Lecture 11: Linux Programming on Android and Proxy

Lecturer: Prof. Zichen Xu

Introduction

- GTK+ (GIMP toolkit) : A library for creating graphical user interfaces(GUI)

- One example developed with GTK+

- Other GUI software in Linux

Basic Knowledge of GTK+

- GTK is essentially an object oriented application programmers interface (API). Although written completely in C, it is implemented using the idea of classes and callback functions (pointers to functions).

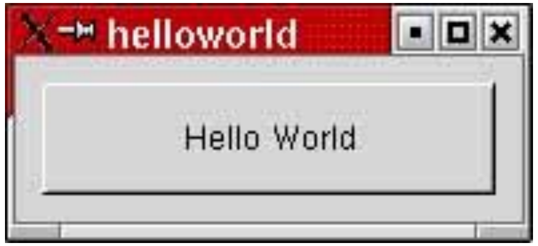
- GLib: A third component. It contains a few replacements for some standard calls, as well as some additional functions for handling linked lists, etc.

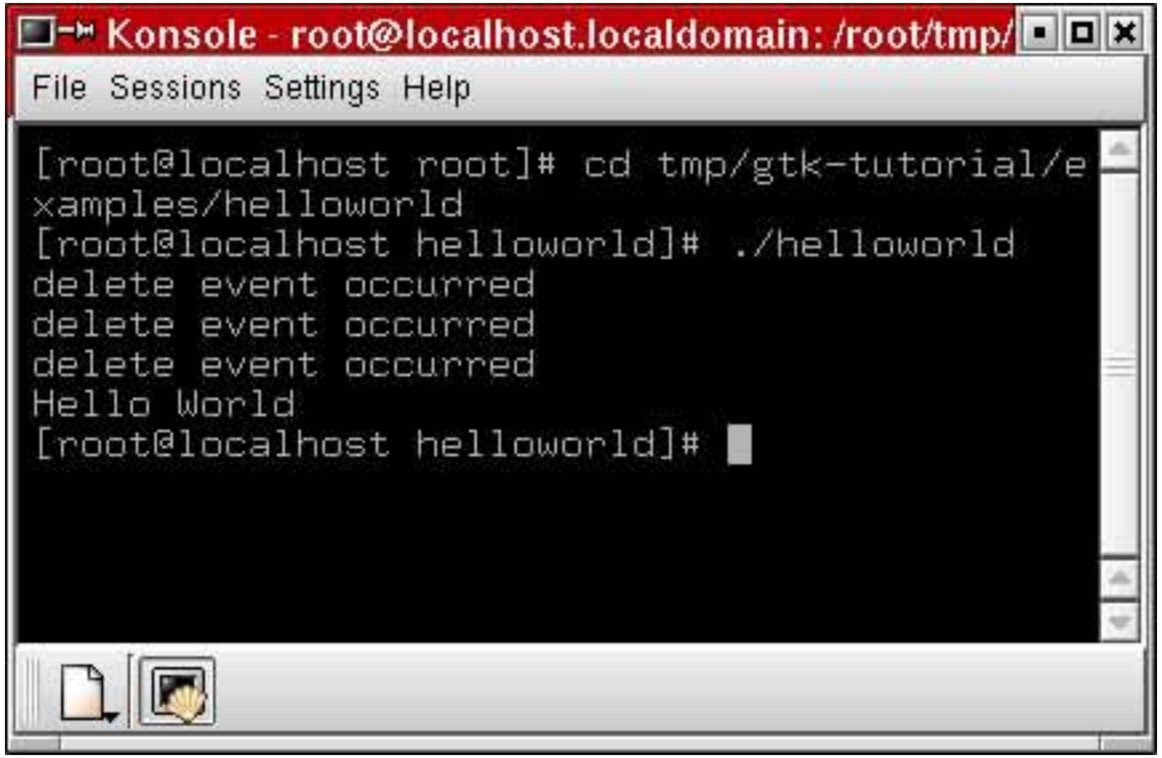
Example: Hello World!

```
• #include <gtk/gtk.h>

• void hello( GtkWidget *widget, gpointer   data ){
•     g_print ("Hello World\n");
• }

• gint delete_event( GtkWidget *widget, GdkEvent  *event, gpointer   data ){
•     g_print ("delete event occurred\n");
•     return(TRUE);
• }

• void destroy( GtkWidget *widget, gpointer   data ){
•     gtk_main_quit();
• }
```

```
int main( int   argc, char *argv[] ){
    GtkWidget *window;
    GtkWidget *button;
    gtk_init(&argc, &argv);
    window = gtk_window_new (GTK_WINDOW_TOPLEVEL);
    gtk_signal_connect (GTK_OBJECT (window), "delete_event",
                        GTK_SIGNAL_FUNC (delete_event), NULL);
    gtk_signal_connect (GTK_OBJECT (window), "destroy",
                        GTK_SIGNAL_FUNC (destroy), NULL);
    gtk_container_set_border_width (GTK_CONTAINER (window), 10);
    button = gtk_button_new_with_label ("Hello World");
    gtk_signal_connect (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (hello), NULL);
    gtk_signal_connect_object (GTK_OBJECT (button), "clicked",
                        GTK_SIGNAL_FUNC (gtk_widget_destroy),
                        GTK_OBJECT (window));
    gtk_container_add (GTK_CONTAINER (window), button);
    gtk_widget_show (button);
    gtk_widget_show (window);
    gtk_main ();
    return(0);}
```

Output of Example

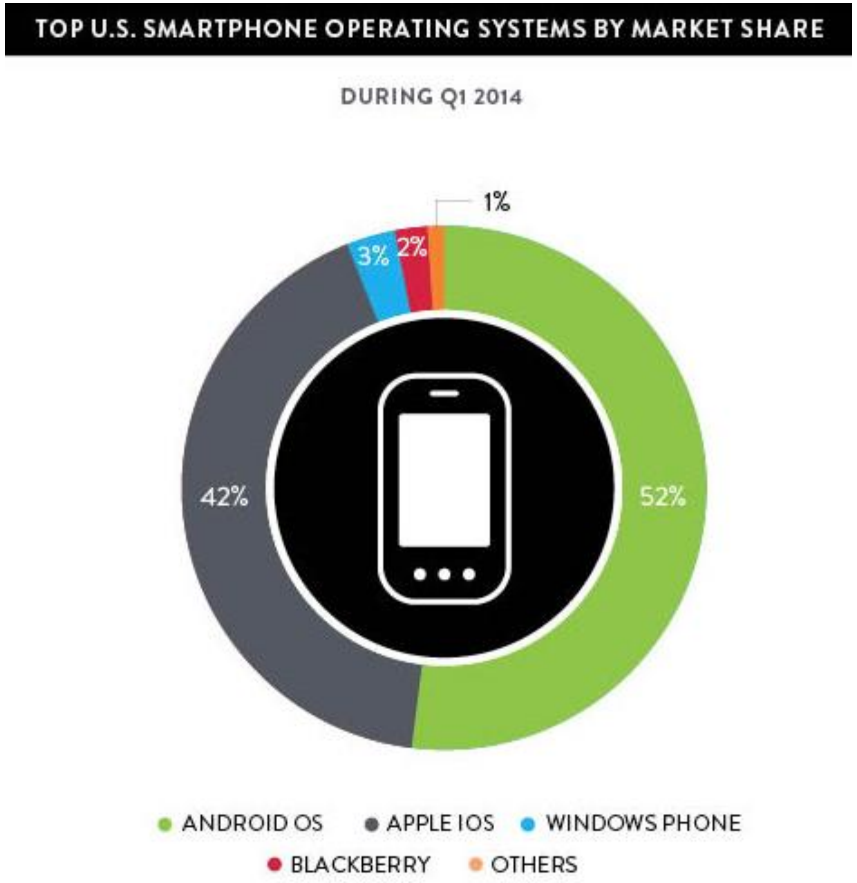




```
[root@localhost root]# cd tmp/gtk-tutorial/e
xamples/helloworld
[root@localhost helloworld]# ./helloworld
delete event occurred
delete event occurred
delete event occurred
Hello World
[root@localhost helloworld]# 
```

Introduction to Android

- Popular mobile device OS: 52% of U.S. smartphone market [8]
- Developed by Open Handset Alliance, led by Google
- Google claims 900,000 Android device activations [9]



Source: [8]

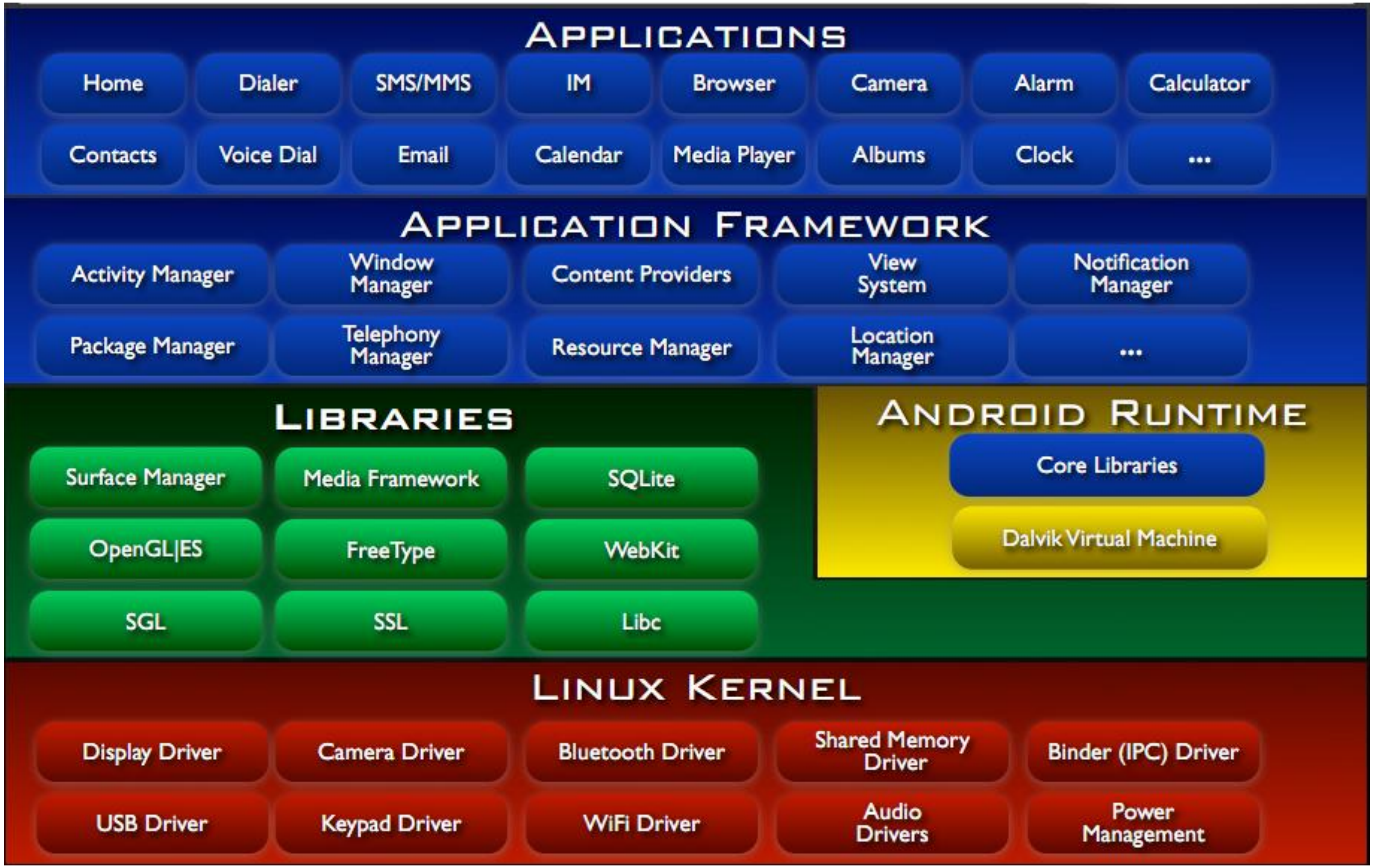
What is Android

- Android is an operating system for mobile devices such as *smartphones* and *tablet* computers. It is developed by the Open Handset Alliance led by Google.

- Android has beaten Apple iOS, being the leading mobile operating system from first quarter of 2011

- Version: Android 1.0, 1.1 to 1.5 (Cupcake), 1.6 (Donut), 2.0/2.1 (Eclair), 2.2 (Froyo), **2.3 (Gingerbread)**, to **3.0 (Honeycomb**), **4.0 (Ice Cream Sandwich)**, 5.0 (Lollipop) Marshmallow, Nougat, Oreo.

Android Architecture



APPLICATIONS

Home, Dialer, SMS/MMS, IM, Browser, Camera, Alarm, Calculator

Contacts, Voice Dial, Email, Calendar, Media Player, Albums, 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

OpenGL|ES, FreeType, WebKit

SGL, SSL, Libc

ANDROID RUNTIME

Core Libraries

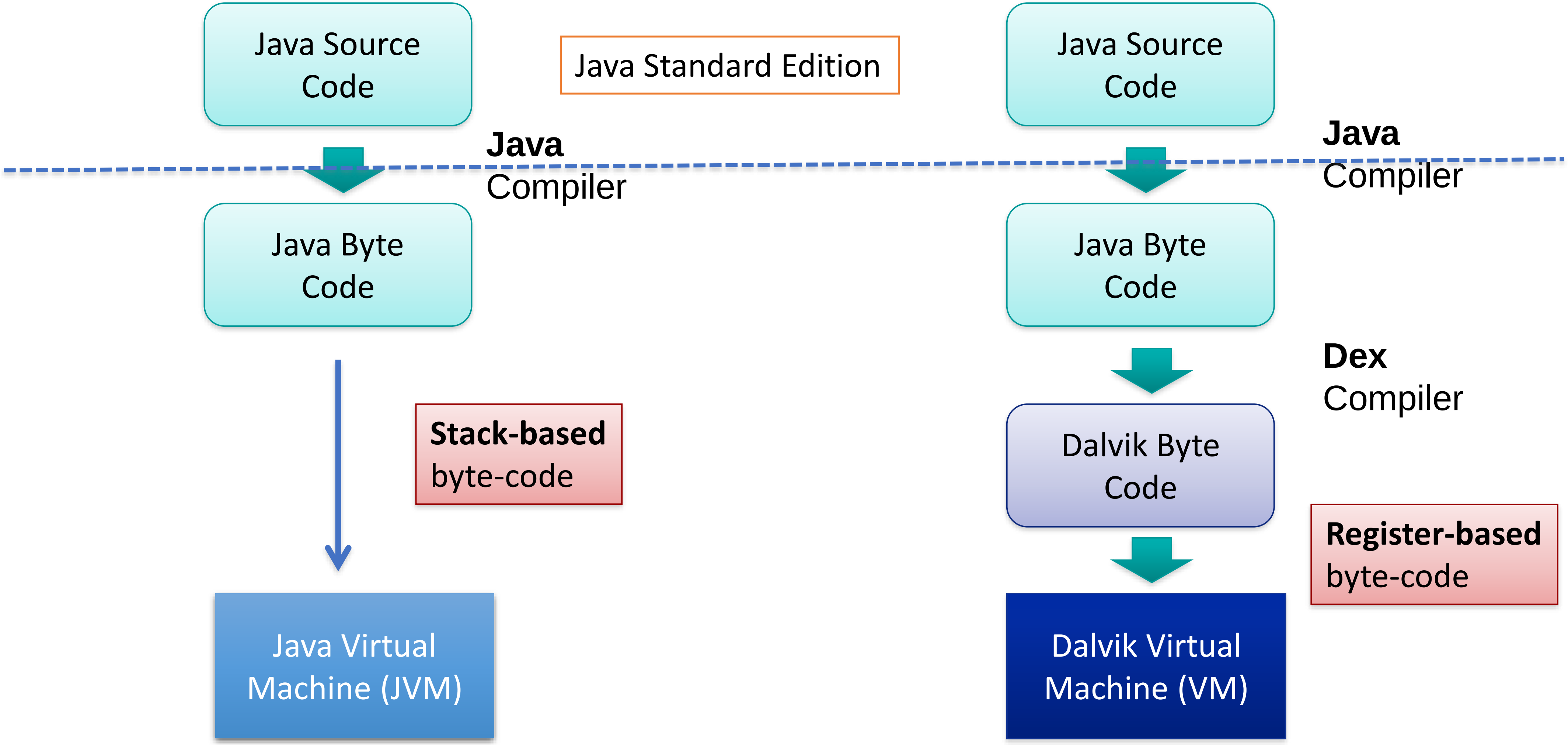
Dalvik Virtual Machine

LINUX KERNEL

Display Driver, Camera Driver, Bluetooth Driver, Shared Memory Driver, Binder (IPC) Driver

USB Driver, Keypad Driver, WiFi Driver, Audio Drivers, Power Management

Dalvik Java Virtual Machine (JVM)



Java Standard Edition

Java Source Code

Java Compiler

Java Byte Code

Stack-based byte-code

Java Virtual Machine (JVM)

Java Source Code

Java Compiler

Java Byte Code

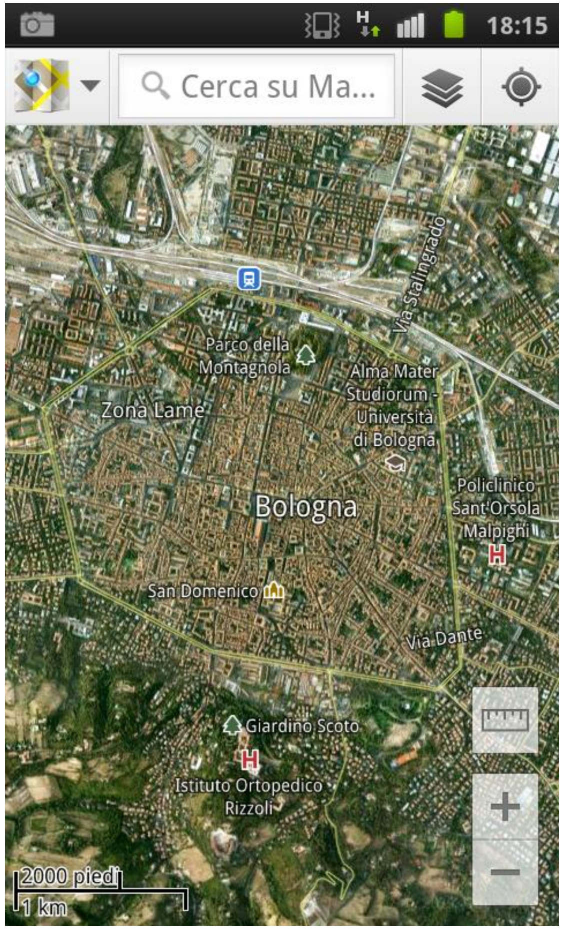
Dex Compiler

Dalvik Byte Code

Register-based byte-code

Dalvik Virtual Machine (VM)

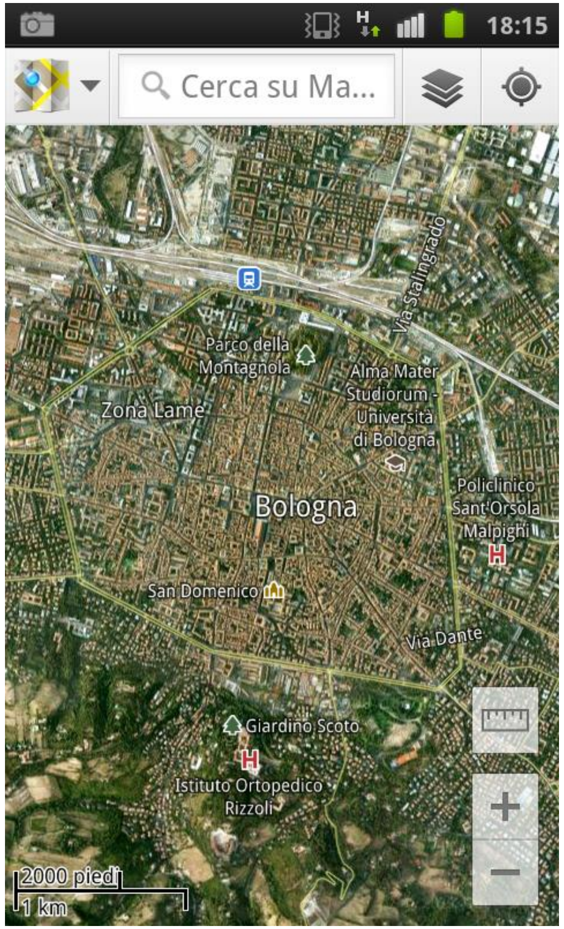
Android Applications Design



APPLICATION **DESIGN**:

- **GUI** Definition
- **Events** Management
- Application **Data** Management
- **Background** Operations
- **User** Notifications

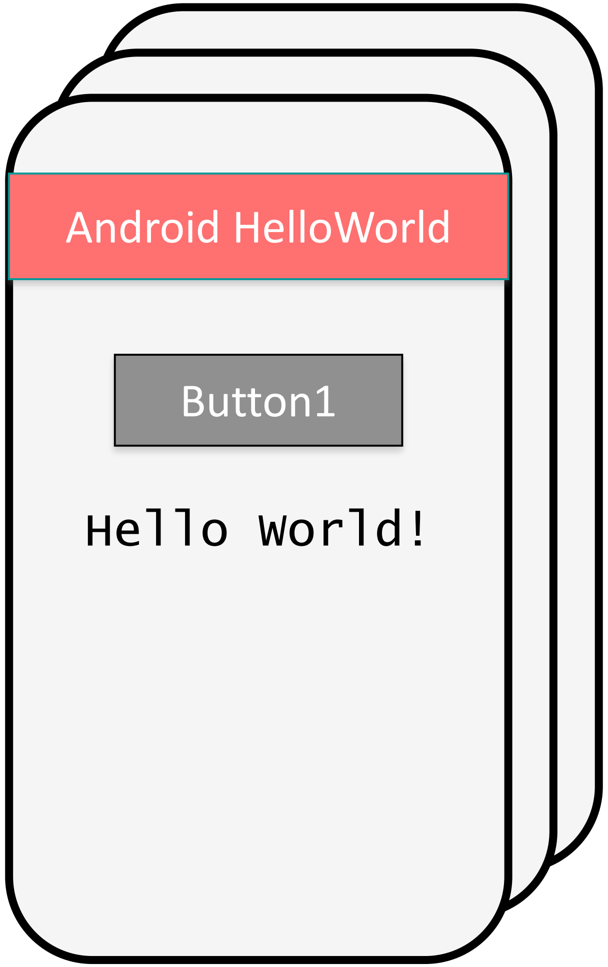
Android Applications Design



APPLICATION **COMPONENTS**

- **Activities**
- **Intents**
- **Services**
- **Content Providers**
- **Broadcast Receivers**

Android Components: Activities



- An **Activity** corresponds to a **single screen** of the **Application**.
- An Application can be composed of *multiples screens* (Activities).
- The **Home Activity** is shown when the user launches an application.
- Different activities can exhange information one with each other.

Android Components: Activities

- Each activity is composed by a list of *graphics components*.
- Some of these components (also called **Views**) can interact with the user by handling **events** (e.g. Buttons).
- Two ways to build the graphic interface:

PROGRAMMATIC APPROACH

Example:

```
Button button=new Button (this);
TextView text= new TextView();
text.setText(“Hello world”);
```

Android Components: Activities

- Each activity is composed by a list of *graphics components*.
- Some of these components (also called **Views**) can interact with the user by handling **events** (e.g. Buttons).
- Two ways to build the graphic interface:

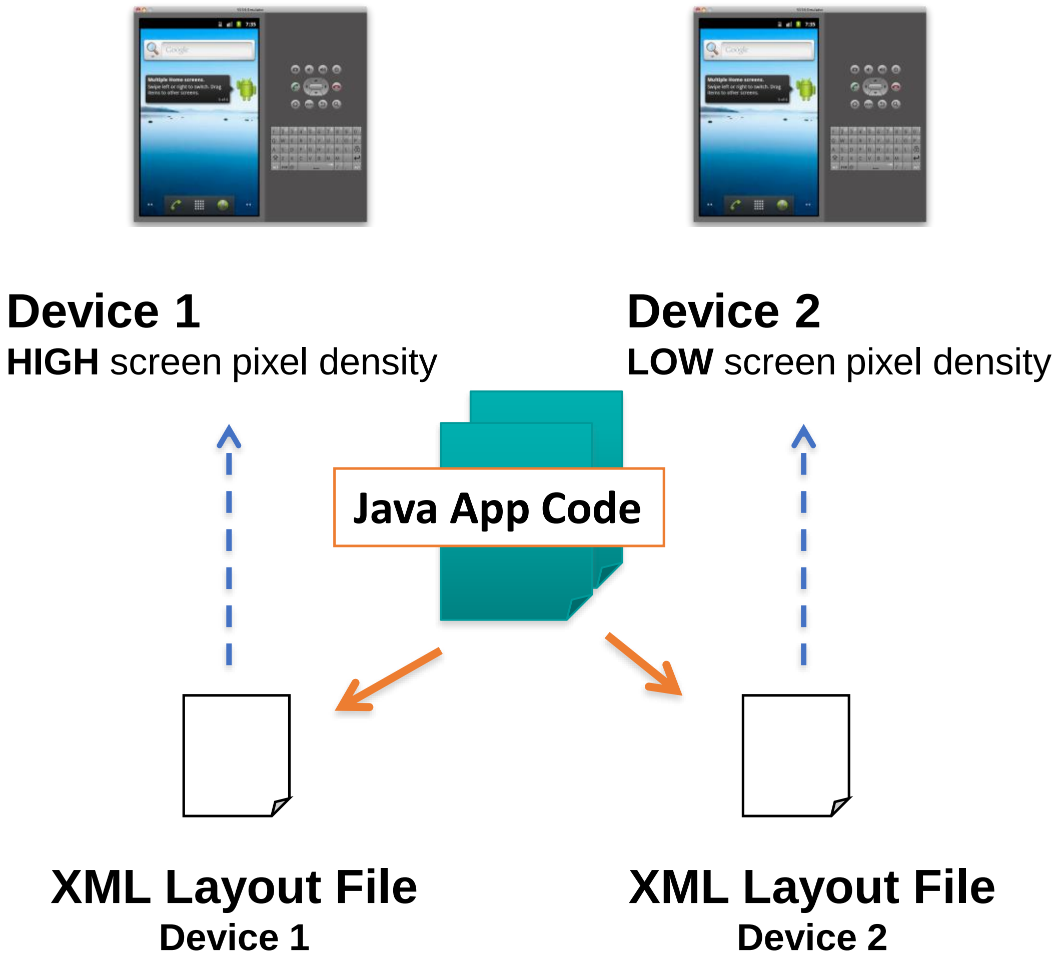
DECLARATIVE APPROACH

Example:

```
< TextView android.text=@string/hello” android:textcolor=@color/blue
android:layout_width=“fill_parent” android:layout_height=“wrap_content” />
< Button android.id=“@+id/Button01” android:textcolor=“@color/blue”
android:layout_width=“fill_parent” android:layout_height=“wrap_content” />
```

Android Components: Activities

EXAMPLE



Device 1
HIGH screen pixel density

Device 2
LOW screen pixel density

Java App Code

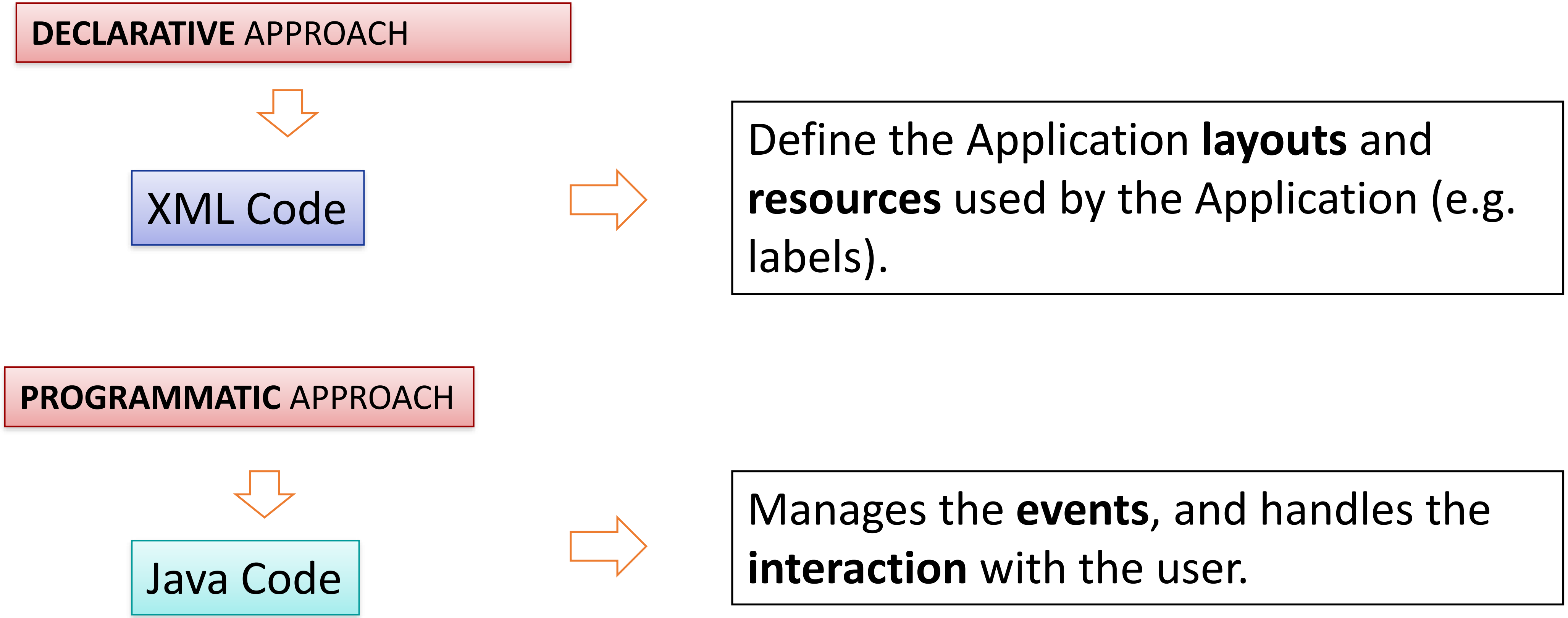
XML Layout File
Device 1

XML Layout File
Device 2

- Build the **application layout** through XML files (like HTML)
- Define **two** different XML **layouts** for two different devices
- At **runtime**, Android detects the current device configuration and loads the appropriate resources for the application
- **No need to recompile**!
- Just add a new XML file if you need to support a new device

Android Components: Activities

- *Android applications typically use both the approaches*!



DECLARATIVE APPROACH

XML Code

Define the Application **layouts** and **resources** used by the Application (e.g. labels).

PROGRAMMATIC APPROACH

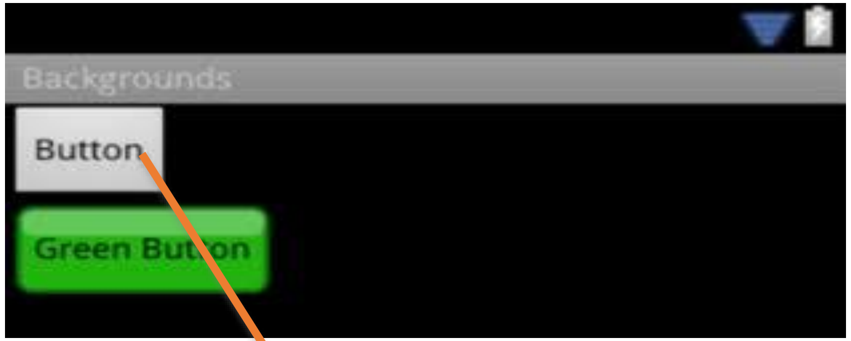
Java Code

Manages the **events**, and handles the **interaction** with the user.

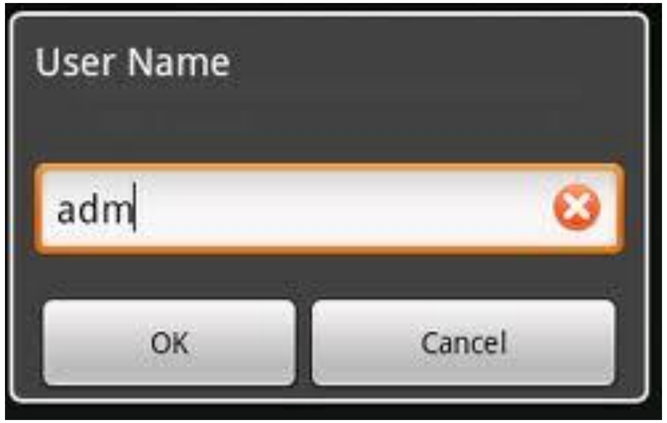
Android Components: Activities

- **Views** can generate **events** (caused by human interactions) that must be managed by the Android-developer.

Button



TextEdit

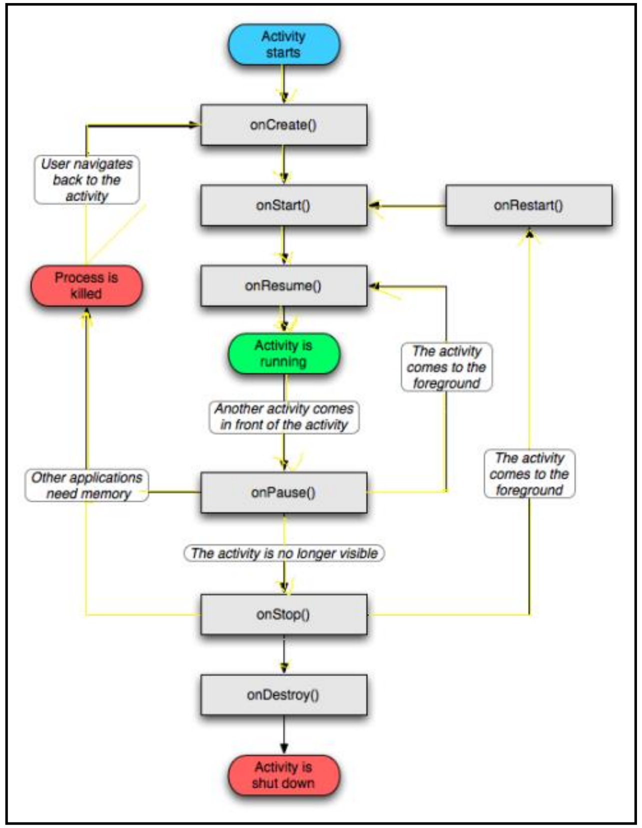

User Name

adm

ESEMPIO

```
public void onClick(View arg0) {
        if (arg0 == Button) {
                // Manage Button events
        }
}
```

Android Components: Activities


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
Other applications need memory
onPause()
The activity comes to the foreground
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

- The **Activity Manager** is responsible for creating, destroying, managing activities.

- Activities can be on different **states**: *starting*, *running*, *stopped*, *destroyed*, *paused*.

- Only one activity can be on the **running** state at a time.

- Activities are organized on a **stack**, and have an event-driven life cycle (details later …)

Android Components: Activities

- Main difference between Android-programming and Java (Oracle) -programming:

  - **Mobile devices have constrained resource capabilities!**

- Activity lifetime depends on **users' choice** (i.e. change of visibility) as well as on **system contraints** (i.e. memory shortage).

- Developer must implement **lifecycle methods** to account for state changes of each Activity …

Android Components: Activities

```
public class MyApp extends Activity {

     public void onCreate() { ... }
     public void onPause()  { ... }
     public void onStop()   { ... }
     public void onDestroy(){ ... }
     ….
}
```

Called when the Activity is **created** the first time.

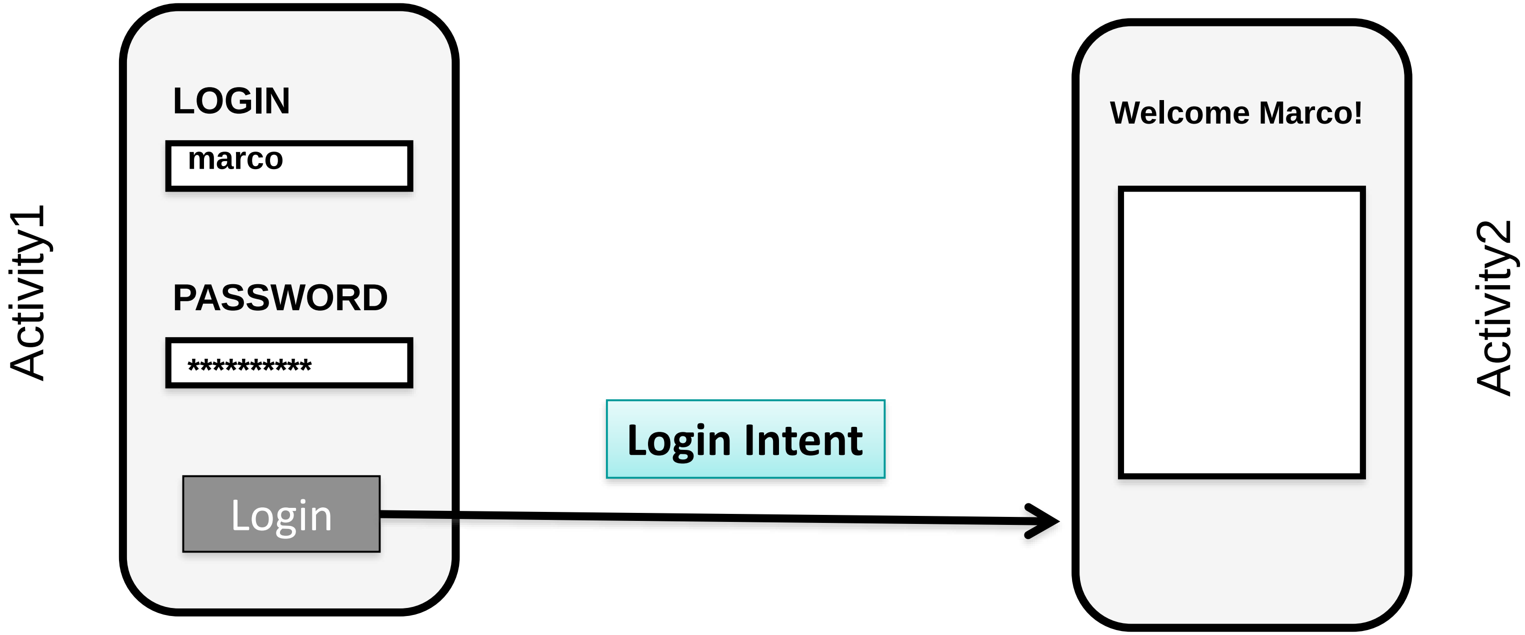
Called when the Activity is **partially visible**.

Called when the Activity is **no longer visible**.

Called when the Activity is **dismissed**.

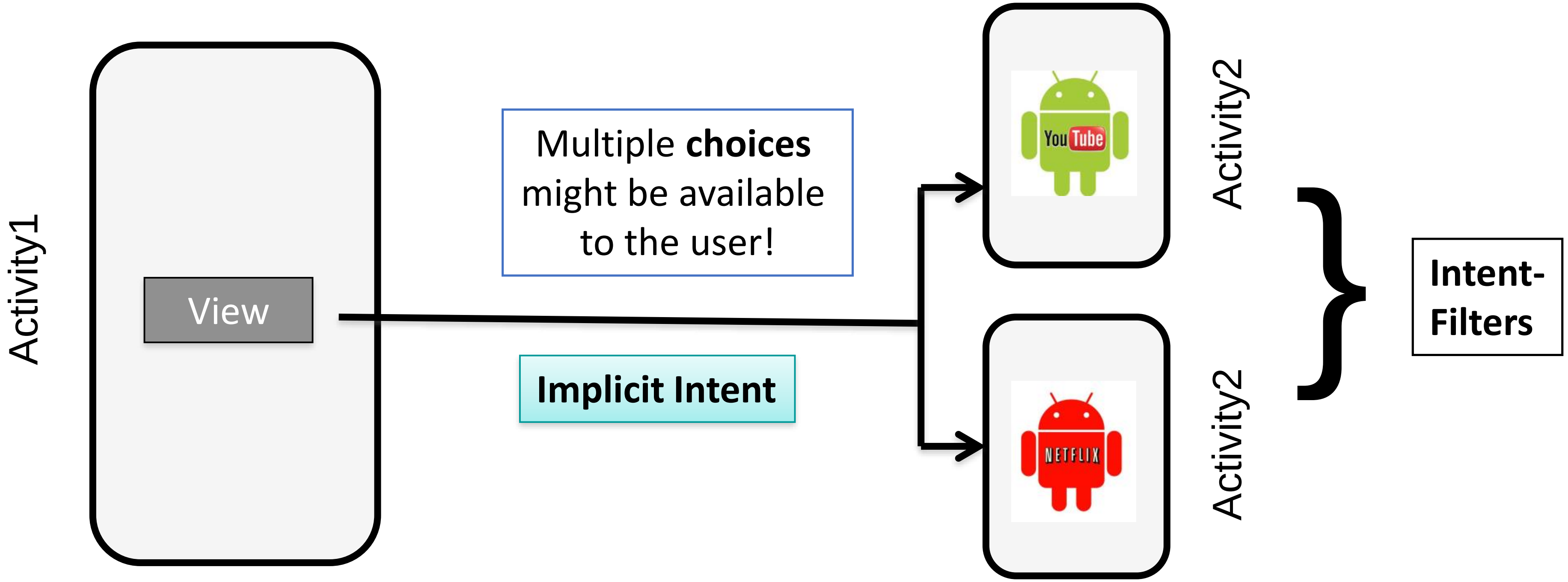
Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Explicit** Intent → The component *(e.g. Activity1)* specifies the destination of the intent *(e.g. Activity 2)*.



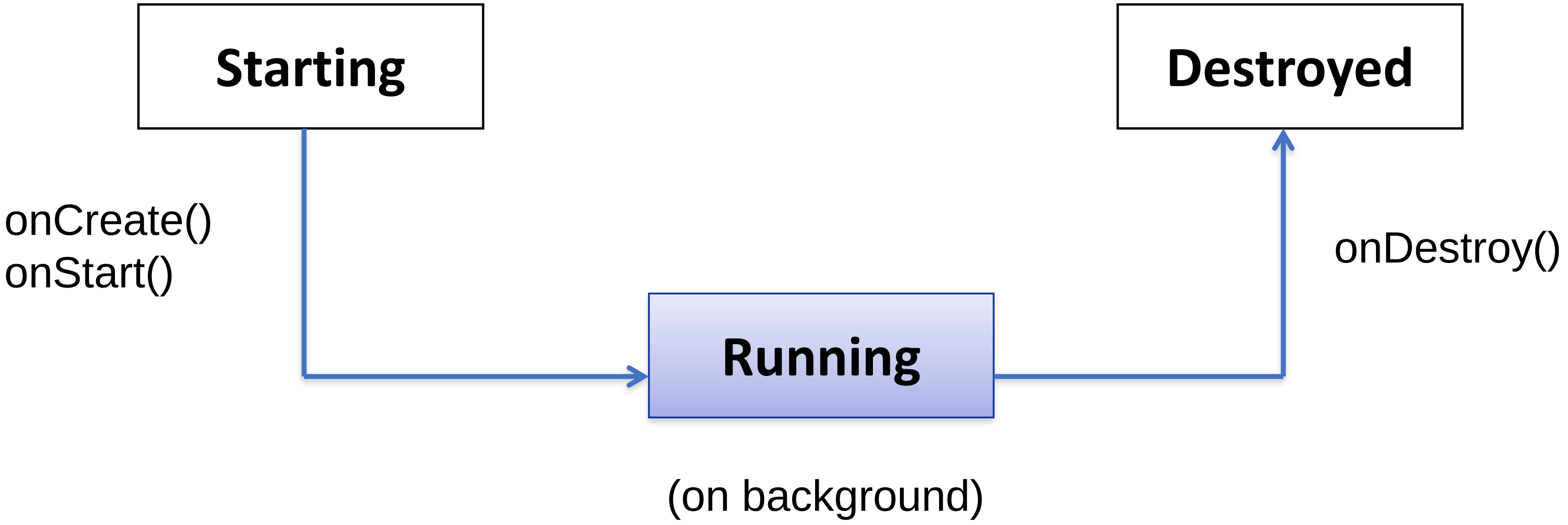
Android Components: Intents

- **Intents**: asynchronous **messages** to activate core Android components (e.g. Activities).
- **Implicit** Intent → The component *(e.g. Activity1)* specifies the type of the intent *(e.g. "View a video")*.



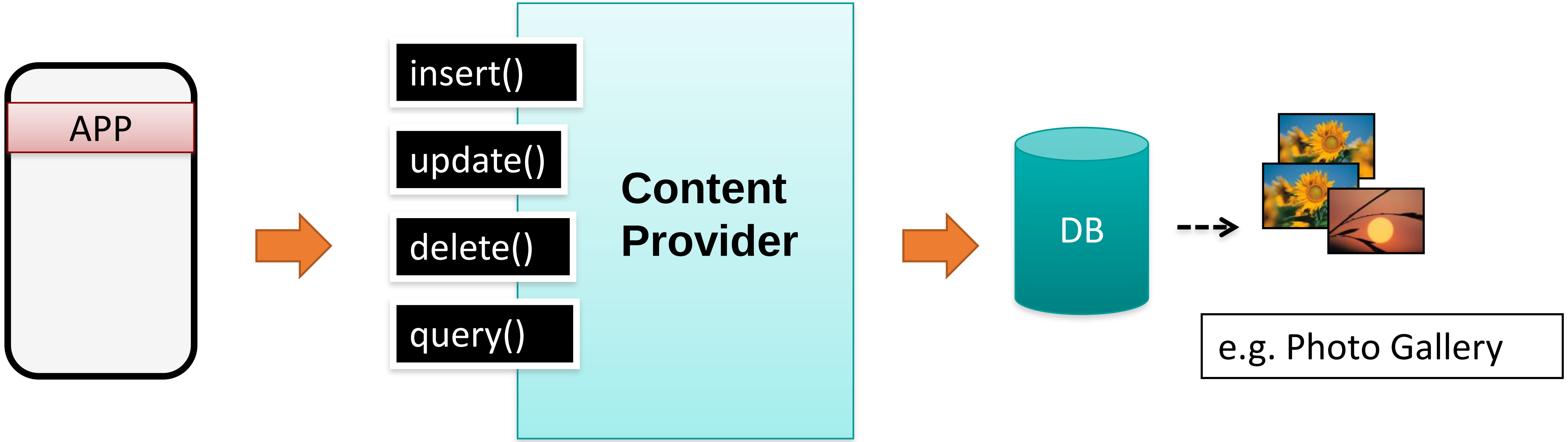
Android Components: Services

- **Services**: like Activities, but run in **background** and do not provide an user interface.
- Used for **non-interactive** tasks (e.g. networking).
- Service life-time composed of 3 states:

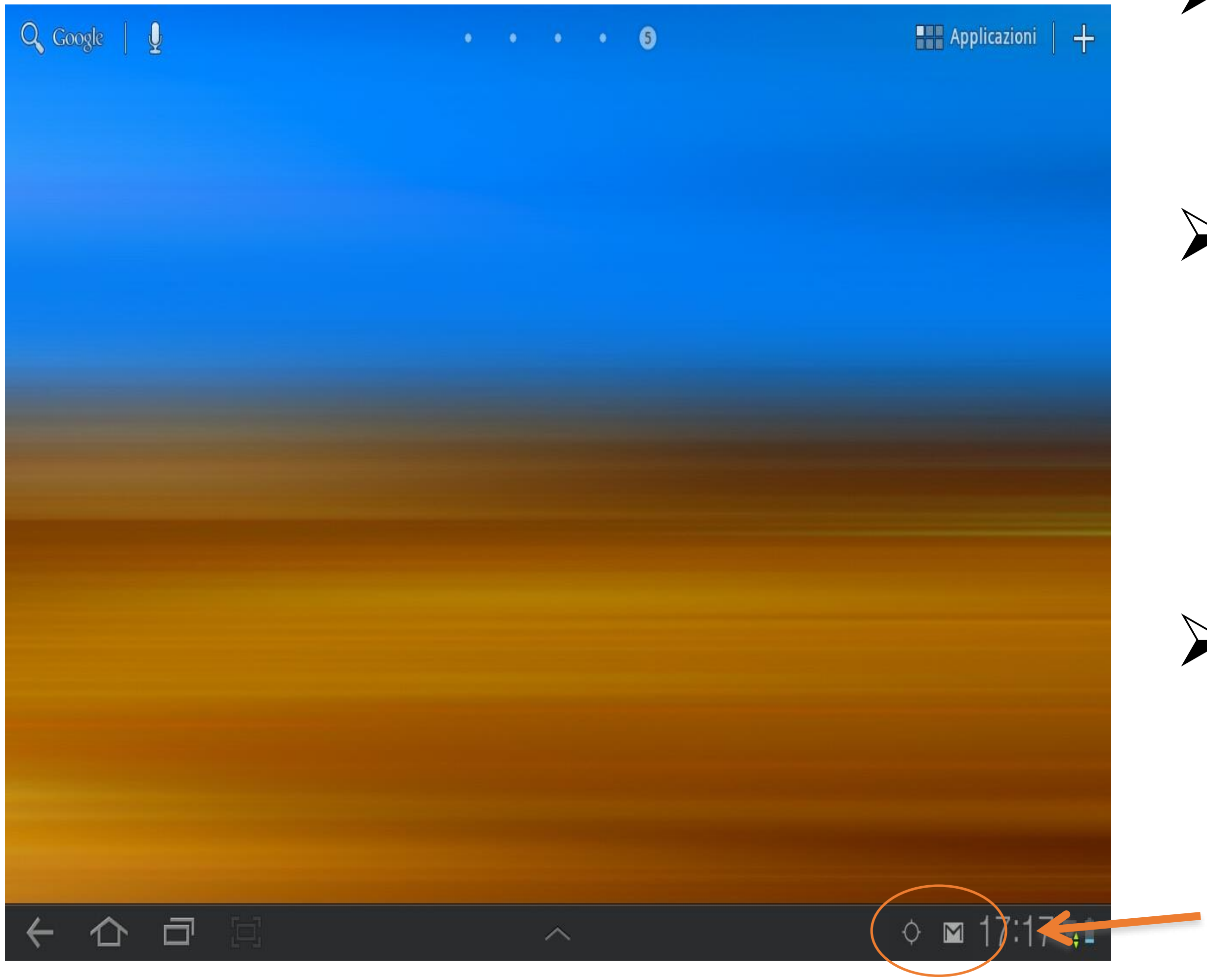


Android Components: Content Providers

- Each Android **application** has its own **private** set of data (managed through *files* or through *SQLite* database).
- **Content Providers**: Standard **interface** to *access and share data among different applications*.



Android Components: Broadcast Receivers



- *Publish/Subscribe* paradigm

- **Broadcast Receivers**: An application can be signaled of **external events**.

- **Notification** types: Call incoming, SMS delivery, Wifi network detected, etc

Android Components: Broadcast Receivers

BROADCAST RECEIVER example

```
class WifiReceiver extends BroadcastReceiver {
       public void onReceive(Context c, Intent intent) {
           String s = new StringBuilder();
           wifiList = mainWifi.getScanResults();
           for(int i = 0; i < wifiList.size(); i++){
               s.append(new Integer(i+1).toString() + ".");
               s.append((wifiList.get(i)).toString());
               s.append("\\n");
           }
           mainText.setText(sb);
       }
   }
```

Android Components: Broadcast Receivers

BROADCAST RECEIVER example

```
public class WifiTester extends Activity {
        WifiManager mainWifi;
        WifiReceiver receiverWifi;
        List<ScanResult> wifiList;
         public void onCreate(Bundle savedInstanceState) {
           …
           mainWifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
           receiverWifi = new WifiReceiver();
           registerReceiver(receiverWifi, new
IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION));
           mainWifi.startScan();
}
```

Android Components: System API

- Using the **components** described so far, Android applications can then leverage the system API …

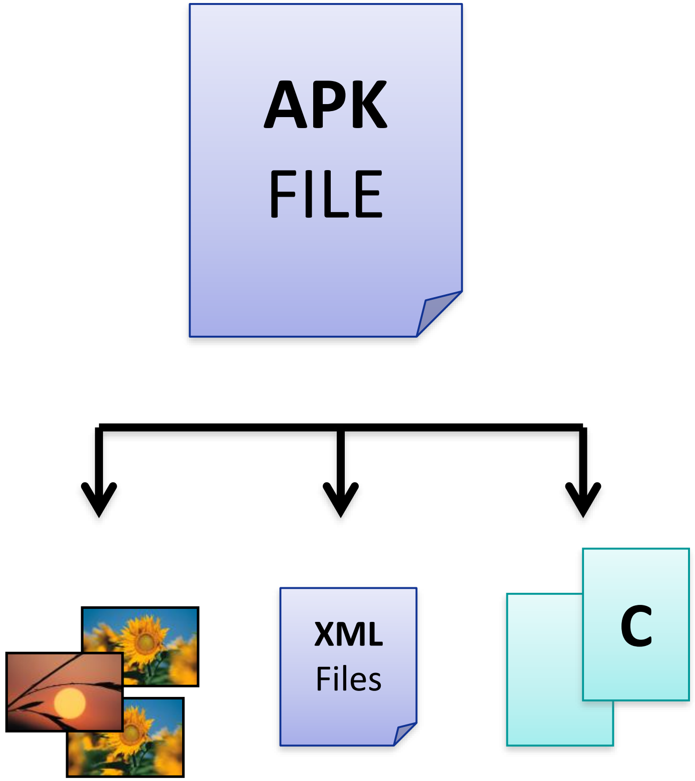
SOME EXAMPLEs …

- *Telephony Manager* data access (call, SMS, etc)
- *Sensor* management (GPS, accelerometer, etc)
- Network *connectivity* (Wifi, bluetooth, NFC, etc)
- *Web* surfing (HTTP client, WebView, etc)
- *Storage* management (files, SQLite db, etc)
- ….

Android Components: Google API

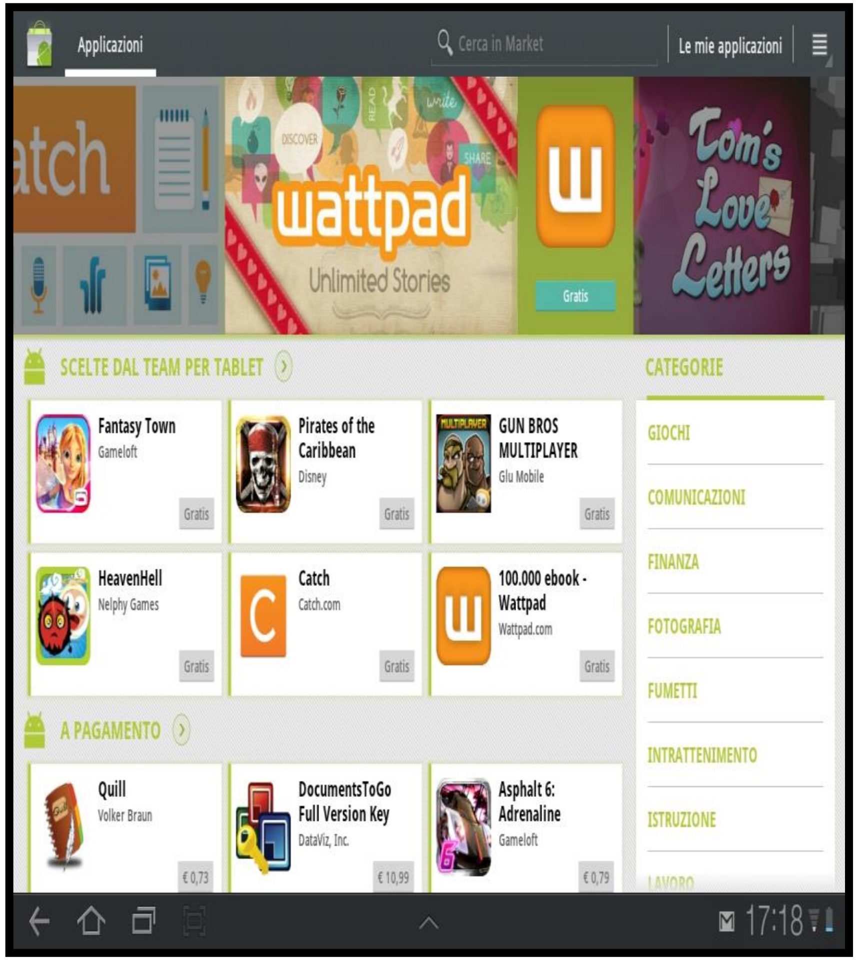
- … or easily interface with other **Google services**:

Distribution



- Each Android **application** is contained on a single **APK** file.
  - Java **Byte-code** (*compiled for Dalvik JVM*)
  - **Resources** (e.g. images. videos, XML layout files)
  - **Libraries** (optimal native C/C++ code)

Android Application Distribution



- ➢Each application must be signed through a **key** before being distributed.

- ➢Applications can be **distributed** via *Web* or via *Stores*.

- ➢**Android Play Store:** application store run by Google … but several other application stores are available (they are just normal applications).

Android Application Security

- ➢Android applications run with a distinct system identity (Linux user ID and group ID), in an **isolated** way.
- ➢Applications must explicitly share resources and data. They do this by declaring the ***permissions*** they need for additional capabilities.
    - ➢Applications statically **declare** the permissions they require.
    - ➢User must **give his/her consensus** during the installation.

ANDROIDMANIFEST.XML

```
<uses-permission android:name=“android.permission.IACCESS_FINE_LOCATION" />

<uses-permission android:name=“android.permission.INTERNET" />
```

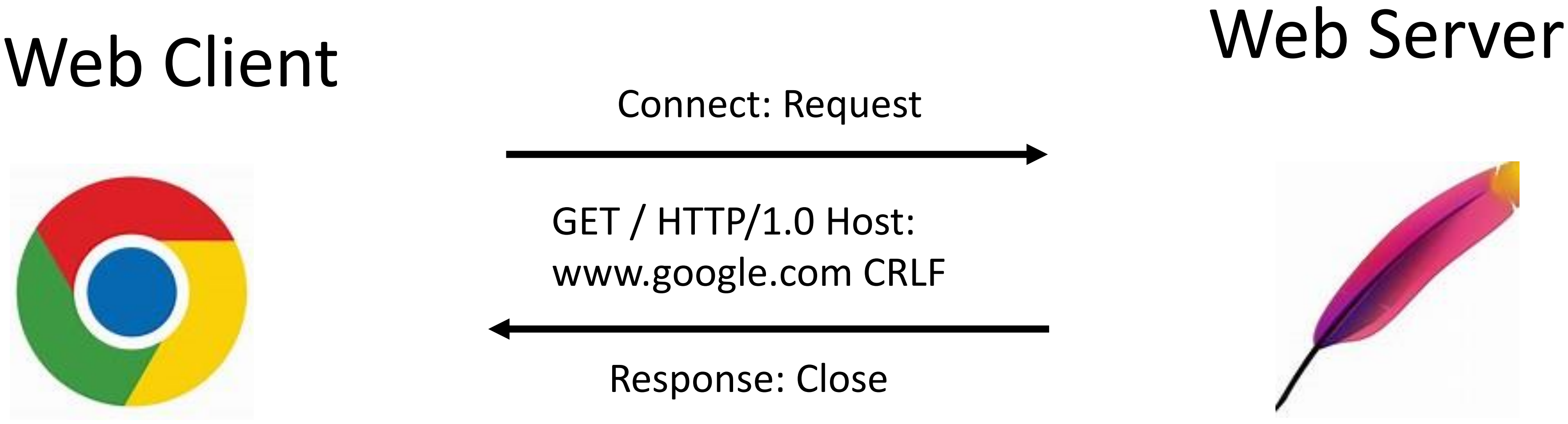
Building a Simple Web Proxy

For the ease of your own life

A Brief History of HTTP

- Mar 1989 - "Information Management: A Proposal"
- Oct 1990 - "WorldWideWeb" coined
- Oct 1994 - W3C founded
- May 1996 - RFC 1945 (HTTP 1.0)
- June 1999 - RFC 2616 (HTTP 1.1)

Anatomy of HTTP 1.0



Web Client

Web Server

Connect: Request

GET / HTTP/1.0 Host: www.google.com CRLF

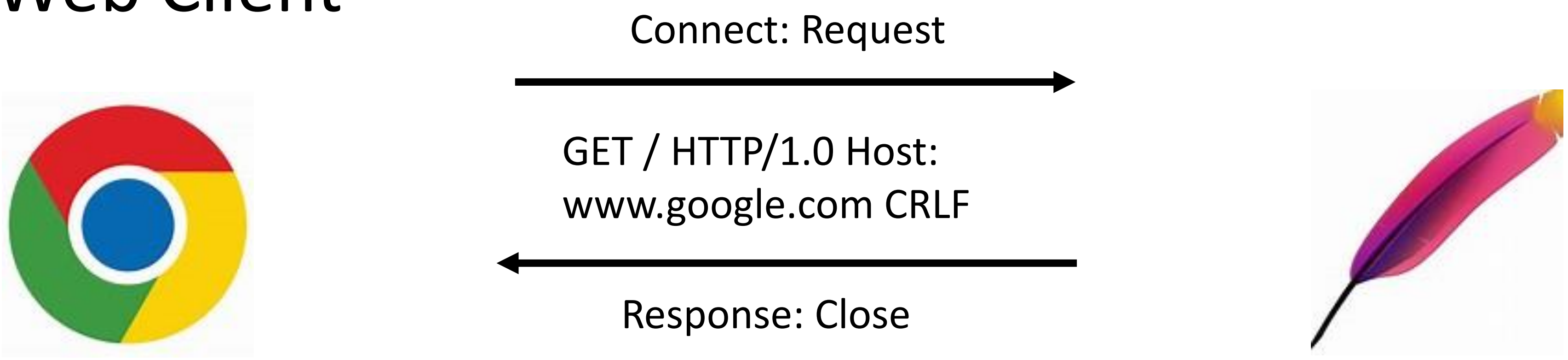
Response: Close

HTTP/1.0 200 OK Date: Sun, 27 May 2018 19:21:24 GMT Content-Type: text/html; CRLF GOOGLE

Anatomy of HTTP 1.0

Web Client

Web Server



Connect: Request

GET / HTTP/1.0 Host: www.google.com CRLF

Response: Close

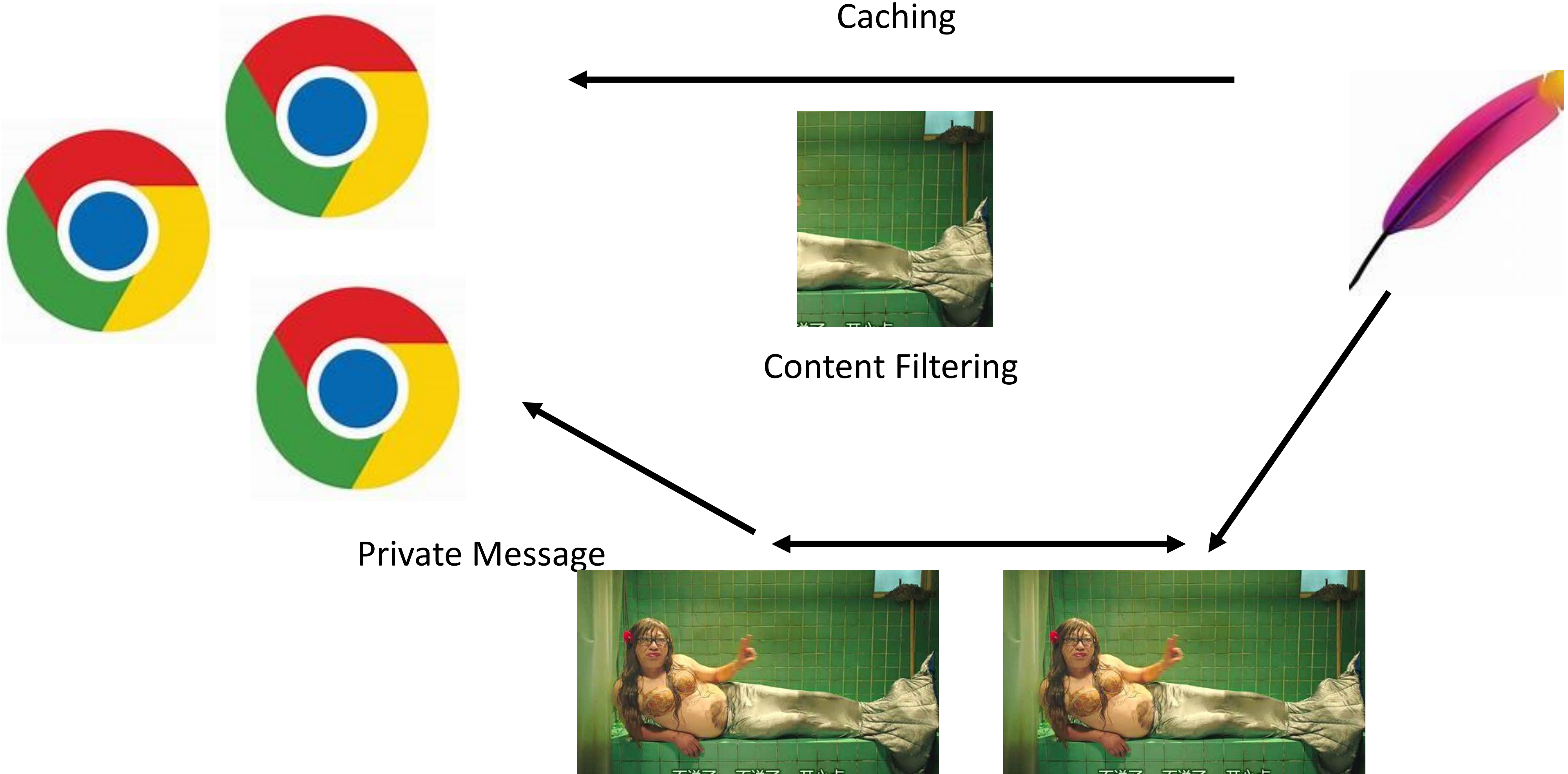
HTTP/1.0 200 OK Date: Sun, 27 May 2018 10:21:24 GMT Content-Type: text/html; CRLF GOOGLE

- Response Status: HTTP/1.0 200 OK
- Response Header: Date: Sun, 27 May 2018 10:21:24 GMT Content-Type: text/html;
- Response Delimiter: CRLF
- Response Body: <html><head> <title>Google</title>

HTTP 1.1 vs 1.0

- Additional Methods (PUT, DELETE, TRACE, CONNECT + GET, HEAD, POST)
- Additional Headers
- Transfer Coding (chunk encoding)
- Persistent Connections (content-length matters)
- Request Pipelining

Why Use a Proxy?



Caching

Content Filtering

Private Message

Building a Simple Web Proxy

- Forward client requests to the remote server and return response to the client
- Handle HTTP 1.0 (GET)
- Single-threaded, non-caching web proxy

- ./proxy

Handling Requests

- What you need from a client request: host, port, and URI path

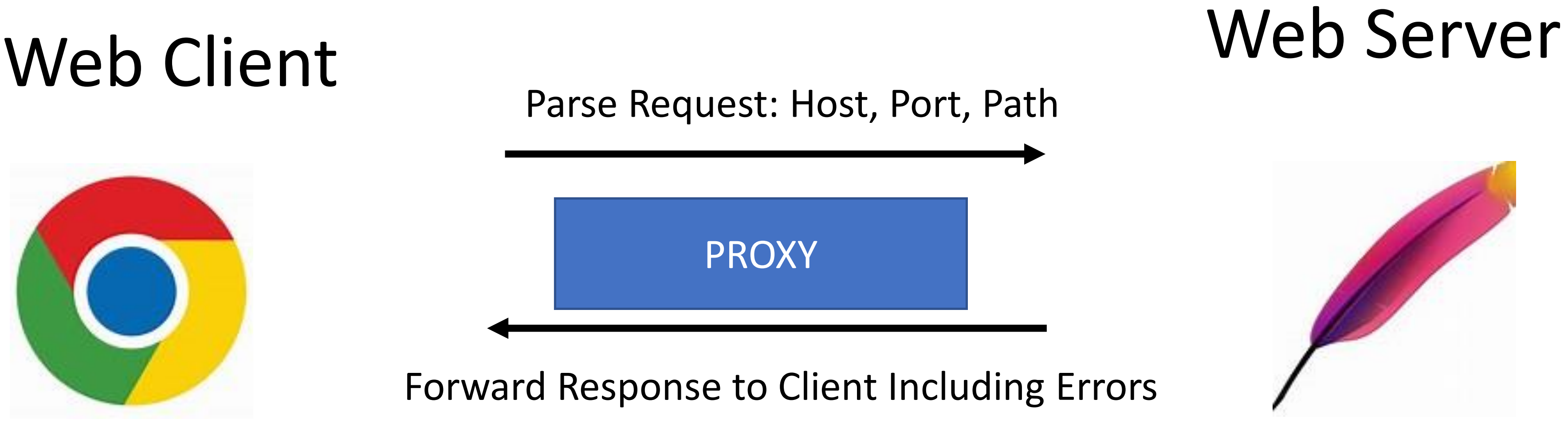
  GET http://www.ncu.edu.cn:80/ HTTP/1.0

- What you send to a remote server:

  GET / HTTP/1.0 Host: www.ncu.edu.cn :80 (Additional headers, if any...)

- Check request line and header format

Handling Responses



Web Client

Parse Request: Host, Port, Path

PROXY

Forward Response to Client Including Errors

Web Server

Handling Errors

- Method != GET: Not Implemented (501)
- Unparseable request: Bad Request (400)
- Keep parsing simple: no need for regex
- Postel’s law: Be liberal in what you accept, and conservative in what you send convert HTTP 1.1 request

  to HTTP 1.0 convert \r to \r\n etc...

Testing Your Proxy

- Telnet to your proxy and issue a request

```
./proxy 5000 > telnet localhost 5000
Trying 127.0.0.1...
Connected to localhost.localdomain (127.0.0.1).
Escape character is '^]’.
GET http://www.google.com/ HTTP/1.0
```

- Direct your browser to use your proxy
- Use the supplied proxy_tester.py

Proxy Guidance

- Assignment page

- RFC 1945 (HTTP 1.0)

- Google, wikipedia, Bing, man pages

References (1)

1. C. Horstmann, *Big Java Late Objects*, Wiley, 2012. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/–/9781118087886
2. J. Bloch, *Effective Java*, 2nd ed., Addison–Wesley, 2008. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/programming/java/9780137150021
3. S.B. Zakhour, S. Kannan, and R. Gallardo, *The Java® Tutorial: A Short Course on the Basics*, 5th ed., Addison–Wesley, 2013. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/programming/java/9780132761987
4. C. Collins, M. Galpin, and M. Kaeppler, *Android in Practice*, Manning, 2011. Online: http://proquest.safaribooksonline.com.proxy.lib.ohio–state.edu/book/programming/android/9781935182924
5. M.L. Sichitiu, 2011, http://www.ece.ncsu.edu/wireless/MadeInWALAN/AndroidTutorial/PPTs/javaReview.ppt
6. Oracle, http://docs.oracle.com/javase/1.5.0/docs/api/index.html
7. Wikipedia, https://en.wikipedia.org/wiki/Vehicle_Identification_Number
8. Nielsen Co., “Smartphone Milestone: Half of Mobile Subscribers Ages 55+ Own Smartphones”, 22 Apr. 2014, http://www.nielsen.com/us/en/insights/news/2014/smartphone-milestone-half-of-americans-ages-55-own-smartphones.html
9. Android Open Source Project, http://www.android.com

References (2)

10. http://bcs.wiley.com/he-bcs/Books?action=index&itemId=1118087887&bcsId=7006
11. B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes, and D. Lea, Java Concurrency in Practice, Addison-Wesley, 2006, online at http://proquest.safaribooksonline.com/book/programming/java/0321349601
12. https://developer.android.com/guide/components/activities.html
13. https://developer.android.com/guide/topics/ui/declaring-layout.html#CommonLayouts
14. https://cloud.genymotion.com/page/doc/#collapse4
15. http://blog.zeezonline.com/2013/11/install-google-play-on-genymotion-2-0/

Conclusion

- We have started on some unique programming models in Linux
- We have talked a lot on the new programming architecture
- We will have one more class on the real topic
- Reading Assignment: the all chapters in your textbook