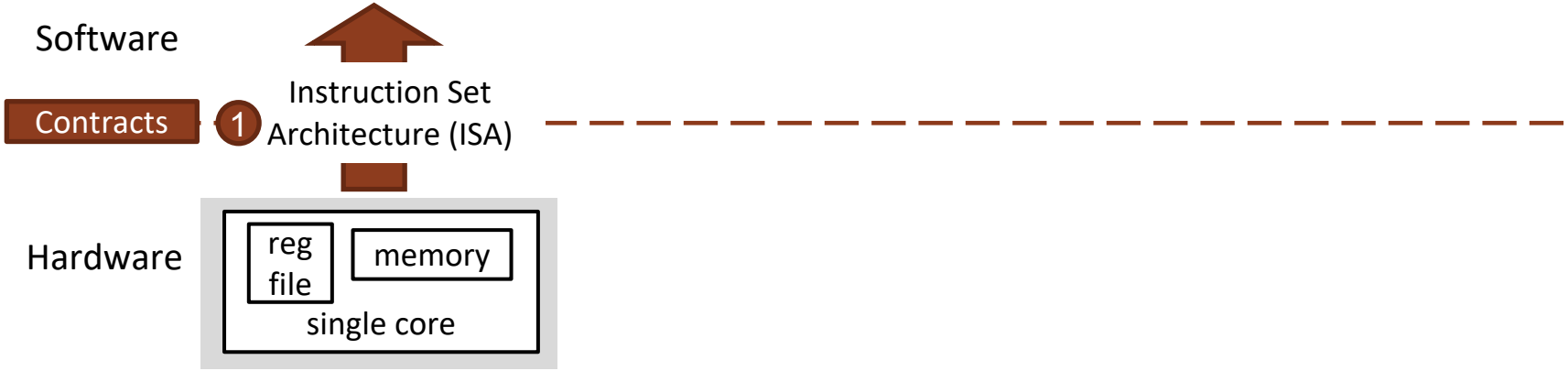
Scalable Assurance via Verifiable Hardware-Software Contracts

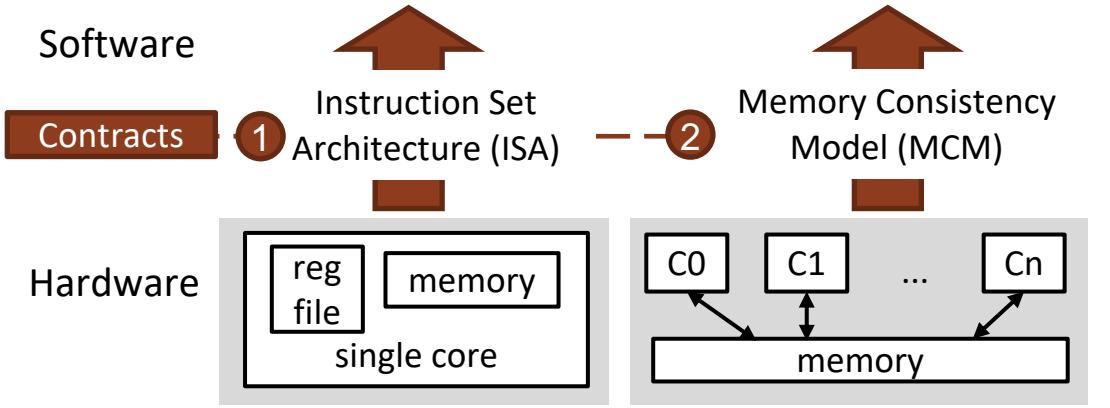
Yao Hsiao[1], Dominic P. Mulligan[2], Nikos Nikoleris[2],
Gustavo Petri[2], Caroline Trippel[1]

[1]Stanford University, [2]ARM Research

Hardware-software contracts expose hardware correctness and security guarantees to software



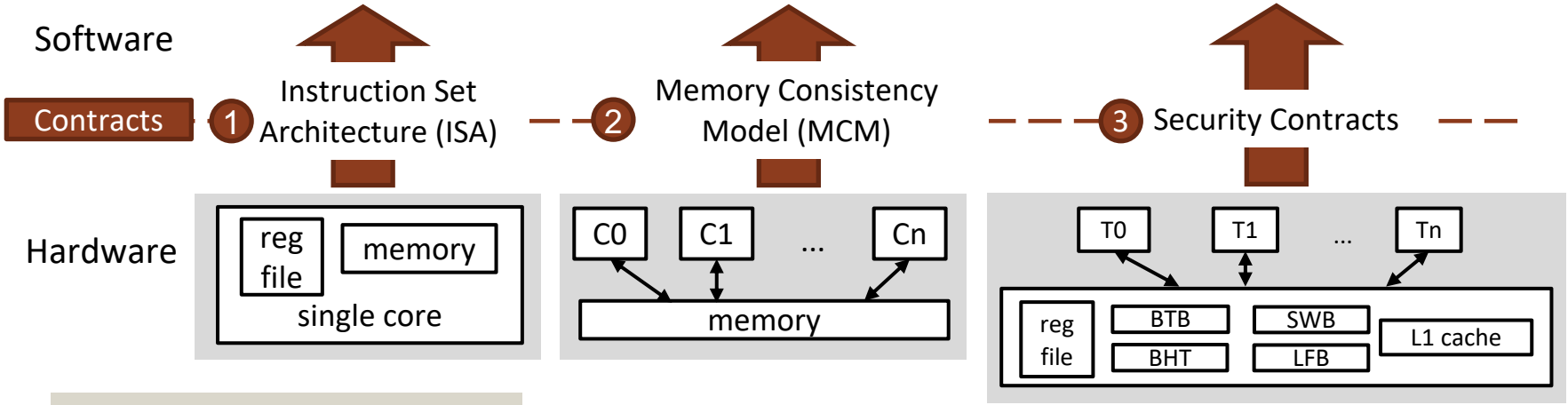
Hardware-software contracts expose hardware correctness and security guarantees to software



x86-tso: Forbids reordering writes
W x → W y in program order

C11→Power, PLDI'12
C11→x86, POPL'11
C11→ARMv7, Sewell+ '16
C11→ARMv8, Sewell+ '16
Java->Power ECOOP'15
Java→x86, ECOOP'15

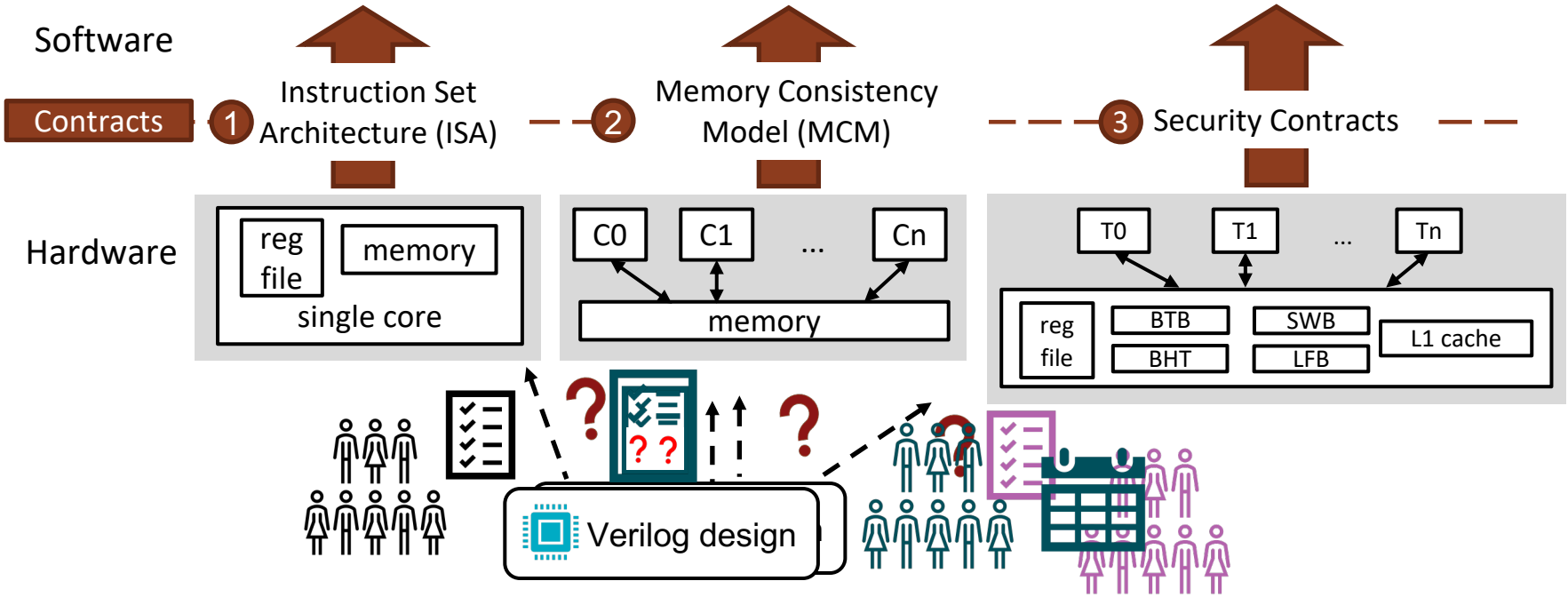
Hardware-software contracts expose hardware correctness and security guarantees to software



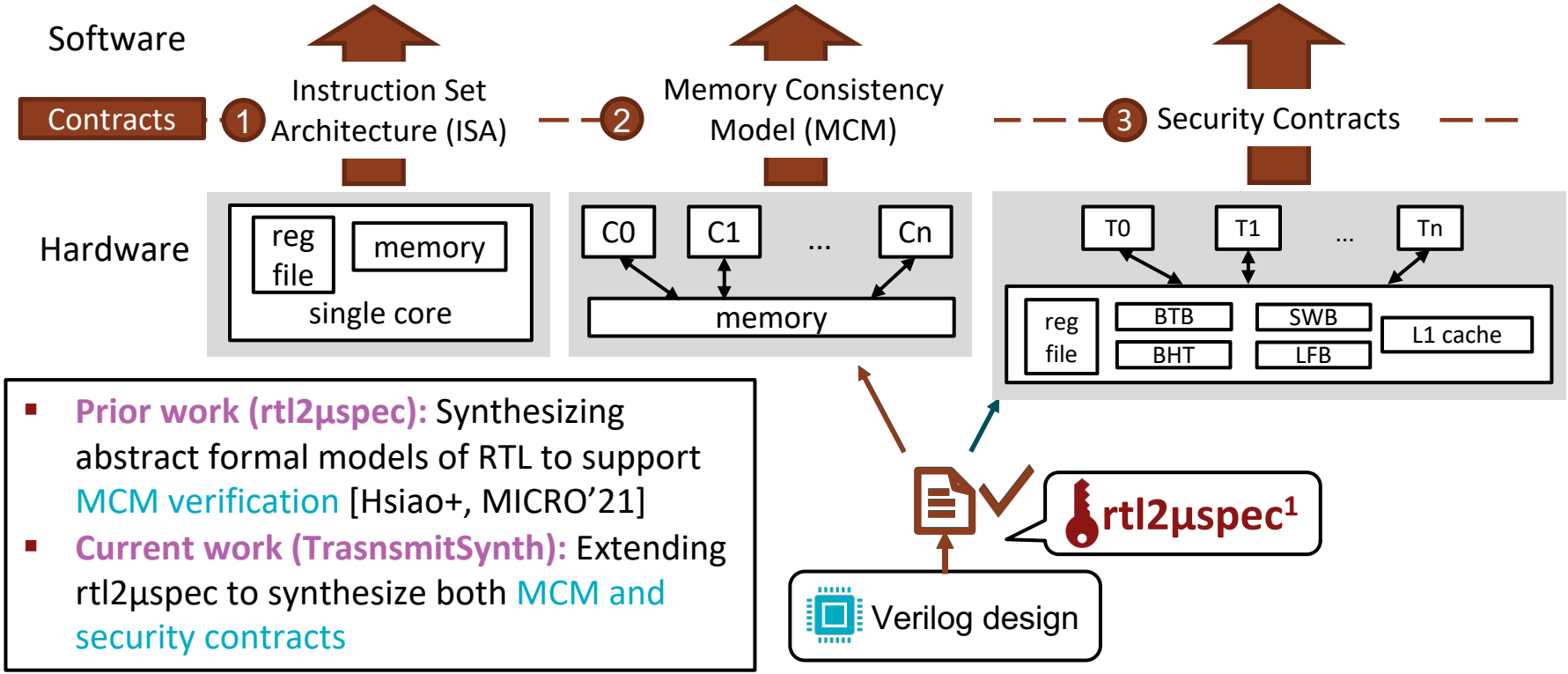
Spetector, S&P'20
Pitchfork, PLDI'20
Blade, POPL'21
Bisec/Haunted, NDSS'21
Clou, ISCA'22

Constant-time programming: Arithmetic instructions except division are constant time

Hardware-assurance challenge: A gap exists between hardware-software contracts and the RTL they abstract



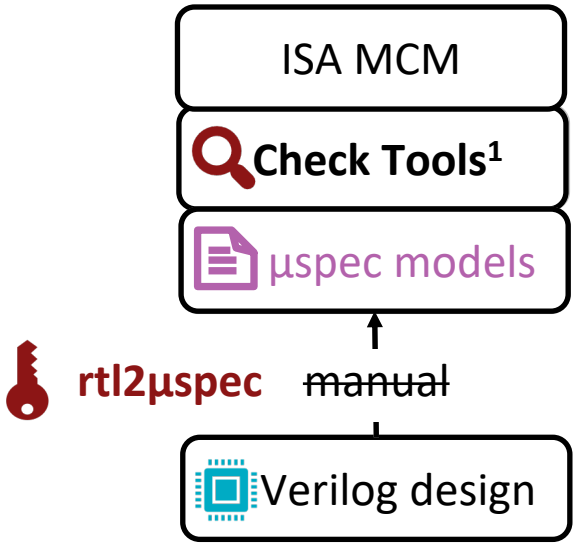
Can we **synthesize** formal hardware-software contracts from RTL?



- **Prior work (rtl2µspec):** Synthesizing abstract formal models of RTL to support MCM verification [Hsiao+, MICRO'21]
- **Current work (TrasnsmitSynth):** Extending rtl2µspec to synthesize both MCM and security contracts

[1]Hsiao et al. "Synthesizing Formal Models of Hardware from RTL for Efficient Verification of Memory Model Implementations." MICRO'21

Check Tools: Automated tools for conducting formal verification of hardware memory model implementations



Ordering rule for hardware

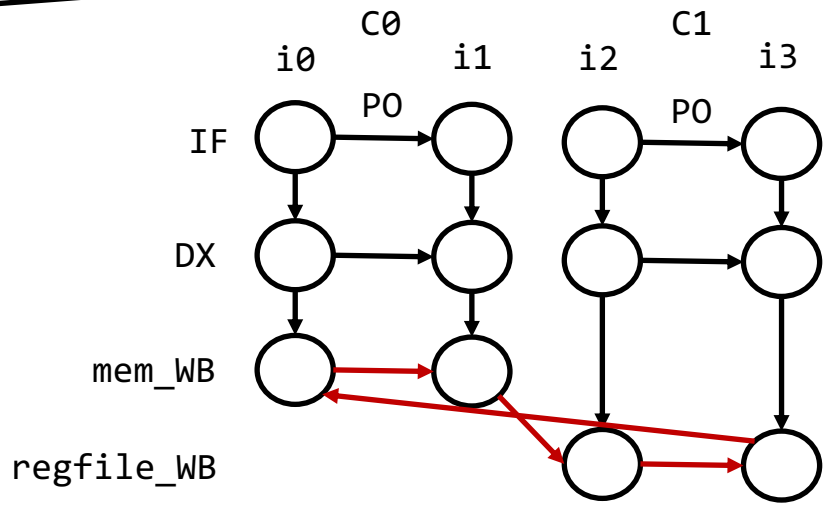
```
Axiom Ld_exe_path:
forall microops i0,
IsAnyRead i0 ⇒ AddEdges
[((i0, IF), (i0, DX))
((i0, DX), (i0, WB))
....]
Axiom “st_exe_path:
......
```

[1]http://check.cs.princeton.edu

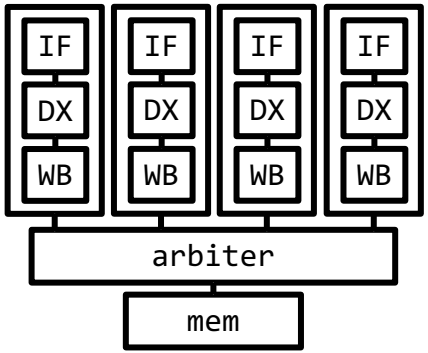
Microarchitectural happens-before (μhb) analysis

Core 0	Core 1
(i0) W[x] = 1;	(i2) R[y] = 1;
(i1) W[y] = 1;	(i3) R[x] = 0;

Forbidden on Sequential Consistency (SC)



RISC-V multi-V-scale[1]

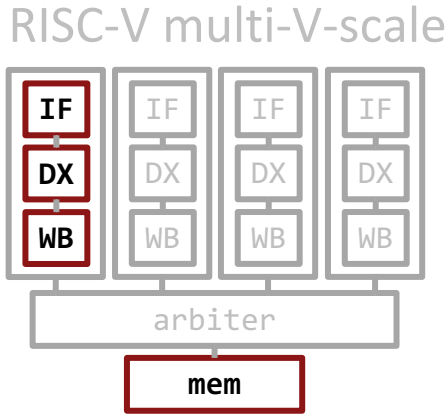


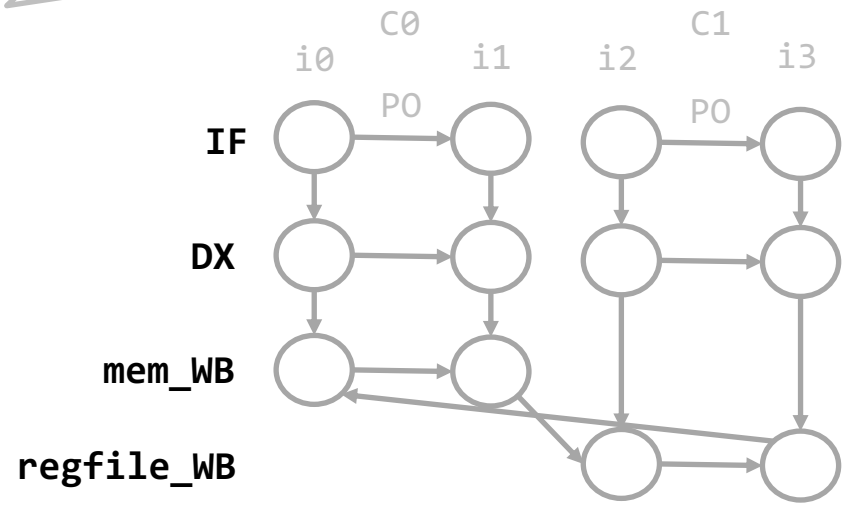
[1]Albert Magyar. 2016. A Verilog implementation of the RISC-V Z-scale microprocessor. https://github.com/ucb-bar/vscale.

μhb analysis: **Hardware locations**

```
Core 0          Core 1
(i0) W[x] = 1;  (i2) R[y] = 1;
(i1) W[y] = 1;  (i3) R[x] = 0;
```

Forbidden on SC



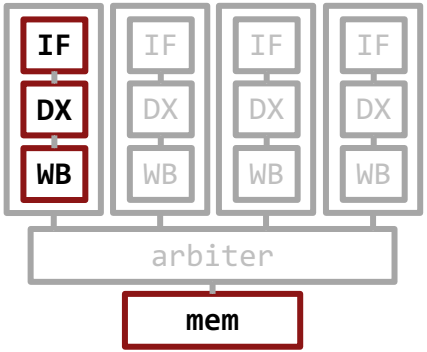


hardware state elements

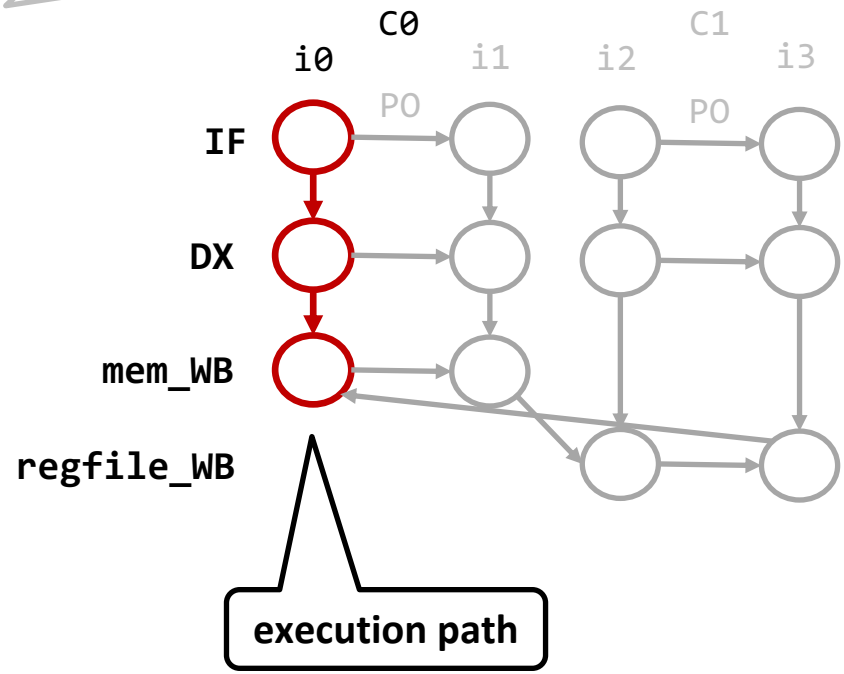
μhb analysis: **Instruction execution paths**

Core 0	Core 1
(i0) W[x] = 1;	(i2) R[y] = 1;
(i1) W[y] = 1;	(i3) R[x] = 0;

RISC-V multi-V-scale



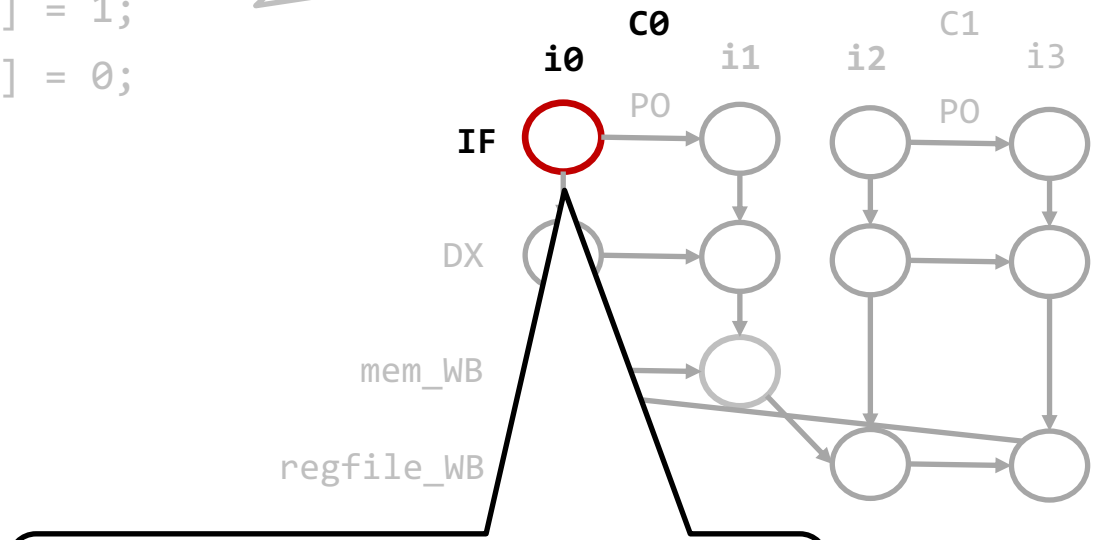
Forbidden on SC



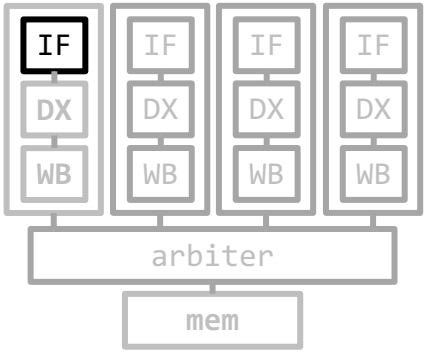
μhb analysis: **μhb nodes** a.k.a. hardware events

```
    Core 0                Core 1
(i0) W[x] = 1;    (i2) R[y] = 1;
(i1) W[y] = 1;    (i3) R[x] = 0;
```

Forbidden on SC



RISC-V multi-V-scale

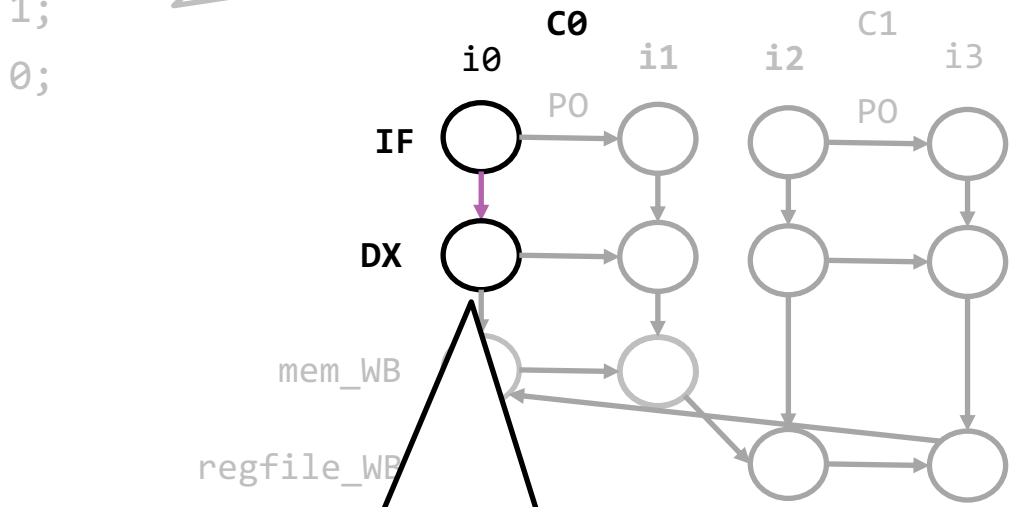


μhb nodes: microarchitectural events, <instruction, state element(s)> pair

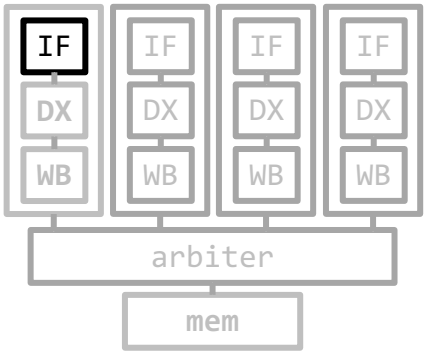
μhb analysis: **μhb edges** a.k.a. happens-before relations

```
    Core 0              Core 1
(i0) W[x] = 1;    (i2) R[y] = 1;
(i1) W[y] = 1;    (i3) R[x] = 0;
```

Forbidden on SC



RISC-V multi-V-scale

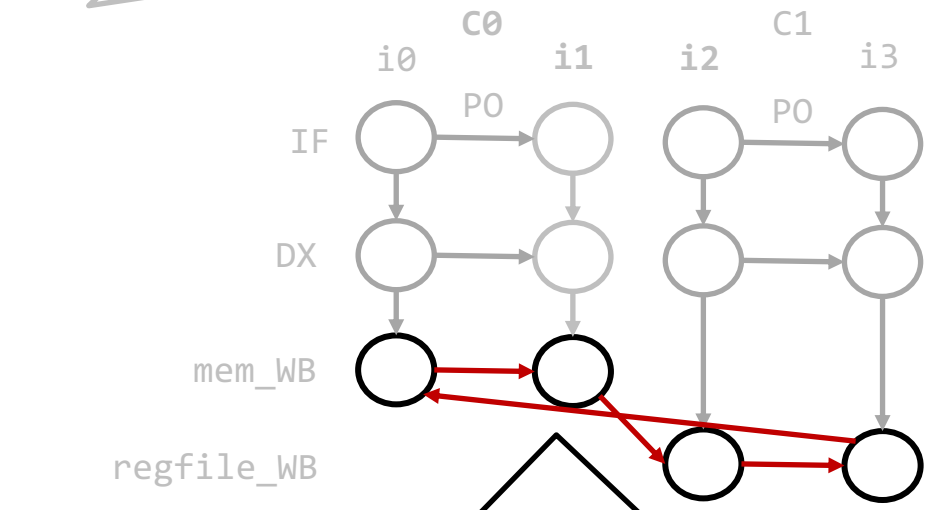


μhb edges: happens-before relationships between nodes

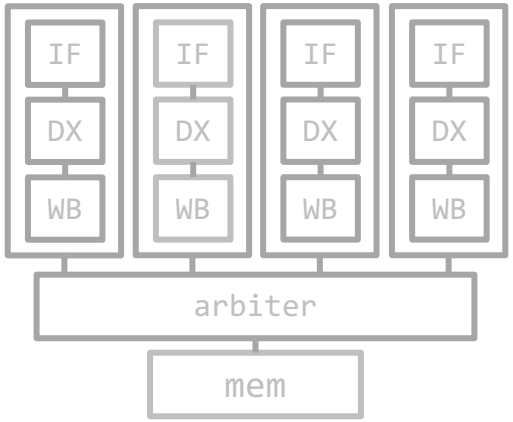
Reasoning about **execution observability** with μhb analysis

```
Core 0          Core 1
(i0) W[x] = 1;  (i2) R[y] = 1;
(i1) W[y] = 1;  (i3) R[x] = 0;
```

Forbidden on SC



RISC-V multi-V-scale



Cyclic μhb graph ⟷ Execution is not observable on the target hardware

Microarchitectural specifications a.k.a. μspec models

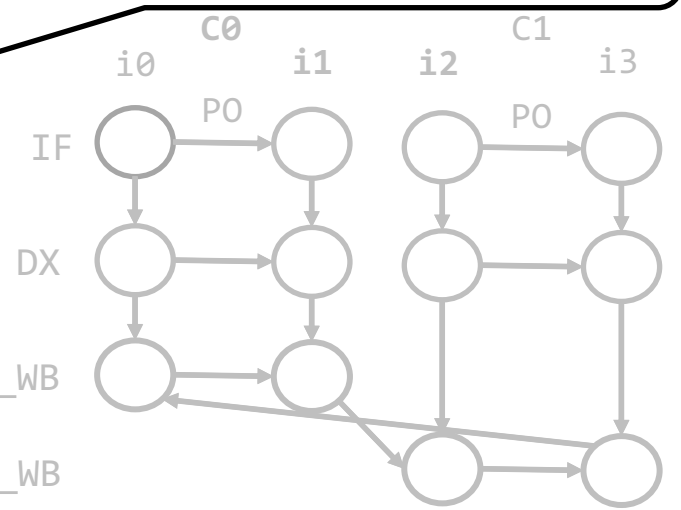
```
    Core 0          Core 1
(i0) W[x] = 1;   (i2) R[y] = 1;
(i1) W[y] = 1;   (i3) R[x] = 0;
```

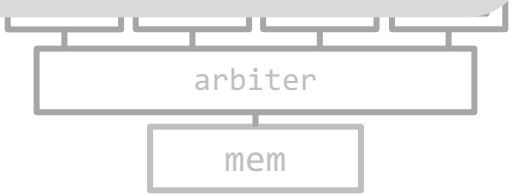
Check Tools: search all ways that a program may execute on hardware, according to μspec model.

Manually written ☹

```
Axi
forall microops i0,
IsAnyRead i0 ⇒ AddEdges [
((i0, IF), (i0, DX)),
((i0, DX), (i0, WB)),
....]
....
```

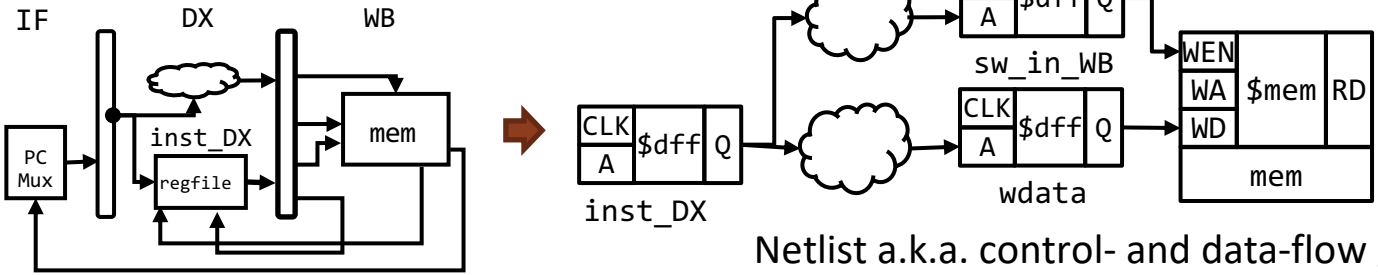
μspec models





μspec model of a microarchitecture specifies the space of all possible μhb graphs—i.e., the space of all possible hardware-specific program executions.

Automated synthesis of μspec models from RTL with rtl2μspec [Hsiao+, MICRO'21]



Netlist a.k.a. control- and data-flow graph

(System)Verilog design

SVA property embedding

JasperGold

Over-approximation of all axioms

Proven set of axioms

μspec

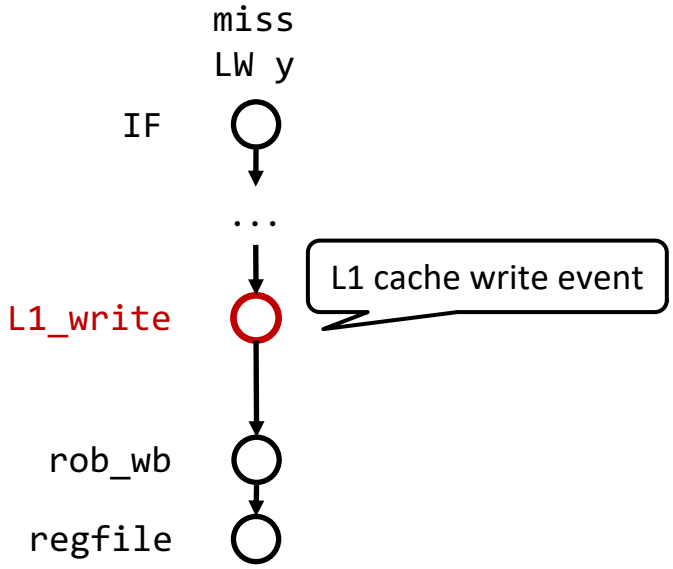
Open-source RISC-V multi-V-scale case study: 6.84 mins serial proof time w/120 SVA properties evaluated (> 780x performance improvement over prior work [Manerakr+ MICRO'17])

Hardware security verification with μspec models [Trippel+, MICRO'18]

① LW [base+secret]
② LW [y]

1. Load cache miss path (y != base+secret)

tag	vld	data
base+secret	1	...
y	1	....



A **transmit instruction** (or transmitter) is an instruction which modulates hardware resources differently as a function of its operands, results, or data-at-rest in a microarchitecture, resulting in **leakage.**

Hardware security verification with μspec models [Trippel+, MICRO'18]

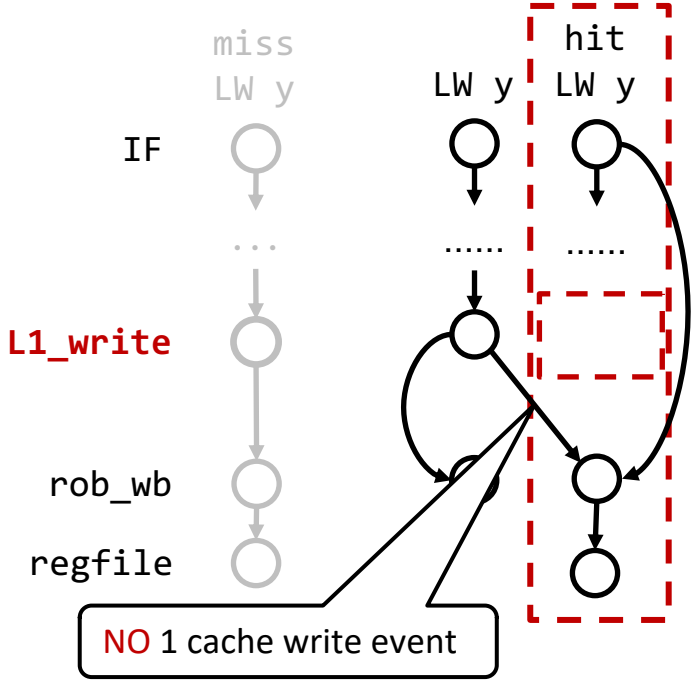
① LW [base+secret]
② LW [y]

1. Load cache miss path (y != base+secret)

tag	vld	data
base+secret	1	...
y	1	....

2. Load cache hit path (y == base+secret)

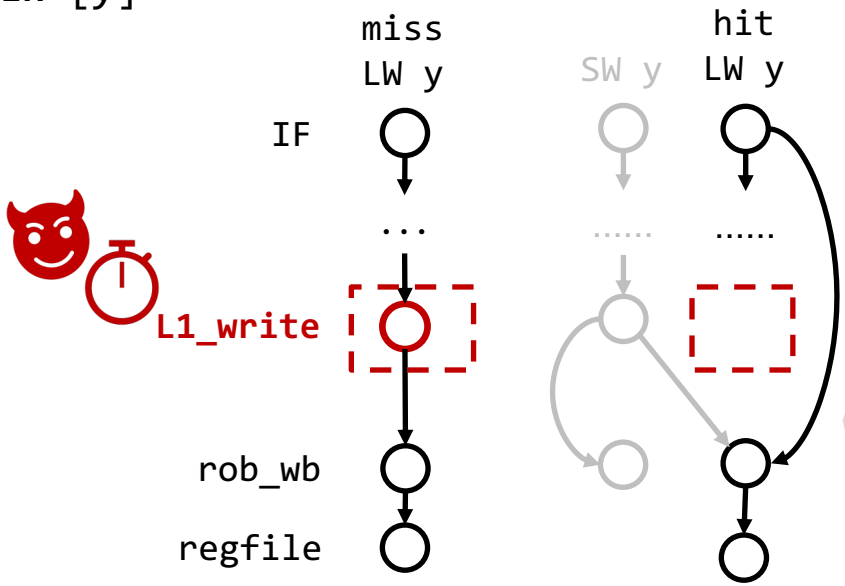
tag	vld	data
base+secret	1	...
-	-	-



A **transmit instruction** (or transmitter) is an instruction which modulates hardware resources differently as a function of its operands, results, or data-at-rest in a microarchitecture, resulting in **leakage.**

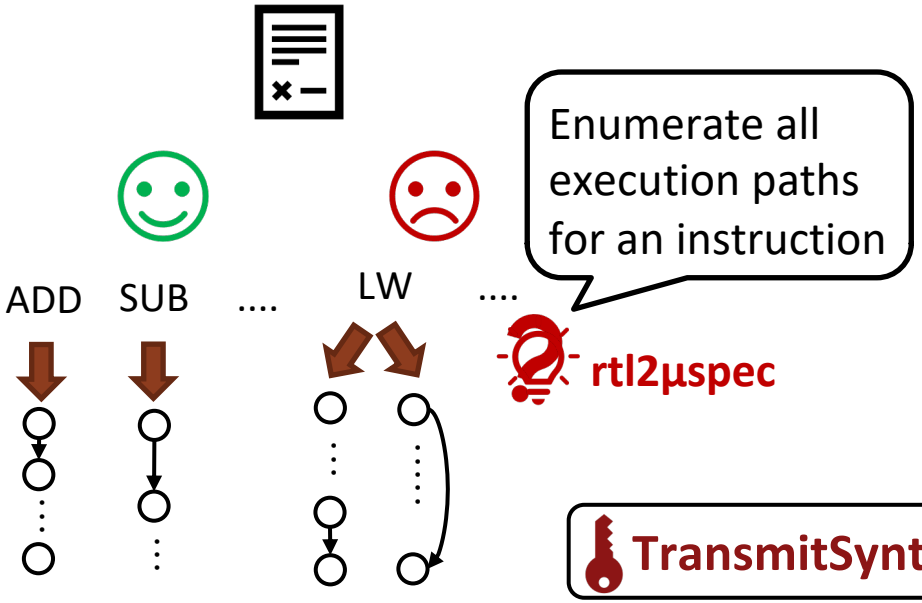
Key insight: If an instruction is a transmitter, it can instantiate more than one execution path in a μhb graph

① LW [base+secret]
② LW [y]




```
Axiom Ld_exe_path:
forall microops i0,
IsAnyRead i0 ⇒ AddEdges [
((i0, IF), (i0, DX)),
((i0, L1_write), (i0, rob_wb),
....]  // miss_path
⊕
AddEdges [
((i0, IF), (i0, DX)),
((i0, IF), (i0, rob_wb)),
....] // hit path
```

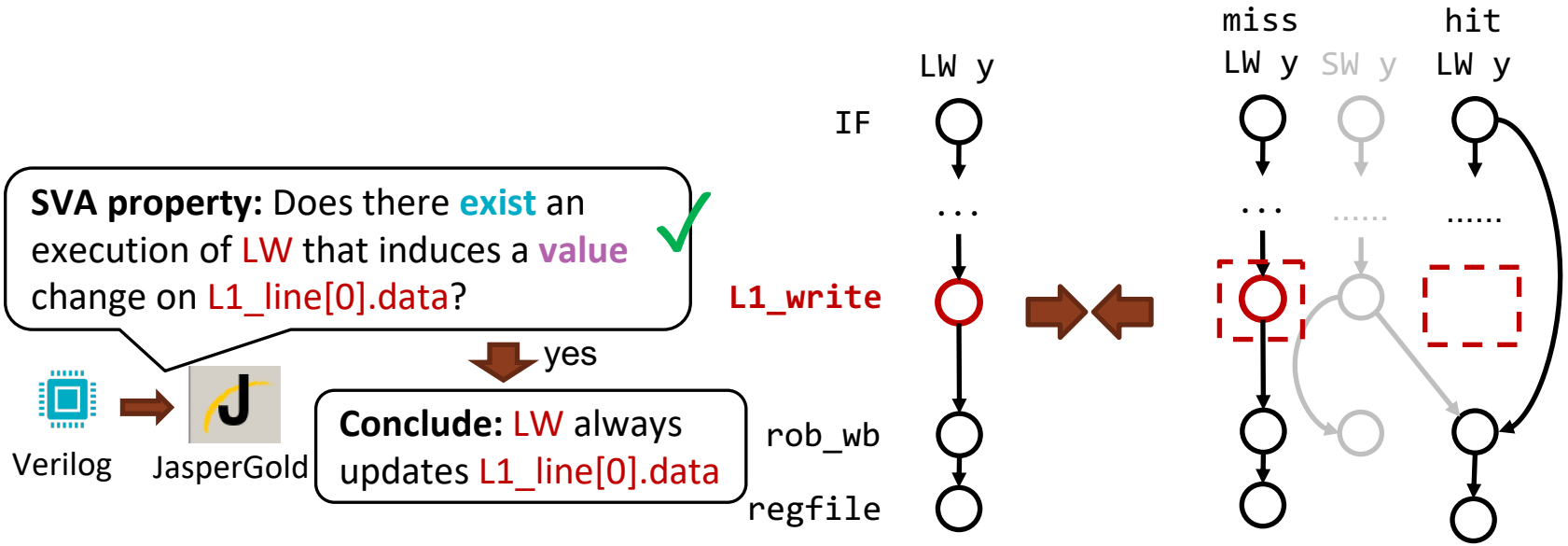
TransmitSynth: Automated synthesis of security contracts from RTL via identification of transmit instructions



Key limitation of rtl2µspec: the single execution path assumption.

- Each instruction type updates the same state elements exactly once every time it executes.
- rtl2µspec cannot recognize:
  - "Sometimes-updated" flops
  - "Repeatedly-updated" flops

Resolving the single execution path assumption: recognizing **sometimes-updated flops** with **write-enable analysis**



Resolving the single execution path assumption: recognizing **sometimes-updated flops** with **write-enable analysis**

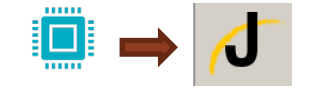
Challenge: updating a flip-flop does not imply a value change.

SVA property: Does there **exist** an execution of LW that induces a **value change** on L1_line[0].data?

Solution: conduct static analysis of RTL netlist to derive write enables for flip-flops.

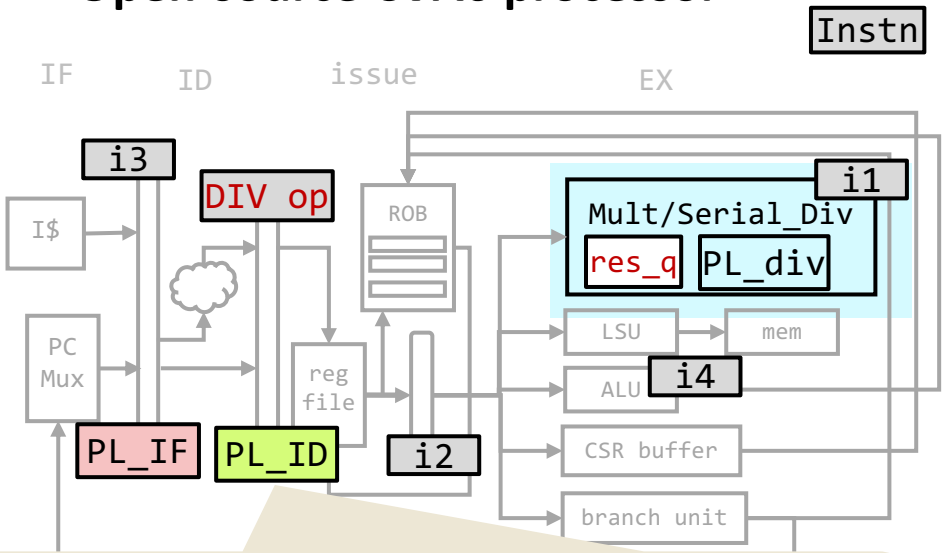
SVA property: Does the execution of LW **always/ever** cause L1_line[0].data's **write-enable to be asserted?**

Resolving the single execution path assumption: recognizing **sometimes-updated flops** with **write-enable analysis**



SVA property: Does the execution of **DIV** always/ever cause res_q's write enable to be asserted **when DIV is at PL_DIV ?**

Open-source CVA6 processor[1]



Performing Location: A design region which consists of an instruction identifier, optional instruction tracking logic, and a set of datapath registers. For an instruction to update a particular datapath register it must be residing in its associated performing location.

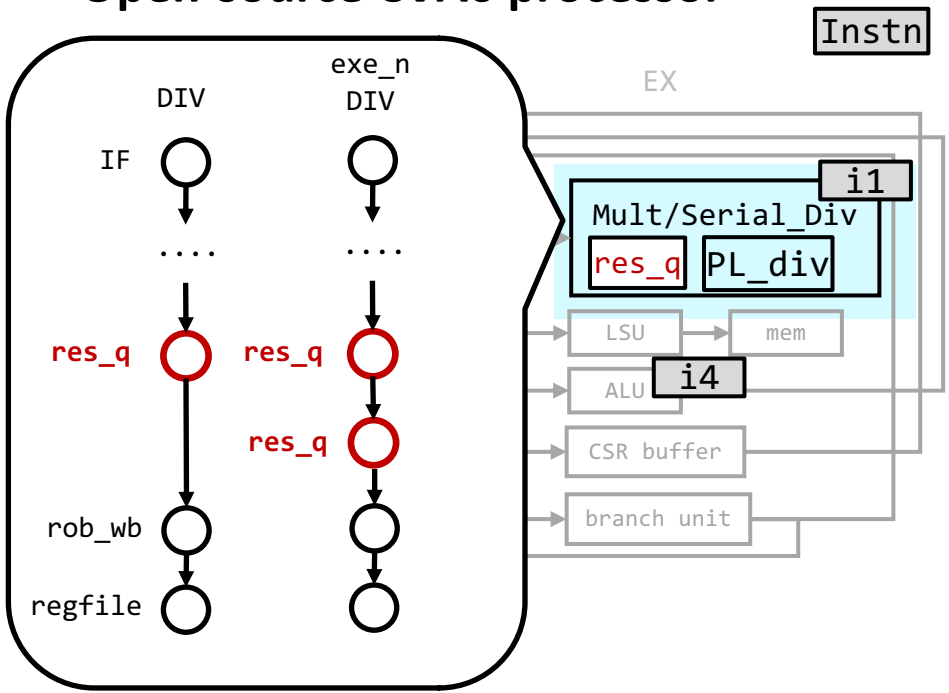
[1]Zaruba et al. https://github.com/openhwgroup/cva6

Resolving the single execution path assumption: recognizing **repeatedly-updated flops with performing location (PL)**

SVA property: Does the execution of **DIV** always/ever cause res_q's write enable to be asserted?

SVA property: Does the execution of **DIV** always/ever cause res_q's write enable to be asserted **when DIV is at PL_DIV ?**

Open-source CVA6 processor[1]



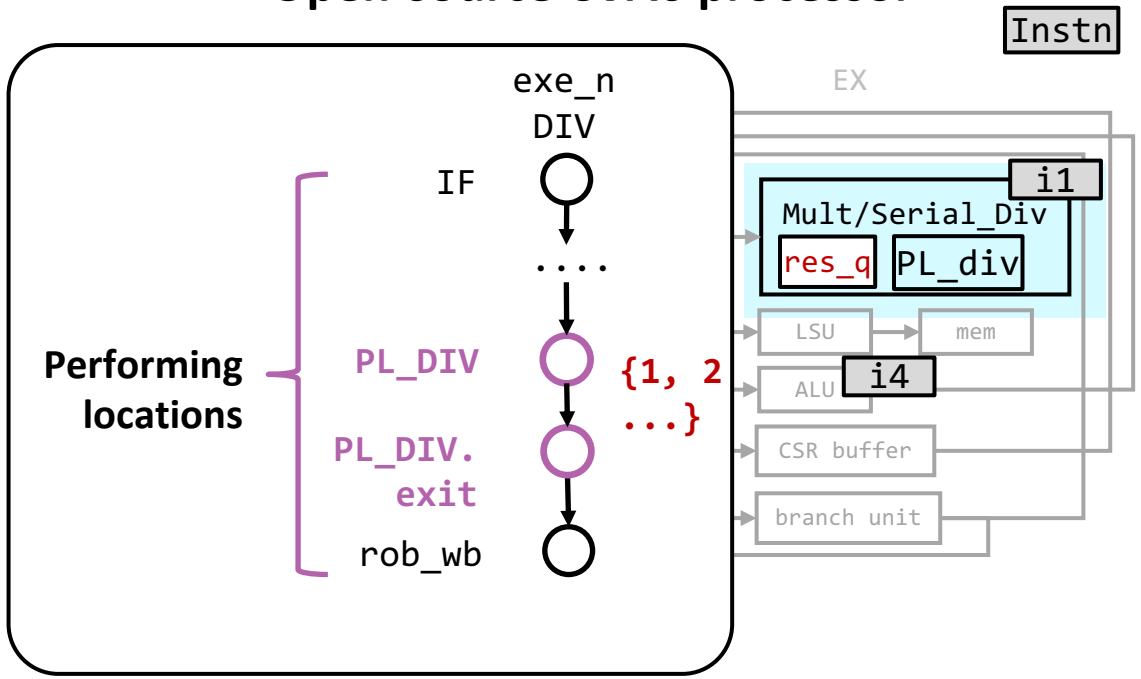
[1]Zaruba et al. https://github.com/openhwgroup/cva6

Resolving the single execution path assumption: recognizing **repeatedly-updated flops with staying properties**

Verilog → JasperGold

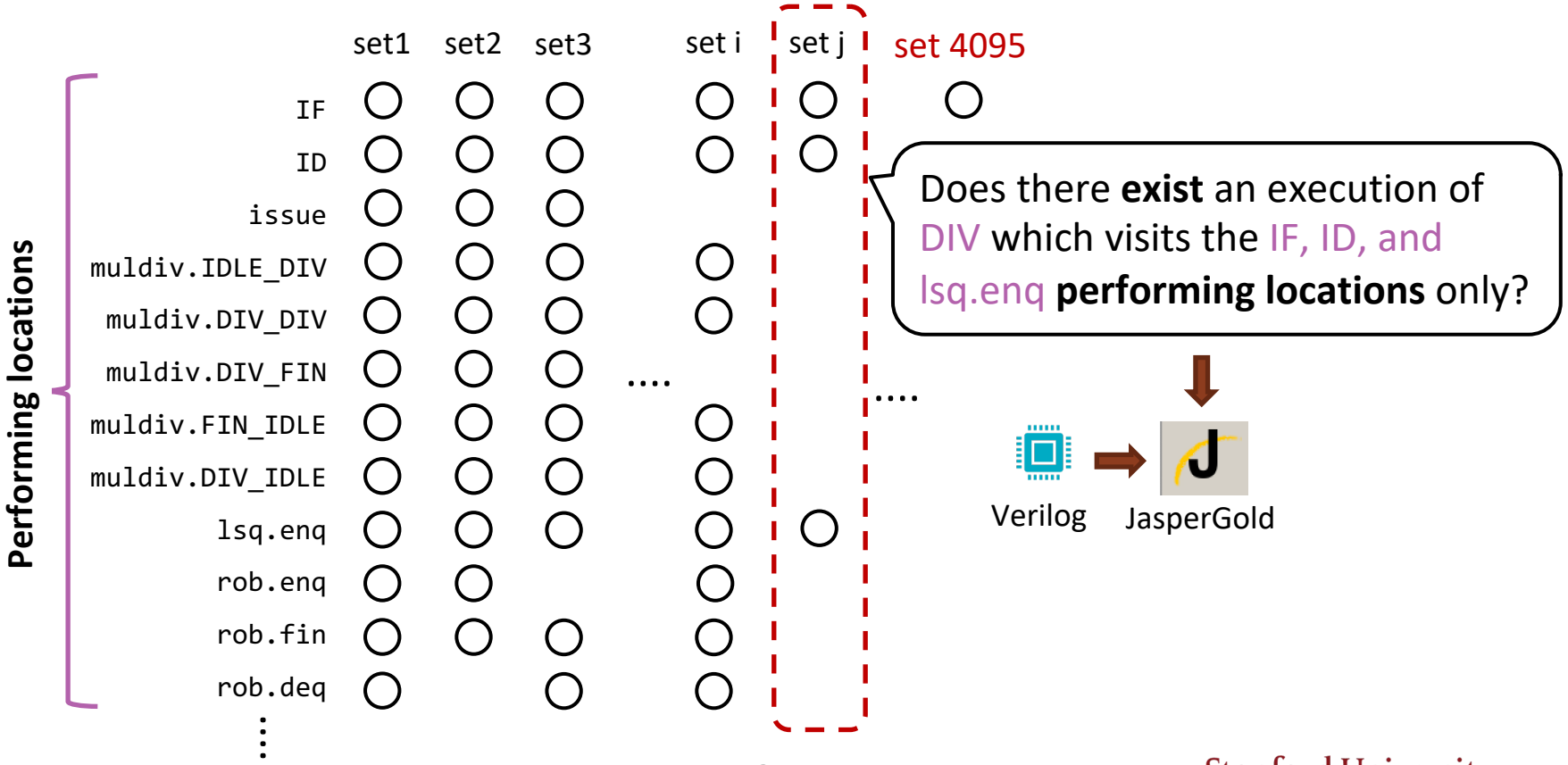
SVA property 2: Does the execution of **DIV** stay at **PL_DIV for 2 cycles? 3 cycles? .... n cycles?**

Open-source CVA6 processor[1]

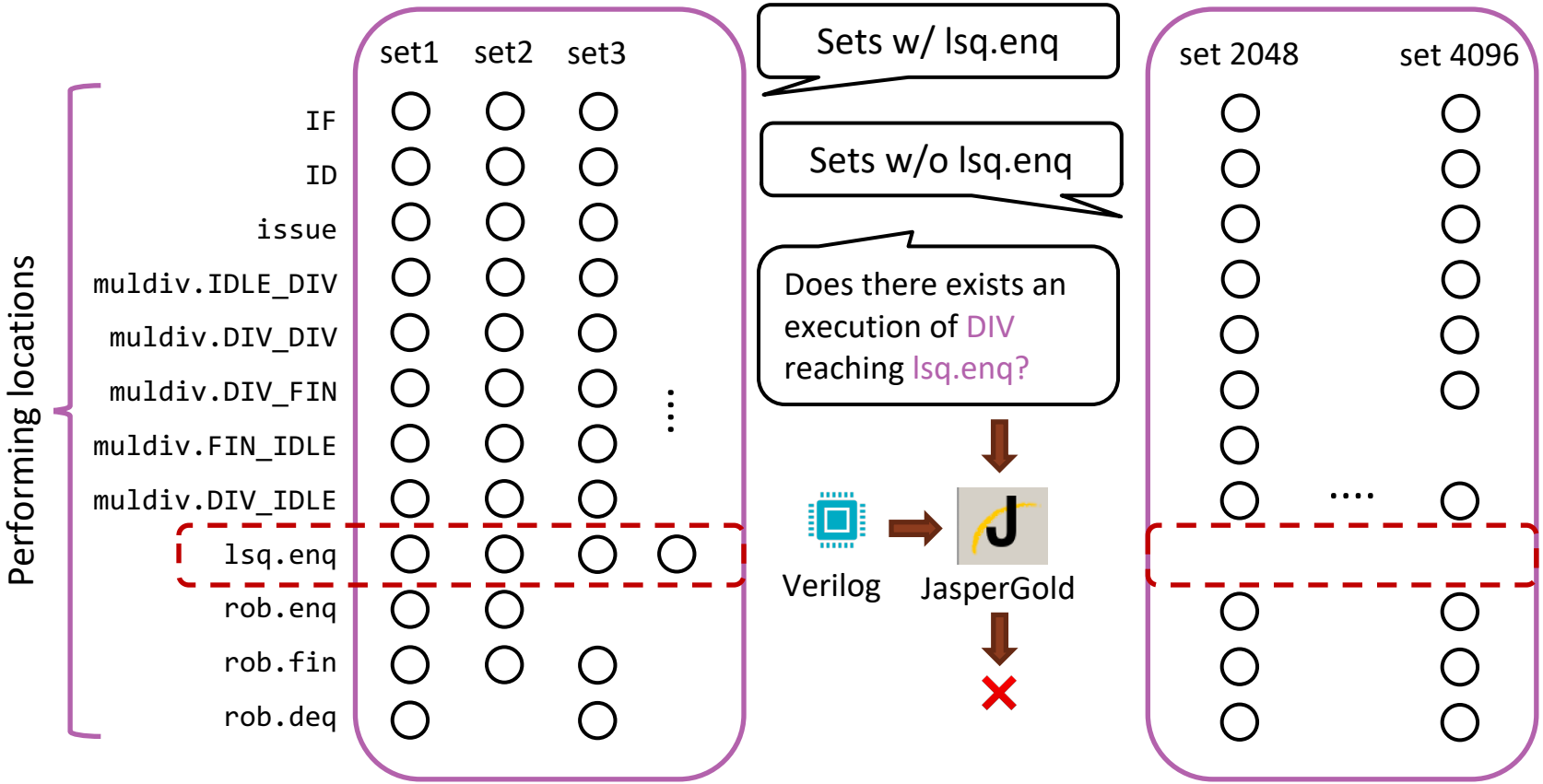


[1]Zaruba et al. https://github.com/openhwgroup/cva6

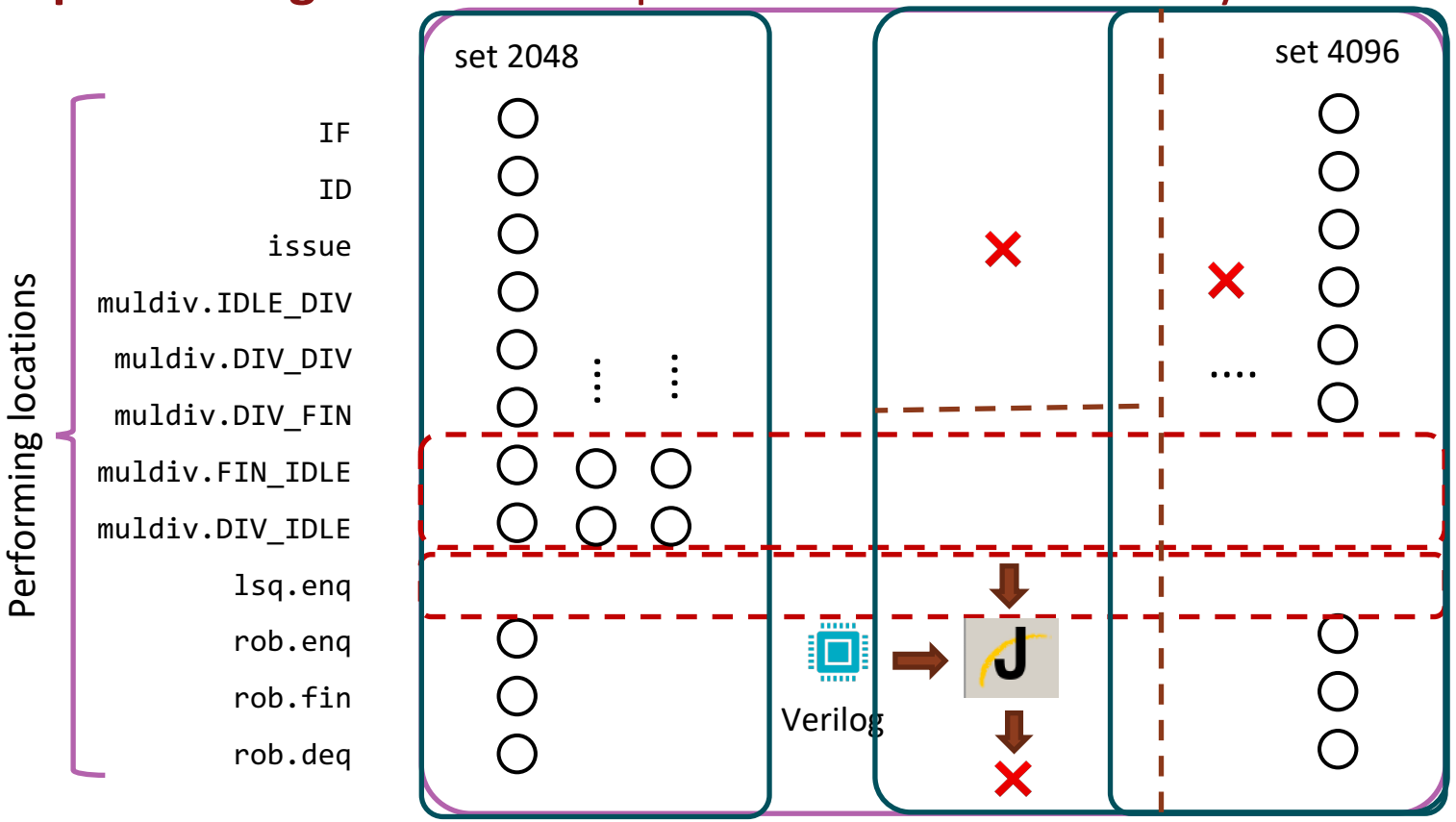
Synthesize coarse-grained execution paths: a set of visited **performing locations** + a partial order in which they are visited



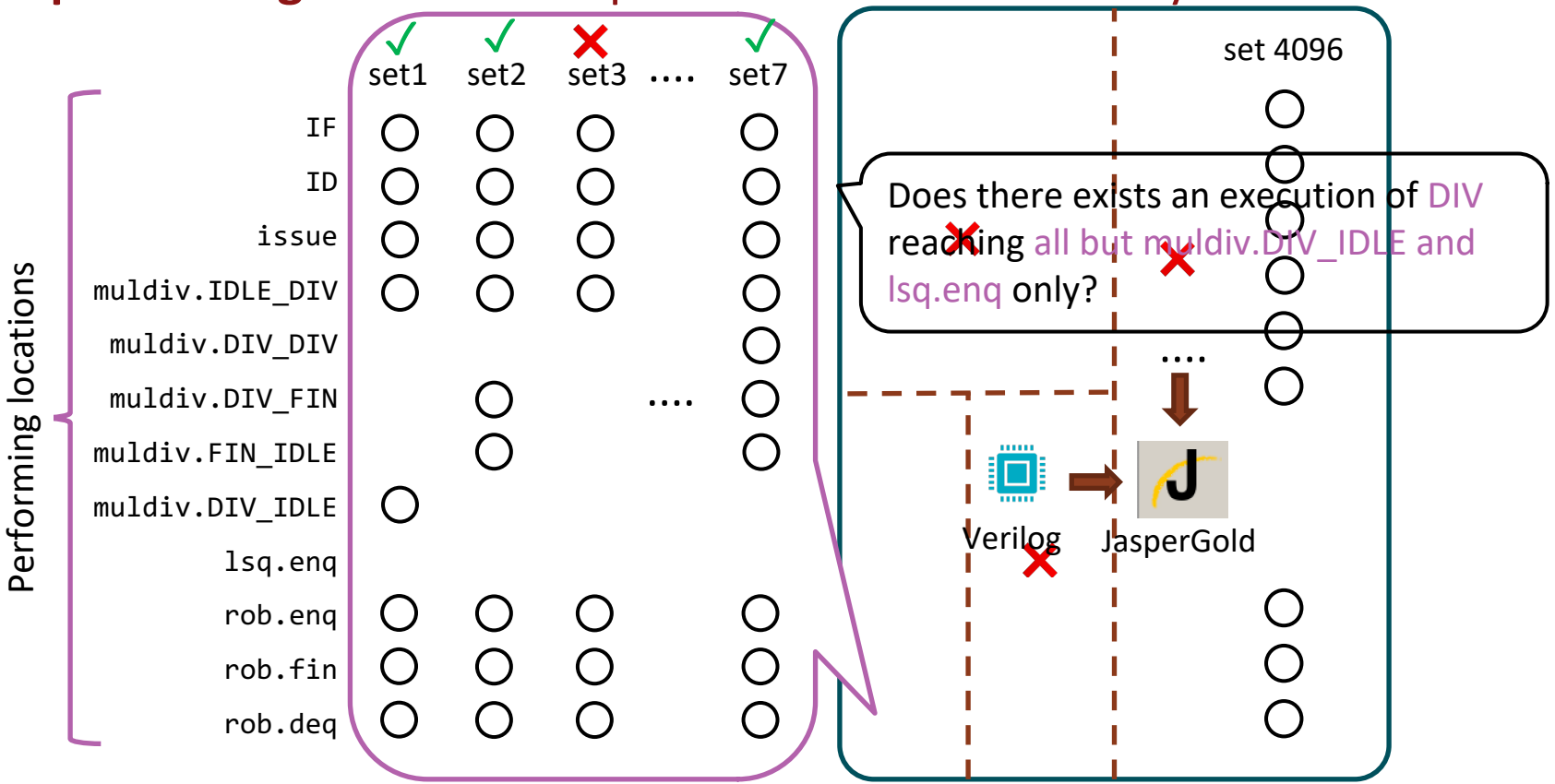
Synthesize coarse-grained execution paths: a set of visited **performing locations** + a partial order in which they are visited



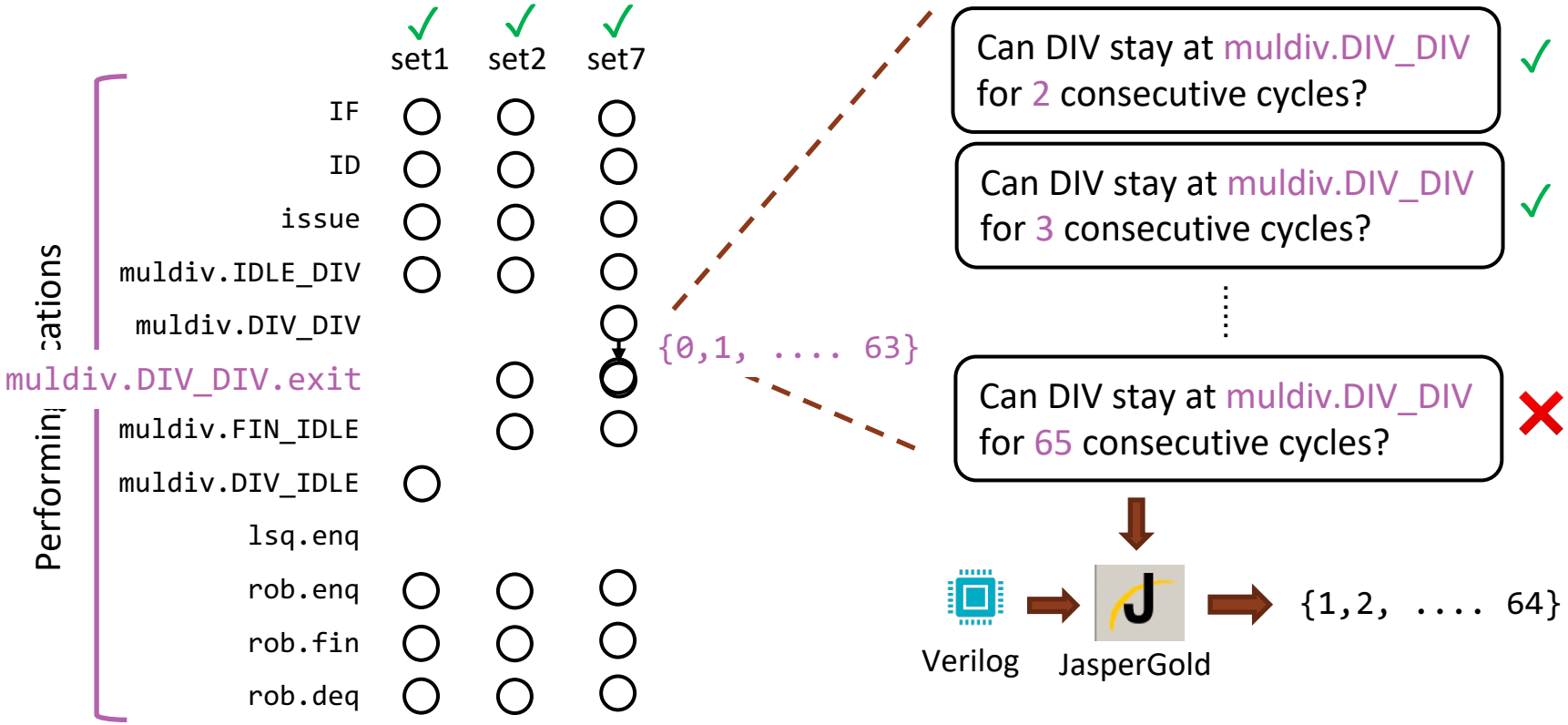
Synthesize coarse-grained execution paths: a set of visited **performing locations** + a partial order in which they are visited



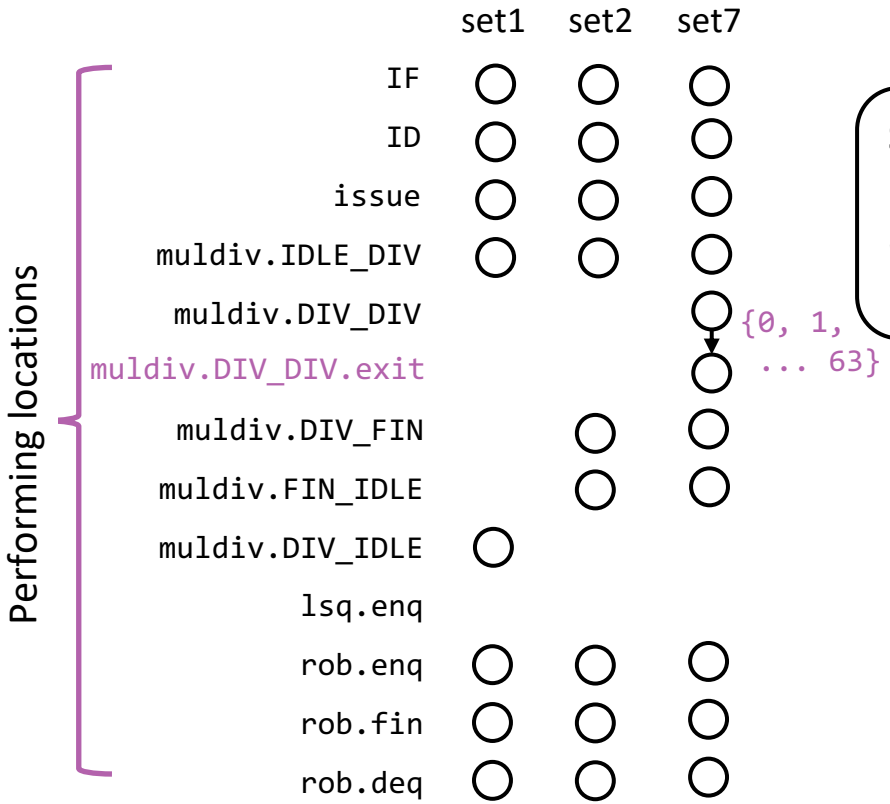
Synthesize coarse-grained execution paths: a set of visited **performing locations** + a partial order in which they are visited



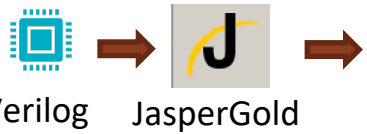
Synthesize coarse-grained execution paths: a set of visited **performing locations** + a partial order in which they are visited

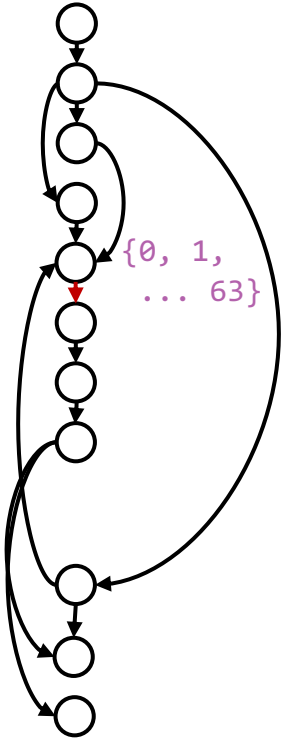


Synthesize coarse-grained execution paths: a set of visited performing locations + a **partial order** in which they are visited



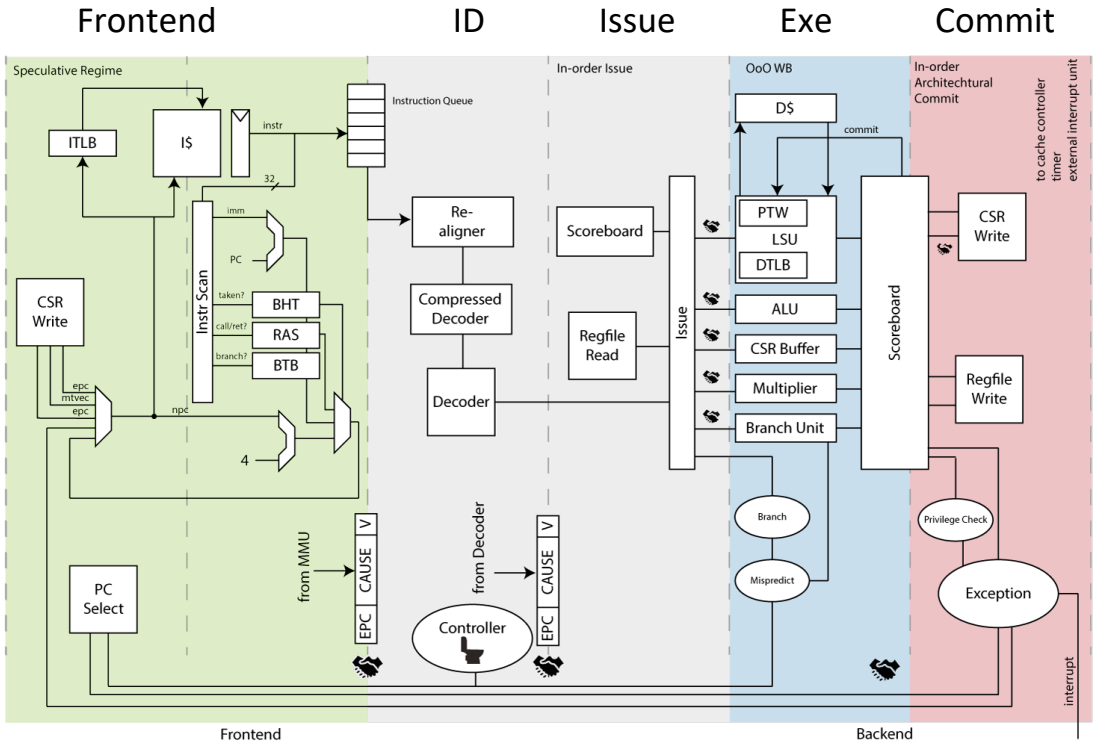
SVA property: For an ordered pair of nodes (u, v), does DIV always reach u before it reaches v?





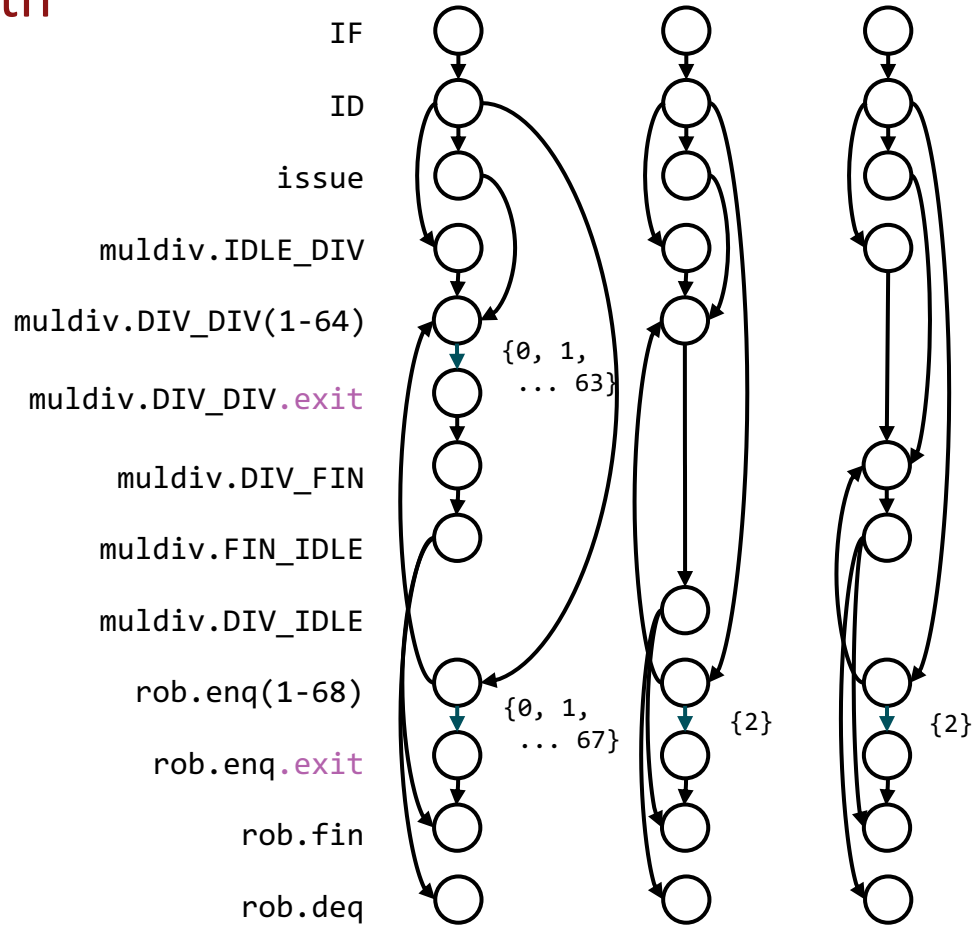
Open-source RISC-V CVA6 processor case study with proof-of-concept TransmitSynth methodology and tool

- RV64I, M, A, and C
- Single in-order issue
- Out-of-order write-backs
- Speculation
- DIV executes variably as function of its operands



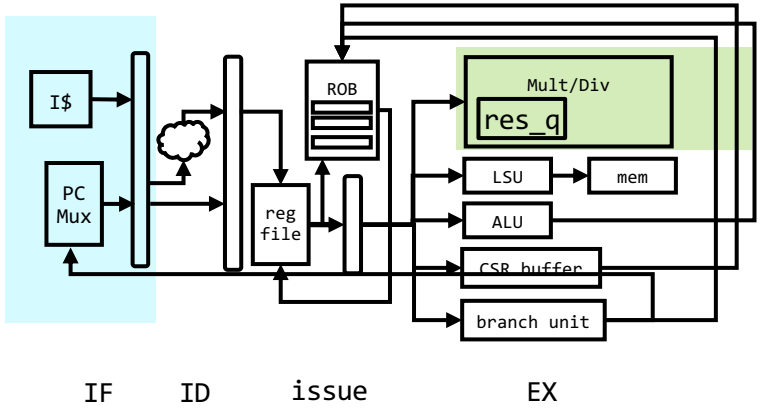
Discovering Transmitters with TransmitSynth

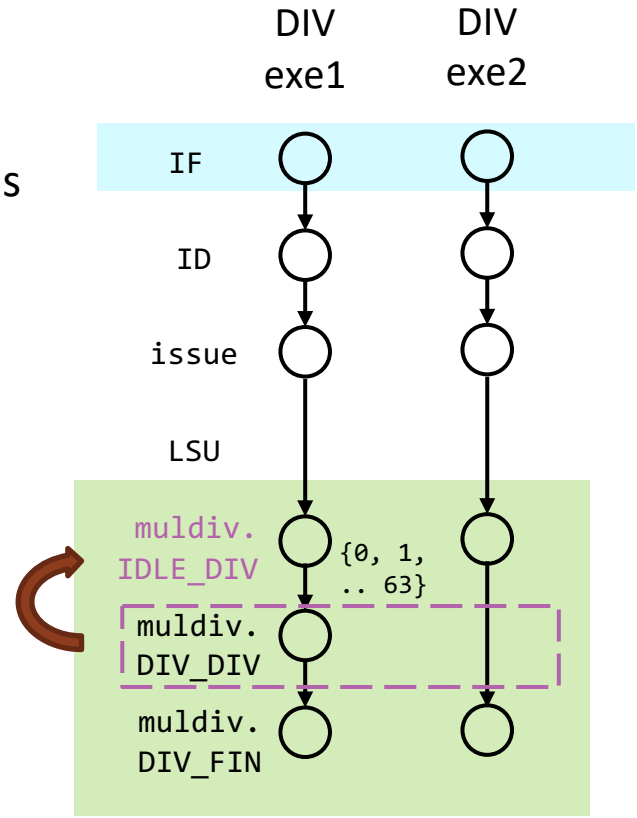
- DIV has a total of **66 execution paths!**
- Runtime to synthesize paths with TransmitSynth: **96 minutes** of serial **(parallelizable)** proof time
- Note that CVA6 has 4.54x more flops and 16.2x more gates than the multi-V-scale



Ongoing work

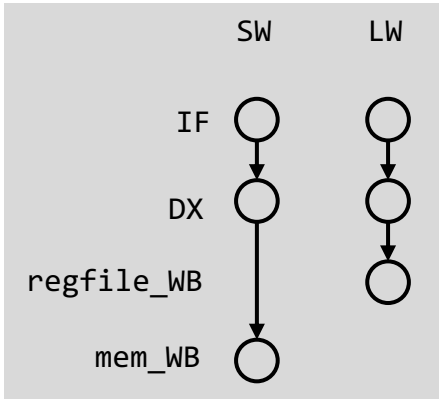
- What information is leaked?
  - What caues the different execution
  - Dataflow behavior on some performing locations
- Finer-grained execution path
  - What are updated during a performing location

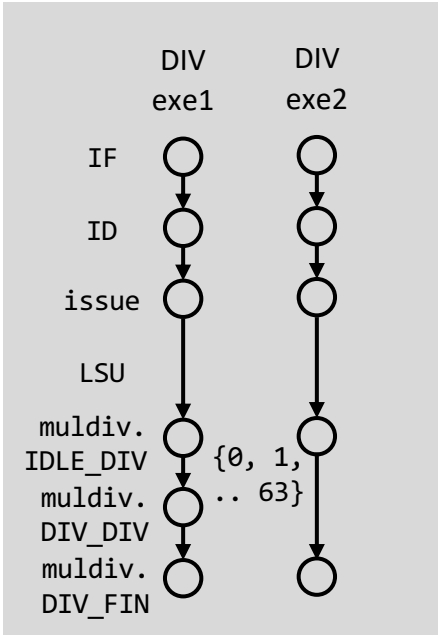


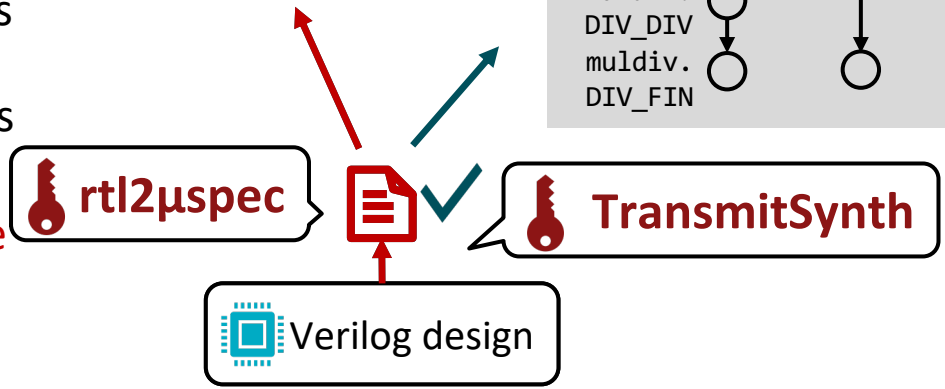


Conclusions: Automated synthesis of hardware-software contracts from RTL is feasible

- **rtl2µspec[1]:** Automated synthesis of µspec models from RTL
  - Synthesized a µspec model of the open-source RISC-V multi-V-scale processor in 6.84 minutes serial proof time
- **TransmitSynth:** Automated synthesis of security contracts from RTL via identification of transmit instructions
  - Synthesized 66 paths for DIV instructions on the open-source RISC-V CVA6 processor in 96 minutes serial proof time







[1]https://github.com/yaohsiaopid/rtl2uspec

Thanks!

yaohsiao@stanford.edu