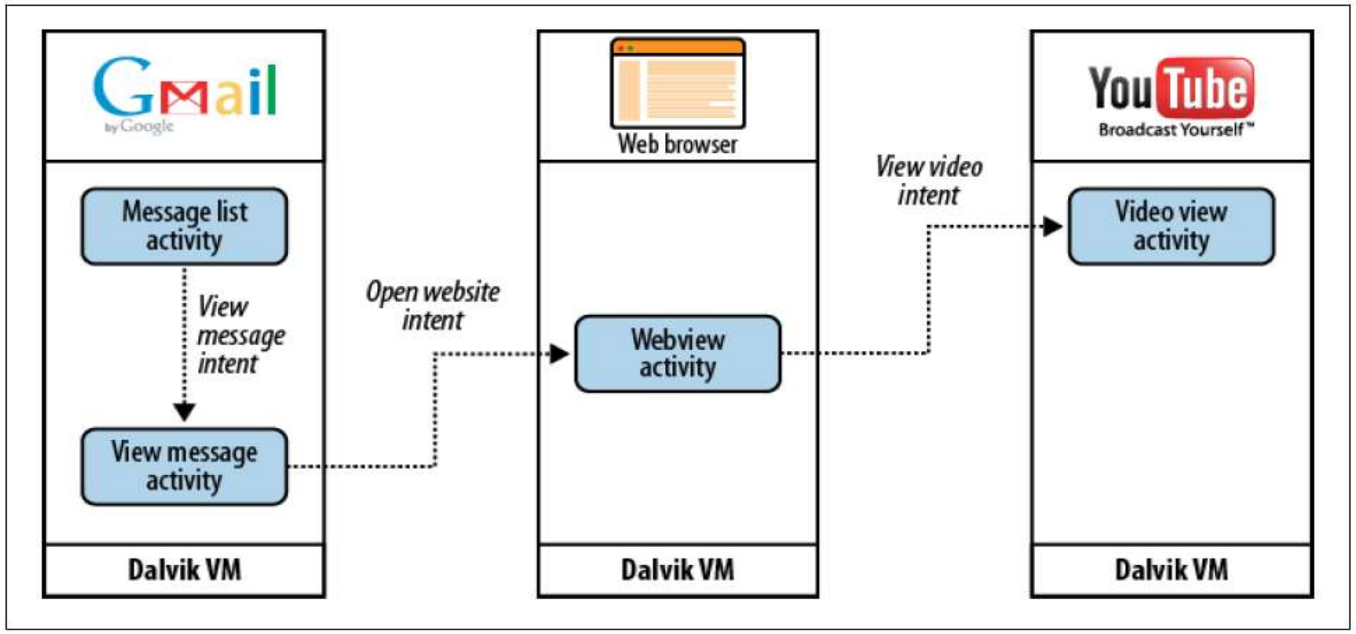
# Intents

*Intents are **messages** that are sent among the major building blocks. They **trigger** an activity to start up, tell a service to **start** or **stop**, or are simply broadcasts. Intents are **asynchronous**, meaning <u>the code that sends them doesn't have to wait for them to be completed</u>.*



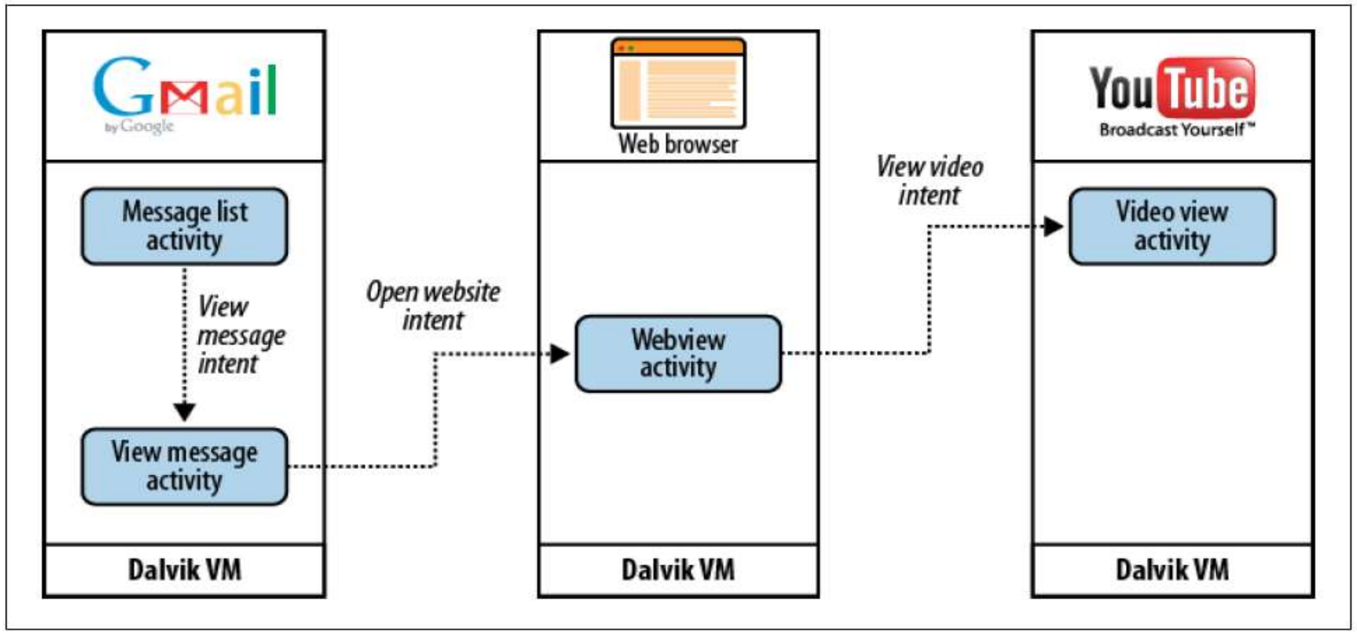
# Explicit & Implicit Intents

- In an explicit intent, the sender clearly spells out which specific component should be on the receiving end.
- In an implicit intent, the sender specifies the type of receiver.



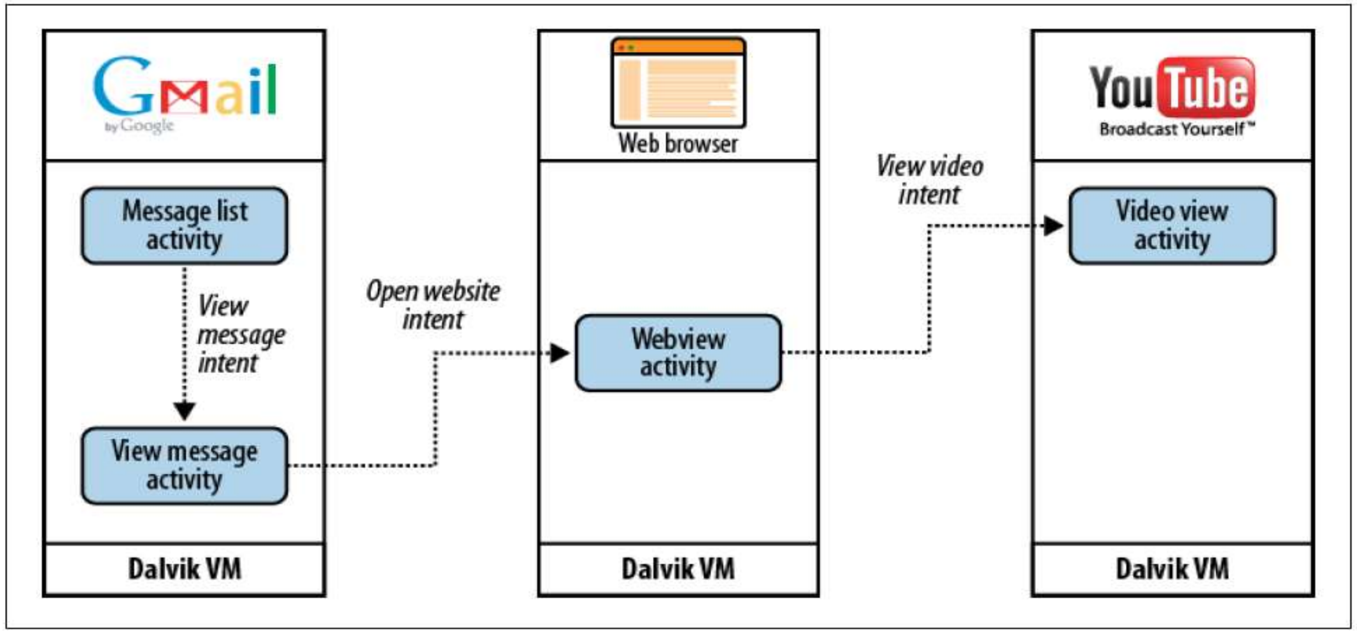
# Explicit & Implicit Intents

- For example, your activity could send an intent saying it simply wants someone to open up a web page.
- In that case, any application that is capable of opening a web page could "compete" to complete this action.
- When you have competing applications, the system will ask you which one you'd like to use to complete a given action. You can also set an app as the default.



# WHY? Explicit & Implicit Intents

- This type of messaging allows the user to replace any app on the system with a custom one. For example, you might want to download a different SMS application or another browser to replace your existing ones.

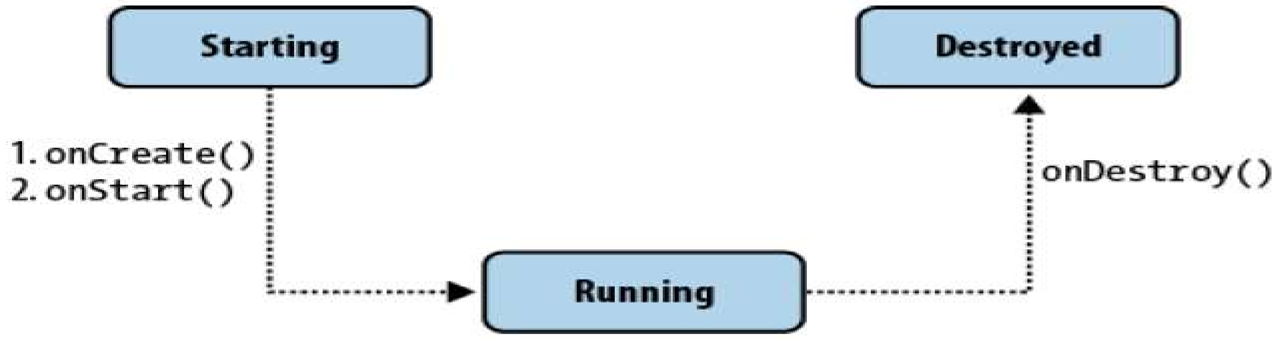


## Services

- Services run in the background and don’t have any user interface components.
- They can perform the same actions as activities, but without any user interface.
- Services are useful for actions that we want to perform for a while, regardless of what is on the screen.
- **<u>For example</u>**, you might want your music player to play music even as you are flipping between other applications.

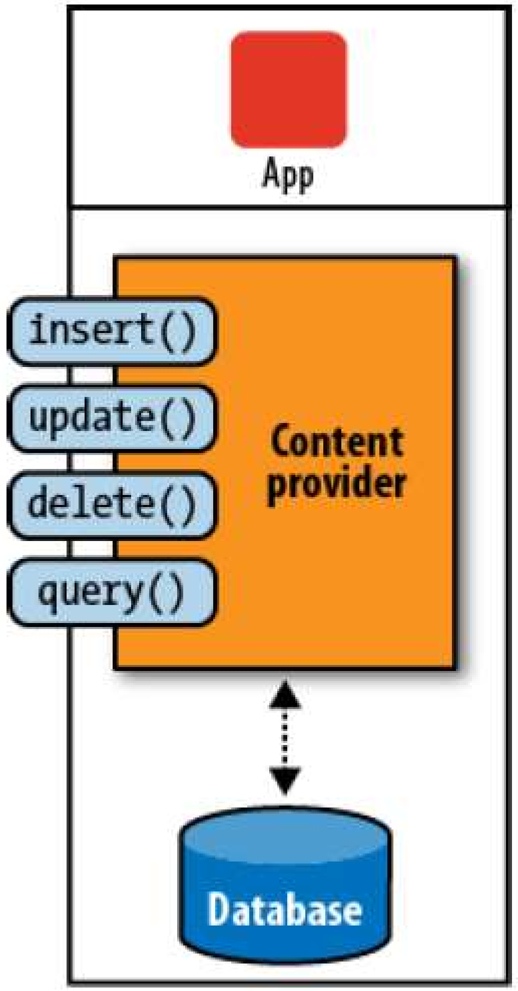
# Services Life Cycle

- You either start a service or stop it.
- Services are controlled by the developer, and not so much by the system.
- Developers have to be mindful to run services so that they don't consume shared resources unnecessarily, such as the CPU and battery.



# Content Providers

- Content providers are interfaces for **sharing data between applications.**
- By default, Android runs each application in its own sandbox so that all **data that belongs to an application is totally isolated** from other applications on the system.
- Although small amounts of data can be passed between applications via intents, content providers are much better suited for sharing persistent data between possibly large datasets.



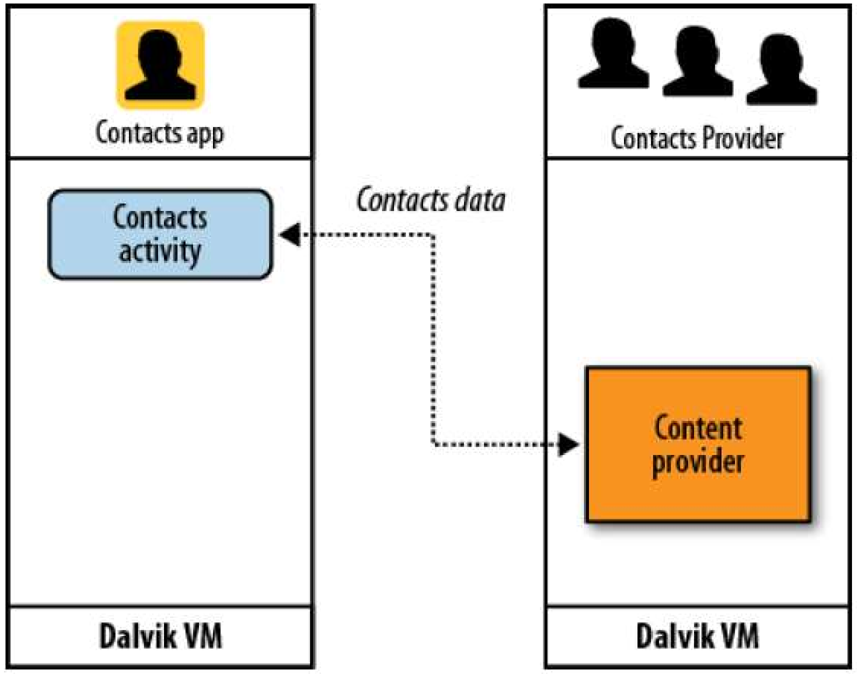
Ex. Sharing contacts among all applications

.... another example?

# Ex. Contacts App

- Contacts app uses Contacts Provider, a totally **separate application**, to retrieve data about users' contacts. The Contacts app itself doesn't have any contacts data, and Contacts Provider doesn't have any user interface.

- This separation of data storage and the actual user interface application offers **greater flexibility to mash up various parts of the system**. For example, a user could install an **alternative address book application** that uses the same data as the default Contacts app.



# Broadcast Receivers

- What happens when an SMS arrives, a call comes in, the battery runs low, or the system gets booted?
- These events get broadcasted, and any number of receivers could be triggered by them.

- What's a receiver?
- Receiver = Observer code. *The receiver is simply dormant code that gets activated once an event to which it is subscribed to happens.*

$\}$ Reduce brightness after the battery runs under 20%

## Broadcast Receivers

- *Broadcast receivers themselves* ***do not have any visual representation****, and* ***not actively running in memory****. But when triggered, they get to* ***execute some code****, such as starting an activity, a service, or something else.*

# Application Context

- All of the previous building blocks, together, they make an application, right?
- Ok, how to make an application share the data and resources between various building blocks?
- You make them live all together inside something called the Application Context.

- An application context gets **created whenever the first component of this application is started up**, regardless of whether that component is an activity, service, or something else.
- Application context **lives as long as your application is alive.**

```
You can easily
obtain a reference
to the context by
calling one these
two methods:

Context.getApplica
tionContext()

or

Activity.getApplic
ation()
```

## Summary

- Smartphone
- Android
  - Distribution
  - Version, API
  - Android OS
  - Main building blocks