Dalvik and ART

Dalvik and ART

- In this lecture we will explore the differences between Dalvik and ART the two runtimes that can be used by Android
  - Dalvik is the original run time and has been present since version 1.0
  - ART (Android RunTime) is an alternative that has been made available since version 4.4 and will be replacing Dalvik
    - Is the only runtime available in 5.0

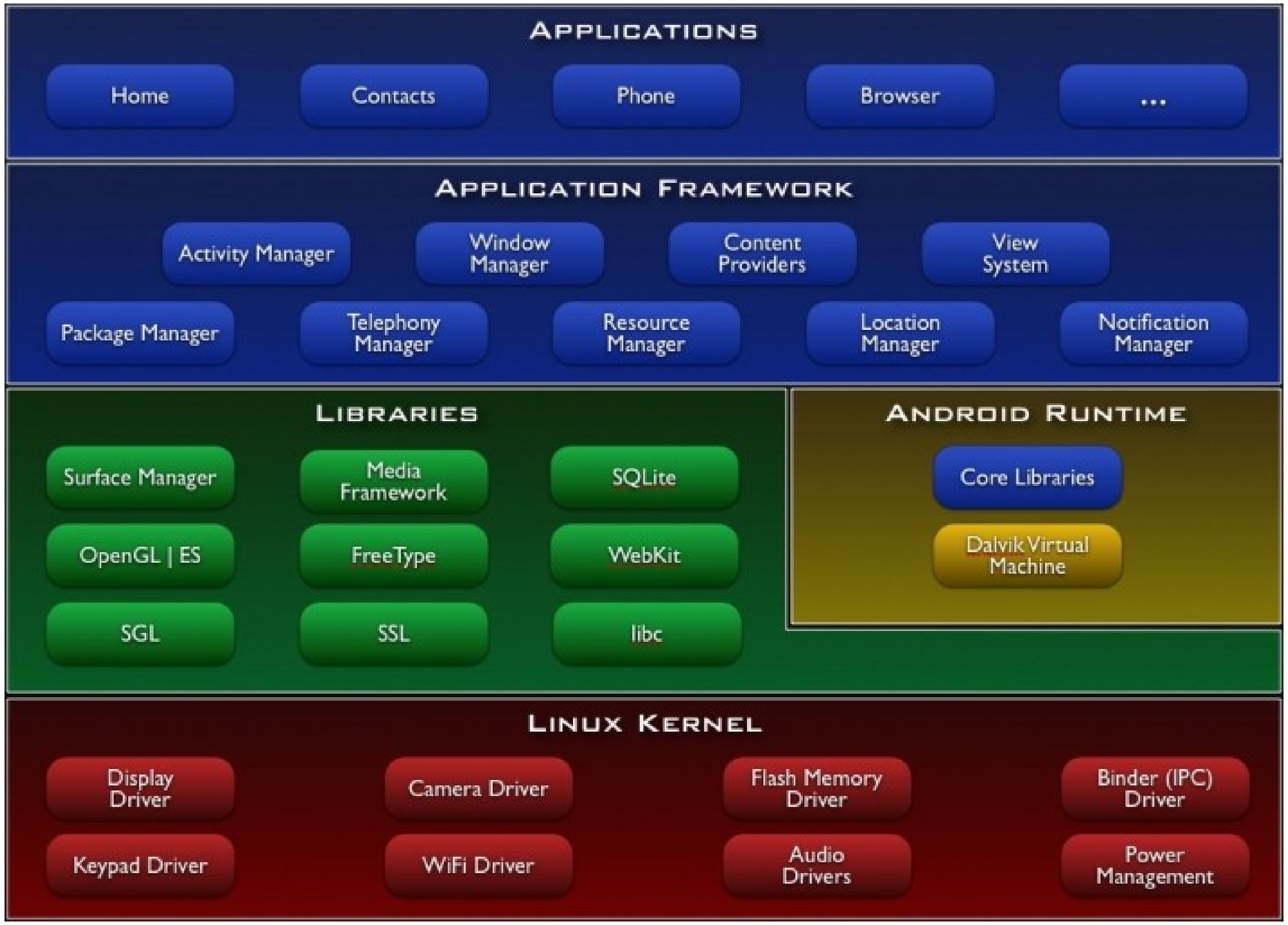
Dalvik as a process virtual machine

- Dalvik is a process virtual machine much in the same way that the java virtual machine is also a process virtual machine
  - Designed to hide the differrences in architecture between different processors
- Was chosen because there was no set hardware CPU architecture for devices
  - Most devices are ARM based but anything else including x86 and MIPS and run Android too

Dalvik as a process virtual machine

- By using an intermediate instruction set the virtual machine can translate it to the instruction set of the processor
  - Provided there is a full dalvik implementation on that architecture
- Enables the same code to run on any processor regardless of instruction
  - However, there are performance penalties as we will see later.

Dalvik Architecture



Dalvik Architecture

- The Linux kernel is there to manage all the normal day to day OS operations such as process scheduling/security/privileged operations
  - The idea here is to reuse a component that is open source and long standing as it is reasonably reliable and has good performance
- Any bug fixes in mainline linux kernel in server or desktop use will find their way into Android
  - Instead of building an OS from scratch and encountering the same bugs again and again

Dalvik Architecture

- In the Dalvik run time each Android application is given its own process and its own instance of the Dalvik VM
  - Dalvik is designed to switch between VMs quickly and efficiently. Also a security measure as we will see later
- All underlying functionality such as threading and memory management is handed off to the linux kernel
  - Something its quite effective at doing already

Dalvik Architecture

- The native libraries implement core functionality that will be used by all applications
  - Includes things like SQLite and OpenGL ES. These are wrapped in Java classes to provide the necessary functionality.
  - Enables some retrieval in performance from that lost by running interpreted instead of native code
  - Again mostly standard stuff that is stable and long existing in the open source world. So similar results to the kernel.

Dalvik Architecture

- The application framework consists of the whole Android API. This is what you will interact with when you are developing applications
  - Provides a single unified java interface to access all components of android.
- Is there to hide the low level implementation details of individual libraries and the interaction with the kernel
  - So regardless of the development platform your application will run on any device.

Minimum Spec Required

- The minimum specification that was required by dalvik at the time was very modest compared to today's devices
  - Initially CPUs were restricted to ARM but was opened up to x86 and MIPS based architectures
  - Minimum memory was 128MB RAM and 256 MB flash based storage (for OS + apps)
    - To be backed up by external storage (generally SD card based)

Minimum Spec Required

- Screens were only expected to be QVGA at the time
  - 160x120 Resolution
  - Dire compared to today's standards

What Dalvik was required to support

- Dalvik was written to support the following set of objectives. These are what you are limited by with a mobile device
  - Limited CPU and RAM, battery power
- No swap space.
  - Was used to suppliment main memory in desktop/server environments but with limited slow space on mobile its not feasible

What Dalvik was required to Support

- Diverse set of devices
  - Ranging from cheap low power devices all the way up to expensive high powered smartphones.
- Sandboxed application runtime
  - For security reasons that we will see in a later lecture

How processes are separated from each other

- To sepearate processes from each other they are run within their own dalvik virtual machine
  - They do not know that their are other virtual machines or applications running on the device
- Means that all communication between applications is mediated by android itself.
  - Again another securtity measure that we will discuss in detail later

Stack based virtual machines vs register based virtual machines

- Stack based machines use a stack based in memory to store operands and data.
  - The advantage of this approach is that code is small and easy to write using a reverse polish notation
- Register based machines use large list of registers for storing data and performing operations
  - While this requires more code to write applications (load store operations mainly) there is a potential gain in performance as the code on a register machine can sometimes be quicker

Difference between Dalvik and Java Hotspot

- Dalvik is an example of a register based machine
  - This model was chosen as the performance benefits were considered necessary at the time given the low power of devices
- Java Hotspot is an example of a stack based machine
  - Built for easy application building and small code sizes. Use of a JIT helps with optimising performance of code

SDK and NDK

- Because of the need to have a process virtual machine android provides a Native development kit
  - Provided in cases of applications that require native speed for all or part of their code
- C++ based kit that compiles directly to native code
  - Removes the need for the interpreter as no translation is necessary

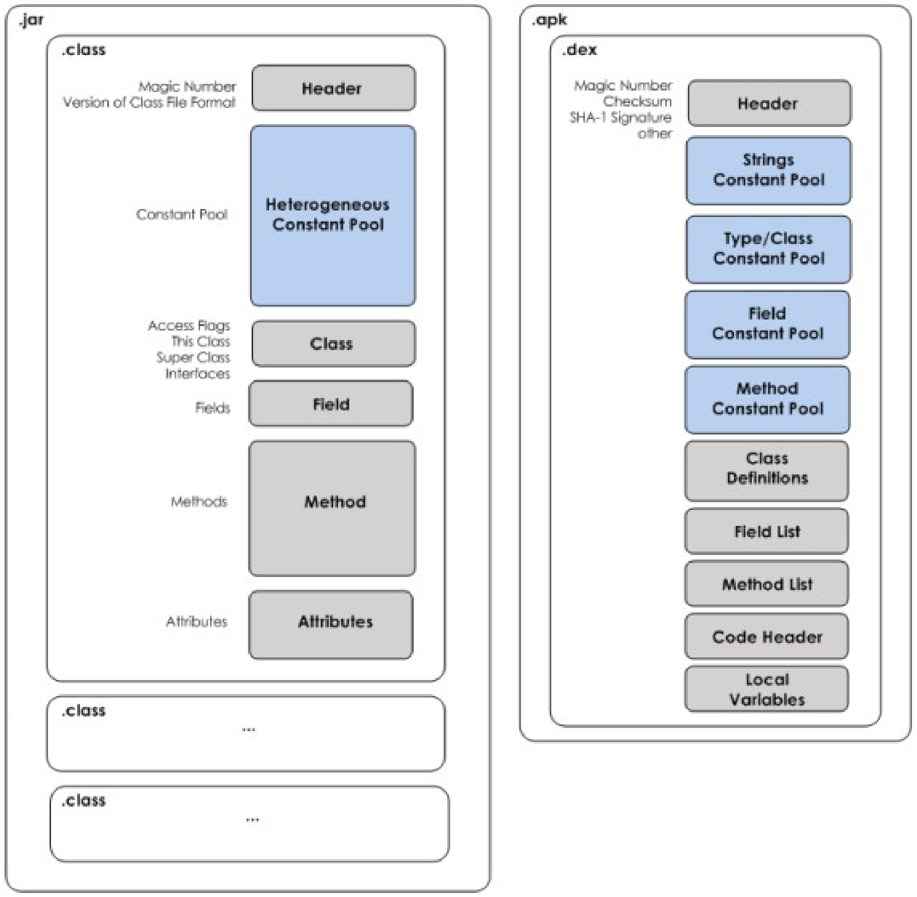
SDK and NDK

- However, it should be avoided unless **really** necessary
  - As it restricts your app to a single instruction set
  - It doesn't provide you with the full facilities that the SDK provides through java
  - And with ART coming in there is even less reason to use this to do any development

Dex file format

- Again due to the limitations on space Android does not use the jar format for creating applications
  - Considered to take too much space on a space limited system
- The dex file format was designed to do the same job as the jar format
  - But redesigned elements to reduce the space required by a jar. (smaller size == more applications)

Dex and Java jar comparison



How a jar is put together

- The jar format consists of a number of .class files and an overall manifest.
  - Each class is kept seperate however they all share the same structure
- Each class contains a header
  - Information such as format type, the magic numbers to ID a java file, which version of java it is built to.

How a jar is put together

- A class section for denoting the relationship between this class and other classes
  - Such as the super class, access flags, and implemented interfaces.
- Then we have standard fields and methods
- And finally a Heterogenous Constant Pool.

How a jar is put together

- The constant pool is heterogenous because is contains constants of many different types.
  - Each constant must store its type as well as its constant value.
  - But because the constant pool is private to a class file there is the possiblitity of having multiple definitions of the same constants in different class files.

How a dex is put together

- A dex file like a jar contains a number of class files however, they are combined together into a single space
  - That contains all class names and method names.
- The idea here is that it is to reduce redundancy in particularly with constant pools
  - To reduce the amount of data that needs to be stored.

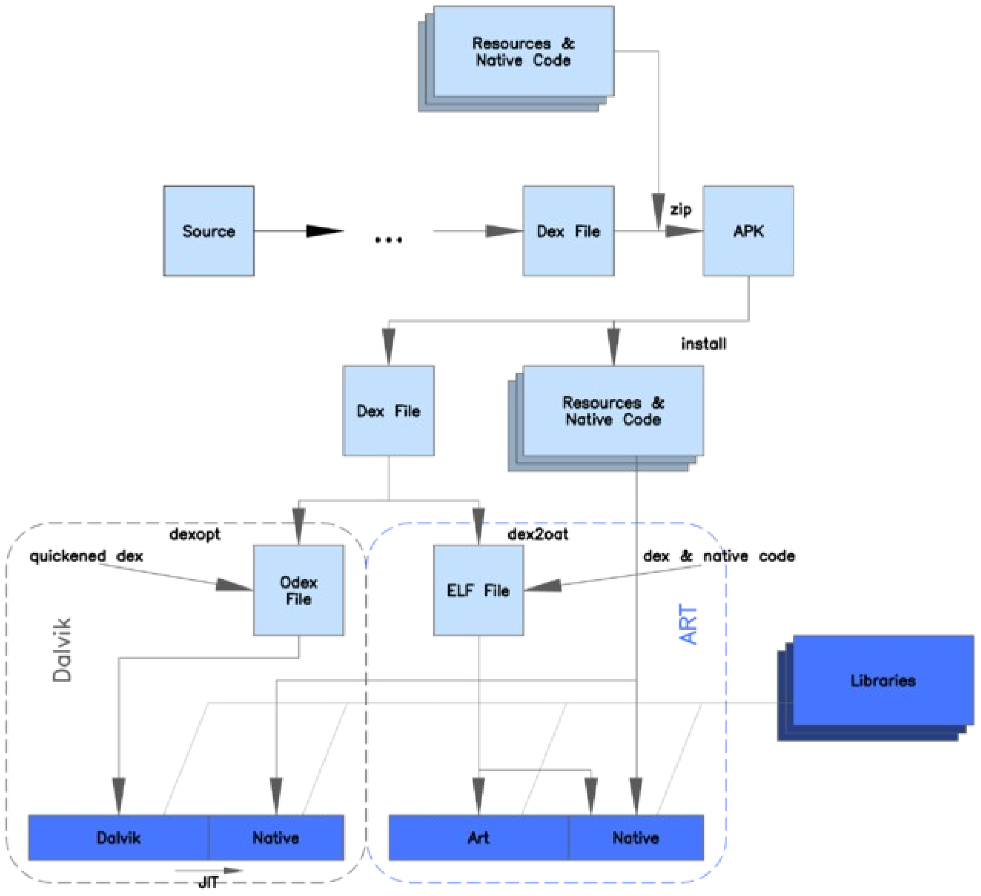
Constant pools

- A dex unlike jars will use a number of homogenous constant pools as a way of removing type information from constants
  - By reducing the type information it is possible that the size of the dex can be vastly reduced
- For example in Davide Hringer's paper on dalvik the constant pool on average in a jar took 61% of the file size
  - Significant savings were made here by using seperate pools and indexing

Android Run Time

- Starting with Android 4.4 the Android Run Time will be replacing Dalvik
  - With android 5.0 it is now the only choice as Dalvik has been removed
- Makes a number of changes to the run time that aim to improve performance and battery life
  - We will explore those here

Android Run Time



Main difference between ART and Dalvik

- ART introduces Ahead of Time compilation for all applications that are installed on the device
  - All dex byte code is compiled to native machine instructions. To reduce the overheads of interpretation
  - Improves performance and reduces battery usage
  - However increses the install times for applications with the introduction of the compilation phase

Performance and battery benefits

- Also eliminates the need for a full process virtual machine.
  - Saves on memory and resources
- However, applications must still be sandboxed as before
  - No idea on how android does it but something like a chroot jail would work well here.

Garbage collection in Dalvik

- Dalvik's garbage collection mechanism causes two application pauses.
  - The first is an enumeration phase. To discover all the objects in use by an application.
  - The second is a marking phase. To denote all those objects that are reachable. Anything that is not reachable is removed.
- Gives two points in the application where there would be stutter.

Garbage Collection in ART

- ART offloads some of the Garbage collection process to the application itself
  - Means less work for the garbage collector thus reducing the number of pauses
- Also introduces a Large Object Space that is kept seperate from the main heap
  - Thus the heap is not as fragmented and the GC doesn't need to be called as often