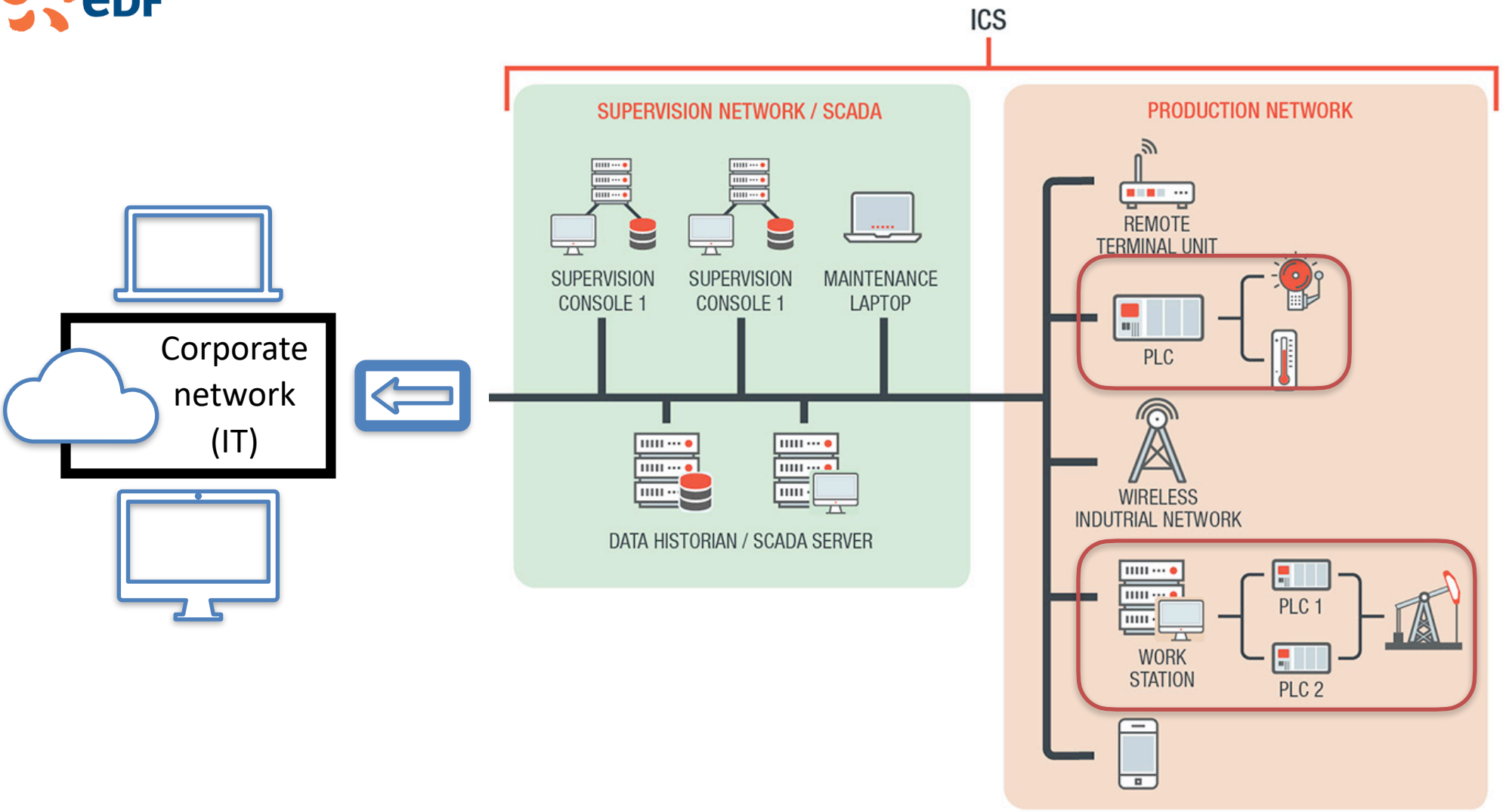
Using hardware signals to detect malware in critical embedded systems

THCon 26

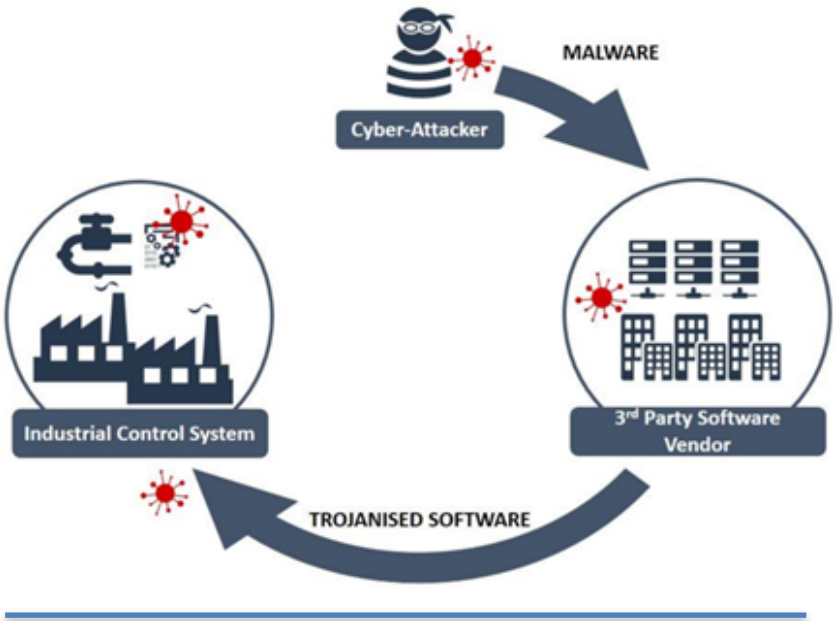
Lucas Georget[1,2], Vincent Migliore[1], Vincent Nicomette[1], Philippe Leleux[1], Arthur Villard[2], Frédéric Silvi[2]

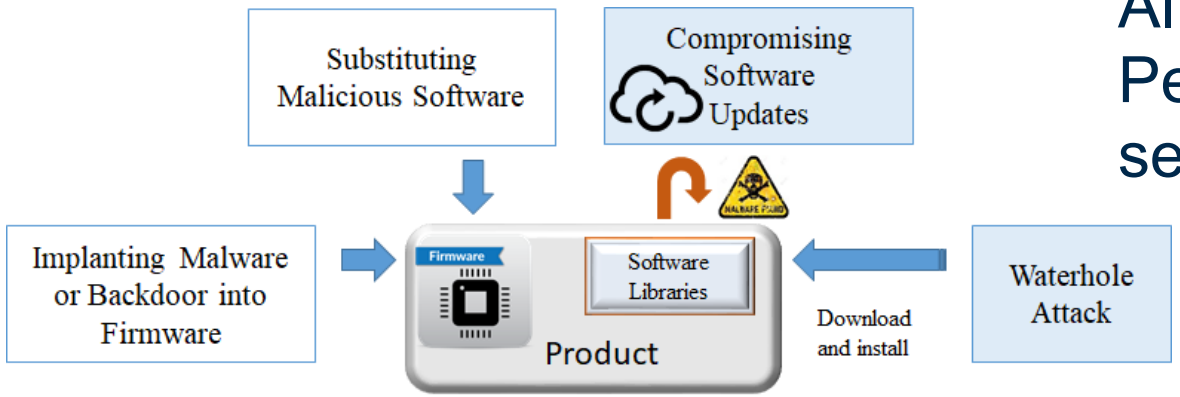
[1] LAAS-CNRS / [2] EDF R&D

Context: industrial control systems (OT)



Threat model: supply chain attack (trojans)

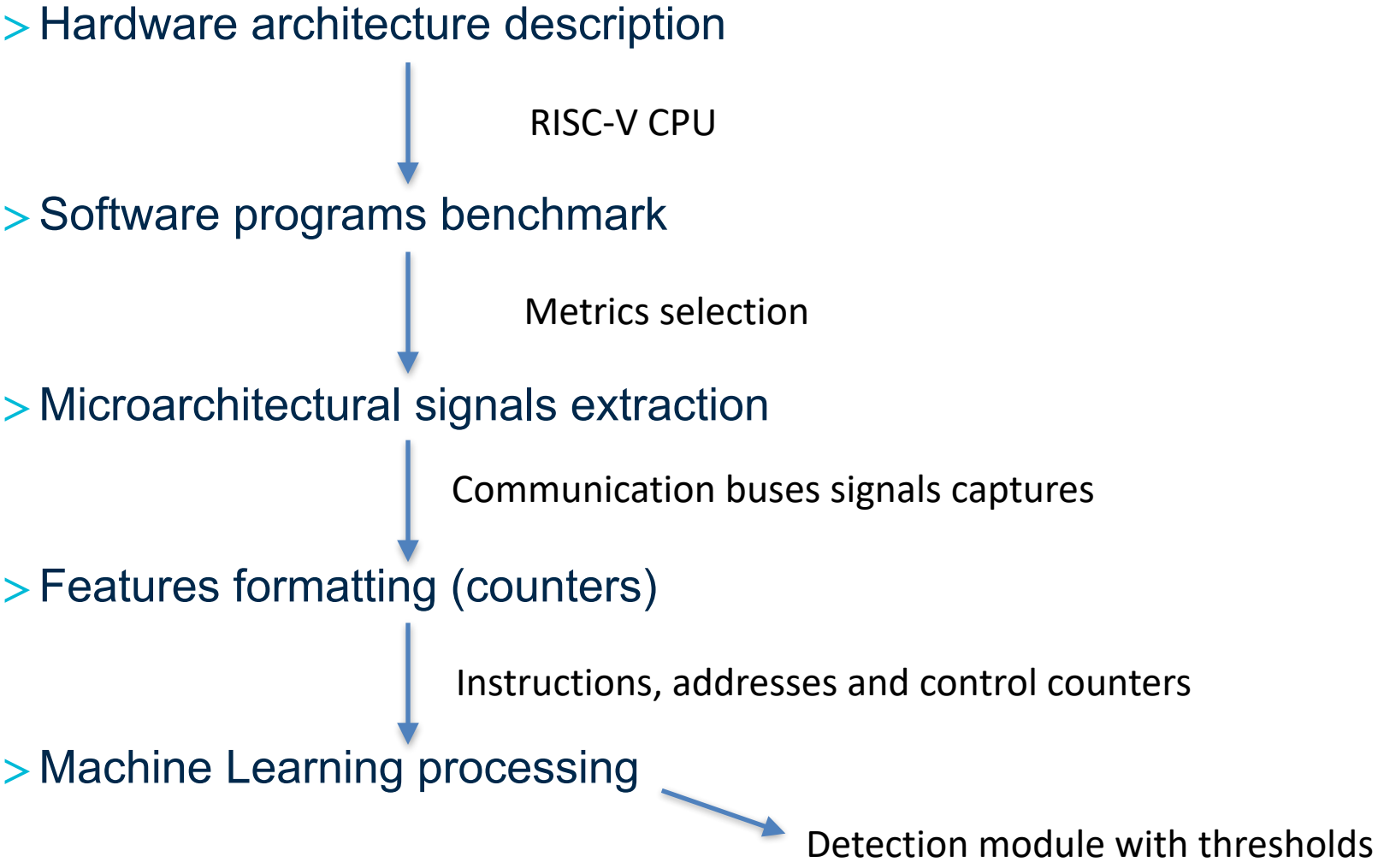




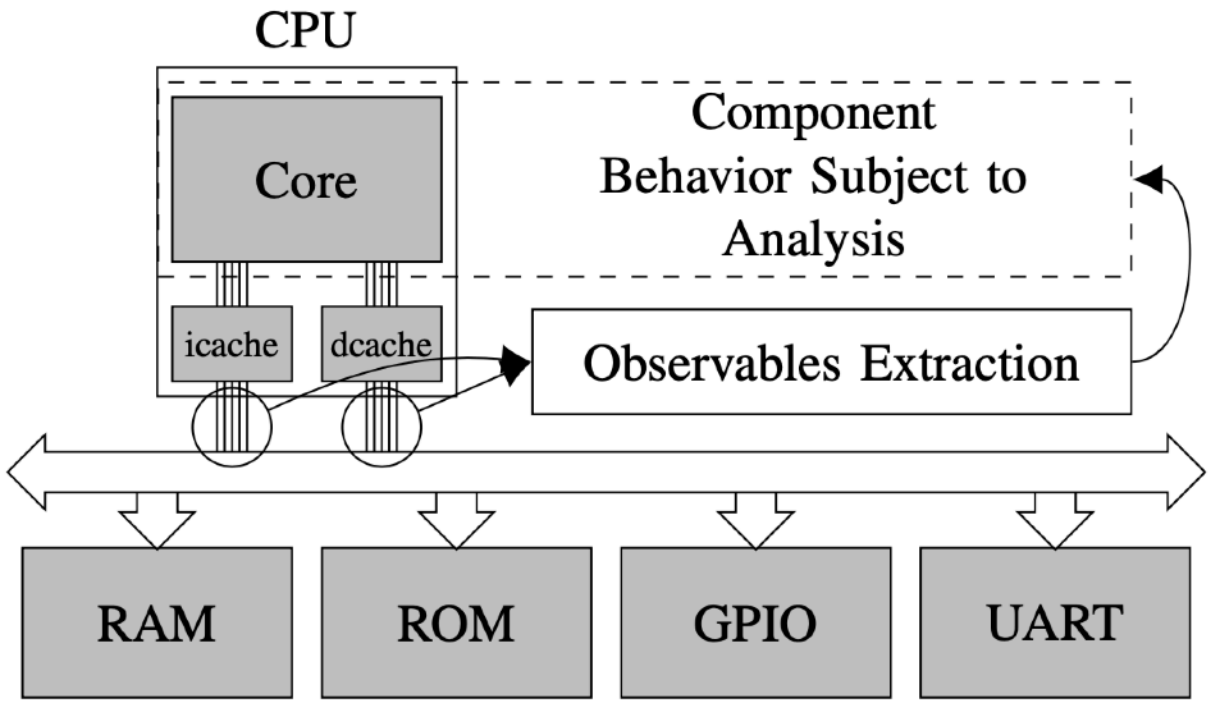
> Objective:
detect and classify malicious subtle variations of a legitimate program

> Microarchitectural problem:
Alternative to Hardware Performance Counters with a security focus!

Methodology of the approach



Signal probing: integrated logic analyzer



White: known microarchitecture

Grey: black boxes

```
analyzer_signals = [
    # IBus (could also just added as self.cpu.ibus)
    self.cpu.ibus.stb,
    self.cpu.ibus.cyc,
    self.cpu.ibus.adr,
    self.cpu.ibus.we,
    self.cpu.ibus.ack,
    self.cpu.ibus.sel,
    self.cpu.ibus.dat_w,
    self.cpu.ibus.dat_r,

    # DBus (could also just added as self.cpu.dbus)
    self.cpu.dbus.stb,
    self.cpu.dbus.cyc,
    self.cpu.dbus.adr,
    self.cpu.dbus.we,
    self.cpu.dbus.ack,
    self.cpu.dbus.sel,
    self.cpu.dbus.dat_w,
    self.cpu.dbus.dat_r,
]
```

Communication buses
(Wishbone, AXI, AXI-Lite)

RISC-V instructions: splitting the signal

31 27	26 25	24 20	19 15	14 12	11 7	6 0	
funct7		rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]			rs1	funct3	rd	opcode	I-type
imm[11:5]		rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]		rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]					rd	opcode	U-type
imm[20|10:1|11|19:12]					rd	opcode	J-type

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	≥ 80b

Table 9.1: RISC-V base opcode map, inst[1:0]=11

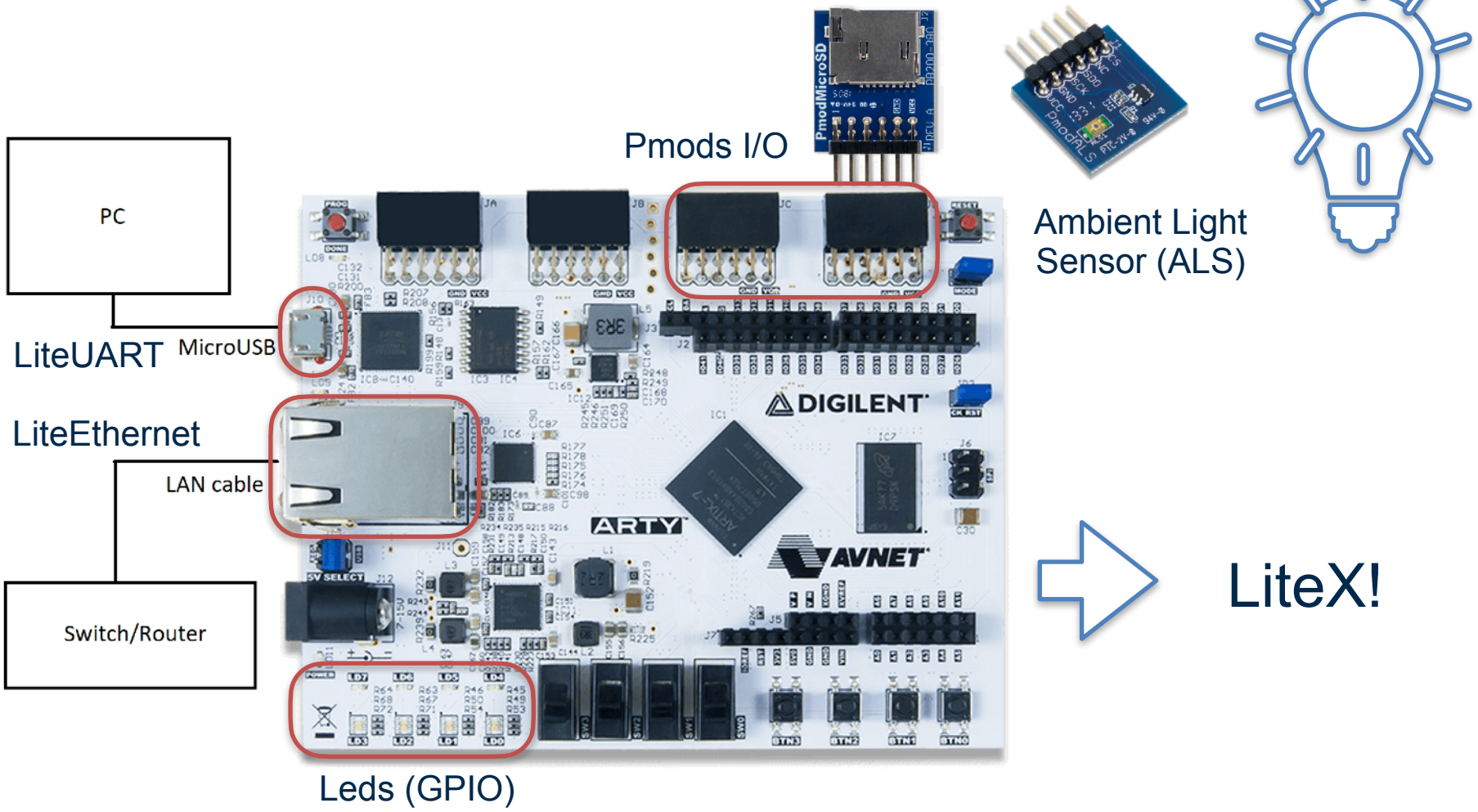
> And more for compressed instructions!

Data addresses: areas accessed

- Which zones?
  - ROM
  - MAIN_RAM
  - SRAM / SDRAM
  - CSRs
    - Submodules
    - Accessors

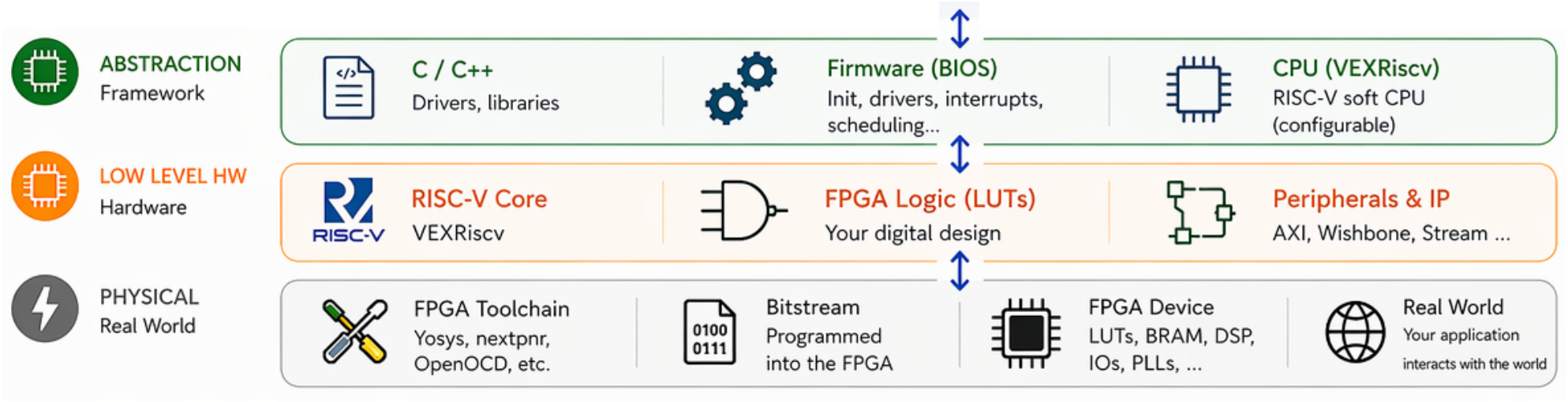
```
litex> mem_list
Available memory regions:
ROM         0x00000000 0x20000
SRAM        0x01000000 0x2000
MAIN_RAM    0x40000000 0x10000000
CSR         0x82000000 0x10000
```

Setup: programmable logic controller



LiteX last news!

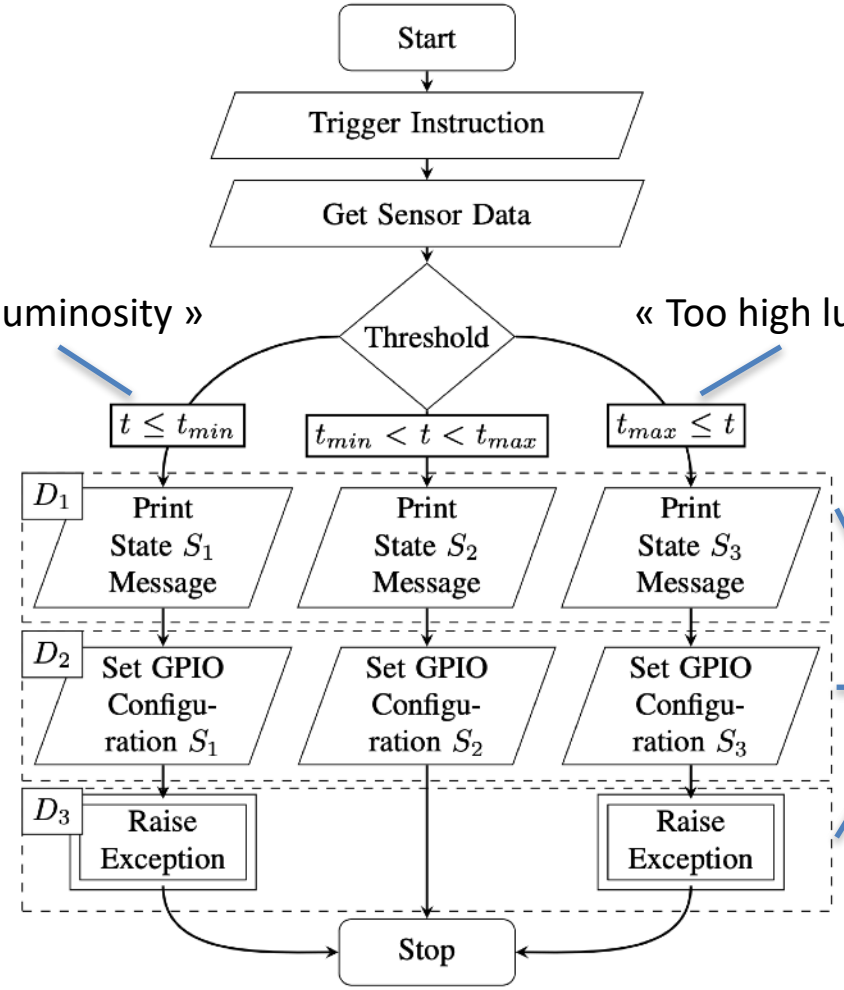
Your program/OS :)



Custom baremetal bios: firmware corruptions

« Too low luminosity »

« Too high luminosity »



Illegitimate deviations

```
        __   _ __      _ __
       / /  (_) /____ | |/_/
      / /__/ / __/ -_)>  <
     /____/_/\__/\__/_/|_|
   Build your hardware, easily!

 (c) Copyright 2012-2022 Enjoy-Digital
 (c) Copyright 2007-2015 M-Labs

 BIOS built on Dec  5 2022 23:35:03
 BIOS CRC passed (7c5b41e8)

 LiteX git sha1: a4cc859d

--========== SoC ==========--
CPU:            VexRiscv @ 1MHz
BUS:            WISHBONE 32-bit @ 4GiB
CSR:            32-bit data
ROM:            128KiB
SRAM:           8KiB
MAIN-RAM:       64KiB

--========== Initialization ============--

--========== Boot ==========--
Booting from serial...
Press Q or ESC to abort boot completely.
sL5DdSMmkekro
Timeout
Executing booted program at 0x40000000

--========== Liftoff! ==========--

LiteX minimal demo app built Dec  5 2022 23:34:50

Available commands:

ALS     - Get ALS
Printf  - wrong print
Leds    - wrong leds
Alarm   - wrong alarm
Malicious - All wrong

litex-demo-app>
```

ML Analysis: software corruptions diagnostic

- Clustering
  - ACP/t-SNE
- Classification
  - XgBoost
  - DecisionTree

TABLE III
ACCURACY SCORES BY COMBINATION

Features	*transition counters*	*state counters*	Both
Instructions	84.17%	90.67%	90.67%
Data	90.67%	99%	99%
Both	92.29%	99.67%	99.67%

- Better results than Hardware Performance Counters!
  (On CVA6: 72,5% HPCs / 90% with our custom signals)

More knowledge about microarchitectural behaviors

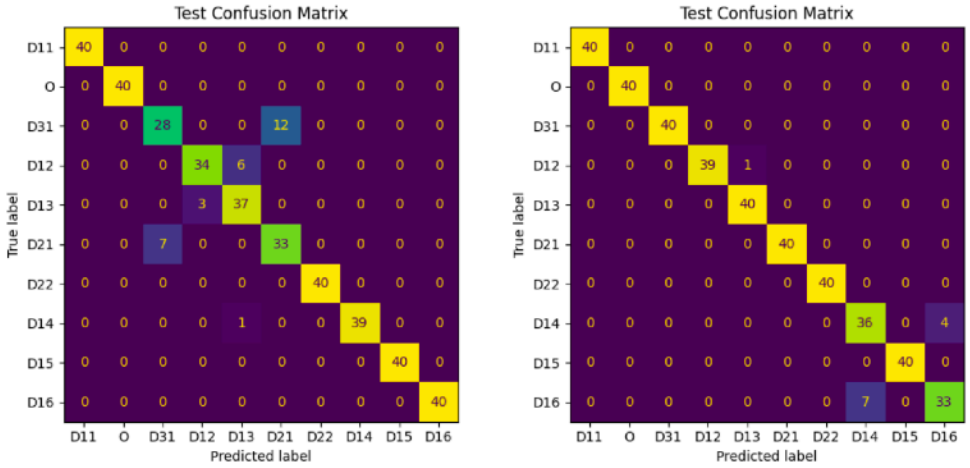


Fig. 4. Confusion matrices for the *instruction-only* (left) and *data-only* (right) models.

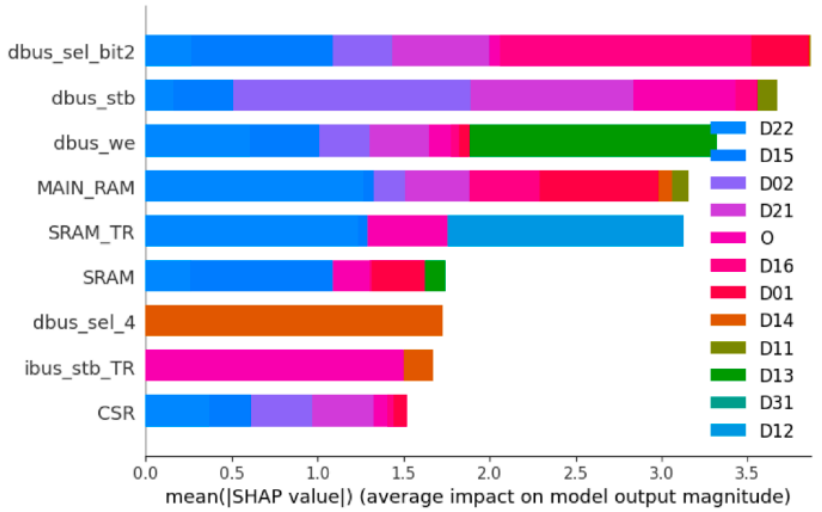


Fig. 5. Features relevance with program classification contribution

Embedding a lite hardware detection module

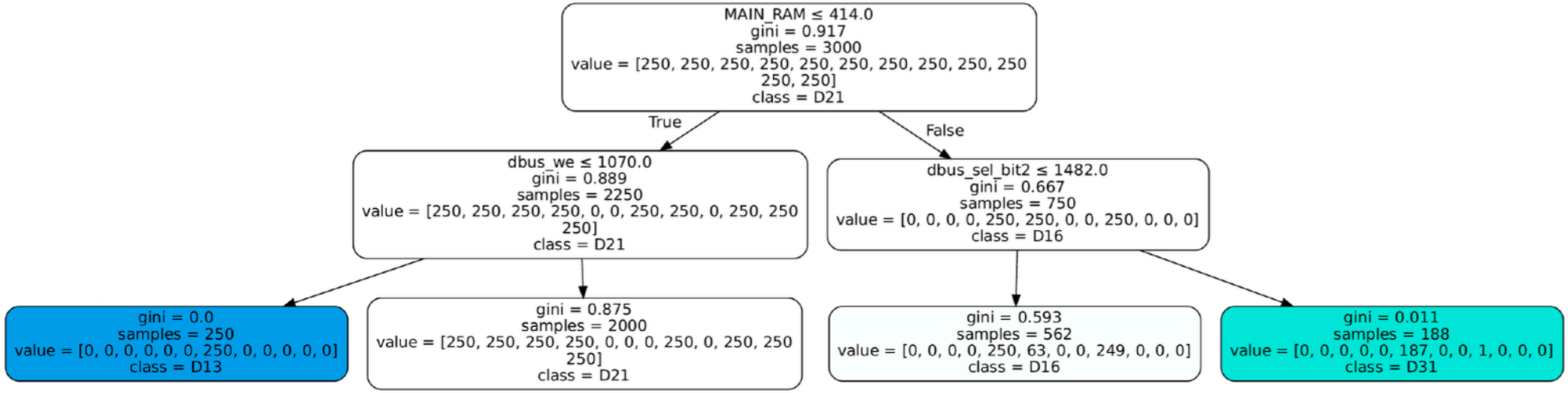
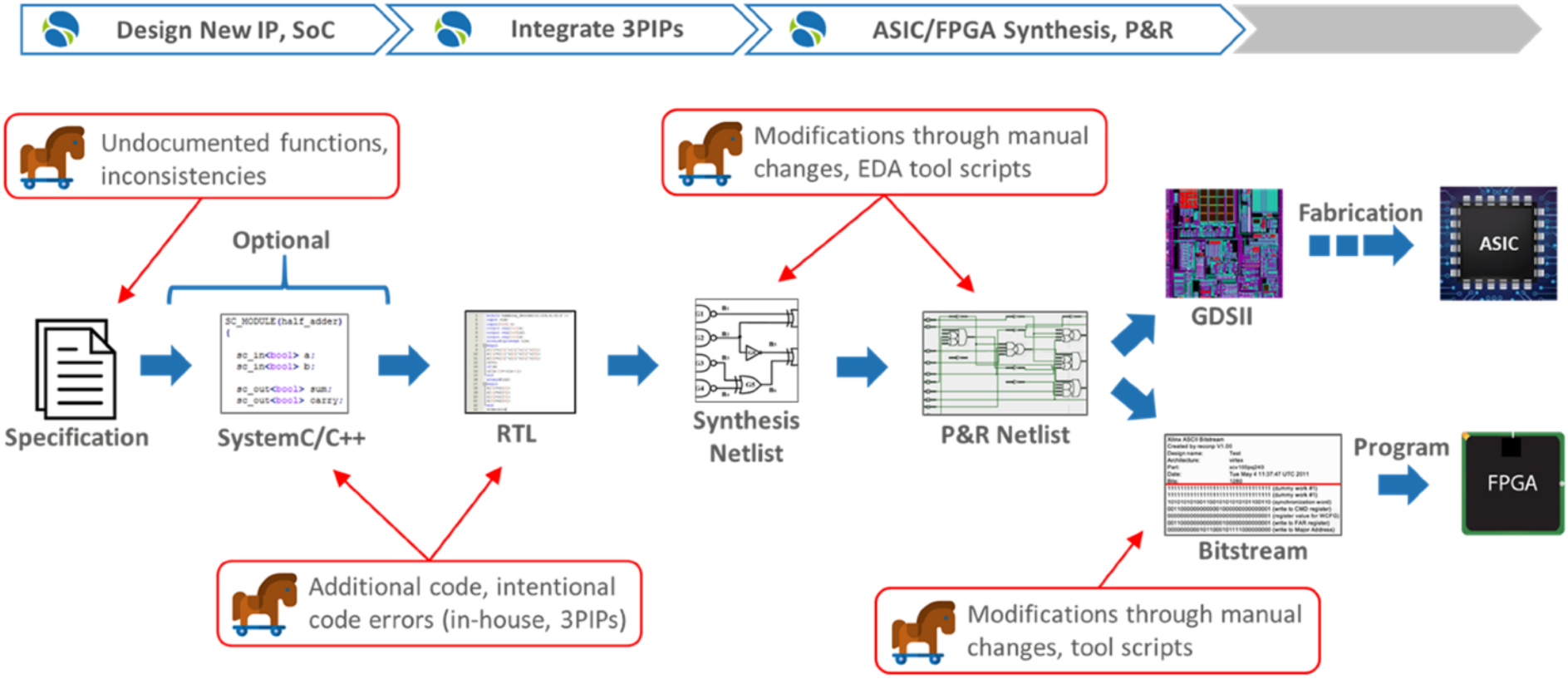


Fig. 3. Part of the Decision Tree from relevant features

Current work: hardware trojans

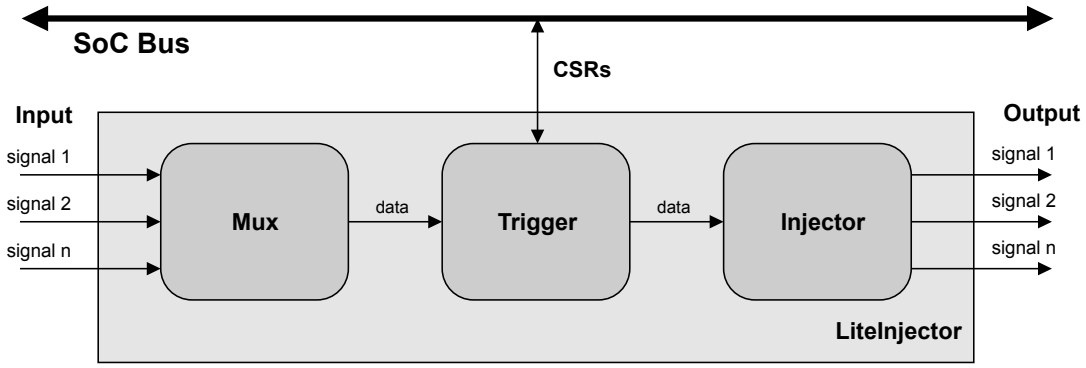


> Thanks to many people! Questions?

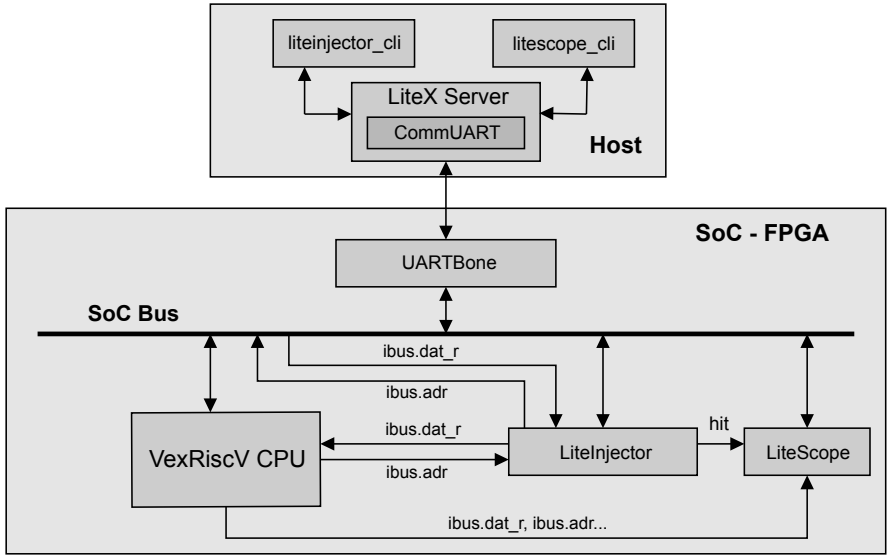
Bibliography: slides

- 2: Industrial Control System (TrendMicro)
- 3: Securing supply chains (TCG) - Supply Chain Attack (InfoSec)
- 10: Use LiteScope To Debug A SoC (Enjoy-Digital)
- 11: The RISC-V Instruction Set Manual, Volume I (RISC-V)
- 13: RISC-V Linux in LiteX/Rocket on FPGA Arty A7-100T (SHC)
- 15: MicroPython on LiteX (LiteX-Hub)

LiteInjector: A LiteX Extension for Fault Injection



Architecture of the LiteInjector emulator



Example SoC that use LiteInjector and LiteScope

> https://github.com/labsticc-arcad/LiteInjector

Henault, Adam and Tanguy, Philippe and Lapôtre, Vianney
2025 28th Euromicro Conference on Digital System Design (DSD)

Future works?

- Self-hosted Linux FPGA system (gateware)
  - And then play with HAL (Hardware Analyzer)