Android Architecture

Overview

- History of Android Architecture
- Five Layers
  - Linux Kernel
  - Android Runtime
  - Libraries
  - Application Framework
  - Applications
- Summary

History

- 2003 –Founded
  - No product for two years, funded by Andy Rubin
    - Planned the next generation of smartphones
    - **Open source** evolution of "Danger"
- 2005 – Purchased by Google
  - Sooner or G1?
- 2007 – Publically announced
- 2008 – Sold first phone

G1

Previous Versions

Lollipop
09/2014

Marshmallow
09/2014

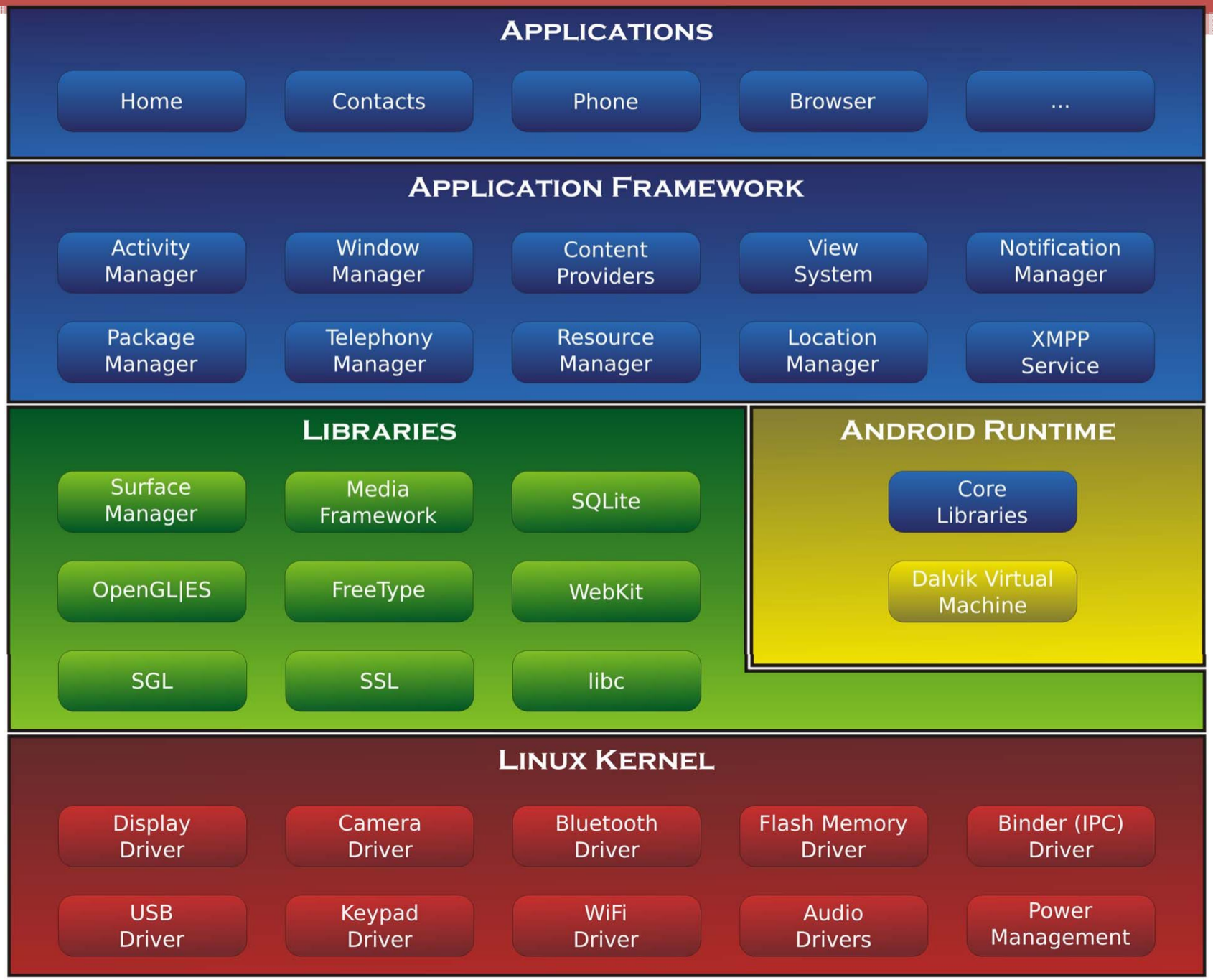
Previous Versions

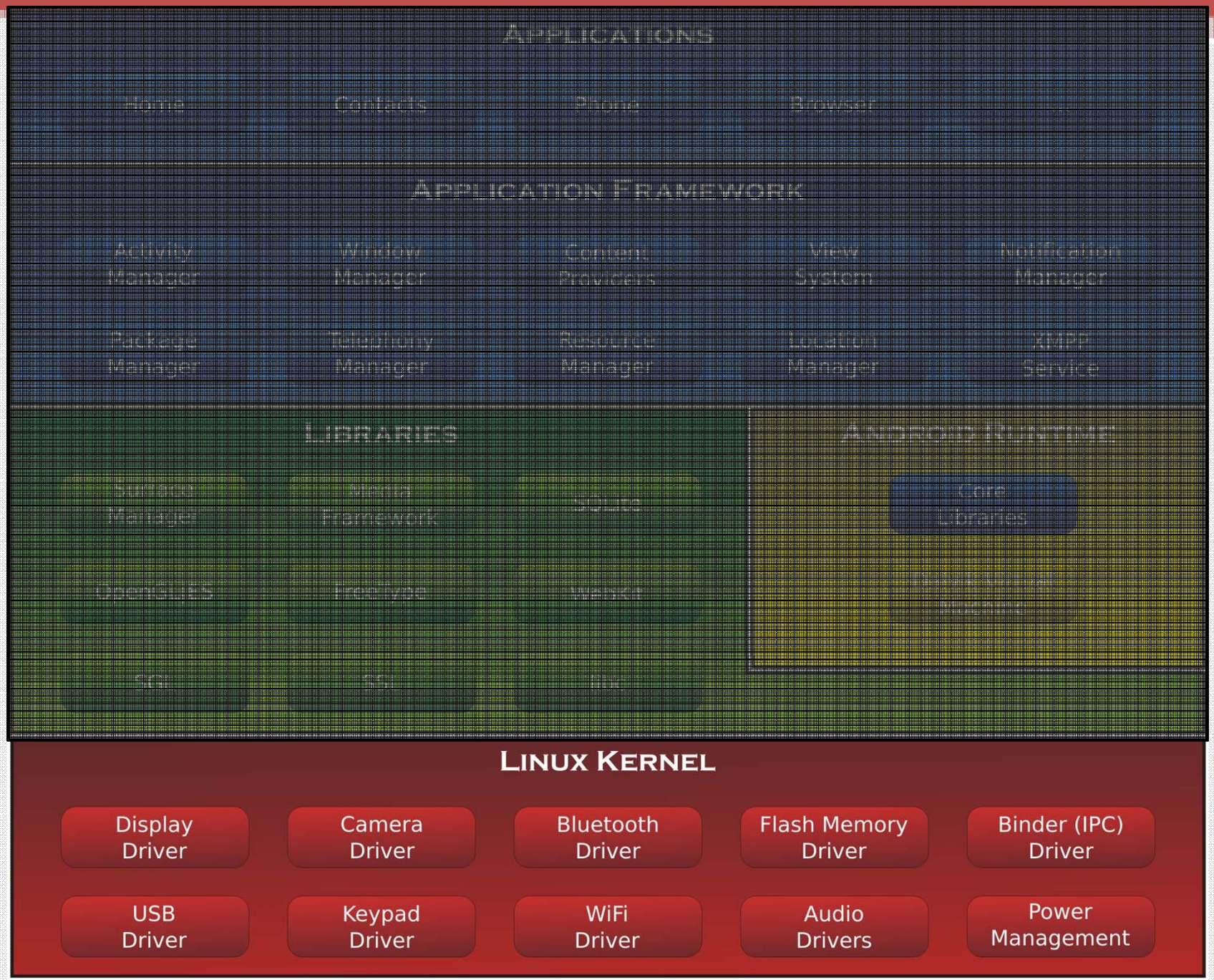
Cupcake

Ice Cream Sandwich

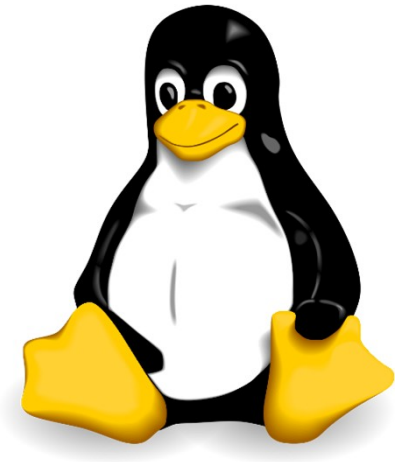
Lollipop

- Unnamed (1.0 + 1.1)
- **C**upcake (1.5)
- **D**onut (1.6) – Quick Search Box
- **É**clair (2.1) – High Density Displays, Traffic + Navigation
- **F**royo (2.2) – Voice Control, Hotspot, Speed
- **G**ingerbread (2.3) – Simpler, Battery Life, More apps
- **H**oneycomb (3.0) – Flexible interface, tablets
- **I**ce Cream Sandwich (4.0) - Customization
- **J**elly Bean (4.1) – Google Now, actionable notifications
- **K**itKat (4.4) – “Ok Google”, voice control variety
- **L**ollipop (5.0) – fluid tactile screens
- **M**arshmallow (6.0) – battery life, app permissions, UI


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
XMPP 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


Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

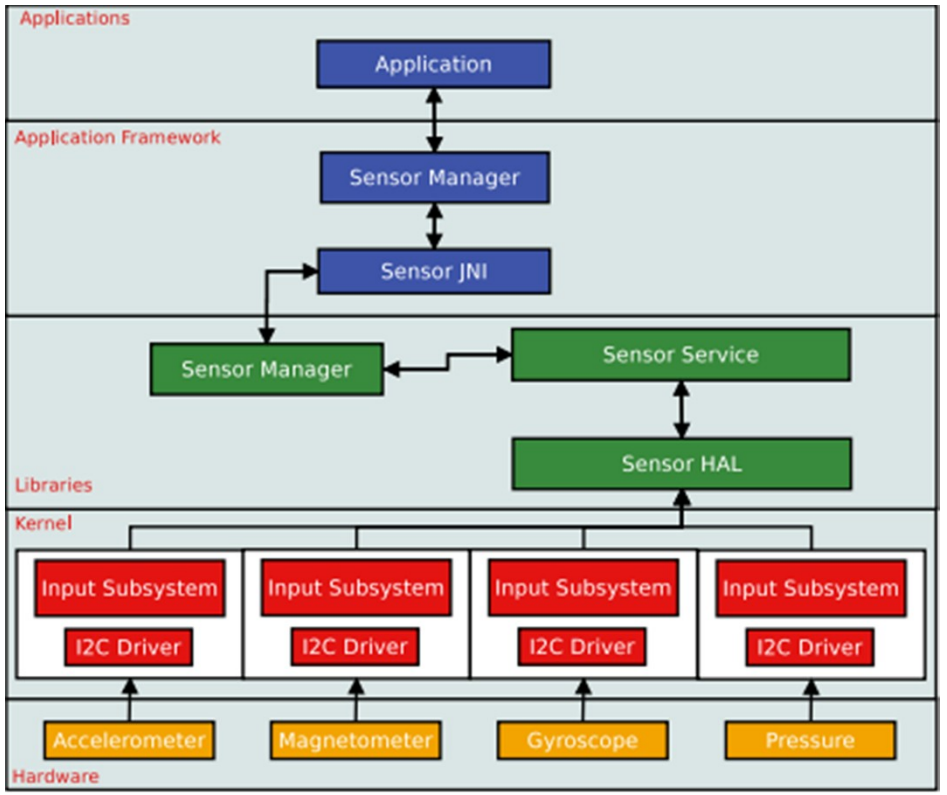
Linux Kernel



- 3.6 with ~115 patches
- Generic System Services
  - Permissions
  - Memory and Process management
  - File & Network I/O
  - Device Drivers
- Preemptive Multitasking
- Lean, efficient, and secure
- Open Source

Hardware Abstraction Layer (HAL)

- Software hooks between stack and hardware
- Hardware Specific
  - Allows Applications to be hardware ignorant



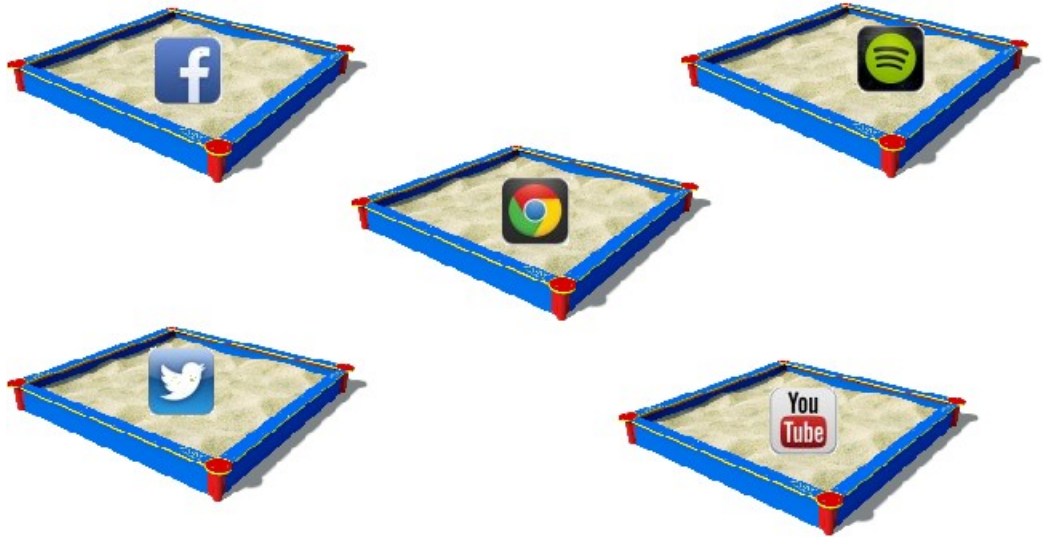

Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service
Libraries
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
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
- Core Java libraries
  - Specific to Android development
    - Apple: Swift (Objective C)
    - Windows: Visual C++ (C++), Changes with OS
  - Wrappers around C/C++ libraries
- ART (Android Runtime VM)
  - Replaced Dalvik in Lollipop (Android 5.0)
  - Advantages over Dalvik
    - AOT (Ahead of Time) Compilation
    - Improved Garbage Collection

Dalvik Virtual Machine

- Executes Android Applications
  - Each Application runs within its own VM
    - Each app is “sandboxed”
- Memory Management
- Multi-threading




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
XMPP 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

Libraries

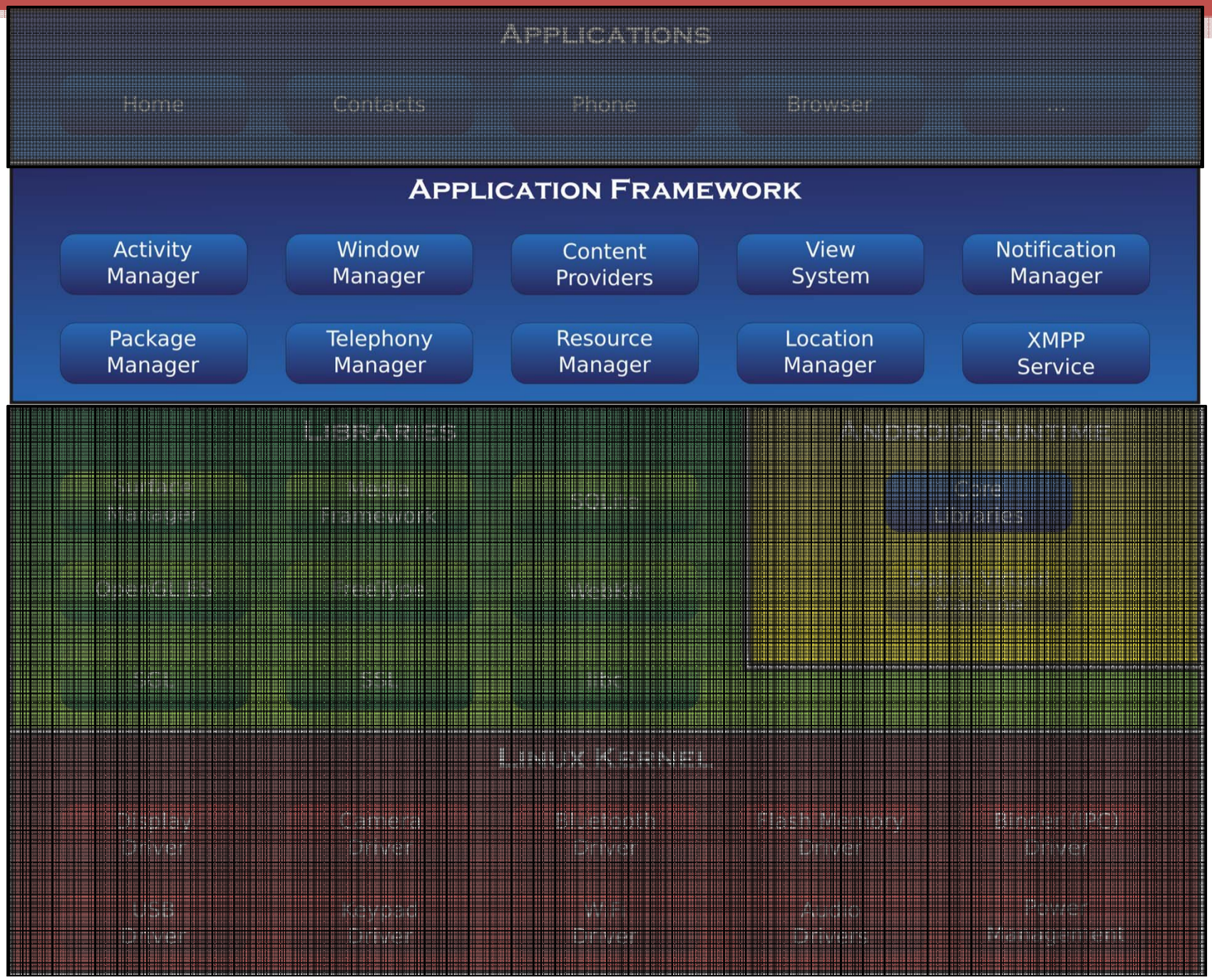
- C/C++
- Play and record audio and video
- Internet Security
- User interface building
- Graphics
- Database access

Library Examples

- WebKit
  - Web Browser Engine
- OpenGL
  - High Performance Graphics
  - Render 2D or 3D Graphic Content
- libc
  - Generic C library
- SQLite
  - Storage and sharing of application data

Library Examples Cont.

- Surface Manager
  - Off-screen buffering
    - Apps can't directly draw into screen
    - Drawings go to off-screen buffer
    - Combined with other drawings
    - Reason behind window transparency
- Media Framework
  - Provides media codecs allowing recording and playback of different types of media


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
XMPP 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

Application Framework

- Higher Level Services to Applications
- Environment in which applications are run and managed
- Package Manager
  - Keeps track of installed Applications
  - Apps can communicate with other Apps on device
- Window Manager
  - Manages main window that comprises Application

Application Framework Cont.

- View System
  - Provide Common User Interface Elements
    - Icons
    - Buttons
    - Text Entry
    - Etc.
- Content Providers
  - Databases that allow application to store and share structured info

Application Framework Cont.

- Location Manager
  - Allows application to receive location and movement info generated by GPS
- Activity Manager
  - Manages activity life cycle of applications
- Telephony Manager
  - Manages all voice calls

Application Framework Cont.

- Resource Manager
  - Manage various types of resources used in applications
  - Allows access to non-code embedded resources
    - Strings
    - Color settings
    - UI Layout
- Notifications Manager
  - Allows applications to display alerts


Applications
Home
Contacts
Phone
Browser
...

Applications

- Hosts Android Applications
- Written in Java
  - Access to all Android APIs
- Executed in the VM (Dalvik or ART)
- Examples
  - SMS client app
  - Dialer
  - Web Browser
  - Contact manager

Conclusion

- **Designed** for mobile and flexibility
  - Both in software and harrdware
- 5 Layers
- Application Development
  - Simple
    - Java
  - Access to all aspects of thhe Kernel
    - Open Source
    - APIs

Android Application Components

Android Application Components

- Application components are the essential building blocks of an Android application.
- These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact.

Android Application Components

- Activities
  - They dictate the UI and handle the user interaction to the smart phone screen.
- Services
  - They handle background processing associated with an application.
- Broadcast Receivers
  - They handle communication between Android OS and applications.
- Content Providers
  - They handle data and database management issues.

Activity

- An activity represents a single screen with a user interface,in-short Activity performs actions on the screen.
- For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
- If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.
- An activity is implemented as a subclass of Activity class as follows –

```
public class MainActivity extends Activity  {
}
```

Service

- A service is a component that runs in the background to perform long-running operations.
- For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.
- A service is implemented as a subclass of Service class as follows –

```
public class MyService extends Service
{
```

Broadcast Receivers

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
- A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcaster as an Intent object.

```
public class MyReceiver  extends BroadcastReceiver
{
   public void onReceive(context,intent){}
}
```

Content Providers

- A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. The data may be stored in the file system, the database or somewhere else entirely.
- A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider
extends  ContentProvider {
    public void onCreate(){}
}
```

Additional Components

- Fragments
  - Represents a portion of user interface in an Activity.
- Views
  - UI elements that are drawn on-screen including buttons, lists forms etc.
- Layouts
  - View hierarchies that control screen format and appearance of the views.
- Intents
  - Messages wiring components together.
- Resources
  - External elements, such as strings, constants and drawable pictures.
- Manifest
  - Configuration file for the application.

Overview

- History of Android Architecture
- Five Layers
  - Linux Kernel
  - Android Runtime
  - Libraries
  - Application Framework
  - Applications
- Summary

History

- 2003 –Founded
  - No product for two years, funded by Andy Rubin
    - Planned the next generation of smartphones
    - **Open source** evolution of “Danger”
- 2005 – Purchased by Google
  - Sooner or G1?
- 2007 – Publically announced
- 2008 – Sold first phone

G1

Previous Versions

Lollipop
09/2014

Marshmallow
09/2014

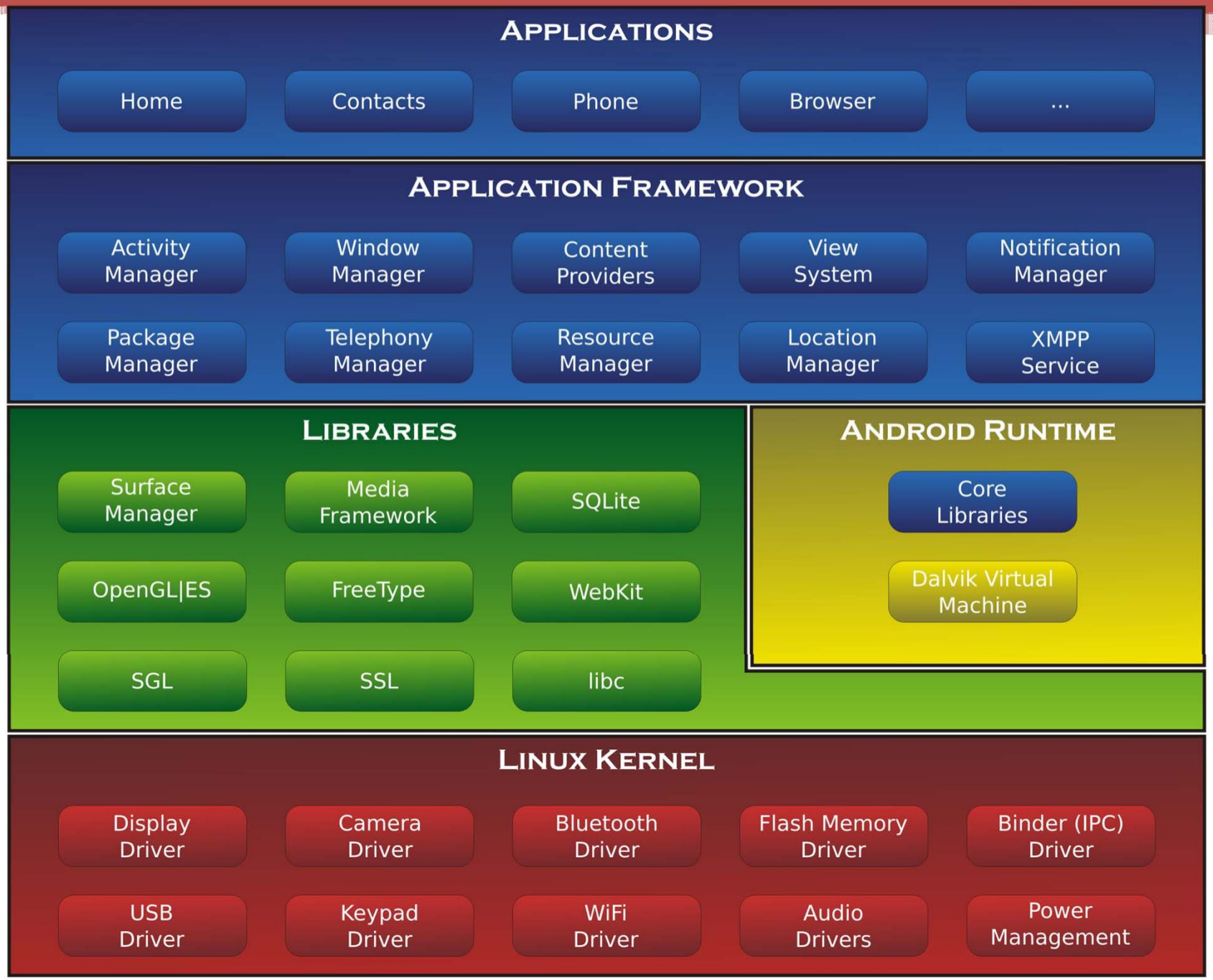
Previous Versions

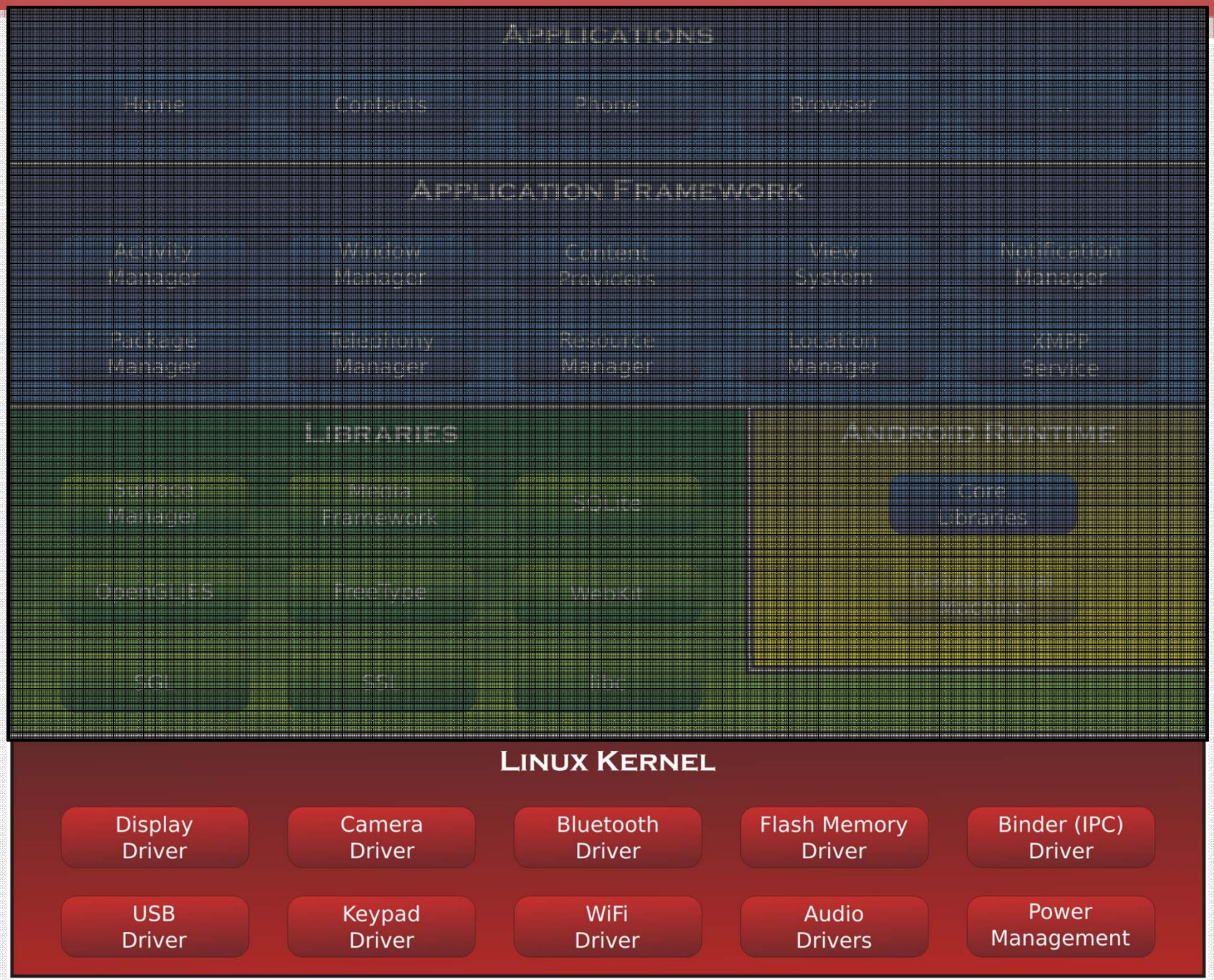
Cupcake

Ice Cream Sandwich

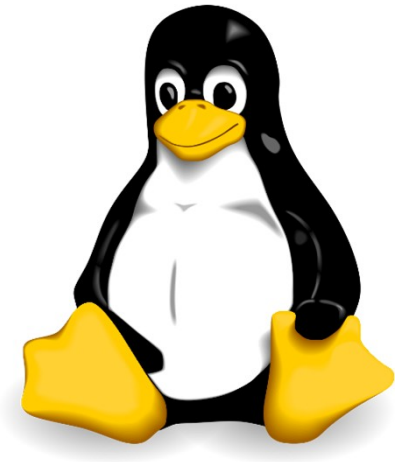
Lollipop

- Unnamed (1.0 + 1.1)
- **C**upcake (1.5)
- **D**onut (1.6) – Quick Search Box
- **É**clair (2.1) – High Density Displays, Traffic + Navigation
- **F**royo (2.2) – Voice Control, Hotspot, Speed
- **G**ingerbread (2.3) – Simpler, Battery Life, More apps
- **H**oneycomb (3.0) – Flexible interface, tablets
- **I**ce Cream Sandwich (4.0) - Customization
- **J**elly Bean (4.1) – Google Now, actionable notifications
- **K**itKat (4.4) – “Ok Google”, voice control variety
- **L**ollipop (5.0) – fluid tactile screens
- **M**arshmallow (6.0) – battery life, app permissions, UI


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
XMPP 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


Linux Kernel
Display Driver
Camera Driver
Bluetooth Driver
Flash Memory Driver
Binder (IPC) Driver
USB Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

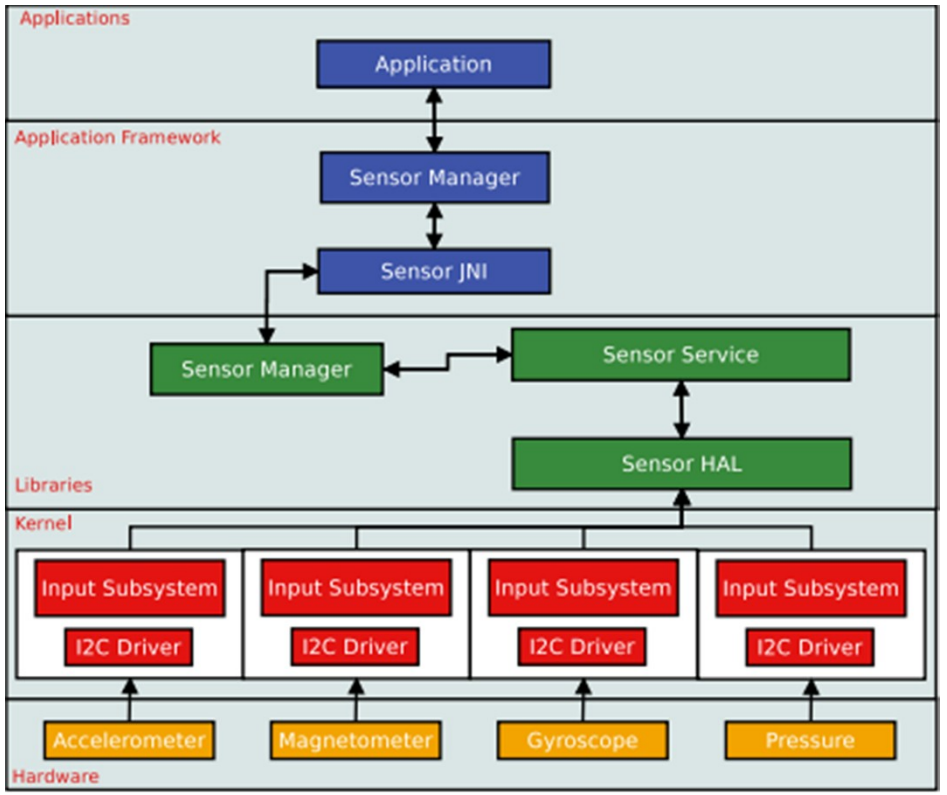
Linux Kernel



- 3.6 with ~115 patches
- Generic System Services
  - Permissions
  - Memory and Process management
  - File & Network I/O
  - Device Drivers
- Preemptive Multitasking
- Lean, efficient, and secure
- Open Source

Hardware Abstraction Layer (HAL)

- Software hooks between stack and hardware
- Hardware Specific
  - Allows Applications to be hardware ignorant



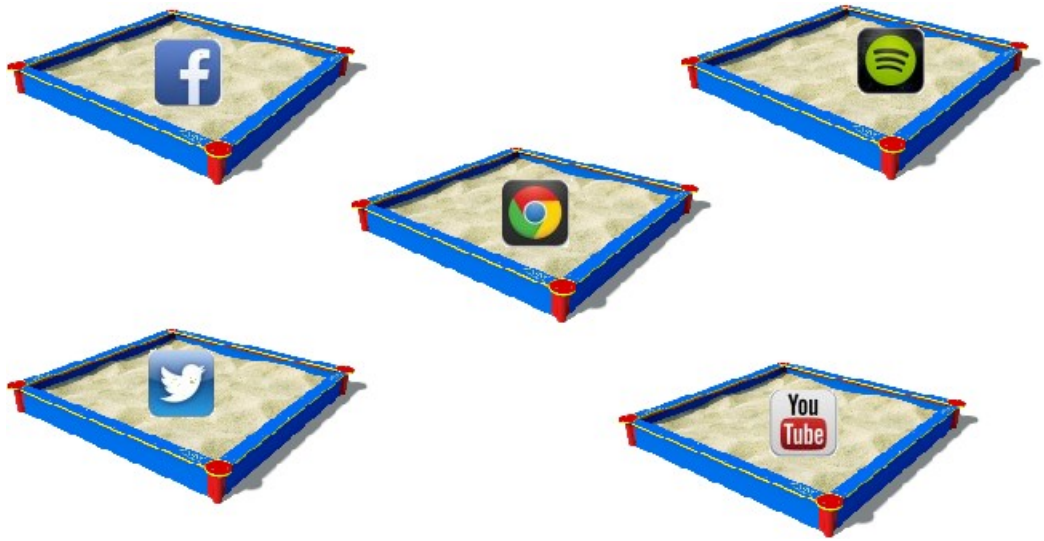

Applications
Home
Contacts
Phone
Browser
...
Application Framework
Activity Manager
Window Manager
Content Providers
View System
Notification Manager
Package Manager
Telephony Manager
Resource Manager
Location Manager
XMPP Service
Libraries
Surface Manager
Media Framework
SQLite
OpenGL|ES
FreeType
WebKit
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
- Core Java libraries
  - Specific to Android development
    - Apple: Swift (Objective C)
    - Windows: Visual C++ (C++), Changes with OS
  - Wrappers around C/C++ libraries
- ART (Android Runtime VM)
  - Replaced Dalvik in Lollipop (Android 5.0)
  - Advantages over Dalvik
    - AOT (Ahead of Time) Compilation
    - Improved Garbage Collection

Dalvik Virtual Machine

- Executes Android Applications
  - Each Application runs within its own VM
    - Each app is “sandboxed”
- Memory Management
- Multi-threading




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
XMPP 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

Libraries

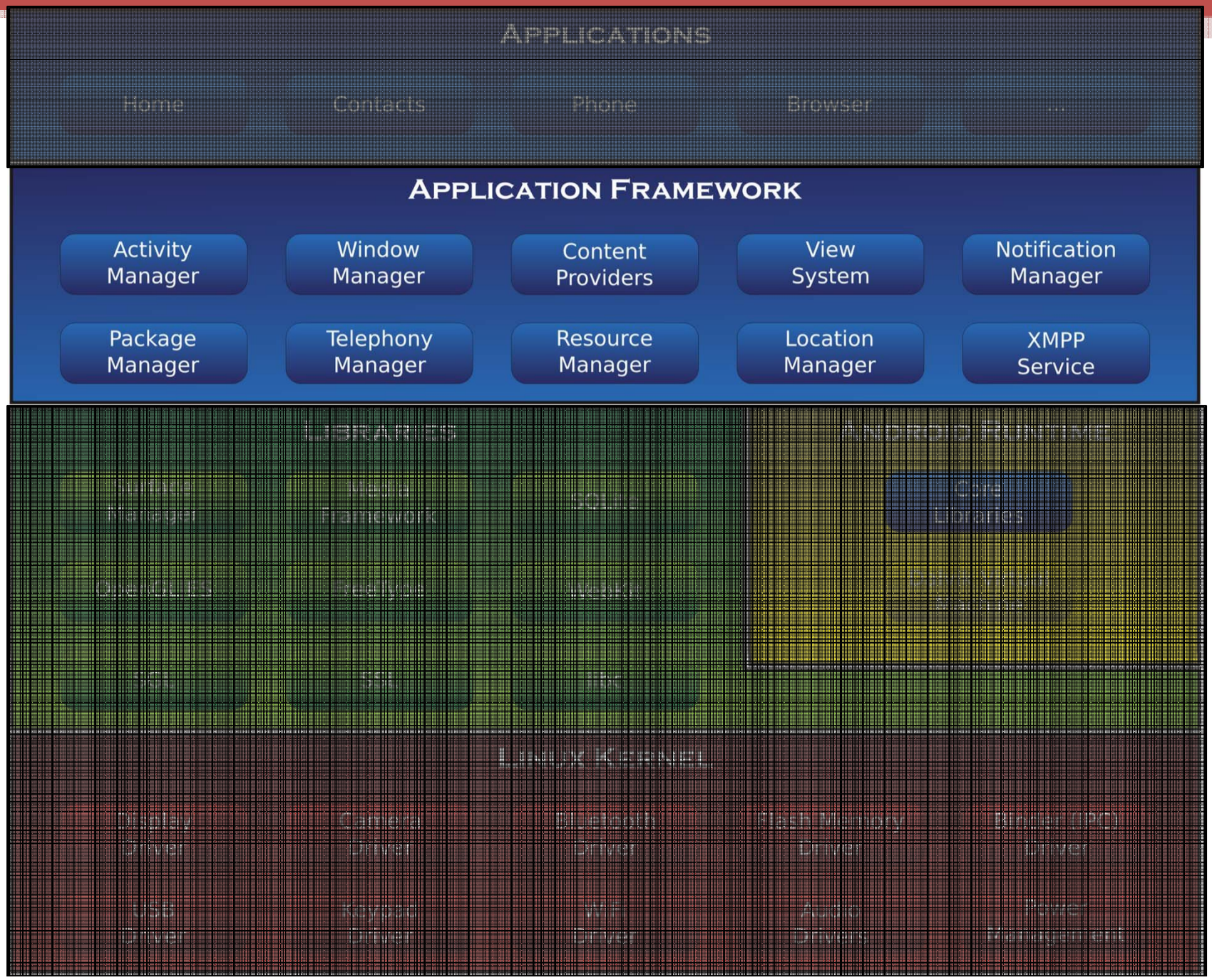
- C/C++
- Play and record audio and video
- Internet Security
- User interface building
- Graphics
- Database access

Library Examples

- WebKit
  - Web Browser Engine
- OpenGL
  - High Performance Graphics
  - Render 2D or 3D Graphic Content
- libc
  - Generic C library
- SQLite
  - Storage and sharing of application data

Library Examples Cont.

- Surface Manager
  - Off-screen buffering
    - Apps can't directly draw into screen
    - Drawings go to off-screen buffer
    - Combined with other drawings
    - Reason behind window transparency
- Media Framework
  - Provides media codecs allowing recording and playback of different types of media


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
XMPP 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

Application Framework

- Higher Level Services to Applications
- Environment in which applications are run and managed
- Package Manager
  - Keeps track of installed Applications
  - Apps can communicate with other Apps on device
- Window Manager
  - Manages main window that comprises Application

Application Framework Cont.

- View System
  - Provide Common User Interface Elements
    - Icons
    - Buttons
    - Text Entry
    - Etc.
- Content Providers
  - Databases that allow application to store and share structured info

Application Framework Cont.

- Location Manager
  - Allows application to receive location and movement info generated by GPS
- Activity Manager
  - Manages activity life cycle of applications
- Telephony Manager
  - Manages all voice calls

Application Framework Cont.

- Resource Manager
  - Manage various types of resources used in applications
  - Allows access to non-code embedded resources
    - Strings
    - Color settings
    - UI Layout
- Notifications Manager
  - Allows applications to display alerts


Applications
Home
Contacts
Phone
Browser
...

Applications

- Hosts Android Applications
- Written in Java
  - Access to all Android APIs
- Executed in the VM (Dalvik or ART)
- Examples
  - SMS client app
  - Dialer
  - Web Browser
  - Contact manager

Conclusion

- **Designed** for mobile and flexibility
  - Both in software and harrdware
- 5 Layers
- Application Development
  - Simple
    - Java
  - Access to all aspects of thhe Kernel
    - Open Source
    - APIs

Android Application Components

Android Application Components

- Application components are the essential building blocks of an Android application.
- These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact.

Android Application Components

- Activities
  - They dictate the UI and handle the user interaction to the smart phone screen.
- Services
  - They handle background processing associated with an application.
- Broadcast Receivers
  - They handle communication between Android OS and applications.
- Content Providers
  - They handle data and database management issues.

Activity

- An activity represents a single screen with a user interface,in-short Activity performs actions on the screen.
- For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
- If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.
- An activity is implemented as a subclass of Activity class as follows –

```
public class MainActivity extends Activity  {
}
```

Service

- A service is a component that runs in the background to perform long-running operations.
- For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.
- A service is implemented as a subclass of Service class as follows –

```
public class MyService extends Service
{
```

Broadcast Receivers

- Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.
- A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcaster as an Intent object.

```
public class MyReceiver  extends BroadcastReceiver
{
   public void onReceive(context,intent){}
}
```

Content Providers

- A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. The data may be stored in the file system, the database or somewhere else entirely.
- A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider
extends  ContentProvider {
    public void onCreate(){}
}
```

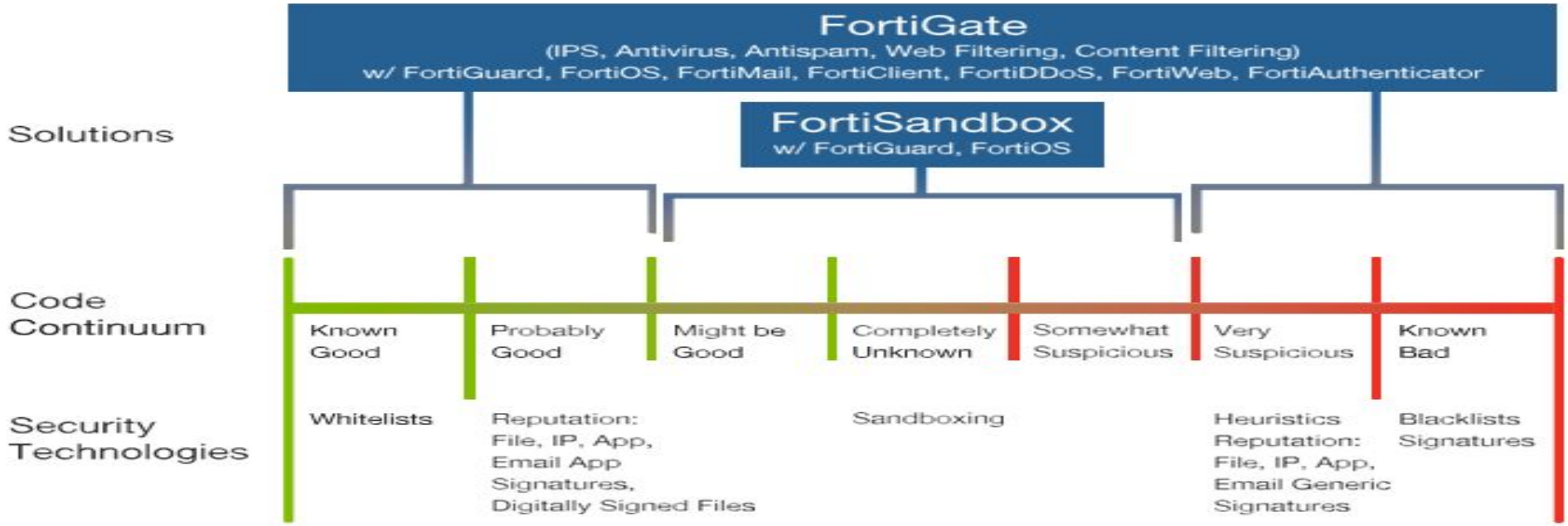
Additional Components

- Fragments
  - Represents a portion of user interface in an Activity.
- Views
  - UI elements that are drawn on-screen including buttons, lists forms etc.
- Layouts
  - View hierarchies that control screen format and appearance of the views.
- Intents
  - Messages wiring components together.
- Resources
  - External elements, such as strings, constants and drawable pictures.
- Manifest
  - Configuration file for the application.

What is Sandbox?

A sandbox is a safe isolated environment that replicatesan end user operating environment where you can run code, observe it and rate it based on activity rather than attributes. You can run executable files, allow contained network traffic and more that can contain hidden malware in a sandbox.

The sandbox provides a safe environment in which to execute and observe malicious code such as file/disc operations, network connections, registry/system configuration changes, etc.



Why Do You Need Sandboxing for Protection?

Organizations breached by Advanced Persistent Threats (APTs) are all over the news and sandboxing is the latest hot thing being touted to protect you from APTs. Why? Why sandboxing? What does a sandbox solution give you that you don't already get from your existing layers of security?

A Sandbox gives you a chance to see into the future, into the unknown.

We don't live in a black and white world, where everything is known to be good or bad. The code that runs over your network spans a continuum from known good code to known bad or that includes malicious code. A lot is simply unknown. You are likely already running a number of security technologies to help protect your organization from malicious code and you are probably running technologies that help you identify good code.

However, like most organizations, you are still at risk from the unknown. And that unknown gap in the code continuum is a significant one.


Code Continuum
Known Good
Probably Good
Might be Good
Completely Unknown
Somewhat Suspicious
Very Suspicious
Known Bad
Security Technologies
Whitelists
Reputation: File, IP, App, Email App Signatures, Digitally Signed Files
Sandboxing
Heuristics
Reputation: File, IP, App, Email Generic Signatures
Blacklists
Signatures

How does sandboxing fit into the many layers of security ?

Consider the lifecycle of an attack and how security technologies play a part in protecting an organization.

Step 1: An attacker starts with reconnaissance on the target. They then craft a clever email, often with a malicious link (or file) in it. And send the email to the target. This is where your antispam/antiphishing solution may block the email. But if it doesn't: the email goes to the target where the attacker hopes the target will click on the malicious link.

Step 2: If the target clicks on the link, traffic will go out to a web site to establish communication. This is where your web filter may block the traffic but if it doesn't:

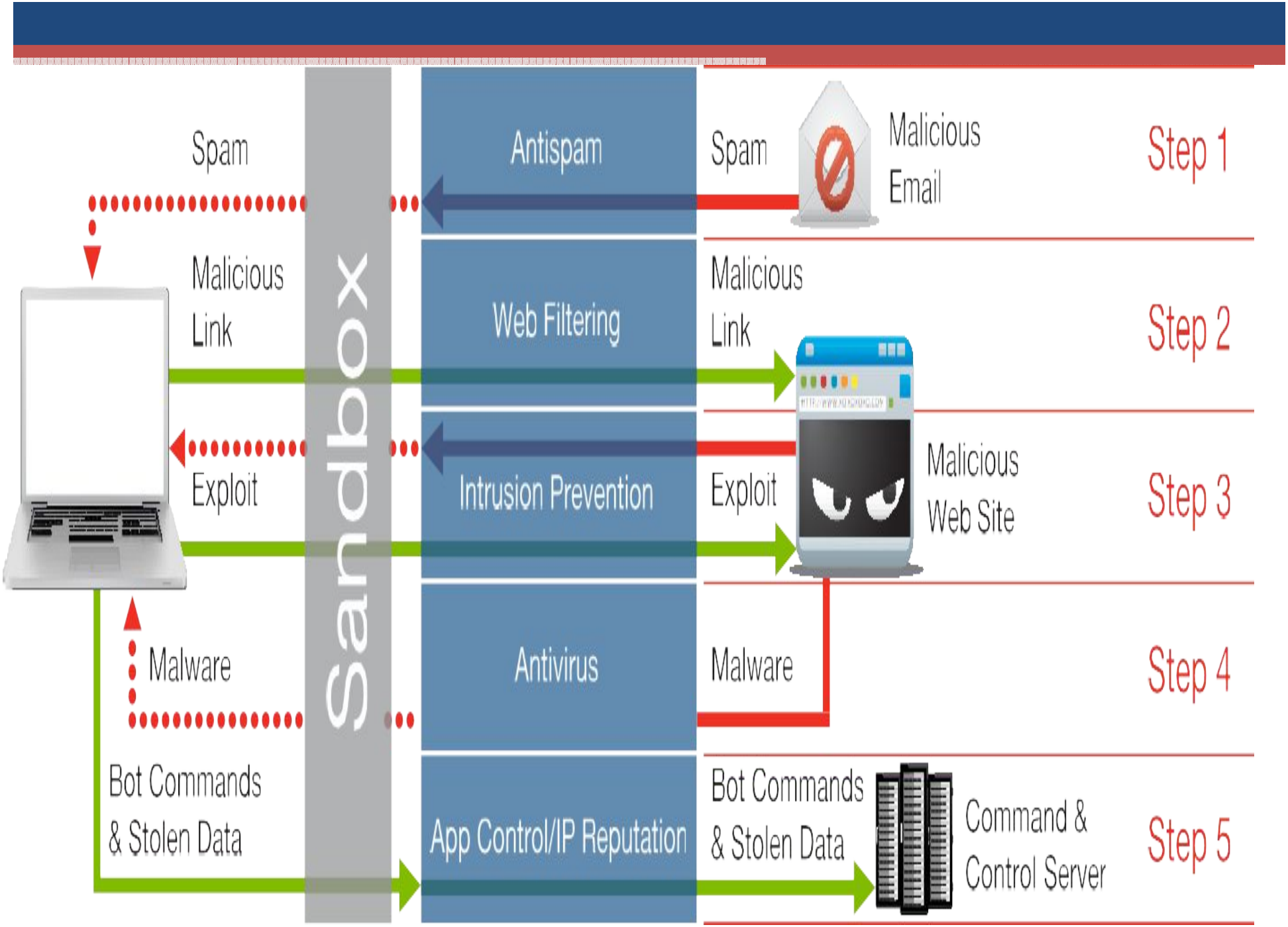
that malicious web site starts attacking your organization.

Step 3: The malicious web site will usually launch exploit attacks at the target to gain access to the system. This is where your intrusion prevention system (IPS) attempts to block the attack, but if it doesn't: a tunnel is opened up and the malicious site can launch malware into

your organization.

Step 4: With malware seeking entry, ideally your antimalware will protect you, but if it doesn't the attacker gets executable code into your system where it can run.

Step 5: Once the malicious code is running, it usually looks to access credentials, move laterally in search of sensitive data and collect/stage it within your organization. But in order to complete its mission, itneeds to exfiltrate that data out to a command & control server. This is where your application control, IP reputation, botnet and other protections come intoplay. But if these technologies don't block this traffic: You are breached.


Spam
Malicious
Link
Exploit
Malware
Bot Commands
& Stolen Data
Sandbox
Antispam
Web Filtering
Intrusion Prevention
Antivirus
App Control/IP Reputation
Spam
Malicious
Email
Step 1
Malicious
Link
Step 2
Exploit
Malicious
Web Site
Step 3
Malware
Step 4
Bot Commands
& Stolen Data
Command &
Control Server
Step 5

Android application inter-process communication:

IPC is a general concept that means inter-process communication.
When it comes to Android, IPC covers the following two situations:
Communication between applications
Communication of processes in a multi-process application (Application whose components such as Activity, Service, Receiver, Provider are run in different processes)

There are 3 basic methods used for IPC on Android:

- Android Interface Definition Language (AIDL): It allows you to define the programming interface that both the client and service agree upon in order to communicate with each other using interprocess communication (IPC). On Android, one process cannot normally access the memory of another process.

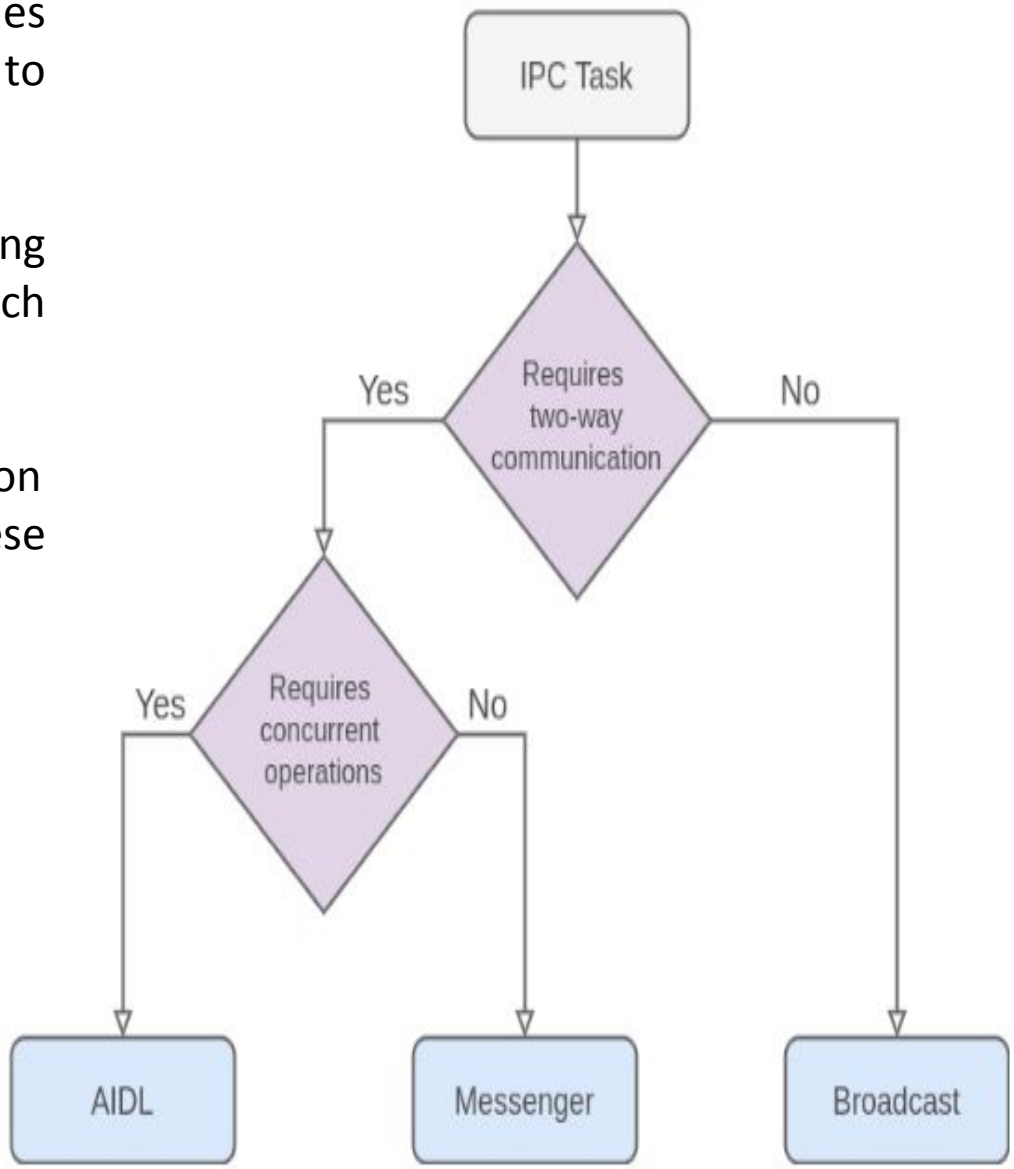
- Messenger: Messenger is a Handler sent to the remote process. Messenger is much easier to implement than the AIDL technique because it creates and uses AIDL files in the background. The developer does not need to bother with this implementation.

- Broadcast: Broadcasts can be sent by the system or applications. Applications can register to listen to specific broadcasts with Broadcast Receiver.

- It is also possible to use traditional Linux techniques (such as socket communication shared files) to perform IPC.

- However, official documents recommend choosing one of the three methods specific to Android, which we mentioned above, in terms of security.

- Advantages such as **authenticating** the application that wants to communicate are possible in these three methods.



What are Android app permissions?

As the name suggests, permissions govern what an app is allowed to do and access. This ranges from reading the data stored on your phone, such as contacts and media files, through to using hardware including your handset's camera or microphone. Granting permission allows the app to use the feature. Denying access prevents it from doing so. Simple enough.

Apps cannot automatically grant themselves permissions, these have to be confirmed by the user via an on-screen prompt. Apps will ask you to accept each of their requested permissions the first time you launch them via a popup that asks you to "allow" or "deny" each request. This will also reoccur on startup if you deny permissions or if an app is updated to require new permissions.

1. **How to grant or deny Android permissions on a per-app basis**

2. **How to check Android permissions by type**

3. **Which Android app permissions to allow and deny**

Android Partition:

Unless you have been using your Android phone just for calls, SMS, browsing and basic apps, you should know that Android uses several partitions to organize files and folders on the device.

Each of these partitions has a distinct role in the functionality of the device, but not many Android users know the significance of each partition and its contents.

/boot
/system
/recovery
/data
/cache
/misc
/sdcard
/sd-ext

/boot

This is the partition that enables the phone to boot, as the name suggests. It includes the kernel and the ramdisk. Without this partition, the device will simply not be able to boot.

/System

This partition basically contains the entire operating system, other than the kernel and the ramdisk. This includes the Android user interface as well as all the system applications that come pre-installed on the device.

/recovery

The recovery partition can be considered as an alternative boot partition that lets you boot the device into a recovery console for performing advanced recovery and maintenance operations on it.

/data

Also called userdata, the data partition contains the user's data – this is where your contacts, messages, settings and apps that you have installed go.

/cache

This is the partition where Android stores frequently accessed data and app components.

/misc

This partition contains miscellaneous system settings in form of on/off switches.

/sdcard

This is not a partition on the internal memory of the device but rather the SD card. In terms of usage, this is your storage space to use as you see fit, to store your media, documents, ROMs etc. on it.

/sd-ext

This is not a standard Android partition, but has become popular in the custom ROM scene. It is basically an additional partition on your SD card that acts as the /data partition when used with certain ROMs that have special features useful on devices with little internal memory allotted to the /data partition.

File System

- Sometimes you might prefer to use the traditional file system to store your data.
- For example, you might want to store the text of poems you want to display in your applications.
- In Android, you can use the classes in the java.io package to do so.
- The first way to save files in your Android application is to write to the device's internal storage.
- The first way to save files in your Android application is to write to the device's internal storage.
- Then use its write() method to write the string to the file.
- To ensure that all the bytes are written to the file, use the flush() method. Finally, use the close() method to close the file.
- To read the content of a file, you use the FileInputStream class, together with the InputStreamReader