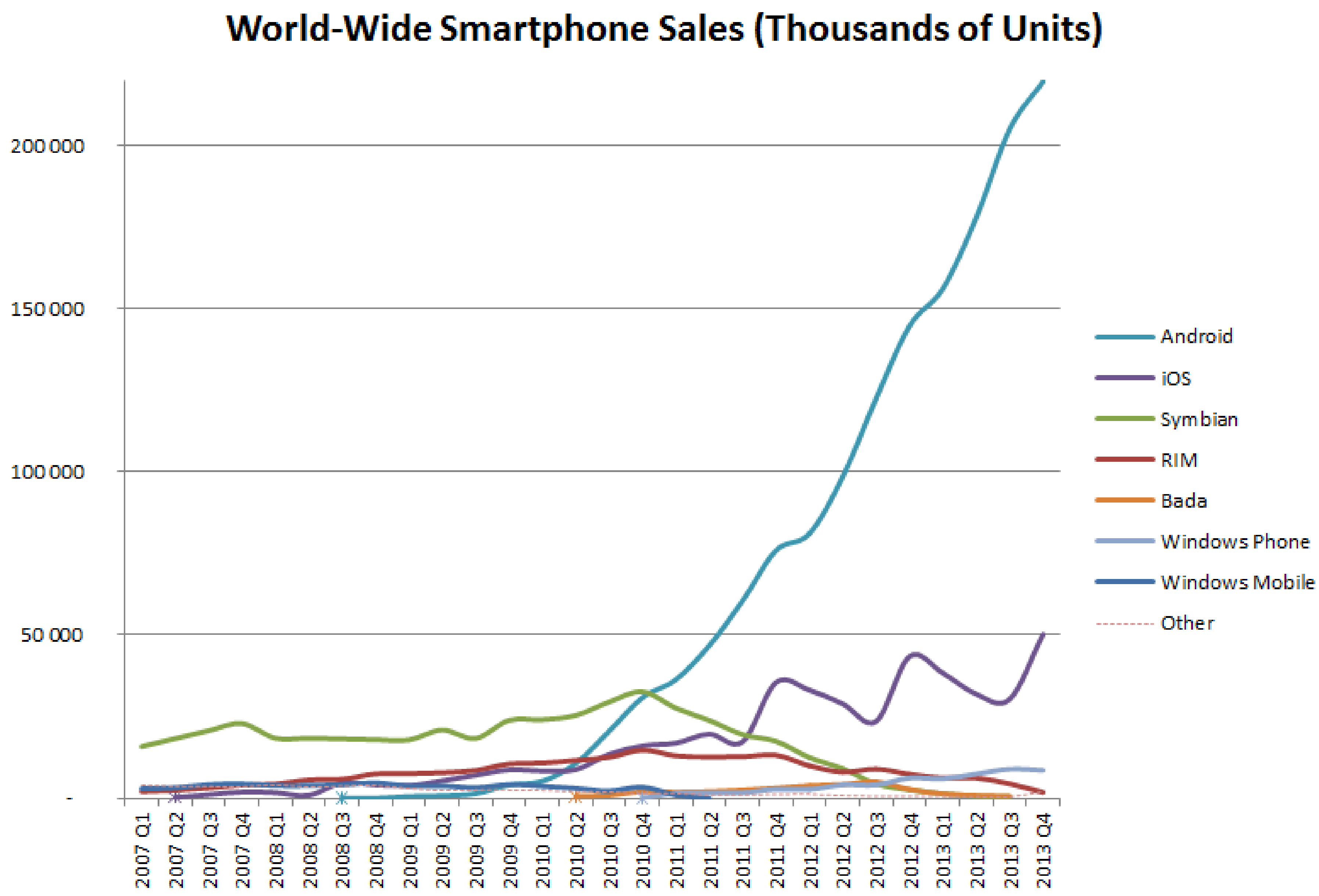
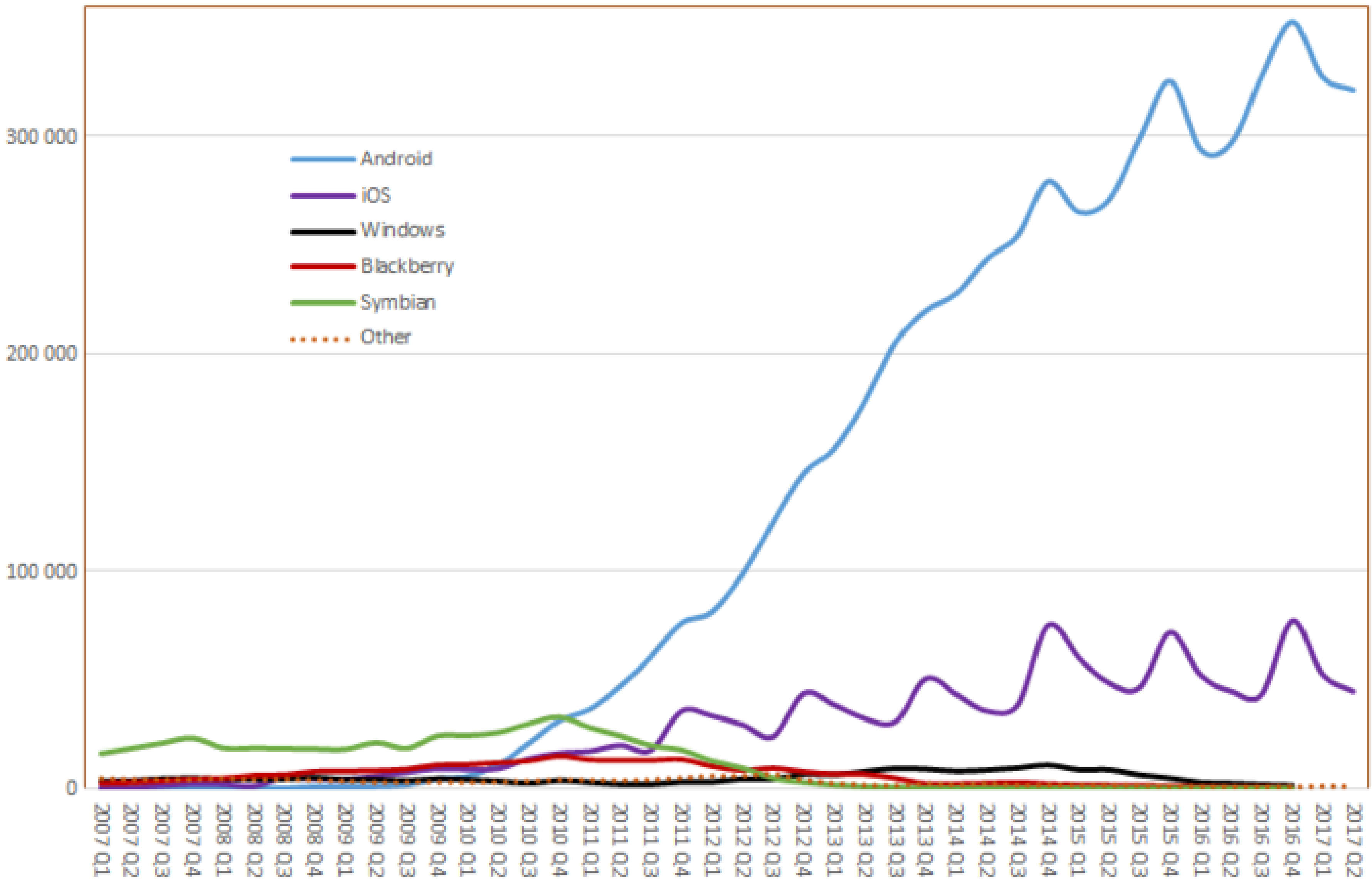
Introduction to Android

CS 436 Software Development on Mobile

By Dr.Paween Khoenkaw


World-Wide Smartphone Sales (Thousands of Units)
200 000
150 000
100 000
50 000
-
2007 Q1
2007 Q2
2007 Q3
2007 Q4
2008 Q1
2008 Q2
2008 Q3
2008 Q4
2009 Q1
2009 Q2
2009 Q3
2009 Q4
2010 Q1
2010 Q2
2010 Q3
2010 Q4
2011 Q1
2011 Q2
2011 Q3
2011 Q4
2012 Q1
2012 Q2
2012 Q3
2012 Q4
2013 Q1
2013 Q2
2013 Q3
2013 Q4
Android
iOS
Symbian
RIM
Bada
Windows Phone
Windows Mobile
Other

World-Wide Smartphone Sales (Thousands of Units)



What is Android ?

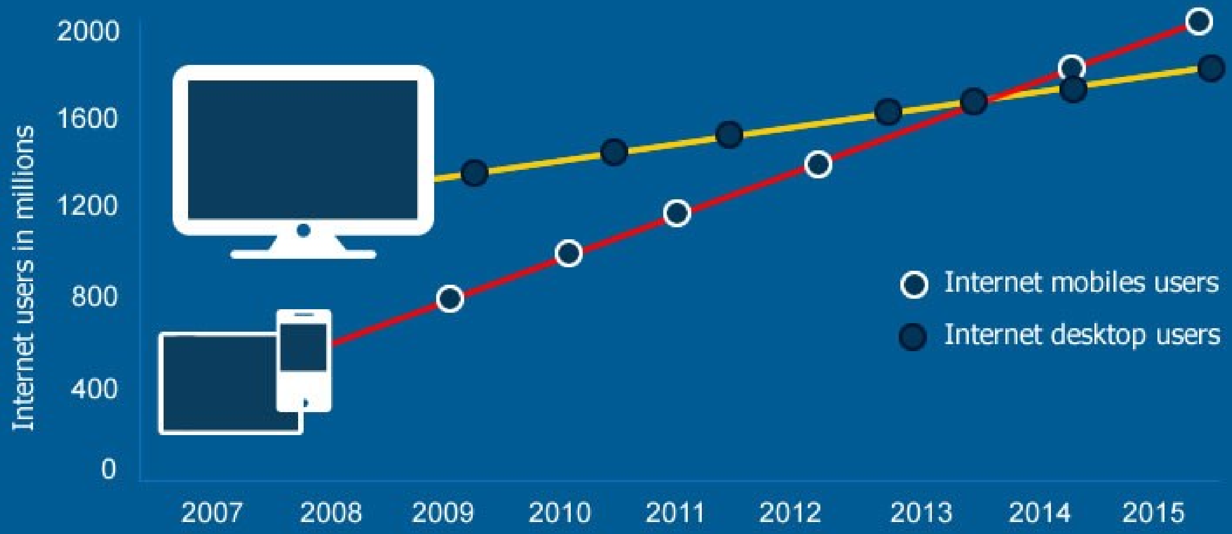
- Complete embedded operating system
- Cutting-edge mobile user experience
- Software stack for building application
- Open platform

Why Android was created ?

- Full phone software stack including application
- Designed as a platform for software development
- Open
- Free
- Community support
- Java Phone

Internet usage - Mobiles VS. Ordinateurs

The projection of global internet users conducted by Morgan Stanley Research: Mobiles VS. Computers from 2007 to 2015.



Android inc.

-Android, Inc. was founded in October 2003 by Andy Rubin.
-The early intentions were to develop an operating system for digital cameras.
-The company diverted its efforts toward producing a smartphone operating system.
-Operating system that would rival Symbian and Microsoft Windows Mobile

Andrew E. "Andy" Rubin

Handset Manufacturers

- Motorola
- HTC smart mobility
- Samsung
- LG Electronics

Mobile Operators

- China Mobile
- KDDI Designing The Future
- NTT DoCoMo
- Sprint
- Telecom Italia
- Telefónica
- T-Mobile

open handset alliance

Software

- Ascender Corporation
- eBay
- esmertec
- Google
- livewire
- Living Image
- Nuance
- PV
- SkyPoP
- SONiVOX sound that rocks.

Semiconductor

- Audience
- Broadcom
- Intel
- Marvell
- nVidia
- Qualcomm
- SiRF
- Synaptics
- Texas Instruments

Commercialization

- Aplix Corporation
- noser
- tat
- Wind River

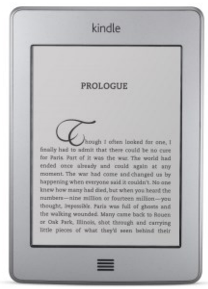
- **Software platform from Google and the Open Handset Alliance**
- **August 2005, Google acquired Android, Inc.**
- **November 2007, Open Handset Alliance formed to develop open standards for mobile devices**
- **October 2008, Android available as open source**
- **December 2008, 14 new members joined Android project**

Eric Schmidt, Andy Rubin, and Hugo Barra

Android Versions

- Android Open Source Project (AOSP)
  - Open Source

Firefox OS

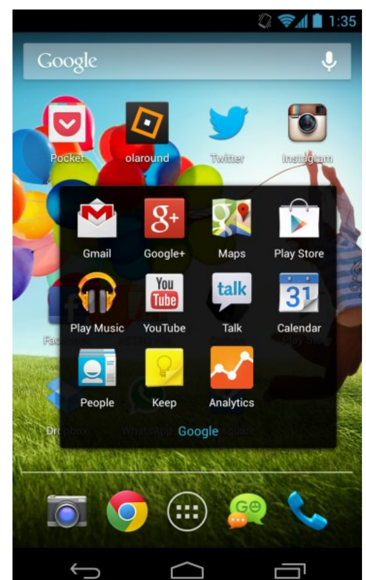


Kindle OS

LINEAGE

HarmonyOS

- Google Apps API
  - Closed Source
  - Google Play Service



Android version history

2008 Aug 18

[Android 0.9]

320x480 HVGA

2008 Sep 23

[Android 1.0]

Apple pie

API 1.0

320x480 HVGA

320x480 HVGA

2009 Feb 9

[Android 1.1]

Banana bread

API 2.0

320x480 HVGA

2009 Apr 30

[Android 1.5]

Cupcake

API 3.0

Bluetooth A2DP, AVRCP support

Soft-keyboard with text-prediction

Record/watch videos

320x480 HVGA

Android version history

2009 Sep 15

Android 1.6
Donut
API 4.0
Gesture framework
Turn-by-turn navigation
800×480 WVGA

2009 Oct 26

Android 2.0
Éclair
API 5.0
Digital zoom
Live Wallpapers
Updated UI
800×480 WVGA

2010 May 20

Android 2.2
Froyo
API 8.0
Flash 10.1
JIT implementation
USB Tethering
Applications installation to the expandable memory
Upload file support in the browser
Animated GIFs
800×480 WVGA

Android version history

2010 Dec 6

Android 2.3

Gingerbread

API 9.0

Near Field Communication support

Native VoIP/SIP support

Video call support

1366×768 WXGA

2011 Feb 22

Android 3.0

Honeycomb

API 11.0

Multi core support

Better tablet support

Updated 3D UI

"Private browsing“

Open Accessory API

USB host API

Mice, joysticks, gamepads... support

Android version history

2011 Oct 19

Android 4.0

Ice Cream Sandwich

API 14.0

Facial recognition (Face Unlock)

UI use Hardware acceleration

Web browser, allows up to 16 tabs

Updated launcher

Android Beam exchange data through NFC

Resizable widgets

Video stabilization

GoogleNow

Android 4.1

Jelly Bean

Triple buffering in the graphics pipeline

Extends vsync timing across all drawing and animation

CPU input boost

Bi-Directional Text and Other Language

Support Android Beam

Google Cloud Messaging for Android

App Encryption

Smart App Updates

Android version history

13 November 2012

<u>Android 4.2</u>

Jelly Bean

Multiple user accounts

Support for wireless display

24 July 2013

<u>Android 4.3</u>

Jelly Bean

4K resolution

Android version history

3 September 2013

17 October 2014

Android 4.4

Kitkat

Can run on low-end devices

NFC host card emulation

Android 5.0

Lolipop

New design (Material design)

Speed improvement

Battery consumption improvement

Android version history

5 October 2015

22 August 2016

Android 6
Marshmallow

USB Type-C support
Fingerprint Authentication support
Better battery life with "deep sleep"
Permissions dashboard
Android Pay
MIDI support

Android 7
Nougat

Unicode 9.0 emoji
Better multitasking
Multi-window mode (PIP, Freeform window)
Seamless system updates (with dual system partition)
Better performance and code size thanks to new JIT Compiler

Android version history

21 August 2017

December 5, 201

Android 8.0

Oreo

Multi-display support
2 times faster boot time
Downloadable fonts
Integrated printing support

Android 8.1

Neuron Network API
Shared Memory
Android Oreo Go Edition

Android version history

May 08, 2018

Android 9.0

Pie

New user interface for the quick settings menu
Improved Adaptive Brightness feature
A new gesture-based system interface
A new "Lockdown" mode which disables biometric authentication once activated.

Android version history

September 3, 2019

Android 10 (API 29)

- New permissions to access location in background and to access photo, video and audio files.
- Support for the AV1 video codec, the HDR10+ video format and the Opus audio codec.
- A native MIDI API, allowing interaction with music controllers.
- Support for the WPA3 Wi-Fi security protocol.
- Support for foldable phones.

Android Handsets

HTC G1

Samsung i7500

HTC Hero

Motorola Cliq

Sony X10

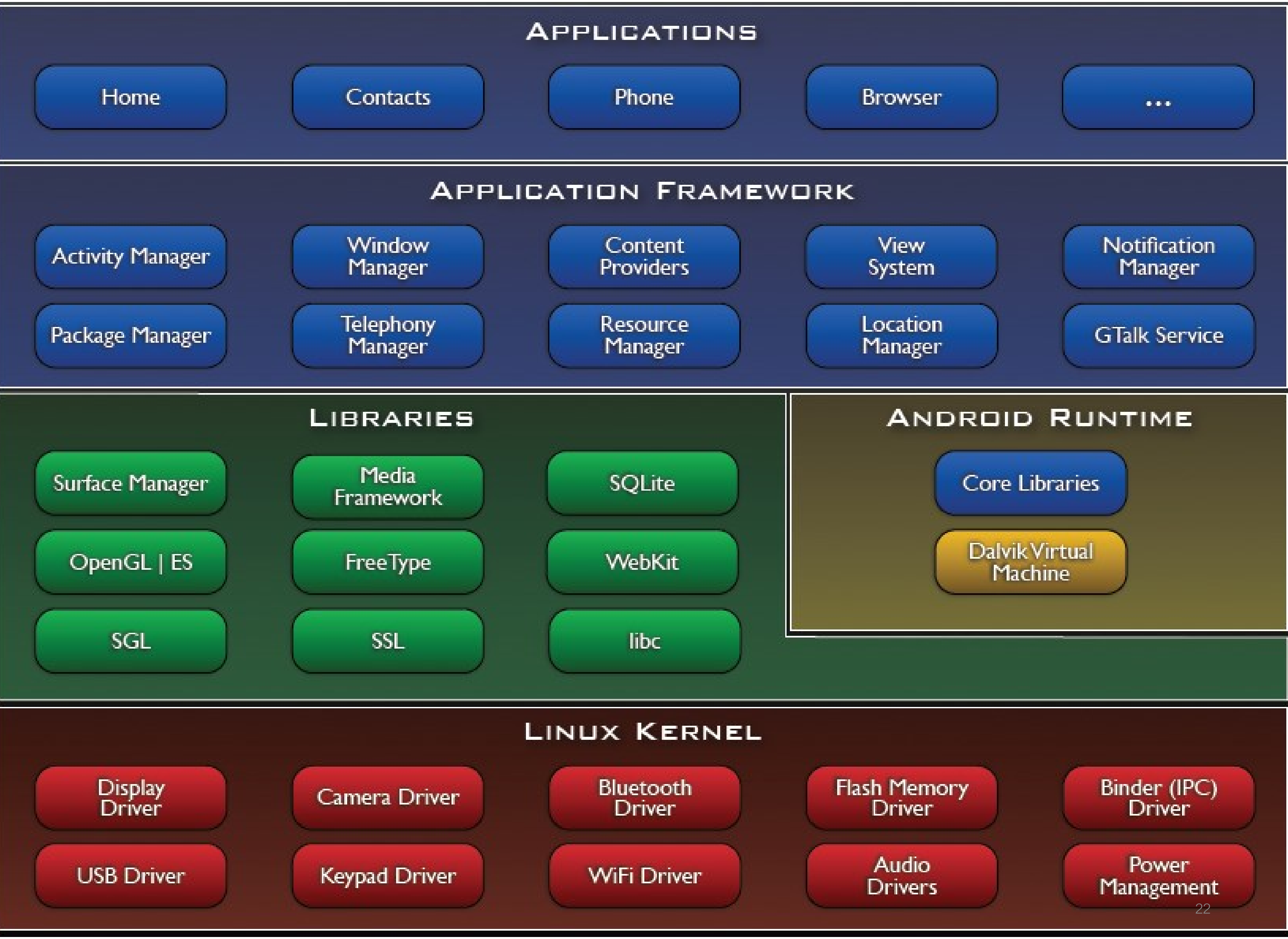
HTC Magic

Samsung Moment

Motorola Droid

HTC Tattoo

nexus one

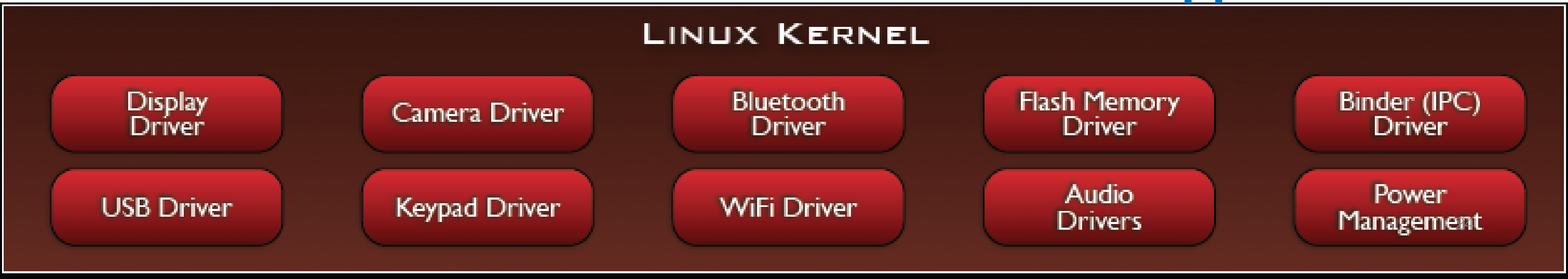

Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
GTalk Service
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

The Kernel

Why Linux kernel ?

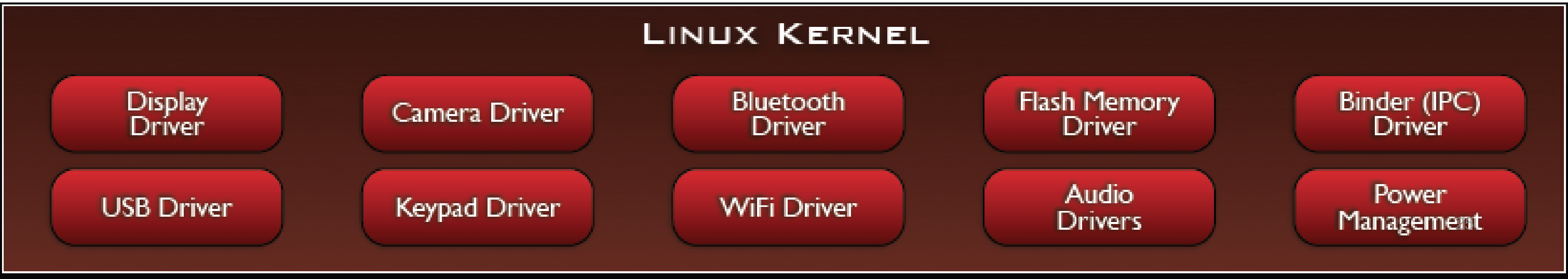
- Great memory and process management
- Permissions-based security model
- Proven driver model
- Support for shared libraries
- It's already open source!

- Standard Linux 2.6.24 Kernel
- Patch of "kernel enhancements" to support Android



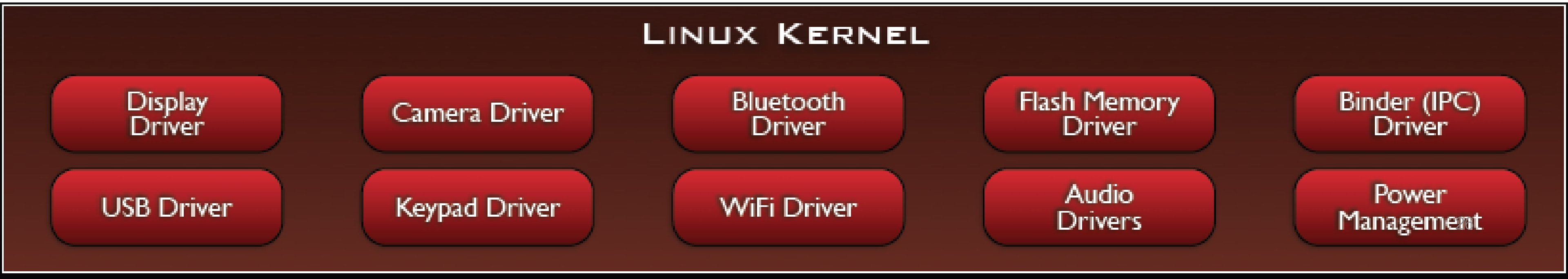
Android is not Linux ！

- Android is built on the Linux kernel, but Android is not Linux
- No native windowing system
- No glibc support
- Does not include the full set of standard Linux utilities



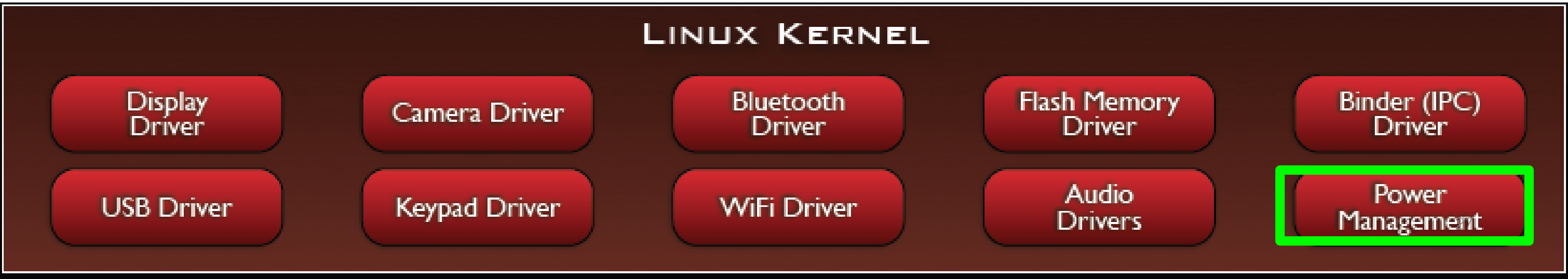
Kernel Enhancements

- Alarm
- Ashmem
- Binder
- Power Management
- Low Memory Killer
- Kernel Debugger
- Logger



Power Management

- Built on top of standard Linux Power Management (PM)
- More aggressive power management policy
- Components make requests to keep the power on through *“wake locks”*
- Supports different types of wake locks



Libraries

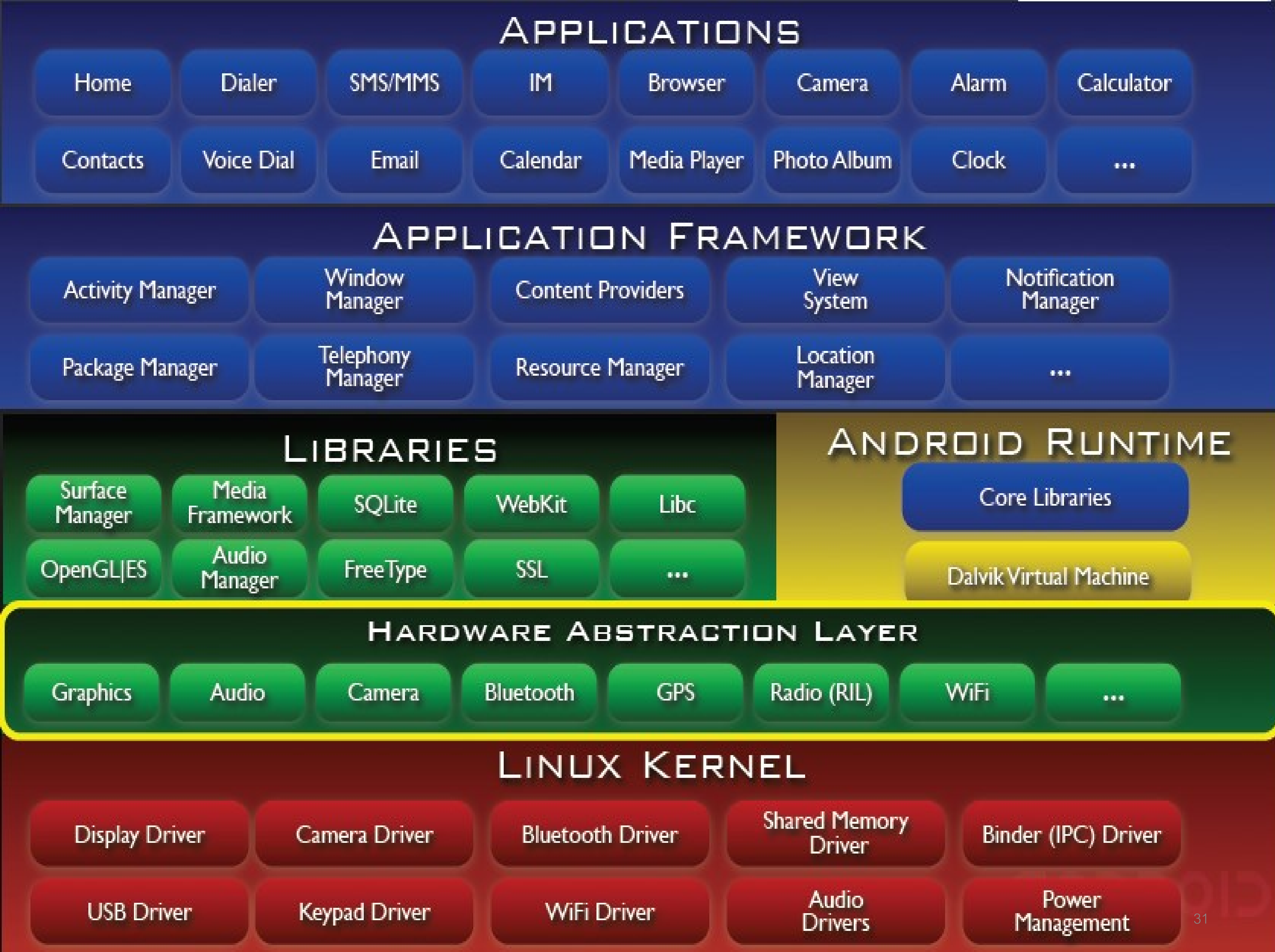
Libraries

- C/C++
- Draw Pixel
- Multimedia / Codec
- Communication
- Database
- Browser
- Fonts

Why android using BSD Libc ?

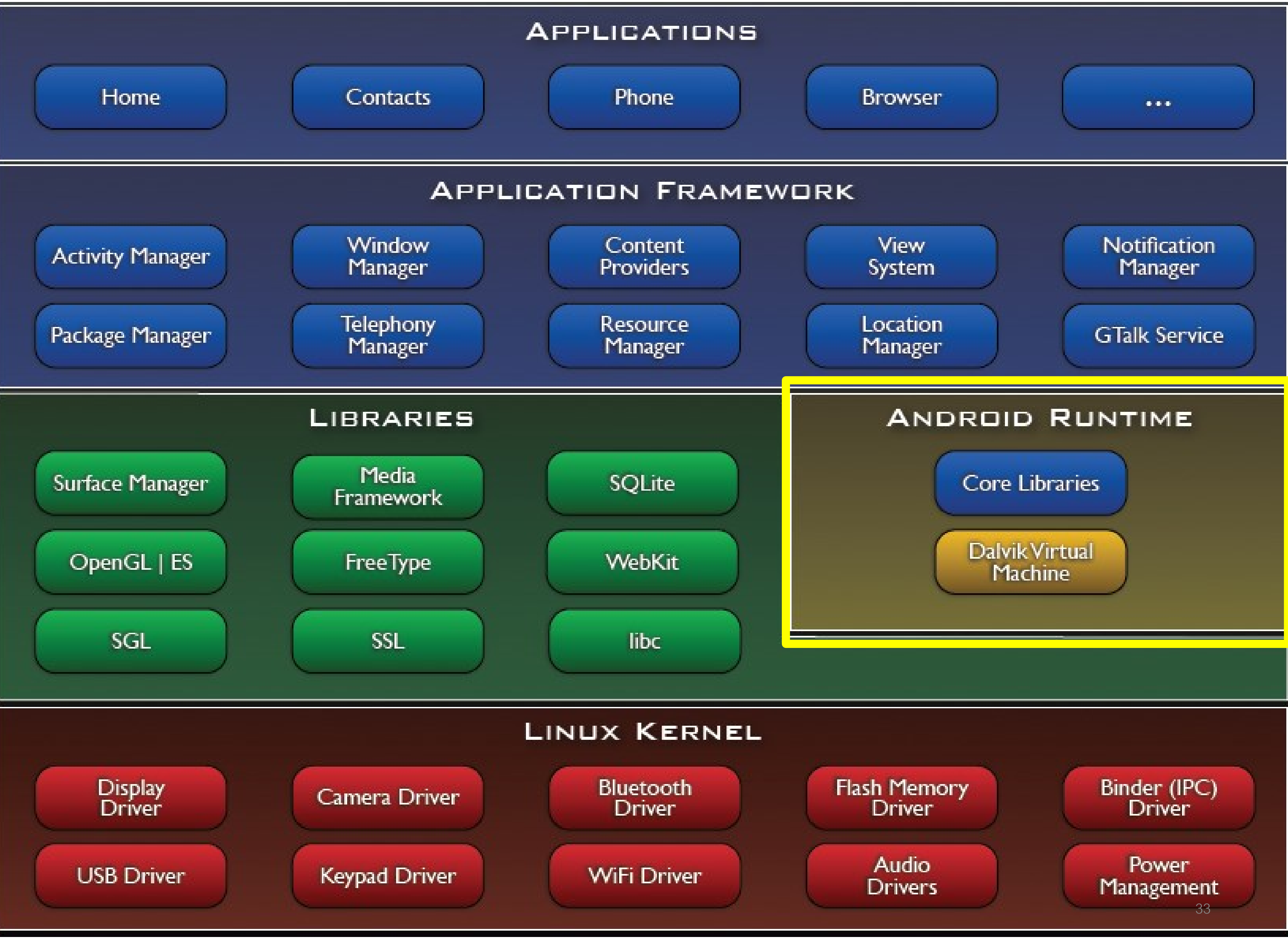
- License: we want to keep GPL out of user-space
- Size: will load in each process, so it needs to be small
- Fast: limited CPU power means we need to be fast

X Doesn't support certain POSIX features
X Not compatible with Gnu Libc (glibc)
X All native code must be compiled against bionic


Applications
Home
Dialer
SMS/MMS
IM
Browser
Camera
Alarm
Calculator
Contacts
Voice Dial
Email
Calendar
Media Player
Photo Album
Clock
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
...
Libraries
Surface Manager
Media Framework
SQLite
WebKit
Libc
OpenGL|ES
Audio Manager
FreeType
SSL
...
Android Runtime
Core Libraries
Dalvik Virtual Machine
Hardware Abstraction Layer
Graphics
Audio
Camera
Bluetooth
GPS
Radio (RIL)
WiFi
...
Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Shared Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Why do we need a user-space HAL?

- Not all components have standardized kernel driver interfaces
- Kernel drivers are GPL which exposes any proprietary IP
- Android has specific requirements for hardware drivers


Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
GTalk Service
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
Android Runtime
Core Libraries
Dalvik Virtual Machine
Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Android Runtime
Dalvik Virtual Machine





Designed for embedded environment

- Supports multiple virtual machine processes per device
- Highly CPU-optimized bytecode interpreter
- Uses runtime memory very efficiently

Application runs in sand box

```
public static long sumArray(int[] arr) {
    long sum = 0;
    for (int i : arr) {
        sum += i;
    }
    return sum;
}
```

JavaVM

```
0000: lconst_0
0001: lstore_1
0002: aload_0
0003: astore_3
0004: aload_3
0005: arraylength
0006: istore 04
0008: iconst_0
0009: istore 05
000b: iload 05          // rl ws
000d: iload 04          // rl ws
000f: if_icmpge 0024    // rs rs
0012: aload_3           // rl ws
0013: iload 05          // rl ws
0015: iaload            // rs rs ws
0016: istore 06         // rs wl
0018: lload_1           // rl rl ws ws
0019: iload 06          // rl ws
001b: i2l               // rs ws ws
001c: ladd              // rs rs rs rs ws ws
001d: lstore_1          // rs rs wl wl
001e: iinc 05, #+01     // rl wl
0021: goto 000b
0024: lload_1
0025: lreturn
```

- **25 bytes**
- **14 dispatches**
- **45 reads**
- **16 writes**

Rs=read stack
Rw=write stack
Rl=read local
Wl=write local

Virtual Machine Showdown: Stack Versus Registers

Yunhe Shi, David Gregg, Andrew Beatty
Department of Computer Science
University of Dublin, Trinity College
Dublin 2, Ireland
{yshi, David.Gregg, Andrew.Beatty}@cs.tcd.ie

M. Anton Ertl
Institut für Computersprachen
TU Wien
Argentinierstraße 8
A-1040 Wien, Austria
anton@complang.tuwien.ac.at

ABSTRACT

Virtual machines (VMs) are commonly used to distribute programs in an architecture-neutral format, which can easily be interpreted or compiled. A long-running question in the design of VMs is whether stack architecture or register architecture can be implemented more efficiently with an interpreter. We extend existing work on comparing virtual stack and virtual register architectures in two ways. Firstly, our translation from stack to register code is much more sophisticated. The result is that we eliminate an average of more than 47% of executed VM instructions, with the register machine bytecode size only 25% larger than that of the corresponding stack bytecode. Secondly we present an implementation of a register machine in a fully standard-compliant implementation of the Java VM. We find that, on the Pentium 4, the register architecture requires an average of 32.3% less time to execute standard benchmarks if dispatch is performed using a C switch statement. Even if more efficient threaded dispatch is available (which requires labels as first class values), the reduction in running time is still approximately 26.5% for the register architecture.

Categories and Subject Descriptors

D.3 [**Software**]: Programming Language; D.3.4 [**Programming Language**]: Processor—*Interpreter*

General Terms

Performance, Language

Keywords

Interpreter, Virtual Machine, Register Architecture, Stack Architecture

1. MOTIVATION

Virtual machines (VMs) are commonly used to distribute programs in an architecture-neutral format, which can easily be interpreted or compiled. The most popular VMs, such as the Java VM, use a virtual stack architecture, rather than the register architecture that dominates in real processors.

A long-running question in the design of VMs is whether stack architecture or register architecture can be implemented more efficiently with an interpreter. On the one hand stack architectures allow smaller VM code so less code must be fetched per VM instruction executed. On the other hand, stack machines require more VM instructions for a given computation, each of which requires an expensive (usually unpredictable) indirect branch for VM instruction dispatch. Several authors have discussed the issue [12, 15, 11, 16] and presented small examples where each architecture performs better, but no general conclusions can be drawn without a larger study.

The first large-scale quantitative results on this question were presented by Davis et al. [5, 10] who translated Java VM stack code to a corresponding register machine code. A straightforward translation strategy was used, with simple compiler optimizations to eliminate instructions which become unnecessary in register format. The resulting register code required around 35% fewer executed VM instructions to perform the same computation than the stack architecture. However, the resulting register VM code was around 45% larger than the original stack code and resulted in a similar increase in bytecodes fetched. Given the high cost of unpredictable indirect branches, these results strongly suggest that register VMs can be implemented more efficiently than stack VMs using an interpreter. However, Davis et al's work did not include an implementation of the virtual register architecture, so no real running times could be presented.

This paper extends the work of Davis et al. in two respects. First, our translation from stack to register code is much more sophisticated. We use a more aggressive copy propagation approach to eliminate almost all of the stack load and store VM instructions. We also optimize constant load instructions, to eliminate common constant loads and move constant loads out of loops. The result is that an average of more than 47% of executed VM instructions are eliminated. The resulting register VM code is roughly 25% larger than the original stack code, compared with 45% for Davis et al. We find that the increased cost of fetching more VM code involves only 1.07 extra real machine loads per VM instruction eliminated. Given that VM dispatches are much more expensive than real machine loads, this indicates strongly that register VM code is likely to be much

VEE'05, June 11-12, 2005, Chicago, Illinois, USA.

47% of instructions were eliminate

Bytecode size only 25% increase

ACM VEE'05, June 11-12, 2005

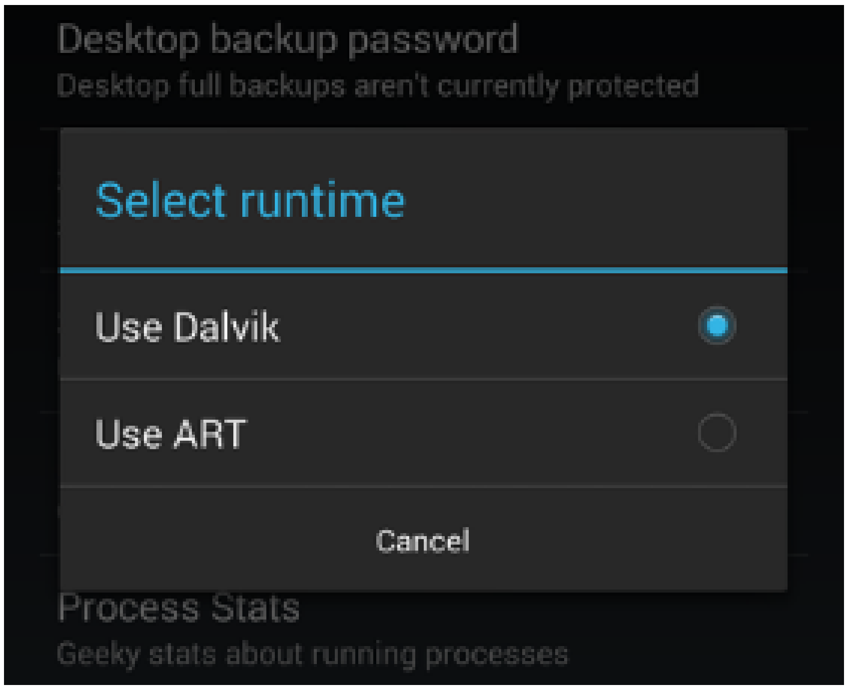
```
0000: const-wide/16 v0, #long 0
  0002: array-length v2, v8
  0003: const/4 v3, #int 0
  0004: move v7, v3
  0005: move-wide v3, v0
  0006: move v0, v7
  0007: if-ge v0, v2, 0010          // r r
  0009: aget v1, v8, v0             // r r w
  000b: int-to-long v5, v1          // r w w
  000c: add-long/2addr v3, v5       // r r r r w w
  000d: add-int/lit8 v0, v0, #int 1   // r w
  000f: goto 0007
  0010: return-wide v3
```

```java
public static long sumArray(int[] arr) {
    long sum = 0;
    for (int i : arr) {
        sum += i;
    }
    return sum;
}
```

DalVikVM

- **18 bytes**
- **6 dispatches**
- **19 reads**
- **6 writes**

ART vs Dalvik



Dalvik is based on **JIT (just in time) compilation**.

ART, on the other hand, compiles the intermediate language, Dalvik bytecode, into a **system-dependent binary**.

Application Framework & Toolkit

Toolkit for android application

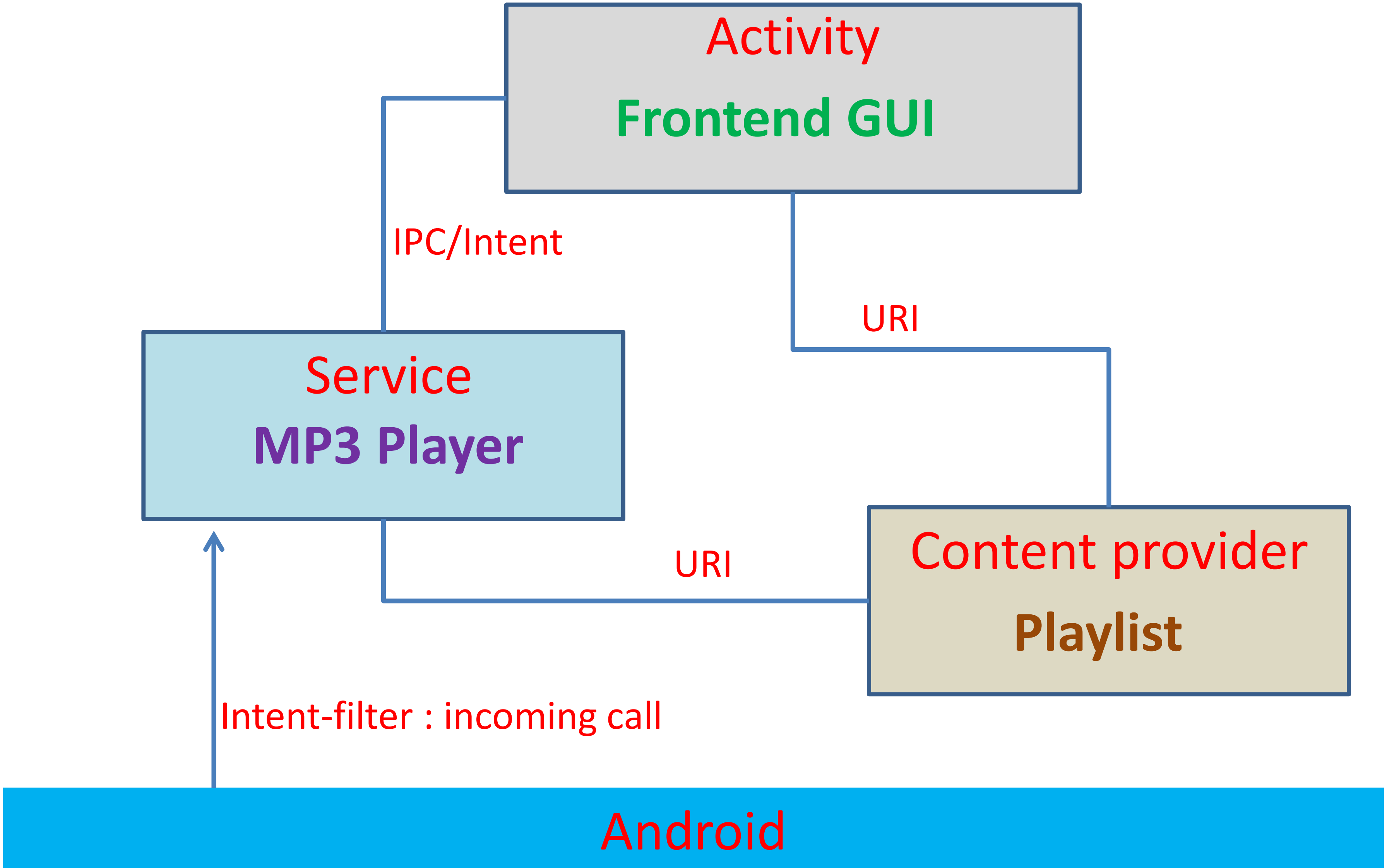
- Activity Manager
- Resource Manager
- Content Provider
- Notification Manager

Android Application

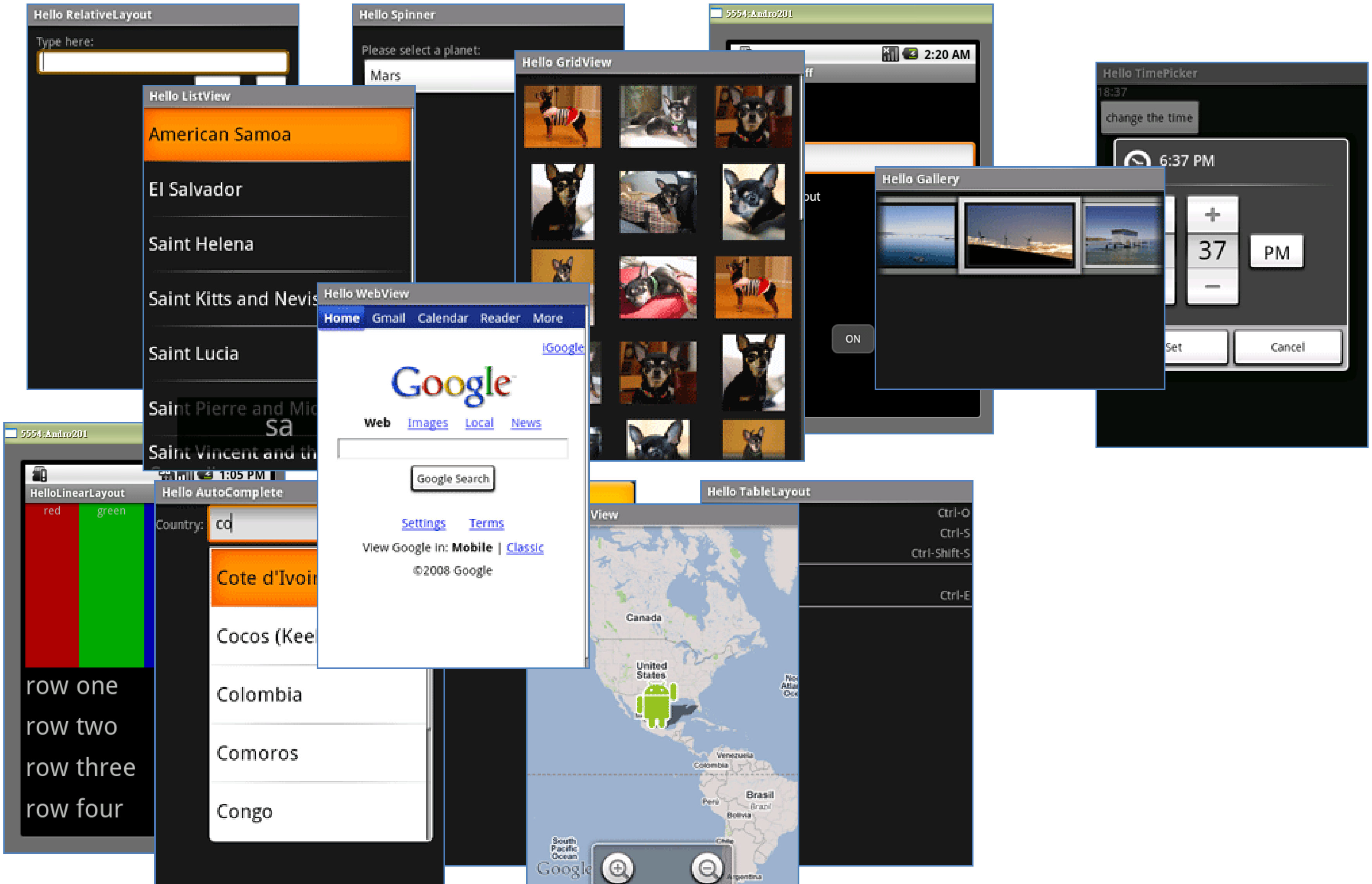
Application Building Blocks

Activity :UI component typically corresponding to one screen.
IntentReceiver: Set and respond to notifications or status changes. Can wake up your app.
Service: Faceless task that runs in the background.
ContentProvider: Enable applications to share data.

Android application : Music Player



User Interface

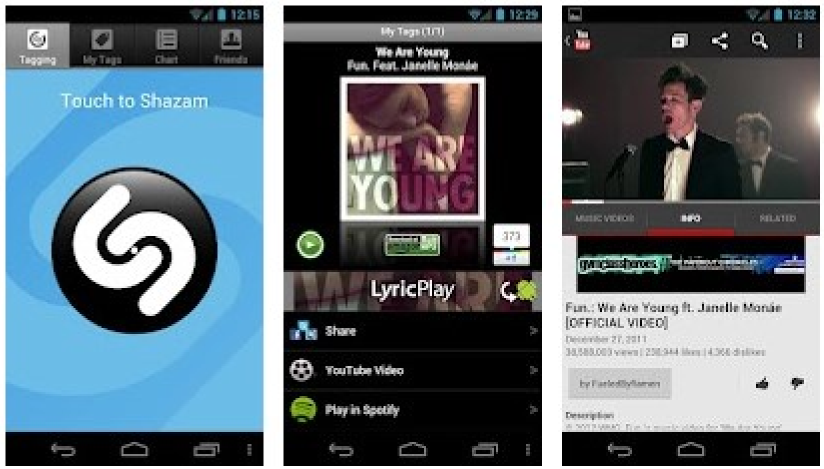


Android Apps

Sleep as android

Run keeper

Android Apps



Shazam

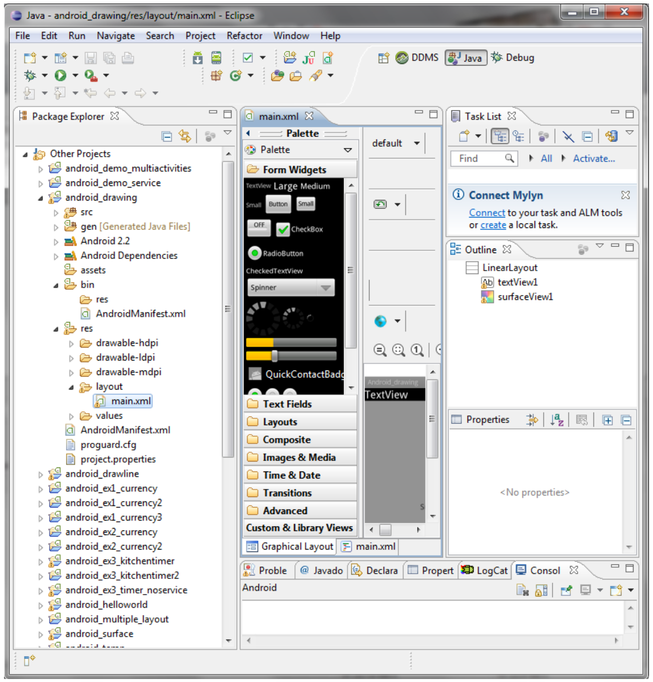
gStrings

Android Apps

Google skymap

Games

Android software development



-J2se JDK

-Android SDK

  -SDK Platform

  -SDK Platform tools

  -SDK Tools

  -Emulator & Images

  -Example

-Eclipse IDE

  -ADT Plug-in

-Android Studio

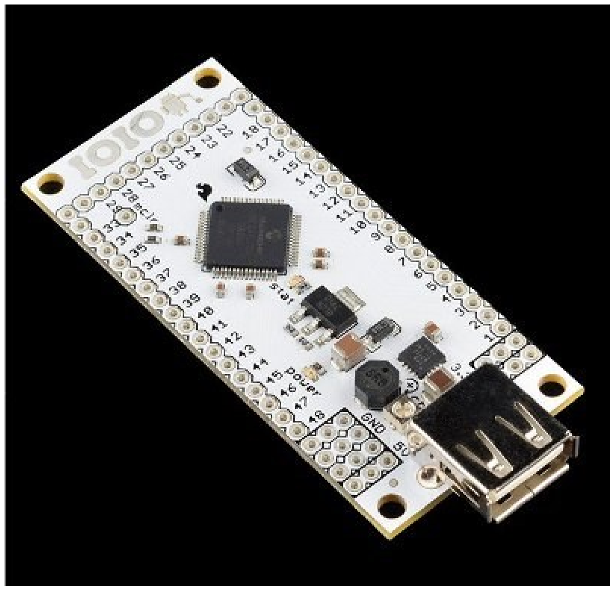
-ADB USB Driver

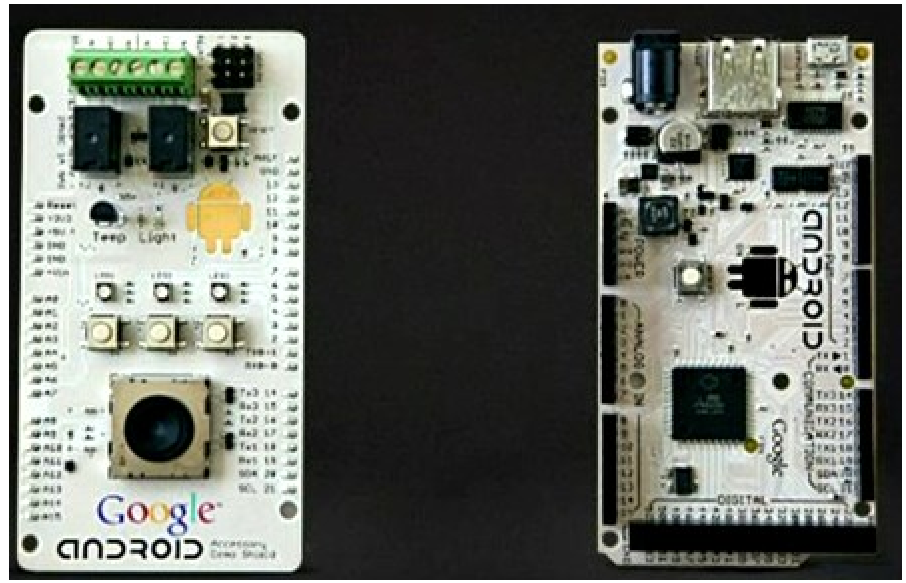
-Internet connection for online install

Emulator limitation

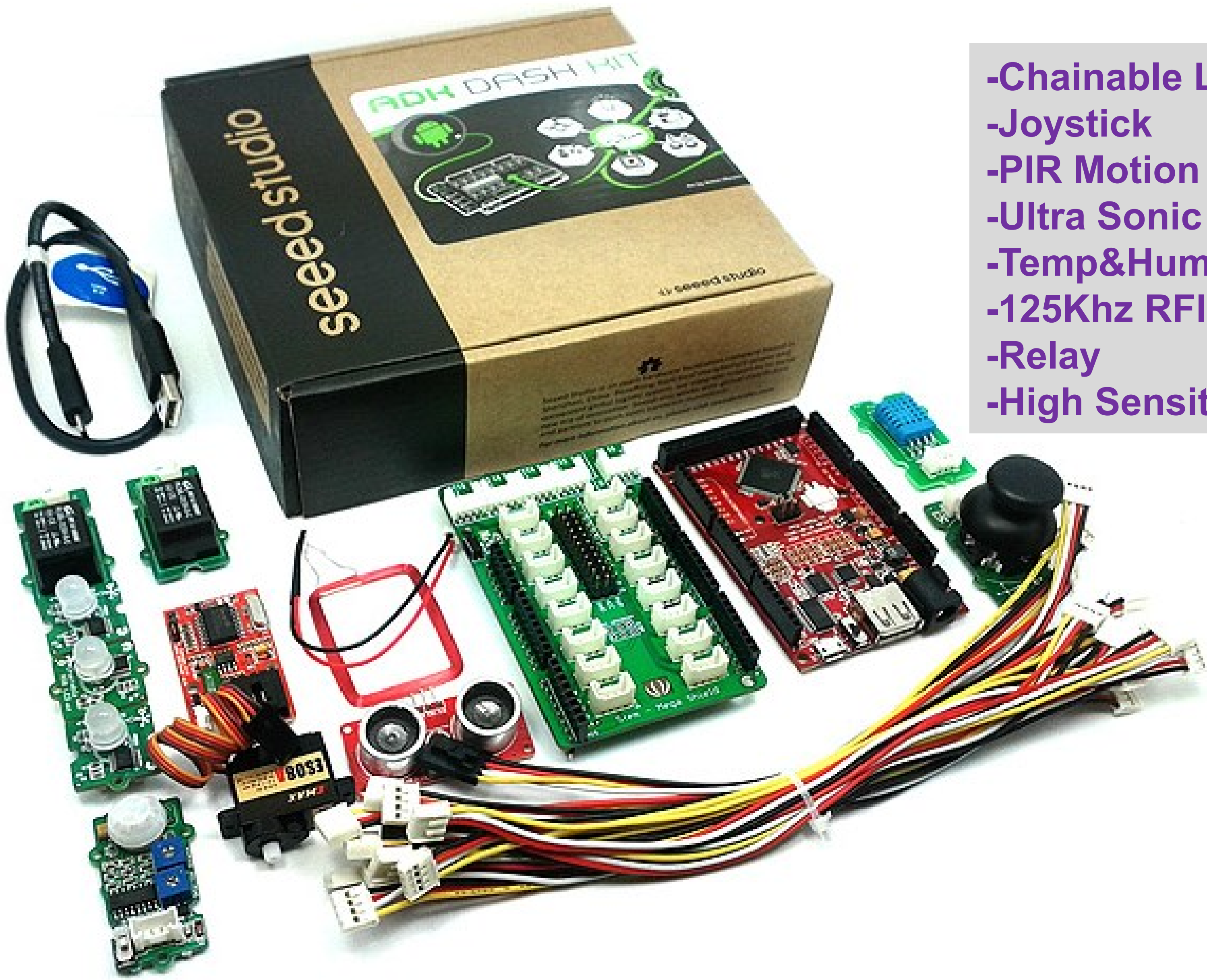
No support for placing or receiving actual phone calls.
No support for USB connections
No support for camera/video capture (input).
No support for device-attached headphones
No support for determining connected state
No support for determining battery charge level
No support for determining SD card insertion/removal
No support for Bluetooth
No support for Multitouch

Android Open Accessory Development Kit (ADK)





IOIO for Android



- -Chainable LED
- -Joystick
- -PIR Motion Sensor
- -Ultra Sonic range finder
- -Temp&Humi Sensor
- -125Khz RFID Card Reader
- -Relay
- -High Sensitive Mini Servo

QA