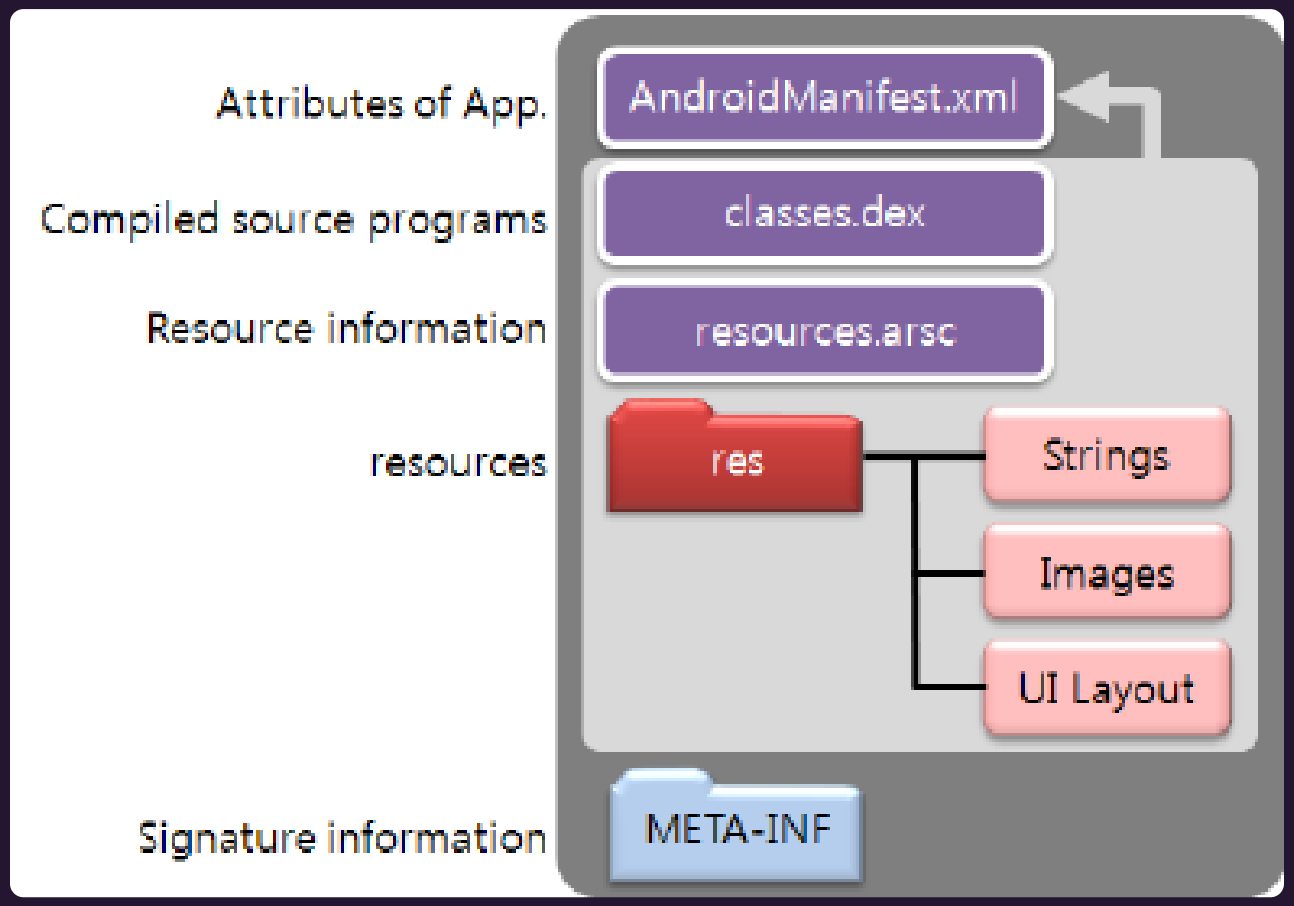
Android Application

by Durgesh Thakare

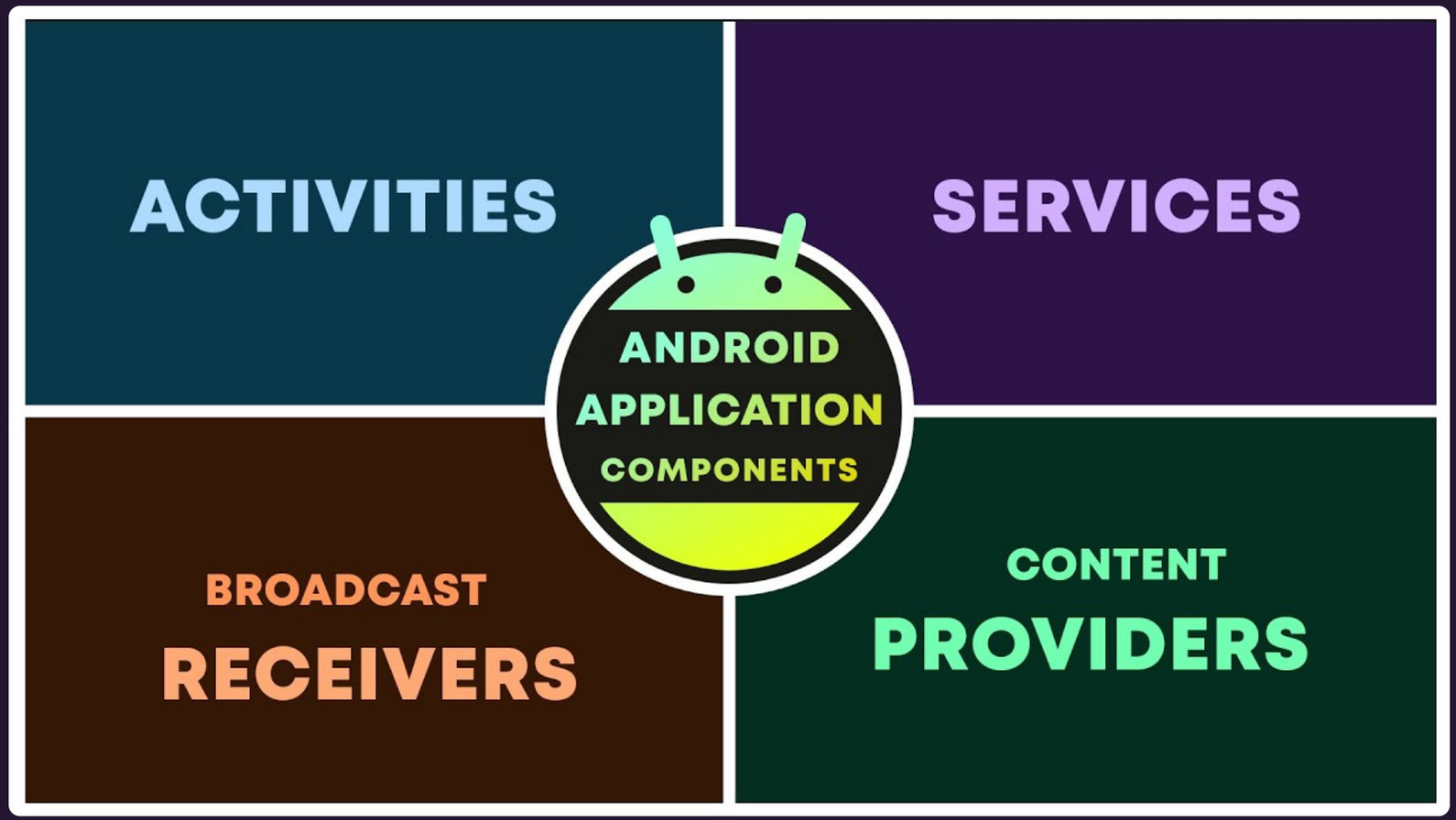
Android Application

An Android application, often referred to as an APK (Android Package), is essentially an archive file or package. An archive file is a container that holds various files and folders. Similarly, an APK contains several crucial files and resources that make up an Android app. These key components include:



1. **Manifest File (AndroidManifest.xml):** This file serves as the blueprint of the app, containing essential information like the app's name, version, package name, permissions, and more.
2. **DEX Files:** These files contain the compiled code of the app. They are executed by the Android Runtime (ART) or Dalvik Virtual Machine (DVM), making the app functional.
3. **Resource Files:** These files house various resources such as images, layouts, strings, and other assets that are used to create the app's user interface and provide content.
4. **Meta-Inf Directory:** This directory often contains important files like manifest files and certificate files that help ensure the integrity and security of the app.
5. **RESOURCES:** In addition to the resource files mentioned above, the RESOURCES directory also includes resource files for different device configurations, such as layout files optimized for different screen sizes or language-specific string files. This allows the app to adapt its appearance and behavior based on the user's device and language settings.

Android Application Components



- Android apps are composed of several components, including Activities, Services, Broadcast Receivers, and Content Providers.
- Activities represent UI screens and user interactions.
- Services handle background tasks.
- Broadcast Receivers listen for system-wide events.
- Content Providers manage data and enable data sharing between apps.

Intent

Intent:in the context of Android refers to a fundamental component of the Android operating system that is used for communication and interaction between different parts of an application and between different Android applications. It plays a crucial role in interprocess communication (IPC) within the Android ecosystem

Purpose of Intents:

- Intents are used to specify an action to be performed, such as opening a specific activity, starting a service, or broadcasting a message.
- They facilitate communication between various components of an app, even if they are in separate parts of the app.

Types of Intents:

- **Implicit Intents:** These do not specify a particular component to execute the action but rather declare a general action to be performed. Implicit intents allow the Android system to choose the appropriate component based on the user's preferences and installed apps. For example, launching a web URL is an implicit intent as the Android system can open it using a web browser of the user's choice.
- **Explicit Intents:** These explicitly define the component (e.g., a specific activity or service) that should handle the action. They are typically used within the same app to navigate between different parts of the app or to start a specific service. Explicit intents are also used when an app wants to ensure that a specific component is used for a particular action.

Android Application Components

1. **Activities:**
    - Activities are the user interface screens in an Android app.
    - Each activity typically corresponds to a single screen with a specific user interface.
    - Activities manage the presentation and user interaction for a specific purpose.
    - An Android app can have multiple activities, and they can interact with each other through Intents.
2. **Services:**
    - Services are background processes that perform tasks without a user interface.
    - They can run indefinitely, even if the user switches to another app.
    - Services are used for tasks such as playing music in the background, fetching data from the internet, or performing long-running operations.
3. **Broadcast Receivers:**
    - Broadcast Receivers are components that listen for system-wide events or broadcast messages.
    - They can respond to events like incoming SMS messages, battery low notifications, or connectivity changes.
    - Broadcast Receivers allow apps to react to events and take specific actions.
4. **Content Providers:**
    - Content Providers manage the app's data and allow it to be shared with other apps.
    - They provide a standard interface for interacting with data stored in databases or other data sources.
    - Content Providers enable data sharing and can be used by other apps to access and modify data, with proper permissions.

These components work together to create a cohesive Android application. For example, an app may have multiple activities that allow users to navigate through various screens, services running in the background to handle data synchronization, broadcast receivers to respond to system events, and content providers to manage data access.

Additionally, Intents serve as a means of communication between these components, allowing them to start activities, trigger services, or send and receive broadcast messages. This modular structure and communication mechanism make Android applications flexible and extensible.

Android Permissions

Permissions in the context of Android refer to a system of security measures that control an app's access to certain device features, data, and services. These permissions are used to protect user privacy, device integrity, and overall security. Apps must request and be granted specific permissions to perform certain actions or access particular resources on an Android device.

Types of Permissions:

- **Normal Permissions:** These are permissions that are automatically granted to an app when it is installed. They typically grant access to harmless or non-sensitive resources, such as internet access or access to the network state. Users do not need to explicitly approve these permissions.**Example:** `INTERNET` permission
- **Runtime Permissions:** These are permissions that are requested and granted at runtime when the app needs to access sensitive resources like the camera, location, or contacts. Users are prompted with a permission dialog, and they can choose to grant or deny the permission. This gives users more control over which resources an app can access and enhances user privacy and security.
- **Special Permissions:** These are additional permissions that require explicit approval from the user and provide access to highly sensitive resources, such as system settings, call logs, or SMS messages. Special permissions are usually reserved for system apps or apps that require advanced functionality. Requesting and being granted special permissions typically involves a more rigorous review process to ensure user safety and prevent misuse of sensitive data.

Declaration in AndroidManifest.xml

Apps declare the permissions they need in the AndroidManifest.xml file. This declaration informs both the system and users about what resources or actions the app requires access to.

```
<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.READ_CONTACTS" />
```

Runtime Permission Handling:

- Apps should check whether they have been granted the necessary permissions at runtime before attempting to use them.
- If a permission is not granted, the app should request it from the user using the appropriate API, provide context for why it needs the permission, and gracefully handle scenarios where the user denies the request.

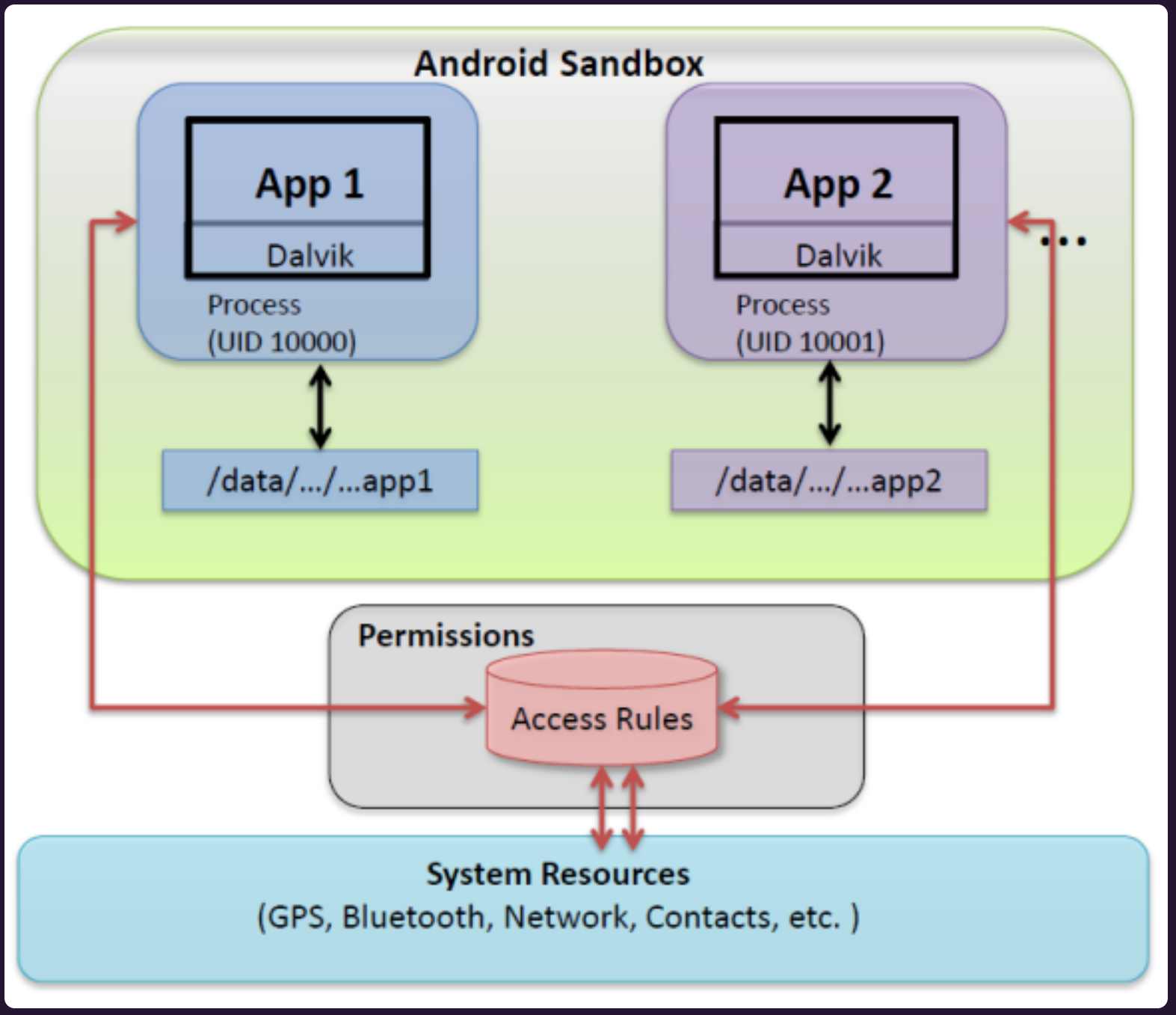
Application Sandboxing

Application sandboxing is a fundamental security mechanism in Android that ensures each app runs in its own isolated environment, separate from other apps, and with limited access to the system's resources

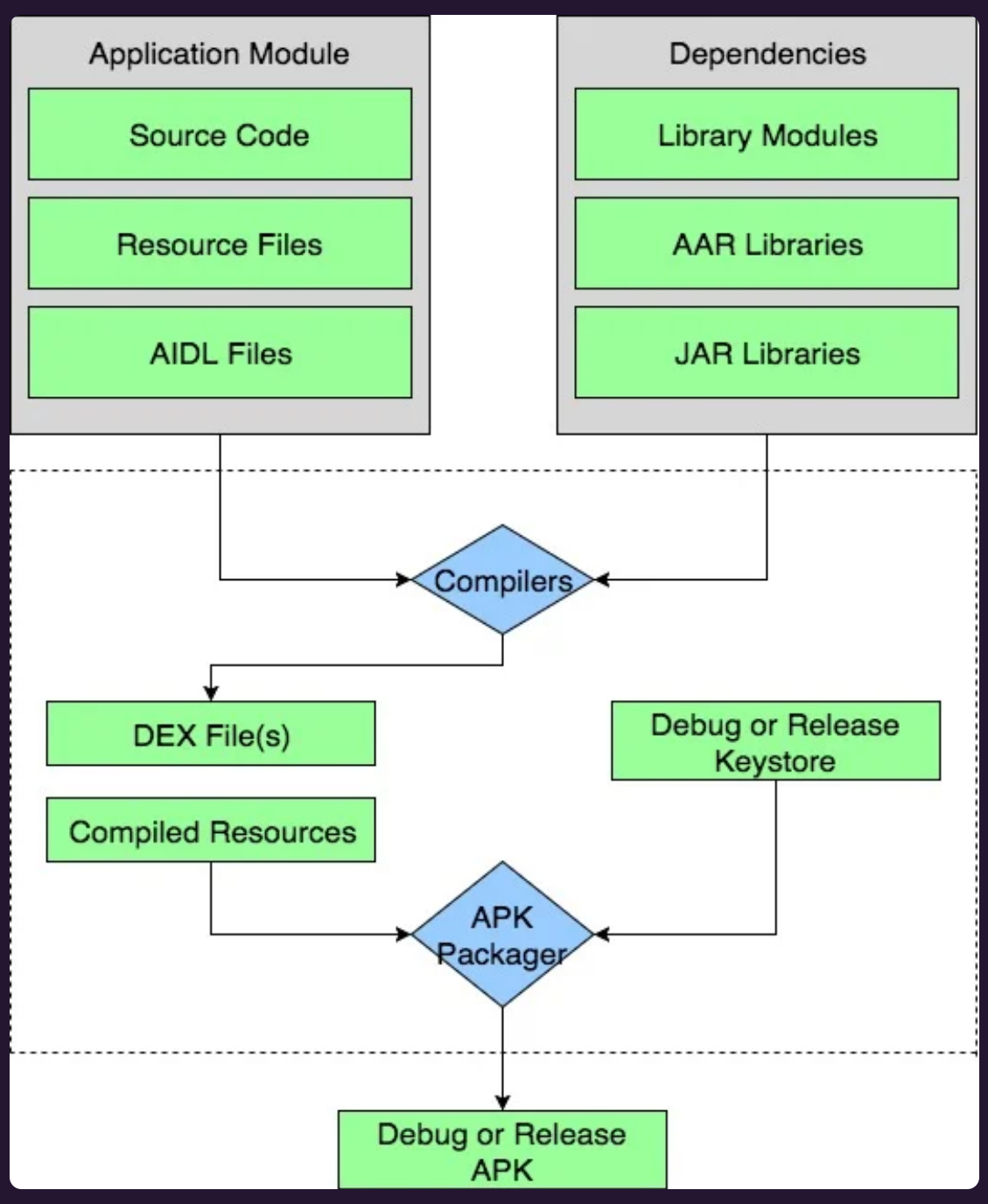
Isolation by User ID (UID): Each app is assigned a unique User ID (UID), such as "10101+app1." This UID is used to segregate app processes, files, and data.

File System Isolation: App data and files are stored in directories specific to their package name, for example, "data/data/com.package1.app1." This separation prevents one app from accessing or modifying another app's data.

Interprocess Communication (IPC): Android provides controlled IPC mechanisms, like Intents and Content Providers, that allow apps to communicate while maintaining security boundaries.



Android Application Build Process



The Android application build process is a series of steps that transform source code and resources into an Android Package (APK) file, which is the installable file for Android applications. Here's a breakdown of the Android application build process:

1. **Source Code and Resource Files:**
    - The development of an Android application begins with writing source code and creating resource files (e.g., layout XML, image assets, strings, and other resources).
    - These source code files and resource files are typically organized within an application module in an Android project.
2. **Dependencies and Library Modules:**
    - Android applications often depend on external libraries or modules to provide additional functionality.
    - These dependencies are declared in the app's build.gradle file, and Android Studio's build system (Gradle) handles the downloading and integration of these dependencies.
    - Library modules within the same project can also be used to modularize and share code and resources.
3. **Compilation and Resource Processing:**
    - The source code (Java or Kotlin) is compiled into intermediate bytecode by the Java Compiler (javac) or Kotlin Compiler.
    - Resource files are processed to generate binary resource files that can be efficiently loaded at runtime.
    - These compilation and resource processing steps are managed by the Android Gradle plugin.
4. **Dex Files Generation:**
    - The intermediate bytecode is further transformed into Dalvik Executable (DEX) files for Android versions prior to Android 5.0 (Lollipop) or into Android Runtime (ART) DEX files for Android 5.0 and later.
    - The DEX files contain the app's executable code and are generated using the Android DEX compiler (dx).
5. **APK Package Creation:**
    - The APK package is created by bundling together all the compiled DEX files, processed resource files, manifest file, assets, and other necessary files.
    - The AndroidManifest.xml file describes the app's configuration, components, and permissions.
    - The APK also includes the digital signature of the app to ensure its integrity and authenticity.
6. **Debug or Release APK:**
    - Two types of APKs can be generated:
        - **Debug APK:** This is used for testing during development. It includes debugging information and may have limited optimizations.
        - **Release APK:** This is the final APK that is signed with a keystore and optimized for distribution on app stores. It does not include debugging information and is suitable for production use.
7. **Keystore Signing:**
    - In the case of a release APK, it is signed with a digital certificate (keystore) to verify the authenticity and integrity of the app.
    - This signing process ensures that the APK has not been tampered with and that it can be installed on Android devices.
8. **Distribution:**
    - Once the APK is generated and signed, it can be distributed through various channels, such as the Google Play Store, app distribution platforms, or manual installation on devices.

In summary, the Android application build process involves compiling source code and processing resources to generate DEX files, which are then bundled into an APK package. Depending on whether it's a debug or release build, the APK may undergo additional optimizations and signing before being distributed to users. This process is managed by the Android Gradle build system and development tools like Android Studio.