Mobile OS architectures

What is an Operating System?

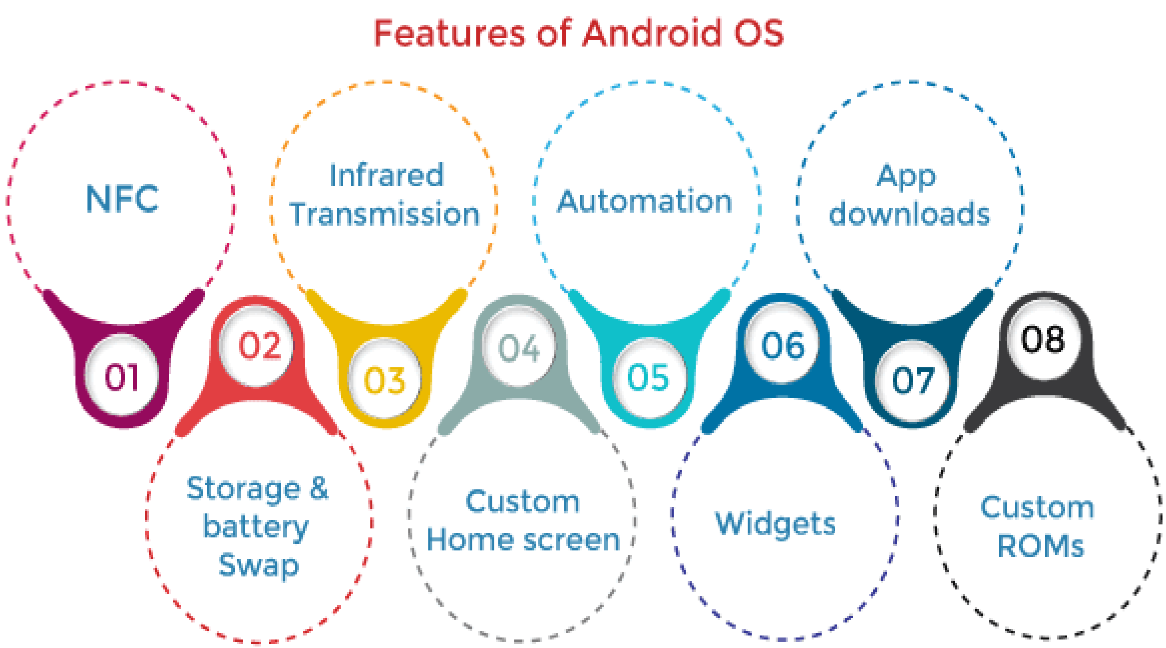
An **operating system (OS) :**is a system software component that **manages** and **controls** the computer **hardware** and **software** **resources** and provides common services to computer programs.

Android Operating System

Android is a mobile operating system based on a **modified version of the Linux kernel** and other **open-source software**, designed primarily for **touchscreen** mobile devices such as **smartphones** and tablets.

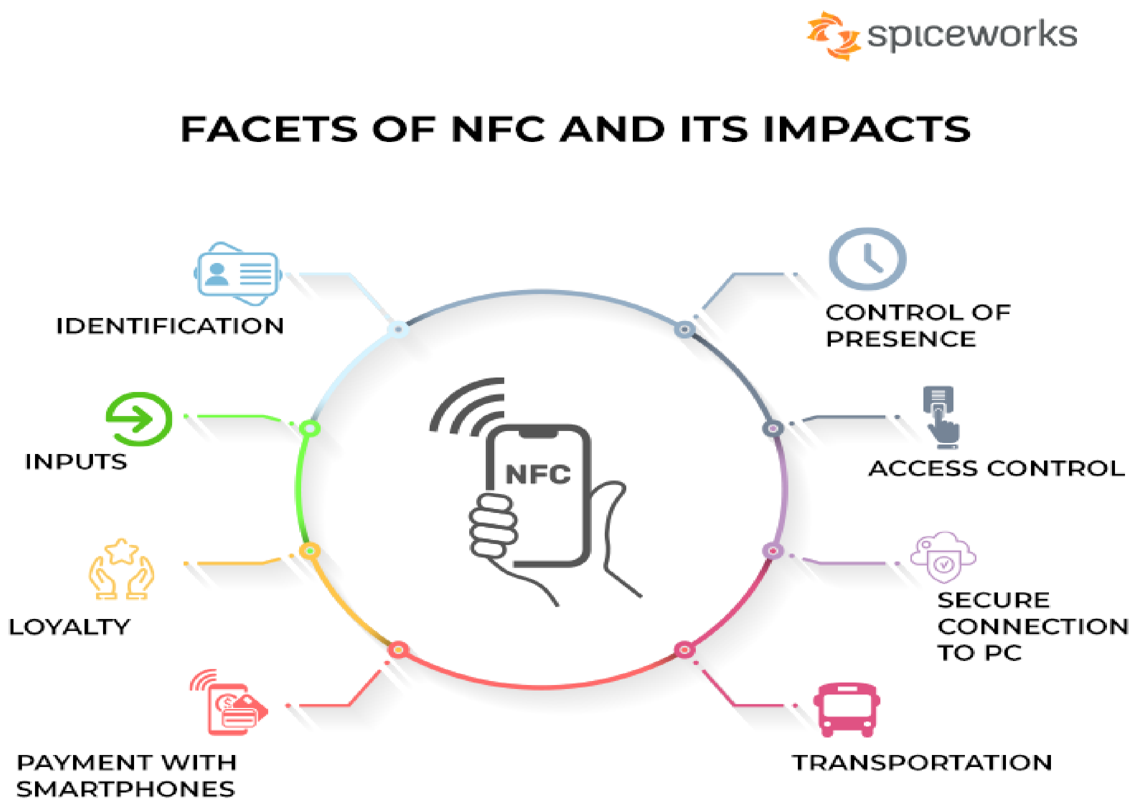
Features of Android Operating System

Below are the following unique features and characteristics of the android operating system, **such as:**



1. **Near Field Communication (NFC)**

   Near Field Communication, commonly abbreviated as **NFC**, is defined as a **wireless personal area network** (**PAN**) technology that connects two compatible devices in **very close proximity** to each other, in order to enable slow but reliable data transfer.



2. **Infrared Transmission**

   The Android operating system supports a built-in infrared transmitter that allows you to use your **phone** or **tablet** **as a remote control.**

3. **Automation**

   The ***Tasker*** app allows control of app permissions and also automates them.

Note: What Is Tasker?

Tasker is a powerful app that can help you automate a wide range of **day-to-day tasks** on your Android smartphone. It can single-handedly replace hundreds of purpose-specific apps and gives you a tool to tweak your device in all sorts of ways.

4. **Custom ROMs**

The process of installing a custom **ROM** typically involves several steps:

1. **Unlocking the Bootloader**: Most Android devices come with a **locked bootloader** that needs to be unlocked for custom **ROM** installation.
2. **Rooting the Device**: not always necessary.
3. **Installing a Custom Recovery**: A custom recovery, like **TWRP** (Team Win Recovery Project), replaces the stock recovery software and provides the **tools** needed to install custom **ROMs**.
4. **Flashing the ROM**: After backing up the device's existing software, the custom ROM can be installed (**flashed**) using the custom recovery environment.

Risks and Considerations

- **Warranty Void**: custom ROM can void the manufacturer's warranty.
- **Security Risks**: Some custom ROMs might not follow the same rigorous security protocols as official releases.
- **Stability Issues**: some ROMs might have bugs or stability issues.
- **Update and Support**: updates depend on the community or developers supporting the ROM.

Popular Custom ROMs

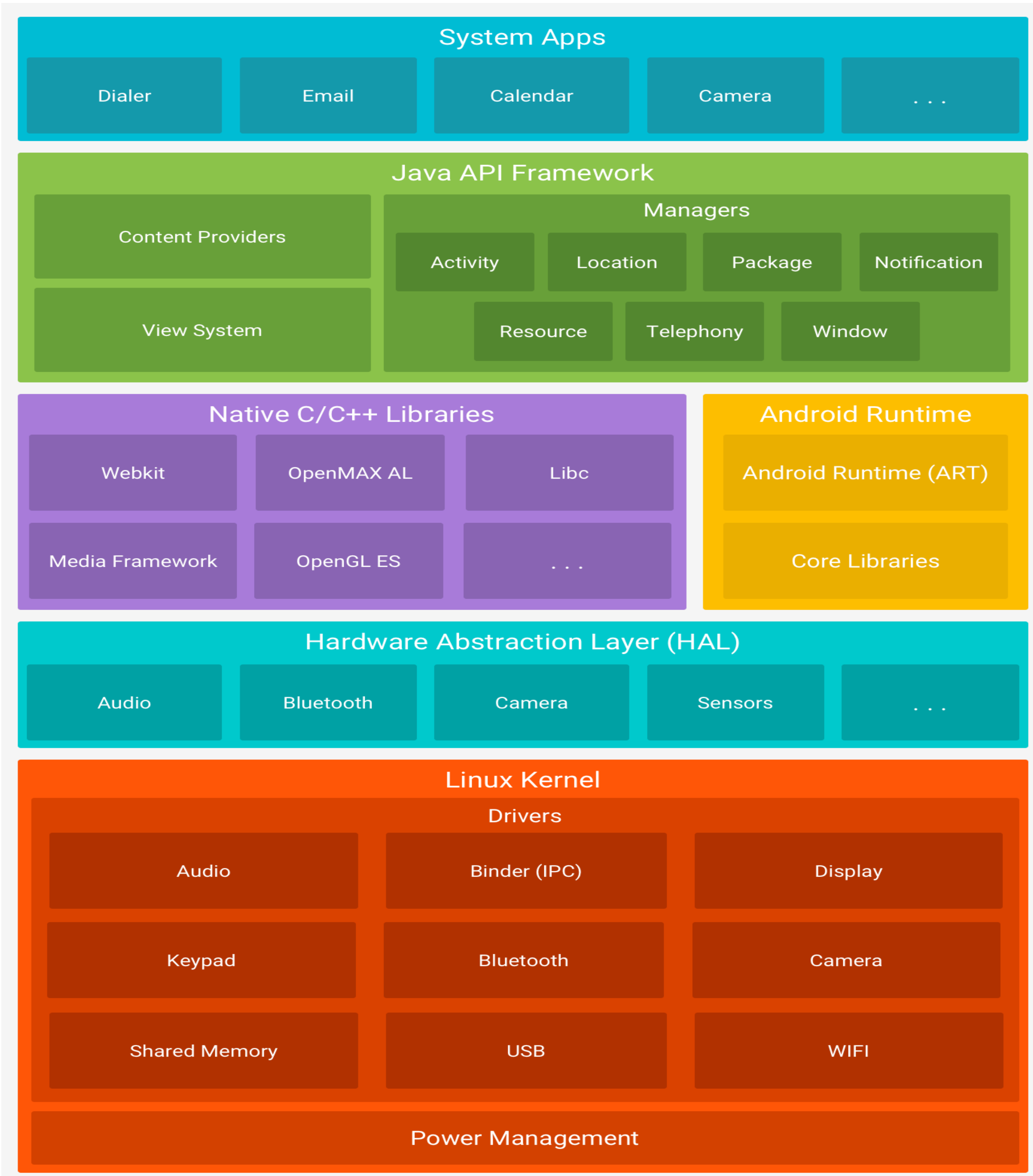
- LineageOS
- Resurrection Remix
- Pixel Experience
- Paranoid Android

Most-Popular-Custom-ROMs-for-Android

5. **Wireless App Downloads**
6. **Storage and Battery Swap**
7. **Custom Home Screens**
8. **Widgets**

Architecture of Android OS

The following diagram shows the major **components** of the **Android platform**.



1. The Linux Kernel

The foundation of the Android platform is the **Linux kernel**.

The features of the Linux kernel are:

- **Security:** The Linux kernel handles the **security** **between** the **application** and the **system**.
- **Memory Management:** It efficiently handles memory management, thereby providing the freedom to develop our apps.
- **Process Management:** It **manages** the process well, **allocates resources** to processes whenever they need them.
- **Network Stack:** It effectively handles **network communication**.
- **Driver Model:** It ensures that the application works properly on the device and hardware **manufacturers** responsible for building their **drivers** into the **Linux build**

2. Hardware Abstraction Layer (HAL)

The hardware abstraction layer (HAL) provides standard **interfaces** that expose device hardware capabilities to the **higher-level** **Java API framework**.

The **HAL** consists of multiple **library modules**, each of which implements an **interface** for a **specific type of hardware component**, such as the **camera** or **Bluetooth** module.

When a **framework API** makes a **call** to access **device hardware**, the Android system **loads** the **library module** for that **hardware** component.

Structure of Android HAL

The Android **HAL** is structured into multiple layers, with each layer responsible for abstracting different aspects of the hardware:

1. **HAL Interface Layers:**
   - defined by **Android** and specify how the operating system **interacts** with **hardware services**.
   - The interfaces are generally specified in **Interface Definition Language (IDL)** files, which recently include **HIDL** (**HAL Interface Definition**

Language) or **AIDL** (**Android Interface Definition Language**) for newer services.

2. **HAL Implementation:**
   - This is the **vendor-specific implementation** that actually communicates with the **hardware**.
   - Each **hardware component manufacturer provides** their own **implementation** of the **HAL** that **adheres** to the **interfaces** defined by Android.

3. **Vendor Modules:**
   - These are **dynamic shared libraries** loaded by the **Android system** at **runtime**.

How Android HAL Works

- **Service Management: Android HAL** uses the **binder IPC (Inter-Process Communication)** mechanism to allow **communication** between the **HAL** and **Java** application layers.
- **Modular Approach:** Each type of **hardware component** (like camera, sensors, audio, etc.) has its **own HAL module**. This modularity allows each component to be **developed**, **updated**, and **maintained** independently of others.

Examples of HAL in Android Devices

- **Camera HAL:** Manages interactions between the camera hardware components and the high-level camera application.
- **Audio HAL:** Deals with the audio management of the device, including audio output and input capabilities across various sound hardware.
- **Sensor HAL:** Handles data from device sensors, providing a uniform interface for motion, orientation, temperature sensors, and more.

3. Android Runtime

For 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 **Android Runtime (ART).**

ART is written to **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. Build tools, such as **d8**, compile Java sources into **DEX** bytecode,

Note: `d8` is a **command-line** tool that Android **Studio** and the Android **Gradle** plugin use to **compile your project's Java bytecode into DEX bytecode** that runs on Android devices. `d8` lets you use **Java 8** language features in your app's code.

Some of the major features of ART include the following:

- **Ahead-of-time (AOT) and just-in-time (JIT) compilation**
  - ART **compiles apps** at the **time** of **installation** into native machine code.
  - applications launch **faster** and use less **CPU** and **battery**.
  - it results in **larger application storage** size because each app includes compiled native code.
- **Optimized garbage collection (GC)**
  - minimizes application pauses,
  - helping to ensure **smoother UI animations** and **improved responsiveness** in apps.
- **Better debugging support.**
  - ART supports a wider range of **development** and **debugging** features that improve **profiling** of **applications**,
  - helping developers understand **performance issues** and **optimize** their code effectively.

How ART Works

When an application is installed on an Android device,

- **ART compiles** the app’s **bytecode** (from DEX files) into native **machine code** using its **AOT** compiler.
- **machine code** is then **executed** by the Android **device’s processor**.
- During execution, **ART** also uses **JIT** compilation techniques to optimize the performance of the native code further.

4. Native C/C++ Libraries

Many core Android system components and services, such as **ART** and **HAL**, are **built from native code** that **require native libraries** written in **C** and **C++.**
The Android platform provides **Java framework APIs** to expose the functionality of some of these native libraries to apps.

- **app:** Provides **access** to the **application model** and is the cornerstone of all Android applications.
- **content:** Facilitates content access, **publishing and messaging between applications** and **application components**.
- **database:** Used to access data **published by content providers** and includes **SQLite** database, management classes.
- **OpenGL:** A Java interface to the **OpenGL ES 3D graphics** rendering **API**.
- **os:** Provides **applications** with access to standard operating system services, including **messages**, system **services** and **inter-process communication**.
- **text:** Used to render and manipulate text on a device display.
- **view:** The fundamental building blocks of application user interfaces.
- **widget:** A rich collection of **pre-built** user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **WebKit:** A set of **classes** intended to **allow web-browsing** capabilities to be built into applications.
- **media:** Media **library** provides support to **play** and **record** an **audio** and video format.
- **surface manager:** It is responsible for managing access to the **display subsystem**.
- **SSL:** **Secure Sockets Layer** is a security technology to **establish** an **encrypted link** between a **web server** and a **web browser**.

5. Java API Framework

Application Framework provides several important **classes** used to **create** an **Android application**.

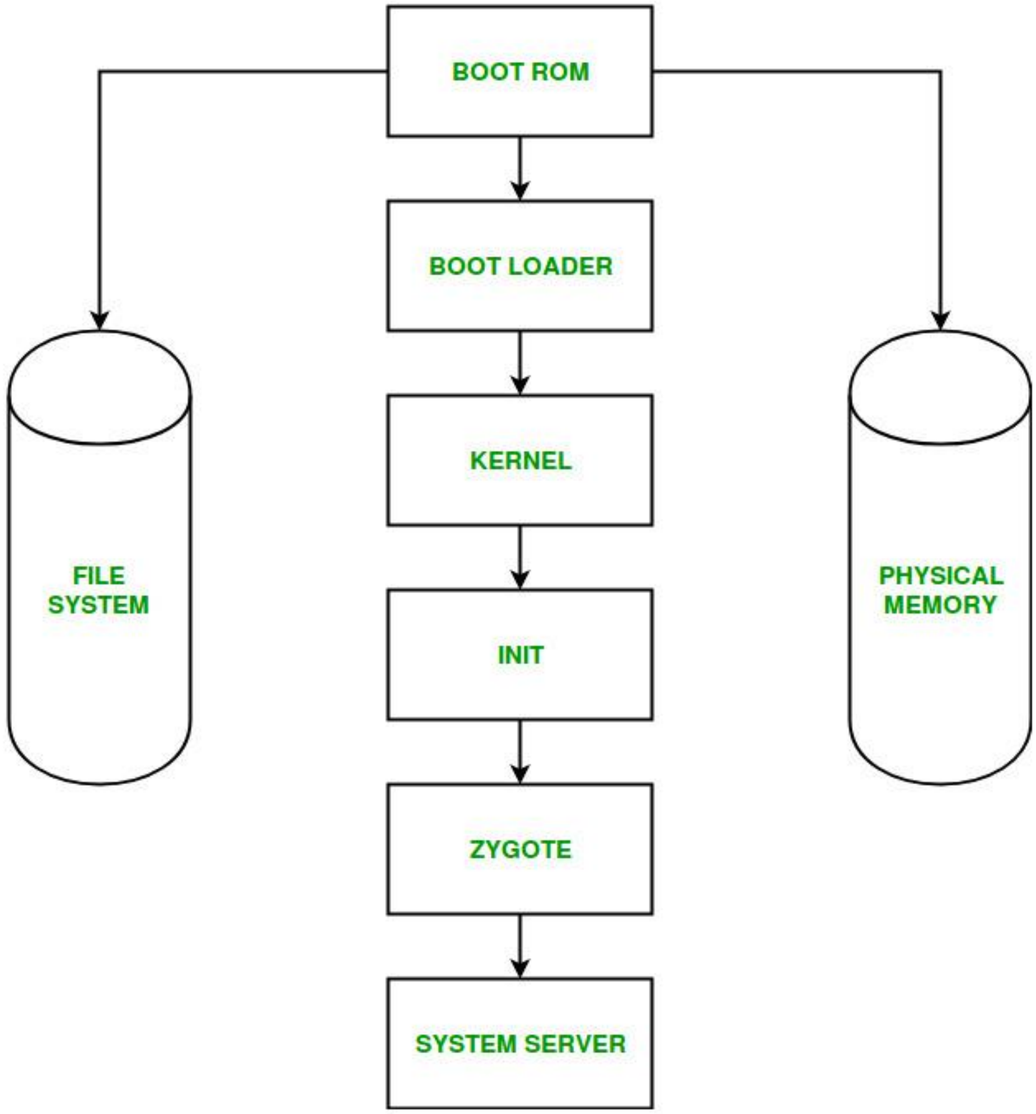
It includes different types of services, such as:

- **Activity Manager:** **Controls** all aspects of the application lifecycle and activity stack.
- **Content Providers:** Allows applications to publish and share data with other applications.
- **Resource Manager:** Provides **access** to **non-code embedded resources** such as strings, colour settings and user interface layouts.
- **Notifications Manager:** Allows applications to display alerts and notifications to the user.
- **View System:** An extensible set of views used to create application user interfaces.

6. System Apps

Android comes with a **set of core apps** for **email**, **SMS messaging**, **calendars**, **internet browsing**, **contacts**, and more.

Android Boot Process



Android Boot Process includes the following **six** steps:

1. **Boot ROM**:
   - Is known as power ON and system startup.
   - Whenever we press the **power button**, the **Boot ROM** code starts executing from a pre-defined location which is hardwired in **ROM**.
   - **Boot ROM loads** the **BootLoader** into **RAM** and starts executing.

2. **BootLoader**:
    - **Bootloaders** is a low-level **code** contains the **instructions** that tell a device how to start up and **find** the system **kernel**.
    - A **Bootloader** is a place where manufacturers put their **locks** and **restrictions**.
    - The bootloader is a code that is executed before any Operating System starts to run.

    The BootLoader executes in 2 Stages:

    a) **first stage**, it **detects** external **RAM** and **loads** a **program** which helps in the second stage.
    b) **second stage**, the bootloader **setups** the **network**, **memory** etc which requires to run Kernel.

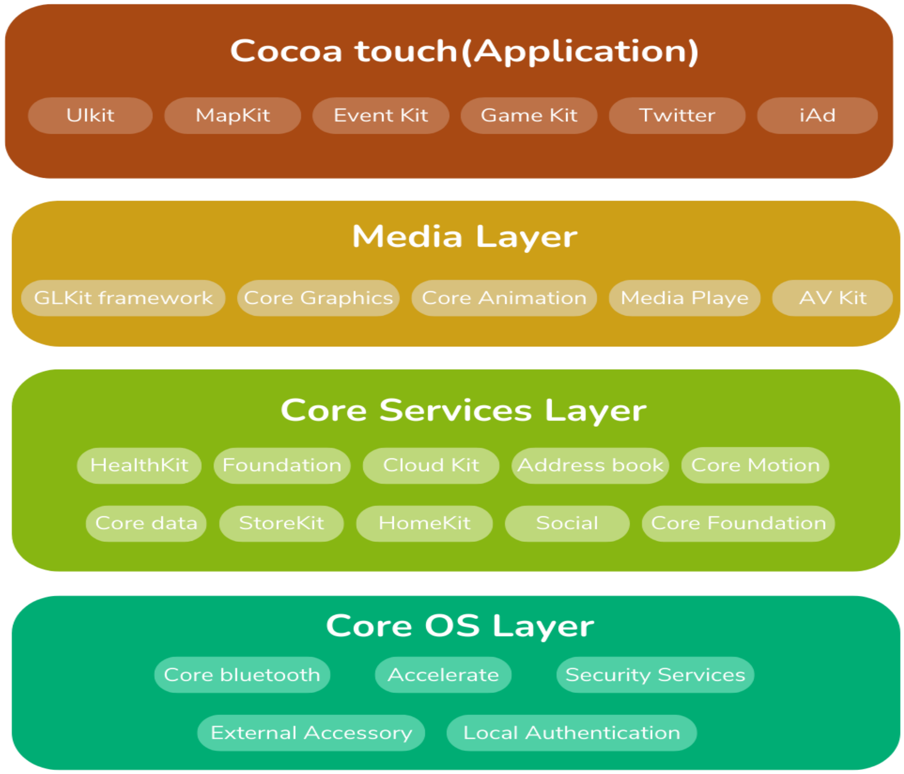
3. **Kernel**:
    - Once **kernel boots**, it starts setup
        - ✓ cache,
        - ✓ protected memory,
        - ✓ scheduling,
        - ✓ loads drivers,
        - ✓ starts kernel daemons,
        - ✓ mounts root file system,
        - ✓ initializing Input/Output,
        - ✓ starts interrupts,
        - ✓ initializes process table.
    - When **kernel finish** system setup first thing it looks for **"init"** in system files and launch **root process** or **first process** of a system.
4. **Init**:
    - **Init** is the very **first process** or we can say that it is the **grandfather** of all the **processes**.

5. **Zygote and Dalvik VM**:
    - The **Zygote** process is a specialized **parent process** for **all** Android **application** processes.
    - It serves as a template for creating new application processes and helps in **optimizing** application **startup times** by **preloading** **common resources and libraries.**

6. **System Servers:** After **zygote** preloads all necessary Java Classes and resources, it starts System Server.
   - The **System server** is the **core** of the **Android system**.
   - The first thing that happens is that the **server** will **load** a **native library** called **android_servers** that **provides interfaces** to **native functionalities**.
   - Then the **native init** method that will **setup native services called**.
   - After setting up the native services it creates the **server thread**.
   - This **thread** will start the **remaining services** in the system according to the necessary start order.
   - Each **service** is **running** in a separate **Dalvik** thread in the **SystemServer**.

   Once system **Services up and running in memory**, Android has **completed boot** process, At this time "**ACTION_BOOT_COMPLETED**" standard broadcast action will fire.

iOS Architecture



There are four abstraction levels in it.

- **Core OS Layer**: This layer forms the foundation of the iOS architecture. It provides essential **functionalities** like **memory management**, **security**, and **task scheduling**. **Frameworks** like **Core Bluetooth**, **Core Foundation**, and **Kernel frameworks** reside in this layer.
- **Core Services Layer**: This layer offers **core functionalities** that **applications** heavily rely on. It **includes services** like **multitasking**, **networking**, **location services**, and **file system access**. Some important **frameworks** in this layer are **Address Book**, **CloudKit**, **Core Motion**, and **Core Location**.
- **Media Layer:** this layer deals with everything **multimedia-related**. It provides **frameworks** for handling **graphics**, **audio**, and **video**. **Core Animation**, **Core Graphics, AVFoundation**.
- **Cocoa Touch:** This **framework** is the **heart of iOS application development**. It provides **UI components (UIKit), user interaction APIs (UIKit),** and **application lifecycle management (Foundation)**.