Sound and Automated Deductive Verifiers for Advanced Properties

Thibault Dardinier

ETH zürich

Software Bugs

Software Bugs

Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error means that websites can leak user details including passwords through 'heartbeat' function used to secure connections

Software Bugs

Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error means that websites can leak user details including passwords through 'heartbeat' function used to secure connections

Log4shell software flaw threatens millions of servers as hackers scramble to exploit it

Internet

Sat 11 Dec 2021

The extreme ease the flaw allows an attacker to access a server is what experts say makes it so dangerous. *(Supplied: Accenture)*

Software Bugs

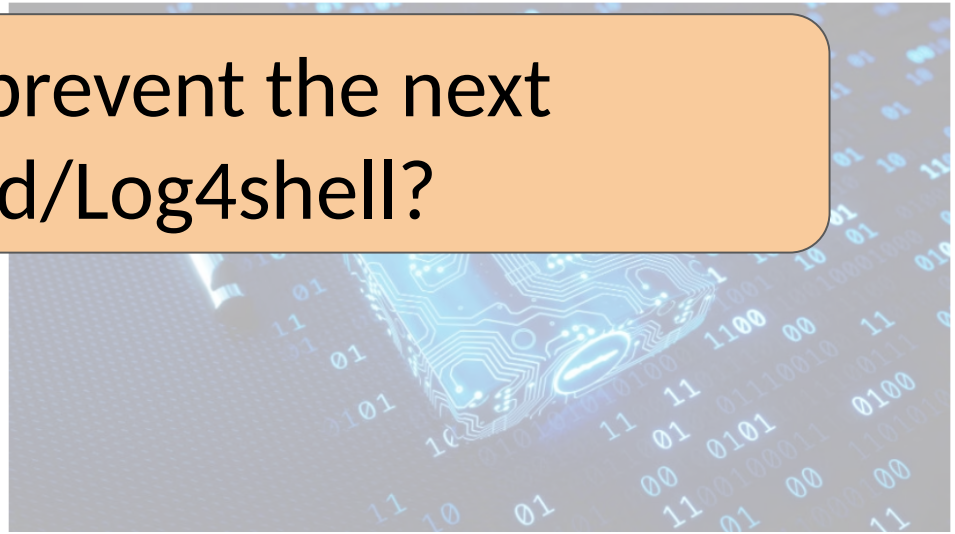
Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error means that websites can leak user details including passwords through 'heartbeat' function used to secure co

Log4shell software flaw threatens millions of servers as hackers scramble to exploit it

Internet

Sat 11 Dec 2021



The extreme ease the flaw allows an attacker to access a server is what experts say makes it so dangerous. *(Supplied: Accenture)*

How can we prevent the next Heartbleed/Log4shell?

Preventing the Next Heartbleed/Log4shell

Testing

Preventing the Next Heartbleed/Log4shell

Testing

Fuzzing

Preventing the Next Heartbleed/Log4shell

Testing

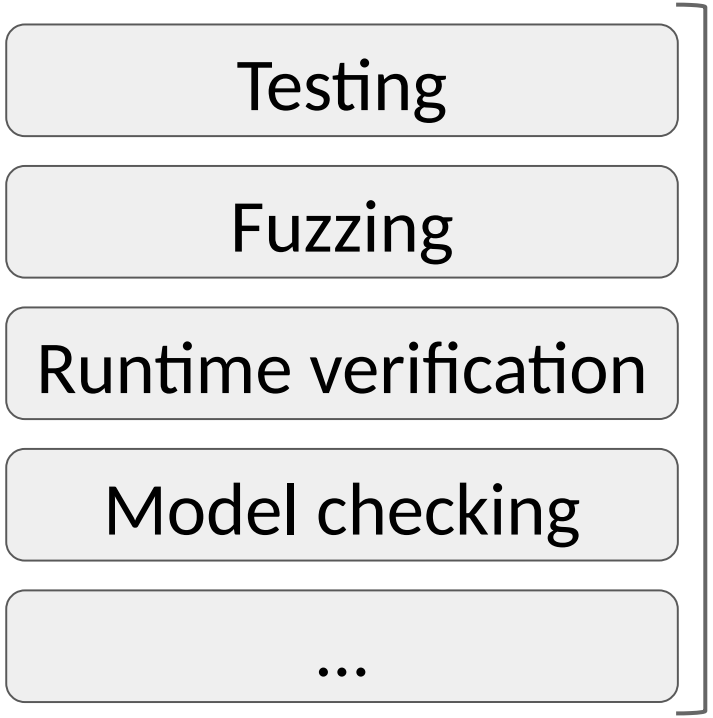
Fuzzing

Runtime verification

Preventing the Next Heartbleed/Log4shell

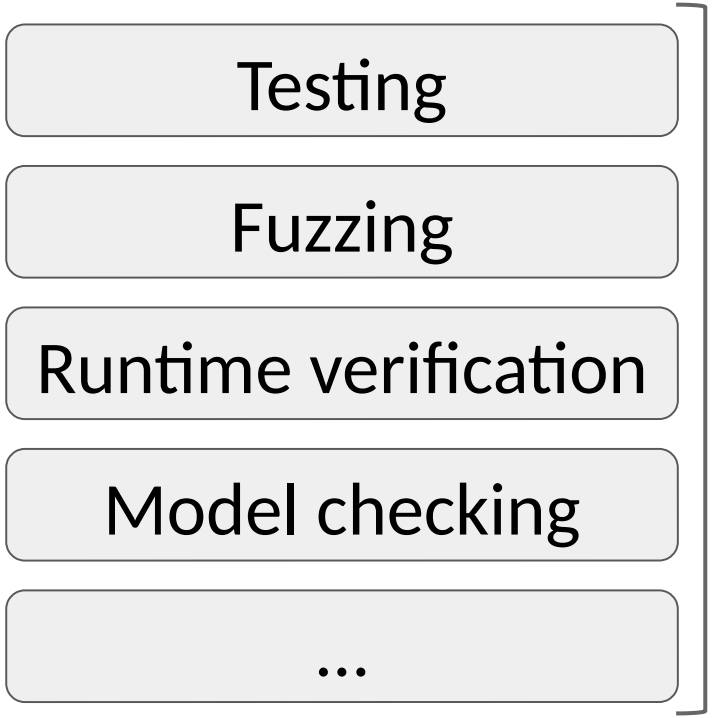
- Testing
- Fuzzing
- Runtime verification
- Model checking

Preventing the Next Heartbleed/Log4shell



can show the **presence** of bugs

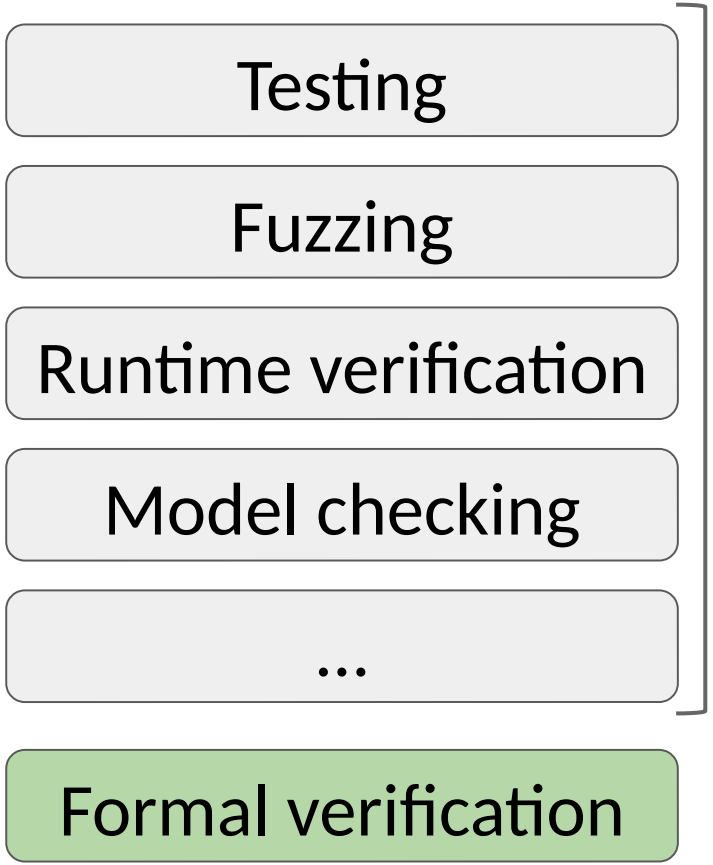
Preventing the Next Heartbleed/Log4shell



Might miss bugs and vulnerabilites that attackers can exploit

can show the **presence** of bugs

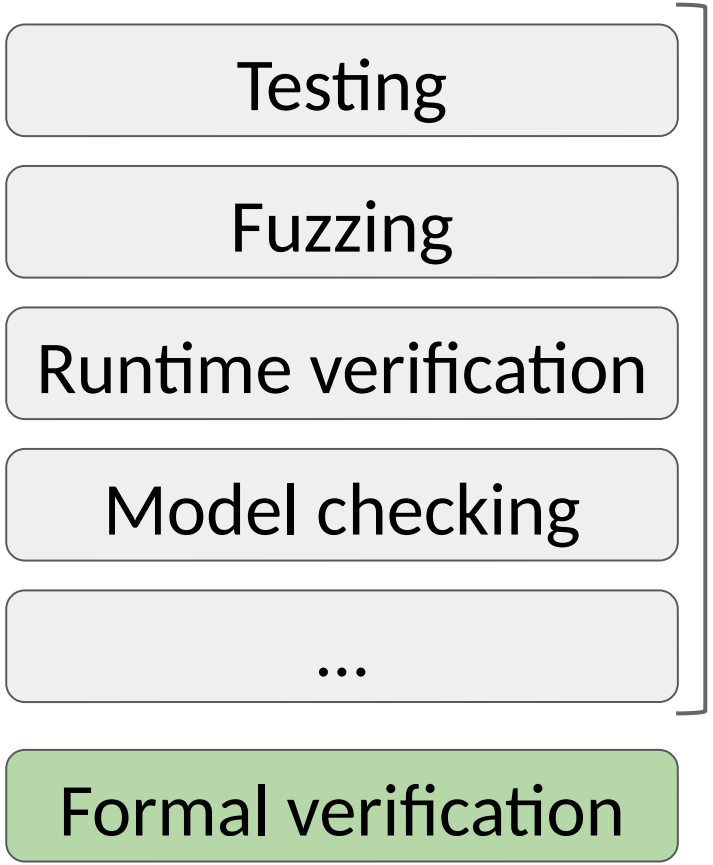
Preventing the Next Heartbleed/Log4shell



can show the **presence** of bugs

can show the **absence** of bugs

Preventing the Next Heartbleed/Log4shell



can show the **presence** of bugs

Can formally prove
the absence of classes of attacks

can show the **absence** of bugs

Preventing the Next Heartbleed/Log4shell

Testing

Fuzzing

Runtime verification

Model checking

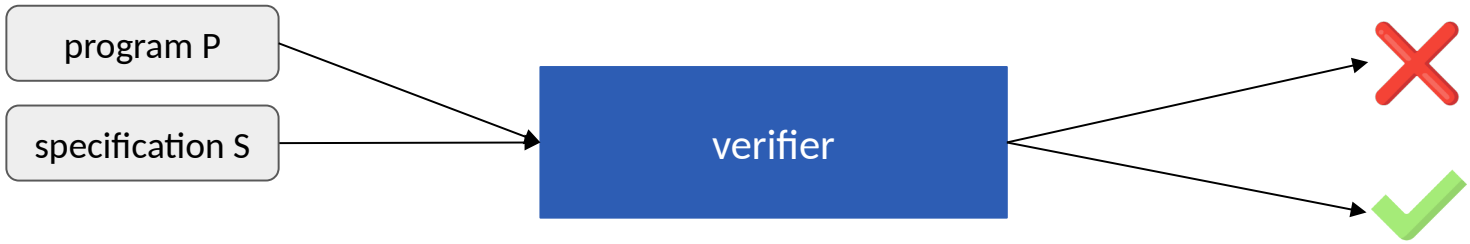
…

can show the **presence** of bugs

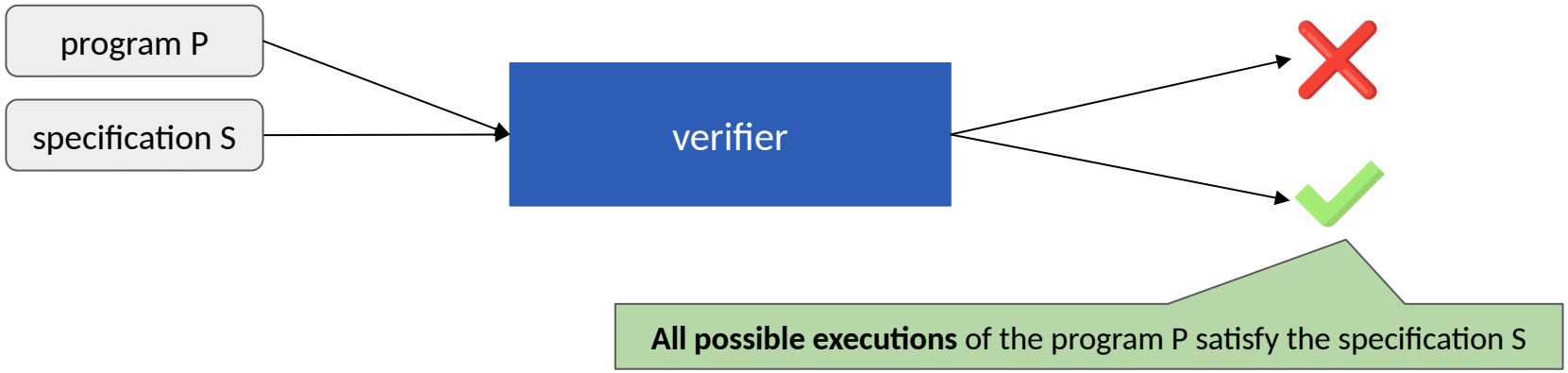
Formal verification

can show the **absence** of bugs

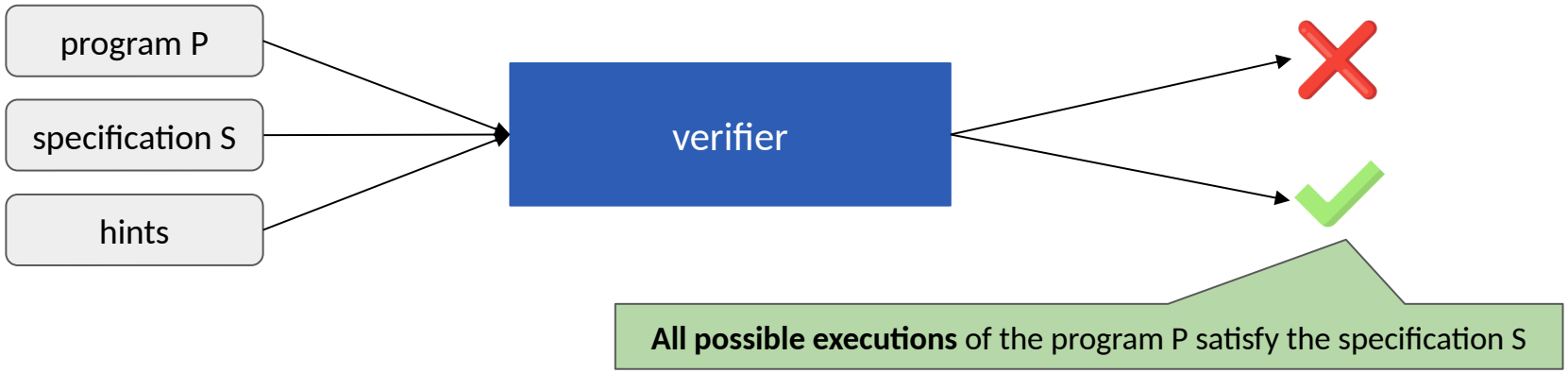
Automated Deductive Verifiers



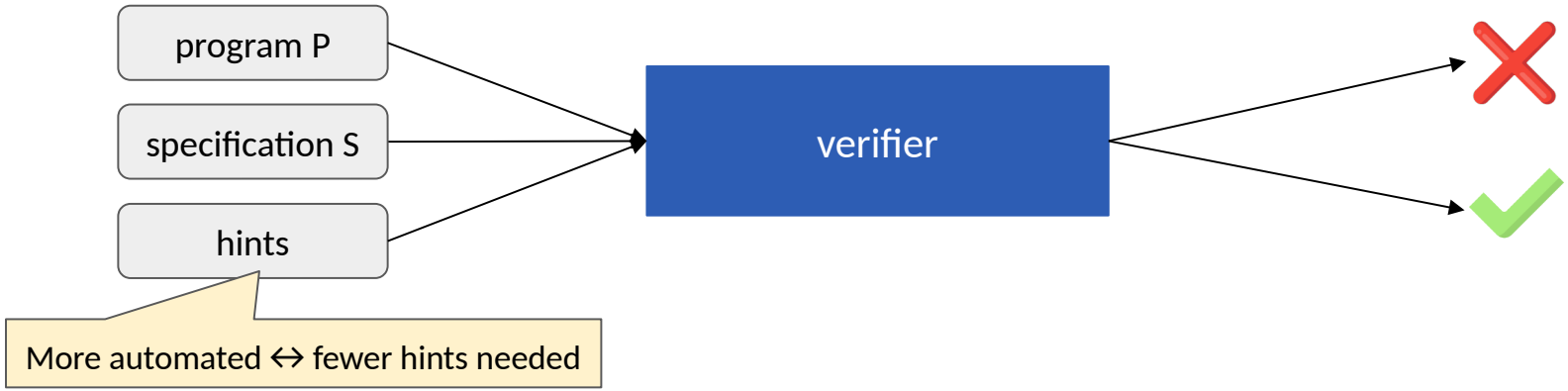
Automated Deductive Verifiers



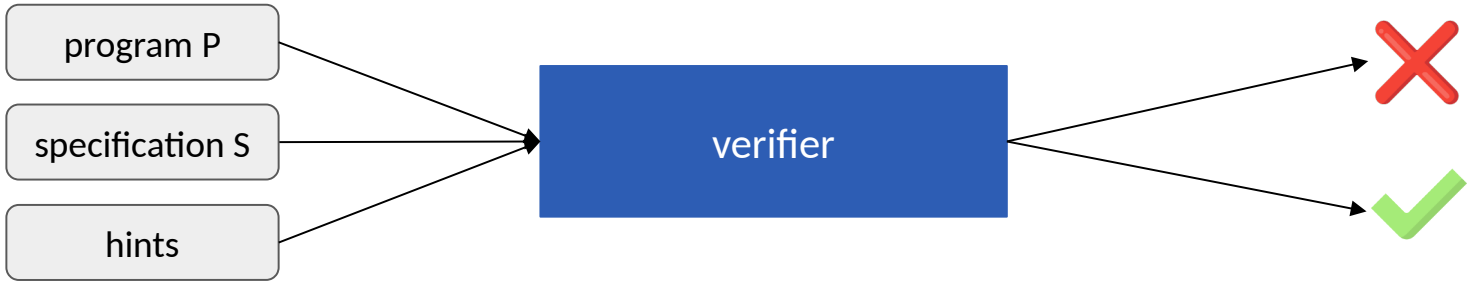
Automated Deductive Verifiers



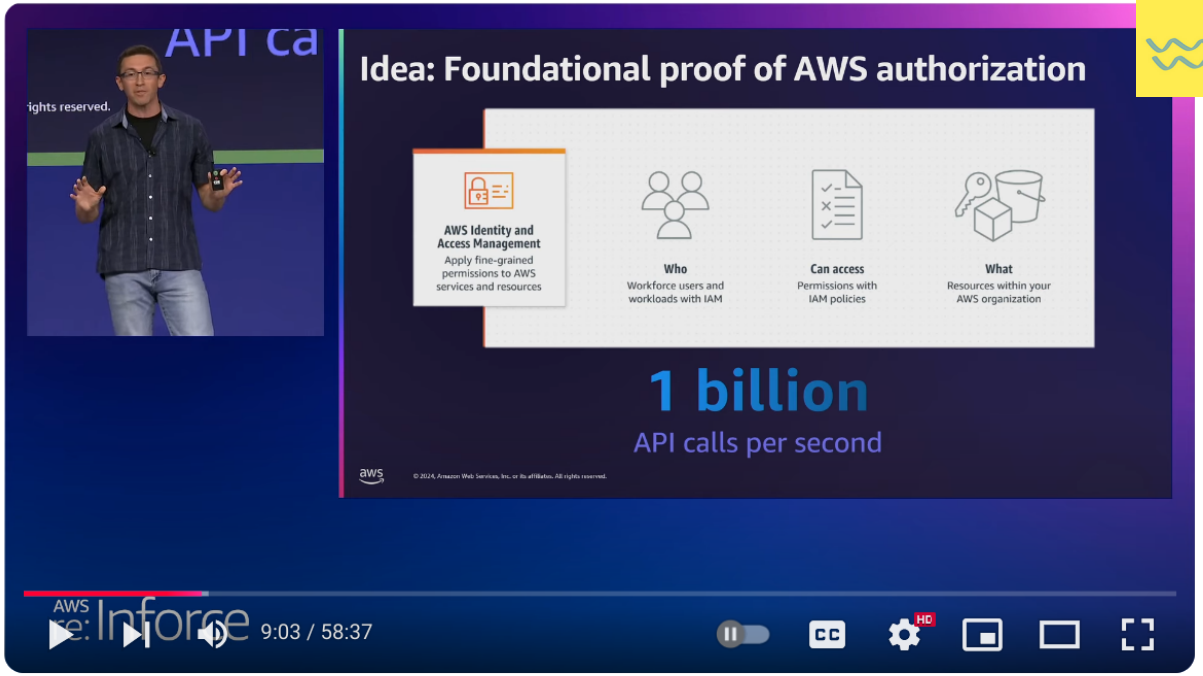
Automated Deductive Verifiers



Automated Deductive Verifiers

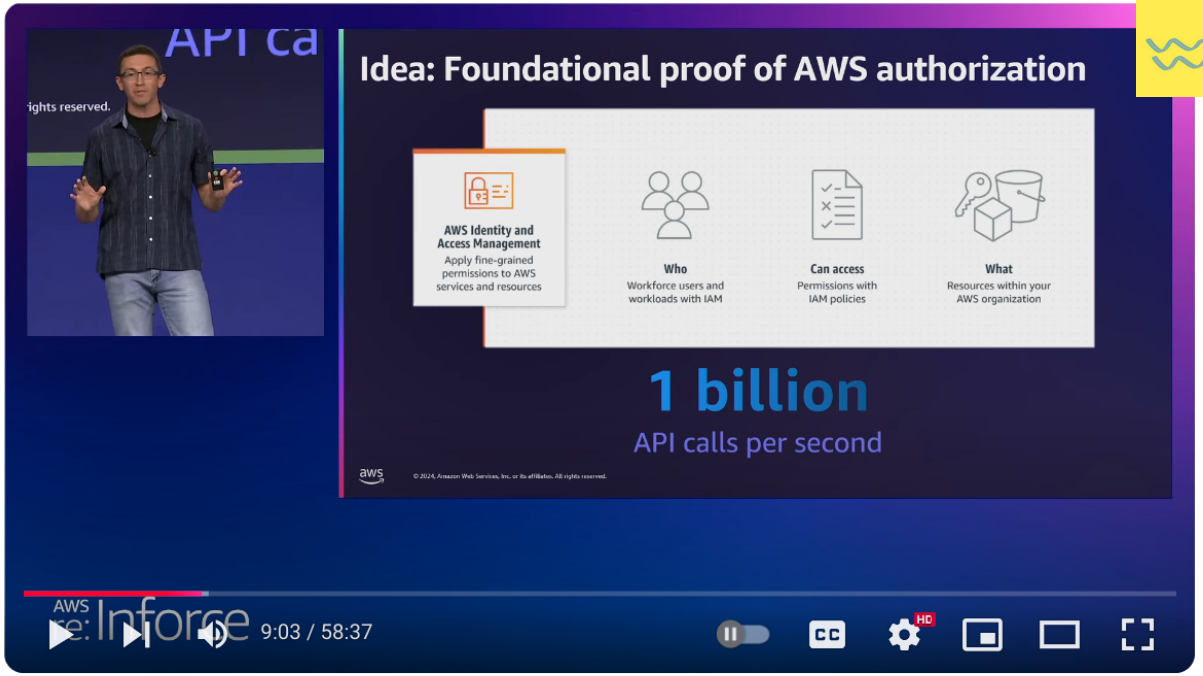


Modern Verifiers are Practical



AWS re:Inforce 2024 - Proving the correctness of AWS authorization (IAM401)

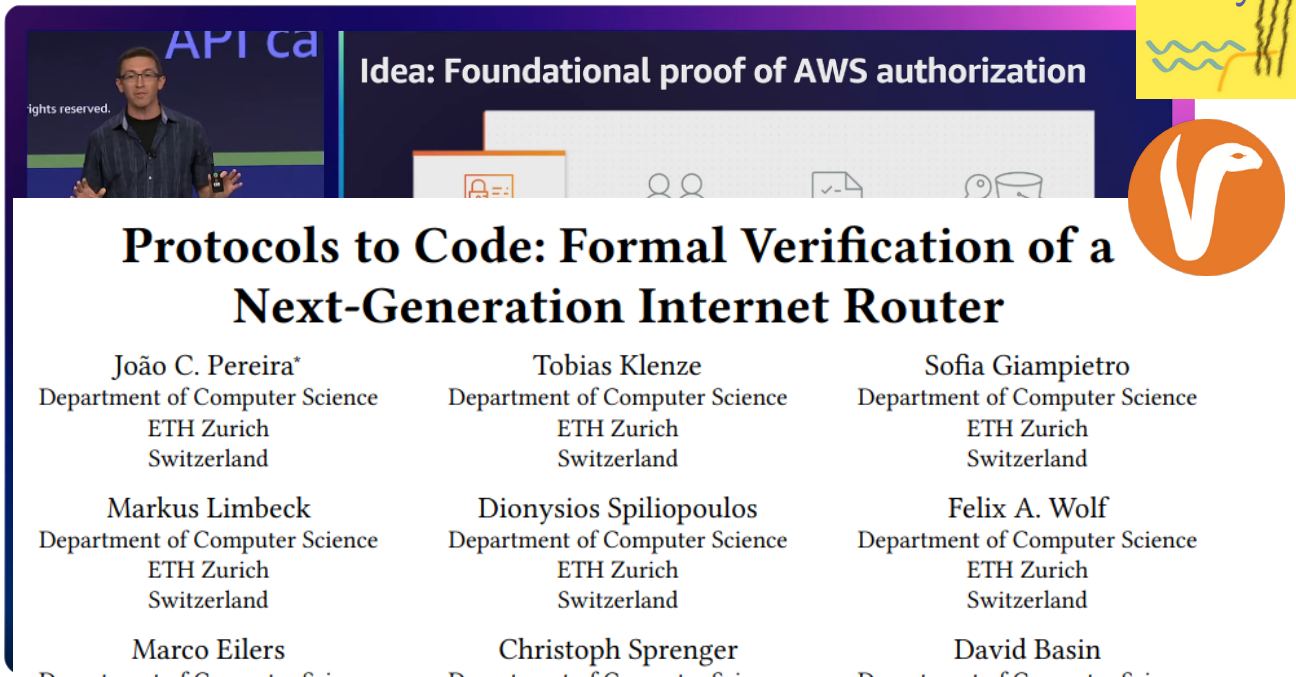
Modern Verifiers are Practical



AWS re:Inforce 2024 - Proving the correctness of AWS authorization (IAM401)

1 billion times/second
3,000 lines of code
3x performance

Modern Verifiers are Practical



Protocols to Code: Formal Verification of a Next-Generation Internet Router

João C. Pereira*
Department of Computer Science
ETH Zurich
Switzerland

Tobias Klenze
Department of Computer Science
ETH Zurich
Switzerland

Sofia Giampietro
Department of Computer Science
ETH Zurich
Switzerland

Markus Limbeck
Department of Computer Science
ETH Zurich
Switzerland

Dionysios Spiliopoulos
Department of Computer Science
ETH Zurich
Switzerland

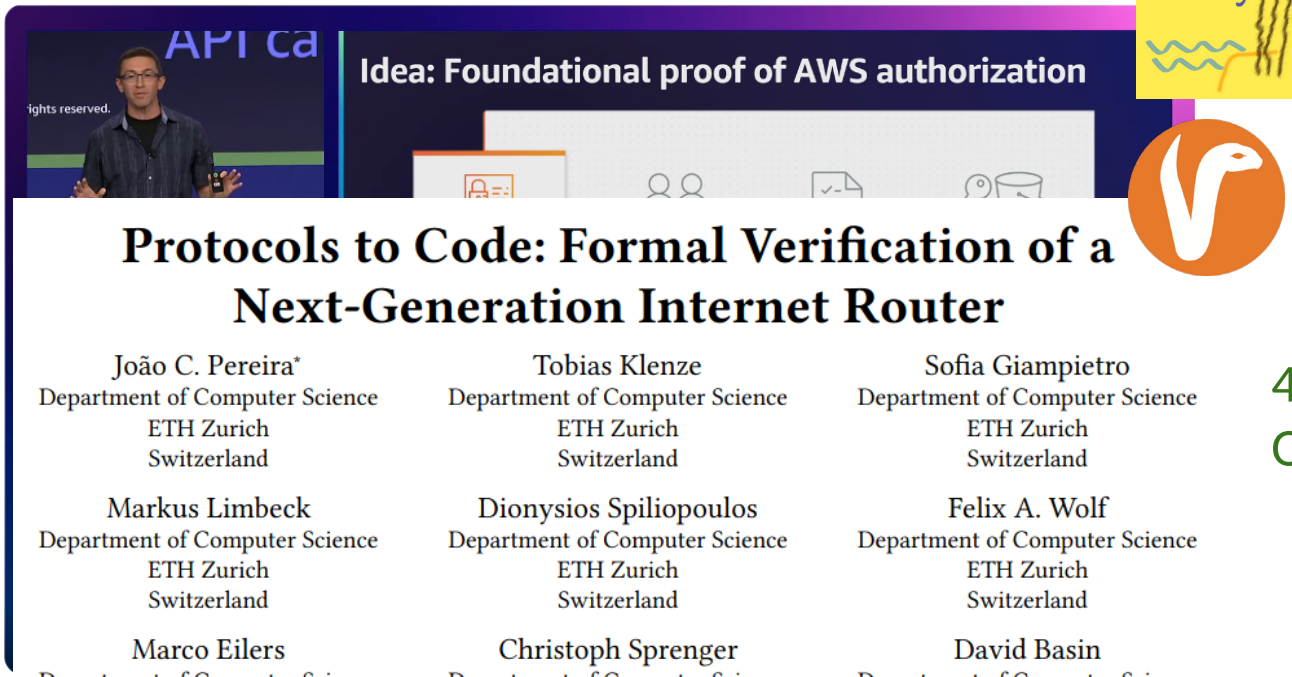
Felix A. Wolf
Department of Computer Science
ETH Zurich
Switzerland

Marco Eilers
Department of Computer Science
ETH Zurich
Switzerland

Christoph Sprenger
Department of Computer Science
ETH Zurich
Switzerland

David Basin
Department of Computer Science
ETH Zurich
Switzerland

Modern Verifiers are Practical



Protocols to Code: Formal Verification of a Next-Generation Internet Router

João C. Pereira*
Department of Computer Science
ETH Zurich
Switzerland

Tobias Klenze
Department of Computer Science
ETH Zurich
Switzerland

Sofia Giampietro
Department of Computer Science
ETH Zurich
Switzerland

Markus Limbeck
Department of Computer Science
ETH Zurich
Switzerland

Dionysios Spiliopoulos
Department of Computer Science
ETH Zurich
Switzerland

Felix A. Wolf
Department of Computer Science
ETH Zurich
Switzerland

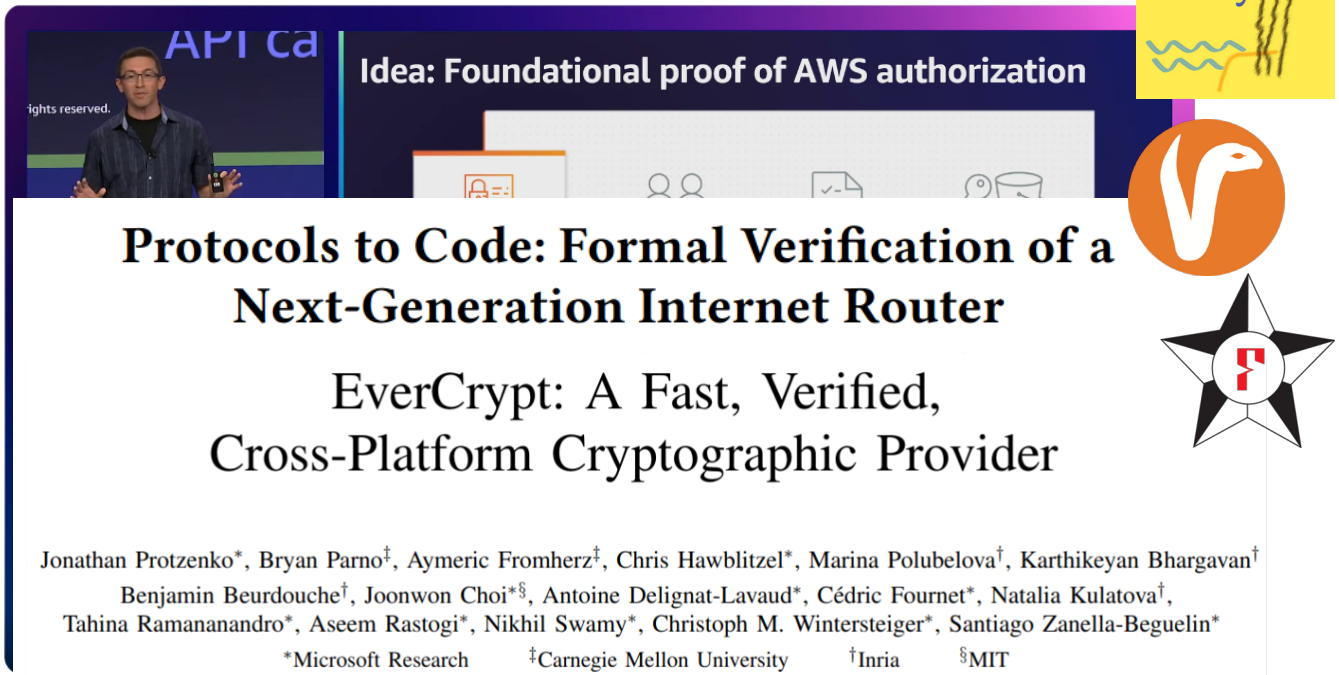
Marco Eilers
Department of Computer Science
ETH Zurich
Switzerland

Christoph Sprenger
Department of Computer Science
ETH Zurich
Switzerland

David Basin
Department of Computer Science
ETH Zurich
Switzerland

4,700 lines of Go code
Optimized implementation

Modern Verifiers are Practical



Protocols to Code: Formal Verification of a Next-Generation Internet Router

EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider

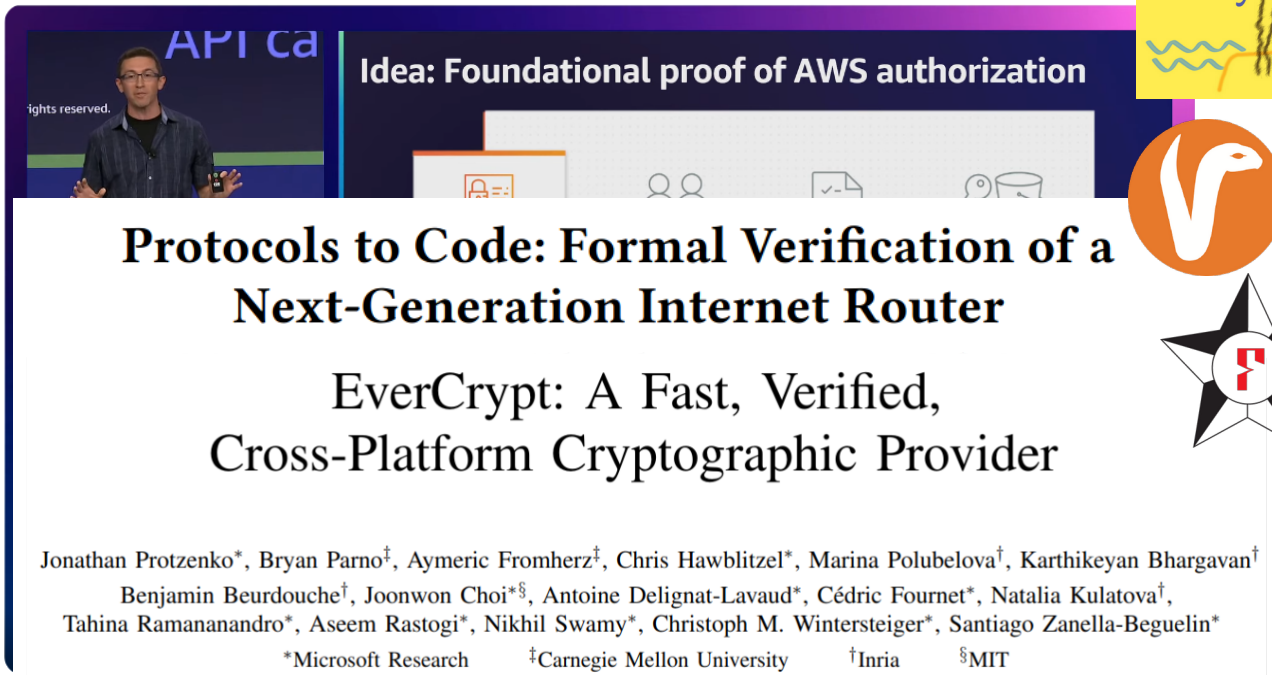
Jonathan Protzenko*, Bryan Parno‡, Aymeric Fromherz‡, Chris Hawblitzel*, Marina Polubelova†, Karthikeyan Bhargavan†
Benjamin Beurdouche†, Joonwon Choi*§, Antoine Delignat-Lavaud*, Cédric Fournet*, Natalia Kulatova†,
Tahina Ramananandro*, Aseem Rastogi*, Nikhil Swamy*, Christoph M. Wintersteiger*, Santiago Zanella-Beguelin*
*Microsoft Research ‡Carnegie Mellon University †Inria §MIT

ETH Zurich
Switzerland

ETH Zurich
Switzerland

ETH Zurich
Switzerland

Modern Verifiers are Practical



Protocols to Code: Formal Verification of a Next-Generation Internet Router

EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider

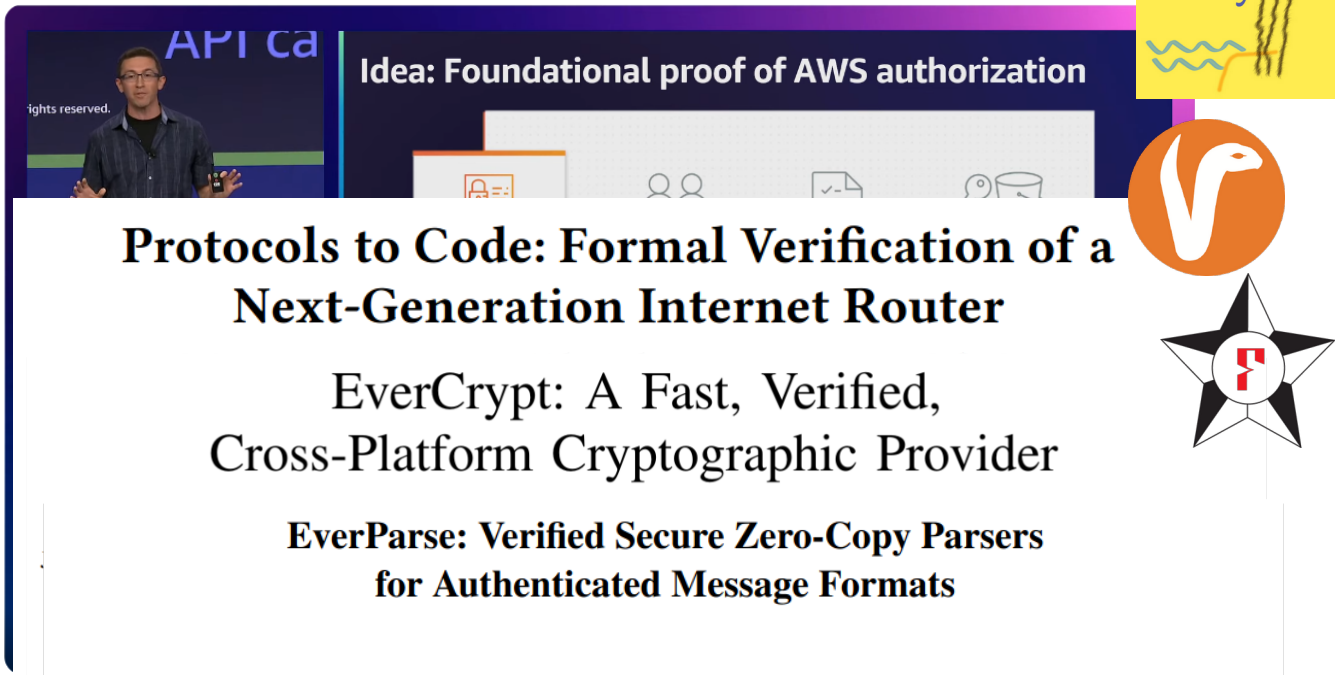
Jonathan Protzenko*, Bryan Parno‡, Aymeric Fromherz‡, Chris Hawblitzel*, Marina Polubelova†, Karthikeyan Bhargavan†
Benjamin Beurdouche†, Joonwon Choi*§, Antoine Delignat-Lavaud*, Cédric Fournet*, Natalia Kulatova†,
Tahina Ramananandro*, Aseem Rastogi*, Nikhil Swamy*, Christoph M. Wintersteiger*, Santiago Zanella-Beguelin*
*Microsoft Research ‡Carnegie Mellon University †Inria §MIT

ETH Zurich Switzerland
ETH Zurich Switzerland
ETH Zurich Switzerland

27,000 LoC (C code)
Deployed in

- Firefox
- Linux kernel
- Python (3.12)

Modern Verifiers are Practical



Protocols to Code: Formal Verification of a Next-Generation Internet Router

EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider

EverParse: Verified Secure Zero-Copy Parsers for Authenticated Message Formats

Tahina Ramananandro* Antoine Delignat-Lavaud* Cédric Fournet* Nikhil Swamy*
Tej Chajed† Nadim Kobeissi‡ Jonathan Protzenko*
*Microsoft Research †Massachusetts Institute of Technology ‡Inria Paris

Parses all packets in Azure

Modern Verifiers are Practical

Dafny

Can we use modern automated verifiers to prevent the next Heartbleed/Log4shell?

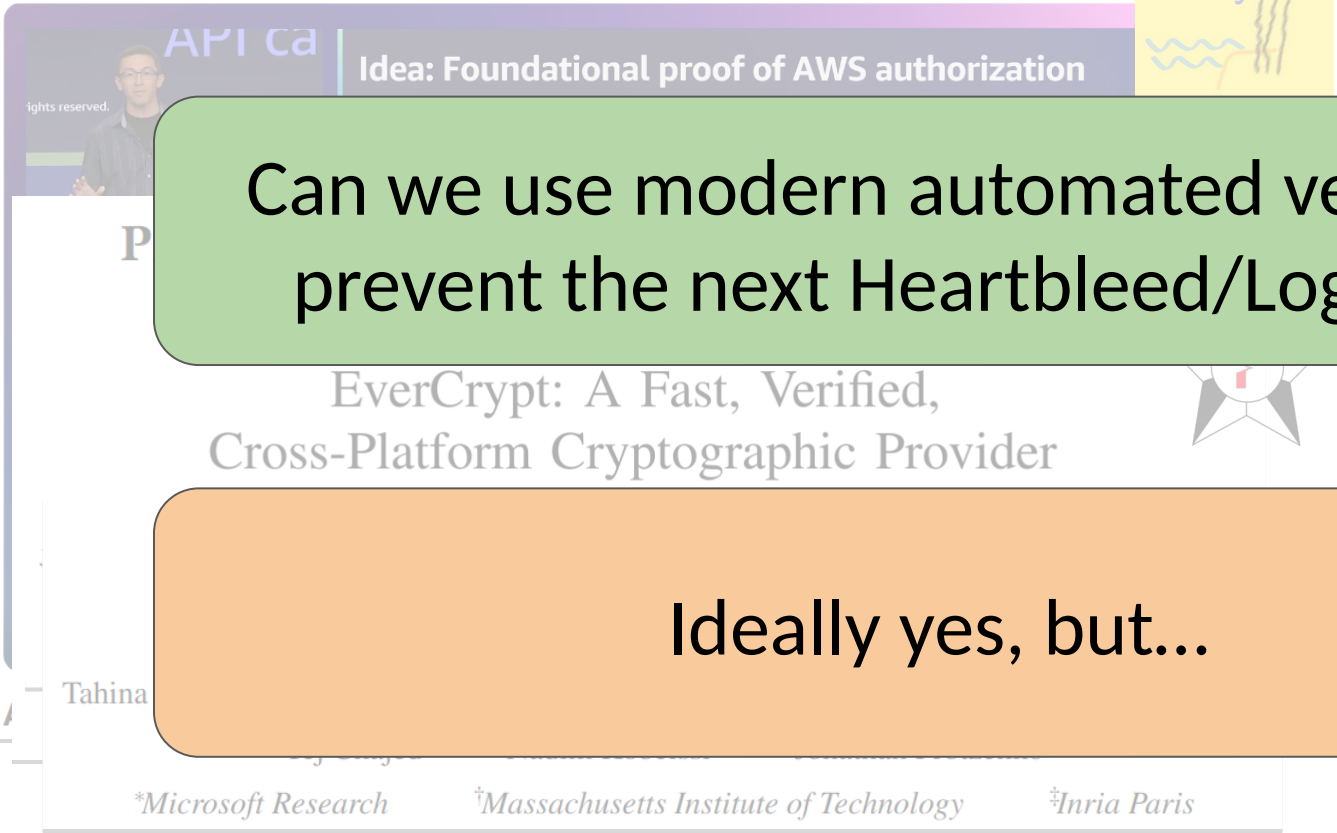
EverCrypt: A Fast, Verified, Cross-Platform Cryptographic Provider

EverParse: Verified Secure Zero-Copy Parsers for Authenticated Message Formats

Tahina Ramananandro* Antoine Delignat-Lavaud* Cédric Fournet* Nikhil Swamy*
Tej Chajed† Nadim Kobeissi‡ Jonathan Protzenko*
*Microsoft Research †Massachusetts Institute of Technology ‡Inria Paris

Parses all packets in Azure

Modern Verifiers are Practical

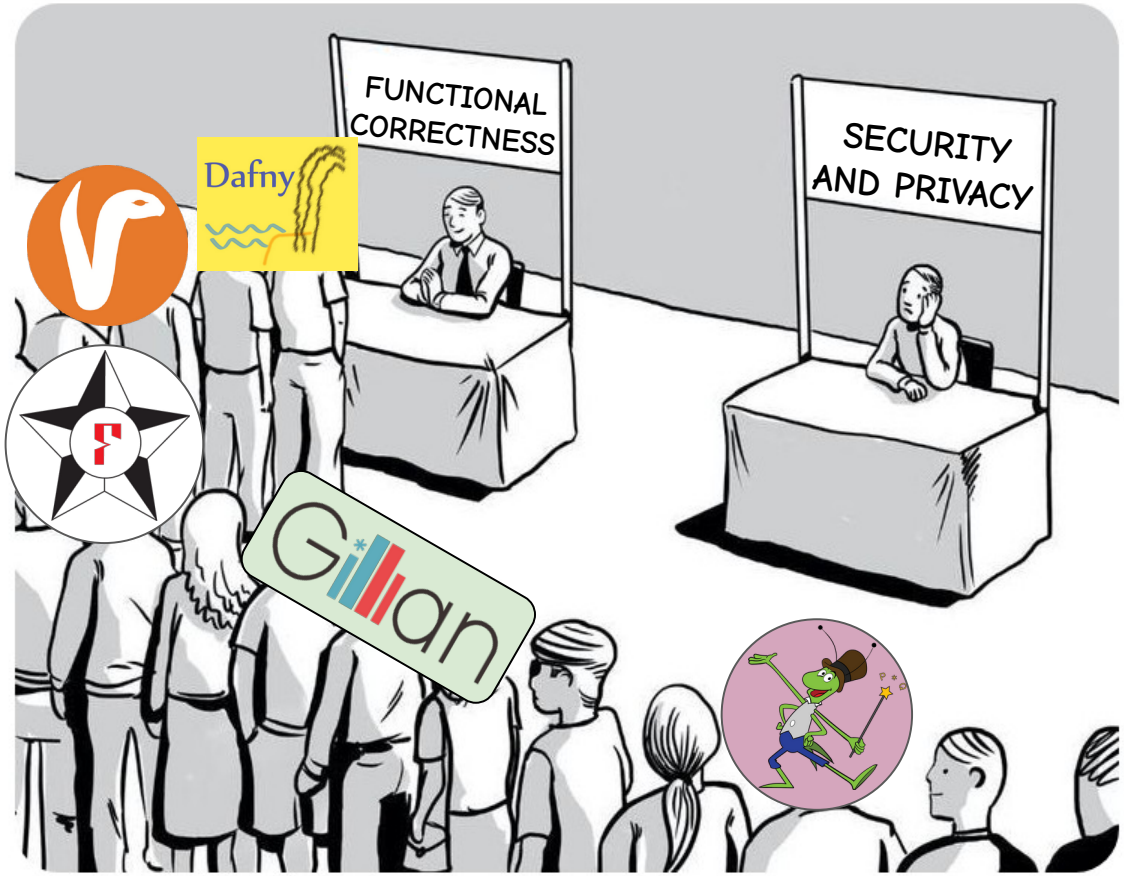


Can we use modern automated verifiers to prevent the next Heartbleed/Log4shell?

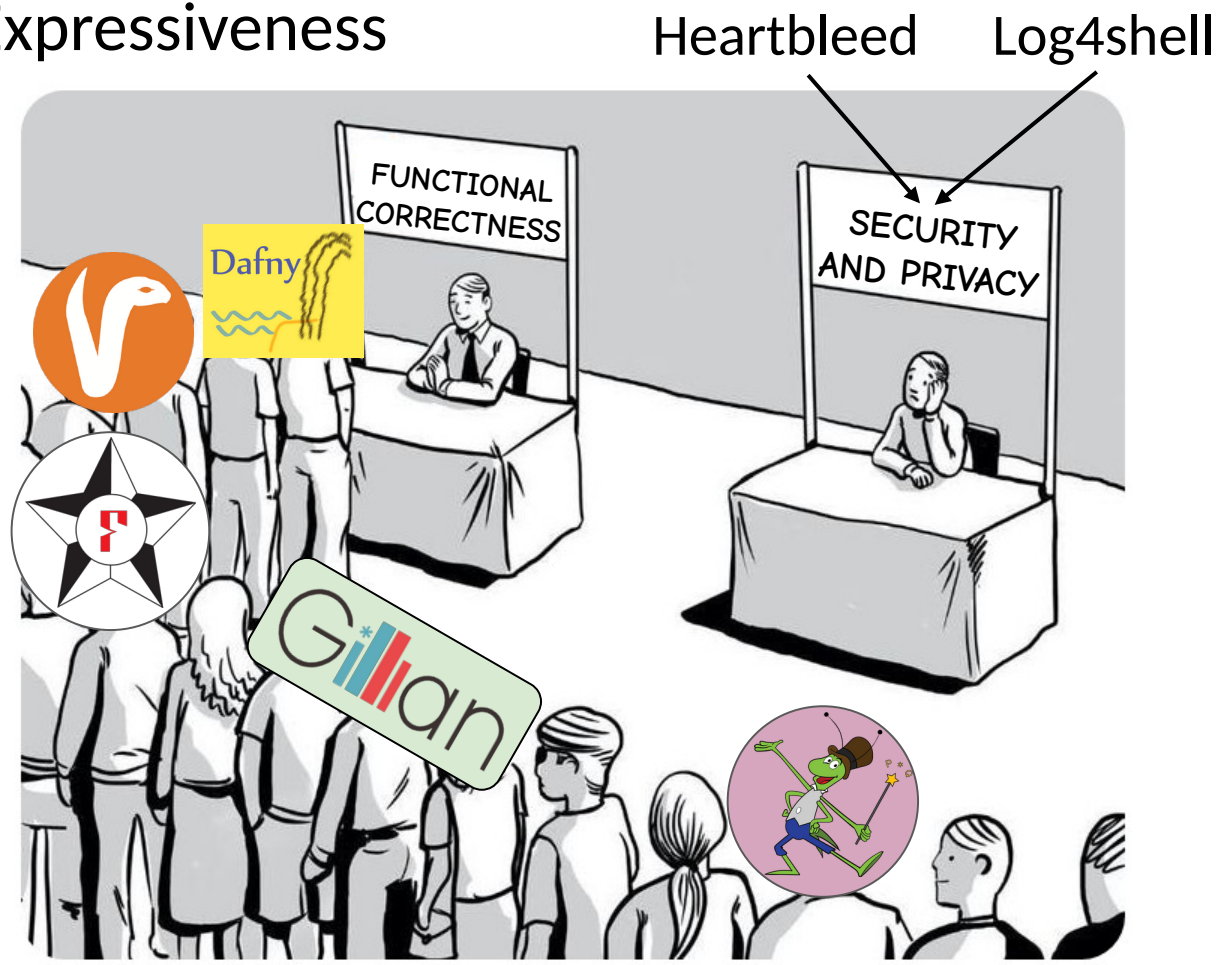
Ideally yes, but…

Azure

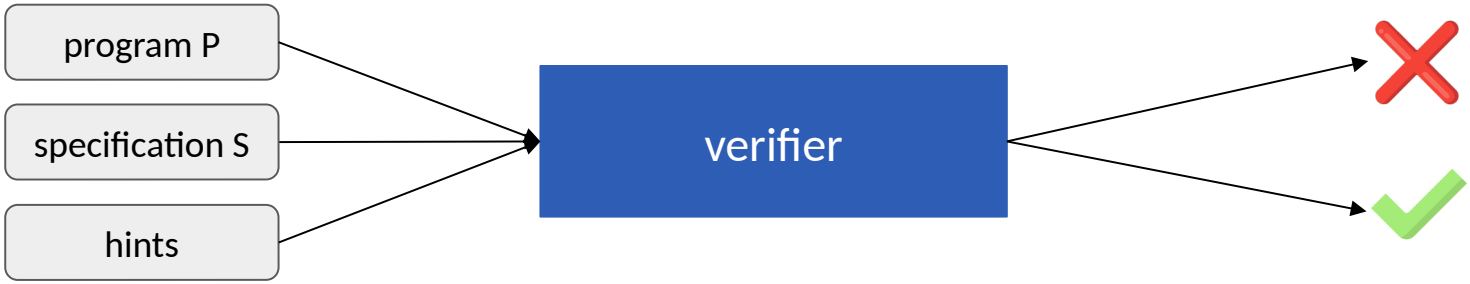
Problem: Expressiveness



Problem: Expressiveness

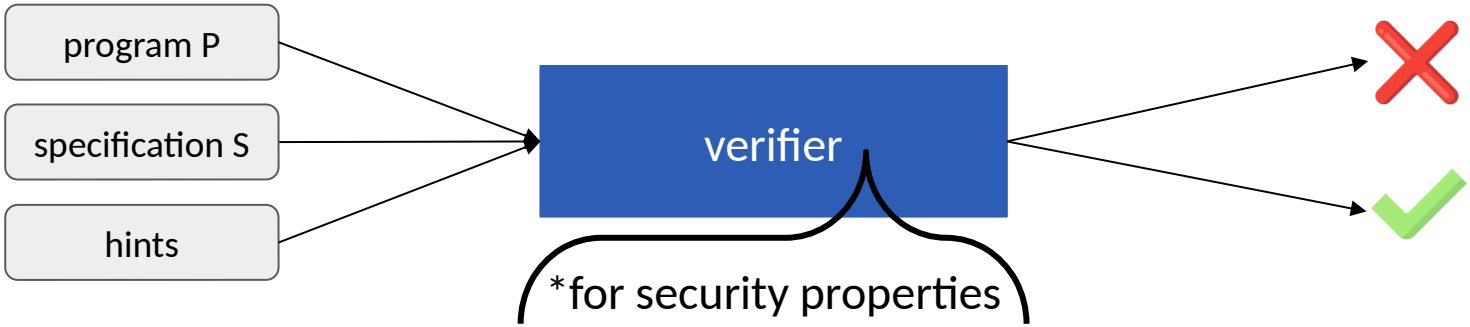


Automated Verifiers



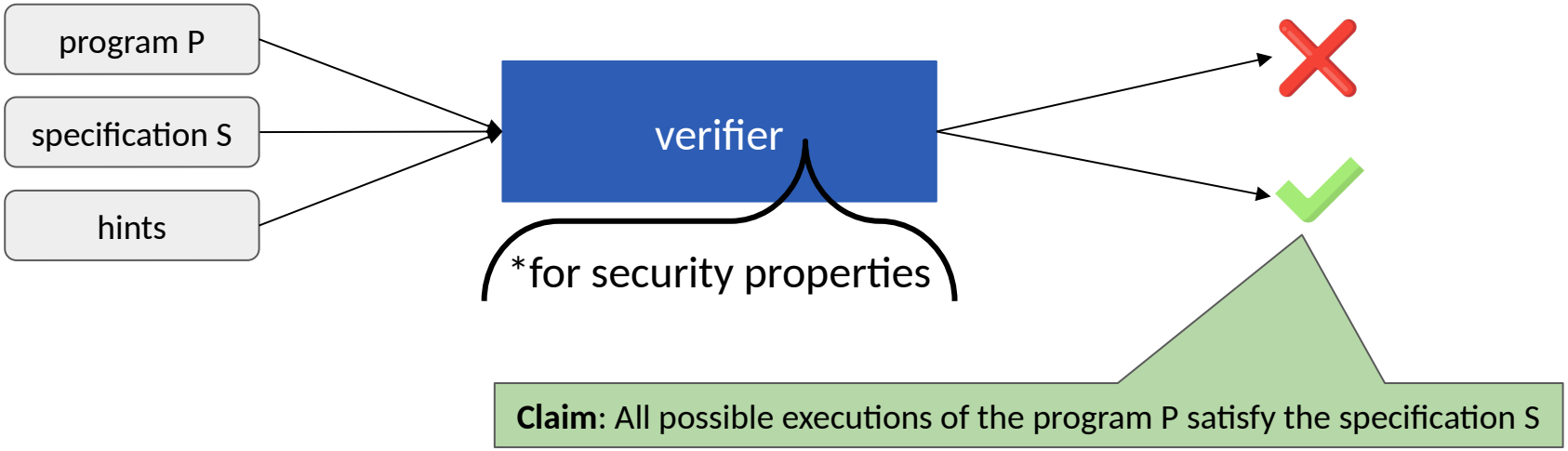
Problem
Expressiveness

Automated Verifiers



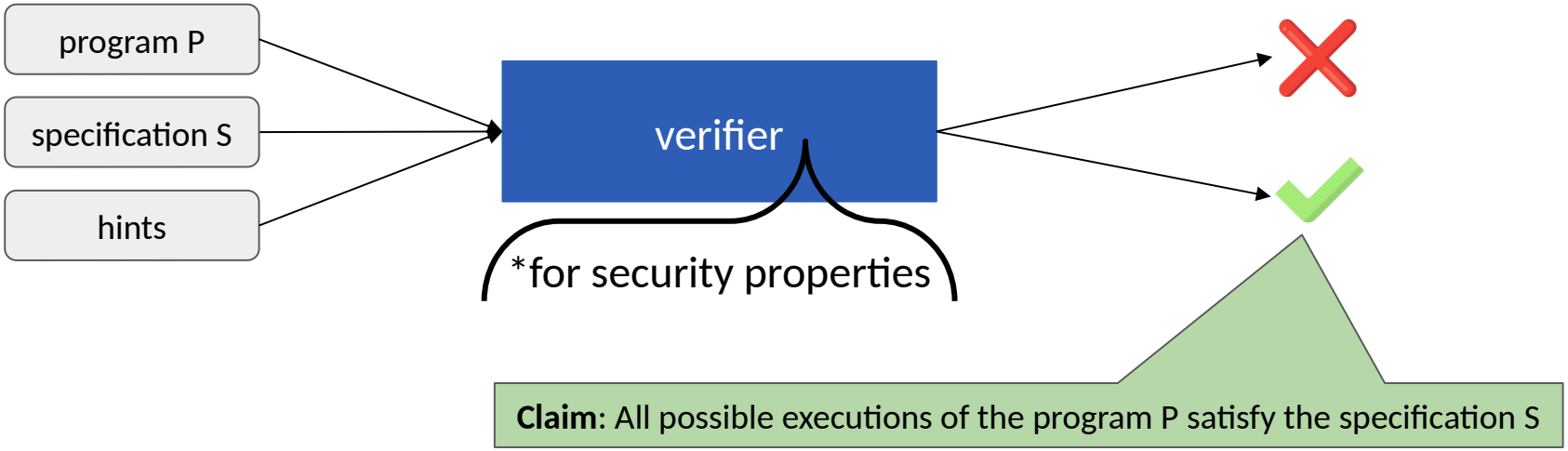
Problem
Expressiveness

Automated Verifiers



Problem
Expressiveness

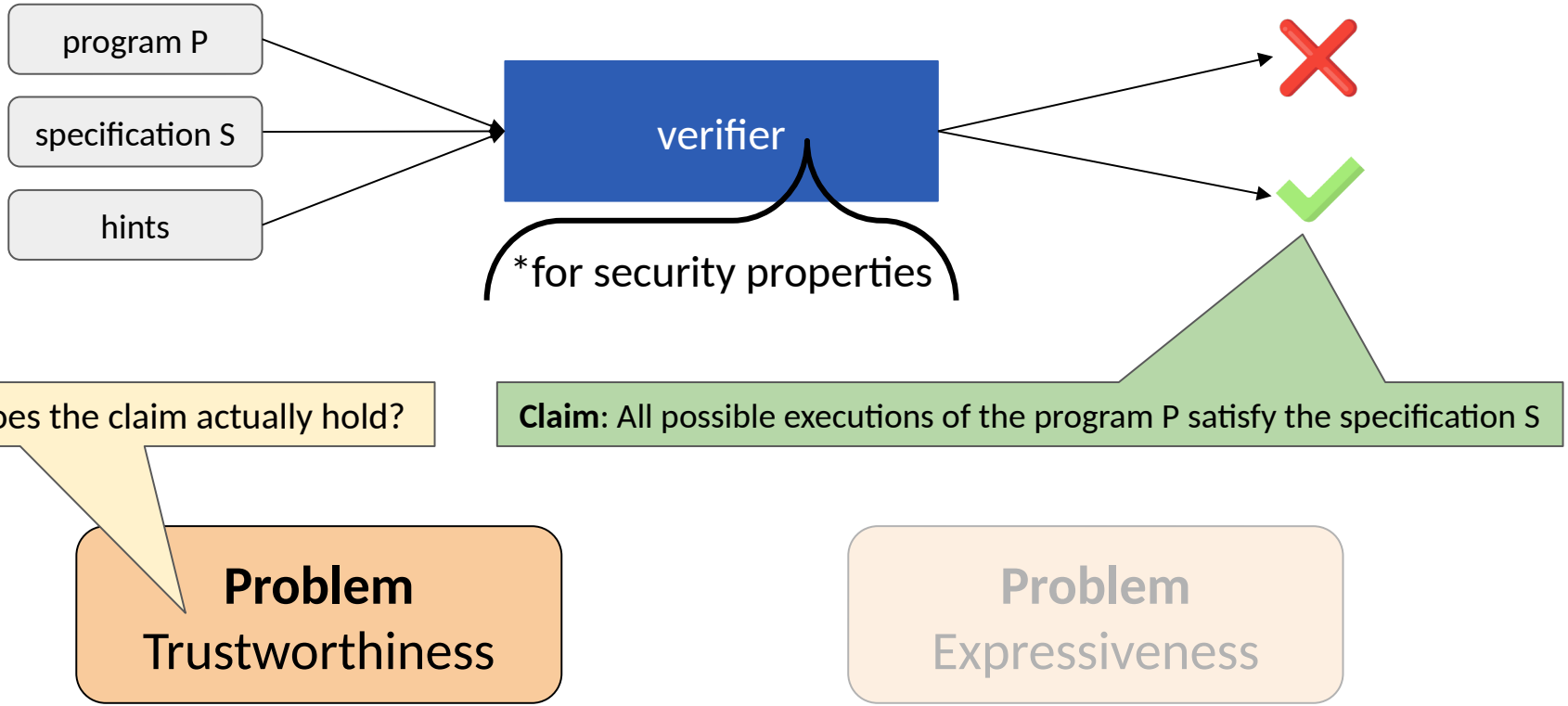
Automated Verifiers



Problem
Trustworthiness

Problem
Expressiveness

Automated Verifiers



Problem: Unsoundnesses Occur Regularly

Problem: Unsoundnesses Occur Regularly

RustanLeino opened last week

Dafny version

4.9.1

Code to produce this issue

```
method Main() {
  var c: object;
  opaque
    ensures fresh(c)
  {
    c := new object;
  }
  assert false;
}
```

Command to run and resulting output

```
% dafny verify test.dfy

Dafny program verifier finished with 1 verified, 0 errors
```

Problem: Unsoundnesses Occur Regularly

RustanLeino opened last week

Dafny version

4.9.1

Code to produce this issue

```
method Main() {
  var c: object;
  opaque
    ensures fresh(c)
  {
    c := new object;
  }
  assert false;
}
```

Command to run and resulting output

```
% dafny verify test.dfy

Dafny program verifier finished with 1 verified, 0 errors
```

Problem: Unsoundnesses Occur Regularly

RustanLeino opened last week

Dafny version

4.9.1

Code to produce this issue

```
method Main() {
  var c: object;
  opaque
    ensures fresh(c)
  {
    c := new object;
  }
  assert false;
}
```

Command to run and resulting output

```
% dafny verify test.dfy

Dafny program verifier finished with 1 verified, 0 errors
```

Problem: Unsoundnesses Occur Regularly

RustanLeino opened last week

Dafny version

4.9.1

Code to produce this issue

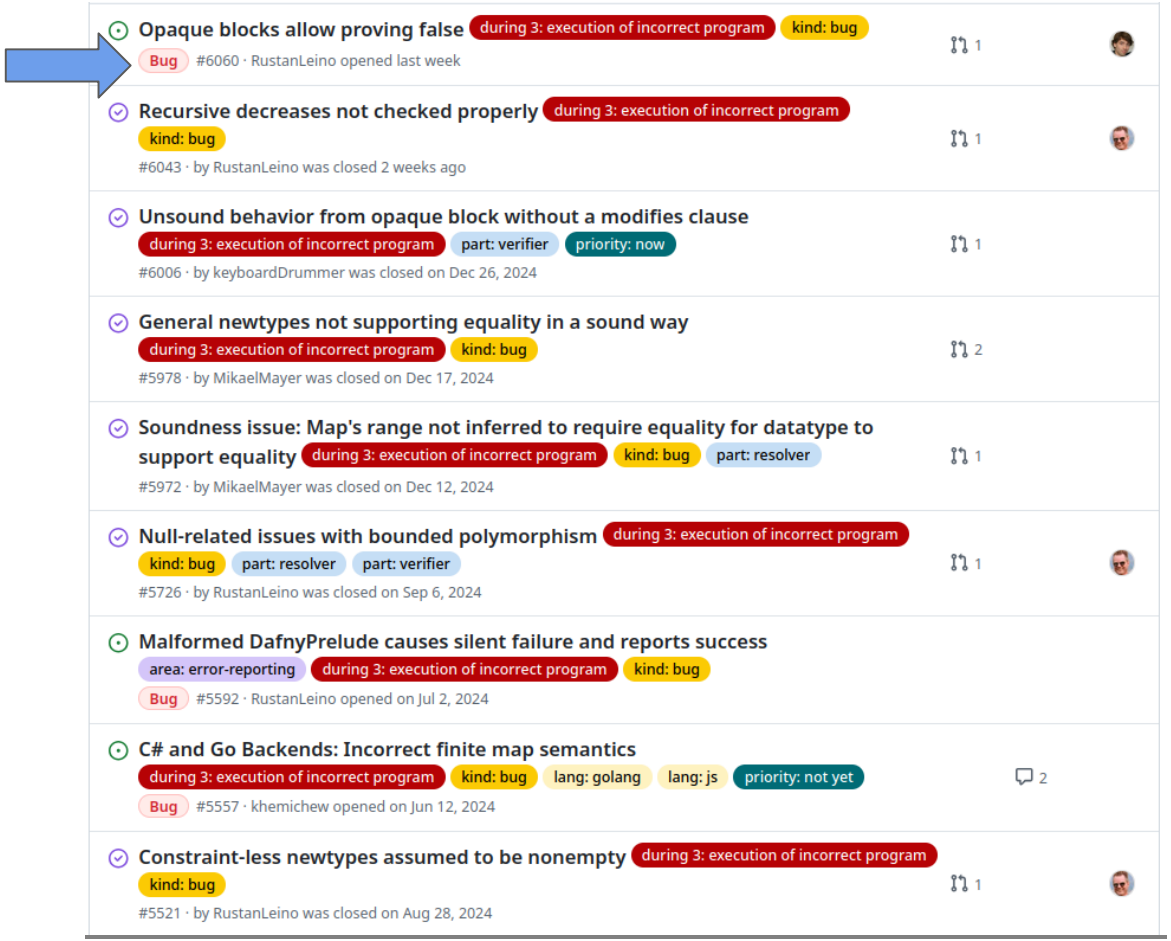
```
method Main() {
  var c: object;
  opaque
    ensures fresh(c)
  {
    c := new object;
  }
  assert false;
}
```

Command to run and resulting output

```
% dafny verify test.dfy

Dafny program verifier finished with 1 verified, 0 errors
```

Problem: Unsoundnesses Occur Regularly



Opaque blocks allow proving false — during 3: execution of incorrect program, kind: bug
Bug #6060 · RustanLeino opened last week — 1

Recursive decreases not checked properly — during 3: execution of incorrect program, kind: bug
#6043 · by RustanLeino was closed 2 weeks ago — 1

Unsound behavior from opaque block without a modifies clause — during 3: execution of incorrect program, part: verifier, priority: now
#6006 · by keyboardDrummer was closed on Dec 26, 2024 — 1

General newtypes not supporting equality in a sound way — during 3: execution of incorrect program, kind: bug
#5978 · by MikaelMayer was closed on Dec 17, 2024 — 2

Soundness issue: Map's range not inferred to require equality for datatype to support equality — during 3: execution of incorrect program, kind: bug, part: resolver
#5972 · by MikaelMayer was closed on Dec 12, 2024 — 1

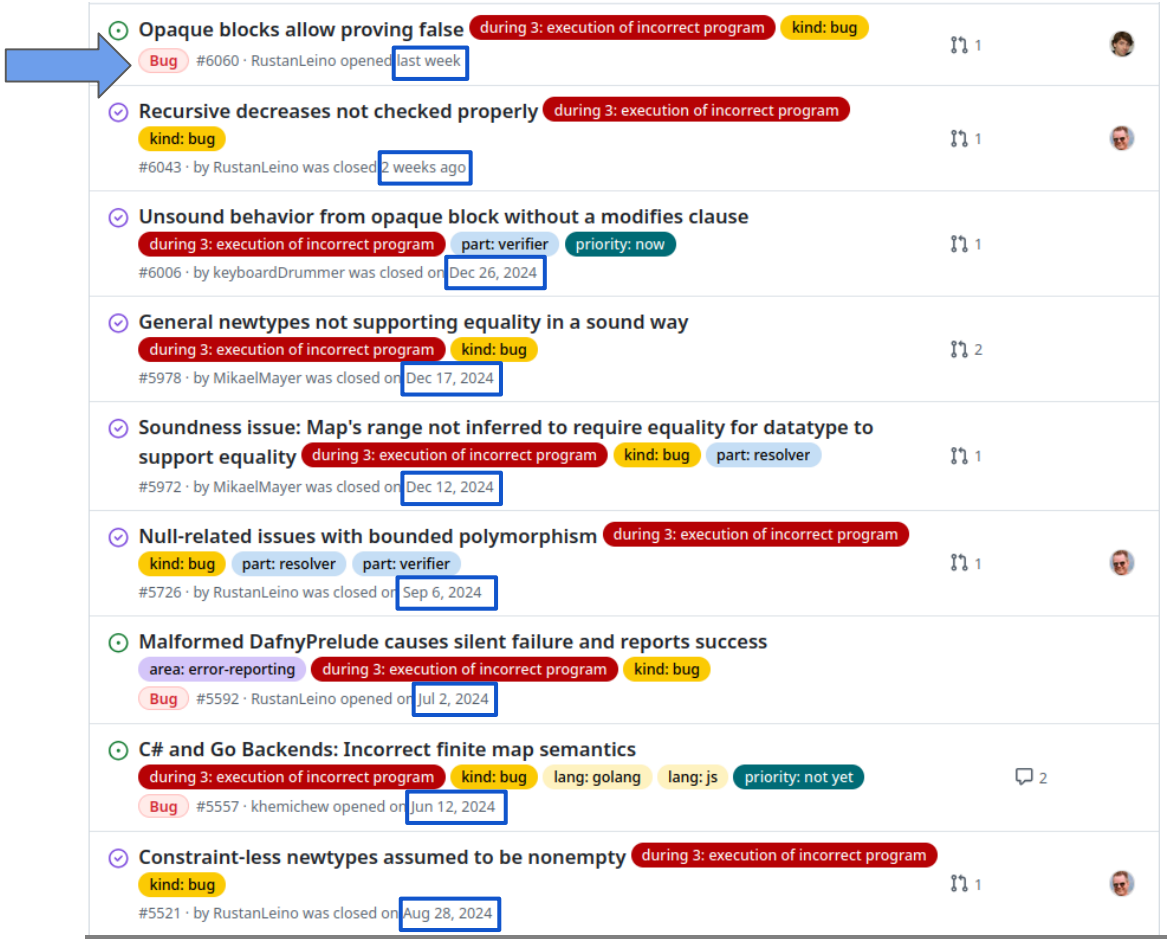
Null-related issues with bounded polymorphism — during 3: execution of incorrect program, kind: bug, part: resolver, part: verifier
#5726 · by RustanLeino was closed on Sep 6, 2024 — 1

Malformed DafnyPrelude causes silent failure and reports success — area: error-reporting, during 3: execution of incorrect program, kind: bug
Bug #5592 · RustanLeino opened on Jul 2, 2024

C# and Go Backends: Incorrect finite map semantics — during 3: execution of incorrect program, kind: bug, lang: golang, lang: js, priority: not yet
Bug #5557 · khemichew opened on Jun 12, 2024 — 2

Constraint-less newtypes assumed to be nonempty — during 3: execution of incorrect program, kind: bug
#5521 · by RustanLeino was closed on Aug 28, 2024 — 1

Problem: Unsoundnesses Occur Regularly



Opaque blocks allow proving false — during 3: execution of incorrect program — kind: bug
Bug #6060 · RustanLeino opened last week — 1

Recursive decreases not checked properly — during 3: execution of incorrect program — kind: bug
#6043 · by RustanLeino was closed 2 weeks ago — 1

Unsound behavior from opaque block without a modifies clause — during 3: execution of incorrect program — part: verifier — priority: now
#6006 · by keyboardDrummer was closed on Dec 26, 2024 — 1

General newtypes not supporting equality in a sound way — during 3: execution of incorrect program — kind: bug
#5978 · by MikaelMayer was closed on Dec 17, 2024 — 2

Soundness issue: Map's range not inferred to require equality for datatype to support equality — during 3: execution of incorrect program — kind: bug — part: resolver
#5972 · by MikaelMayer was closed on Dec 12, 2024 — 1

Null-related issues with bounded polymorphism — during 3: execution of incorrect program — kind: bug — part: resolver — part: verifier
#5726 · by RustanLeino was closed on Sep 6, 2024 — 1

Malformed DafnyPrelude causes silent failure and reports success — area: error-reporting — during 3: execution of incorrect program — kind: bug
Bug #5592 · RustanLeino opened on Jul 2, 2024

C# and Go Backends: Incorrect finite map semantics — during 3: execution of incorrect program — kind: bug — lang: golang — lang: js — priority: not yet
Bug #5557 · khemichew opened on Jun 12, 2024 — 2

Constraint-less newtypes assumed to be nonempty — during 3: execution of incorrect program — kind: bug
#5521 · by RustanLeino was closed on Aug 28, 2024 — 1

Problem: Unsoundnesses Occur Regularly

- **Unsoundness due to bad simplification** `kind/unsoundness`
  #3213 · by mtzguido was closed on Feb 9, 2024
- **Some problems with reflection and simplification of equalities** `component/tactics` `kind/unsoundness`
  #2806 · by mtzguido was closed on Jan 31, 2023
- **F* should disallow degenerate effects** `kind/unsoundness`
  #2659 · 857b opened on Jul 25, 2022
- **A soundness issue?** `kind/unsoundness` `priority/high` `status/has-pr`
  #2072 · by wintersteiger was closed on Dec 15, 2020
- **`WellFounded.axiom1` is incompatible with universe-free SMT encoding** `component/smtencoding` `component/universes` `kind/bug` `kind/unsoundness` `priority/high`
  #2069 · by mtzguido was closed on Jan 14, 2021
- **Non-total layered effects reifying as total?** `component/effect-system` `kind/bug` `kind/unsoundness`
  #2055 · by mtzguido was closed on Jun 1, 2020
- **Incorrect handling of machine signed integers** `component/extraction` `component/libraries` `kind/unsoundness` `milestone/everest-v1` `priority/high` `status/has-pr`
  #1803 · by msprotz was closed on Mar 16, 2020

Problem: Unsoundnesses Occur Regularly

- **Unsoundness due to bad simplification** kind/unsoundness
  #3213 · by mtzguido was closed on Feb 9, 2024
- **Some problems with reflection and simplification of equalities** component/tactics kind/unsoundness
  #2806 · by mtzguido was closed on Jan 31, 2023
- **F* should disallow degenerate effects** kind/unsoundness
  #2659 · 857b opened on Jul 25, 2022
- **A soundness issue?** kind/unsoundness priority/high status/has-pr
  #2072 · by wintersteiger was closed on Dec 15, 2020
- **`WellFounded.axiom1` is incompatible with universe-free SMT encoding** component/smtencoding component/universes kind/bug kind/unsoundness priority/high
  #2069 · by mtzguido was closed on Jan 14, 2021
- **Non-total layered effects reifying as total?** component/effect-system kind/bug kind/unsoundness
  #2055 · by mtzguido was closed on Jun 1, 2020
- **Incorrect handling of machine signed integers** component/extraction component/libraries kind/unsoundness milestone/everest-v1 priority/high status/has-pr
  #1803 · by msprotz was closed on Mar 16, 2020

- **Unsoundness related to QPs with unsatisfiable conditions** quantified-permissions unsoundness
  #842 · by marcoeilers was closed on May 6, 2024
- **Unsound behavior with QPs without condition** quantified-permissions unsoundness
  #833 · by marcoeilers was closed on May 5, 2024
- **Errors caused by refute statement inside of loop with goto** bug unsoundness
  #773 · by bruggerl was closed on Nov 10, 2023
- **Unsoundness due to loop invariant** bug unsoundness
  #740 · by bruggerl was closed on Aug 14, 2023
- **Exhaling an empty QP allows asserting false** bug critical quantified-permissions unsoundness
  #688 · by vakaras was closed on Mar 13, 2023
- **Unsoundness with quantified permissions** bug duplicate quantified-permissions unsoundness
  #636 · by zgrannan was closed on Sep 2, 2022
- **Silicon verifies example that should fail.** unsoundness
  #630 · by zgrannan was closed on Jul 20, 2022
- **Unsoundness: a heap dependent function is instantiated even when its precondition does not hold** bug functions major unsoundness
  #376 · by viper-admin was closed on Nov 25, 2022
- **Strange behaviour of unconstrained quantified permissions** bug critical quantified-permissions unsoundness
  #342 · by viper-admin was closed on Nov 25, 2022

Problem 1
Trustworthiness

Problem 2
Expressiveness

Problem 1
Trustworthiness

How can we ensure that automated verifiers are **sound**?

Problem 2
Expressiveness

Problem 1
Trustworthiness

How can we ensure that automated verifiers are **sound**?

Problem 2
Expressiveness

What about **security** properties?

My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Problem 2
Expressiveness

My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Problem 2
Expressiveness

Provably sound and automated verifier based on *separation logic*

My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Problem 2
Expressiveness

for realistic programs

Provably sound and automated verifier based on *separation logic*

My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Problem 2
Expressiveness

for realistic programs

Provably sound and automated verifier based on *separation logic*

Automated verifier for arbitrary *hyperproperties*

My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Problem 2
Expressiveness

for realistic programs

for security properties

Provably sound and automated verifier based on *separation logic*

Automated verifier for arbitrary *hyperproperties*

My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

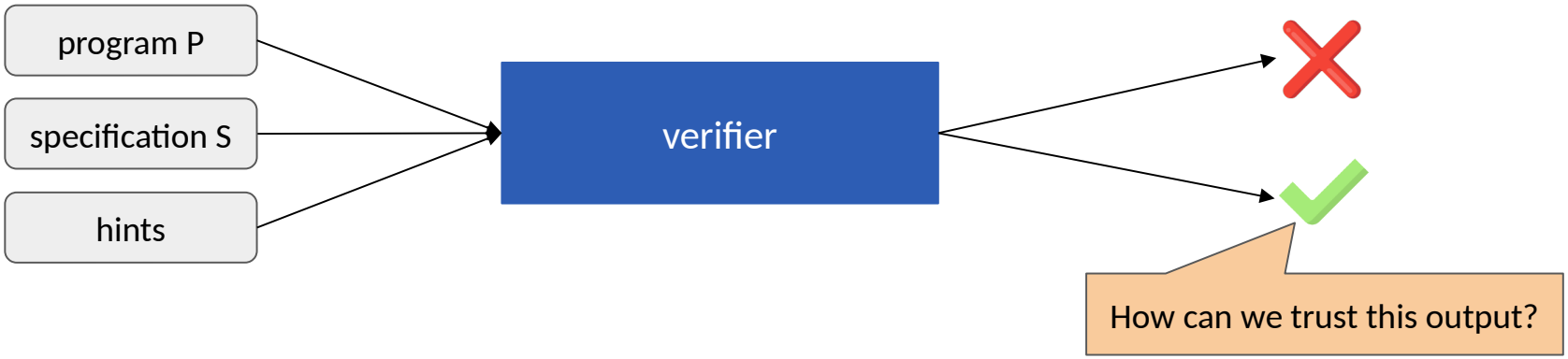
Provably sound and automated verifier based on *separation logic*

Problem 2
Expressiveness

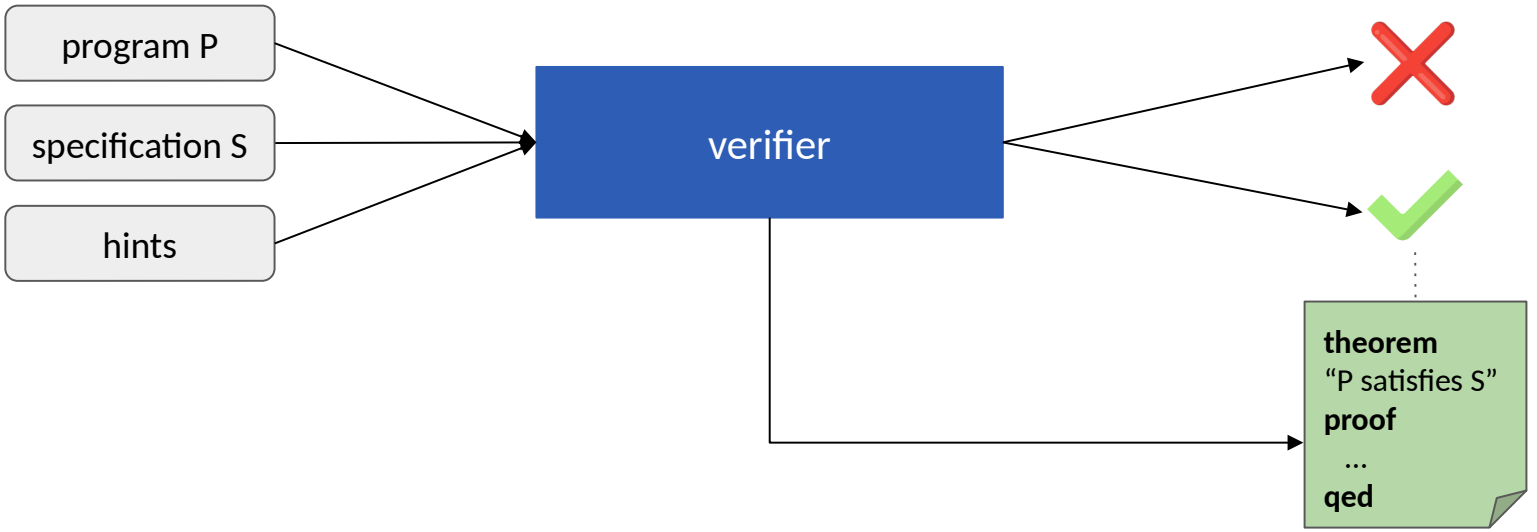
Automated verifier for arbitrary *hyperproperties*

A Trustworthy Automated Verifier

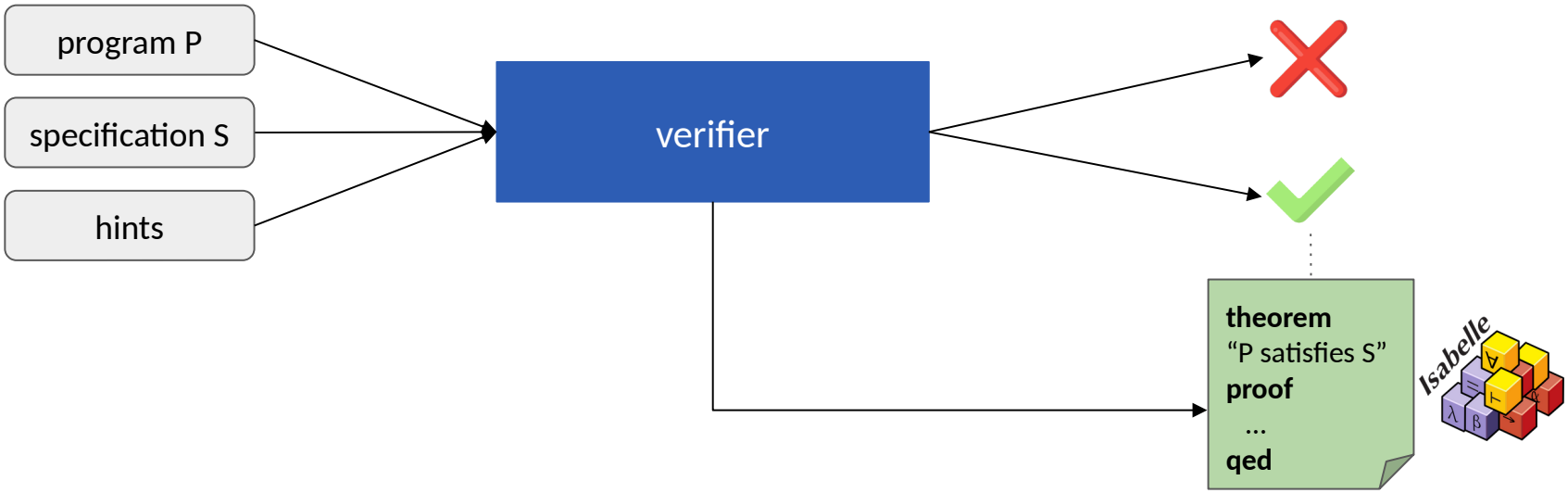
A Trustworthy Automated Verifier



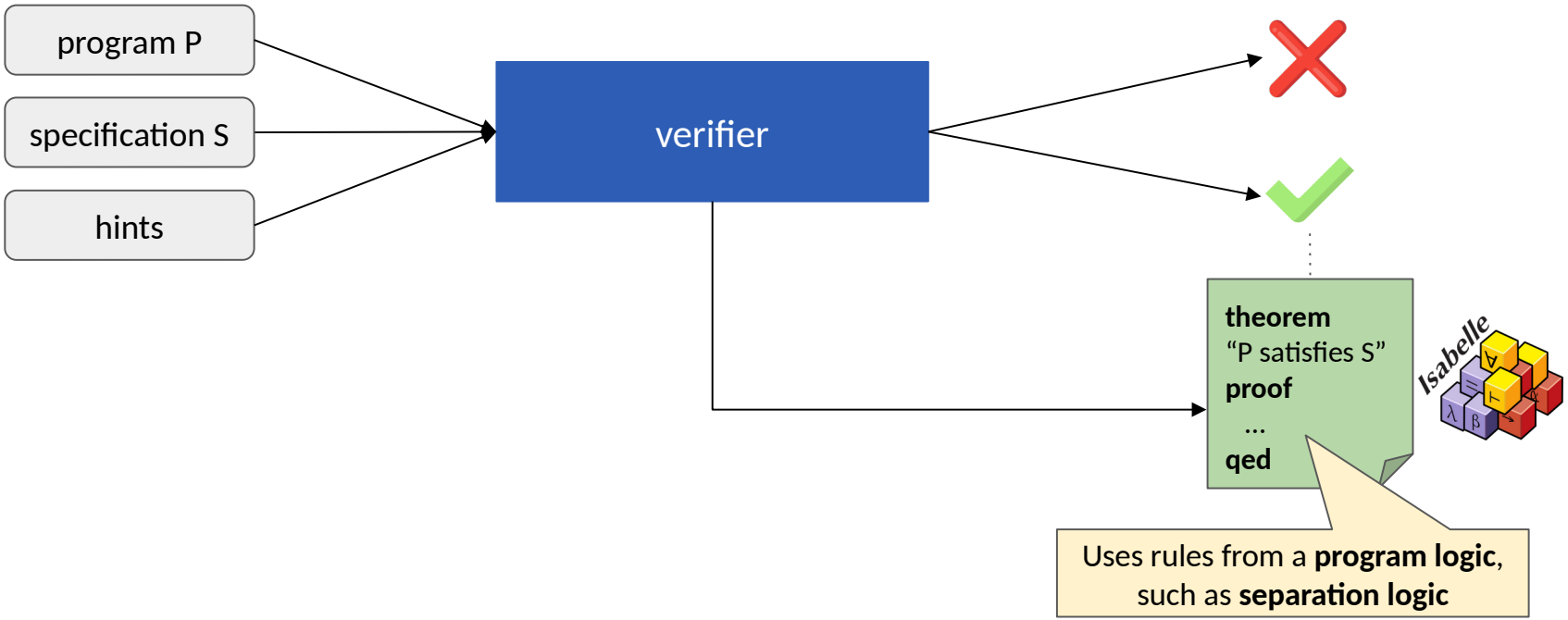
A Trustworthy Automated Verifier



A Trustworthy Automated Verifier



A Trustworthy Automated Verifier

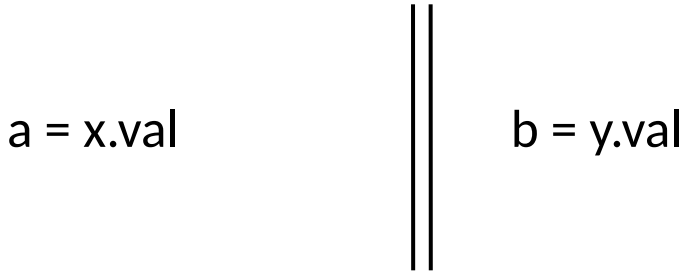


Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.

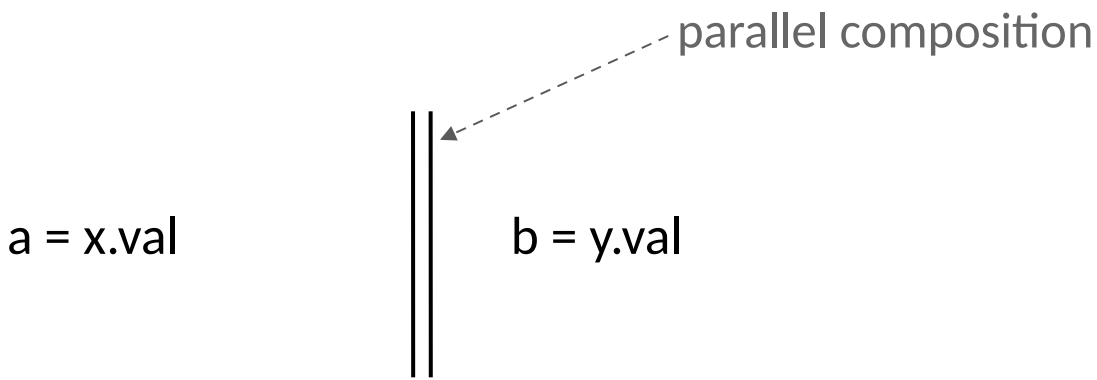
Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.

$\{x.val \mapsto 5 * y.val \mapsto 7\}$

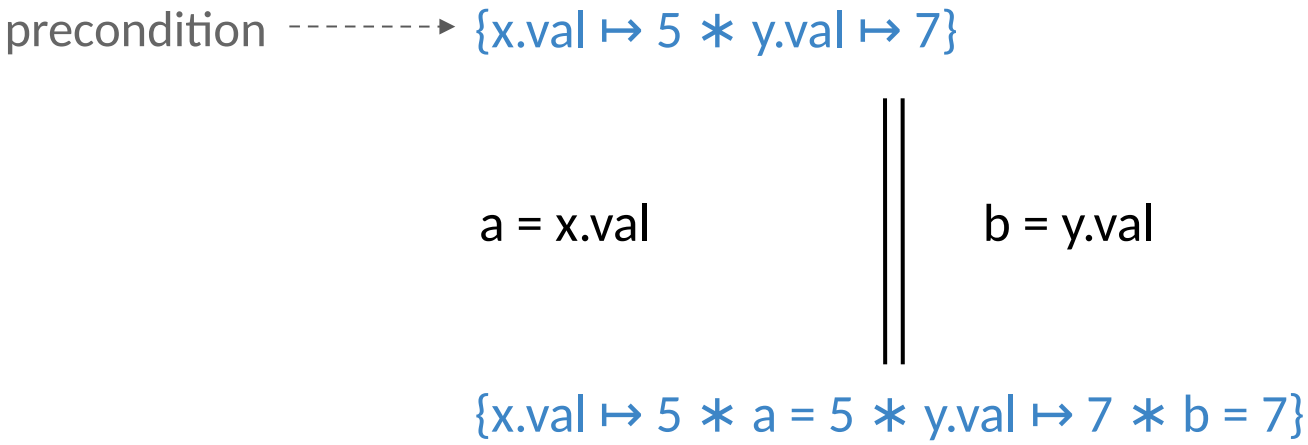
a = x.val

b = y.val

$\{x.val \mapsto 5 * a = 5 * y.val \mapsto 7 * b = 7\}$

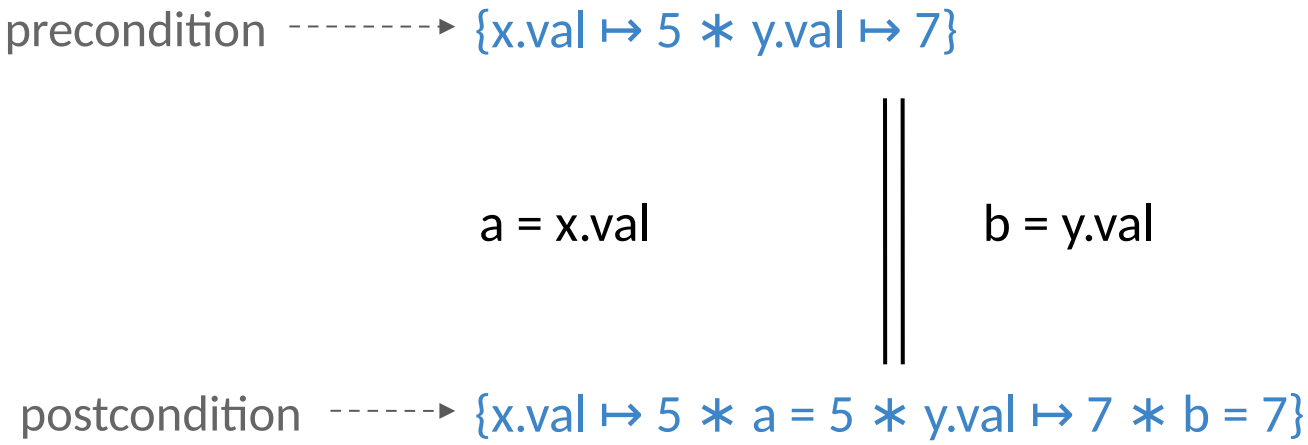
Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.

separating conjunction
conjuncts hold on *disjoint* parts of the heap

$\{x.val \mapsto 5 * y.val \mapsto 7\}$

a = x.val

b = y.val

$\{x.val \mapsto 5 * a = 5 * y.val \mapsto 7 * b = 7\}$

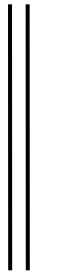
Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.

represents **ownership**
required to read and write

separating conjunction
conjuncts hold on *disjoint* parts of the heap

$\{x.val \mapsto 5 * y.val \mapsto 7\}$



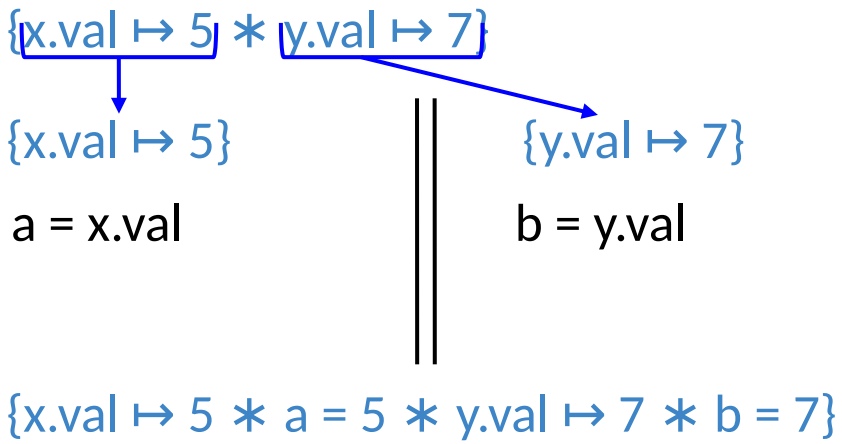
a = x.val

b = y.val

$\{x.val \mapsto 5 * a = 5 * y.val \mapsto 7 * b = 7\}$

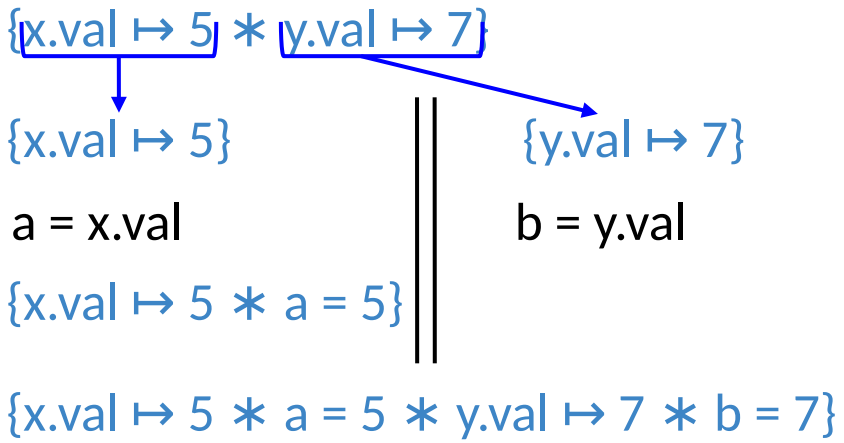
Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



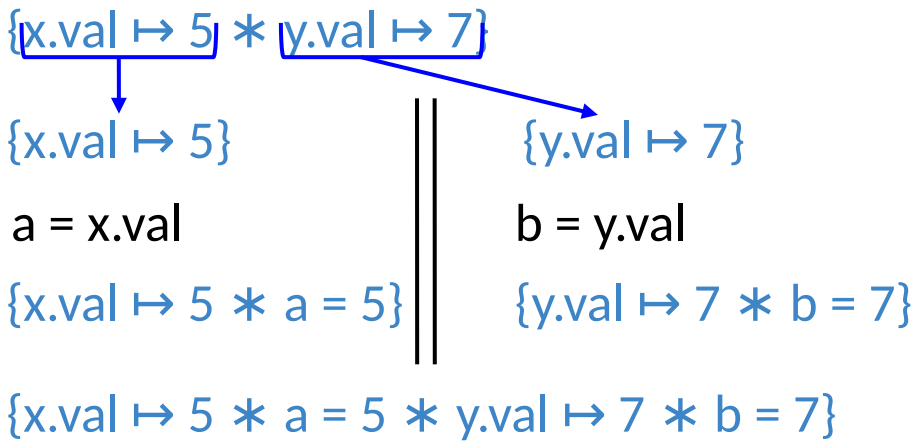
Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



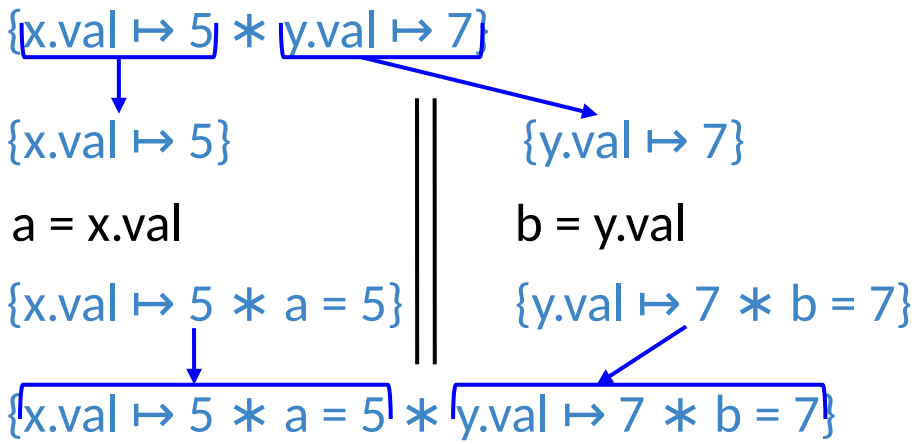
Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



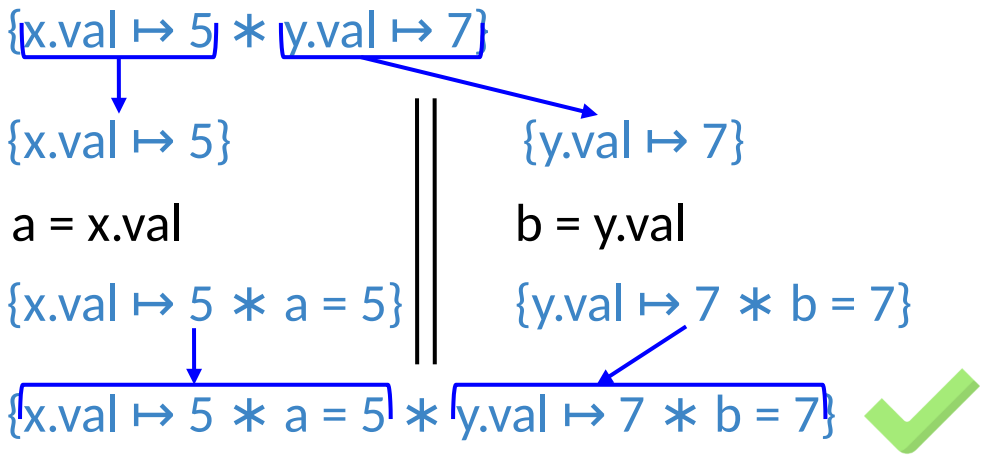
Separation Logic 101

Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.

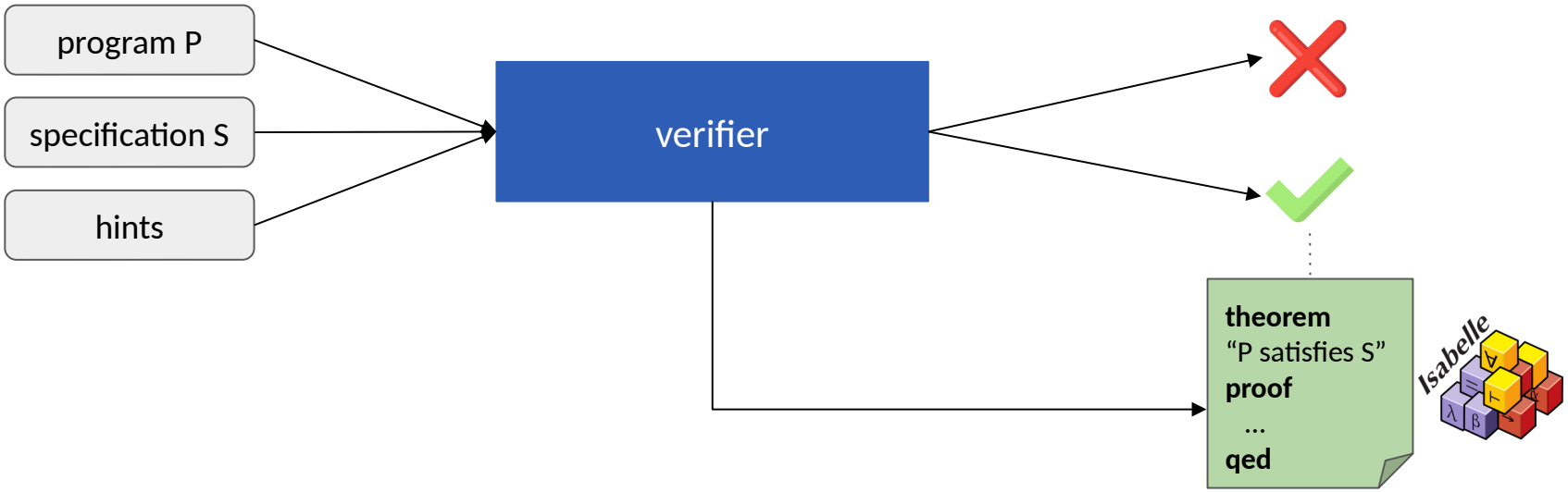


Separation Logic 101

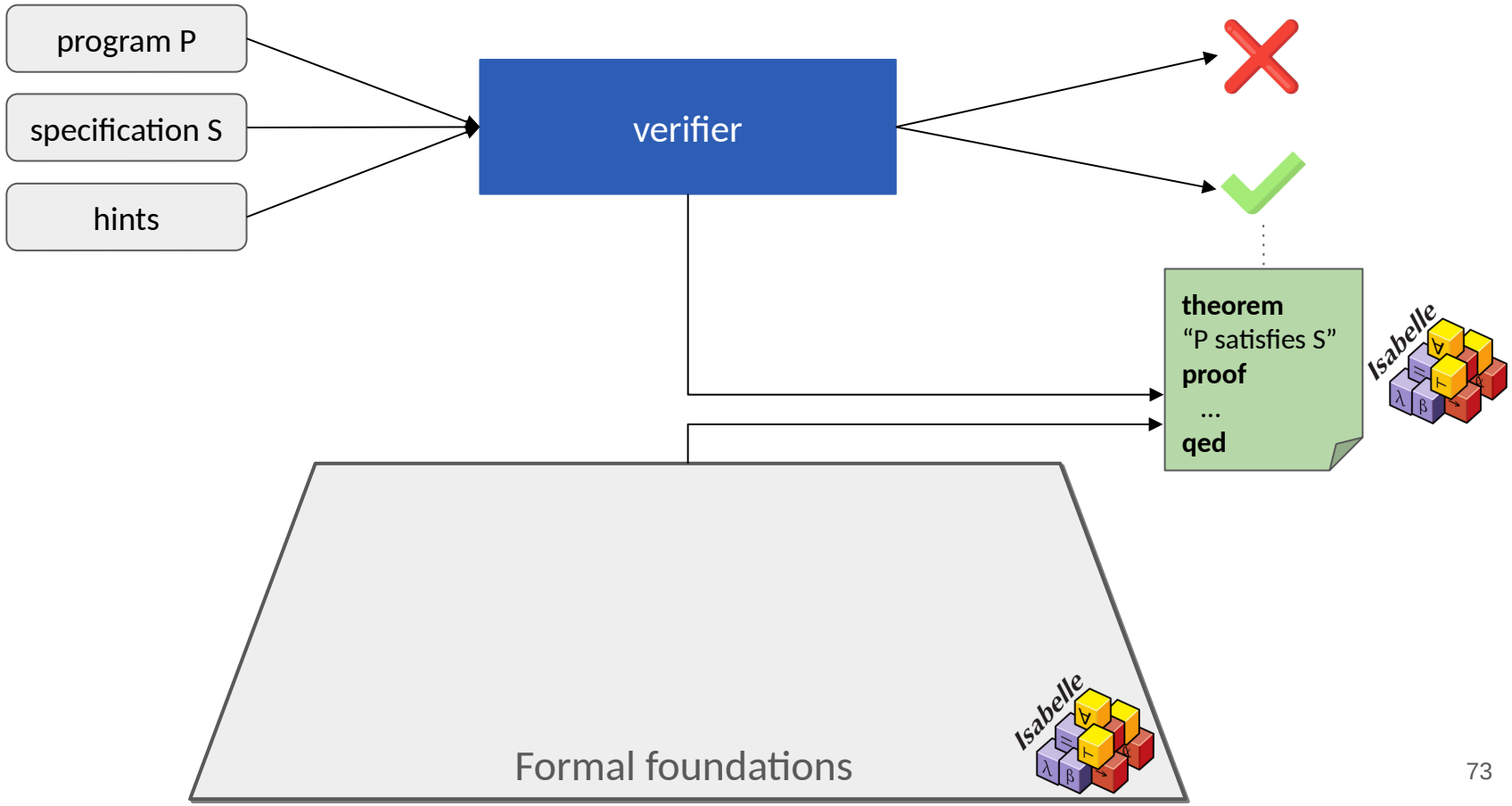
Separation logic: state-of-the-art program logic that enables **modular** reasoning about sequential and concurrent programs that manipulate the heap.



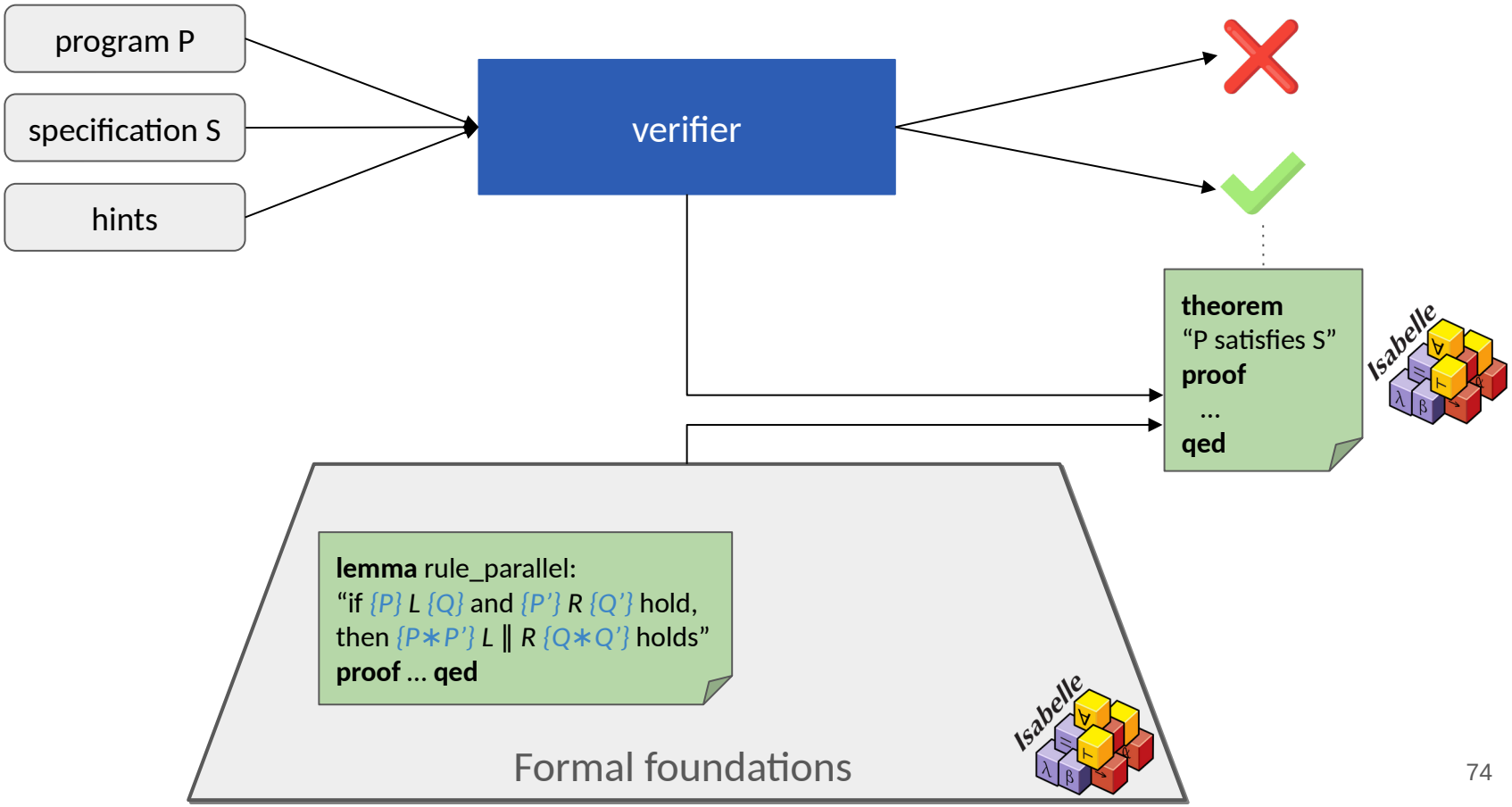
A Trustworthy Automated Verifier



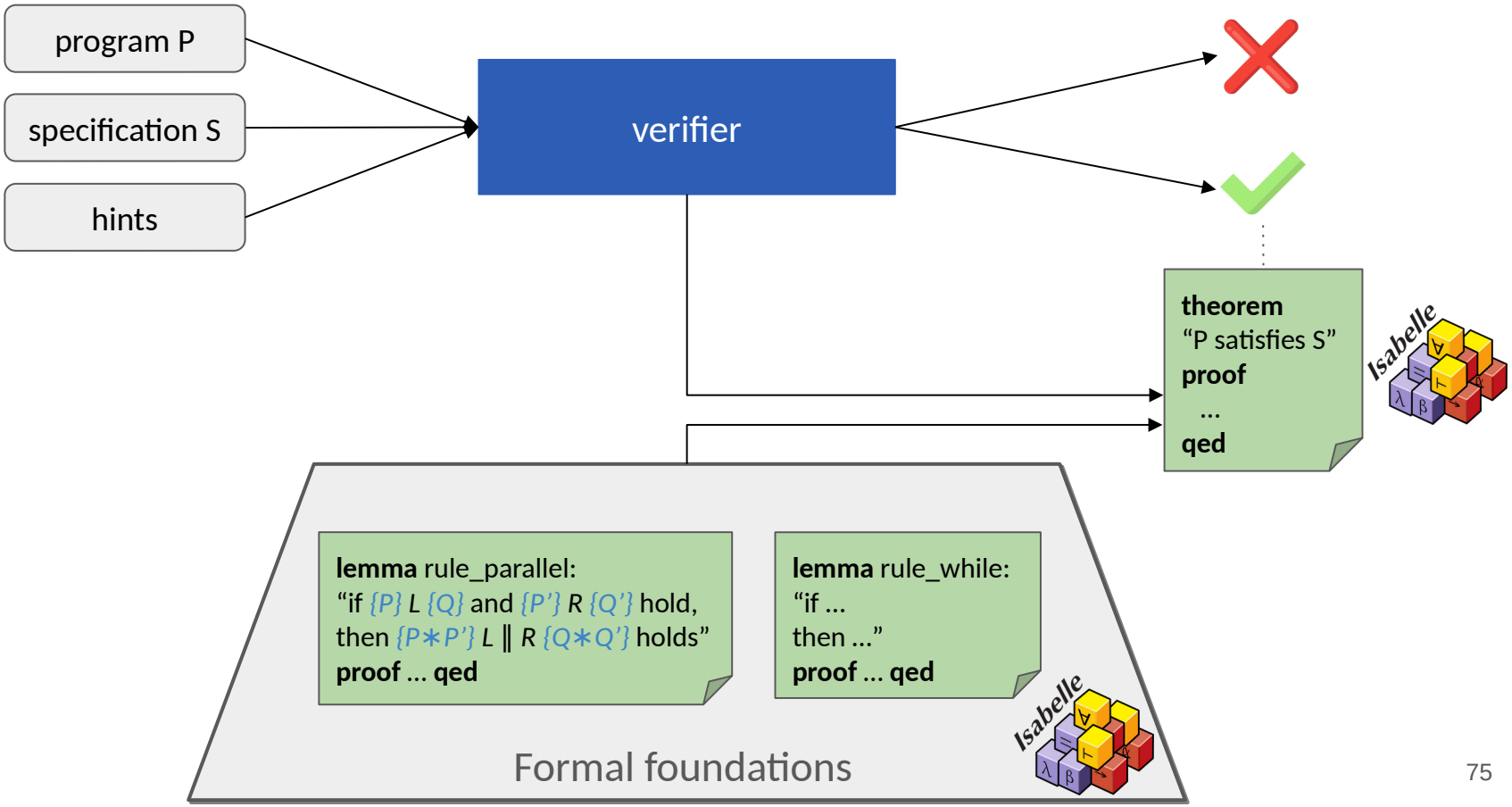
A Trustworthy Automated Verifier



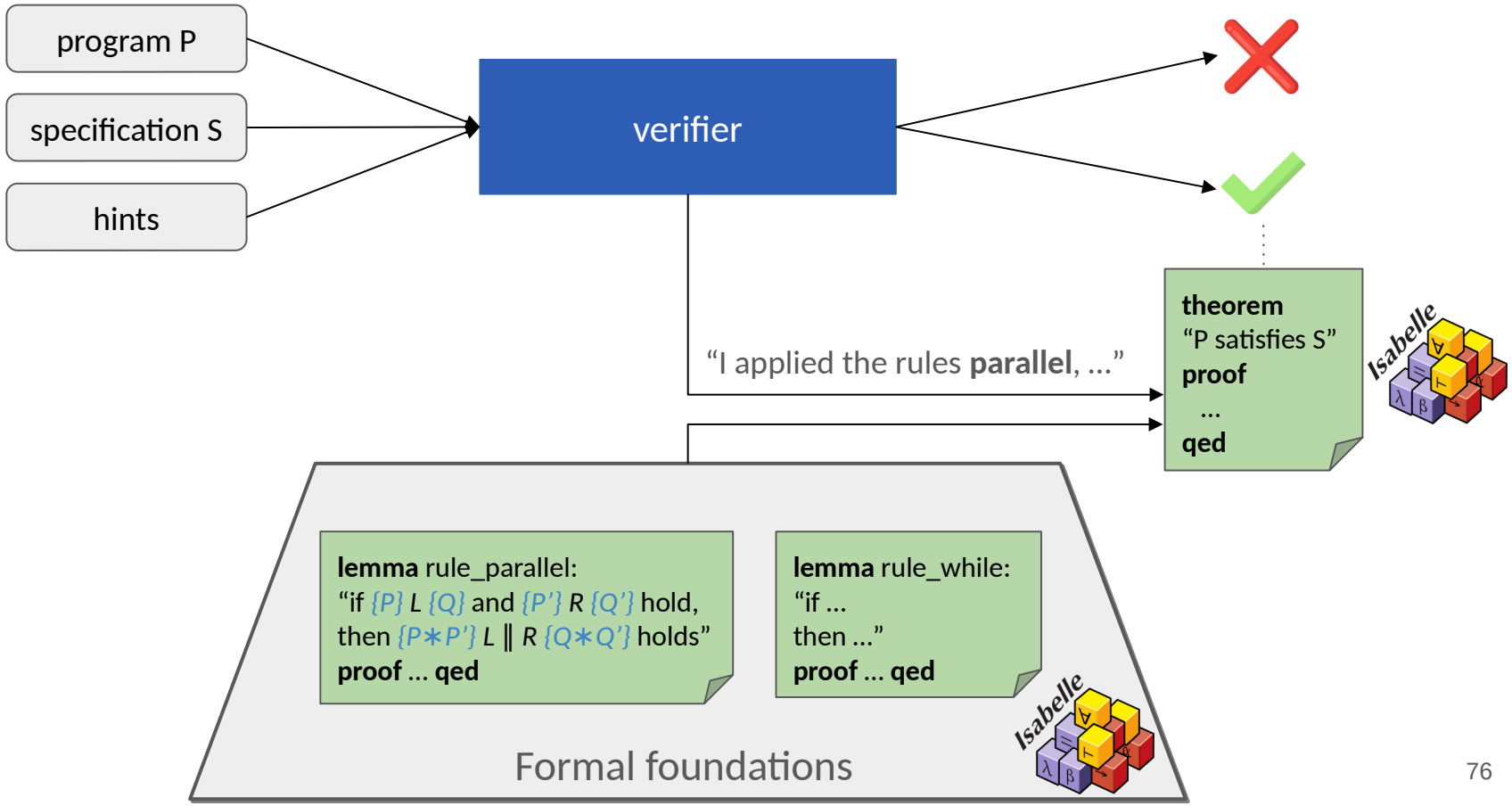
A Trustworthy Automated Verifier



A Trustworthy Automated Verifier

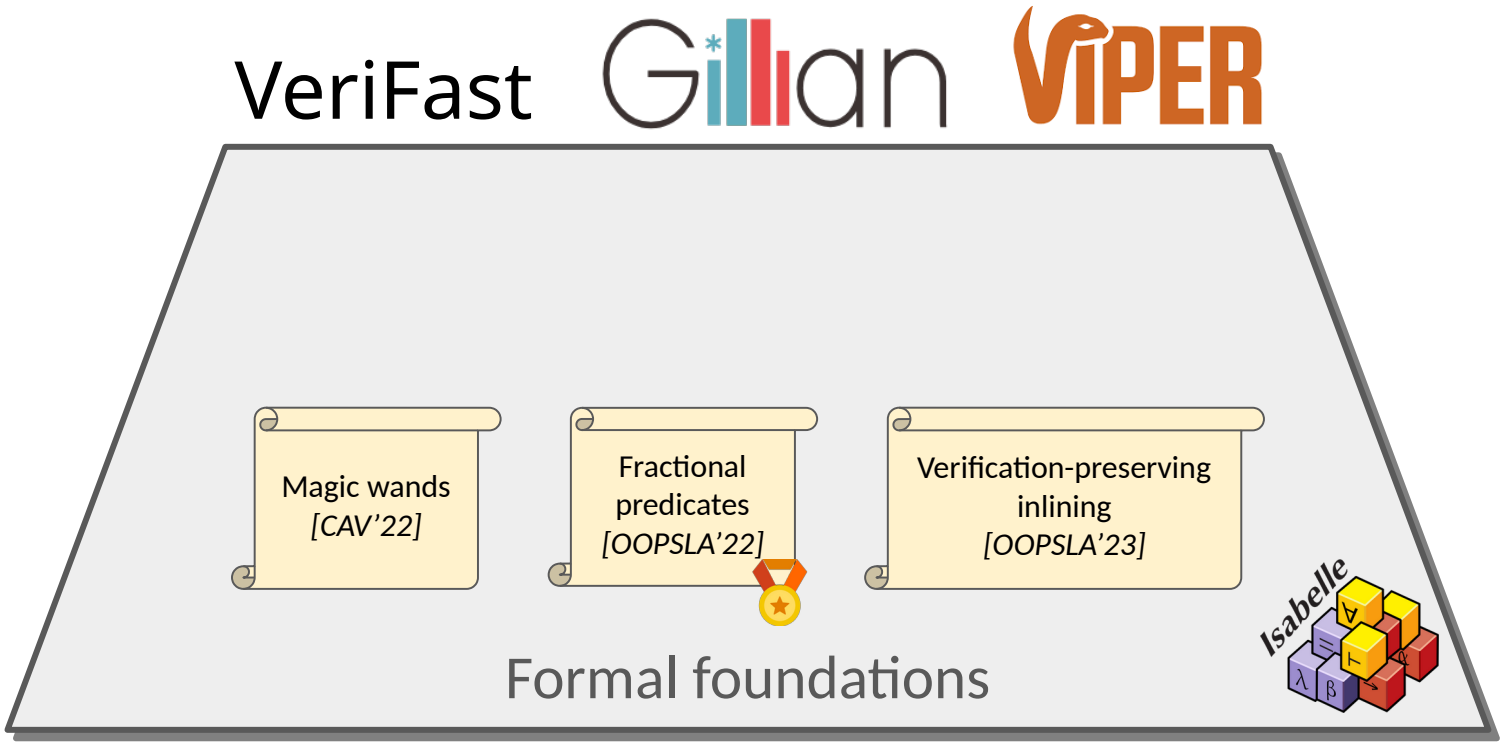


A Trustworthy Automated Verifier

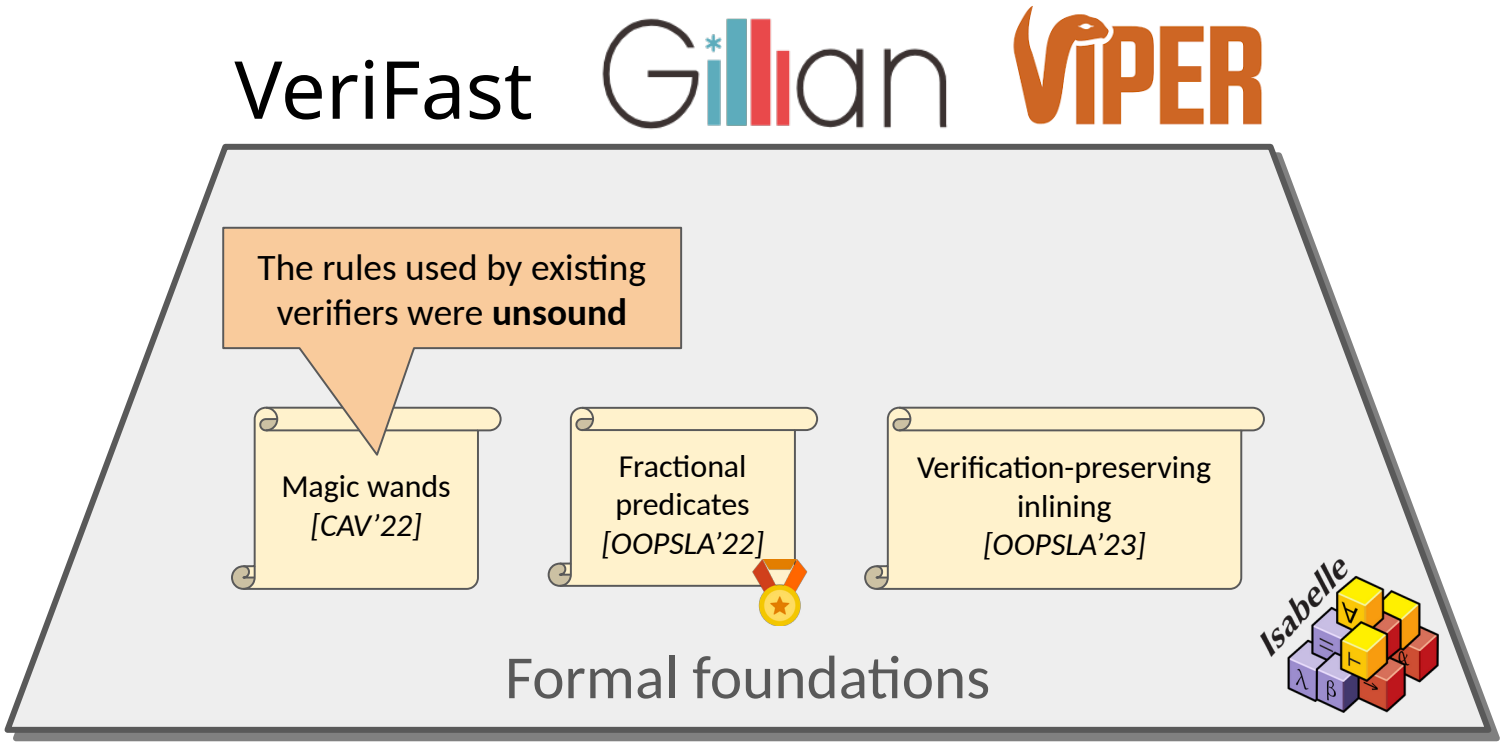


Formal Foundations for Automated Separation Logic Verifiers

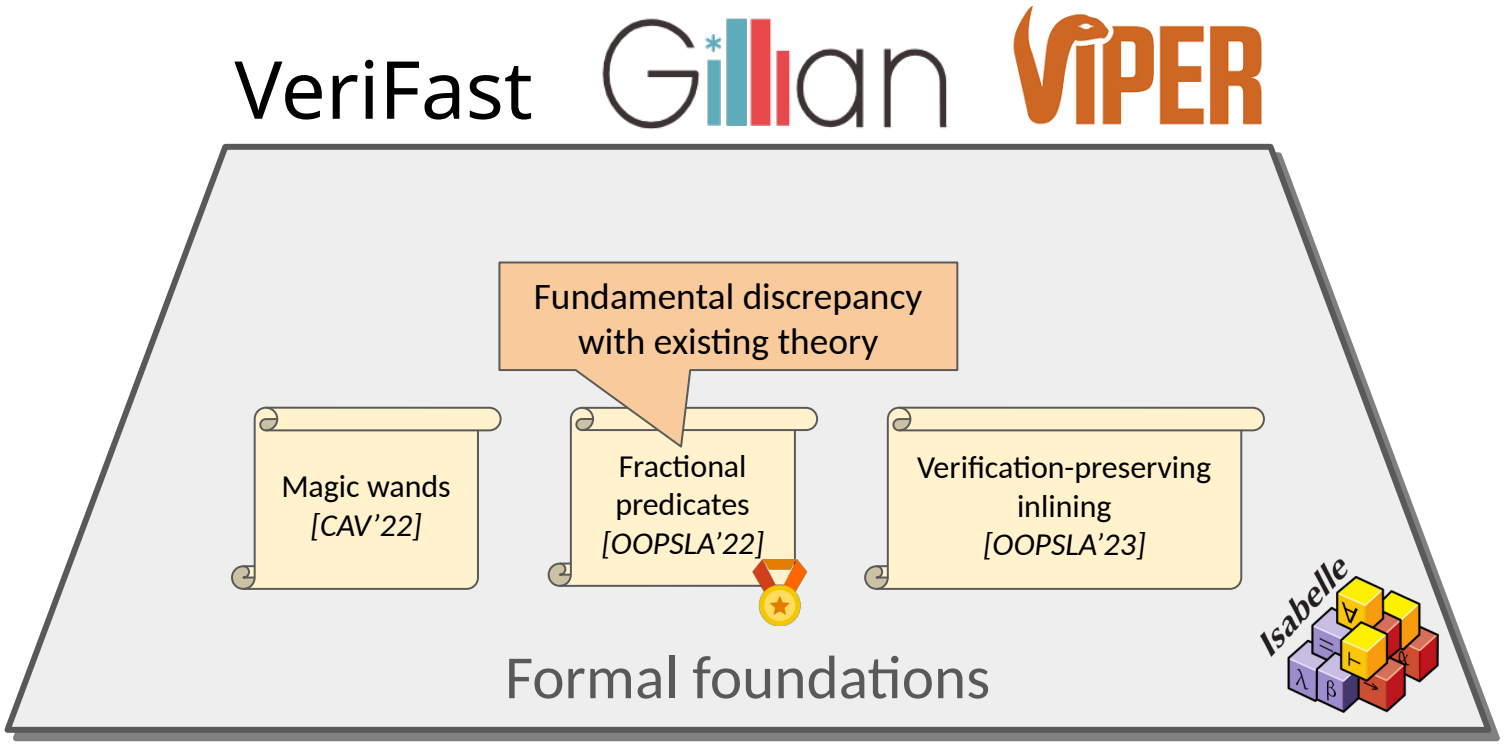
Formal Foundations for Automated Separation Logic Verifiers



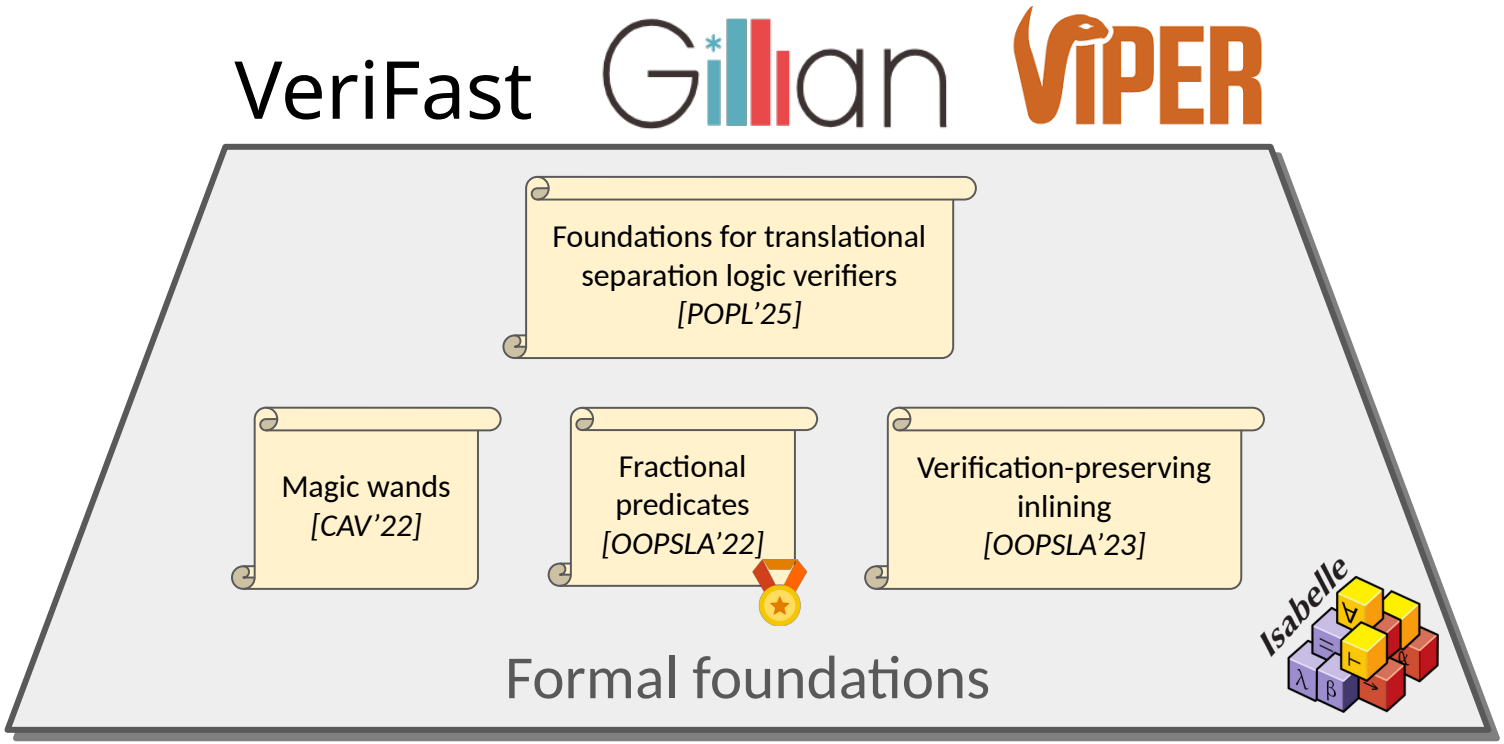
Formal Foundations for Automated Separation Logic Verifiers



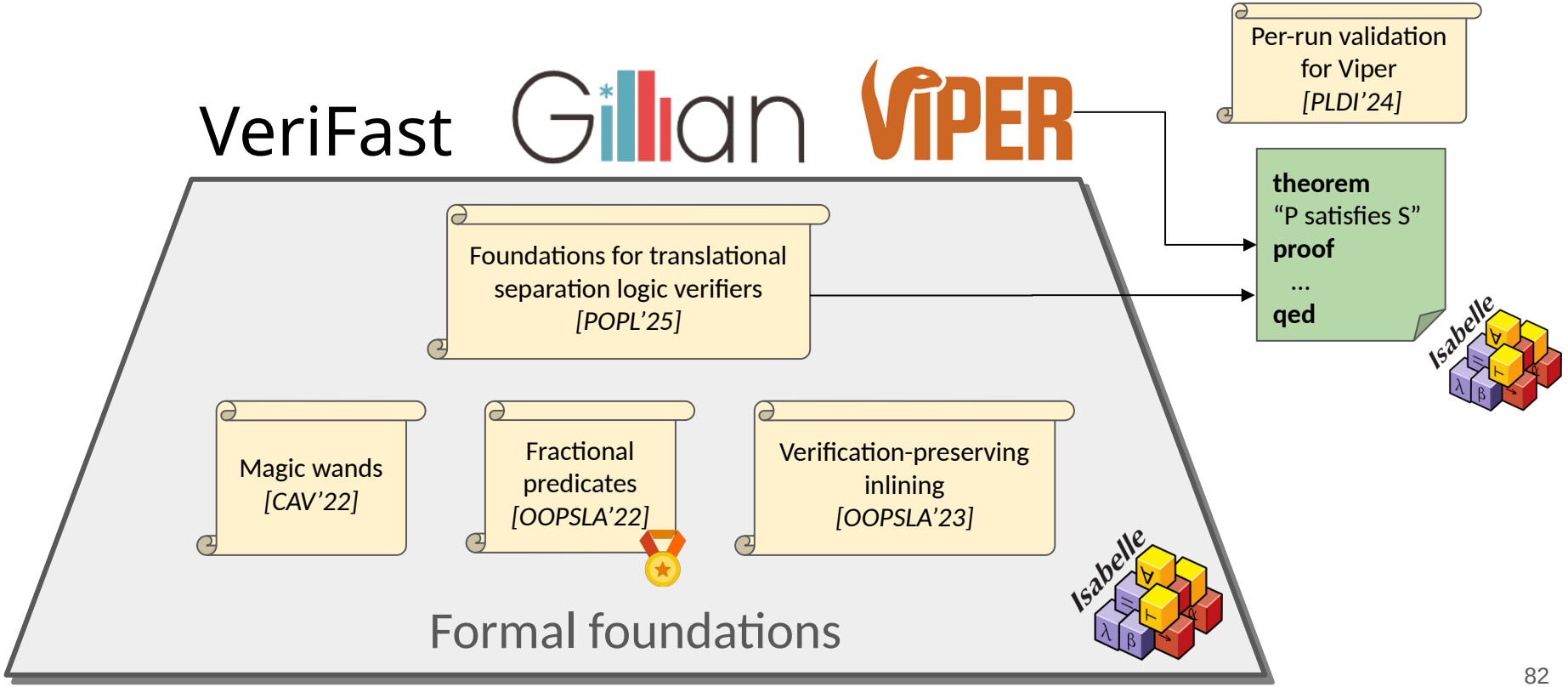
Formal Foundations for Automated Separation Logic Verifiers



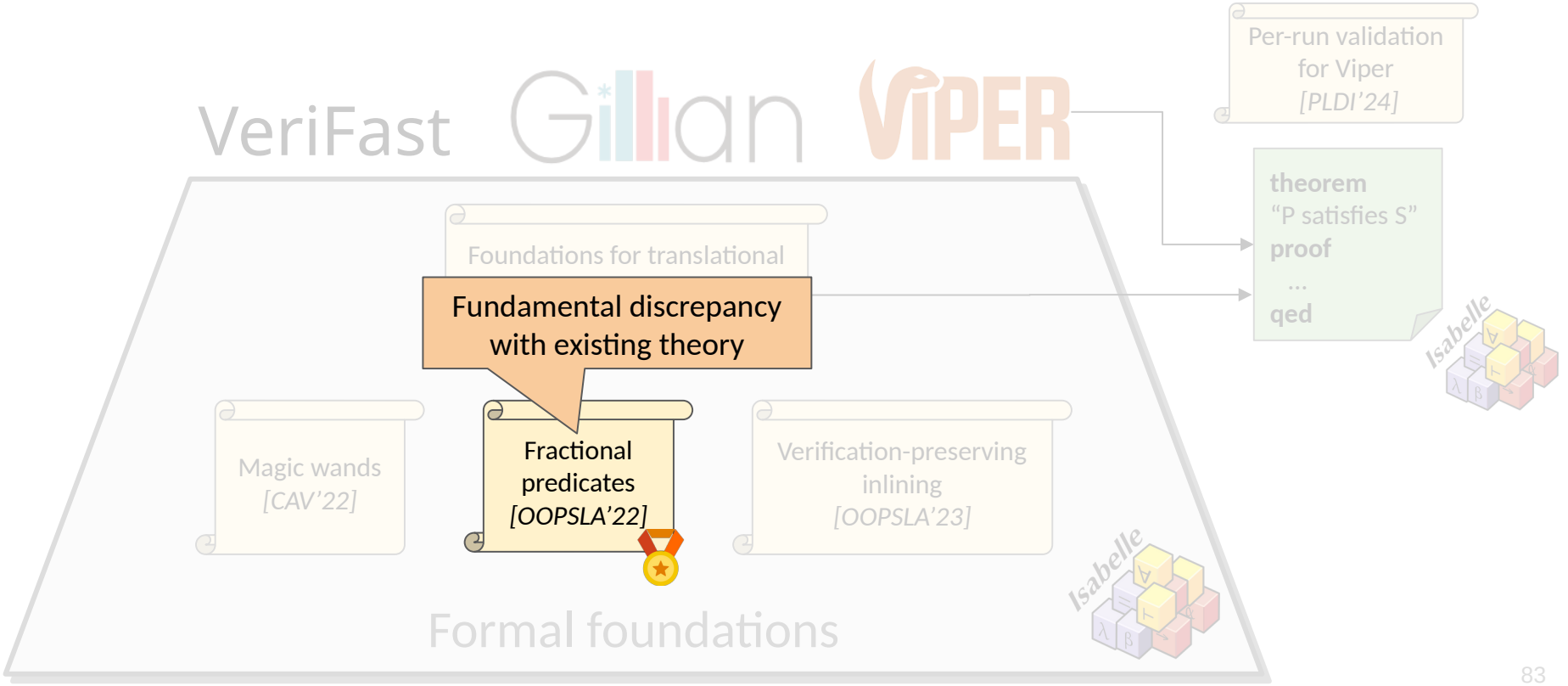
Formal Foundations for Automated Separation Logic Verifiers



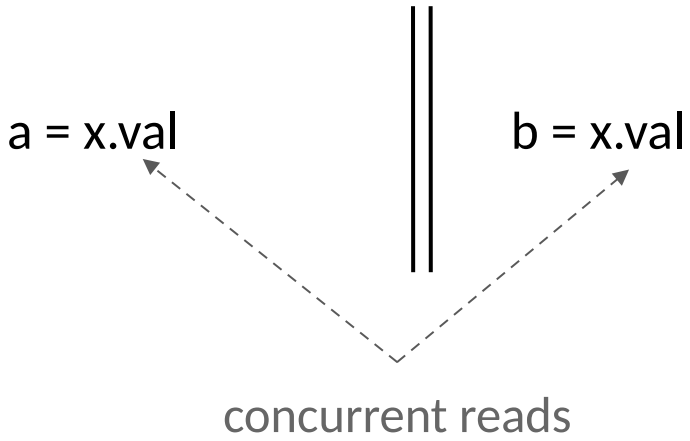
Formal Foundations for Automated Separation Logic Verifiers



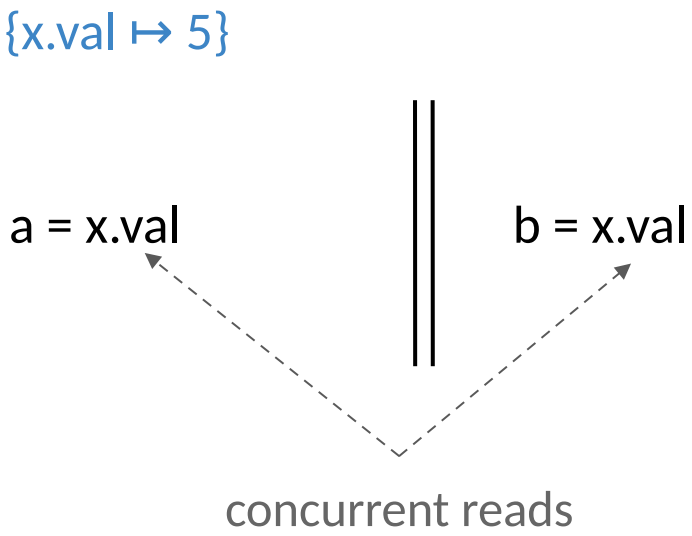
Formal Foundations for Automated Separation Logic Verifiers



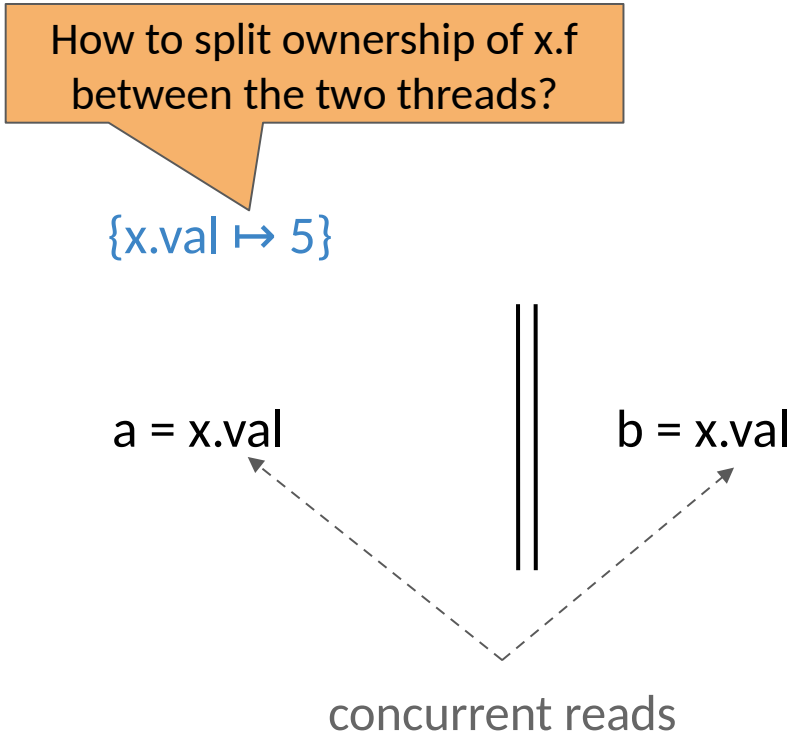
Fractional Permissions



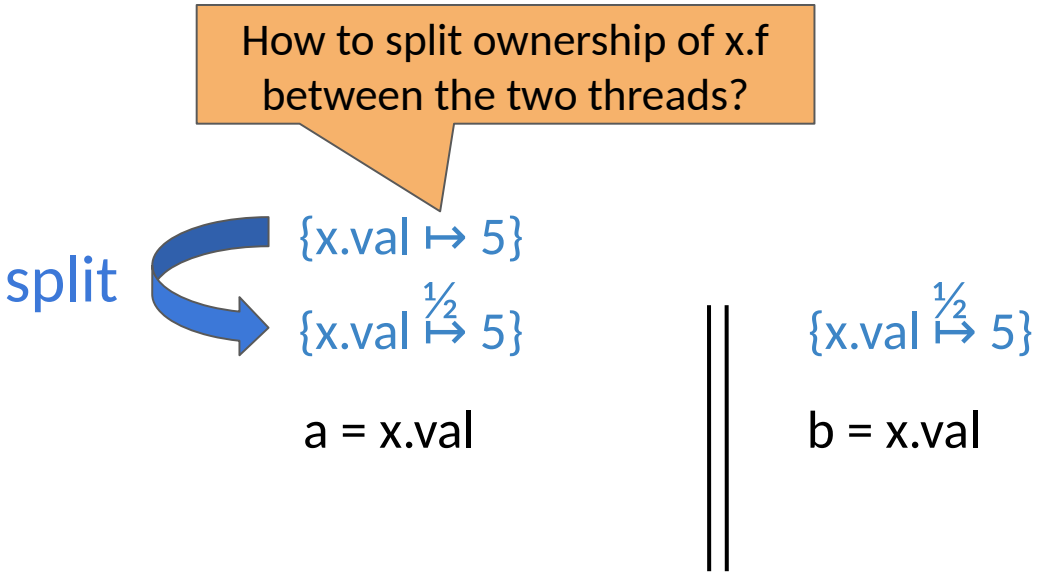
Fractional Permissions



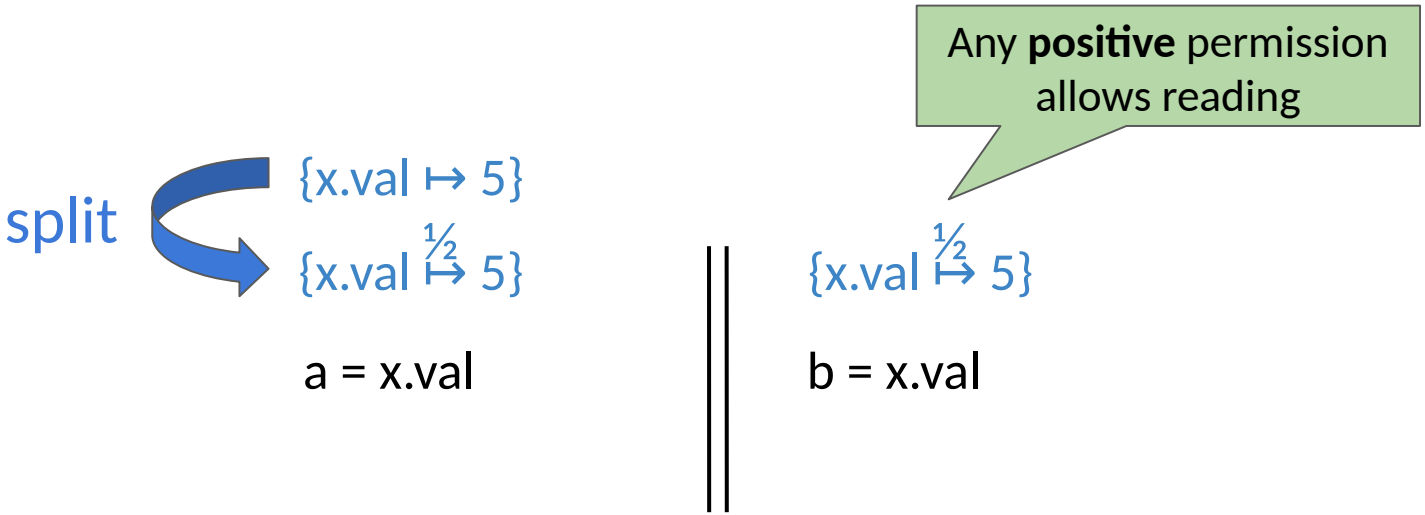
Fractional Permissions



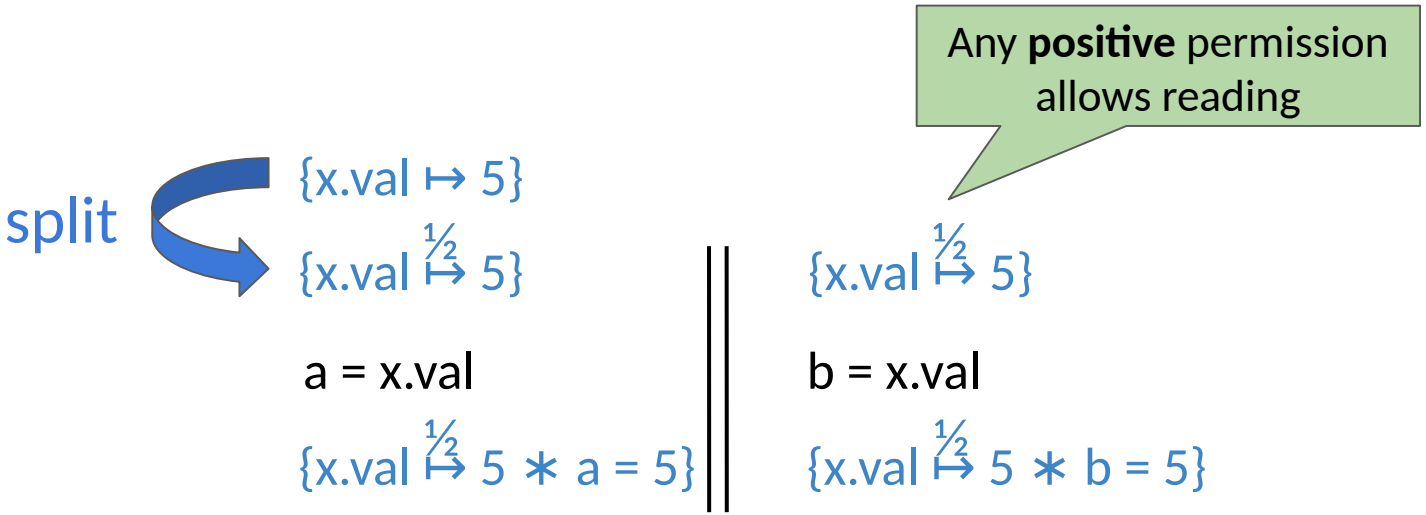
Fractional Permissions



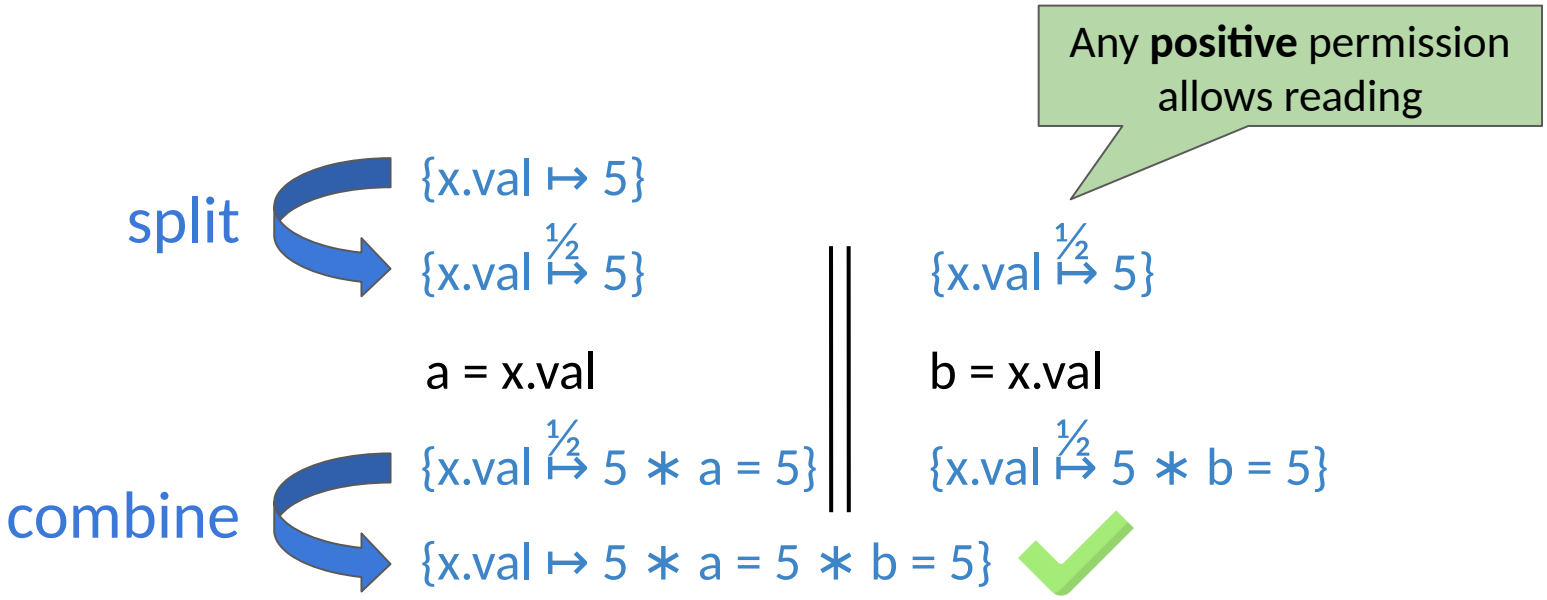
Fractional Permissions



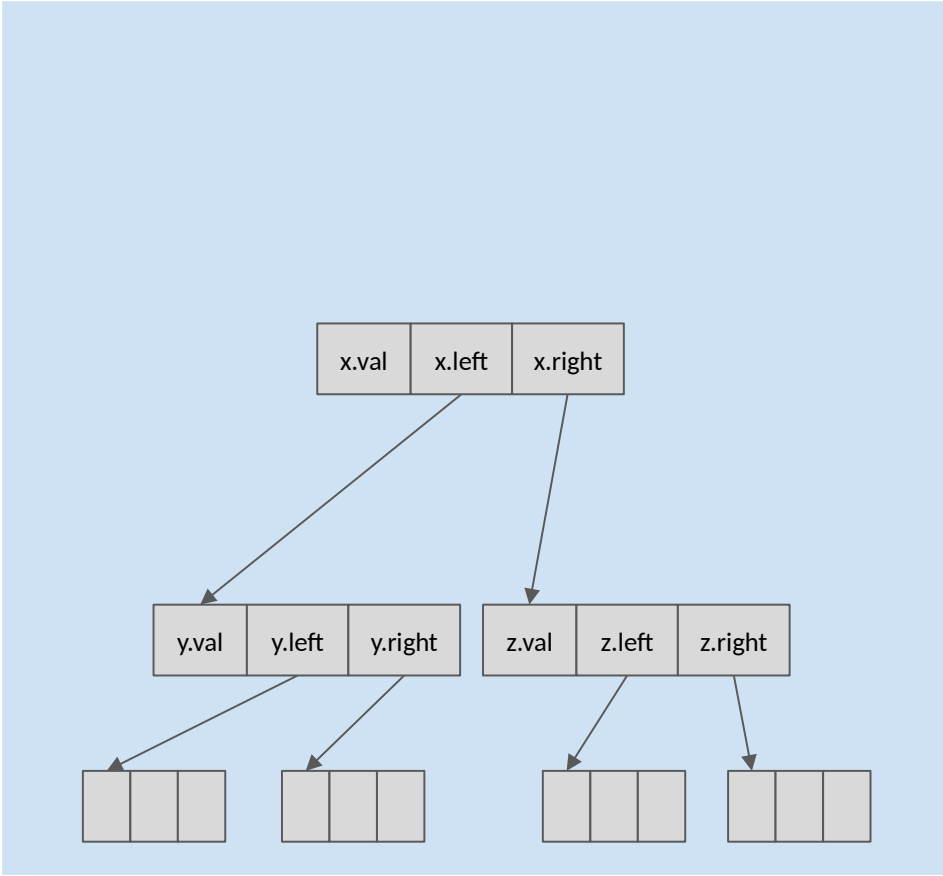
Fractional Permissions



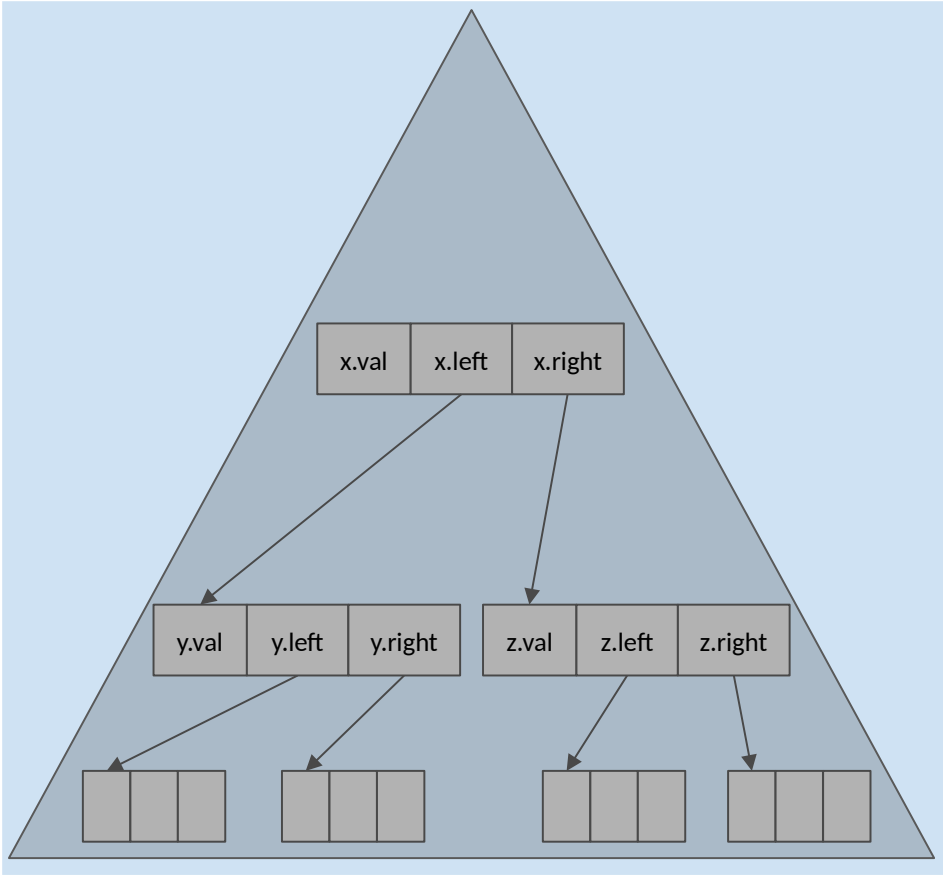
Fractional Permissions



Fractional Predicates



Fractional Predicates

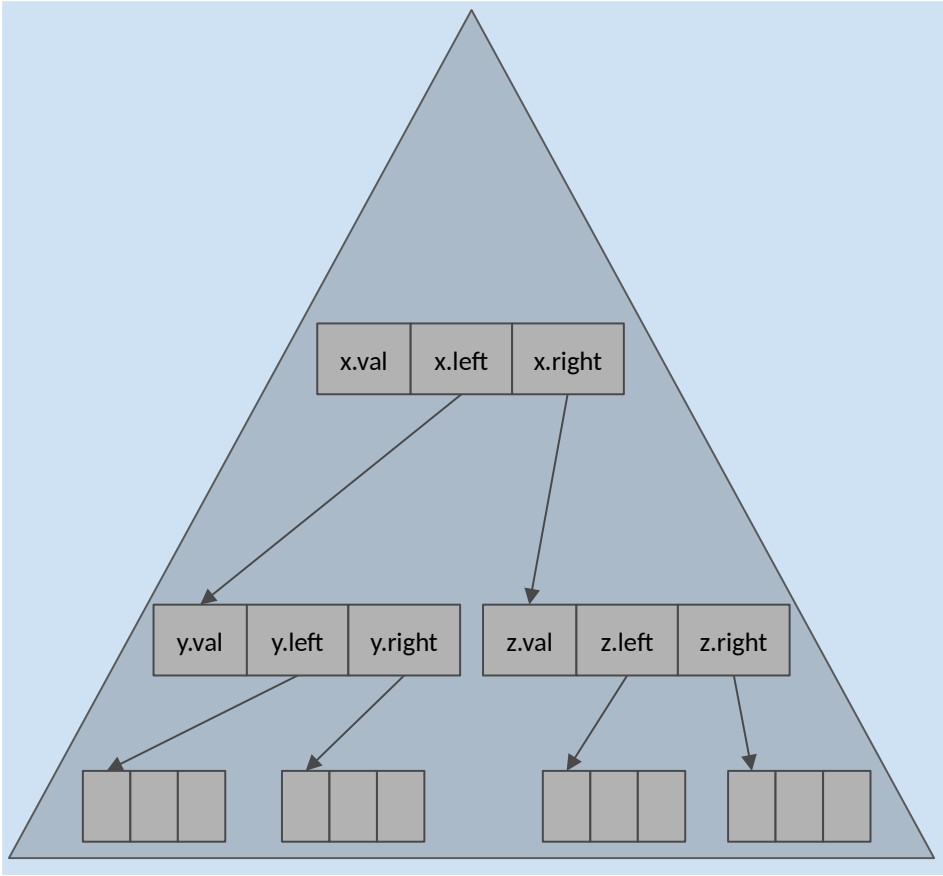


$$\text{tree}(x) \triangleq (x = \textbf{null} \vee (x.\text{val} \mapsto \_ * (\exists l, r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$$

Fractional Predicates

Permits to **read** and **write** nodes

tree(x)



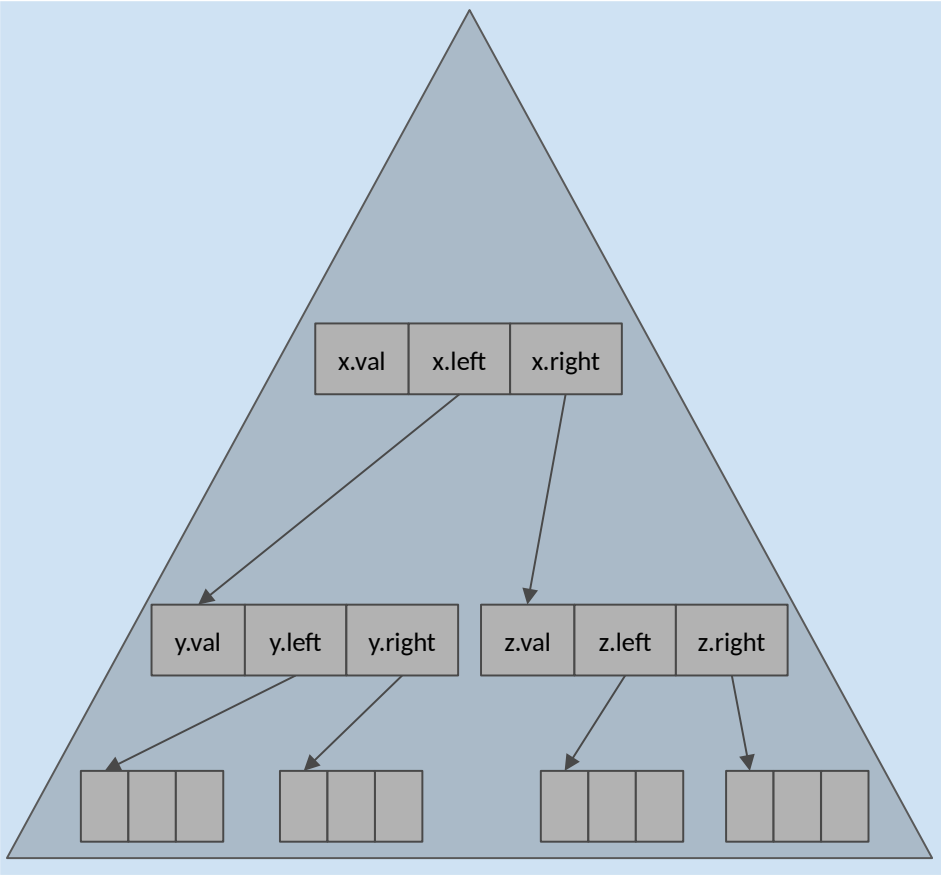
$$\text{tree}(x) \triangleq (x = \textbf{null} \vee (x.\text{val} \mapsto \_ * (\exists l, r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$$

Fractional Predicates

Permits to **read** and **write** nodes

tree(x)

α ⊙ tree(x)



tree(x) ≜ (x = **null** ∨ (x.val ↦ _ ∗ (∃l, r. x.left ↦ l ∗ x.right ↦ r ∗ tree(l) ∗ tree(r))))

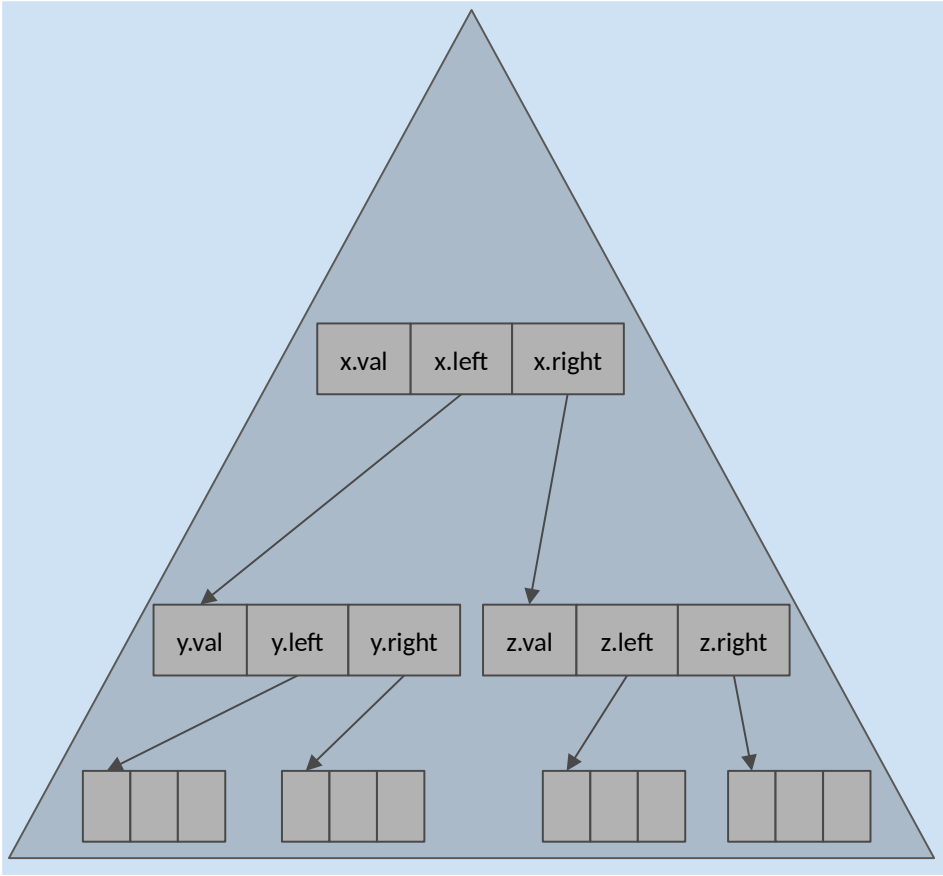
Fractional Predicates

Permits to **read** and **write** nodes

tree(x)

α ⊙ tree(x)

Permits only to **read** nodes



$$\text{tree}(x) \triangleq (x = \textbf{null} \vee (x.\text{val} \mapsto \_ * (\exists l, r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$$

Fractional Predicates – Key Rules

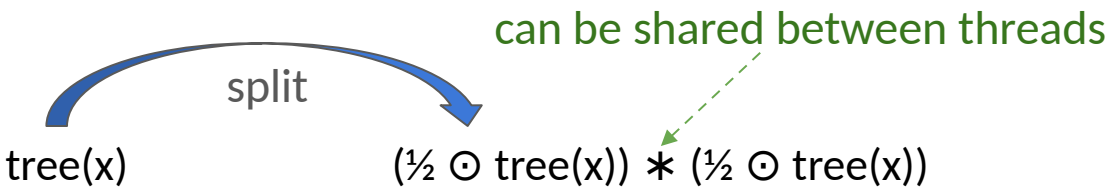
$$\text{tree}(x) \triangleq (x = \textbf{null} \lor (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$$

Fractional Predicates – Key Rules

$\text{tree}(x) \triangleq (x = \textbf{null} \vee (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$

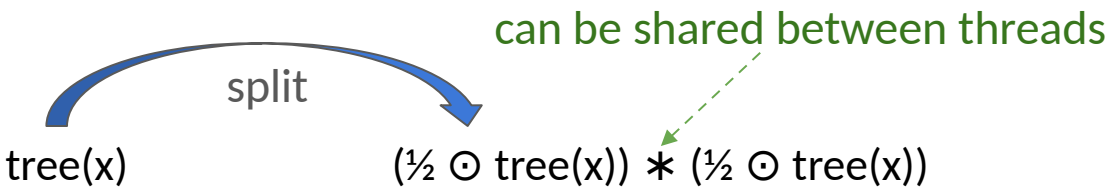
Fractional Predicates – Key Rules

$$\text{tree}(x) \triangleq (x = \textbf{null} \lor (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$$



Fractional Predicates – Key Rules

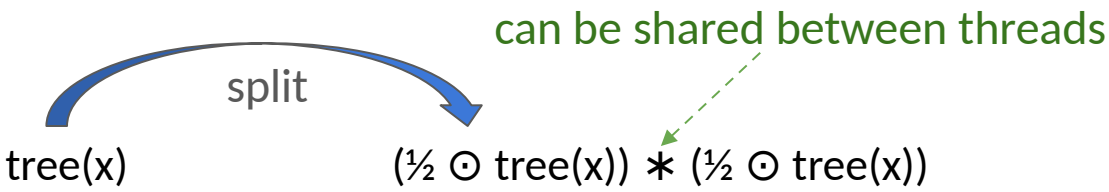
$\text{tree}(x) \triangleq (x = \textbf{null} \lor (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$

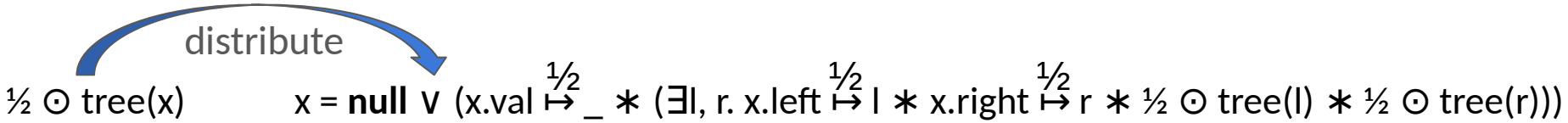


½ ⊙ tree(x)

Fractional Predicates – Key Rules

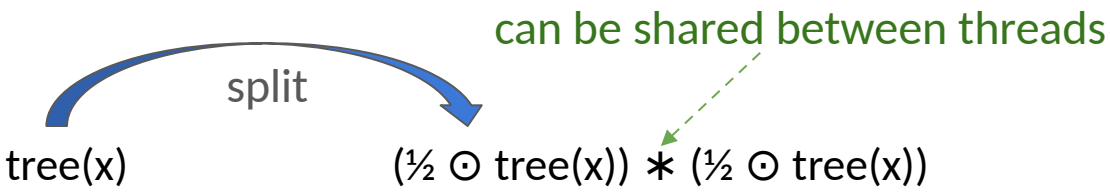
$\text{tree}(x) \triangleq (x = \textbf{null} \lor (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$

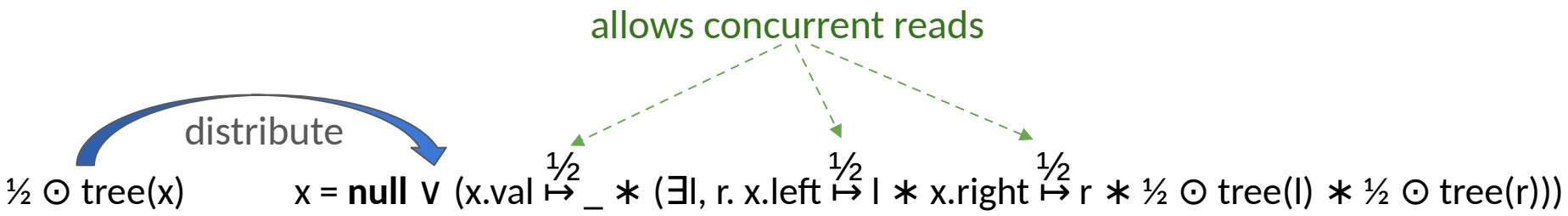




Fractional Predicates – Key Rules

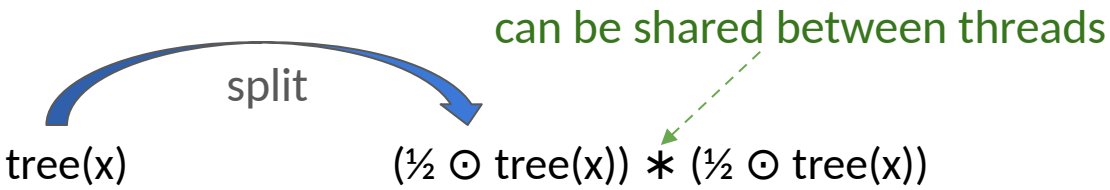
$\text{tree}(x) \triangleq (x = \textbf{null} \lor (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$

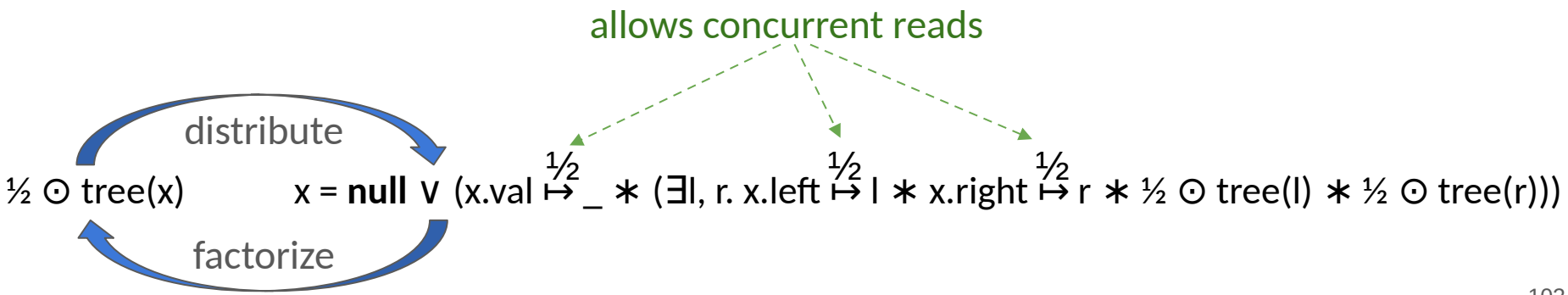




Fractional Predicates – Key Rules

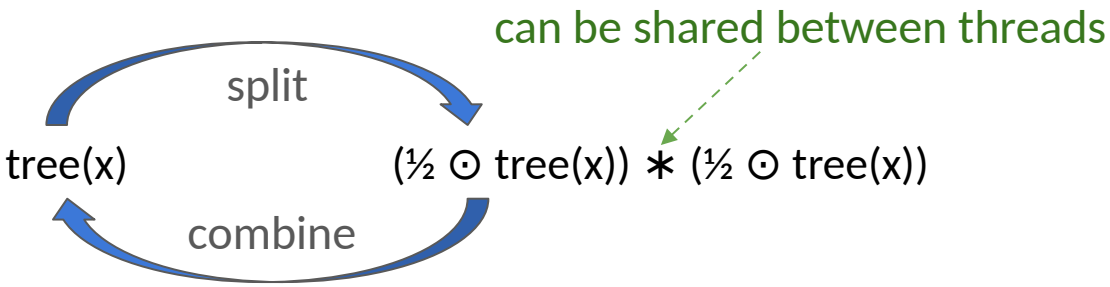
$\text{tree}(x) \triangleq (x = \textbf{null} \lor (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$

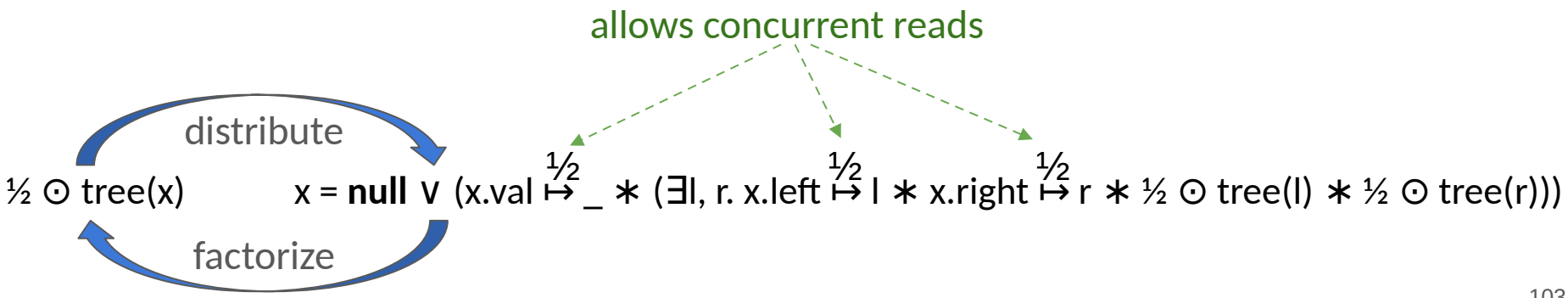




Fractional Predicates – Key Rules

$\text{tree}(x) \triangleq (x = \textbf{null} \lor (x.\text{val} \mapsto \_ * (\exists l.\ \exists r.\ x.\text{left} \mapsto l * x.\text{right} \mapsto r * \text{tree}(l) * \text{tree}(r))))$

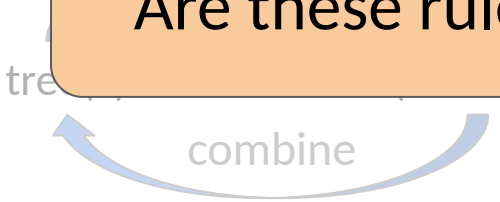




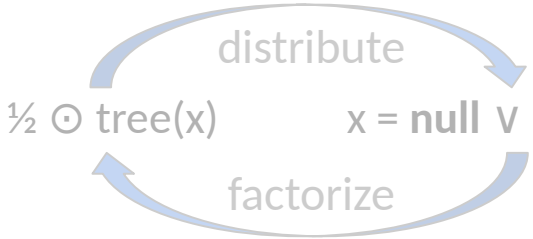
Fractional Predicates – Key Rules

tree(x) ≜ (x = **null** ∨ (x.val ↦ _ ∗ (∃l. ∃r. x.left ↦ l ∗ x.right ↦ r ∗ tree(l) ∗ tree(r))))

Are these rules (used in existing verifiers) **sound**?



allows concurrent reads



Fractional Predicates – Key Rules

tree(x) ≜ (x = **null** ∨ (x.val ↦ _ ∗ (∃l. ∃r. x.left ↦ l ∗ x.right ↦ r ∗ tree(l) ∗ tree(r))))

Are these rules (used in existing verifiers) **sound**?

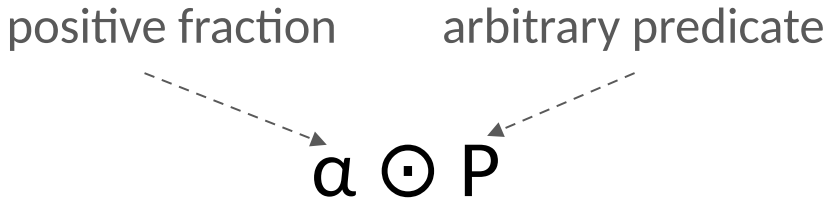
combine

What is the formal meaning of **α ⊙ P**?

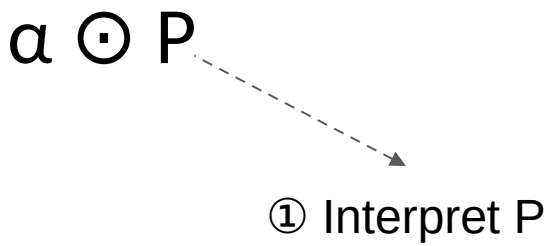
½ ⊙ tree(x) x = **null** ∨ (x.val $\overset{½}{\mapsto}$ _ ∗ (∃l, r. x.left $\overset{½}{\mapsto}$ l ∗ x.right $\overset{½}{\mapsto}$ r ∗ ½ ⊙ tree(l) ∗ ½ ⊙ tree(r)))

factorize

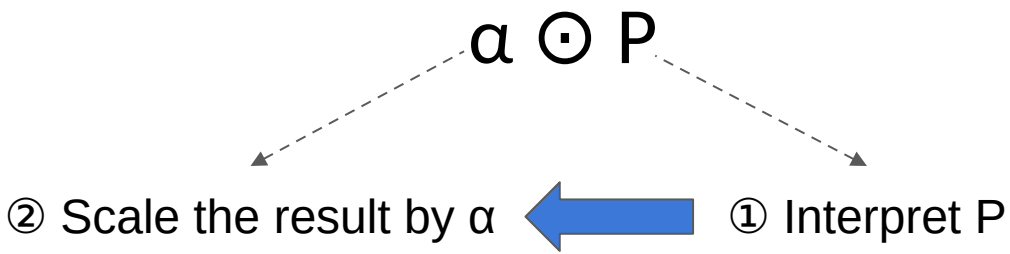
Fractional Predicates – Existing Theory [ESOP'18, CAV'20]



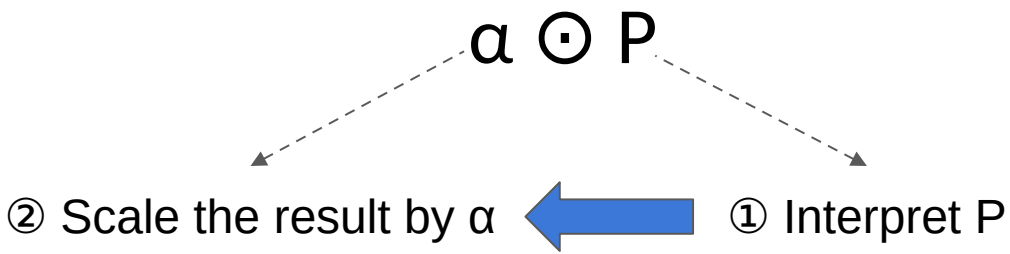
Fractional Predicates – Existing Theory [ESOP'18, CAV'20]



Fractional Predicates – Existing Theory [ESOP'18, CAV'20]

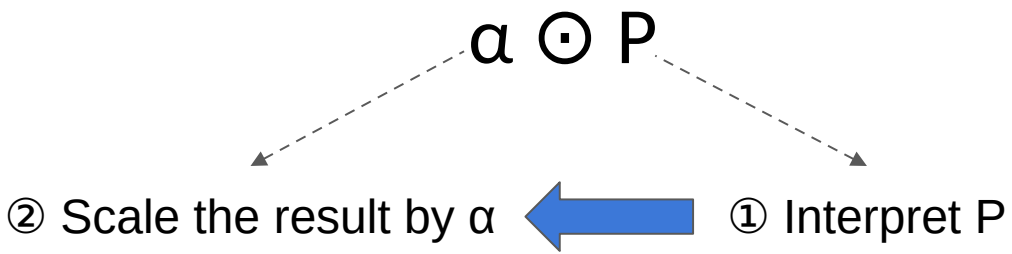


Fractional Predicates – Existing Theory [ESOP'18, CAV'20]



	Split	Combine	Distribute	Factorize
Existing theory	✓	~	✓	✗

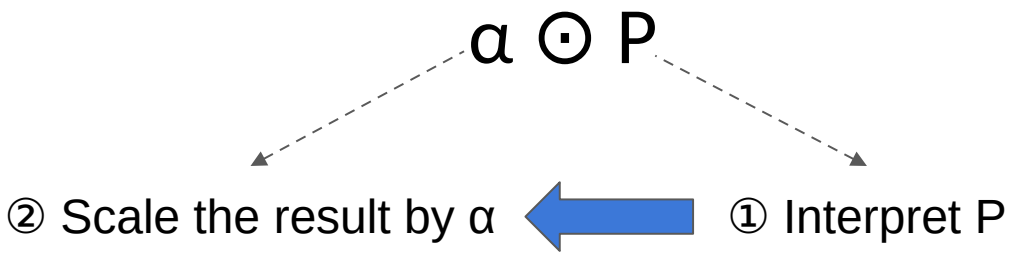
Fractional Predicates – Existing Theory [ESOP'18, CAV'20]



Is this rule fundamentally unsound?

	Split	Combine	Distribute	Factorize
Existing theory	✓	~	✓	✗

Fractional Predicates – Existing Theory [ESOP'18, CAV'20]



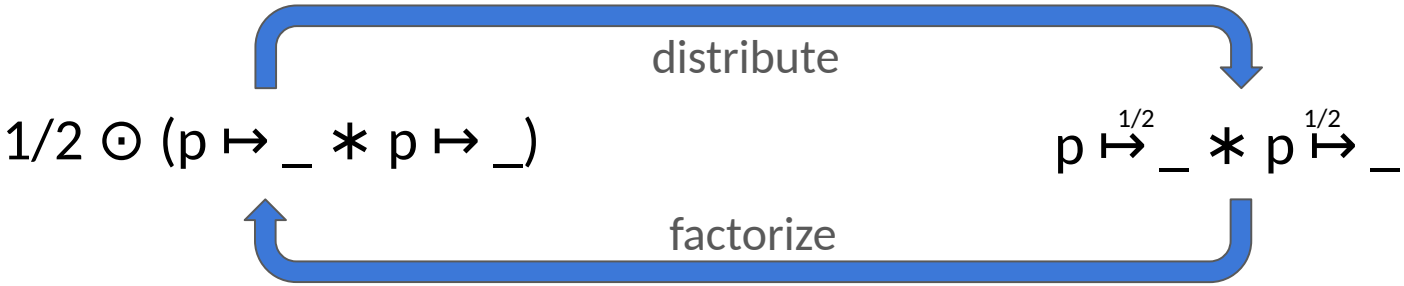
	Split	Combine	Distribute	Factorize
Existing theory	✓	~	✓	✗
Our novel model	✓	✓	✓	✓

Fractional Predicates – Distribute and Factorize

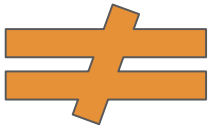
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

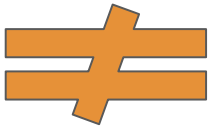
Fractional Predicates – Distribute and Factorize



Fractional Predicates – Distribute and Factorize

$1/2 \odot (p \mapsto \_ * p \mapsto \_)$  $p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

Fractional Predicates – Distribute and Factorize

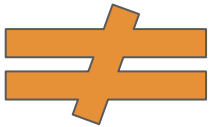
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$  $p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable

Fractional Predicates – Distribute and Factorize

$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

unsatisfiable



$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable

Fractional Predicates – Distribute and Factorize

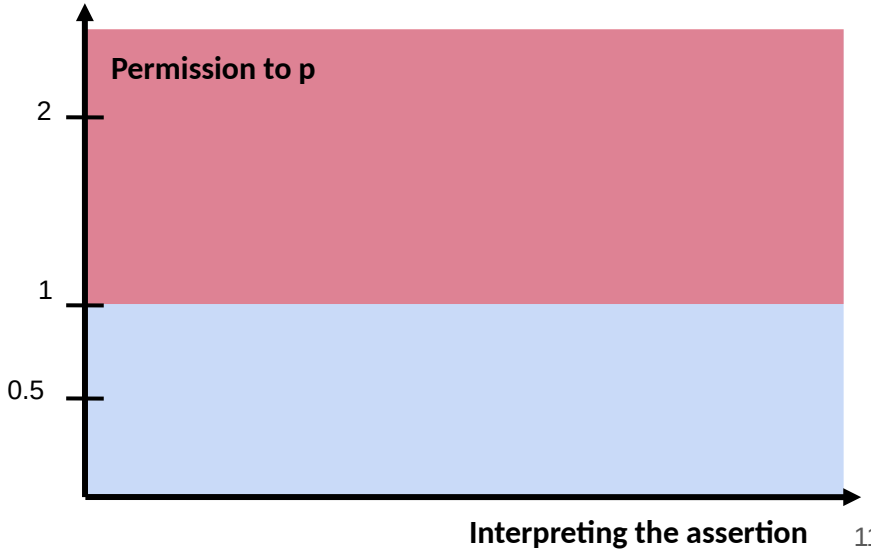
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable



Fractional Predicates – Distribute and Factorize

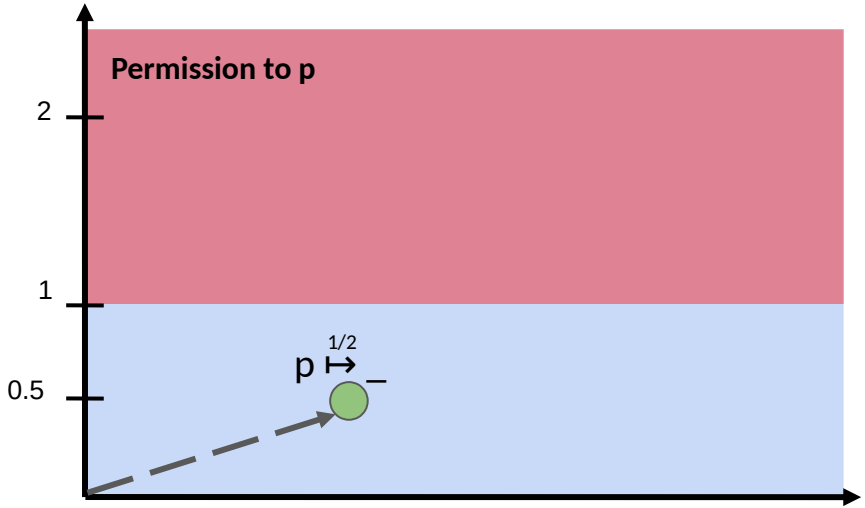
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable



Fractional Predicates – Distribute and Factorize

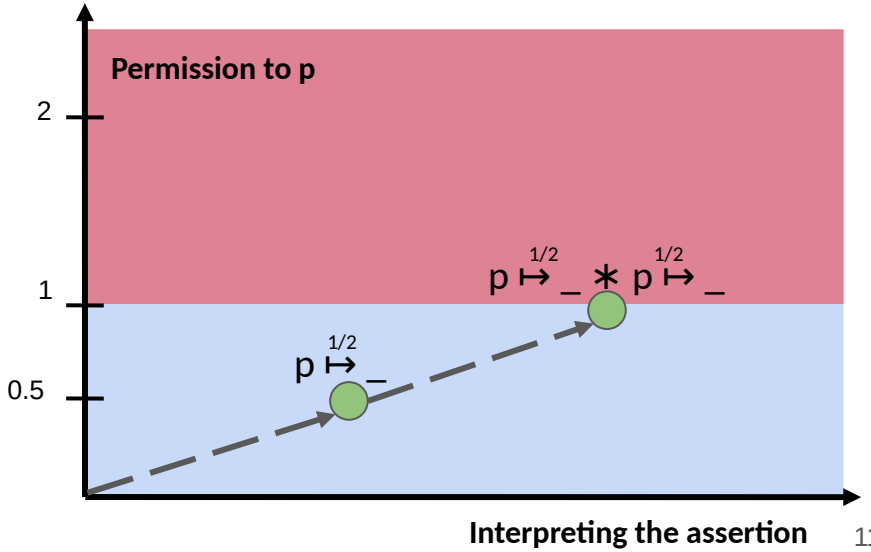
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable



Fractional Predicates – Distribute and Factorize

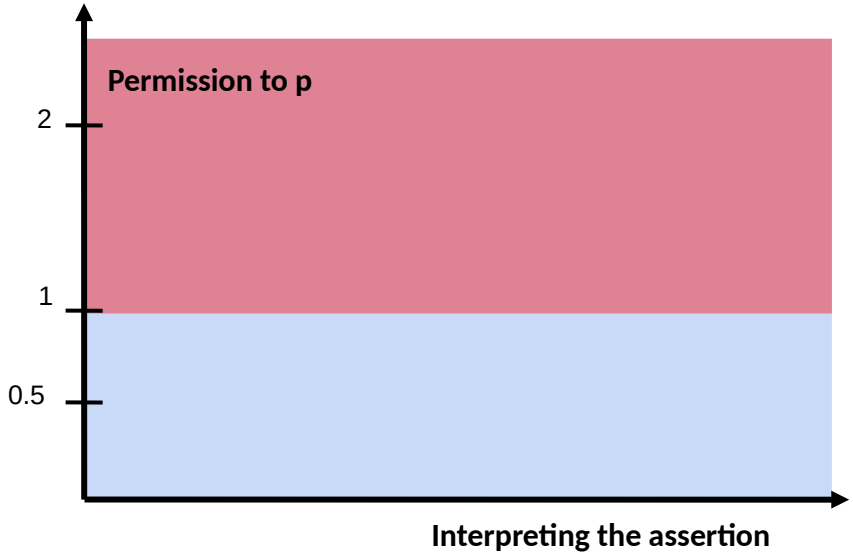
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

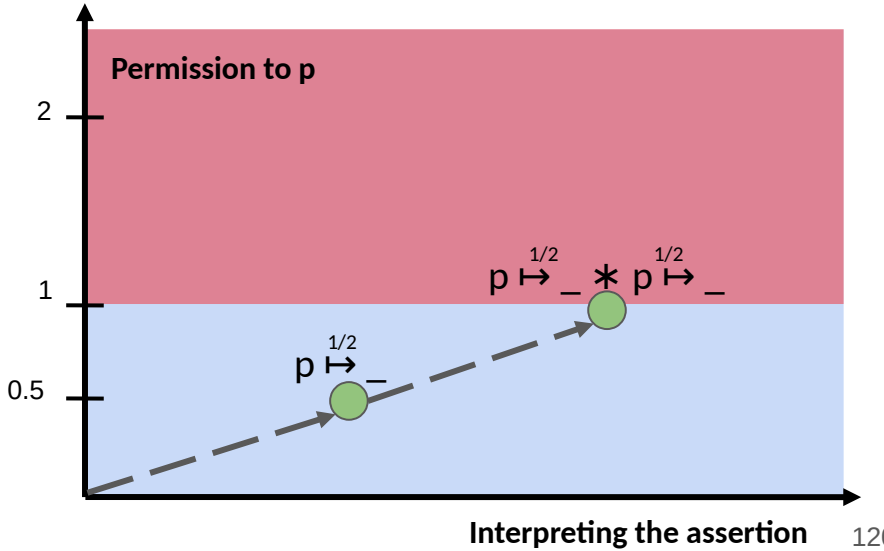
unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable





Fractional Predicates – Distribute and Factorize

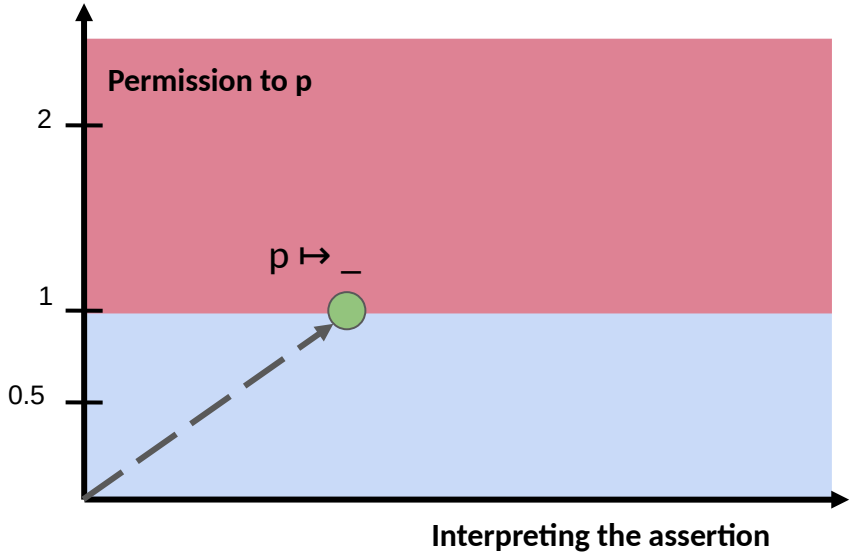
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

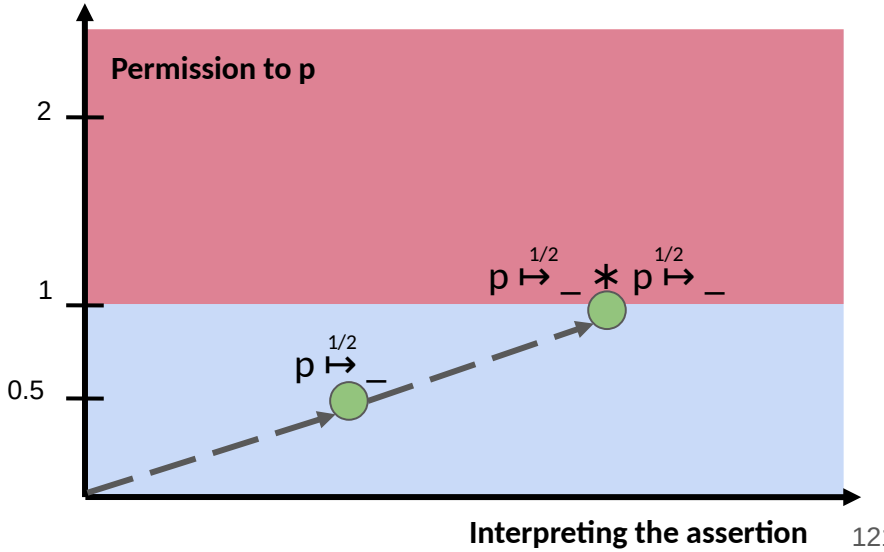
unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable





Fractional Predicates – Distribute and Factorize

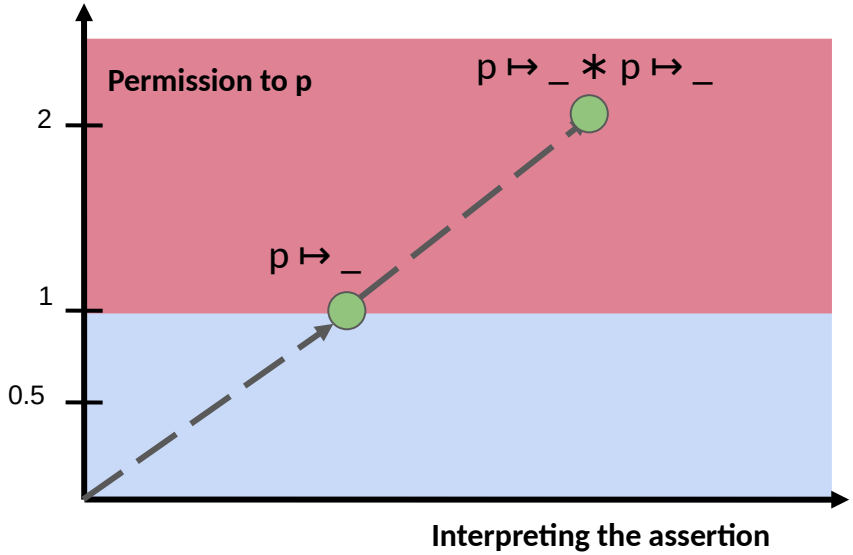
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

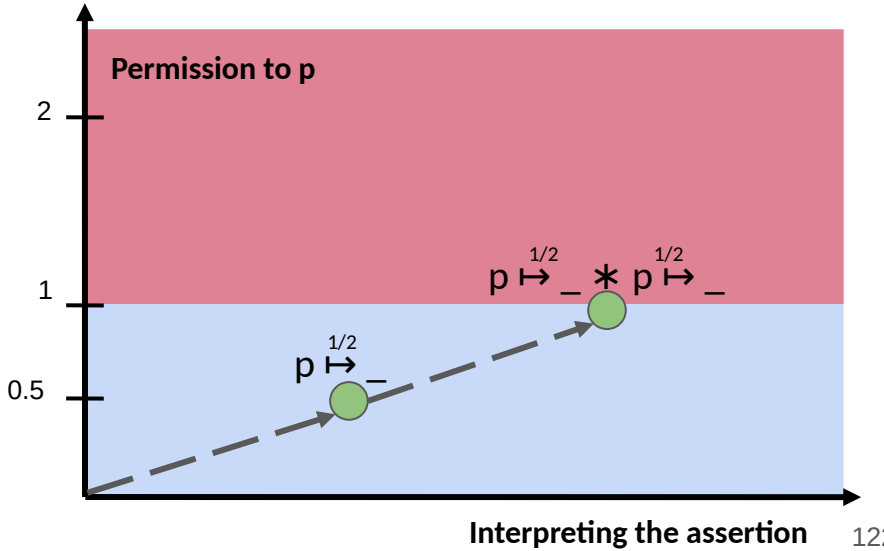
unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable





Fractional Predicates – Distribute and Factorize

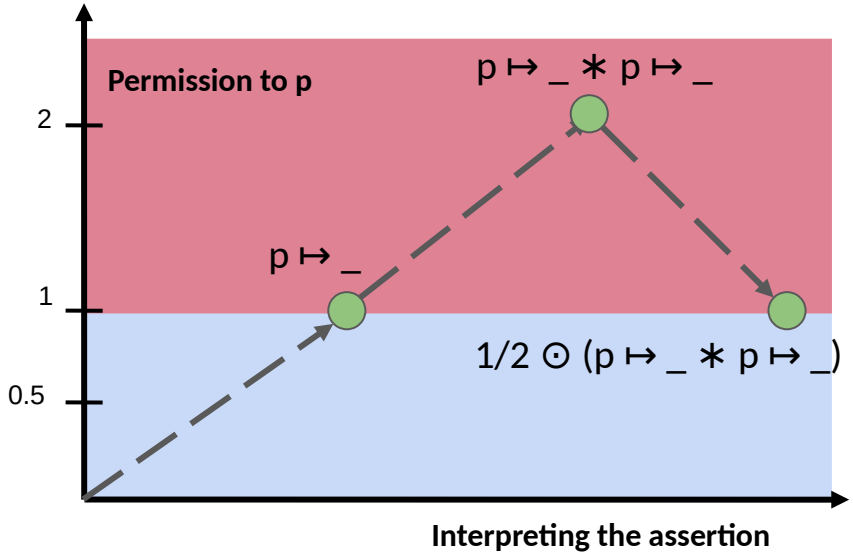
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

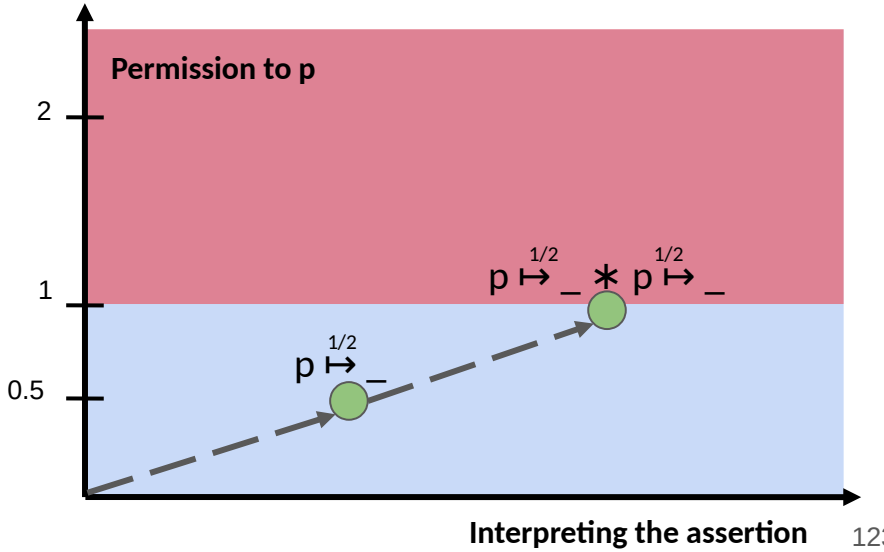
unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable





Fractional Predicates – Distribute and Factorize

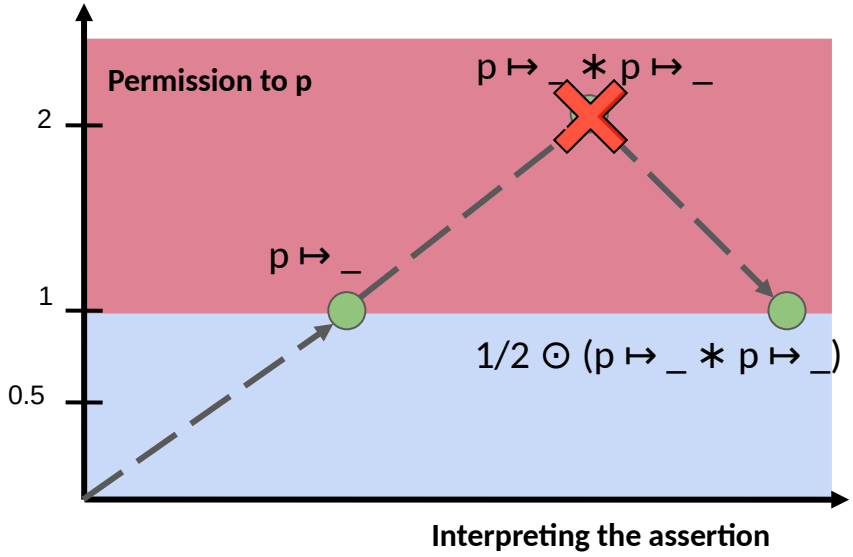
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

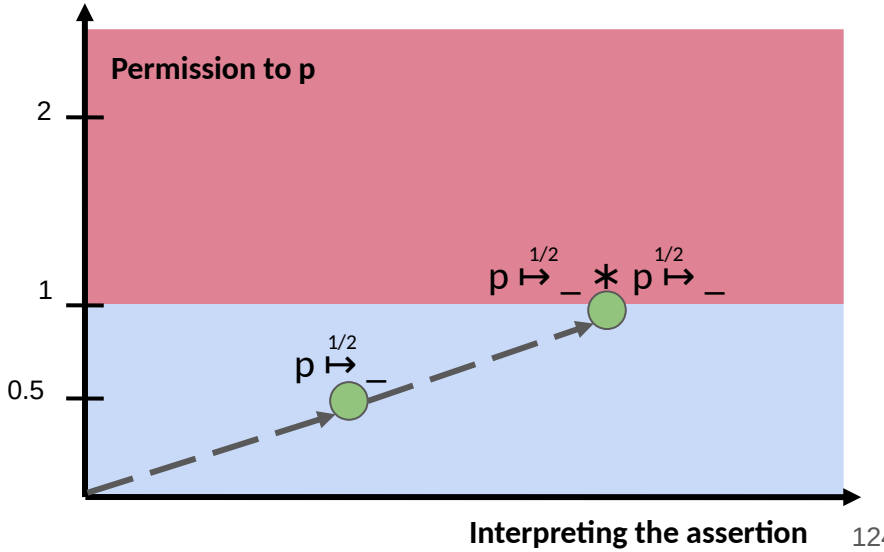
unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable





Fractional Predicates – Distribute and Factorize

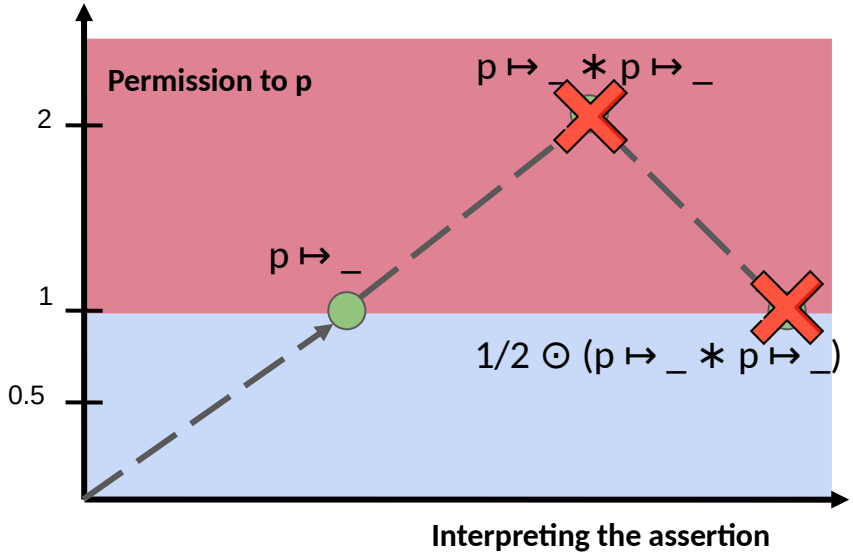
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$

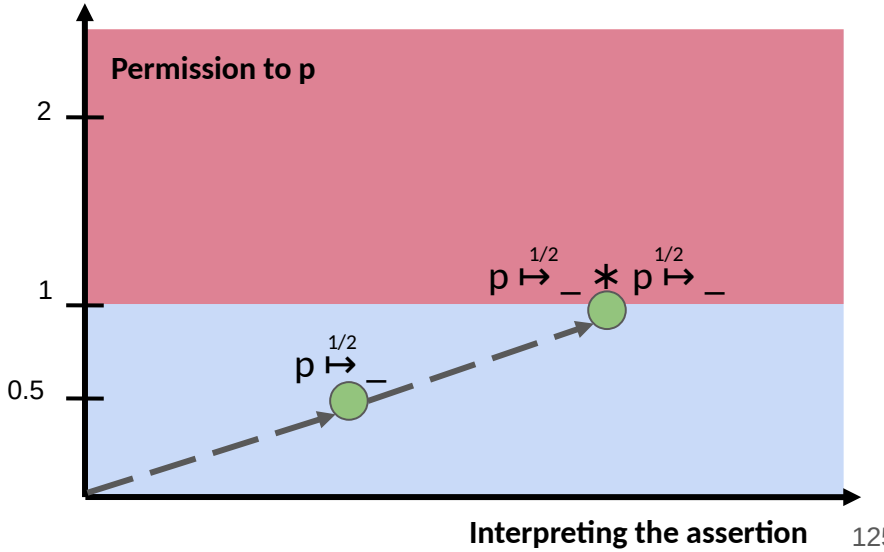
unsatisfiable

$\neq$

$p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

satisfiable





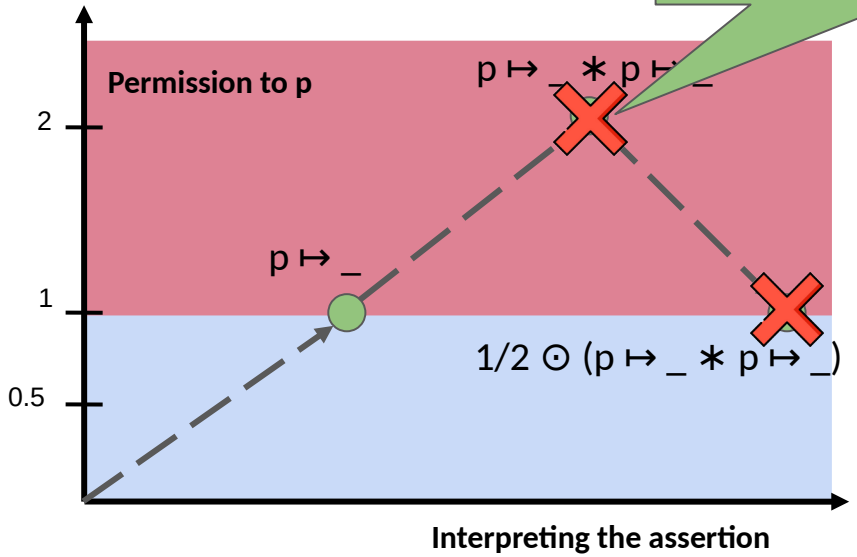
Fractional Predicates – Distribute and Factorize

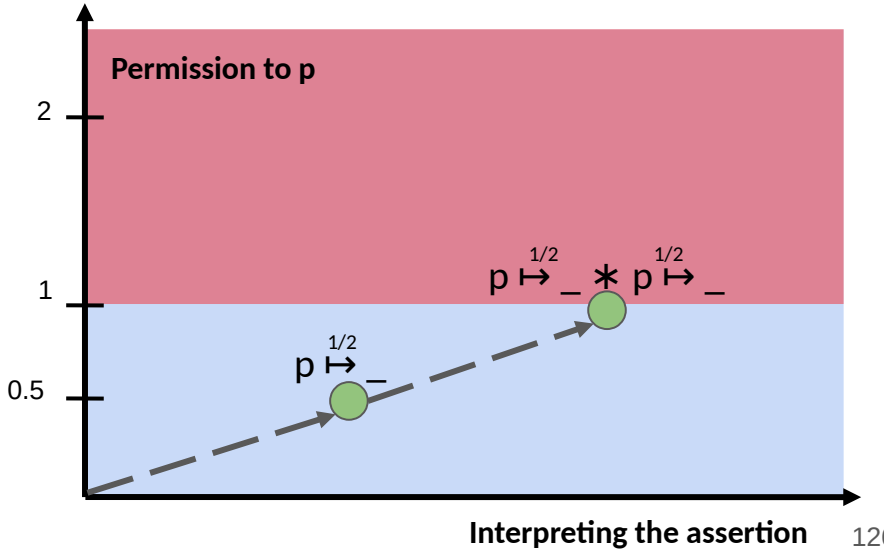
$1/2 \odot (p \mapsto \_ * p \mapsto \_)$ ≠ $p \overset{1/2}{\mapsto} \_ * p \overset{1/2}{\mapsto} \_$

unsatisfiable

satisfiable

Key idea: These states should be **temporarily allowed**





Fractional Predicates – Key Insights

	Split	Combine	Distribute	Factorize
Existing theory	✓	~	✓	✗
Our novel model	✓	✓	✓	✓

Fundamental **novel semantic model** for assertions:
Intermediate invalid states are allowed.

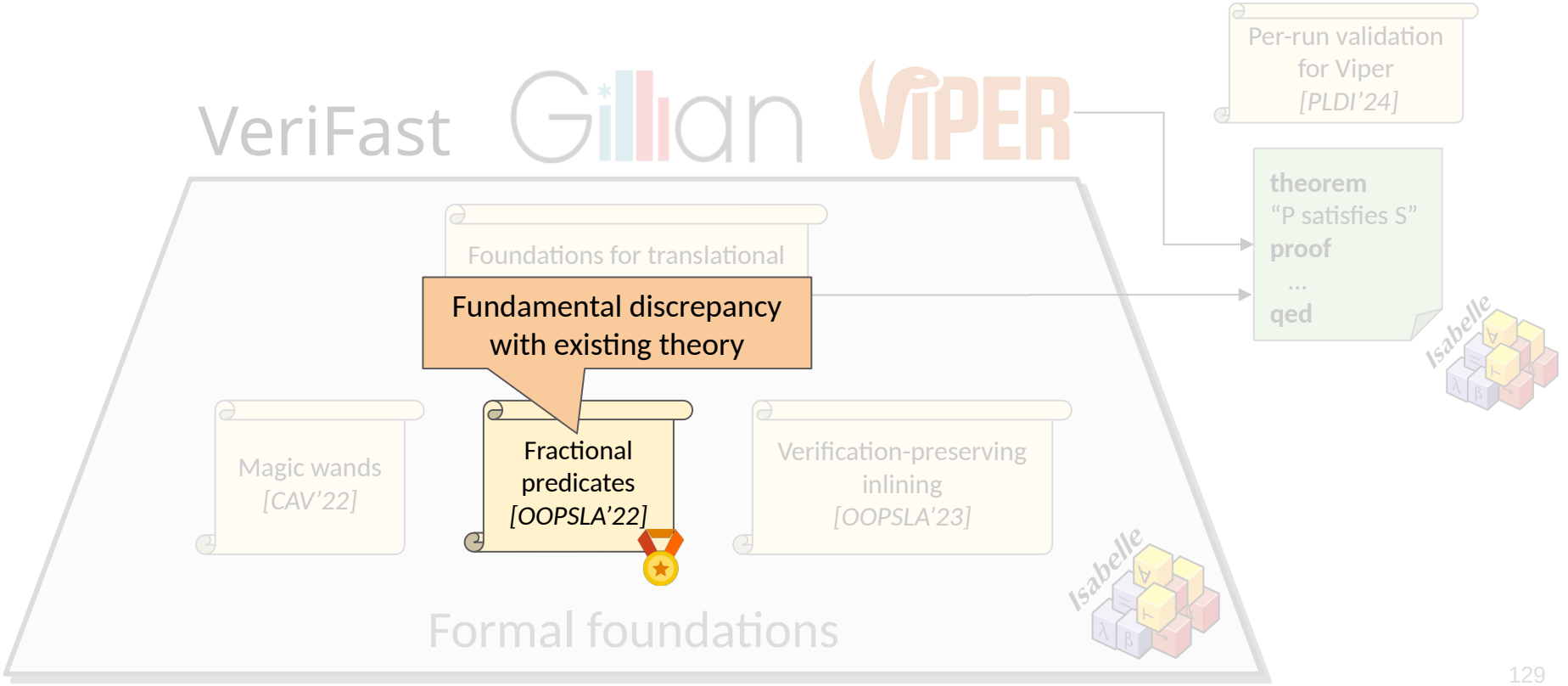
Fractional Predicates – Key Insights

	Split	Combine	Distribute	Factorize
Existing theory	✓	~	✓	✗
Our novel model	✓	✓	✓	✓

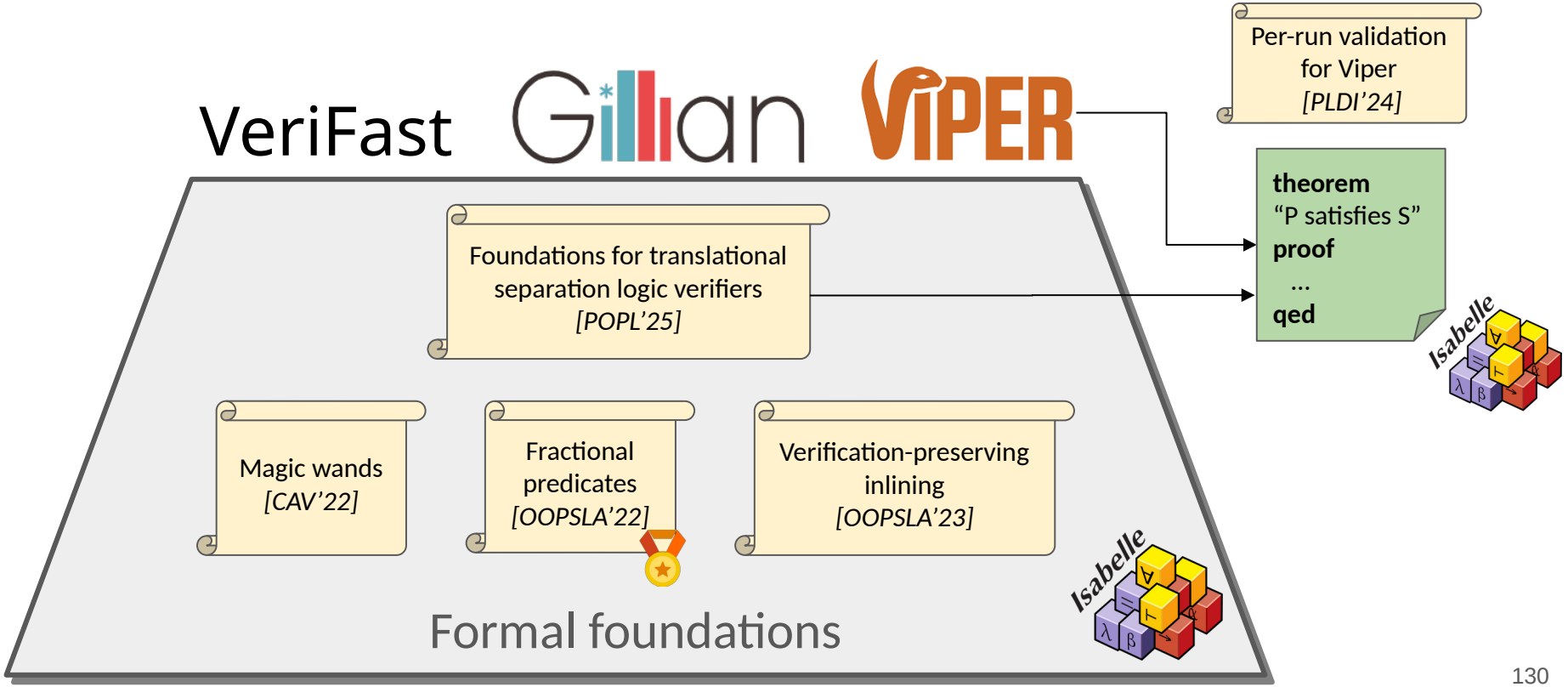
Fundamental **novel semantic model** for assertions: Intermediate invalid states are allowed.

Soundness of the **combine** rule proven via a **novel induction principle** for (co)inductive predicates.

Formal Foundations for Automated Separation Logic Verifiers



Formal Foundations for Automated Separation Logic Verifiers



Formal Foundations for Automated Separation Logic Verifiers

Trustworthy verifier based on separation logic with **state-of-the-art automation**

VeriFast

Gillian

VIPER

Per-run validation for Viper *[PLDI'24]*

theorem
"P satisfies S"
proof
...
qed

Foundations for translational separation logic verifiers *[POPL'25]*

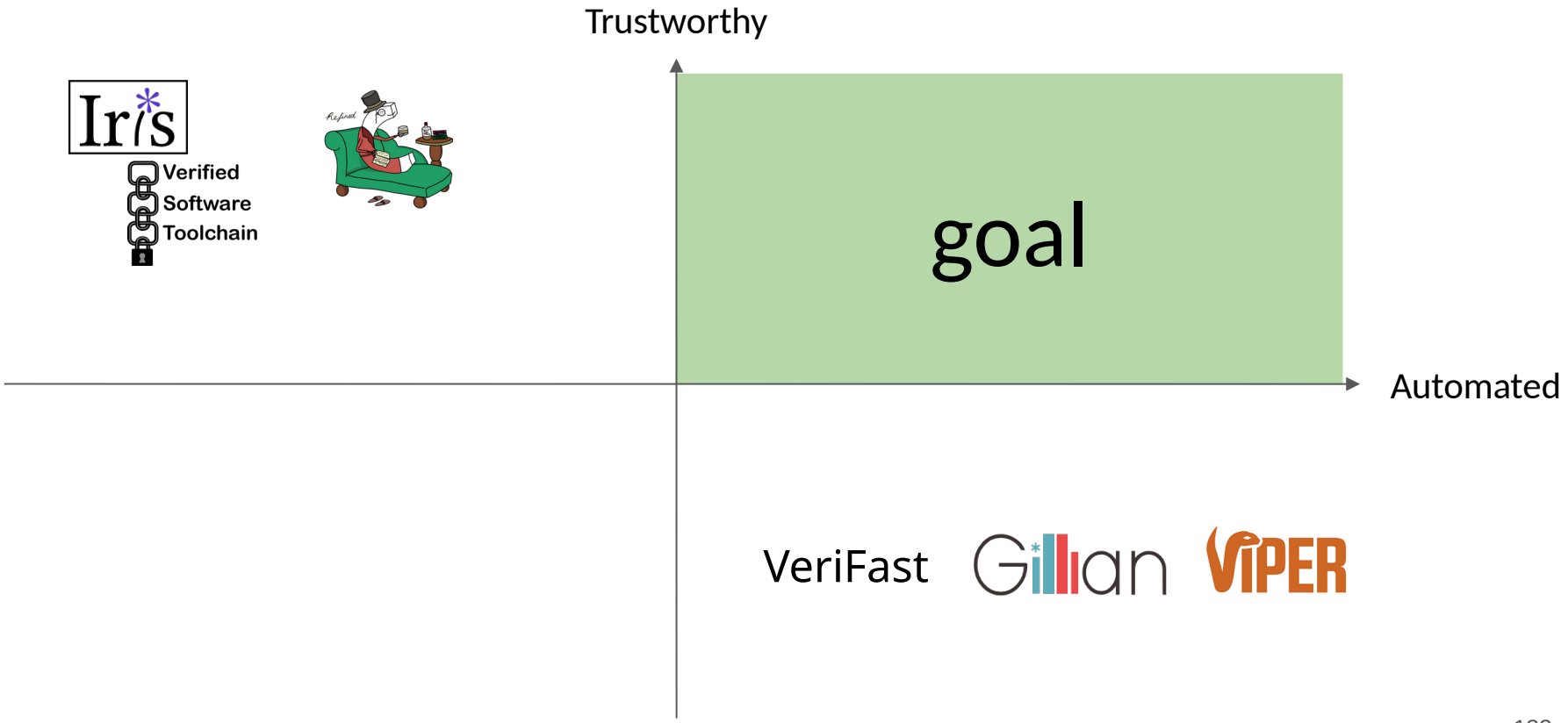
Magic wands *[CAV'22]*

Fractional predicates *[OOPSLA'22]*

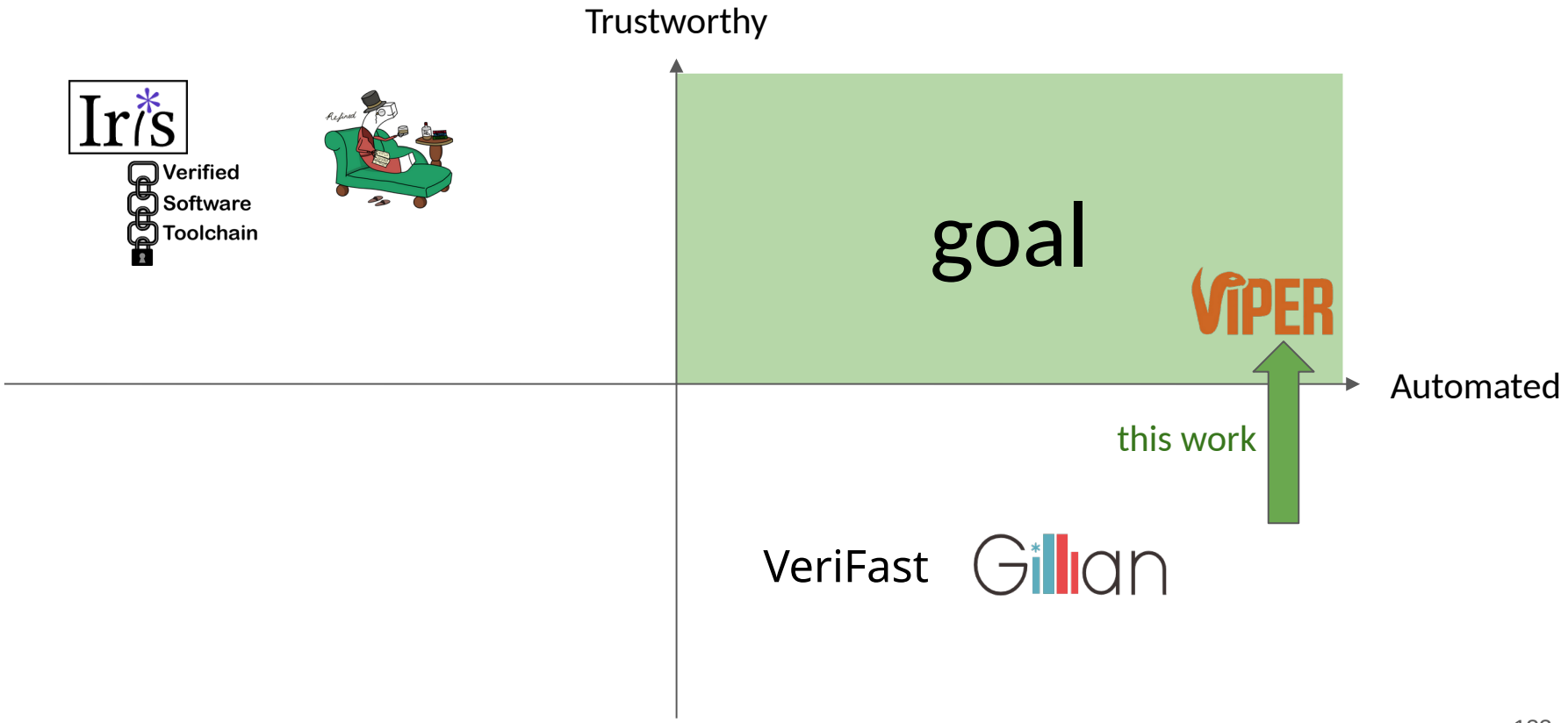
Verification-preserving inlining *[OOPSLA'23]*

Formal foundations

Verifiers Based on Separation Logic



Verifiers Based on Separation Logic



My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Provably sound and automated verifier based on *separation logic*

Problem 2
Expressiveness

Automated verifier for arbitrary *hyperproperties*

My Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Provably sound and automated verifier based on *separation logic*

Problem 2
Expressiveness

Automated verifier for arbitrary *hyperproperties*

Information Flow Issues

Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error means that websites can leak user details including passwords through 'heartbeat' function used to secure connections

Log4shell software flaw threatens millions of servers as hackers scramble to exploit it

Internet

Sat 11 Dec 2021

The extreme ease the flaw allows an attacker to access a server is what experts say makes it so dangerous. *(Supplied: Accenture)*

Information Flow Issues

Confidentiality: Private data leaking to attacker

Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error means that websites can leak user details including passwords through 'heartbeat' function used to secure connections

Log4shell software flaw threatens millions of servers as hackers scramble to exploit it

Internet

Sat 11 Dec 2021

The extreme ease the flaw allows an attacker to access a server is what experts say makes it so dangerous. *(Supplied: Accenture)*

Information Flow Issues

Confidentiality: Private data leaking to attacker

Heartbleed: Hundreds of thousands of servers at risk from catastrophic bug

Code error means that websites can leak user details including passwords through 'heartbeat' function used to secure connections

Integrity: Attacker can execute remote code

Log4shell software flaw threatens millions of servers as hackers scramble to exploit it

Internet

Sat 11 Dec 2021

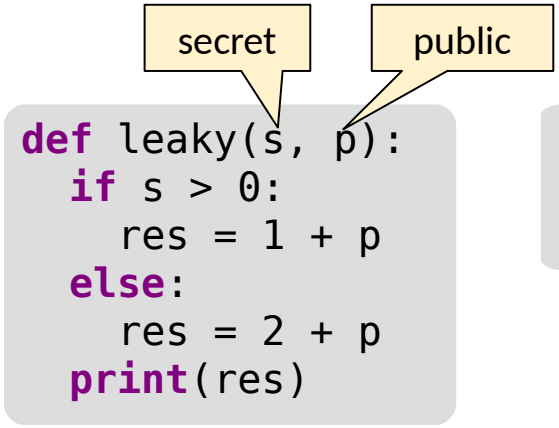
The extreme ease the flaw allows an attacker to access a server is what experts say makes it so dangerous. *(Supplied: Accenture)*

Formalizing Information Flow

```
def leaky(s, p):
  if s > 0:
    res = 1 + p
  else:
    res = 2 + p
  print(res)
```

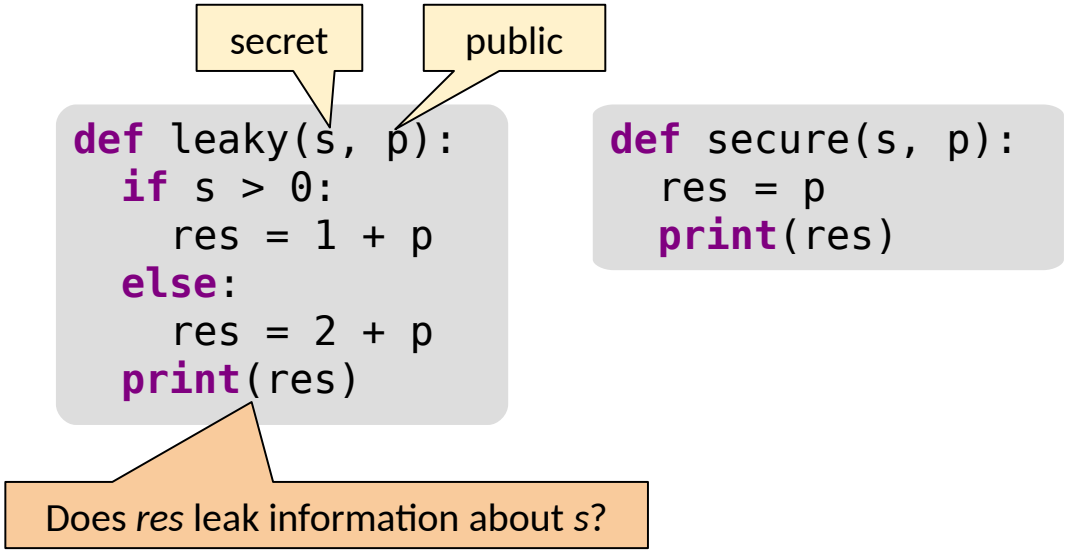
```
def secure(s, p):
  res = p
  print(res)
```

Formalizing Information Flow

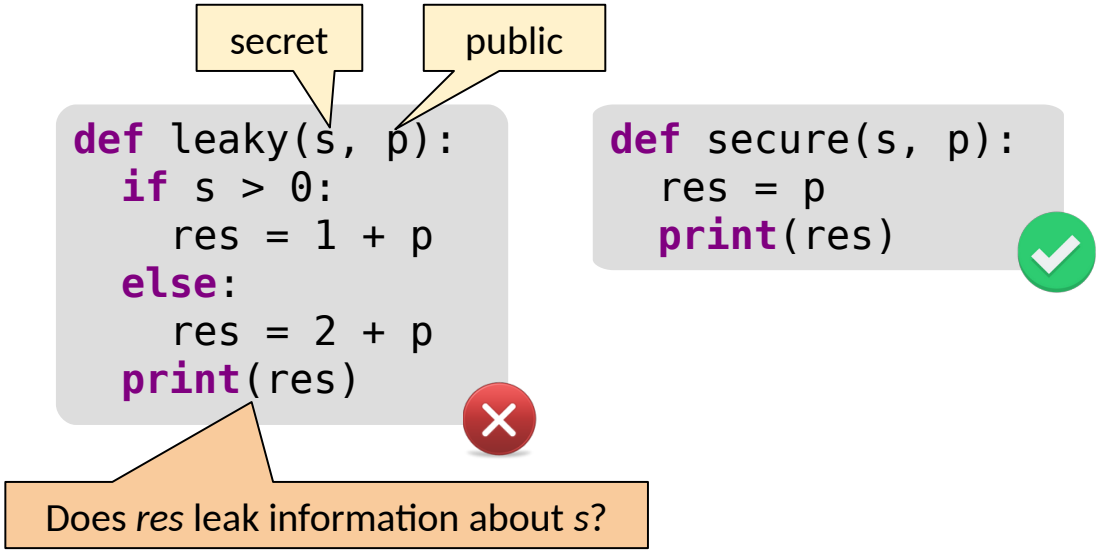



```
def secure(s, p):
  res = p
  print(res)
```

Formalizing Information Flow



Formalizing Information Flow



Formalizing Information Flow

Non-interference ≜ Any two executions with the same public inputs must have the same public outputs.

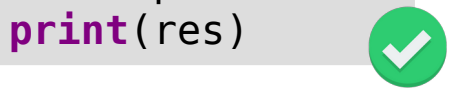
```
def leaky(s, p):
  if s > 0:
    res = 1 + p
  else:
    res = 2 + p
  print(res)
```

```
def secure(s, p):
  res = p
  print(res)
```

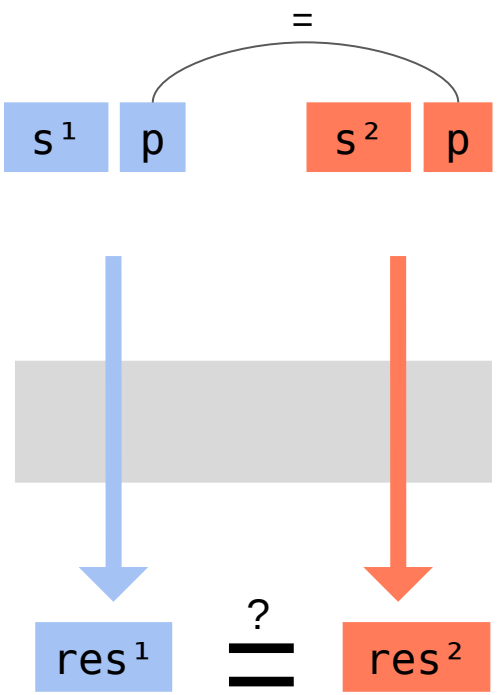
Formalizing Information Flow

Non-interference ≜ Any two executions with the same public inputs must have the same public outputs.

```
def leaky(s, p):
  if s > 0:
    res = 1 + p
  else:
    res = 2 + p
  print(res)
```



```
def secure(s, p):
  res = p
  print(res)
```

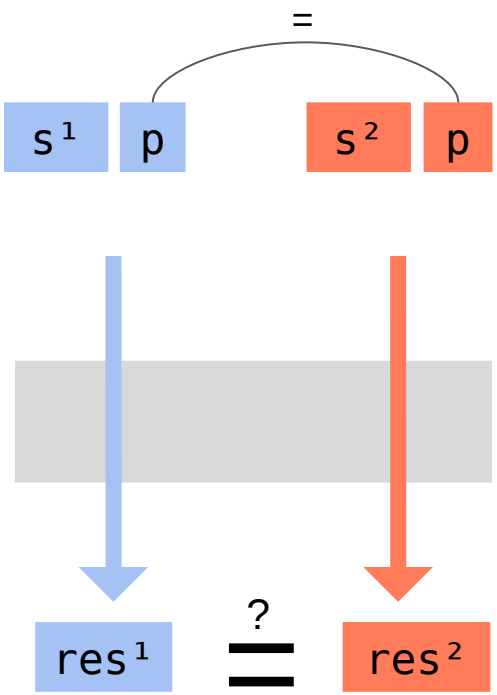


Formalizing Information Flow

Non-interference ≜ Any two executions with the same public inputs must have the same public outputs.

```
def leaky(s, p):
  if s > 0:
    res = 1 + p
  else:
    res = 2 + p
  print(res)
```

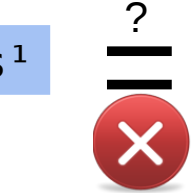
```
def secure(s, p):
  res = p
  print(res)
```



Formalizing Information Flow

Non-interference ≜ Any two executions with the same public inputs must have the same public outputs.

```
def leaky(s, p):
  if s > 0:
    res = 1 + p
  else:
    res = 2 + p
  print(res)
```



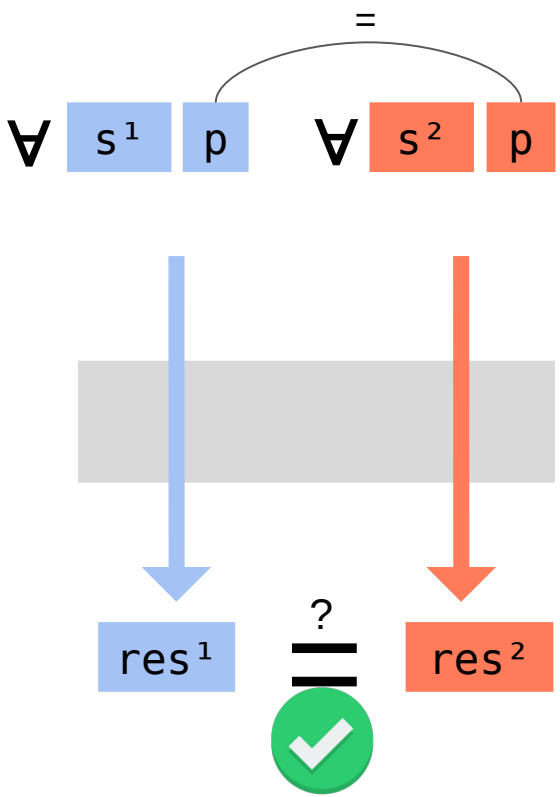
```
def secure(s, p):
  res = p
  print(res)
```

Formalizing Information Flow

Non-interference ≜ Any two executions with the same public inputs must have the same public outputs.

```
def leaky(s, p):
  if s > 0:
    res = 1 + p
  else:
    res = 2 + p
  print(res)
```

```
def secure(s, p):
  res = p
  print(res)
```



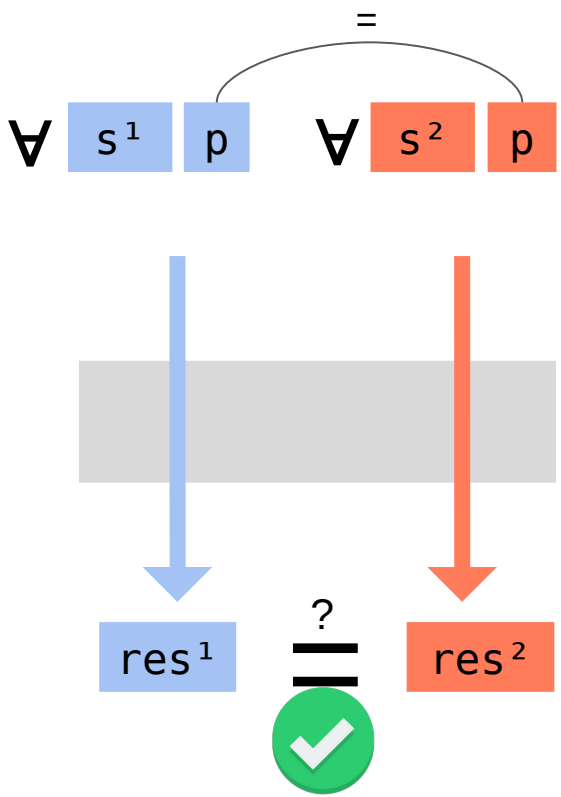
Formalizing Information Flow

Too restrictive for non-deterministic programs

Non-interference ≜ Any two executions with the same public inputs must have the same public outputs.

```
def leaky(s, p):
  if s > 0:
    res = 1 + p
  else:
    res = 2 + p
  print(res)
```

```
def secure(s, p):
  res = p
  print(res)
```

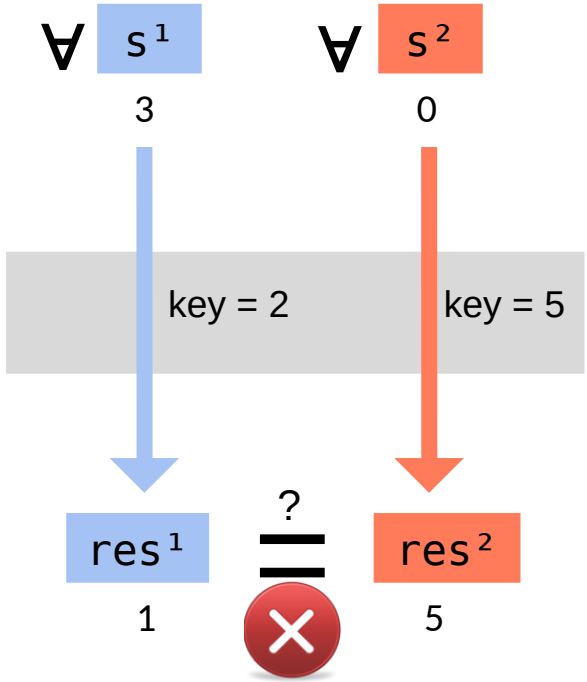


Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```

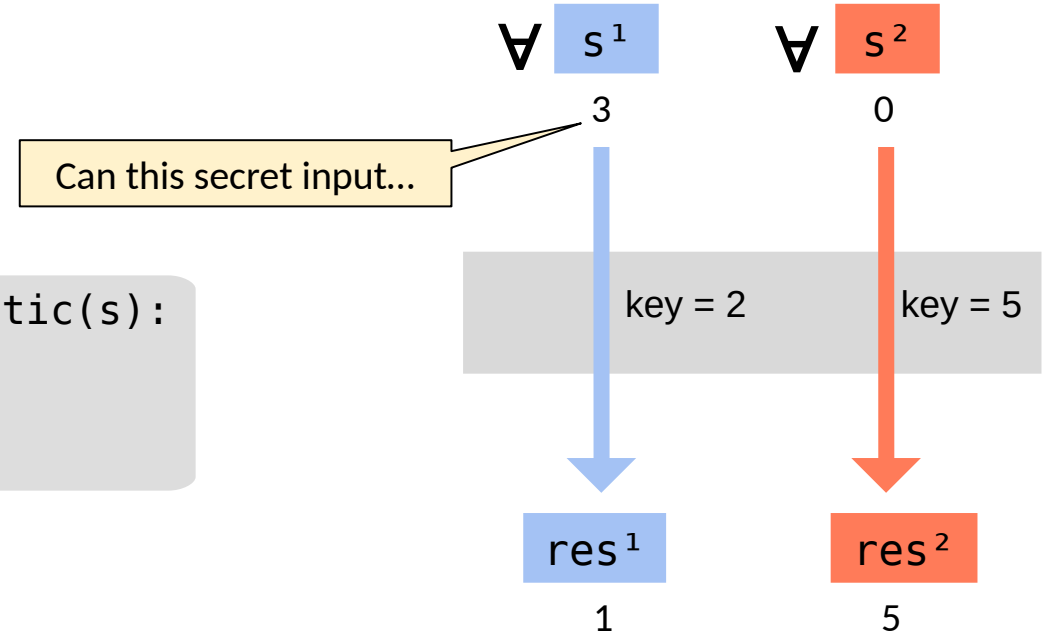
Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



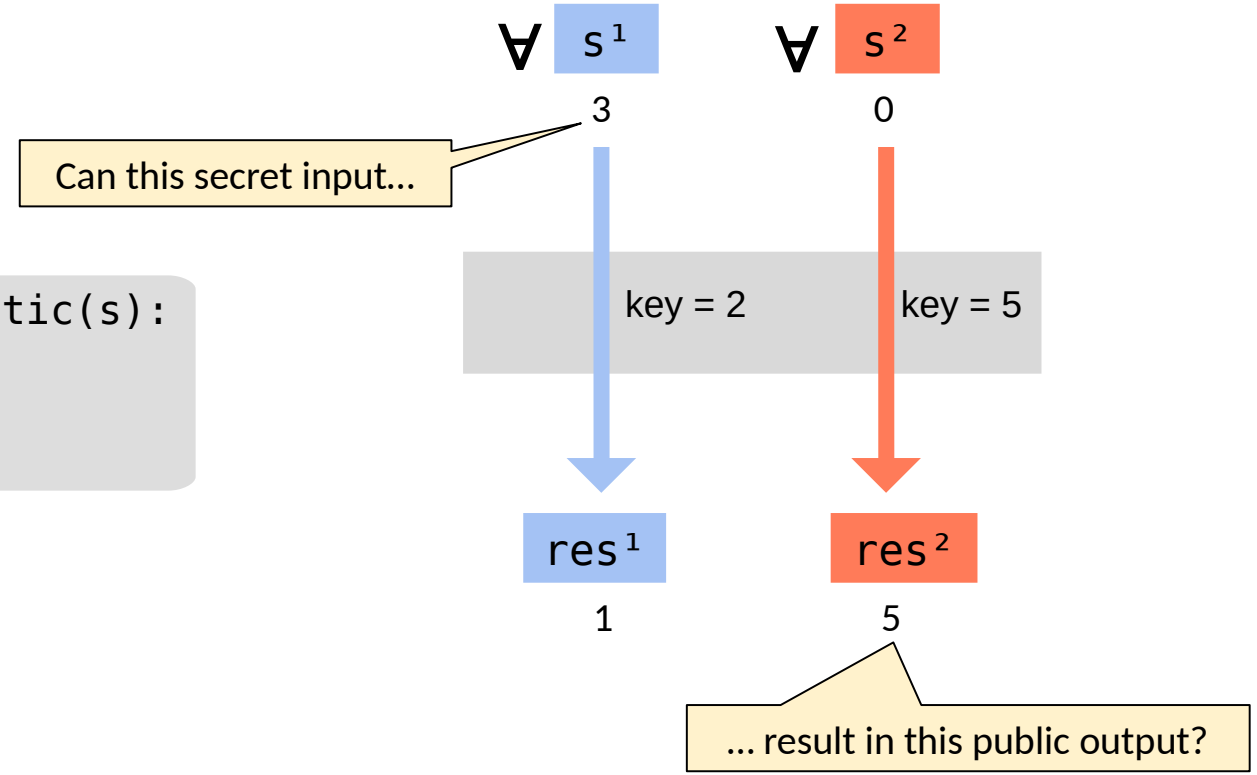
Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



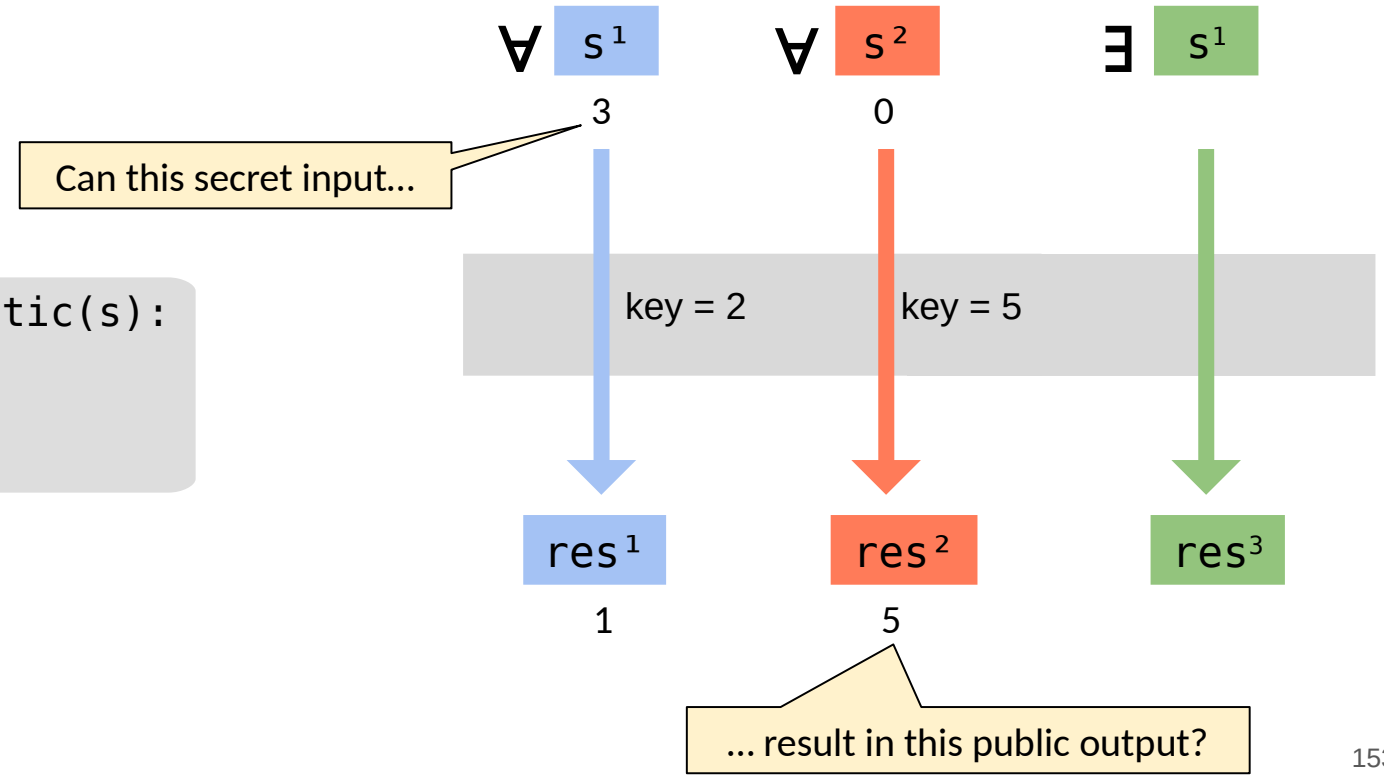
Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



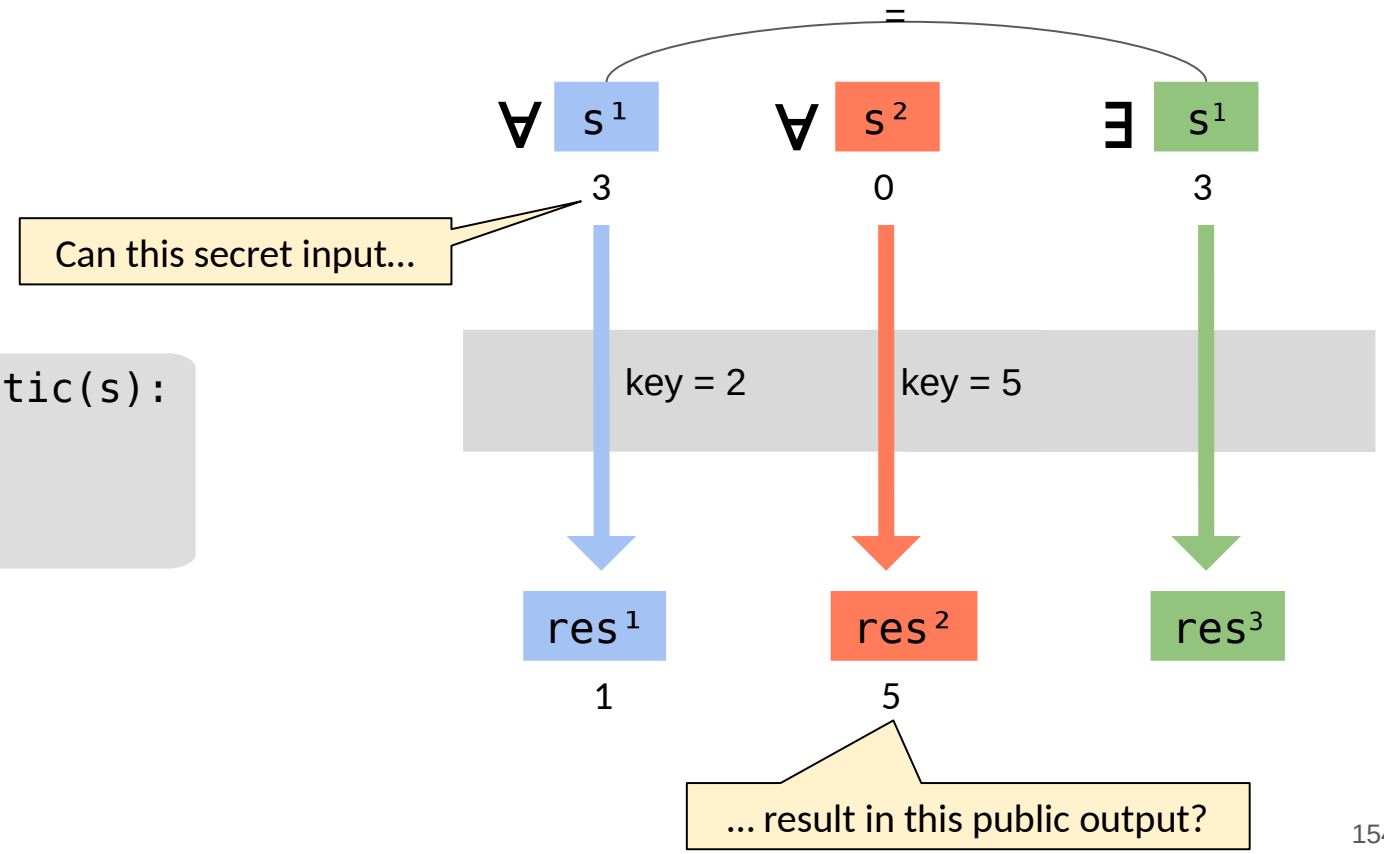
Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



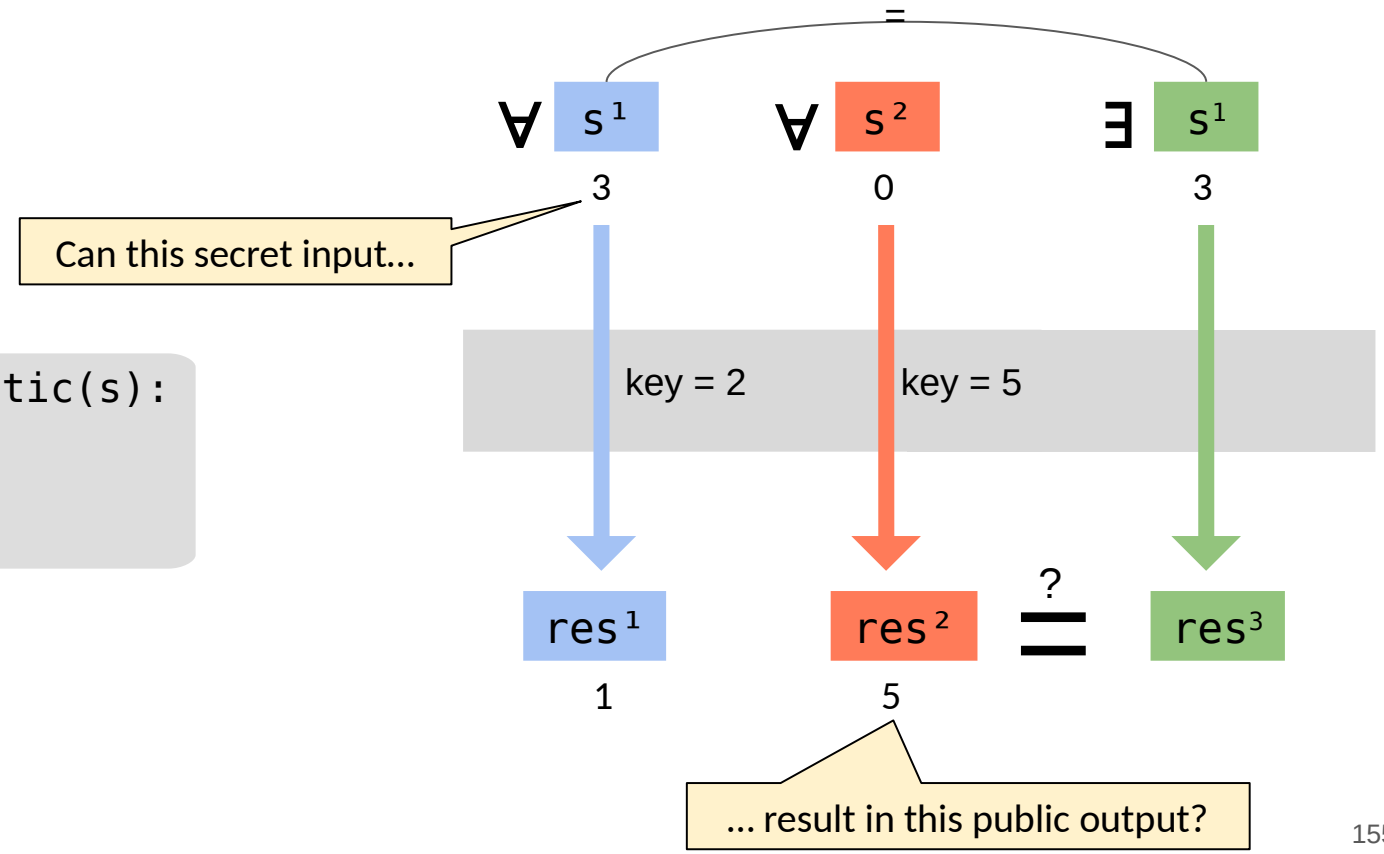
Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



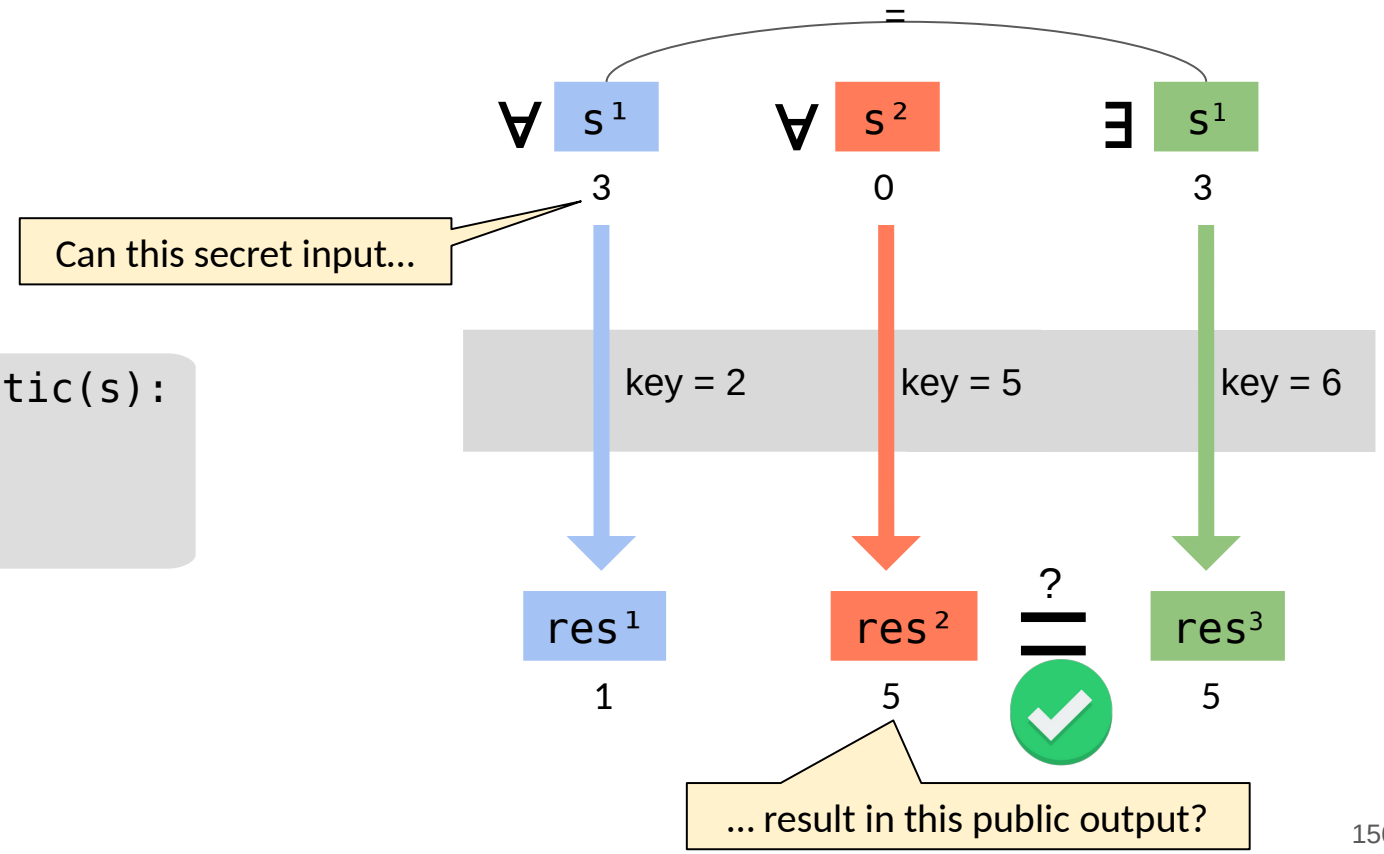
Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



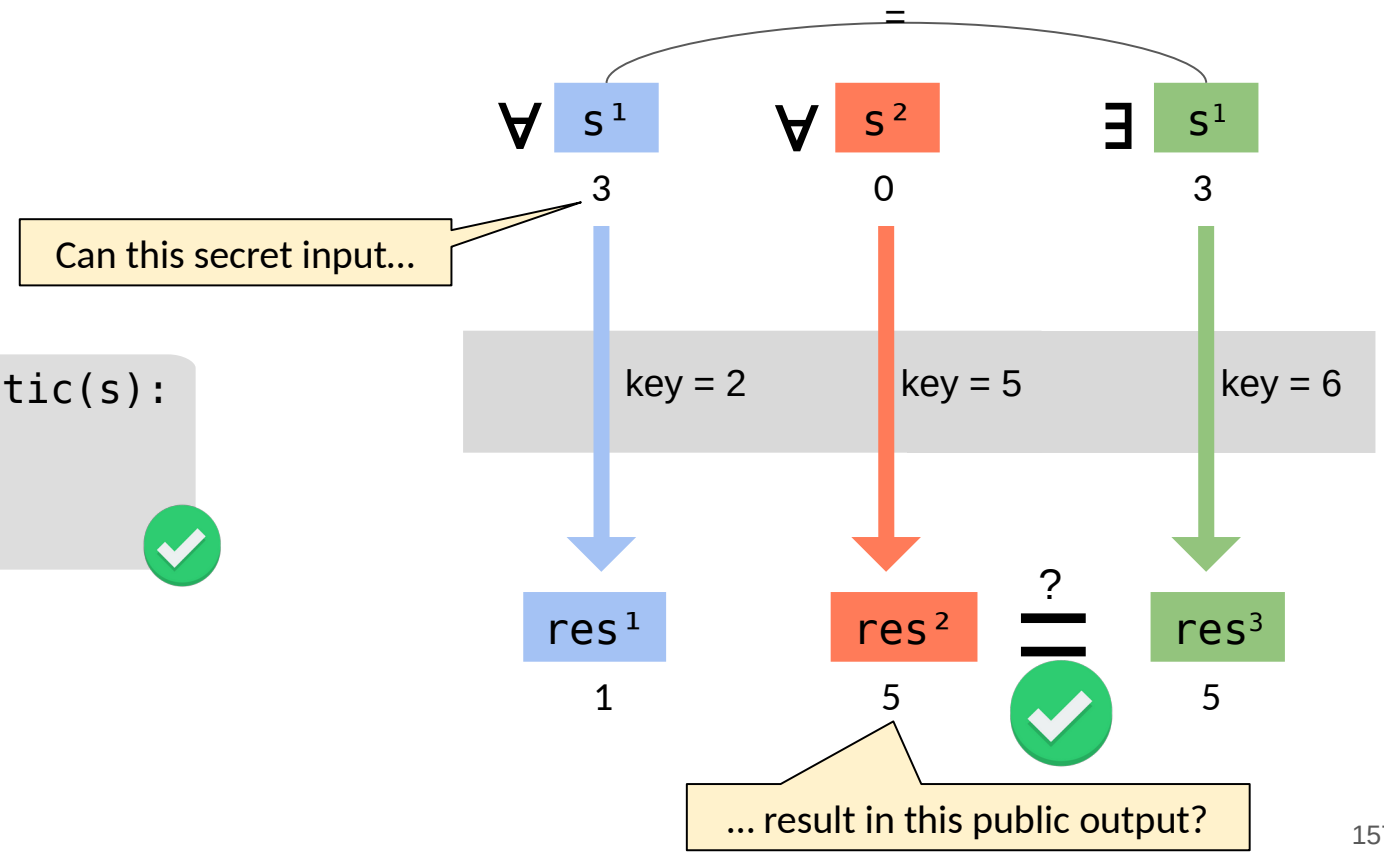
Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



Information Flow Security: *Generalized* Non-Interference

```
def nonDeterministic(s):
    key = nonDet()
    res = s ⊕ key
    print(res)
```



Hyperproperties

Hyperproperty $\triangleq$ property of a **set** of executions

	Type	Example use case
Non-interference	$\forall\forall$	Heartbleed, Log4J
Generalized non-interference	$\forall\forall\exists$	

Hyperproperties

Hyperproperty $\triangleq$ property of a **set** of executions

	Type	Example use case
Non-interference	∀∀	Heartbleed, Log4J
Generalized non-interference	∀∀∃	

"2-safety"

"hyperliveness"

Hyperproperties

Hyperproperty ≜ property of a **set** of executions

	Type	Example use case
Non-interference	∀∀	Heartbleed, Log4J
Generalized non-interference	∀∀∃	
Determinism	∀∀	Hash functions (e.g. for hashmaps)

Hyperproperties

Hyperproperty $\triangleq$ property of a **set** of executions

	Type	Example use case
Non-interference	∀∀	Heartbleed, Log4J
Generalized non-interference	∀∀∃	
Determinism	∀∀	Hash functions (e.g. for hashmaps)
Monotonicity	∀∀	Eventual consistency of CRDTs

Hyperproperties

Hyperproperty $\triangleq$ property of a **set** of executions

	Type	Example use case
Non-interference	∀∀	Heartbleed, Log4J
Generalized non-interference	∀∀∃	
Determinism	∀∀	Hash functions (e.g. for hashmaps)
Monotonicity	∀∀	Eventual consistency of CRDTs
Transitivity	∀∀∀	Custom comparators for sorting

Hyperproperties

Hyperproperty $\triangleq$ property of a **set** of executions

	Type	Example use case
Non-interference	∀∀	Heartbleed, Log4J
Generalized non-interference	∀∀∃	
Determinism	∀∀	Hash functions (e.g. for hashmaps)
Monotonicity	∀∀	Eventual consistency of CRDTs
Transitivity	∀∀∀	Custom comparators for sorting
Existence of a minimum	∃∀	Optimization algorithms

Hyperproperties

Hyperproperty $\triangleq$ property of a **set** of executions

	Type	Example use case
Non-interference	∀∀	Heartbleed, Log4J
Generalized non-interference	∀∀∃	
Determinism	∀∀	Hash functions (e.g. for hashmaps)
Monotonicity	∀∀	Eventual consistency of CRDTs
Transitivity	∀∀∀	Custom comparators for sorting
Existence of a minimum	∃∀	Optimization algorithms
Functional correctness	∀	Everywhere
Reachability	∃	Infer static analyzer

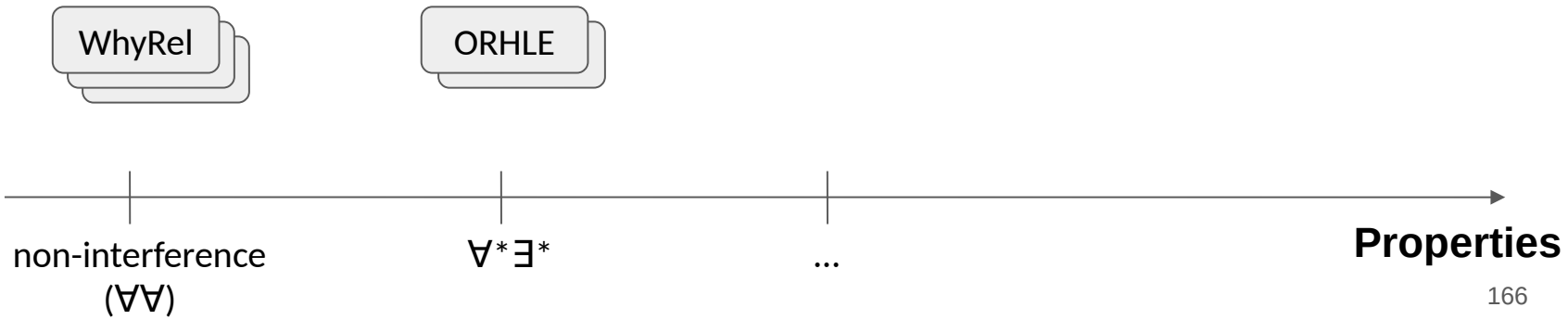
Hyperproperties

Hyperproperty $\triangleq$ property of a **set** of executions

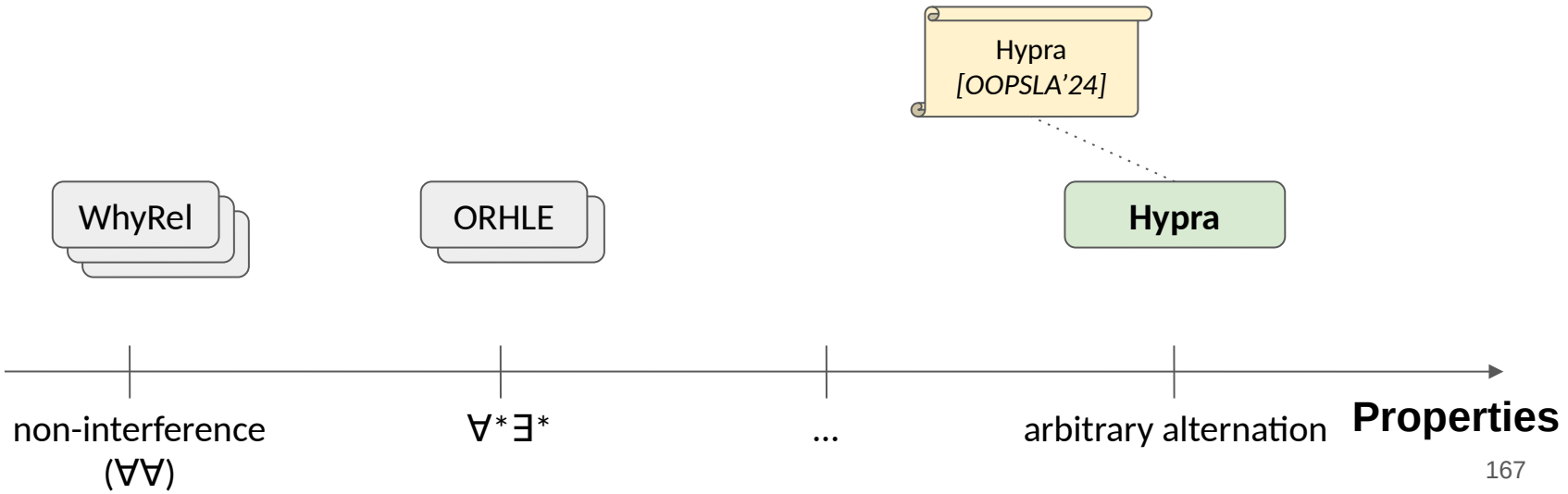
	Type	Example use case
Transitivity	∀∀∀	Custom comparators for sorting
Existence of a minimum	∃∀	Optimization algorithms
Functional correctness	∀	Everywhere
Reachability	∃	Infer static analyzer

Contribution: The first automated verifier that can prove all these hyperproperties.

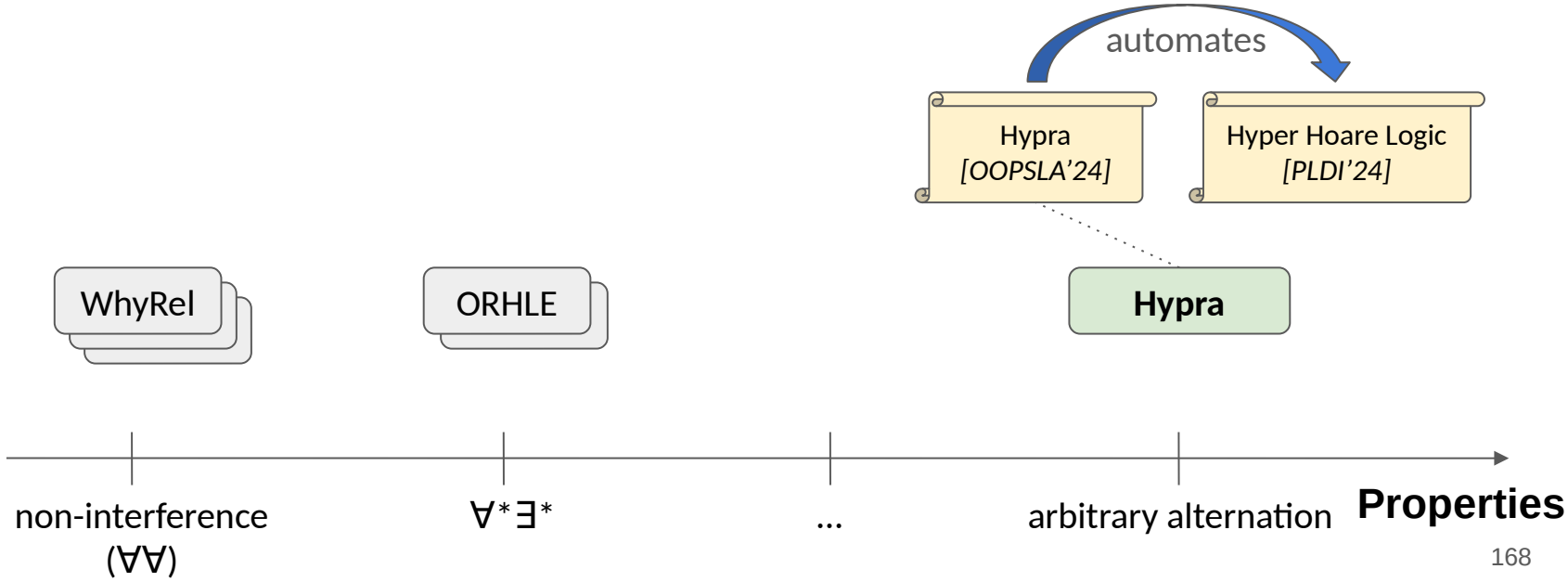
Automated Deductive Verifiers for Hyperproperties



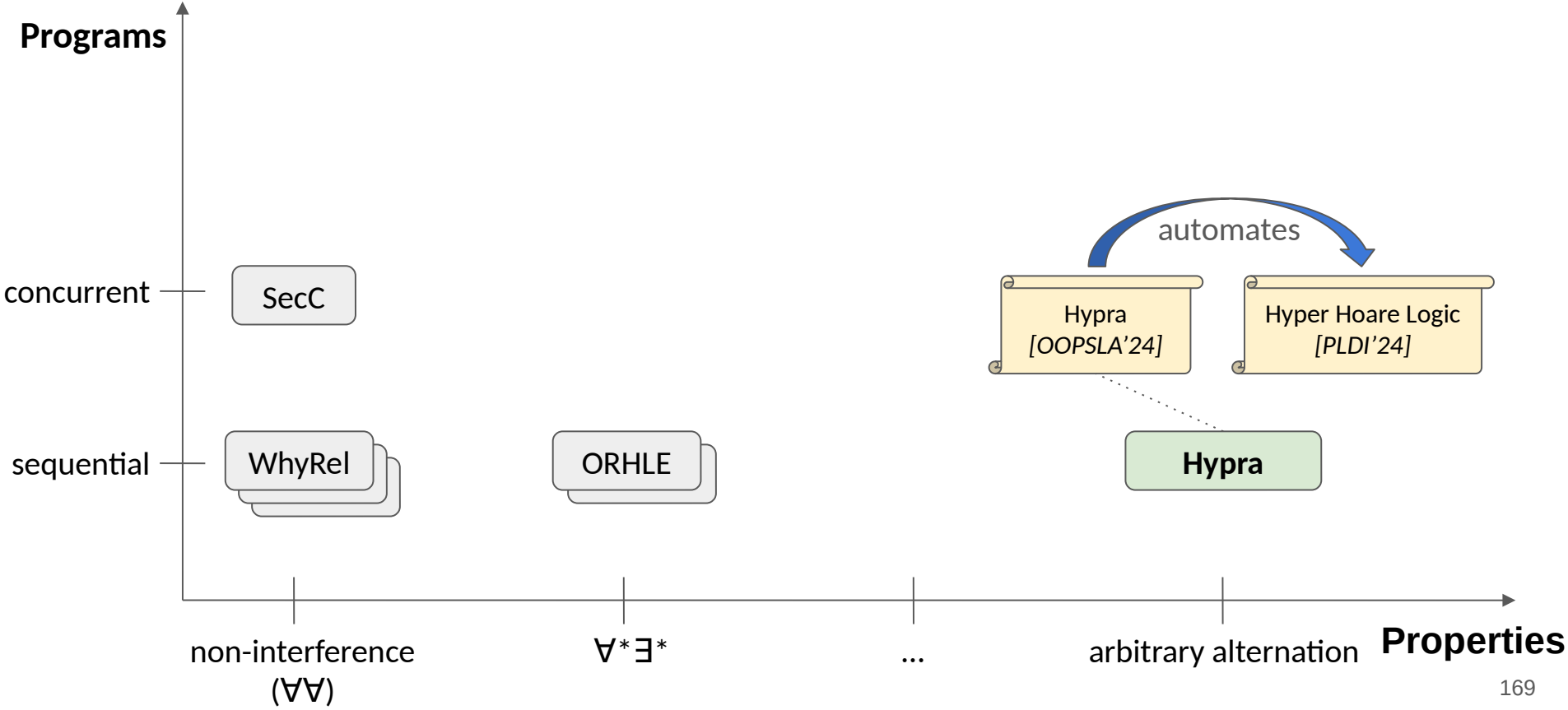
Automated Deductive Verifiers for Hyperproperties



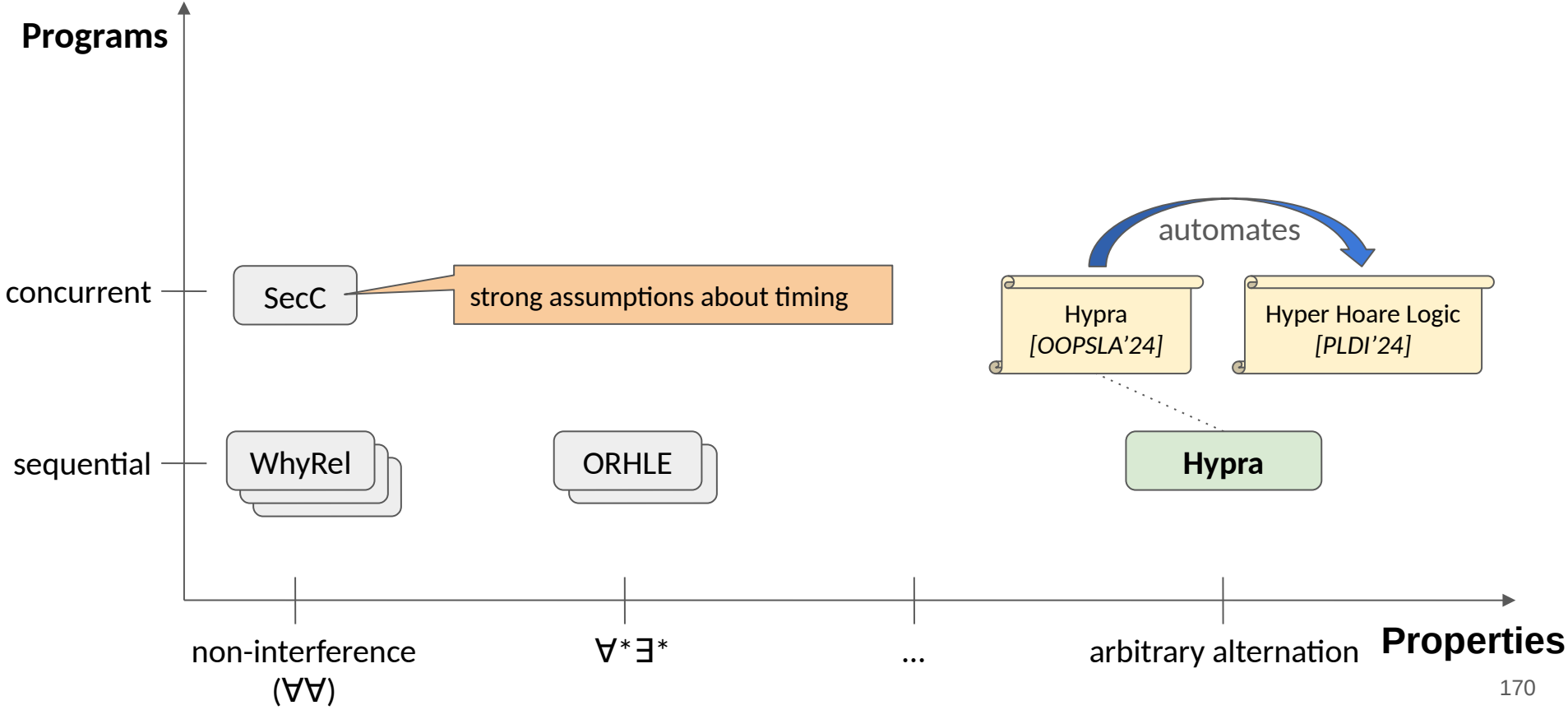
Automated Deductive Verifiers for Hyperproperties



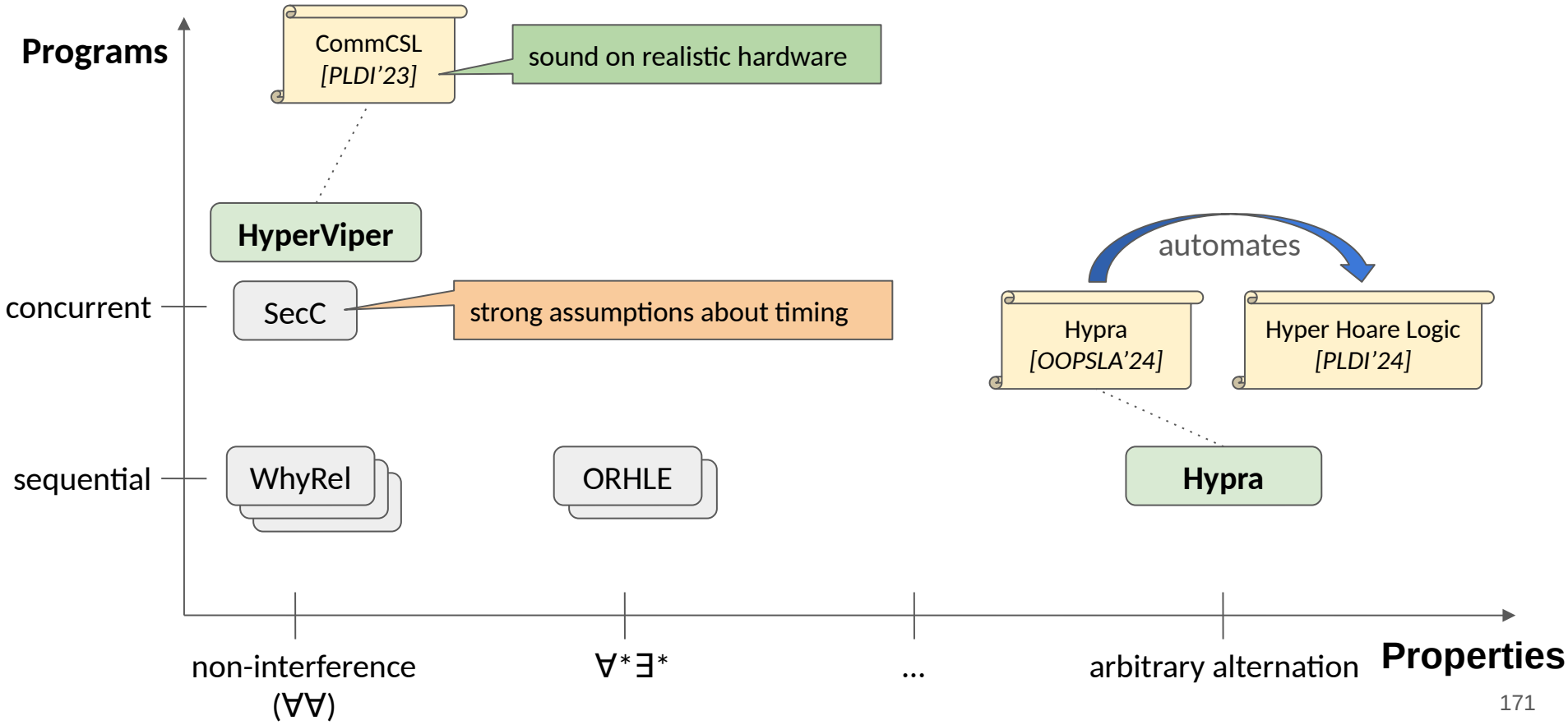
Automated Deductive Verifiers for Hyperproperties



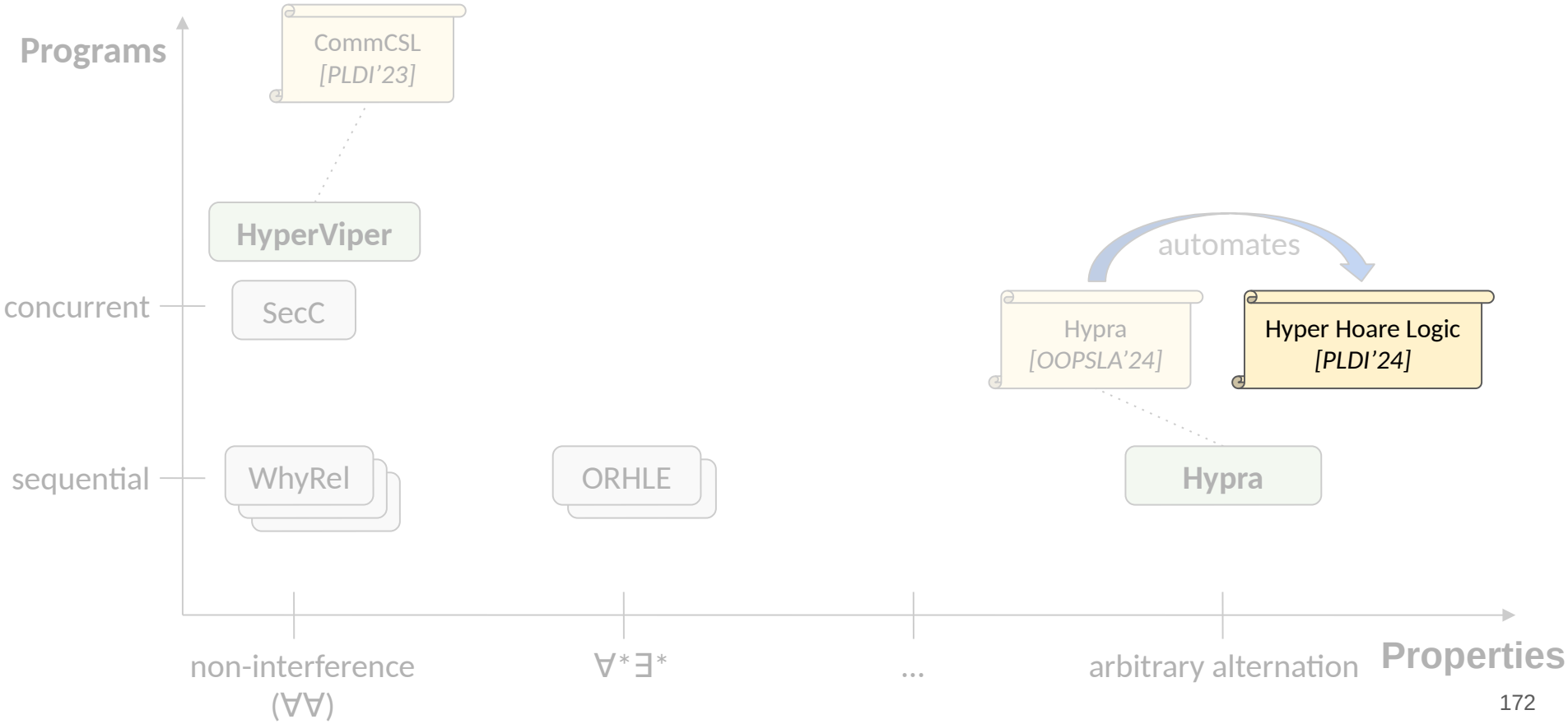
Automated Deductive Verifiers for Hyperproperties



Automated Deductive Verifiers for Hyperproperties

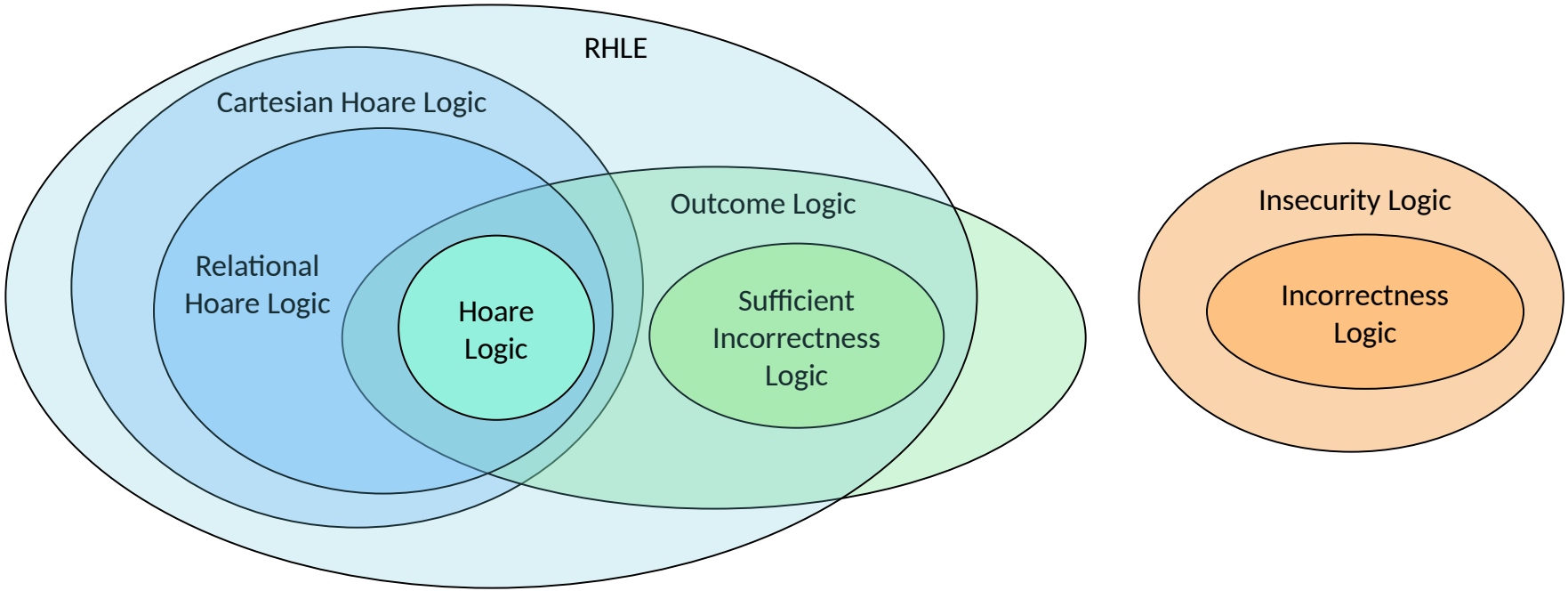


Automated Deductive Verifiers for Hyperproperties



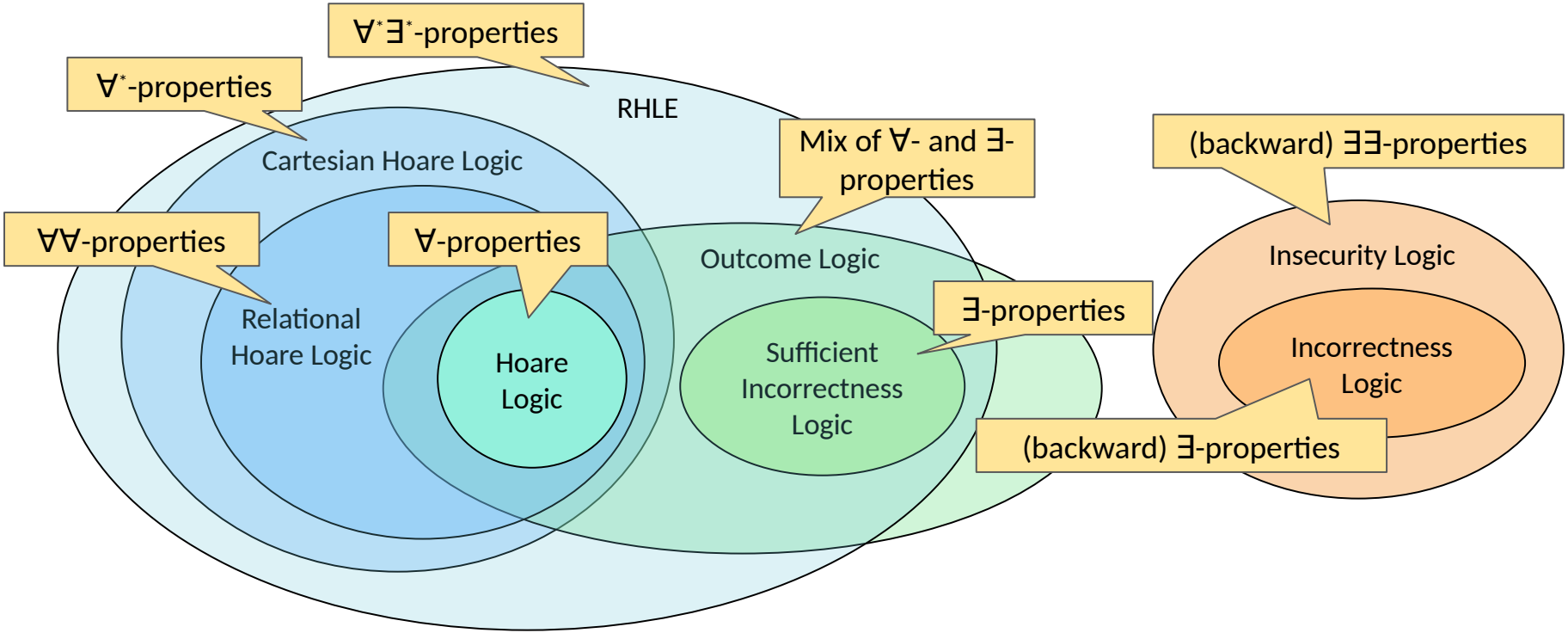
The Need for a Novel Foundation

Expressiveness of judgments restricted to a **single non-deterministic IMP** program (no heap or probabilities)



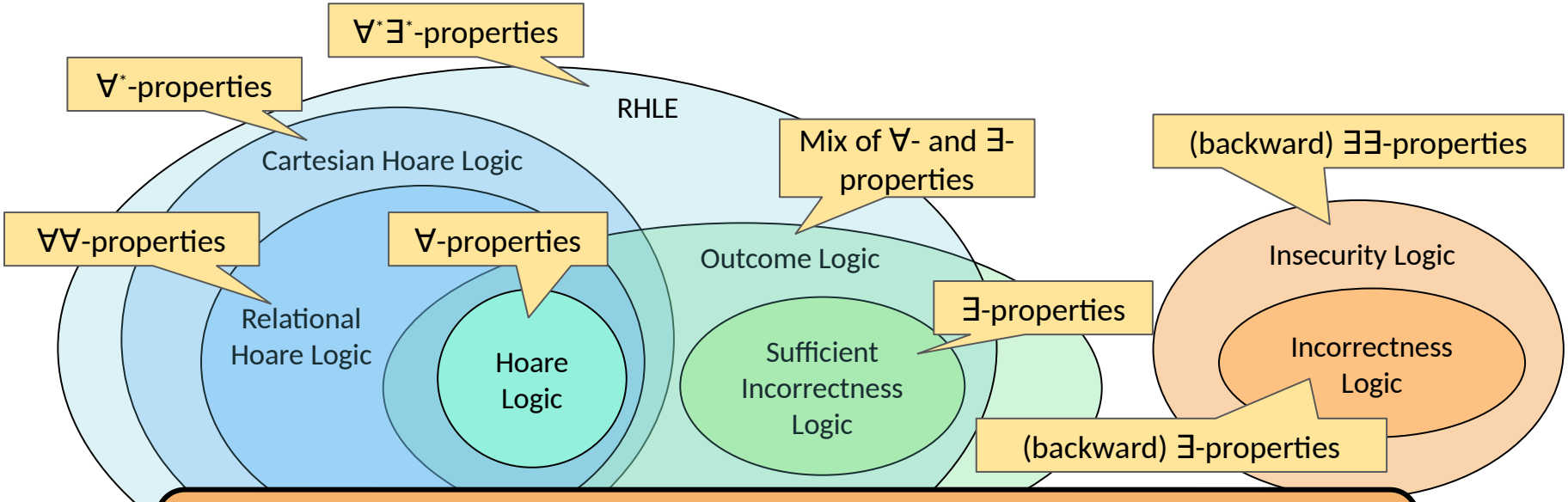
The Need for a Novel Foundation

Expressiveness of judgments restricted to a **single non-deterministic IMP** program (no heap or probabilities)



The Need for a Novel Foundation

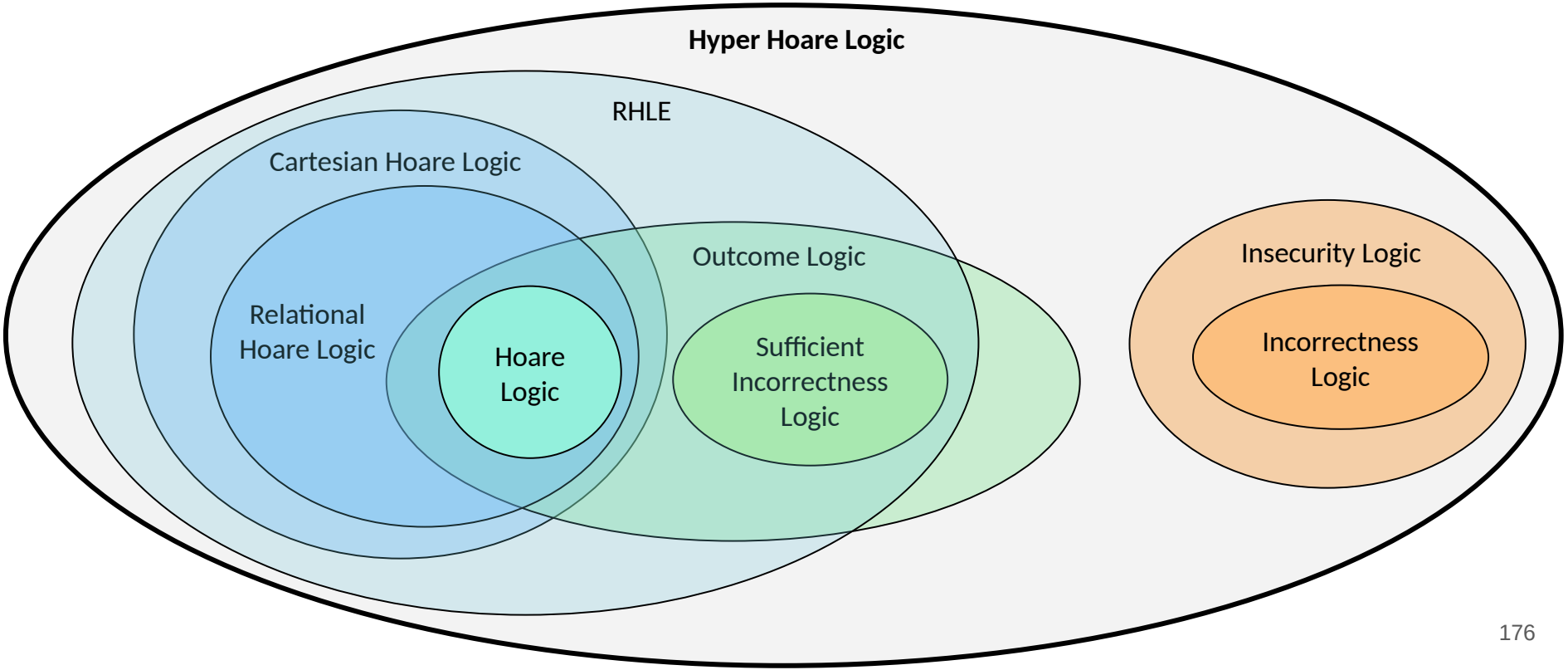
Expressiveness of judgments restricted to a **single non-deterministic IMP** program (no heap or probabilities)



- Limited expressiveness (e.g., $\exists^*\forall^*$-properties)
- Limited compositionality (e.g., $\forall\forall$- and $\forall\exists$-properties)

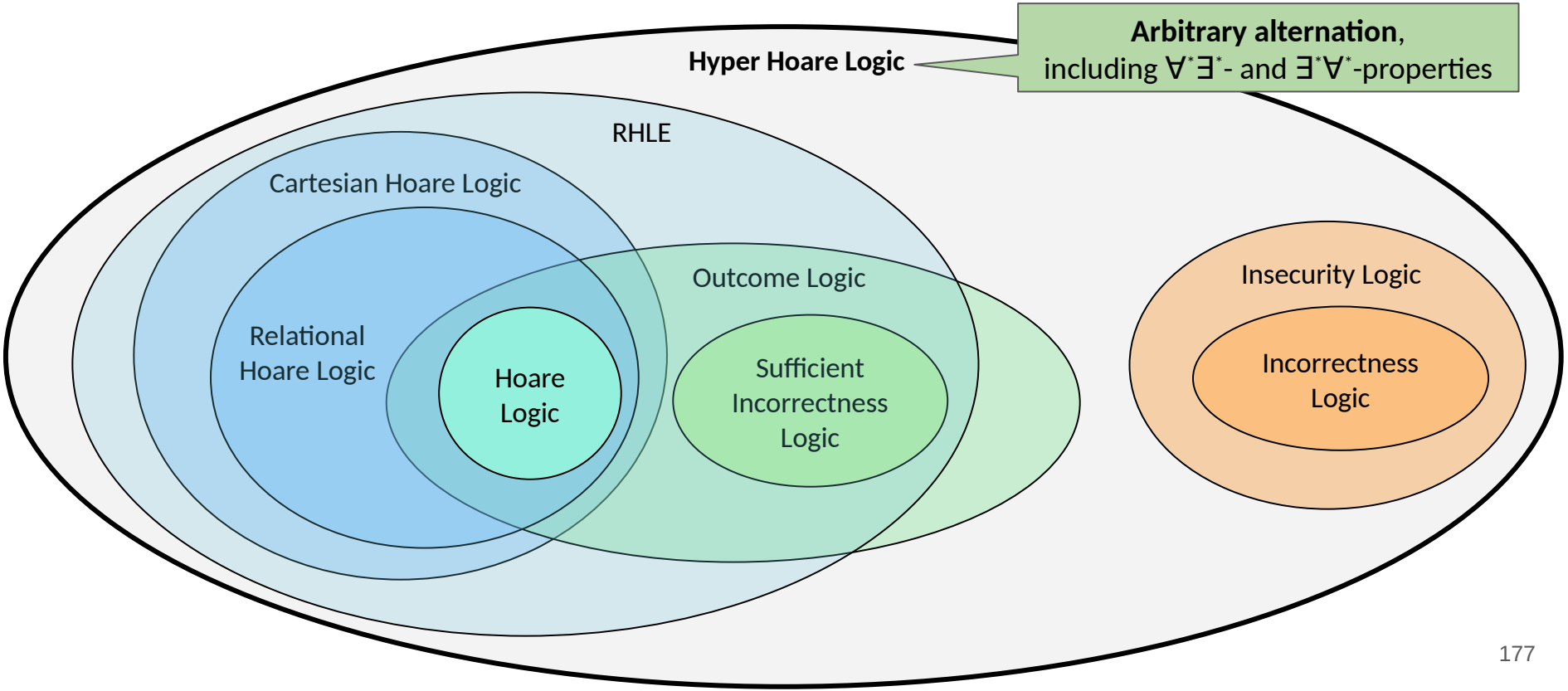
The Need for a Novel Foundation

Expressiveness of judgments restricted to a **single non-deterministic IMP** program (no heap or probabilities)

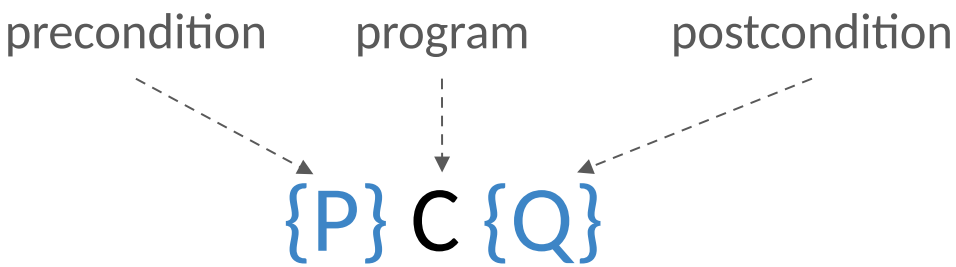


The Need for a Novel Foundation

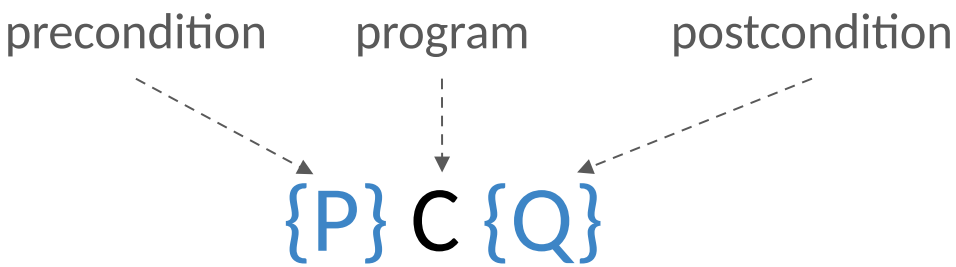
Expressiveness of judgments restricted to a **single non-deterministic IMP** program (no heap or probabilities)



Hyper Hoare Logic – Key Insights



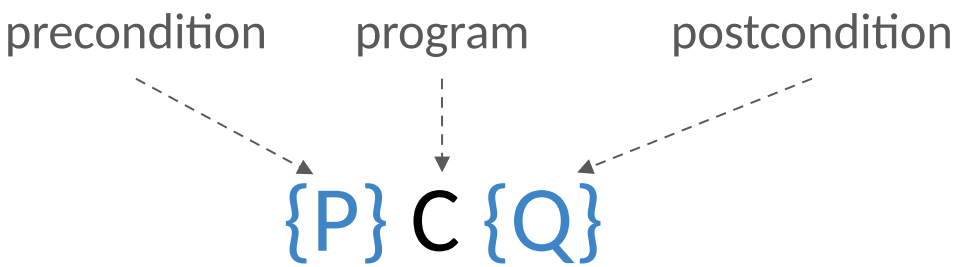
Hyper Hoare Logic – Key Insights



Pre-existing Hoare-like logics

Pre- and postconditions over **fixed** number of states

Hyper Hoare Logic – Key Insights

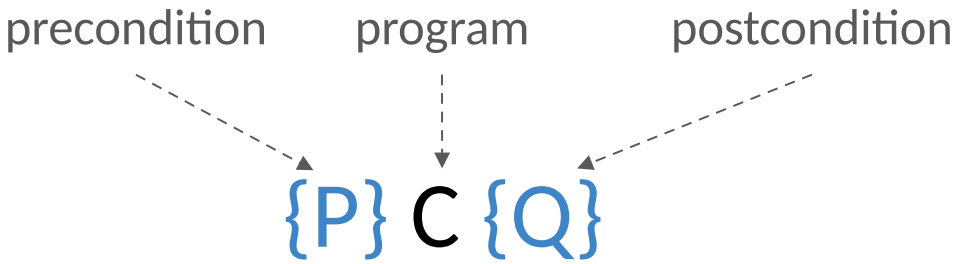


Pre-existing Hoare-like logics

Pre- and postconditions over **fixed** number of states

Quantification over executions **fixed**

Hyper Hoare Logic – Key Insights



Pre-existing Hoare-like logics

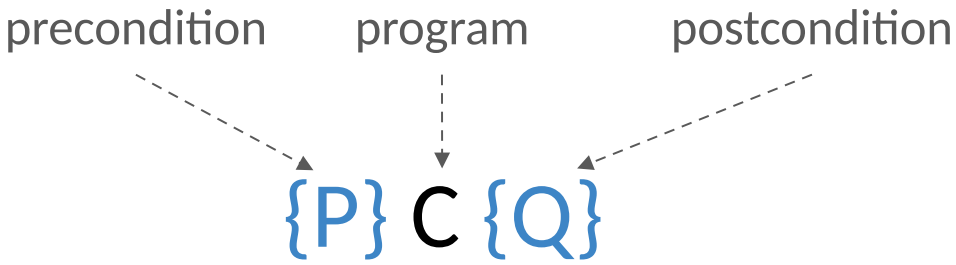
Pre- and postconditions over **fixed** number of states

Quantification over executions **fixed**

Hyper Hoare Logic

Pre- and postconditions over **sets** of reachable states

Hyper Hoare Logic – Key Insights



Pre-existing Hoare-like logics

Pre- and postconditions over **fixed** number of states

Quantification over executions **fixed**

Hyper Hoare Logic

Pre- and postconditions over **sets** of reachable states

Quantification over executions **explicit** in **assertions**

Hyper Hoare Logic – Non-Interference

$\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p}) \}$ C $\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res}) \}$

Hyper Hoare Logic – Non-Interference

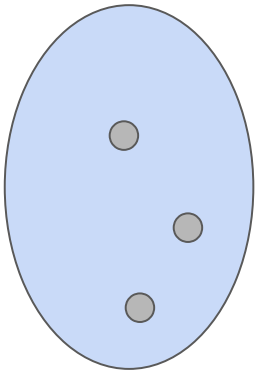
$\{\ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p})\ \}\ \ C\ \ \{\ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res})\ \}$

initial states

Hyper Hoare Logic – Non-Interference

$\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p}) \}$ C $\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res}) \}$

initial states

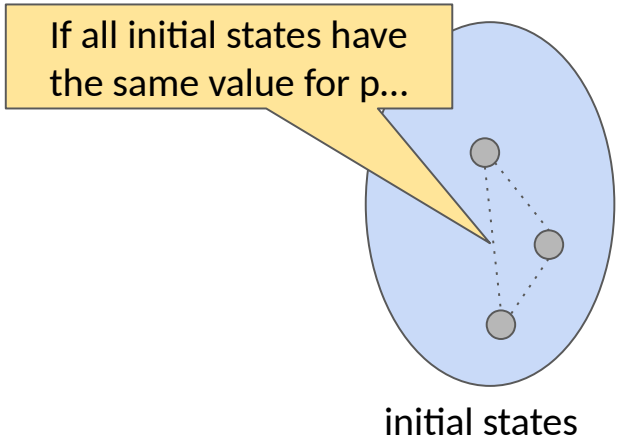


initial states

Hyper Hoare Logic – Non-Interference

$\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p}) \}$ C $\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res}) \}$

initial states

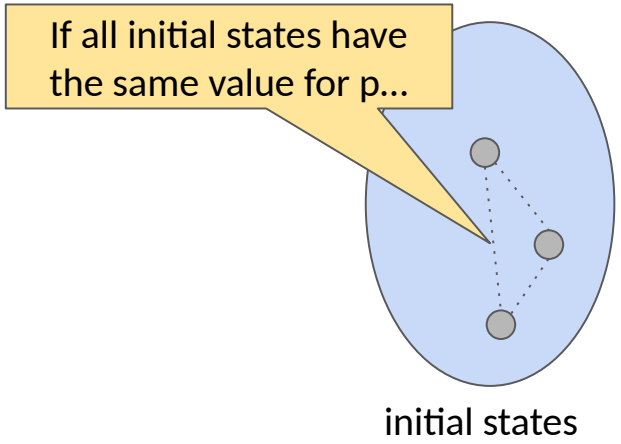


Hyper Hoare Logic – Non-Interference

$\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p}) \}$ C $\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res}) \}$

initial states

reachable states

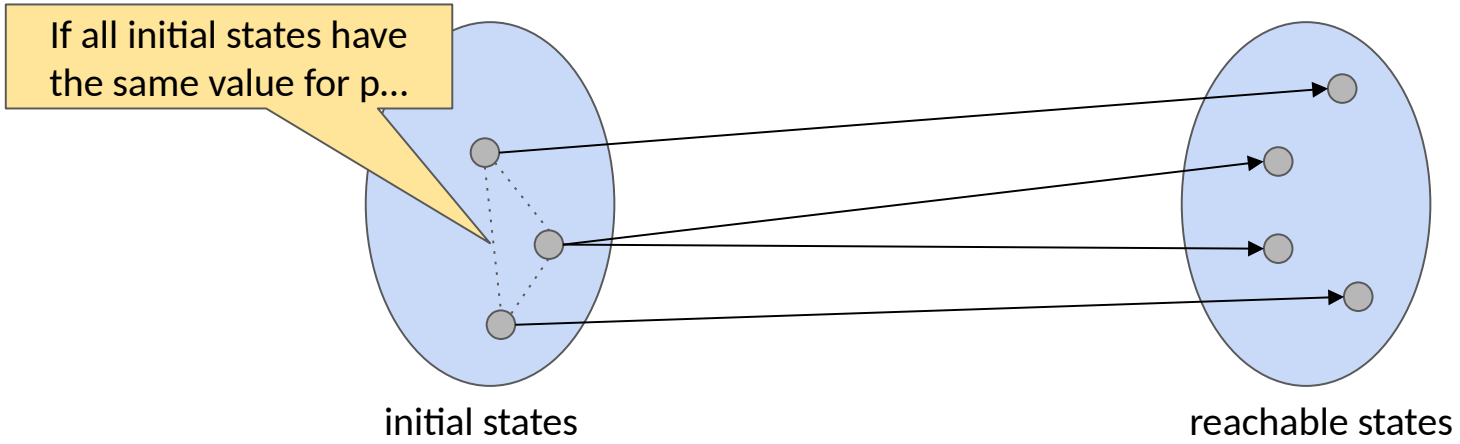


Hyper Hoare Logic – Non-Interference

$$\{\ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p})\ \}\ \ \mathrm{C}\ \ \{\ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res})\ \}$$

initial states

reachable states

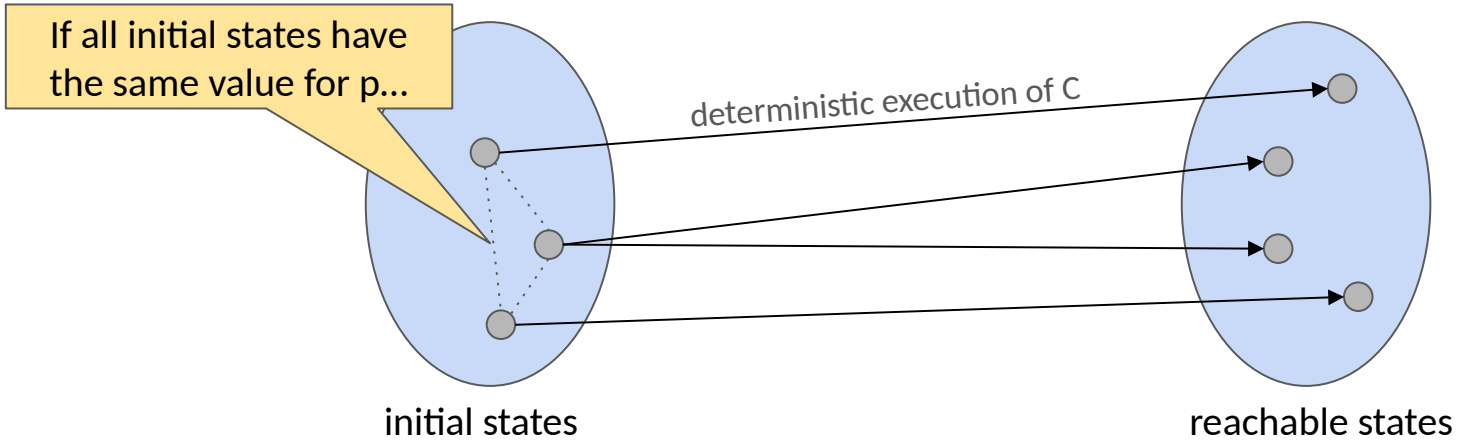


Hyper Hoare Logic – Non-Interference

$$\{\ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p})\ \}\ \ C\ \ \{\ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res})\ \}$$

initial states

reachable states

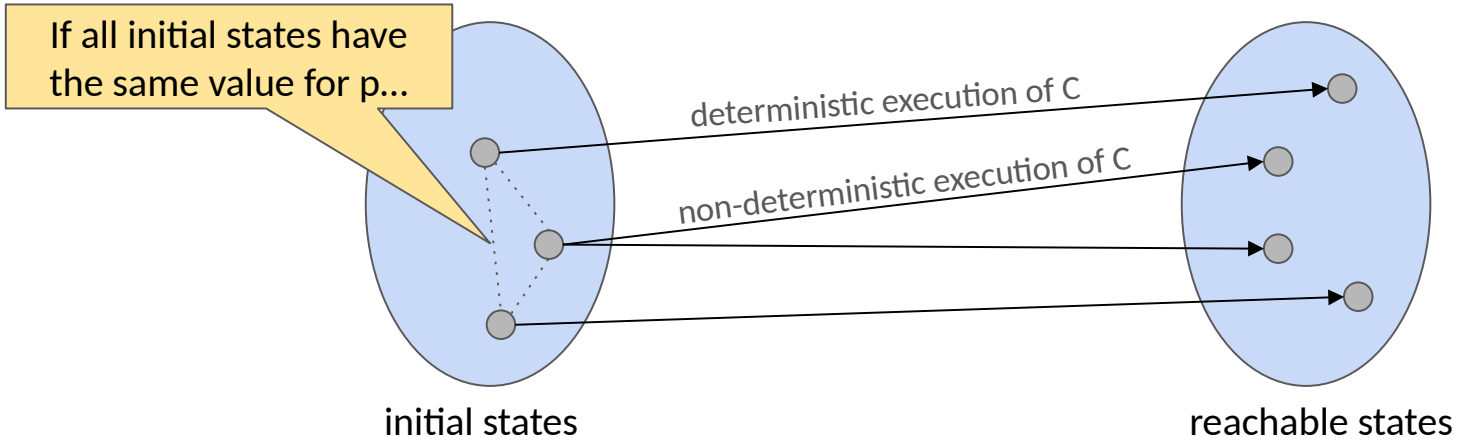


Hyper Hoare Logic – Non-Interference

$\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p}) \}$ C $\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res}) \}$

initial states

reachable states

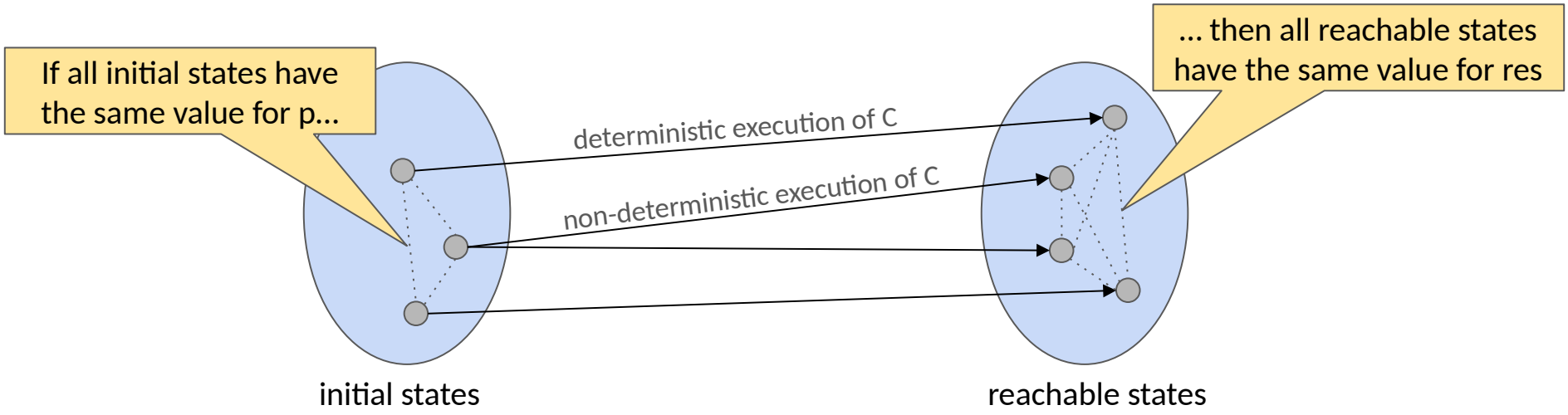


Hyper Hoare Logic – Non-Interference

$\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p}) \}$ C $\{ \forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{res}) = \sigma_2(\mathrm{res}) \}$

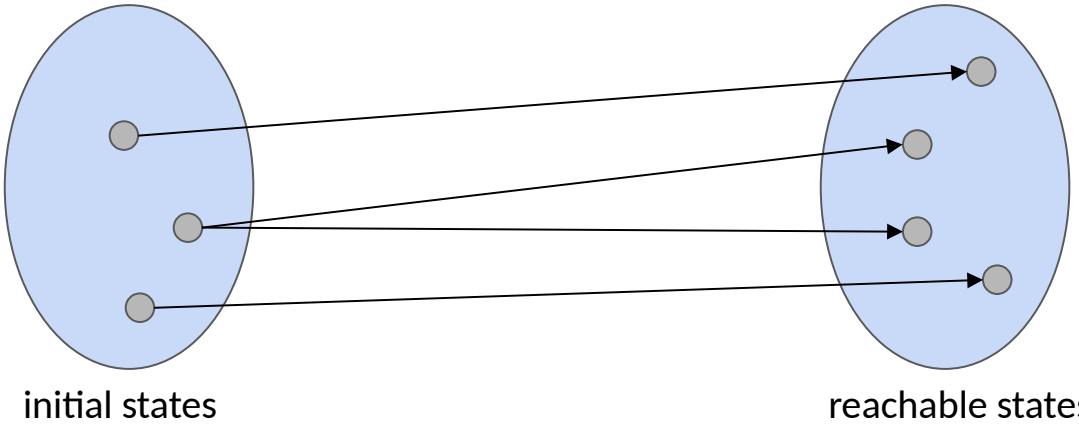
initial states

reachable states



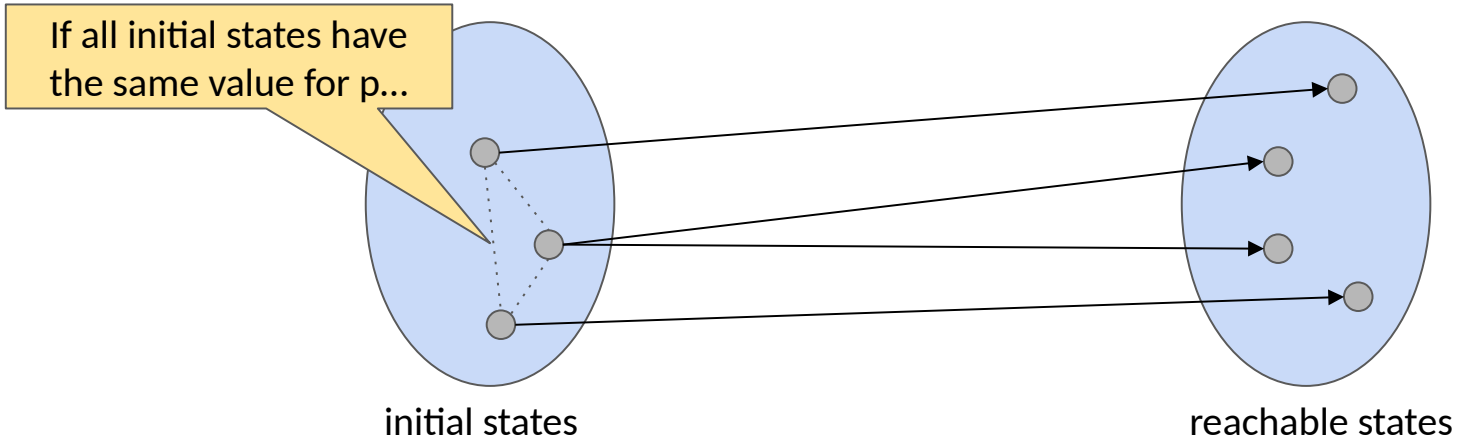
Hyper Hoare Logic – Generalized Non-Interference

$\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p})\}$ C $\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \exists\langle\sigma\rangle.\ \sigma(\mathrm{s}) = \sigma_1(\mathrm{s}) \wedge \sigma(\mathrm{res}) = \sigma_2(\mathrm{res})\}$



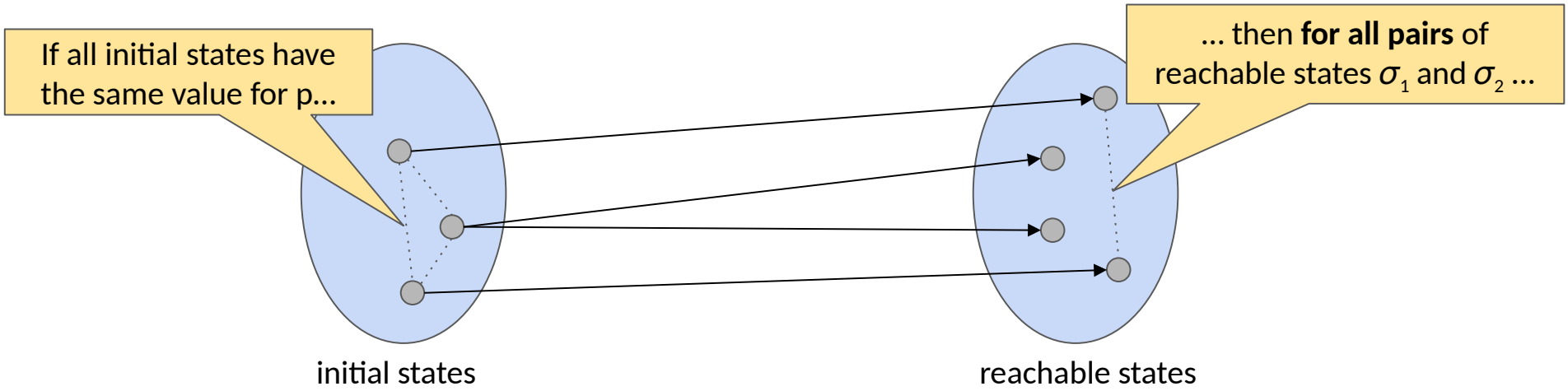
Hyper Hoare Logic – Generalized Non-Interference

$\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p})\}$ C $\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \exists\langle\sigma\rangle.\ \sigma(\mathrm{s}) = \sigma_1(\mathrm{s}) \wedge \sigma(\mathrm{res}) = \sigma_2(\mathrm{res})\}$



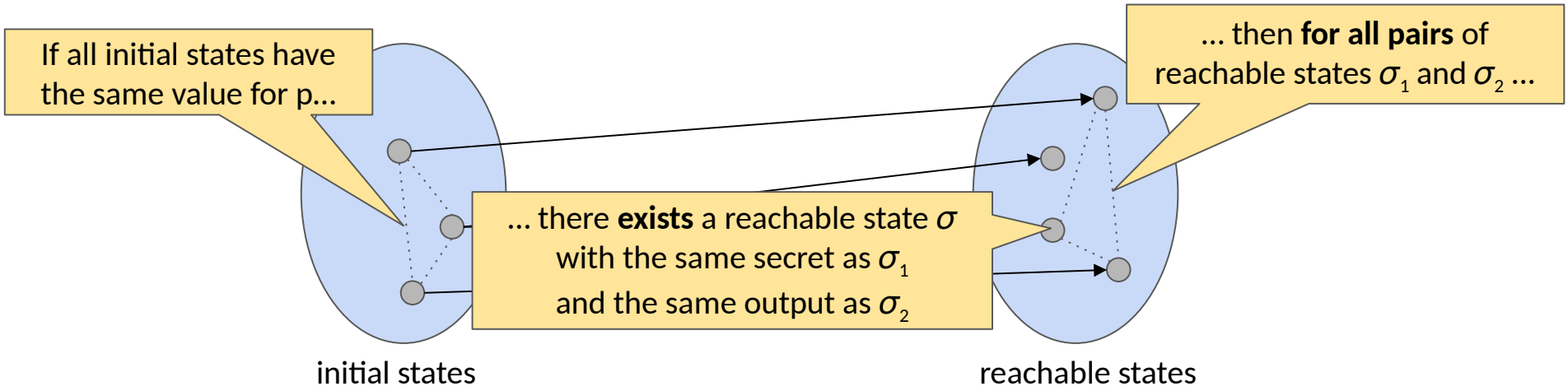
Hyper Hoare Logic – Generalized Non-Interference

$\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p})\}$ C $\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \exists\langle\sigma\rangle.\ \sigma(\mathrm{s}) = \sigma_1(\mathrm{s}) \wedge \sigma(\mathrm{res}) = \sigma_2(\mathrm{res})\}$



Hyper Hoare Logic – Generalized Non-Interference

$\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \sigma_1(\mathrm{p}) = \sigma_2(\mathrm{p})\}$ C $\{\forall\langle\sigma_1\rangle.\ \forall\langle\sigma_2\rangle.\ \exists\langle\sigma\rangle.\ \sigma(\mathrm{s}) = \sigma_1(\mathrm{s}) \wedge \sigma(\mathrm{res}) = \sigma_2(\mathrm{res})\}$



Hyper Hoare Logic – Rules

Minimal set of core rules
Complete

Hyper Hoare Logic – Rules

Minimal set of core rules
Complete

Syntactic rules
Easy to use and automate

Hyper Hoare Logic – Rules

Minimal set of core rules
Complete

Compositionality rules
To compose different types of hyperproperties

Syntactic rules
Easy to use and automate

Hyper Hoare Logic – Rules

Minimal set of core rules
Complete

Compositionality rules
To compose different types of hyperproperties

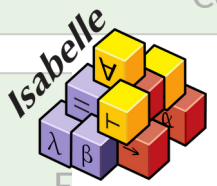
Syntactic rules
Easy to use and automate

Loop rules
Capturing important proof principles

Hyper Hoare Logic – Rules

Minimal set of core rules
Complete

Compositionality rules
To compose different types of hyperproperties



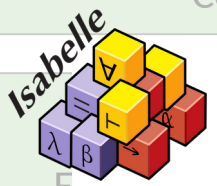
Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties

THIBAULT DARDINIER, ETH Zurich, Switzerland
PETER MÜLLER, ETH Zurich, Switzerland

Hyper Hoare Logic – Rules

Hyper Hoare Logic – Rules

First automated verifier that can establish all previously-mentioned hyperproperties!



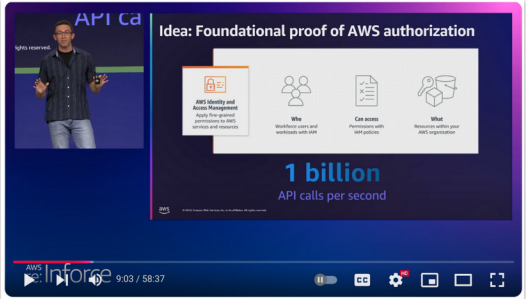
Hypra: A Deductive Program Verifier for Hyper Hoare Logic

THIBAULT DARDINIER*, ETH Zurich, Switzerland
ANQI LI*, ETH Zurich, Switzerland
PETER MÜLLER, ETH Zurich, Switzerland

Hyper Hoare Logic: (Dis-)Proving Program Hyperproperties

THIBAULT DARDINIER, ETH Zurich, Switzerland
PETER MÜLLER, ETH Zurich, Switzerland

Summary



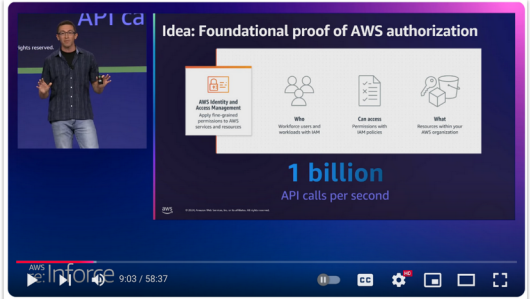
AWS re:Inforce 2024 - Proving the correctness of AWS authorization (IAM401)

1 billion times/second

~3,000 lines of code

~3x performance

Summary

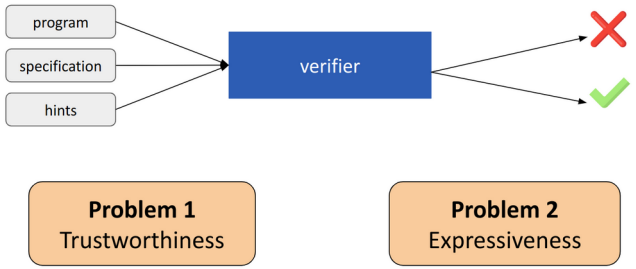


AWS re:Inforce 2024 - Proving the correctness of AWS authorization (IAM401)

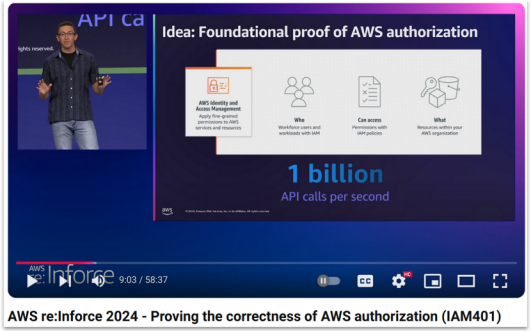
1 billion times/second

~3,000 lines of code

~3x performance



Summary

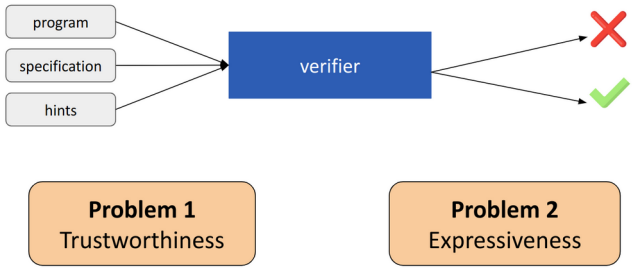


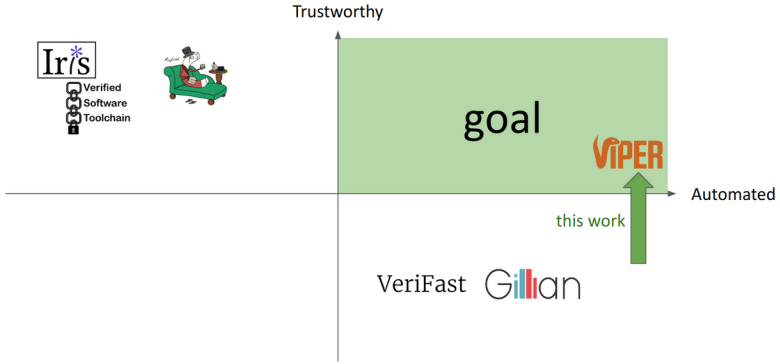
AWS re:Inforce 2024 - Proving the correctness of AWS authorization (IAM401)

1 billion times/second

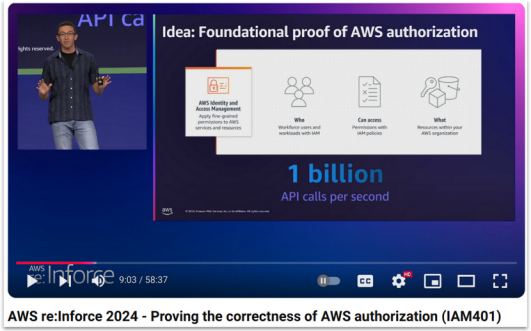
~3,000 lines of code

~3x performance





Summary

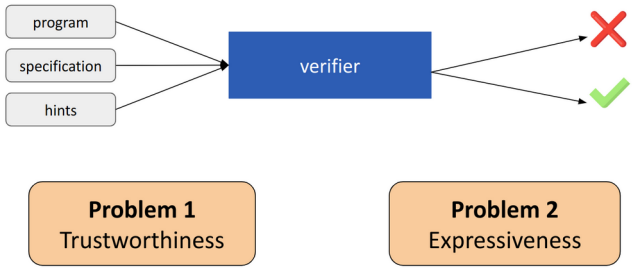


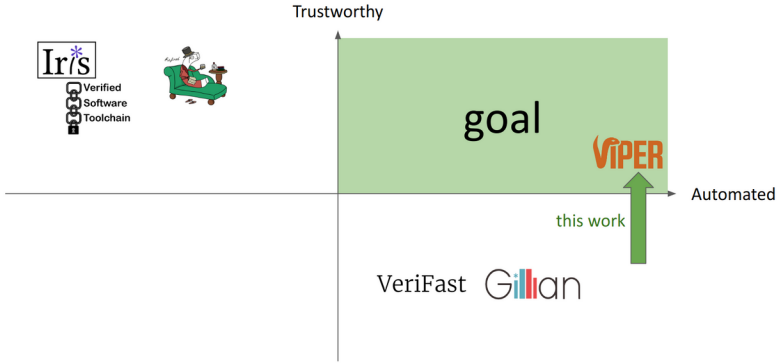
AWS re:Inforce 2024 - Proving the correctness of AWS authorization (IAM401)

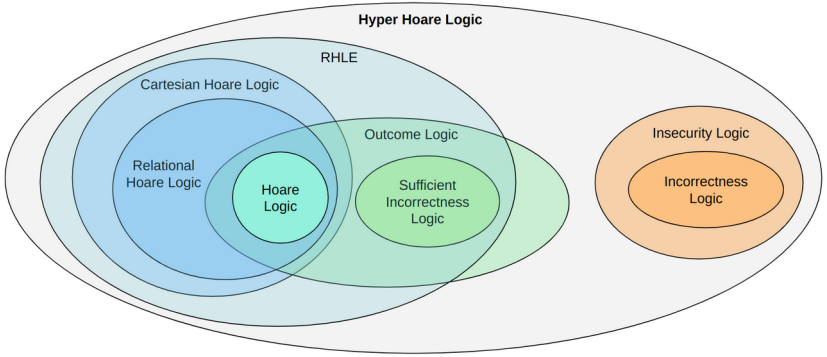
1 billion times/second

~3,000 lines of code

~3x performance







Past Research

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Provably sound and automated verifier based on *separation logic*

Problem 2
Expressiveness

Automated verifier for arbitrary *hyperproperties*

Future Research Directions

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Problem 2
Expressiveness

Future Research Directions

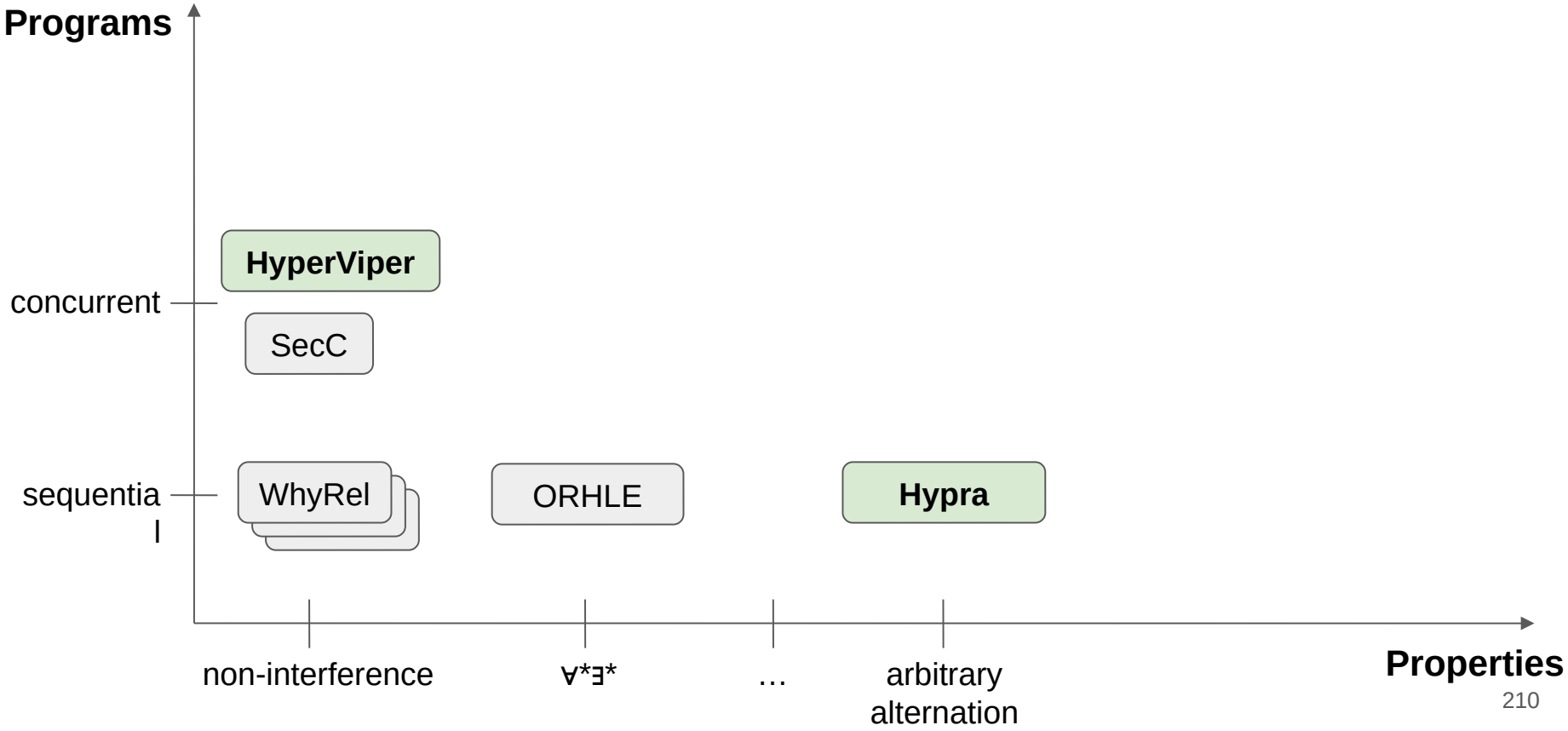
Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

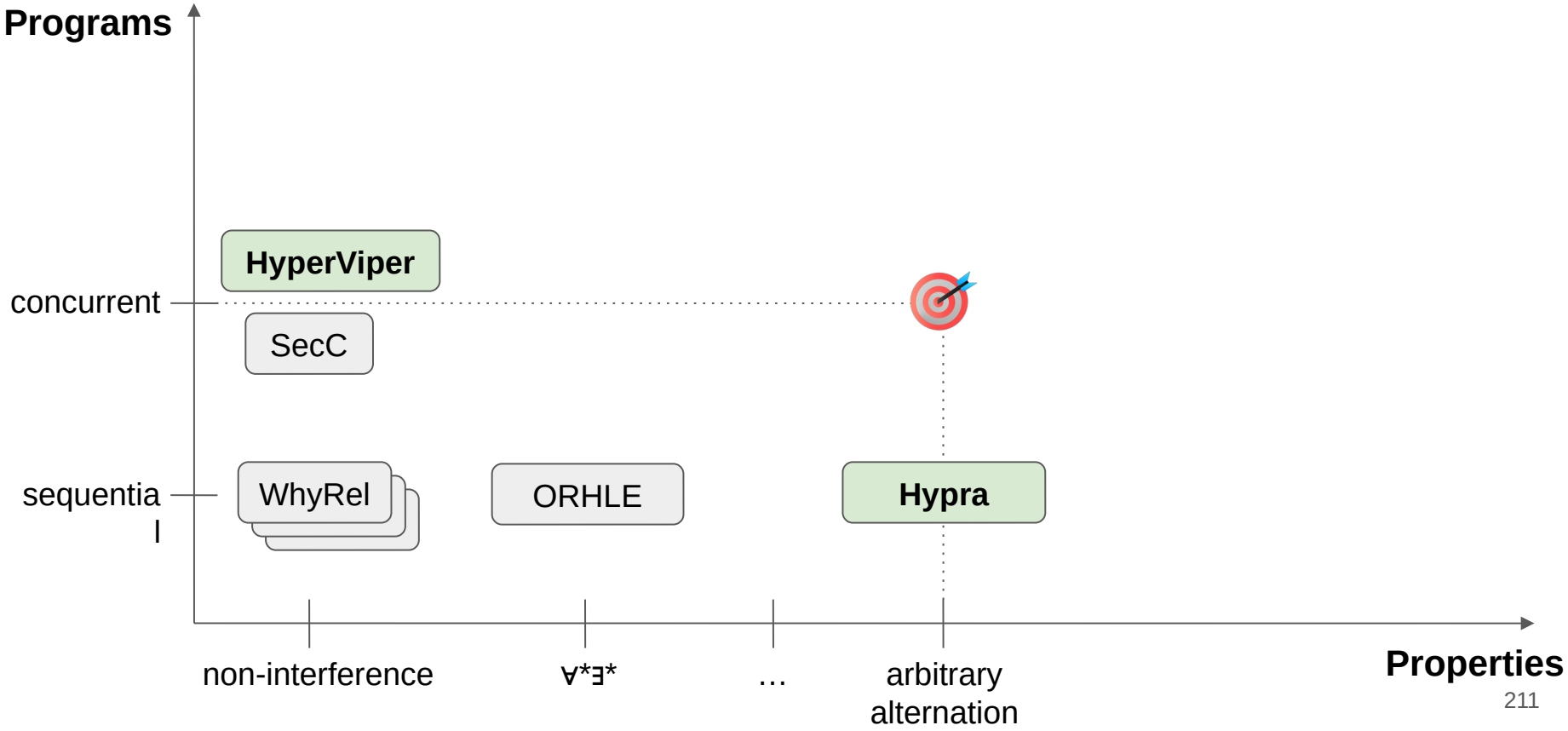
Problem 2
Expressiveness

General-purpose automated verifier for **security** and **privacy** properties

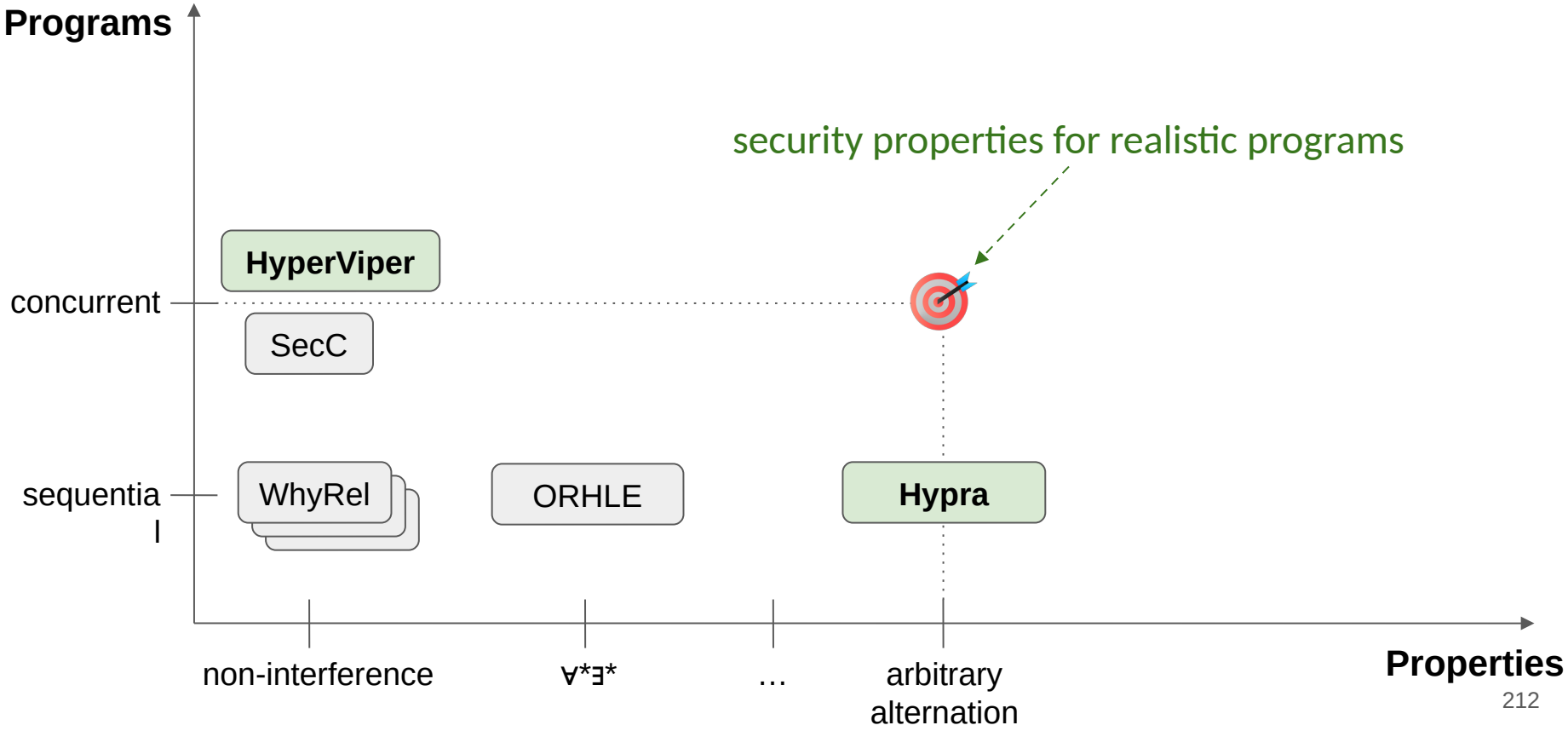
Automated Verifiers for Security and Privacy Properties



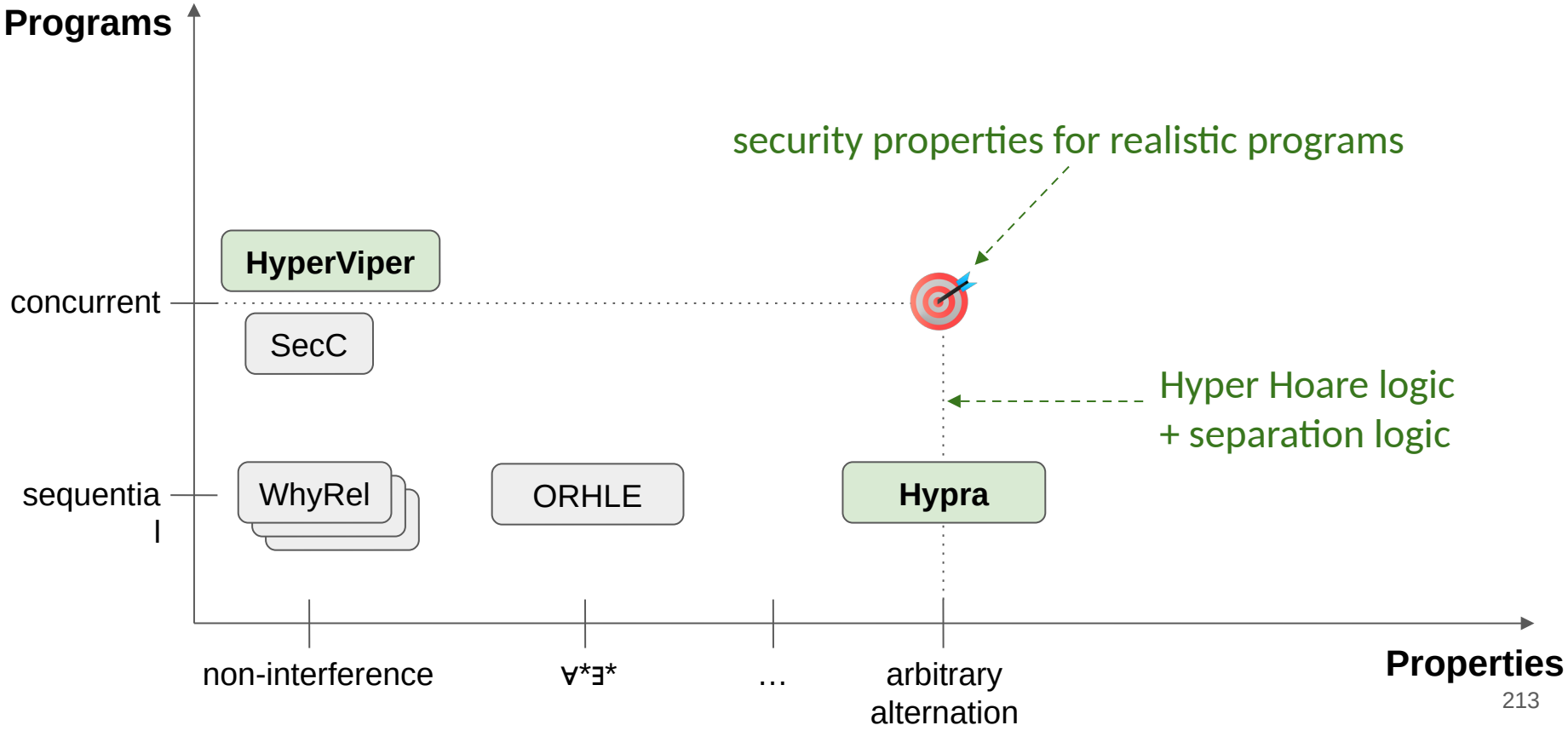
Automated Verifiers for Security and Privacy Properties



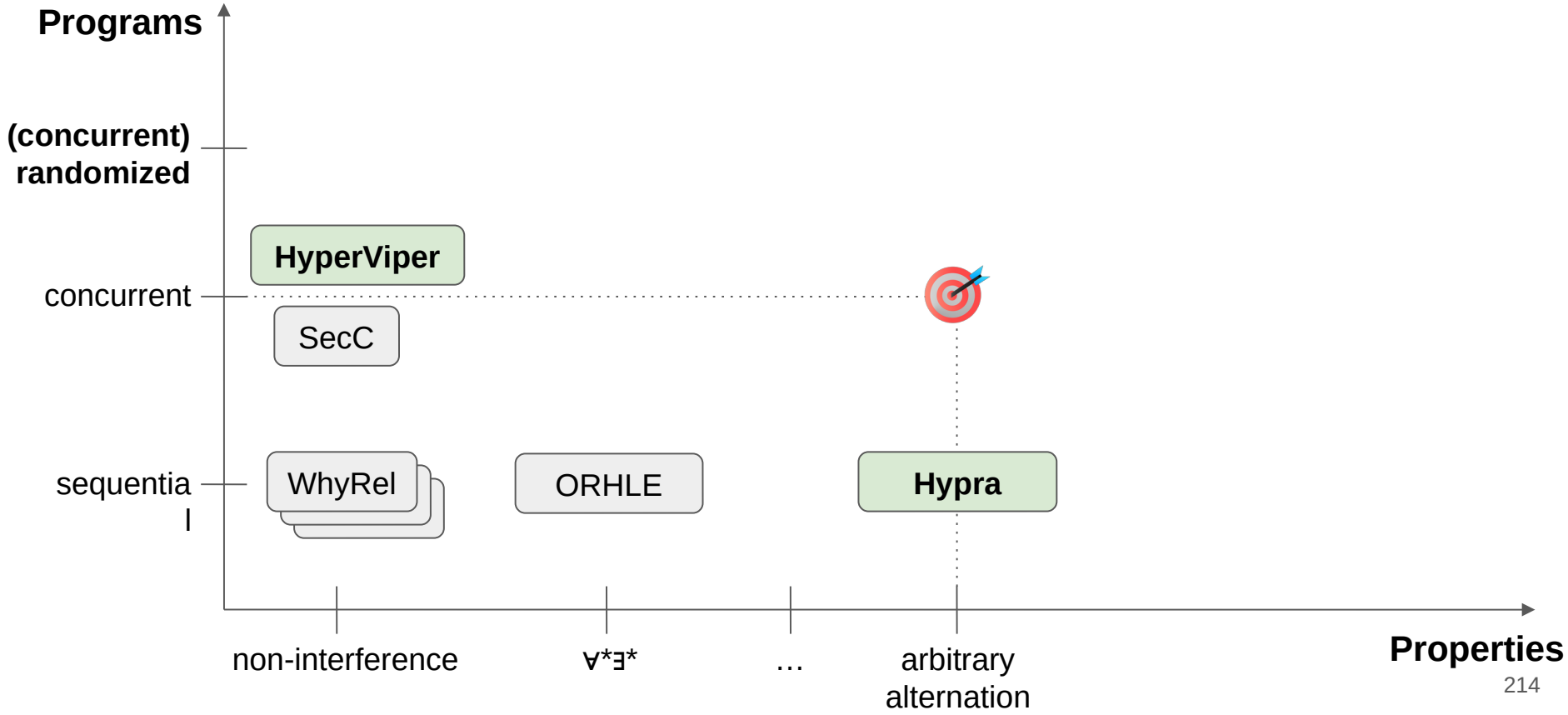
Automated Verifiers for Security and Privacy Properties



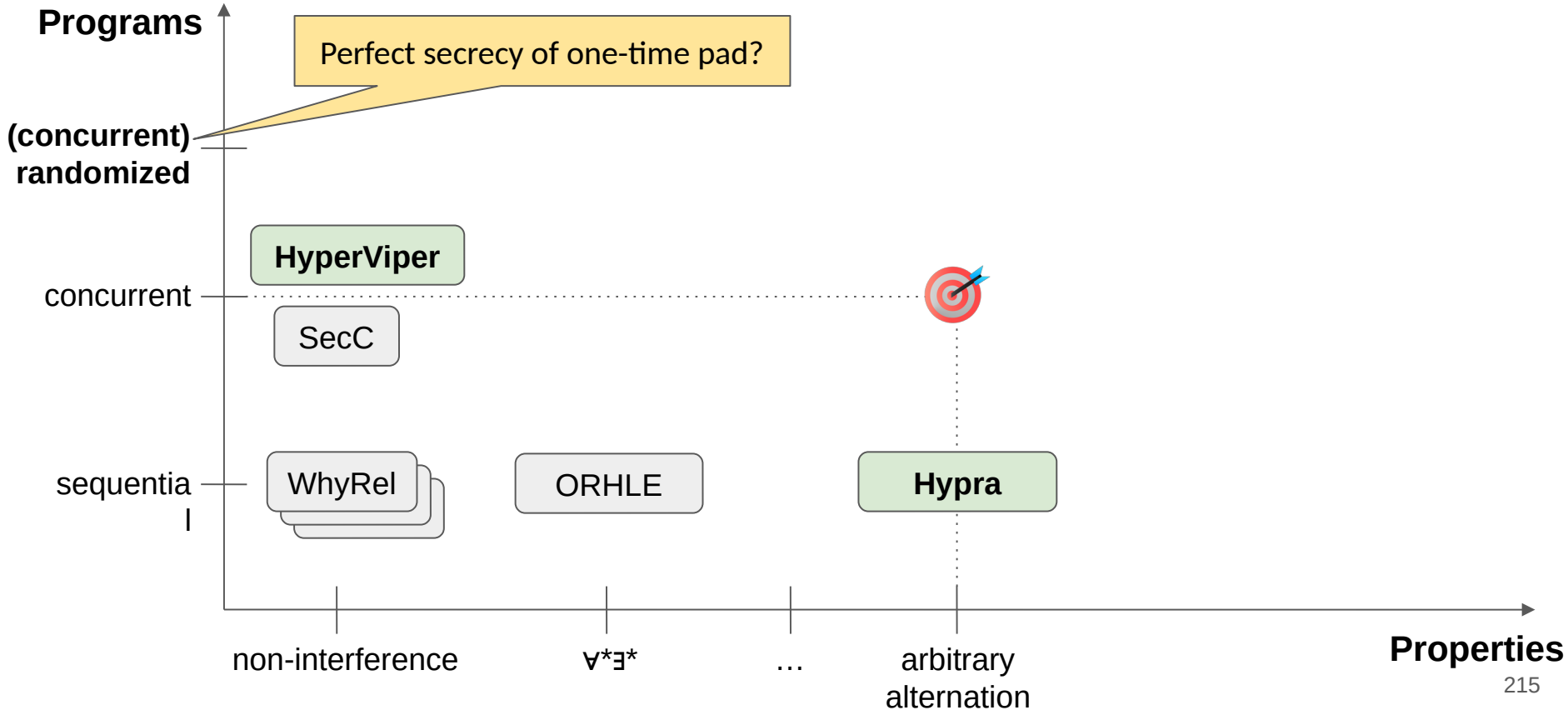
Automated Verifiers for Security and Privacy Properties



Automated Verifiers for Security and Privacy Properties

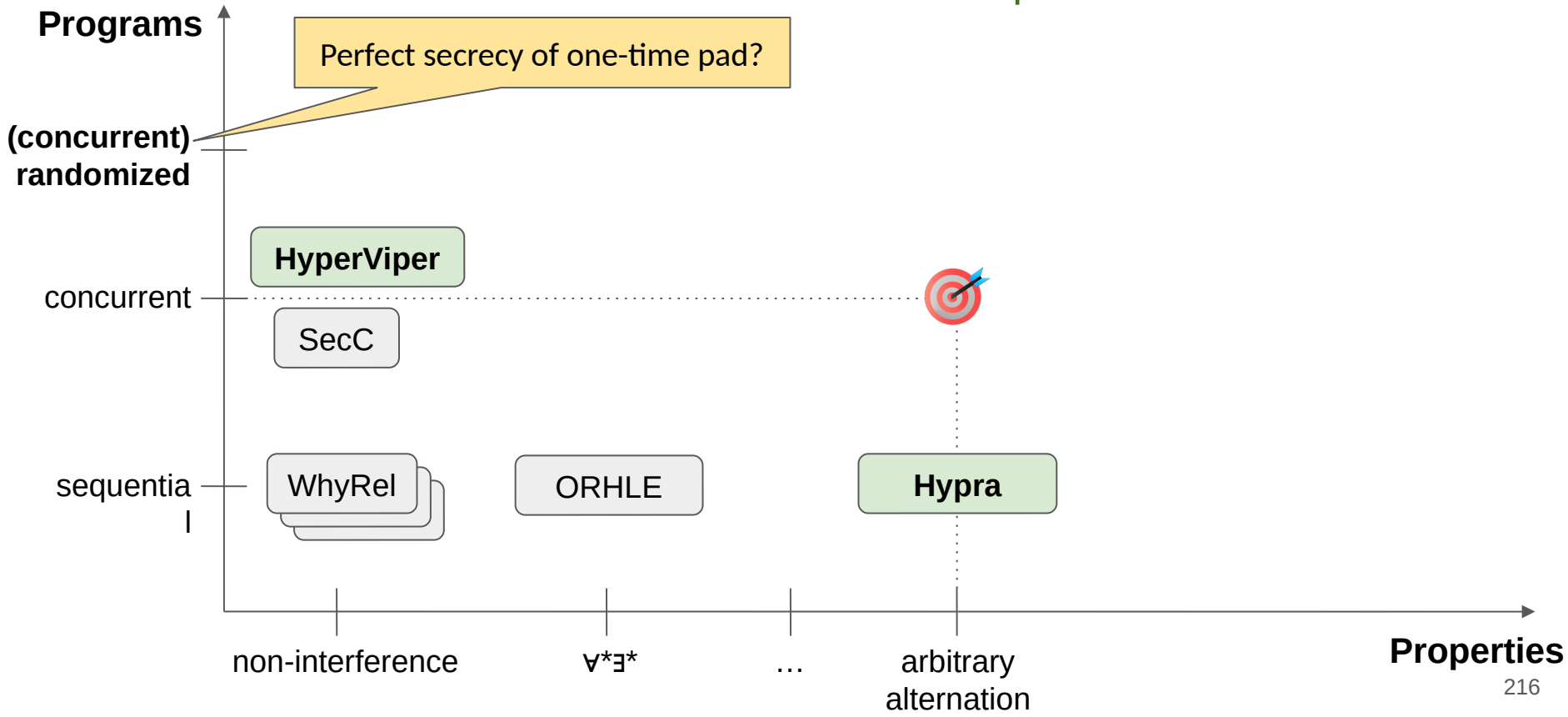


Automated Verifiers for Security and Privacy Properties



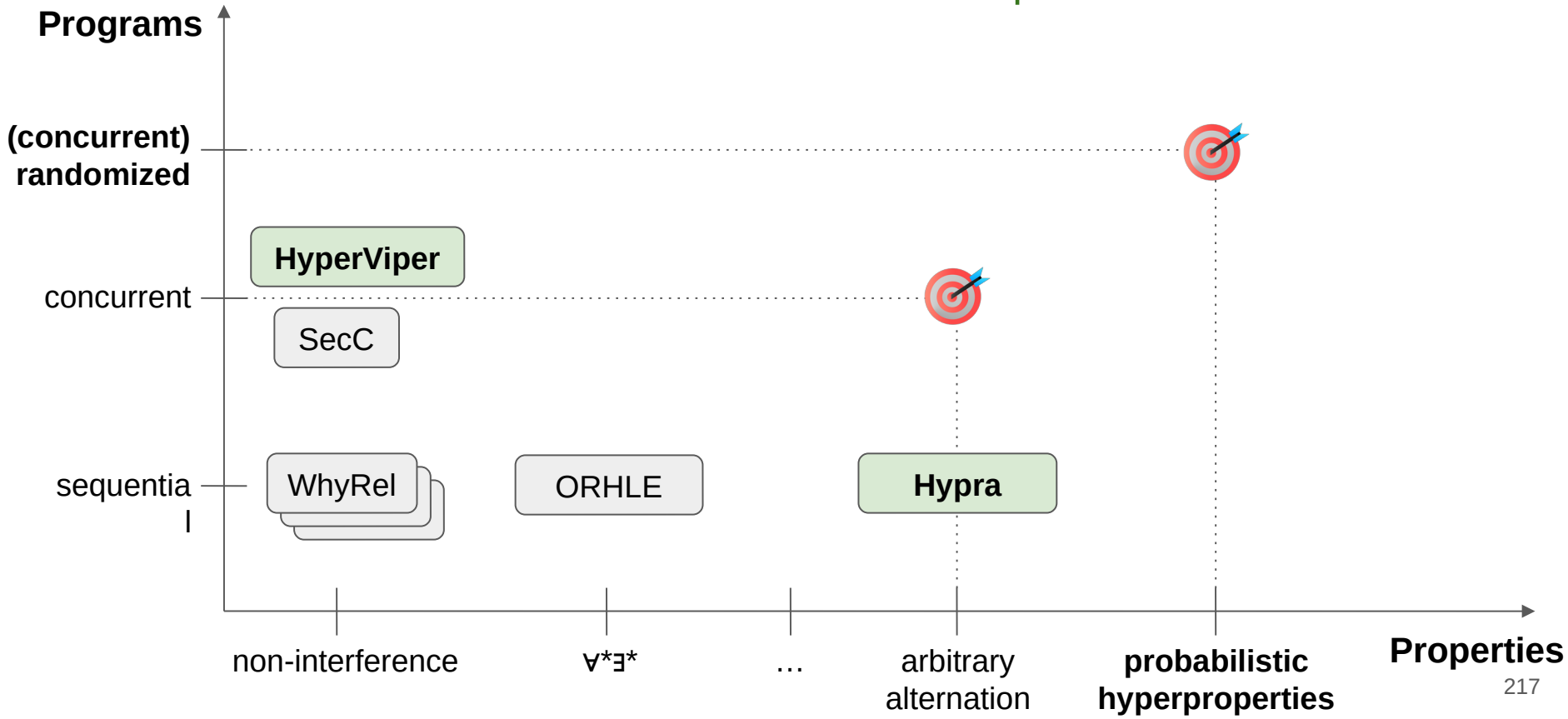
Automated Verifiers for Security and Privacy Properties

+ probabilistic non-interference

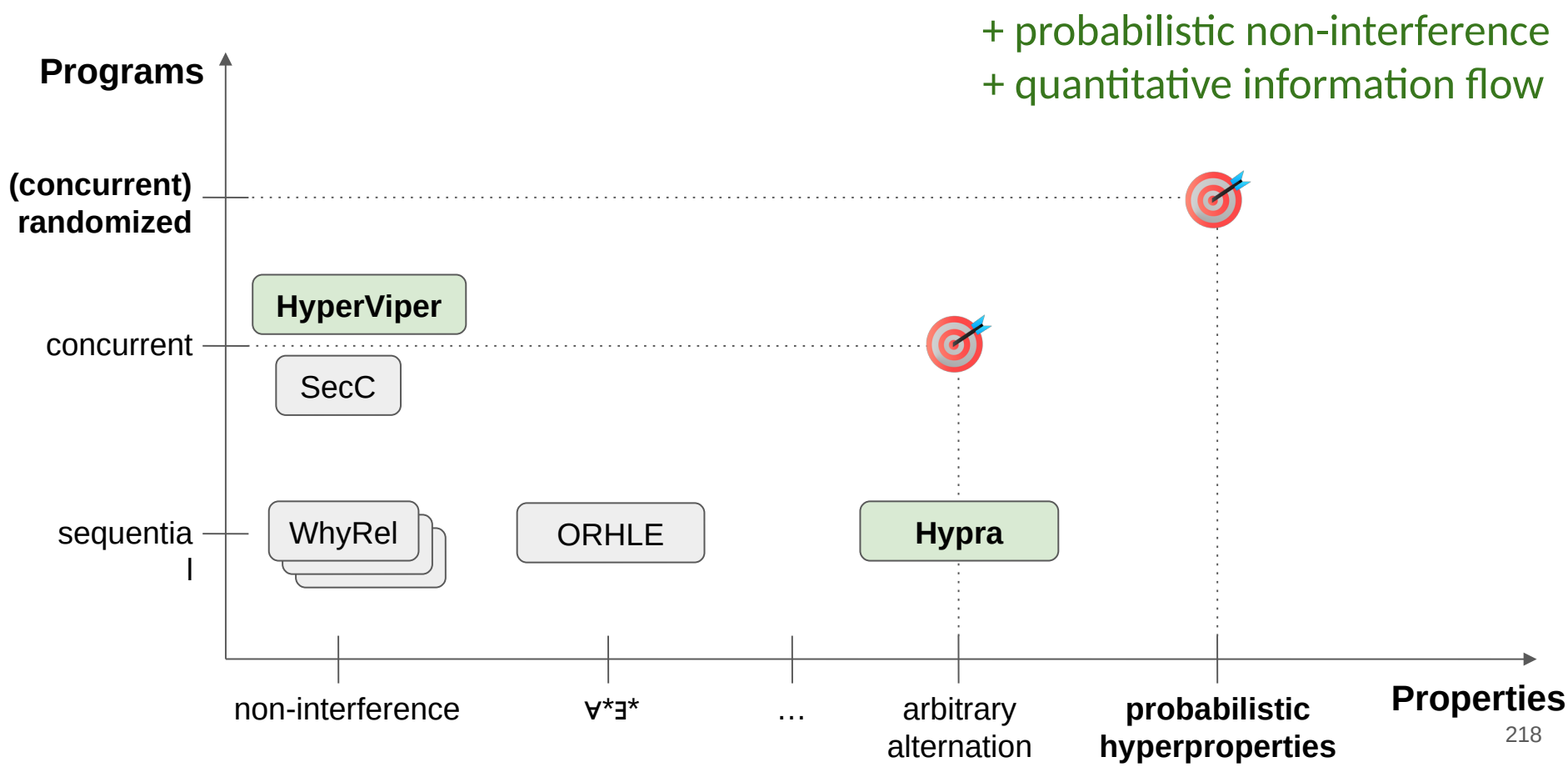


Automated Verifiers for Security and Privacy Properties

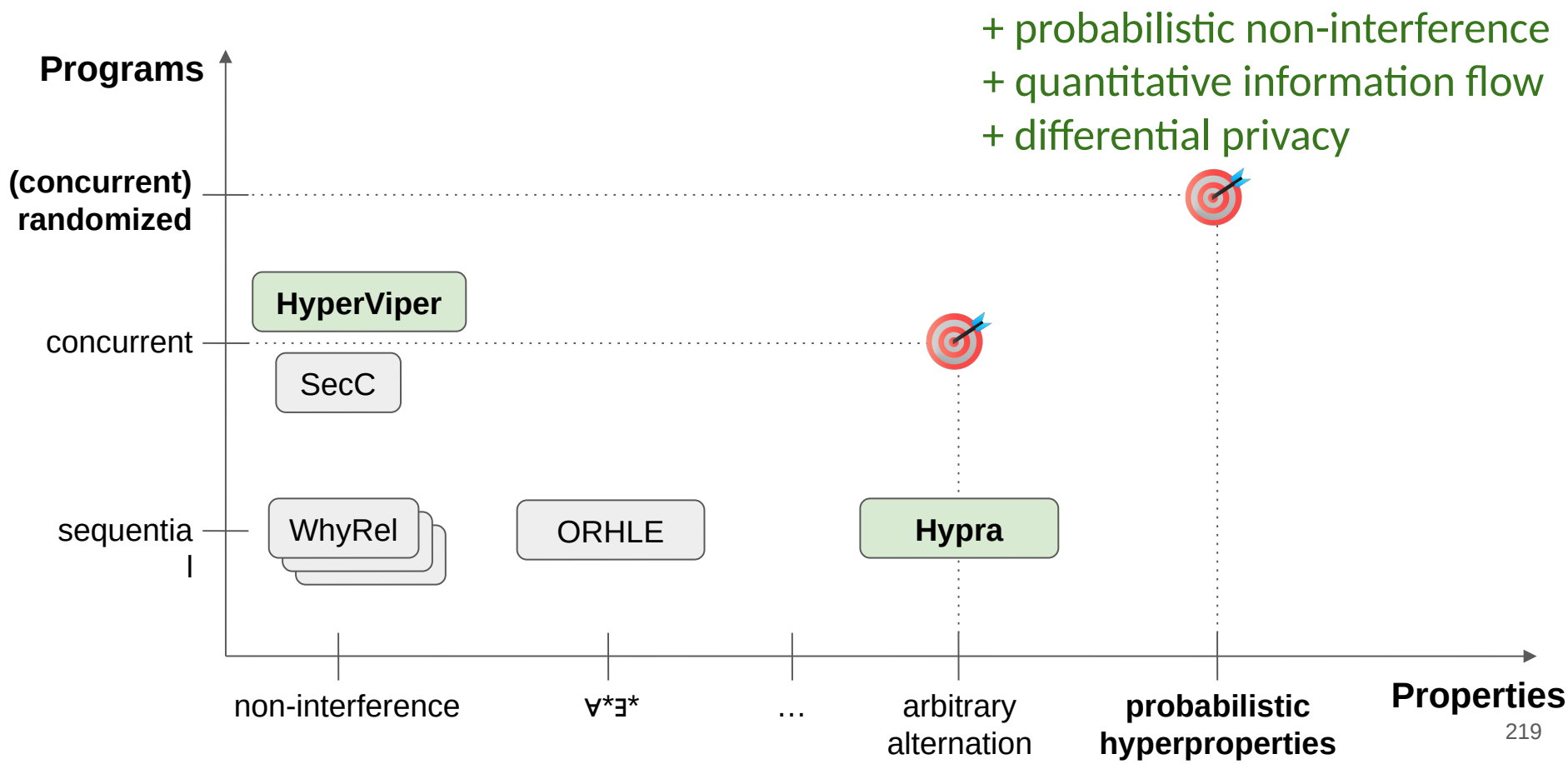
+ probabilistic non-interference



Automated Verifiers for Security and Privacy Properties



Automated Verifiers for Security and Privacy Properties



Future Research Directions

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

Problem 1
Trustworthiness

Problem 2
Expressiveness

General-purpose automated verifier for **security** and **privacy** properties

Future Research Directions

Build **provably sound** and **automated** deductive verifiers for **advanced properties**.

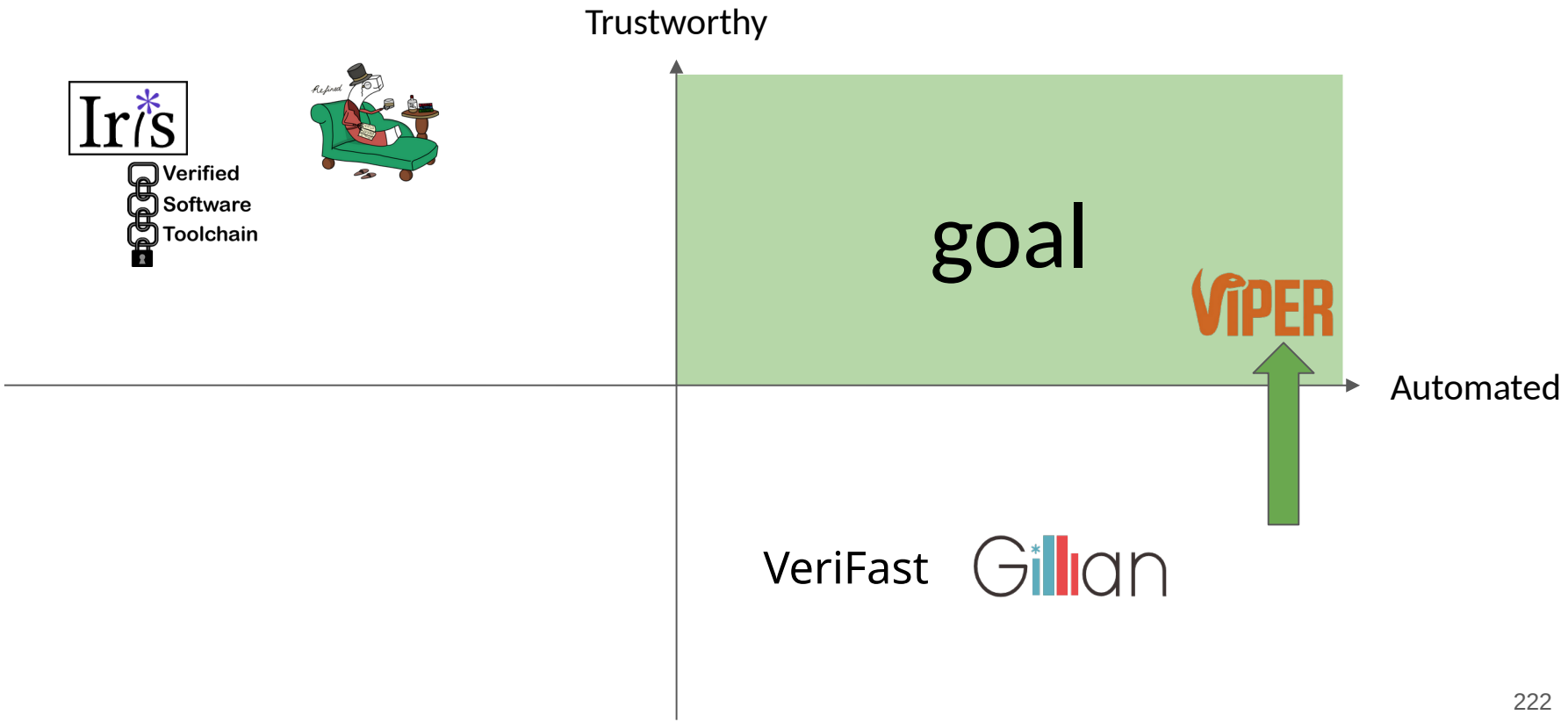
Problem 1
Trustworthiness

A **novel paradigm** to build sound automated verifiers

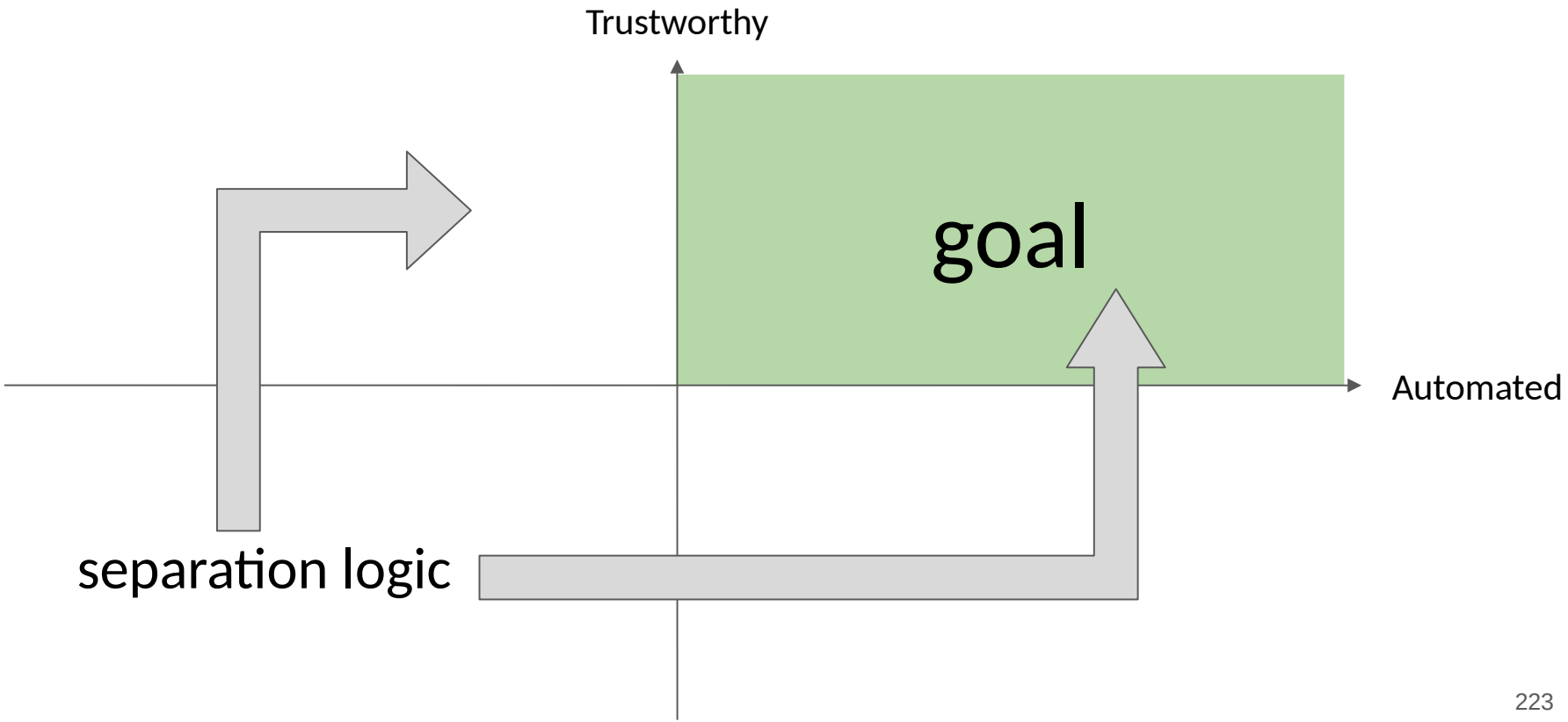
Problem 2
Expressiveness

General-purpose automated verifier for **security** and **privacy** properties

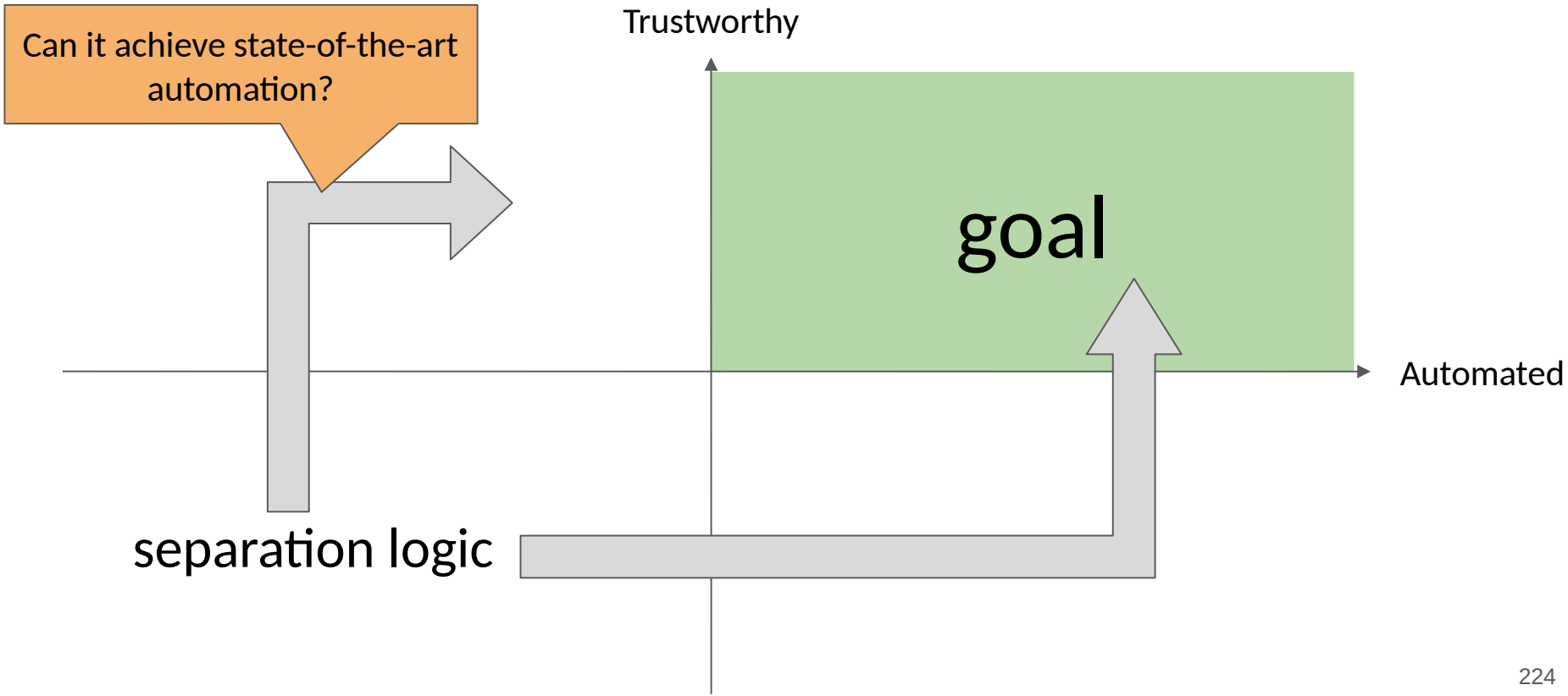
A Novel Paradigm to Build Sound Automated Verifiers



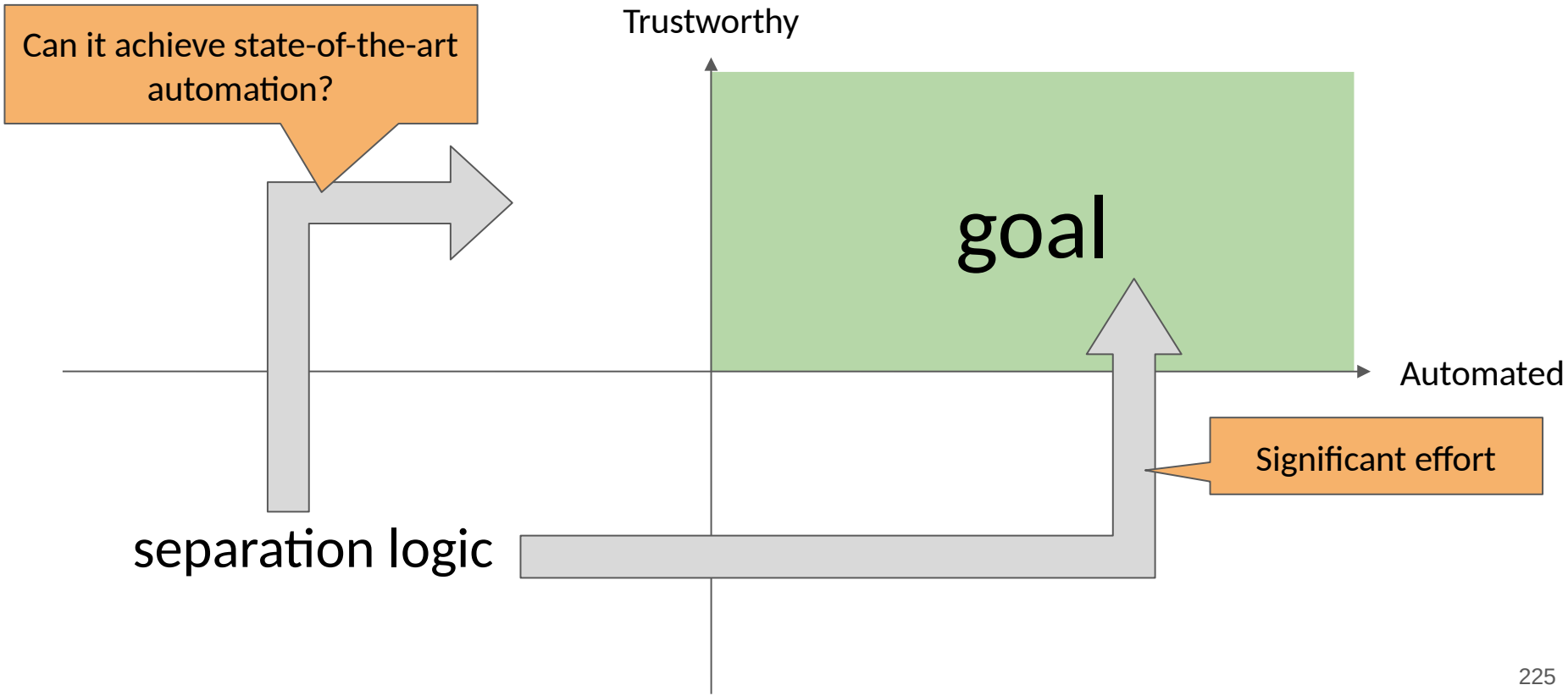
A Novel Paradigm to Build Sound Automated Verifiers



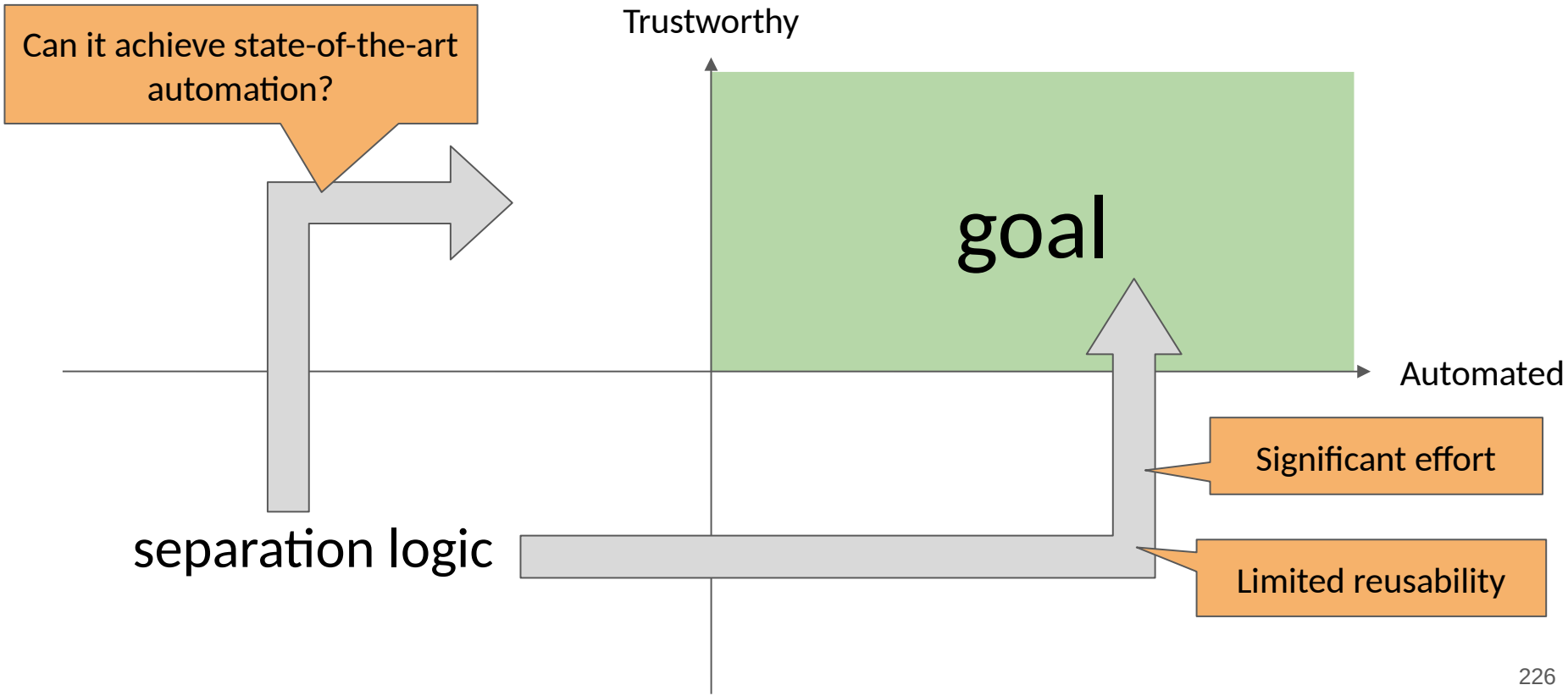
A Novel Paradigm to Build Sound Automated Verifiers



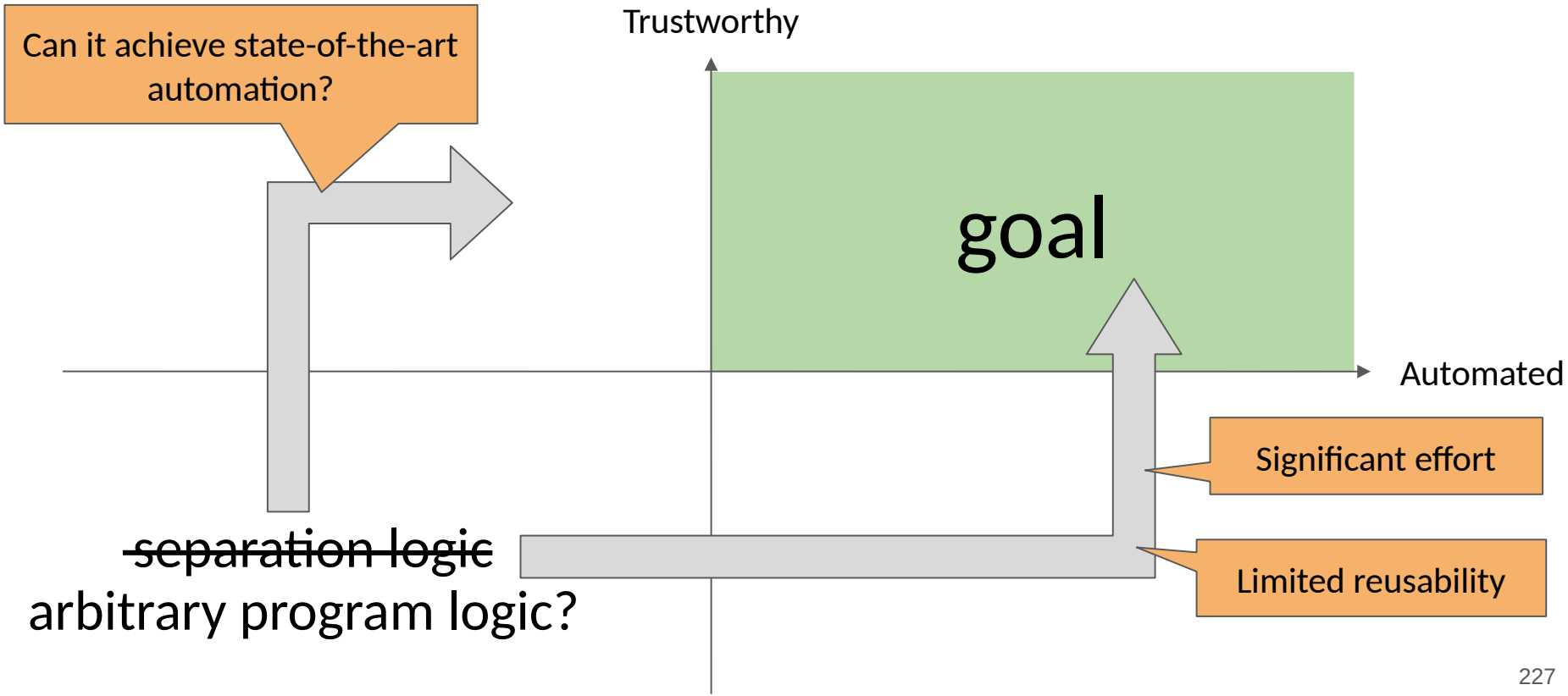
A Novel Paradigm to Build Sound Automated Verifiers



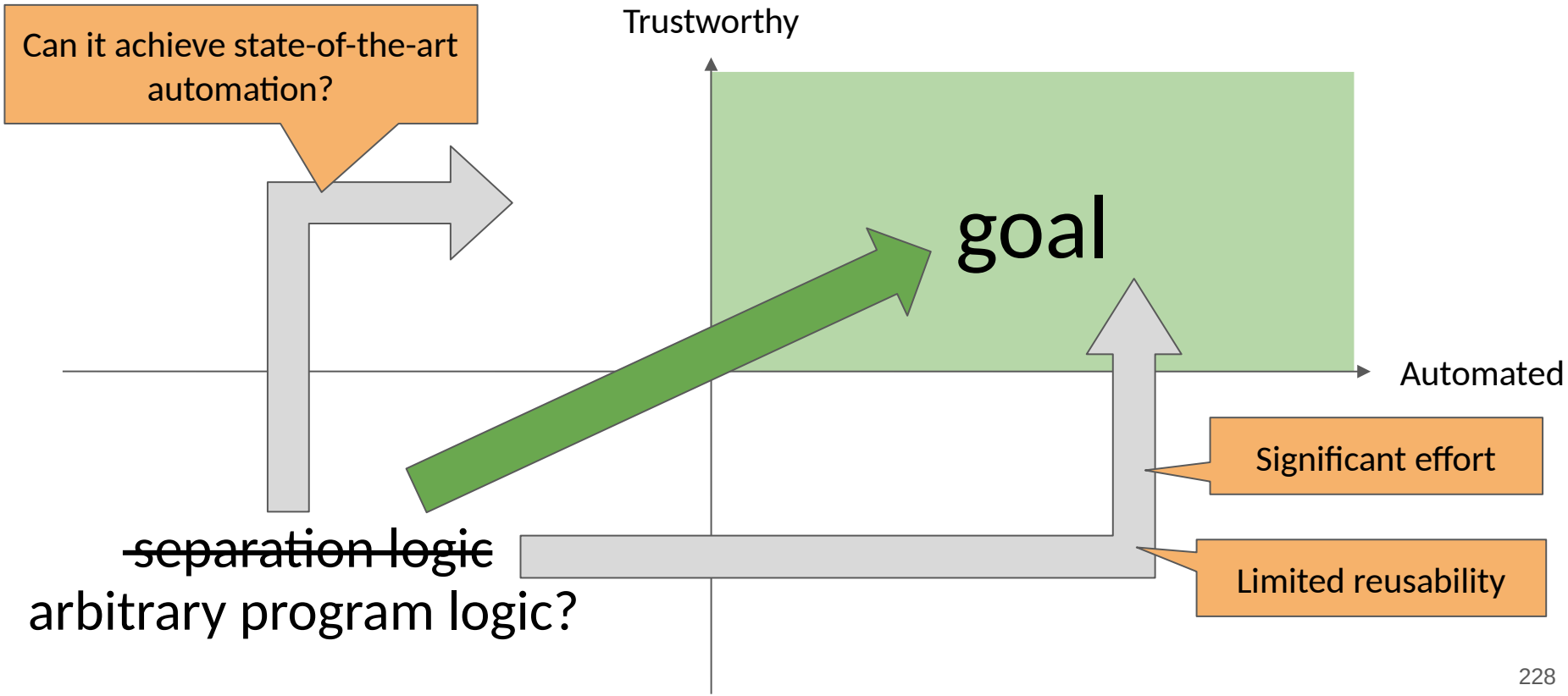
A Novel Paradigm to Build Sound Automated Verifiers



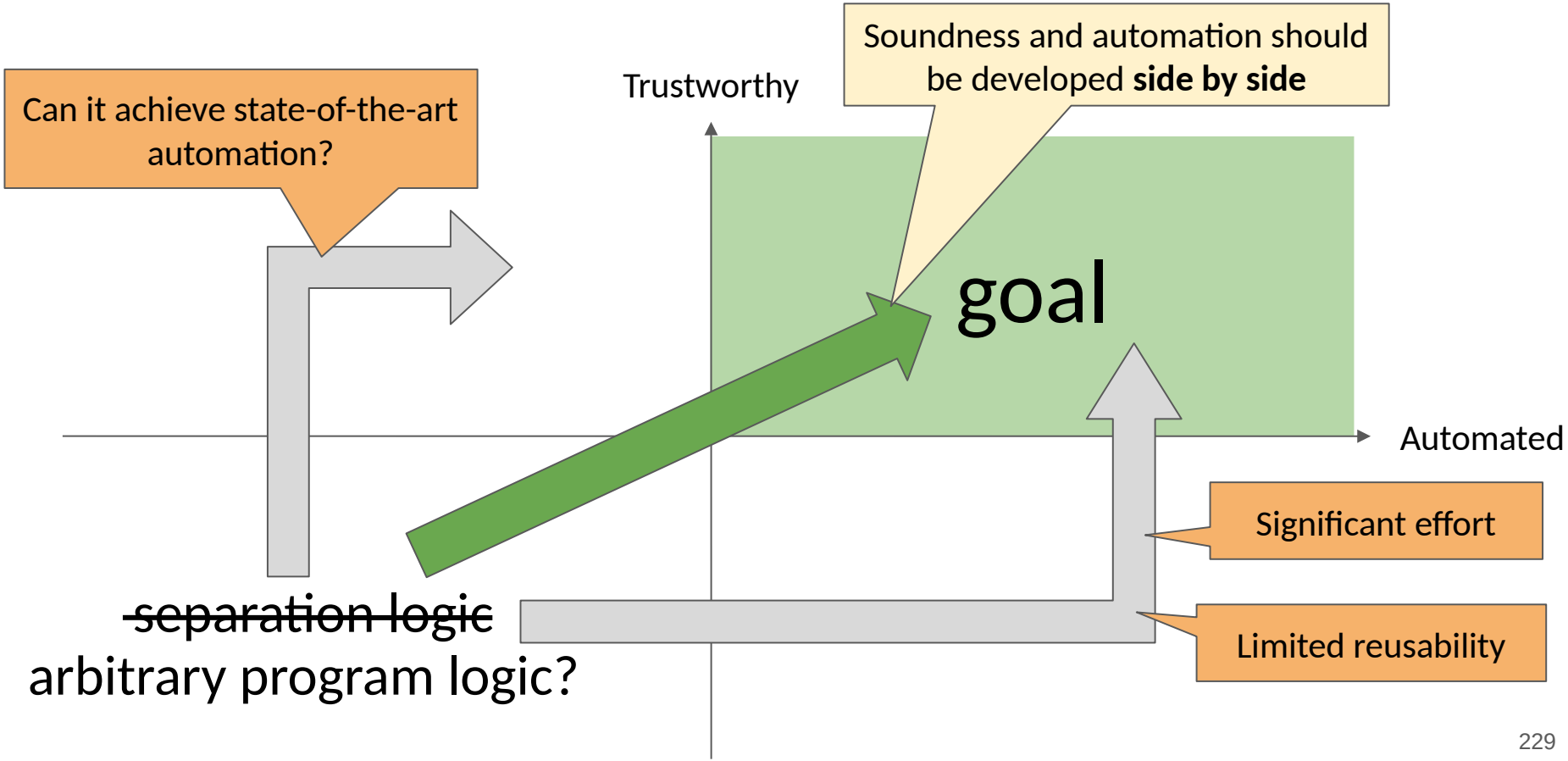
A Novel Paradigm to Build Sound Automated Verifiers



A Novel Paradigm to Build Sound Automated Verifiers

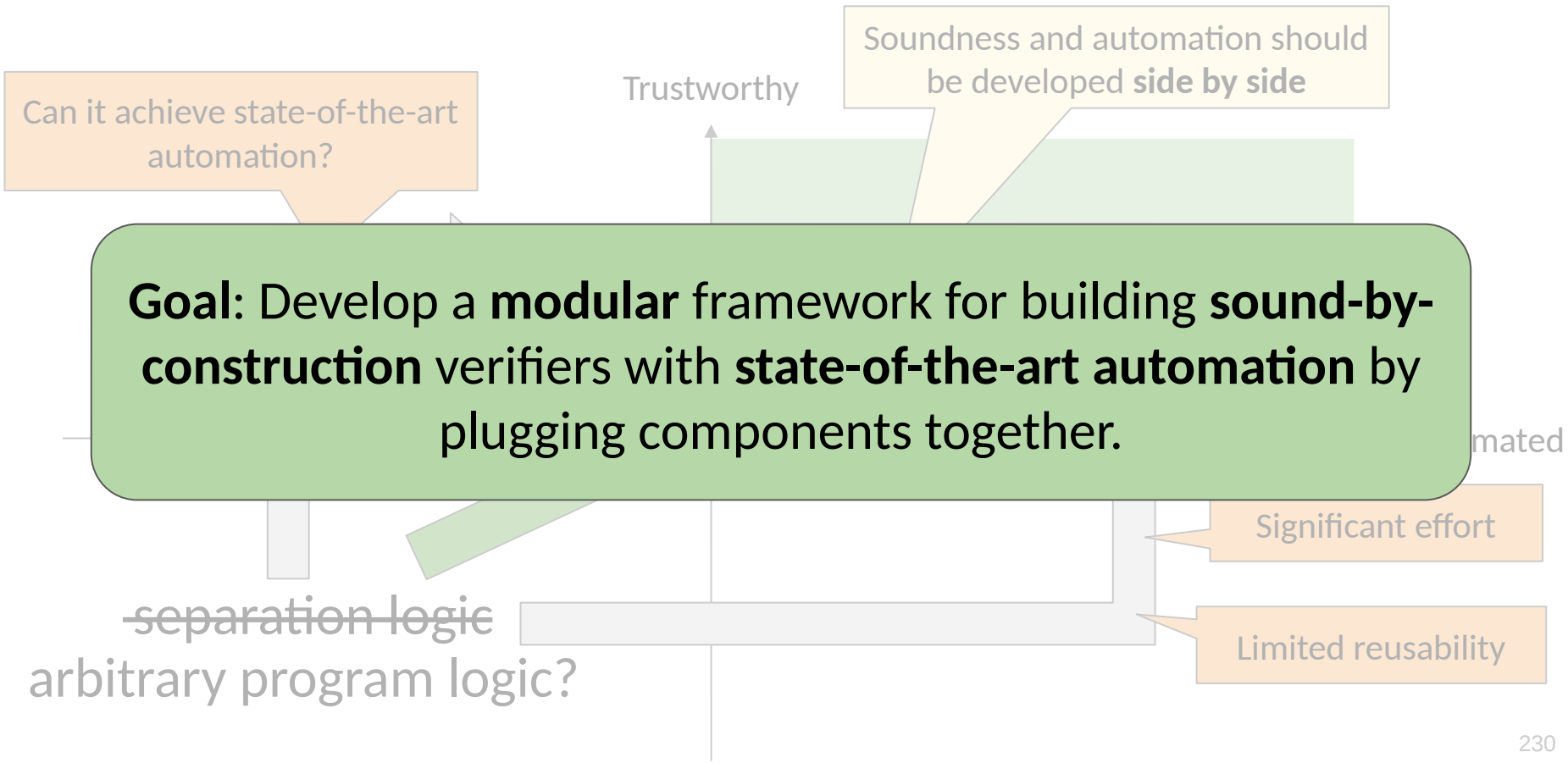


A Novel Paradigm to Build Sound Automated Verifiers



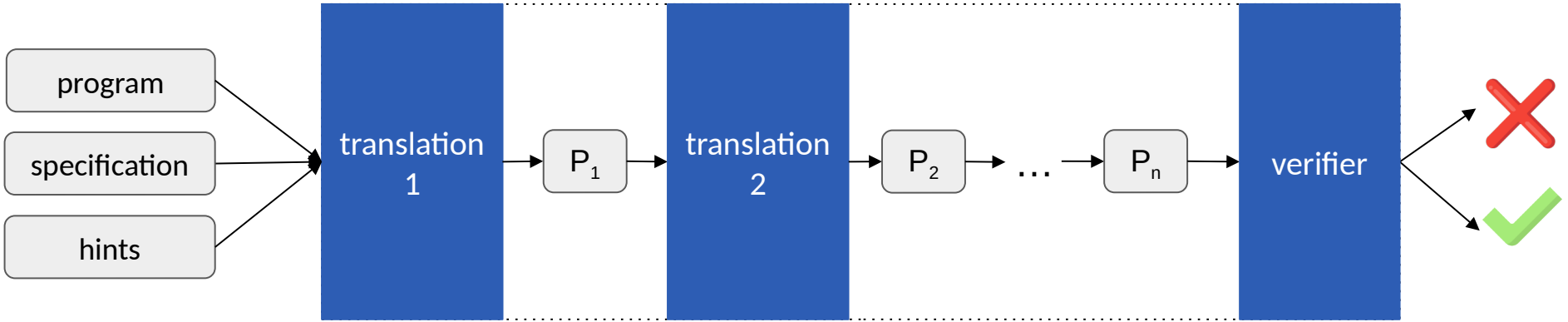
~~separation logic~~
arbitrary program logic?

A Novel Paradigm to Build Sound Automated Verifiers

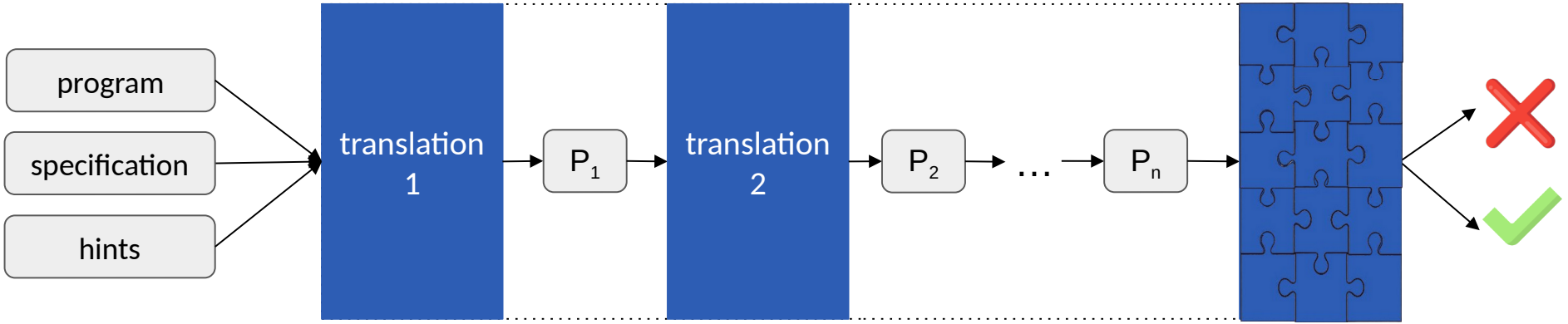


A Novel Paradigm to Build Sound Automated Verifiers

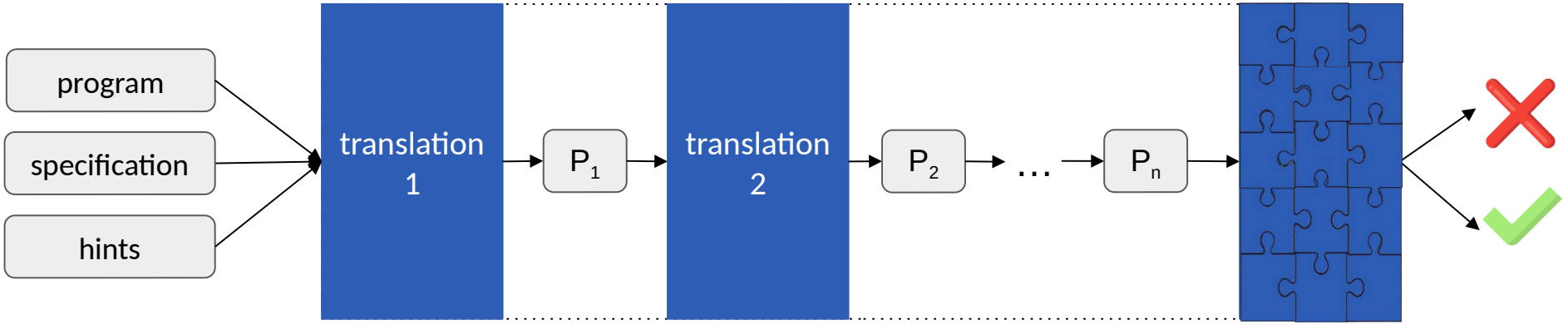
A Novel Paradigm to Build Sound Automated Verifiers



A Novel Paradigm to Build Sound Automated Verifiers

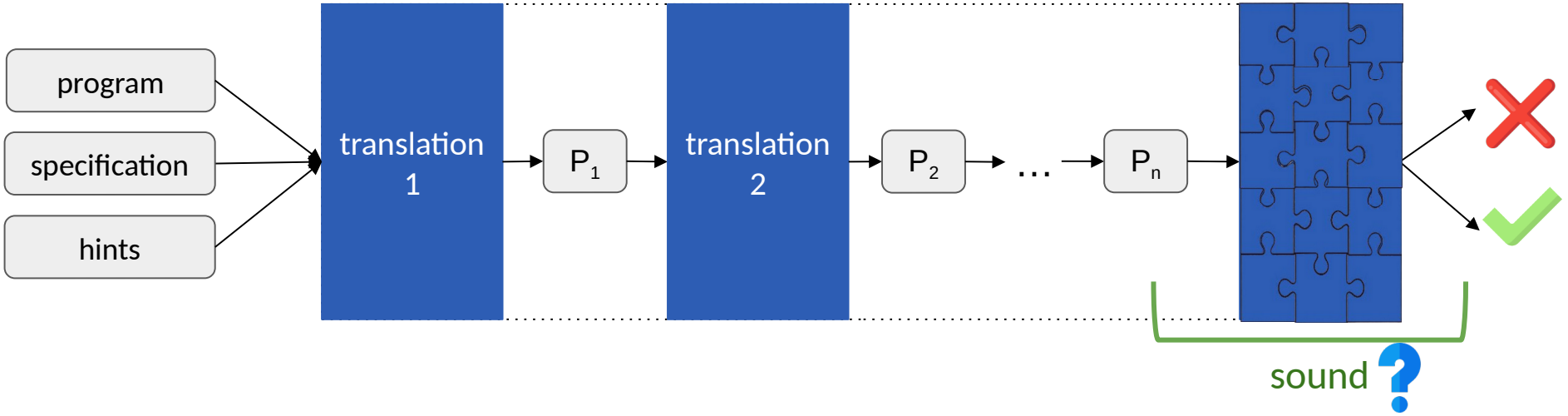


A Novel Paradigm to Build Sound Automated Verifiers



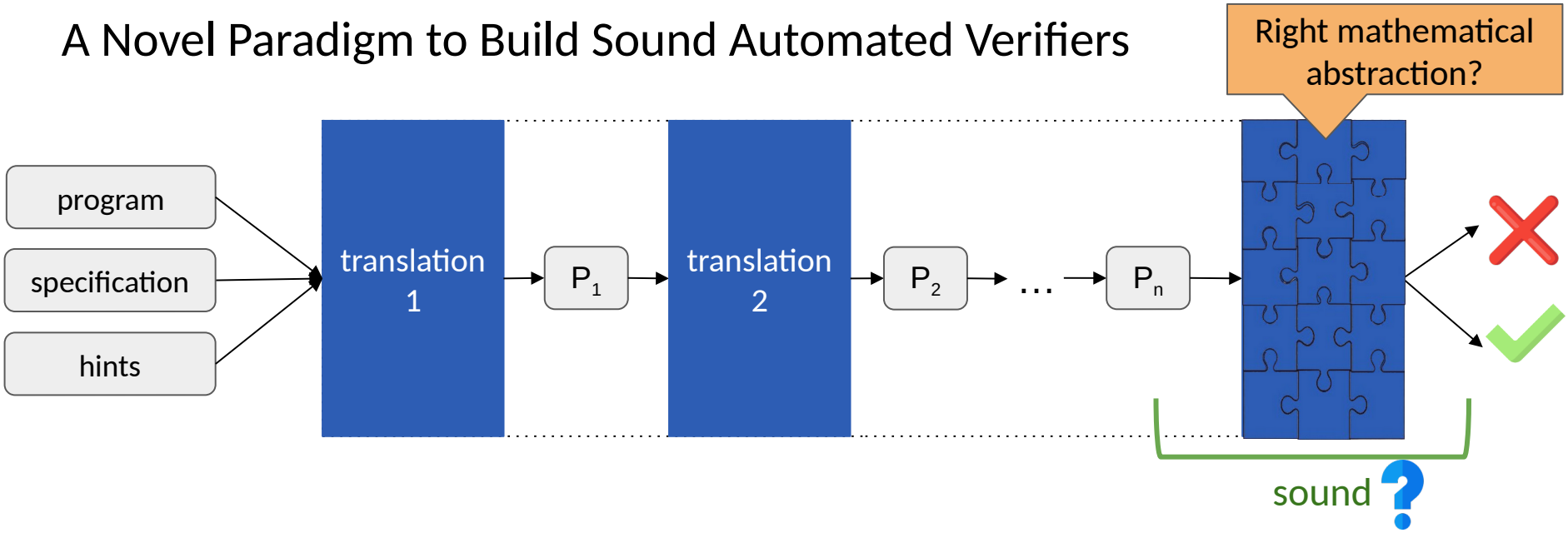
Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

A Novel Paradigm to Build Sound Automated Verifiers



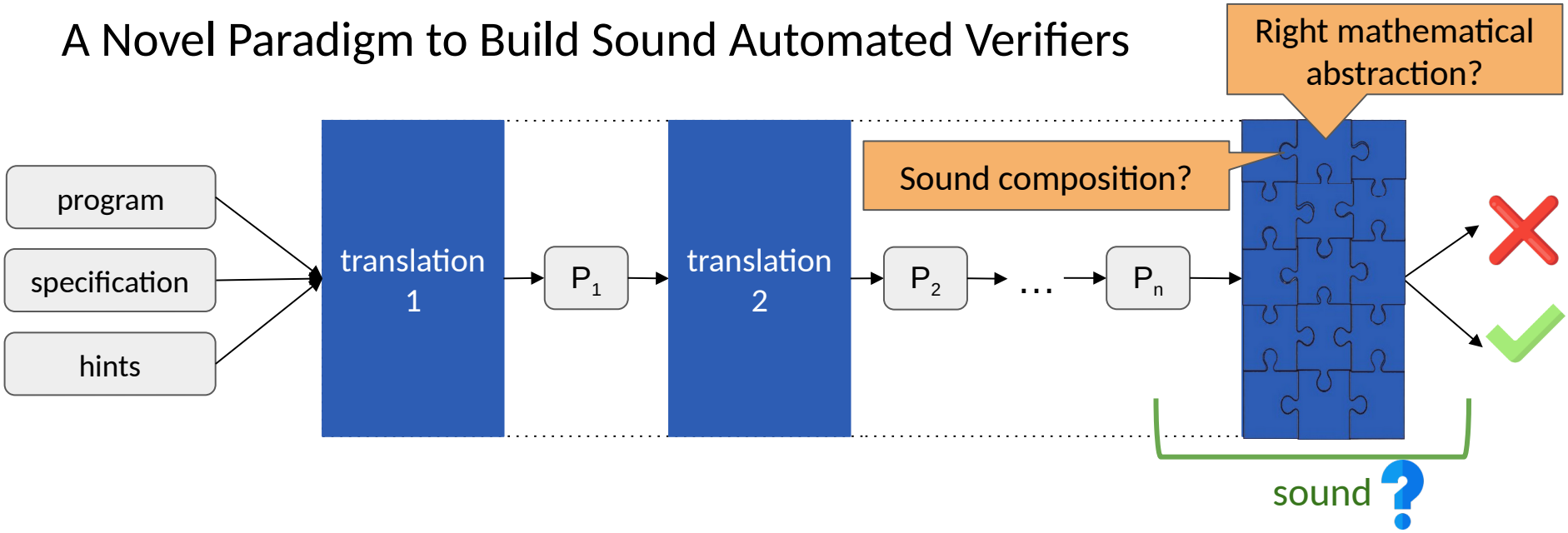
Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

A Novel Paradigm to Build Sound Automated Verifiers



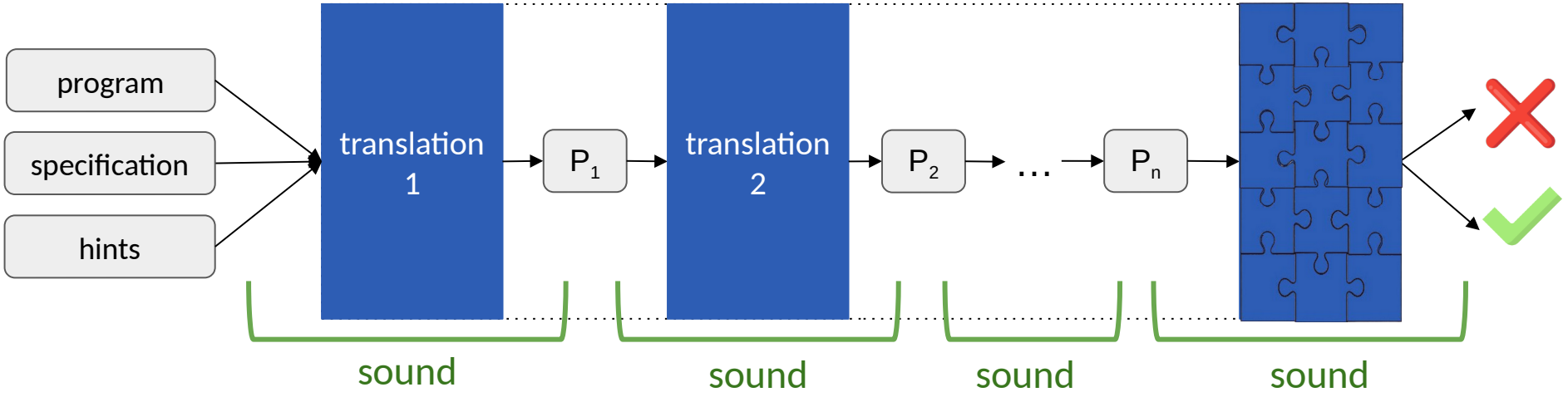
Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

A Novel Paradigm to Build Sound Automated Verifiers



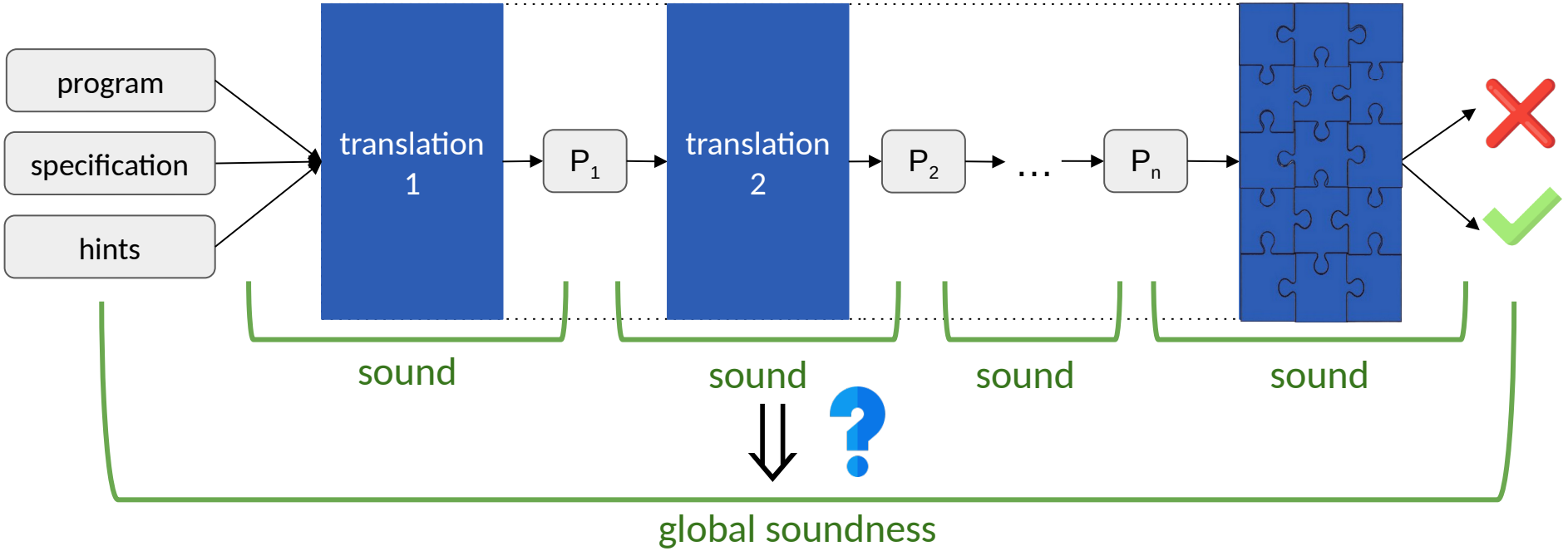
Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

A Novel Paradigm to Build Sound Automated Verifiers



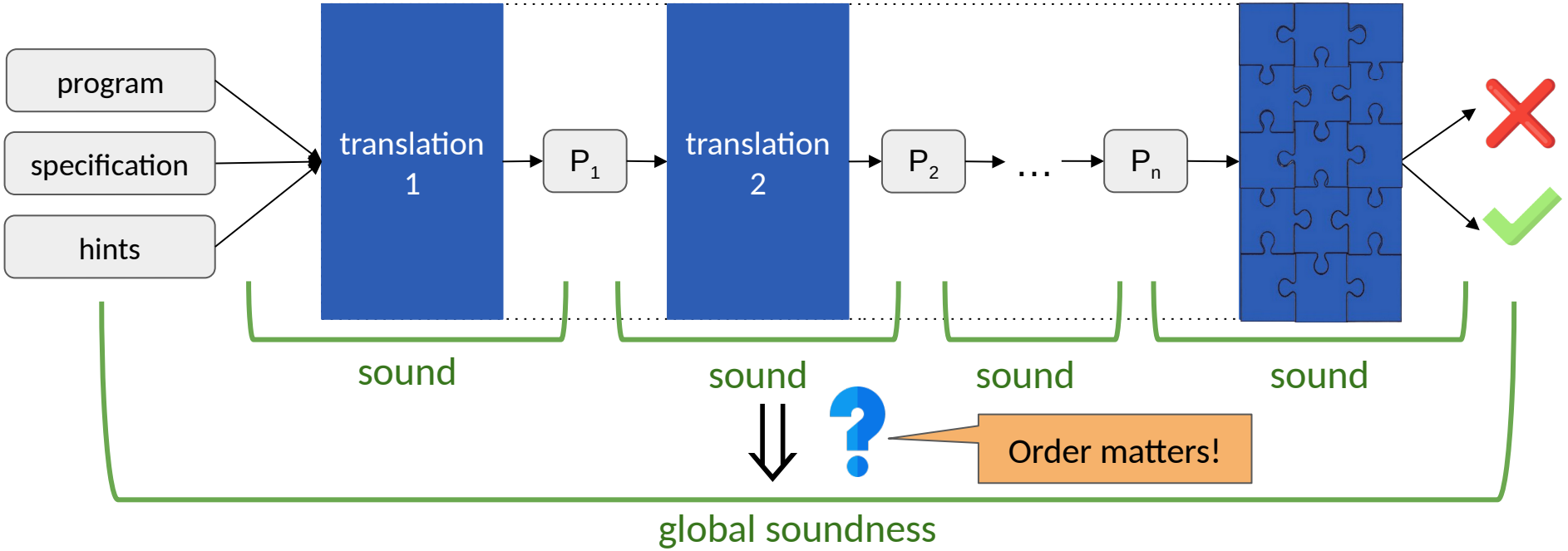
Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

A Novel Paradigm to Build Sound Automated Verifiers



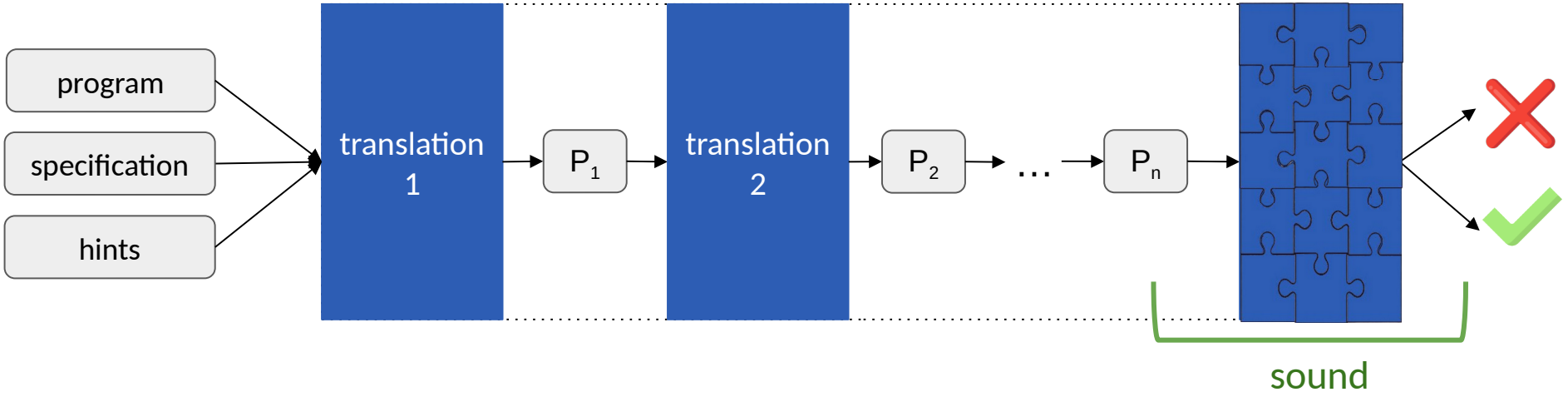
Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

A Novel Paradigm to Build Sound Automated Verifiers



Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

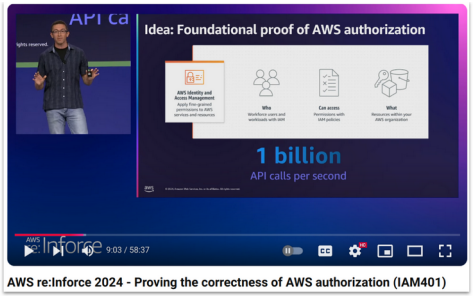
A Novel Paradigm to Build Sound Automated Verifiers



Modular framework to build **sound-by-construction** verifiers with **state-of-the-art automation** by plugging together components.

+ modular soundness
+ modular optimization
+ reusability
+ extensibility

Summary



AWS re:Inforce 2024 - Proving the correctness of AWS authorization (IAM401)

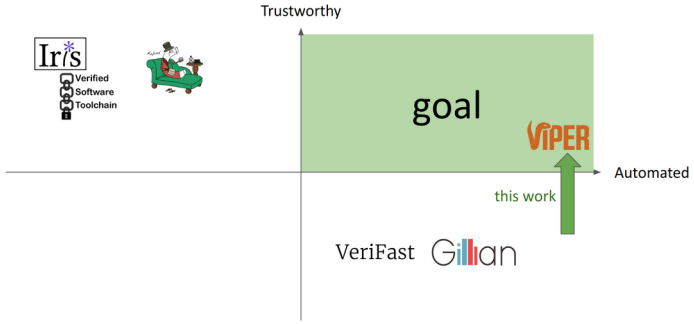
1 billion times/second

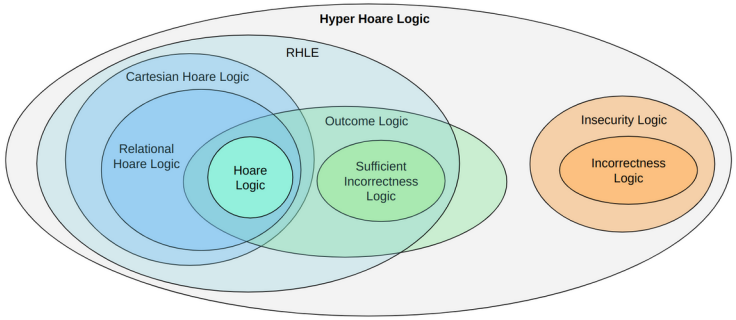
~3,000 lines of code

~3x performance

Problem 1
Trustworthiness

Problem 2
Expressiveness





CommCSL
[PLDI'23]

Hyper Hoare Logic
[PLDI'24]

Hypra *[OOPSLA'24]*

Magic wands
[CAV'22]

Fractional predicates
[OOPSLA'22]

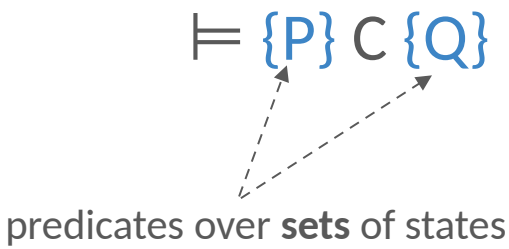
Verification-preserving inlining
[OOPSLA'23]

Per-run validation for Viper
[PLDI'24]

Foundations for translational separation logic verifiers
[POPL'25]

Backup Slides

Hyper Hoare Logic – Triples



Hyper Hoare Logic – Triples

$$\models \{P\}\ C\ \{Q\} \triangleq (\ \forall S.\ P(S) \Rightarrow Q(\text{reach}(C, S))\ )$$

predicates over **sets** of states

Hyper Hoare Logic – Triples

$$\models \{P\}\ C\ \{Q\} \triangleq (\ \forall S.\ P(S) \Rightarrow Q(\mathrm{reach}(C, S))\ )$$

predicates over **sets** of states

set of initial states

Hyper Hoare Logic – Triples

$$\models \{P\}\ C\ \{Q\} \triangleq (\ \forall S.\ P(S) \Rightarrow Q(\text{reach}(C, S))\ )$$

predicates over **sets** of states

set of initial states

set of all states reachable by executing C in some state from S

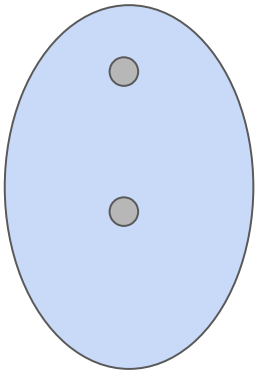
Hyper Hoare Logic – Triples

$$\models \{P\}\ C\ \{Q\} \triangleq (\ \forall S.\ P(S) \Rightarrow Q(\text{reach}(C, S))\ )$$

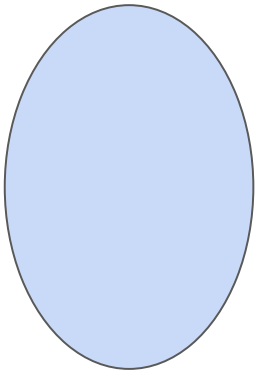
predicates over **sets** of states

set of initial states

set of all states reachable
by executing C in some state from S



S



reach(C, S)

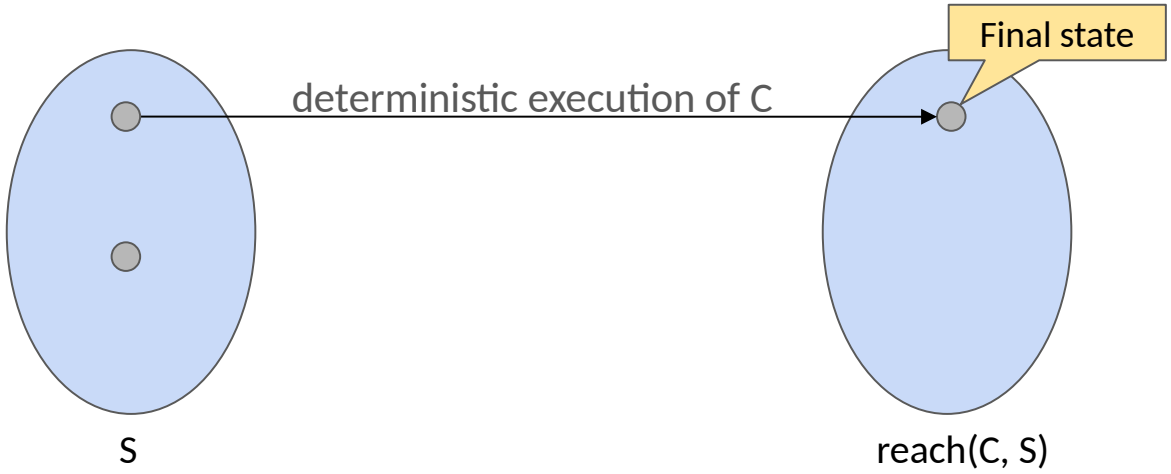
Hyper Hoare Logic – Triples

$$\models \{P\}\ C\ \{Q\} \triangleq (\ \forall S.\ P(S) \Rightarrow Q(\text{reach}(C, S))\ )$$

predicates over **sets** of states

set of initial states

set of all states reachable by executing C in some state from S



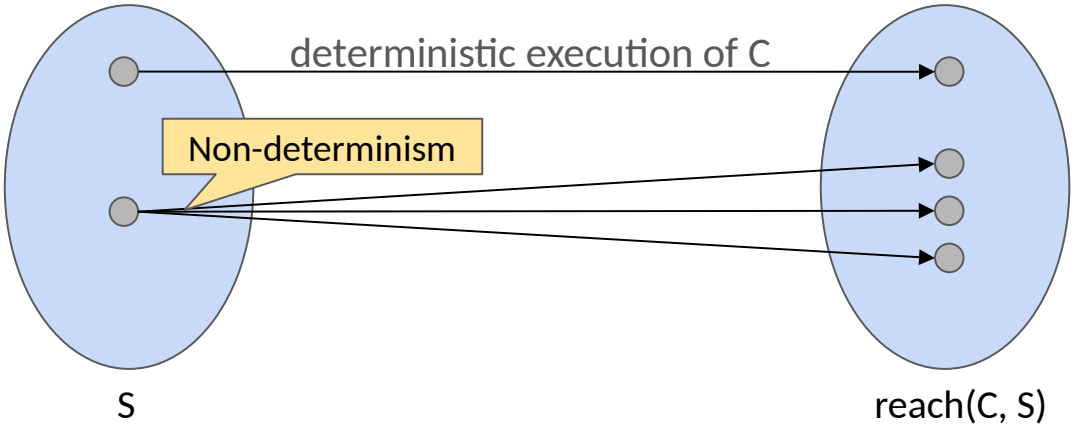
Hyper Hoare Logic – Triples

$$\models \{P\}\ C\ \{Q\} \triangleq (\ \forall S.\ P(S) \Rightarrow Q(\text{reach}(C, S))\ )$$

predicates over **sets** of states

set of initial states

set of all states reachable by executing C in some state from S

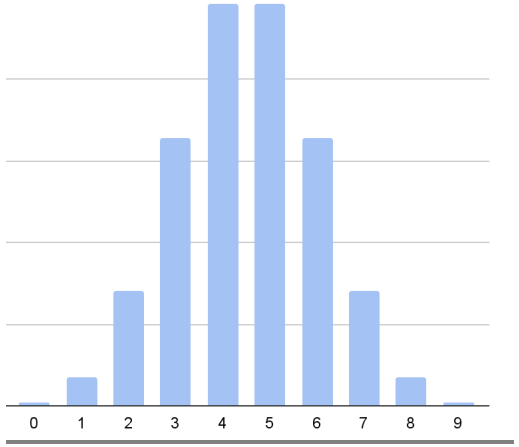


Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
   key := 0
   for i in range(9):
      key += randInt(0,
1)
   res := (s + key) % 10
   print(res)
```

Information Flow Security: Probabilistic Non-Interference

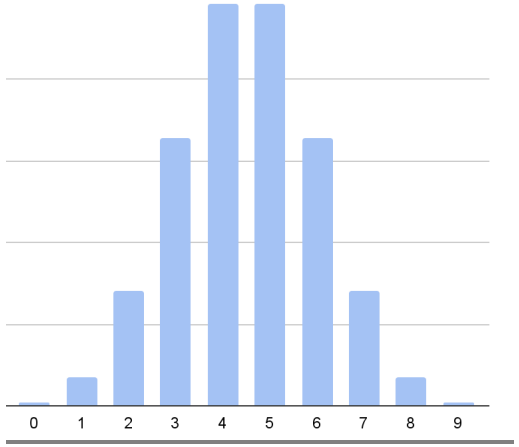
```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```



Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```

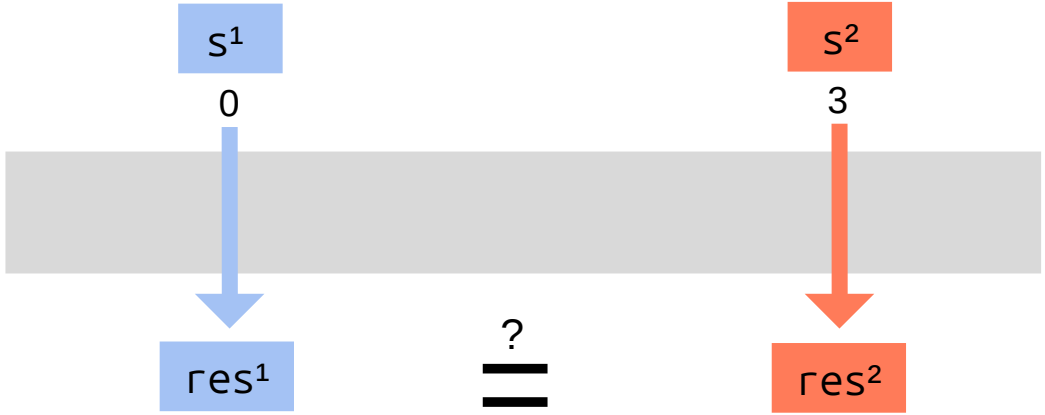
Leaks information!



Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```

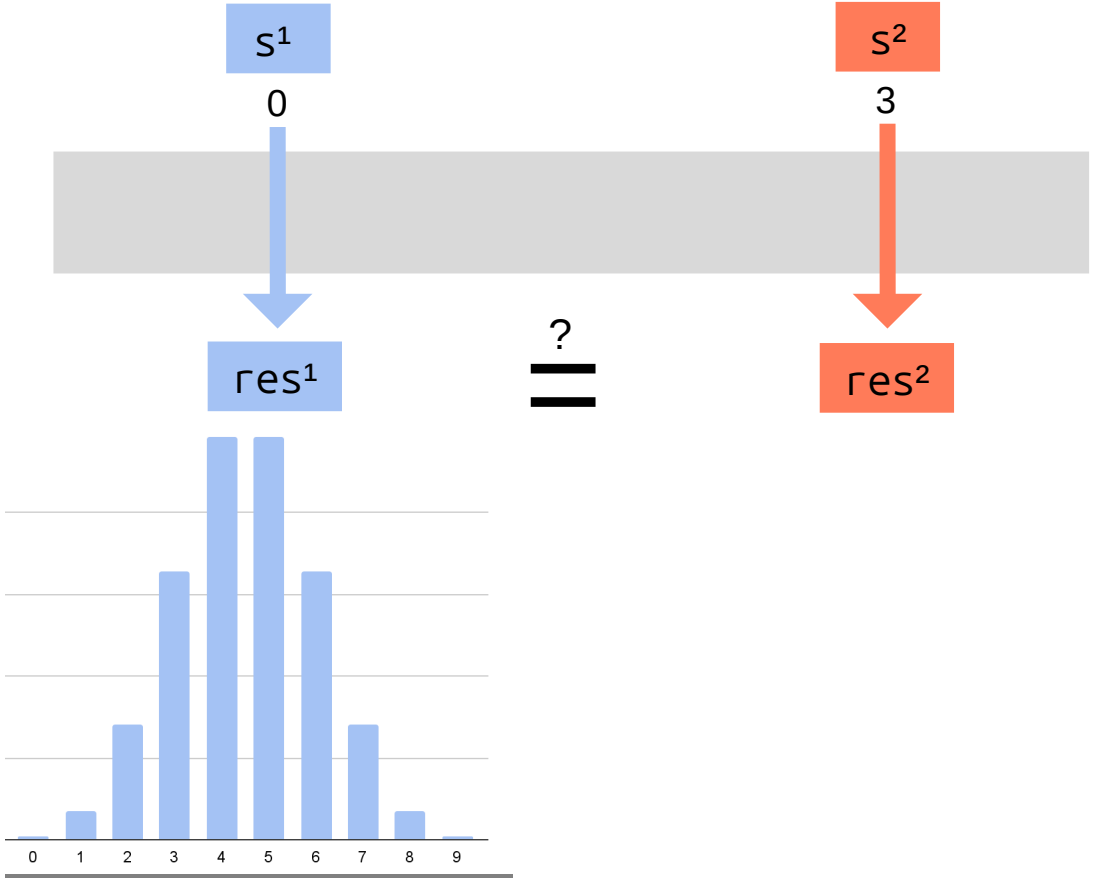
Leaks information!



Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```

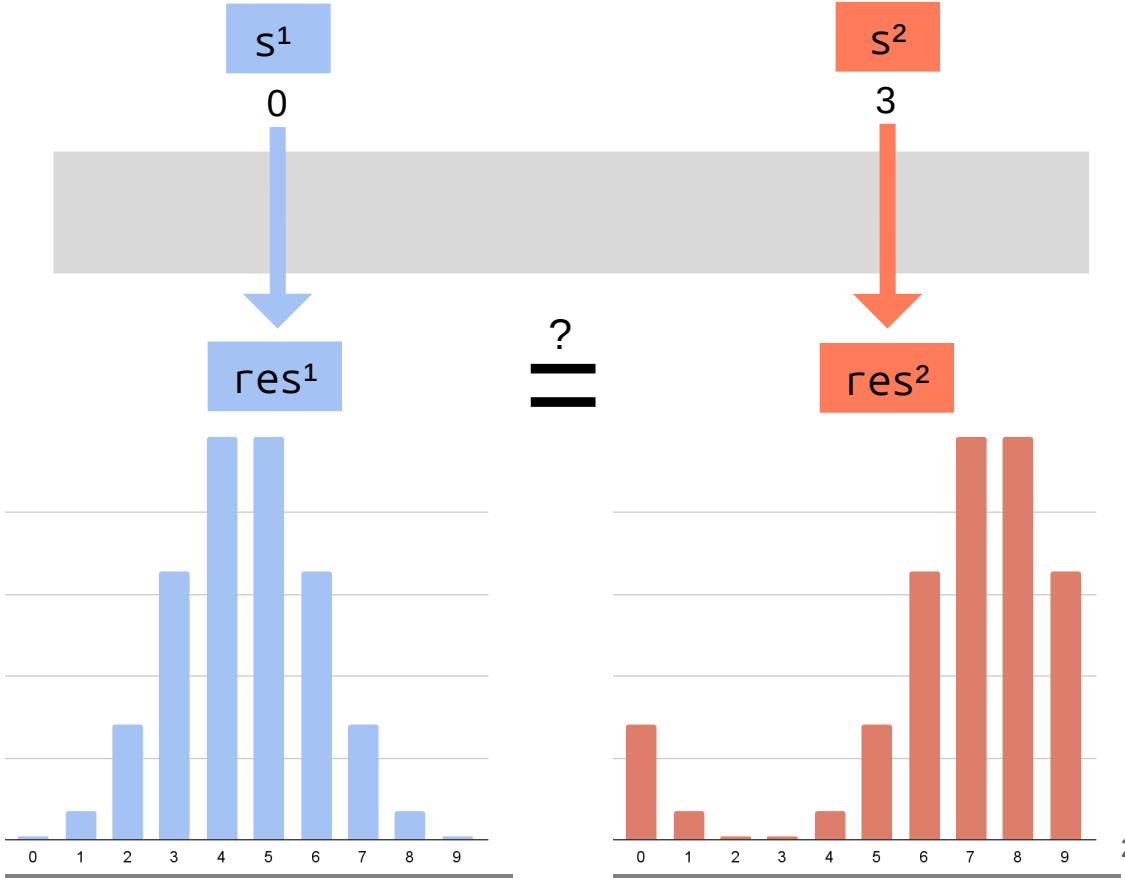
Leaks information!



Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```

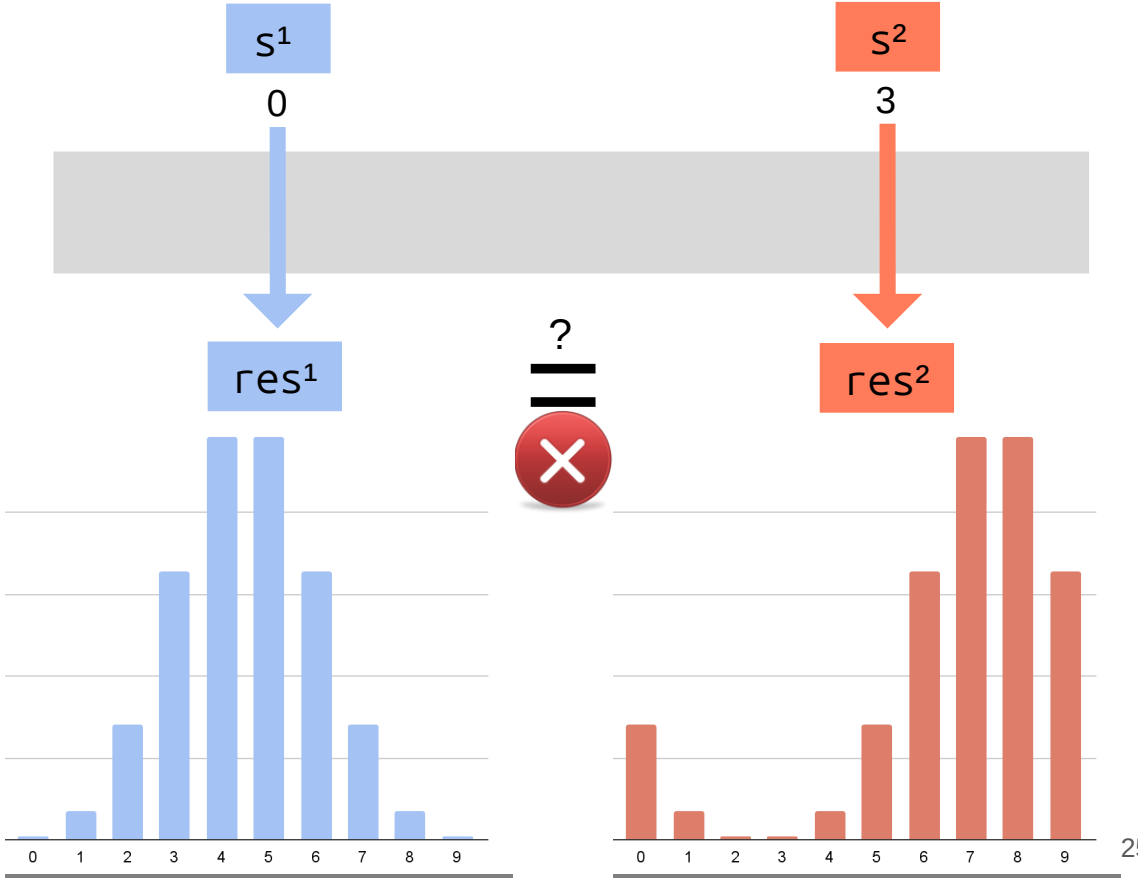
Leaks information!



Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```

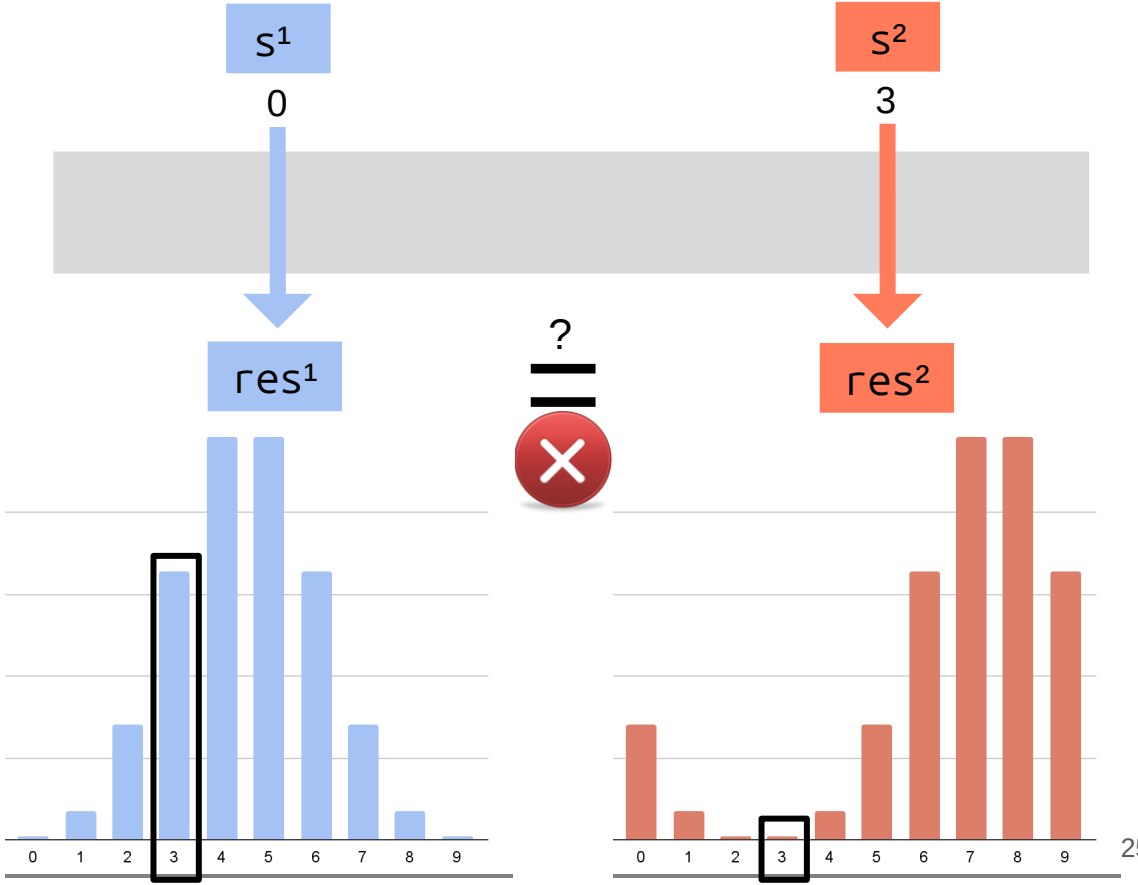
Leaks information!



Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```

Leaks information!



Information Flow Security: Probabilistic Non-Interference

```
def badOneTimePad(s):
    key := 0
    for i in range(9):
        key += randInt(0,
1)
    res := (s + key) % 10
    print(res)
```

Leaks information!