Interplay of Efficient Model Checking and Secure Processor Design: A Case Study on Secure Speculation

Tingzhen Dong*, Kunpeng Wang*, Yuheng Yang, Yu-Wei Fan,
Qinhan Tan, Thomas Bourgeat, Sharad Malik, Mengjia Yan

This Paper

Formally verify **security** on complex **out-of-order CPUs**

This Paper

Formally verify **security** on complex **out-of-order CPUs**

1. **Decoupling** security and functionality can scale verification
2. New abstraction primitive **HUF**

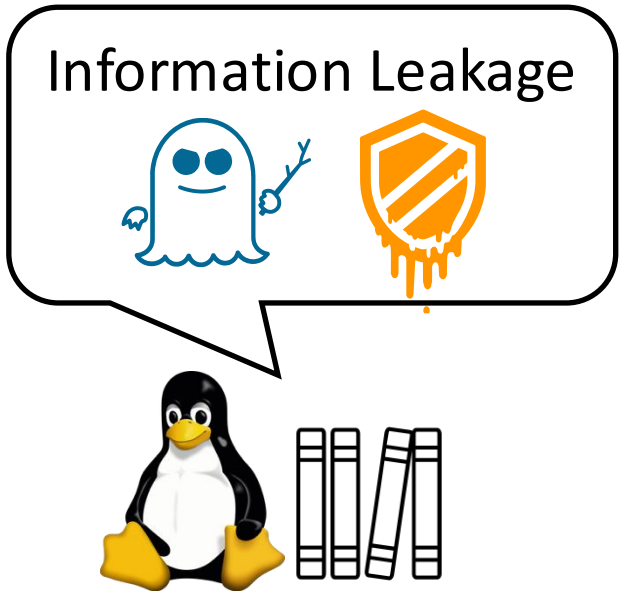
This Paper

Formally verify **security** on complex **out-of-order CPUs**

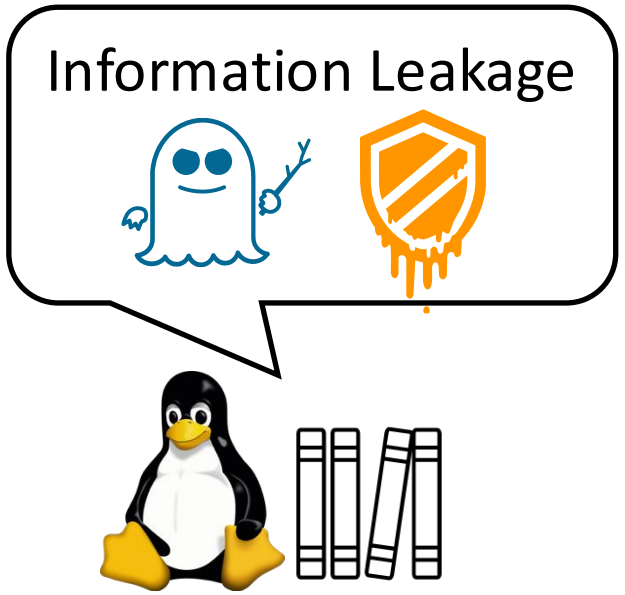
1. **Decoupling** security and functionality can scale verification
2. New abstraction primitive **HUF**

For the first time, we scale **model checking** to verifying secure speculation on **MegaBOOM**

Timing Side-Channel Security

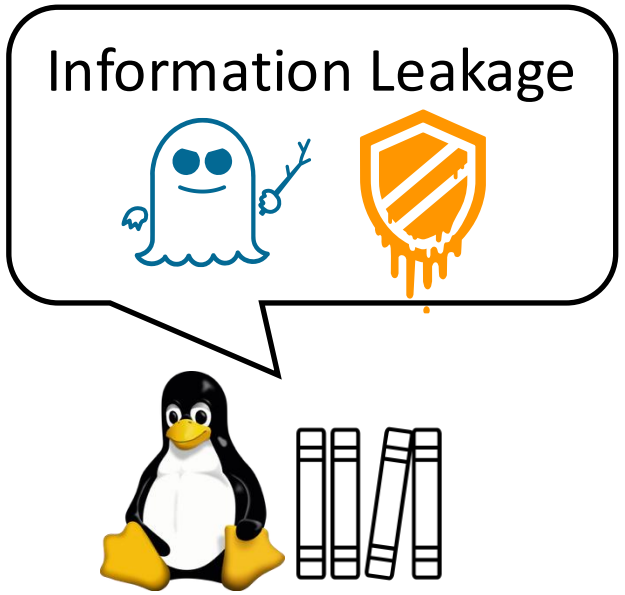


Timing Side-Channel Security



CPU + Spectre Mitigation

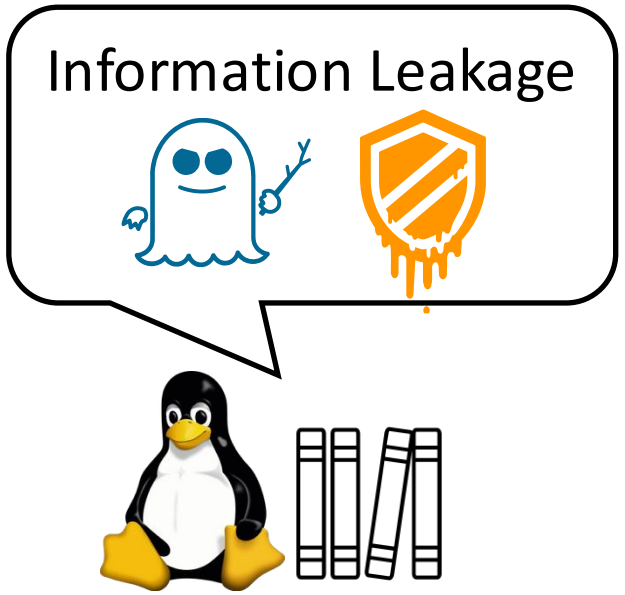
Timing Side-Channel Security



CPU + Spectre Mitigation

Mitigation works properly?
Need **formal verification**
e.g., model checking

Timing Side-Channel Security



CPU + Spectre Mitigation

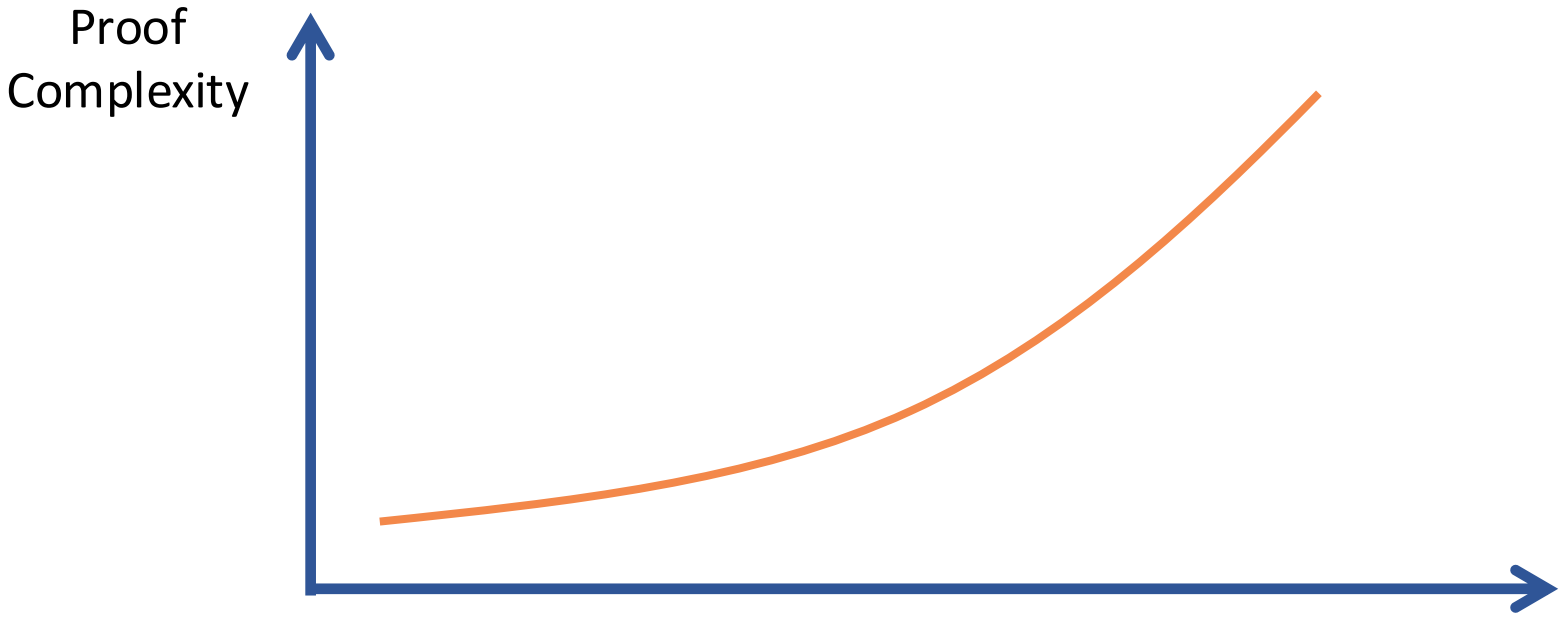
Mitigation works properly?
Need **formal verification**
e.g., model checking

Security Goal: Ensure no information leakage in CPU using formal verification

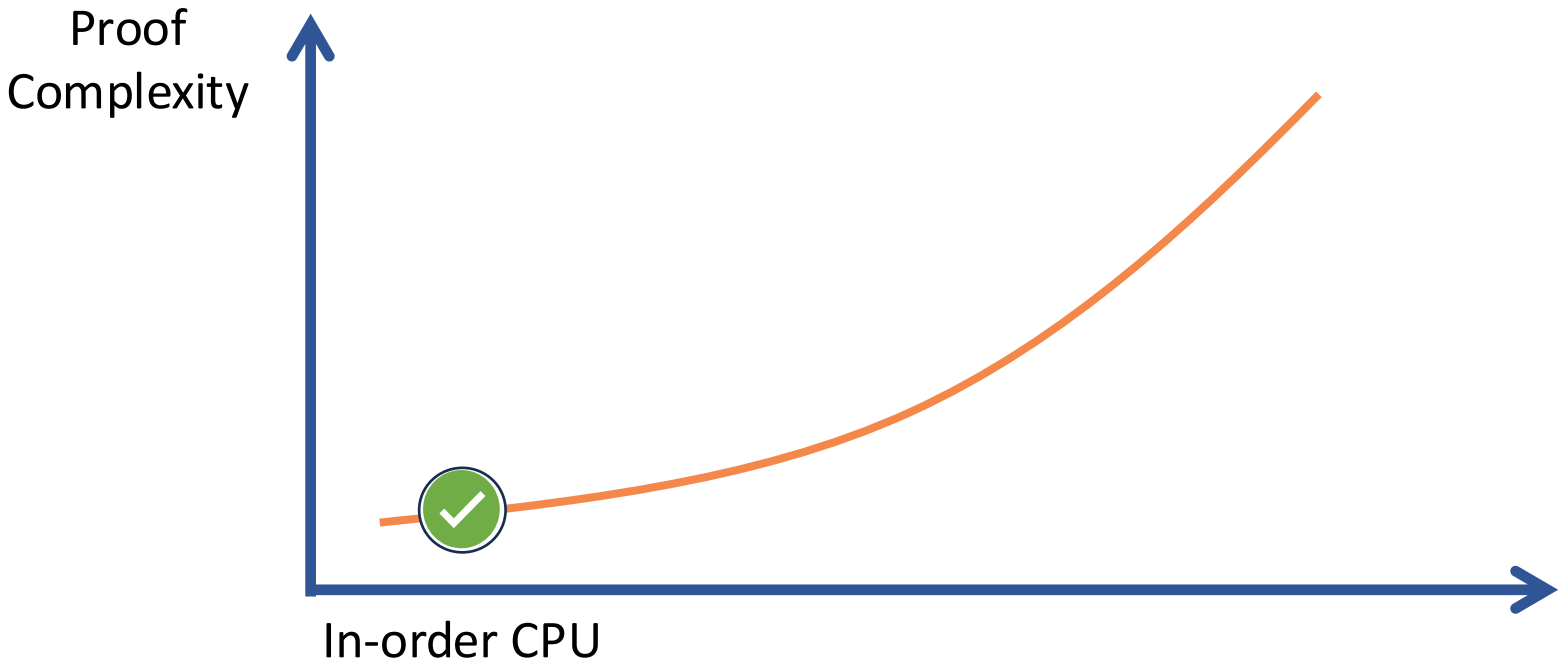
Scalability of Model Checker

Scalability of Model Checker

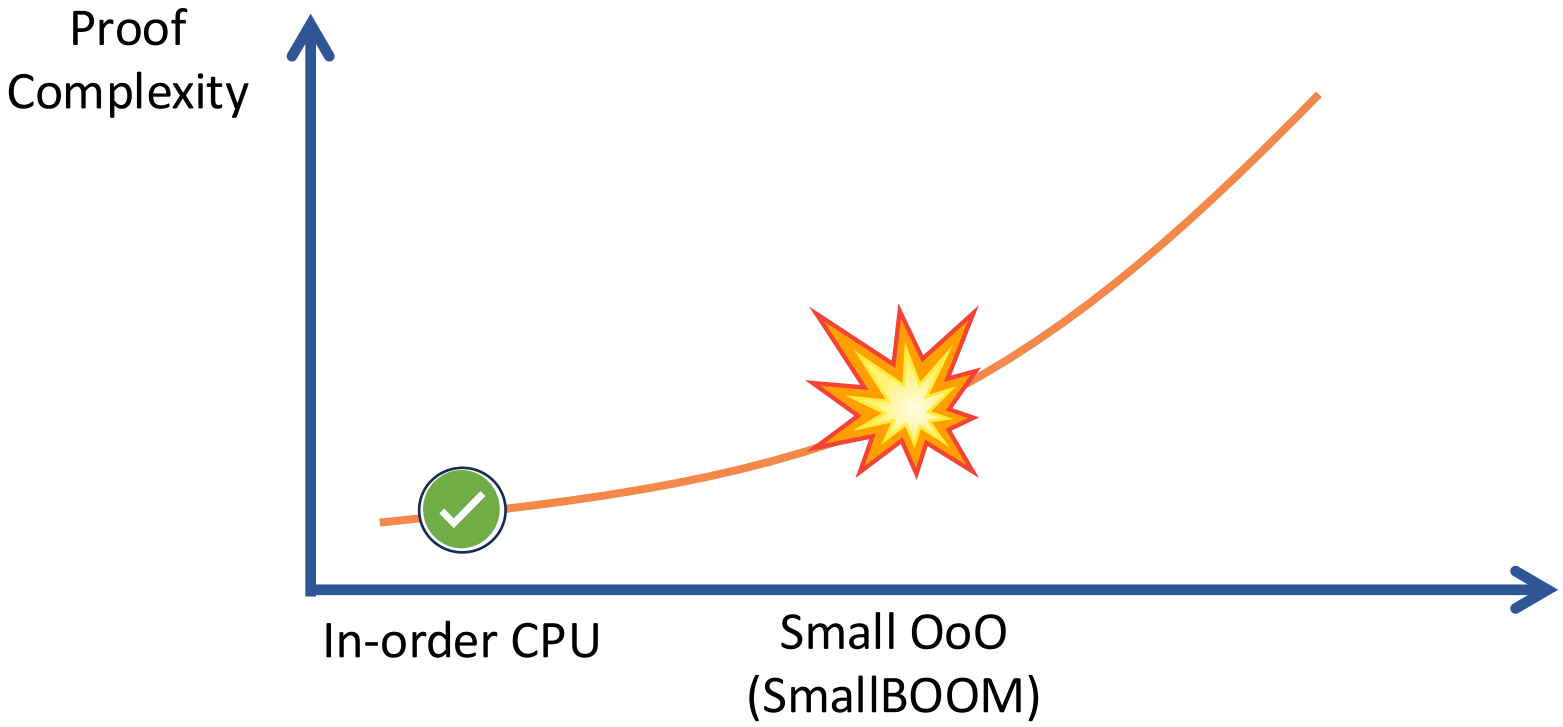
Scalability of Model Checker



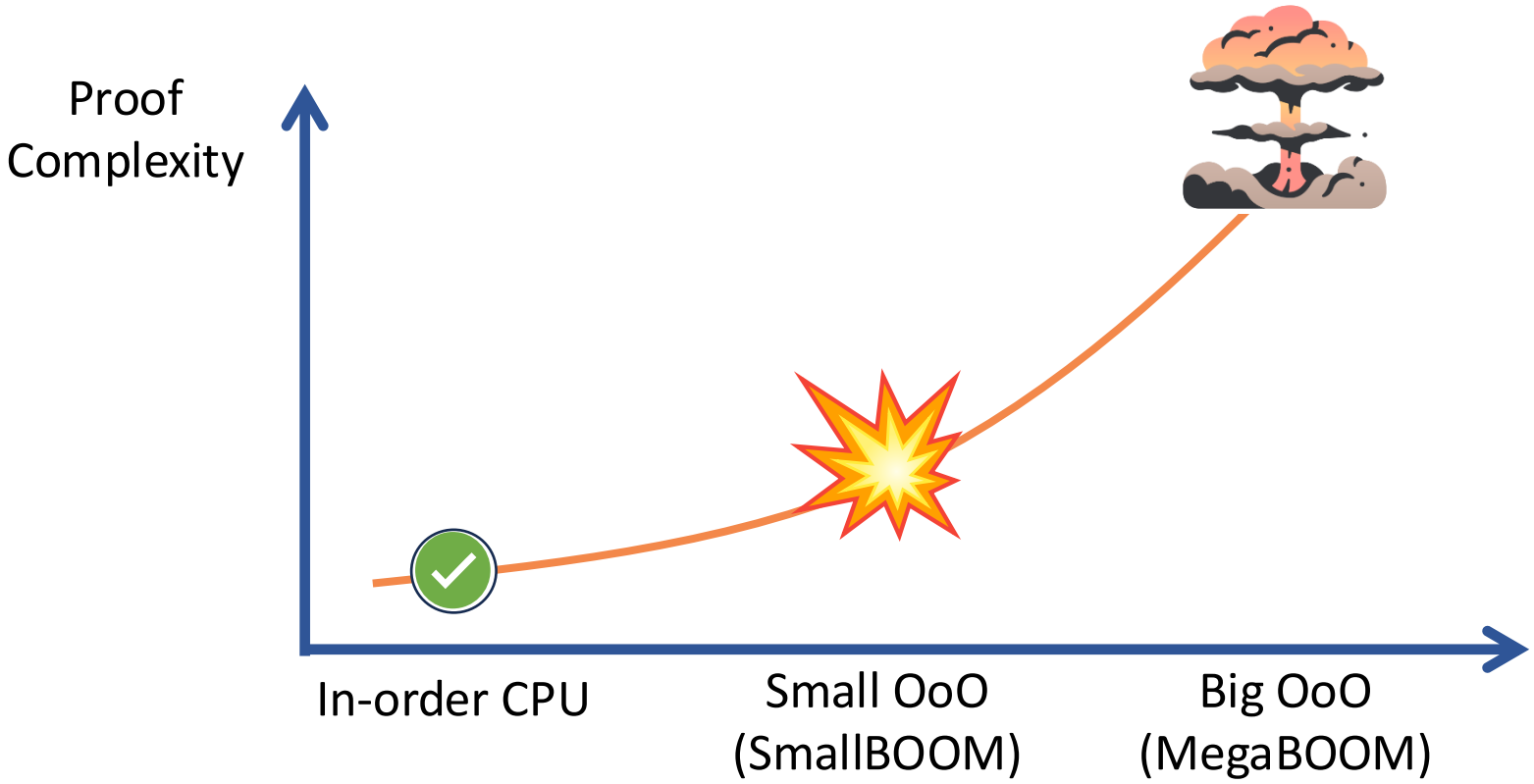
Scalability of Model Checker



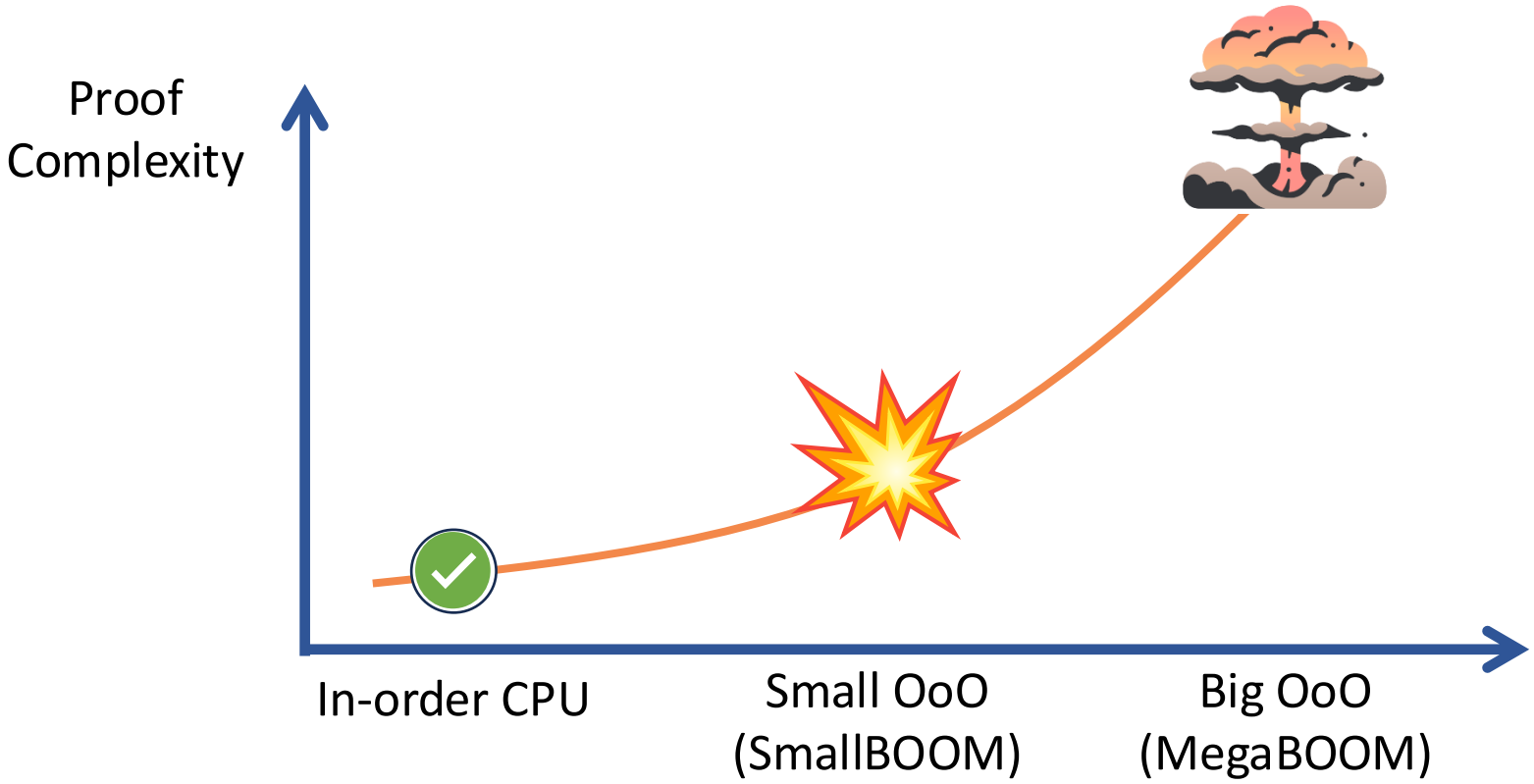
Scalability of Model Checker



Scalability of Model Checker



Scalability of Model Checker



Existing model checkers do not scale to practical designs

Model Checking is a Search Problem

- Search for **invariants**

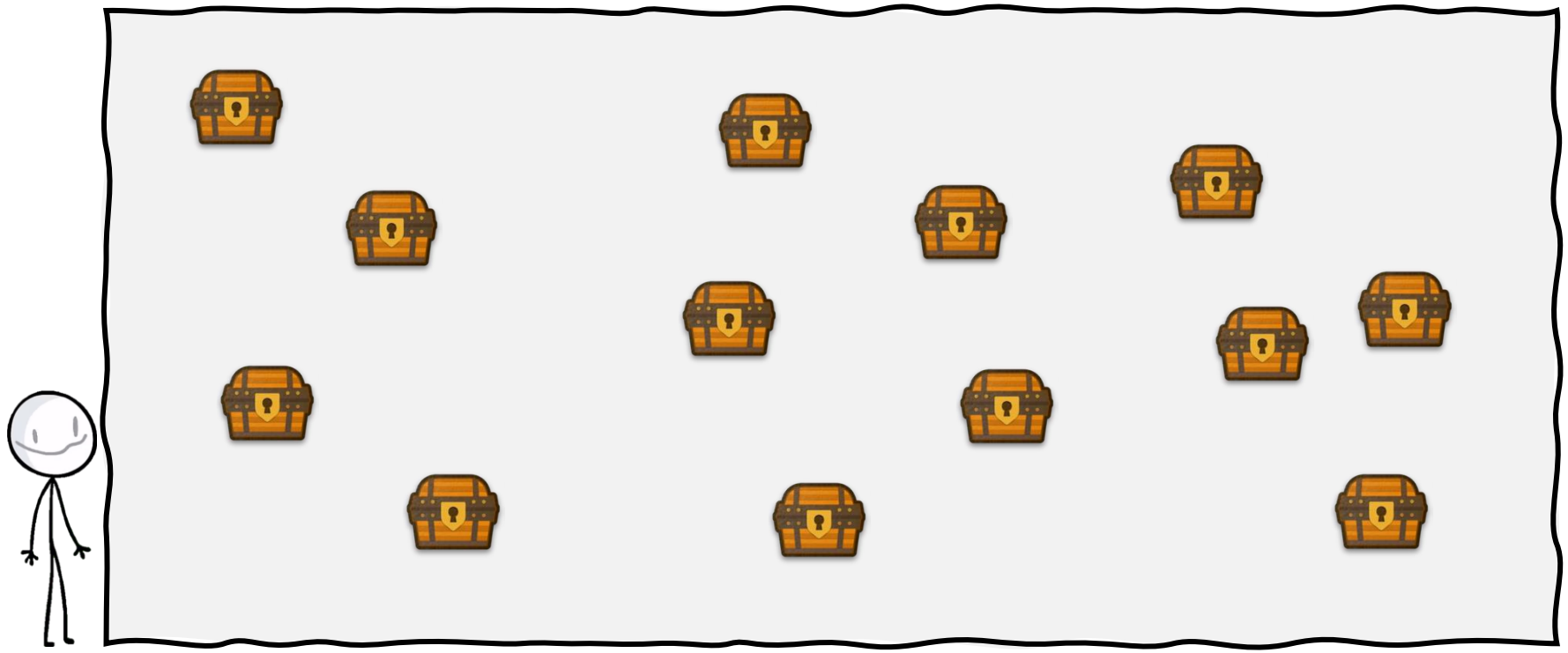
Model Checking is a Search Problem

- Search for **invariants**
- Accumulate **enough useful** invariants

Model Checking is a Search Problem

- Search for **invariants**
- Accumulate **enough useful** invariants

intermediate facts about the design



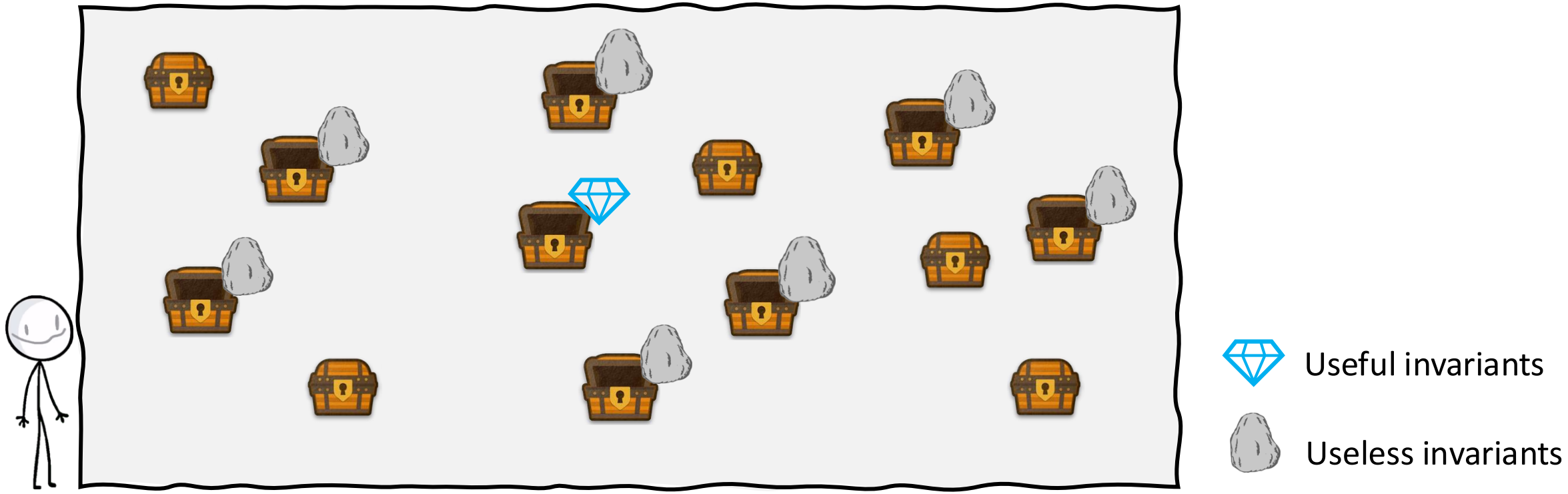
Useful invariants

Useless invariants

Model Checking is a Search Problem

- Search for **invariants**
- Accumulate **enough useful** invariants

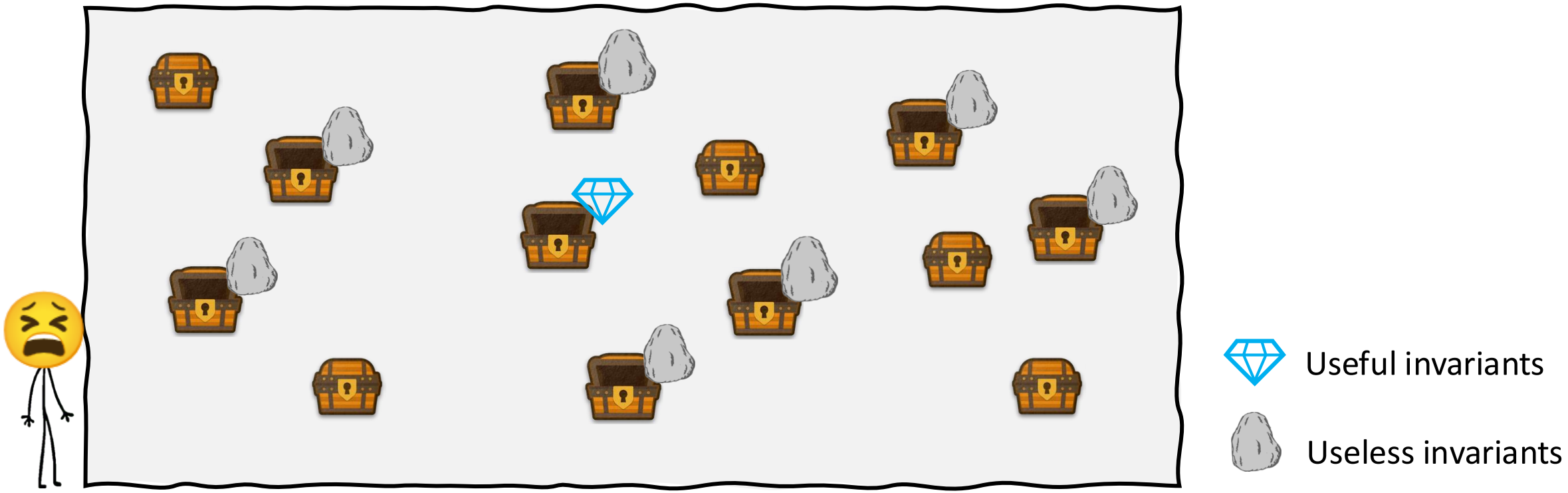
intermediate facts about the design



Model Checking is a Search Problem

- Search for **invariants**
- Accumulate **enough useful** invariants

intermediate facts about the design

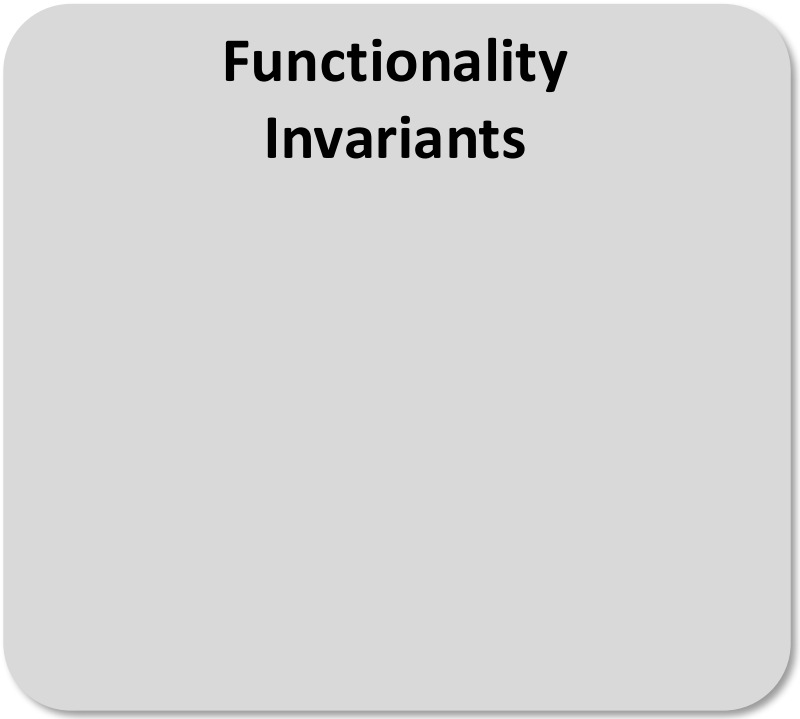


Useful invariants

Useless invariants

Bottleneck of Model Checking Security

Bottleneck of Model Checking Security



Bottleneck of Model Checking Security

	Functionality Invariants
Multiplier	Multiplication result is correct

Bottleneck of Model Checking Security

	Functionality Invariants
Multiplier	Multiplication result is correct
ROB	ROB bookkeeping is consistent with the other stages

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	
ROB	ROB bookkeeping is consistent with the other stages	

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	Computation time is not influenced by secrets
ROB	ROB bookkeeping is consistent with the other stages	

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	Computation time is not influenced by secrets
ROB	ROB bookkeeping is consistent with the other stages	Ready timing is not influenced by secrets

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	Computation time is not influenced by secrets
ROB	ROB bookkeeping is consistent with the other stages	Ready timing is not influenced by secrets
Useful to prove security properties?		

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	Computation time is not influenced by secrets
ROB	ROB bookkeeping is consistent with the other stages	Ready timing is not influenced by secrets
Useful to prove security properties?	**Often not useful** 	

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	Computation time is not influenced by secrets
ROB	ROB bookkeeping is consistent with the other stages	Ready timing is not influenced by secrets
Useful to prove security properties?	**Often not useful** 	**Directly useful** 

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	Computation time is not influenced by secrets
ROB	ROB bookkeeping is consistent with the other stages	Ready timing is not influenced by secrets
Useful to prove security properties?	**Often not useful** 	**Directly useful** 
Inside a timeout model checker: *(running rIC3 on SmallBOOM for 1 hour)*	**>70%**	**<30%**

Bottleneck of Model Checking Security

	Functionality Invariants	Information-flow Invariants
Multiplier	Multiplication result is correct	Computation time is not influenced by secrets
ROB	ROB bookkeeping is consistent with the other stages	Ready timing is not influenced by secrets
Useful to prove security properties?	**Often not useful** 	**Directly useful** 

Our discovery: Model checking wastes time in **functionality** facts that are **unnecessary** for security

Key Contributions

Our Insight:

Decoupling security and functionality can scale verification

Key Contributions

Our Insight:

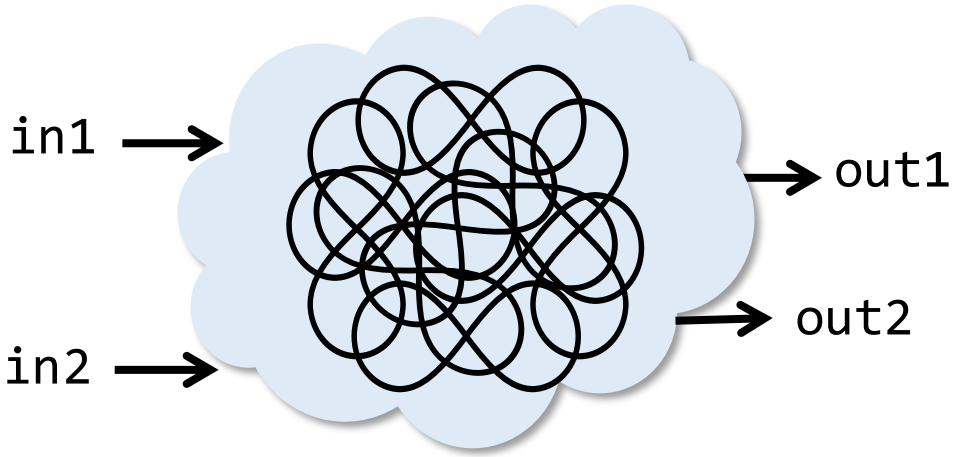
Decoupling security and functionality can scale verification

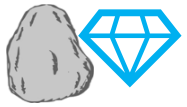
New abstraction primitive **HUF**

- abstract irrelevant functionality
- preserve necessary information flow

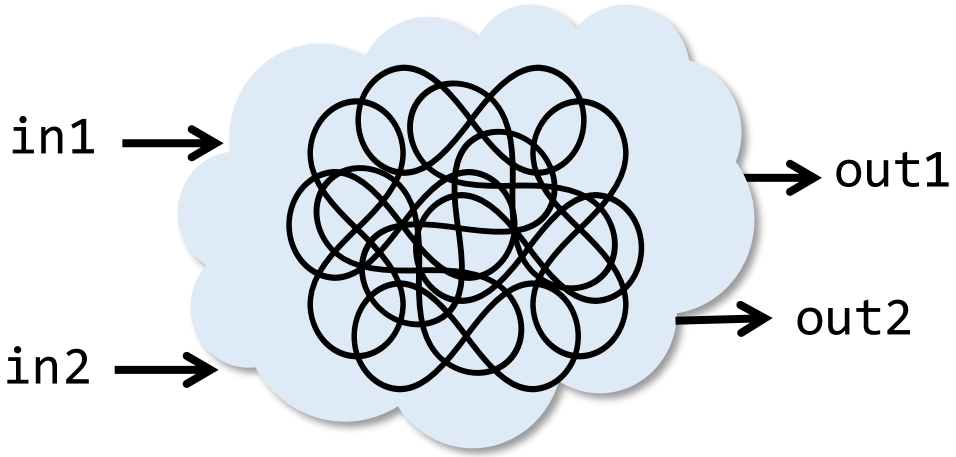
Abstraction Guides Model Checking

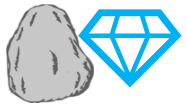
Abstraction Guides Model Checking



Concrete: complete functionality 

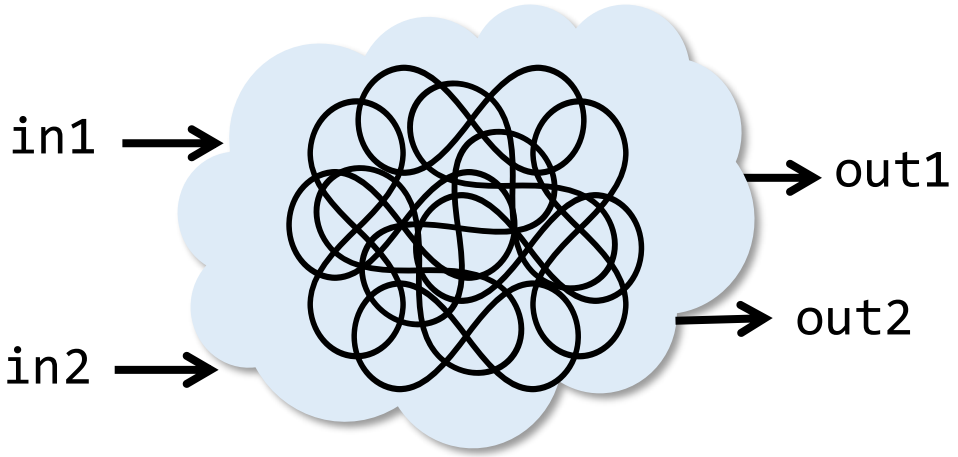
Abstraction Guides Model Checking

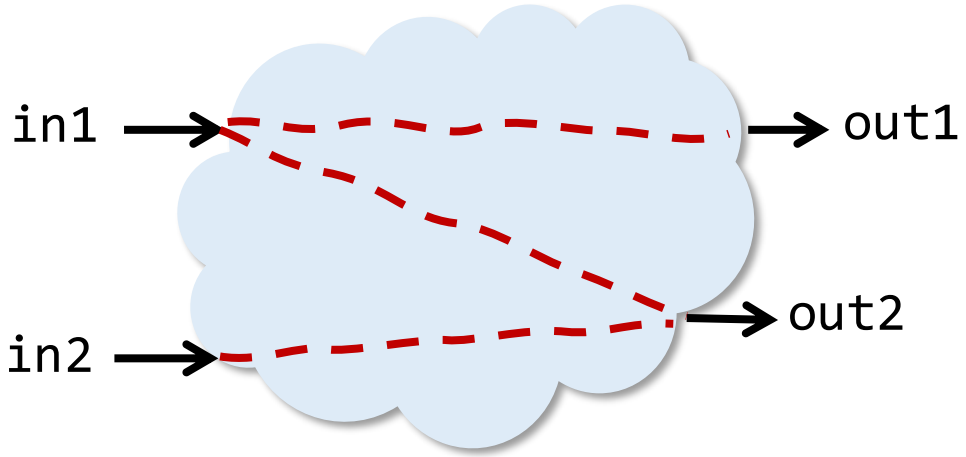


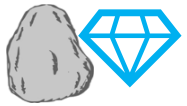
Concrete: complete functionality 

“how inputs influence outputs”

Abstraction Guides Model Checking



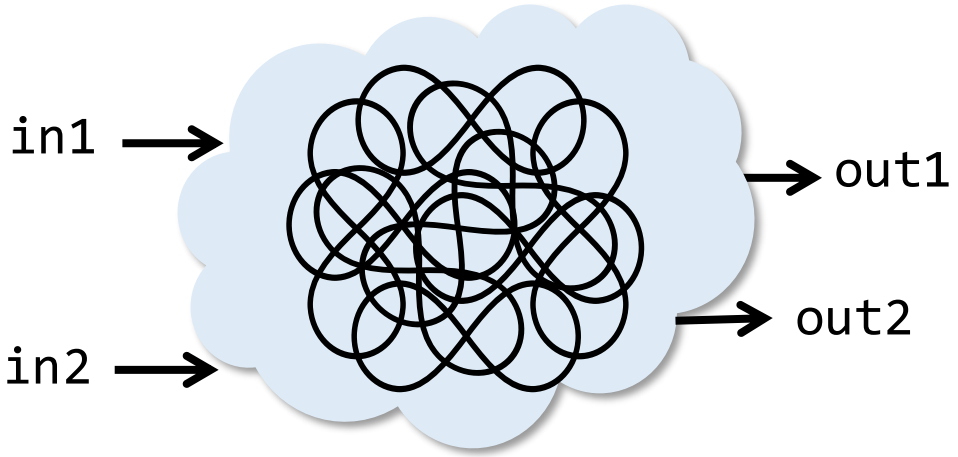


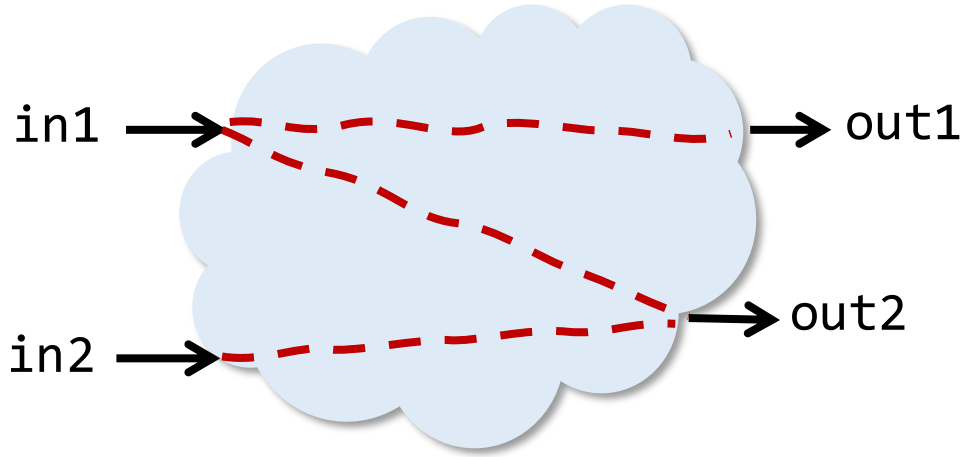
Concrete: complete functionality 

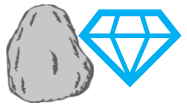
Abstracted: information-flow only 

“how inputs influence outputs”

Abstraction Guides Model Checking





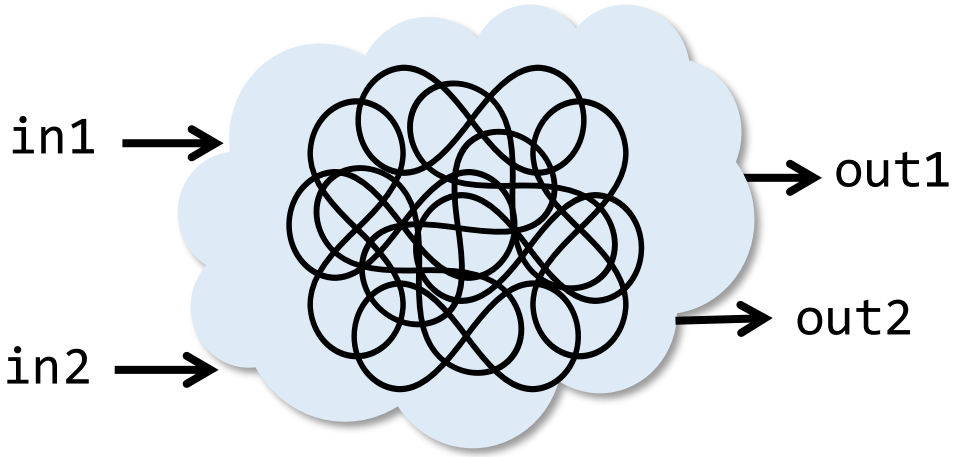
Concrete: complete functionality 

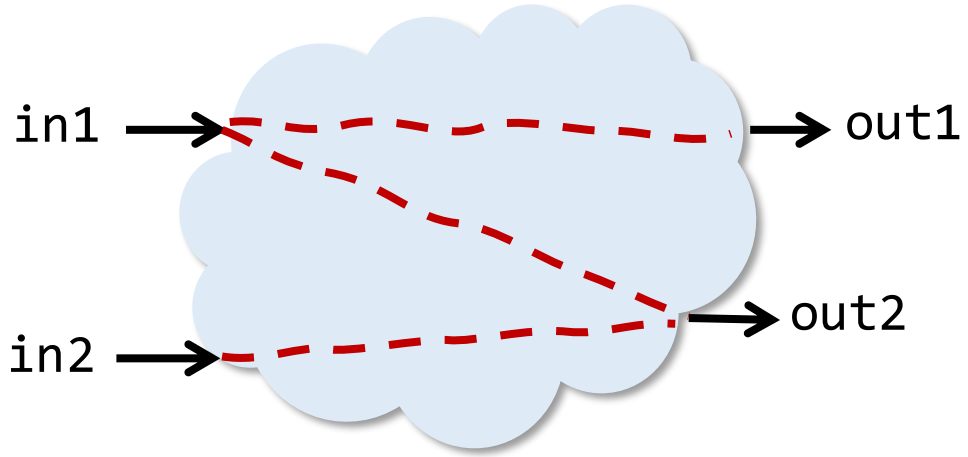
Abstracted: information-flow only 

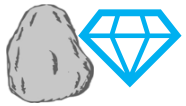
“how inputs influence outputs”

“whether inputs influence outputs”

Abstraction Guides Model Checking





Concrete: complete functionality 

Abstracted: information-flow only 

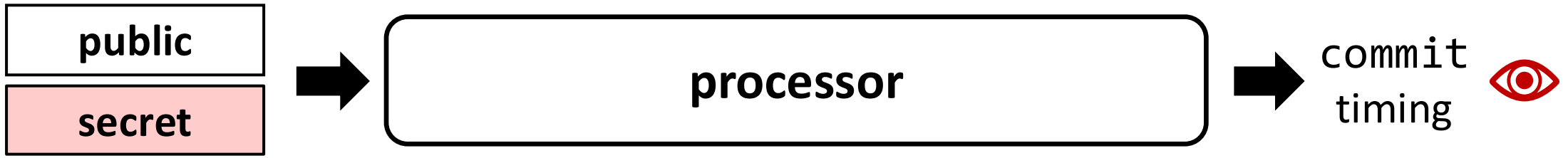
"how inputs influence outputs"

"whether inputs influence outputs"

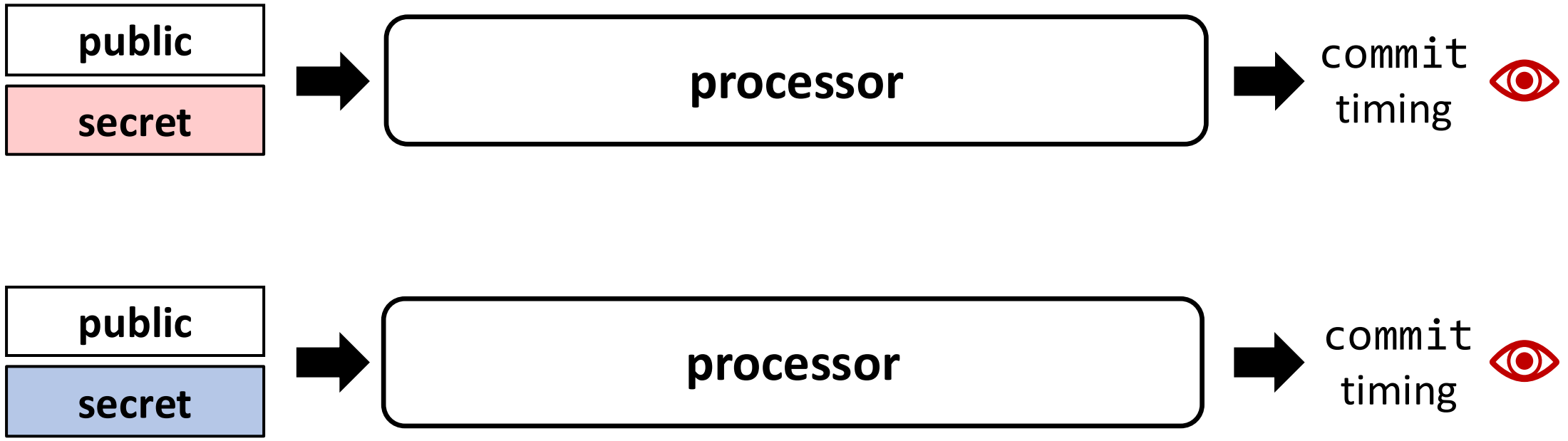
Abstraction removes functionality to promote information-flow studying

Proof Construction: Self-Composition

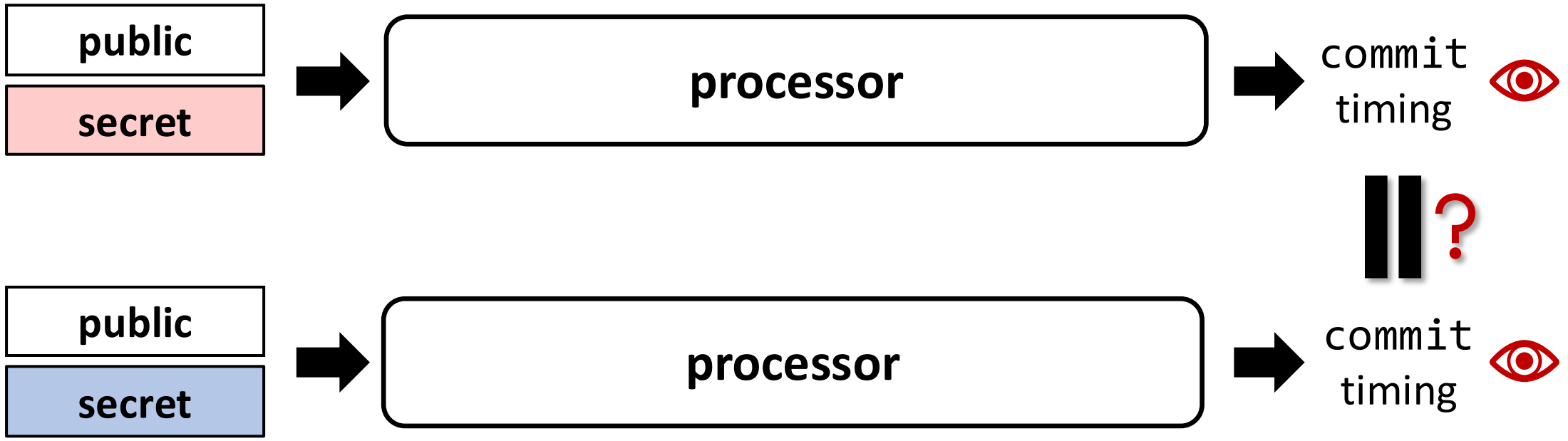
Proof Construction: Self-Composition



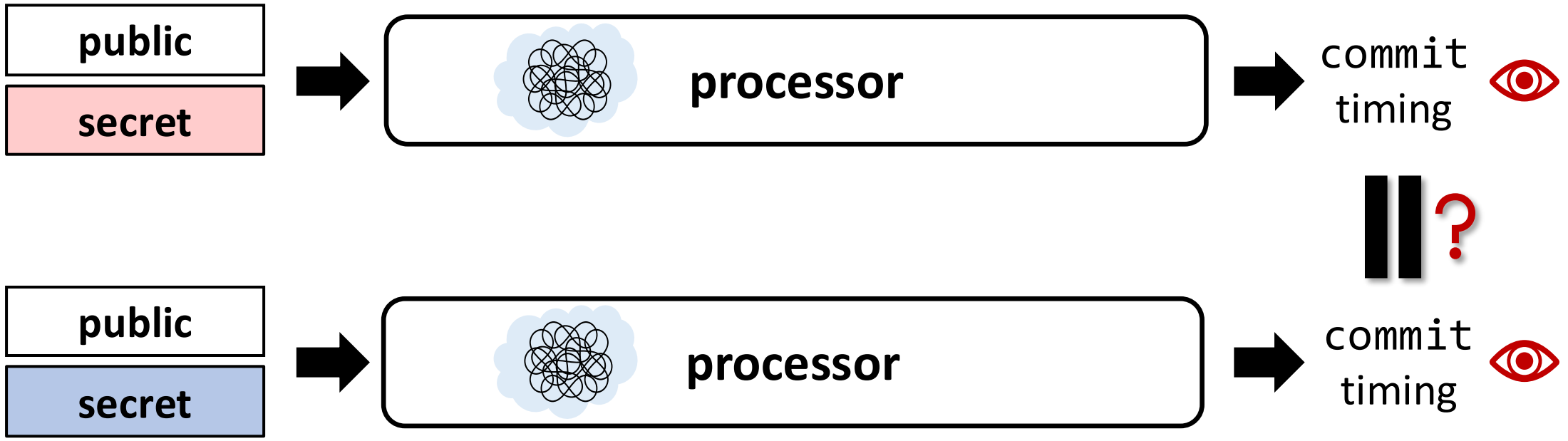
Proof Construction: Self-Composition



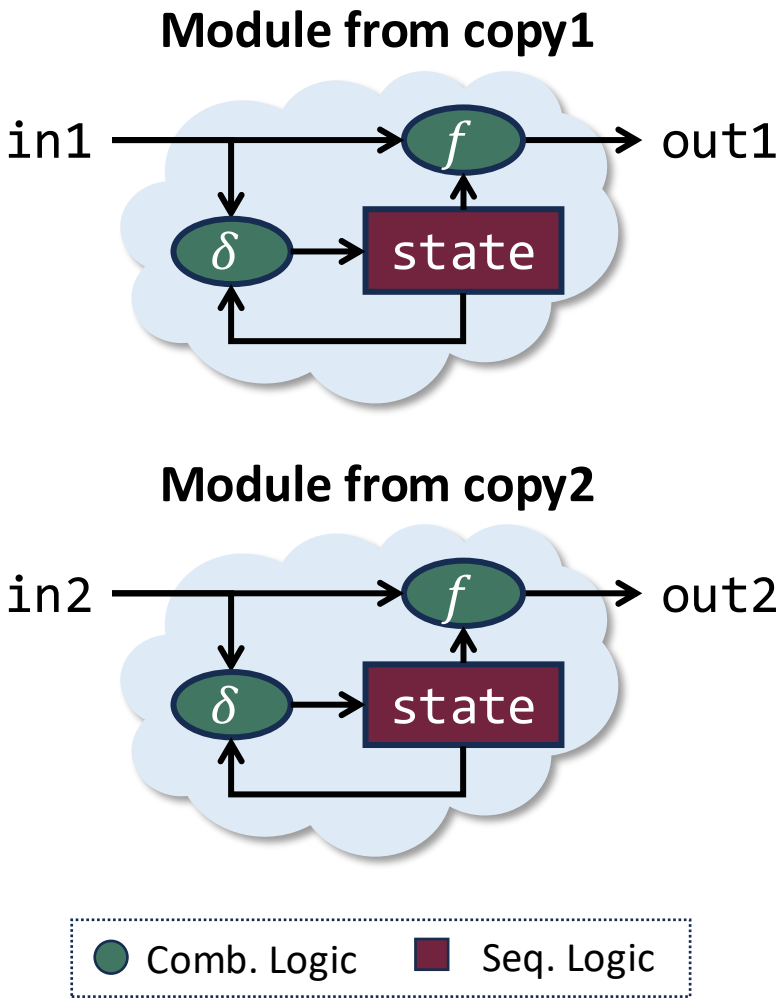
Proof Construction: Self-Composition



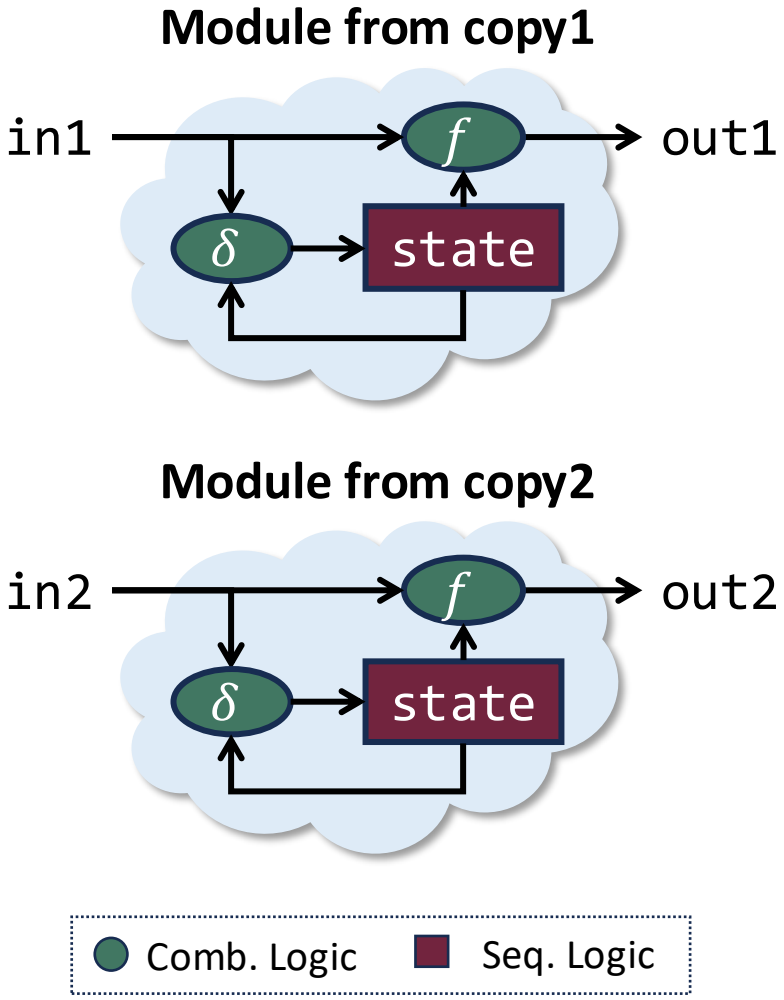
Proof Construction: Self-Composition



HUF Abstracts Sequential Modules



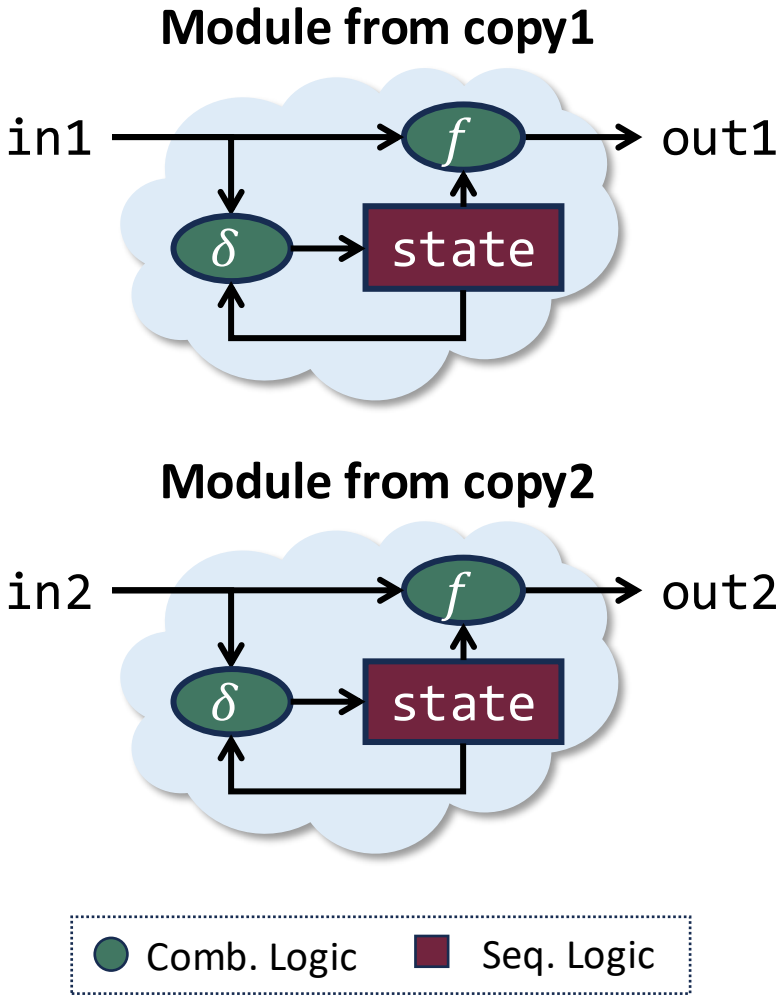
HUF Abstracts Sequential Modules



Goal of HUF

1. model **whether** `in` influences `out`
2. remove f, δ, `state`

HUF Abstracts Sequential Modules



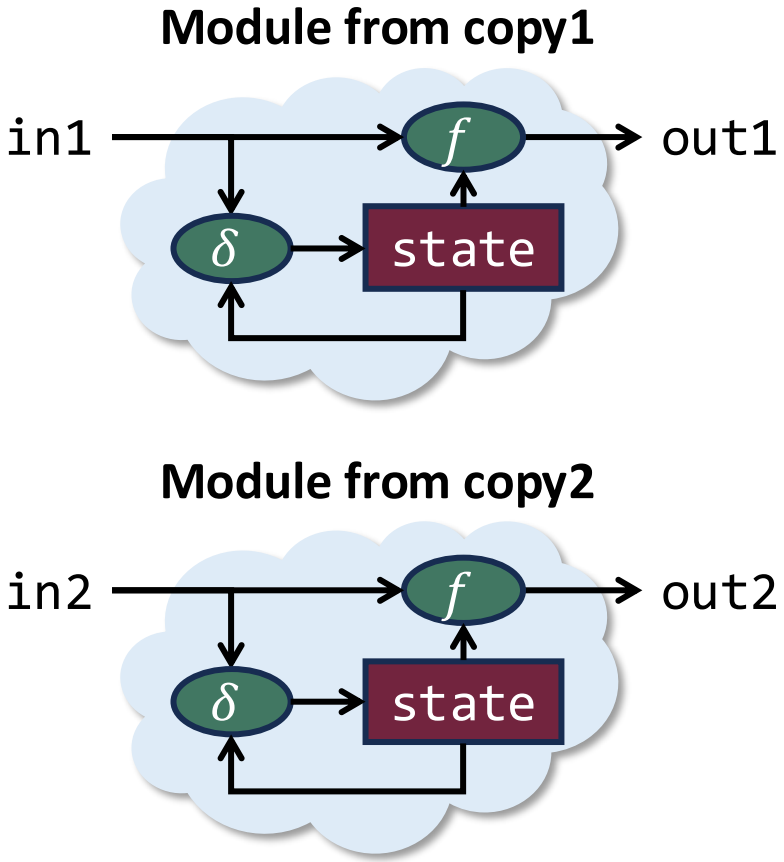
Goal of HUF

1. model **whether** `in` influences `out`
2. remove f, δ, `state`

HUF: **U**ninterpreted **F**unction with **H**istory

Outputs must be equal,
when input histories are equal

HUF Abstracts Sequential Modules



Goal of HUF

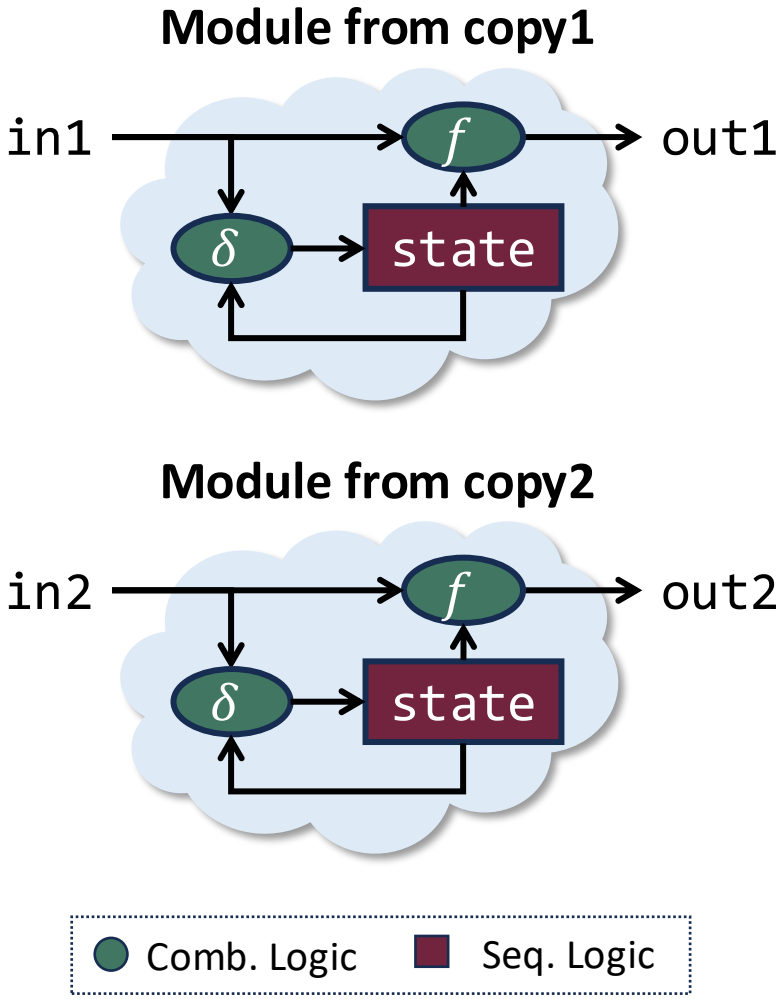
1. model **whether** `in` influences `out`
2. remove f, δ, `state`

HUF: **U**ninterpreted **F**unction with **H**istory

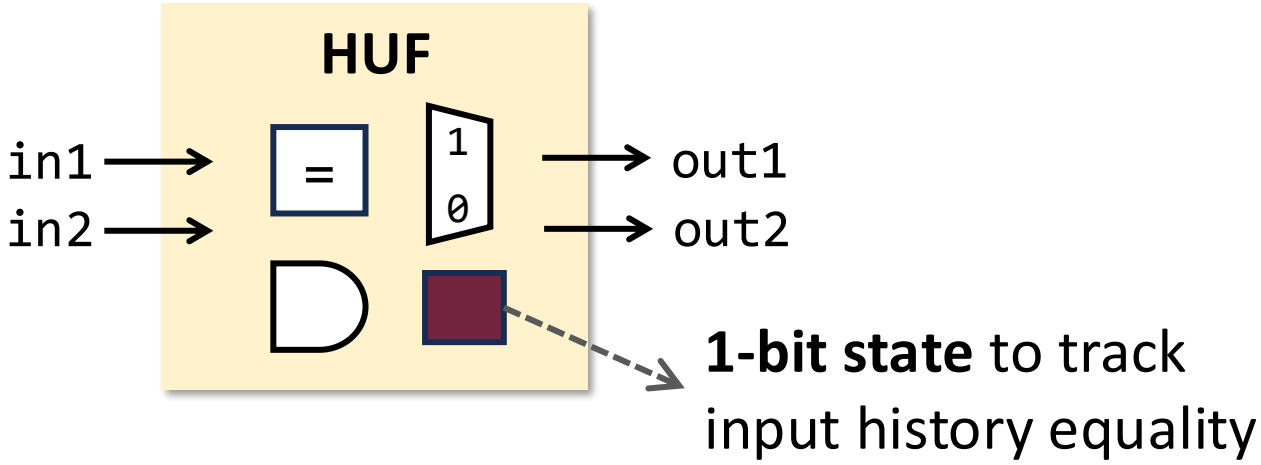
Outputs must be equal,
when input histories are equal

$$\forall t \in \mathbb{N}.\ \texttt{in1[0…t]} = \texttt{in2[0…t]} \implies \texttt{out1[t]} = \texttt{out2[t]}$$

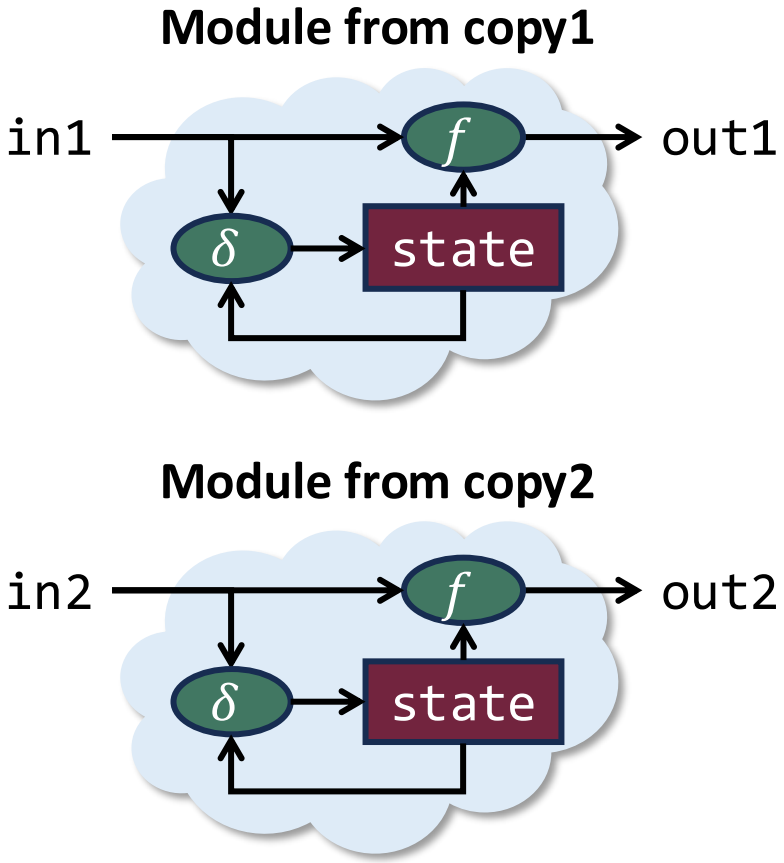
HUF Abstracts Sequential Modules



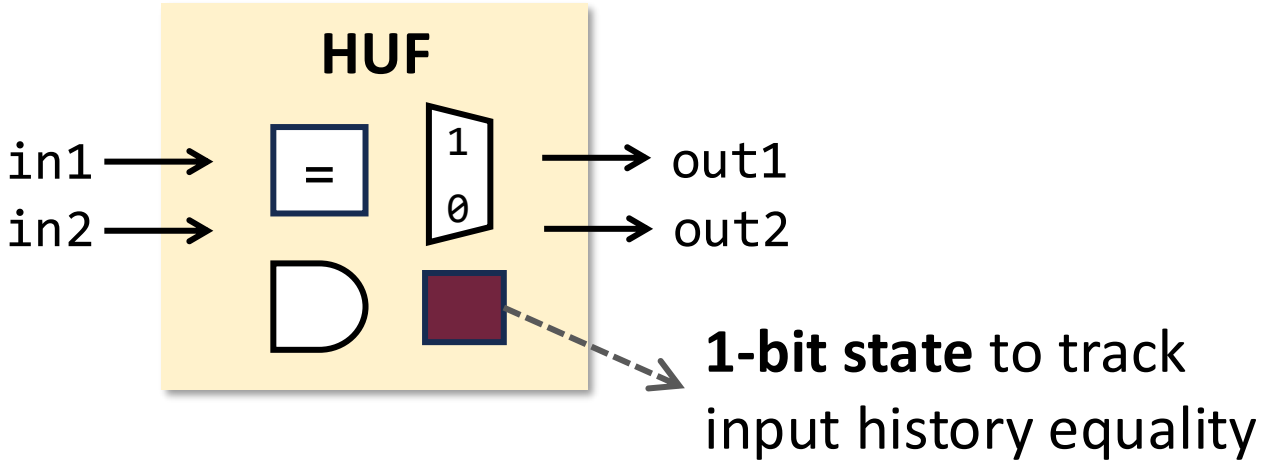
$$\forall t \in \mathbb{N}.\ \texttt{in1}[0\ldots t] = \texttt{in2}[0\ldots t] \implies \texttt{out1}[t] = \texttt{out2}[t]$$



HUF Abstracts Sequential Modules



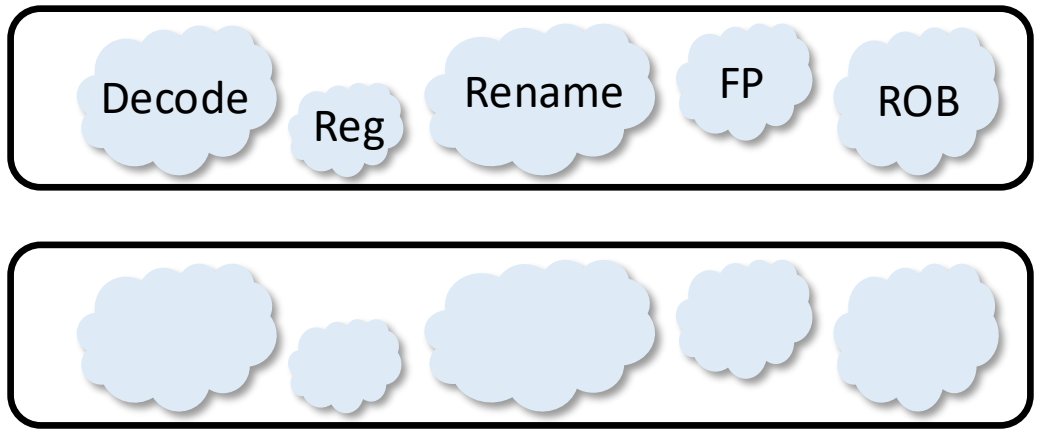
$$\forall t \in \mathbb{N}.\ \texttt{in1}[0\ldots t] = \texttt{in2}[0\ldots t] \implies \texttt{out1}[t] = \texttt{out2}[t]$$



HUF reduces a sequential module to 1-bit state of history + simple gates, enabling efficient invariant searching

HUF Verification Flow

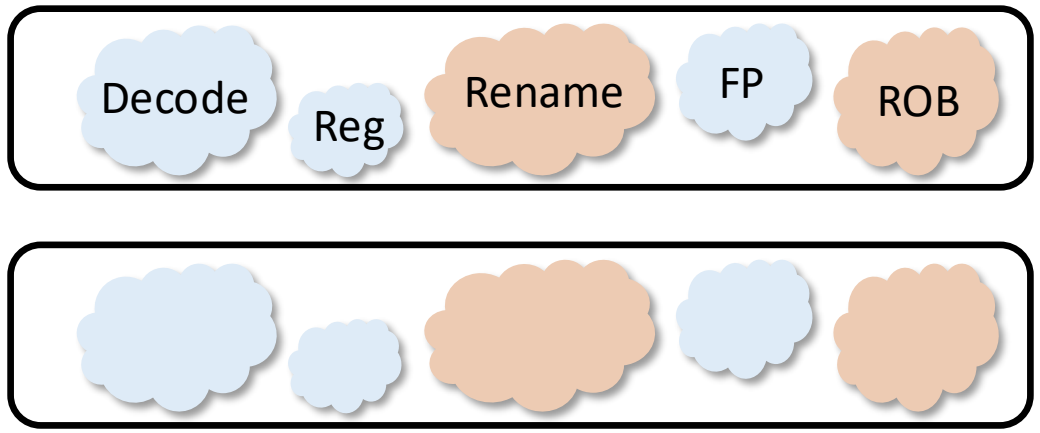
Verification target:
Self-composed processor pair



HUF Verification Flow

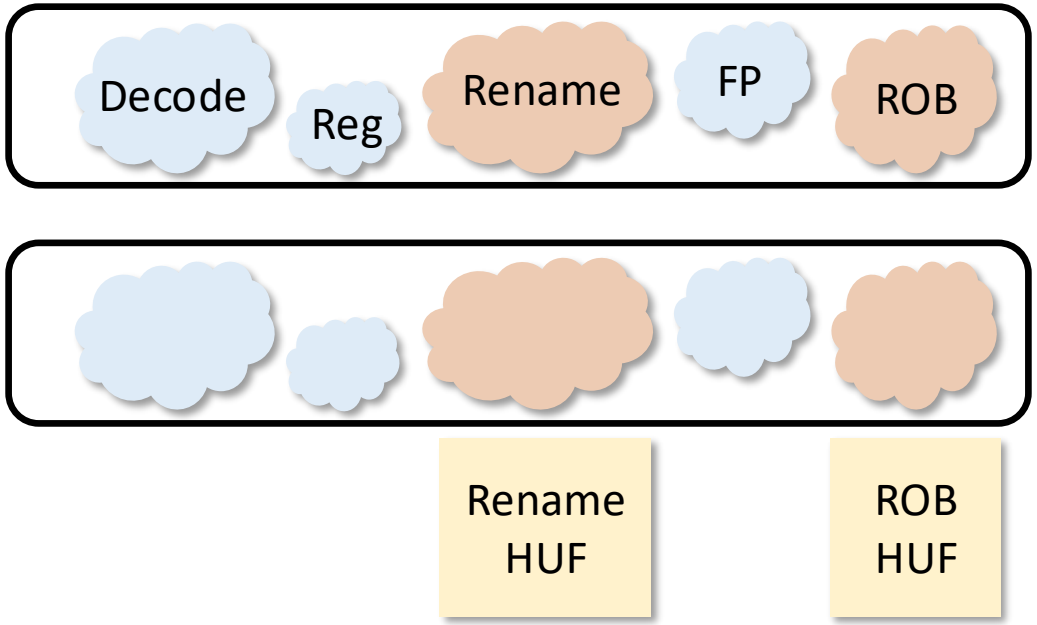
Select modules

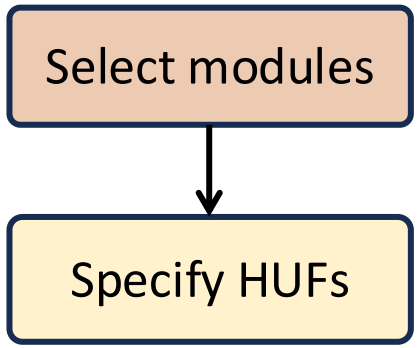
Verification target:
Self-composed processor pair



HUF Verification Flow

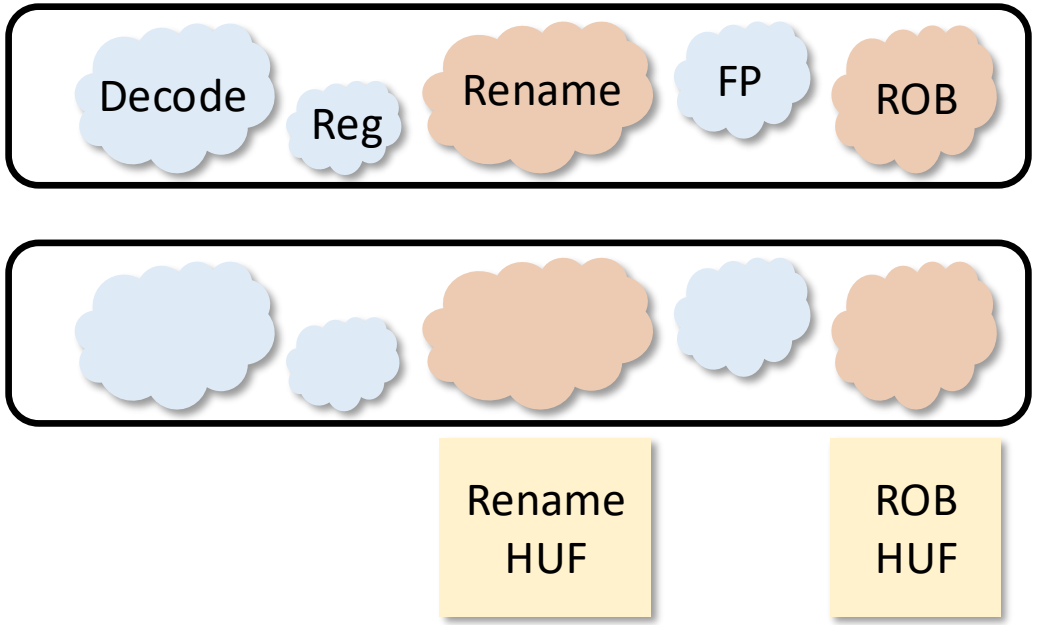
Verification target:
Self-composed processor pair

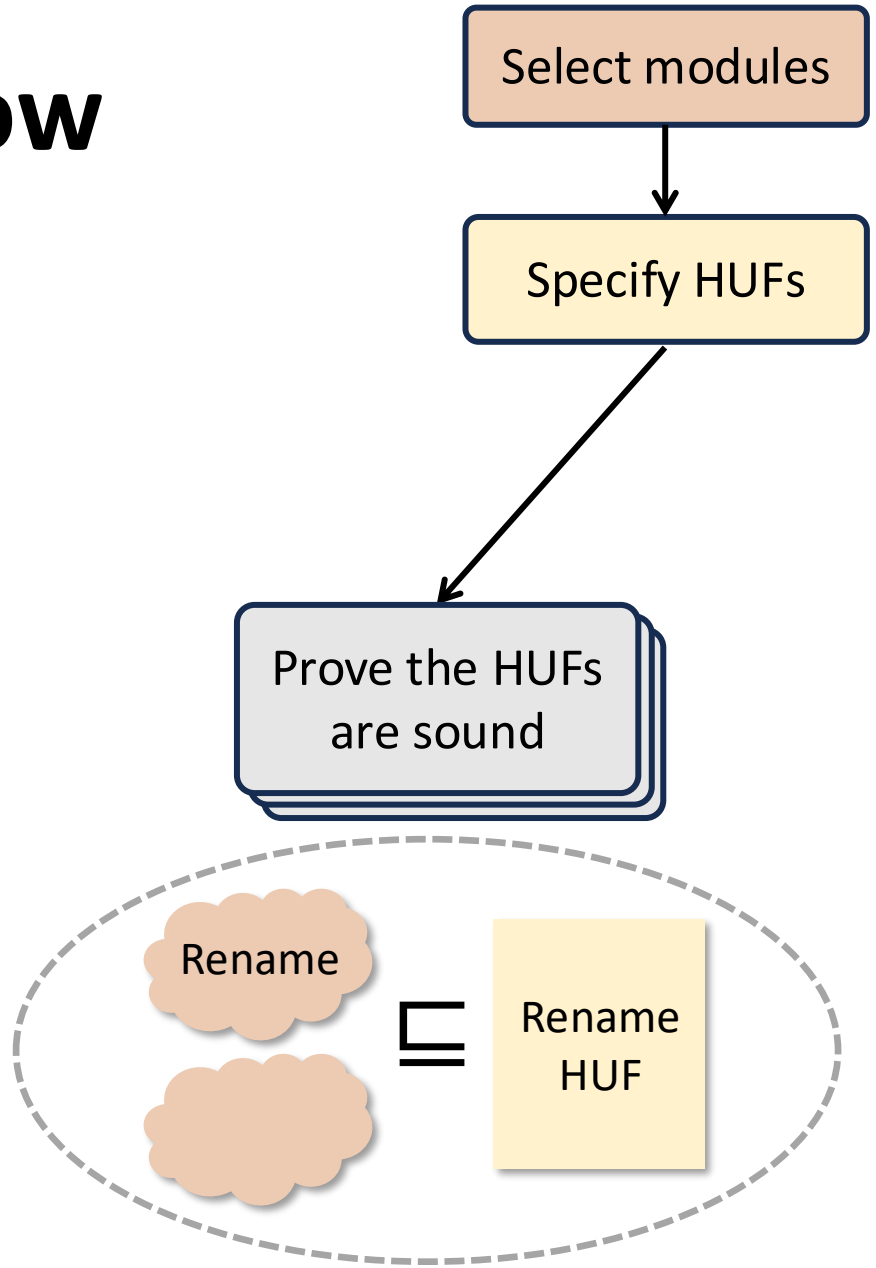




HUF Verification Flow

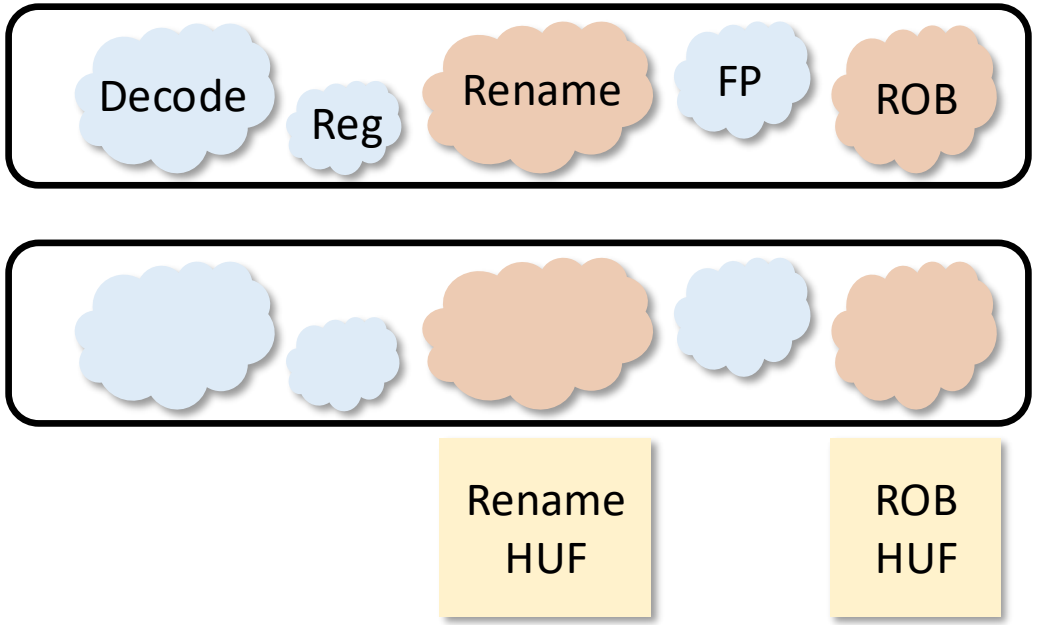
Verification target:
Self-composed processor pair

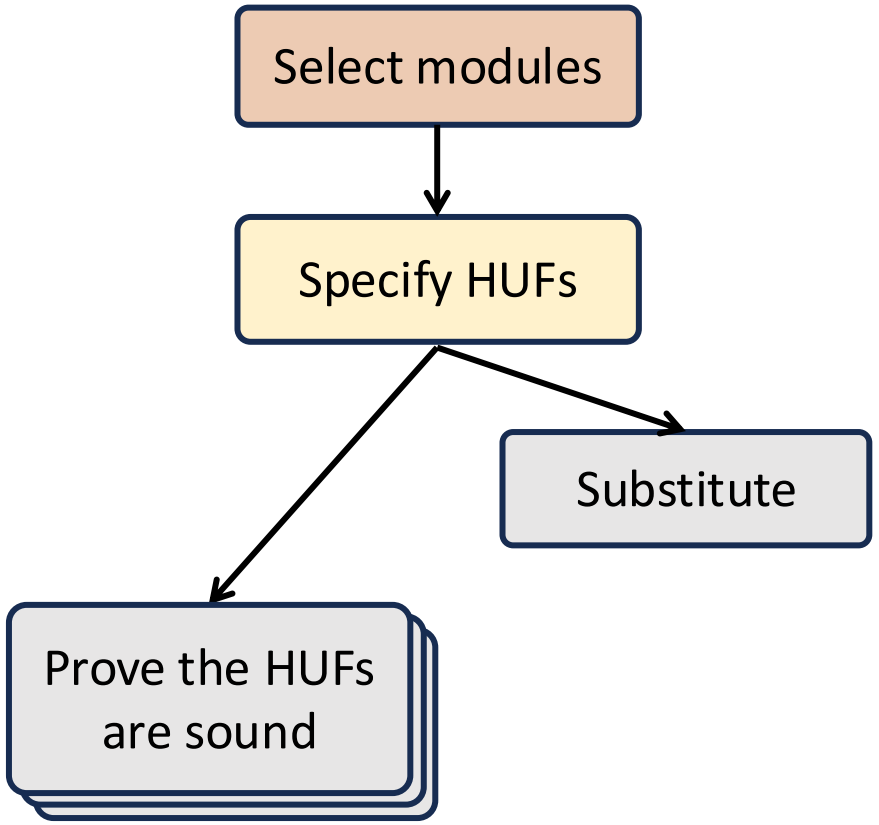




HUF Verification Flow

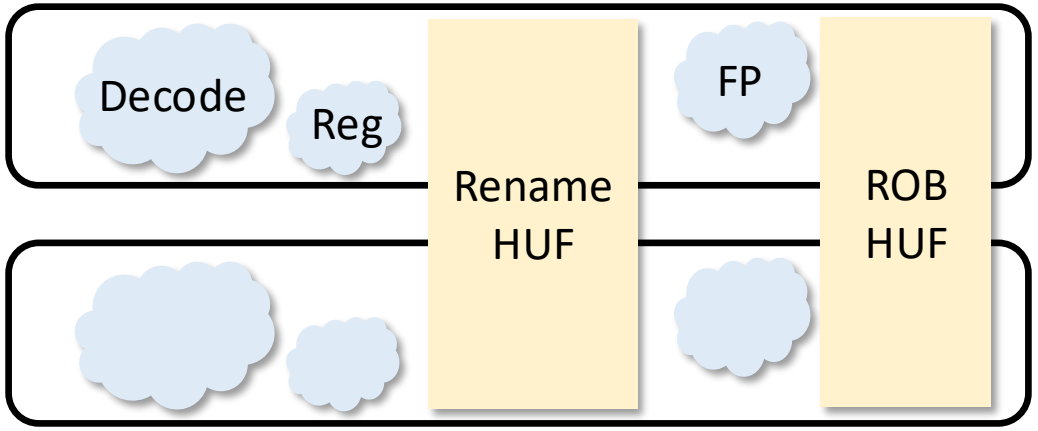
Verification target:
Self-composed processor pair

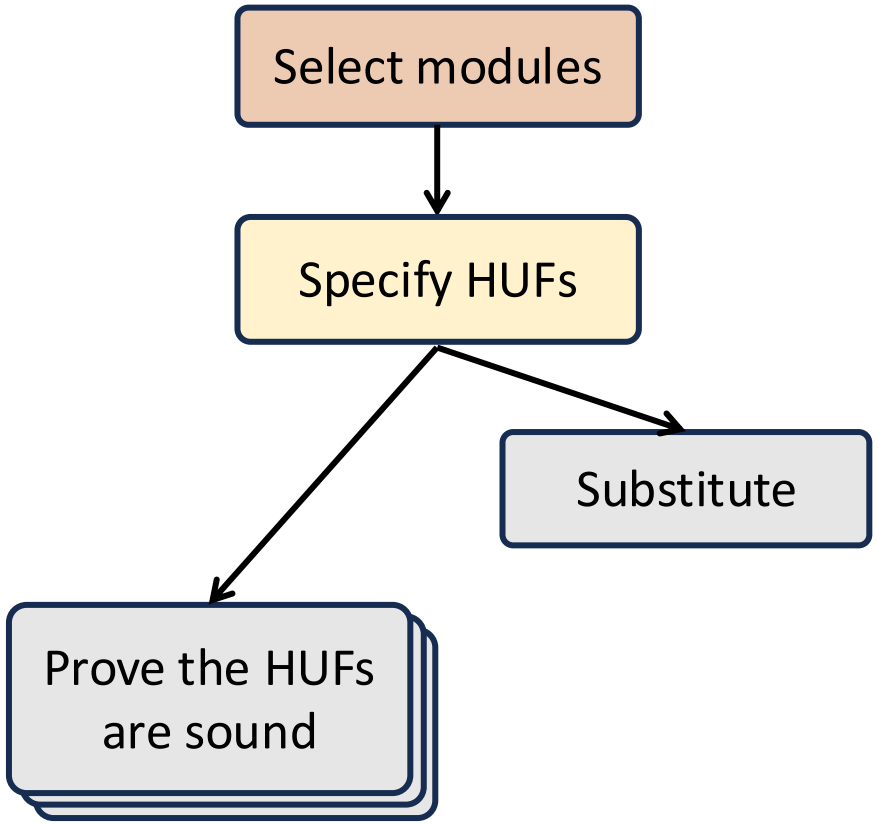




HUF Verification Flow

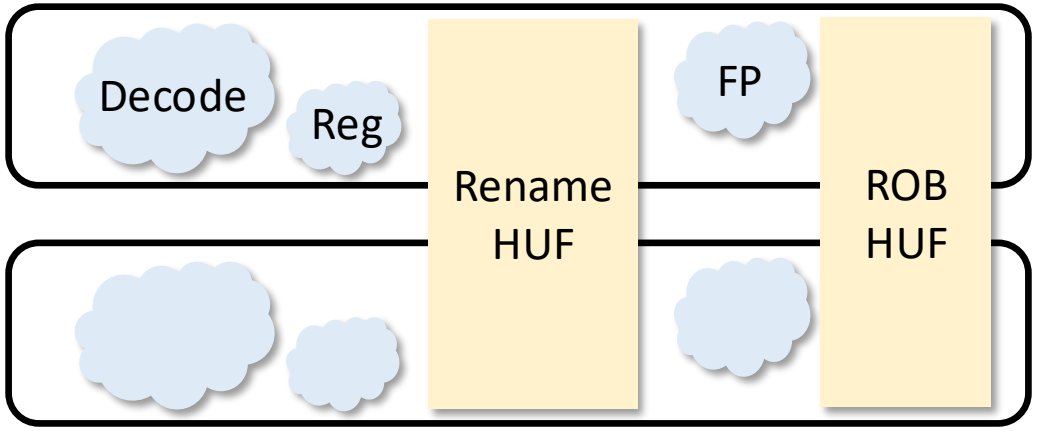
Verification target:
Self-composed processor pair

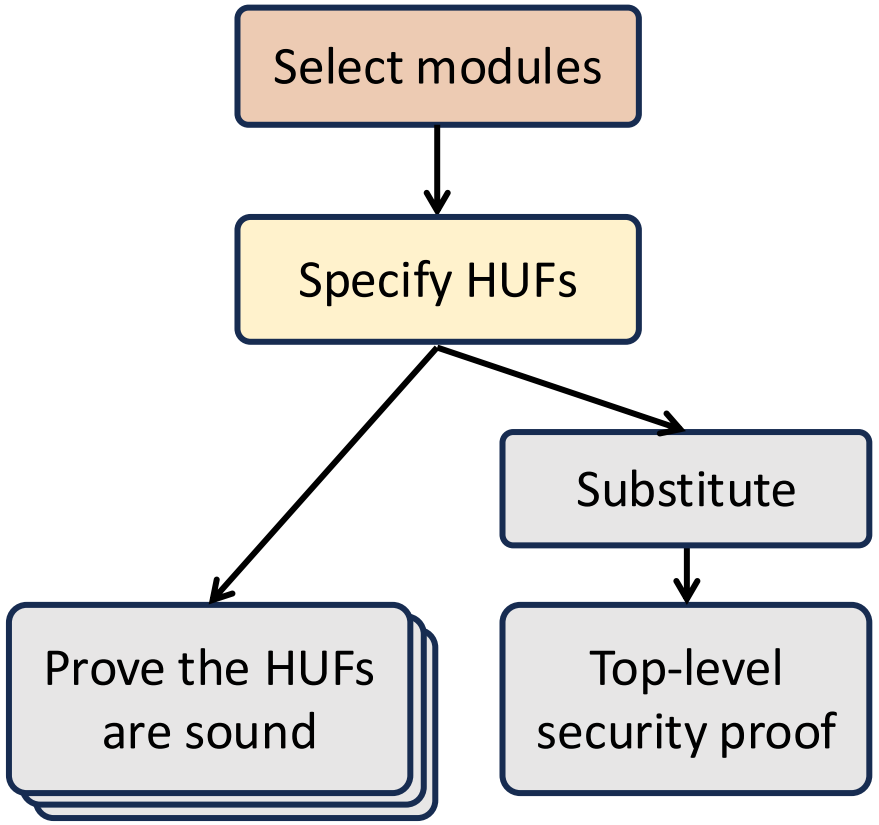




HUF Verification Flow

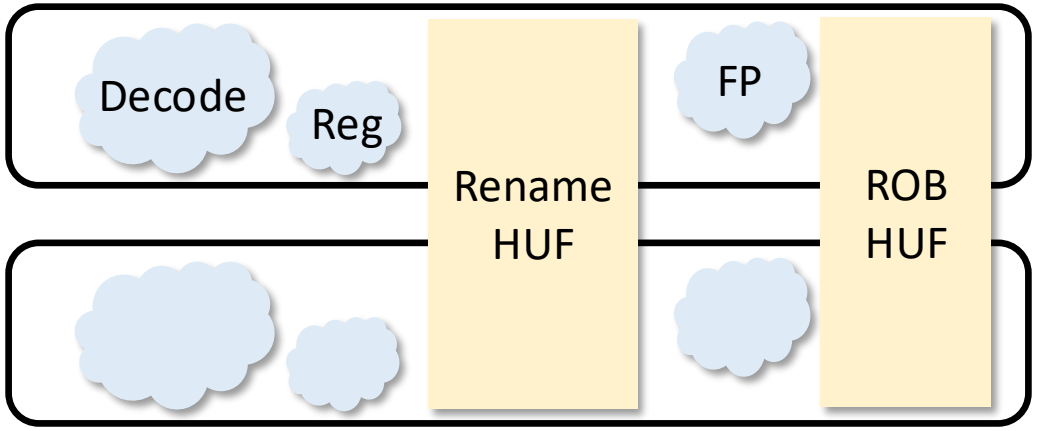
Verification target:
Self-composed processor pair

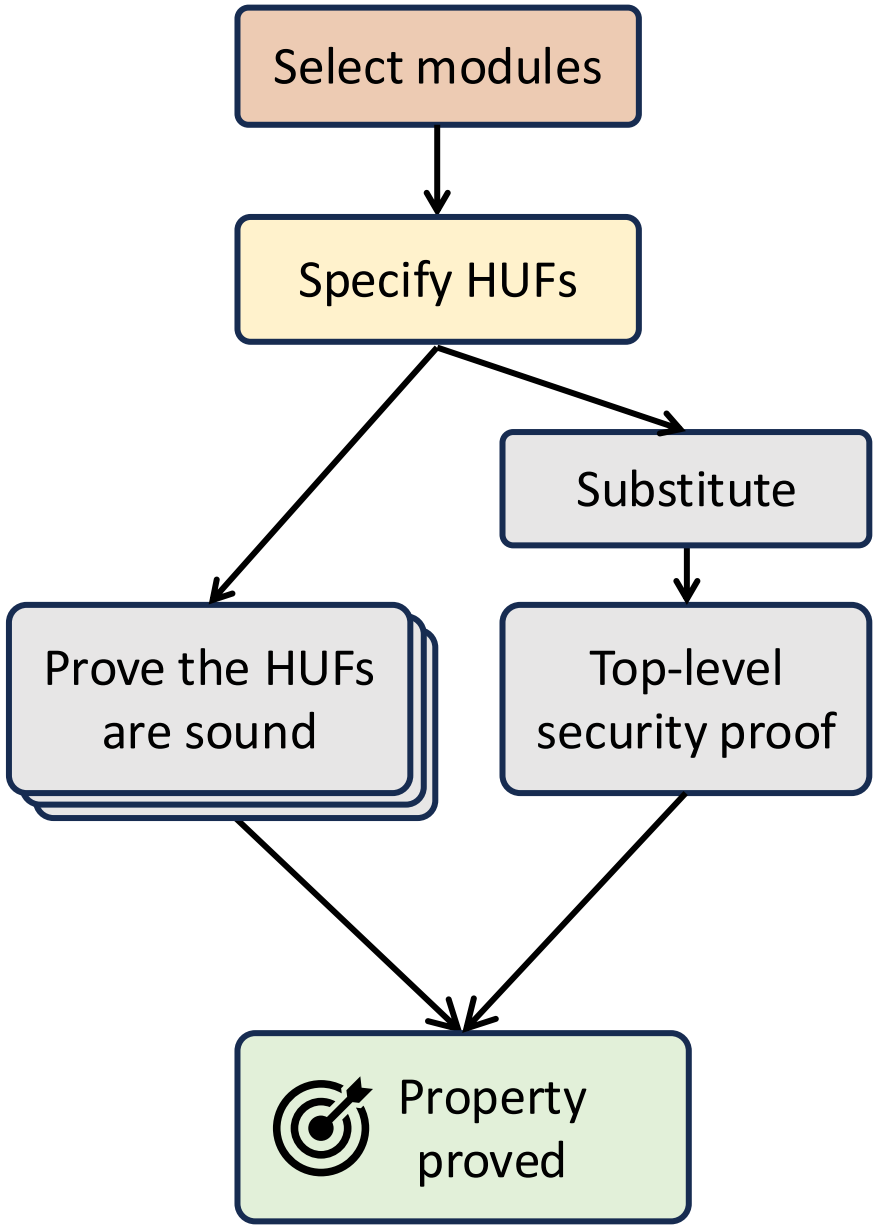




HUF Verification Flow

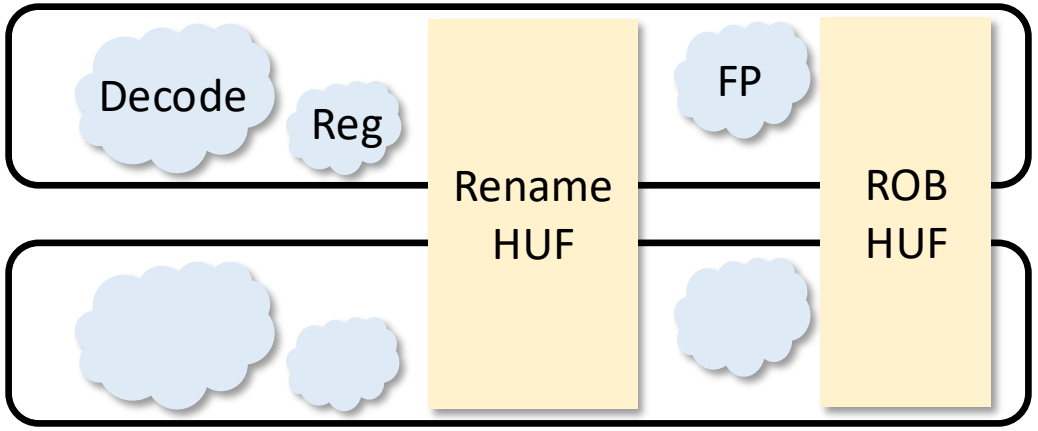
Verification target:
Self-composed processor pair

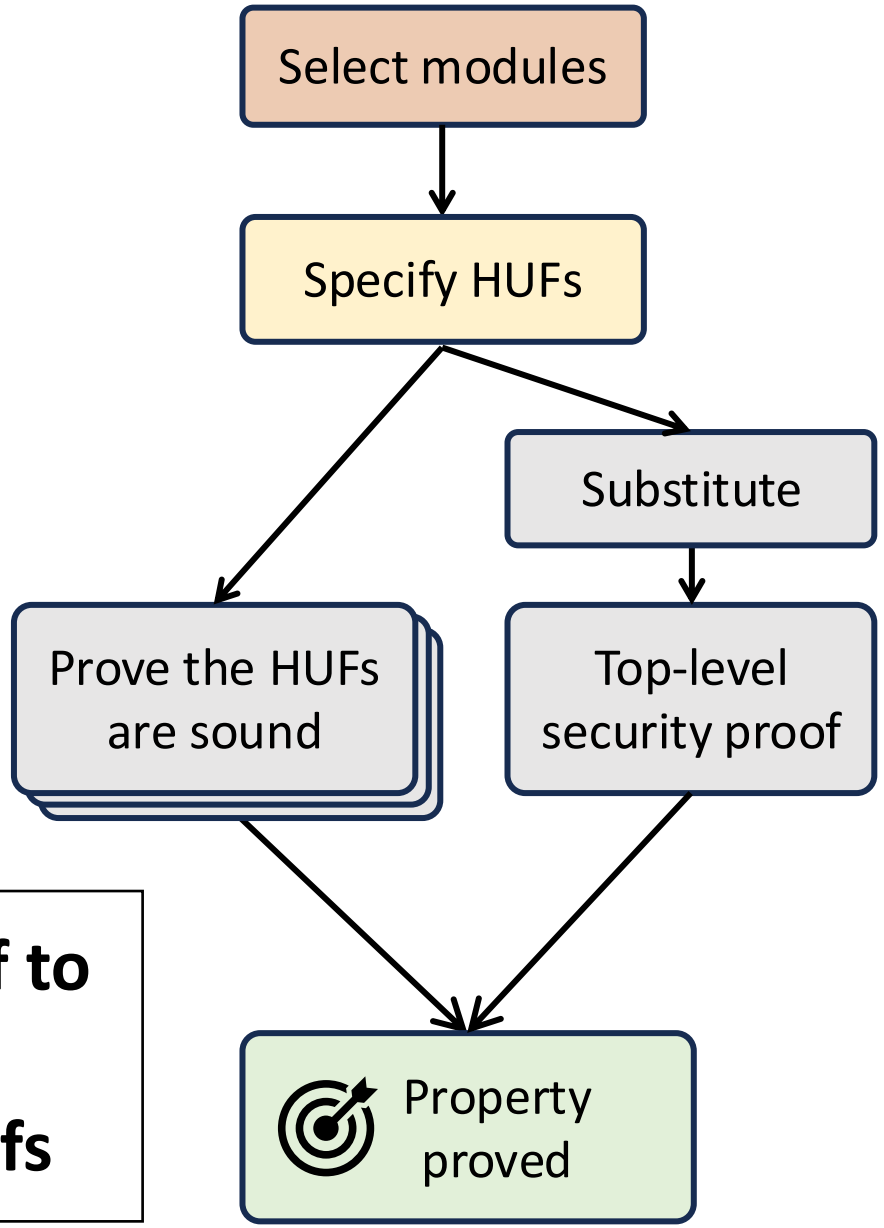




HUF Verification Flow

Verification target:
Self-composed processor pair





HUF decomposes a large monolithic proof to

- **one smaller top-level property proof**
- **multiple module-level soundness proofs**

Verification time is obtained using the open-source model checker rIC3

Evaluation Results

Verification time is obtained using the open-source model checker rIC3

Evaluation Results

Property 1: Constant-Time Instruction Set

Verification time is obtained using the open-source model checker rIC3

Evaluation Results

Property 1: Constant-Time Instruction Set

H-Houdini (160 cores)	**7 mins**	**3 hours**

Verification time is obtained using the open-source model checker rIC3

Evaluation Results

Property 1: Constant-Time Instruction Set

		
H-Houdini (160 cores)	**7 mins**	**3 hours**
HUF **(11 cores)**	**30 s** **(14x faster)**	**15 mins** **(12x faster)**

Verification time is obtained using the open-source model checker rIC3

Evaluation Results

Property 1: Constant-Time Instruction Set

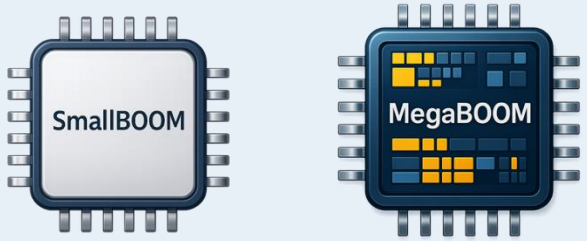
H-Houdini (160 cores)	**7 mins**	**3 hours**
HUF **(11 cores)**	**30 s** **(14x faster)**	**15 mins** **(12x faster)**

Property 2: Secure Speculation

Verification time is obtained using the open-source model checker rIC3

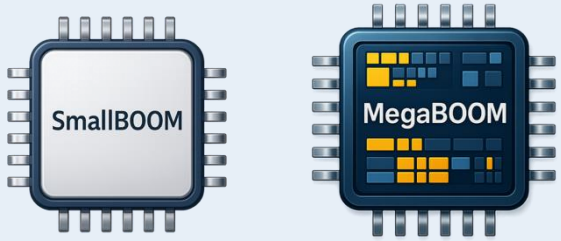
Evaluation Results

Property 1: Constant-Time Instruction Set



H-Houdini (160 cores)	**7 mins**	**3 hours**
HUF **(11 cores)**	**30 s** **(14x faster)**	**15 mins** **(12x faster)**

Property 2: Secure Speculation

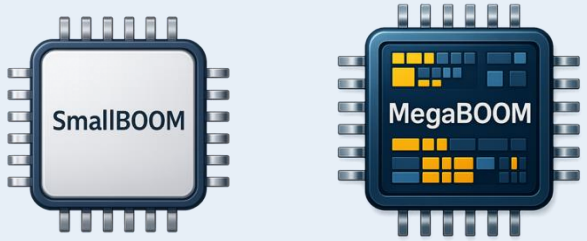


Contract Shadow Logic	✗ **Cannot verify**	✗

Verification time is obtained using the open-source model checker rIC3

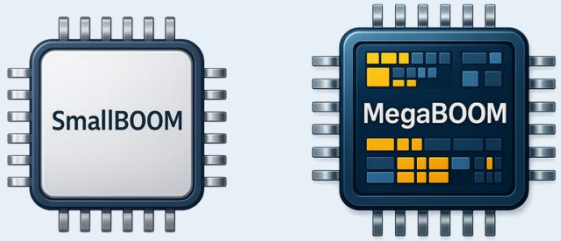
Evaluation Results

Property 1: Constant-Time Instruction Set



	SmallBOOM	MegaBOOM
H-Houdini (160 cores)	**7 mins**	**3 hours**
HUF **(11 cores)**	**30 s** **(14x faster)**	**15 mins** **(12x faster)**

Property 2: Secure Speculation

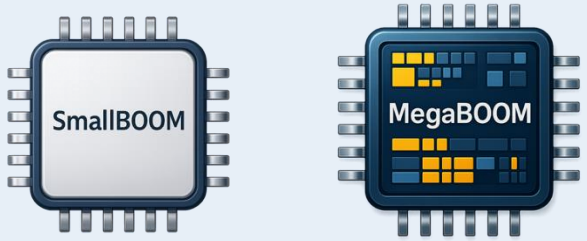


	SmallBOOM	MegaBOOM
Contract Shadow Logic	✗ **Cannot verify**	✗
HUF	✓ **3 mins**	✓ **3 hours**

Verification time is obtained using the open-source model checker rIC3

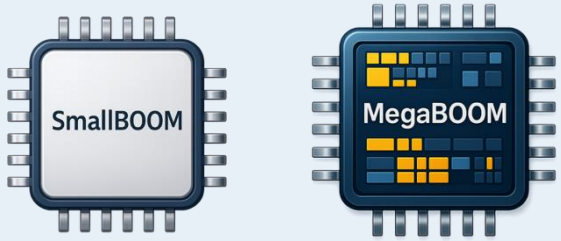
Evaluation Results

Property 1: Constant-Time Instruction Set



	SmallBOOM	MegaBOOM
H-Houdini (160 cores)	**7 mins**	**3 hours**
HUF **(11 cores)**	**30 s** **(14x faster)**	**15 mins** **(12x faster)**

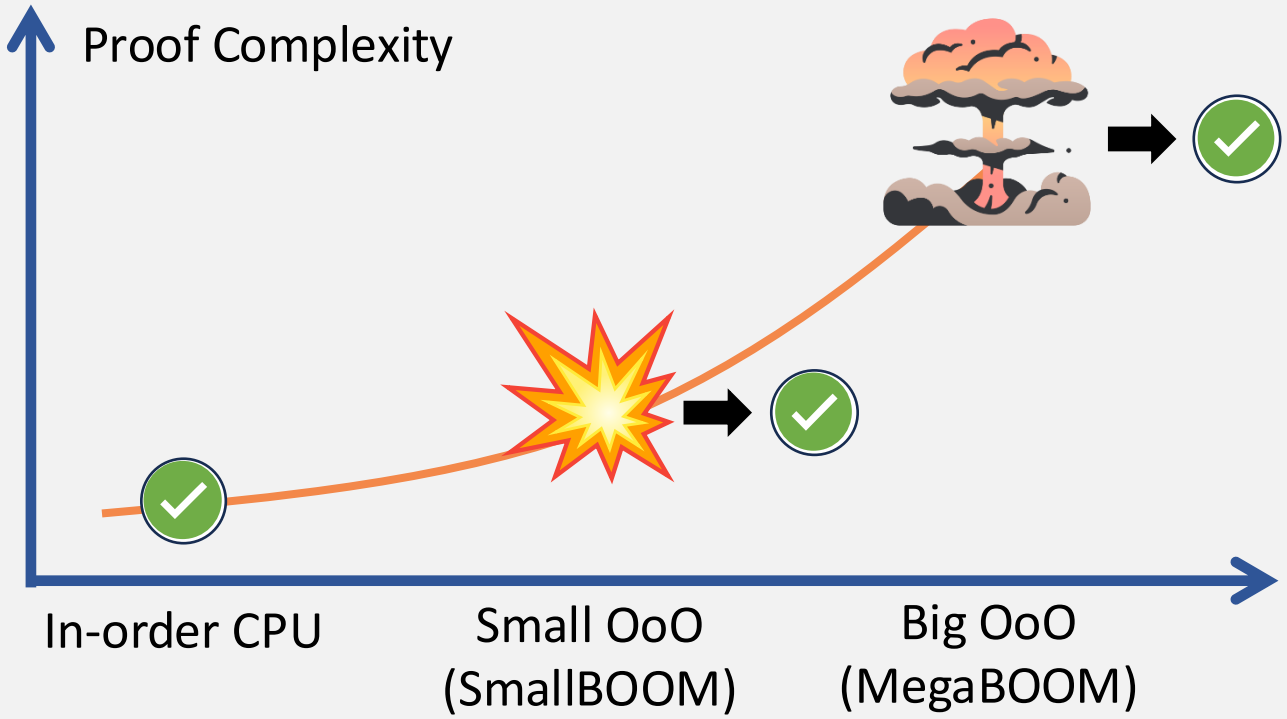
Property 2: Secure Speculation



	SmallBOOM	MegaBOOM
Contract Shadow Logic	✗ **Cannot verify**	✗
HUF	✓ **3 mins**	✓ **3 hours**

HUF scales verification to previously out-of-reach complex OoO CPU

HUF: Uninterpreted Function with History



Thank you!

rogerdtz@mit.edu

Try our artifact!

GitHub.com/MATCHA-MIT/shuffle

Massachusetts Institute of Technology
PRINCETON UNIVERSITY
EPFL
MIT CSAIL
WATCHA