A leading institution at the heart of the digital society

EURECOM
Sophia Antipolis

Analyse de la sécurité des Firmware de systèmes embarqués avancées et défis.

Aurélien Francillon

Colloque IMT "Gestion de crise et numérique"

Cti
Commission des Titres d'Ingénieur

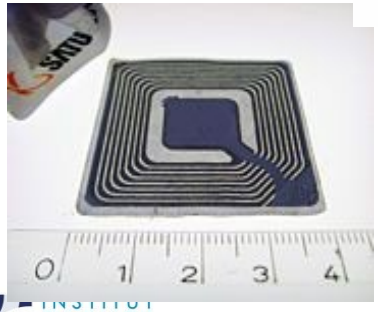
Embedded devices diversity

SmartCards

Connected devices

Sensors

RFID



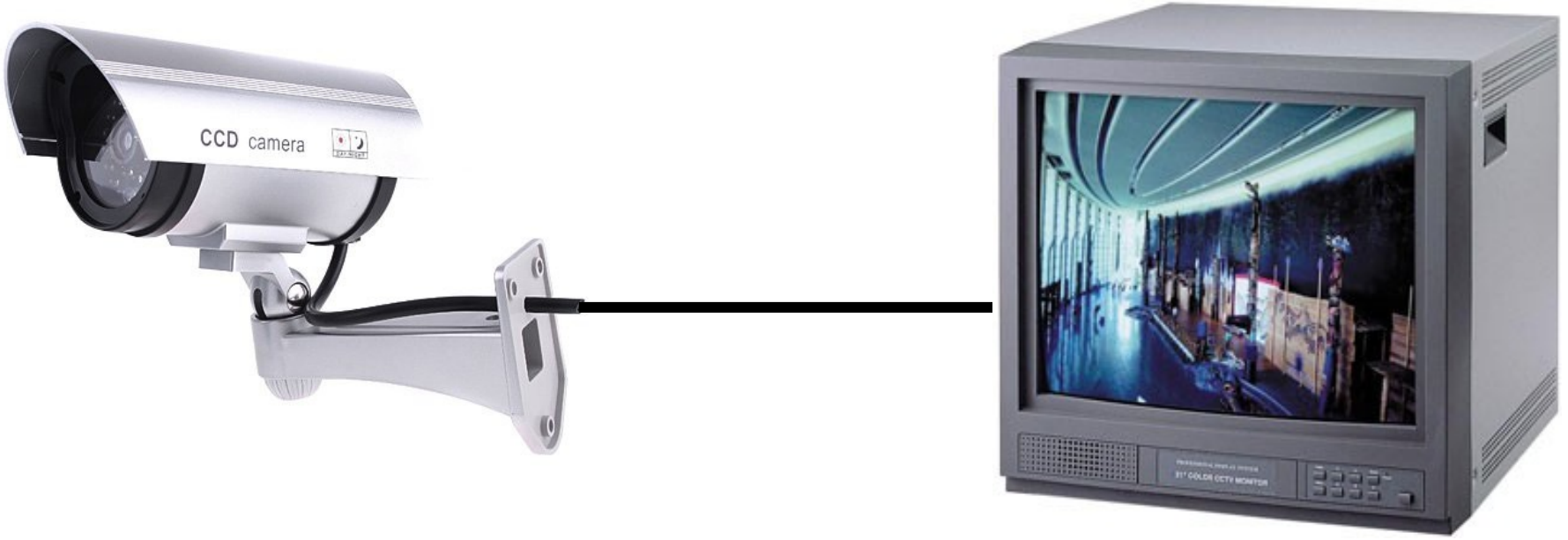
Industrial systems

INSTITUT CARNOT Télécom & Société numérique

EURECOM
Sophia Antipolis

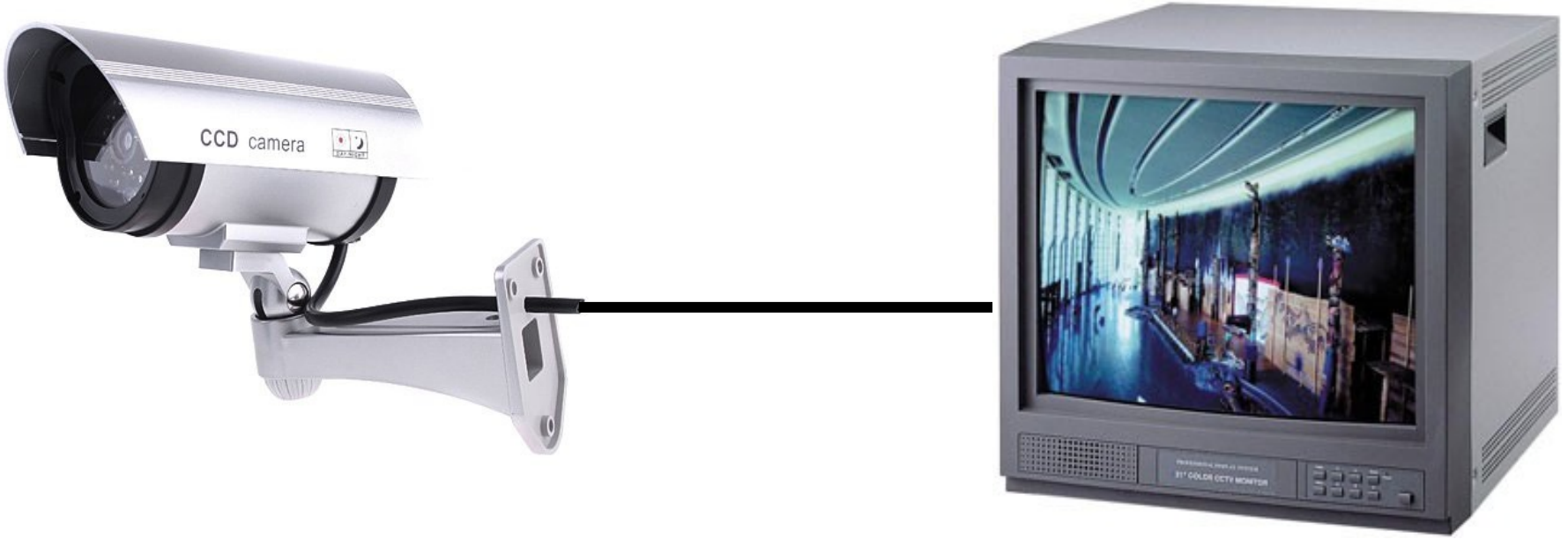
Before IoT

- Video Protection



Before IoT

- Video ~~Protection~~ Surveillance



IoT: composition

Composition Kills

Slide Courtesy of T. Goodspeed

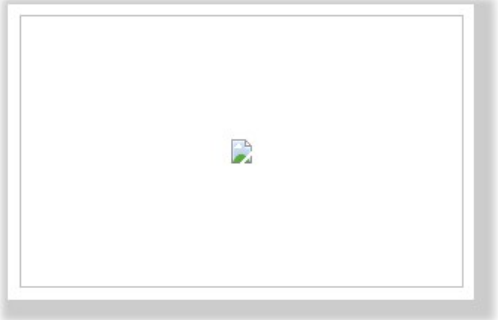
[redacted]

World CCTV cameras online | AvTech DVRs | Foscam cameras | Hikvision DVRs | Panasonic cameras | Linksys cameras | IPCamera can

- United States (9480)
- Korea, Republic Of (4730)
- China (2543)
- France (2440)
- Netherlands (2186)
- Italy (2159)
- Mexico (1893)
- United Kingdom (1768)
- Colombia (1755)
- India (1391)
- Indonesia (1190)
- Turkey (1044)
- Argentina (1007)
- Hong Kong (961)
- Japan (940)
- Canada (861)
- Brazil (841)
- Romania (759)
- Poland (757)
- Spain (755)
- Australia (726)

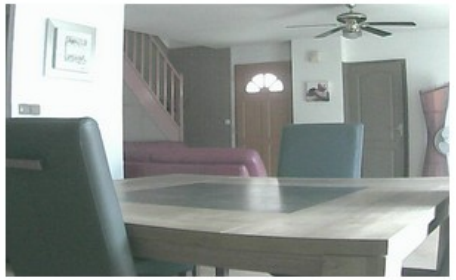
IP cameras: France

page 222 of 407.



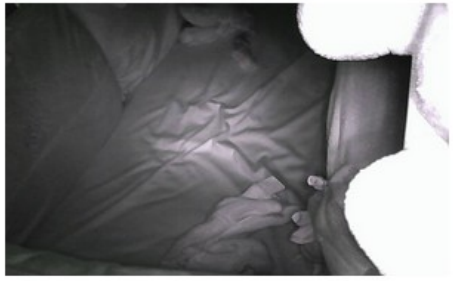
FR, NICE
2 ch.

FR, NICE
8 ch.



FR, NICE
1 ch.

FR, NICE
1 ch.



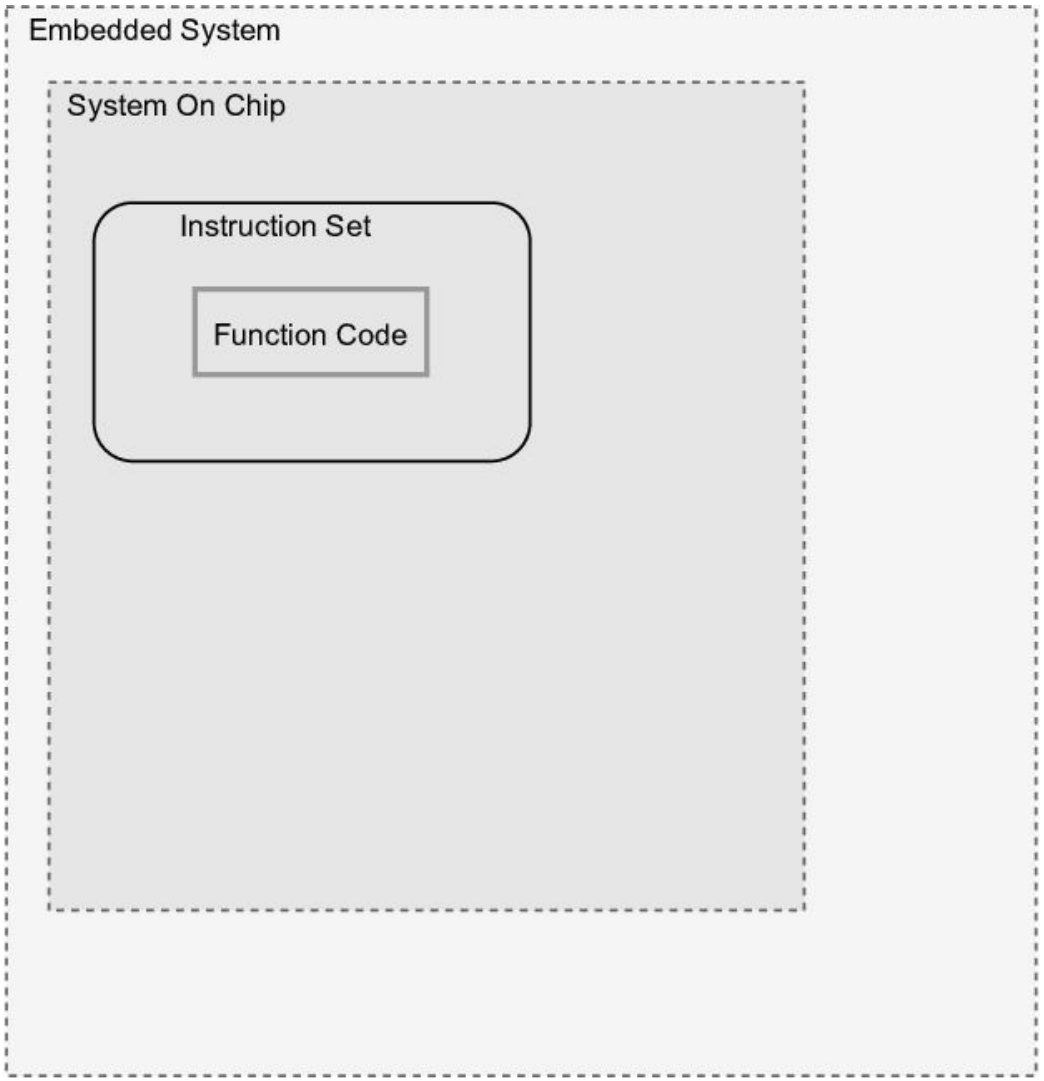
FR, NICE
1 ch.

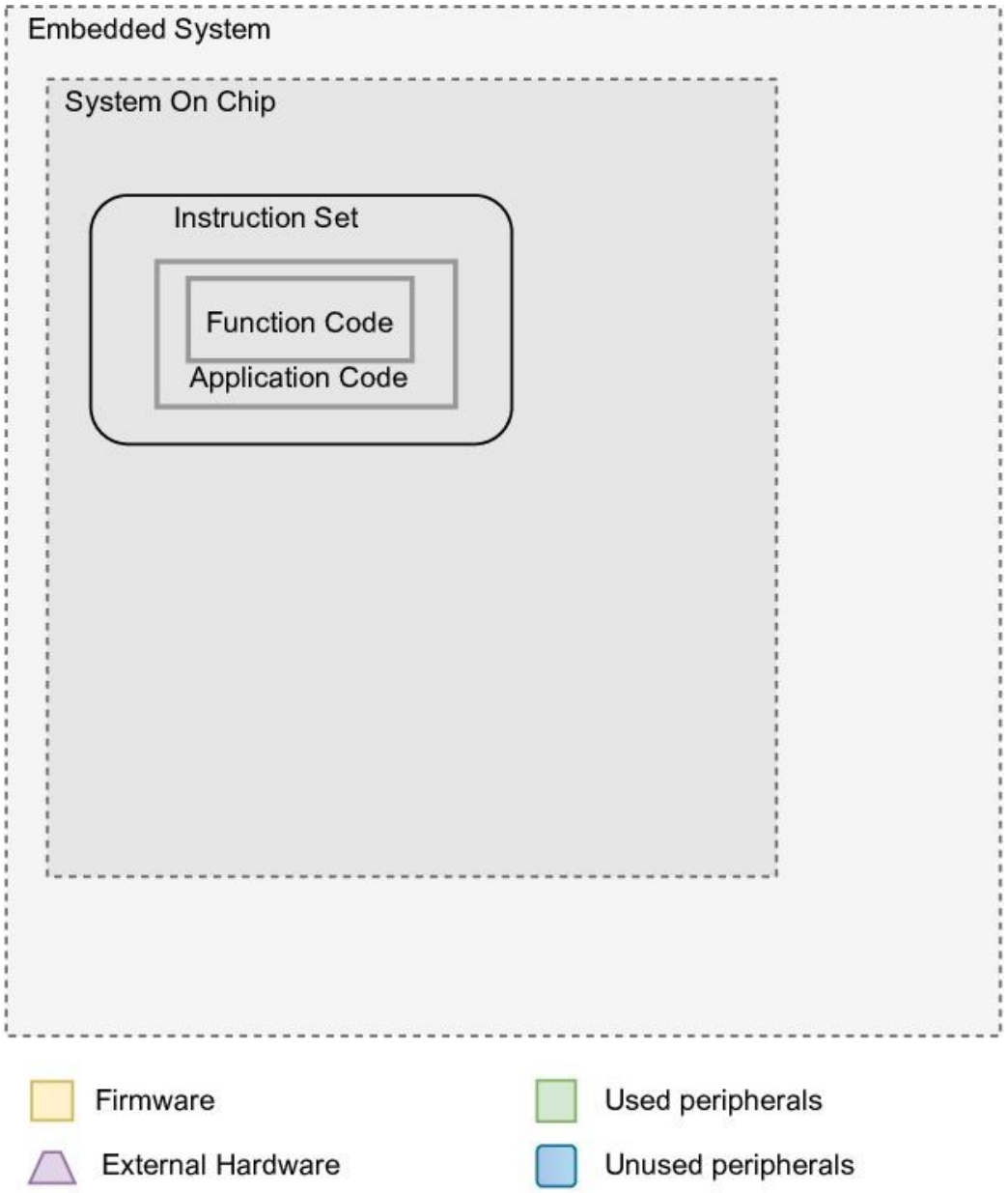
INSTITUT CARNOT Télécom & Société numérique

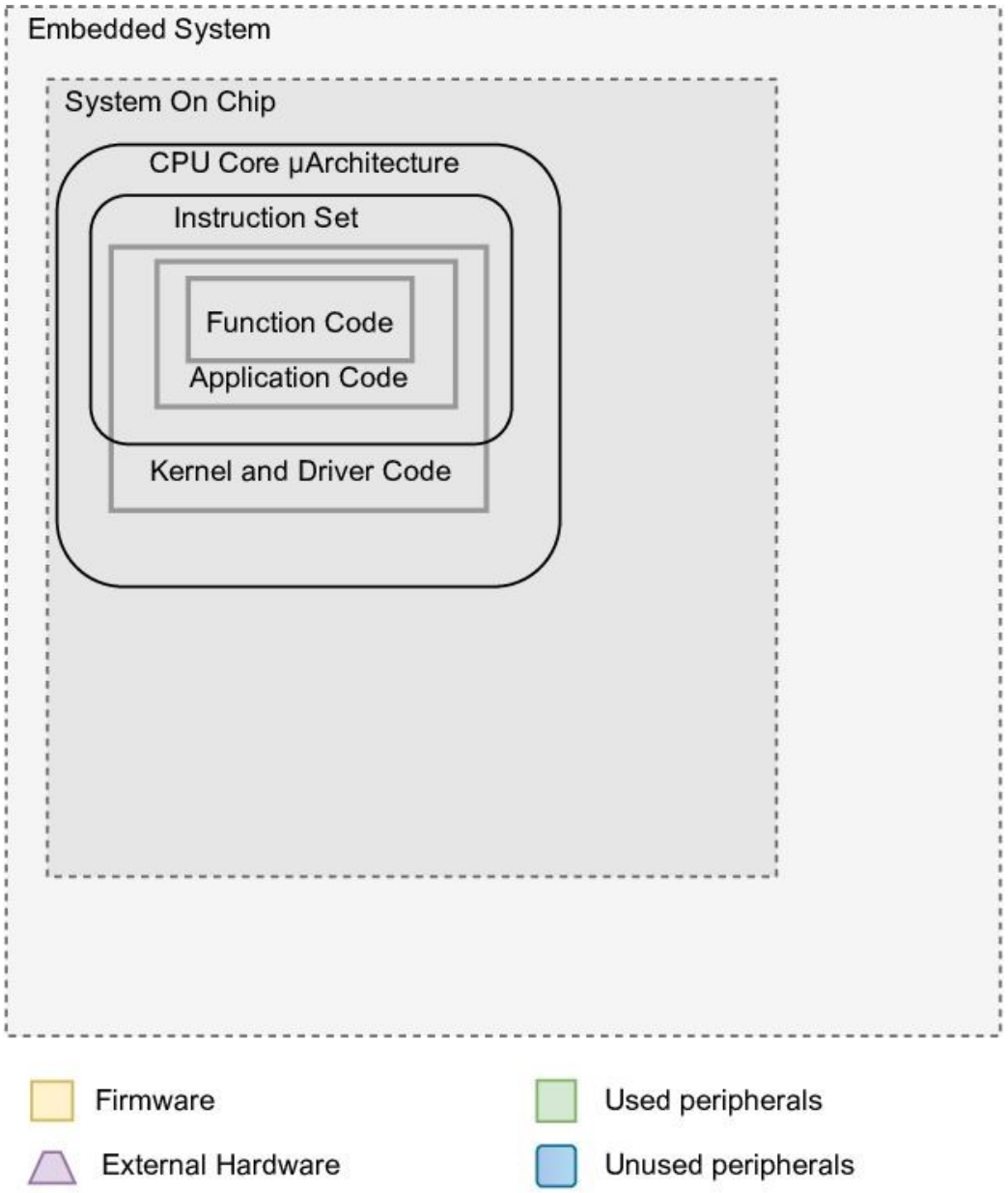
EURECOM Sophia Antipolis

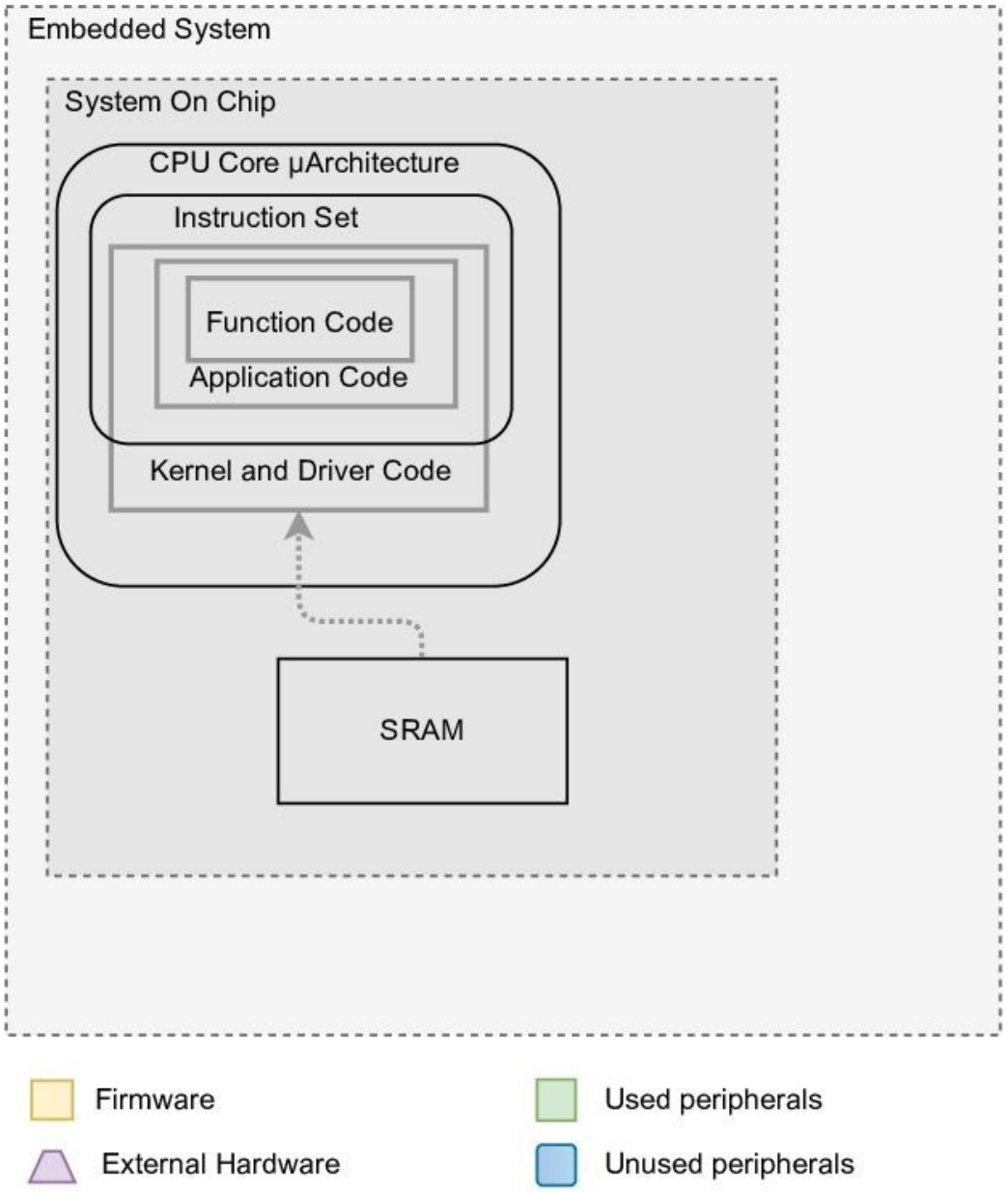
Dynamic Firmware analysis

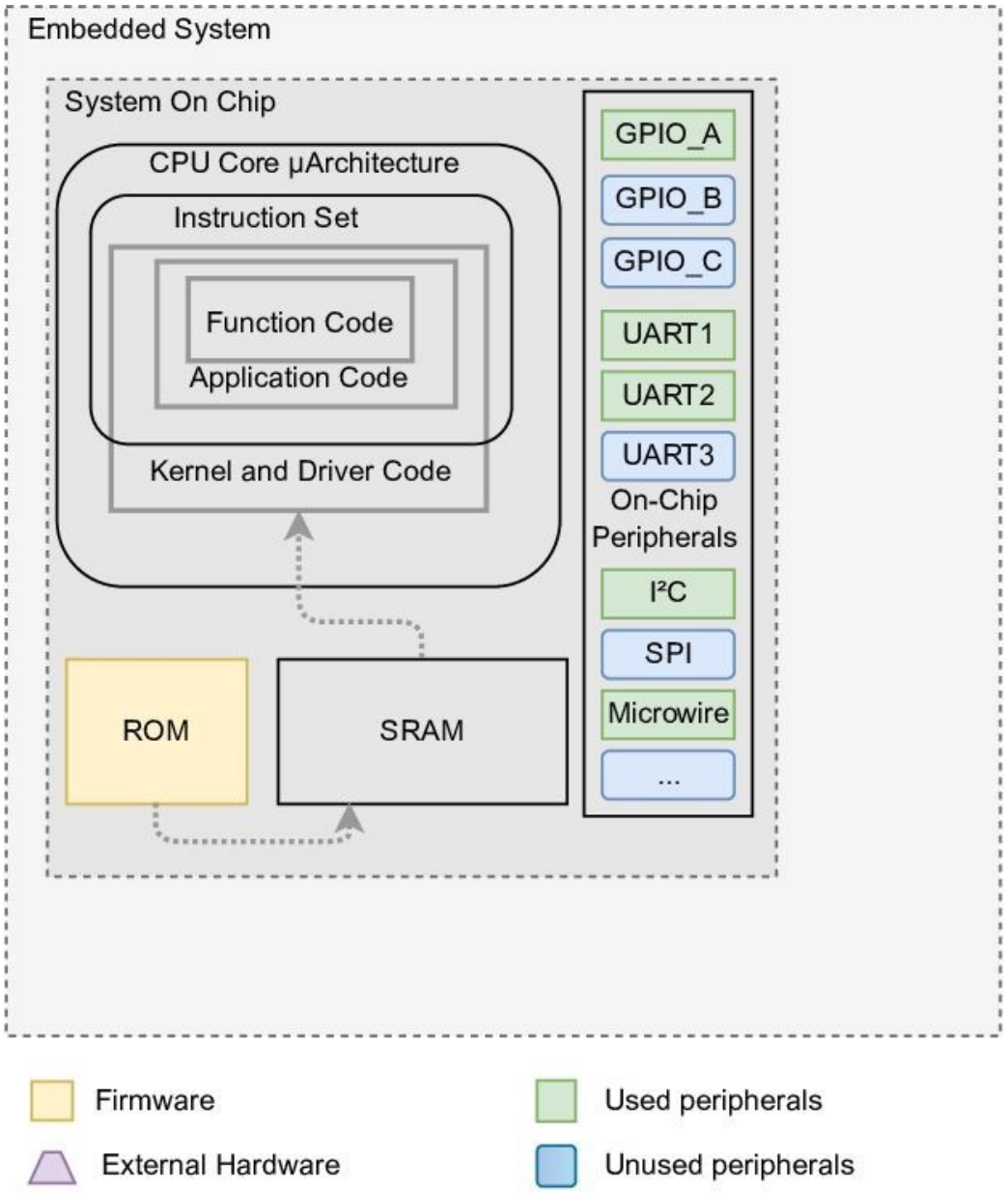
- Executing code on the device :
  - Limited visibility
  - Difficult to perform advanced analysis
- Emulators
  - Looks like the perfect solution
  - E.g., QEMU supports many architectures

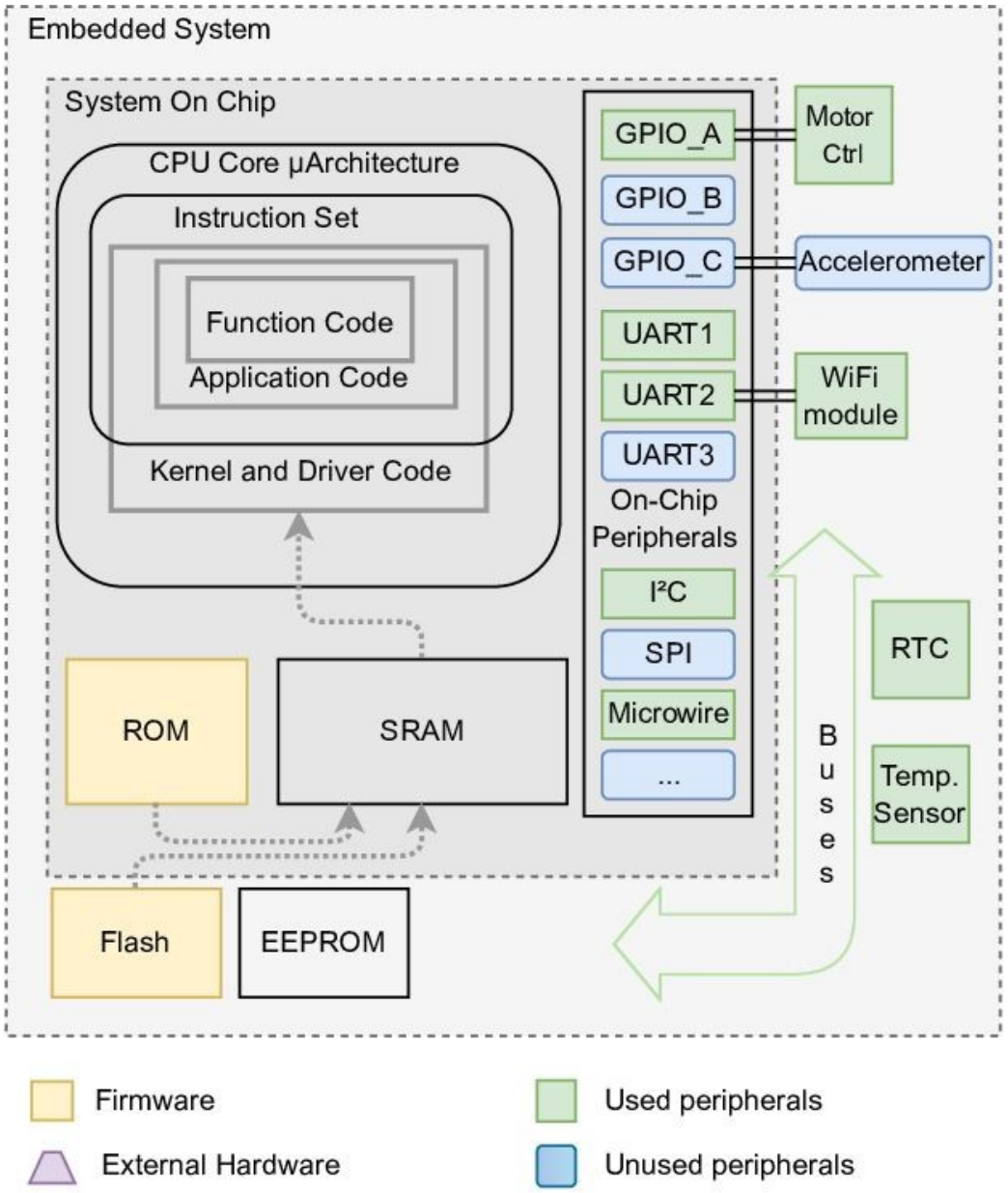

Embedded System
System On Chip
Instruction Set
Function Code

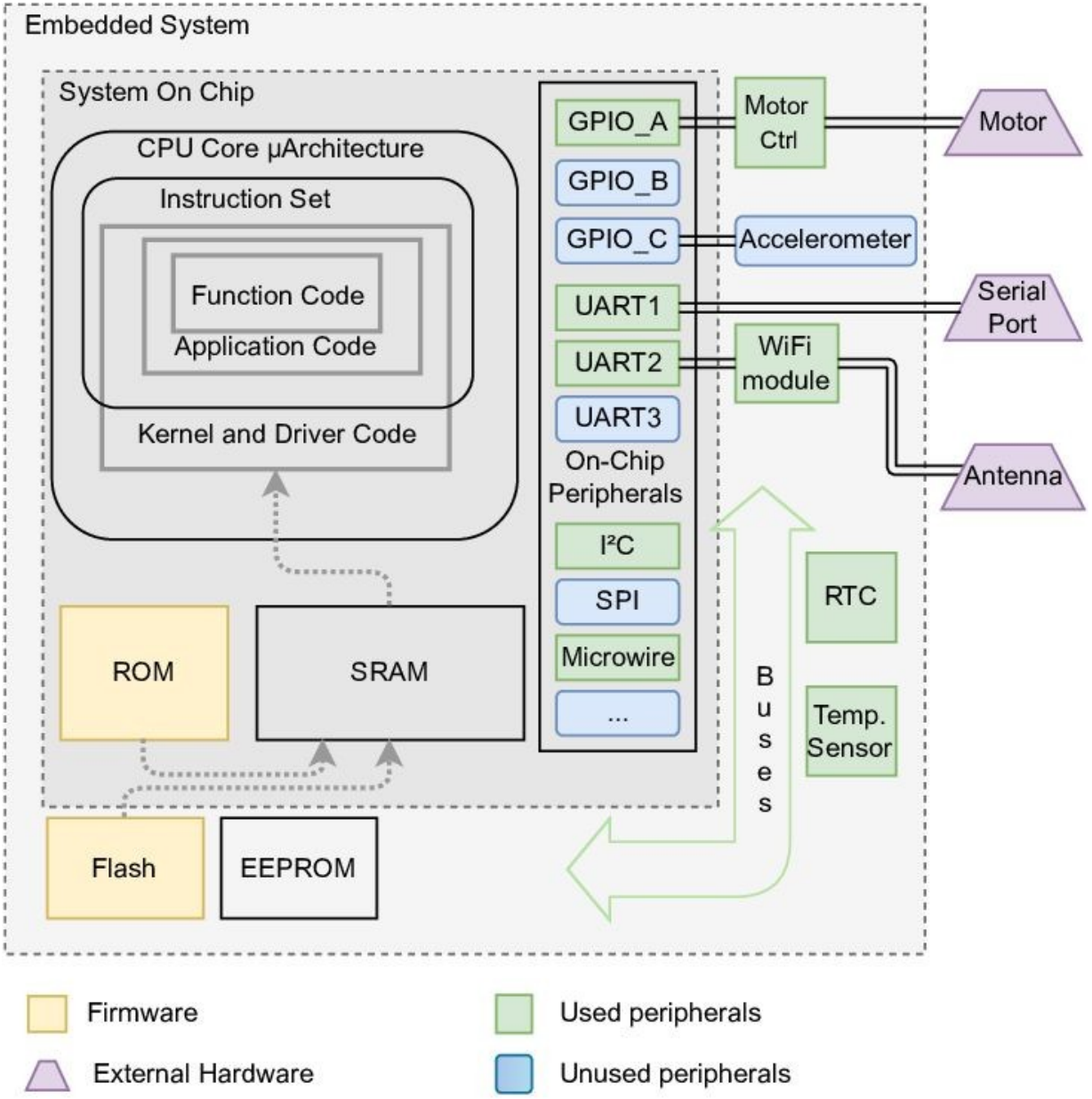

Embedded System
System On Chip
Instruction Set
Function Code
Application Code
Firmware
External Hardware
Used peripherals
Unused peripherals


Embedded System
System On Chip
CPU Core µArchitecture
Instruction Set
Function Code
Application Code
Kernel and Driver Code
Firmware
External Hardware
Used peripherals
Unused peripherals


Embedded System
System On Chip
CPU Core µArchitecture
Instruction Set
Function Code
Application Code
Kernel and Driver Code
SRAM
Firmware
External Hardware
Used peripherals
Unused peripherals


Embedded System
System On Chip
CPU Core µArchitecture
Instruction Set
Function Code
Application Code
Kernel and Driver Code
ROM
SRAM
GPIO_A
GPIO_B
GPIO_C
UART1
UART2
UART3
On-Chip Peripherals
I²C
SPI
Microwire
...
Firmware
External Hardware
Used peripherals
Unused peripherals


Embedded System
System On Chip
CPU Core µArchitecture
Instruction Set
Function Code
Application Code
Kernel and Driver Code
ROM
SRAM
Flash
EEPROM
GPIO_A
GPIO_B
GPIO_C
UART1
UART2
UART3
On-Chip Peripherals
I²C
SPI
Microwire
...
Buses
Motor Ctrl
Accelerometer
WiFi module
RTC
Temp. Sensor
Firmware
External Hardware
Used peripherals
Unused peripherals


Embedded System
System On Chip
CPU Core µArchitecture
Instruction Set
Function Code
Application Code
Kernel and Driver Code
ROM
SRAM
Flash
EEPROM
GPIO_A
GPIO_B
GPIO_C
UART1
UART2
UART3
On-Chip Peripherals
I²C
SPI
Microwire
...
Buses
Motor Ctrl
Motor
Accelerometer
Serial Port
WiFi module
Antenna
RTC
Temp. Sensor
Firmware
External Hardware
Used peripherals
Unused peripherals

Too many peripherals to emulate

(a) Type-1 Linux Systems (DTB corpus)

Arch	|SoC|	Unique P	$\mu \pm \sigma$ P/SoC	$\tilde{x}$ P/SoC
ARM	1,310	6,858	58 ± 26	55
ARM64	430	3,653	58 ± 24	59
MIPS	20	270	21 ± 11	16
PPC	196	1,422	31 ± 19	27

(b) Type-2 and Type-3 ARM Cortex Systems (SVD corpus)

Vendor	|SoC|	Unique P	$\mu \pm \sigma$ P/SoC	$\tilde{x}$ P/SoC
Atmel	147	416	34 ± 10	30
Freescale	133	561	49 ± 13	47
Fujitsu	100	237	44 ± 9	41
NXP	24	374	28 ± 18	21
STMicro	72	852	59 ± 22	58
SiliconLabs	10	62	40 ± 2	40
Spansion	88	193	44 ± 9	42
TI	52	95	27 ± 4	26

SoK: Enabling Security Analyses of Embedded Systems via Rehosting

A. Fasano, T. Ballo, M. Muench, T. Leek, A. Olienik, B. Dolan Gavitt, M. Egele, A. Francillon, L. Lu, N. Gregory, D. Balzatotti, W. Robertson

ACM ASIACCS 2021

Emulators exist but are not sufficient

- Virtual machines are used extensively
  - Usually emulate limited set of basic peripherals
  - Desktop/server operating systems will load the right drivers
- Embedded devices are often very custom
  - Very specific peripherals
  - Firmware will only include the code for the right peripherals
    - won’t execute properly without the right peripheral interactions

Devices access

- Device and debug available: more control over the software execution
- Or completely black-box
- Device available but no debug/firmware
- Development device or commercial

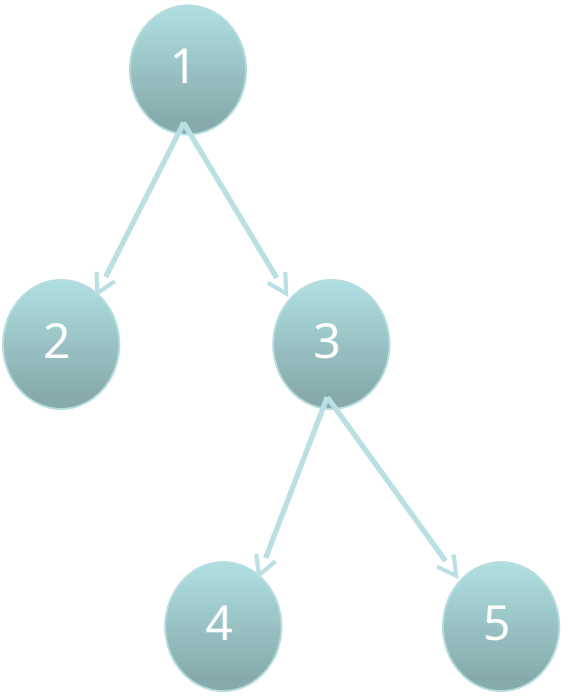
Approaches are as diverse as the devices

- How generic is the code to test ?
  - Web interface, Linux application or bare metal application
- Which analysis methods to use
  - Dynamic: Fuzzing, Symbolic execution ?
  - Static: Simple or advanced binary analysis?

Dynamic analysis techniques for security evaluation

Techniques that are typically used on a PC

- Advanced debugging techniques
  - Tracing
  - Fuzzing
  - Tainting
  - Symbolic Execution



Dynamic analysis techniques for security evaluation

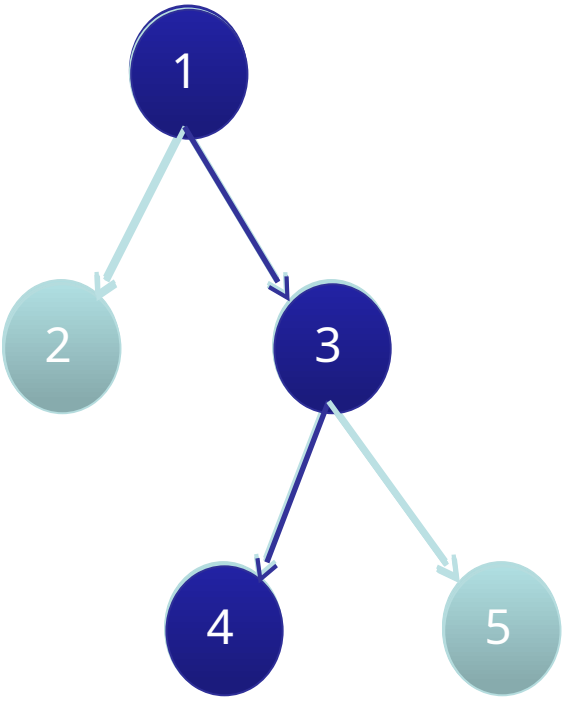
Techniques that are typically used on a PC

- Advanced debugging techniques
  - Tracing
  - Fuzzing
  - Tainting
  - Symbolic Execution

Collecting an execution trace

Trace

1: x = 4
3: x = 4
4: x = 6



Dynamic analysis techniques for security evaluation

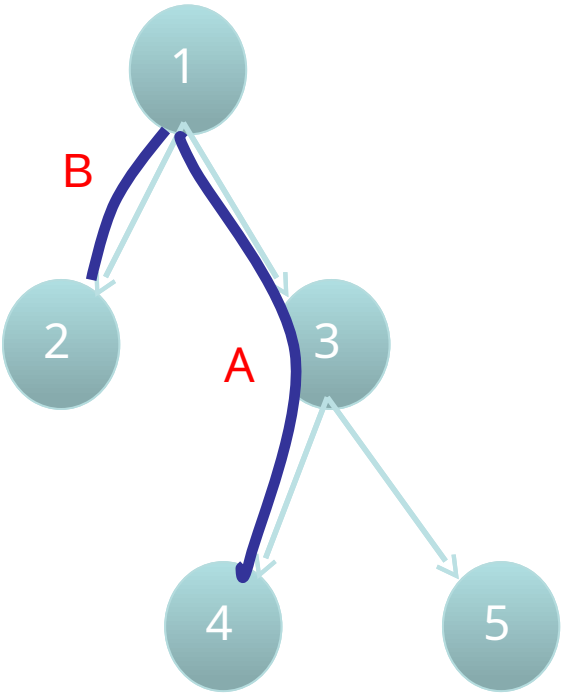
Techniques that are typically used on a PC

- Advanced debugging techniques
  - Tracing
  - Fuzzing
  - Tainting
  - Symbolic Execution

Testing with random input

Testcase A
x = 4

Testcase B
x = 2

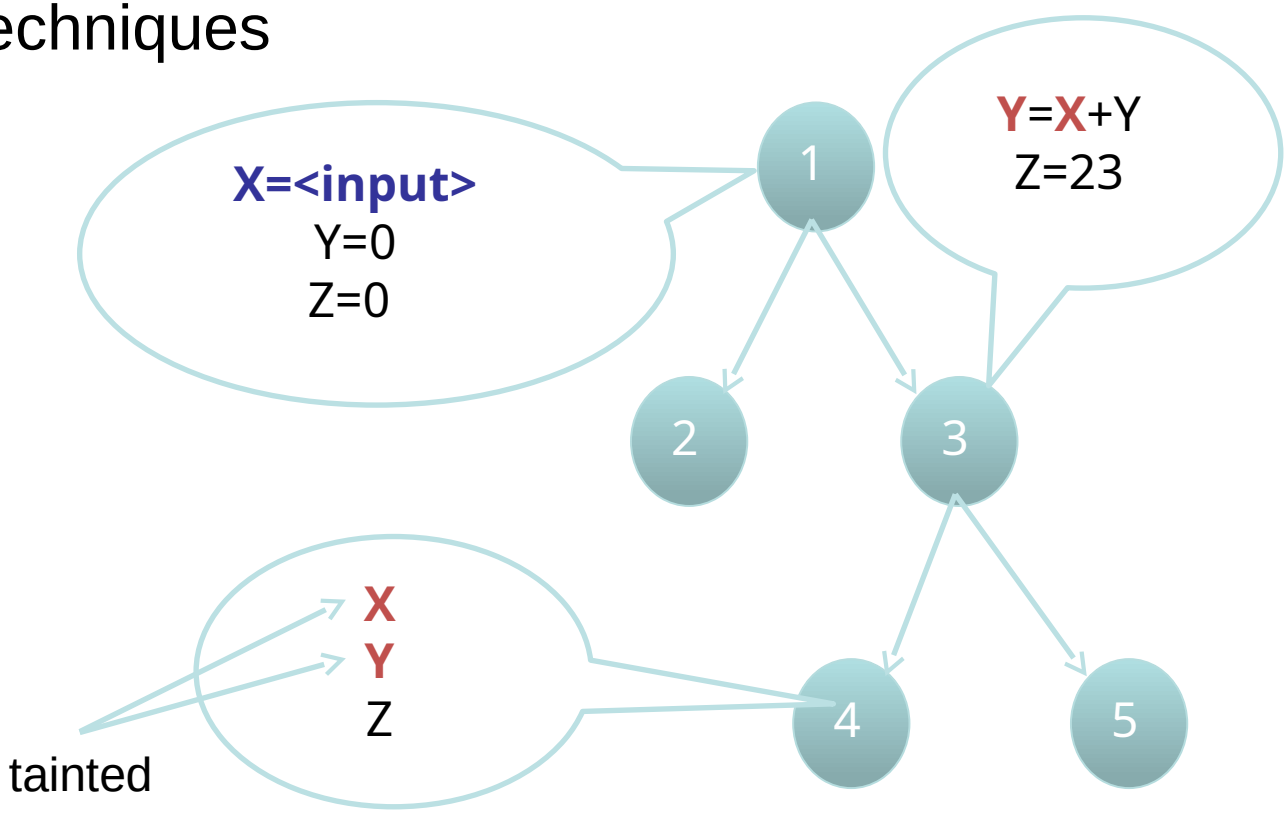


Dynamic analysis techniques for security evaluation

Techniques that are typically used on a PC

- Advanced debugging techniques
  - Tracing
  - Fuzzing
  - Tainting
  - Symbolic Execution

Data flow tracking

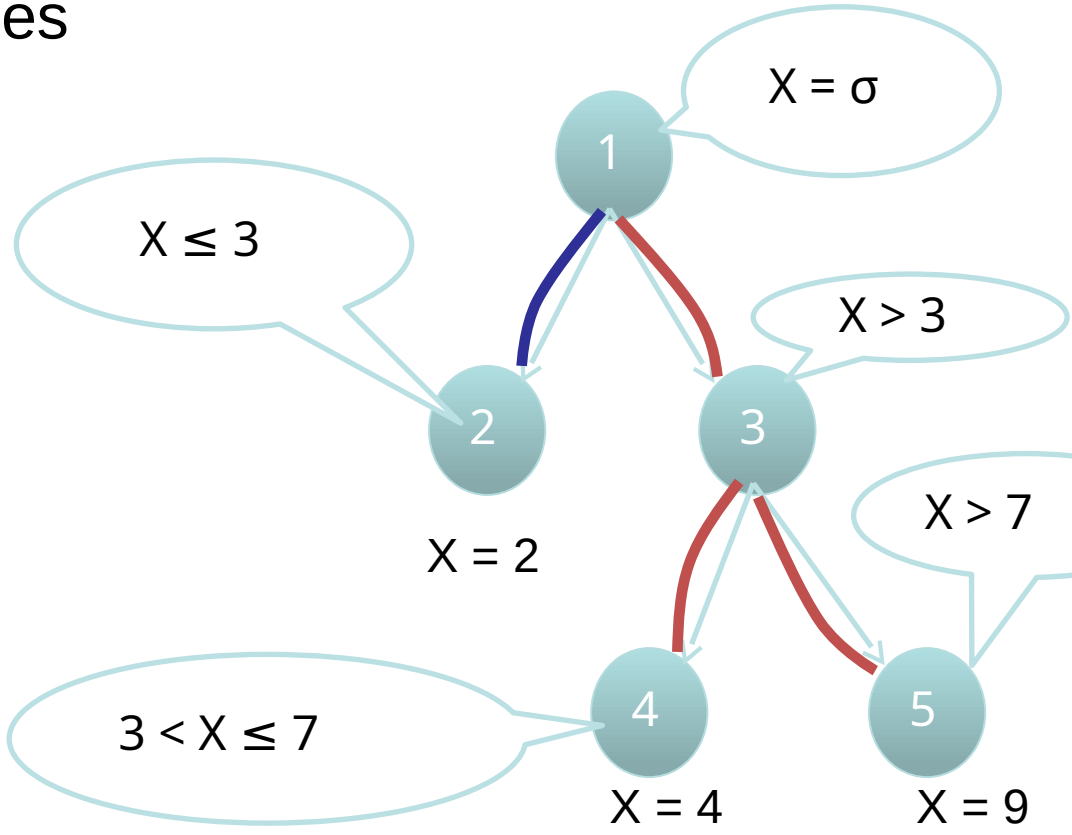


Dynamic analysis techniques for security evaluation

Techniques that are typically used on a PC

- Advanced debugging techniques
  - Tracing
  - Fuzzing
  - Tainting
  - Symbolic Execution

Multipath exploration

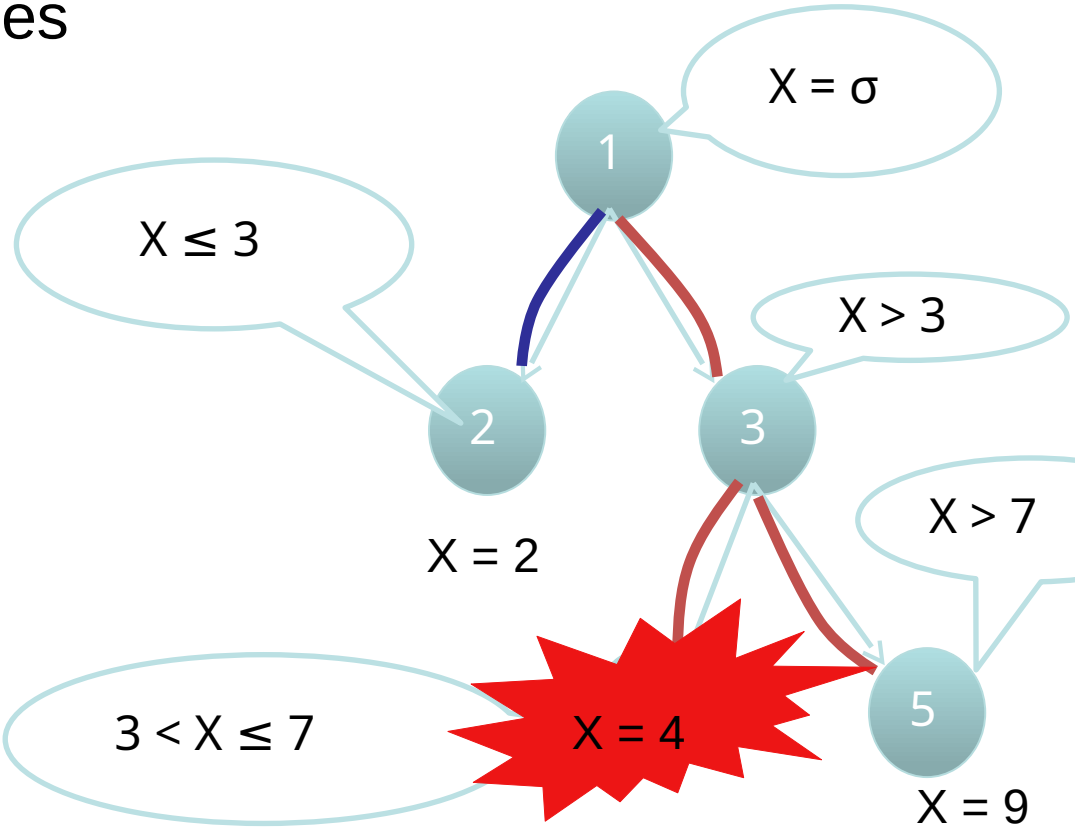


Dynamic analysis techniques for security evaluation

Techniques that are typically used on a PC

- Advanced debugging techniques
  - Tracing
  - Fuzzing
  - Tainting
  - Symbolic Execution

Multipath exploration



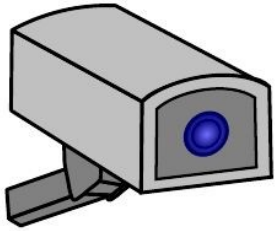
3 Categories of devices

Type-I:

General purpose OS-based

Type-II:

Embedded OS-based



Type-III:

No OS-Abstraction

Measuring effect of device type

	Platform			
	Desktop	Type-I	Type-II	Type-III
Format String	✓	✓	✗	✗
Stack-based buffer overflow	✓	✓	✓ (opaque)	! (hang)
Heap-based buffer overflow	✓	! (late crash)	✗	✗
Double Free	✓	✓	✗	✗ (malfunc.)
Null Pointer Dereference	✓	✓	✓ (reboot)	✗ (malfunc.)

What You Corrupt Is Not What You Crash: Challenges in Fuzzing Embedded Devices
Marius Muench, Jan Stijohann, Frank Kargl, Aurélien Francillon, Davide Balzarotti
NDSS 2018

Rehosting for Dynamic Firmware Analysis

- In general emulation of firmware is difficult:
  - Manual
  - Imperfect
  - Leads to incorrect executions
- Can we automate this?

Rehosting for Dynamic Firmware Analysis

- In general emulation of firmware is difficult: Manual, Imperfect
- Automate it?

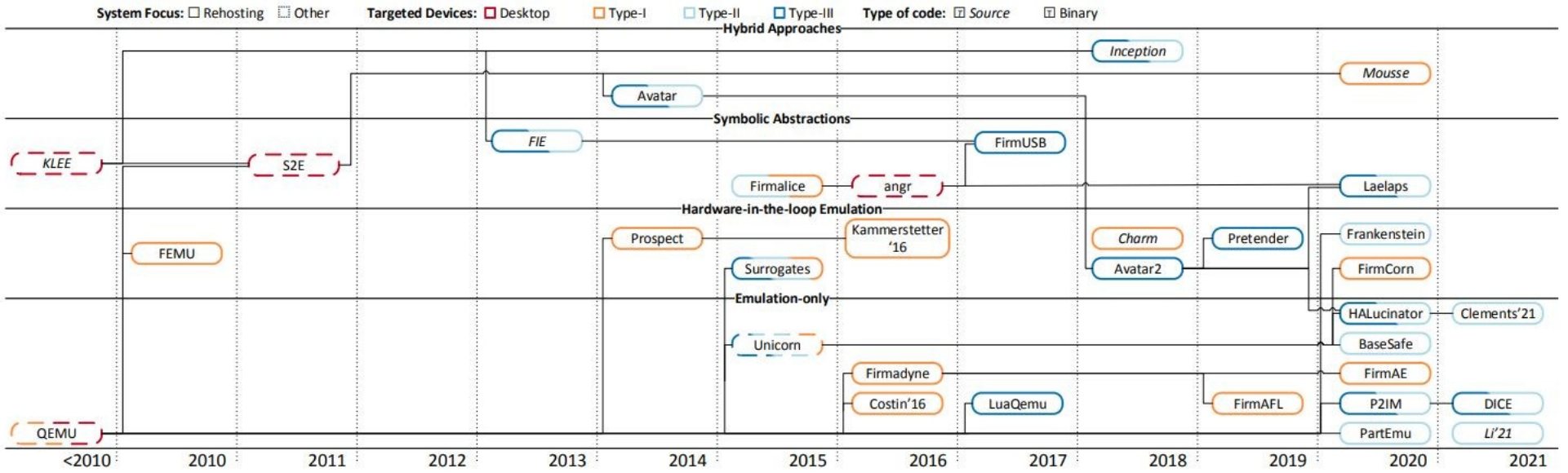


Figure 5: Timeline of Rehosting Systems.

SoK: Enabling Security Analyses of Embedded Systems via Rehosting

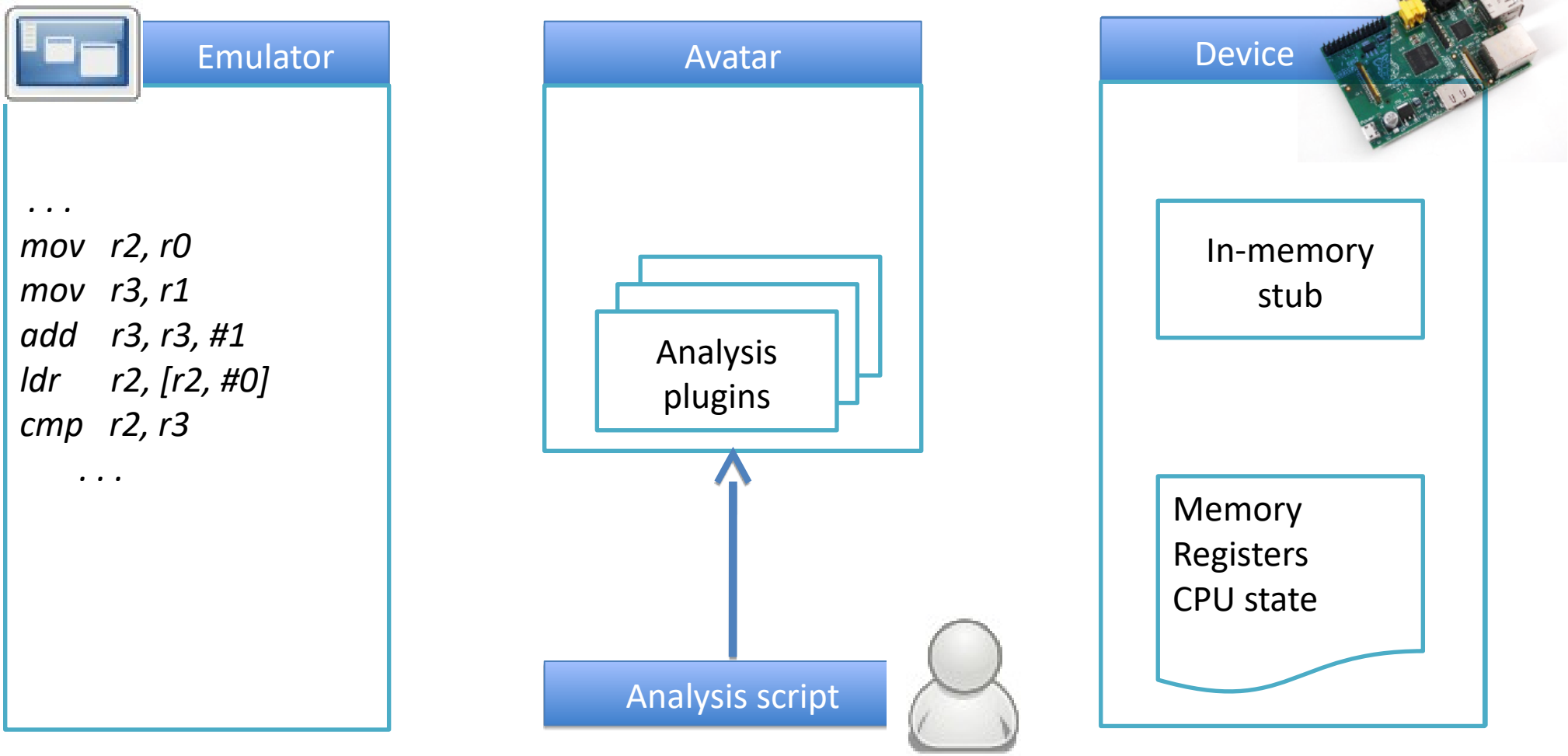
A. Fasano, T. Ballo, M. Muench, T. Leek, A. Olienik, B. Dolan Gavitt, M. Egele, A. Francillon, L. Lu, N. Gregory, D. Balzatotti, W. Robertson

ACM ASIACCS 2021

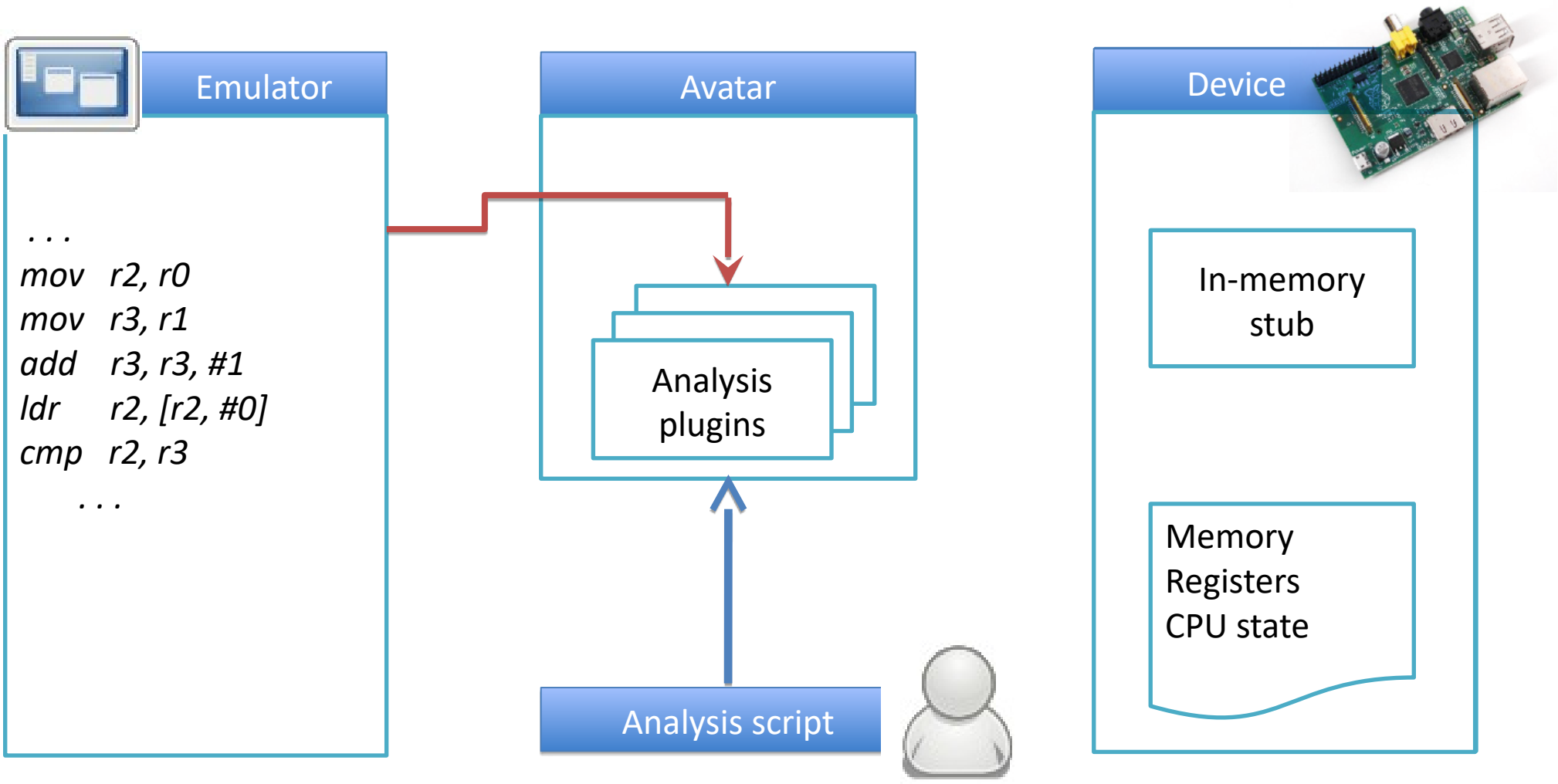
Avatar²

- Our hardware-in-the-loop rehosting approach
- When we have the firmware and the device with debug access

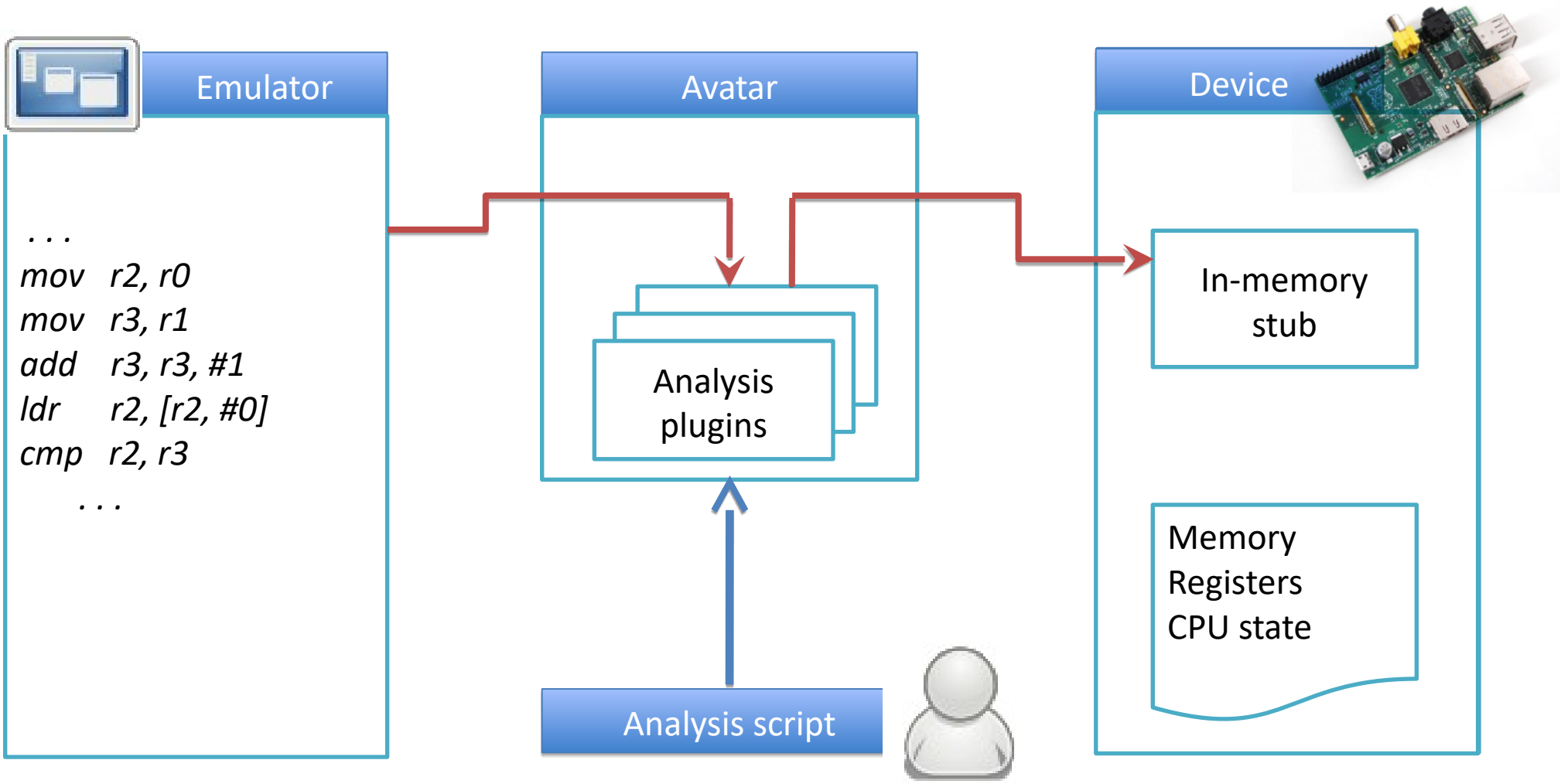
Avatar² overview



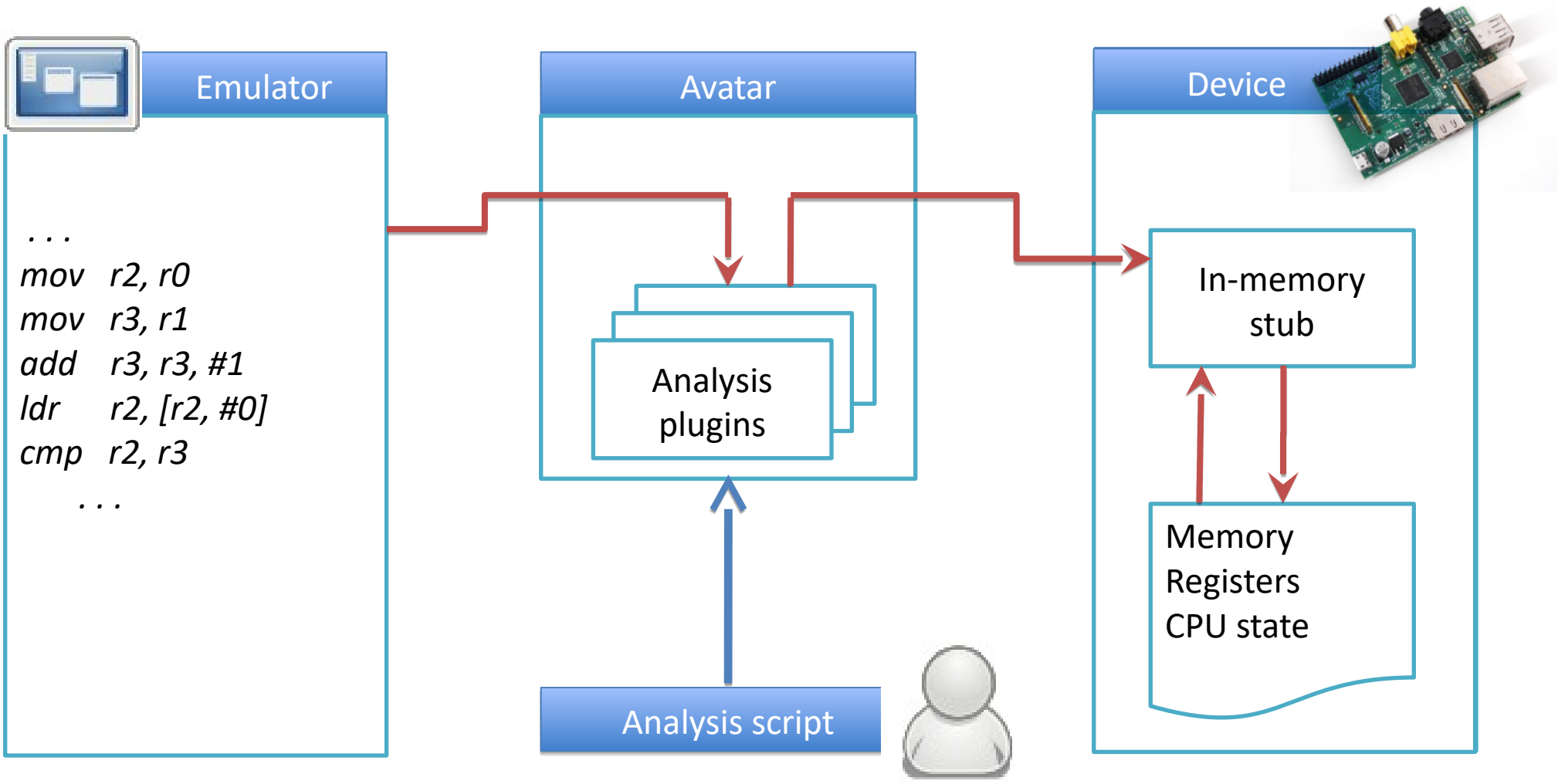
Avatar² overview



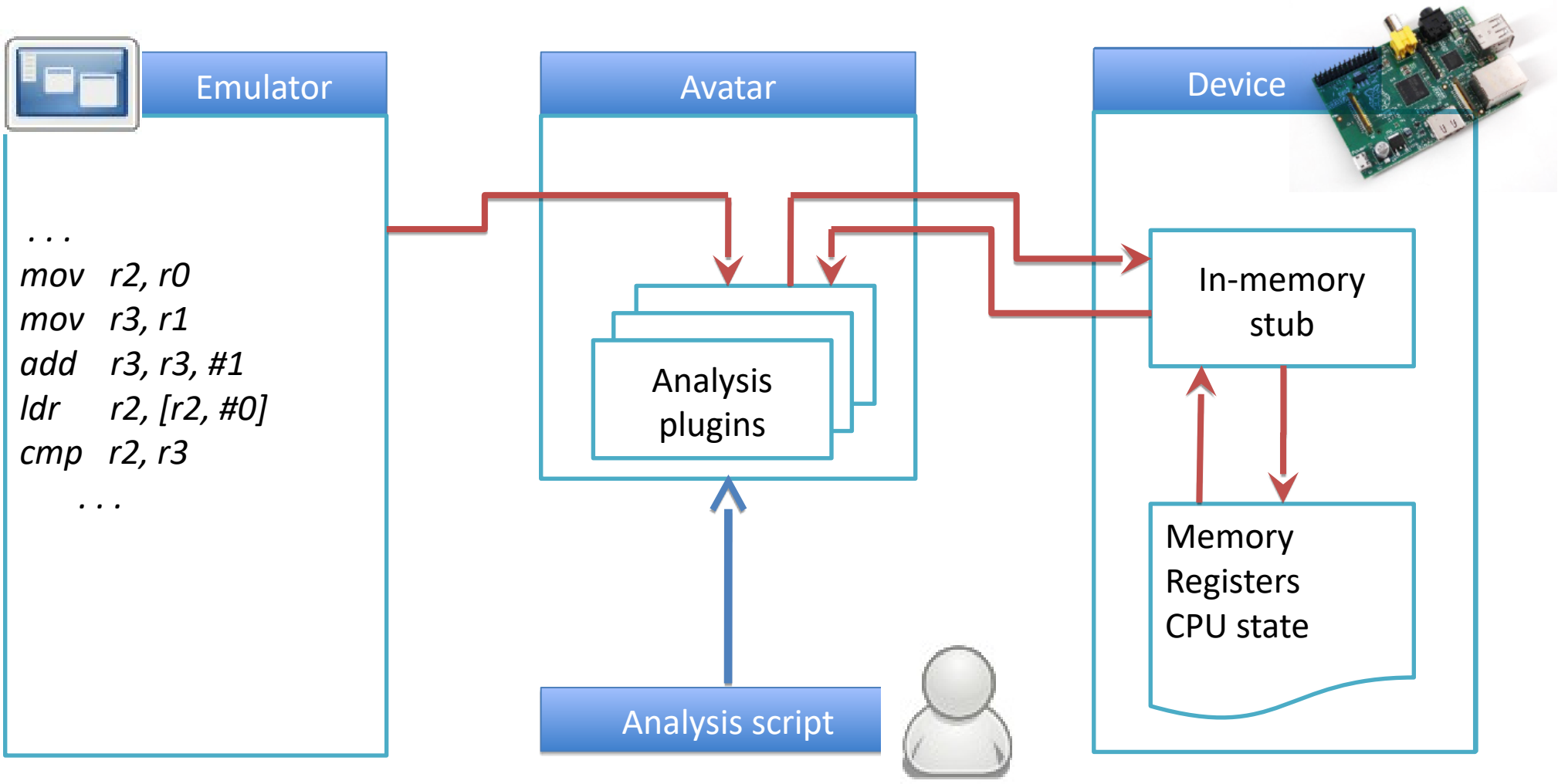
Avatar² overview



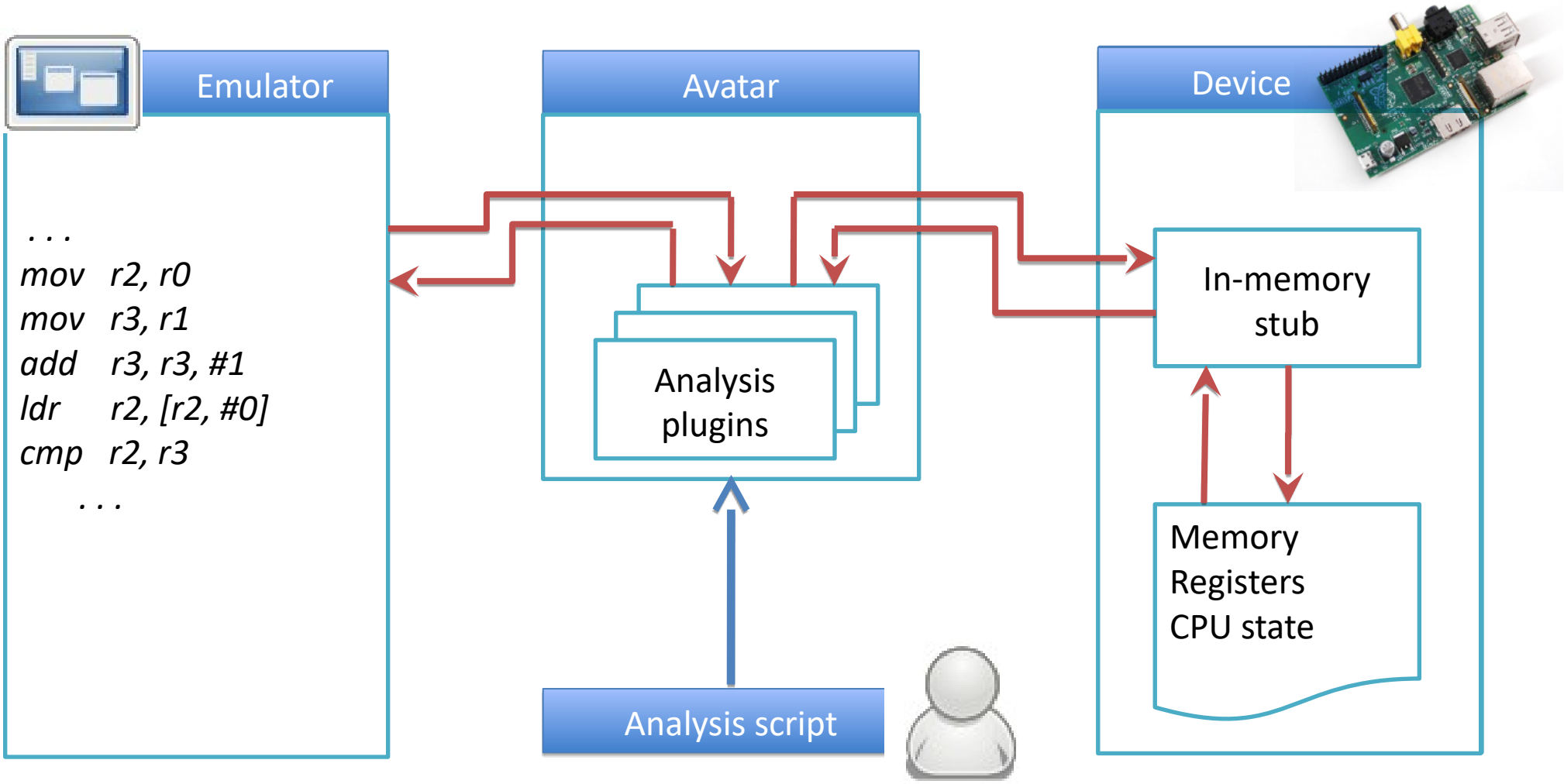
Avatar² overview



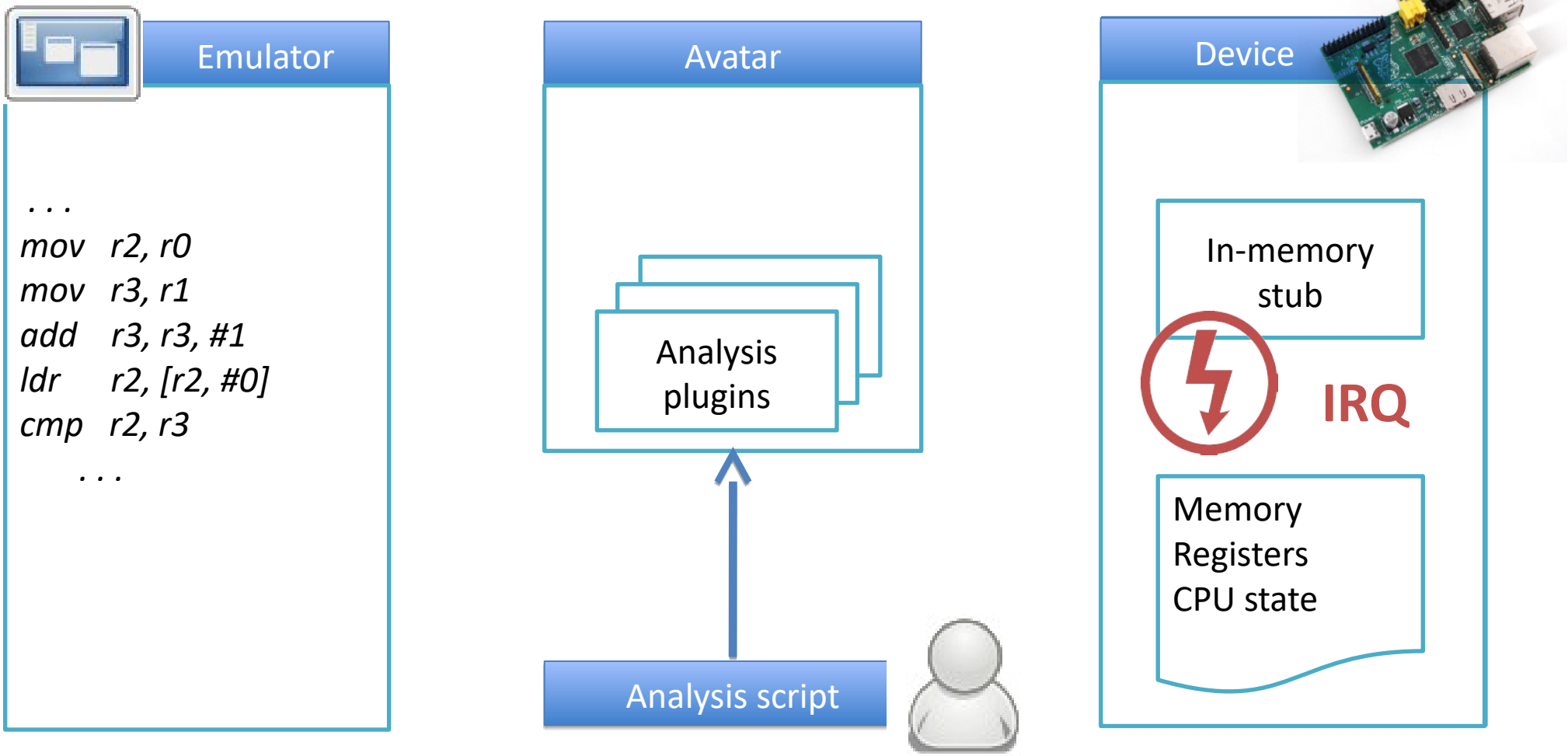
Avatar² overview



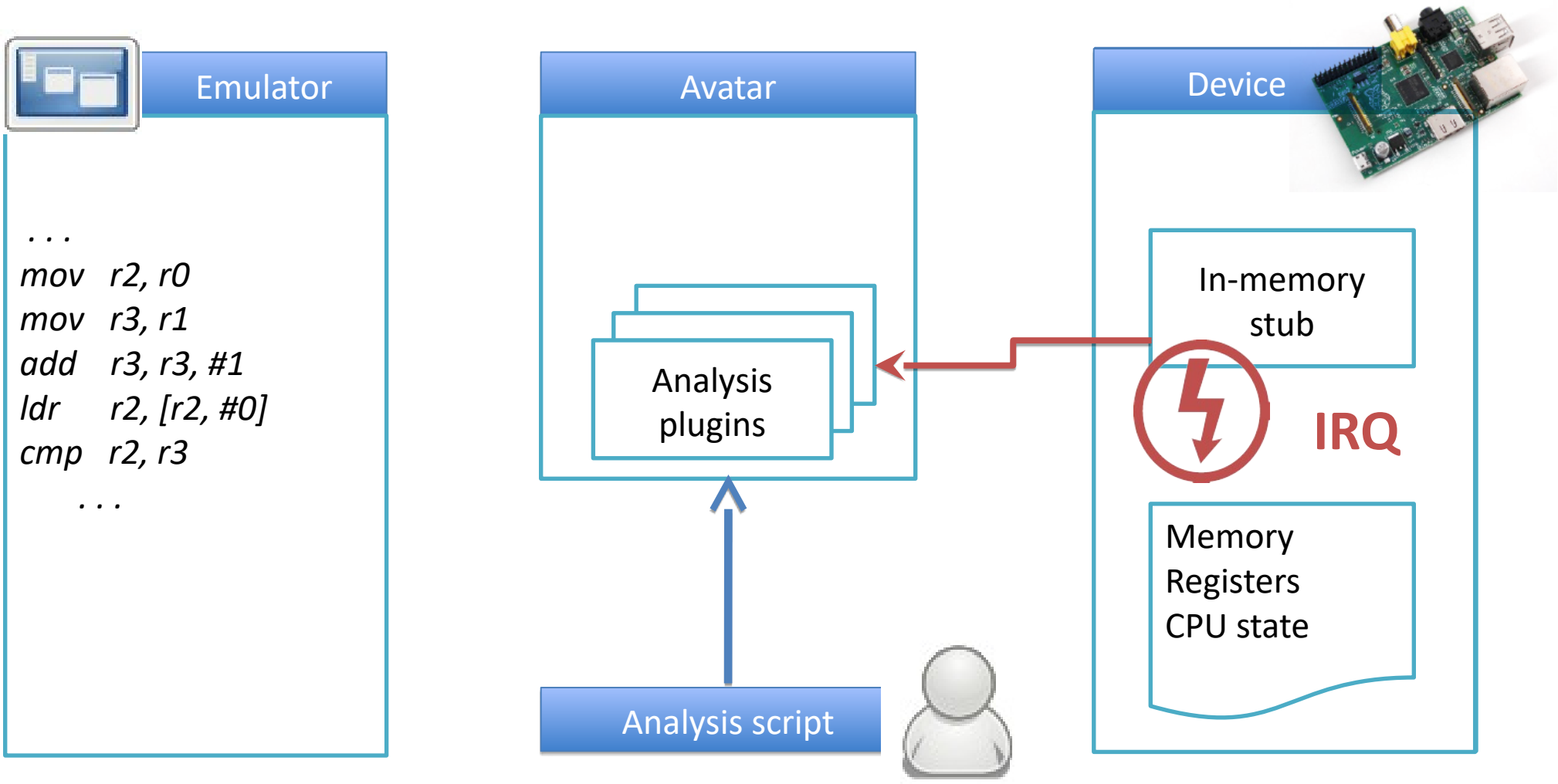
Avatar² overview



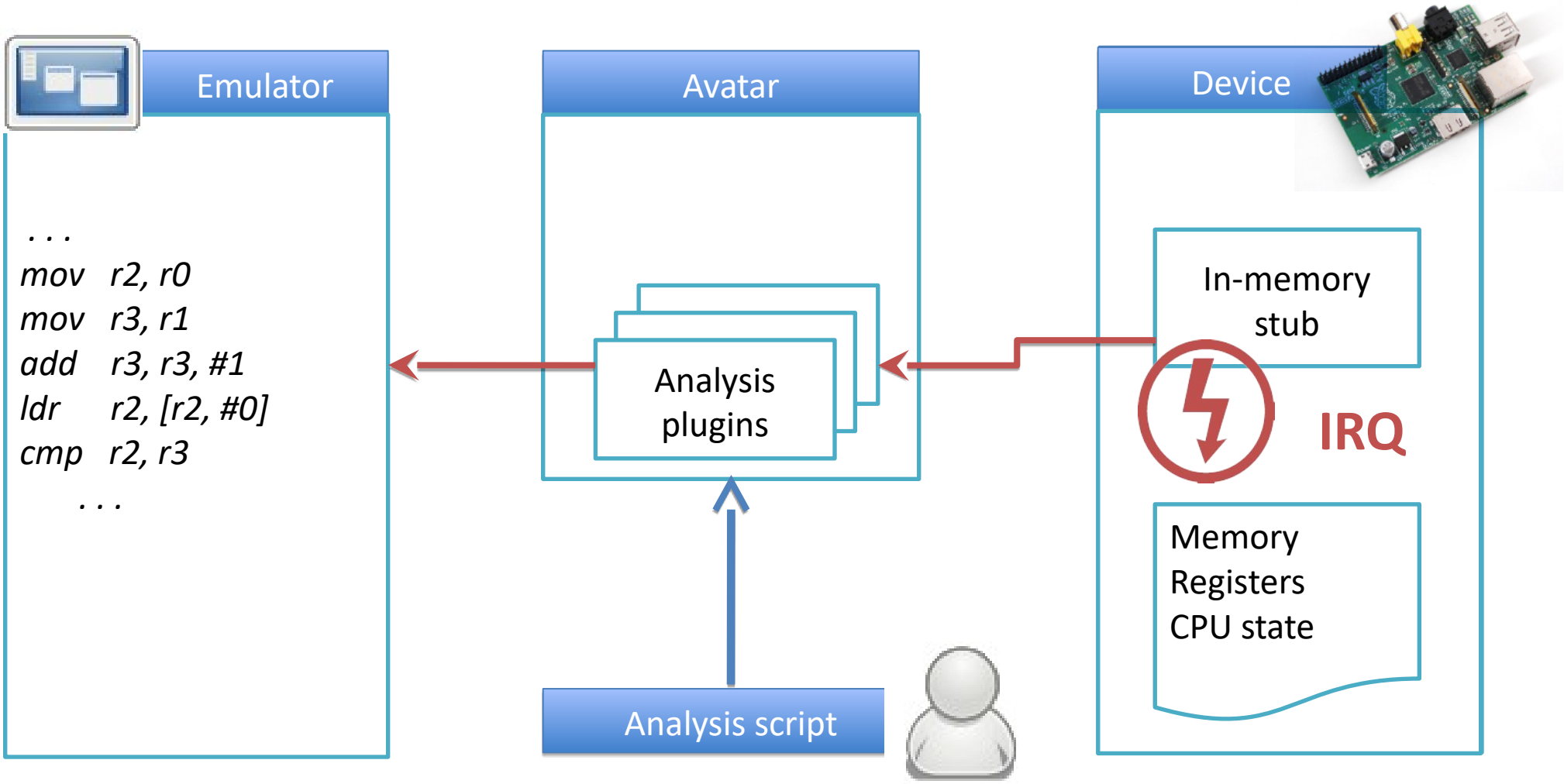
Avatar² overview



Avatar² overview



Avatar² overview



Avatar²

- Successor of Avatar (the first)…
- Complete redesign, maintained
- Open source, all paper examples are available
  - https://github.com/avatartwo/avatar2
- Integrates many tools
  - OpenOCD, Panda, QEMU, Angr

Avatar2 : A Multi-target Orchestration Platform
Marius Muench, Dario Nisi, Aurélien Francillon, Davide Balzarotti
Workshop on Binary Analysis Research 2018

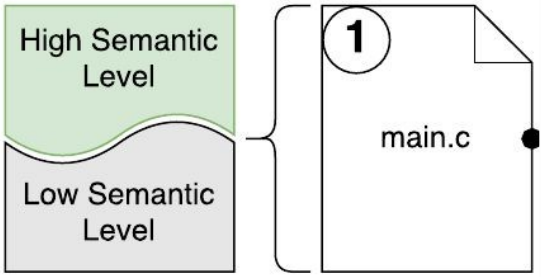
Avatar: A Framework to Support Dynamic Security Analysis of Embedded Systems' Firmwares”
Jonas Zaddach, Luca Bruno, Aurelien Francillon, Davide Balzarotti
NDSS 2014

Inception

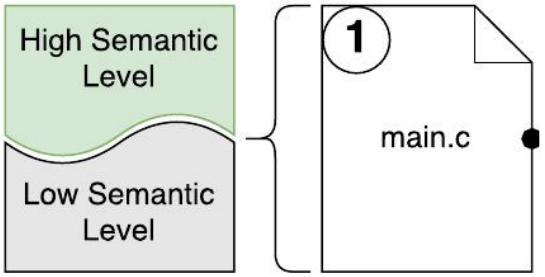
- When we have the device and the source code?
  - There may be some binary blobs
  - Code specific to the hardware
    - Bootloaders, device drivers
- KLEE is a symbolic execution environment
  - Does not handle binary/asm
  - Ignores hardware interaction

"Inception: System-wide Security Testing of Real-World Embedded Systems Software"
N. Corteggiani, G. Camurati, A. Francillon, USENIX Security 2018

Inception Compilation



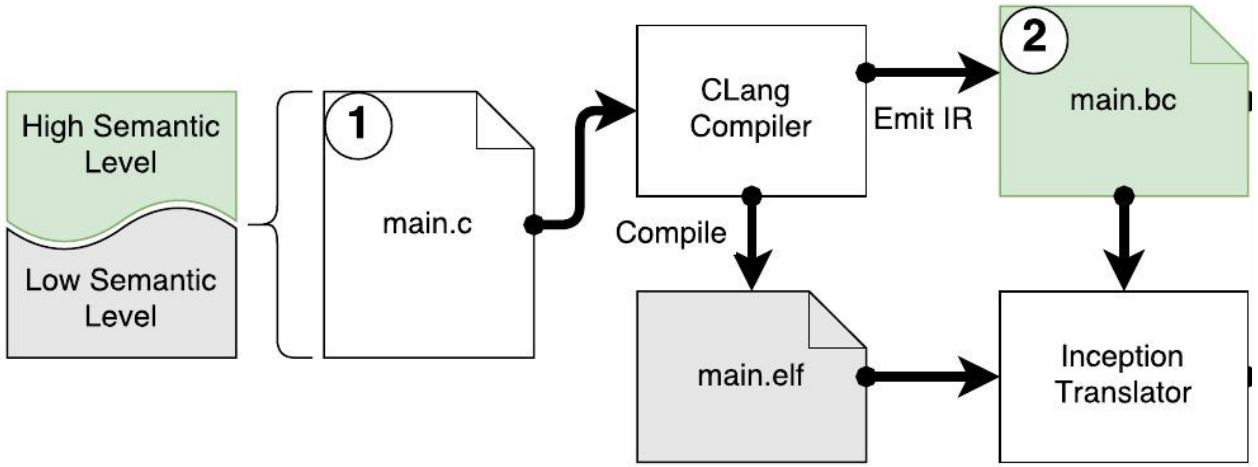
Inception Compilation




```
void uart_send(unsigned char letter) {
__asm volatile("svc #0");
__asm volatile("bx  lr");
}
int main(){
      uart_send(message[i++]);
      return 0;
}
```

1

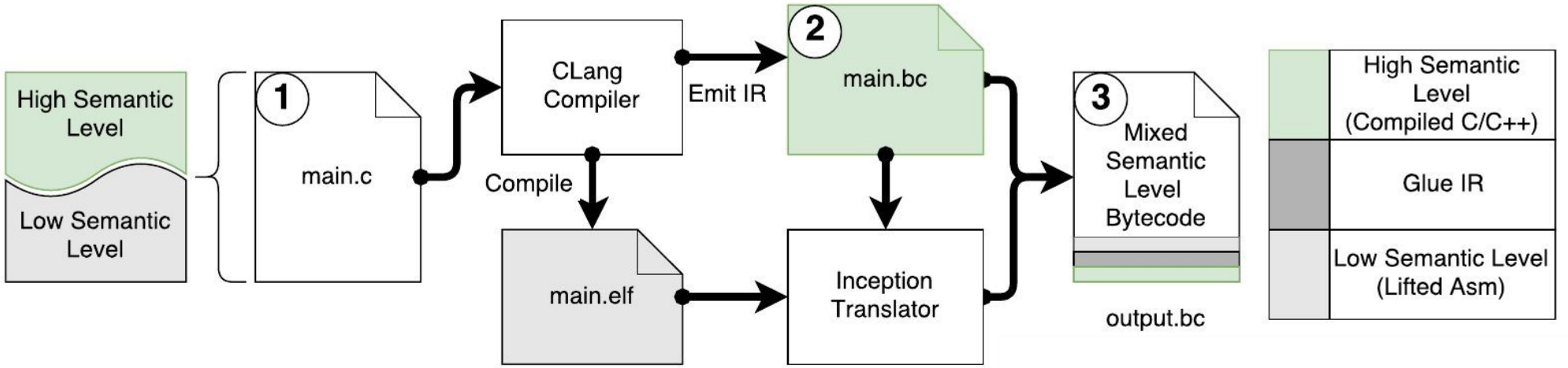
Inception Compilation




```
void uart_send(unsigned char letter) {
__asm volatile("svc #0");
__asm volatile("bx  lr");
}
int main(){
      uart_send(message[i++]);
      return 0;
}
```
1

```
...
 call void asm sideeffect "svc #0", ""()
 call void asm sideeffect "bx  lr", ""()
...
 call void @uart_send(i8 zeroext %1)
 ret void
...
```
2

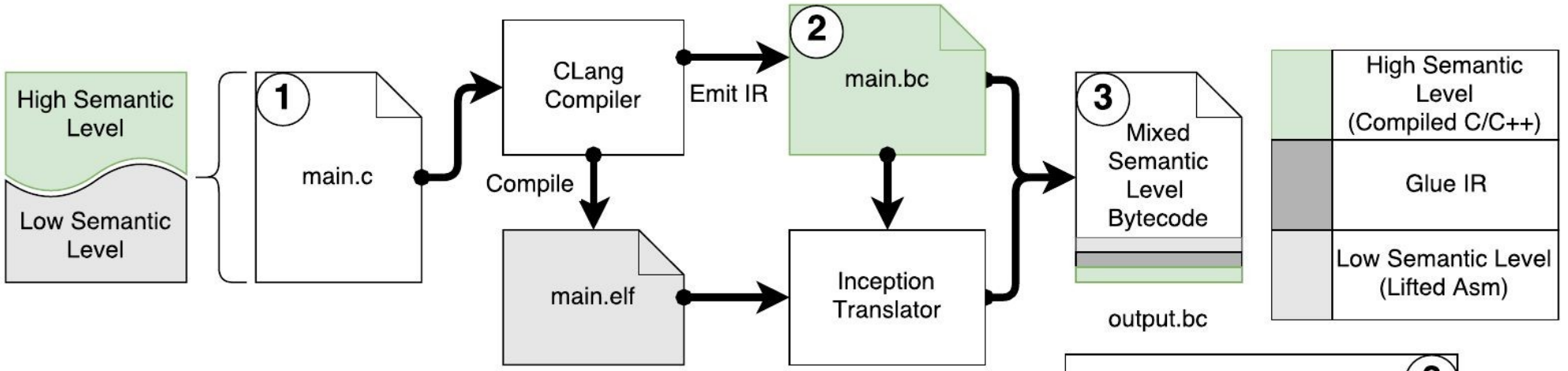
Inception Compilation




```
void uart_send(unsigned char letter) {
__asm volatile("svc #0");
__asm volatile("bx  lr");
}
int main(){
      uart_send(message[i++]);
      return 0;
}
```
1

```
...
 call void asm sideeffect "svc #0", ""()
 call void asm sideeffect "bx  lr", ""()
...
 call void @uart_send(i8 zeroext %1)
 ret void
...
```
2

Inception Compilation




```
void uart_send(unsigned char letter) {
__asm volatile("svc #0");
__asm volatile("bx  lr");
}
int main(){
      uart_send(message[i++]);
      return 0;
}
```
1

```
...
 call void asm sideeffect "svc #0", ""()
 call void asm sideeffect "bx  lr", ""()
...
 call void @uart_send(i8 zeroext %1)
 ret void
...
```
2

```
...
entry:
 %1 = zext i8 %0 to i32
 store i32 %1, i32* @R0
 br label %"uart_send+0"
...
"uart_send+0":
 call void (...)* @inception_sv_call()
 %LR1 = load i32* @LR
 ret void
...
 call void @uart_send(i8 zeroext %1)
 ret void
```
3

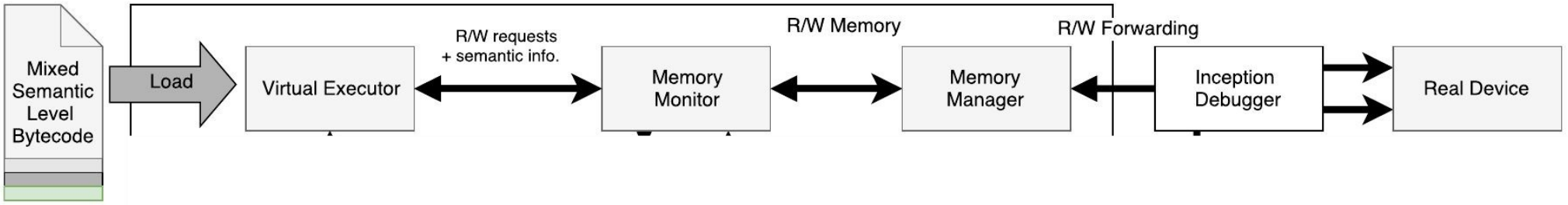
Inception Execution

Inception Execution

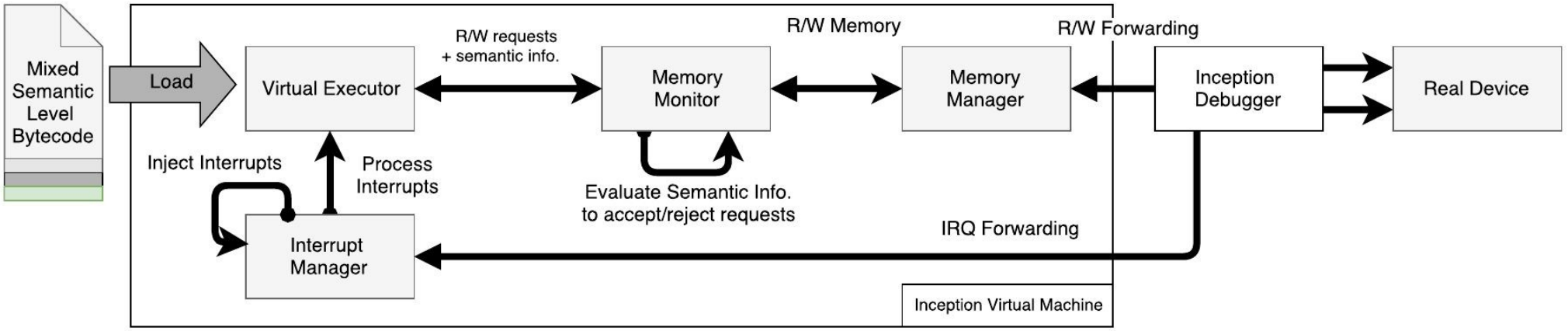
Inception Execution

- Executor => Modified Klee

Inception Execution

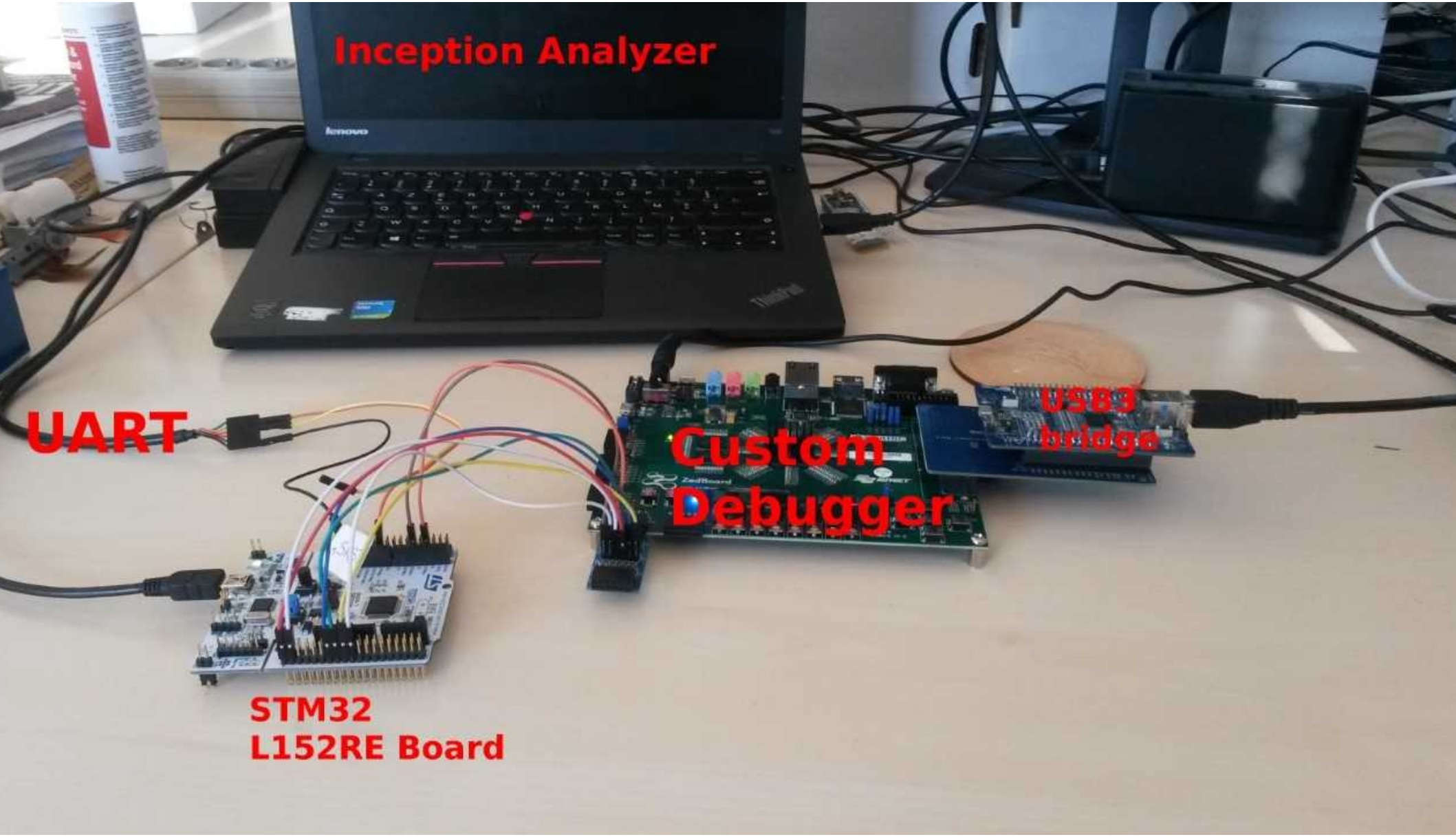


Inception Execution



Memory checks

		Allocation					
		C with KLEE allocator		C native allocator		ASM or Binary	
		Accessed from					
		C	asm	C	asm	C	asm
Dynamic allocation	Check Types	✓	✓	✗	✗	✗	✗
	Red Zone	✓	✓	✗	✗	✗	✗
	Heap Consistency checks	✓	✓	✗	✗	✗	✗
Stack allocation	Check Types	-	-	✓	✓	✗	✗
	Red Zone	-	-	✓	✓	✗	✗
.Data or .BSS allocation	Check Types	-	-	✓	✓	✗	✗
	Red Zone	-	-	✓	✓	✗	✗
Not allocated memory	KLEE detection	-	-	✓	✓	✓	✓


Inception Analyzer
UART
Custom
Debugger
USB3
bridge
STM32
L152RE Board

Symbolic Execution as a compilation

- SymCC : an LLVM compiler pass for embedding symbolic execution into binaries
  - Shows very high performance
  - Used in hybrid fuzzing
- SymQEMU:
  - for binaries
  - integrated in QEMU emulator
- Both projects will soon be used for embedded devices too

Symbolic execution with SymCC: Don't interpret, compile!
S. Poeplau, A. Francillon
29th USENIX Security Symposium, 2020 , Boston, MA
Distinguished Paper Award Winner

SymQEMU: Compilation-based symbolic execution for binaries
S. Poeplau, Francillon, Aurélien,
NDSS 2021

Conclusion

- Embedded software security is difficult
  - Many specific aspects to embedded software
  - Hardware customisation makes every device very different
- Various possible approaches
  - we need to continue develop analysis methods and tools
- A very active field of research
  - a lot of research needed

Thanks

Questions ?

This work together with many people

- Andrei Costin
- Jonas Zaddach
- Giovanni Camuratti
- Nassim Corteggiani
- Apostolis Zarras
- Davide Bazarotti
- Marius Muench
- Dario Nisi
- Paul Olivier...

Institut Mines-Télécom

EURECOM

Sophia Antipolis

Academia

Institut Mines-Télécom

Politecnico di Torino

Aalto University

Technische Universität München

NTNU
Norwegian University of Science and Technology

CHALMERS
UNIVERSITY OF TECHNOLOGY

CZECH TECHNICAL UNIVERSITY IN PRAGUE

Industry and institution

orange

SAP

monaco telecom

BMW GROUP THE NEXT 100 YEARS

Symantec

iABG

Gouvernement Princier
PRINCIPAUTÉ DE MONACO

Dan Guido

Follow @dguido 23.1K followers

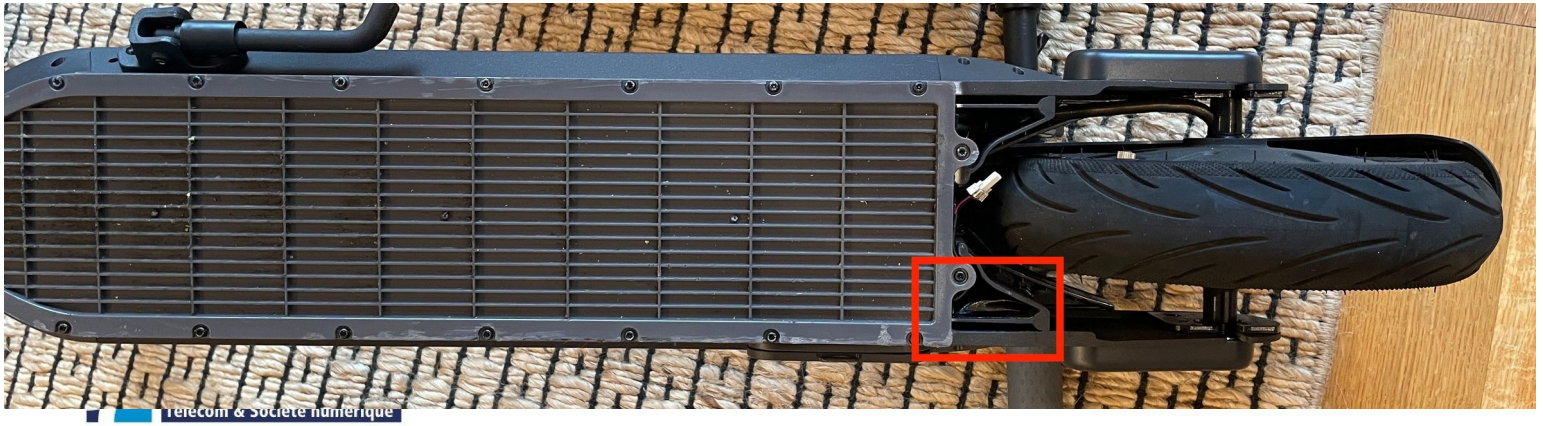
10 Aug, 22 tweets, 7 min read

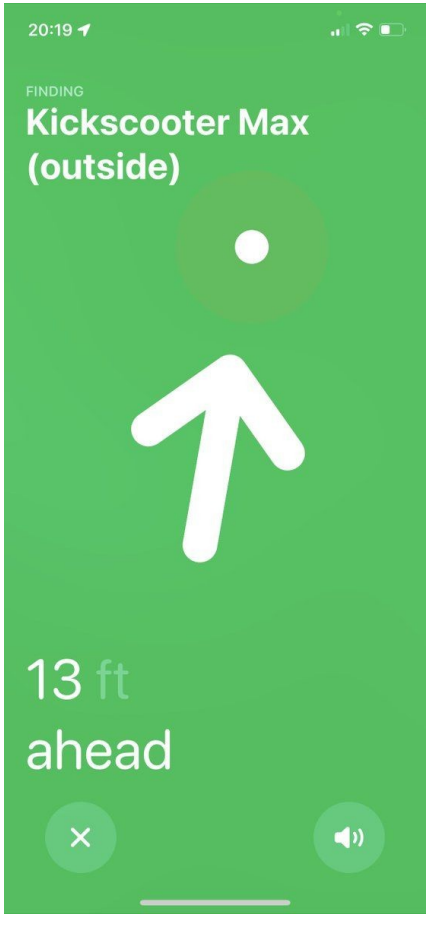
Bookmark Save as PDF + My Authors

My scooter was stolen last week. Unknown to the thief, I hid two Airtags inside it. I was able to use the Apple Find My network and UWB direction finding to recover the scooter today. Here's how it all went down:

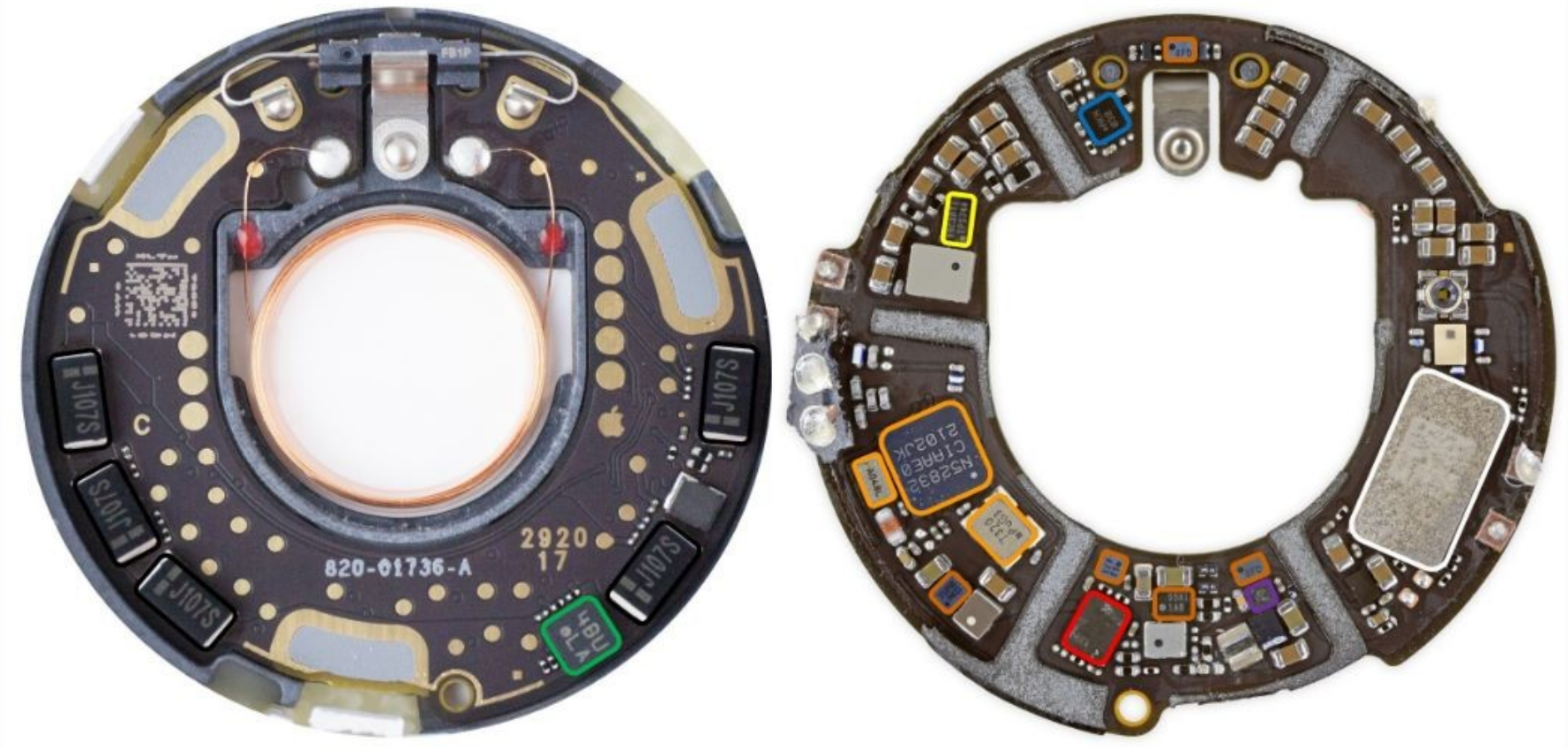
The theft occurred on Monday night. I went out to dinner and locked it to a grate with motorcycle handcuffs. I find them easier to use than a cable lock, but apparently I forgot to lock one cuff. It was gone after ~2 hours.

amazon.com/gp/product/B00…





PCB Overview



- Nordic nRF52832 SoC with BLE and NFC, plus 32MHz and 32.768kHz crystals
- Apple U1 UWB Transceiver
- GigaDevice GD25LE32D 32Mbit NOR flash
- Bosch BMA280 accelerometer
- Maxim MAX98357AEWL audio amplifier
- TI TPS62746 DC-DC buck converter
- TI TLV9001IDPWR opamp
- 100uF Electrolytic Capacitors (5x)
- Unknown. Unable to decode markings

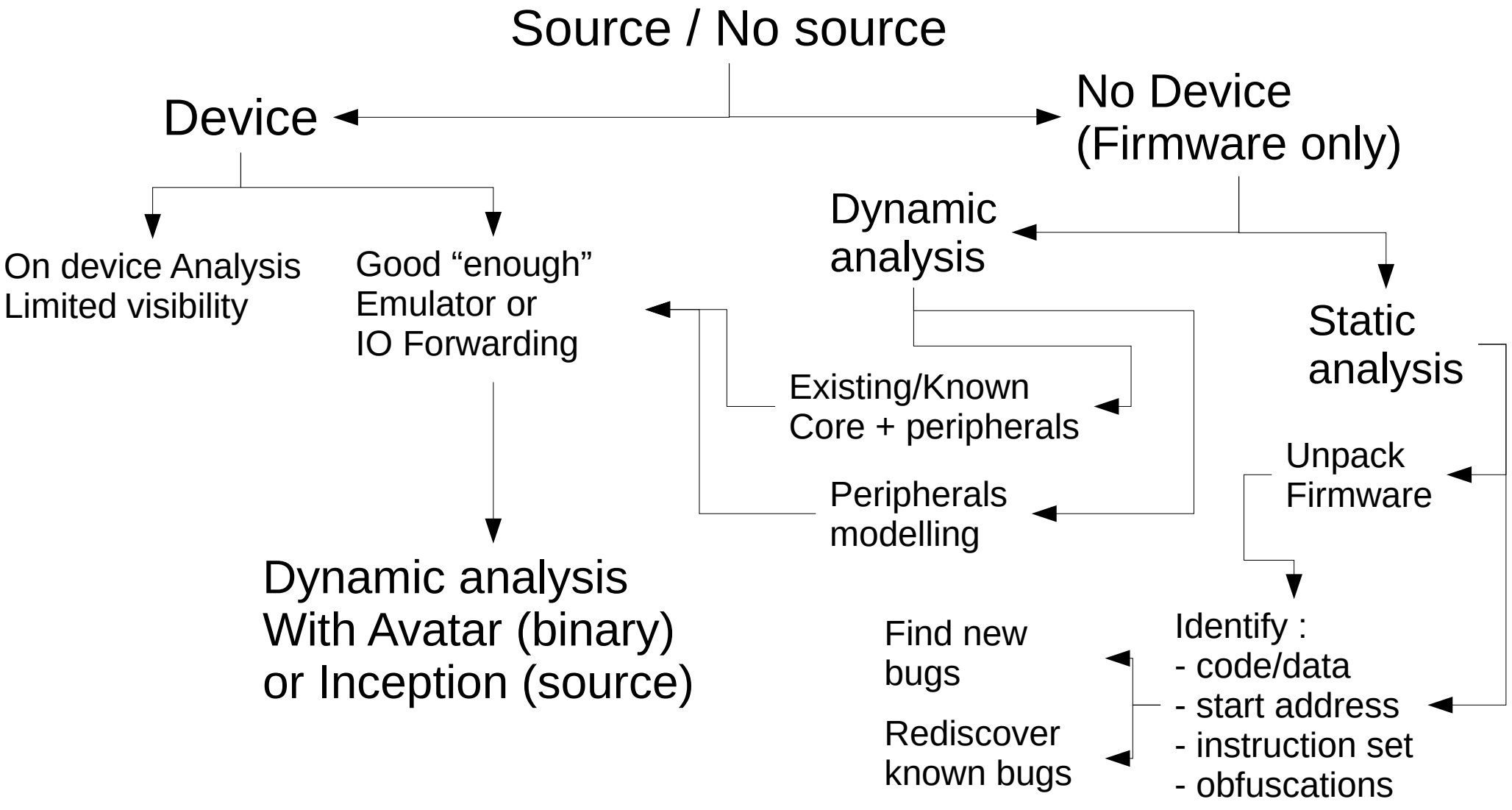
Antennas

There are three antennas inside the AirTag:

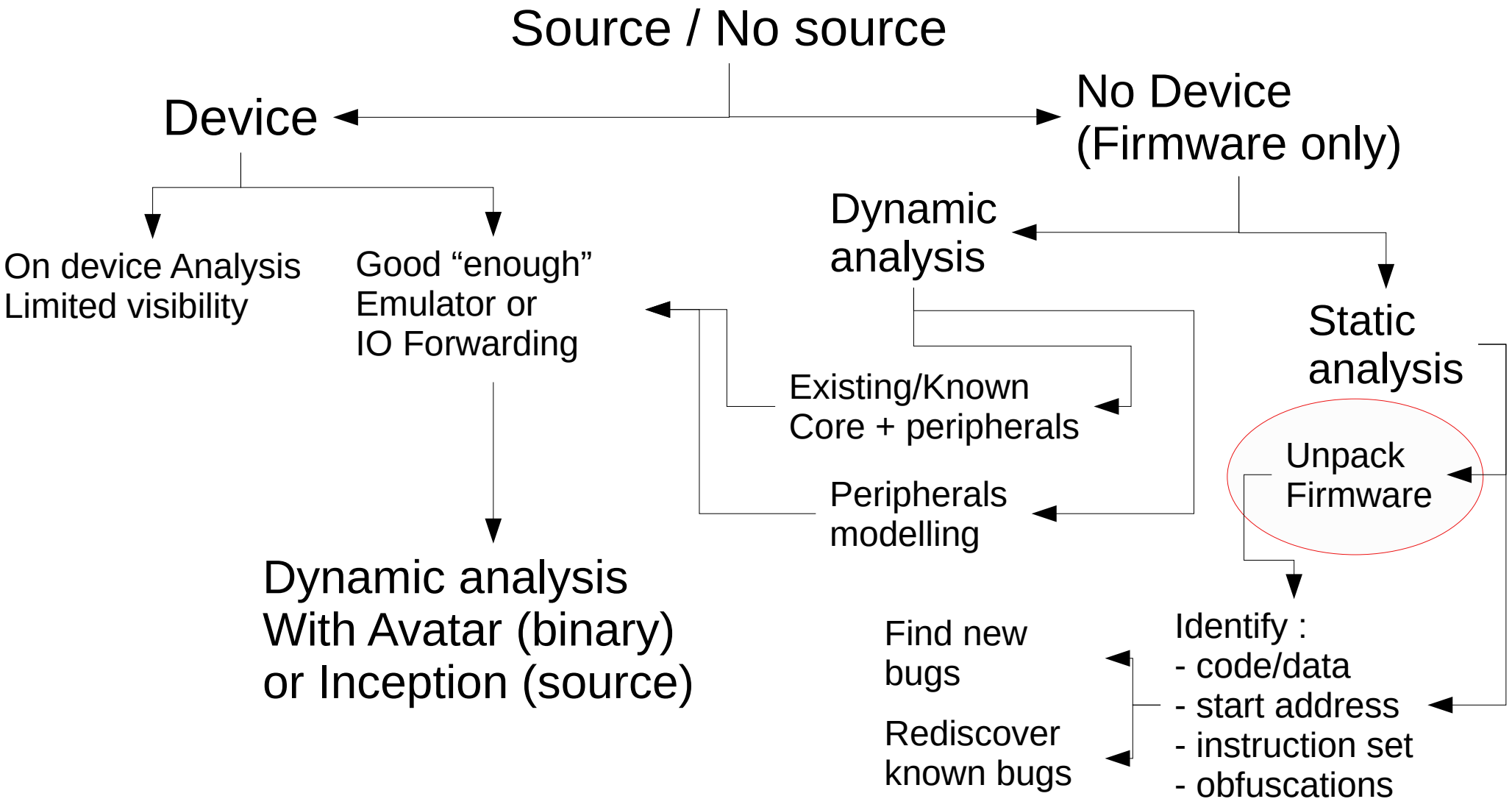
1. Bluetooth Low Energy (left) - 2.4GHz
2. NFC (middle) - 13.56MHz
3. Ultra-Wideband (right) - 6.5-8GHz

They are all etched onto a single piece of plastic using Laser Direct Structuring (LDS) and then soldered to the PCB around the edge. The NFC antenna also has a short trace on the other side of the plastic (connected with a via at each end) to return the inside end of the coil to the PCB.

Firmware analysis options!



Firmware analysis options!



Manual analysis process

firmware

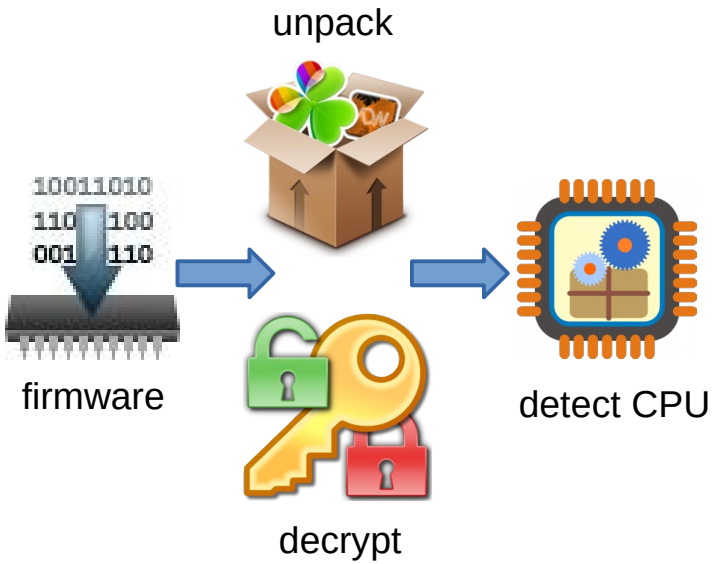
Manual analysis process

IHEX format

```
:100000000C942A000C9434000C9434000C943400AA
:100010000C9434000C9434000C9434000C94340090
:100020000C9434000C9434000C9434000C94340080
:100030000C9434000C9434000C9434000C94340070
:100040000C9434000C9434000C9434000C94340060
:100050000C94340011241FBECFE5D8E0DEBFCDBF25
:100060000E9436000C9445000C9400008FEF87BB73
:100070002CE231E088B3809588BB80E197E2F901FA
:0E0080003197F1F70197D9F7F5CFF894FFCF3C
:00000001FF
```

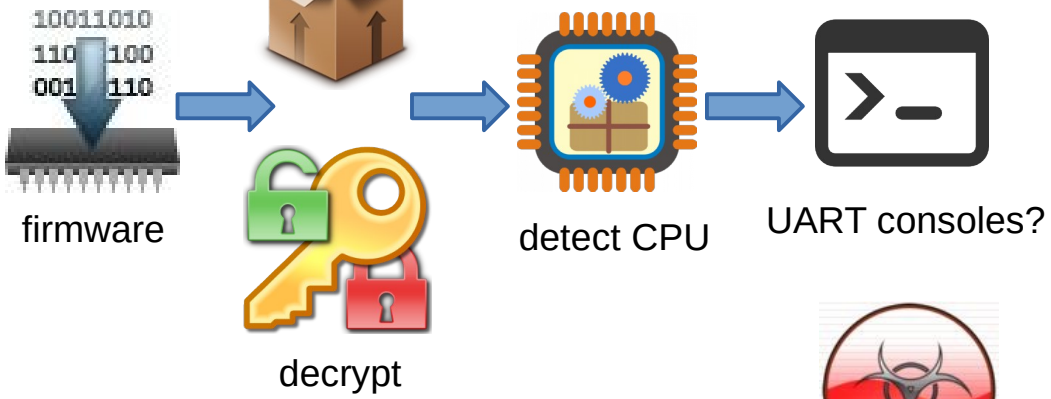
plain text firmware

Manual analysis process



Motorola m68k-based CPU

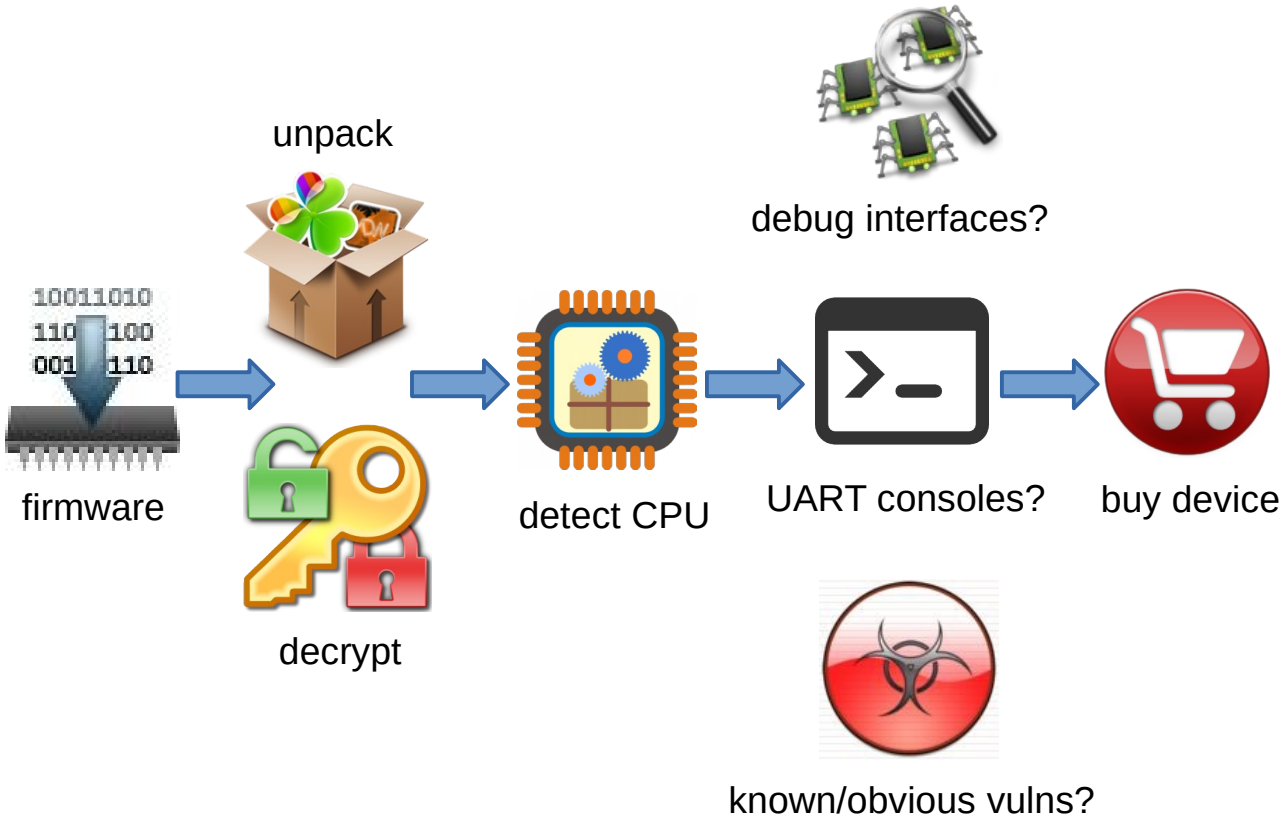
Manual analysis process



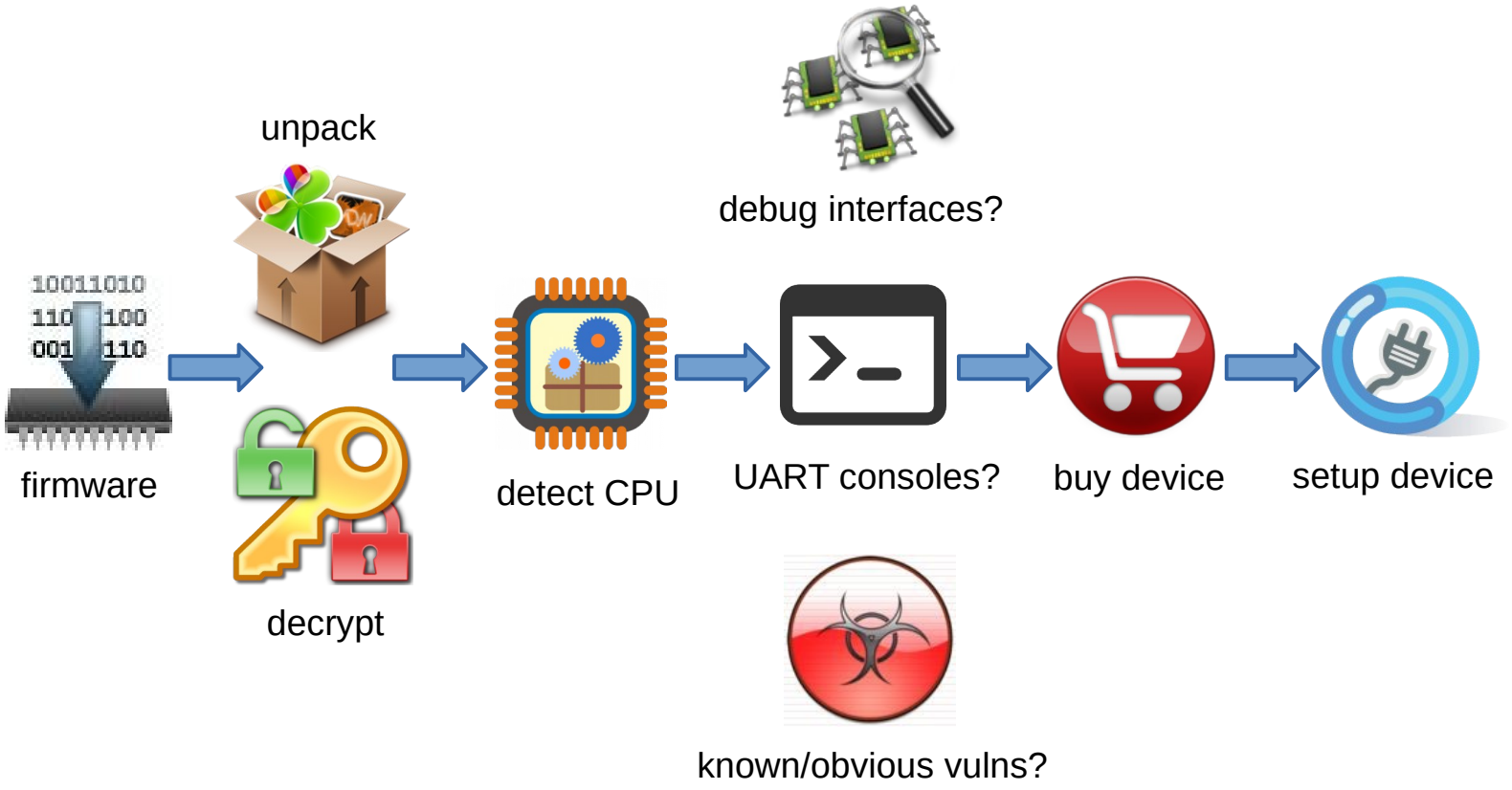
UART “boot>” prompts

802.15.4 functions

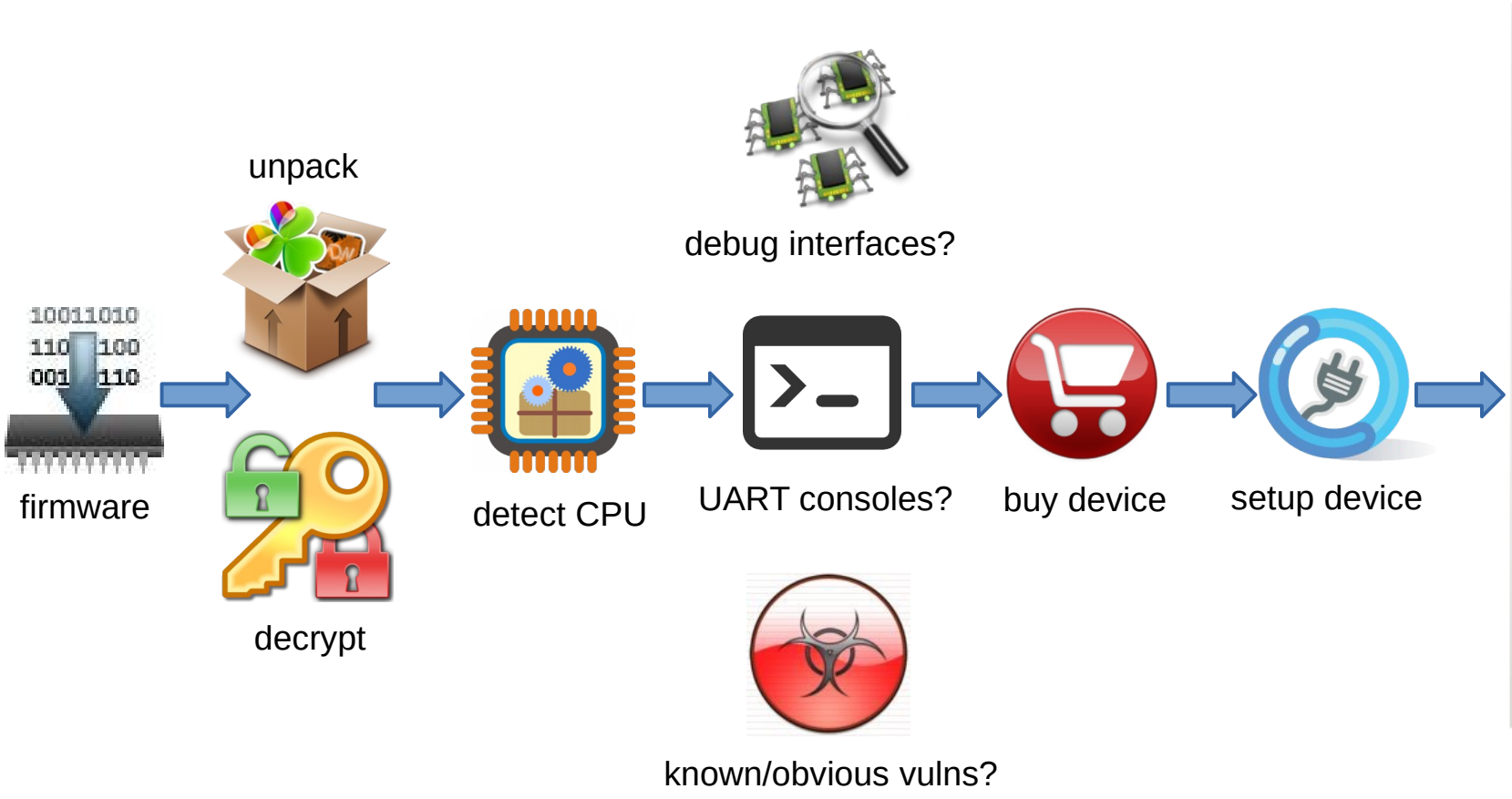
Manual analysis process



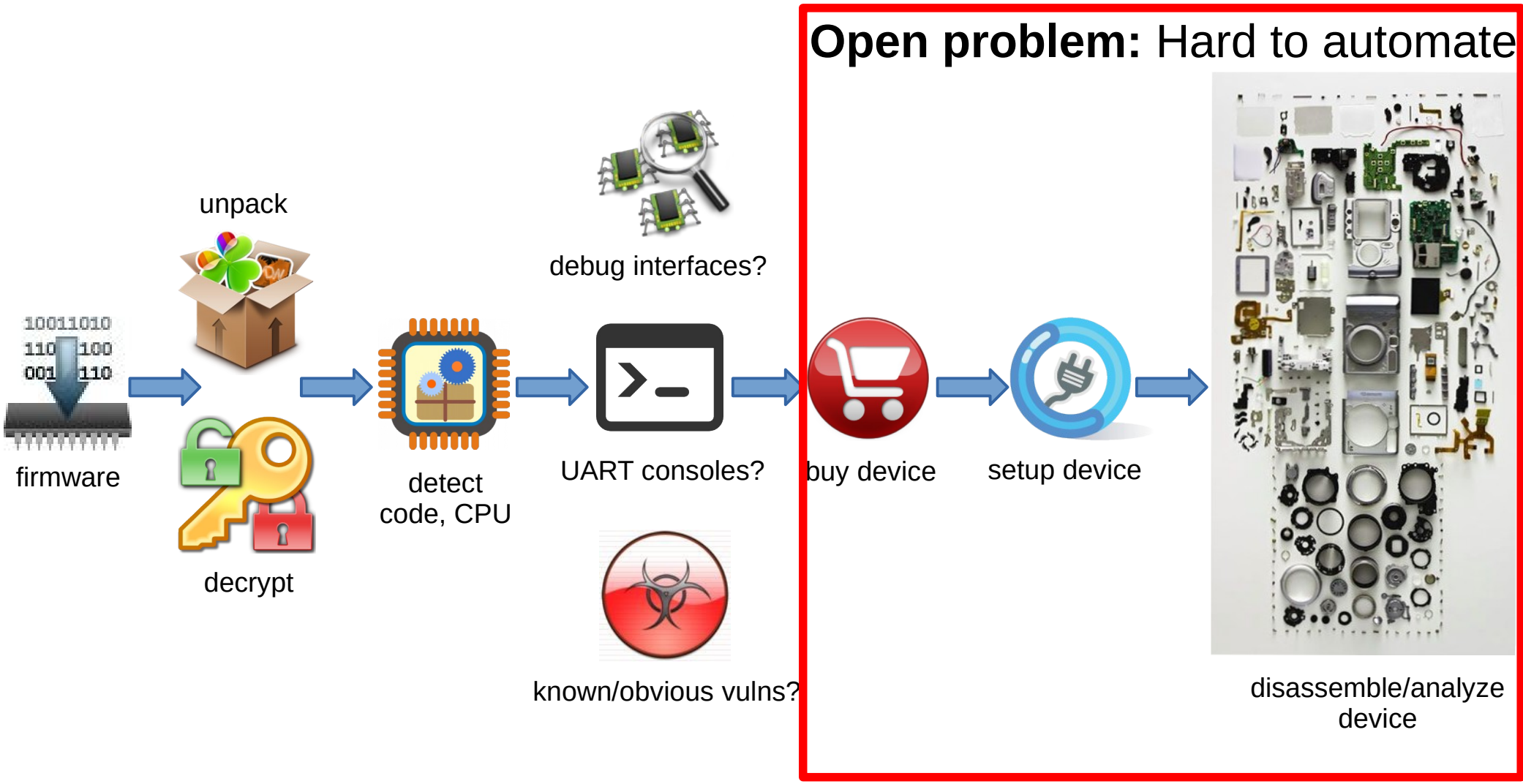
Manual analysis process



Manual analysis process



Manual analysis process



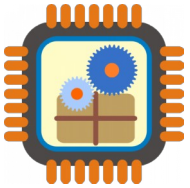
Manual analysis process

Goal: Automate these steps

firmware

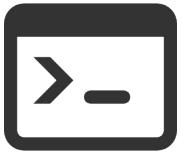
unpack

decrypt



detect
code, CPU

debug interfaces?



UART consoles?

known/obvious vulns?

buy device

setup device

disassemble/analyze
device

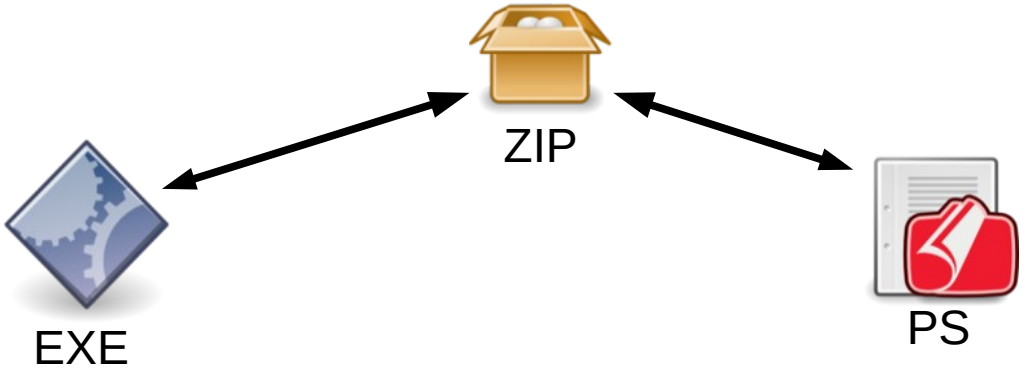
Unpacking & Custom Formats

- How to reliably unpack and learn formats?

ZIP

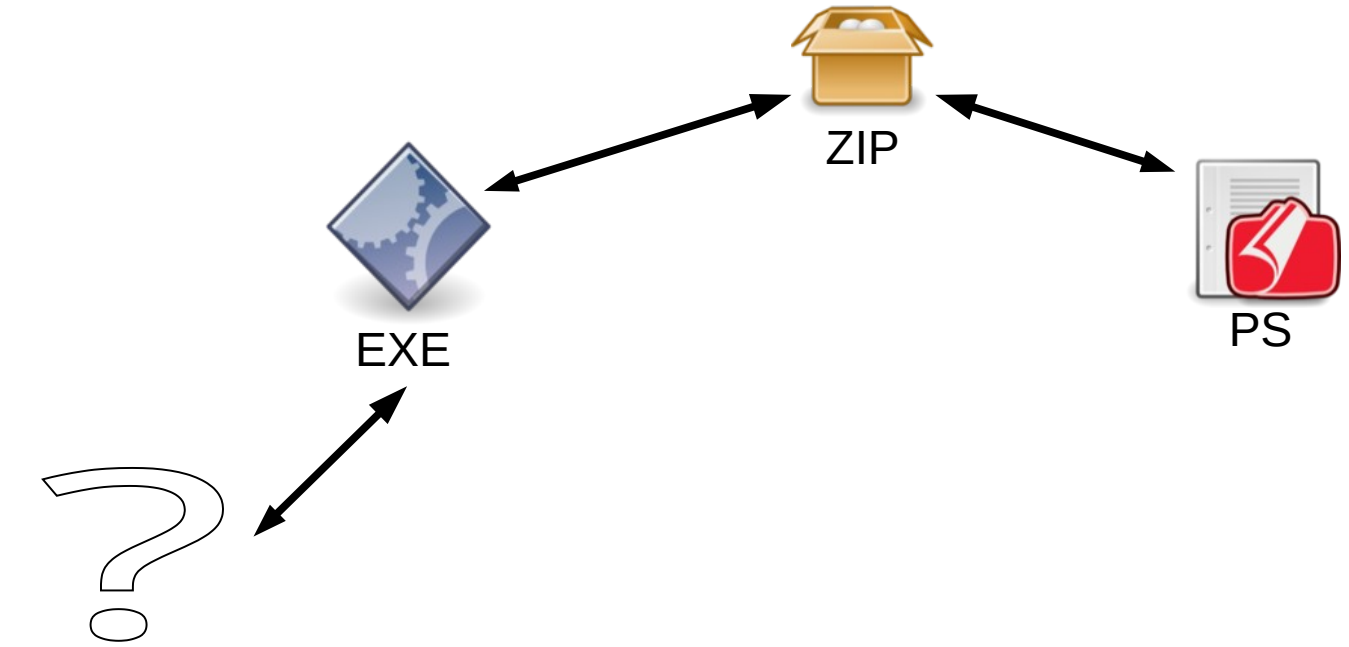
Unpacking & Custom Formats

- How to reliably unpack and learn formats?



Unpacking & Custom Formats

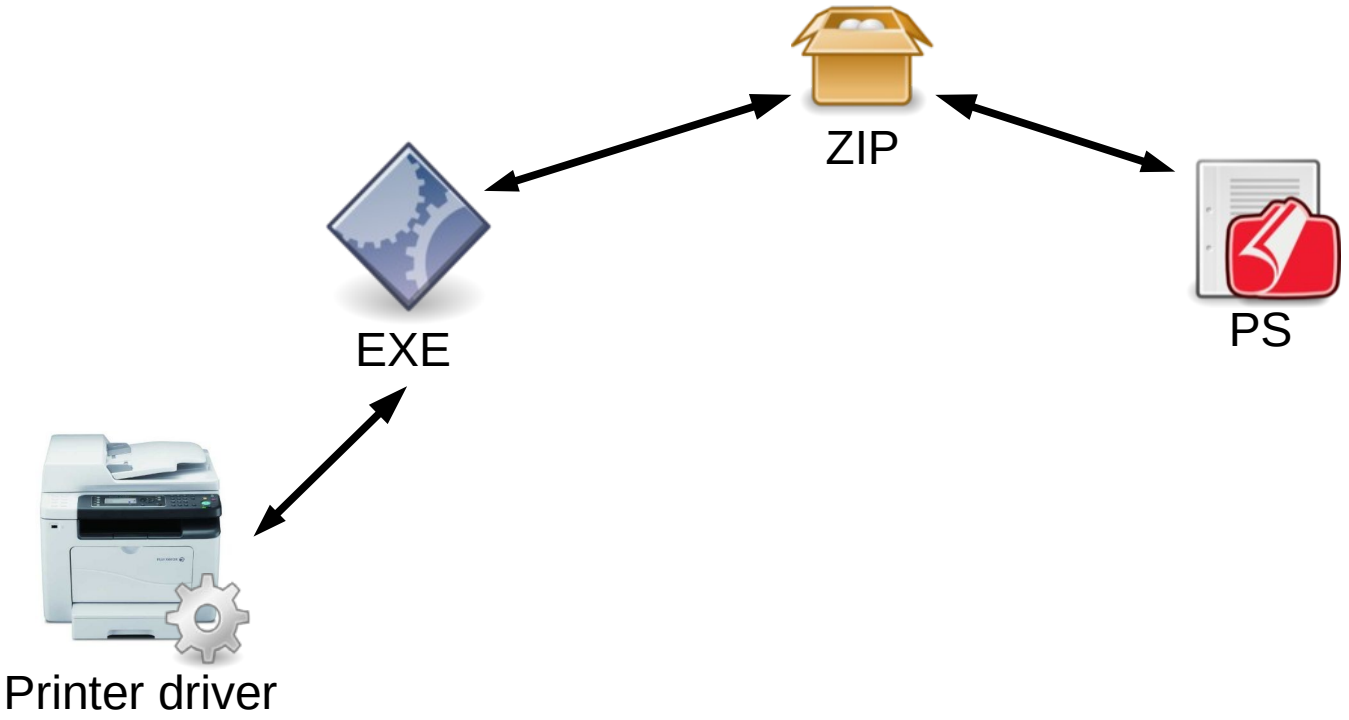
- How to reliably unpack and learn formats?



-Is executable?
-Yes!
-Then is firmware!

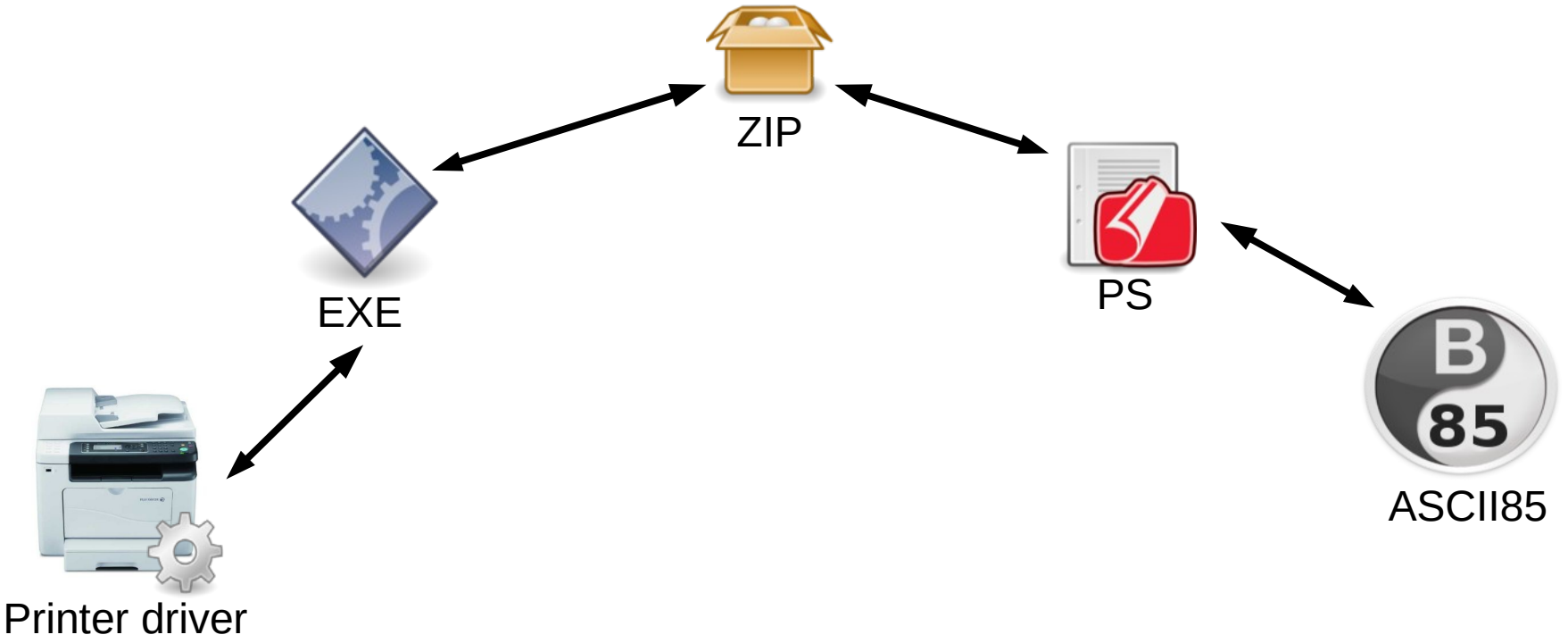
Unpacking & Custom Formats

- How to reliably unpack and learn formats?



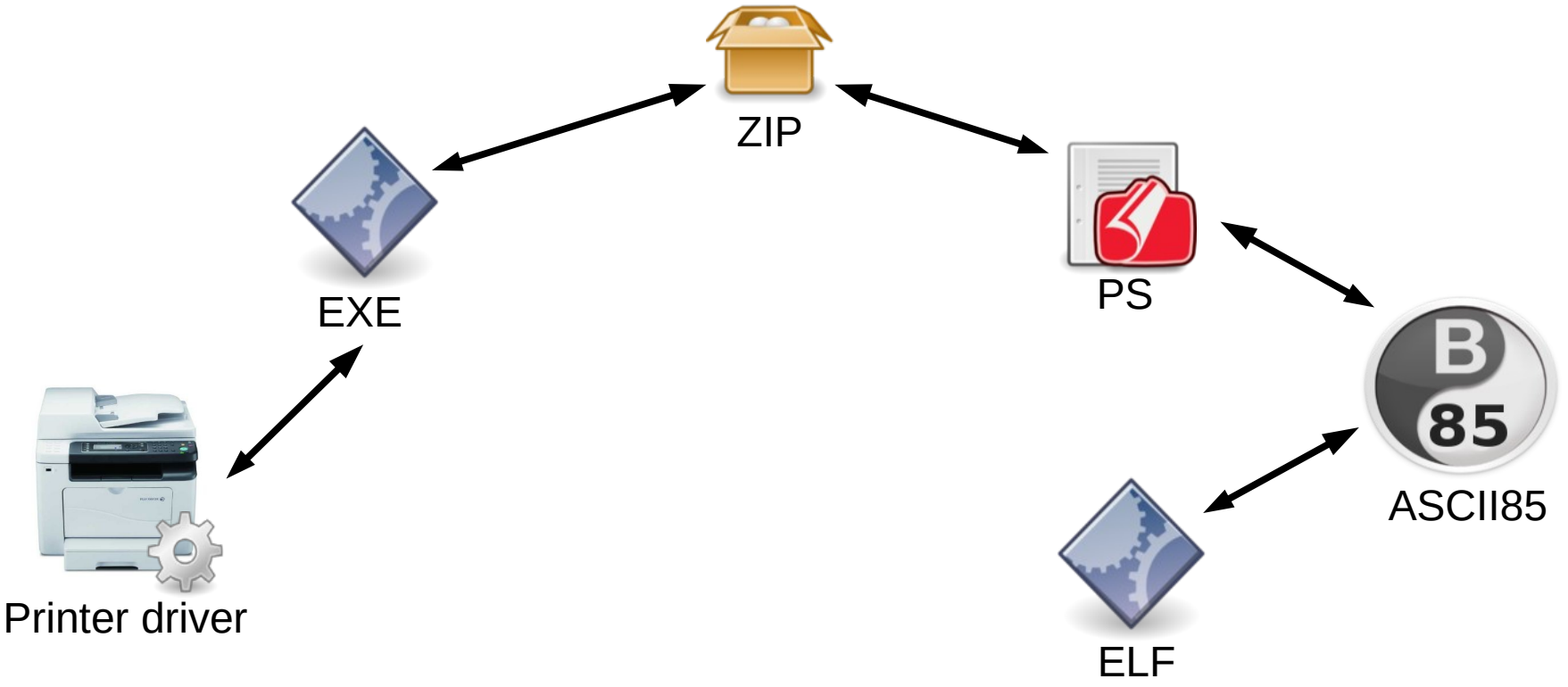
Unpacking & Custom Formats

- How to reliably unpack and learn formats?



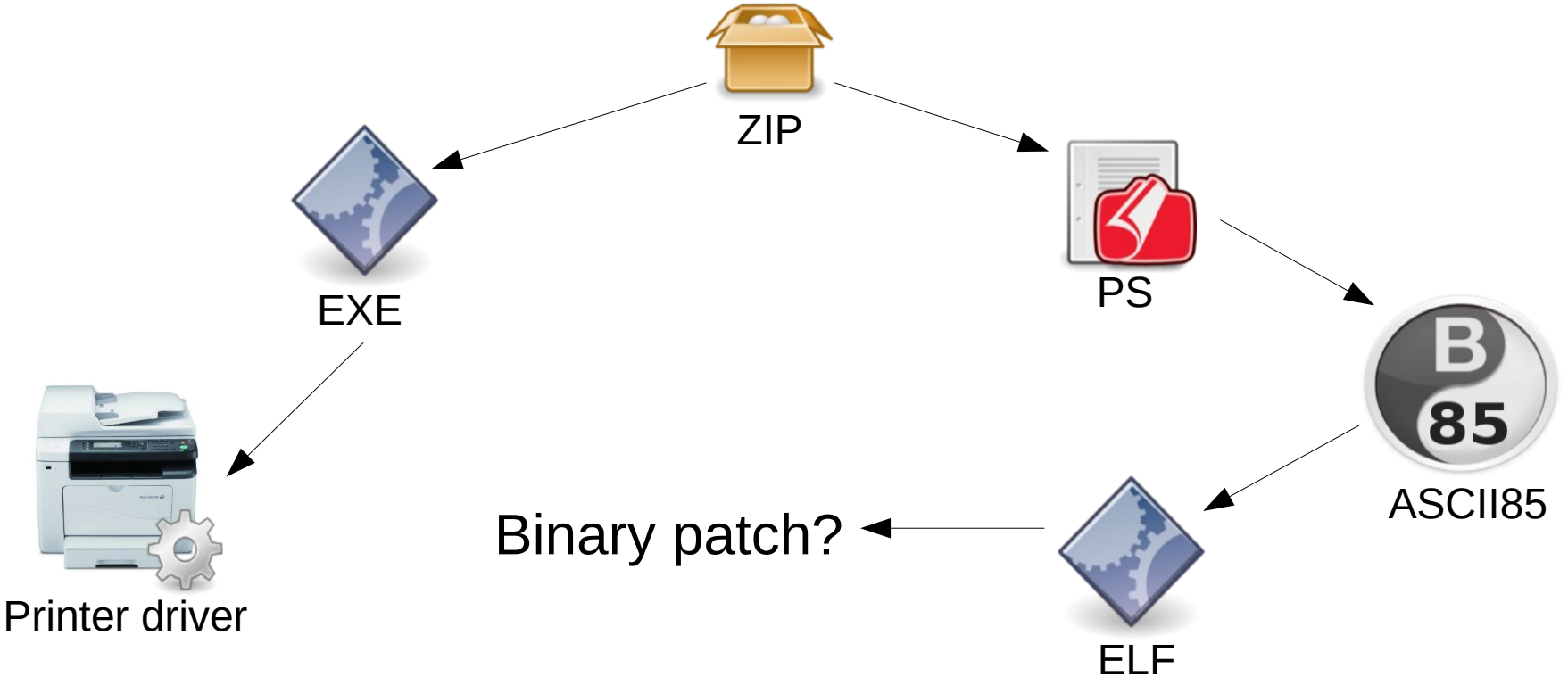
Unpacking & Custom Formats

- How to reliably unpack and learn formats?



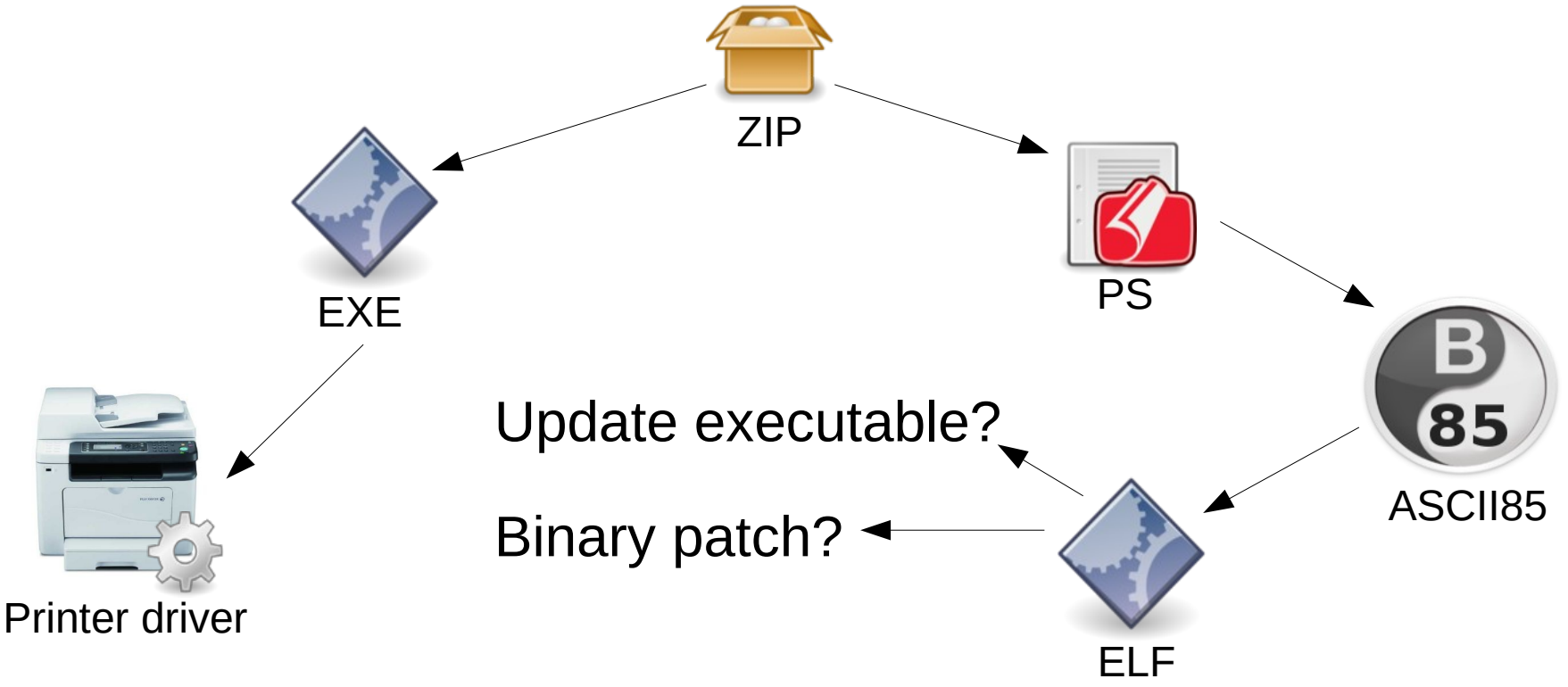
Unpacking & Custom Formats

- How to reliably unpack and learn formats?



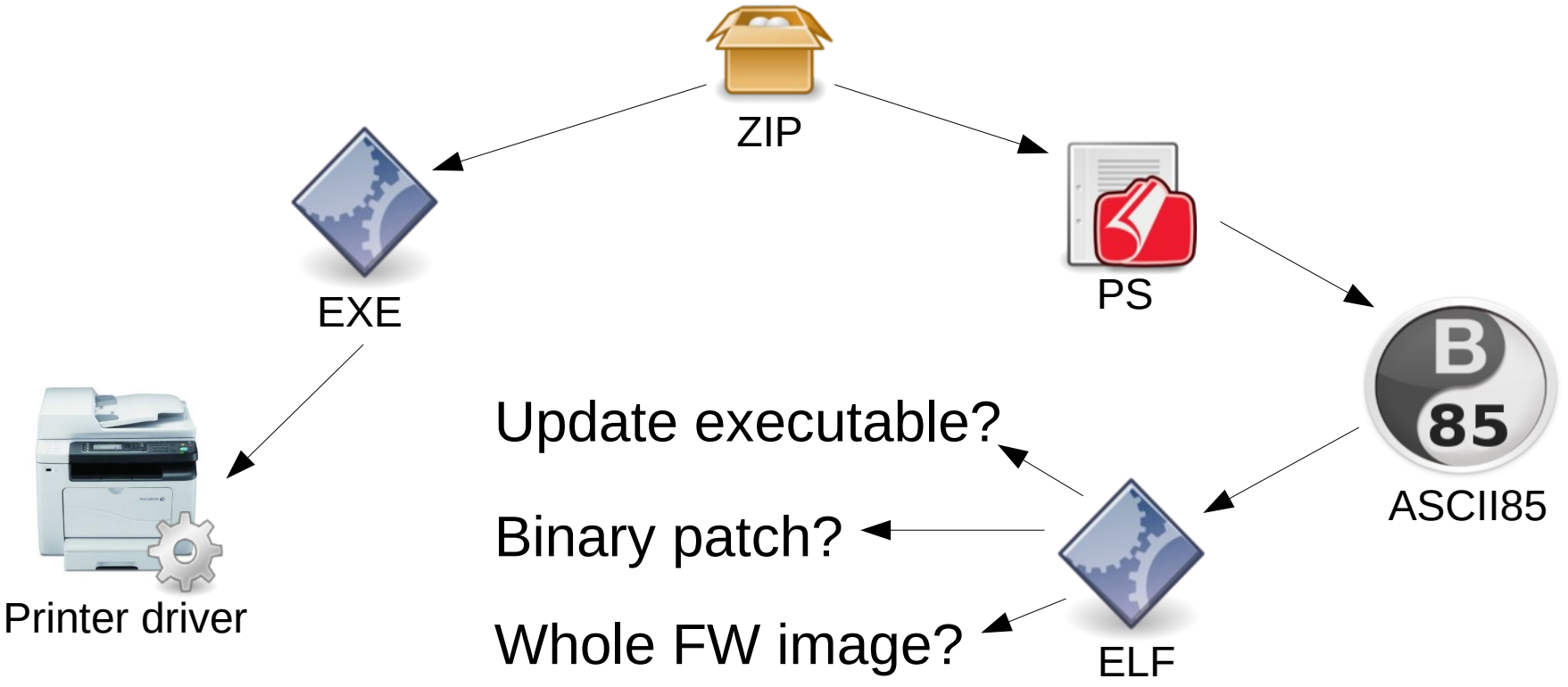
Unpacking & Custom Formats

- How to reliably unpack and learn formats?



Unpacking & Custom Formats

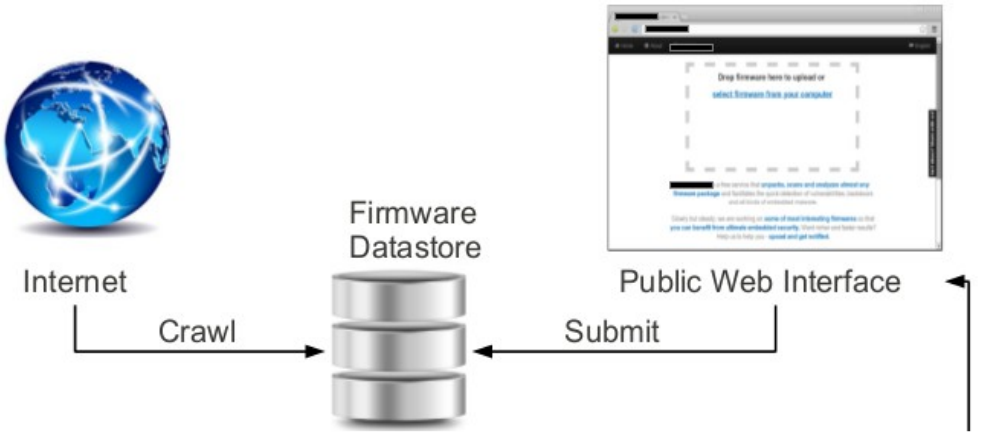
- How to reliably unpack and learn formats?



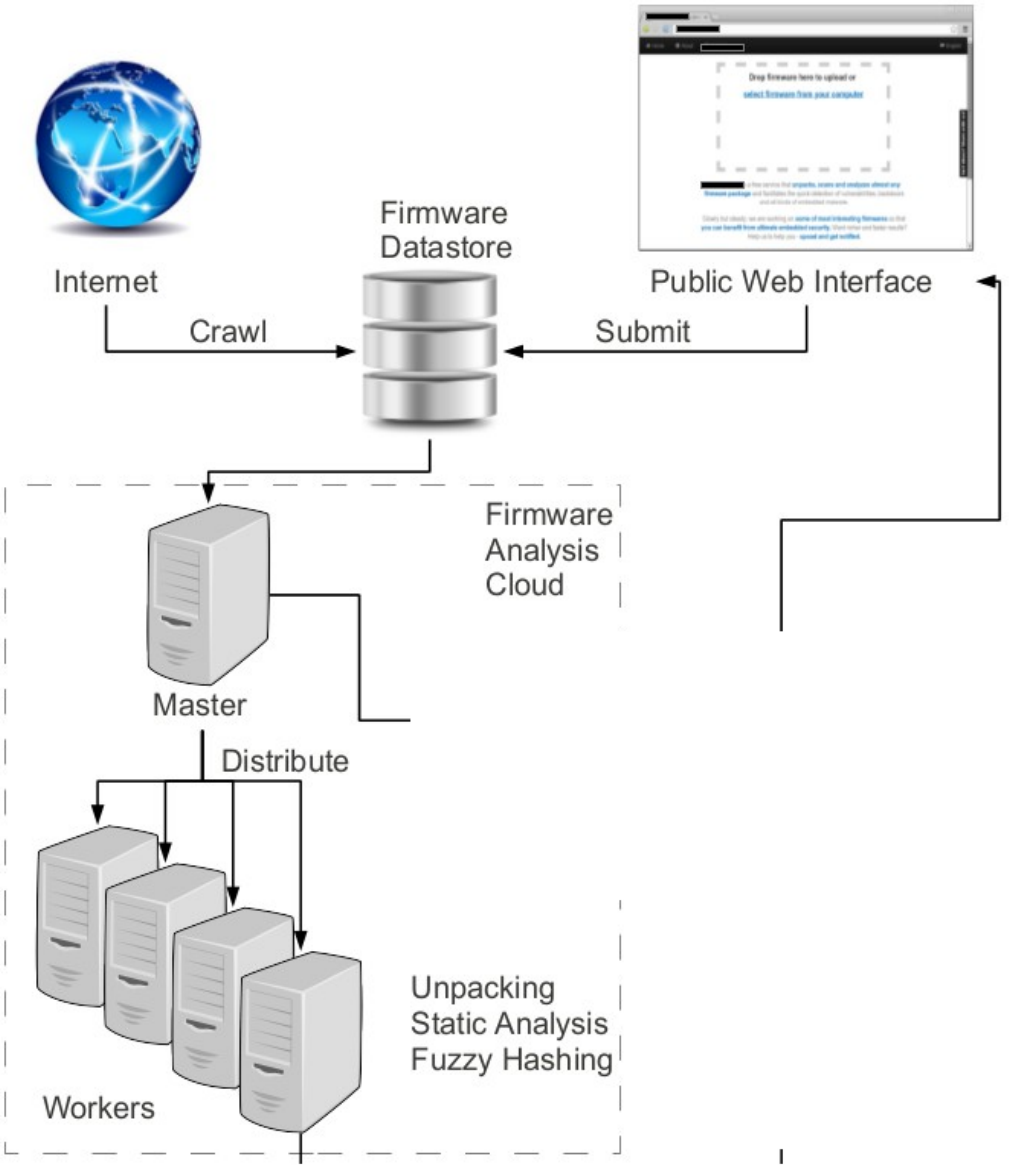
Unpacking & Custom Formats

- Often a firmware image is just a binary blob
  - File carving required
    - Bruteforce at every offset with all known unpackers
  - Have good heuristics to prioritize unpackers
  - Have good heuristics when to stop carving

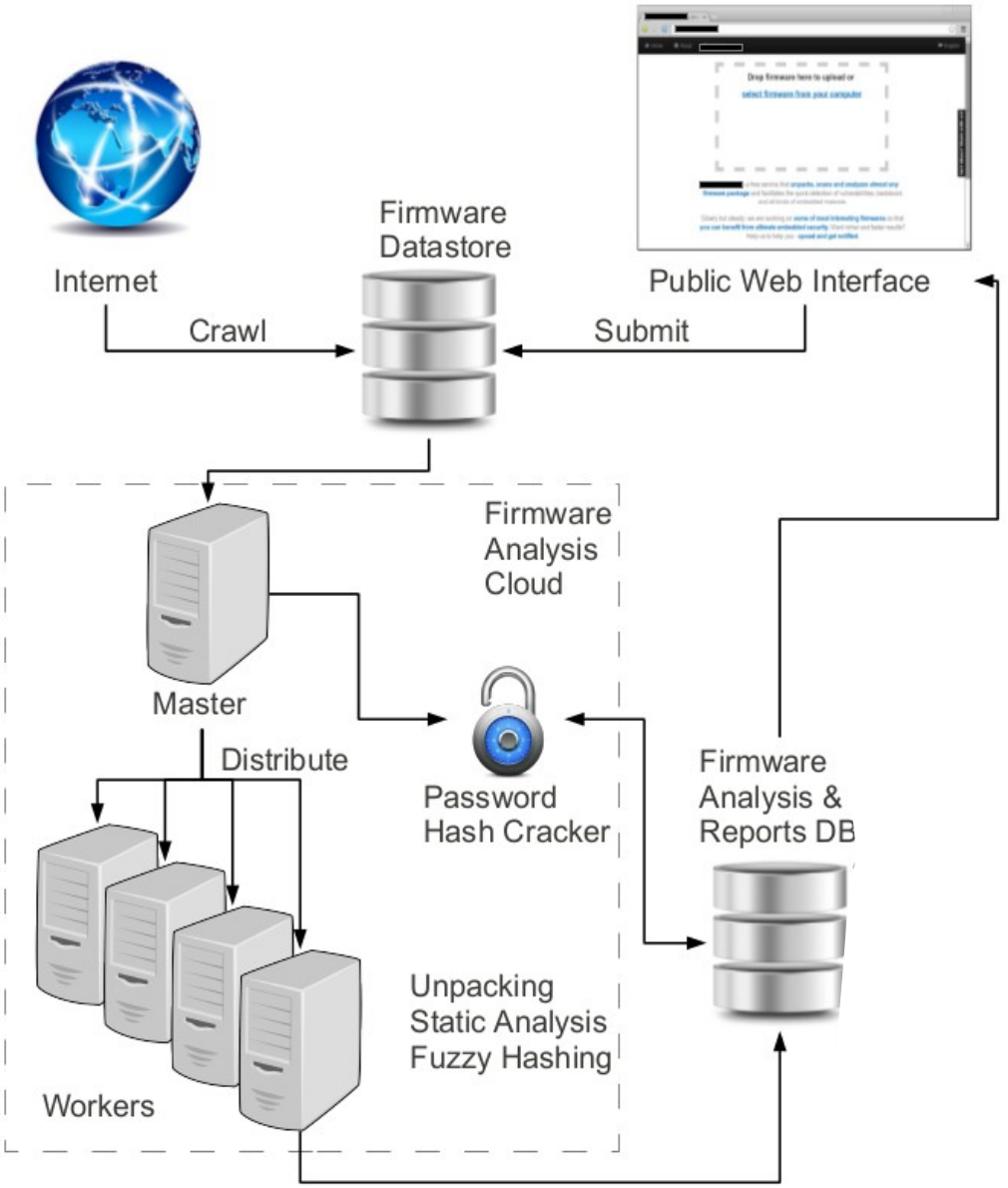
Architecture



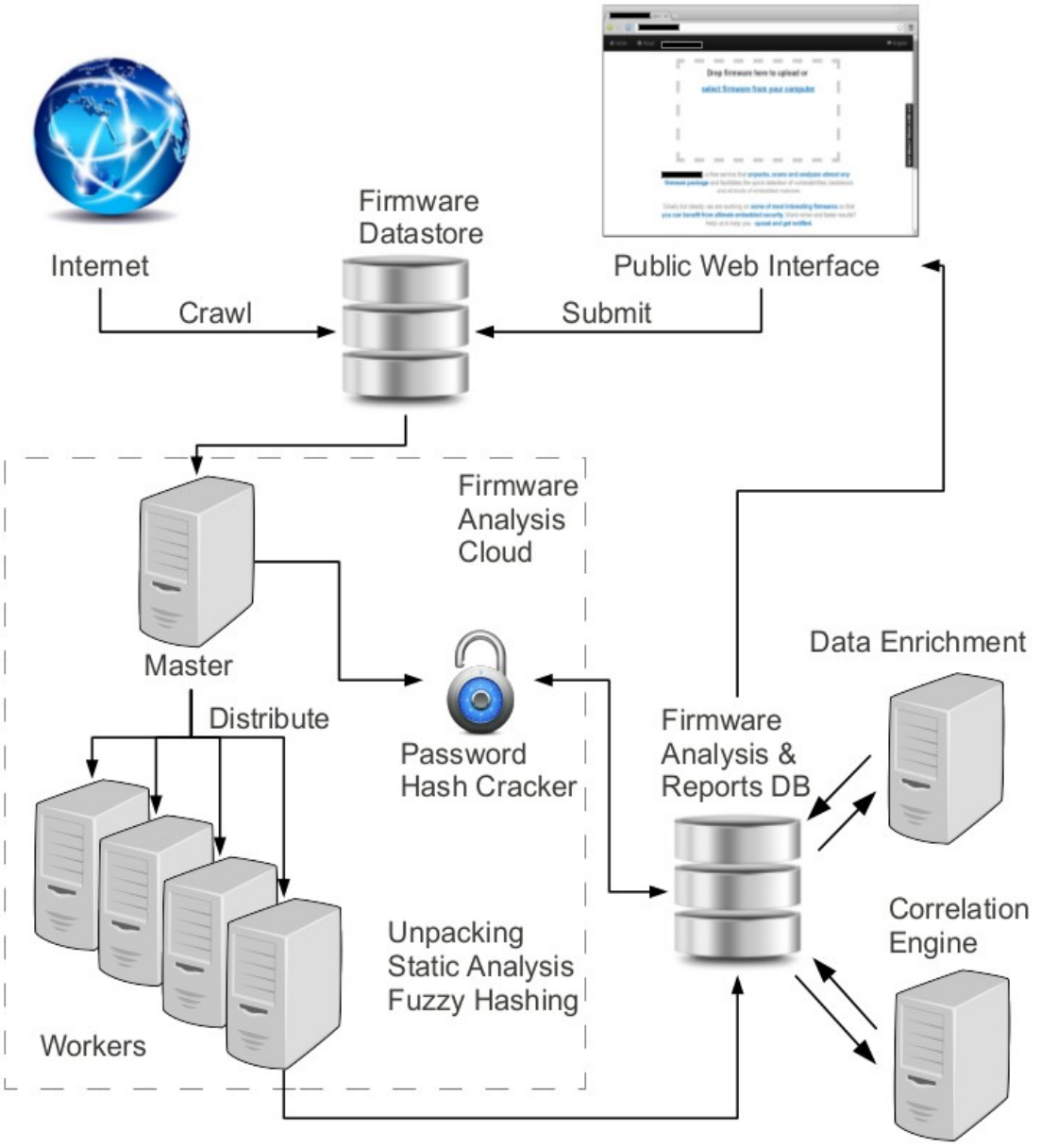
Architecture



Architecture



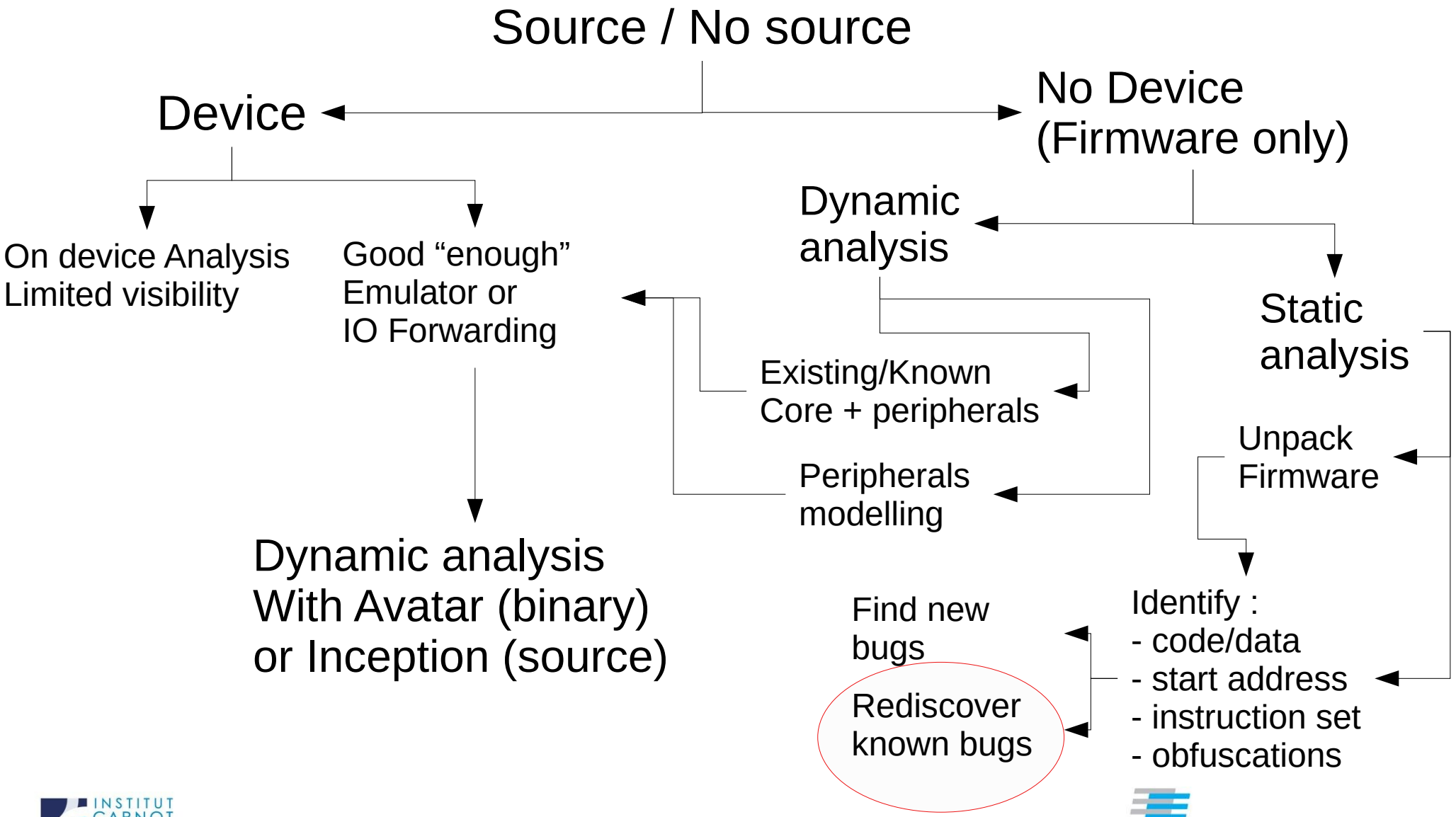
Architecture



Unpacking

- 759 K total files collected
  - ↓ Filter non firmware
- 172 K filtered files (firmware candidates)
  - ↓ Random selection
- 32 K firmwares analyzed
  - ↓ Unpack attempt
- 26 K firmwares unpacked (fully or partially)
  - ↓ Files extraction
- 1.7 M files after unpacking

Firmware analysis options!

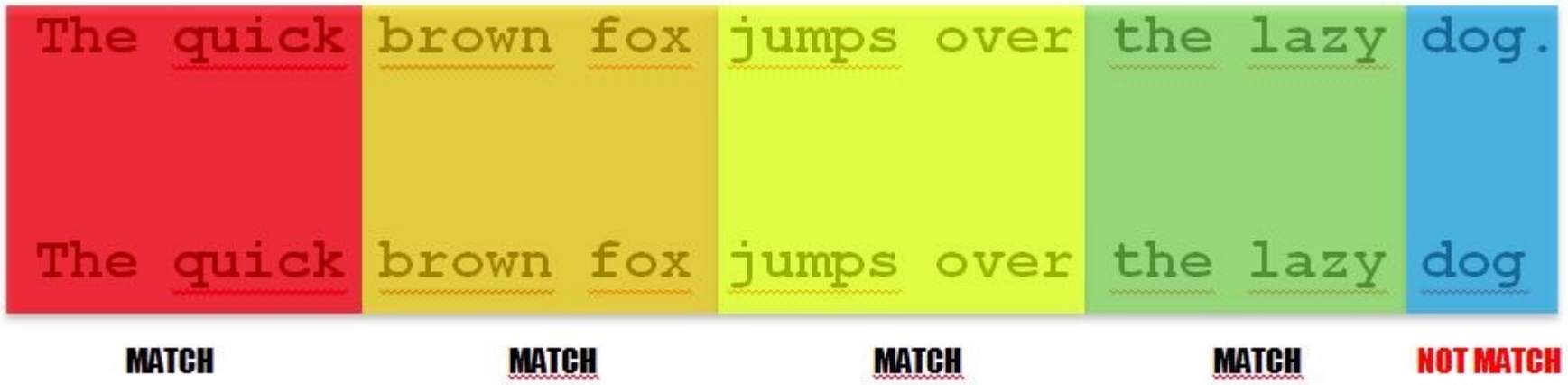


Simple static analysis

- Misconfiguration
  - Web-server configs, Code repositories
- Credentials
  - Weak/Default/Hard-coded
- Data enrichment
  - Versions → Software packages
  - Keywords → Known problems (e.g., telnet, shell, UART, backdoor)
- Correlation and clustering
  - Based on: Fuzzy hashes, Private SSL keys, Credentials

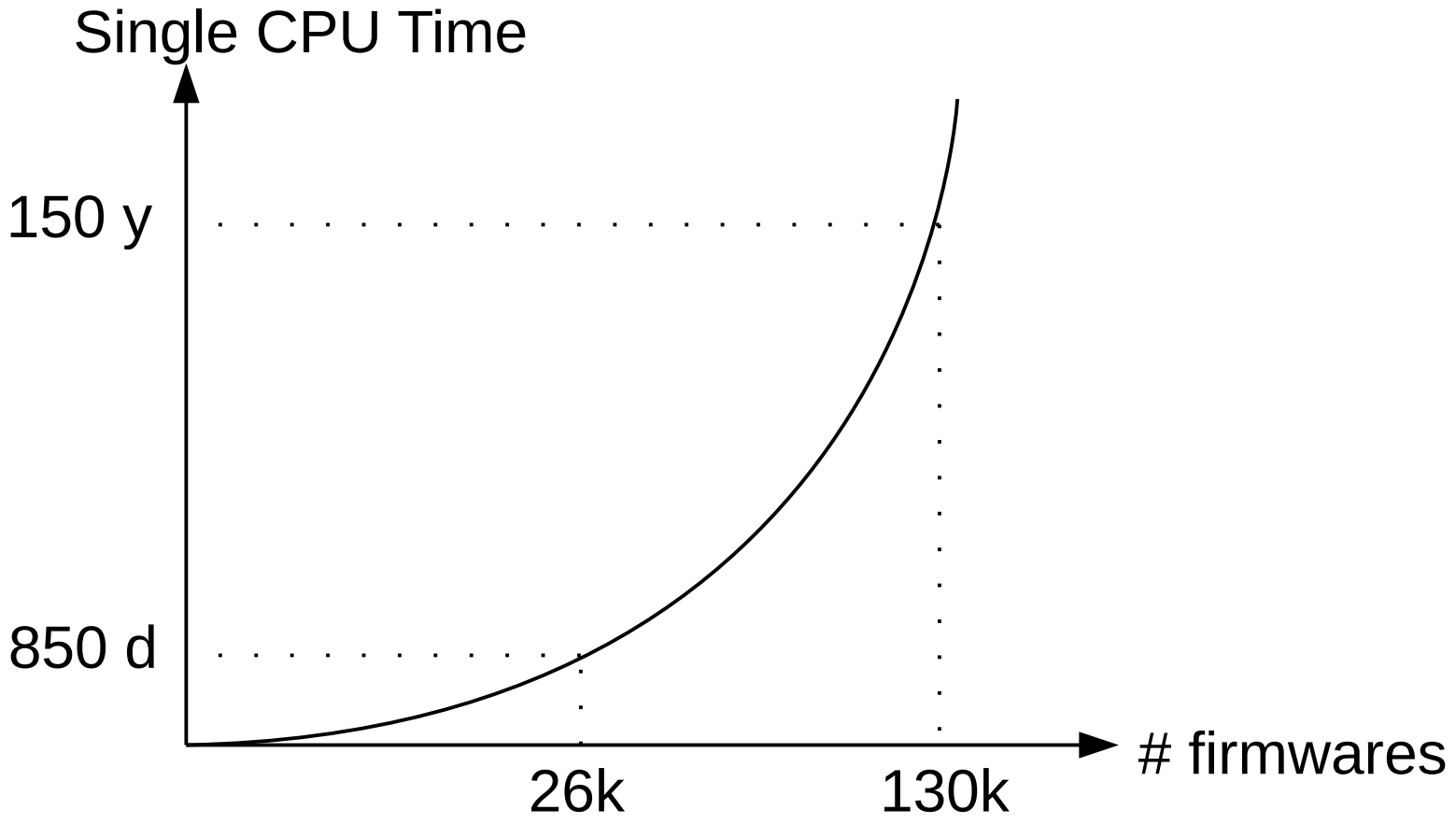
Fuzzy Hashing

- Fuzzy hash=similarity measure of two objects (e.g., files, streams)



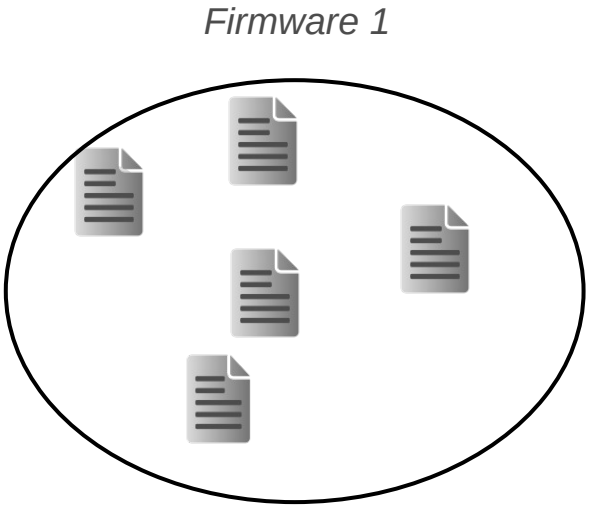
- Gives a “similarity index”

One to One fuzzy hash coparison



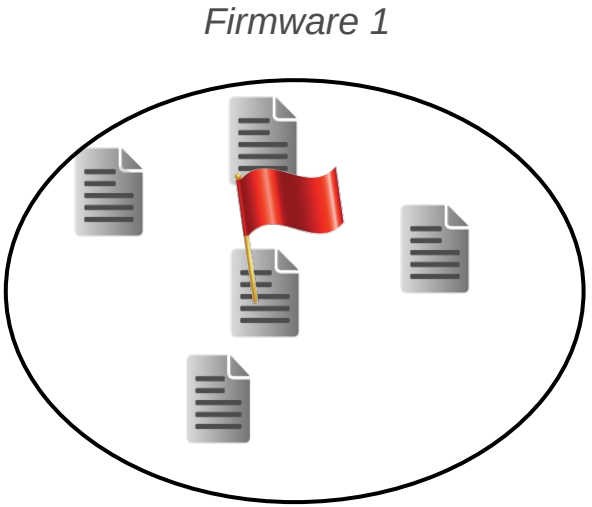
Example: Correlation

- Correlation via fuzzy-hashes (ssdeep, sdhash)
  - E.g., Vulnerability Propagation



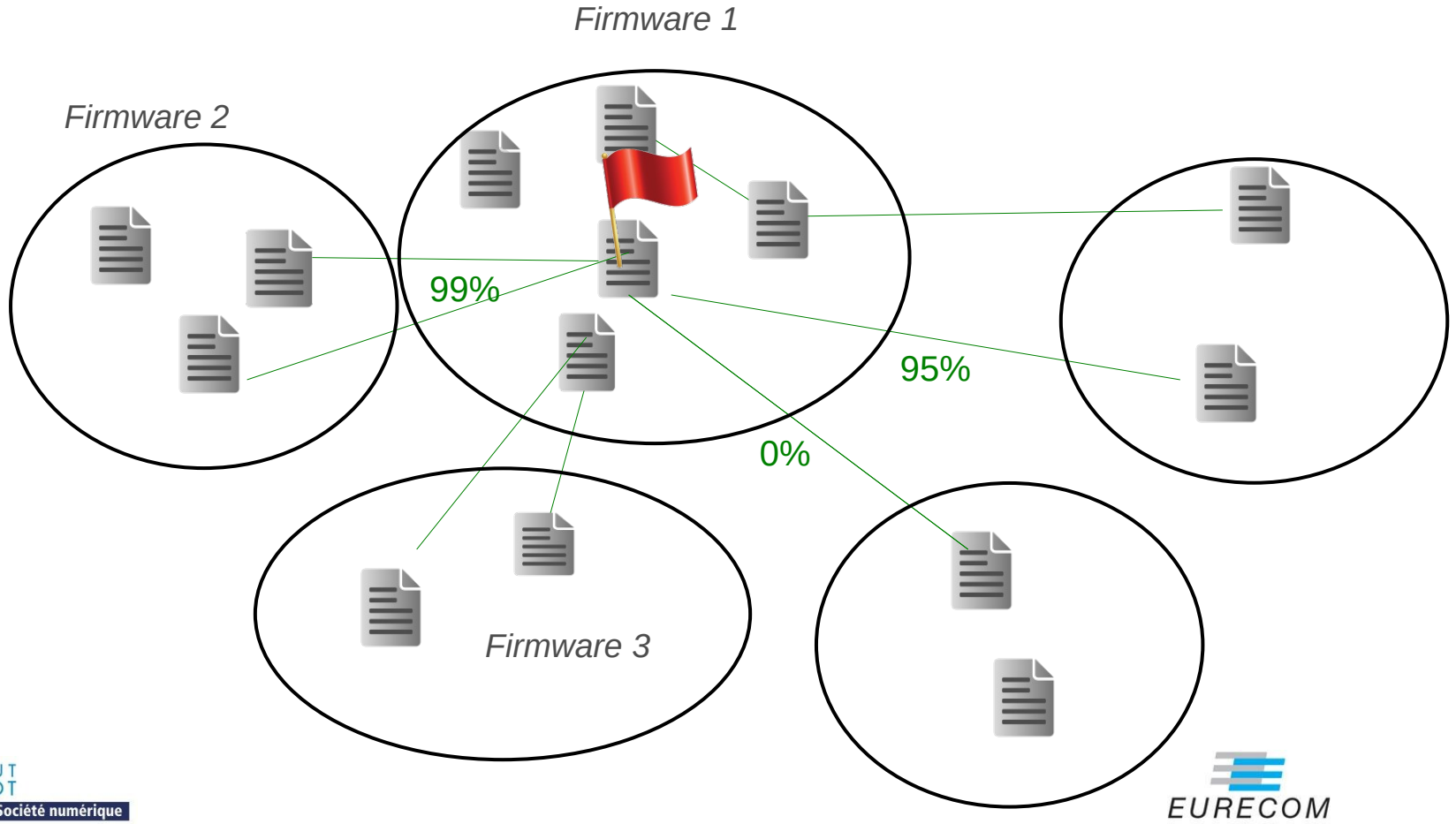
Example: Correlation

- Correlation via fuzzy-hashes (ssdeep, sdhash)
  - E.g., Vulnerability Propagation



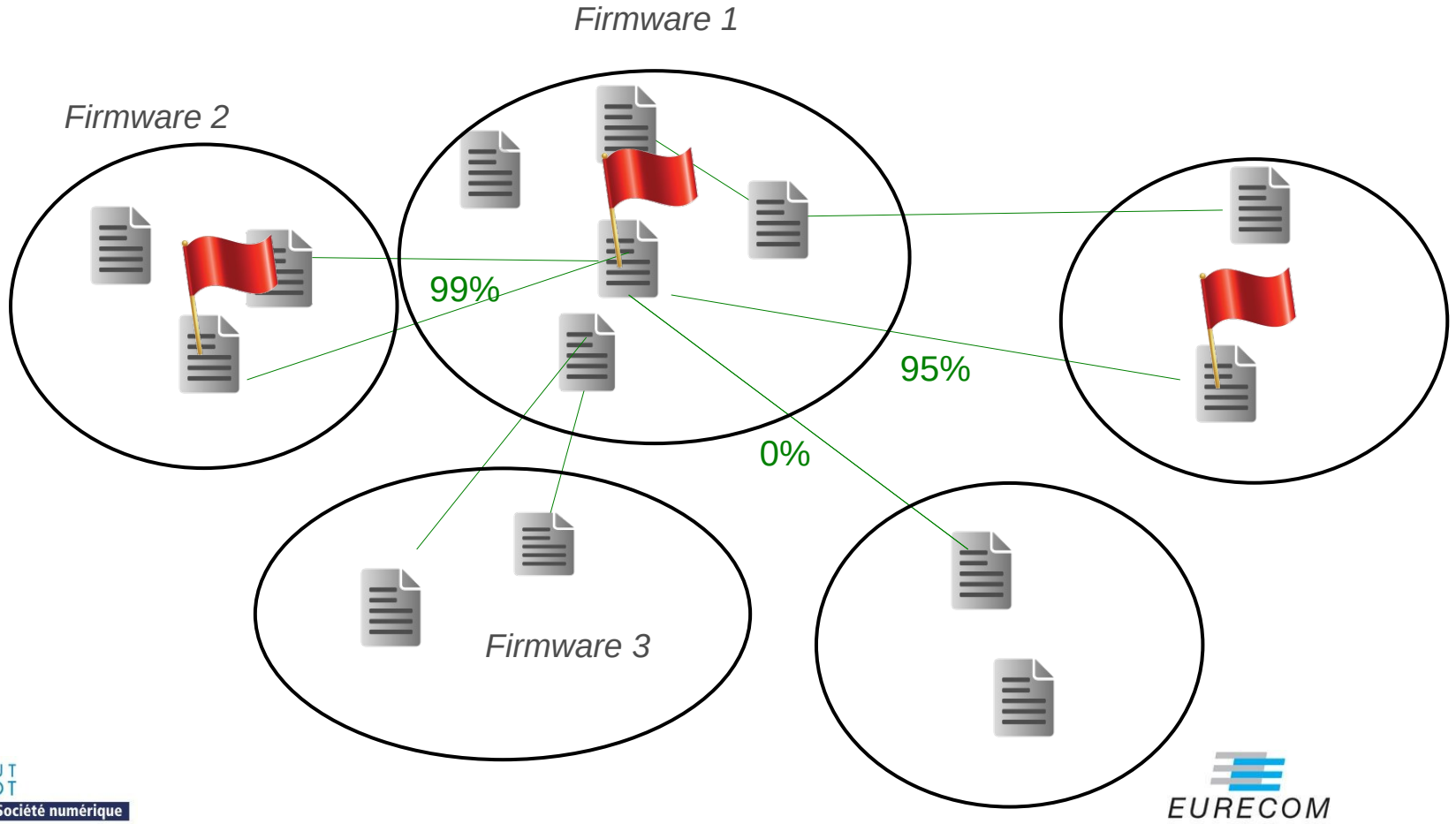
Example: Correlation

- Correlation via fuzzy-hashes (ssdeep, sdhash)
  - E.g., Vulnerability Propagation



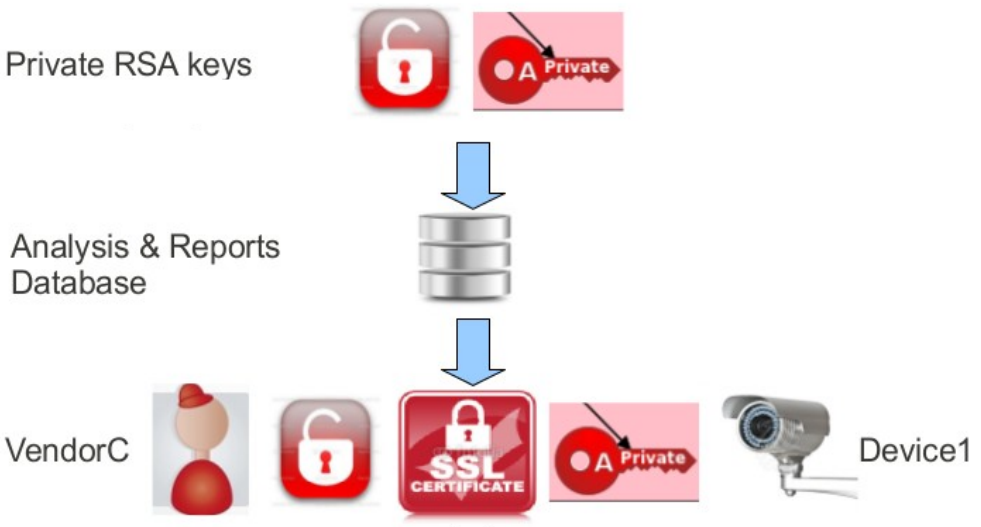
Example: Correlation

- Correlation via fuzzy-hashes (ssdeep, sdhash)
  - E.g., Vulnerability Propagation



RSA Keys

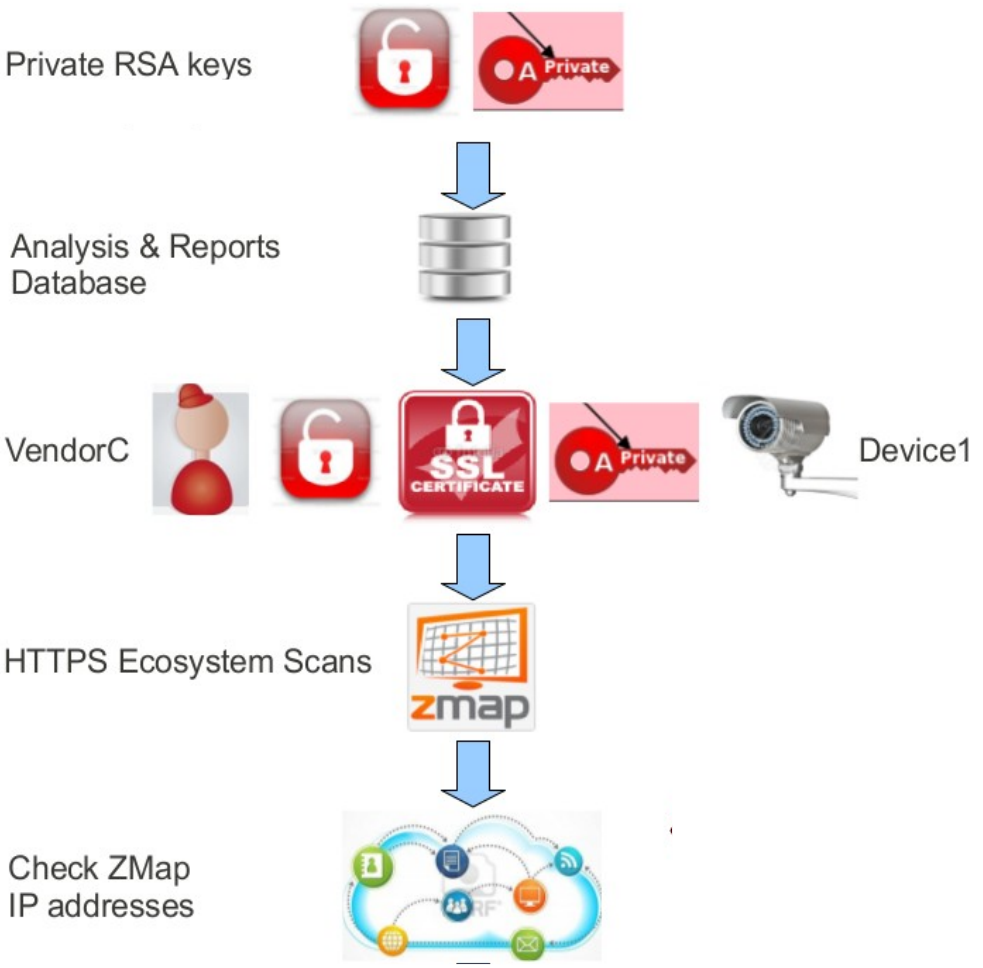
- SSL keys correlation vulnerability propagation



RSA Keys

- SSL keys correlation vulnerability propagation

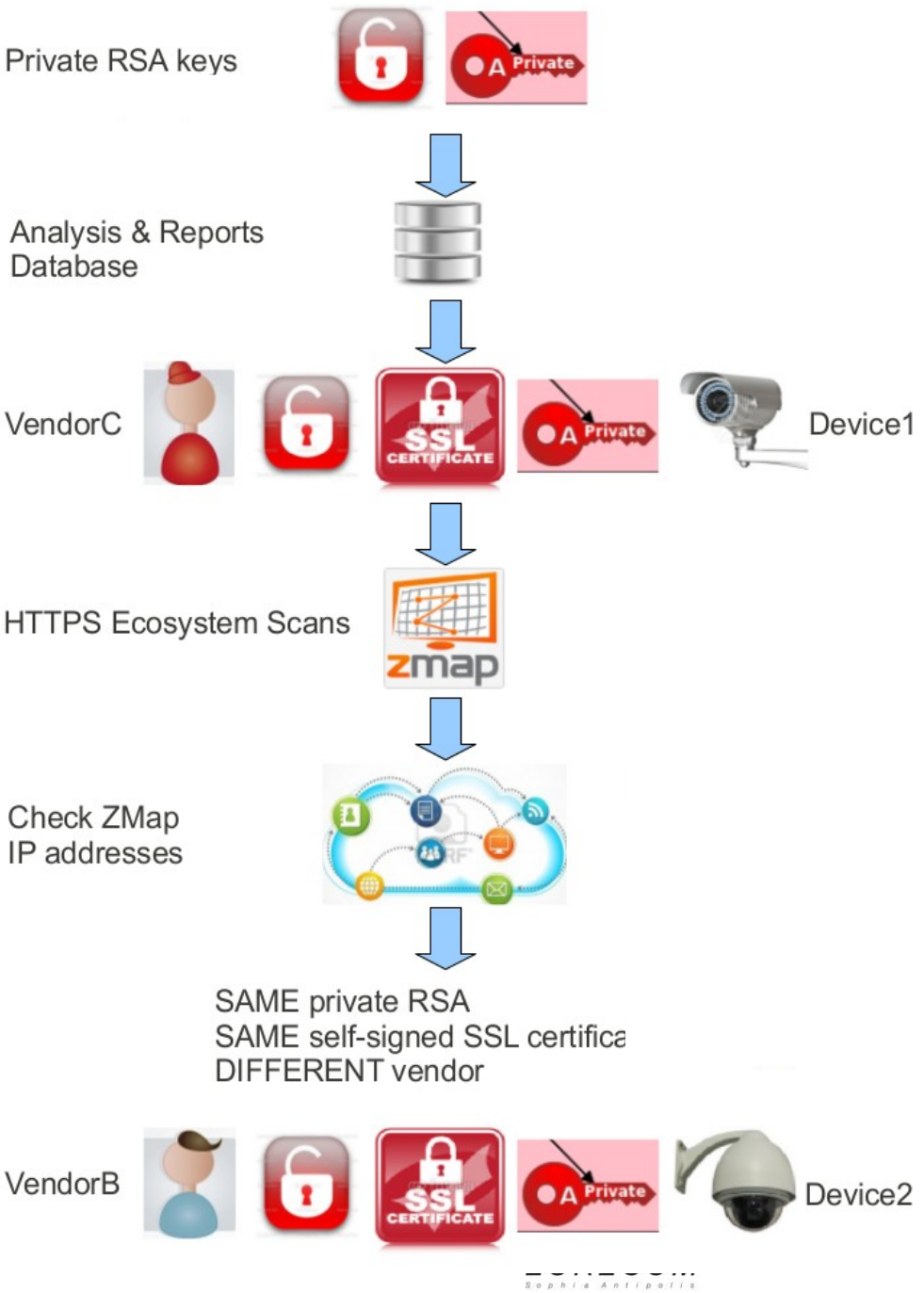
- 1 RSA private key: 30,000 vulnerable devices online



RSA Keys

- SSL keys correlation vulnerability propagation

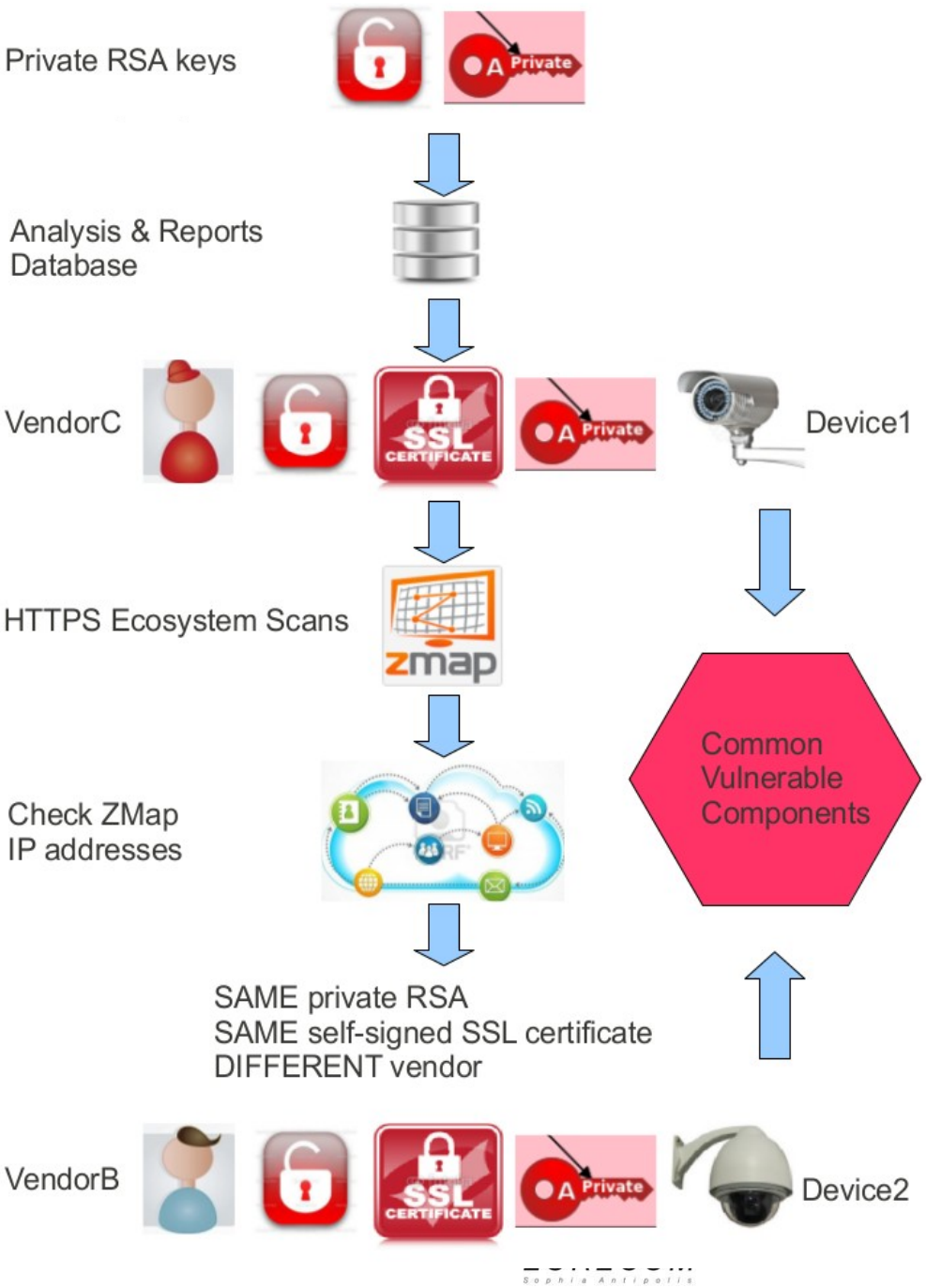
- 1 RSA private key: 30,000 vulnerable devices online
- Not all the same brand



RSA Keys

- SSL keys correlation vulnerability propagation

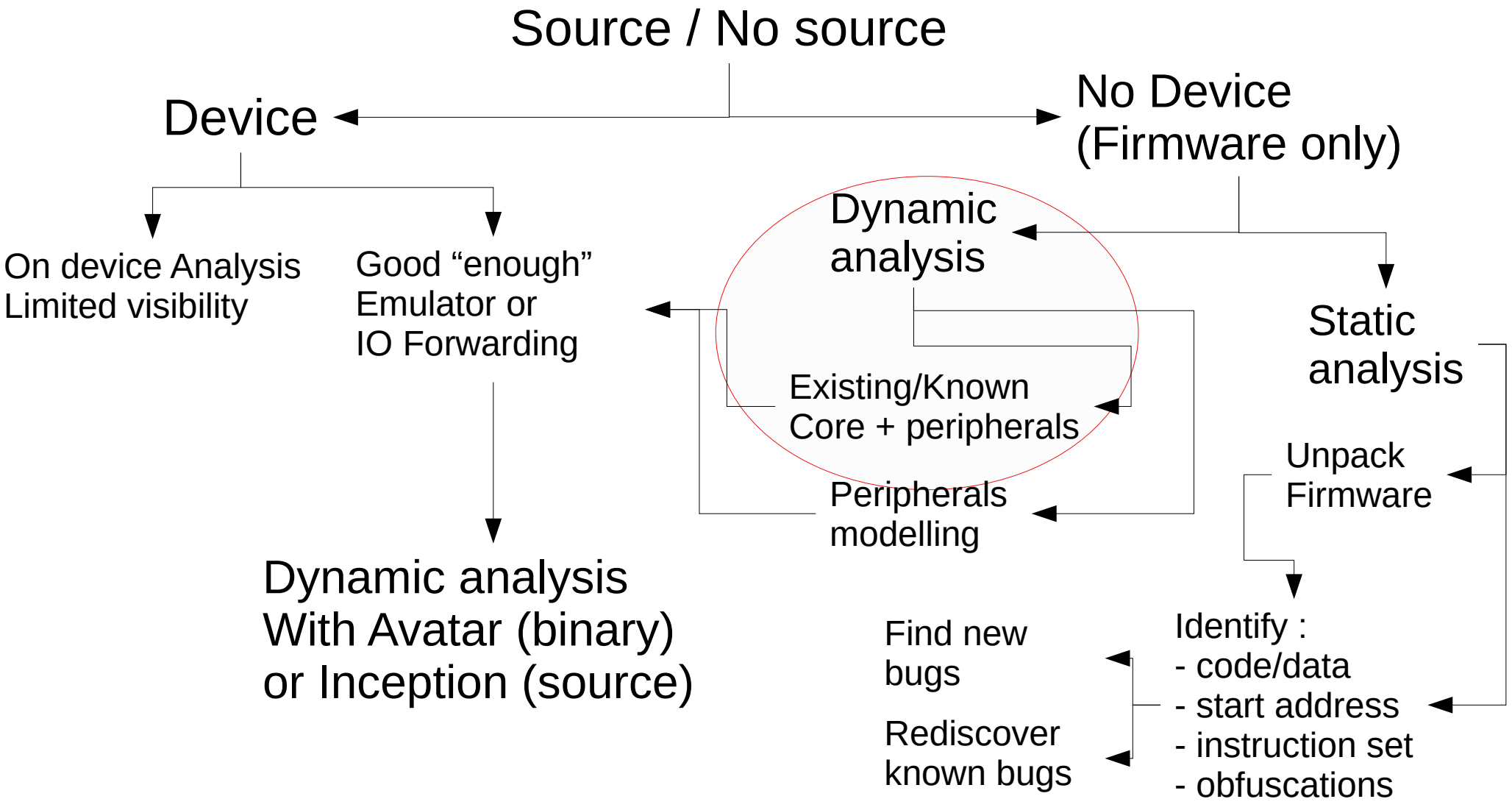
- 1 RSA private key: 30,000 vulnerable devices online
- Not all the same brand



Results: Summary

- 38 new vulnerabilities (CVE)
- Correlated them to 140 K vulnerable online devices
- 693 firmware files affected by at least one vulnerability

Firmware analysis options!



What else can we test in those firmware images?

- High exposure?
- Often privileged?
- Hard to secure?
- Often custom

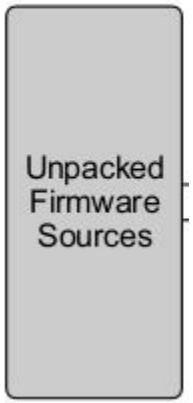
Web interfaces !

“Automated Dynamic Firmware Analysis at Scale: A Case Study on Embedded Web Interfaces”, A. Costin, A. Zarras, A. Francillon, **ACM AsiaCCS** 2016

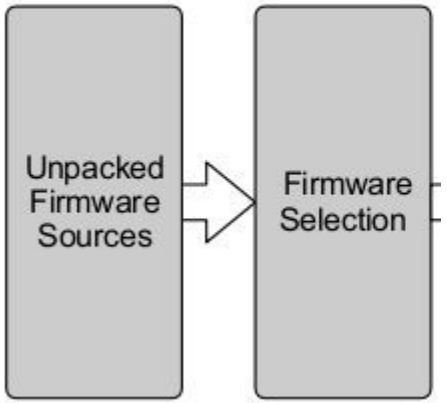
The dynamic analysis approach

- Many testing tools for dynamic analysis
  - Many pen testing tools…
  - Dedicated to find vulnerabilities in normal websites
  - Many are automated
  - Drawback: a lot of false positives
- Idea :
  - Emulate the unpacked firmware images
  - Launch the web interface
  - Use standard tools to test it

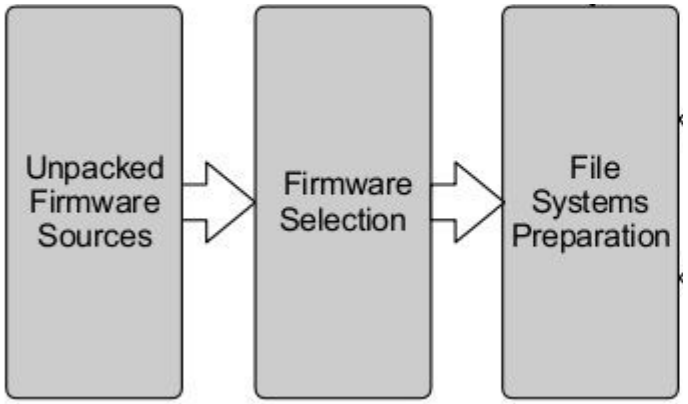
Our Framework



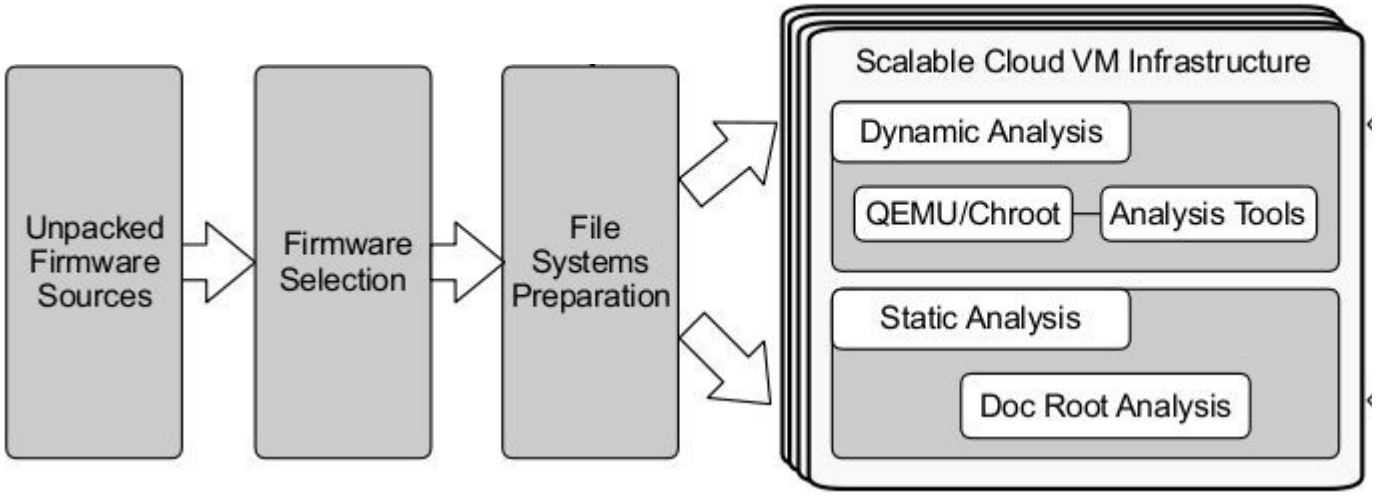
Our Framework



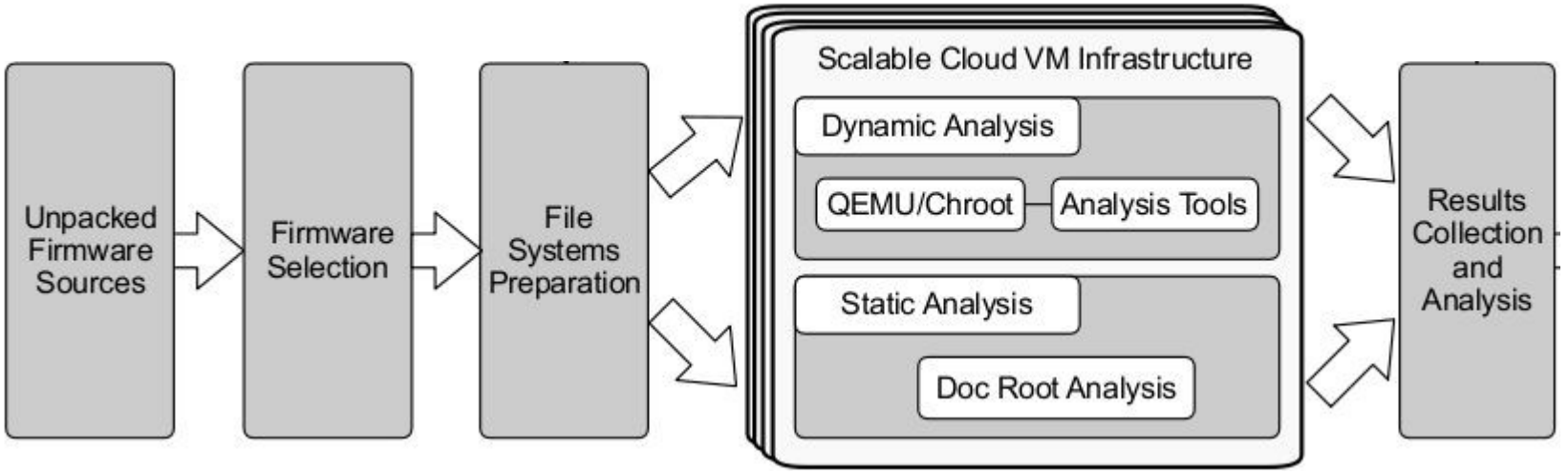
Our Framework



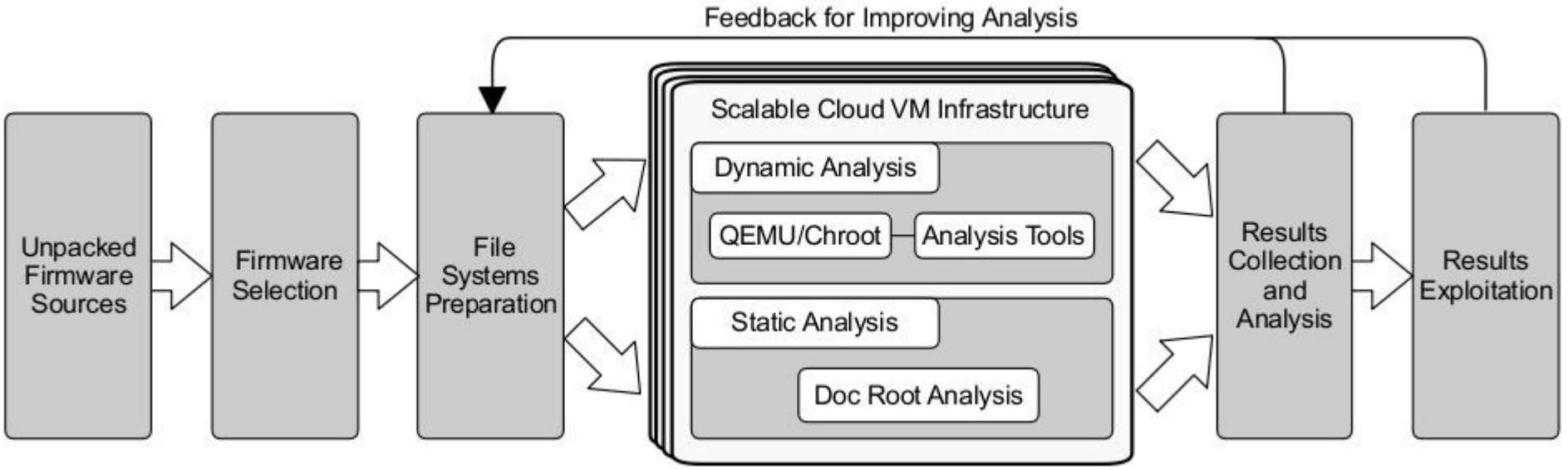
Our Framework



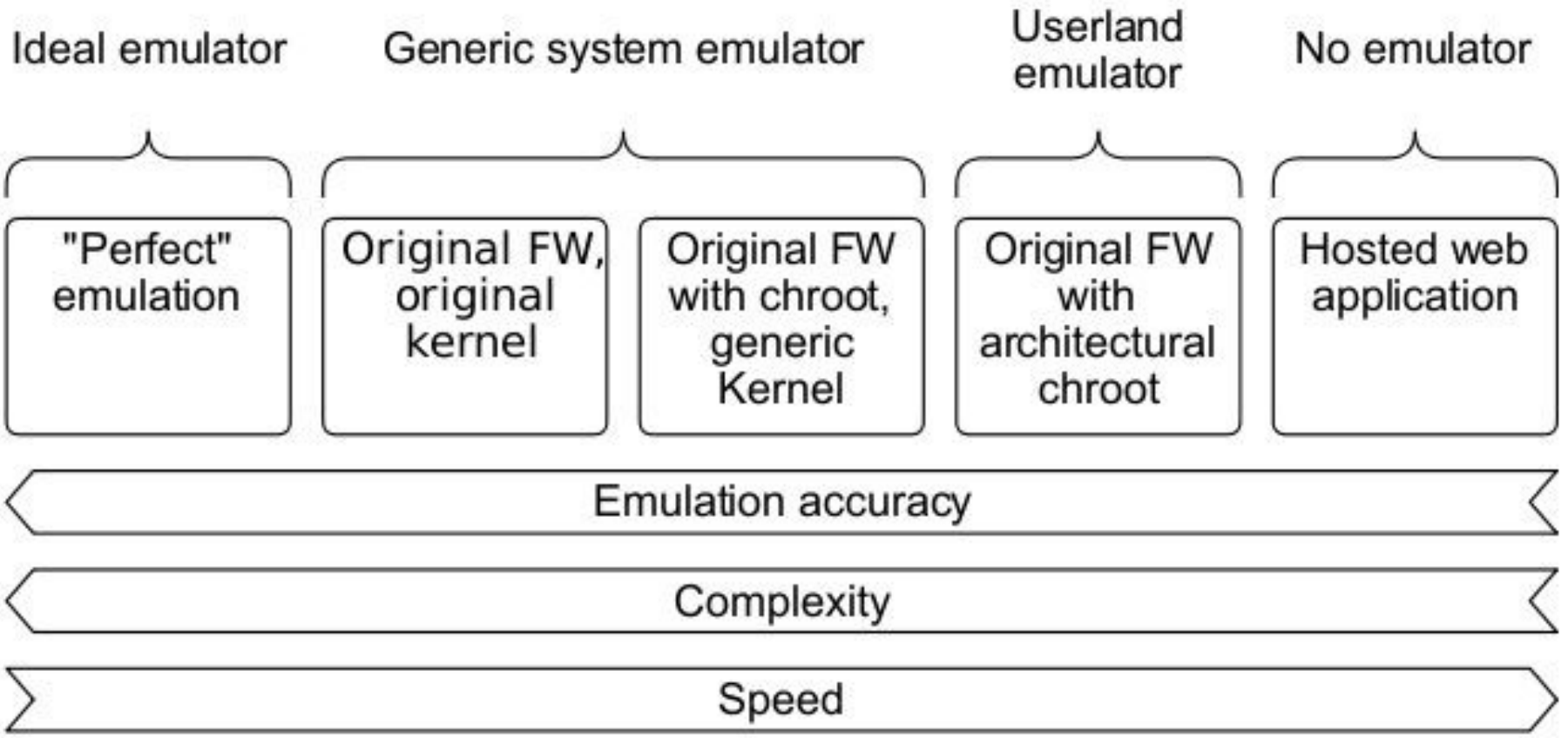
Our Framework



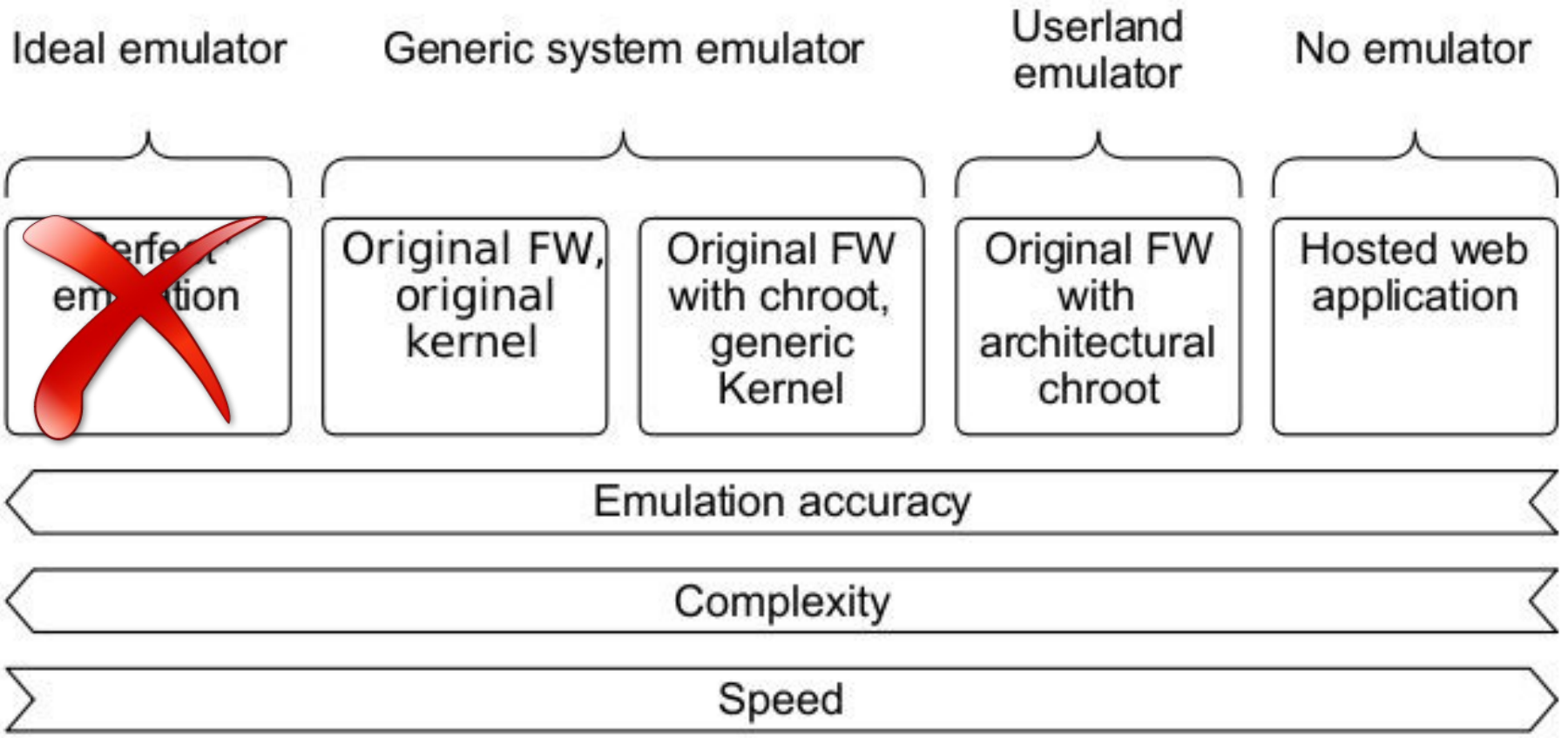
Our Framework



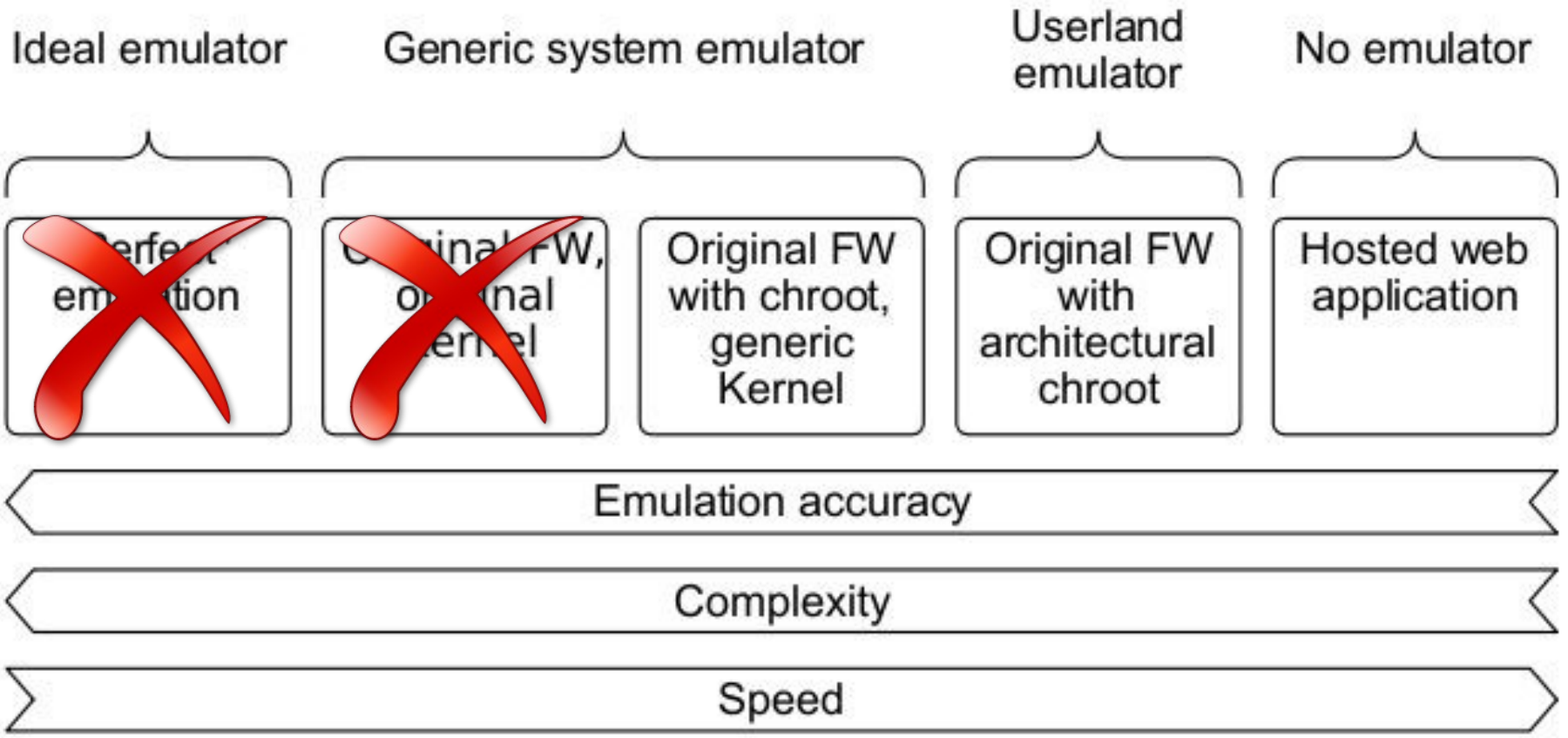
Identified Emulation Types



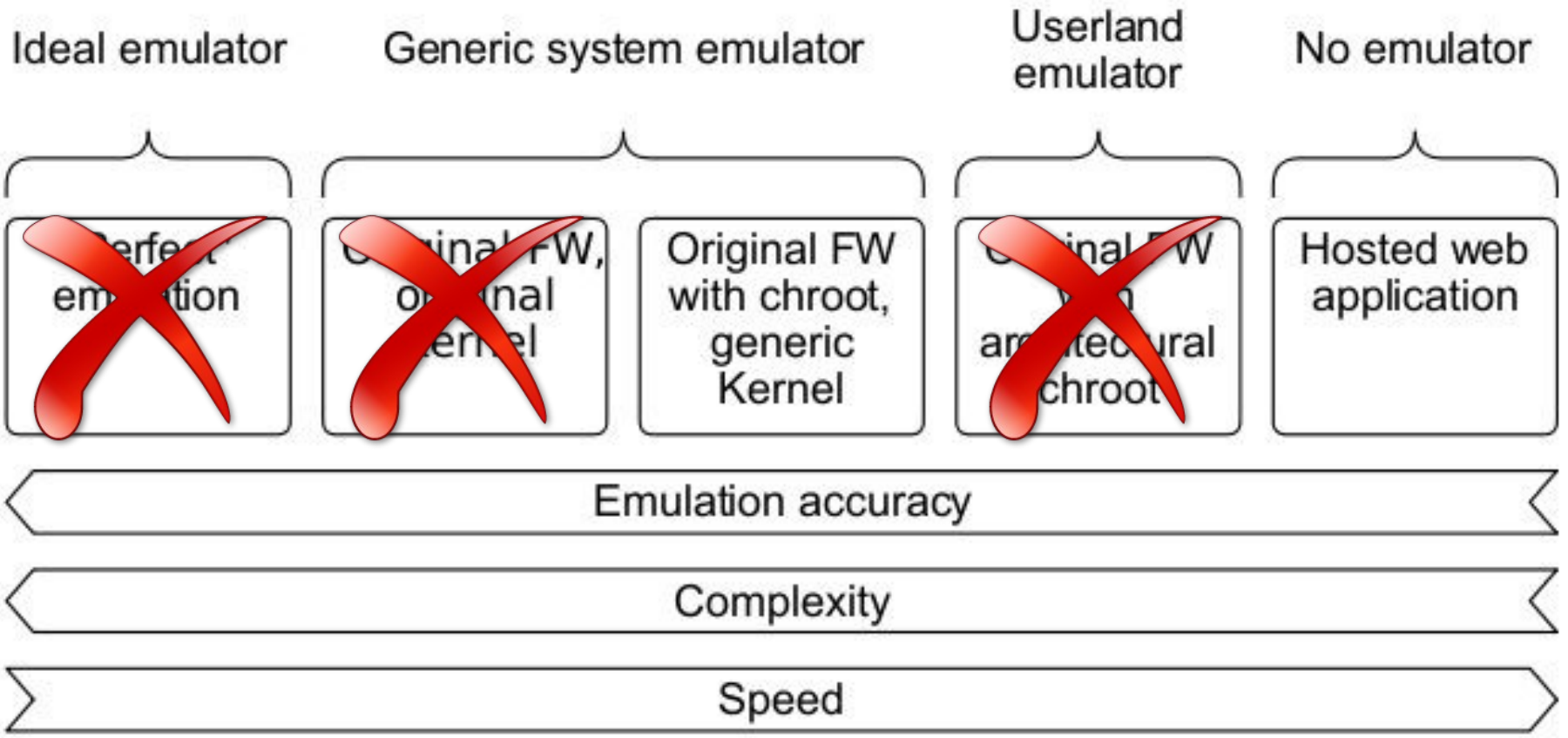
Identified Emulation Types



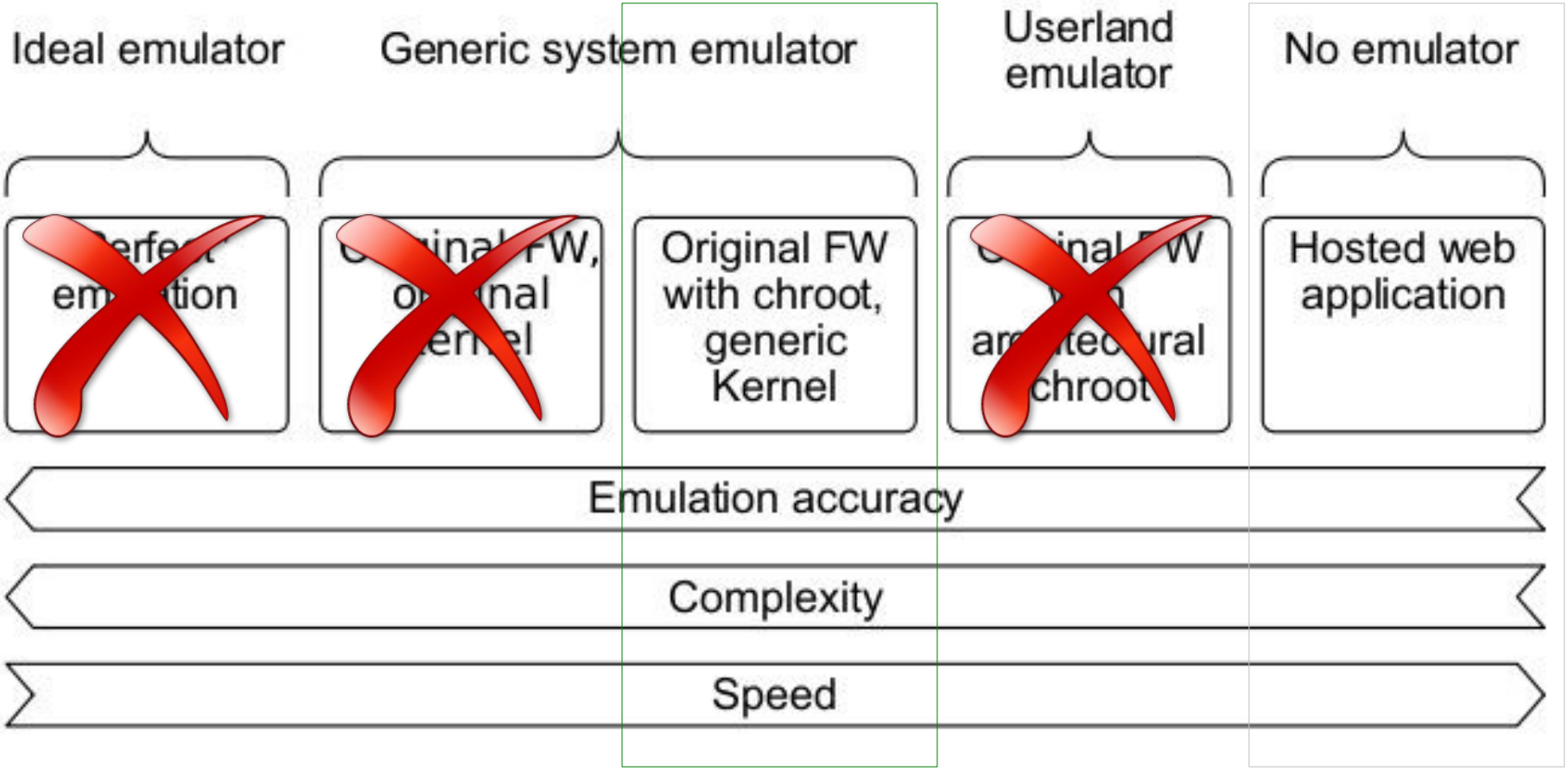
Identified Emulation Types



Identified Emulation Types



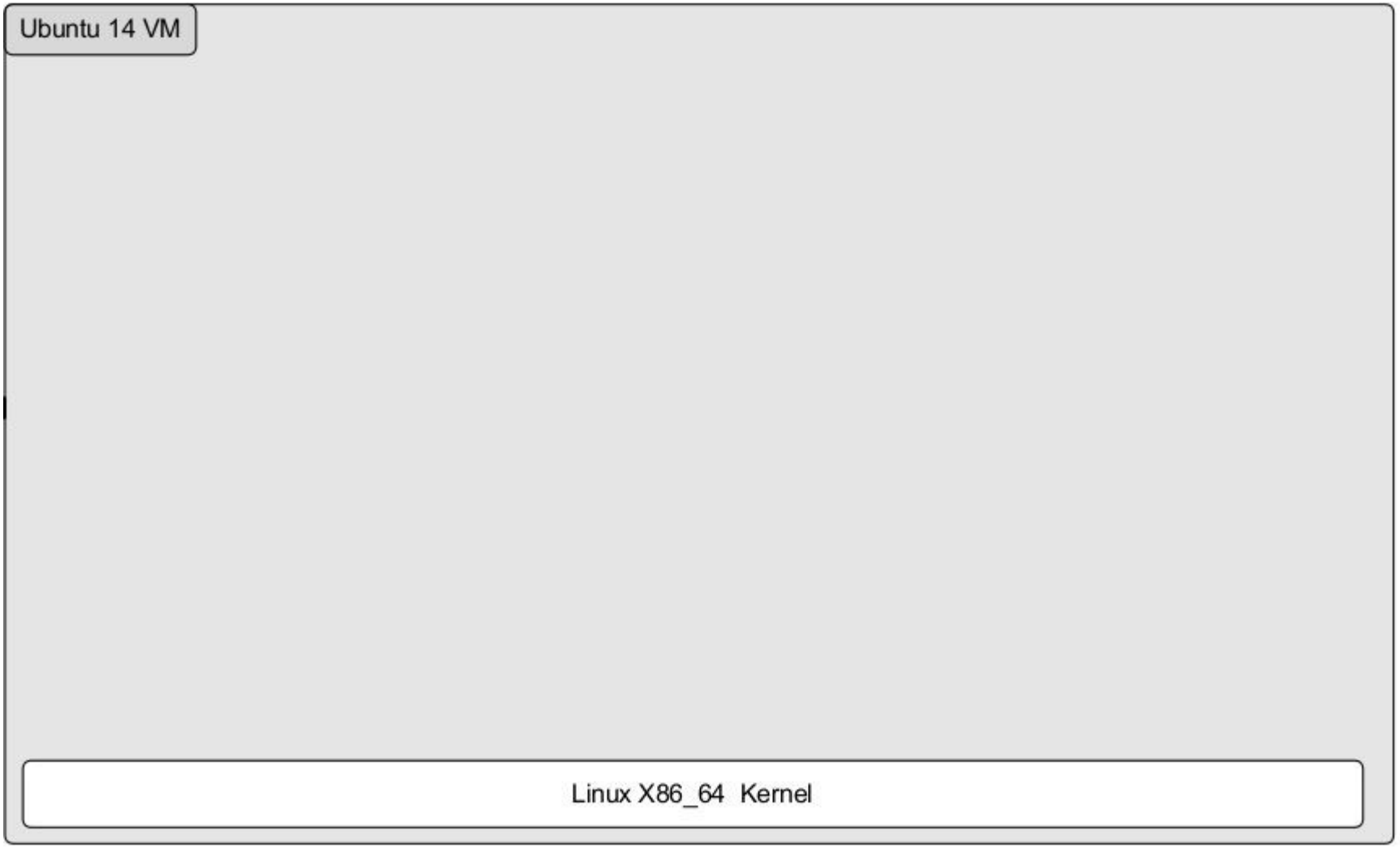
Identified Emulation Types



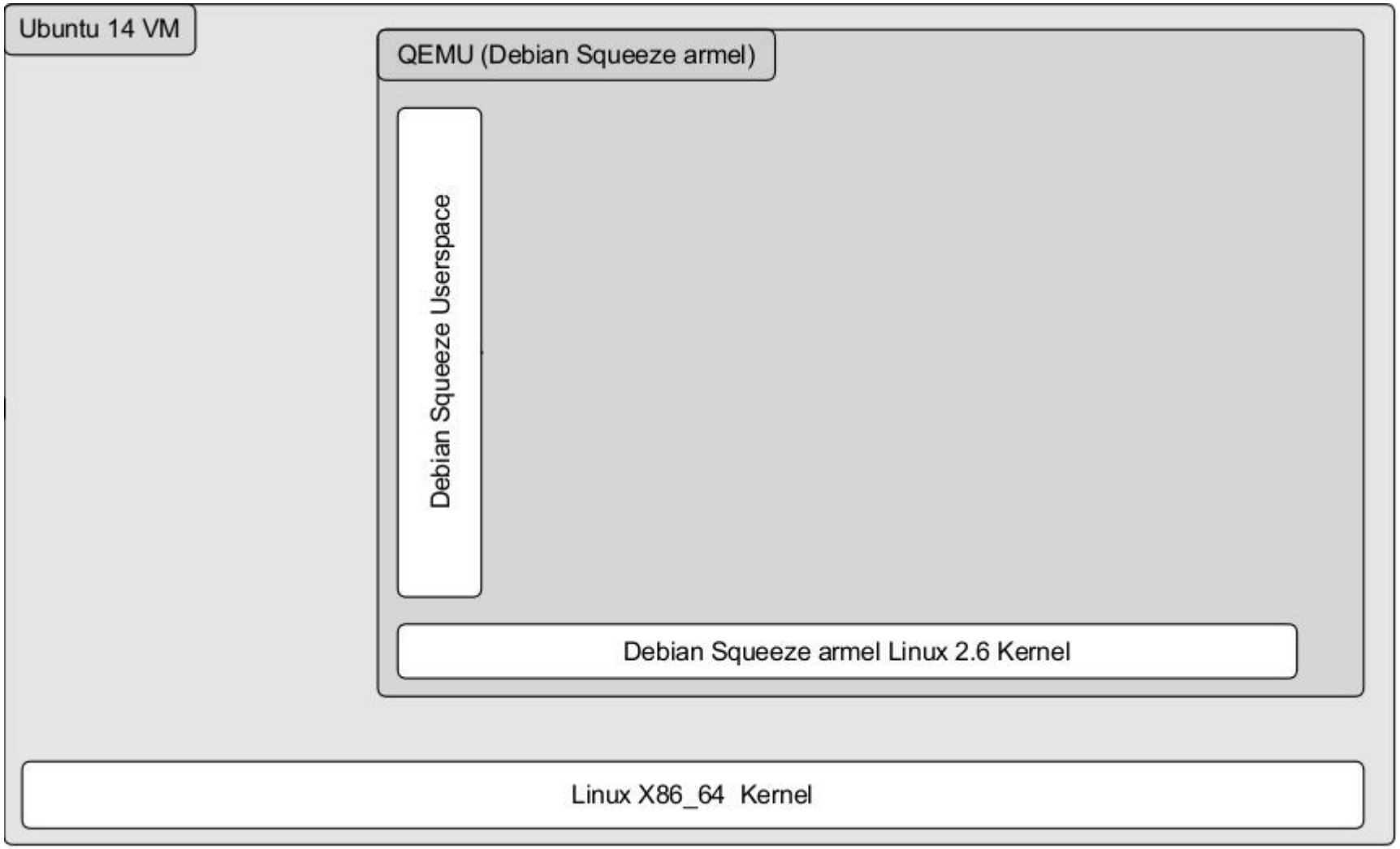
Hosted web interface

- In theory that would be a good idea, but
  - There are many embedded web servers
    - Which cannot be easily installed on a standard distribution
  - Web servers used are often customized
    - Or mostly custom
  - Native CGIs
  - Call some custom libraries (device configuration)
  - Still some relative success with this approach

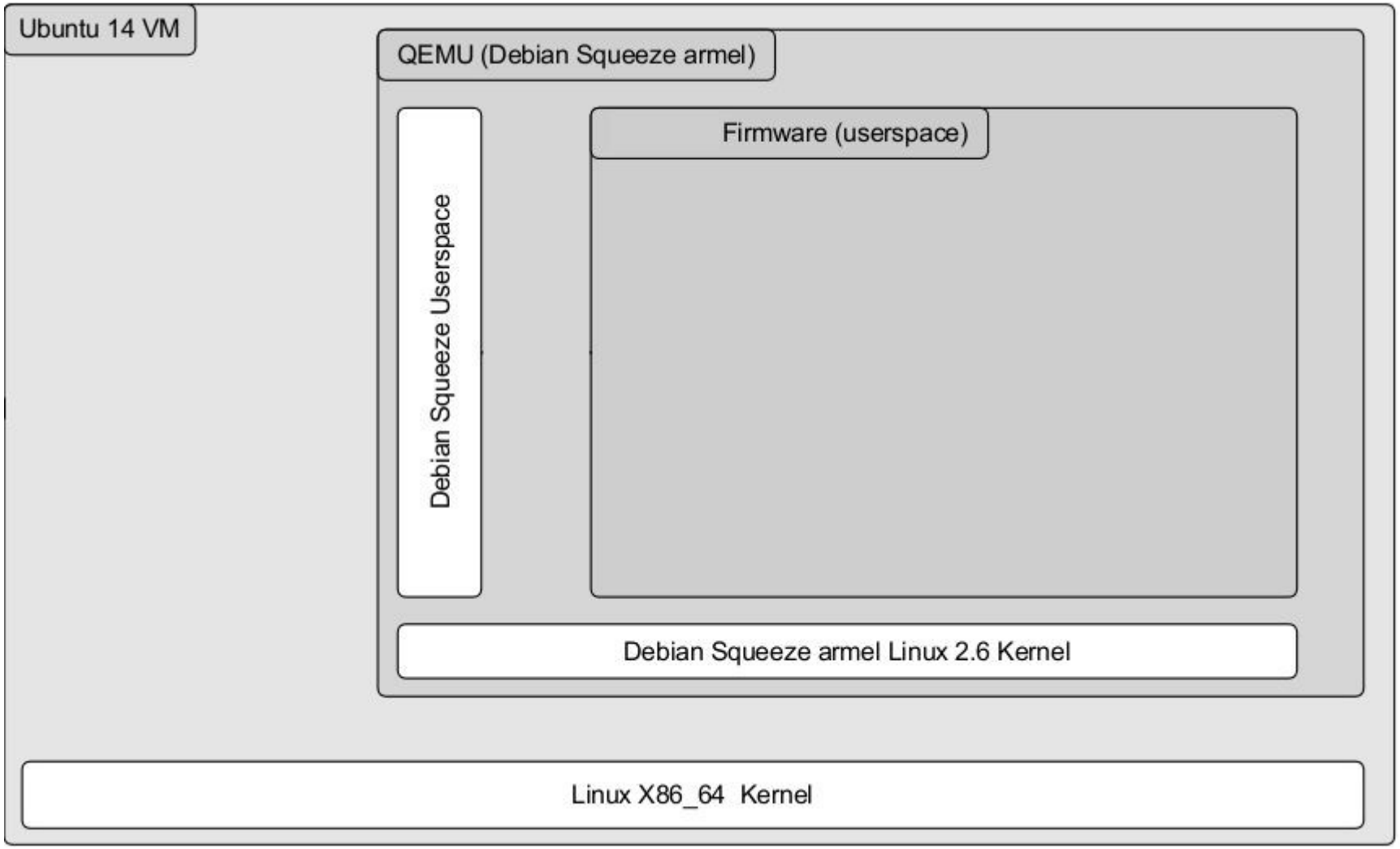
Proposed Emulation Technique



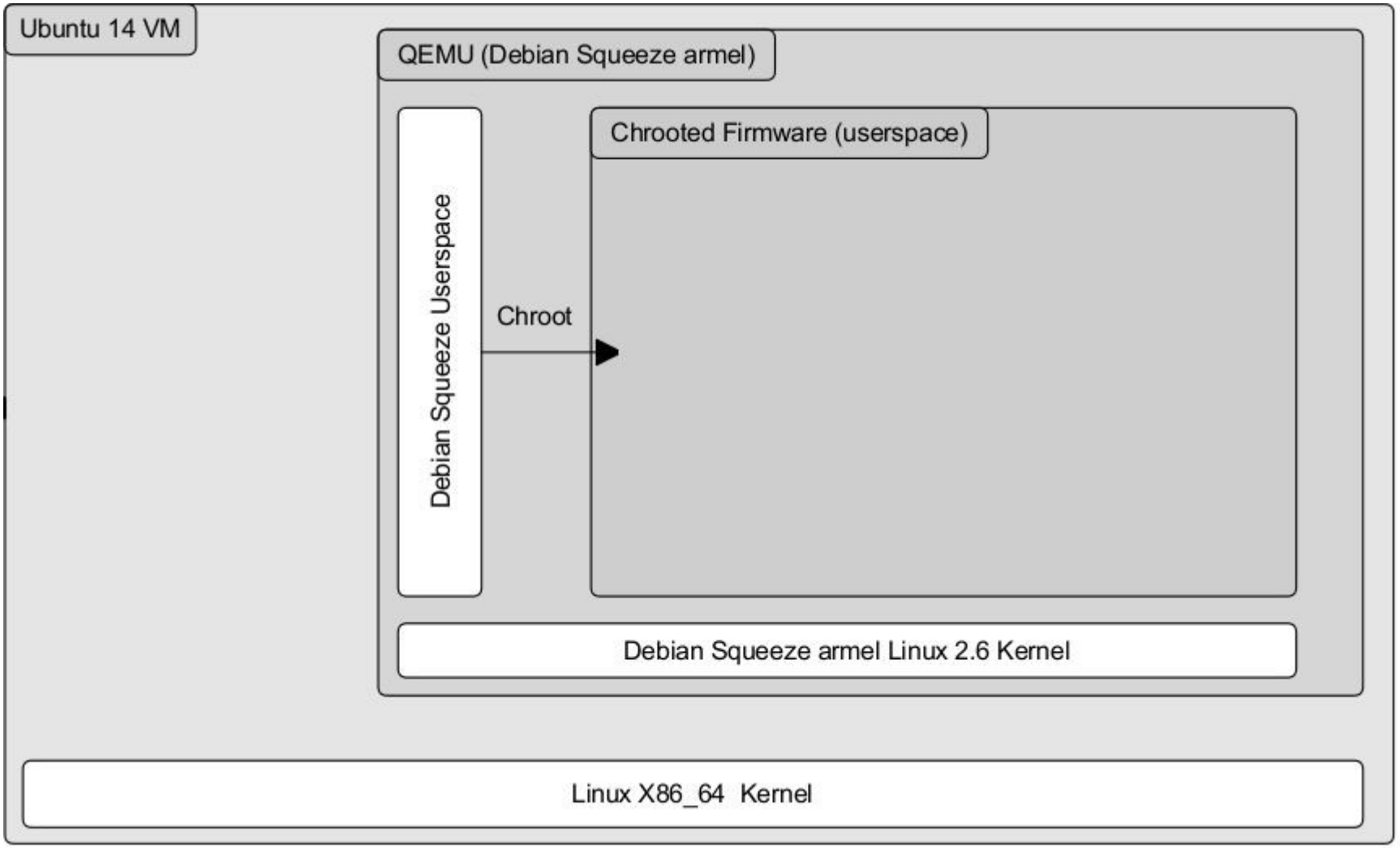
Proposed Emulation Technique



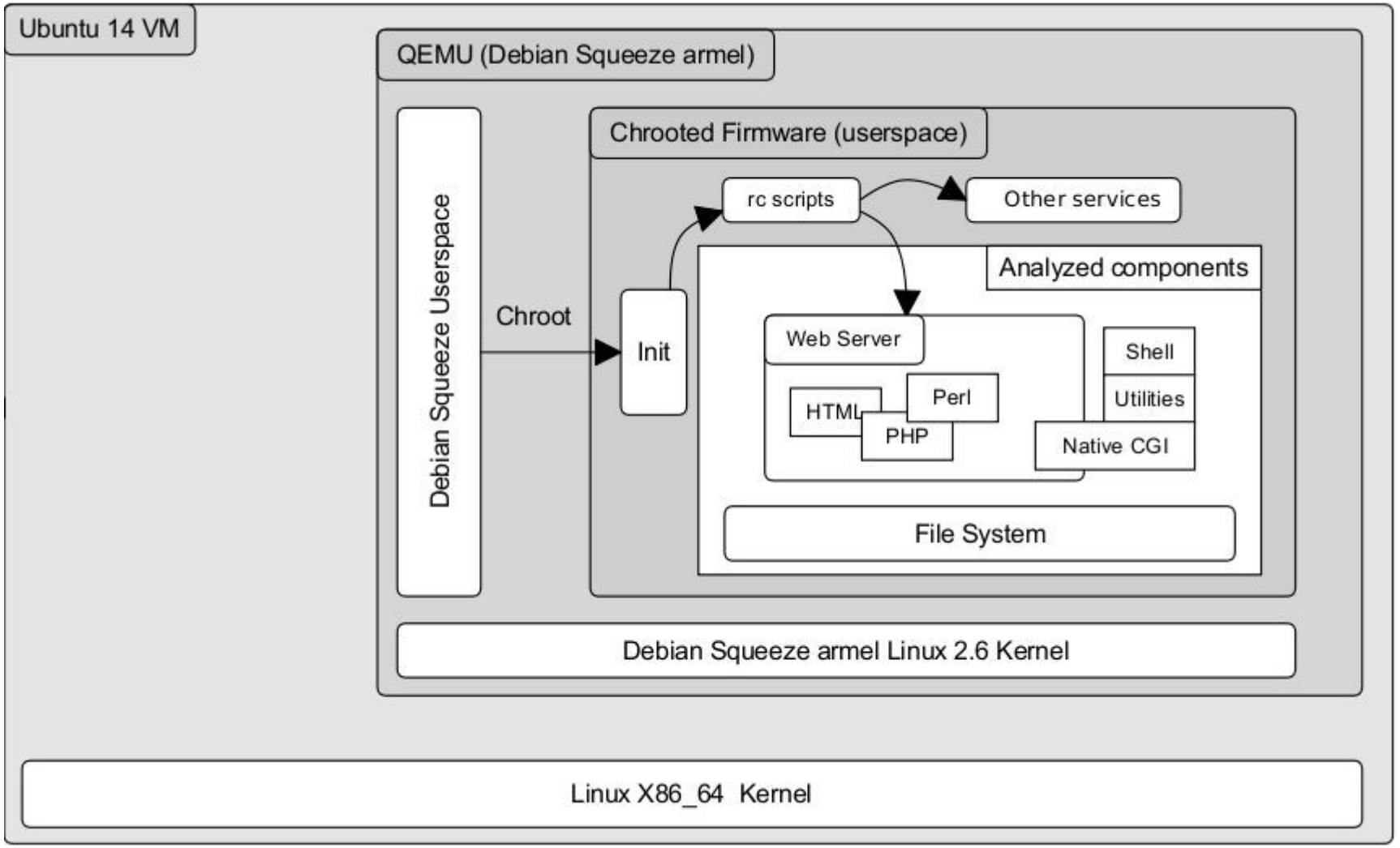
Proposed Emulation Technique



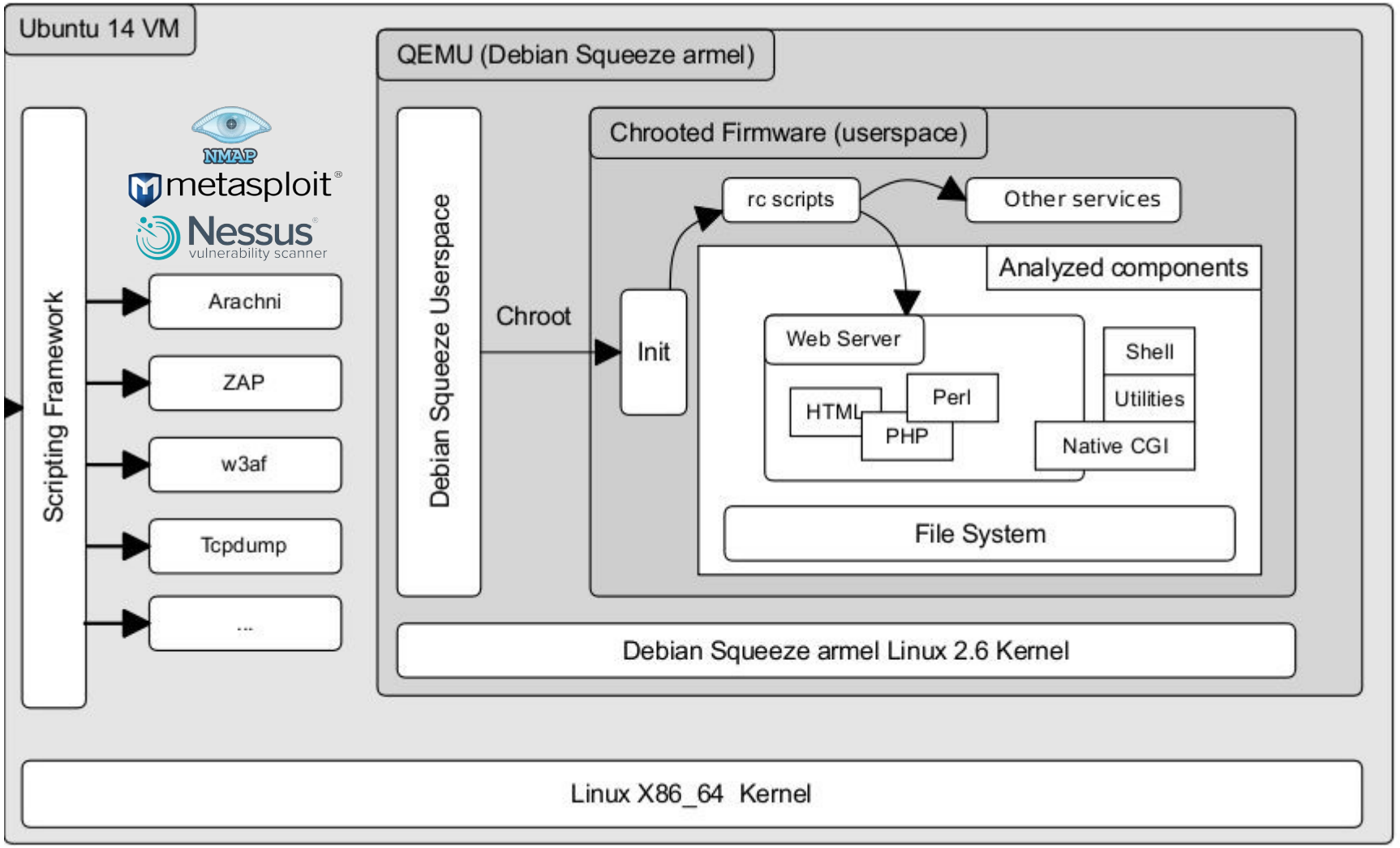
Proposed Emulation Technique



Proposed Emulation Technique



Proposed Emulation Technique



In summary

- We now can run the firmware in an emulator, on a generic kernel
- Works, but
  - We need to manually start daemons, init scripts
  - How does this differs from a real system ?
  - Fails often due to missing custom kernel module, or option

Similar work by CMU, see:
“Towards Automated Dynamic Analysis for Linux-based Embedded Firmware”, Chen, Egel, Woo, Brumley, NDSS 2017
https://github.com/firmadyne/firmadyne

Dataset and Processing

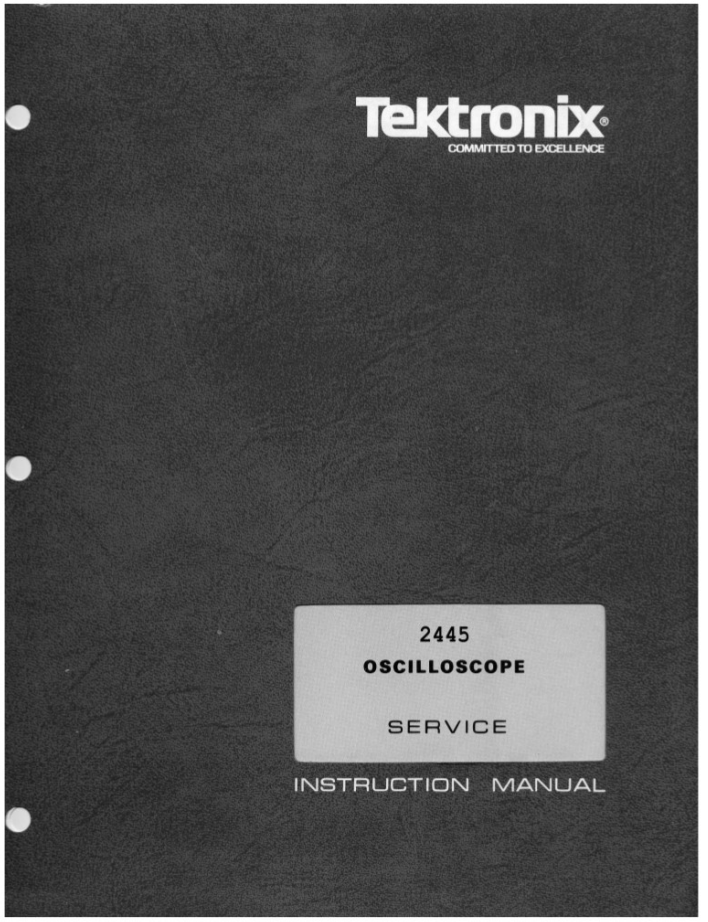
Dataset phase	# of FWs (unique)	# of vendors (unique)
Original dataset	1925	54
Candidates for chroot and web interface emulation	1580	49
Chroot OK	488	17
Web server OK	246	11
High impact vulnerabilities (static + dynamic)	185	13

Vulnerabilities by type

Vulnerability type	**# of issues**	**# of affected FWs**
Command execution	*51*	*21*
Cross-site scripting	*90*	*32*
CSRF	*84*	*37*
Sub-total HIGH impact	*225*	*45 (unique)*
Cookies w/o HttpOnly †	9	9
No X-Content-Type-Options †	2938	23
No X-Frame-Options †	2893	23
Backup files †	2	1
Application error info †	1	1
Sub-total low impact †	5843	23 (unique)
Total	**6068**	**58 (unique)**

Transparency problem

- In the good old times, hardware was documented
  - Tektronix 2445 Service manual 330 pages



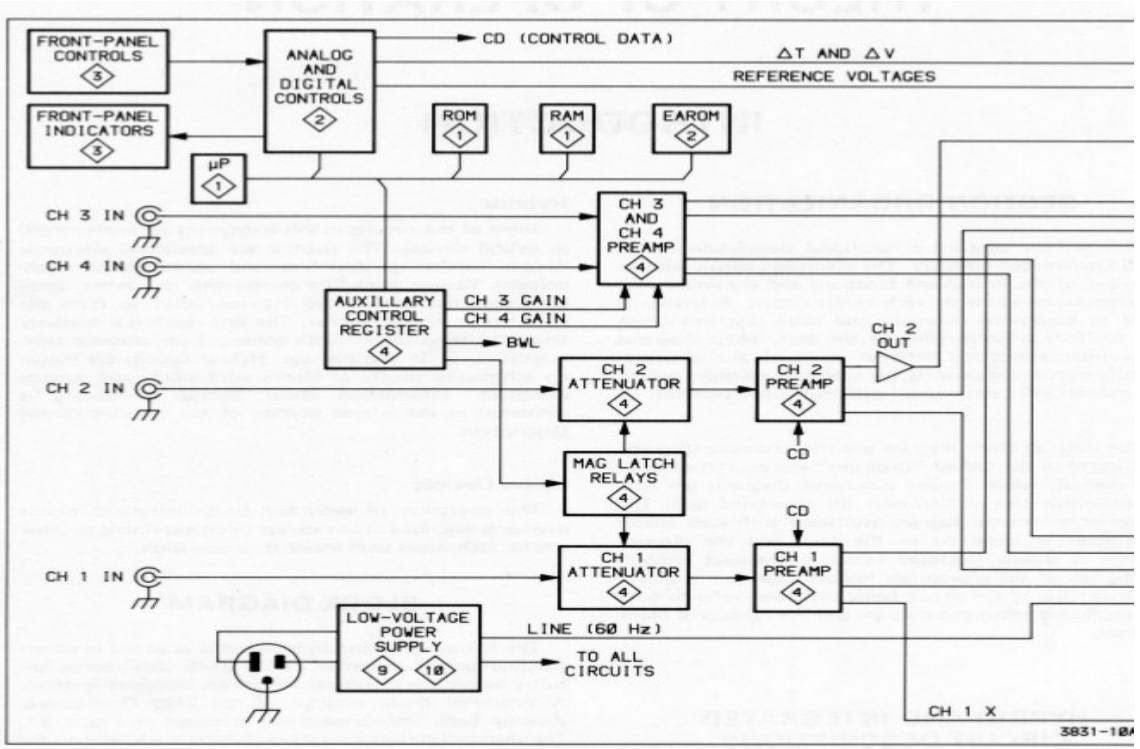


Figure 3-1. Block diagram.

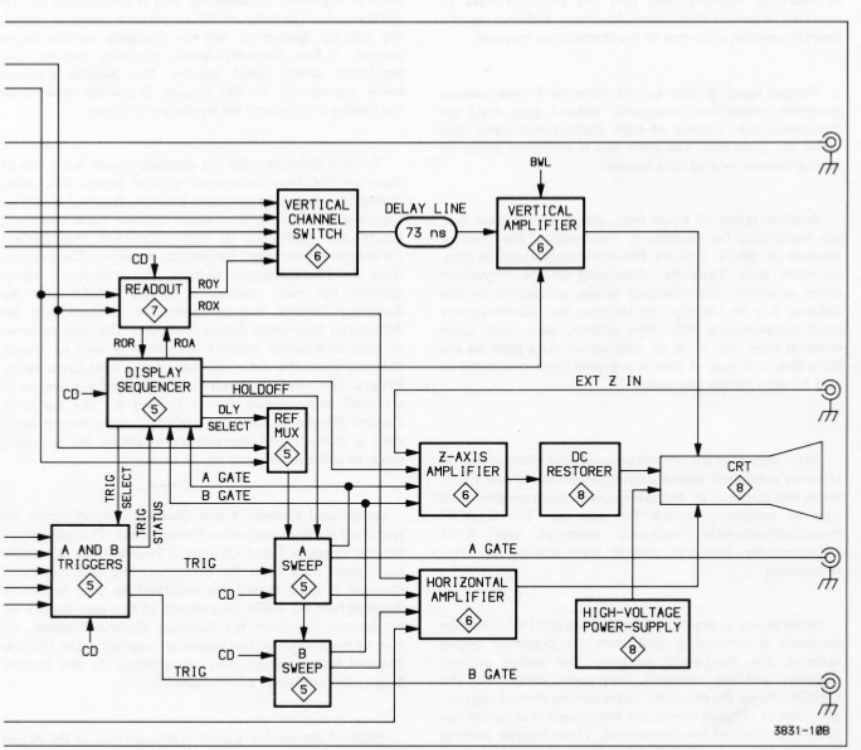
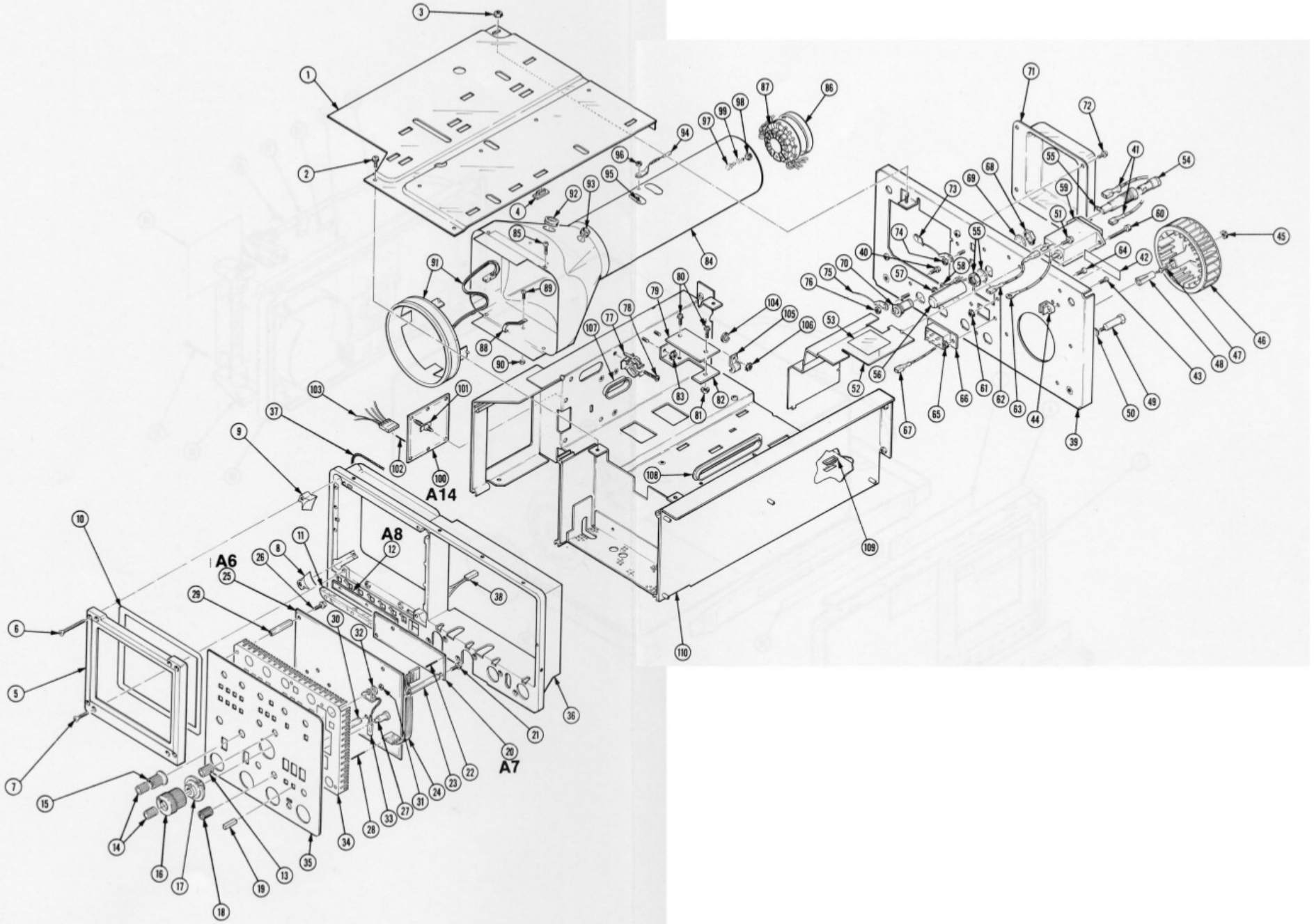
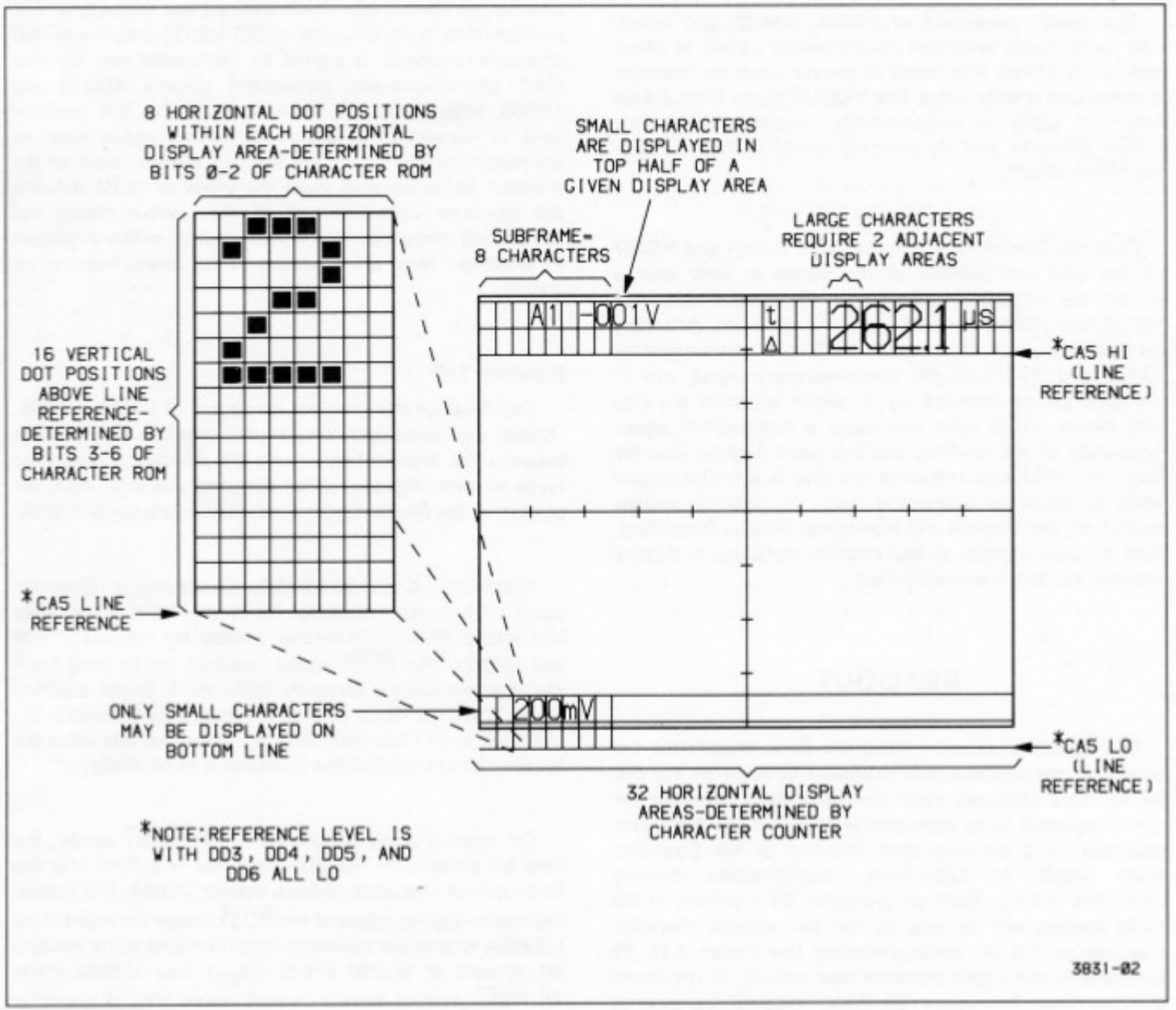


Figure 3-1. Block diagram (cont).


A14
A8
A6
A7


8 HORIZONTAL DOT POSITIONS WITHIN EACH HORIZONTAL DISPLAY AREA-DETERMINED BY BITS Ø-2 OF CHARACTER ROM
SMALL CHARACTERS ARE DISPLAYED IN TOP HALF OF A GIVEN DISPLAY AREA
SUBFRAME= 8 CHARACTERS
LARGE CHARACTERS REQUIRE 2 ADJACENT DISPLAY AREAS
16 VERTICAL DOT POSITIONS ABOVE LINE REFERENCE- DETERMINED BY BITS 3-6 OF CHARACTER ROM
A1 -001V
t Δ
262.1 µs
*CA5 HI (LINE REFERENCE)
*CA5 LINE REFERENCE
200mV
ONLY SMALL CHARACTERS MAY BE DISPLAYED ON BOTTOM LINE
*CA5 LO (LINE REFERENCE)
32 HORIZONTAL DISPLAY AREAS-DETERMINED BY CHARACTER COUNTER
*NOTE: REFERENCE LEVEL IS WITH DD3, DD4, DD5, AND DD6 ALL LO
3831-Ø2

Lack of transparency

Lack of transparency:

- Makes information asymmetry worse
  - Customers will not want to pay more ?
- Makes it harder to analyse devices
  - And detect compromise, analyse attacks
- Makes difficult to build secure systems
  - How to compare offers when no information is public
  - How to learn about security features ?
- Or is there something to hide? Backdoors ?

From an actual smartphone chip...

Dumped a bootloader in Mask ROM (No FBI, it's not an iPhone!)

```
ROM:FFFF23A0 loads_certificates                      ; CODE XREF: sub_FFFF24D4+28↓p
ROM:FFFF23A0                                         ; sub_FFFF2608+30↓p ...
ROM:FFFF23A0                 STMFD   SP!, {R4-R6,LR}
ROM:FFFF23A4                 LDR     R6, =0x8000605C ; address of CA Certificate in use
ROM:FFFF23A8                 MOV     R5, R0
ROM:FFFF23AC                 LDR     R1, [R6,#4]
ROM:FFFF23B0                 MOV     R0, #0
ROM:FFFF23B4                 STR     R1, [R5]
ROM:FFFF23B8                 LDR     R2, [R6]
ROM:FFFF23BC                 CMP     R2, #1          ; if cert == #1 ?
ROM:FFFF23C0                 MOVEQ   R0, #0          ; return 0
ROM:FFFF23C4                 LDMEQFD SP!, {R4-R6,PC}
ROM:FFFF23C8                 CMP     R1, #0
ROM:FFFF23CC                 MOVNE   R0, #1
ROM:FFFF23D0                 LDMNEFD SP!, {R4-R6,PC}
ROM:FFFF23D4                 MOV     R2, #0xB8000000
ROM:FFFF23D8                 LDR     R1, [R2,#0x950]
ROM:FFFF23DC                 AND     R1, R1, #0x1C0000 ; Bits 20:18 COM_GOV_SEL
ROM:FFFF23DC                                         ; Three fuses for majority vote encoding: 0 = Commercial, 1 =>
ROM:FFFF23DC                                         ; Government
ROM:FFFF23E0                 MOV     R1, R1,LSR#18
ROM:FFFF23E4                 CMP     R1, #3
ROM:FFFF23E8                 CMPNE   R1, #5
ROM:FFFF23EC                 CMPNE   R1, #6
ROM:FFFF23F0                 CMPNE   R1, #7
ROM:FFFF23F4                 LDREQ   R0, =certificate_GOV ; if 3/5/6/7 use certificate for government
ROM:FFFF23F8                 BEQ     loc_FFFF2434    ; store ROOT certificate  address
ROM:FFFF23FC                 LDR     R1, [R2,#0x938] ; SEC_BOOT_MODE
ROM:FFFF2400                 TST     R1, #1
ROM:FFFF2404                 BEQ     loc_FFFF243C
```

This talk

- Finding vulnerabilities in embedded devices
  - To secure them (or exploit them)
- What makes this a difficult task?

- Generally two approaches: Static or Dynamic
  - Both have advantages/drawbacks
  - We will mainly focus on dynamic analysis

Analyzing firmware images

1. Collect a large number of firmware images
2. Perform broad but simple static analysis
3. Correlate across firmwares

Many advantages:

- No intrusive online testing, no devices involved
- Scalable

But also many challenges

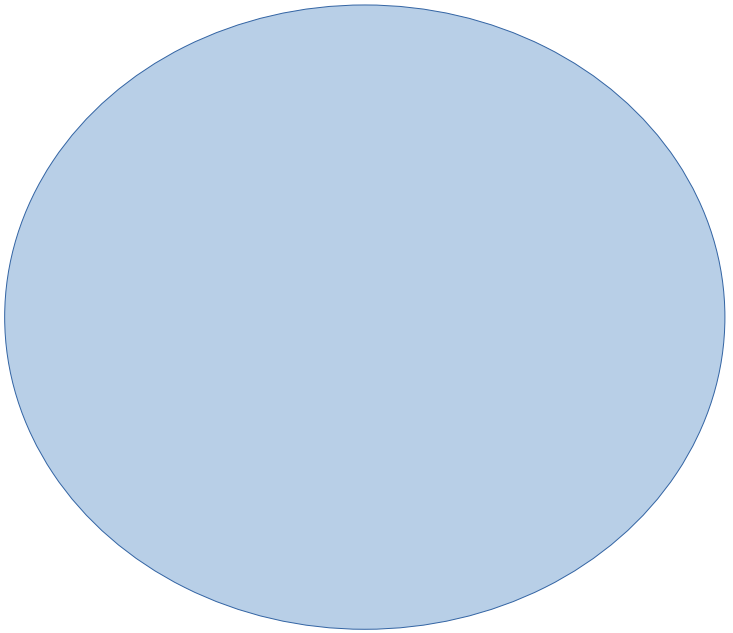
« A Large Scale Analysis of the Security of Embedded Firmwares »
Andrei Costin, Jonas Zaddach, Aurélien Francillon, Davide Balzarotti
USENIX Security 2015

Challenges

- Firmware identification (.exe/.ps/...)
- Firmware Unpacking
- Representative dataset
- Analysis, Scalability
- Results confirmation

Firmware identification

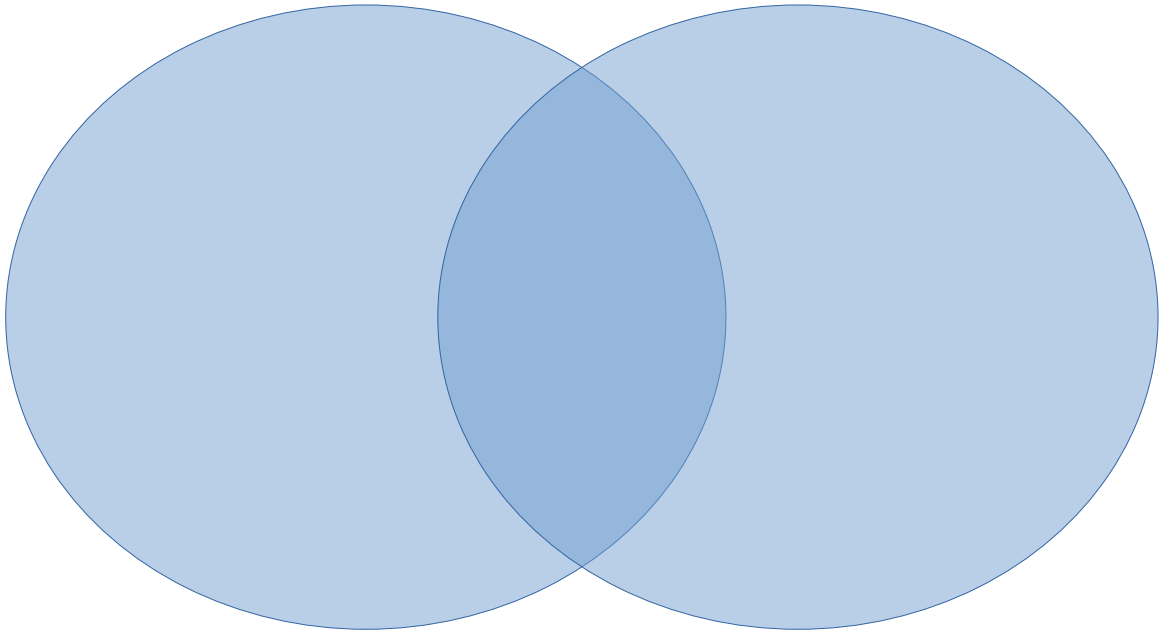
← Clearly a Firmware



Firmware identification

← Clearly a Firmware

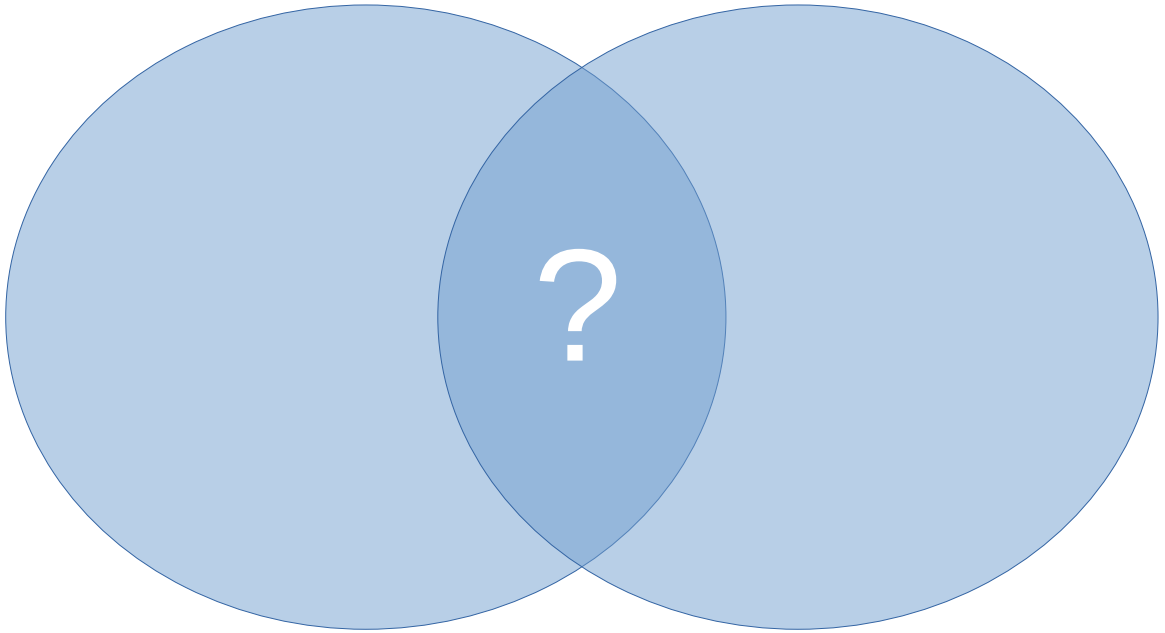
Clearly not a Firmware →



Firmware identification

← Clearly a Firmware

Clearly not a Firmware →



Firmware identification

- E.g., upgrade by printing a PS document

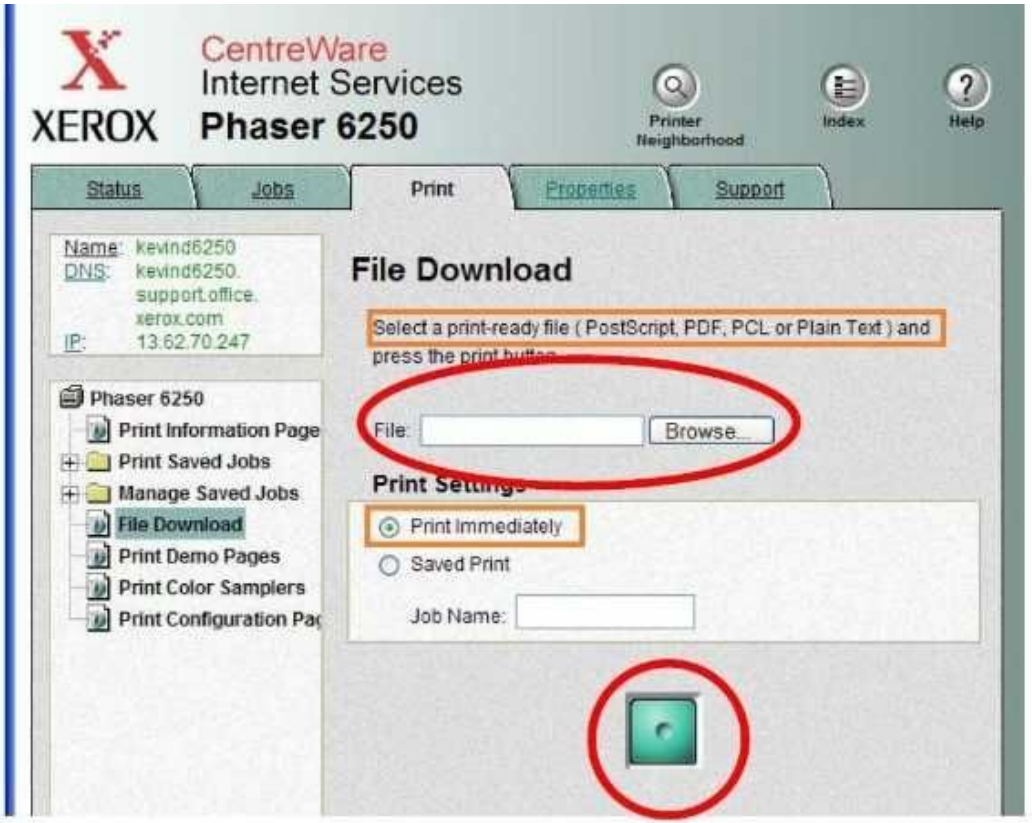


Figure 4: Select the firmware update file and press the green button to send it.

Surprise device found: Fireworks!

- Replacing wires by wireless in a system
- Lack of security
- Anyone can control the fireworks

- Fortunately firmware updates possible and now deployed

Short Paper: A Dangerous `Pyrotechnic Composition': Fireworks, Embedded Wireless and Insecurity-by-Design, A. Costin, A. Francillon, **ACM Wisec** 2014

Device Availability

- Firmware only available
  - E.g., downloaded online
- Emulator available
  - Generic emulator: works if code to analyse is generic
  - Specific emulator rarely available
- Device available
  - Limited access to the device?
  - But need to extract firmware?

Rehosting process