Android

Learning Android

Marko Gargenta

Materials

- Sams Teach Yourself Android Application Development in 24 Hours (Amazon)
- Android Apps for Absolute Beginners (Amazon)
- Android Development Tutorial (http://www.vogella.com/articles/Android/article.html)

Introduction to Android

Android is an operating system based on Linux with a Java programming interface. It is a comprehensive open source platform designed for mobile devices.

- Open software platform for mobile development
- A complete stack -- OS, middleware, applications
- An Open Handset Alliance (OHA) project
- Powered by Linux operating system
- Fast application development in Java
- Open source under the Apache 2 license

Introduction to Android

Android is a complete software stack for a mobile device for developers, users, and manufactures

- Android software development kit (Android SDK) provides all necessary tools to develop Android applications
  - Complier, debugger, device emulator, and virtual machine to run Android programs
- Android allows background processing, provides a rich user interface library, supports 2-D and 3-D graphics using the OpenGL libraries, access to the file system and provides an embedded SQLite database.
- Android applications can re-use components of other applications.
  - trigger camera from your application

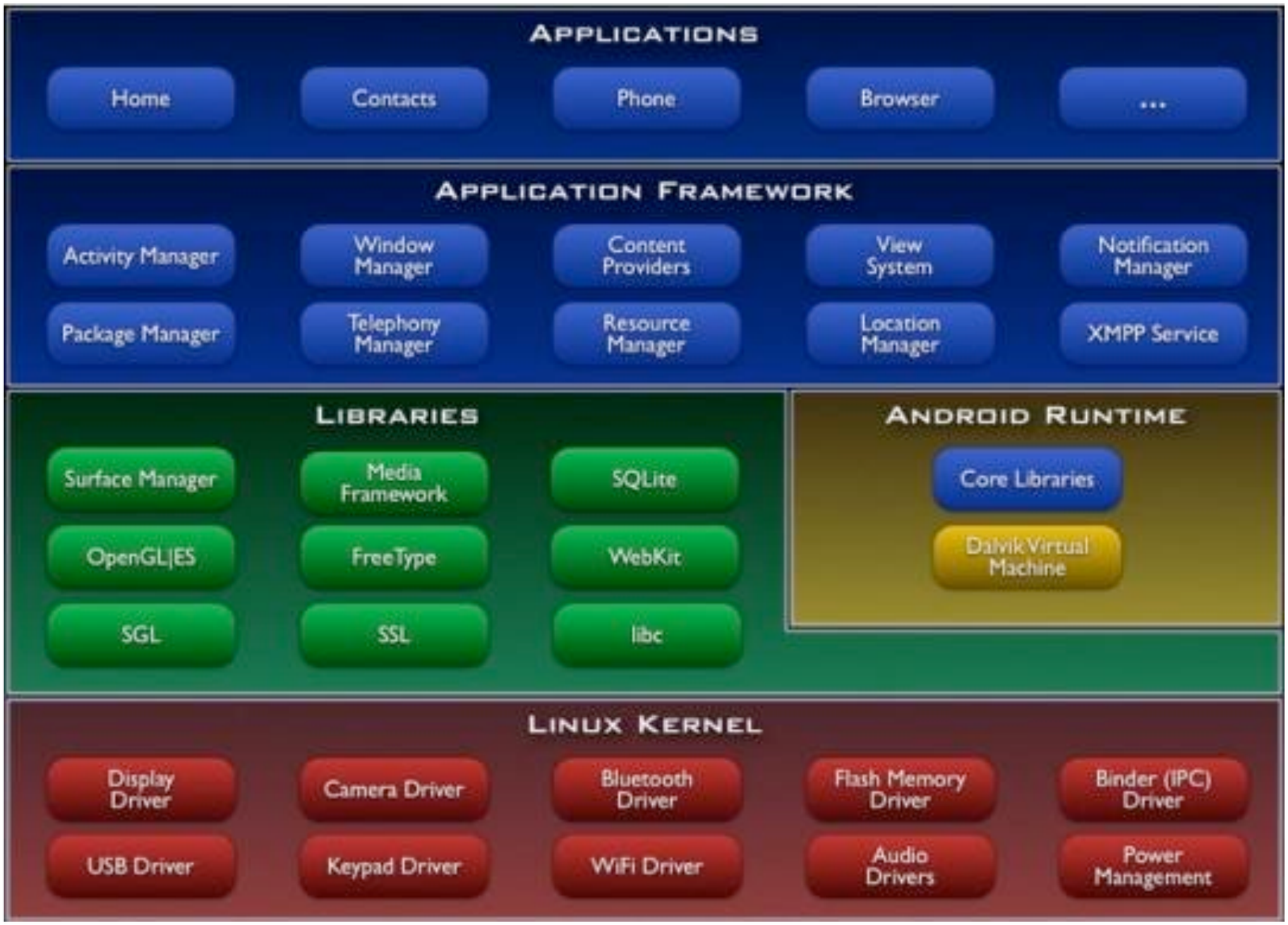
Open Source

- Android is an open source platform, including low-level Linux modules, native libraries, application framework, to applications.
- Android is licensed under business-friendly licenses (Apache).
- As developer, you can start the development from today and access the entire platform source code.

Mobile platform

- Mobile constraints: battery capacity, processing speed, memory size -- User experience
- Mobile diversities: screen size, resolution, chipset. -- Portability

Android is like a piece of cake

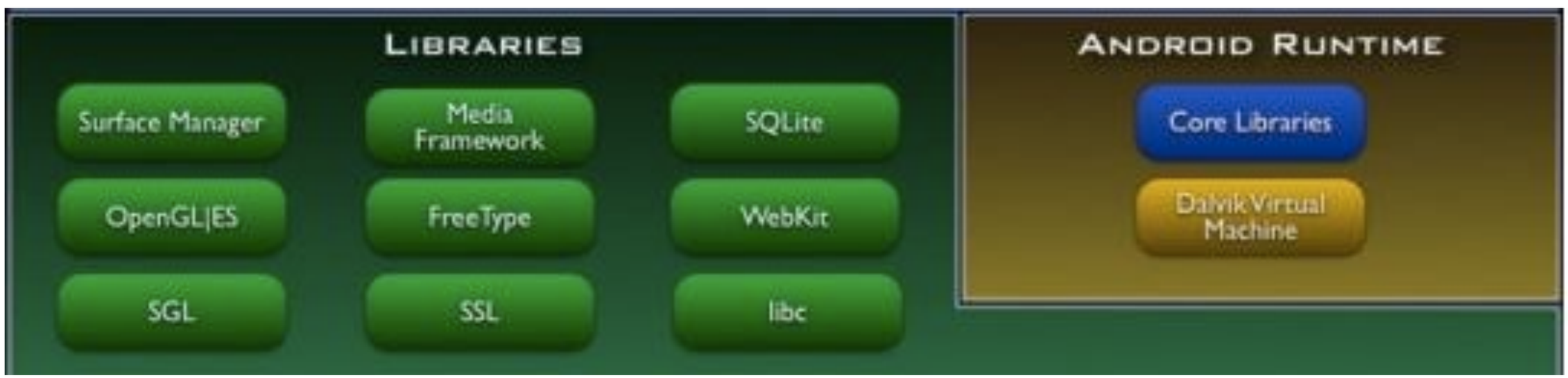


Linux kernel

- Works as a hardware abstraction layer (HAL)
- Portability, security, features
- Device drivers
- Memory management, process management, networking

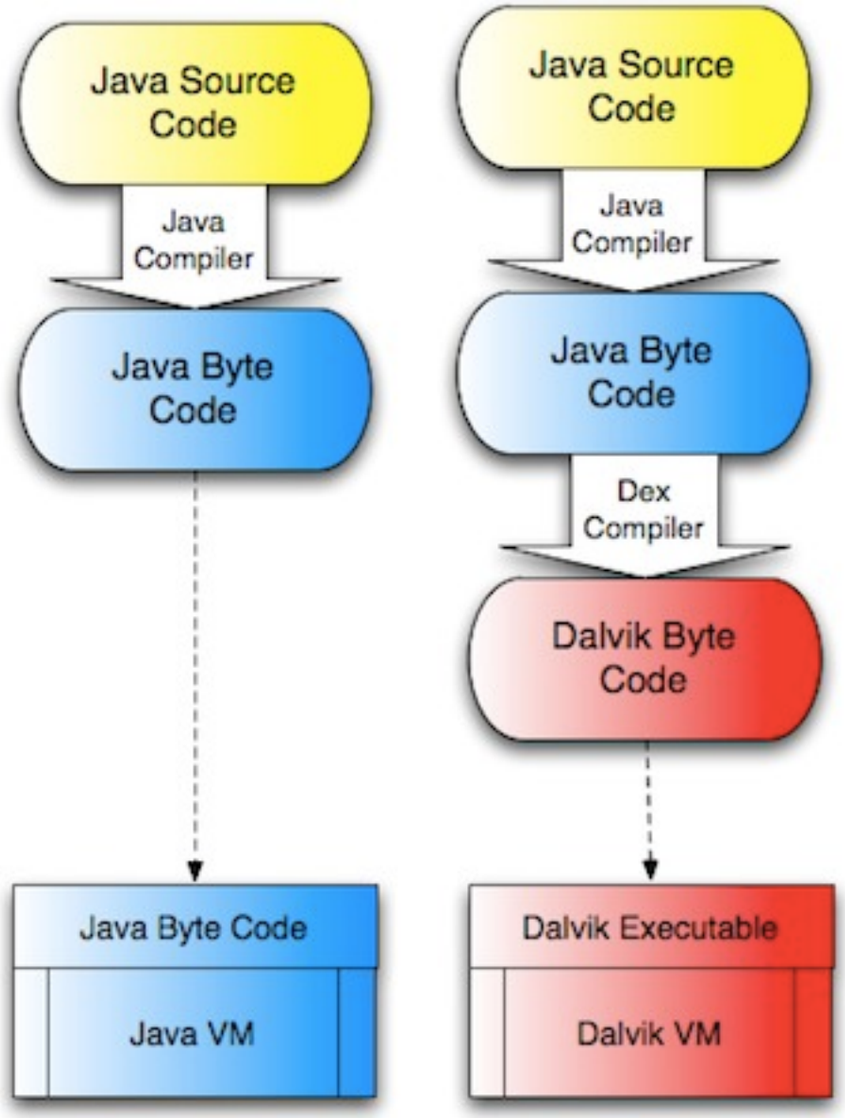
Native Libraries

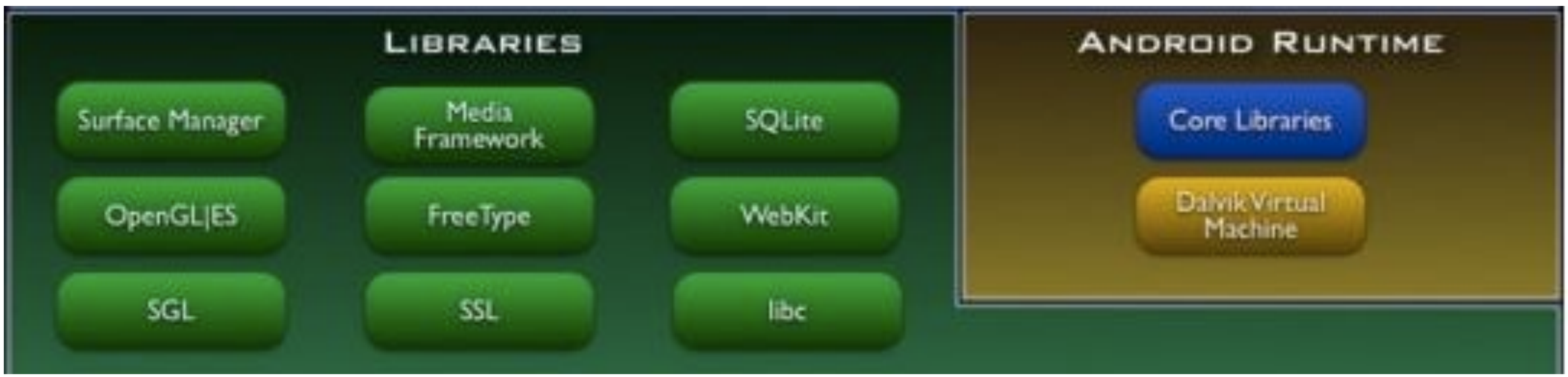
- Native libraries are C/C++ libraries often taken from the open source community in order to provide necessary services to Android application layer, e.g., WebKit, SQLite, OpenGL



Dalvik

- Dalvik is a purpose-built virtual machine designed specifically for Android.



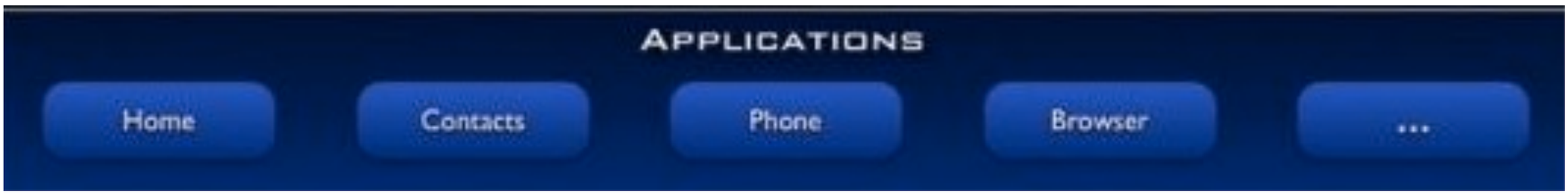


App Framework

- In application framework layer, you will find numerous Java libraries specifically built for Android. You will also find many services (or **managers**) that provide the ecosystem of capabilities your application can tap into, such as location, sensors, WiFi, telephony and so on.

Applications

- APK: an application is a single APK (application package) file, including Dalvik executable code, resources, and native code
- Signing: Android applications must be signed before they can be distributed commercially.
- Distribution: Can be distributed through many different Android stores or markets.



How to learn Android?

- The best/only way is by using it.

The Android development platform

- Eclipse
- ADT
- SDK

Eclipse

- In computer programming, **Eclipse** is a multi-language software development environment comprising a base workspace and an extensible plug-in system for customizing the environment.

ADT

- Android Development Tools (ADT) is a plugin for the Eclipse IDE that is designed to give you a powerful, integrated environment in which to build Android applications.

- ADT extends the capabilities of Eclipse to let you quickly set up new Android projects, create an application UI, add packages based on the Android Framework API, debug your applications using the Android SDK tools, and even export signed (or unsigned) .apk files in order to distribute your application.

SDK

- The Android SDK provides you the API libraries and developer tools necessary to build, test, and debug apps for Android.

Hello World

- Our first application

Manifest File

- Manifest file glues everything together, describing what the application consists of, main building blocks are, permissions required, etc.

Layout XML code

- The layout file specifies the layout of your screen.

Strings

- An XML file containing all the text that the application uses, e.g., names of buttons, labels, and default text and all such strings go into this file.

The R File

- The R file is the glue between the world of Java and the world of resources. It is an automatically generated file and you do not need to modify it.

Java Source Code

- The Java code is what drives everything. It is this code that ultimately gets converted to Dalvik executable and runs your application.

The Emulator

- A software imitate a real Android hardware platform and executing the Android application binary code.

Android Building Blocks

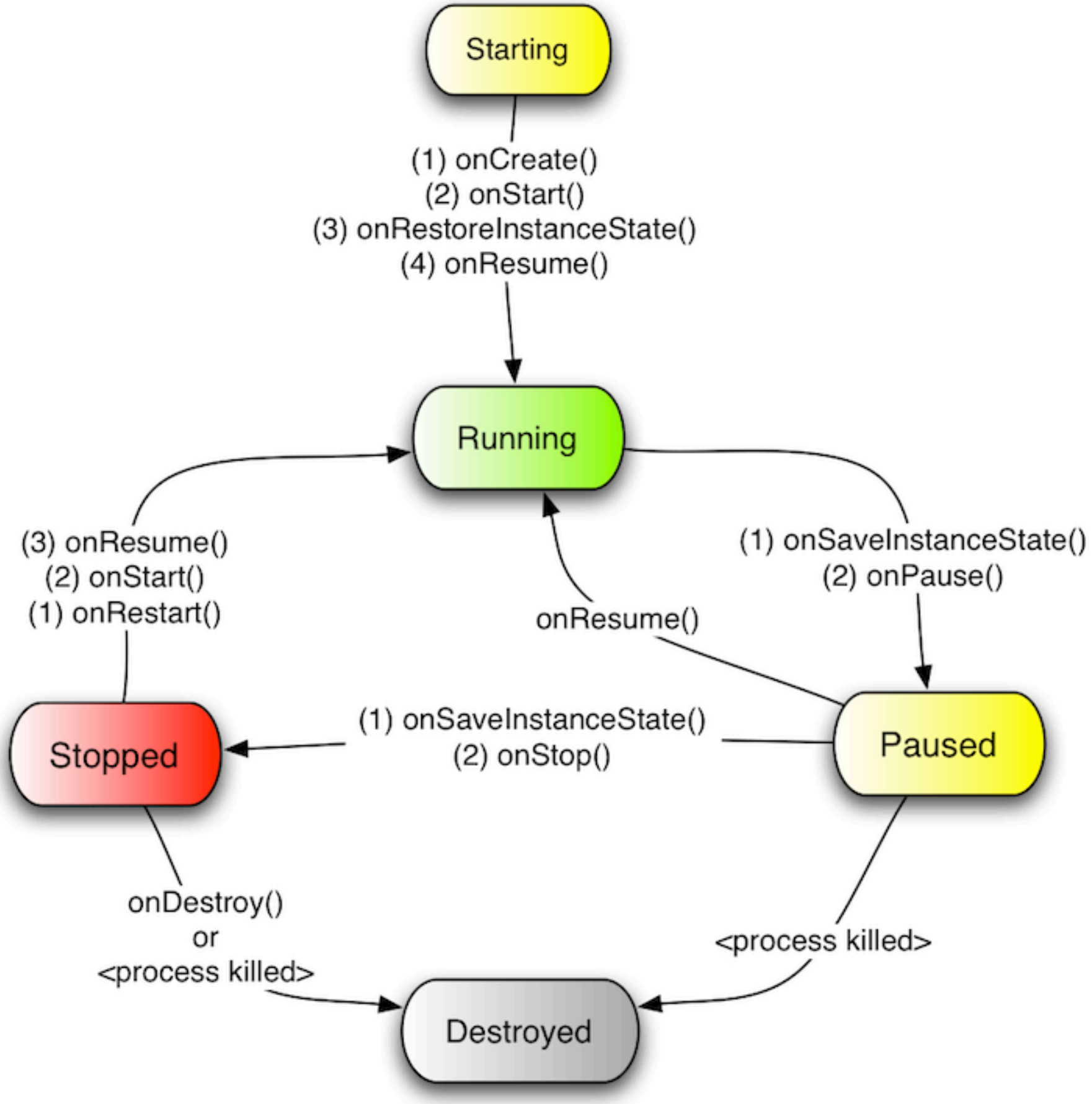
Application Context refers to the application environment and process within all its components are running. It allows to sharing of the data and resources between various Building Blocks.

Activity

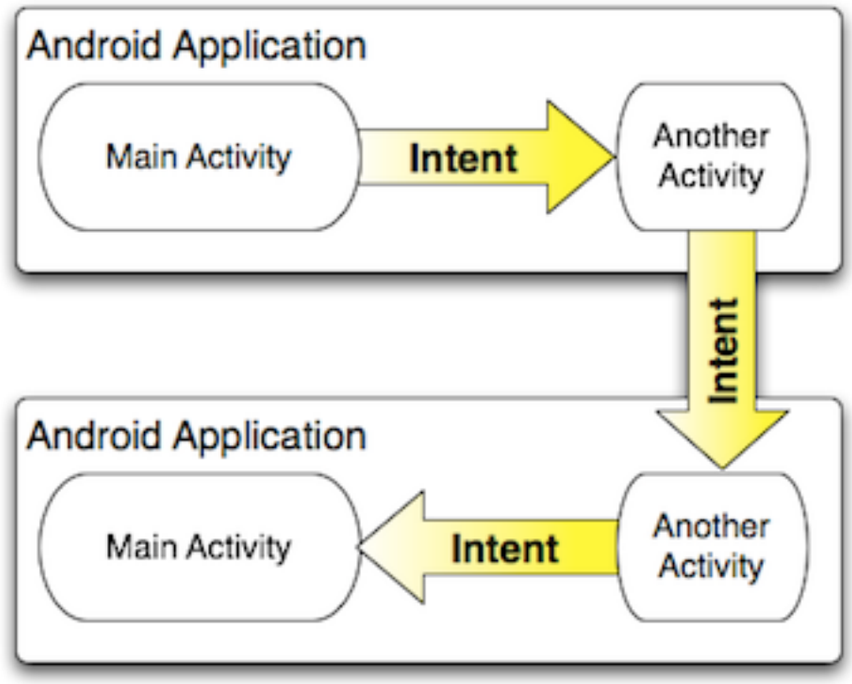
- An activity is usually a single screen that the user sees on the device at one time. An application typically has multiple activities and the user flips back and forth among them. As such, activities are the most visible part of your application
- An activity represents the visual representation of an Android application. activities use views, i.e. user interface widgets as for example buttons and fragments to create the user interface and to interact with the user.
- An Android application can have several activities.

Activity Lifecycle

- Activity manager is responsible for creating, destroying, and overall managing activity.
- Starting, running, paused, stopped, destroyed state



Intent



- Intents are messages that are sent among major building blocks. They trigger an activity to start up, a service to start or stop, or are simply broadcasts. Intents are asynchronous, meaning the code that is sending them doesn't have to wait for them to be completed.

- *Intents* are asynchronous messages which allow the application to request functionality from other Android components, e.g. from *services* or *activities*.

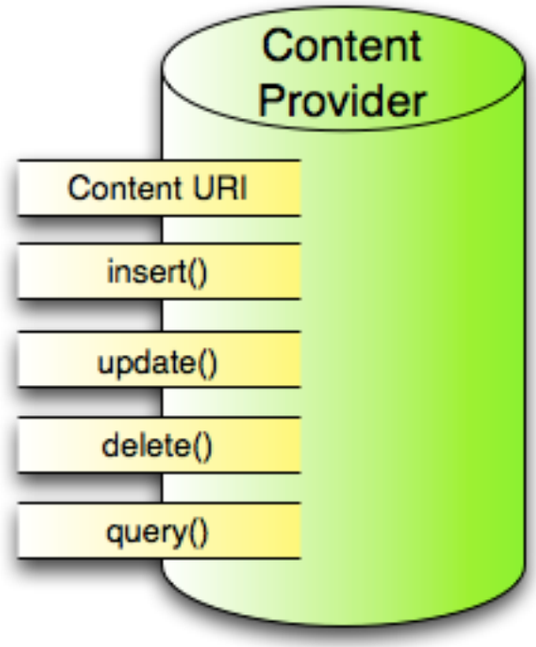
Services

- Services run in the background and don’t have any user interface components. They can perform the same actions as Activities without any user interface. Services are useful for actions that we want to make sure performs for a while, regardless of what is on the screen. For example, you may want to have your music player play music even as you are flipping between other applications.

Content Providers

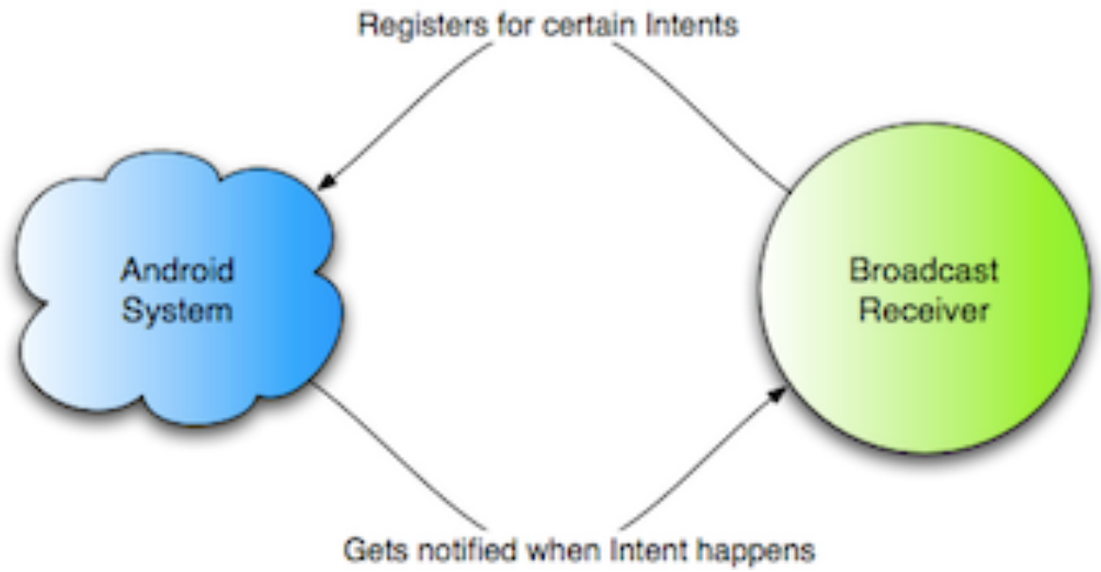
- Content Providers are interfaces for sharing data between applications. Android by default runs each application in its own sandbox so that all data that belongs to an application is totally isolated from other applications on the system.

- While small amounts of data can be passed between applications via Intents, Content Providers are much better suited for sharing persistent data between possibly large datasets.



Broadcast Receivers

- Broadcast receivers is an Android implementation of system-wide publish/ subscribe mechanism.



- The receiver is a dormant code that gets activated once an event it is subscribed to happens

User Interface

User Interface

- Your app's user interface is everything that the user can see and interact with.
- Android provides a variety of pre-build UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app.
- Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus

User Interface

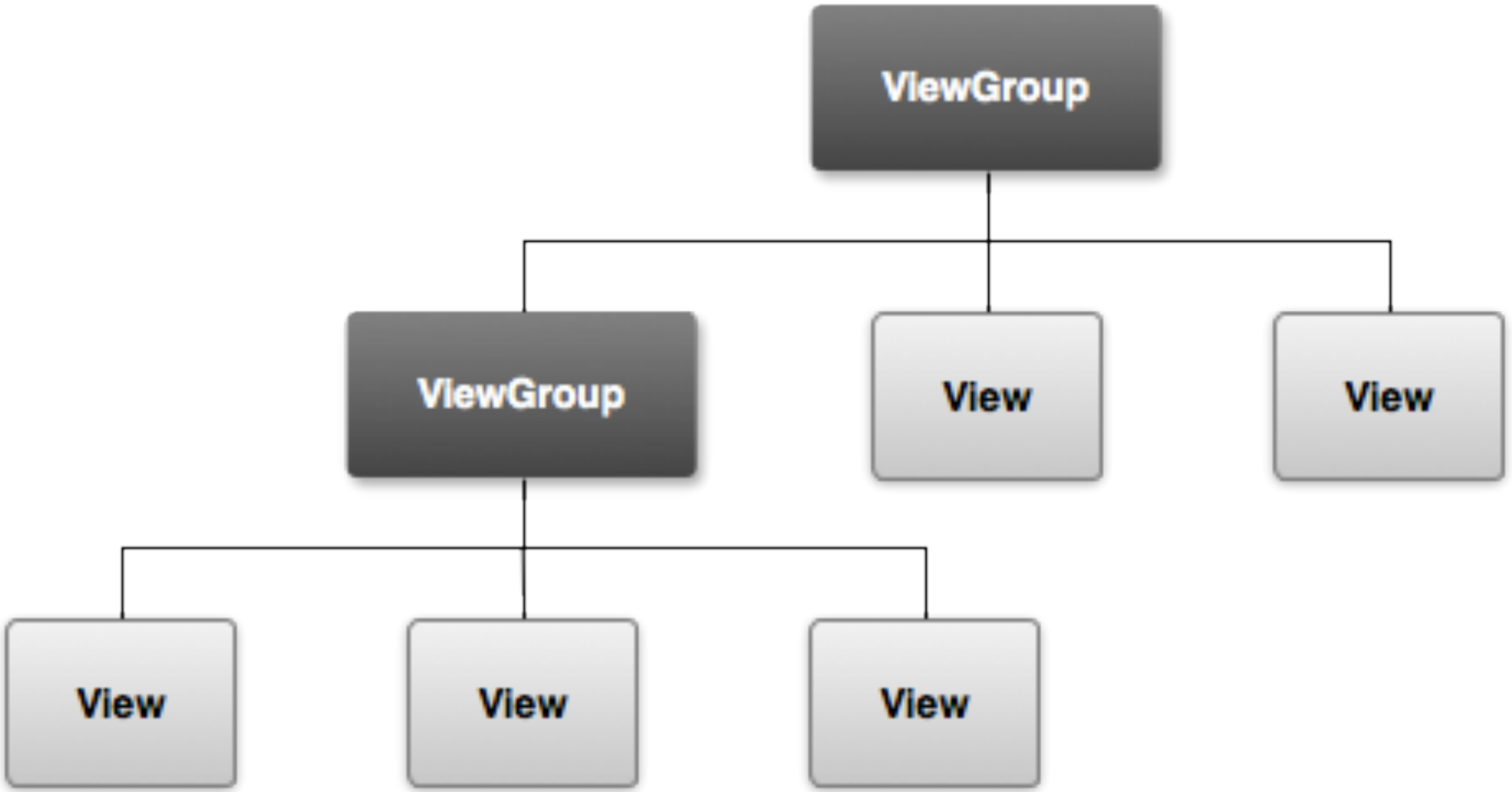
- All user interface elements in an Android app are built using View and ViewGroup objects. A View is an object that draws something on the screen that the user can interact with. A ViewGroup is an object that holds other View (and ViewGroup) objects in order to define the layout of the interface.

- Android provides a collection of both View and ViewGroup subclasses that offer you common input controls (such as buttons and text fields) and various layout models (such as a linear or relative layout).

UI Layout

- The user interface for each component of your app is defined using a hierarchy of `View` and `ViewGroup` objects.
- Each view group is an invisible container that organizes child views, while the child views may be input controls or other widgets that draw some part of the UI.

Layout



Layout Design

Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

- **Instantiate layout elements at runtime**. Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

Layout Design

- The advantage to declaring your UI in XML is that it enables you to better separate the presentation of your application from the code that controls its behavior.

Write the XML

- Using Android's XML vocabulary, you can quickly design UI layouts and the screen elements they contain, in the same way you create web pages in HTML — with a series of nested elements.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/
apk/res/android"
                android:layout_width="fill_parent"
                android:layout_height="fill_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>
```

Load the XML resource

- When you compile your application, each XML layout file is compiled into a `View` resource. You should load the layout resource from your application code, in your `Activity.onCreate()` callback implementation. Do so by calling `setContentView()`, passing it the reference to your layout resource in the form of: `R.layout.layout_file_name`

```
public void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_layout);
}
```

Attributes

- ID: Any View object may have an integer ID associated with it, to uniquely identify the View within the tree.
- `android:id="@+id/my_button"`

Common Layouts

- Linear Layout
- Relative Layout
- List View/ Grid View

Linear Layout

- `LinearLayout` is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the `android:orientation` attribute.

Relative Layout

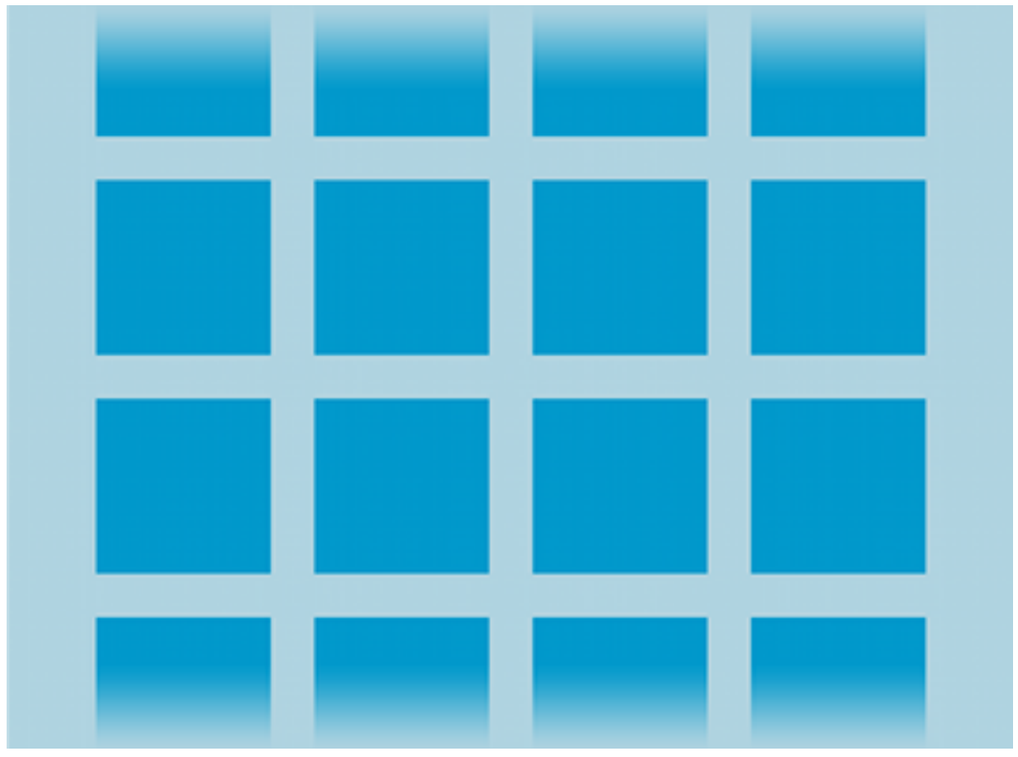
- `RelativeLayout` is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent `RelativeLayout` area (such as aligned to the bottom, left of center).

List View

- `ListView` is a view group that displays a list of scrollable items.

Grid View

- `GridView` is a `ViewGroup` that displays items in a two-dimensional, scrollable grid.



Dynamic UI

- How to build a dynamic UI?

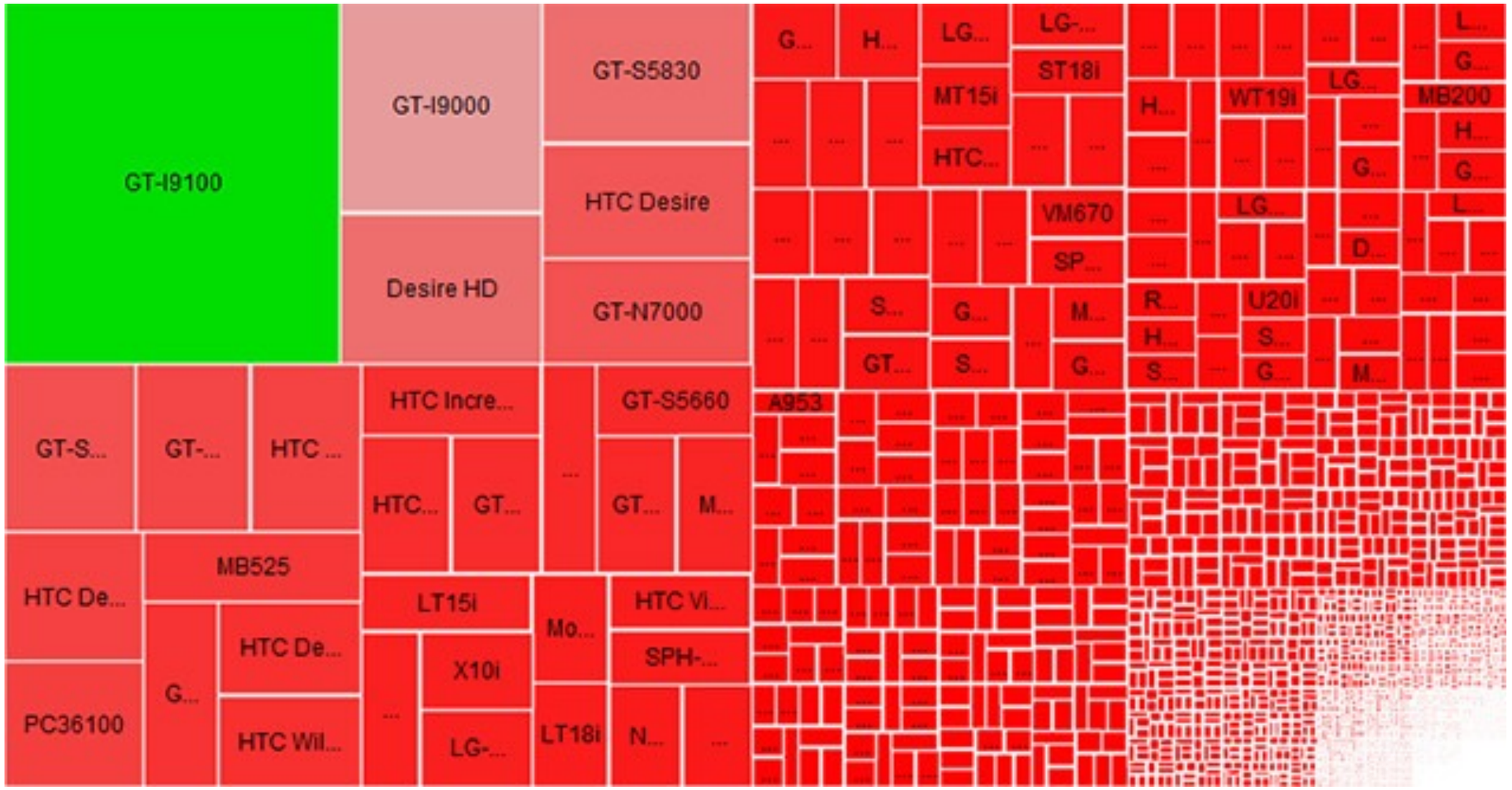
Motivation

http://opensignal.com/reports/fragmentation.php

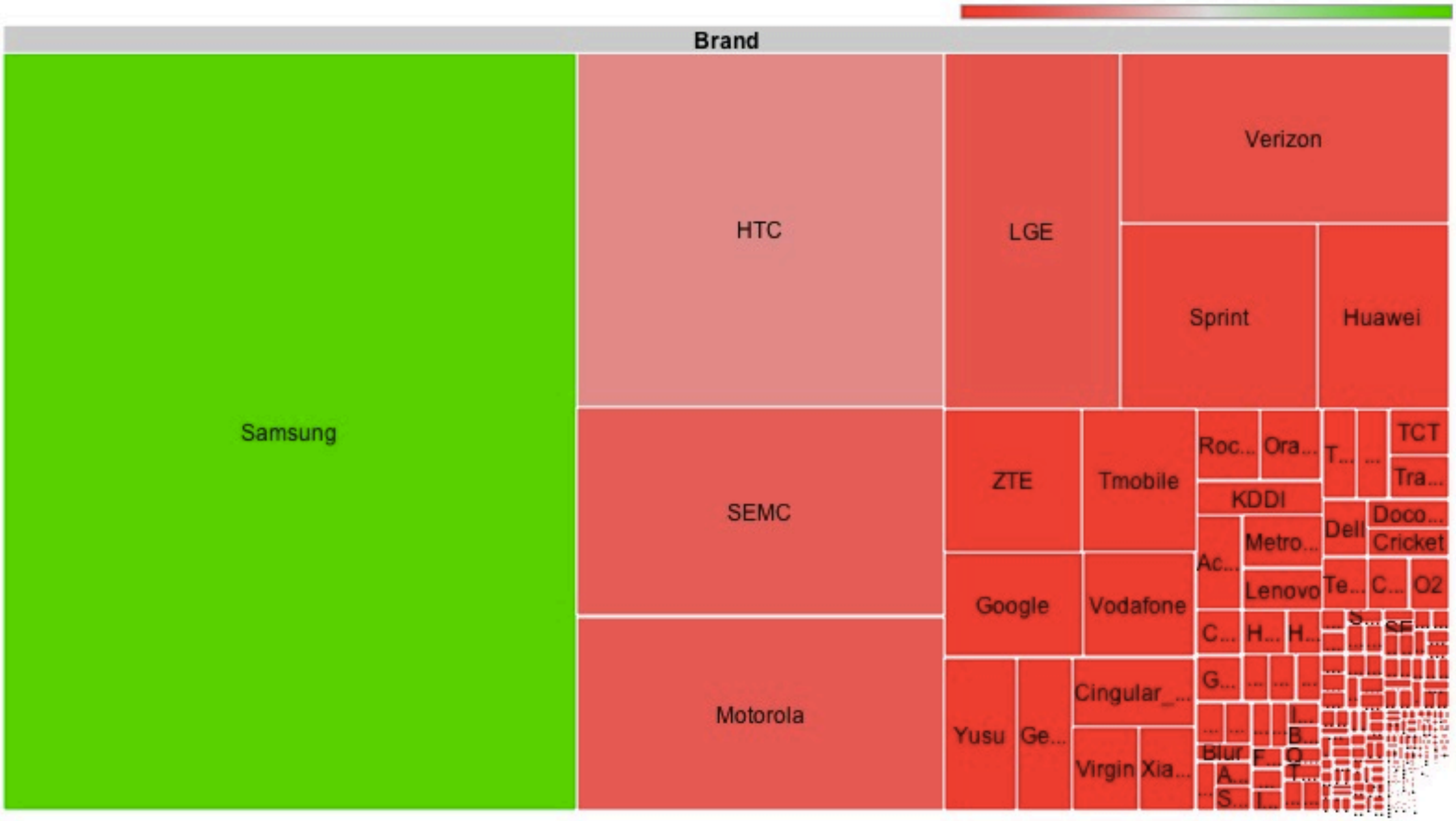
Gone are the days when websites had to be designed for a single screen size.

Motivation

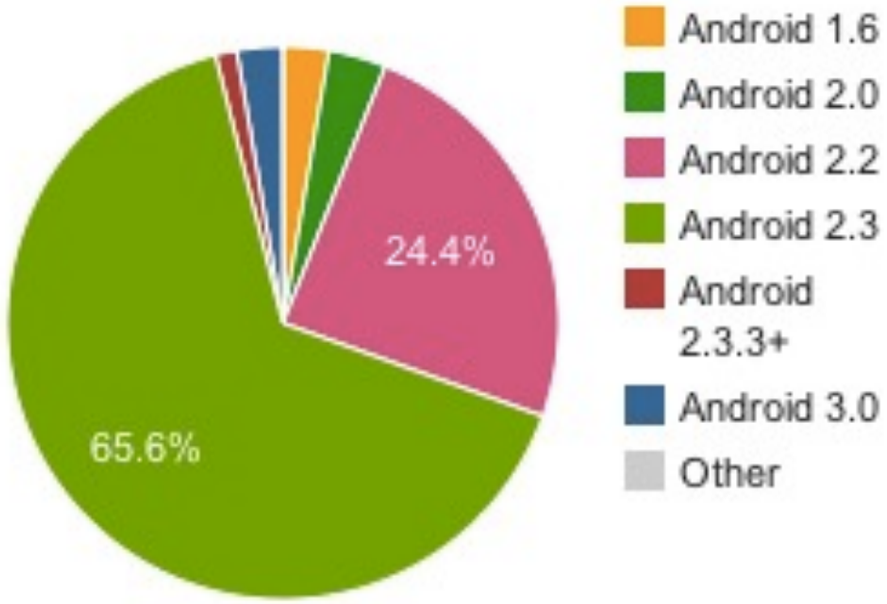
http://opensignal.com/reports/fragmentation.php



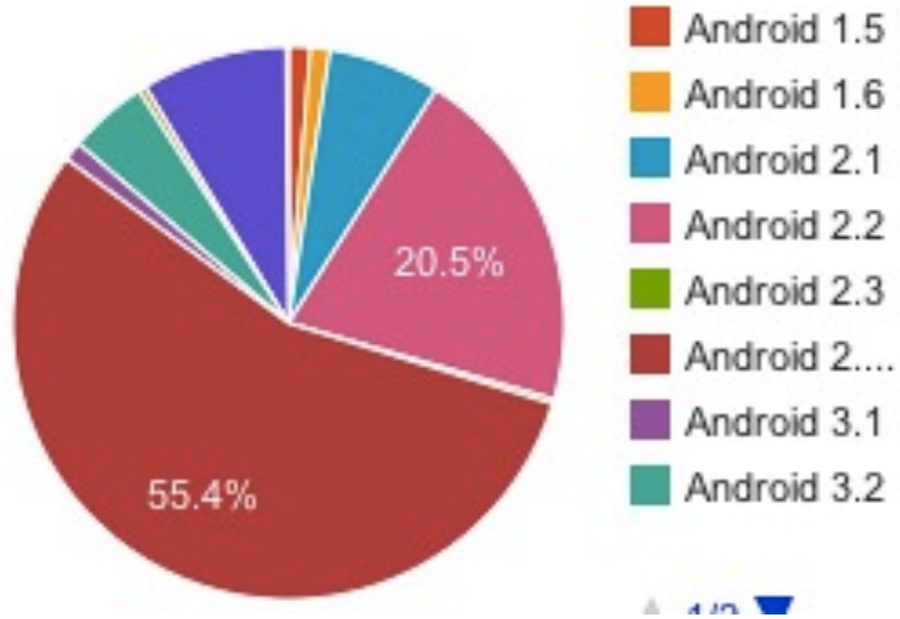
Gone are the days when websites had to be designed for a single screen size.

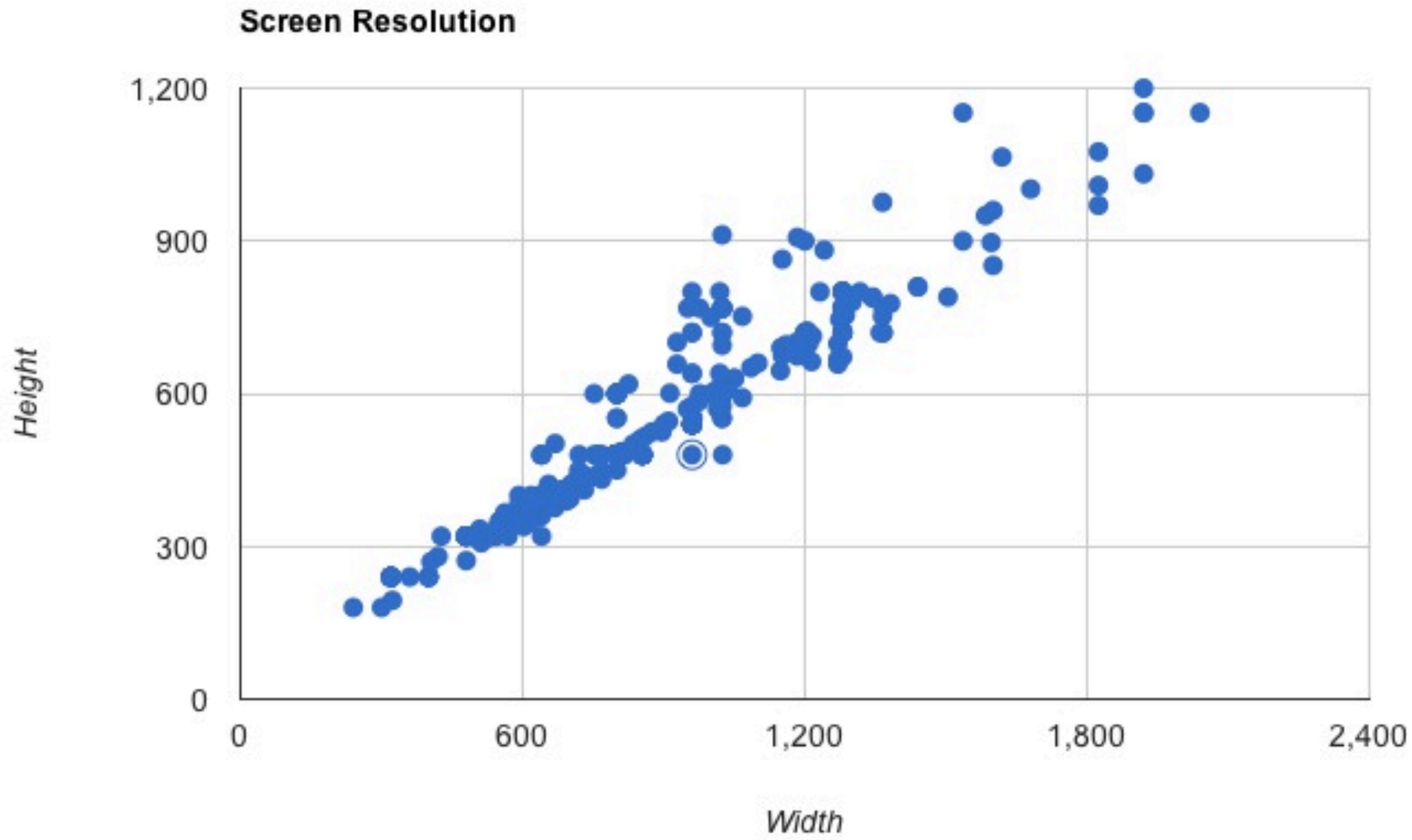


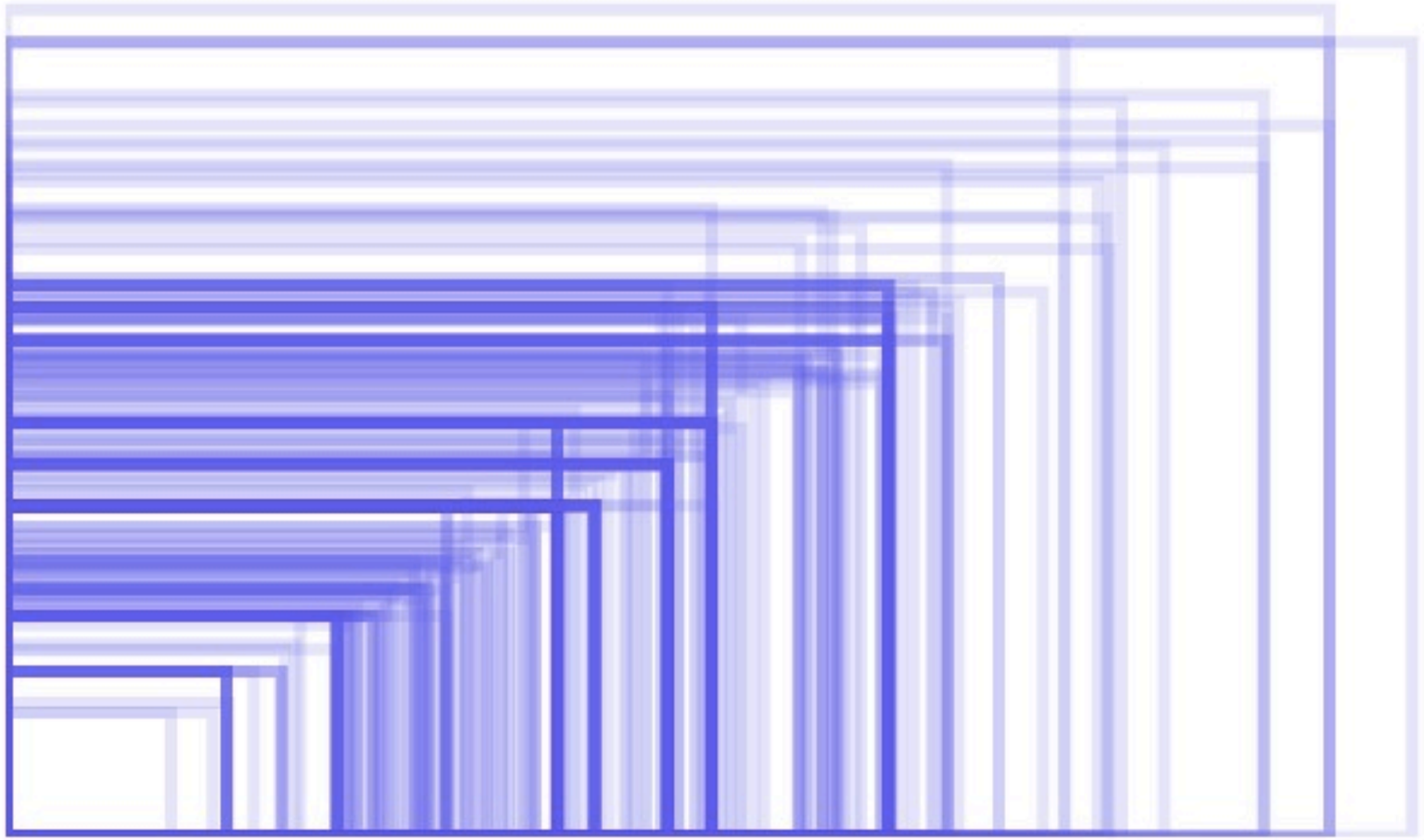
API levels April 2011

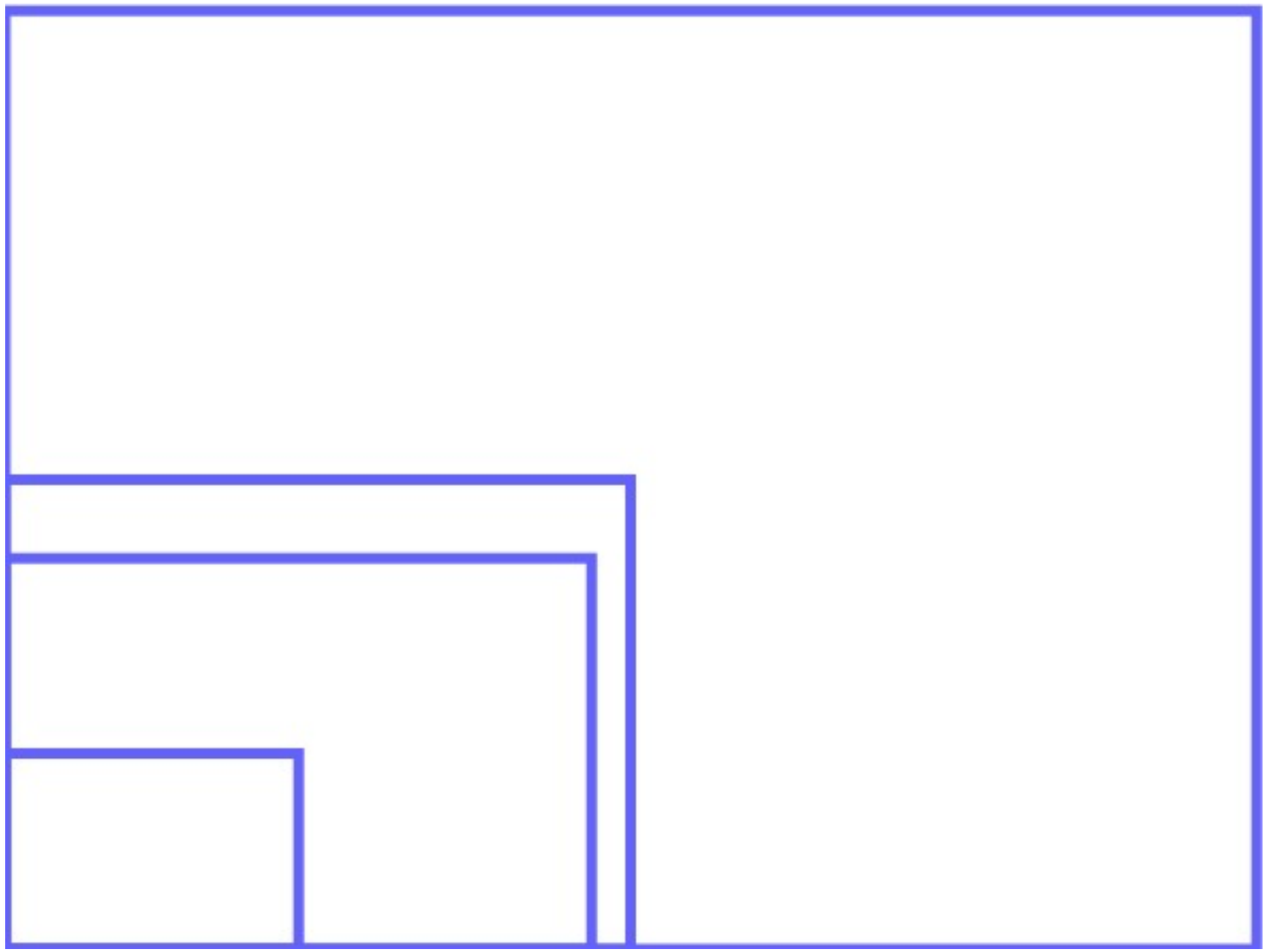


API levels April 2012









Fragment

- To create a dynamic and multi-pane user interface on Android, you need to encapsulate UI components and activity behaviors into modules that you can swap into and out of your activities.
- You can create these modules with the `Fragment` class, which behaves somewhat like a nested activity that can define its own layout and manage its own lifecycle.
- When a fragment specifies its own layout, it can be configured in different combinations with other fragments inside an activity to modify your layout configuration for different screen sizes

Fragment

- You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

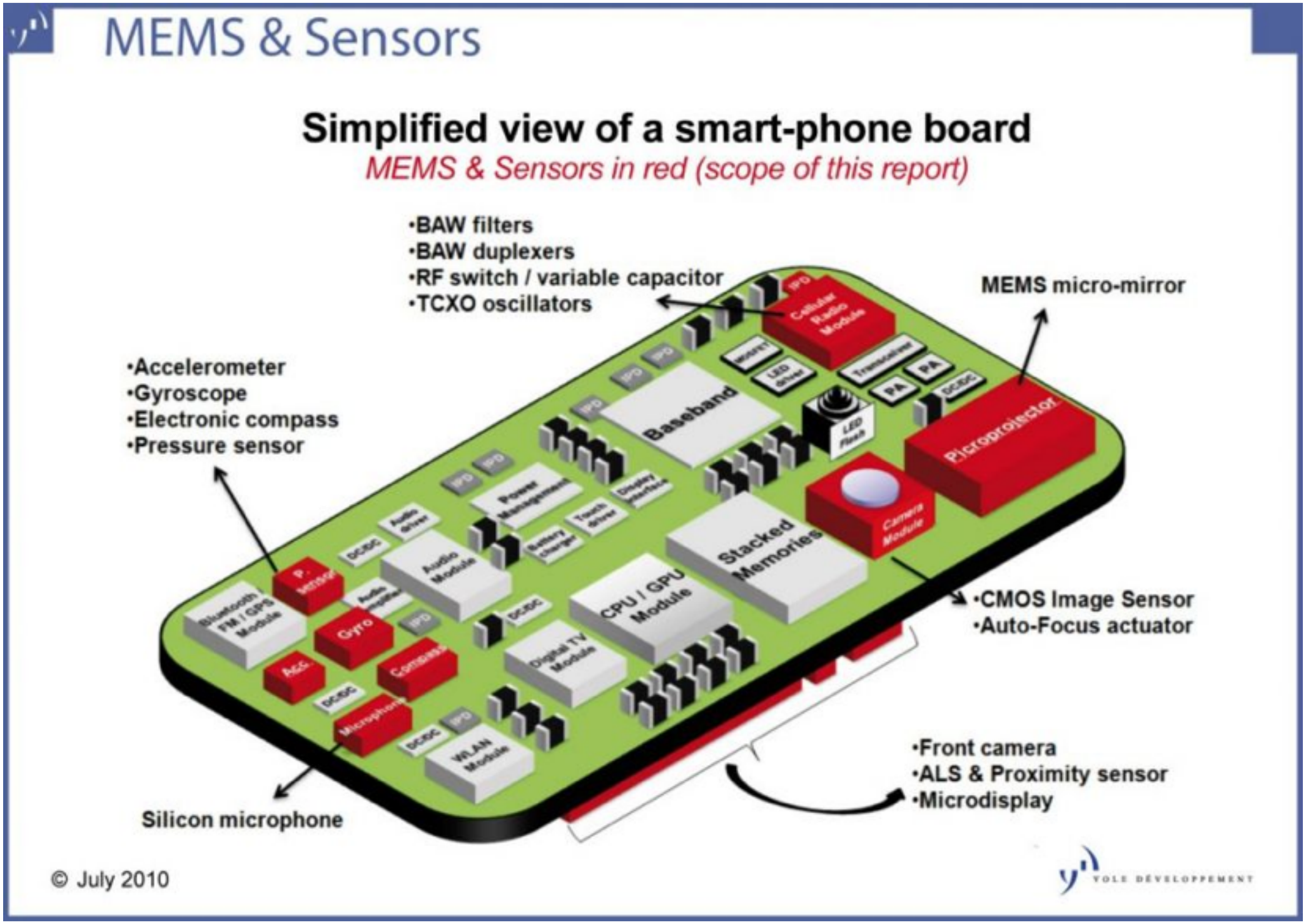
To start

```
import android.support.v4.app.Fragment;
import android.support.v4.app.FragmentManager;
```

extend the `FragmentActivity` class instead of the traditional `Activity` class.

Sensors

Sensors



Sensors

- Most Android-powered devices have built-in sensors that measure motion, orientation, and various environmental conditions. These sensors are capable of providing raw data with high precision and accuracy

Position Sensors

- Measure the physical position of a device.
- This category includes orientation sensors and magnetometers.

Environmental Sensors

- Measure various environmental parameters, such as ambient air temperature and pressure, illumination, and humidity.
- This category includes barometers, photometers, and thermometers.

Motion Sensors

Measure acceleration forces and rotational forces along three axes.

- This category includes accelerometers, gravity sensors, gyroscopes, and rotational vector sensors.

Sensor Framework

Determine which sensors are available on a device.

- Determine an individual sensor's capabilities, such as its maximum range, and resolution.

- Acquire raw sensor data and define the minimum rate at which you acquire sensor data.

- Register and unregister sensor event listeners that monitor sensor changes.

Sensor Manager

- You can use this class to create an instance of the sensor service.
- This class provides various methods for accessing and listing sensors.
- This class also provides several sensor constants that are used to report sensor accuracy, set data acquisition rates, and calibrate sensors.

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
```

```
private SensorManager mSensorManager;
...
mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
if
(mSensorManager.getDefaultSensor(Sensor.TYPE_MA
GNETIC_FIELD) != null){
  // Success! There's a magnetometer.
  }
else {
  // Failure! No magnetometer.
  }
```

Sensor

- You can use this class to create an instance of a specific sensor. This class provides various methods that let you determine a sensor's capabilities.

```
List<Sensor> deviceSensors =
mSensorManager.getSensorList(Sensor.TYPE_ALL)
;
```

SensorEvent

- The system uses this class to create a sensor event object, which provides information about a sensor event.
- A sensor event object includes the following information: the raw sensor data, the type of sensor that generated the event, the accuracy of the data, and the timestamp for the event.

SensorEventListener

- You can use this interface to create two callback methods that receive notifications (sensor events) when sensor values change or when sensor accuracy changes.

- `public final void onSensorChanged(SensorEvent event) {}`
- `public final void onAccuracyChanged(Sensor sensor, int accuracy) {}`

```java
public class SensorActivity extends Activity implements SensorEventListener {
  private SensorManager mSensorManager;
  private Sensor mLight;
  @Override
  public final void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mLight = mSensorManager.getDefaultSensor(Sensor.TYPE_LIGHT);
  }
  @Override
  public final void onAccuracyChanged(Sensor sensor, int accuracy) {
    // Do something here if sensor accuracy changes.
  }
  @Override
  public final void onSensorChanged(SensorEvent event) {
    // The light sensor returns a single value.
    // Many sensors return 3 values, one for each axis.
    float lux = event.values[0];
    // Do something with this sensor value.
  }
  @Override
  protected void onResume() {
    super.onResume();
    mSensorManager.registerListener(this, mLight, SensorManager.SENSOR_DELAY_NORMAL);
  }
  @Override
  protected void onPause() {
    super.onPause();
    mSensorManager.unregisterListener(this);
  }
```

Examples, examples, examples...

Finally

```
private SensorManager mSensorManager;
   ...
@Override
protected void onPause() {
   super.onPause();
   mSensorManager.unregisterListener(this);
}
```

Please Read

- http://developer.android.com/guide/topics/sensors/sensors_motion.html

Services

- http://developer.android.com/guide/components/services.html

Services

- A `Service` is an application component that can perform long-running operations in the background and does not provide a user interface

- For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background

Services

- Another application component can start a service and it will continue to run in the background even if the user switches to another application.
- Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC).

Two Forms

- **Started** by calling **startService( )**

- **Bound** by calling **bindService( )**

Started

- A service is "started" when an application component (such as an activity) starts it by calling `startService()`.
- Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
- Usually, a started service performs a single operation and does not return a result to the caller. For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

Bound

- A service is "bound" when an application component binds to it by calling `bindService()`.
- A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).
- A bound service runs only as long as another application component is bound to it. Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

Step 0: Declaring a service

```
<manifest ... >
  ...
  <application ... >
      <service
android:name=".ExampleService" />
      ...
  </application>
</manifest>
```

- Android:name is the only required attribute
- Declaring intent filters so others can start this service

Local service

- Do not provide intent filter
  - Intent needs to explicitly names the service class
- Ensure the service is private
  - android:exported=”false”

Service class

- onStartCommand( )
- onBind( )
- onCreate( )
- onDestory( )

Bound Services

- A bound service is the server in a client-server interface. A bound service allows components (such as activities) to bind to the service, send requests, receive responses, and even perform interprocess communication (IPC). A bound service typically lives only while it serves another application component and does not run in the background indefinitely.

Todo List

- implement the `onBind()` callback method. This method returns an `IBinder` object that defines the programming interface that clients can use to interact with the service.
- A client can bind to the service by calling `bindService()`. When it does, it must provide an implementation of `ServiceConnection`, which monitors the connection with the service.
- Multiple clients can connect to the service at once. However, the system calls your service's `onBind()` method to retrieve the `IBinder` only when the first client binds. The system then delivers the same `IBinder` to any additional clients that bind, without calling `onBind()` again.
- When the last client unbinds from the service, the system destroys the service

Example

Backup

Java

- Basic Java programming
  - Exceptions
  - Inner Class, Interface
- Advanced topics we will touch:
  - Java IO
  - Java Thread
  - Java Socket

Android Basics

- Credit goes to Google!

Objectives

- Mobile Application Development
- Intro to Android platform
- Platform architecture
- Application building blocks
- Development tools
- Textbook: Hello, Android