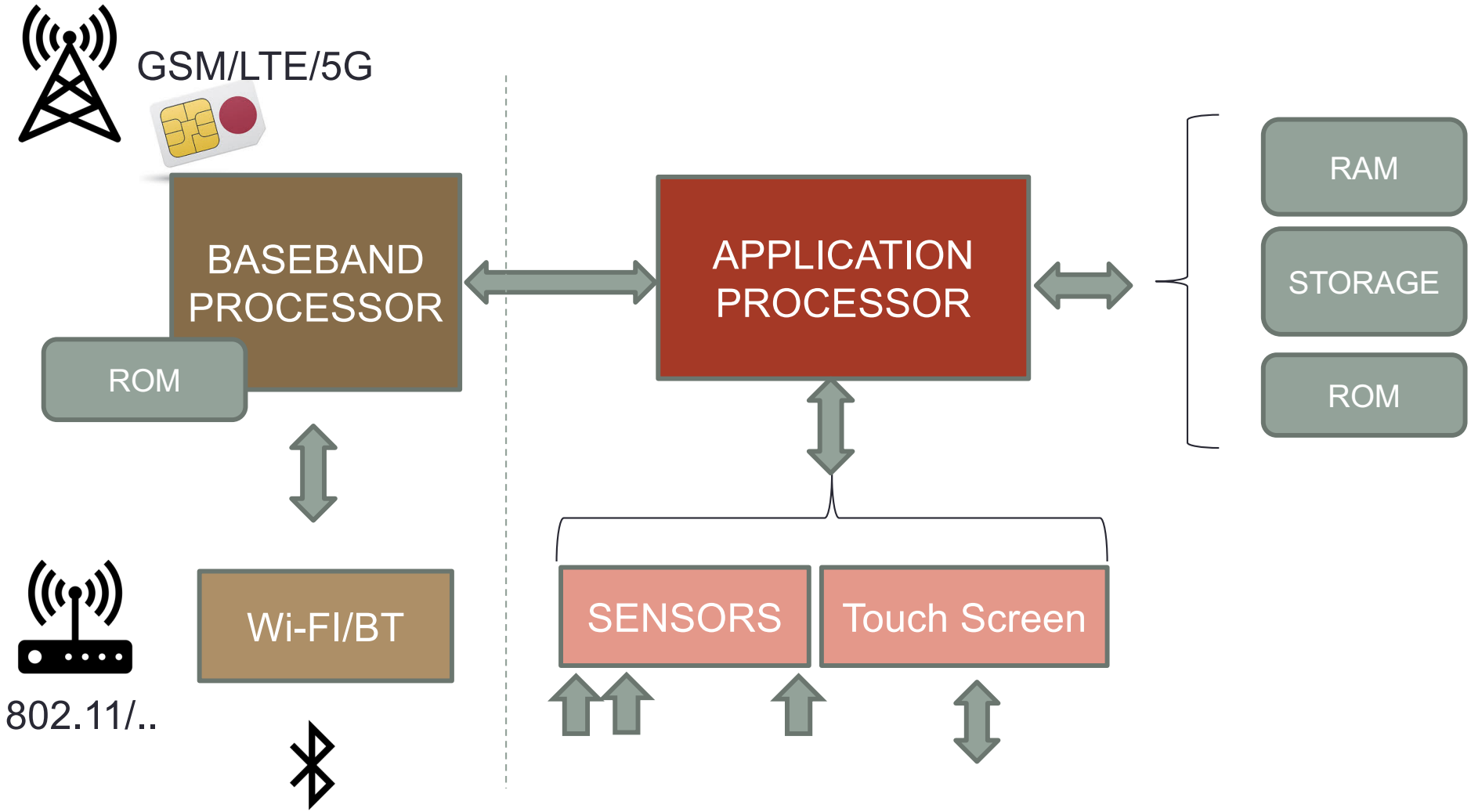
AN INTRODUCTION TO THE ANDROID OPERATING SYSTEM

Roberto Beraldi

Simplified Hardware Architecture

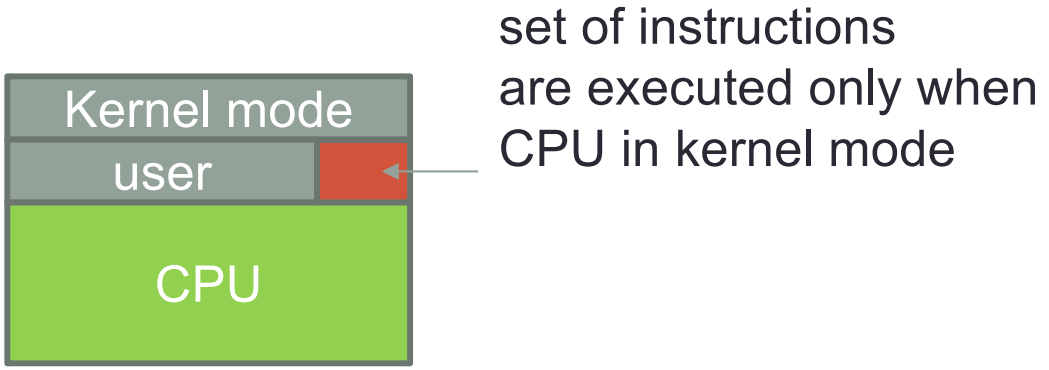


System software

- The **Baseband Processor** has its own Real-Time Operating System (*RTOS*), which is proprietary (backdoor has been found in the past)
  - Software interacting with cell towers must be **certified**
  - It is forbidden trying to access to a BS (you can't write programs that do that)
  - Has **hard timing constrains**
- The **Application Processor** runs a monolithic Linux based kernel
- AP and BP communicate via the Radio Interface Layer
  - RIL deamon

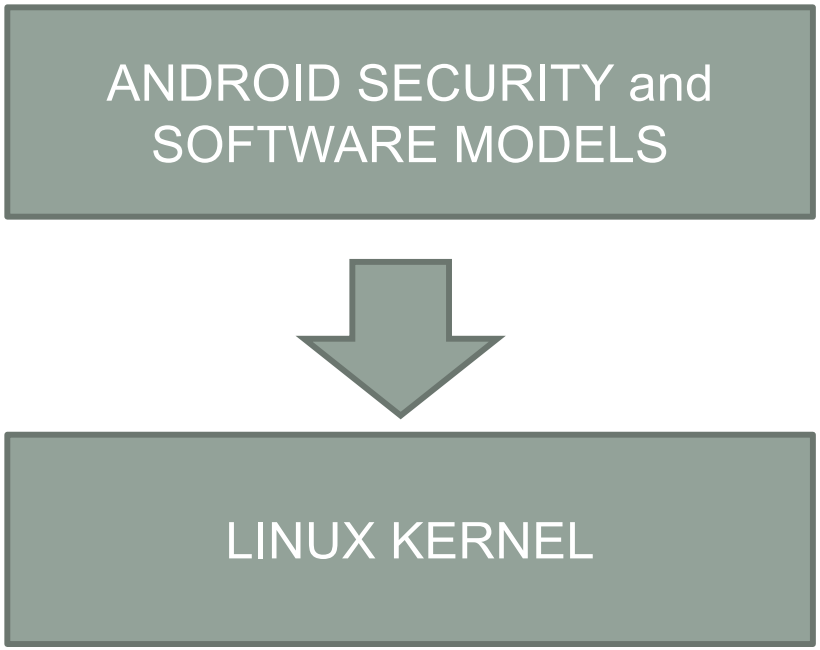
Application Processor

- AP for mobile devices have a very low consumption power (mW).
- In essence, an AP has at least two modes: **user** (unprivileged) mode and **kernel** (privileged) mode encoded in a CPU register as bit
- Any attempt to execute a privileged instruction when the AP is in user mode raises an exception
- For example, change the state of the CPU mode is a privileged instruction
- The only way to switch from user to kernel is by using a specific user-level instruction (e.g. *call*)
- This call changes the mode to kernel and runs a **trusted** handler



Android and linux

- The OS needs to run in kernel mode
- Code that runs in kernel mode must be **trusted code**
- Installing the OS needs a **chain of trust** (verified boot):
  - Before being loaded a code is verified against its integrity (digital signature and certificate)
  - Hardware root of trust
- Installed applications need to be signed too
- The Android security model (permissions, etc.) and software model is implemented exploiting the underlying capabilities of a Linux Kernel



Linux for smartphones (aka Android OS)

- Not a distribution, since it includes important changes to the kernel
- There are important changes related to
  - Memory Management
  - Inter Process Communication
  - Process Management
- Trusted Execution Environment (TEE)
- SELinux
- Several “managers” supports the OS to take management decisions (i.e., kill an application if memory is low, implements permissions, etc):

Some design criteria

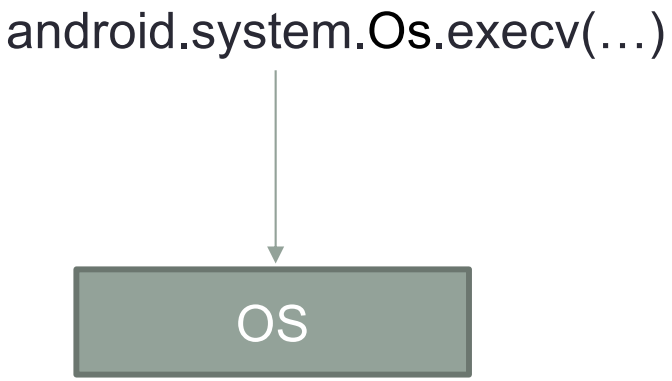
- Energy efficiency drives resource management
- Reducing memory access:
  - Virtual memory, but with no swap files
  - Storing the application state can be delegated to the developer
- Varying the CPU clock speed
- Enhance Security and Privacy
  - TEE
  - Permissions
  - Encrypted file

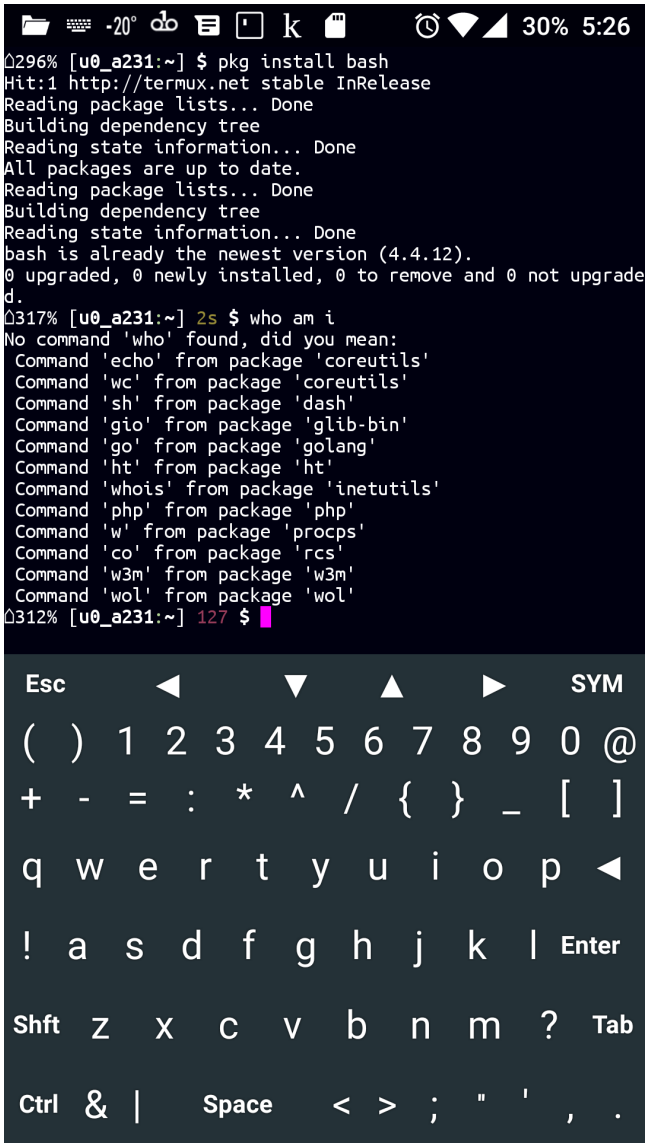
Linux users are mapped to apps

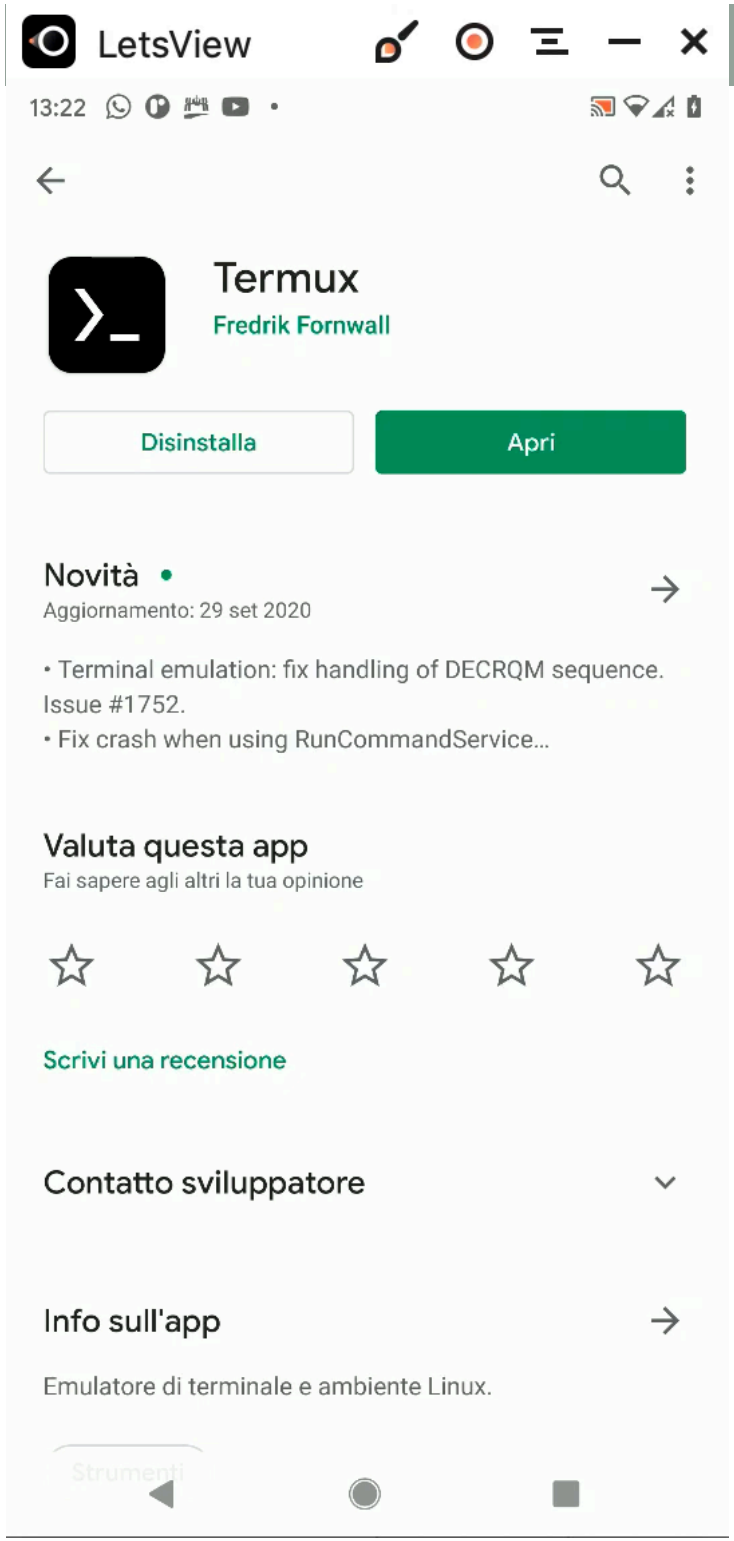
- A Linux user has a unique identification number, UID
- Linux also defines groups, each identified by an id, GID
- A user has a primary group and may belong other groups
- In Android a different UID is given per application (aka **package**) per user
- IDs is mapped to the human readable frormat, **u*XX*_a*YYY***
- The UIDs are not changed during app life on a device.
- Apps signed by the same developer can have the same UID
- Some IDs are reserved

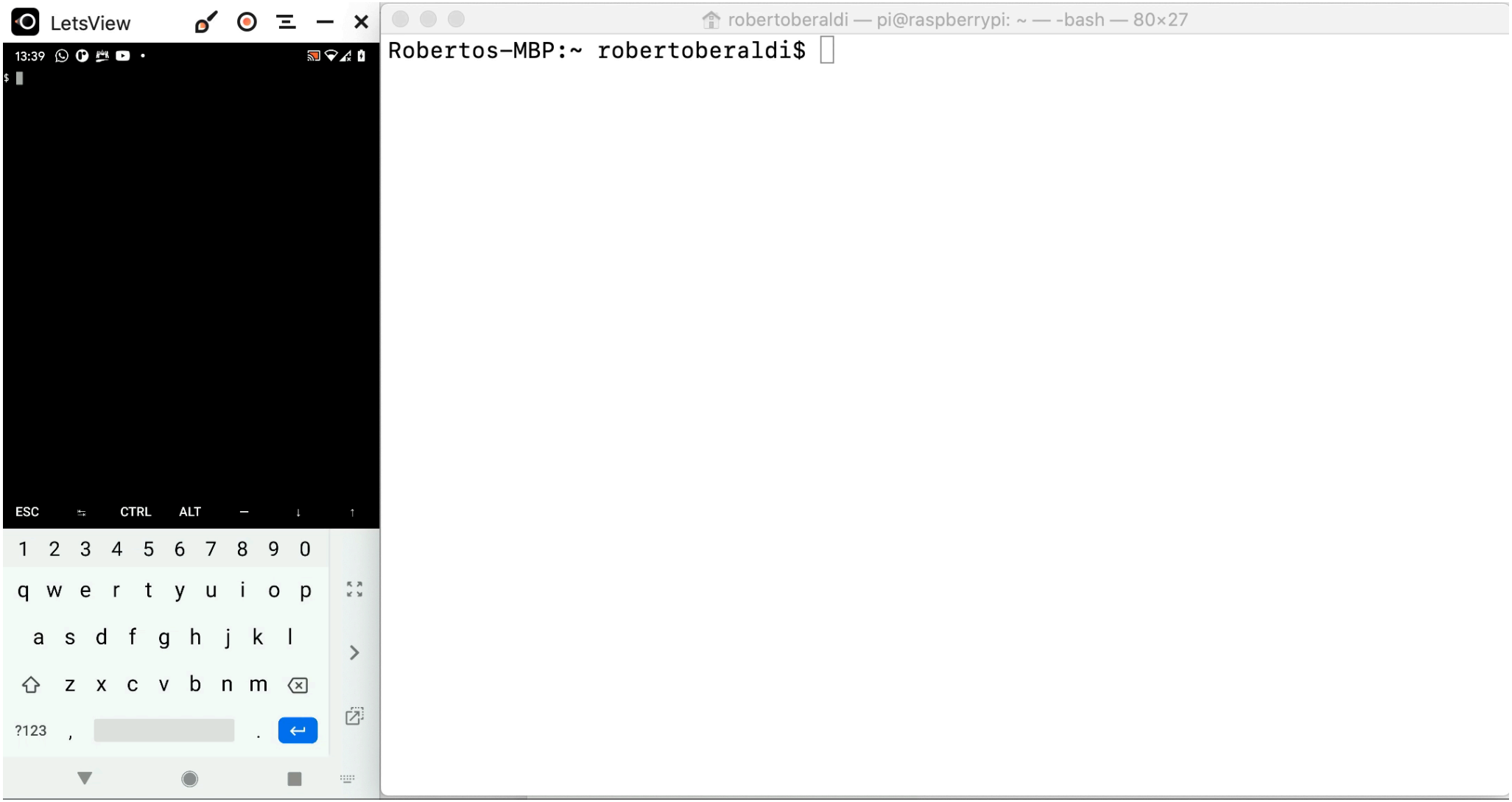
Gaining access to the OS

- Termux





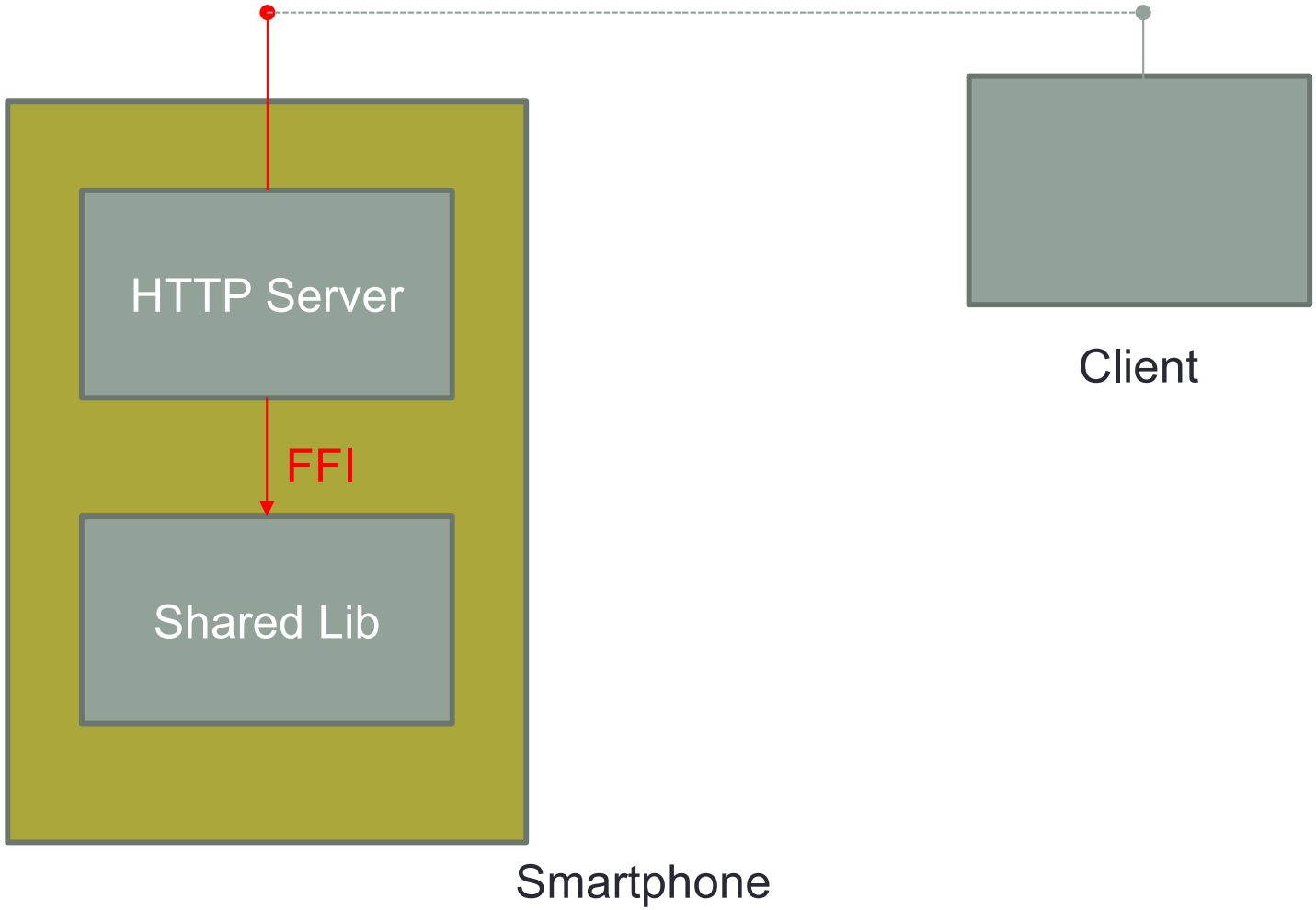

LetsView
13:22
Termux
Fredrik Fornwall
Disinstalla
Apri
Novità
Aggiornamento: 29 set 2020
• Terminal emulation: fix handling of DECRQM sequence. Issue #1752.
• Fix crash when using RunCommandService...
Valuta questa app
Fai sapere agli altri la tua opinione
Scrivi una recensione
Contatto sviluppatore
Info sull'app
Emulatore di terminale e ambiente Linux.
Strumenti

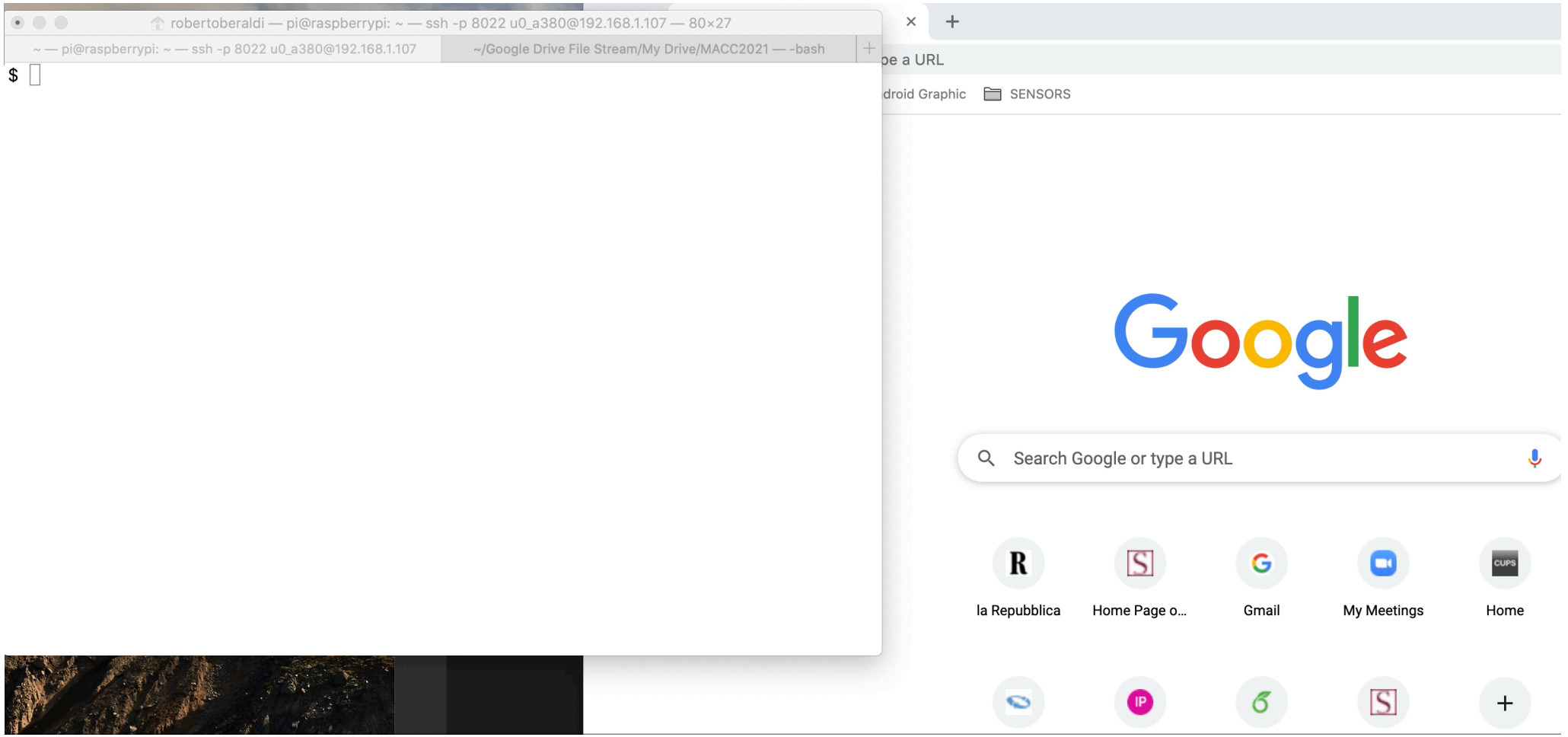

LetsView
13:39
$
ESC
CTRL
ALT
1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
?123
robertoberaldi — pi@raspberrypi: ~ — -bash — 80×27
Robertos-MBP:~ robertoberaldi$

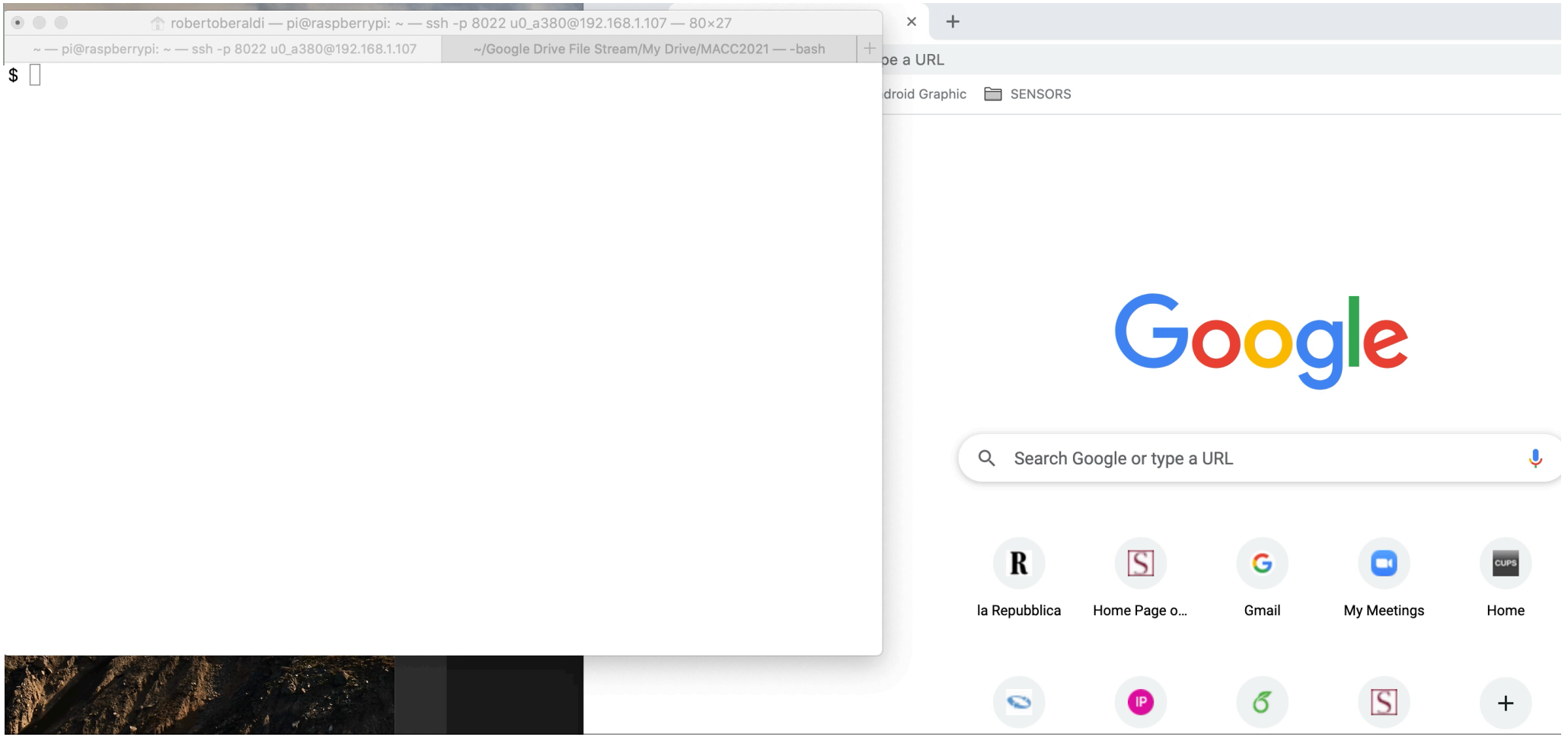
Termux commands

- On the smartphone Use
  - *whoami* to get the userID
  - *ip addr* to get the ip address
  - *passwd* to set the password
  - sshd to start the ssh deamon
  - **python –m http.server**

- On the desktop use:
  - *ssh –p 8022* userID@IPADDRESS


HTTP Server
FFI
Shared Lib
Client
Smartphone


robertoberaldi — pi@raspberrypi: ~ — ssh -p 8022 u0_a380@192.168.1.107 — 80×27
~ — pi@raspberrypi: ~ — ssh -p 8022 u0_a380@192.168.1.107
~/Google Drive File Stream/My Drive/MACC2021 — -bash
$
droid Graphic
SENSORS
Google
Search Google or type a URL
la Repubblica
Home Page o...
Gmail
My Meetings
Home


robertoberaldi — pi@raspberrypi: ~ — ssh -p 8022 u0_a380@192.168.1.107 — 80×27
~ — pi@raspberrypi: ~ — ssh -p 8022 u0_a380@192.168.1.107
~/Google Drive File Stream/My Drive/MACC2021 — -bash
$
droid Graphic
SENSORS
Google
Search Google or type a URL
la Repubblica
Home Page o...
Gmail
My Meetings
Home

Code running on the smartphone (python)

- from http.server import HTTPServer,BaseHTTPRequestHandler
- class Handler(BaseHTTPRequestHandler):
- def do_GET(self):
- self.send_response(200)
- self.send_header('Content-type','text/html')
- self.end_headers()
- self.wfile.write('Hello from the server...'.encode('UTF-8'))
- HTTPServer(('',8000),Handler).serve_forever()

running on the smartphone (C)

- int test(void){
- return 120;
- }

- Compiler options:
- gcc -fPIC -shared -o mylib.so myshared.c

Ahead of Time compilation

- In android any app at installation time is translated into an **executable** file
- This file is an ELF (Extendable and Linkable Format) file
- We now see some detail of this format
- Login using termux in the phone
- Useful tools: **gcc**, **readelf**, **hexdump**, **xxd** and **systcl**, programs to generate a simple a.out file and exploring the executable file

Application isolation via sandbox

- The goal of isolation is to avoid that applications **interfere** one with another, e.g., an app cannot read data of another applications

- Android exploits Linux users to implement isolation

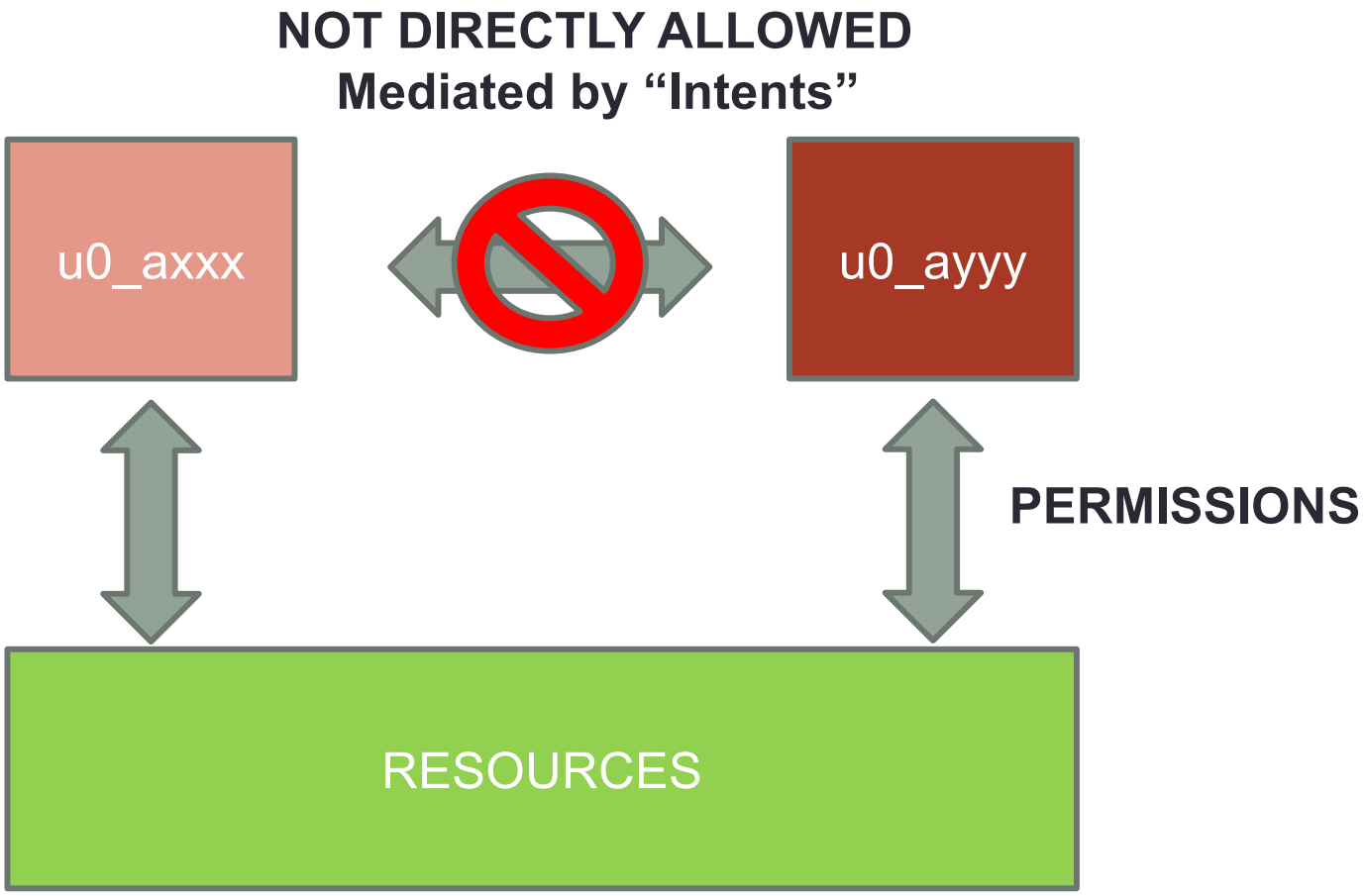
- By mapping applications to users, apps can in fact be sandboxed:

  - Processes get their own virtual memory space
  - Filesystem resources follows the DAC Linux model:
    - For example: --- --- r– restrict reading to only the owner

Sandboxing

- Each application during the installation also receives its own home directory,
  - for instance, */data/data/package_name*, where *package_name* is the name of an Android package, for example, com.ex.ex1
  - Application keeps its private data in this folder.
    - Apps signed with the same certificate may have the same UID and hence share the data
- Linux permissions assigned to this directory allows only the “owner” application to write to and read from this directory.

Application isolation via Permissions



Permissions to use a feature

- The purpose of a *permission* is to protect the privacy of an Android user.
- Android apps must request permission to access sensitive user data (such as contacts and SMS), as well as certain system features (such as camera and internet).

- adb shell pm list features → list all features of the device

Files

- Recall that in Linux uses a Virtual File System (VFS): “Everything is a file”

- The VFS include virtual file systems and files:
  - **/dev** (devices)
  - **/proc (**process)
  - ...

- In fact, beside a regular file, the **type** of a file can also be:
  - **l**ink, **d**irectory, **b**lock, **c**haracter [-,l,d,b,c]

- A file supports three operations:
  - **r**ead, **w**rite, e**x**ecute (listing in case of a directory)

Android permissions and file permissions

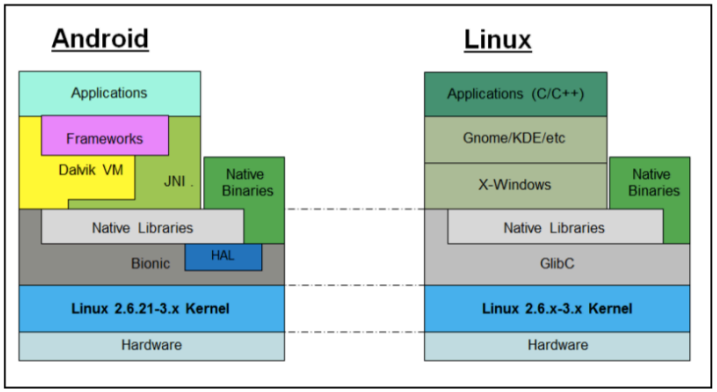
- Using filesystem permissions to files and device drivers, it is possible to limit processes in accessing some functionality of a device.
- If an app has requested the access to some device and the user has approved it, this application is also assigned theLinux group GID of the device
- Therefore, this app receives a possibility to read information from the device
  - For example: /dev/cam device driver.

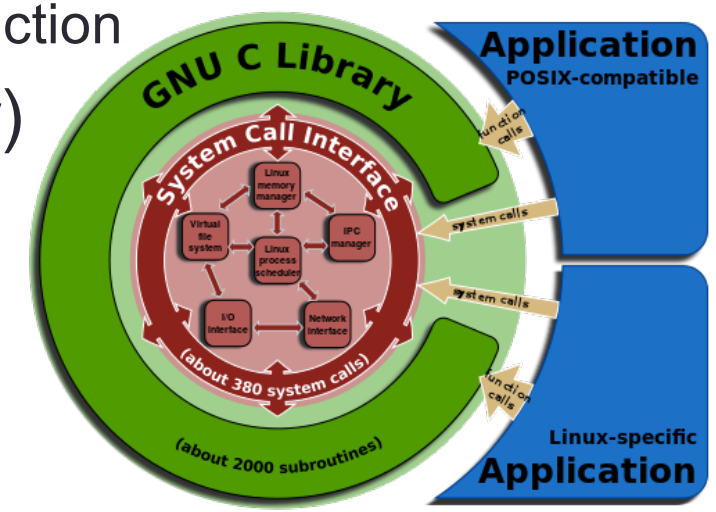
Application classes

- A1. **Native** apps running “directly on the CPU”
  - Source code: C/C++ → Binary for a specific ISA (ELF file)
- A2. **Java Native** Apps compiled for running on a VM
  - Source code: Java, Kotlin.. → bytecode
  - 5.0+: Applications are compiled into native ELF binaries on installation.
- A3. **Hybrid** Apps (based on **Foreign Function Invocation**, JNI)
  - NDK (Native Development Kit): A1 + A2
  - Xmarine: Use C#
- A4. **Mobile Web** apps (installed, but run in the browser)
  - html, JS,CSS
- A5. **Hybrid Web** apps
  - A4+A2: Web apps + **JS Bridge** to call Native code

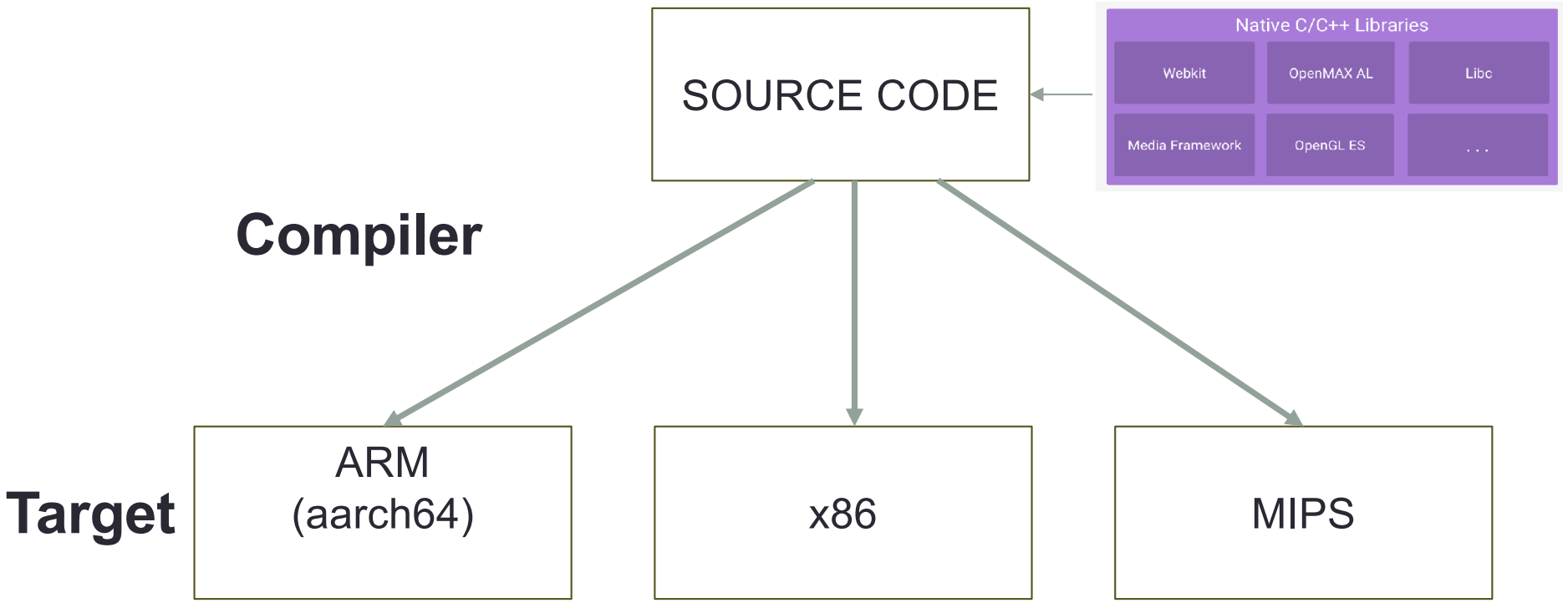
A1. Native applications

- Translated in arch dependent executable code
- Access to shared libraries
- Executed ‘directly’ by the CPU
- Two main development options
- *NDK* (we see later in the course)
  - Blended with java code
  - Source code contains the load library instruction
- *gcc* (for example via Termux or similar)
  - produces **a.out** (ELF format)





A1. Native applications

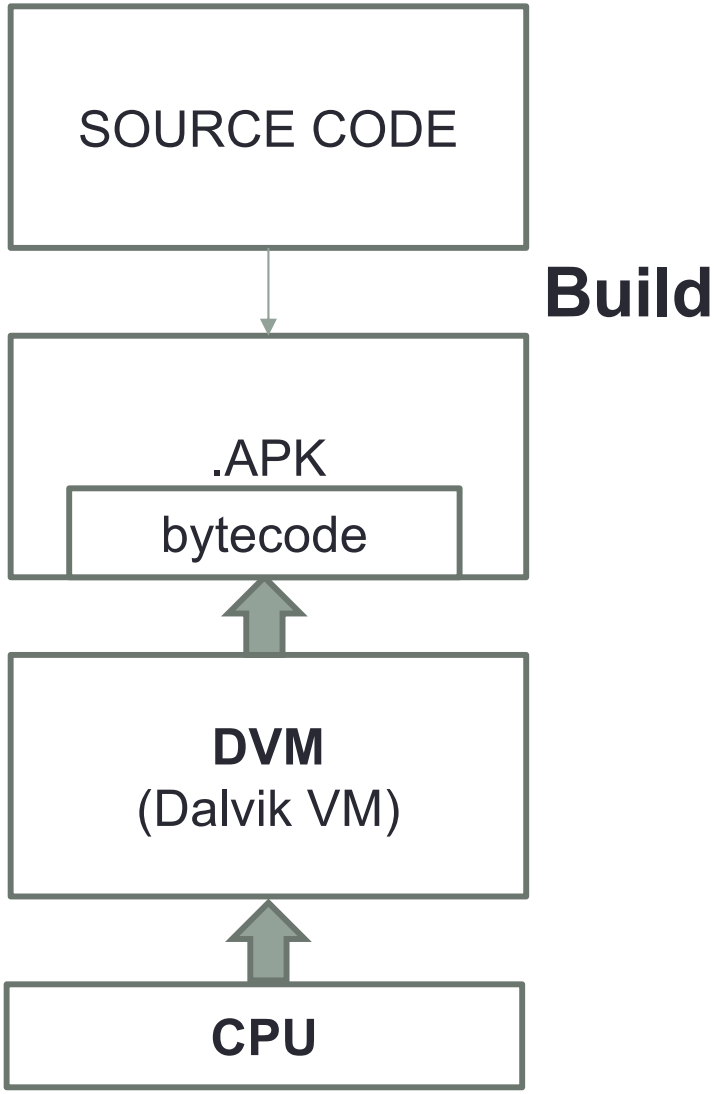


ELF Header

```
$ readelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           AArch64
  Version:                           0x1
  Entry point address:               0x4f0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          6664 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         8
  Size of section headers:           64 (bytes)
  Number of section headers:         24
  Section header string table index: 23
```

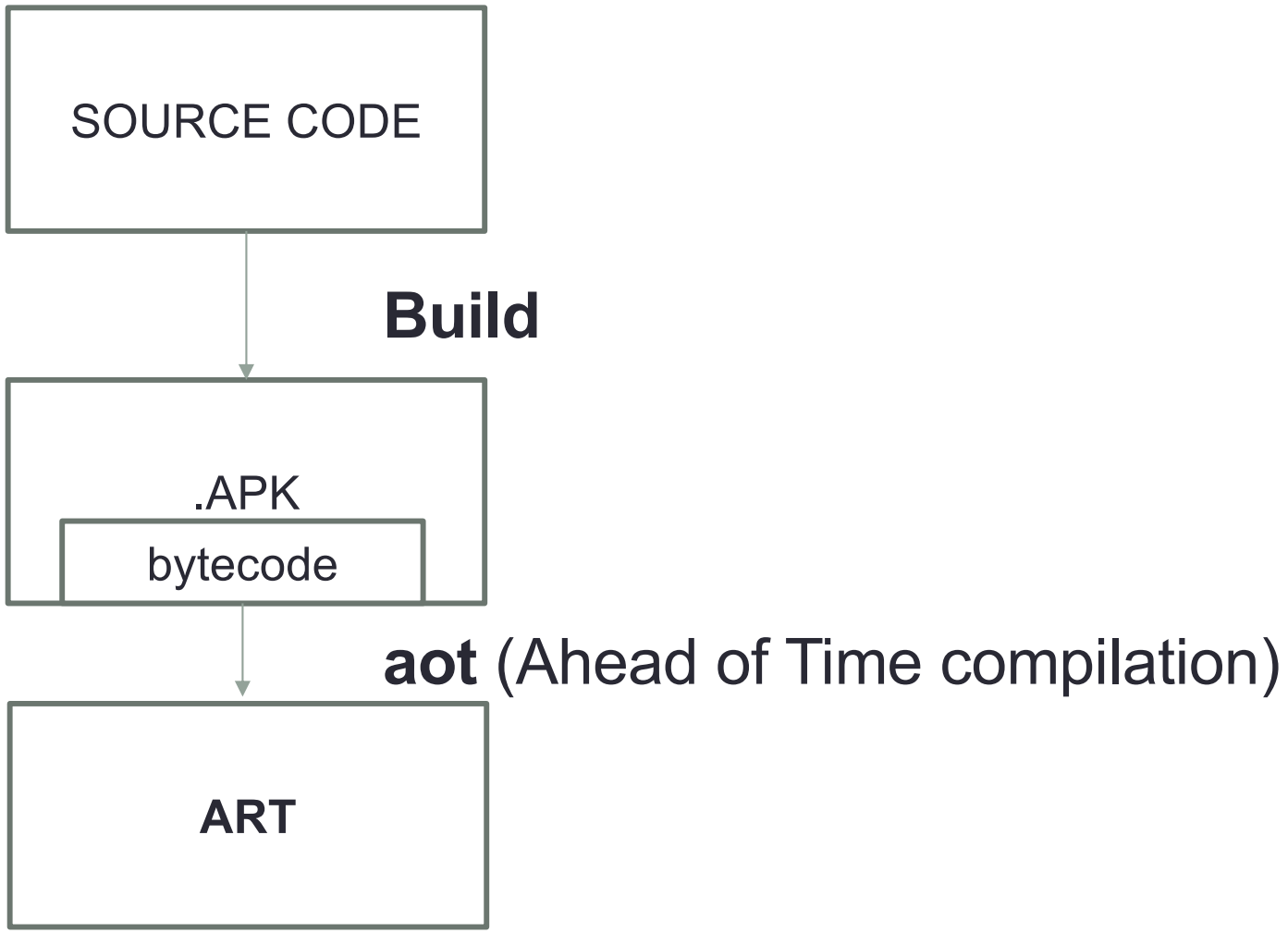
- 16 bytes equal to the MAGIC number
- Type of file
- How binary data are stored
-
- How to make an OS call
- Dynamic (position independent ..)
- ISA specification
-
- Where executable starts

A2. Java Native applications



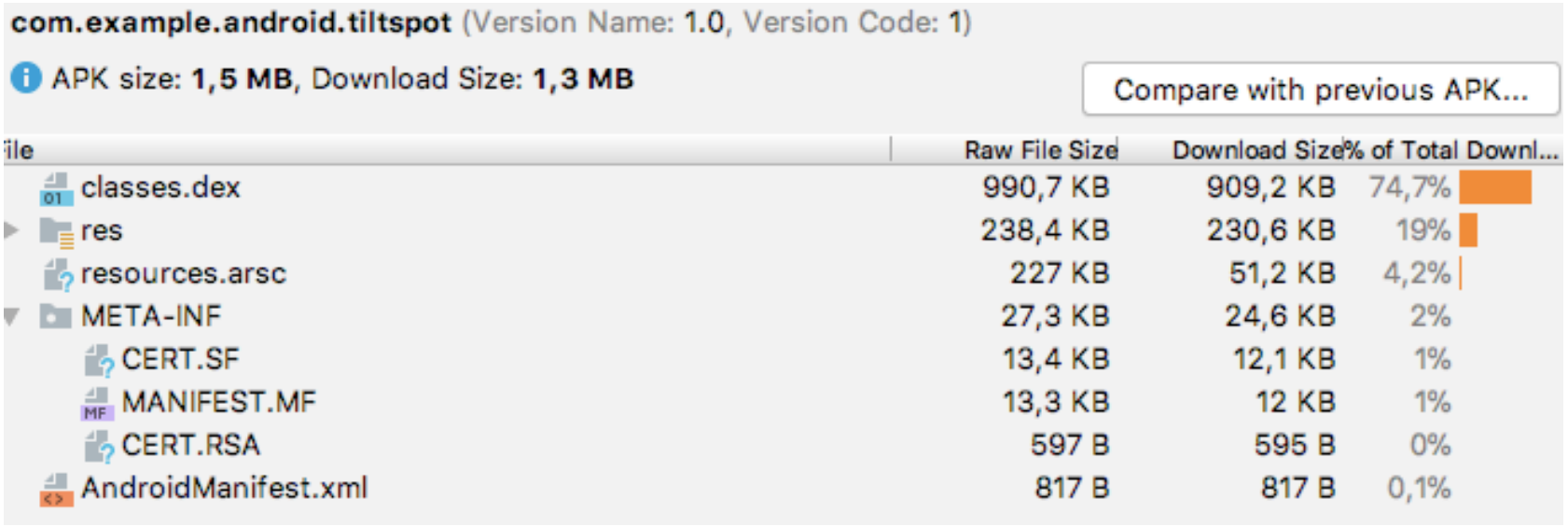
bytecode is hw-arch agnostic

A2. Java Native applications

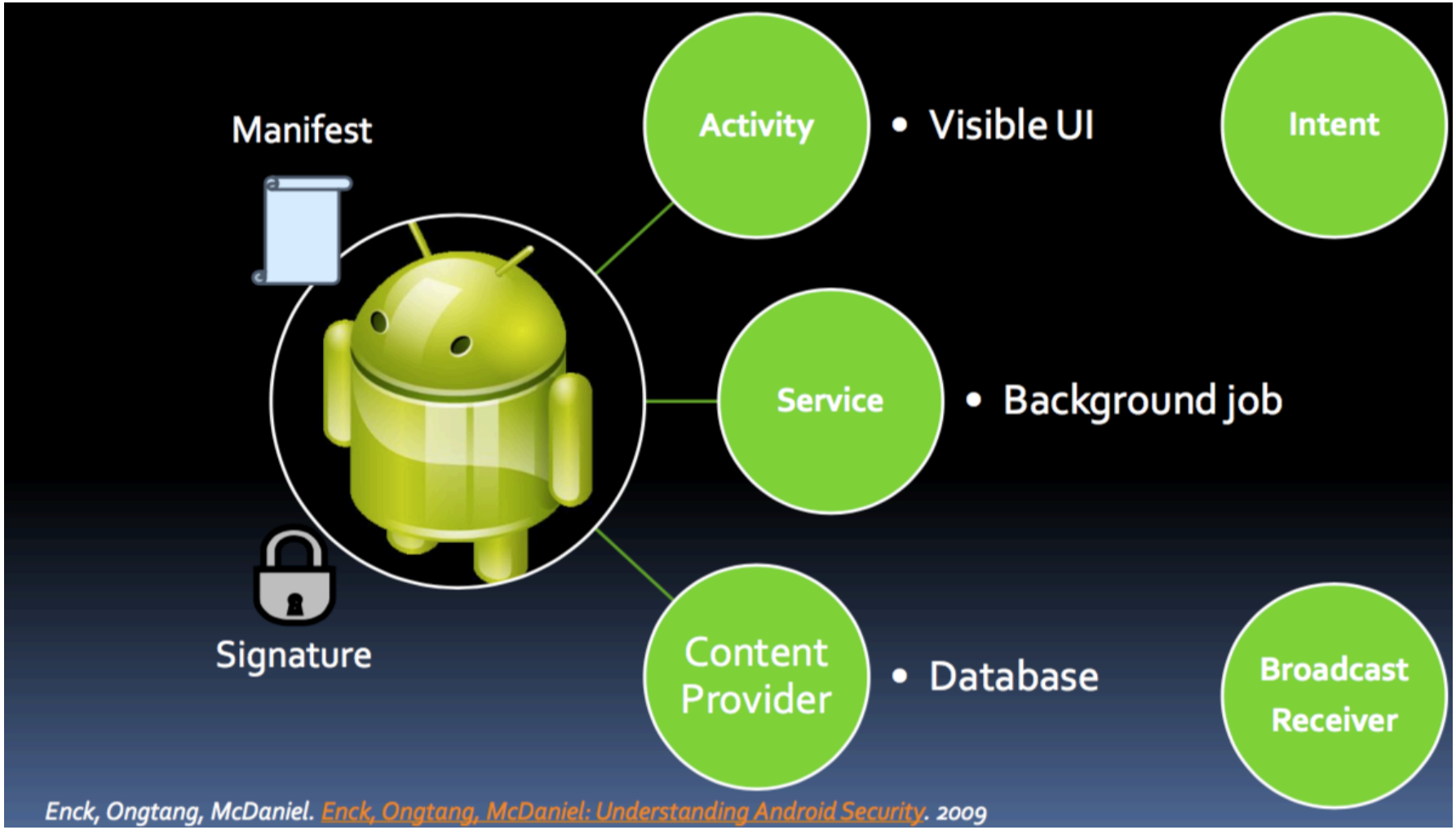


A2. Java Native applications

.apk



A2. Java Native applications



A3. Hybrid apps

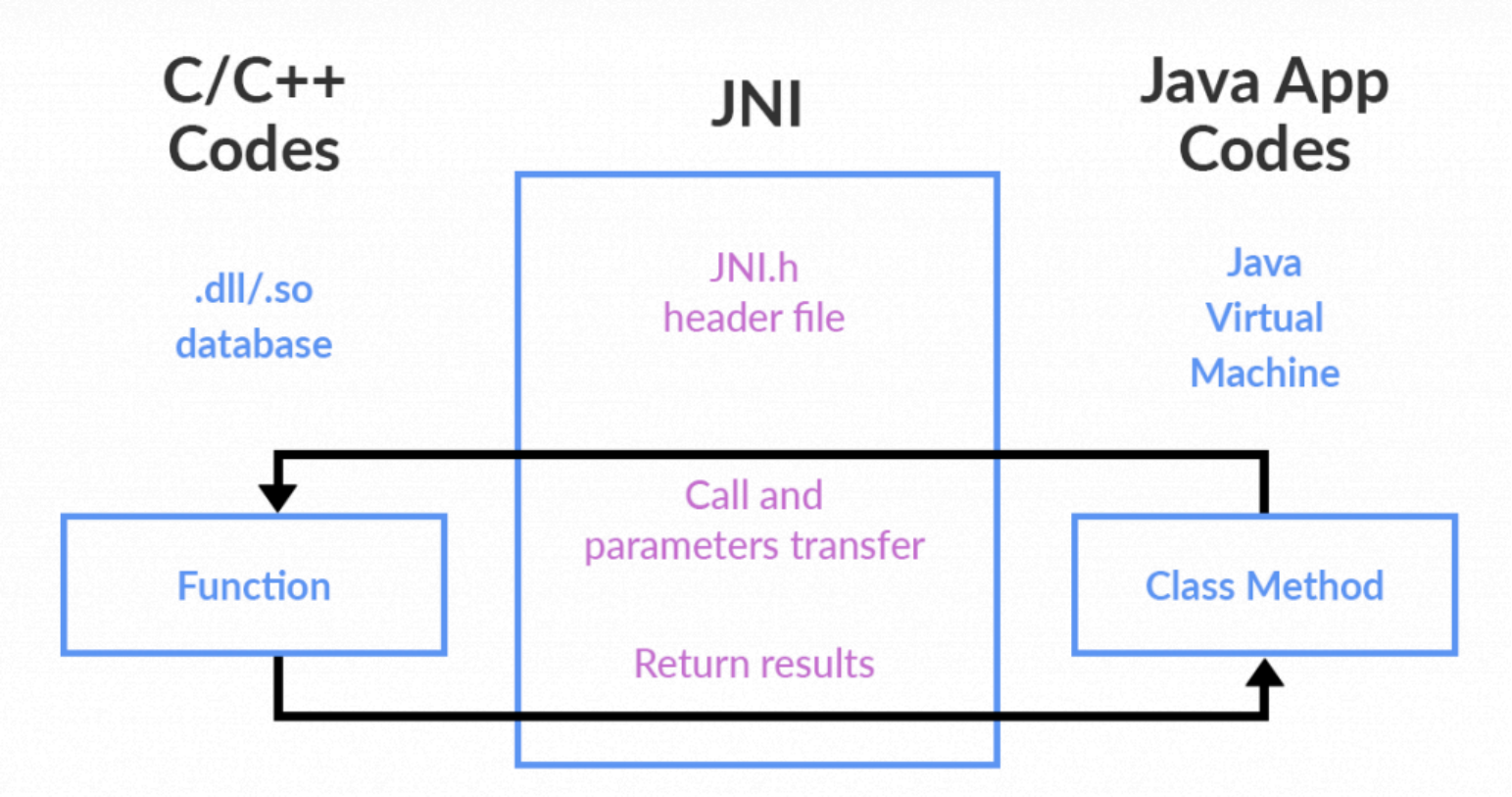
- The key mechanism that allows to mix java and native code is the **Foreign Function Invocation (FFI)** mechanism
- FFI is mechanism by which a program written in one programming language can call routines or make use of services written in another

- Java Native Interface (**JNI**)

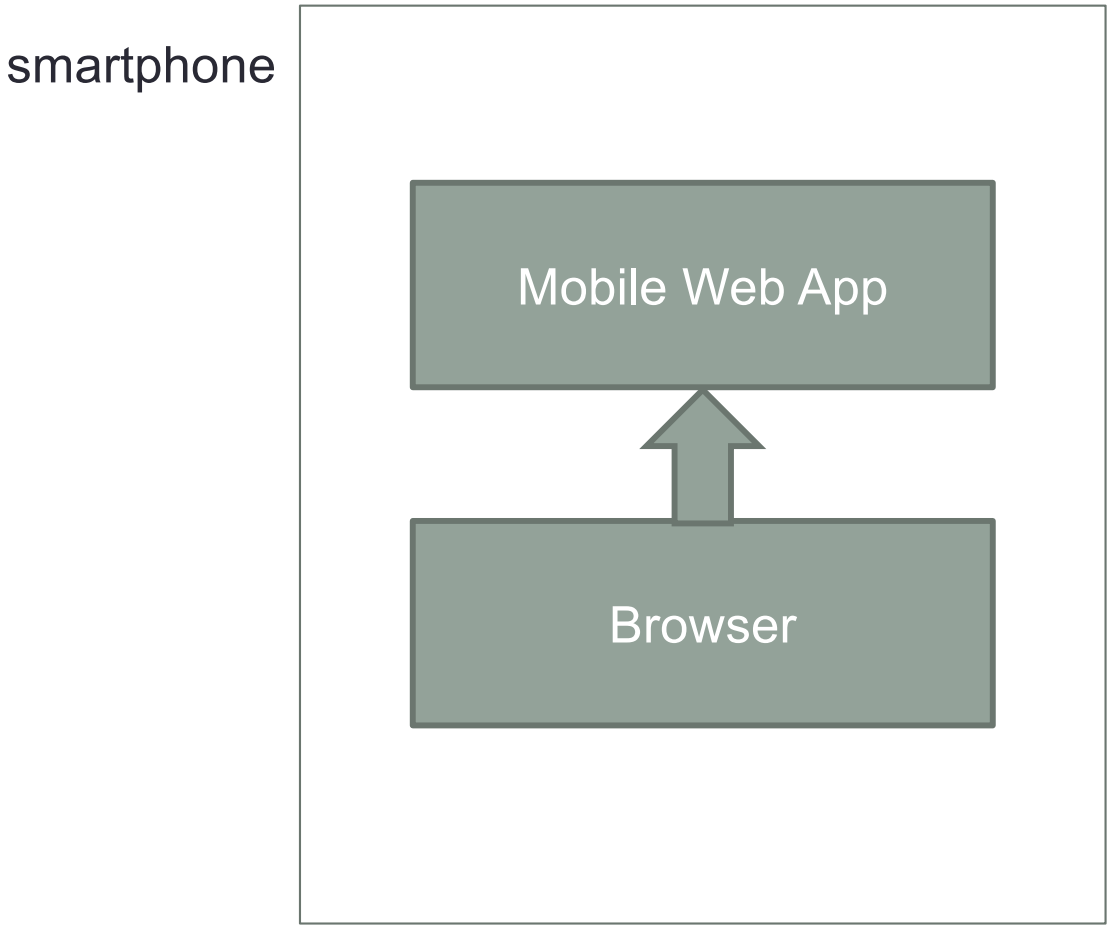
- **JavaScript-Java** Bridge

- But also: **ctypes** modules in Python ...

JNI (Java Native Interface)



A4. Mobile Web apps



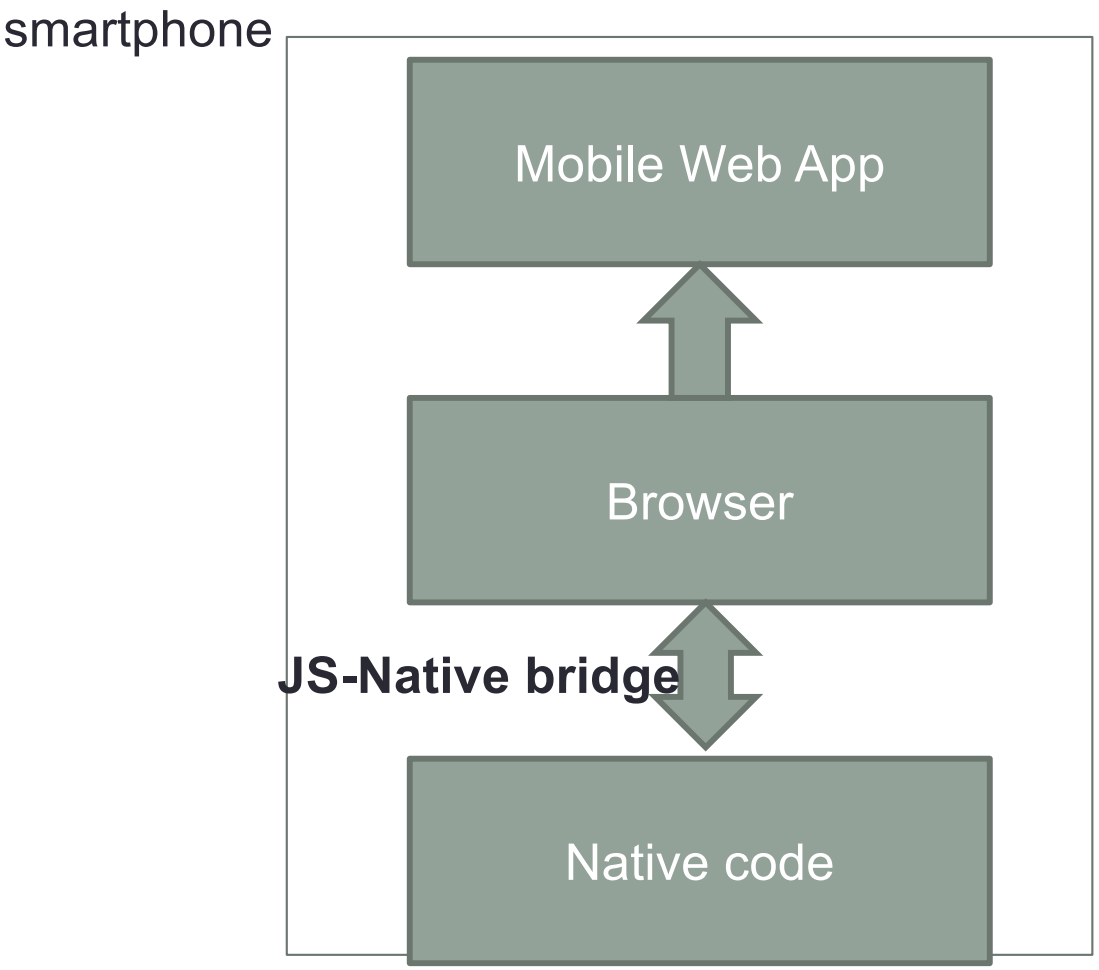
Installed apps

Concurrency and multithreading?

Access to all device features?

Look and feel?

A5. Hybrid Web apps



Installed apps

Concurrency and multithreading?

Access to all device features?

Look and feel?