EMSOFT, New York, 14/10/2019

Will My Program Break on This Faulty Processor?

Formal Analysis of Hardware Fault Activations in Concurrent Embedded Software

Levente Bajczi[1], András Vörös[1,2], *Vince Molnár* [1,2]

[1] FTSRG, Budapest University of Technology and Economics

[2] MTA-BME Lendület Cyber-physical Systems Research Group

Budapest University of Technology and Economics
Department of Measurement and Information Systems
FTSRG Research Group

Authors & Acknowledgments

Levente Bajczi

András Vörös

Vince Molnár

MŰEGYETEM 1782

ftsrg

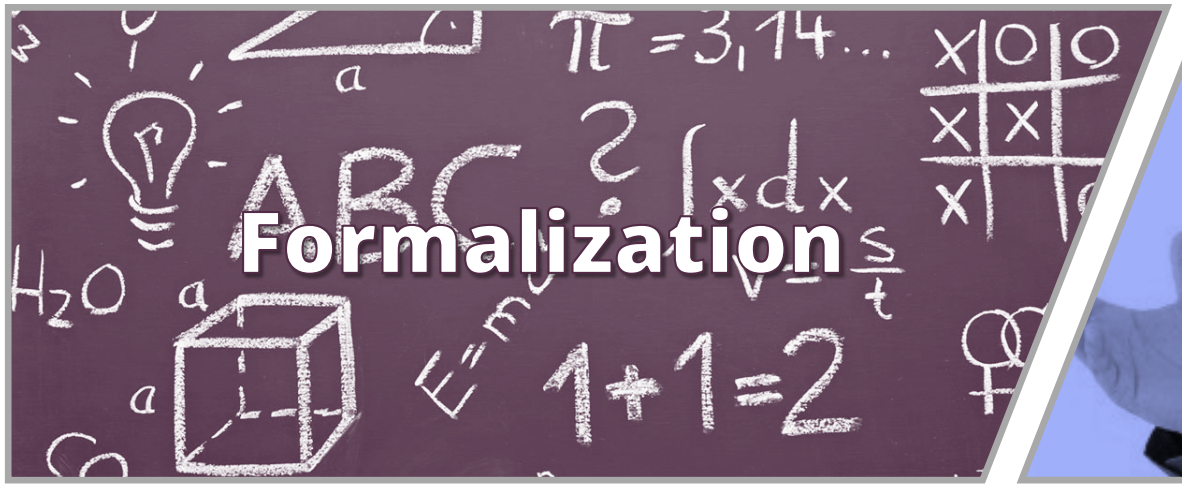
Hungarian Academy of Sciences

Acknowledgments. The research reported in this paper was partially supported by the BME – Artificial Intelligence FIKP grant of EMMI (BME FIKP-MI/SC) and the Nemzeti Tehetség Program, Nemzet Fiatal Tehetségeiért Ösztöndíj 2018 (NTP-NFTÖ-18).

Overview

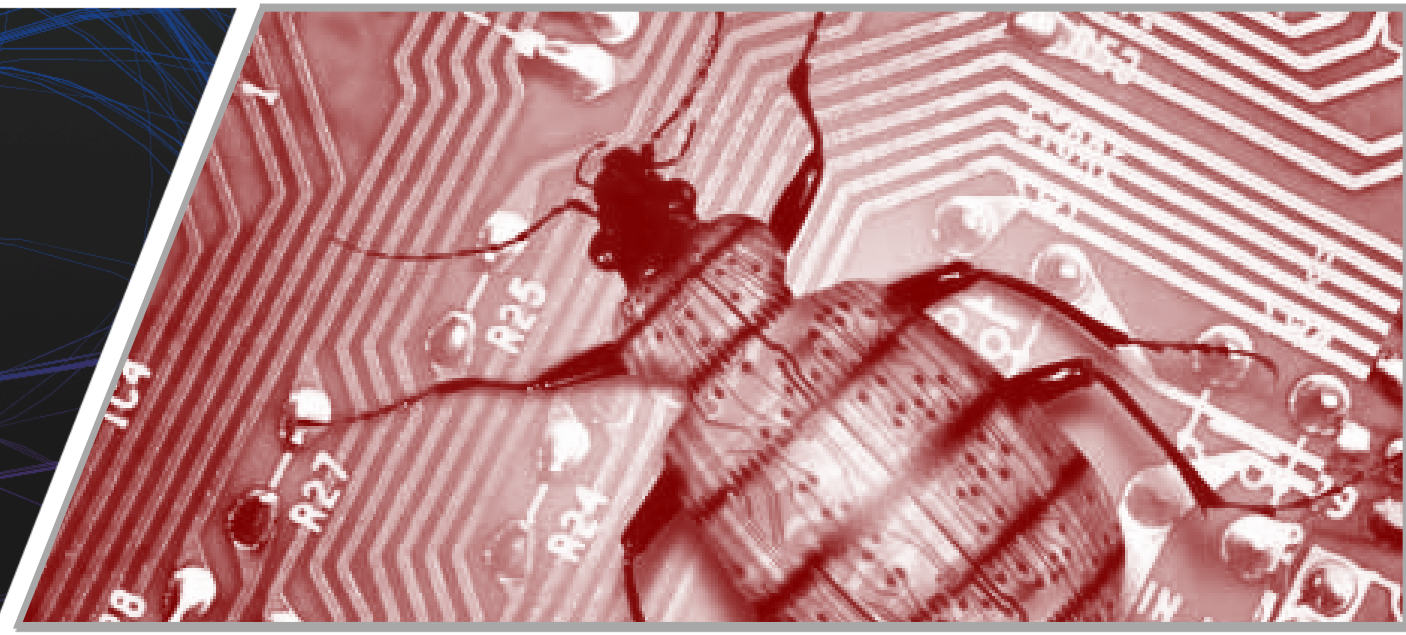

What is the Problem?


Why is It Interesting?

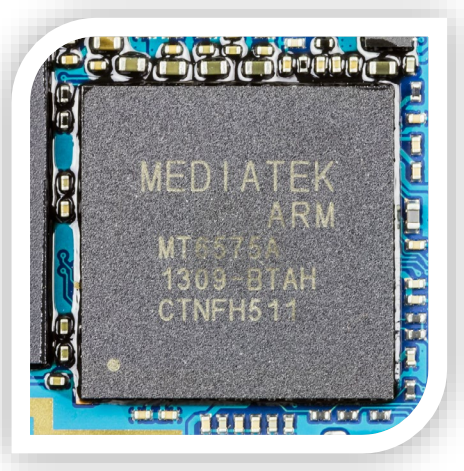

Formalization


(Lack of) Solutions

What is the Problem?



Motivation



ARM Cortex A9

Patched by compiler mappings

```
x = 0;
```

```
1. STORE(x, 1, rel);        1. r1 = LOAD(x, rel);
2. STORE(x, 2, rel);        2. r2 = LOAD(x, rel);
```

```
Forbidden: r1 = 2 && r2 = 1
```

Happens ~80 times in a billion runs

Undetected for **3 years** after release

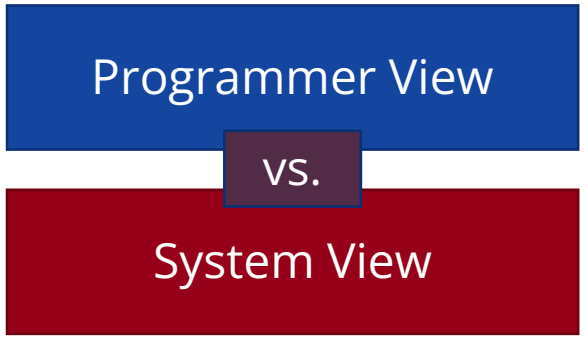
Theoretical Question

If I have a processor with some known problem…

…but I plan to ever run only a single application on it

Will My Program Break on This Faulty Processor?

- Sounds similar to concurrency issues
  - Nondeterministic activation – hard to catch
  - **But:** **activation of HW fault** vs. ~~coding error~~
  - Programmer's **assumptions** are violated
    - …and most analysis tools' assumptions as well

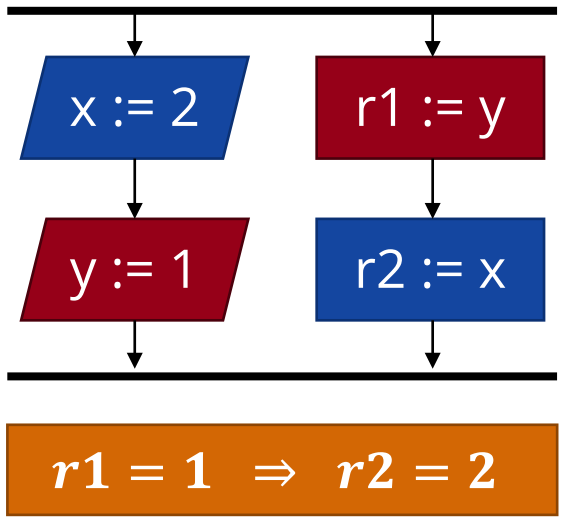


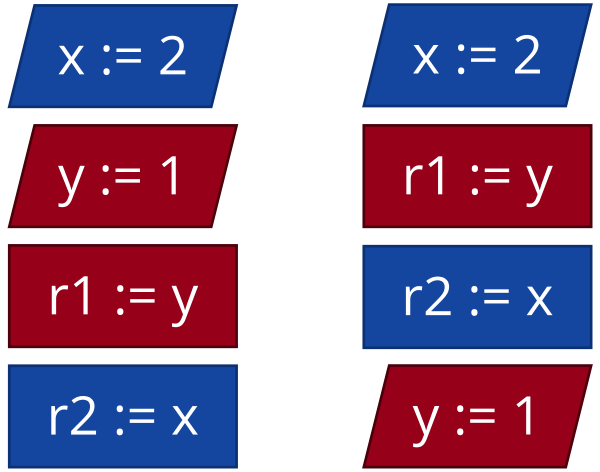
Focus: Memory Consistency Models

- A Memory Consistency Model (MCM) defines
  - How to order memory operations
  - Issued asynchronously by CPU cores
  - To serialize them for the shared memory

Litmus test:
Illustrates the guarantees of an MCM

Intuitive:
Sequential Ordering



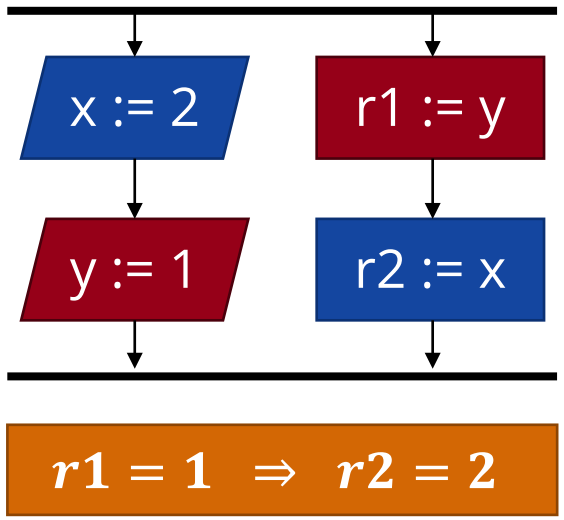


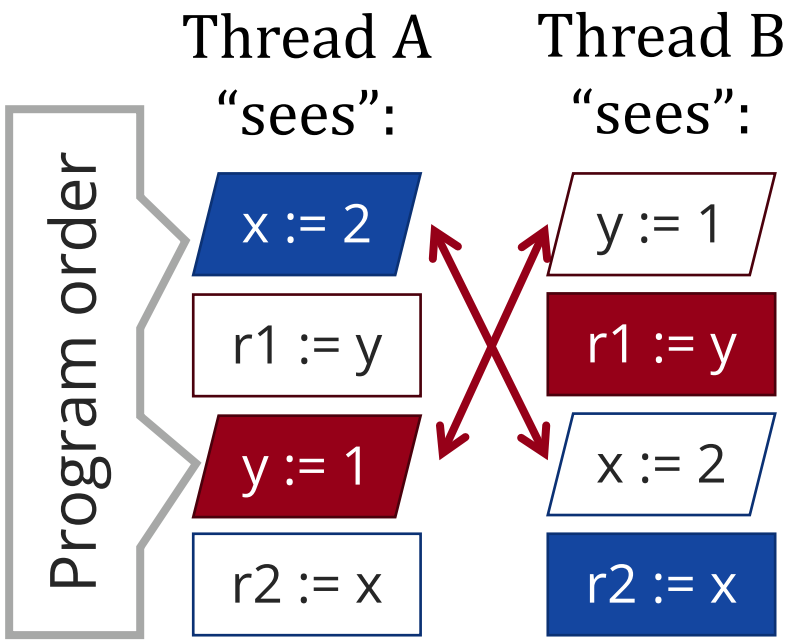
Focus: Memory Consistency Models

- A Memory Consistency Model (MCM) defines
  - How to order memory operations
  - Issued asynchronously by CPU cores
  - To serialize them for the shared memory

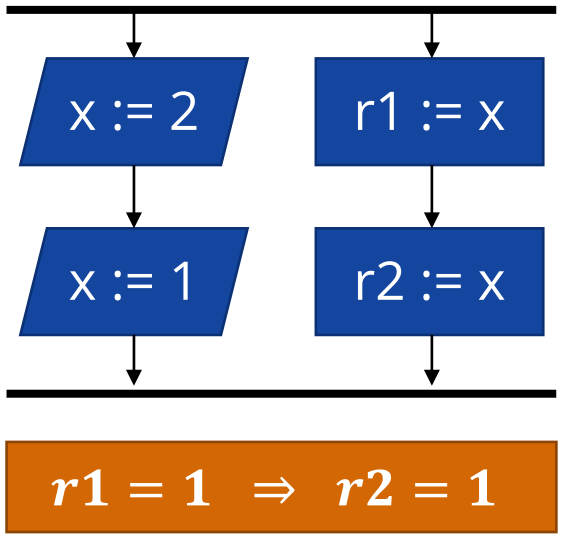
Litmus test:
Illustrates the guarantees of an MCM

Intuitive:
Sequential Ordering





Fast:
Weak Ordering



arm

RISC-V

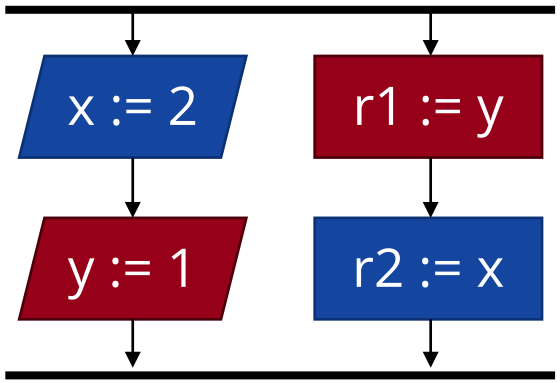
ftsrg

Focus: Memory Consistency Models

- A Memory Consistency Model (MCM) defines
  - How to order memory operations
  - Issued asynchronously by CPU cores
  - To serialize them for the shared memory

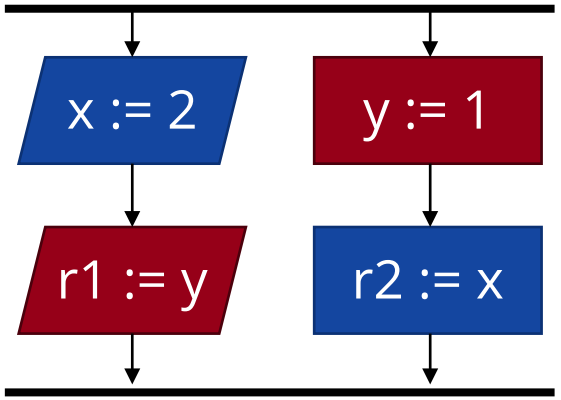
Litmus test:
Illustrates the guarantees of an MCM

Intuitive:
Sequential Ordering



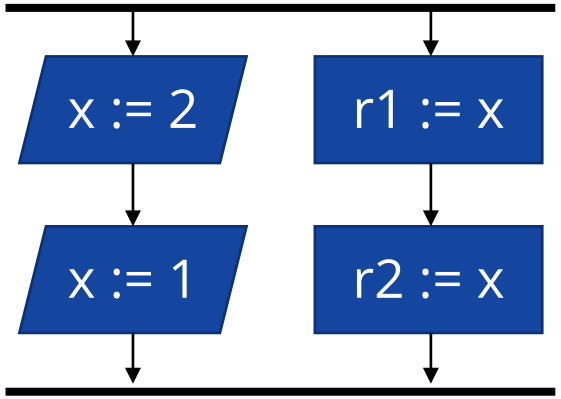
$r1 = 1 \Rightarrow r2 = 2$

Middle ground:
Total Store Order



$r1 = 1 \not\Rightarrow r2 = 2$

Fast:
Weak Ordering



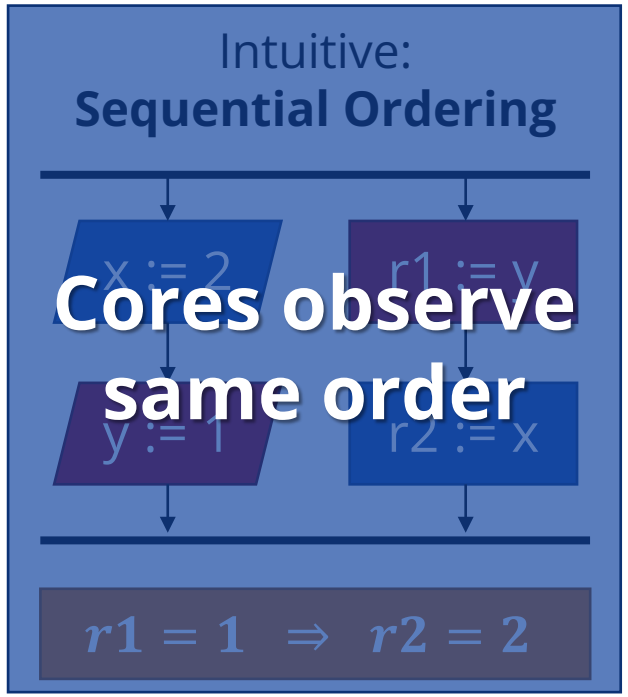
$r1 = 1 \Rightarrow r2 = 1$

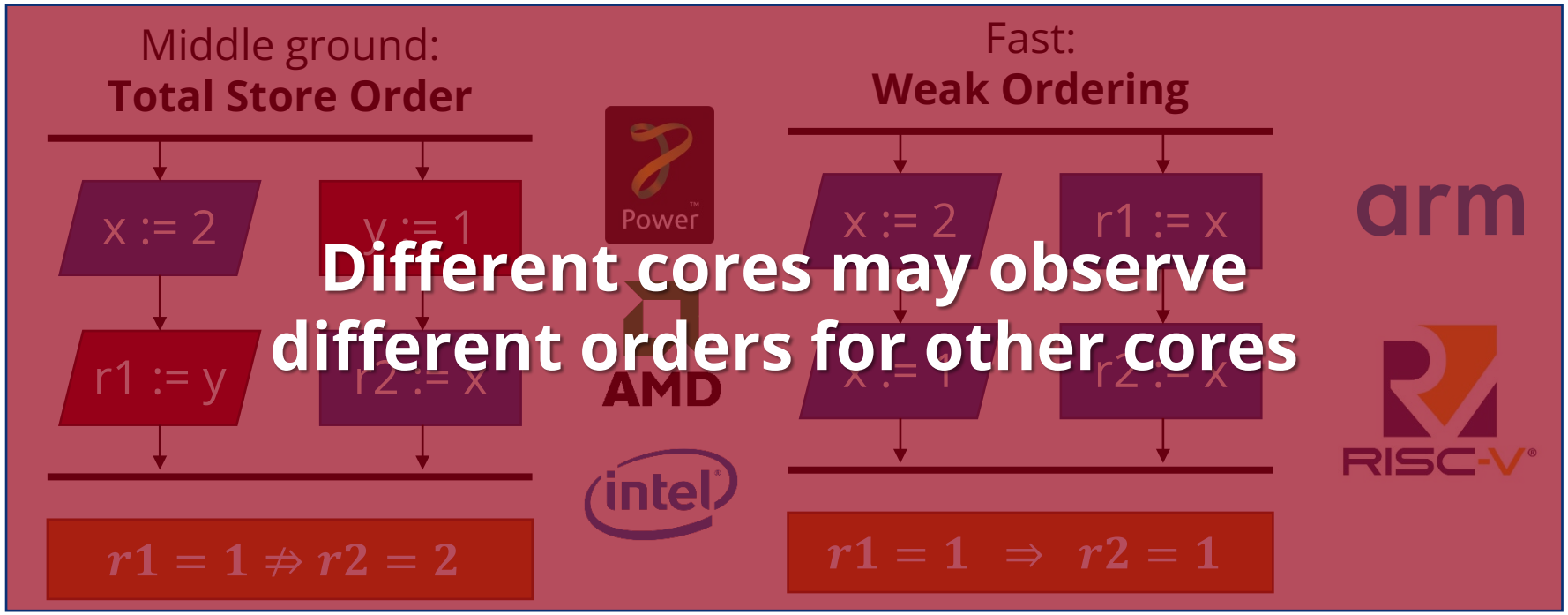
ftsrg

Focus: Memory Consistency Models

- A Memory Consistency Model (MCM) defines
  - How to order memory operations
  - Issued asynchronously by CPU cores
  - To serialize them for the shared memory

Litmus test:
Illustrates the guarantees of an MCM





Foundations: TriCheck

C. Trippel et al: ***TriCheck**: Memory Model Verification at the Trisection of Software, Hardware, and ISA*, ASPLOS 2017

- Formal verification of MCM implementations
  - Specification: C11 memory model
- Method: check **feasibility of forbidden outcomes** of litmus tests
  - Analyzes dependencies between pipeline stages of different operations
    - Compiler
    - Virtual Memory
    - Instruction Set Architecture (ISA)
    - Hardware implementation

	Observable	Non-observable
C11 forbidden	ERROR	OK
C11 allowed	OK	TOO STRICT

- Applied on early RISC-V design and revealed 100+ problems

Why is It Interesting?

Faulty Processors are with Us

- Multi-core CPU
  - Memory consistency models are **complex** and hard to get right
  - Concurrency-related issues are **hard to catch**
- Examples (related to MCM):
  - ARM Read-Read Hazard (recall the first slide)
  - Early version of RISC-V (revealed by TriCheck)
    - **Same address load** occasionally reordered (can be fixed from compiler)
    - Lack of **cumulative lightweight/heavyweight fences** (must change ISA)
    - **Some faults can be kept dormant by rewriting software**
- Custom, purpose-built microprocessors (ASIC)
  - Mass produced from customer-specified designs

Use Cases

I bought 100 000 processors and they turn out to be faulty…

…but I use it in an embedded system with only one program.

Customer ☹

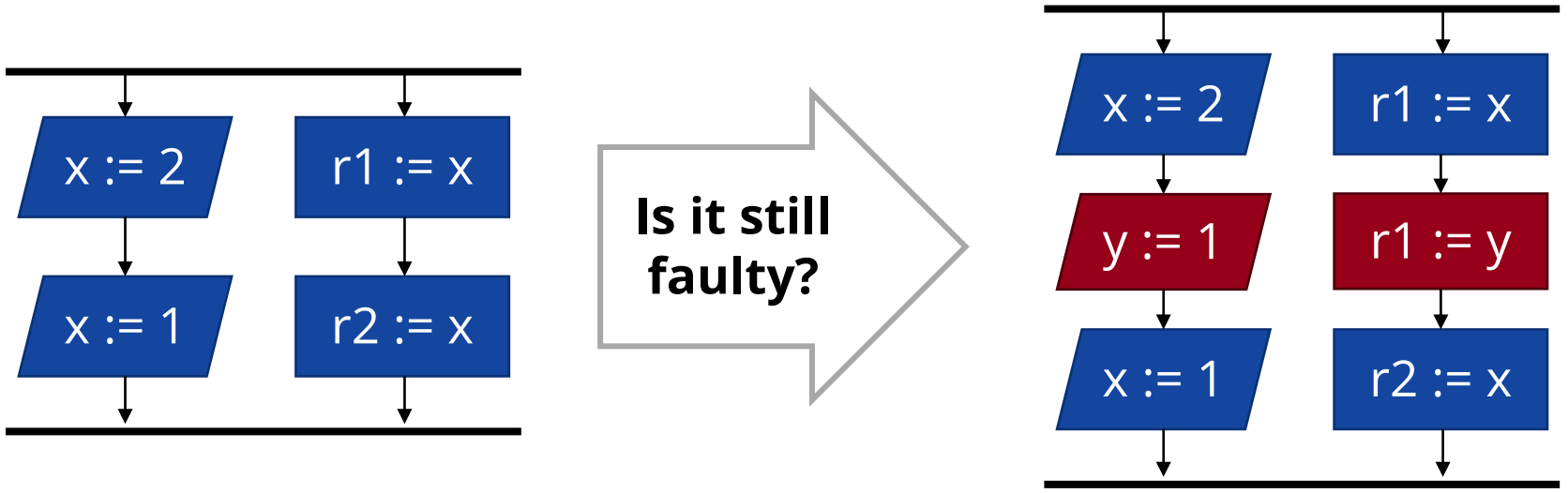
- Return to vendor?
  - I provided the design…
- Fix from software?
  - Which part is affected?
  - Is it affected in the first place?
  - What would be a good patch?

Vendor ☹

- Produce another 100 000?
  - What if it can be fixed from SW?
- I have to provide tooling
  - To diagnose programs
  - Locate fault activations
  - Patch automatically

Theoretically

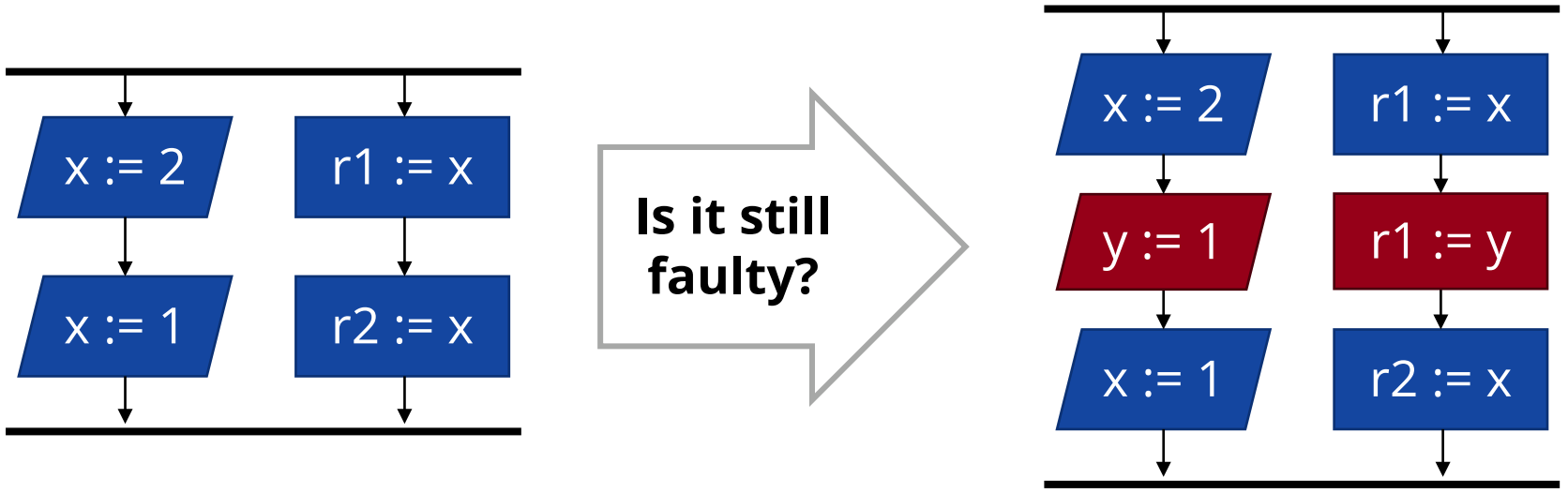
- This should be a simple model checking problem…
  - Check if the program does something like the fault-detecting litmus test
- …or is it?



  - Very few tools handle weak memory models in the first place
  - Those that do have hard-coded semantics
  - An ideal tool would take the (flawed) semantics as input as well

Theoretically

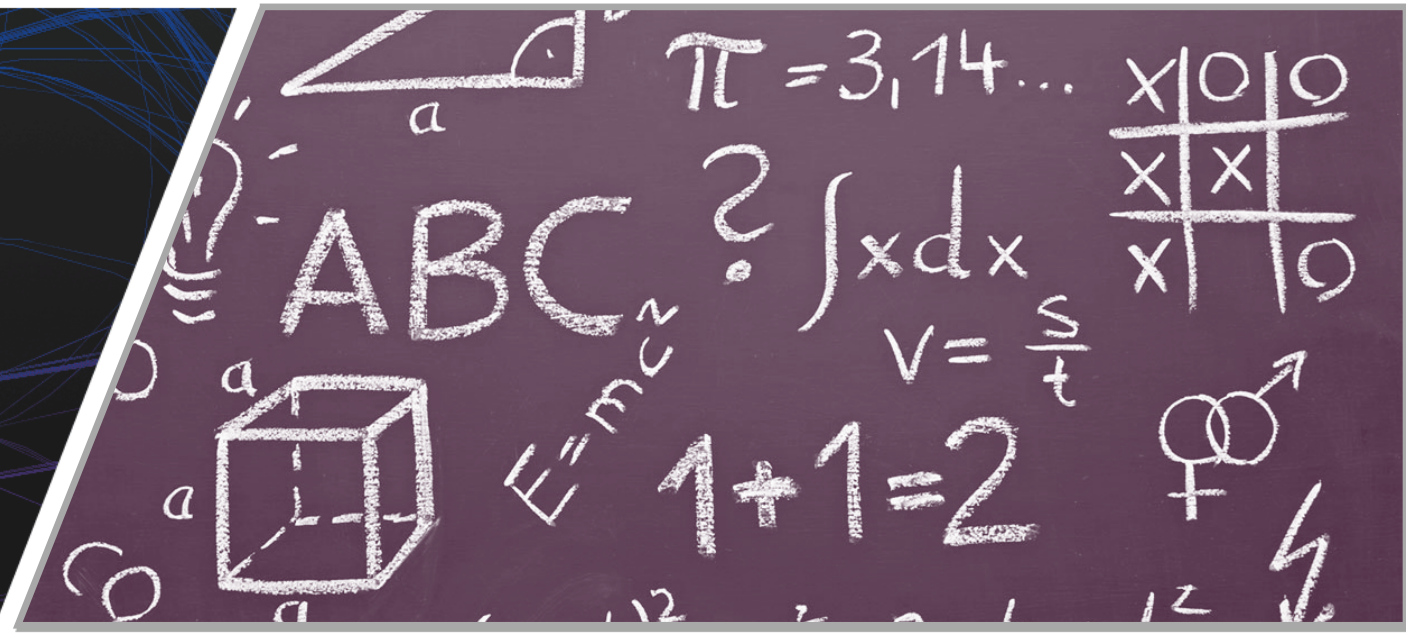
- This should be a simple model checking problem…
  - Check if the program does something like the fault-detecting litmus test
- …or is it?



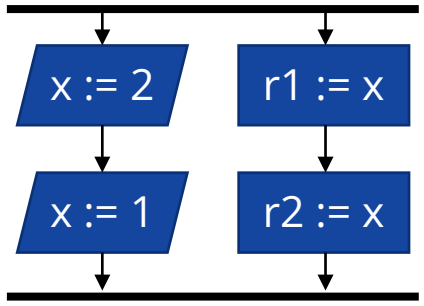
  - Very few tools handle weak memory models in the first place
  - Those that do handle a fixed semantics
  - An ideal tool would take the (flawed) semantics as input as well

This is currently not feasible with off-the-shelf tools

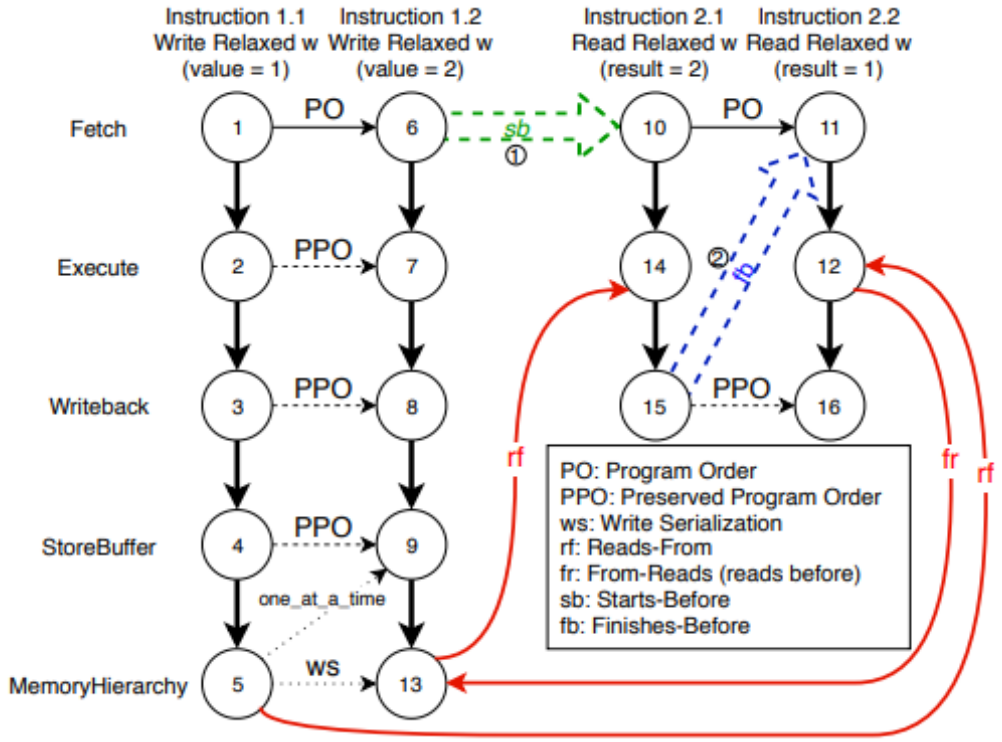
Formal Description



Fundamental Concepts

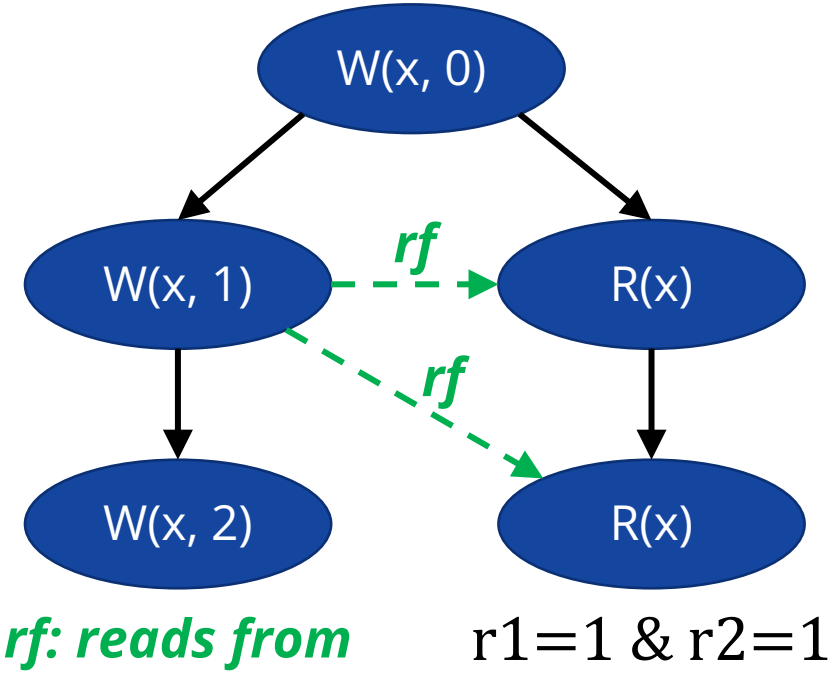


Microarchitecturally Happens-Before Graph



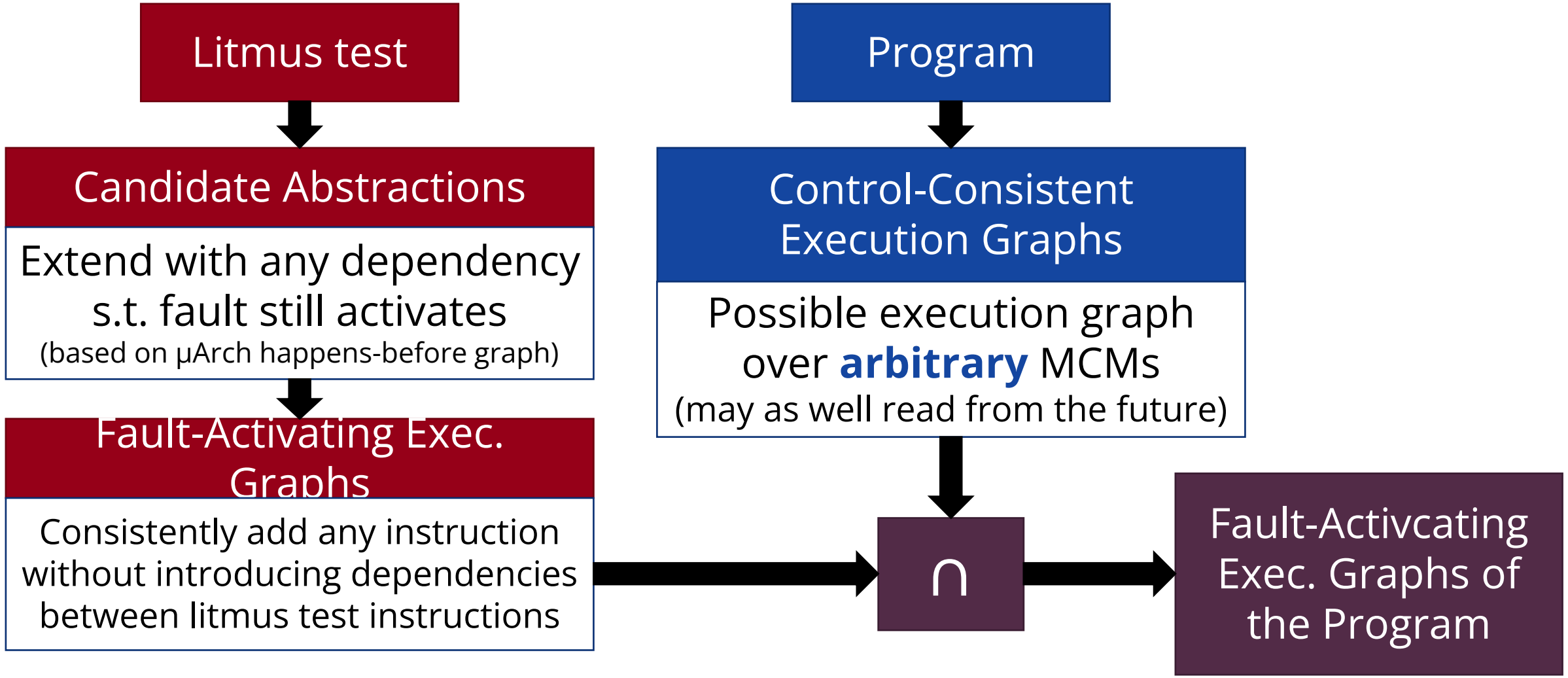
Based on verification results of TriCheck

(Consistent) Execution Graph

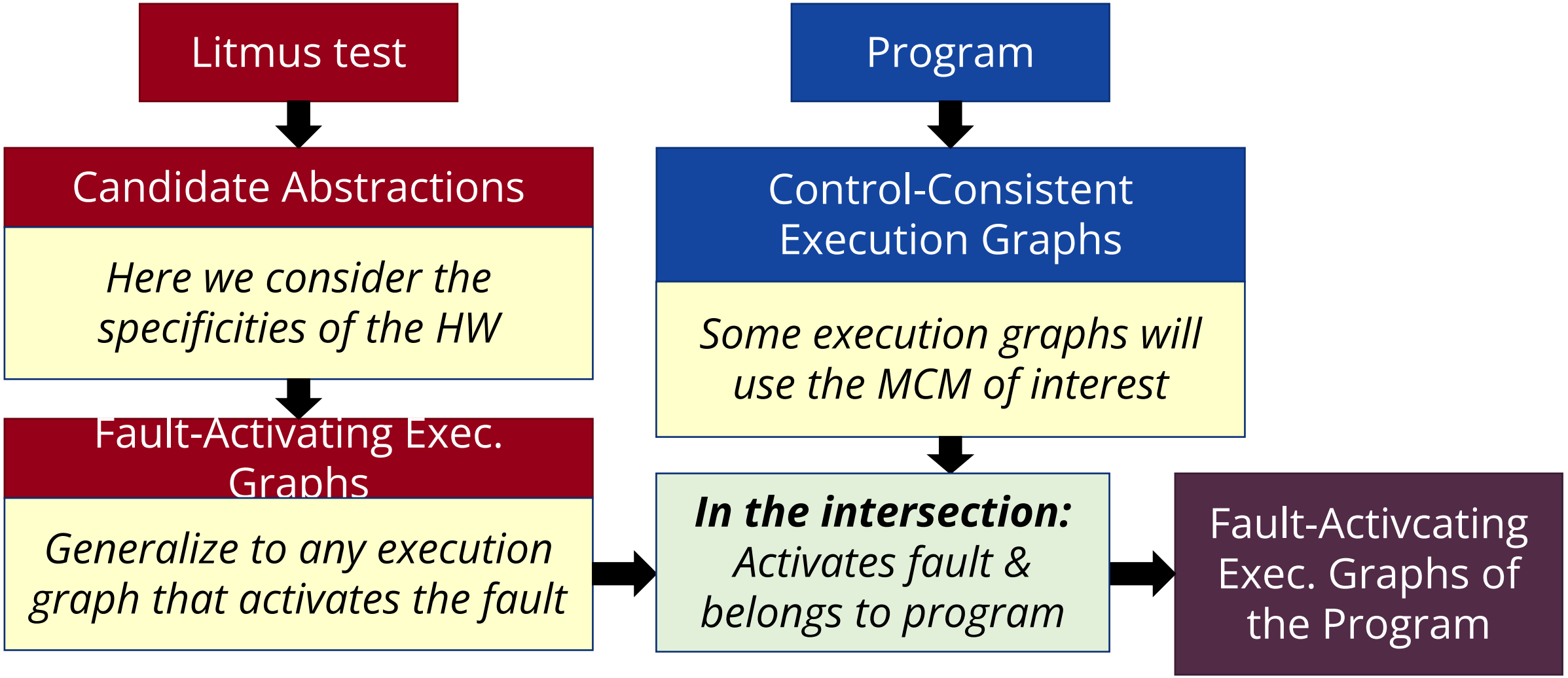


Based on M. Kokologiannakis et al: *Effective stateless model checking for C/C++ concurrency.* POPL 2017

Overview of Main Ideas (see paper for details)



Overview of Main Ideas (see paper for details)

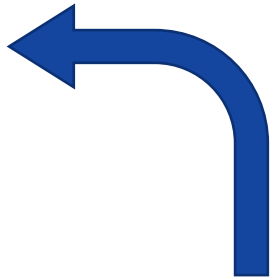


(Lack of) Solutions

Theoretical (Imaginary) Exact Solution

In the paper, we proposed an exact solution

- Building on:
  - C. Trippel et al: ***TriCheck****: Memory Model Verification at the Trisection of Software, Hardware, and ISA*, ASPLOS 2017
  - M. Kokologiannakis et al: *Effective* ***stateless model checking*** *for C/C++ concurrency.* POPL 2017
- Idea:
  - Reduce to pattern matching on consistent execution graphs
  - Iteratively refine matches by considering the HW model
- Stated and proved:



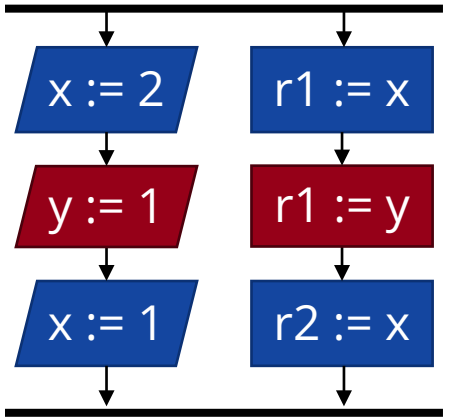
For every fault activating execution there is a correct execution that differs only in the incorrect memory operations.

Lack of Solutions with Model Checkers

Also attempted to implement with off-the-shelf model checkers

- Lack of support for weak memory models
  → **Incomplete solution** (weakly consistent executions are not explored)
- Even in the subset of sequentially consistent behaviors…
  - Pattern matching is **OK**
    - (although not easy)
  - We do not know whether the fault still activates without considering the HW model

  → Either **under-approximate** or **over-approximate**



We chose this to show actual problems

Performance

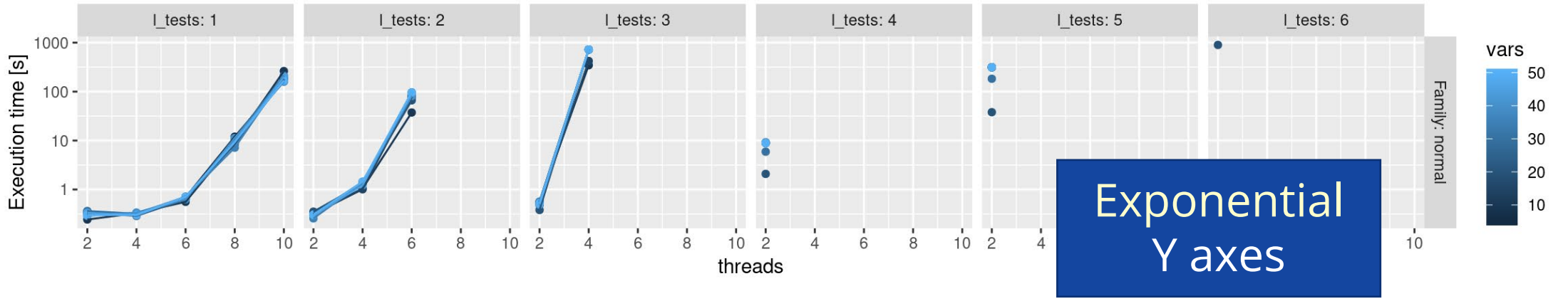
How good are model checkers in **parametric pattern-matching** of **multi-threaded** behaviors on a state space?

- We created very simple test programs
- Parameterized by #threads, #shared variables, #litmus tests
- Method:
  - Generate finite state automata from litmus tests
  - Look for accepting runs on state space
- Tools:
  - **spin**: explicit model checker with partil order reduction
  - **nuXmv**: symbolic model checker based on SAT/SMT solvers

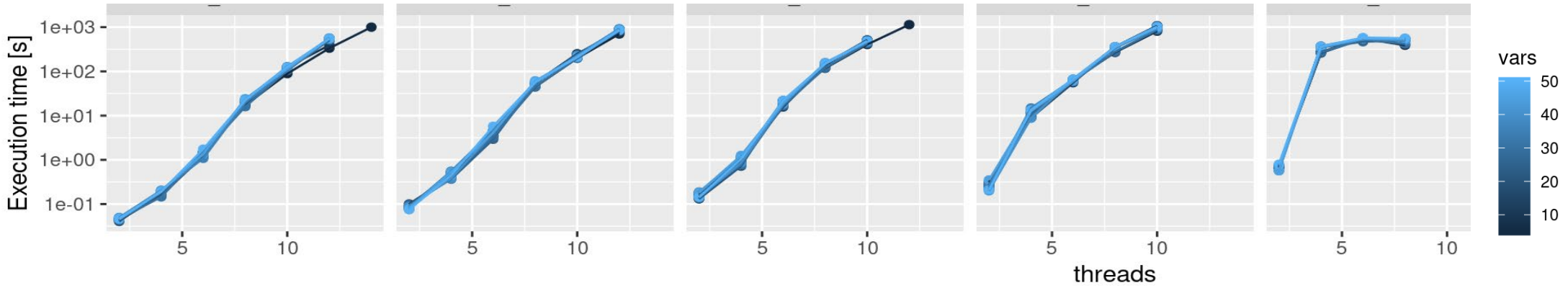
Performance

How good are model checkers in **parametric pattern-matching** of **multi-threaded** behaviors on a state space?

spin



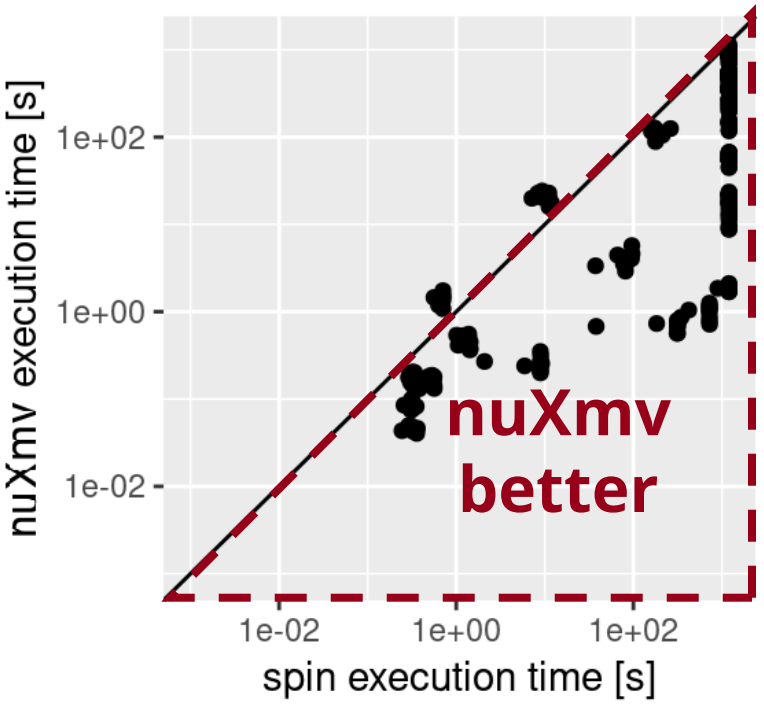
nuXmv



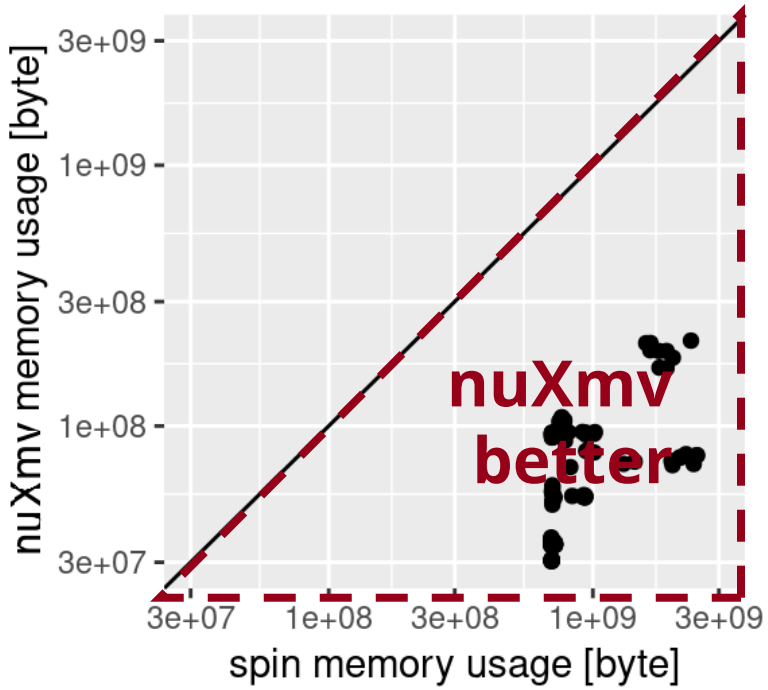
Performance

How good are model checkers in **parametric pattern-matching** of **multi-threaded** behaviors on a state space?

Execution time



Memory usage



Conclusions

Problem

Will My Program Break on This Faulty Processor?

– Focus: Memory Consistency Models

Use Cases

Detection of Fault Activations

– Localization, patch generation

Formally

Compute Fault-Activating Execution Graphs of Program

– Fault described by litmus test

– Harware model used to analyze activation

Solutions?

No Off-the-Shelf Solutions

– Proposed theoretical solution

– Today's model checkers not yet capable