Programming with Android: System Architecture

Federico Montori

Dipartimento di Scienze dell'Informazione

Università di Bologna

Outline

- Android Architecture: An **Overview**
- Android Java **Virtual Machine**
- Android Components: **Activities**
- Android Components: **Intents**
- Android Components: **Services**
- Android Components: **Content Providers**
- Android Application **Distribution** and **Markets**

Android … What?

- **Android** is a *Linux-based* platform for *mobile touchscreen devices* …
  - *Operating System*
  - *Middleware*
  - *Applications*
  - *Software Development Kit* (**SDK**)
- Which kind of **mobile devices** … (examples)

SMARTPHONES

TABLETS

EREADERS

ANDROID TV

GOOGLE GLASSES

?

Android ... What?

SMART FRIDGE

ANDROID MICROWAVE

SMARTPHONES

TABLETS

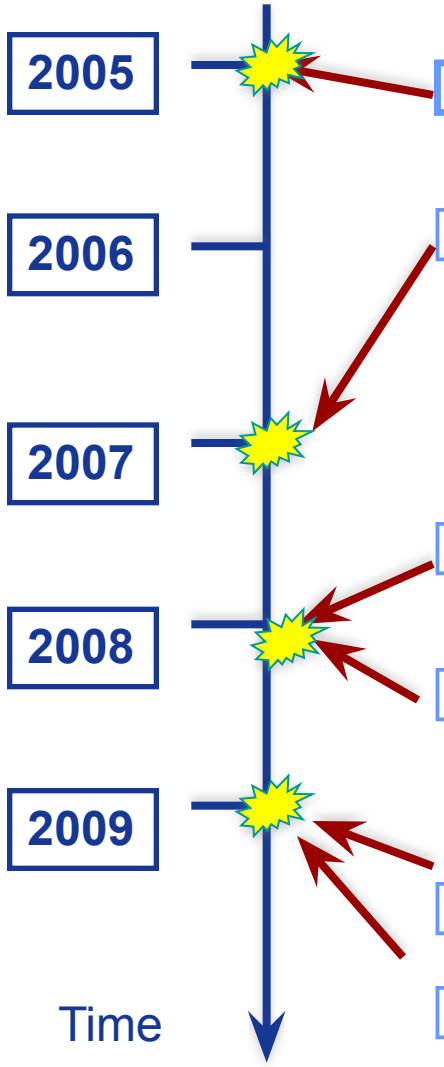
EREADERS

ANDROID TV

GOOGLE GLASSES

?

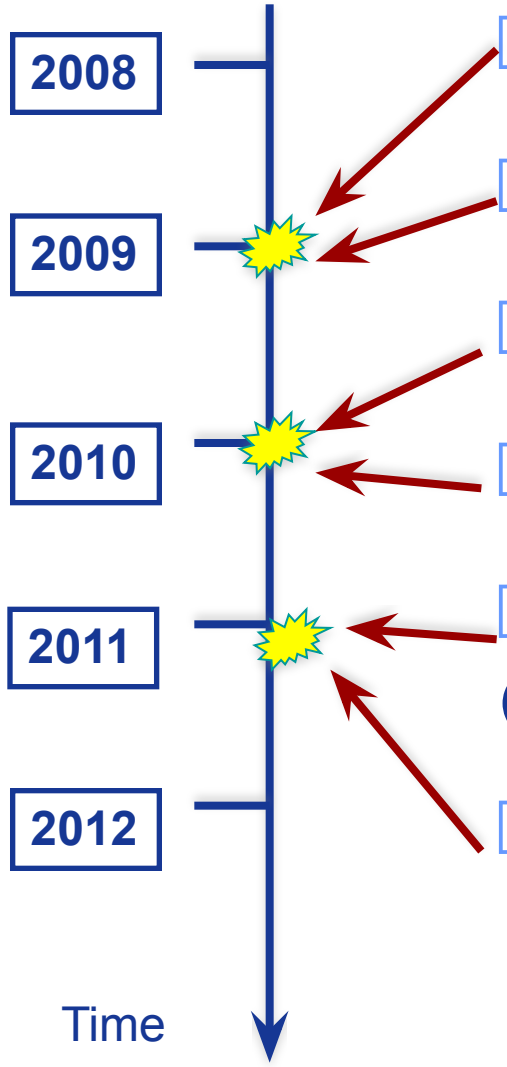
Android … When?



- **Google** buys Android from the **Android Inc.**
- Open Handset Alliance (**OHA**) created for open standards for mobile devices. Partners of OHA: Google, Motorola, Samsung, Vodafone, T-Mobile, etc.
- Android 1.0 Released
- The first Android smartphone: G1 HTC-Dream

  Few Google apps integrated directly with the OS
- Android **1.1** Released
- Android **1.5 (CupCake)** Released
  - External Apps
  - On- screen keyboard

Android … When?



- Android **1.6** (**Donut**) Released
  - Different screen sizes
  - CDMA
- Android **2.0** (**Eclair**) Released
  - speech-to-text
  - pinch-to-zoom

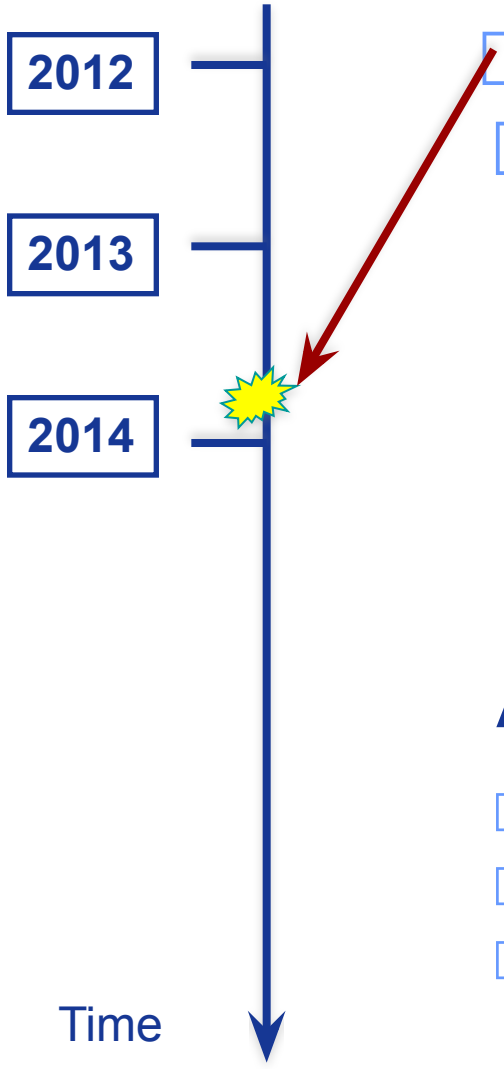
- Android **2.2** (**Froyo**) Released
  - bottom dock
  - voice actions
- Android **2.3** (**Gingerbread**) Released
  - The “design release”
- Android **3.0** (**Honeycomb**) Released

(First version for devices with larger screens such as tablets)

- Android **4.0** (**Ice-Cream Sandwich**) Released. (It merges the 3.x tab centric design and the v2.x phone based design into a single version.)

- swiping for dismissing
- card appearance

Android … When?



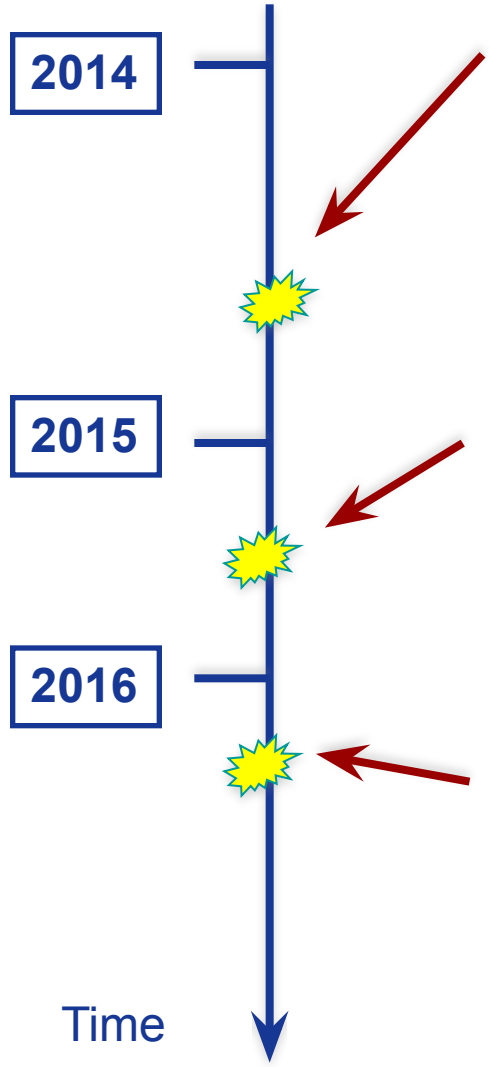
- Android **4.1** (**Jelly Bean**) Released
- Android **4.4** (**Kitkat**) Released
  - Wireless printing capability
  - Ability for applications to use "immersive mode"
  - Performance optimization
  - New experimental runtime virtual machine, ART…

OK GOOGLE!
(but only when the screen is on…)

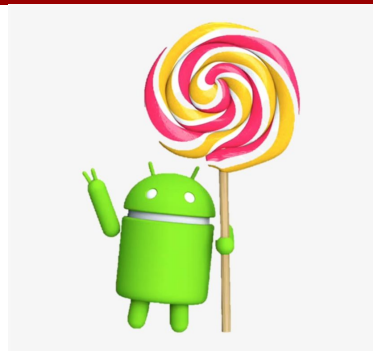
API Level 19 (Android 4.4):

- Support to new embedded sensors (e.g. STEP_DETECTOR)
- Adaptive video playback functionalities
- Read and write SMS and MMS messages (managing default text messaging client)

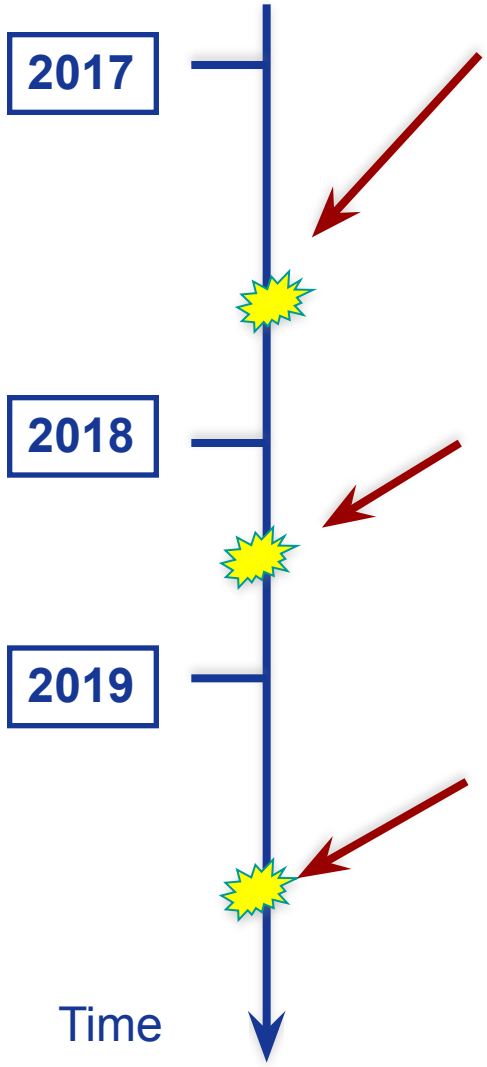
Android … When?



- Android **5.0** (**Lollipop**) Released
  - Material Design!
  - OK Google (the true one)
  - Lots of bugs...
- Android **6.0** (**Marshmallow**) Released
  - Fingerprint
  - USB-C
  - Runtime Permissions!!!
- Android **7.0** (**Nougat**) Released
  - Google Assistant
  - Split screen, data saver...
  - ...Pixel!!!



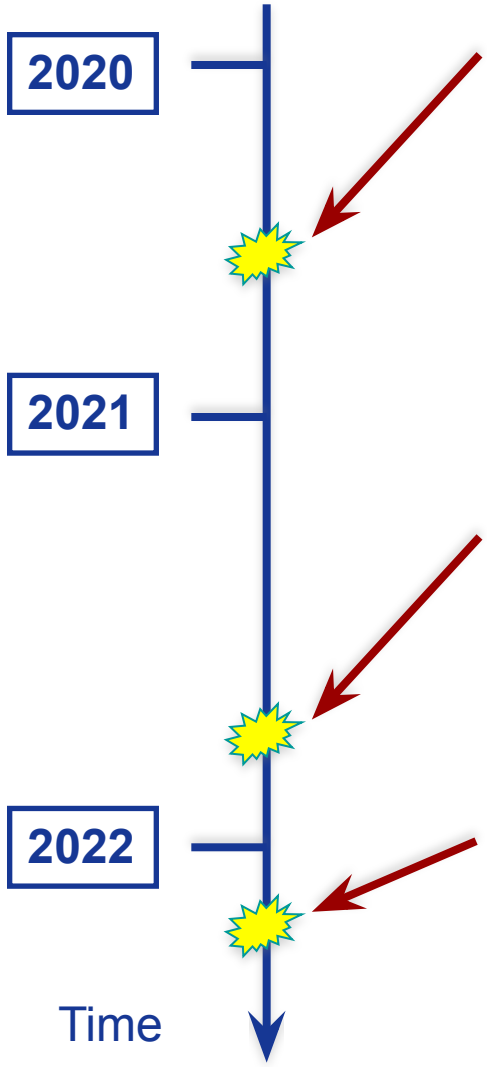
Android … When?



- Android **8.0** (**Oreo**) Released
  - Notification Channels and Snooze
  - picture-in-picture
  - Android Apps on Chromebooks
- Android **9.0** (**Pie**) Released
  - Brightness & Battery Management
  - Hybrid gestures system
  - Privacy & Security
- Android **10.0** (**Q**) Released
  - Expanded permission
  - swipe-driven
  - ...No more sweets!!!

android

Android … When?



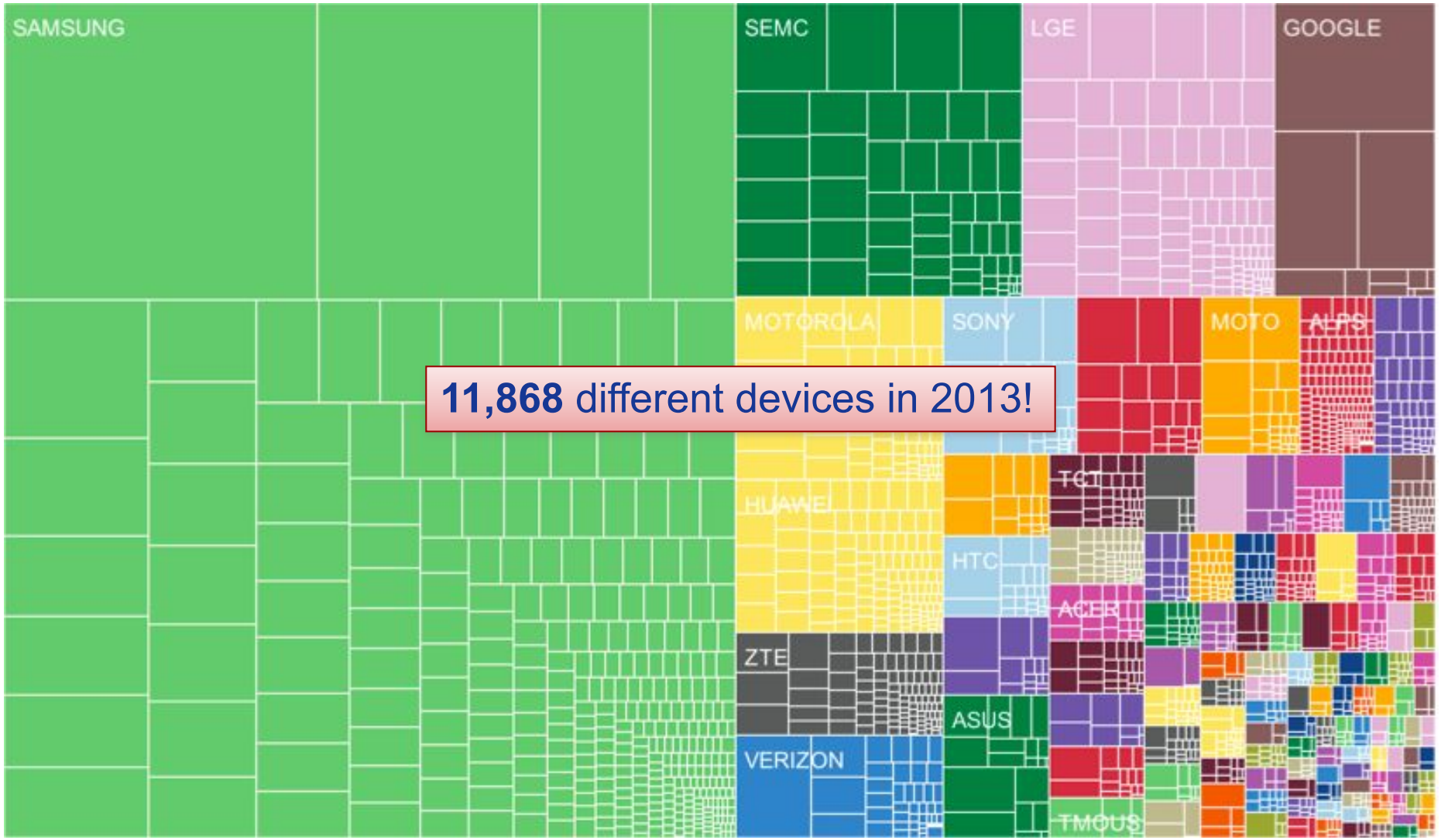
- In February Android **11.0** (**R**) Released
  - One-time permissions for temporary features (location, microphone and camera)
  - Exposure notification and privacy fixes

- In October Android **12.0** (**S**) Released
  - Location can be blurred even if required
  - Mostly optimizations and graphical improvements

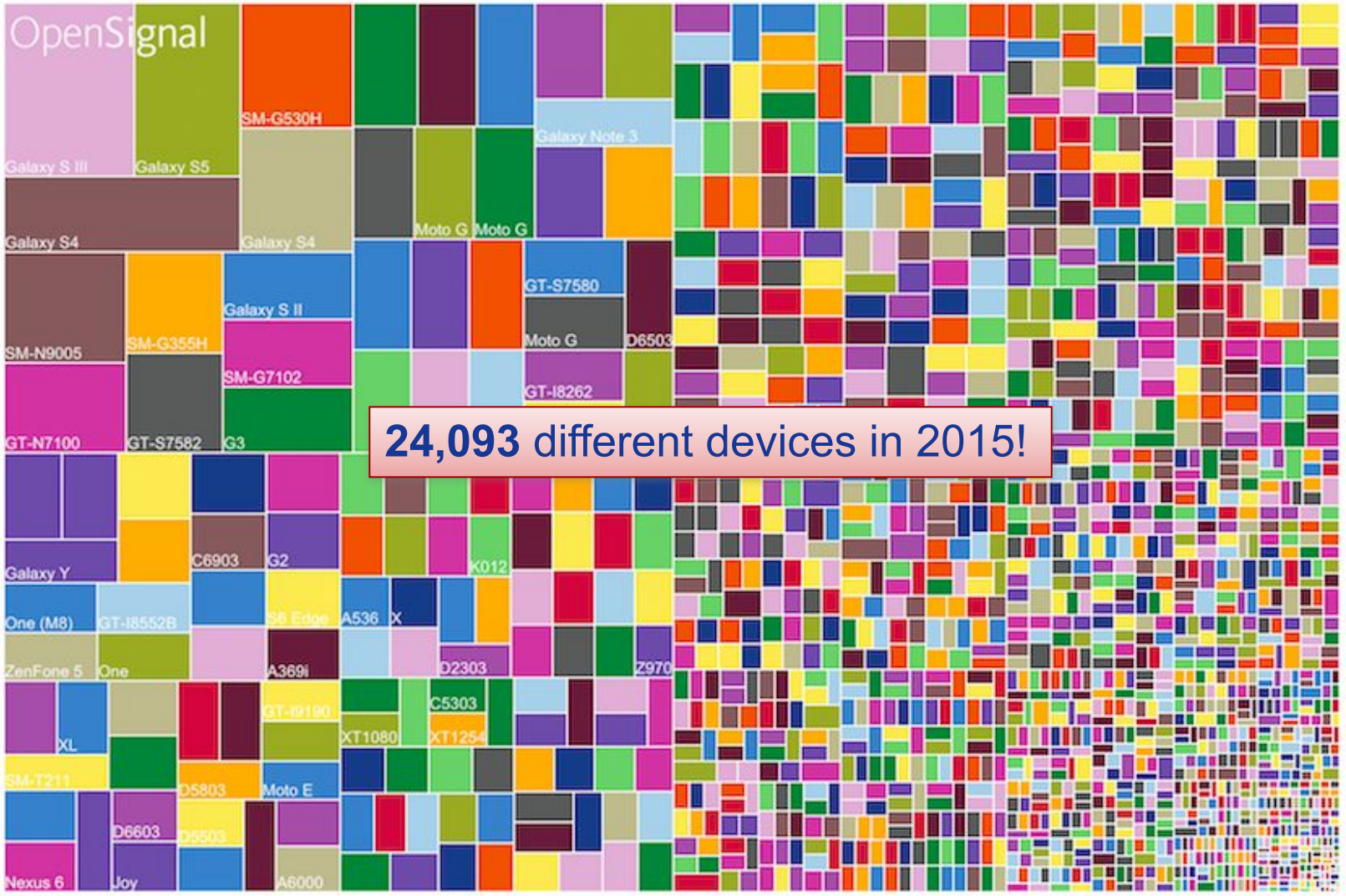
- Android **13.0** (**Tiramisu**) is in Developer preview now

Android … heterogeneity

http://opensignal.com/reports/fragmentation-2013/

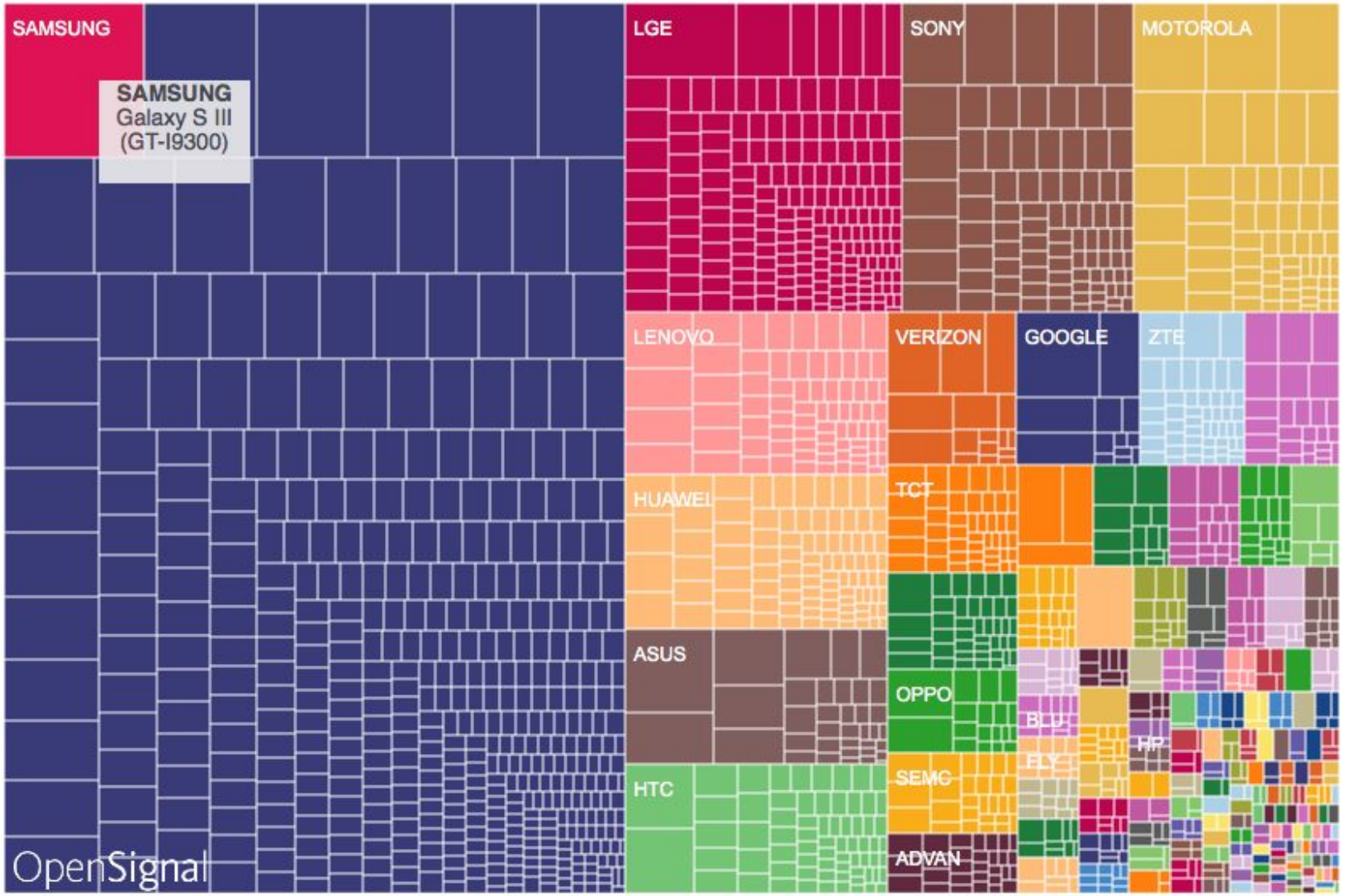


Android … heterogeneity



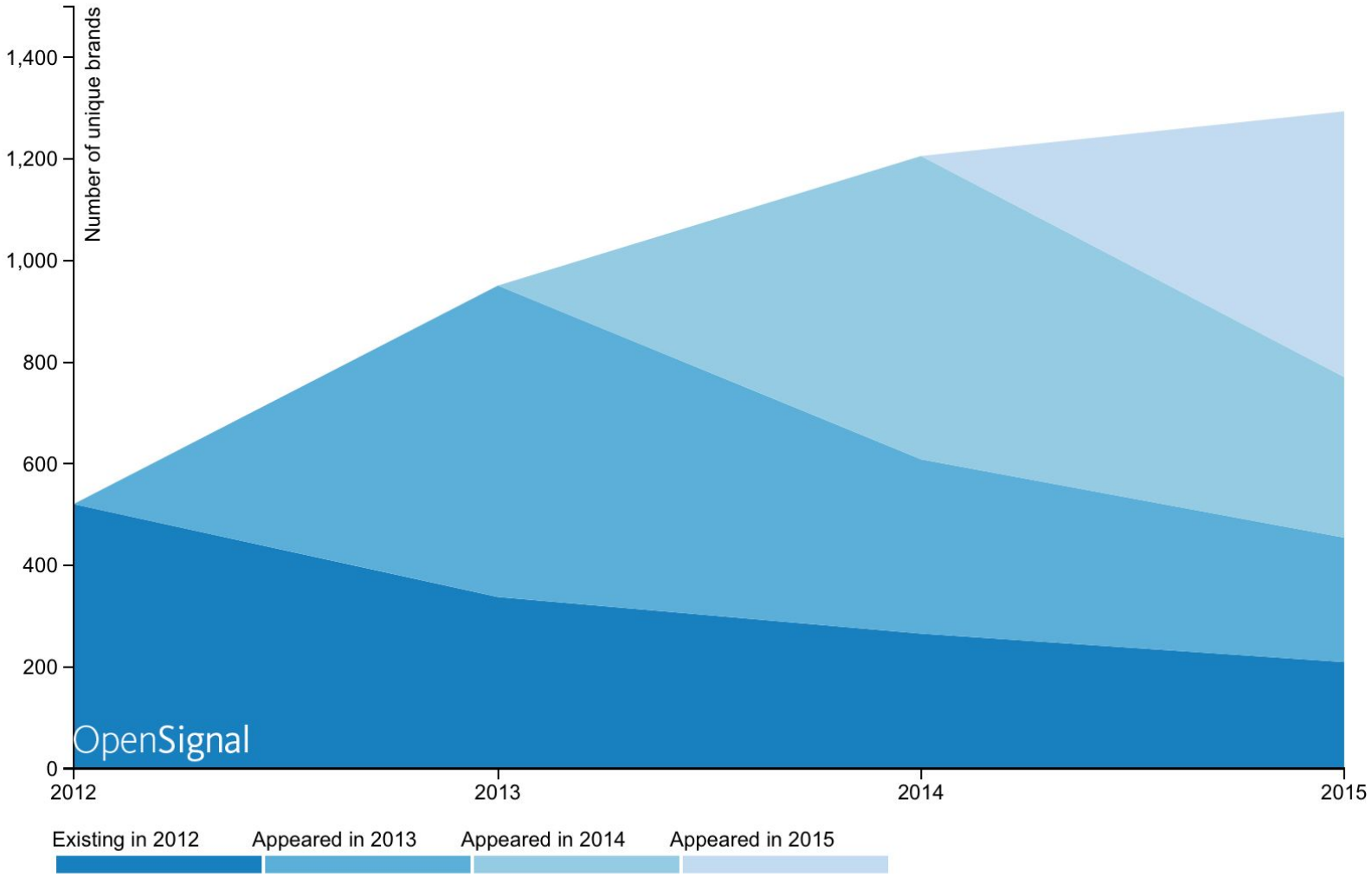
http://opensignal.com/reports/2015/08/android-fragmentation/

Android … heterogeneity

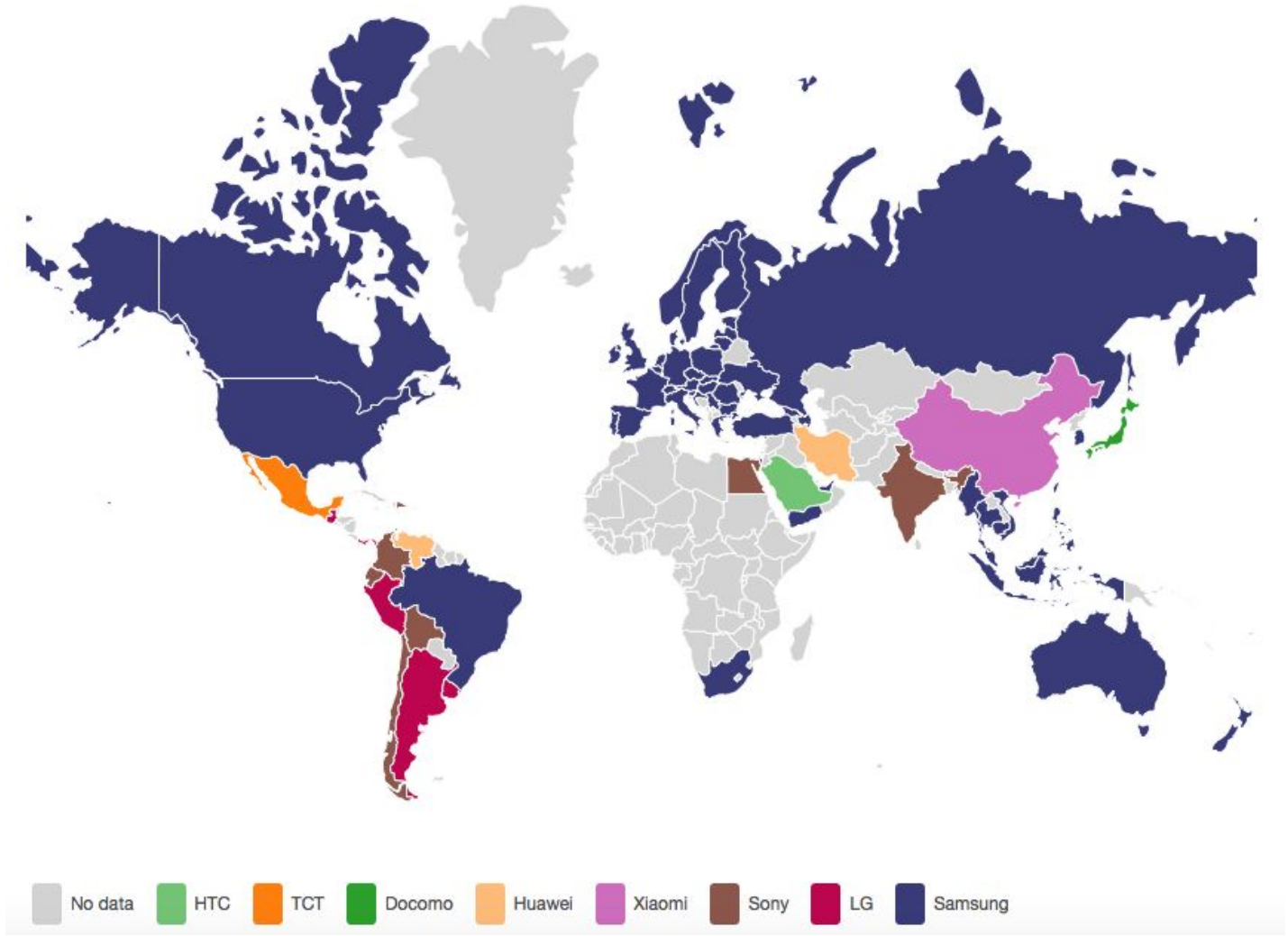


http://opensignal.com/reports/2015/08/android-fragmentation/

Android … heterogeneity

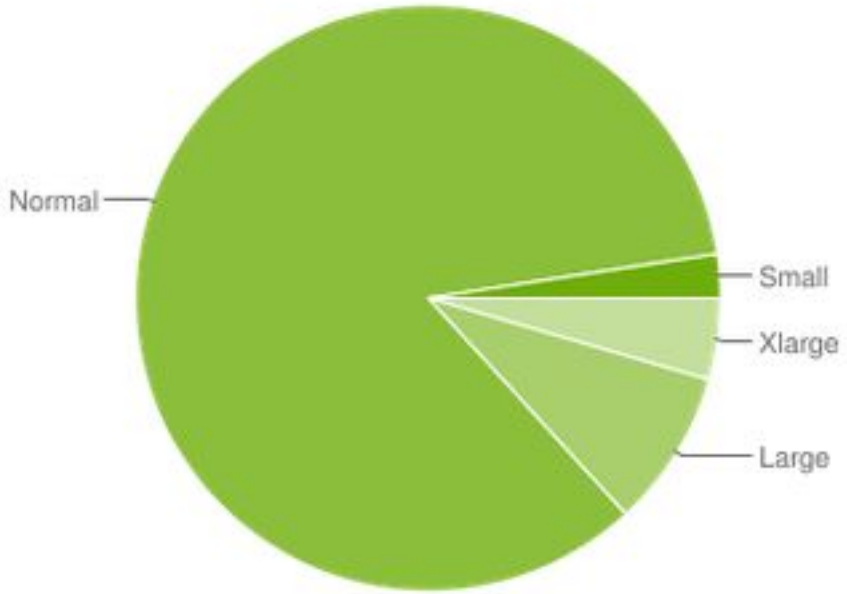


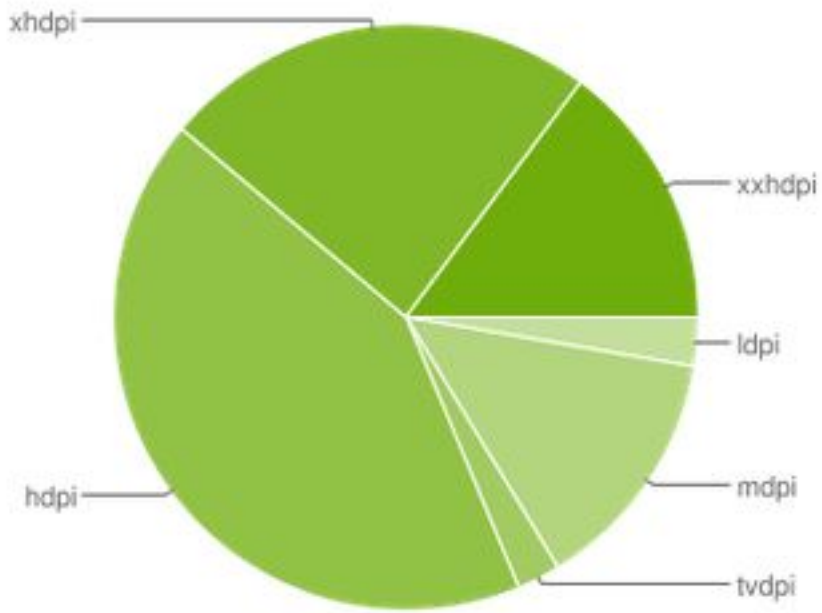
Android … heterogeneity



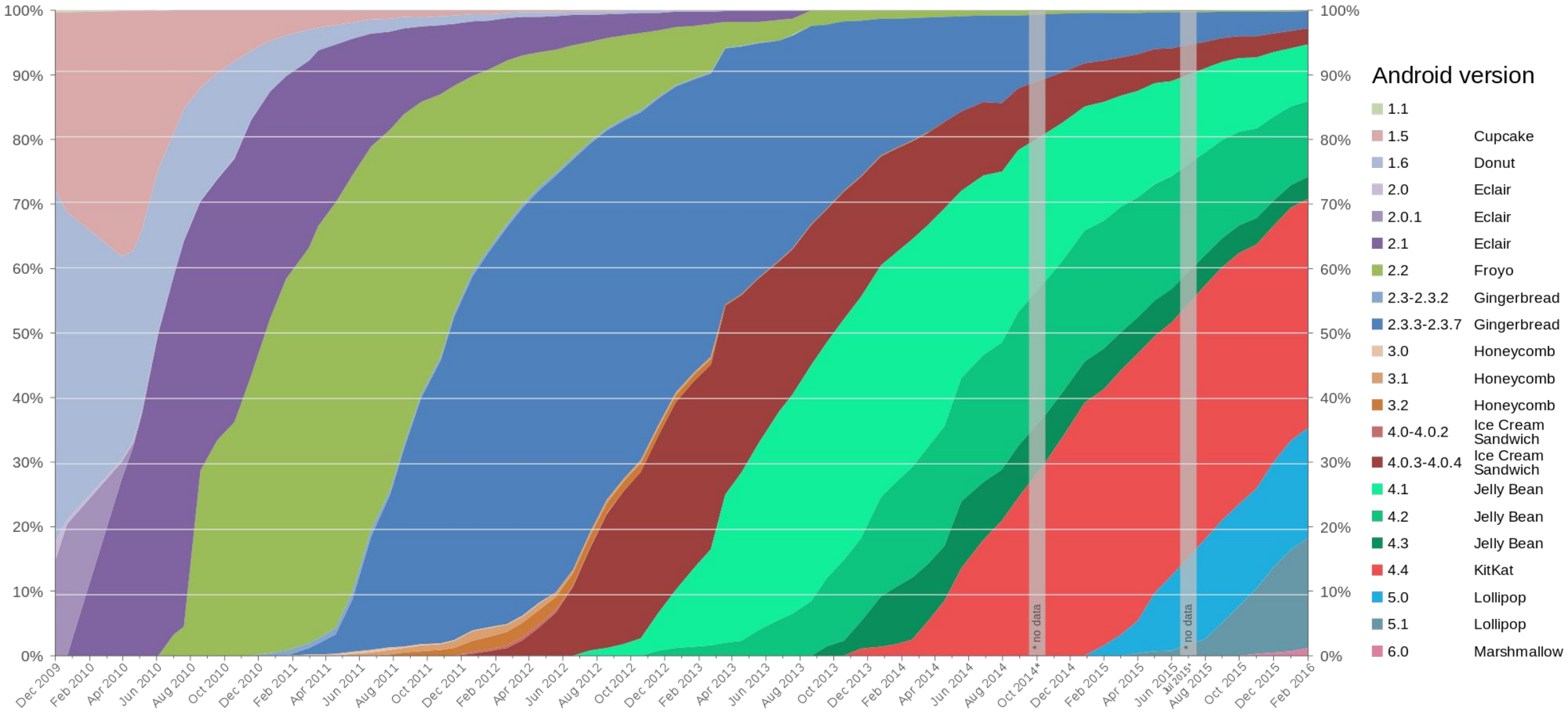
Android … heterogeneity

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	2.4%						2.4%
Normal		5.1%	0.1%	41.5%	22.9%	14.8%	84.4%
Large	0.3%	5.0%	2.3%	0.6%	0.5%		8.7%
Xlarge		3.5%		0.3%	0.7%		4.5%
Total	2.7%	13.6%	2.4%	42.4%	24.1%	14.8%	

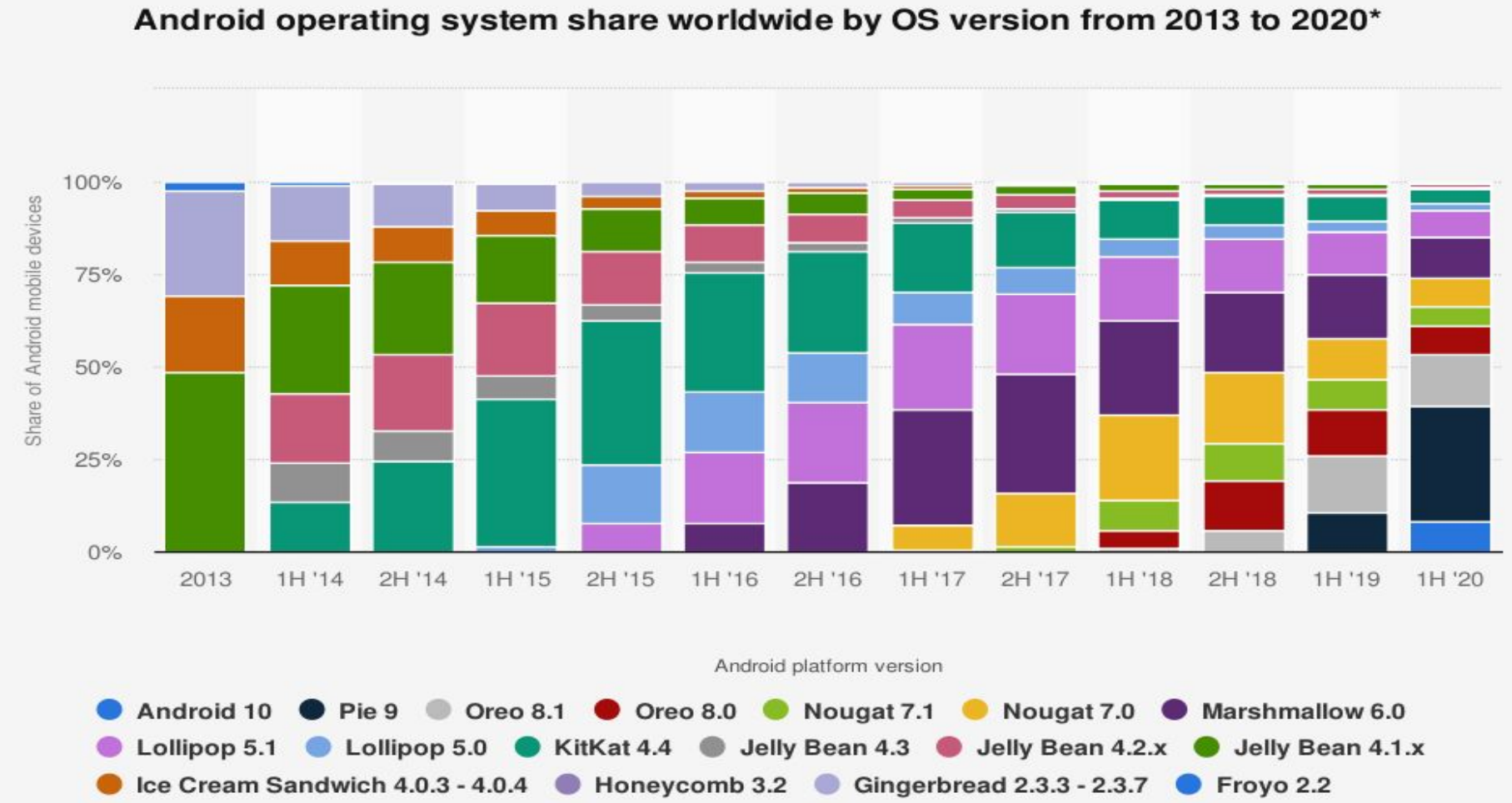




Android … heterogeneity

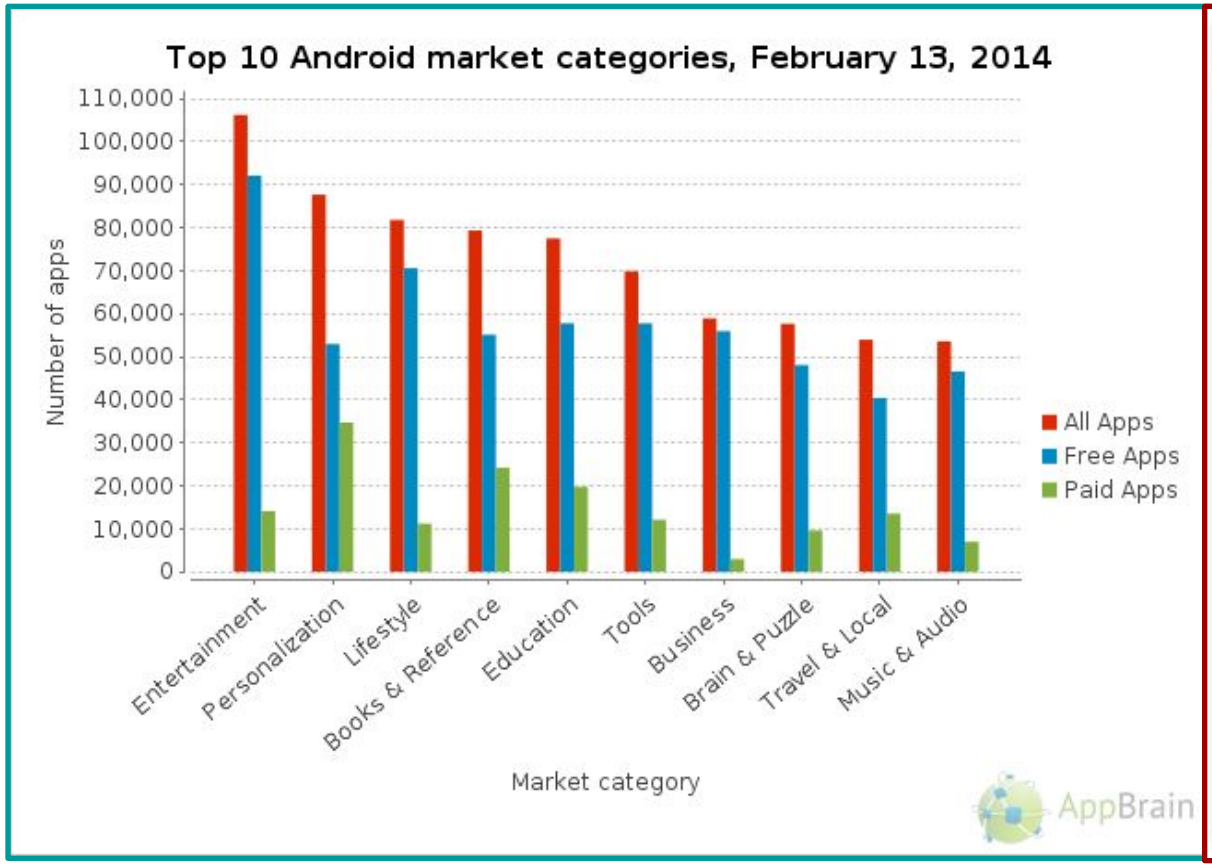


Android … heterogeneity



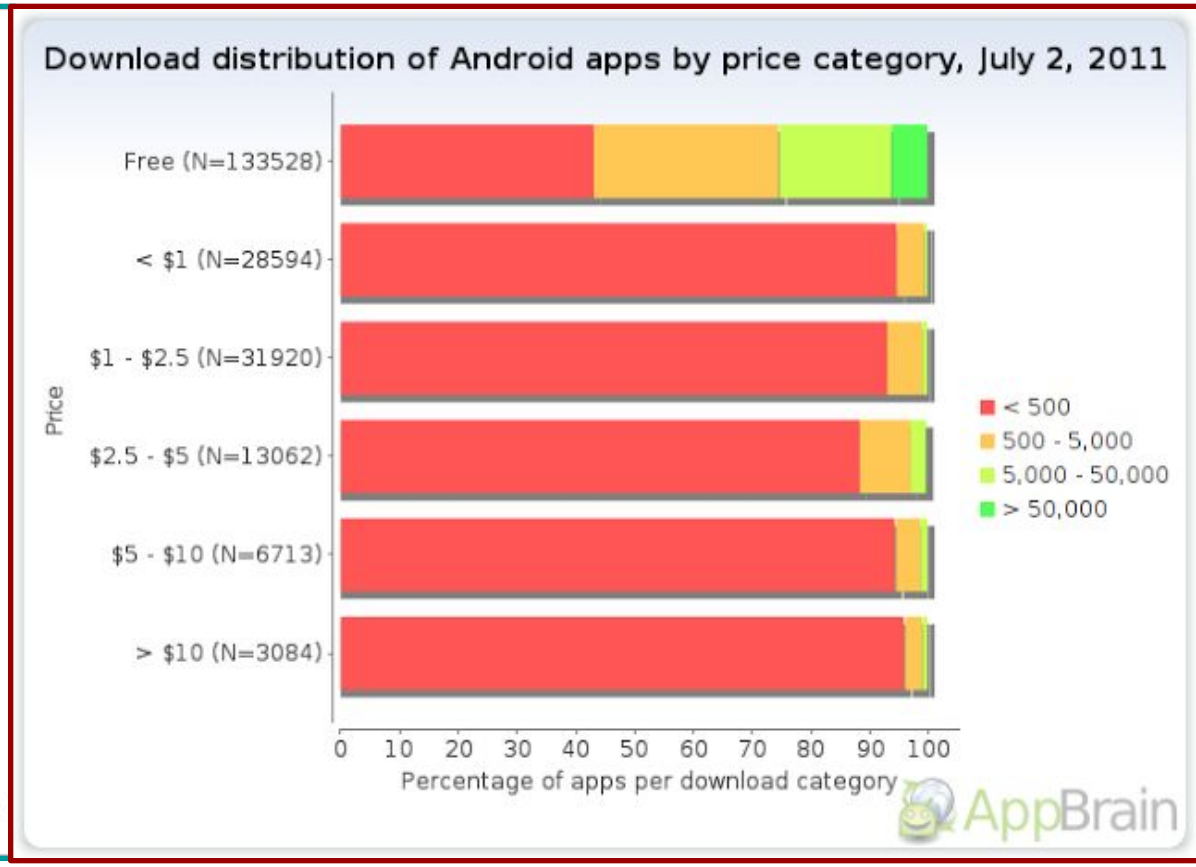
Android ... When?

ANDROID APP CATEGORIES



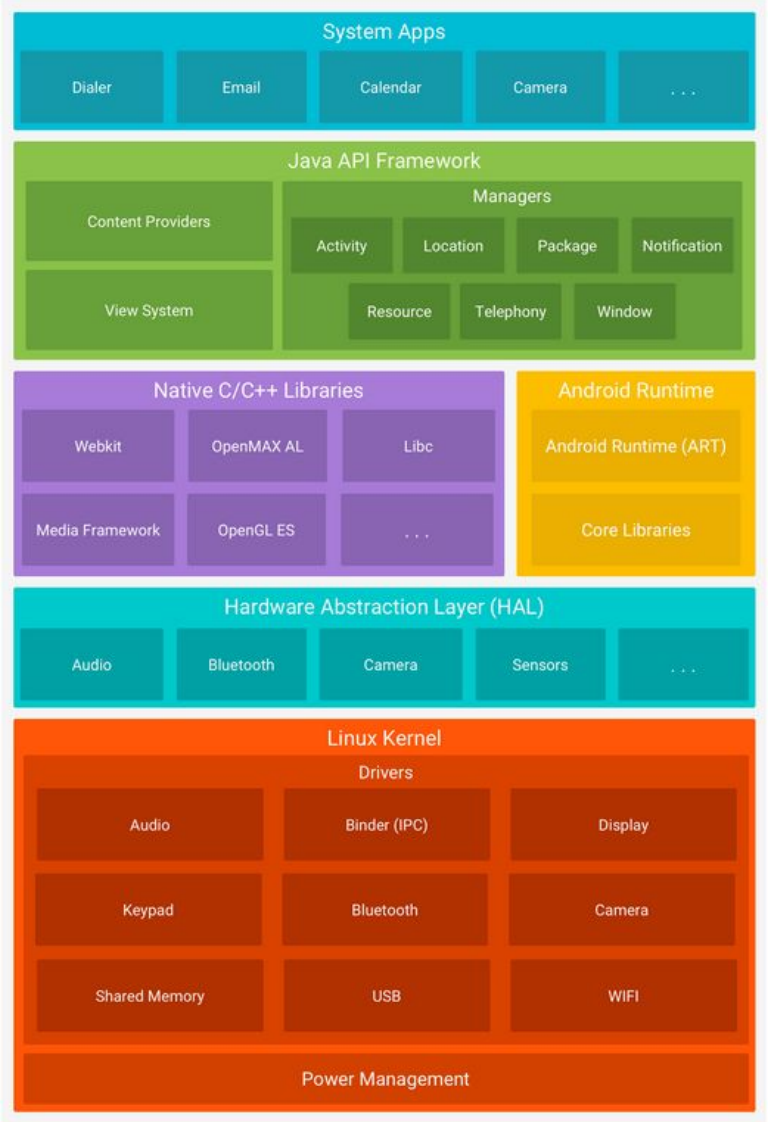
http://www.appbrain.com/stats/android-market-app-categories

ANDROID APP PRICE



http://www.onlinemarketing-trends.com/2011/07/android-marketplace-top-5-statistics.html

The Android Architecture



The Android Architecture

Built on top of **Linux kernel**

Advantages:

- **Portability** (i.e. easy to compile on different hardware architectures)
- **Security** (e.g. secure multi-process environment)
- **Power** Management
- **Android Runtime (ART)** relies on the kernel for threads and memory management
- **Manufacturers** build drivers on top of a reliable kernel

Kernel Security

- User based permission model
- Processes are isolated
- Inter-process communication (IPC)
- Resources are protected from other processes
- Each application has its own User ID (UID)
- Application Sandbox (process isolation)
- Verified boot

Kernel Security

- Android 5.0:
  - Mandatory Access Control (MAC) between system and apps, all third-party apps ran within the same SELinux context so inter-app isolation was primarily enforced by UID-based sandbox.
- Android 8.0:
  - limited system calls available to user-level apps
- Android 9.0:
  - all non-privileged apps with SDK version >= 28 must run in individual SELinux sandboxes, providing MAC on a per-app basis
- Android 10:
  - apps have a limited raw view of the filesystem, with no direct access to paths like /sdcard/DCIM. However, apps retain full raw access to their package-specific paths

The Android Architecture

HAL

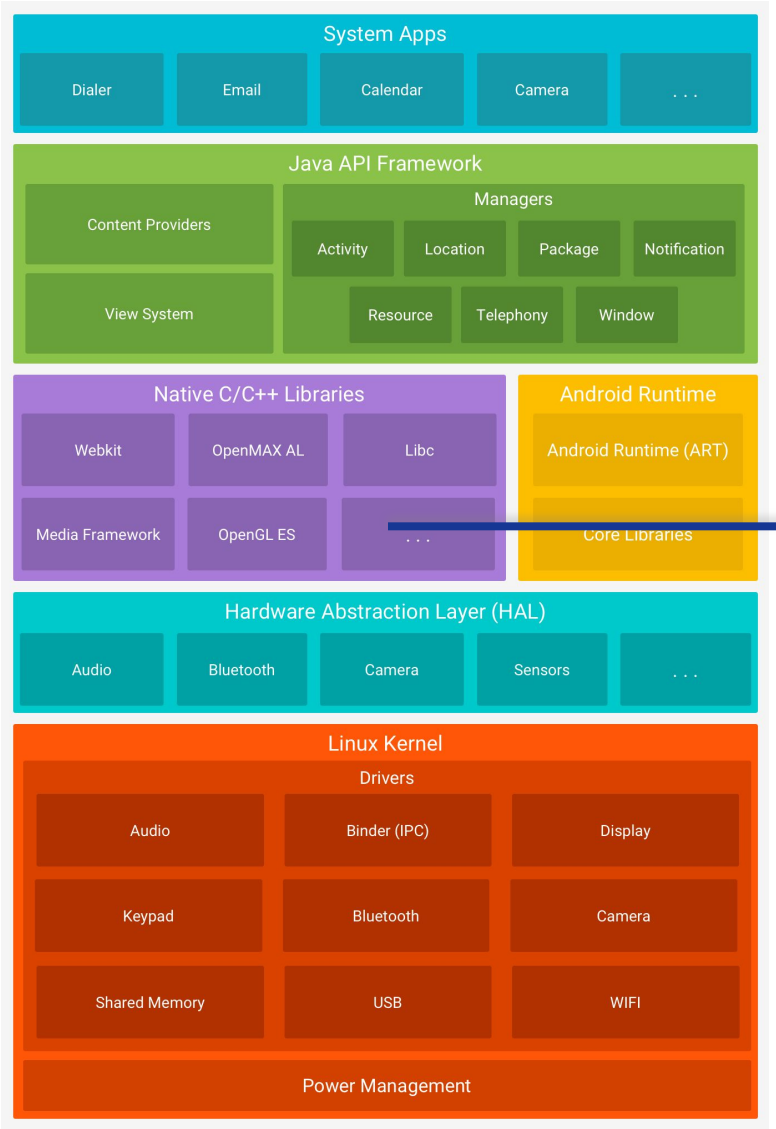
Advantages:

- **Shadows** the real device
- **Manages** different devices of the same type
- **Standard interfaces** to expose lower level capabilities to higher level APIs

HAL

- **Standard interface** that manufacturers have to implement – Android is **agnostic** about lower level driver implementations
- Application developers rely on **common APIs**
  - **Depending on the hardware**, **appropriate libraries** are loaded

The Android Architecture



Native **Libraries** (C/C++ code)

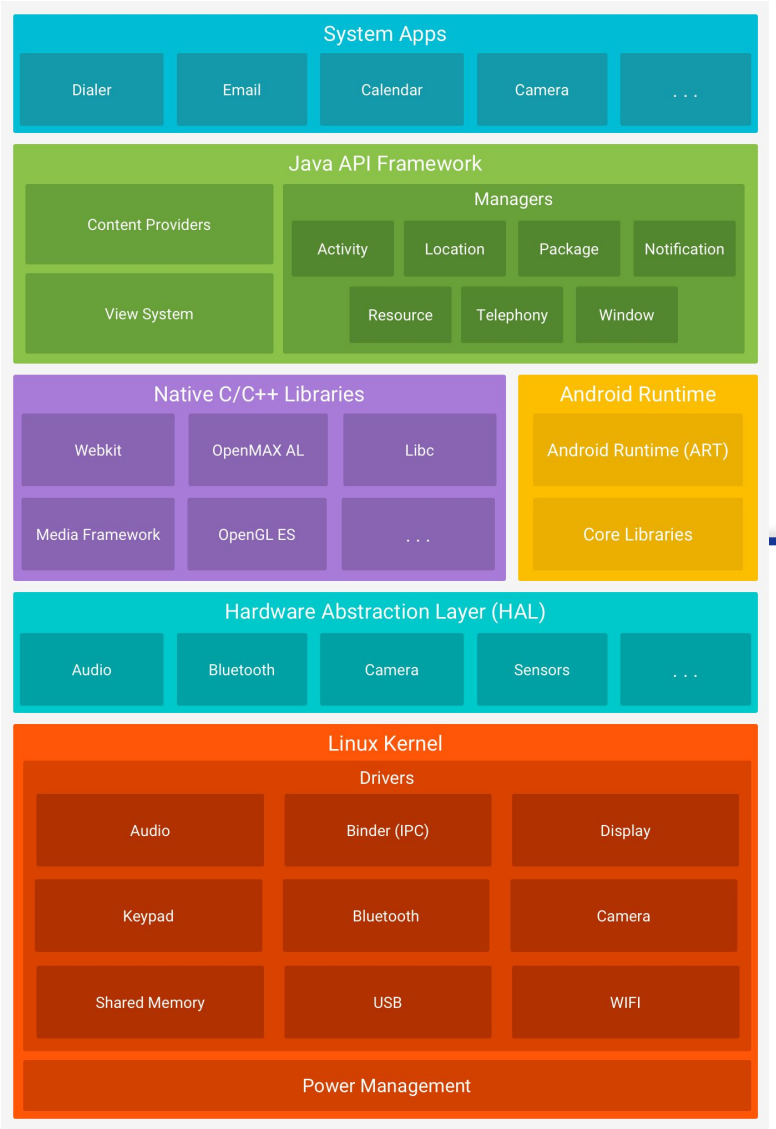
- **Graphics** (Surface Manager)
- **Multimedia** (Media Framework)
- **Database DBMS** (SQLite)
- **Font Management** (FreeType)
- **WebKit**
- **C libraries** (Bionic)
- ….

Android NDK

- Enables **C/C++** coding
- Useful if you want to **interact/extend** with some native libraries
  - **Performance**
  - **Reuse** your C/C++ libraries
- **JAVA APIs** are provided for most used libraries
- NDK can be installed as an **Android Studio plugin**

```
public class myNDKActivity extends Activity {
      public native void doNothing():
}
```

The Android Architecture

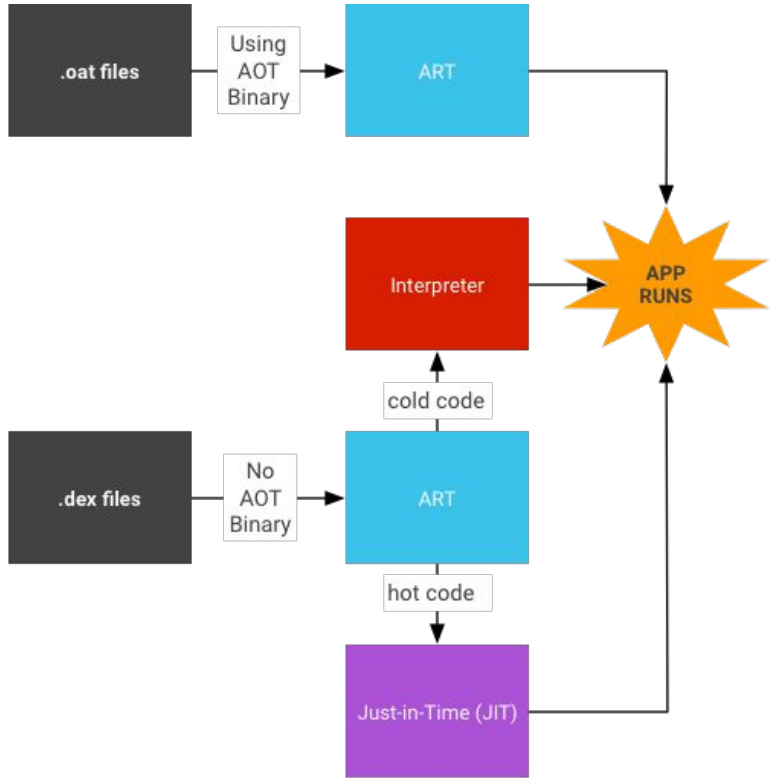


ART (VM)

- **Novel** Java Virtual Machine implementation (not using the Oracle JVM)
- **Optimized** for memory-constrained devices
- **Faster** than Oracle JVM
- **ART** optional from 4.4, mandatory from 5.0

ART

- Starting from Android 5.0, ART is used instead of Dalvik
  - Several enhancements such as stack size, error handling, AOT...
  - more at https://developer.android.com/guide/practices/verifying-apps-art.html
- Designed to run multiple VM on low end devices
- Runs DEX bytecode
- Ahead-of-time (AOT) and Just-in-time (JIT) compilation
  - AOT: At install time, ART compiles APPs using an on-device tool called dex2oat
    →Code compiled at installation
  - JIT: code profiling
    →Code partially interpreted when compiled not available
- Optimized Garbage collection



The Android Architecture

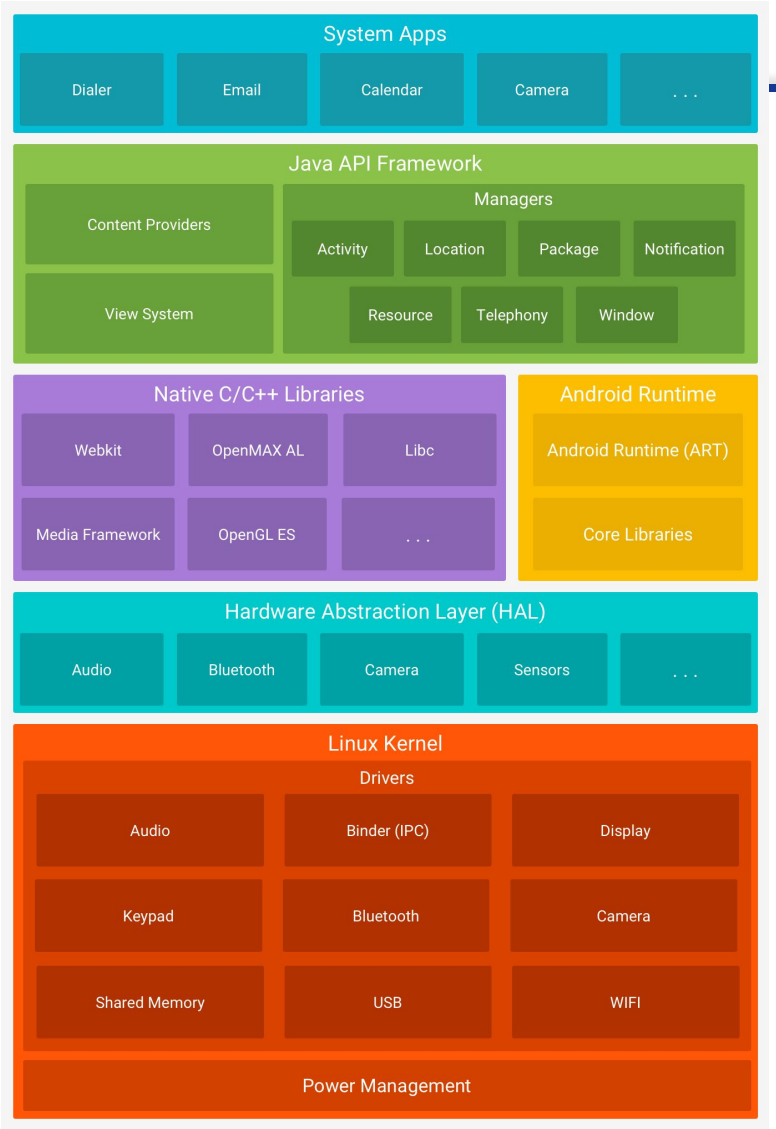
APIs (Core Components of Android)

- Activity Manager
- Packet Manager
- Telephony Manager
- Location Manager
- Contents Provider
- Notification Manager
- ….

Java APIs

- **View System**
  - Through which you build the **APP UI**
- **Resource Manager**
  - Through which you handle **resources**
- **Notification Manager**
  - Through which you can access to different kind of **notifications**
- **Activity Manager**
  - Which handles the **Activity lifecycle** and provides a **back stack**
- **Content Providers**
  - To **share data** among APPs

The Android Architecture



Applications

(Written in **Java/Kotlin** code)

- **Android Play Store**
- **Entertainment**
- **Productivity**
- **Personalization**
- **Education**
- **Geo-communication**
- **….**

Android Applications Design

APPLICATION **DESIGN**:

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

Android Applications Design

APPLICATION COMPONENTS

- Activities & Fragments
- 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 exchange 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:

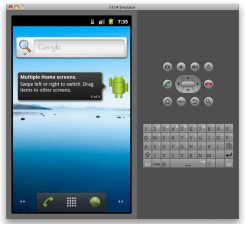
DECLARATIVE APPROACH

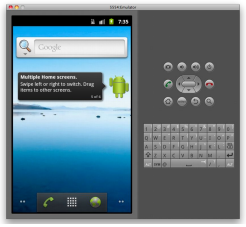
Example:

```
< 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

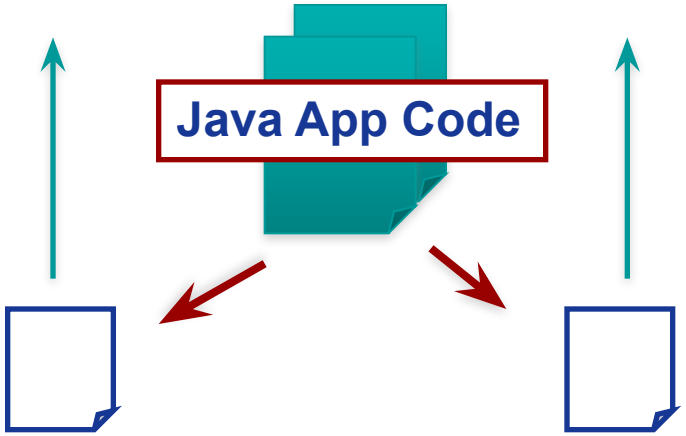
EXAMPLE





Device 1
HIGH screen pixel density

Device 2
LOW screen pixel density



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

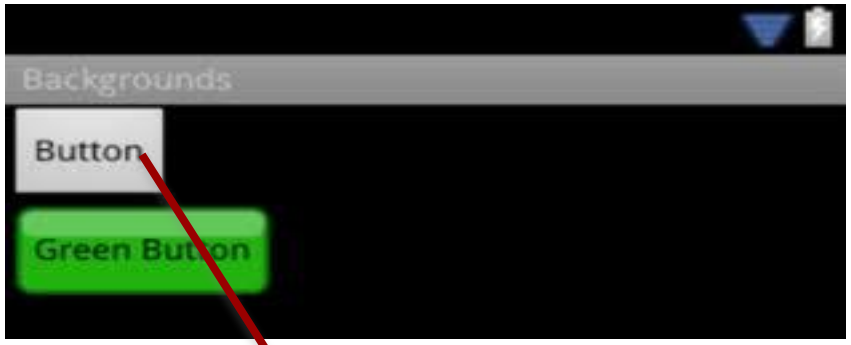
⇩

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 through **CALLBACKS** (from now on you need to know what these are)

Button

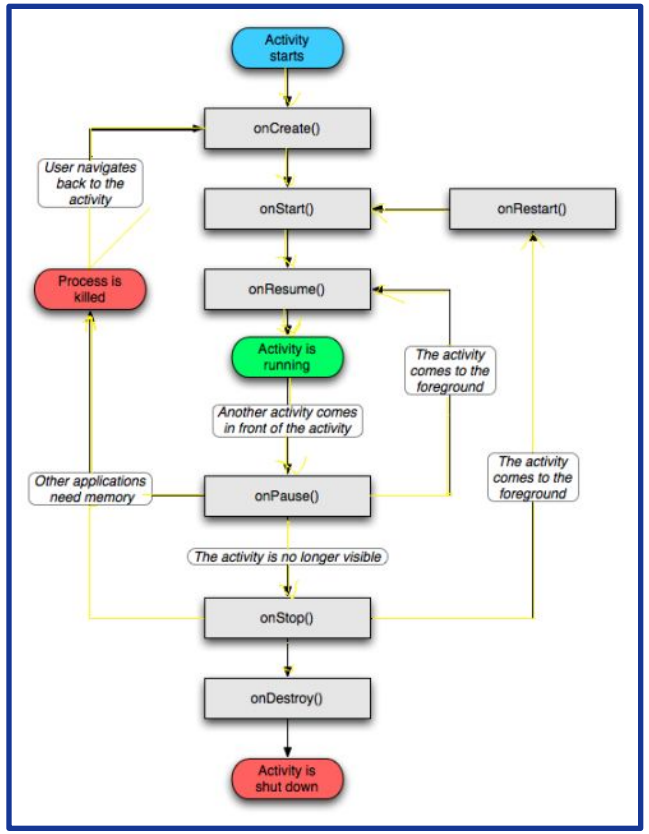


TextEdit

ESEMPIO

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

Android Components: Activities



- 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 constraints** (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.

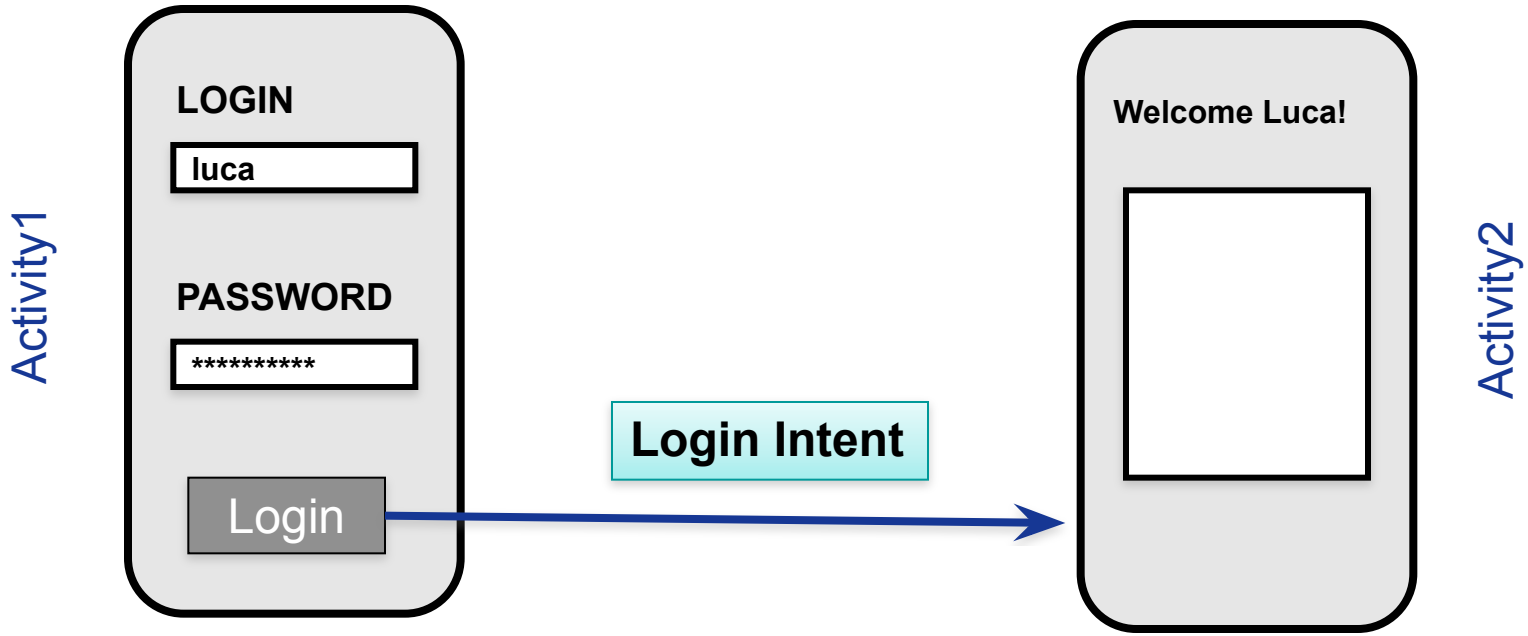
Called when the Activity is **partially visible**.

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

Called when the Activity is **dismissed**.

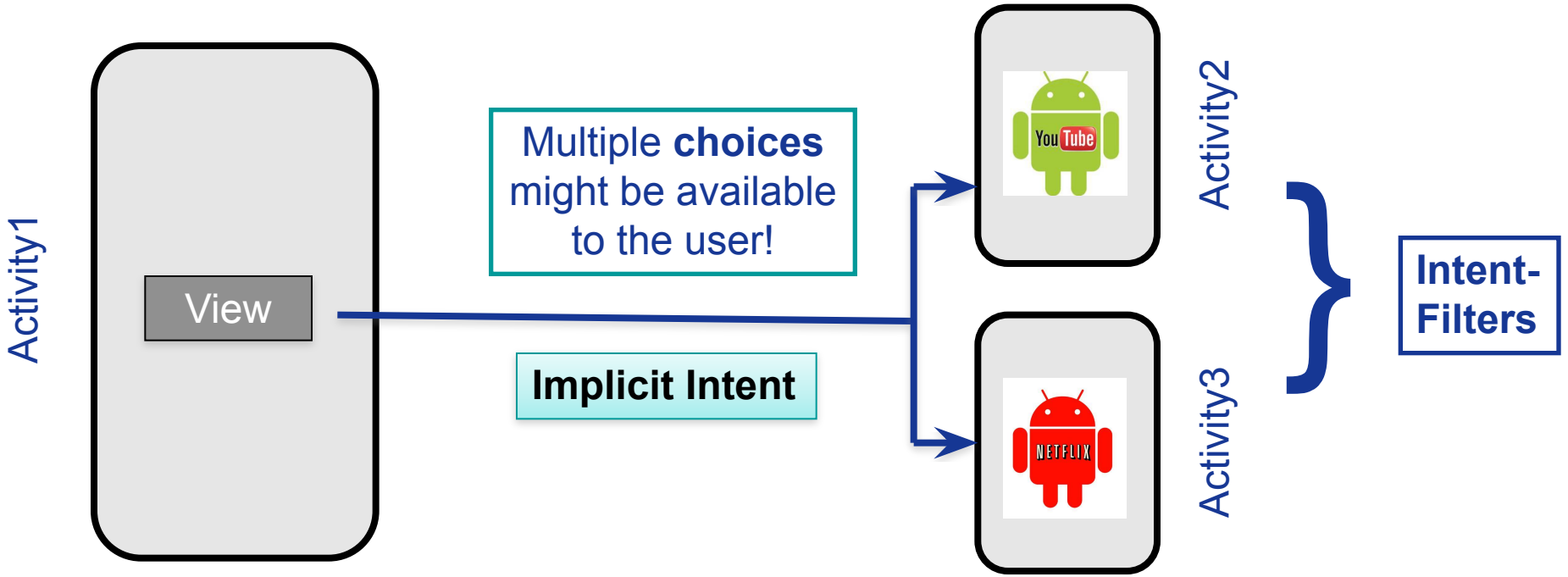
Android Components: Intents

- **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. Activity2)*.



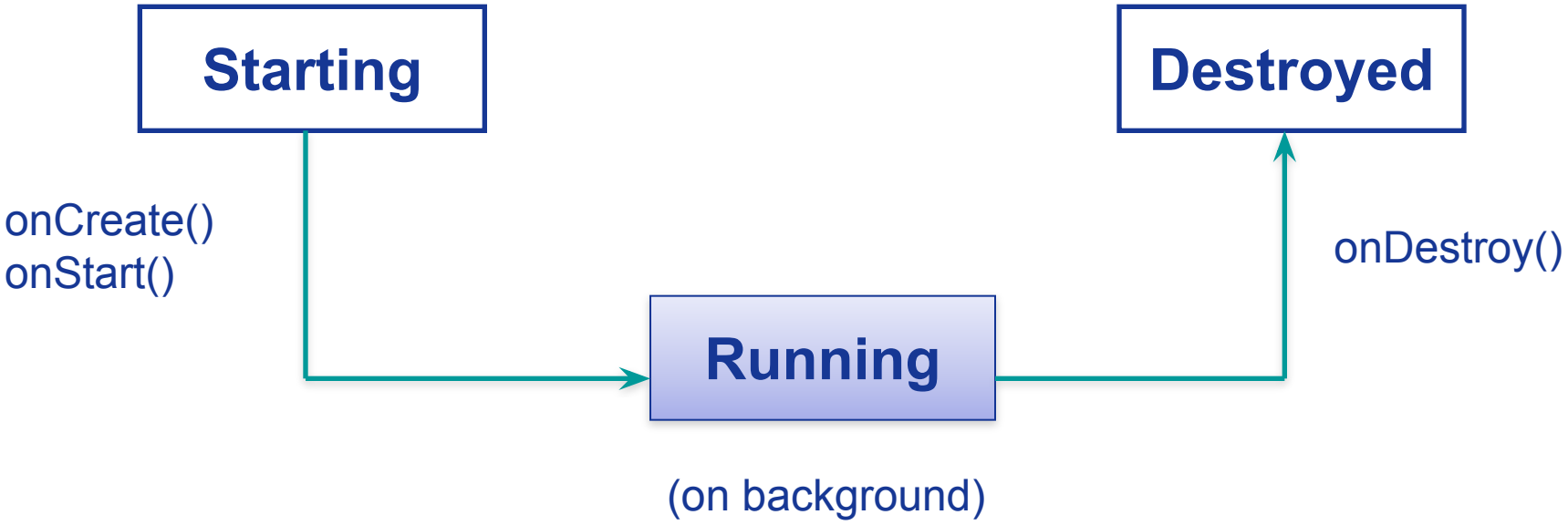
Android Components: Intents

- **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")*.



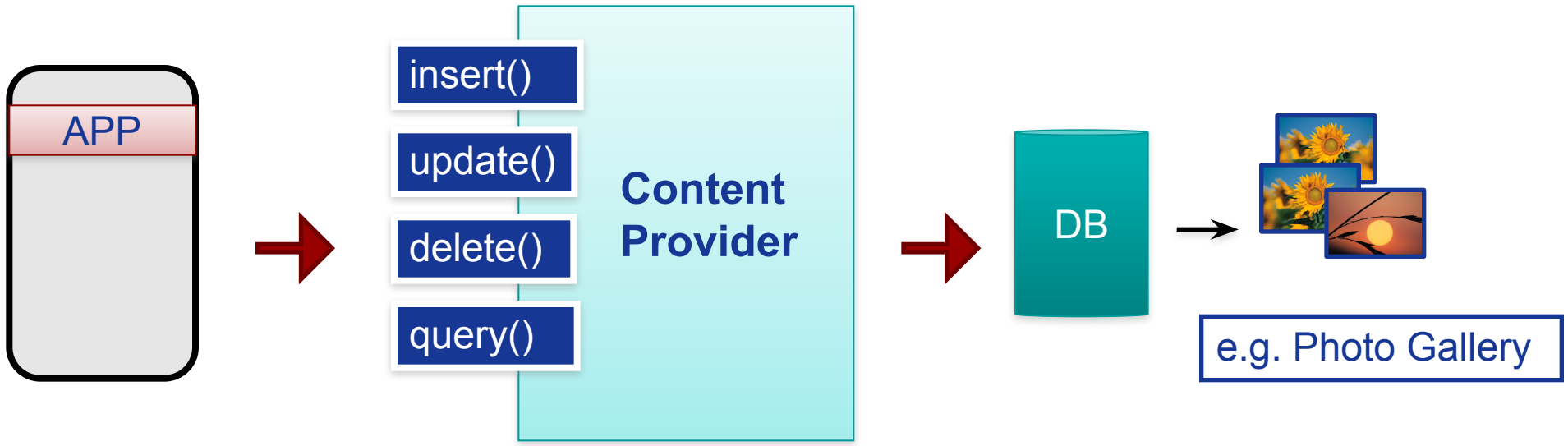
Android Components: Services

- **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:

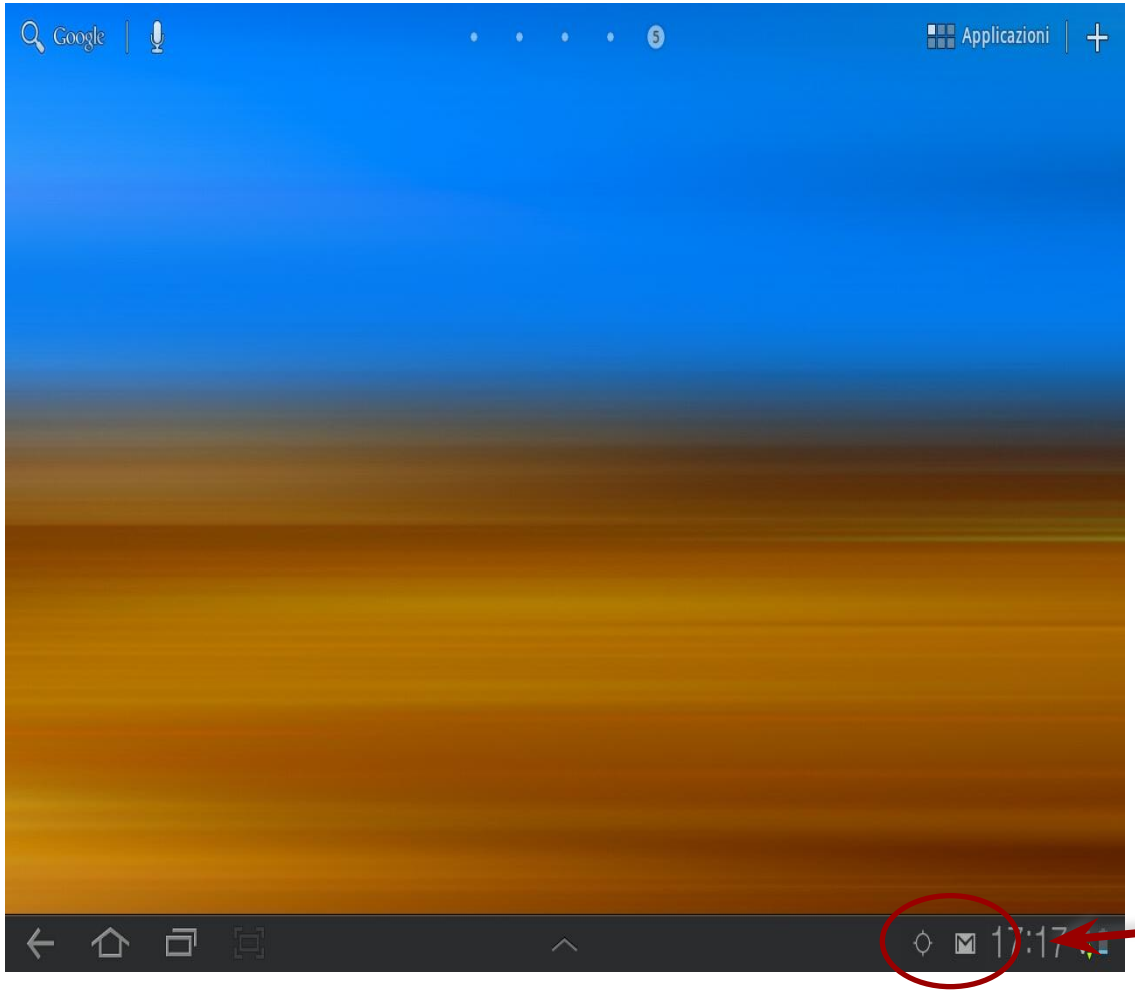


Android Components: Content Providers

- 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(s);
        }
    }
```

Android Components: System API

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

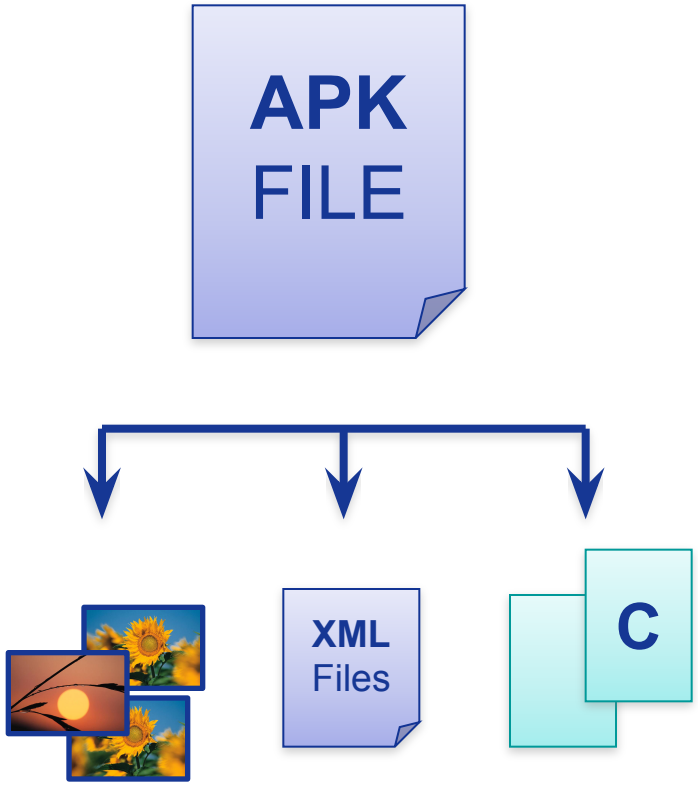
SOME EXAMPLEs …

- *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**:

Android Application Distribution



- Each Android **application** is contained on a single **APK** file.
  - Java **Byte-code**
  - **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. https://developer.android.com/studio/publish/app-signing
- 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.
  - Everything changes starting from 6.0

ANDROIDMANIFEST.XML

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

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

Permissions from 6.0

- Android Marshmallow (6.0) introduces runtime permissions
  - Permissions are not requested at install-time, but when they are used
- **Bad behavior**: request everything on the first screen of the first launch of the app
- **Good behavior**: request permission only when needed
  - APP should work (with limited functions) even if some permissions are not granted
- Things about permissions evolved further...