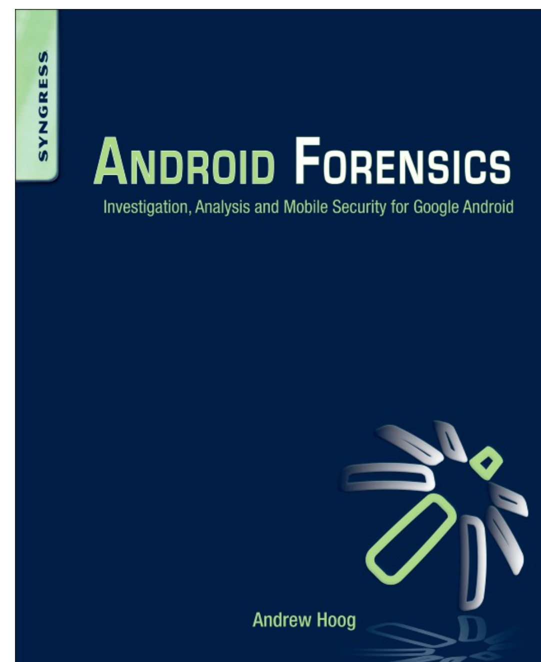
VIAFORENSICS

viaforensics.com

Android forensics

boot process, security, system, rooting, dumping, analysis, etc.



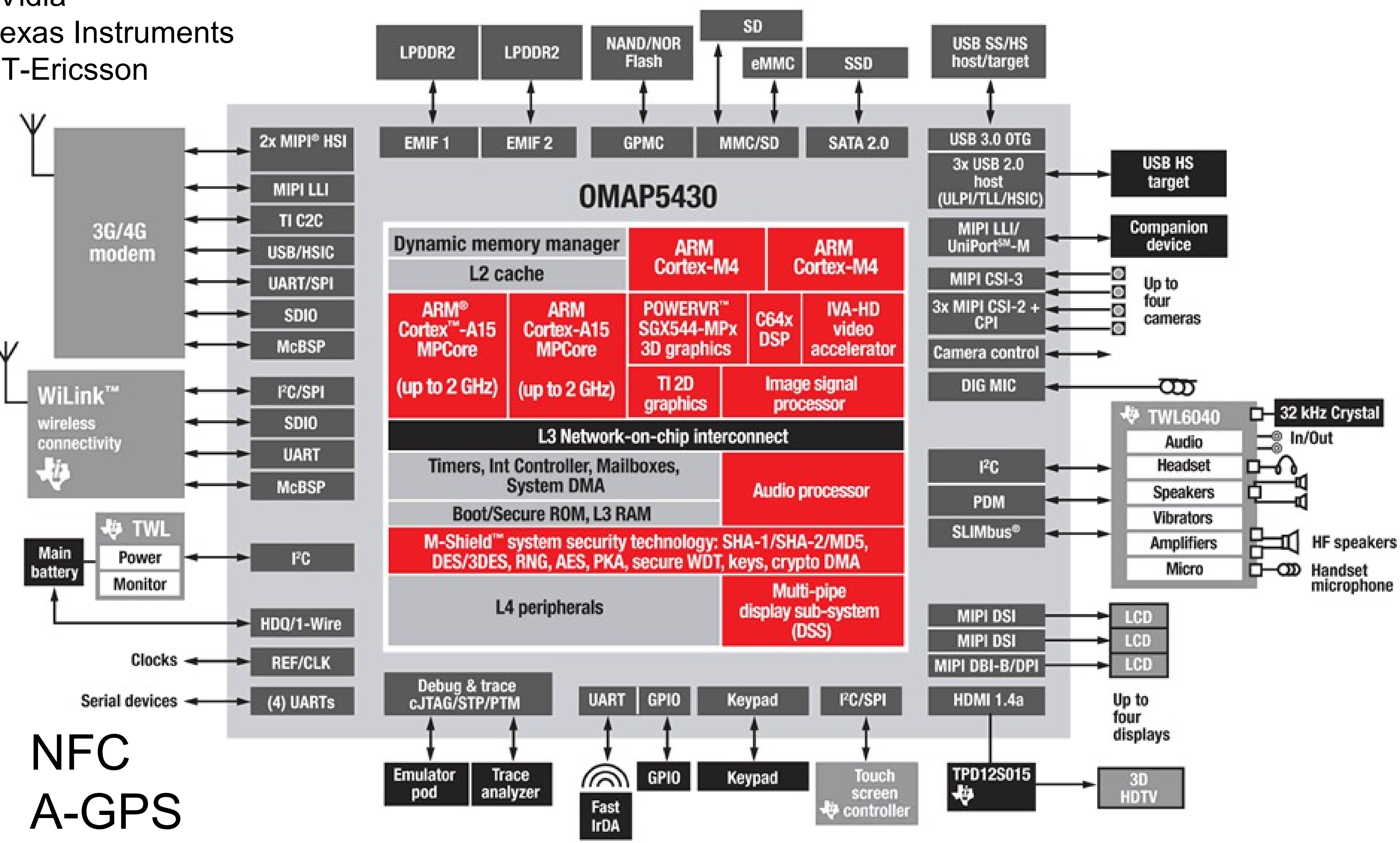
Android and mobile forensics

- Any interaction with the smartphone will change the device in some way
  - Use judgment, explain modifications and choices made
- Further complicating Android forensics is the sheer variety of devices, Android versions, and applications
  - The permutations of devices and Android versions alone are in the thousands and each device plus platform has unique characteristics
- While a logical analysis of every Android phone is achievable, the vast combinations make the full physical acquisition of every Android device likely unachievable
  - Even a minor difference in the Android version may require extensive testing and validation
- However the open source aspect of Android greatly assists in the fundamental understanding a forensic analyst requires, making Android an ideal platform to work on

At least 5 MF of SoC
- Samsung
- Qualcomm
- MediaTek
- Intel (x86)
- nVidia
- Texas Instruments
- ST-Ericsson

Android hardware platforms

TI OMAP5430 SoC

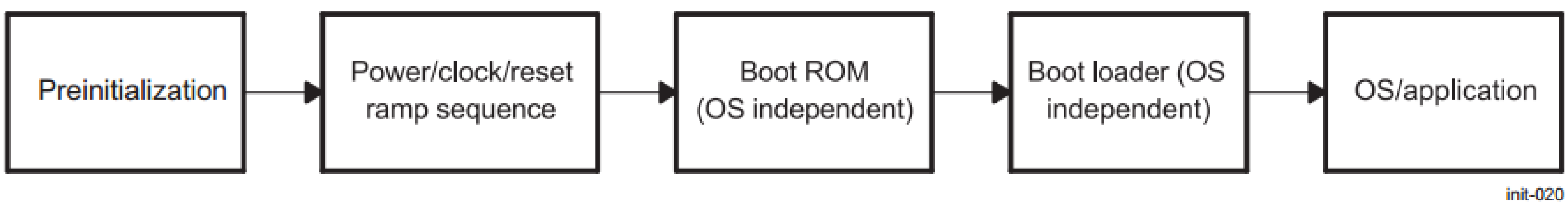


NFC
A-GPS

ROM and bootloaders

- Android devices, like any other computer, have a fairly standard boot process which allows the device to load the needed firmware, OS, and user data into memory to support full operation
- Although the boot process itself is well defined, the firmware and ROM varies by manufacturer and by device
- OMAP35x Technical Reference Manual (Rev. X), page 3399 ->
  - http://www.ti.com/product/omap3530

Figure 25-1. Initialization Process



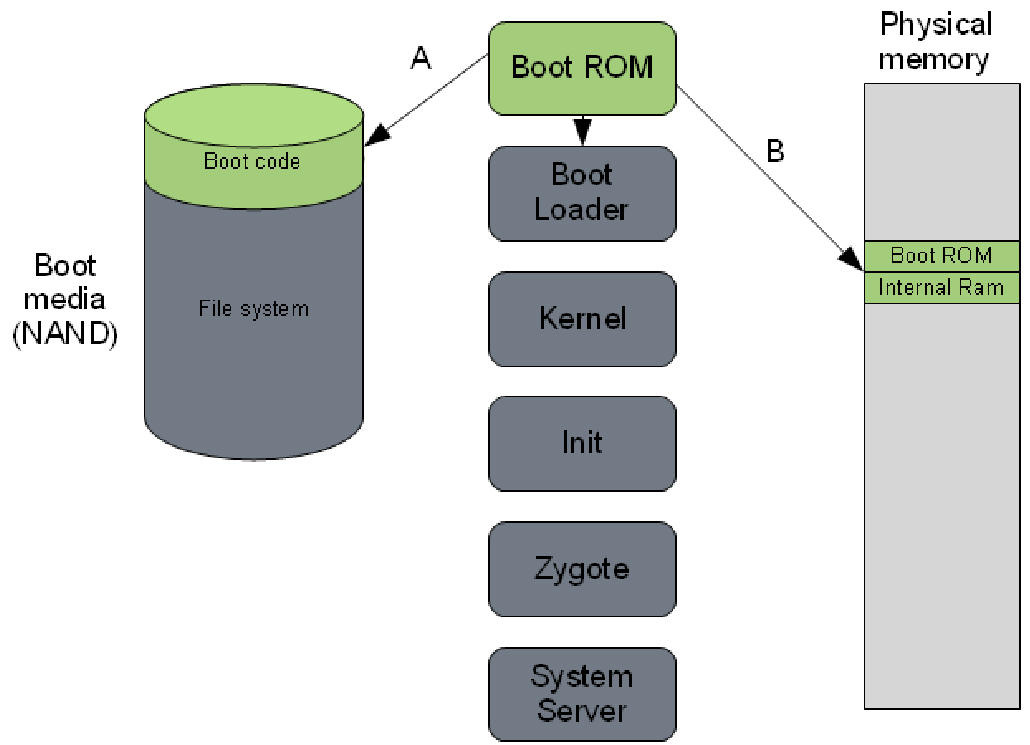
The first two steps in the initialization process are hardware-oriented; however, they require understanding of the process of configuring those system interface pins (balls on the device) that have software-configurable functionality. This configuration is an essential part of chip configuration and is application-dependent. This chapter refers to those pins and the associated configuration registers that are vital for correct device initialization.

Android boot process 1

Power on and boot ROM code execution

- Mobile platforms and embedded systems has some differences compared to Desktop systems in how they initially start up.
- At power on the CPU will be in a state where no initializations have been done. Internal clocks are not set up and the only memory available is the internal RAM.
- When power supplies are stable the execution will start with the Boot ROM code. This is a small piece of code that is hardwired in the CPU ASIC (Application Specific Integrated Circuit).

- **A.**
  The Boot ROM code will detect the boot media using a system register that maps to some physical balls on the ASIC. This is to determine where to find the first stage of the bootloader.
- **B.**
  Once the boot media sequence is established the Boot ROM will try to load the first stage bootloader to internal RAM.
- Once the bootloader is in place the Boot ROM code will perform a jump and execution continues in the bootloader.



Android boot process 2

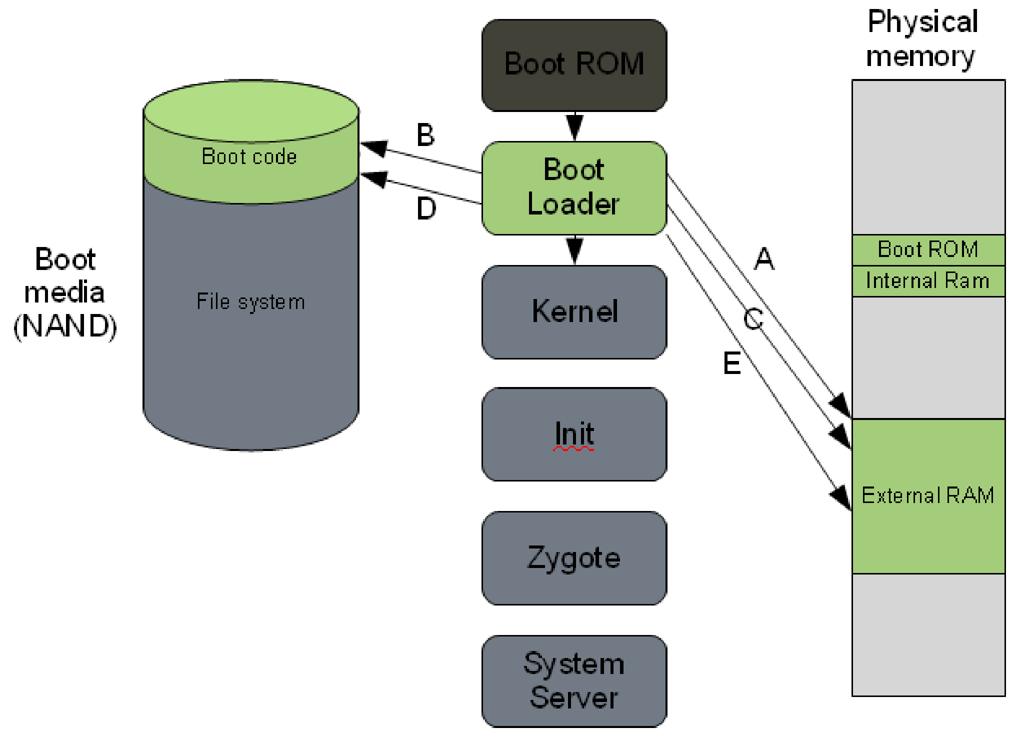
The bootloader

- The bootloader is a special program separate from the Linux kernel that is used to set up initial memories and load the kernel to RAM. On desktop systems the bootloaders are programs like GRUB. In embedded Linux uBoot is often the bootloader of choice. Device manufacturers often use their own proprietary bootloaders.

A. The first bootloader stage will detect and set up external RAM.

B. Once external RAM is available and the system is ready the to run something more significant the first stage will load the main bootloader and place it in external RAM.

- **C.** The second stage of the bootloader is the first major program that will run. This may contain code to set up file systems, additional memory, network support and other things. On a mobile phone it may also be responsible for loading code for the modem CPU and setting up low level memory protections and security options.
- **D.** Once the bootloader is done with any special tasks it will look for a Linux kernel to boot. It will load this from the boot media (or some other source depending on system configuration) and place it in the RAM. It will also place some boot parameters in memory for the kernel to read when it starts up.
- **E.** Once the bootloader is done it will perform a jump to the Linux kernel, usually some decompression routine, and the kernel assumes system responsibility.



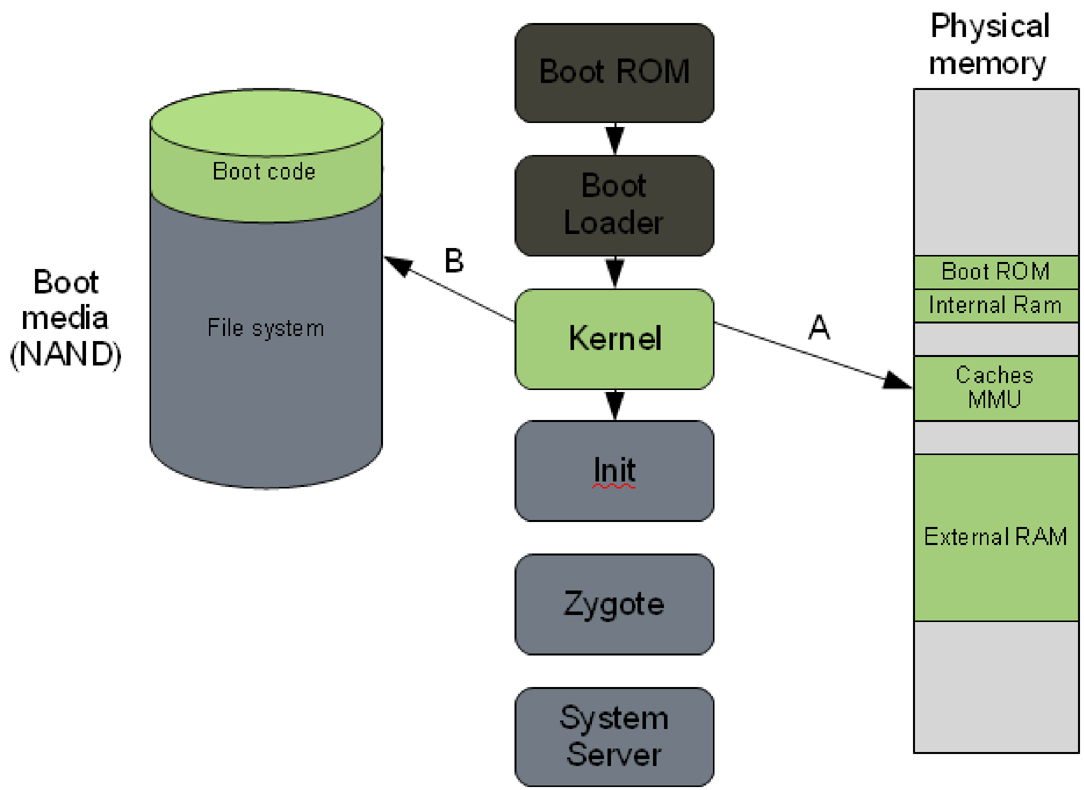
Android boot process 3

The Linux kernel

- The Linux kernel starts up in a similar way on Android as on other systems. It will set up everything that is needed for the system to run. Initialize interrupt controllers, set up memory protections, caches and scheduling.

- **A.**
  Once the memory management units and caches have been initialized the system will be able to use virtual memory and launch user space processes.

- **B.**
  The kernel will look in the root file system for the init process (found under /system/core/init in the Android open source tree) and launch it as the initial user space process.



Android boot process 4

The init process

- The init process is the “grandmother” of all system processes. Every other process in the system will be launched from this process or one of its descendants.

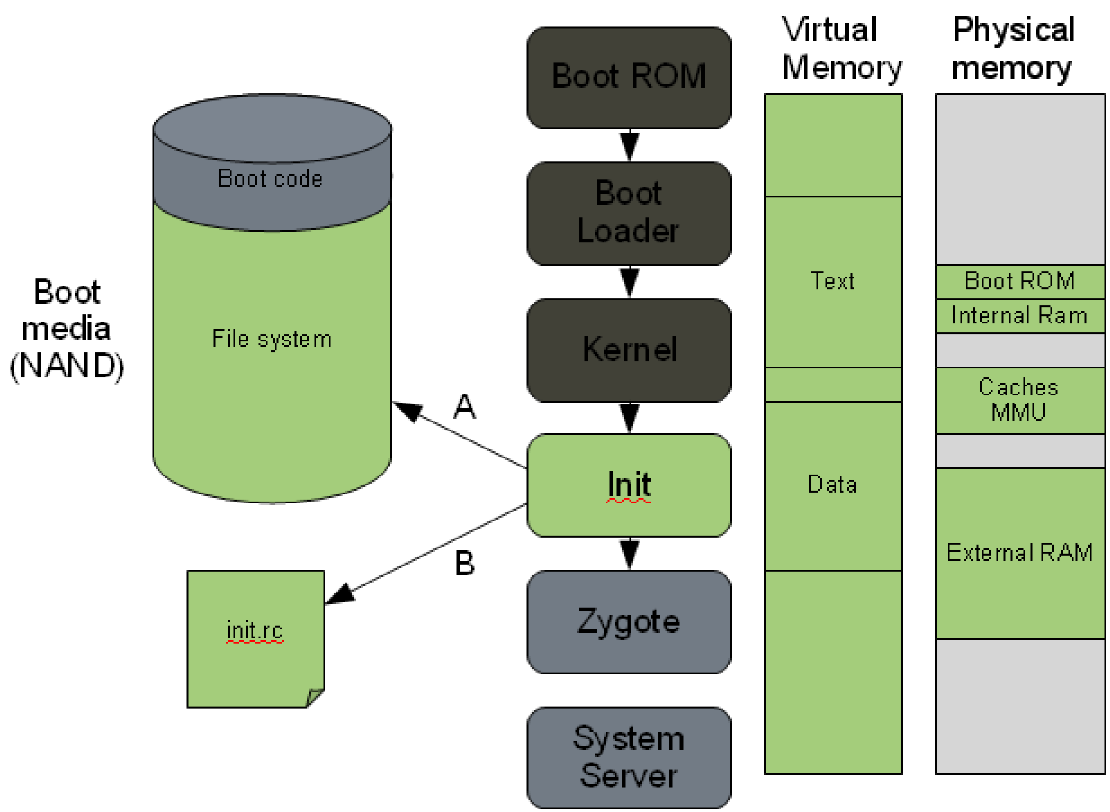
A.

The init process in Android will look for a file called init.rc. This is a script that describes the system services, file system and other parameters that need to be set up.

The init.rc script is placed in /system/core/ rootdir in the Android open source project.

B.

The init process will parse the init script and launch the system service processes.

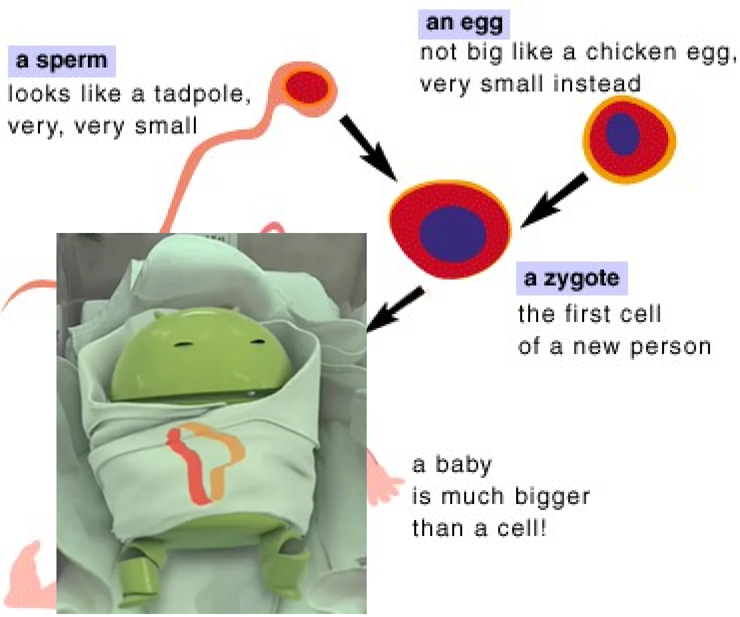


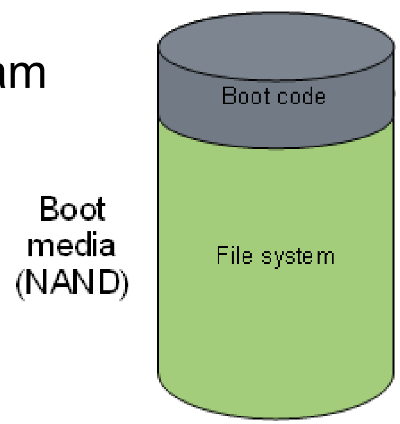
Android boot process 5

Zygote and Dalvik

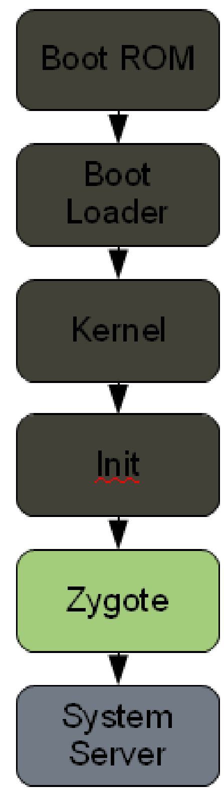
- The Zygote is launched by the init process and will basically just start executing and initialize the Dalvik VM (so .dex files can run)
  - Zygote also loads up system libraries
  - If the Zygote finds out that a new app is starting
  - Zygote forks the process, in this way giving all Dalvik VMs (and apps) access to system libraries

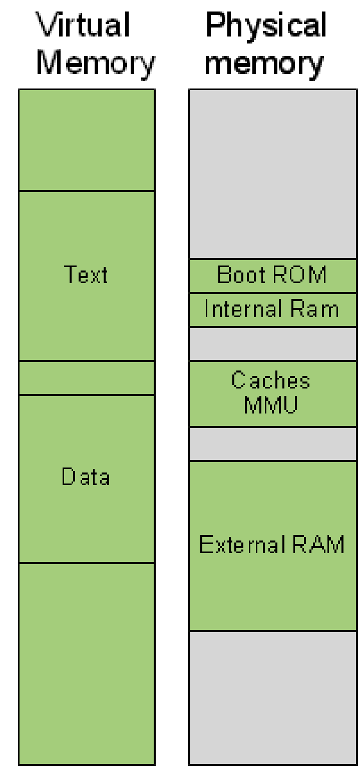
The birth of a Android apk program






```
fork()
if (child) {
    load the app java class
    run it
}
/* I'm parent */
wait and check if it's okay
```





Android boot process 6 and 7

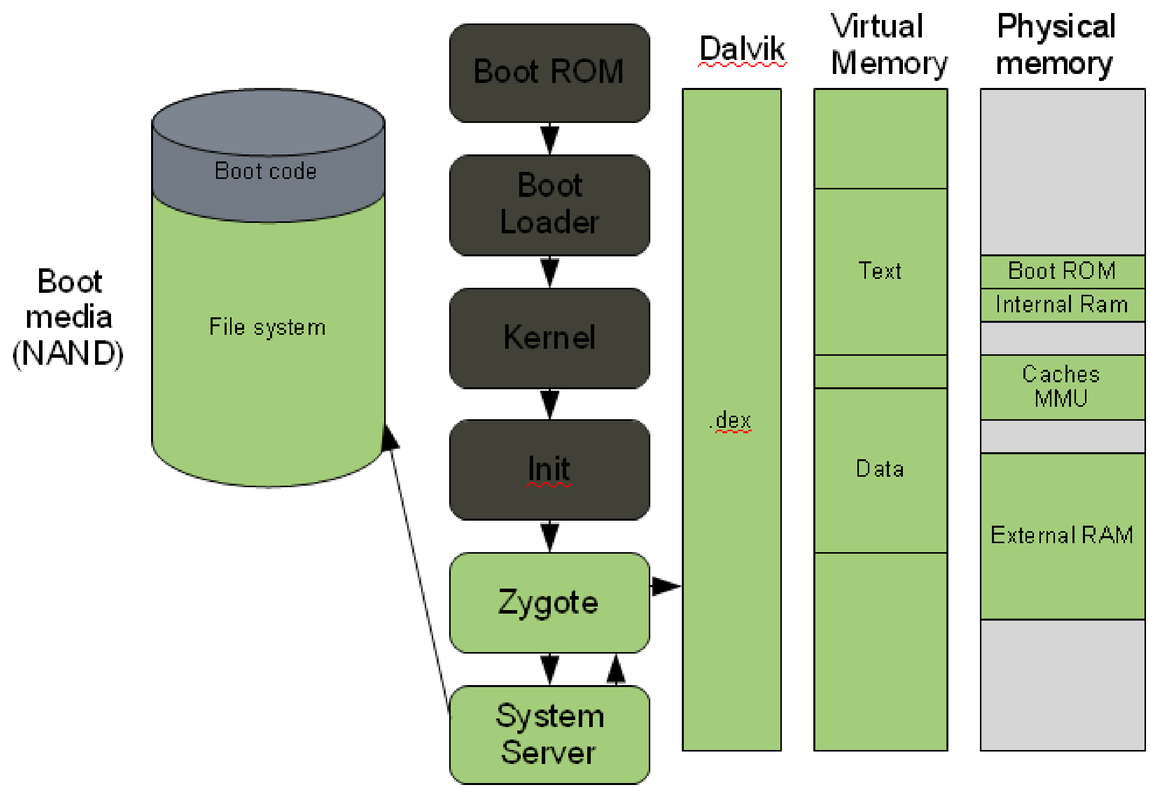
The system server

- The system server is the first java component to run in the system. It will start all the Android services such as telephony manager and bluetooth etc.
- Start up of each service is currently written directly into the run method of the system server.

- ## 7. Boot completed

- Once the System Server is up and running and the system boot has completed there is a standard broadcast action called: ACTION_BOOT_COMPLETE

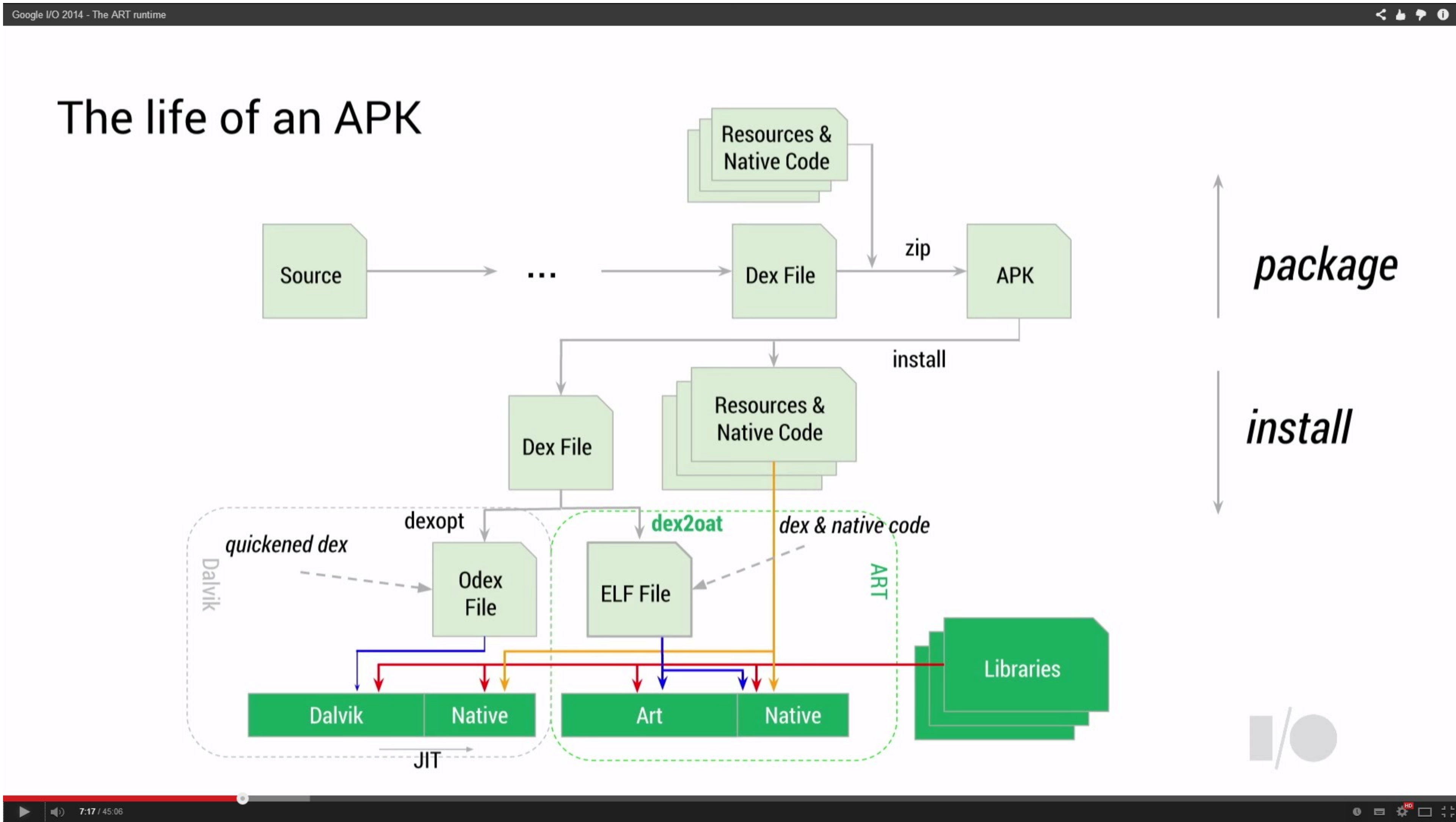
- To start your own service. For example register an alarm or otherwise make your application perform some action after boot you should register to receive this broadcast intent.



Google I/O 2014 - The ART runtime

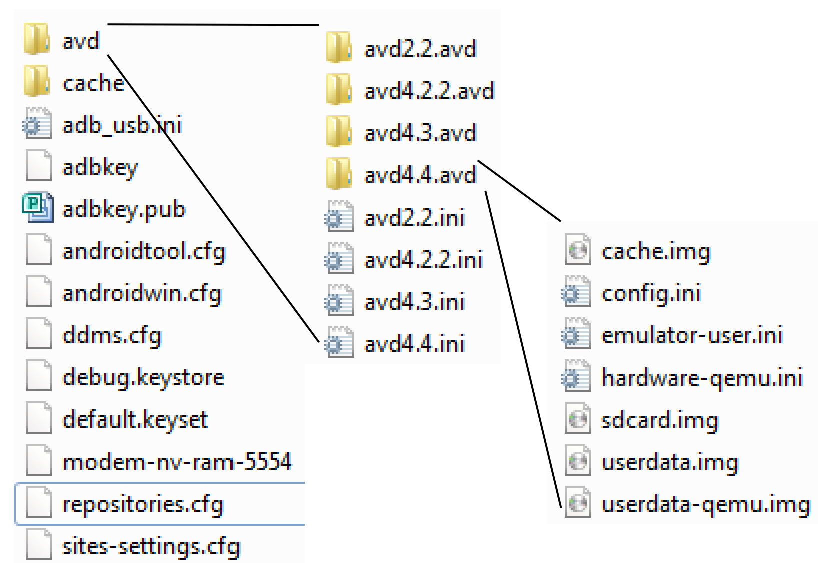
http://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l

https://www.youtube.com/watch?v=EBlTzQsUoOw



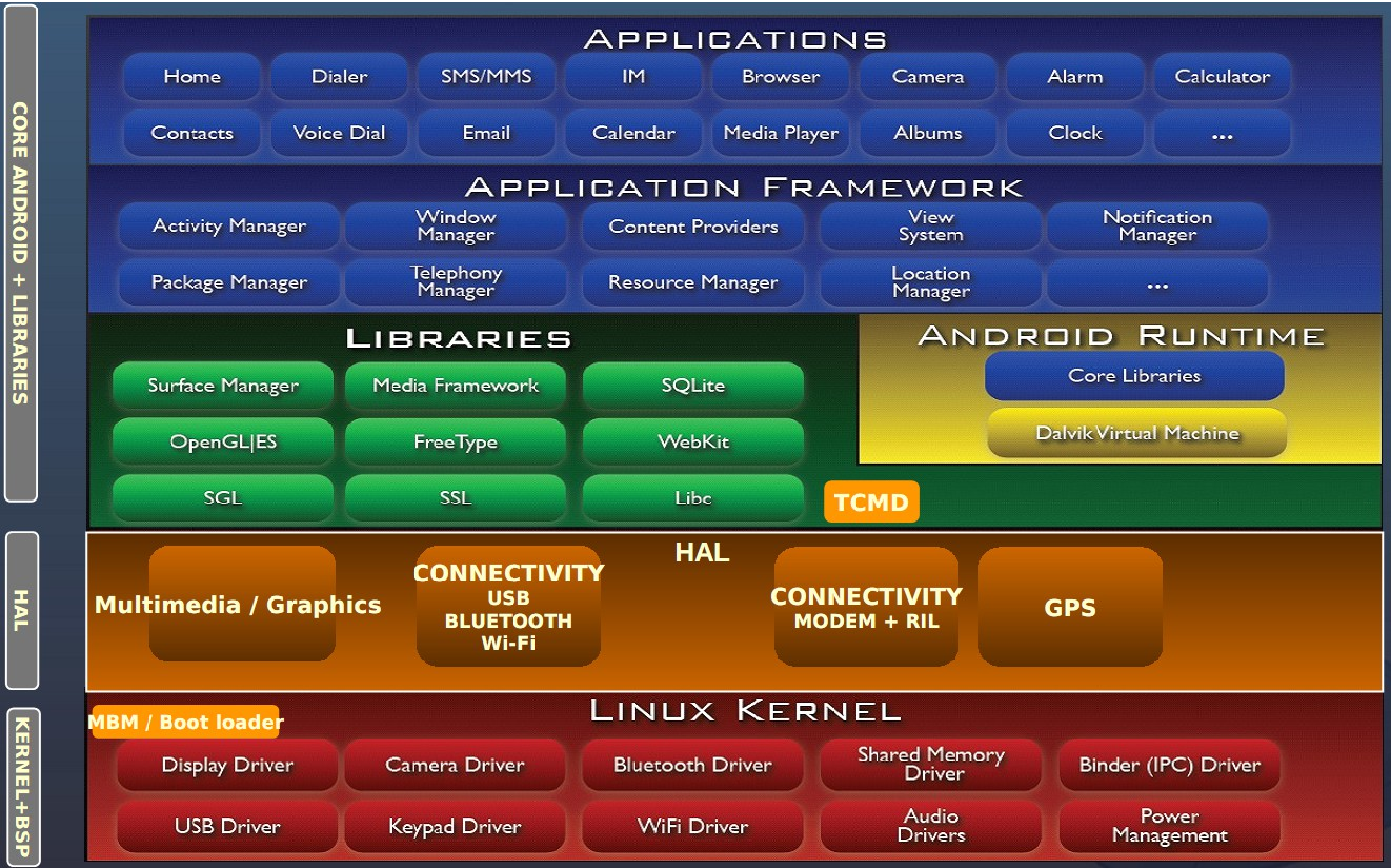
Android SDK and ADB

- The Android software development kit (SDK) provides developer tools, documentation and utilities that can assist significantly in the forensic or security analysis of a device
  - The ADB (Android Debug Bridge) is **essential** to understand
  - USB debugging turns on the adbd daemon on device which runs as root if device is rooted, otherwise as an user with only needed privileges
  - http://developer.android.com/tools/help/adb.html#commandsummary
- Forensic analysts and security engineers can learn about Android and how it operates by leveraging the emulator and examining the network, file system, and data artifacts
- AVD files
  - <users-home/username>/.android
  - System-images in SDK folder
- Dalvik VM
  - Decompile and reverse engineer .dex files
- NDK (Native Developer Kit)
  - Cross-compiled code – tools etc.



Android OS (architecture)

http://source.android.com/devices/tech/index.html



Android Core OS

http://arstechnica.com/gadgets/2014/11/android-5-0-lollipop-thoroughly-reviewed/

→ Android OS

Phone
Calculator
Clock
Downloads
Contacts
Settings
Lock screen
Navigation bar
Status bar
Notification panel
Recent apps
Power menu
Fonts
Initial setup
Application framework
Application runtime (ART)
Linux kernel & drivers
Hardware support

→ Google Play Services

Google Settings app
Ads
In-app purchases
Initial setup
Cloud-to-device messaging
Account authentication
Account syncing
Google+ sign-in
Google+ sharing APIs
Google+ photo syncing
Photosphere support
Drive APIs
Cast APIs
Maps APIs
Play Games APIs
Location APIs
Security (DRM) APIs
Wearable APIs
Wallet APIs
Fit APIs
Malware scanner
Remote wipe
Remote location
App indexing
App analytics

→ Google Play Store

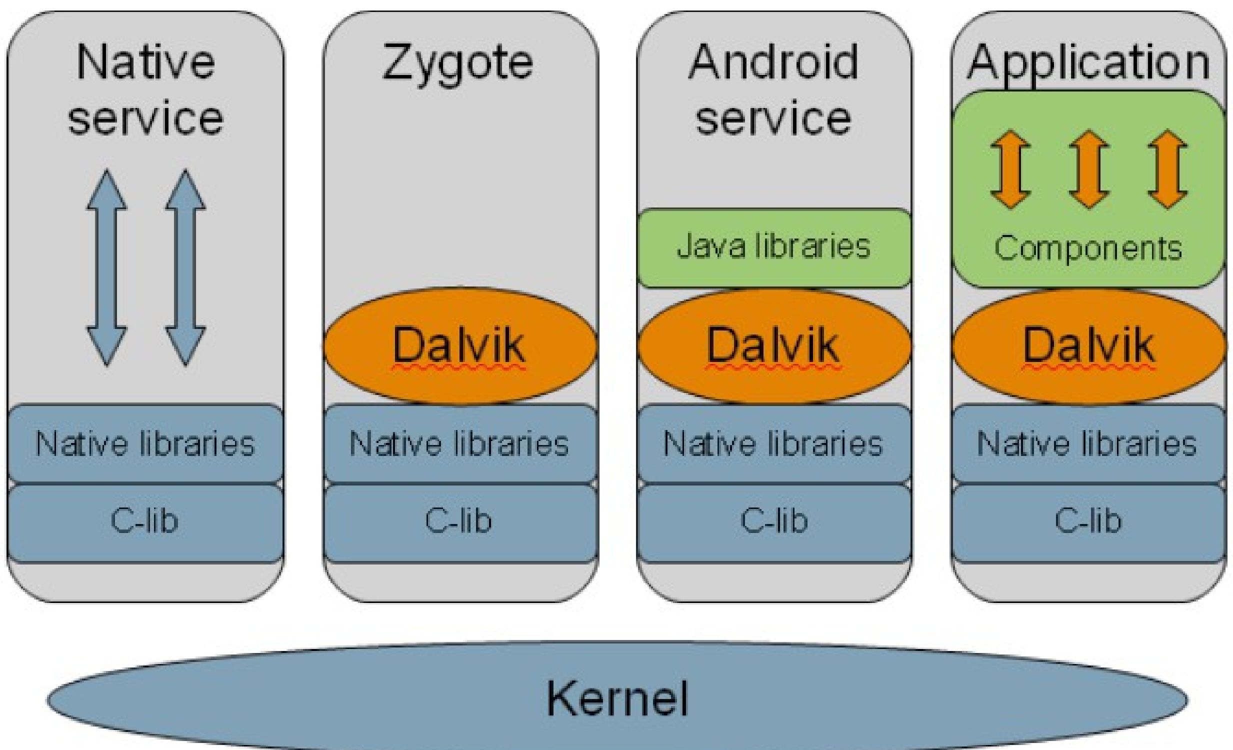
Play Store
Play Services
Google Now Launcher
Keyboard
Camera
Text-to-speech engine
Search/Now
Calendar
Chrome
Maps
Street View
Gmail
Email
Hangouts
Google+
Google+ Photos
Drive/Docs/Sheets/Slides
YouTube
Cloud Print
Keep
Wallet
Play Books
Play Music
Play Movies & TV
Play Newsstand
Play Games
WebView
Voice

Android security approach 0?

Android security approach 1

Strong base – The Linux level sandbox
And the developers digital signature

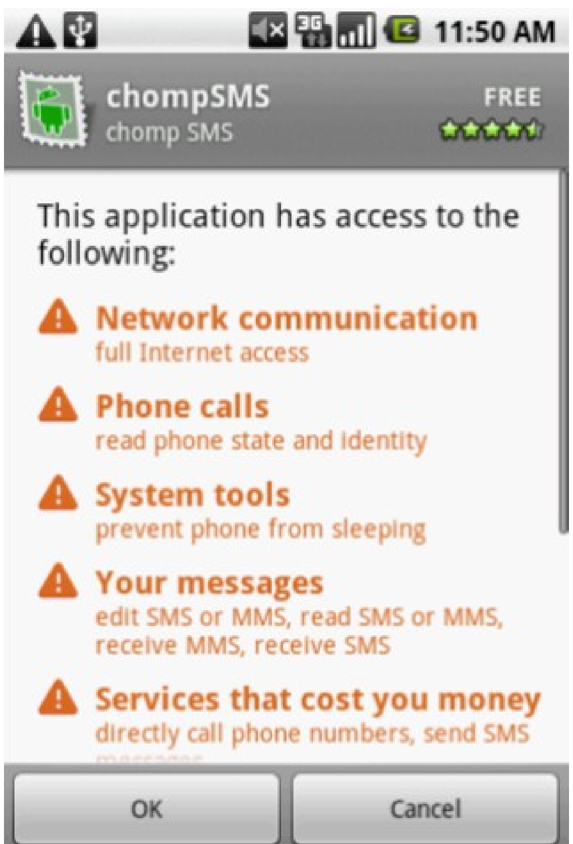
```
D:\tmp>adb shell
root@generic_x86:/ # cd /data/data
cd /data/data
root@generic_x86:/data/data # ls -al
ls -al
drwxr-x--x u0_a0     u0_a0               2013-08-29 11:20 com.android.backupconfirm
drwxr-x--x u0_a16    u0_a16              2013-09-27 14:36 com.android.browser
drwxr-x--x u0_a33    u0_a33              2013-10-03 09:41 com.android.calculator2
```

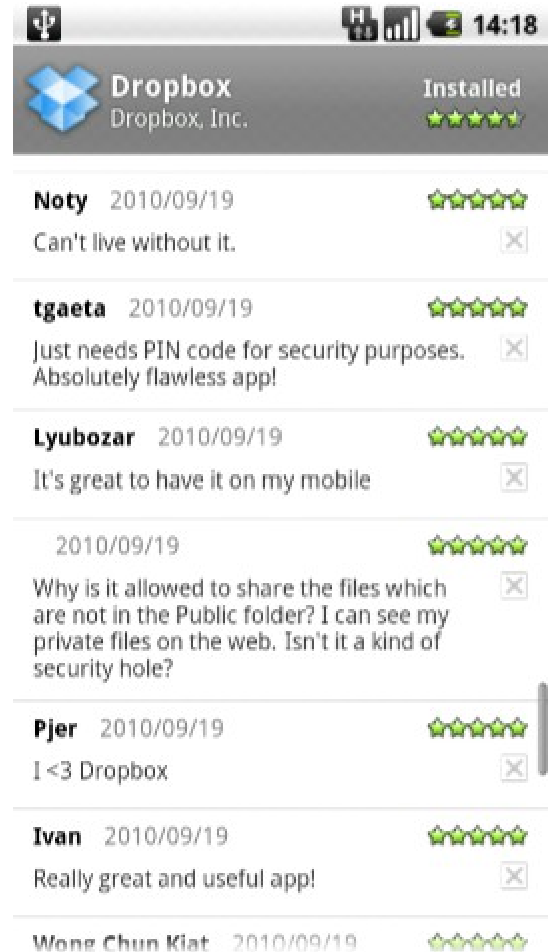


Android security approach 2

Permissions and Community/Peer review

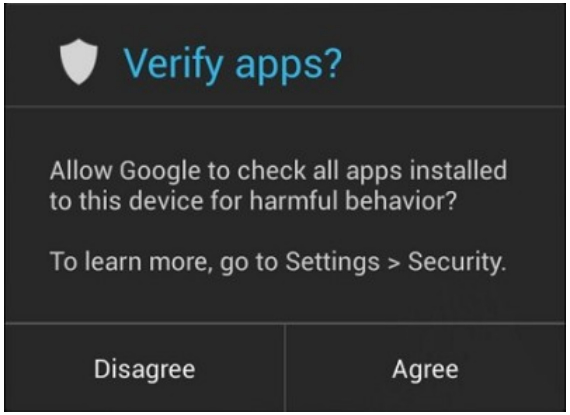
- checked at install time





Android security approach 3

- The ”Bouncer” scanning all apps on Google Play
  - Using tech from virustotal etc.
  - Simulating apps running on device
- Remotely malware removal
  - Cleaning users devices from remote
  - http://android-developers.blogspot.se/2010/06/exercising-our-remote-application.html



- Settings > Security > Verify apps
  - From Android 4.2 Jelly Bean
  - Scan apps which are ”side loaded”
  - http://support.google.com/nexus/bin/answer.py?hl=en&answer=2812636&topic=2812015&ctx=topic
- From Android 4.4 SELinux is in enforcing mode
  - http://selinuxproject.org

Android security approach 4

- From Android 5 SELinux is in **full** enforcing mode
  - In short, Android is shifting from enforcement on a limited set of crucial domains (installd, netd, vold and zygote) to everything (more than 60 domains)
- Default encryption by vold
  - New Android 5 devices is encrypted at first boot and cannot be returned to an unencrypted state
    - Howto disable encryption: http://www.xda-developers.com/android/disable-data-encryption-nexus-6/
  - Devices upgraded to Android 5 and then encrypted may be returned to an unencrypted state by factory data reset
- Dm-verity (full support in Android 5)
  - dm-verity is block level integrity check mechanism (prevent rootkits and other changes to the storage layer)
- Android Security Overview
  - https://source.android.com/devices/tech/security/index.html

Android file systems and data structures

- Android applications primarily store data in two locations, internal and external storage (emulated or real SD card)
- Internal apps data are found in the following subdirectories

Table 4.1 Common /data/data/<packageName> Subdirectories

shared_prefs	Directory Storing Shared Preferences in XML Format
lib	Custom library files an application requires
files	Files the developer saves to internal storage
cache	Files cached by the application, often cache files from the web browser or other apps that use the WebKit engine
databases	SQLite databases and journal files

- App data on external storage are usually stored in the [external_path]/Android/data/<packagename> folder
- SQLite databases are a rich source of forensic data
- Network – log files with time stamps, user name, files etc.
- Linux kernel log file (dmesg) and debug messages via logcat (system and app messages)
- Dumpsys provides information on services, memory, and other system details

ADB dumpstate and bugreport

- Dumpstate combines portions of previous debugs with system information
  - # adb shell dumpstate
- Bugreport combines logcat, dumpsys, and dumpstate debug output in a single command, and displays on screen for the purpose of submitting a bug report.

Table 4.3 Dumpstate Sections

Section	File or Command
Stack traces	N/A
Device info	N/A
System	N/A
Memory info	/proc/meminfo
Cpu info	top -n 1 -d 1 -m 30 -t
Procrank	(procrank)
Virtual memory stats	/proc/vmstat
Vmalloc info	/proc/vmallocinfo
Slab info	/proc/slabinfo
Zoneinfo	/proc/zoneinfo
System log	logcat -v time -d *:v
Event log	logcat -b events -v time -d *:v
Radio log	logcat -b radio -v time -d *:v
Network interfaces	netcfg
Network routes	/proc/net/route
Arp cache	/proc/net/arp
Dump Wi-Fi firmware log	su root dhdutil -i eth0 upload /data/local/tmp/ wlan_crash.dump
System properties	N/A
Kernel log	dmesg
Kernel wakelocks	/proc/wakelocks
Kernel cpufreq	/sys/devices/system/cpu/cpu0/cpufreq/stats/ time_in_state

Table 4.3 Dumpstate Sections *(Continued)*

Section	File or Command
Vold dump	vdc dump
Secure containers	vdc asec list
Processes	ps -p
Processes and threads	ps -t -p -p
Librank	librank
Binder failed transaction log	/proc/binder/failed_transaction_log
Binder transaction log	/proc/binder/transaction_log
Binder transactions	/proc/binder/transactions
Binder stats	/proc/binder/stats
Binder process state	sh -c cat /proc/binder/proc/* -p
File systems and free space	df
Package settings	/data/system/packages.xml: 2011-01-26 09:18:02
Package uid errors	/data/system/uiderrors.txt: 2010-11-14 22:52:26
Last kmsg	/proc/last_kmsg
Last radio log	parse_radio_log /proc/last_radio_log
Last panic console	/data/dontpanic/apanic_console
Last panic threads	/data/dontpanic/apanic_threads
Blocked process wait channels	N/A
Backlights	N/A
Dumpsys	dumpsys

Partitions and file system support

- cat proc/filesystems
  - "nodev" means virtual file system that are not written to any physical device
- df (disk free) and mount command
- cat proc/mtd and cat /proc/partitions

Table 4.5 MTD Partitions Size Conversions

Size (hex)	Name	Size (decimal, bytes)	Size (KB)	Size (MB)
0xa0000	misc	655,360	640	0.6
0x480000	recovery	4,718,592	4608	4.5
0x300000	boot	3,145,728	3072	3.0
0xf800000	system	260,046,848	253952	248.0
0xa0000	local	655,360	640	0.6
0x2800000	cache	41,943,040	40960	40.0
0x9500000	datadata	156,237,824	152576	149.0

```
ahoog@ubuntu:~$ adb shell cat /proc/mtd
dev:    size   erasesize  name
mtd0: 000a0000 00020000 "misc"
mtd1: 00480000 00020000 "recovery"
mtd2: 00300000 00020000 "boot"
mtd3: 0f800000 00020000 "system"
mtd4: 000a0000 00020000 "local"
mtd5: 02800000 00020000 "cache"
mtd6: 09500000 00020000 "datadata"
```

```
cat /proc/filesystems
nodev   sysfs
nodev   rootfs
nodev   bdev
nodev   proc
nodev   tmpfs
nodev   binfmt_misc
nodev   debugfs
nodev   sockfs
nodev   usbfs
nodev   pipefs
nodev   anon_inodefs
nodev   devpts
        ext3
        ext2
        ext4
nodev   ramfs
nodev   hugetlbfs
        vfat
        msdos
        iso9660
        fuseblk
nodev   fuse
nodev   fusectl
        yaffs
        yaffs2
nodev   mqueue
nodev   selinuxfs
```

System file systems

- **rootfs** is where the kernel mounts the root file system (the top of the directory tree, noted with a forward slash) at startup
- The **devpts** file system is used to provide simulated terminal sessions on an Android device, similar to connecting to a traditional Unix server
- **sysfs** is another virtual file system that contains configuration and control files for the device
- **cgroups** is used to track and aggregate tasks in the Linux file system
- The **proc** file system provides detailed information about kernel, processes, and configuration parameters in a structured manner
- **tmpfs** is a file system that stores all files in virtual memory backed by RAM and, if present, the swap or cache file for the device

tmpfs and eMMC

- The tmpfs is often readable by the shell user and forensic programs can be copied and executed in tmpfs without modifying the NAND flash or SD card
- The standard installation has four tmpfs mount points
  - The **/dev directory** contains device files that allow the kernel to read and write to attached devices such as NAND flash, SD card, character devices, and more
  - The **/mnt/asec** and **/mnt/sdcard/.android_secure** directories allow apps to be stored on the SD card instead of /data/data, which provides more storage
  - **/app-cache** stores cache files from web browser etc.
- Since 2011 most new devices use a regular block device (eMMC) instead of raw NAND flash
  - YAFFS is single threaded and experience bottlenecks in multi-core systems
  - Ext4 is usually used for: /system, /data and /cache, on some newer models F2FS from Samsung is used instead
  - VFAT in Linux == FAT32 and is usually mounted **/mnt/sdcard**, **/mnt/emmc**, **/storage/emulated/, /mnt/emulated, /mnt/secure/asec** (encrypted apk files), but other virtual paths can be mounted as well

Mounted file systems 1

- Running the mount command returns the mounted file systems and their options, example:
  - tmpfs /dev tmpfs rw,seclabel,nosuid,relatime,mode=755 0 0
  - The ”0 0” entry at end determines whether or not the file system is archived by the dump command and the pass number that determines the order in which the file system checker (fsck) checks the device/partition for errors at boot time.

Table 4.12 Output of Mount Command Overview

Device Name	Mount Point	File System Type	Options	Notes
rootfs	/	rootfs	ro,relatime	This is the ro (read-only) root file system mount at /
tmpfs	/dev	tmpfs	rw,relatime, mode=755	The device directory is mounted as tmpfs and has permissions set to 755 that are read, write, and execute for root (rwx) and read/execute for everyone else
/dev/block/mtdblock6	/data/data	yaffs2	rw,nosuid, nodev,relatime	While the /data directory is an ext3, the /data/data where app data is stored is a YAFFS2 file system. It is mounted to allow read/write access, does not allow setuid (which would allow other users to execute programs using the permission of file owner), does not interpret any file as a special block device, and updates the file access time if older than the modified time
/dev/block/vold/179:9	/mnt/sdcard	vfat	See SD card numbered list	See SD card numbered list

Mounted file systems 2

- The /mnt/sdcard has many options
  - /dev/block/vold/179:0 /storage/sdcard vfat rw,dirsync,nosuid,nodev,noexec,relatime,uid=1000,gid=1015,fmask=0702,dmask=0702, allow_utime=0020,codepage=cp437,iocharset=iso8859-1,shortname=mixed,utf8,errors=remount-ro 0 0

1. **rw**: mounted to allow read/write
2. **dirsync**: all updates to directories are done synchronously
3. **nosuid**: does not allow setuid (which would allow other users to execute programs using the permission of file owner)
4. **nodev**: does not interpret any file as a special block device
5. **noexec**: does not let all files execute from the file system
6. **relatime**: updates the file access time if older than the modified time
7. **uid=1000**: sets the owner of all files to 1000
8. **gid=1015**: sets the group of all files to 1015
9. **fmask=0702**: sets the umask applied to regular files only (set permissions

- - - rwxr-x, or user=none, group=read/write/execute,other=read/execute)

10. **dmask=0702**: sets the umask applied to directories only (set permissions

- - - rwxr-x, or user=none, group=read/write/execute,other=read/execute)

11. **allow_utime=0020**: controls the permission check of mtime/atime.
12. **codepage=cp437**: sets the codepage for converting to shortname characters on FAT and VFAT file systems.
13. **iocharset=iso8859-1**: character set to use for converting between 8-bit characters and 16-bit Unicode characters. The default is iso8859-1. Long file names are stored on disk in Unicode format.
14. **shortname=mixed**: defines the behavior for creation and display of file names that fit into 8.3 characters. If a long name for a file exists, it will always be the preferred display. Mixed displays the short name as is and stores a long name when the short name is not all upper case.
15. **utf8**: converts 16-bit Unicode characters on CD to UTF-8.
16. **errors=remount-ro**: defines the behavior when an error is encountered; in this case, remounts the file system read-only.

Partition layout for EMMC based devices

- There is no /proc/mtd on emmc
- It may be difficult to connect a partition with a name (data, system, recovery etc.)
- The mount command just gives a by-name reference for all mounts as
  - /dev/block/platform/msm_sdcc.1/**by-name**/userdata /data ext4 rw,nosuid,nodev, ...
- Some units have /proc/emmc or /proc/dumchar_info populated with this info
- Some units have it revealed under the /sys/devices by the Linux kernel
- Sometimes you have to extract the recovery.fstab file from a recovery image
- Read more
  - https://github.com/ameer1234567890/OnlineNandroid/wiki/How-To-Gather-Information-About-Partition-Layouts

```
cat /proc/partitions
179  0 15388672 mmcblk0
179  1    65536 mmcblk0p1
179  2      512 mmcblk0p2
179  3      512 mmcblk0p3
179  4     2048 mmcblk0p4
179  5      512 mmcblk0p5
179  6    22528 mmcblk0p6
179  7    22528 mmcblk0p7
179  8      780 mmcblk0p8
179  9      780 mmcblk0p9
179 10      780 mmcblk0p10
179 11      512 mmcblk0p11
179 12      512 mmcblk0p12
179 13      512 mmcblk0p13
179 14     2048 mmcblk0p14
179 15      512 mmcblk0p15
179 16      512 mmcblk0p16
179 17      512 mmcblk0p17
179 18      512 mmcblk0p18
179 19    16384 mmcblk0p19
179 20    16384 mmcblk0p20
179 21   860160 mmcblk0p21
179 22   573440 mmcblk0p22
179 23 13798400 mmcblk0p23
179 24      512 mmcblk0p24
179 25      495 mmcblk0p25
```