Fault Security Analysis and Verification: Challenges and New Directions

Damien Couroussé – CEA List, Univ. Grenoble Alpes, France
Karine Heydemann – Thales DIS, France & LIP6, Sorbonne Univ., France
Mathieu Jan – CEA List, Univ. Grenoble Alpes, France

Fault injection attacks (FI / FIA)

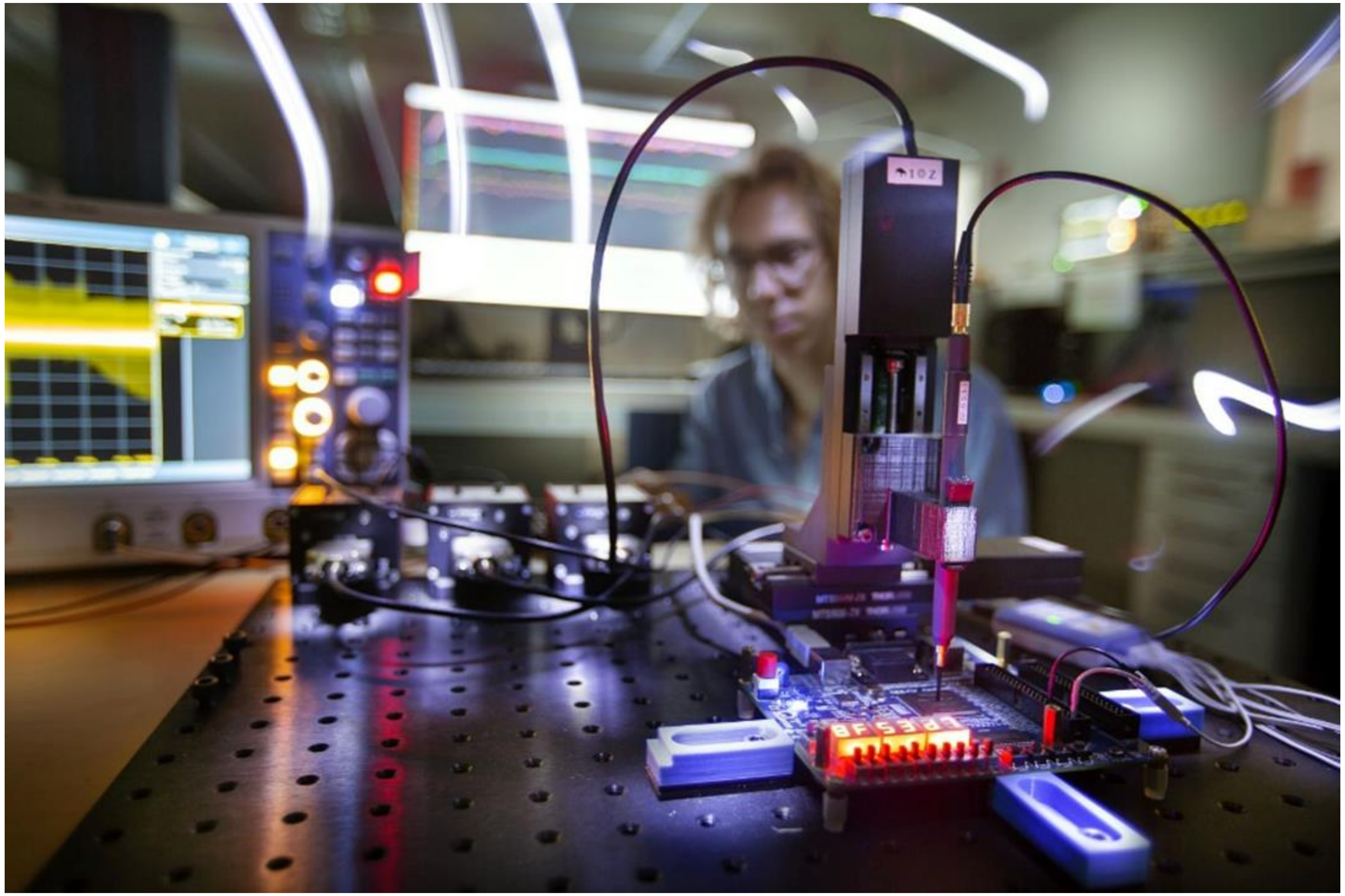
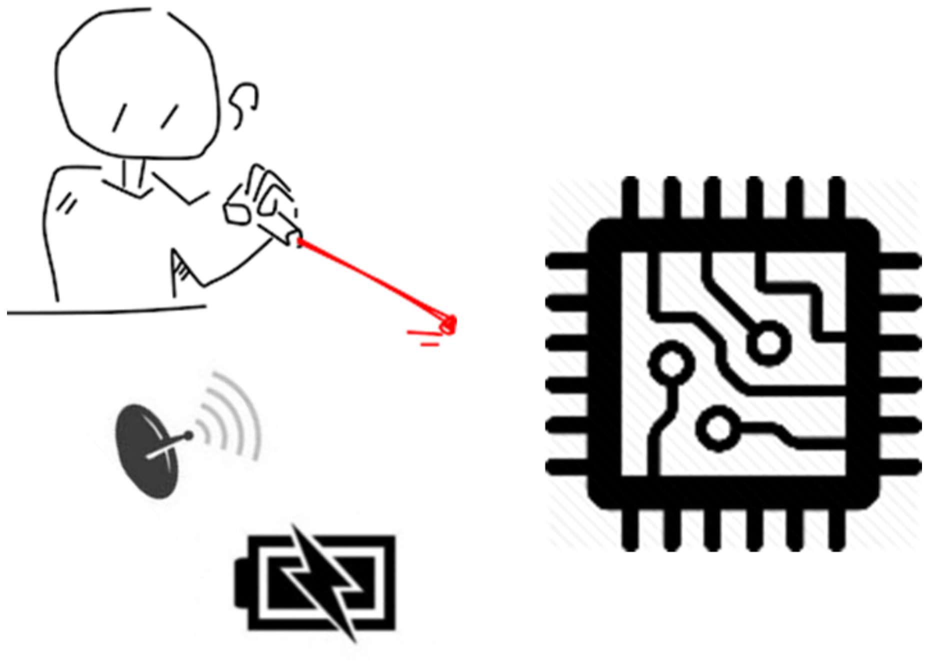
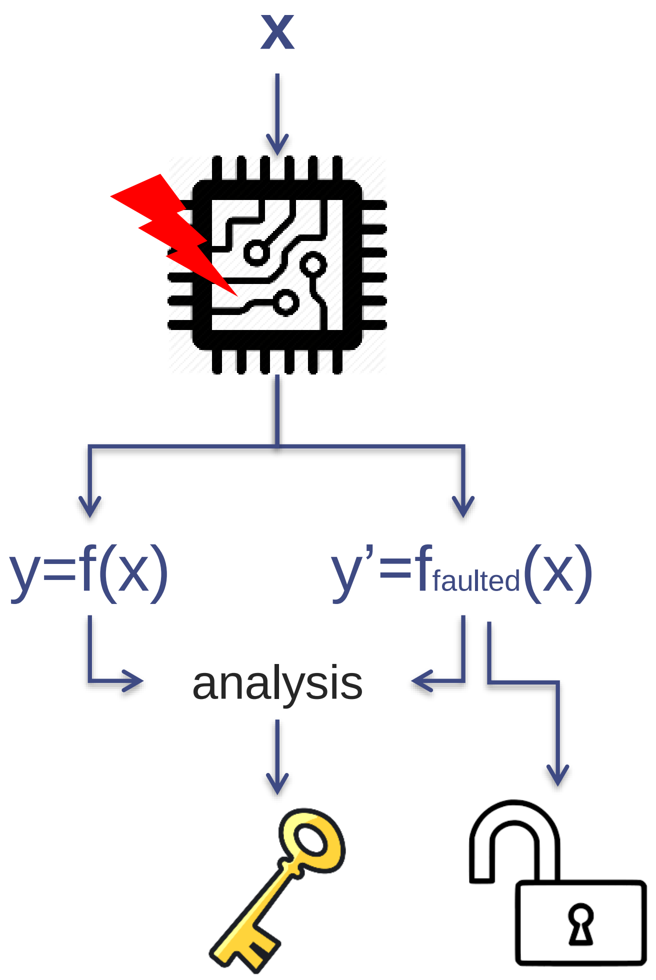


Photo credit: Christian MOREL / CEA

Fault injection attacks, conceptually





Highly effective against cryptographic implementations

Can leverage software vulnerabilities [Cui & Rousley, 2017]

Processors security wrt. fault injection

FI can target many elements in complex SoCs

e.g. **memory hierarchy** [Trouchkine, 2021]

The attack setup around FI can be elaborated

e.g. **bypassing a secure boot**, on complex SoCs [Vasselle, 2020] [Fanjas, 2023]

Fault effects are diverse and hard to understand

black-box characterization [Trouchkine, 2021]

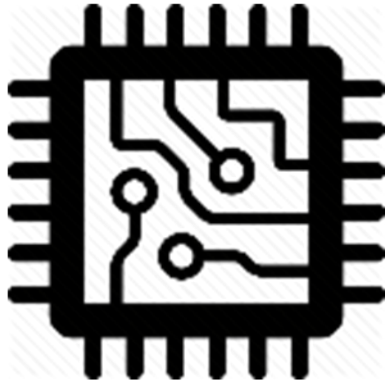
characterization methodology [Proy, 2019]

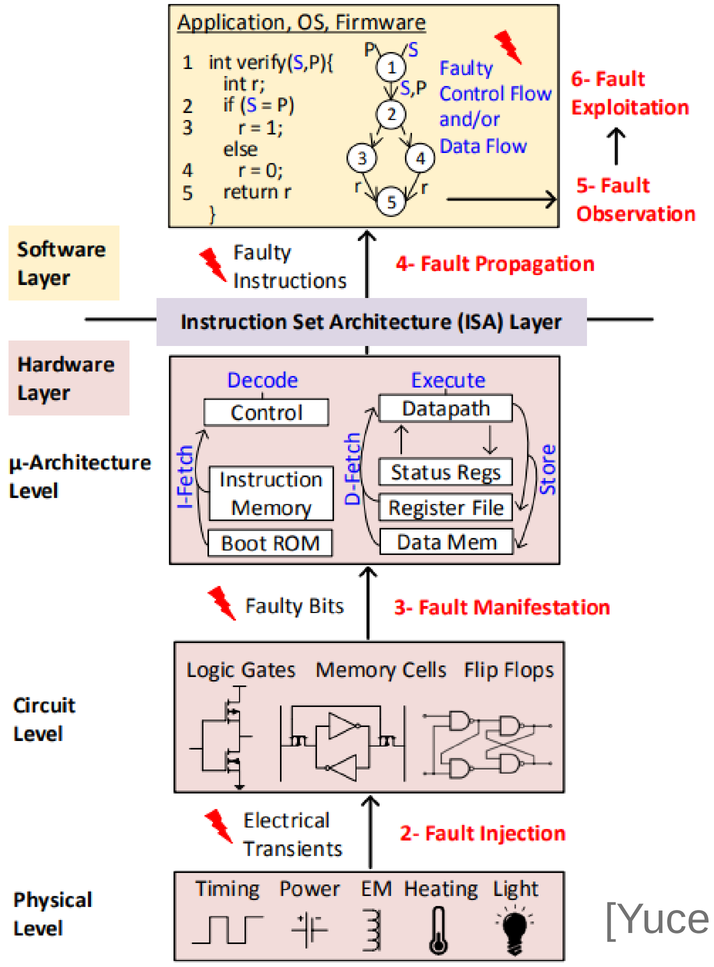
Finding a needle in a haystack?

- Fault exploitation is expensive
- Characterization of fault effects is challenging
  - limited observability
  - effects depends on FI setup
- BUT lots of needles available
  - Large attack surface
  - Faults can induce many unexpected, exploitable effects

[Trouchkine, 2021] Electromagnetic fault injection against a complex CPU, toward new micro-architectural fault models 10.1007/s13389-021-00259-6
[Fanjas, 2023] Exploration of system-on-chip secure-boot vulnerability to fault-injection by side-channel analysis 10.1007/978-3-031-25319-5_2
[Vasselle, 2020] Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot-Extended Version 10.1109/TC.2018.2860010
[Proy, 2019] A First ISA-Level Characterization of EM Pulse Effects on Superscalar Microarchitectures: A Secure Software Perspective 10.1145/3339252.3339253

Fault injection attacks, behind the scenes





[Yuce, 2018]

Fault injection attacks, behind the scenes

Different abstraction layers involved

- Circuit level: initial fault effect
- Software level: consequences of the lower-level fault effects

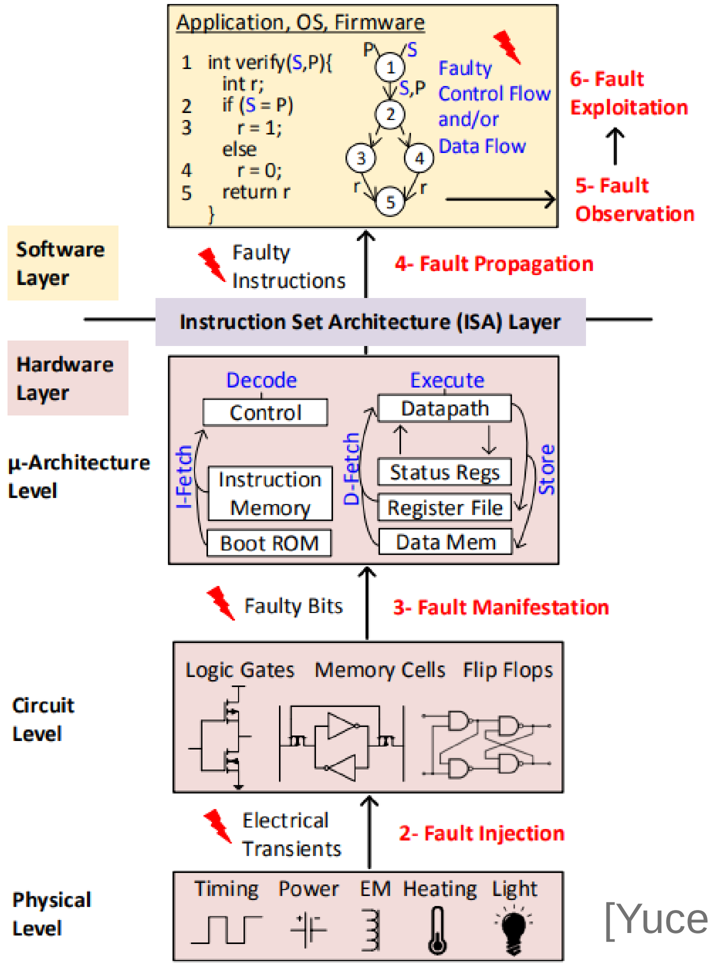
Turning attention to processor microarchitecture

- FI on processor pipelines can bypass SW protections [Yuce, 2016]
- Importance of hidden microarchitectural registers [Laurent, 2021]
- Microarchitectural fault effects are leveraged by specific SW conditions: init. state, run program [Tollec, 2022]

Fault effects depend on the current *system state*

- Faults can have no effect
- Faults can manifest after unknown amount of time
- Software system state = execution context

Joint HW-SW analysis is mandatory!



[Yuce, 2018]

[Laurent, 2021] Bridging the Gap between RTL and Software Fault Injection. 10.1145/3446214
[Yuce, 2016] Software Fault Resistance is Futile: Effective Single-Glitch Attacks. 10.1109/FDTC.2016.21
[Tollec, 2022] Exploration of fault effects on formal RISC-V microarchitecture models. 10.1109/FDTC57191.2022.00017

Security analysis

Security evaluation

- *In situ*: real system, real fault injection bench
  - E.g. certification
- Representative / accurate
  - of attacker capabilities
  - of system robustness
- Non-exhaustive

(out of the scope of this talk)

Security verification

- Model-based:
  - HW,
  - SW,
  - attacker
- Non-representative / accurate
  - of target system
  - of real fault effects
- Exhaustive

Security verification

Status

- Faults modeling incurs extra analysis complexity
  - State space explosion
    - Nb possible states
    - New transitions between reachable states
  - Multiple faults: combinatorial explosion
- Microarchitectural HW models + SW
  - Increase of models size

Challenges

- Growing complexity of real case studies
  - Large HW designs, large programs (SW)
- Exhausive verification wrt. model size explosion

➔ Simulation

- Efficient evaluation of model behaviour using *concrete* input state
- Can evaluate large models
- Exhausitivity is impractical: iterate ∀ input states, ∀ fault instances

➔ Formal methods

- Designed to address exhaustivity
- Cannot address large models
  - Especially challenging wrt. FIA

Outline

Formal Modeling for Microarchitectural Fault Injections

S. Tollec, M. Asavoae, D. Couroussé, K. Heydemann, and M. Jan “µArchiFI: Formal Modeling and Verification Strategies for Microarchitectural Fault Injections,” in *FMCAD*, 2023.

https://zenodo.org/records/7958412
https://github.com/CEA-LIST/uArchiFI *

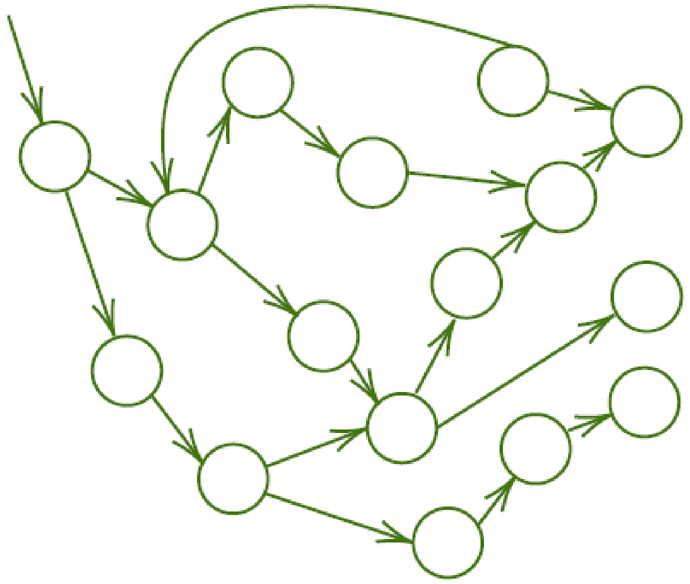
* Currently empty, will be populated soon!

Modeling: faulty HW transition systems

Hardware modeling

Transition system $\mathcal{M} = (S, S_0, X, T)$ where

- A *system state* $s \in S$ corresponds to a valuation of circuit registers, i.e., $s := \langle r_1, \ldots, r_n \rangle$.
- An *input* $x \in X$ is a vector $x := \langle i_1, \ldots, i_m \rangle$.
- $S_0 \subseteq S$ is the set of initial states,
- $T : S \times X \to S$ is the transition function of the circuit.



—Hardware

Need for a tool that automatically:

- Parses hardware description languages
- Builds a hardware transition system

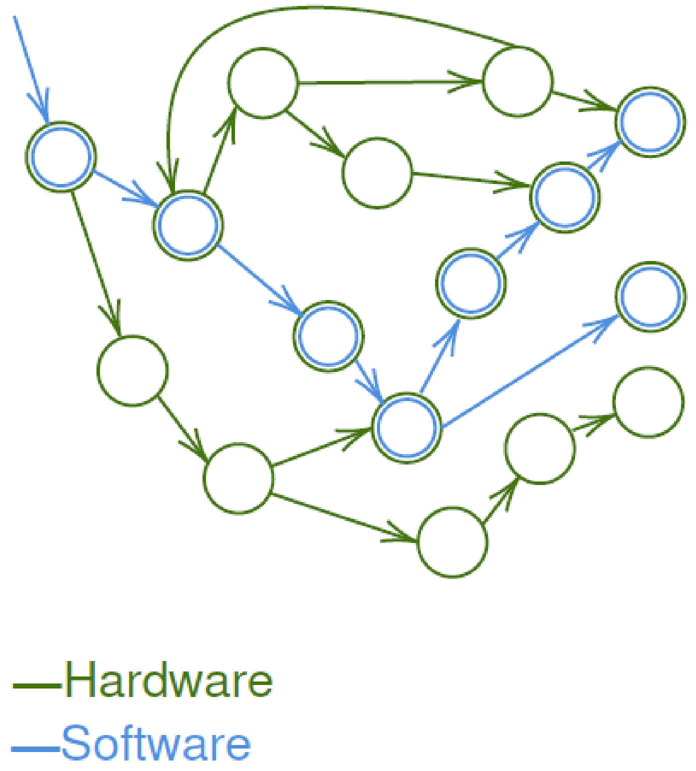
Modeling: faulty HW transition systems

Software program mapping

The program is encoded in the initial state of a memory modeled simultaneously with the processor, i.e., S_0.

Requirements

- Initialize the initial state of the transition system
- Simulate the system up to the desired state



Modeling: faulty HW transition systems

Fault injection model

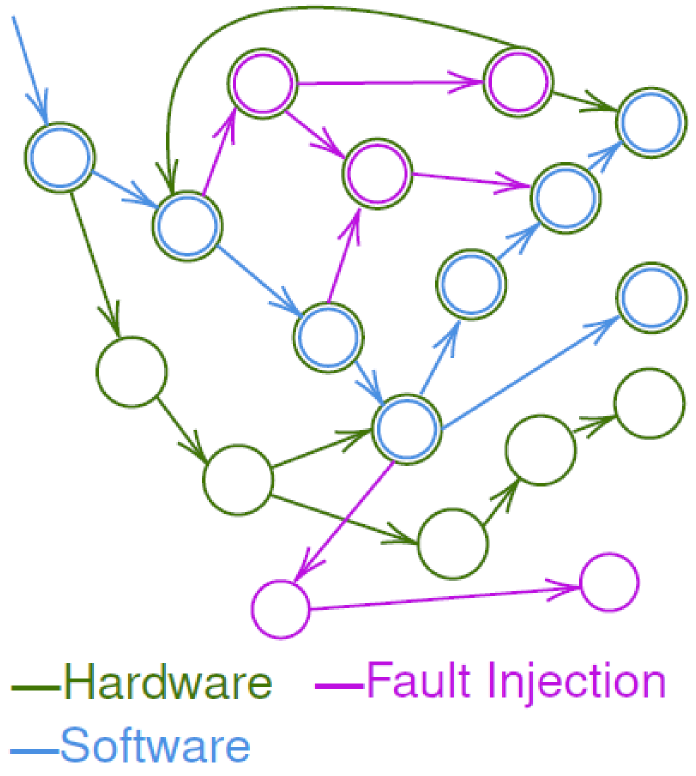
Fault model $\mathcal{F} \subseteq \mathcal{L} \times \mathcal{T} \times \mathcal{E}$ where

- $\mathcal{L}$ is the set of possible locations of the fault,
- $\mathcal{T}$ is the timing range of the fault injection,
- $\mathcal{E}$ is the set of possible effects of the fault.

  E.g., bit-flip, byte-reset, symbolic value

Need for a tool that automatically:

- Modifies the transition system according to the fault model



Modeling: faulty HW transition systems

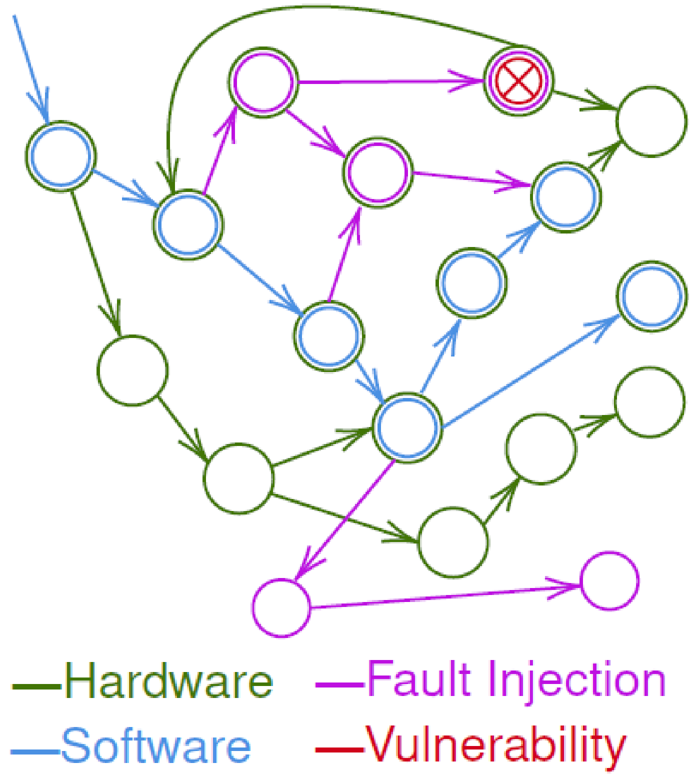
Attacker model

Attacker model $\mathcal{A} = (\mathcal{F}, \varphi, N)$ where

- $\mathcal{F}$ is the fault model,
- φ is the *attacker goal* defined as a reachability property on the transition system,
- N is the maximum number of fault injections.

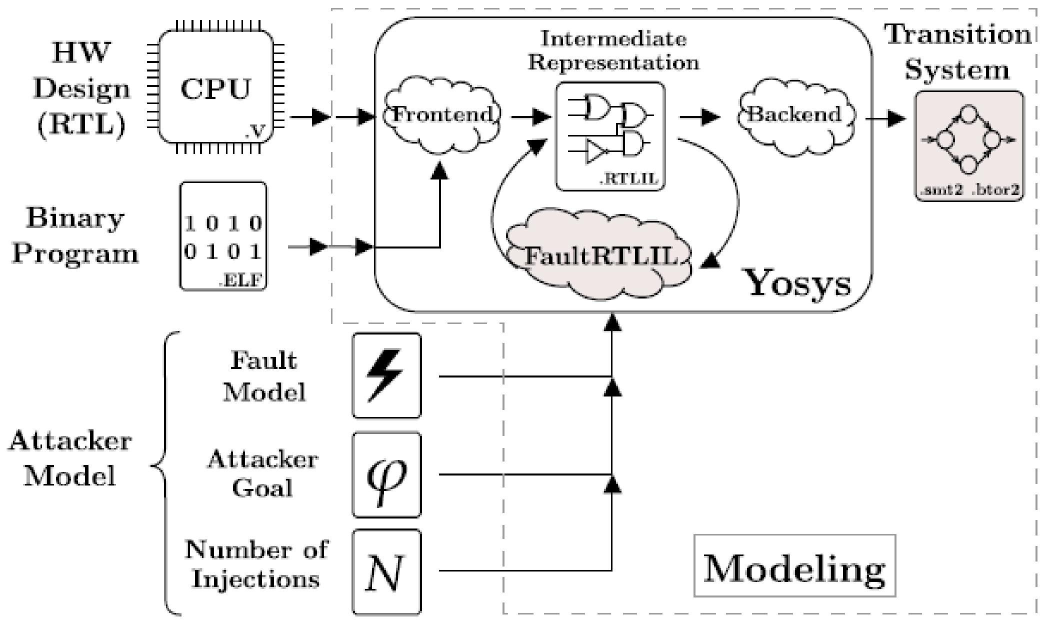
Need for a verification procedure that automatically:

- Finds whether the attacker goal is reachable
- Provides a counterexample to understand the propagation of the fault and its final consequences



µArchiFI implementation: system modeling

µArchiFI workflow



[FDTC, 2022]

- Highlights subtle fault effects in microarchitecture
- Analyses consequences in software

µArchiFI infrastructure (based on Yosys)

- Frontend: Hardware description languages, e.g., Verilog
- Yosys Intermediate Representation (RTLIL): Graph with gates and connections
- Takes an attacker model as input
- Formal backend: Aiger, SMV, Btor2, SMTLib

Transition system generation

- Bind the HW design and the binary program
- Simulate the system up to the desired state
- Include the attacker model

µArchiFI in practice: three use cases

Use case names		*I - Robust Software*	*II - Robust Hardware*	*III - Cryptographic Software*
Hardware design	name:	**CV32E40P (Riscy)** - RISC-V - 4 stages	**Secure Ibex** - RISC-V - 2 stages - dual core	**Ibex** - RISC-V - 2 stages
	gates:	2842	4422	1983
	FFs:	179	211	114
	size*(GE):	89954	61452	26327
Software program		VerifyPIN_V7 [Dur+16]	VerifyPIN_V1 [Dur+16]	Key Schedule (AES) [kok19]
Attacker Goal φ		Bypass authentication without triggering SW alert	Bypass authentication without triggering HW alert	Set to 0 a byte in the penultimate round key
Fault model $\mathcal{F}$	location:	Sequential logic Control Path	Sequential logic Redundant CPU Core	Combinational logic Execute stage of CPU
	effect:	Symbolic	Symbolic	Reset
	timing:	60:*	*	*
Number of FIs N		1	5	2
BMC depth k		75	46	38
Verification results		φ is reachable	φ is unreachable	φ is unreachable (φ reachable with N=4)

* *when synthesized with the open-source Nangate45 standard cell library*

Limited HW size

Limited SW size

Bounded verification (~100 cycles)

Fault-Resistant Partitioning of Secure CPUs

CHES 2024
→ Thursday, September 5
→ Fault Resistance I

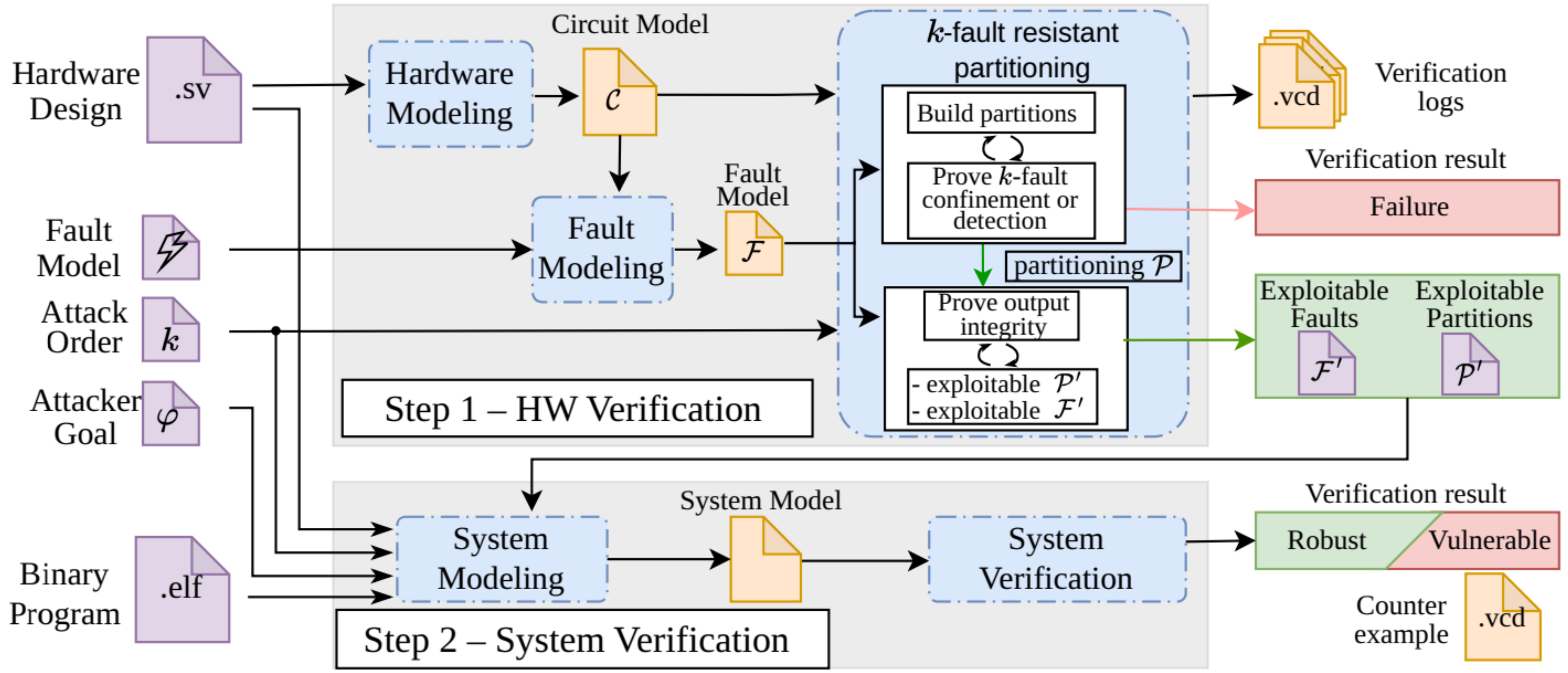


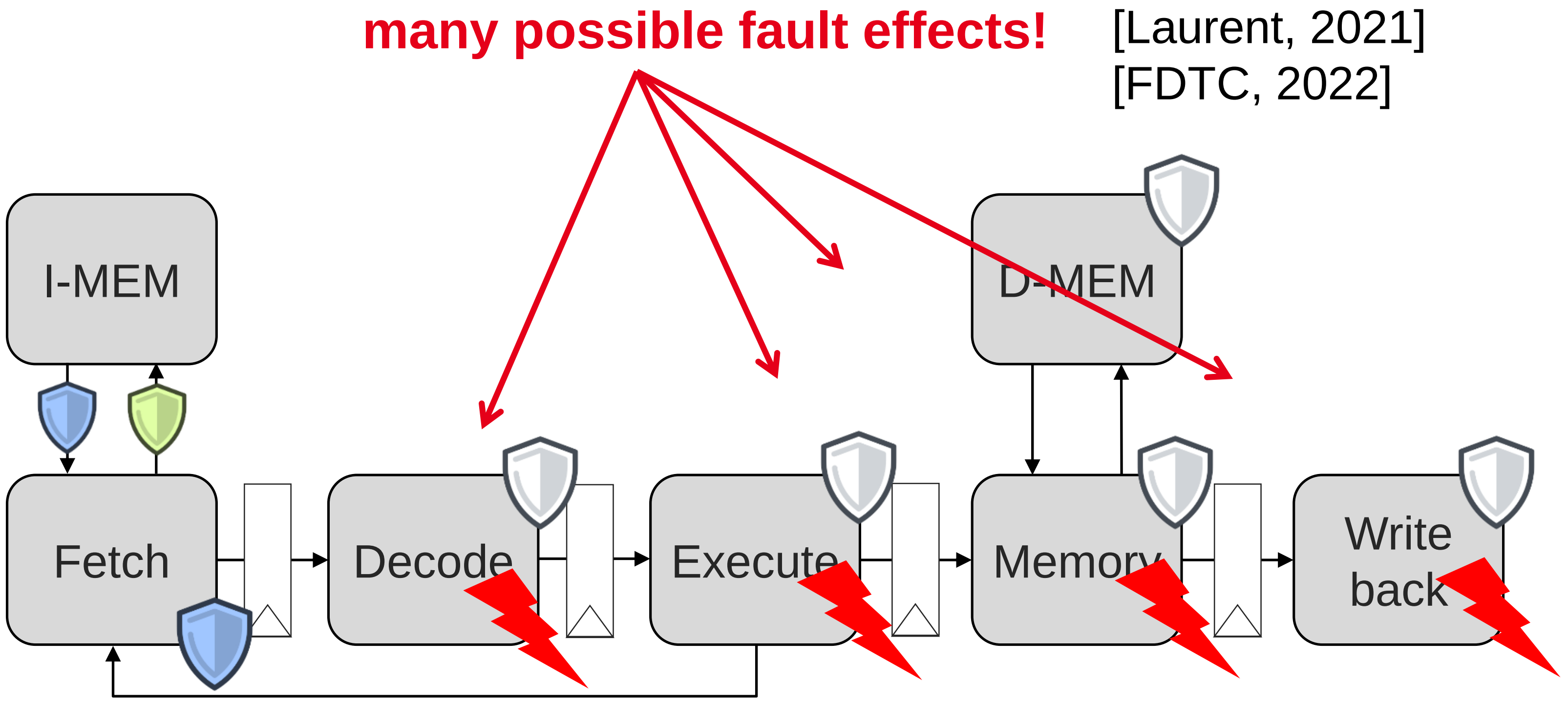
Figure 3: Co-verification methodology to evaluate SW/HW systems against faults attacks.

Protecting the Microarchitecture

T. Chamelot, D. Couroussé, and K. Heydemann “**MAFIA**: Protecting the Microarchitecture of Embedded Systems Against Fault Injection Attacks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2023.

Problem: Faults targeting control signals

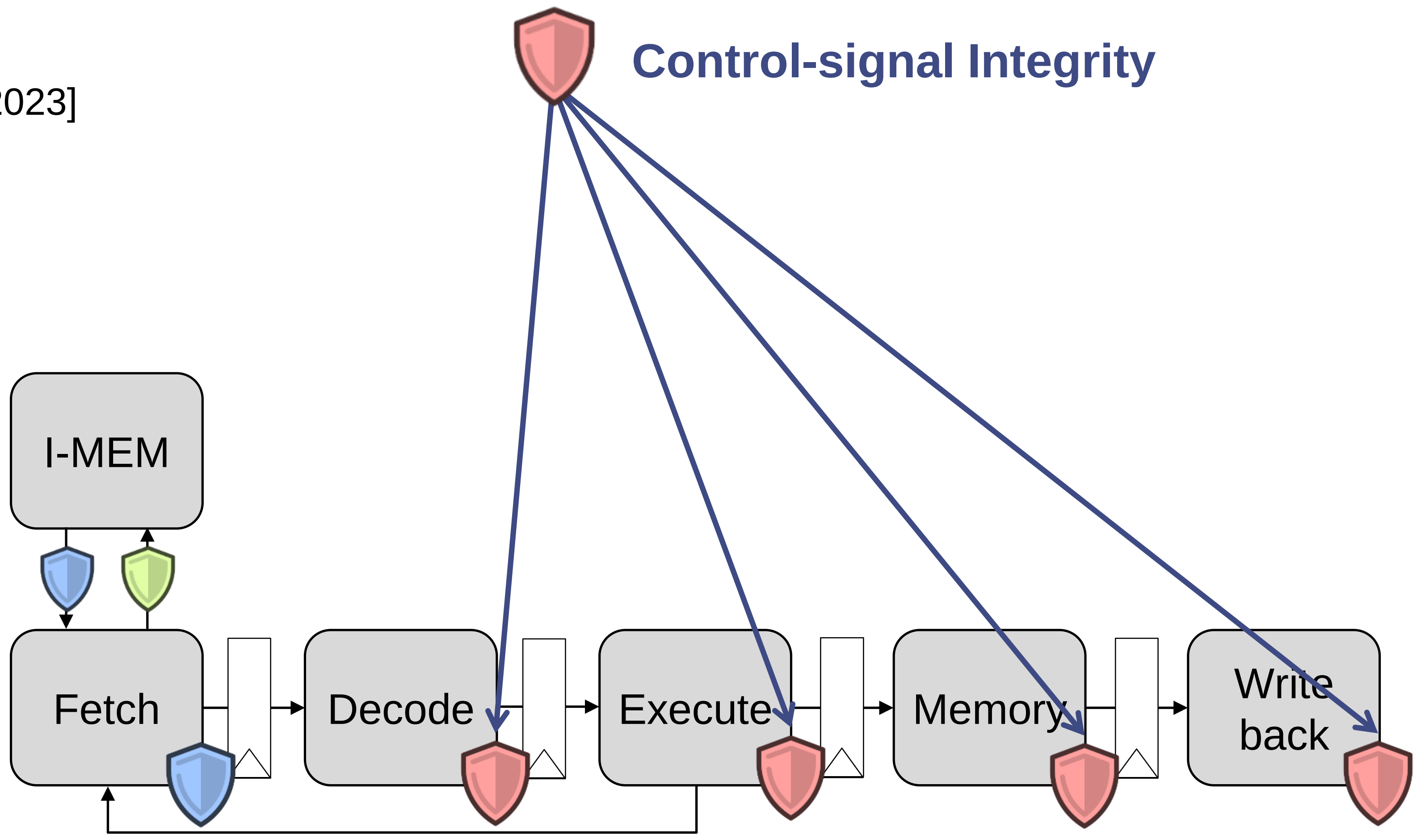
- 🛡 Data integrity
- 🛡 Code authenticity / integrity
- 🛡 Control-flow integrity



[Laurent, 2021] Bridging the Gap between RTL and Software Fault Injection. 10.1145/3446214
[FDTC, 2022] Exploration of fault effects on formal RISC-V microarchitecture models. 10.1109/FDTC57191.2022.00017

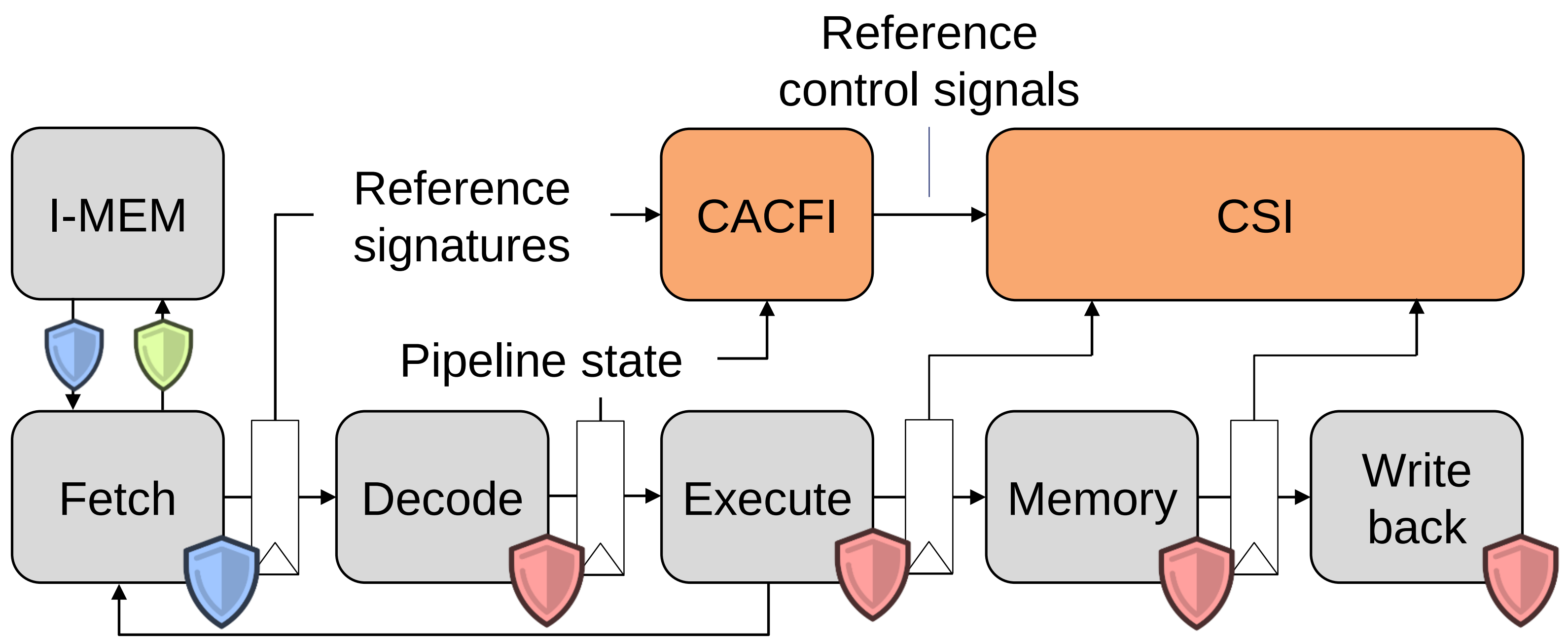
Problem: Faults targeting control signals

- 🛡 Data integrity
- 🛡 Code authenticity / integrity
- 🛡 Control-flow integrity
- 🛡 **Control-signal integrity** [TCAD, 2023]



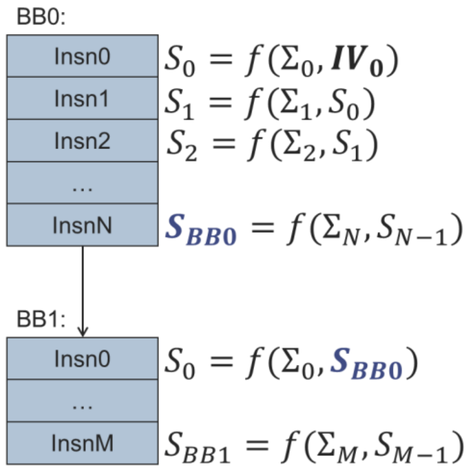
MAFIA: Protection of the microarchitecture against fault injection attacks

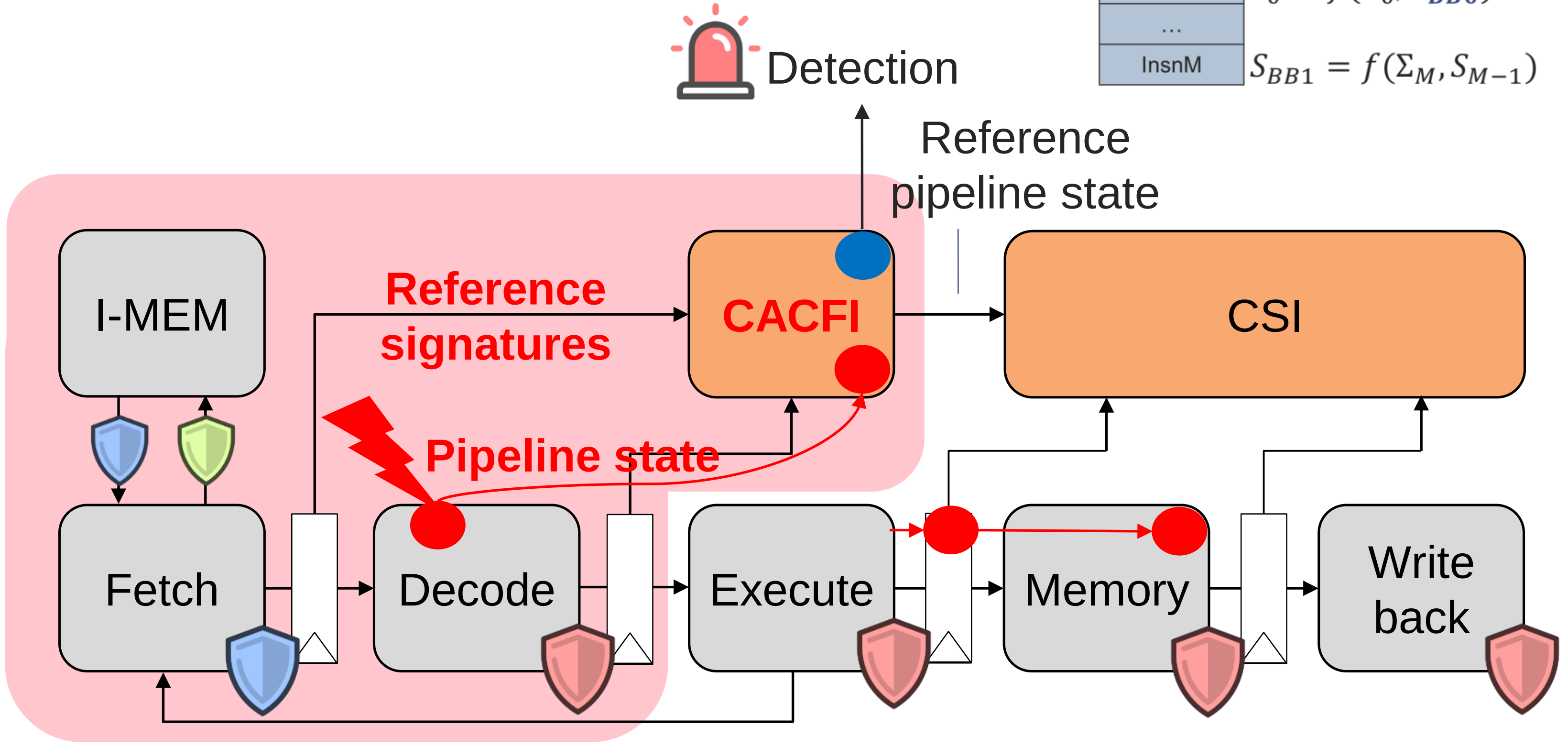
- Data integrity (not supported)
- Code authenticity / integrity
- Control-flow integrity
- **Control-signal integrity**



Code Authenticity and Control-Flow Integrity (CACFI)

- Code authenticity / integrity ← signature function $\boldsymbol{f}$
- Control-flow integrity ← signature chaining
- Control-signal integrity ← signature computed from **pipeline state** values

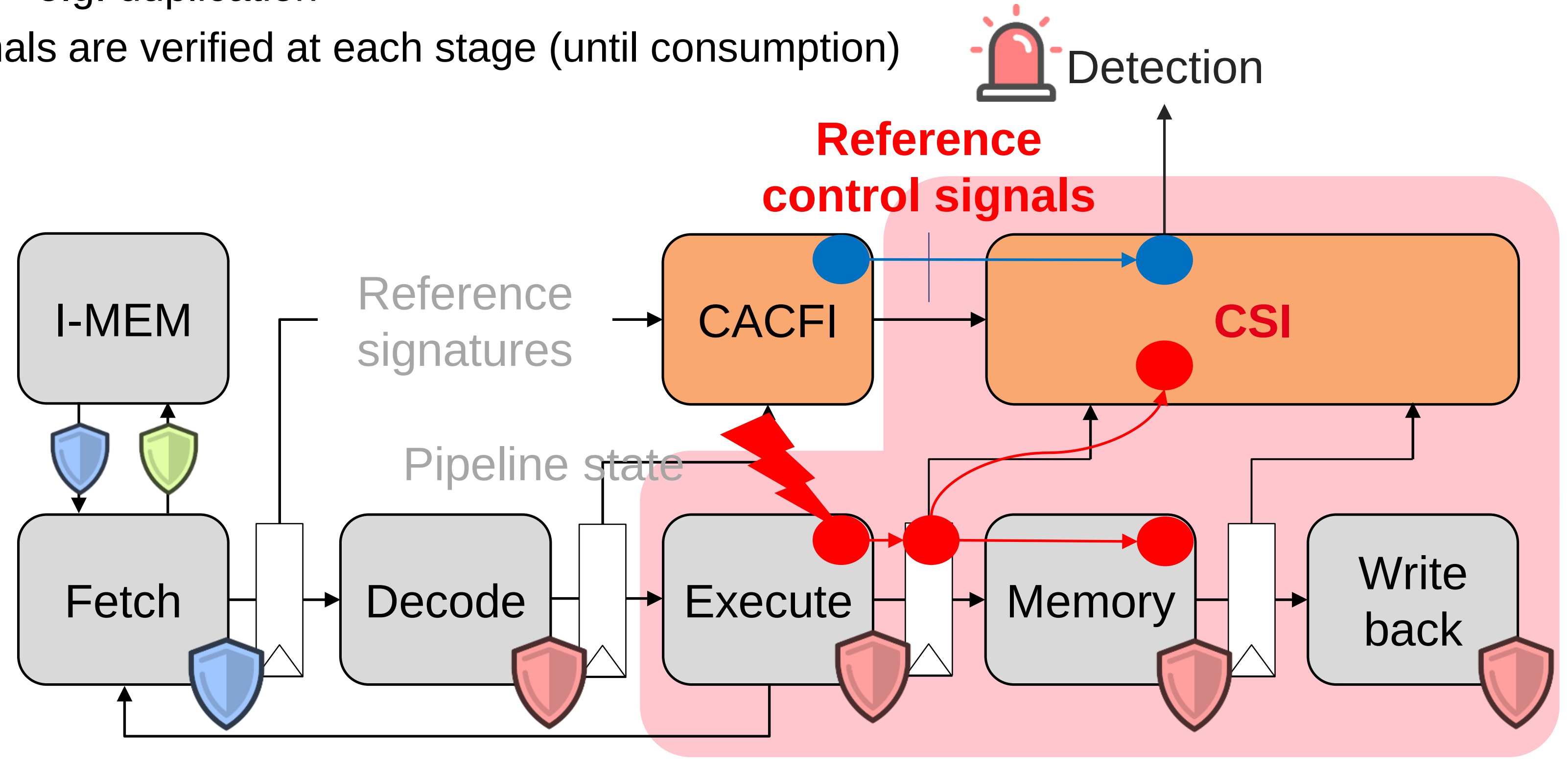

BB0:
Insn0
Insn1
Insn2
…
InsnN
$S_0 = f(\Sigma_0, \boldsymbol{IV_0})$
$S_1 = f(\Sigma_1, S_0)$
$S_2 = f(\Sigma_2, S_1)$
$\boldsymbol{S_{BB0}} = f(\Sigma_N, S_{N-1})$
BB1:
Insn0
…
InsnM
$S_0 = f(\Sigma_0, \boldsymbol{S_{BB0}})$
$S_{BB1} = f(\Sigma_M, S_{M-1})$



Control-Signal Integrity (CSI)

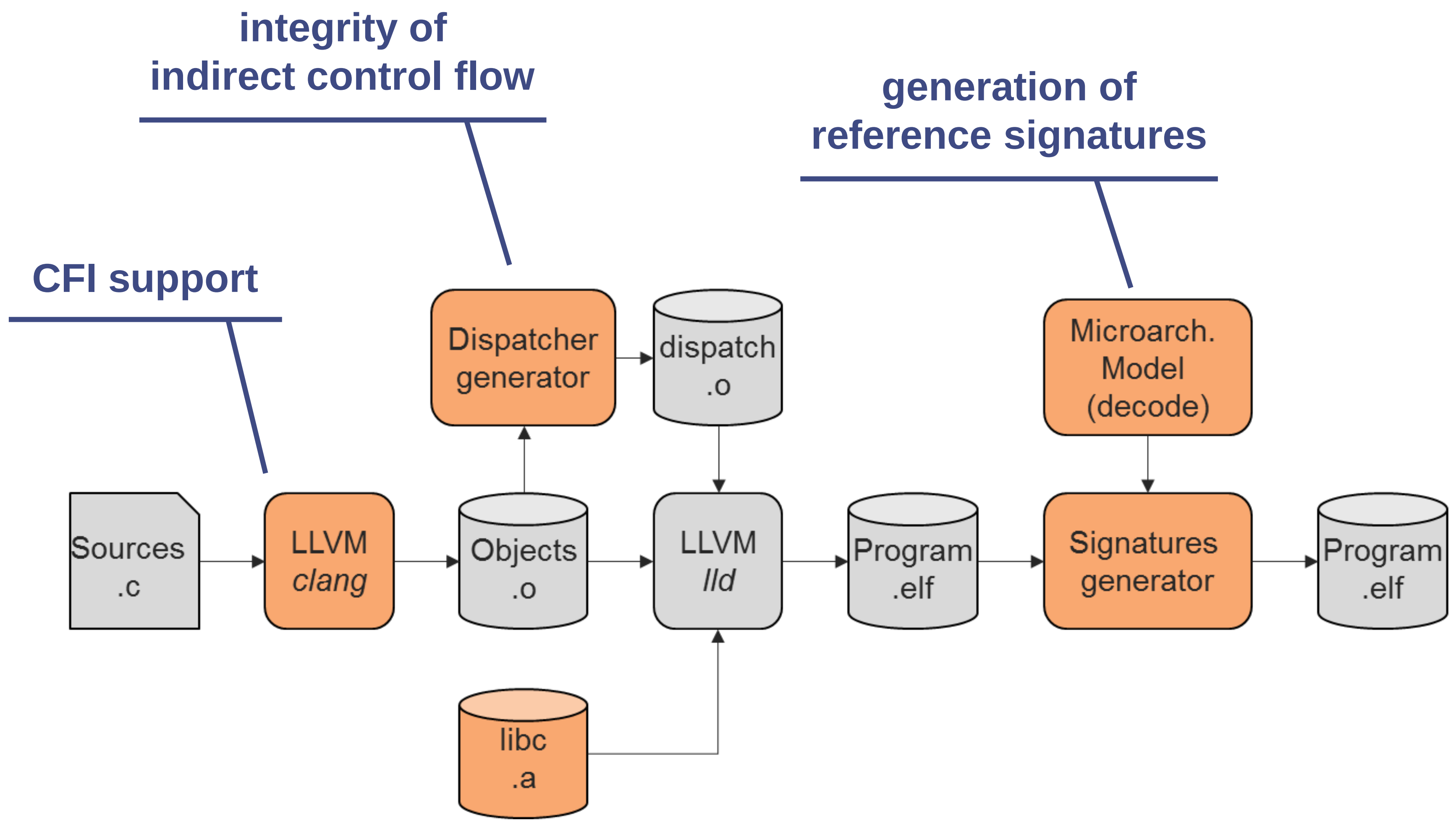
Control-signal integrity

- Integrity ensured by redundancy
  e.g. duplication
- Signals are verified at each stage (until consumption)



Code authenticity / integrity
Control-flow integrity
Control-signal integrity

MAFIA: Software support



Experimental evaluation

Methodology

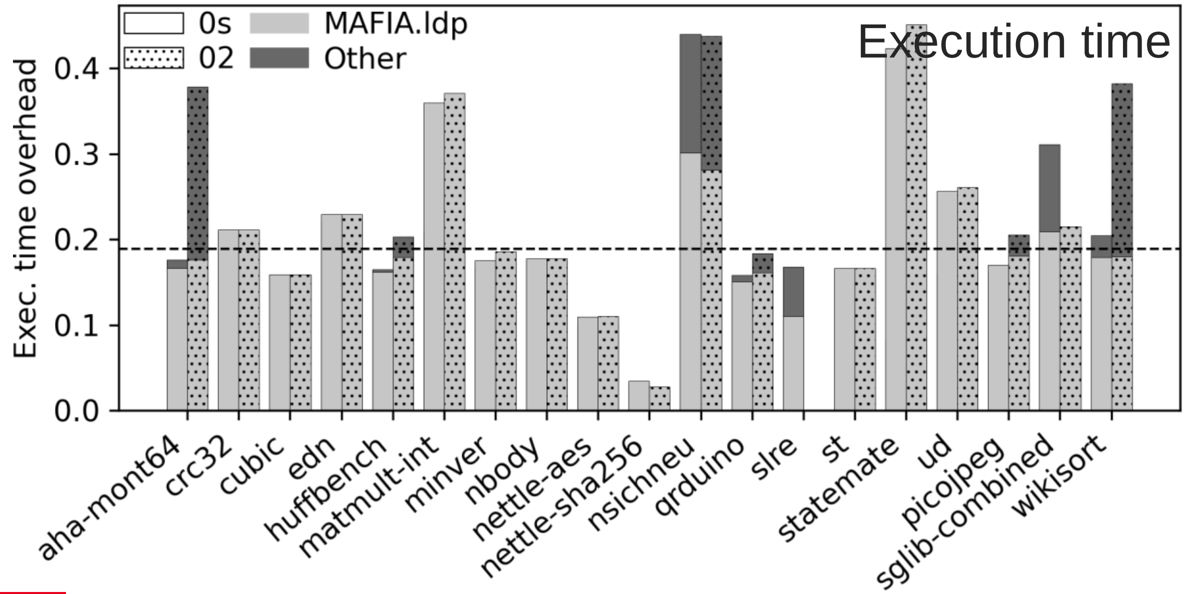
- ASIC synthesis. 22nm FDSOI @ 400MHz
- RTL simulation of Embench IoT
  - All the code is instrumented (signature continuity)
  - Verifications in each basic block of the benchmarked functions

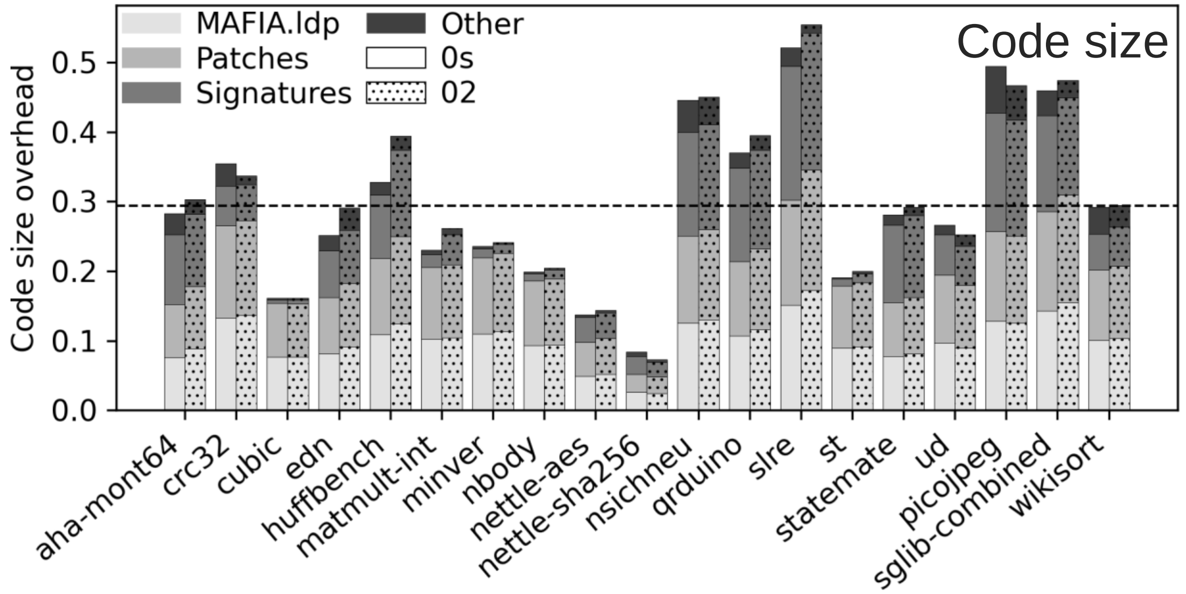
Hardware evaluation

Surface CV32E40P :	50kGE	
Surface CRC32 :	55kGE	+6,5% (+5kGE)
Surface Prince :	64kGE	+23,8% (+13kGE)

Software evaluation CRC32

Code size overhead:	+29,4%
Execution time overhead:	+18,4%





Benchmarking

Benchmarking: supporting development of security and reproducible research

Objectives

- Validate / evaluate analysis tools: security analysis results, analysis computation time
- Replicate documented attacks & countermeasures

Needs

Provide representative implementations, of variable complexity

Analysis

- Target implementation: complete, detailed (E.g. netlist + binary program)
- Attacker model: faults, attack objectives…
- Complexity metrics

Development of countermeasures

- Source code
- Targeted security: properties, coverage of each protection

Pitfalls

Consider cryptography, but not only

Analysis

- Various abstraction levels possible:
  - SW: source code, compiler IR, binary code
  - HW: RTL, netlist (back-annotated?)
- The implementation model must match the fault/attacker model
- Sensitivity to the input state

Development of countermeasures

- Impact of compiler + synthesis flow

Benchmarking processor security

Open-sourcing secure implementations and analysis tools is not enough!

Objectives

- Validate / evaluate analysis tools: security analysis results, analysis computation time
- Replicate documented attacks & countermeasures

Development of representative benchmarks?

→ HW + SW countermeasures

- **Target implementation**
  - binary code,
  - RTL / netlist,
  - Initial system state (program inputs, …)
  - Associated source code and documented toolchain.
- **Attacker model**
  - Fault model
  - Attacker capabilities: controlled / observable variables (eg.inputs), etc.
  - Attacker objectives →
    - in SW: target program address + predicates on data
    - in HW: target state
- **Attack scenarios**: instances of attacker model allowing to reproduce a vulnerability (if relevant)

FISSC: the Fault Injection and Simulation Secure Collection [Dureuil, 2016]

- SW benchmarks targeting FI
- Collection of C programs w/ multiple variants of source-level hardening
- Fault model: branch inversion
- Two attacker models

Fault Security Analysis and Verification: Challenges and New Directions

Damien Couroussé – CEA List, Univ. Grenoble Alpes, France
Karine Heydemann – Thales DIS, France & LIP6, Sorbonne Univ., France
Mathieu Jan – CEA List, Univ. Grenoble Alpes, France