# Unit - 1
# Introduction to Mobile Operating System

## ❖ Introduction :-

- A Mobile Operating System (OS) serves as the software foundation for mobile devices, including smartphones and tablets. It is responsible for managing both hardware and software resources, providing a user-friendly interface, and enabling the installation and execution of applications.

## ❖ Additional requirement for Mobile OS :-

### ➢ Security :-
- Continuous improvement of security features such as enhanced encryption, secure boot processes, and secure app sandboxing.
- Implementation of advanced biometric authentication methods and multifactor authentication for device access.

### ➢ Privacy Controls :-
- Robust privacy settings allowing users granular control over app permissions and data access.
- Clear and easily understandable privacy policies for users, ensuring transparency in data collection and usage by the OS and applications.

### ➢ Optimized Performance :-
- Efficient resource management to ensure smooth performance on a wide range of devices, from entry-level to flagship smartphones.
- Regular optimization updates to maintain responsiveness and speed throughout the device's lifecycle.

### ➢ Compatibility and Interoperability :-
- Seamless integration with a variety of devices, including wearables, smart home devices, and other IoT (Internet of Things) gadgets.
- Standardized protocols for easy communication and data sharing between Android devices and other platforms.

What is IoT? The Internet of Things (IoT) describes the network of physical objects—“things”—that are embedded with sensors,
software, and other technologies for the purpose of connecting and exchanging data
with other devices and systems over the internet.

## ➢ User Interface Customization :-

- Improved customization options for the user interface, allowing users to personalize their experience without compromising system stability.
- Support for innovative UI/UX features to enhance user interaction and accessibility.

## ➢ Energy Efficiency :-

- Advanced power management features to optimize battery life, including improved background process handling and power-saving modes.
- Support for emerging technologies that enhance energy efficiency, such as advancements in battery technology and power consumption optimization.

## ➢ Developer Tools and APIs :-

- Comprehensive development tools and APIs (Application Programming Interfaces) that empower developers to create innovative and feature-rich applications.
- Regular updates to the Android SDK (Software Development Kit) with new features, performance improvements, and bug fixes.

## ➢ Accessibility Features:

- Enhanced accessibility features catering to a diverse user base, including improved screen readers, voice control, and support for assistive technologies.
- Collaboration with accessibility communities to gather feedback and continuously improve the accessibility of the OS.

## ➢ Regular Updates and Support:

- Timely and consistent software updates to address security vulnerabilities, introduce new features, and improve overall stability.
- Prolonged support for older devices, ensuring a more extended and secure lifecycle for existing hardware.

## ➢ Environmental Sustainability :-

- Initiatives to reduce the environmental impact, such as optimizing resource usage, promoting energy efficiency, and supporting eco-friendly practices in device manufacturing and disposal.

## ❖ Constraints of Smart Mobile OS :-

### ➢ Hardware Limitations :

- Mobile OS must work within the constraints of the hardware it runs on. Older devices may struggle to support newer OS versions, limiting their performance and feature capabilities.

### ➢ Fragmentation:

- Android, in particular, faces the challenge of fragmentation due to the wide variety of devices and manufacturers. This can lead to issues with software compatibility and consistency across devices.

### ➢ Security Concerns:

- Smart mobile OS are often targeted by malware, viruses, and other security threats. Users can be vulnerable to data breaches, identity theft, and other malicious activities if the OS does not have robust security measures.

### ➢ Battery Life:

- Optimizing battery life remains a significant challenge. Advanced features and background processes can lead to increased power consumption, affecting the overall user experience.

### ➢ Network Dependence:

- The effectiveness of certain features, such as cloud services and real-time data synchronization, depends heavily on network connectivity. Limited or unreliable network access can hinder the performance of these features.

### ➢ Limited Storage:

- Mobile devices often come with limited storage space. The OS needs to manage storage efficiently, and users may face challenges when dealing with large app sizes and multimedia files.

### ➢ User Privacy Concerns:

- As mobile OS become more integrated with various services and applications, concerns about user privacy arise. Users may worry about data collection, tracking, and the security of personal information.

### ➢ Over-the-Air Updates:

- While over-the-air updates are a convenient way to deliver new features and security patches, they may pose challenges for users with limited data plans or slow internet connections.

## ➢ Learning Curve:

- Introducing new features and updates can result in a learning curve for users. If changes are too drastic or frequent, it may lead to confusion and frustration among users.

## ➢ Regulatory Compliance:

- Compliance with various regulations and standards, such as data protection laws, can be challenging. Mobile OS developers need to stay updated on global and regional compliance requirements.

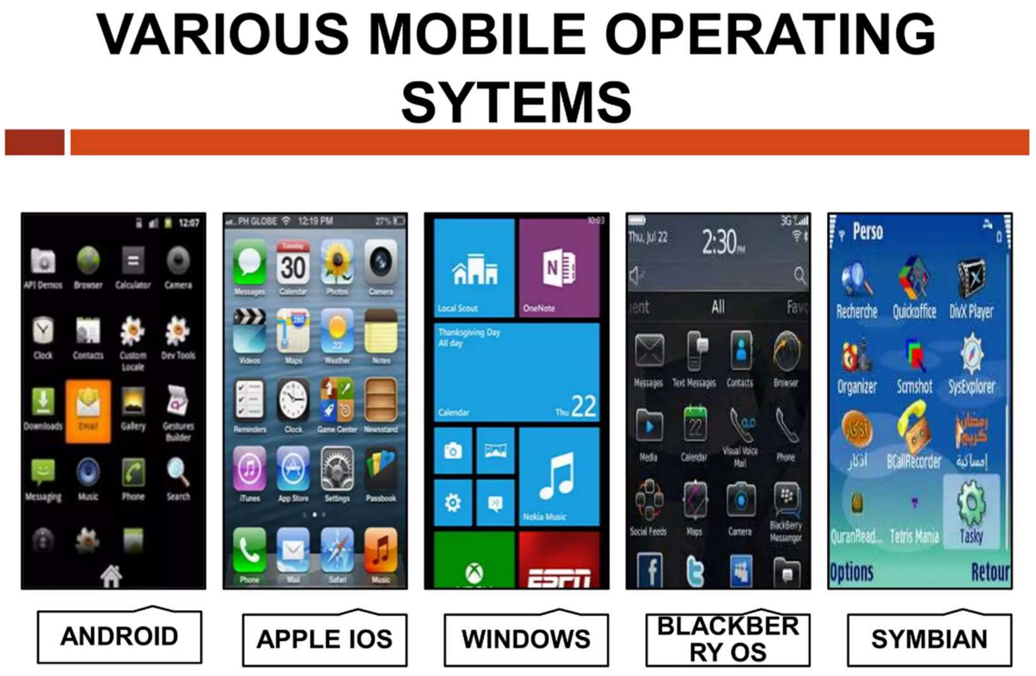
## ➢ Cross-Platform Compatibility:

- Ensuring seamless compatibility with different devices and platforms can be challenging, especially when users expect a consistent experience across smartphones, tablets, wearables, and other smart devices.

## ➢ App Store Policies:

- Adhering to the policies of app stores (e.g., Google Play Store, Apple App Store) can be a constraint for app developers. Stricter policies may limit the type and functionality of apps that can be distributed through these platforms.

## ❖ Types of Mobile OS :-



### ➢ iOS:

- iOS, developed by Apple, is known for its closed ecosystem and seamless integration with Apple's hardware. It prioritizes user experience, security, and performance. App distribution is exclusively through the Apple App Store.

### ➢ Android:

- Android, an open-source OS developed by Google, powers a vast range of devices from different manufacturers. It provides high customization options for users and developers. The Google Play Store is the primary app distribution platform.

### ➢ BlackBerry:

- BlackBerry OS, with its focus on security and productivity, was historically popular among business users. However, its market share has declined with the rise of other platforms.

### ➢ Microsoft:

- Windows Mobile, developed by Microsoft, aimed for a consistent experience across devices. However, it struggled to gain traction and has been largely phased out.

### ➢ Symbian:

- Once dominant, Symbian was a widely used OS, especially by Nokia. However, its market share declined with the emergence of more modern mobile platforms.

## ❖ Generalized Architecture of Mobile OS :-

### ➢ Kernel Layer:

- At the core of the mobile OS is the kernel, which is responsible for low-level hardware interactions, memory management, process management, and device drivers. It acts as an interface between the hardware and the higher layers of the operating system.

### ➢ Hardware Abstraction Layer (HAL):

- The Hardware Abstraction Layer provides a standardized interface between the hardware and the upper layers of the operating system. It allows the OS to interact with various hardware components without needing to understand the specifics of each device.
-

### ➢ Device Drivers:

- Device drivers are specific software components that enable the OS to communicate with various hardware peripherals such as cameras, sensors, touchscreens, and more. These drivers ensure proper functioning and interaction with external devices.

## ➢ System Libraries:

- System libraries consist of precompiled code libraries that provide essential functions and services to applications and the operating system itself. These libraries include functions for graphics rendering, networking, security, and other core functionalities.

## ➢ Runtime Environment:

- This layer includes the runtime environment for running applications. It consists of components like the Virtual Machine (e.g., Dalvik or ART in Android, or the iOS runtime environment for Swift and Objective-C), which executes application code and manages memory.

## ➢ Middleware Layer:

- Middleware acts as a bridge between the lower-level OS components and the application layer. It includes components such as databases, communication protocols, and other services that applications can use.

## ➢ Application Framework:

- The application framework provides a set of high-level services and APIs (Application Programming Interfaces) that simplify application development. It includes components for user interface management, input handling, and access to device features.

## ➢ User Interface (UI):

- The UI layer is responsible for presenting information to the user and gathering input. It includes components for handling graphical elements, touch events, and user interactions. The UI layer may include elements such as widgets, themes, and UI controls.

## ➢ Applications:

- At the top of the stack are the user-facing applications. These applications leverage the services provided by the lower layers of the OS to deliver specific functionalities. Applications can be pre-installed (system apps) or installed by users from an app store.

## ➢ App Runtime:

- Some mobile operating systems include a runtime environment specifically designed for running applications. For example, Android uses the Dalvik or ART runtime for executing Java-based applications.

## ❖ Comparison of Mobile OS:

### ➢ iOS :-

▪ **Strengths:**

- **Seamless Integration:** Offers a unified experience across Apple devices, allowing seamless transitions and synchronization through iCloud.
- **Premium User Experience:** Known for its smooth and intuitive user interface, as well as consistent hardware and software design.
- **Strong Security:** Closed ecosystem and strict app store policies contribute to a more secure environment.

▪ **Weaknesses:**

- **Limited Customization:** Compared to Android, iOS provides limited options for customization, restricting users in terms of appearance and functionality.
- **Exclusive to Apple Devices:** iOS is only available on Apple devices, limiting choice for users who prefer other hardware.

### ➢ Android :-

▪ **Strengths:**

- **High Customization:** Android is highly customizable, allowing users to personalize their devices extensively with different launchers, widgets, and themes.
- **Diverse Hardware Support:** Runs on a wide range of devices from various manufacturers, providing users with more choices in terms of form factor and pricing.
- **Large App Ecosystem:** Google Play Store boasts a vast collection of apps catering to different needs.

▪ **Weaknesses:**

- **Fragmentation:** Due to the open-source nature and various manufacturers, Android devices often run different versions of the OS, leading to fragmentation.
- **Potential Security Concerns:** The openness of Android can make it more susceptible to malware and security vulnerabilities, especially on devices with outdated software.

### ➢ BlackBerry:

- **Strengths:**
  - **Focus on Security:** Historically known for strong security features, especially appealing to enterprise users.
  - **Productivity Features:** BlackBerry devices were popular for their physical keyboards and productivity-focused features.
- **Weaknesses:**
  - **Declining Market Share:** BlackBerry's market share has significantly decreased over the years, impacting app development and support.
  - **Limited App Ecosystem:** The BlackBerry app store is not as extensive as those of iOS and Android.

### ➢ Microsoft (Windows Mobile):

- **Strengths:**
  - **Consistent Experience:** Provided a consistent user experience across Windows-powered devices.
- **Weaknesses:**
  - **Phased Out:** Microsoft has largely phased out Windows Mobile, discontinuing support and development.
  - **Limited App Support:** Windows Mobile struggled to compete with iOS and Android in terms of the app ecosystem.
    .

### ➢ Symbian:

- **Strengths:**
  - **Once Dominant:** Symbian was once a dominant mobile OS, especially in the pre-smartphone era.
- **Weaknesses:**
  - **Declined in Popularity:** Lost market share and popularity due to the rise of more modern and user-friendly platforms.
  - **Outdated:** Became outdated in terms of features and user interface compared to contemporary mobile OS.

## ❖ Android Operating System :-

## ➢ Introduction :-

- The Android Operating System, developed by the Open Handset Alliance (OHA), is an open-source platform designed for mobile devices. It offers a versatile environment for developers and has become the most widely used mobile OS globally.

### ❖ History of Android OS :-

- **2003:**
  - **Foundation:** Android Inc. was founded by Andy Rubin, Rich Miner, Nick Sears, and Chris White in October 2003. The company aimed to create an advanced operating system for digital cameras initially.

- **2005:**
  - **Google Acquisition:** In August 2005, Google acquired Android Inc. This acquisition was seen as a strategic move to enter the mobile market.

- **2007:**
  - **Open Handset Alliance (OHA):** In November 2007, Google announced the formation of the Open Handset Alliance, a consortium of several hardware, software, and telecommunication companies committed to advancing open standards for mobile devices.

- **2008:**
  - **Android OS Debut:** The first version of the Android operating system, Android 1.0, was officially released in September 2008 with the HTC Dream (also known as the T-Mobile G1) as the first Android-powered device.

- **2009:**
  - **Cupcake (Android 1.5):** In April 2009, the first major update, Android 1.5 Cupcake, was released, introducing features like on-screen keyboards and video recording.

- **2010:**
  - **Eclair (Android 2.0/2.1):** Android 2.0 Eclair, released in October 2009, brought improvements to the user interface and added features like turn-by-turn navigation.

- **2011:**
  - **Gingerbread (Android 2.3):** Released in December 2010, Gingerbread focused on improving performance and user experience.

- **2011:**
  - **Honeycomb (Android 3.0):** Introduced in February 2011, Honeycomb was specifically designed for tablets, optimizing the Android experience for larger screens.

- **2011:**
  - **Ice Cream Sandwich (Android 4.0):** Unveiled in October 2011, this version aimed to bridge the gap between smartphone and tablet interfaces, offering a more unified experience.

- **2012:**
  - **Jelly Bean (Android 4.1/4.2/4.3):** Rolled out from July 2012, Jelly Bean brought performance improvements, a smoother interface, and features like Google Now.

- **2014:**
  - **KitKat (Android 4.4):** Released in October 2013, KitKat focused on optimizing the operating system for devices with lower hardware specifications.

- **2014:**
  - **Material Design:** Introduced with Android Lollipop (5.0) in October 2014, Material Design brought a more consistent and visually appealing design language to Android.

- **2015:**
  - **Marshmallow (Android 6.0):** Released in October 2015, Marshmallow focused on improving battery life, introducing granular app permissions, and enhancing overall performance.

- **2016:**
  - **Nougat (Android 7.0/7.1):** Released in August 2016, Nougat brought split-screen multitasking, improved notifications, and other refinements.

- **2017:**
  - **Oreo (Android 8.0/8.1):** Released in August 2017, Oreo included features like picture-in-picture mode, notification dots, and improved battery life management.

- **2018:**
  - **Pie (Android 9):** Released in August 2018, Pie introduced gesture-based navigation, a digital wellbeing dashboard, and adaptive battery features.

- **2019:**
  - **Android 10:** Released in September 2019, Android 10 emphasized privacy features, introduced a system-wide dark mode, and improved gesture navigation.

- **2020:**
  - **Android 11:** Released in September 2020, Android 11 focused on conversation management, enhanced privacy controls, and improved media controls.

- **2021:**
  - **Android 12:** Released in October 2021, Android 12 brought a major redesign with Material You, focusing on personalization, privacy dashboard, and smoother animations.

## ❖ Versions of Android OS:

### ➢ Key Features by Version:

- **Android 1.0 (2008):**
  - Commercial debut on the HTC Dream (T-Mobile G1).
  - Basic features included notifications, app multitasking, and the Android Market.

- **Android 1.5 Cupcake (2009):**
  - Introduced on-screen keyboards and video recording.
  - First version to use dessert-themed codenames.

- **Android 1.6 Donut (2009):**
  - Brought improved search functionality and support for different screen sizes.

- **Android 2.0/2.1 Eclair (2009):**
  - Introduced a refreshed user interface and live wallpapers.
  - Added support for multiple accounts and Exchange email.

- **Android 2.2 Froyo (2010):**
  - Featured performance improvements and the introduction of Adobe Flash support.
  - USB tethering and portable Wi-Fi hotspot functionality were added.

- **Android 2.3 Gingerbread (2010):**
  - Focused on refining the user interface and improving performance.
  - Introduced support for front-facing cameras and near field communication (NFC).

- **Android 3.0 Honeycomb (2011):**
  - Specifically designed for tablets with a holographic user interface.
  - Featured multitasking, widgets, and tabbed browsing.

- **Android 4.0 Ice Cream Sandwich (2011):**
  - Unified the smartphone and tablet user interfaces.
  - Introduced the Holo design language and facial recognition unlock.

- **Android 4.1/4.2/4.3 Jelly Bean (2012):**
  - Brought smoother user interface transitions and Google Now.
  - Introduced Daydream screensavers and multi-user support for tablets.

- **Android 4.4 KitKat (2013):**
  - Focused on optimizing the operating system for devices with lower hardware specifications.
  - Introduced the "Ok Google" voice command.

- **Android 5.0/5.1 Lollipop (2014):**
  - Brought Material Design, a more visually appealing and consistent design language.
  - Enhanced notifications and introduced the ART runtime.

- **Android 6.0 Marshmallow (2015):**
  - Focused on improving battery life with Doze mode.
  - Introduced granular app permissions and Google Now on Tap.

- **Android 7.0/7.1 Nougat (2016):**
  - Introduced split-screen multitasking and improved notification controls.
  - Included features like Daydream VR and a Vulkan API for gaming.

- **Android 8.0/8.1 Oreo (2017):**
  - Introduced picture-in-picture mode and notification dots.
  - Focused on optimizing battery life and improving system performance.

- **Android 9 Pie (2018):**
  - Introduced gesture-based navigation and Adaptive Battery.
  - Included a Digital Wellbeing dashboard for monitoring screen time.

- **Android 10 (2019):**
  - Enhanced privacy controls and introduced a system-wide dark mode.
  - Improved gesture navigation and Live Caption feature.

- **Android 11 (2020):**
  - Focused on conversation management and introduced bubble notifications.
  - Improved device and media controls.

- **Android 12 (2021):**
  - Introduced the Material You design with enhanced personalization.
  - Emphasized a more dynamic and colorful user interface.

## ❖ Architecture of Android OS:

- **Linux Kernel:**
  - At the core of Android is the Linux kernel, responsible for device drivers, memory management, security, process management, and networking.

- **Hardware Abstraction Layer (HAL):**
  - The Hardware Abstraction Layer provides a standardized interface between the Android framework and the device-specific hardware. It allows the upper layers to communicate with various hardware components without needing to understand the specifics of each device

.

- **Native C/C++ Libraries:**

  - These libraries include essential components written in native languages (C/C++) that provide core functionalities:
    - **Libc:** Standard C library.
    - **Libm:** Math library.
    - **Surface Manager:** Manages the display subsystem.
    - **Media Framework:** Provides support for audio and video playback.

- **Android Runtime (ART):**

  - The ART is the runtime environment for Android applications. It replaces the earlier Dalvik Virtual Machine.
  - ART executes applications compiled into a more efficient binary format, improving performance and reducing memory usage.

- **Core Libraries:**

  - These Java-based libraries provide essential functionalities for Android applications:
    - **Java Core Libraries:** Standard Java libraries for data structures, utilities, and networking.
    - **Android Libraries:** Libraries specific to Android, such as the Android Support Library.

- **Android Application Framework:**
  - This layer provides a set of high-level services and APIs that simplify application development:
    - **Activity Manager:** Manages the lifecycle of applications and provides a framework for user interactions.
    - **Package Manager:** Manages the installation and removal of applications.
    - **Content Providers:** Allows applications to share data with each other.
    - **View System:** Provides the building blocks for creating user interfaces.

- **System Apps and User Apps:**
  - Android includes various system applications (e.g., phone, contacts, settings) and user-installed applications.
  - Each application runs in its own process and is isolated from other applications for security and stability.

- **Java API Framework:**
  - The Java API Framework includes the set of APIs that developers use to build Android applications:
    - **Android API:** APIs for accessing Android-specific features.
    - **Java API:** Standard Java APIs for application development.

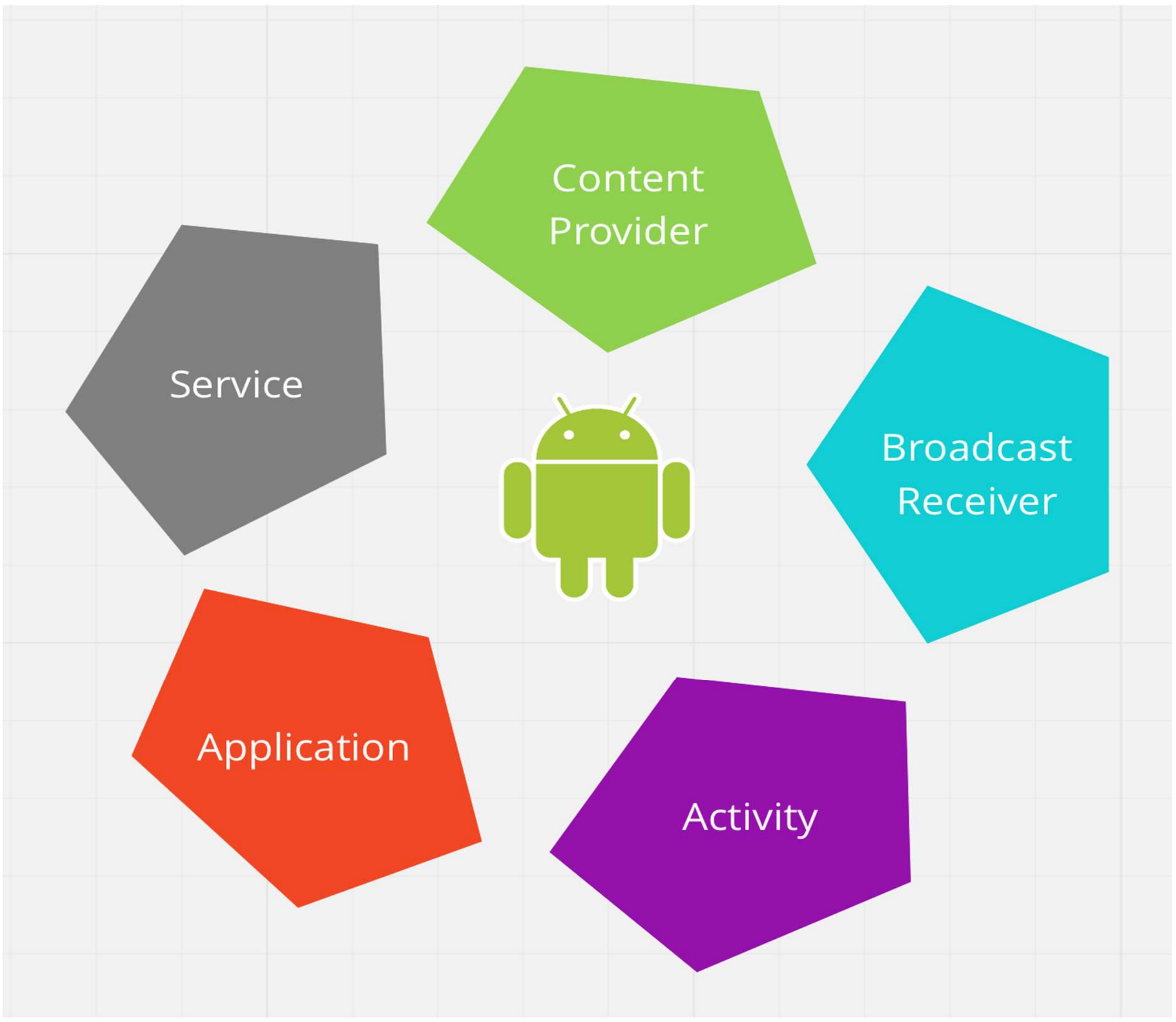
- **Application Layer:**
  - The top layer is where user applications run, utilizing the Android framework and libraries to provide various functionalities.

- **Key Components and Services:**
  - **Activity Manager:** Manages the lifecycle of applications and provides a common navigation backstack.
  - **Window Manager:** Manages the window hierarchy and user interface.
  - **Content Providers:** Facilitates data sharing between applications.
  - **View System:** Provides UI components for creating the user interface.

- **Additional Components:**
  - **Notification Manager:** Manages notifications shown to the user.
  - **Location Manager:** Manages device location.
  - **Package Manager:** Manages application packages and their installation.

- **Android Security Model:**
  - **Linux Security Model:** Utilizes Linux's user-based security model to isolate applications.
  - **Application Sandboxing:** Each application runs in its own process with limited access to system resources.
  - **Permissions System:** Controls what resources and data an application can access.

## ❖ Building Blocks of Android Application:



**1. Activities:**

- Definition: Activities represent the UI and user interaction of an application.
- Lifecycle: Activities go through a lifecycle with methods like onCreate, onStart, and onDestroy.
- Purpose: They serve as the entry point for user interactions and can be standalone or interconnected.

**2. Services:**

- Definition: Services are background processes that perform long-running operations independently of the UI.
- Tasks: They handle tasks such as playing music, handling network transactions, or other operations that don't require a user interface.
- Independence: Services can run in the background even if the application is not in the foreground.

**3. Broadcast Receivers:**

- Definition: Broadcast Receivers respond to system-wide broadcasts or events.
- Events: They allow components to react to changes or events, even when the application is not actively running.
- Example: Responding to a battery low warning or incoming SMS.

**4. Content Providers:**

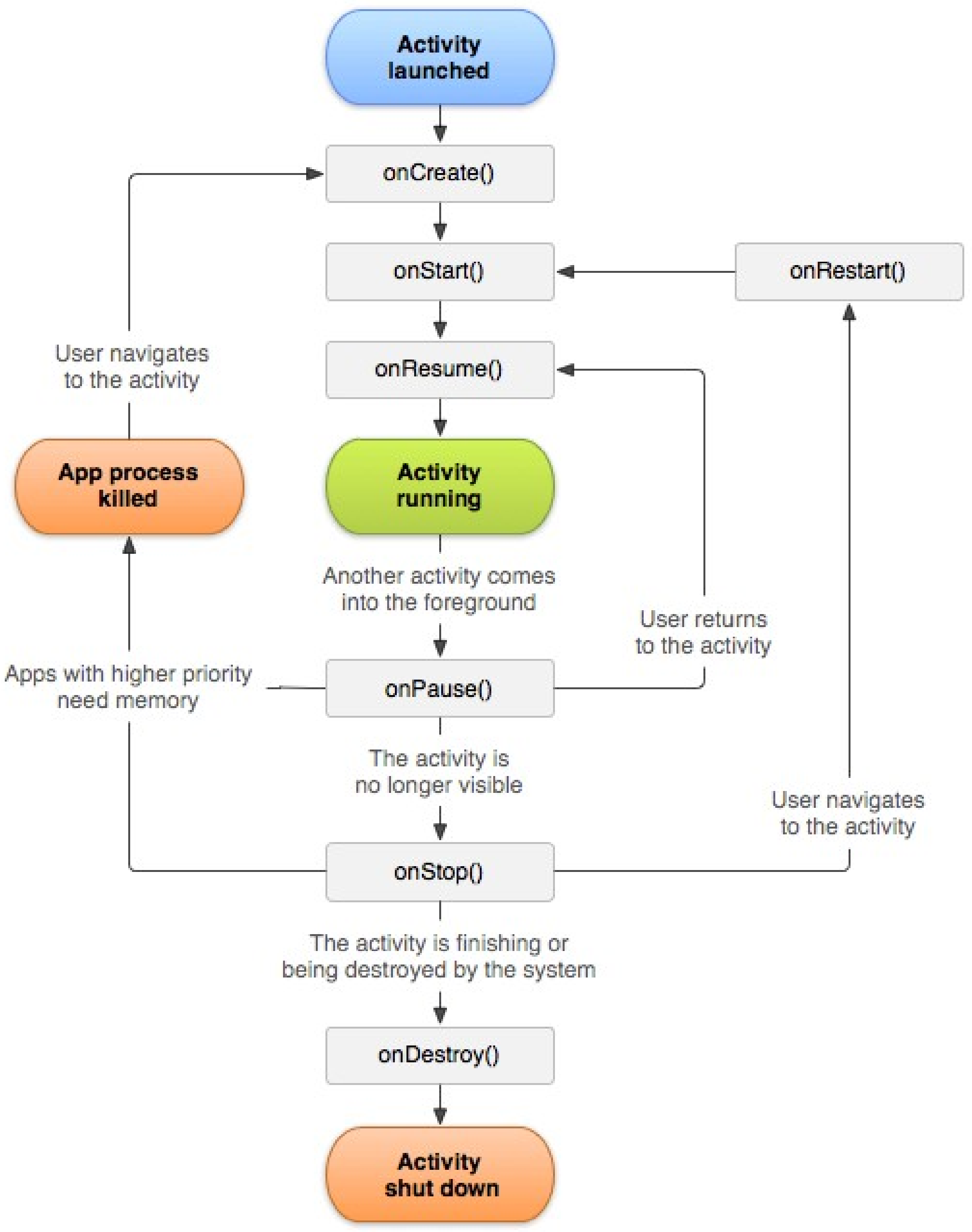
- Definition: Content Providers manage shared sets of application data.
- Data Sharing: They enable secure data sharing between applications in a standardized manner.
- Example: Managing contacts, calendar events, or other shared resources.

**5. Intents:**

- Definition: Intents facilitate communication between components.
- Requesting Actions: They allow one component to request an action from another, either within the same application or between different applications.
- Example: Starting an activity, service, or sending a broadcast.

## ❖ Work with Activity:-

### ➢ Activity Lifecycle:


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

**1. onCreate():**

- **Purpose:** This is the first method called when the activity is created.
- **Actions:** Typically, you perform one-time initialization tasks here, such as inflating the layout, initializing variables, or setting up UI components.
- **Important:** Always call **super.onCreate(savedInstanceState)** to ensure the correct behavior of the superclass.

**2. onStart():**

- **Purpose:** Called when the activity is becoming visible to the user.
- **Actions:** Commonly used to start components that will remain visible while the user interacts with the app. Initialization tasks that are needed each time the activity becomes visible can be placed here.

**3. onResume():**

- **Purpose:** Called when the activity is about to start interacting with the user.
- **Actions:** This is a critical stage where you typically start animations, resume ongoing operations, or refresh UI components. User input is received during this phase.

**4. onPause():**

- **Purpose:** Called when the activity is no longer in the foreground but still visible.
- **Actions:** This is a good place to pause or adjust operations that should not continue while the activity is not in focus. Examples include stopping animations, releasing resources, or unregistering listeners.

**5. onStop():**

- **Purpose:** Called when the activity is no longer visible to the user.
- **Actions:** Use this method to release resources that are not needed while the activity is not visible. You might also save persistent data or perform cleanup tasks.

**6. onRestart():**

- **Purpose:** Called when the activity is restarting after being stopped.
- **Actions:** This method provides an opportunity to perform any tasks that should happen every time the activity is restarted, such as refreshing data from a database.

**7. onDestroy():**

- **Purpose:** Called before the activity is destroyed.
- **Actions:** Cleanup tasks and final resource releases should be performed here. This is the last method called before the activity is removed from memory.