SYNOPSYS® 新思

Test Generation for RISC-V HPC Verification Challenges

Yujie Fan, Application Engineer

2025/7/18

Legal Disclosure

CONFIDENTIAL INFORMATION

The information contained in this presentation is the confidential and proprietary information of Synopsys. You are not permitted to disseminate or use any of the information provided to you without the prior written consent of an authorized officer of Synopsys.

IMPORTANT NOTICE

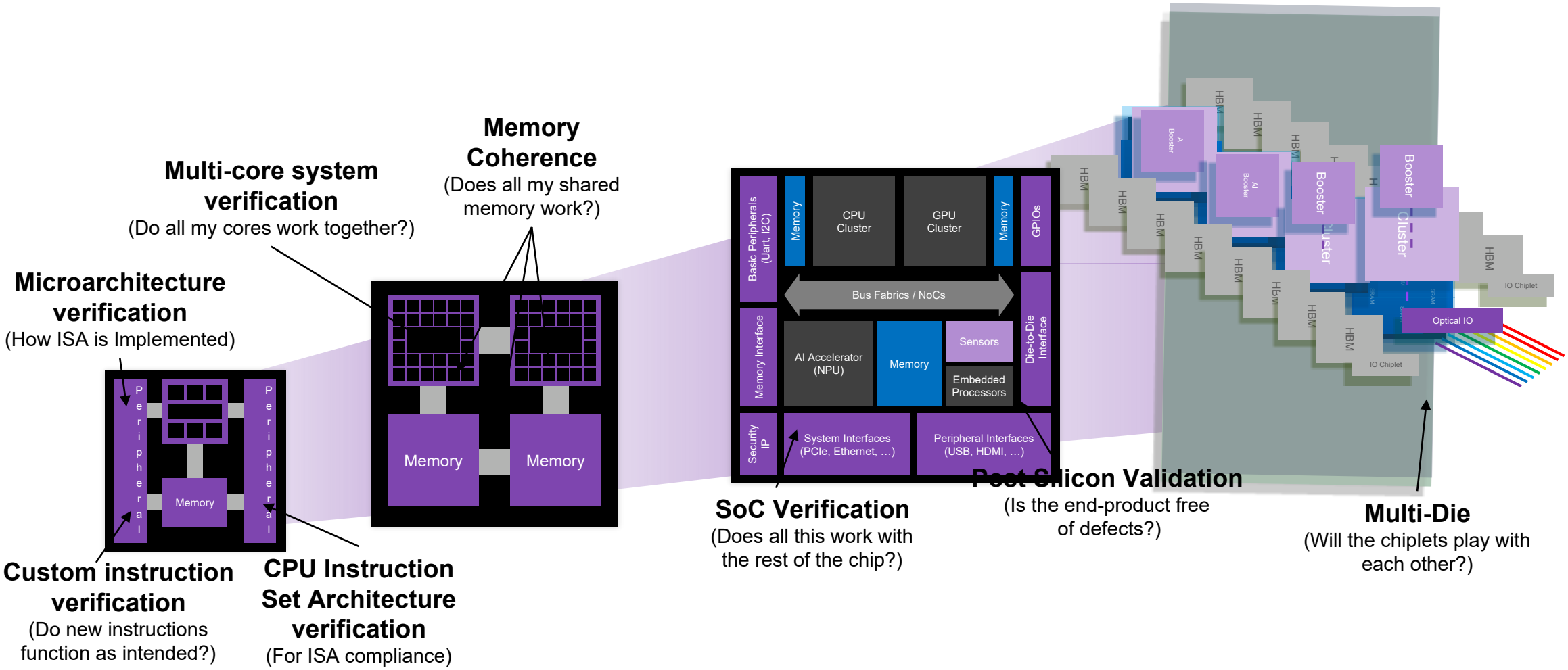
In the event information in this presentation reflects Synopsys' future plans, including but not limited to Synopsys' financial targets, expectations and objectives; strategies related to our products, technology and services; business and market outlook, business opportunities and strategies; and more, such information is based on current estimates, provided as of the date of this presentation and are subject to change. Synopsys undertakes no duty to, and does not intend to, update any forward-looking statement, whether as a result of new information, future events or otherwise, unless required by law.

Agenda

- Verification Challenges for HPC Implementations
- RISC-V Features for HPC Use-Cases
- STING Mechanisms for RISC-V HPC Design Verification
- Bugs Found

Verification Challenges for HPC Implementations

From CPU → Subsystems → Chiplets → Multi-die Systems → …

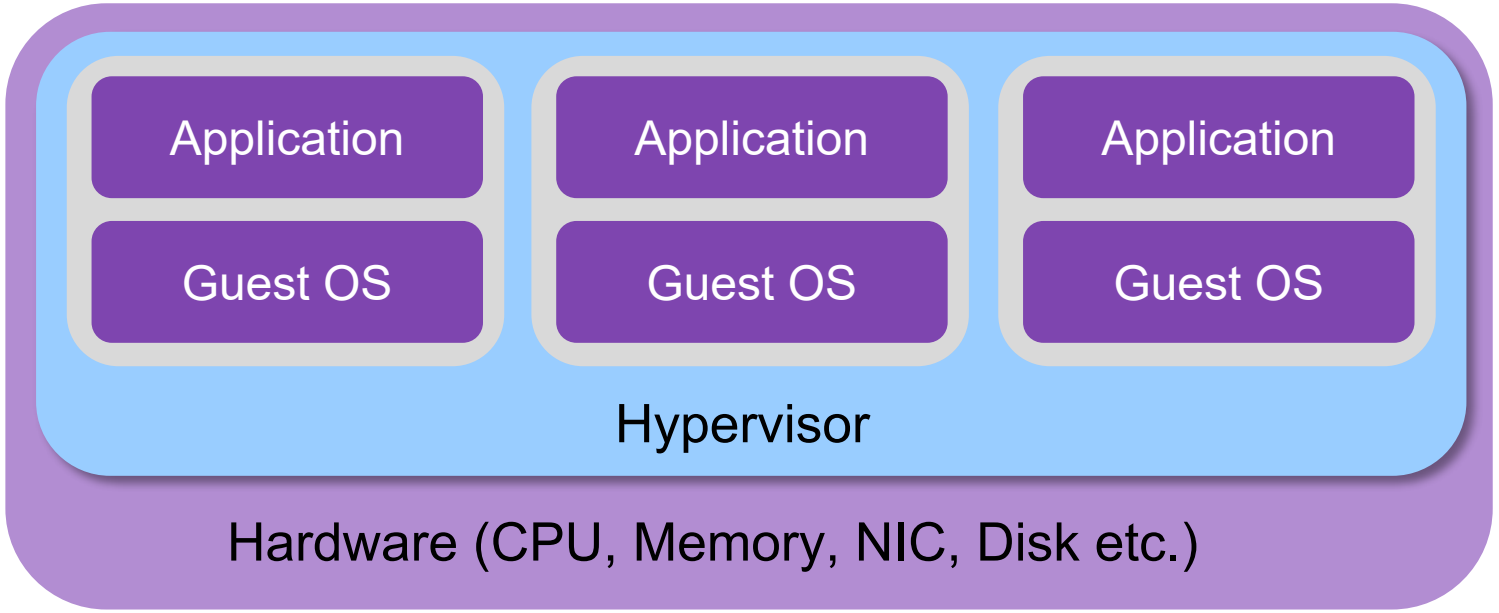


*"System Validation at ARM: Enabling our Partners to Build Better Systems", https://bit.ly/3WLCvhr, https://bit.ly/3CEMBd0

Complex Use Cases

CPU/System Virtualization

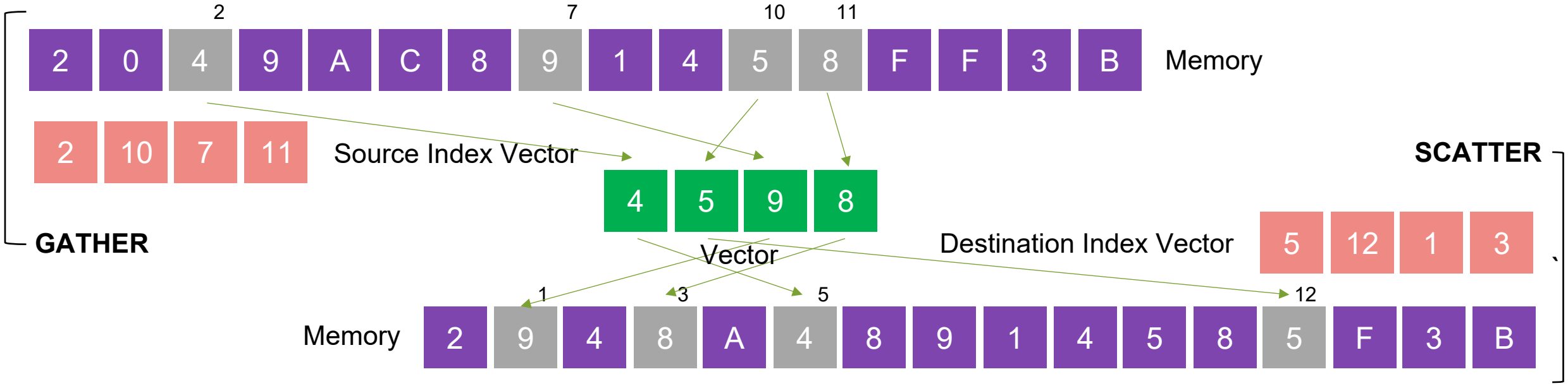
- Test generator should be capable of replicating scenarios from the OS/hypervisor software stack
- Use cases of System MMU, Interrupt and Data Virtualization and Security very important for advanced systems



Complex Use Cases (continued)

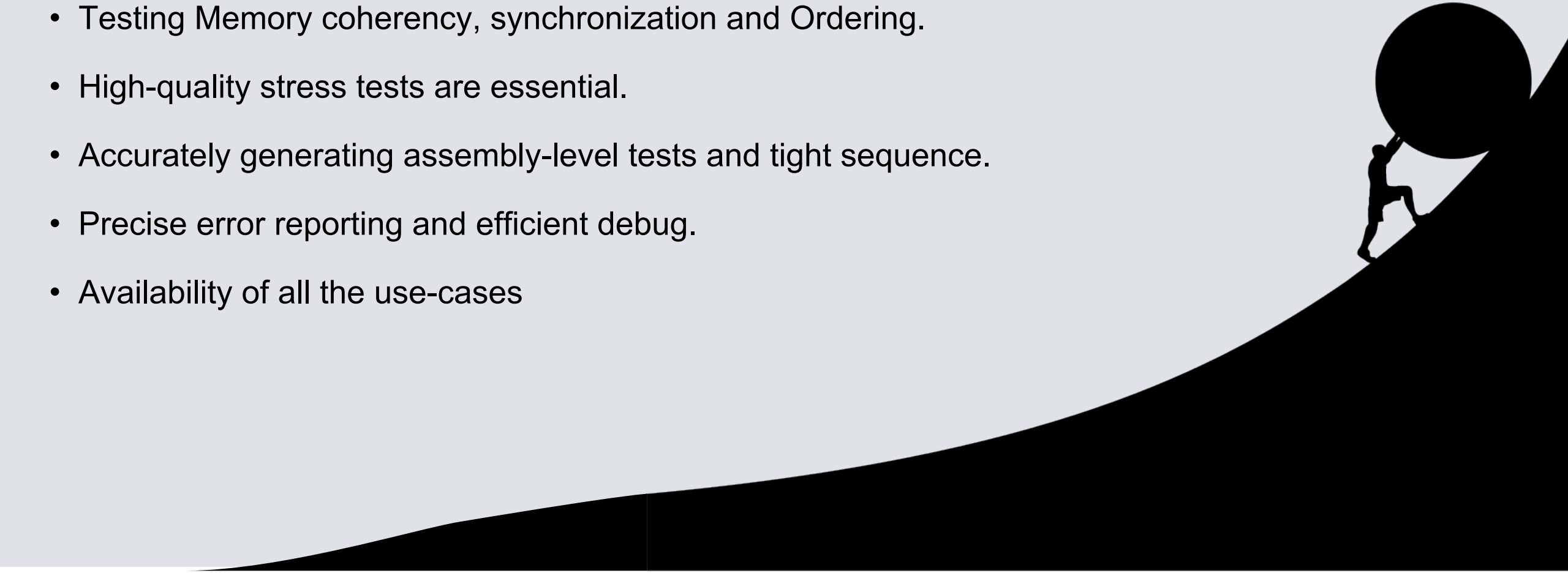
Advanced Extensions such as Vectors and Cache Management

- Advanced extensions such as Cache Management, MMU and Vectors need feature rich test generation
- Large number of real-world use cases/workloads must be run and checked
- High-quality and long running stress tests with interesting cross products are essential



Key Challenges in RISC-V HPC Verification

- Lack of Know-how and High-Quality Random tests (*esp. for Hyervisor, MMU, Vector*).
- Testing Memory coherency, synchronization and Ordering.
- High-quality stress tests are essential.
- Accurately generating assembly-level tests and tight sequence.
- Precise error reporting and efficient debug.
- Availability of all the use-cases



RISC-V Features for HPC Use-Cases

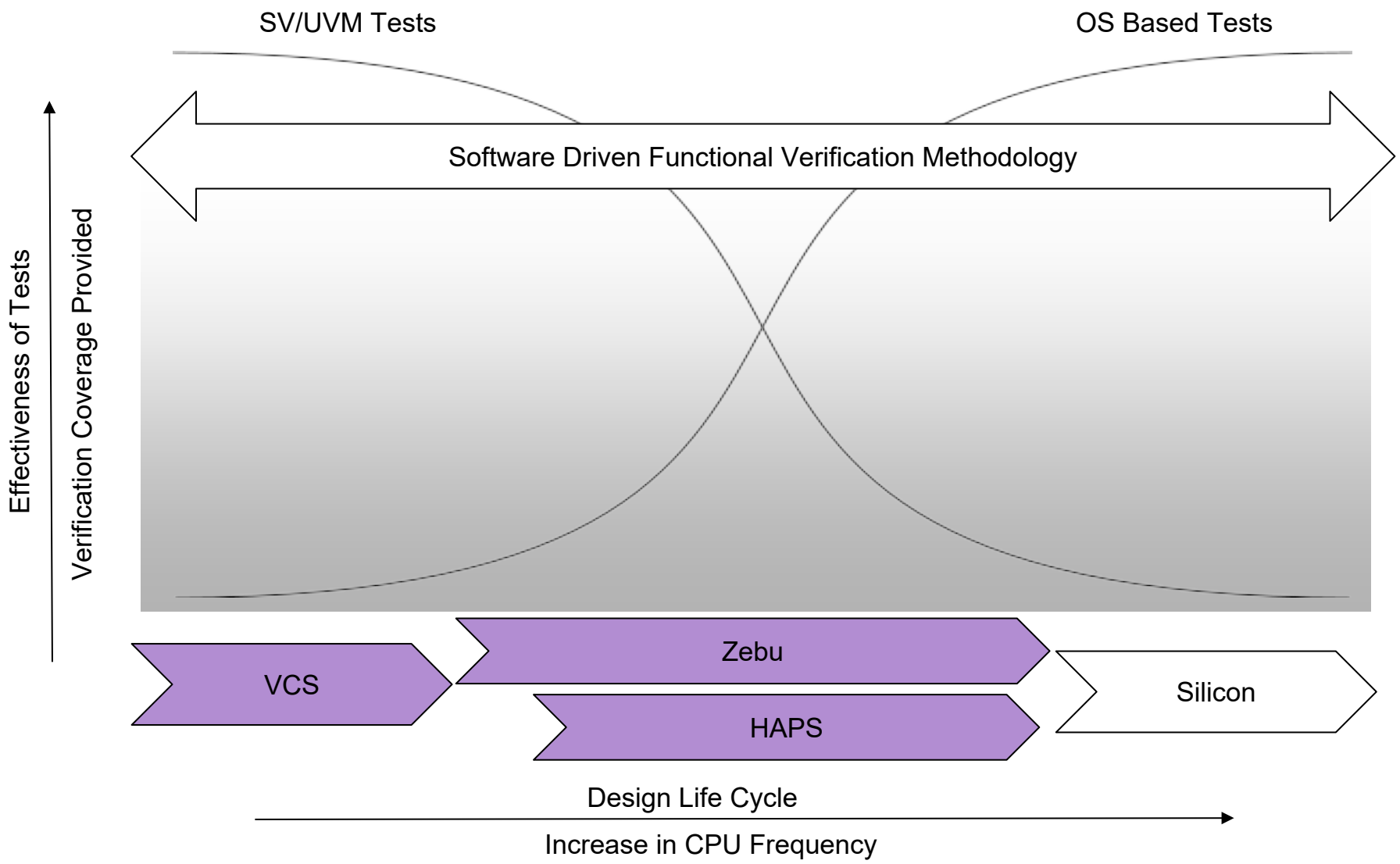
STING On RISC-V (continued)

★	Unprivileged Specification	YES
★	**Privileged Specification**	YES
★	RV32I Base Integer	YES
★	RV64I Base Integer	YES
★	RV128I Base Integer	NO
★	RV32E Base Integer	YES
★	M Extension (Multiply)	YES
★	**A Extension (Atomics)**	YES
★	F Extension (Single Precision FP)	YES
★	D Extension (Double Precision FP)	YES
★	Q Extension (Quad Precision FP)	NO
★	Zfh Extension (Half Precision FP)	YES
★	C Extension (Compressed)	YES
★	Zc* (New Compressed)	YES
★	Zicond Extension	YES
★	ZawrsExtension	YES
★	Zihintntl Extension	YES
★	Ssqosid Extension	YES
★	K Extension (Cryptography)	YES
★	Control Flow Integrity (CFI) Extension	WIP
★	B Extension (Bit manipulation)	YES
★	P Extension (Packed SIMD)	NO
★	Zam Extension (Misaligned Atomics)	WIP
★	Z*inx Extension (FP in Integer Registers)	NO
★	**V Extension (Vectors)**	YES
★	Vector Crypto	YES
★	Bfloat-16 (Zfbf*/Zvfbf*) Extensions	YES
★	N Extension (User Interrupts)	YES
★	Zihintpause Extension (Pause)	YES
★	Zicbo* Extension (Cache Management)	YES
★	Svinval Extension (Advanced MMU)	YES
★	Svnapot Extension (Advanced MMU)	YES
★	Svpbmt Extension (Advanced MMU)	YES
★	Sscofpmf Extension	YES
★	Smstateen Extension	YES
★	Indirect CSR Access Extension	YES
★	Machine Privilege Implementation	YES
★	**Hypervisor Privilege Implementation**	YES
★	Supervisor Privilege Implementation	YES
★	User Privilege Implementation	YES
★	CSR Coverage	YES
★	MMU Regimes	YES*
★	PMP (Physical Memory Protection)	YES
★	Svepmp (Advanced PMP)	YES
★	CLIC	YES
★	CLINT	YES
★	PLIC	YES
★	AIA (Advanced Interrupt Architecture)	YES
★	Debug Architecture	YES
★	Trace Architecture	NO

Test generation for extensions defined under RVA22 and RVA23 profiles supported now

STING Mechanisms for RISC-V HPC Design Verification

STING: Software Driven RISC-V DV Solution



Allow **consistent execution on all verification environments**

Test stimulus once developed can be reused easily across the design life cycle

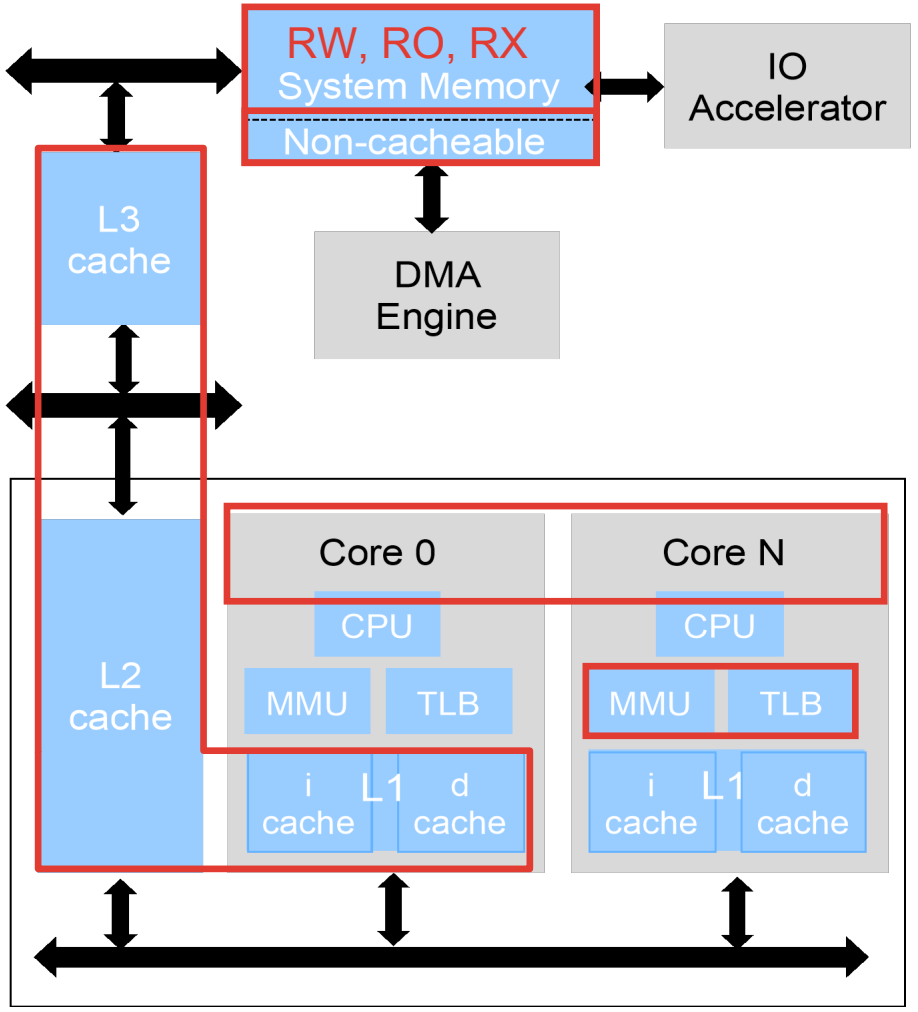
Failures hit on silicon can be easily migrated to an earlier stage for faster debug

Early enabling of software based stimulus increases the chances of hitting complex bugs early

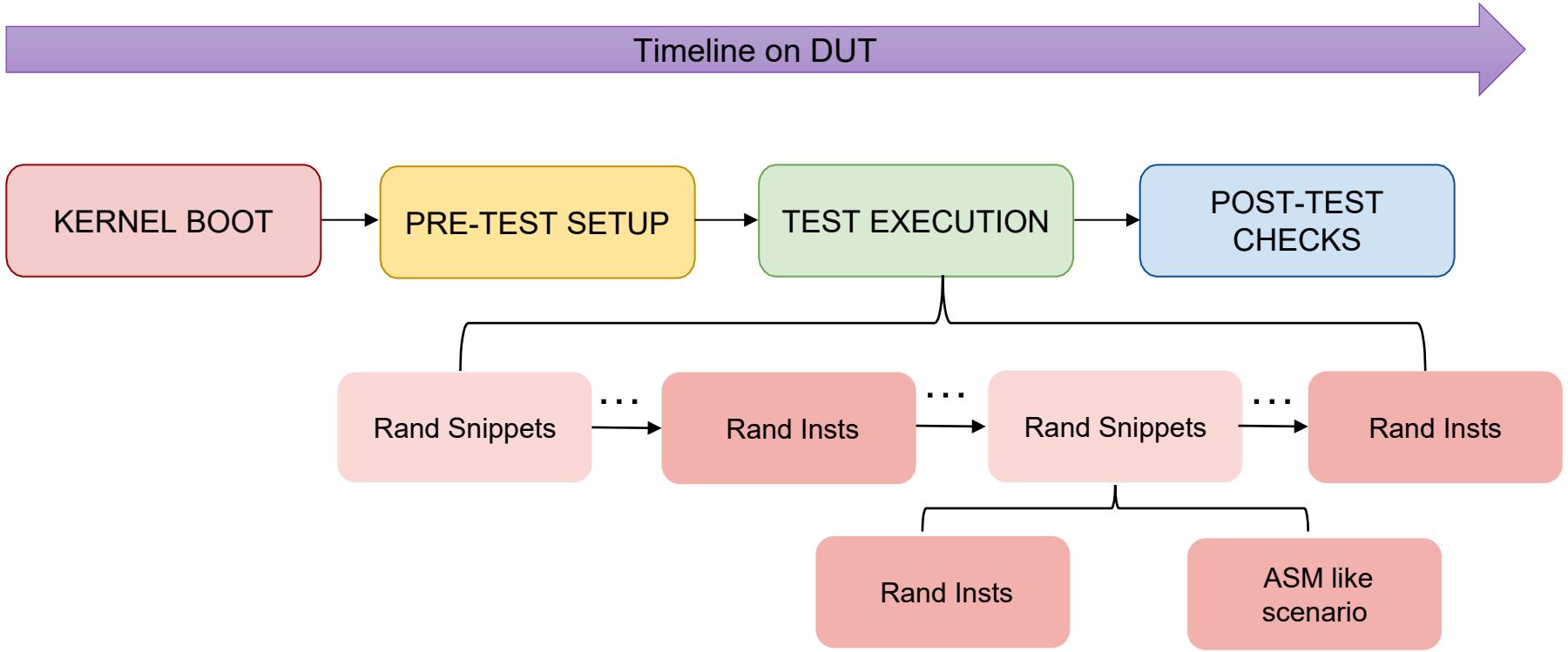
Save on duplicate efforts spent on verification and validation by acting as a bridge between the two methodologies

STING: Understands RISC-V, Memory and System

- Large number of cores
- ISA and extension sets
- Multi-level cache hierarchies
- Non-cacheable memories
- External device memories and system registers
- Different Privilege Mode: M/S/U/H
- Virtual memory systems (MMU) including
  - Page based coherency
  - Stage-1 & Stage-2 table-walks
  - MMU mode (SV32/39/48/57)
- PMP & PMA
- Test generator must be aware of **system memory** as well as the **interesting u-arch data flows** to target



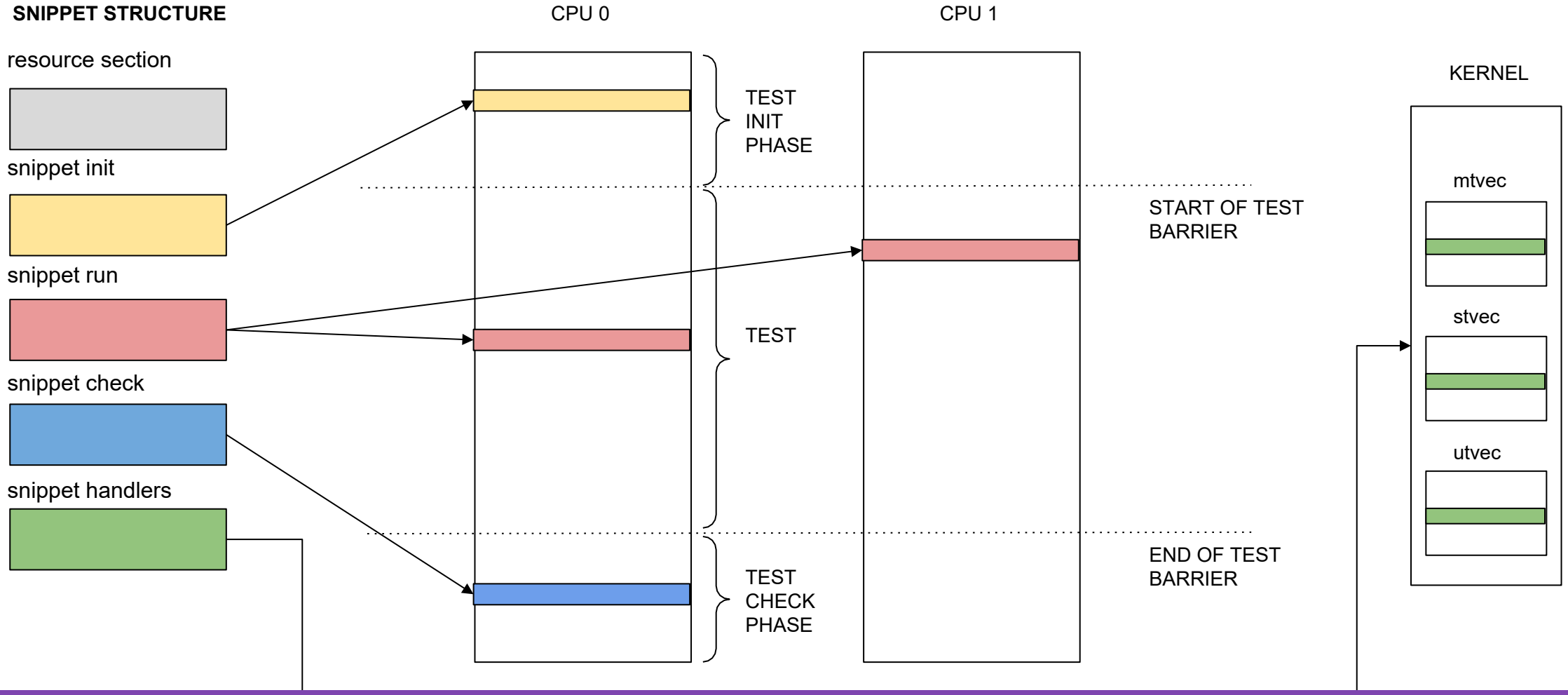
STING: Test Structure



- ★ Programmable sections in kernel/harness.
- ★ Custom interrupt and exception handlers.
- ★ Configurable and Scalable random tests based on rich tests library.
- ★ Easy to create stress and concurrency MP tests and scenarios.

Sting: Robust CPU Test Development Framework

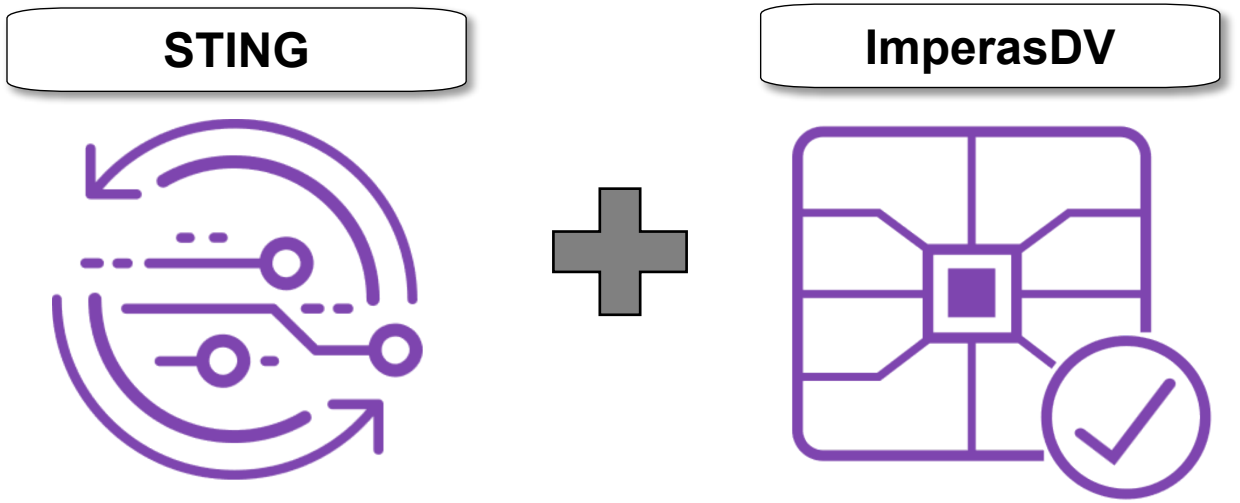
supporting construct tests rom low level assembly code to high level C++ tests



10K+ Out-of-Box Snippets are ready in test library .

STING can be used with ImperasDV

Test generation, checking, and coverage for RISC-V CPU developers



- STING provides the framework for **test generation**
- Instead of relying on self-checking, ImperasDV provides **lock-step compare**
- ImperasDV adds **asynchronous event verification** (impossible with self-checking)
- ImperasDV adds **RISC-V ISA Functional Coverage**

A Complete RISC-V CPU Verification Solution

Bugs Found

STING - Bugs Found

- ★ *"**Deadlock condition** existed when a TLB Miss for an older load/store instruction waits for its page-table-walk which cannot complete because newer stores have been issued and filled up certain miss-handling buffers in the load/store unit. This was uncovered by STING exercising streams of loads/stores with virtual memory enabled."*
- ★ *"Design had an **optimization issue** to convert a conditional branch over a single instruction into a predicated operation. There was a corner case bug in the implementation of this logic which used to cause **register corruption** when the 2 instructions (a "branch" and a "move") were separated by a pipeline flush in some scenarios."*
- ★ *"**Page table walk returning incorrect address translations** due to a bug in flushing of newer instructions when an older flush was taking place in the same cycle."*
- ★ *"**Stall condition** when a multiply instruction was in progress converted caught a pipeline issue in multiply unit that converted one type of multiply to another type of multiply."*
- ★ *"Back-to-back divides preceded by a long-latency memory bus read **caused the second divide to hang.**"*
- ★ *"STING **found a lot of nuances with floating-point rounding modes** and signaling/quiet NaNs. RISC-V has some quirks particularly with respect to the sNaN/qNaN handling."*
- ★ *"After tuning for our cache configuration, STING **did a good job stressing the cache controller.** Made sure a **lot of cache conflicts** were occurring. We support multiple outstanding misses so it found things like window conditions when one outstanding miss was getting filled and a request to that same line was being handled by the LSU. Windows around when data is available vs. directory state through the pipeline. Some windows that led to a cache line getting fetched and filled with "old" data while a write-back with new data was in progress."*
- ★ *"Few privileged **CSRs were getting sign-extended incorrectly** for some of the trap exceptions."*
- ★ *"**Unexpected execution** of instructions and trap exceptions **in the shadow of branch**"*
- ★ *"Issues with fence.i implementation resulting in **incorrect execution of self modifying code sequences**"*
- ★ *"Not found directly by STING: but much of the testing is built upon running STING tests while applying external stimulus of various forms. Debug, interrupts, etc. **Having sufficiently interesting code being executed by STING while that external stimulus was on-going help find a number of good issues.**"*
- ★ *"**PMP execute check wrong for the grain prior to a valid grain.** The problem occurs when attempting to execute from a PMP grain just prior to a configured PMP region. The defect lets the checks for the prior grain use the configuration of the next grain, which can cause exceptions to falsely fire, or falsely not fire."*
- ★ *"Corner-case hang requiring a combination of: - Completed but uncommitted loads or partial stores in the LSQ. - A dram slave request hits the LSU, matching one of the entries from #1 - An outstanding device request."*
- ★ *"**1 cycle window where wrong instruction text was serviced from the fetch buffer on a backwards branch,** relating to a specific case where an instruction cache line boundary is being crossed on the fetch buffer ingest side."*
- ★ *"**FSM in the DCACHE not cleaning the state correctly** for consecutive custom instructions that cancels each other."*
- ★ *"DCACHE not incrementing the "free entries" counter which leads to a counter leak that could **potentially block the core from doing any memory operation**"*
- ★ *"**Thread 0 starves Thread 1** (of the same core) when both threads are using the same resources (VPU, ALU) and one of the threads is doing a long latency operation (e.g. div)."*
- ★ *"**Livelock in the shared ICACHE** due to a bad LRU implementation."*
- ★ *"**LRAM clock gating triggered too early** and caused some writes to be lost."*
- ★ *"A pipeline optimization for multiplication operations results into a **deadlock condition**"*

SYNOPSYS® 新思

Thank you