Android Architecture

Android – Mobile OS

Android

- What is android? Linux 2.6 based operating system for mobile devices.
- Open source and released under Apache Licence (Carriers can modify it before distributing).
- Google acquired android in 2005.
- Android 1.0 released 2008.
- In 2012 Android 4.2 released
- Improvements include support for new devices:
  - Cameras
  - Multi core CPU
  - Barometer
  - etc

Design principles – What is required of a mobile OS?

How do these differ from the needs of a desktop system?

- Long battery life.
- Fast boot up.
- Fast response.
- Applications (Programming environment)
- Security

- Consider how the above are met by the system design.

Mobile hardware differences

- Battery
- Touch screen
- Portable (Mobile CPU)
- More limited memory
- Fewer devices

Different Components Of A Smartphone

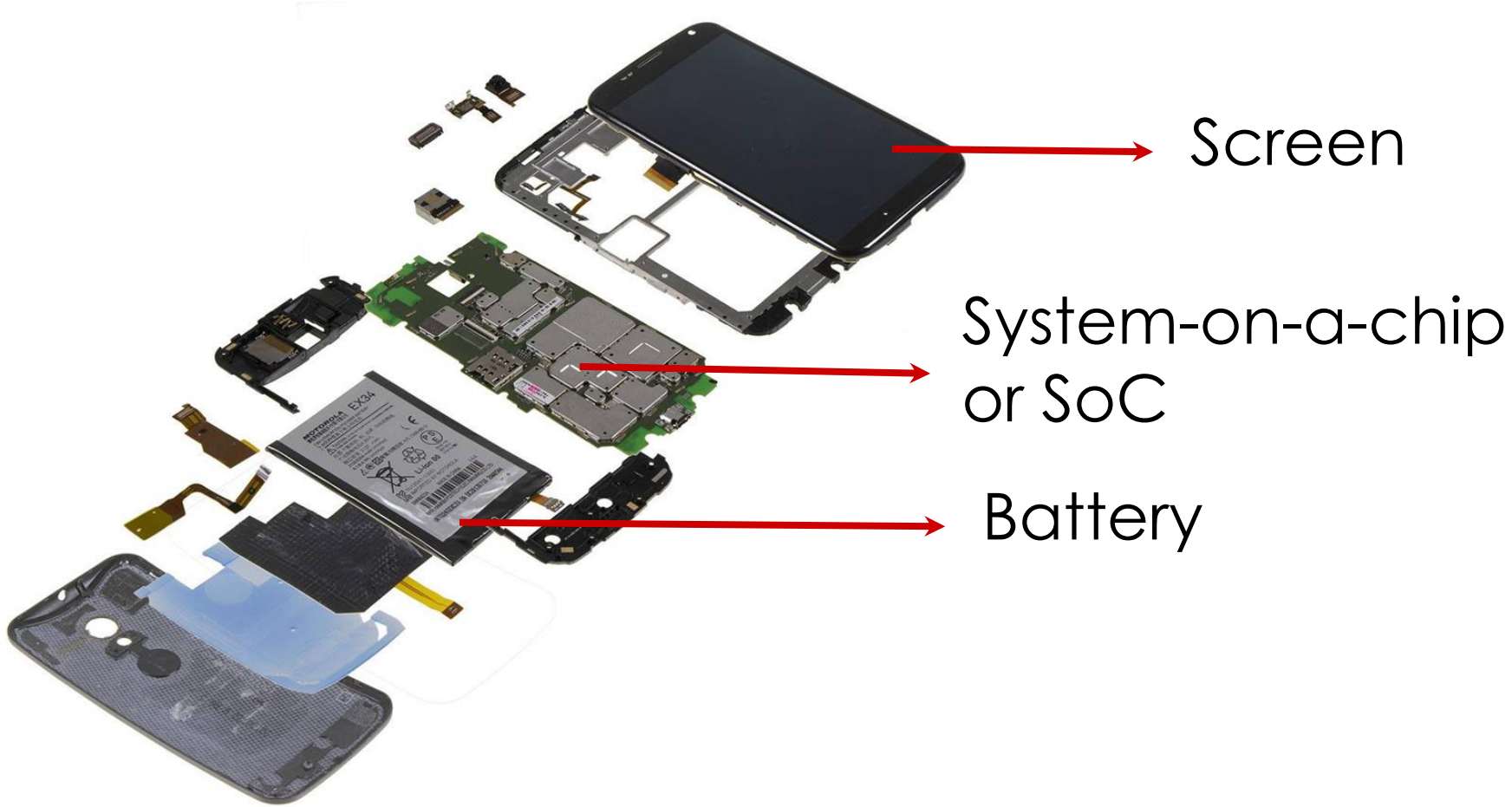


Image Source: https://fossbytes.com/whats-inside-smartphone-depth-look-parts-powering-everyday-gadget/

System-on-a-chip or SoC

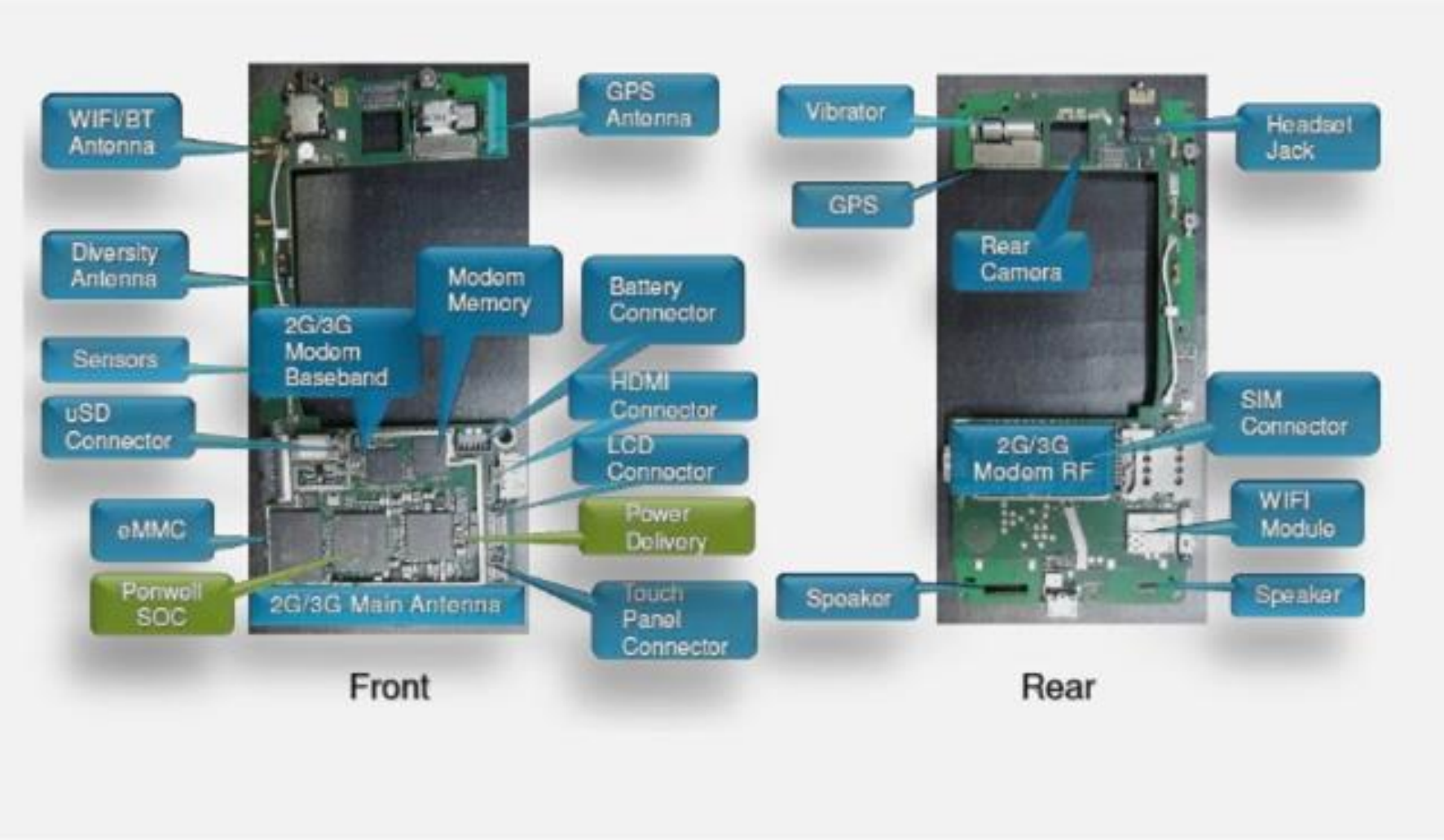
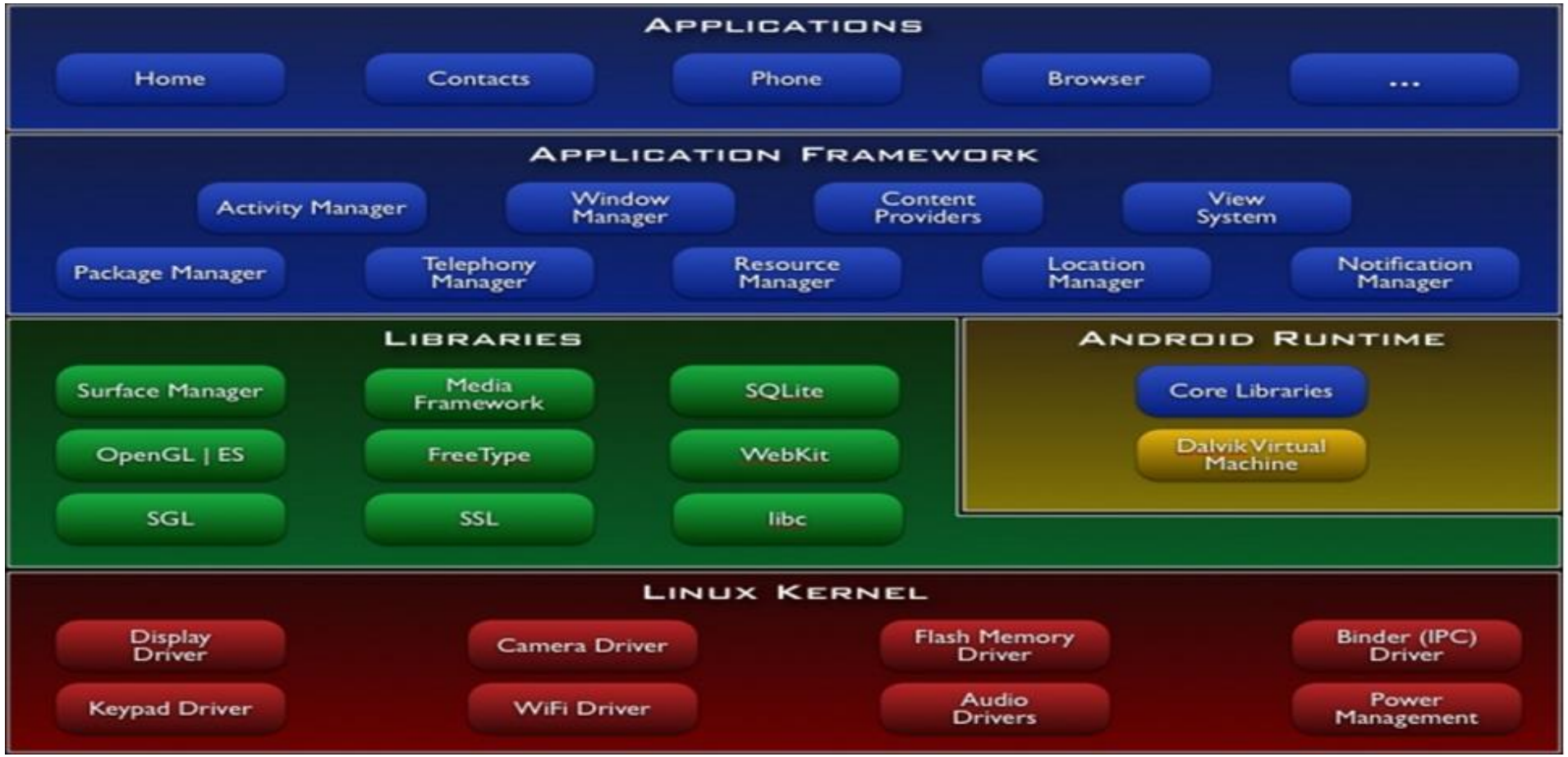


Image Source: https://www.slideshare.net/ruliandi/system-on-chip-soc-44502780

Sensors

Image Source:https://www.mdpi.com/1424-8220/19/9/2164

Android Architecture



Source: http://developer.android.com/about/versions/index.html

Android Architecture

- The architecture is based on 2.6 kernel. Android use Linux kernel as its hardware abstraction layer.
- Linux provides proven core features on which to develop the Android operating system.

Source: http://developer.android.com/about/versions/index.html

Linux Kernel Features

- **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
- It is the process of controlling and coordinating device memory by assigning suitable memory blocks to various running programs.

- **Process Management:** It manages the process well, allocates resources to processes whenever they need them.
- It refers to the device available resources and how it can be perfectly allocated to the running processes to optimize overall system performance.

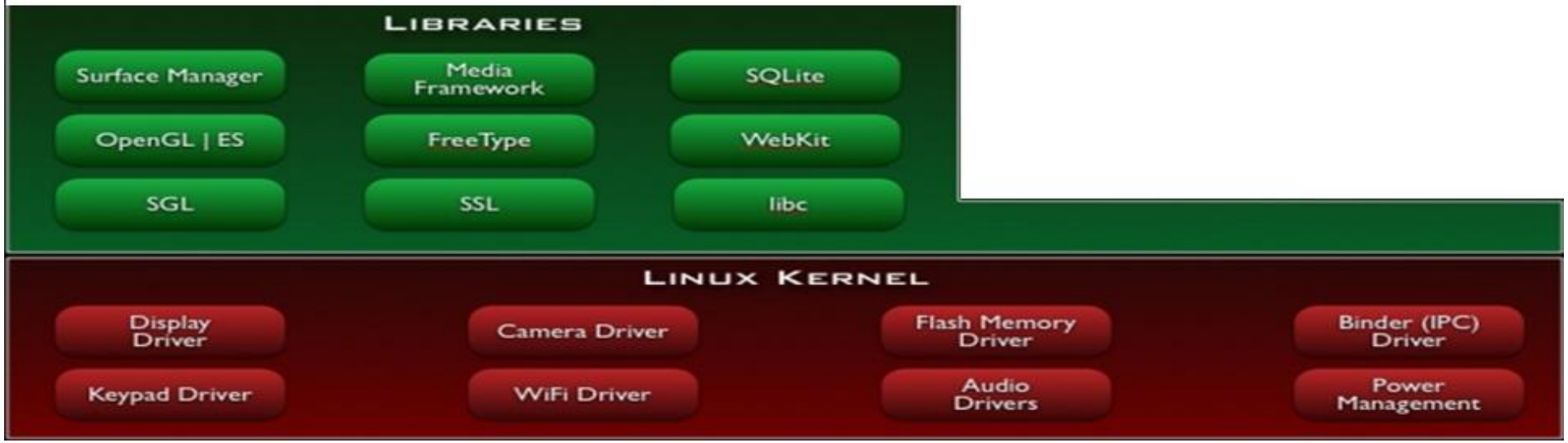
- **Network Stack:** It effectively handles the network communication.

- **Driver Model**:The main purpose of device drivers is to provide abstraction by acting as a translator between a hardware device and the operating system, helping in running atached hardware (display, WI-FI and audio).

- **Security**: The Linux kernel handles the security between the application and the system.

Android Architecture

- Native libraries consist of various C/C++ core libraries with numerous of open source tools.



Source: http://developer.android.com/about/versions/index.html

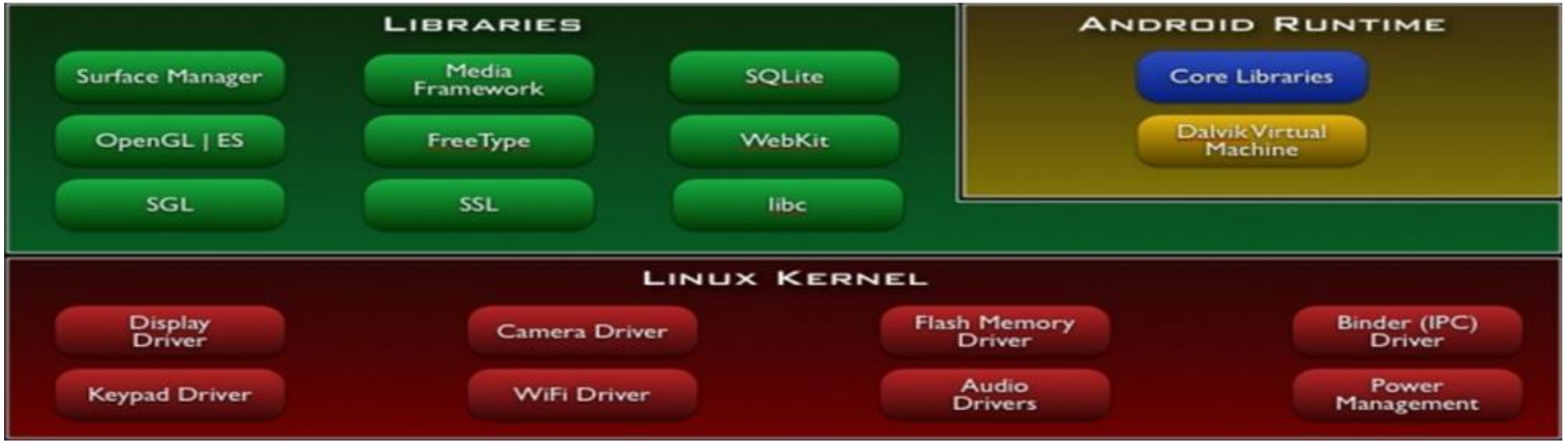
Libraries

- **Surface Manager:** It is responsible for composing different drawing surfaces onto the screen.
- **OpenGL/ES & SGL:** This cross-language, cross-platform application program interface (API) is used to produce 3D and 2D computer graphics, respectively.
- **Media frameworks:** These libraries allow you to play and record audio and video.
- **FreeType:** It is a free, high-quality and portable font engine.

Libraries

- **Secure Socket Layer (SSL):** These libraries are there for Internet security.
- **SQLite:** It uses as the core of most of the data storage.
- **WebKit:** This open source web browser engine provides all the functionality to display web content and to simplify page loading.

Android Architecture



Source: http://developer.android.com/about/versions/index.html

Core Libraries

- Mainly consists of 3 parts:
  - JAVA Libraries
  - Android Libraries
  - DVM Libraries
- Standard JAVA libraries is a must alongside specific libraries for Android to control and access media, databases etc.

Core Libraries

- DVM Libraries are specifically used to interact with the Dalvik Virtual Machine

Android Runtime

- Before **Android 4.4** Google used **Dalvik Runtime**
- Recently, it uses **Android Runtime (ART)**
- Dalvik is based on **JIT** (**just in time**) compilation (source->.dex->m/c code).
- Dalvik ensured smaller memory footprint and uses less physical space on the device.
- ART, on the other hand, compiles the intermediate language, Dalvik bytecode.
- **ART->Pre compilation**

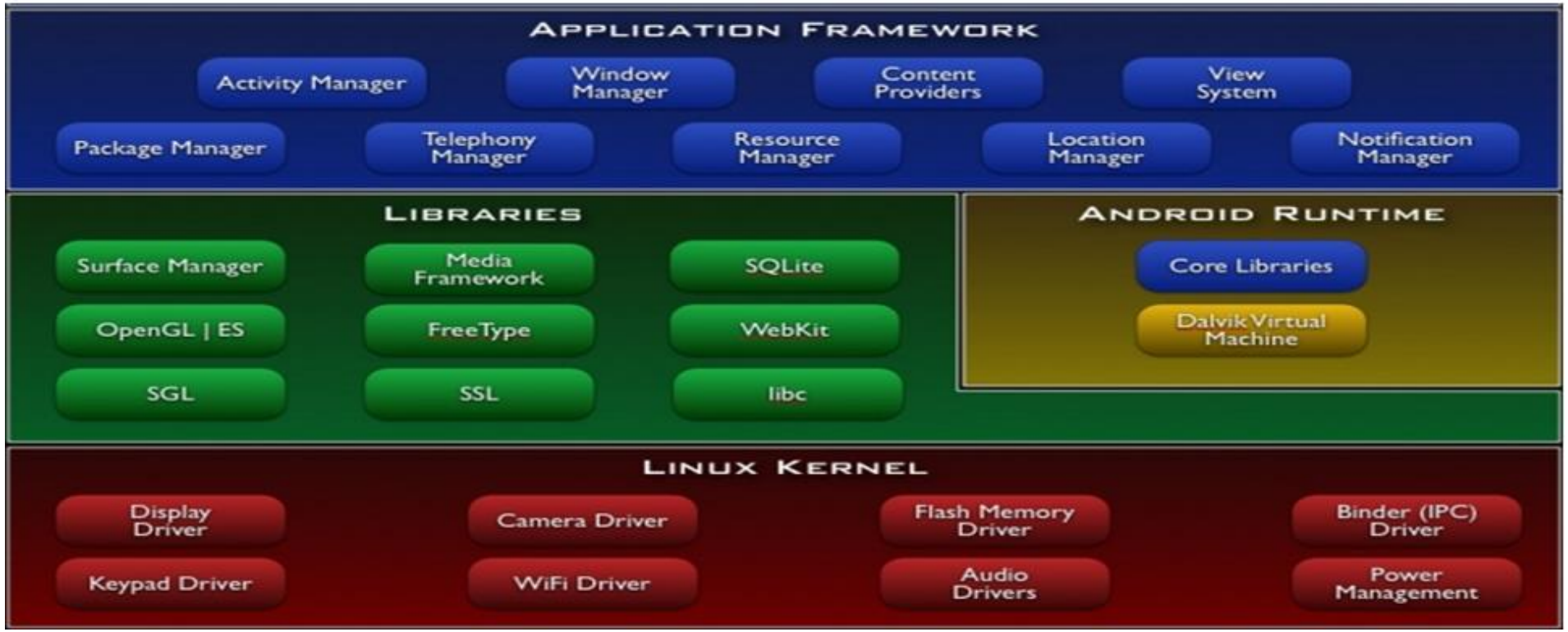
Android Runtime

- ART, on the other hand, compiles the intermediate language, Dalvik bytecode.
- The output of ART is a system-dependent binary
- With no need for JIT compilation executes much faster
- Also ensures less CPU usage resulting in less battery drainage.

Android Runtime

- ART has disadvantages like:
  - Larger memory footprint
  - Longer installation process for the apps

Android Architecture



Source: http://developer.android.com/about/versions/index.html

Application Framework

- The Android team has built on a known set proven libraries, built in the background.
- All of it these is exposed through Android interfaces.
- **These interfaces warp up all the various libraries and make them useful for the Developer**.

Bunch of classes and methods

Application Framework

- **Activity Manager:** It manages the activity lifecycle and the activity stack.
- **Telephony Manager:** It provides access to telephony services as related subscriber information, such as phone numbers. For example: One would require telephony manager to get information like sim serial number, telephone system sort, IMEI etc.

Application Framework

- **Location Manager:** It finds the device's geographic location.
- **Notification Manager:** Android allows to put notification into the titlebar o your application. The user can expand the notification bar and by selecting the notification the user can trigger another activity.
- Class to notify the user of events that happen. Alerting the user by flashing the backlight, playing a sound, or vibrating, LED blink etc.
- **Package Manager:** Class for retrieving various kinds of information related to the application packages that are currently installed on the device.

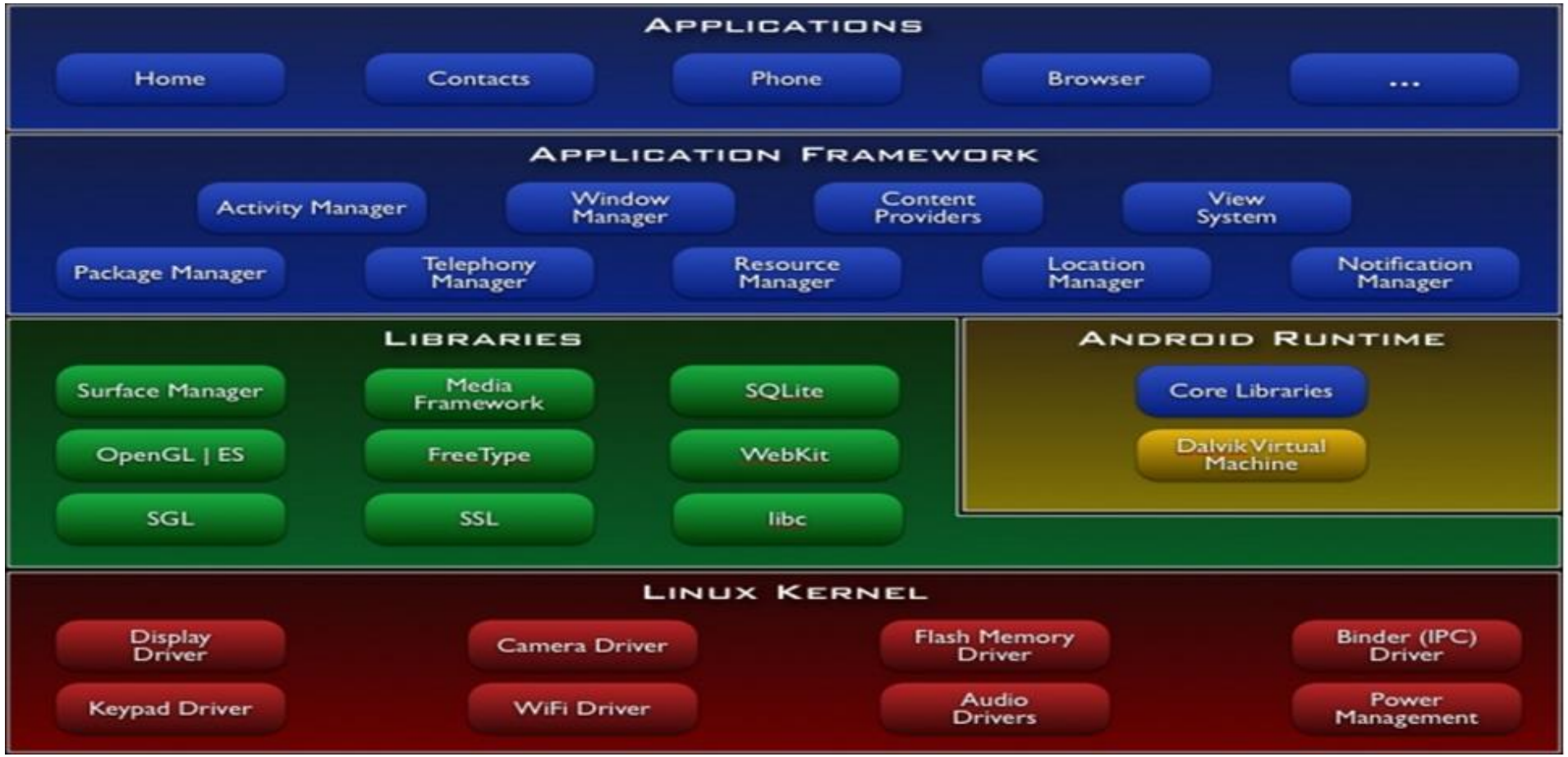
Application Framework

- **View System:** It builds the user interface by handling the views and layouts.
- **Resource Manager:** Provides a tool to visualize the drawables, colors, and layouts across your app project in a consolidated view.
- **Window Manager:** a system service, which is responsible for managing the z-ordered list of windows, which windows are visible, and how they are laid out on screen.

Application Framework

- **Content Providers:** Manages access to a central repository of data.

Android Architecture



Source: http://developer.android.com/about/versions/index.html

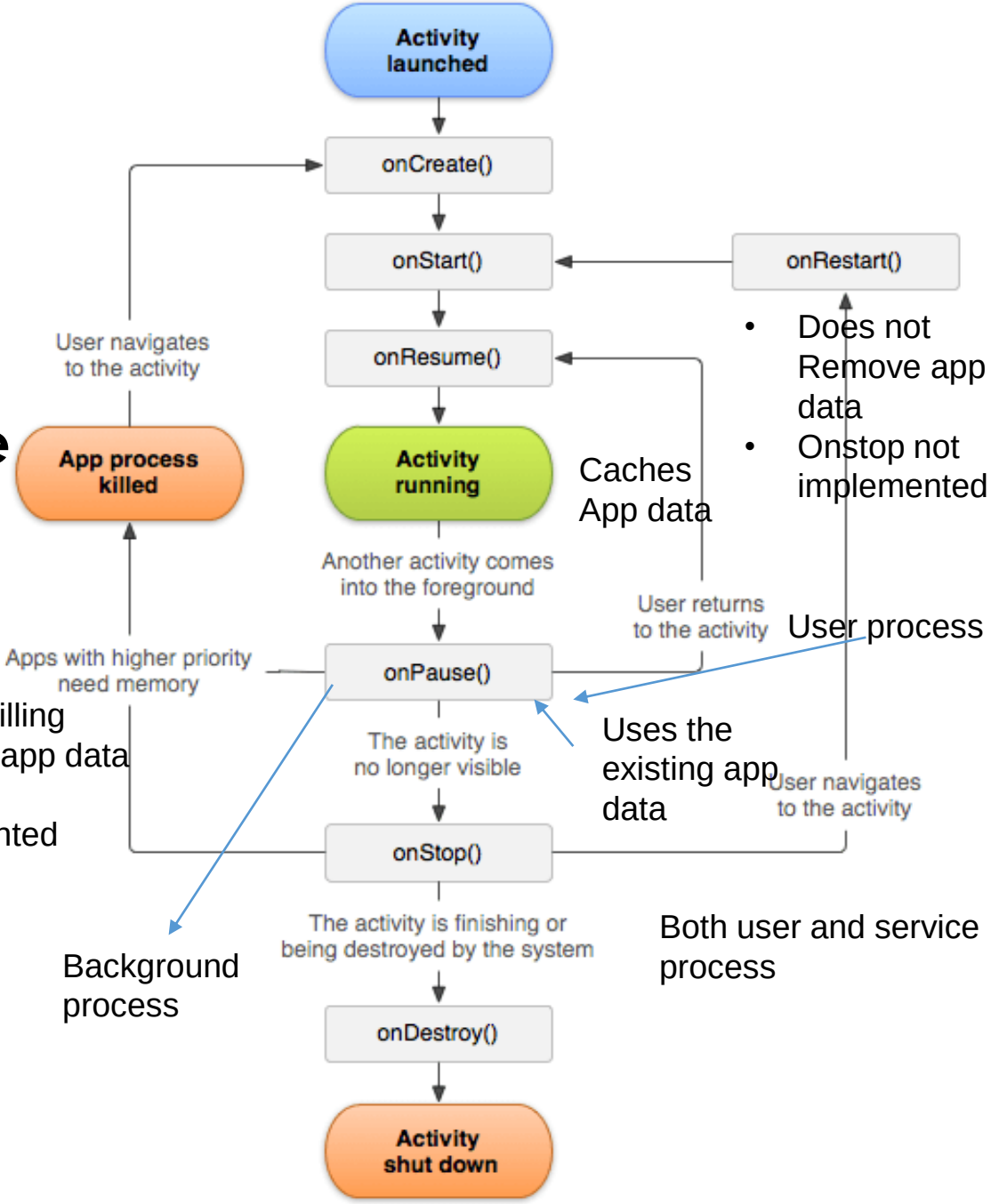
Application

- **Topmost layer** of the architecture
- Applications are installed in this layer
- All the standard applications, games, messages, contacts etc. constitute this layer

Application Lifecycle

- **Methods of Activity class**
- **Defines how the activity will behave**
- **7 Methods are:**
  - **onCreate()**
  - **onStart()**
  - **onResume()**
  - **onPause()**
  - **onStop()**
  - **onRestart()**
  - **onDestroy()**

- Foreground process--High priority user process Browsers, Whatsapp etc
- Service process (music, sensor sample)



Application life cycle

Method	Purpose
onCreate()	Is called when the activity is first created. In **many cases** this serves as the entry point.
onStart()	Is called when the activity is becoming visible to the user
onResume()	Is called once the user starts interacting with the app
onPause()	Is called when the app is not in visibility
onStop()	Is called when the app is removed from visibility
onRestart()	Is specifically called to start the app once stopped
onDestroy()	Once to be destroyed by the system or is finishing the lifecycle

AppCompatActivity

```
public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
    }
}
```

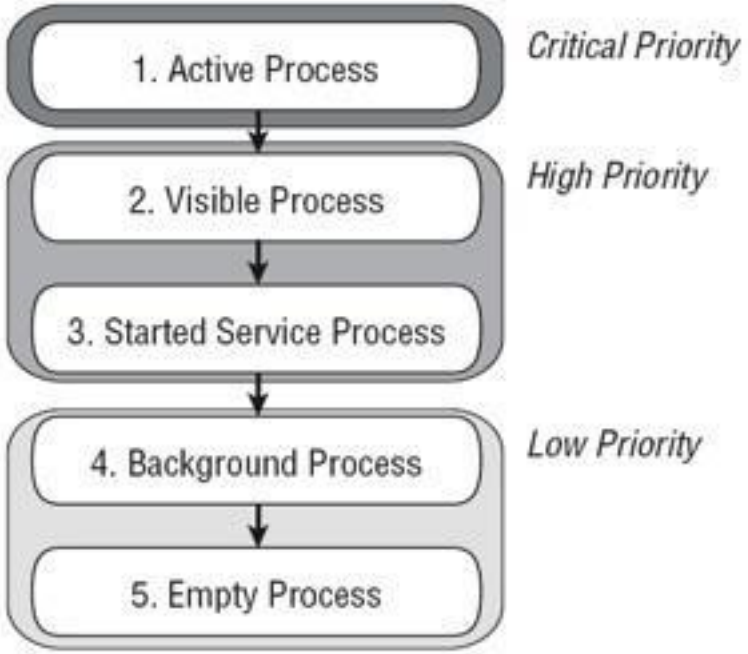
Image Source: https://www.javatpoint.com/android-life-cycle-of-activity

Process Management

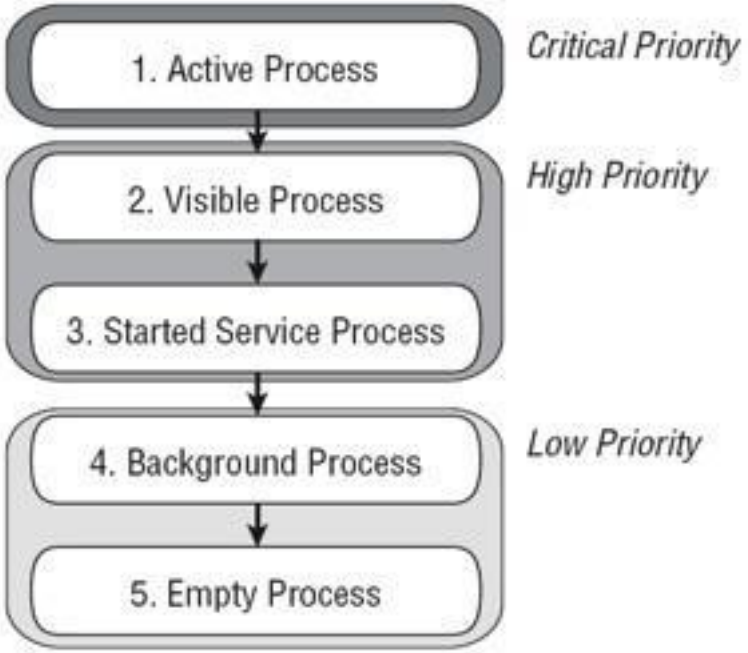
- Differences between mobile app cycle and desktop app cycle?
  - Key principles:
    - Android does not usually kill an app, they keep running even after you switch, but saves state
    - Task killers?
    - Android kills apps when memory usage too high.
      - But saves it's state for quick restart.
- Uses Linux's time sliced scheduling policy based on priority

Process Priorities

- Split into background and foreground.
- **Foreground Processes:** The process currently being interacted with
  - Has a critical priority
  - Usually limited in count at any given time

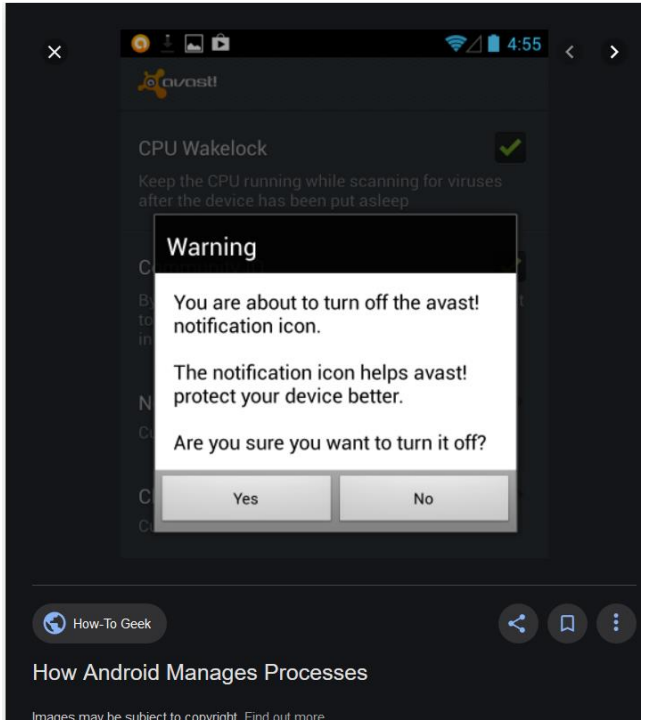


Process Priorities



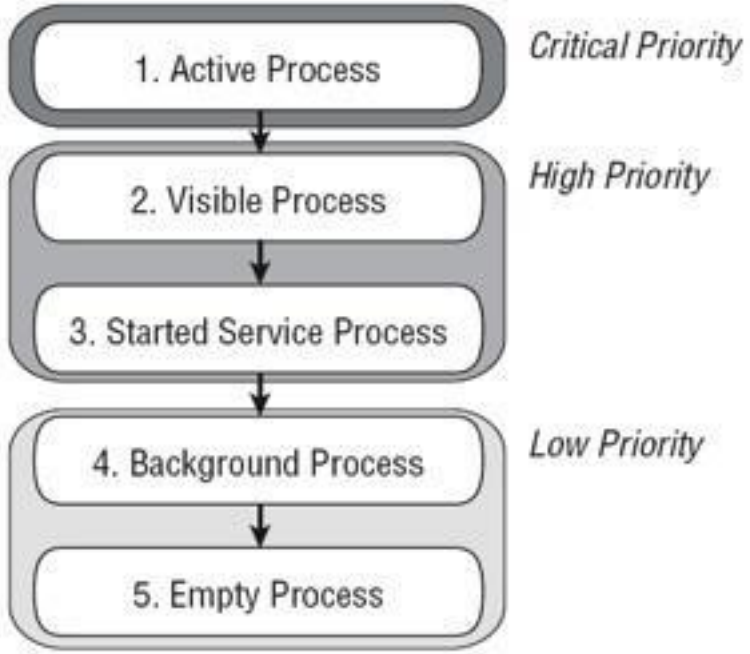
- **Visible Processes:** App not in foreground, yet affecting the visibility
  - An app visible in background of a dialog in foreground
  - This happens when an Activity is only partially obscured (by a non-full-screen or transparent Activity).
  - There are generally very few visible processes,
  - they'll be killed only under extreme circumstances to allow active processes to continue.

Still foreground process



Process Priorities

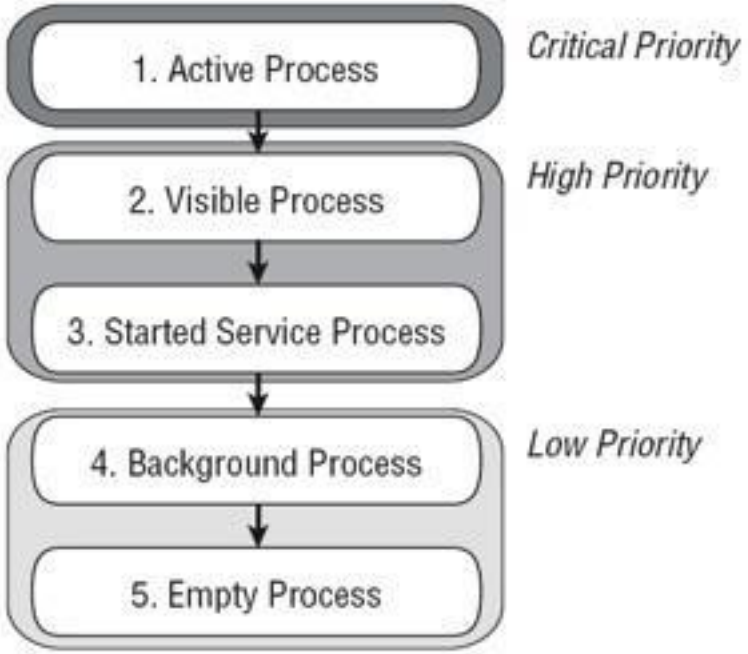
- **Service Processes:** Is not tied to any app that is visible
  - Example can be a music being played when you are using some other app
  - Never killed by Android process until absolutely necessary



Background process with special status

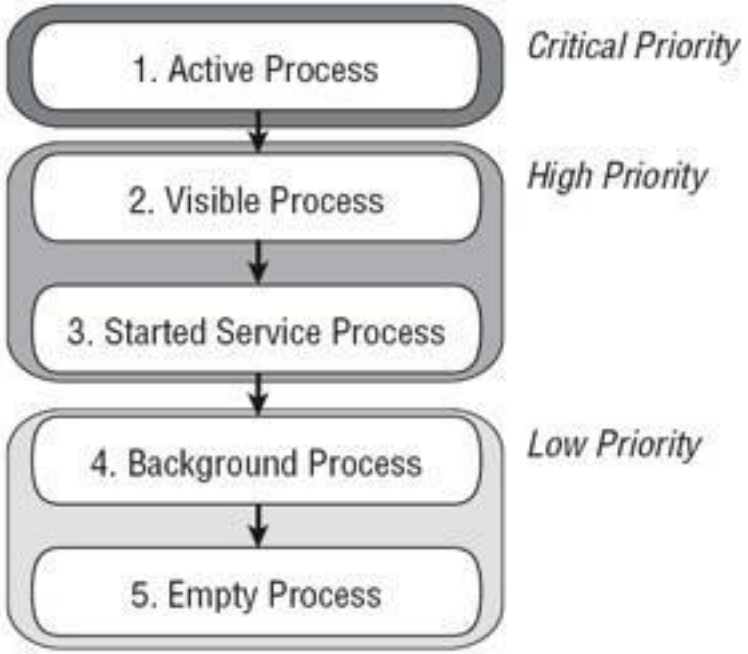
Process Priorities

- **Background Processes:** Neither visible nor affecting the user experience
  - Low priority
  - Many are running at any given time
  - Usually the **paused** apps
  - Do not consume CPU or any other non-memory resource
  - There will generally be a large number of background processes that Android will kill using a **last-seen-first-killed** pattern in order to obtain resources for foreground processes.



Process Priorities

- **Empty Processes:** With no valid app data
  - Least priority
  - To improve overall system performance, Android will often retain an application in memory after it has reached the end of its lifetime.
  - Android maintains this cache to improve the start-up time of applications when they're relaunched.
  - These processes are routinely killed, as required.



Example 1

- Let’s say you turn on your phone and open a music app.
- While you use it, the music app will be a foreground process.
- When you start playing music and leave the music app, the music will continue playing as a service process.

Example 2

- Let’s look at Angry Birds as another example.
- Angry Birds would be a foreground process while you were playing it.
- When you switch Angry Birds and enter the Gmail app to view your email, Angry Birds becomes a background process (because it doesn’t have to do anything), while Gmail becomes the foreground process.
- When you switch back to Angry Birds, it will become your foreground process and the game will resume quickly.
- Angry Birds wasn’t using resources in the background — aside from some RAM — but it resumes quickly because it remained cached and ready to resume.
- ----------
- When you swype out Angry bird and enter gmail->

Angry bird-> empty process

Manages Processes

- Android does a good job of automatically managing these processes, which is why you don't need a task killer on Android.
- When Android needs more system resources, it will start killing the least important processes first. Android will start to kill empty and background processes to free up memory if you're running low.
- If you need more memory — for example, if you're playing a particularly demanding game on a device without much RAM, Android will then start to kill service processes,
  - your streaming music and file downloads may stop.

Android Apps Can Start in Response to Events

- Android apps can also start in response to events.
- For example, a developer could program their app to automatically run at startup and run a service in the background.
- Apps can start in response to a variety of other events, such as when you take a picture, when your data connection changes, and so on

Memory management

- Android uses its own virtual machine to manage application memory.
- Dalvik VM allows multiple instances of VM to be running efficiently in parallel.
- Dalvik VM uses .dex files for running

Disk I/O

	Flash	Hard Disk Drives
Random Access	~0.1ms	5-10ms
File fragment impact	No	Greatly impacted
Total power	½ to 1/3 of HDD	Up to 15+ watts
Reliability	Reliable	Less reliable due to mechanical parts
Write longevity	Limited number of writes.	Less of a problem
Capacity	<=512GB	Up to 4TB
Price	$1.5 - 2.0 GB	$0.10 - 0.20 GB

Memory Hierarchy

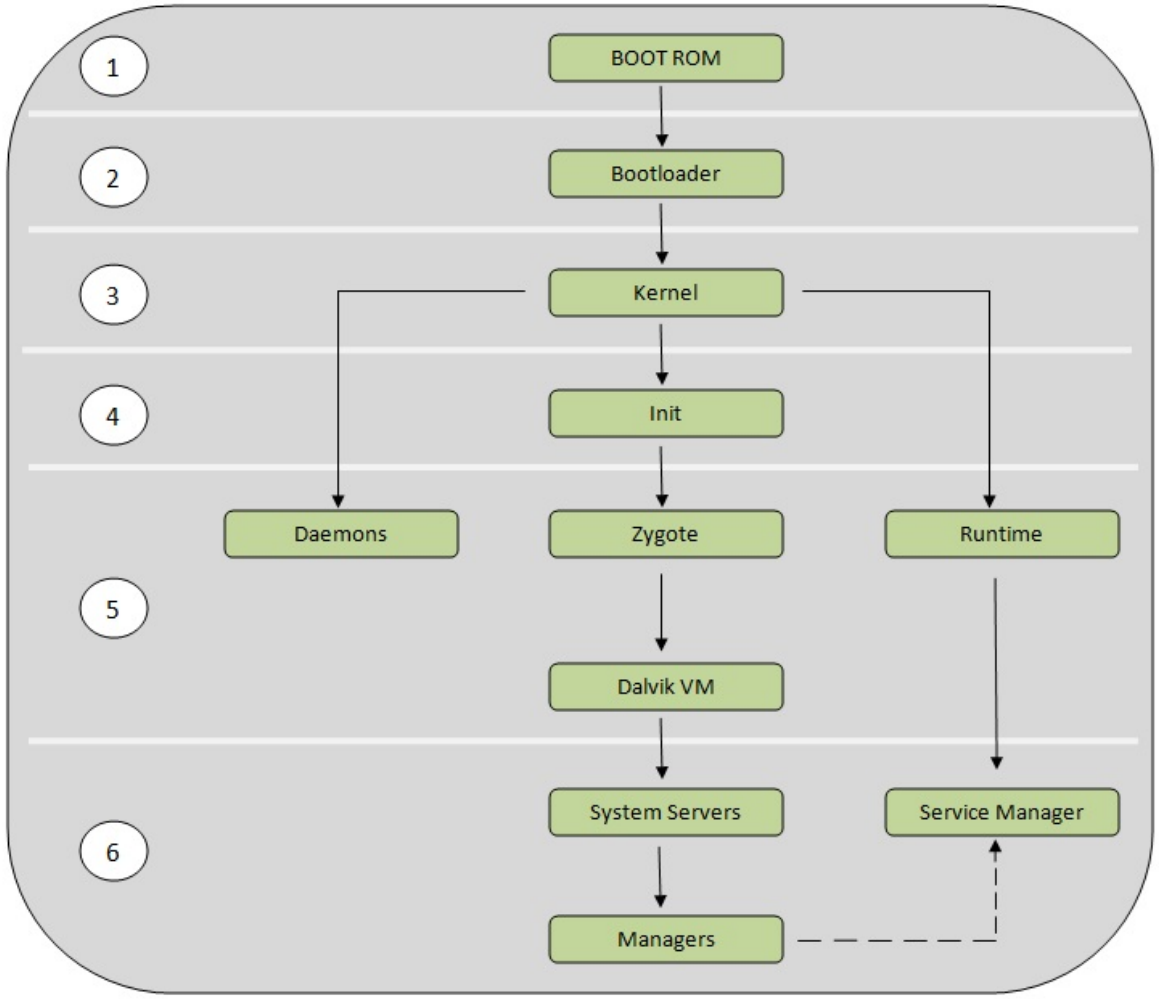
- Primary Memory: RAM (usually ranging from 2GB to 8GB)
- Flash Memory (Internal Storage): (Usually ranging from 16 to 128GB)
  - Technologically advanced versions of EEPROMs
  - Faster than traditional microSD cards (used as external storage)

Android File System

- Supports multiple different file systems (based on Linux Kernel).
- Usually yaffs2/vfat/ext4, depending on device manufacturer.
- Partitions:
  - /boot (Included android kernel)
  - /system (Android GUI and pre-installed applications). – Read only
  - /recovery (Backup)
  - /data (User data)
  - /cache (Frequently accessed data)
  - /misc (Contains misc system settings in form of on/off switches)
  - /sdcard (SD card)

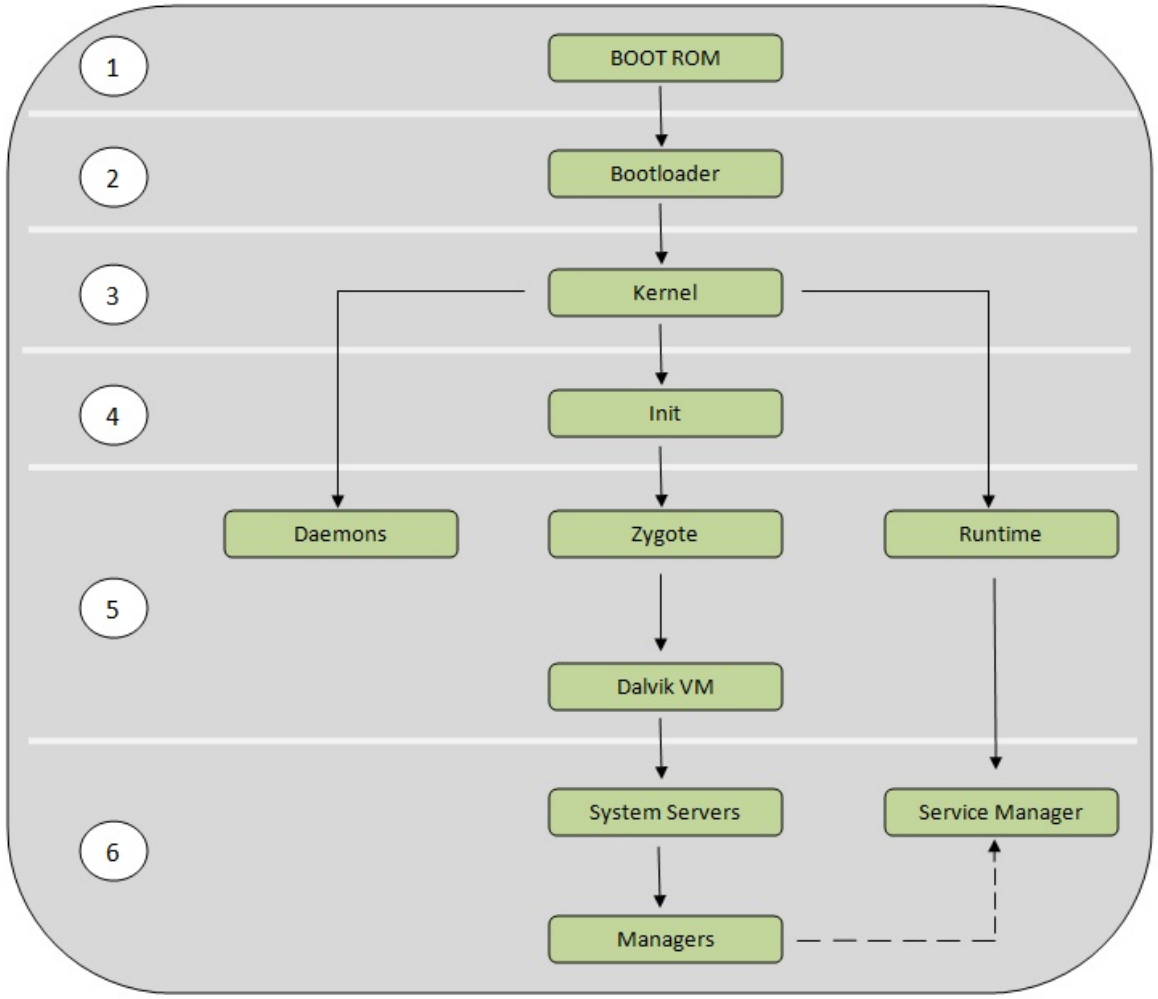
Boot-Sequence

- **Boot ROM:** When we press the power button, the Boot ROM (EEPROM) code starts executing from a predefined location which is hardwired in ROM. It loads the Bootloader into RAM and starts executing.



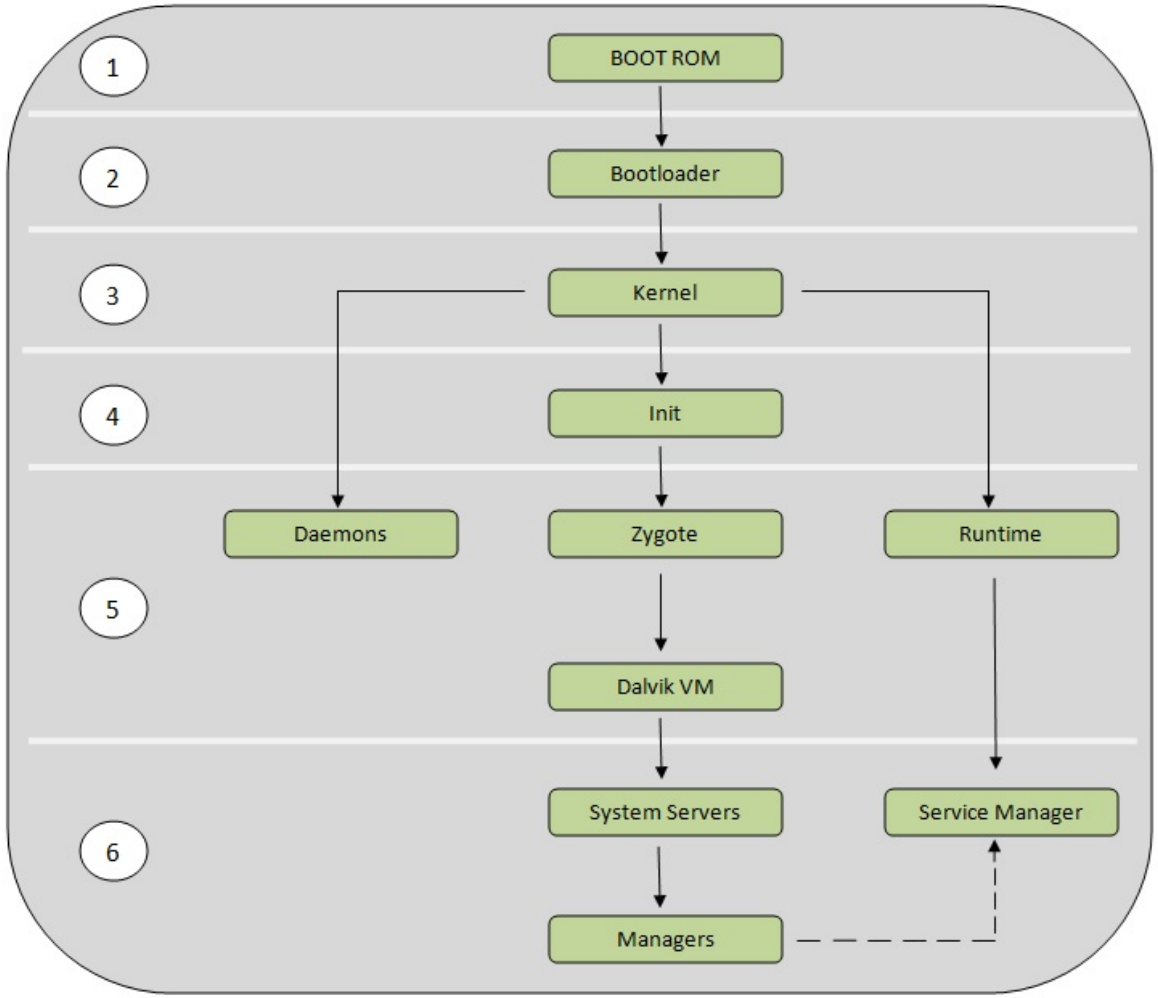
Boot-Sequence

- **Bootloader:** The bootloader is a small program which runs before Android does. This is NOT part of the Android operating system. The bootloader is the place where manufacturer puts their locks and restrictions.



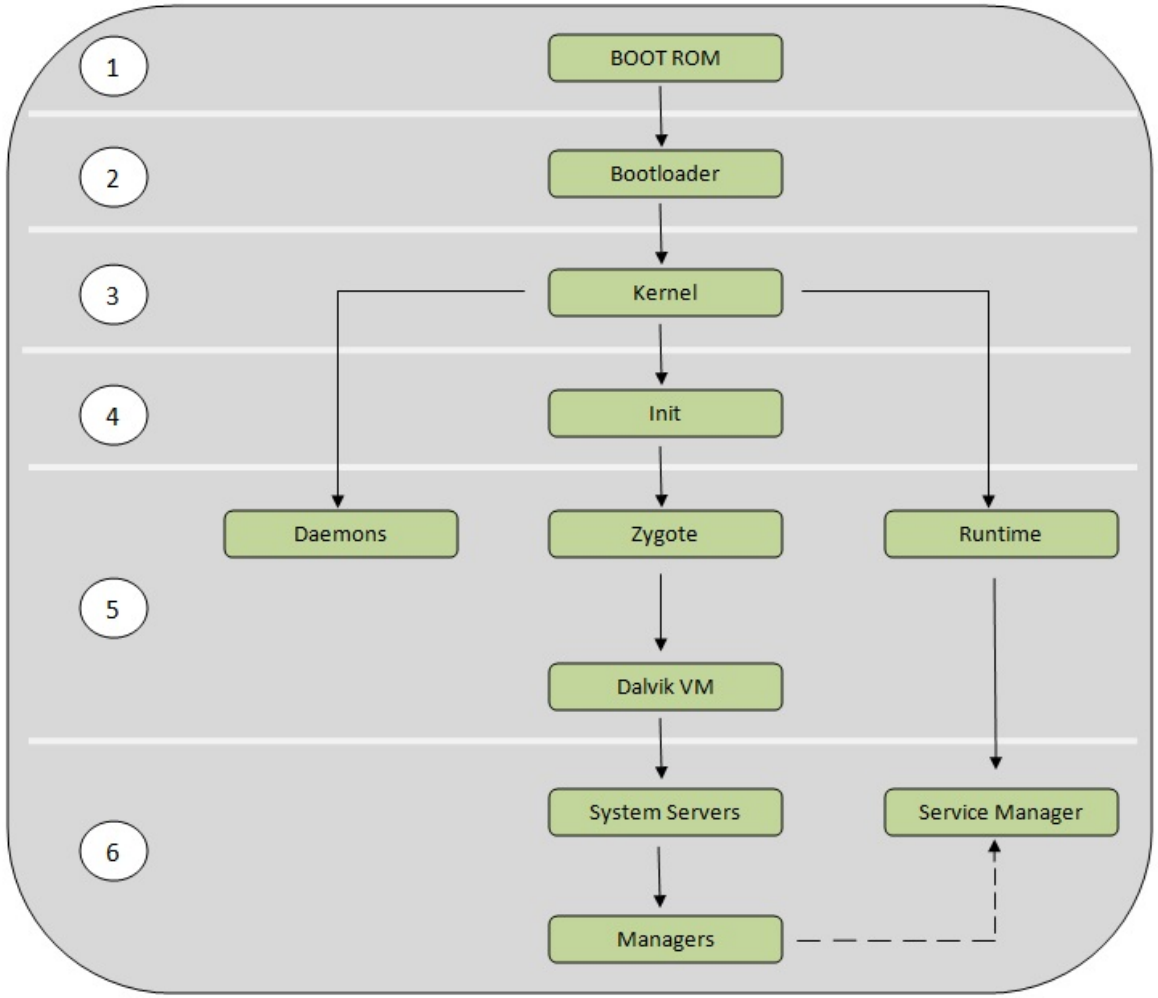
Boot-Sequence

- **Bootloader:** Bootloader has two stages:
  - First stage it detects RAM
  - In the second stage, the bootloader setups the network, memory, etc, which requires to run kernel



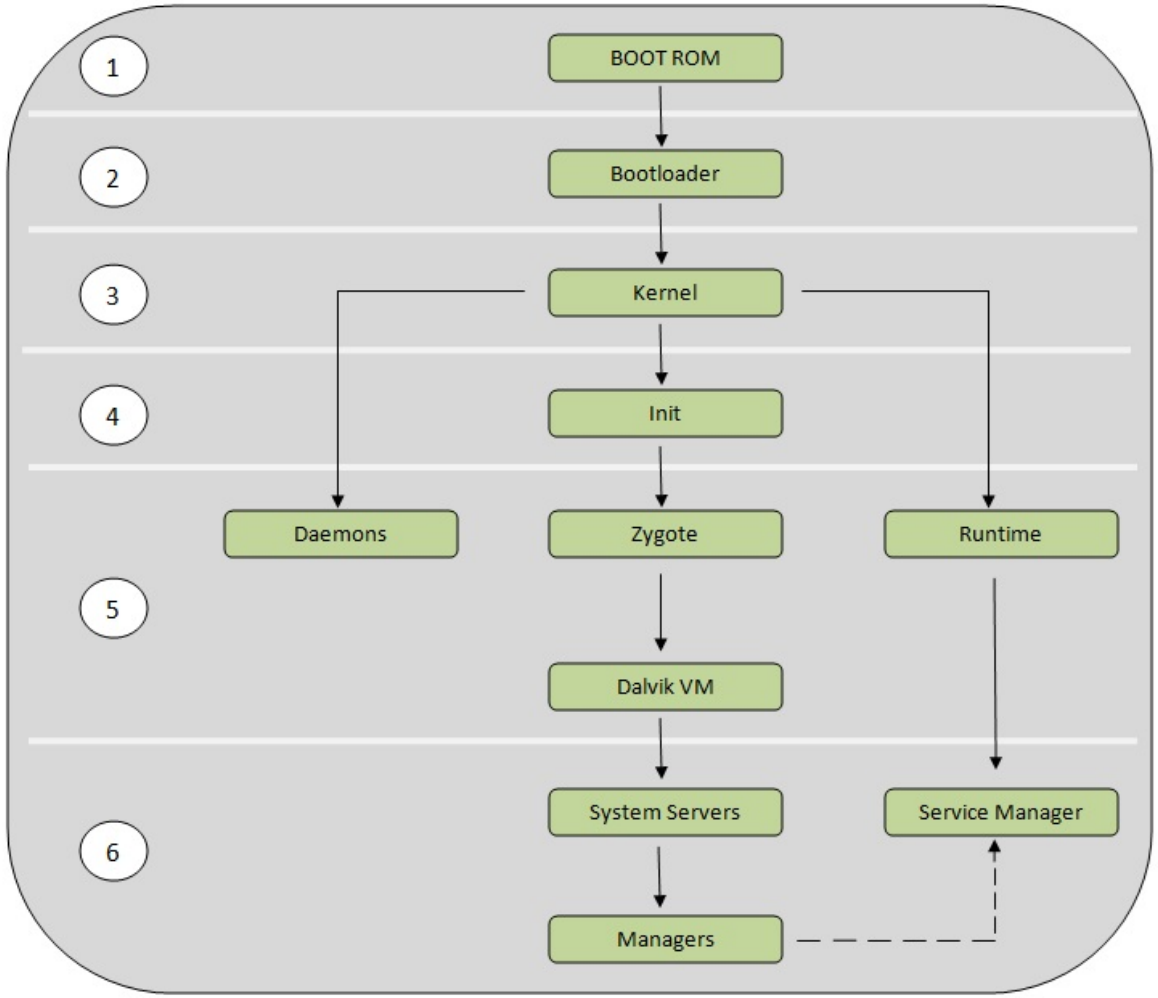
Boot-Sequence

- **Kernel:** As the kernel launches, is starts to setup cache, protected memory, scheduling and loads drivers.
- **init:** Init is the very first process
  - Mounts the basic linux directories like /sys , /dev or /proc
  - Runs init.rc script



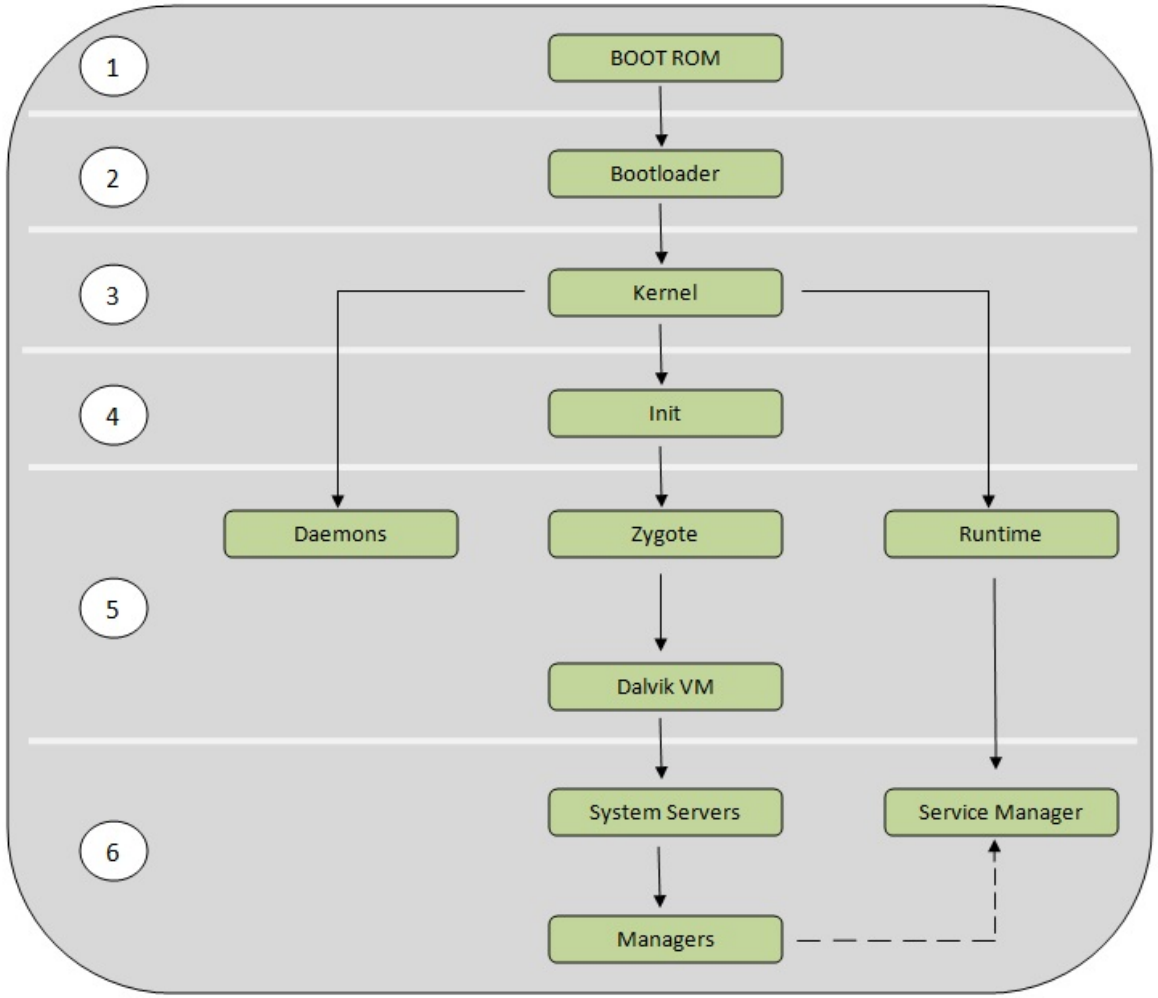
Boot-Sequence

- init.rc provides a set of generic initialization instructions required for booting
- **Zygote and Dalvik VM:** The Zygote enables code sharing across the Dalvik VM.



Boot-Sequence

- Zygote enables Android to launch apps in Memory efficient way.
- Zygote pre-loads and initializes all the core library classes required by the **Android Runtime**



Boot-Sequence

- **System Services and Managers:** Zygote then initializes several system services like battery service, mount service etc.
- Zygote initializes several managers as well like Telephony Manager, Activity Manager.