Automatically Uncovering Hardware Side-Channels in Processor RTL with Multi-μPATH Synthesis

Yao Hsiao[1], Nikos Nikoleris[2], Artem Khyzha[2], Dominic P. Mulligan[3],
Gustavo Petri[3], Christopher W. Fletcher[4], Caroline Trippel[1]

[1]Stanford University, [2]Arm, [3]Amazon Web Service, [4]University of California, Berkeley

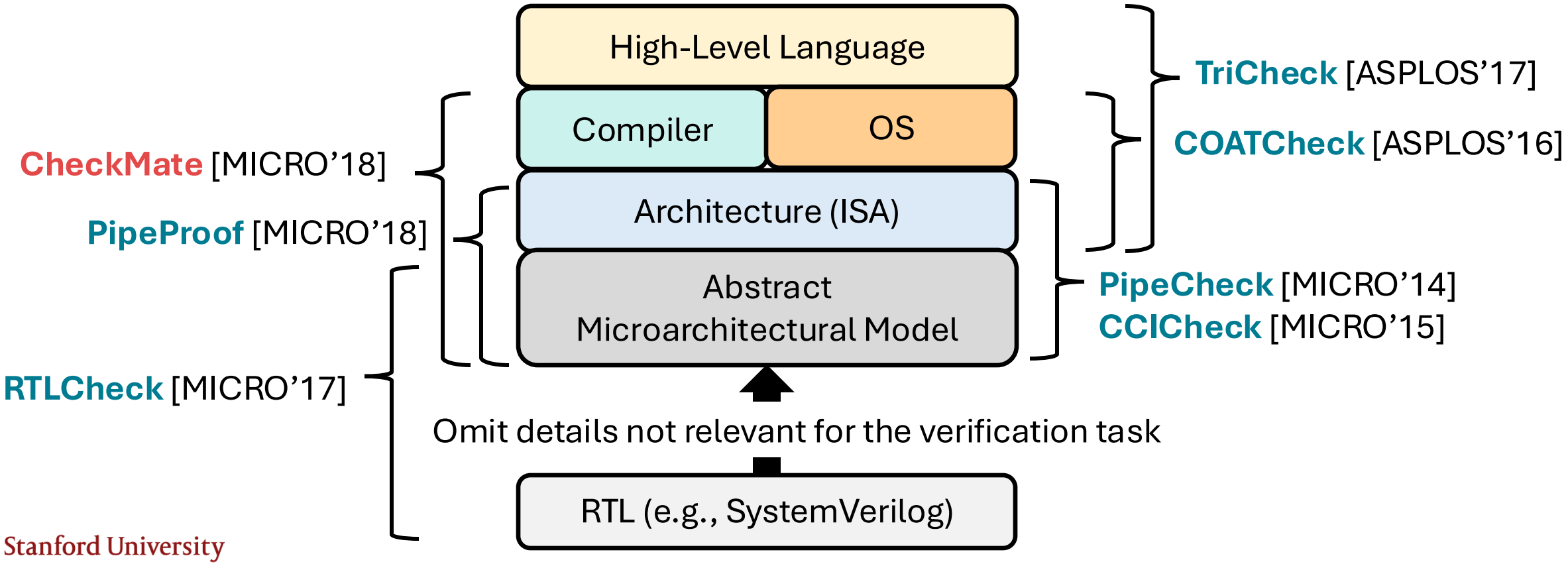
Original Work is published in 2024 International Symposium on Microarchitecture (MICRO)

2025 OSCAR workshop

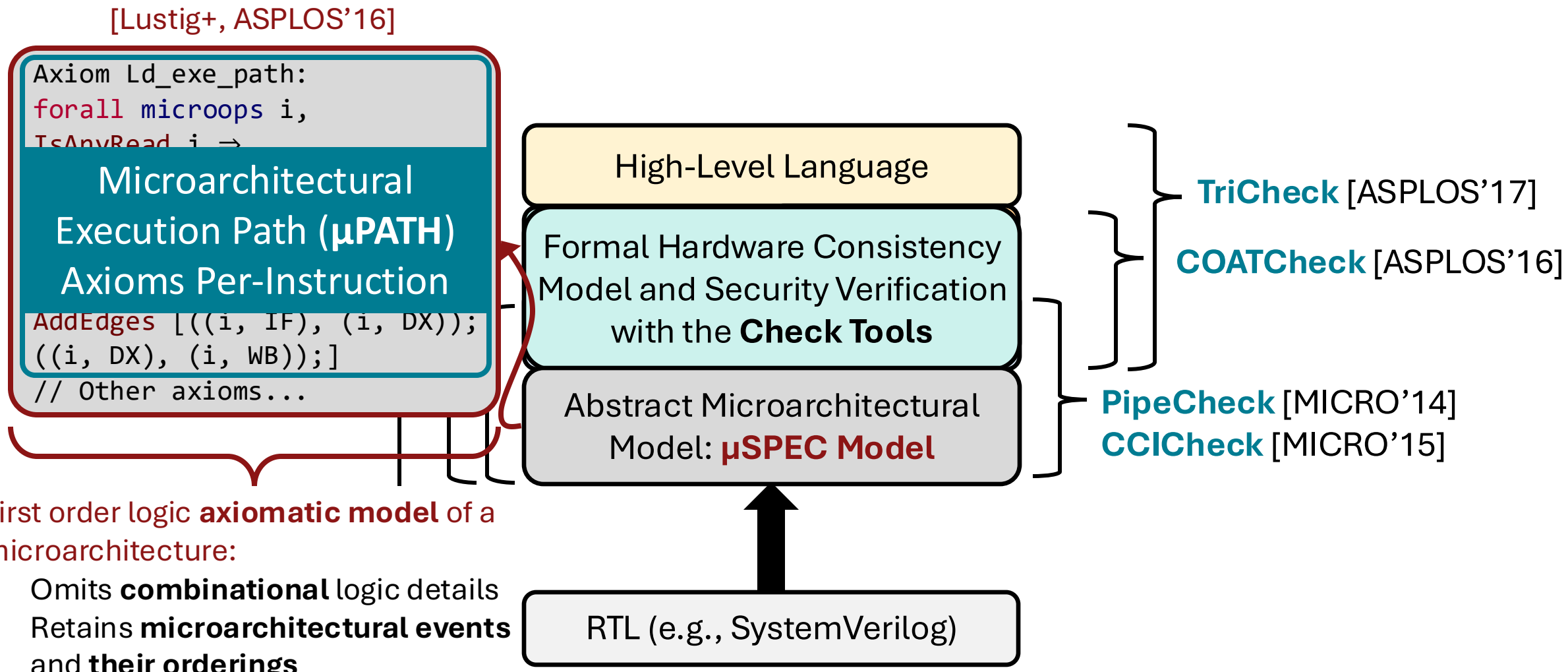
Stanford University

Design Abstraction Helps Scale Formal Hardware Verification

Example: The **Check Tools** automate **memory consistency model** (MCM) and **security** verification of processor microarchitectures.



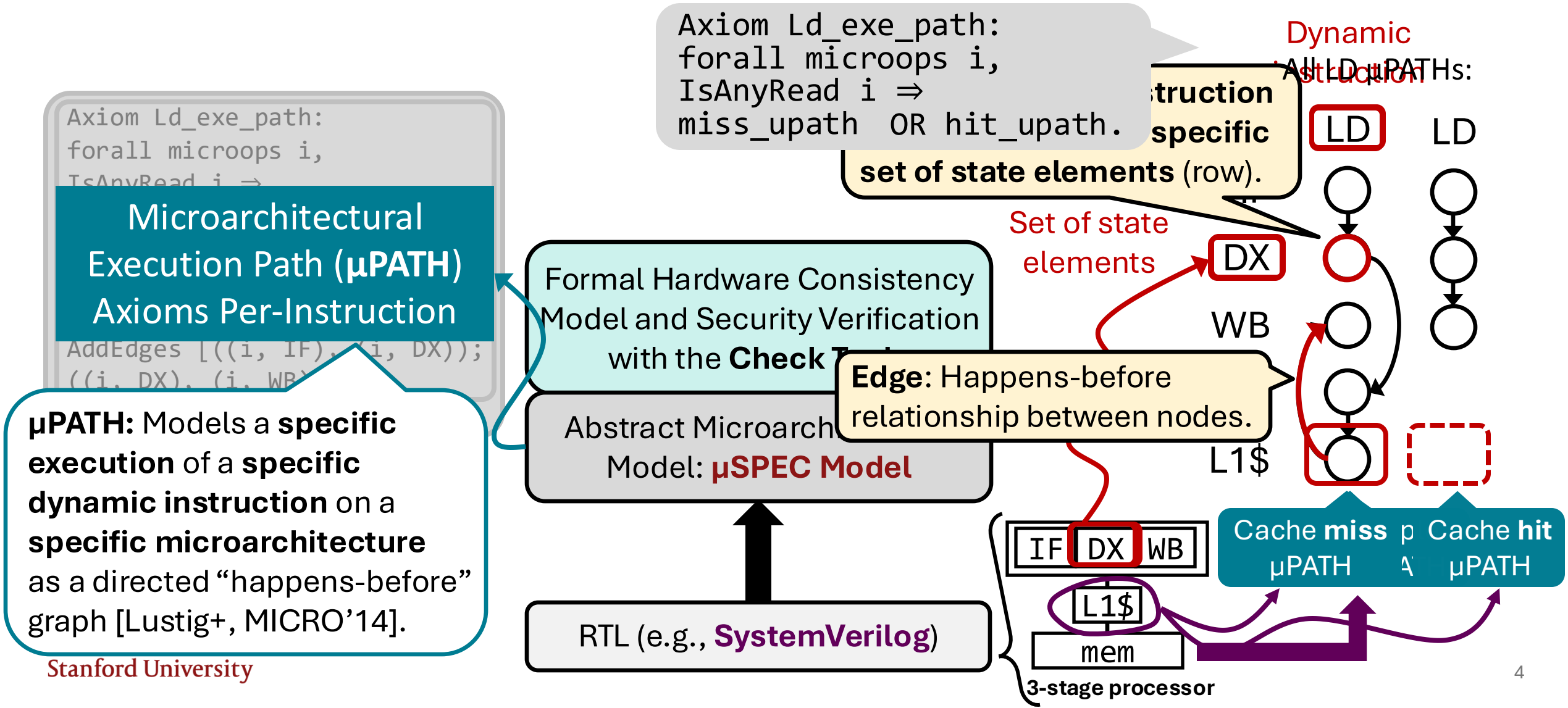
Axiomatic Microarchitectural Models Enable Formal Analysis



First order logic **axiomatic model** of a microarchitecture:

- Omits **combinational** logic details
- Retains **microarchitectural events** and **their orderings**

Axiomatic Microarchitectural Models Enable Formal Analysis



Verification Challenge: How to Verify that a μSPEC Accurately Represents a SystemVerilog Microarchitecture

μSPEC looks **quite different** from SystemVerilog!

```
Axiom Ld_exe_path:
forall microops i,
IsAnyRead i ⇒
AddEdges [((i, IF), (i, DX));
```

Handwritten

```
((i, L1$), (i, WB));] \/
AddEdges [((i, IF), (i, DX));
((i, DX), (i, WB));]
// Other axioms...
```

Formal Hardware Consistency Model and Security Verification with the **Check Tools**

Abstract Microarchitectural Model: **μSPEC Model**

μSPEC-RTL Verification Gap

RTL (e.g., **SystemVerilog**)

```
always @(posedge clk) begin
  if (!rst_n) begin
    ...
    id_op <= if_op;
    id_ex_vld <= if_ex_vld;
    ...
```

Designer Inspection

Problem: Does μSPEC accurately represent the RTL?

Roadmap

Synthesizes **all μPATH axioms** required to define a **formally verified** μspec model (i.e., ~50% of μspec model) .

- **Background**: The Microarchitecture-μSPEC Model Verificatio[illegible]allenge

- **RTL2MμPATH**: Synthesizing ("Uncovering") All μPATHs per Instruction from Advanced SystemVerilog Processors

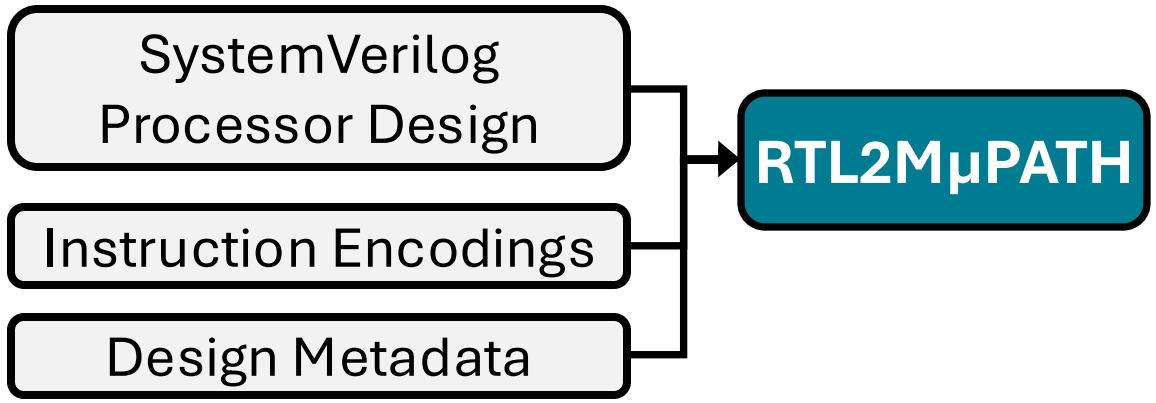
- **Key Insight**: μPATH Variability (>1 μPATH) is a Strong Indicator of a Hardware Side-Channel
- **SynthLC**: Synthesizing Formally Verified "Leakage Signatures" from SystemVerilog Processors

Application of μPATH synthesis for uncovering all hardware side-channels!

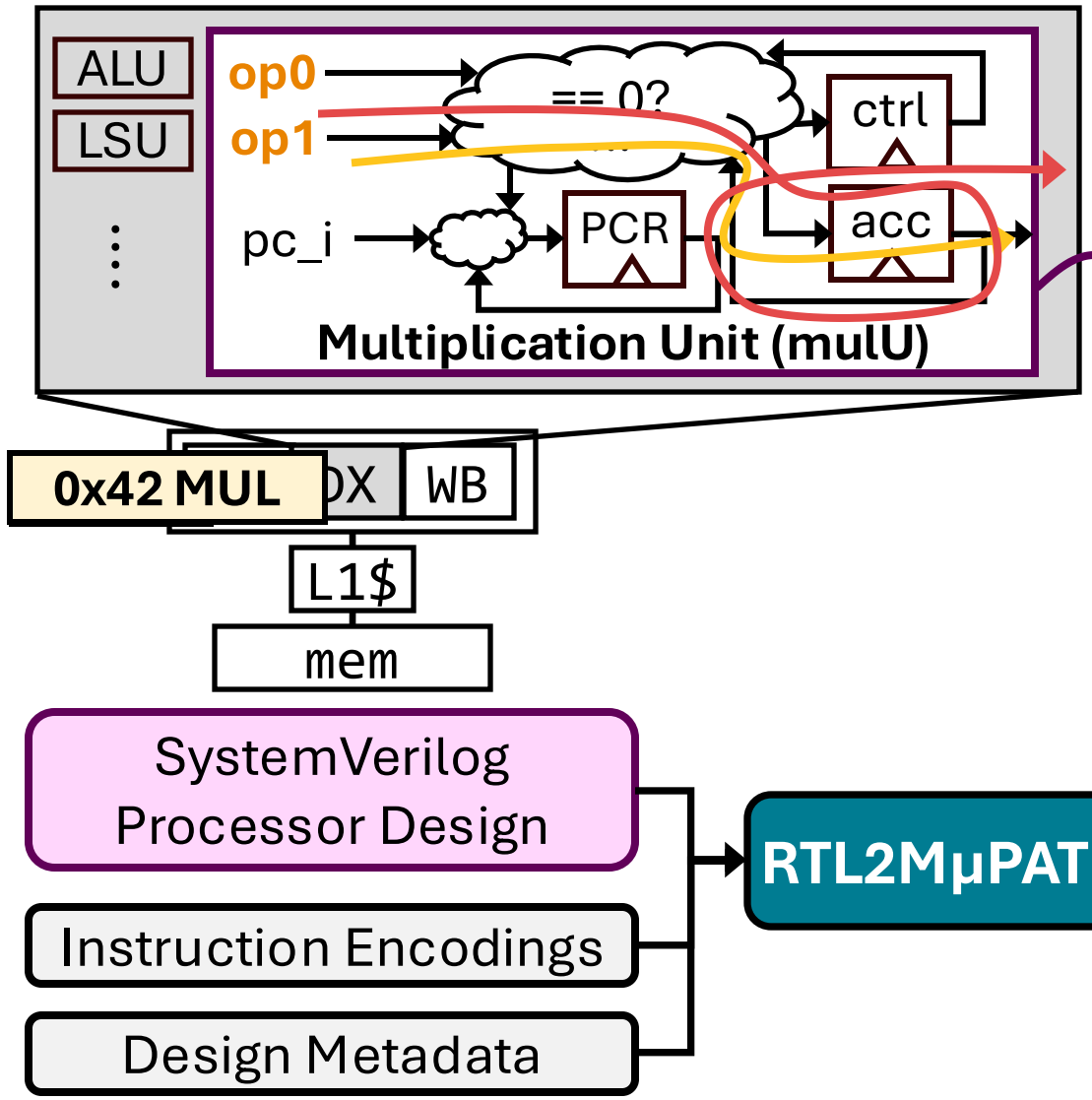
Roadmap

- **Background**: The Microarchitecture-μSPEC Model Verification Challenge
- **RTL2MμPATH**: Synthesizing ("Uncovering") All μPATHs per Instruction from Advanced SystemVerilog Processors
- **Key Insight**: μPATH Variability (>1 μPATH) is a Strong Indicator of a Hardware Side-Channel
- **SynthLC**: Synthesizing Formally Verified "Leakage Signatures" from SystemVerilog Processors

Overview of RTL2MμPATH: Multi-μPATH Synthesis from RTL



Overview of RTL2MμPATH: Multi-μPATH Synthesis from RTL



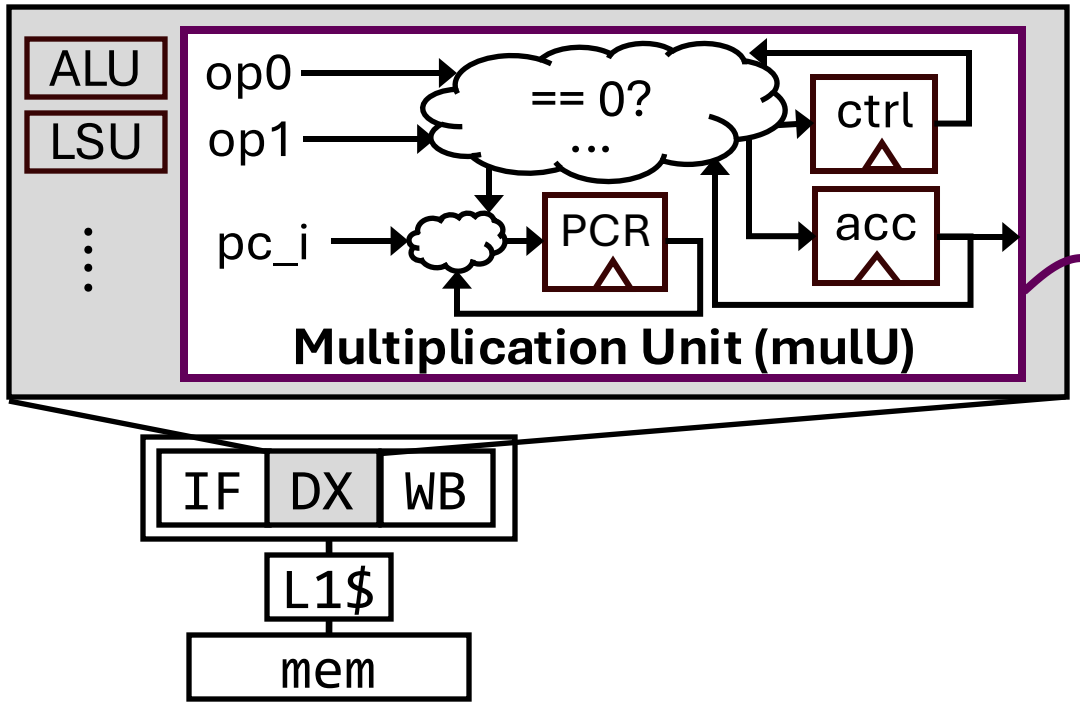
visits mulU by putting its PC value the

the active i mulU's MUL's own (N) operands

```
MUL_mulU(MUL^N i0):
 if (i0.op0 == 0 ||
     i0.op1 == 0):
   return fast_path  # 1 cyc
return slow_path   # 3 cyc
```

Zero-Skip Optimization

Overview of RTL2MμPATH: Multi-μPATH Synthesis from RTL




```
MUL_mulU(MUL^N i0):
 if (i0.op0 == 0 ||
     i0.op1 == 0):
   return fast_path  # 1 cyc
return slow_path    # 3 cyc
```

Zero-Skip Optimization

SystemVerilog Processor Design

Instruction Encodings

Design Metadata

RTL2MμPATH

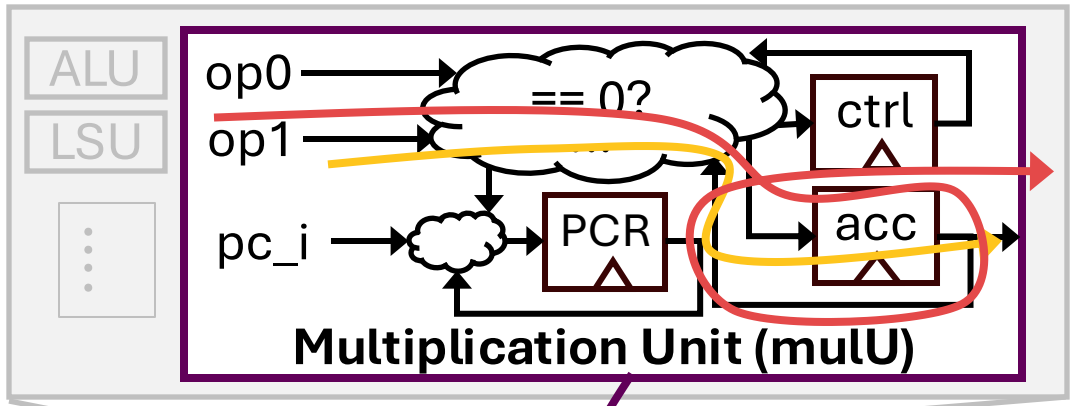
Techniques

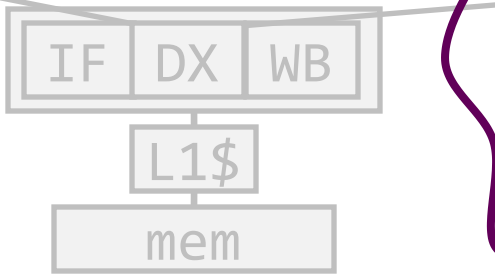
- Netlist Analysis
- SystemVerilog Assertion (SVA) Generation from Templates
- Model Checking

```
assert (property)
```

```
prove assert (property)
```

Overview of RTL2MμPATH: Multi-μPATH Synthesis from RTL





SystemVerilog Processor Design

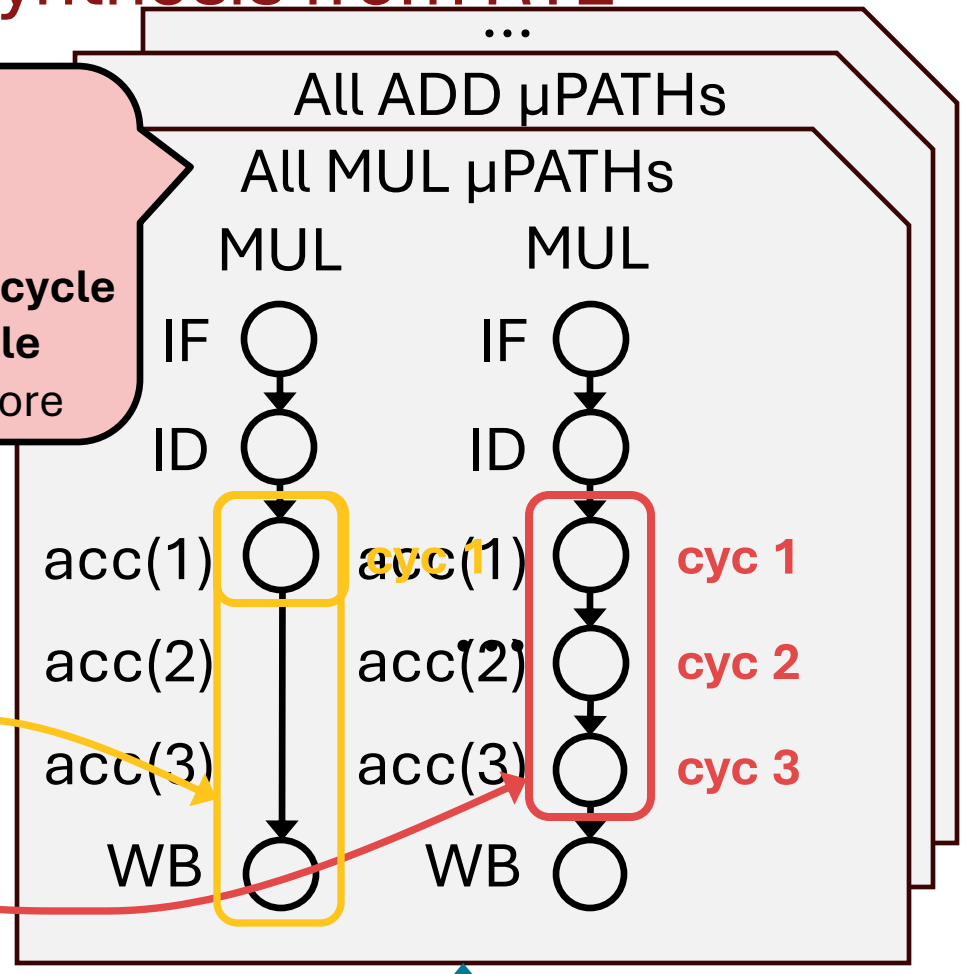
Instruction Encodings

Design Metadata

Cycle-accurate μPATHs (new!):
- Nodes: State updates **in a cycle**
- Edges: **1-cycle** happens-before

```
MUL_mulU(MUL^N i0):
 if (i0.op0 == 0 ||
     i0.op1 == 0):
   return fast_path  # 1 cyc
return slow_path   # 3 cyc
```

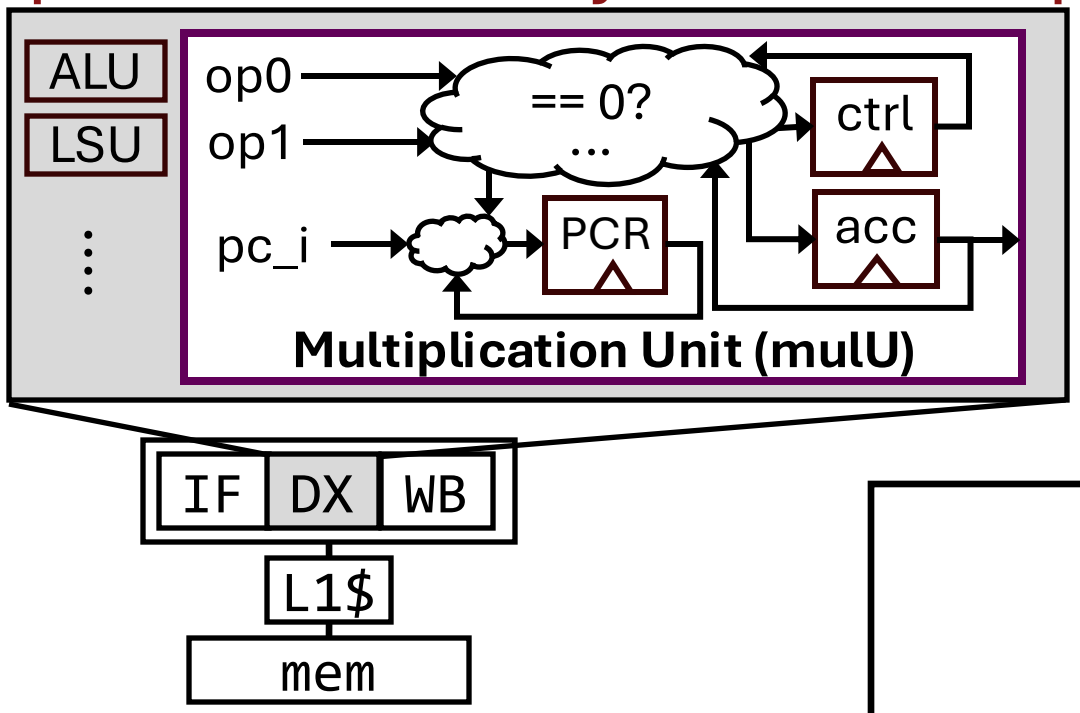
Zero-Skip Optimization

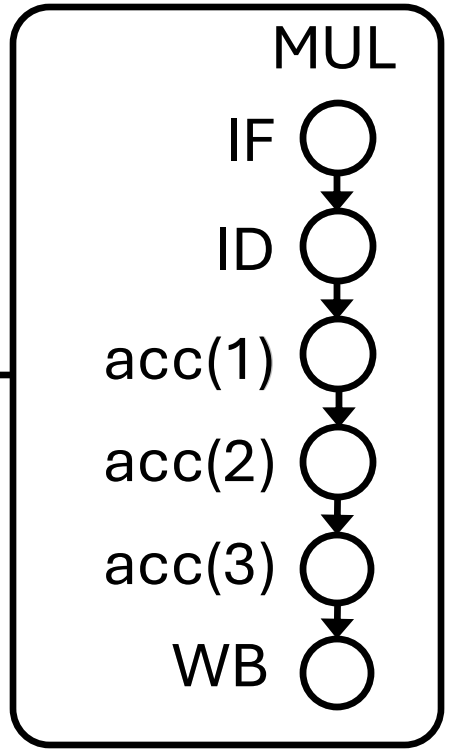


RTL2MμPATH

Complete set of μPATHs for each instruction

Conceptualizing Nodes in a μPATH: A Key Challenge to Automated μPATH Discovery with RTL2MμPATH





...

SystemVerilog Processor Design

Instruction Encodings

Design Metadata

RTL2MμPATH

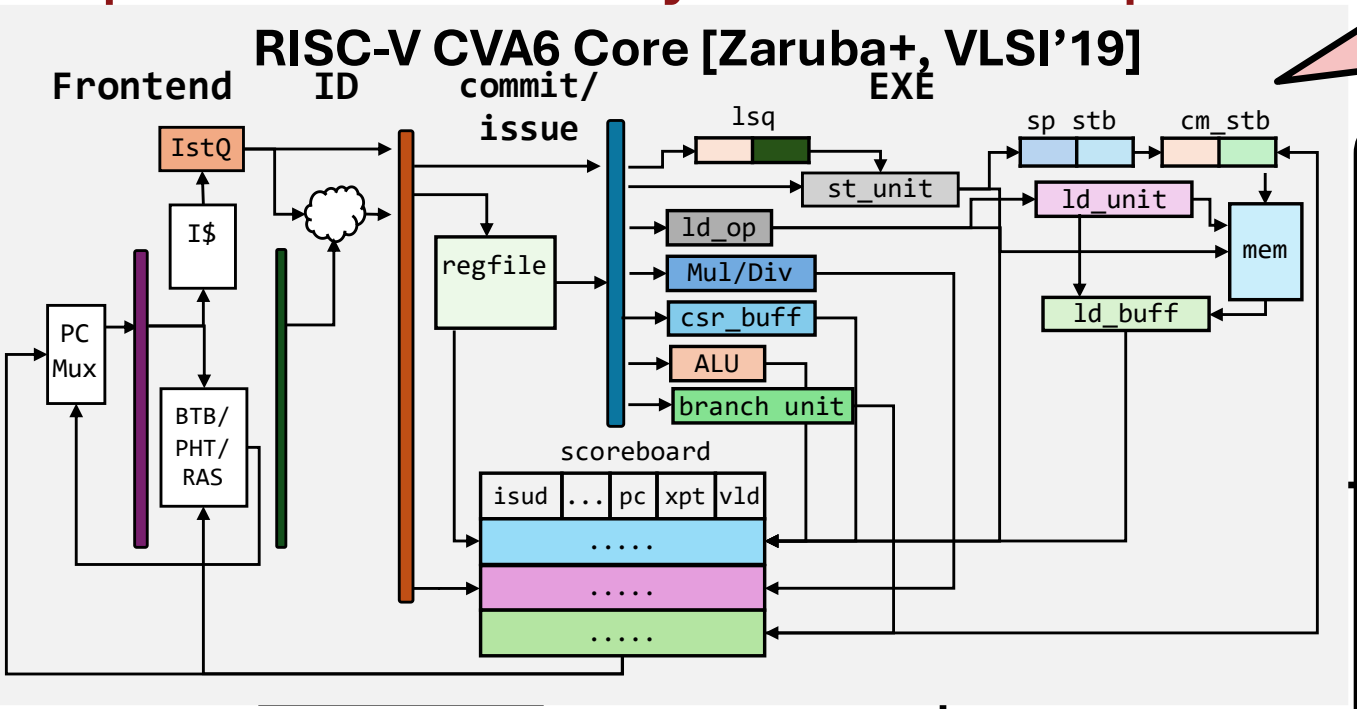
`cover (MUL exhibit μPATH)`

Asks a **model checker** whether μPATH is **reachable** by MUL in **any execution**.

All reachable MUL μPATHs

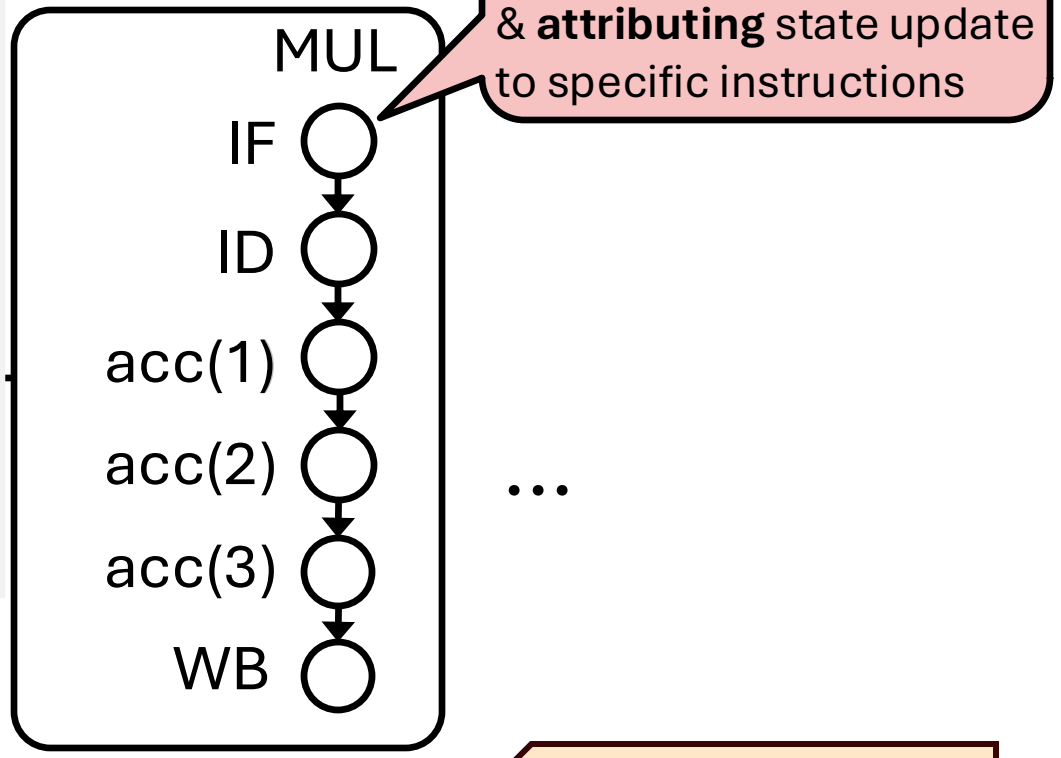
Discard unreachable MUL μPATHs

Conceptualizing Nodes in a μPATH: A Key Challenge to Automated μPATH Discovery with RTL2MμPATH



Many instructions (colors) **in-flight at the same time!**

How to recognize a node? Requires **detecting** & **attributing** state update to specific instructions



SystemVerilog Processor Design

Instruction Encodings

Design Metadata

RTL2MμPATH

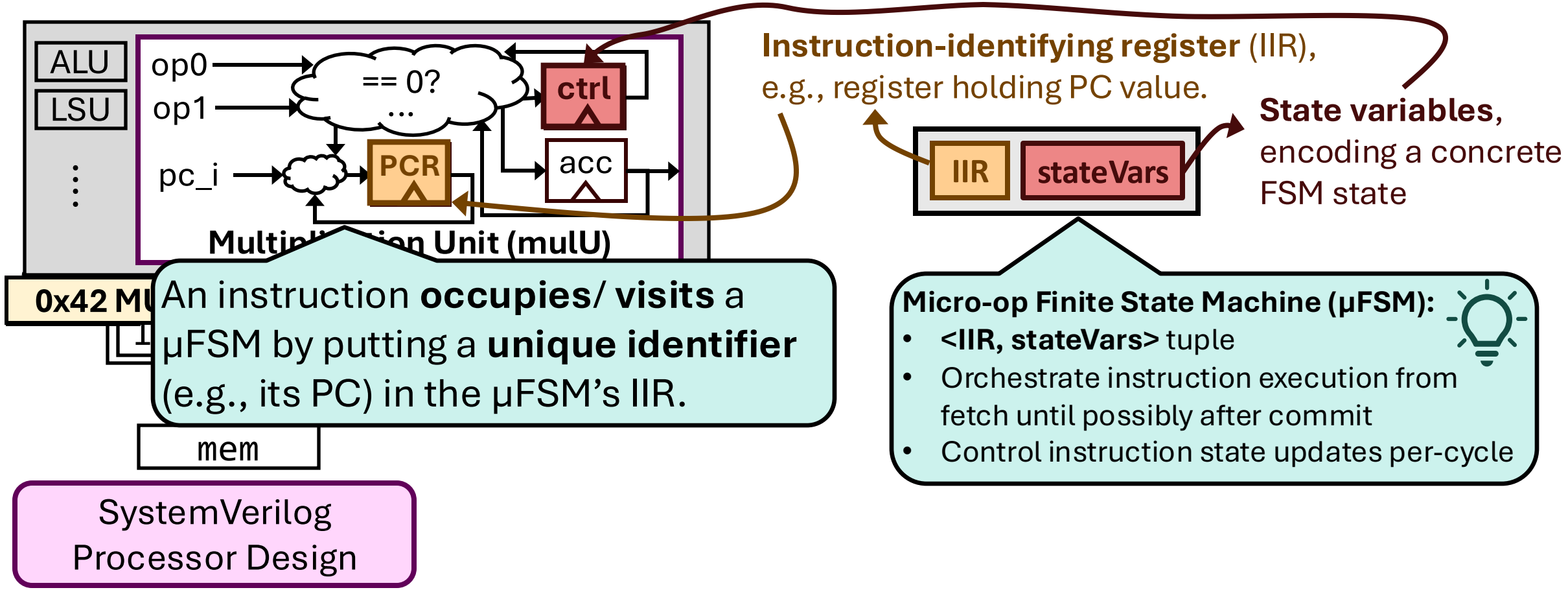
`cover (MUL exhibit μPATH)`

Asks a **model checker** whether μPATH is **reachable** by MUL in **any execution**.

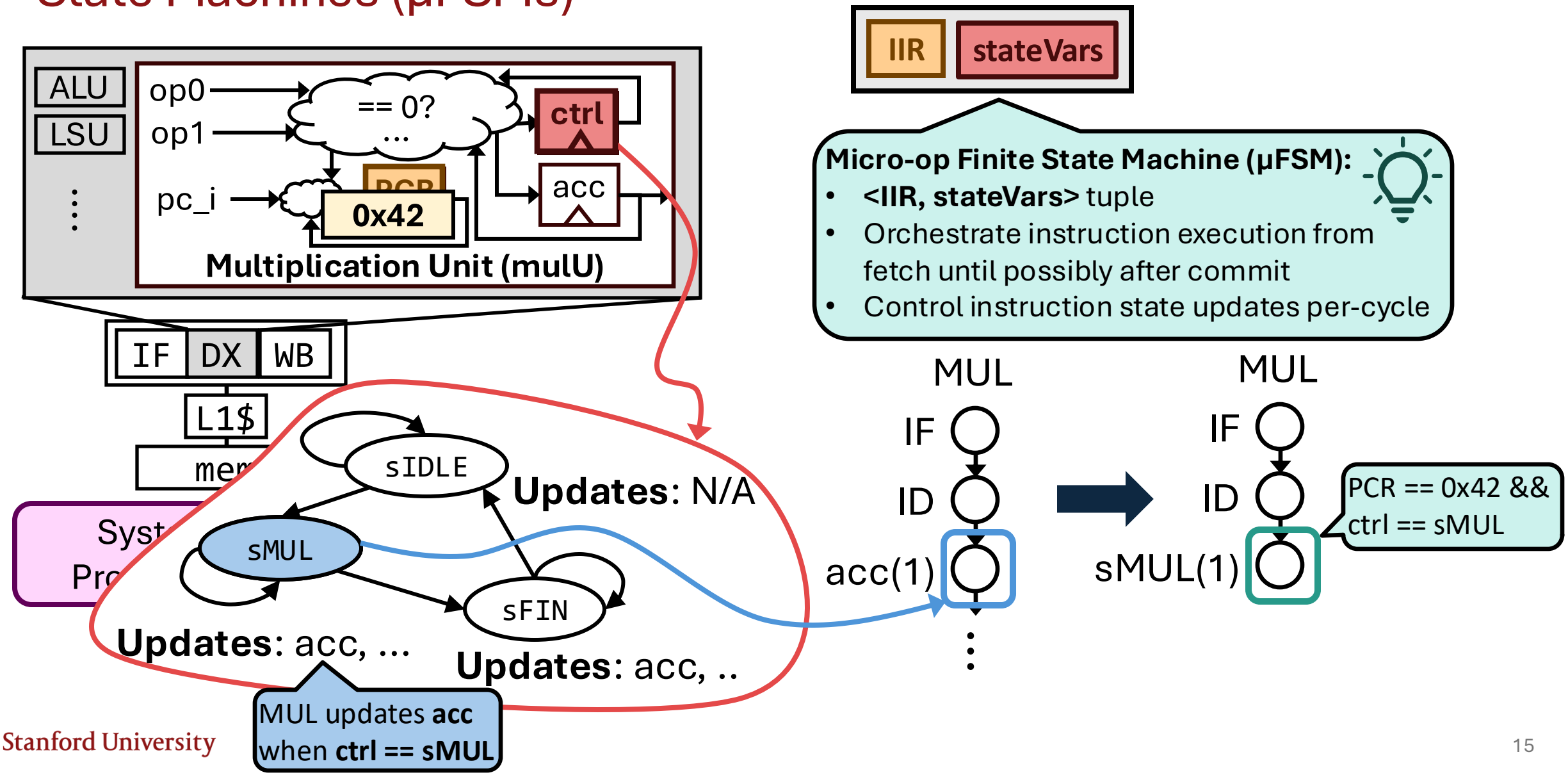
All reachable MUL μPATHs

Discard unreachable MUL μPATHs

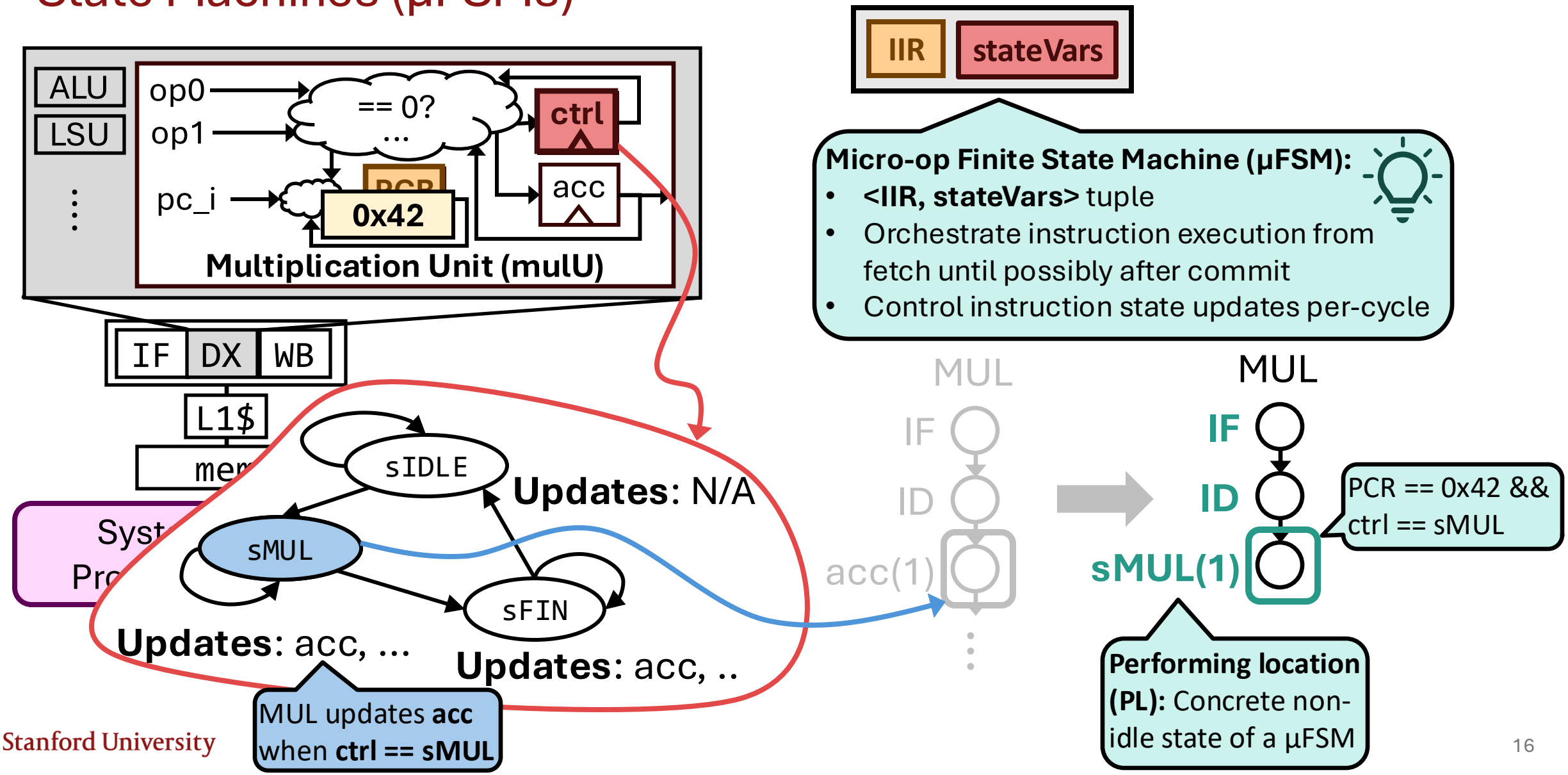
Our Solution: Expressing Nodes in μPATHs using Micro-op Finite State Machines (μFSMs) from a Processor's Control Path



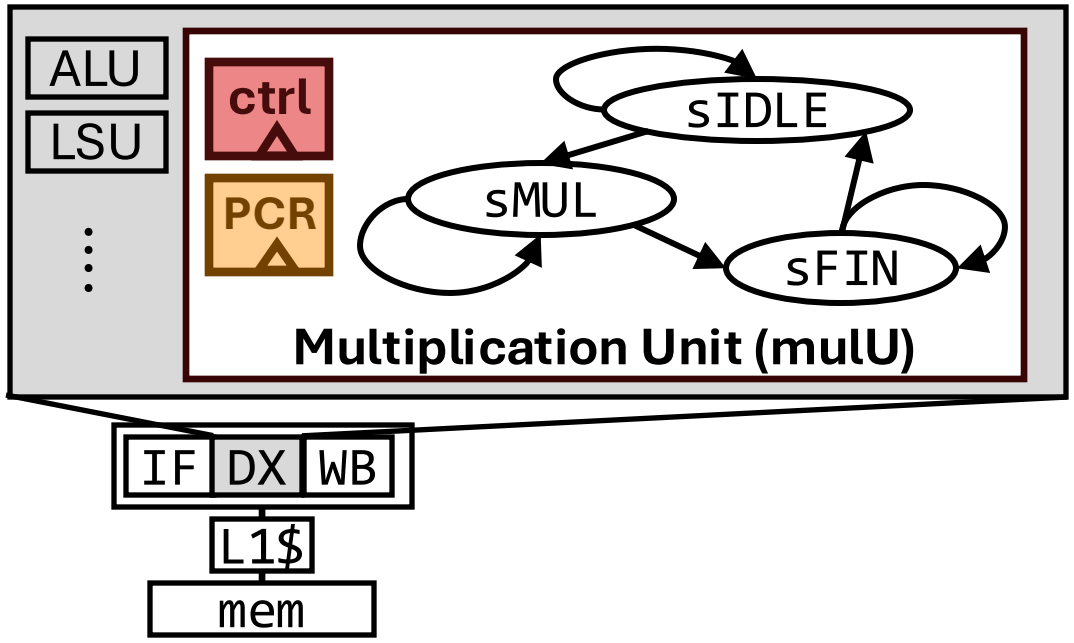
Our Solution: Expressing Nodes in μPATHs using Micro-op Finite State Machines (μFSMs)



Our Solution: Expressing Nodes in μPATHs using Micro-op Finite State Machines (μFSMs)



RTL2MμPATH: Synthesizing μPATHs from Processor Design



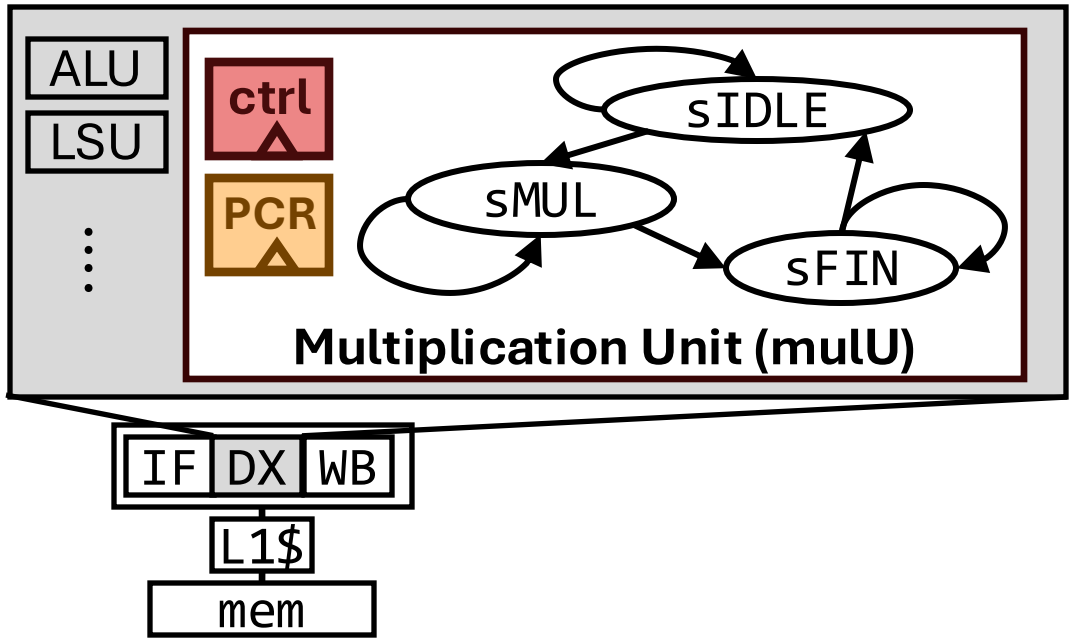
SystemVerilog Processor Design

Instruction Encodings

Design Metadata (including **μFSMs**)

RTL2MμPATH

RTL2MμPATH: Synthesizing μPATHs from Processor Design



SystemVerilog Processor Design

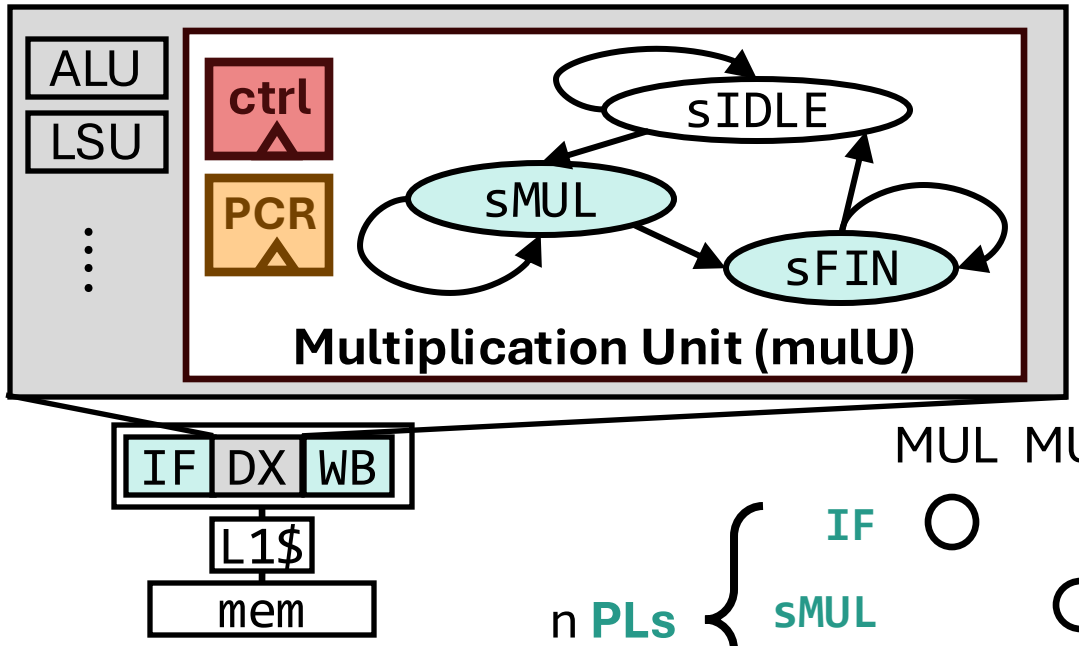
Instruction Encodings

Design Metadata (including **μFSMs**)

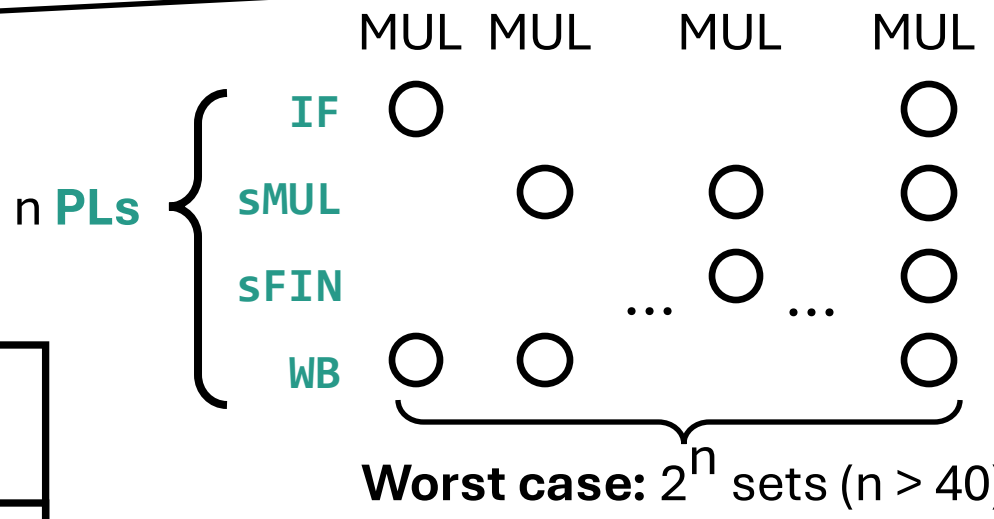
RTL2MμPATH

Step 1: Synthesizing **sets of nodes** that can form reachable μPATHs

RTL2MμPATH: Synthesizing μPATHs from Processor Design



Step 1: Synthesizing **sets of nodes** that can form reachable μPATHs



Worst case: 2^n sets (n > 40)

Step 1A: Enumerate all possible **PLs** (concrete **μFSM states**) with netlist analysis

Ask a model checker...

```
Can any instruction
visit <PL>?
```

```
Can <inst> visit <PL>?
```

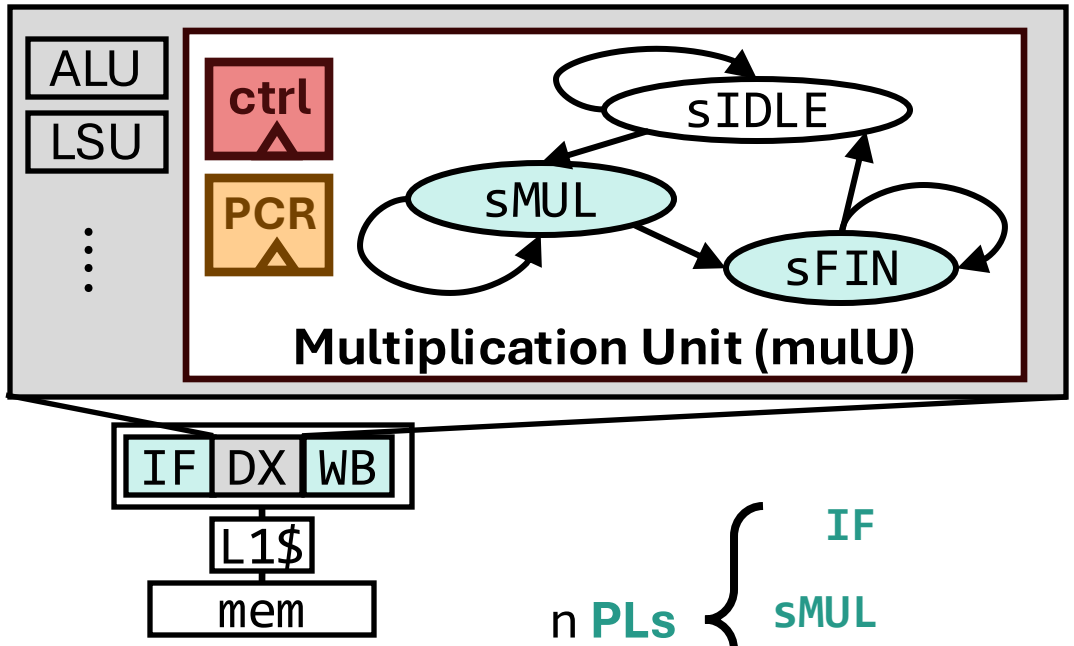
```
Can <inst> visit <PL1>
without visiting <PL2>?
```

```
Can <inst> visit both
<PL1> and <PL2>?
```

Step 1B: SVA property-driven μPATH pruning

- SystemVerilog Processor Design
- Instruction Encodings
- Design Metadata (including **μFSMs**)

RTL2MμPATH: Synthesizing μPATHs from Processor Design



Step 1: Synthesizing **sets of nodes** that can form reachable μPATHs

Step 2: Synthesizing **full μPATHs** by adding edges to reachable node sets

Details in the paper!

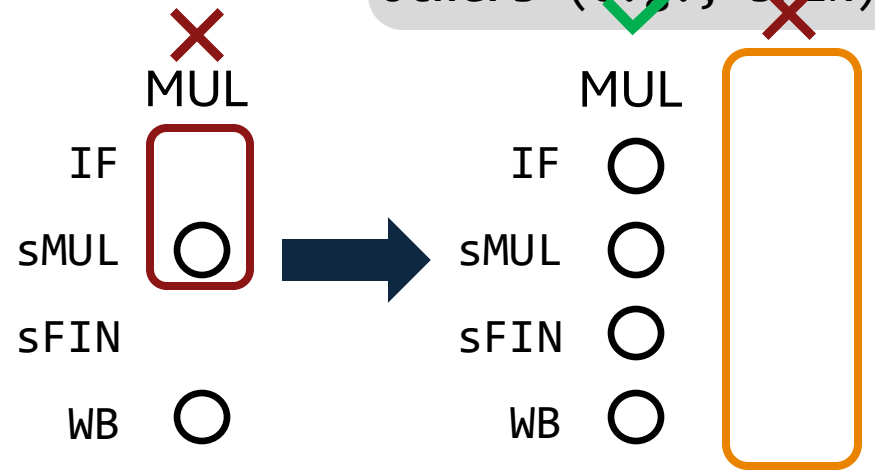
Can <MUL> visit <IF, sMUL, WB> but no others (e.g., sFIN)?

SystemVerilog Processor Design

Instruction Encodings

Design Metadata (including **μFSMs**)

n **PLs** { IF, sMUL, sFIN, WB }



Candidate sets of nodes (**10s to ~1k**)

Step 1A: Enumerate all possible **PLs** (concrete **μFSM states**) with netlist analysis

Step 1B: SVA property-driven μPATH pruning

Step 1C: Embed node sets as SVA properties to deduce reachability

Roadmap

- **Background**: The Microarchitecture-μSPEC Model Verification Challenge
- **RTL2MμPATH**: Synthesizing ("Uncovering") All μPATHs per Instruction from Advanced SystemVerilog Processors
- **Key Insight:** μPATH Variability (>1 μPATH) is a Strong Indicator of a Hardware Side-Channel
- **SynthLC**: Synthesizing Formally Verified "Leakage Signatures" from SystemVerilog Processors

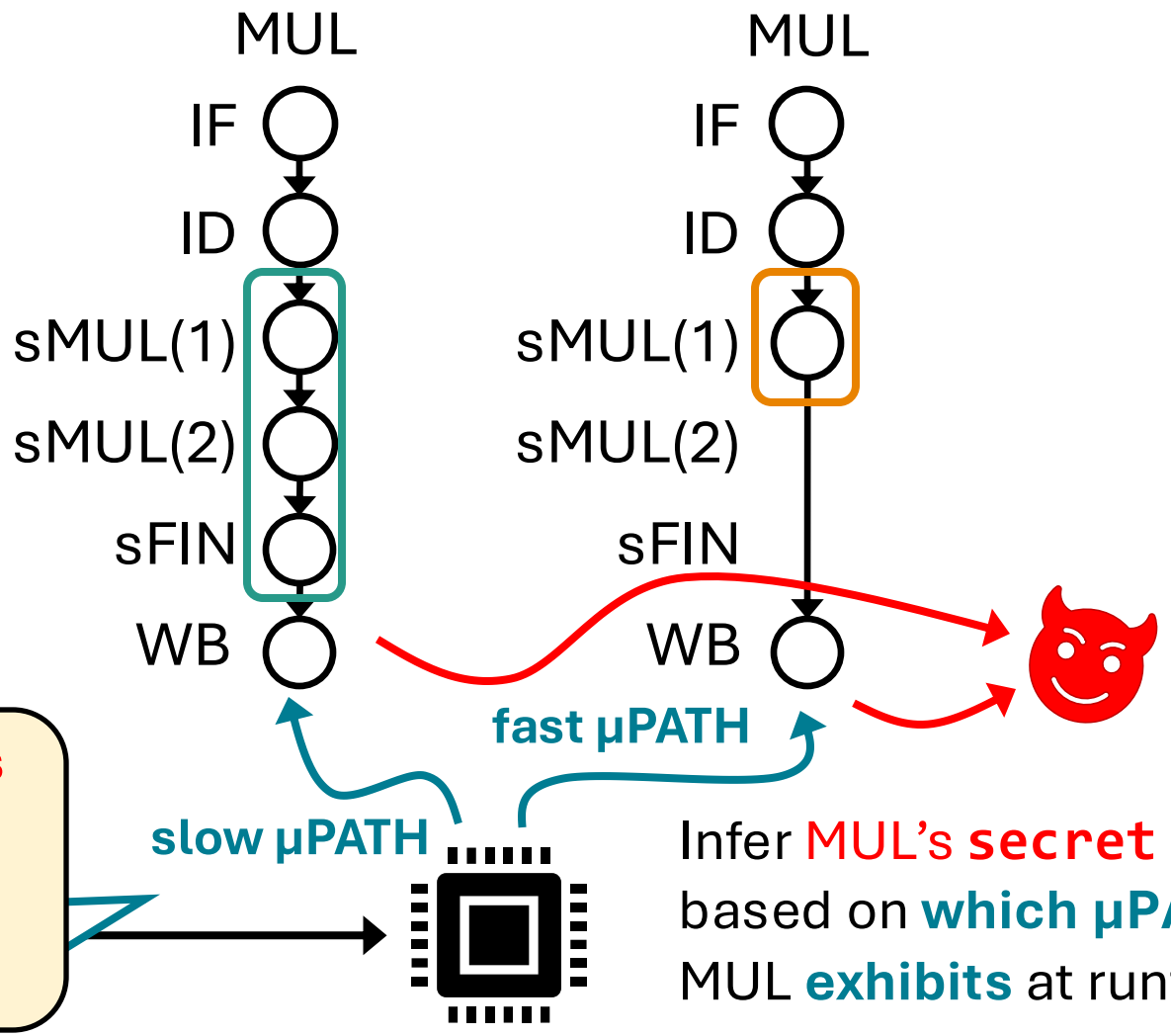
Operand-Dependent μPATH Variability on a Microarchitecture Implies Existence of Hardware Side-Channels

A typical pattern of victim program in a **classic side-channel attack**:

```
unsafe_instruction secret
```

Example on toy processor:

```
MUL secret op1
```



An instruction that **creates** and **exhibits** operand-dependent **μPATH variability** (>1 μPATH) can leak its operand(s) via a **hardware side-channel**!

Infer MUL's **secret** based on **which μPATH** MUL **exhibits** at runtime!

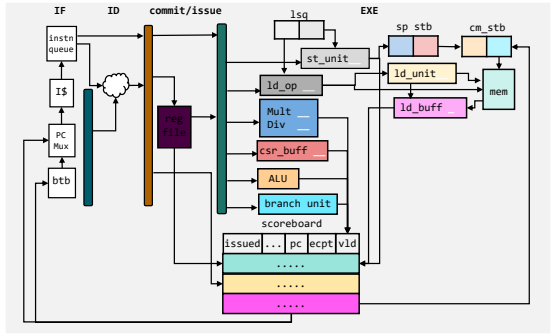
Operand-Dependent μPATH Variabilities on a Microarchitecture Imply Existence of Hardware Side-Channels

A **more subtle pattern** of victim program in a **side-channel attack**:

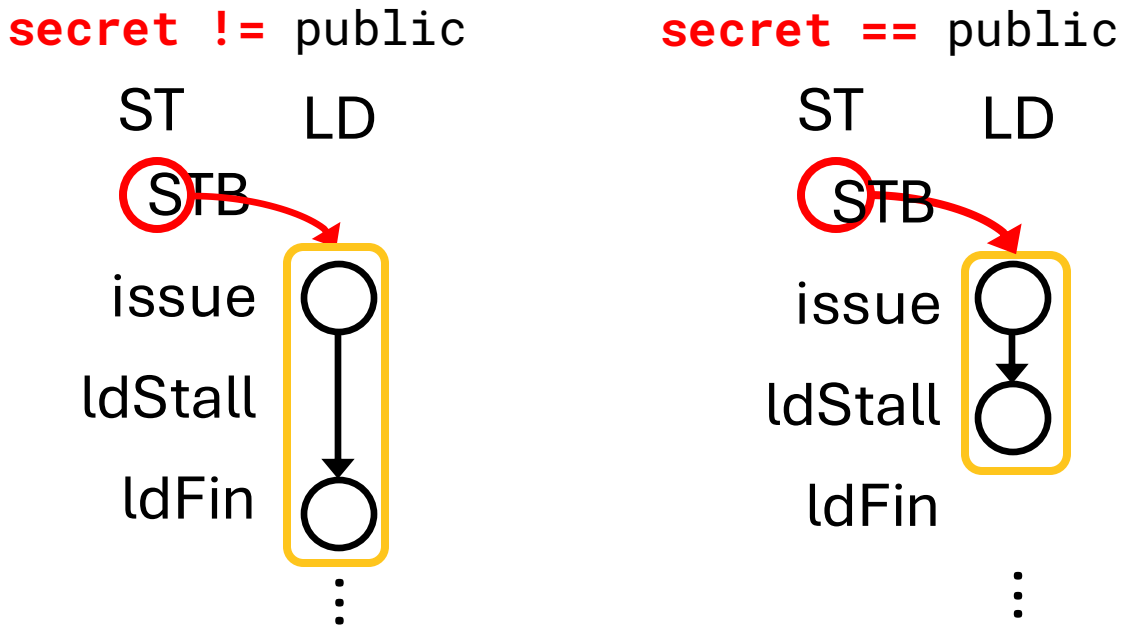
```
unsafe_instruction secret
unsafe_instruction public
```

Example on RISC-V CVA6 Core:

```
ST [secret]
LD [public]
```



CVA6 [Zaruba+, VLSI'19]

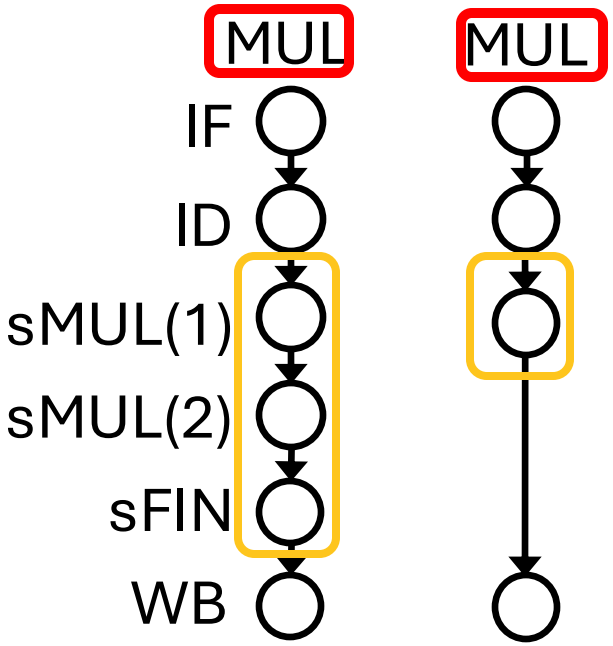


Infer ST's **secret** based on **which μPATH** **LD exhibits** at runtime!

Hardware Side-Channel Defenses for a Microarchitecture Minimally Requires Identifying Unsafe Instructions that Leak Their Operands

A typical pattern of victim program in a classic side-channel attack:

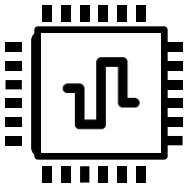
```
unsafe_instruction secret
```



slow μPATH

fast μPATH

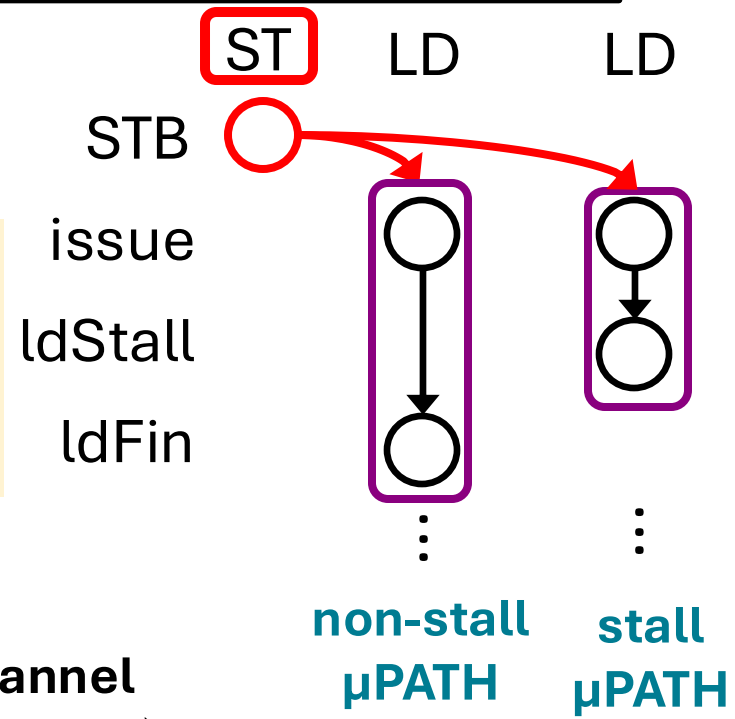
Explicit channel



Creates **operand-dependent hardware resource usage**, which manifests as **operand-dependent μpath variability** for **itself (←)** or some **other instruction (→)**

A more subtle pattern of victim program in a side-channel attack:

```
unsafe_instruction secret
unsafe_instruction public
```



non-stall μPATH

stall μPATH

Implicit channel

Hardware Side-Channel Defenses in Hardware or Software Require Characterizing a Microarchitecture's Side-Channels

Designer Inspection!

RTL Processor Design

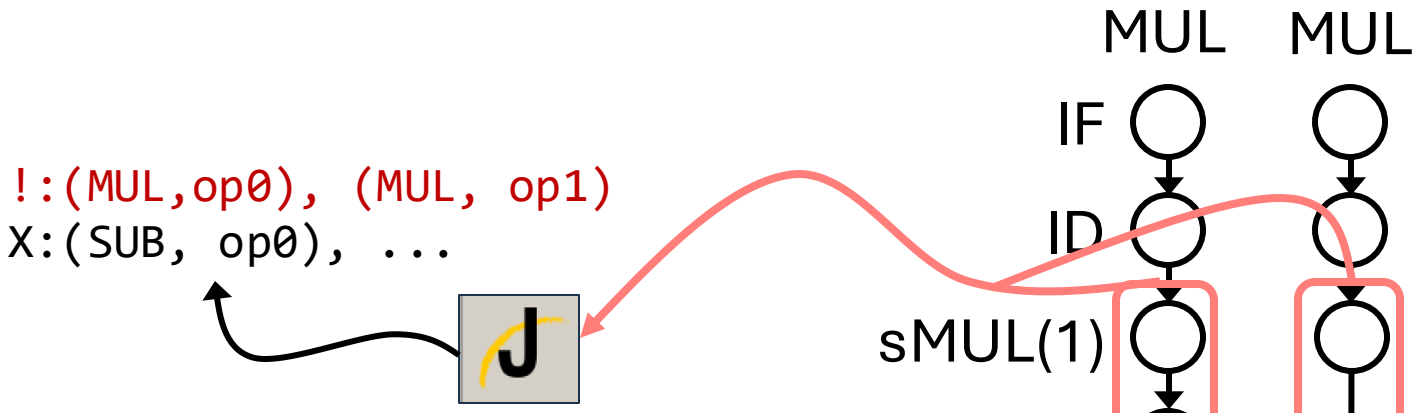
	Defenses	Microarchitectural Components
#1	**CT** [e.g., Cauligi+, SecDev'17], **SCT** [Mosier+, SP'24], **SpecShield** [Barber+, PACT'19], **ConTExt** [Schwarz+, NDSS'20]	unsafe_instruction **secret**
#2	**MI6** [Bourgeat+, MICRO'19]	Contention-based dynamic channels
		Static channels
#3	**OISA** [Yu+, NDSS'19]	Input-dependent arithmetic units
#4	**STT** [Yu+, MICRO'19] **SDO** [Yu+, ISCA'20] **SPT** [Choudhary, MICRO'21]	Explicit channel
		Implicit channel
		Implicit branches
		Prediction-based channels
		Resolution-based channels
#5	**SDO** [Yu+, ISCA'20]	Data-oblivious variants
#6	**Dolma** [Loughlin+, ISCA'21]	Variable-time micro-ops
		Contention-based dynamic channels
		Inducive micro-ops
		Resolvent micro-ops
		Prediction resolution points
		Persistent state modifying micro-ops

Roadmap

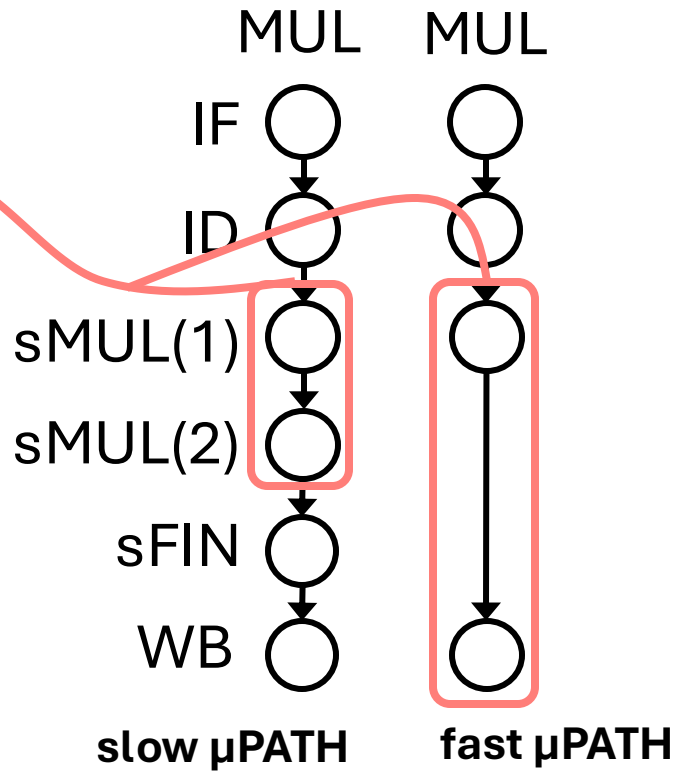
- **Background**: The Microarchitecture-μSPEC Model Verification Challenge
- **RTL2MμPATH**: Synthesizing ("Uncovering") All μPATHs per Instruction from Advanced SystemVerilog Processors
- **Key Insight**: μPATH Variability (>1 μPATH) is a Strong Indicator of a Hardware Side-Channel
- **SynthLC**: Synthesizing Formally Verified "Leakage Signatures" from SystemVerilog Processors

Foundational to the design of hardware side-channel defenses

SynthLC: Attributing μPATH Variability to Unsafe Instruction Operands and Synthesizing Formally-Verified Leakage Signatures



Ask a model checker if **every candidate (T, op) pair** contributes to the divergence using **symbolic information flow** analysis

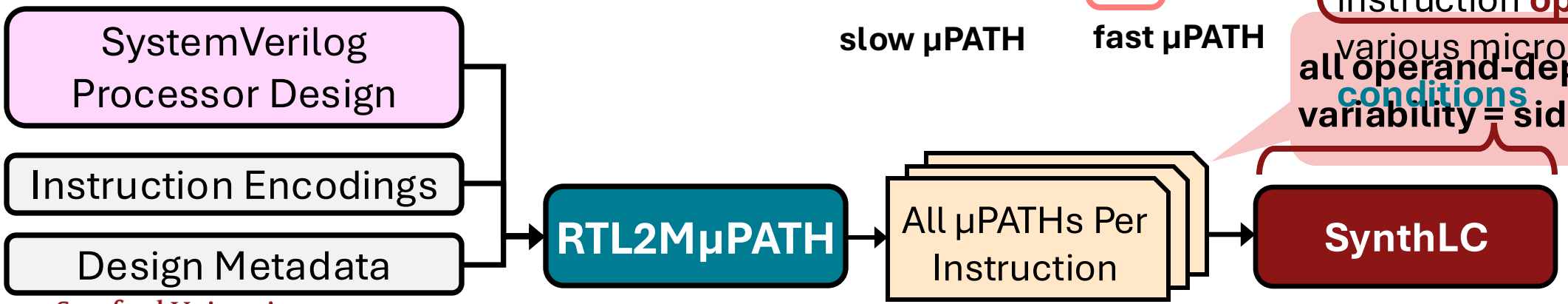


Defense #1

unsafe_instruction **secret**

Automatically attribute μPATH variability to instruction **operands** under various microarchitectural conditions

all operand-dependent μPATH variability = side-channel



Details in the paper!

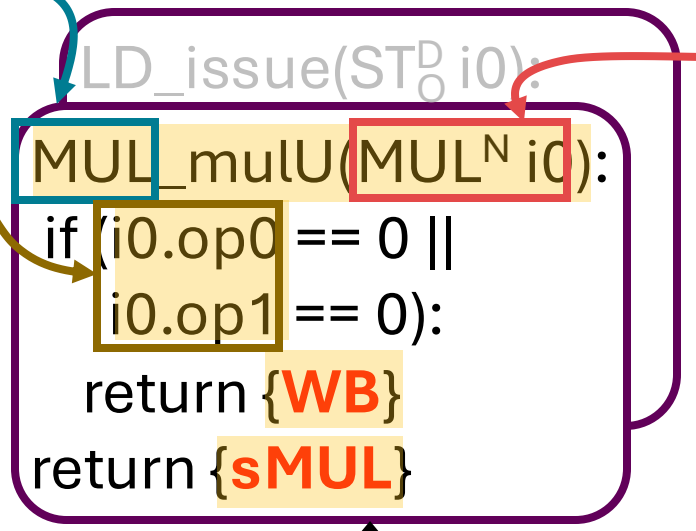
SynthLC Synthesizes a Complete Set of Verified Leakage Signatures…

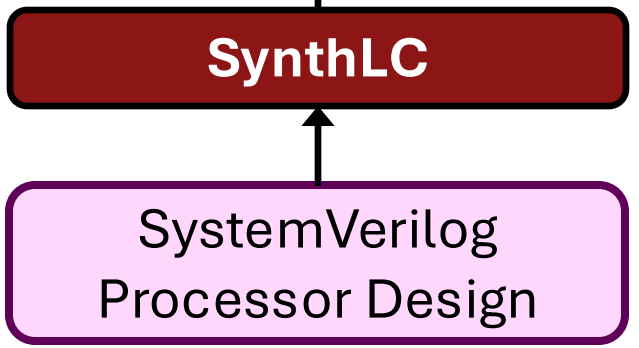
instruction exhibiting μPATH variability

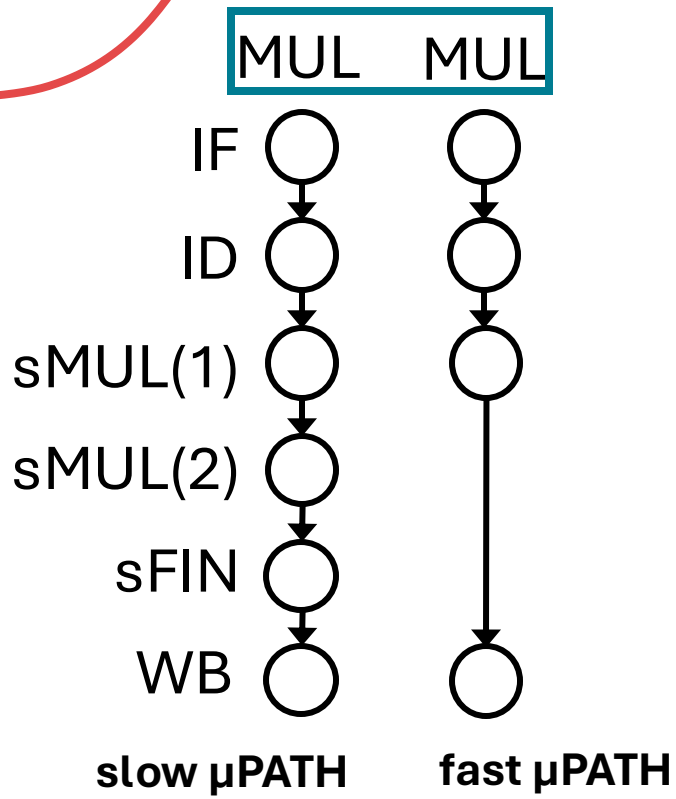
unsafe operands

unsafe instructions

Leakage Signature:
Characterizes a specific instance of **unsafe-instruction-operand-dependent μPATH variability** for a specific instruction, i.e., a side-channel.





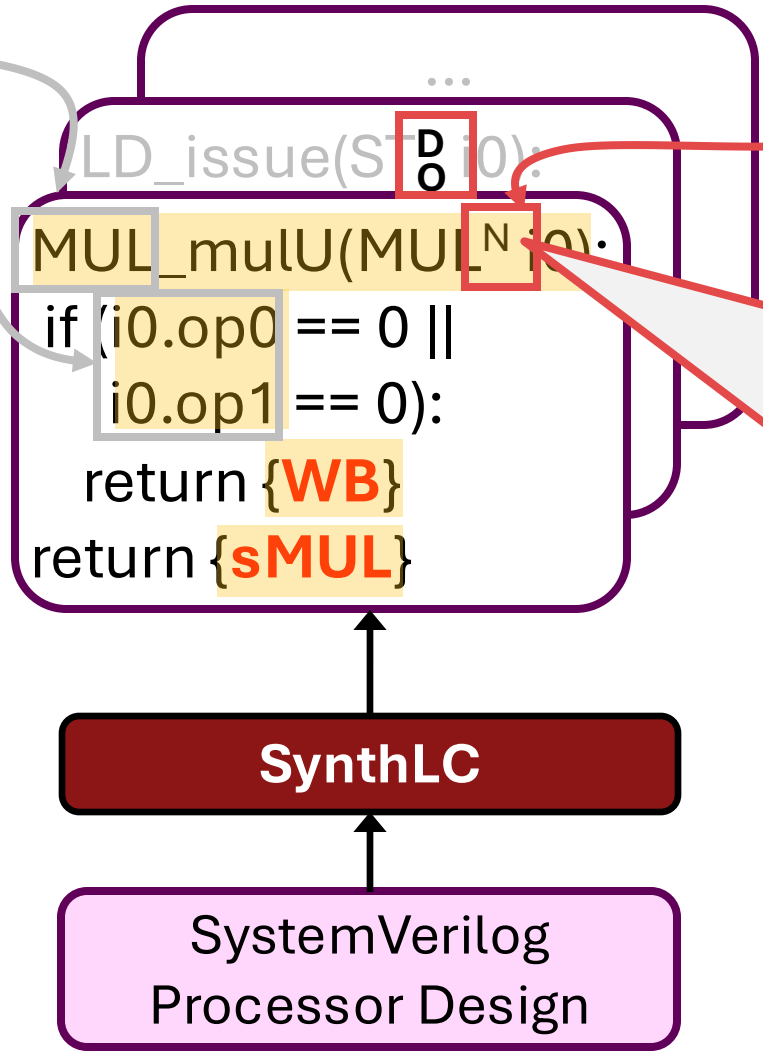


SynthLC Synthesizes a Complete Set of Verified Leakage Signatures…

instruction exhibiting μPATH variability

unsafe operands

Leakage Signature:
Characterizes a specific instance of **unsafe-instruction-operand-dependent μPATH variability** for a specific instruction, i.e., a side-channel.



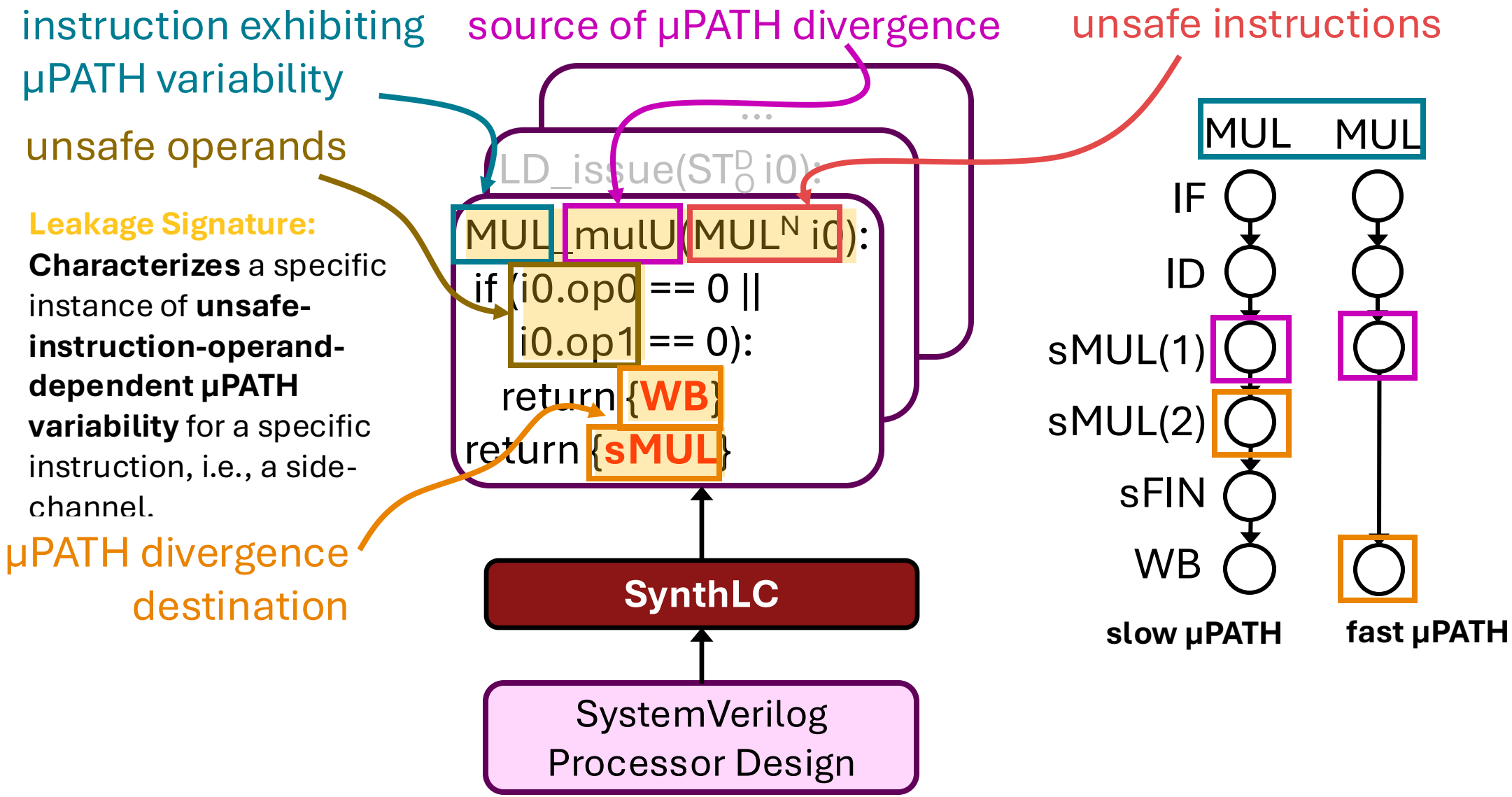
unsafe instructions

Details in the paper!

T^{N}	Intrinsic (same)
T^{D}_{O}	Dynamic (older)
T^{D}_{Y}	Dynamic (younger)
T^{S}	Static (older)

Runtime conditions for unsafe instruction to create μPATH variability

SynthLC Synthesizes a Complete Set of Verified Leakage Signatures…



Leakage Signatures: A Unifying Framework for Characterizing Hardware Side-Channels

source of μPATH divergence

unsafe instructions

unsafe operands

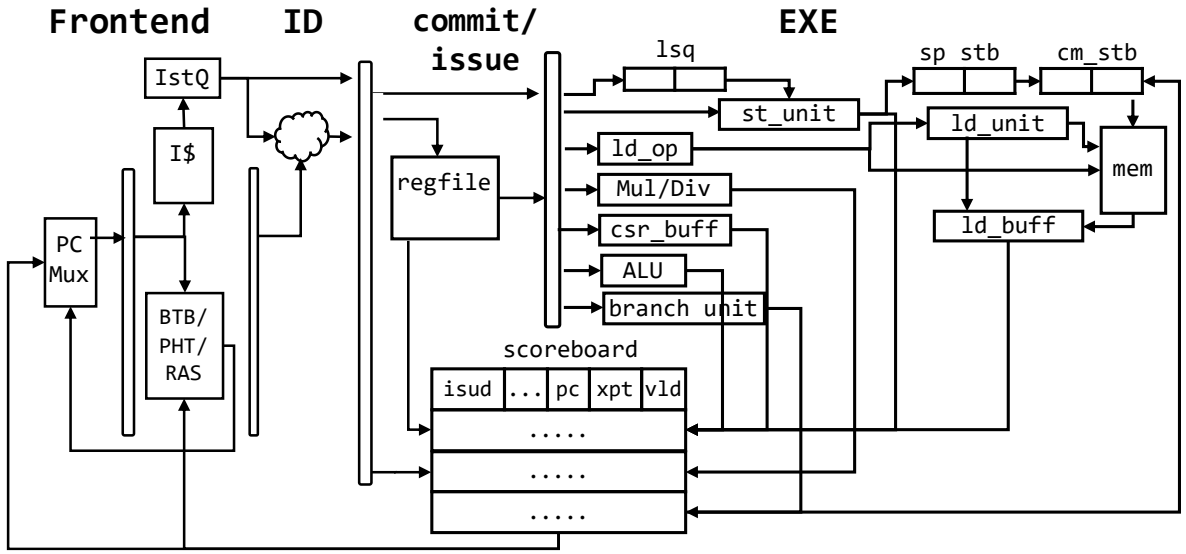
instruction exhibiting μPATH variability

	Defenses	Microarchitectural Components	μPATHs	Leakage Signature Components					
				P	src	T^N	T^D	T^S	args
#1	**CT** [e.g., Cauligi+, SecDev'17], **SCT** [Mosier+, SP'24], **SpecShield** [Barber+, PACT'19], **ConTExt** [Schwarz+, NDSS'20]	unsafe_instruction **secret**	-	-	-	✓	✓	✓	✓
#2	**MI6** [Bourgeat, MICRO'19]	Contention-based dynamic channels	-	✓	✓	✓	✓	-	-
		Static channels	-	✓	✓	-	-	✓	-
#3	**OISA** [Yu+, NDSS'19]	Input-dependent arithmetic units	-	-	✓	✓	-	-	✓
#4	**STT** [Yu+, MICRO'19] **SDO** [Yu+, ISCA'20] **SPT** [Choudhary, MICRO'21]	Explicit channels	-	✓	✓	✓	-	-	✓
		Implicit channels	-	✓	✓	-	✓	✓	✓
		Implicit branches	-	✓	-	-	✓	✓	✓
		Prediction-based channels	-	✓	✓	-	-	✓	✓
		Resolution-based channels	-	✓	✓	-	✓	-	✓
#5	**SDO** [Yu+, ISCA'20]	Data-oblivious variants	✓	-	-	✓	-	-	✓
#6	**Dolma** [Loughlin+, ISCA'21]	Variable-time micro-ops	-	-	-	✓	-	-	✓
		Contention-based dynamic channels	-	✓	✓	✓	✓	-	✓
		Inducive micro-ops	-	✓	-	-	✓	-	✓
		Resolvent micro-ops	-	-	-	-	✓	-	✓
		Prediction resolution points	-	✓	✓	-	✓	-	✓
		Persistent state modifying micro-ops	-	-	-	-	-	✓	✓

CVA6 Core and Cache Case Study

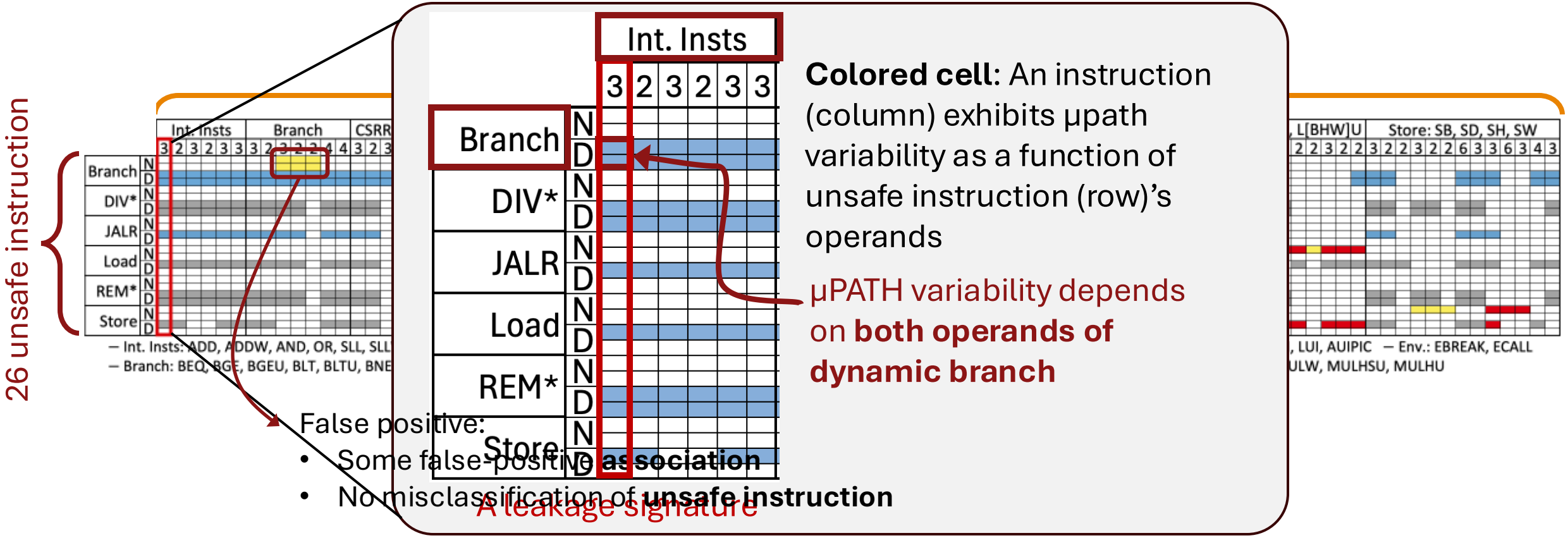
- Open-source RISC-V CVA6 processor
  - 64-bit, 6-stage, single-issue core
  - **Speculation** and limited **out-of-order write-back** with diverse functional units (ALU, LSU, Mul/Div, CSR buffer)
  - Write-through set-associative cache
- 72 instructions in **RV64I base ISA + M extension** (RV64IM)
- Synthesize leakage signatures separately (~modularity) from **Core** and **Data Cache** respectively

First formal side-channel analysis of a realistic processor cache!



CVA6 Core [Zaruba+, VLSI'19]

CVA6 Core: Results



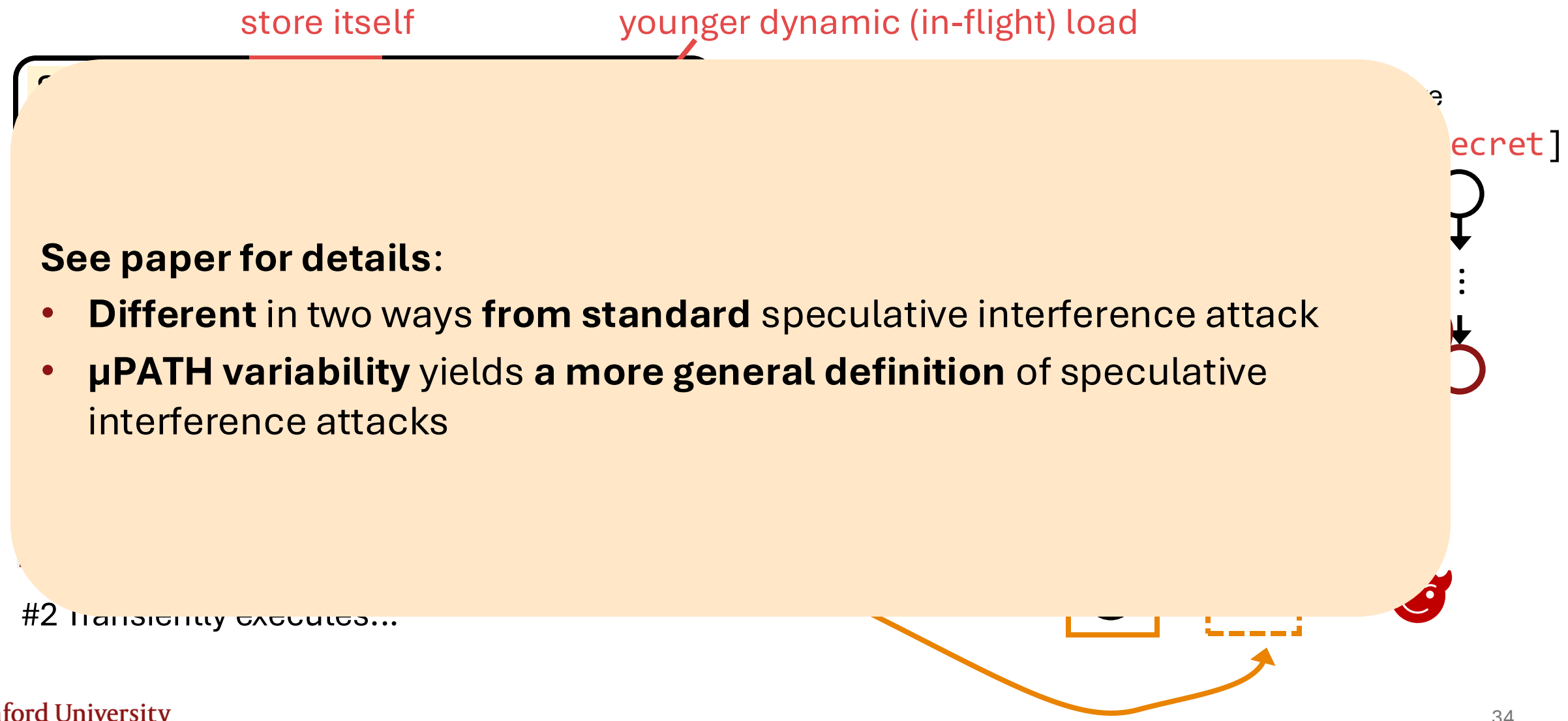
RTL2MμPATH:

- 124,459 properties, ~4 min per property
- ~16% undetermined

SynthLC:

- 30,774 properties, ~2 min per property
- ~14% undetermined

Leakage Signatures Captures Various Kinds of Side-Channels: A New Class of Speculative Interference Attack (SIA) [Behnia+, ASPLOS'21]



See paper for details:

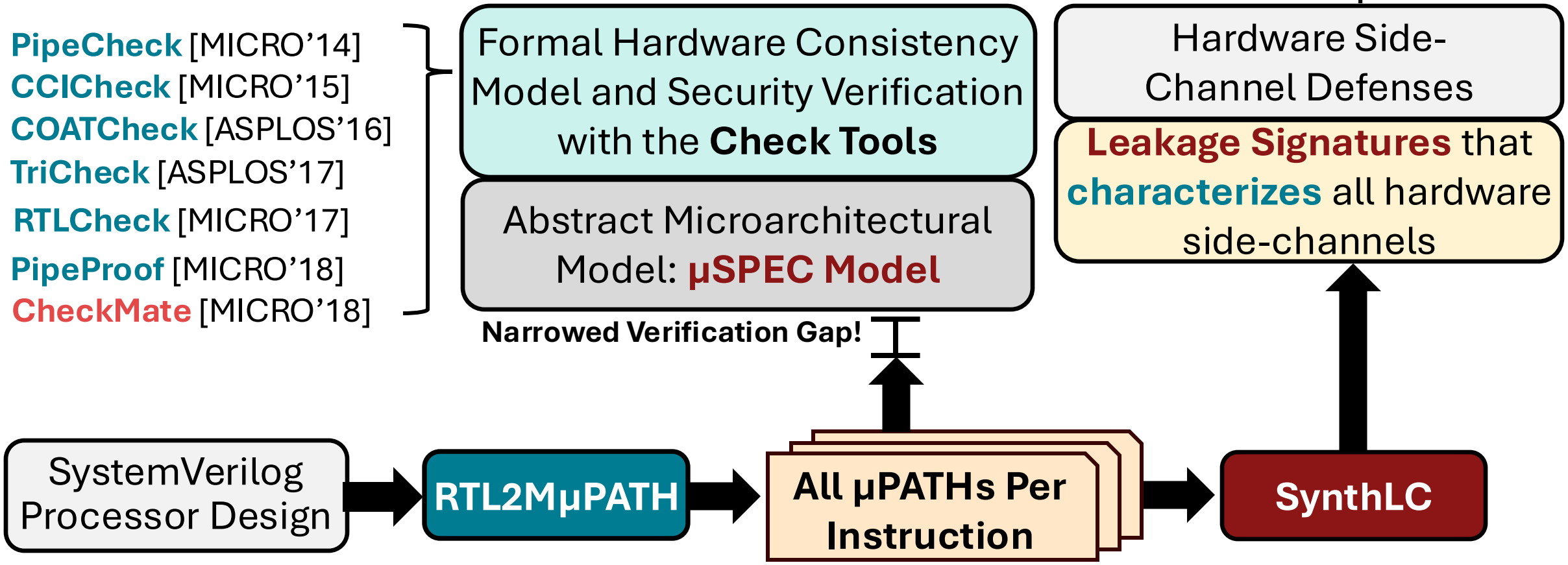
- **Different** in two ways **from standard** speculative interference attack
- **μPATH variability** yields **a more general definition** of speculative interference attacks

In the paper…

- Formalisms
  - Defining "stateless"/"stateful" channels and "active"/"passive" attacks using transponders
  - Replacement of 2-trace side-channel free property with many 1-trace properties and symbolic information taint tracking
- RTL2MμPATH and SynthLC Implementation and Usage
  - Required SystemVerilog design metadata
  - SystemVerilog Assertions (SVAs) generation from templates
- Evaluation
  - Side-channels discovered in CVA6 Core and Cache: store-to-load stalling, serial divider/remainder, jump and branches, channels in cache involving various structures
  - New functional bugs discovered with RTL2MμPATH for the CVA6 core

Takeaways

CT [e.g., Cauligi+, SecDev'17], **SCT** [Mosier+, SP'24], **SpecShield** [Barber+, PACT'19], **ConTExt** [Schwarz+, NDSS'20], **Dolma** [Loughlin+, ISCA'21], **OISA** [Yu+, NDSS'19], **MI6** [Bourgeat, MICRO'19], **STT** [Yu+, MICRO'19], **SDO** [Yu+, ISCA'20], **SPT** [Choudhary, MICRO'21]



Thank you!

yaohsiao@stanford.edu

https://github.com/yaohsiaopid/SynthLC

Artifacts Evaluated - Reusable, Available