

# Verification and Synthesis of Hardware-Software Leakage Contracts

Jan Reineke @  UNIVERSITÄT  
DES  
SAARLANDES

*Joint work with*

Gideon Mohr @ Universität des Saarlandes

Marco Guarnieri, Zilong Wang @ IMDEA Software

Klaus v. Gleissenthall @ VU Amsterdam

This work has received funding from an Intel Strategic Research Alliance (ISRA) and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101020415)"



# The Need for New HW/SW Contracts

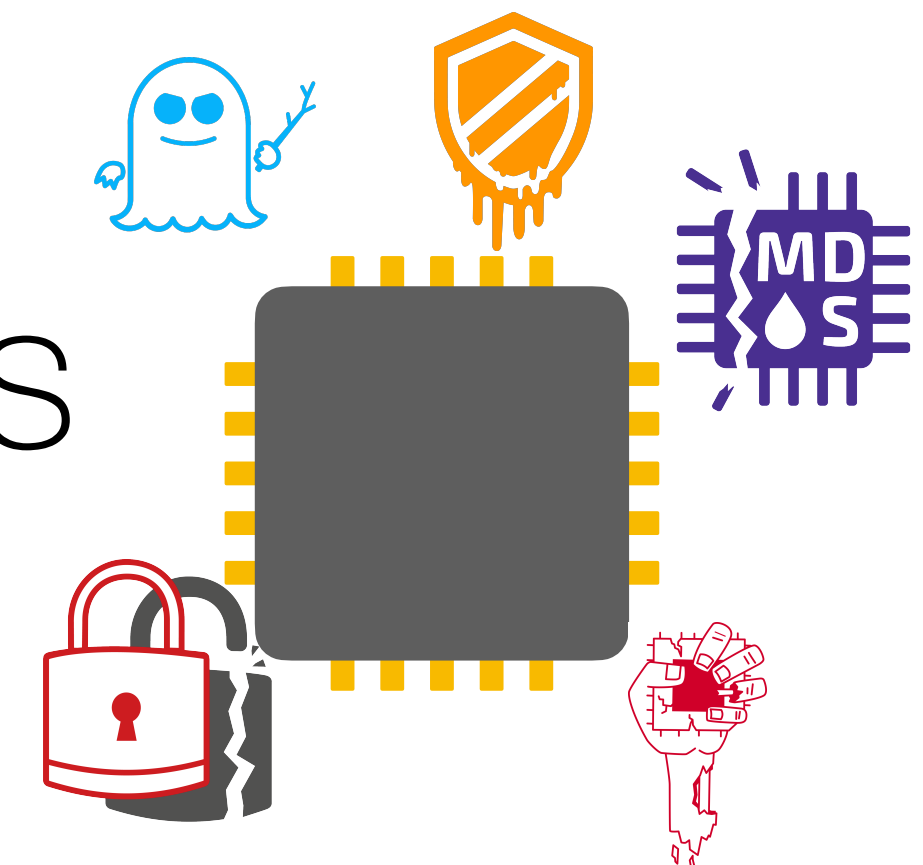
# Instruction Set Architectures - A Broken Abstraction

**Impossible** to program securely

**Instruction Set Architecture**

**No guarantees  
about side channels**

**Admit insecure** microarchitectures



# ***Are these programs vulnerable to timing attacks?***

```
if (secret)
    wait 1 second
else
    wait 100 seconds
return 42
```

Yes, on any microarchitecture.

***Are these programs vulnerable to timing attacks?***

```
return input * secret
```

It depends.  
Does multiplication have  
data-dependent timing?

# ***Are these programs vulnerable to timing attacks?***

```
if (secret)
    return 42
else
    return 23
```

It depends.  
Yes on most pipelined processors.

# ***Are these programs vulnerable to timing attacks?***

```
nirvana = Mem[secret]  
return Mem[42]
```

It depends.

Yes on processors with a cache.

# ***Are these programs vulnerable to timing attacks?***

```
assert(x > A.size)
if (x < A.size)
    y = A[x]
    z = B[y]
```

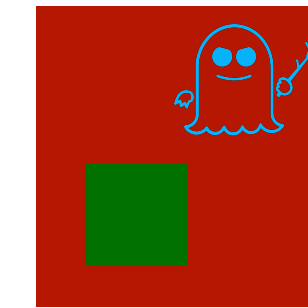
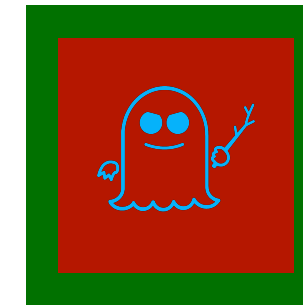
Leaks value of A[x] via cache  
on Intel/AMD processors 2010-2018.

# Leakage Contracts

**Enable secure programming:**

constant-time cryptographic code

sandboxing untrusted code



**HW/SW Leakage Contract**



**Capture side-channel leakage**

**Admit secure microarchitectures**

# Contract Verification and Synthesis



**Enable secure programming:**

constant-time cryptographic code  
sandboxing untrusted code

**HW/SW Leakage Contract**



**Capture side-channel leakage**



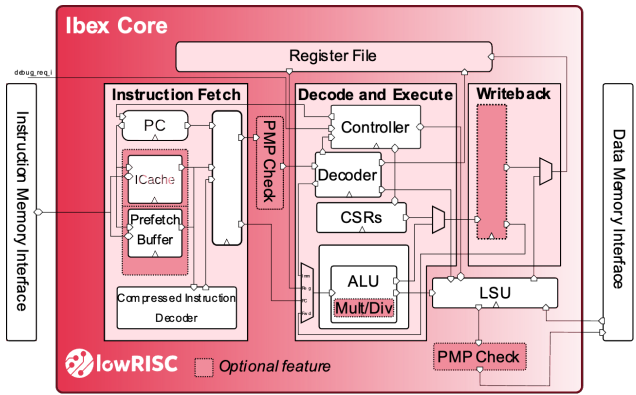
**Contract Synthesis**

**Features**

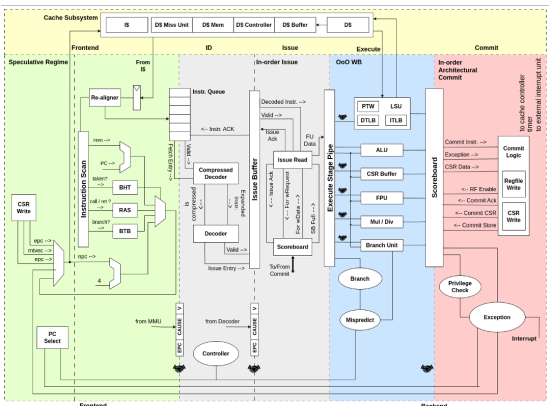
RISC-V Open-Source Cores



DarkRISCV



Ibex



CVA6

# Our work in this context

Guarnieri, Köpf, Reineke, and Vila

**Hardware-Software Contracts for Secure Speculation S&P 2021**

Wang, Mohr, Gleissenthall, Reineke, Guarnieri

**Specification and Verification of Side-channel Security for Open-source Processors via Leakage Contracts CCS 2023**

Mohr, Guarnieri, Reineke

**Synthesizing Hardware-Software Leakage Contracts for RISC-V Open-Source Processors DATE 2024**

Wang, Mohr, Gleissenthall, Reineke, Guarnieri

**Synthesis of Sound and Precise Leakage Contracts for Open-Source RISC-V Processors CCS 2025**

# Outline

**1. Contracts and Contract Templates**

**2. Verification**

**3. Synthesis**

# Outline

## **1. Contracts and Contract Templates**

## 2. Verification

## 3. Synthesis

# Contracts

**Contracts** specify which **program executions** a side-channel adversary **cannot distinguish**

## Contract

ISA

+

Observations

Captures how  
program is executed

What leaks  
about an execution

# An Example Contract

Composition of  
Contract Atoms

“For every multiplication the second operand leaks and  
for every memory access the memory address leaks.”

Condition on  
ISA State

Leakage of Part  
of ISA State

# An Example Contract

“For every multiplication the second operand leaks *and*  
for every memory access the memory address leaks.”

```
return input * secret
```

**vulnerable**

```
return secret * input
```

**not vulnerable**

```
if (secret)
    return 42
else
    return 23
```

**not vulnerable**

```
nirvana = Mem[secret]
return Mem[42]
```

**vulnerable**

# Contract Atoms - What could possibly leak (non-speculatively)?

mul a0, t0, t1

$a0 \leftarrow t0 * t1$

Instruction Register names  
and values

lw a1, -4(t2)

$a1 \leftarrow \text{mem}[t2 + (-4)]$

Memory addresses  
and values

Leakage sources:

- Instruction type
- Register names
- Register values
- Immediate values
- Memory addresses
- Memory values

# Contract Atoms - What could possibly leak (non-speculatively)?

Contract Atom =

Condition based on  
Instruction Type

x

Applicable  
Leakage Sources

→ Several hundred contract atoms for RISC-V I+M

Contract = ISA model + subset of contract atoms

# Contract Satisfaction

For any pair of programs and inputs  $(P_1, I_1)$  and  $(P_2, I_2)$ :

$$\text{Adv}(\text{HW}(P_1, I_1)) \neq \text{Adv}(\text{HW}(P_2, I_2)) \implies \text{Obs}(\text{ISA}(P_1, I_1)) \neq \text{Obs}(\text{ISA}(P_2, I_2))$$

Adversary  
distinguishable

Cycle-accurate  
processor model  
+  
Adversary model

Contract  
distinguishable

ISA model  
+  
Contract atoms

# Contract Satisfaction

For any pair of programs and inputs  $(P_1, I_1)$  and  $(P_2, I_2)$ :

$$\text{Obs}(\text{ISA}(P_1, I_1)) = \text{Obs}(\text{ISA}(P_2, I_2)) \implies \text{Adv}(\text{HW}(P_1, I_1)) = \text{Adv}(\text{HW}(P_2, I_2))$$



Contract  
indistinguishable



Adversary  
indistinguishable

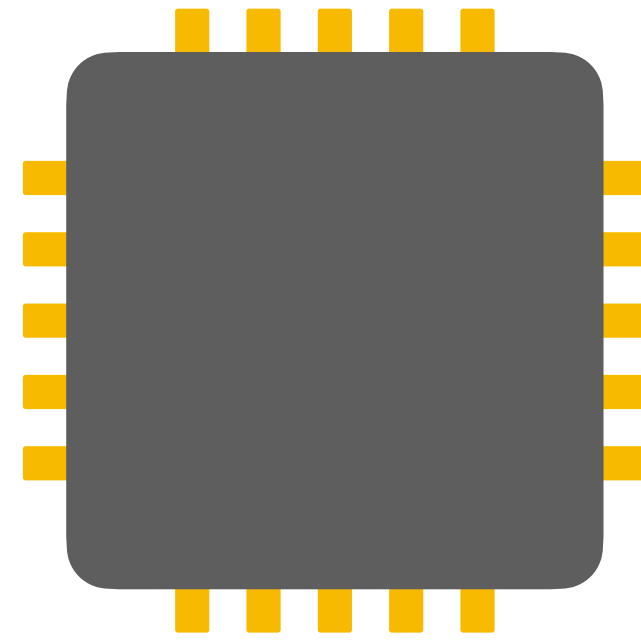
# Outline

1. Contracts and Contract Templates

**2. Verification**

3. Synthesis

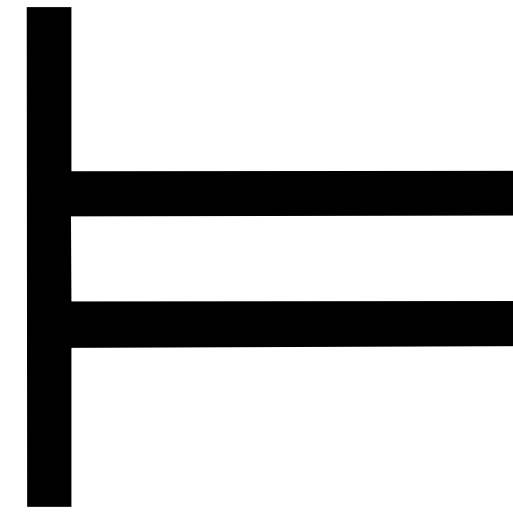
# Contracts for Real-world ISAs + CPUs



## Microarchitecture

*RTL designs  
+ Adversary model*

Open-source  
RISC-V cores



*Automatic proof*

SMT solvers  
+ Invariant  
inference



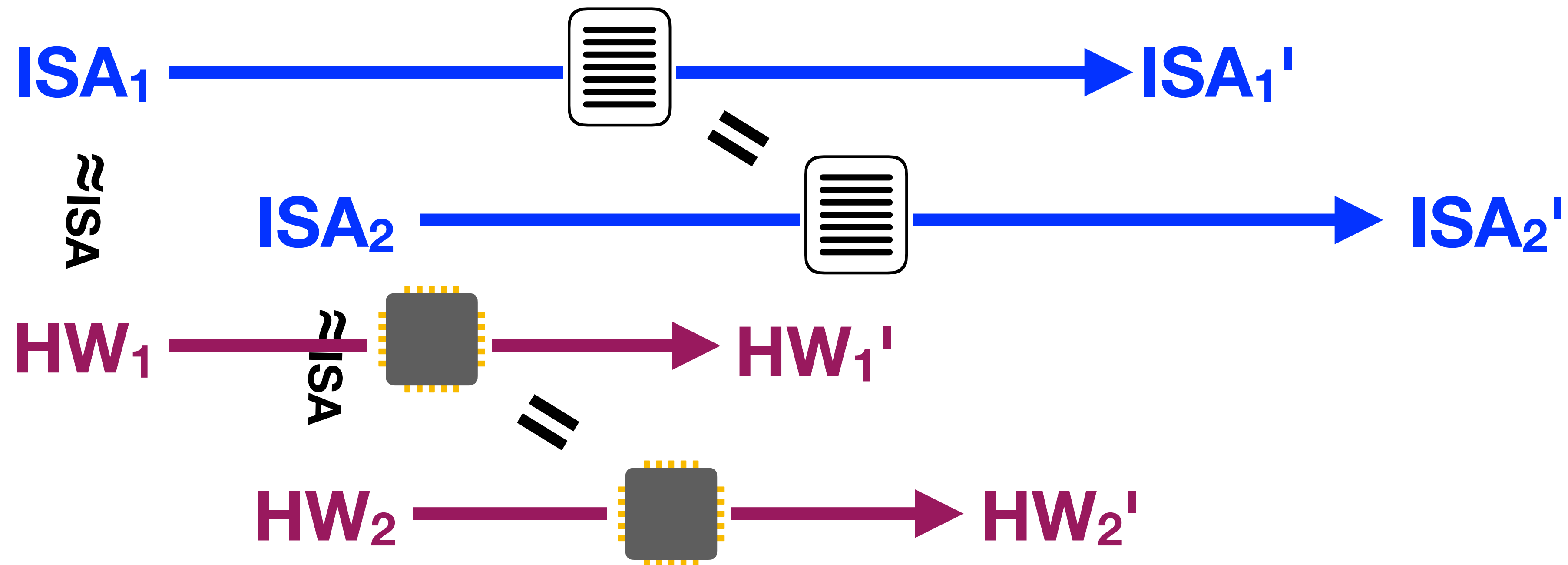
## Contract

*ISA spec.  
+ Contract atoms*

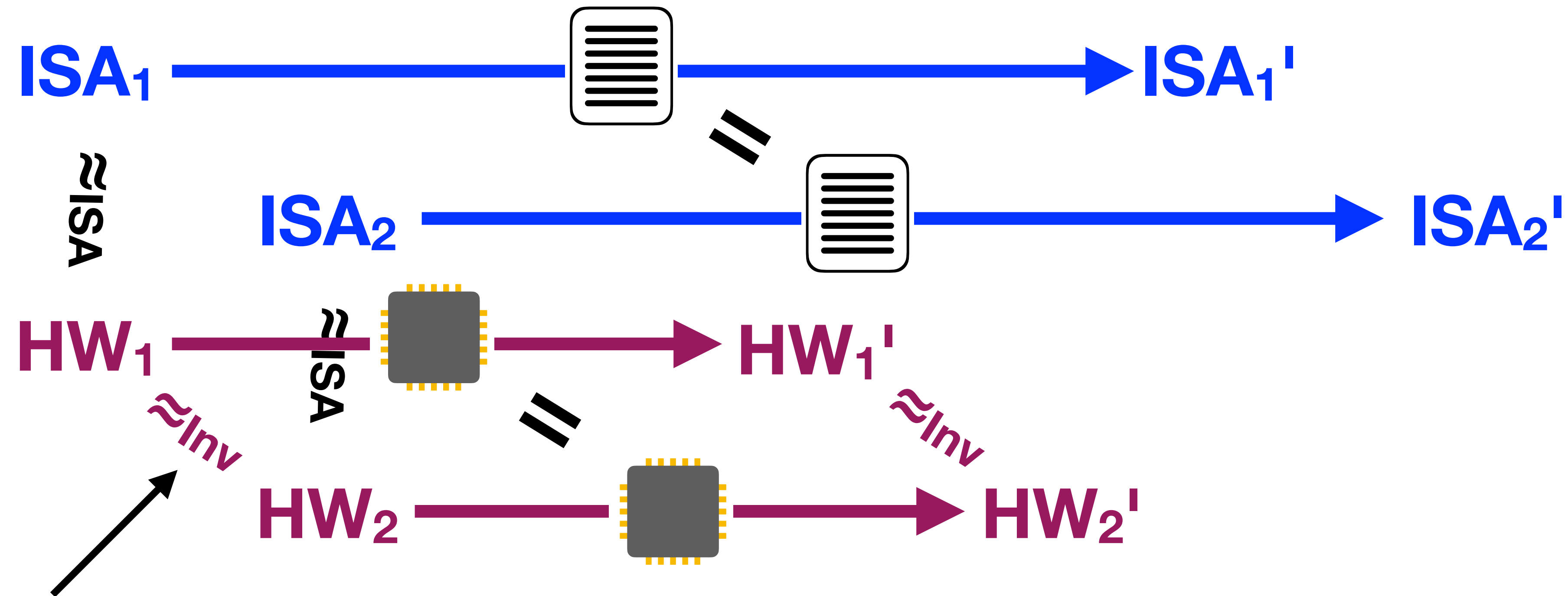
Separate functional  
and security concerns

# Bounded Verification: 4-way Product Circuit

**Contract Satisfaction:** For any pair of programs and inputs  $(P_1, I_1)$  and  $(P_2, I_2)$ :

$$\text{DocumentIcon} (P_1, I_1) = \text{DocumentIcon} (P_2, I_2) \implies \text{ChipIcon} (P_1, I_1) = \text{ChipIcon} (P_2, I_2)$$


# Unbounded Verification: Product Circuit + Induction



**Relational  
Inductive  
Invariant**

Works on small cores, e.g. the  
2-stage RISC-V Sodor

Scalability limited due to  
4-way product circuit

... synthesized using Houdini algorithm

(Flanagan+Leino: Houdini, an Annotation Assistant for ESC/Java, Formal Methods Europe 2001)

***Key Insight:***

# Decoupling Functional and Security Concerns

**ISA Satisfaction**

← **Functional** correctness

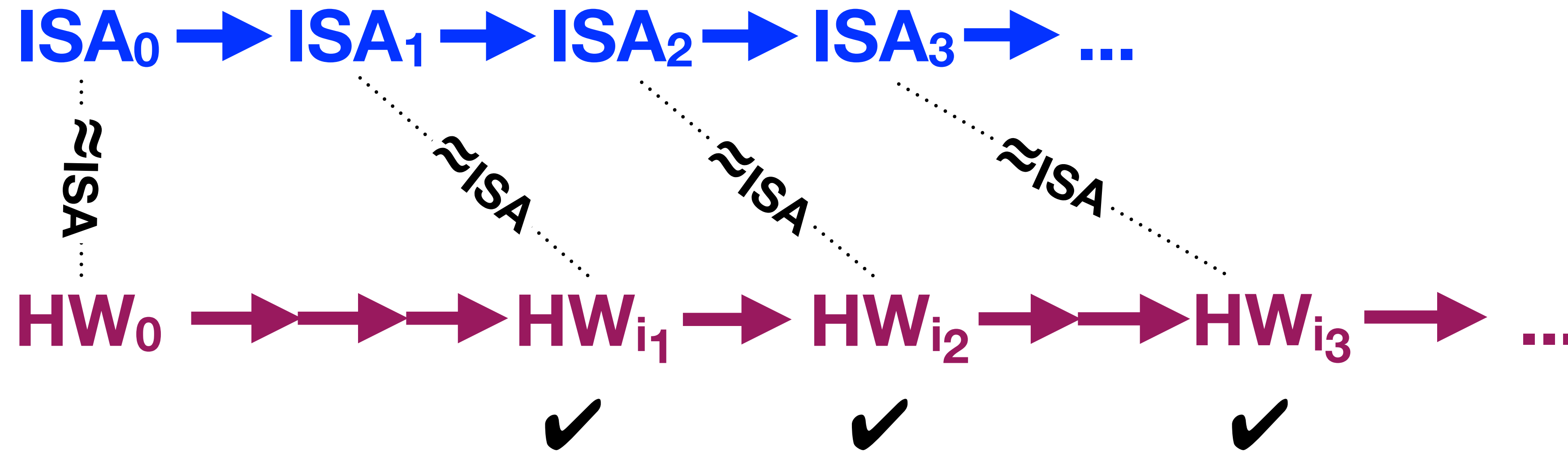
+ **Leakage Satisfaction**

← **Security** correctness  
← **security** correctness

**Contract Satisfaction**



# ISA Satisfaction



The ISA trace is embedded in the HW trace.

# From Contract to Leakage Satisfaction

## Contract Satisfaction

$$\text{Obs}(\text{ISA}(P_1, I_1)) = \text{Obs}(\text{ISA}(P_2, I_2)) \implies \text{Adv}(\text{HW}(P_1, I_1)) = \text{Adv}(\text{HW}(P_2, I_2))$$

## Leakage Satisfaction

$$\text{Obs}(\mathbf{HW}_r(P_1, I_1)) = \text{Obs}(\mathbf{HW}_r(P_2, I_2)) \Rightarrow \text{Adv}(\text{HW}(P_1, I_1)) = \text{Adv}(\text{HW}(P_2, I_2))$$

## Decoupling Theorem

**ISA Satisfaction**  $\implies$

(**Contract Satisfaction**  $\iff$  **Leakage Satisfaction**)

# Verifying Leakage Satisfaction

## Leakage Satisfaction

$$\text{Obs}(\mathbf{HW}_r(P_1, I_1)) = \text{Obs}(\mathbf{HW}_r(P_2, I_2)) \Rightarrow \text{Adv}(\text{HW}(P_1, I_1)) = \text{Adv}(\text{HW}(P_2, I_2))$$

Can be checked with 2-way product circuit + Induction

Verified DarkRISCV, Sodor, Ibex

# Evaluation

Adversary observes when instructions retire

| <i>Processor</i>     | <i>Strongest contract</i>                | <i>Verification time (in minutes)</i> |                                              |
|----------------------|------------------------------------------|---------------------------------------|----------------------------------------------|
| <b>DarkRISCv-2</b>   | <b>I</b>                                 | 7.2                                   | Aligned: 3 cycles<br>Misaligned: 5 cycles    |
| <b>DarkRISCv-3</b>   | <b>I</b>                                 | 11.1                                  |                                              |
| <b>Sodor-2</b>       | <b>I+B</b>                               | 97.8                                  | Cache hit vs miss                            |
| <b>Ibex-small</b>    | <b>I+B+O<sub>d</sub>+A</b>               | 1479.1                                | <b>Scalability:</b><br>Modular verification? |
| <b>Ibex-cache</b>    | <b>I+B+O<sub>d</sub>+M</b>               | 1396.7                                |                                              |
| <b>Ibex-mult-div</b> | <b>I+B+O<sub>d</sub>+O<sub>m</sub>+A</b> | 1291.9                                |                                              |

**I**: program counter and instruction  
**B**: outcome of branches  
**A**: word-alignment of memory addresses

**M**: memory access  
**O<sub>d</sub>**: divisor is/isn't 0  
**O<sub>m</sub>**: operands of multiplications  
multiplication in  $\log_2(\text{operand})$  cycles

# Outline

1. Contracts and Contract Templates

2. Verification

**3. Synthesis**

# Our work in this context

Guarnieri, Köpf, Reineke, and Vila

**Hardware-Software Contracts for Secure Speculation** S&P 2021

Wang, Mohr, Gleissenthall, Reineke, Guarnieri

**Specification and Verification of Side-channel Security for Open-source Processors via Leakage Contracts** CCS 2023

Mohr, Guarnieri, Reineke

**Synthesizing Hardware-Software Leakage Contracts for RISC-V Open-Source Processors** DATE 2024

Wang, Mohr, Gleissenthall, Reineke, Guarnieri

**Synthesis of Sound and Precise Leakage Contracts for Open-Source RISC-V Processors** CCS 2025

# Contract Synthesis: Goals

For a given processor, find contract from template s.t.:

## 1. Processor satisfies contract

For any pair of programs and inputs  $(P_1, I_1)$  and  $(P_2, I_2)$ :

$$\text{Processor}(P_1, I_1) \neq \text{Processor}(P_2, I_2) \implies \text{Contract}(P_1, I_1) \neq \text{Contract}(P_2, I_2)$$

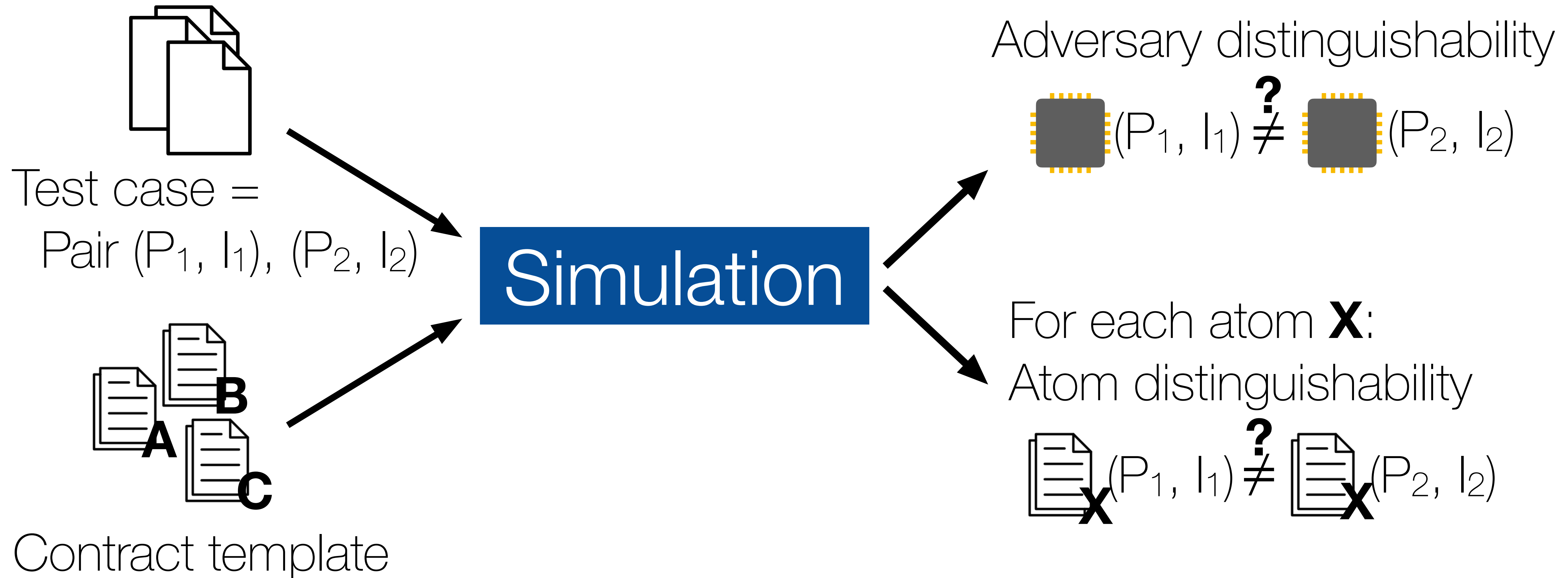
## 2. Contract is as precise as possible

As few programs and inputs  $(P_1, I_1)$  and  $(P_2, I_2)$  as possible s.t.:

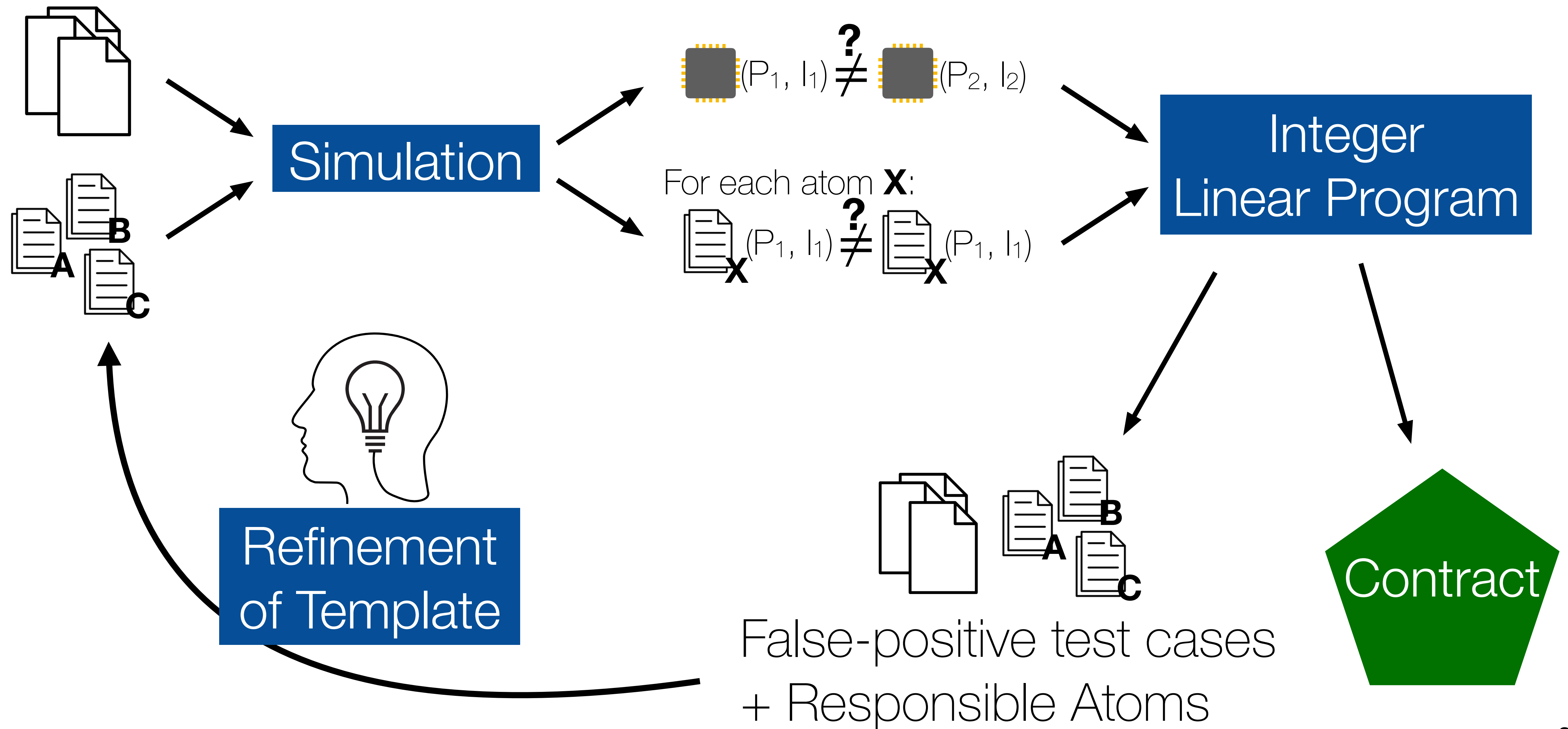
$$\text{Contract}(P_1, I_1) \neq \text{Contract}(P_2, I_2) \wedge \text{Processor}(P_1, I_1) = \text{Processor}(P_2, I_2)$$

# Contract Synthesis: Methodology (1/2)

Use test cases as proxy for contract satisfaction

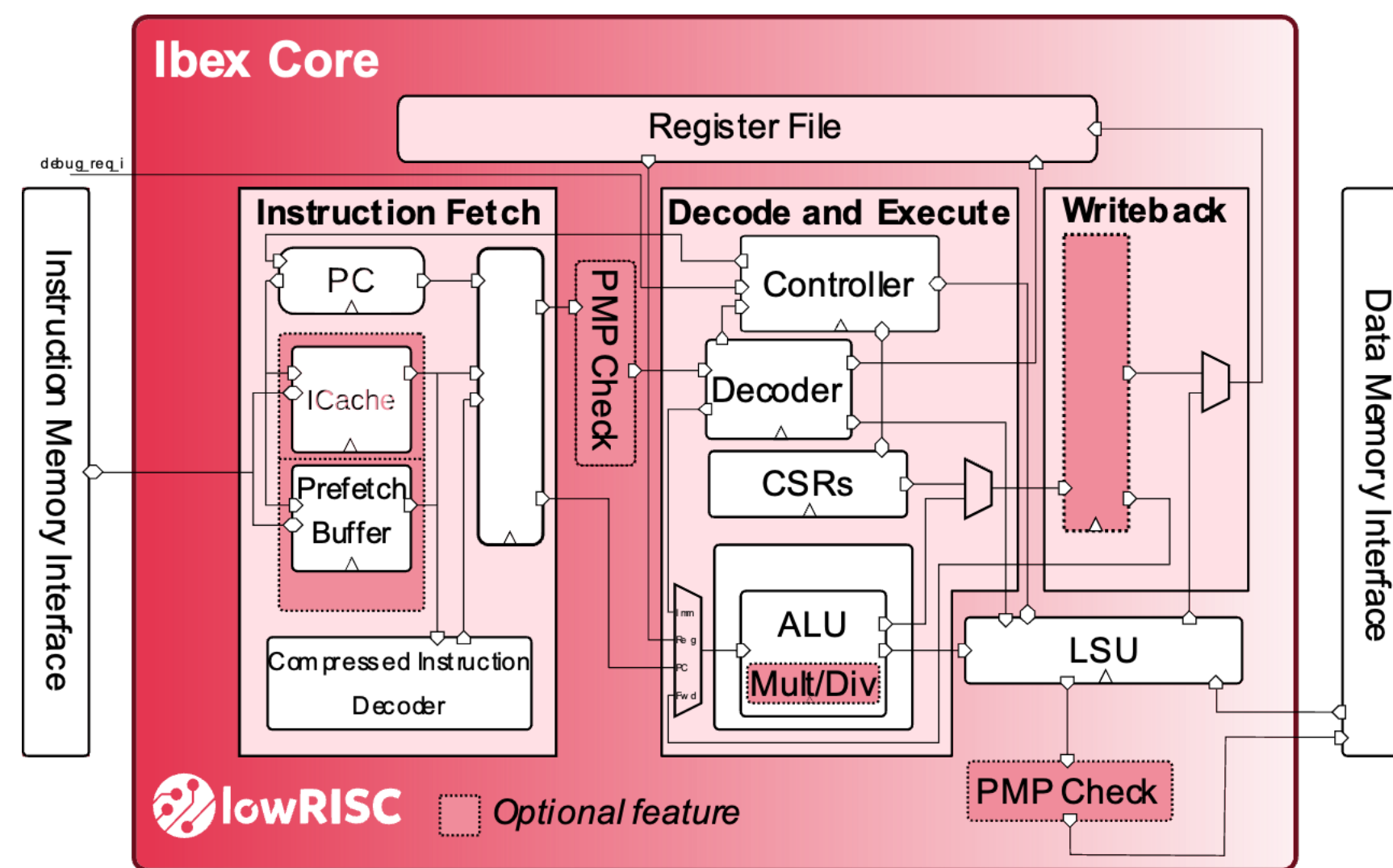


# Contract Synthesis: Methodology (2/2)

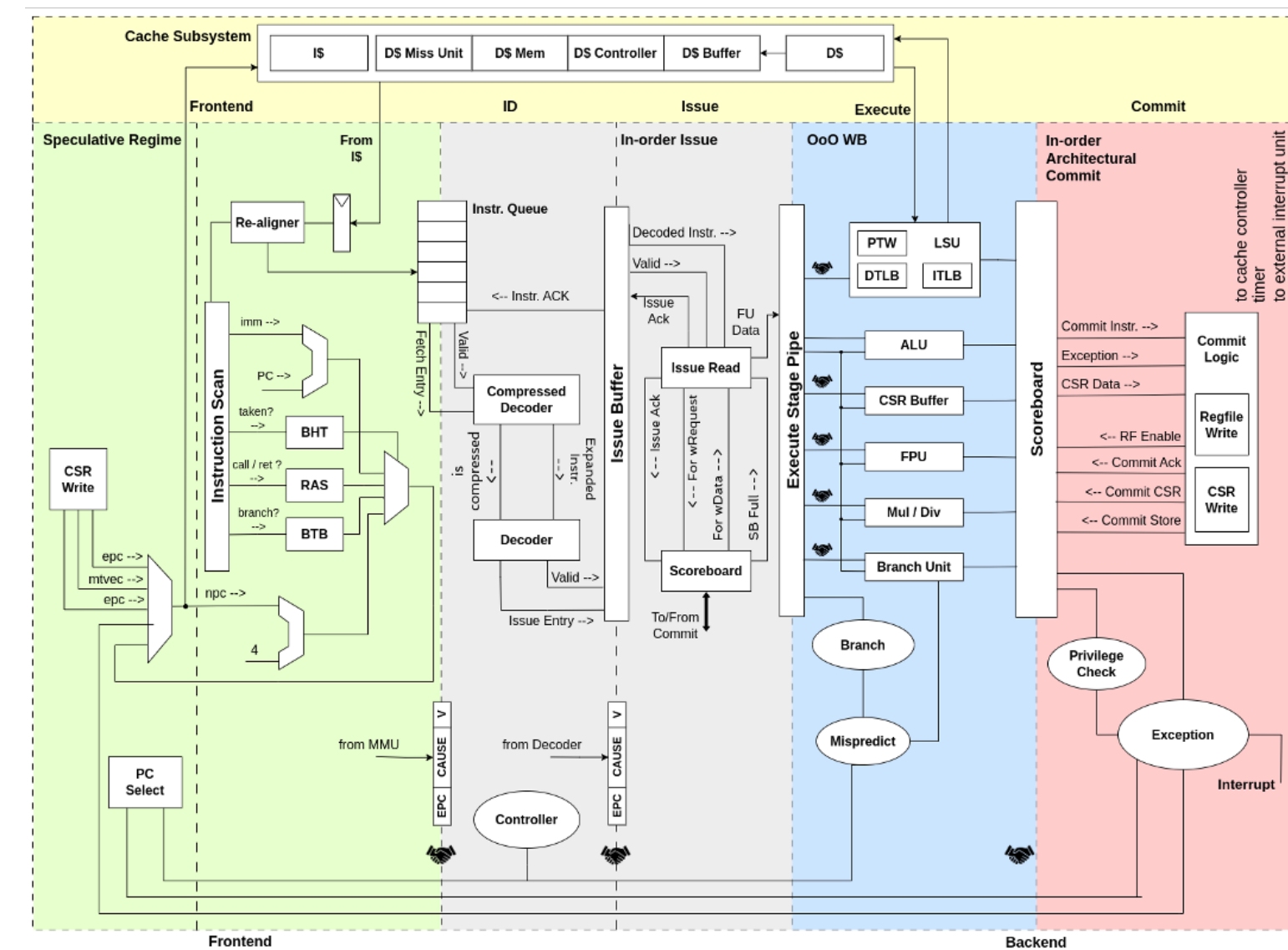


# Experimental Evaluation

Implemented using Icarus Verilog, Google OR-Tools, and RISC-V Formal Interface



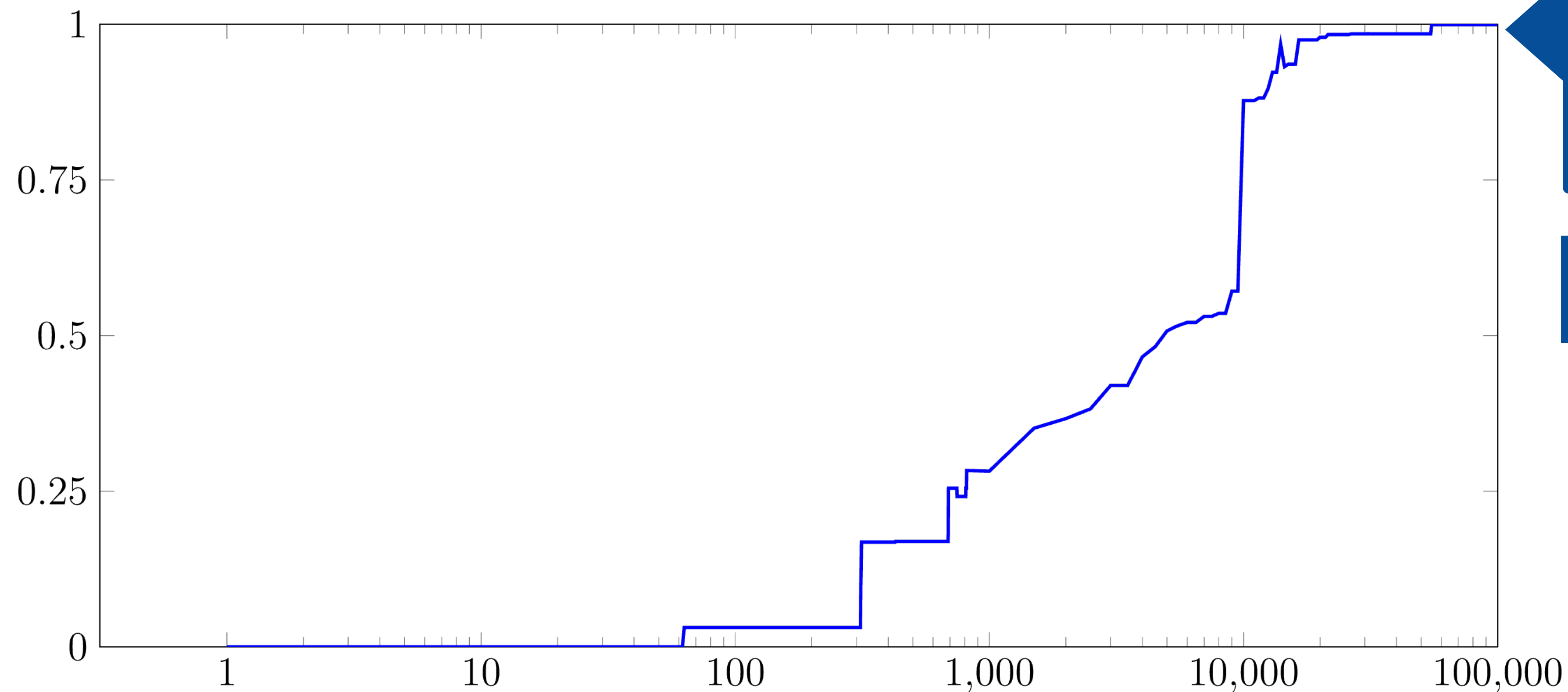
“**Ibex** is a production-quality open-source 32-bit RISC-V CPU core”



“**CVA6** is an application-class 6-stage RISC-V CPU capable of booting Linux”

# Performance on Ibex - Sensitivity

$$\text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$



Final sensitivity:  
99.94%

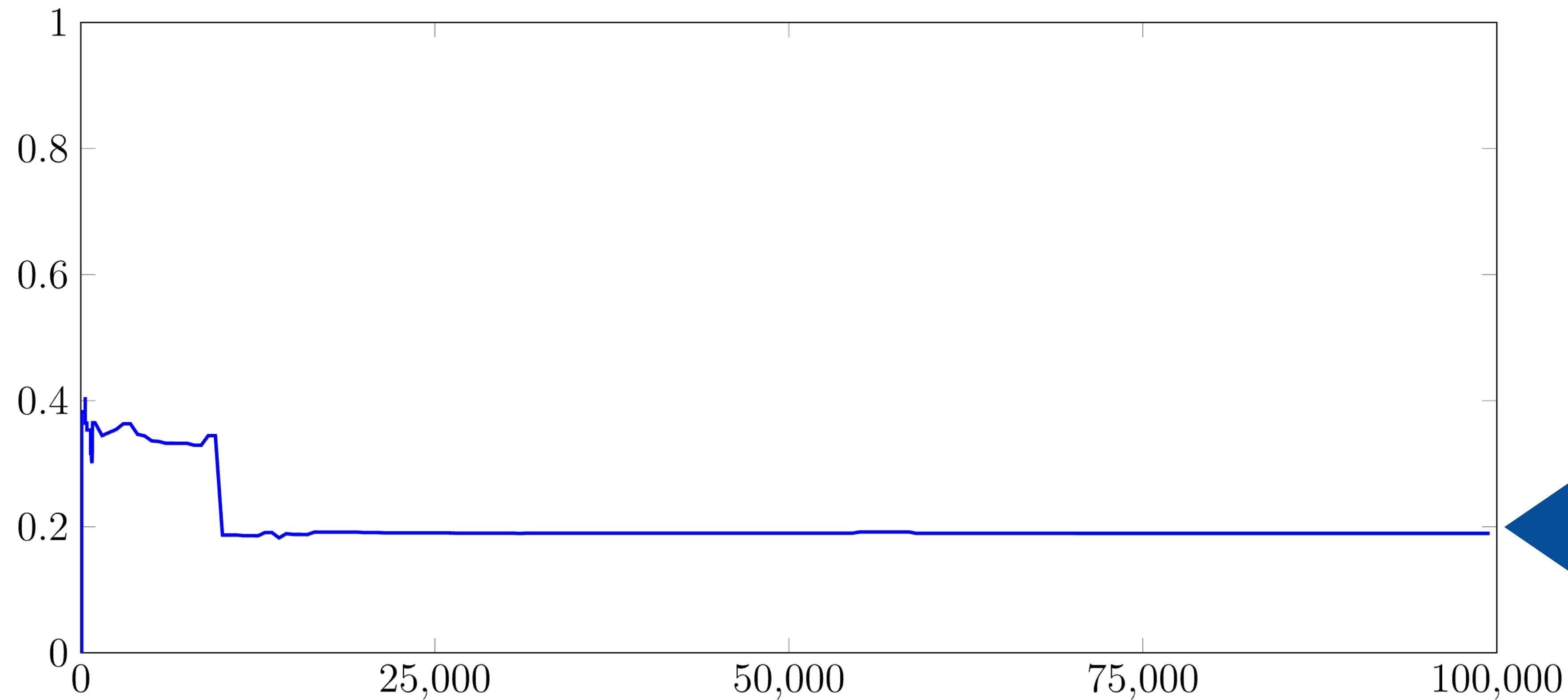
CCS 2025:

BMC to detect leaks

Sensitivity w.r.t. a set of 2,000,000 test cases

# Performance on Ibex - Precision

$$\textit{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$



Final precision:  
18.95%

Precision w.r.t. a set of 2,000,000 test cases

# Refining Contract Template - Ibex Memory Interface

```
li a0, 0x100      a0 ← 0x100  
lw a1, 0(a0)      a1 ← mem[a0]
```

---

Program 1

```
li a0, 0x102      a0 ← 0x102  
lw a1, 0(a0)      a1 ← mem[a0]
```

---

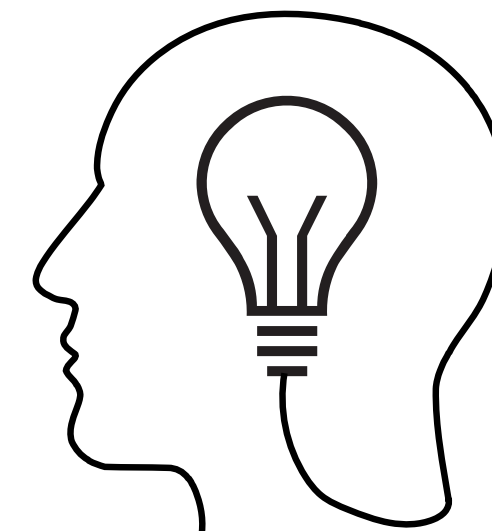
Program 2

Slower!

Address is not aligned

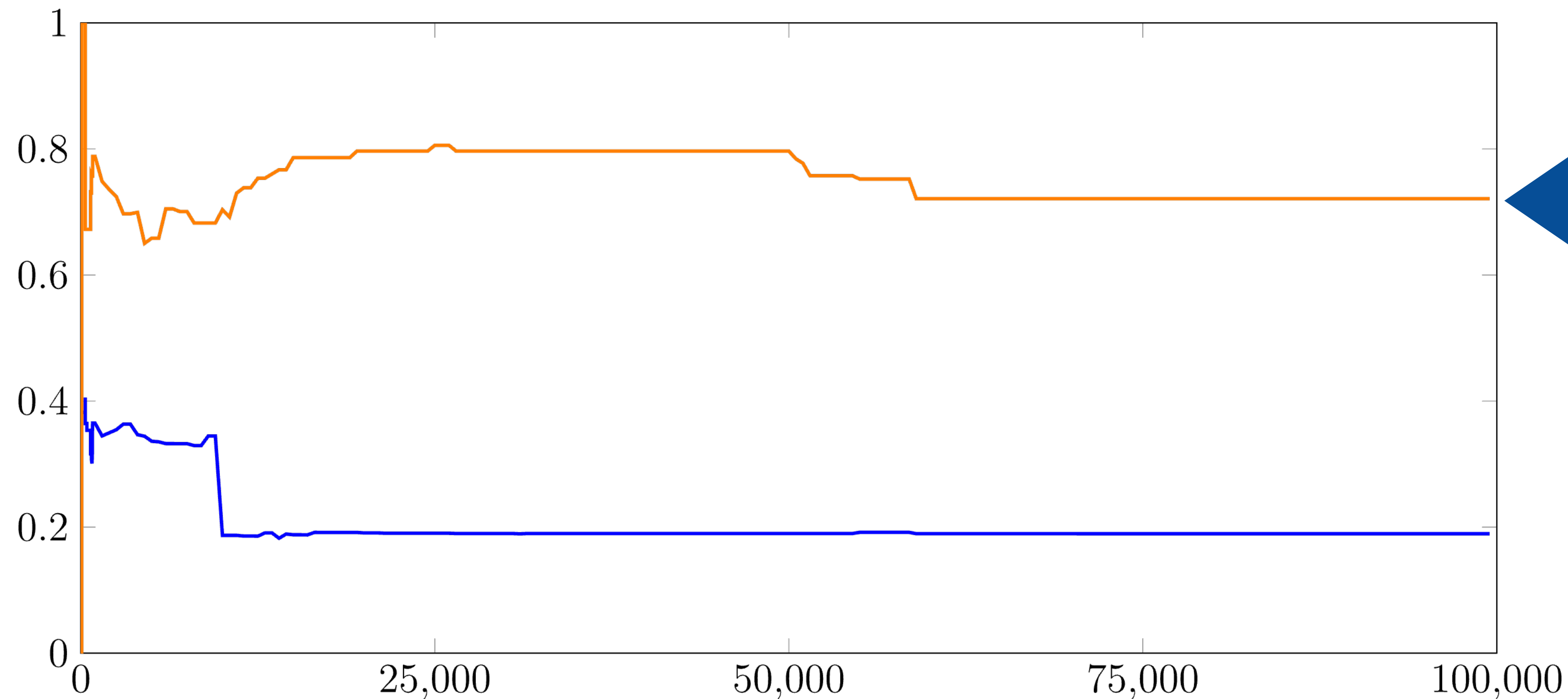
Refinement of template to capture:

- alignment of accesses
- branch outcomes
- data dependencies



# Refined Contract Template - Precision

$$\textit{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$



Final precision:  
72.11%

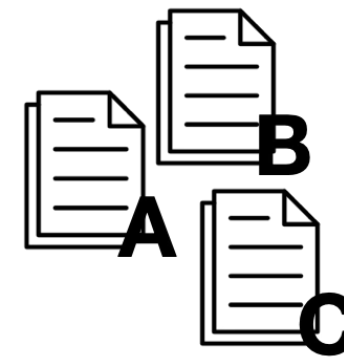
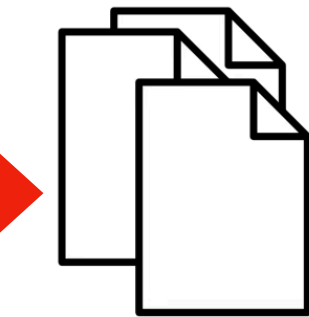
CCS 2025:

- capture MUL/DIV value leakage
- 99.9% precision

Precision w.r.t. a set of 2,000,000 test cases

# Contract Synthesis: Challenges

*Fuzzing:*  
New test cases?



Simulation

$$\text{Chip}(P_1, I_1) \stackrel{?}{\neq} \text{Chip}(P_2, I_2)$$

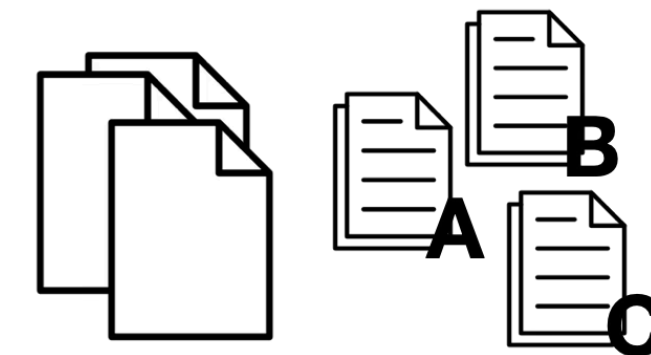
For each atom  $\mathbf{x}$ :

$$\text{Chip}_{\mathbf{x}}(P_1, I_1) \stackrel{?}{\neq} \text{Chip}_{\mathbf{x}}(P_1, I_1)$$

Integer  
Linear Program



Refinement  
of Template



False-positive test cases  
+ Responsible Atoms

Contract

*Synthesis:*  
Generate new atoms?

CCS 2025:

*Verification:*  
Sound contract?

# Open Challenges



## Enable secure programming:

constant-time cryptographic code  
sandboxing untrusted code

*Contract-aware  
secure compilation?*

*Scalable contract-aware  
static analysis?*

## HW/SW Leakage Contract



## Admit secure microarchitectures

*Richer contract notions?  
OS isolation guarantees?*

*Scalable contract  
verification + synthesis?*

*Contract-aware  
microarchitecture design?*