Leibniz Universität Hannover

TUHH
Technische Universität Hamburg

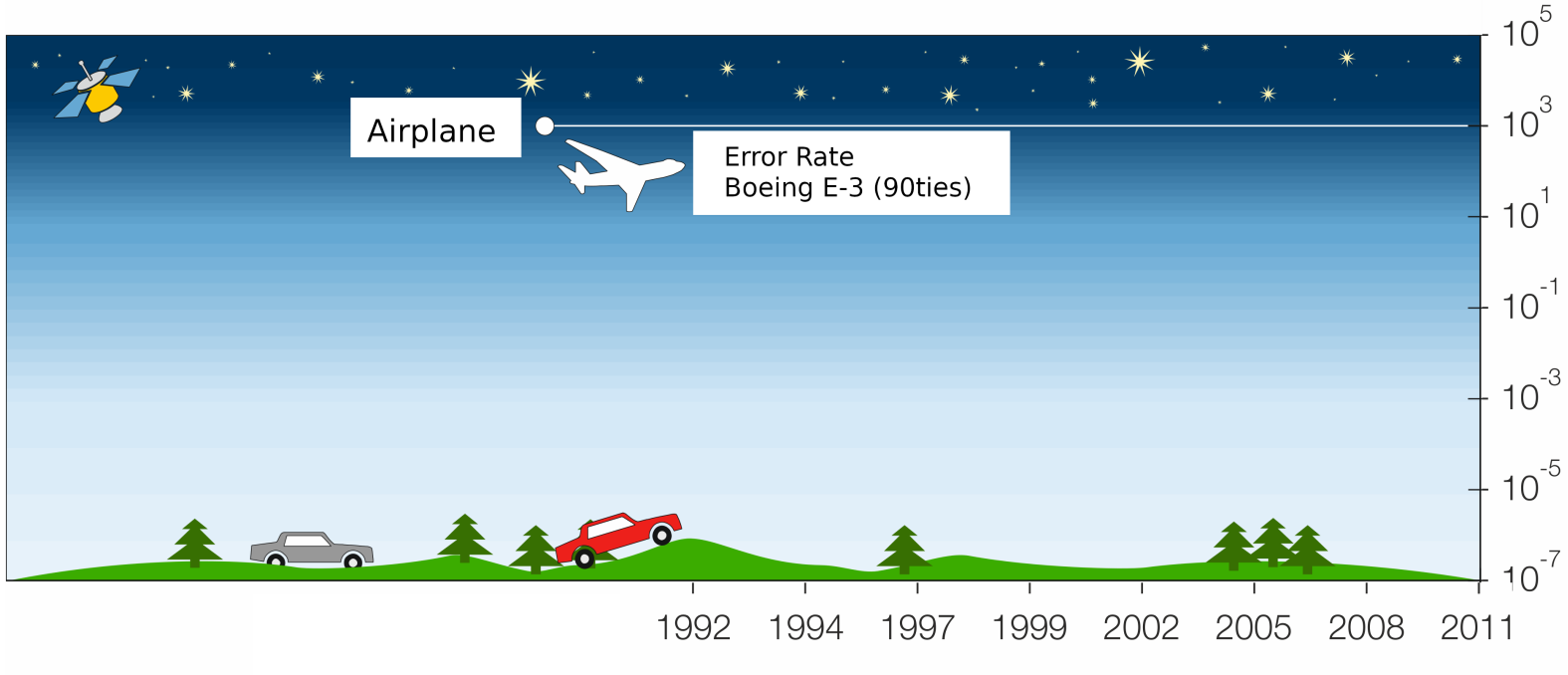
SailFAIL: Model-Derived Simulation-Assisted ISA-Level Fault-Injection Plattforms

SAFECOMP 2022

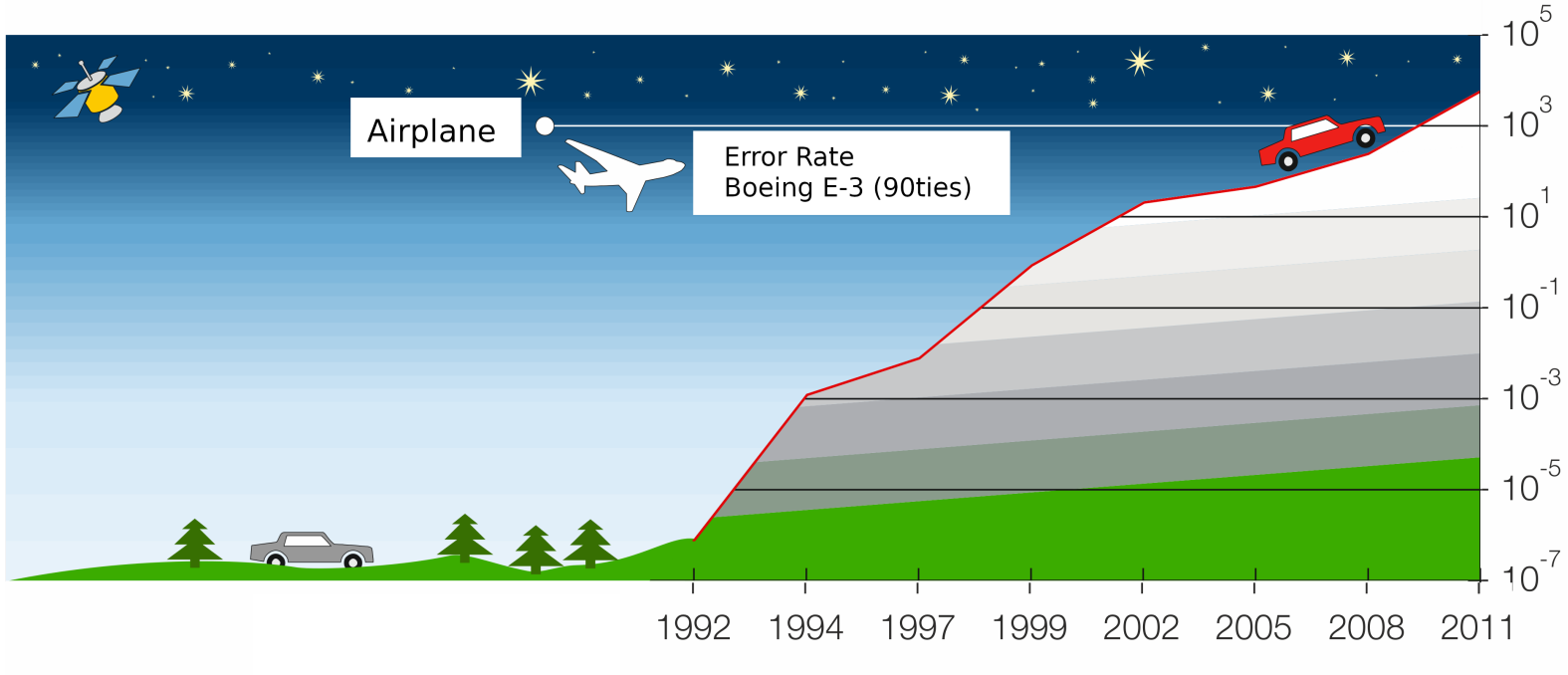
Christian Dietrich, Malte Bargholz, Yannick Loeck, Marcel Budoj, Luca Nedaskowskij, Daniel Lohmann

September 7, 2022

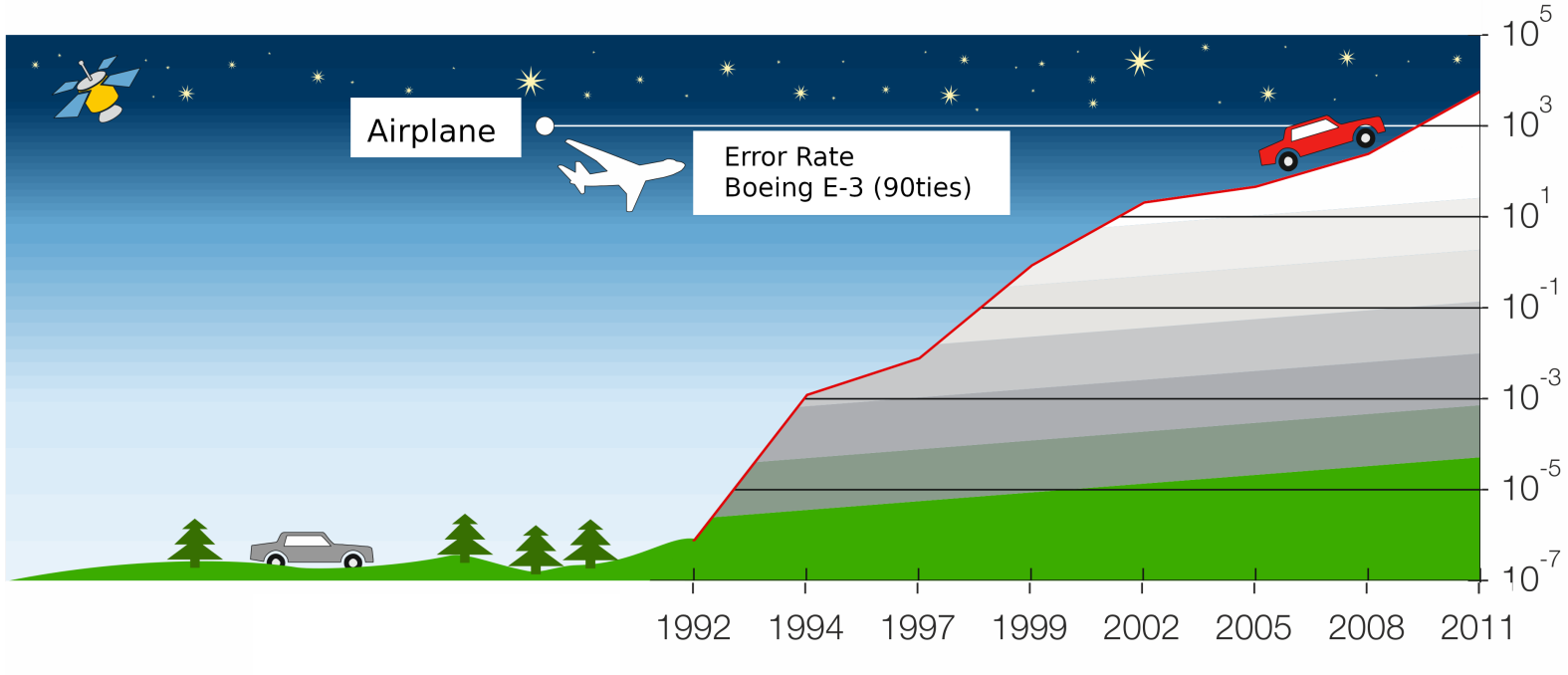
Soft Errors are a Problem



Soft Errors are a Problem



Soft Errors are a Problem



"Toyota claimed the 2005 Camry's main CPU had error detecting and correcting RAM. ***It didn't.****"*

Source: Investigation Report, EDN Network, 28. Oktober 2013

Resilience Assessment by Fault-Injection

TUHH
Technische Universität Hamburg

- How susceptible is my software to soft errors (bit-flips)?
  - Radiation Experiments: realistic but expensive/slow.
  - HAFI/FPGA: systematic, but requires specialized FPGA pool
  - Simulation-Assisted FI: systematic, scales out, efficient for ISA-level

Resilience Assessment by Fault-Injection

TUHH
Technische Universität Hamburg

- How susceptible is my software to soft errors (bit-flips)?
  - Radiation Experiments: realistic but expensive/slow.
  - HAFI/FPGA: systematic, but requires specialized FPGA pool
  - Simulation-Assisted FI: systematic, scales out, efficient for ISA-level

- **SAFI**: Required Tooling and Challenges
  - Fault Planning
  - Simulator Platform
  - Campaign Manager
  - Result Analysis
  - Accelerating HW Development (RISC-V)
  - Specialized ISA extensions (for resilience)
  - Same Behavior? Simulator $\leftrightarrow$ Real Hardware

Resilience Assessment by Fault-Injection

TUHH
Technische Universität Hamburg

- How susceptible is my software to soft errors (bit-flips)?
  - Radiation Experiments: realistic but expensive/slow.
  - HAFI/FPGA: systematic, but requires specialized FPGA pool
  - Simulation-Assisted FI: systematic, scales out, efficient for ISA-level

- **SAFI**: Required Tooling and Challenges
  - Fault Planning
  - Simulator Platform
  - Campaign Manager
  - Result Analysis
  - Accelerating HW Development (RISC-V)
  - Specialized ISA extensions (for resilience)
  - Same Behavior? Simulator $\leftrightarrow$ Real Hardware

Our Approach

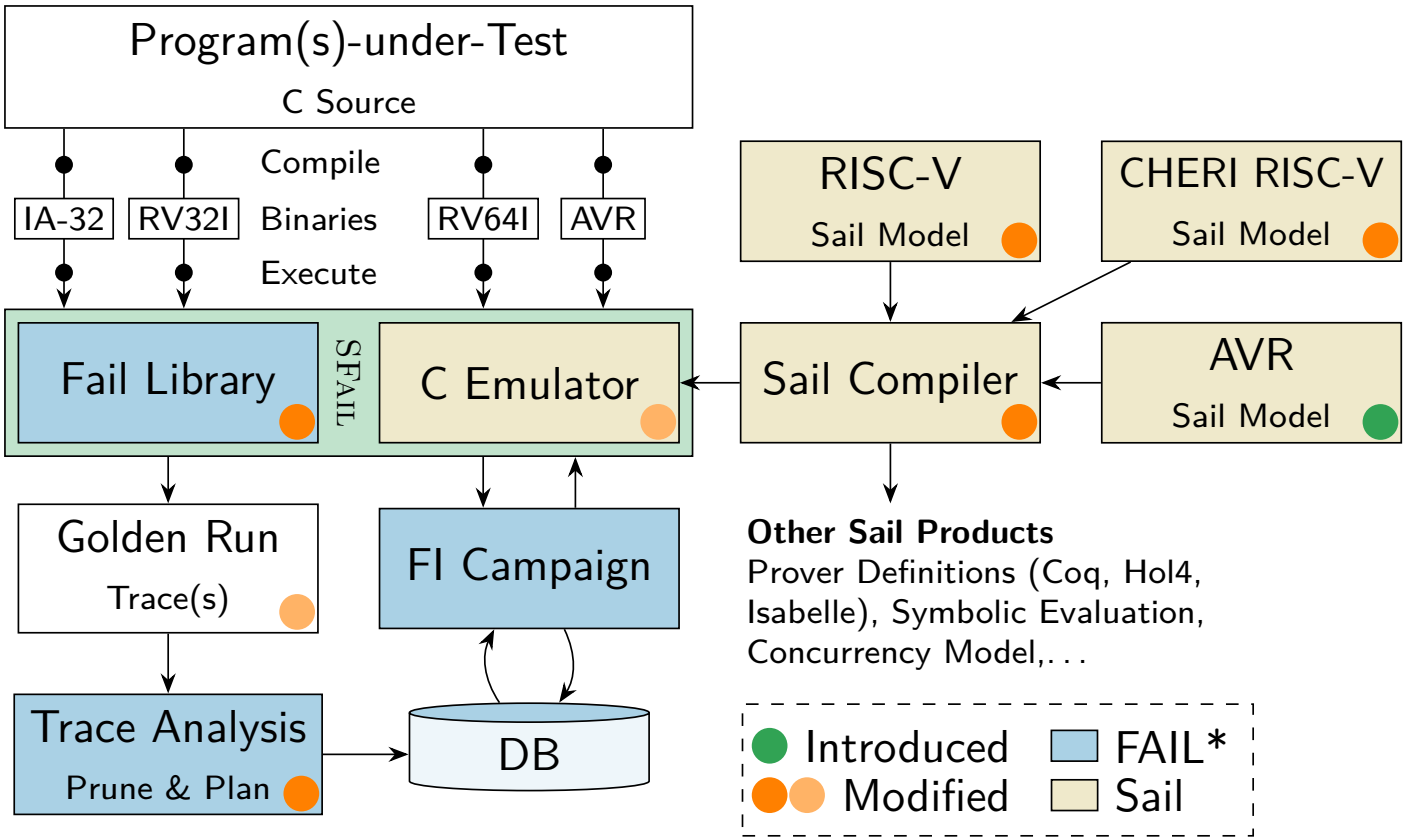
- Derive SAFI simulator from formal ISA-level CPU
- Combine with existing FI toolchain (FAIL*)

Overview

TUHH
Technische Universität Hamburg

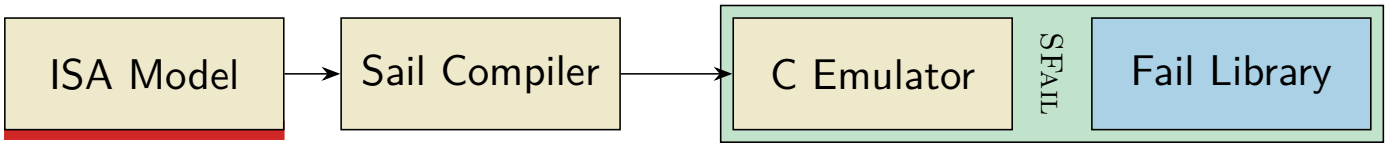
- Motivation
- **SailFAIL: Model-Derived Fault Injectors**
- Case-Study: CHERI RISC-V
- Conclusion

SailFAIL: Overview



- Combine existing tools:
  - FAIL*: SAFI toolchain that supports multiple backends (Bochs, Gem5) golden-run recording, fault planning, campaign management
  - Sail: Language to describe ISA-level semantics; many models (RISC-V) Ships with a "model→C" emulator compiler

Sail Models



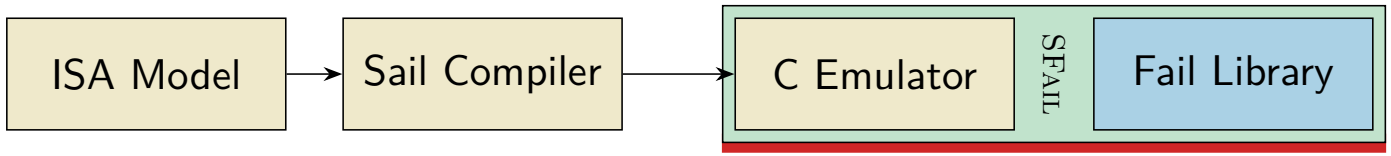

```
register PC  : bits(22)
register nPC : bits(22)
register SP  : bits(16)

function clause decode  0b1101 @ (offset : bits(12))
    = AVR_RCALL(offset)

function clause execute AVR_RCALL(offset) = {
    write_dmem(SP, nPC);
    SP  = SP - 2;
    nPC = nPC + (offset * 2)
}
```

- Sail: Modeling Language for ISA semantics
  - Pattern matching, dependent typing, scattered definitions
  - Definitions for model checkers, symbolic executions, and a **C emulator**

SAFI Platform

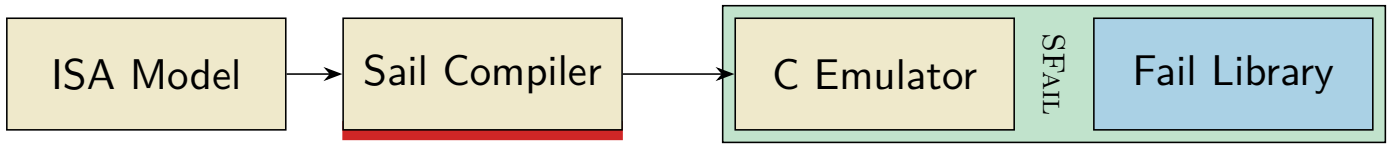


- SAFI platforms require emulator hooks
  - Observation: register/memory accesses, breakpoints, traps, interrupts
  - Control: start, stop, save/restore, forward execution
  - Injection: Modify the volatile state (memory and registers)

- Our Approach: Hooks and Automated Modifications

(A) Insert high-level semantic callbacks into the model
(B) Model-independent state save/restore and access mechanisms
(C) Generated emulator traces register accesses

SAFI Platform



- SAFI platforms require emulator hooks
  - Observation: register/memory accesses, breakpoints, traps, interrupts
  - Control: start, stop, save/restore, forward execution
  - Injection: Modify the volatile state (memory and registers)

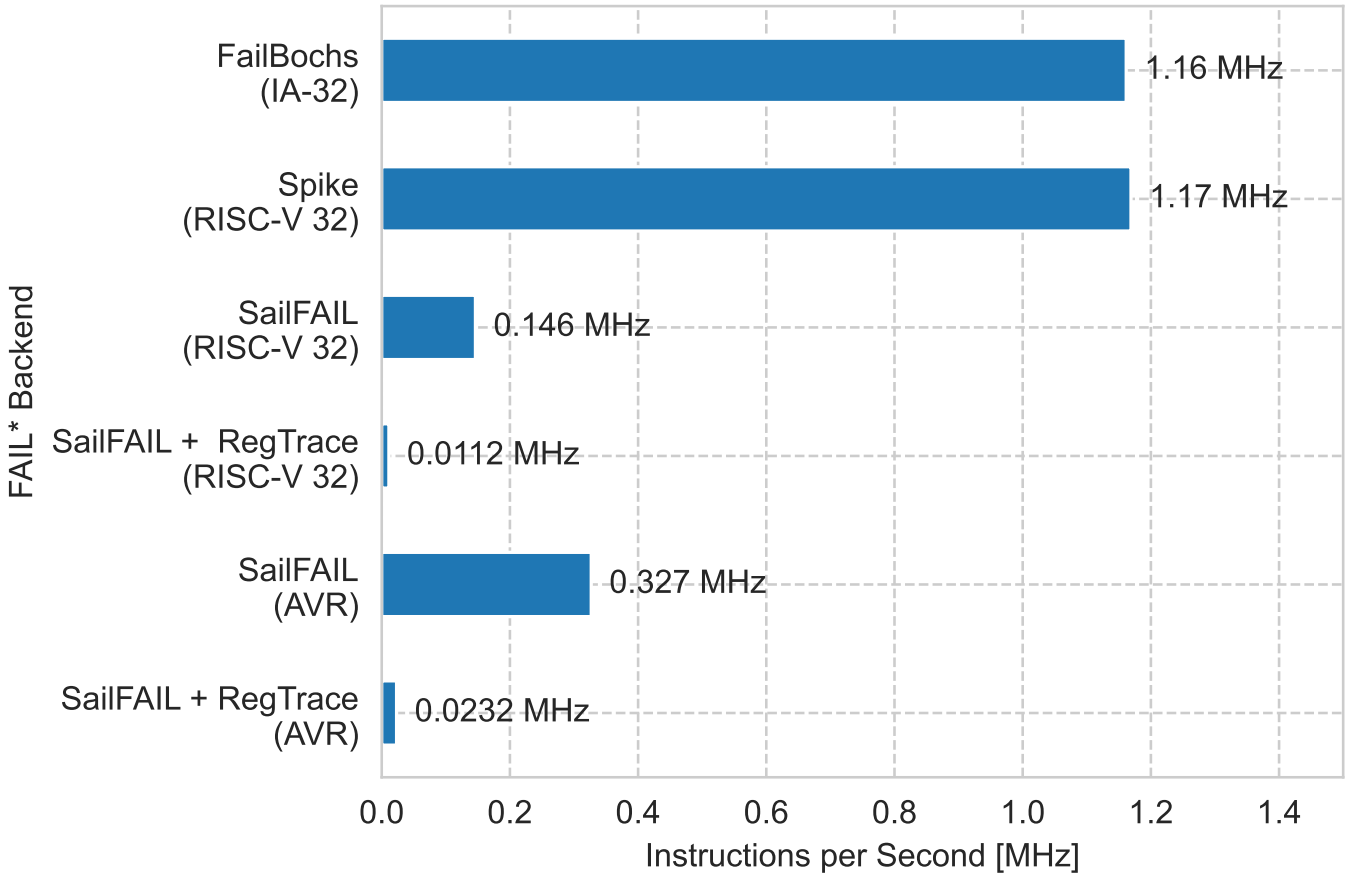
- Our Approach: Hooks and Automated Modifications

(A) Insert high-level semantic callbacks into the model
(B) Model-independent state save/restore and access mechanisms
(C) Generated emulator traces register accesses

- **Bit-precise register access**
  - Modified compiler inserts tracing for register accesses
  - Allows for precise bit-field access tracking

$\Rightarrow$ Fibonacci(500, AVR): $9.42 \cdot 10^5$ inj. $\Rightarrow$ $7.9 \cdot 10^5$ inj. (-16.17 %)

SailFAIL: Simulation Performance



- Intel Xeon Gold 6262 CPU with 2.10 GHz
- Sail emulators are slower than hand-crafted emulators
- Checkpoint save/restore: SailFail RISC-V (24 ms) vs. Bochs: 540 ms
- Register tracing is rather slow, but is required only once

Overview

TUHH
Technische Universität Hamburg

- Motivation
- SailFAIL: Model-Derived Fault Injectors
- **Case-Study: CHERI RISC-V**
- Conclusion

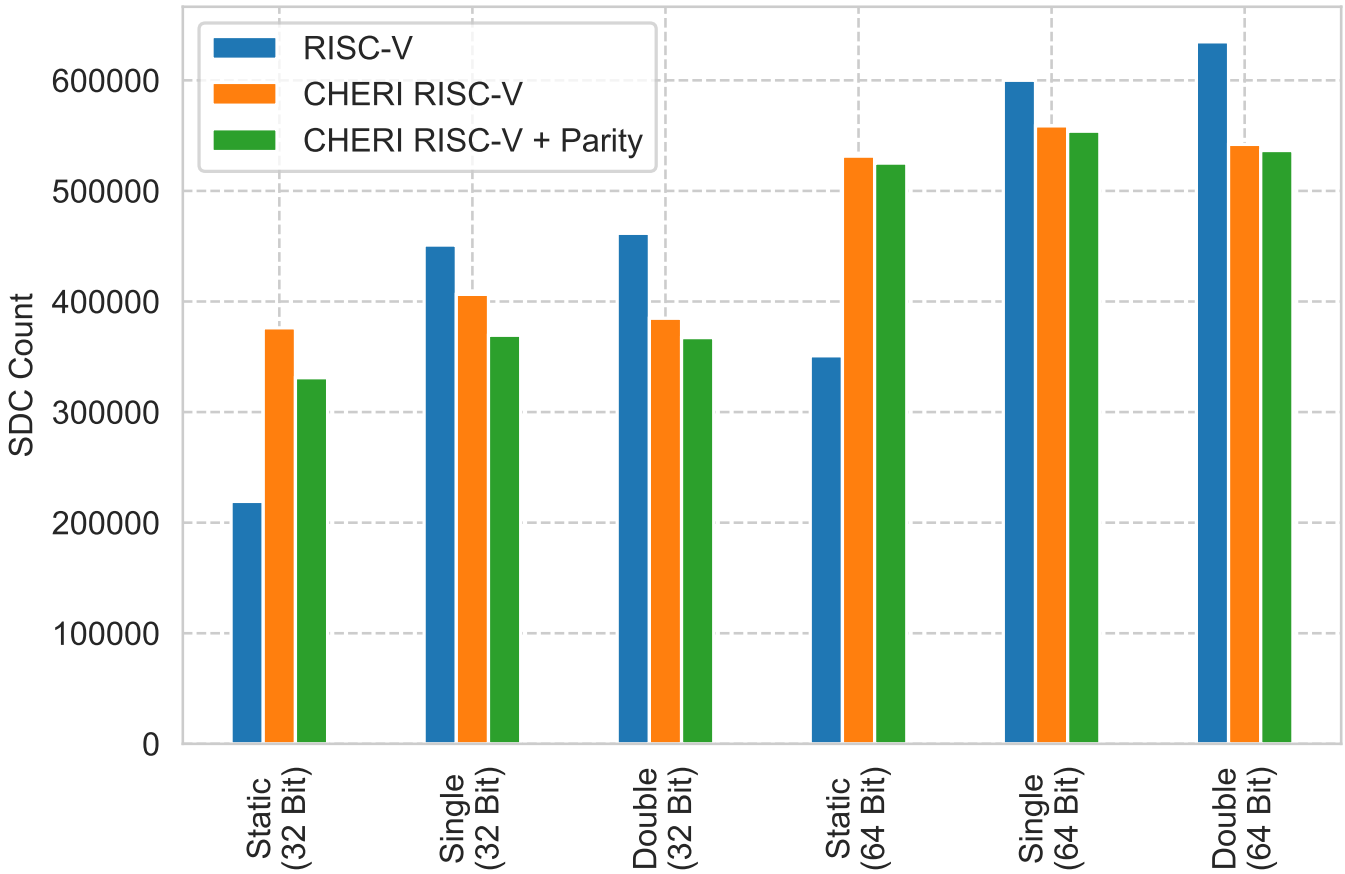
Case Study: Bubblesort on (CHERI) RISC-V

- CHERI: Hardware-Assisted User-Space Capability
  - In a nutshell: User-controlled, HW-enforced unforgeable fat pointers
  - Pointers are wider but accesses are more restricted
  - Question: Are CHERI programs more or less susceptible to soft errors?
- With SailFAIL: Derive six FI platforms from two Sail models
  - 32-bit/64-bit RISC-V with and without CHERI extension
  - Additional Variant: Parity-Checked CHERI capabilities
- Three Bubblesort Variants
  - Static array, single-linked list, double-linked List.
  - Same Algorithm, same data, different capability granularity.
- Uniform memory FI, full FS coverage, weighted absolute SDCs

Case Study: Bubblesort on (CHERI) RISC-V

- CHERI: Hardware-Assisted User-Space Capability
  - In a nutshell: User-controlled, HW-enforced unforgeable fat pointers
  - Pointers are wider but accesses are more restricted
  - Question: Are CHERI programs more or less susceptible to soft errors?
- With SailFAIL: Derive six FI platforms from two Sail models
  - 32-bit/64-bit RISC-V with and without CHERI extension
  - Additional Variant: Parity-Checked CHERI capabilities
- Three Bubblesort Variants
  - Static array, single-linked list, double-linked List.
  - Same Algorithm, same data, different capability granularity.
- Uniform memory FI, full FS coverage, weighted absolute SDCs

Case-Study: Results



- CHERI is less robust with coarse-grained capabilities
- 32 $\rightarrow$ 64 bit: Robustness is not halved
- Parity-Checking: Improves SDC rate by up to 12 percent.
- Double-Linked: RISC-V suffers, CHERI RISC-V benefits

Overview

TUHH
Technische Universität Hamburg

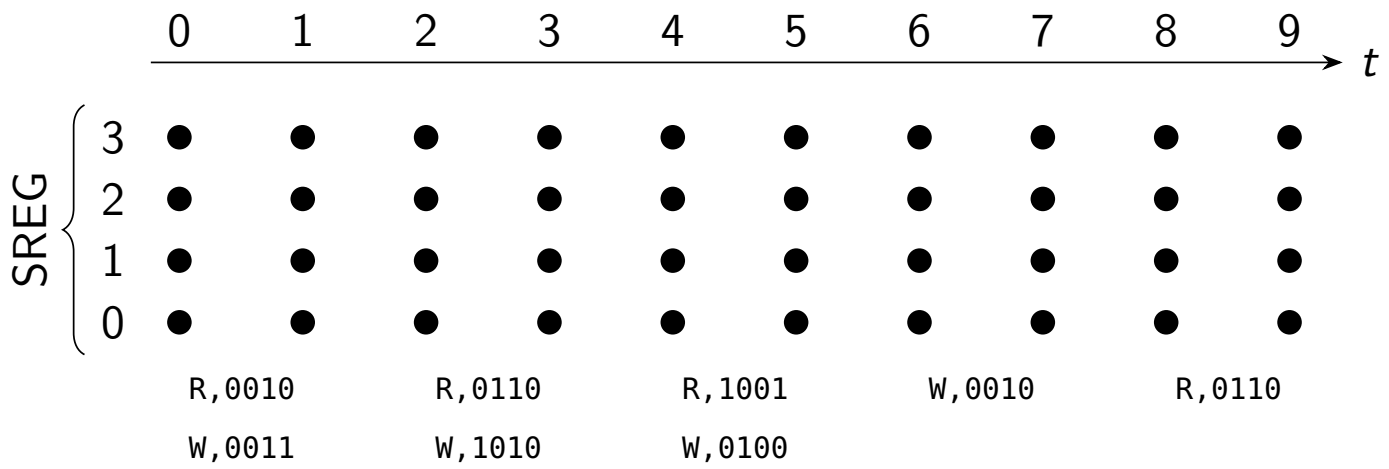
- Motivation
- SailFAIL: Model-Derived Fault Injectors
- Case-Study: CHERI RISC-V
- **Conclusion**

Conclusion

TUHH
Technische Universität Hamburg

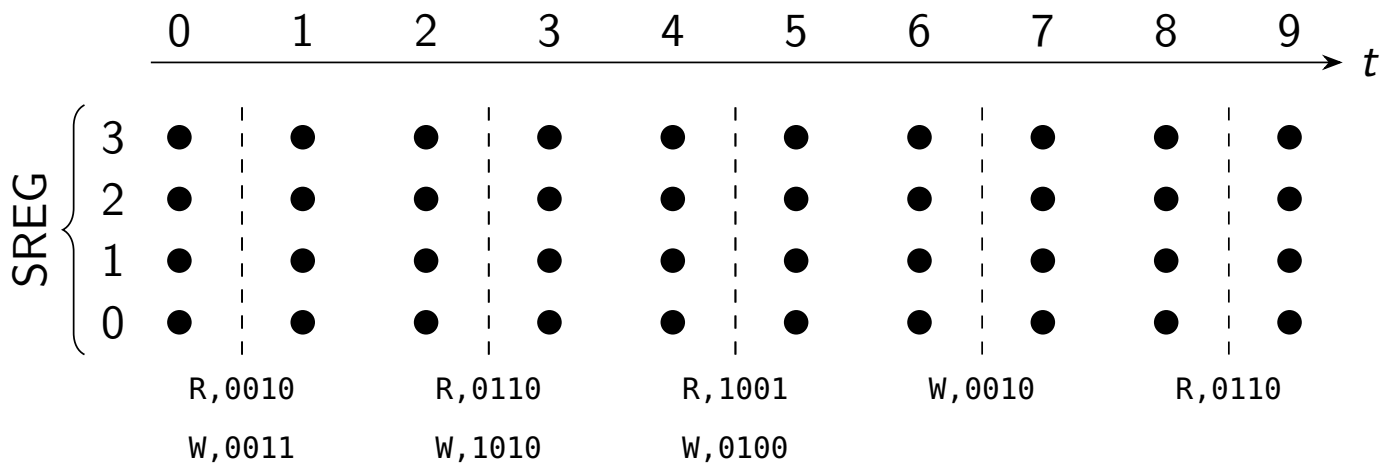
- SAFI requires simulator platform
  - Hard to obtain, maintain, and to get correct
  - Custom ISA Extensions require adapted tool chain
- SailFAIL: Derive platforms from formal Sail Models
  - Automatically introduce register access tracing
  - Bit-precise tracing and pruning of CPU registers
  - Five new backends for FAIL*
- Case-Study: Bubblesort on (CHERI-) RISC-V
  - CHERI: larger attack surface, but sometimes fosters robustness
  - Capabilities should contain a parity bit

Bit-Precise Def-Use Pruning



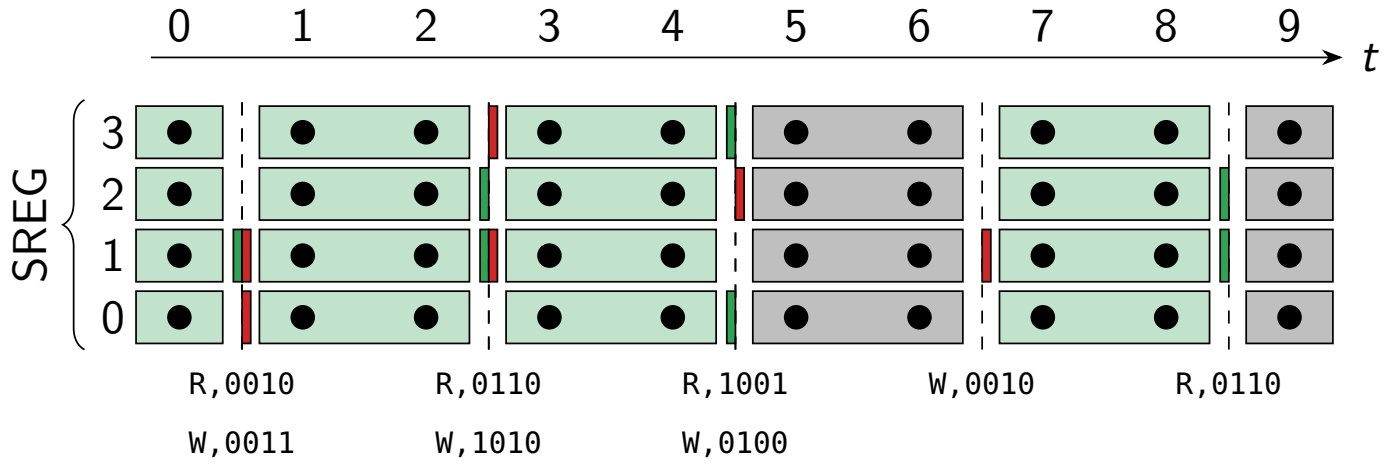
- Standard Fault Planing Technique: Def-Use Pruning
  - Partition fault space into equivalence intervals
  - One FI per interval that ends in a use/read (green)

Bit-Precise Def-Use Pruning



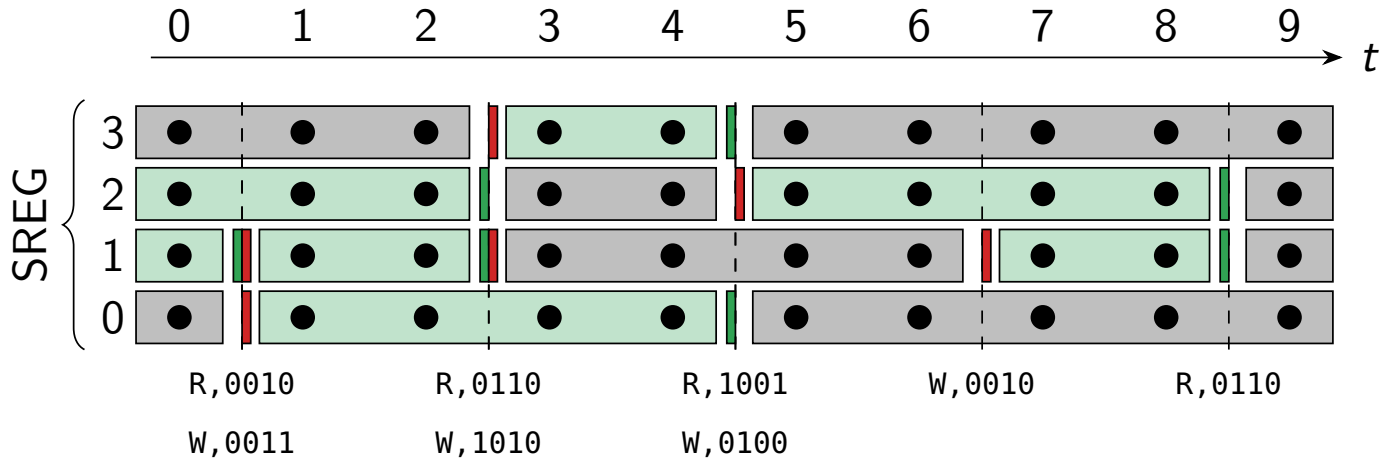
- Standard Fault Planing Technique: Def-Use Pruning
  - Partition fault space into equivalence intervals
  - One FI per interval that ends in a use/read (green)

Bit-Precise Def-Use Pruning



- Standard Fault Planing Technique: Def-Use Pruning
  - Partition fault space into equivalence intervals
  - One FI per interval that ends in a use/read (green)

Bit-Precise Def-Use Pruning



- Standard Fault Planing Technique: Def-Use Pruning
  - Partition fault space into equivalence intervals
  - One FI per interval that ends in a use/read (green)
- Better Pruning with Bit-Precise Access Tracing
  - Register bits are manipulated independently (e.g., CSRs)
  - Partition only if bit is actually accessed
- AVR: CRC32 over first 500 Fibonacci numbers
  - SREG: 1-bit access: 96.4 % instructions, 2-bit access: 3.6%
  - $9.42 \cdot 10^5$ injections $\Rightarrow$ $7.9 \cdot 10^5$ injections (-16.17 %)