Laboratorio Mobile and Social Sensing Systems

Smart = Communication+ Computing + Sensors
Communication: variety of technologies (wi-fi, cellular, bluetooth, NFC)
Computing: powerful processors (4-8 cores, GPU), able to execute VMs
Sensors: position (GPS), temperature, ambient light, pressure, camera, microphone, proximity (for the distance with the face of the user), fingerprint, heartrate, pressure, motion sensing, UWB (Ultra wideband), that allows to transmit a signal with a short spike in the signal and this can be used to estimate the distance between the two devices, computing the difference between the transmission of the signals, multiplied by the speed of light: d= (tr- tt)*c/2.
Small errors in the time measurements though will reflect strongly on the distance due to the order size of c. We need a strong accuracy, and this can be given by UWB.

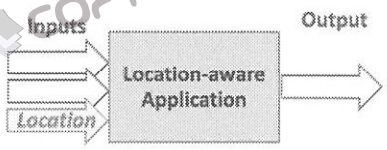
Different are the possible **communication technologies**, each suited for a different purpose:

Network Type	Speed	Range	Power	Common Use
WLAN	600 Mbps	45 m – 90 m	100 mW	Internet.
LTE (4G)	5-12 Mbps	35km	120 – 300 mW	Mobile Internet
3G	2 Mbps	35km	3 mW	Mobile Internet
Bluetooth	1 – 3 Mbps	100 m	1 W	Headsets, audio streaming.
Bluetooth LE	1 Mbps	100+ m	.01–.5 W	Wearables, fitness.
NFC	400 kbps	20 cm	200 mW	Mobile Payments

A communication with WiFi typically requires less energy than cellular connection, strong advantage.
Bluetooth requires the smallest amount of energy then other technologies.

Humans use a computing device while moving, we talk about **mobile computing**.
We have continuous network connectivity and the point of connection might change (Wi-Fi AP, cell tower).
From the user's perspective: same approach of desktop machine.
User always initiates all activities (launching apps).
Examples: Web browsing on a train, chatting while walking
We have no interaction with the surrounding environment.
Infrastructures (network equipment) is used just for providing connectivity.

In **location-aware computing**, the position must be one of the programs inputs.
Different position will have different output.
Examples: Maps app or Museum app.



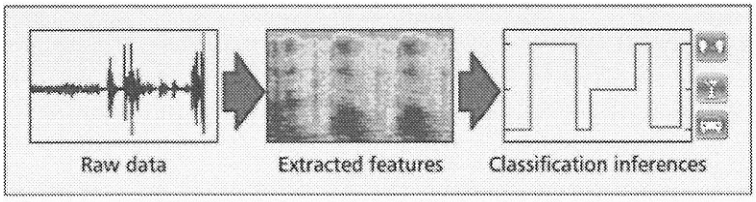
If behavior/output does not change when location changes the application is not location-aware.
Applications are executed on smartphones just for convenience.
In some cases, classification can be not so clear: ticket reservation app could use location to show the closest movie theaters, this is location-aware.

During last decades resources increased significantly, disk capacity, cpu speed, available, with the exception of energy.
We have to provide some energy saving techniques:

• CPUs able to adapt to the computational needs (most of the time smartphone is idle).
Ex. we can have distinct cores, each with different computational power and so different requirements of energy, and we can use the right one according to the need.
• Turn off screen aggressively
• Application specific methods: in video streaming reduce resolution, tuning it on the amount of power we have
• Make the user aware of remaining energy: with this battery level you can browse the web for 1h, or play music for 2h

Pervasive computing = Ubiquitous computing, technologies that disappear.
Goal: to assist humans in tasks, ex. reminders, suggestions, ease interaction with the environment, well-being, health, ...
System needs to understand user's intention and sensors are generally used to collect information about the physical space. System initiates activity, pro-active, user's explicit input may be not required.
Example: Google assistant, driving conditions to home, work, providing info about significant changes in weather conditions or suggest good restaurants in the nearby.
An example of pervasive computing can be obtained with wearable devices.
Wearables can provide patients with personalized health data, which could assist with self-diagnosis and behavior change interventions
Examples of applications:
• heart rate can be measured with an oximeter built into a ring
• muscle activity with an electromyographic sensor embedded into clothing
• stress with an electodermal sensor incorporated into a wristband
• physical activity or sleep patterns via an accelerometer in a watch
• a female's most fertile period can be identified with detailed body temperature tracking
• levels of mental attention can be monitored with a small number of non-gelled electroencephalogram (EEG) electrodes
• levels of social interaction (also known to affect general wellbeing) can be monitored using proximity detections to others with Bluetooth- or Wi-Fi-enabled devices
• pedometers can measure chronic conditions and provide info for sport professionals

Smartphones turned many pervasive applications into reality and they are used to sense the user or surrounding environment.
Machine learning commonly used to process sensor data.
Action to be inferred is hand-labelled to generate training data.
Sensor data is mined for combinations of sensor readings corresponding to action.
Data will be then opportunely preprocessed.



Example: Smartphone detects user's activity (e.g. walking, running, sitting) by classifying accelerometer sensor data.
Other pervasive systems are smart homes, smart buildings or smart cities.
Smart Homes: Continuously monitors elders who live in smart home, automatically dials 911 if elder ill, fall. Falls kill many old people who live alone.

Smart Buildings: Senses presence of people, ambient temperature, people flow, dynamically adjusts heating/cooling, they can save significant fraction of energy bill.
Smart Cities: Real time data from sensors embedded in street used to direct drivers to empty parking spots. Reduce traffic jam caused by people looking for parking.

Android

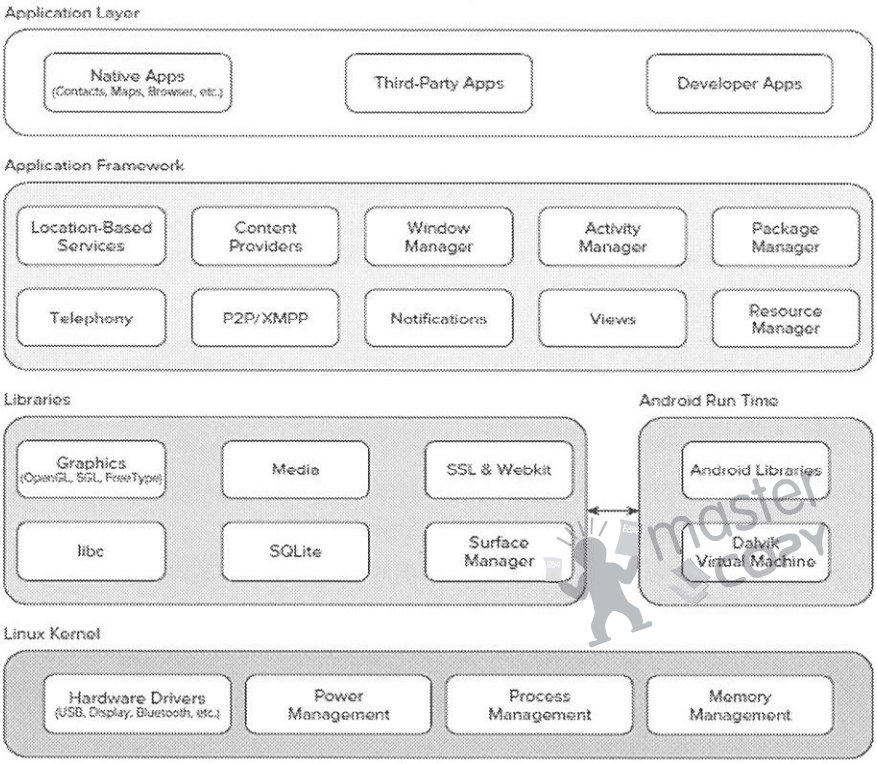
Android is the world's most popular mobile operating system, it's mainly open source and owned by Google, with the Open Handset Alliance, made of more than 80 companies.
History:

- 2005 Google buys Android Inc.
- 2007 OHA is announced and Android is presented
- 2008 Android SDK 1.0 is released
- 2010 Nexus One is commercialized
- 2012 800K+ Android devices activated every day

Android can be executed on a range of devices: smartphones, tablets (from 3.0), TVs (from 5.0), Autos (from 5.0), Wears (from 4.4), and Glasses.
Supported HW platforms:

- Supports ARM, x86, MIPS architectures (ARM from the beginning)
- Both 32bit and 64bit (the latter since Android 5.0)
- ARM/x86/MIPS or 32/64, transparent at the application level (unless you need native (C) code)

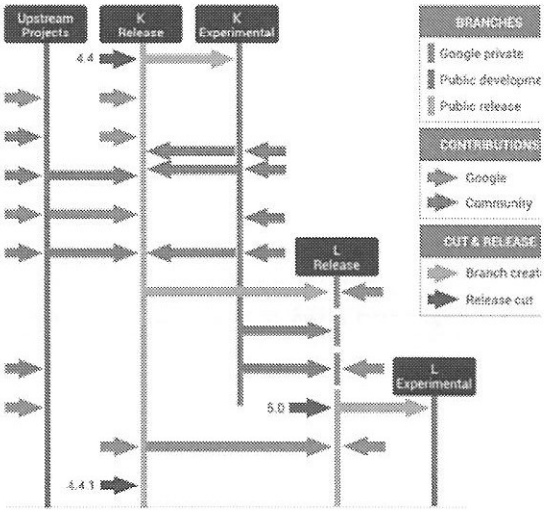
The **software stack** of the Android OS is the following:



We have a standard Linux kernel at the lowest level with few changes, mostly related to the scheduler for energy efficiency.
It takes care of managing processes, memory, and on top of it we have the android runtime, with a VM to execute the code and the Android libraries.
It uses a set of libraries, in particular Bionic, a smaller library available according to licensed preferred than the standard one.
At the application framework we have other libraries to interact with general functionalities.

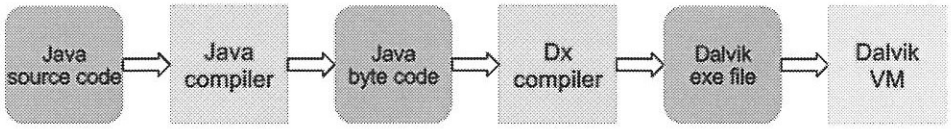
The activity manager manages all activities of the application.
The application layer relies with this set of components.

The **development model** is the following.
The current releasee of android is the K Release and then we have a parallel branching use to implement new features, then merged into the current release. When a new release is needed a new branch is created, at the beginning private, not findable in the Android Open Source project and then everything becomes public.
AOSP maintains the software.
Software is ported by OEMs to run on their own hardware.



Development tools are free, and no specific license is needed, and also is the distribution of the application, possible as APK file, basically an archive that contains code and resources.
Each file contains:
- dex executable
- resources (xml files, images, …)
- native libs
- Distribution is possible with app stores: Google Play Store and others from third parties, with the Web: just link the APK on a webpage, or by email.

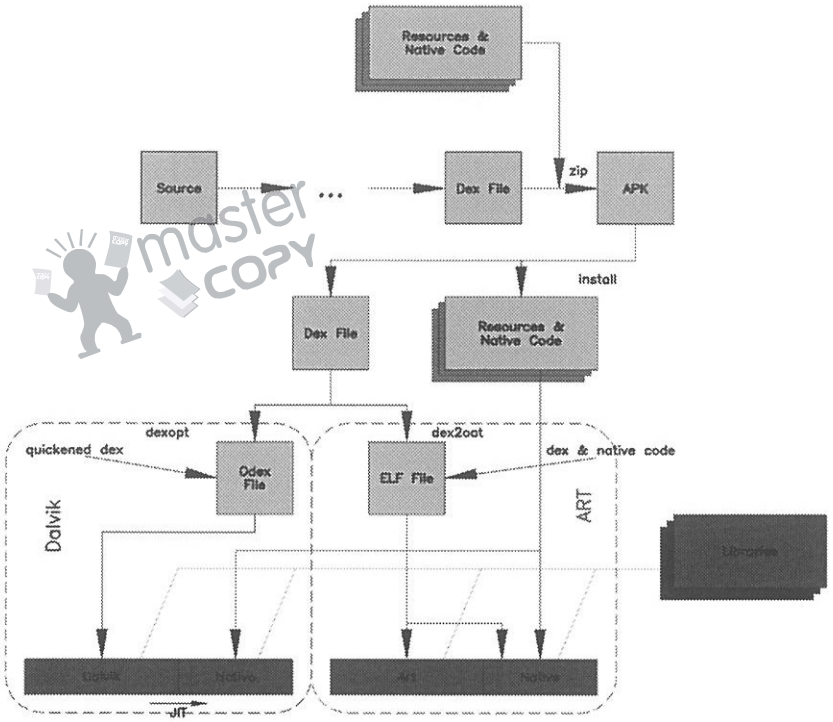
Dalvik VM ensures developers never have to worry about a particular HW implementation.
The Dalvik VM executes Dalvik executable files, a format optimized to ensure minimal memory footprint.
Developers create dex executables by transforming Java/Kotlin language compiled classes using the tools supplied within the SDK.



Java code is given in input to the compiler that produces bytecode, which is given in input into the Dx compiler that produces an executable file for the Dalvik VM.
The Dalvik VM is registry based instead of the stack-based of the Java VM and it can execute code more efficiently on devices with limited resources.
The commercial reasons let us looks like that we execute Dalvik code, not java code.
The version used by Java is approximately JSE without GUI, but we have Android specific libraries.

The Dalvik VM has an interpreted execution, it does the operation based on the underlying platform. This approach was not so efficient in terms of speed of execution, so an approach used was just to use compilation, translating the code of functions in native code so that the next time it will be executed the native code will be executed without having the overhead of interpretation, just compiling the code when needed, just-in-time compilation.
There were still some penalties, so they moved to **ART**: Android Run-Time (default since 5.0).

We download the Dalvik executable but at execution time everything is translated into native code, of course we pay though installation requirements and more space needed but with advantages at execution speed.
ART is so faster (depends on application, some benchmarks 2x), and uses less energy (no compilation), spent only once.



If we use ART everything from APK is translated into Linux binary file and then executed.

Advantages:

- Runs on a large variety of devices and platforms
- Easier to build pervasive applications
- Development tools are free of charge and available for different OSes
- Larger user base
- Code is open source, even if actually some components are proprietary (e.g. Maps, Play store)

Disadvantages for different versions of Androids:

- Different APIs
- Available hardware depends on devices (even if same API level)
- Different UI models (some buttons have disappeared)
- Different appearance of UI elements
- Some vendors introduce their own customizations to the UI and may provide
- additional libraries
- If you use native code, you have to provide compiled binaries for the processor architectures you want to support