

# Mobile Systems

Thierry Sans

# History of mobile OSes

➔ Early "smart" devices are PDAs (touchscreen, Internet)

Symbian, first modern mobile OS

- released in 2000
- run in Ericsson R380, the first "smartphone" (mobile phone + PDA)
- only support proprietary programs



# History of mobile OSes

Many smartphone and mobile OSes followed up

- Palm OS (2001)
- Windows CE (2002)
- Blackberry (2002)

One more thing ...



## Introduction of iPhone (2007)

- 4GB flash memory, 128 MB DRAM, multi-touch interface
- runs iOS only proprietary apps at first but App Store opened in 2008, allow third party apps



# Android – an unexpected rival of the iPhone

Android Inc. founded by *Andy Rubin et al.* in 2003

- original goal is to develop an OS for digital camera
- shift focus on Android as a mobile OS

The startup had a rough time [story]

- run out of cash, landlord threatens to kick them out
- later bought by Google
- no carrier wants to support it except for T-Mobile
- while preparing public launch of Android, iPhone was released

Android 1.0 released in 2008 (HTC G1)

- In 2019, ~87% of mobile OS market (iOS ~13%)

# Why are mobile OSes interesting?

Now an essential device part of people's daily life  
(sometimes the only computing device)

- ➔ Mobile OSes and traditional OSes share the same core abstractions ... but also have many unique designs

# Design considerations for mobile OS

Resources are very constrained

- Limited memory
- Limited storage
- Limited battery life
- Limited processing power
- Limited network bandwidth
- Limited size

- ➔ User perception are important: Latency  $\gg$  throughput  
Users will be frustrated if an app takes several seconds to launch
- ➔ Environment are frequently changing  
Cellular signals from strong to weak and then back to strong

# Process management in mobile OS

In desktop/server - an application = a process

Not true in mobile OSes

- When you see an app present to you it does not mean an actual process is running
- Multiple apps might share processes
- An app might make use of multiple processes
- When you "close" an app, the process might be still running

➔ Different user-application interaction patterns



# Process management in mobile OS

Multitasking is a luxury in mobile OS

- Early versions of iOS did not allow multi-tasking mainly because of battery life and limited memory
  - Only one app runs in the foreground, all other user apps are suspended
  - OS's tasks are multi-tasked because they are assumed to be well-behaving
- ➔ Starting with iOS 4, the OS APIs allow multi-tasking in apps but only available for a limited number of app types

# Memory management in mobile OS

Most desktop and server OSes today support swap space

Mobile OSes typically do not support swapping

- iOS asks applications to voluntarily relinquish allocated memory
- Android will terminate an app when free memory is running low

➡ App developers must be very careful about memory usage

# Storage in mobile OS

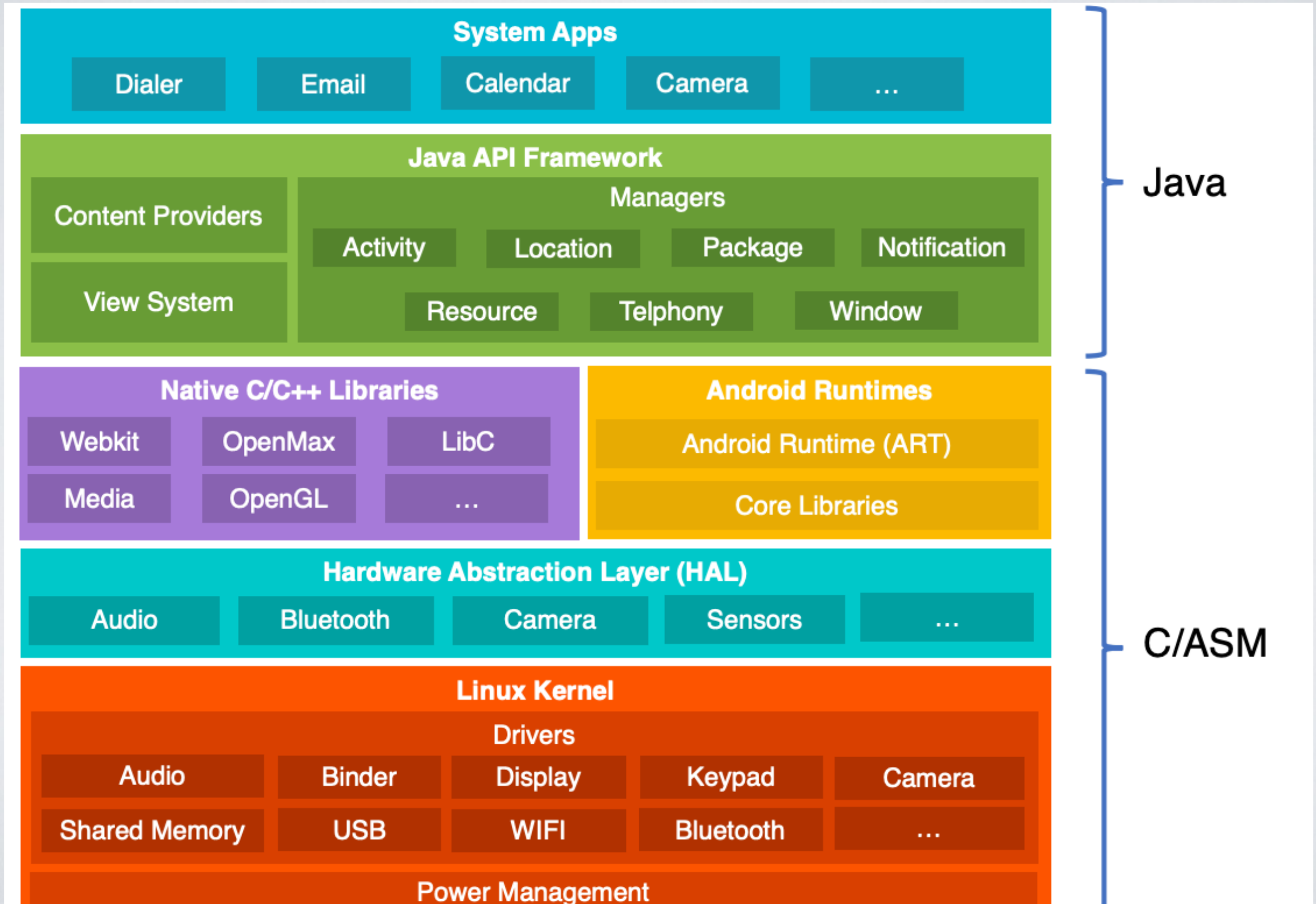
App privacy and security is hugely important in mobile device

- Each app has its own private directory that other apps cannot access
- Only shared storage is external storage

High-level abstractions

- Files
- Database (SQLite)
- Preferences (key-value pairs)

# Android OS stack





# Linux kernel vs. Android kernel

➔ Linux kernel is the foundation of Android platform

New core code

- binder - interprocess communication mechanism
- shmem - shared memory mechanism
- logger

Performance/power

- wakelock
- low-memory killer
- CPU frequency governor

➔ and much more ... 361 Android patches for the kernel

# Android runtime

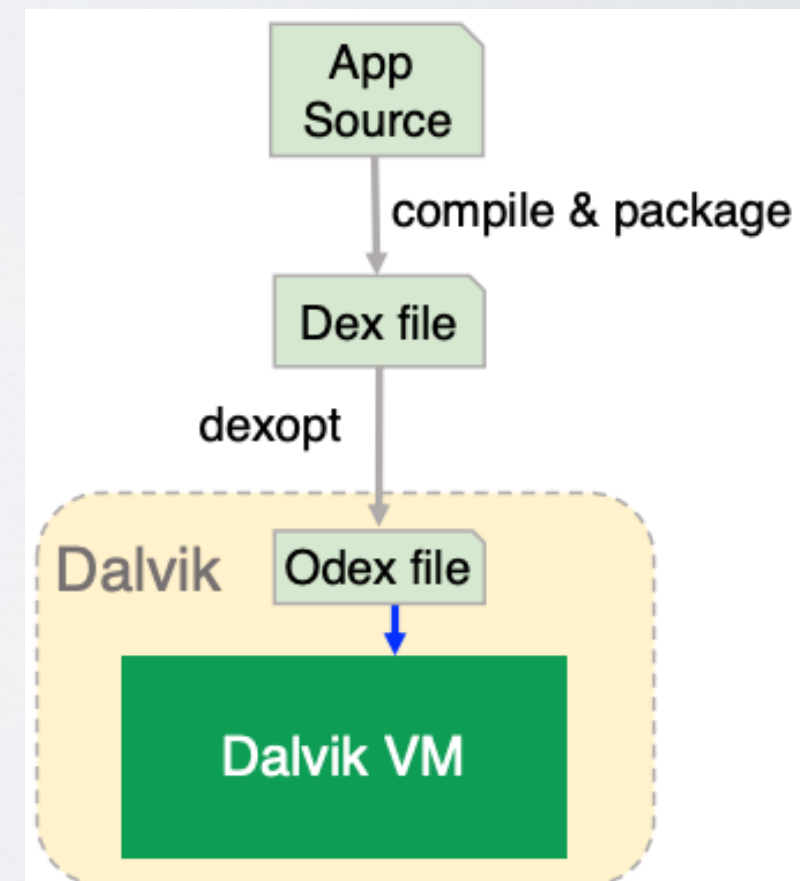
- ➔ Runtime - a component provides functionality necessary for the execution of a program  
E.g., scheduling, resource management, stack behavior

## Prior to Android 5.0 - Dalvik

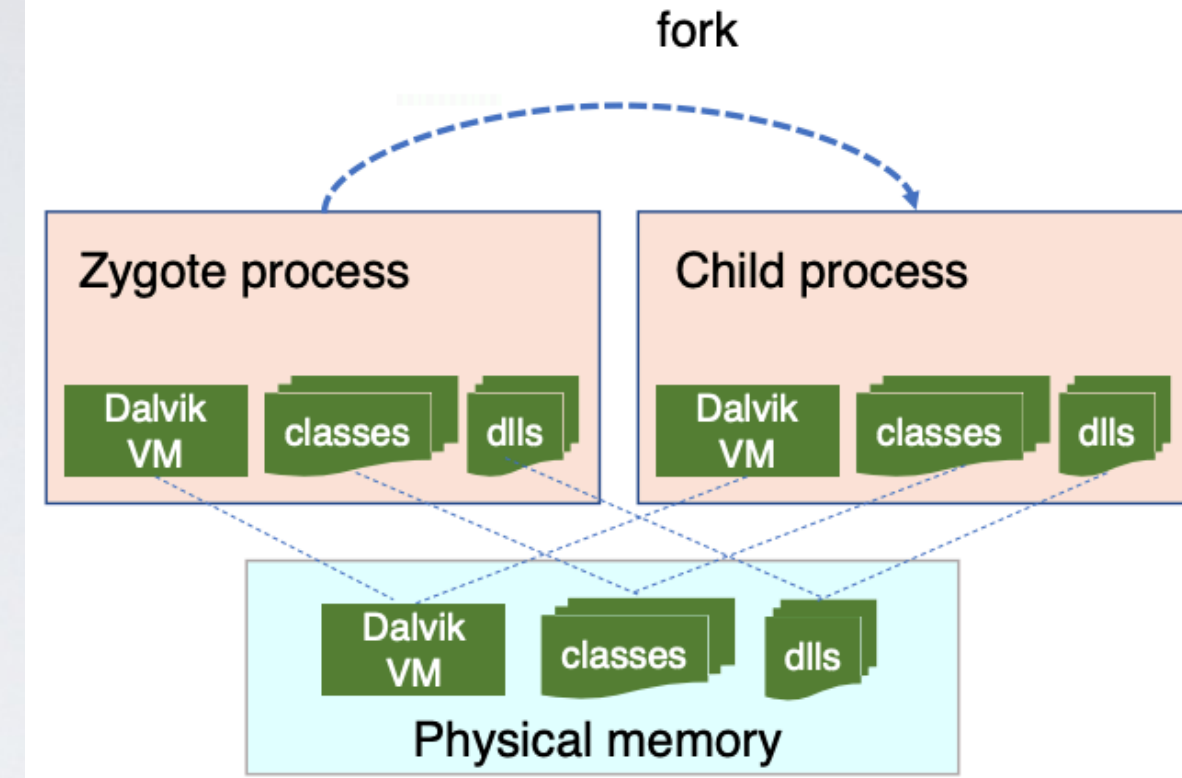
- Each Android app has its own process, runs its own instance of the Dalvik virtual machine (process virtual machine)
- The VM executes the Dalvik executable (.dex) format
- Register-based compared to stack-based of JVM

## After Android 5.0 - ART

- Backward compatible for running Dex bytecode
- New feature - Ahead-Of-Time (AOT) compilation
- Improved garbage collection



# Android process creation



All Android apps derive from a process called Zygote

- Zygote is started as part of the init process
- Preloads Java classes, resources, starts Dalvik VM
- Registers a Unix domain socket
- Waits for commands on the socket
- Forks off child processes that inherit the initial state of VMs

➔ Uses Copy-on-Write

only when a process writes to a page will a page be allocated

# Java API framework

The main Android OS from app point of view

- Provide high-level services and environment to apps
- Interact with low-level libraries and Linux kernel

Some components

- Activity Manager - manages the lifecycle of apps
- Package Manager - keeps track of apps installed
- Power Manager - wakelock APIs to apps



# Native C/C++ libraries

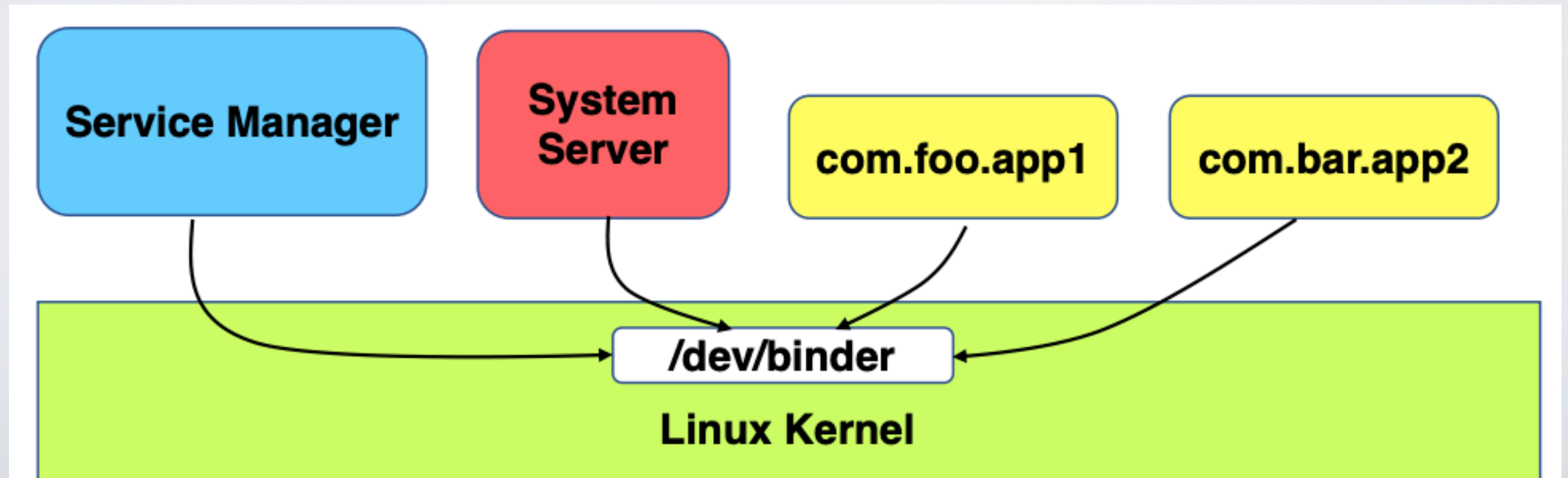
Many core Android services are built from native code

- Require native libraries written in C/C++
- Some of them are exposed through the Java API framework as native APIs e.g. Java OpenGL API

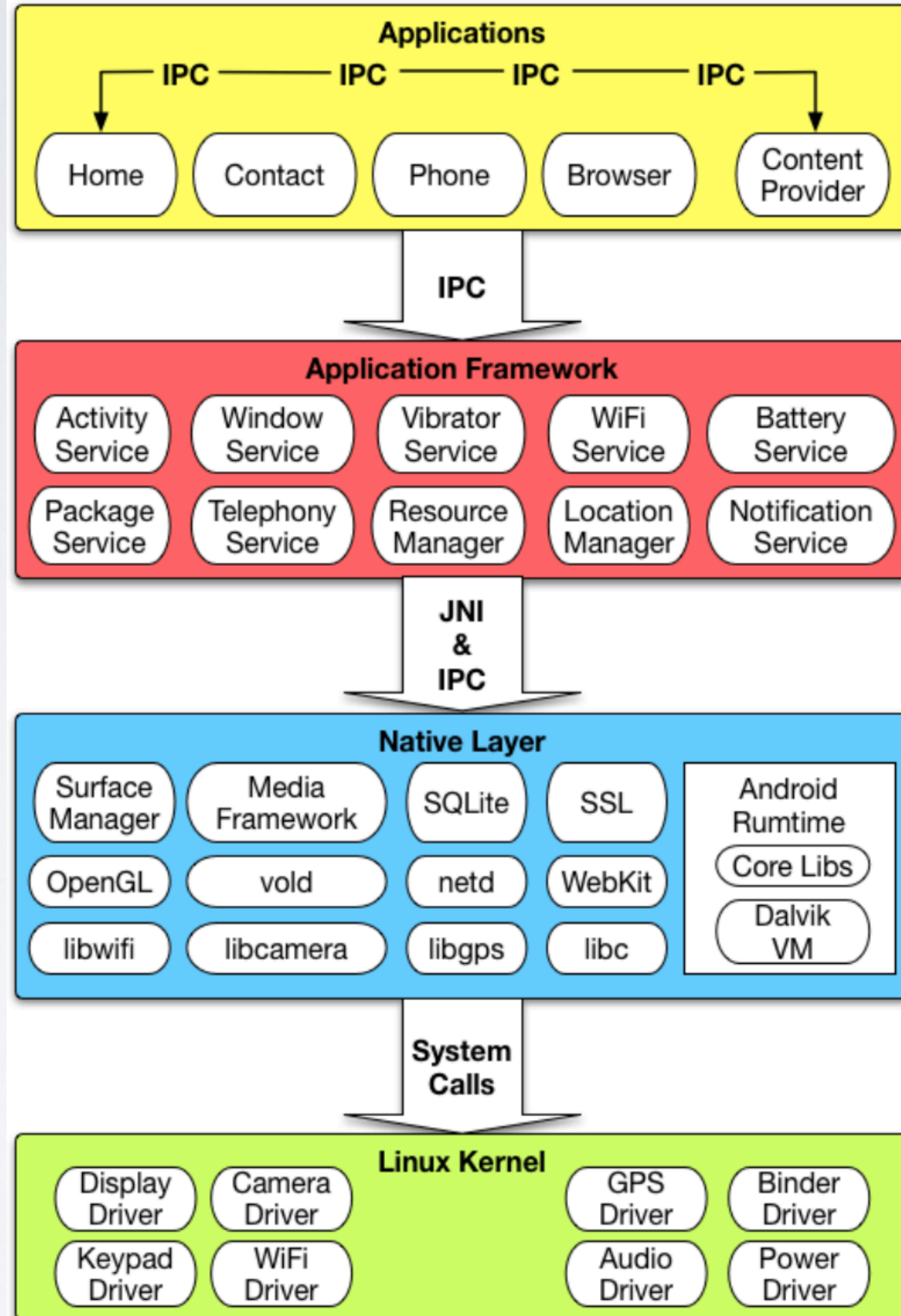
➔ Technique: JNI – Java Native Interface  
app developer can use Android NDK to include C/C++ code (common in gaming apps)

# Android Binder IPC

**Android Binder IPC** allows communication among apps, between system services, and between app and system service



IPC is pervasive  
in Android



# Binder is implemented as an RPC

1. Developer defines methods and object interface in an `.aidl` file

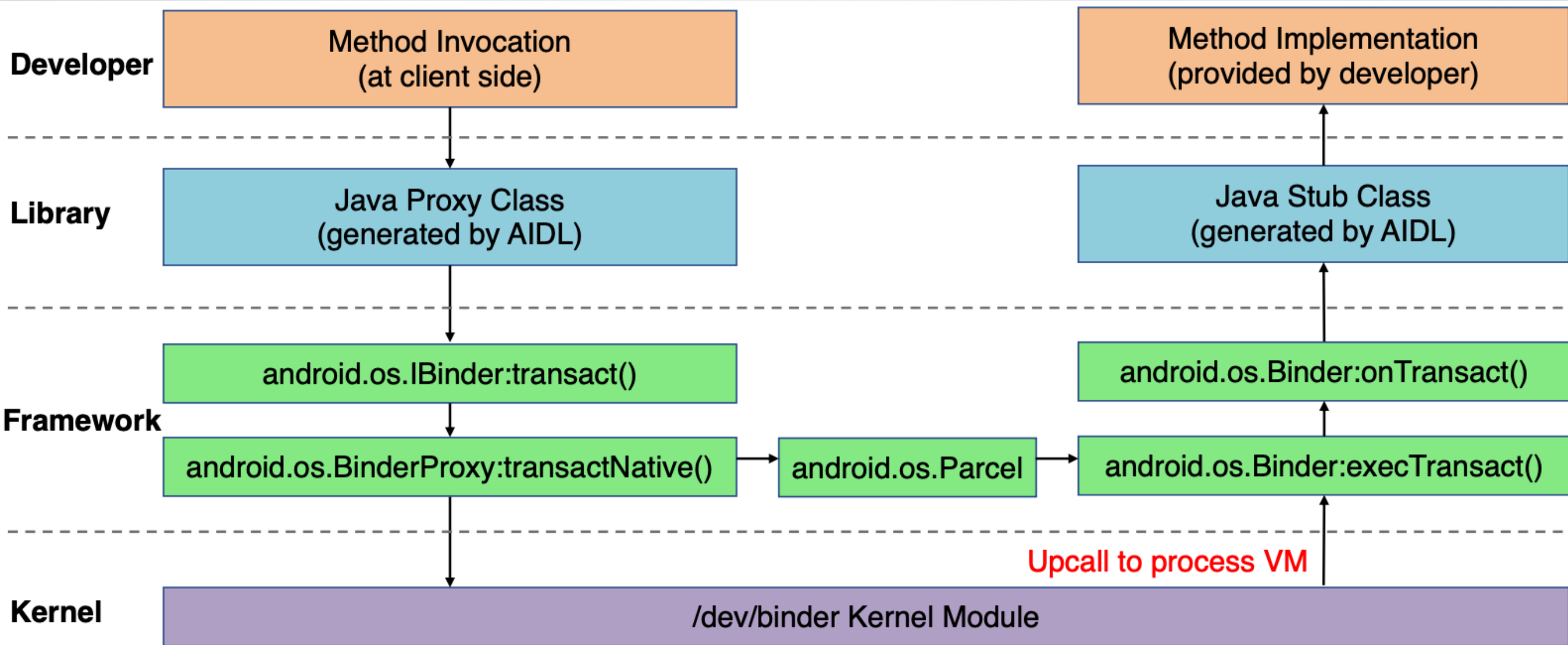
```
package com.example.android; // IRemoteService.aidl

/** Example service interface */
interface IRemoteService {
    /** Request the process ID of this service, to do evil things with it. */
    int getPid();
    /** Pause the service for a while */
    void pause(long time);
}
```

2. Android SDK generates a stub Java file for the `.aidl` file and exposes the stub in a Service
3. Developer implements the stub methods
4. Client copies the `.aidl` file to its source
5. Android SDK generates a stub (a.k.a proxy)
6. Client invoke the RPC through the stub



# Binder information flow



# Some other interesting topics in mobile OSes

- Energy management
- Dealing with misbehaving apps
- Security

# Summary

➡ Smartphone has become an ubiquitous computing device

Mobile OS is an interesting and challenging subject

- Constrained resources
- Different user interaction patterns
- Frequently changing environment
- Untrusted, immature third-party apps

Some unique design choices

- Application  $\neq$  process
- Multitasking
- No swap space
- Private storage

# Acknowledgments

Some of the course materials and projects are from

- Ryan Huang - teaching CS 318 at *John Hopkins University*
- David Mazière - teaching CS 140 at *Stanford*