Bauhaus-Universität Weimar

Mobile Information Systems

Lecture 10: Android Internals

© 2015-20 Dr. Florian Echtler
Bauhaus-Universität Weimar
<florian.echtler@uni-weimar.de>

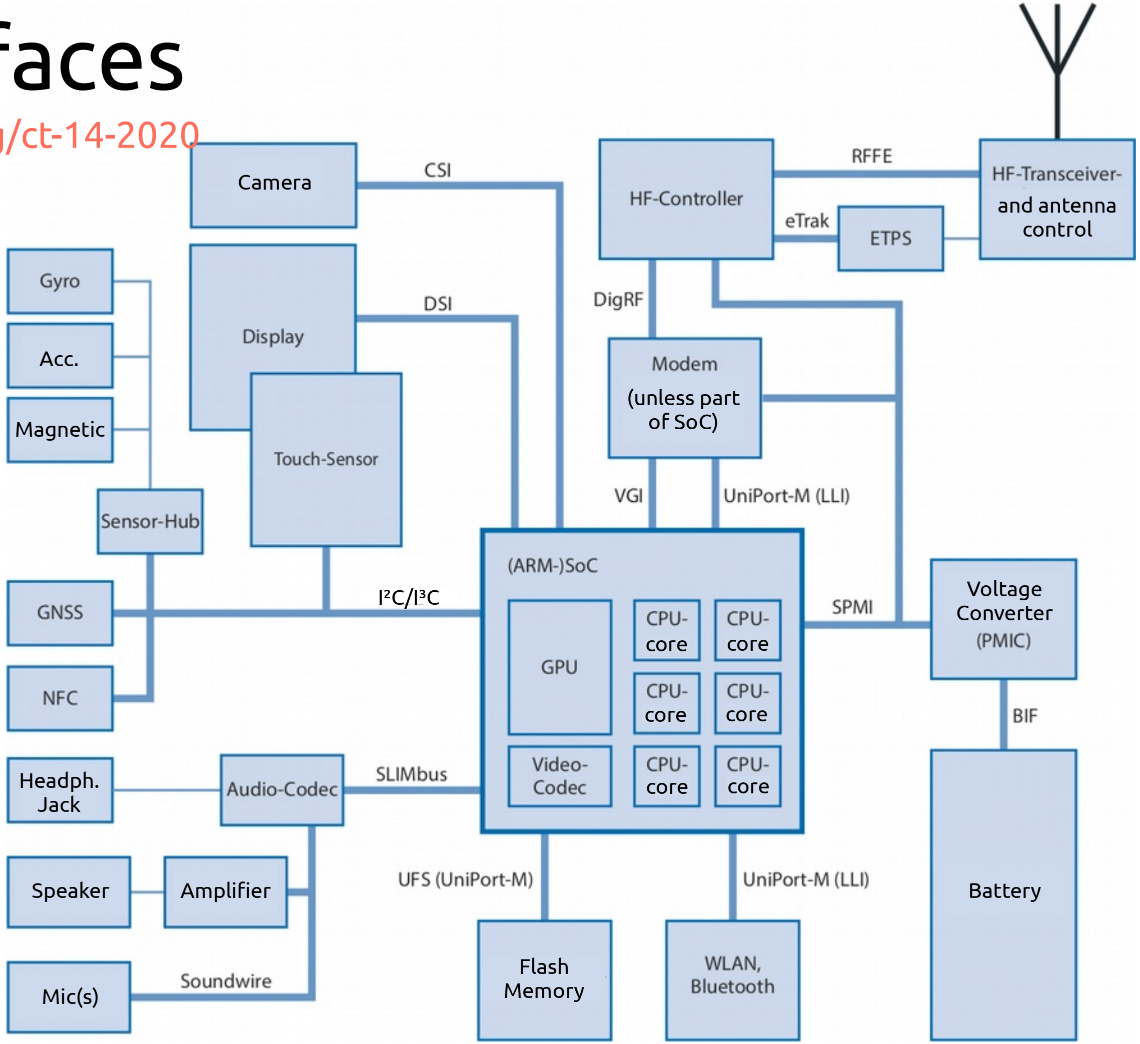
A look under the hood

Image source (PD): https://commons.wikimedia.org/...old_jeep_from_world_war_two.jpg

MIPI Hardware Interfaces

Image source (FU): https://shop.heise.de/katalog/ct-14-2020

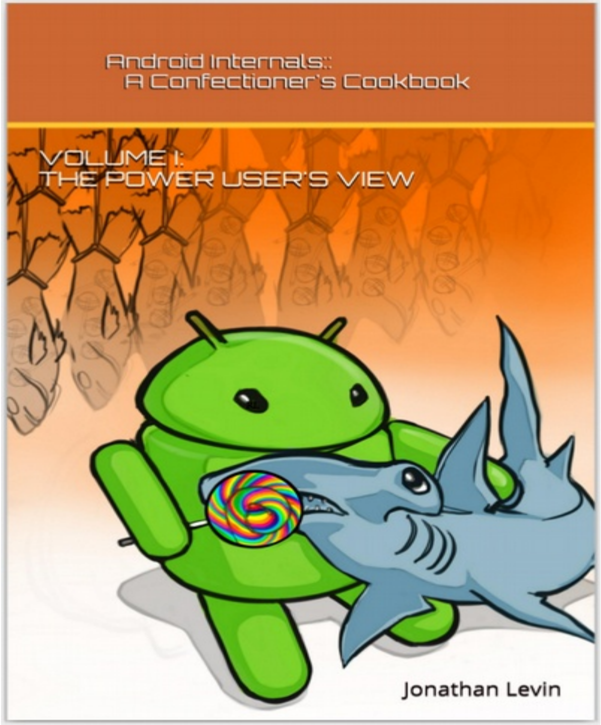
- MIPI = Mobile Industry Processor Interface
- Multiple internal buses:
  - Highspeed: CSI, DSI, Uniport
  - Low-speed: I²C, I³C, SPMI, BIF, SLIMbus
- (Mostly) abstracted away through kernel/HAL



Book recommendation

Image source (PD): http://newandroidbook.com/AIvI-M-RL1.pdf

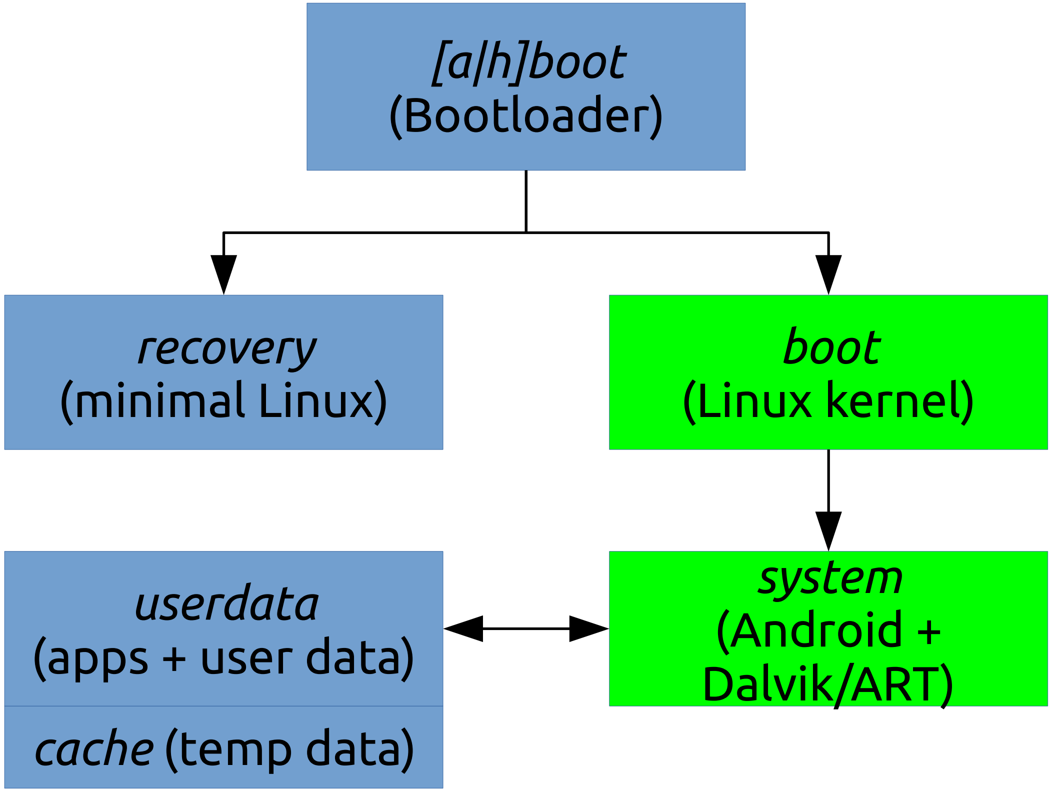
- “Android Internals: A Confectioner’s Cookbook” by Jonathan Levin
- http://newandroidbook.com/
- Older edition freely available (up to Marshmallow PR1)



Android Internals (1)

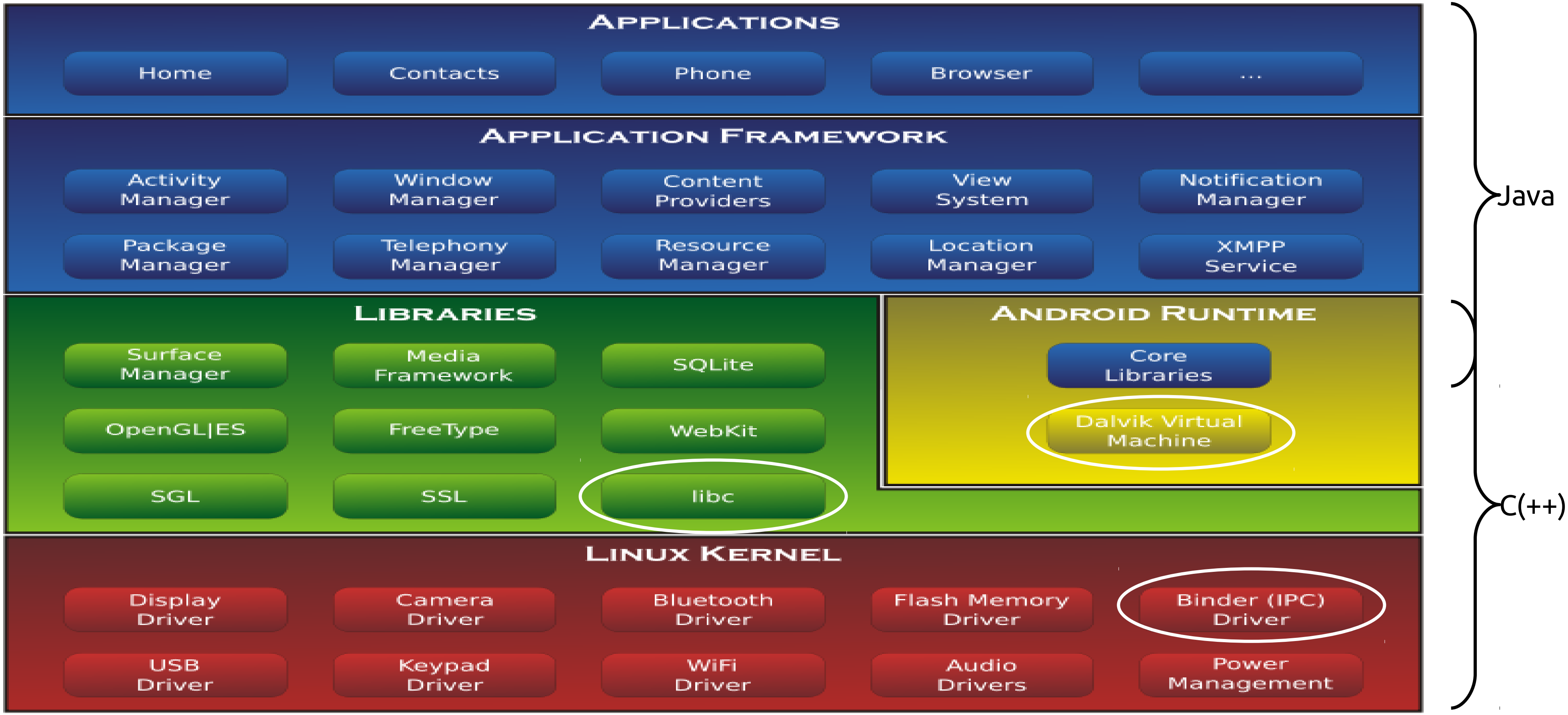
- Android storage partitions (generic)
  - *[a/h]boot* → bootloader (fastboot)
  - *boot* → Linux kernel + intital ramdisk (initrd)
  - *recovery* → Linux kernel + minimal toolset
  - *system* → main Android system
  - *userdata* → user data storage (possibly encrypted)
  - *cache* → temporary storage
- … plus many vendor-specific extensions
- Note: SD card mounted as storage extension for userdata

Android Internals (2)



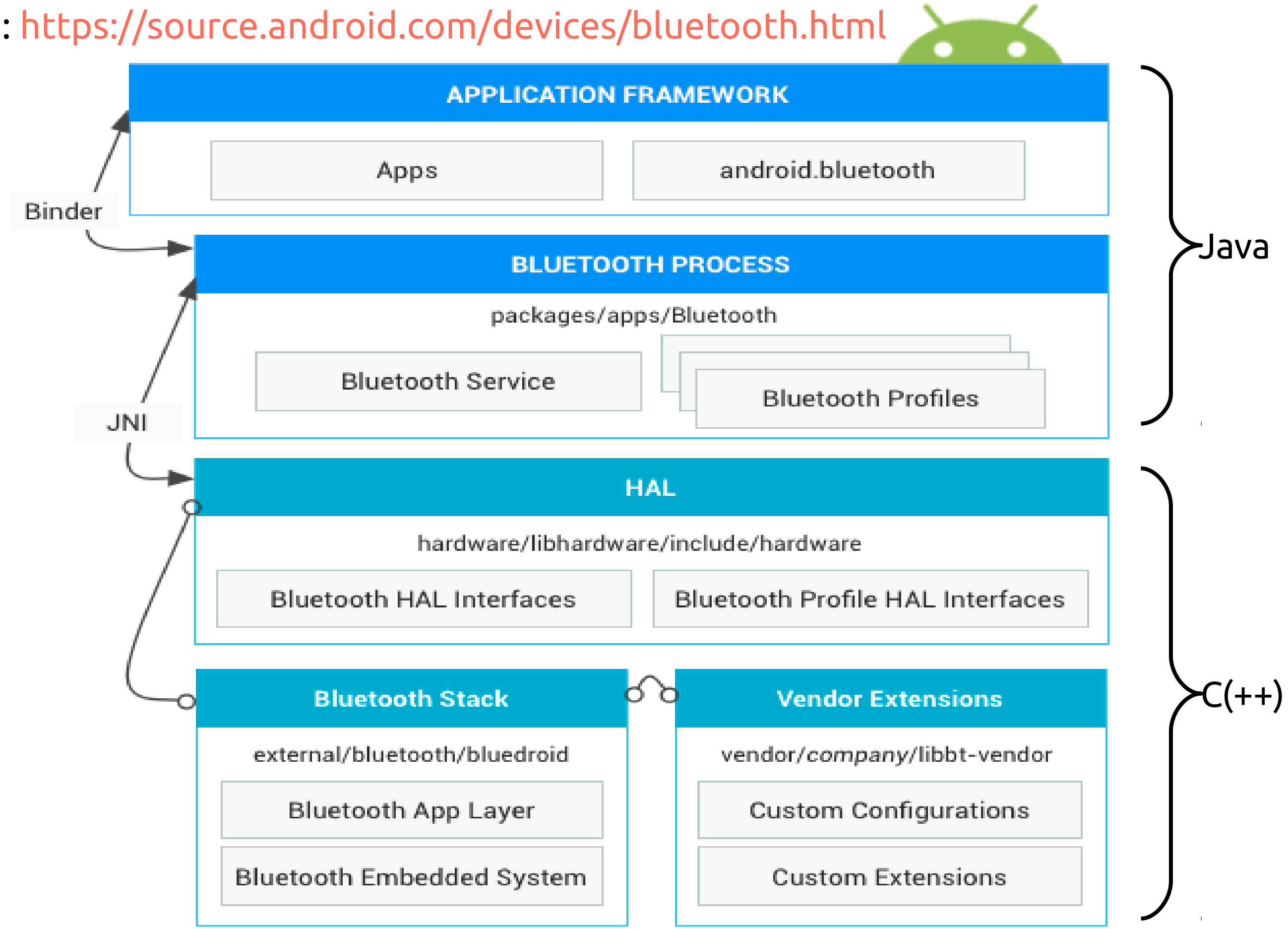
Android Internals (3)

Image source (CC): https://commons.wikimedia.org/wiki/File:Android-System-Architecture.svg



Example: Bluetooth stack

Image source (FU): https://source.android.com/devices/bluetooth.html



Linux kernel

- Android = Linux?
  - Uses new IPC = Inter-Process Communication:
  - *Binder* (in addition to sockets, shared mem, ...)
  - Part of official kernel since February 2015
- Closed hardware drivers
  - HW vendors want to keep their secrets
  - Only binary drivers, no source code
    → difficult to replace/modify kernel
  - Conflict with GPL (often leads to legal battles, see e.g.
    https://blog.sebastian-schmid.de/.../xiaomi-gpl-violation.html)

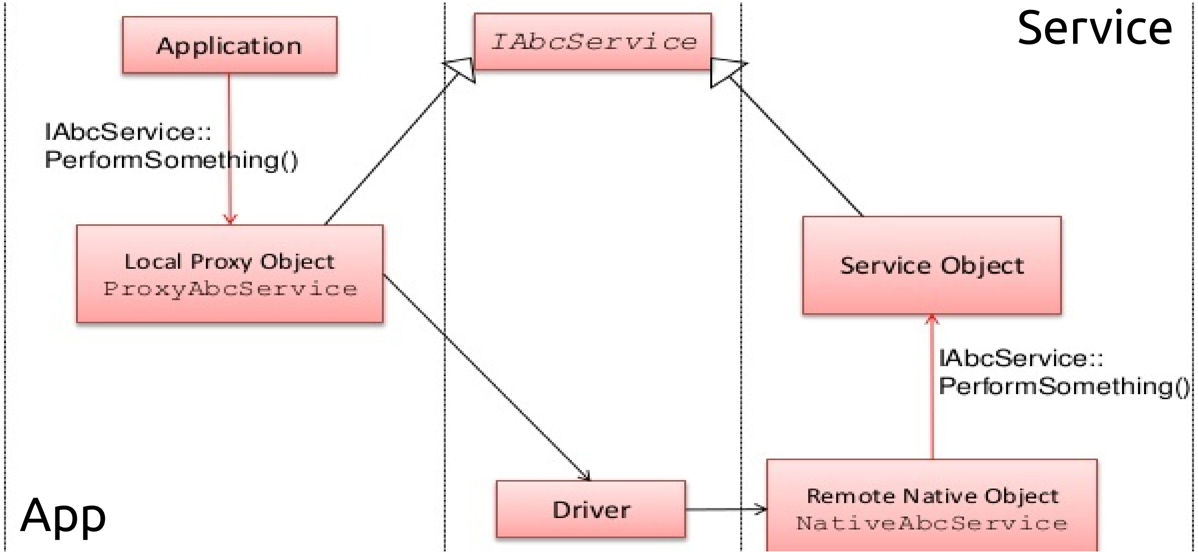
Linux tools on Android

- Can get a standard shell with `adb shell`
- Standard(-ish) tools available: `ps`, `top`, `grep` ...
- Many static binaries work (why?)
- Other Android-specific tools, e.g.:
  - `monkeyrunner` → generates random touch/keypress events for UI testing
  - `app_process` → allows scripts to interact with Dalvik/Java components, e.g. start activity

Binder IPC

Image source (FU): http://www.slideshare.net/pchethan/android-binder-ipc-implementation

- Provides RPC = Remote Procedure Call
- Communication apps ↔ services
- Synchronous or asynchronous modes



libc vs. Bionic

- libc = Standard C Library
  - Used on most Unix-like systems
  - Provides basic C functions, e.g. `printf`, `strlen`, …
- Bionic: Google-specific replacement for libc
  - BSD license instead of (L)GPL
  - Smaller & faster
  - Limited C++ support (no exception handling, STL needs to be linked separately)

Java VM

- Original design goal:
  - Support multiple CPU archs (arm*, mips, x86) …
  - … with identical app binaries (no rebuild etc.)

→ Requires virtual machine (VM)

  - Java already provides most features
  - Machine-independent bytecode
  - Bad reputation for high resource usage

→ Own adaptation of Java VM: Dalvik

(named after Norwegian fishing village!?)

Stack-based vs. register-based

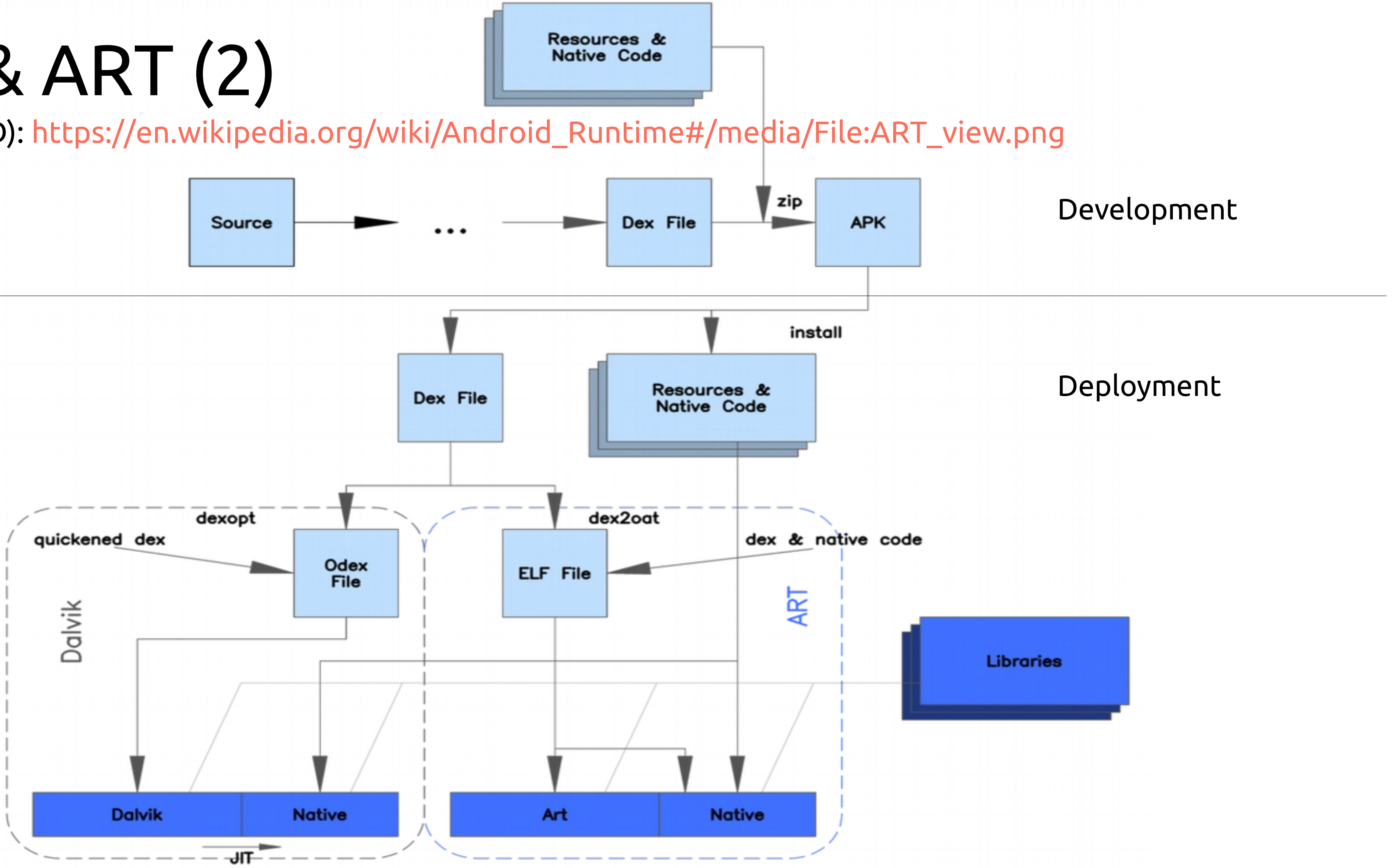
- Stack-based VM:
  - Very simple compiler (part of many grad courses)
  - Needs lots of RAM accesses (many push/pop ops)
- Register-based VM:
  - Generally easier to optimize for low-end hardware (e.g. store intermediate values in CPU registers)
  - Less RAM accesses required
  - Drawback: compiler more complex

Dalvik & ART (1)

- Dalvik VM default up to 4.4
  - Register machine instead of stack machine
  - Better speed & resource usage
  - JIT = Just-In-Time compiler (since 2.2)
  - Performance-critical sections converted to native
- ART default from 5.0
  - AOT = Ahead-Of-Time compiler
  - Creates native Linux ELF executables at install time
  - Tradeoff: storage space ↔ execution speed

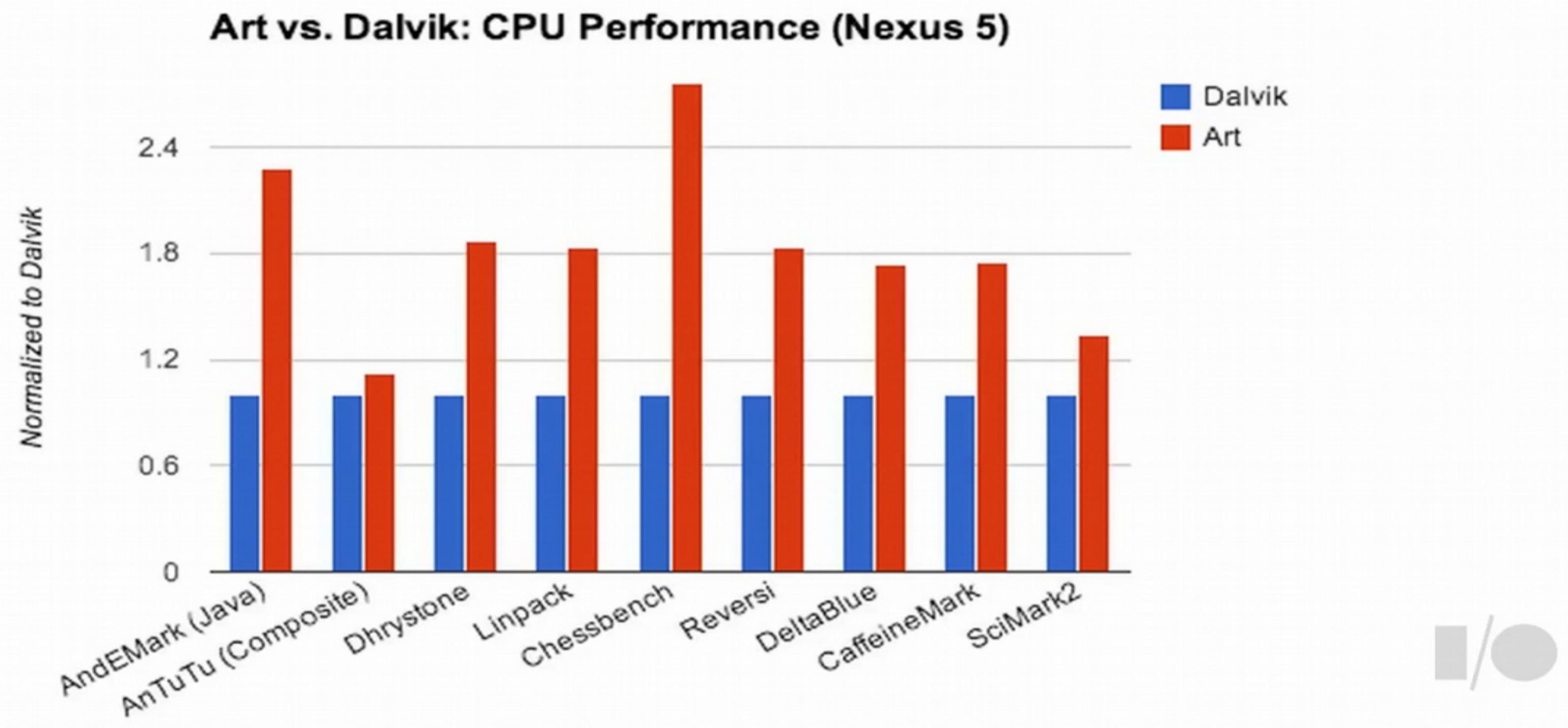
Dalvik & ART (2)

Image source (PD): https://en.wikipedia.org/wiki/Android_Runtime#/media/File:ART_view.png



Dalvik & ART (3)

Image source (FU): http://anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l/



APK & DEX

- Compilation process:
  - Java source (.java) → `javac` → Java class (.class)
  - Since Android Studio 3.0: also Kotlin (.kt) source
  - Class files → `dx` → Dalvik Executable (.dex)
    - converts bytecode (Java VM → Dalvik VM)
    - removes duplicate data (e.g. strings)
  - DEX files + resources (e.g. images) → zip → APK
- On device: JIT/AOT compilation to native code
  - Question: why on device and not at compile time?

SDK & NDK

- Android SDK: Java components & interfaces
- Android NDK = Native Development Kit
  - C++ compiler and libraries
  - Used for performance-critical code
    (less important with ART)
  - Can be used for system & shell tools
  - Can be used to include existing C(++) libraries

Security measures/caveats (1)

- On Linux level:
  - Separation of privileges: separate user IDs for each app, service etc.
  - Minimum amount of permissions for each ID (nearly none by default)
  - Exception: UID 0 - “root”, omnipotent, only very few processes by default
  - Additional restrictions through Linux Capabilities (fine-grained permissions) and SELinux policies

Security measures/caveats (2)

- On Java level:
  - Dalvik checks APK signature on installation
  - App process has no permissions/capabilities
  - All system services accessed through binder
  - Services check `<uses-permission>` tags

Security measures/caveats (3)

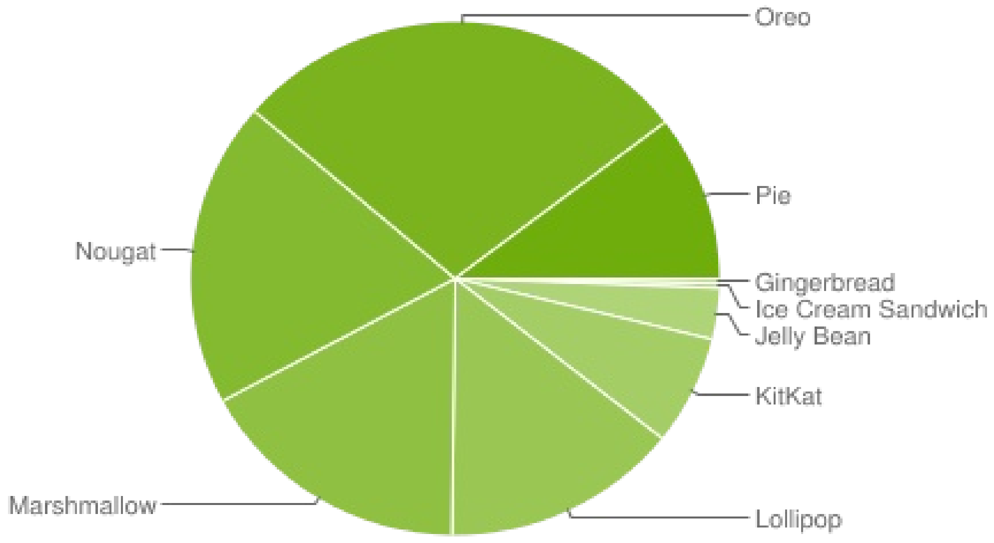
- “Rooting” a device: getting access to UID 0 (“root”) as regular user
  - Allows to modify system components, access restricted information
  - Unlocked bootloader → patched *boot* partition
  - Sometimes uses existing, unpatched security holes
  - May open up new, additional security issues

Apps & services (1)

Image source (FU): https://developer.android.com/about/dashboards/index.html, data 19-05-01 – 19-05-07

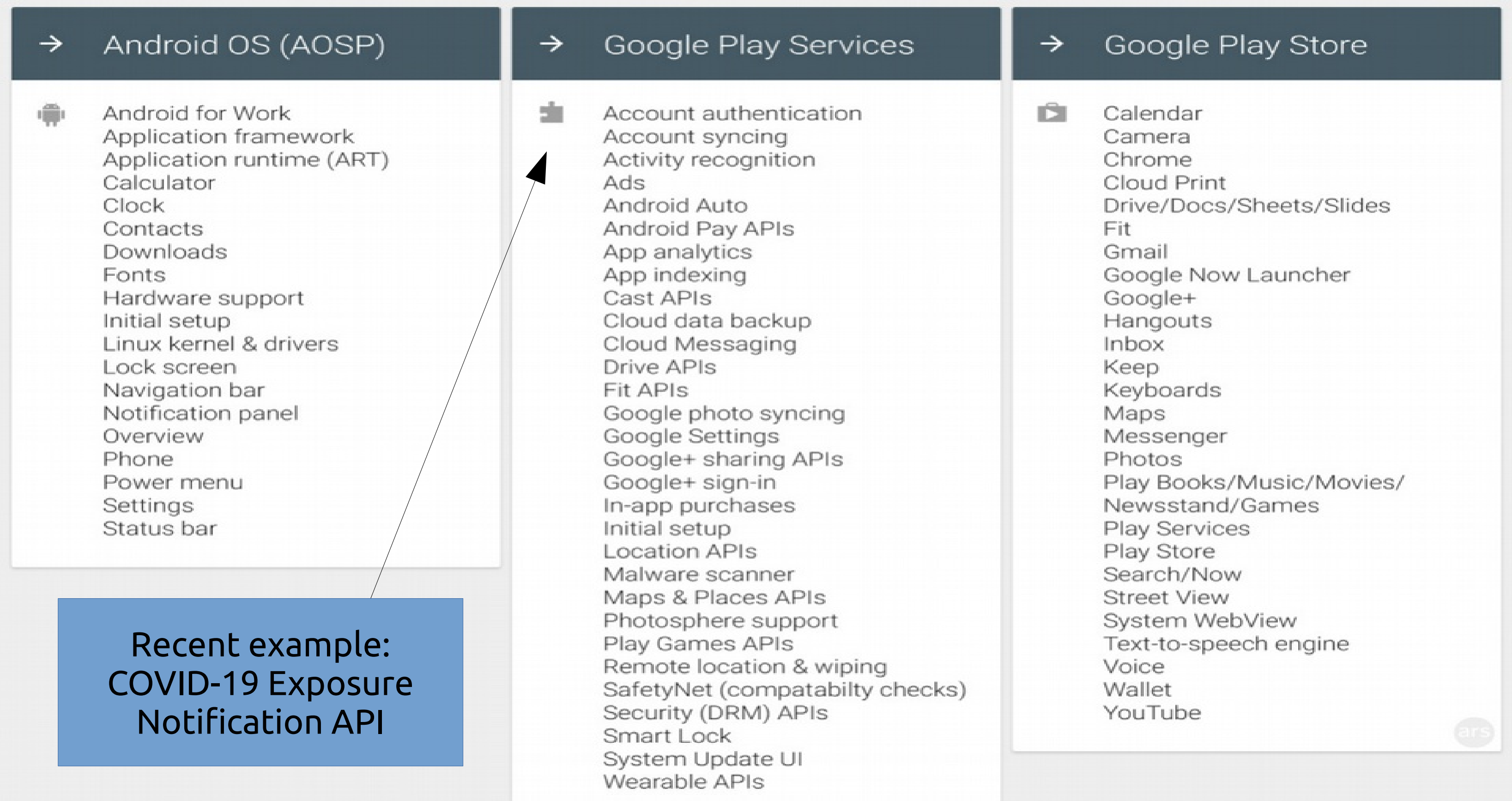
- Problem: fragmentation due to sloppy vendor updates
  → move more services out of core OS (cf. Project Treble)
- Data source: play store access from May 2019

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.3%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.3%
4.1.x	Jelly Bean	16	1.2%
4.2.x		17	1.5%
4.3		18	0.5%
4.4	KitKat	19	6.9%
5.0	Lollipop	21	3.0%
5.1		22	11.5%
6.0	Marshmallow	23	16.9%
7.0	Nougat	24	11.4%
7.1		25	7.8%
8.0	Oreo	26	12.9%
8.1		27	15.4%
9	Pie	28	10.4%



Apps & services (2)

Image source (FU): http://arstechnica.com/...play-services-data-backup-and-more/



Recent example:
COVID-19 Exposure
Notification API

HTML5 vs. native apps (1)

- HTML5
  - \+ No installation required, just URL (QR code, NFC)
  - \+ Possible to also target iOS, desktops, …
  - – Additional layer of indirection (WebView/browser)
    → less performance (in general)

    Note: modern ARM CPUs have *dedicated Javascript* instructions
  - – Hardware access difficult or impossible (but improving, e.g. camera access in Javascript possible)
  - – Browser compatibility issues (Firefox ↔ Chrome ↔ WebKit)
    → *Return of the Evil Browser Switch*

HTML5 vs. native apps (2)

- “Native”
  - + Generally faster (although very hard to quantify)
  - + Better access to hardware (sensors, camera, …)
  - + Payment services provided by app store
  - – Requires installation from app store or APK file
  - – In Android context: still means Java/Kotlin (+JIT/AOT)
- Summary: many different trade-offs (again)
  - – Simple cross-platform apps → HTML5
  - – Performance, hardware access needed → native

Decompiling & reverse-engineering

- Often necessary to look into existing apps …
  - … for security research
  - … for reverse engineering
- Two major options available: dex2jar & jd-gui, jadx (relatively new, but all-in-one solution)
- Create somewhat readable Java source code
- Counter measure: Proguard
  - Obfuscates & compresses source code
  - Variable/method names converted to a...z, aa, ab, ...

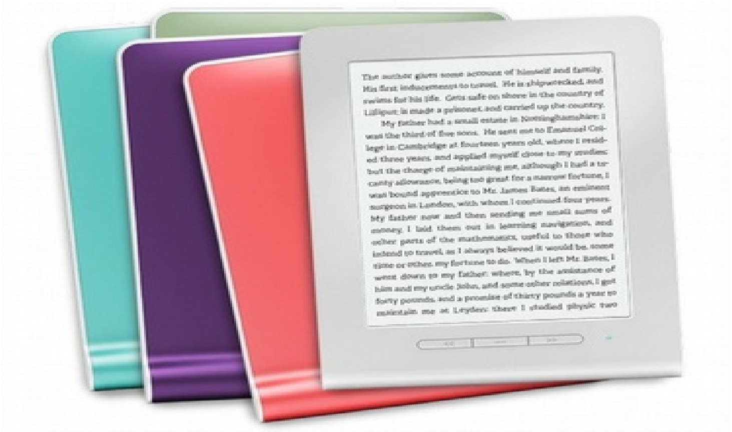
RE case study: txtr Beagle

Image source (FU): http://de.txtr.com/beagle/

- Announced as “10 € e-reader”, company broke
- Very simple device: e-ink display, battery, flash storage, Bluetooth + small CPU
- Books converted to images + transferred via Android app
- Goal: reverse-engineer Bluetooth protocol

→ decompile Android app

→ protocol now open source
(see https://github.com/schierla/jbeagle)



The End


OUR SERVERS ARE USING TOO MUCH ELECTRICITY. WE NEED TO VIRTUALIZE.
I DID MY PART BY READING ABOUT VIRTUALIZATION IN A TRADE JOURNAL. NOW YOU DO THE SOFTWARE PART.
WHY IS YOUR PART TAKING SO LONG?
www.dilbert.com
scottadams@aol.com
3-12-08 © 2008 Scott Adams, Inc./Dist. by UFS, Inc.