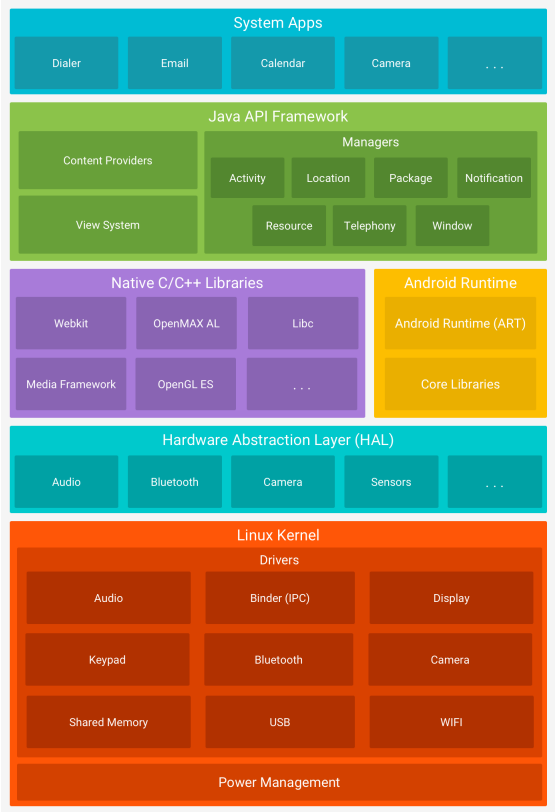
Contents

1 Architecture

Introduction

- A software stack for mobile devices
- Operating system kernel
- Standard middleware
  - Android library support
- Key applications / user interfaces
  - Vendor specific modifications



Structure

- LK: threading, low level memory management, driver
- HAL: libs for hardware module
- AR: virtual machine
- NCL: fundamental core functionalities
- API: programming interface
- APP: system apps can be customised.

1.1 Kernel

1.1.1 Modifications

Modifications made by android to linux OS

- wakelocks
  - Keep the phone awake
- binder
  - Interprocess communication mechanism, and remote method invocation system.
  - One Android process can call a routine in another Android process
- ashmem
  - Android Shared Memory
  - A component of the Android operating system that facilitates memory sharing and conservation
- LMK kills processes when memory is low
- alarm manager
  - Wakes up the phone when necessary

1.2 Hardware support

- Bluetooth - BlueZ
- GPS Manufacturer provided libgps
- Wifi `wpa-supplicant`
- Display Standard framebuffer driver
- Keyboard Standard input event
- Lights Manufacturer provided liblights.so
- Audio Manufacturer provided libaudio.so
- Camera Manufacturer provided libcamera.so
- Power Management wakelocks kernel patch
- Sensors Manufacturer provided libsensors.so
- Radio Manufacturer provided libril.so

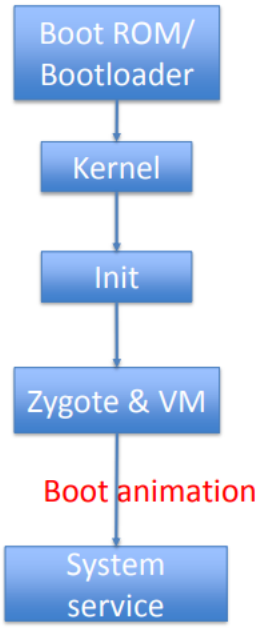
1.3 Security

Applications are sandboxed

- A security mechanism for separating running applications and data
- This allows applications run in a different context, so if one app crashed, others can stay uneffected
- On Android, each app runs as its own **user**, which guarantees that different users are unable to interfere with each other, access each others files and so on. Root can access the entire system
- Own process, own VM, own UID/AID for different app

2 System boot process

- **Boot ROM/ Bootloader:** Load bootloader into RAM, detect external RAM, setup network, memory, etc.
- **Kernel:** Setup cache, protected memory, scheduling and loads drivers.
- **Init:** Mounts directories like /sys , /dev or /proc Runs init.rc script
- **Zygote & VM:** Enables code sharing across the android VM for quick start of separate VM for different apps preloadClasses(), preloadResources()
- **System service:** Power manager, activity manager, telephony registry, package manger, context manager, system contact providers, etc.



3 Zygote

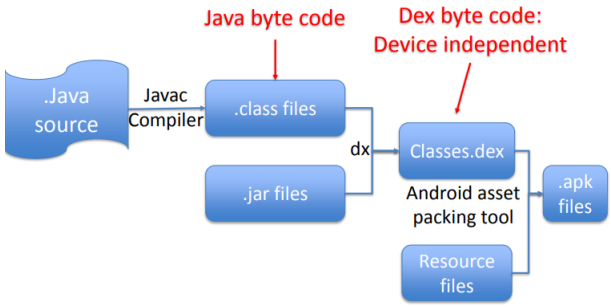
- Initialised process that has all core libraries linked in
- Load all java.*, android.* classes at boot time
- Initially create a single android VM process Referencing classes loaded above
- When user runs an application:
  - Creates a copy of itself in a separate address space
  - **Does not** copy memory, instead refers to original memory until modified
  - Because each container recieves a map, these resources are shared between applications, eliminating the need for each new fork of the VM to keep its own copy of classes and resource.

4 Memory

In many places, Android shares the same dynamic RAM across processes using explicitly allocated shared memory regions. Android uses paging and mmap **instead of providing swap space**, which means any memory your application touches **cannot be paged out** unless you **release all references**.

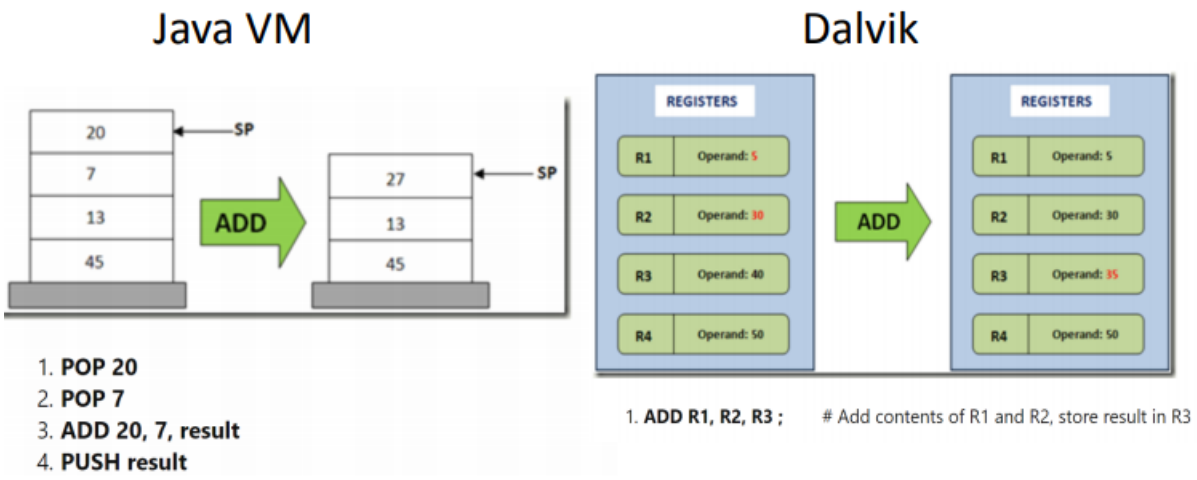
5 Android compilation

- Applications are written in Java
  - Run on Googles own VM Dalvik/ Android Run Time
  - Uses its own bytecode (DEX) format
- Code compiled using standard Java tools then convert to DEX format
  - Multiple class files can be put in to a single .dex file.
- Code, data and resource files packed into a .apk file



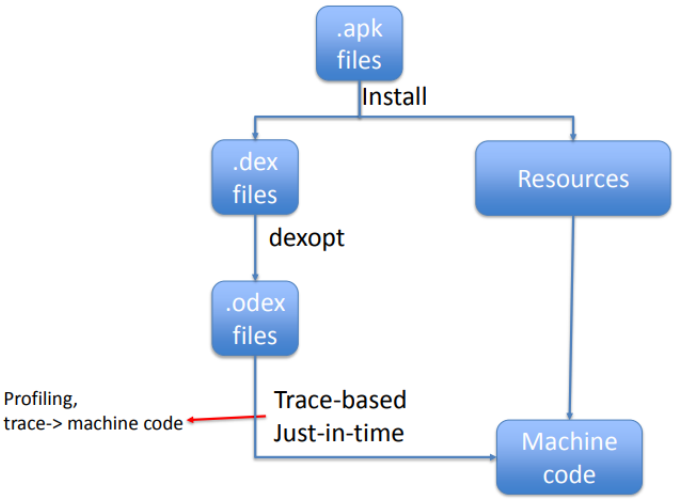
5.1 Dalvik

- Dalvik architecture is *register based* rather than *stack based*, which make's it optimised to use less space
- Execute its own Dalvik byte code rather than Java bytecode
- Dalvik interprets .dex files
  - Post-processes .class files
  - Size reduction
  - JIT compilation to native ARM instructions
- **Target**: slow cpu, no swap, low RAM, battery powered device.
- This approach allows us to execute **less** instructions, but the instructions are **larger**

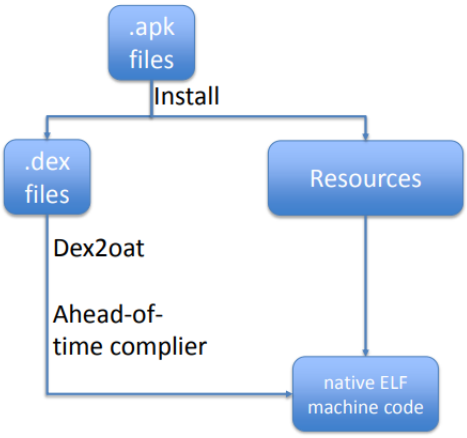


6 Runtime

6.1 Environment with Darvik



6.2 Environment with ART



7 ART

Android run time (ART) is the next version of Dalvik. It has two main new features. Ahead-of-Time (AOT) compilation, which improves speed (particularly startup time) and reduces memory footprint (no JIT) and improved Garbage Collection (GC)

- **Pros**
  - Apps run faster as DEX bytecode translation done during installation
  - Reduces start-up time of applications as native code is directly executed
  - Improves battery performance as power utilised to interpreted byte codes line by line is saved
- **Cons**
  - App Installation takes more time because of DEX bytecodes conversion into machine code
  - More internal storage is required to store the fully converted machine code at installation

8 Android 7.0

Android 7.0 adds a JIT compiler with code profiling to ART that constantly improves the performance of Android apps as they run

9 Programming Models

9.1 Comparison

Traditional OS applications:

- A single entry point
- Main OS loads the program into a process and executes it

Java applications:

- A Java VM is instantiated
- Loads all classes used by the application
- Executes main

Component based model

- Multiple application entry points
- The point through which the system can enter the application

9.2 Android components

Activities

- Dictate the UI and handle the user interaction to the smart phone screen.

Services

- Mechanism for doing something long-running in the background
- Handle background processing associated with an application.

Broadcast Receivers

- Respond to broadcast messages from the OS / other apps

Content Providers

- Make data available to / make use of data from other apps

placeholder