Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier

LIRMM

UNIVERSITÉ MONTPELLIER 2 SCIENCES ET TECHNIQUES

cnrs

Fault Injection Tools Based on Virtual Machines

ReCoSoC'14

Maha Kooli, Pascal Benoit, Giorgio Di Natale, Lionel Torres, Volkmar Sieh

Clereco

Cross-Layer Early Reliability Evaluation for the Computing cOntinuum

Motivation

- Availability
- Reliability
- Fault Tolerance

Space Shuttle Columbia, February 1, 2003

Motivation

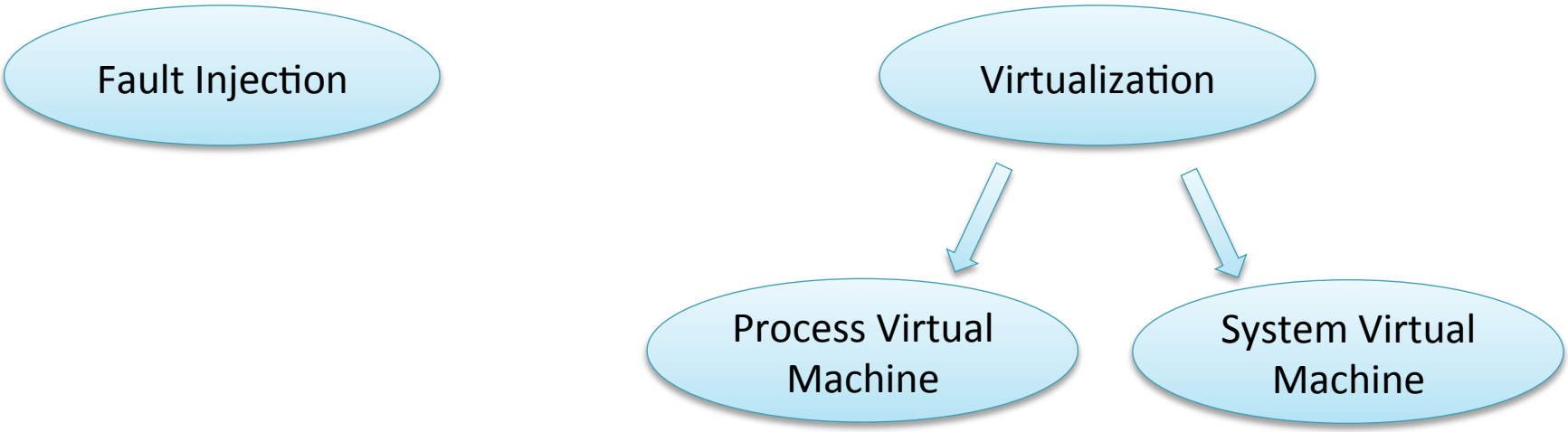
Fault Injection

Motivation

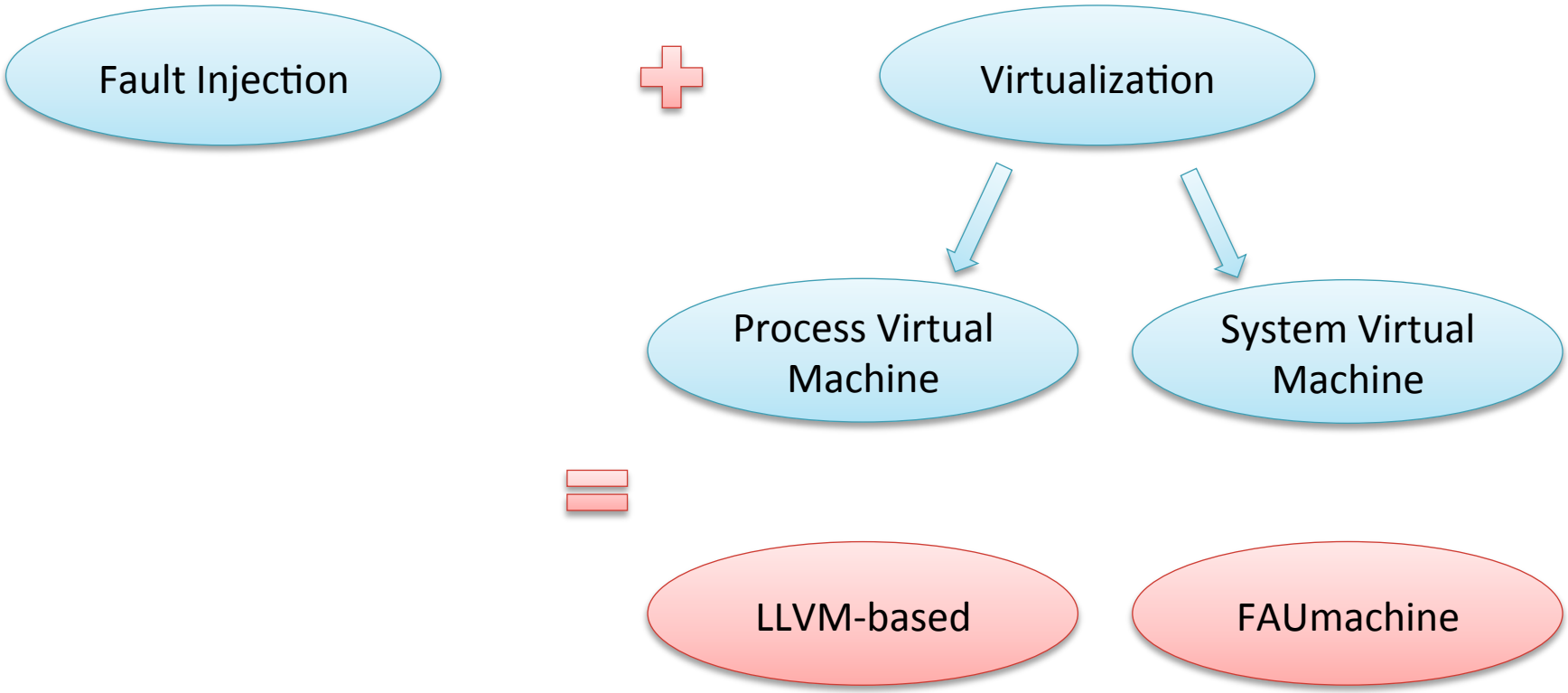
Fault Injection

Virtualization

Motivation


Fault Injection
Virtualization
Process Virtual Machine
System Virtual Machine

Motivation


Fault Injection
Virtualization
Process Virtual Machine
System Virtual Machine
LLVM-based
FAUmachine

Outline

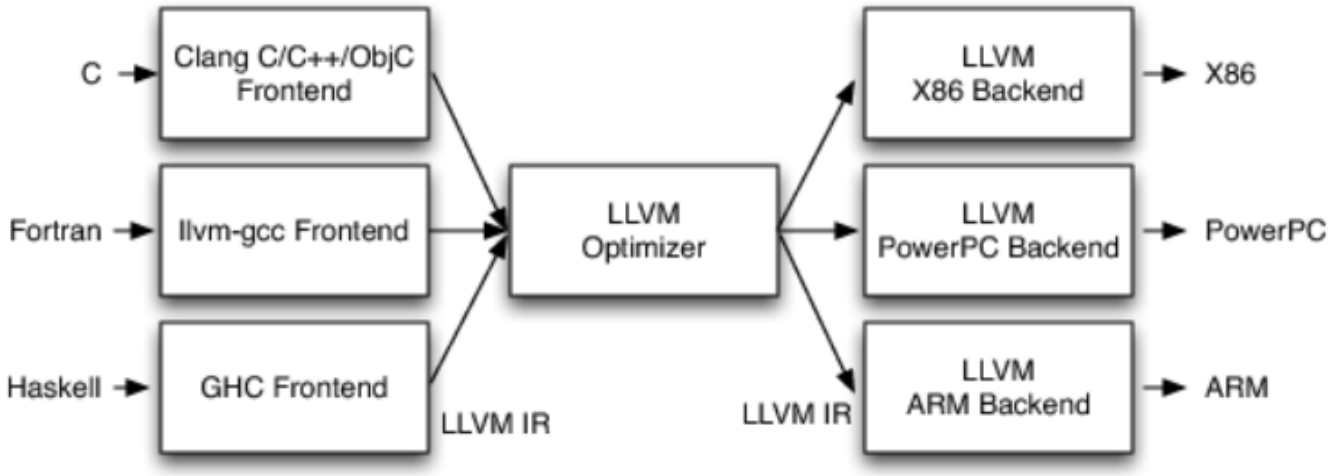
1. LLVM-based
    1. LLFI
    2. KULFI
    3. Comparison
2. FAUmachine
    2. Fault Model
    3. Implementation of FI in CPU Registers
    4. Discussion
3. Comparison and Conclusion

Outline

1. LLVM-based
   1. LLFI
   2. KULFI
   3. Comparison
2. FAUmachine
   2. Fault Model
   3. Implementation of FI in CPU Registers
   4. Discussion
3. Comparison and Conclusion

LLVM (Low Level Virtual Machine)

- A compiler infrastructure provide high-level information to compiler transformations.
- A framework for a virtual Instruction Set Architecture (ISA) definition.



L. Chris, "LLVM," in *The Architecture of Open Source Applications*, A.Brown and G. Wilson, Eds., vol. I, ch. 11.

LLVM (Low Level Virtual Machine)

- LLVM's Code Representation: Intermediate Representation (IR)
  - The form it uses to represent code in the compiler.
  - Independent from the source language and the target machine

⇨ Define an abstraction layer to make the information at software level and the information at hardware level, compatible and easily exchangeable.

LLFI

- An LLVM based fault injection tool (IR code level)
- Identify source code level characteristics of EDC causing faults

- **Fault Model:**
  - Considered Faults:
    - Functional units: the ALU and the address computation for loads and stores
  - Not considered Faults:
    - Memory components (caches) => Protected at the architectural level using ECC or parity.
    - Control logic of the processor => This is a small portion of the processor area
    - Instructions => Handled through control-flow checking techniques

KULFI

- LLVM -level fault injector tool (instruction level)
- Helps simulate faults occurring within CPU state elements

- **Fault Model:**
  - **Static faults:** permanent faults, injected during compile time
  - **Dynamic faults:** transient faults, injected during program execution

FAULT INJECTION

1. Configure the compilation and fault injection parameters and options

```
compileOption:
    …
runOption:
    - run:
        numOfRuns: 2
        fi_type: bitflip
    - run:
        numOfRuns: 3
        fi_type: stuck_at_0
    - run:
        numOfRuns: 1
        fi_type: stuck_at_1
```

input.yaml

FAULT INJECTION

2. Build a single IR file with the LLFI tool compiletoIR

```
int sum(int N){
  int i;
  int s = 0;
  for(i = 1; i <= N; ++i){
     s += i;
  }
  return s;
}
```

compileIR

```
; <label>:6
  %7 = load i32* %i
  %8 = load i32* %s
  %9 = add nsw i32 %8, %7
  store i32 %9, i32* %s
  br label %10
```

```
; <label>:6
  %7 = load i32* %i,
  %fi2 = call i32 @injectFault1(i64 11, i32
  %7, i32 27, i32 0, i32 1)
  %8 = load i32* %s
  %fi3 = call i32 @injectFault1(i64 12, i32
  %8, i32 27, i32 0, i32 1)
  %9 = add nsw i32 %fi3, %fi2
  %fi4 = call i32 @injectFault1(i64 13, i32
  %9, i32 8, i32 0, i32 1)
  store i32 %fi4, i32* %s
  br label %10
```

3. Inject fault in the IR code, and run the execution

Fault Injection Results

- ***std_output***: Output results from the tested application
  - Wrong results
- ***error_output****:* Failure reports (program crashes, hangs, etc.)
  - 3 crashes / 6 FI runs
- ***llfi_stat_output****:* Fault injection statistics

```
FI stat: fi_type=bitflip, fi_index=23, fi_cycle=59,
fi_reg_index=0, fi_bit=33
```

COMPARISON

	LLFI	KULFI
Principal Function	Identify source code level heuristics of EDC causing faults	Simulate faults occurring within CPU state elements
Fault Model	• Transient hardware faults that occur in the processor • Permanent hardware faults (stuck-at)	• Static faults: permanent faults, injected during compile time • Dynamic faults: transient faults, injected during program execution
Fault Injection	• Inject fault in the intermediate code level of the application (IR) • Inject a single bit fault into the destination register	• Inject fault in the intermediate code level (IR) (LLVM bitcode level) • Inject a single bit faults into both data and address registers
Validation	• Assembly code level fault injection (PINFI)	• Compare the error resilience of algorithms for both SDC and crash causing errors
Feedbacks	• Uses more recent version of LLVM	• Provides more precise control over the fault injection process • Easier to control

Outline

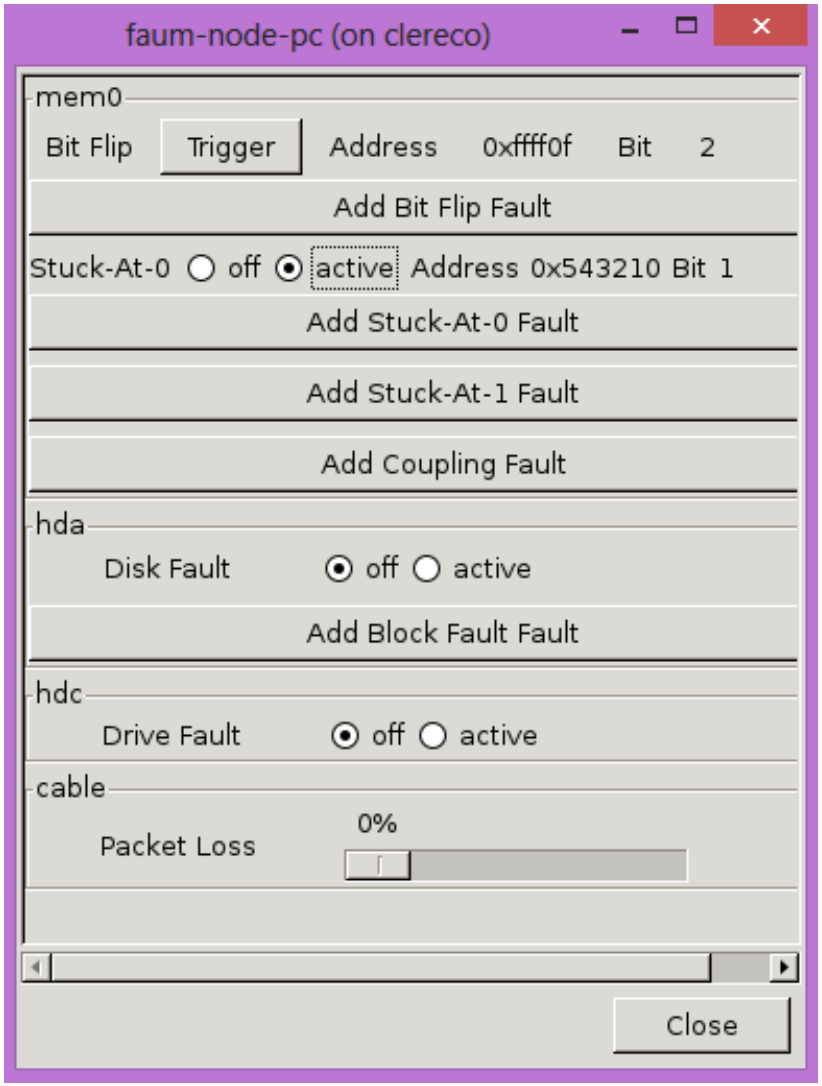
1. LLVM-based
   1. LLFI
   2. KULFI
   3. Comparison
2. **FAUmachine**
   2. **Fault Model**
   3. **Implementation of FI in CPU Registers**
   4. **Discussion**
3. Comparison and Conclusion

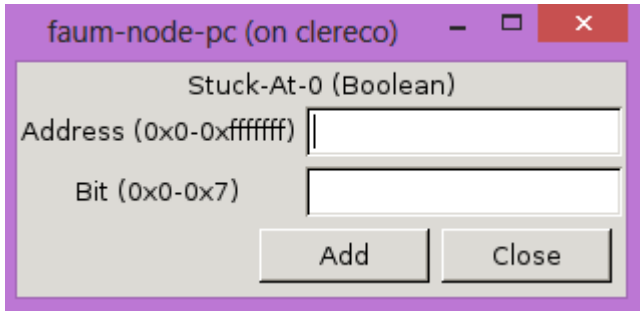
FAUMACHINE

- Virtual machine similar to QEMU or Virtual Box
- It supports Just-In-Time (JIT) compiling
- It supports Linux as OS, and i386 and x86_64 as hardware

- **Memory:** Transient bit flips and Permanent stuck-at and coupling faults
- **Disk/CDROM:** Transient and permanent block and whole disk faults
- **Network:** Transient, Intermittent and Permanent send and receive faults

Fault Injection via GUI





Fault Injection via VHDL Script

Fault Injection via VHDL Script

```
architecture behaviour of fi is
        signal err : boolean;
begin
        process
        begin
                shortcut_bool_out(
                   err,
                   ":pc:mem0",
                   "stuck_at_0/0x543210/0");
                err <= true;
        end process;
end behaviour;
```

Define the signal for the fault

Fault Injection via VHDL Script

```
architecture behaviour of fi is
        signal err : boolean;
begin
        process
        begin
                shortcut_bool_out(
                    err,
                    ":pc:mem0",
                    "stuck_at_0/0x543210/0");
                err <= true;
        end process;
end behaviour;
```

Connect the actual fault

Fault Injection via VHDL Script

```
architecture behaviour of fi is
        signal err : boolean;
begin
        process
        begin
                shortcut_bool_out(
                    err,
                    ":pc:mem0",
                    "stuck_at_0/0x543210/0");
                err <= true;
        end process;
end behaviour;
```

The signal of the fault

Fault Injection via VHDL Script

```
architecture behaviour of fi is
        signal err : boolean;
begin
        process
        begin
                shortcut_bool_out(
                   err,
                   ":pc:mem0",
                   "stuck_at_0/0x543210/0");
                err <= true;
        end process;
end behaviour;
```

The path to the instantiated component

Fault Injection via VHDL Script

```
architecture behaviour of fi is
        signal err : boolean;
begin
        process
        begin
                shortcut_bool_out(
                   err,
                   ":pc:mem0",
                   "stuck_at_0/0x543210/0");
                err <= true;
        end process;
end behaviour;
```

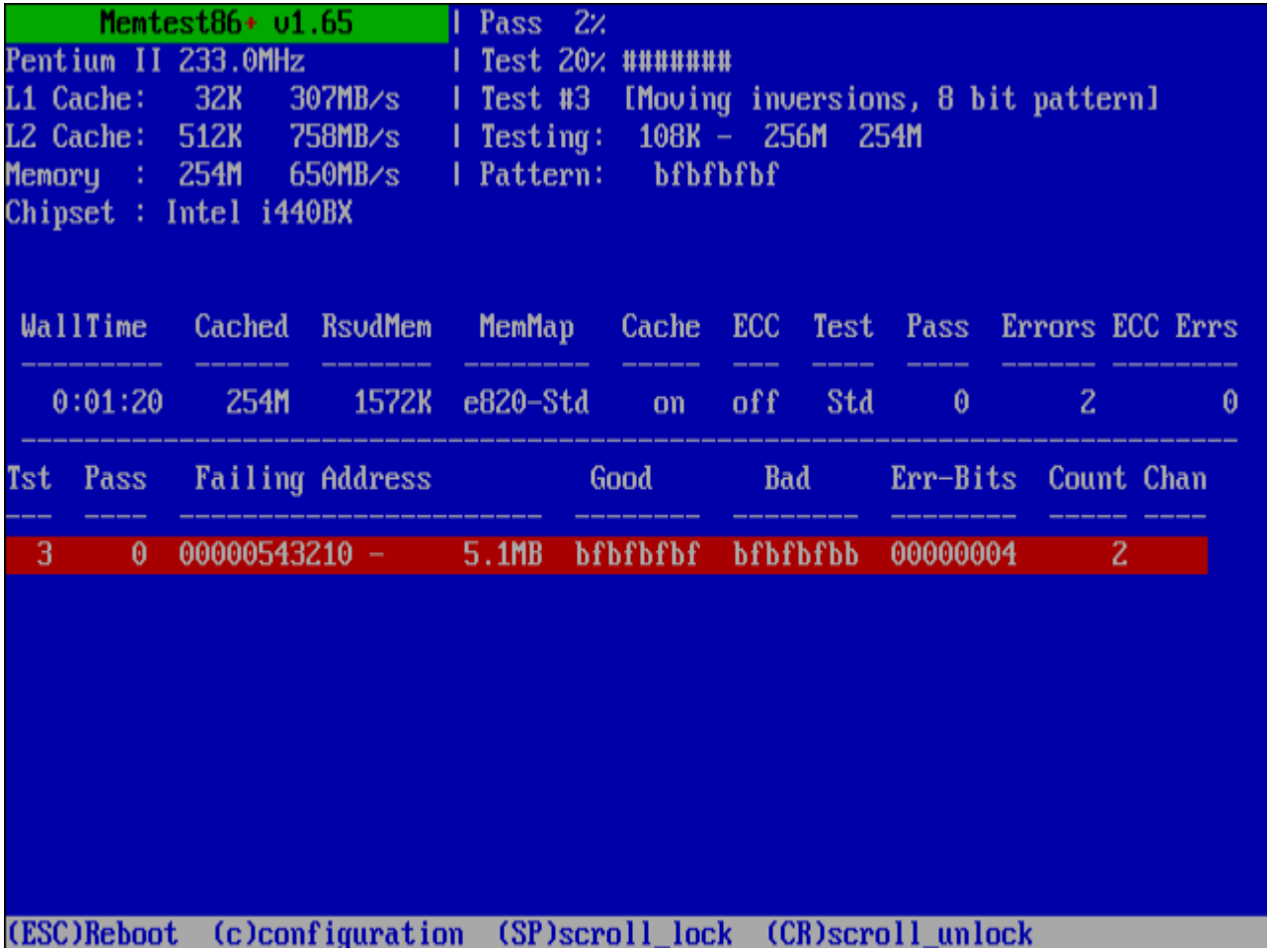
← Fault Parameters

Fault Injection via VHDL Script

```
architecture behaviour of fi is
        signal err : boolean;
begin
        process
        begin
                shortcut_bool_out(
                   err,
                   ":pc:mem0",
                   "stuck_at_0/0x543210/0");
                err <= true;
        end process;
end behaviour;
```

Activate the fault

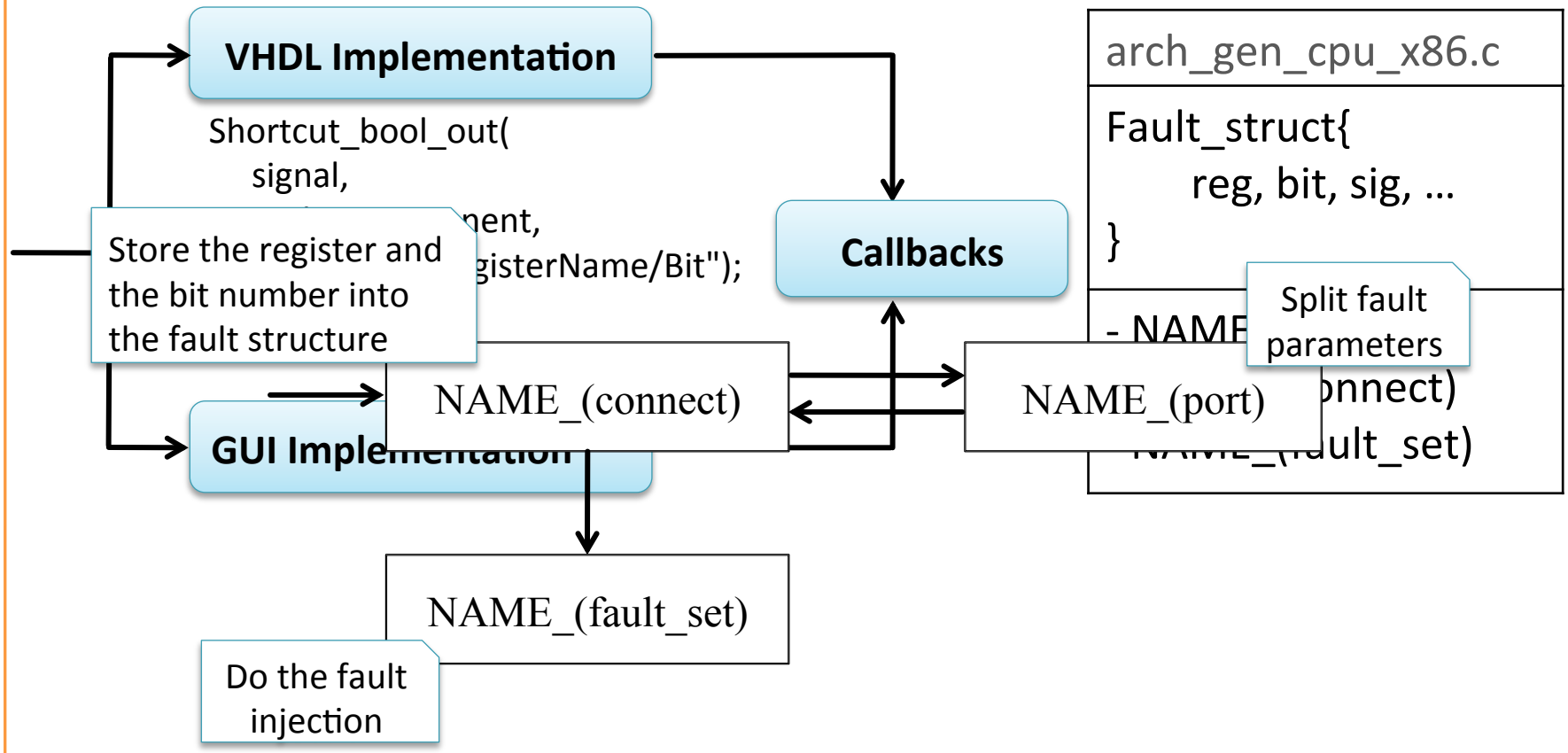
Fault Injection Result



CPU Registers FI

- FAUmachine did not support injecting fault in the CPU registers
- Implementing the bit flip and the stuck-at faults requires the modification of the JIT compiler of FAUmachine.
- The C-code of the simulated hardware of FAUmachine tries to reflect the faulty behavior of the real hardware => Make possible to add new fault injection capabilities

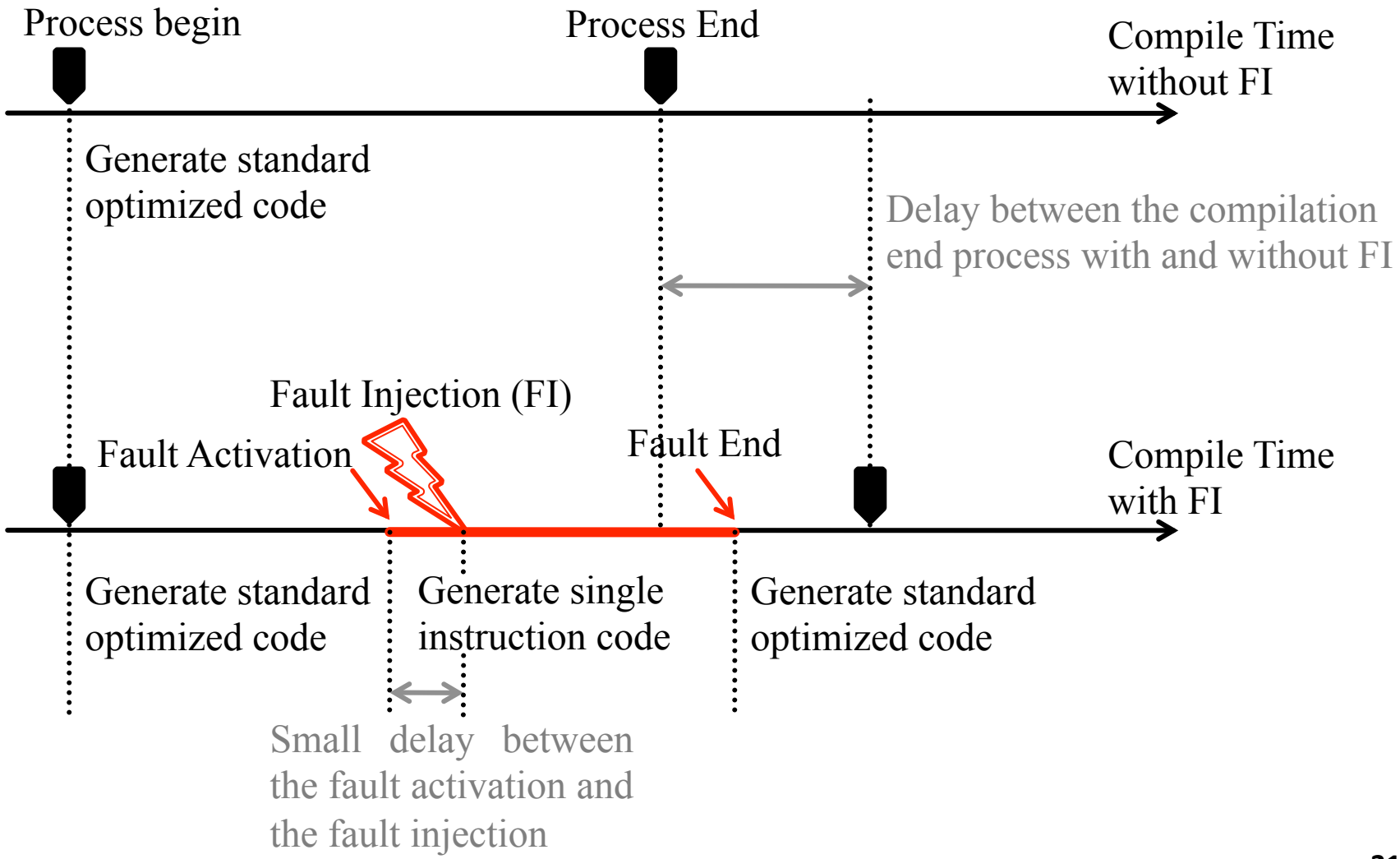
CPU Registers FI - Conception



CPU Registers FI - Discussion

- The infrastructure needed when the fault injection function is called
- Many other internal mechanisms need to be handled
- The challenge: the modification of **the JIT compiler**

CPU Registers FI - Bitflip



CPU Registers FI - Stuck-at

```
add $1, %eax
```

Compiling instruction without fault injection

```
T0 = env->reg_eax;
T1 = 1;
do_add();
env->reg_eax = T2;
```

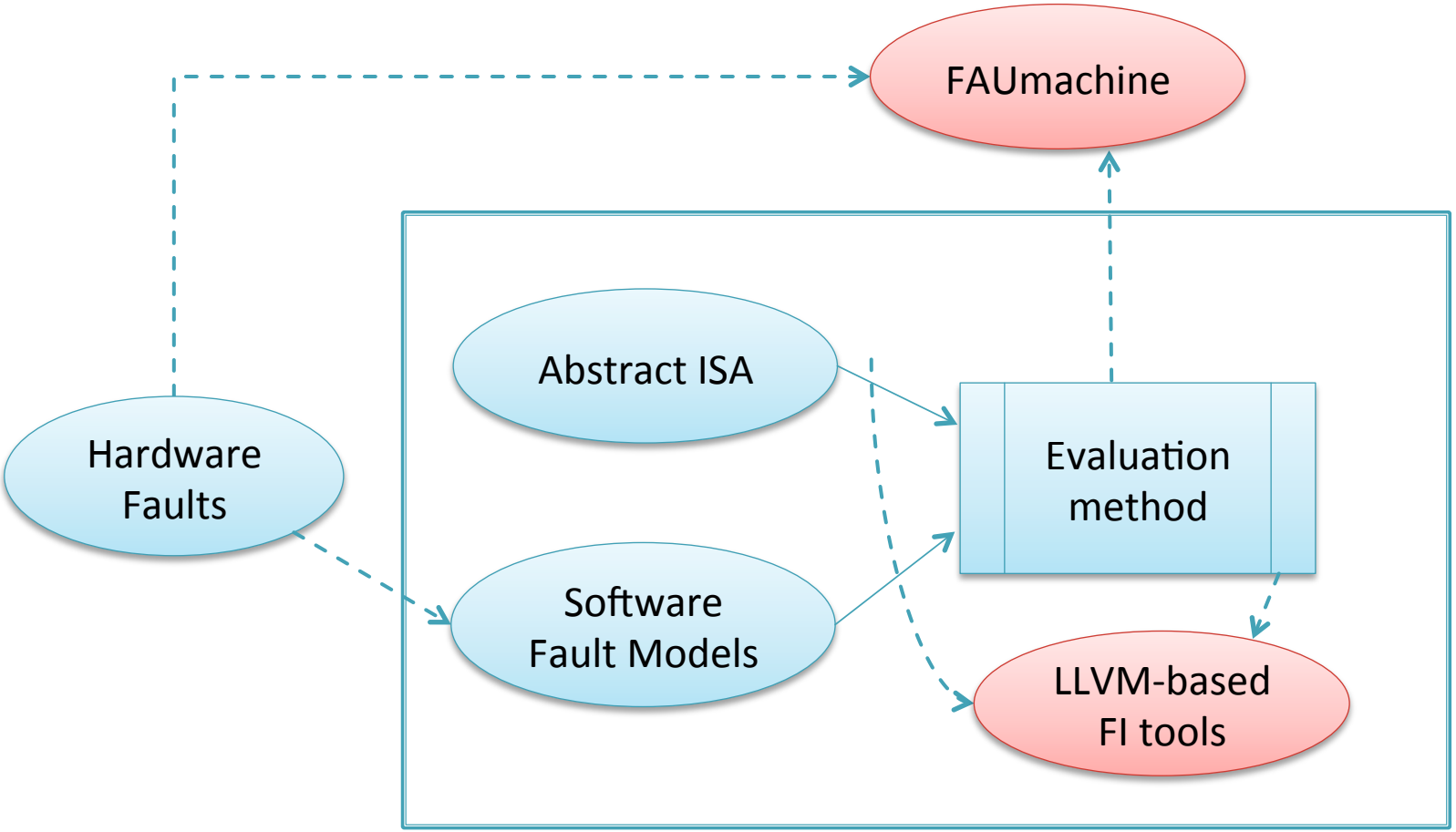
Compiling instruction with injection of stuck-at 1 into register %eax in bit 2

```
T0 = env->reg_eax;
T1 = 1;
do_add();
env->reg_eax = T2;
env->reg_eax |= 1 << 2; /* Fault-Injection */
```

Outline

1. LLVM-based
   1. LLFI
   2. KULFI
   3. Comparison
2. FAUmachine
   2. Fault Model
   3. Implementation of FI in CPU Registers
   4. Discussion
3. **Comparison and Conclusion**

FAUmachine vs LLVM-based



Perspective

- Adapt the LLVM-based Fault Injection tools to be able to handle the defined fault models
- Define evaluation methods of the reliability without using fault injection
- Compare the results of the two methods in a high level
- Validate the reliability evaluation in the software level with the result of the evaluation of the global system

Thanks for your attention

Questions?