Android Training

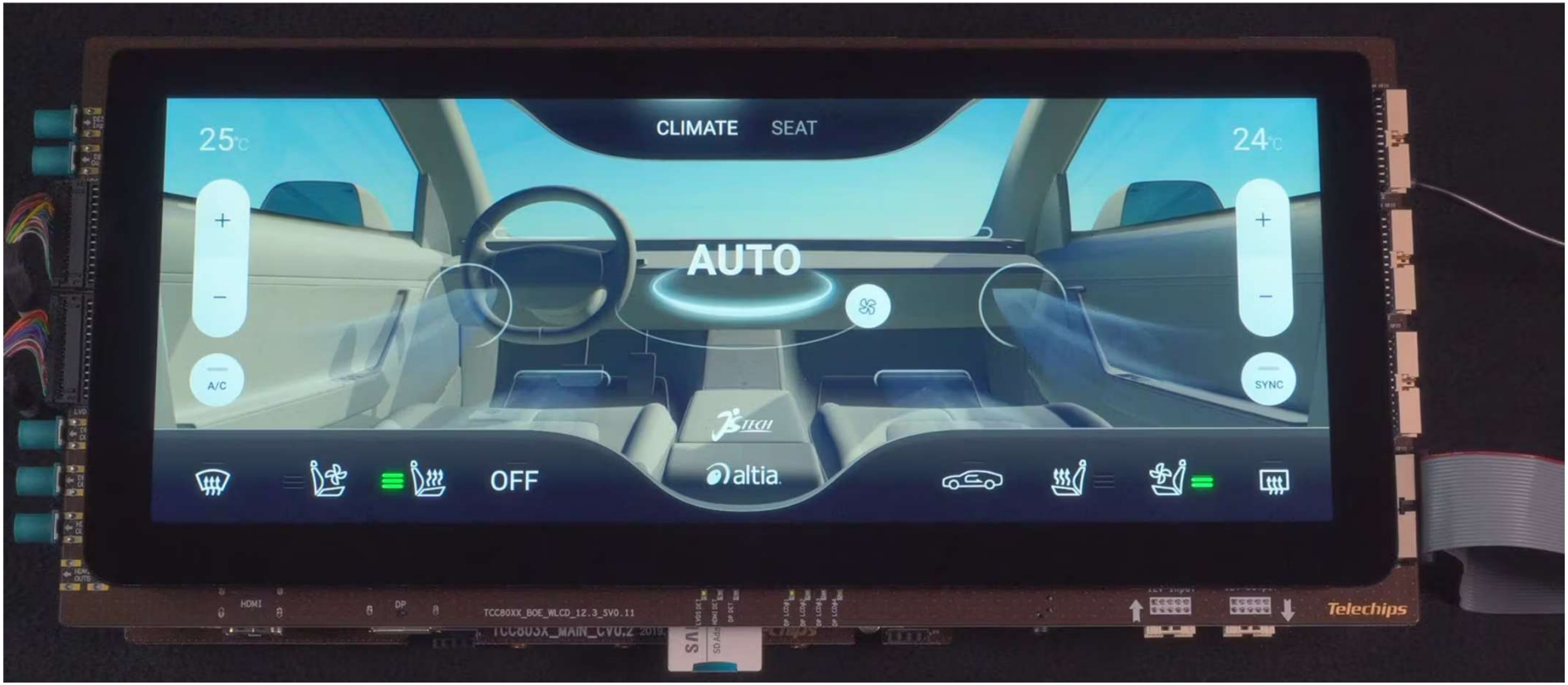
Day	Date	Content Delivery Date	Topics	Comments
Day 1 :	08 November 2025	Nov 4th	> Creating new device and adding Lunch option in AAOS > Benefits of AIDL as Interface Description in Android-14 > Transitioning from HIDL to AIDL > AAOS Media Player - Architecture - Media Source detection and streaming - Play/puase, next/prev, fwd/rewind event flow and handling > Audio Manager: - Source switching - Audio attributes and priority management - Early Audio - IVI and cluster sounds	All the media player needs to be develeoped on android, also include, Source Switiching, Audio Attributes, early audio.
Day2:	15 November 2025	Nov 11th	> Using SOME/IP in integrated cockpit enviornment - Service discovery - Comm between integrated cockpit ECUs > Using VOIP in integrated cockpit enviornment > IPC Mechanism between AAOS Layers > IPC between IVI and Cluster using Telechip SoC > Multidisplay mgmt , android and cluster (trainer will check and revert) > Integrating 3rd party apps like Spotify, YouTube etc. with AAOS	As Spotify integration can happen either with GAS or by having Spotify Automotive SDK from Spotify. Hence, we will go through the steps of integration only (with and without GAS).
Day3:	22 November 2025	Nov 18th	Security Measures in Android Automotive - Secure boot, encryption methods, and secure communication. - SE Linux security policies - Secure signing process - Multiuser Management - Setting up user authentication (PIN-based).	
Day4:	29 November 2025	Nov 25th	- Android Bootup optimization - V2X - GAS - Future trends	All theoritical discussions
Day 5	6th December 2025		Recap session	
	HW/SW setup		Platform Setup The Hardware Platform: Telechip D3 TCX805x EVB IVI: Android Automotive OS-14 Cluster: Linux (Yocto release provided by Telechip)	

Android

Day#1 - Recap

Agenda

- Recap session
- Platform Setup
- The Hardware Platform: Telechip D3 TCX805x EVB
- IVI: Android Automotive OS-14
- Cluster: Linux (Yocto release provided by Telechip)


CLIMATE
SEAT
25°C
24°C
AUTO
A/C
SYNC
OFF
altia
HDMI
DP
TCC80XX_BOE_WLCD_12.3_SV0.11
Telechips

IVI

IVI (In-vehicle Infotainment) is a system that delivers a combination of information and entertainment content/services. Moreover, infotainment systems are in-built car computers that combine a wide range of functions – from digital radios to in-built reversing cameras.

The IVI can be described as a combination of vehicle systems which are used to deliver entertainment and information to the driver and the passengers through audio/ video interfaces, control elements like touch screen displays, button panel, voice commands, and more.

IC

An instrument cluster is a board with several different gauges and warning lights mounted in it. The instrument cluster is mounted in the dashboard, just behind the steering wheel. The driver depends on the gauges and indicators to keep tabs on the vehicle’s status. Some of the most common gauges and indicators include:

- Speedometer
- Tachometer
- Fuel gauge
- Oil pressure gauge
- Odometer
- Turn signal indicators

Telechip TCC805x Development Platform

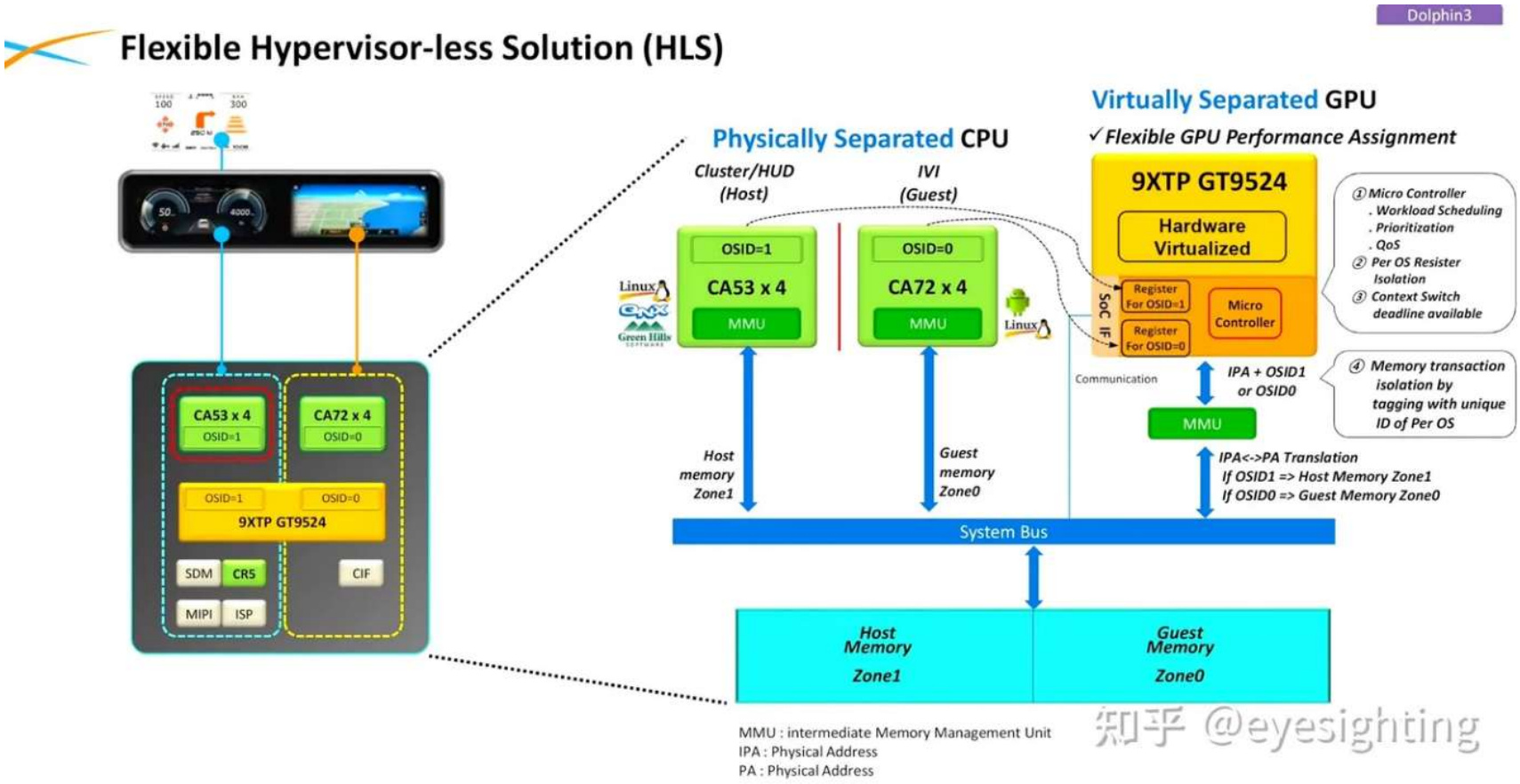
TCC8050_53 Evaluation Board (EVB) is based on the TCC805x (Dolphin3) Processor family which is ideal (SoC) System on Chip, targeted at IVI/Cockpit and ADAS,

TCC805x is based on **Arm® Cortex®-A72 Quad core** and **Arm Cortex-A53 Quad core**. It provides great performance of 2D/3D graphic engine for rich and vivid GUI with Imagination PowerVR series9XTP Core and supports HW Virtualization for Hypervisor-Less Cockpit (HLC) solution.

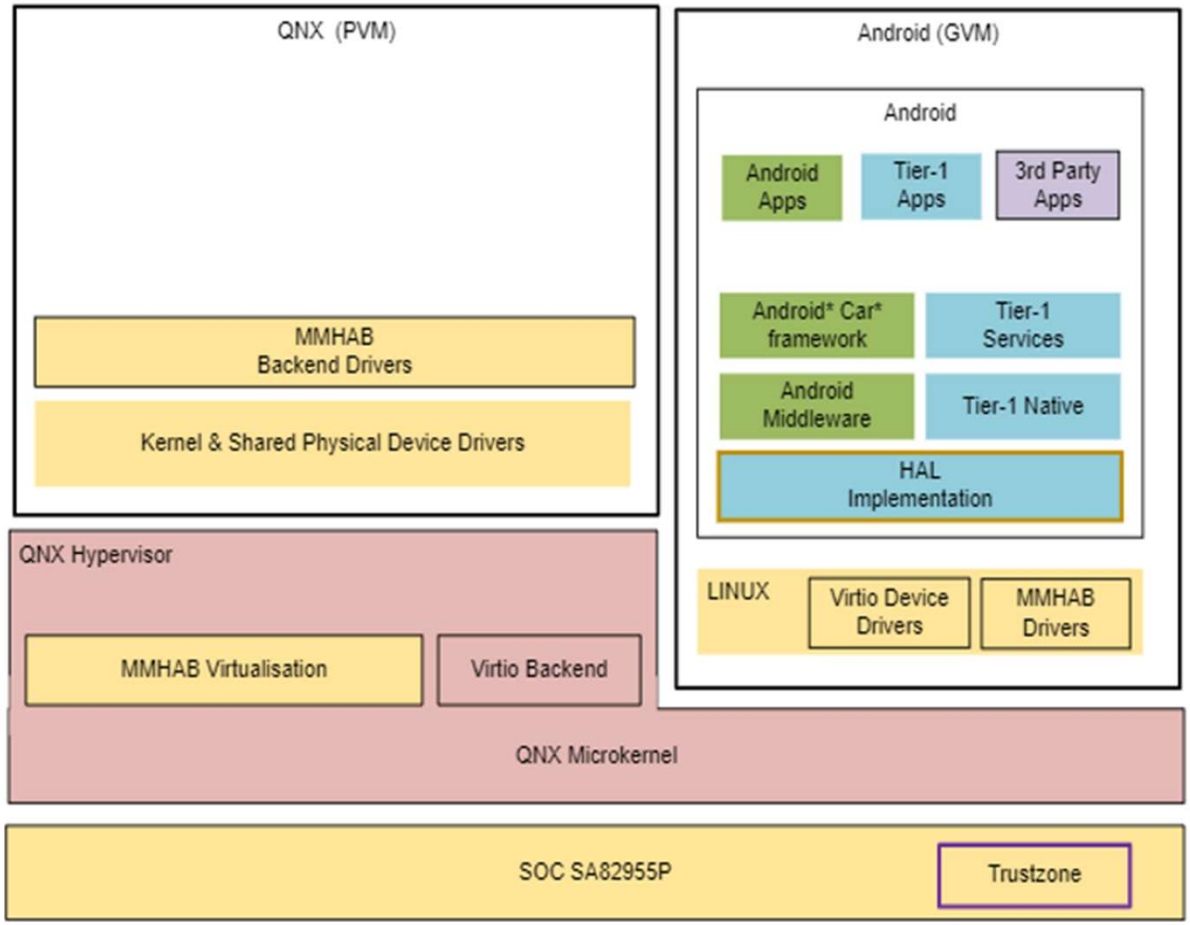
TCC805x supports not only multi-display and multi-channel camera input, but also embodies Image Signal Processing sub-system and MICOM sub-system that supports isolated safety island.

In addition, TCC805x can run operating systems such as Android™, Linux®, and QNX®.

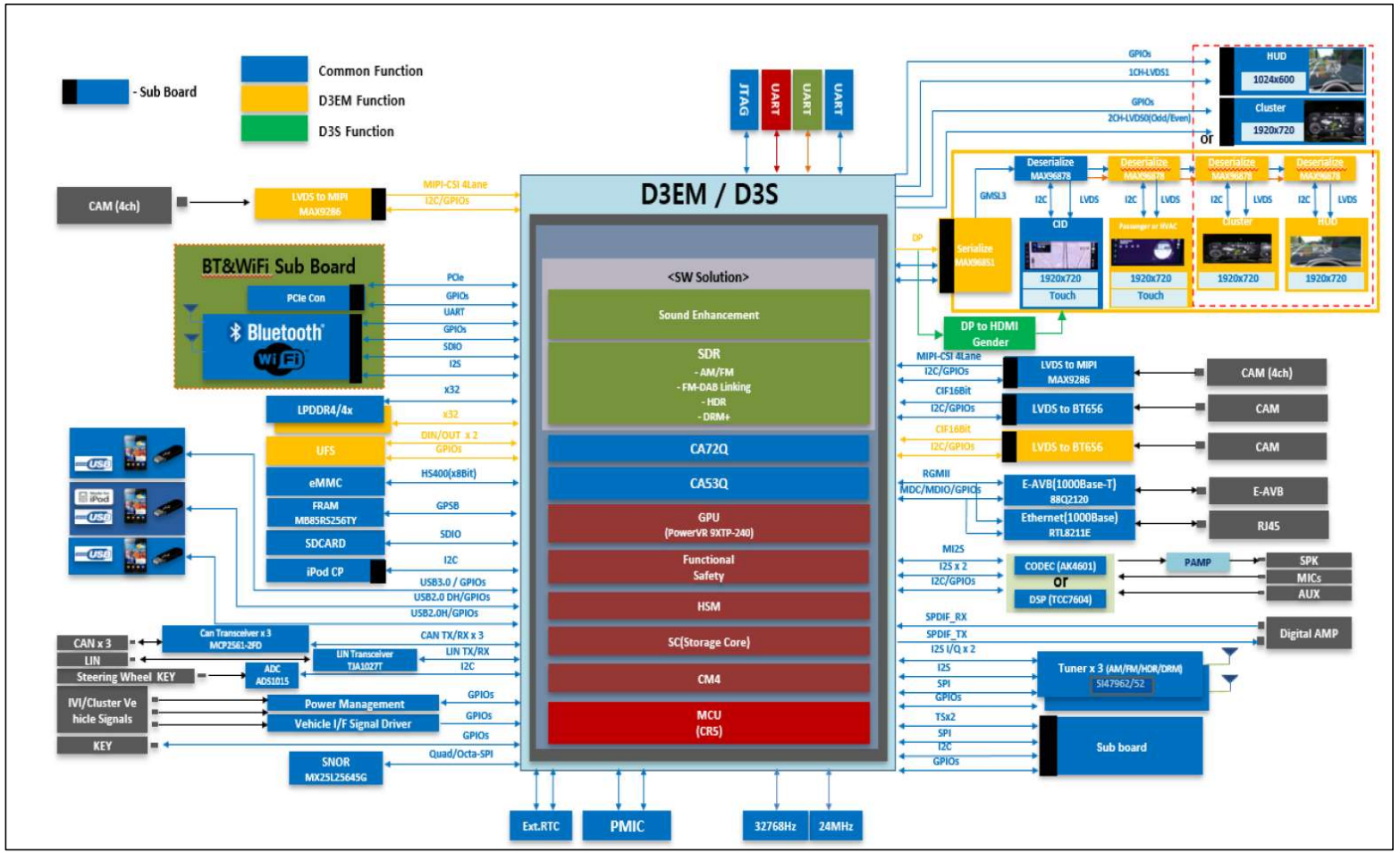
Hypervisor-less Solutions



Qualcomm SA82955 Development Platform



Telechip TCC805x Development Platform



Day 1 Agenda

1. Introduction to Android
2. Android Version and History
3. Android Eco-system
4. Android Components
5. Installation of Android Studio - setup Environment - SDK - Emulator
6. Android Architecture
7. Language used for Development
8. Android Application Development - Basic Components
9. Android AOSP Folder Structure and Important Modules
10. Android Build System
11. Android Advance topics
12. Introduction to Android Automotive
13. Overview of Android Automotive
14. History, evolution, and key features.
15. Android Automotive vs. Android Auto.
16. Overview of Android Automotive OS vs. Android Auto.
17. Core differences, use cases, and market adoption.
18. Architecture of Android Automotive OS
19. Core Components and Layers

Practical Session =>

1. Setting Up Android Automotive Environment on Laptop
2. Configuring and exploring the Android Automotive environment.
3. Downloading and flashing Android Automotive for Telechips HW.
4. Set up the Android Automotive environment on Telechips and explore its features
5. Device Configuration and Build option

A few things to keep in mind:

- There isn't a single human being that understands all of Android
- Even within the "Android team(s)", very specialized know-how
- Despite years of work on this, I'm still learning
- Use source.android.com ("official" internals "book“ for Android)
- To learn Android Seek people that share their know-how

From where to learn Android ?

https://source.android.com/

Introduction to Android

- Android is an operating system based on Linux, specifically developed for mobile devices such as cell-phones and tablets.
- includes all the frameworks and that let you create and deploy your own custom applications
- Andy Rubin is the founder of Android Inc. and developed the Android Operating system in 2003.
- In 2005 Google acquired the Android operating system from Android Inc
- Currently Google is Developing and Maintaining the Android
- The android operating system is used on more than 3 billion active devices across the globe

Timeline 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., cofounded
by Andy Rubin (CEO)
Brin/Page met Rubin at Stanford talk 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 ~50M$

2007 - November:
Open Handset Alliance announced along with Android

Android Version and History

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

Android Version and History

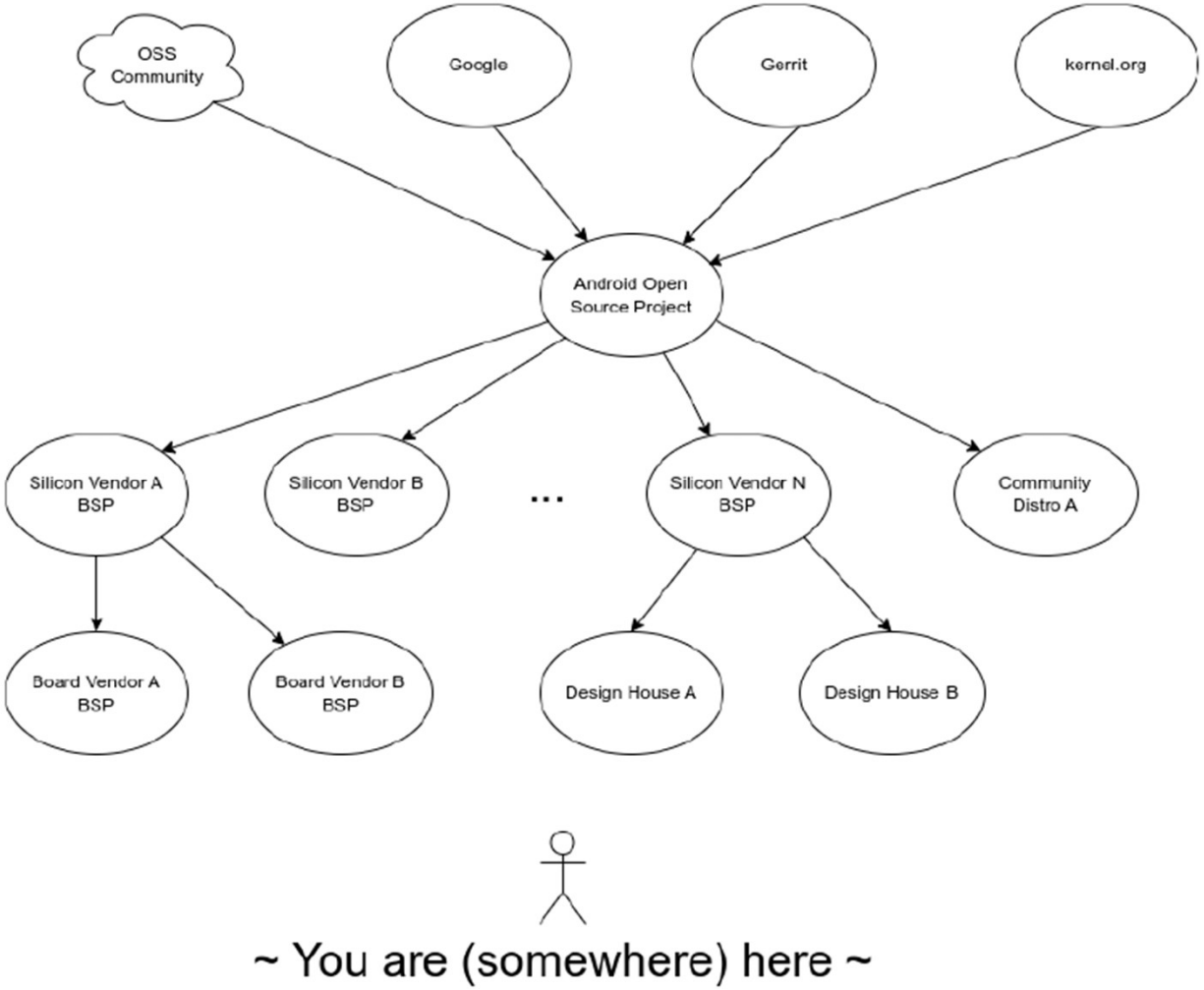
2012 – Nov: Android 4.2 / Jelly Bean – Multiuser support
2013 – July: Android 4.3 / Jelly Bean -- BLE /GLES 3.0
2013 – Oct: Android 4.4 / KitKat -- Low RAM optimizations
2014 – Nov: Android 5.0 / Lollipop - ART and other pixie dust
2015 – Mar: Android 5.1 / Lollipop - Multi-SIM card
2015 – Oct: Android 6.0 / Marshmallow – New permission model
2016 – Aug: Android 7.0 / Nougat – Multiwindow
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, multicamera API, indoor wifi positioning
2019 – Aug: Android 10.0 / "No name" – Apex
2020 – Sept: Android 11.0 – Generic Kernel Image
2021 – Oct: Android 12.0 – CrosVM
2022 – Aug: Android 13.0 – Kleaf kernel build

Android Version and History

- The android version names starts English alphabets and with sweet names, to identify the name they give API levels.
- Every Year Google Releases new version with new features, bug fixes & Security patches
- Each release information and details we can see in Google Website
- **Currently Latest Version is Android 15**

Android Version and History

Android Open-Source Project Ecosystem Flow



As a AOSP, BSP developer how it matters to me

Post shipping considerations: what now?

- How long will Google support this AOSP?
- How long will my SoC vendor povide updates?
- How long is my kernel supported for?
- What about OTA / Upgrade?
  - Mechanism
  - Backend
  - Fallback
- What about runtime device monitoring?
- What about device provisioning?
- etc.?

Android Eco-system

Android Ecosystem key components are,

- Android OS & Regular Updates
- Google Services and Apps
- Android app development
- Customization and ecosystem Integration
- Security and Privacy
- Google Devices (in next slide)

Android Eco-system

- First Android Phone is HTC Dream (T-Mobile G1) developed in September 2008
- First Android Tablet is Archos 5 Internet Tablet introduced in September 2009
- First Android Automotive is Car Audi A8 introduced in 2011
- The First Android Watch is LG G Watch in July 2014
- The First Smart Home/TV is Nexus Player 2014

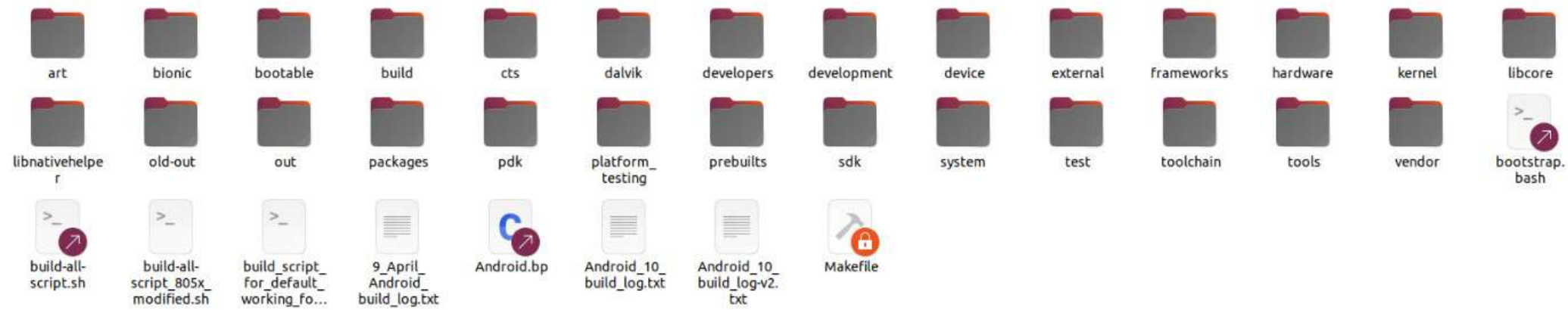
Android Download and Build

https://source.android.com/docs/setup/download

Get "repo":
Fetch the AOSP:
Make sure you fetch a tagged release, say 13.0:
http://android.googlesource.com/

```
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~
$ chmod a+x ~/bin/repo
$ repo init -u https://android.googlesource.com/platform/manifest
> -b android13-gsi
$ repo sync
```

Telechip Android 10 folder Structure



Android Directory Structure

Directory	Description
art	Android Runtime
bionic	Android's Implementation of C library. (it's LibC library's replacement)
bootable	OTA and update tools (bootloader, diskinstaller, recovery)
buid	The main entry point of the android build system
cts	Compatibility Test Suite
dalvik	Dalvik VM, Delvik tools and Core libraries
development	Development tools, Platform engineering tools and sample apps
device	Device-specific files and components
external	Copy of external projects used by AOSP

Android Directory Structure

Directory	Description
framework	System services, android. *, Android-related cmds, etc.
hardware	Hardware support libs, HAL, HIDL
libcore	Test code for dalvik, DOM, JSON, and other support files Apache Harmony
ndk	Build scripts & helper files for building the NDK
packages	Stock Android apps, providers, etc.
prebuilt	Prebuilt binaries, cross compilations toolchains for different development machines
sdk	The SDK
system	System core files (commands, debugging, bootstat)
out	Build generated directory, output is placed here

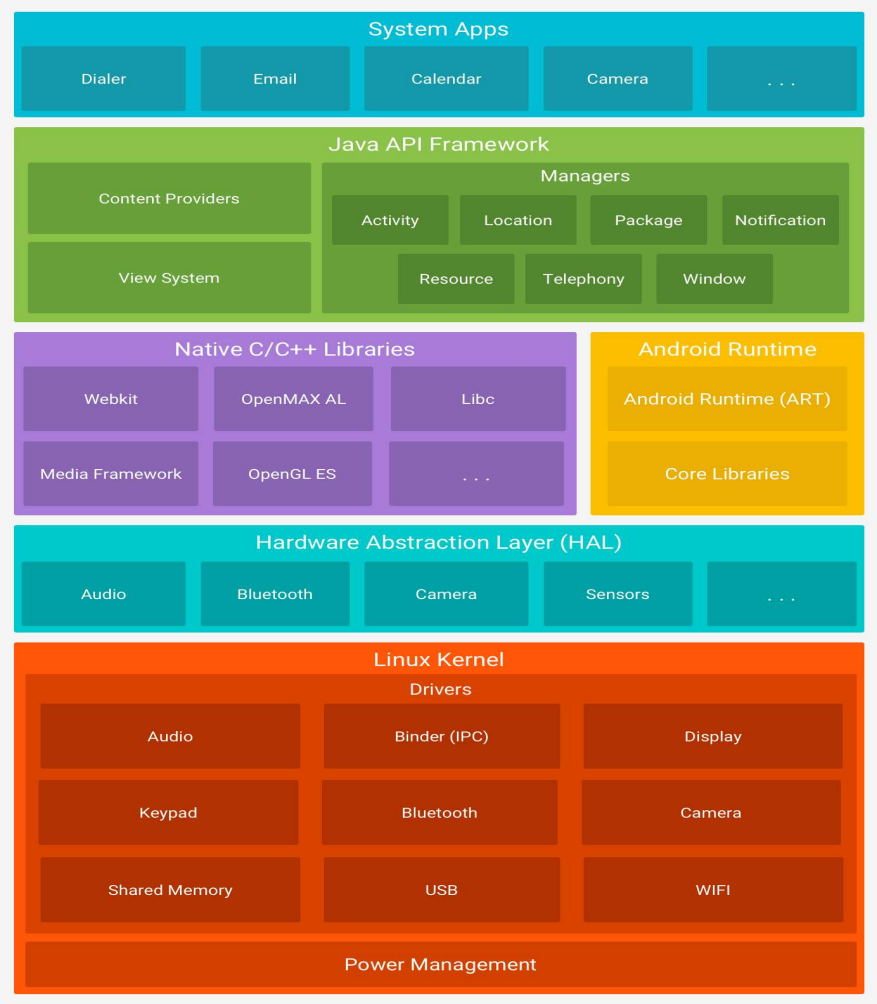
Android Directory Structure

Set up build environment:
$. build/envsetup.sh
$ lunch

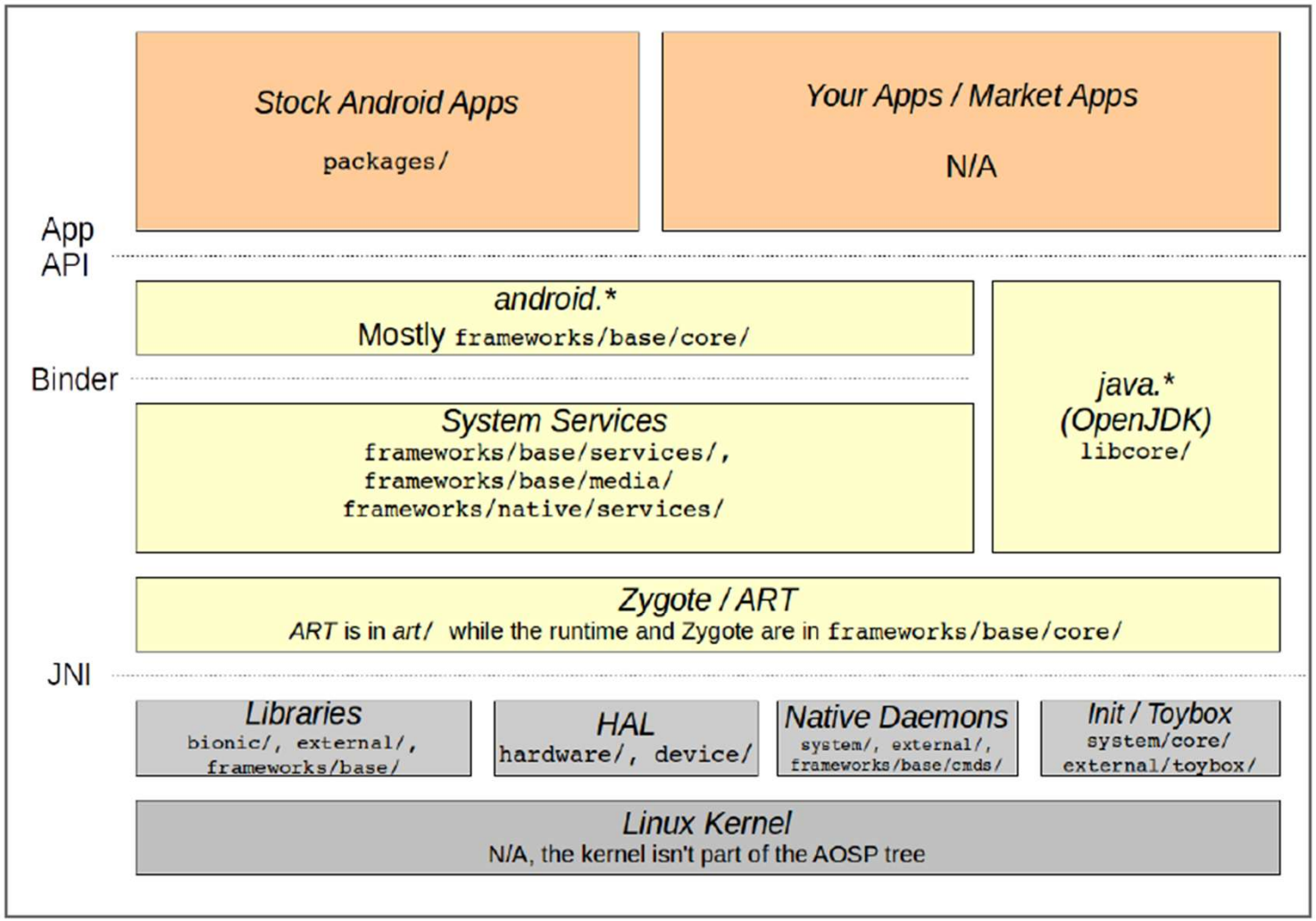
Build AOSP :
$ make –j8

Android Architecture

- Application Layer : System Apps and 3rd Party Apps
- Android Framework Layer : Framework resources
- Android Libs and ART Layer : Android Code compilation and also provides environment for applications developing in C or C++
- HAL Layer : Hardware devices interaction start
- Linux Kernel Layer : Base layer where Android Bootup will start



Android Directory Structure



Oracle Vs Google

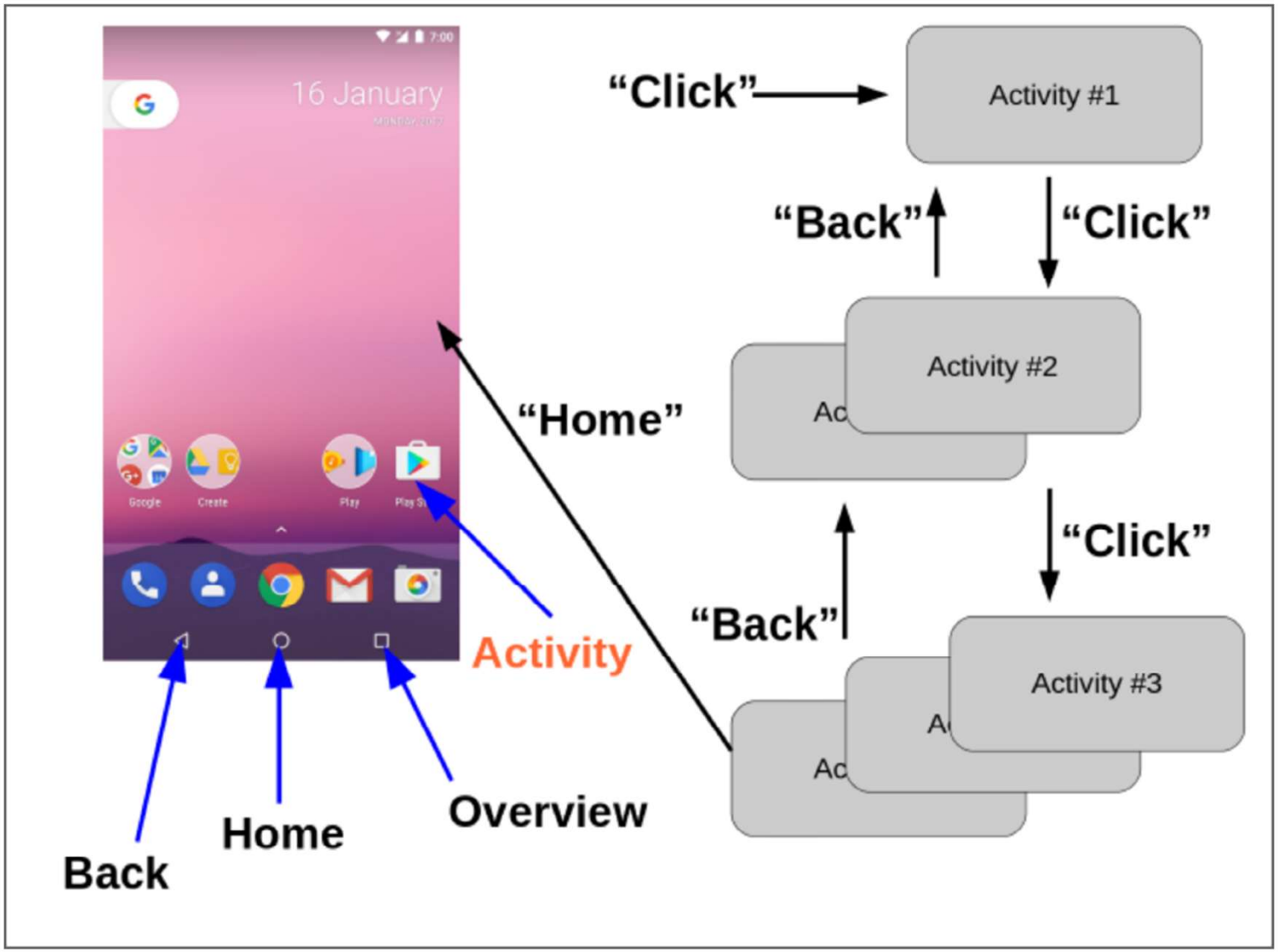
Oracle v. Google

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.

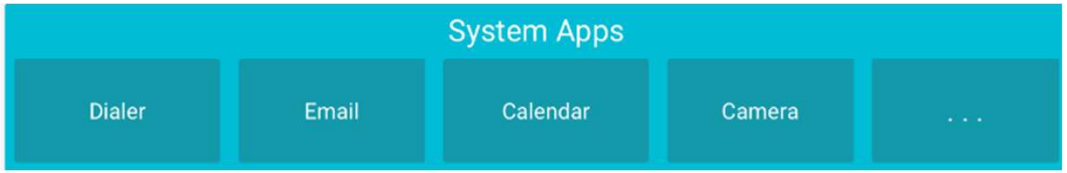
In October 2010, IBM left Apache Harmony to
join work on Oracle's OpenJDK, leaving the project practically orphaned.
...
In Spring of 2012 Oracle lost both on Copyright and Patent fronts
Oracle appealed


16 January
Activity
Back
Home
Overview
"Click"
Activity #1
"Back"
"Click"
Activity #2
"Home"
"Click"
"Back"
Activity #3

Android Architecture

Android Applications

Role：The top layer where developers create and manage Android applications using the Android Framework APIs.



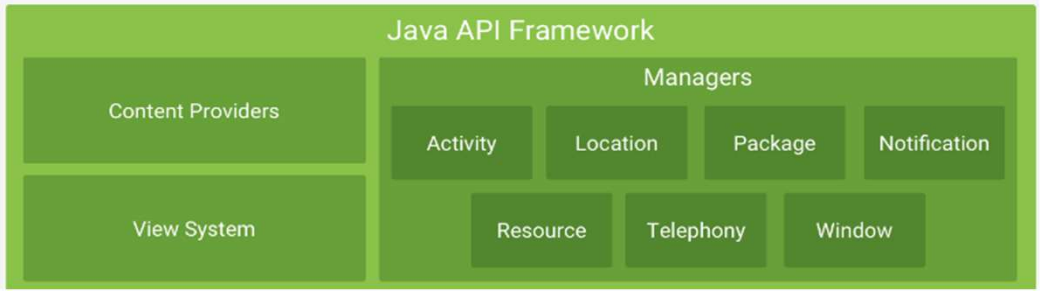
Components :

- Activities: Represents a single screen with a user interface.
- Services: Handles background tasks and operations.
- Broadcast Receivers: Responds to broadcast messages from the system or other applications.
- Content Providers: Manages and provides access to application data
- .

Android Architecture

Android Framework

Role：Provides high-level APIs for building Android applications. It offers the essential tools and services needed for app development



Components :

- Activity Manager: Manages the lifecycle of activities and tasks, handles task switching and process management.
- Window Manager: Manages the display and layout of windows and views on the screen.
- Package Manager: Handles application installation, uninstallation, and management of app permissions.
- Resource Manager: Manages app resources like strings, layouts, and drawables.
- Notification Manager: Handles notifications and alerts to the user.
- Content Providers: Facilitates data sharing between applications.

Android Architecture

Android Run Time (ART)

Role : Executes Android applications. ART has replaced Dalvik VM as the runtime environment for running apps.

Components :

- ART (Android Runtime): Includes AOT (Ahead-Of-Time) and JIT (Just-In-Time) compilation to optimize app performance.
- Garbage Collector: Manages memory allocation and deallocation for efficient memory usage.

Android Architecture

Native Libraries C/C++

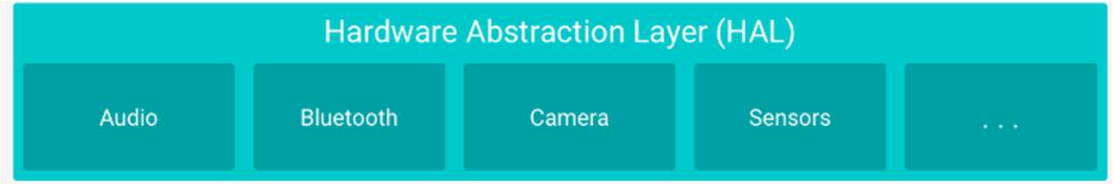
Role : Provide essential functionalities and services needed by Android applications and system components

Components :

- Core Libraries: Include Java-based libraries for common functionalities (e.g., collections, I/O, networking).
- Media Framework: Handles audio, video, and graphics processing (e.g., ExoPlayer, OpenGL ES).
- SQLite: Provides a lightweight database for local data storage.
- WebKit: Used for rendering web content in WebView

Android Architecture

HAL



Role:
HAL provides a standardized interface
between the hardware and the Android framework, allowing Android to communicate with different hardware components .

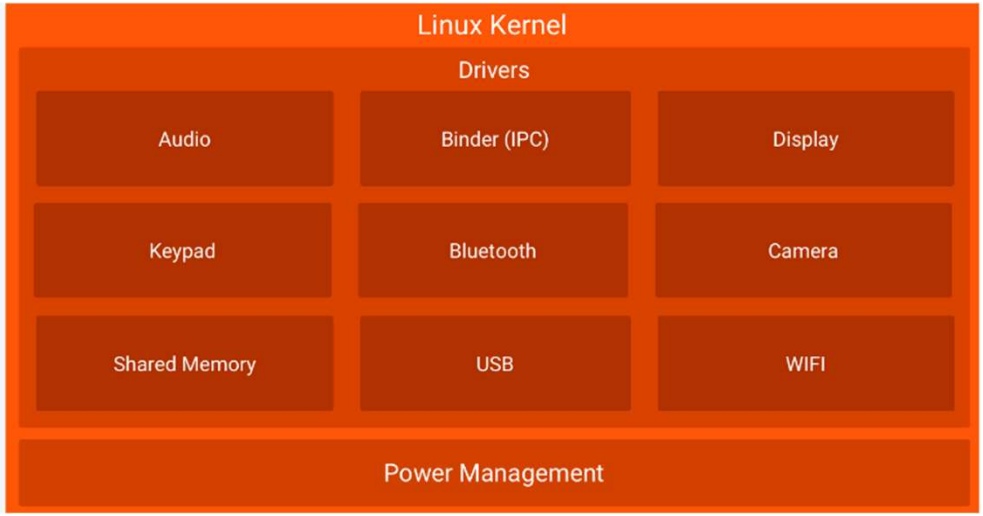
Components :

- HAL Interfaces: Define methods for interacting with hardware components such as sensors, cameras, and GPS

- Implementation: Each hardware vendor provides specific implementations for these interfaces, ensuring compatibility with the Android framework.

Android Architecture

Linux Kernel



Role:

The Linux kernel is the core of the Android operating system .It handles low-level operations and system services.

Components :

- Kernel: Manages hardware abstraction, process management, memory management and system calls.
- Drivers: Provides drivers for various hardware components like the display, camera, and network interfaces.
- Hardware Abstraction: Manages communication between hardware and the higher-level Android framework

Languages used for Development

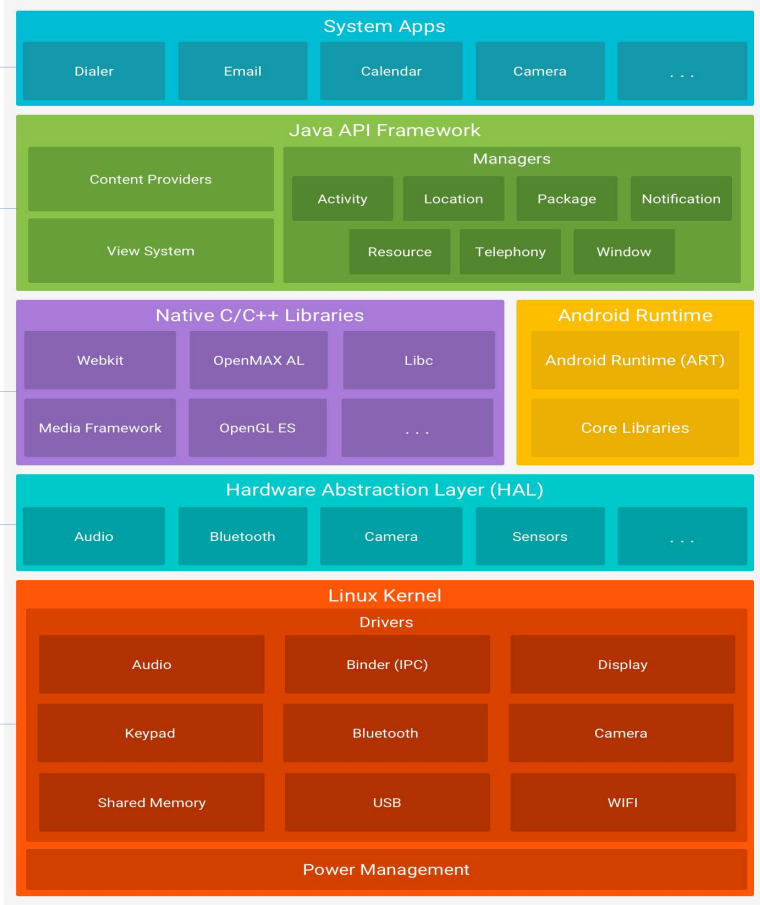
Java /Kotlin

Java /Kotlin for new Applications

C/C++

C/C++

Native C



Android Studio, SDK & Emulator

Android Studio is the IDE where developers write, test, and debug their applications. It integrates various tools and utilities needed for Android development.

Android SDK provides the core tools and libraries required for building Android apps. It includes essential components like platform tools, build tools, and libraries that Android Studio uses to compile and run applications.

Android Emulator is a powerful tool for Android developers, providing a versatile environment for testing and debugging applications across various devices and Android versions

Android Studio, SDK & Emulator

Android Studio

- Android Studio is official IDE used for Android Application Development
- Android Studio is Developed and Supported by Google & it is Open Source
- It integrates code editing, debugging, performance monitoring, and testing tools into a single environment, providing developers with everything needed to create high-quality Android applications

- How to Download Android Studio? Link
- How to Install the Android Studio ? Link

Android Studio, SDK & Emulator

Android Studio Overview

1 – Toolbar

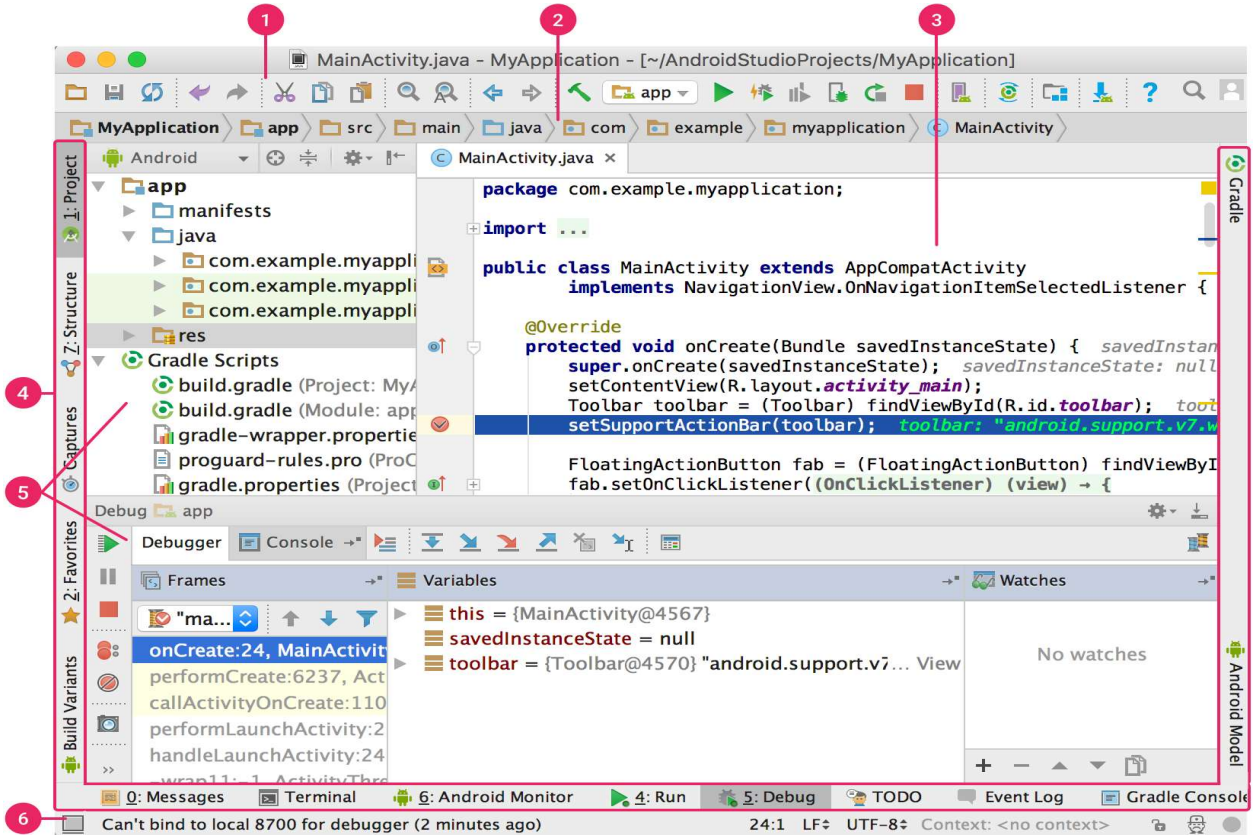
2 - Navigation Bar

3 – Editor window

4 – Toolbar Winodw

5- Tool Windows

6 – Status Bar



Android Studio, SDK & Emulator

Android SDK

- The Android SDK is a vital resource for Android developers
- Android SDK Providing all the necessary tools, libraries, and documentation to build, test, and debug Android applications
- Android includes components like SDK tools, platform tools, libraries, documentation, and the Android Emulator, all of which are essential for developing high-quality Android apps.
- We can Download SDK 2 ways, Using Android Studio & Manual Download from The Android Developer Website link below.

- How to Download Android SDK? Link

Android Studio, SDK & Emulator

Android Emulator

- Emulator is like Virtual Device in Android Studio
- We can’t have all the devices(Different Versions,Different Sizes) we used for development means Physical Devices
- With the Android Emulators we can Navigate, Debug , Build & Run our applications
- We can create emulators for Various Android Devices like Mobile, Tablet , Watch and Automotive

Q & A

Thank you

Android Automotive

AAOSP Agenda

1. Introduction to Android Automotive
2. Overview of Android Automotive
3. History, evolution, and key features.
4. Android Automotive vs. Android Auto.
5. Overview of Android Automotive OS vs. Android Auto.
6. Core differences, use cases, and market adoption.
7. Architecture of Android Automotive OS
8. Core Components and Layers

Practical Session =>

1. Setting Up Android Automotive Environment on Laptop
2. Configuring and exploring the Android Automotive environment.
3. Downloading and flashing Android Automotive for Telechips HW.
4. Set up the Android Automotive environment on Telechips and explore its features

Why Use Android in Cars?

Automotive Industry Expectations for In-Car OS:

- Intuitive Touch Interface
- Vast App & Developer Ecosystem:
- Always Connected

User Expectations for In-Car OS:

- Real-Time Updated Data
- Voice Assistants
- Customizable Interfaces

Why Android selected for Automotive ?

- **Open-Source Flexibility:**
  - Customizable Platform
  - Rapid Innovation
  - Cost-Effective Development

- **Strong Ecosystem & Future-Proofing:**
  - Google Automotive Services (GAS)
  - AI & ML Integration

Challenges of Using Android in Cars ?

Challenges:

- Existing Car Environment
- Security Concerns
- Long Lifecycle Challenges
- Efficient Resource Management and Power Consumption
- Rear-View Camera & Real-Time Requirements
- Google’s Influence
- Device Longevity vs. Software Evolution
- Regulatory & Compliance Hurdles

AAOSP Boot time

- Typically > 20 seconds
- But there are regulatory requirements for some critical services to be available within a few seconds, e.g. exterior camera

AOSP Software update cycle

- AOSP has major release every 12 months, security updates every month, each major release supported for 3 years
- SoC vendors support kernel and drivers for typically 3 years

AAOSP Hardware Requirements

- Android requires fast CPU, lots of RAM, lots of flash storage
- Requirements increase every year with each new release

AAOSP Flash Memory lifespan

- Lifespan of vehicle > 10 years
- **eMMC Flash Memory**, commonly used in automotive systems may wear out in < 5 years
  - Why ? Becoz Flash memory has a **limited number of erase/write cycles** (typically **3,000 cycles** for standard eMMC).
  - For example, writing 32GB per day on 16GB eMMC with 3k erase/write cycles
- Scenario is worse because of flash memory **write amplification**, caused by garbage collection and wear levelling
  - Writes to flash = writes from host * amplification factor

Automotive Android

Android Auto

NOTE => This is NOT Android Automotive

- Android Auto is:
  - A proprietary SDK written mostly in C++ with support for Linux, Android, and QNX
  - A protocol for connecting an Android phone to a car head unit
  - Projects apps from your phone to the car's infotainment system.
  - Requires a connected smartphone to function.
  - It is based on Android Open Accessory Protocol over USB and (recently) WiFi also used
- Uses Protocol Buffers to divide data into channels:
  - output from phone: display data (H.264), audio (PCM)
  - input to phone: car sensors and inputs

Android Automotive OS (AAOSP)

- Android Automotive OS is part of AOSP
- But, it is not a production-ready solution
- You need:
  - integration with the vehicle: CAN bus, sensors, audio, power, ...
  - apps that integrate with the vehicle, e.g. to turn lights on and off
  - home screen (the default is only a demo)
  - app store, navigation and voice recognition
  - back-end services for all of the above
- Google has a solution ...

Google Automotive Services (GAS)

- Non-free services on top of AOSP
  - similar to Google Mobile Services (GMS) in the smartphone world
- Includes
  - Play Store
  - Google Assistant
  - Google Maps
- Per-unit license
- Must pass tests: CTS, VTS, ATS, ..
- Must install Google apps

No GAS

- Without GAS, you need to find alternative apps and services
  - typically, a combination of in-house and third party
  - some tier one companies have SDKs that you can use

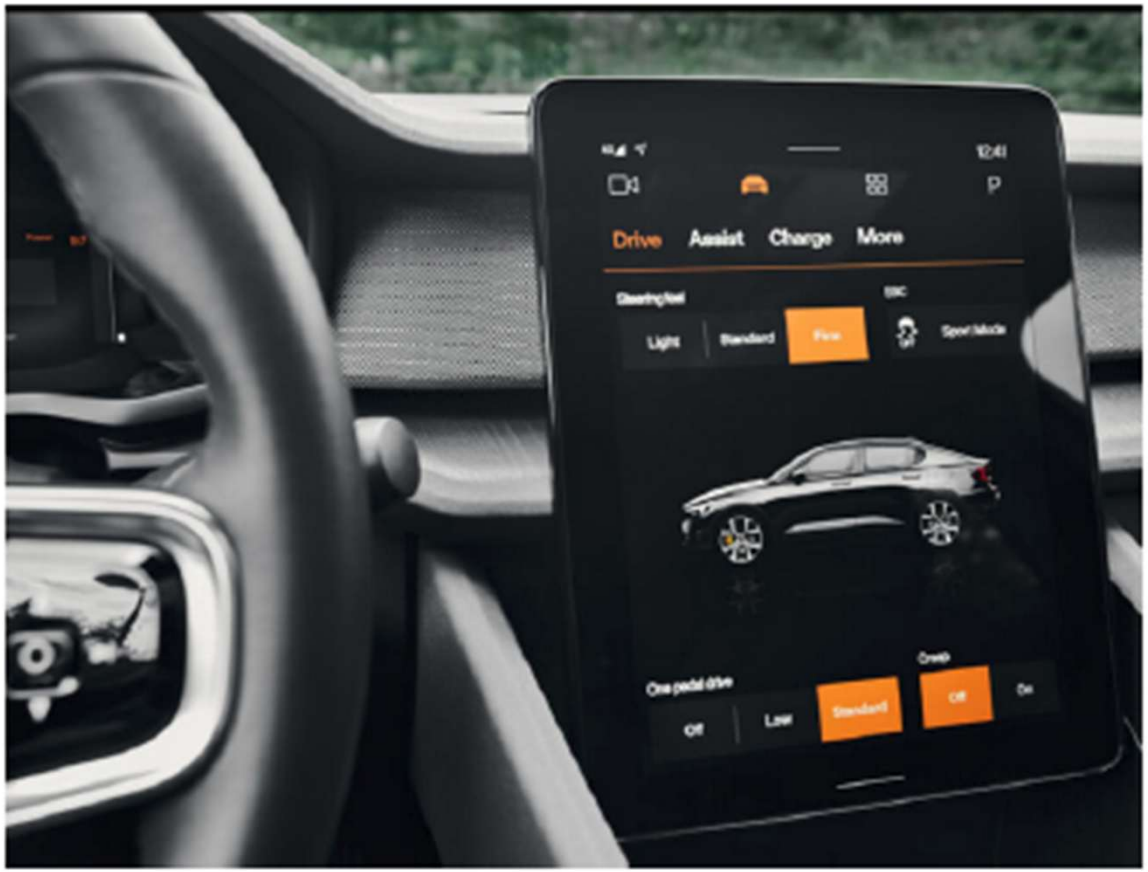
Android Auto vs Automotive AOSP

2014: **Android Auto**

- Screen cast / projection from smart phone to head unit display
- An SDK integrated into the head unit (which is usually not running Android)
- Mirrors apps like Google Maps, Music, and Messages from your phone to your car's display.
- Apple CarPlay is a similar concept

2017: **Android Automotive OS**

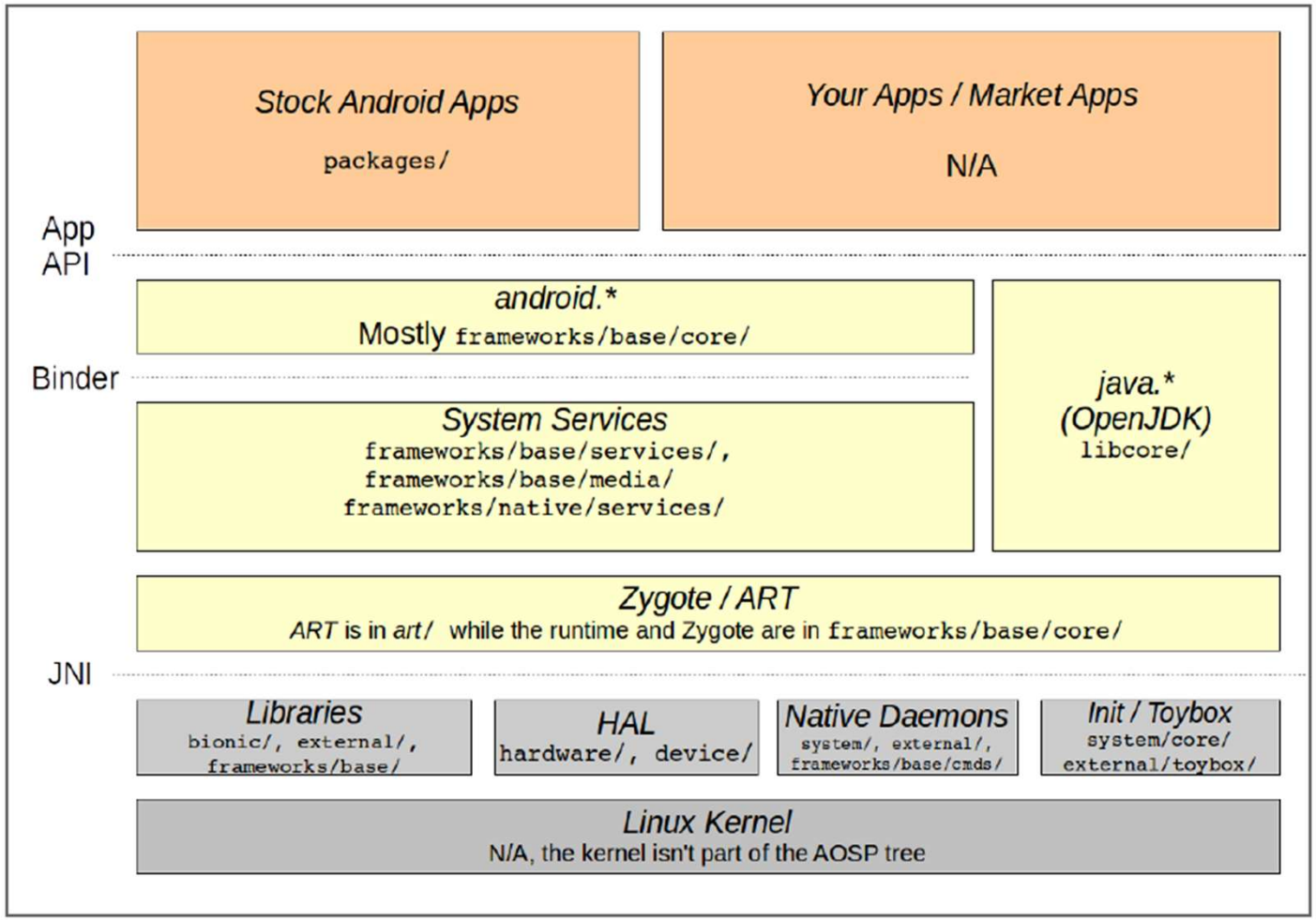
- A full-fledged Android In-Vehicle Infotainment (IVI) system.
- Runs directly on the car's head unit without relying on a smartphone.
- Android has been used in IVI for a long time, e.g. Honda (based on M/6) and Hyundai (based on JB/4.2).



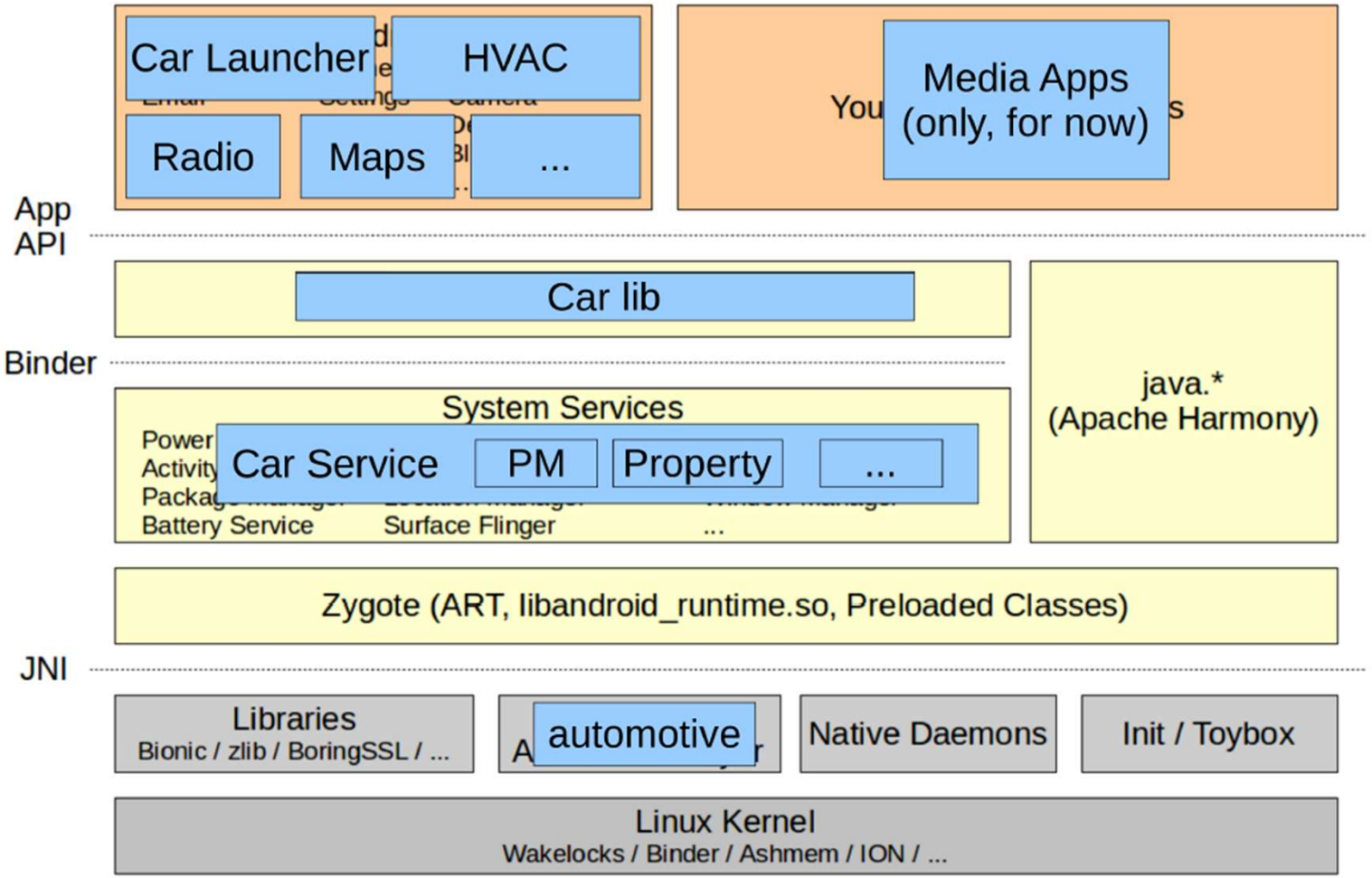
The Polestar 2 was the first vehicle to run Android Automotive OS

Automotive AOSP Architecture

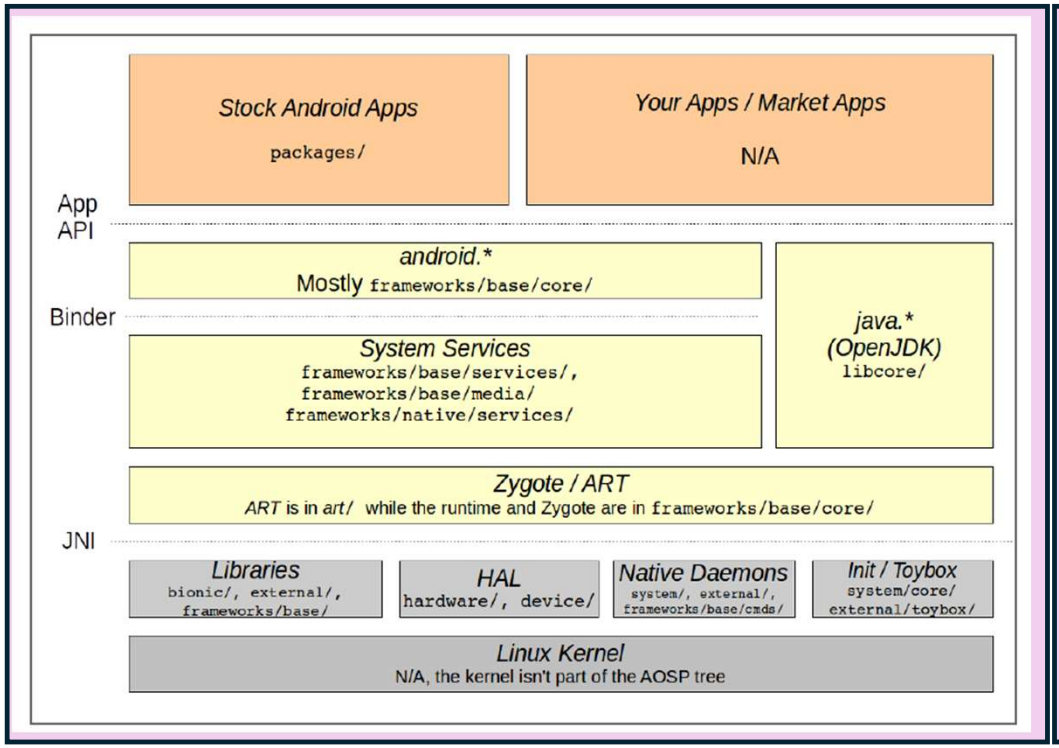
Android AOSP Architecture

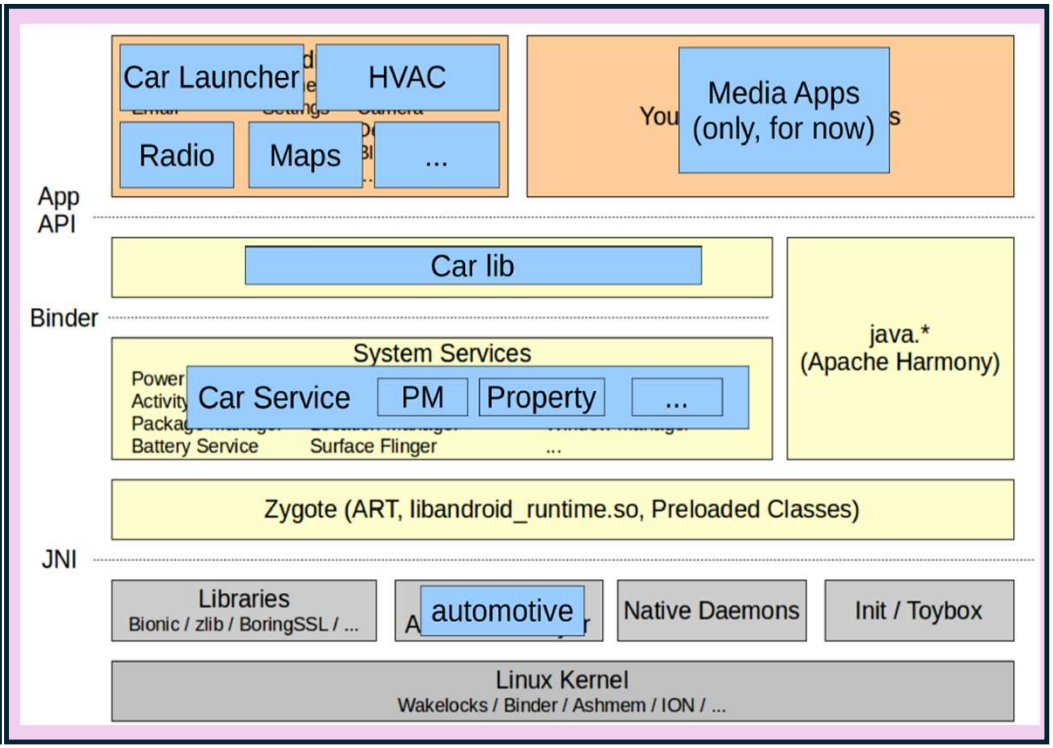


Automotive AOSP (AAOSP) Architecture

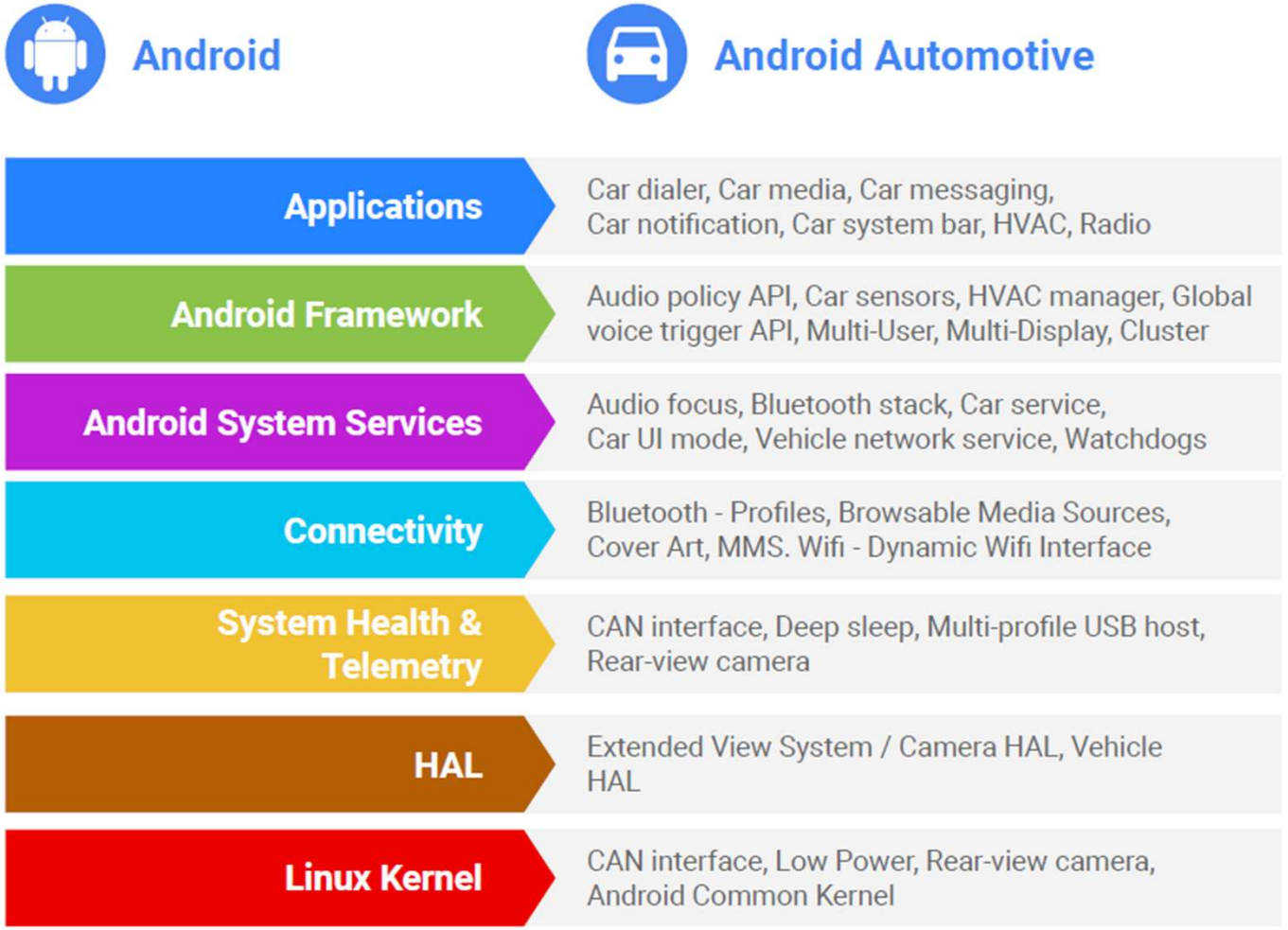


AOSP vs AAOSP Architecture



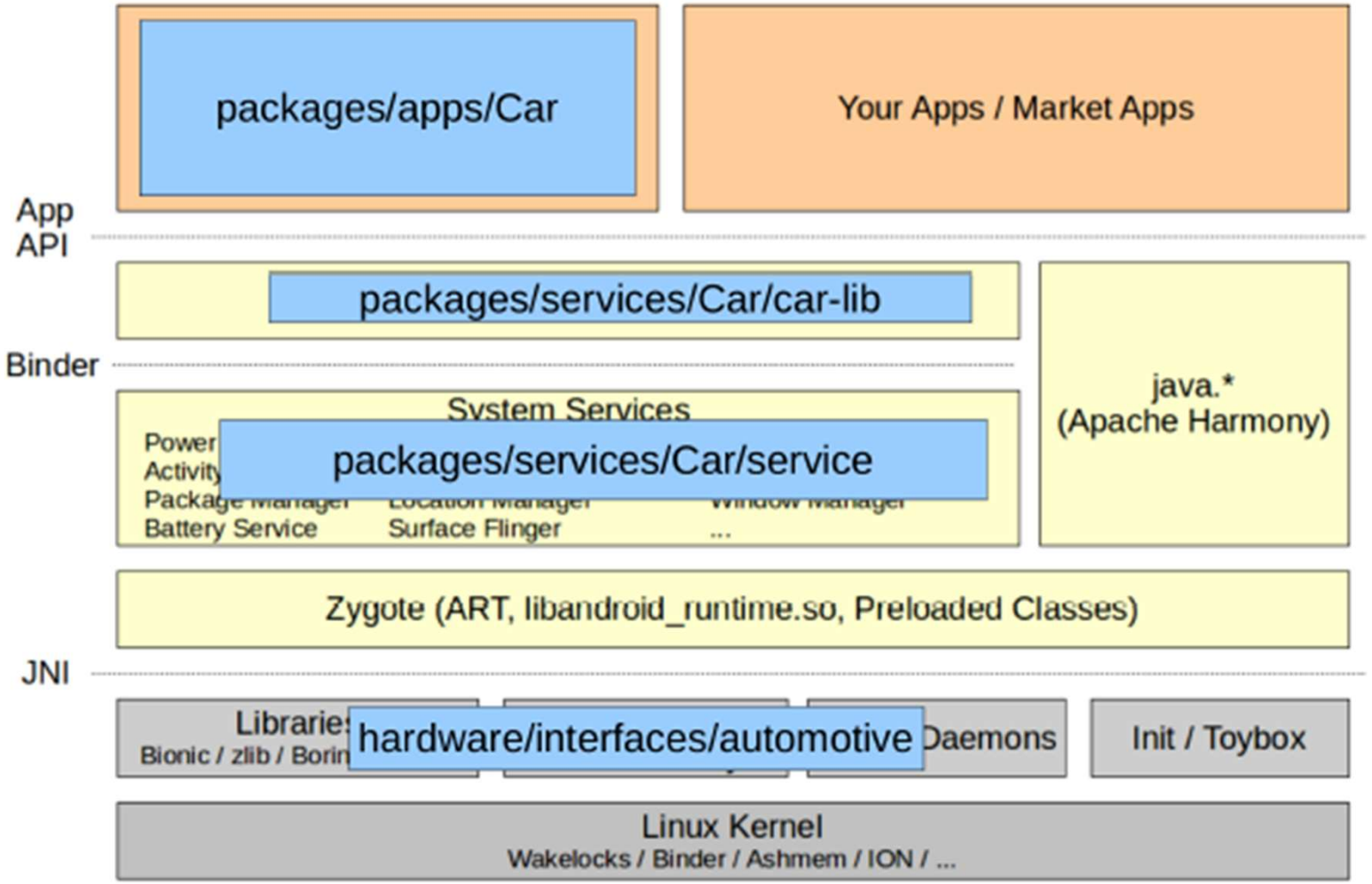


AOSP vs AAOSP

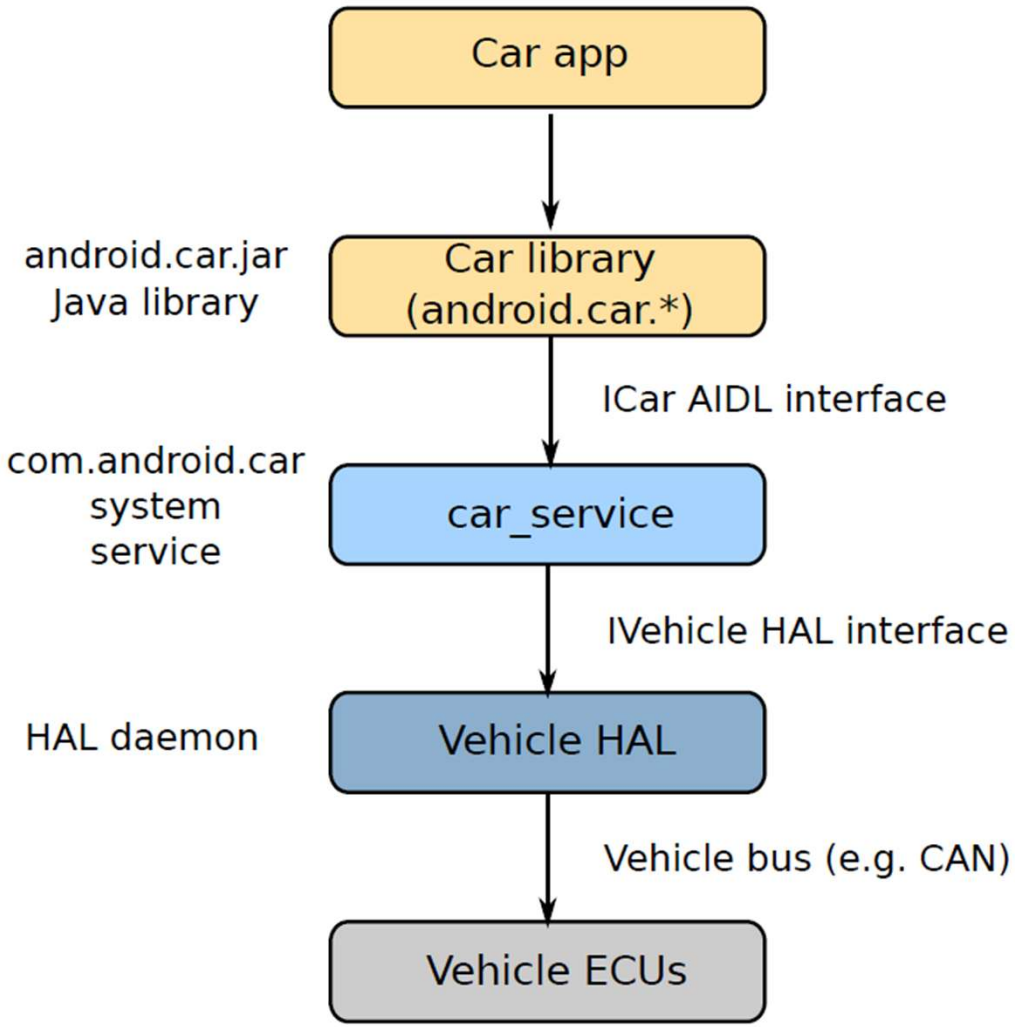


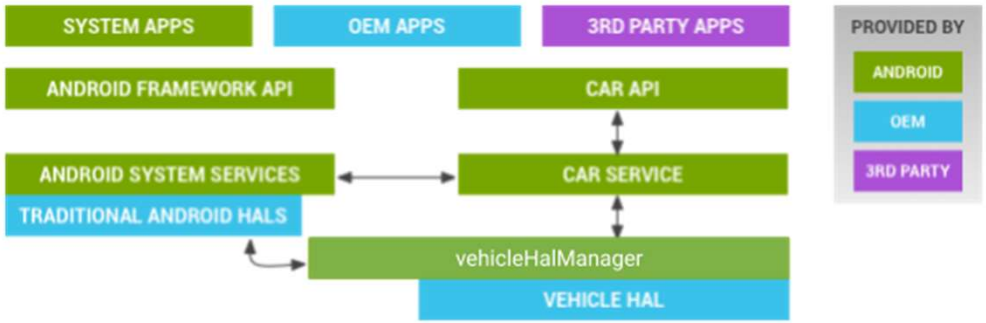
The Automotive AOSP (AAOSP) optimized and extended **Android Operating System (AOSP)** for automotive infotainment systems

AAOSP Directory Structure



AAOSP Architecture





Car apps

- AOSP contains a basic set of demo car apps
- These are all system apps, intended to be shipped with the head unit
- OEMs are expected to write their own versions using these as a guide

Car Manager

- Interface library for Car Service
- Contains the android.car packages required to write car apps, including android.car.Car

Car Services

- System service acting as a container for many (20 - 30) app level services
- Service named car_service
- Mostly a wrapper for VHAL properties

Example Car Services

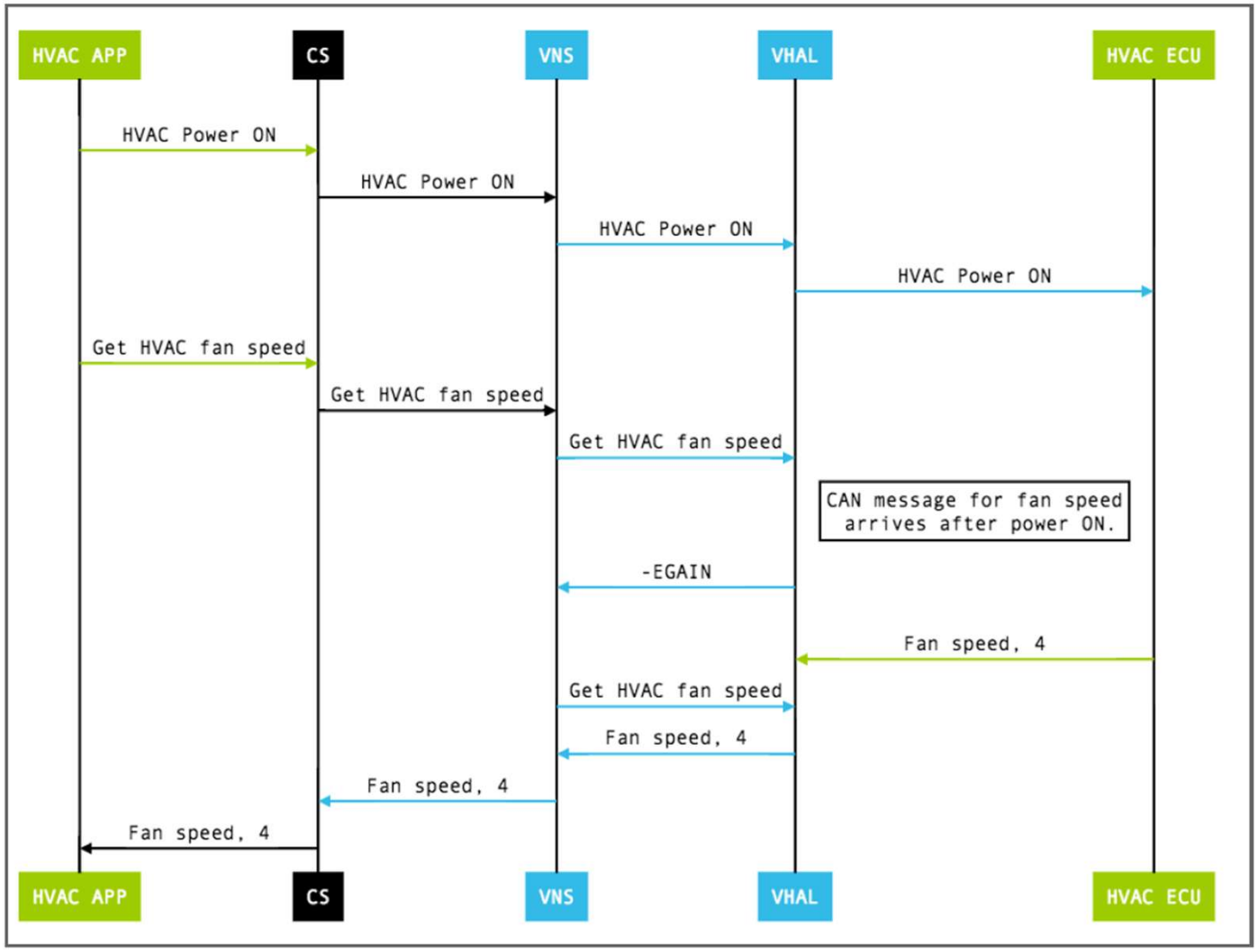
- CarUserService
- SystemActivityMonitoringService
- CarPowerManagementService
- CarPropertyService
- CarDrivingStateService
- CarUXRestrictionsService
- CarPackageManagerService
- CarInputService
- GarageModeService
- CarUserNoticeService
- AppFocusService
- CarAudioService
- CarNightService
- InstrumentClusterService
- CarLocationService
- CarBugreportManagerService

Vehical HAL

- The Vehicle HAL (VHAL) mediates between Android and the vehicle
- It Allows apps and the Android framework to
  - monitor variables, e.g. speed
  - control variables, e.g. side window position
- Vehicle variables are represented as vehicle properties
- Properties have names such as PERF_VEHICLE_SPEED and WINDOW_POS

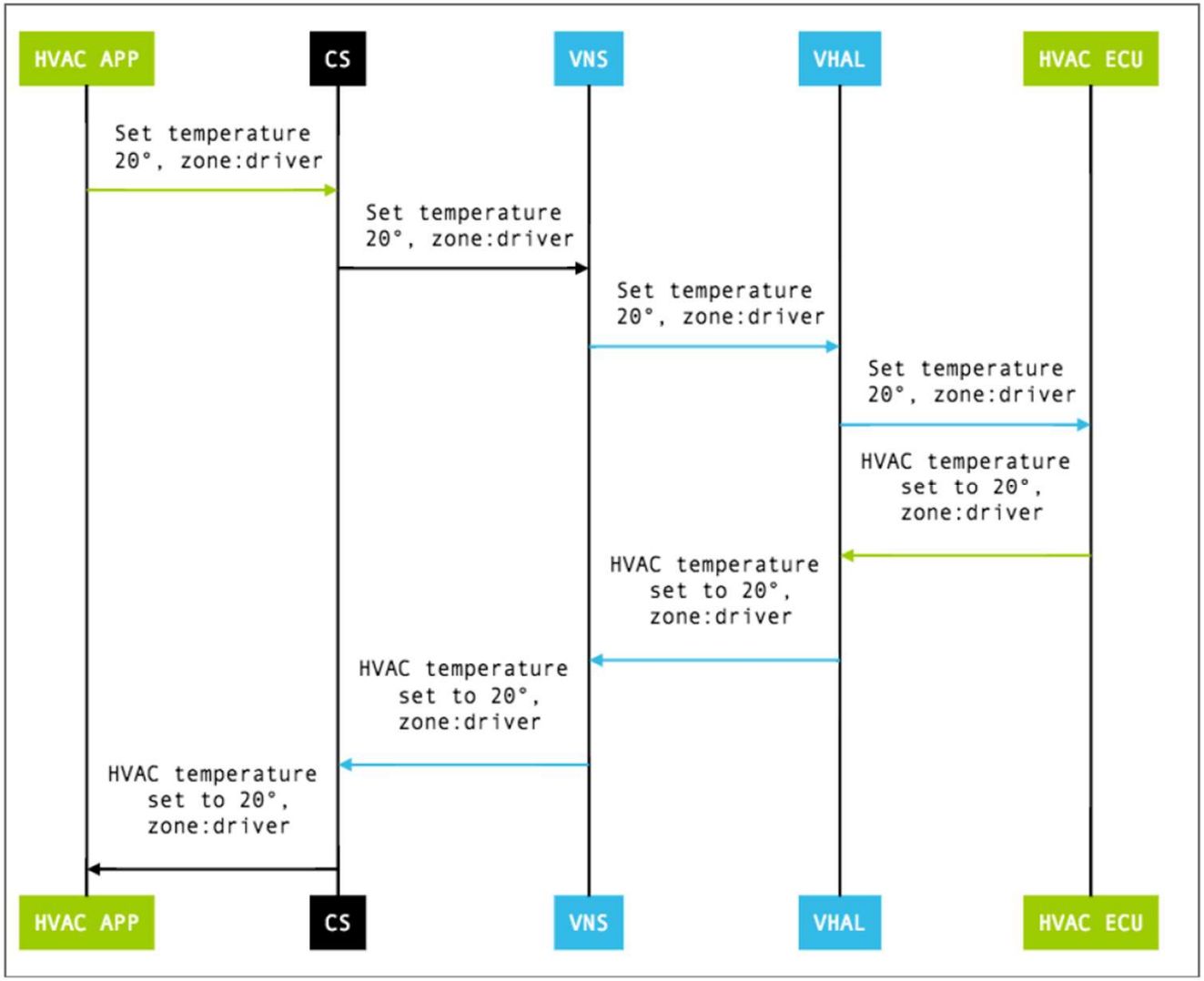
Get calls

Sequence Diagram how an HVAC (Heating, Ventilation, and Air Conditioning) fan speed "get" request flows through the automotive Android (AOSP) stack.



Set calls

Sequence diagram showing the "set" operation for HVAC temperature



Automotive AOSP A14 Build Steps

The best operating systems for building AOSP T/13 or U/14 are Ubuntu 20.04

Step 1 :
You need to install some extra packages for the AOSP build, as described here…
https://source.android.com/source/initializing.html

Step 2:

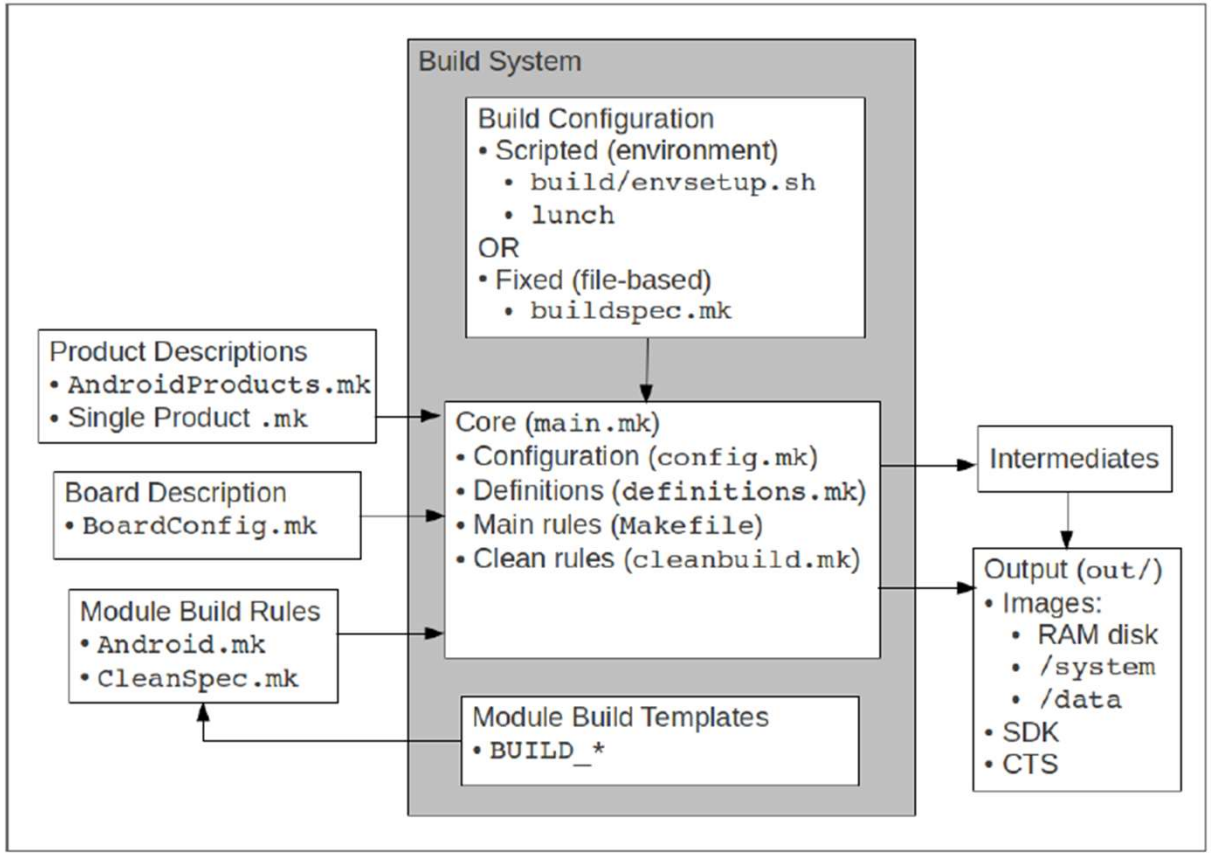
```
$ sudo apt install flex bison gcc-multilib g++-multilib git gperf \
libxml2-utils make zlib1g-dev zip curl \
patch vim lzop tree python python-mako ccache libncurses5-dev
libssl-dev \
graphviz qemu-kvm eog
```

Step 3:
Set python3 as default. For Ubuntu 20.04 –
$ sudo apt install python-is-python3

Step 4:
Install repo
$ curl https://storage.googleapis.com/git-repo-downloads/repo > $HOME/bin/repo
$ chmod a+x $HOME/bin/repo
$ export PATH=$(pwd):${PATH}

Step 5:
Install AOSP
$ mkdir aosp
$ cd aosp
$ repo init -u https://android.googlesource.com/platform/manifest -b android-14.0.0_r21
$ repo sync -c

Android Build System Architecture diagram



Lunch

- The lunch command selects the device to build

- Sets up the shell environment, including:
  - $ANDROID_BUILD_TOP -> Top of the AOSP source tree
  - $ANDROID_PRODUCT_OUT -> Staging area for product files
  - $OUT -> Same as $ANDROID_PRODUCT_OUT
  - $PATH -> Includes build tools in
    - aosp/out/host/linux-x86/bin
    - and other places

Step 5:

$ cd $HOME/aosp

$ source build/envsetup.sh

$ lunch **lunch aosp_cf_x86_64_phone-eng**

$ make –j$(nproc) Or make –j32

NOTE - The lunch menu is assembled by reading **device/*/*/AndroidProducts.mk**

NOTE -
build type is one of =>

user -> limited access; suited for production
userdebug -> like "user" but with root access, suitable for debugging
eng -> development configuration with additional debugging tools

After the build

- All intermediate and binary files are put in directory **out/**

- Host executables are put in **out/host/linux-x86/bin/**
  - Utilities used during the build
  - Debug and dev tools such as adb and fastboot

- Target files are put in **out/target/product/[product name]**
  - This path is stored in $ANDROID_PRODUCT_OUT and also in $OUT

Our Customized Device configuration

Step 6:
Get XRDA3 (AOSP) and XRDA3CAR (AAOSP) device configuration and Lunch option –

```
$ cd $HOME/aosp
$ git clone https://github.com/BuildandShip-stack/xrda3-local-manifest/tree/android14
.repo/local_manifests \
-b android14
$ repo sync –c
```

Step 7:

```
$ cd $HOME/aosp
$ source build/envsetup.sh
$ lunch
```

Select **xrda3car-userdebug** from the lunch menu

```
$ make -j32
```

Creating a new device

- Device configurations can be in **device/** or **vendor/** directories
- There is a two-level hierarchy
  - organization/product
  - The examples that follow use **xrda3**
- The two key files are AndroidProducts.mk and BoardConfig.mk
- The next few slides describe these essential files...

AndroidProducts.mk

- This is the top-level makefile for a product
- **PRODUCT_MAKEFILES** lists one or more makefiles
- (Since Q/10) **COMMON_LUNCH_CHOICES** contains items to add to lunch menu
- A simple example:
  - PRODUCT_MAKEFILES := $(LOCAL_DIR)/xrda3.mk
  - COMMON_LUNCH_CHOICES := \
  - xrda3-eng
- Search path for AndroidProducts.mk is
  - device/*
  - vendor/*
- $LOCAL_DIR is the path to the directory containing the AndroidProducts.mk file

Product makefiles (e.g. xrda3.mk)

- One per product
- Variable names mostly begin with PRODUCT_
- Must
  - inherit detailed product definition, normally a generic product followed by local
  - product
  - set PRODUCT_NAME to base name of the product makefile
  - set PRODUCT_DEVICE to the directory that contains the BoardConfig.mk file
- Simple example -> File name xrda3.mk

```
$(call inherit-product, $(LOCAL_PATH)/device.mk)

PRODUCT_NAME := xrda3
PRODUCT_DEVICE := xrda3
PRODUCT_BRAND := XRDA3_Cybernetics
PRODUCT_MODEL := XRDA3 Phone
```

- Meaning of variables

Tag	System property	Usage
PRODUCT_NAME	ro.product.name	Copied to $TARGET_PRODUCT
PRODUCT_DEVICE	ro.product.device	Location of BoardConfig.mk; directory in $OUT
PRODUCT_MODEL	ro.product.model T	ext displayed in About - Model number
PRODUCT_BRAND	ro.product.brand	Build fingerprint (mustn’t contain spaces

Pre-defined products

- Phone: $AOSP/build/target/product/aosp_base_telephony.mk
- Tablet (phone without telephony): $AOSP/build/target/product/aosp_base.mk
- TV: $AOSP/device/google/atv/products/atv_base.mk
- Automotive: $AOSP/device/generic/car/common/car.mk

device.mk

- Provides overrides of properties inherited in the product make file and adds new ones
- The next few slides describe these variables:
  - PRODUCT_PACKAGES
  - PRODUCT_COPY_FILES
  - PRODUCT_PROPERTY_OVERRIDES

Warning: these three variables are space separated lists
If you assign using **:=** you overwrite anything already in the list. Append to the list by using **+=**instead

PRODUCT_PACKAGES

- A list of all the Android modules and packages to be built
- Example

```
PRODUCT_PACKAGES += \
        lmkd \
        logcat \
        helloworld-app
```

- Note: it is not a build error if a module with that name does not exist
- For example, if you mistype lmkd as lkmd the build will succeed, but (obviously) lmkd will not be
- included

PRODUCT_COPY_FILES

- A list of files to be copied to the device file system
- Format:

  PRODUCT_COPY_FILES += source_path:dest_path
- source_path is relative to shell variable $ANDROID_BUILD_TOP
- dest_path is relative to shell variable $OUT
- Example:

  PRODUCT_COPY_FILES += \
  $(LOCAL_PATH)/init.ranchu.rc:vendor/etc/init/hw/init.ranchu.rc \
  $(LOCAL_PATH)/bootanimation.zip:system/media/bootanimation.zip

Copying whole directories

- Problem: you want to copy a directory containing pre-built binaries and configuration files
- You don’t want to have many individual **PRODUCT_COPY_FILES** statements
- Solution: find-copy-subdir-files
- Usage: find-copy-subdir-files,[match],[source],[dest]
- Example: copy all files in device/rpiorg/rpi3/prebuilt/vendor to target directory /vendor:

```
# Prebuilt
PRODUCT_COPY_FILES += \
        $(call find-copy-subdir-files,*,device/rpiorg/rpi3/prebuilt/vendor,vendor)
```

PRODUCT_PROPERTY_OVERRIDES

- A list of system properties to add to **/system/build.prop**
- There are very many system properties, look at other devices for examples and read the code to find out what they do
- Example:

```
PRODUCT_PROPERTY_OVERRIDES += \
        wifi.interface=wlan0 \
        wifi.supplicant_scan_interval=15
```

BoardConfig.mk

- Defines characteristics of the board

- Variable names mostly begin with BOARD_
  - CPU architecture and type
  - Sizes of file system partitions

- Search path for BoardConfig.mk is
  - device/*
  - vendor/*
  - build/target/board/$PRODUCT_DEVICE/BoardConfig.mk

vendorsetup.sh (before Q/10)

- In Q/10, the lunch menu is populated by COMMON_LUNCH_CHOICES in AndroidProducts.mk
- Previously, it was populated by vendorsetup.sh
- Format add_lunch_combo PRODUCT_NAME-[user | userdebug | eng]
- Example
  add_lunch_combo marvin-eng

Devices in Android

AOSP Devices: Goldfish

- Goldfish is the well-known Android Emulator
- Primarily for testing applications in Android Studio

Product	**device/generic/goldfish/AndroidProducts.mk**:
ARM64 phone emulator	sdk_phone64_arm64
x86_64 phone emulator	sdk_phone64_x86_64
x86_64 tablet emulator	sdk_tablet_x86_64
	device/generic/car/AndroidProducts.mk:
ARM64 car emulator	sdk_car_arm64
x86_64 car emulator	sdk_car_x86_64
x86_64 car emulator	sdk_car_portrait_x86_64

AOSP Devices: Cuttlefish

- Cuttlefish is a headless emulator designed to test AOSP, used by Google as part of their CI for AOSP
- Cuttlefish devices in device/google/cuttlefish/AndroidProducts.mk

Product	**device/generic/goldfish/AndroidProducts.mk**:
ARM64 car	aosp_cf_arm64_auto
ARM64 phone	aosp_cf_arm64_phone
x86_64 foldable	aosp_cf_x86_64_foldable
x86_64 phone	aosp_cf_x86_64_phone
x86_64 TV	aosp_cf_x86_64_tv
x86_64 car	aosp_cf_x86_64_auto
x86 phone (32-bit)	aosp_cf_x86_phone
x86 TV (32-bit)	aosp_cf_x86_tv

AOSP Devices: trout

- Trout uses the same virtio hardware platform as Cuttlefish, but configured for Android Automotive
- Intended for Automotive platforms that run within a hypervisor
- device/google/trout/AndroidProducts.mk

aosp_trout_arm64
aosp_trout_x86

AOSP Devices: Pixel

- Device configuration for Pixel handsets in U/14

Code name	Product	Path in `device/`
Flame	Pixel 4	`google/coral`
Coral	Pixel 4 XL	`google/coral`
Sunfish	Pixel 4a	`google/sunfish`
Bramble	Pixel 4a/5G	`google/bramble`
Redfin	Pixel 5	`google/redfin`
Barbet	Pixel 5a/5G	`google/barbet`
Oriole	Pixel 6	`google/oriole`
Raven	Pixel 6 Pro	`google/raven`
Bluejay	Pixel 6a	`google/bluejay`
Panther	Pixel 7	`google/pantah`
Cheetah	Pixel 7 Pro	`google/pantah`
Lynx	Pixel 7a	`google/lynx`
Tangorpro	Pixel Tablet	`google/tangorpro`
Felix	Pixel Fold	`google/felix`
Shiba	Pixel 8	`google/shusky`
Husky	Pixel 8 Pro	`google/shusky`

AOSP Devices: Dev Boards

- Device configuration for Development boards in U/14

Board	Lunch target	Path
Beagle X15	beagle_x15	`device/ti/beagle_x15`
Dragonboard 845	db845c	`device/linaro/dragonboard`
HiKey 620/960	hikey/hikey960	`device/linaro/hikey`
Poplar (*)	poplar	`device/linaro/poplar`
SEI 510/610 (**)	yukawa	`device/amlogic/yukawa`
VIM3/VIM3L	yukawa	`device/yukawa`

Android Emulators

Goldfish

- well-known Android emulator, used for testing apps
- usually installed along with Android Studio ...
- ... but also part of AOSP

Cuttlefish (since P/9)

- used to test the Android operating system
- more accurate emulation of real hardware

Goldfish and Ranchu

- Before A-L/5, emulated hardware was named **Goldfish** based on a forked copy
- of QEMU version 1 – deprecated
- Since L/5, the target is named **Ranchu**, based on a forked copy of QEMU version 2
- Note: this change has not been propagated consistently: "goldfish" still appears in many of the build files, even though the real target is "ranchu"

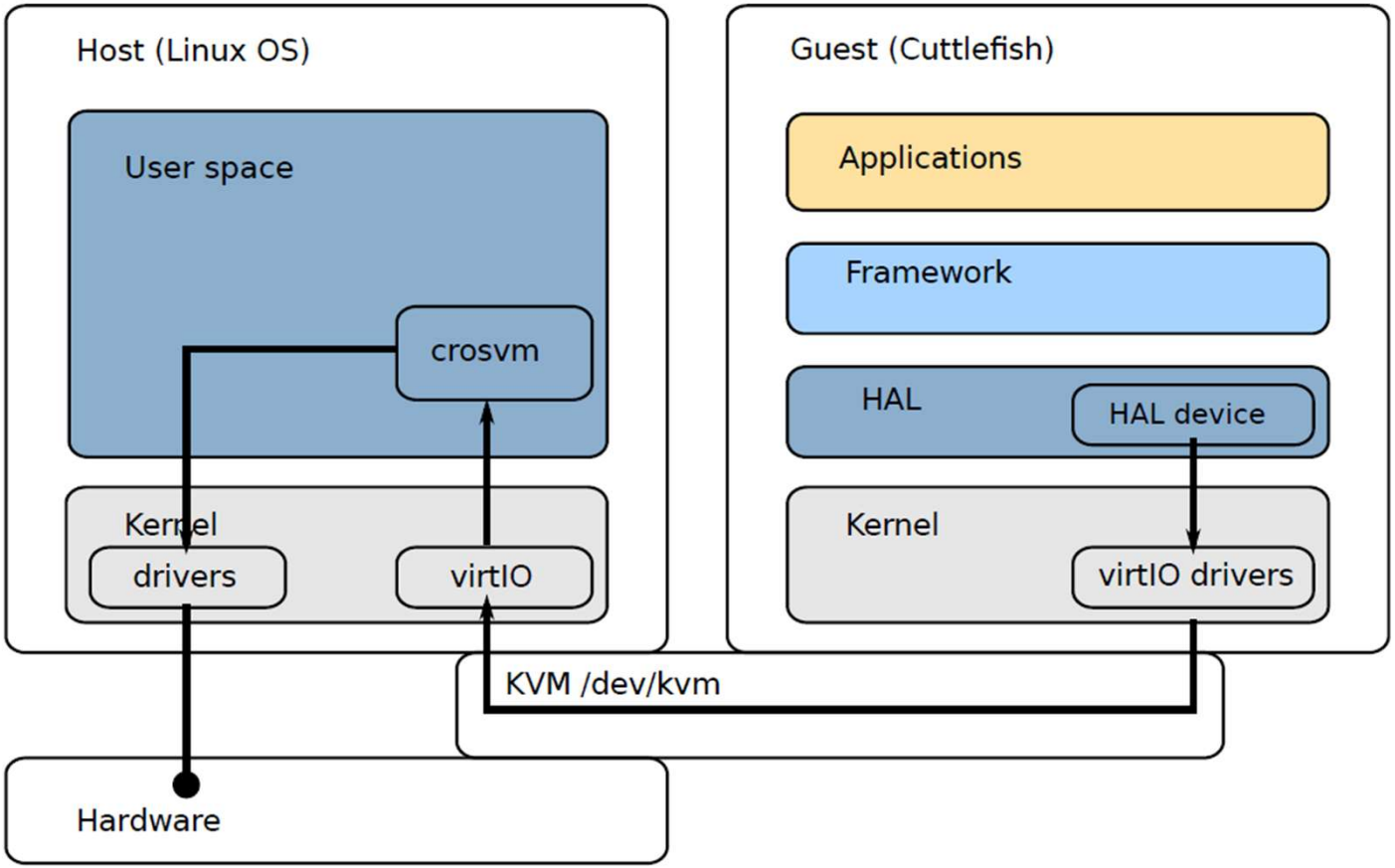
Goldfish

- Goldfish is the classic Android emulator that comes with Android Studio.
- It is *optimized for app developers*—ideal for testing and developing Android apps.
- It uses a QEMU-based emulator and supports typical AVD (Android Virtual Device) images for various Android versions.
- Goldfish is easy to set up and runs on multiple host operating systems (Windows, macOS, Linux).
- Limitations: Goldfish does not closely simulate all hardware behaviors. It is less suited for platform/deep OS development since it lacks high fidelity with real hardware and OS initialization (e.g., bootloader, HAL).

Cuttlefish

- Cuttlefish is a newer, *high-fidelity* virtual device platform designed for AOSP and platform developers.
- It closely replicates real hardware and is used for testing/customizing the Android OS itself (AOSP), not just apps.
- Runs as a virtual machine (using CrosVM or QEMU) and supports scalable, “cloud-friendly” use (can launch many instances for CI or large-scale testing).
- Offers full compatibility with AOSP mainline development and is used by Google internally.
- Designed to run only on Linux (usually Ubuntu), and the setup is more complex and resource-intensive.
- Limitations: Not meant for typical app development workflows; intended mostly for system/framework development and validation.

Cuttlefish



- Headless: you communicate with it via ADB or remote desktop
- Most I/O is via Linux virtio drivers (block, net, serial, gpu)
- Uses crosvm (Chrome OS Virtual Machine Monitor) (*)
- **Uses Linux KVM (Kernel-based Virtual Machine) for hardware acceleration**

- (*) Earlier versions were based on QEMU

Cuttlefish

- Cuttlefish runs as a guest OS on Linux
- Host and guest OS must be the same architecture, usually x86
- These are the Cuttlefish targets for x86 and x86_64
  - aosp_cf_x86_64_foldable-userdebug
  - aosp_cf_x86_64_phone-userdebug
  - aosp_cf_x86_64_tv-userdebug
  - aosp_cf_x86_64_auto-userdebug
  - aosp_cf_x86_phone-userdebug
  - aosp_cf_x86_tv-userdebug
- Note in previous slides the **marvin** device inherits from aosp_cf_x86_64_phone-userdebug
- And **marvincar** device inherits from aosp_cf_x86_64_auto-userdebug

Steps to Install Cuttlefish Emulator

Step 1 :
Check if KVM is installed by checking for the existence of /dev/kvm, without it no emulator will work because Cuttlefish runs in a hypervisor domain

```
$ ls /dev/kvm
```

Step 2:
Download the cuttlefish emulator

```
$ cd ~/android/
$ sudo apt install -y git devscripts config-package-dev debhelper-compat golang curl
$ git clone https://github.com/google/android-cuttlefish
$ cd android-cuttlefish
$ git checkout 17ccfffd06139cd69d3b4cd4f0179b43b20aa573
$ for dir in base frontend; do cd $dir; debuild -i -us -uc -b -d; cd .. done
$ sudo dpkg -i ./cuttlefish-base_*_*64.deb || sudo apt-get install -f
$ sudo dpkg -i ./cuttlefish-user_*_*64.deb || sudo apt-get install -f
$ sudo usermod -aG kvm,cvdnetwork,render $USER
$ sudo reboot    # NOTE => This last instruction will reboot your computer
```

Step 3 :

Launch cuttlefish with the WebRTC interface:

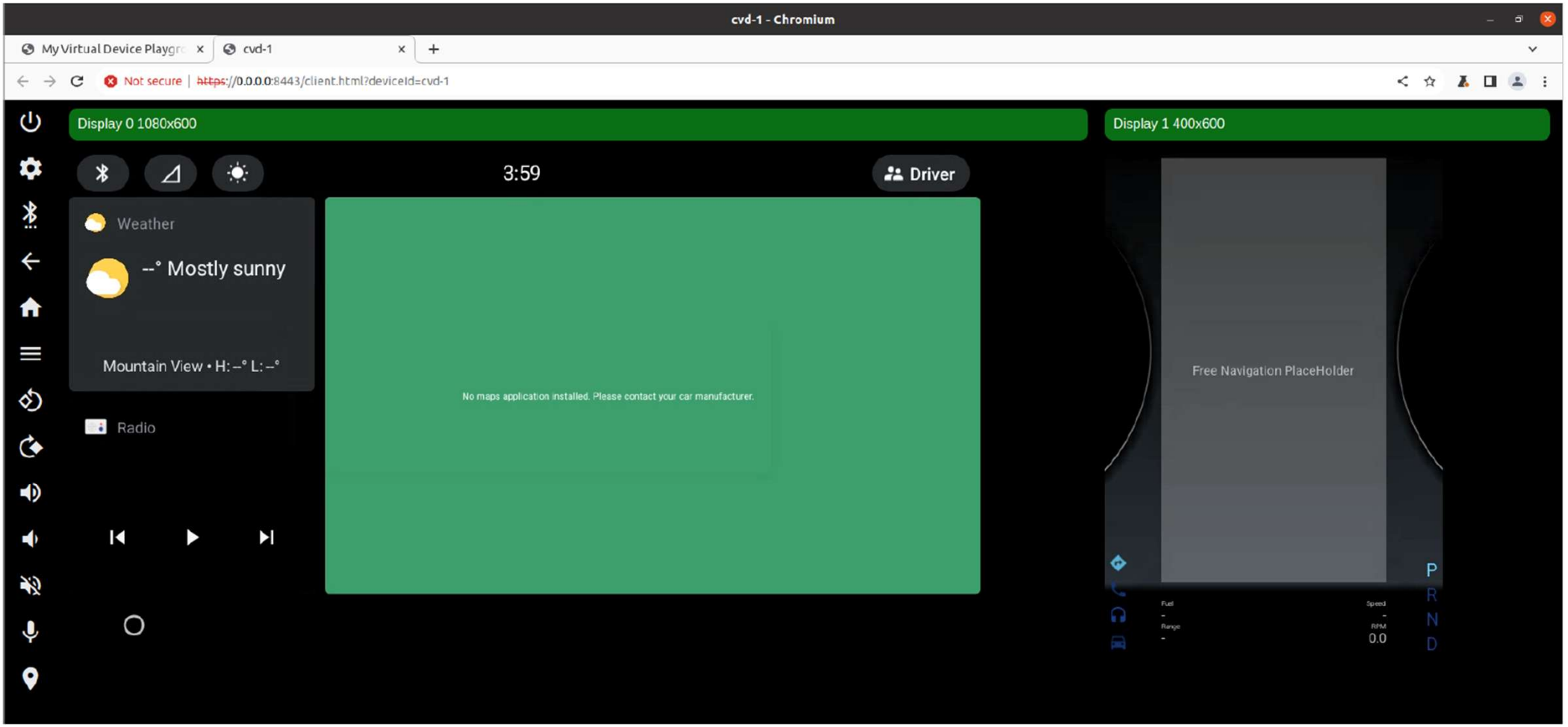
```
$ cd $HOME/aosp
$ launch_cvd -start_webrtc
```

Output ->

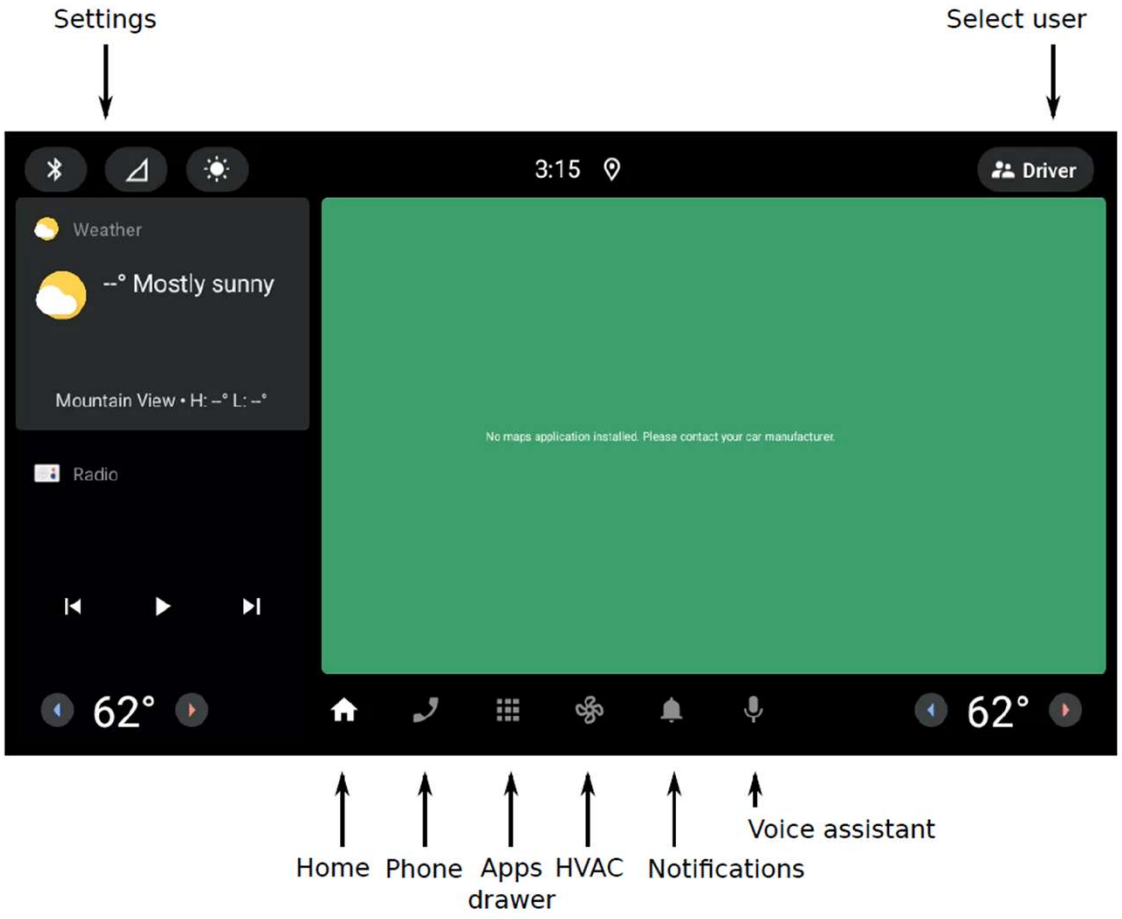
```
init: starting service 'adbd'...
VIRTUAL_DEVICE_DISPLAY_POWER_MODE_CHANGED
VIRTUAL_DEVICE_BOOT_STARTED
VIRTUAL_DEVICE_BOOT_COMPLETED
Virtual device booted successfully
```

NOTE ->

- **Cuttlefish GUI runs on port 8443**
- Launch a browser (must be Chrome or Chromium).
- Enter URL https://0.0.0.0:8443


cvd-1 - Chromium
My Virtual Device Playgr
cvd-1
Not secure | https://0.0.0.0:8443/client.html?deviceId=cvd-1
Display 0 1080x600
3:59
Driver
Weather
--° Mostly sunny
Mountain View • H: --° L: --°
Radio
No maps application installed. Please contact your car manufacturer.
Display 1 400x600
Free Navigation PlaceHolder
Fuel
-
Range
-
Speed
-
RPM
0.0
P
R
N
D

This is the default Automotive home screen for U/14



CVD STATUS

In another terminal, run lunch command again and then check if CVD is running:

$ cd $HOME/aosp

$ source and lunch

$ cvd_status

ADB

Check that there is an ADB device available.

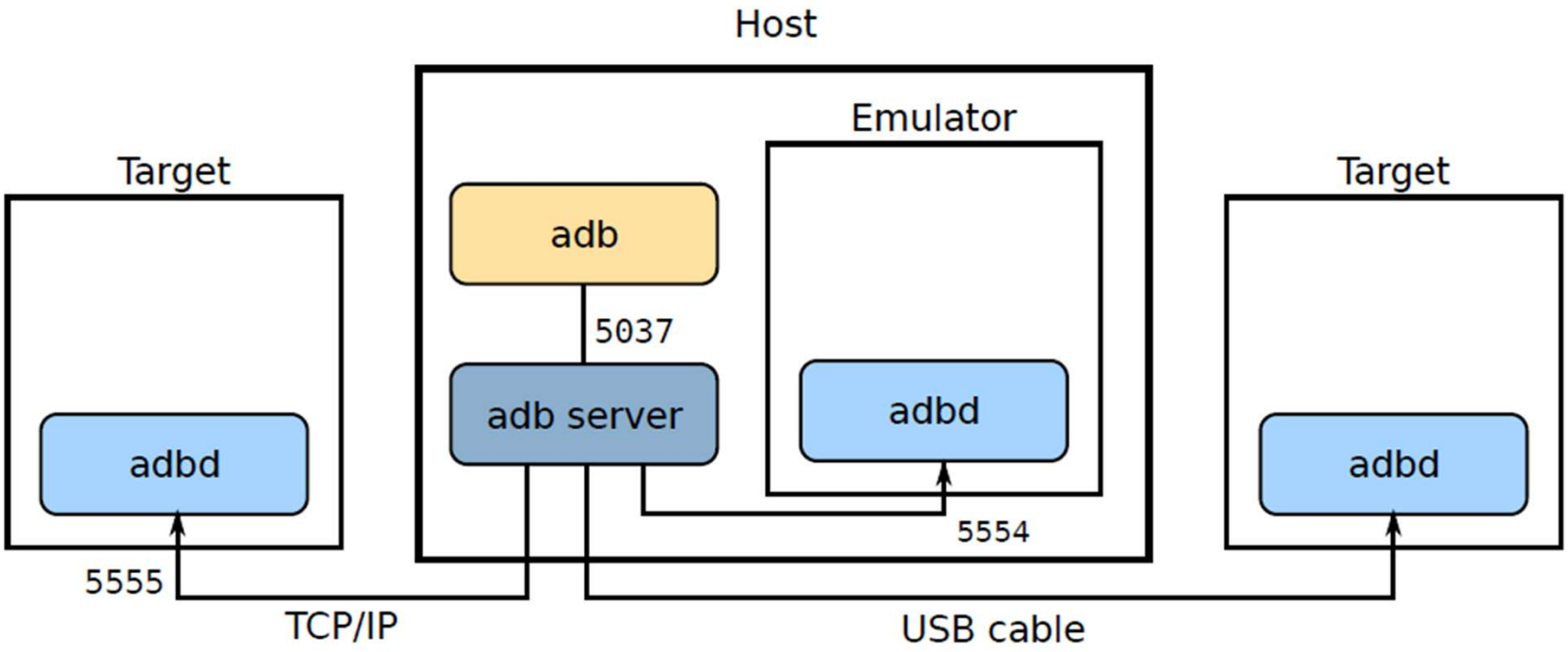
$ adb devices

$ adb shell

marvincar:/ $

ADB and Logcat

ADB Android Debug Bridge



ADB Android Debug Bridge Connection over USB

- adb devices lists devices connected to the adb server
- Each has a unique identifier or serial number
- Pass this identifier when sending commands:
  - $ adb devices
    List of devices attached
    emulator-5554 device
    199e5e2d device
  - $ adb -s emulator-5554 shell

NOTE

- Don’t need -s device-name if
  - only one device
  - only one device connected via USB: use -d
  - only one emulator: use -e

ADB Android Debug Bridge (common issue)

$ adb devices
List of devices attached
??????????? no permissions

Run the command lsusb to find the vendor and product ids
$ lsusb
Bus 001 Device 002: ID 18d1:4e40 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Add this rule to /etc/udev/rules.d/51-android.rules:
SUBSYSTEM=="usb", ATTR{idVendor}=="18d1", ATTR{idProduct}=="4e40", MODE="0666"
Then restart udev
$ sudo udevadm control --reload; sudo udevadm trigger

ADB Android Debug Bridge Connection over TCP

- adbd (on the device) listens on TCP port 5555
  - you can change the port with property service.adb.tcp.port

- Connect from adb client using the IP address of the target device (if you know it), or using the name Android.local like this:
  - $ adb connect Android.local:5555
  - connected to 192.168.42.2:5555
  - $ adb shell
  - #

Starting the ADB server

- The ADB server will be started when you run the first ADB client
- Example

  $ adb shell

  * daemon not running; starting now at tcp:5037

  * daemon started successfully
- You can use these commands to control the server
  - adb start-server # ensure that there is a server running
  - adb kill-server # kill the server if it is running

Useful ADB Commands

Command	Description
shell	start a command shell
push <local> <remote>	copy file/dir to device
pull <remote> [<local>]	copy file/dir from device
sync [<directory>]	sync changes in <directory> to /system on the target. Default <directory> is `$OUT/system`
sync -l [<directory>]	list files to sync, but don't copy them
logcat	View Android logs
install <file>.apk	Install an application package
uninstall <package>	Uninstall a package
reboot	Reboot
reboot recovery	Reboot into recovery mode
reboot bootloader	Reboot into bootloader
remount	Attempt to remount `/system` for read and write

Logcat

- The command **logcat** displays logs
- The default format is
  - [timestamp] [PID] [TID] [Priority] [Tag]: [Message]

- Example:
- 08-07 09:03:30.704 1565 1565 I ActivityManager: System now ready

- Log Priority (starting with the highest)
  - S Silent
  - F Fatal
  - E Error
  - W Warning
  - I Information (default)
  - D Debug
  - V Verbose

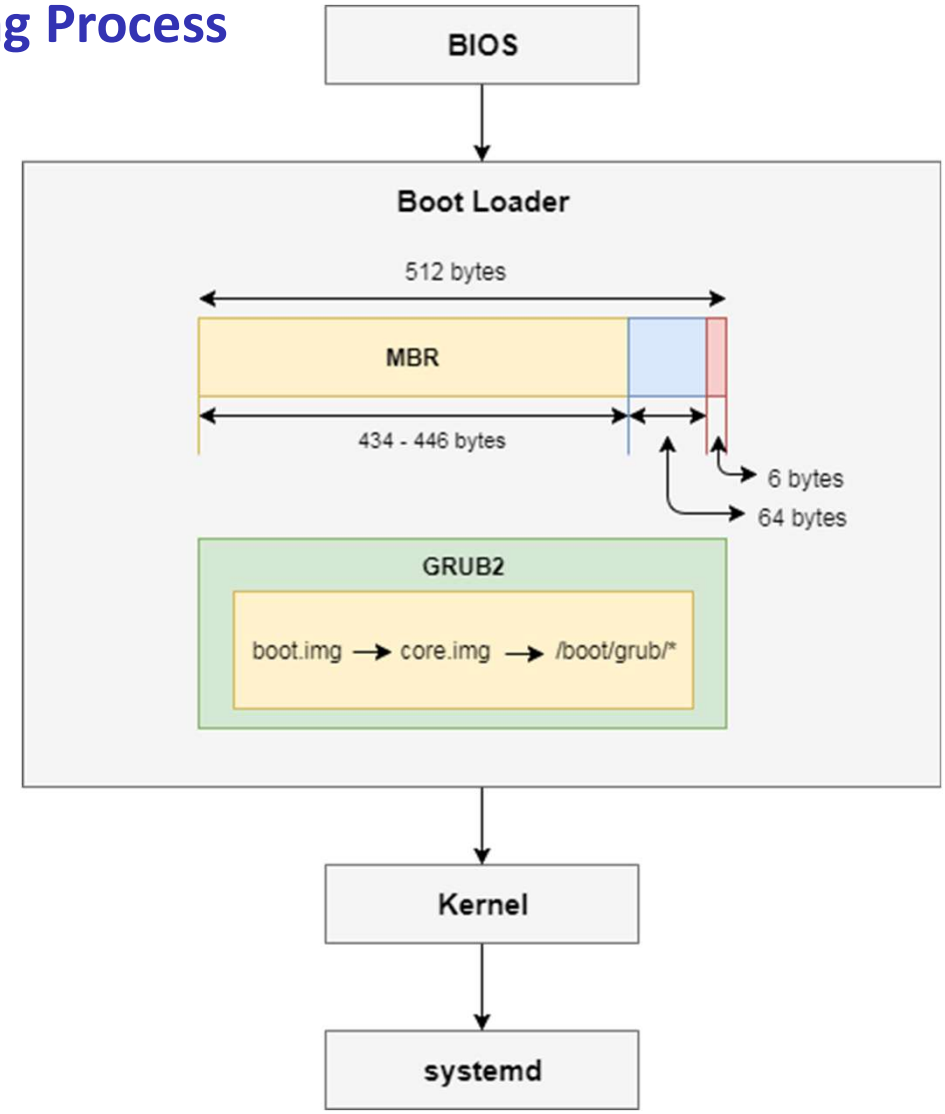
Useful Logcat Commands

logcat has many options: here are some that will be useful during this course

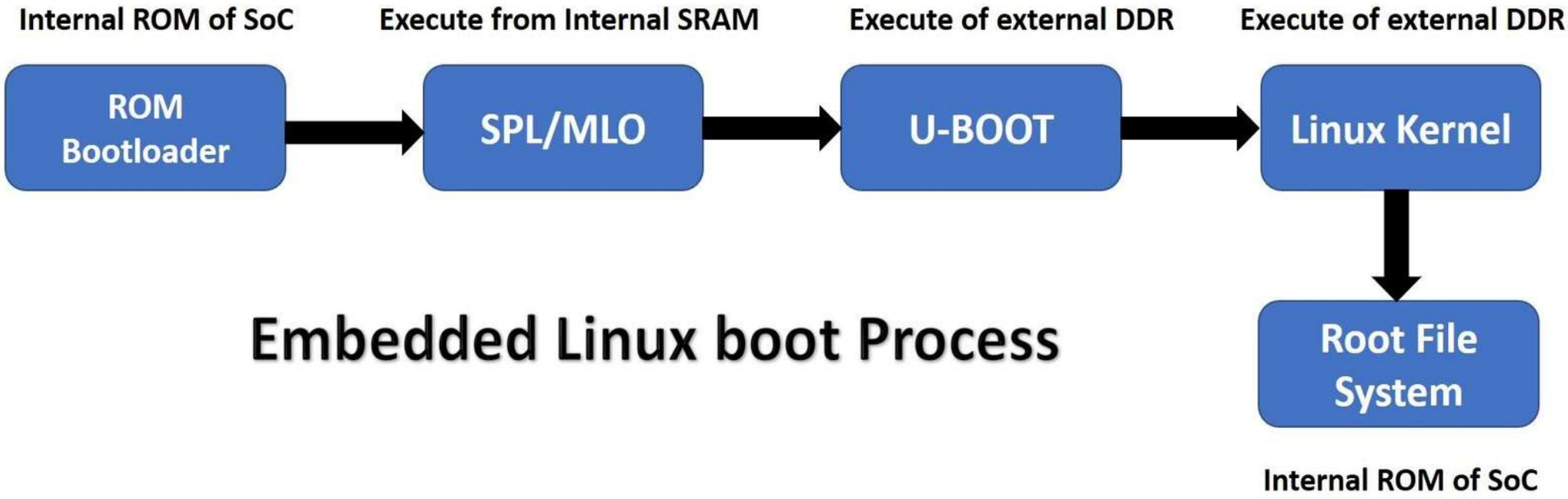
-b [buffer]	display [buffer], where buffer is 'main', 'system', 'radio', 'events', 'crash', 'default' or 'all' Without this option, displays 'main', 'system', and 'crash'
-c	clear buffers
-d	display the log and then exit (don't block)
-G <size>	Set size of buffers, selected by -b. Size may be suffixed with K or M
-g	information about buffer sizes and usage
-S	display statistics about the logs, including 'chatty' programs (next slide)
-s	set default filter to silent, equivalent to '*:S'

System Boot Flow

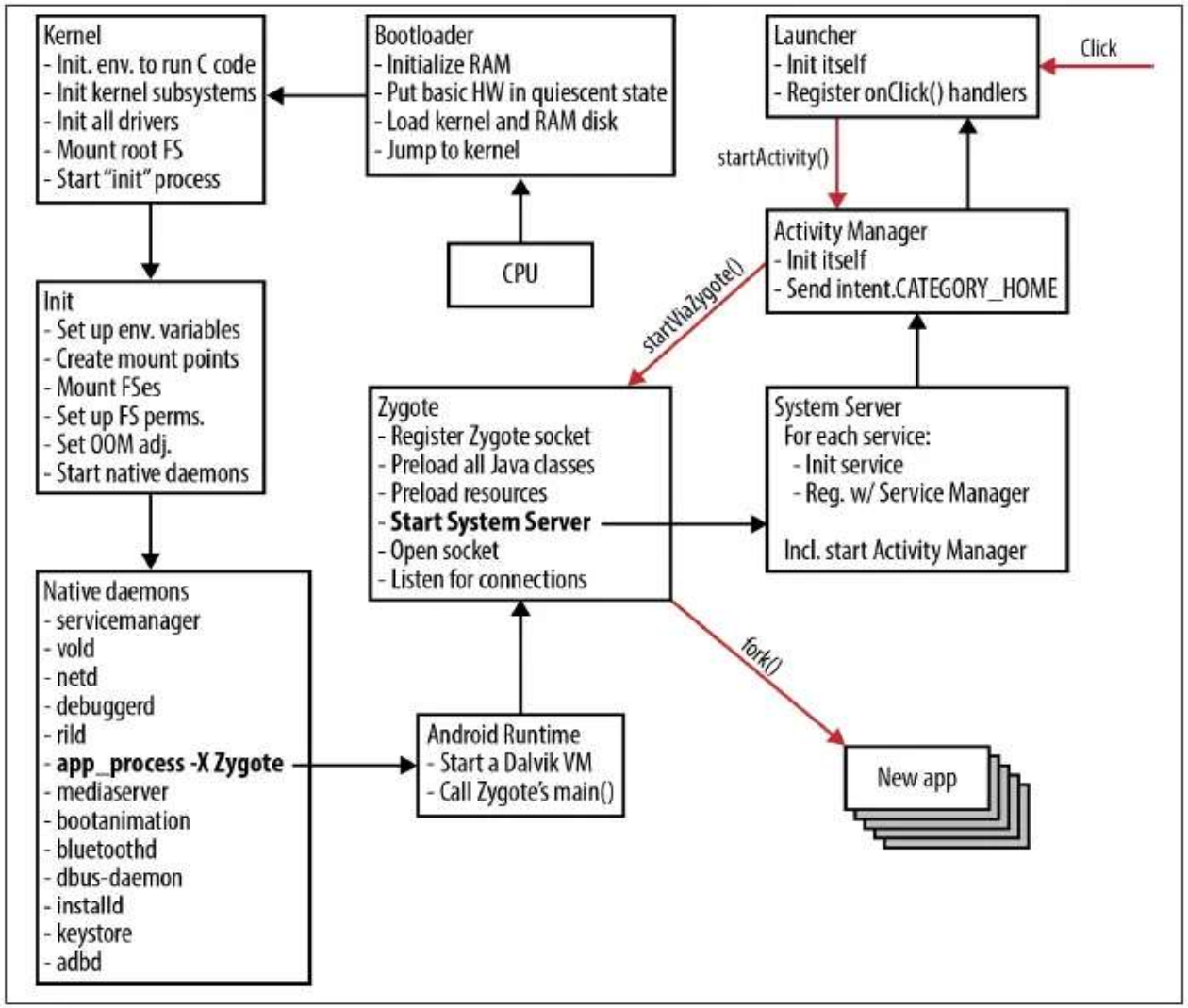
Linux PC / Ubuntu Booting Process



Embedded Linux Booting Process



Android Booting Process



Difference Between
Classic Android (Android 4.x → 10)
vs
Modern Android (Android 12 → 15)

Category	Classic Android (Android 4.x → 10)	Modern Android (Android 12 → 15)
Architecture	Monolithic & device-specific Android stack tightly coupled with hardware.	Modular, Treble-compliant architecture separating system and vendor layers for independent updates.
Partition Layout	Only a few fixed partitions: `/boot`, `/system`, `/vendor`, `/userdata`.	**Dynamic Partitions**: `/system`, `/vendor`, `/product`, `/system_ext`, `/vendor_dlkm`, `/odm_dlkm`, enabling flexible OTA updates.
Boot Image Format	`boot.img` contained kernel + ramdisk + DTB.	`boot.img` + **`vendor_boot.img`** introduced (contains vendor ramdisk + DTBO).
Boot Verification	Basic Verified Boot (AVB 1.0) or dm-verity; OEM-defined verification flow.	**AVB 2.0** with rollback protection and complete chain-of-trust from Boot ROM → vbmeta → partitions.
Kernel Model	Per-device kernel tree (OEM customized heavily).	**Generic Kernel Image (GKI 2.0)** — common base kernel maintained by Google + vendor-specific modules (`vendor_dlkm`).
Kernel Modules	Mostly built-in or statically compiled.	Split into generic (`system_dlkm`) and vendor (`vendor_dlkm`) loadable modules.
Hardware Abstraction Layer (HAL)	Based on **HIDL** (C++/binderized XML interfaces).	Migrated to **AIDL-based HALs** for stability and backward compatibility (from A12 onward).
Init System	Single `/init.rc` script with device-specific rc extensions.	Modular init configuration — partition-specific init rc files: `/system/etc/init/`, `/vendor/etc/init/`, `/product/etc/init/`.
Zygote	Single 32-bit Zygote; spawns SystemServer and app processes.	Dual Zygote model (Zygote64 and Zygote32) to handle mixed 32/64-bit apps. Runs under strict SELinux domain.
Runtime (VM)	**Dalvik VM** (JIT only).	**ART (Android Runtime)**: AOT + JIT hybrid, per-app compilation, better performance and power efficiency.
System Services	Started by SystemServer, but all defined under `/system` only.	Services distributed across partitions (`system`, `system_ext`, `vendor`, `product`) using stable AIDL/Binder interfaces.
Security & SELinux	SELinux optional or permissive.	**SELinux enforced** by default (Android 12+) with strict sandboxing and namespace separation.
Binder IPC	Single global binder space.	**Isolated Binder contexts** (e.g., vendor binder vs system binder) to separate vendor processes from framework.
Storage Access	Legacy filesystem permissions; global storage access.	**Scoped Storage** model — apps restricted to sandboxed directories.

Category	Classic Android (Android 4.x → 10)	Modern Android (Android 12 → 15)
Graphics Stack	SurfaceFlinger monolithic, vendor-dependent EGL/GL drivers.	Modular **HWC3 + Gralloc4** interfaces, vendor composition through AIDL HALs.
Audio Stack	AudioFlinger & Audio HAL (legacy HIDL).	**AIDL Audio HAL + Volume Group HAL + policy modularization** introduced.
Connectivity Stack	Native daemons like `netd`, `wpa_supplicant`, etc.	Modularized network stack: `NetworkStack.apk` as a privileged mainline module.
Update Mechanism	OTA update replaces entire system image (downtime, risk).	**Seamless A/B updates**, **Dynamic Partitions**, and **snapshot merges** for near-zero downtime updates.
Vendor Dependence	OEM had to rebuild full image for every Android upgrade.	Vendor can reuse binary blobs; only framework image needs updating (thanks to Treble).
Treble Support	Not available (pre-A8).	Fully Treble-compliant; HALs must be versioned and stable.
Mainline Modules (Project Mainline)	Not supported.	**Project Mainline** → Core components (Media, NNAPI, Conscrypt, etc.) updatable via Google Play (APEX modules).
Virtualization Support	None (baremetal).	Introduced **pKVM / Cuttlefish / Crosvm** — secure virtualization for Android on Arm64 and x86.
Performance Class / Device Policy	Not defined formally.	Defined **Performance Classes (PC13, PC14, PC15)** to ensure predictable app experience tiers.
Developer Images	Full system rebuild required for testing.	**GSI (Generic System Image)** can boot on any Treble-compliant device for validation.
Power Management	Device-specific Power HALs.	**AIDL Power HAL 3.0** with adaptive thermal + performance hinting APIs.
Display / Multi-Display Support	Basic single-display support.	Extended **multi-display and automotive display stack** (CarDisplayService, DisplayArea policy).
Automotive (AAOS)	Framework integrated with main Android image.	Independent AAOS architecture (separate CarService, VHAL, VHAL Binder interface).
Security Updates	OEM + carrier dependent.	Modular APEX + Mainline ensures monthly framework & kernel security patches from Google.

The Android partition table

Classic Android Partition Roles (Pre-A/B)

Partition Name	Mounted Path / Use	What It Contains	Purpose / Why Needed
bootloader	N/A	Low-level boot code (PBL/SBL/LK/U-Boot/ABL)	Initializes DRAM, clocks, verifies signature, loads kernel. Essential for boot chain.
boot	-	**Kernel + Ramdisk + DTB** (packaged as `boot.img`)	Contains the Linux kernel and initial ramdisk. The device boots from here into user space (`/init`).
system	`/system`	Framework, Android OS binaries, default apps	Holds core Android framework, libraries, and system apps (Settings, Launcher, etc.).
vendor	`/vendor`	HAL binaries, device-specific libs, firmware	OEM/vendor-specific drivers and HALs needed for the SoC and peripherals. Decouples BSP from system image.
data	`/data`	User data, installed apps, app data, cache	Stores everything belonging to the user. Wiped during factory reset.
cache	`/cache`	Temporary OTA and Dalvik cache data	Used by OTA updater and app runtime cache (obsolete in newer Android).
recovery	Bootable image (alternate boot path)	Mini Linux + recovery app (UI + update tools)	Used to factory reset or flash updates manually (Volume+Power boot).
misc	N/A	Boot flags, reboot reason, etc.	Small config area used by bootloader and recovery to communicate (boot modes).
radio / modem	N/A	Modem firmware blobs	Stores baseband firmware for cellular modem.
persist	`/persist`	Factory calibration data (Wi-Fi, camera, sensors, IMEI)	Holds non-volatile data that must survive factory reset.
dtbo	-	Device Tree Overlays	Hardware configuration overrides used at boot.
vbmeta	-	AVB metadata and verification keys	Used by Android Verified Boot (AVB 2.0) for integrity checks.

A/B (Seamless Update) Partition Scheme (Android 7–10)

Partition	Purpose	How It Works in A/B
boot_a / boot_b	Kernel + ramdisk	Device runs from current slot (say `_a`); OTA flashes new image into inactive slot (`_b`).
system_a / system_b	Android framework	Seamless OTA: after update, bootloader switches slot.
vendor_a / vendor_b	HAL + BSP	Each slot carries matching vendor libs for its system image.
product_a / product_b	Optional partition for OEM apps	Also duplicated for A/B safety.
vbmeta_a / vbmeta_b	Verification metadata	Ensures chain of trust for each slot.
userdata	Common (non-A/B)	Shared, not duplicated — user data persists.

NOTE => A/B system means *two sets of partitions* → "slot A" and "slot B".

Why introduced:

- To avoid update failures or "bricks".
- If update on slot `_b` fails, bootloader keeps using `_a` (safe rollback).

Dynamic Partitions (Android 10+)

Dynamic partitions were introduced to **remove hard-coded partition sizes** and **enable resizing during OTA**.

- A new **super partition** wraps logical partitions:
- `super -> [ system_a, vendor_a, product_a, system_ext_a, odm_a, ... ]`
- Logical partitions behave like real ones but are stored in one big container.
- **`lpmake`** tool manages creation and resizing.
- Managed by **update_engine** during OTA.

Why Dynamic Partition -

- Easier OTA without reflashing partition tables.
- Future-proof (OEM can add/remove logical partitions).
- Enables Google's modular architecture (Project Treble).

Modern Android (A12–A15) Storage Layout

Partition	What It Does (Modern Android)	Notes
boot	Contains kernel + ramdisk for GKI	Verified by AVB chain. May include recovery-ramdisk (recovery merged).
vendor_boot	Vendor-specific ramdisk + DTBO	Allows vendor init scripts and kernel modules to load separately from system.
vbmeta / vbmeta_system / vbmeta_vendor	AVB metadata and keys	Provide verified-boot chain per partition group.
system	Core Android framework (`/system`)	GSI / framework layer. Now purely generic.
system_ext	OEM's extension to `/system`	For partially customized framework or settings.
product	Product-specific apps and overlays	OEM UI, apps, and themes.
vendor	HALs, device drivers, firmware	Contains vendor AIDL/HIDL HALs and binary blobs.
vendor_dlkm	Vendor loadable kernel modules	Supports GKI 2.0 modularization.
odm / odm_dlkm	Optional device modules	For ODM-level (board-specific) HALs or modules.
data	User data partition	Apps, settings, user content. File-based encryption mandatory.
metadata	AVB metadata, encryption keys, OTA metadata	Used for verified boot, FBE, and OTA state tracking.
super	Logical container for dynamic partitions	Holds system/vendor/product/system_ext, resizable.
persist	Calibration, Wi-Fi MAC, camera lens data	Non-volatile, preserved through OTA/factory reset.
misc	Bootloader <-> Android message area	Reboot reason, recovery command flags.
recovery (optional)	Recovery ramdisk (if not merged)	On modern devices, merged into boot/vendor_boot.

Partition Information for TCC805x EVB

FWDN Partition	Size (KB)	Label Name	Block Device	Description	File Name
Partition 1	2048	bl3_ca72_a	mmcblk0p1	Main Core Boot Loader A	ca72_bl3.rom
Partition 2	2048	bl3_ca72_b	mmcblk0p2	Main Core Boot Loader B	ca72_bl3.rom
Partition 3	2048	bl3_ca53_a	mmcblk0p3	Sub Core Boot Loader A	ca53_bl3.rom
Partition 4	2048	bl3_ca53_b	mmcblk0p4	Sub Core Boot Loader B	ca53_bl3.rom
Partition 5	36864	boot_a	mmcblk0p5	Kernel image A	boot.img
Partition 6	36864	boot_b	mmcblk0p6	Kernel image B	boot.img
Partition 7	4124672	super	mmcblk0p7	Resizable image (system.img + vendor.img)	super.img
Partition 8	8192	dtbo_a	mmcblk0p8	Device tree Overlay A	dtbo.img
Partition 9	8192	dtbo_b	mmcblk0p9	Device tree Overlay B	dtbo.img
Partition 10	153600	cache	mmcblk0p10	Cache partition	cache.img
Partition 11	1024	env	mmcblk0p11	Environment	-
Partition 12	5120	splash	mmcblk0p12	splash partition	-
Partition 13	1024	misc	mmcblk0p13	misc partition	-
Partition 14	1024	subcore_misc	mmcblk0p14	Sub-core misc partition	-
Partition 15	1024	tcc	mmcblk0p15	tcc partition	-
Partition 16	8192	sest	mmcblk0p16	sest partition	-
Partition 17	40960	subcore_boot_a	mmcblk0p17	Sub-core Kernel Image	tc-boot-tcc8050-sub.img
Partition 18	40960	subcore_boot_b	mmcblk0p18	Sub-core Kernel Image	tc-boot-tcc8050-sub.img
Partition 19	409600	subcore_root_a	mmcblk0p19	Sub-core Device tree	telechips-subcore-image-tcc8050-sub.ext4
Partition 20	409600	subcore_root_b	mmcblk0p20	Sub-core Device tree	telechips-subcore-image-tcc8050-sub.ext4
Partition 21	1024	subcore_dtb_a	mmcblk0p21	Sub-core root file system	tcc8050-linux-subcore_sv1.0-tcc8050-sub.dtb
Partition 22	1024	subcore_dtb_b	mmcblk0p22	Sub-core root file system	tcc8050-linux-subcore_sv1.0-tcc8050-sub.dtb
Partition 23	1024	vbmeta_a	mmcblk0p23	Hashtree used for Android Verified Boot (AVB)	vbmeta.img
Partition 24	1024	vbmeta_b	mmcblk0p24	Hashtree used for Android Verified Boot (AVB)	vbmeta.img
Partition 25	8192	tamper_evidence	mmcblk0p25	tamper_evidence partition	-
Partition 26	16384	metadata	mmcblk0p26	metadata partition	-
Partition 27	1000	subcore_env	mmcblk0p27	Sub-core Environment	-
Partition 28	40000	subcore_splash	mmcblk0p28	Sub-core splash partition	splash_1920x720x32.img
Partition 29	0	userdata	mmcblk0p29	User storage	-

Note:
Partitions (`vbmeta`, `tamper_evidence`, and `metadata`) are used for Android Verified Boot (AVB).

- For more details, refer to https://android.googlesource.com/platform/external/avb/+/master/README.md.

Practice

- Telechip build, and flash step
- A14 Android build, and flash step
- A14 AAOSP Demo with Emulator

Q & A

Thank you

The Android Build System

Packages and modules

- An Android product is built from **packages** and **modules**
  - a package is an Android application (APK)
  - a module is anything else, e.g. native daemon, library, script

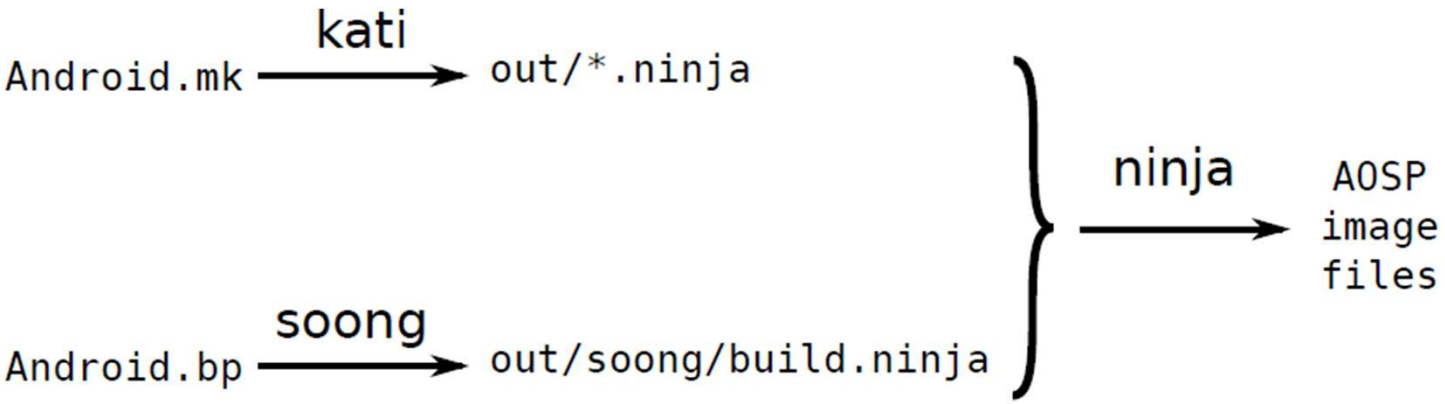
- The list is in Make variable PRODUCT_PACKAGES

- Packages and modules are defined in **Android.bp** and **Android.mk** files

NOTE : you can generate a list of all packages and modules in using our script -> **list-product-packages.sh** to count the number of packages and modules

AOSP build system

- Up to M/6 the build system was based on GNU Make
- In N/7 and later, the build is run by **ninja**
- Ninja works from a pre-processed manifest generated by **kati** and **soong**



Blueprint and Android.bp

- Starting with O/8, Android.mk files are being replaced by Android.bp files
- Android.bp files are written in **Blueprint** ("JSON-like")
- Blueprint is only used in AOSP
- Reference: https://source.android.com/setup/build

Example Android.bp

- A simplified version of the Android.bp for logcat

```
system/logging/logcat
|-- Android.bp
|-- logcat.cpp
```

```
cc_binary {
name: "logcat",
srcs: ["logcat.cpp"],
shared_libs: ["libbase", "libprocessgroup",],
cflags: ["-Werror"],
}
```

- The module is called **"logcat“**
- Has one source file: **logcat.cpp**
- Links with libraries **"libbase",** and **"libprocessgroup“**
- Builds an executable which will be installed into **$OUT/system/bin/logcat**

https://cs.android.com/android/platform/superproject/main/+/main:system/logging/logcat/Android.bp

Blueprint syntax

- C-style comments: // and /* */
- Divided into **modules**, e.g. cc_binary
- The body of a module is delineated by braces ’{’ and ’}’
- The body is a list of properties consisting of name: value,
- Values can be
  - Boolean, e.g. vendor: true,
  - String, e.g. name: "lmkd",
  - List, e.g. shared_libs: ["liblog", "libprocessgroup"],

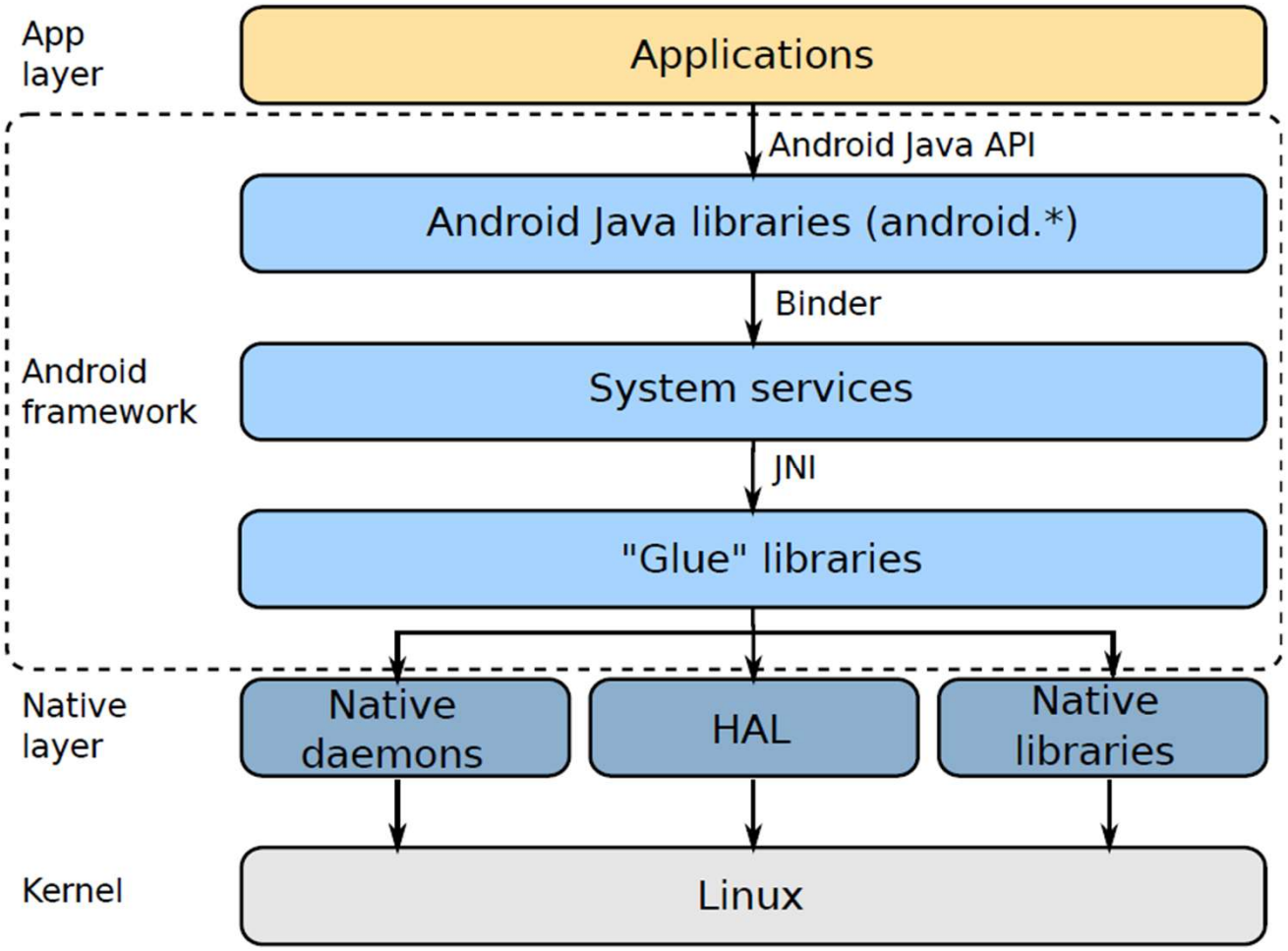
Blueprint modules

Examples of types of module

`cc_binary`	Native binary
`cc_library_shared`	Shared library
`cc_library_static`	Static library
`cc_binary_host`	Host binary
`cc_library_host_shared`	Host shared library
`cc_library_host_static`	Host static library
`java_library`	Java library
`android_app`	Android app
`prebuilt_etc`	Prebuilt installed into etc
`cc_prebuilt_binary`	Prebuilt installed into bin

The Android Framework

Architecture



Android Framework layer

- Android Java libraries
  - implement the Android API
  - have the same privilege as the calling app

- System Services
  - a collection of objects that make up the Android operating system
  - run in different process to the app
  - have elevated privileges
  - enforce the Android permissions

- "Glue" libraries
  - connect system services (written mostly in Java) to the native layer

System Services

- The framework implements an object-oriented operating system on top of a conventional Unix-like operating system (Linux).
- Object-oriented means a collection of services that can be invoked remotely.
- Services perform system-level tasks on behalf of the caller, e.g.:
  - Making the device vibrate (VibratorService)
  - Managing power states (PowerManager)
  - Managing activity state (Activity Manager)
- Services use system UIDs that allow them to access system resources
- There are several tools that can be used to monitor and interact with system services

System Services

- Listing the main system services:

```
# service list
```

- Listing vendor system services (may not be built by default on Cuttlefish):

```
# vndservice list
```

- Finding out what process is running the main system server process:

```
# ps -A | grep system_server
```

- Dumping the internal state of all system services:

```
# dumpsys
```

System Services

- Dumping the internal state of a single system service (Activity Manager in this case):

```
# dumpsys activity
```

- Checking the logs for the system server process' PID (replace [] value):

```
# logcat --pid=[PID RETURBED BY PS COMMAND ABOVE]
```

- Checking the logs for output from a given system service – WindowManager for ex:

```
# logcat | grep -i windowmanager
```

- Communicate with the activity manager:

```
# am
```

- Communicate with the package manager:

```
# pm
```

- Communicate with the window manager:

```
# wm
```

Getting information with dumpsys

- Most services implement the dump() function which displays useful debug information about the service
- You can call dump() using the command dumpsys

```
# dumpsys SurfaceFlinger
Build configuration: [sf NO_RGBX_8888] [libui] [libgui]
Sync configuration: [using: EGL_KHR_fence_sync]
Visible layers (count = 6)
+ LayerDim 0x40b2f4e0 (DimLayer)
  Region transparentRegion (this=0x40b2f644, count=1)
    [  0,   0,   0,   0]
  Region visibleRegion (this=0x40b2f4e8, count=1)
    [  0,   0,   0,   0]
[...]
```

Command shell: cmd

- Some services implement a command-line shell
- Call it using cmd [service name]
- For example, the display service:

```
# cmd display
Display manager commands:
  help
    Print this help text.

  set-brightness BRIGHTNESS
[...]
```

- cmd is implemented by onShellCommand. For an example, see
- frameworks/base/services/core/java/com/android/server/display/
- DisplayManagerService.java

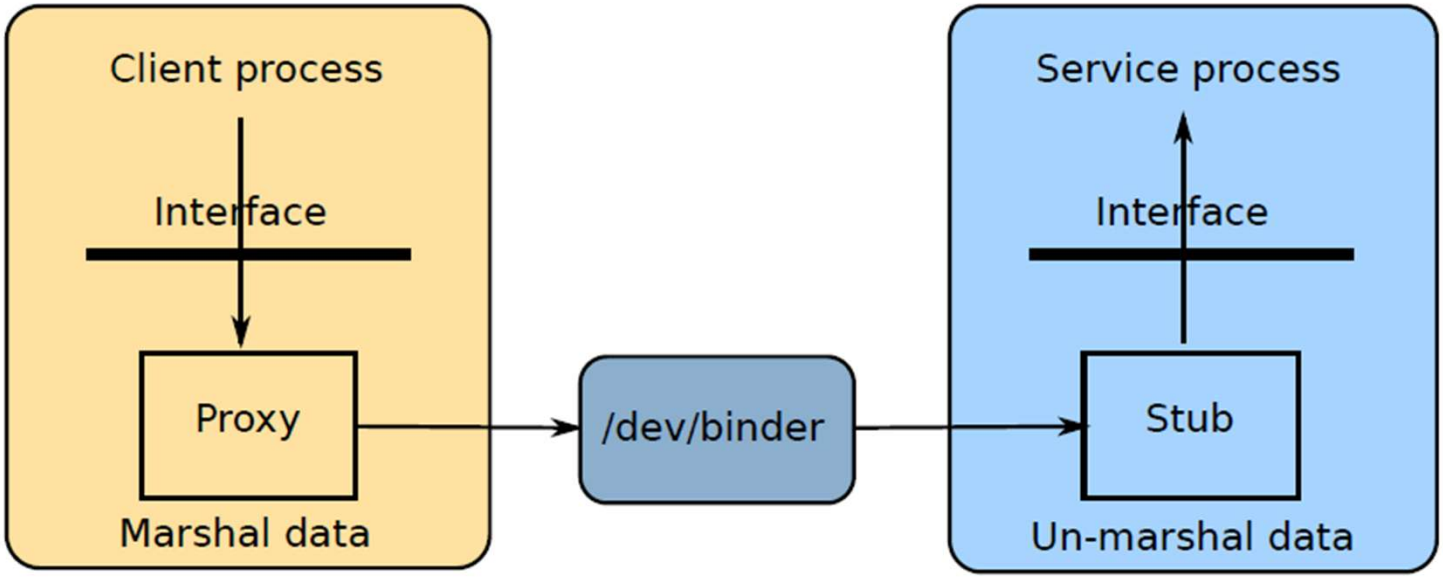
Practice :- show statusbar Example

Binder

- Binder is a lightweight, high-performance IPC framework built into both Android's user-space and the Linux kernel. It allow to communication between apps, system services, and hardware abstraction layers (HALs).
- We call the client a **manager** : Wants to use a service (e.g., LocationManager, MediaPlayer).
- And we call the server a **service :** Provides a function (e.g., LocationService, MediaService).
- In Binder messages are called **parcels**: Data container for requests; it includes sender's credentials (UID, PID) and has a size limit (usually 1 MiB).
- In Binder the interfaces are defined in **AIDL** AIDL (Android Interface Definition Language): Defines the interface the client and server use to communicate, making it easier to use remote services in code.

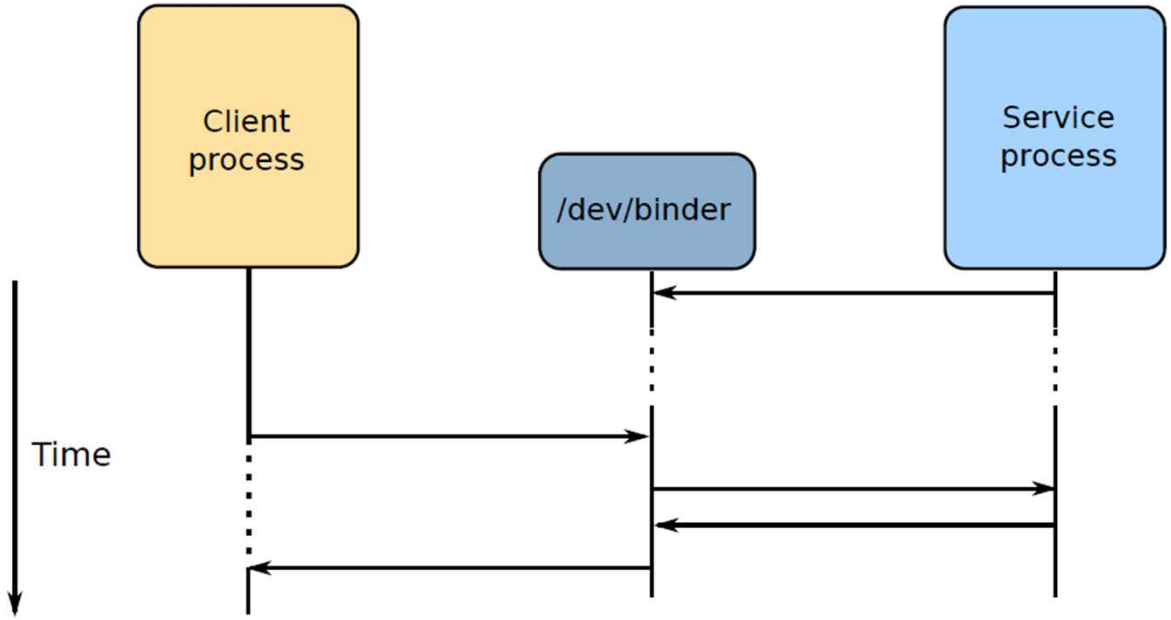
Remote invocation

- Remote invocation of a method on the service is achieved using a proxy in the client and a stub in the service
- The data elements are marshalled (serialized) in the proxy and unmarshalled in the stub



Control flow

- Parcels generated by the manager (client) are routed via binder to the service

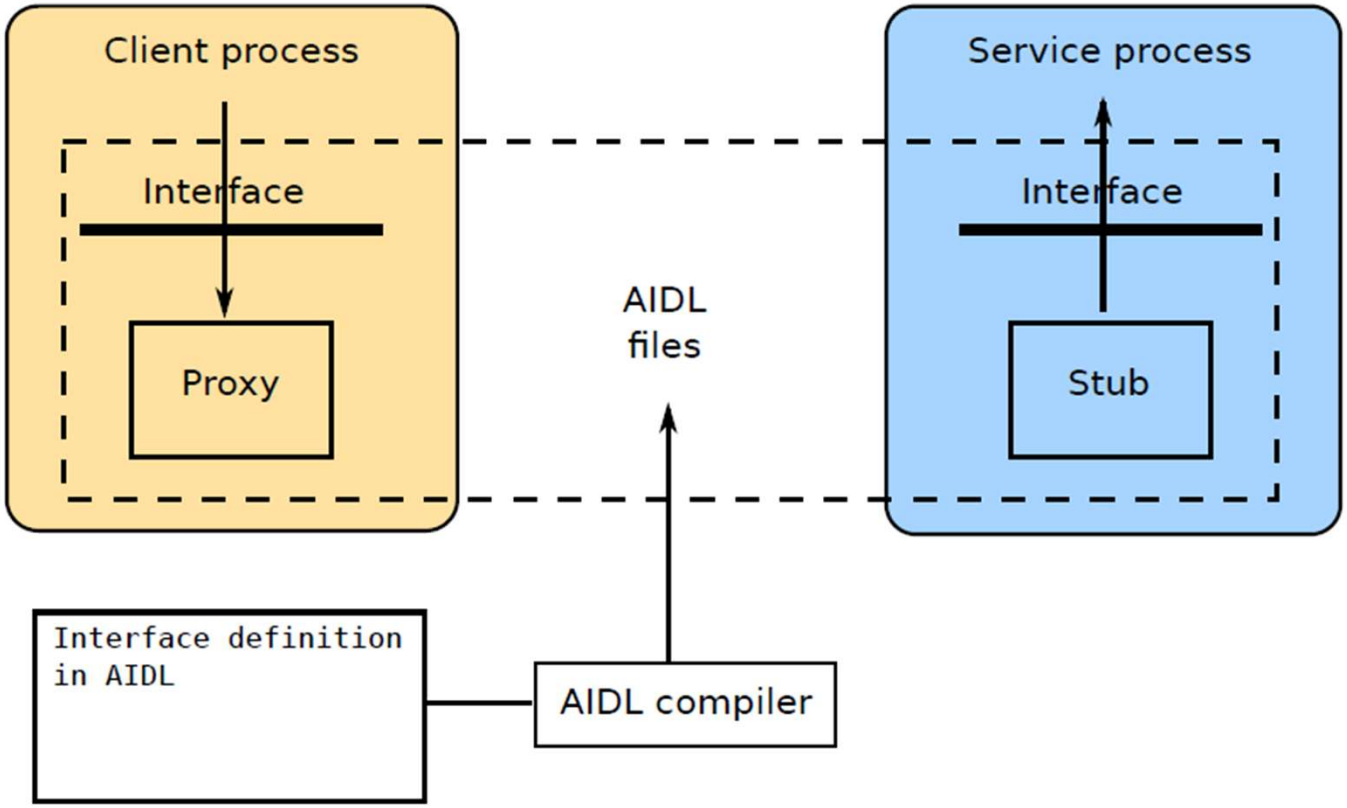


Practice :- Binder Example

AIDL

AIDL

- The Android Interface Definition Language takes care of creating the code for the proxy and stub for system services written in Java



AIDL language

- AIDL is "Java-like“
- Example: part of IPowerManager.aidl

```
package android.os;

import android.os.WorkSource;
import android.os.PowerSaveState;

interface IPowerManager {
[...]
        boolean isInteractive();
        boolean isPowerSaveMode();
        PowerSaveState getPowerSaveState(int serviceType);
        boolean setPowerSaveMode(boolean mode);
        boolean isDeviceIdleMode();
        boolean isLightDeviceIdleMode();
[...]
}
```

https://source.android.com/devices/architecture/aidl/overview

Call semantics

- By default, calls are synchronous
- Declare a method **oneway** to make it asynchronous
- Example: from IPowerManager.aidl

```
[...]
interface IPowerManager {
[...]
        oneway void powerHint(int hintId, int data);
[...]
```

- A oneway call does not block, it simply sends the parcel and returns immediately
- The implementation of the interface receives this as a regular call from the Binder thread pool

AIDL data types

- Simple types: boolean, char, int, long, float
  - String
  - CharSequence

- Objects that implement the Parcelable interface
  - writeToParcel: writes the current state of the object to a parcel
  - readFromParcel: sets the state from a parcel
  - Objects are passed by reference and must be marked in AIDL as in, out or inout, e.g.

- boolean setAdaptivePowerSavePolicy(in BatterySaverPolicyConfig config);

Compiling AIDL

- The **aidl compiler** takes the interface definition and generates two classes
  - Proxy: same name as the interface (e.g. IPowerManager)
  - Stub: named [interface].Stub, e.g. (IPowerManager.Stub)

- Use the proxy when writing the manager

- Use the stub for the service

- Since O/8 you can generate C++ bindings using **aidl-cpp**. See - system/tools/aidl/docs/aidl-cpp.md for details

Example: SimpleManager

- Define the interface, ISimpleManager.aidl

```
package com.example.simplemanager;
interface ISimpleManager {
int addInts (int a , int b);
String echoString (String s);
}
```

- Generates classes
  - manager proxy: ISimpleManager
  - service stub: ISimpleManager.Stub

Implementing the service

- Extend AIDL-generated stub class ISimpleManager.Stub and implement the interface Methods

- ISimpleServiceImpl.java

```
package com.example.simpleservice;
import com.example.simplemanager.ISimpleManager;
class ISimpleServiceImpl extends ISimpleManager.Stub {
    public int addInts(int a, int b) {
        return a + b;
    }
    public String echoString(String s) {
        return s;
    }
}
```

Dumpsys

- Implement dump() if you want to return status information via dumpsys

  # dumpsys simpleservice

- The prototype is

```
void dump(FileDescriptor fd, PrintWriter writer, String[] args)
fd      The raw file descriptor that the dump is being sent to
writer  The PrintWriter to which you should dump your state
args    Additional arguments to the dump request
```

  - For example

```
import java.io.PrintWriter;
import java.io.FileDescriptor;
class ISimpleServiceImpl extends ISimpleManager.Stub {
[...]
    public void dump(FileDescriptor fd, PrintWriter pw, String[] args) {
                pw.println("Dump from simple service");
                pw.println("end");
    }
}
```

Practice :- System Service Example

Hardware Abstraction Layers

What is the HAL for?

- The Hardware Abstraction Layer connects system services in the Framework to the hardware
- OEMs can tune the HAL to work with their specific hardware
- HALs are part of the BSP, located on the vendor partition
- Most HALs are implemented as daemons (native services)
- Uses Binder message passing
- Beginning in O/8, HAL interfaces defined in **HIDL** (Hardware Interface Definition Language)
- From R/11, HAL interfaces transition from HIDL to **Stable AIDL** (Android Interface Definition Language)

Project Treble: objectives

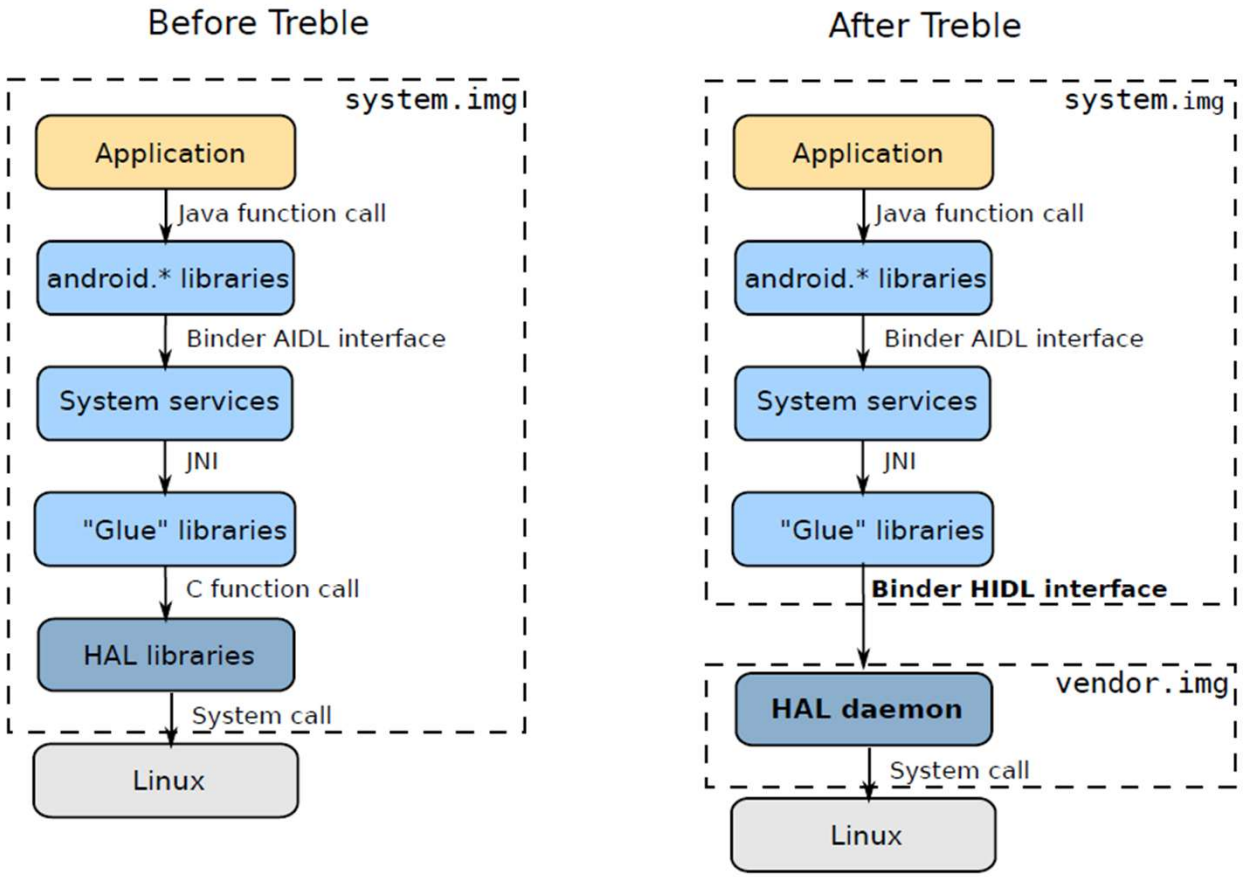
- Prior to O/8, everything (framework, HAL, pre-installed apps) was combined into a single image file, system.img
- In O/8, as part of **Project Treble**, Android was split into two parts: **system** and **vendor**
- In recent versions of AOSP -
  - system: contains framework, framework extensions and pre-installed apps deployed across system.img, system_ext.img and product.img
  - vendor: contains the board support package and device additions in vendor.img and odm.img
- **System and vendor parts can be updated independently**
  - for example, you can update the system part to U/14, leaving the vendor part on T/13
  - makes it easier for OEMs to update a product with a new version of AOSP

Project Treble: objectives

- The vendor part consists of the HAL and supporting configuration files and libraries
- The HAL API is **strongly versioned** so that old and new HALs can coexist
- The interface uses **message passing** (binder) rather than procedure calls; each HAL is implemented as a daemon (*)
- The HAL versions are documented in the Vendor INterFace, **VINTF**
- Dependencies between the vendor and system parts are controlled by linker namespaces that are part of the **VNDK**

- (*) there are exceptions – Its not applicable for passthrough HALs

Project Treble:



HALs and applications

- Generally, it is not possible for an application to call a HAL daemon directly
- Prevented by SEPolicy
- Necessary because HIDL does not pass UID/PID to the daemon, so it cannot do any authentication
- (also, Android permissions model does not extend down as far as HAL)

List of mandatory device HALs

These HALs are marked with `optional="false"` in `hardware/interfaces/compatibility_matrices/compatibility_matrix.5.xml`

Interface	Version
android.hardware.audio	6.0
android.hardware.audio.effect	6.0
android.hardware.gatekeeper	1.0
android.hardware.graphics.allocator	2.0,2.0,4.0
android.hardware.graphics.composer	2.1-4
android.hardware.graphics.mapper	2.1,3.0,4.0
android.hardware.health	2.1
android.hardware.keymaster	3.0,4.0-1
android.hardware.power	

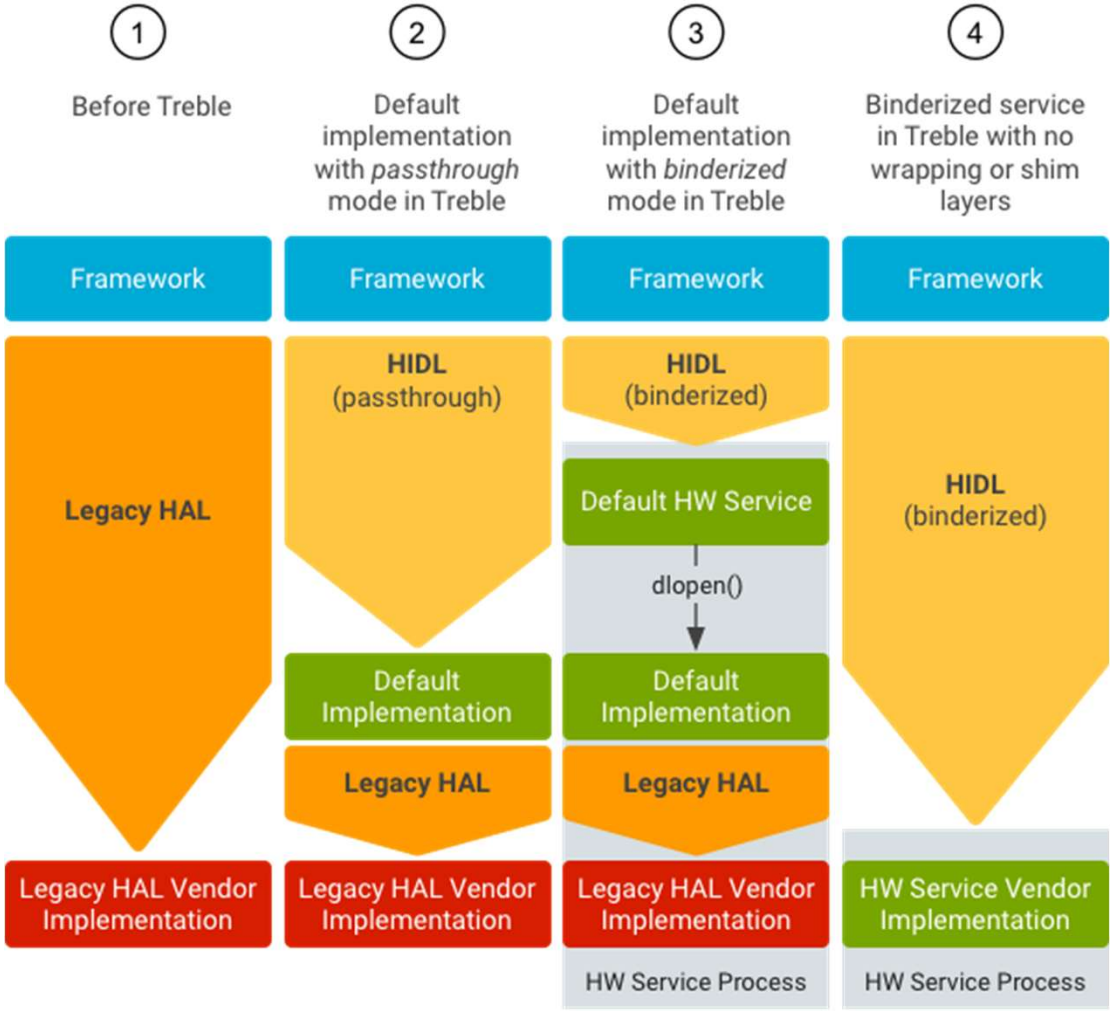
List of optional device HALs

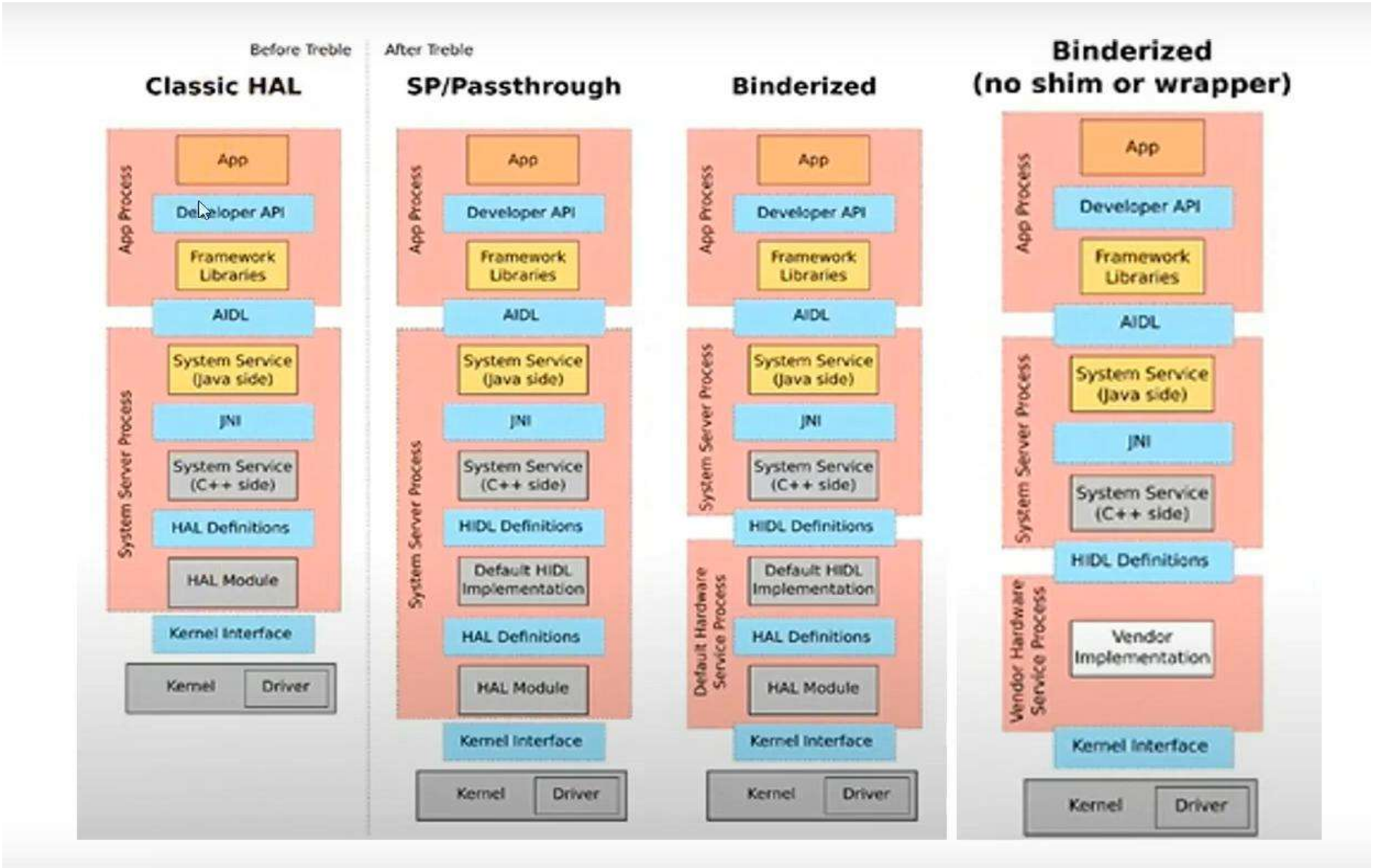
Interface	Version
android.hardware.vibrator	1.0-3
android.hardware.vr	1.0
android.hardware.weaver	1.0
android.hardware.wifi	1.0-3
android.hardware.wifi.hostapd	1.0-1
android.hardware.wifi.supplicant	1.0-2

Interface	Version
android.hardware.atrace	1.0
android.hardware.authsecret	1.0
android.hardware.biometrics.face	1.0
android.hardware.biometrics.fingerprint	2.1
android.hardware.bluetooth	1.0
android.hardware.bluetooth.audio	2.0
android.hardware.boot	1.0
android.hardware.broadcastradio	2.0
android.hardware.camera.provider	2.4-5
android.hardware.cas	1.0
android.hardware.configstore	1.1
android.hardware.confirmationui	1.0
android.hardware.contexthub	1.0

Interface	Version
android.hardware.drm	1.0-2
android.hardware.dumpstate	1.0
android.hardware.gnss	2.0
android.hardware.health.storage	1.0
android.hardware.input.classifier	1.0
android.hardware.ir	1.0
android.hardware.light	2.0
android.hardware.media.c2	1.0
android.hardware.media.omx	1.0
android.hardware.memtrack	1.0
android.hardware.neuralnetworks	1.0-2
android.hardware.nfc	1.2
android.hardware.oemlock	1.0

Interface	Version
android.hardware.power	1.0-3
android.hardware.power.stats	1.2
android.hardware.radio	1.0-2
android.hardware.radio.config	1.1
android.hardware.renderscript	1.0
android.hardware.secure_element	1.0
android.hardware.sensors	2.0
android.hardware.soundtrigger	2.0-2
android.hardware.tetheroffload.config	1.0
android.hardware.tetheroffload.control	1.0
android.hardware.thermal	2.0
android.hardware.usb	1.0-2
android.hardware.usb.gadget	1.0


1
Before Treble
2
Default implementation with *passthrough* mode in Treble
3
Default implementation with *binderized* mode in Treble
4
Binderized service in Treble with no wrapping or shim layers
Framework
Framework
Framework
Framework
Legacy HAL
HIDL (passthrough)
HIDL (binderized)
Default HW Service
dlopen()
HIDL (binderized)
Default Implementation
Default Implementation
Legacy HAL
Legacy HAL
Legacy HAL Vendor Implementation
Legacy HAL Vendor Implementation
Legacy HAL Vendor Implementation
HW Service Vendor Implementation
HW Service Process
HW Service Process


Before Treble
After Treble
Classic HAL
SP/Passthrough
Binderized
Binderized
(no shim or wrapper)
App Process
App
Developer API
Framework Libraries
AIDL
System Server Process
System Service (Java side)
JNI
System Service (C++ side)
HAL Definitions
HAL Module
Kernel Interface
Kernel
Driver
HIDL Definitions
Default HIDL Implementation
Default Hardware Service Process
Vendor Hardware Service Process
Vendor Implementation

HIDL

HIDL

- HAL daemons communicate via Binder messages sent via /dev/hwbinder with manager hwservicemanager
- Interfaces defined in a C++ style language called **HIDL**
- Tool **hidl-gen** creates proxy/stub classes in C++ or Java
- Most HAL interfaces are defined in hardware/interfaces
  - also some in frameworks/hardware/interfaces
  - and system/hardware/interfaces

HIDL

- HAL daemons communicate via Binder messages sent via /dev/hwbinder with manager hwservicemanager
- Interfaces defined in a C++ style language called **HIDL**
- Tool **hidl-gen** creates proxy/stub classes in C++ or Java
- Most HAL interfaces are defined in hardware/interfaces
  - also some in frameworks/hardware/interfaces
  - and system/hardware/interfaces

HIDL example

hardware/interfaces/light/2.0/ILight.hal

```
package android.hardware.light@2.0;

interface ILight {
    setLight(Type type, LightState state) generates (Status status);
    getSupportedTypes() generates (vec<Type> types);
};
```

- Version **@2.0**: allows backwards compatibility - framework will choose the latest version of HAL supported by vendor
- Return value defined by **generates**
- Variables are typed (e.g. Status, Type) – will check this in next slide

HIDL example

hardware/interfaces/light/2.0/ILight.hal

```
package android.hardware.light@2.0;

enum Status : int32_t {
    SUCCESS,
    LIGHT_NOT_SUPPORTED,
    BRIGHTNESS_NOT_SUPPORTED,
    UNKNOWN,
};
enum Type : int32_t {
    BACKLIGHT,
    KEYBOARD,
    BUTTONS,
    BATTERY,
    NOTIFICATIONS,
    ATTENTION,
    BLUETOOTH,
    WIFI,
    COUNT,
};
[...];
```

hidl-gen

- **hidl-gen** generates code from the HAL file
- Example:

```
$ hidl-gen -L c++-impl -o device/sirius/marvin/light \
-r vendor.example:vendor/sirius android.hardware.light@2.0
$ tree device/sirius/marvin
device/sirius/marvin
`-- light
    |-- Light.cpp
    `-- Light.h
```

The template HAL code

Light.cpp

```
#include "Light.h"

namespace android::hardware::light::implementation {

    Return<::android::hardware::light::V2_0::Status> Light::setLight(::android::hardware::ligh
t::V2_0::Type type, const ::android::hardware::light::V2_0::LightState& state) {
        // TODO implement
        return ::android::hardware::light::V2_0::Status {};
    }

    Return<void> Light::getSupportedTypes(getSupportedTypes_cb _hidl_cb){
        // TODO implement
        return Void();
    }
}
```

Callbacks

- Simple types are returned by the function
- Strings, arrays and structures are returned as callbacks using `_hidl_cb`:
- Example

```
Return<void> Light::getSupportedTypes(getSupportedTypes_cb _hidl_cb){
    std::vector<V2_0::Type> types;
    types.push_back(V2_0::Type::BACKLIGHT);
    _hidl_cb(types);

    return Void();
}
```

Implementing the daemon

- Code to set the HAL daemon (native service) running

```
#include "Light.h"

int main() {
    sp<ILight> service = new Light();

    configureRpcThreadpool(1, true);
    status_t status = service->registerAsService();
    if (status == OK) {
        ALOGI("Light HAL Ready");
        joinRpcThreadpool();
    }

    ALOGE("Cannot register Light HAL service");
    return 1;
}
```

Instances

- Sometimes you need several instances of the same HAL
  - Example: IBroadcastRadio has an instance for each broadcast statndard
- Give the instance name as a parameter to `registerAsService`
  - Default name is "default"

This shows two instances of IBroadcastRadio, "amfm" and "dab":

```
    status_t status = service->registerAsService("dab");
```

```
$ lshal | grep IBroadcastRadio
DM,FC Y android.hardware.broadcastradio@2.0::IBroadcastRadio/amfm   0/2   456   678 248
DM,FC Y android.hardware.broadcastradio@2.0::IBroadcastRadio/dab    0/2   456   678 248
```

Running the HAL daemon

- Install the daemon into `/vendor/bin/hw` or `/system/bin/hw`
- Install the rc init script into `/vendor/etc/init` or `/system/etc/init`

In the case of the lights HAL, this rc file would be needed:

`light-service.rc`

```
service light-hal-2-0 /vendor/bin/hw/android.hardware.light@2.0-service.example
    interface android.hardware.light@2.0::ILight default
    class hal
    user system
    group system
```

Building - Android.bp

```
cc_binary {
    name: "android.hardware.light@2.0-service.example",
    vendor: true,
    relative_install_path: "hw",
    srcs: [
        "Light.cpp",
        "service.cpp",
    ],
    shared_libs: [
        "libhidlbase",
        "libutils",
        "android.hardware.light@2.0",
        "liblog",
    ],
    init_rc: ["light-service.rc"],
    vintf_fragments: ["light-service.xml"],
}
```

Must set `proprietary: true` or `vendor: true`

Setting `relative_install_path: "hw"` installs into `/vendor/bin/hw`

Listing HALs

- HIDL HALs are hardware services, which uses **/dev/hwbinder**
- We can list them using **lshal**

```
maneesh@maneesh:/media/maneesh/e88ea721-fad1-4d7c-86d4-79b39aaac659/Forvia-training-Android14/aosp$ adb devices
List of devices attached
0.0.0.0:6520    device

maneesh@maneesh:/media/maneesh/e88ea721-fad1-4d7c-86d4-79b39aaac659/Forvia-training-Android14/aosp$ adb shell
xrda3car:/ $ lshal
Warning: Skipping "android.frameworks.sensorservice@1.0::ISensorManager/default": cannot be fetched from service manager (null)
Warning: Skipping "android.hardware.audio.effect@7.0::IEffectsFactory/default": cannot be fetched from service manager (null)
Warning: Skipping "android.hardware.audio@7.0::IDevicesFactory/default": cannot be fetched from service manager (null)
Warning: Skipping "android.hardware.audio@7.1::IDevicesFactory/default": cannot be fetched from service manager (null)
Warning: Skipping "android.hardware.automotive.evs@1.0::IEvsEnumerator/default": cannot be fetched from service manager (null)
Warning: Skipping "android.hardware.automotive.evs@1.1::IEvsEnumerator/default": cannot be fetched from service manager (null)
Warning: Skipping "android.hardware.media.c2@1.0::IComponentStore/software": no information for PID 524, are you root?
Warning: Skipping "android.hidl.allocator@1.0::IAllocator/ashmem": no information for PID 374, are you root?
Warning: Skipping "android.hidl.base@1.0::IBase/ashmem": cannot be fetched from service manager (null)
Warning: Skipping "android.hidl.base@1.0::IBase/default": cannot be fetched from service manager (null)
Warning: Skipping "android.hidl.base@1.0::IBase/software": cannot be fetched from service manager (null)
Warning: Skipping "android.hidl.manager@1.0::IServiceManager/default": no information for PID 148, are you root?
| All HIDL binderized services (registered with hwservicemanager)
VINTF R Interface                                                  Thread Use Server Clients
FM    ? android.frameworks.sensorservice@1.0::ISensorManager/default N/A        N/A
DM,FC ? android.hardware.audio.effect@7.0::IEffectsFactory/default   N/A        N/A
DM,FC ? android.hardware.audio@7.0::IDevicesFactory/default          N/A        N/A
DM,FC ? android.hardware.audio@7.1::IDevicesFactory/default          N/A        N/A
FM    ? android.hardware.automotive.evs@1.0::IEvsEnumerator/default  N/A        N/A
FM    ? android.hardware.automotive.evs@1.1::IEvsEnumerator/default  N/A        N/A
FM    Y android.hardware.media.c2@1.0::IComponentStore/software      N/A        524
FM    Y android.hardware.media.c2@1.1::IComponentStore/software      N/A        524
FM    Y android.hardware.media.c2@1.2::IComponentStore/software      N/A        524
FM    Y android.hidl.allocator@1.0::IAllocator/ashmem                N/A        374
X     ? android.hidl.base@1.0::IBase/ashmem                          N/A        N/A
X     ? android.hidl.base@1.0::IBase/default                         N/A        N/A
X     ? android.hidl.base@1.0::IBase/software                        N/A        N/A
DC,FM Y android.hidl.manager@1.0::IServiceManager/default            N/A        148
FM    Y android.hidl.manager@1.1::IServiceManager/default            N/A        148
FM    Y android.hidl.manager@1.2::IServiceManager/default            N/A        148
FM    Y android.hidl.token@1.0::ITokenManager/default                N/A        148

| All HIDL interfaces getService() has ever returned as a passthrough interface;
| PIDs / processes shown below might be inaccurate because the process
| might have relinquished the interface or might have died.
| The Server / Server CMD column can be ignored.
| The Clients / Clients CMD column shows all process that have ever dlopen'ed
```

Practice :- HIDL HAL Example

AIDL for HAL

AIDL for HALs

- From R/11 onwards HAL interfaces are migrating from HIDL to **stable AIDL**
- New HAL interfaces will have interfaces written in AIDL only; HIDL isdeprecated
- https://source.android.com/devices/architecture/aidl/aidl-hals

The migration from HIDL to AIDL

- The table below(*) shows how the number of AIDL HALs has increased since R/11

Version	Total HIDL HALs	HIDL only HALs	Total AIDL HALs	AIDL only HALs
11	91	86	11	6
12	92	72	28	12
13	92	48	58	21
14	93	35	80	28

- There are still HALs that only have an HIDL interface. They are all deprecated interfaces, and can be ignored from now onwards.

- **NOTE :** This table and numbers are generated using the script $HOME/android/count-hals.sh… SO based on your Android version it may increase or decrease.

AIDL used for HAL

- AIDL uses /dev/binder, **not** /dev/hwbinder
- Use NDK backend for vendor code (e.g. HAL daemons) so that you link with libbinder_ndk and not the vendor libbinder (which has an unstable API)

Directory layout of an interface

`hardware/interfaces/light/aidl`

```
|-- aidl_api
|   |-- android.hardware.light
|       |-- 1                          <-- interface version 1
|       |   |-- android
|       |       |-- hardware
|       |           |-- light
|       |               |-- BrightnessMode.aidl
|       |                   [...]
|       |-- 2                          <-- interface version 2
|       |   |-- android
|       |       |-- hardware
|       |           |-- light
|       |               |-- BrightnessMode.aidl
|       |                   [...]
|       |-- current                    <-- current version (copy of version 2)
|-- android
|   |-- hardware                       <-- interface *with comments*
|       |-- light
|           |-- BrightnessMode.aidl
|           |-- FlashMode.aidl
|           |-- HwLight.aidl
|           |-- HwLightState.aidl
|           |-- ILights.aidl
|           |-- LightType.aidl
|-- Android.bp
|-- default                            <-- sample implementation
|-- vts                                <-- tests
```

Building the AIDL interface

hardware/interfaces/light/aidl/Android.bp

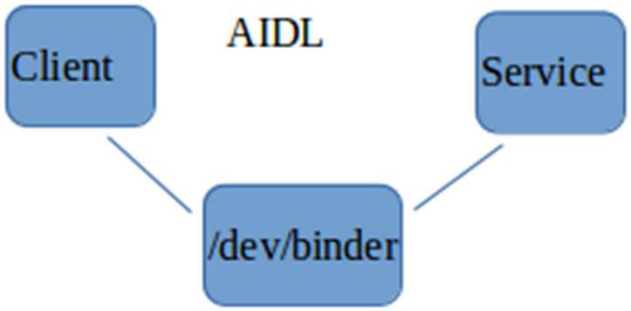
```
maneesh@maneesh:/media/maneesh/e88ea721-fad1-4d7c-86d4-79b39aaac659/Forvia-training-Android14/aosp$ cat hardware/interfaces/light/aidl/Android.bp
package {
    // See: http://go/android-license-faq
    // A large-scale-change added 'default_applicable_licenses' to import
    // all of the 'license_kinds' from "hardware_interfaces_license"
    // to get the below license kinds:
    //   SPDX-license-identifier-Apache-2.0
    default_applicable_licenses: ["hardware_interfaces_license"],
}

aidl_interface {
    name: "android.hardware.light",
    vendor_available: true,
    srcs: [
        "android/hardware/light/*.aidl",
    ],
    stability: "vintf",
    backend: {
        java: {
            sdk_version: "module_current",
        },
    },
    versions_with_info: [
        {
            version: "1",
            imports: [],
        },
        {
            version: "2",
            imports: [],
        },

    ],

}
maneesh@maneesh:/media/maneesh/e88ea721-fad1-4d7c-86d4-79b39aaac659/Forvia-training-Android14/aosp$
```

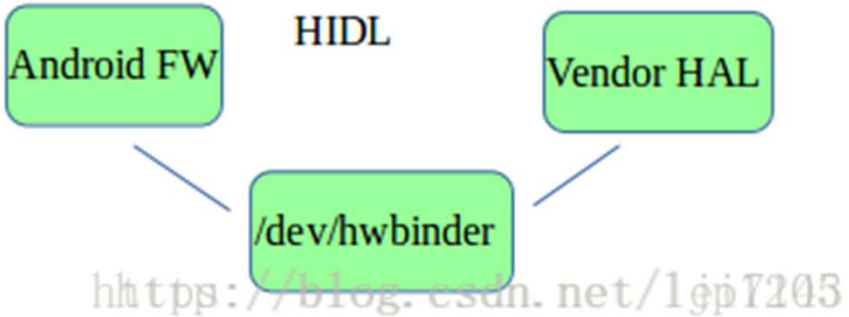
Listing HALs

- AIDL HALs are system services, using **/dev/binder**
- We can list them using **service list**
- They do not appear when you run **lshal** (which uses **/dev/hwbinder**)

```
130|xrda3car:/ $ service list | grep [^[]android.hardware
22      android.hardware.audio.core.IConfig/default: []
23      android.hardware.audio.core.IModule/bluetooth: []
24      android.hardware.audio.core.IModule/default: []
25      android.hardware.audio.core.IModule/r_submix: []
26      android.hardware.audio.core.IModule/stub: []
27      android.hardware.audio.core.IModule/usb: []
28      android.hardware.audio.effect.IFactory/default: []
29      android.hardware.authsecret.IAuthSecret/default: []
30      android.hardware.automotive.audiocontrol.IAudioControl/default: []
31      android.hardware.automotive.can.ICanController/default: []
32      android.hardware.automotive.evs.IEvsEnumerator/default: [android.hardware.automotive.evs.IEvsEnumerator]
33      android.hardware.automotive.evs.IEvsEnumerator/hw/0: []
34      android.hardware.automotive.ivn.IIvnAndroidDevice/default: []
35      android.hardware.automotive.occupant_awareness.IOccupantAwareness/default: []
36      android.hardware.automotive.remoteaccess.IRemoteAccess/default: []
37      android.hardware.automotive.vehicle.IVehicle/default: []
38      android.hardware.biometrics.face.IFace/default: []
39      android.hardware.bluetooth.IBluetoothHci/default: []
40      android.hardware.bluetooth.audio.IBluetoothAudioProviderFactory/default: []
41      android.hardware.boot.IBootControl/default: []
42      android.hardware.broadcastradio.IBroadcastRadio/amfm: []
43      android.hardware.broadcastradio.IBroadcastRadio/dab: []
44      android.hardware.cas.IMediaCasService/default: [android.hardware.cas.IMediaCasService]
45      android.hardware.confirmationui.IConfirmationUI/default: []
46      android.hardware.contexthub.IContextHub/default: []
47      android.hardware.drm.IDrmFactory/clearkey: [android.hardware.drm.IDrmFactory]
48      android.hardware.gatekeeper.IGatekeeper/default: []
49      android.hardware.gnss.IGnss/default: []
50      android.hardware.graphics.allocator.IAllocator/default: [android.hardware.graphics.allocator.IAllocator]
```

AIDL vs HIDL





Practice :- AIDL HAL Example

Telechip IPC Comm

Telechips IPC Overview

- IPC allows A72, A53, and MCU to exchange data.
- Used for -
  - vehicle signals
  - Diagnostics
  - subsystem control
- Fast, low-latency communication using mailbox hardware.

IPC Driver (Linux char device)

- Exposes: /dev/tcc_ipc_ap, /dev/tcc_ipc_micom
- Handles buffering, IOCTL, blocking reads

Mailbox (MBOX) Driver

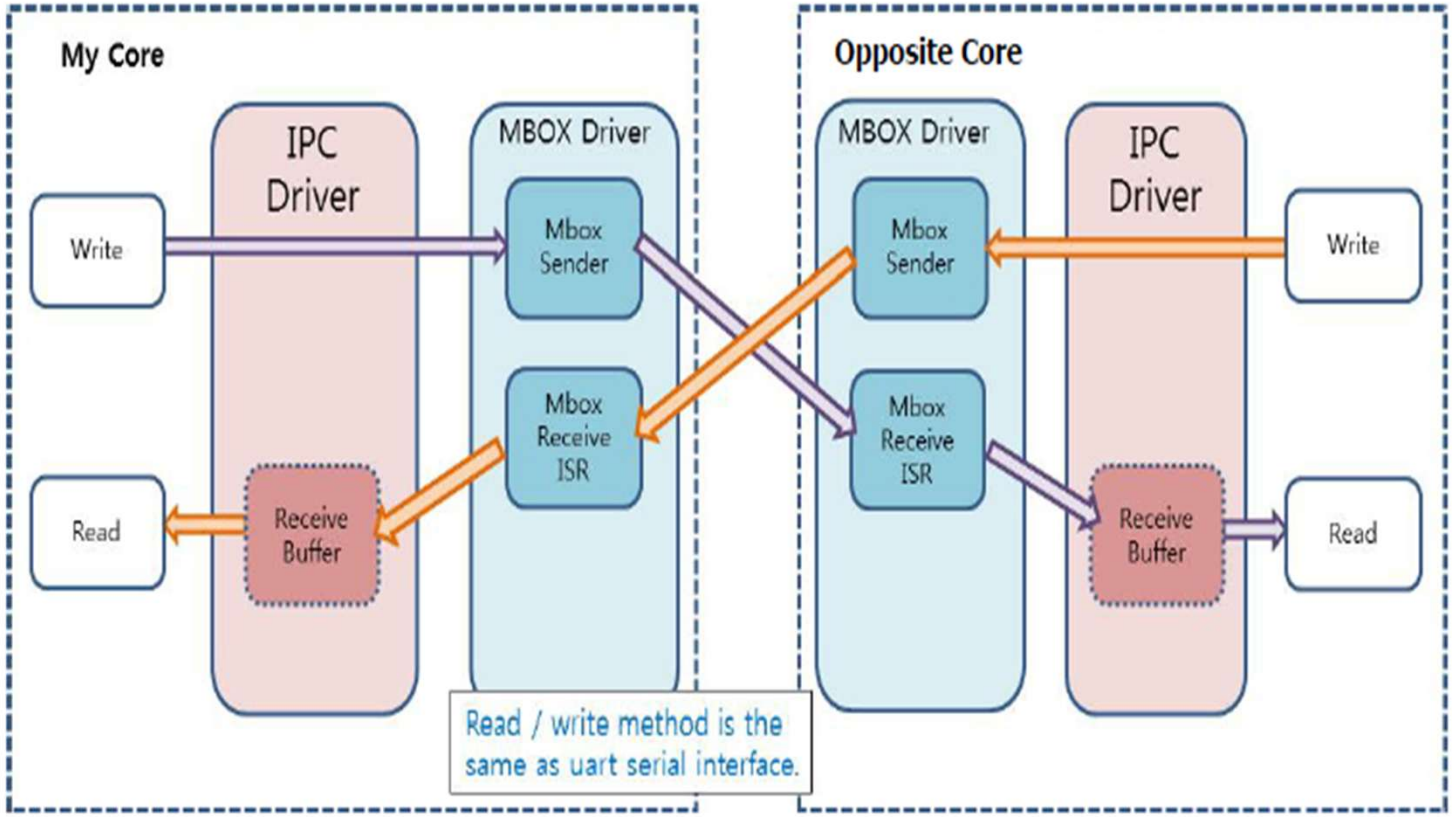
- Manages mailbox registers
- Handles interrupts between cores

Mailbox Hardware (MBOX HW)

- Dedicated hardware peripheral
- Generates doorbell IRQs
- Transfers message headers/ACK signals

https://developer.arm.com/documentation/ddi0306/b/CHDGHAIG

Telechips IPC Architecture



Mailbox Hardware

- Dedicated hardware peripheral in Telechips SoC.
- Provides channels for TX/RX.
- Generates interrupts to opposite core.

Why Mailbox?

- Very fast
- Non-blocking
- Works across heterogeneous cores
- No need for shared memory polling

IPC Driver Read/Write Flow

- TX Buffer: 512 bytes max.
- RX Buffer: 512 KB.
- IOCTL: readiness, ping test, timeout, flush.
- Behavior similar to UART but interrupt-driven.

Practice :- Telechip IPC Driver Code walkthrough

Practice :- Telechip IPC Example