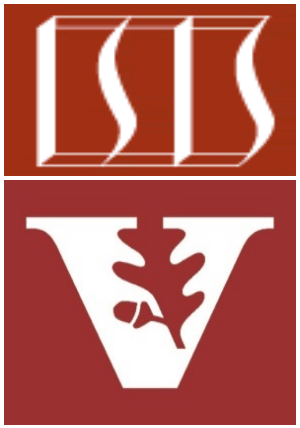
Infrastructure Middleware (Part 2): Android Runtime Execution Environment

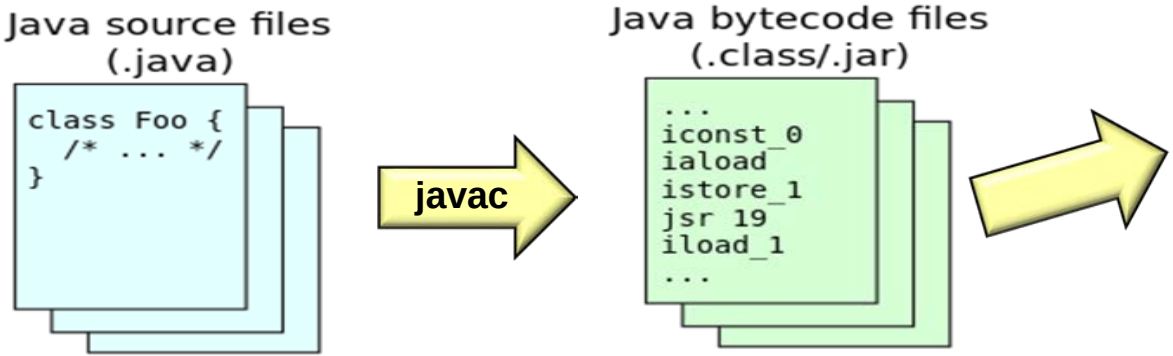


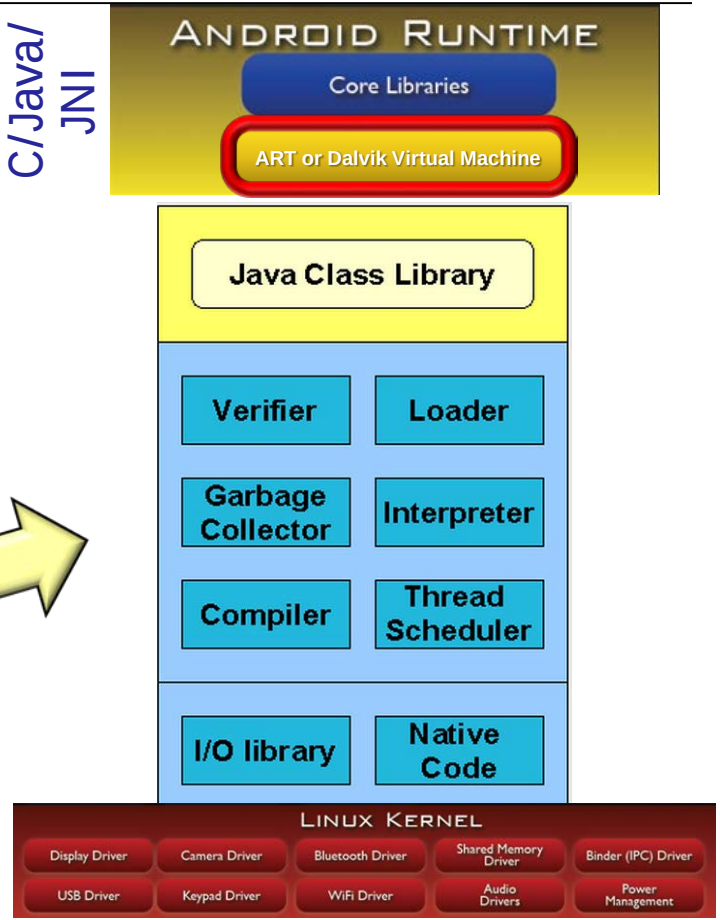
Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Institute for Software
Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA

Learning Objectives in this Part of the Lesson

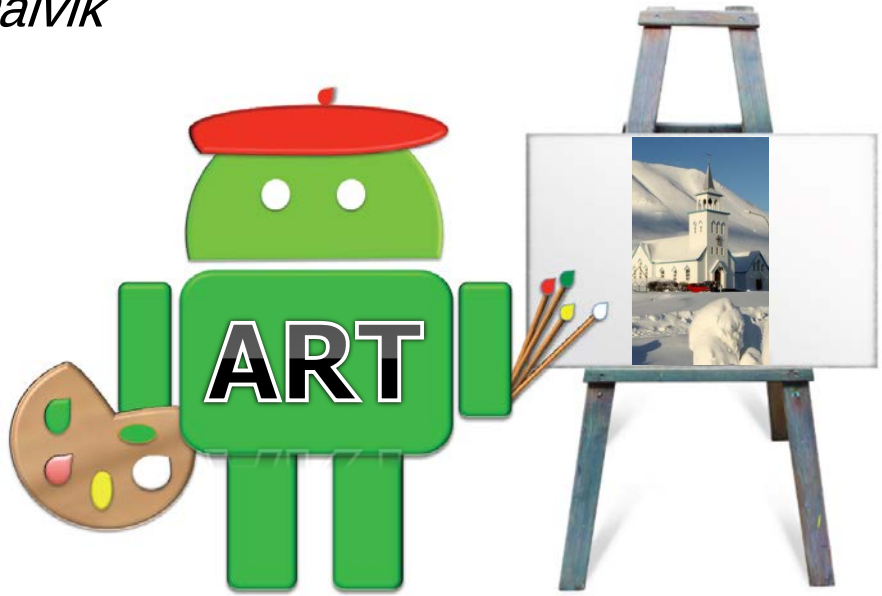
- Understand the role of the execution environment in Android's Runtime layer





Learning Objectives in this Part of the Lesson

- Understand the role of the execution environment in Android's Runtime layer
- Know the two execution environments that have been part of Android's Runtime: *ART & Dalvik*



Learning Objectives in this Part of the Lesson

- Understand the role of the execution environment in Android's Runtime layer
- Know the two execution environments that have been part of Android's Runtime: *ART & Dalvik*

Apps rarely access ART or Dalvik directly, but it's useful to understand what they do

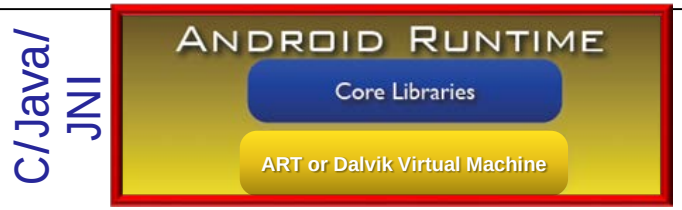
Overview of the Android Runtime's Execution Environment

Android Runtime's Execution Environment

- Android's Runtime layer is largely used to execute Java apps on mobile devices

Android Runtime's Execution Environment

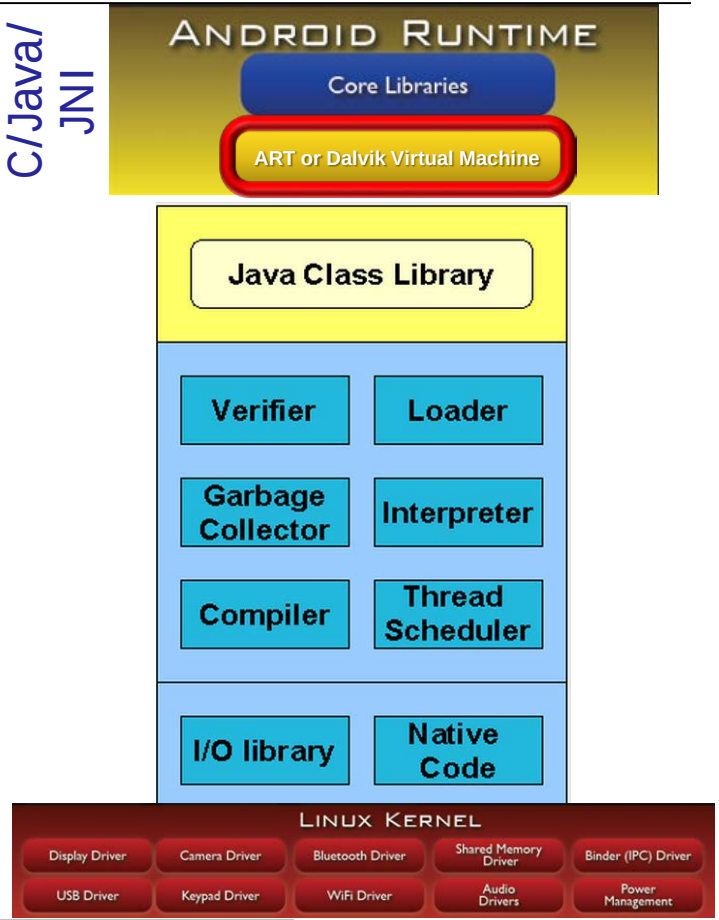
- Android's Runtime layer is largely used to execute Java apps on mobile devices
  - These apps can also now run in the Chrome browser on laptops & desktops



See arstechnica.com/gadgets/2016/04/it-looks-like-the-google-play-store-is-headed-to-chrome-os

Android Runtime's Execution Environment

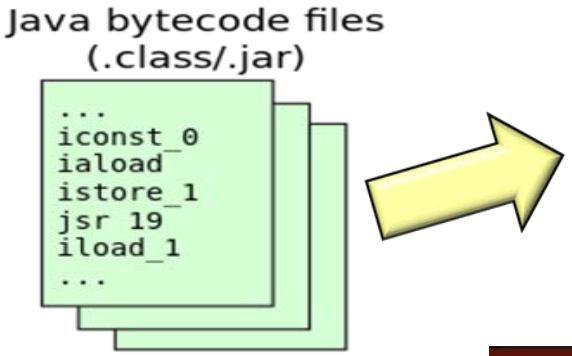
- Android's Runtime layer contains an execution environment that resides atop the Linux kernel

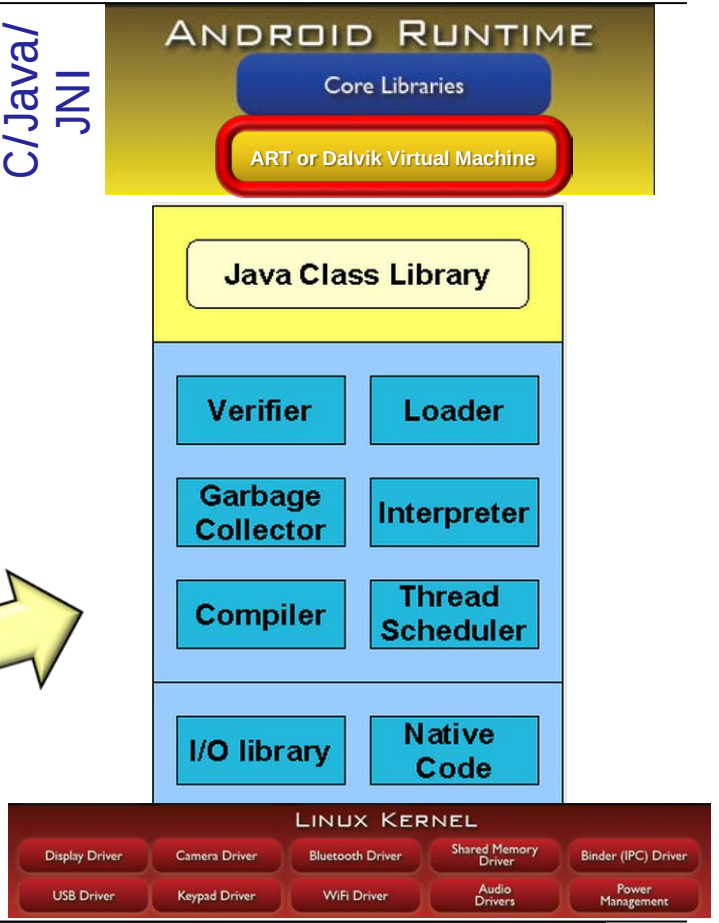


See en.wikipedia.org/wiki/Virtual_machine

Android Runtime's Execution Environment

- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process

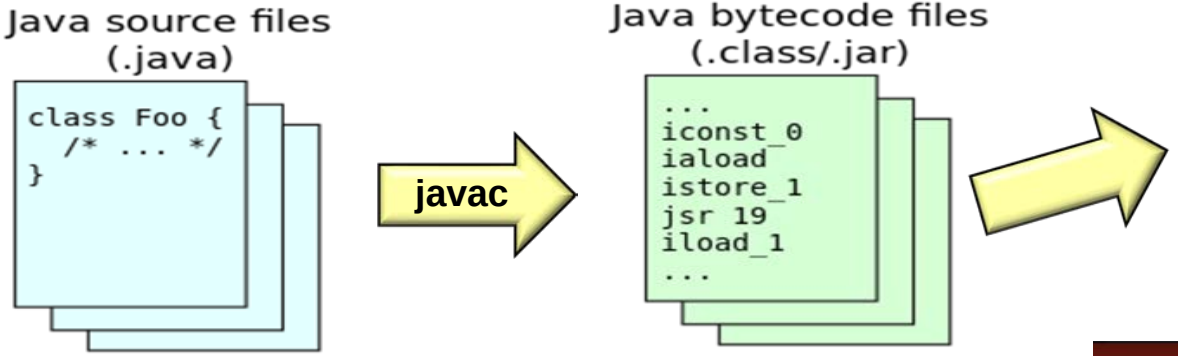




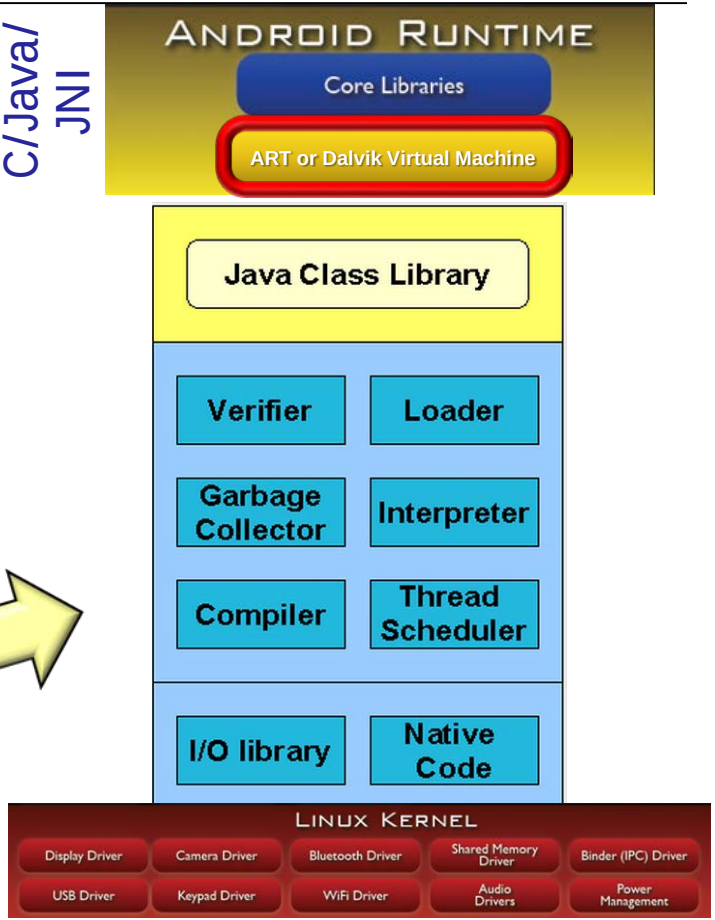
See en.wikipedia.org/wiki/Virtual_machine#Process_virtual_machines

Android Runtime's Execution Environment

- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
    - This code generated is from Java source files by the javac compiler



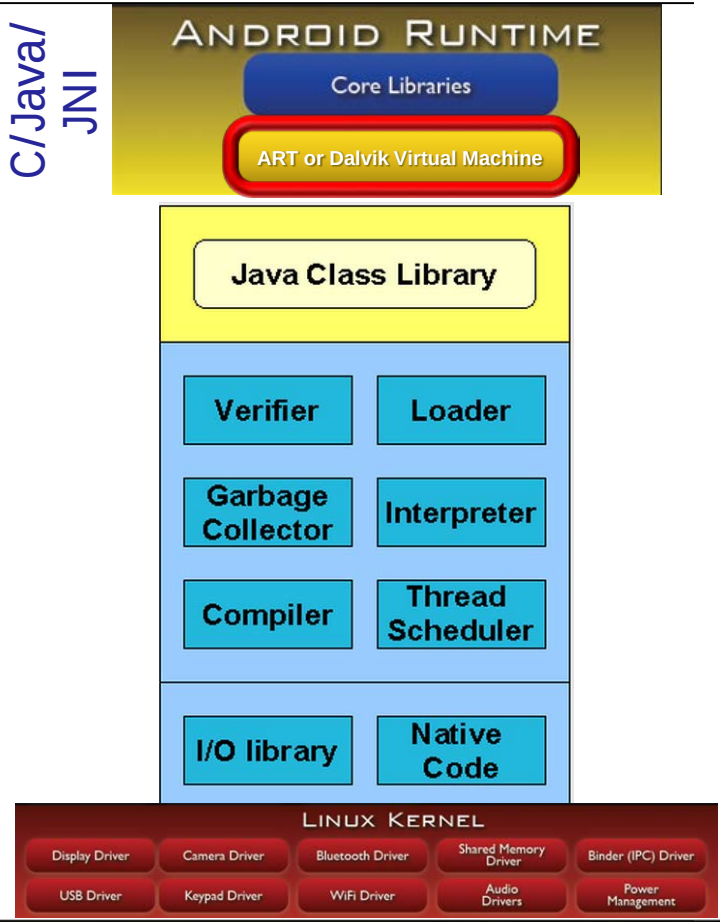
C/Java/ JNI



See source.android.com/source/jack.html

Android Runtime's Execution Environment

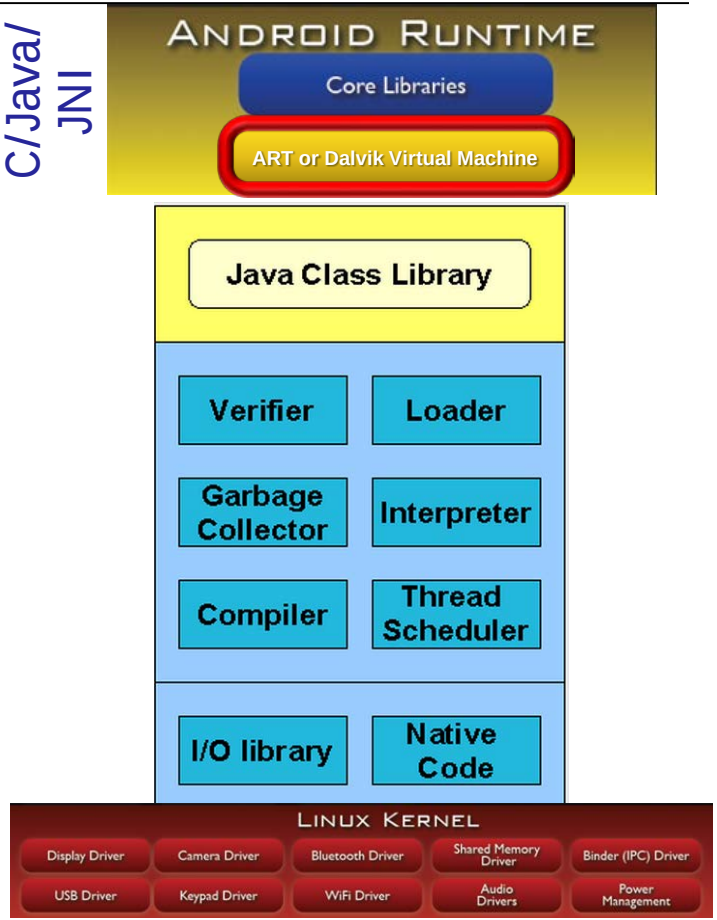
- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits



See developer.android.com/guide/components/processes-and-threads.html#Processes

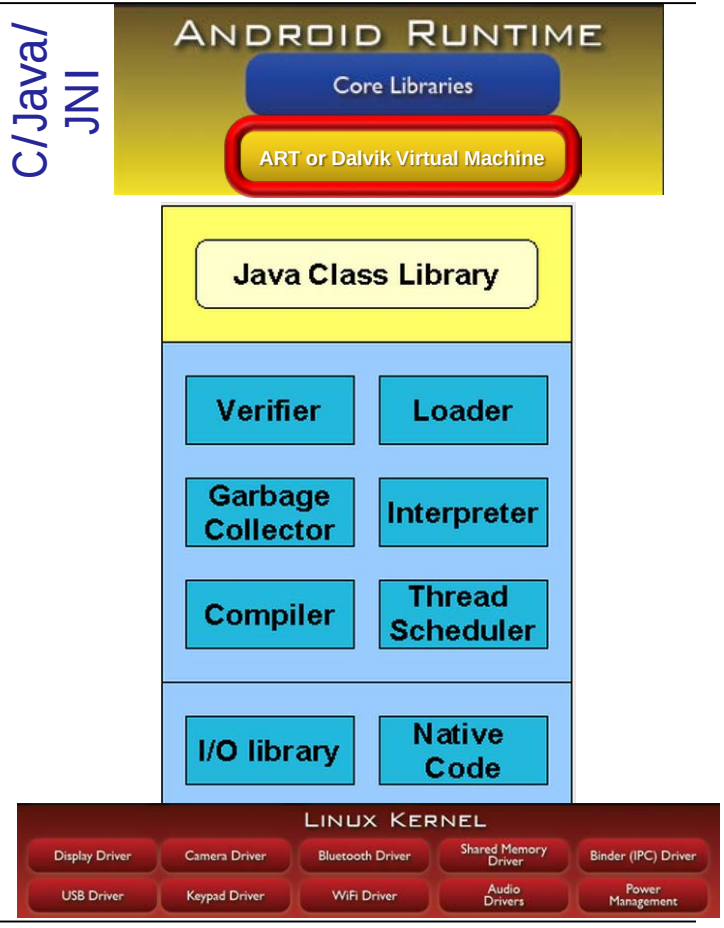
Android Runtime's Execution Environment

- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
    - A process can run apps or system services



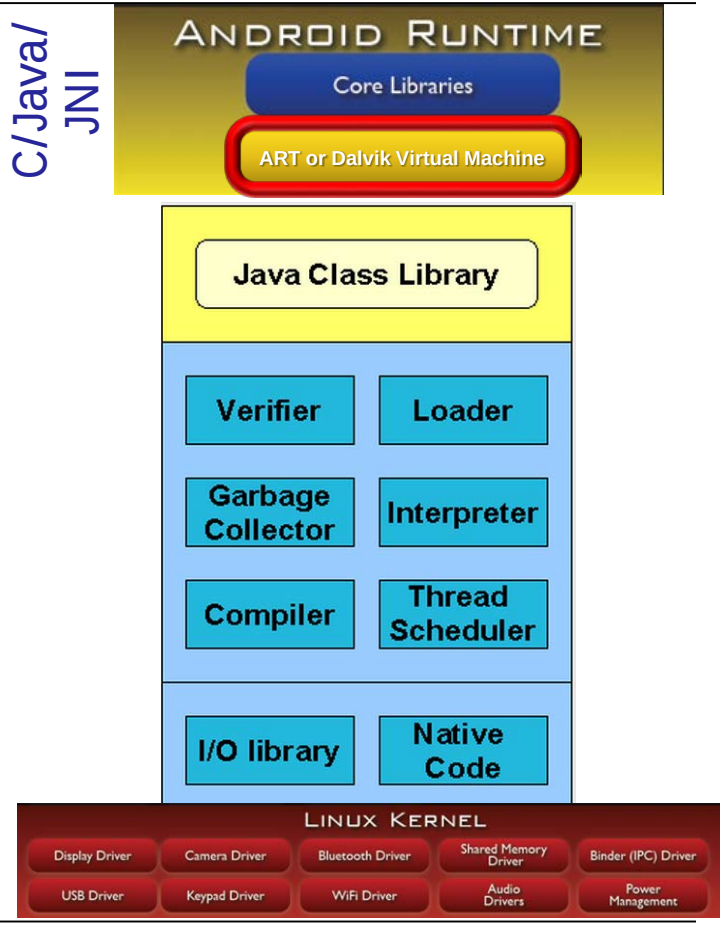
Android Runtime's Execution Environment

- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
  - It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware



Android Runtime's Execution Environment

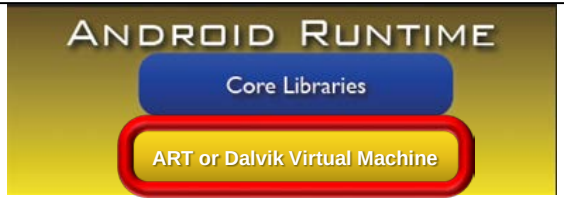
- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
  - It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware
    - e.g., Intel x86, ARM, emulator, etc.

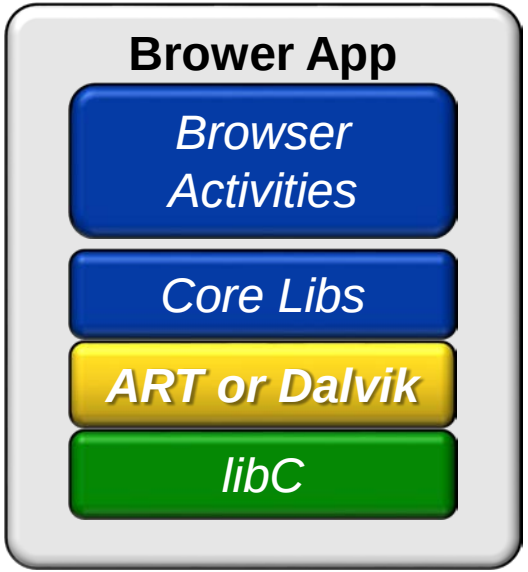


Android Runtime's Execution Environment

- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
  - It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware
- Android apps typically run in their own process & own instance of the execution environment

C/Java/ JNI



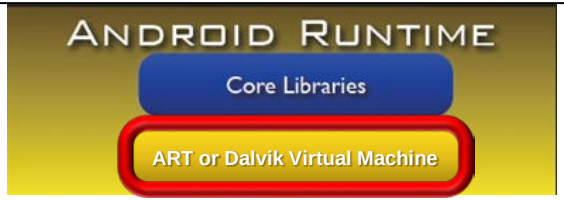


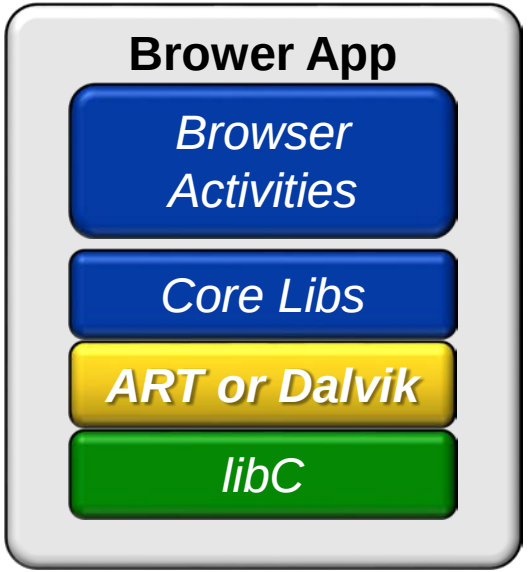
See developer.android.com/guide/topics/processes/process-lifecycle.html

Android Runtime's Execution Environment

- Android's Runtime layer contains an execution environment that resides atop the Linux kernel
  - Executes app byte code and/or native code (typically) inside a single Linux process
  - This environment is created when a process starts & is destroyed when the process exits
  - It enhances portability & productivity by shielding higher layers of Android from low-level details of the underlying hardware
- Android apps typically run in their own process & own instance of the execution environment
  - Apps can also share the same process

C/Java/ JNI



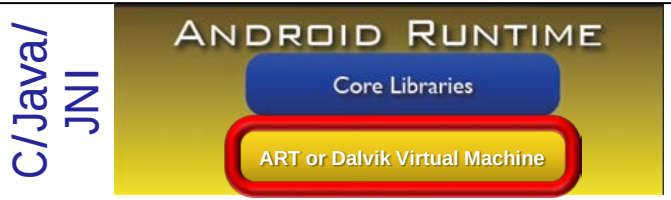


See developer.android.com/guide/topics/manifest/application-element.html#proc

Evolution of the Android Execution Environment

Evolution of the Android Execution Environment

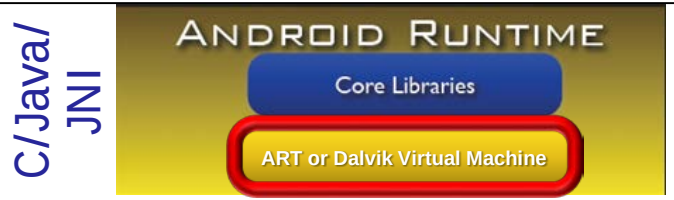
- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)



See www.momob.in/2010/general/bhavis/difference-of-dvm-and-jvm-in-the-android-world

Evolution of the Android Execution Environment

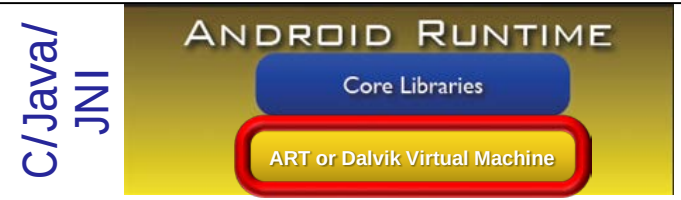
- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode



See en.wikipedia.org/wiki/Dalvik_(software)

Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
    - Dalvik uses a "register machine" model



Dalvik bytecode

General design

IN THIS DOCUMENT

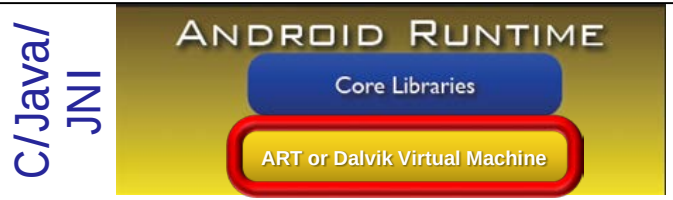
General design
Summary of bytecode set
packed-switch-payload format
sparse-switch-payload format
fill-array-data-payload format
Mathematical operation details

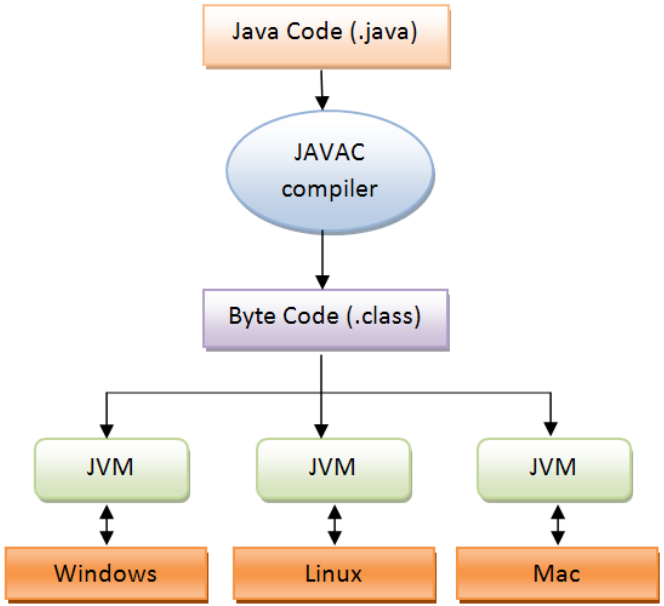
- The machine model and calling conventions are meant to approximately imitate common real architectures and C-style calling conventions:
  - The machine is register-based, and frames are fixed in size upon creation. Each frame consists of a particular number of registers (specified by the method) as well as any adjunct data needed to execute the method, such as (but not limited to) the program counter and a reference to the `.dex` file that contains the method.
  - When used for bit values (such as integers and floating point numbers), registers are considered 32 bits wide. Adjacent register pairs are used for 64-bit values. There is no alignment requirement for register pairs.
  - When used for object references, registers are considered wide enough to hold exactly one such reference.
  - In terms of bitwise representation, `(Object) null == (int) 0`.
  - The *N* arguments to a method land in the last *N* registers of the method's invocation frame, in order. Wide arguments consume two registers. Instance methods are passed a `this` reference as their first argument.
- The storage unit in the instruction stream is a 16-bit unsigned quantity. Some bits in some instructions are ignored / must-be-zero.
- Instructions aren't gratuitously limited to a particular type. For example, instructions that move 32-bit register values without interpretation don't have to specify whether they are moving ints or floats.
- There are separately enumerated and indexed constant pools for references to strings, types, fields, and methods.
- Bitwise literal data is represented in-line in the instruction stream.
- Because, in practice, it is uncommon for a method to need more than 16 registers, and because needing more than eight registers *is* reasonably common, many instructions are limited to only addressing the first 16 registers. When reasonably possible, instructions allow references to up to the first 256 registers. In addition, some instructions have variants that allow for much larger register counts, including a pair of catch-all `move`

See source.android.com/devices/tech/dalvik/dalvik-bytecode.html

Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
    - Dalvik uses a "register machine" model
      - In contrast, the Java platform uses a "stack machine" model

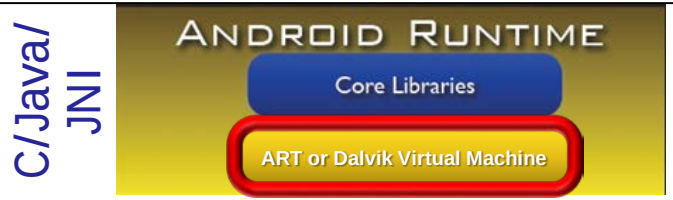


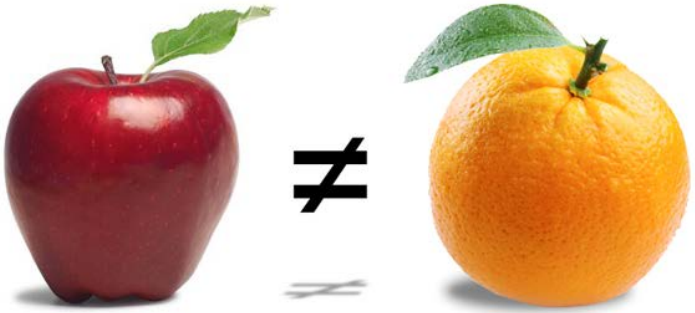


See en.wikipedia.org/wiki/Java_bytecode

Evolution of the Android Execution Environment

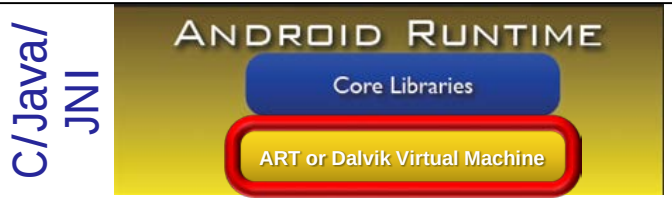
- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
    - Dalvik uses a "register machine" model
      - In contrast, the Java platform uses a "stack machine" model
      - These two types of bytecode are not directly compatible

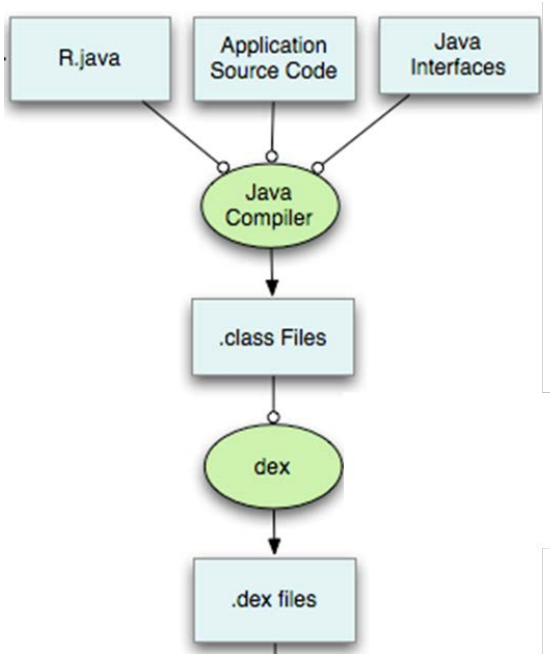




Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
    - Dalvik uses a "register machine" model
    - **dx** program transforms Java bytecode in class files into .dex-formatted bytecodes

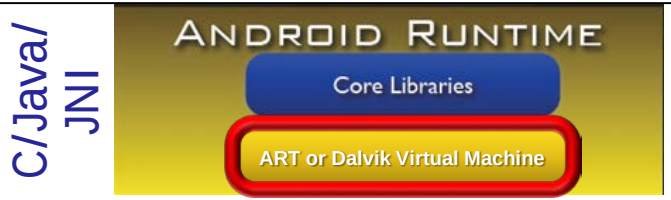


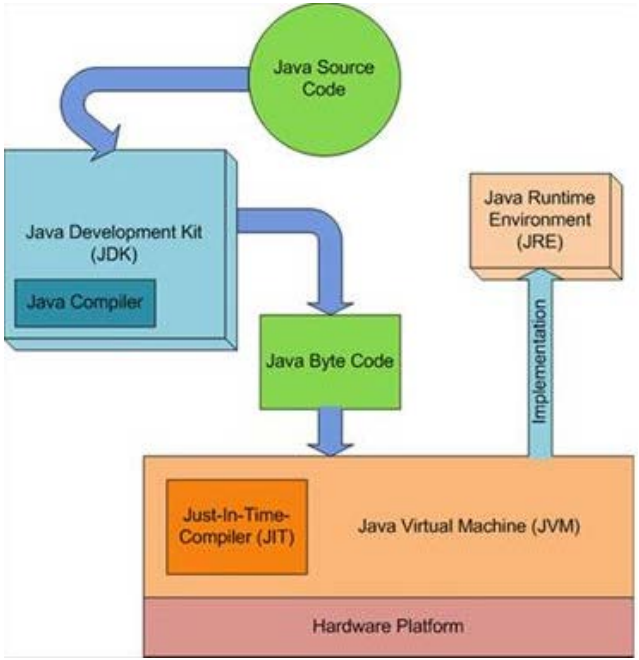


See sites.google.com/site/io/dalvik-vm-internals

Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
    - Dalvik uses a "register machine" model
    - **dx** program transforms Java bytecode in class files into .dex-formatted bytecodes
    - A Just-in-time (JIT) compiler was added to Dalvik later

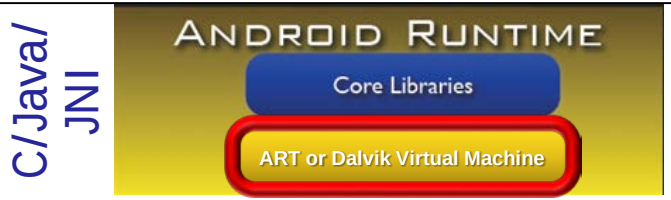


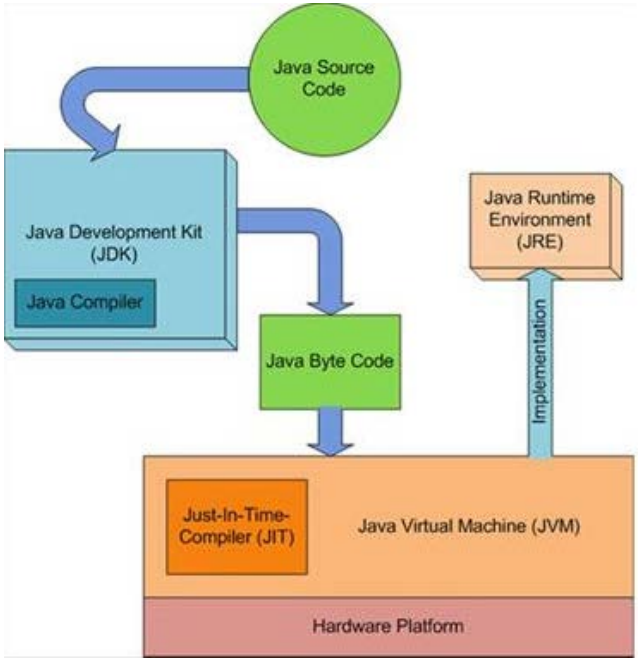


See android-developers.blogspot.com/2010/05/dalvik-jit.html

Evolution of the Android Execution Environment

- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
    - Dalvik uses a "register machine" model
    - **dx** program transforms Java bytecode in class files into .dex-formatted bytecodes
  - A Just-in-time (JIT) compiler was added to Dalvik later
    - A JIT optimizes bytecode dynamically while a program runs

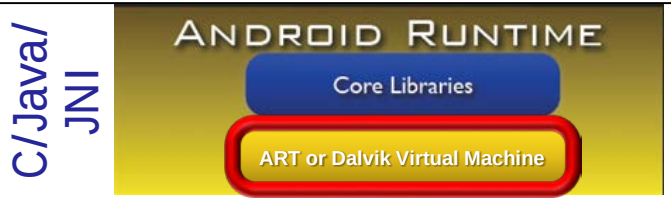




See en.wikipedia.org/wiki/Just-in-time_compilation

Evolution of the Android Execution Environment

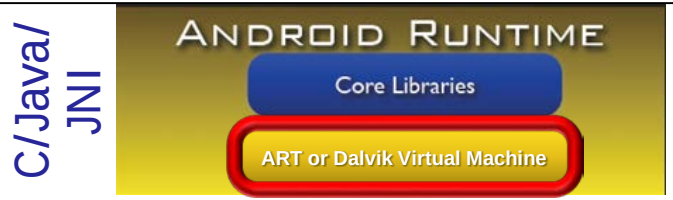
- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced w/an improved "Android Runtime" (ART)



See source.android.com/devices/tech/dalvik/art.html

Evolution of the Android Execution Environment

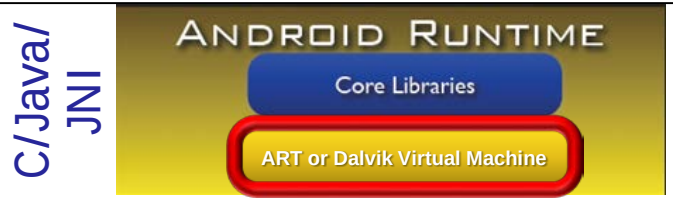
- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced w/an improved "Android Runtime" (ART), e.g.,
    - An "ahead-of-time" (AOT) compiler
      - **dex2oat** compiles .dex files into native code when app is installed



See source.android.com/devices/tech/dalvik/#AOT_compilation

Evolution of the Android Execution Environment

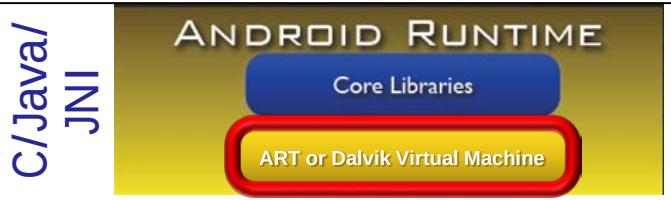
- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced w/an improved "Android Runtime" (ART), e.g.,
    - An "ahead-of-time" (AOT) compiler
    - Better garbage collector (GC)
      - e.g., fewer GC pauses & parallel execution



See source.android.com/devices/tech/dalvik/gc-debug.html#art_gc_overview

Evolution of the Android Execution Environment

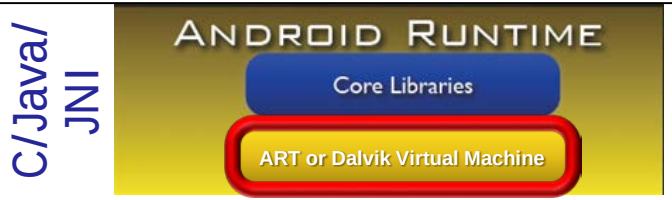
- Android apps are typically written in Java, but don't run in a standard Java VM (JVM)
  - Originally, the Dalvik VM (DVM) was used to interpret so-called Dalvik bytecode
  - Dalvik has been replaced w/an improved "Android Runtime" (ART), e.g.,
    - An "ahead-of-time" (AOT) compiler
    - Better garbage collector
    - A JIT that further optimize ART's AOT compiled code at runtime

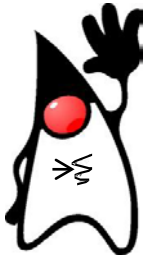


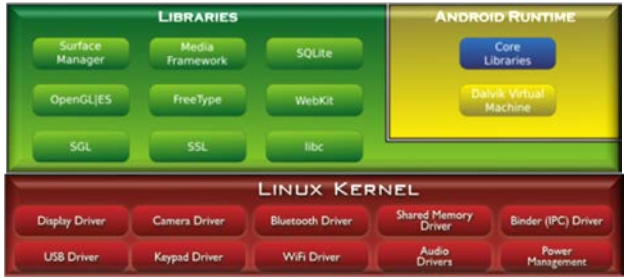
See source.android.com/devices/tech/dalvik/jit-compiler.html

Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android's execution environments implement core Java concurrency features

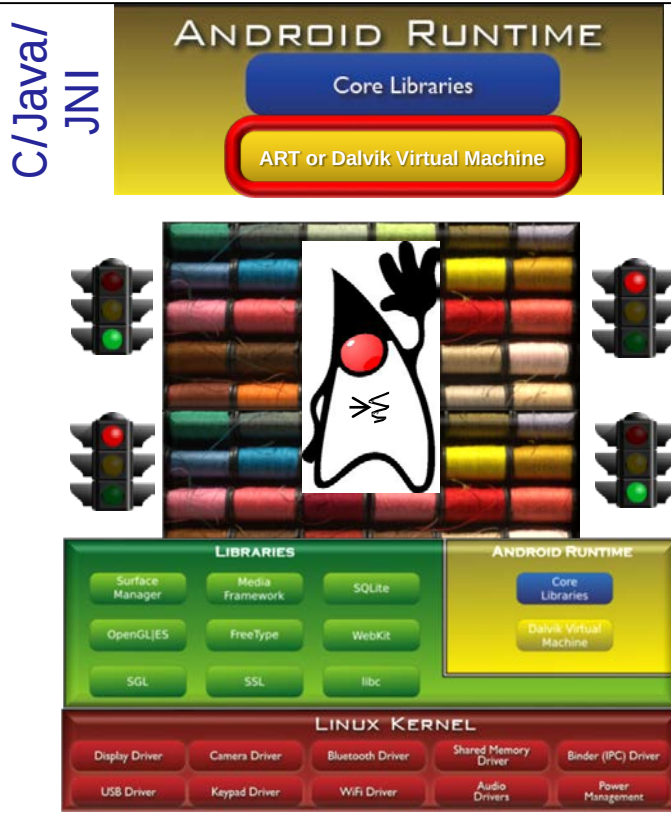






Evolution of the Android Execution Environment

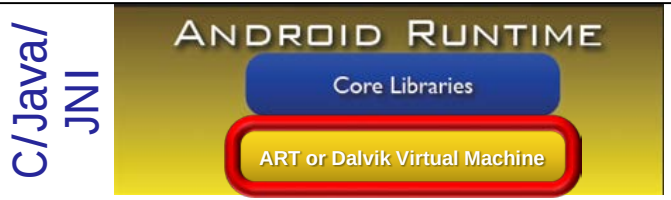
- Irrespective of whether ART or Dalvik is used, Android's execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language

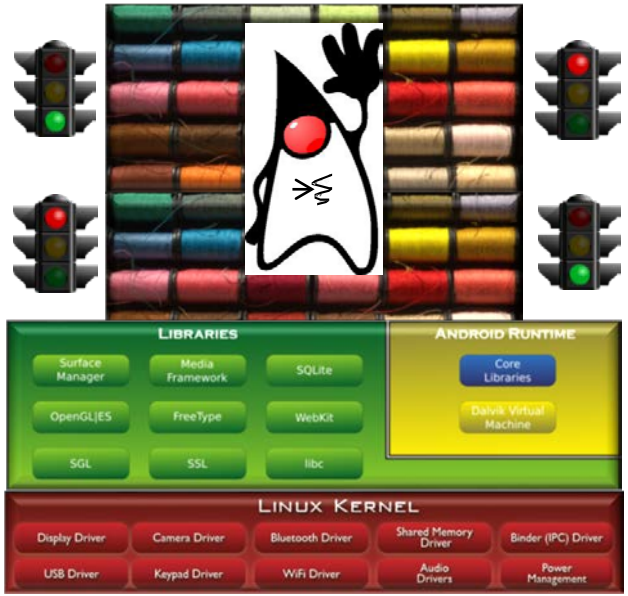


See docs.oracle.com/javase/tutorial/essential/concurrency

Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android's execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language, e.g.
    - Threads that run computations concurrently

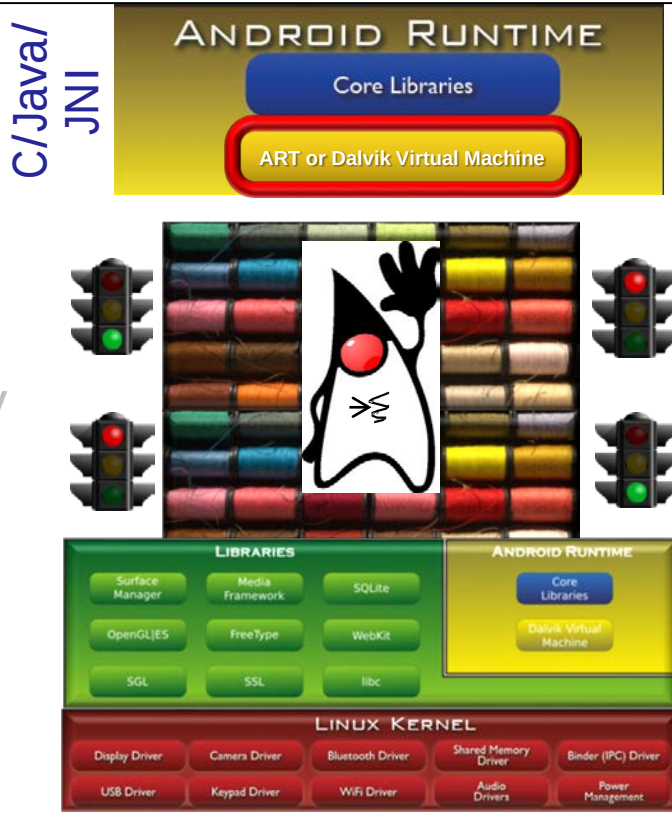




See docs.oracle.com/javase/tutorial/essential/concurrency/threads.html

Evolution of the Android Execution Environment

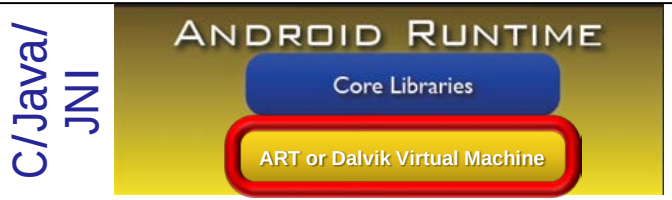
- Irrespective of whether ART or Dalvik is used, Android's execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language, e.g.
    - Threads that run computations concurrently
    - Build-in monitor objects w/synchronization & notification features

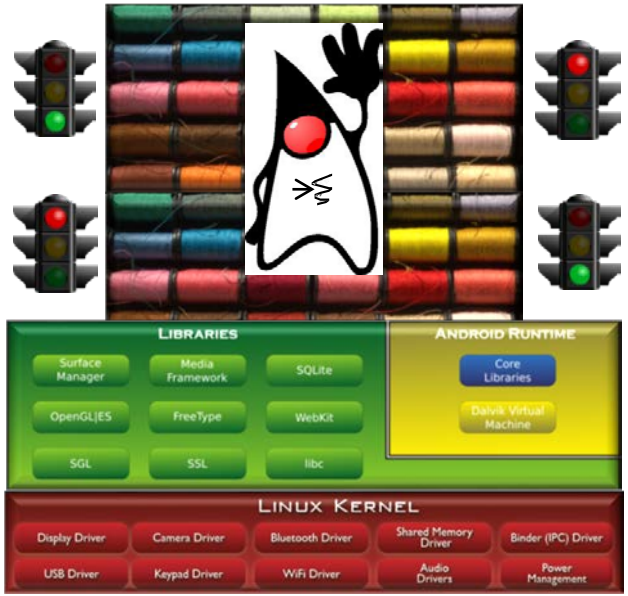


See www.artima.com/insidejvm/ed2/threadsynch.html

Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android's execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language



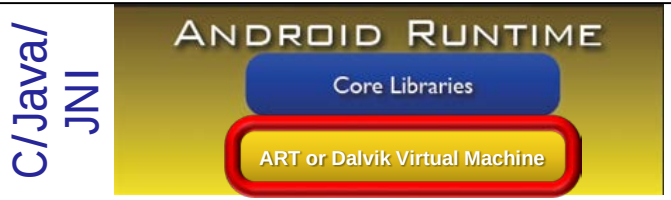


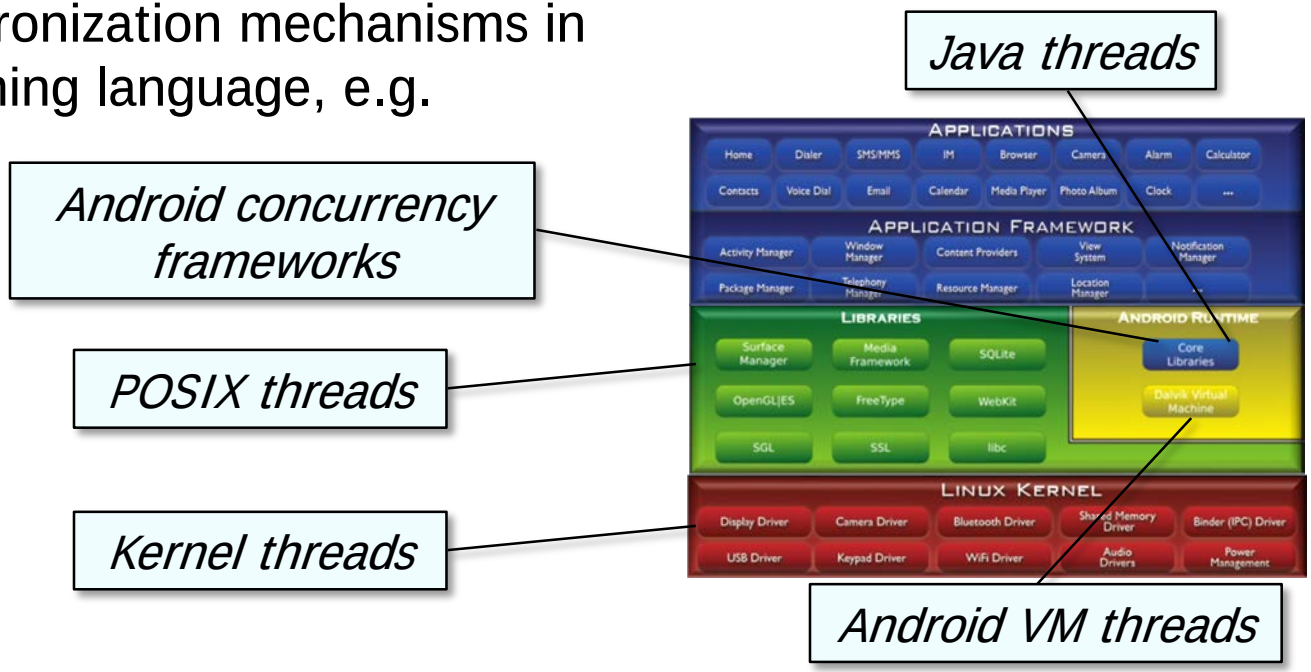
My website contains many more digital learning resources related to Android & Java concurrency

See www.dre.vanderbilt.edu/~schmidt/DigitalLearning

Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android's execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language, e.g.

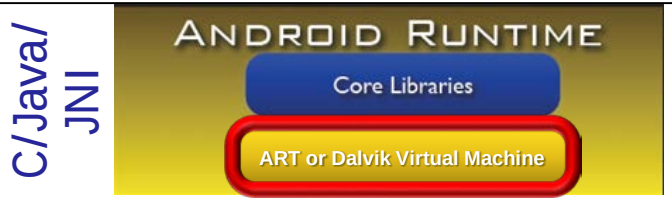


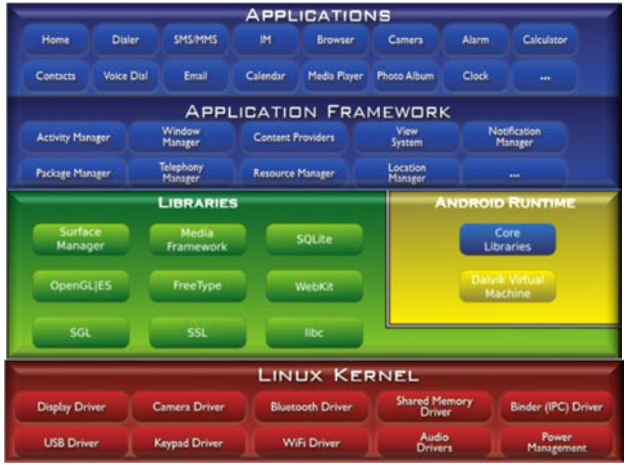


Android's concurrency features involve capabilities at multiple layers

Evolution of the Android Execution Environment

- Irrespective of whether ART or Dalvik is used, Android's execution environments implement core Java concurrency features, e.g.
  - Threading & synchronization mechanisms in the Java programming language
  - There's also support for multi-core hardware that's now widely available for mobile devices





See www.androidauthority.com/fact-or-fiction-android-apps-only-use-one-cpu-core-610352

End of Infrastructure Middleware (Part 2): the Android Runtime Execution Environment