ISIS-3510

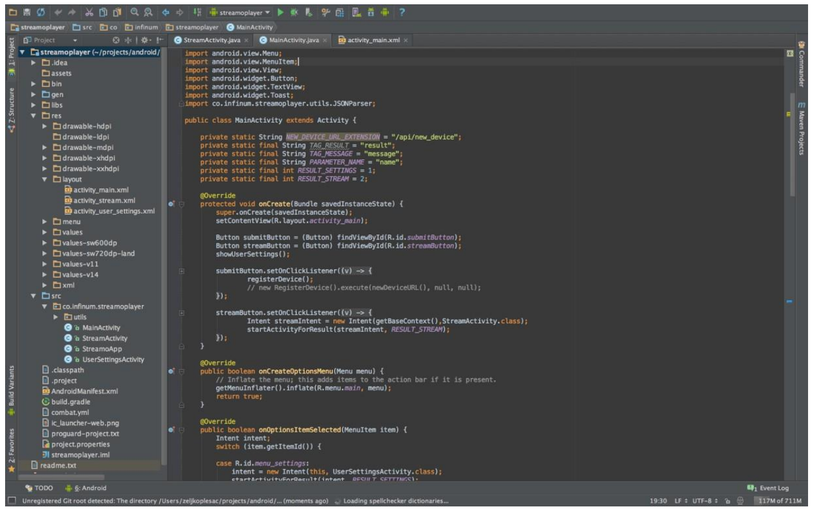
Mobile App Development 3.0

(a.k.a., Software Engineering for Mobile Applications)

THE ANDROID OS

WHAT IS THE DIFFERENCE BETWEEN SDK, API, AND IDE?

IDE



THIRD PARTY LIBRARIES/TOOLS

AdMob

unityAds

WeChat

Retrofit

Google Analytics

amazon web services

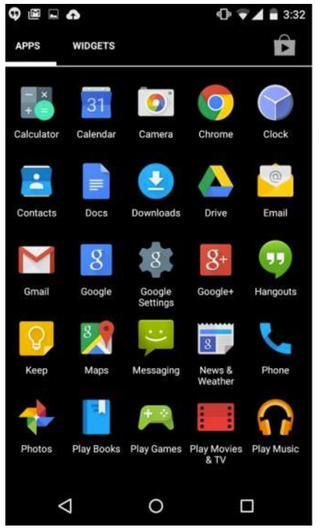
crashlytics

cucumber

SDK

android SDK

ANDROID OS



IDE (Integrated Development Environment)

image taken from https://www.techrepublic.com/article/best-ide-software/

It is the integrated environment where applications are developed, facilitating code writing and compilation

android studio

eclipse

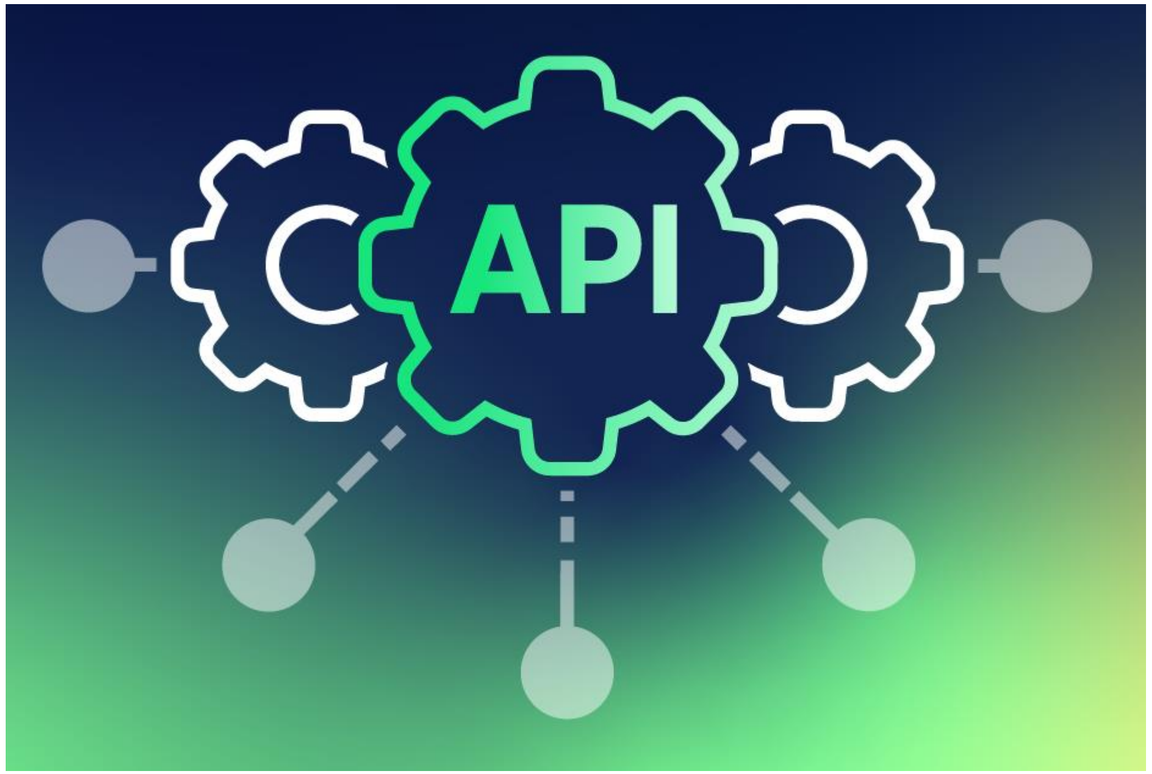
Visual Studio

PC

Imagen tomada de https://ed.team/blog/editor-vs-ide-que-usar-para-programar

IDE (Integrated Development Environment)

API (Application Programming Interface)



- It is a set of definitions and protocols that allow different applications to communicate with each other.
- Define the rules and structures that enable software to interact with other software
- Depending on the context in which they are used, APIs can have different functions and categories.

API (Application Programming Interface)

- HTTP services APIs

Retrofit is an HTTP client library for Android and Java that simplifies communication with web services. Retrofit is based on HTTP and is commonly used to interact with RESTful APIs.

- Android API Framework

SDK (Software Development Kit)

It is a set of **tools that developers use to create applications** for the Android operating system. It includes:

•**Libraries and APIs**: Provides the necessary tools to interact with Android system functionalities, such as accessing the camera, sensors, storage, and more.

•**Compiler and development tools**: Allows writing, debugging, and compiling the application's source code.

•**Android emulator**: Simulates Android devices on a computer, enabling the testing and debugging of applications without the need for a physical device.

Image taken from https://medium.com/helpshift-engineering/reverse-engineer-your-favorite-android-app-863a797042a6

ANDROID SDK

Set of utilities and tools that are executed on the developer machine

The SDK is required to create Android apps, by using command line or an IDE

ADB: provides remote access via USB/WiFi to utilities and services running on the device (e.g., shell unix)

AAPT: allows browsing, creation, and edition of .jar and .apk files

DX: compiles source code files to DEX

ADB (Android Debug Bridge)

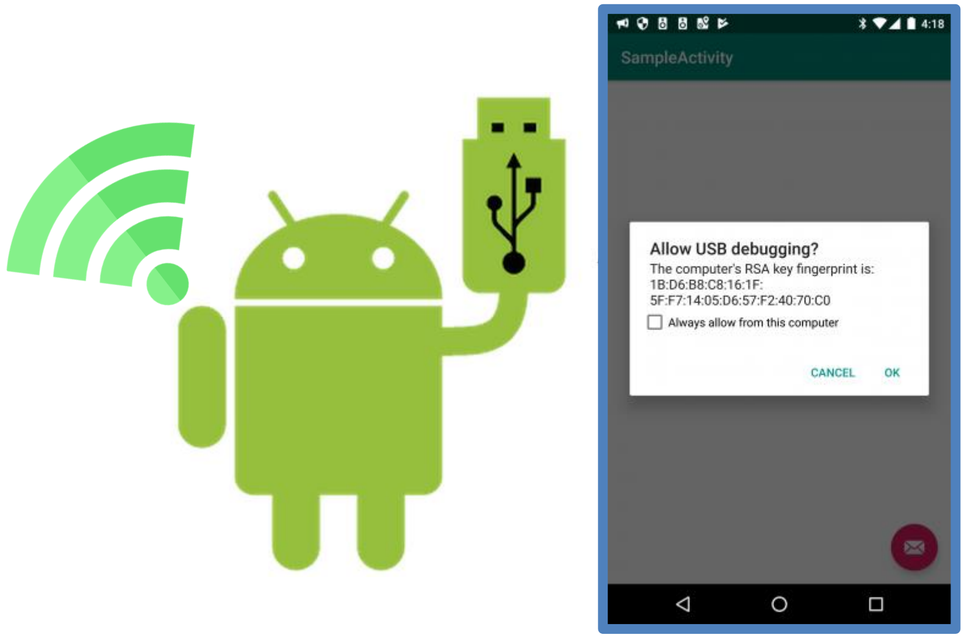


Image taken from https://www.kodeco.com/621437-android-debug-bridge-adb-beyond-the-basics?page=2#toc-anchor-003

This tool allows you to perform many development tasks, including:

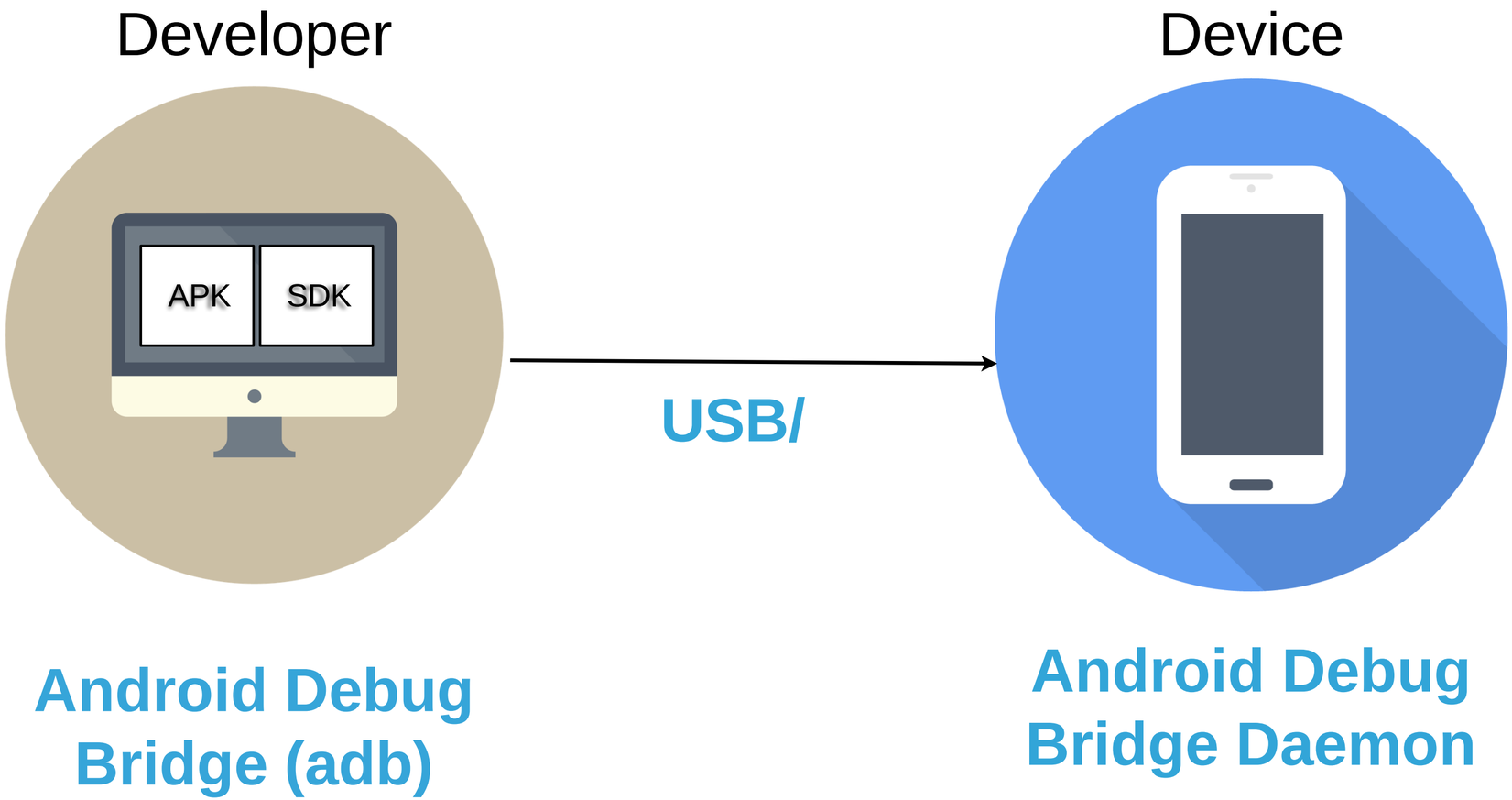
- Install/Uninstall apps to a device
- Debugging apps.
- Viewing device and app logs.
- Execute commands in the device
- Transfer files

```
Android Debug Bridge version 1.0.32

 -a                            - directs adb to listen on all interfaces for a connection
 -d                            - directs command to the only connected USB device
                                 returns an error if more than one USB device is present.
 -e                            - directs command to the only running emulator.
                                 returns an error if more than one emulator is running.
 -s <specific device>          - directs command to the device or emulator with the given
                                 serial number or qualifier. Overrides ANDROID_SERIAL
                                 environment variable.
 -p <product name or path>     - simple product name like 'sooner', or
                                 a relative/absolute path to a product
                                 out directory like 'out/target/product/sooner'.
                                 If -p is not specified, the ANDROID_PRODUCT_OUT
                                 environment variable is used, which must
                                 be an absolute path.
 -H                            - Name of adb server host (default: localhost)
 -P                            - Port of adb server (default: 5037)
 devices [-l]                  - list all connected devices
                                 ('-l' will also list device qualifiers)
 connect <host>[:<port>]       - connect to a device via TCP/IP
                                 Port 5555 is used by default if no port number is specified.
 disconnect [<host>[:<port>]]  - disconnect from a TCP/IP device.
                                 Port 5555 is used by default if no port number is specified.
                                 Using this command with no additional arguments
                                 will disconnect from all connected TCP/IP devices.

device commands:
  adb push [-p] <local> <remote>
                               - copy file/dir to device
                                 ('-p' to display the transfer progress)
  adb pull [-p] [-a] <remote> [<local>]
                               - copy file/dir from device
                                 ('-p' to display the transfer progress)
                                 ('-a' means copy timestamp and mode)
  adb sync [ <directory> ]     - copy host->device only if changed
                                 (-l means list but don't copy)
                                 (see 'adb help all')
  adb shell                    - run remote shell interactively
```

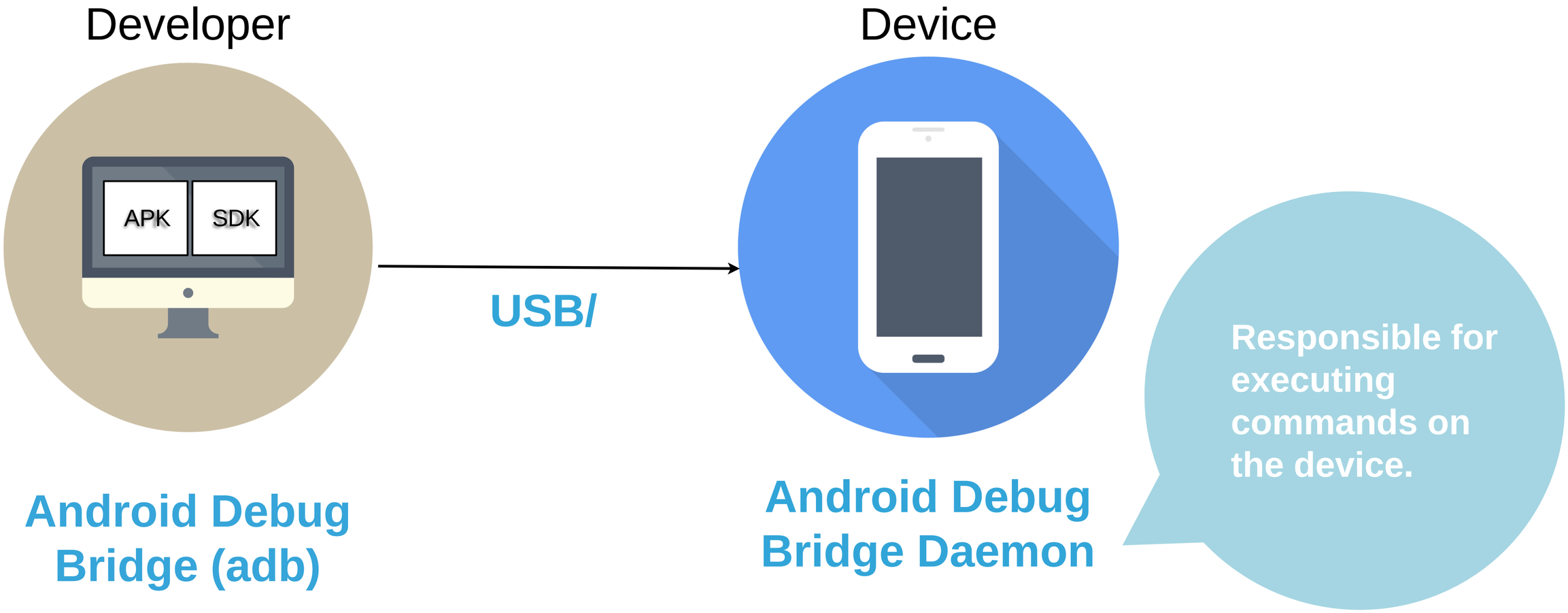
AAPT (Android Asset Packaging Tool)

```
Android Asset Packaging Tool

Usage:
 aapt l[ist] [-v] [-a] file.{zip,jar,apk}
   List contents of Zip-compatible archive.

 aapt d[ump] [--values] WHAT file.{apk} [asset [asset ...]]
   badging          Print the label and icon for the app declared in APK.
   permissions      Print the permissions from the APK.
   resources        Print the resource table from the APK.
   configurations   Print the configurations in the APK.
   xmltree          Print the compiled xmls in the given assets.
   xmlstrings       Print the strings of the given compiled xml assets.

 aapt p[ackage] [-d][-f][-m][-u][-v][-x][-z][-M AndroidManifest.xml] \
        [-0 extension [-0 extension ...]] [-g tolerance] [-j jarfile] \
        [--debug-mode] [--min-sdk-version VAL] [--target-sdk-version VAL] \
        [--app-version VAL] [--app-version-name TEXT] [--custom-package VAL] \
        [--rename-manifest-package PACKAGE] \
        [--rename-instrumentation-target-package PACKAGE] \
        [--utf16] [--auto-add-overlay] \
        [--max-res-version VAL] \
        [-I base-package [-I base-package ...]] \
        [-A asset-source-dir]  [-G class-list-file] [-P public-definitions-file] \
        [-S resource-sources [-S resource-sources ...]] \
        [-F apk-file] [-J R-file-dir] \
        [--product product1,product2,...] \
        [-c CONFIGS] [--preferred-configurations CONFIGS] \
        [raw-files-dir [raw-files-dir] ...] \
        [--output-text-symbols DIR]

   Package the android resources.  It will read assets and resources that are
   supplied with the -M -A -S or raw-files-dir arguments.  The -J -P -F and -R
   options control which files are output.

 aapt r[emove] [-v] file.{zip,jar,apk} file1 [file2 ...]
   Delete specified files from Zip-compatible archive.
```

It handles resources (images, XML files), compiles them, and packages them into the APK, and generates `R.java` to reference resources in the code.



Interaction with the device: commands execution and services invocation



Interaction with the device: commands execution and services invocation

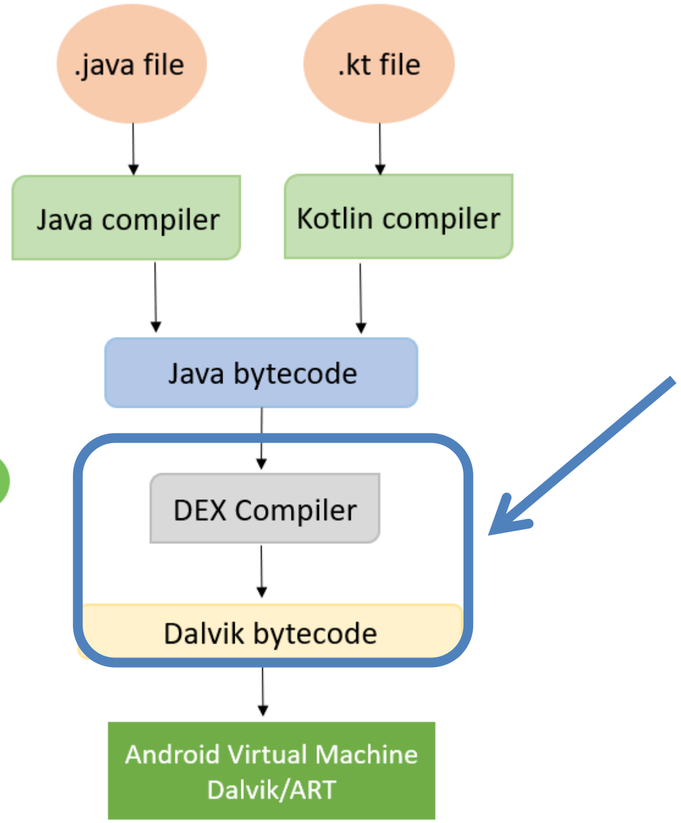
AAPT (Android Asset Packaging Tool)

```
Android Asset Packaging Tool

Usage:
 aapt l[ist] [-v] [-a] file.{zip,jar,apk}
   List contents of Zip-compatible archive.

 aapt d[ump] [--values] WHAT file.{apk} [asset [asset ...]]
   badging          Print the label and icon for the app declared in APK.
   permissions      Print the permissions from the APK.
   resources        Print the resource table from the APK.
   configurations   Print the configurations in the APK.
   xmltree          Print the compiled xmls in the given assets.
   xmlstrings       Print the strings of the given compiled xml assets.

 aapt p[ackage] [-d][-f][-m][-u][-v][-x][-z][-M AndroidManifest.xml] \
        [-0 extension [-0 extension ...]] [-g tolerance] [-j jarfile] \
        [--debug-mode] [--min-sdk-version VAL] [--target-sdk-version VAL] \
        [--app-version VAL] [--app-version-name TEXT] [--custom-package VAL] \
        [--rename-manifest-package PACKAGE] \
        [--rename-instrumentation-target-package PACKAGE] \
        [--utf16] [--auto-add-overlay] \
        [--max-res-version VAL] \
        [-I base-package [-I base-package ...]] \
        [-A asset-source-dir]  [-G class-list-file] [-P public-definitions-file] \
        [-S resource-sources [-S resource-sources ...]] \
        [-F apk-file] [-J R-file-dir] \
        [--product product1,product2,...] \
        [-c CONFIGS] [--preferred-configurations CONFIGS] \
        [raw-files-dir [raw-files-dir] ...] \
        [--output-text-symbols DIR]

   Package the android resources.  It will read assets and resources that are
   supplied with the -M -A -S or raw-files-dir arguments.  The -J -P -F and -R
   options control which files are output.

 aapt r[emove] [-v] file.{zip,jar,apk} file1 [file2 ...]
   Delete specified files from Zip-compatible archive.
```

Although AAPT is often run automatically during the build process in a IDE like Android Studio, developers may use AAPT directly from the **command line** to perform tasks such as:

- **Resource Compilation:** Converts resources (images, XML files) into a format that can be packaged into the APK file.
- **Unpacking and Exploring APKs**: Extracts and examines the contents of an APK file, which is useful for debugging and verifying the packaged resources.
- **Generating R.java** which contains references to the app's resources.
- **Testing and Validation** of application resources before final compilation

DX (Dalvik Executable)



Is a tool in the Android SDK that converts Java bytecode (.class) into a format executable on Android, known as Dalvik bytecode (.dex) → used by the Android runtime (Dalvik VM or ART in newer versions) to execute the code.

The main functions of DX are:

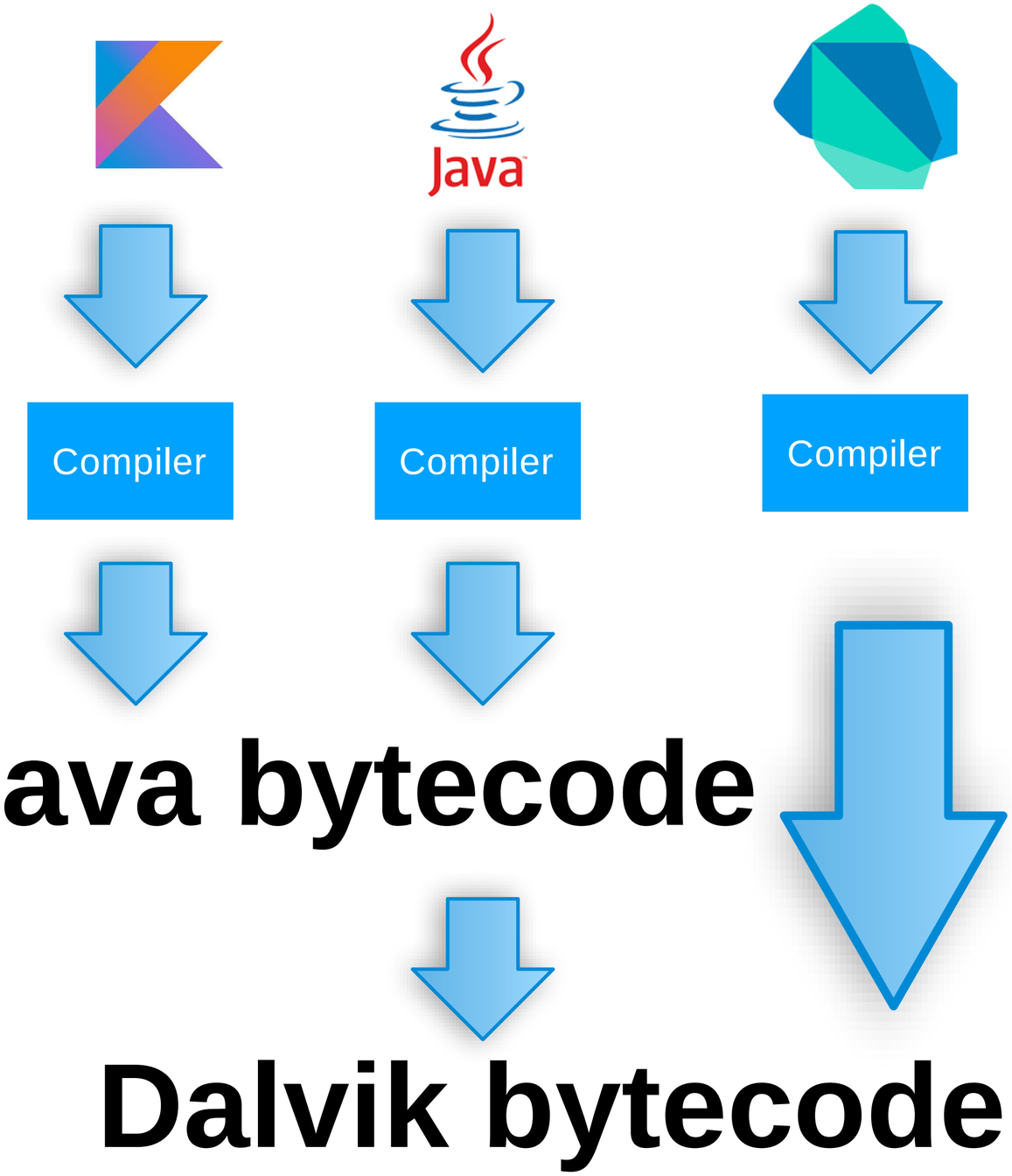
- **Bytecode Conversion: Converts Java bytecode generated by the compiler into Dalvik bytecode.**
- Optimization: Performs optimizations on the code to enhance performance on Android devices.

The resulting .dex file is packaged with the application's resources into the APK for distribution and installation on Android devices.

HOW IS IT POSSIBLE TO EXECUTE A JVM-BASED LANGUAGE IN A MOBILE DEVICE?

JVM LANGUAGES

Source code of a JVM-based language is compiled to java bytecode



APKS GENERATION

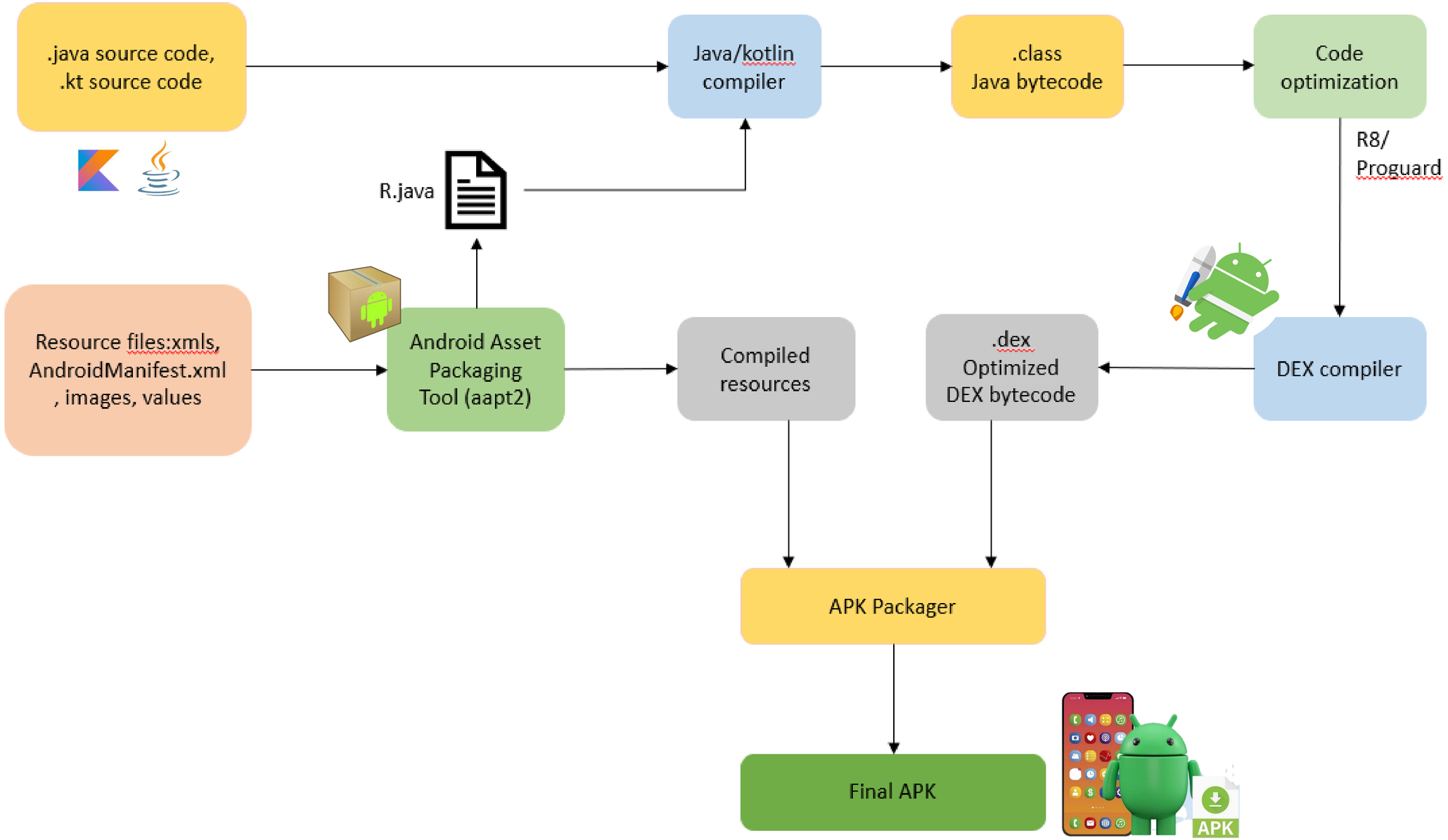
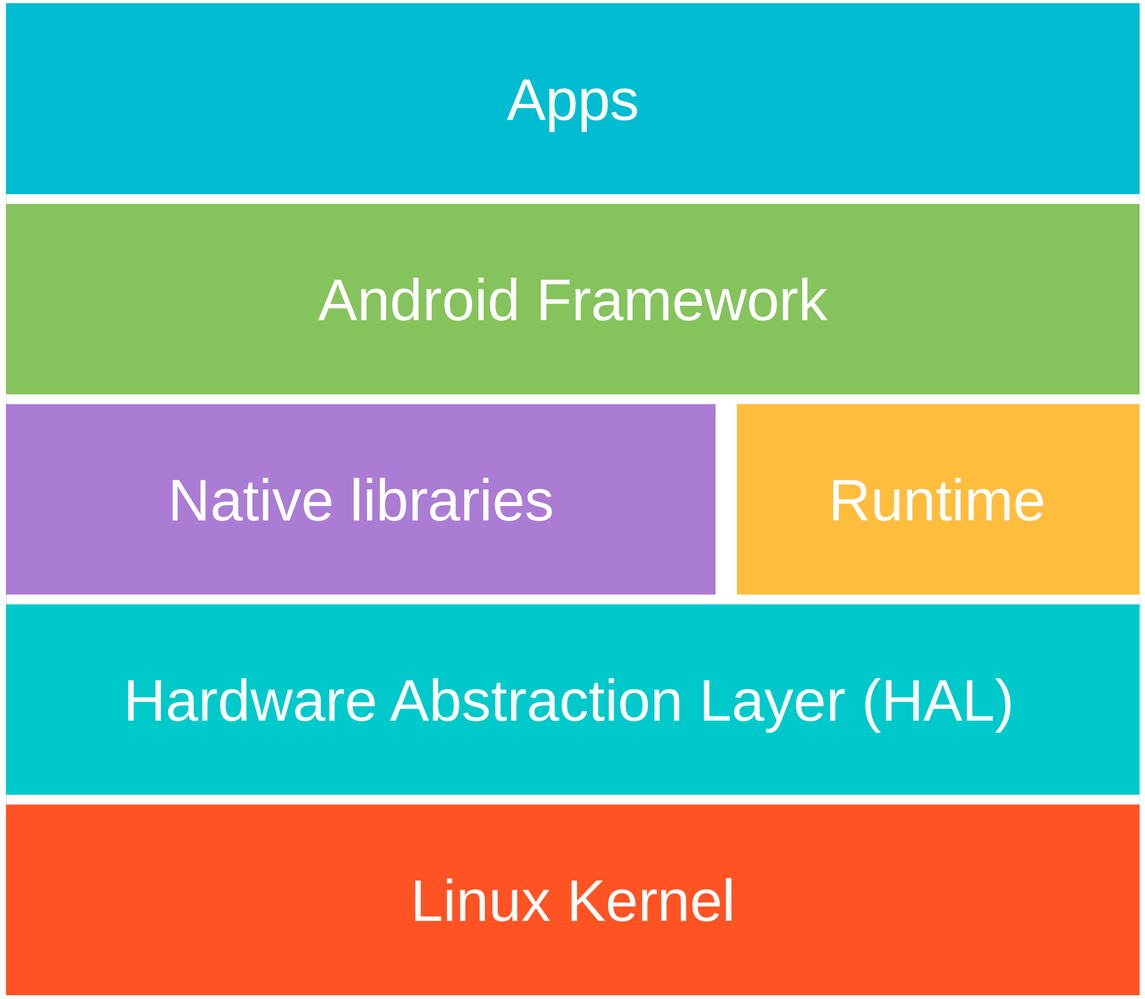


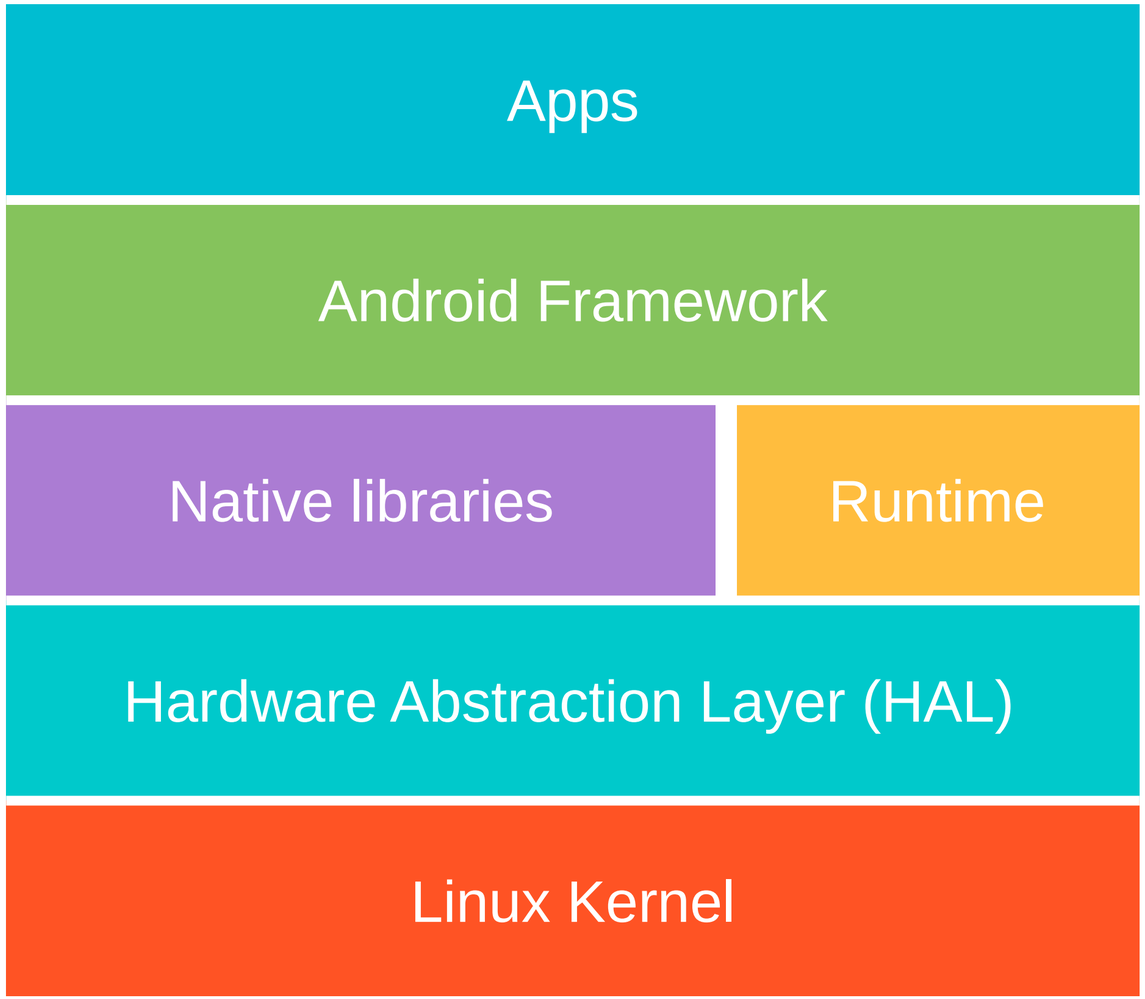
Image taken from https://me-abhishek92.medium.com/deep-dive-into-android-build-process-4282f2af6465

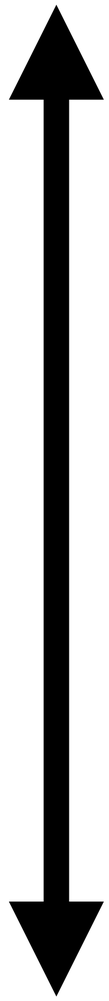
THE ANDROID OS STACK



https://developer.android.com/guide/platform/index.html

THE ANDROID OS STACK



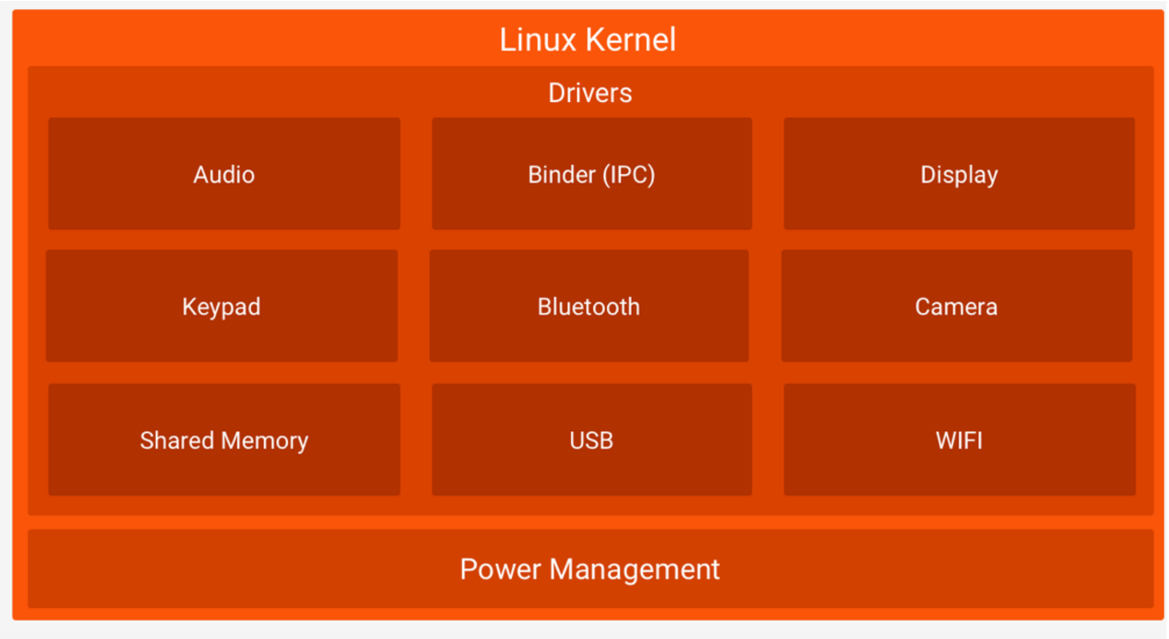


USER

HARDWARE

https://developer.android.com/guide/platform/index.html

THE ANDROID OS: KERNE



Provides services such as threading, memory management, networking, processes, security

It is the core of the operating system

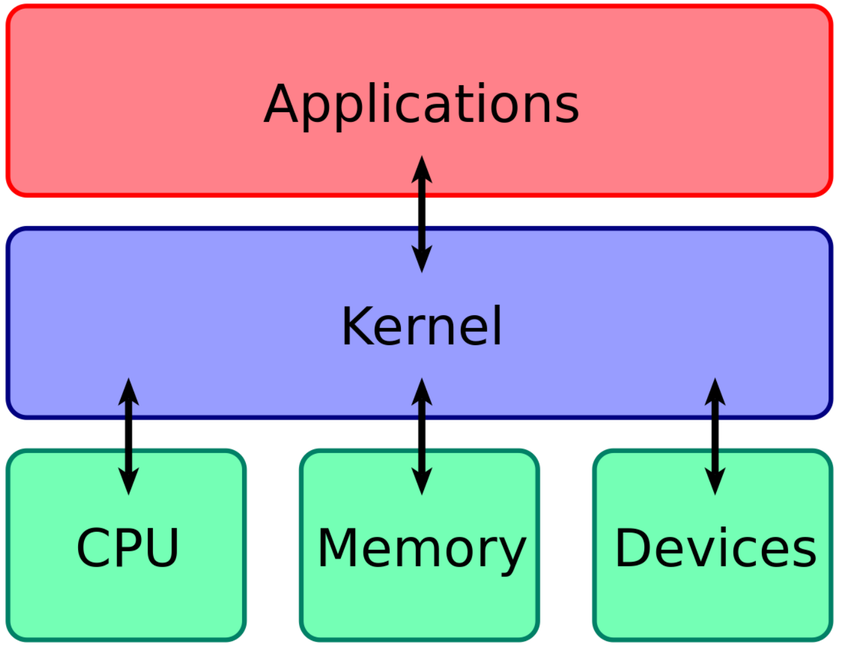
The linux kernel is a monolithic one

https://developer.android.com/guide/platform/index.html

KERNEL???

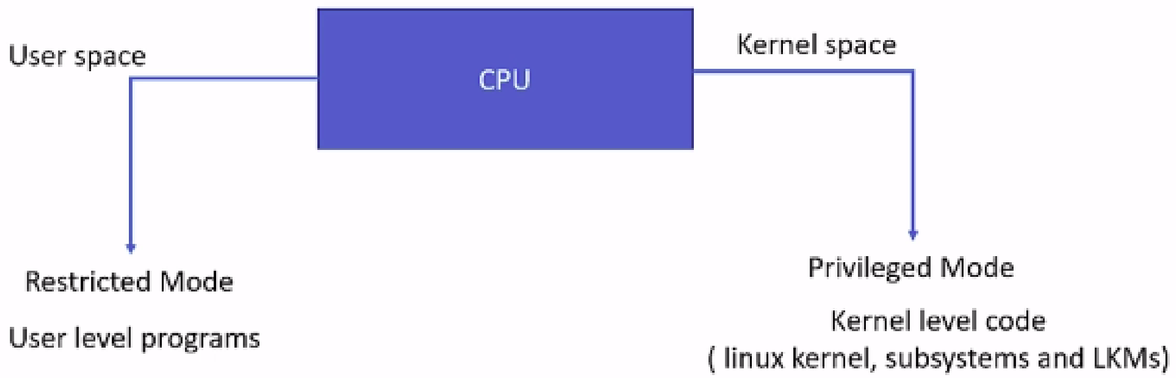
The kernel is a fundamental part of the operating system:

- acts as an **intermediary between** the system **hardware** and the **software**.
- its main function is to **manage system resources**, such as the CPU, memory, and input/output devices
- provide an **interface** for applications **to interact with** the **hardware** without having to manage these details directly.



KERNEL

User space Vs Kernel Space



Kernel Space: This is where the operating system kernel runs. Code in this space has full access to all hardware resources and the entire system memory.

User Space: This is where applications and user programs run. This space is separated from the kernel space to protect the operating system and other programs from errors or malicious behavior in the applications.

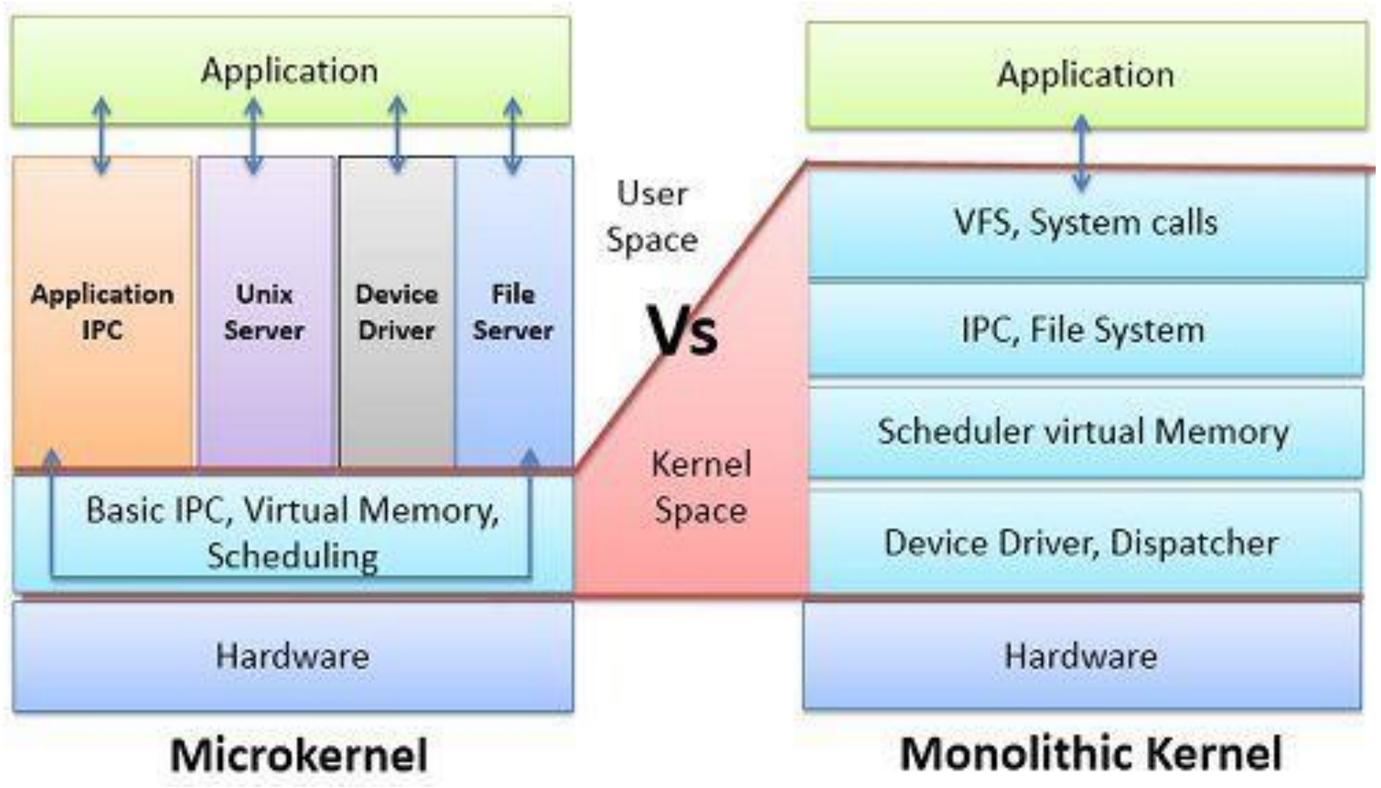
MICROKERNEL MONOLITIC

Only essential functions run in kernel mode (e.g., IPC). Other components, such as device drivers and the file system, run in user mode.

The entire operating system (including process, memory, and device management) runs in kernel mode as a single block of code

Advantages: Modularity and maintainability (kernel has less code).

Disadvantages: Lower performance due to the overhead of communication between the kernel and user-mode services.

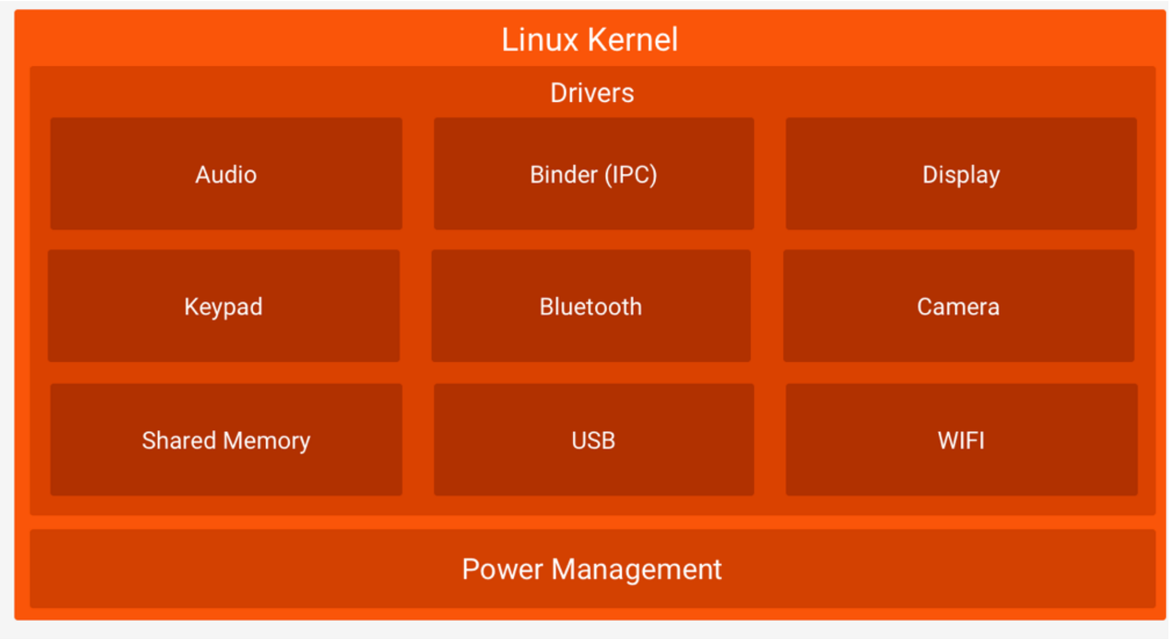


Advantages: High efficiency and fast performance due to the close integration of its components.

Disadvantages: It can be more difficult to maintain and extend, as any error in the kernel code can potentially affect the entire system.

https://techdifferences.com/difference-between-microkernel-and-monolithic-kernel.html

THE ANDROID OS: KERNE



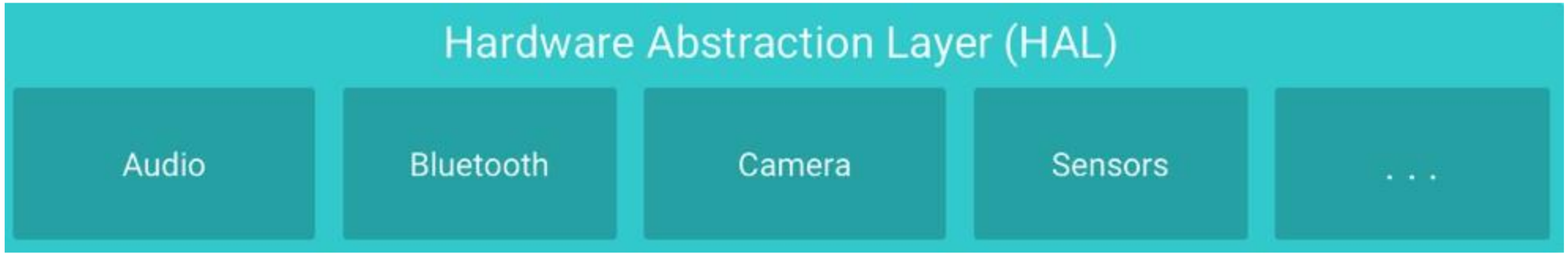
Google modified the linux kernel. Examples of the features added by Google to the kernel are:

- **Alarm timers**
- **Ashmem (Android Shared Memory)**
- **Binder (Inter Process Communication)**
- **Power management (wakelocks)**
- **Low Memory Killer (Viking Killer)**
- **Kernel Debugger**
- **Logger**

http://elinux.org/Android_Kernel_Features

https://developer.android.com/guide/platform/index.html

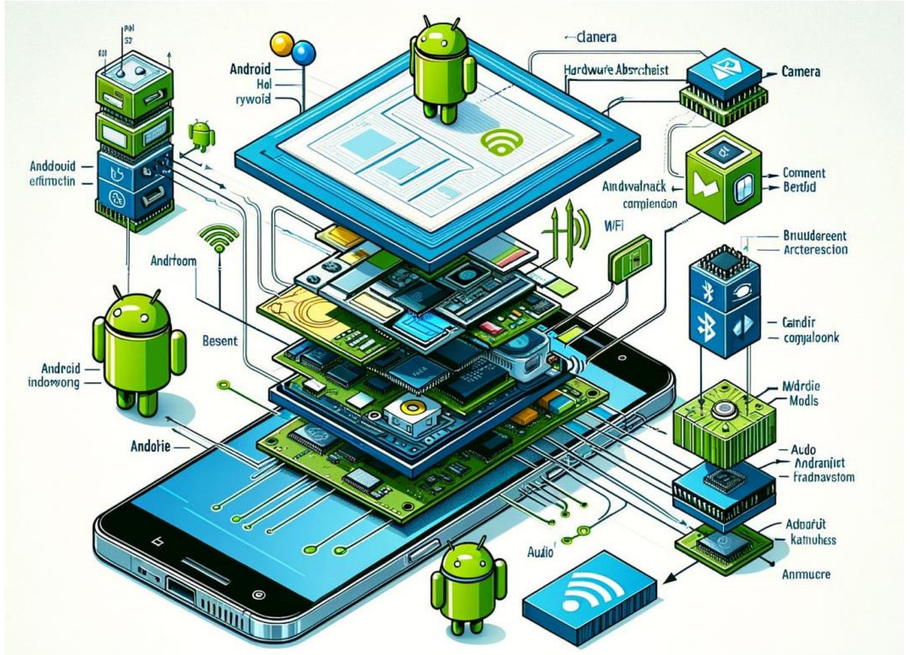
THE ANDROID OS: HAL



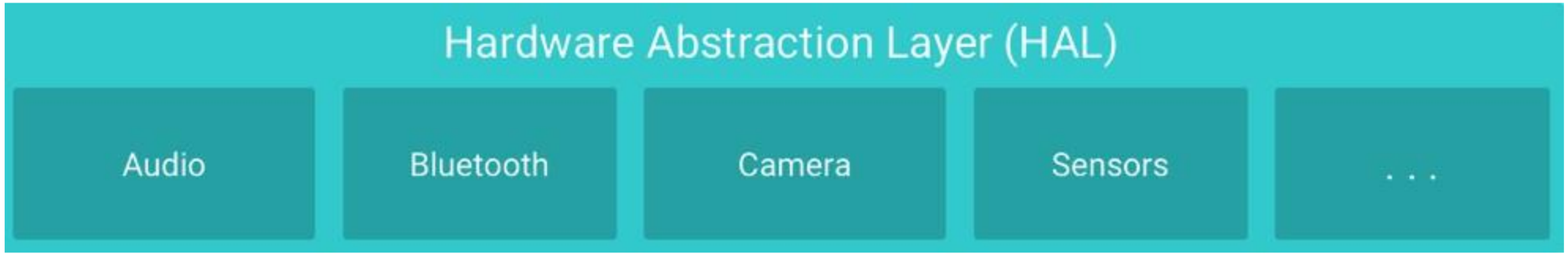
https://developer.android.com/guide/platform/index.html

Abstraction layer between the kernel and the rest of layers.

The HAL is a set of interfaces that OEMs (Original Equipment Manufacturer) must implement for each hardware components



THE ANDROID OS: HAL



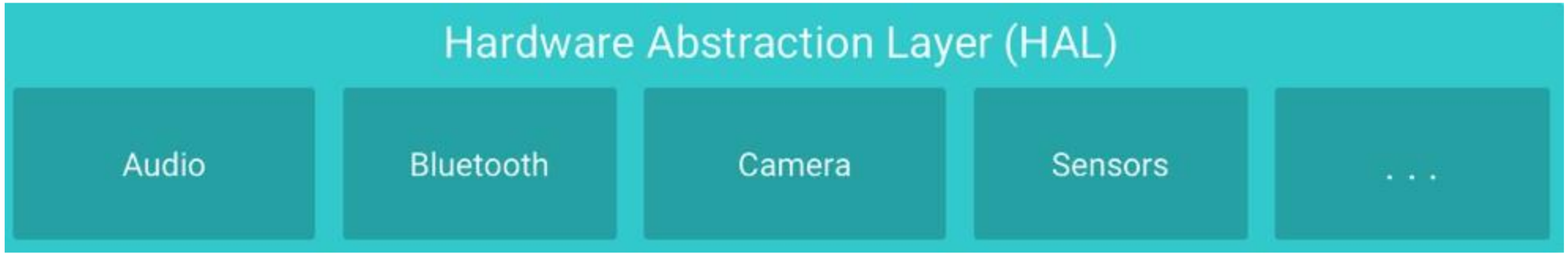
https://developer.android.com/guide/platform/index.html

Abstraction layer between the kernel and the rest of layers.

The HAL is a set of interfaces that OEMs must implement for each hardware components

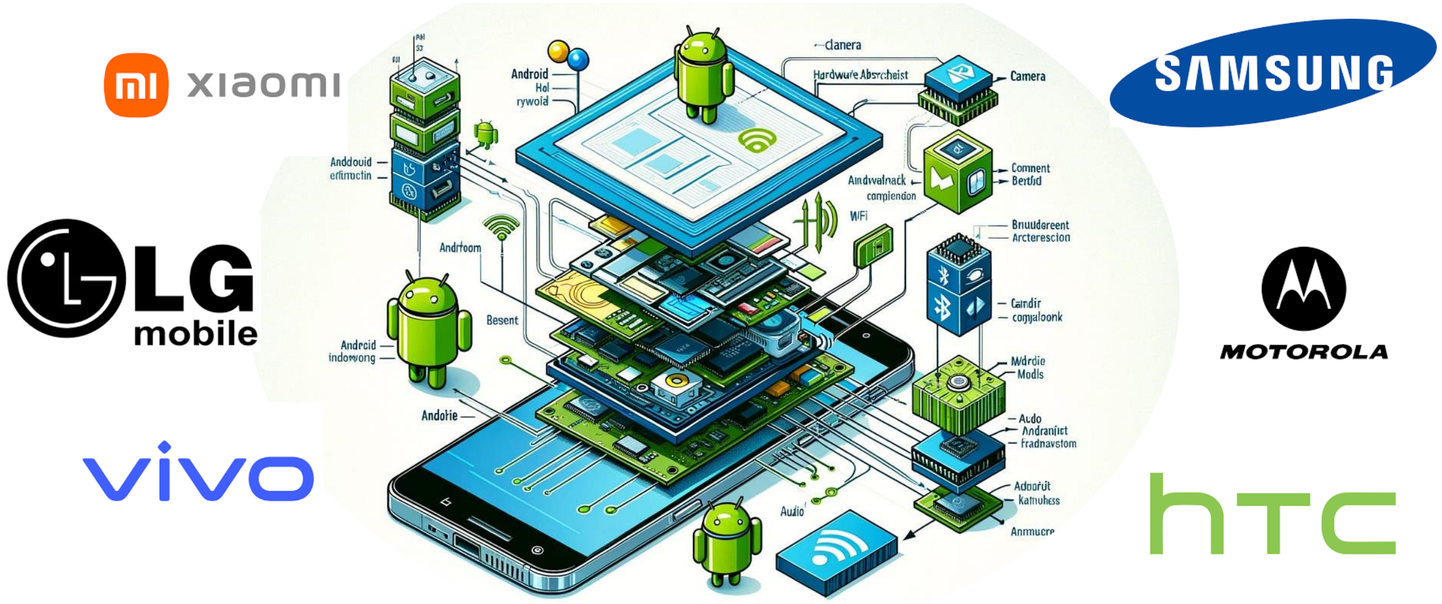
WHY DOES THE ANDROID OS NEED THE HAL?

THE ANDROID OS: HAL

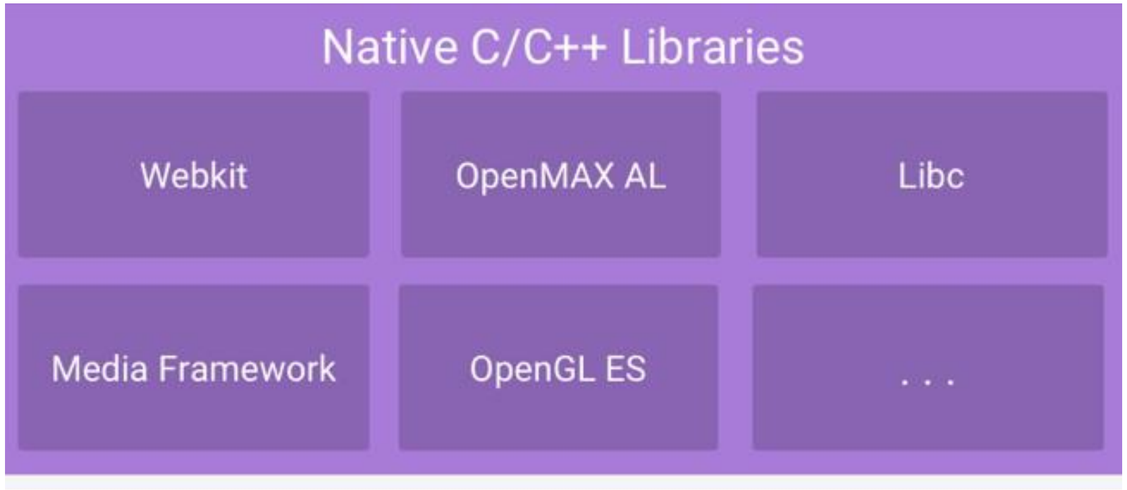


https://developer.android.com/guide/platform/index.html

The HAL is a set of interfaces that **OEMs (Original Equipment Manufacturer)** must implement for each hardware components



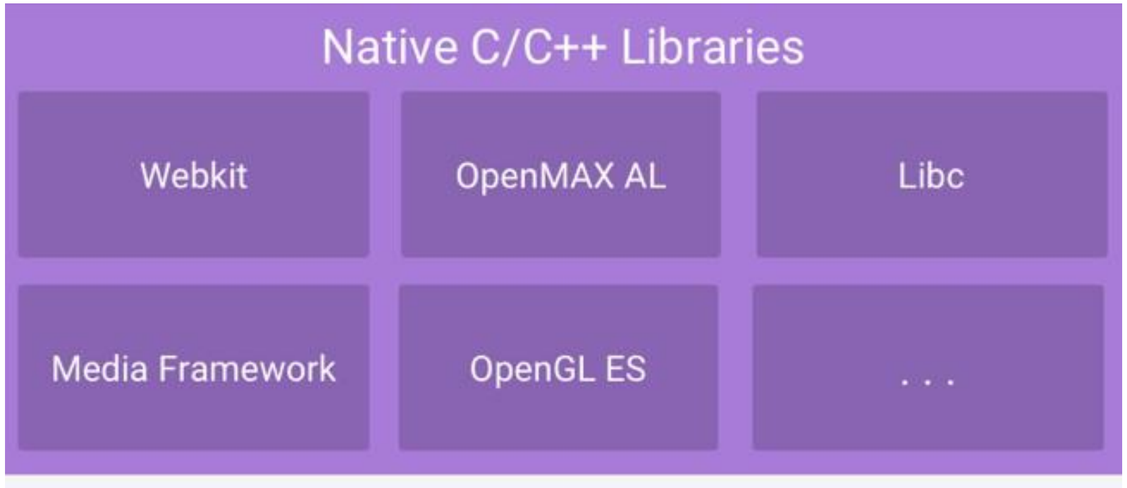
THE ANDROID OS: NATIVE LIB



Libraries written in C/C++ that interact directly with the kernel (via the HAL) and provide core functionalities to the Android framework.

https://developer.android.com/guide/platform/index.html

THE ANDROID OS: NATIVE LIB



https://developer.android.com/guide/platform/index.html

Libraries written in C/C++ that interact directly with the kernel (via the HAL) and provides core functionalities to the Android framework.

Bionic: implementation of libc for embedded systems

WebKit: render engine for HTML, CSS, etc.

Media framework (Stagefright): supports multimedia features

THE ANDROID OS: RUNTIME

Provides the execution environment for the Android apps, i.e., the virtual machine (ART) and core libraries required to execute apps.

Android versions previous to Android 5.0 Lollipop had the Dalvik Virtual Machine (DVM)

https://developer.android.com/guide/platform/index.html

IS THE ANDROID VIRTUAL MACHINE REGISTER-BASED OR STACK-BASED?

```
c = a + b
```

Stack-based bytecode

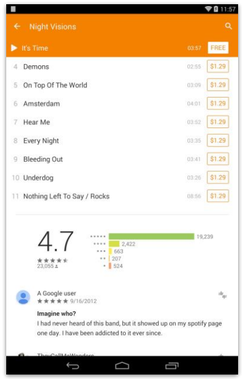
```
iload_1
iload_2
iadd
istore_3
```

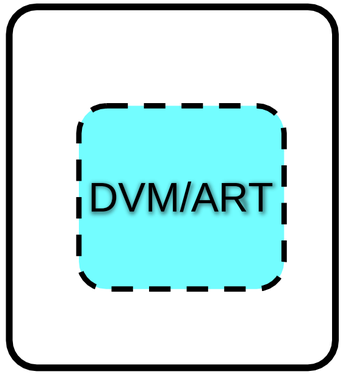
Register-based bytecode

```
move r10, r1
move r11, r2
iadd r10, r10, r11
move r3, r10
```

* r1, r2, and r3 are local variables

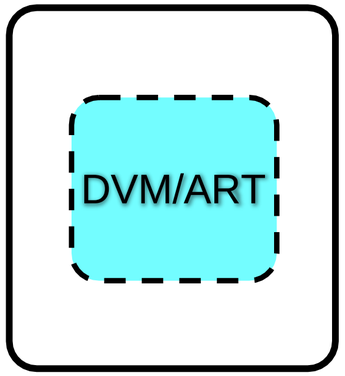
APK EXECUTION





App 1

Linux process



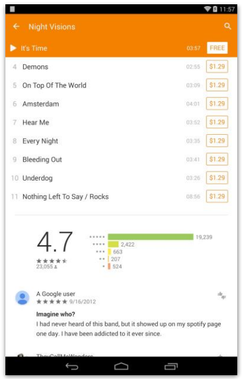
App 2

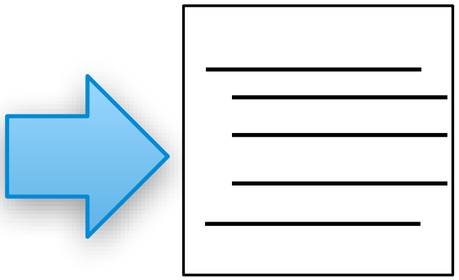
Linux process

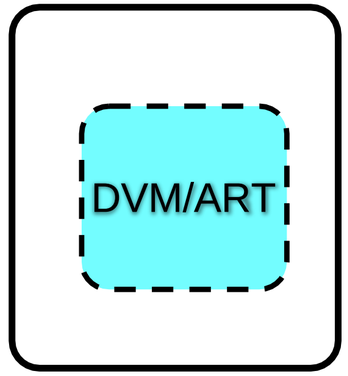
Each app runs on a VM, and each VM is a linux process

Each app runs on a VM, and each VM is a linux process

APK EXECUTION



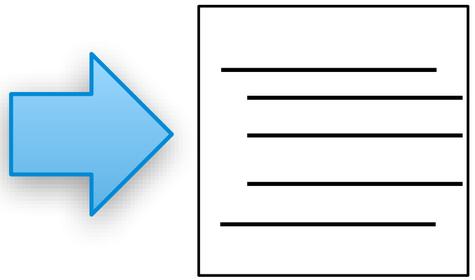


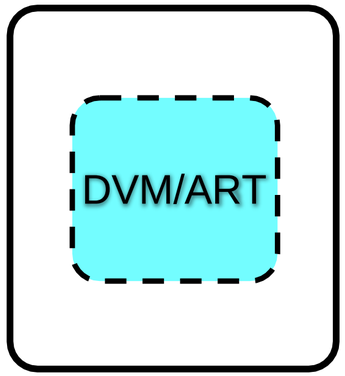


App 1

Dalvik Executable (DEX)-based files

Linux process



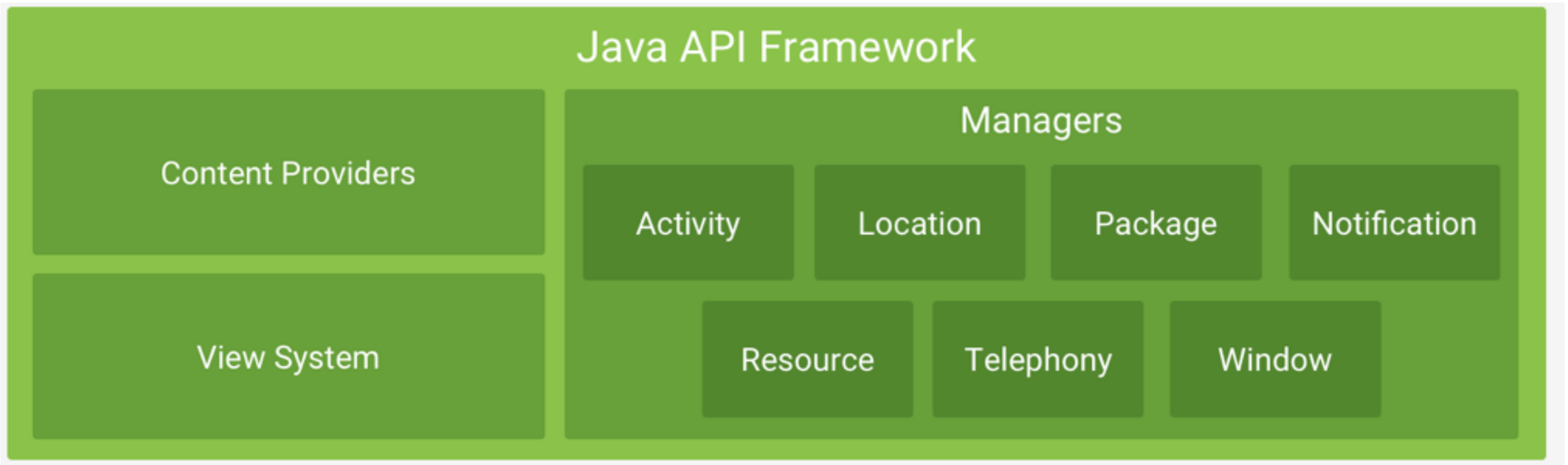


App 2

Linux process

This enhances the security and stability of the Android operating system by ensuring that each application runs in a controlled and isolated environment, which guarantees that applications are separated, and their execution does not affect other applications or the OS

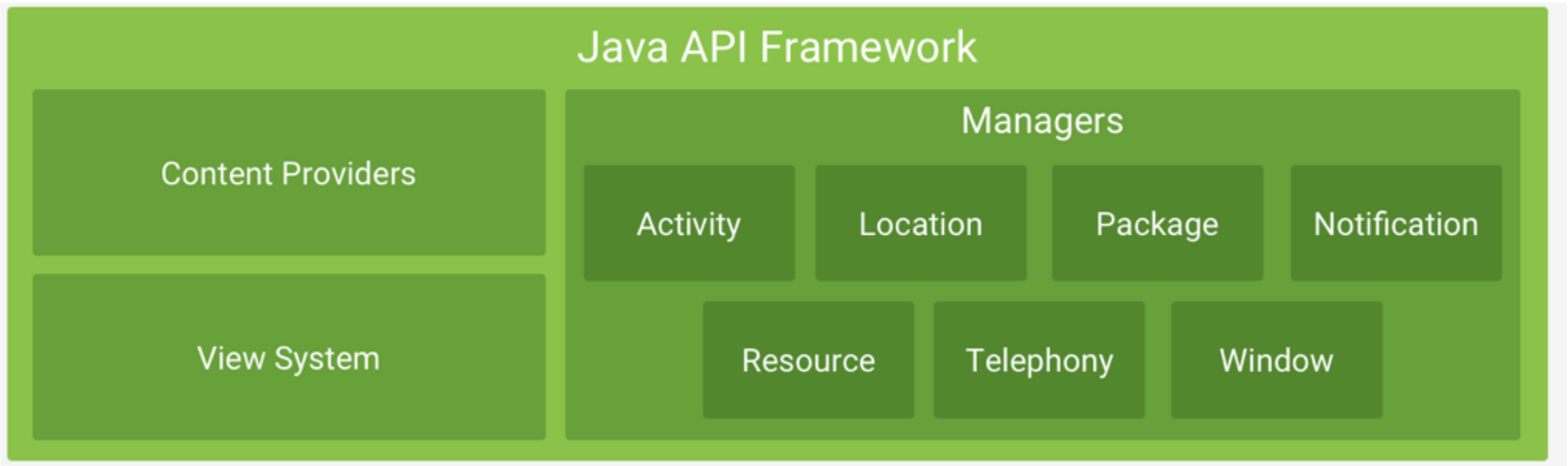
THE ANDROID OS: FRAMEWORK



This layer includes the Android API(android.*) and the services (managers) used by the apps

The services are exposed as OS utilities that can be invoked remotely (via adb) or via API invocations

THE ANDROID OS: FRAMEWORK

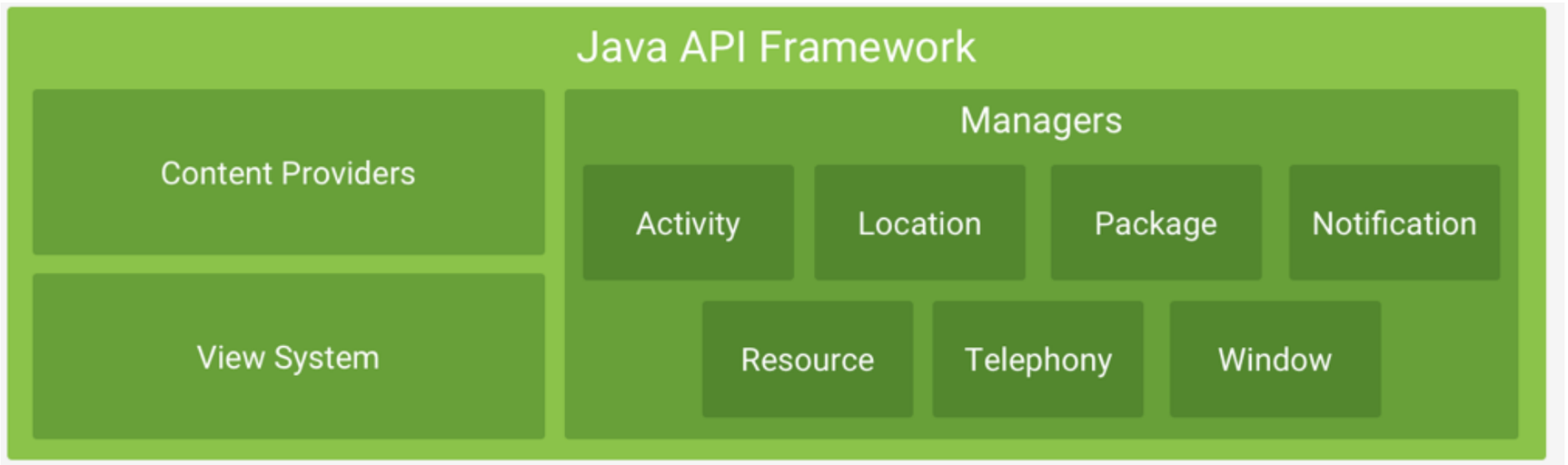


Content providers: encapsulate data access to local data from other apps (e.g., contacts)

Activity manager: activities life cycle and stack

Notification manager: allow apps to paint custom alerts in the status bar

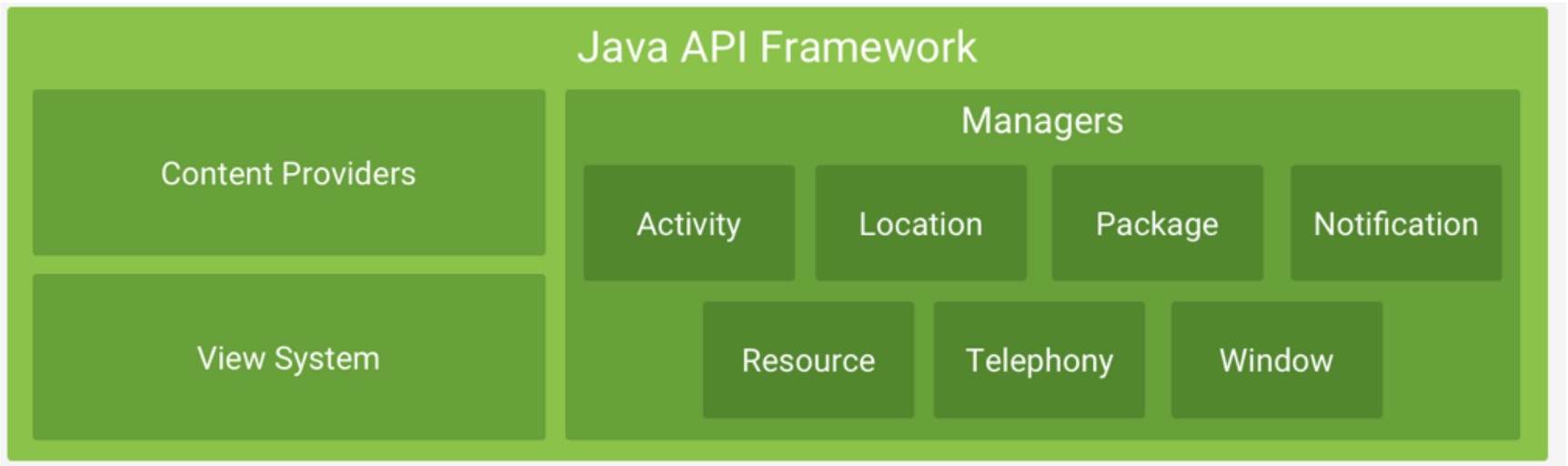
THE ANDROID OS: FRAMEWORK



View system: API for UI components and events

Windows manager: it is responsible for managing the windows order (z-order), layout and visibility, transitions/ animation when opening/closing apps

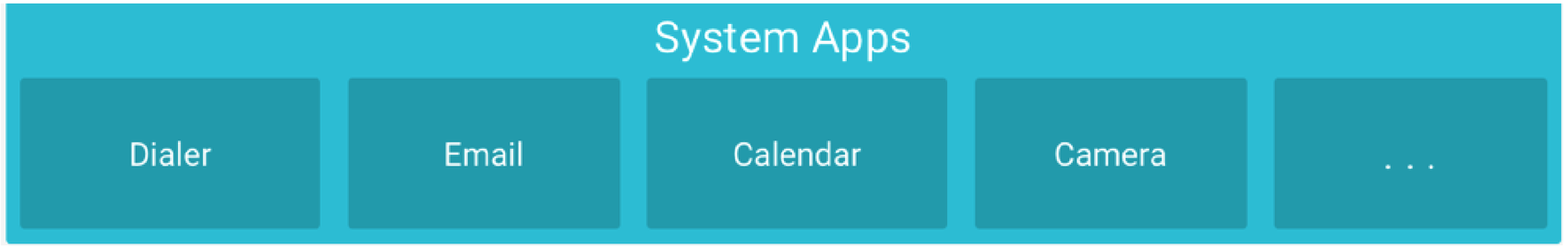
THE ANDROID OS: FRAMEWORK



Package manager: provides methods for manipulating and querying the packages installed on a device

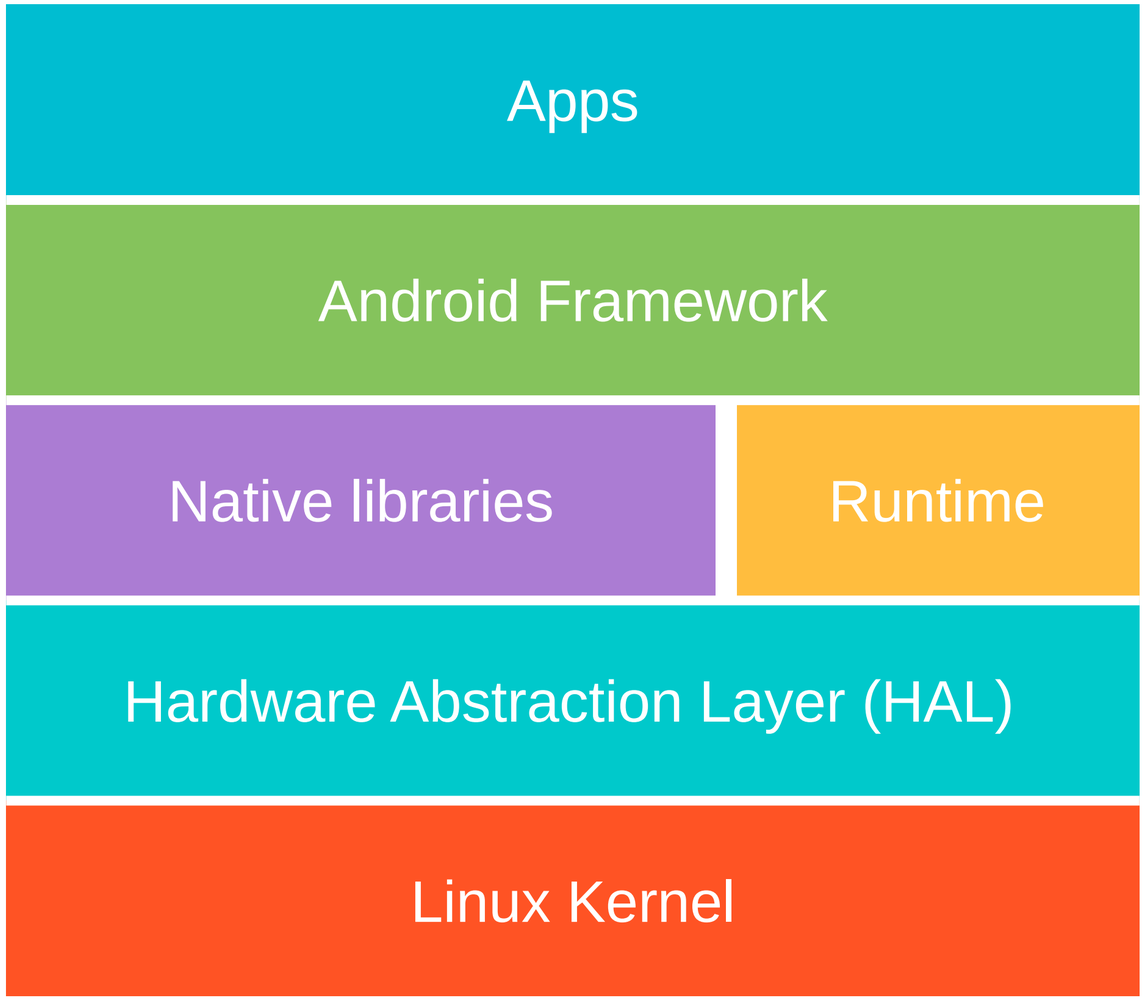
Resource manager: allows apps to access non-code resources (e.g., strings, images, layout files, etc)

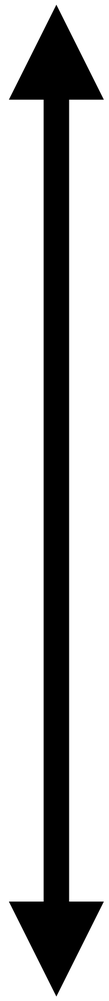
THE ANDROID OS: APPS



This is the stack layer closest to the user. It includes the apps pre-deployed with the OS, and the apps created by users.

THE ANDROID OS STACK



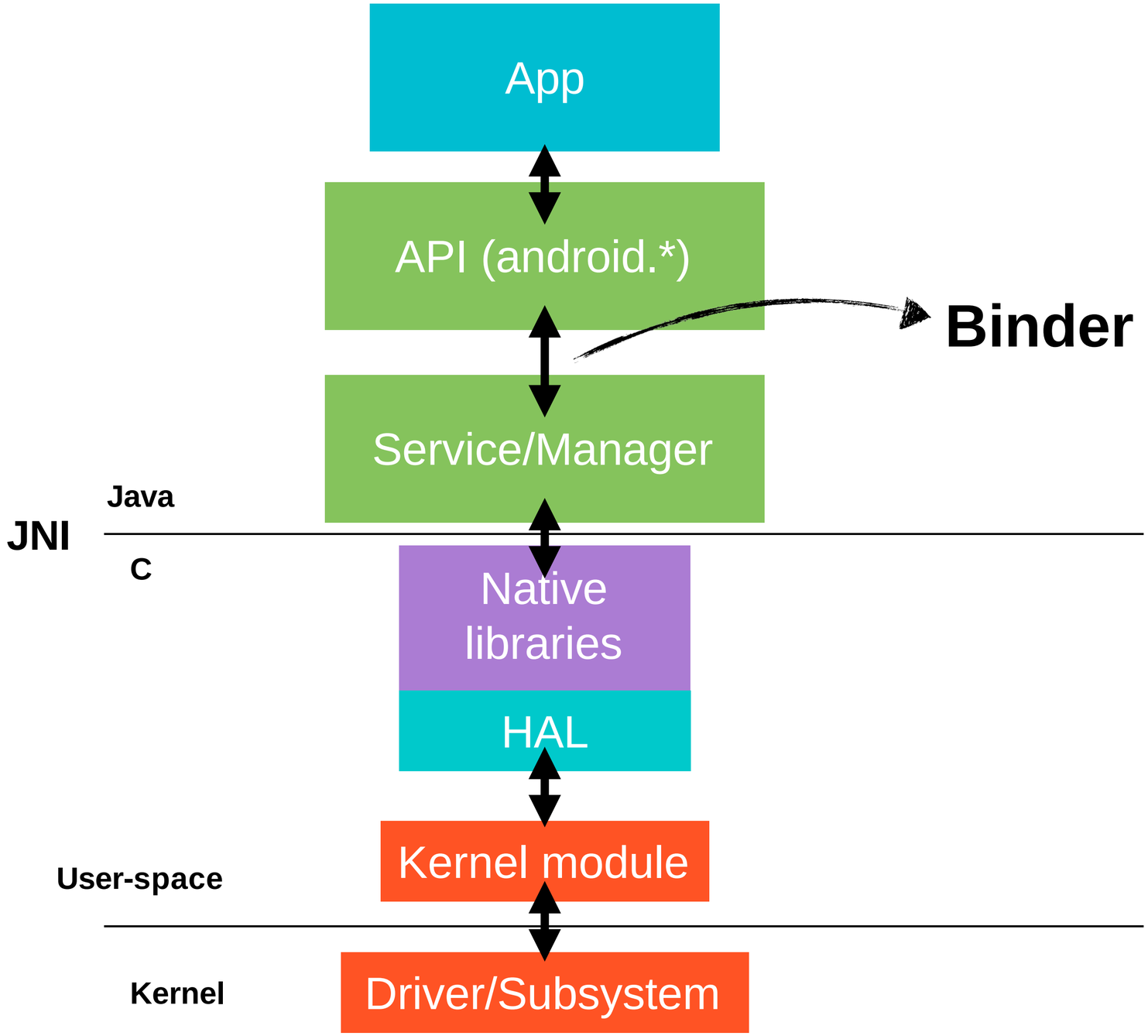


USER

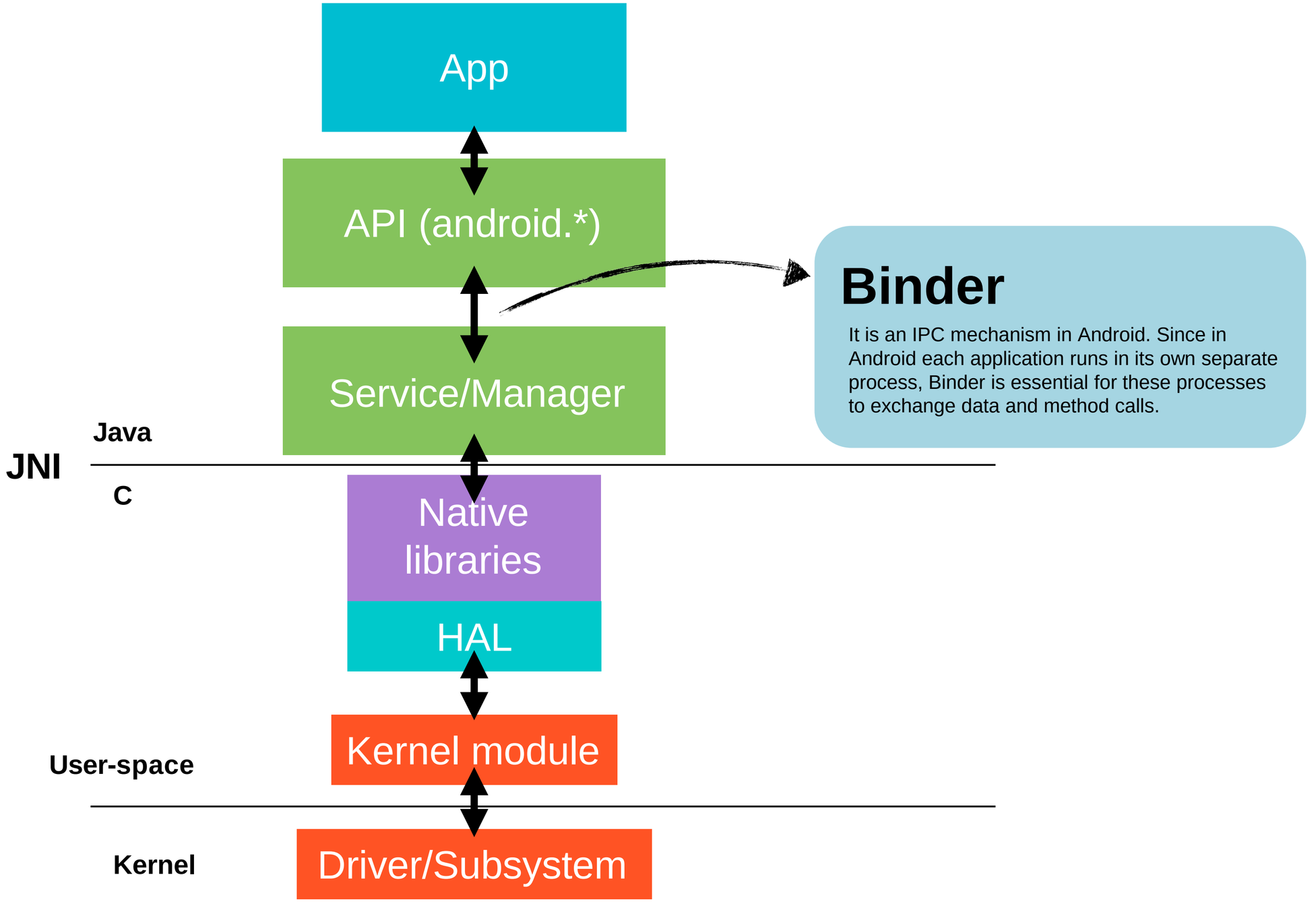
HARDWARE

https://developer.android.com/guide/platform/index.html

THE ANDROID OS STACK



THE ANDROID OS STACK



WHAT IS THE DIFFERENCE BETWEEN USER AND KERNEL SPACE?

SER AND KERNEL SPAC

Virtual memory (RAM) in an OS is separated into **kernel space** and **user space,** because of memory and hardware protection purposes

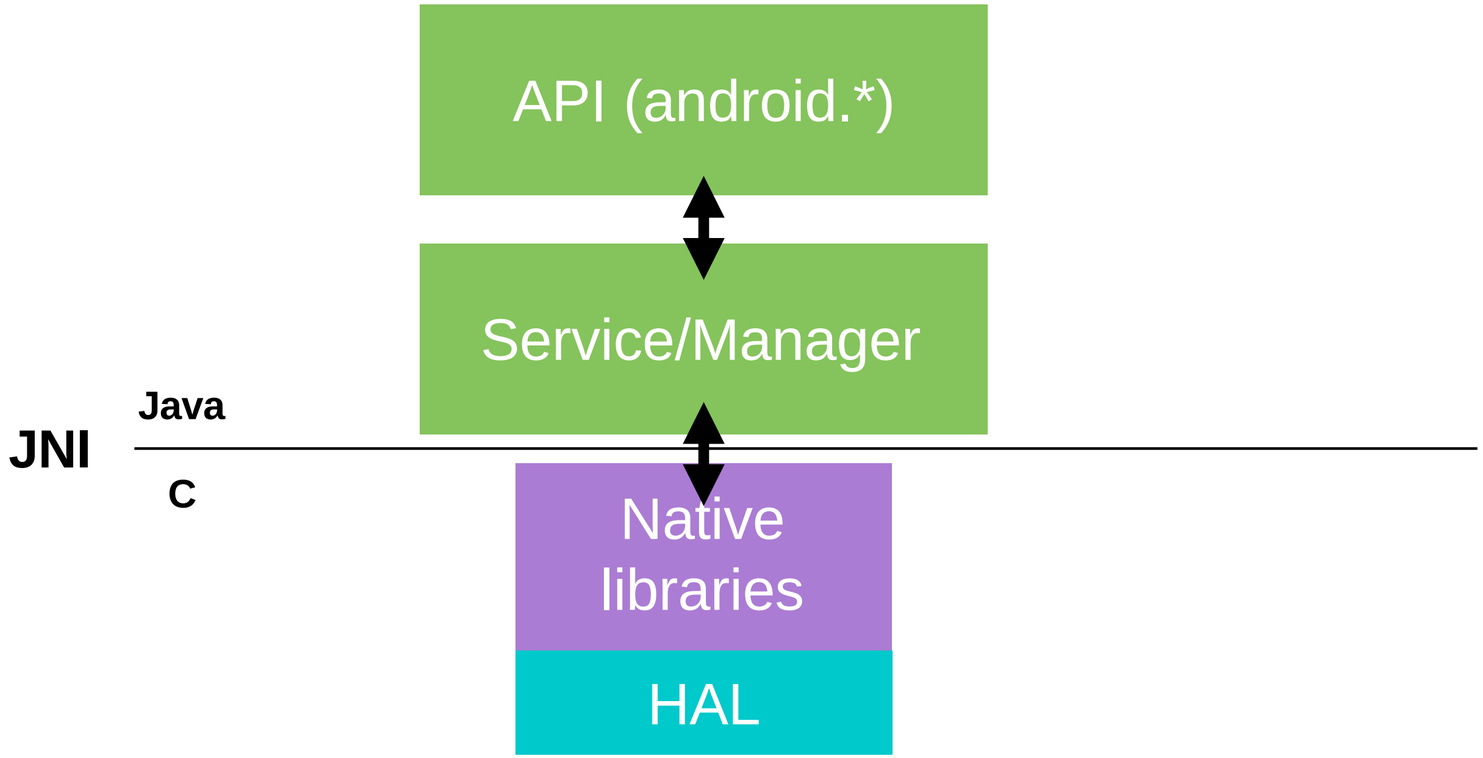
Kernel space: reserved for the kernel, drivers, and extensions; full access to hardware and memory

User space: code that runs outside the kernel; each process runs on its own memory space, and can not access memory of other processes

WHAT IS JNI?

WHAT IS THE BINDER?

JNI (Java Native Interface): enables java code running in a VM to call and be called by applications/libraries written in C/C++.



JNI is a part of the virtual machine (DVM/ART)

Because Android apps and processes run on **user space**, data exchange requires IPC mechanisms.

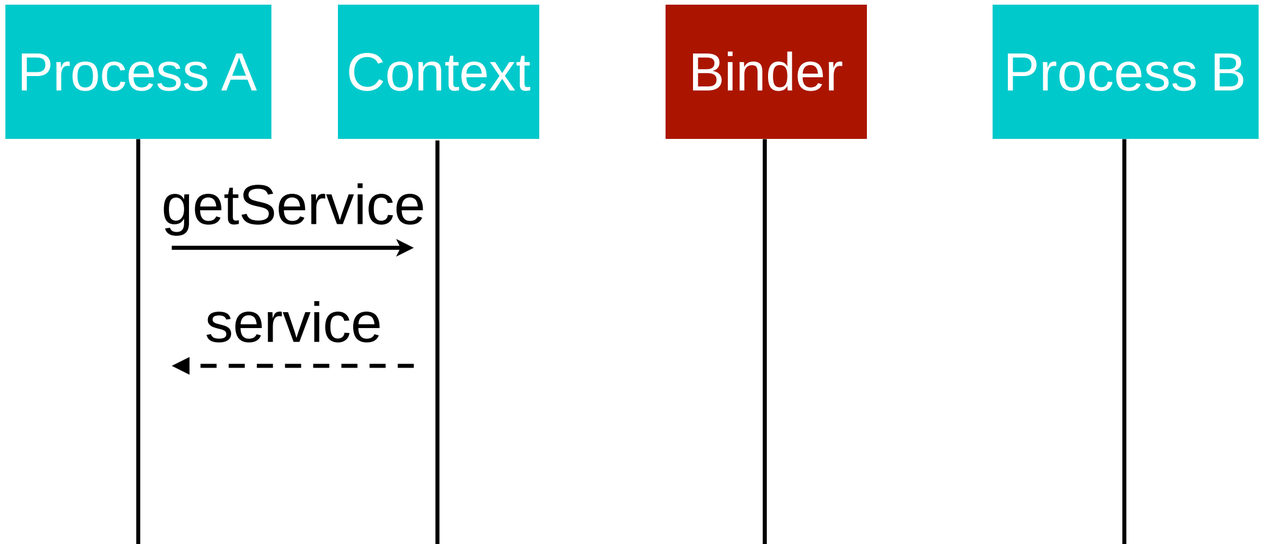
The binder allows for invocation of methods on remote objects as if they were local objects methods.

Because Android apps and processes run on **user space**, data exchange requires IPC mechanisms.

The binder allows for invocation of methods on remote objects as if they were local objects methods.

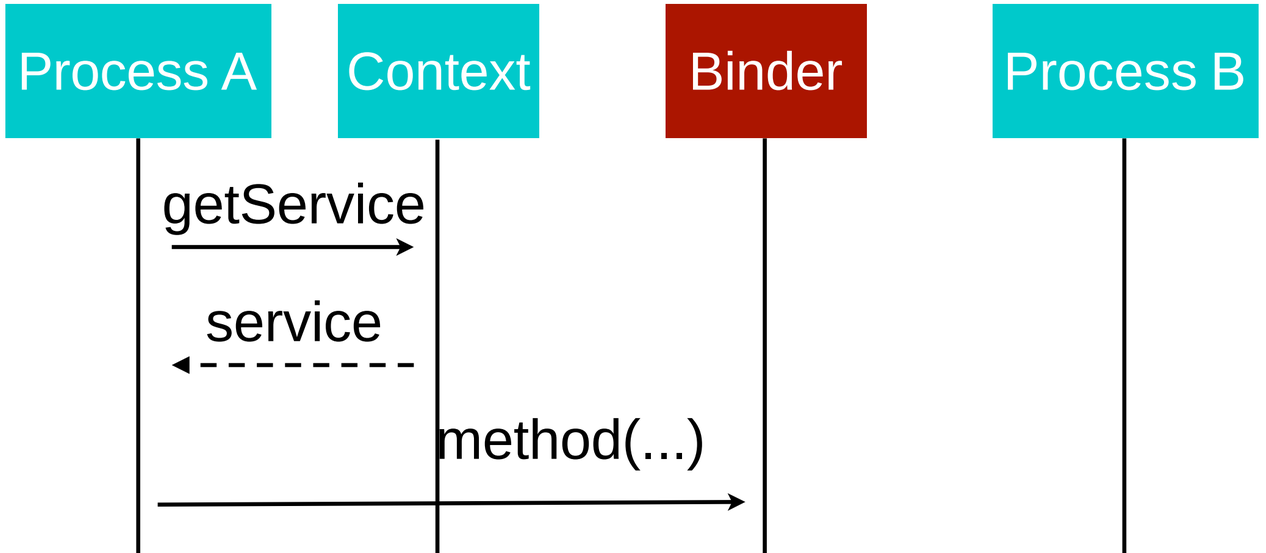
Because Android apps and processes run on **user space**, data exchange requires IPC mechanisms.

The binder allows for invocation of methods on remote objects as if they were local objects methods.



Because Android apps and processes run on **user space**, data exchange requires IPC mechanisms.

The binder allows for invocation of methods on remote objects as if they were local objects methods.



Because Android apps and processes run on **user space**, data exchange requires IPC mechanisms.

The binder allows for invocation of methods on remote objects as if they were local objects methods.