Android Internals Overview

Marko Gargenta
marakana.com

About Marko Gargenta

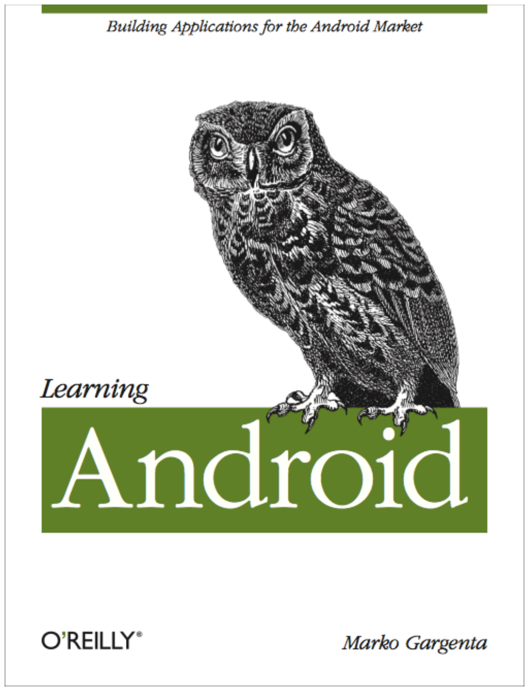
Developer of ***Android Bootcamp*** for Marakana.

Instructor for 1,000s of developers on Android at Qualcomm, Cisco, Motorola, Texas Instruments, Sony-Ericsson, Sharp, NetGear, DoD and other great orgs.

Author of ***Learning Android*** published by O'Reilly.

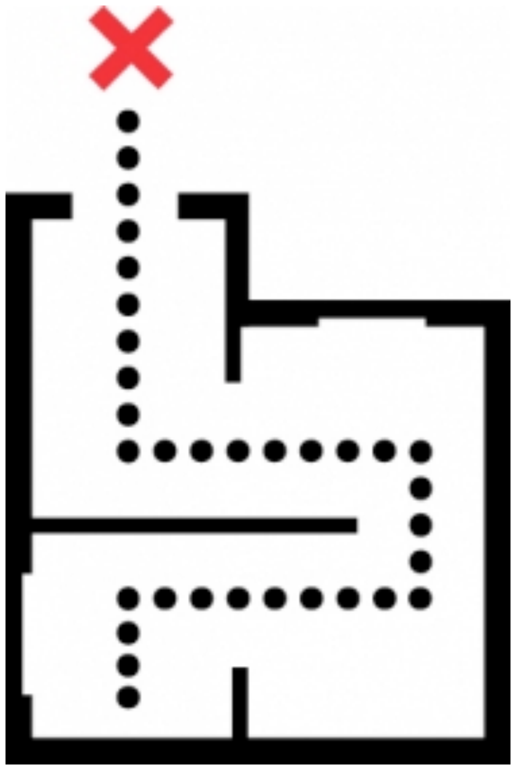
Speaker at OSCON, ACM, IEEE, SDC, AnDevCon.

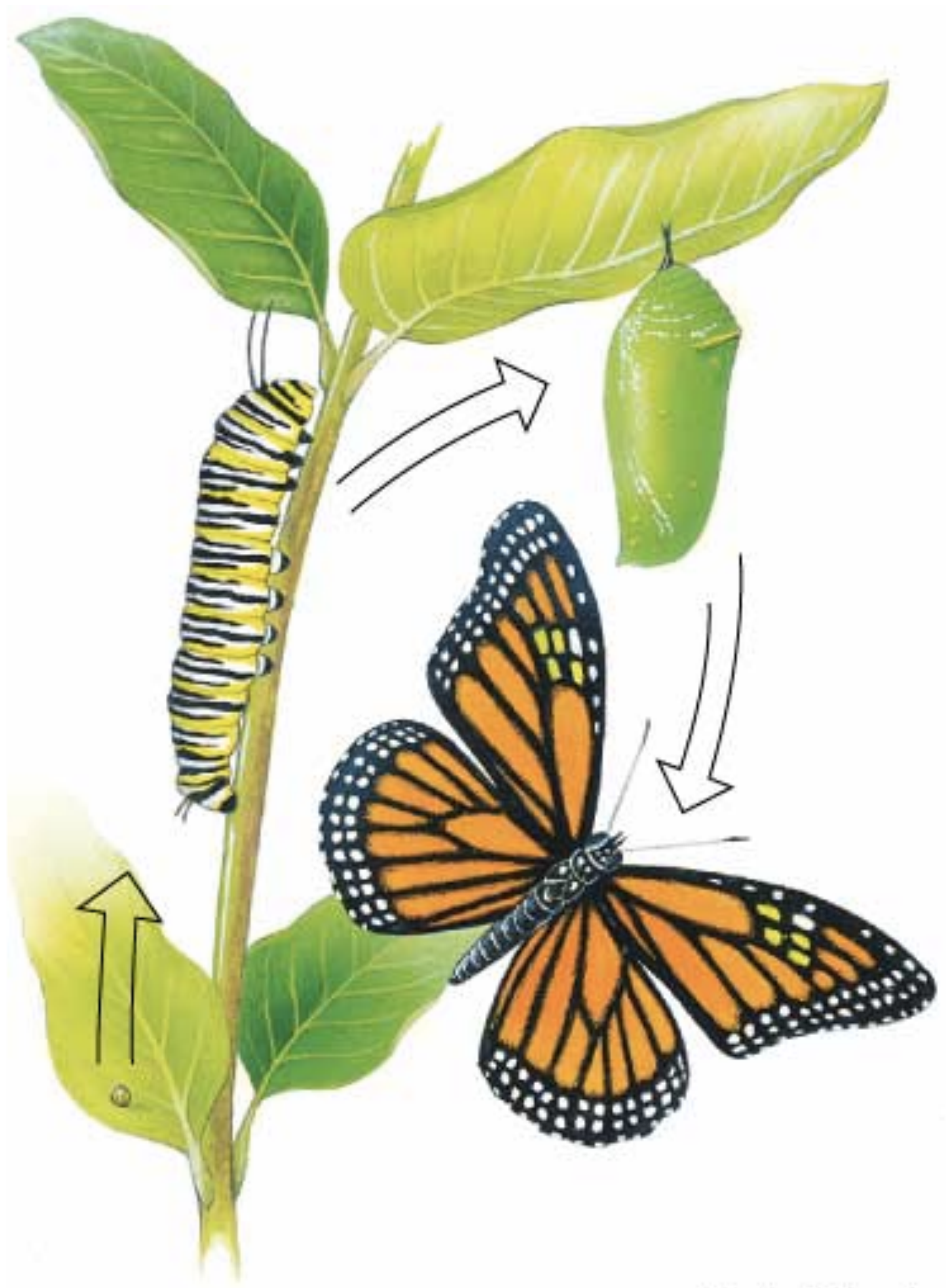
Founder of SFAndroid.org



Agenda

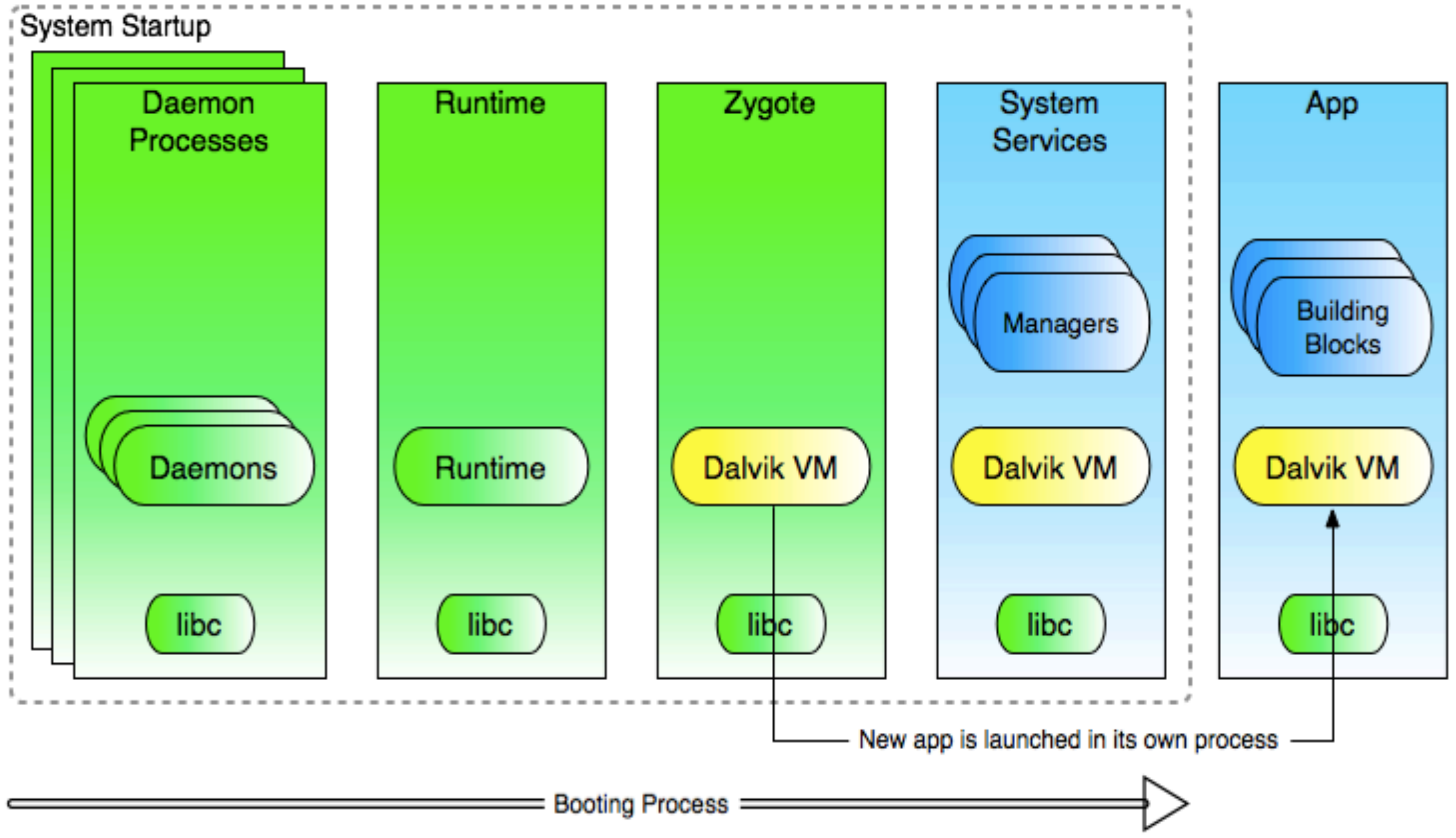
- Android Startup & Runtime
- Layer Interaction
- NDK
- AIDL
- Summary



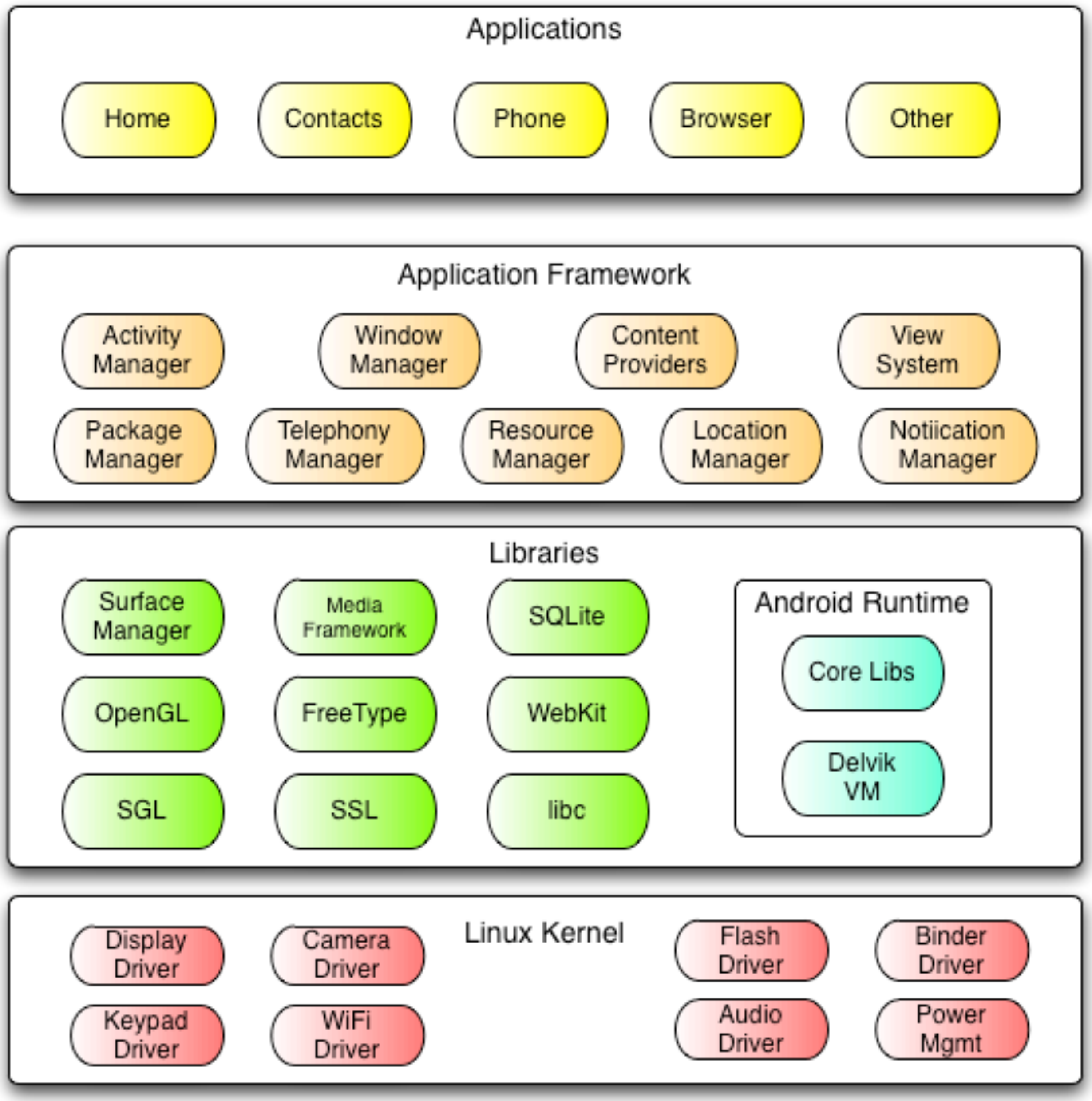


ANDROID STARTUP & RUNTIME

Runtime Overview



The Stack


Applications
Home
Contacts
Phone
Browser
Other
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notiication Manager
Libraries
Surface Manager
Media Framework
SQLite
OpenGL
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libs
Delvik VM
Linux Kernel
Display Driver
Camera Driver
Flash Driver
Binder Driver
Keypad Driver
WiFi Driver
Audio Driver
Power Mgmt

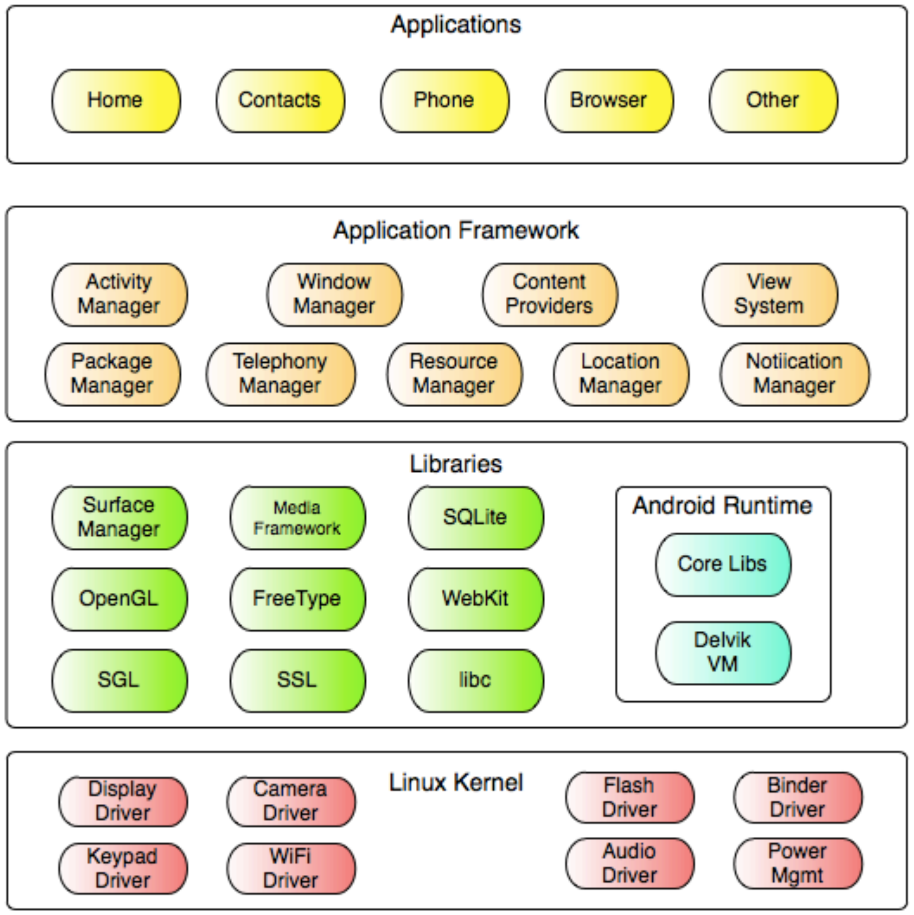
LAYER INTERACTION

Layer Interactions

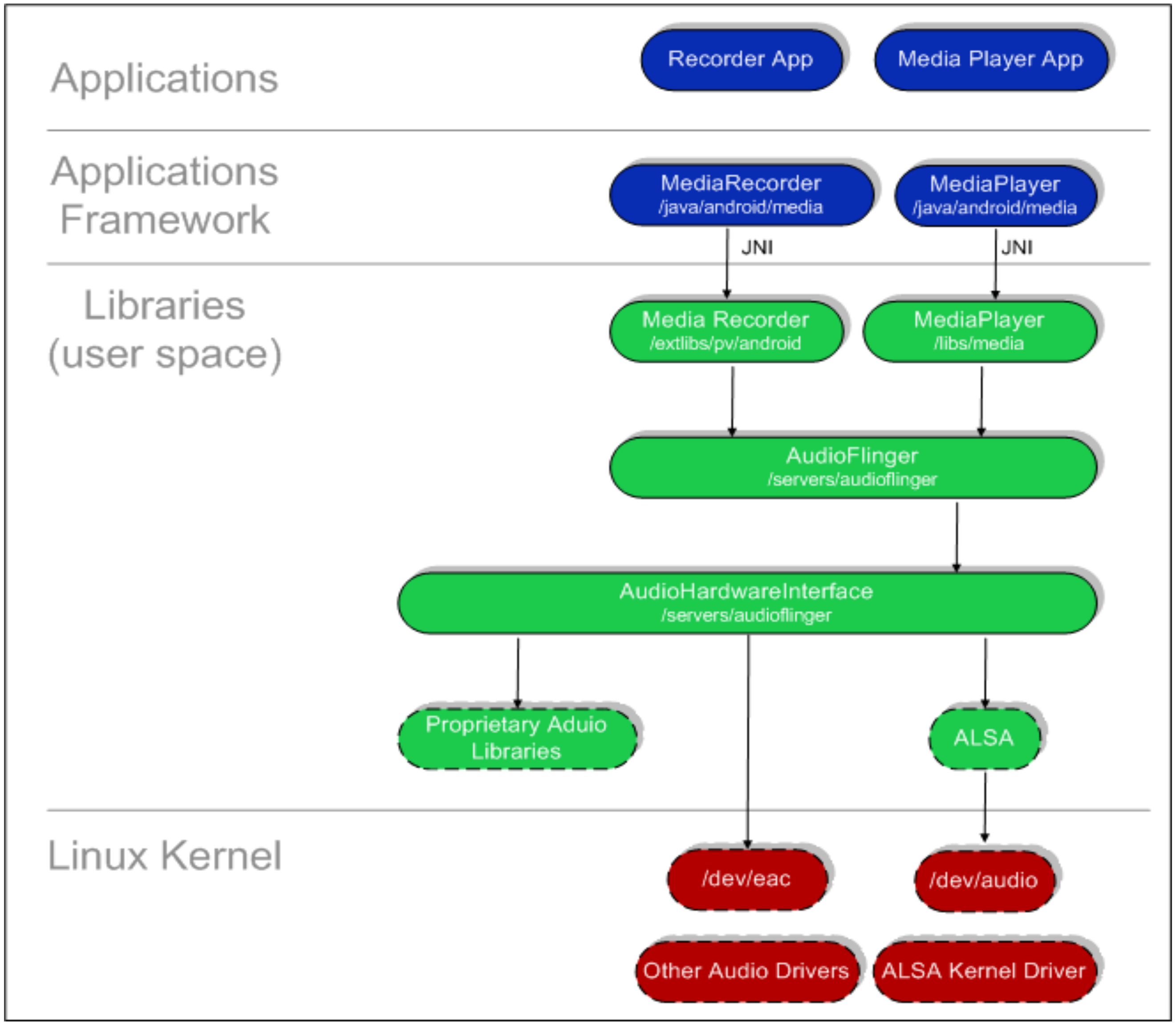
There are three main scenarios for your app to talk to native library:

- Directly
- Via native service
- Via native daemon

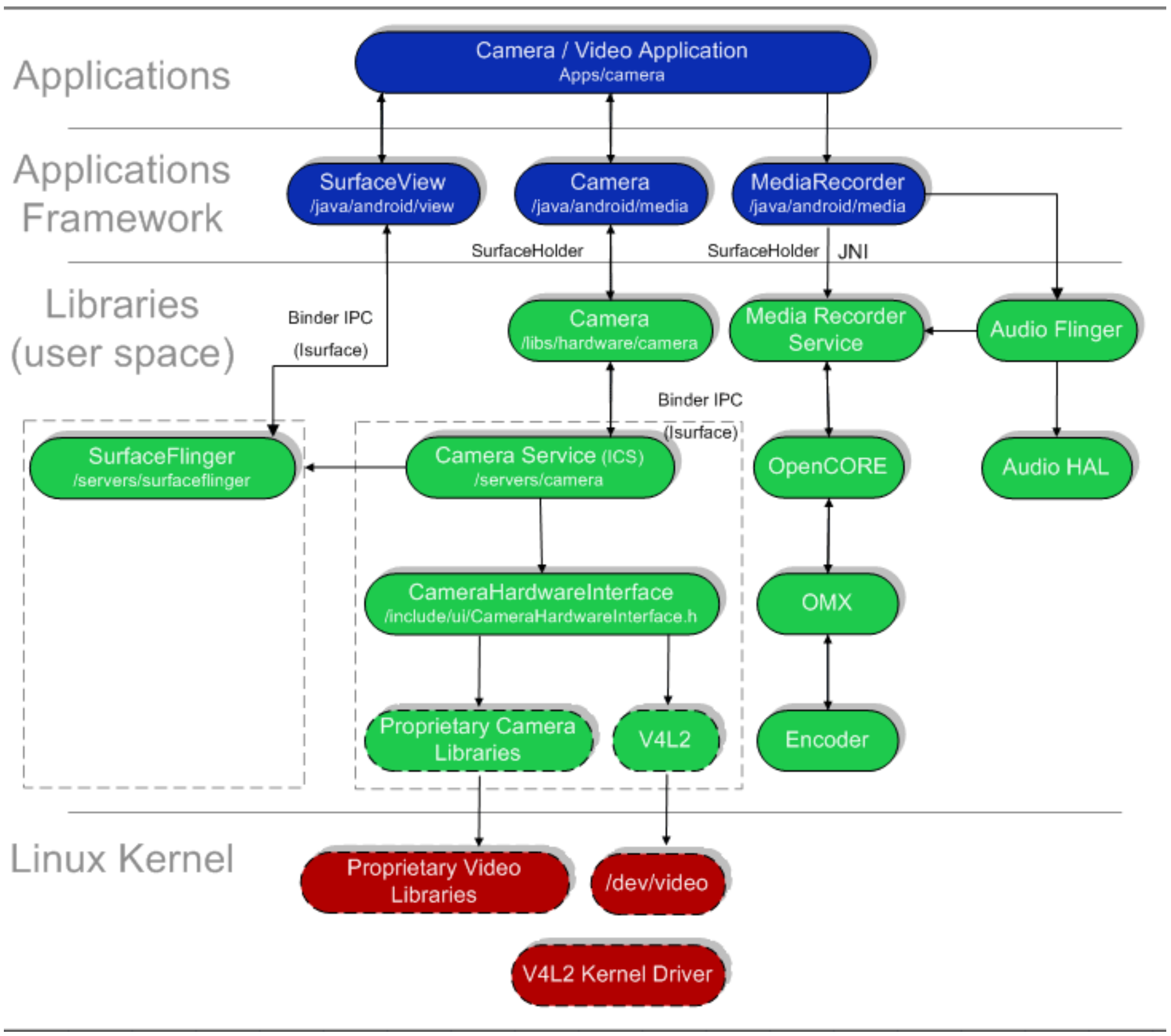
It will depend on the type of app and type of native library which method works best.



Audio Framework


Applications
Recorder App
Media Player App
Applications Framework
MediaRecorder /java/android/media
MediaPlayer /java/android/media
JNI
JNI
Libraries (user space)
Media Recorder /extlibs/pv/android
MediaPlayer /libs/media
AudioFlinger /servers/audioflinger
AudioHardwareInterface /servers/audioflinger
Proprietary Aduio Libraries
ALSA
Linux Kernel
/dev/eac
/dev/audio
Other Audio Drivers
ALSA Kernel Driver

Camera and Video Framework

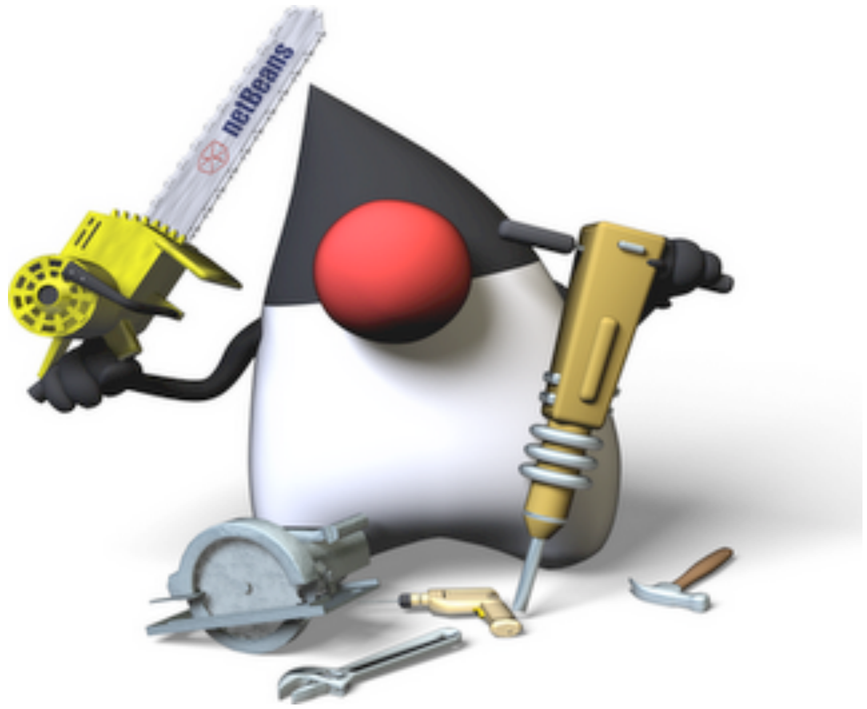


Java Native Interface

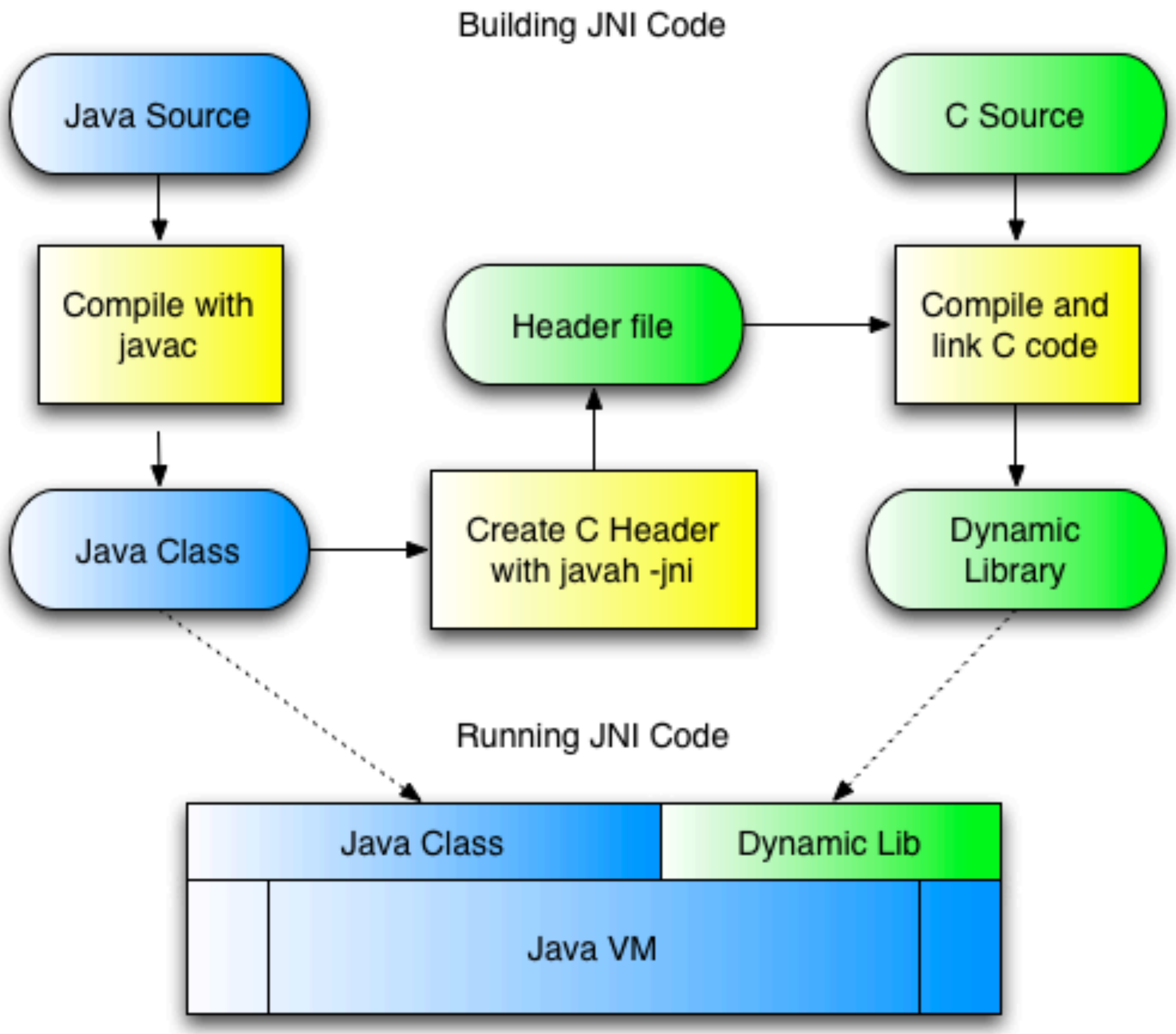
JNI defines naming and coding convention so that Java VM can find and call native code.

JNI is built into JVM to provide access to OS I/O and others.

Your JNI code runs on top Java VM.



Building and Running JNI Code

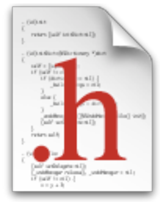


NATIVE DEVELOPMENT KIT

What's in NDK?

Tools to build and cross-compile your native code for the device architecture (ARMv5TE and ARMv7-A; x86 in future releases)

A way to package your library into the APK file so you can distribute your application easily



A set of native system headers that will be supported for the future releases of Android platform (libc, libm, libz, liblog, libjnigrahics, OpenGL/OpenSL ES, JNI headers, minimal C++ support headers, and Android native app APIs)

(some) documentation, sample code and examples

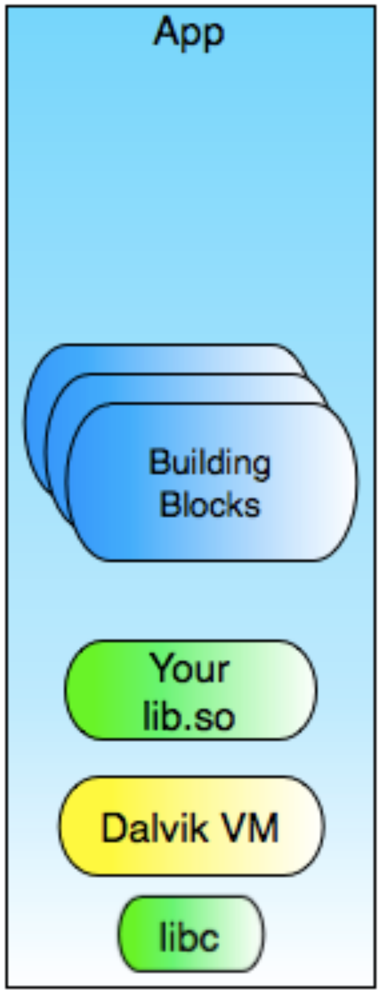
Why NDK?

NDK allows you to develop parts of your Android application in C/C++.

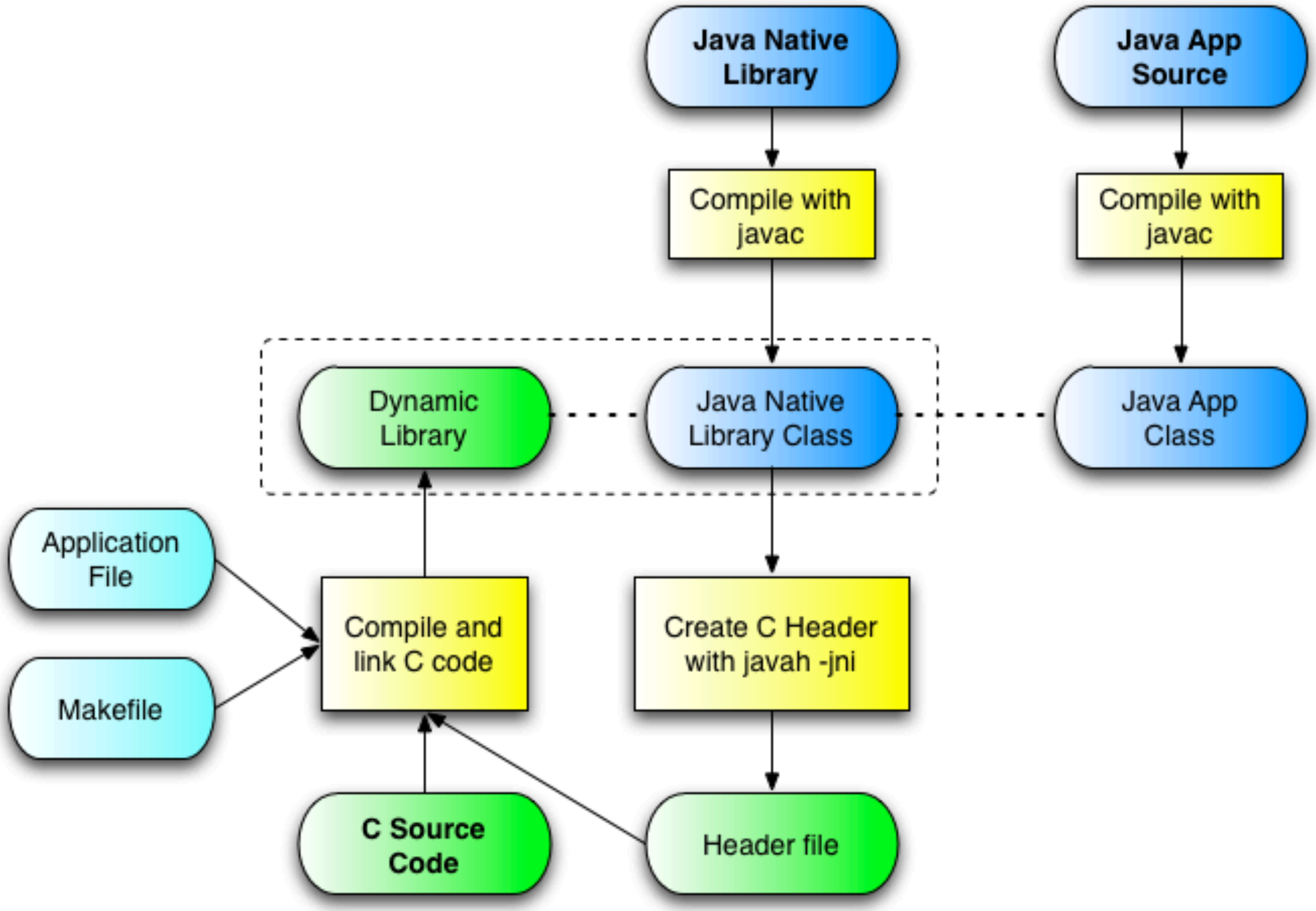
You cannot develop native-only apps in NDK – though 2.3 now supports `NativeActivity`, which allows handling lifecycle callbacks in native code, but access to Services and Content Providers still requires JNI.

NDK code still subject to security sandboxing.

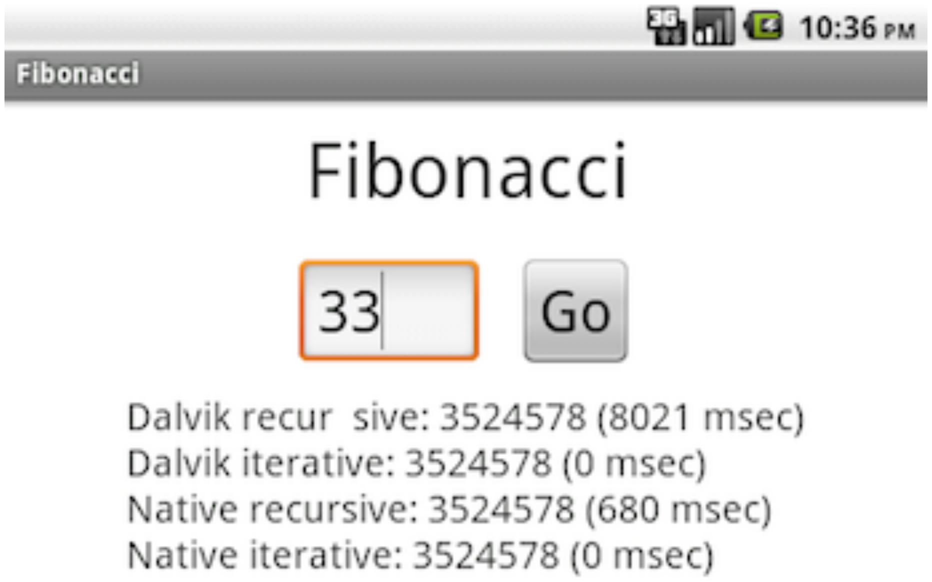
Main motivation for native code is performance (CPU-intensive, self-contained, low-memory footprint code) and the re-use of legacy code.



Using NDK


Java Native Library
Java App Source
Compile with javac
Compile with javac
Dynamic Library
Java Native Library Class
Java App Class
Application File
Makefile
Compile and link C code
Create C Header with javah -jni
C Source Code
Header file

Fibonacci Example



```
Dalvik recur  sive: 3524578 (8021 msec)
Dalvik iterative: 3524578 (0 msec)
Native recursive: 3524578 (680 msec)
Native iterative: 3524578 (0 msec)
```

Fibonacci algorithm is easy to implement both in Java and C, to compare speed differences.

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$

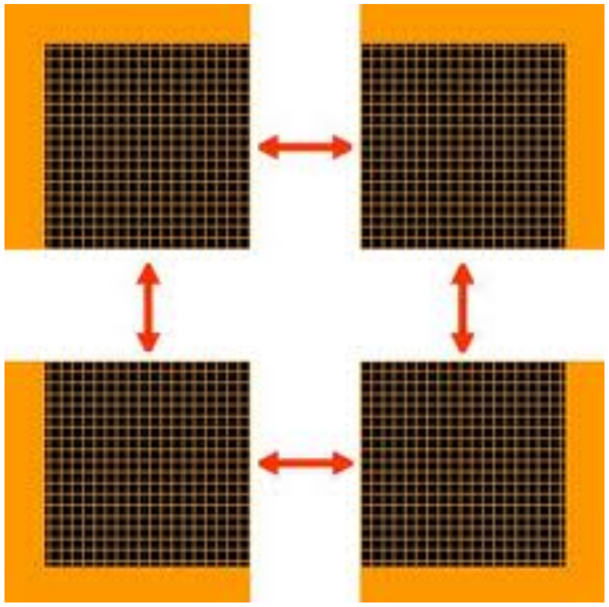
$$F_n = F_{n-1} + F_{n-2}, \qquad F_0 = 0 \quad \text{and} \quad F_1 = 1.$$

Implementing Java Library

1. Use Java keyword `native` to declare future C methods
2. Use `System.loadLibrary()` to load the native module
3. Process your Java library with `javah –jni` to get the header file s

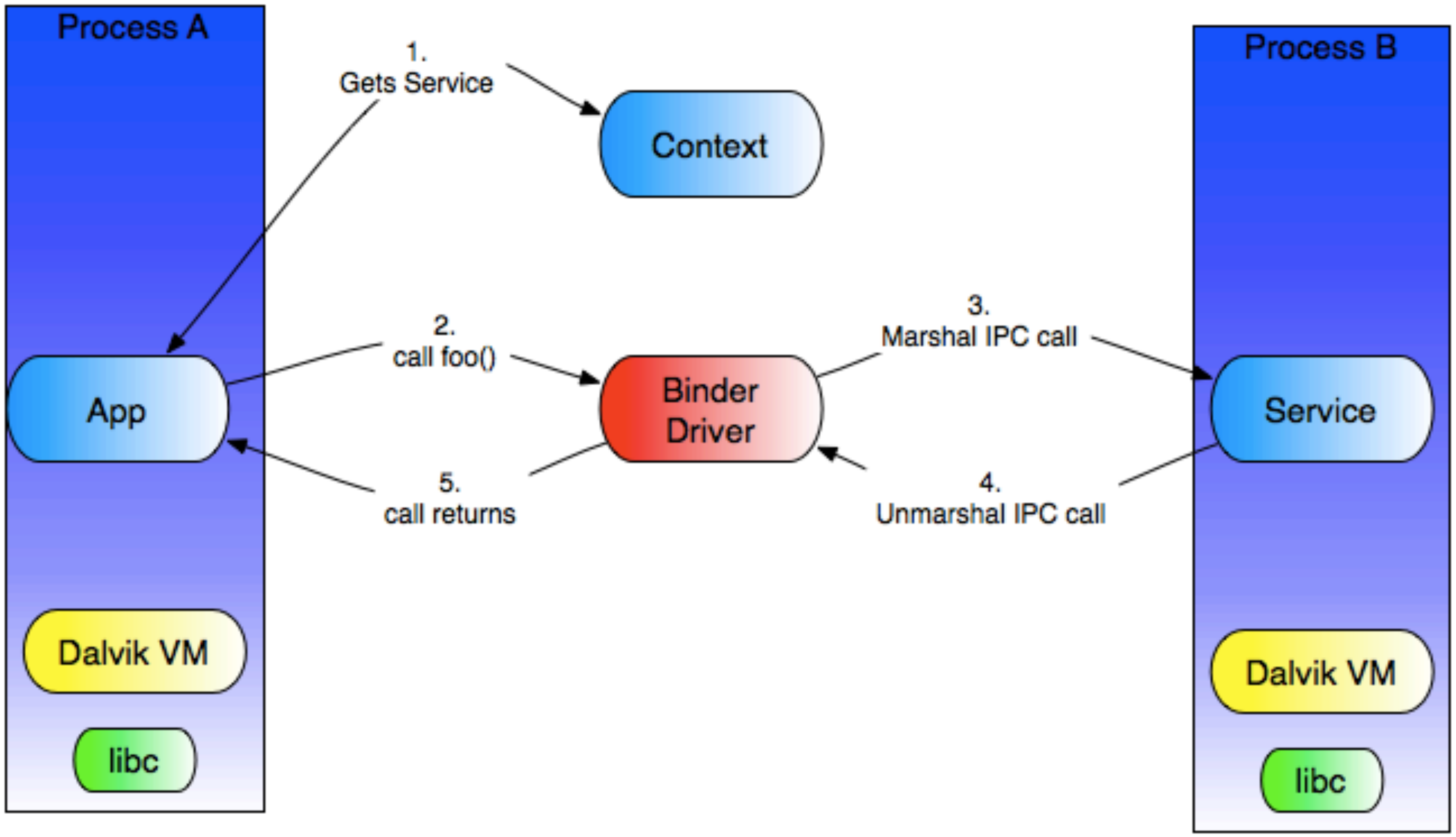
Implementing C Code

1. Place all your C code in `/jni/` folder
2. Copy header file into this folder
3. Create C file and include the header file
4. Create `Android.mk` file
5. Run `ndk-build` in root of your project to create library module in `/lib/`



ANDROID INTERFACE DEFINITION LANGUAGE

Binder IPC



High-performance IPC: shared memory, per-process thread pool, synchronous

Implementing Remote Server

1. Define the interface as an AIDL file
2. Implement the remote service
3. Declare it in Android Manifest

Implementing Remote Client

1. Copy the AIDL file
2. Implement `ServiceConnection` object
   - Use `Stub().asInterface()` to cast `IBinder` to service
3. Bind to the service
4. Call the remote service
   - Handle `RemoteException`

Summary

Android is open and complete system for mobile development. It is based on Java and augmented with XML, with lower levels written in C/C++.

It takes about 3-5 days of intensive training to learn Android application development for someone who has basic Java (or similar) experience.

Marko Gargenta, Marakana.com
marko@marakana.com
+1-415-647-7000