SVM: A Synthesizable Approach to Efficient RISC-V CPU Verification

Yinan Xu

Institute of Computing Technology, Chinese Academy of Sciences

Beijing Institute of Open Source Chip

June 2026, Bologna, Italy

Verification: A Key Bottleneck in Chip Development

206%

2007 to present verification engineer growth

5:1

Verification-design engineer ratio

86%

Failed first silicon chip projects

75%

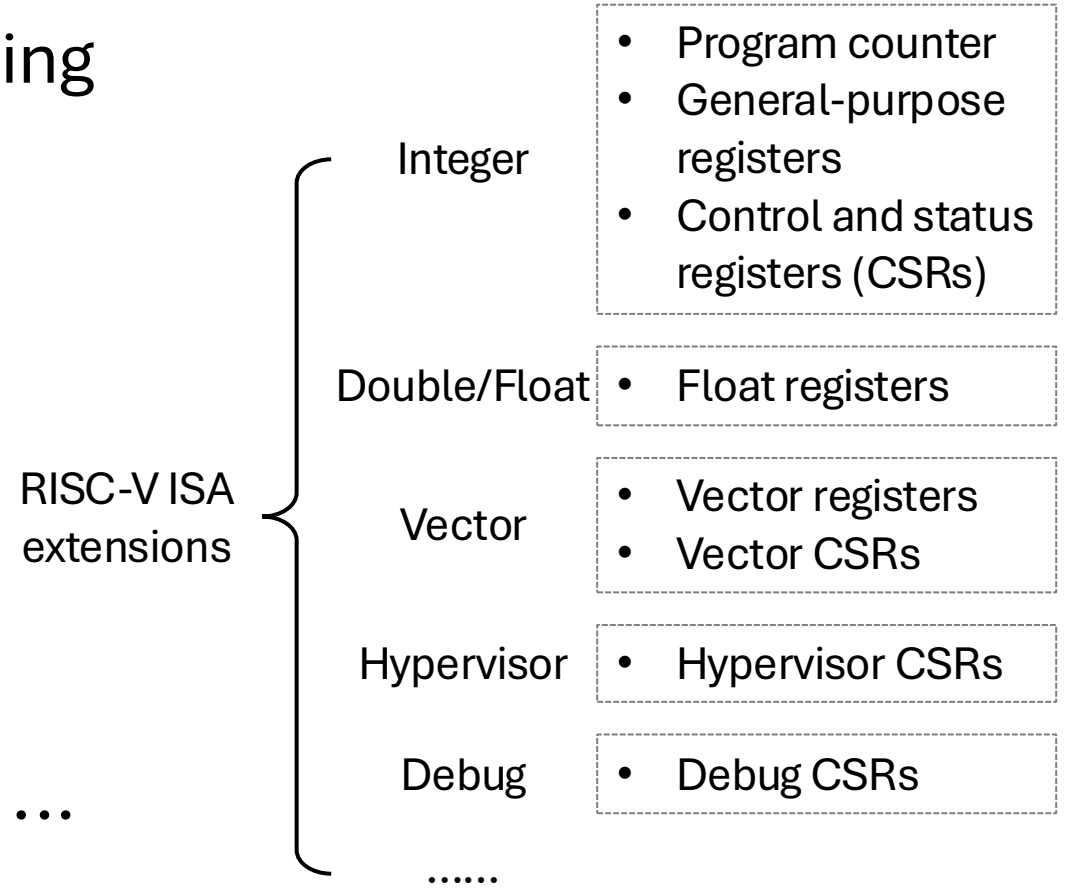
Exceeded planned schedule chip projects

Despite heavy investment, chip verification quality and efficiency still fall short of expectations

Source: https://resources.sw.siemens.com/en-US/white-paper-2024-wilson-research-group-ic-asic-functional-verification-trend-report/

Trend #1: Rising ISA Complexity

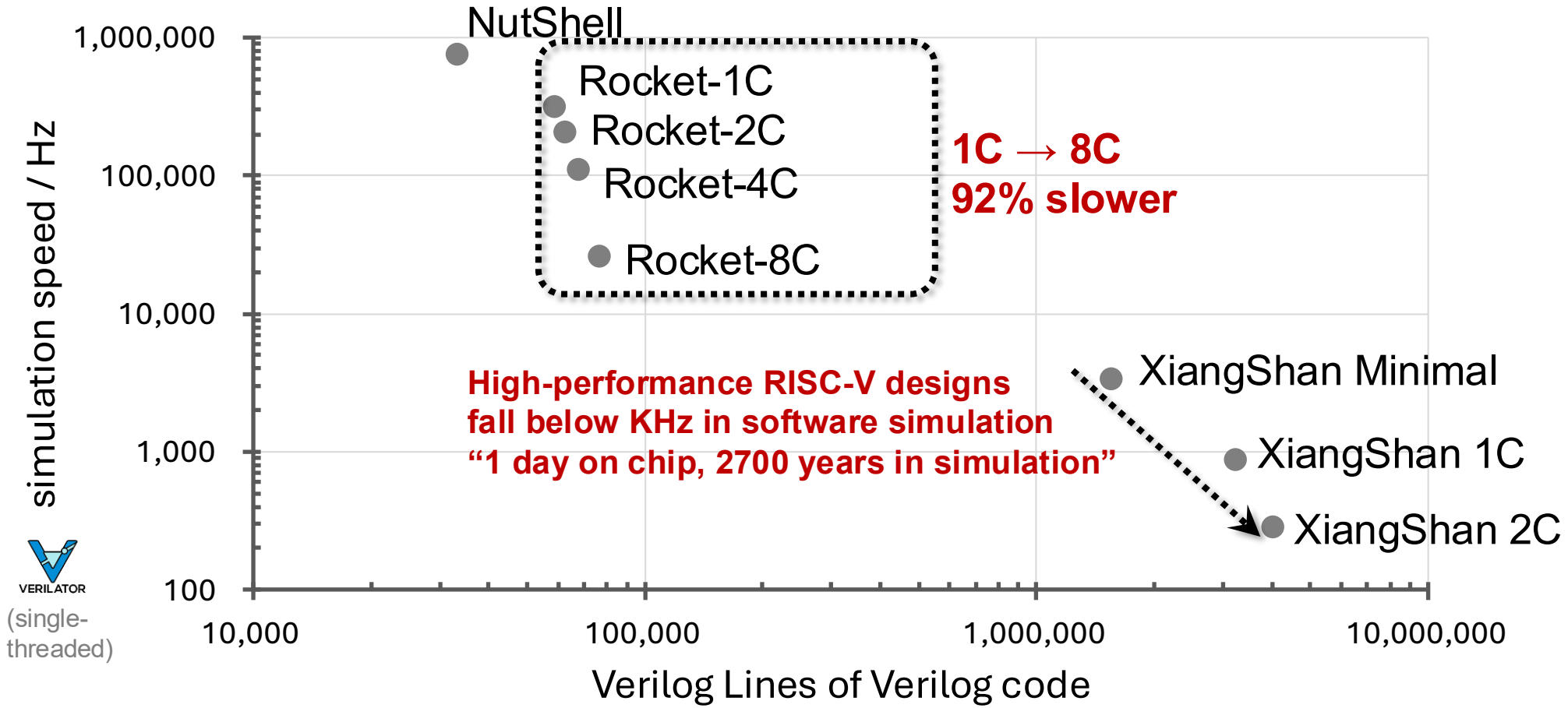
- RISC-V ISA complexity is rapidly increasing
- Example: RVA23 Profile
  - 57 mandatory extensions
  - 830-page ISA manual
  - 2X the length of the 2019 manual
- Features, variables, optional behaviors, …



Question: how to verify?

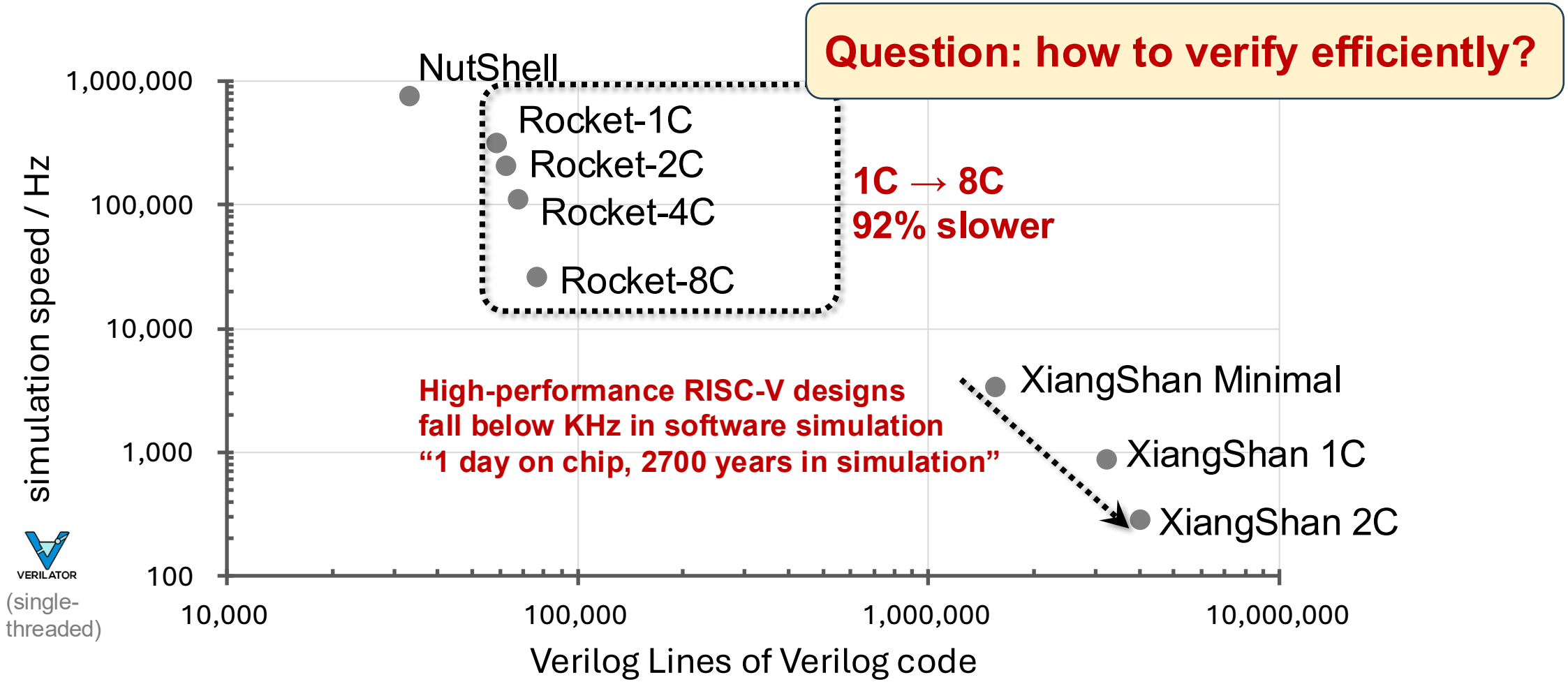
Trend #2: Slow RTL Simulation

- As processor scale grows, software simulation speed drops sharply



Trend #2: Slow RTL Simulation

- As processor scale grows, software simulation speed drops sharply



Background: Co-Simulation Verification (DiffTest)

RTL (Verilog, …)

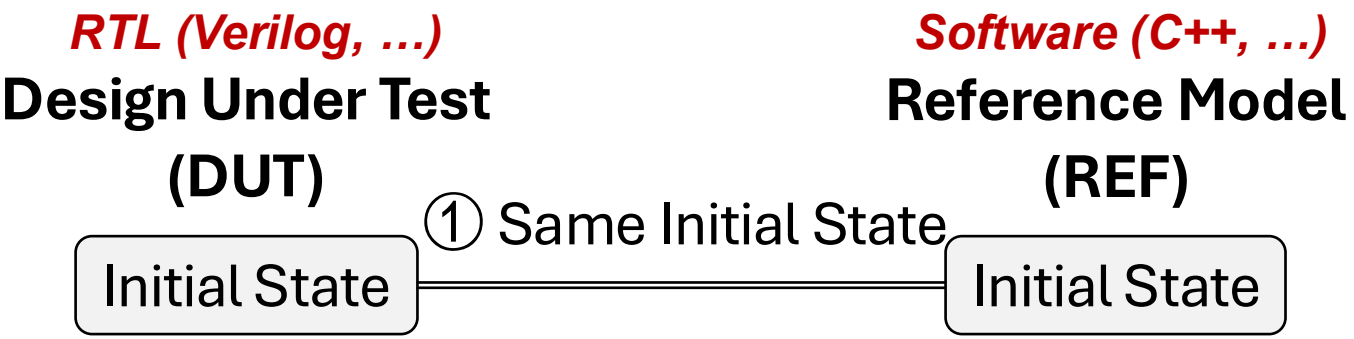
Design Under Test

(DUT)

Initial State

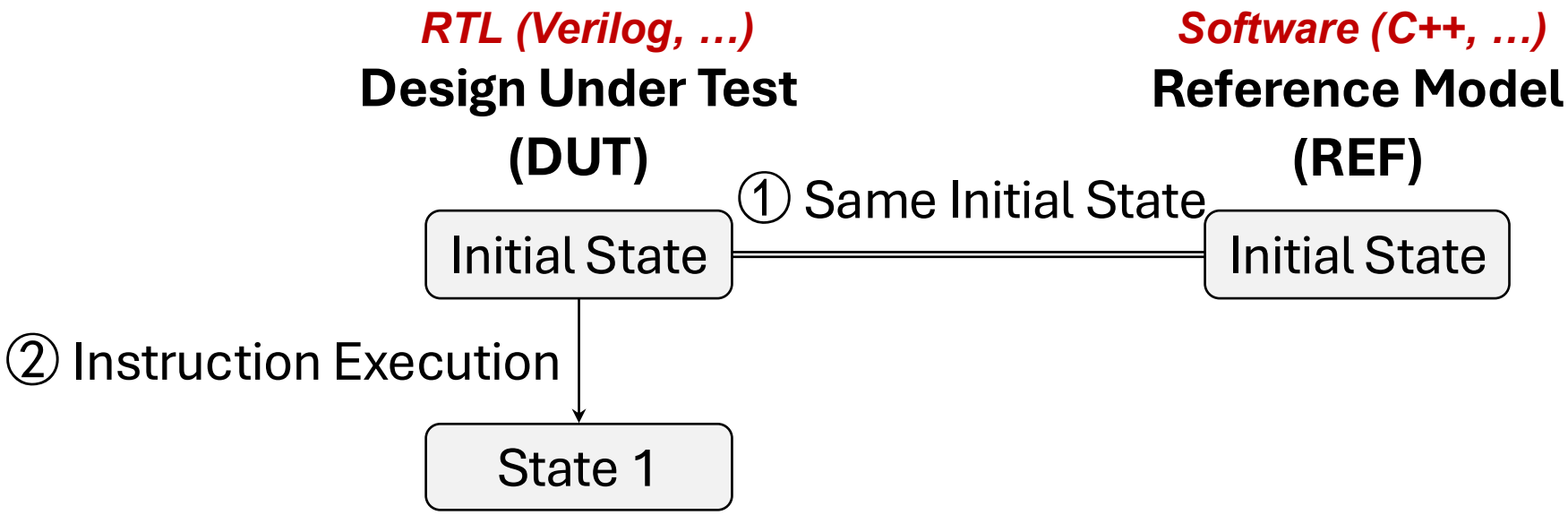
Repo: https://github.com/OpenXiangShan/difftest

Background: Co-Simulation Verification (DiffTest)



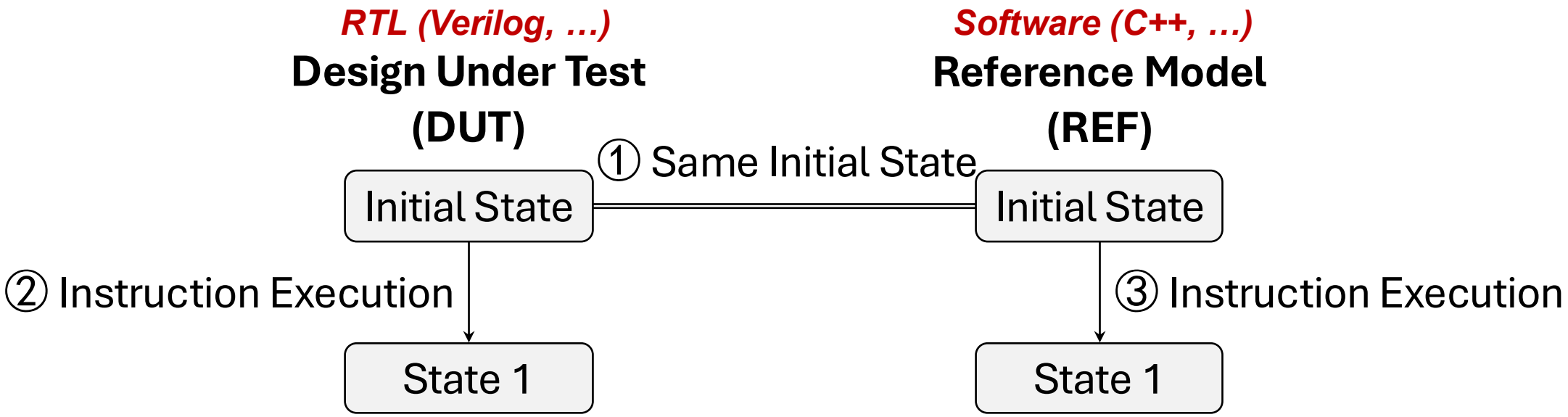
Repo: https://github.com/OpenXiangShan/difftest

Background: Co-Simulation Verification (DiffTest)



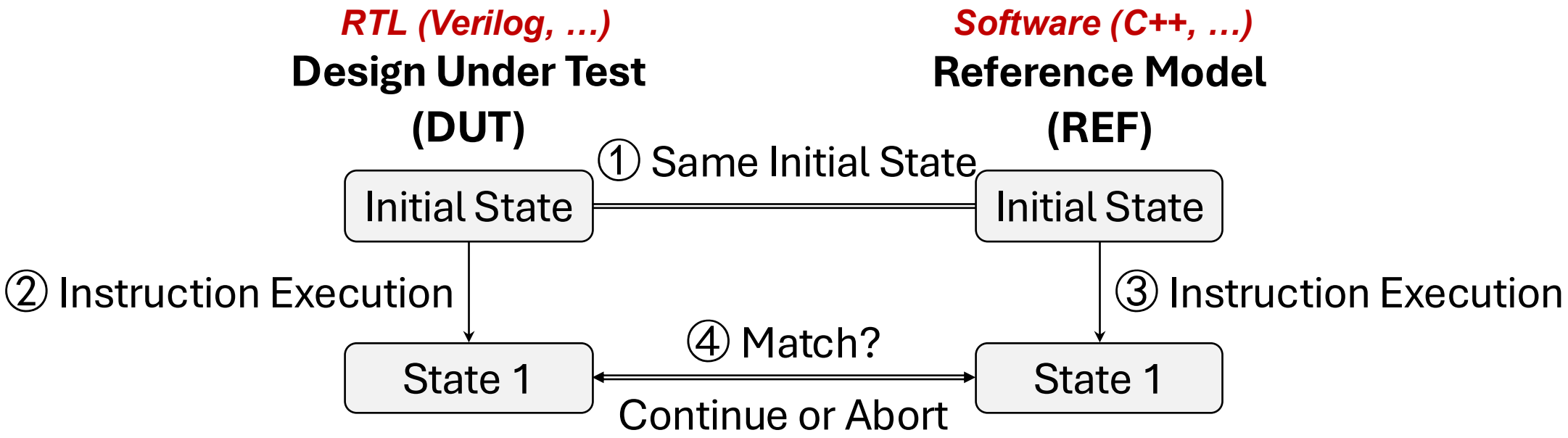
Repo: https://github.com/OpenXiangShan/difftest

Background: Co-Simulation Verification (DiffTest)



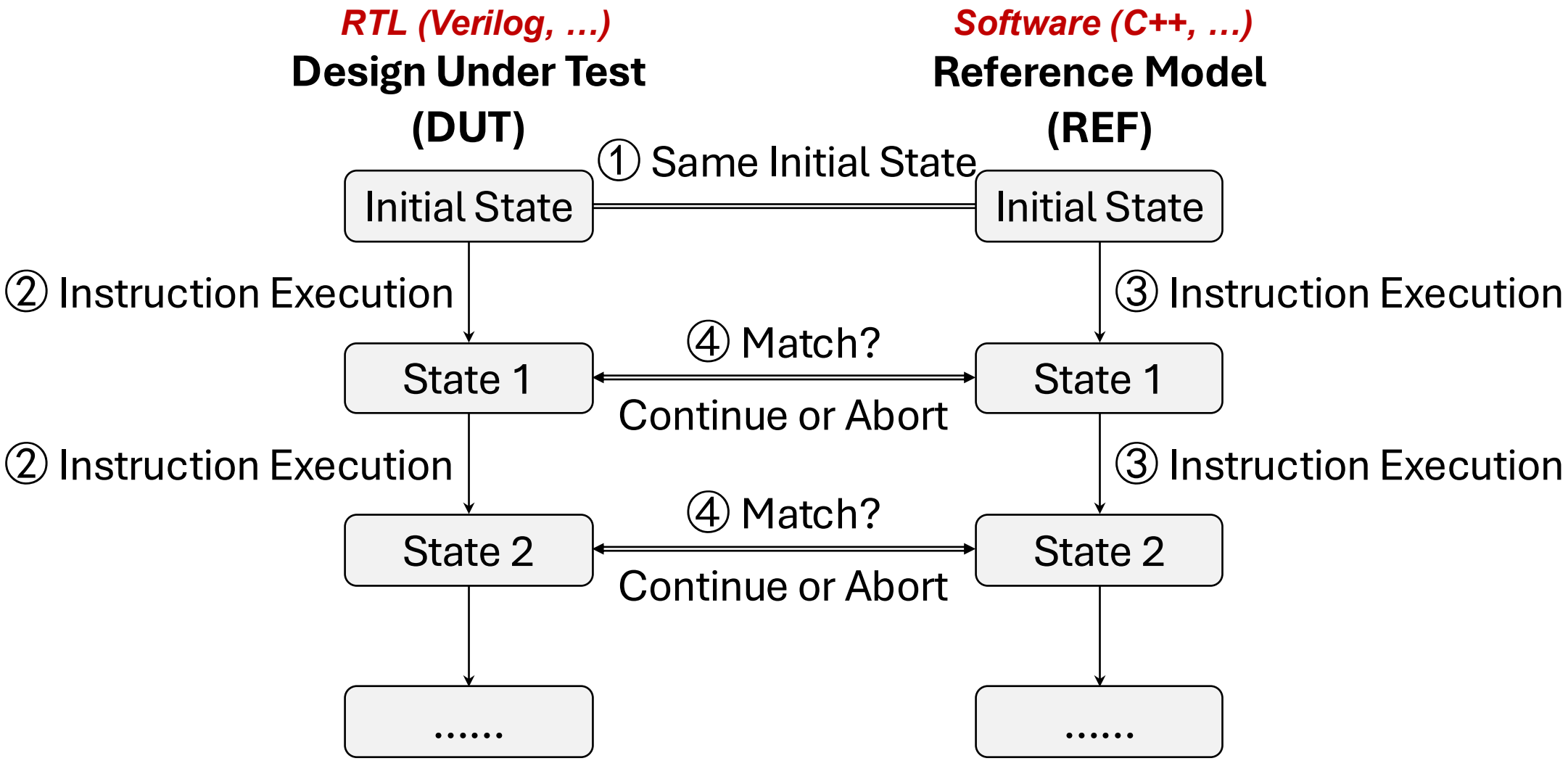
Repo: https://github.com/OpenXiangShan/difftest

Background: Co-Simulation Verification (DiffTest)



Repo: https://github.com/OpenXiangShan/difftest

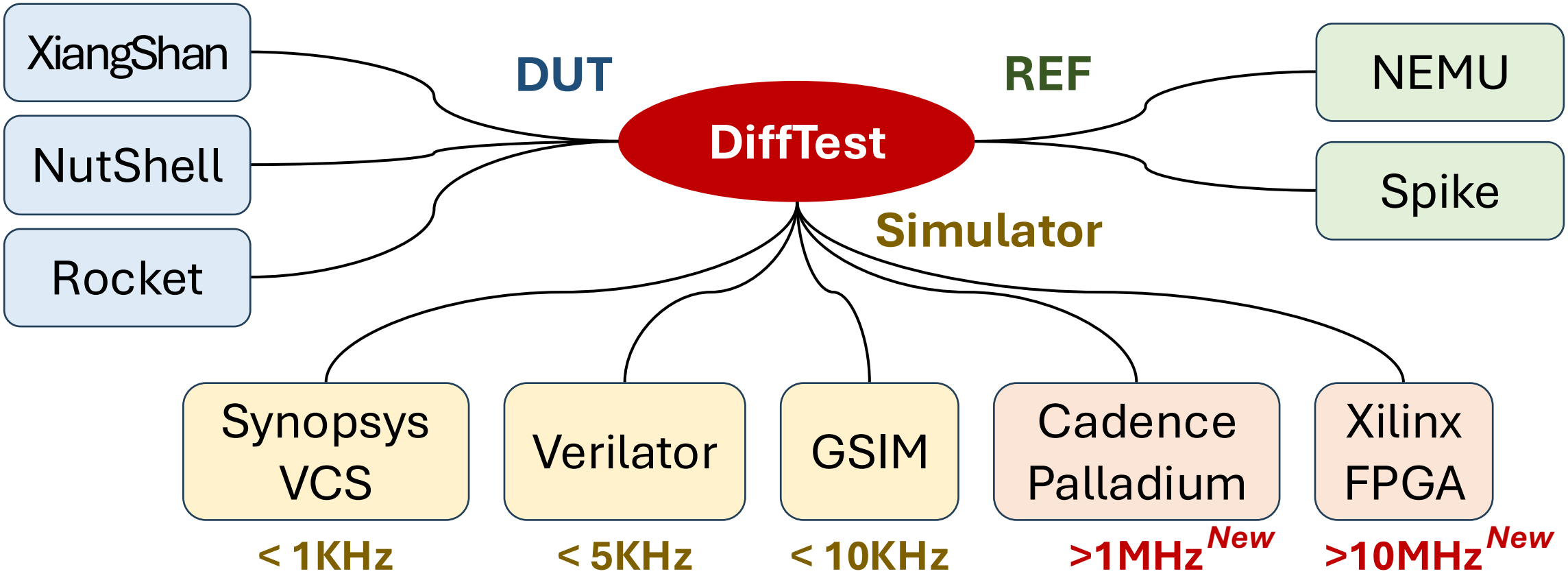
Background: Co-Simulation Verification (DiffTest)



Repo: https://github.com/OpenXiangShan/difftest

DiffTest-H: Emulator/FPGA-accelerated verification

- DiffTest now supports ***hardware-accelerated*** co-simulation
  - Today 10:55-11:05 at Demo Theater, *DiffTest-H: FPGA-Accelerated RISC-V Co-simulation Verification Beyond 10 MHz*



Hardware-Accelerated Processor Verification

- Key idea: accelerate RTL simulation by orders of magnitude with FPGAs
- Architecture: keep REF on the host; communicate with RTL
  - RTL side: emulators / FPGAs accelerate DUT simulation
  - Host side: x86 / ARM CPUs run REF software logic
  - Link: PCIe, Ethernet, InfiniBand, etc.

- Emulator, FPGA, etc
- 1~100 MHz

- x86/ARM CPU
- >> 100 MHz

Hardware-Accelerated Processor Verification

- Key idea: accelerate RTL simulation by orders of magnitude with FPGAs
- Architecture: keep REF on the host; communicate with RTL
  - RTL side: emulators / FPGAs accelerate DUT simulation
  - Host side: x86 / ARM CPUs run REF software logic
  - Link: PCIe, Ethernet, InfiniBand, etc.

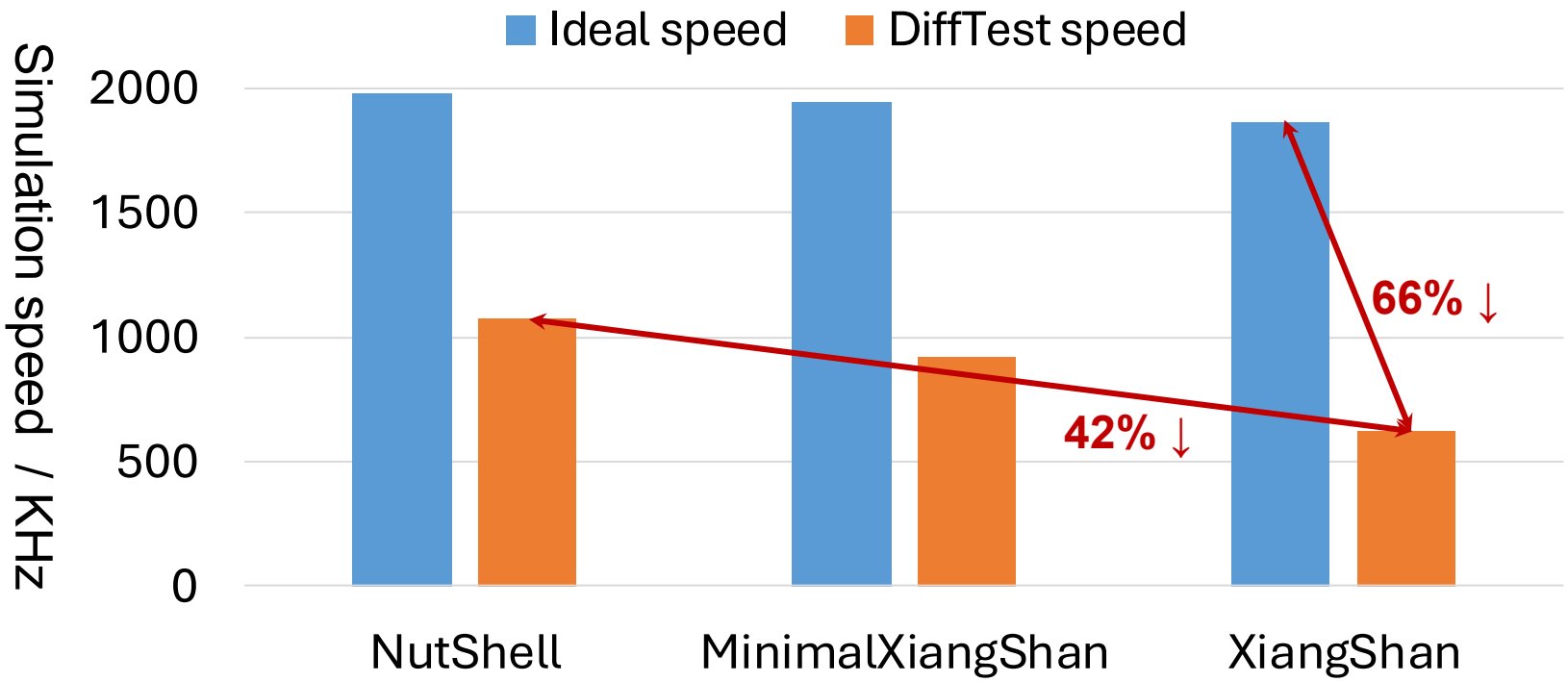
- Emulator, FPGA, etc
- 1~100 MHz

- x86/ARM CPU
- >> 100 MHz

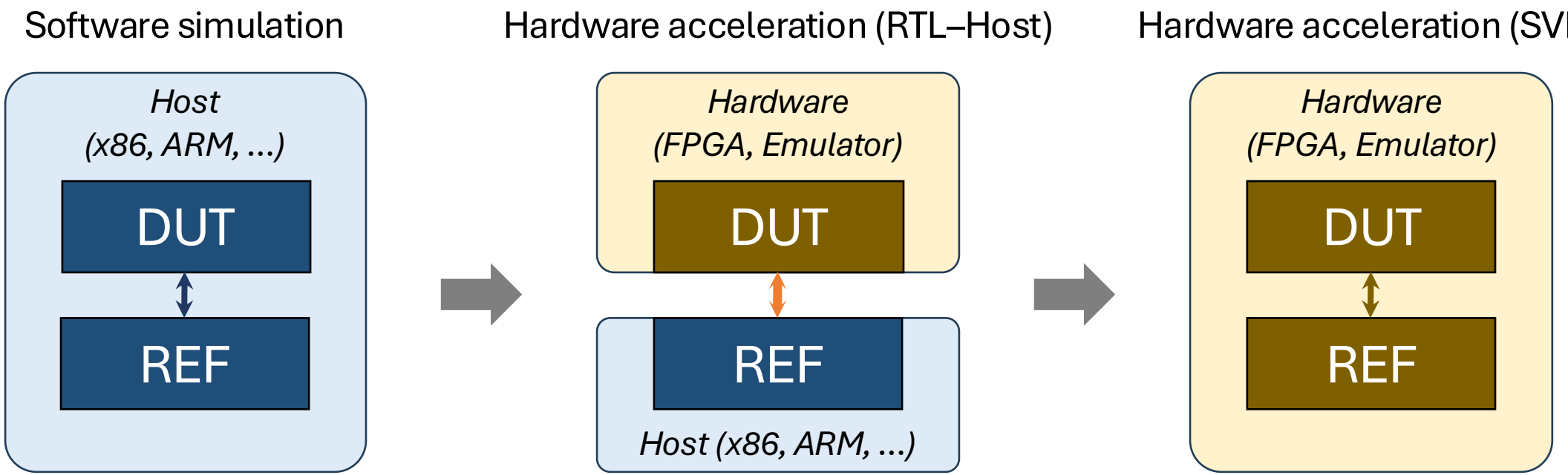
Limitation: verification data communication creates simulation performance overhead

RTL-Host architecture: communication overhead

- DiffTest-H gives limited acceleration for complex XiangShan CPUs
- Example: Cadence Palladium results
  - Even after cutting communication by 99.8%, speed is still below ideal and scales poorly



Ours: Synthesizable Verification Method (SVM)



Native Slow

Fast yet Link-Constrained

Native Fast

This work: Synthesizable Verification Method (SVM)

- A different approach: implement all co-simulation logic as synthesizable hardware
- Verification-data communication becomes on-chip wiring with no overhead

- Emulator, FPGA, etc
- 1~100 MHz

- x86/ARM CPU
- >> 100 MHz

This work: Synthesizable Verification Method (SVM)

- A different approach: implement all co-simulation logic as synthesizable hardware
- Verification-data communication becomes on-chip wiring with no overhead

- Emulator, FPGA, etc
- 1~100 MHz

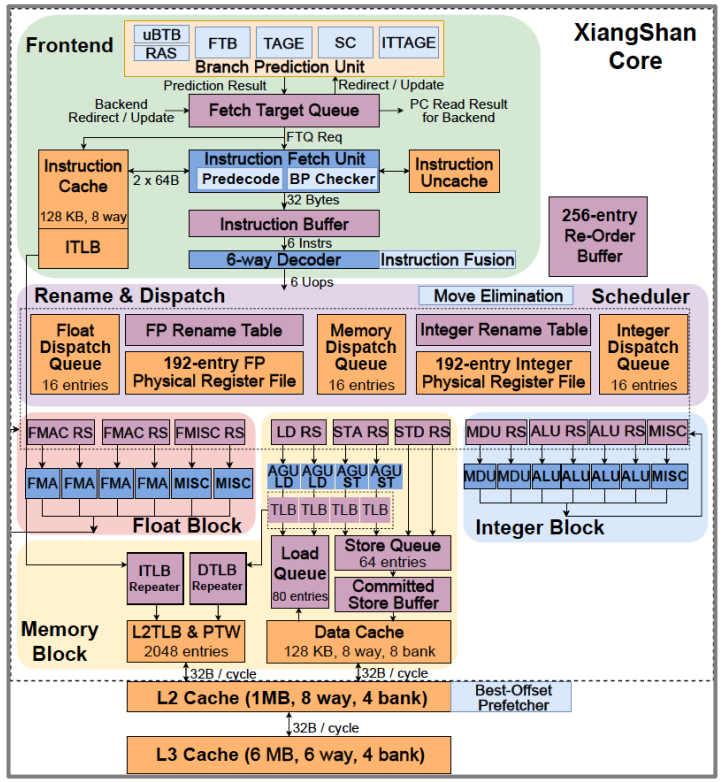
- x86/ARM CPU
- >> 100 MHz

- **Two challenges of implementing SVM**
  - ***Architecture***: ensure hardware REF execution efficiency
  - ***Coding***: ensure hardware REF correctness and debuggability

Challenge #1: Hardware REF Execution Efficiency

- Co-simulation **instruction throughput** is limited by the slower of DUT and REF
- DUT: *complex* microarchitecture with high instruction execution efficiency

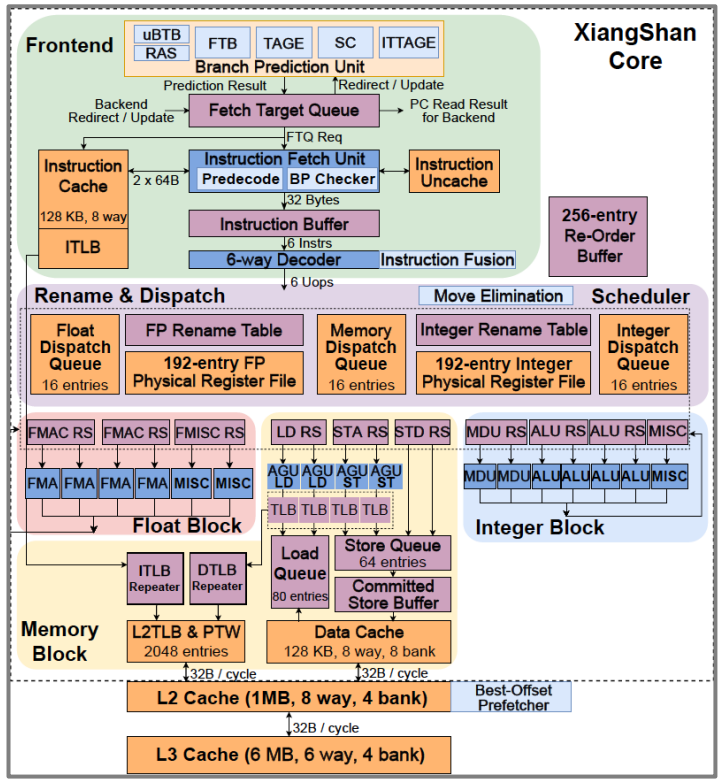
DUT microarchitecture complex design

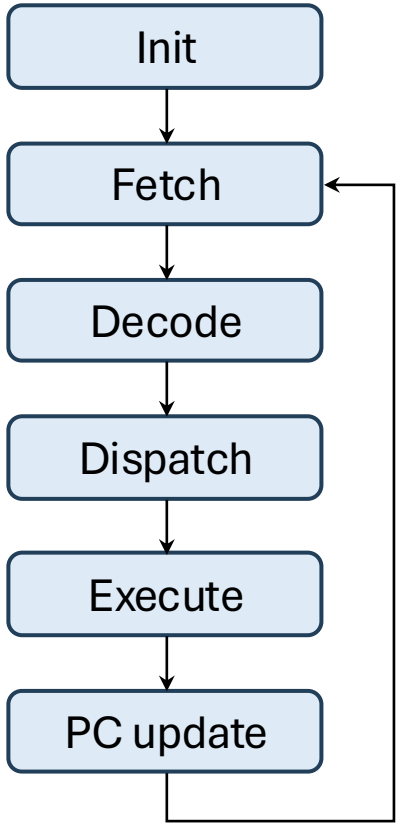


Challenge #1: Hardware REF Execution Efficiency

- Co-simulation **instruction throughput** is limited by the slower of DUT and REF
- DUT: *complex* microarchitecture with high instruction execution efficiency
- REF: kept *simple* for reliability — *how can its execution efficiency be improved?*

DUT microarchitecture
complex design





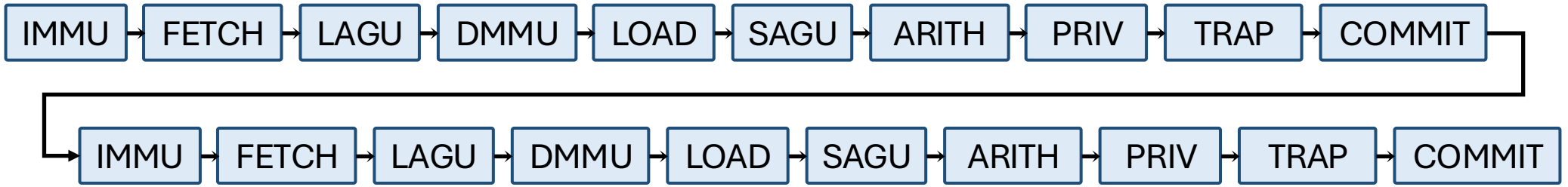
REF microarchitecture
simple design

Key Technique #1: Hardware Reference Model (SRef)

- Workflow: DUT commits *N* instructions; SRef executes *N* instructions and compares results

Key Technique #1: Hardware Reference Model (SRef)

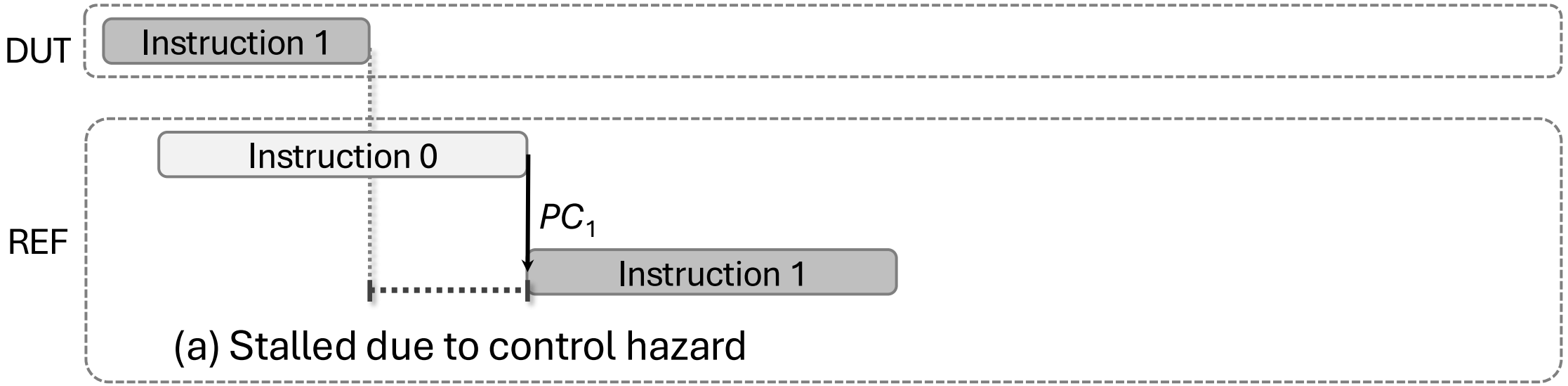
- Workflow: DUT commits *N* instructions; SRef executes *N* instructions and compares results
- Simple architecture: fully pipelined and non-blocking
  - Expectation: SRef completes within *N* × pipeline_length cycles
  - If this holds, SRef can co-simulate with processors of any IPC



Example: after the DUT commits 2 instructions,
SRef executes those 2 instructions through the pipeline stages

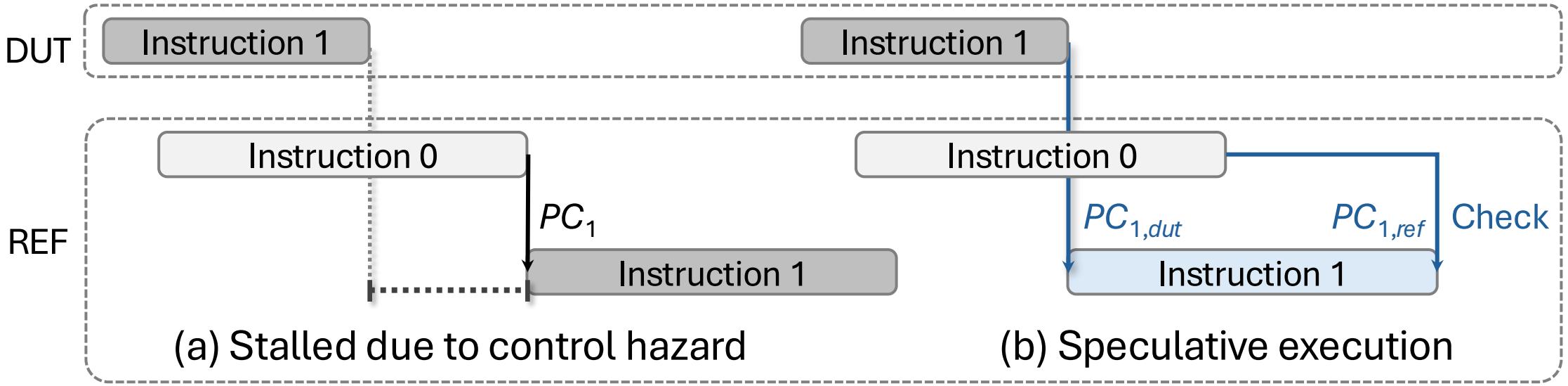
Key Technique #1: Main Pipeline of SRef (RCore)

- **Problem #1**: inter-instruction control/data hazards cause pipeline stalls



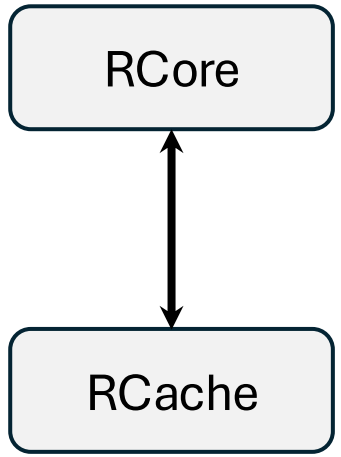
Key Technique #1: Main Pipeline of SRef (RCore)

- **Problem #1**: inter-instruction control/data hazards cause pipeline stalls
- **Approach**: speculative execution using DUT information
  - Use DUT results to guess first and verify later, eliminating dependencies



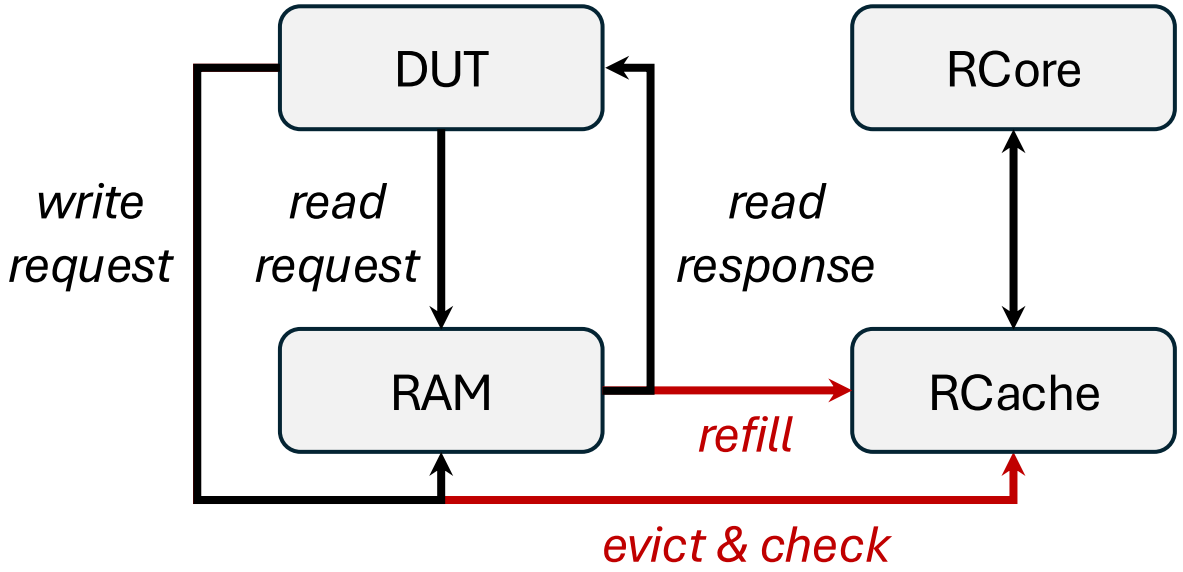
Key Technique #2: Cache System of SRef (RCache)

- **Problem #2**: structural hazards limit REF memory bandwidth



Key Technique #2: Cache System of SRef (RCache)

- **Problem #2**: structural hazards limit REF memory bandwidth
- **Approach**: use DUT memory-access results to avoid redundant REF memory access
  - Equivalent: RCache stores only REF/DUT RAM differences
  - DUT read: refill RCache by address
  - DUT write: check matching block, evict it, and report mismatch on error



Challenge #2: REF Correctness and Debuggability

- Existing REF models are usually *software ISA simulators*; concise, regular design helps ensure reliability and correctness
  - E.g., C/C++ provides high-level data structures for description

Category	Software description	
Instruction	instruction templates (insns)	`WRITE_RD(sext_xlen(RS1 + RS2))`
CSR	CSR base class csr_t	`csr_t::verify_permissions`
		`csr_t::read`
		`csr_t::write`
Constant	riscv-opcodes	`#define MATCH_ADD 0x33`
Operation	decode.h	`get_field/set_field`
Logging	stdio.h	`printf, display`

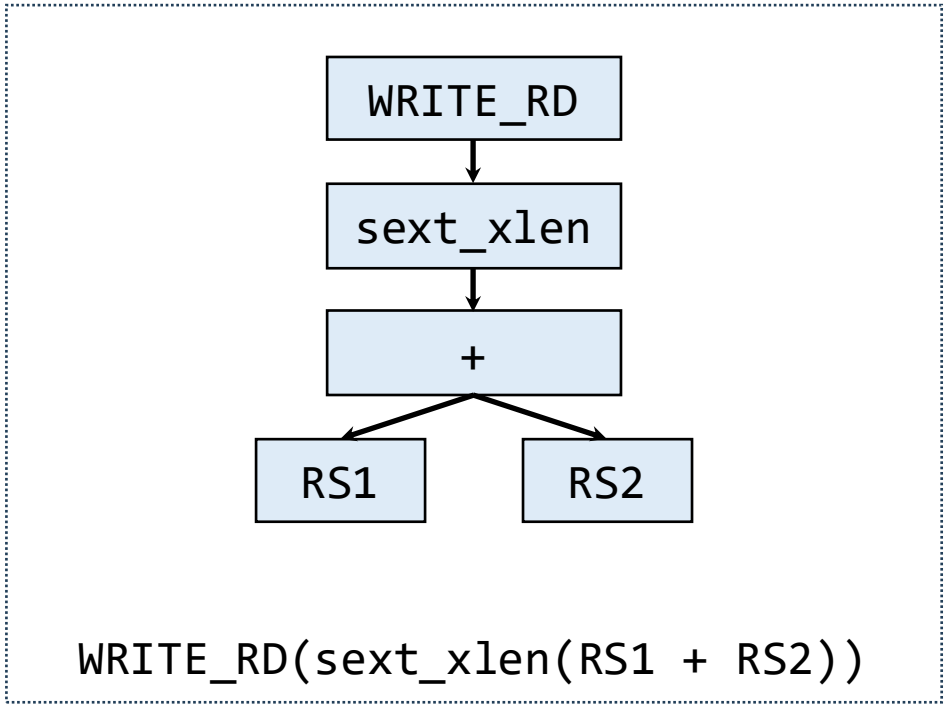
Challenge #2: REF Correctness and Debuggability

- Existing REF models are usually *software ISA simulators*; concise, regular design helps ensure reliability and correctness
  - E.g., C/C++ provides high-level data structures for description
- **Problem**: lack efficient hardware descriptions for synthesizable REF

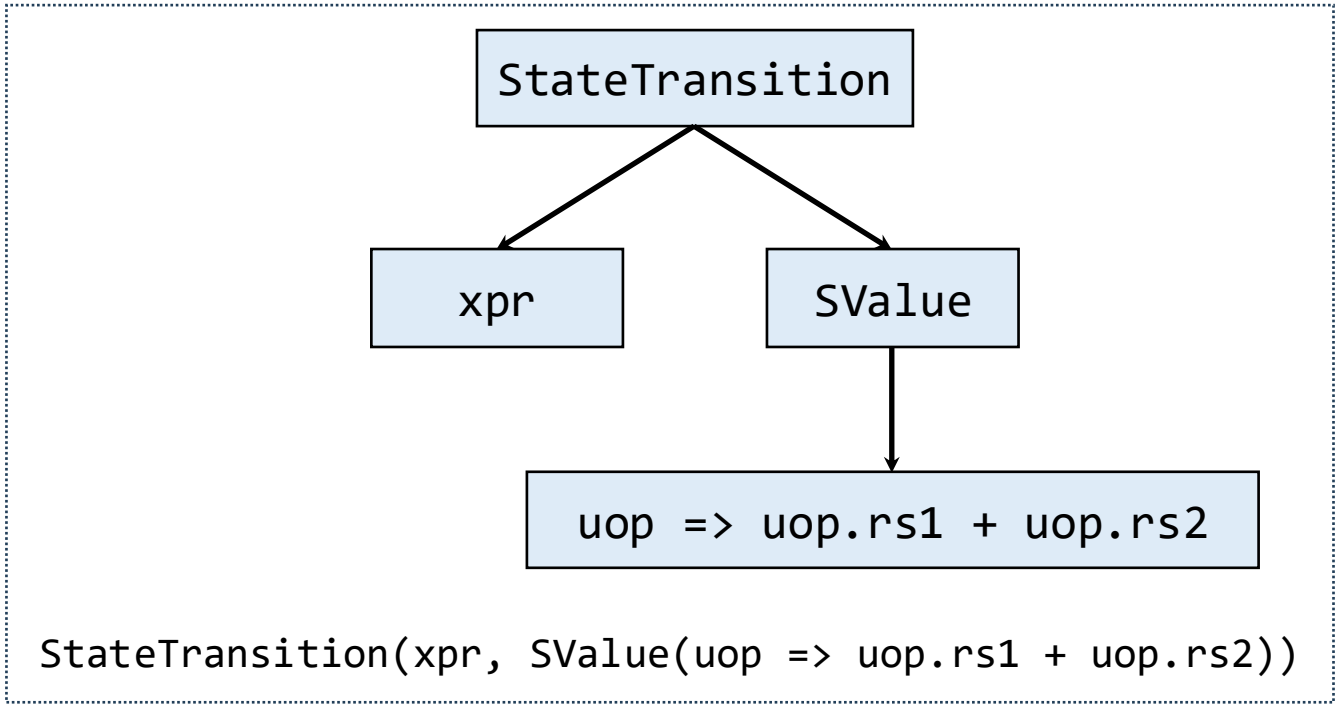
Category	Software description		Hardware
Instruction	instruction templates (insns)	`WRITE_RD(sext_xlen(RS1 + RS2))`	???
CSR	CSR base class csr_t	`csr_t::verify_permissions`	???
		`csr_t::read`	???
		`csr_t::write`	???
Constant	riscv-opcodes	`#define MATCH_ADD 0x33`	???
Operation	decode.h	`get_field/set_field`	???
Logging	stdio.h	`printf, display`	???

Key Technique #3: Semantic Code Migration

- **Idea**: automatically *migrate* key ISA semantics *from software REF*
  - **Example**: build an instruction-operation tree to migrate instruction functionality



(a) Spike



(b) Ours

Key Technique #4: Synthesizable Debugging Primitives

- Software verification frameworks support assertions, error logs, counters, and other basic debugging mechanisms
- **Approach**: REF converted into a standalone general-purpose CPU for debugging
  - Raise errors using hardware assertions, result checks, etc., and switch REF state
  - Verification mode: check correctness of DUT execution results
  - Debug mode: debug the failure context after an error

Debugging need	Typical software mechanism	This work
assertions	assert	hardware assertion extractor
error logs	display, printf	debug mode + UART
counters	counter	hardware counters
exception handling	signal, sig_handler	state controller and checker
	coredump	state recorder

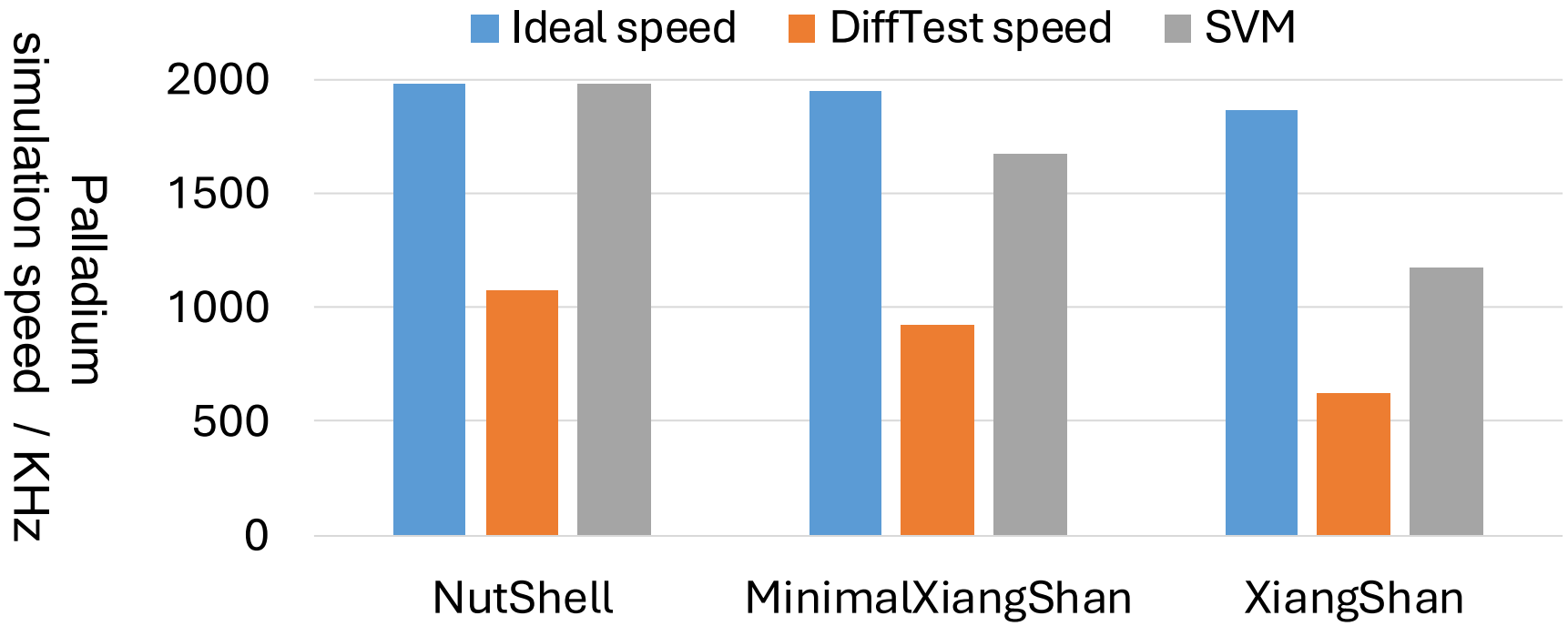
SVM Toolchain

- Implemented a synthesizable verification toolchain for RISC-V CPUs
  - https://github.com/poemonsense/SVM
  - https://github.com/OpenXiangShan/difftest/tree/svm

- Compatible with DiffTest user-side interfaces; takes over its simulation logic
  - Supports XiangShan, NutShell, and other configurations
  - Supports RISC-V I/M/A/C/B extensions
  - Supports M/S/U privilege modes and privileged operations
  - Automatically parses semantics from Spike source code

Evaluation: Approaching native simulation speeds

- Apply SVM to co-simulation verification of XiangShan and NutShell
  - FPGA: NutShell with 512KB RCache reaches 60MHz, 10X faster than DiffTest
  - Palladium: NutShell reaches 1.9MHz, matching native speed and 90% faster than DiffTest



Thanks!


ICT
中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES