MAFIA: Protecting the Microarchitecture of Embedded Systems Against Fault Injection Attacks

Thomas Chamelot, Damien Couroussé, Karine Heydemann

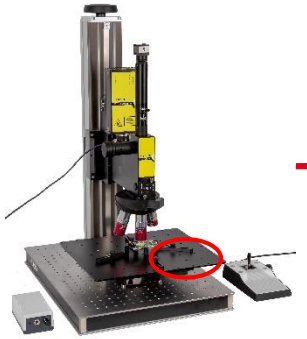
Published to IEEE TCAD 2023

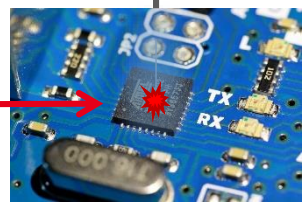
TASER 2023

Fault injection attacks everywhere

Software

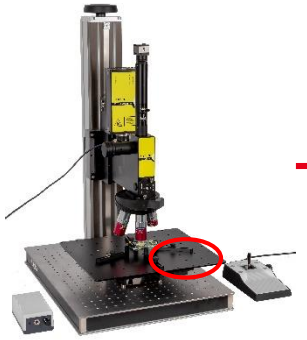
Sensitive data
/
Access control

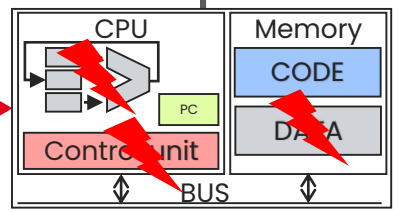




CPU
Crypto IP
ASIC

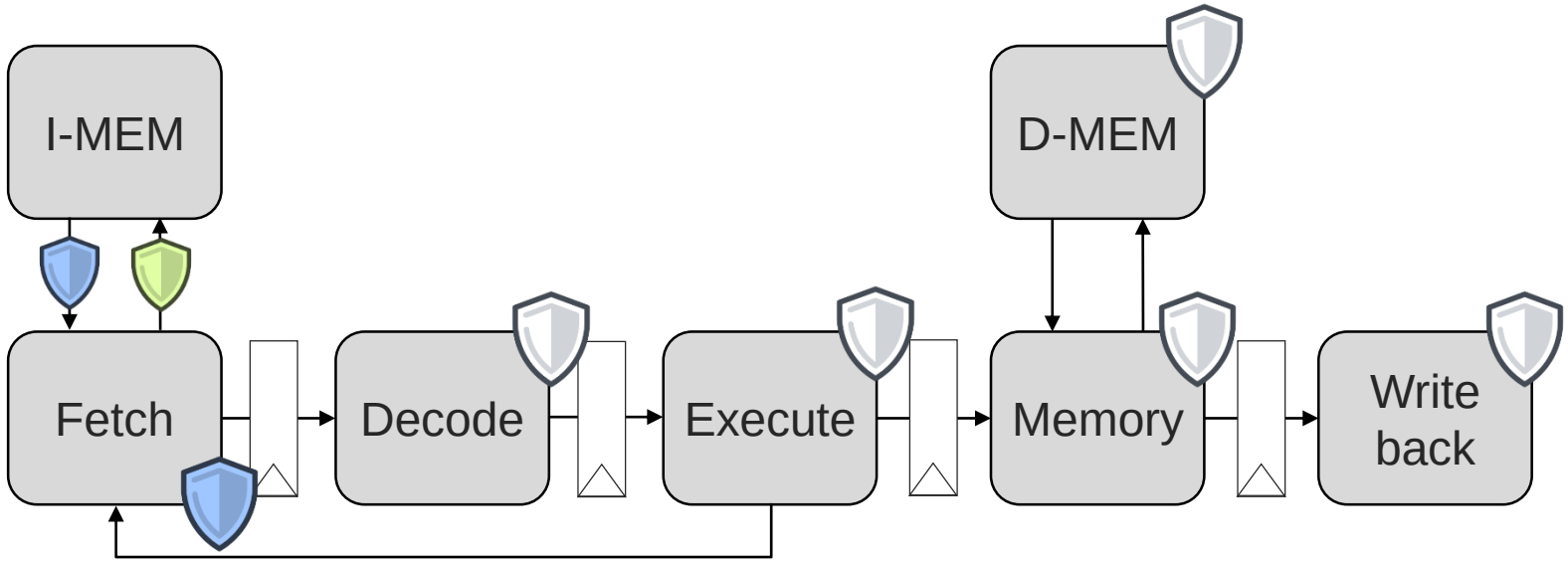
Fault injection attacks everywhere





State of the Art: Security Properties

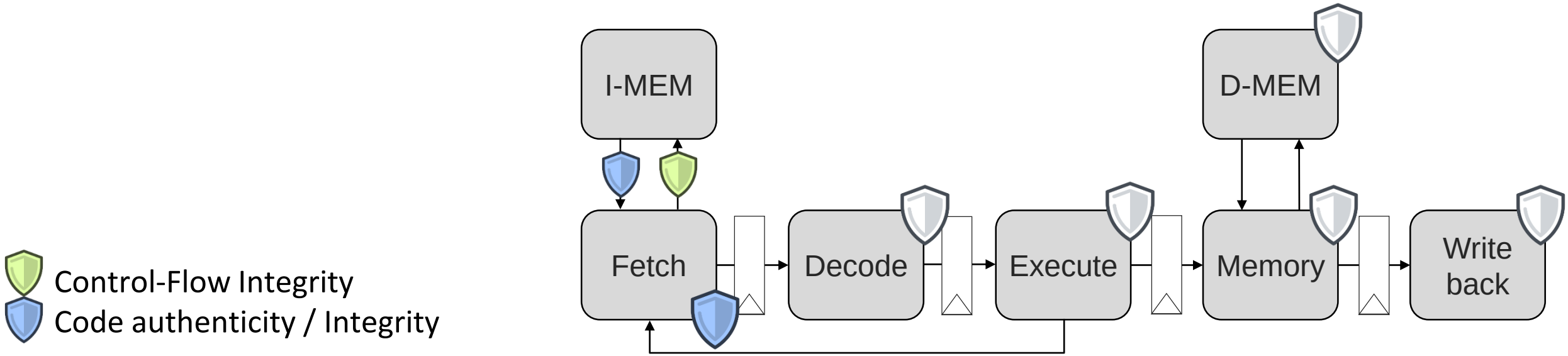
- Data integrity – not considered in this work
- Code authenticity / integrity
- Control-flow integrity
  - Direct branches / calls
  - Indirect branches / calls
  - Branchless instructions sequences (a.k.a. *basic blocks*)
    - Execution of all the instructions (e.g. no skip)
    - In correct order



Problem: faults targeting control signals

A simple loop code:

```
loop:
    addi t0, t0, #-1
    bne t0, zero, loop
```

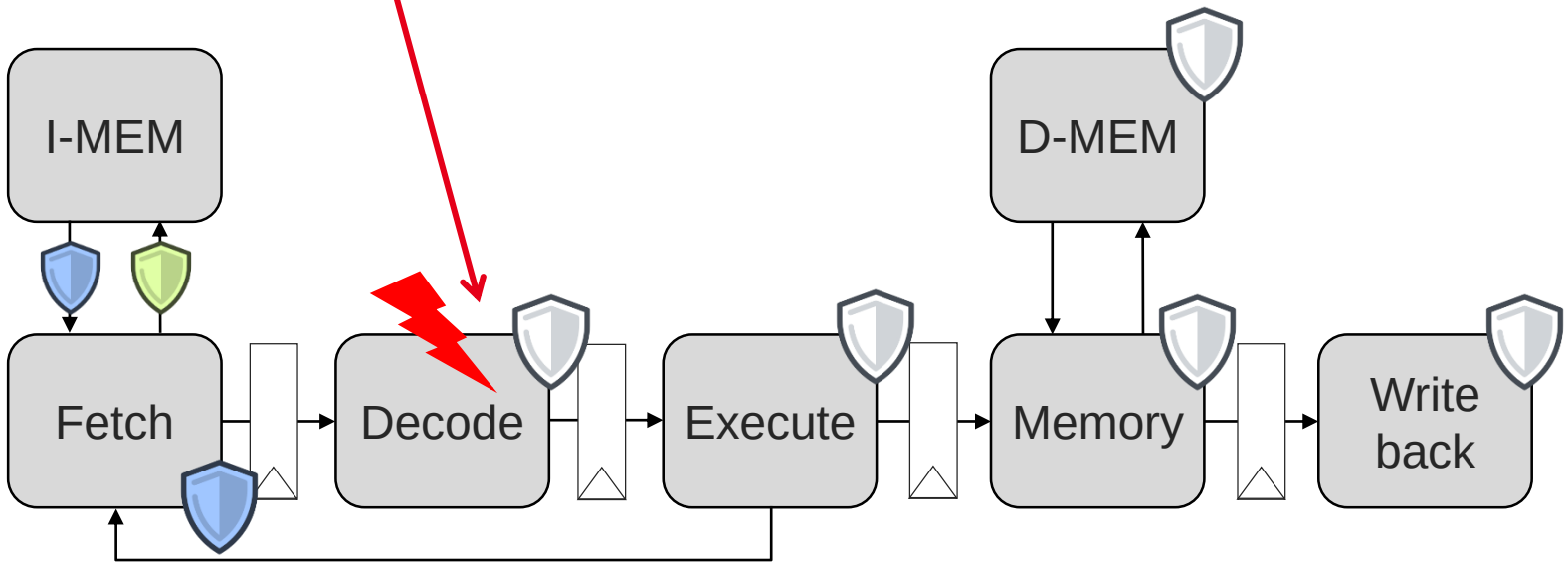


Problem: faults targeting control signals

A simple loop code:

```
loop:
    addi t0, t0, #-1
    bne t0, zero, loop
```

beq ← **Fault on instruction decode**



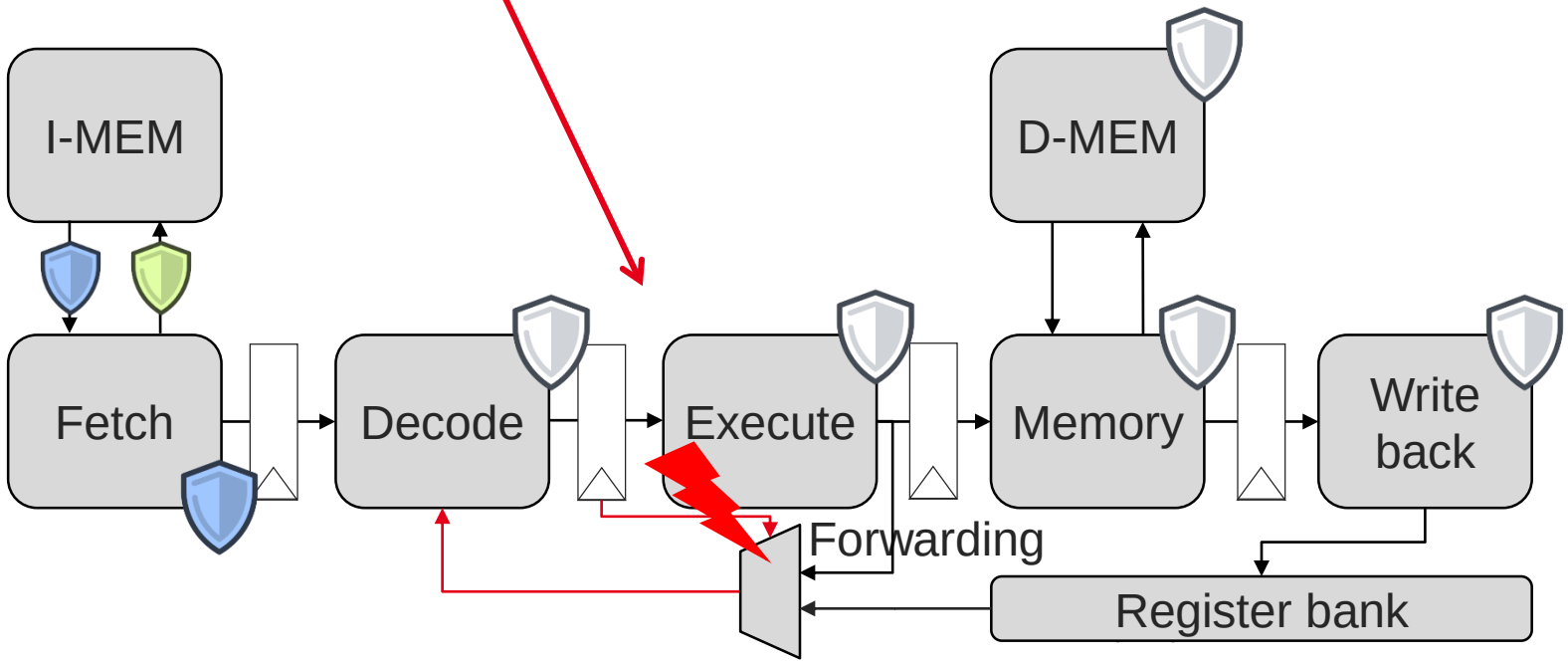
Control-Flow Integrity
Code authenticity / Integrity

Problem: faults targeting control signals

A simple loop code:

```
loop:
    addi t0[n], t0[n-1], #-1
    bne ~~t0[n]~~, zero, loop
```

t0[n-1] ← **Fault on forwarding**



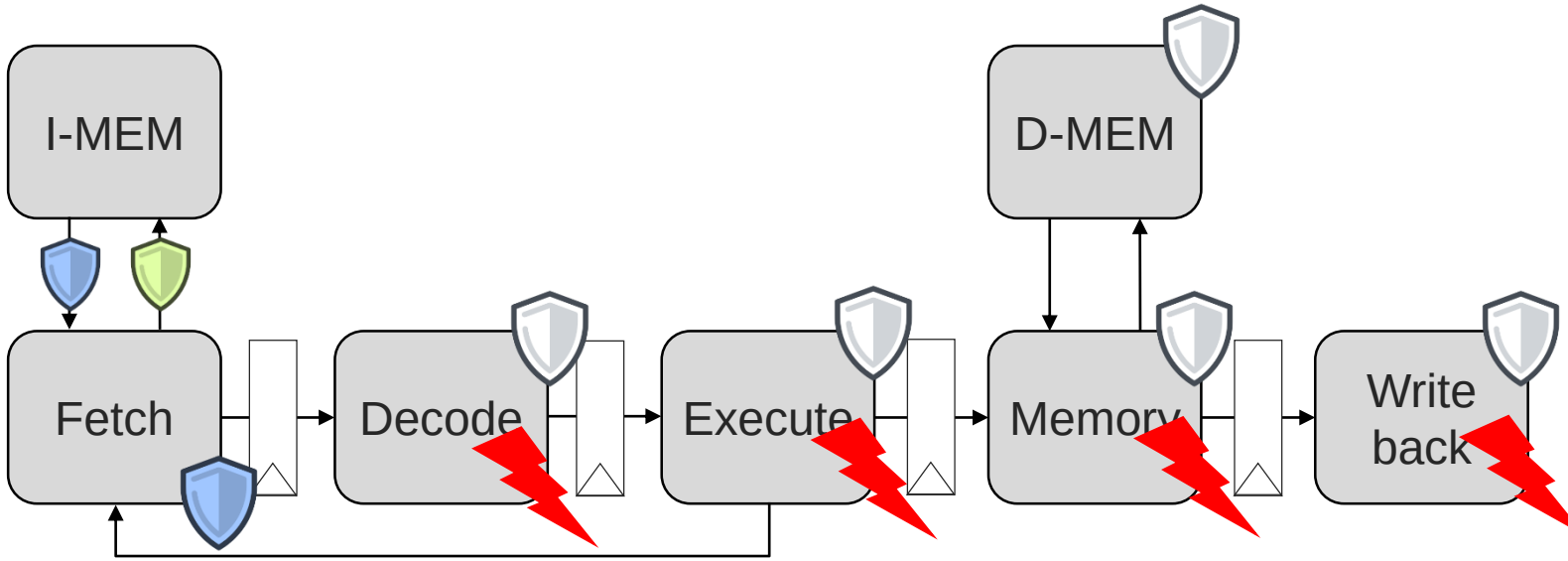
Control-Flow Integrity
Code authenticity / Integrity

Problem: faults targeting control signals

- … and many other

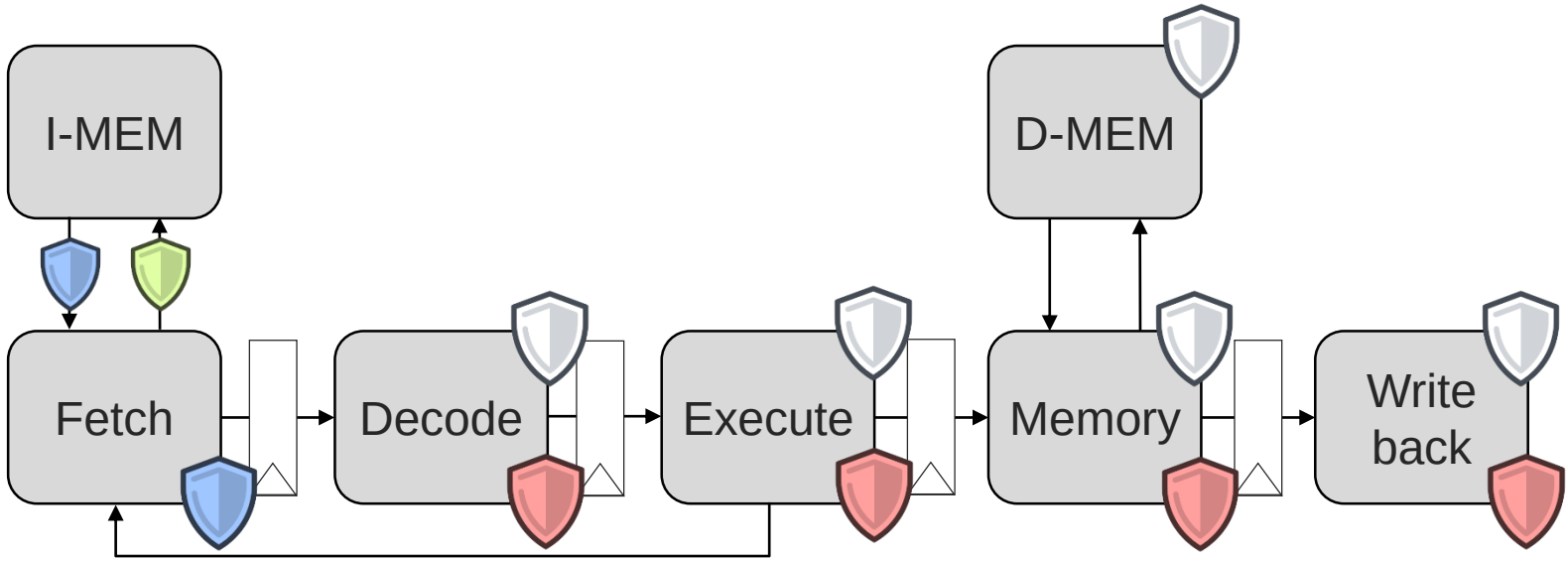
J. Laurent, V. Beroulle, C. Deleuze, F. Pebay-Peyroula, and A. Papadimitriou, "Cross-layer analysis of software fault models and countermeasures against hardware fault attacks in a RISC-V processor," Microprocessors and Microsystems, 2019.

S. Tollec, M. Asavoae, D. Couroussé, K. Heydemann, and M. Jan, "Exploration of Fault Effects on Formal RISC-V Microarchitecture Models," in FDTC, 2022.



Objective:
full protection of the processor instruction path

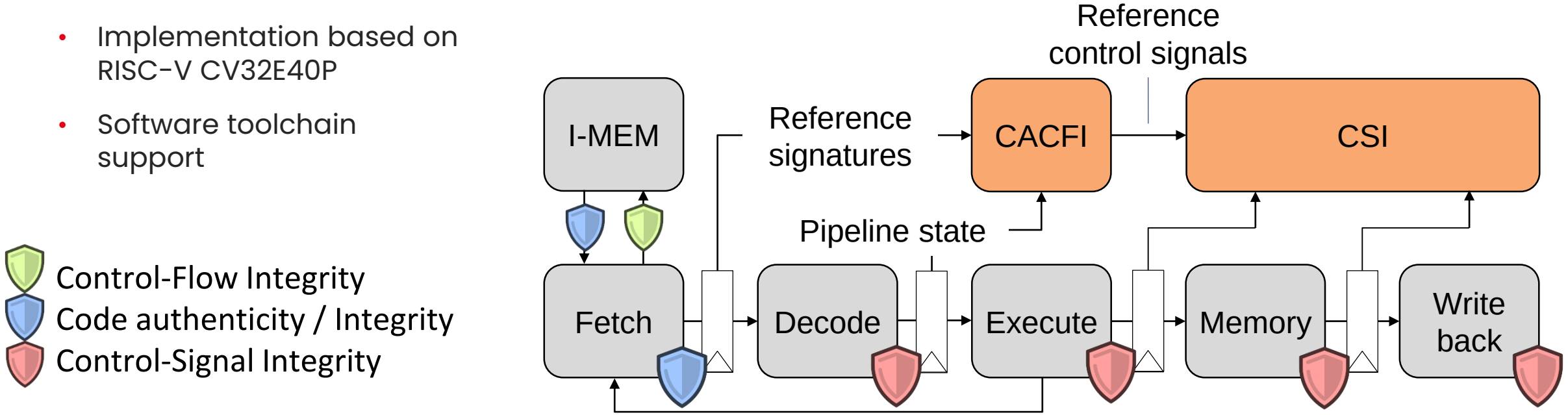
- Data integrity – not considered in this work
- Code integrity
- Control-flow integrity
  - Direct branches / calls
  - Indirect branches / calls
  - Branchless instructions sequences (a.k.a. *basic blocks*)
- **Control-signals integrity**



MAFIA: protection of the microarchitecture against fault injection attacks

- Full coverage of in-order processor instruction path
  - Control-Signal Integrity
  - Code Authenticity / Code Integrity
  - Control-Flow Integrity
- Full support of embedded software stacks:
  - Indirect function calls, interrupts
- Implementation based on RISC-V CV32E40P
- Software toolchain support

- **CACFI**: Code Authenticity and Control-Flow Integrity module
  - Signature-based
  - → **Pipeline State**
- **CSI**: Control-Signal Integrity module

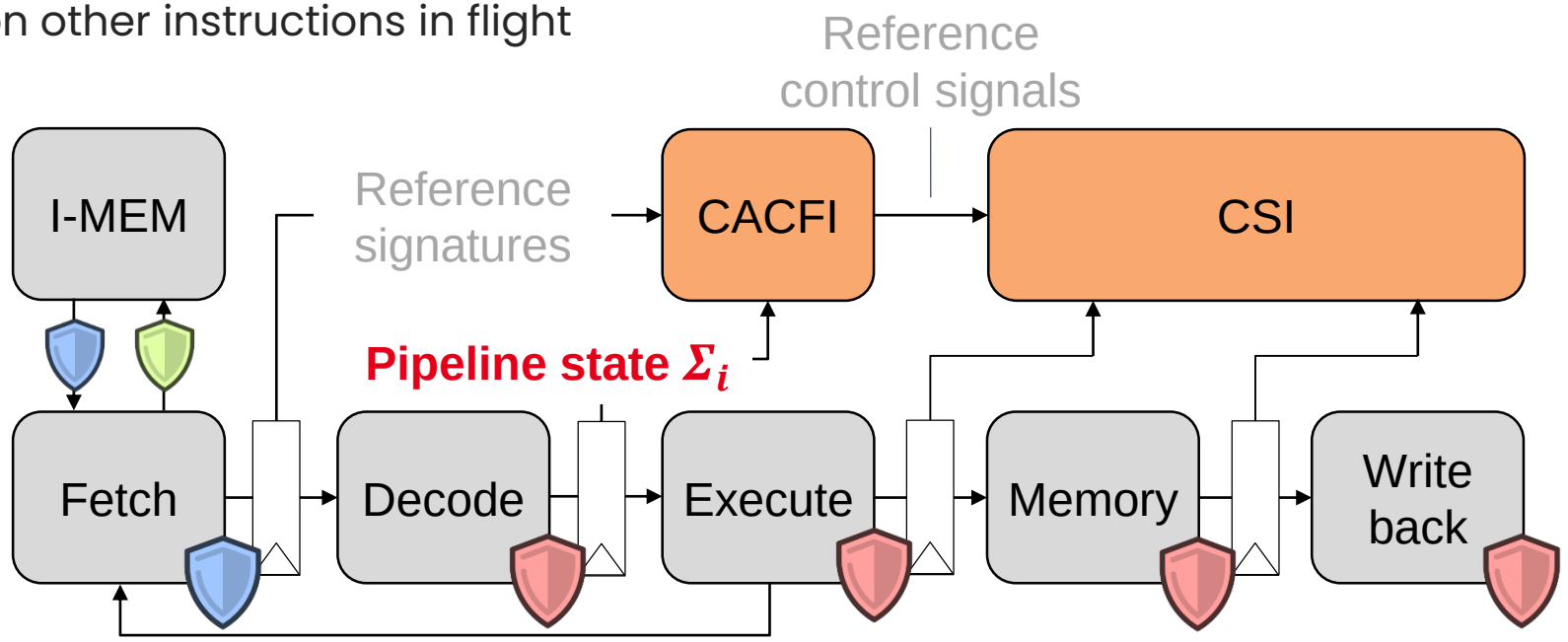


Pipeline State

Selection of control signals emitted by the decode stage

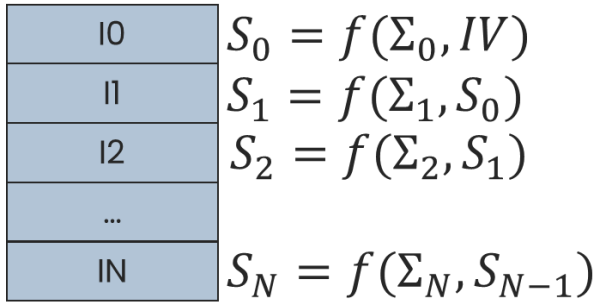
- Requirement: deterministic value at compilation time (program invariant)
- Static signals – depending only on the decoded instruction:
  - Operands, operation selection
  - Immediate values
- Dynamic signals – depending on other instructions in flight
  - Forwarding

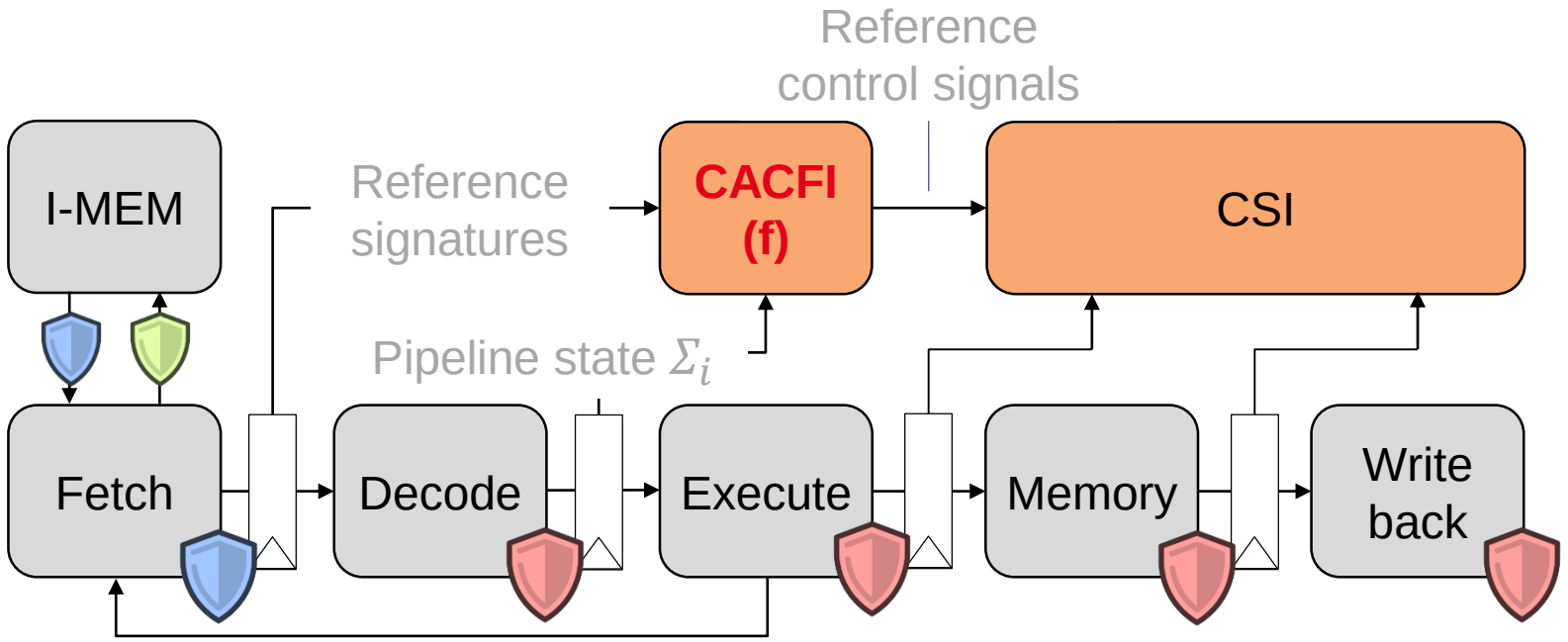
Control-Flow Integrity
Code authenticity / Integrity
Control-Signal Integrity



Signature function (CACFI)

- Pipeline state Σ_i (instruction I_i)
- Signature function f: $S_i = f(\Sigma_i, IV)$
- Based on generalized path signature analysis [Wilken et Shen, 1990]
- Security properties
  - Collision resistance
  - Error preservation
  - Non-associativity
- Hardware constraints
  - Computation in 1 CPU cycle
  - Low area
- Supports **code authenticity** or **code integrity**

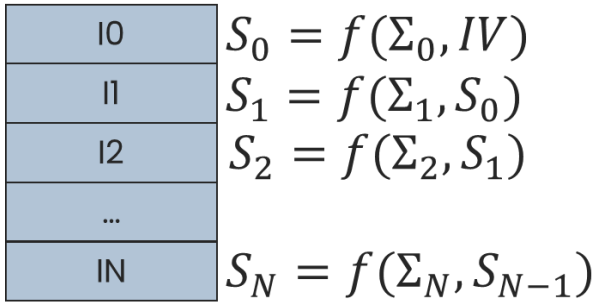




K. Wilken and J. P. Shen, "Continuous signature monitoring: Low-cost concurrent detection of processor control errors," IEEE TCAD, 1990.

Pipeline state uniqueness

- Each instruction is associated to a *unique* signature value, computed from the current pipeline state value
- We want to protect some dynamic signals, e.g. forwarding control signals
- Dynamic signals may take different values according to the execution path



BB0: forwarding

```
add t0, t0, #1
add t1, a0, t0
load t1, 0(t1)
```

BB1: forwarding

```
mov t0, #0
```

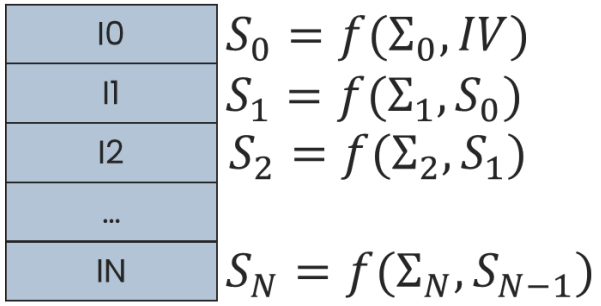
no forwarding

BB2:

```
add t0, t0, #1
…
bne t0, #16, BB2
```

Pipeline state uniqueness

- Each instruction is associated to a *unique* signature value, computed from the current pipeline state value
- We want to protect some dynamic signals, e.g. forwarding control signals
- Dynamic signals may take different values according to the execution path



BB0: forwarding

```
add t0, t0, #1
add t1, a0, t0
load t1, 0(t1)
```

BB1: no forwarding

```
mov t0, #0
nop
```

BB2:

```
add t0, t0, #1
…
bne t0, #16, BB2
```

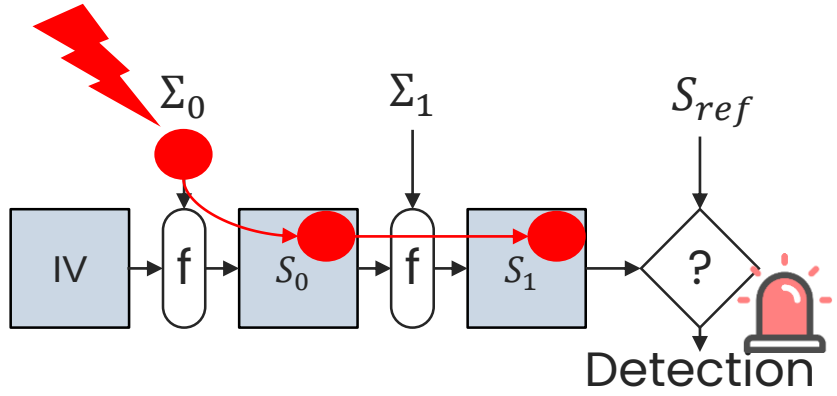
no forwarding

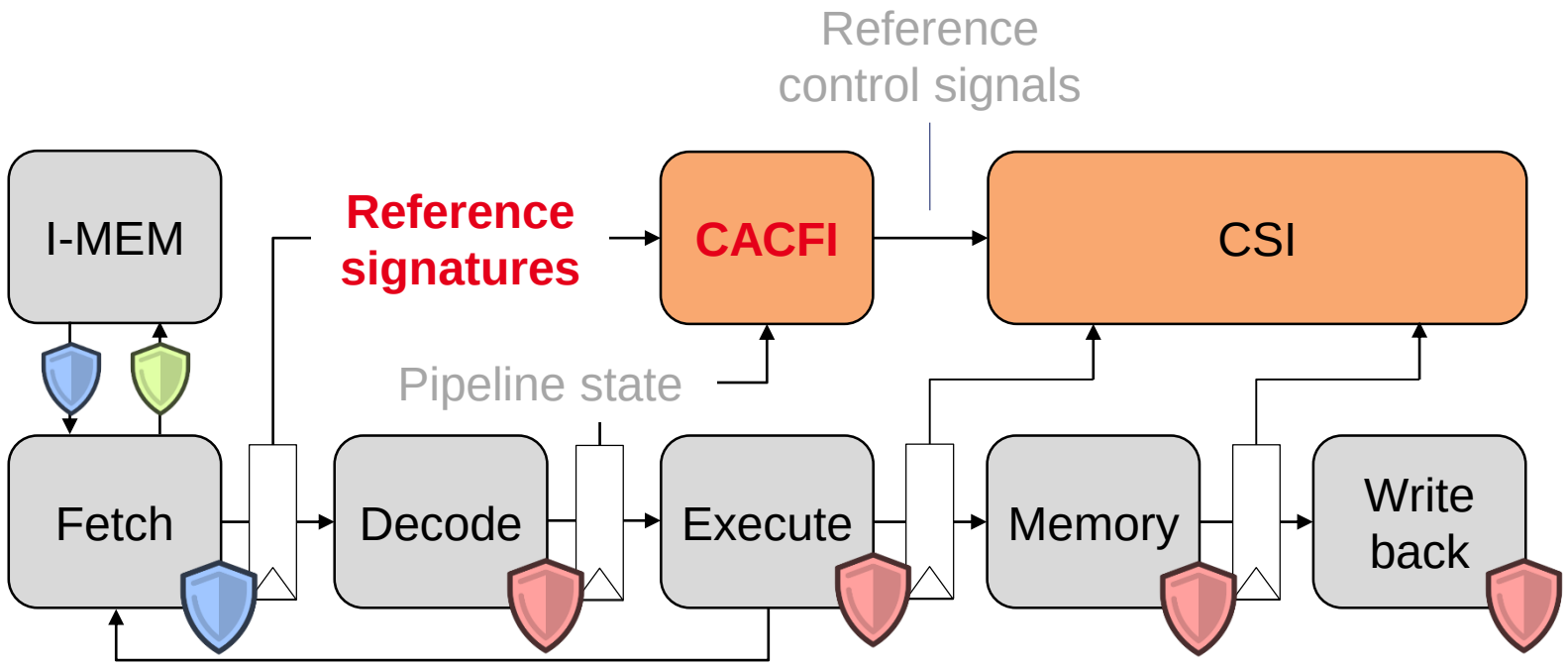
Solution

- The compiler:
  - verifies the pipeline state uniqueness;
  - inserts instructions for breaking dependencies when needed.

Signature verification (CACFI)

- 1 signature associated to each instruction
- Error preservation
  - → Verifications can be used anywhere
- Verification triggered by dedicated (control-flow) instructions
  - Load reference signature
  - Verify signature
  - Proceed with control-flow

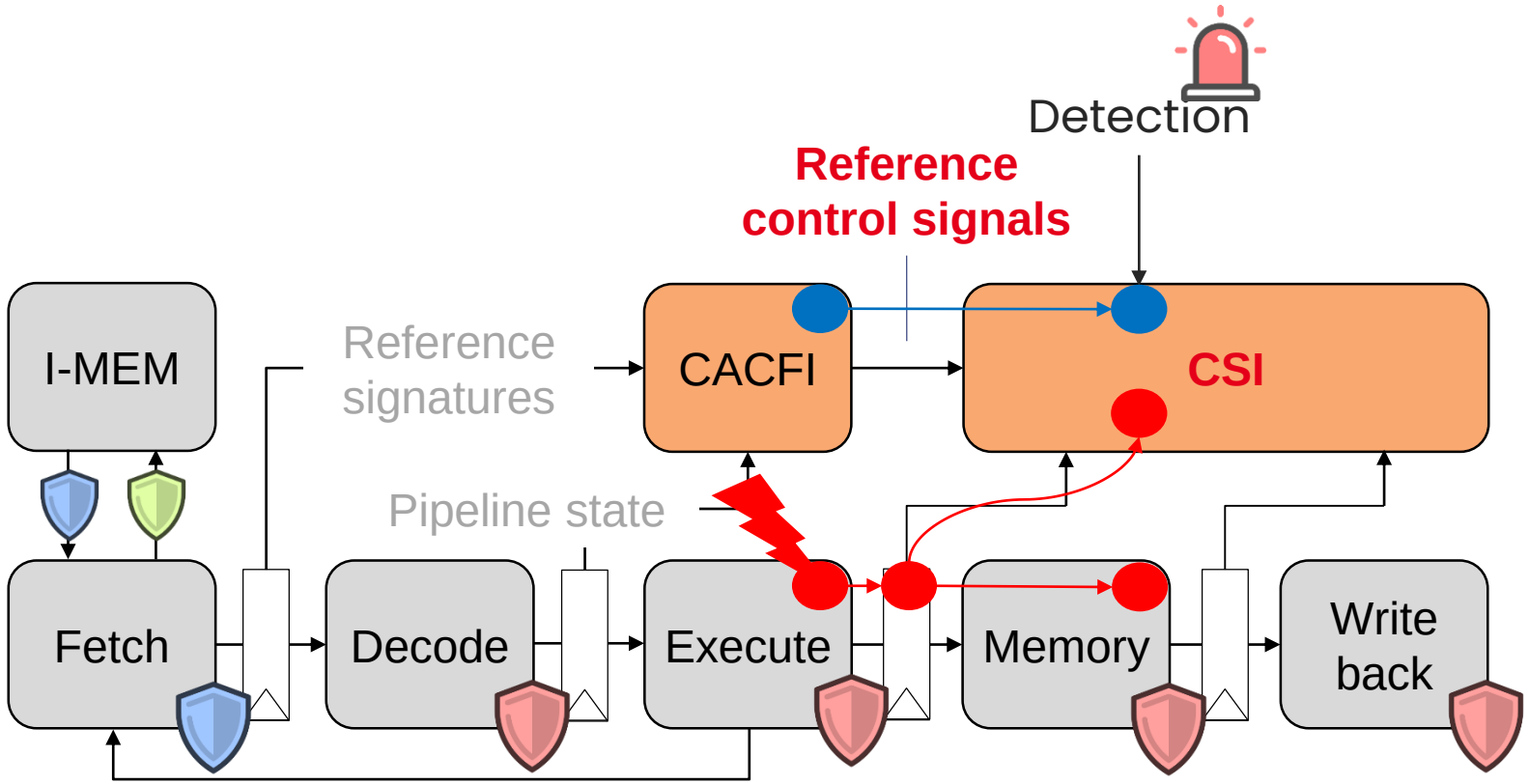




Control-Signal Integrity (CSI module)

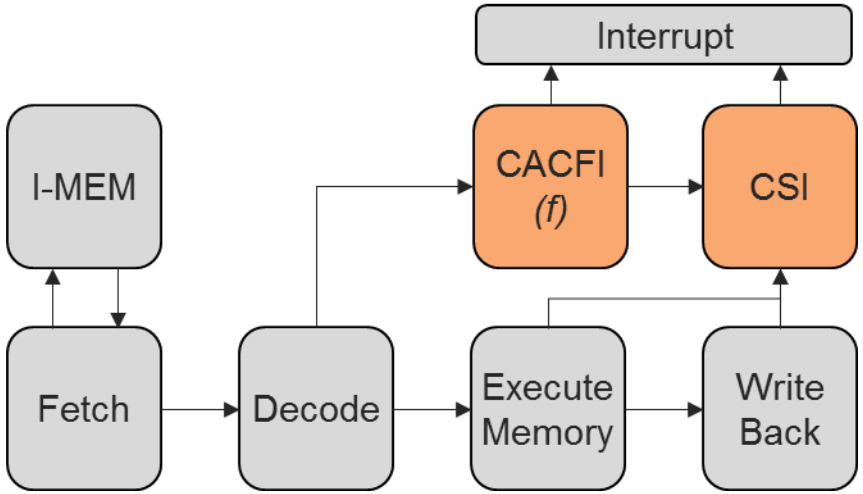
- Input: duplication of pipeline state signals, from the CACFI module
- Signals are verified at each stage (until consumption)
- Supports any redundancy scheme. E.g.:
  - Simple copy
  - Complementary copy
  - XOR (signal, constant)

Control-Flow Integrity
Code authenticity / Integrity
Control-Signal Integrity



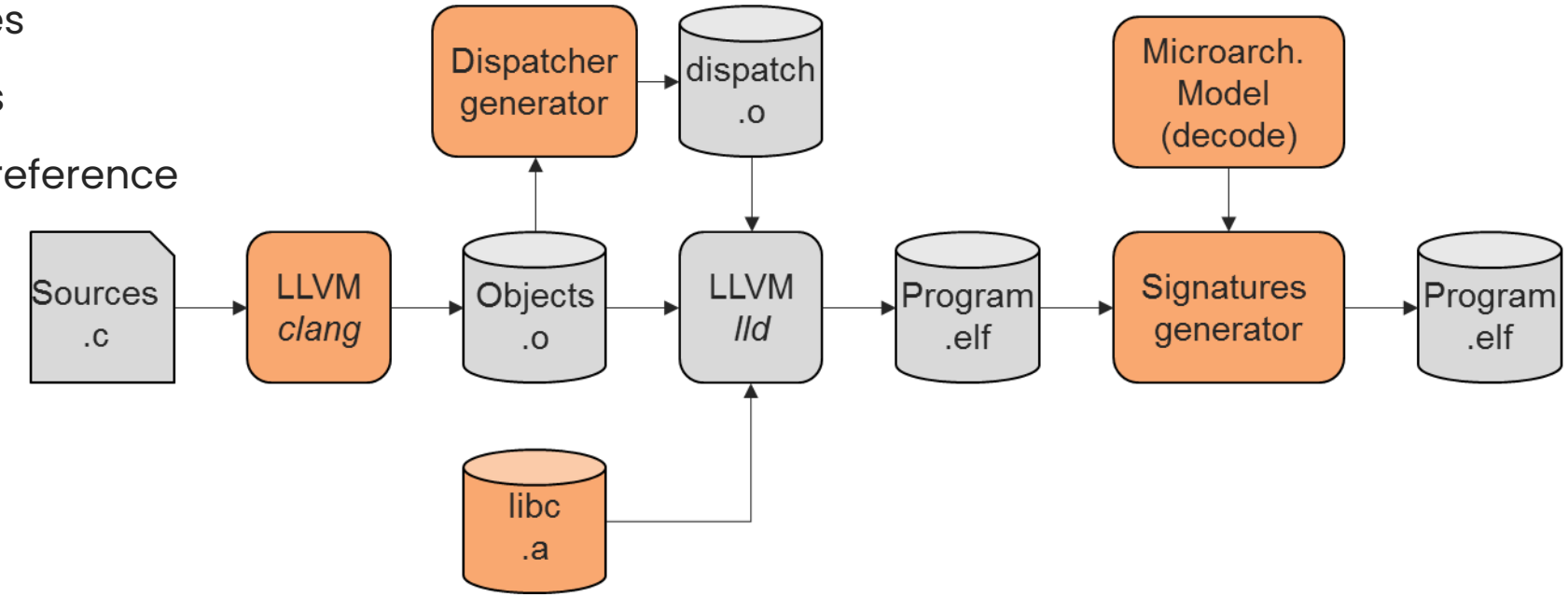
Hardware implementation

- Processor: RISC-V 32-bits CV32E40P
  - ISA: RV32I(MC)
  - 4-stage, in-order pipeline
- Two implementations with different signature functions f:
  - CRC32 – code integrity, detects up to 8 bit-flips
  - CBC-MAC Prince – code authenticity
- ASIC synthesis 22nm FDSOI @ 400MHz
- Formal verification of the pipeline state coverage



Software support

- Dedicated instructions
  - Loading of signature patches
  - Signature verifications
- Pipeline state uniqueness: preventing control signals variability due to e.g. forwarding
- Removal of indirect branches
- Dispatchers for indirect calls
- Generation of patches and reference signatures



Experimental evaluation

Methodology

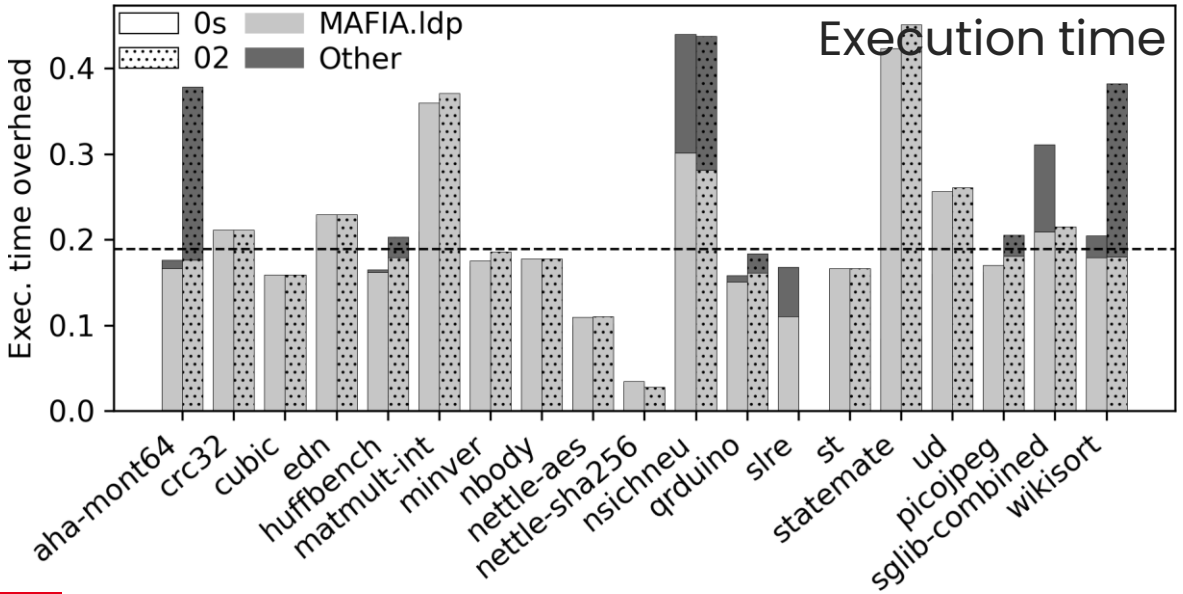
- ASIC synthesis. 22nm FDSOI @ 400MHz
- RTL simulation of Embench IoT
  - All the code is instrumented (signature continuity)
  - Verifications in each basic block of the benchmarked functions

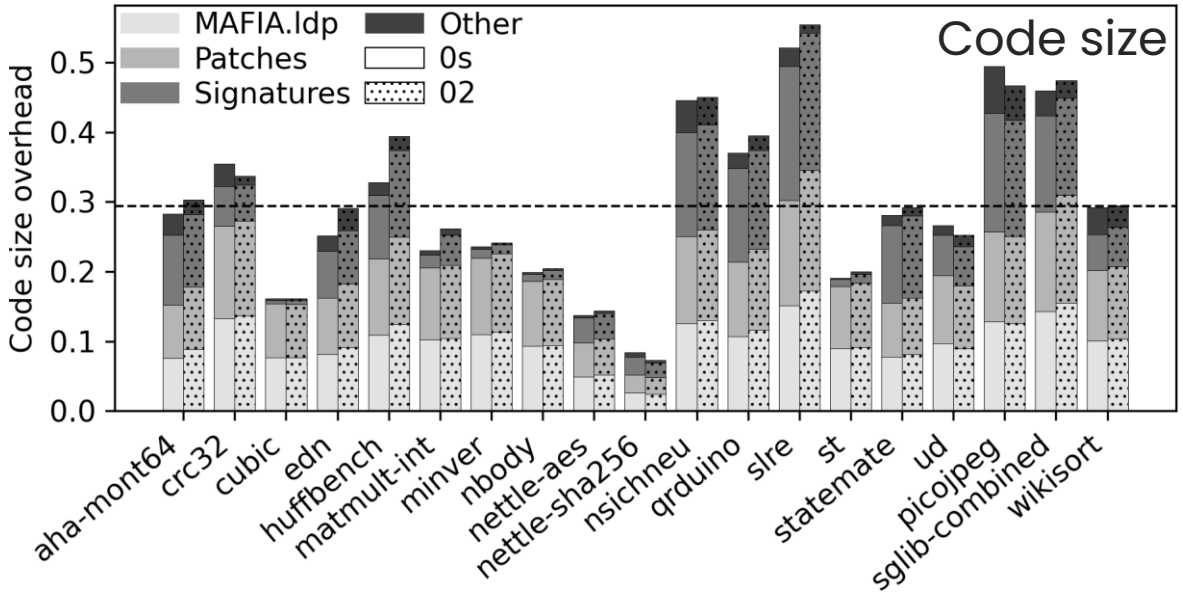
Hardware evaluation

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

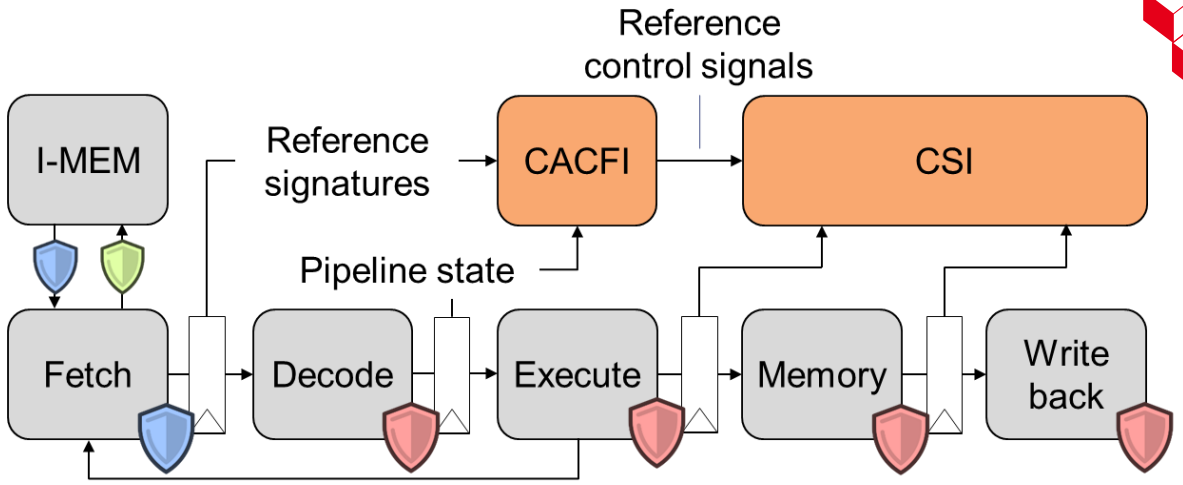
Software evaluation CRC32

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





cea list



Merci

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), vol. accepted for publication, 2023.
https://doi.org/10.1109/TCAD.2023.3276507

damien.courousse@cea.fr

Open positions! ☺

CHES poster!

anr THALES SORBONNE UNIVERSITÉ