



Android Architecture

Nguyen Tran (Nguyen.TranLeHoang@vn.bosch.com)

Sep 27, 2024

Day 1

Group working



QUIZ (1/10)



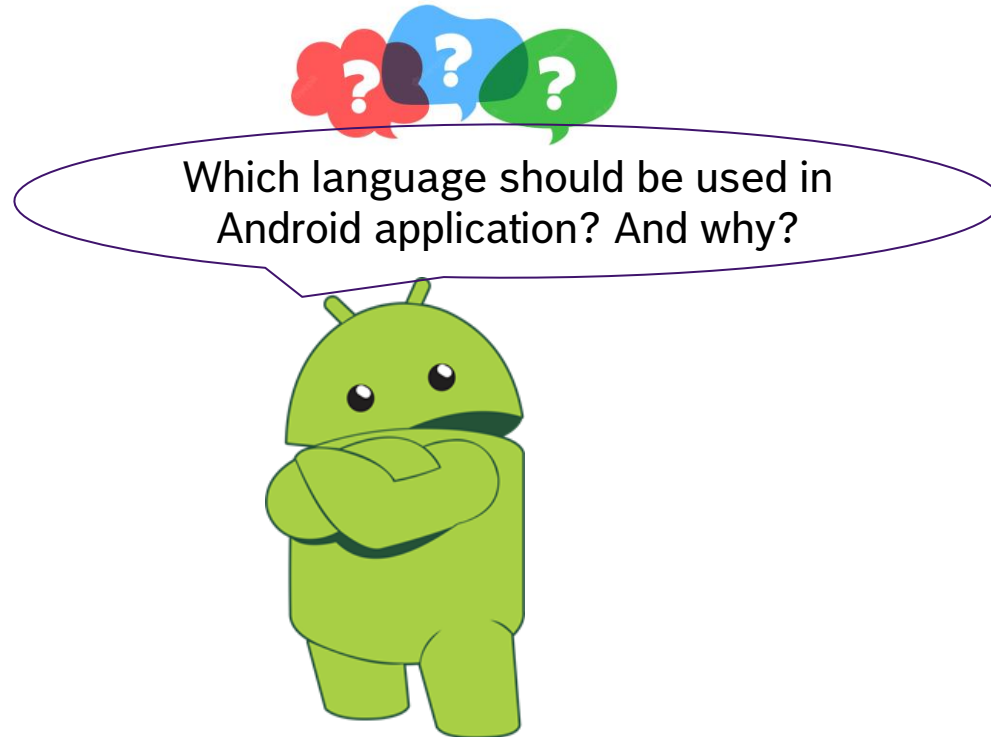
QUIZ (2/10)



QUIZ (3/10)



QUIZ (4/10)



QUIZ (5/10)



Why did Google recommend Kotlin to App developers?



QUIZ (6/10)



Which strategy did Google use to promote Android?



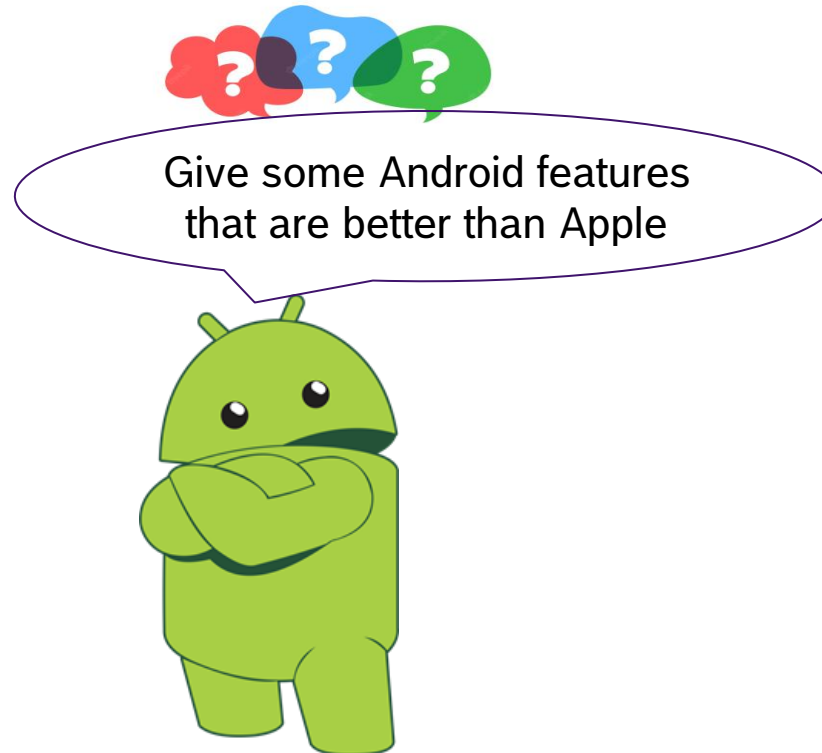
QUIZ (7/10)



Please give some benefits to
learn/develop Android platform?



QUIZ (8/10)



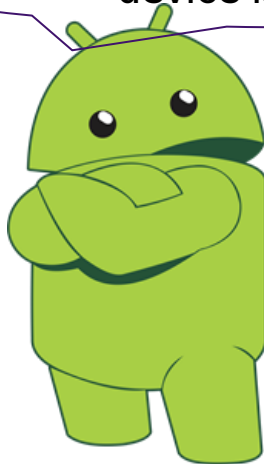
QUIZ (9/10)



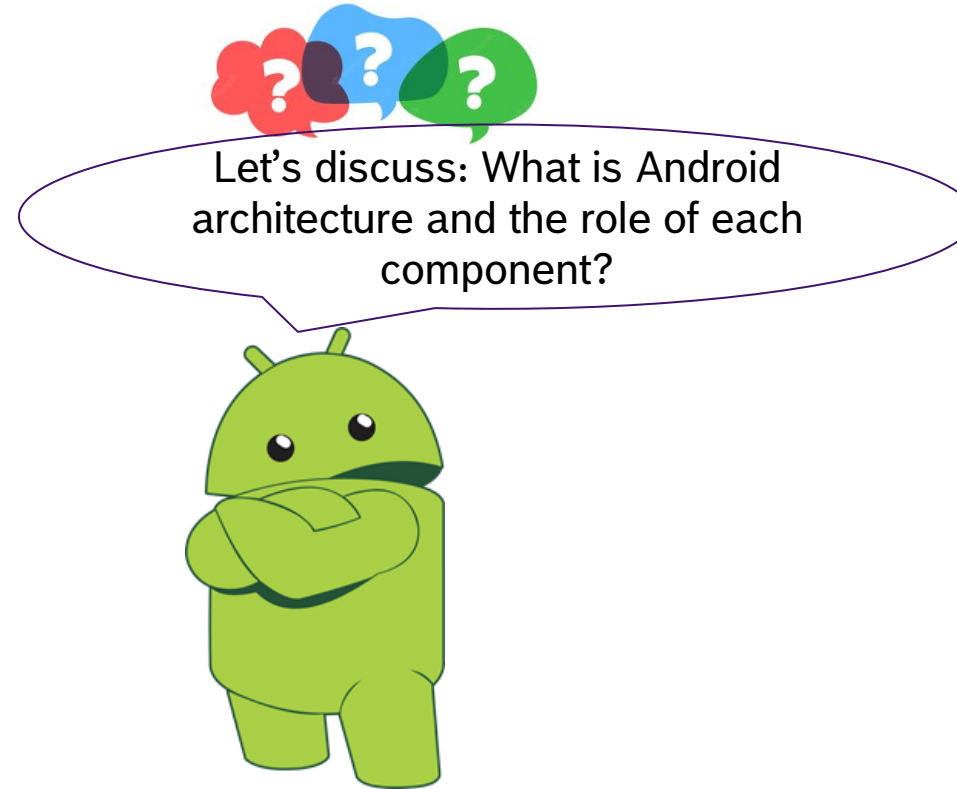
QUIZ (10/10)



What is API level? What will happen if an app having greater API level than device is installed?



Group working



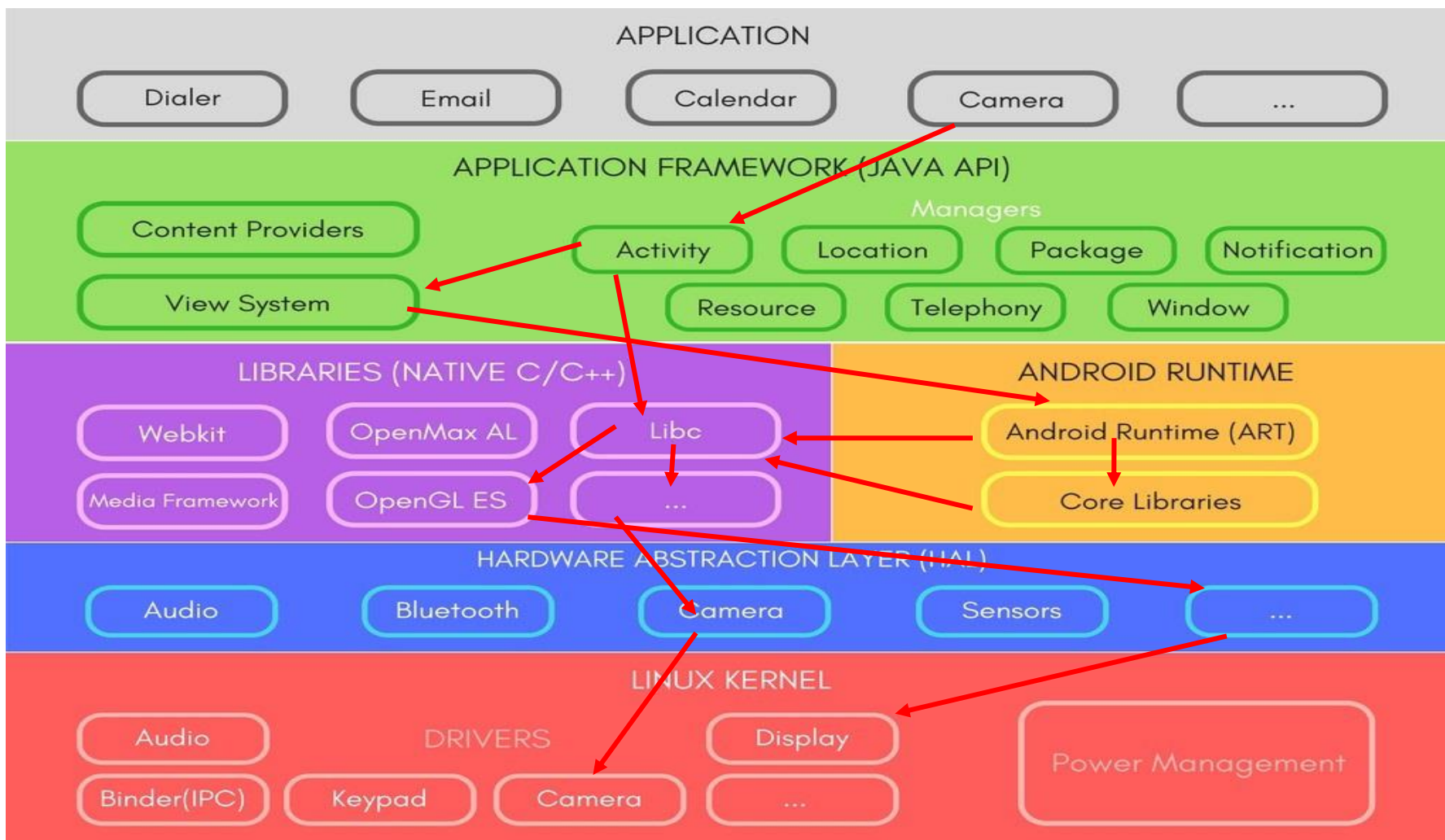
Agenda

- **Architecture**
 - **Overview**
 - **Application**
 - **Application Framework**
 - **Libraries and Android Runtime**
 - **HAL**
 - **Kernel**

Agenda

- **Architecture**
 - **Overview**
 - Application
 - Application Framework
 - Libraries and Android Runtime
 - HAL
 - Kernel

Overview



Agenda

- **Architecture**

- Overview
- **Application**
- Application Framework
- Libraries and Android Runtime
- HAL
- Kernel

Application

- Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, etc.
- Apps are included with the platform, but third-party apps can become the user's default.
- This is the place where an application is installed and used.
- Android app developers will develop their application at this layer.



Application

- There are 2 programming languages for Android developers: Java and Kotlin.
- However, Kotlin is recommended for Android app development since May 7th, 2019 by Google.



Application

- Every Android application runs in its own process. Whenever there's a request that should be handled by a particular component, Android will:
 - make sure that the application process of the component is running.
 - starting it if necessary.
 - if an appropriate instance of the component is available, launch it.



Application

- A Linux process encapsulating an Android application is created for the application when some of its code needs to be run, and will remain running until:
 - it is no longer needed, OR
 - the system needs to reclaim its memory for use by other applications.



Application

- An unusual and fundamental feature of Android is that an application process's lifetime is not directly controlled by the application itself. Instead, it is determined by the system through a combination of:
 - the parts of the application that the system knows are running.
 - how important these things are to the user, and
 - how much overall memory is available in the system.

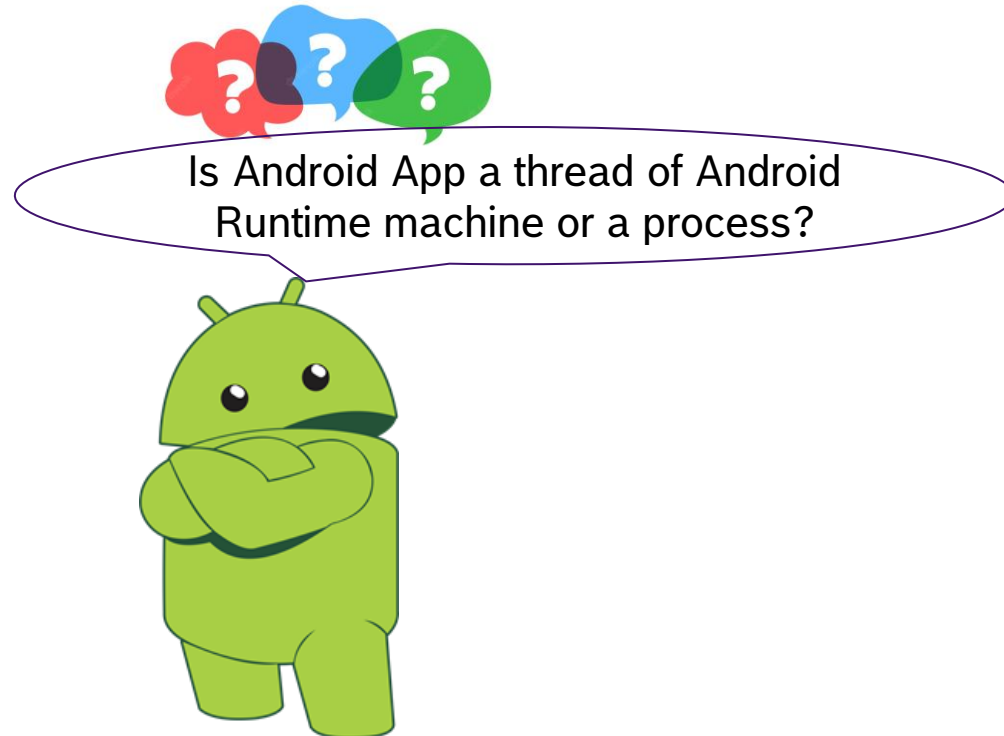


Day 2

Group working



QUIZ (1/5)



QUIZ (2/5)



QUIZ (3/5)



When “Back” button is pressed,
did Android app exit? Why?



QUIZ (4/5)



What will happen if Android
is out of memory?



QUIZ (5/5)



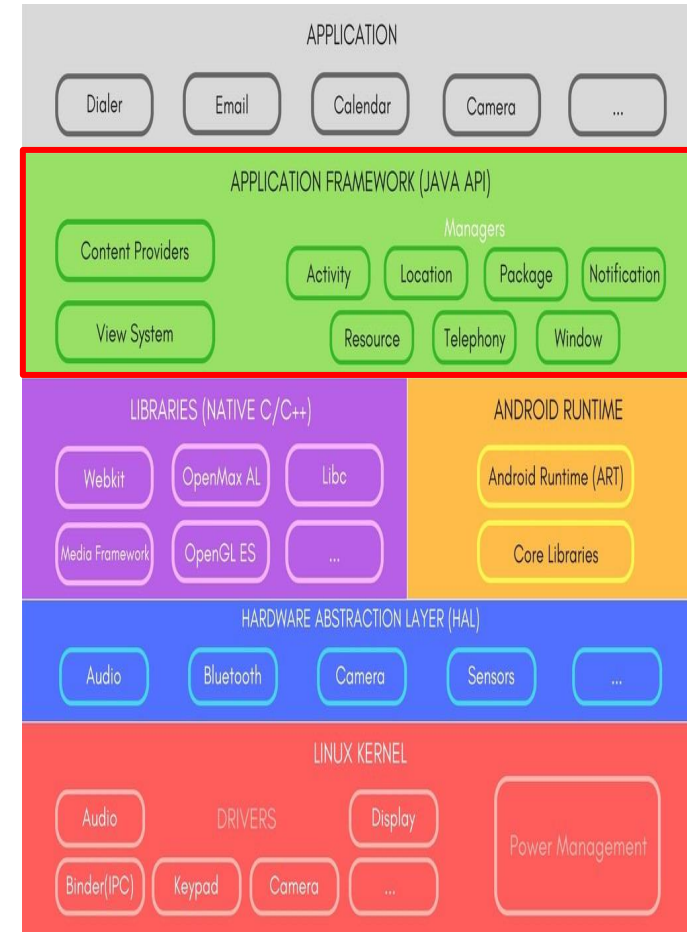
Agenda

- **Architecture**

- Overview
- Application
- **Application Framework**
- Libraries and Android Runtime
- HAL
- Kernel

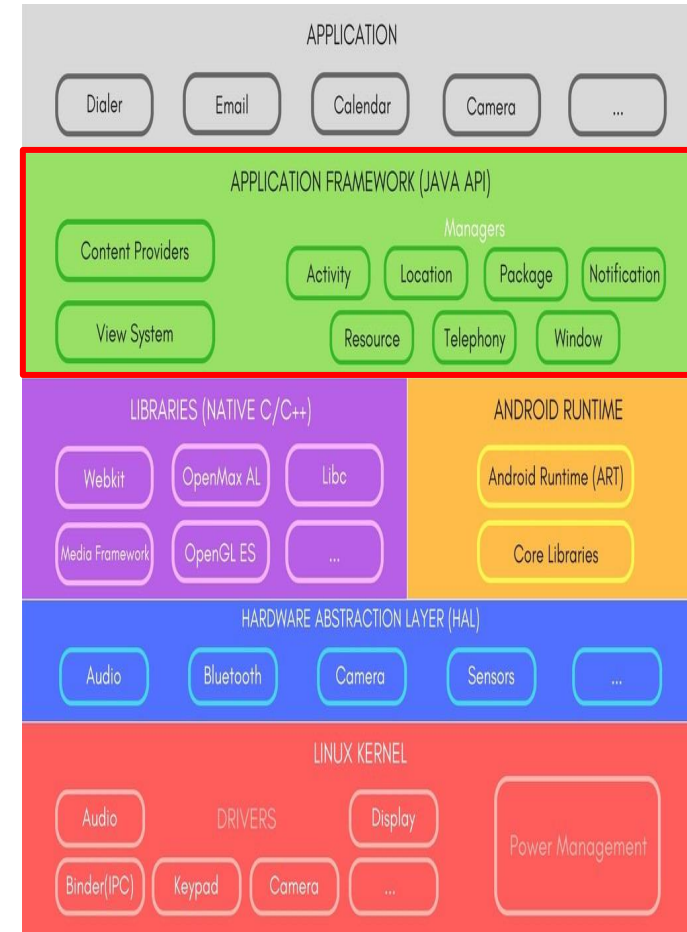
Application Framework

- The Application Framework layer provides many higher-level services to applications in the form of Java classes.
- Application developers are allowed to make use of these services in their applications.



Application Framework

- Through the use of Java package naming, Android frameworks are divided into separate namespaces, according to their functionality.
- Packages in the android.* namespace are available for use by developers.
- Packages in com.android.* are internal.
- Android also supports most of the standard Java runtime packages in the java.* namespace.



Application Framework

Package Name	API	Contents
android.app	1	Application Support
android.content		Content providers
android.database		Database support, mostly SQLite
android.graphics		Graphics support
android.opengl		OpenGL Graphics support
android.hardware		Camera, input and sensor support
android.location		Location support
android.media		Media support
android.net		Network support built over java.net APIs
android.os		Core OS Service and IPC support
android.provider		Built-in Android content-providers
android.sax		SAX XML Parsers
android.telephony		Core Telephony support
android.text		Text rendering
android.view		UI Components (similar to iOS's UIView)
android.webkit		Webkit browser controls
android.widget		Application widgets
android.speech	3	Speech recognition and Speech-to-Text
android.accounts	4	Support for account management and authentication.
android.gesture		Custom gesture support
android.accounts	5	User account support
android.bluetooth		Bluetooth support
android.media.audiofx	9 (G)	Audio Effects support
android.net.sip		Support for VoIP using the Session Initiation Protocol (RFC3261)
android.os.storage		Support for Opaque Binary Blobs (OBB)



Application Framework

android.nfc		Support for Near Field Communication
android.animation	11 (H)	Animation of views and objects
android.drm		Digital Rights Management and copy protection
android.renderscript		RenderScript (OpenCL like computation language)
android.hardware.usb	12	USB Peripheral support
android.mtp		MTP/PTP support for connected cameras, etc
android.net.rtp		Support for the Real-Time-Protocol (RFC3501)
android.media.effect	14 (I)	Image and Video Effects support
android.net.wifi.p2p		Support for Wi-Fi Direct (Peer-To-Peer)
android.security		Support for keychains and keystores
android.net.nsd	16 (J)	Neighbor-Service-Discovery through Multicast DNS (Bonjour)
android.hardware.input		Input device listeners
android.hardware.display	17	External and virtual display support
android.service.dreams		"Dream" (screensaver) support
android.graphics.pdf	19 (K)	PDF Rendering
android.print[.pdf]		Support for external printing
android.app.job	21 (L)	Job scheduler
android.bluetooth.le		Bluetooth Low-Energy (LE) support
android.hardware.camera2		The new camera APIs
android.media.[browse/projection/session/tv]		Media browsing and TV support
android.service.voice		Activation by "hot words" (e.g. "OK Google")
android.system		uname(), poll(2) and fstat[vfs] (2)
android.service.carrier	22	SMS/MMS support (CarrierMessagingService)
android.hardware.fingerprint	23 (M)	Fingerprint sensor support
android.security.keystore		Cryptographic key generation & storage
android.service.choser		Application deep linking



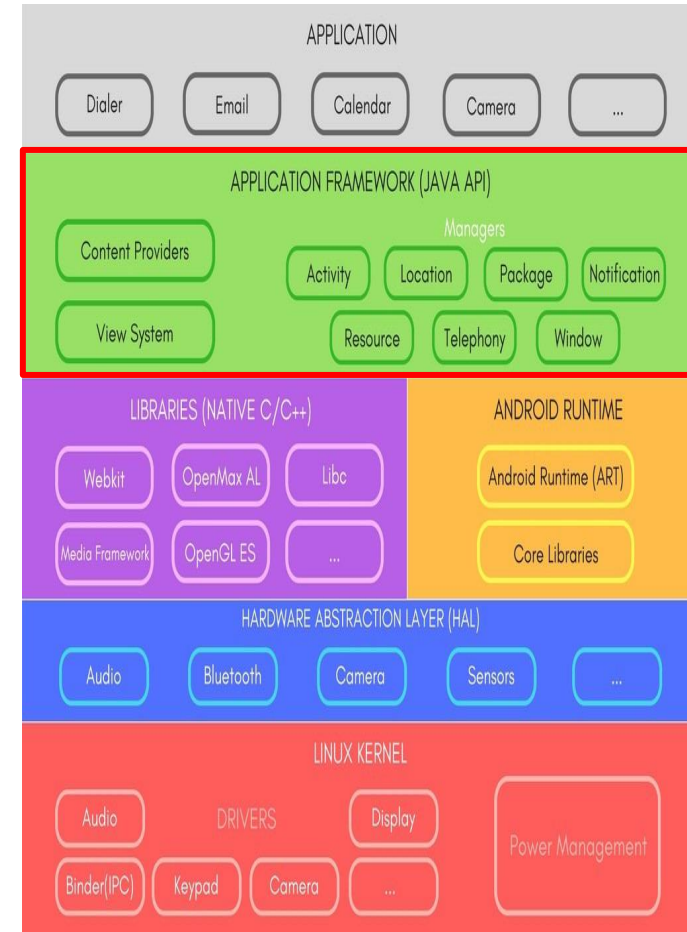
Application Framework

- There are 5 key services in Android framework:
 - Activity managers.
 - Content providers.
 - Resource managers.
 - Notification managers.
 - View system.



Application Framework

- There are 5 key services in Android framework:
 - Activity managers.
 - Content providers.
 - Resource managers.
 - Notification managers.
 - View system.



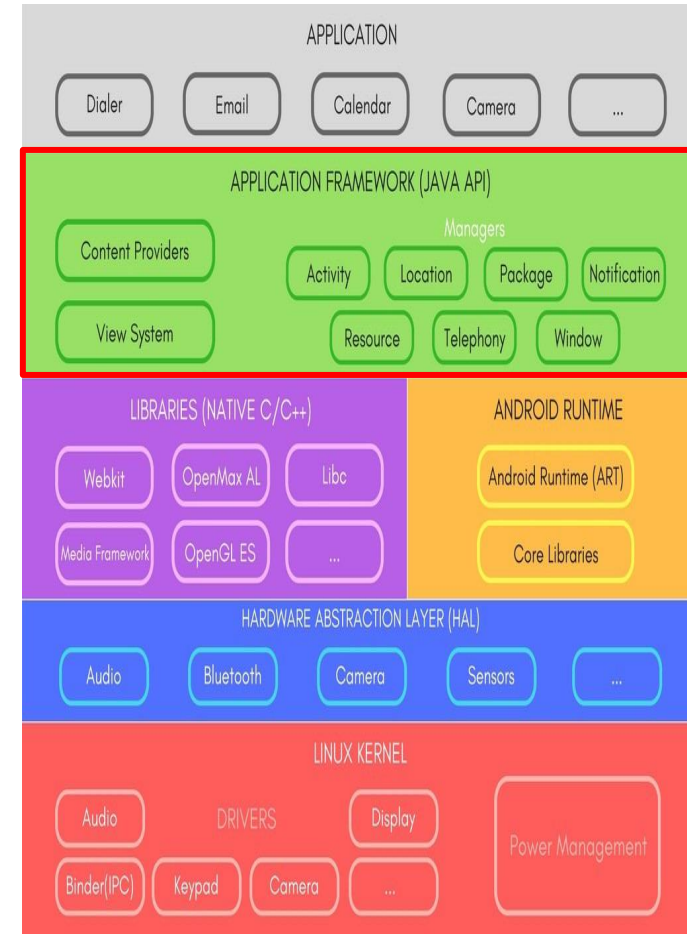
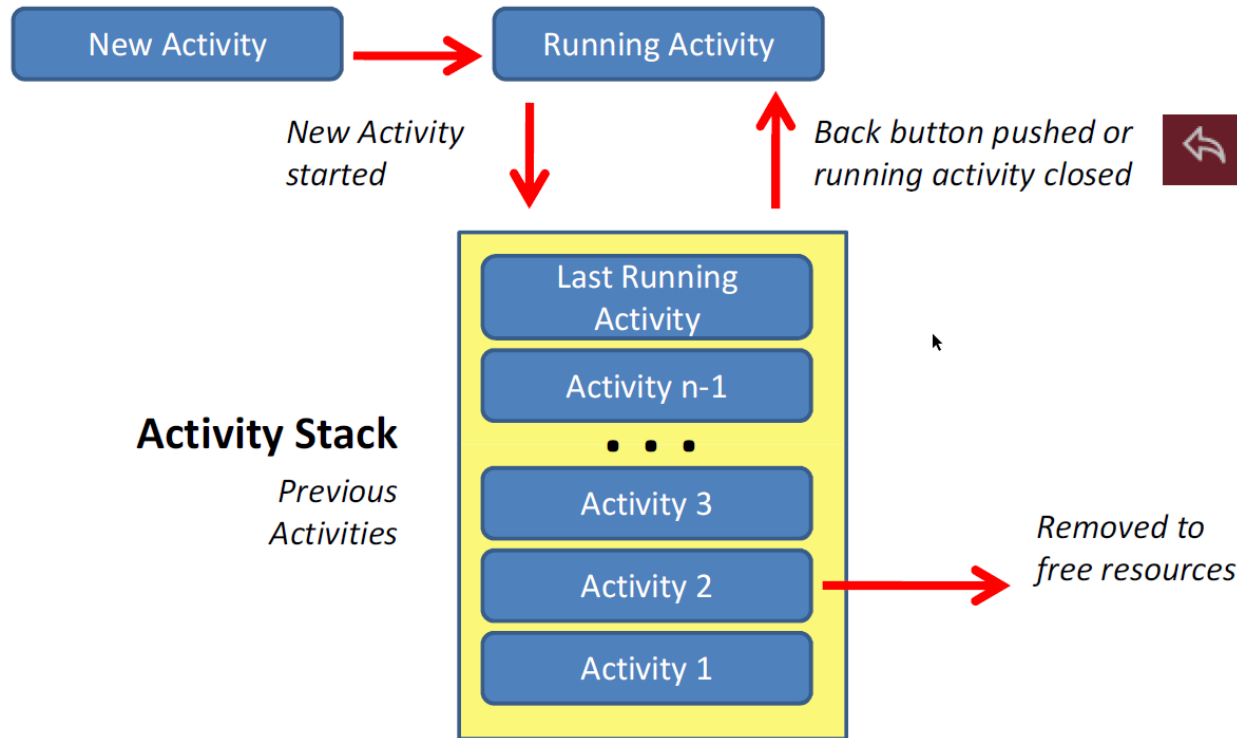
Application Framework

- Activity managers:
 - Controls all aspects of the application lifecycle and activity stack.
 - Activity stack is the one managing all activities in the system.
 - When a new activity is started , it is placed on the top of the stack and becomes the running activity -- the previous activity always remains below it in the stack, and will not come to the foreground again until the new activity exits.
 - If the user presses the Back Button the next activity on the stack moves up and becomes active.



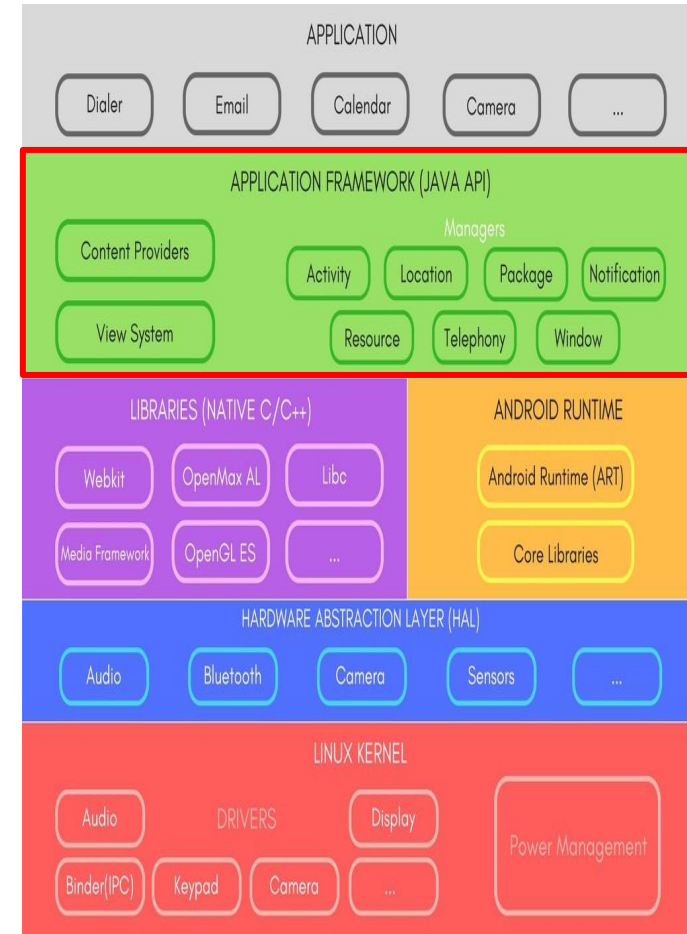
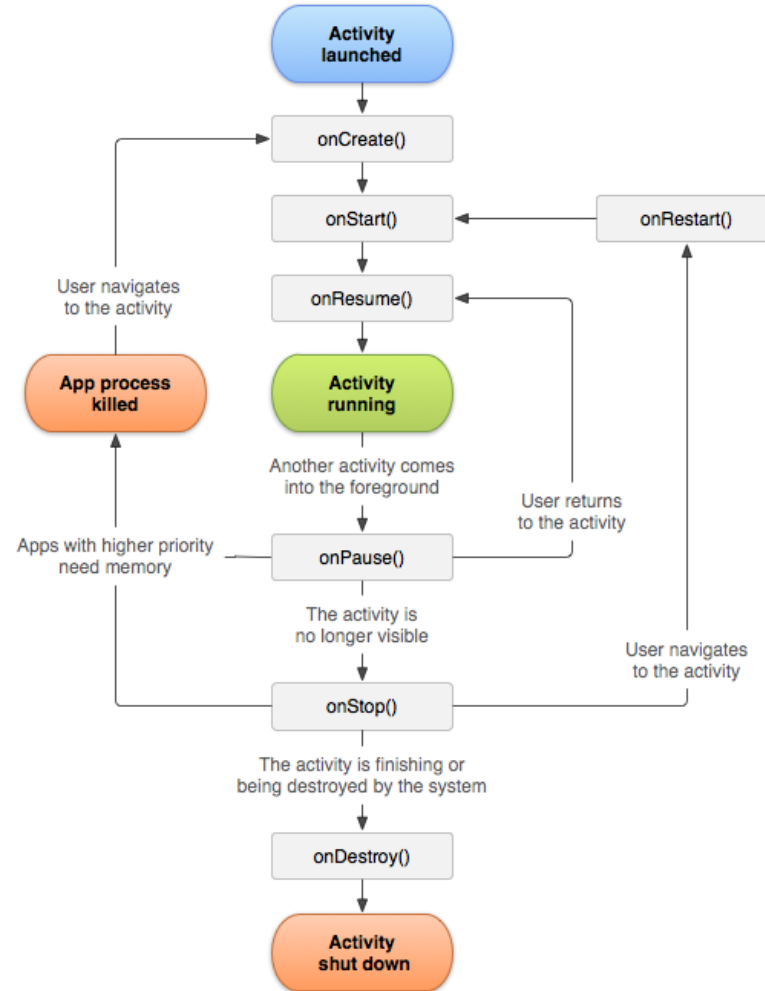
Application Framework

- Activity managers:



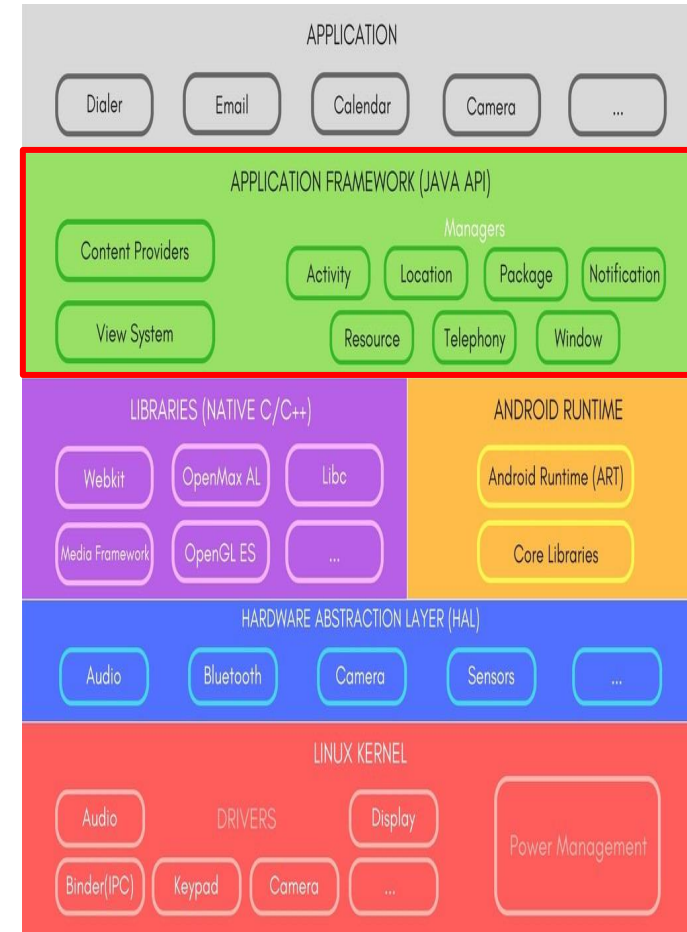
Application Framework

- Activity managers:



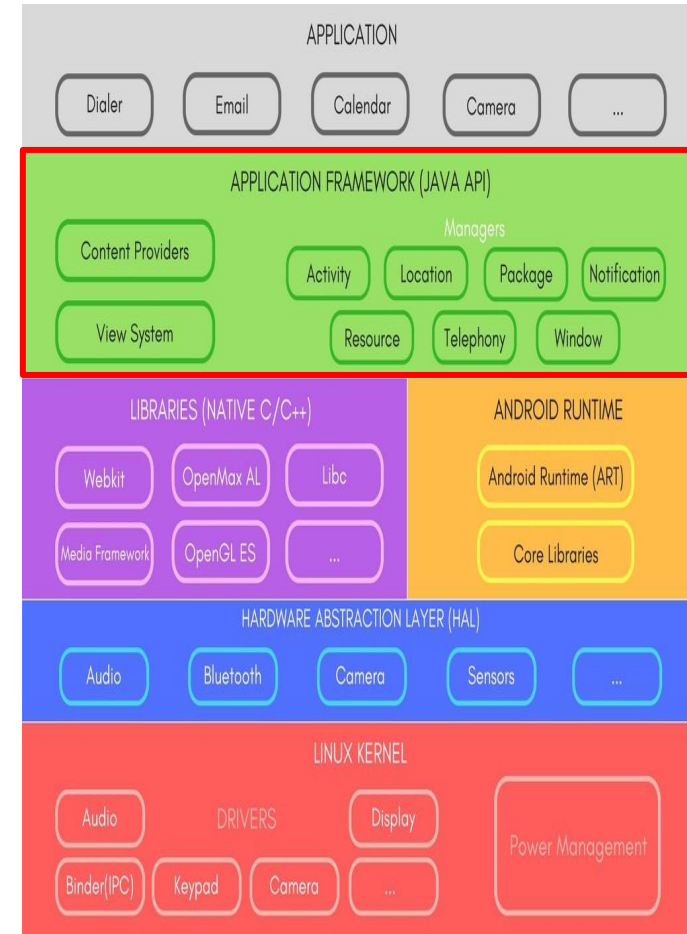
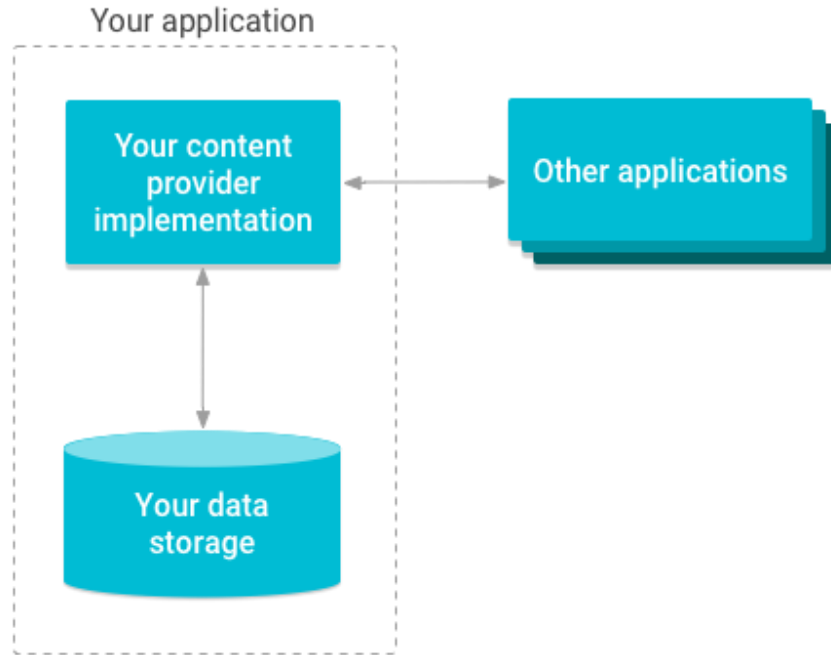
Application Framework

- There are 5 key services in Android framework:
 - Activity managers.
 - Content providers.
 - Resource managers.
 - Notification managers.
 - View system.



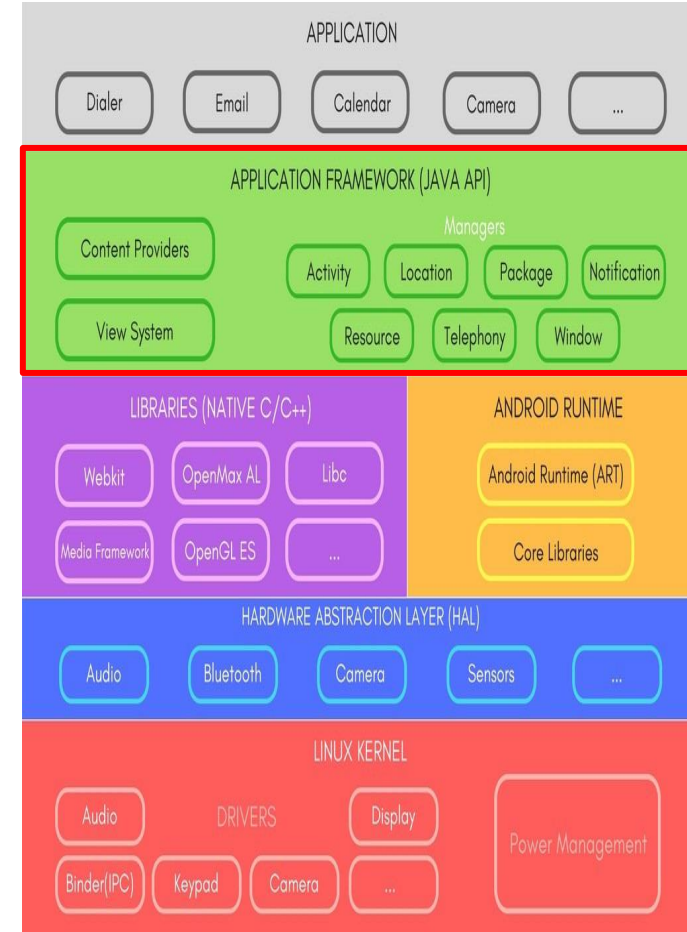
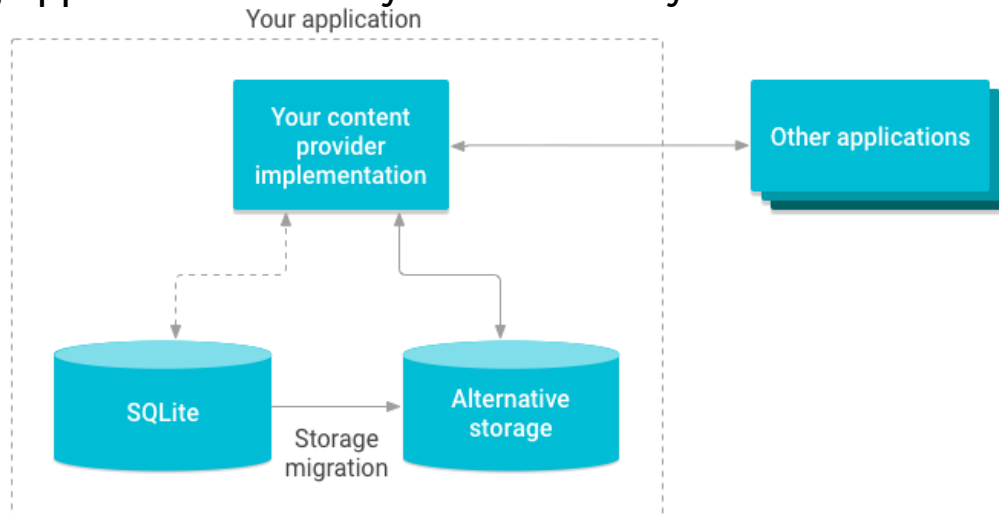
Application Framework

- Content Providers:
 - Allows applications to publish and share data with other applications (eg: Contacts app, etc.)



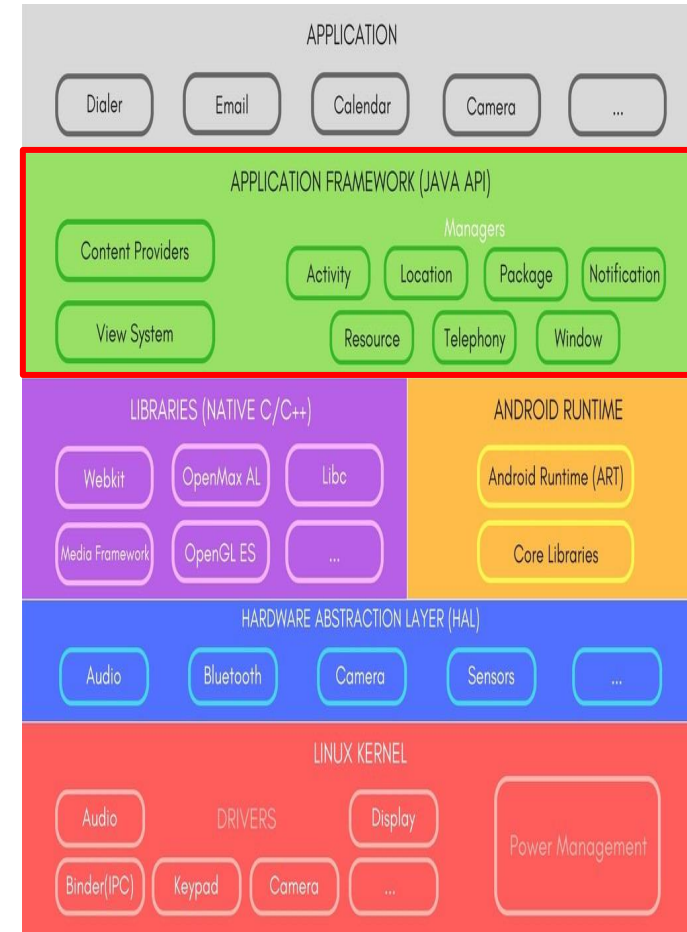
Application Framework

- Content Providers:
 - Use content providers if you plan to share data. If you don't plan to share data, you may still use them because they provide a nice abstraction, but you don't have to.
 - This will ensure that your application data is modified without affecting other existing applications that rely on access to your data.



Application Framework

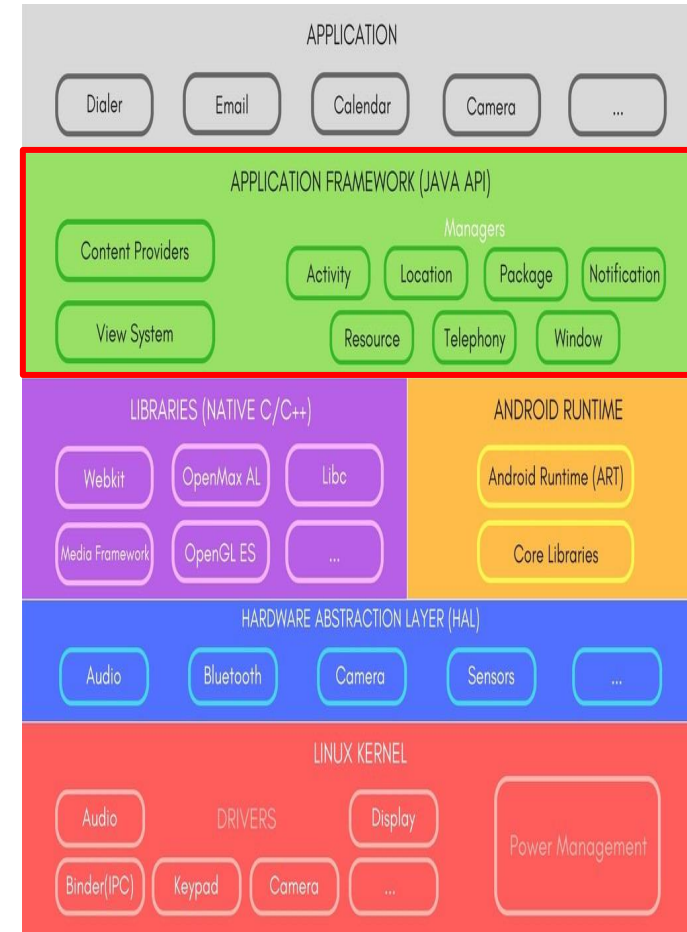
- There are 5 key services in Android framework:
 - Activity managers.
 - Content providers.
 - Resource managers.
 - Notification managers.
 - View system.



Application Framework

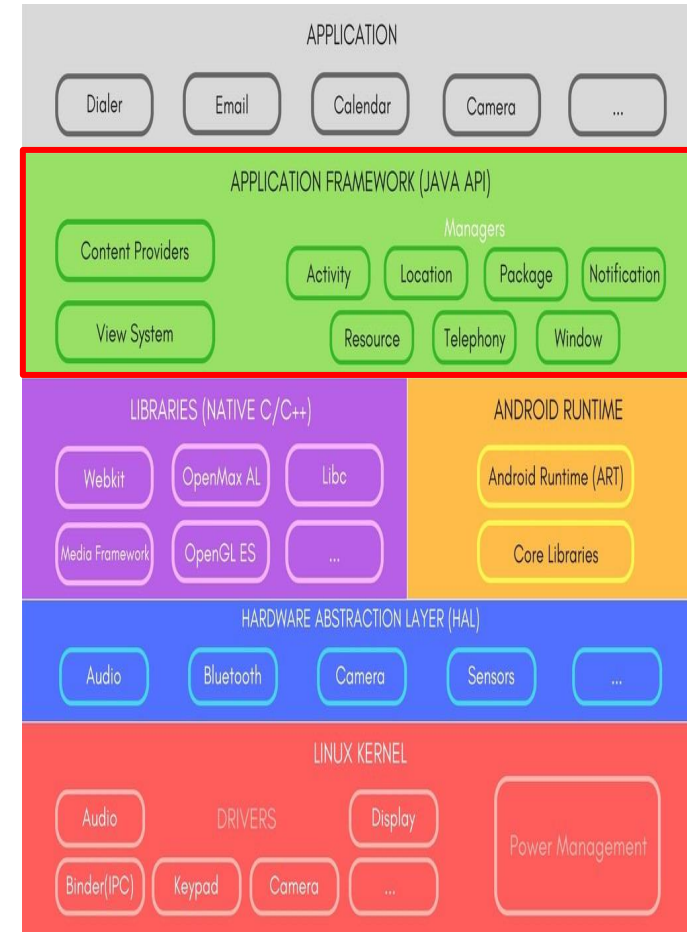
- Resource Manager:
 - Provides access to non-code embedded resources such as strings, color settings, user interface layouts, etc.
 - Each type of resource is placed in a specific subdirectory of project's res/ directory.

```
MyProject/  
  src/  
    MyActivity.java  
  res/  
    drawable/  
      graphic.png  
    layout/  
      main.xml  
      info.xml  
    mipmap/  
      icon.png  
    values/  
      strings.xml
```



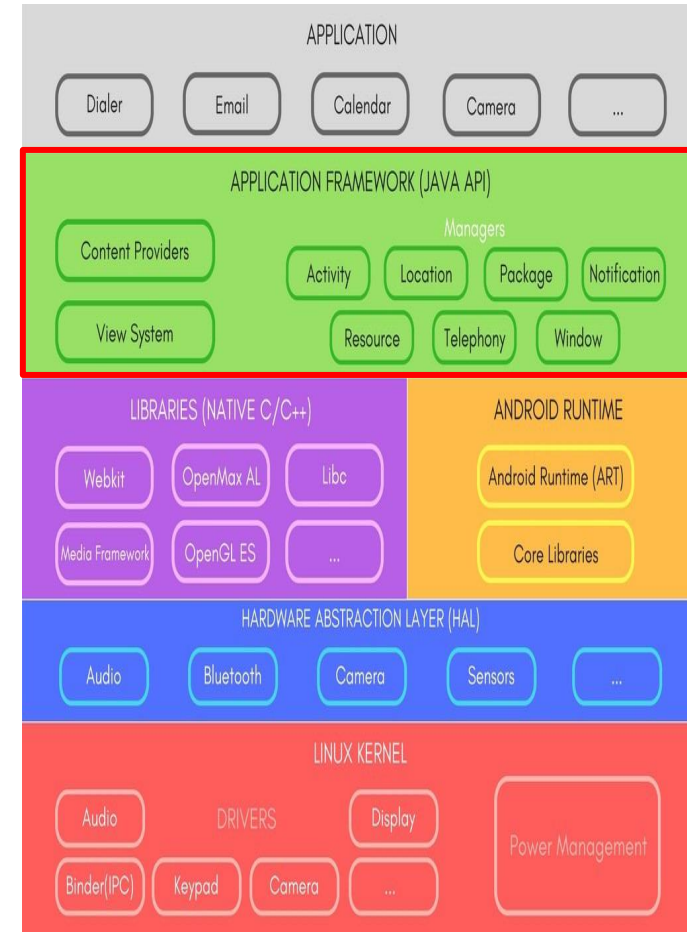
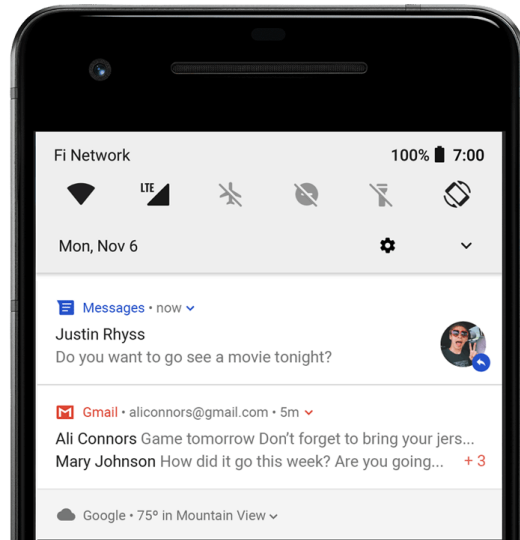
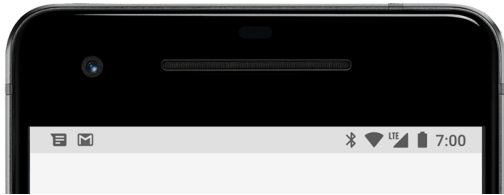
Application Framework

- There are 5 key services in Android framework:
 - Activity managers.
 - Content providers.
 - Resource managers.
 - Notification managers.
 - View system.



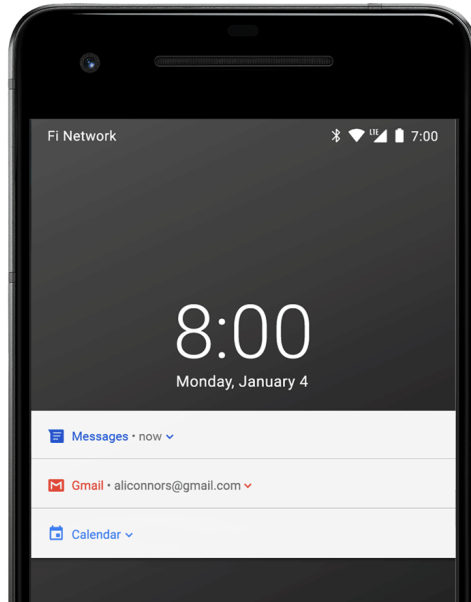
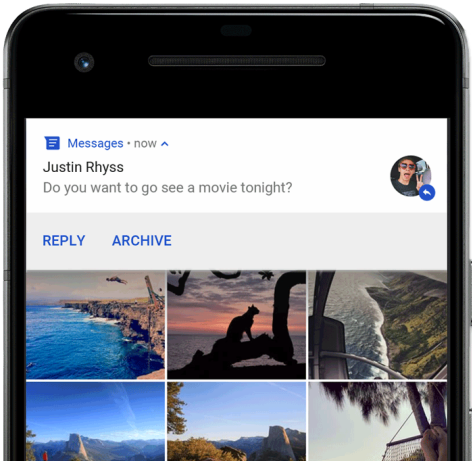
Application Framework

- Notifications Manager:
 - Allows applications to display alerts and notifications to the user.
 - Users can tap the notification to open your app or take an action directly from the notification.



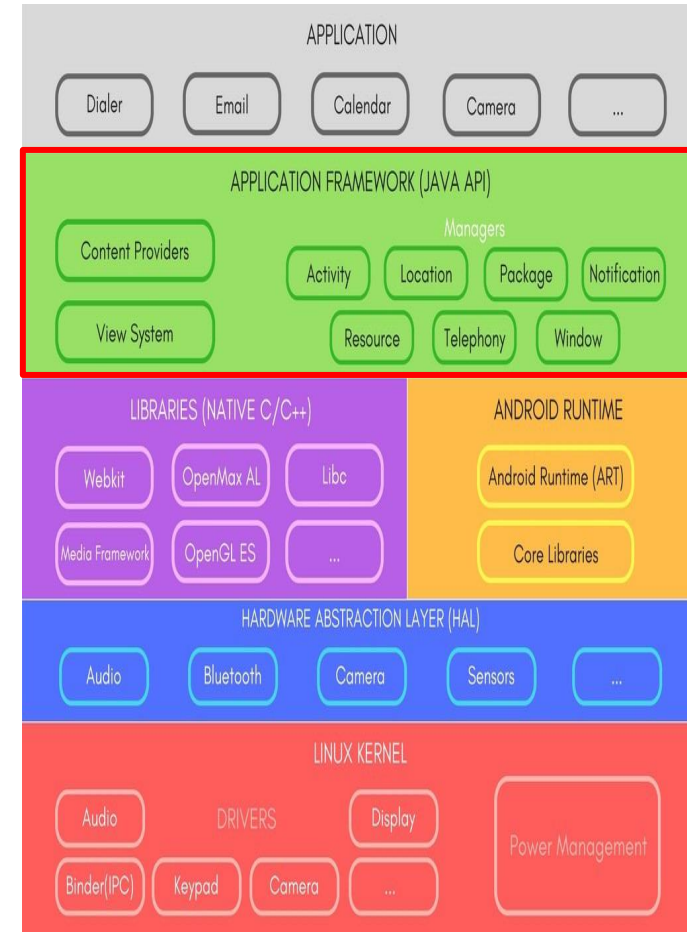
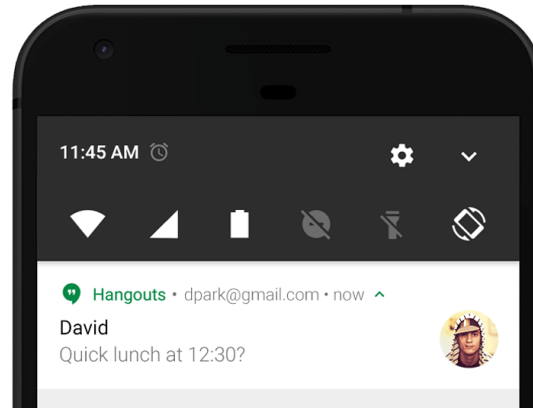
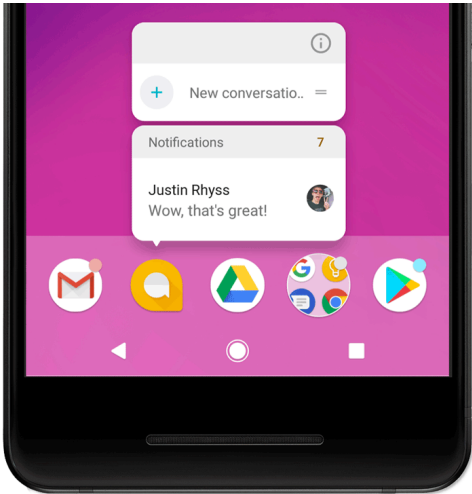
Application Framework

- Notifications Manager:
 - Allows applications to display alerts and notifications to the user.
 - Users can tap the notification to open your app or take an action directly from the notification.



Application Framework

- Notifications Manager:
 - Allows applications to display alerts and notifications to the user.
 - Users can tap the notification to open your app or take an action directly from the notification.



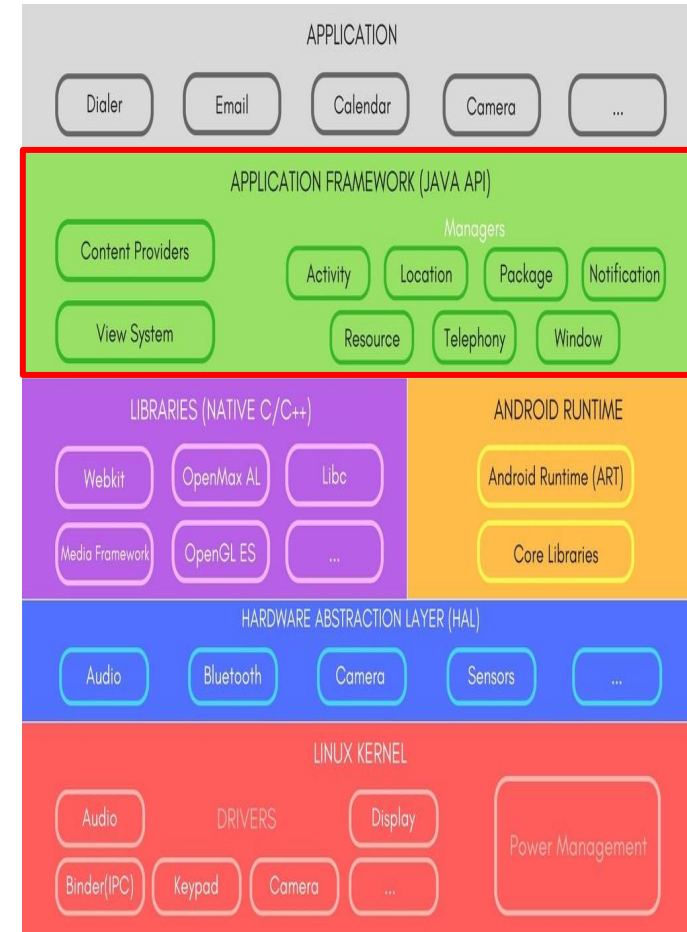
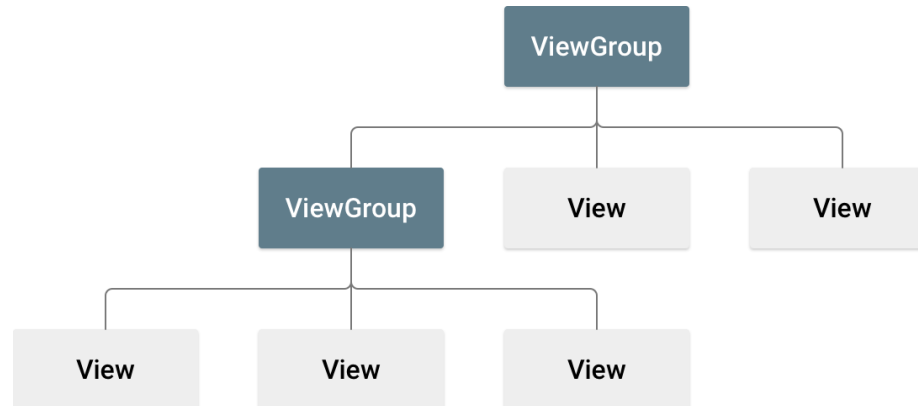
Application Framework

- There are 5 key services in Android framework:
 - Activity managers.
 - Content providers.
 - Resource managers.
 - Notification managers.
 - View system.



Application Framework

- View system:
 - An extensible set of views used to create application user interfaces.
 - All elements in the layout are built using a hierarchy of View and ViewGroup objects.
 - A View usually draws something the user can see and interact with.
 - A ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects.



Application Framework

- View system:
 - The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView.
 - The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure (eg: LinearLayout, ConstraintLayout).
 - Layout can be declared either in XML file or at runtime.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a TextView" />
    <Button android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hello, I am a Button" />
</LinearLayout>
```

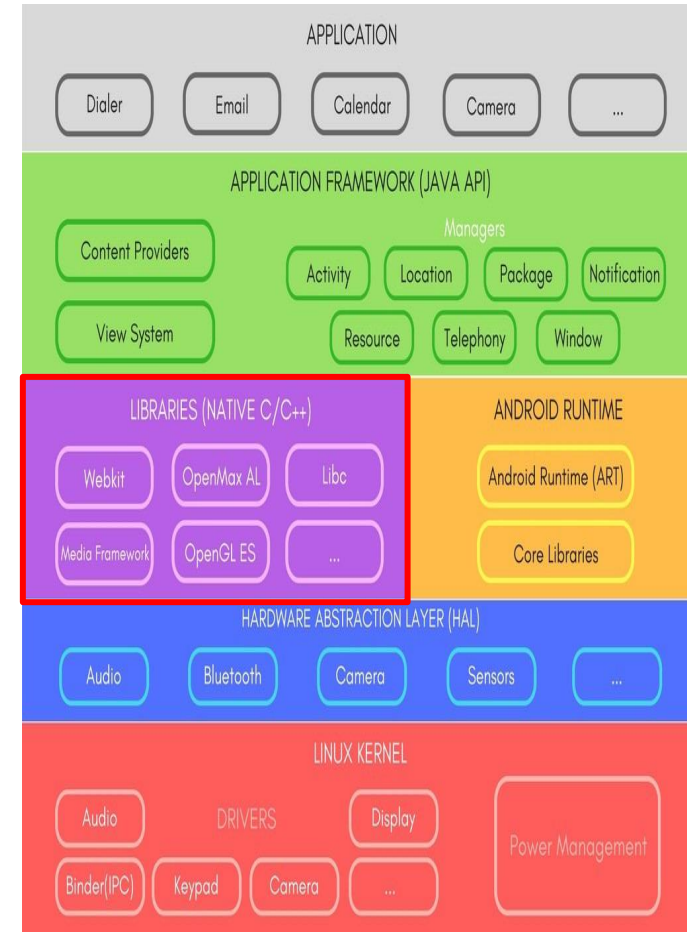


Agenda

- History
- Application field
- Market share
- **Architecture**
 - Overview
 - Application
 - Application Framework
 - **Libraries and Android Runtime**
 - HAL
 - Kernel

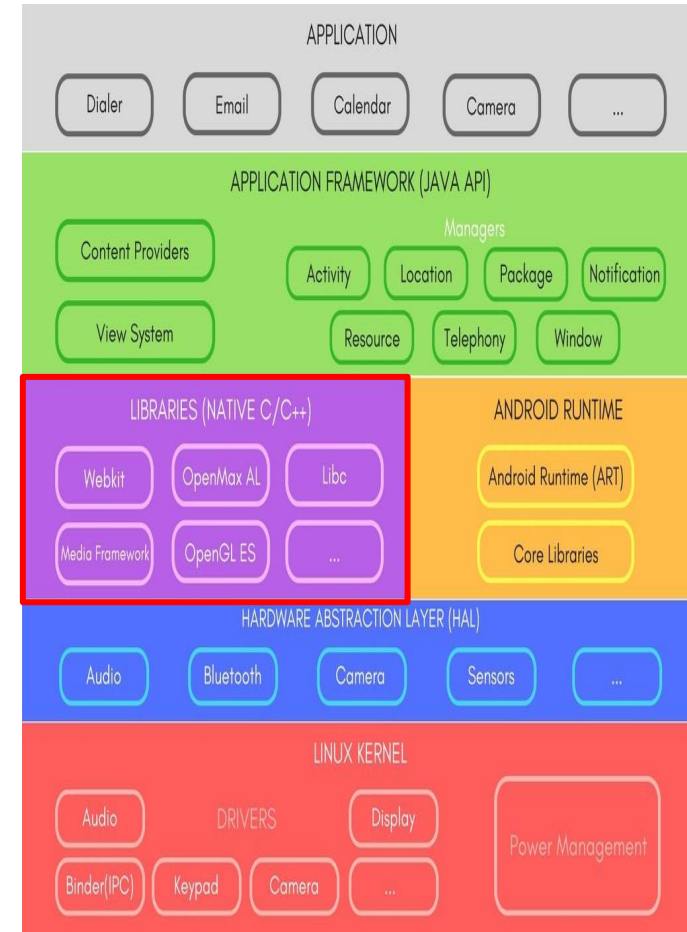
Libraries

- Gather a lot of Android specific libraries to interact at a low level with the system, but third parties libraries as well.
- Bionic is the C library, SurfaceManager is used for drawing surfaces on the screen, etc.
- The libraries mostly provide wrappers over kernel Androidisms, or implement additional functionality in user-mode.
- The functionality of these native libraries were exposed to Android application via Java framework APIs.



Libraries

- Binaries are usually located in `/system/bin`, and `/xbin` (with a few critical binaries located in `/sbin`).
- All of them are ELF binaries.
- Most binaries are usually the same across all devices, being part of the AOSP.



Android Runtime

- Handle the execution of Android applications.
- Almost entirely written from scratch by Google.
- Contain ART, the virtual machine that executes every application that you run on Android, and the core library for the Java runtime, coming from Apache Harmony Project.
- Also contain system daemons, init executable, basic binaries, etc.



Android Runtime

- Each app runs in its own process and with its own instance of the Android Runtime (ART).
- ART is written to run multiple virtual machines on low-memory devices by executing DEX files, a bytecode format designed specially for Android that's optimized for minimal memory footprint.



JNI

- A Java framework to call and be called by native applications written in other languages. Mostly used for:
 - Writing Java bindings to C/C++ libraries
 - Accessing platform specific features
 - Writing high performance sections
 - It is used extensively across the Android user space to interface between the Java Framework and the native daemons.
- Since Gingerbread, Java methods can be called through JNI.



Agenda

- **Architecture**

- Overview
- Application
- Application Framework
- Libraries and Android Runtime
- **HAL**
- Kernel

Hardware Abstraction Layer (HAL)

- Provide standard interfaces that expose device hardware capabilities to the higher-level Java API framework.
- Consist of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or bluetooth module.
- When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

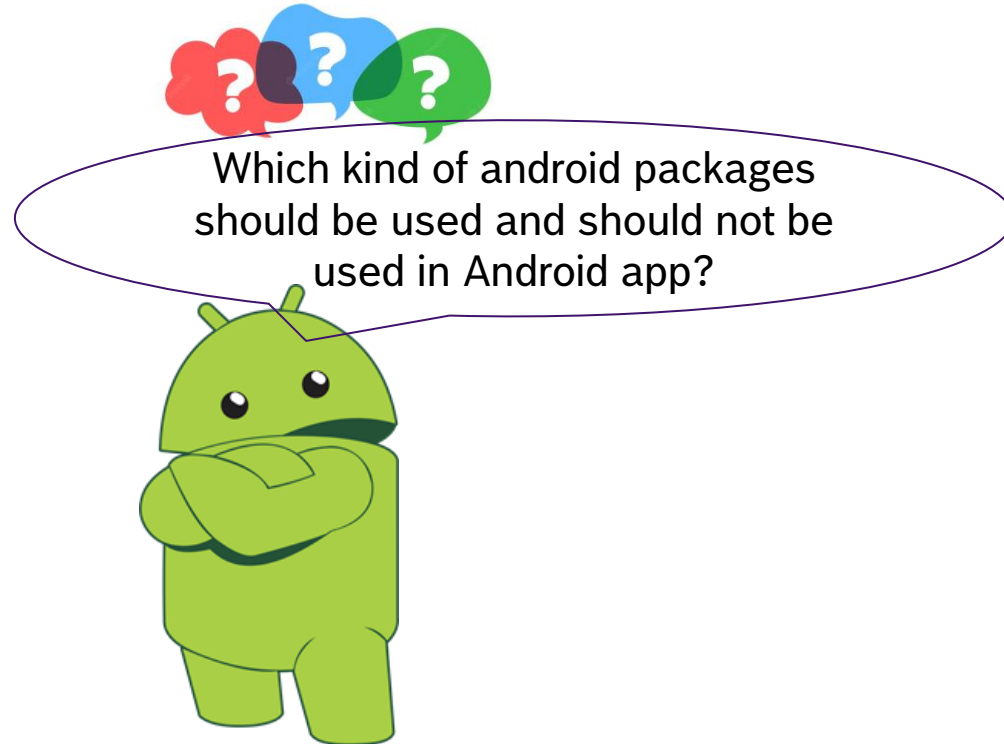


Day 3

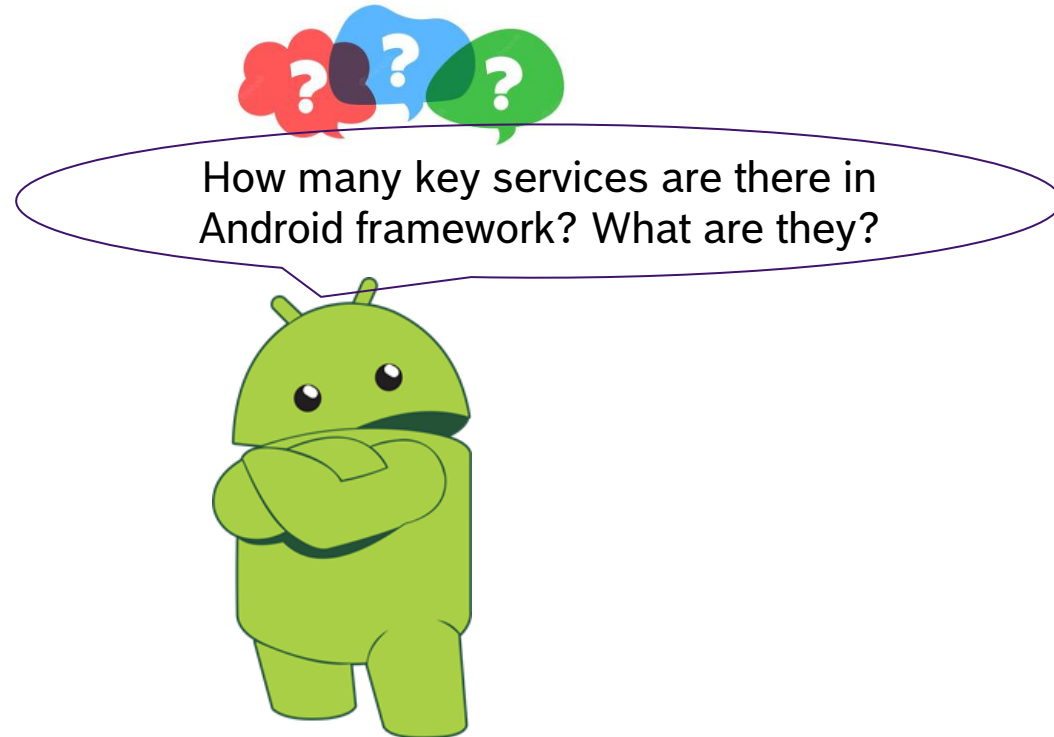
Group working



QUIZ (1/10)



QUIZ (2/10)



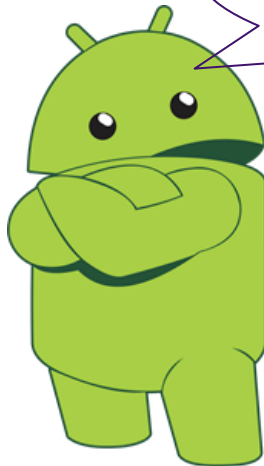
QUIZ (3/10)



What will happen if we press back continuously after this case:

- + Open app A (Activity A1->A2 ->A3)
 - + Open app B (Activity B1 -> B2 ->B3),
 - + Open app C (Activity C1->C2->C3->C4)
- Out of memory did not happen.

QUIZ (4/10)



What is the differences
between “**onCreate**”,
“**onStart**” and “**onResume**”?

QUIZ (5/10)



What is Content Provider?
Why do we need to use
Content Provider?

QUIZ (6/10)



What is view system in
Android?

QUIZ (7/10)



When do we declare
View/ViewGroup in XML file
and/or at runtime?

QUIZ (8/10)



Can a Linux console app run in Android? Why?

QUIZ (9/10)



Where are executable files located in Android? What's about library?

QUIZ (10/10)



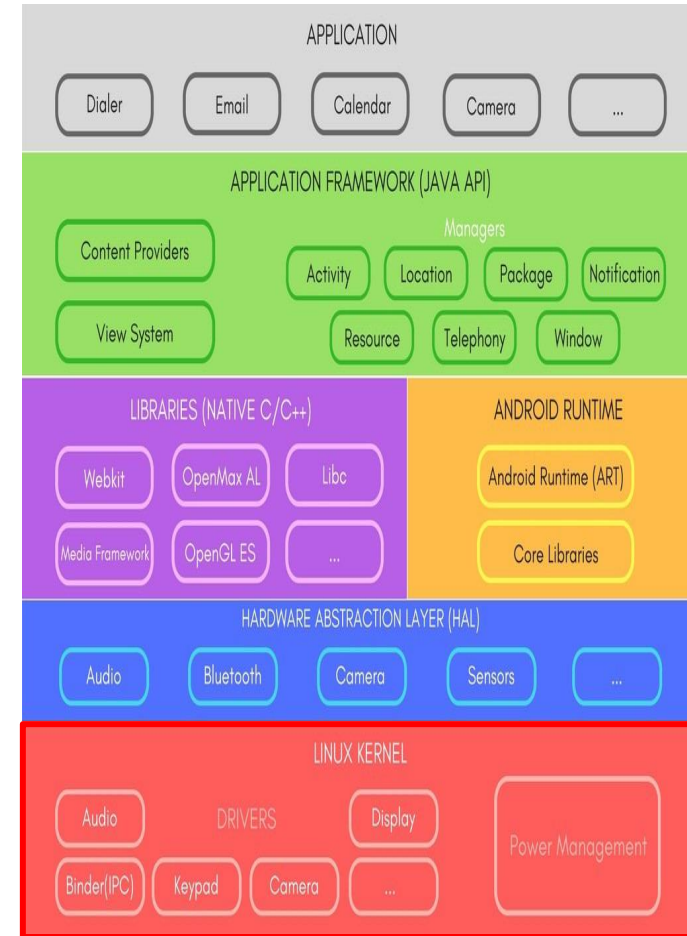
What is HAL? When should we **use** HAL? When should we **not use** HAL?

Agenda

- **Architecture**
 - Overview
 - Application
 - Application Framework
 - Libraries and Android Runtime
 - HAL
 - **Kernel**

Overview

- The foundation of the Android platform is the Linux kernel.
- For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.
- Using a Linux kernel allows Android to take advantage of key security features and allows device manufacturers to develop hardware drivers for a well-known kernel.



Wakelock

- Every CPU has a few states of power consumption, from being almost completely off, to working at full capacity.
- These different states are used by the Linux kernel to save power when the system is run.
- For example, when the lid is closed on a laptop, it goes into “suspend”, which is the most power conservative mode of a device, where almost nothing but the RAM is kept awake.



Wakelock

- This is a good strategy for a laptop, it is not necessarily good for mobile devices.
- For example, you don't want your music to be turned off when the screen is off.
- Android implemented a new mechanism called “wakelock”.
- The main idea is instead of letting the user decide when the devices need to go to sleep, the kernel decided itself.



Wakelock

- Android allows applications and kernel drivers to actively prevent the system from going to suspend, keeping it awake.
- This means that the applications and drivers must use the wakelock APIs:
 - Applications do so through the abstraction provided by the API.
 - Drivers must do it themselves, which prevents to directly submit them to the kernel.



Wakelock

- Kernel space APIs:

```
#include <linux/wakelock.h>
void wake_lock_init(struct wakelock *lock,
                    int type,
                    const char *name);
void wake_lock(struct wakelock *lock);
void wake_unlock(struct wakelock *lock);
void wake_lock_timeout(struct wakelock *lock, long timeout);
void wake_lock_destroy(struct wakelock *lock);
```

- User space APIs:

```
$ echo foobar > /sys/power/wake_lock
$ echo foobar > /sys/power/wake_unlock
```

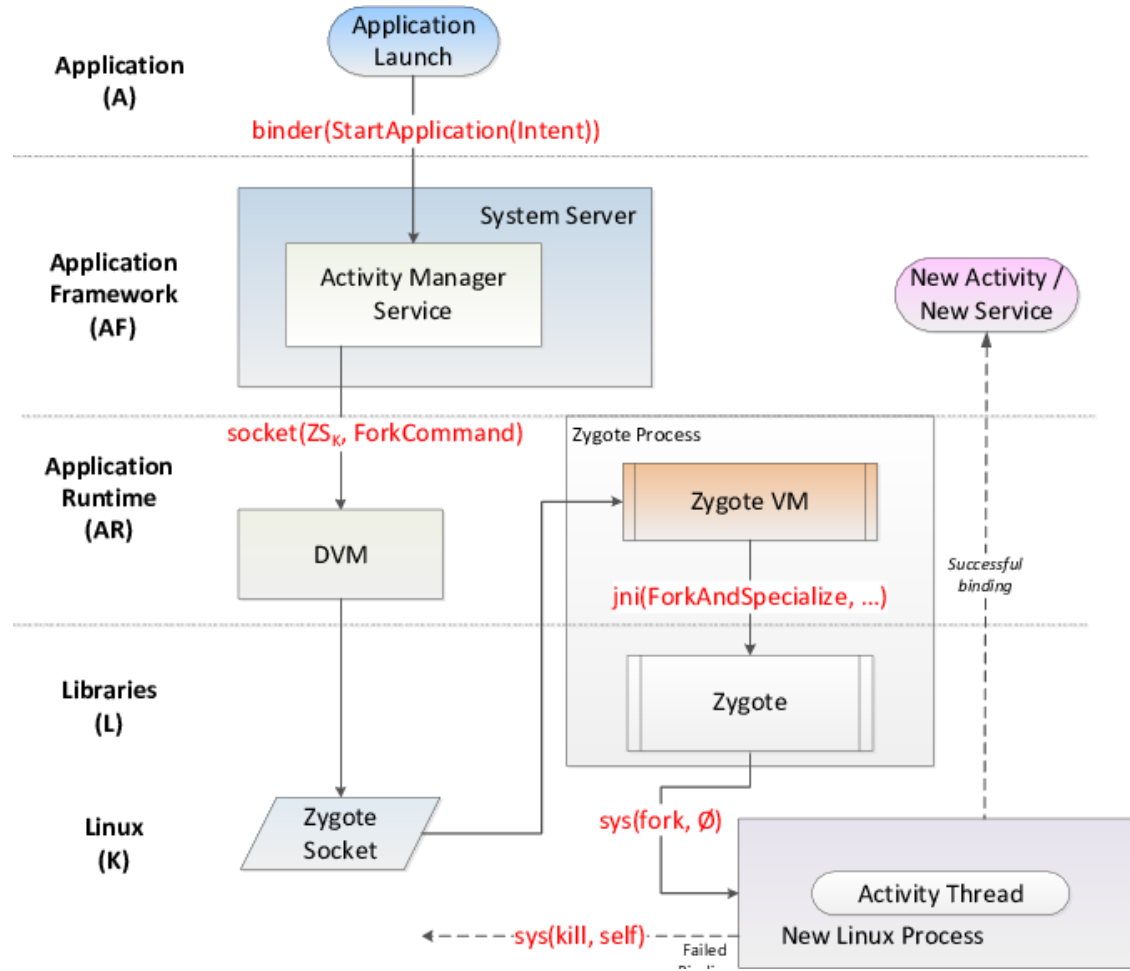


Binder

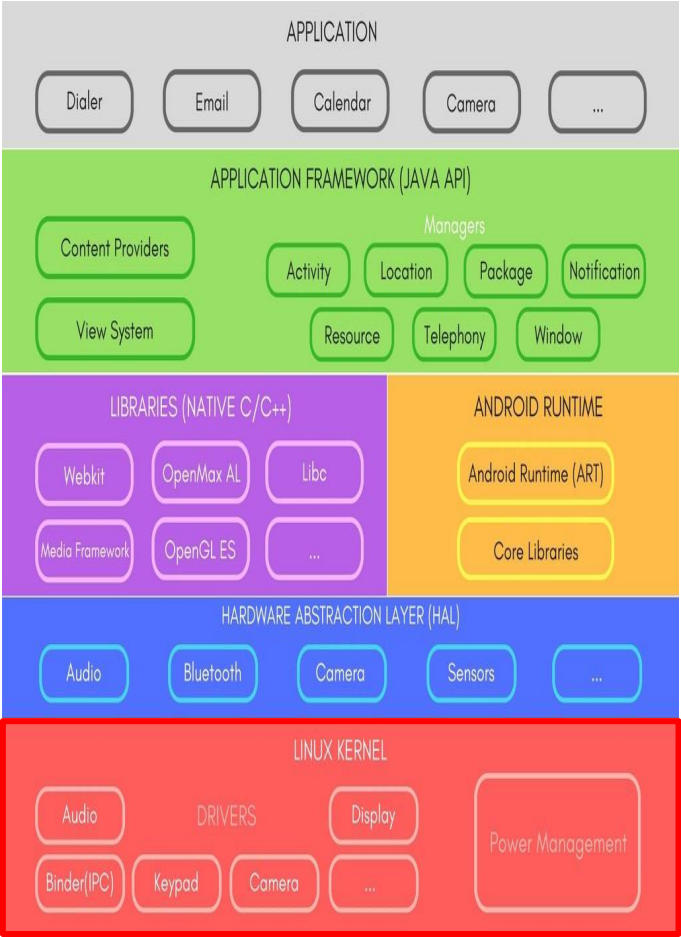
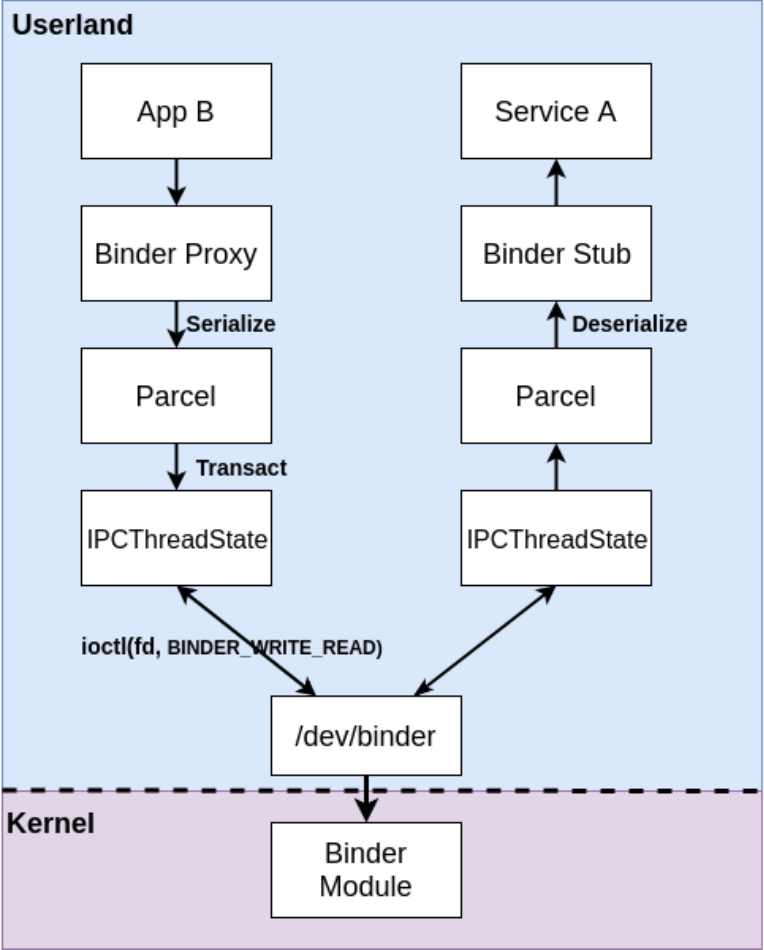
- An RPC/IPC mechanism
- One of the very basic functionalities of Android. Without it, Android cannot work.
- Every call to the system servers go through Binder, just like every communication between applications, and even communication between the components of a single application.



Binder



Binder



Logger

- In a regular Linux distribution, two components are involved in the system's logging:
 - Linux' internal mechanism: accessible with the **dmesg** command and holding the output of all the calls to **printk()** from various parts of the kernel.
 - A syslog daemon: handles the user space logs and usually stores them in the **/var/log** directory.



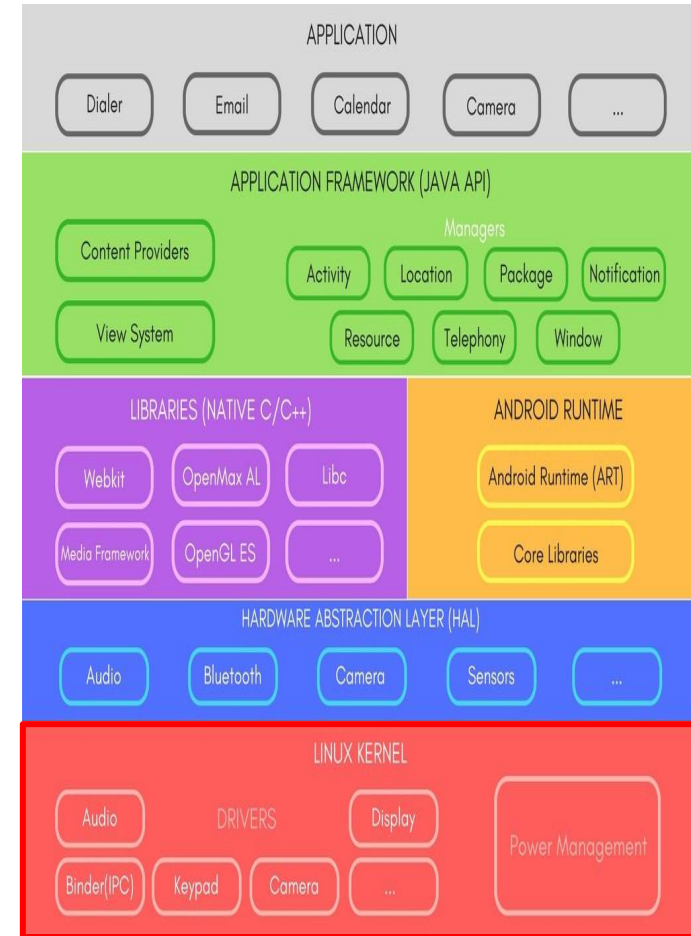
Logger

- From Android developers' point of view, this approach has two issues:
 - Generate expensive task switches as the calls to **syslog()** go through as socket.
 - Every call writes to a file, which probably writes to a slow storage device or to a storage device where writes are expensive.

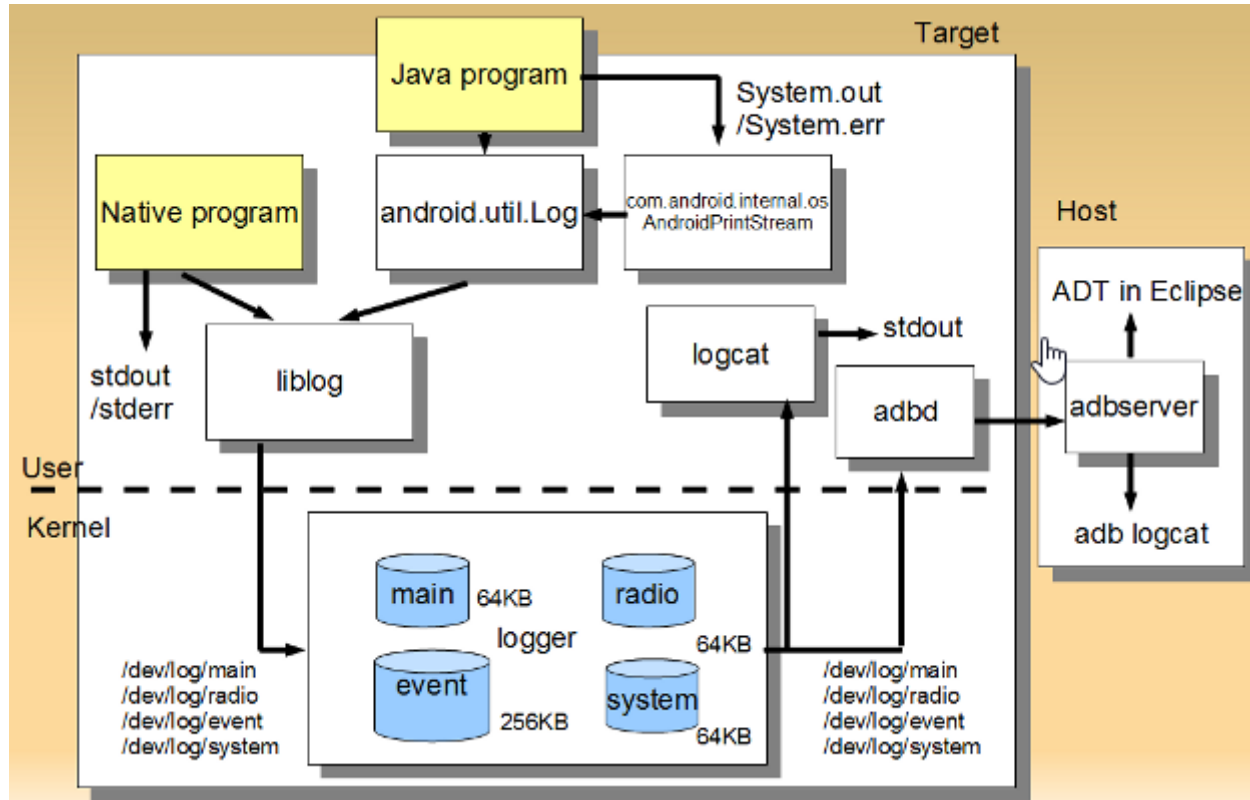


Logger

- Android logger will solve this issue.
- It is a kernel driver using 4 circular buffers in the kernel memory area.
- The buffers are exposed in the **/dev/log** directory and it can be accessed through the liblog library.
 - /dev/log/radio: radio related messages (64KB).
 - /dev/log/event: system/hardware events (256KB).
 - /dev/log/system: system/framework message (64KB).
 - /dev/log/main: everything else (64KB).

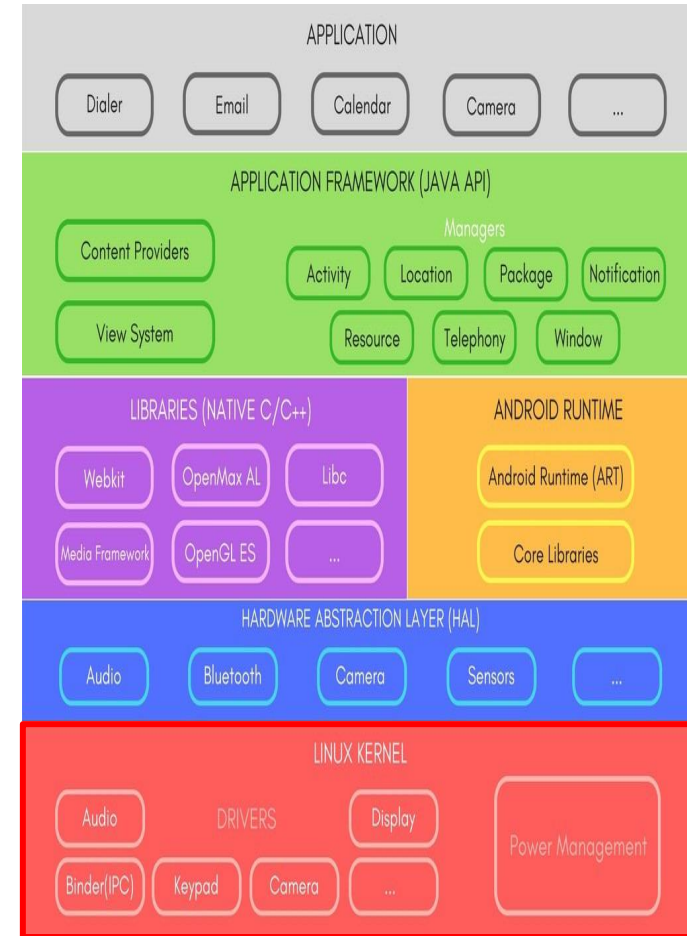


Logger



Logger

- File permission for each file is 0622:
 - Owner/group RW, others: W only
 - Owner = root; group = log
 - Anyone can write logs, root or log group can read them.
- Logger is used by the Android system and applications to write to logger, and by the logcat command to access them.
- This allows to have an extensive level of logging across the entire AOSP.



Day 4

Group working



Warming-up (1/5)

Shopping

Warming-up (2/5)

Đuổi hình bắt chữ



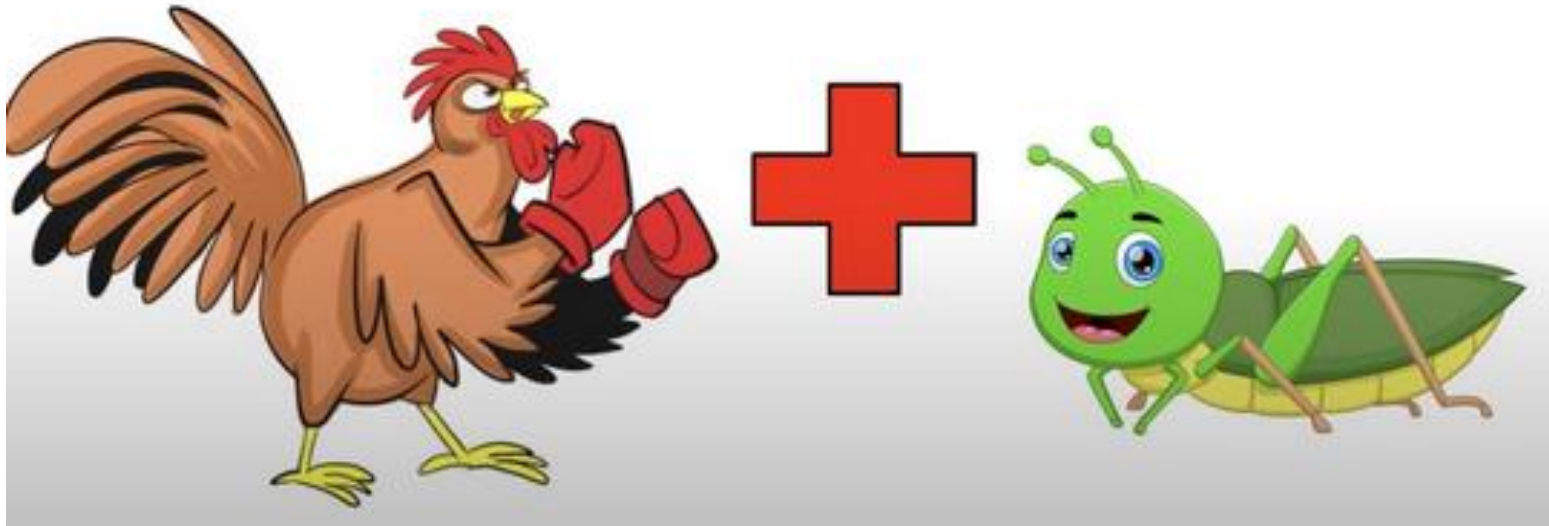
Đây là tỉnh thành nào?

Warming-up (3/5)



Warming-up (4/5)

ĐÂY LÀ TRÒ CHƠI GÌ



Warming-up (5/5)

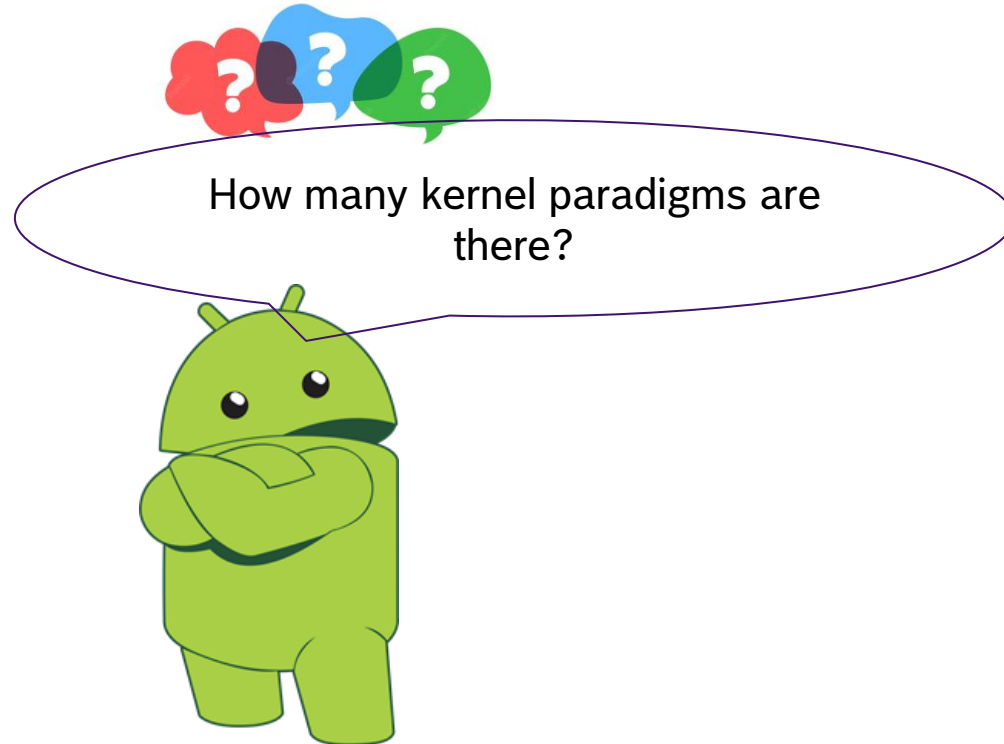
Đuổi hình bắt chữ

100
GRAM

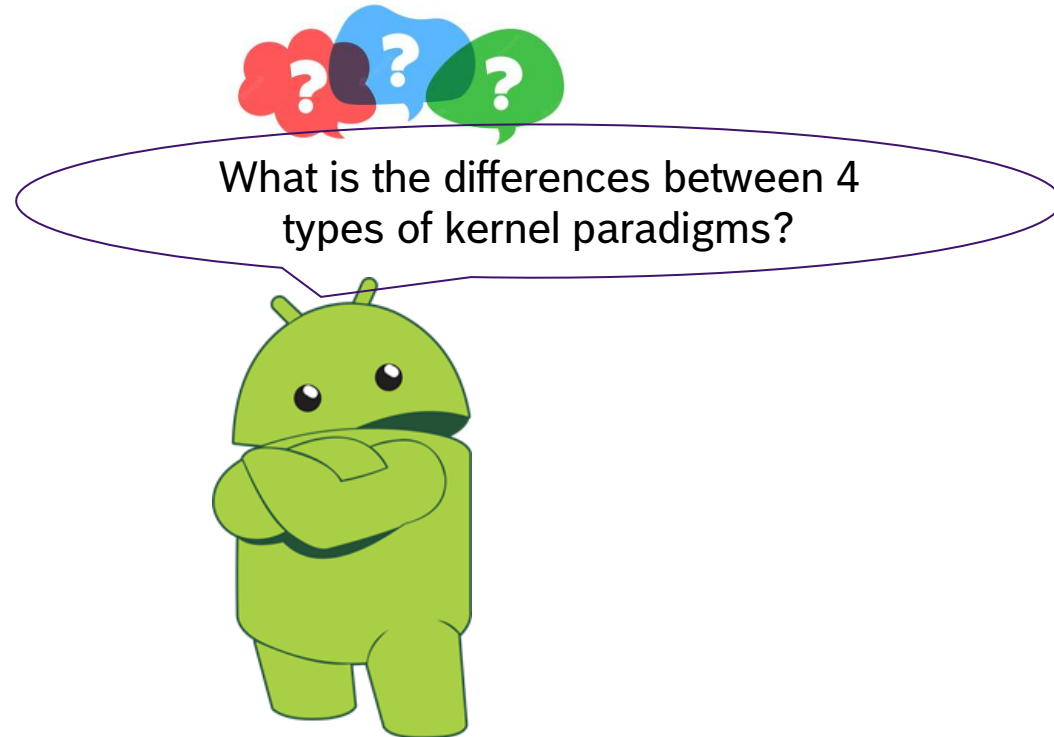


Đây là tỉnh thành nào?

QUIZ (1/10)



QUIZ (2/10)



QUIZ (3/10)



Which kernel paradigm is the most availability? Why?



QUIZ (4/10)



Which kernel paradigm is the most security? Why?

QUIZ (5/10)



Which kernel paradigm is the fastest response? Why?

QUIZ (6/10)



What is Android kernel paradigm? Why?

QUIZ (7/10)



Can Android use Linux power management mechanism?
Why?

QUIZ (8/10)



How can a music be played back on Android when the screen is off?

QUIZ (9/10)



Why does Google not use Linux logging mechanism?

QUIZ (10/10)



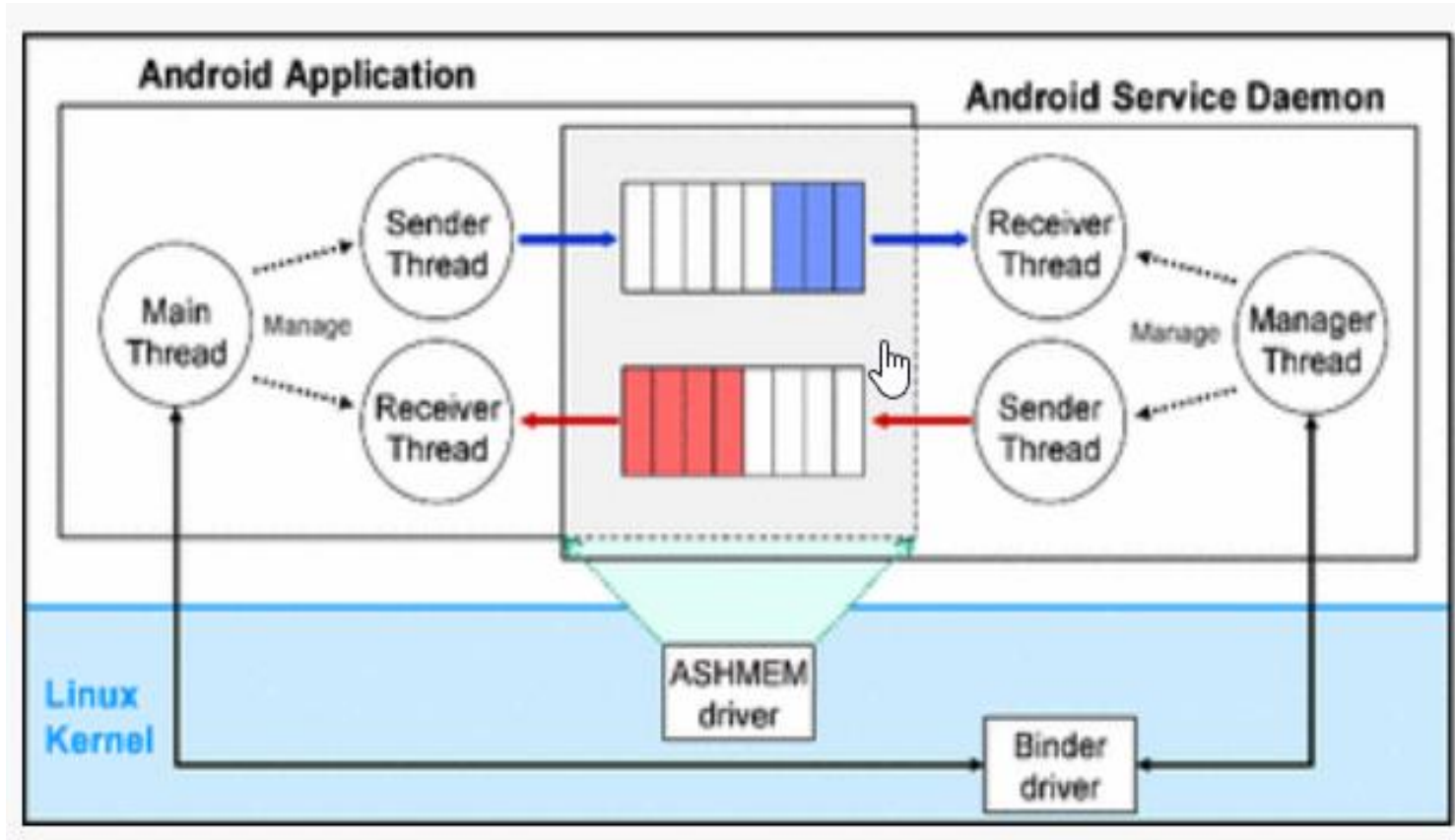
How many circular buffers are there in Android Logging mechanism. What are they?

Anonymous Shared Memory (ashmem)

- Another kind of IPC mechanism.
- It allows to share memory between processes.
- It is similar to POSIX SHM but different behaviors:
 - It uses reference counting to determine when it can destroy memory region.
 - It can shrink mapped region if system needs memory.
- Applications can open a character device (**/dev/ashmem**) and create a memory region and uses **Binder** to share corresponding file descriptor with other process if they want to share memory region.

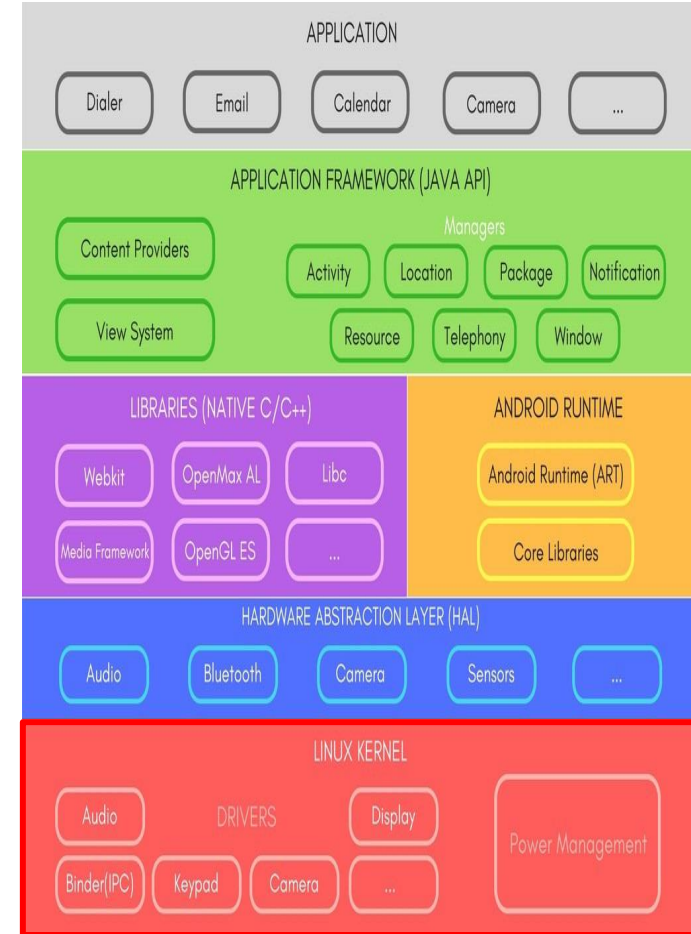


Anonymous Shared Memory (ashmem)



Alarm driver

- The timer mechanisms available in Linux were not sufficient for the power management policy that Android was trying to set up.
- High Resolution Timers can wake up a process, but don't fire when the system is suspended.
- Google gave a new mechanism called Real Time Clock. It can wake up the system if it is suspended.
- However, it cannot wake up a particular process.



Alarm driver

- Alarm timers was developed on top of the Real Time Clock and High Resolution Timers.
- These timers will be fired even if the system is suspended, waking up the device to do some actions
- When the application is woken up, a wakelock should be used to prevent the system going to suspended state again.



Low memory killer

- When the system goes out of memory, Linux throws the OOM Killer to cleanup memory greedy processes.
- However, this behaviour is not predictable at all, and can kill very important components of a phone (Telephony stack, Graphic subsystem, etc) instead of low priority processes.
- The main idea is to have another process killer, that kicks in before the OOM Killer. This process killer will base on LRU algorithm and process priority in the system.



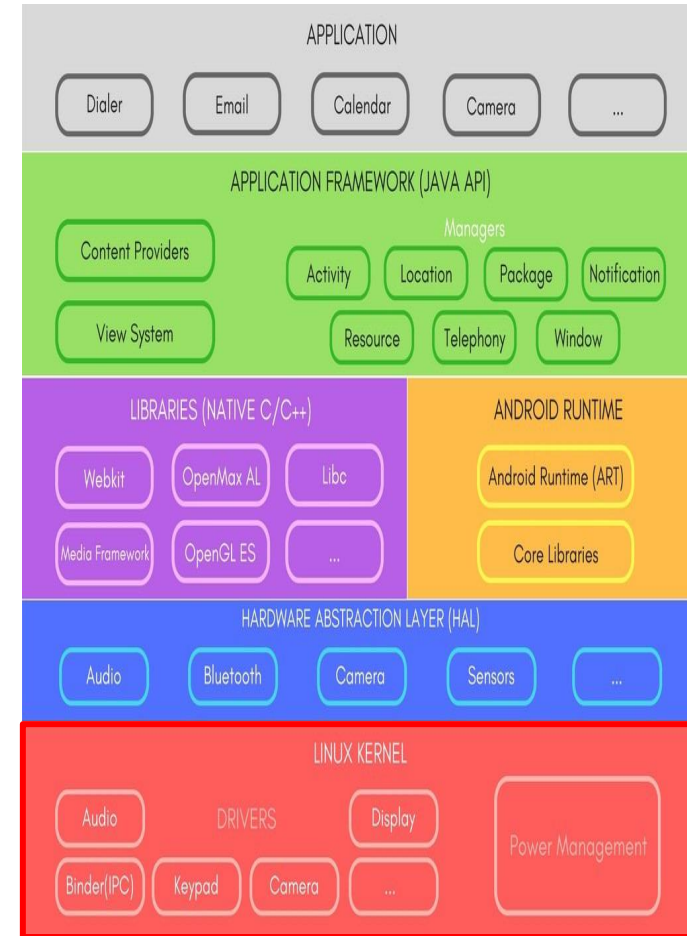
Low memory killer

- First of all, it will notify applications so that they can save their state, then begins to kill non-critical background processes, and then the foreground applications.
- Free memory is called before the OOM Killer. Therefore, foreground applications will not be killed (the system will never run out of memory).



The ION Memory Allocator

- ION was introduced since Ice Cream Sandwich (4.0).
- Its role is to allocate memory in the system and to allow different HW components to share buffers, without any copy from an user space application.
- For example, if you want to retrieve an image from a camera, and push it to the JPEG hardware encoder from an user space application.



The ION Memory Allocator

- Usually, Linux memory allocators can only allocate a buffer that is up to 512 pages, 4kiB per page.
- However, Android has a lot of vendors. Each vendor has specific mechanism to allocate larger physically contiguous memory areas (eg: nvmap for nVidia, CMEM for TI, etc.).
- ION is here to unify the interface to allocate memory in the system, no matter on which SoC you're running on.



The ION Memory Allocator

- It uses a system heaps available on a the system.
- By default, you have three different heaps:
 - *System*: Memory virtually contiguous memory, made by **vmalloc**.
 - *System contiguous*: Physically contiguous memory, made by **kmalloc**.
 - *Carveout*: Large physically contiguous memory, **preallocated at boot**.
- It also has a user space interface so that processes can allocate memory to work on.



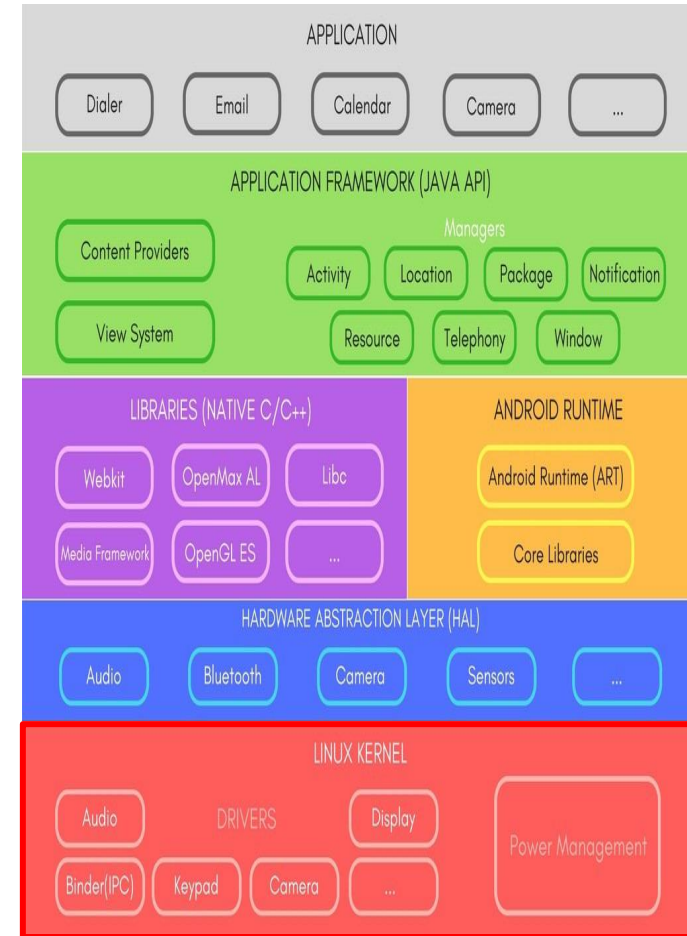
Network Security

- In the standard Linux kernel, every application can open sockets and communicate over the network.
- It is not good for mobile devices due to privacy policies.
- Therefore, Android provides network access with the permission granted from the application.
- We need the granted permission to be able to access IP, Bluetooth, raw sockets, etc.



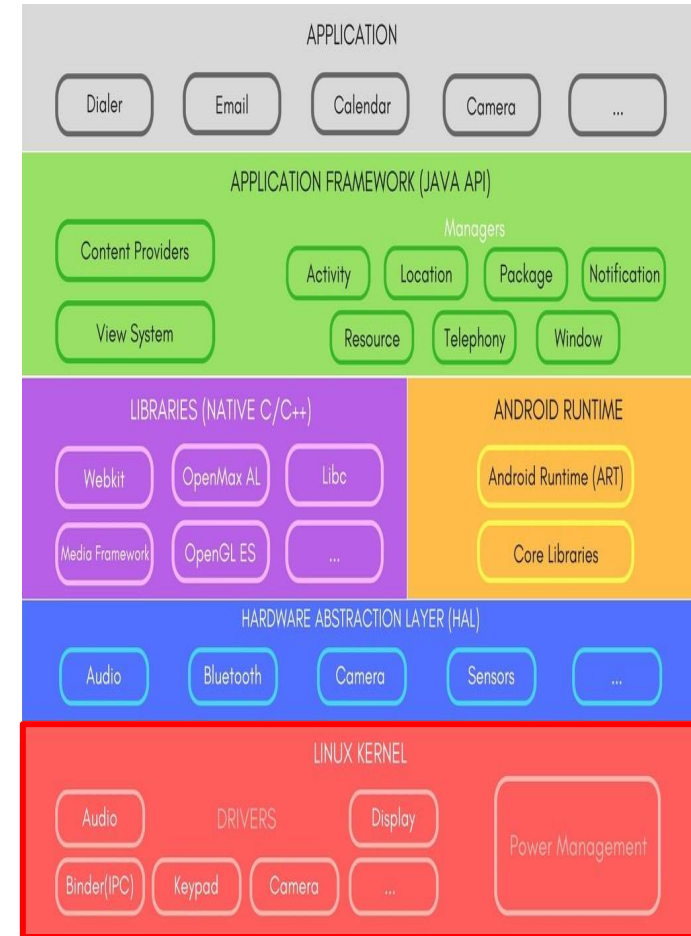
RAM Console

- A mechanism for preserving kernel panic output: thread dump and last dmesg log.
- However, this has been deprecated in newer releases. Currently, it used pstorefs.



Sync driver

- A mechanism allows fast synchronization primitives, between consumers and producers used primarily by Android's Graphics stack (in particular, surfaceflinger).

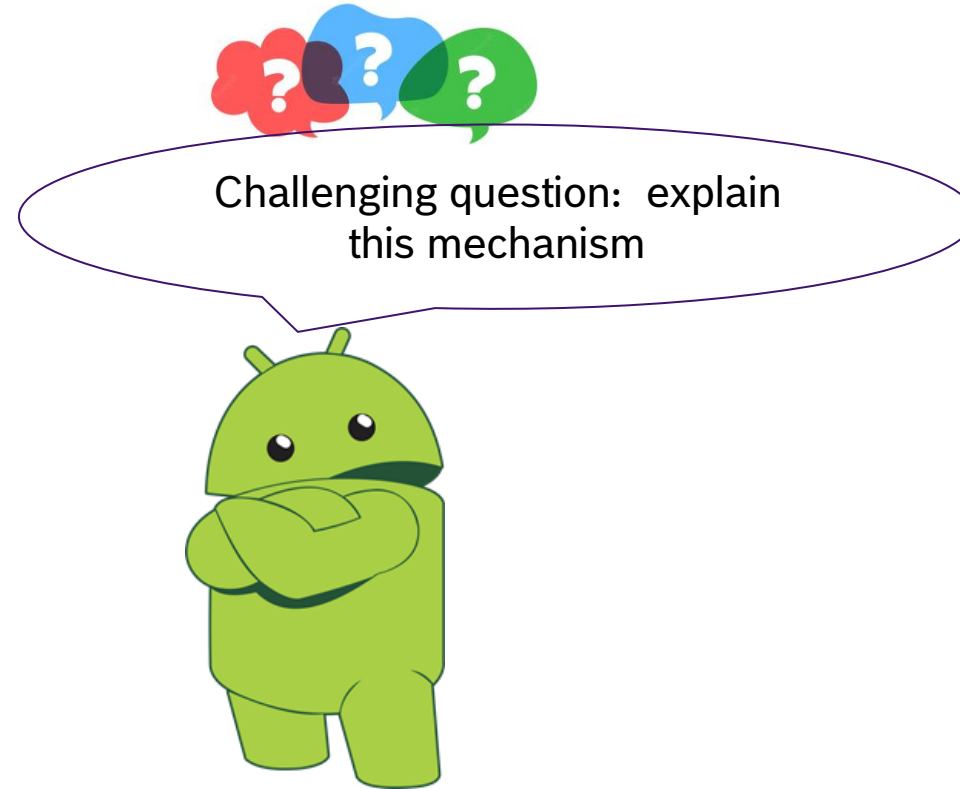


Timed Output and GPIO

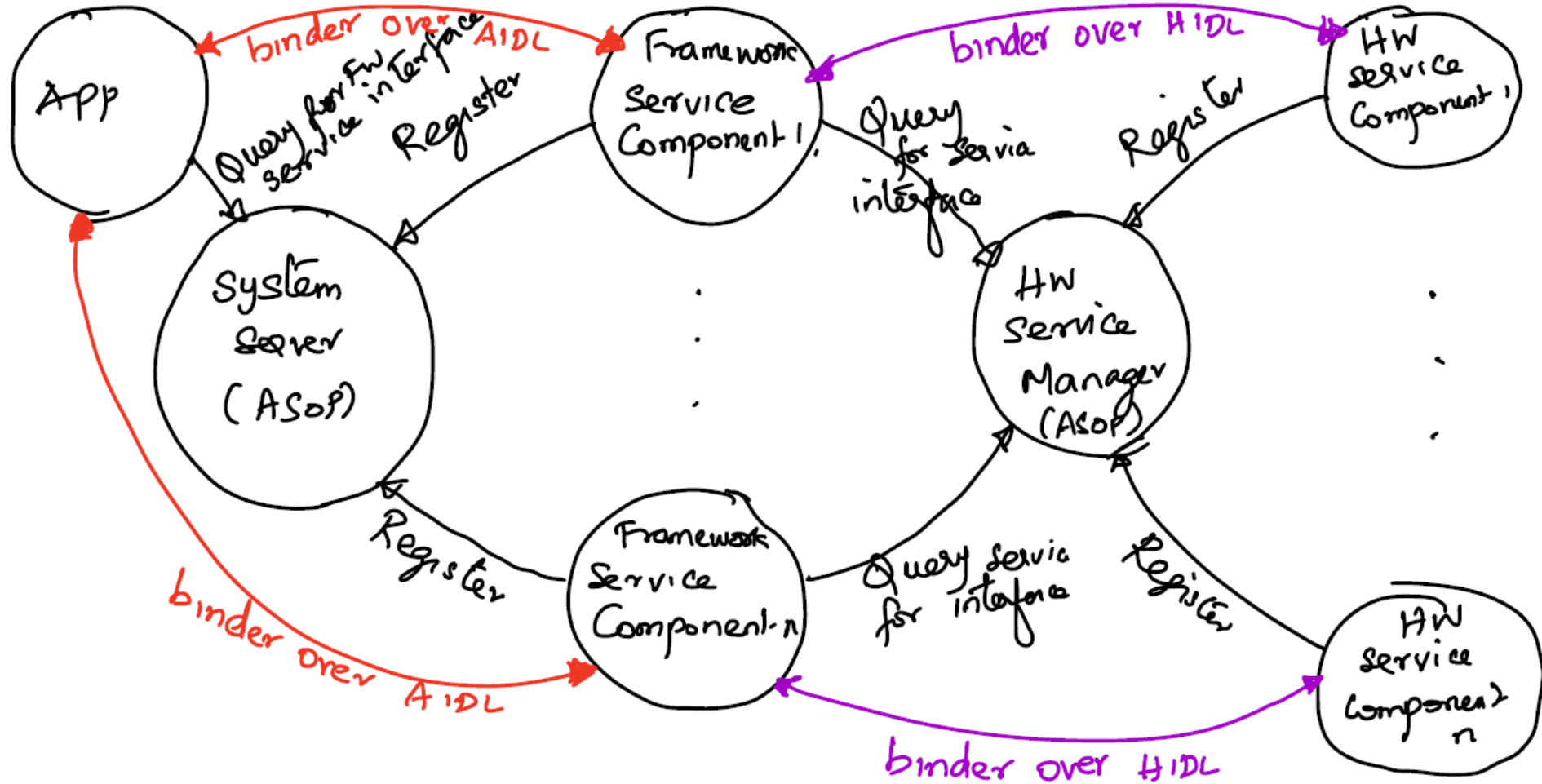
- A mechanism to allow user mode programs to access GPIO registers from user space, and automatically reset their values after a timeout.
- The main client of this is the device vibrator functionality:
 - The framework (via the HAL) can write a millisecond value into **/sys/timed_output/vibrator/enable**, to start the device vibration, which automatically quiets down after the timeout specified.



Group working



Binding mechanism




Exercises

Exercises

Download and install Android studio in your machine:

- 1) Create slides for some Android studio tool:
 - a) build, debug (breakpoint, step in, step out, step over, ...)
 - b) Emulator (AVD, Genymotion Emulator and add its Plugin to Android Studio)
 - c) Adb
 - d) Logcat and filter.
 - e) Real-time profilers
 - f) Unit test
- 2) Create slides and create demo (if applicable):
 - a) Different Types of Activities in Android Studio
 - b) 'res' folder
 - c) Different Android Layout
 - d) AndroidManifest file structure
 - e) Checkout and build Android Emulator in CCS2 project (*)



Thank for your listening!