

Embedded Android





These slides are made available to you under a Creative Commons Share-Alike 3.0 license. The full terms of this license are here: <https://creativecommons.org/licenses/by-sa/3.0/>

Attribution requirements and misc., PLEASE READ:

- This slide must remain as-is in this specific location (slide #2), everything else you are free to change; including the logo :-)
- Use of figures in other documents must feature the below “Originals at” URL immediately under that figure and the below copyright notice where appropriate.
- You are free to fill in the “Delivered and/or customized by” space on the right as you see fit.
- You are FORBIDDEN from using the default “About” slide as-is or any of its contents.

(C) Copyright 2010-2019, Opersys inc.

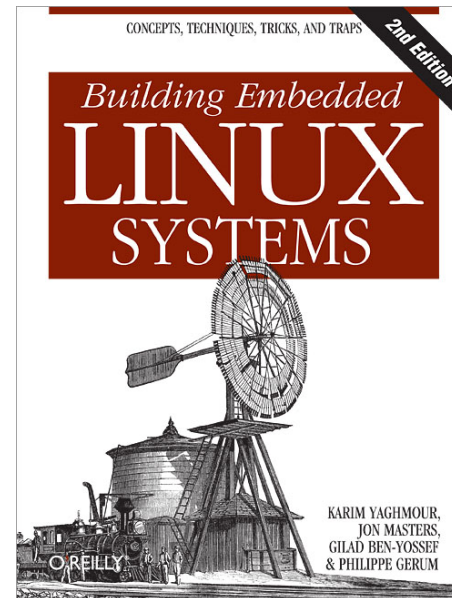
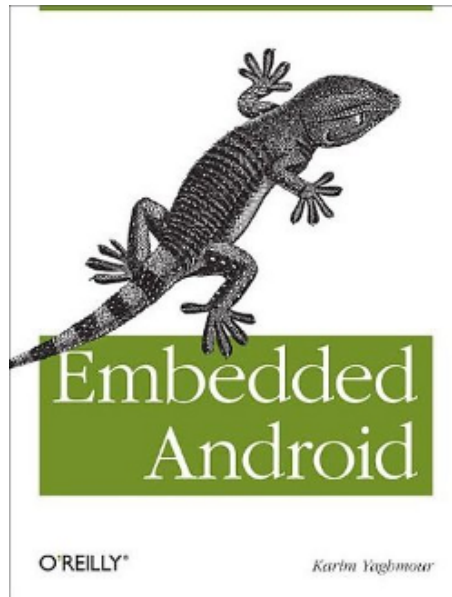
These slides created by: Karim Yaghmour

Originals at: www.opersys.com/training/embedded-android

Delivered and/or customized by

About

- Author of:



- Introduced Linux Trace Toolkit in 1999
- Originated Adeos and relayfs (kernel/relay.c)
- Training, Custom Dev, Consulting, ...

About Android

- Huge
- Fast moving
- Stealthy
- Increasingly complex

About Android

- Huge
- Fast moving
- Stealthy
- Increasingly very complex

Mainly:

- Internals-specifics are subject to change

Therefore:

- Must learn to relearn every new release

Goals

- Master the intricacies of all components making up Android, including kernel Androidisms
- Get hands-on experience in building and customizing Android-based embedded systems
- Learn basics of Android app development
- Familiarize with the Android ecosystem

Format

- Tracks:
 - Lecture
 - Exercises
- Fast pace
- Lots of material

Requirements

- Embedded systems development
- C, C++
- Basic Unix/Linux command line interface experience
- Java (working knowledge of) and/or fast learner ;)

Knowledge Fields

- Main fields:
 - Embedded systems
 - Linux kernel internals
 - Device driver development
 - Unix system administration
 - GNU software development
 - Java-based development
 - Android app development
- Vastly different fields, few (if any) master all
- Fluency required to tackle tough problems

Topics

- Setting context:
 - Introduction to embedded Android
 - Concepts and internals
 - Android Open Source Project (AOSP)
 - Kernel basics
- Android Essentials:
 - Embedded Linux root FS
 - Native Android user-space
 - System Server
 - Hardware Abstraction Layer
 - Android Framework
 - Quick Java Introduction

- Advanced/Specialized:
 - Kernel internals
 - Device driver development overview
 - Graphics stack
 - Security
 - Memory management
 - Treble
 - Android Things
 - Debugging and Performance Analysis

Courseware

- These slides
- Exercises
- Android documentation:
 - source.android.com
 - developer.android.com
 - tools.android.com
- Android Open Source Project:

'Use the Source, Luke, use the Source. Be one with the code.' -- Linus Torvalds

Hands-On Environment

- Host:
 - Ubuntu 64-bit 16.04
 - For 8.1/Oreo: 150GB / AOSP (source ~ 35GB, build ~ 85GB, git ~30GB)
- Hikey LeMaker 96boards:
 - Kirin 620 SoC
 - ARM® Cortex™-A53 Octa-core 64-bit up to 1.2GHz (ARM v8 instruction set)
 - 2GB LPDDR3 SDRAM @ 800MHz
 - 8GB eMMC on board storage
 - 96boards form-factor w/ expansion connectors
- Emulator
- Other targets previously used:
 - Nexux 7 2013 (“flo”) -- Qualcomm
 - Minnowboard Max -- x86_64
 - Inforce IFC6410 -- Qualcomm
 - Panda Board – TI Omap
 - BeagleBone – TI Sitara

Introduction to Embedded Android

- Basics
- History
- Ecosystem
- Legal framework
- Platform and hardware requirements
- Development tools

1. Basics

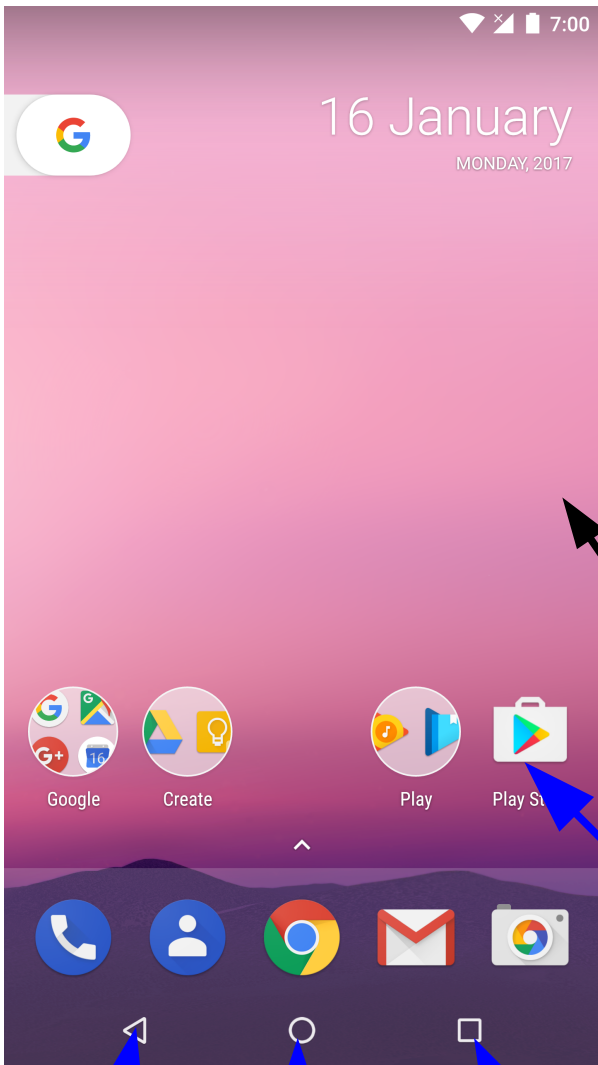
- Features
- UX Concepts
- App Concepts

1.1. Features (old snapshot)

- Application framework enabling reuse and replacement of components
- Dalvik virtual machine optimized for mobile devices
- Integrated browser based on the open source WebKit engine
- Optimized graphics powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- SQLite for structured data storage
- Media support for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)
- GSM Telephony (hardware dependent)
- Bluetooth, EDGE, 3G, and WiFi (hardware dependent)
- Camera, GPS, compass, and accelerometer (hardware dependent)
- Rich development environment including a device emulator, tools for debugging, memory and performance profiling, and the Android Studio IDE

1.2. UX Concepts

- Browser-like
- Swipe – Pinch – Zoom
- No user-concept of “task”
- Main keys:
 - HOME
 - BACK
 - OVERVIEW (recent apps)
- App-model allows users to safely install/test almost anything



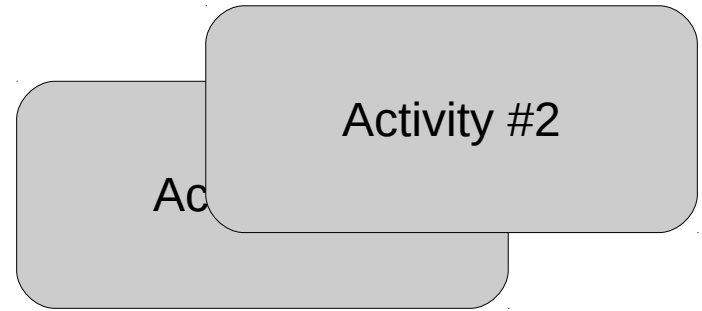
“Click” →



Activity #1

“Back” ↑

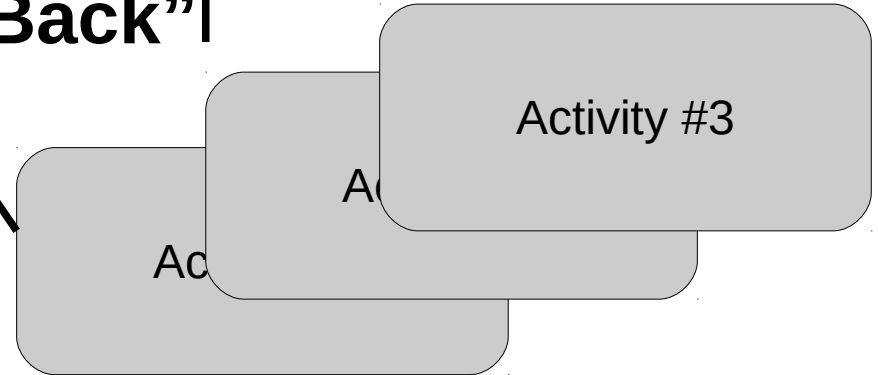
“Click” ↓



Activity #2

“Click” ↓

“Back” ↑



Activity #3

“Home”

Activity

Overview

Back

Home

1.3. App Concepts

- No single entry point (No main() !?!?)
- Unlike Windows or Unix API/semantics in many ways
- Processes and apps will be killed at random: developer must code accordingly
- UI disintermediated from app “brains”
- Apps are isolated, very
- Behavior predicated on low-memory conditions

2. History

- 2002:
 - Sergey Brin and Larry Page started using Sidekick smartphone
 - Sidekick one of 1st smartphones integrating web, IM, mail, etc.
 - Sidekick was made by Danger inc., co-founded by Andy Rubin (CEO)
 - Brin/Page met Rubin at Stanford talk he gave on Sidekick's development
 - Google was default search engine on Sidekick
- 2004:
 - Despite cult following, Sidekick wasn't making \$
 - Danger inc. board decided to replace Rubin
 - Rubin left. Got seed \$. Started Android inc. Started looking for VCs.
 - Goal: Open mobile hand-set platform
- 2005 - July:
 - Got bought by Google for undisclosed sum :)
- 2007 - November:
 - Open Handset Alliance announced along with Android

- 2008 - Sept.: Android 1.0 is released
- 2009 - Feb.: Android 1.1
- 2009 - Apr.: Android 1.5 / Cupcake
- 2009 - Sept.: Android 1.6 / Donut
- 2009 - Oct.: Android 2.0/2.1 / Eclair
- 2010 - May: Android 2.2 / Froyo
- 2010 - Dec.: Android 2.3 / Gingerbread
- 2011 - Jan : Android 3.0 / Honeycomb – Tablet-optimized
- 2011 – May: Android 3.1 – USB host support
- 2011 – Nov: Android 4.0 / Ice-Cream Sandwich – merge Gingerbread and Honeycomb
- 2012 – Jun: Android 4.1 / Jelly Bean – Platform Optimization
- 2012 – Nov: Android 4.2 / Jelly Bean - Multi-user support
- 2013 -- July: Android 4.3 / Jelly Bean -- BLE / GLES 3.0
- 2013 -- Oct: Android 4.4 / KitKat -- Low RAM optimizations
- 2014 -- Nov: Android 5.0 / Lollipop - ART and other pixie dust
- 2015 -- Mar: Android 5.1 / Lollipop - Multi-SIM card
- 2015 -- Oct: Android 6.0 / Marshmallow -- New permission model
- 2016 – Aug: Android 7.0 / Nougat – Multi-window
- 2016 – Oct: Android 7.1 / Nougat – A/B updates
- 2017 – Aug: Android 8.0 / Oreo – Treble / PIP
- 2017 – Dec: Android 8.1 / Oreo – Low-end device optimizations
- 2018 – Aug: Android 9.0 / Pie – AI, multi-camera API, indoor wifi positioning

3. “Flavors”

- Phone
- Tablet
- Wear (watches)
- TV
- Car:
 - “Auto”
 - “Automotive”
- Things (IoT)

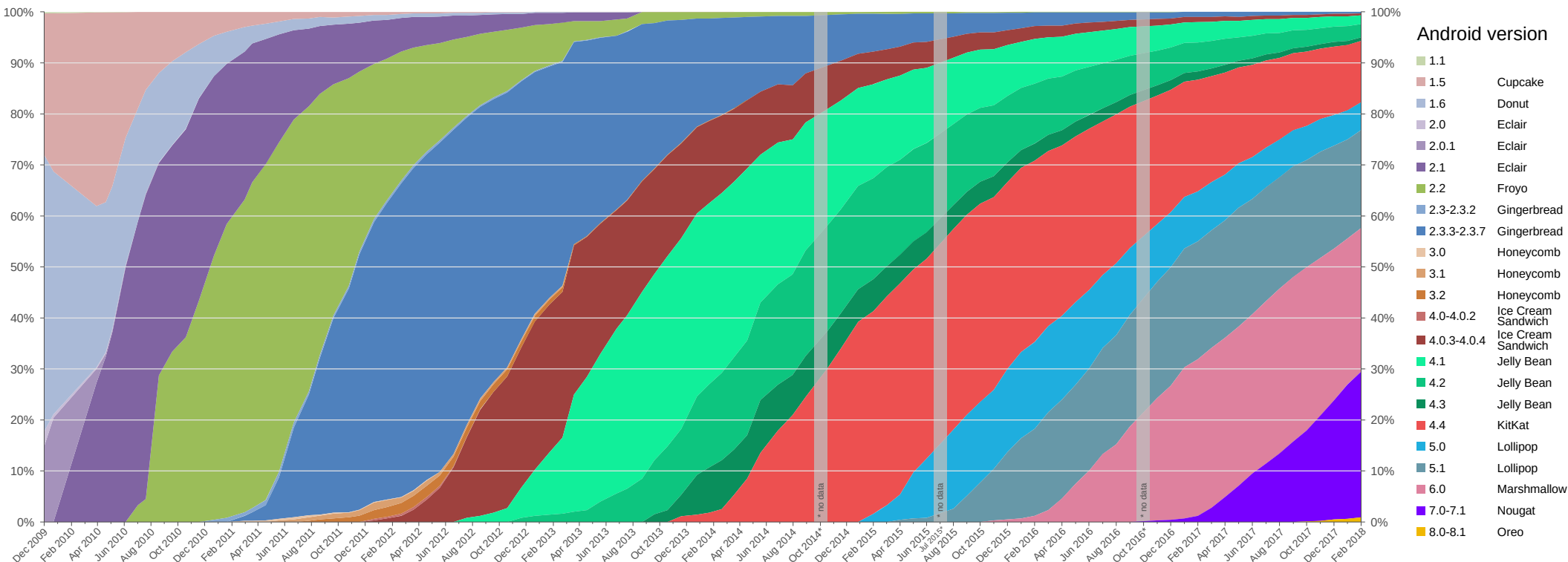
3.1. Specifics

- Sources for new “flavors” not typically available at launch:
 - Closed launches
 - Gradual release once APIs/functionality stable
- Similarities:
 - Generally same codebase (Brillo was a bit different, but it wasn't called “Android”).
 - Same architecture
 - Same internal mechanisms
 - Same build system
 - Same C library
 - Etc.

- Differences:
 - Specific HALs
 - Specific System Services
 - Specific “system apps” -- tailored launcher, etc.
 - Tailored/trimmed build
 - Special attention to certain parts of the stack – benefits other form-factors as well.

4. Ecosystem

- >2B monthly active devices worldwide
- 2.7M apps (vs. 2.2M for Apple's app store)
- 88% global smartphone marketshare (iOS is 12%)



4.1. Who's playing?

EVERYBODY

4.2. Open Handset Alliance

- “... a group of 80 technology and mobile companies who have come together to accelerate innovation in mobile and offer consumers a richer, less expensive, and better mobile experience. Together we have developed Android™, the first complete, open, and free mobile platform.”
- Unclear what OHA does or what benefits, if any, members derive
- Not an organization with board members, staff, etc. ... just an “Alliance”
- Google's Android team are the lead on all bleeding edge dev, all else tag along
- OHA is largely inactive / absent
- Comprised of:
 - Mobile Operators: Sprint, T-Mobile, Vodafone, NTT Docomo, ...
 - Handset Manufacturers: HTC, Motorola, LG, Samsung, Sony Ericsson, ...
 - Semiconductor Companies: ARM, Freescale, Intel, NVIDIA, Qualcomm, TI, ...
 - Software Companies: Google, ...
 - Commercialization Companies: ...

5. Legal Framework

- Code access
- Code licenses
- Branding use
- Google's own Android Apps
- Alternative App stores
- Oracle v. Google

5.1. Code Access

- Parts:
 - Kernel
 - Android Open Source Project (AOSP)
- Kernel:
 - Should have access to latest shipped version => GPL requirement
 - Google-maintained forks at android.googlesource.com
- AOSP:
 - *Usually* Code-drops every year
 - Official AOSP branches at android.googlesource.com
 - Managed by “repo” tool, an overlay to “git”

5.2. Code Licenses

- Kernel:
 - GNU General Public License (a.k.a. GPL)
- AOSP:
 - Mostly Apache License 2.0 (a.k.a. ASL)
 - Having GPL-free user-space was a design goal
 - Even BlueZ (GPL) is now gone -- starting 4.2
 - Some key components in BSD: Bionic and Toybox
 - “external/” directory contains a mixed bag of licenses
- May be desirable to add GPL/LGPL components:
 - BusyBox
 - glibc

5.3. Branding Use

- Android Robot:
 - Very much like the Linux penguin
- Android Logo (A-N-D-R-O-I-D w/ typeface):
 - Cannot be used
- Android Custom Typeface:
 - Cannot be used
- Android in Official Names:
 - As descriptor only: “for Android”
 - Most other uses require approval
- Android in Messaging:
 - Allowed if followed by a generic: “Android Application”
- Compliance through CDD/CTS/VTs involved in “approval”



5.4. Google's own Android Apps

- The non-AOSP apps:
 - Google Mobile Services (GMS)
 - Play Store
 - YouTube
 - Maps
 - Gmail
 - Photos
 - ...
- Require:
 - CDD/CTS/MTS Compliance
 - Signed agreement w/ Google
- Inquiries: android-partnerships@google.com

5.5. Alternative “App Stores”

- Many app stores out there:
 - Amazon App Store
 - GetJar
 - Slide Me
 - Yandex
 - AppBrain
 - Samsung Galaxy Apps
 - F-Droid
 - ...
- Nothing precluding you from having your own

5.6. Oracle v. Google

- Filed August 2010
- Patent infringement:
 - 6,125,447; 6,192,476; 5,966,702; 7,426,720; RE38,104; 6,910,205; and 6,061,520
- Copyright infringement:
- Android does not use any Oracle Java libraries or JVM in the final product.
- Android relies on Apache Harmony and Dalvik instead.
- In October 2010, IBM left Apache Harmony to join work on Oracle's OpenJDK, leaving the project practically orphaned.
- ...
- In Spring of 2012 Oracle lost both on Copyright and Patent fronts
- ...
- Oracle appealed
- ...

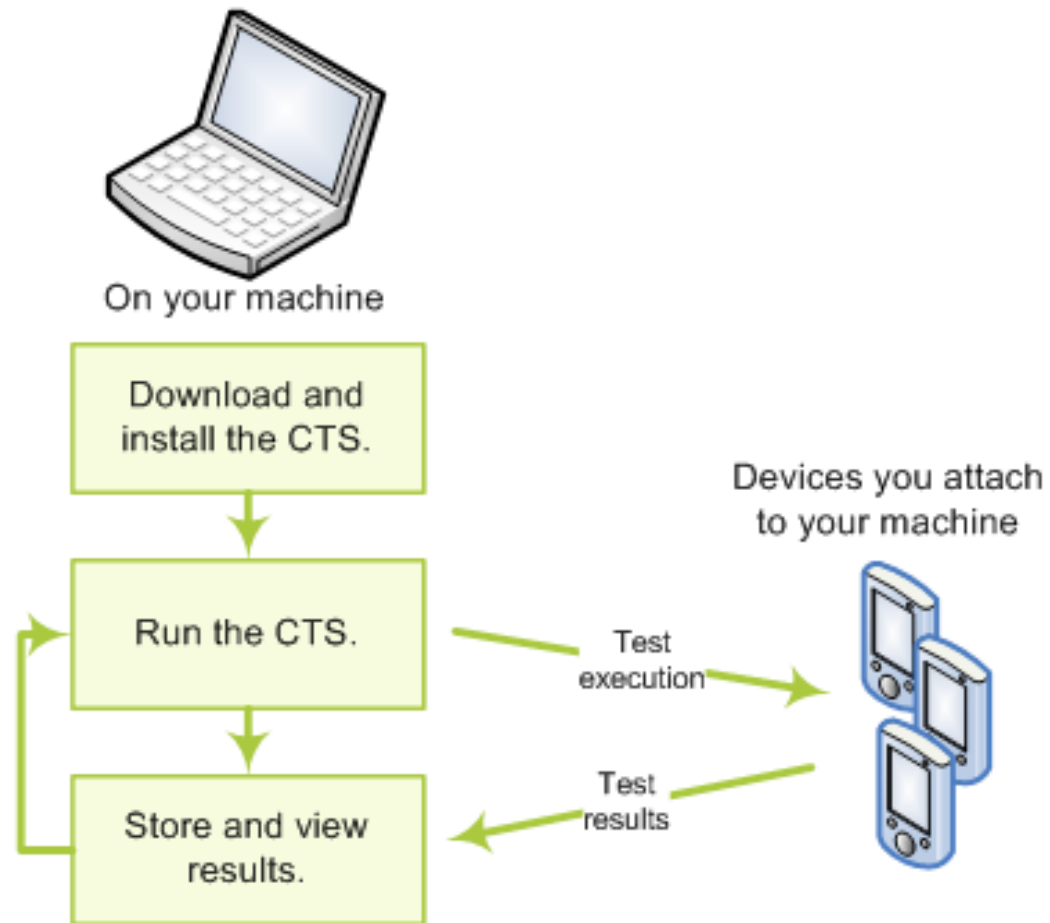
6. Platform and Hardware requirements

- In principle:
 - Android runs on top of Linux
 - Therefore: if it runs Linux, it can run Android
- Known to have been made to work on:
 - ARM
 - x86
 - MIPS
 - SuperH
- Put in all sort of devices:
 - Washers, micro-wave ovens, car systems, etc.

6.1. Compliance Definition Document

- Software: MUST conform to AOSP
- Application Packaging Compatibility: support “.apk” files
- Multimedia Compatibility: decoders, encoders, recording, ...
- Developer Tool Compatibility: adb, ddms, Monkey
- Hardware compatibility:
 - Display and Graphics
 - Input Devices
 - Data Connectivity
 - Cameras
 - Memory and Storage
 - USB
- Performance Compatibility
- Security Model Compatibility
- Software Compatibility Testing
- Updatable Software: MUST include mechanism to update

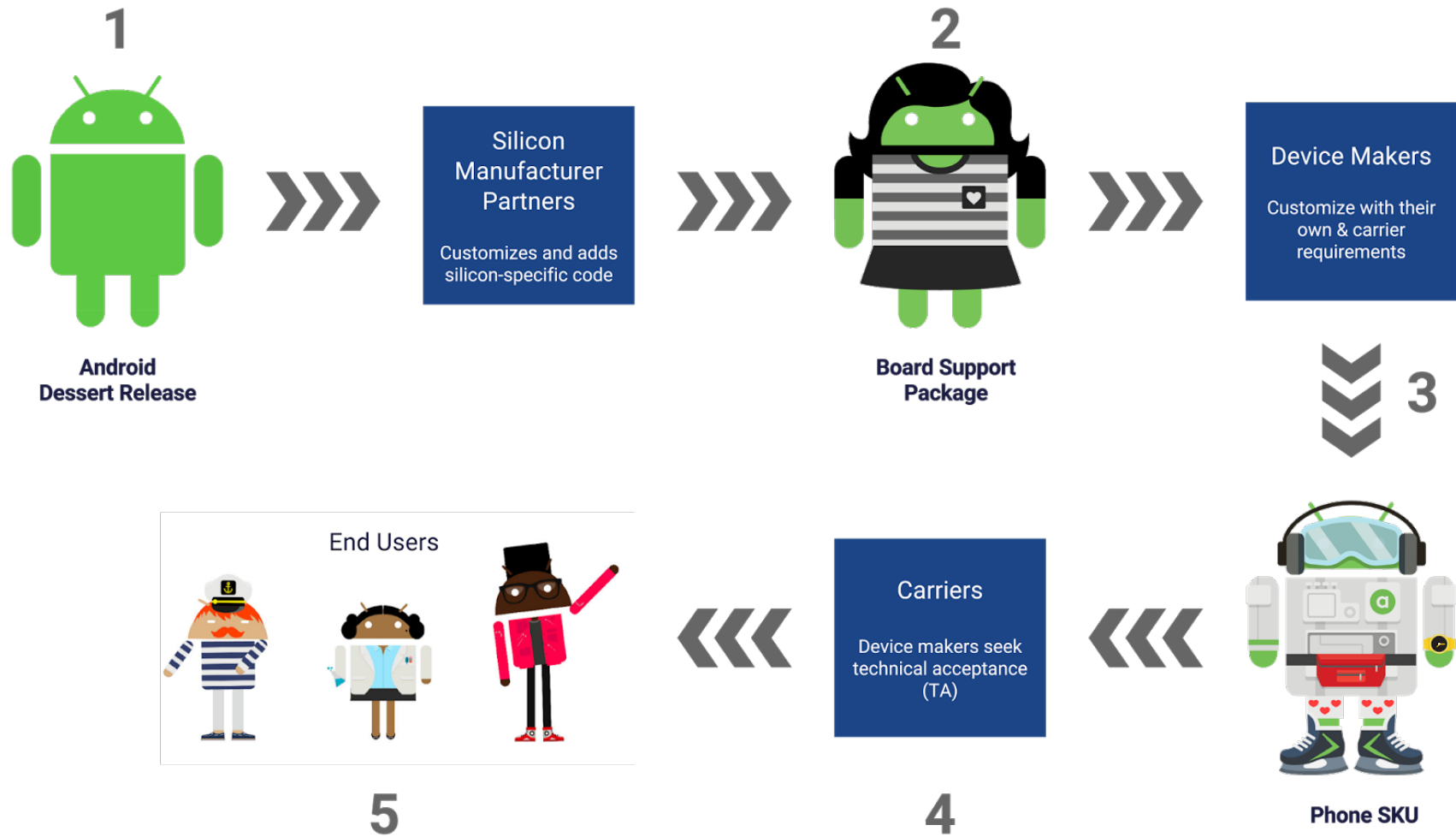
6.2. Compatibility Test Suite



7. Project Treble

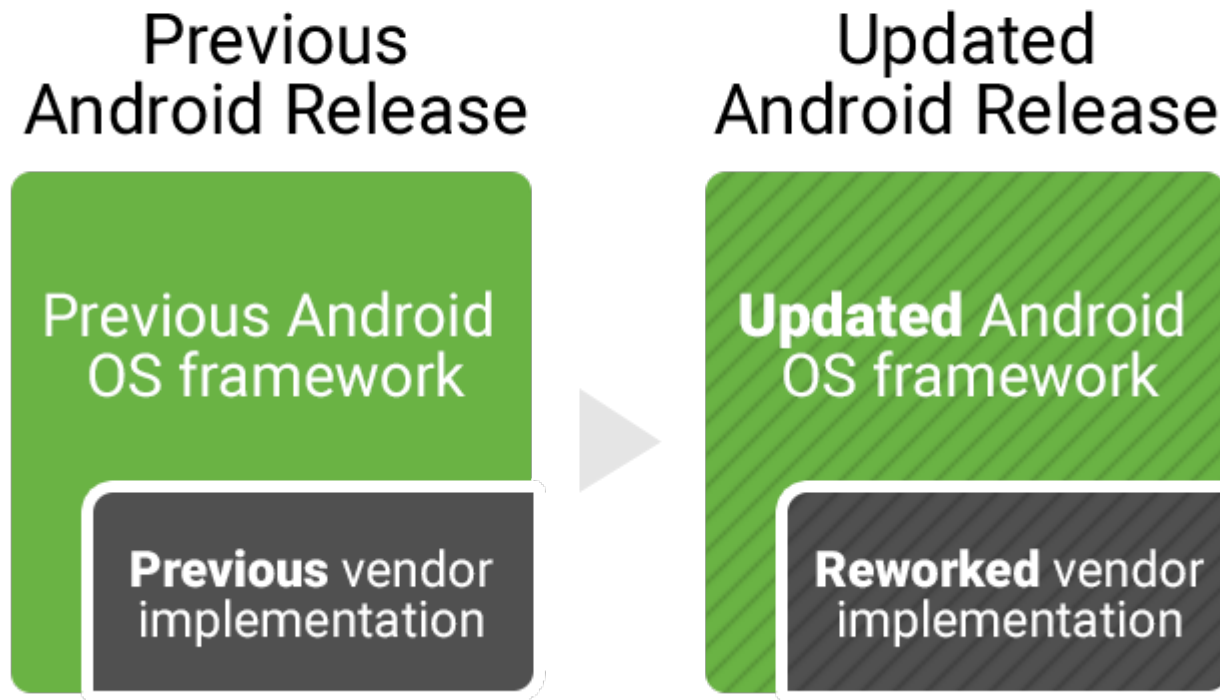
- Introduced with 8.x
- Major rework of stack internals
- Goal: easier migration to new versions
- Motivation – as of early 2018:
 - Vast majority of Apple devices run current release
 - Vast majority of Android devices don't even run the previous release yet.
- Tested by Vendor Test Suite (VTS)
- VTS now required for GMS certification

7.1. Releases before Treble



7.2. Updates before Treble

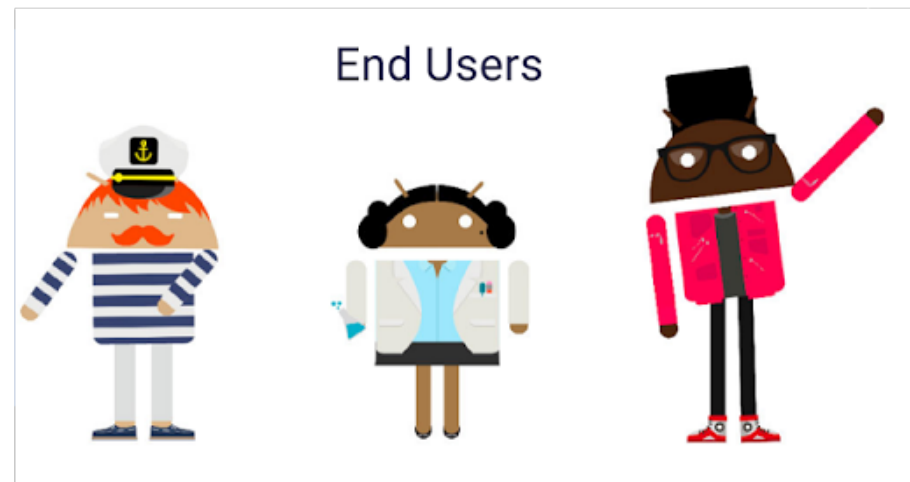
Before Treble



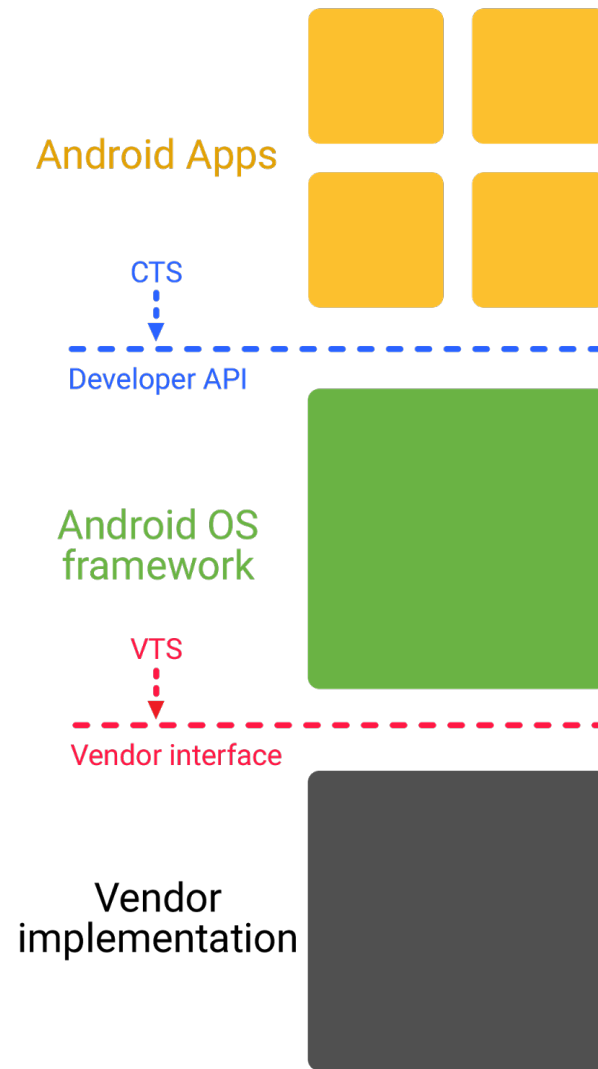
7.3. Treble's goal for updates



**Android
Dessert Release**

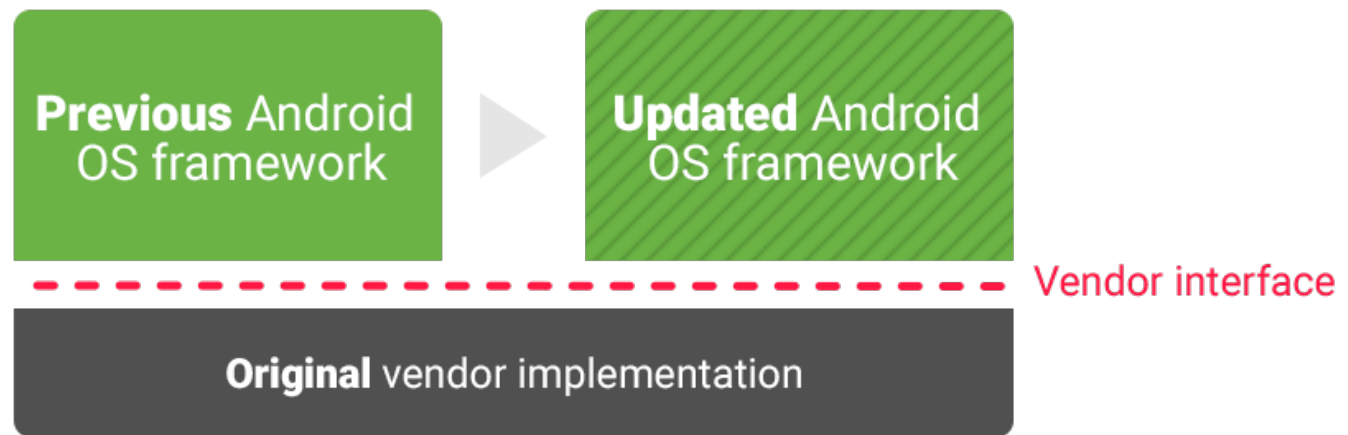


7.4. New Treble layers



7.5. Update with Treble

With Treble



8. Development tools

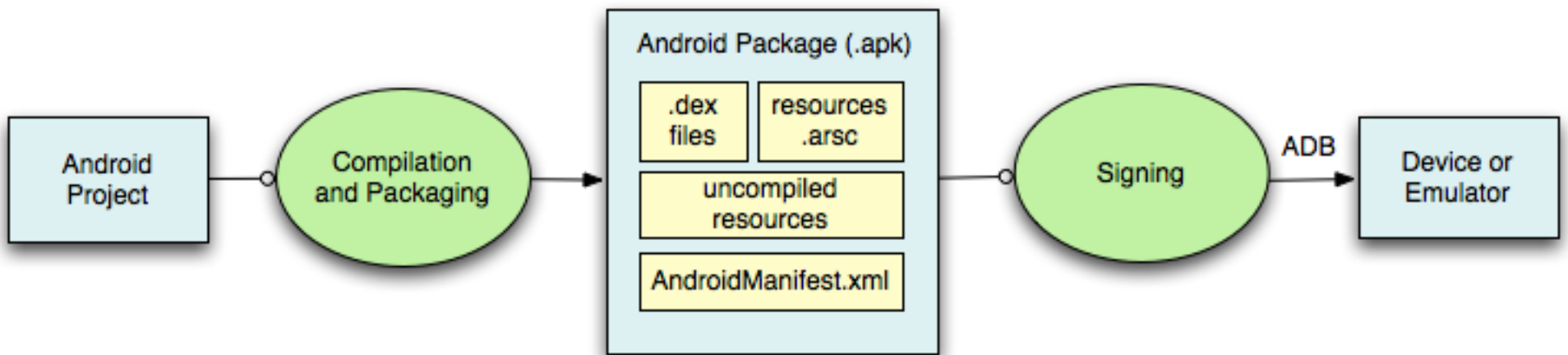
- Requirements
- App dev tools and resources
- App debugging

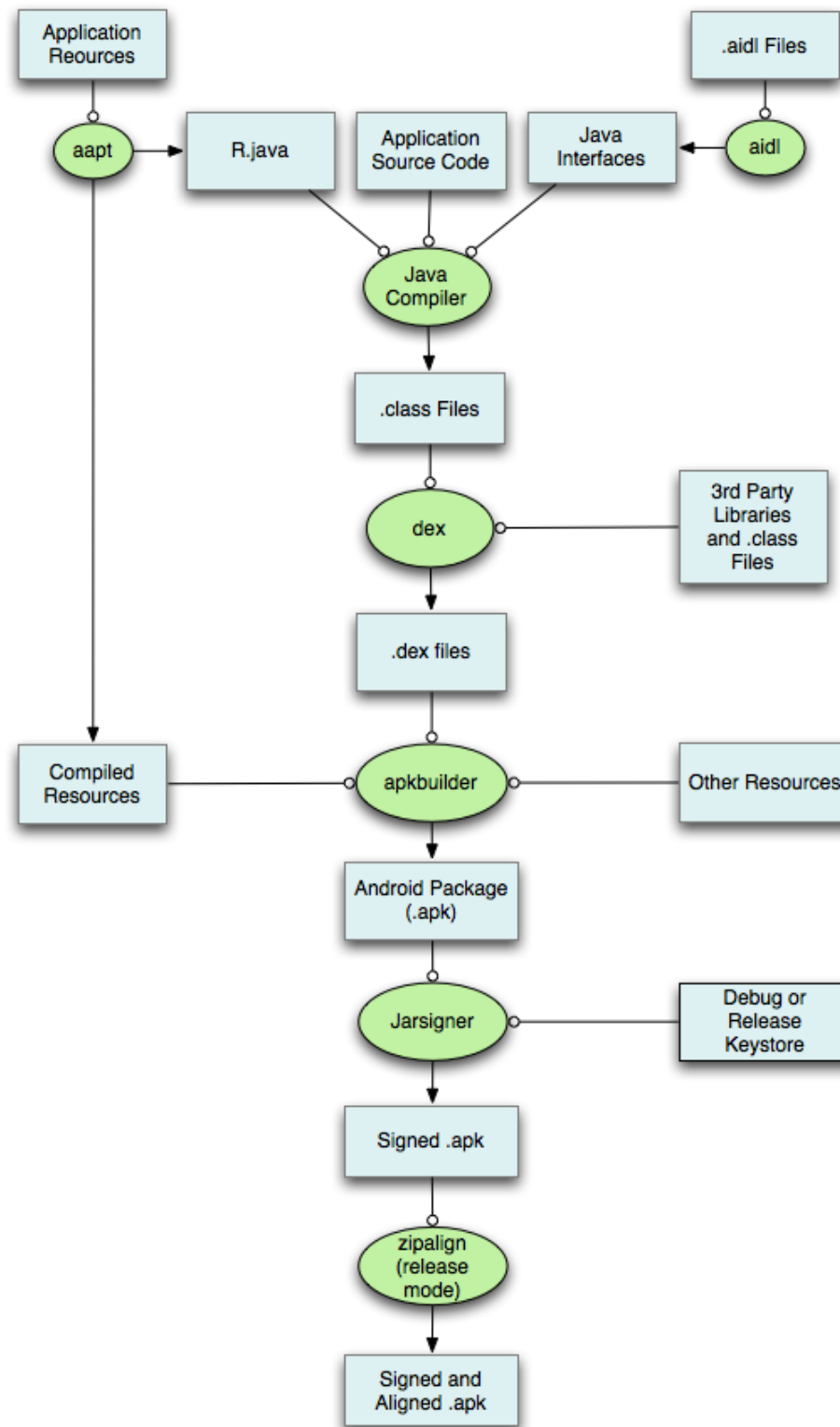
8.1. Requirements

- App development and debugging:
 - Windows / Mac / Linux workstation
 - JDK
 - Android Studio
 - Highly recommended: real device(**S**)
- Platform development:
 - GNU cross-dev toolchain
 - JTAG debugger
 - ... more on this later

8.2. App dev tools and resources

- SDK:
 - android – manage AVDs and SDK components
 - apkbuilder – creating .apk packages
 - dx – converting .jar to .dex
 - adb – debug bridge
 - ...
- Emulator – QEMU-based ARM emulator
 - Use KVM for x86 instead
- NDK: GNU toolchain for native binaries
- Documentation: developer.android.com





8.3. App debugging

- adb
- monitor
- monkeyrunner
- traceview
- logcat
- Android Studio

Concepts and Internals

1. Linux Concepts
2. Android Concepts
3. Overall Architecture
4. System startup
5. Linux Kernel
6. Hardware Support
7. Native User-Space
8. Android's Java
9. JNI
10. System Server
11. Calling on Services
12. Activity Manager
13. Binder
14. HAL
15. Inside Treble
16. Stock AOSP Apps

1. Linux Concepts

- Processes (fork()) and his friends)
- Signals (kill() ... or be killed)
- Sockets / Pipes / Fifos / SysV IPC
- Hardware devices as files (/dev)
- Daemons
- Shell / scripts
- Users (root vs. everyone else -- # vs. \$)
- ELF files
- GNU toolchain
- ... 40 years of Unix

2. Android Concepts

- Components
- Intents
- Manifest file
- Component lifecycle
- Processes and threads
- Remote procedure calls
- Permissions
- Storage
- Native development

2.1. Components

- 1 App = N Components
- Apps can use components of other applications
- App processes are automagically started whenever any part is needed
- Ergo: N entry points, !1, and !main()
- Components:
 - Activities
 - Services
 - Broadcast Receivers
 - Content Providers

2.2. Intents

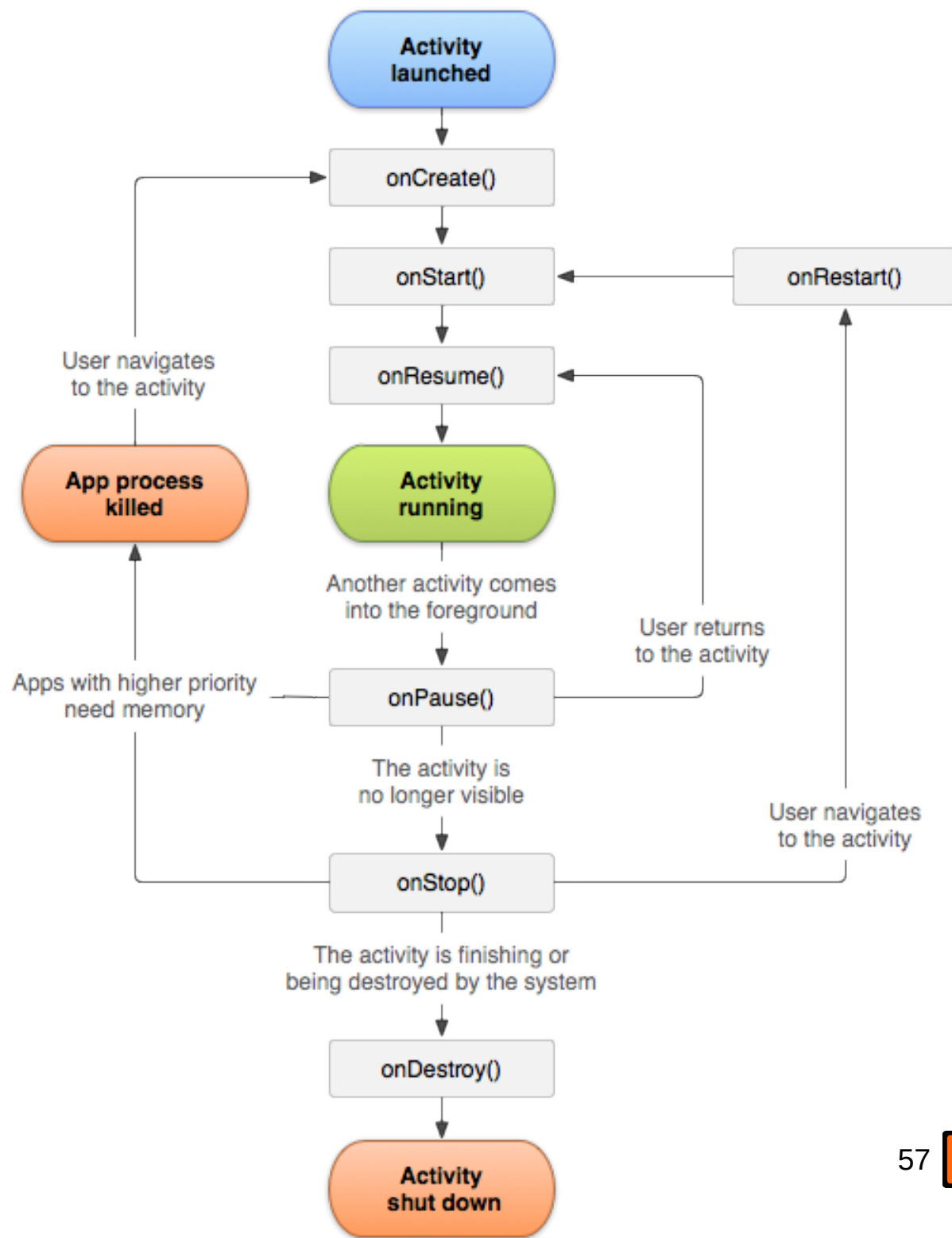
- Intent = asynchronous message w/ or w/o designated target
- Like a polymorphic Unix signal, but w/o required target
- Intents “payload” held in Intent Object
- Intent Filters specified in Manifest file

2.3. Manifest file

- Informs system about app's components
- XML format
- Always called AndroidManifest.xml
- Activity = `<activity> ... static`
- Service = `<service> ... static`
- Broadcast Receiver:
 - Static = `<receiver>`
 - Dynamic = `Context.registerReceiver()`
- Content Provider = `<provider> ... static`

2.4. Component lifecycle

- System automagically starts/stops/kills processes:
 - Entire system behaviour predicated on low memory
- System triggers Lifecycle callbacks when relevant
- Ergo: Must manage Component Lifecycle
- Some Components are more complex to manage than others



2.5. Processes and threads

- Processes
 - Default: all callbacks to any app Component are issued to the main process thread
 - <activity>—<service>—<recipient>—<provider> have process attribute to override default
 - Do NOT perform blocking/long operations in main process thread:
 - Spawn threads instead
 - Process termination/restart is at system's discretion
 - Therefore:
 - Must manage Component Lifecycle
- Threads:
 - Create using the regular Java Thread Object
 - Android API provides thread helper classes:
 - Looper: for running a message loop with a thread
 - Handler: for processing messages
 - HandlerThread: for setting up a thread with a message loop

2.6. Remote procedure calls

- Android RPCs = Binder mechanism
- No Sys V IPC due to in-kernel resource leakage
- Binder is a low-level functionality, not used as-is
- Instead: must define interface using Interface Definition Language (IDL)
- IDL fed to aidl Tool to generate Java interface definitions

2.7. Security/Permissions

- Most security enforced at process level: UID, GID
- Permissions enforce restrictions on:
 - Per-process operations
 - Per-URI access
- Applications are sandboxed
- Specific permissions required to “exit” sandbox
- Decision to grant access based on:
 - Certificates
 - User prompts
- All permissions must be declared statically

2.8. Data storage

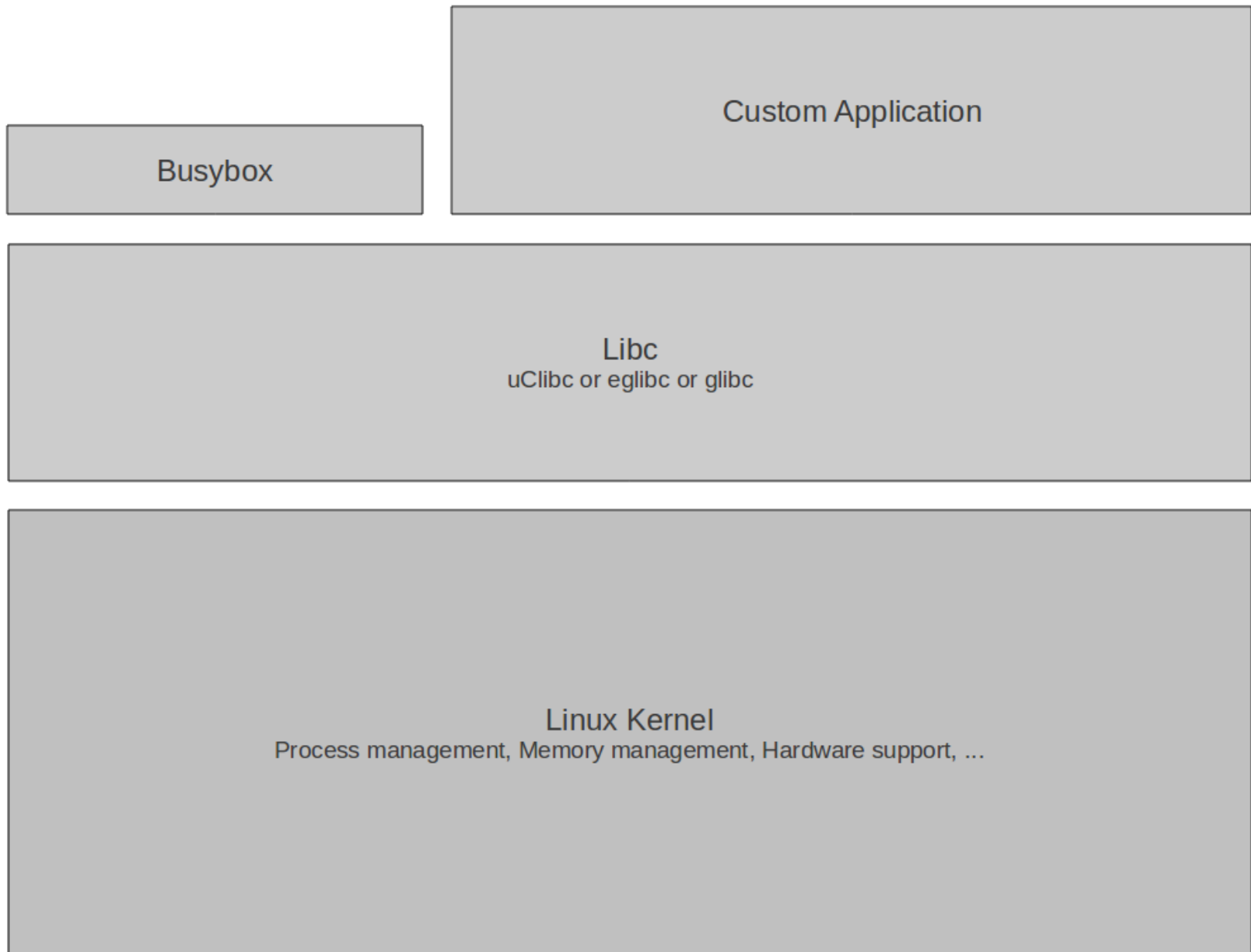
- Shared preferences
 - Private primitive key-pair values
- Internal storage
 - Private data on device memory
- External storage
 - Public data on shared external device (SD) -- emulated
- SQLite DB
 - Private DB
- Network connection
 - Web-based storage (REST)

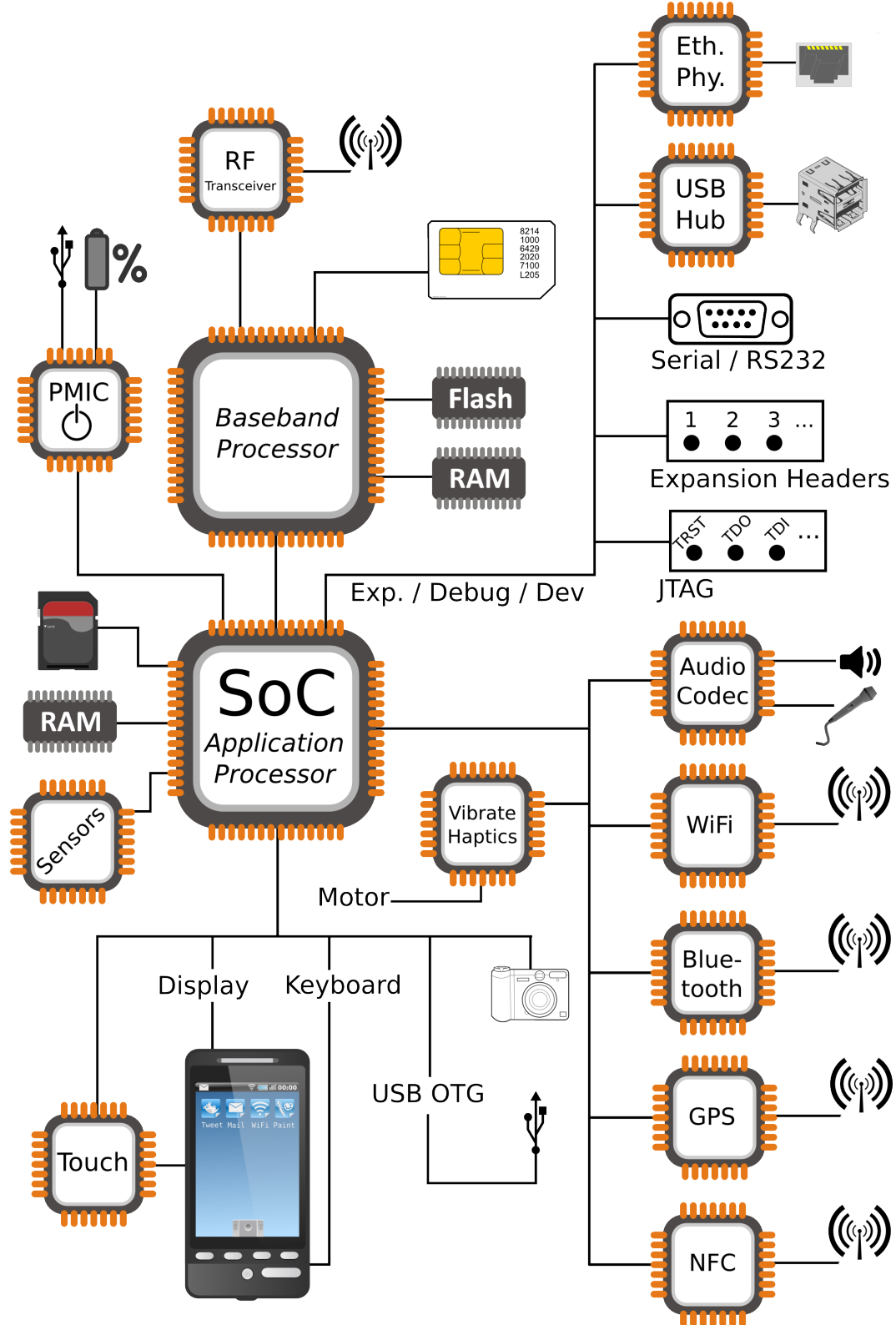
2.9. Native development

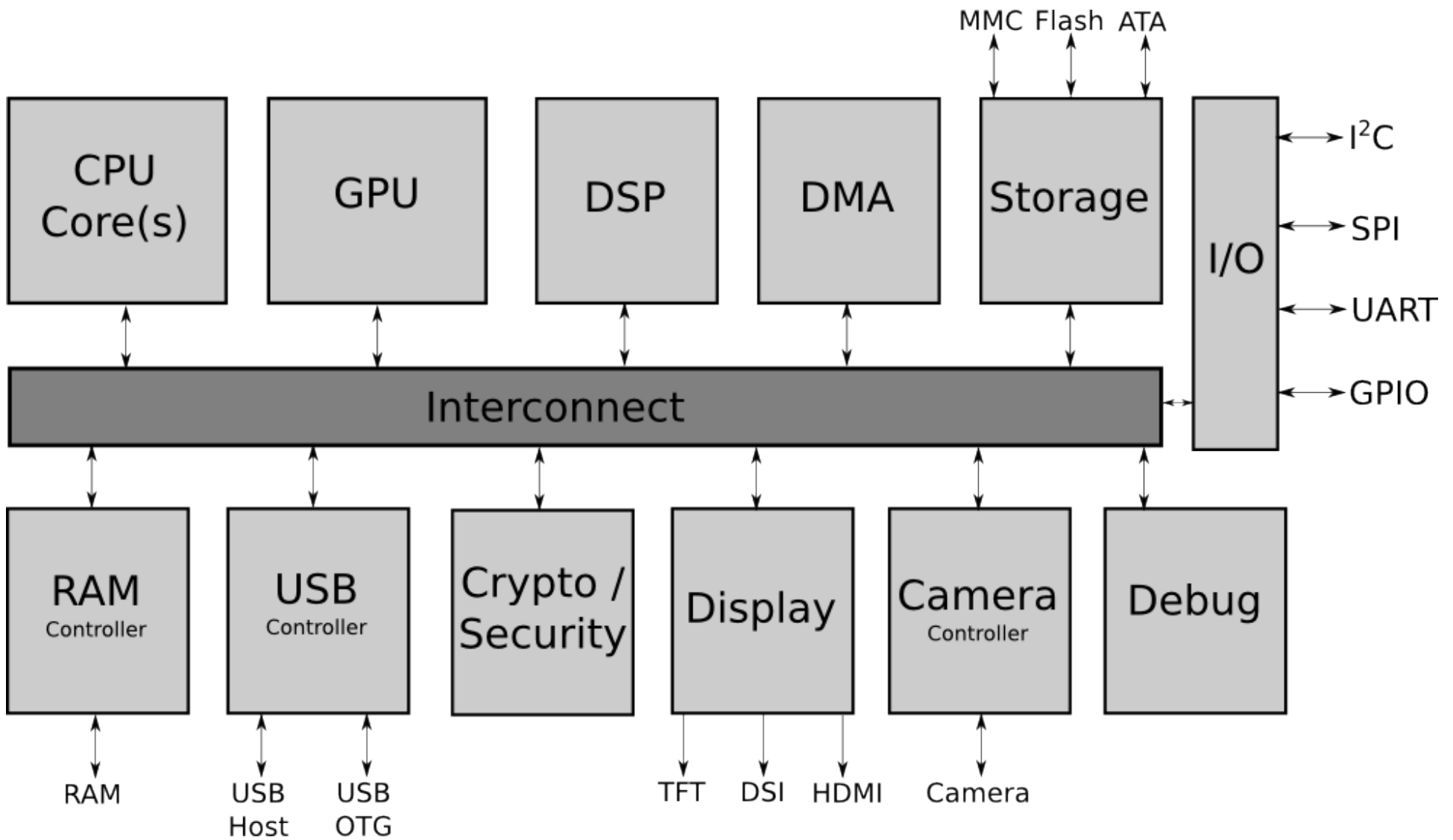
- Useful for:
 - Porting existing body of code to Android
 - Developing optimized native apps, especially for gaming
- Provides:
 - Tools and build files to generate native code libraries from C/C++
 - Way to embed native libs into .apk
 - Set of stable (forward-compatible) native libs
 - Documentation, samples and tutorials
- Enables:
 - Calling native code from Java using JNI
 - Implementing fully native apps (since 2.3)
- Doesn't allow you to:
 - Compile traditional Linux/Unix apps as-is

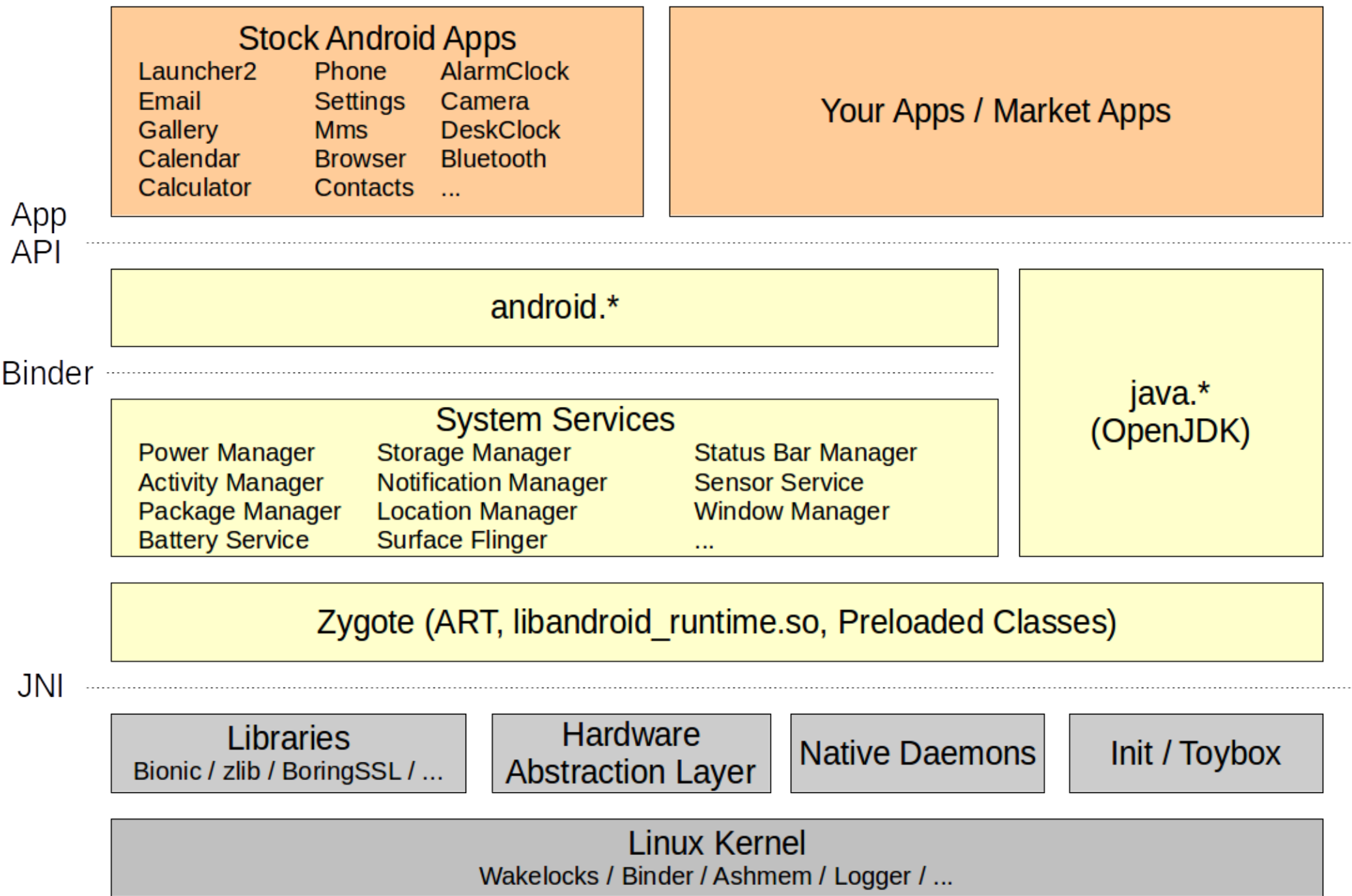
3. Architecture

- Embedded Linux
- Modern phone/tablet
- System-on-Chip (SoC)
- Android



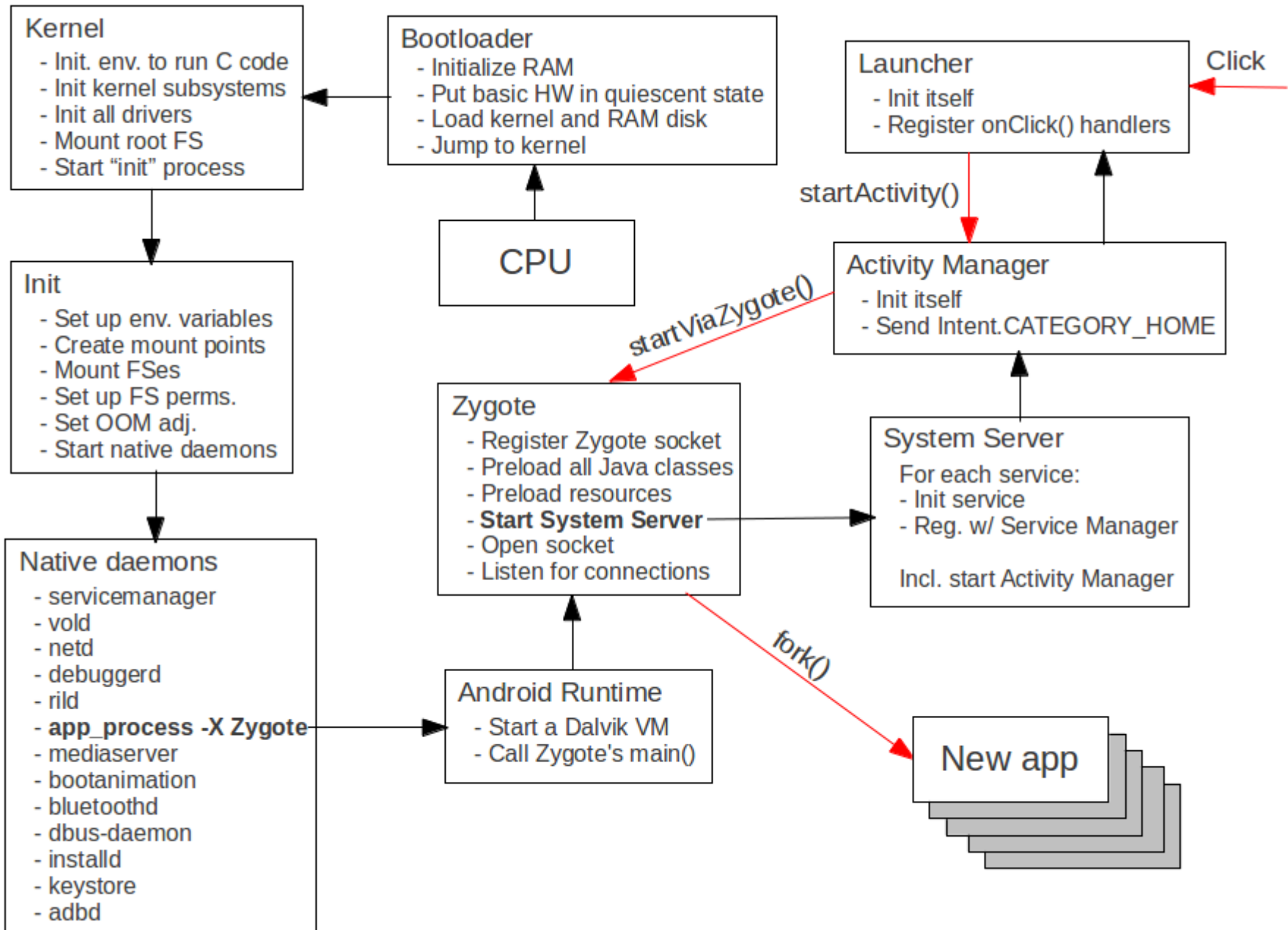






4. System Startup

- Bootloader
- Kernel
- Init
- Zygote
- System Server
- Activity Manager
- Launcher (Home)



4.1. Bootloader

- “Reference” bootloader:
 - <https://github.com/travisg/lk>
 - <https://www.codeaurora.org/blogs/little-kernel-based-android-bootloader>
- aosp/bootable/bootloader
 - Empty these days ...
- aosp/bootable/recovery
 - UI-based recovery boot program
 - Accessed through magic key sequence at boot
 - Usually manufacturer specific variant

- Storage layout:
 - Bootloader
 - “boot” image -- Kernel + RAM disk
 - “system” partition -- /system
 - “data” partition -- /data
 - “cache” partition -- /cache
 - “vendor” partition -- /vendor
 - “recovery” partition -- Alternate boot image

4.2. Kernel

- Early startup code is very hardware dependent
- Initializes environment for the running of C code
- Jumps to the architecture-independent `start_kernel()` function.
- Initializes high-level kernel subsystems
- Mounts root filesystem
- Starts the `init` process

4.3. Android Init

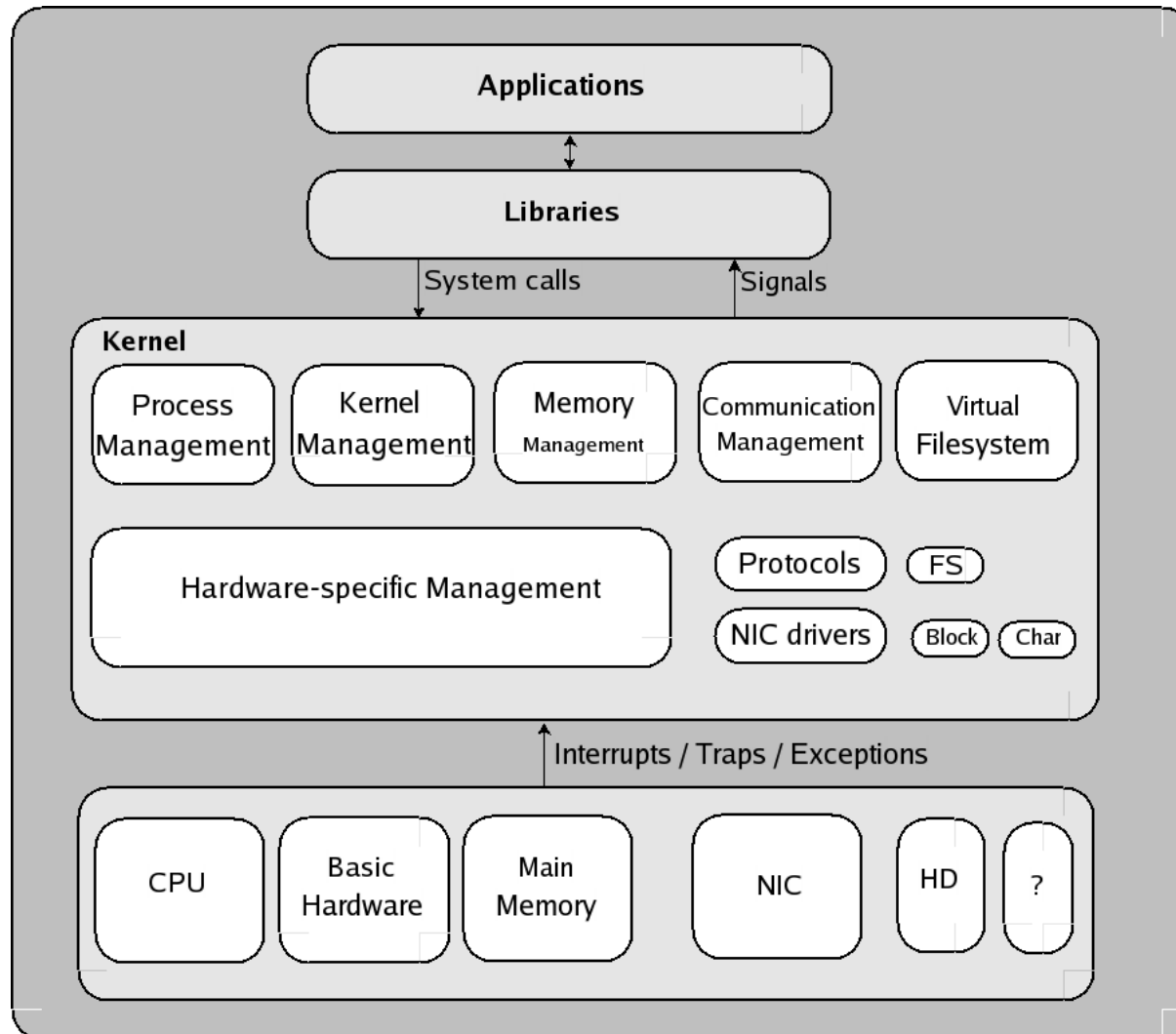
- Open, parses, and runs /init.rc:
 - Create mountpoints and mount filesystems
 - Set up filesystem permissions
 - Set OOM adjustments properties
 - Start daemons:
 - adbd
 - servicemanager (binder context manager)
 - vold
 - netd
 - rild
 - app_process -Xzygote (Zygote)
 - mediaserver
 - ...

4.4. Zygote, etc.

- Init:
 - `app_process -Xzygote (Zygote)`
- `frameworks/base/cmds/app_process/app_main.cpp`:
 - `runtime.start("com.android.internal.os.Zygote", ...`
- `frameworks/base/core/jni/AndroidRuntime.cpp`:
 - `startVM()`
 - Call Zygote's `main()`
- `frameworks/base/core/java/com/android/internal/os/ZygoteInit.java`:
 - ...

- preloadClasses()
 - startSystemServer()
 - ... magic ...
 - Call SystemServer's run()
- frameworks/base/services/java/com/android/server/SystemServer.java:
 - Start **all** system services/managers
 - Start ActivityManager:
 - Send Intent.CATEGORY_HOME
 - Launcher2 kicks in

5. Linux Kernel



6. Hardware Support

- Not accessed directly
- Use of HALs
- One HAL for each hardware type
- Before Oreo: HAL “modules” are .so files
- Treble: HALs are published APIs and can be binderized

- Activity Recognition
- Audio
- Bluetooth
- Camera
- ConsumerIr
- Framebuffer
- Fingerprint
- Fused Location

- GPS
- Gralloc
- HWcomposer
- Keymaster
- Lights
- NFS
- Power
- Sensors

7. Native User-Space

- Mainly
 - /data => User data
 - /system => System components
 - /cache => Cache (& OTA update)
- Also found:
 - /dev
 - /proc
 - /sys
 - /sbin
 - /mnt
 - Etc.
- Key directories gone ...

8. Android's Java

- Oracle (Sun) Java =
Java language + JVM + JDK libs
- Android Java =
Java language + ART + OpenJDK libs

8.1. ART

- “Android RunTime”
- 64 bit
- Multi-core
- AOT first, JIT second
- Better debugging capabilities (profiling/stack/crashes)
- Introduced as alternative to Dalvik
- Has now (largely) replaced Dalvik

9. JNI – Java Native Interface

- Call gate for other languages, such as C, C++
- If you know C#: Equivalent to .NET's pinvoke
- Usage: include and call native code from App
- Tools = NDK ... samples included
- Check out “*JNI Programmer's Guide and Specification*” -- no longer “freely available”

10. System Services

- The Android OS
- >100 or so on a modern Nougat
- 5 or 6 new at every major release
- Use “service list” to see current list

System Services

System Server

Java-built Services

Power Manager	Mount Service
Activity Manager	Notification Manager
Package Manager	Location Manager
Battery Service	Search Service
Window Manager	Wallpaper Service
Status Bar	Headset Observer
Clipboard Service	...

C-built Services

Sensor Service

Surface Flinger

Media Service

Audio Flinger
Media Player Service
Camera Service
Audio Policy Service

Includes:

- StageFright
- Audio effects
- DRM framework

Phone App

JNI

Native Methods for
Java-built Services

Hardware Abstraction Layer

11. Calling on System Services

- Use `getSystemService`
- Ex: NotificationManager Object reference:

```
String ns = Context.NOTIFICATION_SERVICE;
```

```
NotificationManager mNotificationManager = (NotificationManager) \  
getSystemService(ns);
```

- Prepare your content
- Call on the object:

```
mNotificationManager.notify(HELLO_ID, notification);
```

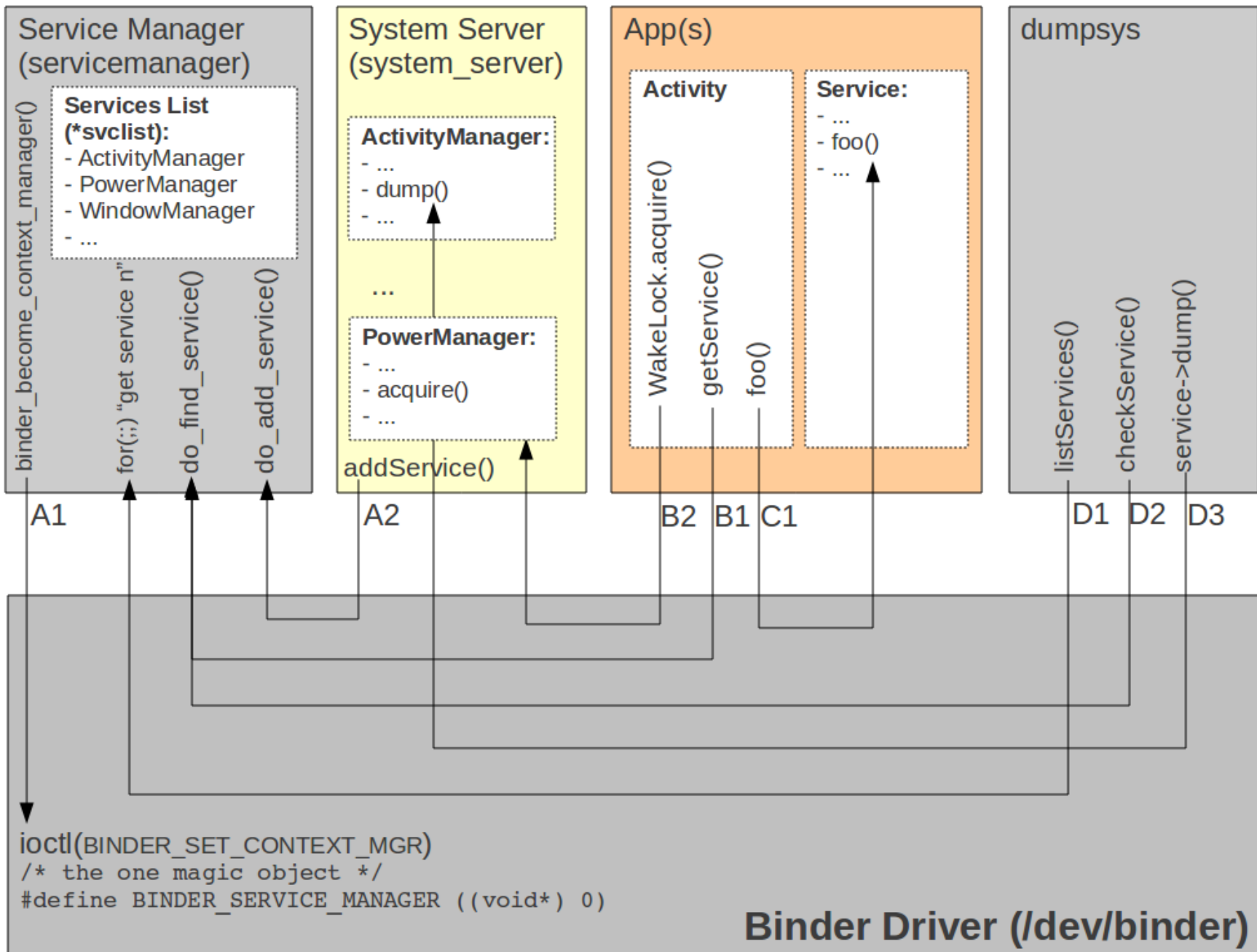
12. ActivityManager

- Start new Activities, Services
- Fetch Content Providers
- Intent broadcasting
- OOM adj. maintenance
- Application Not Responding
- Permissions
- Task management
- Lifecycle management

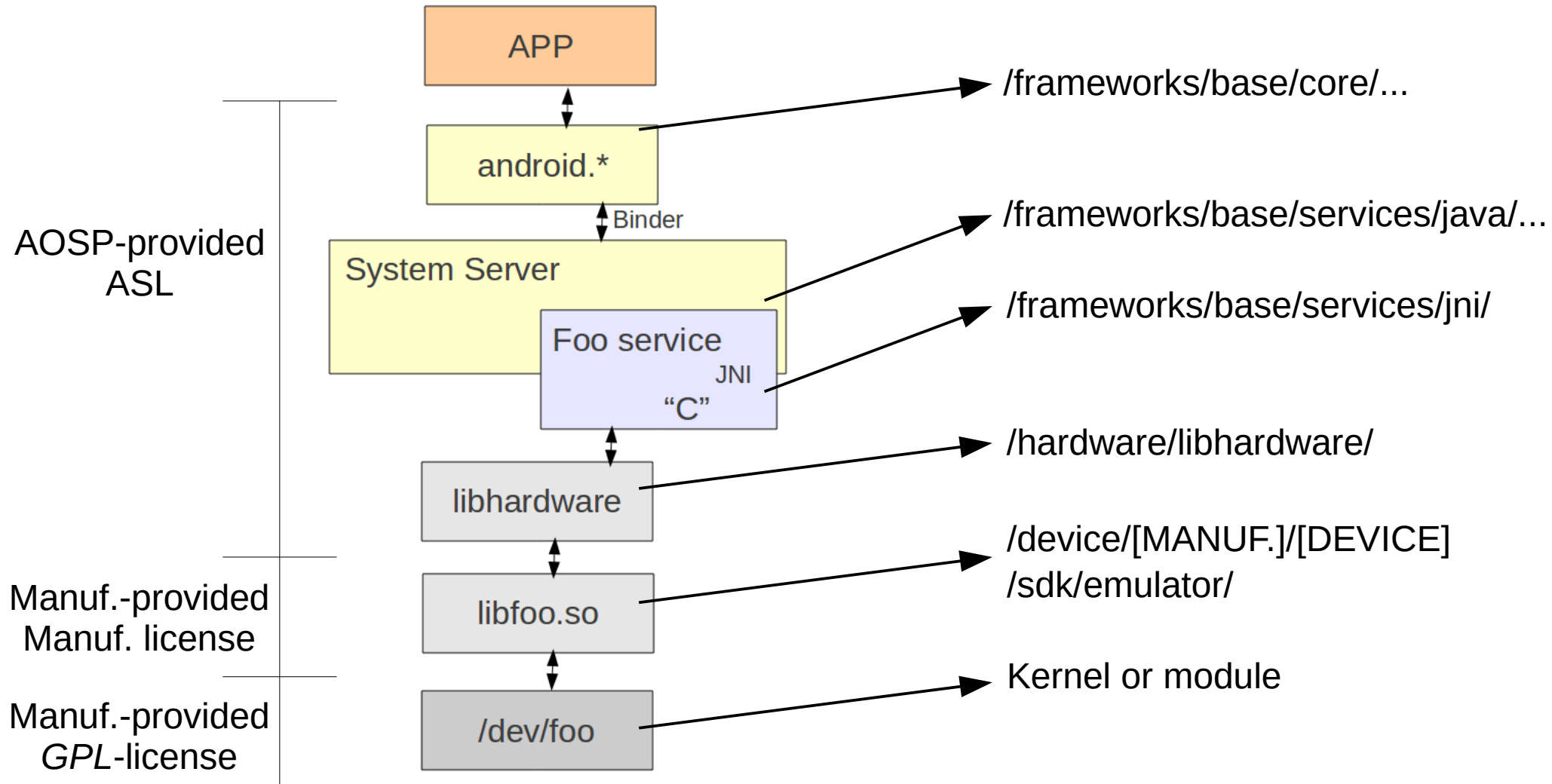
- Ex. starting new app from Launcher:
 - onClick(Launcher)
 - startActivity(Activity.java)
 - *<Binder>*
 - ActivityManagerService
 - startViaZygote(Process.java)
 - *<Socket>*
 - Zygote

13. Binder

- CORBA/COM-like IPC
- Data sent through “parcels” in “transactions”
- Kernel-supported mechanism
- /dev/binder
- Check /sys/kernel/debug/binder/*
- android.* API connected to System Server through binder.



14. Hardware Abstraction Layer



15. Inside Treble

- Linux kernel standardization on LTS releases
- Hardware Interface Definition Language (HIDL)
- Vendor NDK (VNDK)
- Vendor Interface Object (VINTF)
- Vendor Test Suite (VTS)

15.1. Treble's impact

- Some modifications now off-limits:
 - System service internals
 - HAL definition reworking/extension
 - Core library replacement by tweaking dependent components' APIs.
 - Tweaking existing SE policies
 - Any change that impacts OTA
 - Any change that breaks VTS
- Need to make sure partners/vendors are creating Treble-compliant BSPs, additions, codebase, etc.

16. Stock AOSP Apps

/packages/apps

AccountsAndSettings
AlarmClock
Bluetooth
Browser
Calculator
Calendar
Camera
CertInstaller
Contacts
DeskClock
Email
Gallery
HTMLViewer

Launcher2
Mms
Music
PackageInstaller
Protips
Provision
QuickSearchBox
Settings
SoundRecorder
SpeechRecorder
Stk
VoiceDialer

/packages/providers

ApplicationProvider
CalendarProvider
ContactsProvider
DownloadProvider
DrmProvider
GoogleContactsProvider
MediaProvider
TelephonyProvider
UserDictionaryProvider

/packages/inputmethods

LatinIME
OpenWnn
PinyinIME

Working with the Android Open Source Project

- Tools and location
- Content
- Building
- Build tricks
- Build system architecture
- Output images
- Using adb
- Custom toolchains and dev kits
- Compatibility Test Suite
- Basic hacks

1. Tools and location

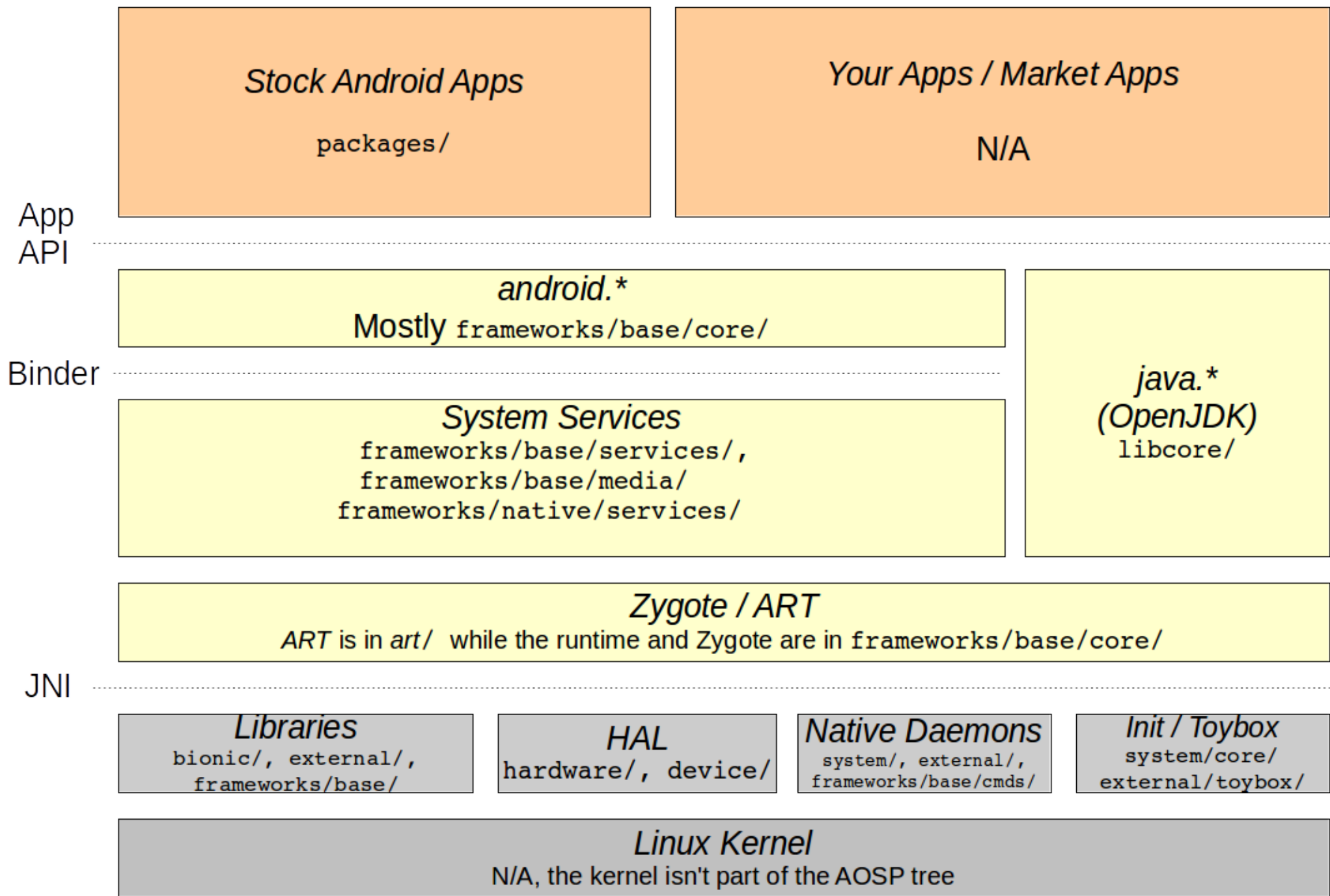
- Location:
 - <http://android.googlesource.com/>
- Get “repo”:

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo  
$ chmod a+x ~/bin/repo
```
- Fetch the AOSP:
 - Make sure you fetch a tagged release
 - Pie:

```
$ repo init -u https://android.googlesource.com/platform/manifest -  
b android-9.0.0_r36  
$ repo sync
```

2. Content

art	Android Runtime
bionic	C library replacement
bootable	Reference bootloader
build	Build system
cts	Compatibility Test Suite
dalvik	Dalvik VM
development	Development tools
device	Device-specific files and components
external	Copy of external projects used by AOSP
frameworks	System services, android.*, Android-related cmds, etc.
hardware	Hardware support libs
libcore	Apache Harmony
ndk	The NDK
packages	Stock Android apps, providers, etc.
prebuilt	Prebuilt binaries
sdk	The SDK
system	pieces of the world that are the core of the embedded linux platform at the heart of Android.



3. Building

- Requires Ubuntu -- works with 18.04, 16.04, 14.04, and 12.04
- See <https://source.android.com/source/initializing.html>
- Required packages
- Possible fixes
- Building on Mac

- Set up build environment:

```
$ ./build/envsetup.sh
```

```
$ lunch
```

- Launch build and go watch tonight's hockey game:

```
$ make -j8
```

- ... though you should check your screen at breaks ...

- Flash:

```
$ adb reboot bootloader
```

```
$ fastboot oem unlock
```

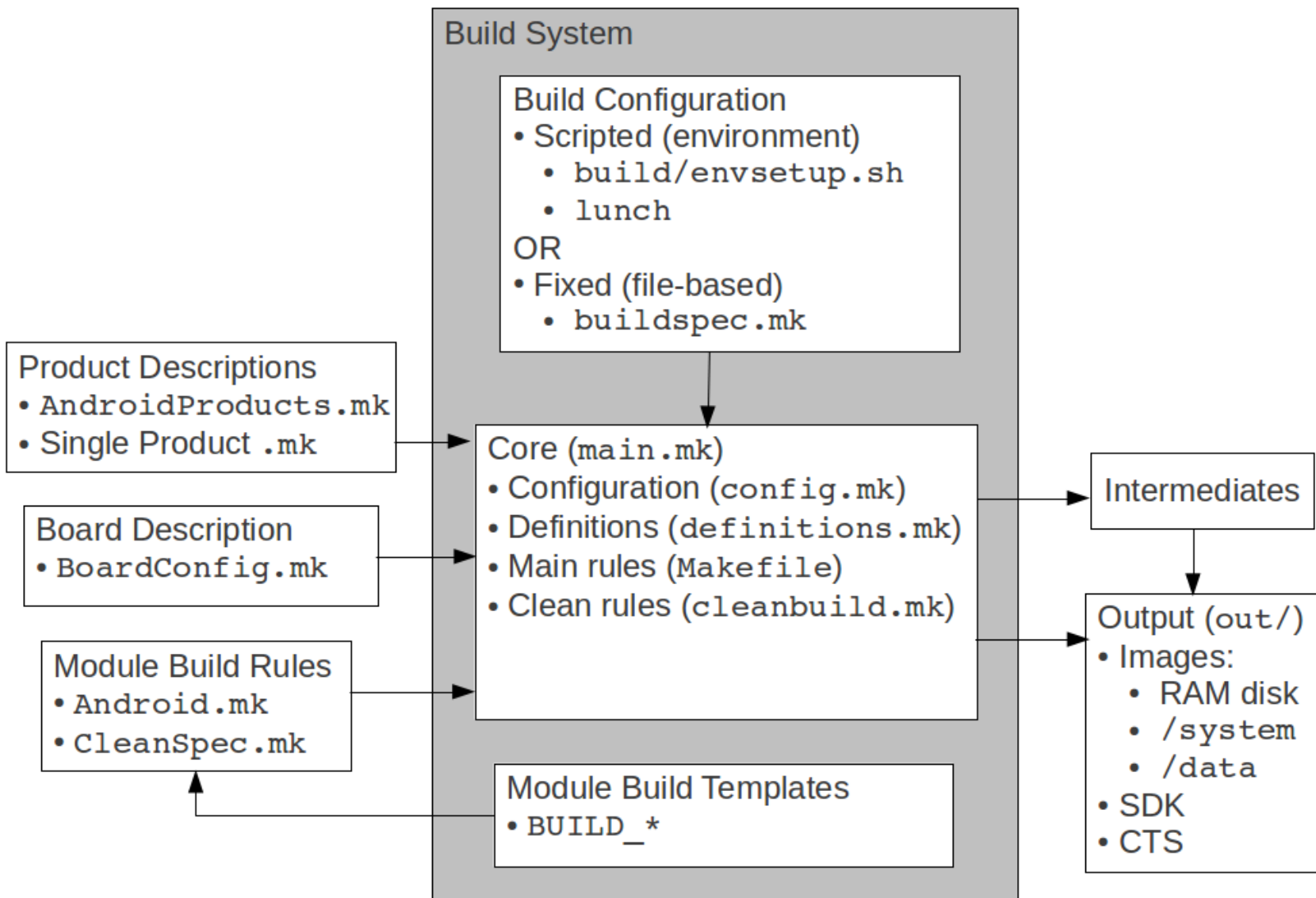
```
$ fastboot flashall
```

- Need to reuse envsetup.sh and lunch on every new shell

4. Build Tricks

- Commands (from build/envsetup.sh):
 - godir
 - croot
 - mm
 - m
- Most important:
 - hmm
 - make help
- Speeding up the Build:
 - CPU
 - RAM
 - SSD
 - CCACHE
 - \$ export USE_CCACHE=1
 - Seed out with pre-built binaries from your local repo

5. Build System Architecture



6. Output Images

- All output and build in [aosp]/out/
- Images at [aosp]/out/target/product/hikey/:
 - boot.img
 - cache.img
 - ramdisk.img
 - system.img
 - userdata.img
- Kernel is in:
 - device/linaro/hikey-kernel
- Emulator overrides -- if you use qemu:
 - -kernel
 - -initrd

7. Using adb

- Can use to control/interface w/ running AOSP, including emulator.

- Shell:

```
$ adb shell
```

```
#
```

Host

Target

- Dumping the log:

```
$ adb logcat
```

- Copying files to/from target:

```
$ adb push foo /data/local
```

```
$ adb pull /proc/config.gz
```

8. Custom Toolchains and Dev Kits

- Rationale
- SDK generation
- NDK generation

8.1. Rationale

- SDK:
 - Providing other internal teams or external developers access to your modified/custom Android APIs.
- NDK:
 - Same as SDK rationale
- Custom cross-dev toolchain:
 - To avoid having to use a binary toolchain from 3rd party.
 - To control the build parameters used to create the toolchain. Ex.: use uClibc instead of glibc.

8.2. SDK generation

- Building the SDK:
 - \$. build/envsetup.sh
 - \$ lunch sdk-eng
 - \$ make sdk
- If API modified, do this before make:
 - \$ make update-api
- Location: [aosp]/out/host/linux-x86/sdk/
- Using a custom SDK:
 - Modify Android Studio config

8.3. NDK generation

- Build

```
$ cd ndk/build/tools
```

```
$ export ANDROID_NDK_ROOT=[aosp]/ndk
```

```
$ ./make-release --help
```

```
$ ./make-release
```

IMPORTANT WARNING !!

This script is used to generate an NDK release package from scratch for the following host platforms: linux-x86

This process is **EXTREMELY LONG** and may take **SEVERAL HOURS** on a dual-core machine. If you plan to do that often, please read docs/DEVELOPMENT.TXT that provides instructions on how to do that more easily.

Are you sure you want to do that [y/N]

9. Compatibility Test Suite

- Android Compatibility Program:
 - Source code to Android stack
 - Compatibility Definition Document (CDD) – Policy
 - Compatibility Test Suite (CTS) – Mechanism
- Each Android version has own CDD & CTS
- CTS:
 - Part of AOSP
 - Run from host using USB over to attached device
 - Based on JUnit
 - Runs various test apps on target
 - Relies on ADB
 - Provides report to be analyzed and/or sent back to Google



On your machine

Download and install the CTS.

Run the CTS.

Store and view results.

Devices you attach to your machine



Test execution

Test results

- Report:
 - .zip file containing XML files and screen-shots
 - Sent to: cts@android.com
- Building the CTS:

```
$ . build/envsetup.sh  
$ make cts
```
- Launching the CTS:

```
$ cd out/host/linux-x86/bin/  
$ ./cts  
$ cts_host >  
$ cts_host > help  
...
```

- Using the CTS:

```
$ cts_host > ls --plan
```

List of plans (8 in total):

Signature

RefApp

VM

Performance

AppSecurity

Android

Java

CTS

```
$ ./cts start --plan CTS
```

- Areas covered:
 - Signature tests
 - Platform API tests
 - Android Runtime tests
 - Platform Data Model
 - Platform Intents
 - Platform Permissions
 - Platform Resources

10. Basic Hacks

- Add device
- Add app
- Add app overlay
- Add native tool or daemon
- Add native library

10.1. Adding a new device

- Create directory in device/
 - device/acme/coyotepad/
- Makefile checklist:
 - AndroidProducts.mk
 - full_coyotepad.mk
 - BoardConfig.mk
 - Android.mk
- Menu integration:
 - vendorsetup.sh

10.1.1. AndroidProducts.mk

```
PRODUCT_MAKEFILES := \  
$(LOCAL_DIR)/full_coyotepad.mk
```

10.1.2. full_coyotepad.mk

```
$(call inherit-product,  
$(SRC_TARGET_DIR)/product/languages_full.mk)  
# If you're using 4.2/Jelly Bean, use full_base.mk instead of  
full.mk  
$(call inherit-product, $(SRC_TARGET_DIR)/product/full.mk)  
DEVICE_PACKAGE_OVERLAYS :=  
PRODUCT_PACKAGES +=  
PRODUCT_COPY_FILES +=  
PRODUCT_NAME := full_coyotepad  
PRODUCT_DEVICE := coyotepad  
PRODUCT_MODEL := Full Android on CoyotePad, meep-meep
```

10.1.3. BoardConfig.mk

```
TARGET_NO_KERNEL := true  
TARGET_NO_BOOTLOADER := true  
TARGET_CPU_ABI := armeabi  
BOARD_USES_GENERIC_AUDIO := true  
USE_CAMERA_STUB := true
```

10.1.4. Android.mk

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)
ifneq ($(filter coyotepad,$(TARGET_DEVICE)),)
include $(call all-makefiles-under,$(LOCAL_PATH))
endif
```

10.1.5. vendorsetup.sh

```
add_lunch_combo full_coyotepad-eng
```

10.2. Adding new applications

- Can't use Android Studio -- Because of Gradle
- Local:
 - Add app to device/acme/coyotepad/
 - Add Android.mk to app
 - Add to PRODUCT_PACKAGES in full_coyotepad.mk
- Global:
 - Add application in [aosp]/packages/apps
 - Add an appropriate Android.mk file to project
 - Add project to PRODUCT_PACKAGES in [aosp]/build/target/product/core.mk

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := $(call all-java-files-under, src)
LOCAL_PACKAGE_NAME := HelloWorld
include $(BUILD_PACKAGE)
```


10.3. Adding an app overlay

- `device/acme/coyotepad/overlay/`
- `full_coyotepad.mk`:
 - `DEVICE_PACKAGE_OVERLAYS := device/acme/coyotepad/overlay`

10.4. Adding a native tool or daemon

- Local:
 - device/acme/coyotepad/
- Global:
 - system/core/
 - system/
 - frameworks/base/cmds/
 - frameworks/native/cmds/
 - external/

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := hello-world
LOCAL_MODULE_TAGS := optional
LOCAL_SRC_FILES := hello-world.cpp
LOCAL_SHARED_LIBRARIES := liblog
include $(BUILD_EXECUTABLE)
```

10.5. Add a native library

- Local:
 - device/acme/coyotepad/
- Global:
 - system/core/
 - frameworks/base/libs/
 - frameworks/native/libs/
 - external/

```
LOCAL_PATH:= $(call my-dir)
include $(CLEAR_VARS)
LOCAL_MODULE := libmylib
LOCAL_MODULE_TAGS := optional
LOCAL_PRELINK_MODULE := false
LOCAL_SRC_FILES := $(call all-c-files-under,.)
include $(BUILD_SHARED_LIBRARY)
```

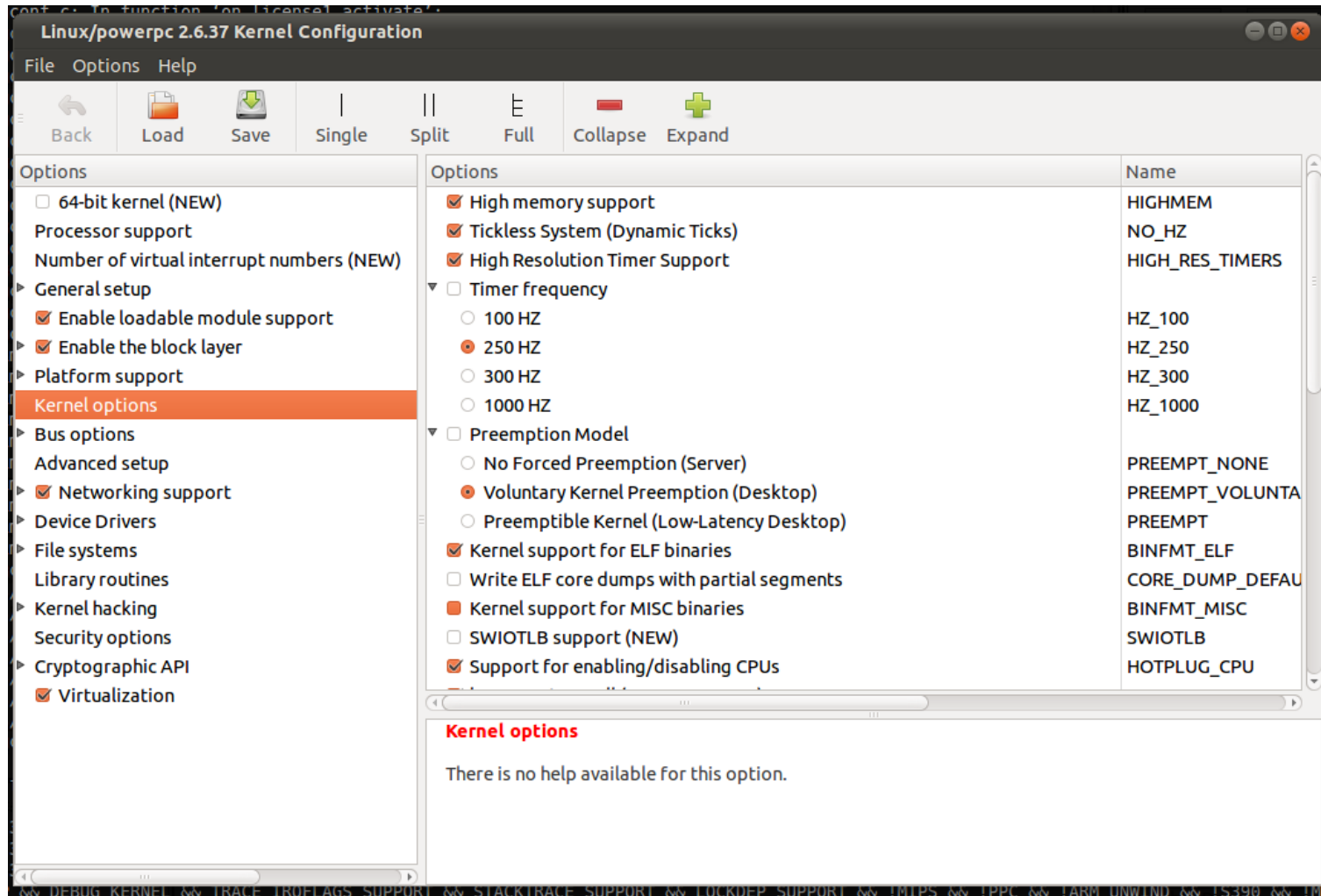
Kernel basics

1. Selecting a kernel
2. Configuring the kernel
3. Compiling the kernel
4. Installing the kernel
5. Useful boot parameters
6. Kernel patches

1. Selecting a kernel

- Google:
 - <http://android.googlesource.com>
- Vanilla:
 - <http://www.kernel.org>
- ~~Either way ... you're screwed:~~
 - ~~Android kernel is a fork~~
 - ~~No resolution in sight~~
 - ~~**Cannot** use vanilla kernel as-is ... androidisms~~
- Recent kernels have most Androidisms
- Android team continues to add stuff in their fork
- Talk to your SoC vendor

2. Configuring the kernel



- Main options:
 - Processor support
 - General setup
 - Enable loadable module support
 - Enable the block layer
 - Platform support
 - Kernel options
 - Bus options
 - Advanced setup
 - Networking support
 - Device Drivers
 - File systems
 - Library routines
 - Kernel hacking
 - Security options
 - Cryptographic API

- Kernel architecture name (ARCH=):
 - x86 => x86
 - ARM => arm
 - ARM64 => arm64
 - PPC => powerpc
 - mips => mips
 - sh => sh
- Some options are visible only to certain archs

- The fact that an option is displayed doesn't mean it's supported.
- The fact that an option isn't displayed doesn't mean it isn't supported.
- Configuration methods:
 - `make config`
 - `make oldconfig`
 - `make menuconfig`
 - `make xconfig`
- All config methods generate a `.config` file

- Configs will also generate headers & symlinks
- Set up temporary env. var. for cross-building:

```
$ export TARGET=[your_aosp]/prebuilt/linux-x86/toolchain/arm-eabi-4.4.0/bin/arm-eabi
```
- To start configuration menu for your target:

```
$ make ARCH=arm CROSS_COMPILE=${TARGET}- menuconfig
```
- Some items can be configured as modules and loaded dynamically at runtime.
- Once the configuration is done, quit the menu and save your configuration. This will create a .config file.
- Some targets have preset configurations:

```
$ make ARCH=arm64 CROSS_COMPILE=${TARGET}- \  
> hikey_defconfig
```

- Can use the menus provided by menuconfig and xconfig to load and save various configurations.
- Simplest way to manipulate configurations: copy the .config files by hand.
- To reuse an old configuration, simply copy the .config back to the kernel's tree and:

```
$ make ARCH=arm CROSS_COMPILE=${TARGET}- oldconfig
```
- Place all your .config files in your `${PRJROOT}/kernel` directory for easy access.
- Rename your .config files using meaningful names: `2.6.37.config`, `2.6.37-frame-buf.config`, ...

3. Compiling the kernel

1. Building the kernel:

- Example command – depends on target:

```
$ make ARCH=arm CROSS_COMPILE=${TARGET}- zImage
```

- Generates a kernel image compressed via gzip
- zImage may not be valid target. Other targets include: vmlinux, bzImage, ulmage, culmage, ...
- Use “make ARCH=... help” to find out valid targets
- ARCH variable indicates which architecture subdirectory is to be used.
- CROSS_COMPILE is used to form the name of the tools. For example, `$(CROSS_COMPILE) gcc` becomes `arm-linux-gcc`. Hence the trailing “-”.

2. Building the modules:

```
$ make ARCH=arm CROSS_COMPILE=${TARGET}- modules
```

- **IF YOU NEED TO** restore kernel tree to distribution state:

```
$ make ARCH=arm CROSS_COMPILE=${TARGET}- distclean
```

4. Installing the kernel¹

- Managing multiple kernel images:
 - Images in `${PRJROOT}/images` must be properly identified.
 - For each kernel, there are 4 files to put in `{PRJROOT}/images`:
 - The uncompressed image `=> vmlinux`
 - The compressed image `=> depends on arch`
 - The kernel symbols `=> System.map`
 - The kernel configuration file `=> .config`
 - Compressed image is *usually* in the `arch/YOUR_ARCH/boot` directory.

1. This is for **future** reference. `$PRJROOT` is undefined at this point

- See the exercise set for target image we are using.
- Location of images can be seen by looking at `arch/YOUR_ARCH/Makefile`
- Images placed in `${PRJROOT}/images` must follow same naming convention as `.config` file:

```
$ cp arch/arm/boot/zImage \  
> ${PRJROOT}/images/zImage-2.6.37  
  
$ cp System.map ${PRJROOT}/images/System.map-2.6.37  
  
$ cp vmlinux ${PRJROOT}/images/vmlinux-2.6.37  
  
$ cp .config ${PRJROOT}/images/2.6.37.config
```

- Installing the kernel modules:

- Must install modules in directory following the previously adopted naming convention:

```
$ make ARCH=arm CROSS_COMPILE=${TARGET}- \  
> INSTALL_MOD_PATH=${PRJROOT}/images/modules-2.6.37 \  
> modules_install
```

- `INSTALL_MOD_PATH` is prepended to the default `/lib/modules`. Hence, modules installed in:

```
${PRJROOT}/images/modules-2.6.37/lib/modules
```

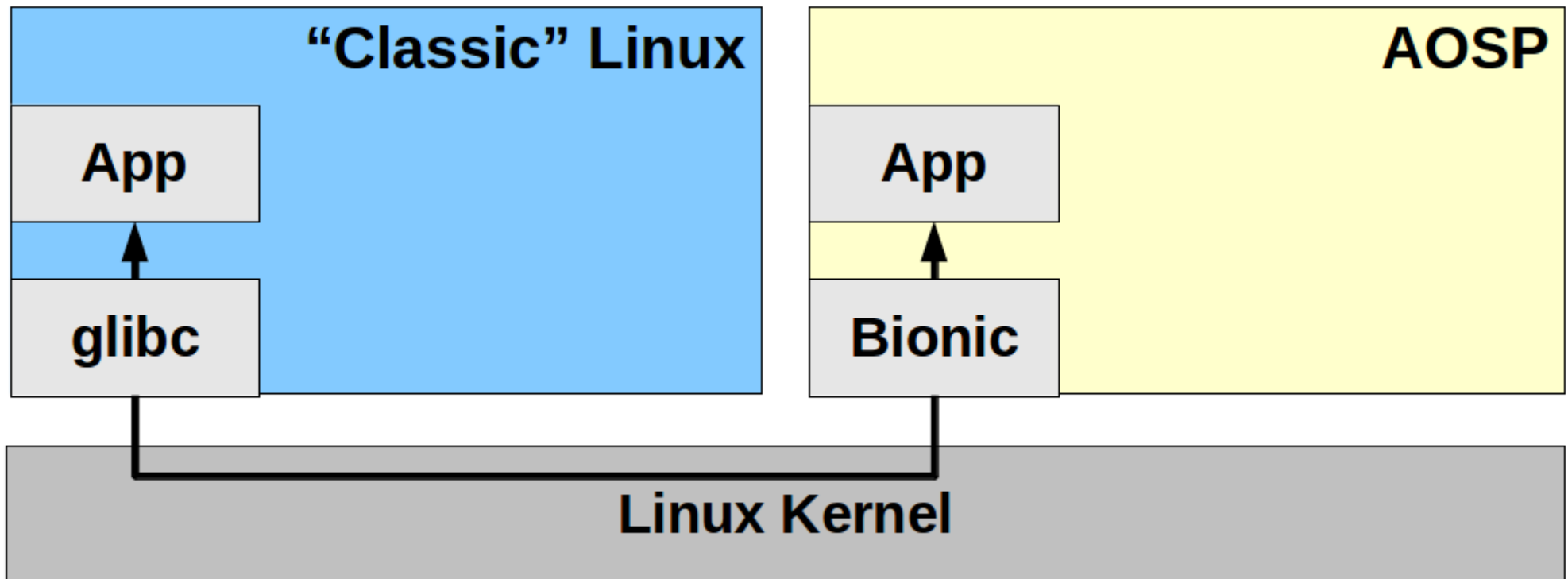
Embedded Linux Root FS

1. Legacy Integration
2. Architecture
3. Tools
4. Embedded Linux Workspace
5. Basic root filesystem structure
6. Libraries
7. Kernel modules
8. Device files
9. Main system applications
10. Auto-generating filesystems
11. Creating a cross-development toolchain

1. Legacy Integration

- What if I ... ?:
 - have been using my custom rootfs forever
 - really need glibc
 - have a huge stack based on “legacy Linux”
 - don't want to deal with AOSP's build system
 - want BusyBox
 - want to unify with a std Linux desktop
 - need things the AOSP doesn't provide
 - have a ton of Linux drivers and want those to be used by Android
 - ...
- Know this: filesystem integration is trivial:
 - Change build system to copy your custom rootfs
 - Change init scripts to start you own daemons
 - Use sockets to communicate
- Know this too: std Linux desktop integration is NOT

2. Architecture



Host

AOSP (bionic)

build/
system/
frameworks/
packages/
external/

PRJROOT (glibc)

sysapps/
rootfs/
build-tools/
.../

Kernel

Target

- ramdisk.img => /
 - /bin
 - /lib
- system.img => /system
- userdata.img => /data

Kernel

3. Tools

- GNU cross-development toolchain:
 - gcc - compiler
 - as - assembler
 - ld - linker
 - gdb/gdbserver - debugger
 - etc.
- C library: uClibc, eglibc or glibc

4. Embedded Linux Workspace

- Need to organize the components used during cross-platform development. Workspace layout:

bootldr:	target bootloader (s)
build-tools:	toolchain build packages and sources
debug:	debugging tools
doc:	project documentation
images:	binary images ready to be used on target
kernel:	sources and build directories for target kernels
project:	your own custom code for the target
rootfs:	root filesystem as seen on the target
sysapps:	sources for target's system applications
tmp:	temporary data and experiments
tools:	toolchain and all other tools required to build software for the target.

4.1. Workspace env. vars. script

- Complete workspace script (devex)

```
export PROJECT=emblinux
export PRJROOT=/home/karim/${PROJECT}
export TARGET=arm-unknown-linux-gnueabi
export PATH=${PATH}:[BBONE_WS_DIR]/bin
cd $PRJROOT
```

- To use this script:

```
$ ._devex
```

- Possible values for \$TARGET:

- ARM: arm-linux, arm-unknown-linux-gnueabi
- MIPS: mips-linux, mipsel-unknown-linux-gnu
- I386: i386-linux, i586-geode-linux-uclibc

5. Basic root filesystem structure

- Unix FS structured for multi-user systems
- Some directories not necessary for embedded
- Filesystem Hierarchy Standard (FHS):
 - /bin => Essential user binaries
 - /boot => Bootloader and kernel images
 - /dev => Device files
 - /etc => System configuration
 - /home => User home directories
 - /lib => Essential shared libs and kernel modules
 - /mnt => Temporary mount point
 - /opt => Add-on software packages
 - /sbin => Essential system binaries
 - /tmp => Temporary files
 - /usr => Secondary hierarchy (mostly user apps)
 - /var => Variable data generated by daemons

- Non-essential multi-user dirs:
 - /home, /mnt, /opt, /root
- Depends on bootloader:
 - /boot
- ***Traditionally*** “essential”:
 - /bin, /dev, /etc, /lib, /proc, /sbin, /usr, /tmp, /var
- Careful with “/etc”, Android needs it to point to “/system/etc” for Dbus config ... Just hack it.
- Contain their own hierarchy:
 - /usr, /var

- What are all these binaries directories for?
 - /bin => Essential binaries for user and admin
 - /sbin => Essential binaries for admin
 - /usr/bin => Non-essential user and admin binaries
 - /usr/sbin=> Non-essential admin binaries
- What are all those libraries directories for?
 - /lib => Essential system libraries
 - /usr/lib => Non-essential libraries
- The kernel does not force FS layout. Layout is “universally” agree upon (i.e. FHS.)

- To start working on rootfs:
`$ cd ${PRJROOT}/rootfs`
- Create core rootfs directories:
`$ mkdir bin lib sbin usr var`
- Create the /usr hierarchy:
`$ mkdir usr/{bin,lib,sbin}`
- Create the /var hierarchy:
`$ mkdir var/{lib,lock,log,run,tmp}`
`$ chmod 1777 var/tmp`

6. Libraries

1.glibc

2.uClibc

6.1. glibc

- glibc components:
 - Actual shared libraries:
 - Format: libLIB_NAME-GLIBC_VER.so
 - Examples: libm-2.3.2.so, libc-2.3.2.so
 - Major revision version symbolic links:
 - Format: libLIB_NAME.so.MAJOR_REV_VER
 - Examples: libdl.so.2, libc.so.6
 - Version-independent symbolic links to the major revision version symbolic links:
 - Format: libLIB_NAME.so
 - Examples: libdl.so, libm.so
 - Static library archives:
 - Format: libLIB_NAME.a
 - Examples: libdl.a, libm.a

- For target, need:
 - The actual shared libs
 - The major revision version symbolic links
- Also need dynamic linker:
 - Actual linker: ld-GLIBC_VER.so
 - Symbolic link to linker:
 - x86, ARM, SH, m68k => ld-linux.so.MAJOR_REV_VER
 - MIPS, PPC => ld.so.MAJOR_REV_VER
- Must determine exact library components required.
- BELS table 6.2 contains complete list

- Most important components:
 - `ld` => the dynamic linker
 - `libc` => the C library
 - `libm` => the math library
 - `libdl` => the shared objects manipulation library
- Must determine exact dependencies of your applications.
- Native `ldd` is not cross-platform-capable
- Can use `readelf` or `uclibc-ldd`:

- Using readelf:

```
$ arm-linux-readelf -a ${PRJROOT}/rootfs/bin/busybox \  
> | grep "Shared library"  
0x00000001 (NEEDED) Shared library: [libc.so.0]
```

- Using uclibc-ldd:

```
$ arm-uclibc-ldd ${PRJROOT}/rootfs/bin/busybox  
libc.so.0 => /home/karim/example-sys/tools/uclibc/lib/libc.so.0  
/lib/ld-uClibc.so.0 => /lib/ld-uClibc.so.0
```

- Copying important libraries to target rootfs:

```
$ cd ${TARGET_PREFIX}/lib  
$ for file in libc libcrypt libdl libm \  
> libpthread libresolv libutil  
> do  
> cp $file-*.so ${PRJROOT}/rootfs/lib  
> cp -d $file.so.[*0-9] ${PRJROOT}/rootfs/lib  
> done  
$ cp -d ld*.so* ${PRJROOT}/rootfs/lib
```

- Copying all libraries (replace “arm” if need be):

```
$ cp -d ${PRJROOT}/tools/arm-unknown-linux-gnueabi/arm-unknown-  
linux-gnueabi/lib/* ${PRJROOT}/rootfs/lib
```

- Stripping all target libraries for space efficiency:

```
$ arm-unknown-linux-gnueabi-strip ${PRJROOT}/rootfs/lib/*.so*
```

6.2. uClibc

- Same naming conventions as glibc
- Implements most of the glibc components:
 - ld, libc, libcrypt, libdl, libm, libpthread, libresolv, libutil.
- uClibc libraries can coexist with glibc libraries in target's /lib directory.
- Copying important libraries to target rootfs:

```
$ cd ${PREFIX}/uClibc/lib
$ for file in libuClibc ld-uClibc libc libdl \
> libcrypt libm libresolv libutil
> do
> cp $file-*.so ${PRJROOT}/rootfs/lib
> cp -d $file.so.[*0-9] ${PRJROOT}/rootfs/lib
> done
```

- Copying all uClibc components:

```
$ cd ${PREFIX}/uClibc/lib
```

```
$ cp *-*.so ${PRJROOT}/rootfs/lib
```

```
$ cp -d *.so.[*0-9] ${PRJROOT}/rootfs/lib
```

- No need to strip uClibc libraries, they are stripped by the uClibc build script.

7. Kernel modules

- Kernel modules are located in `/lib/modules`, so they must be installed in `${PRJROOT}/rootfs/lib/modules`.
- Copying modules built earlier:

```
$ cp -a ${PRJROOT}/images/modules-2.6.37/* \  
> ${PRJROOT}/rootfs
```
- Module loading customization (`/etc/modprobe.conf` or `/etc/modprobe.d/`)

8. Device files

- All devices in Linux are seen as files (except Ethernet interfaces.)
- Typical workstation distros use udev
- Keep a copy of Documentation/devices.txt handy
- See BELS table 6.3 for core set of /dev entries
- Properties of each /dev node:
 - Filename (node name)
 - Type (char / block)
 - Major number (What type of device?)
 - Minor number (Which instance of the device?)
 - Permission bits
- No need to create these entries since AOSP does it for us

9. Main system applications

- Unix systems rely on a common set of commands
- Standard distros have one binary per command
- May compile each relevant command one-by-one or use packages that provide many commands in a single binary:
 1. Busybox
 2. Distro

9.1. BusyBox

- Main package used in embedded Linux to provide core set of Unix commands: busybox.net

[, [[, acpid, add-shell, addgroup, adduser, adjtimex, arp, arping, ash, awk, base64, basename, beep, blkid, blockdev, bootchartd, brctl, bunzip2, bzip2, cal, cat, catv, chat, chatr, chgrp, chmod, chown, chpasswd, chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond, crontab, cryptpw, cttyhack, cut, date, dc, dd, dealloct, delgroup, deluser, depmod, devmem, df, dhcprelay, diff, dirname, dmesg, dnsd, dnsdomainname, dos2unix, du, dumpkmap, dumpleases, echo, ed, egrep, eject, env, envdir, envuidgid, ether-wake, expand, expr, fakeidentd, false, fbset, fbsplash, fdflush, fdformat, fdisk, fgconsole, fgrep, find, findfs, flock, fold, free, freeramdisk, fsck, fsck.minix, fsync, ftpd, ftpget, ftpput, fuser, getopt, getty, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump, hostid, hostname, httpd, hush, hwclock, id, ifconfig, ifdown, ifenslave, ifplugd, ifup, inetd, init, insmod, install, ionice, iostat, ip, ipaddr, ipcalc, ipcrm, ipcs, iplink, iproute, iprule, iptunnel, kbd_mode, kill, killall, killall5, klogd, last, length, less, linux32, linux64, linuxrc, ln, loadfont, loadkmap, logger, login, logname, logread, losetup, lpd, lpq, lpr, ls, lsattr, lsmod, lspci, lsusb, lzcat, lzma, lzop, lzopcat, makedevs, makemime, man, md5sum, mdev, mesg, microcom, mkdir, mkdosfs, mke2fs, mkfifo, mkfs.ext2, mkfs.minix, mkfs.vfat, mknod, mkpasswd, mkswap, mktemp, modinfo, modprobe, more, mount, mountpoint, mpstat, mt, mv, nameif, nbd-client, nc, netstat, nice, nmeter, nohup, nslookup, ntpd, od, openvt, passwd, patch, pgrep, pidof, ping, ping6, pipe_progress, pivot_root, pkill, pmap, popmaildir, poweroff, powertop, printenv, printf, ps, pscan, pwd, raidautorun, rdate, rdev, readahead, readlink, readprofile, realpath, reboot, reformime, remove-shell, renice, reset, resize, rev, rm, rmdir, rmmmod, route, rpm, rpm2cpio, rtcwake, run-parts, runlevel, runsv, runsvdir, rx, script, scriptreplay, sed, sendmail, seq, setarch, setconsole, setfont, setkeycodes, setlogcons, setsid, setuidgid, sh, sha1sum, sha256sum, sha512sum, showkey, slattach, sleep, smemcap, softlimit, sort, split, start-stop-daemon, stat, strings, stty, su, sulogin, sum, sv, svlogd, swapoff, swapon, switch_root, sync, sysctl, syslogd, tac, tail, tar, tcpsvd, tee, telnet, telnetd, test, tftp, tftpd, time, timeout, top, touch, tr, traceroute, traceroute6, true, tty, ttysize, tunctl, udhcpc, udhcpd, udpsvd, umount, uname, unexpand, uniq, unix2dos, unlzma, unlzop, unxz, unzip, uptime, usleep, uudecode, uuencode, vconfig, vi, vlock, volname, wall, watch, watchdog, wc, wget, which, who, whoami, xargs, xz, xzcat, yes, zcat, zcip

- Download BusyBox (1.26.2) to your `${PRJROOT}/sysapps` directory and extract it there.
- Move to the directory for the rest of the setup:

```
$ cd ${PRJROOT}/sysapps/busybox-1.26.2
```

- Configuration of BusyBox's options:

```
$ make menuconfig
```

```
----- BusyBox Configuration -----
Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> selects a feature,
while <N> will exclude a feature. Press <Esc><Esc> to exit, <?> for
Help. Legend: [*] feature is selected [ ] feature is excluded

[*] General Configuration --->
Build Options --->
Installation Options --->
Archival Utilities --->
Coreutils --->
Console Utilities --->
Debian Utilities --->
Editors --->
Finding Utilities --->
Init Utilities --->
Login/Password Management Utilities --->
Miscellaneous Utilities --->
Linux Module Utilities --->
Networking Utilities --->
Process Utilities --->
Another Bourne-like Shell --->
System Logging Utilities --->
Linux System Utilities --->
Debugging Options --->
-----
Load an Alternate Configuration File
Save Configuration to an Alternate File
```

- “Busybox Settings”:

- Cross-compiler prefix:

```
$(TARGET) -
```

- BusyBox installation prefix:

```
$(PRJROOT)/rootfs
```

- Build:

```
$ make
```

- Install:

```
$ make install
```

- Only one binary has been installed: /bin/busybox
- All commands are symbolic links to /bin/busybox
- Determining the command issued done through main's `argv[]` and `argc`.
- Creating arbitrary links doesn't work
- BusyBox can be told to create hard-links
- Full command doc on web and in package
- Customizing the paths for the various shells:

```
# Set path
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
```

10. Auto-generating FSes/distros

- Yocto
- Buildroot
- PTXdist
- OpenWRT
- LTIB
- OpenEmbedded
- Gentoo

11. Creating a cross-dev toolchain

- **crosstool-ng**: successor to crosstool
- Available at:
<http://crosstool-ng.org>
- Downloads, patches, builds, installs, etc.
- Comprises **23** steps
- Menuconfig-based
- Supports uClibc, glibc and eglibc
- Supports ARM, Blackfin, MIPS, PowerPC, SH, ...
- Fairly well maintained

- Must make sure the following are installed on Ubuntu in order to use crosstool-ng:
 - gawk
 - texinfo
 - automake
 - libtool
 - cvs
 - libncurses5-dev
- Use “sudo apt-get install” to get those

- Download and extract to `${PRJROOT}/build-tools`

- Configure crosstool:

```
$ cd crosstool-ng-1.10.0/
```

```
$ ./configure
```

- Build and install crosstool-ng:

```
$ make
```

```
$ make install
```

- Configure crosstool:

```
$ cd ${PRJROOT}/build-tools
```

```
$ ct-ng menuconfig
```

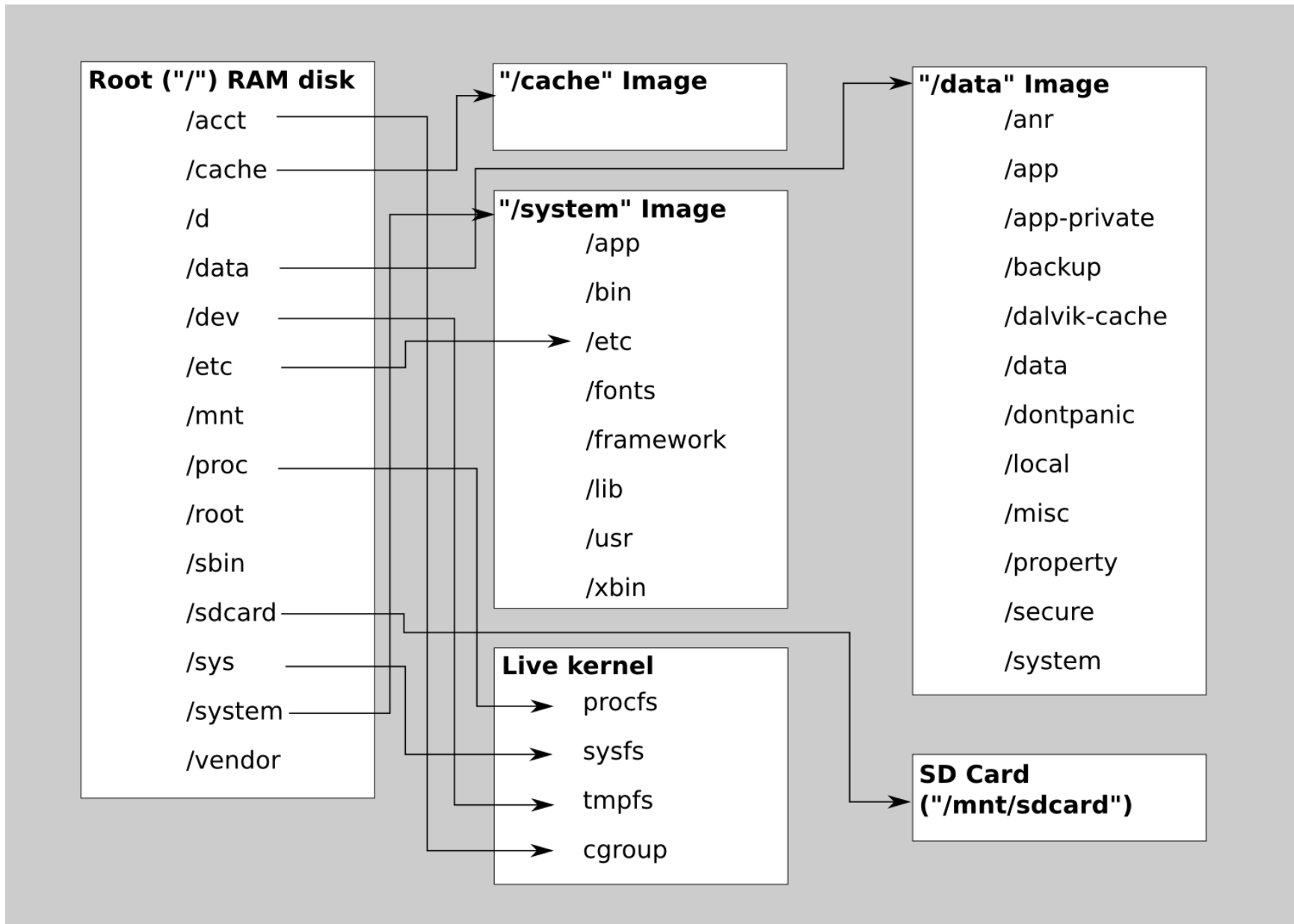
- Options:
 - Paths->Prefix directory: `${PREFIX}/${CT_TARGET}`
 - Target options->architecture: arm
 - OS->Target OS: linux
 - C library->C library: glibc
 - C library->Extra flags: `-U_FORTIFY_SOURCE`
 - Debug facilities: gdb & strace
- Build the toolchain:

```
$ ct-ng build
```

Native Android User-Space

1. Filesystem layout
2. Architecture vs. filesystem
3. Build system and filesystem
4. adb
5. Command line
6. Init
7. ueventd
8. Bionic

1. Filesystem layout



1.1. Root directory

- /acct => Control Group mount point (Documentation/cgroups.txt)
- /cache => cache flash partition
- /d => Symlink to /sys/kernel/debug
- /data => Android's "/data" filesystem
- /dev => Device nodes
- /etc => Symlink to /system/etc
- /mnt => Temporary mount point
- /proc => procfs
- /root => unused
- /sbin => eventd and adbd
- /sdcard => SD card mountpoint
- /sys => sysfs
- /system => Android's "/system" filesystem
- /vendor => Symlink to /system/vendor

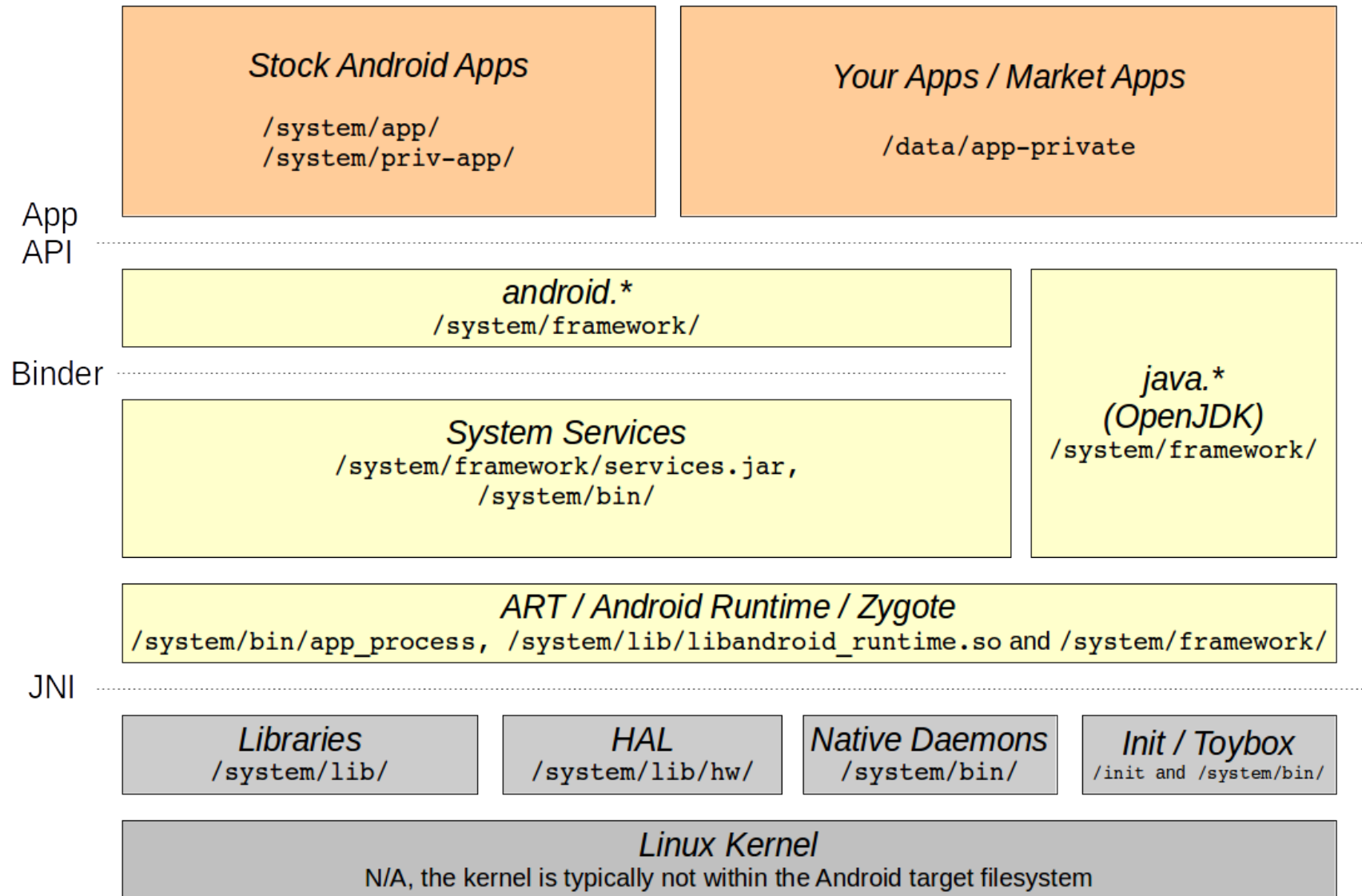
1.1. /system

- /app => Stock apps installed
- /bin => Native binaries and daemons
- /etc => Configuration files
- /fonts => TTFs
- /framework => Android framework .jar files
- /lib => Native libraries
- /usr => Miniature “/usr”
- /xbin => Optional/Extra binaries

1.2. /data

- /anr => ANR traces
- /app => App install location
- /app-private => Protected apps
- /backup => For Backup Manager
- /dalvik-cache => Dalvik DEX cache
- /data => App data
- /dontpanic => Last panic output (console + threads) for “dumpstate”
- /local => Shell-writable space
- /misc => Misc. data (wifi, vpn, bluetooth, ...)
- /property => Persistent system properties (country, lang., ...)
- /secure => Secure Android data available
- /system => System data

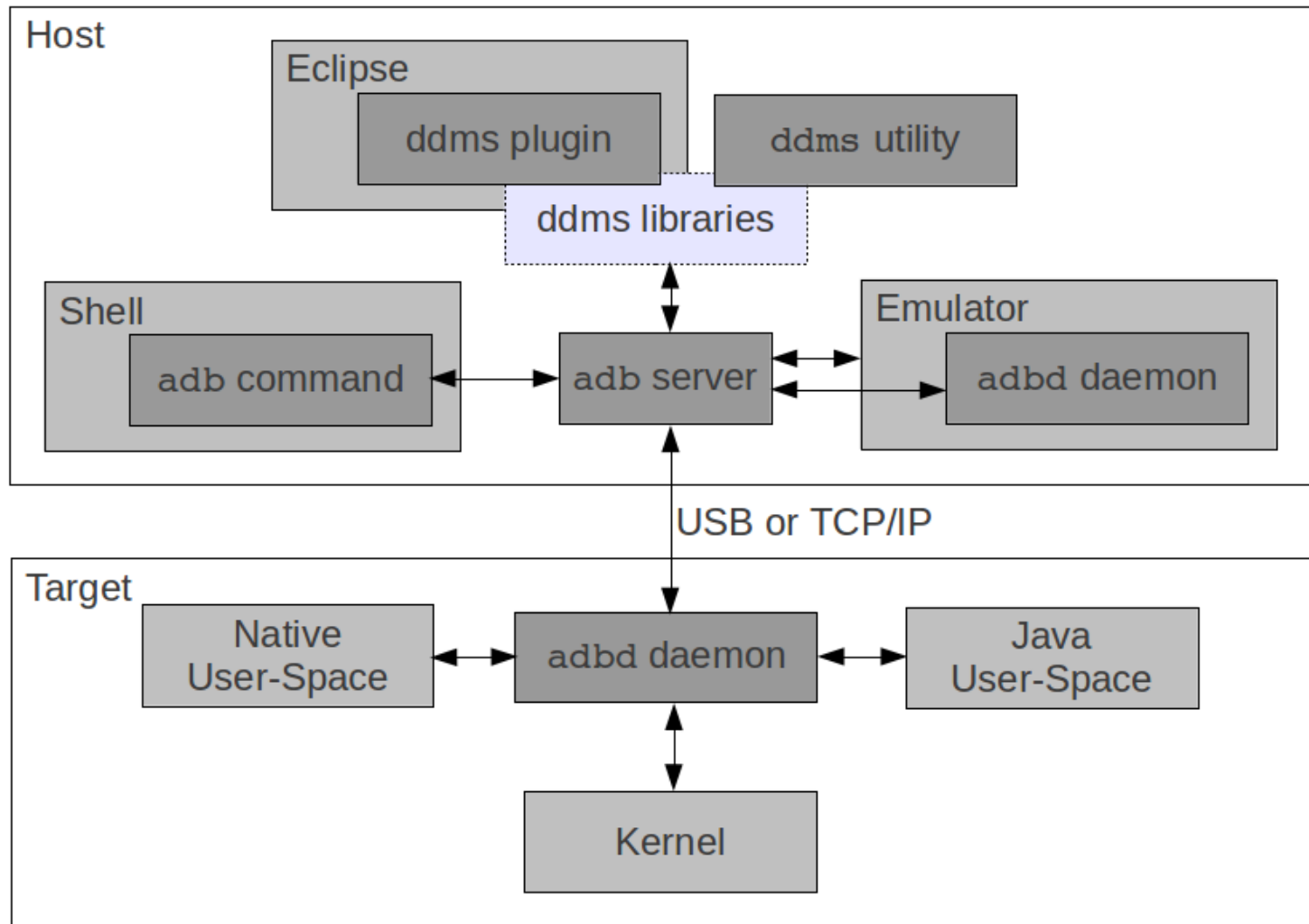
2. Architecture vs. filesystem



3. Build system and filesystem

- Build templates
 - BUILD_EXECUTABLE = /system/bin
 - BUILD_SHARED_LIBRARY = /system/lib
 - BUILD_PACKAGE = /system/app
- Default rights and ownership
 - system/core/include/private/android_filesystem_config.h
 - User IDs / names
 - system/core/libcutils/fs_config.cpp
 - android_dirs struct
 - android_files struct

4. adb



4.1. What adb can do

- See online help for full details
- Managing device connections
- Remote commands:
 - shell
 - log dumping
 - bug reports
 - port forwarding
 - dalvik debugging

- Filesystem commands
 - push
 - pull
 - sync
 - install / uninstall
- State-altering commands
 - reboot
 - run as root
 - switching connection type
 - controlling the emulator
- Tunneling PPP

5. Command line

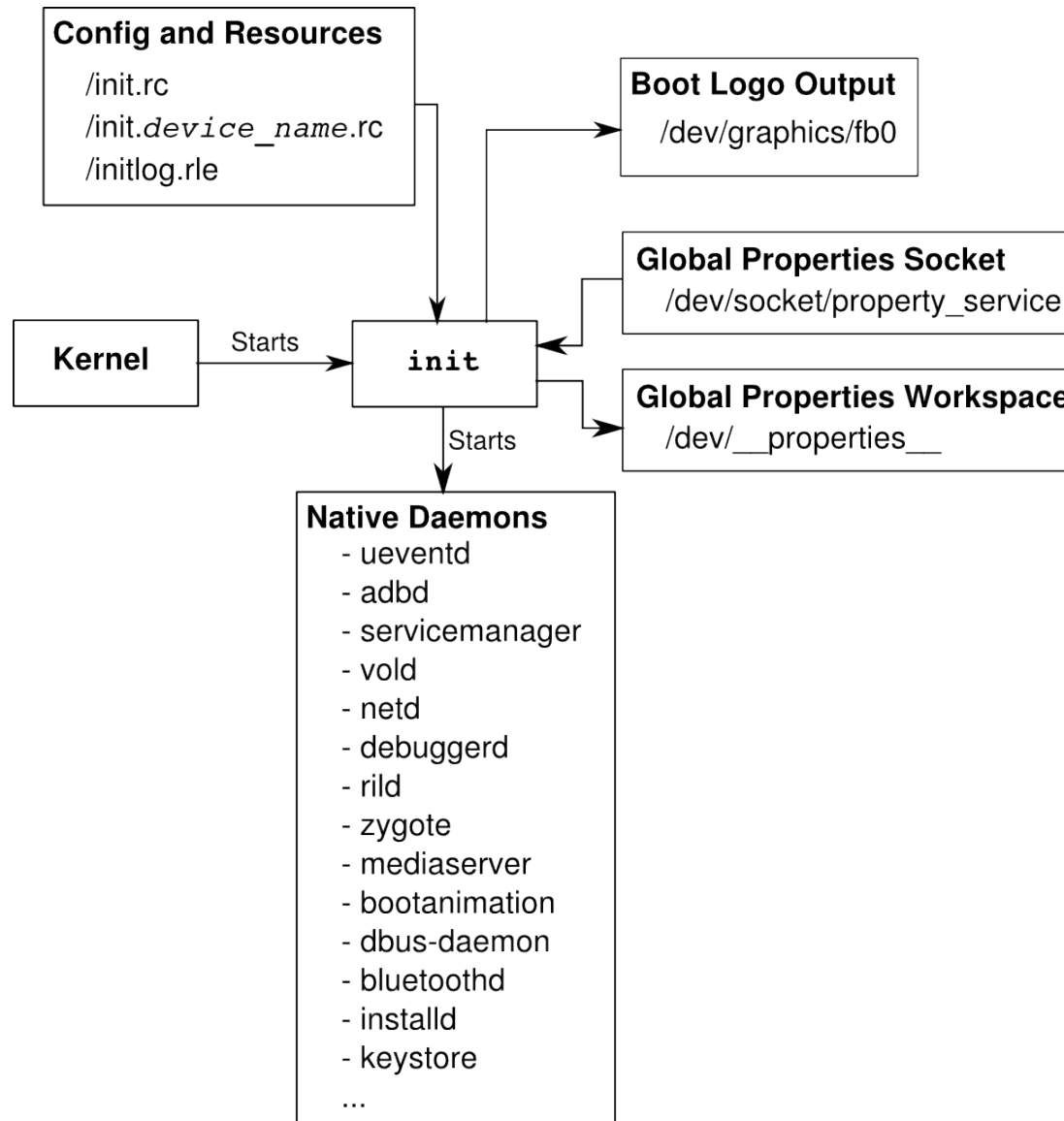
- Shell:
 - NetBSD shell up to 2.3/GB
 - MirBSD Korn shell since 4.0/ICS
- Toolbox:
 - Traditional Linux commands
 - Global properties
 - Input events
 - Services control
 - Logging
 - ioctl
 - Device wiping
 - etc.

- In aosp:
 - /system/core/toolbox
- In filesystem:
 - /system/bin/toolbox
- Provides

alarm date getevent insmod ls mv powerd renice schedtop
 smd top dd getprop ioctl lsmmod nandread printenv rm
 sendevent start umount cat hd ionice lsof netstat ps rmdir
 setconsole stop uptime chmod df id kill mkdir
 newfs_msdos r rmmmod setkey sync vmstat chown dmesg
 ifconfig ln readtty rotatefb setprop syren watchprops cmp
 exists iftop log mount notify reboot route sleep wipe

- logcat
- netcfg
- debuggerd

6. Init



6.1. Config files

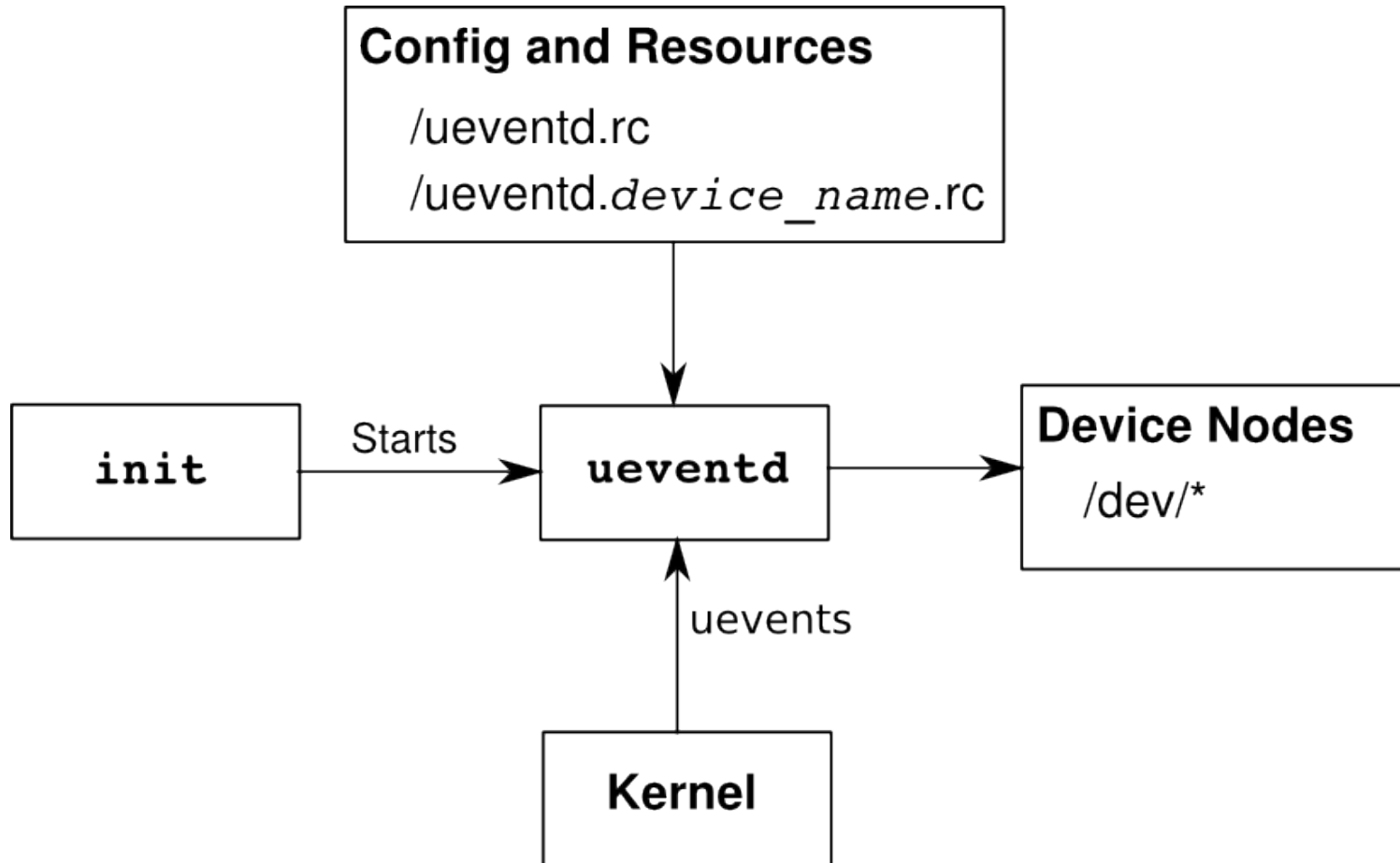
- Location:
 - /init.rc
 - /init.[board].rc
- Semantics:
 - actions
 - Property triggers
 - Commands
 - services (not related to system services or apps)
 - Parameters

6.2. Global properties

- /dev/socket/property_service
- /dev/__properties__
- foo.bar.property
- ro.*
- persist.* => /data/property
- ctl.*
- net.change

- Std files:
 - /system/build.prop
 - /default.prop
- Other files:
 - /system/default.prop
 - /data/local.prop
- Code:
 - `property_get("dalvik.vm.heapsize", heapsizeOptsBuf+4, "16m");`

7. ueventd



7.1. /ueventd.rc

```
/dev/null 0666 root root
```

```
/dev/zero 0666 root root
```

```
/dev/full 0666 root root
```

```
/dev/ptmx 0666 root root
```

```
/dev/tty 0666 root root
```

8. Bionic

- In aosp:
 - /bionic
- In filesystem:
 - /system/lib
- Provides:
 - libc
 - libm
 - libdl
 - libstd++
 - libthread_db
 - linker

System Services and Framework Internals

1. Kickstarting the Framework
2. Utilities and Commands
3. Native Daemons
4. System Services Internals
5. Hardware Abstraction Layer
6. Android Framework

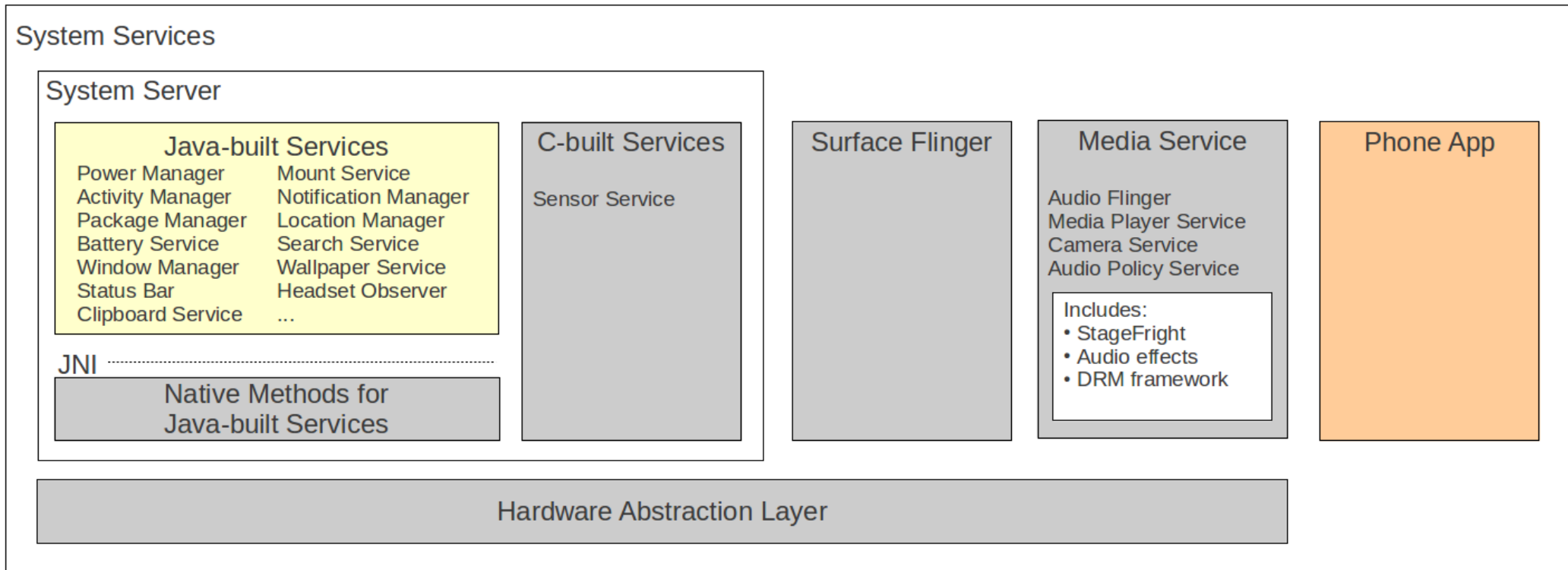
1. Kickstarting the Framework

- Core building blocks
- System services
- Boot animation
- Dex optimization
- Apps startup

1.1. Core building blocks

- Service manager
- Zygote:
 - ART
 - Preloaded classes
 - libandroid_runtime.so

1.2. System services



1.3. Boot animation

- Default boot animation
- bootanimation.zip
- `setprop debug.sf.nobootanimation 1`

1.4. Dex optimization

- Files:
 - .jar
 - .apk
- Directories:
 - BOOTCLASSPATH
 - /system/etc/permission/platform.xml
 - /system/framework
 - /system/app
 - /vendor/app
 - /data/app
 - /data/app-private

1.5. Apps startup

- Input methods
- Persistent apps
- Home screen
- `BOOT_COMPLETED` intent
- `APPWIDGET_UPDATE` intent

2. Utilities and Commands

- General-purpose commands
 - service
 - dumphsys
 - dumpstate
 - rawbu
- Service-specific utilities
 - am
 - pm
 - wm
 - svc

- ime
- input
- monkey
- bmgr
- stagefright
- Dalvik utilities
 - dalvikvm
 - dvz
 - dexdump

3. Native daemons

- servicemanager
- logd
- Imkd
- installd
- vold
- netd
- rild
- keystore

3.1. servicemanager

- In aosp:
 - `/frameworks/base/cmds/servicemanager/`
- In filesystem:
 - `/system/bin/`
- Provides:
 - Context management for binder
 - Service index for entire system

3.2. logd

- In AOSP
 - `/system/core/logd`
- In filesystem:
 - `/system/bin/logd`
- Provides:
 - Logging services
 - Manages multiple buffers

3.3. Imkd

- In AOSP
 - `/system/core/Imkd`
- In filesystem:
 - `/system/bin/Imkd`
- Provides:
 - Low-memory kernel handler

3.4. installd

- In aosp:
 - `/frameworks/base/cmds/installd`
- In filesystem:
 - `/system/bin/`
- Provides:
 - Package install/uninstall
 - Sanity checks and verifications
 - Interfaces with Package Manager service

3.5. vold

- In aosp:
 - /system/vold/
- In filesystem:
 - /system/bin/
- Provides:
 - Volume mounter
 - Auto-mount
 - Auto-format mounted devices

3.6. netd

- In aosp:
 - `/system/netd/`
- In filesystem:
 - `/system/bin/`
- Provides:
 - Management of aspects of networking
 - Interfaces with Network Management service

3.7. rild

- In aosp:
 - /hardware/ril/mock-ril/
- In filesystem:
 - /system/bin/
- Provides:
 - “Radio Interface Layer” to phone hardware

3.8. keystore

- In aosp:
 - frameworks/base/cmds/keystore/
- In filesystem:
 - /system/bin
- Provides:
 - Store key-value pairs
 - Security keys for connecting to network infrastructure

4. System Services Internals

- Services run by System Server
- Observing the System Server
- Calling on system services
- Inside a few system services
- Creating your own system service

4.1. Services run by the System Server

Entropy Service
Power Manager
Activity Manager
Telephone Registry
Package Manager
Account Manager
Content Manager
System Content Providers
Battery Service
Lights Service
Vibrator Service
Alarm Manager
Init Watchdog
Sensor Service
Window Manager
Bluetooth Service

Device Policy
Status Bar
Clipboard Service
Input Method Service
NetStat Service
NetworkManagement Service
Connectivity Service
Throttle Service
Accessibility Manager
Mount Service
Notification Manager
Device Storage Monitor
Location Manager
Search Service
DropBox Service
Wallpaper Service

Audio Service
Headset Observer
Dock Observer
UI Mode Manager Service
Backup Service
AppWidget Service
Recognition Service
Status Bar Icons
DiskStats Service
ADB Settings Observer

4.1.1. Some stats

- frameworks/base/services/core/java/com/android/server:
 - 15 M
 - ~550 files
 - 326 kloc
- Activity manager:
 - 2.6M
 - 60+ files
 - 58 kloc

4.2. Observing the System Server

- Logcat
- dumpsys

4.2.1. logcat

- Find the System Server's PID

```
$ adb shell ps | grep system_server
system 63 32 120160 35408 ffffffff afd0c738 S system_server
```

- Look for its output:

```
$ adb logcat | grep "63)"
```

```
...
D/PowerManagerService( 63): bootCompleted
I/TelephonyRegistry( 63): notifyServiceState: 0 home Android Android 310260 UMTS CSS not supp...
I/TelephonyRegistry( 63): notifyDataConnection: state=0 isDataConnectivityPossible=false reason=null
interfaceName=null networkType=3
I/SearchManagerService( 63): Building list of searchable activities
I/WifiService( 63): WifiService trying to setNumAllowed to 11 with persist set to true
I/ActivityManager( 63): Config changed: { scale=1.0 imsi=310/260 loc=en_US touch=3 keys=2/1/2 nav=3/1 ...
I/TelephonyRegistry( 63): notifyMessageWaitingChanged: false
I/TelephonyRegistry( 63): notifyCallForwardingChanged: false
I/TelephonyRegistry( 63): notifyDataConnection: state=1 isDataConnectivityPossible=true reason=simL...
I/TelephonyRegistry( 63): notifyDataConnection: state=2 isDataConnectivityPossible=true reason=simL...
D/Tethering( 63): MasterInitialState.processMessage what=3
I/ActivityManager( 63): Start proc android.process.media for broadcast
com.android.providers.downloads/.DownloadReceiver: pid=223 uid=10002 gids={1015, 2001, 3003}
I/RecoverySystem( 63): No recovery log file
W/WindowManager( 63): App freeze timeout expired.
...
```

4.2.2. dumphys

Currently running services:

SurfaceFlinger

accessibility

account

activity

alarm

appwidget

audio

backup

...

wifi

window

DUMP OF SERVICE SurfaceFlinger:

+ Layer 0x396b90

z= 21000, pos=(0, 0), size=(480, 800), needsBlending=1, needsDithering=1, invalidat ...

0]

name=com.android.launcher/com.android.launcher2.Launcher

client=0x391e48, identity=6

[head= 1, available= 2, queued= 0] reallocMask=00000000, inUse=-1, identity=6, status=0

format= 1, [480x800:480] [480x800:480], freezeLock=0x0, dq-q-time=53756 us

...

4.3. Calling on System Services

- Use `getSystemService`
- Ex: NotificationManager Object reference:

```
String ns = Context.NOTIFICATION_SERVICE;
```

```
NotificationManager mNotificationManager = (NotificationManager) \n    getSystemService(ns);
```

- Prepare your content
- Call on the object:

```
mNotificationManager.notify(HELLO_ID, notification);
```

4.4. Inside a few System Services

- Get the AOSP ... repo, etc.
- Tricks:
 - xref.opersys.com
 - Import into Android Studio
 - Use reverse-engineering tools:
 - Imagix
 - Rationale
 - Lattix
 - Scitools
 - ...
- Be patient, this isn't documented anywhere ...

4.4.1. ActivityManager

- Start new Activities, Services
- Fetch Content Providers
- Intent broadcasting
- OOM adj. maintenance
- Application Not Responding
- Permissions
- Task management
- Lifecycle management

- Ex. starting new app from Launcher:
 - onClick(Launcher)
 - startActivity(Activity.java)
 - *<Binder>*
 - ActivityManagerService
 - startViaZygote(Process.java)
 - *<Socket>*
 - Zygote

4.4.2. Package Manager

- 25 kloc
- 1.2M
- Installation / removal
- Permissions
- Intent resolution (also IntentResolver.java)
- Called by Activity Manager

4.4.3. Window Manager

- Main thread
- Window manipulation
- Wallpaper handling
- Orientation
- Focus
- Layering
- Input event management

4.4.4. Notification Manager

- Toasts
- Notifications
- Sound playback (see NotificationPlayer.java)

4.4.5. Power Manager

- Wakelocks
- Sleep
- Brightness
- Lock

4.4.6. Network Management Service

- Talks to “netd” /system/netd
- Interface configuration
- Tethering
- DNS

4.4.7. Mount Service

- Mount / Unmount
- Format
- USB mass storage
- OBB

4.4.8. Location Manager

- Manage location providers
- `getBestProvider()`
- Proximity alerts
- Last known location

4.4.9. Status Bar Manager

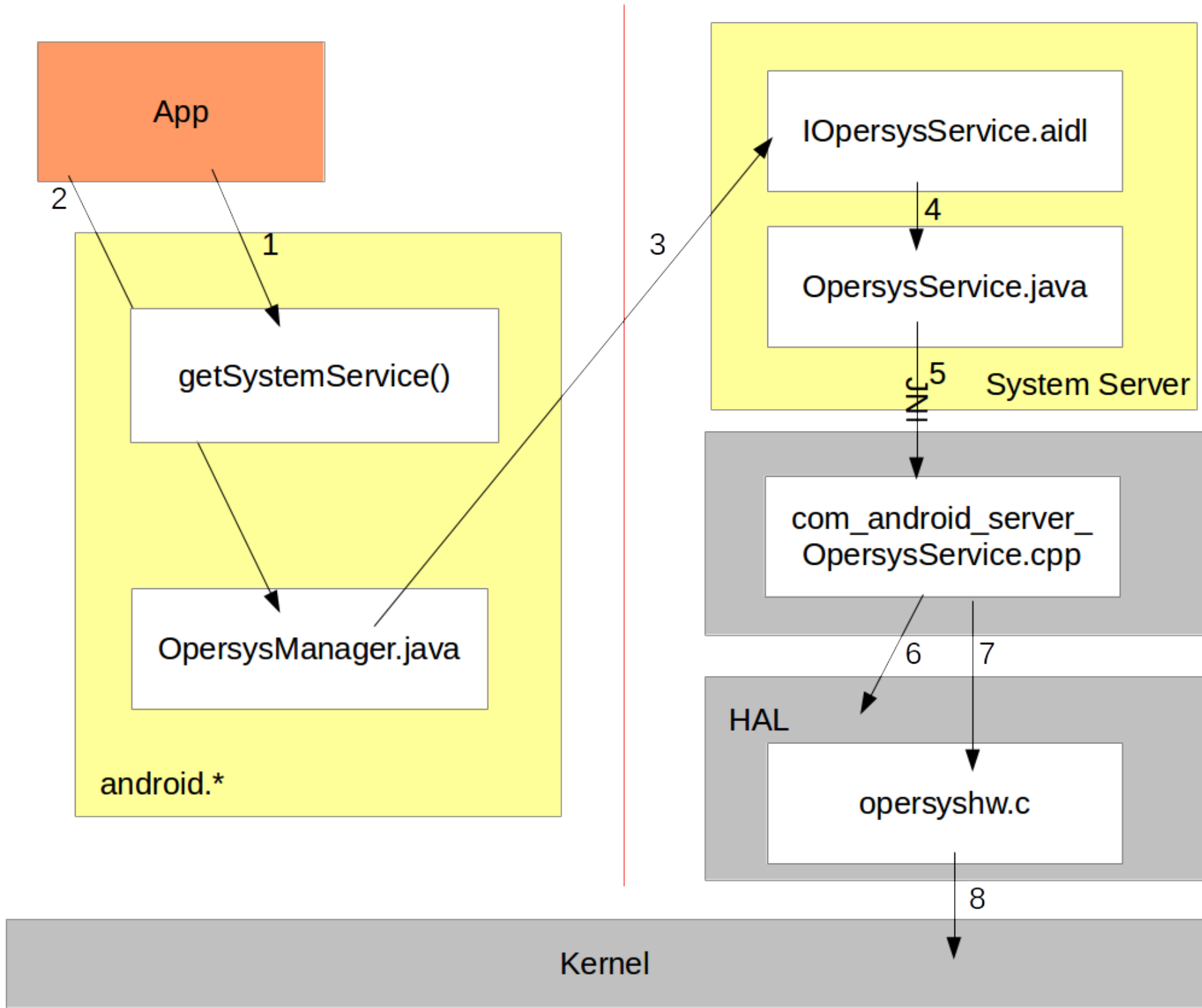
- Expand / collapse
- Icon visibility
- Reveal callbacks
- Callbacks for notification manager

4.4.10. Backup Manager

- Enable / disable
- Transport management
- backupNow()
- ...

4.5. Creating your own System Service

- Add your code to:
frameworks/base/services/core/java/com/android/server/
- Have the SystemServer.java init+reg. your service
- Define hardware API for apps
- Expose through:
 - frameworks/base/core/java/android/os/[server].aidl
- Call on native “driver” code through JNI
- Implement or connect to appropriate driver
- Create an app that calls on service
- May need to create new SDK ...



4.5.1. OpersysService.java

```
package com.android.server;

import android.content.Context;
import android.os.IOpersysService;
import android.util.Log;

public class OpersysService extends IOpersysService.Stub {
    private static final String TAG = "OpersysService";
    private Context mContext;
    private int mValue = 0;

    public OpersysService(Context context) {
        super();
        mContext = context;
        Log.i(TAG, "System server started");
    }

    public void test(int val) {
        Log.i(TAG, "test " + val);
        MValue = val;
    }
}
```

4.5.2. IOpersysService.aidl

```
package android.os;
/**
 * {@hide}
 */
interface IOpersysService {
String read(int maxLength);
int write(String mString);
void test(int val);
}
```

4.5.3. frameworks/base/Android.mk

...

```
core/java/android/os/IPowerManager.aidl \  
core/java/android/os/IOperSystemService.aidl \  
core/java/android/os/IRemoteCallback.aidl \  
...
```

...

4.5.4. SystemServer.java

Should eventually be Context.OPERSYS_SERVICE

...

```
try {  
    Slog.i(TAG, "Opersys Service");  
    ServiceManager.addService("opersys", new OpersysService(context));  
} catch (Throwable e) {  
    Slog.e(TAG, "Failure starting OpersysService Service", e);  
}
```

...

4.5.5. HelloServer.java

```
package com.opersys.helloserver;

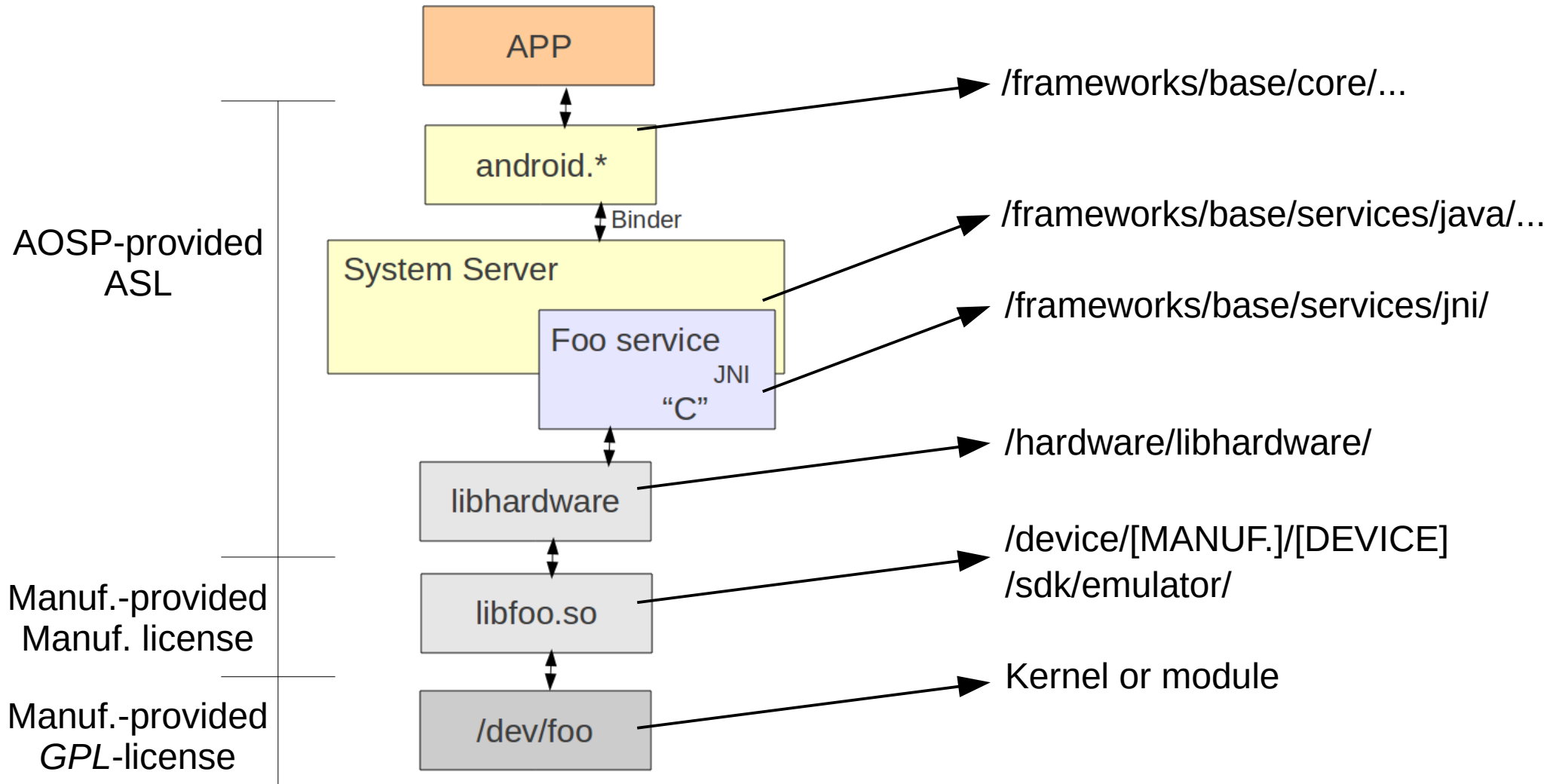
import android.app.Activity;
import android.os.Bundle;
import android.os.ServiceManager;
import android.os.IOpersysService;
import android.util.Log;

public class HelloServer extends Activity {
    private static final String DTAG = "HelloServer";

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        IOpersysService om =
IOpersysService.Stub.asInterface(ServiceManager.getService("opersys"));
        try {
            Log.d(DTAG, "Going to call service");
            om.test(20);
            Log.d(DTAG, "Service called succesfully");
        }
        catch (Exception e) {
            Log.d(DTAG, "FAILED to call service");
            e.printStackTrace();
        }
    }
}
```

5. Hardware Abstraction Layer



- [aosp]/hardware/libhardware/include/hardware
 - gps.h
 - lights.h
 - sensors.h
- [aosp]/hardware/ril/include/telephony/
 - ril.h
- Examples in [aosp]/device/*/*
- Using JNI to call C functions

6. Android Framework

- Location and components
- android.*
- Customization

6.1. Location and components

- [aosp]/frameworks/base
 - /cmds => native cmds and daemons
 - /core => android.* and com.android.*
 - /data => Fonts and sounds
 - /graphics => 2D & Renderscript
 - /include => “C” includes
 - /keystore => security key store
 - /libs => “C” libraries
 - /location=> Location provider
 - /media => Stagefright, codecs, etc.
 - /native => Native code for some frameworks components
 - /obex => Bluetooth obex
 - /opengl => GL library and java code
 - /packages => A few core packages (Status Bar)
 - /services => System server
 - /telephony => Phone related functionality
 - /tools => A few core tools (aapt, aidl, ...)
 - /voip => RTP & SIP interfaces
 - /vpn => VPN functionality
 - /wifi => Wifi manager, monitor, etc.

6.2. android.*

accessibilityservice	content	hardware	pim	speech
accounts	database	inputmethodservice	preference	test
annotation	dcm	net	provider	text
app	debug	nfc	security	util
appwidget	emoji	os	server	view
bluetooth	gesture	service	webkit	widget

6.3. Extending API – System service

- frameworks/base/core/java/android/
 - app/ContextImpl.java
 - content/Context.java
 - os/OpersysManager.java

6.3.1. - app/ContextImpl.java

```
...
import android.os.IOpersysService;
import android.os.OpersysManager;
...
    registerService(OPERSYS_SERVICE, new ServiceFetcher() {
        public Object createService(ContextImpl ctx) {
            IBinder b =
                ServiceManager.getService(OPERSYS_SERVICE);
            IOpersysService service =
                IOpersysService.Stub.asInterface(b);
            return new OpersysManager(service);
        }
    });
...

```

6.3.2. content/Context.java

```
...  
/**  
 * Use with {@link #getSystemService} to retrieve a  
 * {@link android.nfc.NfcManager} for using NFC.  
 *  
 * @see #getSystemService  
 */  
public static final String NFC_SERVICE = "nfc";  
  
/** The Opersys service */  
public static final String OPERSYS_SERVICE = "opersys";  
...
```

6.3.3. os/OpersysManager.java

```
package android.os;

import android.os.IOpersysService;

public class OpersysManager
{
    public void test(int value)
    {
        try {
            mService.test(value);
        } catch (RemoteException e) {
        }
    }

    public String read(int maxLength) {...}

    public int write(String stringVal) {...}

    public OpersysManager(IOpersysService service)
    {
        mService = service;
    }

    IOpersysService mService;
}
```

Project Treble

1. Distribution scheme before 8.x
2. Motivations
3. Enforcement
4. Traditional Android Release Flow
5. Treble's Intended Release Flow
6. Underneath the Framework
7. Linux Kernel
8. HIDL
9. VNDK
10. VINTF
11. VTS
12. Implications
13. Recommendations

1. Distribution scheme before 8.x

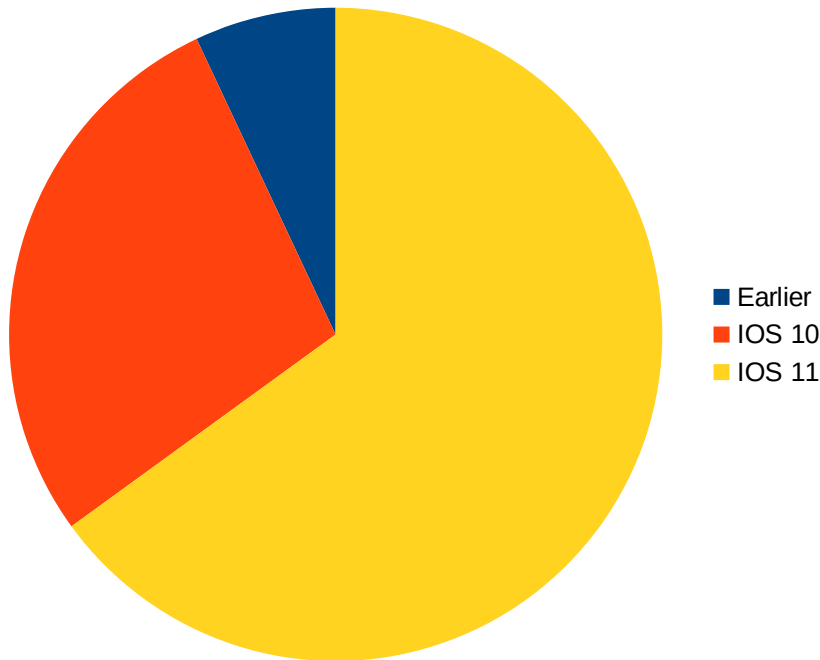
- Google releases new version of Android
- Ecosystem customizes it to new devices – sometimes quite extensively
- Google enforces Compatibility Test Suite (CTS) – guaranteeing developer API conformance
- Devices get certified and ship with GMS
-
- Devices rarely got upgrades

2. Motivations

- Updates: Android vs. iOS
- Android version propagation history
- “Official” Google Treble pitch
- Nature of Android

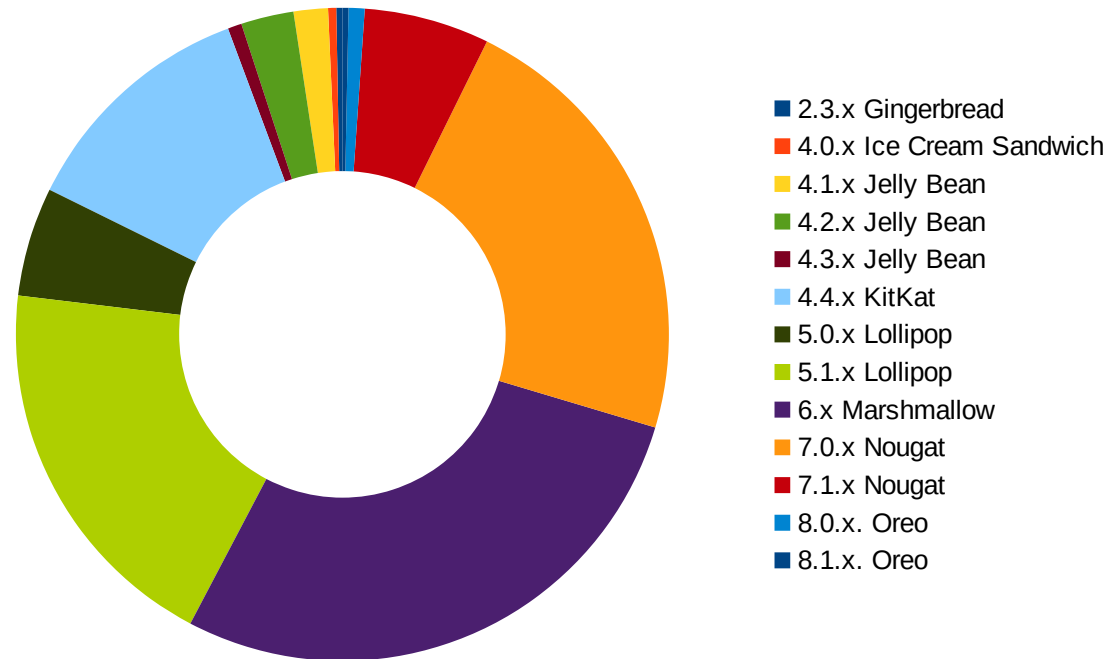
2.1. Updates: Android vs. iOS

iOS



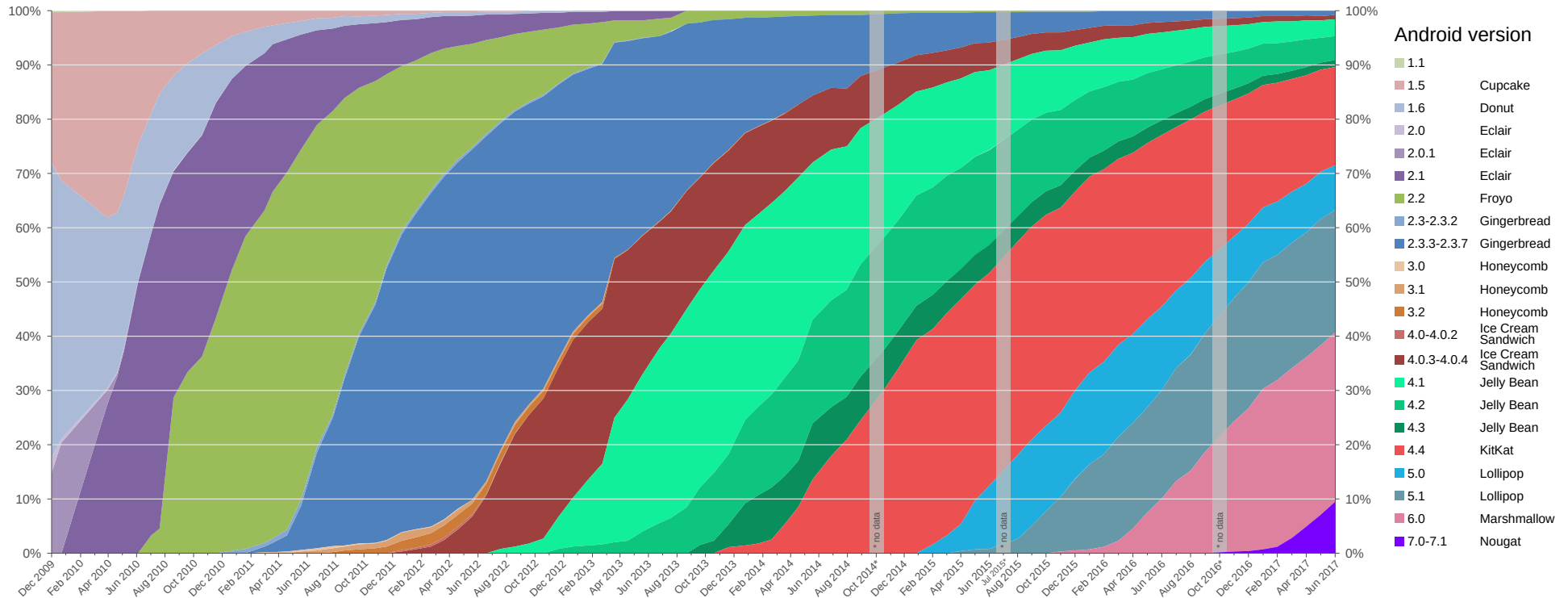
Data from Apple as of January 18, 2018
<https://developer.apple.com/support/app-store/>

Android



Data from Google as of February 5, 2018
<https://developer.android.com/about/dashboards/>

2.2. Version propagation history



- Implications:
 - Android comes out with cool new features
 - Perception is iOS gets them first
- Disclaimer: This is my reading

2.3. “Official” Google Treble pitch

- From source.android.com:

“The Android 8.0 release includes Project Treble, a major re-architect of the Android OS framework designed to make it easier, faster, and less costly for manufacturers to update devices to a new version of Android.”
- In short:
 - Recognition that updates are a problem
 - Identification of update issues:
 - Difficult
 - Long
 - Costly

2.4. Nature of Android

- Anyone is free to download Android
- Google enforces certification for GMS
- Google has no ability to force upgrades
- Google can only incentivize

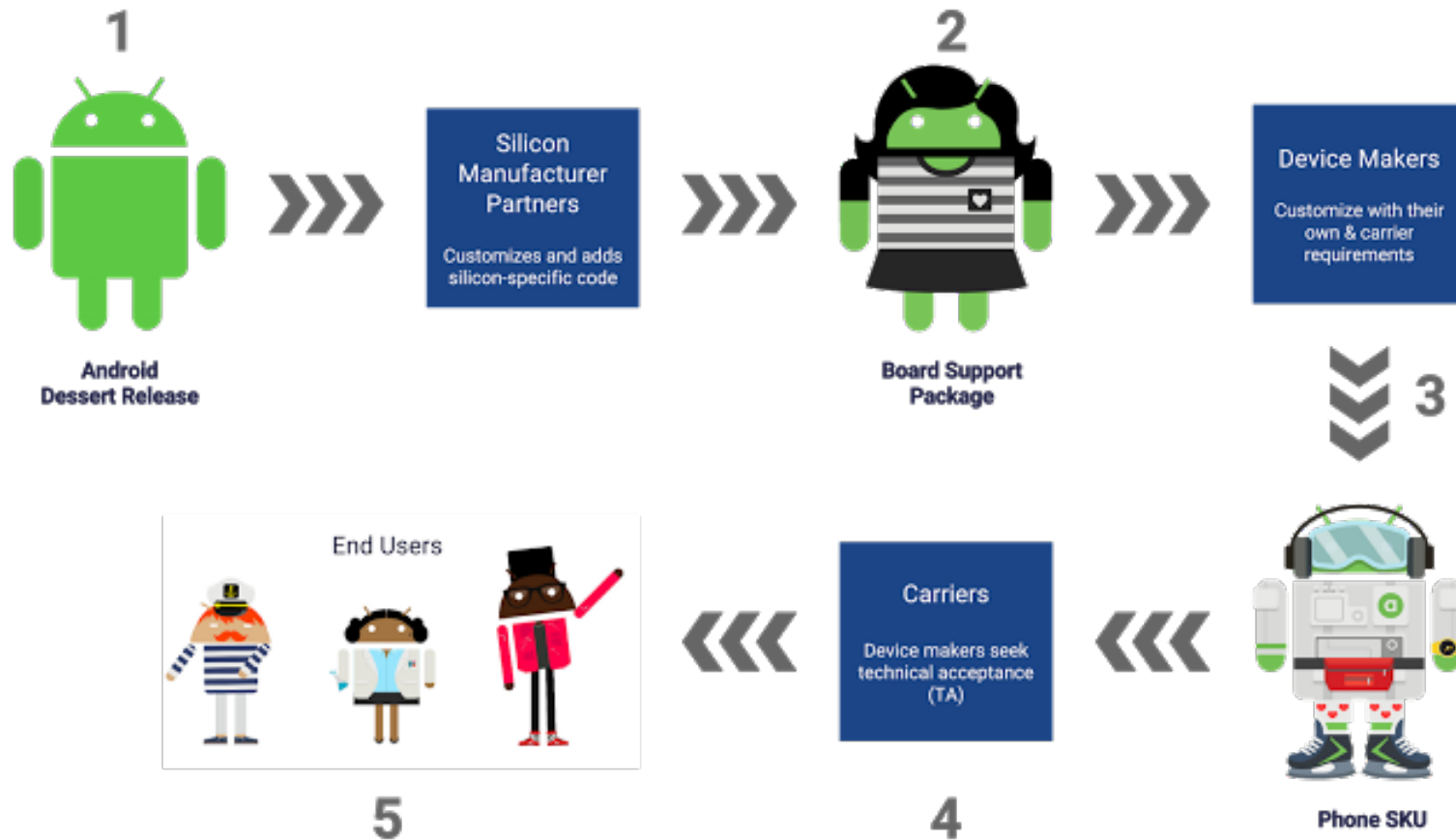
3. Enforcement

- “To ensure forward compatibility of vendor implementations, the new vendor interface is validated by the Vendor Test Suite (VTS), which is analogous to the Compatibility Test Suite (CTS).”
- In sum:
 - Path to upgrade has been charted
 - But ... Comply or we won't certify
- Not much choice – Android is the “only game in town” aside from Apple's iOS.
- Worthy end-goal:
 - 4 to 5 years of upgrades on a single device

4. Traditional Android Release Flow

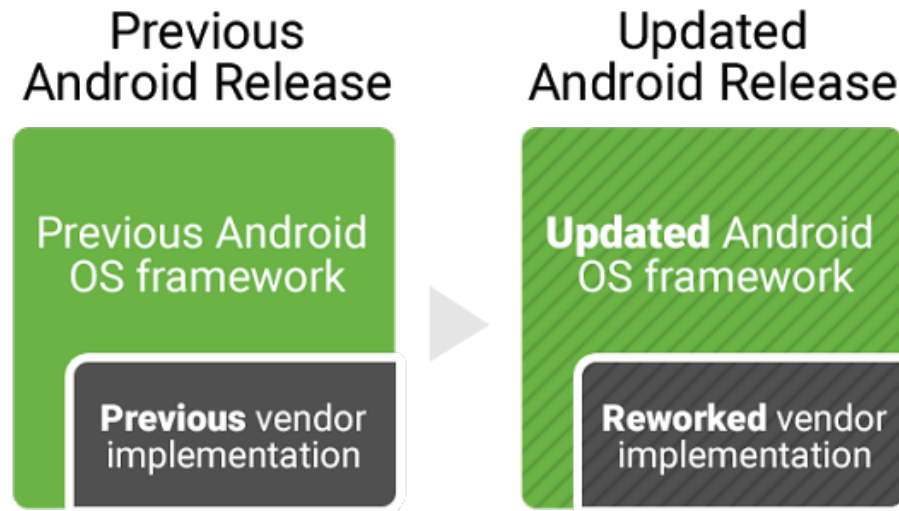
- Release flow
- Simplified internals view
- Example phone circa early 2014
- Upgrade after 5.x/Lollipop release
- Upgrade after 6.x/Marshmallow release

4.1. Release Flow



4.2. Simplified Internals View

Before Treble

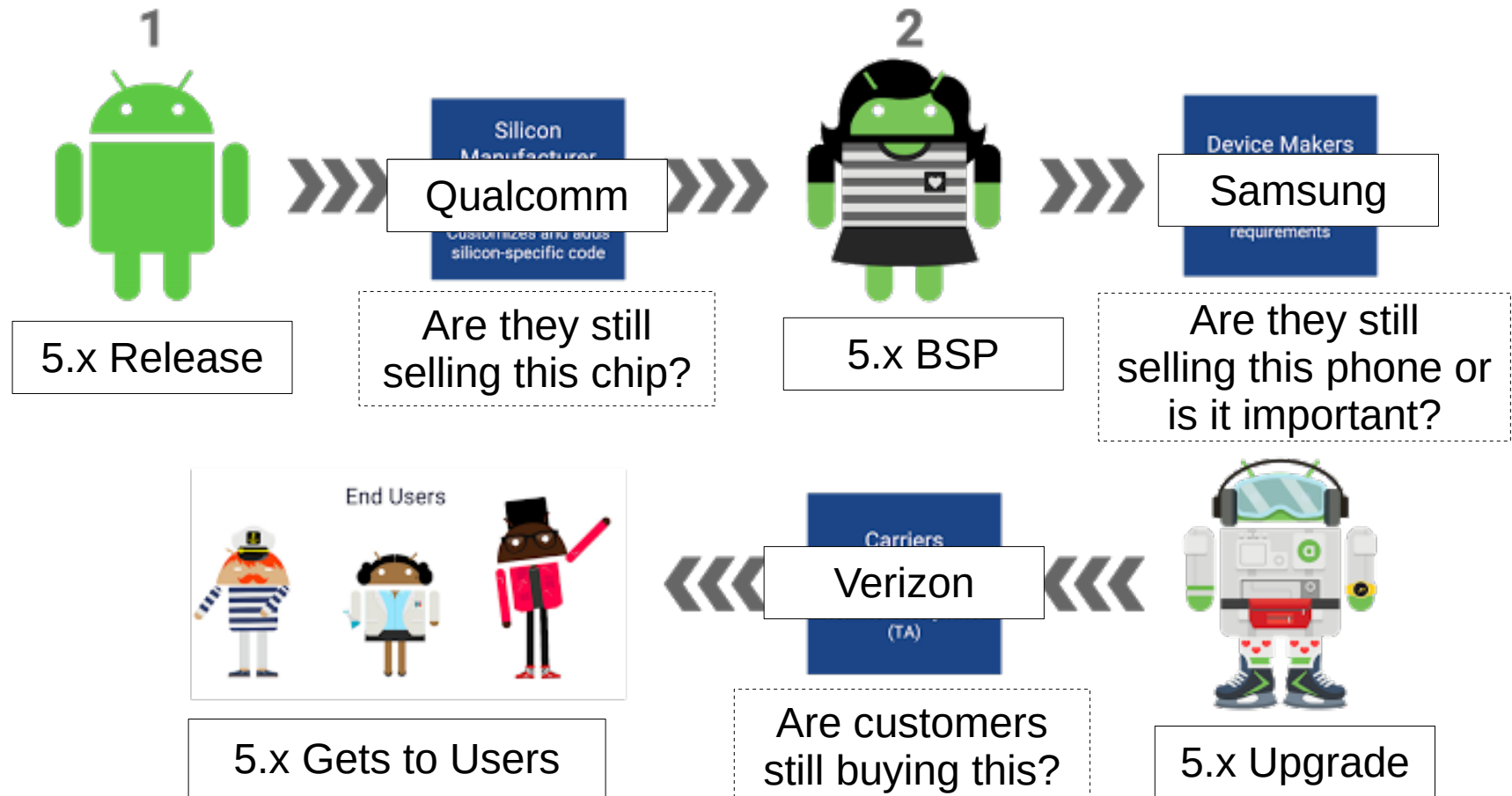


4.3. Example Phone Circa Early 2014

- Just for the sake of example ...
- Say you bought a phone in early 2014:
 - Samsung device
 - Qualcomm chipset
 - Verizon network
- Runs Android 4.4/KitKat at time of purchase

Note: brands/companies used for illustration purposes only, no specific device is implied.

4.4. Upgrade after 5.x/Lollipop release



November 2014

4.5. Upgrade to 6.x/Marshmallow?

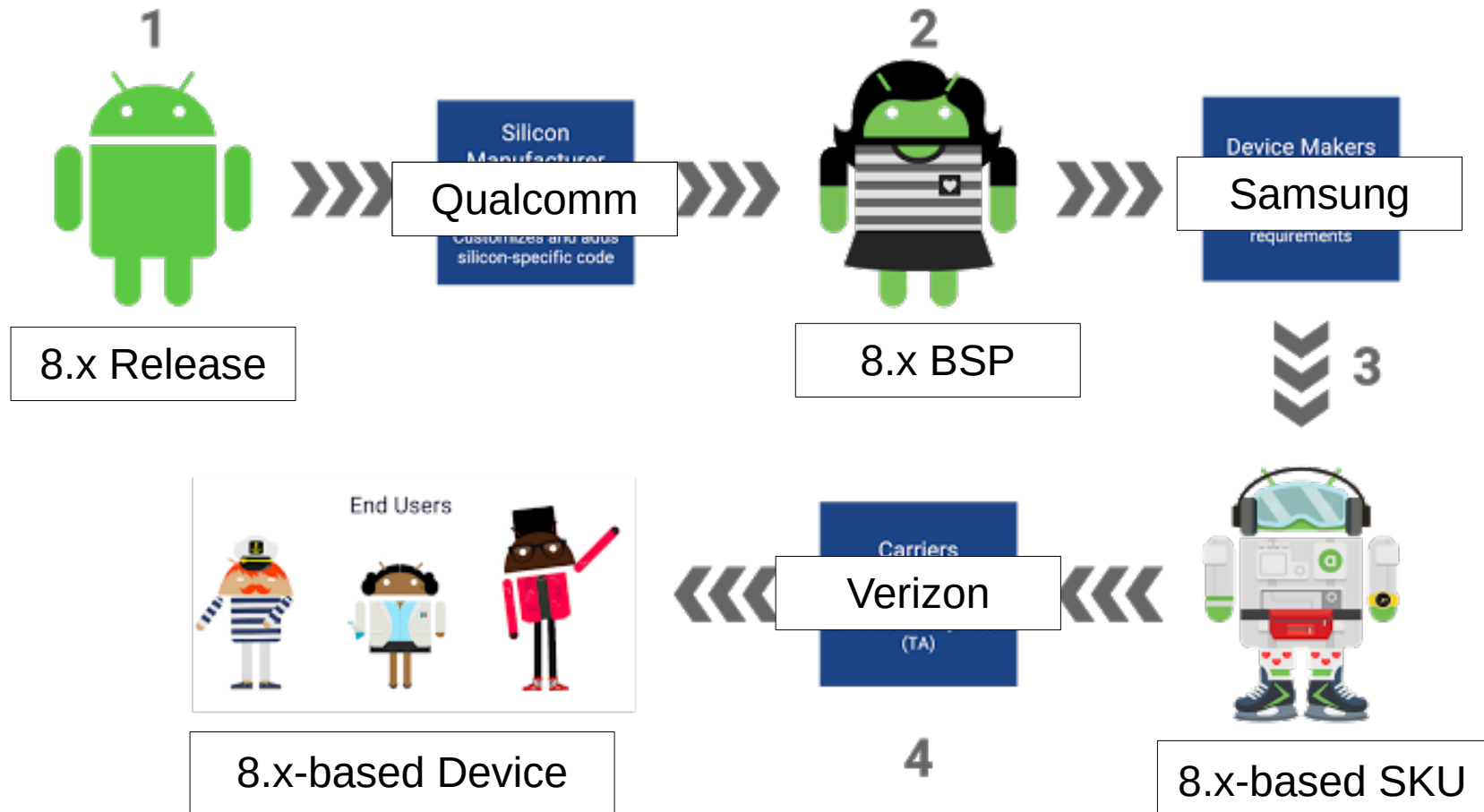


October 2015

5. Treble's Intended Release Flow

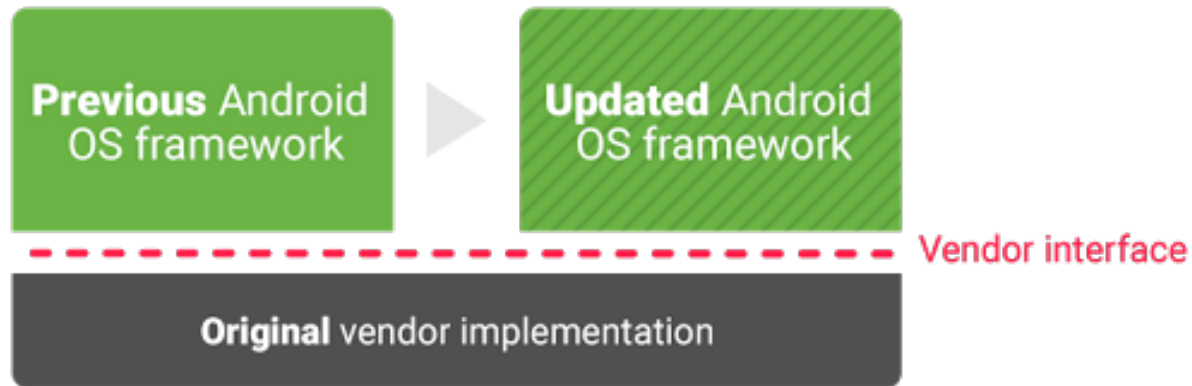
- Initial release flow
- New internals paradigm
- Overall reworked internals
- Targeted upgrade flow
- Theory of future release cycles
- Requirement

5.1. Initial Release Flow

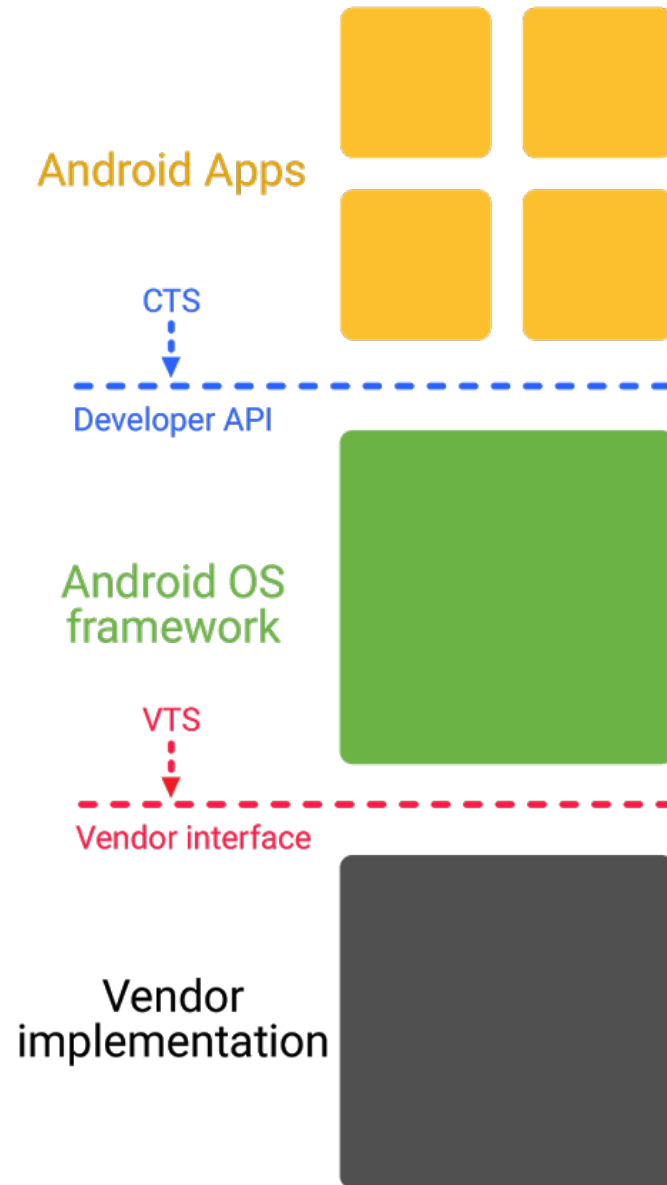


5.2. New Internals Paradigm

With Treble



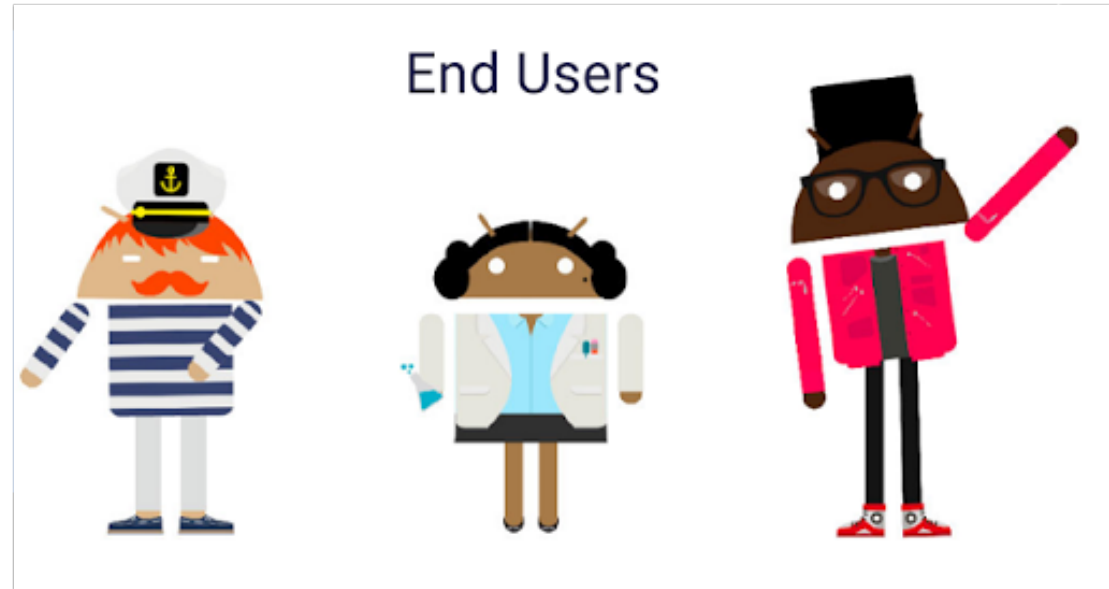
5.3. Overall Reworked Internals



5.4. Targeted Upgrade Flow



**Android
Dessert Release**



5.5. Theory of Future Release Cycles



■ ■ ■

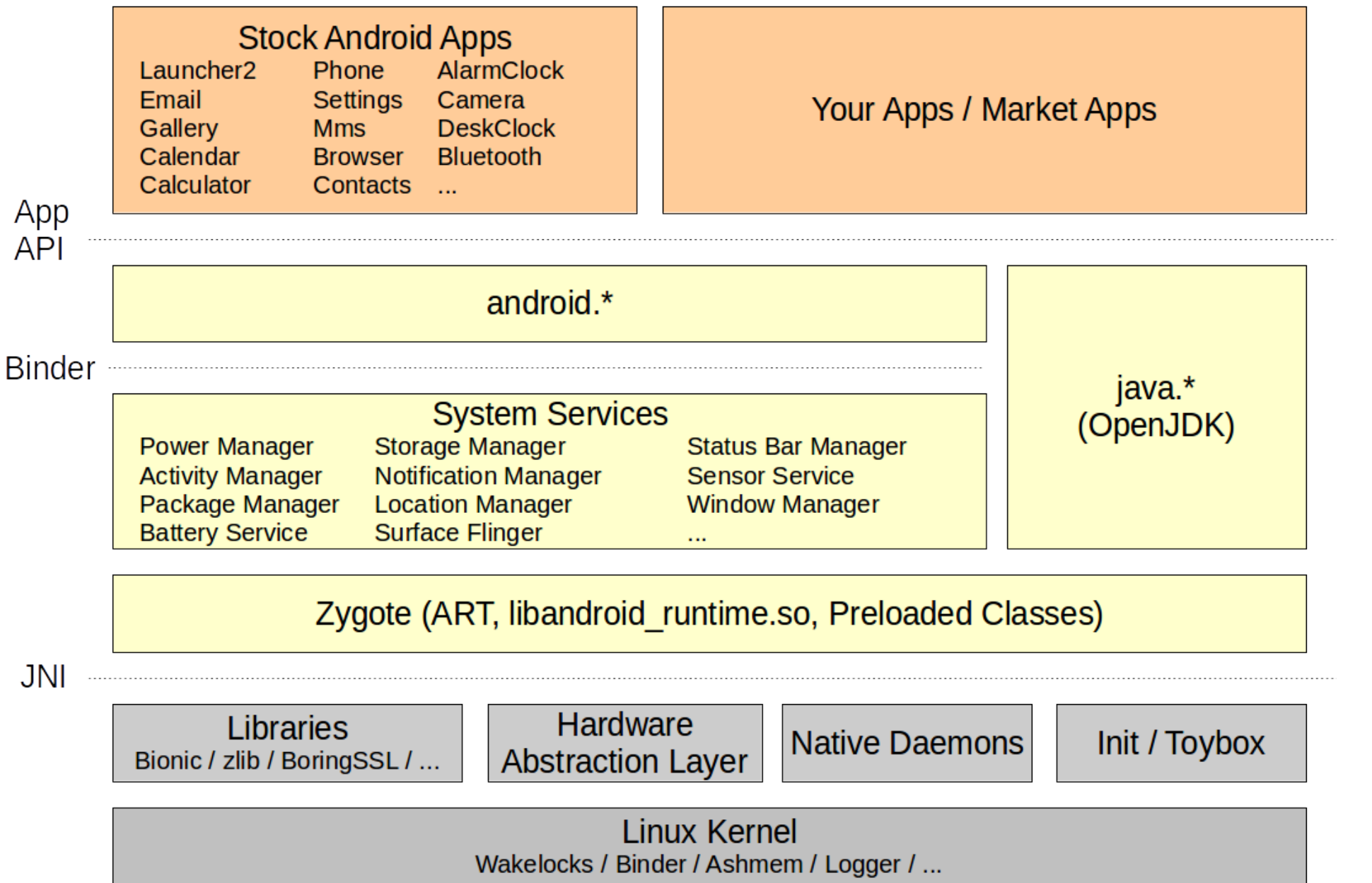
5.6. Requirement

- Google must continue to support old vendor interfaces.
- For an initial 8.x device, future framework release must support 8.x hardware enablement:
 - 9.x / P
 - 10.x / Q
 - 11.x / R
 - ...
- Aim seems to be 4-5 releases

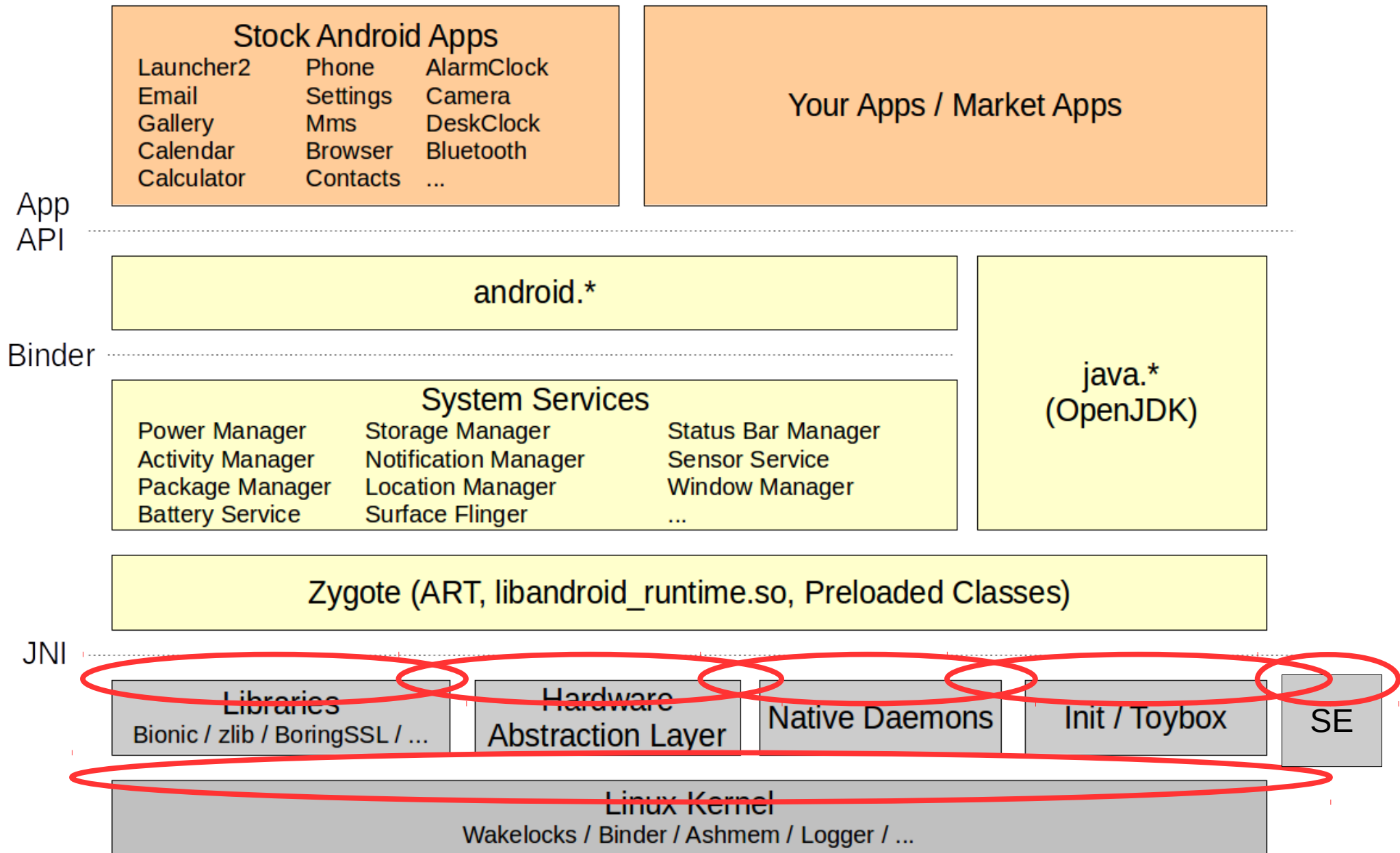
6. Underneath the Framework

- Architecture
- Integration points
- Certification before Treble
- Certification after Treble
- What's added by Treble?

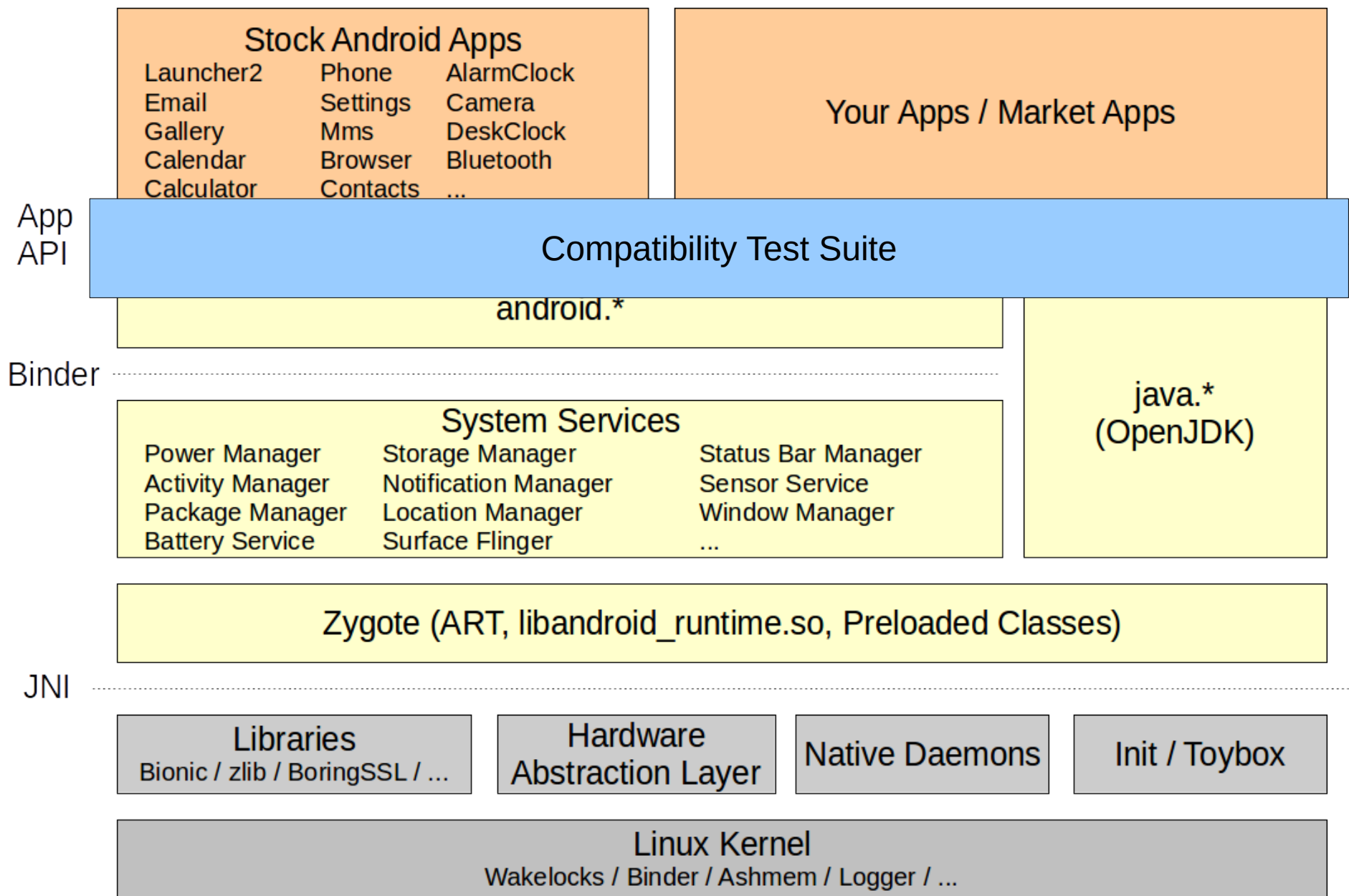
6.1. Architecture



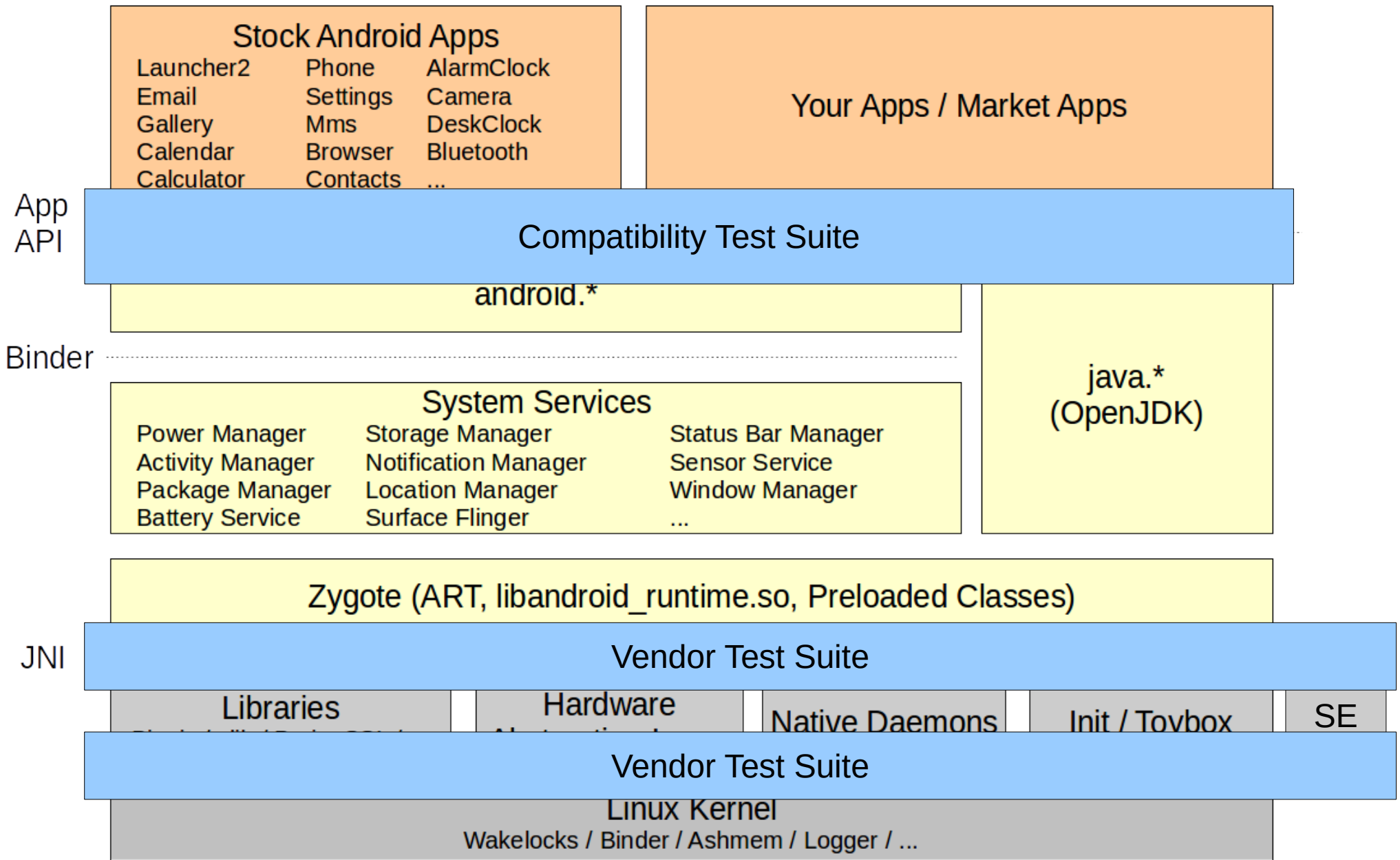
6.2. Integration Points



6.3. Certification Before Treble

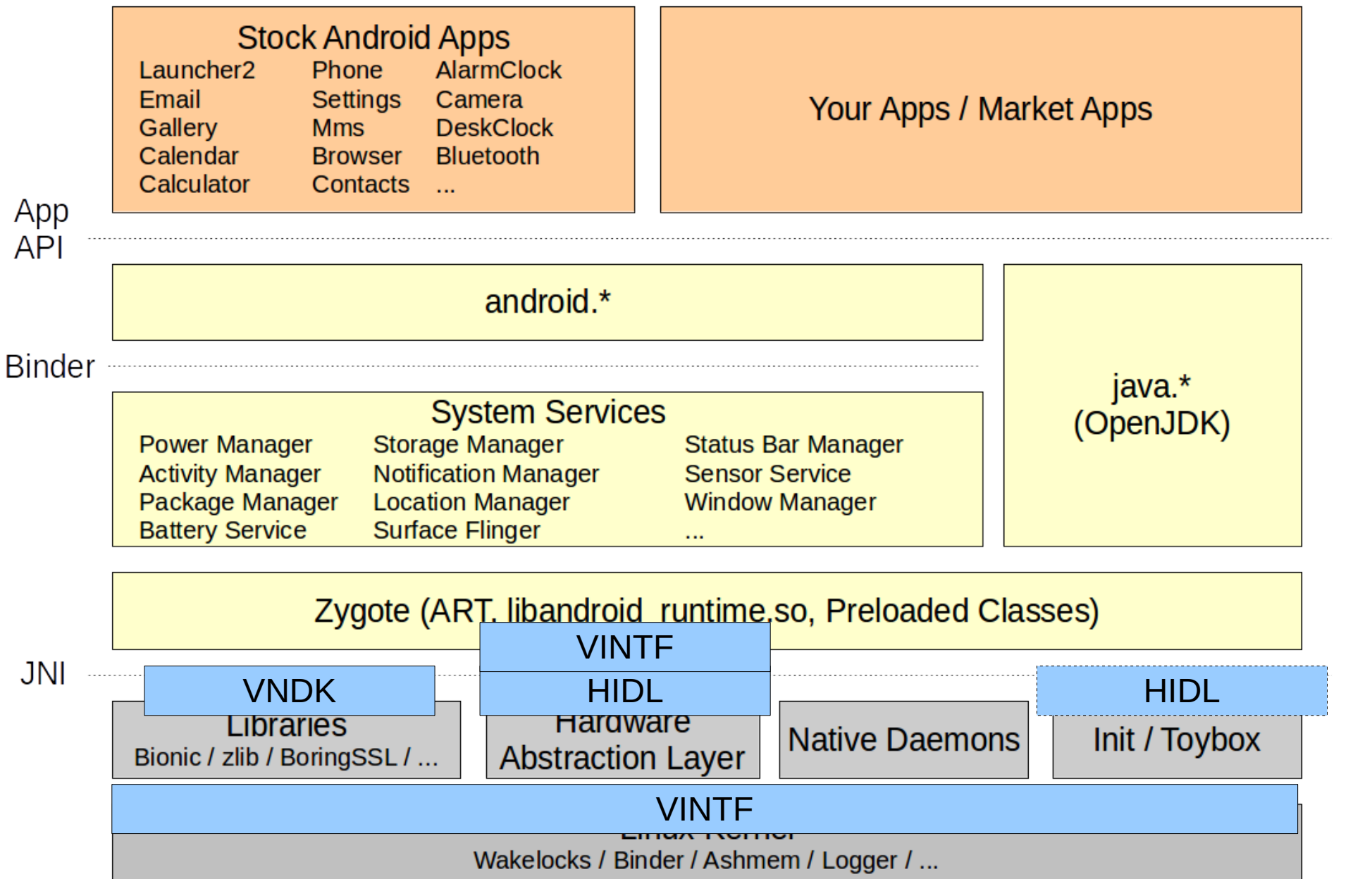


6.4. Certification After Treble



6.5. What's added by Treble?

- Linux kernel requirements
- HIDL – Hardware Interface Definition Language
- VNDK – Vendor NDK
- VINTF – Vendor Interface
- VTS – Vendor Test Suite



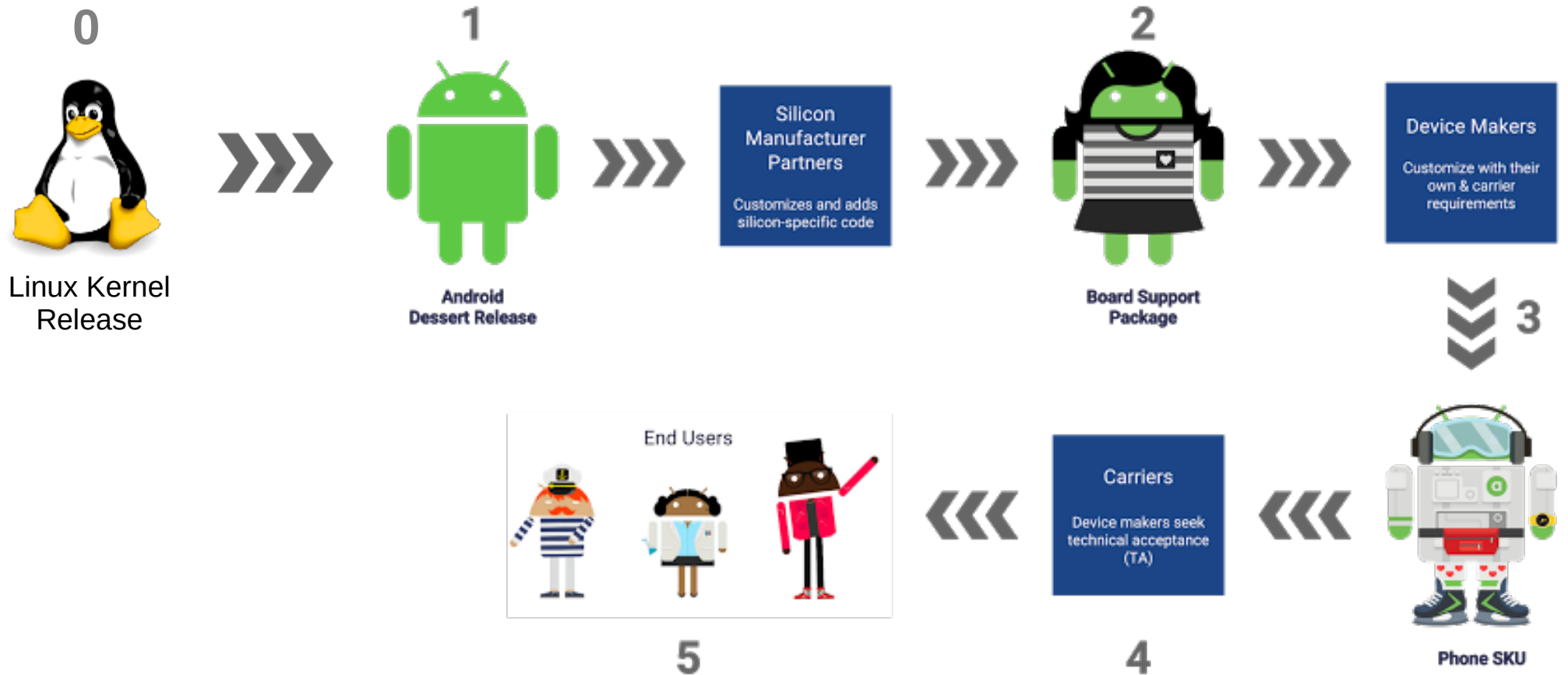
7. Linux Kernel

- Basics
- Release flow
- Real-life
- Treble and the kernel
- Links

7.1. Basics

- Necessary for Android
- Existed prior to Android
- Decision to use based on Linux success
- Google downstream from kernel developers

7.2. Release Flow



7.3. Real-Life

- End-of-Life kernels very common on Android
- Security issues
- While invisible, major implications for user
- Inconsistent kernel versions between vendors
- Inconsistent updates between vendors

7.4. Treble and the Kernel

- Separate deliverables based on each player's role in the update ecosystem.
- VTS-enforced aspects
- Basic rules for:
 - Kernel versions – use of LTS
 - Basic config
 - System call ABI/API
 - FS + FS features
- LTS 4.4 supported for 6 years instead of 2
- Kernel customization through:
 - Kernel config overrides
 - Loadable modules
 - DT overlays

7.5. Links

- Main doc from Google:
 - <https://source.android.com/devices/architecture/kernel/>
- LTS:
 - <https://www.kernel.org/category/releases.html>

8. HIDL

- HAL's role
- Traditional HAL before 8.x
- HIDL
- Links

8.1. HAL's Role

- Per-device-type hardware abstraction
- Example:
 - SurfaceFlinger uses hwcomposer HAL
 - Location uses gps HAL
 - Lights uses lights HAL
 - Etc.
- Google specifies HAL signature
- Manufacturer/SoC vendor provides HAL implementation
- Reference implementations:
 - Leads devices found in AOSP
 - SoC vendor reference designs/boards in BSP

8.2. Traditional HAL Before 8.x

- Google specifies HALs as “C” header files
- HAL module author uses header in implementation
- Resulting binary shipped as part of release
- Modules loaded at boot time by system services
- Headers could (and did) change between versions
- Required reworking, rebuilding, reshipping new version
- To update to a new version of Android:
 - All modules had to be updated, be they trivial or difficult

8.3. HIDL

- “Hardware Interface Definition Language”
- New layer under system services
- Formalized **and** versioned HAL interface definitions
- Similar to AIDL, yet different
- Example HIDLs in 8.x/Oreo:
 - graphics/composer 2.1
 - gnss 1.0
 - Etc.
- Once published, a given HIDL definition is immutable:
 - Even in 9.x/P, graphics/composer 2.1 and gnss 1.0 will be the same as in 8.x.
- Fresh port assumes using latest HAL sig available
- Most importantly:
 - Modules created against a given signature should continue to work so long as that signature is supported.
- Depends on Google, but incentives are aligned
- More on this in tomorrow's presentation

8.4. Links

- General doc:
 - <https://source.android.com/devices/architecture/hidl/>
- C++
 - <https://source.android.com/devices/architecture/hidl-cpp/>
- Java
 - <https://source.android.com/devices/architecture/hidl-java/>

9. VNDK

- Before
- What VNDK does
- Links

9.1. Before

- “Anything goes” under the hood
- Sometimes taken to extremes
- Ex. major handset manufacturer w/ hot new device:
 - Expecting:
 - Trade press scrutiny
 - Benchmarks
 - So ... tweak everything for performance and battery:
 - Libraries
 - Core components
 - Anything and everything under the sun
- Result: Significant delta between canonical AOSP and real code/binaries shipping in devices.
- Therefore: Difficult to upgrade

9.2. What VNDK Does

- Provides a pre-defined way to mitigate/manage differences between Google-minted libs and vendor libs.
- Quite a few concepts introduced by VNDK to define library categories and explain how they are to be used.
- Boring/Tedious explanation (very dry)
- Refer to official doc
- Will need to assign this to someone on your team
- Google is trying to play funambulist on this

9.3. Links

- Main doc:
 - <https://source.android.com/devices/architecture/vndk/>

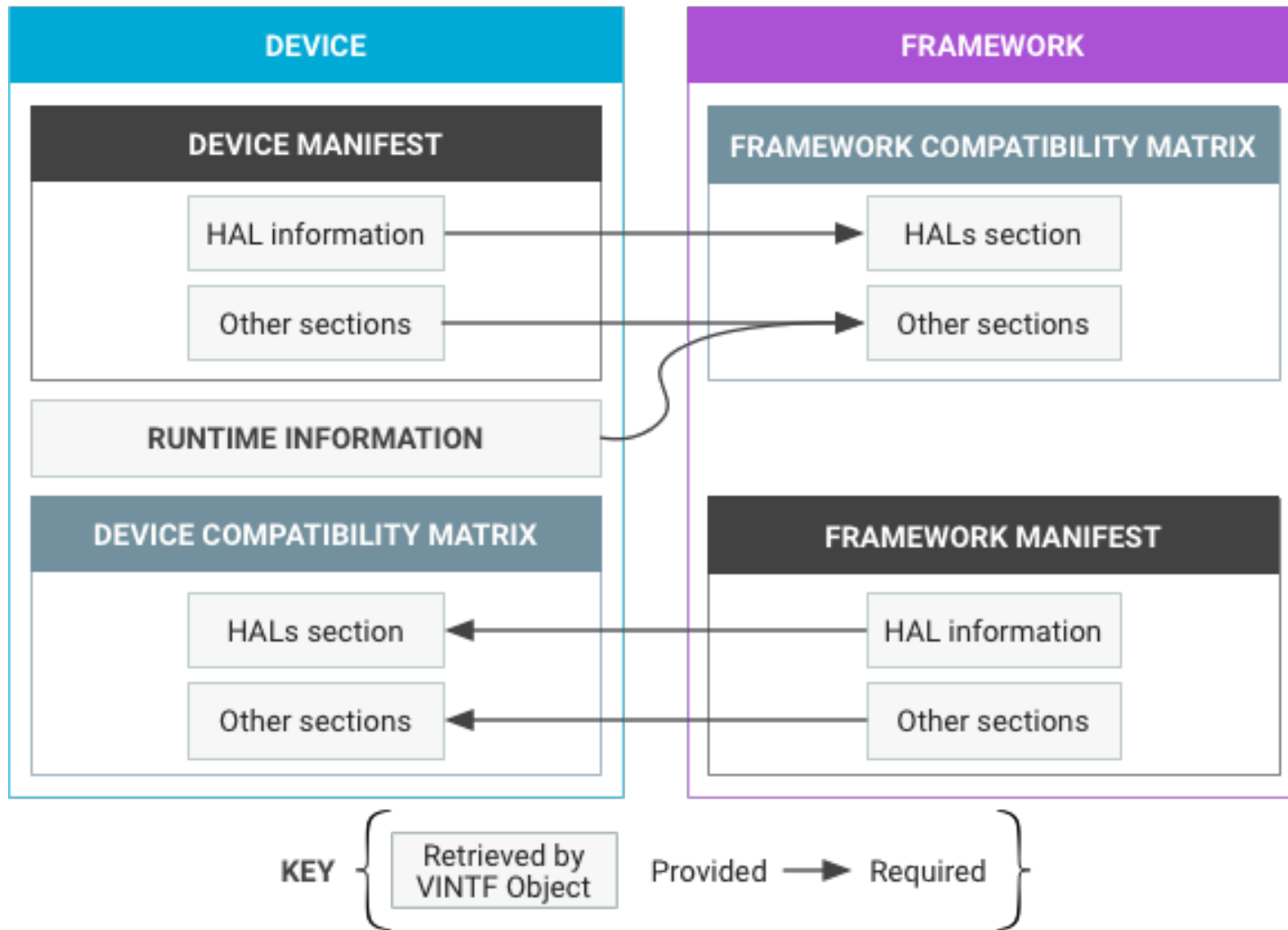
10. VINTF

- Basics
- Architecture
- Details
- Links

10.1. Basics

- Need to formalize version “handshake” between device and OTA service on upgrade.
- Specifically, need to match:
 - HIDL versions
 - Kernel specifics
 - SE policies
 - Android Verified Boot (AVB) version
- Solution:
 - Vendor Interface Object (VINTF)

10.2. Architecture



10.3. Details

- Device manifest:
 - Is device's HW enablement supported by framework?
- Framework manifest:
 - Does new framework provide what the device needs?
- Device compatibility matrix:
 - What does device expect of new framework?
- Framework compatibility matrix:
 - What does new framework expect of device?
- Google doc specifies:
 - Manifest format
 - Compatibility matrices
 - Matching rules
- Need a good understanding as well
- Likely needs a person to be responsible for this as well

10.4. Links

- Main doc:
 - <https://source.android.com/devices/architecture/vintf/>

11. VTS

- Similar to CTS
- Ensures that all the above is followed
- Test categories:
 - HIDL HAL Tests
 - Kernel Tests
 - LTP
 - Linux Kselftest
 - VNDK (Vendor Native Development Kit) Tests
 - Performance Tests
 - Fuzz Tests
 - Security Tests
- Even w/o certification, passing CTS will simplify OTA

- Links:
 - Main doc:
 - <https://source.android.com/compatibility/vts/>
 - More details (user manual, etc.):
 - <https://android.googlesource.com/platform/test/vts/+master/README.md>

12. Implications

- Significant reduction in customization for GMS
- Non-exhaustive no-go list:
 - System service internals
 - HAL definition reworking/extension
 - Core library replacement by tweaking dependent components' APIs
 - Tweaking existing SE policies
 - Any change that impacts OTA
 - Any change that breaks VTS
- Even permitted changes are complex, ex:
 - Tweaking libraries vs. VNDK rules
 - Using vendor-specific services instead of tweaking existing ones

- Big benefit in adhering to “vanilla” even if no GMS interest:
 - Simplified OTA
- If no GMS, still as open as before to tweak everything:
 - Price = reduced OTA capability, likely no path to upgrade, etc.
- Greater scrutiny required over SoC/partner work to ensure Treble “conformance”.
- Incentive is there for SoC vendors since some of their customers need GMS certification.
- Varying degrees of quality from board vendors and outsourcing partners:
 - Do your homework

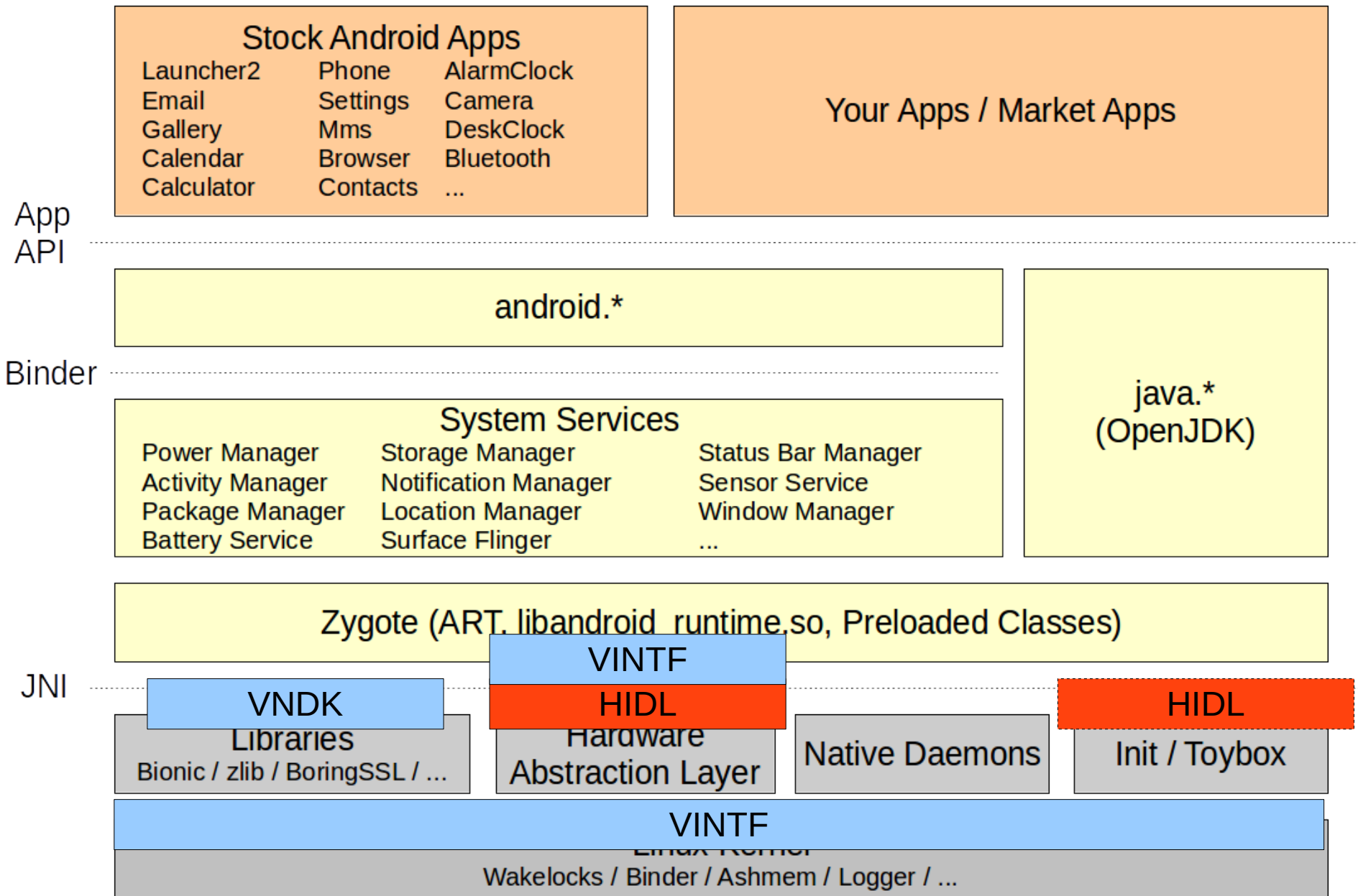
13. Recommendations

- Several years before Treble benefits materialize
- Solid ground established by Google
- Likely some organic learning left to iron kinks out
- In the mean time:
 - Make sure you have a firm understanding of how Treble changes your Android device design, development, integration and support plans.
 - Have your technical teams review the Treble documentation made available by Google and possibly seek training so they can start familiarizing themselves with its intricacies.
 - If certification is on your agenda, start paying special attention to the VTS in your planning. If you have CTS as part of your checklists, a good start is adding equivalent entries for VTS.
 - When selecting partners or suppliers, make sure their work and/or the software they deliver falls in line with Treble's requirements, especially if certification is a target.

Treble's HIDL in Detail

1. Architecture Recap
2. Basics
3. HAL Architecture Rework
4. Walkthrough
5. Adding a New HIDL
6. Support Infrastructure

1. Architecture Recap



2. Basics

- HAL's role
- Traditional HAL before 8.x
- HIDL
- Links

2.1. HAL's Role

- Per-device-type hardware abstraction
- Example:
 - SurfaceFlinger uses hwcomposer HAL
 - Location uses gps HAL
 - Lights uses lights HAL
 - Etc.
- Google specifies HAL signature
- Manufacturer/SoC vendor provides HAL implementation
- Reference implementations:
 - Leads devices found in AOSP
 - SoC vendor reference designs/boards in BSP

2.2. Traditional HAL Before 8.x

- Google specifies HALs as “C” header files
- HAL module author uses header in implementation
- Resulting binary shipped as part of release
- Modules loaded at boot time by system services
- Headers could (and did) change between versions
- Required reworking, rebuilding, reshipping new version
- To update to a new version of Android:
 - All modules had to be updated, be they trivial or difficult

2.3. HIDL

- “Hardware Interface Definition Language”
- New layer under system services
- Formalized **and** versioned HAL interface definitions
- Similar to AIDL, yet different
- Example HIDLs in 8.x/Oreo:
 - graphics/composer 2.1
 - gnss 1.0
 - Etc.
- Once published, a given HIDL definition is immutable:
 - Even in 9.x/P, graphics/composer 2.1 and gnss 1.0 will be the same as in 8.x.
- Fresh port assumes using latest HAL sig available
- Most importantly:
 - Modules created against a given signature should continue to work so long as that signature is supported.
- Depends on Google, but incentives are aligned
- More on this in tomorrow's presentation

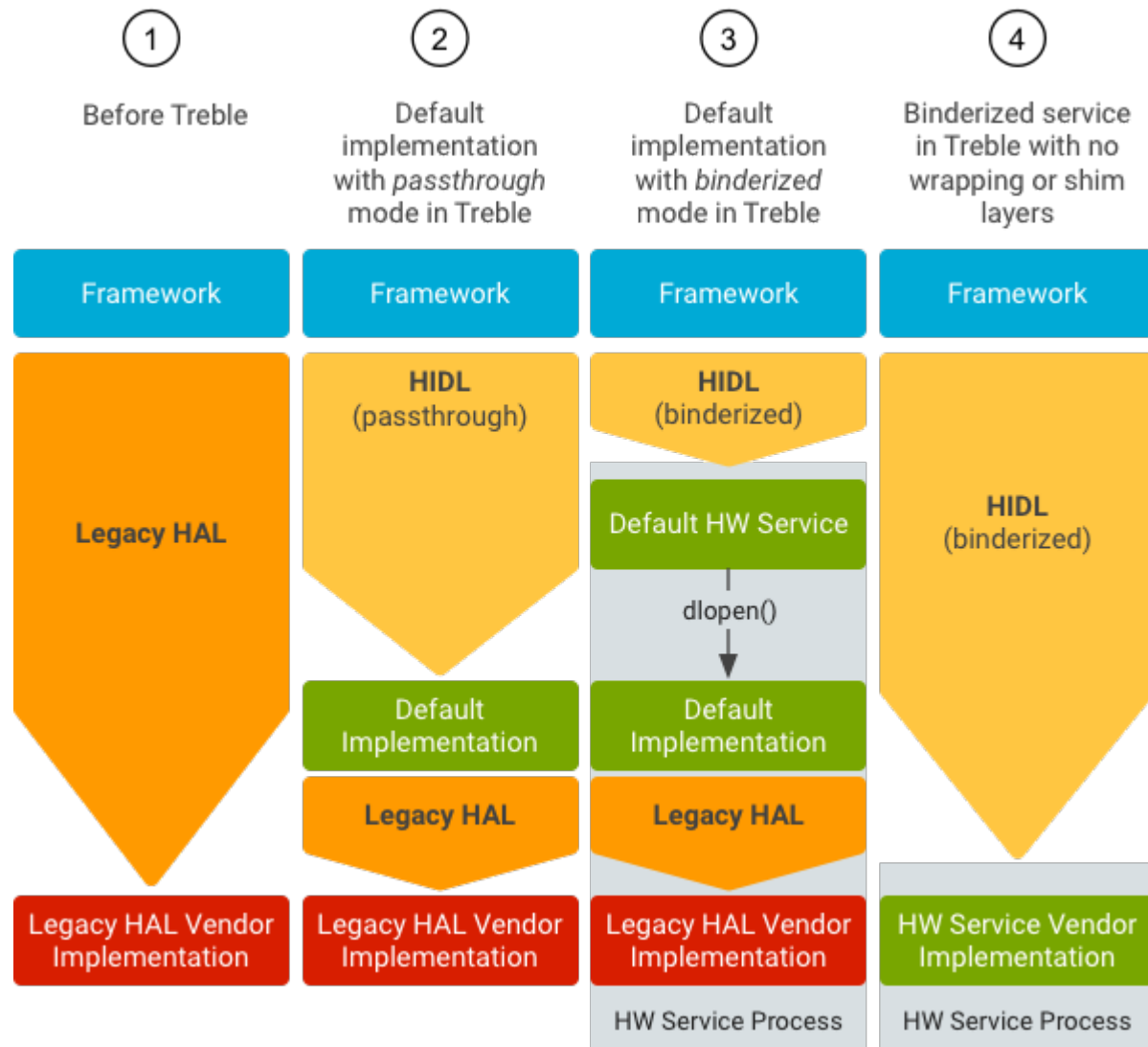
2.4. Links

- General doc:
 - <https://source.android.com/devices/architecture/hidl/>
- C++
 - <https://source.android.com/devices/architecture/hidl-cpp/>
- Java
 - <https://source.android.com/devices/architecture/hidl-java/>

3. HAL Architecture Rework

- Overall architecture
- Detailed architecture / Java
- Detailed architecture / C++

3.1. Overall Architecture

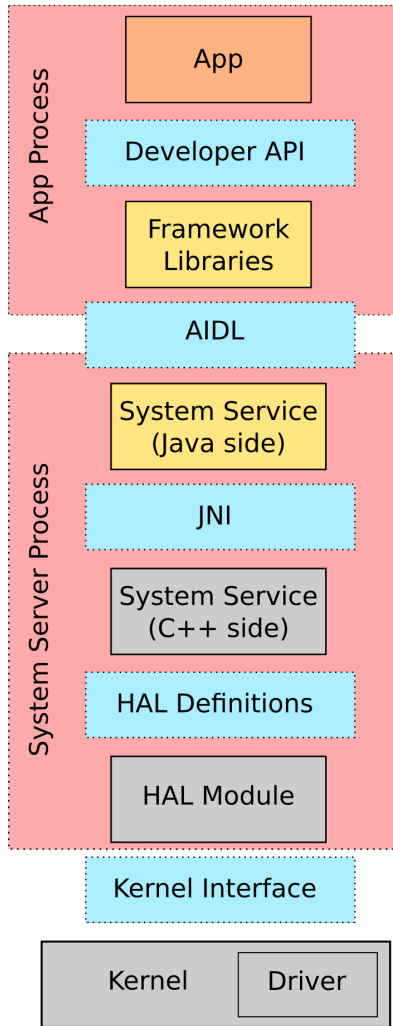


3.2. Detailed Architecture / Java

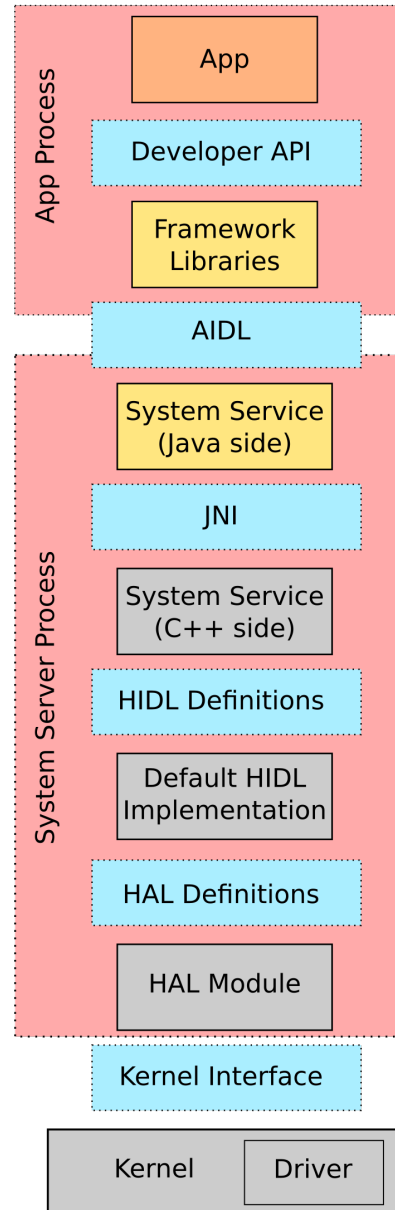
Before Treble

After Treble

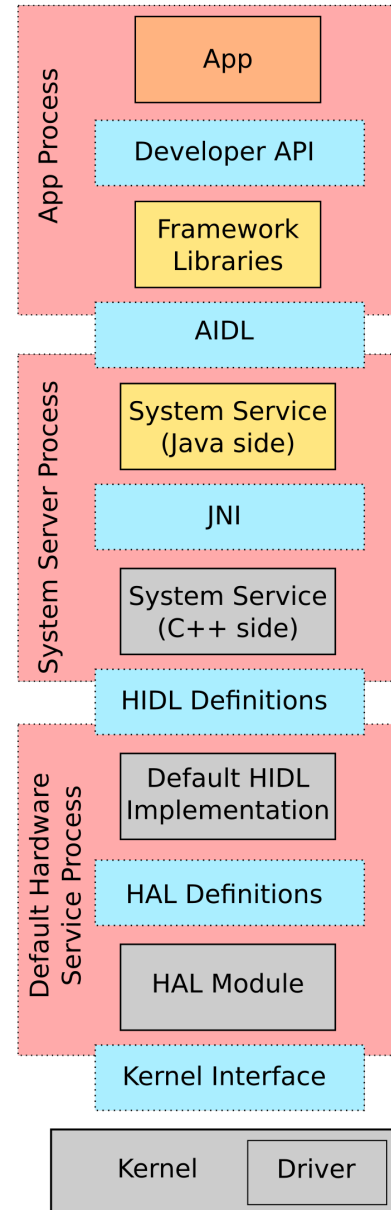
Classic HAL



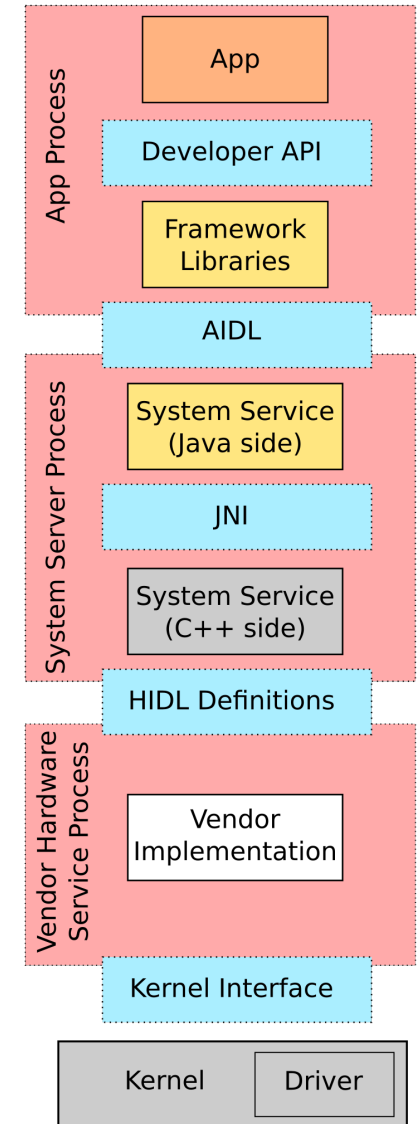
SP/Passthrough



Binderized



Binderized (no shim or wrapper)

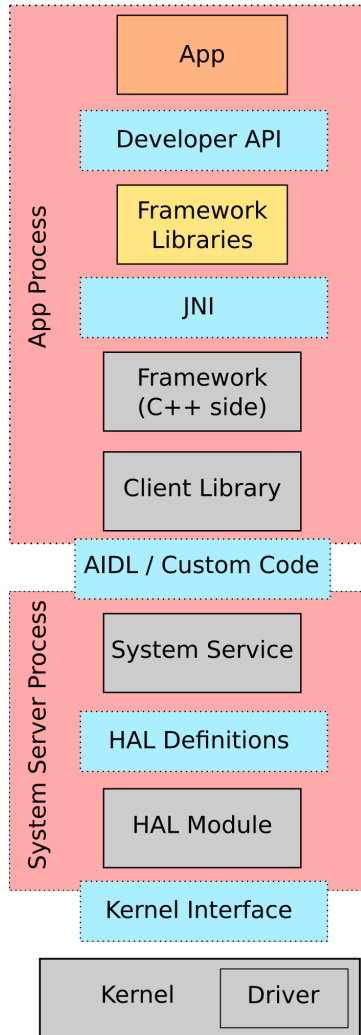


3.3. Detailed Architecture C++

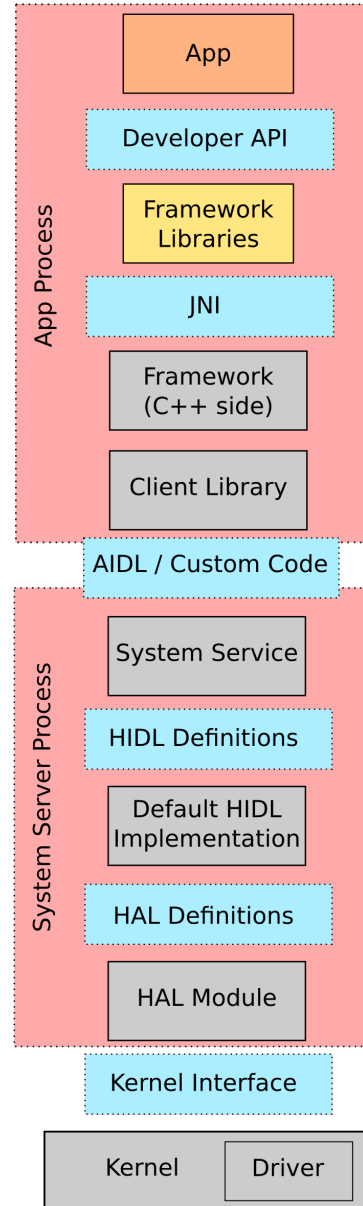
Before Treble

After Treble

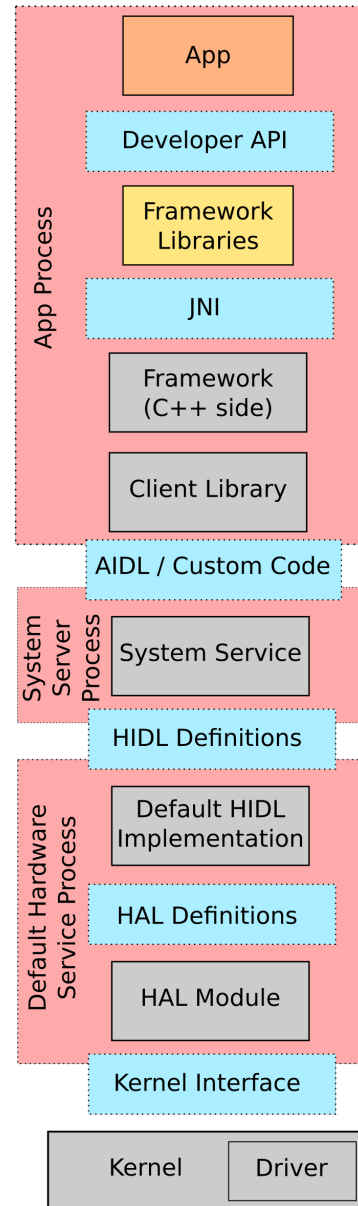
Classic HAL



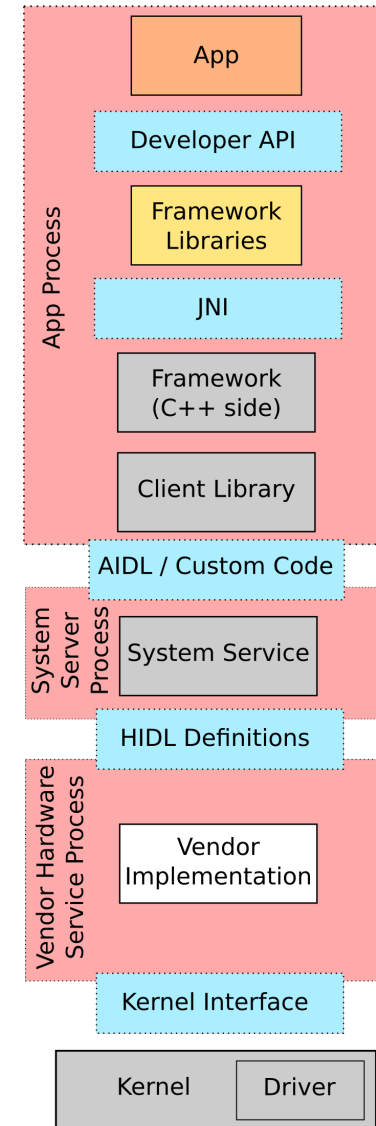
SP/Passthrough



Binderized



Binderized (no shim or wrapper)



4. Walkthrough

- JNI Layer
- HIDL Layer
- HIDL Glue

4.1. JNI Layer

- JNI Java<->C/C++ method/function registration -- same before and after Treble:
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/frameworks/base/services/java/com/android/server/SystemServer.java
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/frameworks/base/services/java/com/android/server/SystemServer.java#367
 - `System.loadLibrary("android_servers");` ==> Loads `libandroid_servers.so`
- This file defines which JNI files to compile into `libandroid_servers.so`:
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/frameworks/base/services/core/jni/Android.mk
 - Including: `com_android_server_lights_LightsService.cpp`
- This file defines the rules to build `libandroid_servers.so`:
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/frameworks/base/services/Android.mk
- Another file compiled into `libandroid_servers.so` is `onload.cpp`:
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/frameworks/base/services/core/jni/onload.cpp
 - That file contains a function called: `JNI_OnLoad()`
 - `JNI_OnLoad()` is automatically called by ART when `libandroid_servers.so` is loaded.
- `JNI_OnLoad` contains calls to JNI registration functions, such as `register_android_server_LightsService(env)`:
 - [http://aosp.operators.com/xref/android-8.1.0_r1/xref/frameworks/base/services/core/jni/com_android_server_lights_LightsService.cpp#register_android_server_LightsService\(\)](http://aosp.operators.com/xref/android-8.1.0_r1/xref/frameworks/base/services/core/jni/com_android_server_lights_LightsService.cpp#register_android_server_LightsService)
 - `JniRegisterNativeMethods()`
- `jniRegisterNativeMethods()` calls into the ART VM to register C/C++ calls against a Java class.

4.2. HIDL Layer

- http://aosp.opersys.com/xref/android-8.1.0_r9/xref/frameworks/base/services/core/java/com/android/server/lights/LightsService.java:
 - LightsService constructor
 - setLightLocked calls on setLight_native
- [http://aosp.opersys.com/xref/android-8.1.0_r9/xref/frameworks/base/services/core/jni/com_android_server_lights_LightsService.cpp:setLight_native\(\)](http://aosp.opersys.com/xref/android-8.1.0_r9/xref/frameworks/base/services/core/jni/com_android_server_lights_LightsService.cpp:setLight_native())
 - LightHal::associate();
 - Ilight::getService();
- [http://aosp.opersys.com/xref/android-8.1.0_r9/xref/hardware/interfaces/light/2.0/default/Light.cpp:HIDL_FETCH_ILight\(\)](http://aosp.opersys.com/xref/android-8.1.0_r9/xref/hardware/interfaces/light/2.0/default/Light.cpp:HIDL_FETCH_ILight())
 - GetLightDevice()
 - hw_get_module ()
 - dlopen() ==> results in lights.[hw-board].so to be loaded

4.3. HIDL Glue

- There are two main paths with HIDL:
 - Same-Process (passthrough)
 - Binderized
- This build file can generate BOTH options:
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/hardware/interfaces/light/2.0/default/Android.mk
 - android.hardware.light@2.0-impl.so ==> default same-process implementation
 - android.hardware.light@2.0-service ==> binderized remote process
- The default binderized process in the case of lights (i.e. android.hardware.light@2.0-service) essentially loads android.hardware.light@2.0-impl.so.
- Example *legacy* lights module implementation:
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/device/huawei/angler/liblight/lights.c
 - This is the file that would be calling into drivers in the kernel.
- The decision to use passthrough vs. binderized service depends on the board-specific manifest.xml:
 - http://aosp.operators.com/xref/android-8.1.0_r9/xref/device/generic/goldfish/manifest.xml

5. Adding a New HIDL

- New interface definition
- Use of update-makefiles.sh
- Use of hidl-gen
- See/implement default implementation
- Tweak default implementation makefiles
- Add entry in product manifest

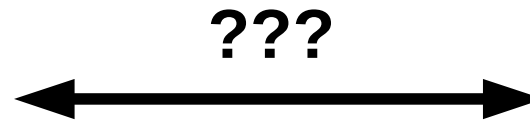
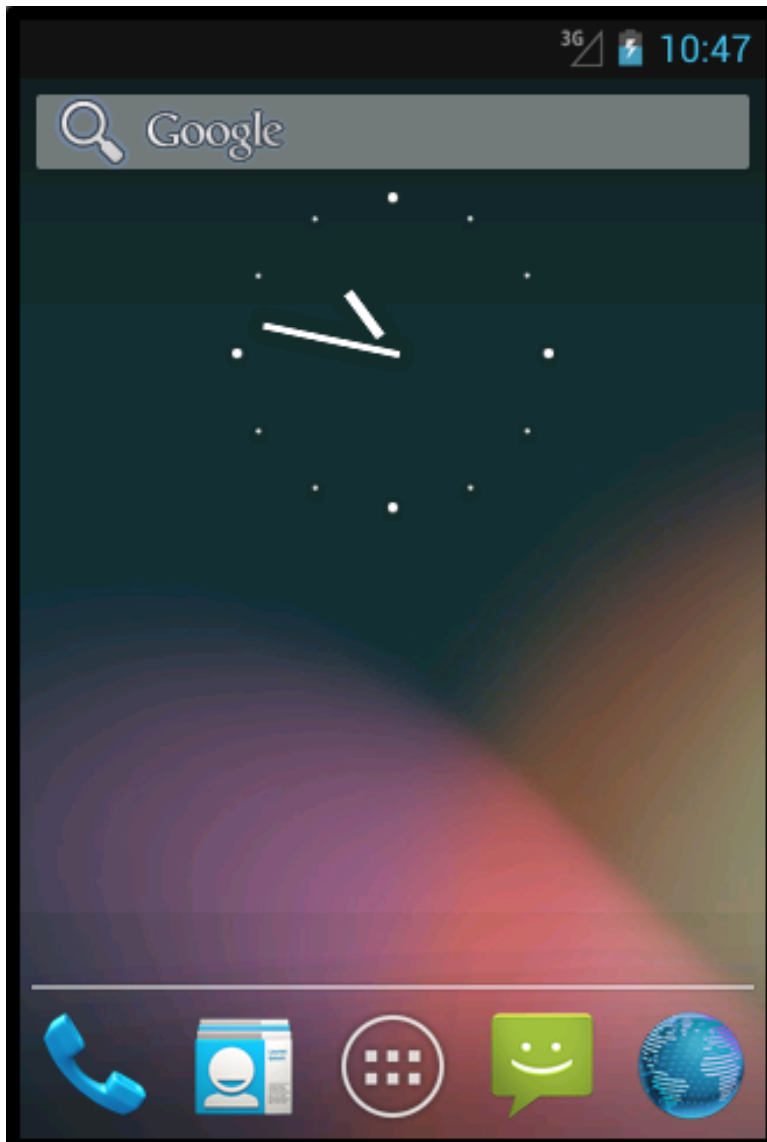
6. Support Infrastructure

- HIDL tools (host + device) + libraries
- Manifest file parts + device-specific mk files
- Online doc/reference
- On-device:
 - File locations
 - Loaded files in address space
 - Processes
 - Etc.

Inside Android's UI

- Android's UI, what's that?
- Architecture Basics
- Display Core
- OpenGL
- Input Layer
- Relevant Apps and Services
- System Startup
- References and Pointers

1. Android's UI, what's that?



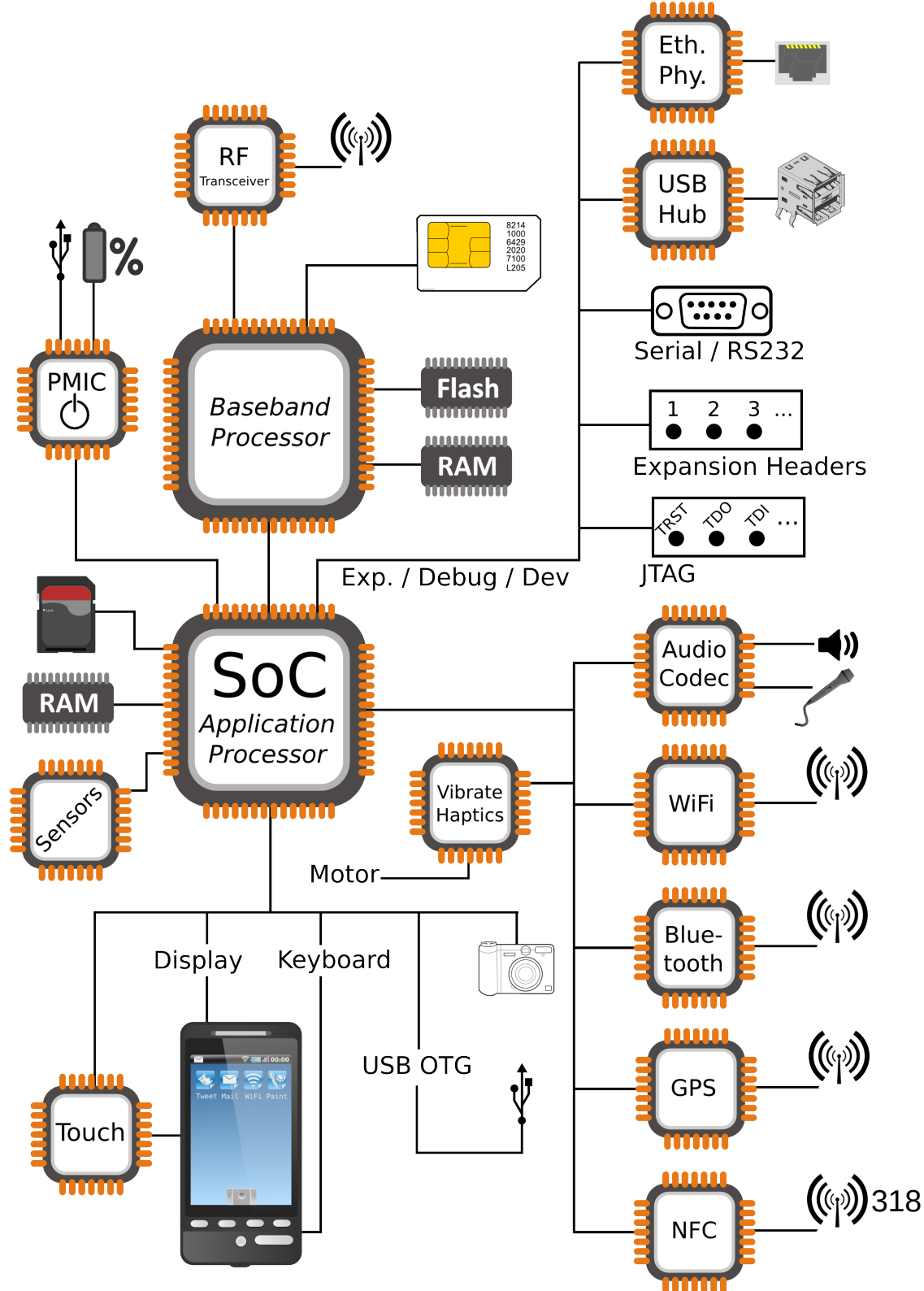
- SoC / GPU
- Touch input
- LCD
- Keyboard

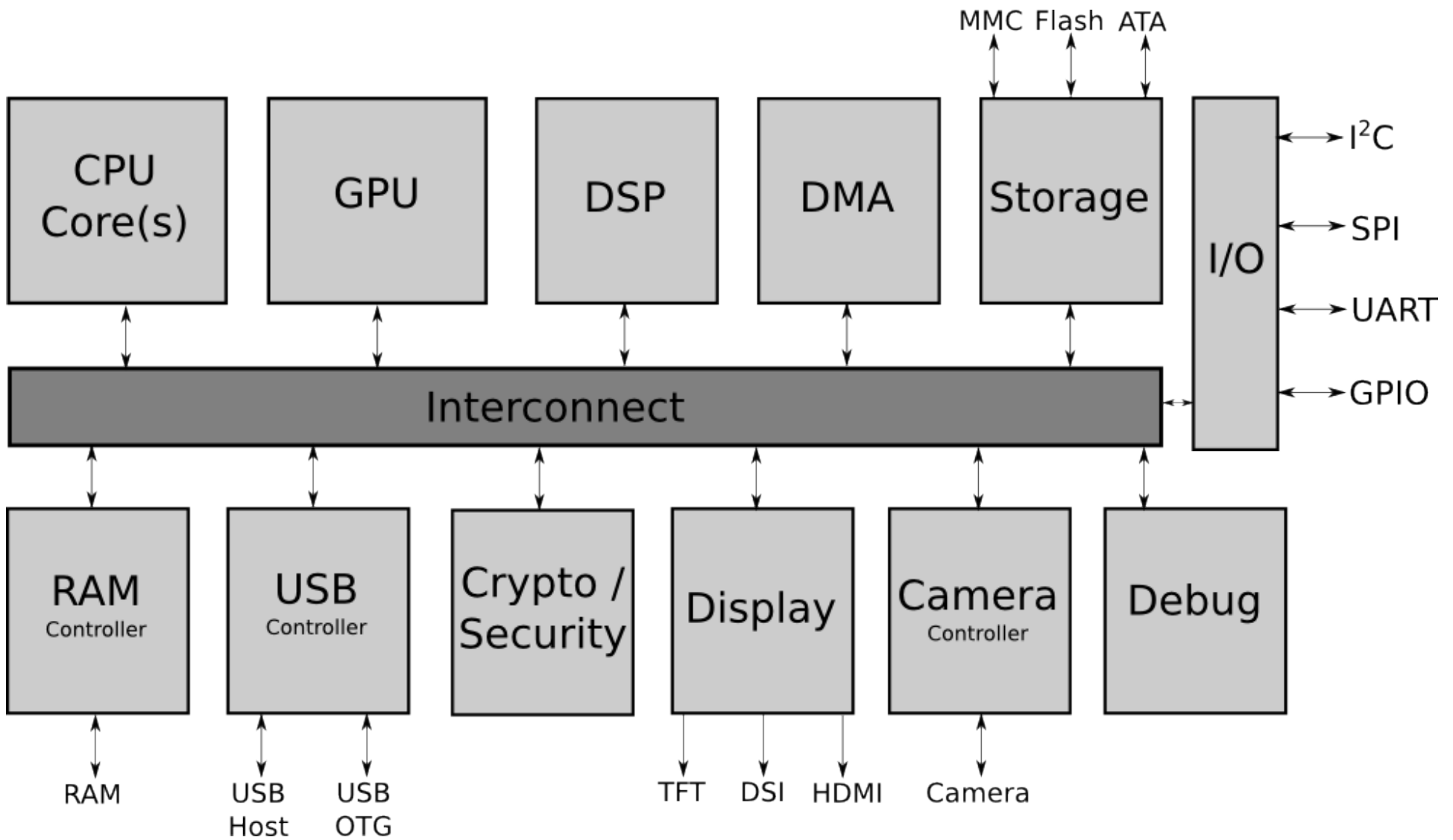
1.1. What's NOT covered here

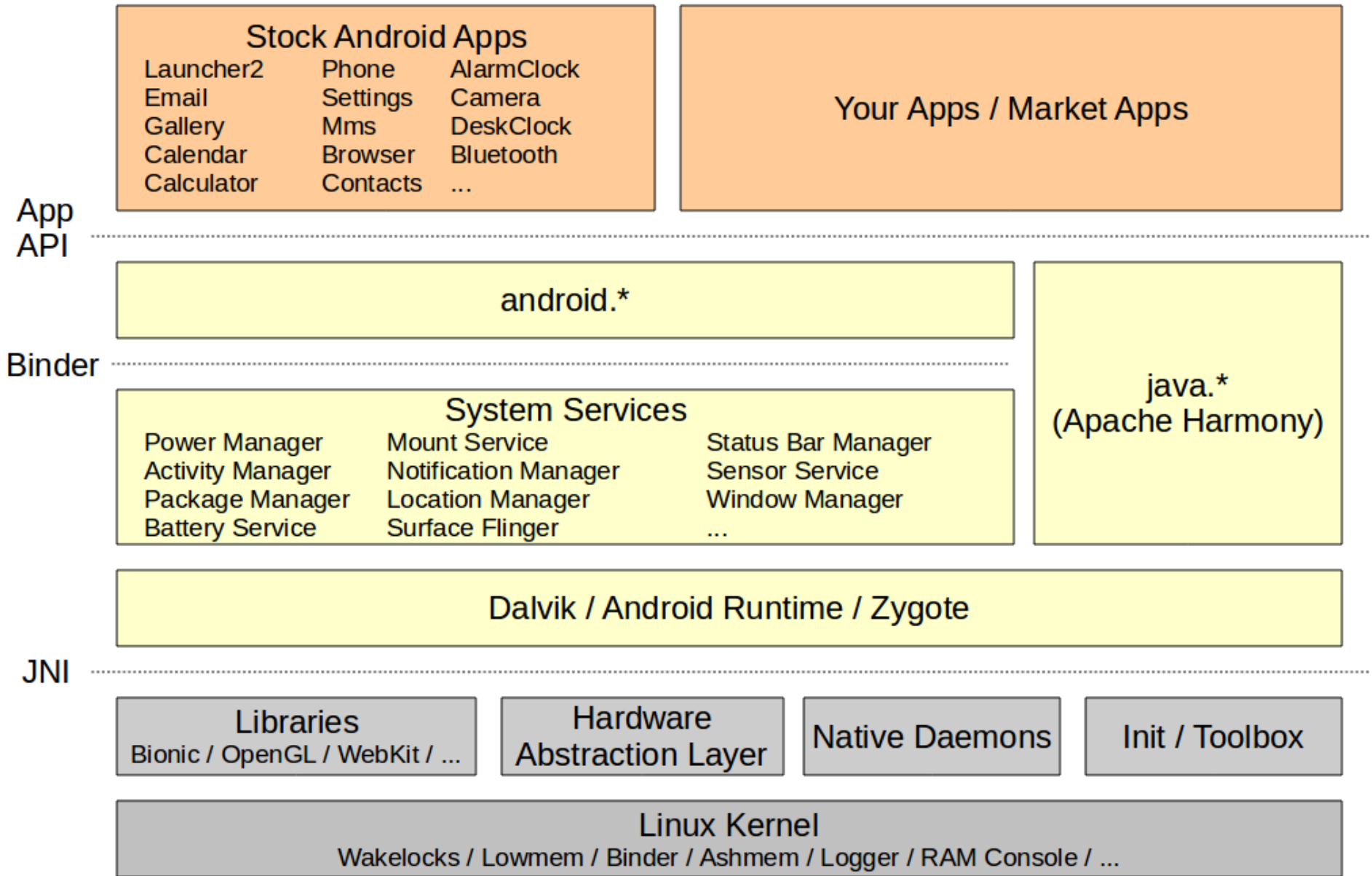
- Media layer
- StageFright
- Video playback
- Camera
- DRM
- Etc.

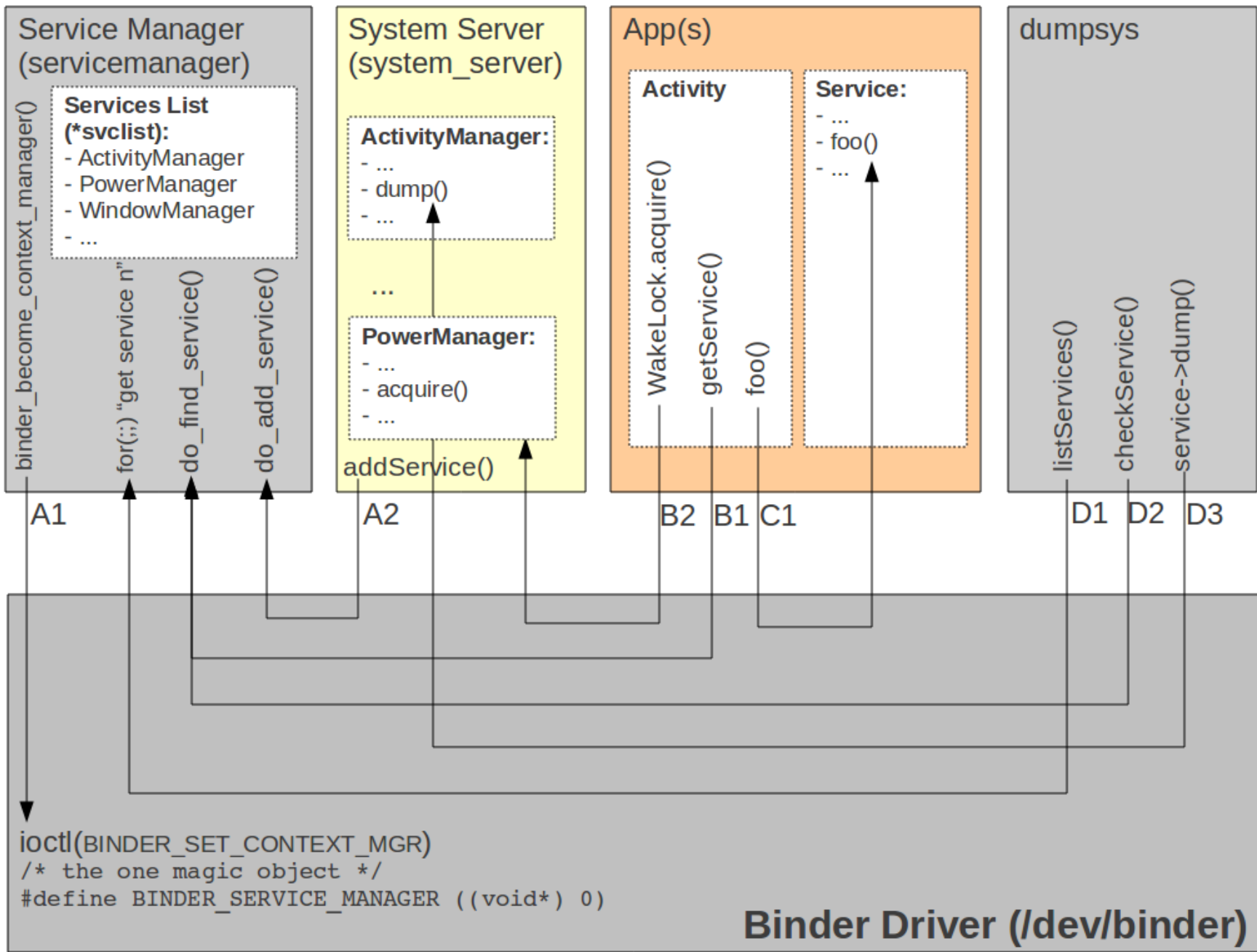
2. Architecture Recap

- Hardware used to run Android
- AOSP
- Binder
- System Services
- HAL









System Services

System Server

Java-built Services

Power Manager	Mount Service
Activity Manager	Notification Manager
Package Manager	Location Manager
Battery Service	Search Service
Window Manager	Wallpaper Service
Status Bar	Headset Observer
Clipboard Service	...

C-built Services

Sensor Service

Surface Flinger

Media Service

Audio Flinger
Media Player Service
Camera Service
Audio Policy Service

Includes:

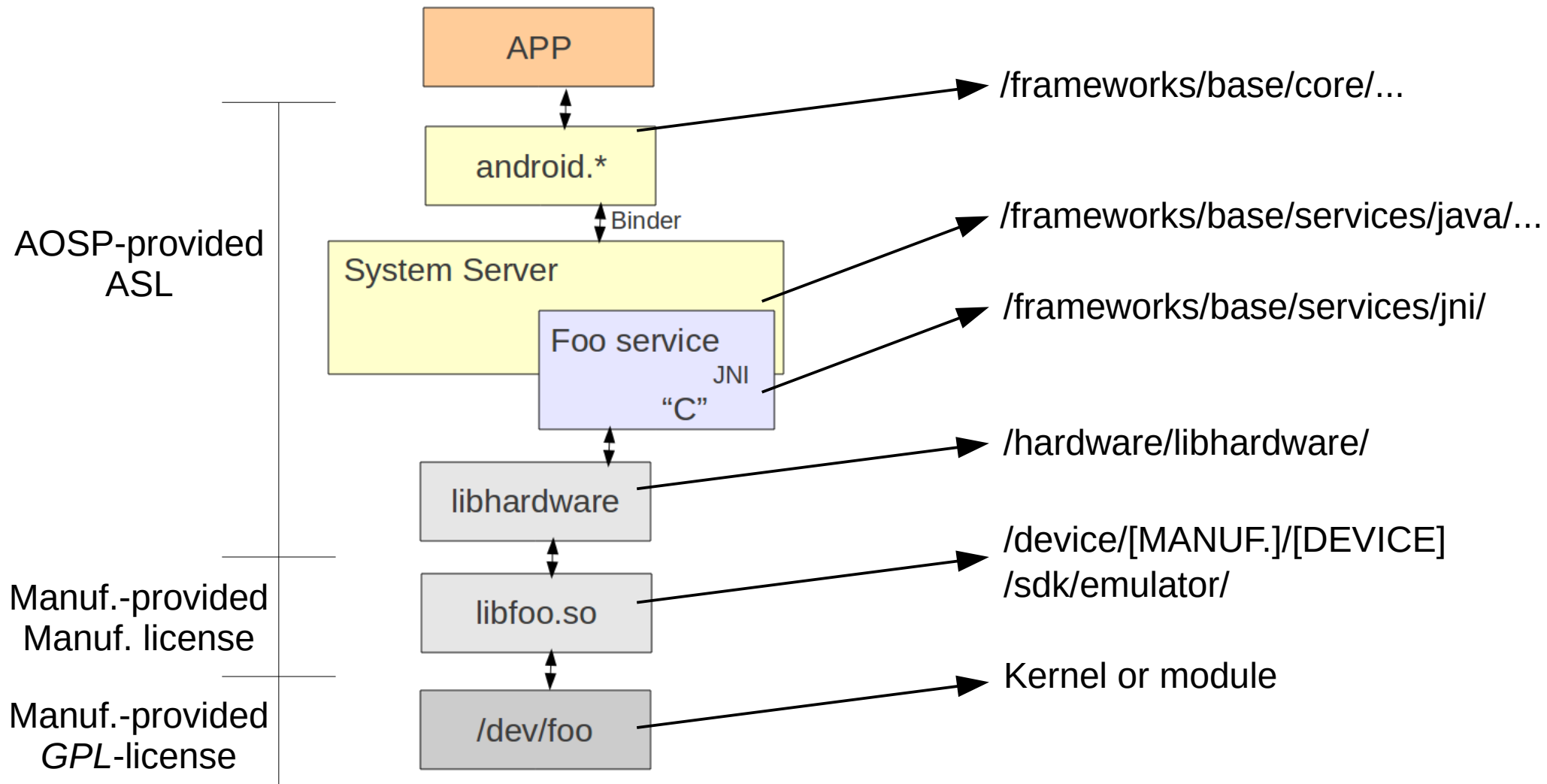
- StageFright
- Audio effects
- DRM framework

Phone App

JNI

Native Methods for
Java-built Services

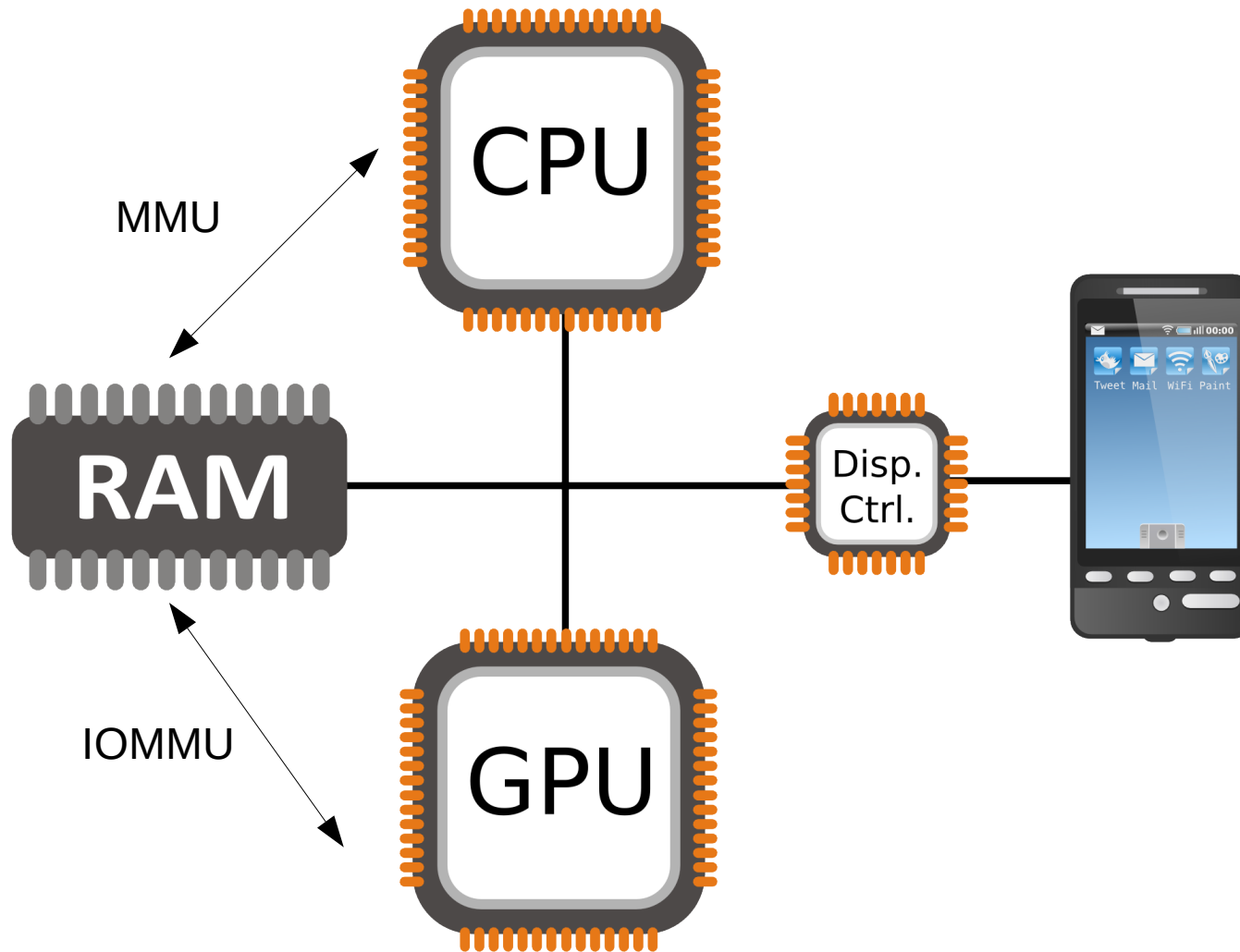
Hardware Abstraction Layer



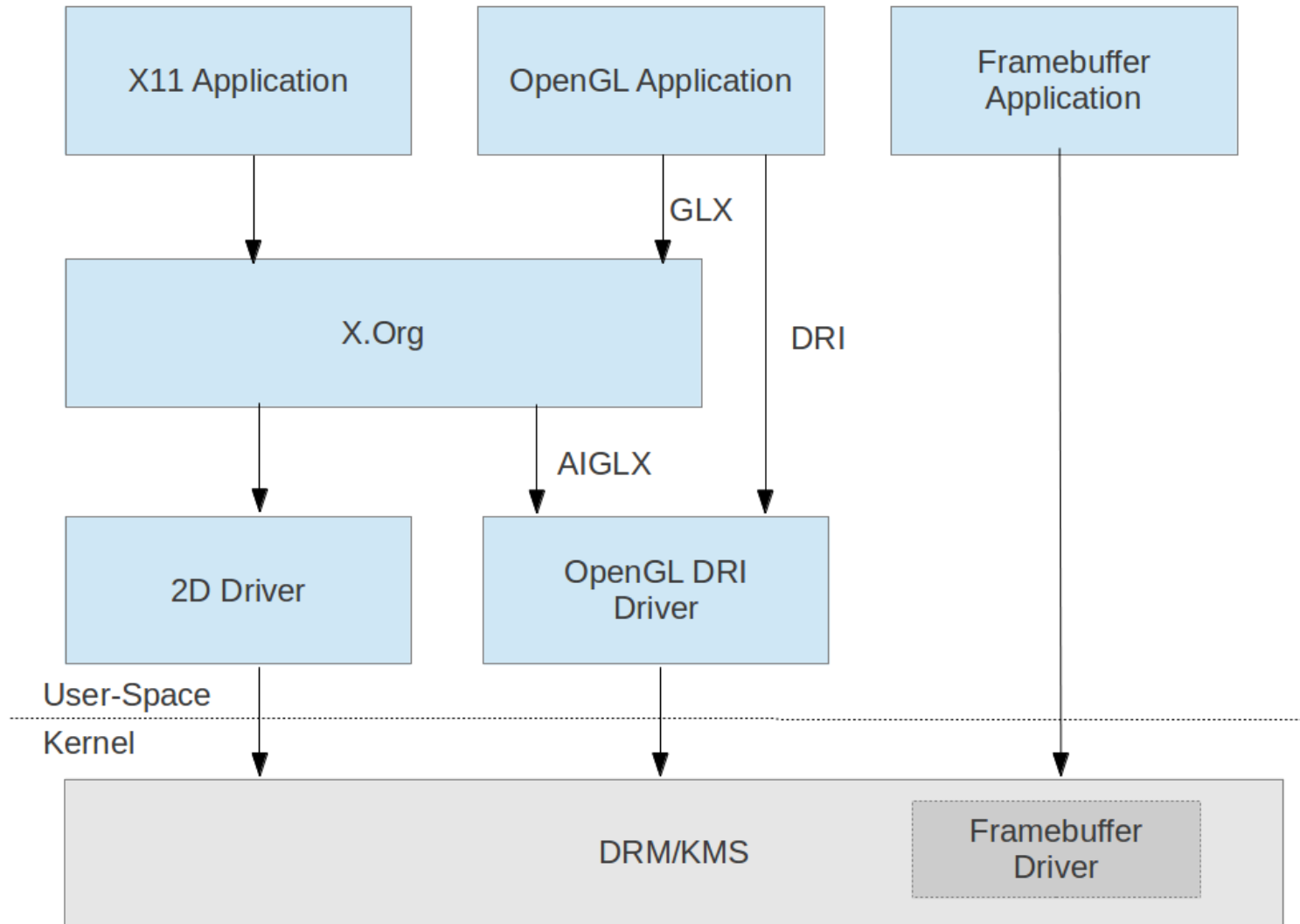
3. Display Core

- Display Hardware
- Classic Linux display stack
- Display stack in Android
- Kernel driver
- HAL definition
- HAL module
- Surface Flinger
- Window Manager
- Walkthrough

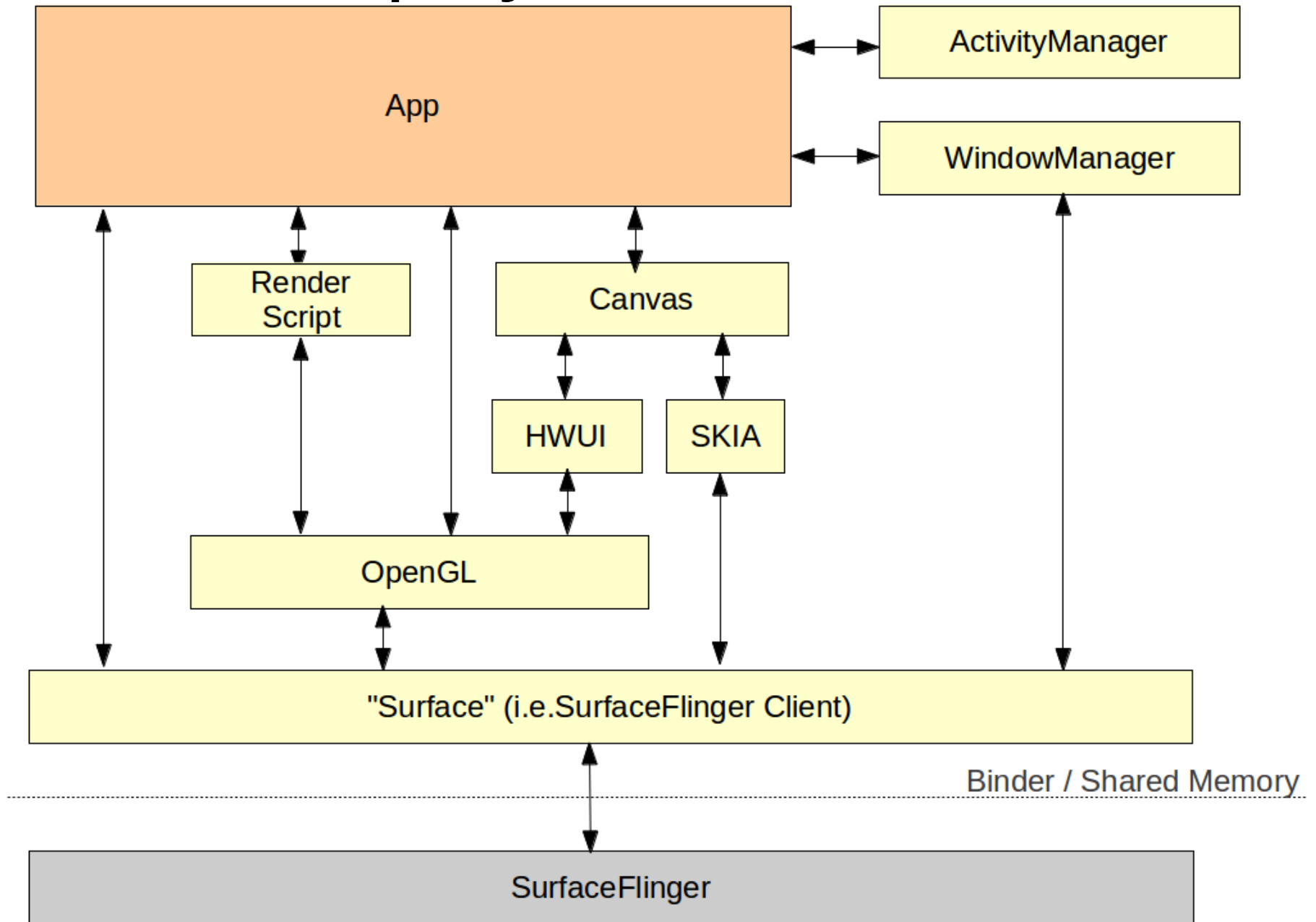
3.1. Display Hardware

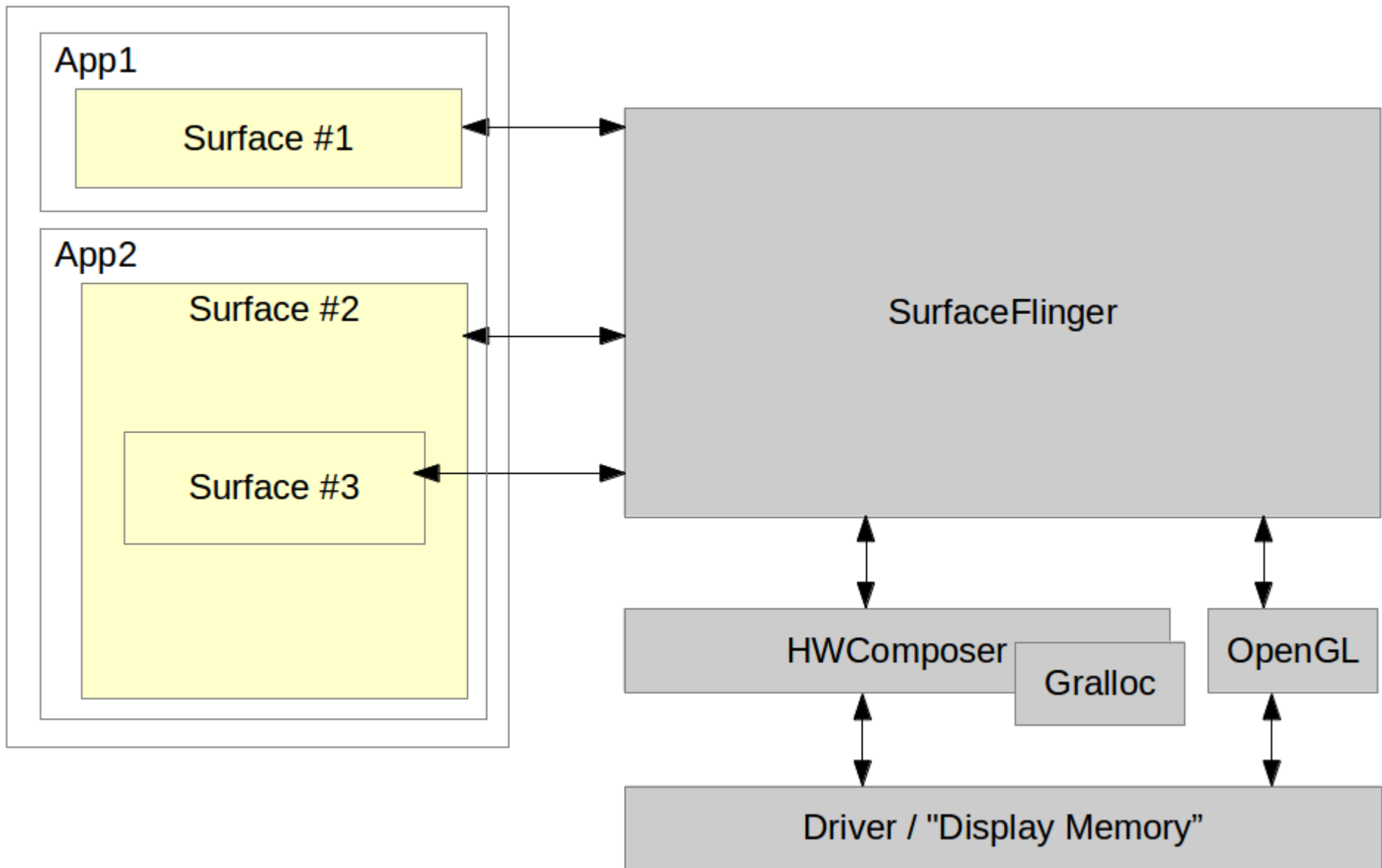


3.2. Classic Linux display stack



3.3. Display stack in Android





3.4. Kernel driver

- Video memory management
- Mode setting
- Checking of parameters
- Motorola Xoom:
 - /dev/nvhdcpl
 - /dev/nvhost-ctrl
 - /dev/nvhost-display
 - /dev/nvhost-dsi
 - /dev/nvhost-gr2d
 - /dev/nvhost-gr3d
 - /dev/nvhost-isp
 - /dev/nvhost-mpe
 - /dev/nvhost-vi
 - /dev/nvmap
 - /dev/tegra-crypto
 - /dev/tegra_avp
 - /dev/tegra_camera
 - /dev/tegra_fuse
 - /dev/tegra_rpc
 - /dev/tegra_sema
- ... whatever hides in hwcomposer HAL module

3.5. HAL Definition

- `/hardware/libhardware/include/hardware/hwcomposer.h`
- `struct hwc_procs`:
 - `invalidate()`
 - `vsync()`
- `struct hwc_composer_device`:
 - `prepare()`
 - `set()`
 - `dump()`
 - `registerProcs()`
 - `query()`
 - `*()`

3.6. HAL module

- Skeleton `/hardware/libhardware/modules/hwcomposer.cpp`
- `/system/lib/hw/hwcomposer.BOARD.so`
- `/system/lib/hw/gralloc.BOARD.so`
- Ex. - Mot Xoom:
 - `hwcomposer.tegra.so`
 - `gralloc.tegra.so`
- Surface Flinger hook:
 - `/frameworks/native/services/surfaceflinger/DisplayHardware`
 - `HWComposer.cpp`
 - Provides fake vsync if none is provided in HW

3.7. Surface Flinger

- Actual server:
 - /frameworks/native/services/surfaceflinger
- Client side:
 - /frameworks/native/libs/gui
- Client / Server interface:
 - ISurfaceComposerClient.cpp
 - ISurfaceComposer.cpp
- This is NOT an aidl'ed service
- All communication is manually marshalled/unmarshalled

3.8. Window Manager

- Server side:
 - /frameworks/base/services/java/com/android/server/wm/
 - WindowManagerService.java
 - Session.java
- Client side:
 - /frameworks/base/core/java/android/view/
 - WindowManager.java
 - WindowManagerImpl.java
 - ViewRootImpl.java
- Interfaces:
 - IWindowManager.aidl
 - IWindowSession.aidl
- Parameters (incl. z-order):
 - See WindowManager.java

3.9. Walkthrough

- Activity Manager relies on Activity Thread
- AT calls on `attach()` and `makeVisible()`
- `makeVisible` does `wm.addView()`
- `wm.addView()` - this also called by `StatusBar` to display itself
 - Creates a new `ViewRootImpl`
 - Call on its `setView()`
- `setView()` calls on `mWindowSession.addToDisplay(...)`
- This results in call to WM's `addWindow()`
- `ViewRootImpl`'s `performTraversals()`
 - Calls on `layoutWindow()`
 - Calls to WM session's `layout()`
 - Call to WM's `layoutWindow()`
 - Call to `createSurfaceLocked()`
 - `new Surface(...)`

frameworks/base/core/java/android/**/*

LocalActivityManager.java: startActivity()

- moveToState()
- startActivityNow()

ActivityThread.java: startActivityNow()

- performLaunchActivity()
- attach() -- gets AM handle and ties to it
- handleResumeActivity()
- makeVisible()

Activity.java: makeVisible()

- wm.addView()

WindowManagerGlobal.java: addView()

- root = new ViewRootImpl()
- root.setView()

ViewRootImpl.java: setView()

- mWindowSession.addToDisplay()

IWindowSession.aidl: addToDisplay()

frameworks/base/services/java/com/android/server/wm/*

Session.java: addToDisplay()

- mService.addWindow()

WindowManagerService.java: addWindow()

...

frameworks/base/core/java/android/*/*

ViewRootImpl.java: performTraversals()

- relayWindow()

- mWindowSession.relayout()

frameworks/base/services/java/com/android/server/wm/*

Session.java: relayWindow()

- mService.relayWindow()

WindowManagerService.java: relayWindow()

- surface = winAnimator.createSurfaceLocked();

WindowStateAnimator.java: createSurfaceLocked()

- new Surface();

4. OpenGL

- What's OpenGL?
- What's in a modern-day GPU?
- Software layers involved
 - Kernel driver
 - EGL libs
 - Native interface
 - Java interface
 - Software GL implementation

4.1. What's OpenGL?

- It's just an API ... nothing but an API ...
- Check out Wikipedia
- Multiple versions out
- “ES” versions for embedded use
- Up to ES 3.2
- Android support up to ES 3.1

4.2. What's in a modern-day GPU?

- A tremendous amount of parallel processing units
- “SIMD”-like instruction set
- Video decoding/encoding capabilities
- ...

4.3. Software layers involved

- Kernel driver
- GL libraries
- Native GL API
- Java GL API

4.4. Kernel driver

PROPRIETARY

4.5. EGL libs

- /frameworks/base/native/opengl/libs
- Entry point: /system/lib/libEGL.so
- Looks for /system/lib/egl/egl.cfg
- /system/lib/egl - Mot Xoom:
 - egl.cfg
 - libEGL_perfhud.so
 - libEGL_tegra.so
 - libGLES_android.so
 - libGLESv1_CM_perfhud.so
 - libGLESv1_CM_tegra.so
 - libGLESv2_perfhud.so
 - libGLESv2_tegra.so
- elg.cfg:
 - 0 0 tegra

4.6. Native interface

- /frameworks/native/opengl/include
 - EGL
 - ETC1
 - GLES
 - GLES2
 - KHR

4.7. Java interface

- GL libs required by libandroid_runtime.so
- /frameworks/base/opengl/java/android/opengl:
 - ...

4.8. Software GL implementation

- `/frameworks/native/opengl/libagl`

5. Input Layer

- Kernel side - “std” Linux input layer:
 - `/dev/input/*`
- No HAL use
- Native lib:
 - `libinput`
 - `/frameworks/base/services/input`
- Input Manager Service:
 - `/frameworks/base/services/java/com/android/server/input`
 - Started and directly tied to Window Manager
- Specific config files (see source.android.com)
- Soft keyboard:
 - `/frameworks/base/core/java/android/inputmethodservice`
- Input methods:
 - `/packages/inputmehods`
 - <http://developer.android.com/guide/topics/text/creating-input-method.html>

6. Relevant Apps and Services

- Launcher
- StatusBar
- Wallpaper Manager Service
- Notification Service
- App Widgets

6.1. Launcher

- An app like any other
- See `/packages/app/Launcher2`

6.2. StatusBar

- A unique app
- See `/frameworks/base/packages/SystemUI`
- Connects to Status Bar Manager and gives an interface it can use to call back into Status Bar
- Can use `setIcon()` to display icons on the right
- Provides a CPU usage add-on that renders straight on rest of display using higher z-order

6.3. Wallpaper Manager Service

- See [/frameworks/base/services/java/com/android/server/WallpaperManagerService.java](#)

6.4. Notification Service

- Toasts
- Status bar notifications
- Gets handle to Status Bar Service at instantiation
- Uses handle to communicate with Status Bar

6.5. App Widgets

- See `/frameworks/base/services/java/com/android/server/AppWidgetService.java`

7. System Startup

- Kernel
- Init
- Boot animation
- Launcher

7.1. Boot animation

- Started by Surface Flinger
- “bootanim” binary
- /frameworks/base/cmds/bootanimation
- Relies on bootanimation.zip w/ PNGs (nothing but)
- See https://github.com/CyanogenMod/android_vendor_cm/tree/jellybean/prebuilt/common/bootanimation
- Must contain a desc.txt:
 - <width> <height> <fps>
 - p <count> <pause> <path>
 - p <count> <pause> <path>

8. References and Pointers

- “Use the source, Luke”
- Jim Huang's “Android Graphics”
- Benjamin Zores' “Linux Magazine / France” articles
- MIPS article on graphics internals:
<http://developer.mips.com/2012/04/11/learning-about-android-graphics-subsystem/>
- Stéphane Marchesin's “Linux Graphics Drivers: an Introduction”
<http://source.android.com/tech/input/index.html>

Running Code in Android

1. Programming Languages
2. Standard App Mechanisms
3. Special App Mechanisms
4. Starting Apps
5. Native Utilities and Daemons
6. Java Utilities (and Daemons)
7. System Services
8. Shell Scripts
9. init.rc Commands and Services
10. C Libraries
11. Java Libraries
12. SDK add-ons

1. Programming Languages

- Java:
 - Apps
 - Platform
- C:
 - Apps
 - Platform
- JavaScript / CSS / HTML
 - WebKit object
- C#:
 - Mono for Android
- Misc.
 - Any language for which there's a Linux compiler / interpreter

2. Standard App Mechanisms

- Components:
 - Activity
 - Service
 - Content Provider
 - Broadcast Receiver
- “Application” Component
- Widgets

2.1. “Application” Component

- See packages/apps/Phone:
 - Especially src/com/android/phone/PhoneApp.java

```
/**
 * Top-level Application class for the Phone app.
 */
public class PhoneApp extends Application {
    PhoneGlobals mPhoneGlobals;

    public PhoneApp() {
    }

    @Override
    public void onCreate() {
        if (UserHandle.myUserId() == 0) {
            // We are running as the primary user, so should bring up the
            // global phone state.
            mPhoneGlobals = new PhoneGlobals(this);
            mPhoneGlobals.onCreate();
        }
    }
}

...
```

2.2. Widgets

- See:

<https://developer.android.com/guide/topics/appwidgets>



3. Special App Mechanisms

- Foreground services
- Persistent apps
- Sync adapters
- Backup agents
- Input methods engines
- Alarm services
- Live wallpapers
- Account managers
- Device administrators
- “Core” app

3.1. Foreground services

- Sticky notification icon
- Ex:
 - Skype
 - Avast
- See:
 - <https://developer.android.com/guide/components/services.html#Foreground>
 - <https://developer.android.com/reference/android/app/Service.html#startForeground%28int,%20android.app.Notification%29>
 - https://developer.android.com/reference/android/app/Notification.html#FLAG_FOREGROUND_SERVICE

3.2. Persistent apps

- Flag in <application> decl. in manifest:
 `android:persistent="true"`
- For “system” apps only
- Will cause app to be kept alive by ActivityManager
- IOW:
 - It'll be automagically restarted if it dies
 - Lifecycle won't result in it dying

3.3. Sync adapters

- For sync'ing with a cloud service
- Typically for REST-based apps
- Ex.: a Twitter-like feed
- See:
 - <http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html>

3.4. Backup Agents

- Triggered by Backup Manager
- See:
 - <https://developer.android.com/guide/topics/data/backup.html>

3.5. Input methods Engines

- Virtual on-screen keyboards
- See:
 - <https://developer.android.com/guide/topics/text/creating-input-method.html>

3.6. Alarm services

- Cause Intent to trigger in the future:
- See:
 - <https://developer.android.com/reference/android/app/AlarmManager.html>

3.7. Live wallpapers

- Animated wallpapers
- See:
 - <http://www.vogella.com/articles/AndroidLiveWallpaper/article.html>

3.8. Account managers

- For managing accounts in Settings
- See:
 - <https://developer.android.com/reference/android/accounts/AccountManager.html>

3.9. Device administrators

- Enterprisification feature:
- See:
 - <https://developer.android.com/guide/topics/admin/device-admin.html>

3.10. “core” app

- Use this tag in <manifest> tag:
 `coreApp="true"`
- Used by:
 - StatusBar
 - SettingsProvider
 - See frameworks/base/packages/
- See “onlyCore” variable in SystemServer.java
- Causes only core apps to start if set to “true”
- Default hard-coded as “false”
- PackageManagerService.java has detail of how it's used

4. Starting Apps

- Intents
- Content Resolvers

5. Native Utilities and Daemons

- Compile as part of AOSP:
 `BUILD_EXECUTABLE`
- Build outside AOSP and merge:
 - Link statically (like `adbd`), or
 - Link dynamically (against Bionic or `glibc` or ...)
- Run like any regular Linux command-line tool
- Start from:
 - `adb shell`
 - `init.rc`
 - shell script
 - Android app (N.B. This will continue running independently)
- Plenty of existing examples: `netd`, `vold`, `installd`, etc.

5.1. Start from Android app

```
Process myUtil;  
BufferedReader myCliOutput;  
  
myUtil =  
    Runtime.getRuntime().exec("my_cli_util -P params");  
  
myCliOutput =  
    new BufferedReader(  
        new InputStreamReader(  
            myUtil.getInputStream()));
```

6. Java Utilities (and Daemons)

- Coded as a “library”
- Launched using `app_process` utility
- Use script to wrap call to `app_process`
- See `frameworks/base/cmds` for examples:
 - `am`
 - `pm`
 - `svc`
 - ...
- No known case of “daemon”, but should be feasible
 - Closest is System Server, but it's started from Zygote, not directly from `app_process`

7. System Services

- Start in:
 - Existing Java system services
 - App (like Phone or NFC)
 - C daemon (like Media Service or SurfaceFlinger)
- Register with Service Manager
- Provide API speak with system service:
 - In existing framework
 - As SDK add-on

8. Shell Scripts

- Android has built-in shell
- Since 4.0, MirBSD Korn Shell
- See: <https://www.mirbsd.org>
- Start shell script like any other native utility
- For example, `init.goldfish.rc` does:

```
service goldfish-setup /system/etc/init.goldfish.sh
  user root
  group root
  oneshot
```

9. init.rc Commands and Services

- Edit relevant `init.*.rc` to:
 - Add commands to existing actions
 - Create new actions based on global property triggers
 - Add new services
- See:
 - `system/core/root/init.rc`
 - `device/[vendor]/[product]/init.*.rc`

10. C Libraries

- Build libs as part of AOSP or using NDK:
`BUILD_SHARED_LIBRARY`
- Use library in native utility or daemon:
`LOCAL_SHARED_LIBRARIES := libfoo ...`
- Provide headers files as needed
- Can load library explicitly in Java, espc. for JNI
- See “HelloJNI” example in NDK for NDK ex.

```

public class HelloJni extends Activity
{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);

        /* Create a TextView and set its content.
         * the text is retrieved by calling a native
         * function.
         */
        TextView tv = new TextView(this);
        tv.setText( stringFromJNI() + " " + pid() );
        setContentView(tv);
    }

    /* A native method that is implemented by the
     * 'hello-jni' native library, which is packaged
     * with this application.
     */
    public native String  stringFromJNI();
    ...
    /* this is used to load the 'hello-jni' library on application
     * startup. The library has already been unpacked into
     * /data/data/com.example.HelloJni/lib/libhello-jni.so at
     * installation time by the package manager.
     */
    static {
        System.loadLibrary("hello-jni");
    }
}

```

11. Java Libraries

- Build as part of AOSP:
`BUILD_JAVA_LIBRARY`
- Will generate a .jar
- See:
 - <http://www.vogella.com/articles/AndroidLibraryProjects/article.html>
 - <http://stackoverflow.com/questions/3642928/adding-a-library-jar-to-an-eclipse-android-project>

12. SDK add-ons

- See device/sample/, espc.:
 - README
 - products/sample_addon.mk
 - sdk_addon/
 - frameworks/PlatformLibrary/
- Creates a ZIP file
- Extract in [sdk-dir]/add-ons/

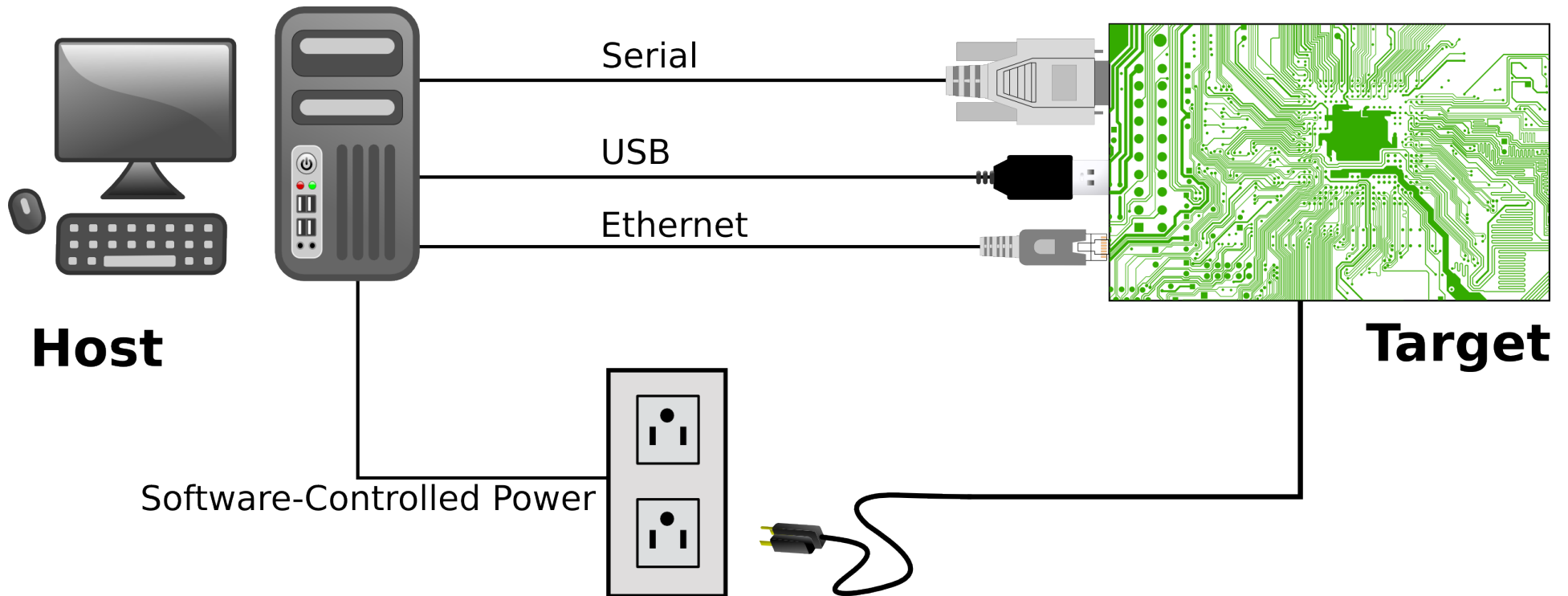
Debugging and Development

1. Development environment
2. Observing and monitoring
3. Interfacing with the framework
4. Working with the AOSP sources
5. Symbolic debugging
6. Detailed dynamic data collection
7. Benchmarking
8. Summing up

1. Development Environment

- Host / Target setup
- IDE / Editor
- Android Studio

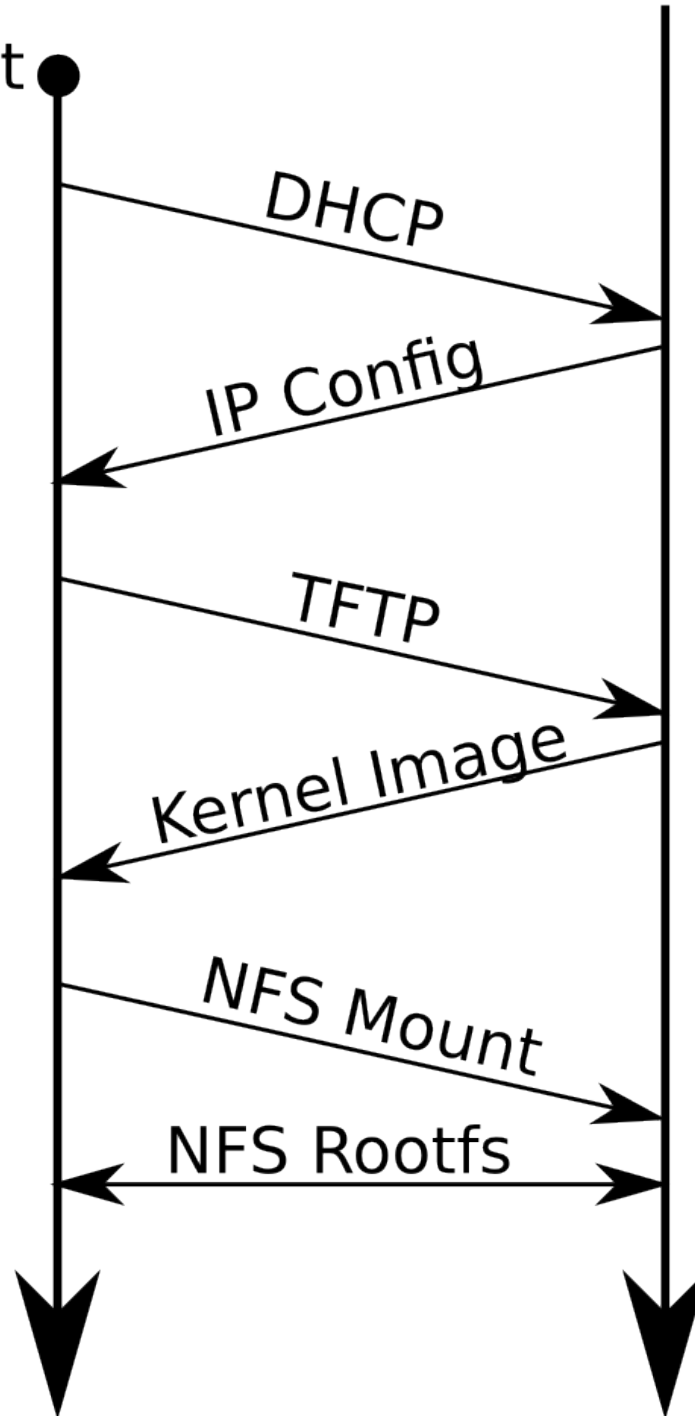
1.1. Host / Target setup



Target

Host

System Boot ●



1.2. IDE / Editor



1.3. Eclipse Setup

- Preparation
- Project importing
- AOSP fixups
- Browsing the sources

1.3.1. Preparation

- AOSP Basics:
 - Get AOSP ... from Google or otherwise
 - Extract if needed
 - Configure, build, etc.
- Eclipse / ADT:
 - Get ADT bundle from developer.android.com
 - Extract
 - Start and update and if needed

- Set up basic classpath file:

```
[aosp]$ cp development/ide/eclipse/.classpath .
```

- Adjust eclipse.ini

- On my ADT bundle, it's:

- adt-bundle-linux-x86_64-20130917/eclipse/eclipse.ini

- Change this:

- XX:MaxPermSize=256m

- Xms40m

- Xmx768m

- To this:

- XX:MaxPermSize=256m

- Xms128m

- Xmx768m

1.3.2. Project importing

- Start Eclipse
- Create new "Java project"
 - Project name = your AOSP name
 - Deselect "Use default location"
 - Location = path to your AOSP
 - Click "Next"
 - Wait a little bit ...
 - Click "Finish"
 - Wait for it to build your project
 - ... it likely will fail ...

1.3.3. AOSP fixups

- Need to fix AOSP classpath file and sources
- Assuming 4.3 here
- Add this:

```
<classpathentry kind="src" path="frameworks/opt/timezonepicker/src"/>  
<classpathentry kind="src" path="frameworks/opt/colorpicker/src"/>  
<classpathentry kind="src" path="frameworks/opt/datetimetypepicker/src"/>  
<classpathentry kind="src"  
path="frameworks/support/v8/renderscript/java/src"/>
```

- Remove this:

```
<classpathentry kind="src"  
path="frameworks/support/renderscript/v8/java/src"/>
```


- Comment out a couple of things:

```
<!-- Redefines android.util.pools which confuses Eclipse
<classpathentry kind="src" path="packages/apps/Gallery2/src"/>
<classpathentry kind="src" path="packages/apps/Gallery2/src_pd"/>
<classpathentry kind="src"
path="packages/apps/Gallery2/gallerycommon/src"/>
-->
<!--
<classpathentry kind="src" path="packages/apps/Nfc/src"/>
<classpathentry kind="src" path="packages/apps/Nfc/nci/src"/>
-->
<!--
<classpathentry kind="src" path="frameworks/ex/carousel/java"/>
-->
```

- Manually build the following (cd to and "mm") -- or remove from .classpath:

```
packages/apps/Stk  
packages/screensavers/WebView  
development/samples/ApiDemos  
development/samples/HelloActivity  
development/samples/Home  
development/samples/LunarLander  
development/samples/NotePad  
development/samples/RSSReader  
development/samples/SkeletonApp  
development/samples/Snake
```

- Edit
packages/apps/Launcher2/src/com/android/launcher
2/DragLayer.java and modify:

```
private boolean isLayoutRtl() {
```

- to

```
public boolean isLayoutRtl() {
```

- **Now: right-click on project and select "Refresh"**
- It might still show "x" on some parts until it's done rebuilding the project

1.3.4. Browsing the sources

- Mouse-over object type to be taken to declaration
- Browse classes through “Outline”
- Browse Call Hierarchy
- View recently viewed files (Ctrl-e)
- Many other shortcuts, see:
 - <http://source.android.com/source/using-eclipse.html>
- Issues:
 - Can't compile with Eclipse ... still need “make”
 - For Java only

2. Observing and Monitoring

- Native
- Framework
- Overall

2.1. Native

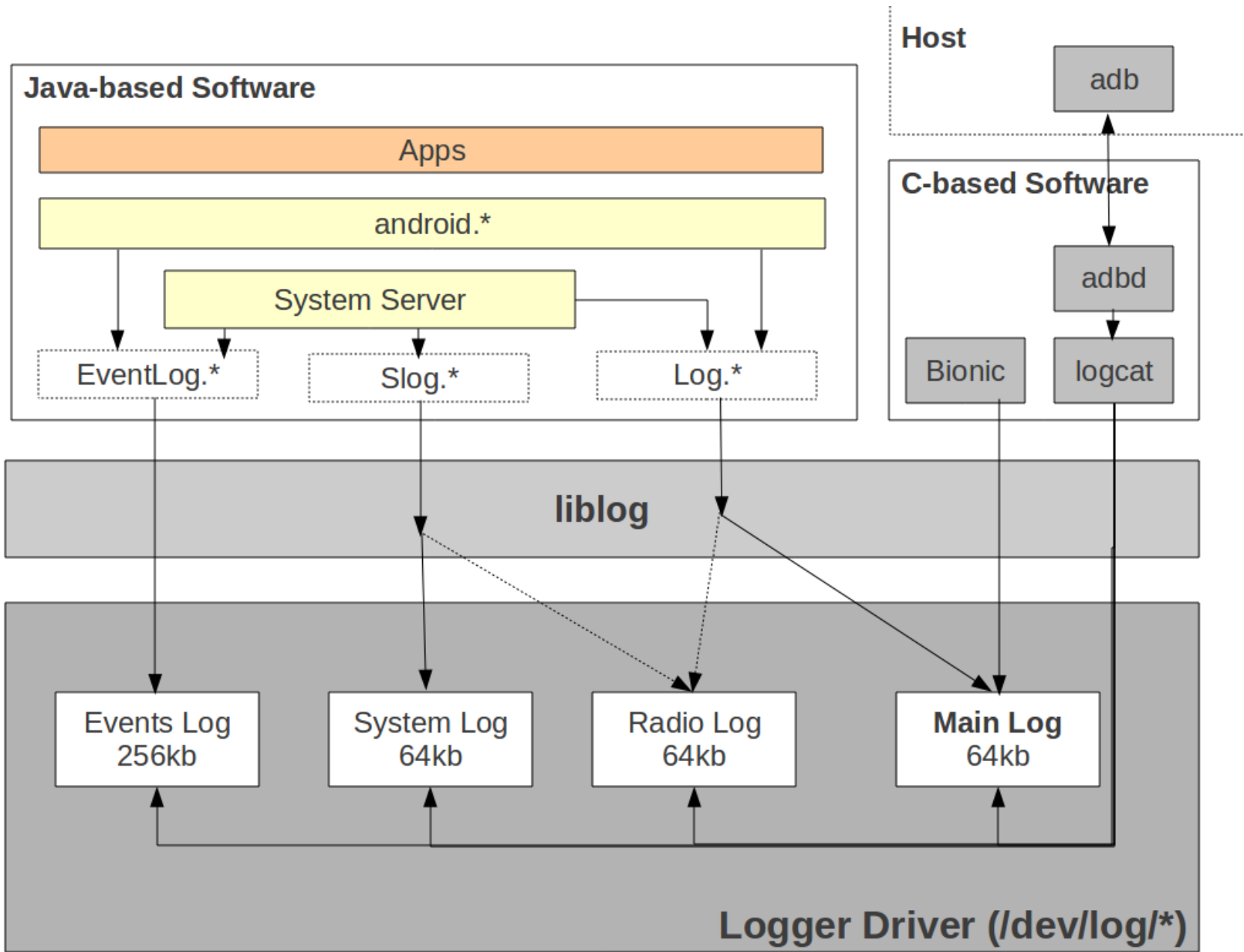
- schedtop
- librank
- procmem
- procrank
- showmap
- latencytop

2.2. Framework

- dumpsys
- service

2.3 Overall

- logcat
- dumpstate / bugreport
- watchprop / getprop



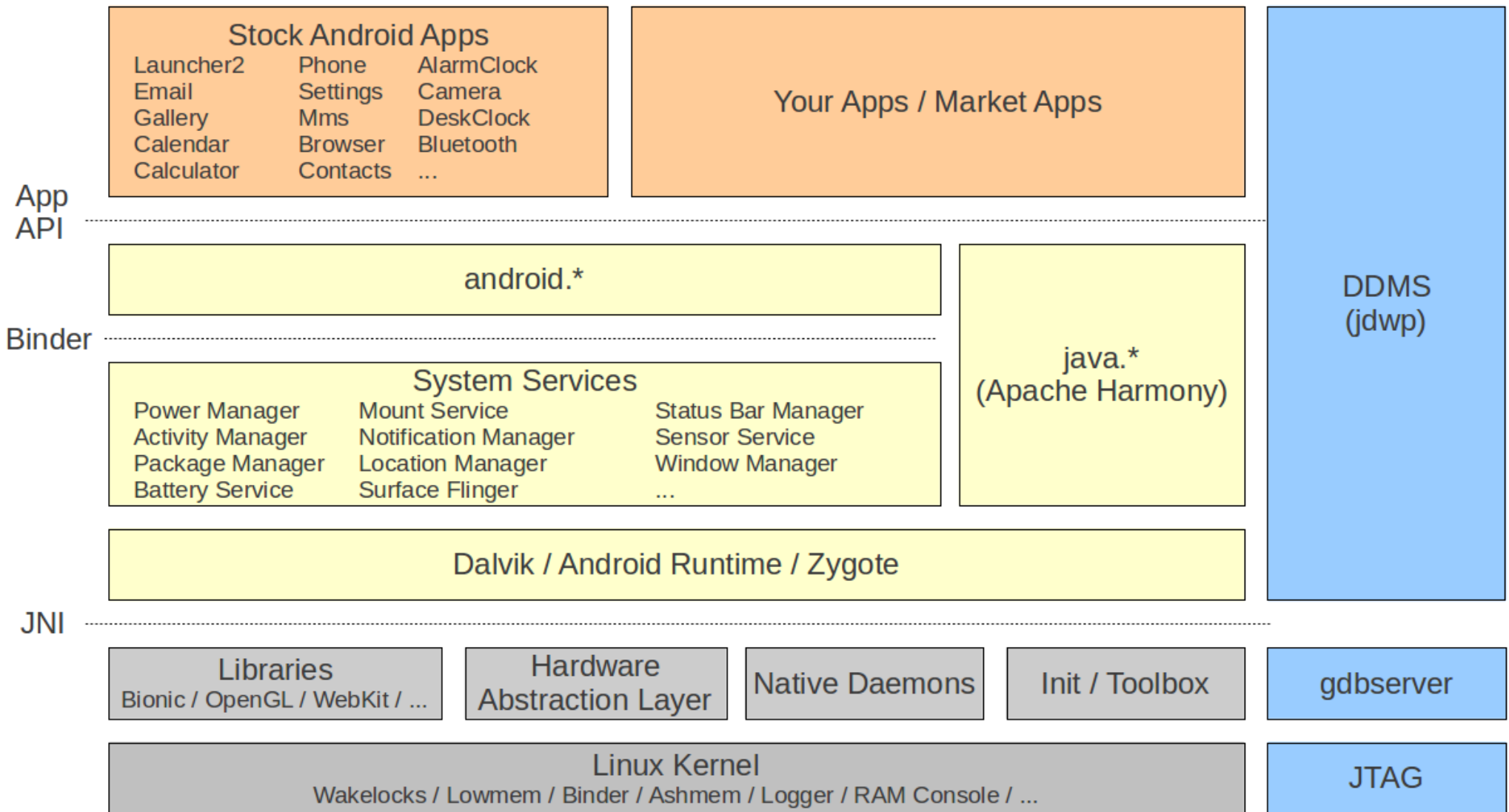
3. Interfacing With the Framework

- start / stop
- service call
- am
- pm
- wm
- svc
- monkey
- setprop

4. Working with the AOSP Sources

- You really need to check `build/envsetup.sh`
- Some tricks:
 - `godir`
 - `croot`
 - `mm`
 - `m`
 - `jgrep`
 - `cgrep`
 - `resgrep`
- It takes time to wrap your head around the tree

5. Symbolic Debugging - basics



5.1. DDMS / Eclipse integration

- Start DDMS:
 - The one from the AOSP's command-line
 - Not the one from Eclipse (“connection refuse”)
- It takes some time to load -- a few **minutes**
- Each process has a separate host-side socket
- Select the process you want to debug:
 - It'll get port 8700

- Go to Eclipse:
 - Run->Debug Configurations->Remote Java Application
 - Connection Type: "Standard (Socket Attach)"
 - Host: localhost
 - Port: 8700


5.2. Starting debug w/ Eclipse

- Order is finicky:
 - Start your device or emulator
 - Start command-line DDMS before Eclipse otherwise you'll get this in logcat:

```
"I/jdwp ( 411): Ignoring second debugger -- accepting and dropping"
```
 - Start Eclipse
 - Eclipse will complain that there's already a DDMS running. Ignore that.

- ✓ • StatusBarManagerService [line: 123] - expa
- ✓ • StatusBarManagerService [line: 134] - colla

ddms [Close]

 Could not open Selected VM debug port (8700). Make sure you do not have another instance of DDMS or of the eclipse plugin running. If it's being used by something else, choose a new port number in the preferences.

[OK]

e.java
outor
icked.
[View v) {

5.3. Debugging

- Select the process you want to debug in DDMS
- Go into Eclipse and click on the debug configuration you created earlier
- Check that the little green bug is beside your process in DDMS
- Again, things can look like they're freezing, this is "normal" for Eclipse ...
- Wait for Eclipse to show your Dalvik process in the "Debug" *window* in the "Debug" *view* -- all threads should show

Dalvik Debug Monitor

File Edit Actions Device

Name

<build> [emulator-5554]

- system_process
- com.android.providers.calendar
- com.android.smspush
- com.android.inputmethod.latin
- com.android.phone
- com.android.musicfx
- com.android.launcher
- android.process.media
- com.android.systemui
- com.android.mms

Saved Filters + -

All messages (no filters)

Dalvik Debug Monitor

File Edit Actions Device

Name	State	Memory	Private Memory
<build> [emulator-5554]	Online		<build> [4.3, debug]
system_process	275	8600 / 8700	
com.android.providers.calendar	609	8601	
com.android.smspush	444	8602	
com.android.inputmethod.latin	371	8603	
com.android.phone	396	8604	
com.android.musicfx	814	8605	
com.android.launcher	407	8606	
android.process.media	498	8607	
com.android.systemui	344	8608	
com.android.mms	672	8609	

5.4. Debugging multiple processes

- In the debug **view** of eclipse, click on "Debug" for every time you change the process in DDMS
- Wait for that process' threads to load in the debug view
- Once threads are loaded, you can actually start debugging

5.5. gdbserver - target side

- First, you'll need to make sure your C code is compiled appropriately. Add this to Android.mk:

```
LOCAL_CFLAGS += -ggdb  
LOCAL_STRIP_MODULE = false
```

- Attaching to running process

```
# gdbserver --attach localhost:2345 30
```

- Start app for debugging with gdbserver prepended

```
# gdbserver localhost:2345 service list
```

- Forward the port on the host:

```
$ adb forward tcp:2345 tcp:2345
```

5.6. gdb - host side

- Load file ****FIRST**** and then attach on host side

```
$ prebuilts/gcc/linux-x86/arm/arm-eabi-4.7/bin/arm-eabi-gdb
GNU gdb (GDB) 7.3.1-gg2
Copyright (C) 2011 Free Software Foundation, Inc.
...
(gdb) file out/target/product/generic/system/bin/service
(gdb) target remote localhost:2345
(gdb) b main
Cannot access memory at address 0x0
Breakpoint 1 at 0x2a00146c: file frameworks/native/cmds/service/service.cpp, line 59.
(gdb) cont
Continuing.
warning: Could not load shared library symbols for 11 libraries, e.g. /system/bin/linker.
...

Breakpoint 1, main (argc=2, argv=0xbe882b74) at frameworks/native/cmds/service/service.cpp:59
59 {
(gdb) n
60     sp<IServiceManager> sm = defaultServiceManager();
(gdb) n
59 {
(gdb) n
60     sp<IServiceManager> sm = defaultServiceManager();
(gdb) n
61     fflush(stdout);
```

5.7. Multi-threaded = #FAIL

```
$ prebuilts/gcc/linux-x86/arm/arm-eabi-4.7/bin/arm-eabi-gdb
GNU gdb (GDB) 7.3.1-gg2
...
(gdb) add-symbol-file out/target/product/generic/system/lib/libandroid_servers.so 0x4AFFC8B8
add symbol table from file "out/target/product/generic/system/lib/libandroid_servers.so" at
.text_addr = 0x4affc8b8
(y or n) y
(gdb) add-symbol-file out/target/product/generic/system/lib/libc.so 0x400339B8
add symbol table from file "out/target/product/generic/system/lib/libc.so" at
.text_addr = 0x400339b8
(y or n) y
(gdb) target remote localhost:2345
Remote debugging using localhost:2345
__ioctl () at bionic/libc/arch-arm/syscalls/__ioctl.S:10
10      mov      r7, ip
(gdb) b com_android_server_power_PowerManagerService.cpp:162
Breakpoint 1 at 0x4b000a34: file
frameworks/base/services/jni/com_android_server_power_PowerManagerService.cpp, line 162.
(gdb) cont
Continuing.
```

```
Program terminated with signal SIGTRAP, Trace/breakpoint trap.
The program no longer exists.
```

```
...
```

- Even if you try attaching to the specific thread in the system server running the system service you're trying to instrument, you'll get the same issue.
- Probably requires rebuilding gdbserver with thread support:
 - <https://sourceware.org/ml/gdb/2009-01/msg00084.html>
 - <http://code.google.com/p/android/issues/detail?id=9713>
- Issues seems to be solved in NDK but not in gdbserver in AOSP:
 - <http://comments.gmane.org/gmane.comp.handhelds.android.ndk/12122>

5.8. How to know what's the address of the library

```
# cat /proc/[system_sever PID]/maps | grep android_servers  
4aff5000-4b007000 r-xp 00000000 1f:00 519 /system/lib/libandroid_servers.so  
4b007000-4b008000 r--p 00011000 1f:00 519 /system/lib/libandroid_servers.so  
4b008000-4b009000 rw-p 00012000 1f:00 519 /system/lib/libandroid_servers.so
```

```
# cat /proc/[system_sever PID]/maps | grep libc.so  
40027000-4006c000 r-xp 00000000 1f:00 601 /system/lib/libc.so  
4006d000-4006f000 r--p 00045000 1f:00 601 /system/lib/libc.so  
4006f000-40071000 rw-p 00047000 1f:00 601 /system/lib/libc.so
```

```
$ objdump -h out/target/product/generic/system/lib/libandroid_servers.so | grep text  
7 .text 00006ae8 000078b8 000078b8 000078b8 2**3  
$ objdump -h out/target/product/generic/system/lib/libc.so | grep text  
7 .text 0002f080 0000c9b8 0000c9b8 0000c9b8 2**3
```

```
4aff5000 + 000078b8 = 0x4AFFC8B8  
40027000 + 0000c9b8 = 0x400339B8
```

<http://linux-mobile-hacker.blogspot.co.uk/2008/02/debug-shared-library-with-gdbserver.html>

5.9. JTAG

- Requires hardware device
- Sometimes interfaces with gdb
- Not Android specific
- Some allow transparent kernel/user-space debug
- Don't know of any that go all the way up to ART

6. Detailed Dynamic Data Collection

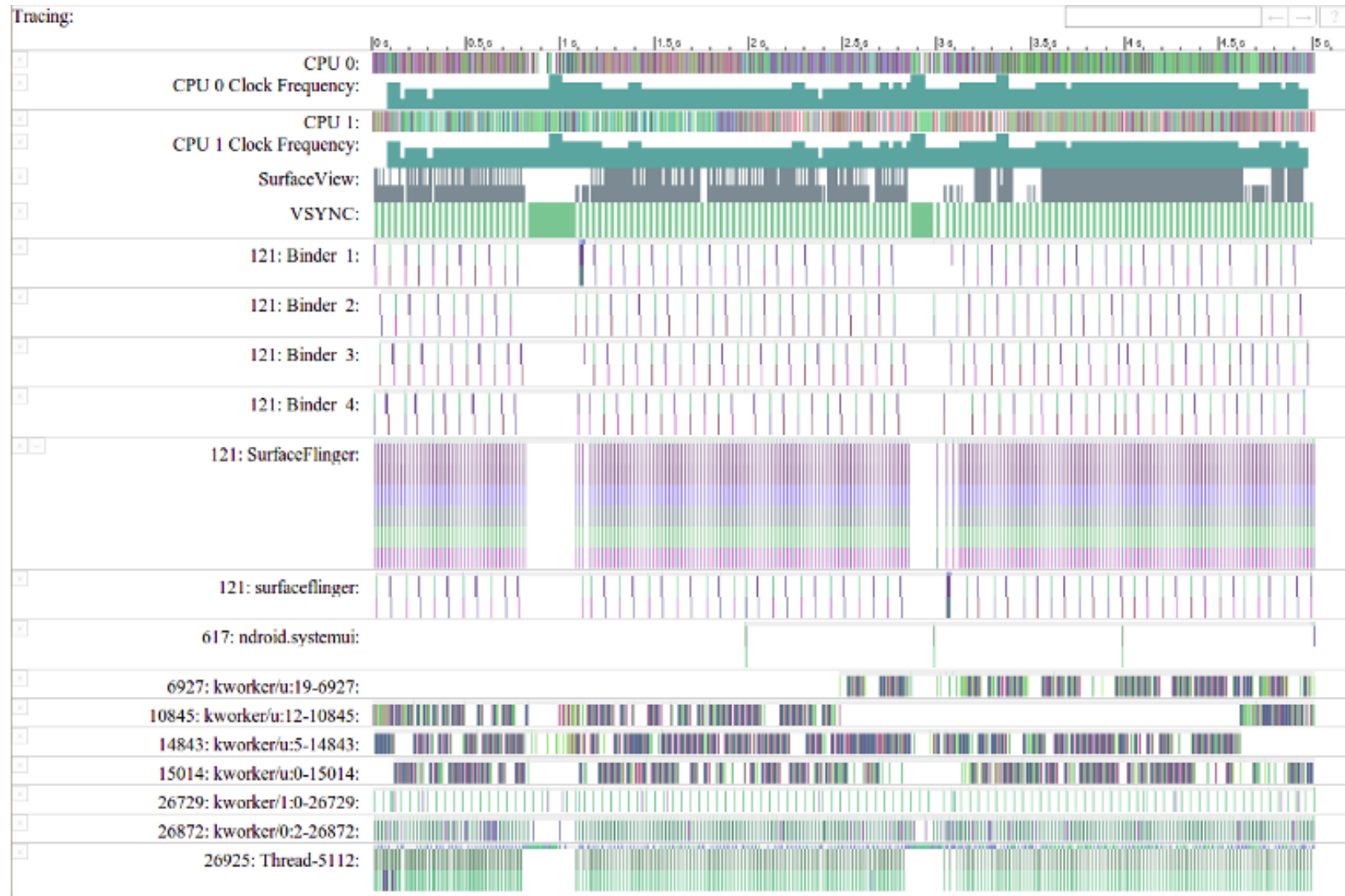
- Logging
- ftrace
- perf

6.1. Logging

- logcat is the most rapid/consistent way to observe dynamic behavior.
- Trivial to add instrumentation points
- **It just works ...**

6.2. ftrace

- With 4.1, Google introduced systrace/atrace
- systrace is a Python script running on host side
- atrace is native Android binary
- systrace calls atrace via ADB
- atrace uses ftrace to capture kernel events
- Stack instrumented to feed events to ftrace
- Google's doc:
 - <https://developer.android.com/tools/help/systrace.html>
 - <https://developer.android.com/tools/debugging/systrace.html>



... trouble is ...

- I can't get it to work !*!@#\$&!#*\$!
- Default goldfish kernel doesn't have ftrace
- Able to build ftrace-enabled kernel for goldfish
- Can trace that system ... so long as I don't use atrace/systrace ... WTF¹?
- Not all Android kernels have ftrace enabled
- Generates HTML file that can only be read by Chrome ... **it doesn't work in Firefox**. NIH?

1: The AOSP sources define WTF as “What a Terrible Failure”. We trust they've done their research.

... still ...

- Have a look at these files:
 - `/external/chromium-trace/systrace.py`
 - `/frameworks/native/cmds/atrace`
 - `/frameworks/base/core/java/android/os/Trace.java`
 - `/erameworks/native/include/utils/Trace.h`
 - `/system/core/include/cutils/trace.h`
 - `/frameworks/native/libs/utils/Trace.cpp`
- Look for:
 - `ATRACE*` in c/cpp files
 - `Trace.traceBegin()/trace.traceEnd()` in Java files

atrace --help

usage: atrace [options] [categories...]

options include:

- a appname enable app-level tracing for a comma separated list of cmdlines
- b N use a trace buffer size of N KB
- c trace into a circular buffer
- k fname,... trace the listed kernel functions
- n ignore signals
- s N sleep for N seconds before tracing [default 0]
- t N trace for N seconds [default 5]
- z compress the trace dump
- async_start start circular trace and return immediatly
- async_dump dump the current contents of circular trace buffer
- async_stop stop tracing and dump the current contents of circular trace buffer
- list_categories list the available tracing categories


```
# atrace --list_categories
    gfx - Graphics
    input - Input
    view - View System
webview - WebView
    wm - Window Manager
    am - Activity Manager
    audio - Audio
    video - Video
camera - Camera
    hal - Hardware Modules
    res - Resource Loading
dalvik - Dalvik VM
```

6.3. perf on Android on ARM



7. Benchmarking

Oxbench

AnTuTu

Passmark

Vellamo

Geekbench2

SunSpider

GLBenchmakr

Quadrant Standard Edition

Linpack

Neocore

3DMark

Epic Citadel

Androbench

CF-bench

SD Tools

RL Benchmark: SQL

Benchmark & Tunning

A1 SD Bench

Quick Benchmark Lite

3DRating benchmark

Smartbench 2011

NenaMark

Rightware Browsermark

An3DBenchXL

CaffeineMark

NBench

Methanol

AndEBench

SmartBench 2012

RealPi

8. Summing Up



- Works relatively well:
 - logcat
 - Eclipse / DDMS
 - Framework tools
- Works ok:
 - gdb/gdbserver
 - native tools
 - ftrace
- Doesn't work (for me):
 - systrace/atrace
 - perf

9. Loose ends

- strace
- debuggerd
- tombstones
- anr traces

Kernel internals

1. Brief history
2. Features
3. General architecture
4. Source layout
5. Process management
6. Filesystems
7. Memory management
8. Communication facilities and interfacing
9. Loadable modules

10. Interrupt and exception management

11. Timing

12. Locking primitives

13. Kernel startup

1. Brief history

- Started on Minix by Linus Torvalds (who wanted to call it Freax ...)
- Rechristened “Linux” by FTP site admin
- Once public, users and contributions grew at a very rapid rate.
- Closed the GNU project's loop
- Most important events:
 - 17 September 1991: Version 0.01
 - 3 December 1991: Version 0.10
 - 8 March 1992: Version 0.95

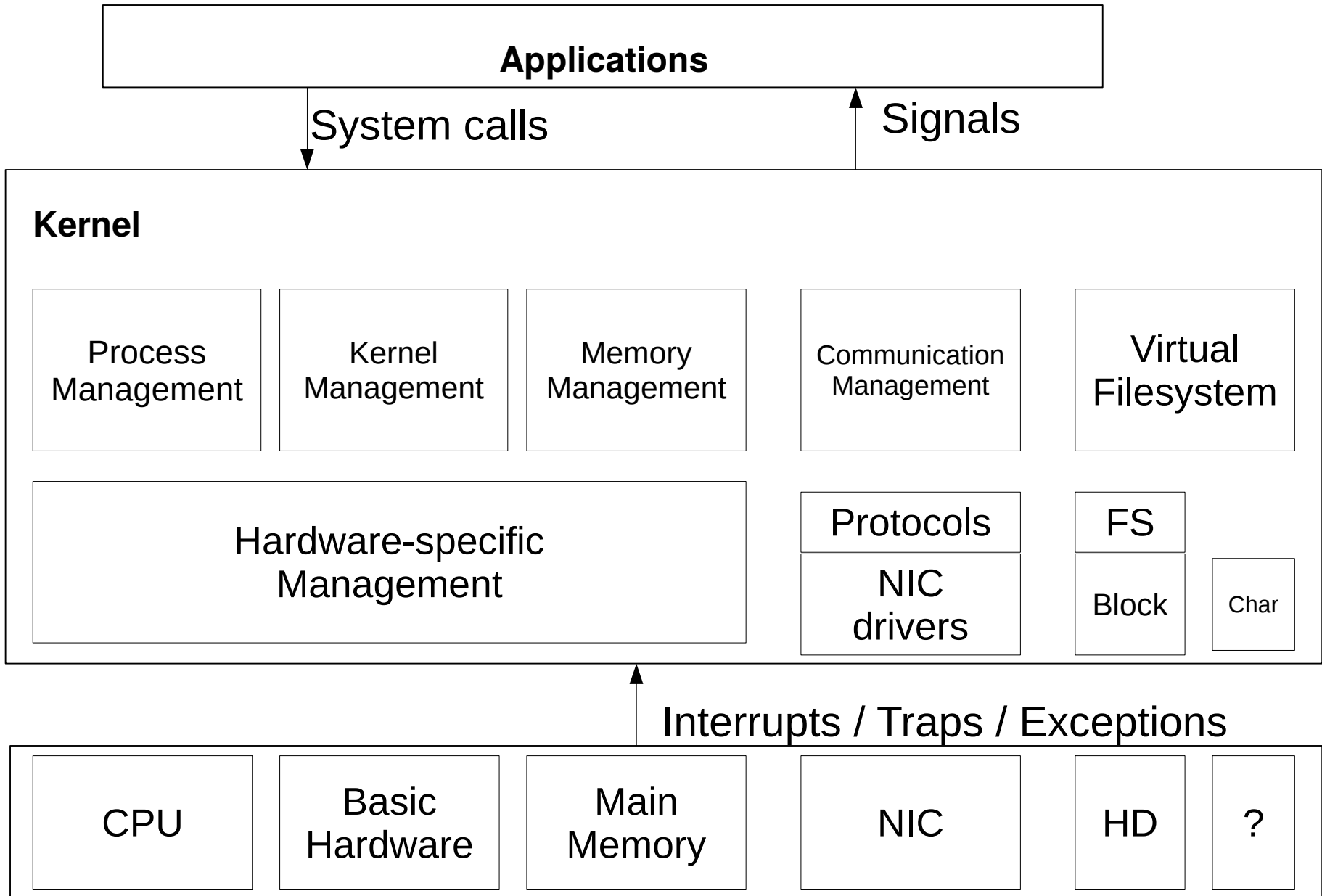
- 13 December 1992: Version 0.99
- 13 March 1994: Version 1.0
- 7 March 1995: Version 1.2.0
- 9 July 1996: Version 2.0
- 26 January 1999: Version 2.2.0
- 4 January 2001 : Version 2.4.0
- 17 December 2003: Version 2.6.0
- 21 July 2011: Version 3.0.0
- 12 April 2015: Version 4.0.0

2. Features

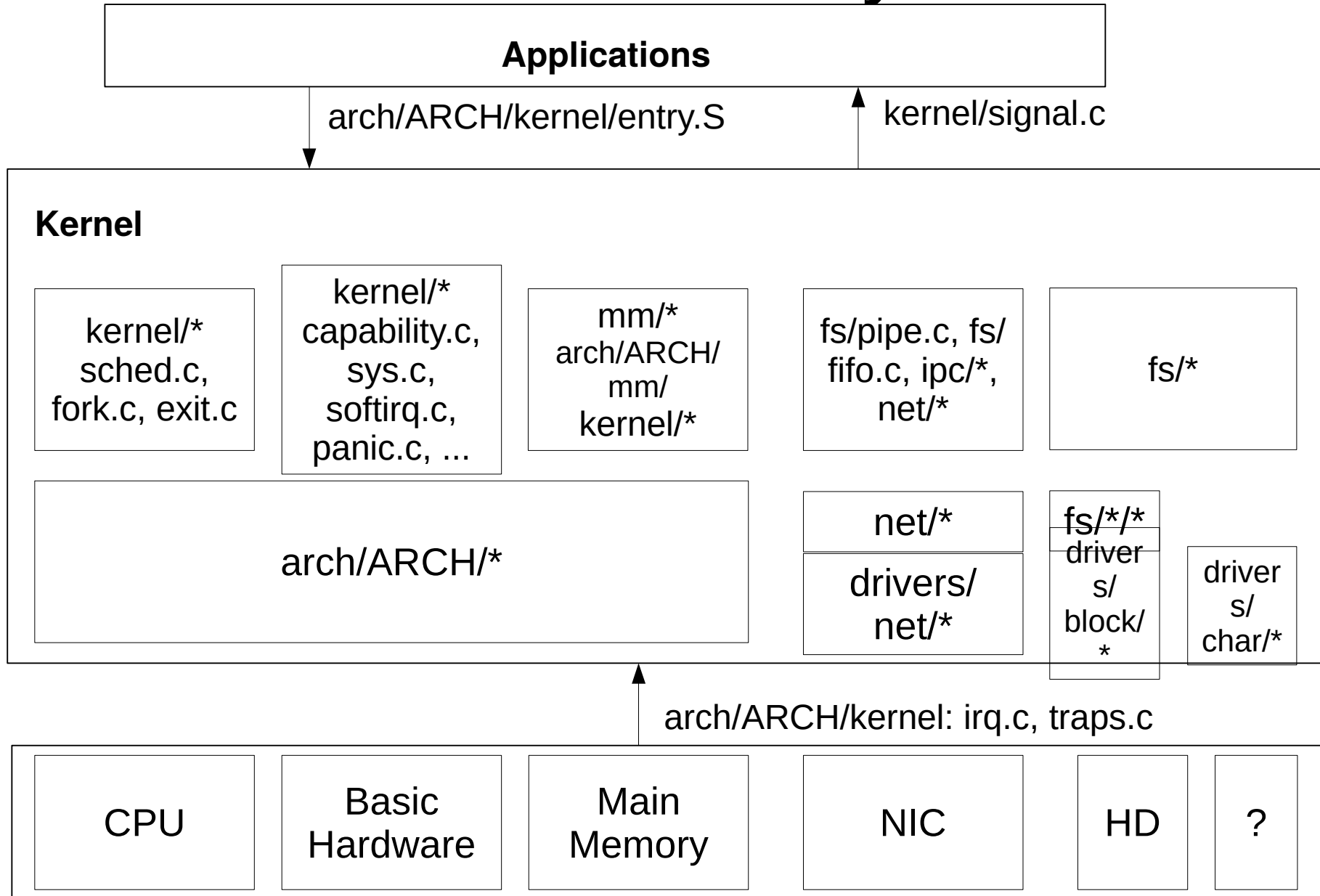
- Portable / Architecture-independent
- Scalable
- Monolithic
- Dynamically extensible (modules)
- Multi-user environment
- Multi-process / Multi-threading
- Memory protection
- Preemptable (starting in 2.5.x) ... but not real-time
- Symmetric multi-processor

- Slew of filesystems
- Slew of networking protocols / NICs
- Quite a few executable formats
- ...

3. General architecture



4. Source layout



arch	112MB =>	architecture-dependent functionality
block	600KB =>	block layer
Documentation	17MB =>	main kernel documentation
drivers	231MB=>	all drivers
fs	31MB =>	virtual filesystem and all fs types
include	20MB =>	complete kernel headers
init	150KB =>	kernel startup code
ipc	224KB =>	System V IPC
kernel	4.7MB =>	core kernel code
mm	2.2MB =>	memory management
net	20MB =>	networking core and protocols
scripts	1.1MB =>	scripts used to build kernel
tools	2.1MB =>	misc. kernel-related tools

- arch/

2.4M alpha

29M arm

1.4M avr32

5.3M blackfin

4.9M cris

1.4M frv

856K h8300

4.6M ia64

8.0K Kconfig

1.4M m32r

5.7M m68k

1.1M m68knommu

1.2M microblaze

11M mips

1.7M mn10300

2.4M parisc

13M powerpc

2.4M s390

636K score

5.4M sh

4.7M sparc

1.9M tile

1.9M um

8.5M x86

1.4M xtensa

- arch/powerpc:

2.0M boot

548K configs

2.0M include

2.1M kernel

384K kvm

216K lib

204K math-emu

472K mm

156K oprofile

3.4M platforms

848K sysdev

392K xmon

- drivers/

accessibility	cpufreq	hwmon	mca	parisc	sbus	uio
acpi	cpuidle	i2c	md	parport	scsi	usb
amba	crypto	ide	media	pci	serial	uwb
ata	dca	idle	memstick	pcmcia	sfi	vhost
atm	dio	ieee802154	message	platform	sh	video
auxdisplay	dma	infiniband	mfd	pnp	sn	virtio
base	edac	input	misc	power	spi	vlynq
block	eisa	isdn	mmc	pps	ssb	w1
bluetooth	firewire	Kconfig	mtd	ps3	staging	watchdog
cdrom	firmware	leds	net	rapidio	tc	xen
char	gpio	lguest	nubus	regulator	telephony	zorro
clocksource	gpu	macintosh	of	rtc	thermal	
connector	hid	Makefile	oprofile	s390	tty	

- **include/**

```
acpi          config  drm      keys    math-emu  mtd    pcmcia  rxrpc  sound  video
asm-generic  crypto  Kbuild  linux  media     net    rdma    scsi   trace  xen
```

- Looking for something:
 - Try `grep`
 - Have a look at the Linux Cross-Referencing project:
 - URL: <http://lxr.linux.no/>
 - Code: <http://lxr.sourceforge.net/>
 - Advanced kernel searching/understanding:
 - CScope: <http://cscope.sourceforge.net/>
 - KScope front-end: <http://kscope.sourceforge.net/>
 - ETAGS (emacs)

5. Process management

- Process descriptor (include/linux/sched.h: task_struct):

Process state	=>	<i>state</i>
Identification	=>	<i>pid, tgid</i>
Relationship	=>	<i>*parent, children</i>
Scheduling	=>	<i>time_slice, sched_class,</i> <i>rt_priority, prio, static_prio</i>
Files	=>	<i>*files</i>
Memory	=>	<i>*mm</i>

- Main process list:
 - Doubly-linked list
 - List head: *init_task*
- Currently running process (macro): *current*
- Task creation:
`kernel/fork.c:do_fork()`
- Threads:
`arch/ARCH/kernel/process.c:sys_clone()`
- *sys_clone()* ends up calling *do_fork()*

- Scheduling:

- Process states: TASK_RUNNING, TASK_INTERRUPTIBLE, TASK_UNINTERRUPTIBLE, TASK_STOPPED, TASK_TRACED, EXIT_ZOMBIE, ...

- Main scheduling function:

kernel/sched/core.c: *schedule()*

- Scheduling policies:

- SCHED_OTHER

- Main scheduling policy for Linux processes

- SCHED_FIFO:

- Process has CPU until it gives it up or no other higher priority task comes along.

- SCHED_RR:

- CPU is shared between “real-time” tasks

- ...

- Address space:

- `include/linux/mm_types.h:mm_struct`
- Fields: **mmap, mm_count, start_code, end_code, start_data, end_data, start_brk, brk, start_stack, arg_start, arg_end, env_start, env_end*

- Special tasks:

- `init` => first process on system
- `ksoftirqd/0` => soft-irq thread for avoiding process starvation
- `events/0` => kernel's work queue handler (was `keventd`)
- `khubd` => USB hub thread
- `kswapd0` => pageout daemon

6. Filesystems

- Virtual filesystem: `fs/*`

<code>attr.c</code>	<code>=></code>	file attributes
<code>block_dev.c</code>	<code>=></code>	block device access
<code>buffer.c</code>	<code>=></code>	buffer cache
<code>char_dev.c</code>	<code>=></code>	char device access
<code>dcache.c</code>	<code>=></code>	dentry (directory entry) cache
<code>notify/</code>	<code>=></code>	directory change notifications
<code>quot/</code>	<code>=></code>	disk quota
<code>exec.c</code>	<code>=></code>	<code>exec()</code> and its variants
<code>fcntl.c</code>	<code>=></code>	<code>fcntl()</code>
<code>fifo.c</code>	<code>=></code>	FIFO handling
<code>file.c</code>	<code>=></code>	manage process' fd array
<code>file_table.c</code>	<code>=></code>	file table manipulation
<code>inode.c</code>	<code>=></code>	inode handling
<code>ioctl.c</code>	<code>=></code>	<code>ioctl()</code>
<code>locks.c</code>	<code>=></code>	file locking

namei.c	=>	pathname lookup
namespace.c	=>	filesystem mounting
open.c	=>	<i>open()</i>
pipe.c	=>	pipe management
readdir.c	=>	directory reading
read_write.c	=>	<i>read()</i> and <i>write()</i> , and variants
select.c	=>	<i>select()</i> and <i>poll()</i>
stat.c	=>	<i>stat()</i>
super.c	=>	filesystem type management

- Root filesystem mounting: `init/do_mounts.c`

prepare_namespace()

- RAM disk handling (within `init/`):

- `do_mounts_initrd.c`: *initrd_load()*
- `do_mounts_rd.c`: *rd_load_image()*, *identify_ramdisk_image()*
- `do_mounts_initrd.c`: *handle_initrd()*,

7. Memory management

- Arch-independent portion: mm/*

bootmem.c	=>	boot memory allocation / handling
filemap.c	=>	handling for <i>mmap()</i> 'ed files
highmem.c	=>	RAM above 896MB / up to 64 GB
memory.c	=>	page and page table manipulation
mlock.c	=>	memory region locking
mmap.c	=>	<i>mmap()</i>
mprotect.c	=>	memory protection mechanisms
mremap.c	=>	<i>mremap()</i>
<i>nommu.c (2.5)</i>	=>	<i>functions for MMU-less processors</i>
oom_kill.c	=>	process killing when short on memory
page_alloc.c	=>	page allocation / freeing
page_io.c	=>	reading / writing swap pages
shmem.c	=>	shared memory management
slab.c	=>	memory allocation for kernel
swap.c	=>	swap default

swapfile.c	=>	swap space management
swap_state.c	=>	swap page caching
vmalloc.c	=>	memory region allocation
vmscan.c	=>	page out daemon

- **Architecture-dependent portion: arch/ARCH/mm/***

fault.c	all =>	page fault handler
init.c	all =>	memory initialization
ioremap.c	x86=>	remapping of I/O range to kernel space
pageattr.c	x86=>	page attributes handling
pgtable.c	ppc=>	page table manipulation
ppc_mmu.c	ppc=>	MMU handling for PPC
tlb.c	ppc=>	TLB flushing

8. Communication facilities and interfacing

- Signals => kernel/signal.c
- Pipes => fs/pipe.c
- FIFOs => fs/fifo.c
- Sockets => net/socket.c
- System V IPC => ipc/ : msg.c, sem.c, shm.c
- System calls => arch/ARCH/kernel/entry.S
- Adding new system calls:
 1. Add entry to arch/ARCH/kernel/entry.S
 2. Add entry to arch/ARCH/include/unistd.h
 3. Add your function to the kernel's code

9. Loadable modules

- Allow dynamic loading and unloading of additional kernel functionality.
- Managed by: kernel/module.c
- In practice, modules are .ko files (.o prior to 2.6)
- A single module's source tree can have a very complex hierarchy.
- Every module must export functions using:

module_init() => called on insmod

module_exit() => called on rmmod

- Module macros:

MODULE_AUTHOR => module's author
MODULE_LICENSE => module's license (taint)
MODULE_DESCRIPTION => module's description
EXPORT_SYMBOL => export symbol for use by other
 modules
MODULE_PARM_DESC => module parameter description

...

- Requesting modules from within the kernel:

*int request_module(const char * name, ...)*

10. Interrupt and exception management

- Relevant files:

- arch/x86/kernel/

- i8259.c => initialization and handling of 8259

- irq.c => arch-dependent irq handling

- entry.S => main assembly entry point

- traps.c => CPU exception handling

- kernel/irq/

- handle.c => arch-independent irq handling (exc. ARM)

- Exception examples:

- debug, overflow, bounds, fp, fault, nmi, etc.

- Exceptions path:

1. arch/x86/kernel/entry.S: hard-coded assembly

2. arch/x86/kernel/traps.c: custom function or macro generated (DO_ERROR).

- Hardware interrupt code:
 - arch/x86/kernel/entry.S: irq_entries_start
- Hardware interrupt path:
 1. Assembly in entry.S
 2. *do_IRQ()* in arch/x86/kernel/irq.c
 3. *__do_IRQ()* in kernel/irq/handle.c:
 1. Acks IRQ using callback from arch/x86/kernel/i8259.c:*mask_and_ack_8259A()*
 2. Calls *handle_IRQ_event()*
 4. Handler provided by device driver is invoked

- Deferring handling with enabled interrupts:
 - Softirq:
 - Statically allocated
 - Reentrant (must use locking mechanisms to protect data)
 - Softirqs of same type can run on many CPUs in the same time.
 - Tasklets:
 - Built on top of softirqs
 - Dynamically allocatable
 - The same tasklet type can't run on 2 CPUs in the same time.
 - Bottom-halves:
 - Built on top of tasklets
 - Statically allocated
 - There can only be one BH running at one time in the entire system.

11. Timing

- Time-keeping variables in the kernel: x86

jiffies => incremented at every clock tick

wall_jiffies => last time xtime was updated

TSC => CPU-maintained counter (64-bit)

- Time handling:

- `arch/x86/kernel/time.c`

- Finding out what time it is:

- `do_gettimeofday()`
- `get_cycles()`
- *jiffies*

12. Locking primitives

- Never do cli / sti
- Always use existing locks to do the dirty-work
- Variants:
 - Spinlocks with IRQ disabling: good for int handlers

```
spin_lock_irqsave(&lock_var, cpu_flags);
```

```
....
```

```
spin_unlock_irqrestore(&lock_var, cpu_flags);
```

- Vanilla spinlocks: good for most code not in int

```
spin_lock(&lock_var);
```

```
....
```

```
spin_unlock(&lock_var);
```

- Read/write locks:

- In reader:

```
read_lock_irqsave(&lock_var, cpu_flags);
```

```
....
```

```
read_unlock_irqrestore(&lock_var, cpu_flags);
```

- In writer:

```
write_lock_irqsave(&lock_var, cpu_flags);
```

```
....
```

```
write_unlock_irqrestore(&lock_var, cpu_flags);
```

- For a complete list, have a look at `include/linux/spinlock.h`.

13. Kernel startup

Explanation for TQM860 PPC board

0. Kernel entry point:

arch/ppc/boot/common/crt0.S: *_start*

1. *_start* calls on:

arch/ppc/boot/simple/head.S: *start*

2. *start* calls on:

arch/ppc/boot/simple/relocate.S: *relocate*

3. *relocate* calls on:

arch/ppc/boot/simple/misc-embedded.c: *load_kernel()*

4. *load_kernel()* initializes the serial line and

uncompresses kernel starting at address 0.

6. *relocate* jumps to address 0x00000000, where kernel start address is.
7. `arch/ppc/kernel/head_8xx.S: __start`
8. `__start` eventually calls `init/main.c:start_kernel()`
9. `load_kernel()` returns to *relocate*
10. `start_kernel()` does:
 1. Locks kernel
 2. `setup_arch()`
 3. `sched_init()`
 4. `parse_args()`
 5. `trap_init()`
 6. `init_IRQ()`

7. *time_init()*

8. *console_init()*

9. *mem_init()*

10. *calibrate_delay()* => *loops_per_jiffy*

11. *rest_init()*

11. *rest_init()* does:

1. Start `init` thread
2. Unlocks the kernel
3. Becomes the idle task

12. The `init` task:

1. `lock_kernel()`

2. `do_basic_setup()` => call various `init()` fcts

3. `prepare_namespace()` => mount rootfs

4. `free_initmem()`

5. `unlock_kernel()`

6. `execve()` on the `init` program (`/sbin/init`)

Device driver overview

- 1.Licensing reminder
- 2.Device driver model
- 3.Writing a char device driver
- 4.Writing a block device driver
- 5.Writing a network device driver
- 6.Time-keeping
- 7.Memory needs
- 8.Hardware access
- 9.Interrupt handling
- 10.Printing out messages to console

1. Licensing reminder

- Although the use of binary-only modules is widespread, Kernel modules are not immune to kernel GPL.
- Many kernel developers have come out rather strongly against binary-only modules.
- If you are linking a driver as built-in, then you are most certainly forbidden from distributing the resulting kernel under any license other than the GPL.
- If you're wary of the GPL, push critical driver-intelligence to user-space.

2. Device driver model

- Device files
 - Everything is a file in Unix, including devices
 - All devices are located in the `/dev` directory
 - Only networking devices do not have `/dev` nodes
 - Every device is identified by major / minor number
 - Can be allocated statically (`devices.txt`)
 - Can be allocated dynamically
 - To see devices present: `$ cat /proc/devices`
 - Alternatives: `devfs`, `sysfs` (used to be `driverfs`)

- Char devices:
 - Stream-oriented devices
 - Manipulated using: *struct file_operations**
- Block devices:
 - Disk-oriented devices
 - Manipulated using: *struct block_device_operations**
- Networking devices
 - All networking devices
 - Manipulated using: *struct net_device**

- Subsystem drivers:

- USB:

*struct usb_driver **

- PCI

- I2C

- ...

3. Writing a char device driver

- Register char dev during module initialization
- Char dev registration: include/linux/fs.h

```
int register_chrdev(unsigned int,  
                   const char *,  
                   struct file_operations *);
```

- First param: Major number
- Second param: Device name (as displayed in /proc/devices)
- Third param: File-ops
 - Defined in include/linux/fs.h
 - Contains callbacks for all possible operations on a char device.

```

struct file_operations {
    struct module *owner;
    loff_t (*llseek) (struct file *, loff_t, int);
    ssize_t (*read) (struct file *, char __user *, size_t, loff_t *);
    ssize_t (*aio_read) (struct kiocb *, char __user *, size_t, loff_t);
    ssize_t (*write) (struct file *, const char __user *, size_t, loff_t *);
    ssize_t (*aio_write) (struct kiocb *, const char __user *, size_t, loff_t);
    int (*readdir) (struct file *, void *, filldir_t);
    unsigned int (*poll) (struct file *, struct poll_table_struct *);
    int (*ioctl) (struct inode *, struct file *, unsigned int, unsigned long);
    long (*unlocked_ioctl) (struct file *, unsigned int, unsigned long);
    long (*compat_ioctl) (struct file *, unsigned int, unsigned long);
    int (*mmap) (struct file *, struct vm_area_struct *);
    int (*open) (struct inode *, struct file *);
    int (*flush) (struct file *);
    int (*release) (struct inode *, struct file *);
    int (*fsync) (struct file *, struct dentry *, int datasync);
    int (*aio_fsync) (struct kiocb *, int datasync);
    int (*fasync) (int, struct file *, int);
    int (*lock) (struct file *, int, struct file_lock *);
    ssize_t (*readv) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*writev) (struct file *, const struct iovec *, unsigned long, loff_t *);
    ssize_t (*sendfile) (struct file *, loff_t *, size_t, read_actor_t, void *);
    ssize_t (*sendpage) (struct file *, struct page *, int, size_t, loff_t *, int);
    unsigned long (*get_unmapped_area) (struct file *, unsigned long, unsigned ...
    int (*check_flags) (int);
    int (*dir_notify) (struct file *filp, unsigned long arg);
    int (*flock) (struct file *, int, struct file_lock *);
};

```

- Call `register_chrdev()` and pass it a valid `file_operations` structure.
- Return 0 from initialization function to tell `insmod` that everything is OK.
- That's it. Every time the device in `/dev` having the same major number as the one you registered is opened, your driver will be called.
- To remove char dev on `rmmmod`:

```
int unregister_chrdev(unsigned int,  
                      const char *);
```


4. Writing a block device driver

- Register block dev during module initialization
- Block dev registration: `include/linux/fs.h`

```
int register_blkdev(unsigned int,  
                    const char *);
```

- First param: Major number
- Second param: Device name
- Disk allocation: `include/linux/genhd.h`

```
struct gendisk *alloc_disk(int minors);
```

- Block queue registration: `include/linux/blkdev.h`

```
extern void blk_init_queue(request_fn_proc *,  
                           spinlock_t *);
```

- Queue of pending I/O operations for device

- First param: Queue handler function
- Second param: Lock for accessing queue
- Call *register_blkdev()*.
- Call *alloc_disk()* and pass it the number of disks.
- Call *blk_init_queue()* and pass it a valid callback.
- Return 0 from init function to tell insmod status

- Now, all block operations on your device (/dev entry with same major number as driver) will be queued to your driver.
- To remove block dev on rmmmod:

```
void blk_cleanup_queue(request_queue_t *);  
void put_disk(struct gendisk *disk);  
int unregister_blkdev(unsigned int, const char *);
```

5. Writing a network device driver

- Register net dev during module initialization
- Net dev registration: include/linux/netdevice.h

```
int register_netdevice(struct net_device *dev);
```

- Param: net device ops
 - Defined in include/linux/netdevice.h
 - Contains all callbacks related to network devices
 - This is a huge structure with A LOT of fields
- Call *register_netdevice()* and pass it a valid *net_device* structure.
- Return 0 as status to insmod

- Your device will need to be *opened* by the kernel in response to an *ifconfig* command.
- Your *open()* function must allocate a packet queue to deal with packets sent to your device.
- Calling your device will depend on packet routing at the upper layers of the stack.
- **To remove:** `unregister_netdev(struct net_device *dev);`

6. Time-keeping

- Simple ways to get the time
 - *jiffies*: updated at every kernel tick
 - *do_gettimeofday()*: good precision depending on arch
 - *get_cycles()*: arch-independent call to get CPU cycle count.
- Being notified in due time:
 - Use timers: `include/linux/timer.h`
 - *jiffies* resolution
 - Use *struct timer_list*: contains expiry and callback
 - Initialize timer: *init_timer()*
 - Add timer to global timer list: *add_timer()*
 - Remove timer before expiry: *del_timer()*

7. Memory needs

- Main kernel memory functions:
 - *kmalloc(size, type)*: up to 128KB of memory
 - *vmalloc(size)*: get large contiguous virtual memory
- Main types of memory allocation:
 - GFP_KERNEL => normal alloc / may sleep
 - GFP_ATOMIC => int handlers / never sleeps
 - GFP_USER => user-space / low priority
 - ... See include/linux/slab.h for full list

8. Hardware access

- Main functions to be able to access hardware:
 - Requesting region: *request_mem_region()*
 - Releasing region: *release_mem_region()*
 - Map physical region to VM: *ioremap()*

9. Interrupt handling

- Setting up an interrupt handler:

```
extern int request_irq(unsigned int,  
                      irq_handler_t, unsigned long, const char *, void *);
```

- First param: IRQ number
- Second param: handler
- Third param: flags for OS int delivery
- Four param: device name
- Fifth param: provide device ID in case of shared interrupts.
- Disabling interrupt handler:

```
void free_irq(unsigned int, void *);
```

10. Printing out messages to console

- Meet the kernel's printf: *printk()*
- Defined: `include/linux/kernel.h`

```
int printk(const char * fmt, ...)
```
- Implemented: `kernel/printk.c`
- Can lose data in cases of large output
- Widely-used throughout kernel sources
- Don't call while holding lock, has lock contention of its own.

A Quick Java Introduction

1. Parts
2. Hello World
3. What's missing from C/C++?
4. What's different?

1. Parts

- JDK
 - Compiler
 - Libraries
- JRE
 - JVM
- Eclipse

2. Hello World

- The code:

```
class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello World!");  
    }  
}
```

- Compiling:

```
$ javac HelloWorld.java
```

- Running:

```
$ java HelloWorld  
Hello World!  
$
```

3. What's missing from C/C++?

- Preprocessor:
 - #include
 - #define
 - #ifdef/#endif
- Definitions:
 - typedef
 - struct
 - union
 - enum

- Keywords:
 - goto
- Pointers:
 - malloc/free
- Strings as arrays of char:
 - Use `“foo”.toCharArray()` to get a char array
 - Use `foo = new String(charArray)` to get a String

4. What's different?

- Garbage collection:

```
String myStr = new String("foo");  
myStr = null;  
myStr = new String("bar");
```

- String concatenation:

```
strC = strA + strB;
```

- Reflection:

```
// Direct call  
new MyClass().foo();
```

```
// Call through reflection reflection  
Class cls = Class.forName("MyClass");  
cls.getMethod("foo", null).invoke(cls.newInstance(), null);
```


- Interfaces:

```
interface Foo
{
    bar();
}
```

```
public class MyClass implements Foo {
    public bar() {
        ...
    }
}
```

- Can implement multiple interfaces (v. multiple inheritance)
- All methods declared as part of a class – no standalone “functions”

- Anonymous classes:

```
public class MyClass {  
    public static void main(String args[]) {  
        new Thread(new Runnable() {  
            @Override  
            public void run() {  
                System.out.println("Hello Anon");  
            }  
        }).start();  
    } // end main  
}
```

- Inheritance:

```
public class Foo extends Bar {  
    ...  
}
```

- Package (first line in source file):

```
package com.foo.mypackage;
```

- Importing (using a package):

```
import com.foo.mypackage;
```

- **Threads:**

```
public class MyThread extends Thread {  
    public void run() {  
        System.out.println("Hello Thread World!");  
    }  
  
    public static void main(String args[]) {  
        (new MyThread()).start();  
    }  
}
```

- **Synchronization:**

```
public synchronized void foo() {  
    ...  
}
```

- Checking type:

```
public class Foo extends Bar {  
    ...  
}  
...  
foo myFoo = new Foo()  
if (myFoo instanceof Bar) {  
    ...  
}
```

- Overrides:

```
public class Bar {  
    public void aMethod() {  
        ...  
    }  
}
```

```
public class Foo extends Bar {  
    @Override  
    public void aMethod() {  
        ...  
    }  
}
```

- Arrays:

```
int[] myArray = new int[100];
```

- JavaDoc

- Junit

- JNI

- Packages:

```
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font
java.awt.geom
java.awt.im
java.awt.im.spi
java.awt.image
java.awt.image.renderable
java.awt.print
java.beans
java.beans.beancontext
java.io
java.lang
java.lang.ref
java.lang.reflect
java.math
java.net
java.nio
java.nio.channels
java.nio.channels.spi
java.nio.charset
java.nio.charset.spi
```

java.rmi
java.rmi.activation
java.rmi.dgc
java.rmi.registry
java.rmi.server
java.security
java.security.acl
java.security.cert
java.security.interfaces
java.security.spec
java.sql
java.text
java.util
java.util.jar
java.util.logging
java.util.prefs
java.util.regex
java.util.zip
javax.accessibility
javax.crypto
javax.crypto.interfaces
javax.crypto.spec
javax.imageio
javax.imageio.event
javax.imageio.metadata
javax.imageio.plugins.jpeg
javax.imageio.spi
javax.imageio.stream
javax.naming
javax.naming.directory
javax.naming.event
javax.naming.ldap
javax.naming.spi

javax.net
javax.net.ssl
javax.print
javax.print.attribute
javax.print.attribute.standard
javax.print.event
javax.rmi
javax.rmi.CORBA
javax.security.auth
javax.security.auth.callback
javax.security.auth.kerberos
javax.security.auth.login
javax.security.auth.spi
javax.security.auth.x500
javax.security.cert
javax.sound.midi
javax.sound.midi.spi
javax.sound.sampled
javax.sound.sampled.spi
javax.sql
javax.swing
javax.swing.border
javax.swing.colorchooser
javax.swing.event
javax.swing.filechooser
javax.swing.plaf
javax.swing.plaf.basic
javax.swing.plaf.metal
javax.swing.plaf.multi
javax.swing.table
javax.swing.text
javax.swing.text.html
javax.swing.text.html.parser
javax.swing.text.rtf
javax.swing.tree
javax.swing.undo

```
javax.transaction
javax.transaction.xa
javax.xml.parsers
javax.xml.transform
javax.xml.transform.dom
javax.xml.transform.sax
javax.xml.transform.stream
org.ietf.jgss
org.omg.CORBA
org.omg.CORBA.DynAnyPackage
org.omg.CORBA.ORBPackage
org.omg.CORBA.TypeCodePackage
org.omg.CORBA.portable
org.omg.CORBA_2_3
org.omg.CORBA_2_3.portable
org.omg.CosNaming
org.omg.CosNaming.NamingContextExtPackage
org.omg.CosNaming.NamingContextPackage
org.omg.Dynamic
org.omg.DynamicAny
org.omg.DynamicAny.DynAnyFactoryPackage
org.omg.DynamicAny.DynAnyPackage
org.omg.IOP
org.omg.IOP.CodecFactoryPackage
org.omg.IOP.CodecPackage
org.omg.Messaging
```

org.omg.PortableInterceptor
org.omg.PortableInterceptor.ORBInitInfoPackage
org.omg.PortableServer
org.omg.PortableServer.CurrentPackage
org.omg.PortableServer.POAManagerPackage
org.omg.PortableServer.POAPackage
org.omg.PortableServer.ServantLocatorPackage
org.omg.PortableServer.portable
org.omg.SendingContext
org.omg.stub.java.rmi
org.w3c.dom
org.xml.sax
org.xml.sax.ext
org.xml.sax.helpers

Thank you ...

karim.yaghmour@opersys.com



Acknowledgements:

- Some figures and snippets taken from Google's Android “Dev Guide” at developer.android.com distributed under the Apache 2.0 license.