7CC003 Distributed And Mobile Computing

Android

What is Android?

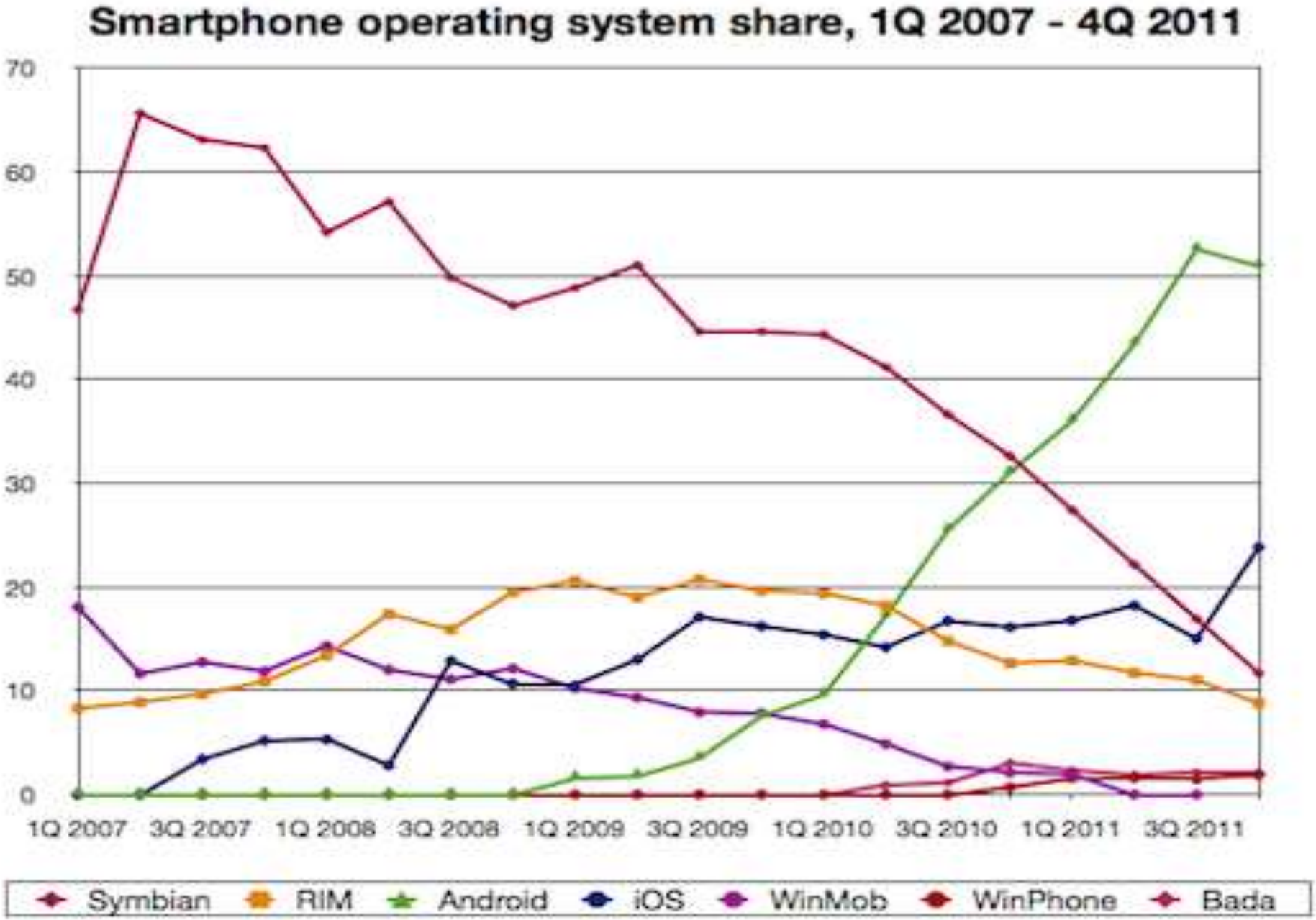
- Android is a software stack for mobile devices that includes an operating system, middleware and key applications. The Android SDK provides the tools and APIs necessary to begin developing applications on the Android platform using the Java programming language.

Brief History

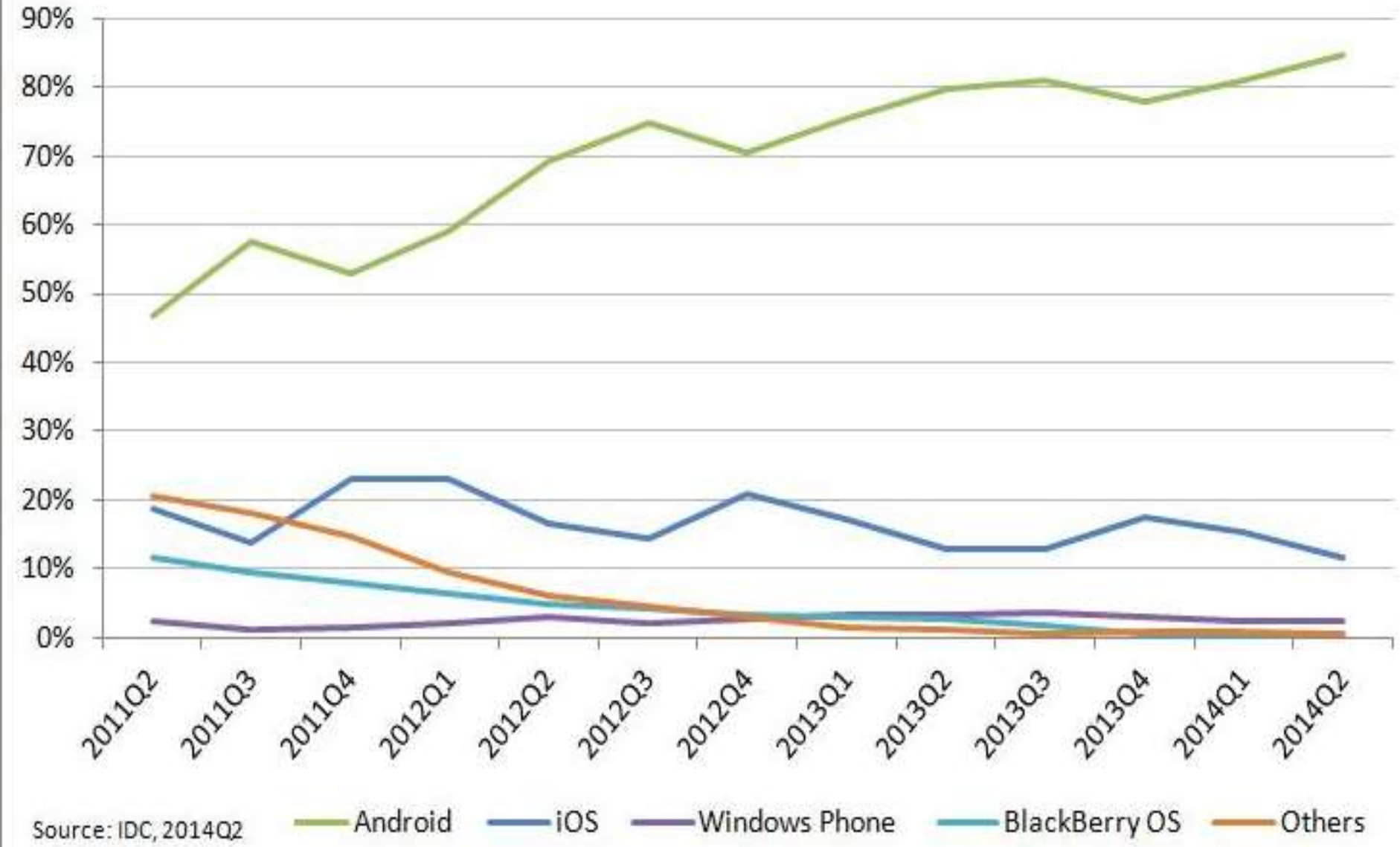
- 2005
  - Google acquires startup Android Inc. to start Android platform
  - Work on Dalvik VM begins
- 2007
  - Open Handset Alliance announced
  - Early look at SDK
- 2008
  - Google sponsors 1st Android Developer Challenge
  - T-Mobile G1 announced
  - SDK 1.0 released
  - Android released open source (Apache License)
  - Android Dev Phone 1 released

Brief History

- 2009
  - SDK 1.5 (Cupcake)
    - new soft keyboard with an "Autocomplete" feature
  - SDK 1.6 (Donut)
- 2010
  - SDK 2.0/2.0.1/2.1 (Éclair)
    - Nexus One released to the public
- 2011
  - SDK 3.0 Ice Cream Sandwich
- 2012
  - SDK 4.0 Jelly Bean
- 2013
  - SDK 4.4 KitKat
- 2015
  - SDK 5.0 Lollipop


Smartphone operating system share, 1Q 2007 - 4Q 2011
70
60
50
40
30
20
10
0
1Q 2007
3Q 2007
1Q 2008
3Q 2008
1Q 2009
3Q 2009
1Q 2010
3Q 2010
1Q 2011
3Q 2011
Symbian
RIM
Android
iOS
WinMob
WinPhone
Bada

Worldwide Smartphone OS Market Share
(Share in Unit Shipments)



Features

- **Application framework** enabling reuse and replacement of components
- **Dalvik virtual machine** optimized for mobile devices
- **Integrated browser** based on the open source WebKit engine

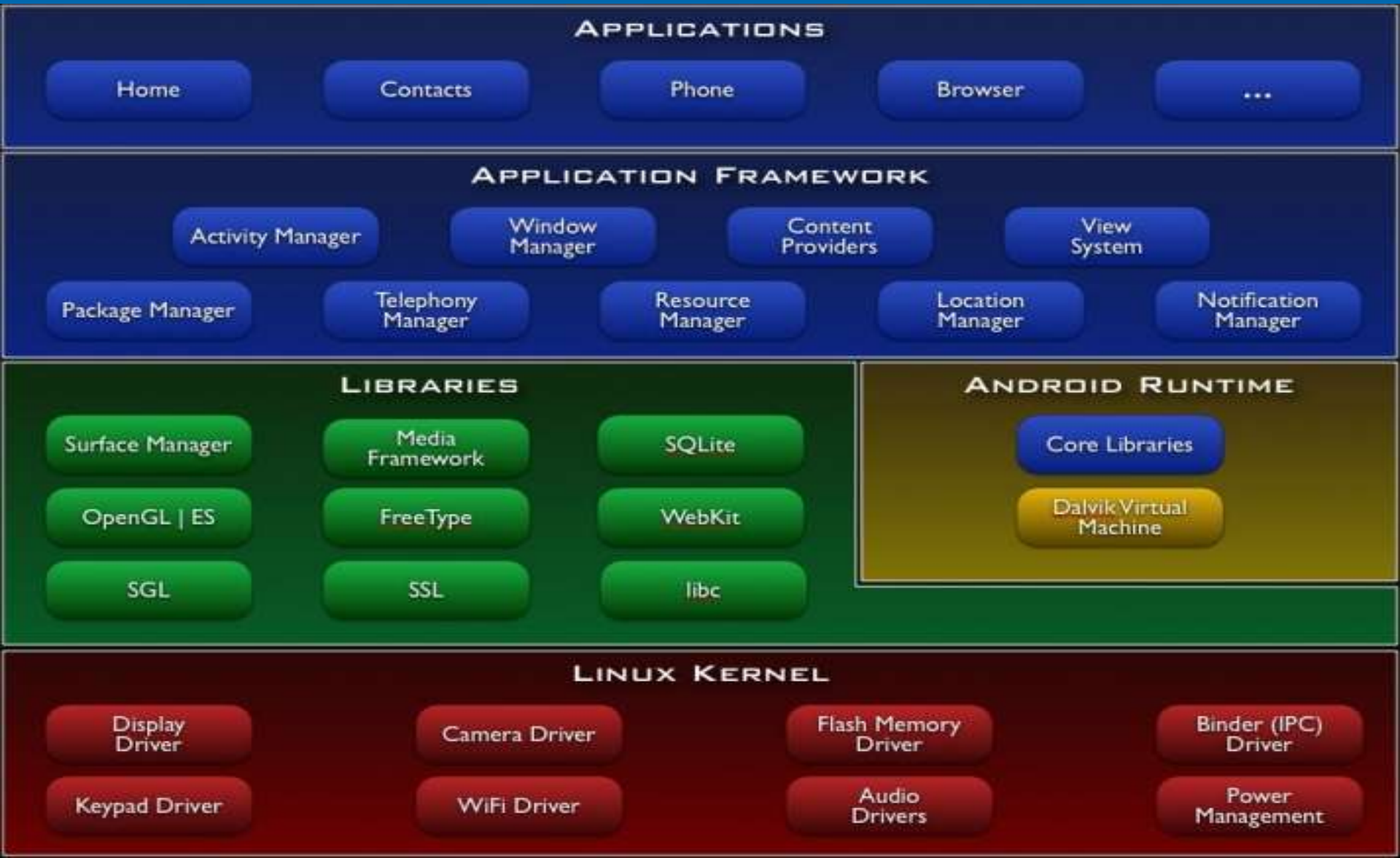
Features

- **Optimized graphics** powered by a custom 2D graphics library; 3D graphics based on the OpenGL ES 1.0 specification (hardware acceleration optional)
- **SQLite** for structured data storage
- **Media support** for common audio, video, and still image formats (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)

Features

- **GSM Telephony** (hardware dependent)
- **Bluetooth, EDGE, 3G, and WiFi** (hardware dependent)
- **Camera, GPS, compass, and accelerometer** (hardware dependent)
- **Rich development environment** including a device emulator, tools for debugging, memory and performance profiling, and a plugin for the Eclipse IDE. New Android Studio IDE currently in development.

Android Architecture



Applications

- Android ships with a set of core applications including an email client, SMS program, calendar, maps, browser, contacts, and others. All applications are written using the Java programming language.

Application Framework

- Developers have full access to the same framework APIs used by the core applications.
- Developers are free to take advantage of the device hardware, access location information, run background services, set alarms, add notifications to the status bar, and much, much more.

Application Framework

- A rich and extensible set of Views that can be used to build an application, including lists, grids, text boxes, buttons, and even an embeddable web browser
- Content Providers that enable applications to access data from other applications (such as Contacts), or to share their own data
- A Resource Manager, providing access to non-code resources such as localized strings, graphics, and layout files

Application Framework

- A Notification Manager that enables all applications to display custom alerts in the status bar
- An Activity Manager that manages the lifecycle of applications and provides a common navigation backstack

Libraries

- Android includes a set of C/C++ libraries used by various components of the Android system. These capabilities are exposed to developers through the Android application framework.

Libraries

- **System C library** - a BSD-derived implementation of the standard C system library (libc), tuned for embedded Linux-based devices
- **Media Libraries** - based on PacketVideo's OpenCORE; the libraries support playback and recording of many popular audio and video formats, as well as static image files, including MPEG4, H.264, MP3, AAC, AMR, JPG, and PNG
- **Surface Manager** - manages access to the display subsystem and seamlessly composites 2D and 3D graphic layers from multiple applications

Libraries

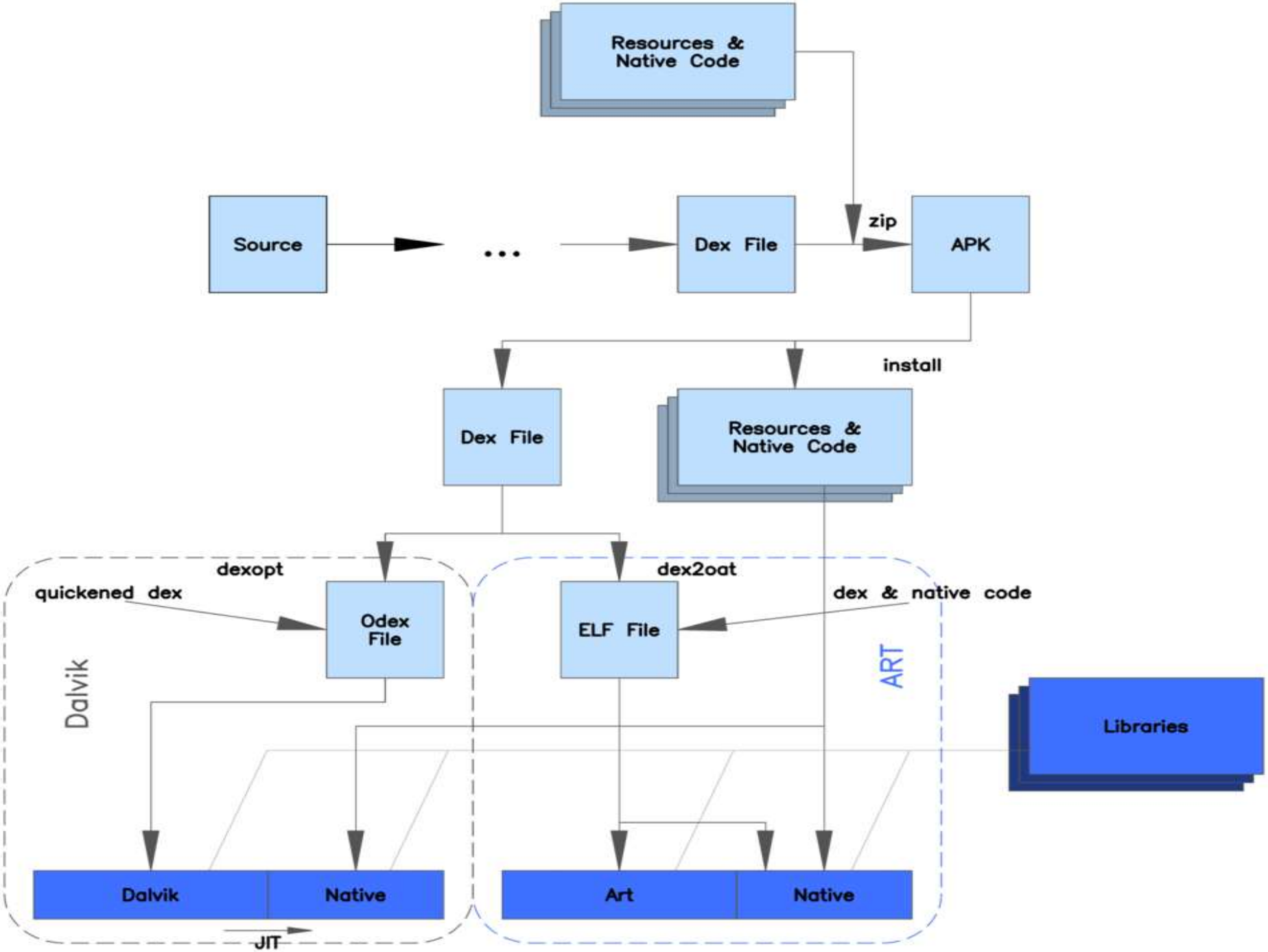
- **LibWebCore** - a modern web browser engine which powers both the Android browser and an embeddable web view
- **SGL** - the underlying 2D graphics engine
- **3D libraries** - an implementation based on OpenGL ES 1.0 APIs; the libraries use either hardware 3D acceleration (where available) or the included, highly optimized 3D software rasterizer

Libraries

- **FreeType** - bitmap and vector font rendering
- **SQLite** - a powerful and lightweight relational database engine available to all applications

Android Runtime

- Android includes a set of core libraries that provides most of the functionality available in the core libraries of the Java programming language.
- Every Android application runs in its own process, with its own instance of the Dalvik virtual machine. Dalvik has been written so that a device can run multiple VMs efficiently. The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint.
- Lollipop has new runtime Android Runtime (ART)


Resources &
Native Code
Source
...
Dex File
zip
APK
install
Dex File
Resources &
Native Code
dexopt
quickened dex
Odex
File
dex2oat
ELF File
dex & native code
Dalvik
ART
Libraries
Dalvik
Native
Art
Native
JIT

Linux Kernel

- Android relies on Linux version 2.6 for core system services such as security, memory management, process management, network stack, and driver model. The kernel also acts as an abstraction layer between the hardware and the rest of the software stack.

Components of Android Applications

- Activity
  - Main component of an application
  - Generally, visible to the user
  - Example: UI for browsing music, selecting songs
- ContentProvider
  - Stores (tabular) data and makes it available to other applications
  - Example: Store the music and metadata
    - Gives other apps access
    - Music recommendation
- Service
  - Handles background work and ongoing tasks
  - Example: plays music in the background
- BroadcastReceiver
  - Receives events from the phone
  - Example: Pause music if headphones are unplugged

Intents

- Android uses Intents to accomplish tasks.
- Intents: abstract description of an operation to be performed.
- Action: The general action to be performed
- Data: What should be operated on
- Component: What will perform the action (optional)

Intent Examples

- ACTION_VIEW http://www.wlv.ac.uk
- Launch an application to view this URL.

- ACTION_DIAL tel: 01902 321000
- Launch an application to call this phone number.

- Note –these intents do not specify the component which should handle them.
- We call these *implicit intents*
- *Explicit intents include the target component*

The AndroidManifest.xml File

- Every application must have an AndroidManifest.xml file (with precisely that name) in its root directory.
- The manifest presents essential information about the application to the Android system, information the system must have before it can run any of the application's code.

The Manifest file

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application (activities, services, broadcast receivers, and content providers)
- It determines which processes will host application components.

The Manifest file

- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It declares the minimum level of the Android API that the application requires.
- It lists the libraries that the application must be linked against.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
   xmlns:android="http://schemas.android.com/apk/res/and
   roid"
      package="test.test"
      android:versionCode="1"
      android:versionName="1.0">
    <application android:icon="@drawable/icon"
   android:label="@string/app_name">
        <activity android:name=".main"
                  android:label="@string/app_name">
            <intent-filter>
                <action
   android:name="android.intent.action.MAIN" />
                <category
   android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="7" />
</manifest>
```

Sample Manifest file

Applications Components

- Activities
- Services
- Content Providers
- Broadcast receivers

Activities

- An *activity* presents a visual user interface for one focused endeavor the user can undertake.
- An application might consist of just one activity or, like a text messaging application, it may contain several.
- Each activity is given a default window to draw in.
- The visual content of the window is provided by a hierarchy of views.

Activities

- Launch an Activity by calling startActivity(Intent)
- Subactivities: startActivityForResult
- Takes an Intent as input, as well as an integer code.
- When subactivity exits, returns a result code.
- Original activity's onActivityResultmethod is called.
- Note –this is asynchronous (non-blocking).

- •Activities form a stack
- New activities appear at the top of the stack (i.e. on screen)
- Pressing back button goes to previous activity (usually)

Services

- A *service* doesn't have a visual user interface, but rather runs in the background for an indefinite period of time.
- For example, a service might play background music as the user attends to other matters.

Broadcast receivers

- A *broadcast receiver* is a component that does nothing but receive and react to broadcast announcements.
- Events or other applications could initiate broadcasts .
- An application can have any number of broadcast receivers to respond to any announcements it considers important.

Content providers

- A *content provider* makes a specific set of the application's data available to other applications.
- The data can be stored in the file system, in an SQLite database, or in any other manner that makes sense.
- Applications do not call these methods directly. Rather they use a ContentResolver object and call its methods instead.

Activating components

- ContentResolver
  - Content providers are activated when they're targeted by a request from a ContentResolver.
- Intents
  - Activities, services, and broadcast receivers are activated by asynchronous messages called *intents*.

Intents

- An intent is an Intent object that holds the content of the message.
- For activities and services, it names the action being requested and specifies the URI of the data to act on.
- For example, it might convey a request for an activity to present an image to the user or let the user edit some text.

Intents

- For broadcast receivers, the Intent object names the action being announced.
- For example, it might announce to interested parties that the camera button has been pressed.

Shutting down components

- A content provider is active only while it's responding to a request from a ContentResolver.
- A broadcast receiver is active only while it's responding to a broadcast message.
- So there's no need to explicitly shut down these components.

Shutting down components

- An activity can be shut down by calling its finish() method. One activity can shut down another activity by calling finishActivity().
- A service can be stopped by calling its stopSelf() method, or by calling Context.stopService().

Shutting down components

- Components might also be shut down by the system when they are no longer being used or when Android must reclaim memory for more active components

Intent filters

- An Intent object can explicitly name a target component.
- If it does, Android finds that component (based on the declarations in the manifest file) and activates it.
- But if a target is not explicitly named, Android must locate the best component to respond to the intent. It does so by comparing the Intent object to the *intent filters* of potential targets.

Intent filters

- A component's intent filters inform Android of the kinds of intents the component is able to handle. Like other essential information about the component, they're declared in the manifest file.

Example

```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity android:name="com.example.project.FreneticActivity"
                  android:icon="@drawable/small_pic.png"
                  android:label="@string/freneticLabel"
                  . . .  >
            <intent-filter . . . >
                <action android:name="android.intent.action.MAIN" />
                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter . . . >
                <action android:name="com.example.project.BOUNCE" />
                <data android:mimeType="image/jpeg" />
                <category android:name="android.intent.category.DEFAULT"
/>
            </intent-filter>
        </activity>
        . . .
    </application>
</manifest>
```

First Filter

- The first filter in the example — the combination of the action "android.intent.action.MAIN" and the category "android.intent.category.LAUNCHER" — is a common one.
- It marks the activity as one that should be represented in the application launcher, the screen listing applications users can launch on the device.
- In other words, the activity is the entry point for the application, the initial one users would see when they choose the application in the launcher.

Second Filter

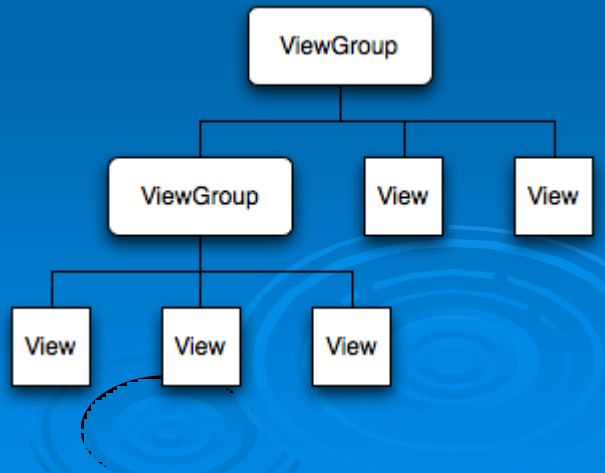
- The second filter declares an action that the activity can perform on a particular type of data.

Activities

- An *activity* presents a visual user interface.
- Each activity is given a default window to draw in.
- The visual content of the window is provided by a hierarchy of views.
- Each view controls a particular rectangular space within the window.

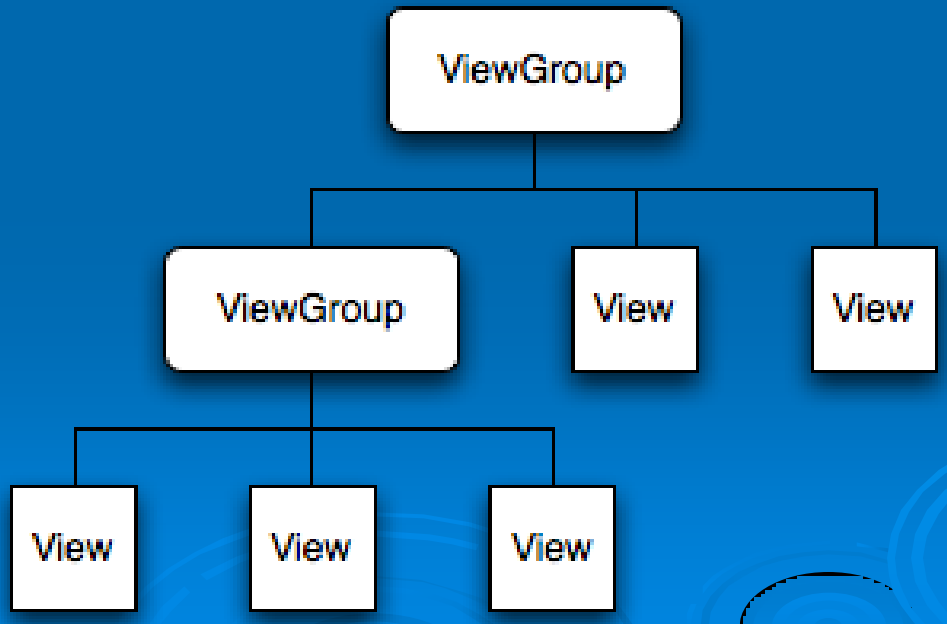
Hierarchy of Views

- Parent views contain and organize the layout of their children.
- Leaf views (those at the bottom of the hierarchy) draw in the rectangles they control and respond to user actions directed at that space.



View Hierarchy

- A view hierarchy is placed within an activity's window by the Activity.setContentView() method.



View Group

- A ViewGroup is a special view that can contain other views (called children.)
- The view group is the base class for layouts and views container
- Some layout paramemeters:

  fill_parent, match_parent, wrap_content

Widgets

- The View class serves as the base for subclasses called "widgets," which offer fully implemented UI objects, like text fields and buttons.
- A widget is a View object that serves as an interface for interaction with the user.

Widgets: Ready-made views in Android

- buttons,
- text fields,
- scroll bars,
- menu items,
- check boxes,
- more

Declaring Layout

- Your layout is the architecture for the user interface in an Activity. It defines the layout structure and holds all the elements that appear to the user.
- You can declare your layout in two ways:
  - **Declare UI elements in XML**
  - **Instantiate layout elements at runtime**

Declare UI elements in XML

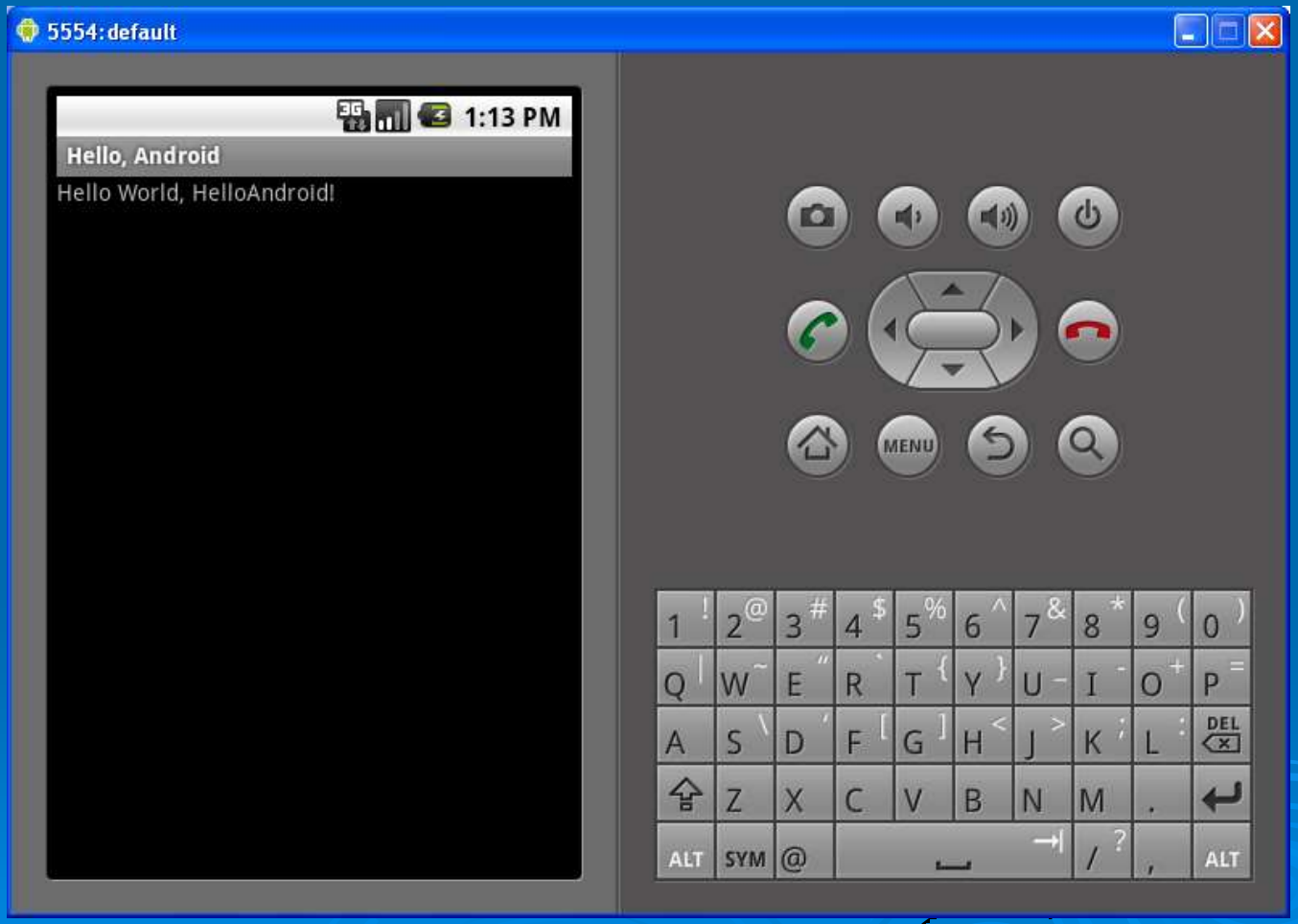
- Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

res/values/strings.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World,
  HelloAndroid!</string>
    <string name="app_name">Hello,
  Android</string>
</resources>
```

Hello World, HelloAndroid!



Instantiate layout elements at runtime

- Your application can create View and ViewGroup objects (and manipulate their properties) programmatically.

src/[package name]/[activity name].java

```
package com.example.helloandroid;

   import android.app.Activity;
   import android.os.Bundle;
   import android.widget.TextView;

   public class HelloAndroid extends Activity {
      /** Called when the activity is first created. */
      @Override
      public void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          TextView tv = new TextView(this);
          tv.setText("Hello World, HelloAndroid!");
          setContentView(tv);
      }
   }
```

Application Resources

- Common files in res/ directory:
  - drawable/icon.png: Icon for the program in launcher
  - layout/main.xml: User interface for main Activity
  - values/strings.xml: Any Strings appearing in UI

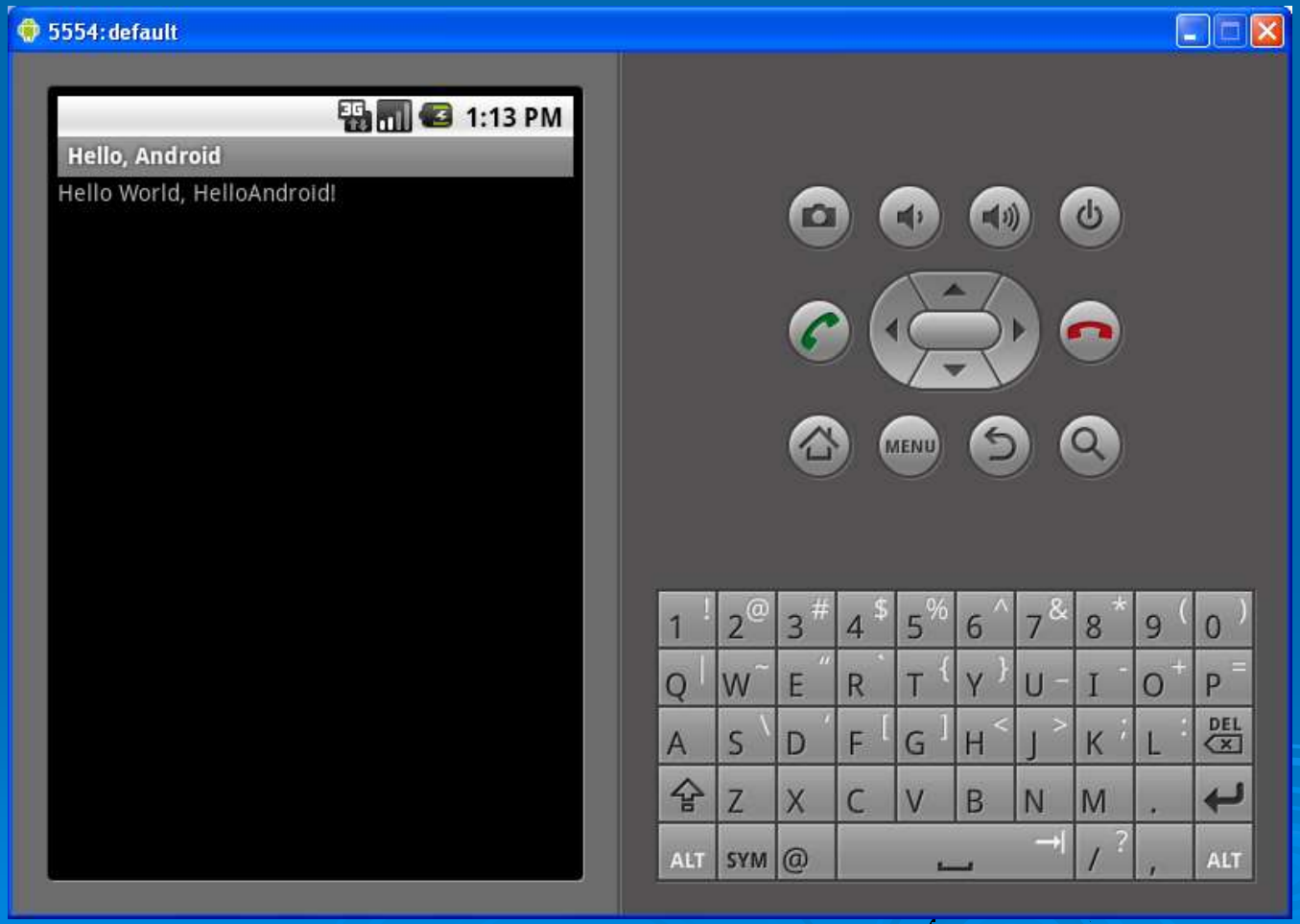
- Resources –separate program logic from "other stuff"

  Strings, images, UI layouts

res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/
  android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

Hello Android



XML files

- 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.
- Your UI descriptions are external to your application code, which means that you can modify or adapt it without having to modify your source code and recompile.

Modified main.xml (1)

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

     <LinearLayout
       android:orientation="horizontal"
       android:layout_width="fill_parent"
       android:layout_height="fill_parent"
       android:layout_weight="1">
       <TextView
          android:text="red"
          android:gravity="center_horizontal"
          android:background="#aa0000"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
```

-

```
       <TextView
          android:text="green"
          android:gravity="center_horizontal"
          android:background="#00aa00"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
       <TextView
          android:text="blue"
          android:gravity="center_horizontal"
          android:background="#0000aa"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
       <TextView
          android:text="yellow"
          android:gravity="center_horizontal"
          android:background="#aaaa00"
          android:layout_width="wrap_content"
          android:layout_height="fill_parent"
          android:layout_weight="1"/>
    </LinearLayout>
```

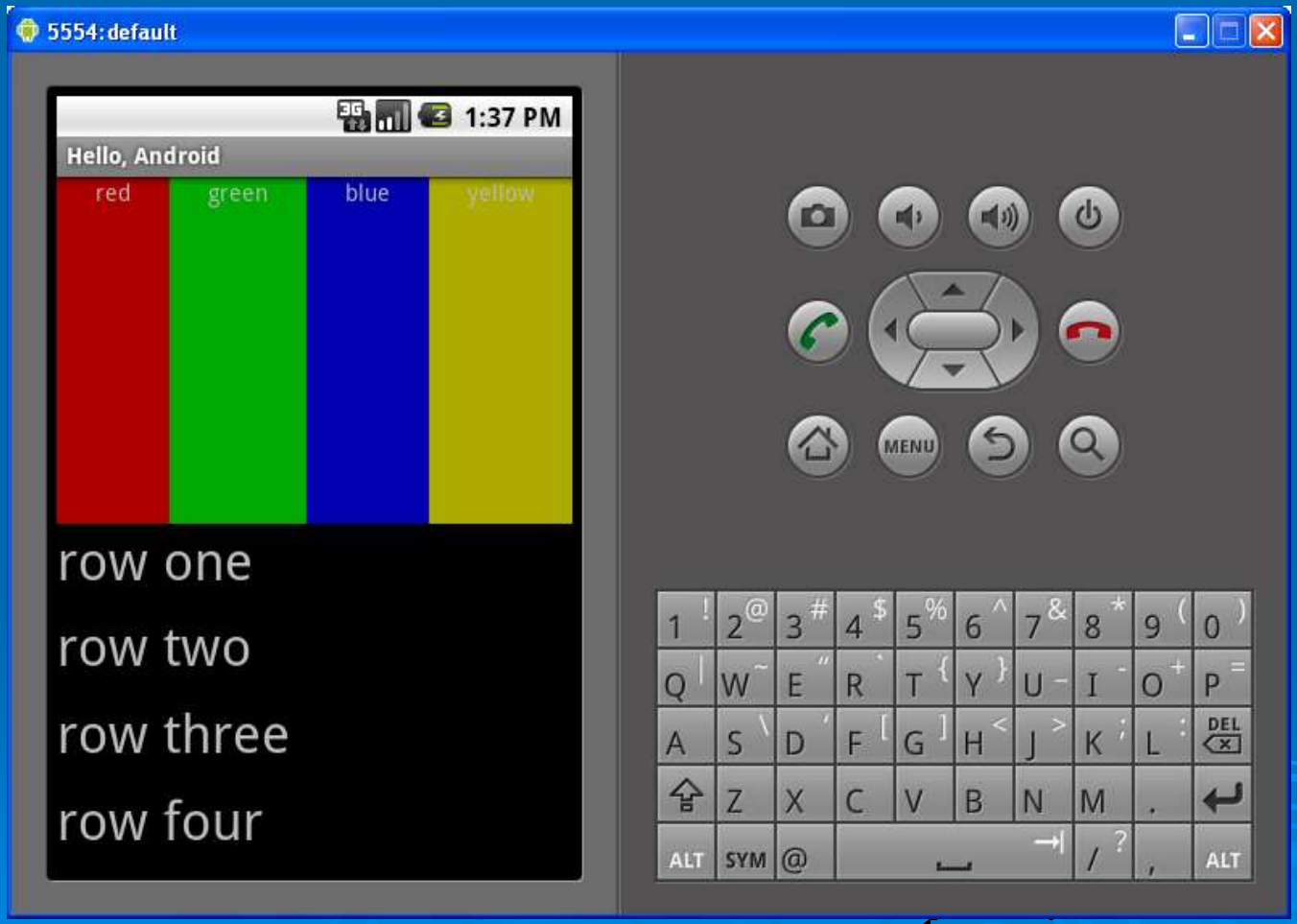
Modified main.xml (2)

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">
    <TextView
        android:text="row one"
        android:textSize="15pt"
        android:layout_width="fill_pare
  nt"
        android:layout_height="wrap_con
  tent"
        android:layout_weight="1"/>
    <TextView
        android:text="row two"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_con
  tent"
        android:layout_weight="1"/>
```

-
```
    <TextView
        android:text="row three"
        android:textSize="15pt"
        android:layout_width="fill_par
ent"
        android:layout_height="wrap
_content"
        android:layout_weight="1"/>
    <TextView
        android:text="row four"
        android:textSize="15pt"
        android:layout_width="fill_par
ent"
        android:layout_height="wrap
_content"
        android:layout_weight="1"/>
  </LinearLayout>

</LinearLayout>
```

Linear Layout



Modified main.xml

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

     <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        …
```

Identify the changes in main.xml

End