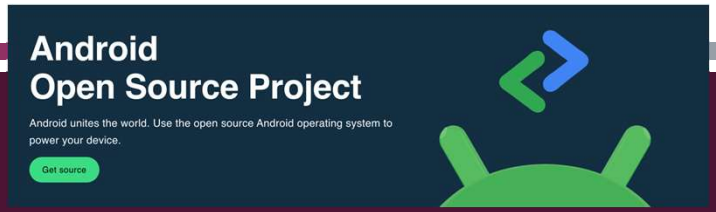
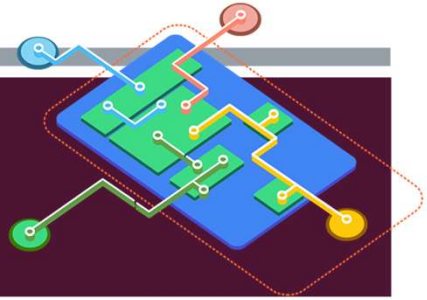
ANDROID SOFTWARE STACK

INTRODUCTION

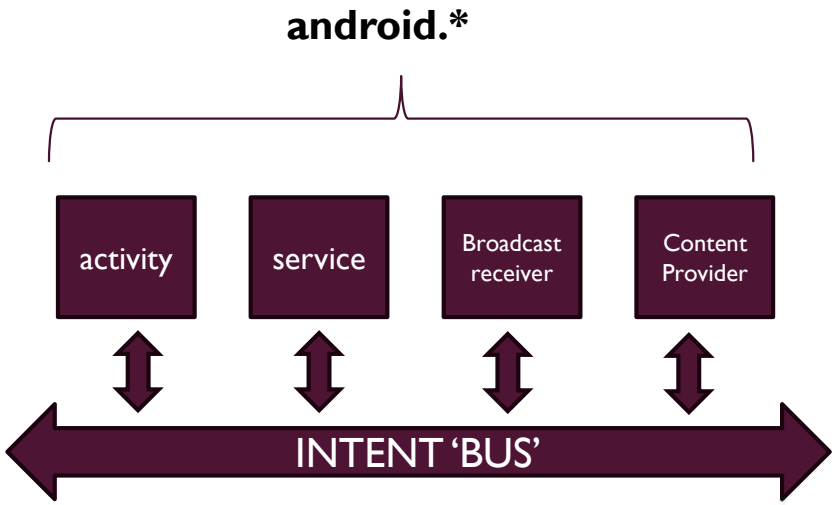




- Android OS is based on a Linux kernel that has been modified to meet the needs of mobile applications. The key modifications include:
- **Memory Management**
  - Low Memory Killer: A mechanism that terminates background processes when memory resources become scarce, ensuring the system remains responsive (this is way components have a lifecycle)
  - Ashmem (Android Shared Memory): A shared memory subsystem designed for efficient inter-process communication with minimal overhead (for example used when an intent has to send a lot of information to another activity)
- **CPU Management**
  - Wake lock: prevents the device from entering sleep mode when specific tasks require the CPU to remain active
- **Communication**
  - Binder: A new inter-process communication (IPC) mechanism that enables efficient and secure communication between different applications and system services, that implements Intent mechanism

APPLICATIONS ARE MADE OF MANAGED COMPONENTS

- **Activity** - Represents a single screen with a user interface
- **Service** - Performs background operations without a user interface
- **BroadcastReceiver** - Receives and handles broadcast messages from the system or other apps
- **ContentProvider** - Manages shared access to app data
- These components are "managed" by the Android system, which means:
- Their **lifecycle** is controlled by the operating system (not by the app itself)
- The system decides when to create, pause, resume, or destroy them
- They must be declared in the **AndroidManifest.xml** file
- The system can terminate them when necessary (e.g., low memory situations)
- They communicate with each other through **Intents** (using Binder as we discussed)



androidx.*

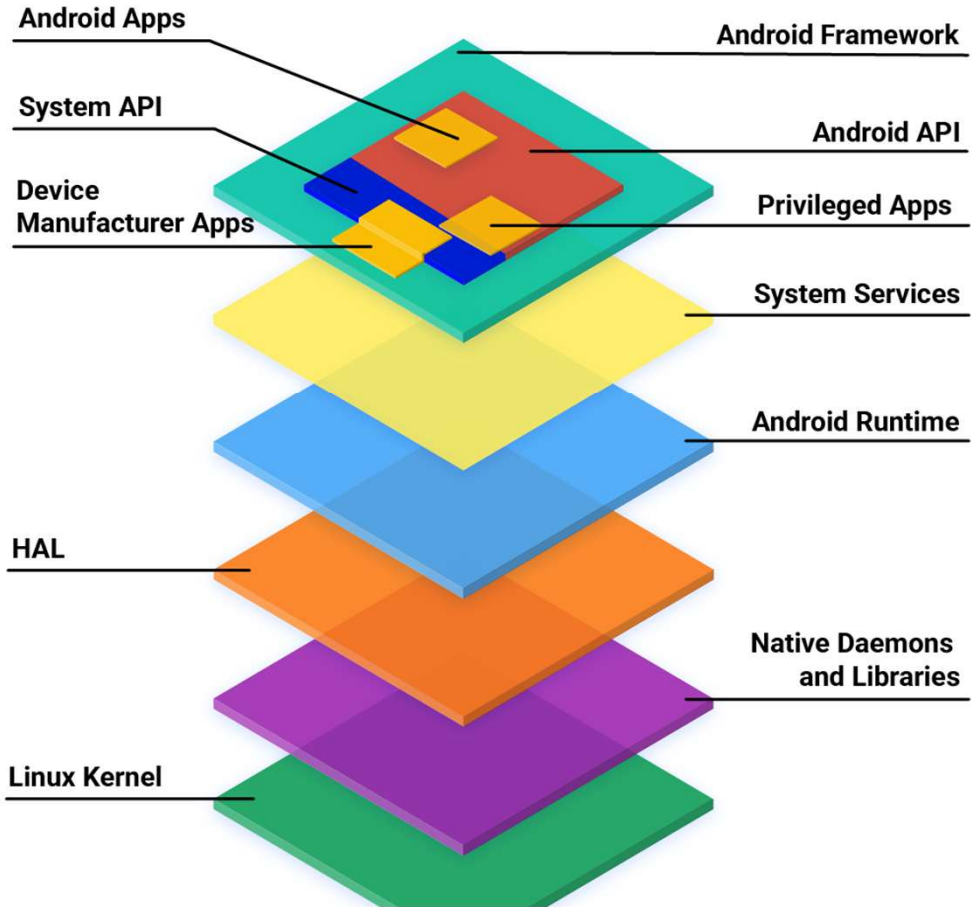
Add new components 'on top' of these

SUPPORTED HW: THE APPLICATION BINARY INTERFACE

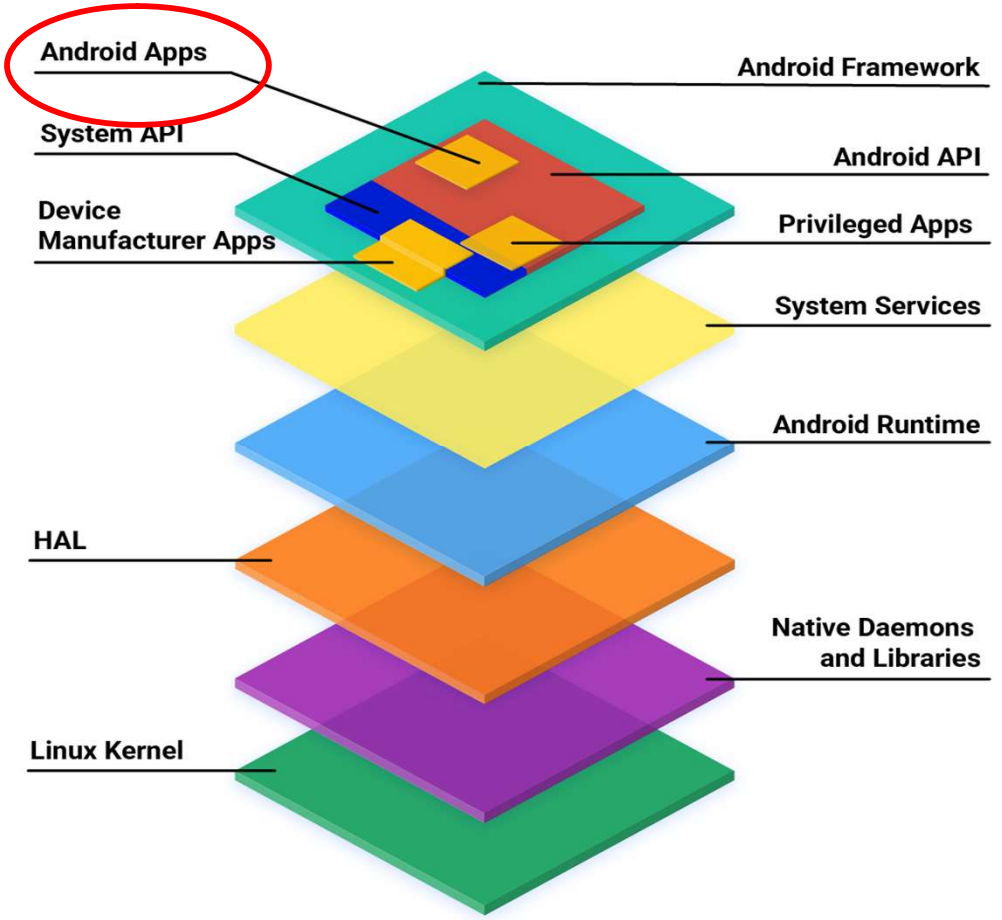
- Different Android devices use different CPUs, which in turn support different instruction sets.
- Each hardware is labelled with a specific Application Binary Interface (**ABI**), that includes
  - The CPU instruction set (and extensions) that can be used.
  - The endianness of memory stores and loads at runtime.
  - Conventions for passing data between applications and the system, including alignment constraints, and how the system uses the stack and registers when it calls functions.
  - How to integrate C++ code

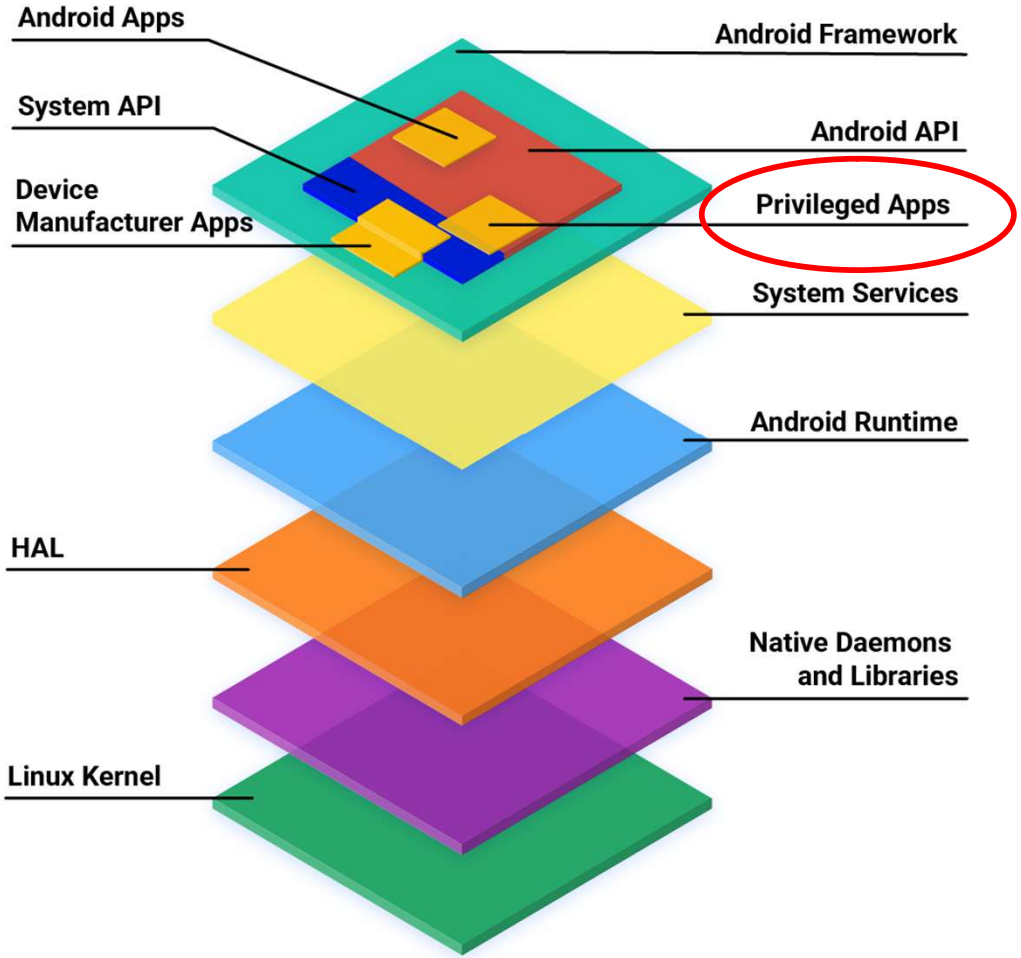
ABI	Supported Instruction Sets	Notes
armeabi-v7a	• armeabi • Thumb-2 • VFPv3-D16	Incompatible with ARMv5/v6 devices.
arm64-v8a	• AArch64	
x86	• x86 (IA-32) • MMX • SSE/2/3 • SSSE3	No support for MOVBE or SSE4.
x86_64	• x86-64 • MMX • SSE/2/3 • SSSE3 • SSE4.1, 4.2 • POPCNT	

- The software stack for AOSP is rather complex and contains the following software **layers**
- We discuss the main layers

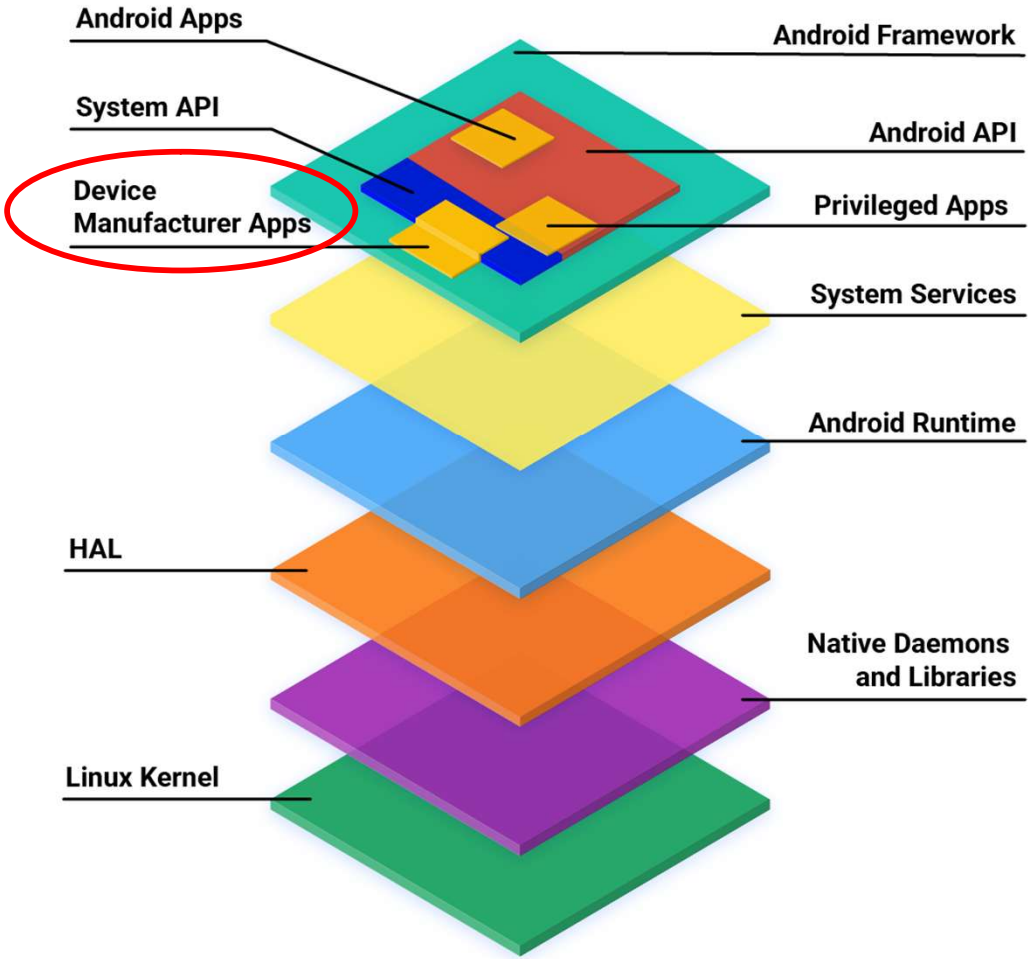


- Any application that a user installs from the Google Play Store or via sideloading (APK).
- Installation: resides in the */data/app* partition, which is readable and writable.
- Sandbox: Each app lives in a "sandbox". An app is a Linux user isolated from other apps and from the system.
- It cannot access data from other apps or change system settings, except through public and well-defined APIs (such as opening the camera with an Intent).
- Permissions: It can request permissions from the user (access to contacts, location, etc.), but can never obtain system-wide permissions (e.g. permission to reboot).
- Uninstallable: The user has total control and can uninstall it at any time.





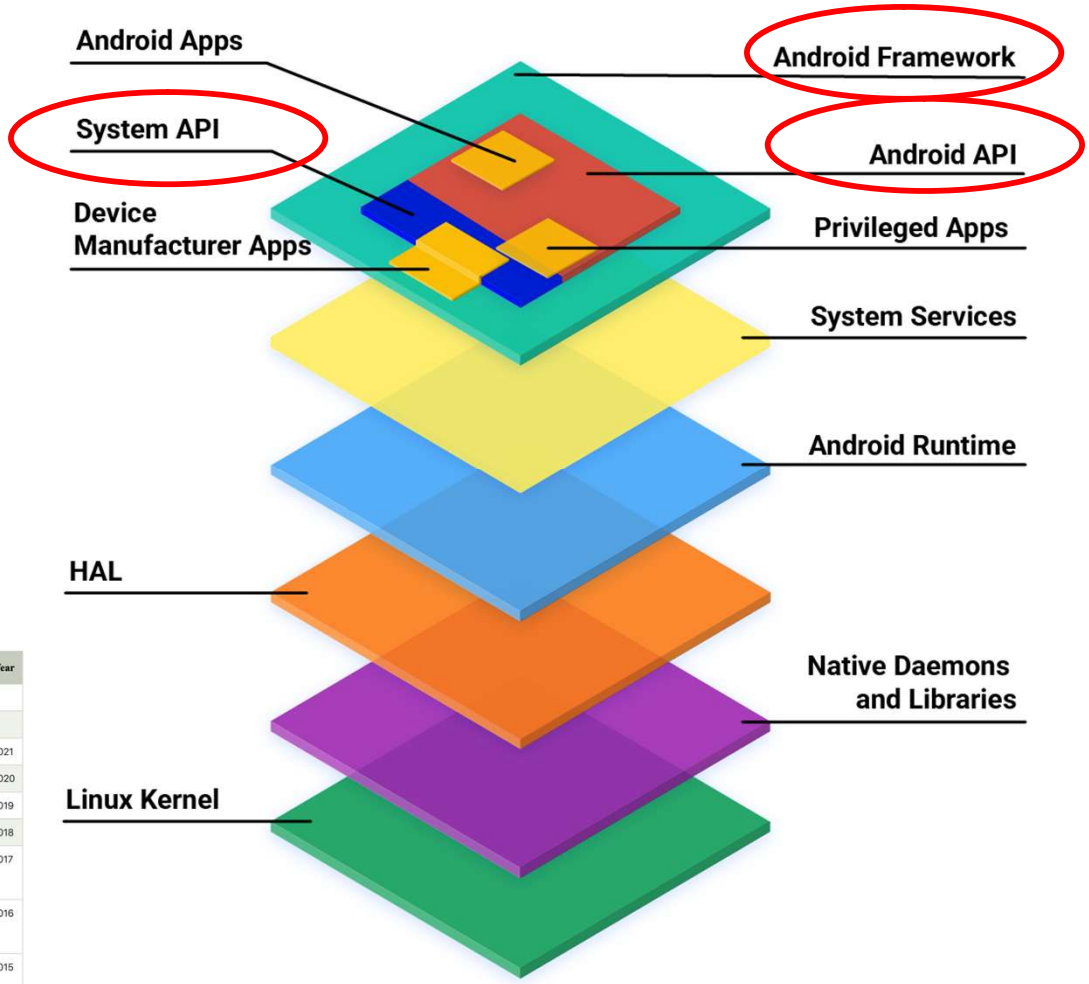
- Installation: Resides in a special folder on the system partition, usually */system/priv-app*. This partition is read-only for the user.
- Special permissions: these permissions are not accessible to regular user apps (interact directly with hardware in non-standard ways).
- The user cannot uninstall them, at most disable them.
- Typical examples: the "Settings" app of the phone, the system interface ("SystemUI", which manages the status bar and notifications), the telephone dialing operator.



- A "Device Manufacturer App" is any app (privileged or not) that the manufacturer (Samsung, Xiaomi, etc.) pre-installs to customize the device and differentiate it from the competition.

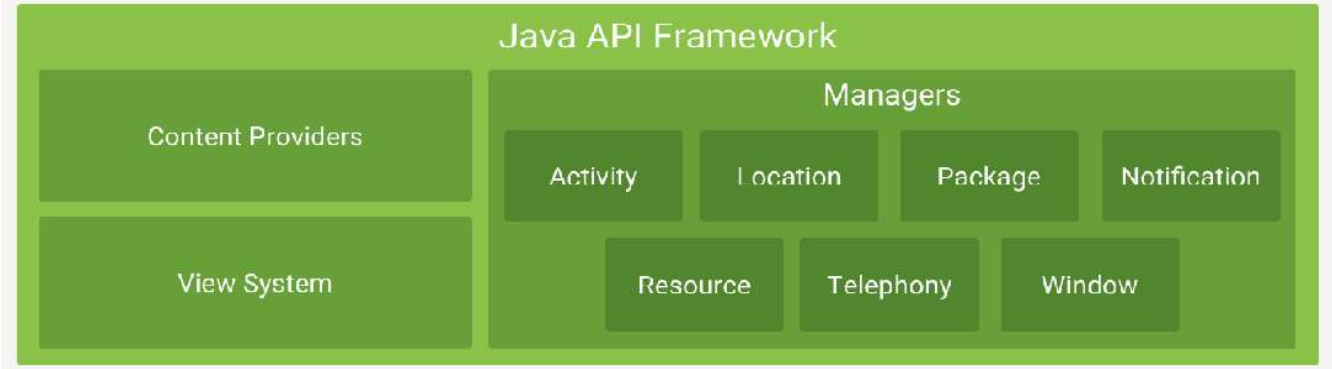
- **Android API** is the portion of the framework are publicly accessible, as accessible as SDK, for example
  - **android.graphics** – A low-level 2D graphics drawing API including colors, points, filters, rectangles and canvases.
- **System APIs**, is the portion of the framework available only to OEMs (Original Equipment Manufacturer), These APIs are marked as @SystemApi in the source code.
- ***Android framework:*** all the API plus several managers

Android Version	API Level / SDK	Version Name	Version Code	Year
Android 13 (Beta)	Level 33	Tiramisu	TIRAMISU	
Android 12	Level 32 (Android 12L)	Snow Cone	S_V2	
	Level 31 (Android 12)		S	2021
Android 11	Level 30 (Android 11)	Red Velvet Cake	R	2020
Android 10	Level 29 (Android 10)	Quince Tart	Q	2019
Android 9	Level 28 (Android 9)	Pie	P	2018
Android 8	Level 27 (Android 8.1)	Oreo	O_MR1	2017
	Level 26 (Android 8.0)		O	
Android 7	Level 25 (Android 7.1)	Nougat	N_MR1	2016
	Level 24 (Android 7.0)		N	
Android 6	Level 23 (Android 6)	Marshmallow	M	2015
Android 5	Level 22 (Android 5.1)	Lollipop	LOLLIPOP_MR1	
	Level 21 (Android 5.0)		LOLLIPOP, L	2014

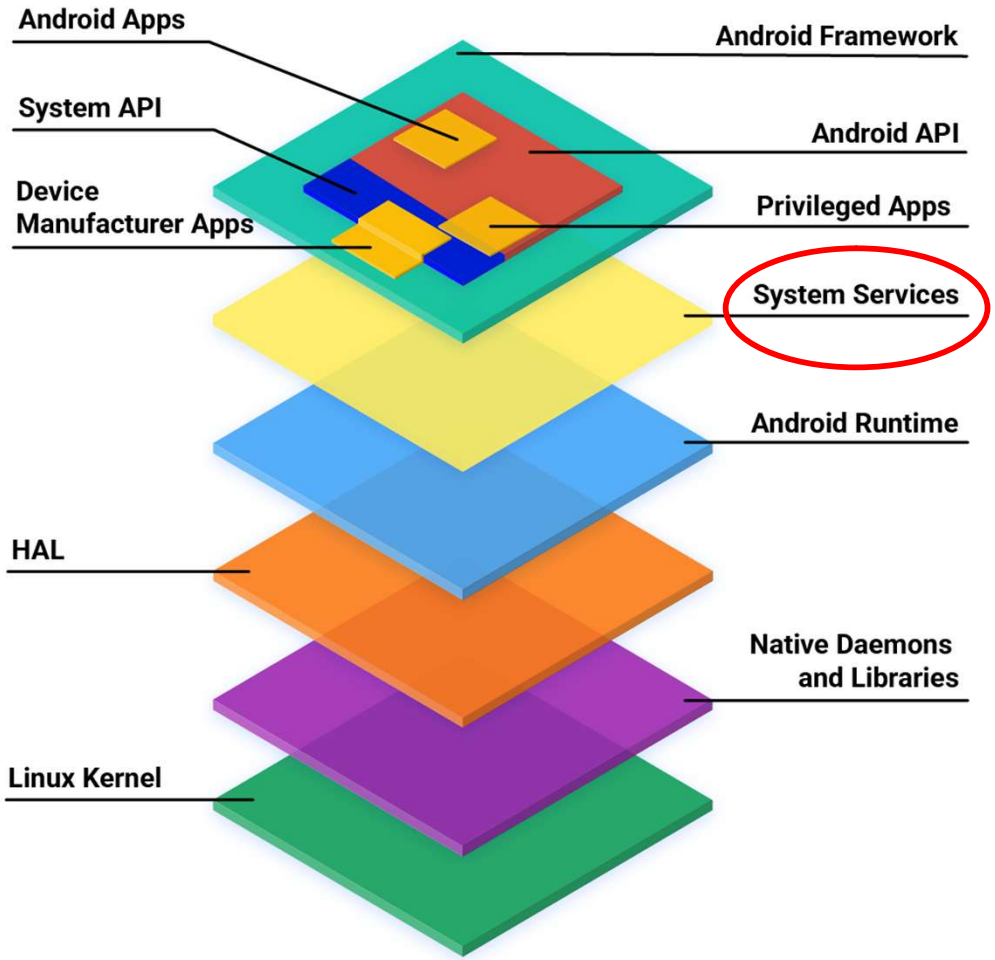


API FRAMEWORK

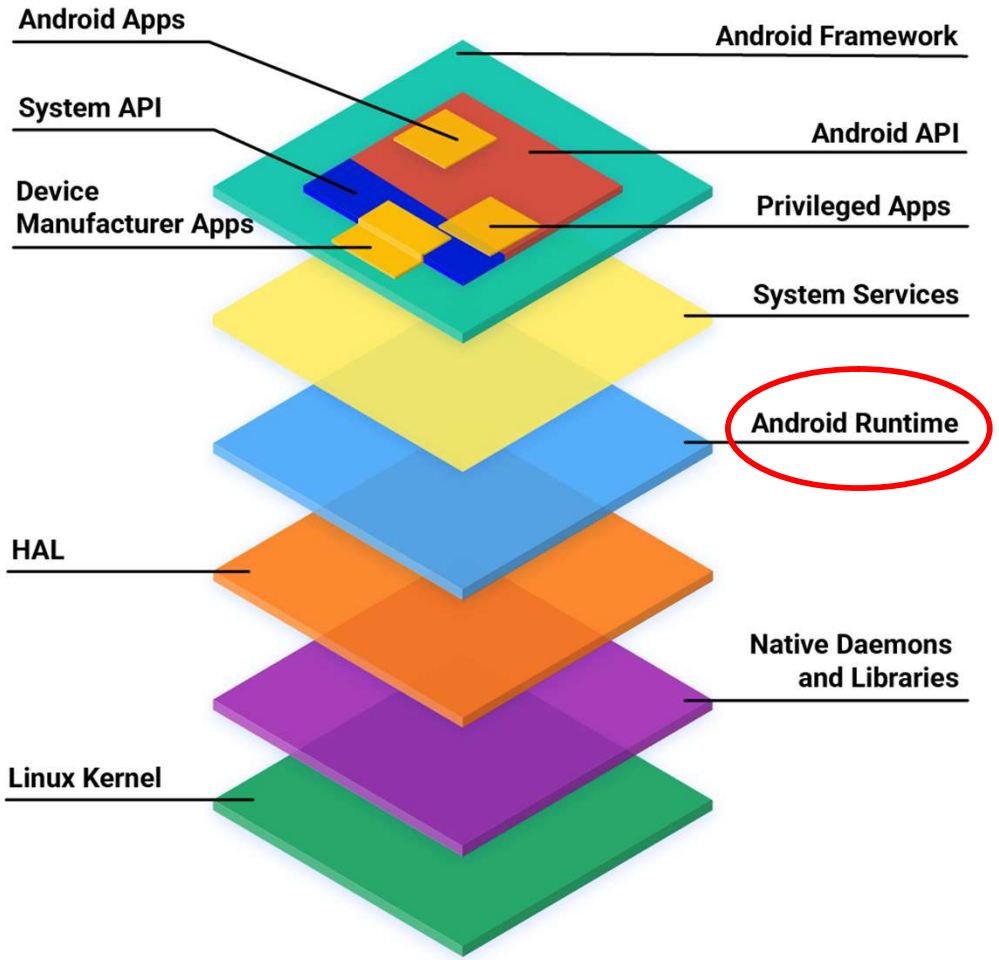
- Any android phone has the Java API framework already installed in it
- The Applications based only on this API do not include the implementation of the API in the APK
- Modern applications are based on **androidx.*** and **kotlin.***; they include the implementation in the APK, in words these libraries are not shared among apps.
- These libraries are included in the code during the build phase as specified in the **gradle** file (see later)
- In the *release build mode* only the portion of the library that is used is included



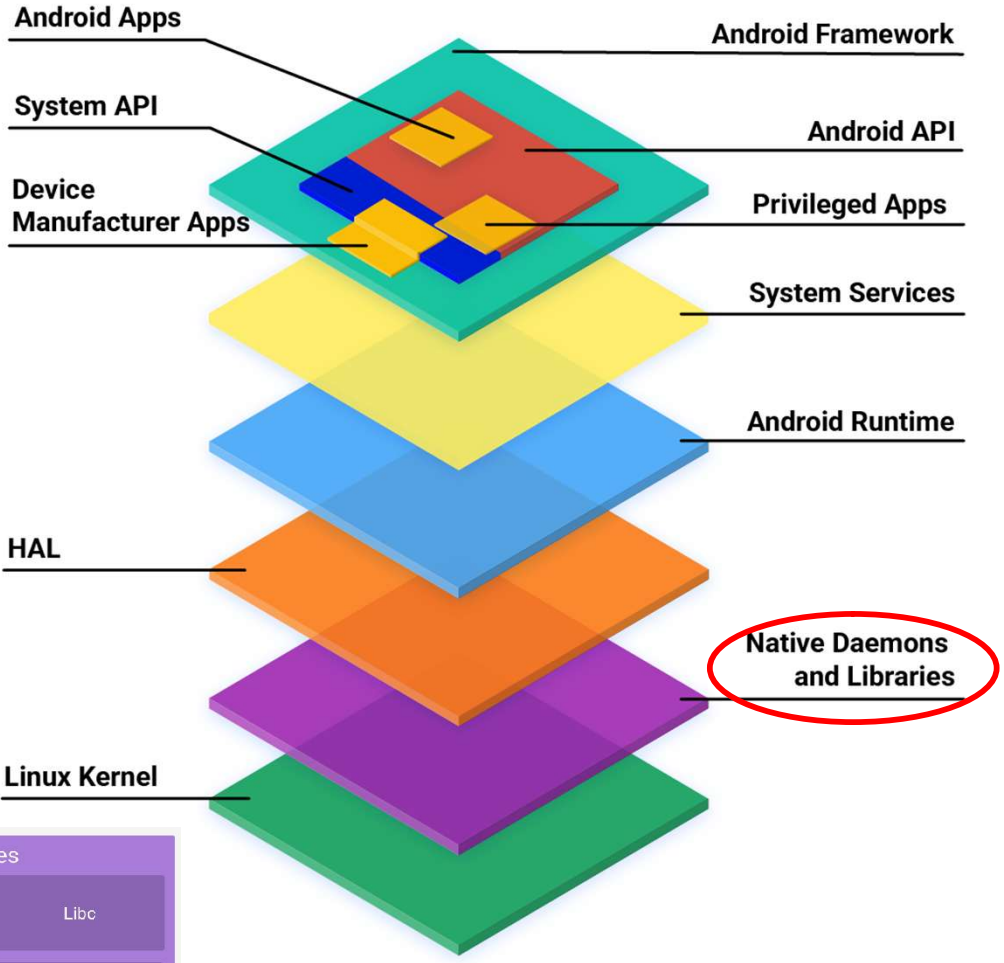
- ***System services*** are modular, focused components such as, *SurfaceFlinger*, *Window Manager*, and *MediaService*.

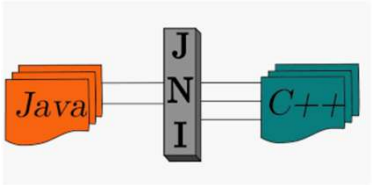


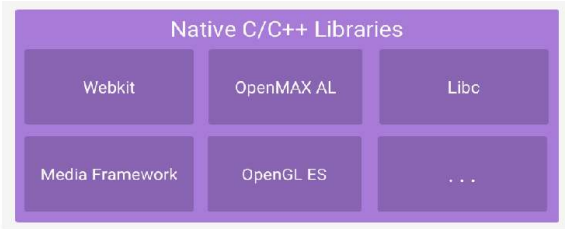
- **Android runtime (ART)** it's a VM.
- ART performs the translation of the app's bytecode into processor-specific instructions that are executed by the device's runtime environment.
- ART uses ahead-of-time (AOT) compilation, and starting in Android 7, it uses a hybrid combination of AOT compilation, just-in-time (JIT) compilation



- **Native daemons** include init, healthd, logd, and storaged.
- These daemons interact directly with the kernel
- **Native libraries** are software libraries written in C/C++, and compiled for the specific architecture e.g., *openGL ES or libc*, *OpenCV*, *tensorflow*
- They are included as kotlin libraries, but the implementation is a proxy via a 'bridge' (JNI) to executable code. Are not executed via the ART
- One can also directly in C/C++ to have high efficiency and link the code with other Kotlin code (Native Development Kit) via Java Native Interface (**JNI**)







PLATFORM CHANNELS

- Platform channels is a mechanism that allows to call Kotlin code from other source code (like JNI)
- It introduce an overhead due to message serialization/deserialization
- They allow to mix platform specific (optimized code) with general usable code (like UI), which allows cross-platform development using a single codebase
- Flutter uses the Dart programming language and platform channels when necessary
- The rendering engine is very efficient

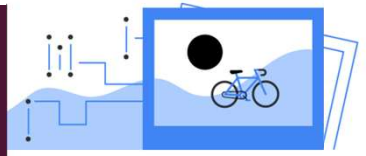
A CLASSIFICATION OF APPLICATIONS BASED

- Platform-native = code uses only the official and standard API (Kotlin/Java for android)
- CPU-native = code compiled directly to machine of a specific CPU code without VM (C/C++ via NDK)
- Hybrid = combines web technologies with native code (PhoneGap,..)
- Cross-platform UI = use a proprietary an efficient rendering engine (flutter)

EXTERNAL LIBRARIES

- Android extend by numerous and powerful libraries, for example for ML and CV

SOME USEFUL LIBRARY: ML KIT



- ML Kit targets on-device machine learning and real-time use cases where for example process a live camera stream
- ML Kit provides a variety of pre-trained, ready-to-use machine learning models that can easily integrate into Android apps. These models cover a range of common use cases
- Custom Models: In addition to pre-trained models, ML Kit allows to use custom TensorFlow Lite models.

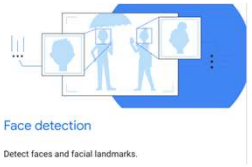
Vision

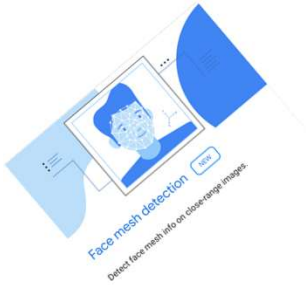
- Text recognition v2
- Face detection
- Face mesh detection (Beta)
- Pose detection (Beta)
- Selfie segmentation (Beta)
- Barcode scanning
- Image labeling
- Object detection and tracking
- Digital ink recognition

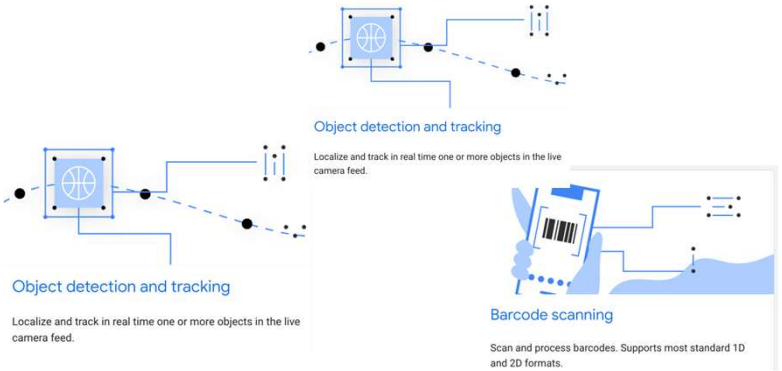
Custom models

Natural language

- Language identification
- Translation
- Smart reply
- Entity extraction (Beta)







EXAMPLES OF ML FEATURES

- Text Recognition:
  - On-device: Recognizes text in images, including Latin and non-Latin scripts.
  - Cloud-based: Offers higher accuracy and supports more languages.
- Barcode Scanning:
  - Detects and decodes various barcode formats (QR codes, barcodes, etc.).
- Image Labeling:
  - Identifies objects and concepts within images.
  - Object Detection and Tracking:
    - On-device: Detects and tracks objects in real-time using the camera.
    - Cloud-based: Offers higher accuracy and a wider range of object categories.
- Face Detection:
  - Detects faces in images and provides facial landmarks and attributes.

EXAMPLES OF ML FEATURES

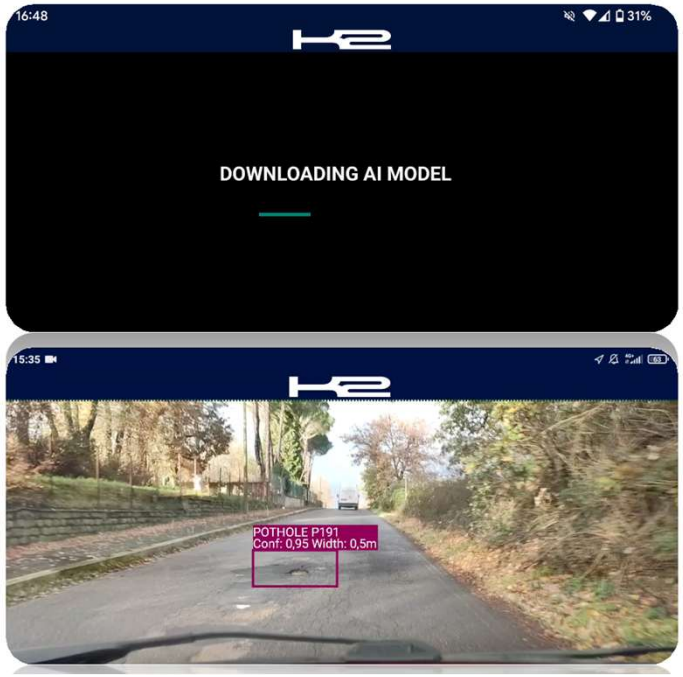
- Digital Ink Recognition:
  - Recognizes handwritten text and shapes.
- Pose Detection:
  - Detects human body poses and provides skeletal landmarks.
- Selfie Segmentation:
  - Separates a person from the background in real-time using the camera. Natural Language APIs:
- Language Identification:
  - Identifies the language of a given text.
- On-device Translation:
  - Translates text between languages offline.
- Smart Reply:
  - Generates suggested replies for incoming messages.

OPENCV FOR ANDROID

- **OpenCV** (**Open-Source Computer Vision Library**) is a library of programming functions mainly for real-time computer vision, including
- **Image and Video Processing**:
  - Image Manipulation: Resize, rotate, crop, and adjust brightness/contrast of images.
  - Object Detection: Identify and locate specific objects within images or video streams. Popular applications include face detection, vehicle detection, and object tracking.
  - Image Filtering: Apply various filters like blurring, sharpening, edge detection to enhance or transform images.
  - Feature Extraction: Extract key features from images for tasks like image recognition and object classification.
- **Video Analysis**:
  - Process video frames for tasks like motion detection, object tracking, and video stabilization.
- **Augmented Reality (AR)**:
  - Marker-based AR: Overlay digital content onto real-world objects detected through markers.
  - Location-based AR: Integrate virtual elements into the user's view based on their location.

USE CASE


Mobile UI
Login
Android ID
**********704f10
organization name
organization password


16:48
31%
DOWNLOADING AI MODEL
15:35
POTHOLE P191
Conf: 0,95 Width: 0,5m


POTHOLE P74
Conf: 0,39 Width: 0,8m
POTHOLE P68
Conf: 0,31 Width: 0,5m

ARCORE

- ARCore is Google’s platform for building augmented reality experiences.
- Using different APIs, ARCore enables a phone to sense its environment, understand the world and interact with information. Some of the APIs are available across Android and iOS to enable shared AR experiences.
  - **Motion tracking** allows the phone to understand and track its position relative to the world.
  - **Environmental understanding** allows the phone to detect the size and location of all type of surfaces: horizontal, vertical and angled surfaces like the ground, a coffee table or walls.
  - **Light estimation** allows the phone to estimate the environment's current lighting conditions.

SceneView Open Community

3D and AR libraries and samples for Android, iOS and React Native (Android and iOS) developers

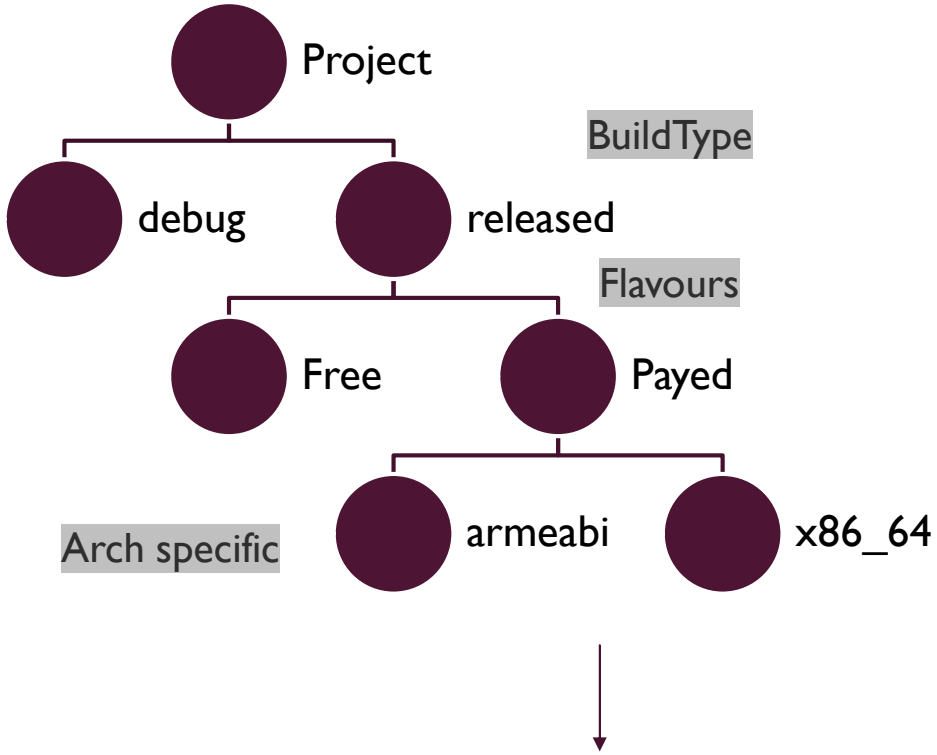
99 followers | France | https://sceneview.github.io | @thomas_gorisse | thomas.gorisse@gmail.com

- SceneView is a 3D and AR Android Composable and View with Google Filament and ARCore.
- This is a Sceneform replacement in Kotlin

FIREBASE

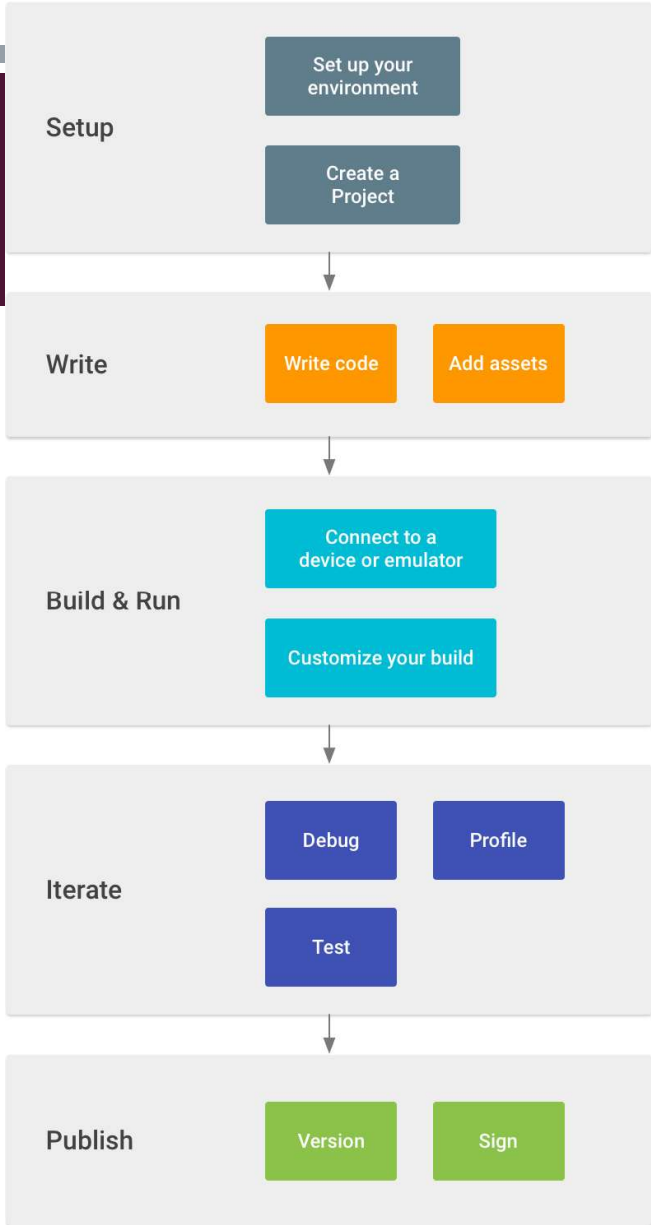
- Firebase for Android is a comprehensive cloud platform that provides a suite of backend services.
- It include services like
- **Authentication**
- **Cloud Firestore**:
  - A flexible, scalable NoSQL document database for mobile and web development that synchronizes data in real-time. It features more powerful querying capabilities than the older Realtime Database.
- **Realtime Database**
  - NoSQL database, which stores data as a single JSON tree and synchronizes changes instantly to all connected clients.
- **Cloud Storage**:
  - A powerful and secure way to store user-generated content, such as photos and videos.
- **Cloud Messaging (FCM)**:
  - A cross-platform messaging solution for sending notifications and messages to users on Android, iOS, and the web.

WORKFLOW AND BUILD VARIANT



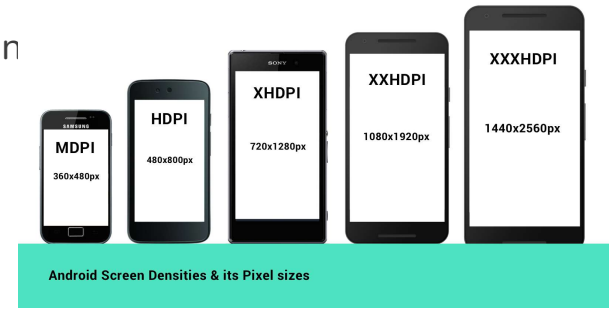
https://developer.android.com/build

https://developer.android.com/build/android-build-structure



EXAMPLE OF VARIANTS

- app-hdpiX86-release.apk:
  - Contains code and resources for hdpi density and x86 ABI only.
- app-hdpiX86_64-release.apk:
  - Contains code and resources for hdpi density and x86_64 ABI only.
- app-mdpiX86-release.apk:
  - Contains code and resources for mdpi density and x86 ABI only.
- app-mdpiX86_64-release.apk:
  - Contains code and resources for mdpi density and x86_64 ABI on

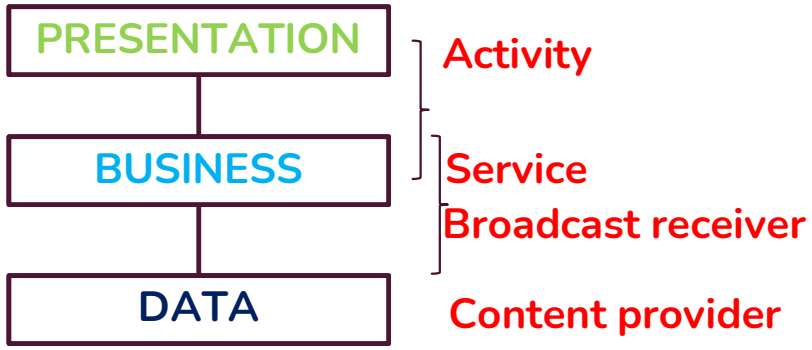


CORE SOFTWARE COMPONENTS (ANDROID.*)

- Classic (and ‘old’) apps are only based on 4 components
- **Activity**
- **Service**
- **Broadcast receiver**
- **Content provider**
- These libraries are in any android phone and then are not included in the APK

SOFTWARE ARCHITETTURE

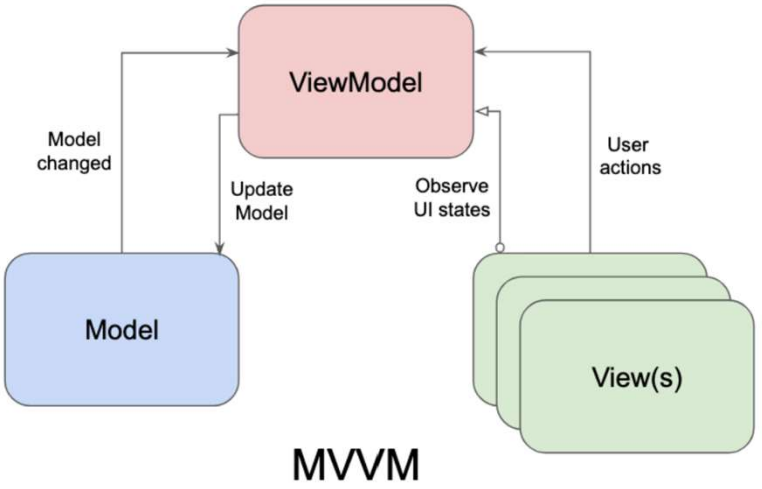
- **Presentation Layer (UI Layer):** responsible for displaying information to the user and handling user interactions.
- **Business Logic Layer**: the core logic of the application, responsible for processing data, making decisions, and enforcing business rules.
- **Data Access Layer (Persistence Layer)**: handles interactions with data sources, such as databases, files, or external services.



JETPACK ARCHITECTURE COMPONENTS (**ANDROIDX.***)

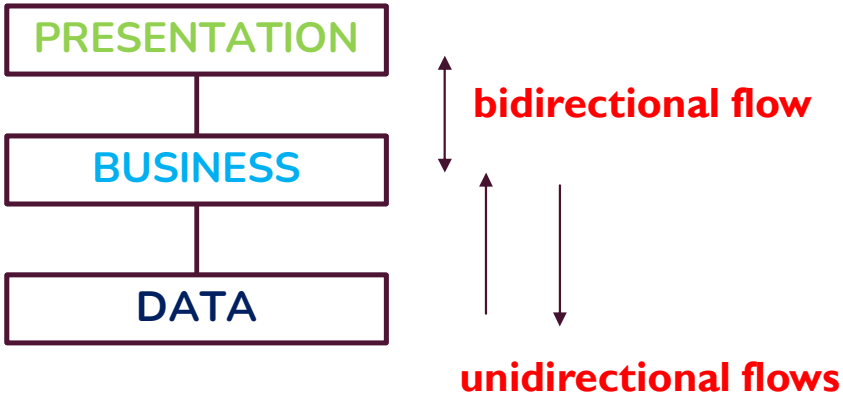
- Data Binding : Declaratively bind UI elements
- Lifecycles : Manages activity and fragment lifecycles
- LiveData : Notify views of any database changes
- Navigation : Handle everything needed for in-app navigation
- Paging : Gradually load information on demand from your data source
- Room : Fluent SQLite database access
- ViewModel : Manage UI-related data in a lifecycle-conscious way
- WorkManager : Manage every background jobs in Android with the circumstances we choose

MODERN APP DEVELOPMENT



MVVM

- **M**odel-**V**iew-**V**iew**M**odel (MVVM)
- **View** implements the presentation Layer
- **ViewModel** implements the business logic
- **Mode** implements the Data Access Layer



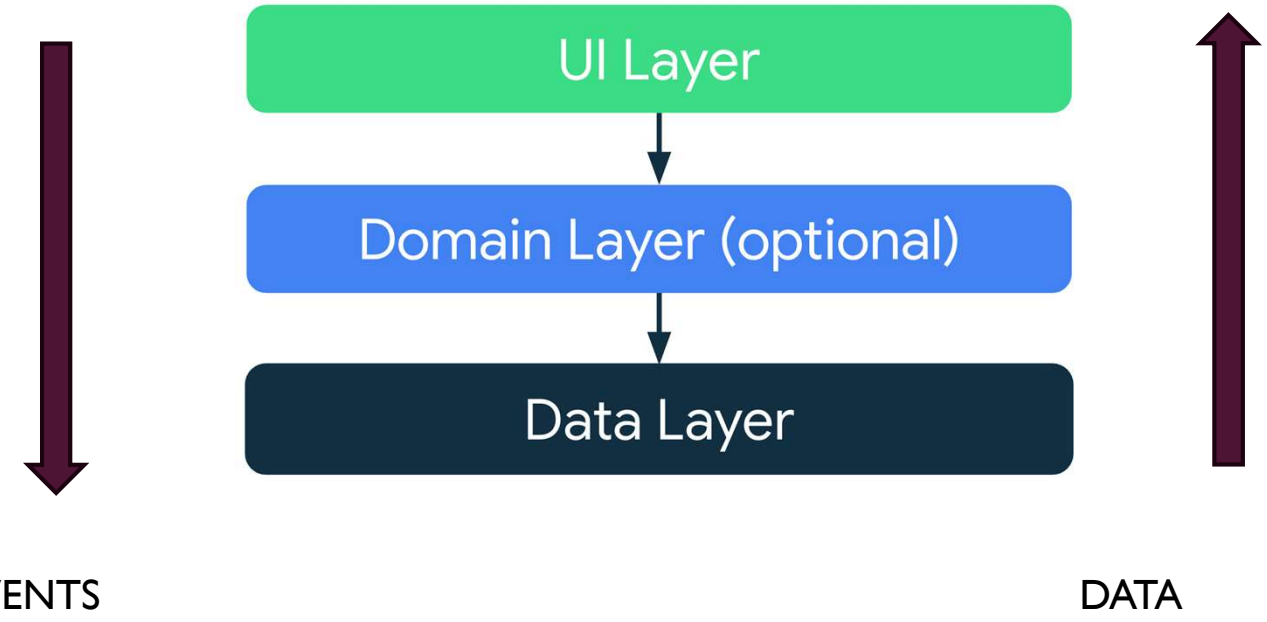
- Data/Event flows in the MVVM:
- One **bidirectional flow** from the View to the ViewModel (user action) and vice versa (observe UI states). Implicit binding and synchronization (reactions to change is 'automatic')
- **Unidirectional flows**: from model to ViewModel to the model (update Model) and from Model to ViewModel (model changed). Two explicit calls

UNIDIRECTIONAL DATA FLOW (UDF)

- In UDF, **data** flows in only one direction.
- The **events** that modify the data flow in the opposite direction.

MODERN APP DEVELOPMENT (MAD)

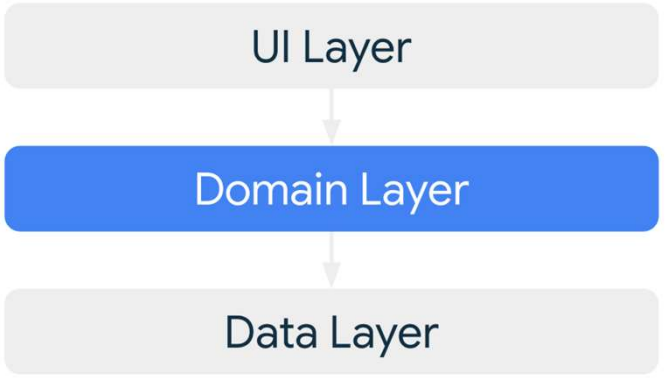
- **UI layer** displays application data on the screen.
- **Domain layer** is an optional layer that sits between the UI and data layers, responsible for encapsulating *complex business logic*.
- **Data layer** contains (simple) business logic of the app and exposes application data.



https://developer.android.com/topic/architecture

DOMAIN LAYER

- The domain layer is *optional*
- It sits between the UI and data layers. Not all apps will need it.
- The domain layer is in fact responsible for encapsulating *complex business logic*, or simple business logic that is reused by **ViewModel**.
- This layer is optional because for example, to handle complexity or favor reusability.



DESIGN PRINCIPLES

- **Separation of concerns.** Split the code in separate focused sections for example, UI-based classes should only contain logic that handles UI and operating system interactions and not business logic.
- **Drive UI from data models.** drive UI elements from data models. Data models represent the data of an app and are *independent* from the UI elements and other components in your app. They they are not tied to the UI and app component lifecycle but will still be destroyed when the OS decides to remove the app's process from memory.
- **Single source of truth.** assign a Single Source of Truth (SSOT) to data. The SSOT is the *owner* of that data, and only the SSOT can modify or mutate it. To achieve this, the SSOT exposes the data using an immutable type, and to modify the data, the SSOT exposes functions or receive events that other types can call.
- **Unidirectional Data Flow** (UDF) pattern. In UDF, a state (data values) flows in only one direction (from the SSOT to the UI element). The events that modify the data flow in the opposite direction.

https://developer.android.com/topic/architecture/recommendations