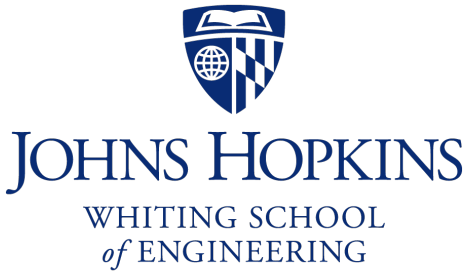
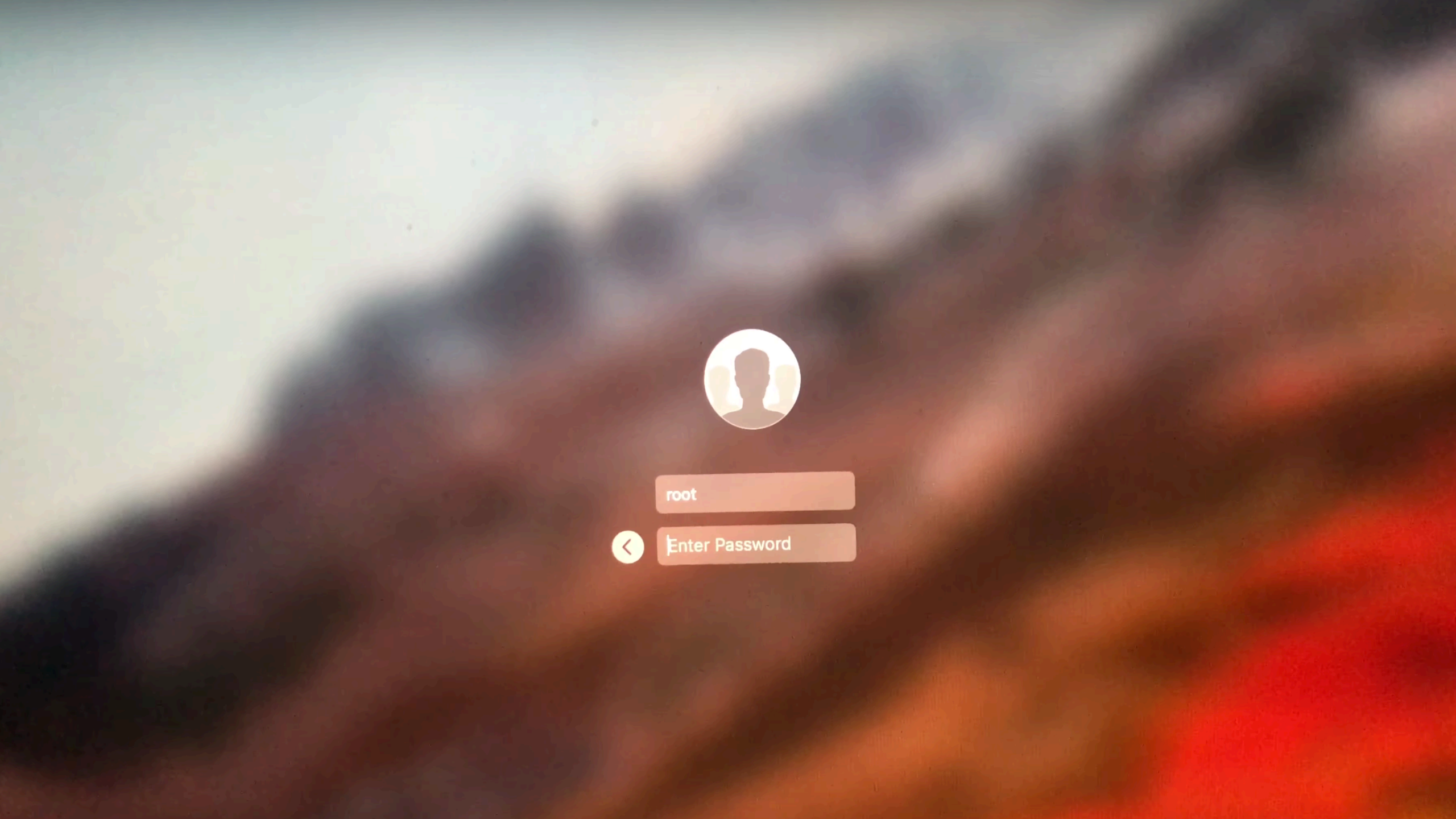
CS 318 Principles of Operating Systems

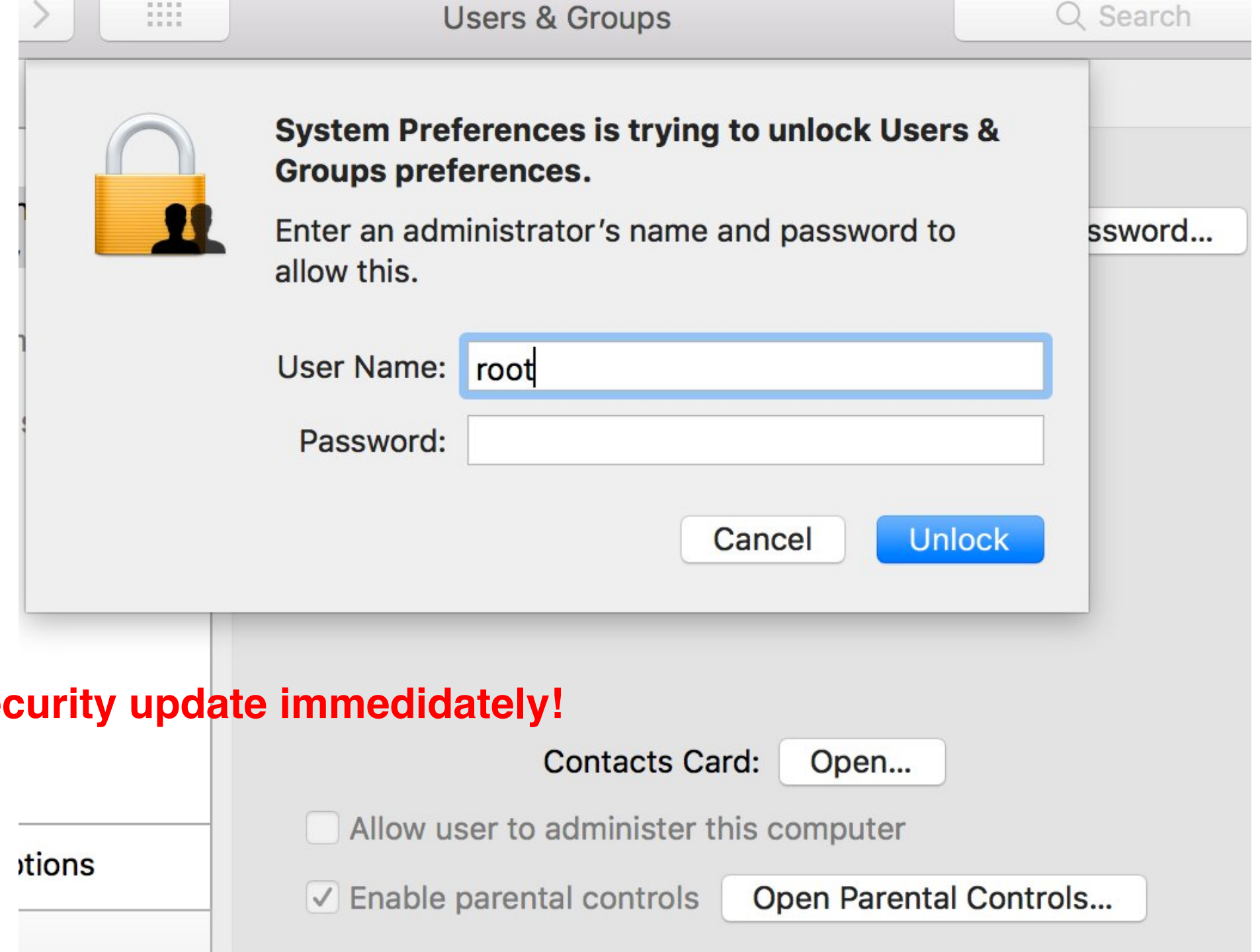
Fall 2017

Lecture 21: Mobile Systems

Ryan Huang




root
Enter Password



Apply the security update immedidately!

Administrivia

- **Lab 4 deadline one week away**
- **Groups of 2 students receive 2-day extra late hour**
- **Groups of 3 students with 1 318 section student receive 1-day extra late-hour**
- **Please, please don't cheat**
- **Homework 5 is released**

Mobile Devices Become Ubiquitous

Google Nexus 6P

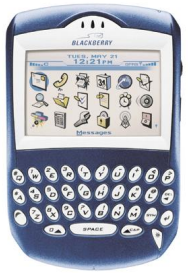
NETWORK	Technology	GSM / CDMA / HSPA / LTE EXPAND
DISPLAY	Type	AMOLED capacitive touchscreen, 16M colors
	Size	5.7 inches (~71.4% screen-to-body ratio)
	Resolution	1440 x 2560 pixels (~518 ppi pixel density)
	Multitouch	Yes
	Protection	Corning Gorilla Glass 4, oleophobic coating
PLATFORM	OS	Android OS, v6.0 (Marshmallow)
	Chipset	Qualcomm MSM8994 Snapdragon 810
	CPU	Quad-core 1.55 GHz Cortex-A53 & Quad-core 2.0 GHz Cortex-A57
	GPU	Adreno 430
MEMORY	Card slot	No
	Internal	32/64/128 GB, 3 GB RAM
CAMERA	Primary	12.3 MP, f/2.0, laser autofocus, dual-LED (dual tone) flash, check quality
	Features	1/2.3' sensor size, 1.55µm pixel size, geo-tagging, touch focus, face detection, HDR, panorama
	Video	2160p@30fps, 720p@240fps, check quality
	Secondary	8 MP, f/2.4, 1080p@30fps
SOUND	Alert types	Vibration; MP3, WAV ringtones
	Loudspeaker	Yes, with front stereo speakers
	3.5mm jack	Yes
COMMS	WLAN	Wi-Fi 802.11 a/b/g/n/ac, dual-band, Wi-Fi Direct, DLNA, hotspot
	Bluetooth	v4.2, A2DP, LE
	GPS	Yes, with A-GPS, GLONASS
	NFC	Yes
	Radio	No
	USB	v2.0, Type-C 1.0 reversible connector
FEATURES	Sensors	Fingerprint, accelerometer, gyro, proximity, compass, barometer
	Messaging	SMS(threaded view), MMS, Email, Push Mail, IM
	Browser	HTML5
	Java	No
		- Fast charging - Active noise cancellation with dedicated mics - MP4/H.264 player - MP3/WAV/eAAC+ player - Photo/video editor - Document editor
BATTERY		Non-removable Li-Po 3450 mAh battery

History of Mobile OS (1)

- **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 OS (2)

- **Many smartphone and mobile OSes followed up**
  - Kyocera 6035 running Palm OS (2001)
    - 8 MB non-expandable memory
  - Windows CE (2002)
  - **Blackberry (2002)**
    - was a prominent vendor
    - known for secure communications
  - Moto Q (2005)
  - Nokia N70 (2005)
    - 2-megapixel camera, bluetooth
    - 32 MB memory
    - Symbian OS
    - Java games

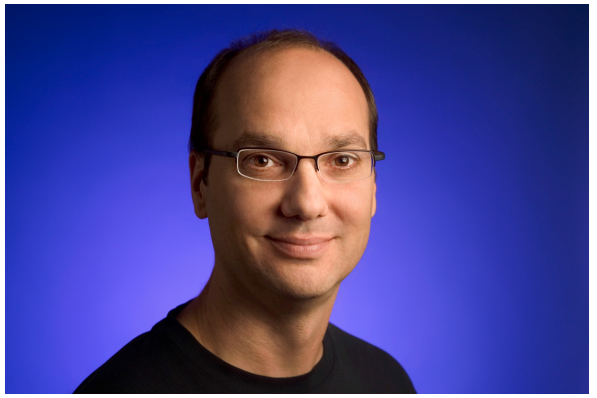


One More Thing…

- **Introduction of iPhone (2007)**
  - revolutionize the smartphone industry
  - 4GB flash memory, 128 MB DRAM, multi-touch interface
  - runs iOS, initially only proprietary apps
  - App Store opened in 2008, allow third party apps

Android – An Unexpected Rival of 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)**
- **Today: ~88% of mobile OS market**
  - iOS ~11%

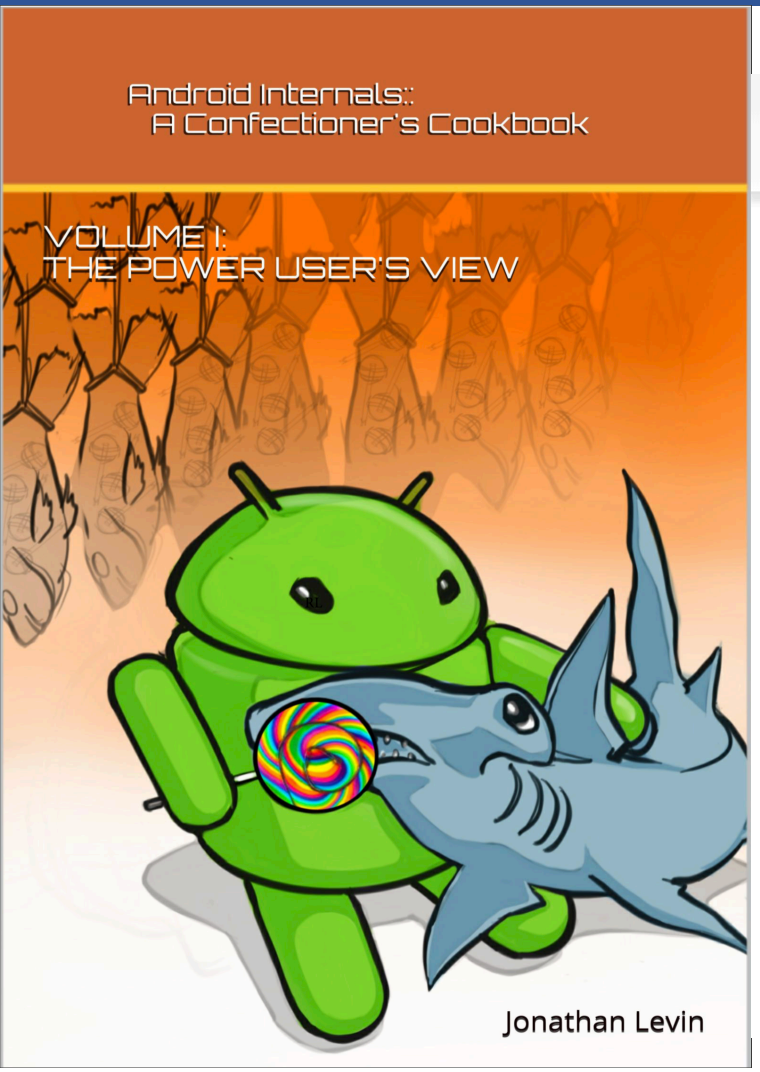


Android Releases

Why Are Mobile OSes Interesting?

- **They are running in every mobile device as an essential part of people's daily life, even for non-technical users**
  - In many developing countries, the only computing device a person has is a phone
- **Mobile OSes and traditional OSes share the same core abstractions but also have many unique designs**
  - Comparing and contrasting helps you understand the whole OS design space
  - You will be surprised to see that some concepts in distributed systems can be applied in mobile OS as well
- **It will make you a more efficient mobile user and developer**

Why Are Mobile OSes Interesting?



Android Internals::Power User's View

Available from these sellers.

Top customer reviews

★★★★★ **As an an Android app developer 99% of the time ...**

By Boris Farber on August 3, 2015

Format: Paperback | **Verified Purchase**

As an an Android app developer 99% of the time you do not need the material written there. However the 1% bug will come, and as we know it comes usually in the most unexpected time followed by extreme pressure.

While tracking that 1% bug you will thank yourself for having this book handy. While chasing the bug you will appreciate the clear picture and the solid flow of the Android architecture the book gives, neither spending a second on fluff, nor stating the obvious.

By getting this clear picture you will be able to grasp where your app is wrong and what is the system behaviour. This will save your day and build your reputation as an Android expert.

For me the most important material presented in the book was around Android vs Linux, File Systems, Framework Service Architecture and Security. You don't know when that 1% bug will come, but the knowledge from the book will definitely help to sort it out.

Design Considerations for Mobile OS

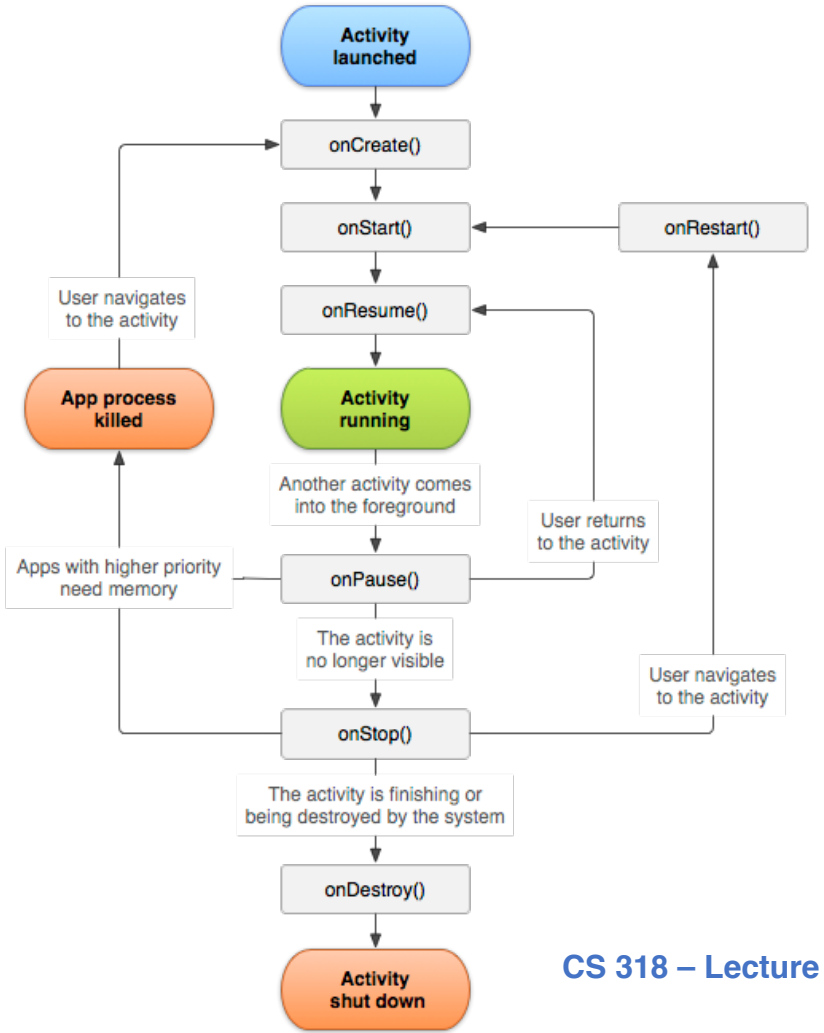
- **Resources are very constrained**
  - Limited memory
  - Limited storage
  - Limited battery life
  - Limited processing power
  - Limited network bandwidth
  - Limited size
- **User perception are important**
  - Latency outweighs throughput
    - Users will be frustrated if an app takes several seconds to launch
- **Environment are frequently changing**
  - The whole point about being mobile
  - Cellular signals from strong to weak and then back to strong

Process Management in Mobile OS (1)

- **In desktop/server: an application = a process**
- **Not true in modern mobile OSes like Android**
  - When you see an app present to you, doesn't 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
    - **Why?**
    - "*all applications are running all of the time*"
- **Different user-application interaction patterns**
  - Check Facebook for 1 min, switch to Reminder for 10s, Check Facebook again
  - Server: launch a job, waits for result

Process Management in Mobile OS (2)

- **Example: Android app Activity lifecycle**



Process Management in Mobile OS (3)

- **Multitasking is a luxury in mobile OS**
  - Early versions of iOS don't allow multi-tasking
    - Not because the CPU doesn't support it
    - 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
- **Different philosophies among mobile OSes**
  - Android gives more freedom to developers: apps are allowed to run in background
    - Define Service class, e.g., to periodically fetch tweets
    - When system runs low in memory, kill an app
      - But what to do when user re-launches the app?

Memory Management in Mobile OS

- **Most desktop and server OSes today support swap space**
  - Allows virtual memory to grow beyond physical memory size
  - When physical memory is full utilized, evict some pages to disk
- **Smartphones use flash memory rather than hard disk**
  - Capacity is very constrained: 16 GB vs. 512 GB
  - Limited number of writes in its lifetime
  - Poor throughput between main memory and flash memory
- **As a result, mobile OSes typically don't support swapping!**
  - iOS ***asks*** applications to voluntarily relinquish allocated memory
  - Android will terminate an app when free memory is running low
  - What about paging?
- **App developers must be very careful about memory usage**

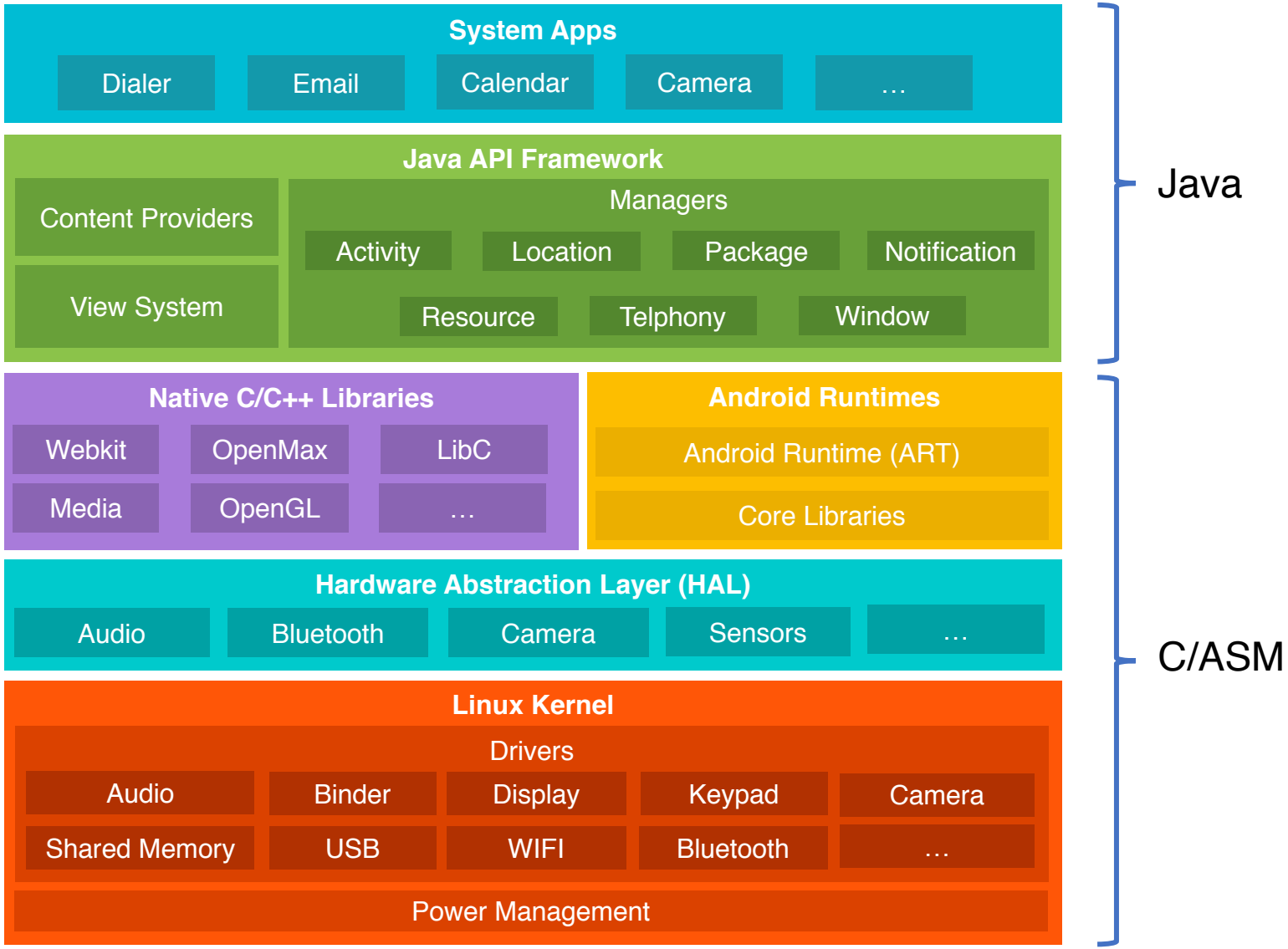
Storage in Mobile OS

- **App privacy and security is hugely important in mobile device**
  - Each app has its own private directory that other app can't access
  - Only shared storage is external storage
    - /sdcard/
- **High-level abstractions**
  - Files
  - Database (SQLite)
  - Preferences (key-value pairs)

```
ryan@orderlab:~$adb shell
shell@shamu:/ $ cd /data/app
shell@shamu:/data/app $ ls
opendir failed, Permission denied
255|shell@shamu:/data/app $ su
root@shamu:/data/app # ls
com.android.chrome-2
com.android.vending-2
com.facebook.katana-1
com.google.android.apps.docs.editors.docs-1
com.google.android.apps.maps-1
com.google.android.apps.messaging-1
com.google.android.gms-2
com.google.android.googlequicksearchbox-1
com.google.android.instantapps.supervisor-2
com.google.android.play.games-1
com.google.android.youtube-1
com.jumobile.manager.systemapp-1
com.ketchapp.stack-1
com.progames01.tanks.playtank-1
com.rovio.angrybirds-1
com.snapchat.android-1
edu.jhu.order.appstatstracker-1
```

A Primer on Android OS

Android OS Stack



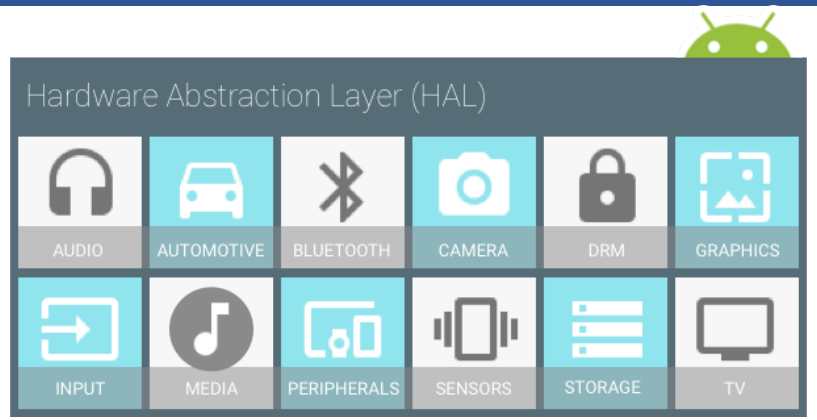
Linux Kernel vs. Android Kernel

- **Linux kernel is the foundation of Android platform**
- **New core code**
  - binder - interprocess communication mechanism
  - ashmem - shared memory mechanism
  - logger
- **Performance/power**
  - wakelock
  - low-memory killer
  - CPU frequency governor
- **and much more . . . 361 Android patches for the kernel**

Some Controversial Changes to Linux Kernel

- **Android Suspend Blockers**
  - System by default is in a sleep state
  - When there is a wakeup signal, CPU wakes up doing some work
  - As soon as the requested work is completed, CPU goes back to sleep state
    - Full system is suspended
  - Called *opportunistic suspend*
    - Different from generic Linux strategy
- **Sounds like a good idea, but devils are in the details**
  - Problem: race condition between system suspend and system wakeup
    - E.g., during suspend, a wakeup event comes, will only be delivered after another wakeup event comes later
      - What if this event is an incoming phone call?
    - Android solution: WakeLock, more control, but can be easily abused

Hardware Abstraction Layer (HAL)



- **Standard interfaces between hardware and Java API framework**
  - Enables Android to be agnostic about lower-level driver implementations
  - Hardware vendors just implement this interface without modifying/affecting high-level systems
  - Stored as a shared library module
- **When a framework API makes a call to access device hardware, Android loads the library module for that hardware component**

Hardware Abstraction Layer (HAL)

- **Generic *module* data structure**
  - defines version, name, etc.
  - a pointer to a method struct that in turn contains pointer to the open function
    - initiate communication with the device
- **Each specific HAL *module* usually extends the data structure**

```
struct hw_module_t {
    uint32_t tag;
    uint16_t version_major;
    uint16_t version_minor;
    const char *id;
    const char *name;
    const char *author;
    struct hw_module_methods_t* methods;
    uint32_t reserved[32-6];
};
```

```
typedef struct camera_module {
    hw_module_t common;
    int (*get_number_of_cameras)(void);
    int (*get_camera_info)(int camera_id, struct camera_info *info);
    int (*set_callbacks)(const camera_module_callbacks_t *callbacks);
} camera_module_t;
```

Hardware Abstraction Layer (HAL)

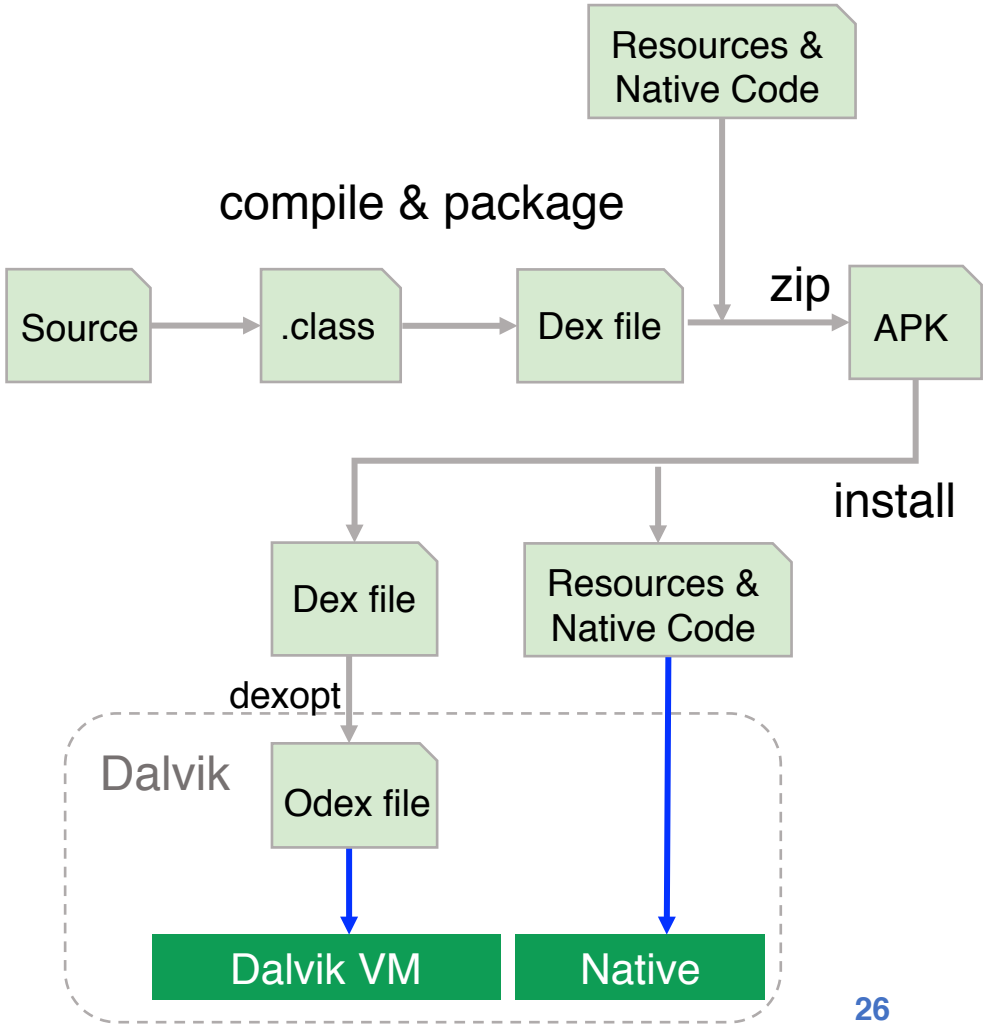
- **Similarly a generic hardware *device* data structure**
- **Each specific HAL *device* extends this data structure**
  - Includes detailed operations to be provided on this device

```
typedef struct camera_device {
    hw_device_t common;
    camera_device_ops_t *ops;
    void *priv;
} camera_device_t;
```

```
typedef struct camera_device_ops {
    …
    int (*start_preview)(struct camera_device *);
    void (*stop_preview)(struct camera_device *);
    int (*start_recording)(struct camera_device *);
    void (*stop_recording)(struct camera_device *);
    int (*take_picture)(struct camera_device *);
    int (*cancel_picture)(struct camera_device *);
    …
} camera_device_ops_t;
```

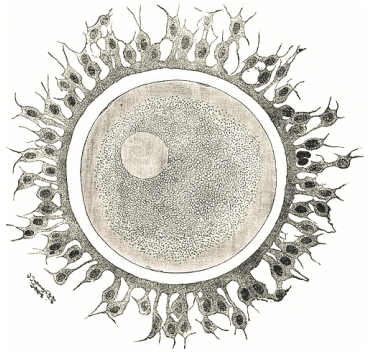
Android Runtime

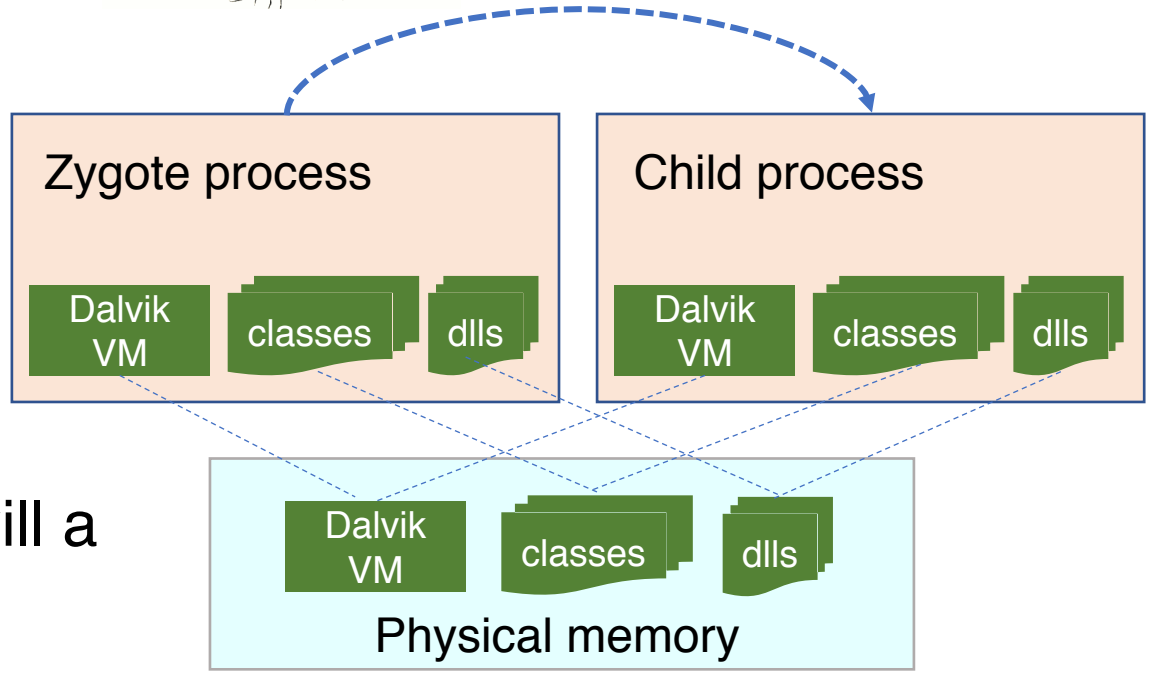
- **What is a runtime?**
  - A component provides functionality necessary for the execution of a program
    - E.g., scheduling, resource management, C's stack behavior
- **Prior to Android 5.0, Dalvik is the runtime**
  - Each Android app has its own process, runs its own instance of the Dalvik virtual machine (*process virtual machine*)
  - The VM executes the Dalvik executable (.dex) format
  - Register-based compared to stack-based of JVM
- **ART introduced in Android 5.0**
  - Backward compatible for running Dex bytecode
  - New feature: Ahead-of-time (AOT) compilation
  - Improved garbage collection



Android Runtime - Zygote

- **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





Native C/C++ Libraries

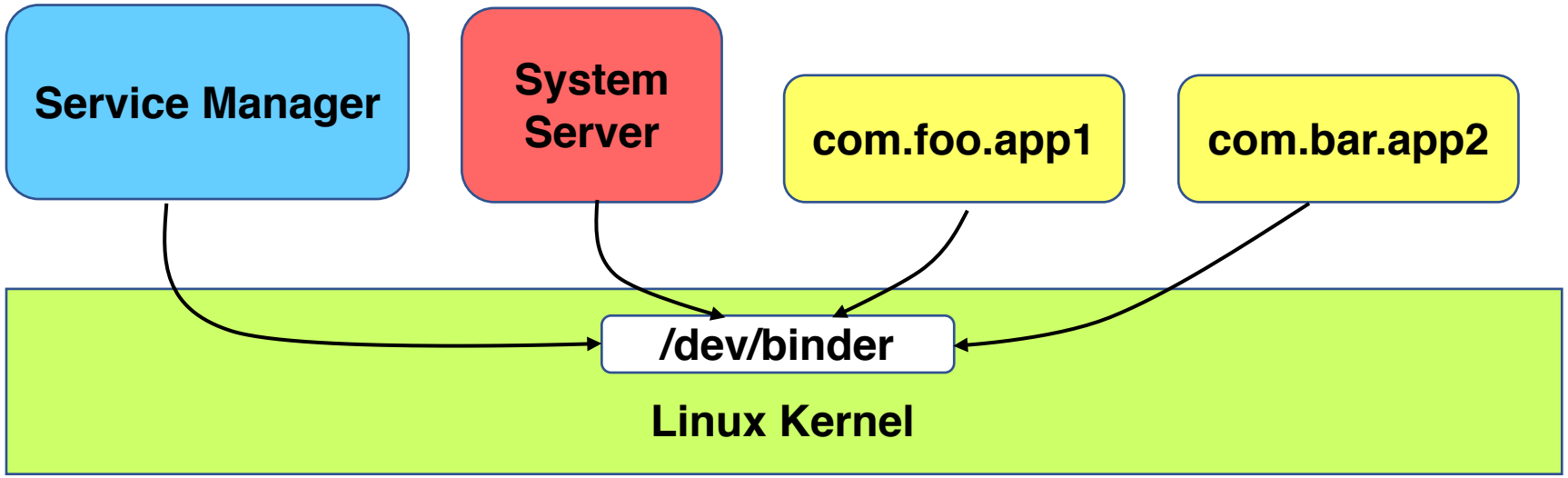
- **Many core Android services are built from native code**
  - Require native libraries written in C/C++
  - Performance benefit
  - Some of them are exposed through the Java API framework as native APIs
    - E.g., Java OpenGL API
- **Technique: JNI – Java Native Interface**
- **App developer can use Android NDK to include C/C++ code**
  - Common in gaming apps

Java API Framework

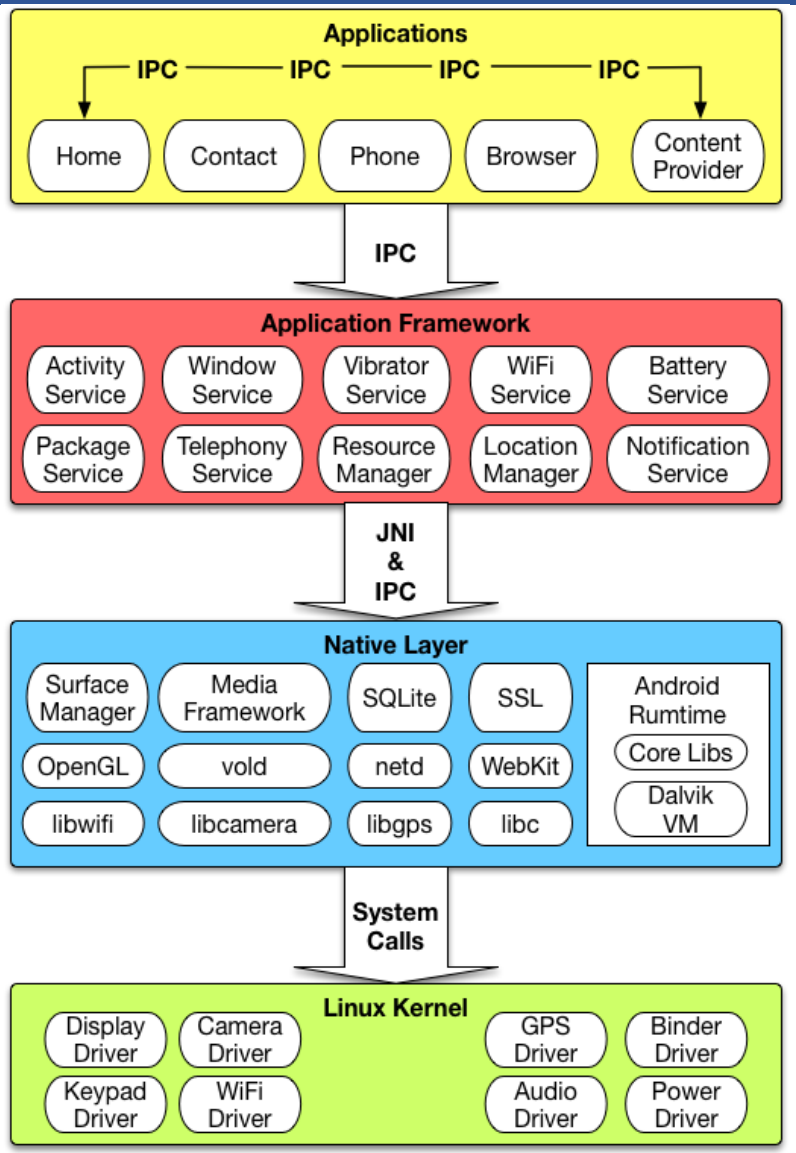
- **The main Android "OS" from app point of view**
  - Provide high-level services and environment to apps
  - Interact with low-level libraries and Linux kernel
- **Example**
  - Activity Manager
    - Manages the lifecycle of apps
  - Package Manager
    - Keeps track of apps installed
  - Power Manager
    - Wakelock APIs to apps

Android Binder IPC

- **An essential component in Android for Inter-Process Communication (IPC)**
  - Allows communication among apps, between system services, and between app and system service
- **Data sent through "parcels" in "transactions"**



IPC Is Pervasive in Android



How Is Binder Implemented: As RPC!

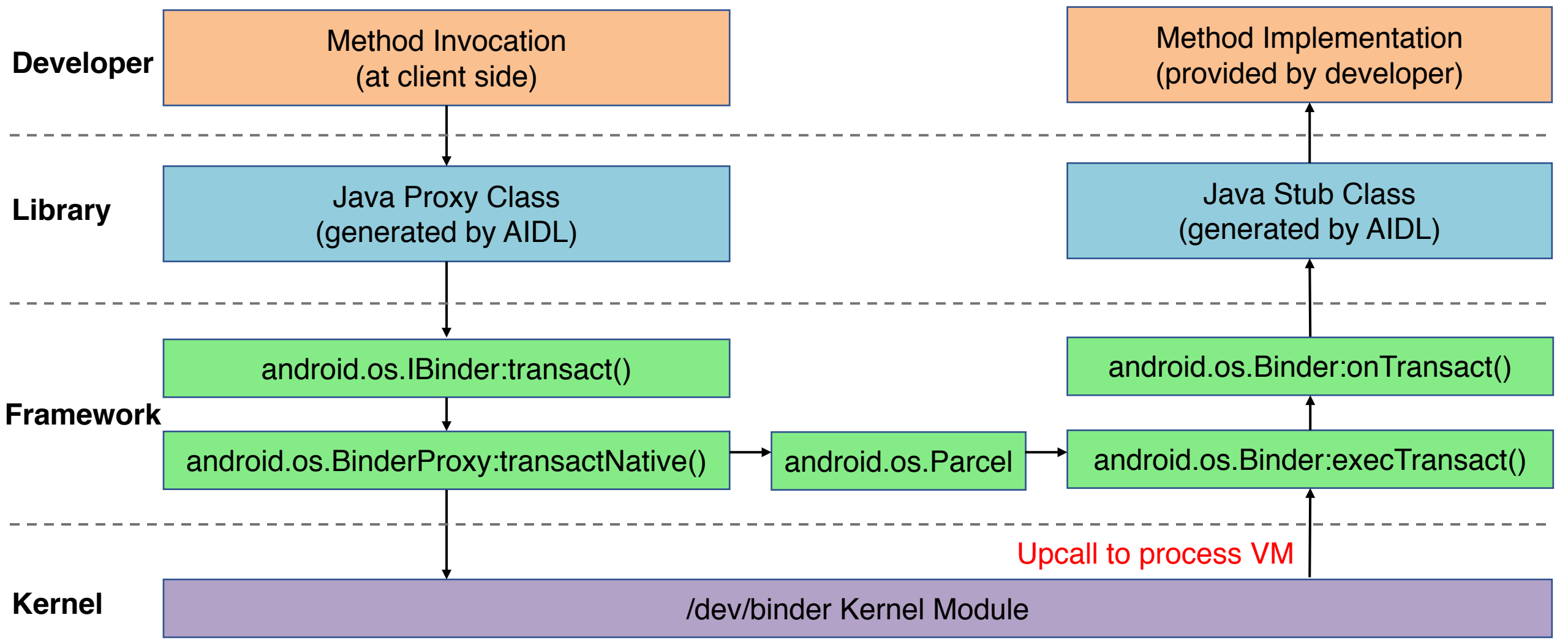
- **Developer defines methods and object interface in an .aidl file**

```
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);
}
```

- **Android SDK generate a stub Java file for the .aidl file**
  - Developer implements the stub methods
  - Expose the stub in a Service
- **Client copies the .aidl file to its source, Android SDK generates a stub (a.k.a proxy) for it as well**
  - Client invoke the RPC through the stub

Binder Information Flow



Passing Objects over IPC

- **Primitive types are automatically translated by the stub**
- **For complex object must let binder know how to serialize and de-serialize**
  - The object needs to implement Parcelable
  - Provides both write and read methods
    - E.g., writeInt, writeInt, writeLong, writeFloat

Some Other Interesting Topics in Mobile OS

- **Energy management**
  - ECOSystem: Managing Energy as a First Class Operating System Resource
  - Drowsy Power Management
- **Dealing with misbehaving apps**
  - DefDroid: Towards a More Defensive Mobile OS Against Disruptive App Behavior
  - eDoctor: Automatically Diagnosing Abnormal Battery Drain Issues on Smartphones
- **Security and safety**
  - CLKSCREW: Exposing the Perils of Security-Oblivious Energy Management
  - Multiprogramming a 64 kB Computer Safely and Efficiently

Summary

- **Smartphone becomes a ubiquitous computing device**
  - Long history but last 5 years have been disruptive
- **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 ≠ process
  - Multitasking
  - No swap space
  - Private storage
- **Android: a very complex ecosystem**

Next Time…

- **System Reliability**