FAME: Fault-attack Aware Microprocessor Extensions for Hardware Fault Detection and Software Fault Response

Bilgiday Yuce, Nahid Farhady Ghalaty, Chinmay Deshpande, Conor Patrick, Leyla Nazhandali, Patrick Schaumont

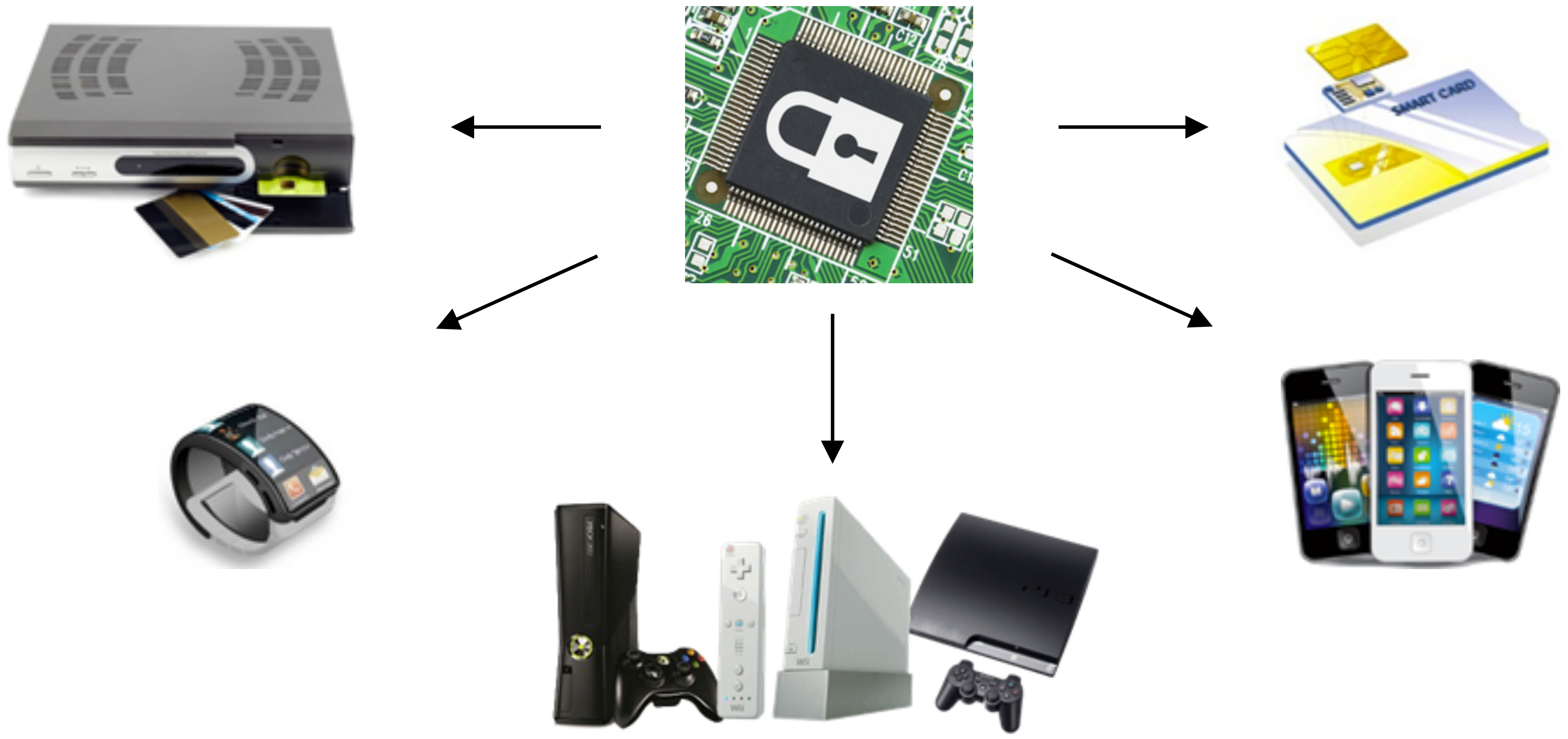
Virginia Tech

HASP 2016

This research was supported in part through the NSF Grant 1441710, and in part through the SRC.

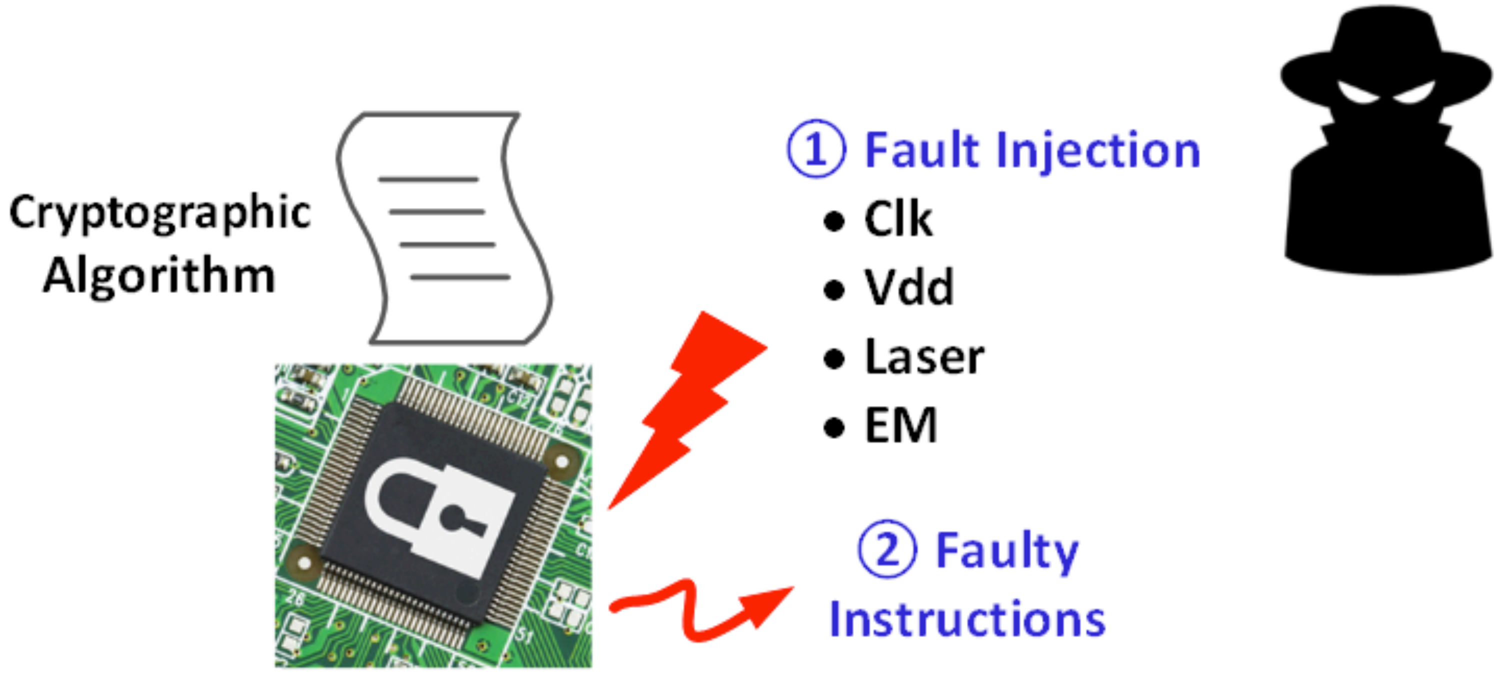
Embedded Systems are Everywhere

- Threat model expands from software into hardware.



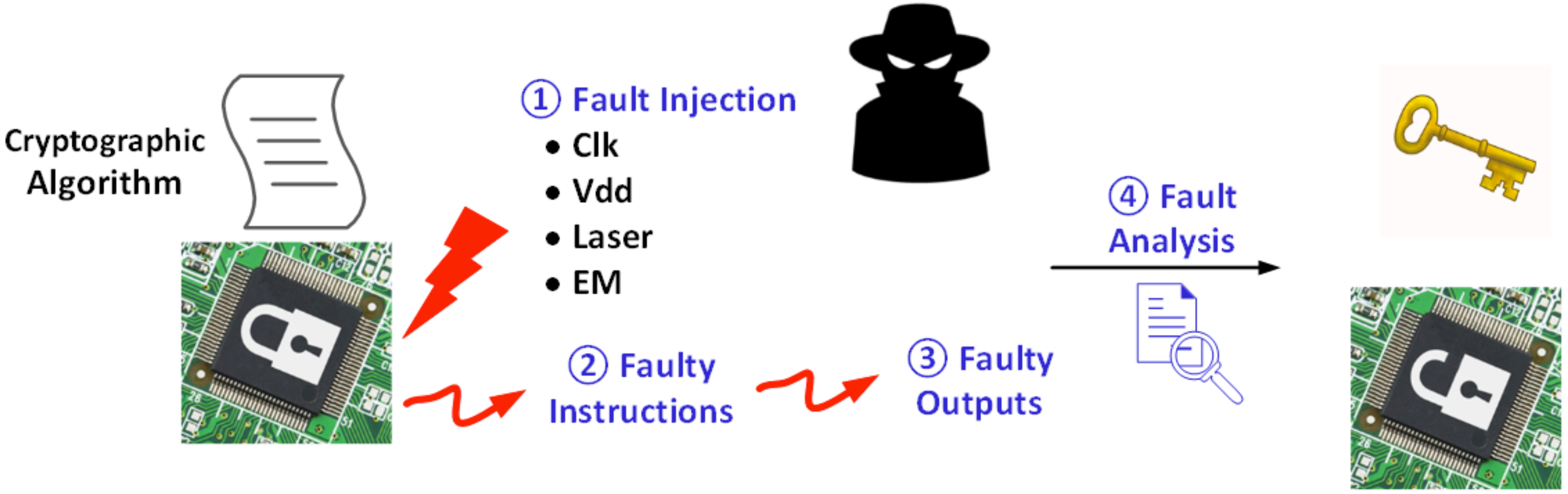
Fault Attacks on Embedded Software

- Inject engineered faults into with a specific security objective in mind



Fault Attacks on Embedded Software

- Inject engineered faults with a specific security objective in mind
- Analyze fault response of the software to break the security



Why are Faults a Security Issue?

- May enable bypass of security checks/actions

```
int   pinCheck (int userPin)   {
          if (userPin == devicePin)   {
             unlockPhone();
             return 0;
          } else   {
             lockPhone();
             return -1;
          }
}
```

Instruction skip

Phone is unlocked
even if userPin is wrong

Why are Faults a Security Issue?

- May enable leakage of secret information
- Even correct output may leak the secret information.

```
// Elliptic Curve Cryptography
//  (Simplified) Scalar Multiplication
…
Q[0] = 2Q[0];
Q[1] = Q[0] + P;
Q[0] = Q[key_bit];
return Q[0];
```

Inject a fault into P

If the fault does not affect the output, key_bit is 0.

How should We Handle Faults?

- **Fault handling** can be separated into Fault Detection and Fault Response
- Fault Detection:
  - It must be low-latency $\longrightarrow$ Hardware Fault Detection
  - It must be hard-to-bypass

How should We Handle Faults?

- Fault handling can be separated into Fault Detection and Fault Response
- Fault Detection:
  - It must be low-latency
  - It must be hard-to-bypass

$\longrightarrow$ Hardware Fault Detection

- Fault Response:
  - It must be application-specific
  - It must be adaptive

$\longrightarrow$ Software Fault Response

How should We Handle Faults?

- Fault handling can be separated into Fault Detection and Fault Response
- Fault Detection:
  - It must be low-latency
  - It must be hard-to-bypass

$\longrightarrow$ Hardware Fault Detection

- Fault Response:
  - It must be application-specific
  - It must be adaptive

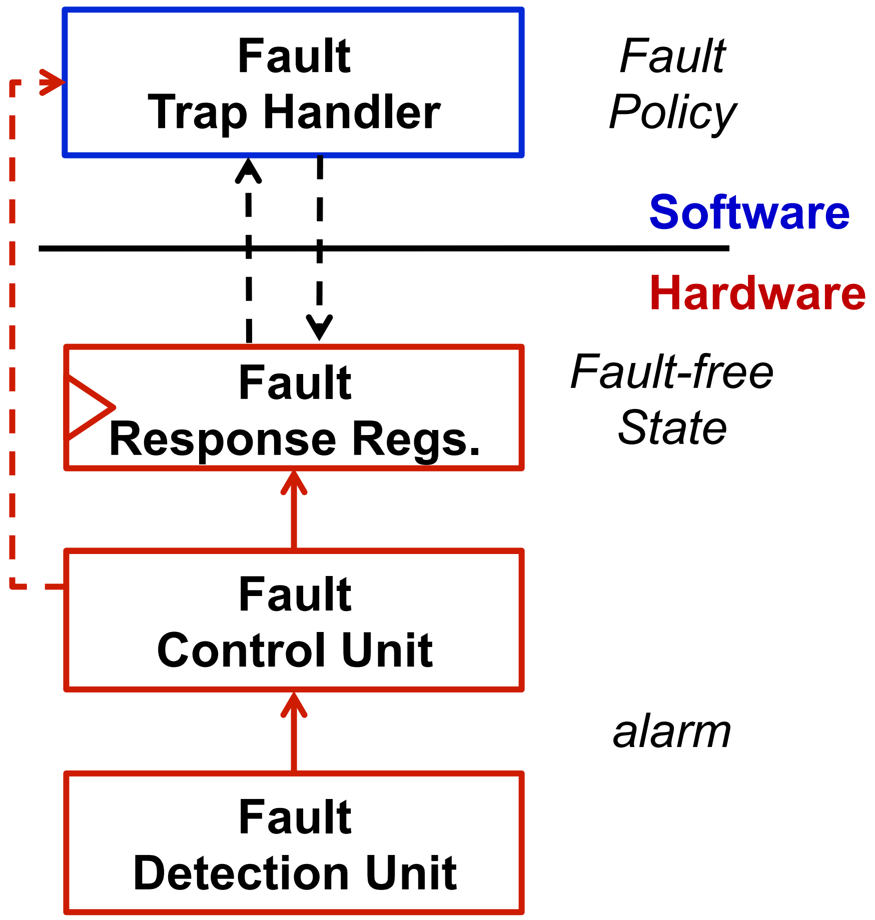
$\longrightarrow$ Software Fault Response

- **Communication** between Fault Detection and Response
  - SW Fault Response must be aware of HW fault status
  - Processor HW must have features to support fault-attack resistant execution of SW Fault Response

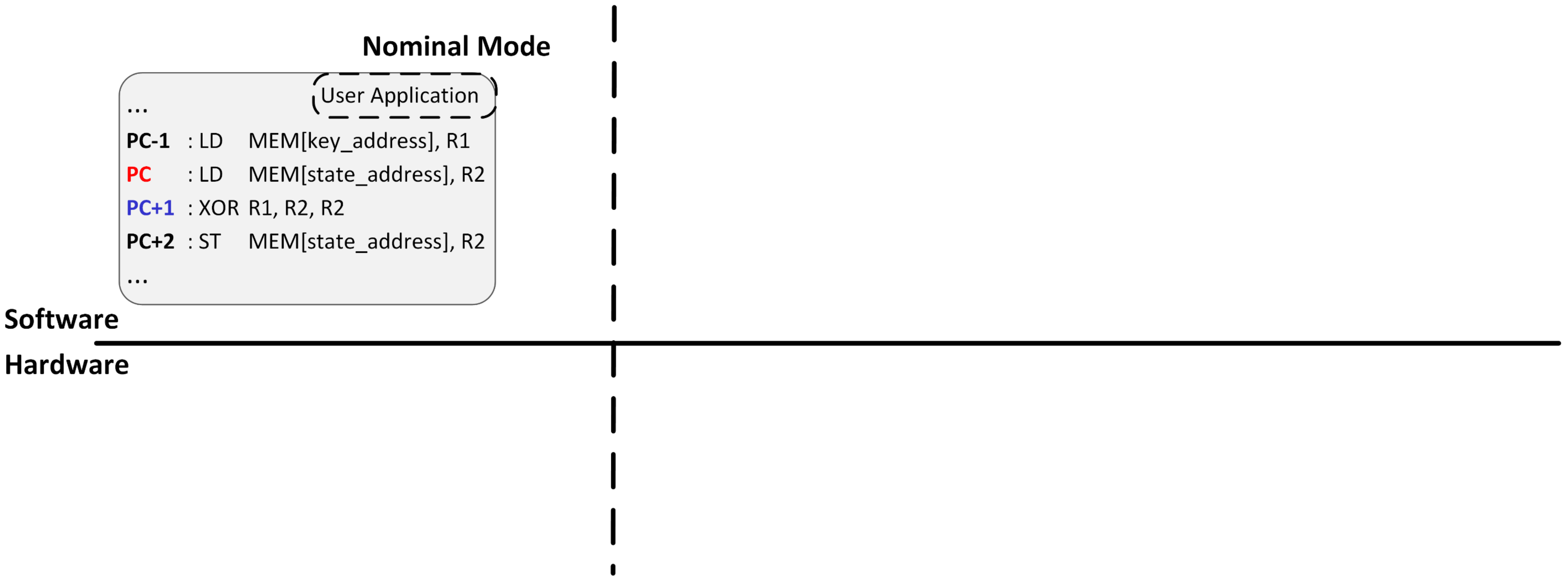
$\longrightarrow$ **HW/SW Approach**

Overview of FAME

- Combination of HW/SW extensions
- Captures faults using fault detectors in HW level
- User-defined fault policy in SW level
- Fault-attack resistant execution of fault policy
- Status & recovery information to fault handler

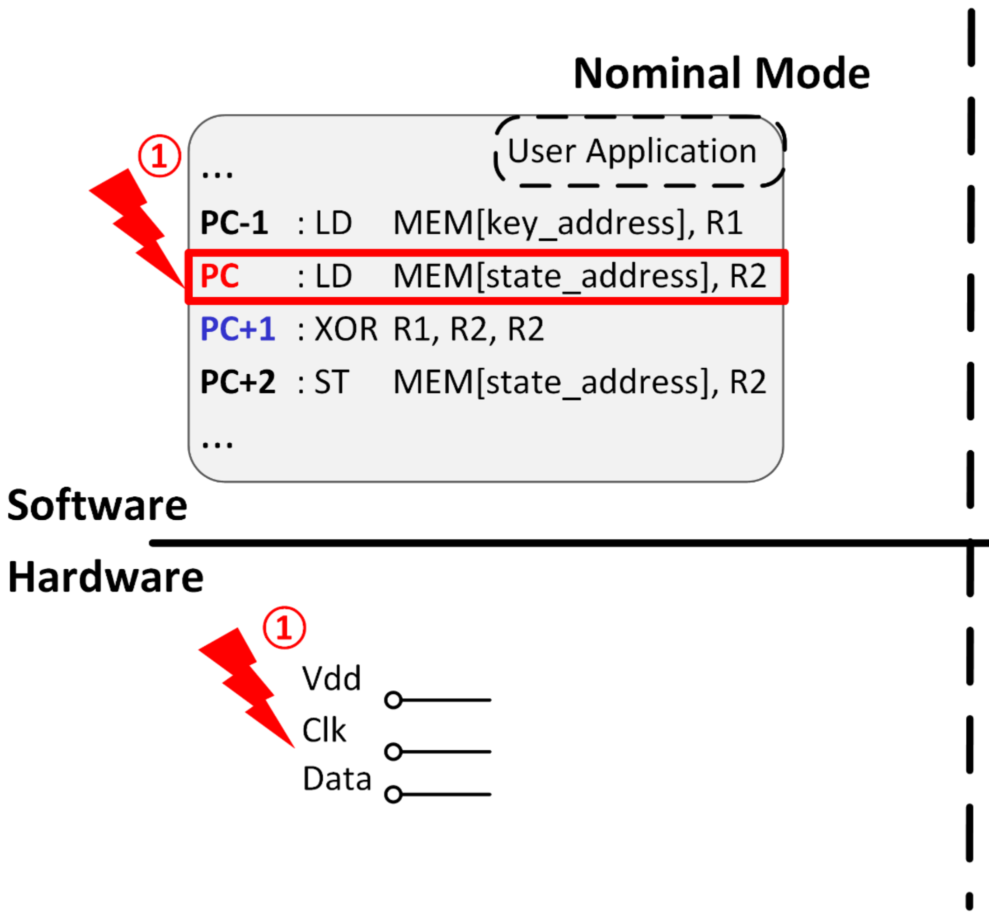


Operation of FAME - 1



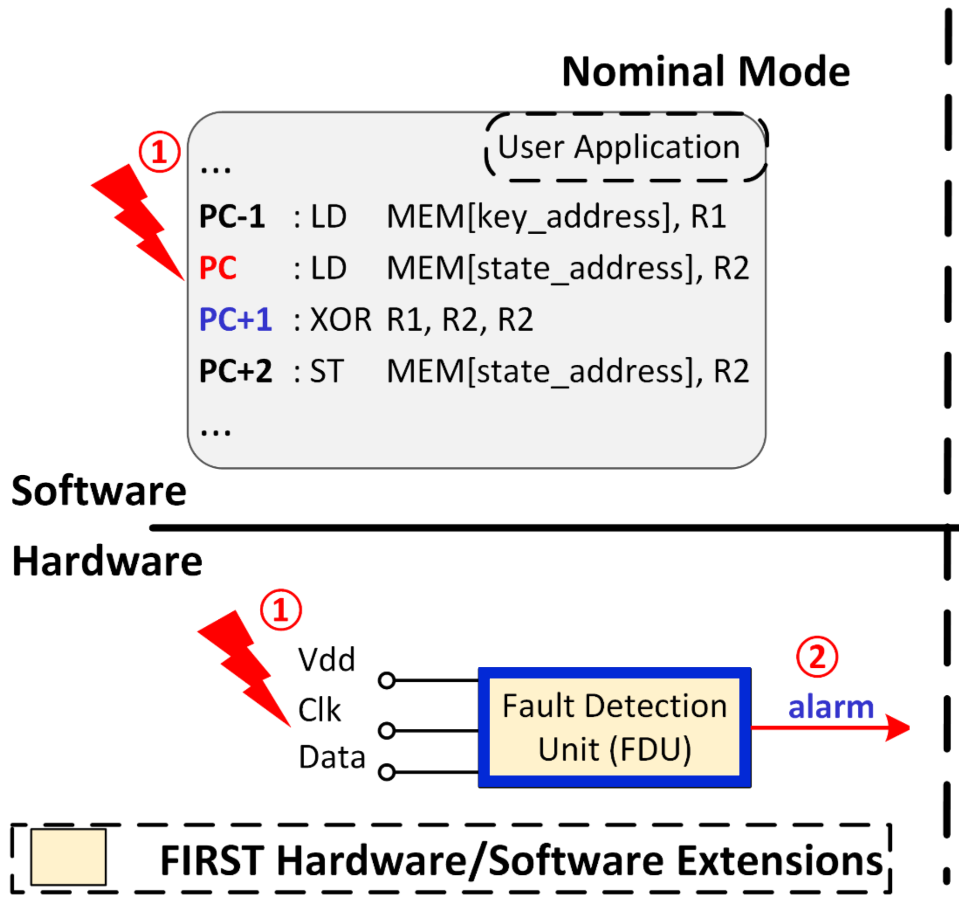
```
...
PC-1  : LD   MEM[key_address], R1
PC    : LD   MEM[state_address], R2
PC+1  : XOR  R1, R2, R2
PC+2  : ST   MEM[state_address], R2
...
```

Operation of FAME - 2

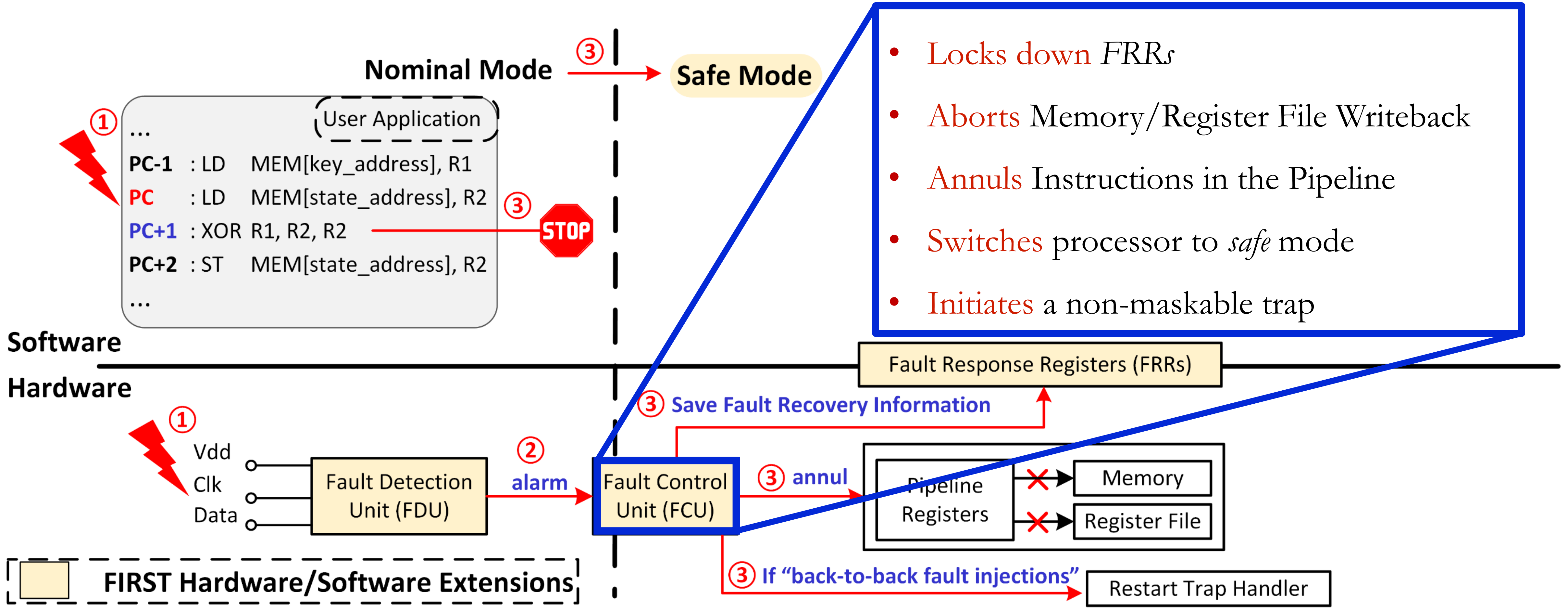


Nominal Mode
User Application
①
...
PC-1 : LD MEM[key_address], R1
PC : LD MEM[state_address], R2
PC+1 : XOR R1, R2, R2
PC+2 : ST MEM[state_address], R2
...
Software
Hardware
①
Vdd
Clk
Data

Operation of FAME - 3


Nominal Mode
User Application
...
PC-1 : LD MEM[key_address], R1
PC : LD MEM[state_address], R2
PC+1 : XOR R1, R2, R2
PC+2 : ST MEM[state_address], R2
...
Software
Hardware
①
①
Vdd
Clk
Data
Fault Detection Unit (FDU)
②
alarm
FIRST Hardware/Software Extensions

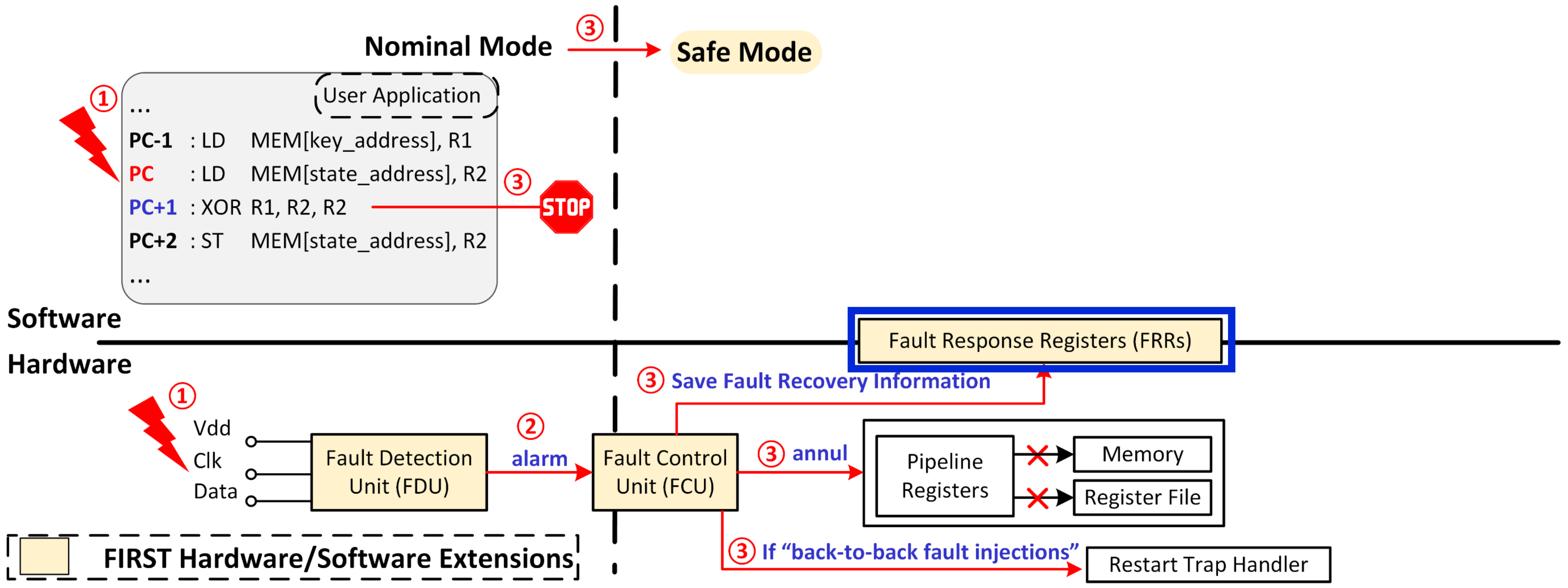
Operation of FAME - 4



Nominal Mode
③
Safe Mode
①
User Application
...
PC-1 : LD MEM[key_address], R1
PC : LD MEM[state_address], R2
PC+1 : XOR R1, R2, R2
PC+2 : ST MEM[state_address], R2
...
③
STOP
Software
Hardware
Fault Response Registers (FRRs)
③ Save Fault Recovery Information
①
Vdd
Clk
Data
Fault Detection Unit (FDU)
② alarm
Fault Control Unit (FCU)
③ annul
Pipeline Registers
Memory
Register File
③ If "back-to-back fault injections"
Restart Trap Handler
FIRST Hardware/Software Extensions

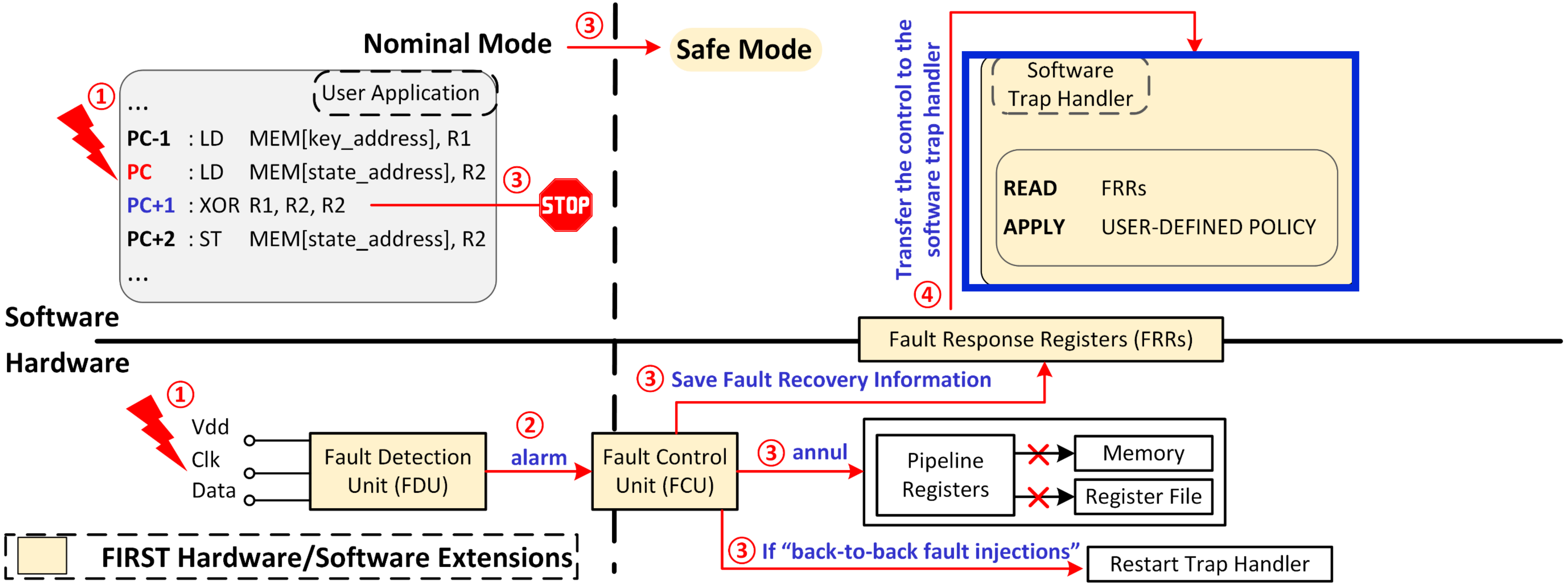
- Locks down *FRRs*
- Aborts Memory/Register File Writeback
- Annuls Instructions in the Pipeline
- Switches processor to *safe* mode
- Initiates a non-maskable trap

Operation of FAME - 4

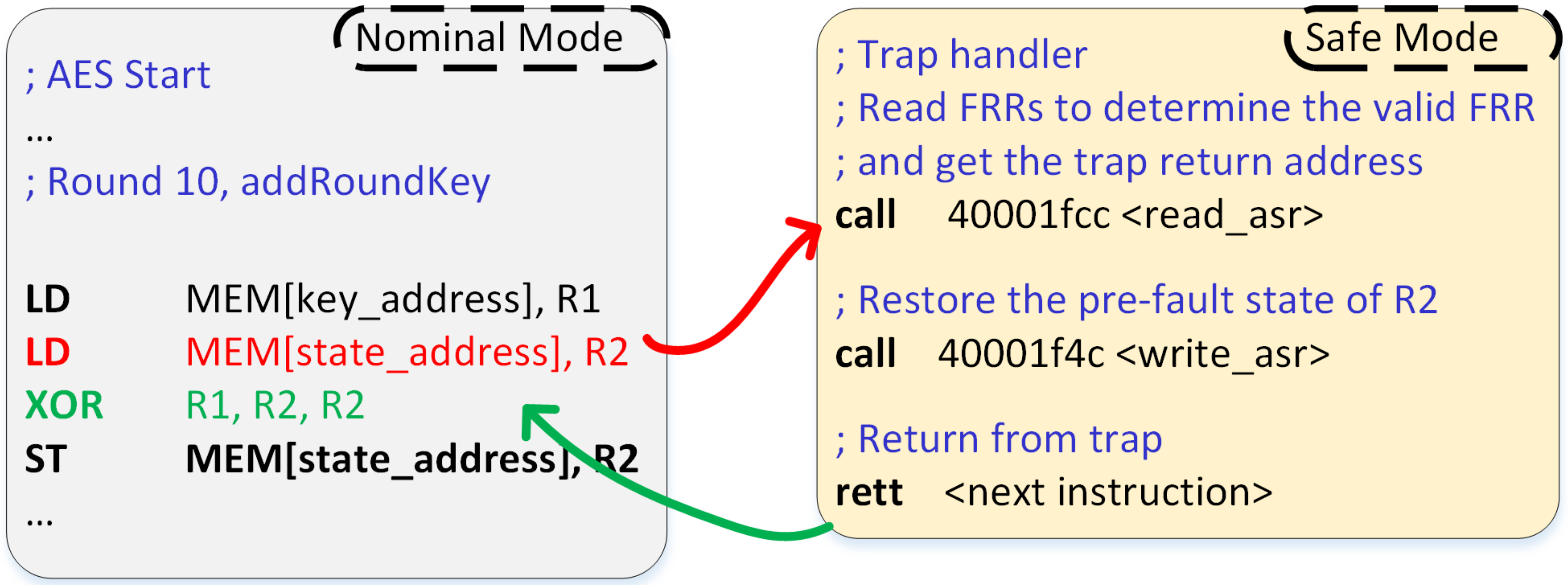


③
Nominal Mode
Safe Mode
①
User Application
...
PC-1 : LD MEM[key_address], R1
PC : LD MEM[state_address], R2
③
PC+1 : XOR R1, R2, R2
STOP
PC+2 : ST MEM[state_address], R2
...
Software
Hardware
Fault Response Registers (FRRs)
③ Save Fault Recovery Information
①
Vdd
Clk
Data
Fault Detection Unit (FDU)
②
alarm
Fault Control Unit (FCU)
③ annul
Pipeline Registers
Memory
Register File
③ If "back-to-back fault injections"
Restart Trap Handler
FIRST Hardware/Software Extensions

Operation of FAME - 5


③
Nominal Mode
Safe Mode
Transfer the control to the software trap handler
Software Trap Handler
READ FRRs
APPLY USER-DEFINED POLICY
①
User Application
...
PC-1 : LD MEM[key_address], R1
PC : LD MEM[state_address], R2
PC+1 : XOR R1, R2, R2
PC+2 : ST MEM[state_address], R2
...
③
STOP
④
Software
Hardware
Fault Response Registers (FRRs)
③ Save Fault Recovery Information
①
Vdd
Clk
Data
Fault Detection Unit (FDU)
② alarm
Fault Control Unit (FCU)
③ annul
Pipeline Registers
Memory
Register File
③ If "back-to-back fault injections"
Restart Trap Handler
FIRST Hardware/Software Extensions

Minimal Trap Handler



Nominal Mode

```
; AES Start
...
; Round 10, addRoundKey

LD      MEM[key_address], R1
LD      MEM[state_address], R2
XOR     R1, R2, R2
ST      MEM[state_address], R2
...
```

Safe Mode

```
; Trap handler
; Read FRRs to determine the valid FRR
; and get the trap return address
call    40001fcc <read_asr>

; Restore the pre-fault state of R2
call    40001f4c <write_asr>

; Return from trap
rett    <next instruction>
```

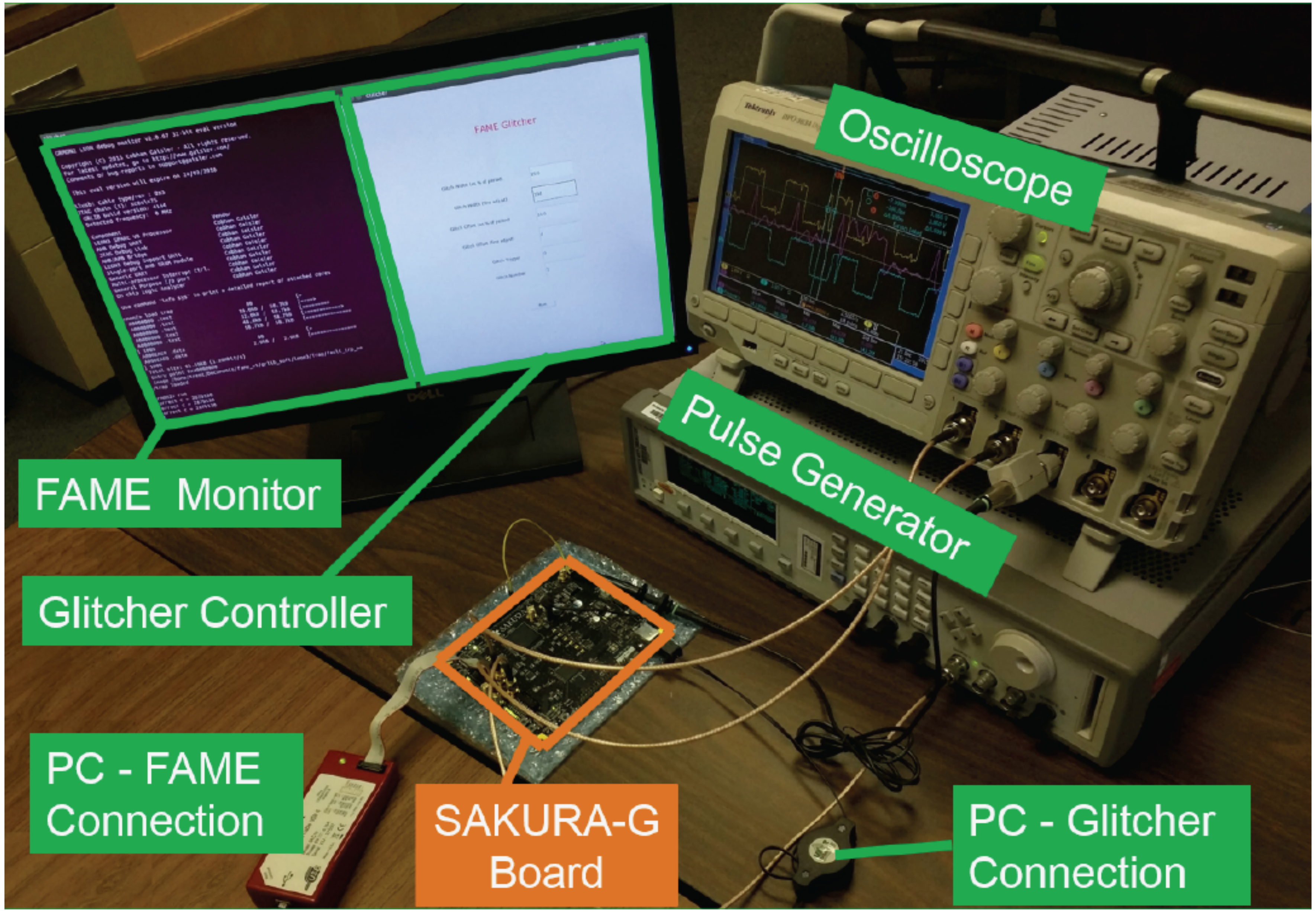
Prototype Design: FAME against Time Violation

- Protects against setup time violation attacks
  - Clock/voltage glitching
  - Voltage underfeeding

- Extends a Leon3 processor to FAME
  - 32-bit, 7-stage RISC Pipeline

- Implemented and tested on a Spartan6 FPGA of a SAKURA-G board

Fault Injection and Evaluation Setup



Case Study: The Cost of FAME

- Hardware Overhead (9% logic, 15% regs)

Component	# LUTs	# Registers
LEON3 (baseline)	3,435	1,275
FCU and FRR	256 (7.5%)	181 (14%)
FDU	53 (1.5%)	3 (1%)

- Software Overhead (application dependent)

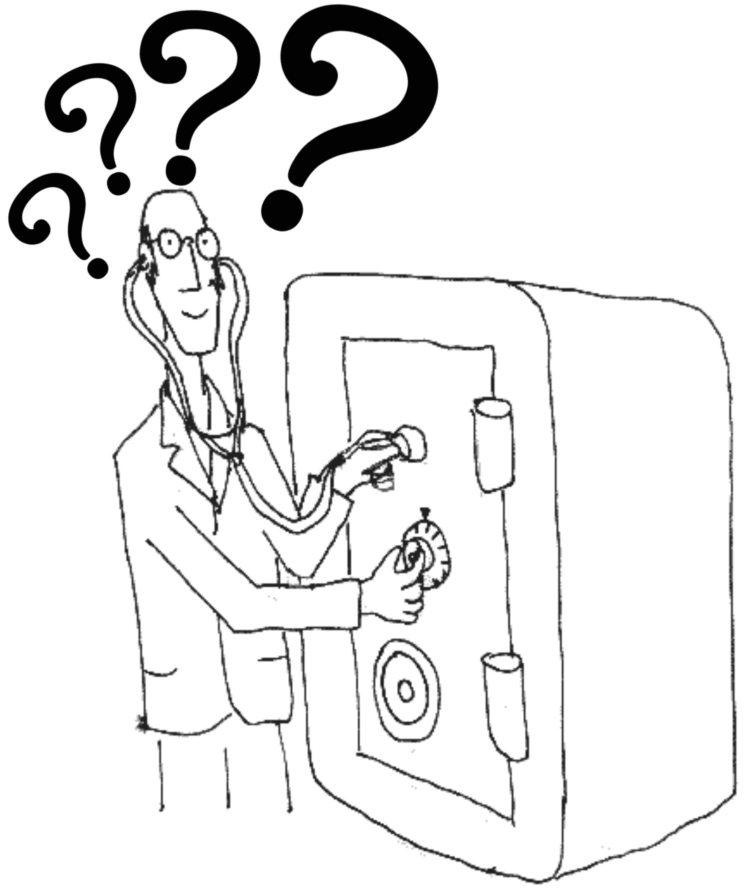
Application	# Cycles	Footprint (Byte)
AES (baseline)	17,631	25,964
AES + fault-Resume	17,810 (+1%)	26,116 (+0.6%)

Conclusions

- FAME provides a HW/SW solution to handle fault attacks:
  - Low-cost
  - Performance-efficient
  - Adaptive
  - Backward-compatible

- FAME is generic
  - Can support multiple fault detectors/sources
  - Can support multiple CPU architectures

Thank you!



How do Existing Methods Handle Faults?

- Full fault-tolerance
  - Information, temporal, or spatial redundancy
  - Either in Software or Hardware

- Detect-and-Despair
  - Mute/Lock the device
  - Initiate a hard-reset event
  - Kill/Destroy the device

Expensive

Multiple Fault Injection

Adaptive Adversary

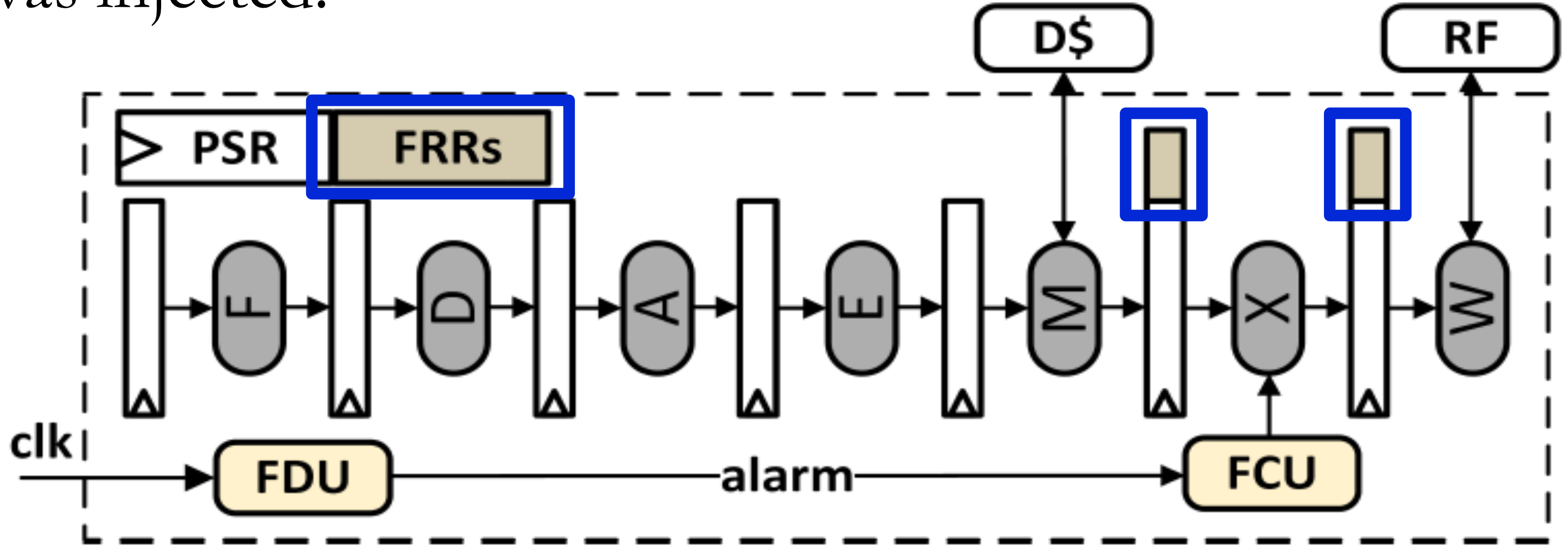
Disable Fault Response

Related Work

- Software countermeasures
  - Instruction Duplication, Application-specific redundancy, Concurrent Error Detection
  - Performance Hit and increased footprint
- Fault tolerant design
  - Redundant hardware design (similar overhead)
- Secure Processors
  - Memory integrity, confidentiality, attestation, isolation, …
  - Do not address faults

How do FRRs Maintain the Backup State?

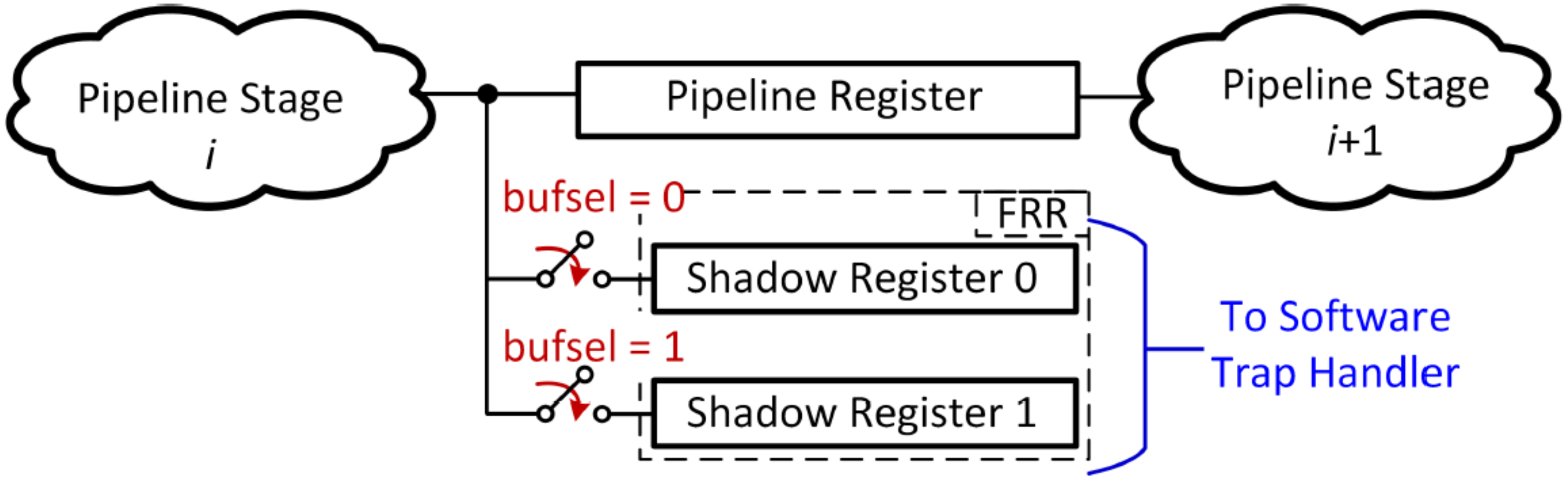
- FRR allow to rewind the *selected* processor state one clock cycle, just before the fault was injected.



- Minimum Content of FRR:
  - Return address to the interrupted program
  - Processor status register
  - Register file inputs of write- back stage

How do FRRs Maintain the Backup State?

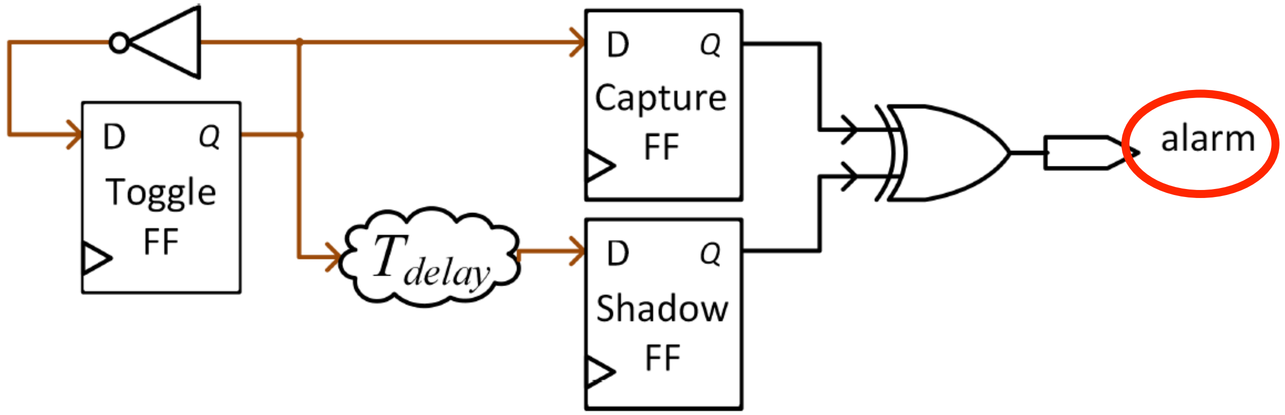
- FRR allow to backtrack *selected* processor state one clock cycle, before the fault was detected.



- Minimum Content of FRR:
  - Return address to the interrupted program
  - Processor's status register
  - Register file inputs of write- back stage

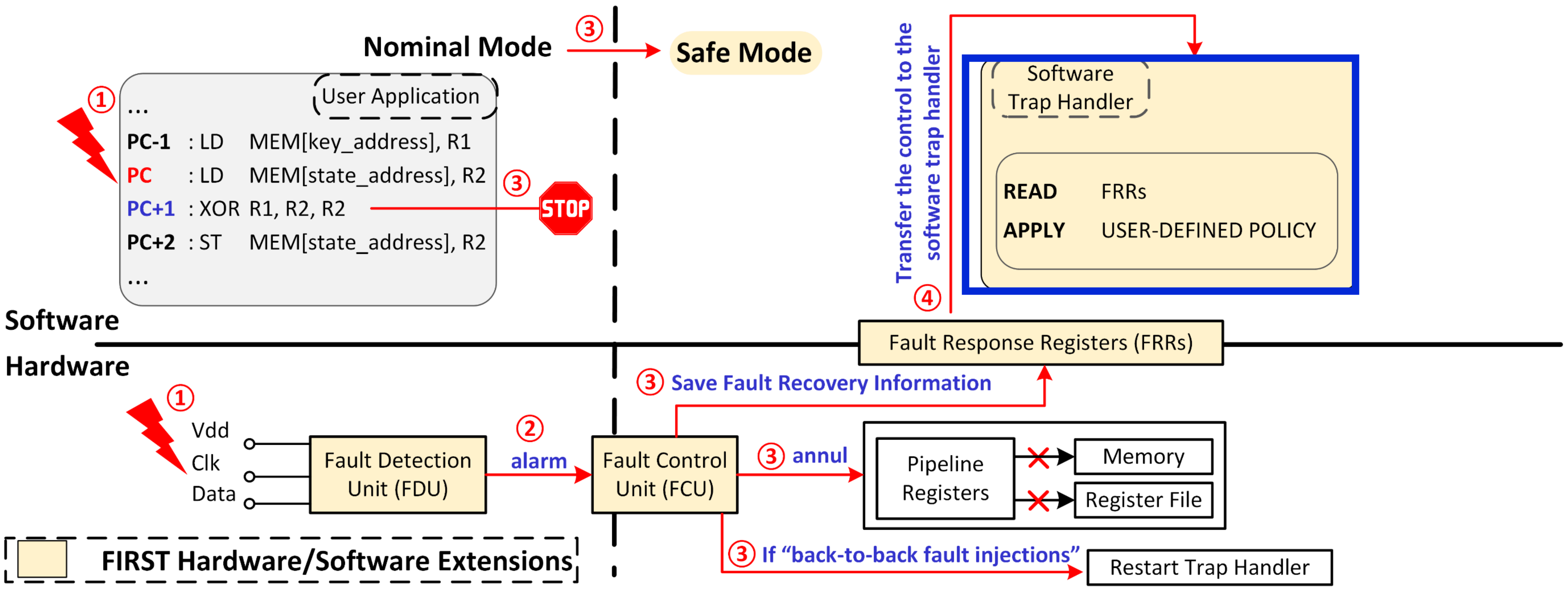
Fault Detection Unit

- Timing Violation Detector
  - caused by glitches
  - caused by voltage starving



- Not limited to glitches.
  - Optical, EM, overvoltage, .. detectors
  - Memory/Register checksum

Operation of FAME - 5



Nominal Mode
③
Safe Mode
①
User Application
...
PC-1 : LD MEM[key_address], R1
PC : LD MEM[state_address], R2
PC+1 : XOR R1, R2, R2
PC+2 : ST MEM[state_address], R2
...
③
STOP
Transfer the control to the software trap handler
④
Software Trap Handler
READ FRRs
APPLY USER-DEFINED POLICY
Software
Hardware
Fault Response Registers (FRRs)
③ Save Fault Recovery Information
①
Vdd
Clk
Data
Fault Detection Unit (FDU)
② alarm
Fault Control Unit (FCU)
③ annul
Pipeline Registers
Memory
Register File
③ If "back-to-back fault injections"
Restart Trap Handler
FIRST Hardware/Software Extensions

Why are Faults a Security Issue?

- May enable leakage of secret information by altering the data flow

```
// (Simplified) AES AddRoundKey
…
state = secretKey ^ state;
ciphertext = state;
return ciphertext;
```

Zeroize the state

Ciphertext will be equal to secretKey