Refinement Types

Niki Vazou
IMDEA Software Institute
FOPSS 2023

Refinement Types

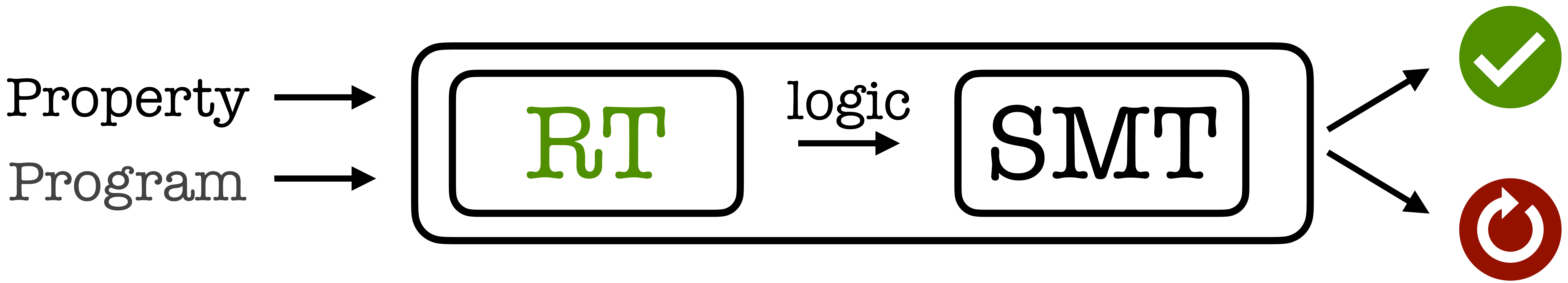
types refined with logical predicates

Existing Type: $(!!) :: [a] \to \mathrm{Int} \to a$

Refinement Type: $(!!) :: xs{:}[a] \to i{:}\{\mathrm{Int} \mid 0 \leq i < \mathrm{len}\ xs\} \to a$

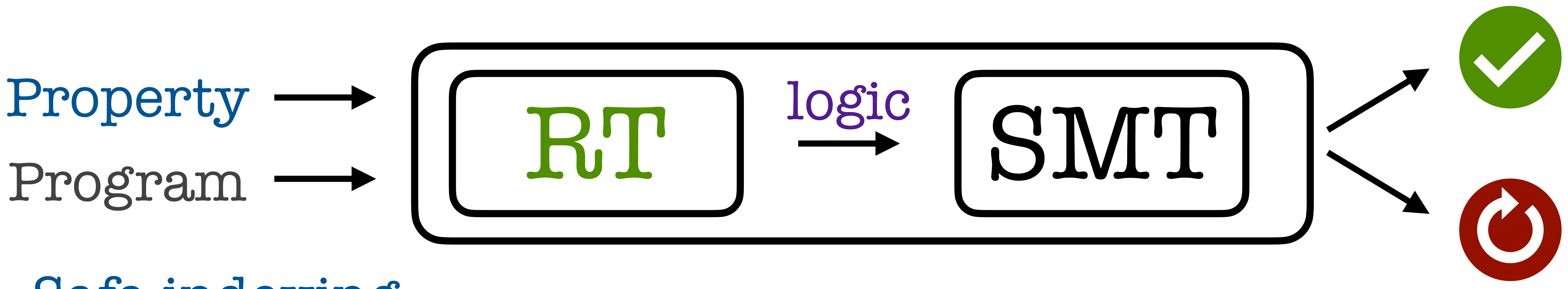
Logical predicate
here encodes safe indexing

Refinement Types



* SMT: A tool that automatically decides validity of logical formulas.

Refinement Types



Safe-indexing

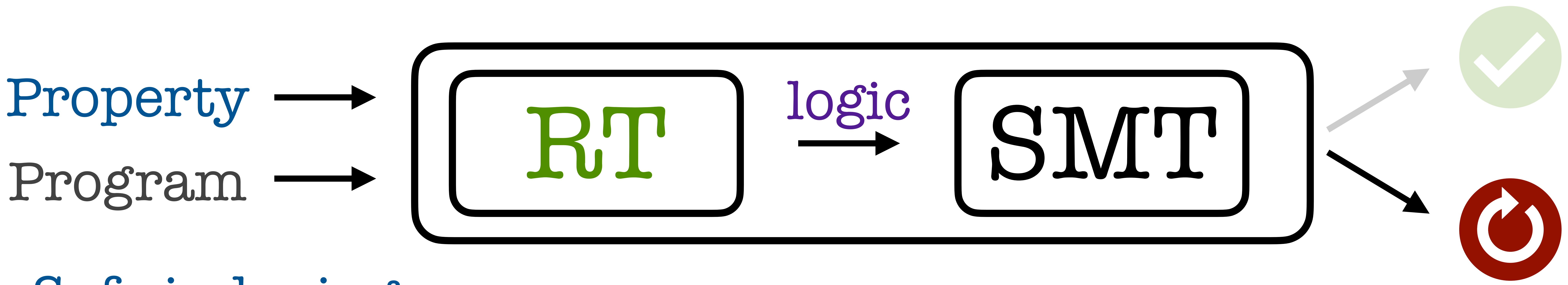
```
xs!!i
```

INFO

$\Rightarrow$

$0 \leq i < \text{len } xs$

Refinement Types



Safe-indexing

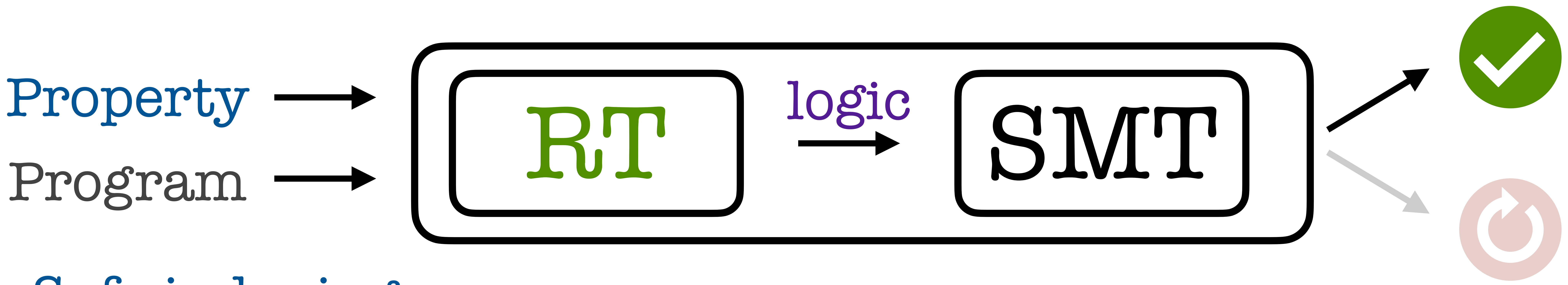
```
xs!!i
```

true

$\Rightarrow$

$0 \leq i < \text{len xs}$

Refinement Types

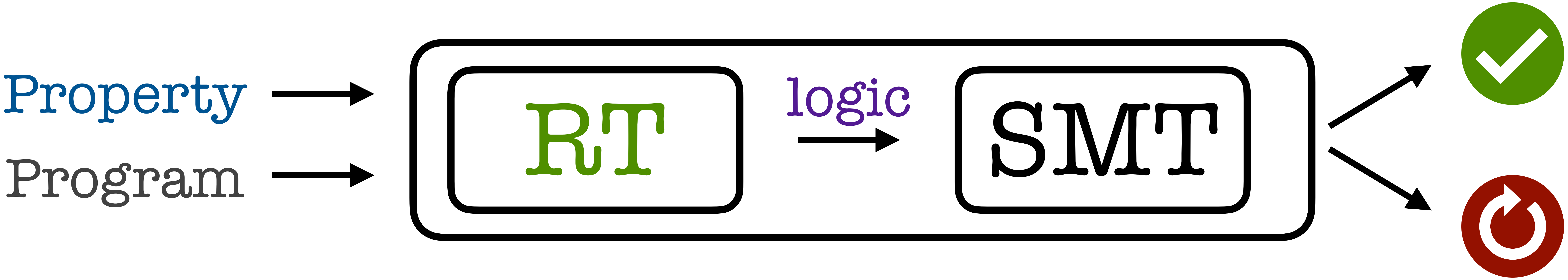


Safe-indexing

```
if 0 <= i && i < length xs
  then Just (xs!!i)
  else Nothing
```

$$0 \leq i < \text{len xs} \Rightarrow 0 \leq i < \text{len xs}$$

Refinement Types

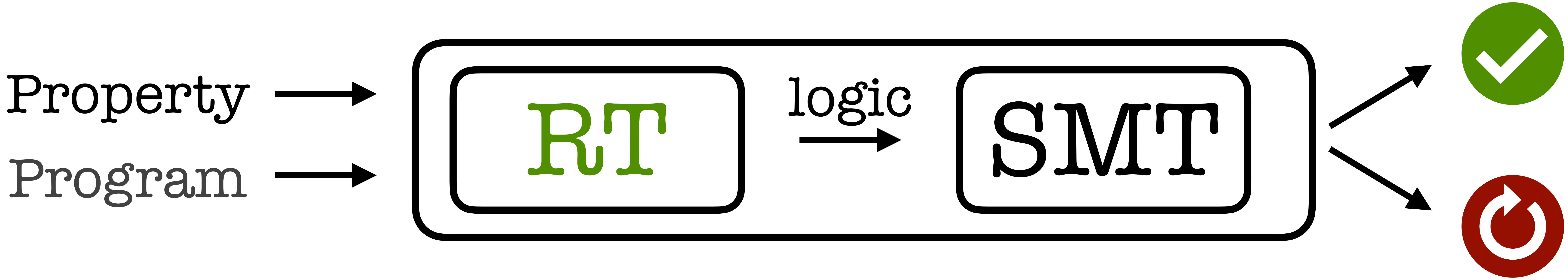


Verification is

Case sensitive

```
if 0 <= i && i < length xs
  then Just (xs!!i)
  else Nothing
```

Refinement Types




```
xs:[{v:Int | 0 /= v }]
```

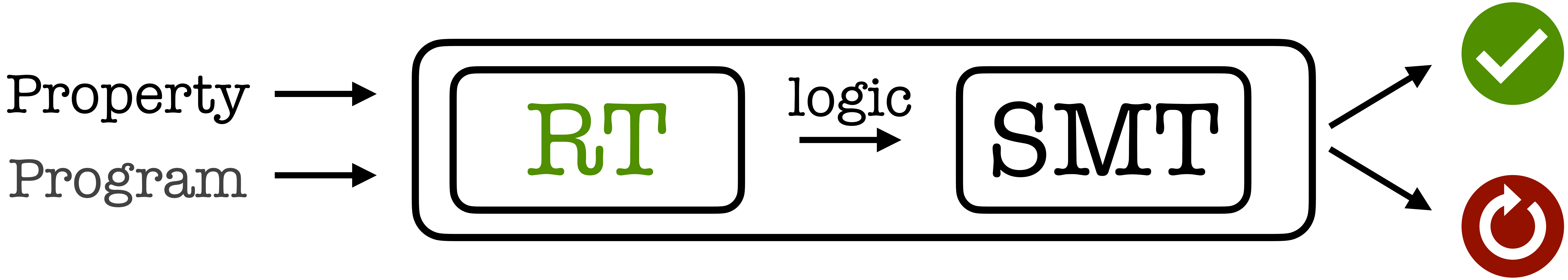
```
42 `div` (xs!!i)
```

Verification is

Type-based

Case sensitive

Refinement Types



Logic is only
SMT decidable theories

Enough for safe indexing,
but what about deep props?

Verification is

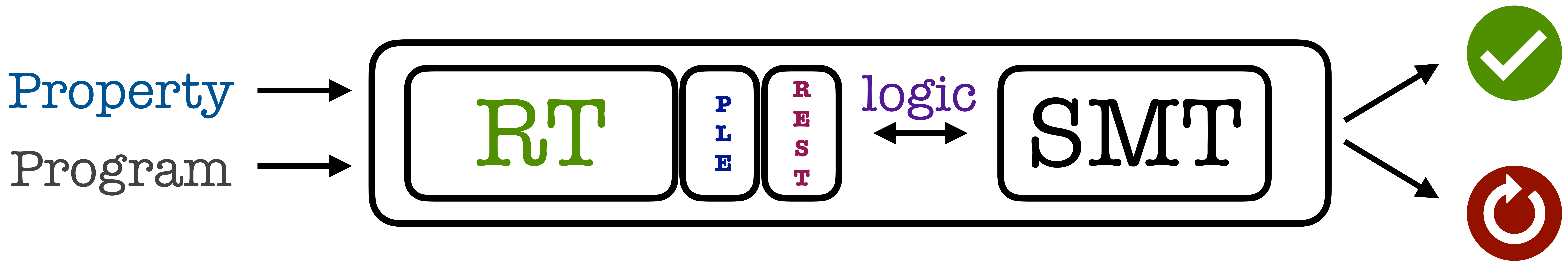
Decidable

Type-based

Case sensitive

*To avoid unpredictable SMT-verification (a.k.a. “the butterfly effect”)

Refinement Types

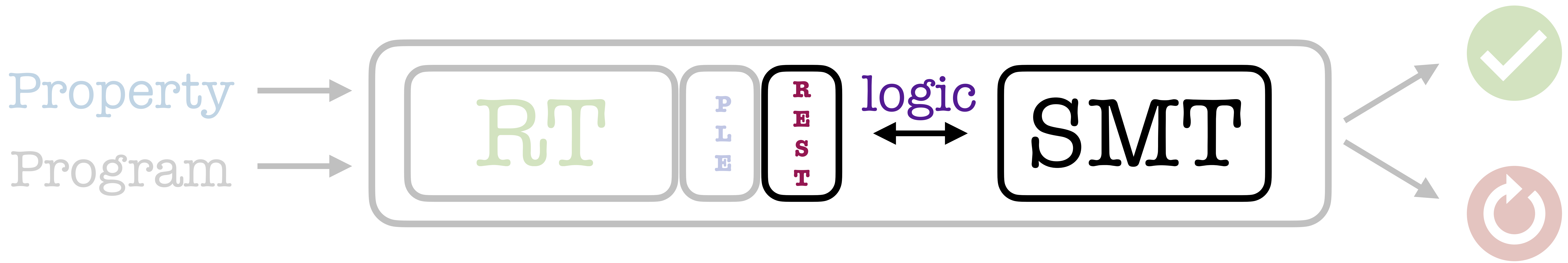


TRICK: layers before SMT

PLE: Proof by Logical Evaluation

REST: REwriting using SmT

REST: REwriting using SmT



Properly instantiate axioms as rewrite rules

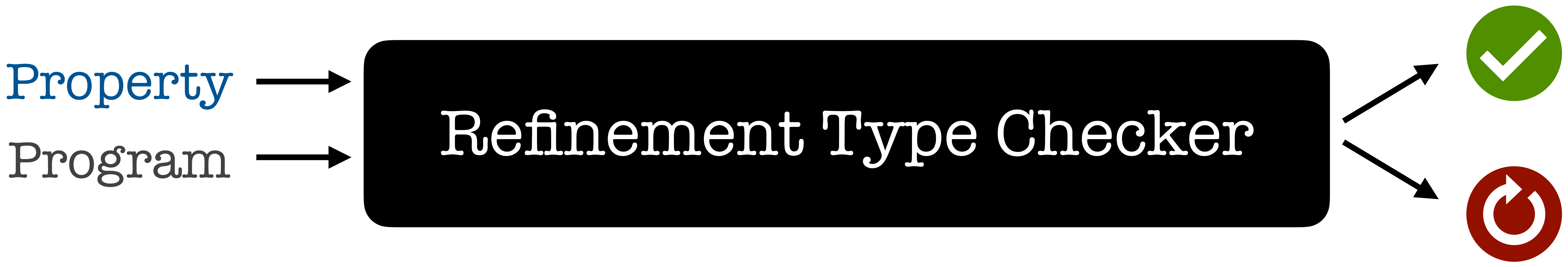
```
{-@ rewriteWith distributivity [assoc, rightId] @-}
```

Online Termination to be permissive but not diverging

Not refinement types specific

REST: Integrating Term Rewriting with Program Verification,
by Grannan, Darulova, Summers, and Vazou. ECOOP'22.

Refinement Types are



Decidable

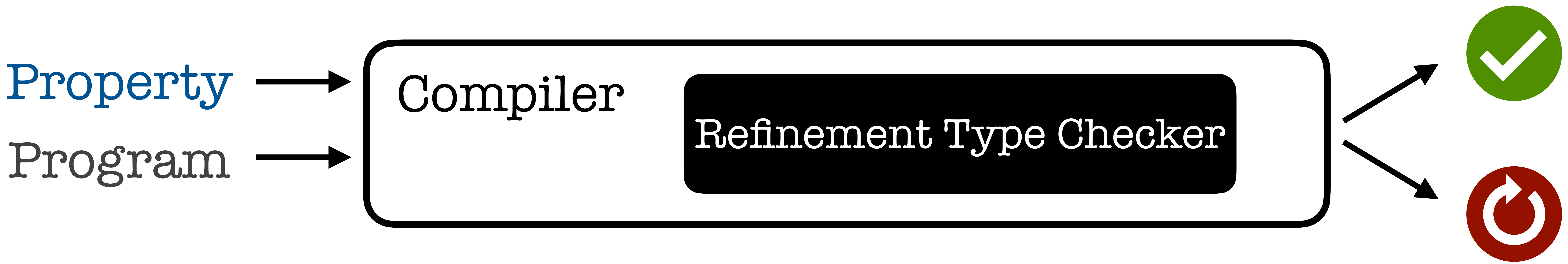
Type-based

Case sensitive

SMT-automated

Language Integrated

Language Integration

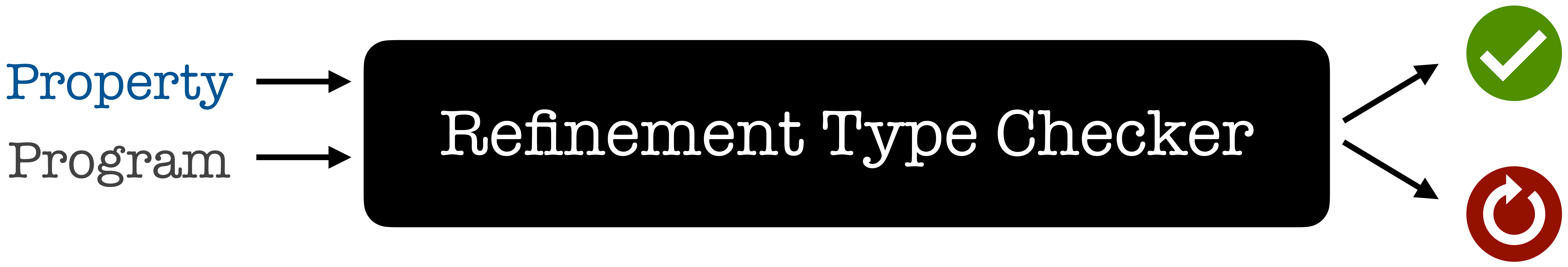


Refinement Type Checker is part of the compiler!

Editor fly-check integration;
Checking is part of the project build;
Cloud testing support; etc

Liquid Haskell as a GHC plugin,
by Di Napoli, Jhala, Löh, and Vazou. HIW'20.

Refinement Types are



Decidable

Type-based

Case sensitive

SMT-automated

Language Integrated

Refinement Types are

✓ Practical

Refinement Types are

✓ Practical

Expressive

General

Sound

Refinement Types are

✓ Practical

Expressive

General

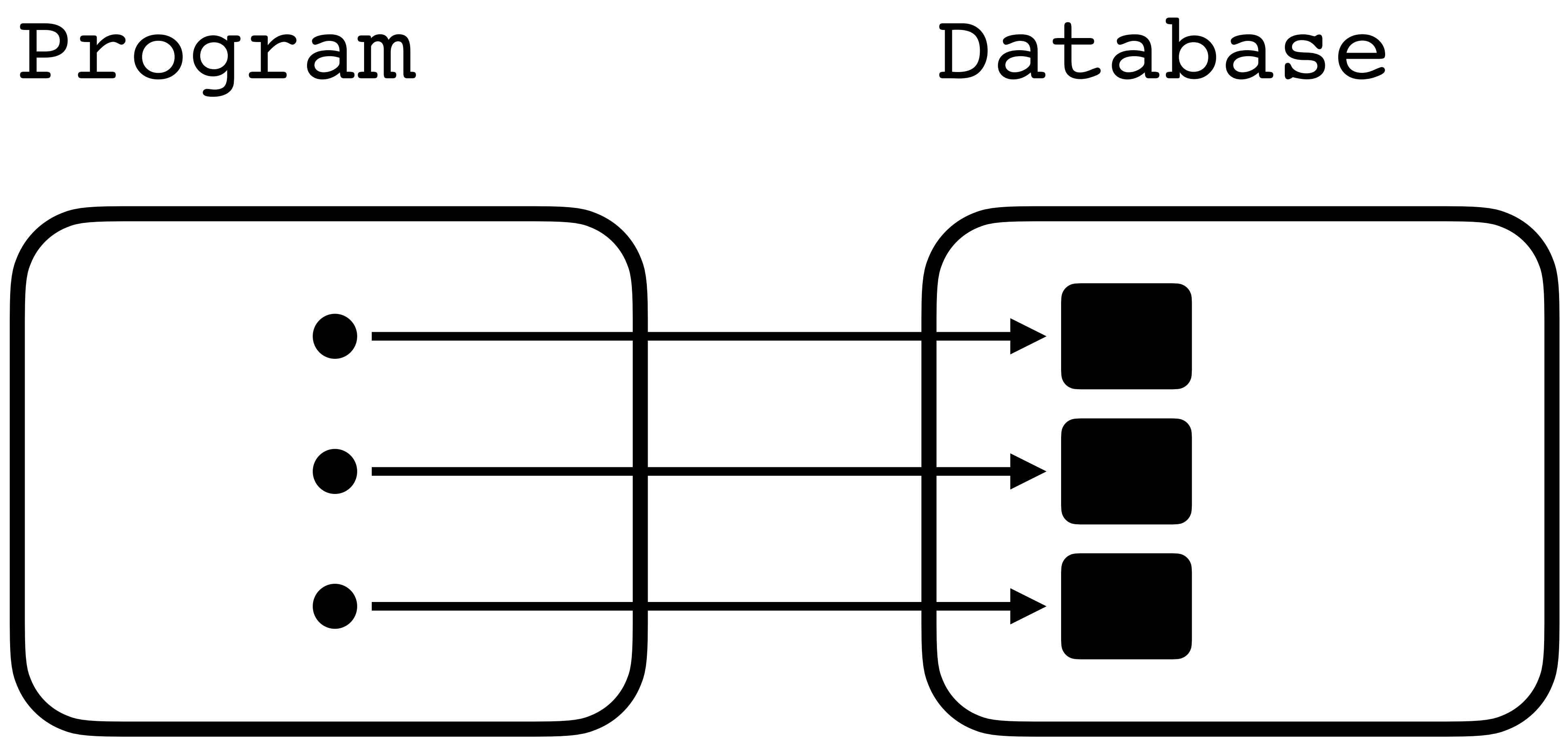
Sound

What can be expressed?

For Decidability: Logic is only SMT decidable theories
e.g., linear arithmetic, uninterpreted functions, data types, etc

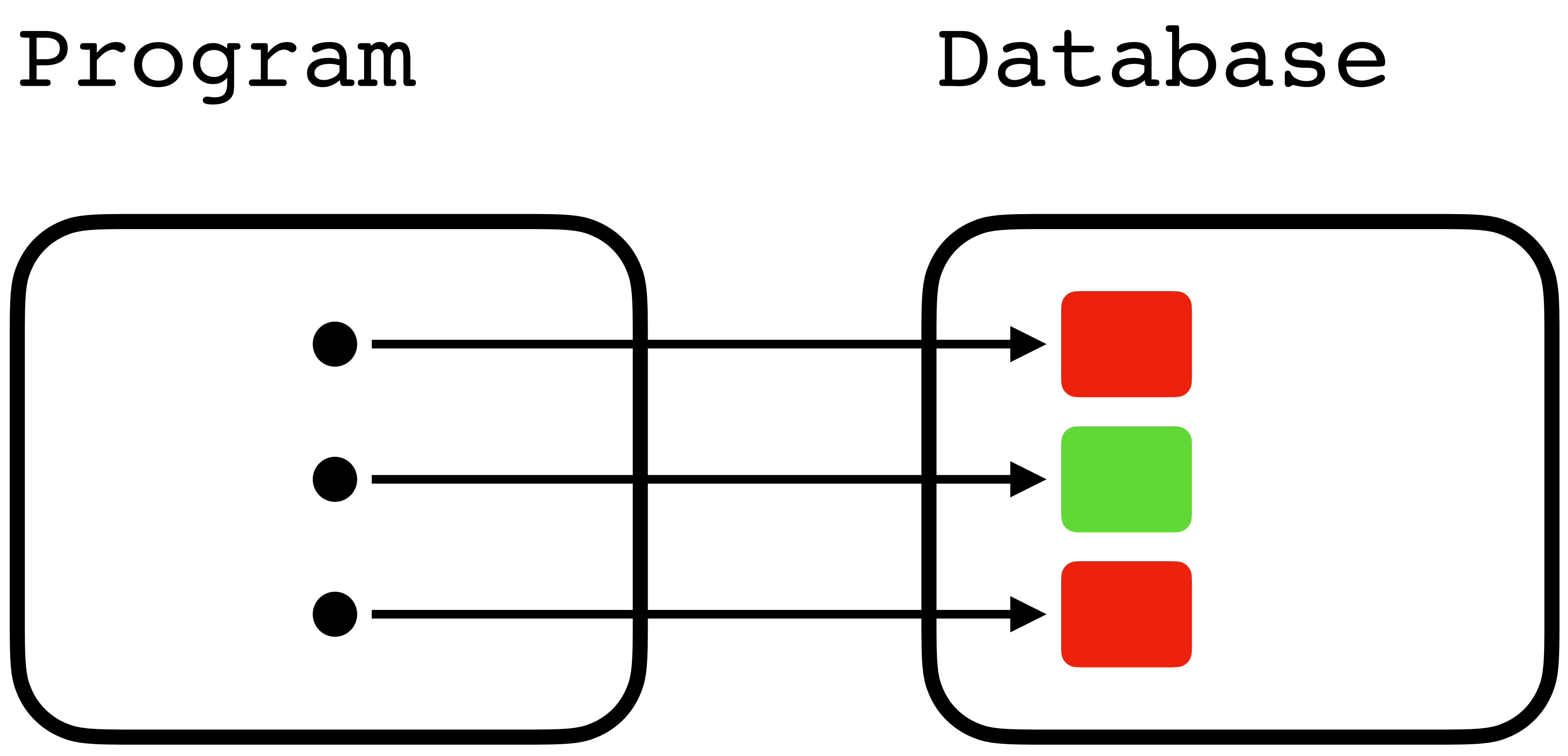
Direct to express: sortedness, safe-indexing
Also expressable: Domain-specific properties

Secure Web Applications



Programs manipulate data

Secure Web Applications



`l:Label = Secret | Public`

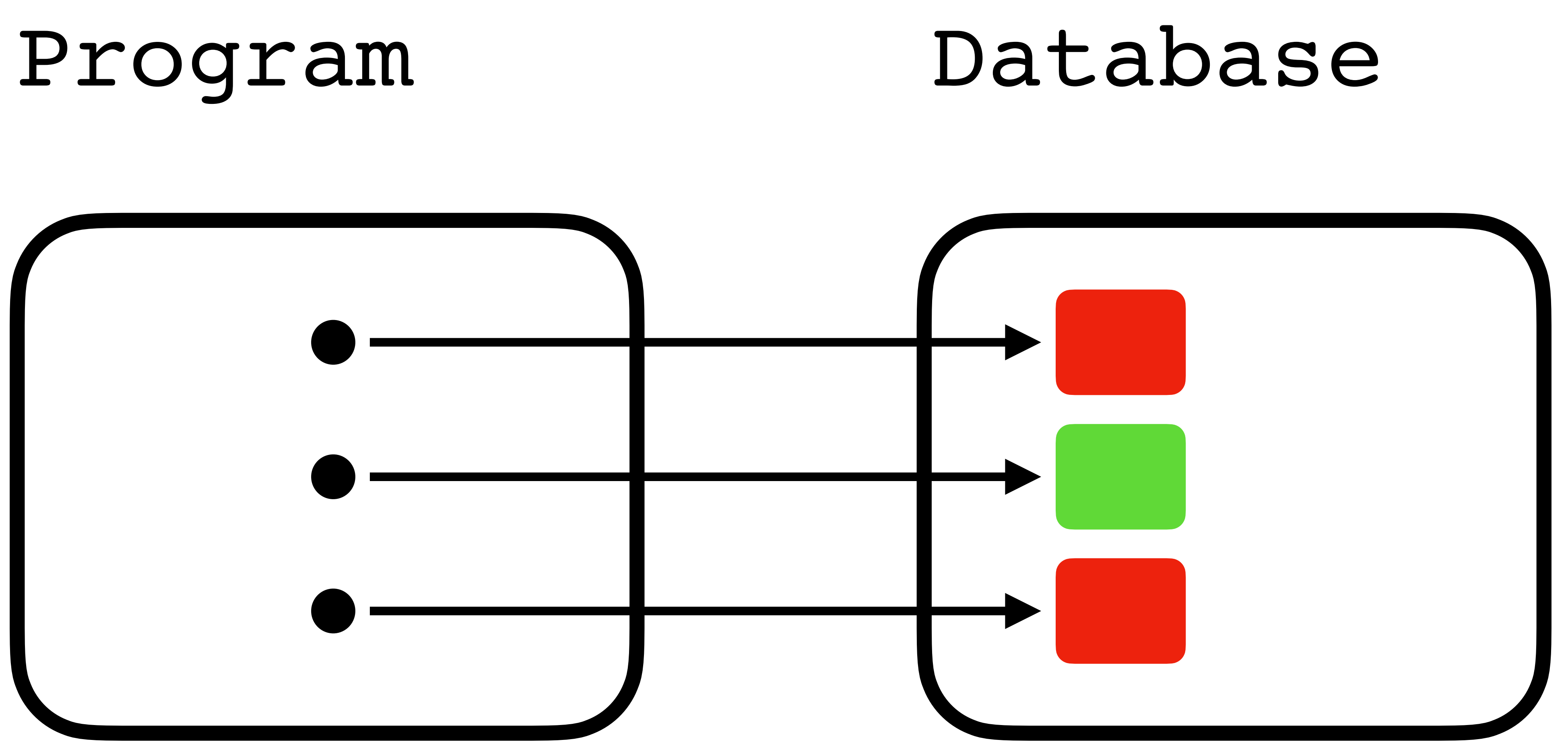
Programs manipulate data

Data are protected with policies

Noninterference:
Different labels cannot interfere

e.g., public programs cannot depend on secret data

Secure Web Applications



`l:Label = Secret | Public`

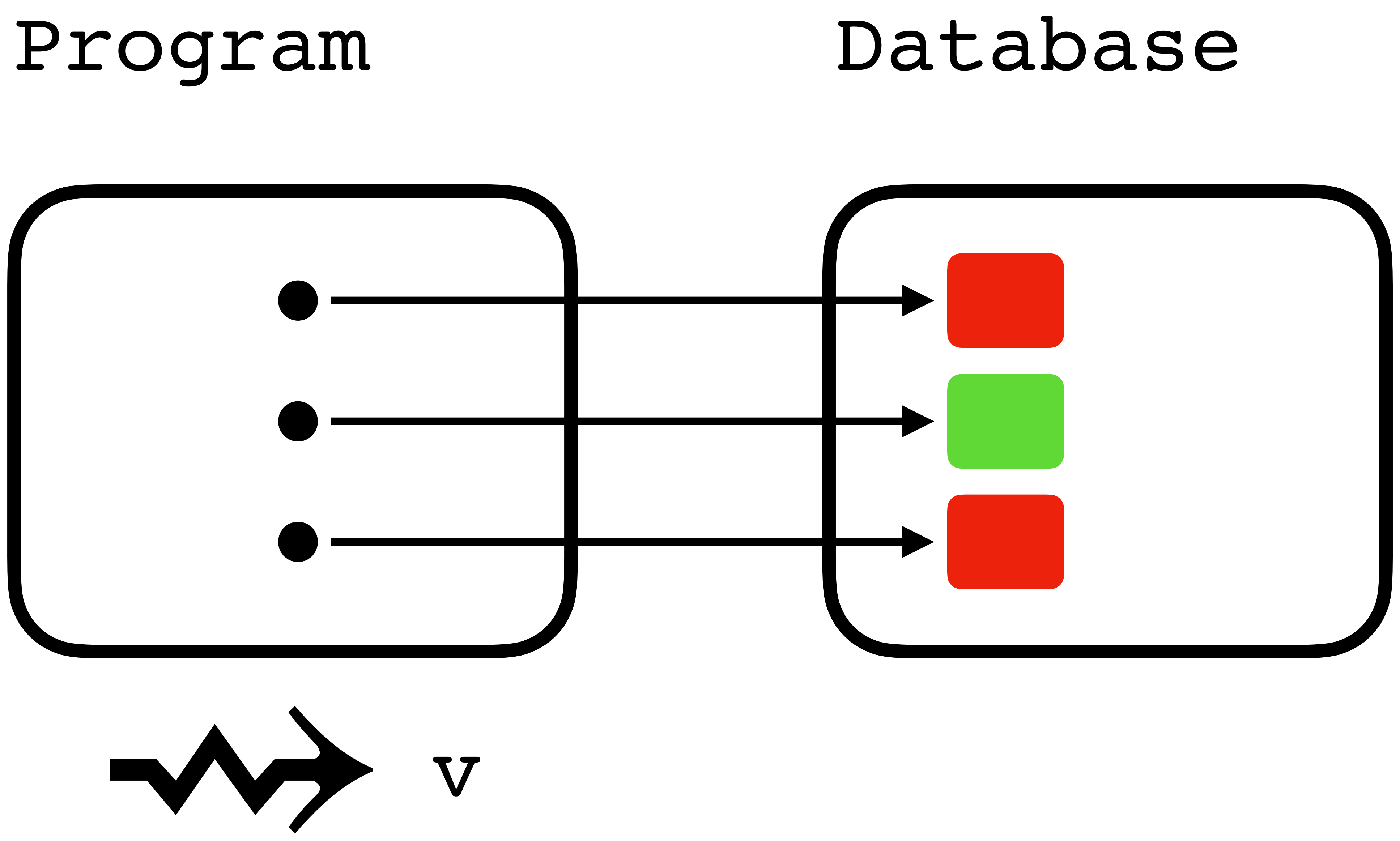
LWeb

Runtime checks to ensure noninterference

Refinement Types to prove noninterference of LWeb

LWeb: Information flow security for multi-tier web applications, by Parker, Vazou, and Hicks. POPL'19.

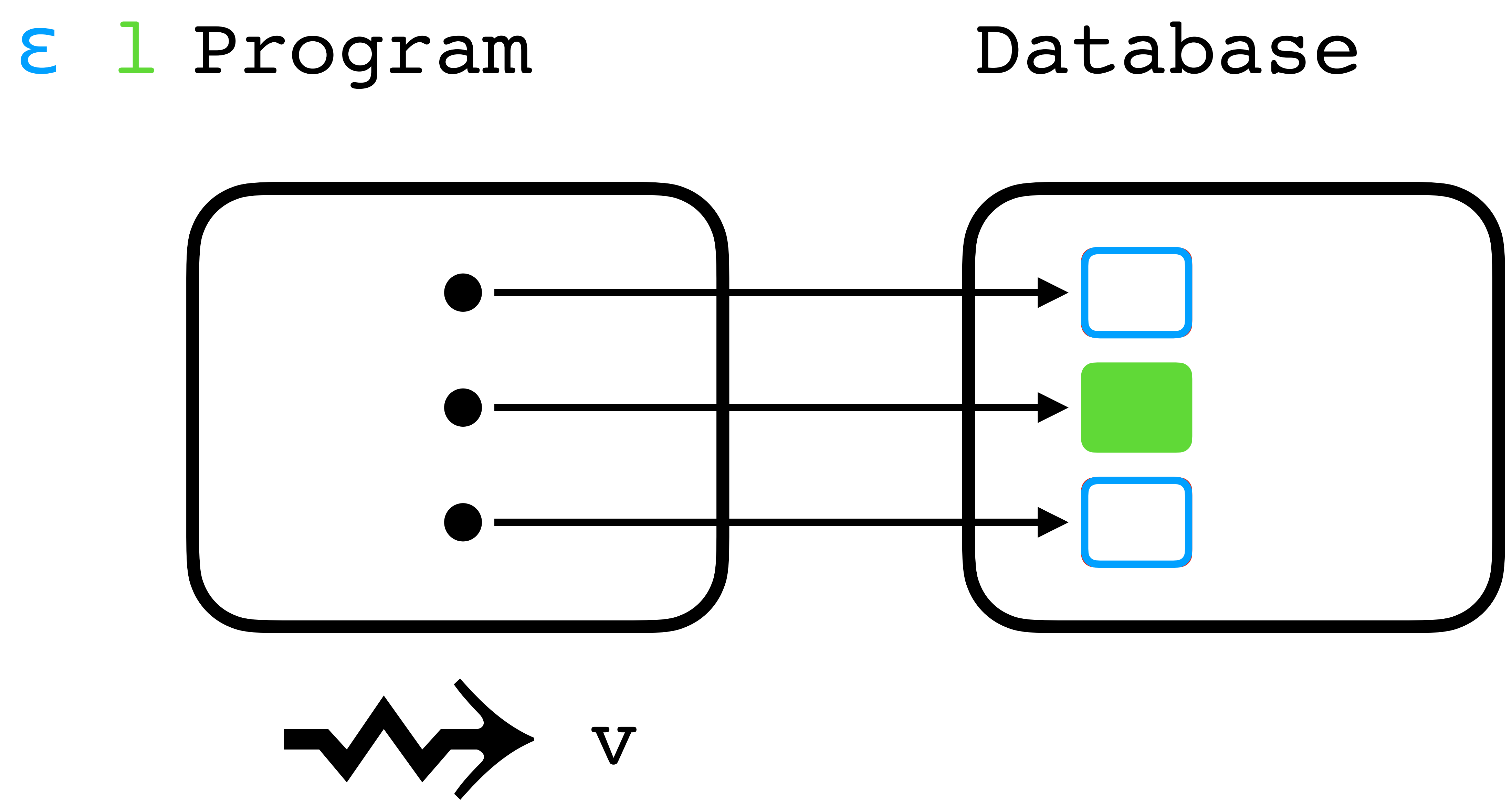
Noninterference



Execution preserves low view:

If we erase all the protected data, then execution remains the same.

Noninterference



Execution preserves low view:

If we erase all the protected data, then execution remains the same.

```
nonInterference :: l:Label → p1:Program → p2:Program
                → { ε l p1 == ε l p2 }
                → { ε l (exec p1) == ε l (exec p2) }
```

Secure Web Applications

LWeb

A framework that enforces label-based, **dynamic**, information flow policies in web applications.

Non interference proved **on a model** with refinement types.

First major proof in Liquid Haskell 5.5K LoC and revealed two design bugs.

Secure Web Applications

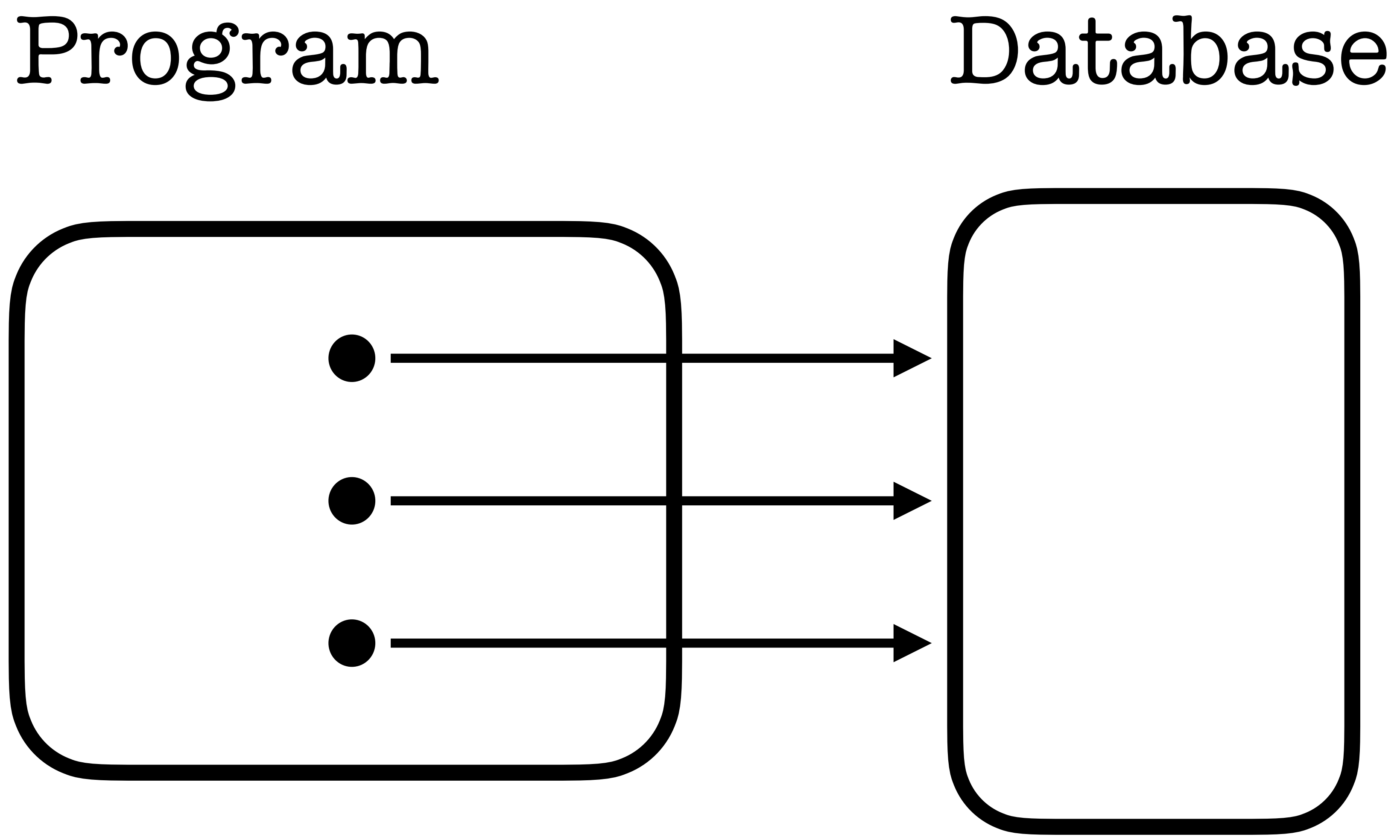
LWeb

STORM

A framework that enforces semantic, **static**, security policies in web applications.
Refinement types ensure policy enforcement at compile time **on executable code**.

STORM: Refinement Types for Secure Web Applications,
by Lehmann, Kunkel, Brown, Yang, Vazou, Polikarpova , Stefan , and Jhala. OSDI'21.

STORM



\+ ORM (Object Relational Mapping)

```
Review
   paper
   reviewer
   content
   score

  Policy v:Viewer r:Review
   isPC v || isAuthor v (getPaper r)
```

STORM secure operators refined to check the policies at refinement type checking

ORM comes with semantic policies

Secure Web Applications

LWeb

STORM

Semantic security policies are checked statically.

Used to develop 3 web applications (~1K Loc each)
- conference management system
- student collaboration system
- video conferencing system

Secure Web Applications

LWeb

STORM

Anosy

Refinement types + abstract domains to track sensitive information leakage.

Anosy: Approximate Knowledge Synthesis with Refinement Types,
by Guria, Parker, Guarnieri, and Vazou. PLDI'22.

Domain Specific Properties

Secure Web Applications

Resource Usage

Distributed Applications

Liquidate your asserts, by Handley, Vazou, and Hutton. POPL'20

Verifying Replicated Data Types, by Liu, Parker, Kuper, Hicks, and Vazou. OOPSLA'20

What can be expressed?

Decidable Properties

Safe-indexing

Domain Specific Properties

Secure Web Applications

Resource Usage

Distributed Applications

Refinement Types are

- ✓ Practical
- ✓ Expressive
- General
- Sound

Refinement Types are

- ✓ Practical
- ✓ Expressive
- General
- Sound

Are Refinement Types General?

$$\text{ML} \xrightarrow{\text{laziness}} \text{Haskell} \xrightarrow[\text{programming}]{\text{meta}} \text{Ruby}$$

Refinement Types for Haskell, by Vazou, Seidel, Jhala, Vytiniotis, Peyton-Jones, ICFP'14.

Type-level computations for Ruby libraries,
by Kazerounian, Guria, Vazou, Foster, Van Horn, PLDI'19.

Are Refinement Types General?

$$\begin{tikzcd}
 & & \text{Ruby} \\
\text{ML} \arrow[r, "\text{laziness}"] & \text{Haskell} \arrow[ur, "\text{meta}", "\text{programming}"'] \arrow[r, "\text{memory}", "\text{management}"'] & \text{Rust}
\end{tikzcd}$$

Flux: Refinement Types for Rust, by Lehmann, Jhala, Vazou.

https://github.com/liquid-rust/flux

Are Refinement Types General?

$$\mathrm{ML} \xrightarrow{\text{laziness}} \mathrm{Haskell}$$

$$\mathrm{Haskell} \xrightarrow[\text{programming}]{\text{meta}} \mathrm{Ruby}$$

$$\mathrm{Haskell} \xrightarrow[\text{mutation}]{\text{state}} \mathrm{Java}$$

$$\mathrm{Haskell} \xrightarrow[\text{management}]{\text{memory}} \mathrm{Rust}$$

LiquidJava, by Gamboa, Santos, Timperley, and Fonseca.

Are Refinement Types General?

$$\text{ML} \xrightarrow{\text{laziness}} \text{Haskell}$$

$$\text{Haskell} \xrightarrow[\text{programming}]{\text{meta}} \text{Ruby}$$

$$\text{Haskell} \xrightarrow[\text{mutation}]{\text{state}} \text{Java}$$

$$\text{Haskell} \xrightarrow[\text{management}]{\text{memory}} \text{Rust}$$

The principles are general,
but should be adjusted to each language.

Refinement Types: A tutorial, by Jhala and Vazou,
Foundations and Trends in Programming Languages'21.

Refinement Types

- ✓ Practical
- ✓ Expressive
- ✓ General
- Sound

Refinement Types

- ✓ Practical
- ✓ Expressive
- ✓ General
- Sound

Are Refinement Types Sound?

A type system is sound when
it only accepts programs that cannot get stuck.

Soundness: If $\vdash e_o : t$ and $e_o \hookrightarrow^* e$,
then either e is a value or $e \hookrightarrow e_i$.

Soundness of Refinement Types

```
{-@ soundness :: e0:Expr -> t:Type -> Prop (HasType Empty e0 t)
              -> e:Expr -> Prop (EvalsTo e0 e)
              -> Either { isValue e } (ei::Expr, Prop (Step e ei)) @-}
```

Soundness: If $\vdash e_o : t$ and $e_o \hookrightarrow^* e$, then either e is a value or $e \hookrightarrow e_i$.

RT²: Mechanizing Refinement Types with Refinement Types, by Borkowski, Vazou, and Jhala (under review).

Soundness of Refinement Types

```
{-@ soundness :: e0:Expr -> t:Type -> Prop (HasType Empty e0 t)
              -> e:Expr -> Prop (EvalsTo e0 e)
              -> Either { isValue e } (ei::Expr, Prop (Step e ei)) @-}
```

HasType **models** refinement type checking.

First polymorphic refinement type formalism

Proved in 19K lines of (Liquid) Haskell.

RT²: Mechanizing Refinement Types with Refinement Types,
by Borkowski, Vazou, and Jhala (under review).

Soundness of Refinement Types

```
{-@ soundness :: e0:Expr -> t:Type -> Prop (HasType Empty e0 t)
              -> e:Expr -> Prop (EvalsTo e0 e)
              -> Either { isValue e } (ei::Expr, Prop (Step e ei)) @-}
```

```
soundness _e0 t e0_has_t _e e0_evals_e = case e0_evals_e of
  Refl e0 → progress e0 t e0_has_t -- e0 = e
  AddStep e0 e1 e0_step_e1 e e1_eval_e → -- e0 → e1 →* e
   soundness e1 t (preservation e0 t e0_has_t e1 e0_step_e1) e e1_eval_e
```

RT²: Mechanizing Refinement Types with Refinement Types,
by Borkowski, Vazou, and Jhala (under review).

Refinement Types

- ✓ Practical
- ✓ Expressive
- ✓ General
- ✓ **Sound**

Unsoundness of Refinement Types

Soundness is proved in a model refinement type checker.

>**5** unsoundness reports*/year in Liquid Haskell

* An example of a program being accepted while it violated the property

Unsoundness of Refinement Types

Soundness is proved in a model refinement type checker.

>**5** unsoundness reports /year in Liquid Haskell

Too many in comparison to sound verifiers:

<**1** unsoundness reports/year in Coq.

*Coq: type-theory based theorem prover, designed to be sound

Liquid Haskell is not Sound

$\textbf{LH} \xrightarrow[\text{423K LoC}]{\text{Implements}} \text{CORE} + \begin{array}{l}\text{Data Types}\\ \text{Inference}\\ \text{Coercions}\end{array}$

Huge Code Base
(error prone)

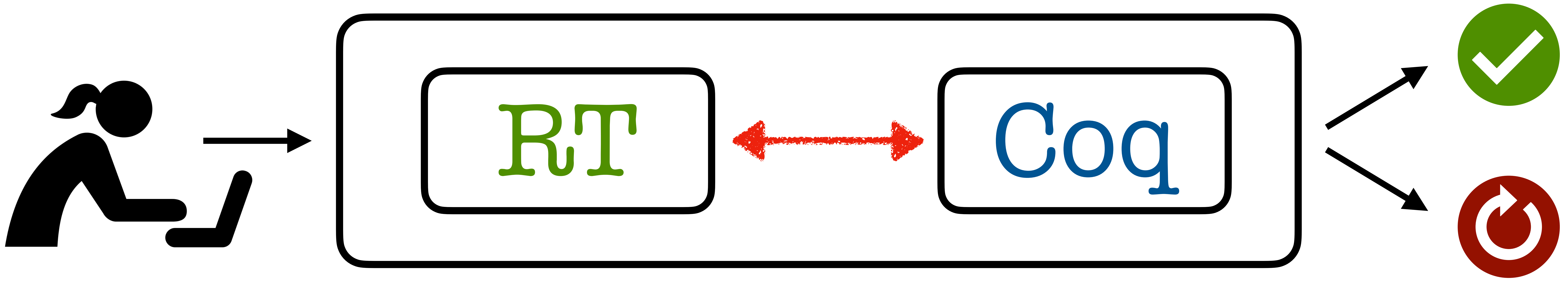
Formalized
model of LH

Non Formalized

Coq is Designed to Be Sound

$$
\begin{tikzcd}
\text{Coq} \arrow[loop left] \arrow[r, "\text{Implements}", "\text{15K LoC}"'] & \text{CIC} \arrow[r, "\text{Reduces}"] & \text{Set Theory}
\end{tikzcd}
$$

GOAL: Make Refinement Types Sound



Intuition:

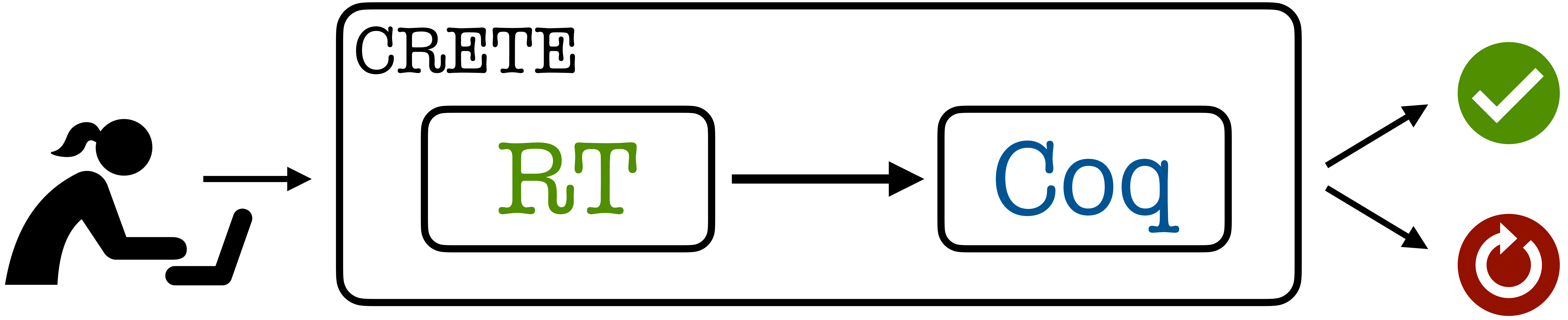
Get the final result by Coq!

Problem:

The gap between RT and Coq is too big

CRETE: Certified Refinement Types,
Vazou, ERC Starting 2021.

GOAL: Make Refinement Types Sound



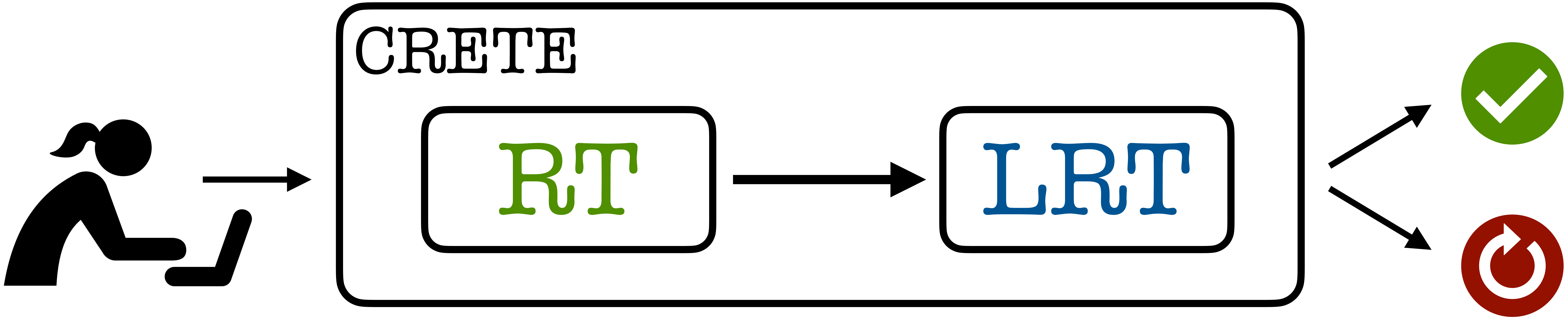
Goal I: RT → Coq

Problem:

The translation is not always be possible

CRETE: Certified Refinement Types,
Vazou, ERC Starting 2021.

GOAL: Make Refinement Types Sound



Goal I: RT → Coq

Goal II: Logic of Refinement Types (LRT)

Impact

Practical: Sound Implementations

Scientific: Set Foundations of Refinement Types

Refinement Types

- ✓ Practical
- ✓ Expressive
- ✓ General
- ✗ Sound

Thanks!