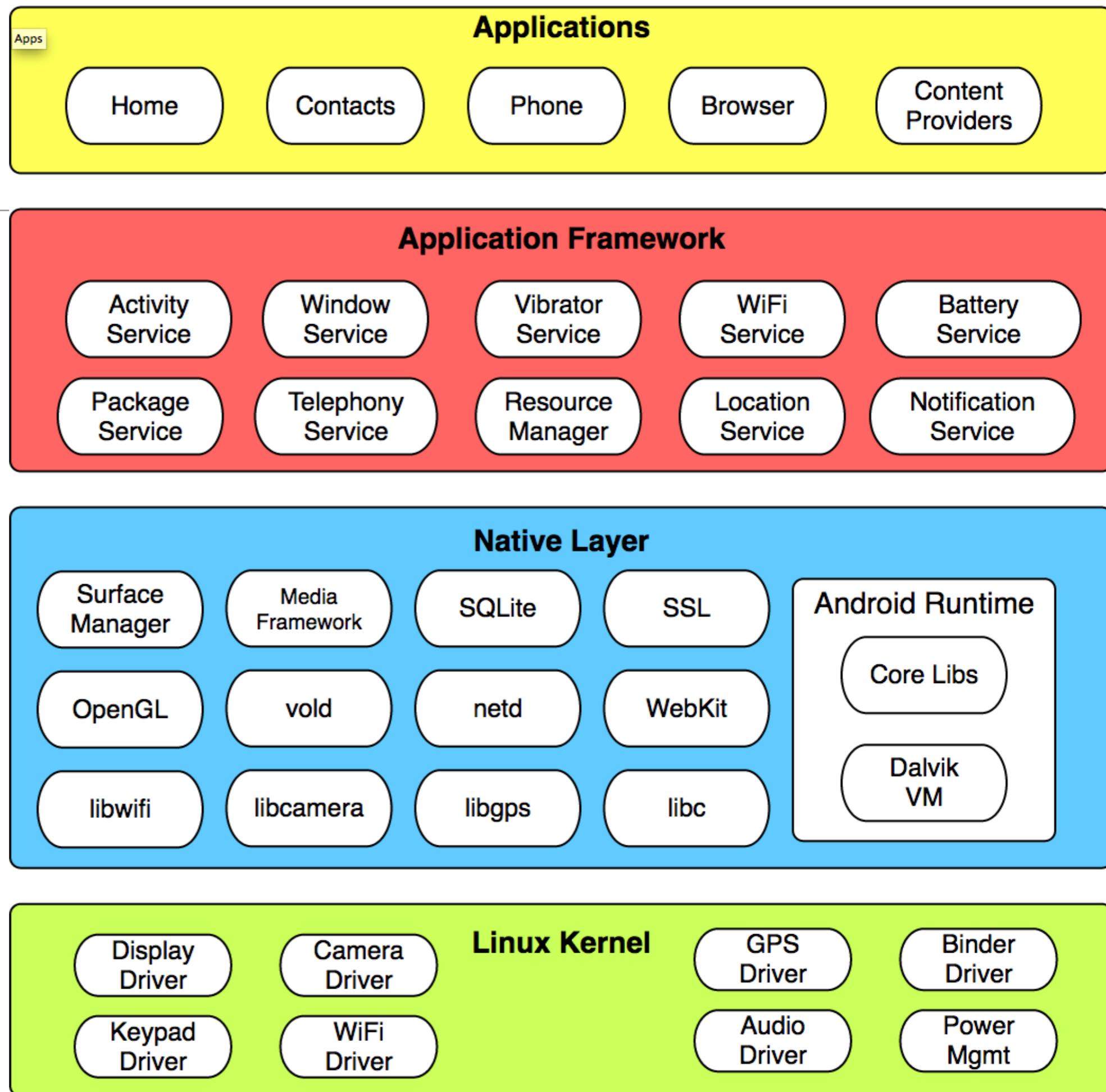


# The Android Stack

---

# Overview

- The Android operating system is like a cake consisting of various layers.
- Each layer has its own characteristics and purpose — but the layers are not always cleanly separated and often seep into one another.



# Android & Linux

- Although Android is based on linux, it is not just another flavour of Linux, in the way that Ubuntu, Fedora, or Red Hat are.
- Many things you'd expect from a typical Linux distribution aren't available in Android, such as the X11 window manager, the ability to add a person as a Linux user or even the glibc standard C library.
- On the other hand, Android adds quite a bit to the Linux kernel, such as
  - an improved power management that is well-suited for mobile battery-powered devices,
  - a very fast interprocess communication mechanisms
  - mechanisms for sand- boxing applications so they are isolated from one another.

# Linux Kernel

- **Portable**

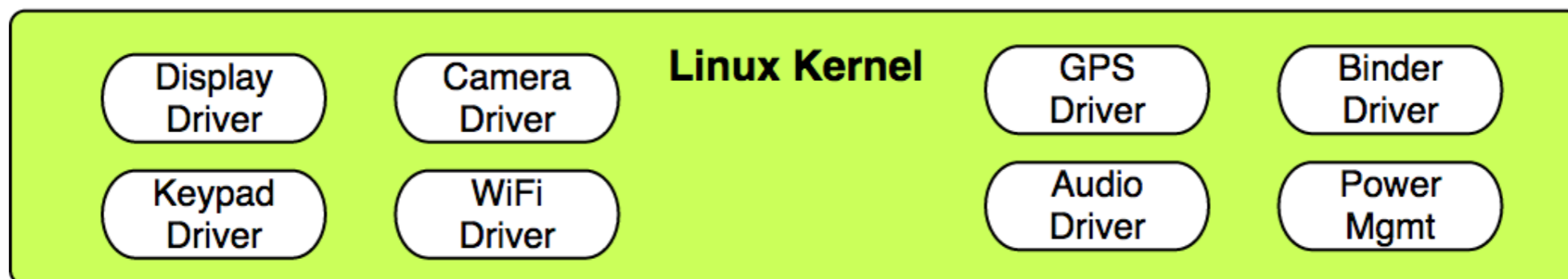
- Most low-level parts of Linux have been written in fairly portable C code, which allows for third parties to port Android to a variety of devices.

- **Secure**

- Linux is a highly secure system, having been tried and tested through some very harsh environments over the decades.
- Android relies heavily on Linux for security, and all Android applications run as separate Linux processes with permissions set by the Linux system, passing many security concerns to the underlying Linux system.
- The kernel is the sole enforcer of Android permissions, providing a simple, powerful, security mechanism. It also allows Android apps access to native code, such as fast C implementations of various libraries via the Java Native Interface.

- **Features**

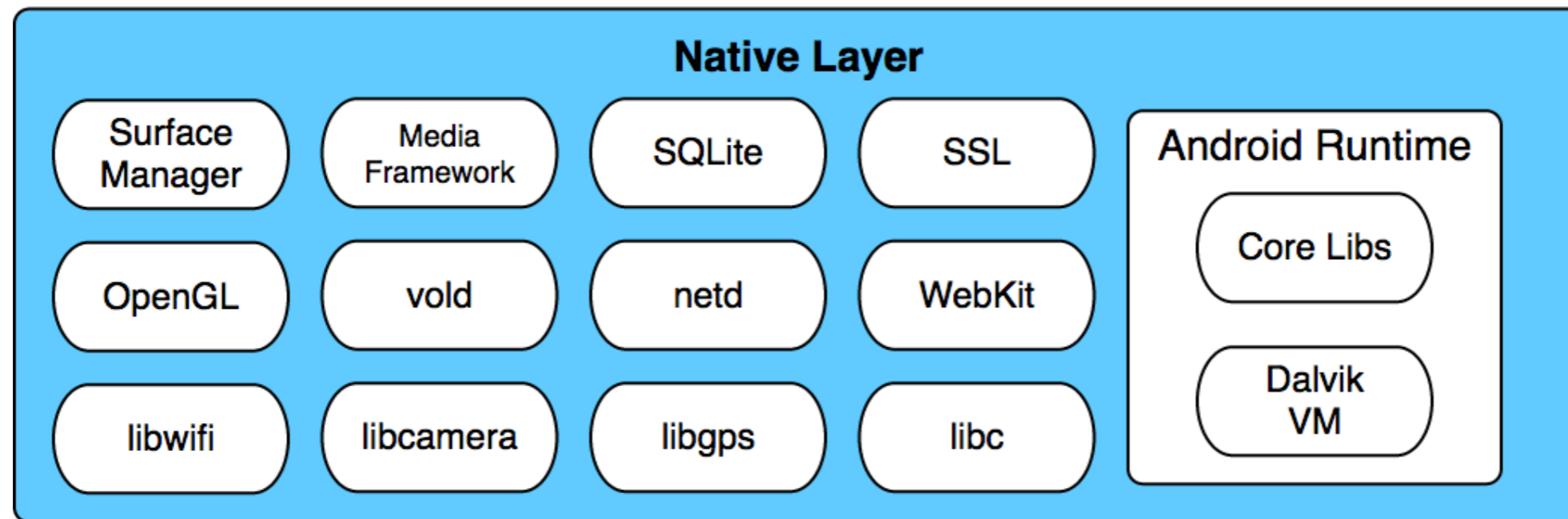
- The Linux kernel comes with a range of features. Android leverages many of them, e.g. support for memory and power management, networking and radio functionality.



# Native Layer

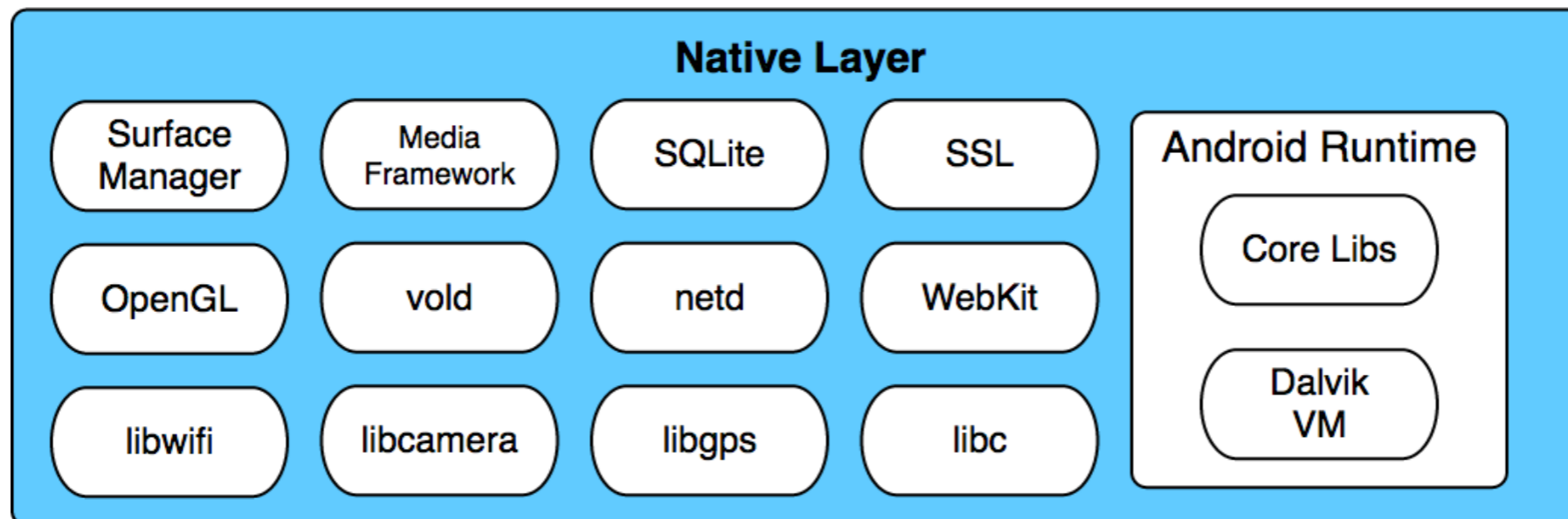
---

- The native libraries are C/C++ libraries. Their primary job is to support the Android Application Framework layer



- Some of these libraries are purpose-built for the Android OS, whereas others are often taken from the open source community in order to complete the operating system.

- **Binder:** A very fast inter-process communication mechanism that allows for one Android app to talk to another.
- **Framework libraries:** Various libraries designed to support system services, such as location, media, package installer, telephony, WiFi, voip, and so on.
- **WebKit:** A fast web-rendering engine used by Safari, Chrome, and other browsers.
- **SQLite:** A full-featured SQL database that the Android app framework exposes to applications.



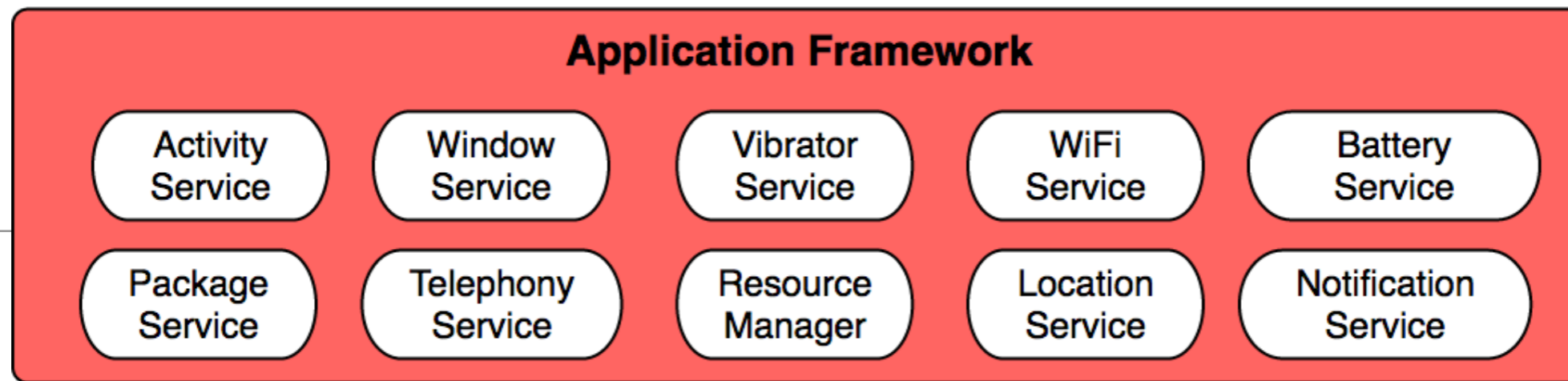
- **Apache Harmony:** An open source implementation of Java libraries.
- **OpenGL:** 3D graphics libraries.
- **OpenSSL:** The secure socket layer, allowing for secure point-to-point connectivity.

# Native Daemons

---

- Native daemons are executable code that usually runs to support some kind of system service. Prominent examples:
  - **Service Manager (servicemanager):** The umbrella process running all other framework services. It is the most critical native daemon.
  - **Radio interface layer daemon (rild):** Responsible for supporting the telephony functionality via GSP or CDMA, usually.
  - **Installation daemon (installd):** Supports management of apps, including installation, upgrades, as well as granting of permissions.
  - **Media server (mediaserver):** Supports camera, audio, and other media services.
  - **Android Debug Bridge (adb):** Supports developer connectivity from your PC to the device (including the emulator) so that you can develop apps for Android.

# Application Frameworks



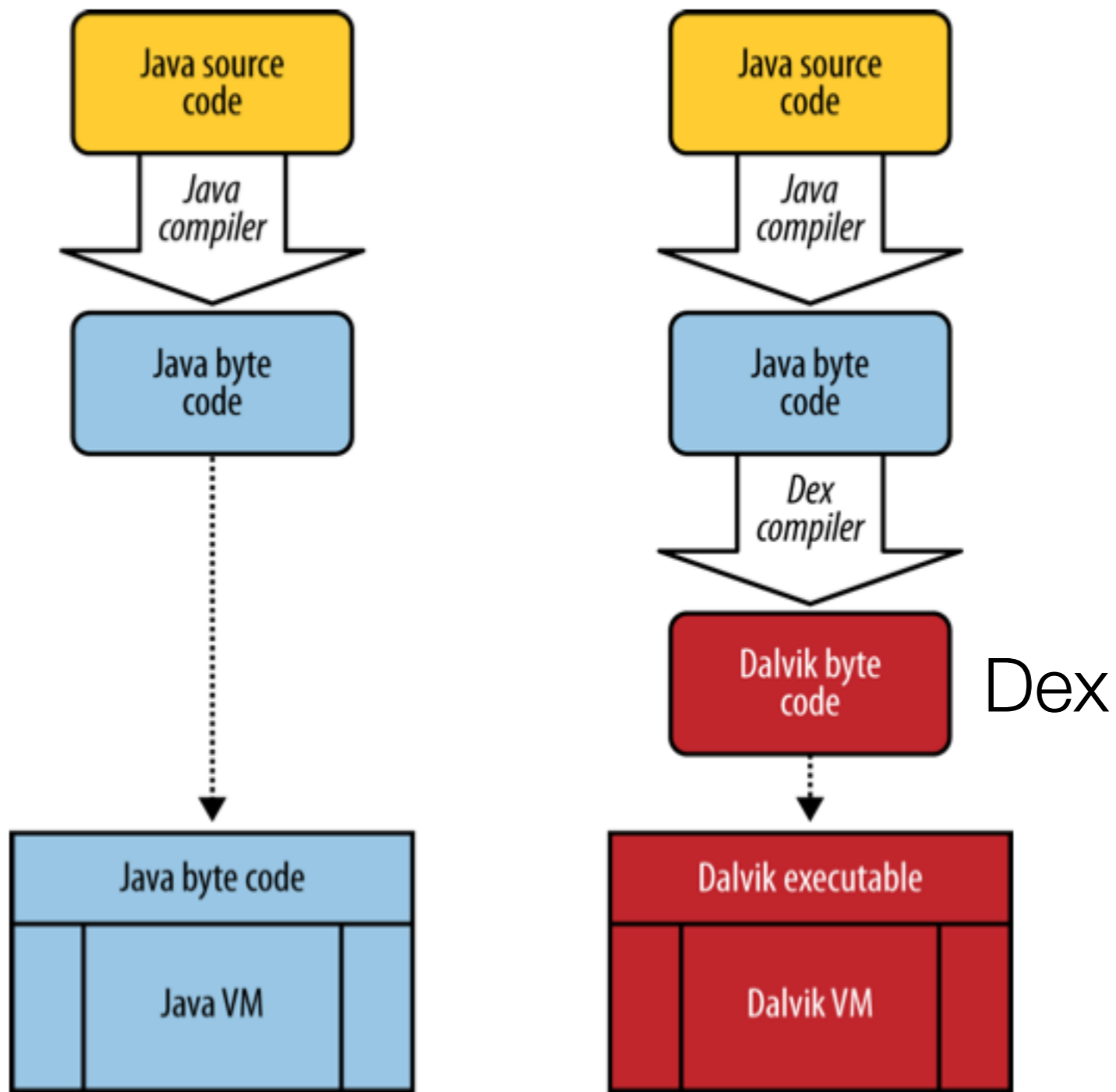
- The application framework is a rich environment that provides numerous libraries and services to help the app developer
- This is the best-documented and most extensively covered part of the platform because it is this layer that empowers developers to get applications to the market.
- In the application framework layer, there are numerous Java libraries specifically built for Android. These purpose-built Android classes live in `android.*` packages.
- There are also most of the standard Java libraries, such as `java.lang.*`, `java.util.*`, `java.io.*`, `java.net.*`, etc, which behave as documented in the oracle documentation
- You will also find many services (or managers) that provide the ecosystem of capabilities your application can tap into, such as location, sensors, WiFi, telephony, etc...



# JVM vs Dalvik

---

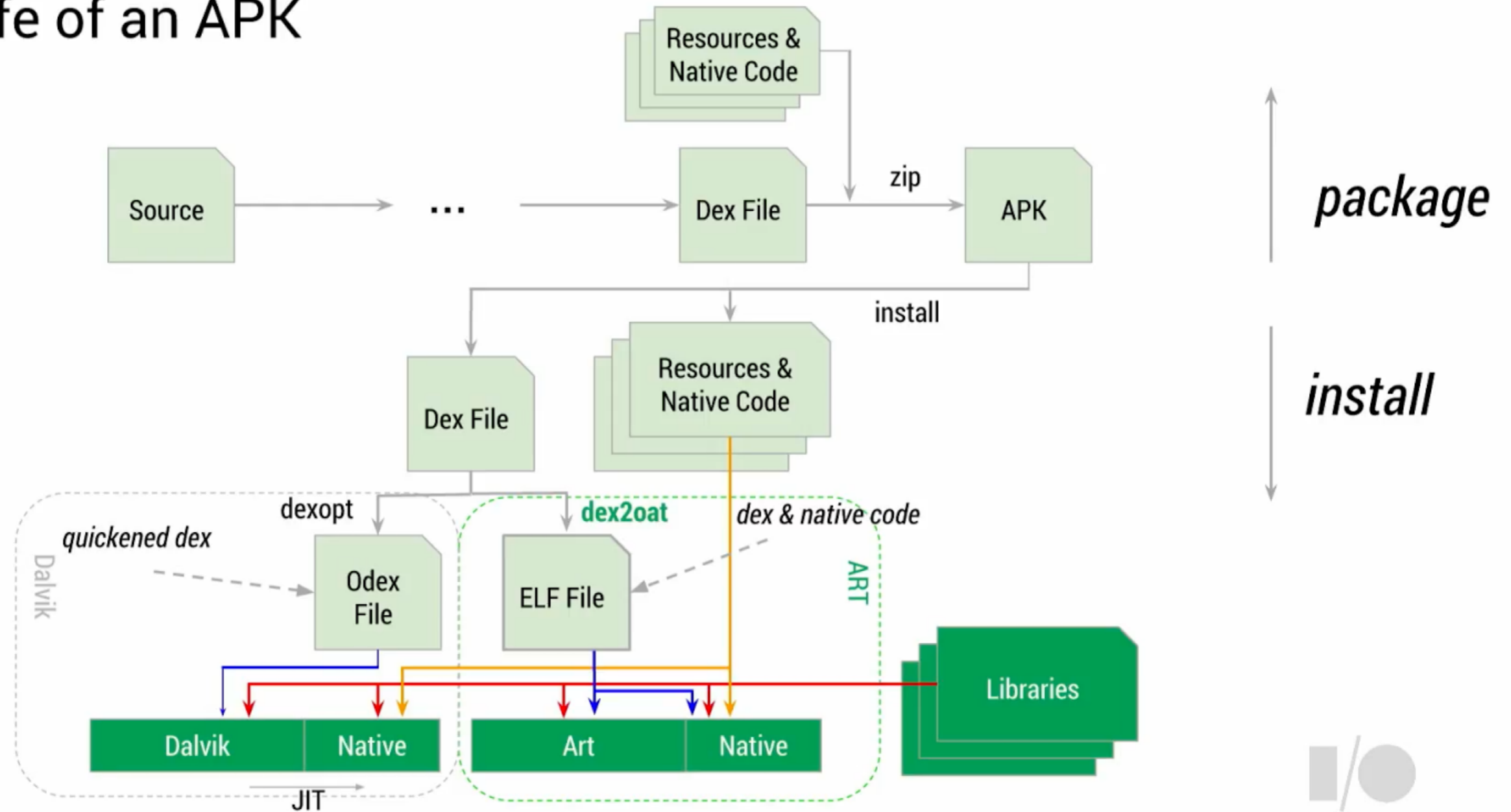
- In Java, you write your Java source file, compile it into Java **byte code** using the Java compiler, and then run this byte code on the Java VM.
- In Android you write the Java source file, and you still compile it to Java byte code using the same Java compiler.
- But at that point, you recompile it once again to Dalvik byte code using the Dalvik compiler - producing a **DEX** file
- It is this Dalvik byte code - **DEX Code** - that is then executed on the Dalvik





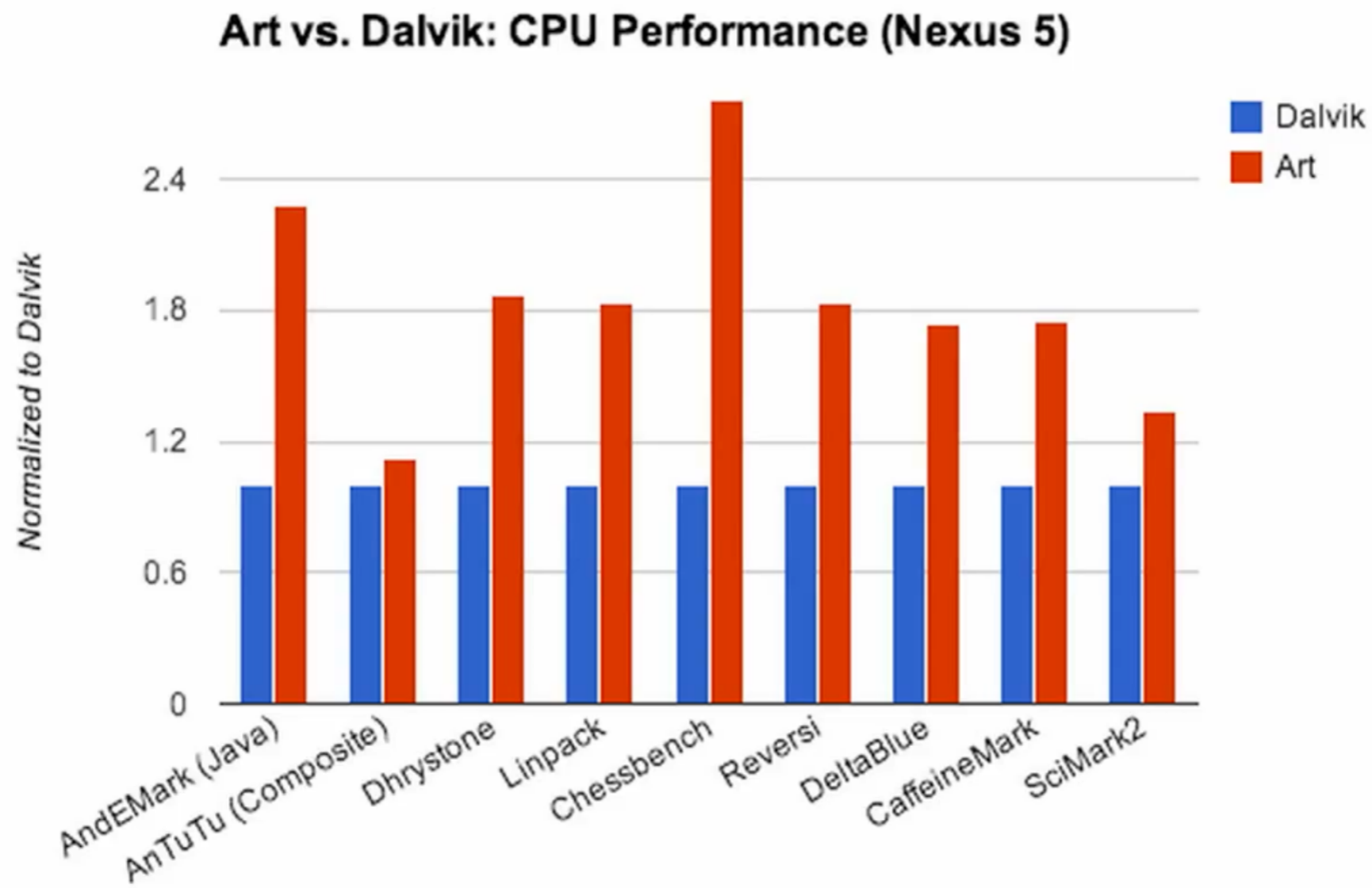
- ART, which stands for Android Runtime, handles app execution in a fundamentally different way from Dalvik.
- The big shift that ART brings, is that instead of being a Just-in-Time (JIT) compiler, it now compiles application code Ahead-of-Time (AOT).
- The runtime goes from having to compile from bytecode to native code each time you run an application, to having it to do it only once, and any subsequent execution from that point forward is done from the existing compiled native code.

# The life of an APK



- ART is compatible with Dalvik's existing byte-code format ("dex").
- From a developer's perspective, there are no changes at all in terms of having to write applications for one or the other runtime and no need to worry about compatibilities.

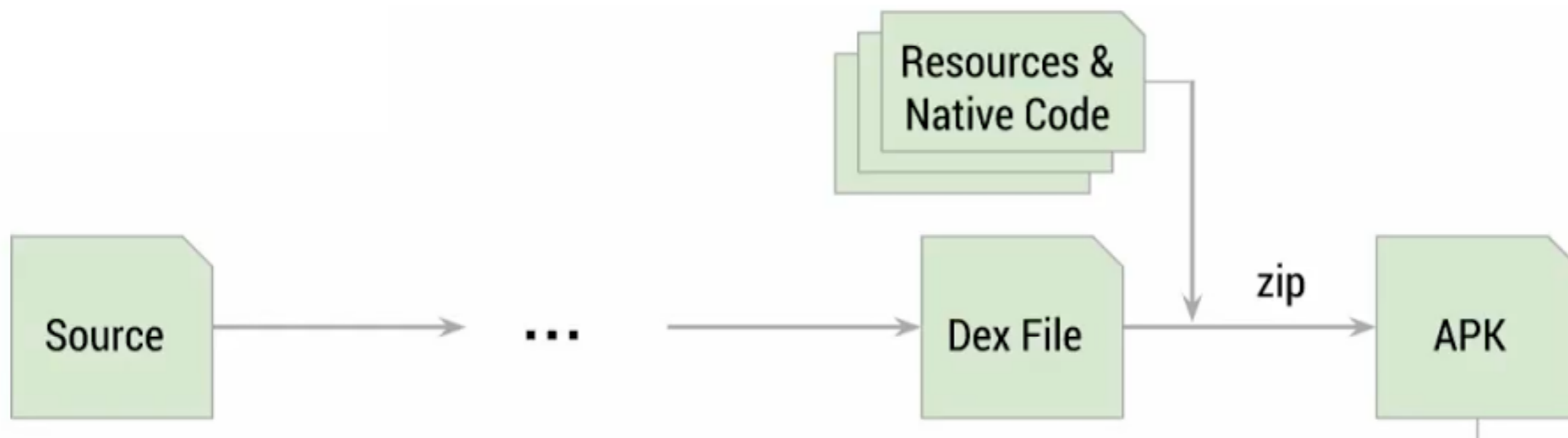
# Performance Boosting Thing, realized

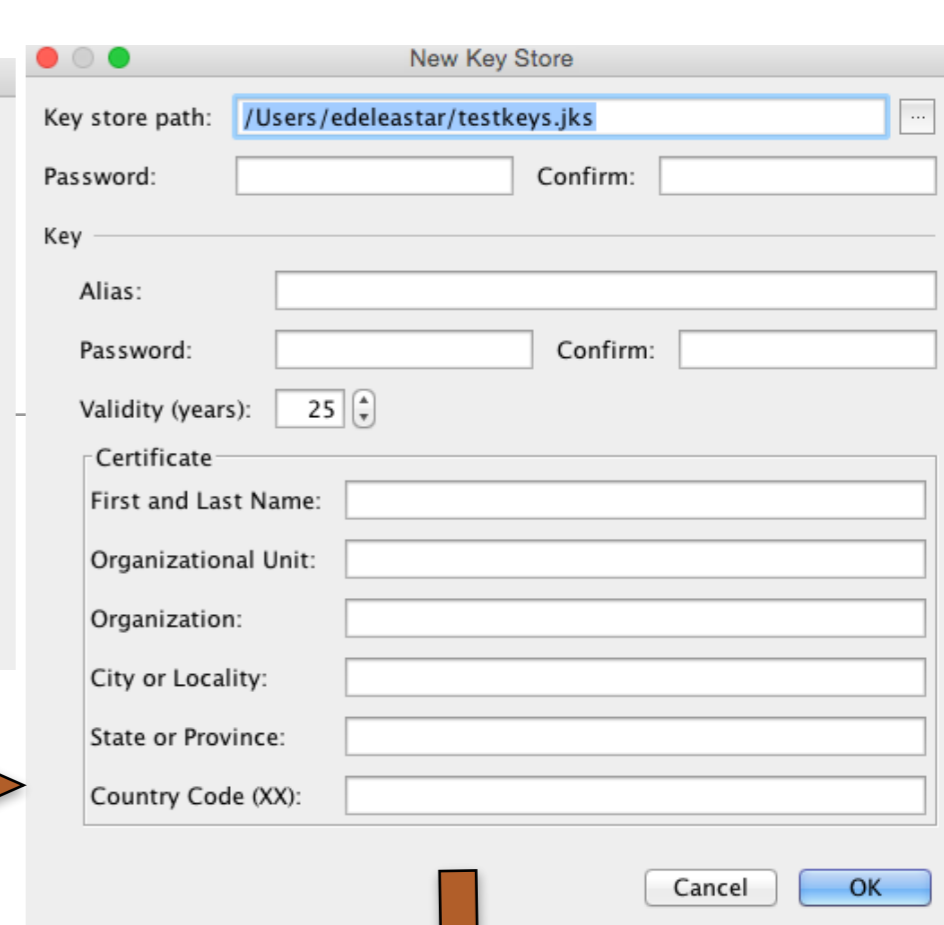
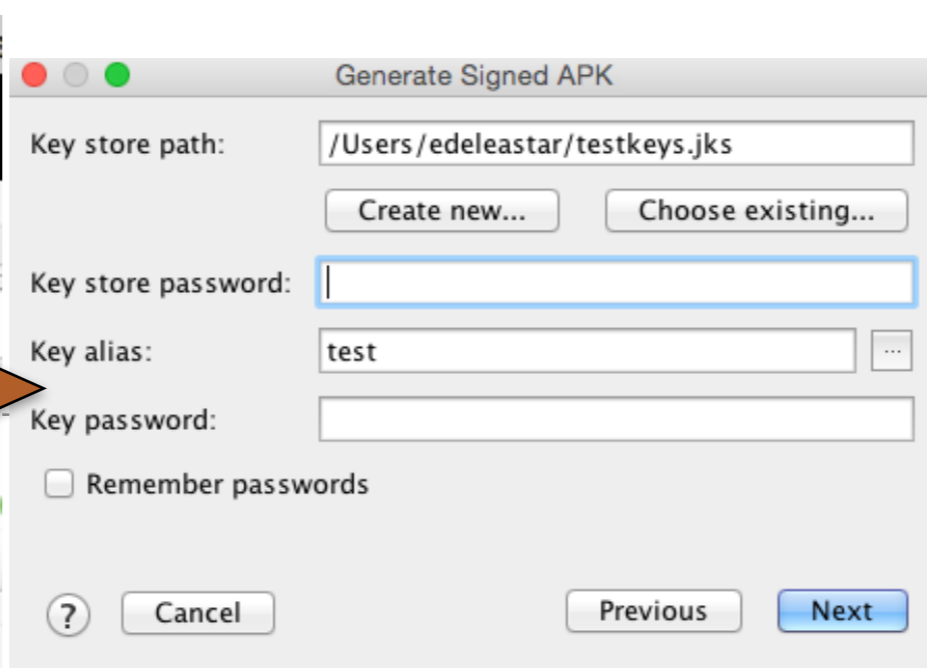
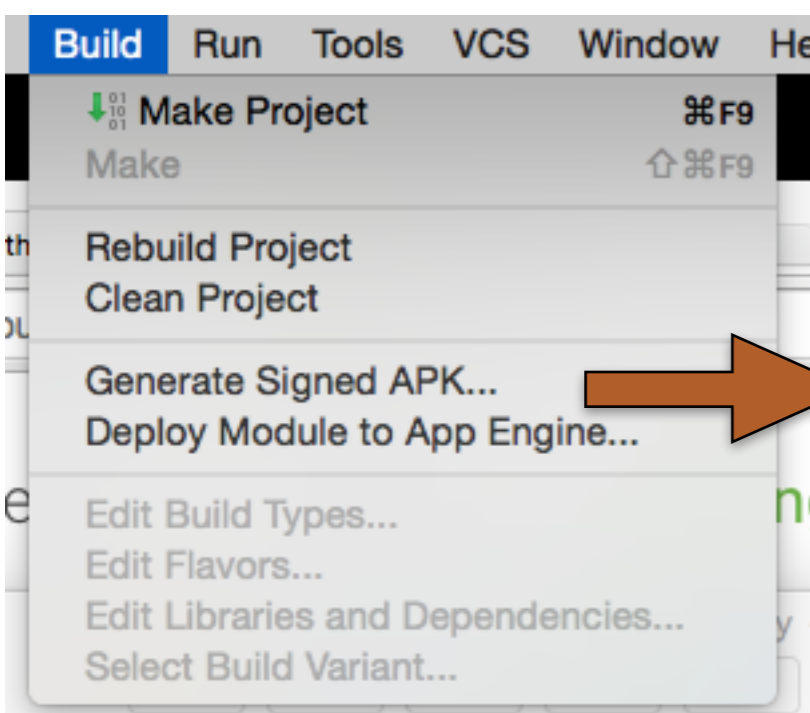


# Applications

---

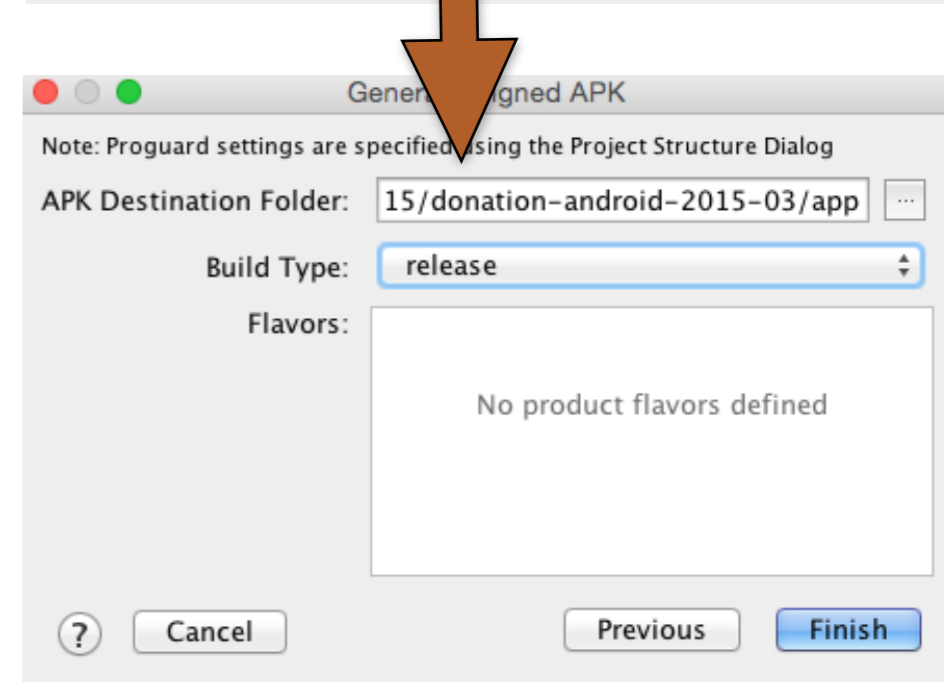
- An application is a single file. We call it an Android application package, or APK for short.
- It is a ZIP file that you can unzip and look inside using any archiving tool





## Signed APK Generation

- Your APK file also contains a digital signature certifying that you are the author of this application. Signatures are in the META-INF folder.
- Android applications must be signed before they can be installed on a device.



# Components of an APK (1)

- **Android Manifest file**

- This is the main file that provides the big picture about your app—all of its components, permissions, version, and minimum API level needed to run it.

- **Dalvik executable**

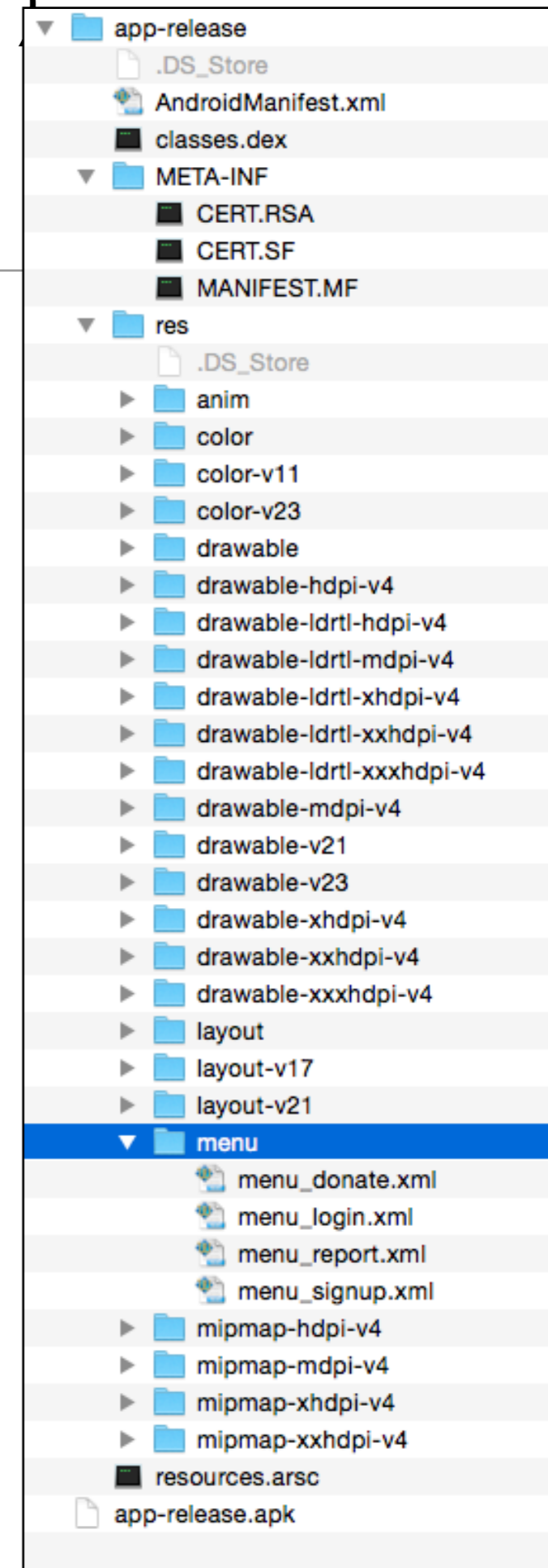
- This is all your Java source code compiled down to a Dalvik executable. The Dalvik executable is the code that runs your application. It is located in a file called classes.dex.

- **Resources**

- Resources are everything that is not code. Your application may contain a number of images and audio/video clips, as well as numerous XML files describing layouts, language packs, and so on. Collectively, these items are the resources.

- **Native libraries**

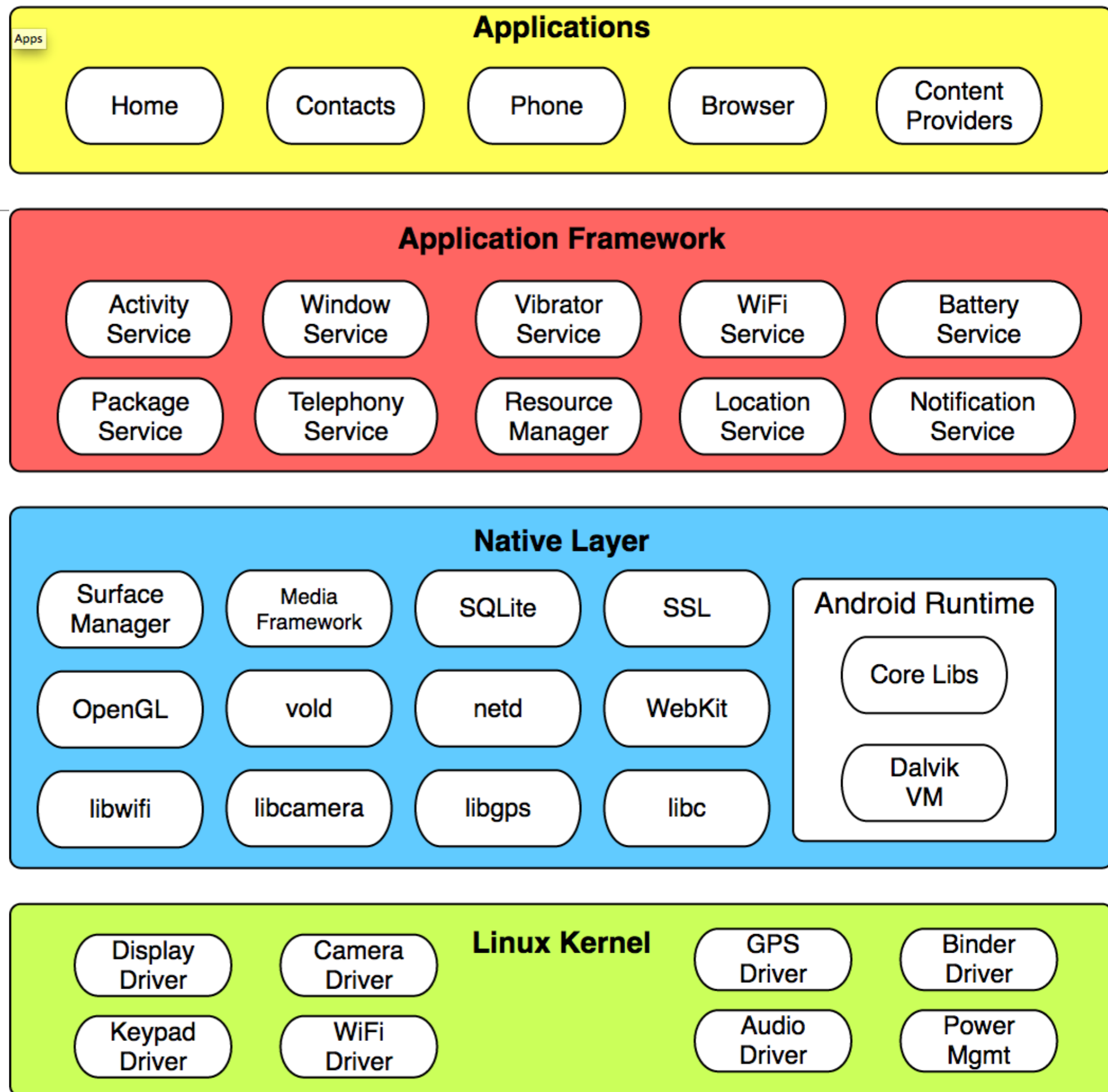
- Optionally, your application may include some native code, such as C/C++ libraries. These libraries could be packaged together with your APK file.





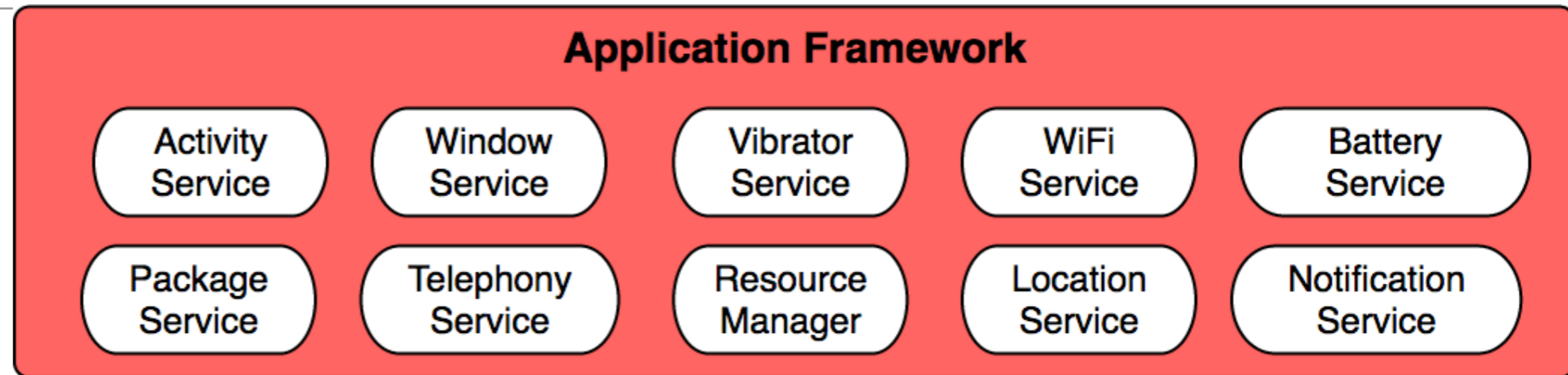
# Android Stack

- Which part should we learn?



# Android Stack

---



- Almost exclusively - the Application Framework
- Learning Resources?

The screenshot shows a web browser window with the URL `developer.android.com/index.html`. The page features a navigation bar with the Android logo, the word "Developers", and menu items for "Design", "Develop", "Distribute", and "Console". A search bar is located on the right. The main content area has a blue background with the heading "Android 6.0 Marshmallow". Below the heading, text announces the availability of the official Android 6.0 SDK and lists new features like runtime permissions, Doze, App Standby, and assist technology. A "Learn more" link is provided. The bottom of the page shows a row of white marshmallows, with the Android robot character holding one. A scroll indicator is visible in the bottom right corner.

Android Developers

Design Develop Distribute Console

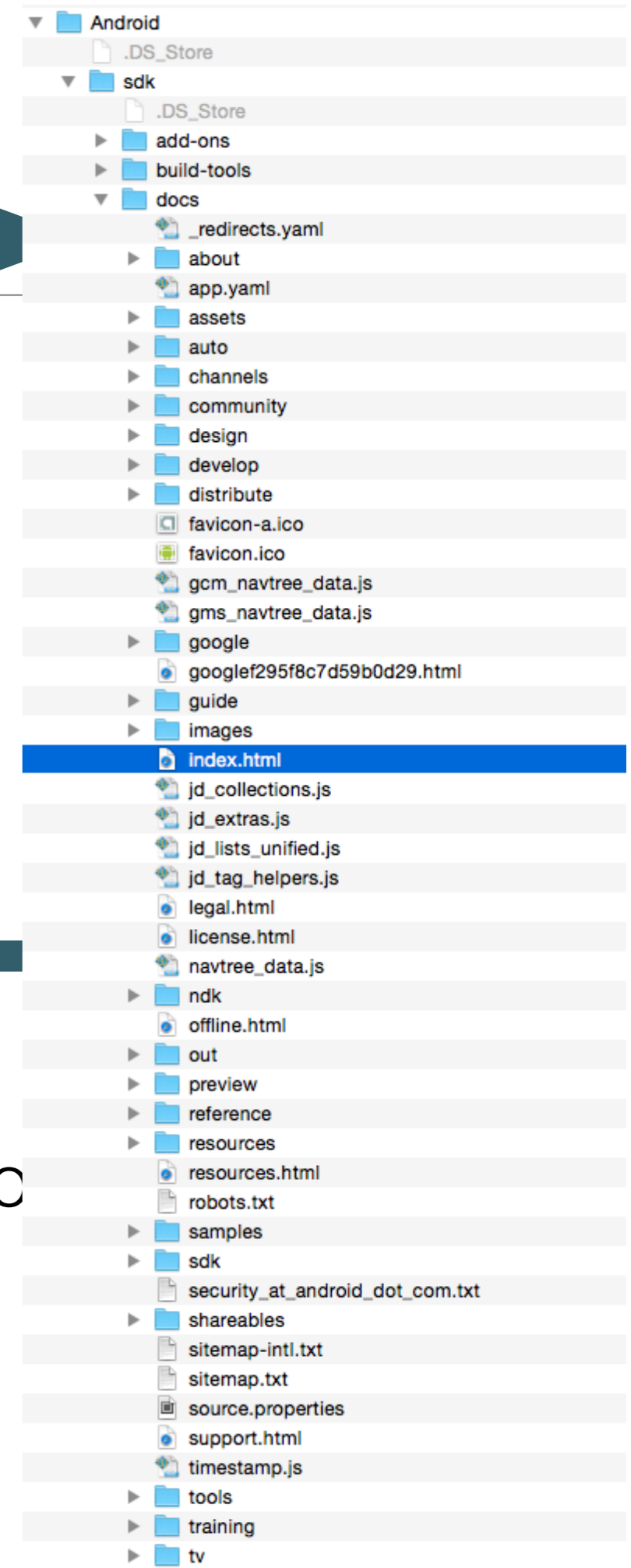
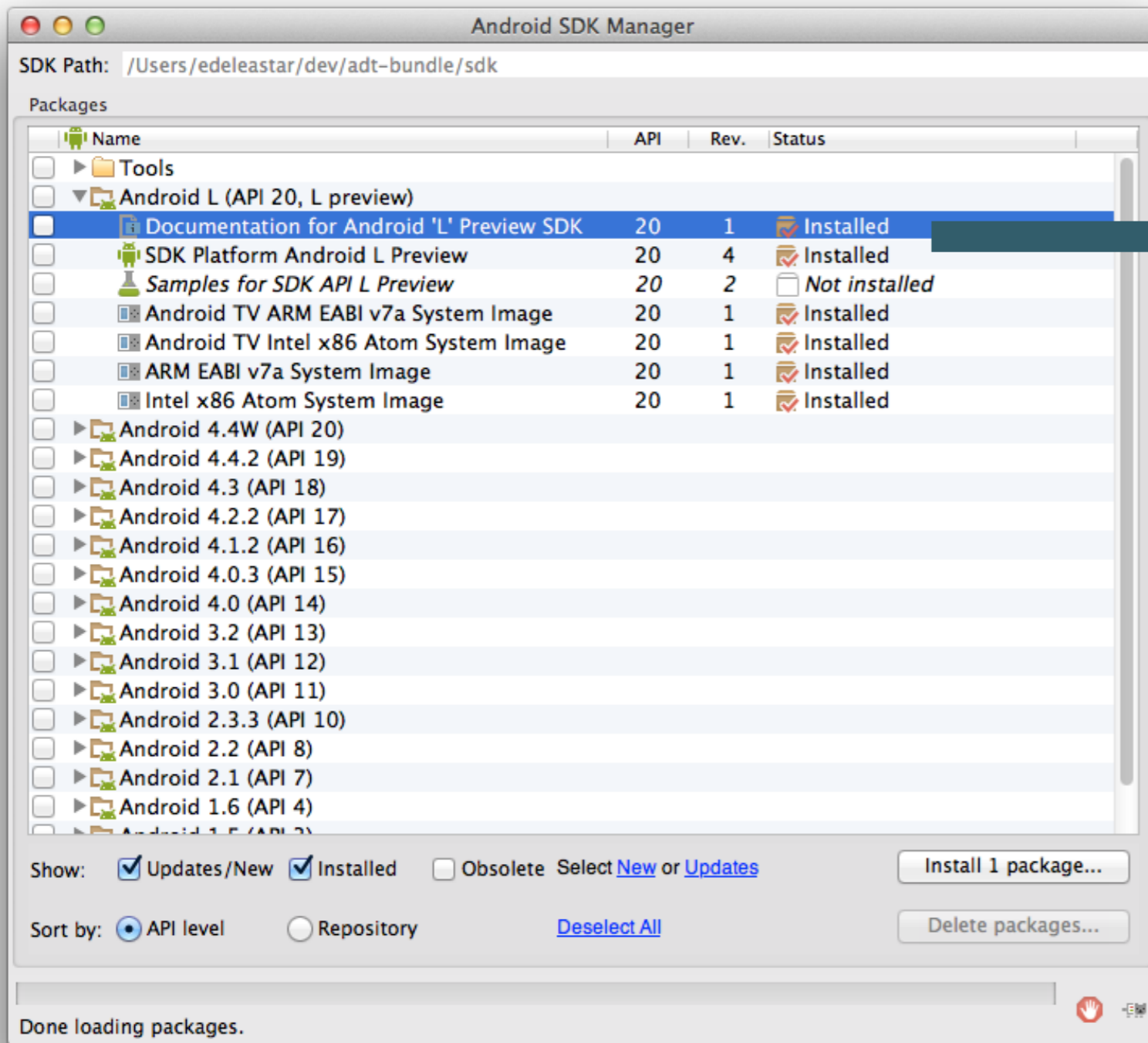
Search

## Android 6.0 Marshmallow

The official Android 6.0 SDK is now available! Explore what's new — runtime permissions, Doze and App Standby power-saving features, new assist technology, and more.

> Learn more

Get the SDK Browse Samples Watch Videos



- Latest version of documentation can be downloaded via the SDK Manager
- Can then be browsed as a static web site

# Design

The screenshot shows a web browser window with the URL `file:///Users/edelestar/dev/Android/sdk/docs/design/index.html`. The browser's address bar includes navigation icons (back, forward, refresh, home) and utility icons (star, chat, search, etc.). The page header features the Android logo, navigation tabs for 'Design', 'Develop', and 'Distribute', and a 'Developer Console' button. The main content area has a large heading 'Up and running with material design' and a paragraph explaining the design metaphor. Below this are sections for 'Introducing material design', 'Downloads for designers', and 'Articles'. The 'Articles' section contains six cards: 'Animation', 'Style', 'Layout', 'Components', 'Patterns', and 'Usability', each with a representative icon. At the bottom, a 'Latest' section is partially visible.

Design | Android Developer

file:///Users/edelestar/dev/Android/sdk/docs/design/index.html

Developers **Design** Develop Distribute Developer Console

## Up and running with material design

Android uses a new design metaphor inspired by paper and ink that provides a reassuring sense of tactility. Visit the [material design](#) site for more resources.

Introducing material design

Downloads for designers

Articles

- MATERIAL DESIGN **Animation**
- MATERIAL DESIGN **Style**
- MATERIAL DESIGN **Layout**
- MATERIAL DESIGN **Components**
- MATERIAL DESIGN **Patterns**
- MATERIAL DESIGN **Usability**

Latest

Getting Started | Android D x Eamonn

developer.android.com/training/index.html

Developers Design **Develop** Distribute Console

Training API Guides Reference Tools Google Services Samples

- Getting Started ^
- Building Your First App v
- Adding the Action Bar v
- Supporting Different Devices v
- Managing the Activity Lifecycle v
- Building a Dynamic UI with Fragments v
- Saving Data v
- Interacting with Other Apps v
- Working with System Permissions v
- Building Apps with Content Sharing v


# Getting Started

Welcome to Training for Android developers. Here you'll find sets of lessons within classes that describe how to accomplish a specific task with code samples you can re-use in your app. Classes are organized into several groups you can see at the top-level of the left navigation.

This first group, *Getting Started*, teaches you the bare essentials for Android app development. If you're a new Android app developer, you should complete each of these classes in order.

If you prefer to learn through interactive video training, check out this trailer for a course about the fundamentals of Android development.

[START THE VIDEO COURSE](#)



Introduction

App Fundamentals

Device Compatibility

System Permissions

App Components

App Resources

App Manifest

User Interface

Animation and Graphics

Computation

Media and Camera

Location and Sensors

# Introduction to Android

Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java language environment. The documents listed in the left navigation provide details about how to build apps using Android's various APIs.

To learn how apps work, start with [App Fundamentals](#).

To begin coding right away, read [Building Your First App](#).

If you're new to Android development, it's important that you understand the following fundamental concepts about the Android app framework:

## Apps provide multiple entry points

Android apps are built as a combination of distinct components that can be invoked individually. For instance, an individual *activity* provides a single screen for a user interface, and a *service* independently performs work in the background.

From one component you can start another component using an *intent*. You can even start a component in a different app, such as an activity in a maps app to show an address. This model provides

## Apps adapt to different devices

Android provides an adaptive app framework that allows you to provide unique resources for different device configurations. For example, you can create different XML layout files for different screen sizes and the system determines which layout to apply based on the current device's screen size.

You can query the availability of device features at runtime if any app features require specific hardware such as a camera. If necessary, you can also declare

Android APIs API level: 22

- android
- android.accessibilityservice
- android.accounts
- android.animation
- android.annotation
- android.app
- android.app.admin
- android.app.assist
- android.app.backup
- android.app.job
- android.app.usage

Select a package to view its members


# Package Index

These are the Android APIs. See all [API classes](#).

<a href="#">android</a>	Contains resource classes used by applications included in the platform and defines application permissions for system features.
<a href="#">android.accessibilityservice</a>	The classes in this package are used for development of accessibility service that provide alternative or augmented feedback to the user.
<a href="#">android.accounts</a>	
<a href="#">android.animation</a>	These classes provide functionality for the property animation system, which allows you to animate object properties of any type. <code>int</code> , <code>float</code> , and hexadecimal color values are supported by default. You can animate any other type by telling the system how to calculate the values for that given type with a custom <a href="#">TypeEvaluator</a> .  For more information, see the <a href="#">Animation</a> guide.



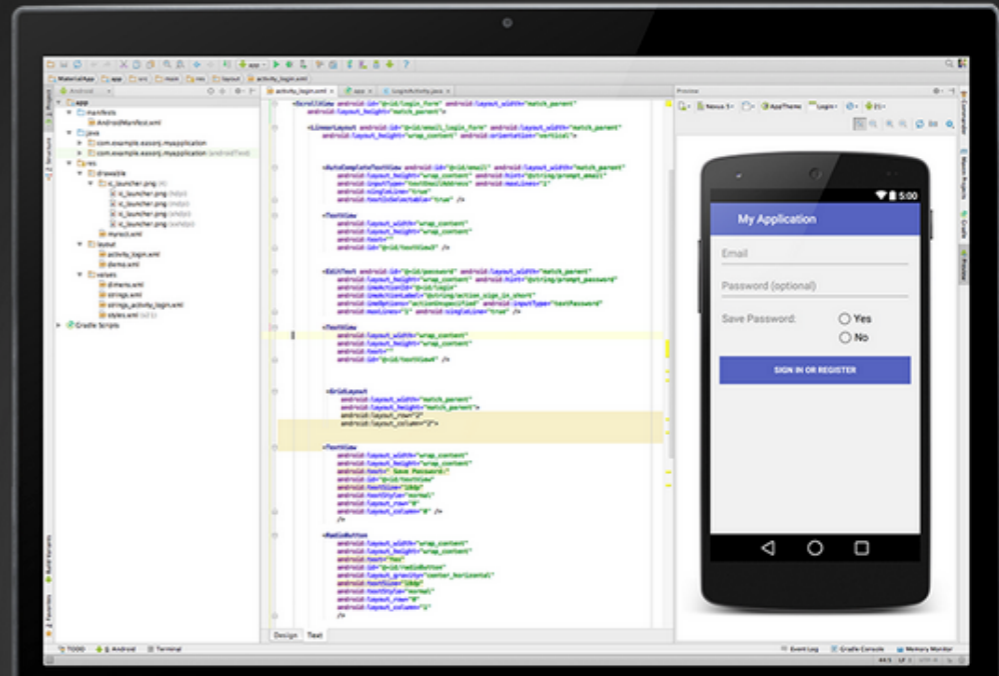
- Download ^
- Installing the SDK
- Adding SDK Packages
- Android Studio v
- Workflow v
- Tools Help v
- Build System v
- Performance Tools v
- Testing Tools v
- Support Library v
- Data Binding Library



# Android Studio

The official Android IDE

- Android Studio IDE
- Android SDK tools
- Android 6.0 (Marshmallow) Platform
- Android 6.0 emulator system image with Google APIs



**DOWNLOAD ANDROID STUDIO FOR MAC**

- [System Requirements](#)
- [Other Download Options](#)
- [Migrating to Android Studio](#)

# Build better apps with Google

Take advantage of the latest Google technologies through a single set of APIs, delivered across Android devices worldwide as part of Google Play services.

Start by setting up the Google Play services library, then build with the APIs you need.

- > [Set up Google Play services](#)
- > [API Reference](#)



About the Samples

What's New

Admin

Background

Connectivity

Content

Input

Media

Notification

RenderScript

Security

Sensors

# Samples

Welcome to code samples for Android developers. Here you can browse sample code and learn how to build different components for your applications. Use the categories on the left to browse the available samples.

Each sample is a fully functioning Android app. You can browse the resources, source files and see the overall project structure. You can copy and paste the code you need, and if you want to share a link to a specific line you can double-click it to get the URL.

## Import Samples from GitHub

Android Studio provides easy access to import Android code samples from GitHub and is the recommended method to retrieve Android code samples.

To import a code sample into Android Studio:

1. In the Android Studio menu, select **File > Import Sample** to open the Import Sample wizard.
2. Select a sample to import and click **Next**.
3. Specify the application name and project location if different from the displayed settings.

# Essentials for a Successful App

A focus on quality should be part of your entire app delivery process: from initial concept through app and UI design, coding and testing and onto a process of monitoring feedback and making improvement after launch.

## Quality Guidelines



### Core App Quality

App quality directly influences the long-term success of your app—in terms of installs, user rating and reviews, engagement, and user retention.



### Tablet App Quality

Tablets are a fast-growing part of the Android installed base that offers new opportunities for your apps.



### Wear App Quality

Wearables are smaller devices that are built for glanceability and require unique apps to provide just the right information at the the right time.

# Recommended Texts

