Fissile Type Analysis: Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

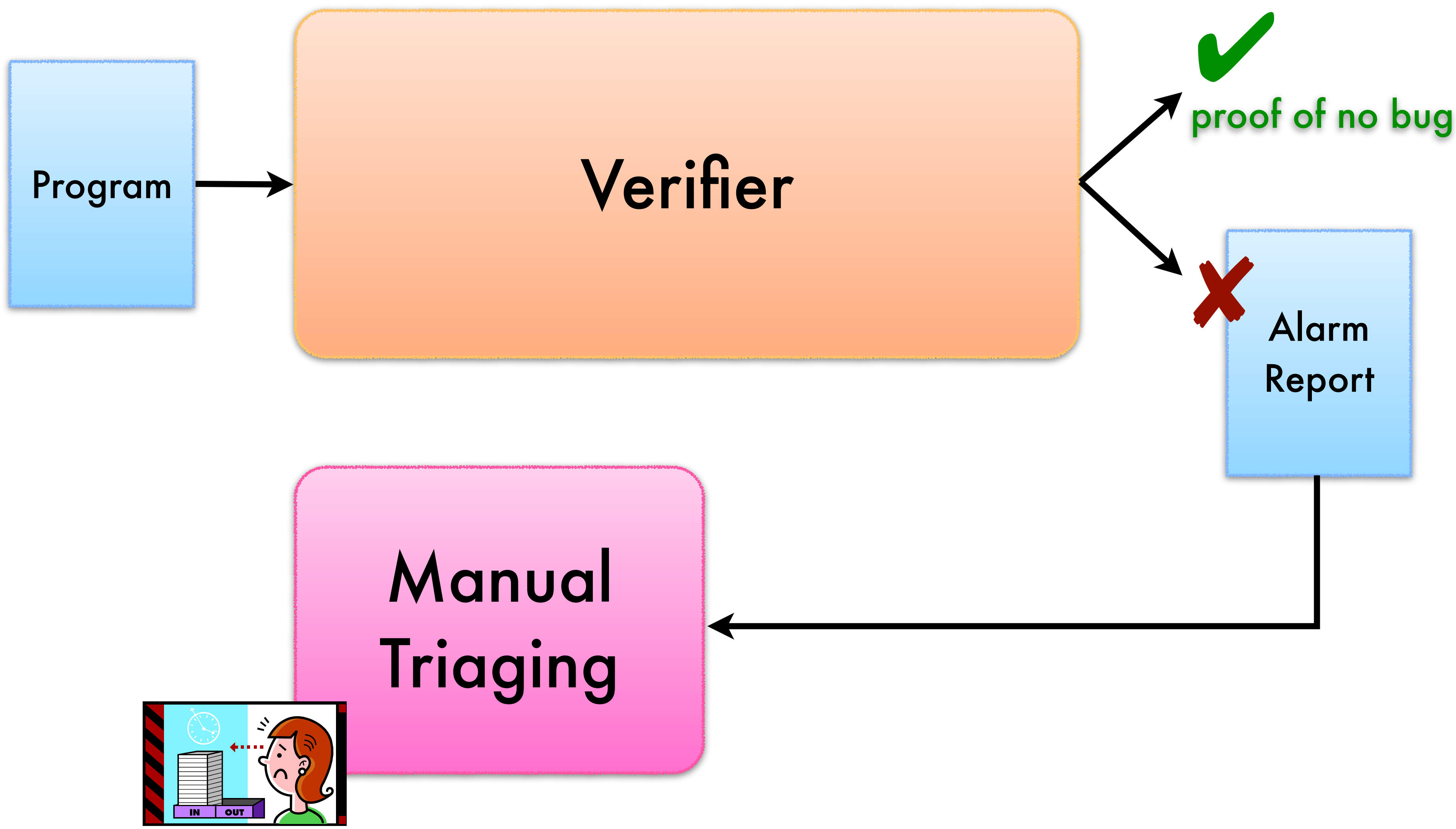
Bor-Yuh Evan Chang
University of Colorado Boulder

University of Maryland, College Park
February 26, 2014

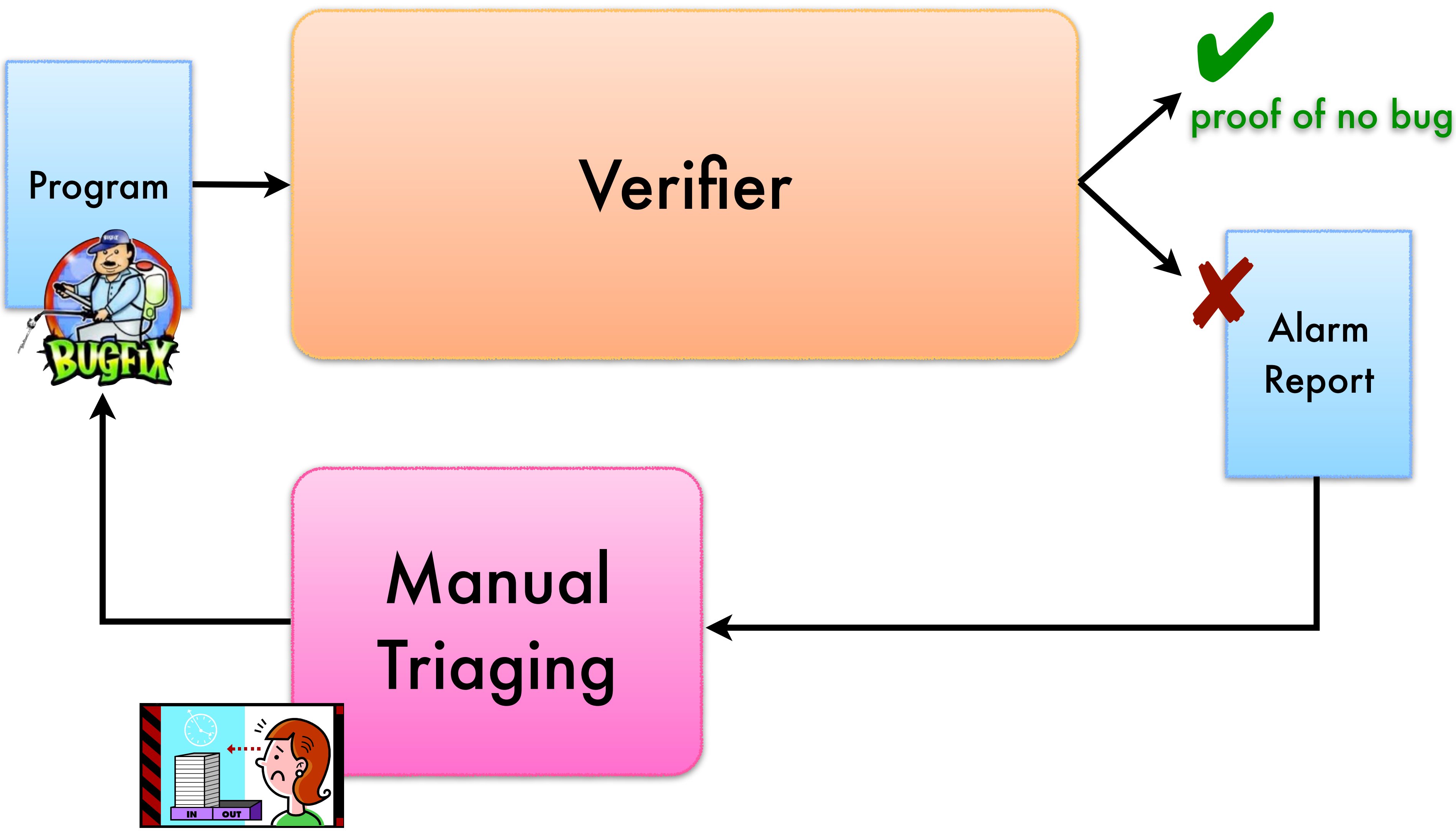
Lab: **Program analysis** in the **whole** bug mitigation **process**

Lab: **Program analysis** in the **whole** bug mitigation **process**

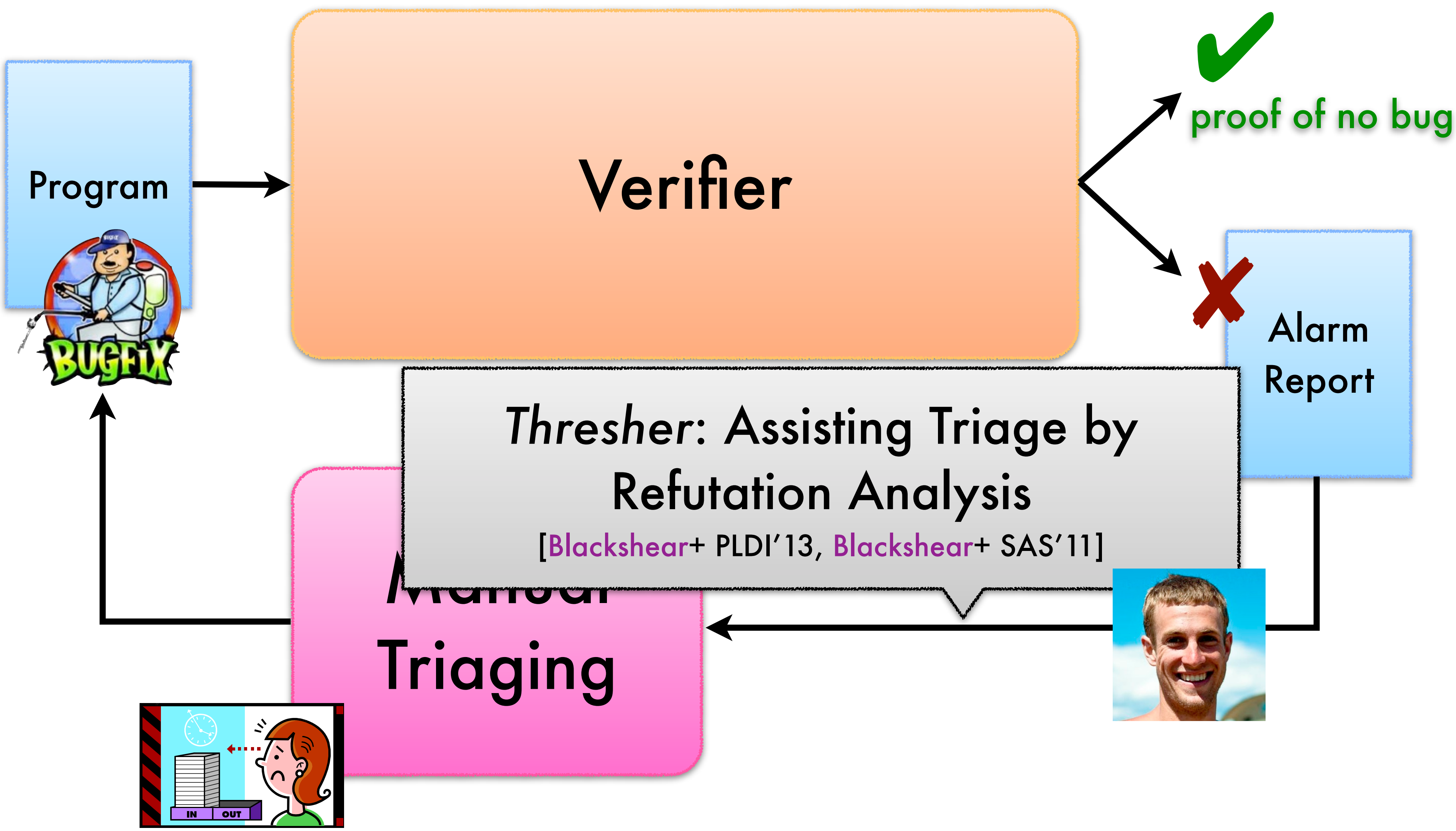
Lab: **Program analysis** in the **whole** bug mitigation **process**



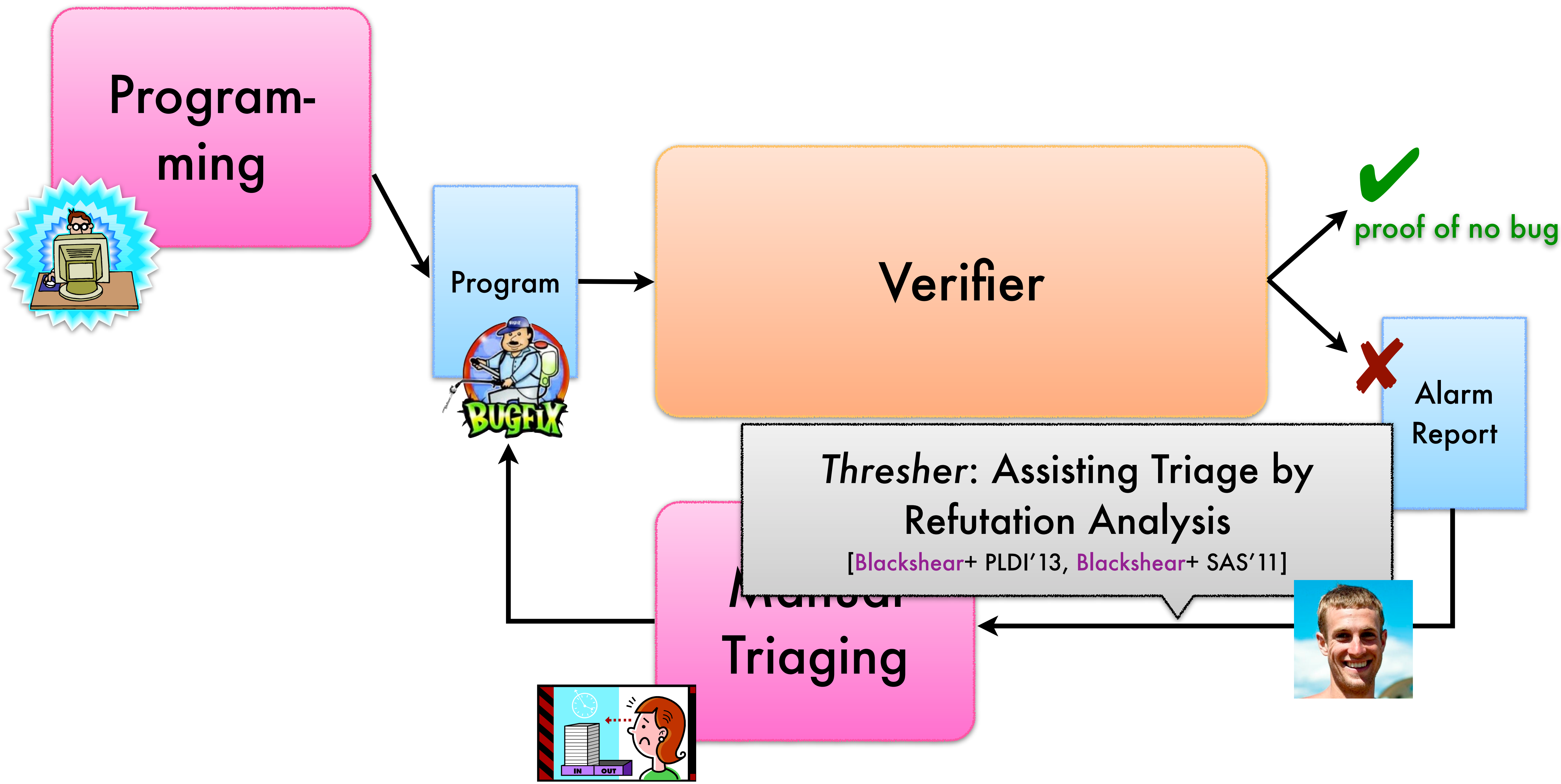
Lab: **Program analysis** in the **whole** bug mitigation **process**



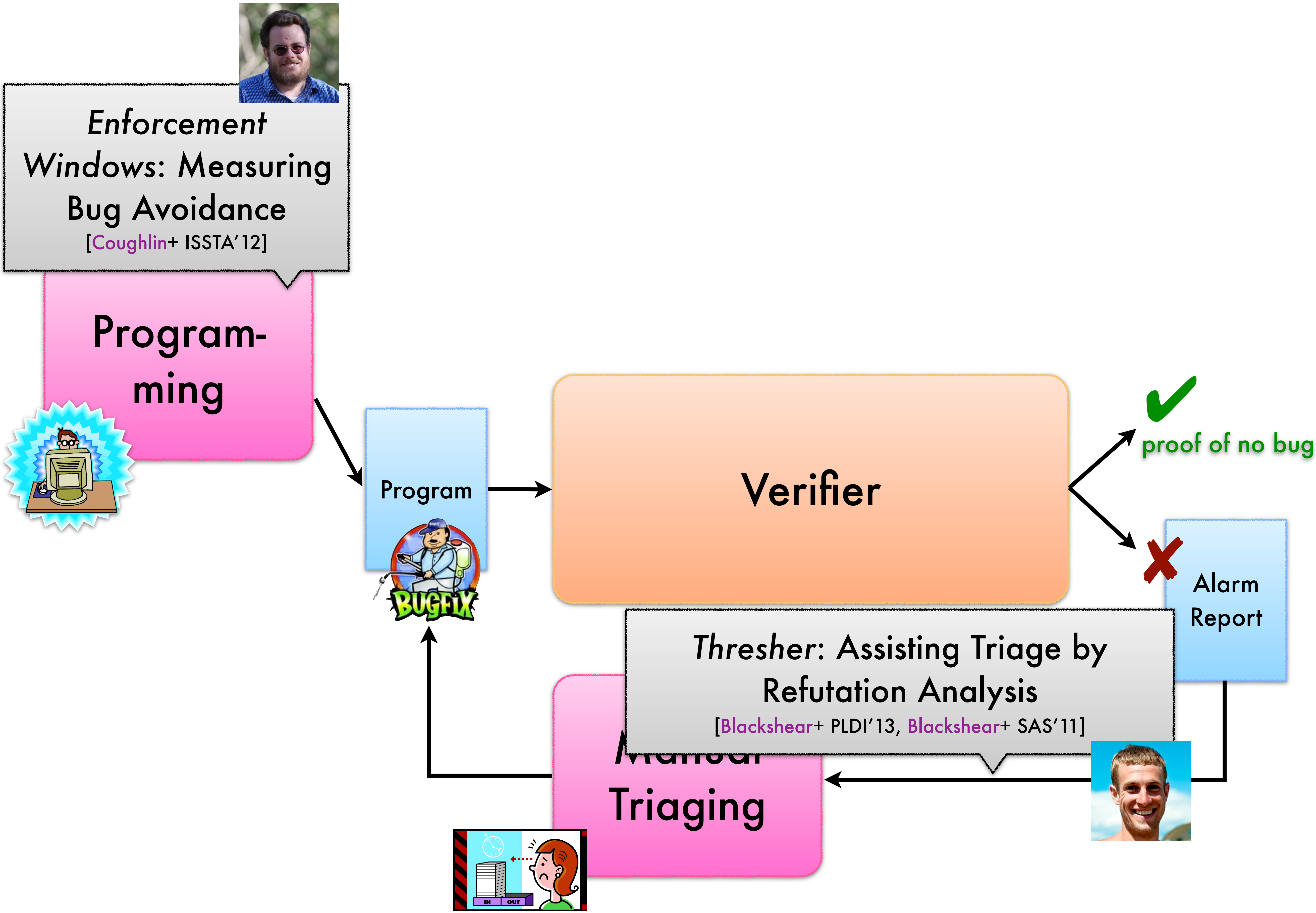
Lab: **Program analysis** in the **whole** bug mitigation **process**



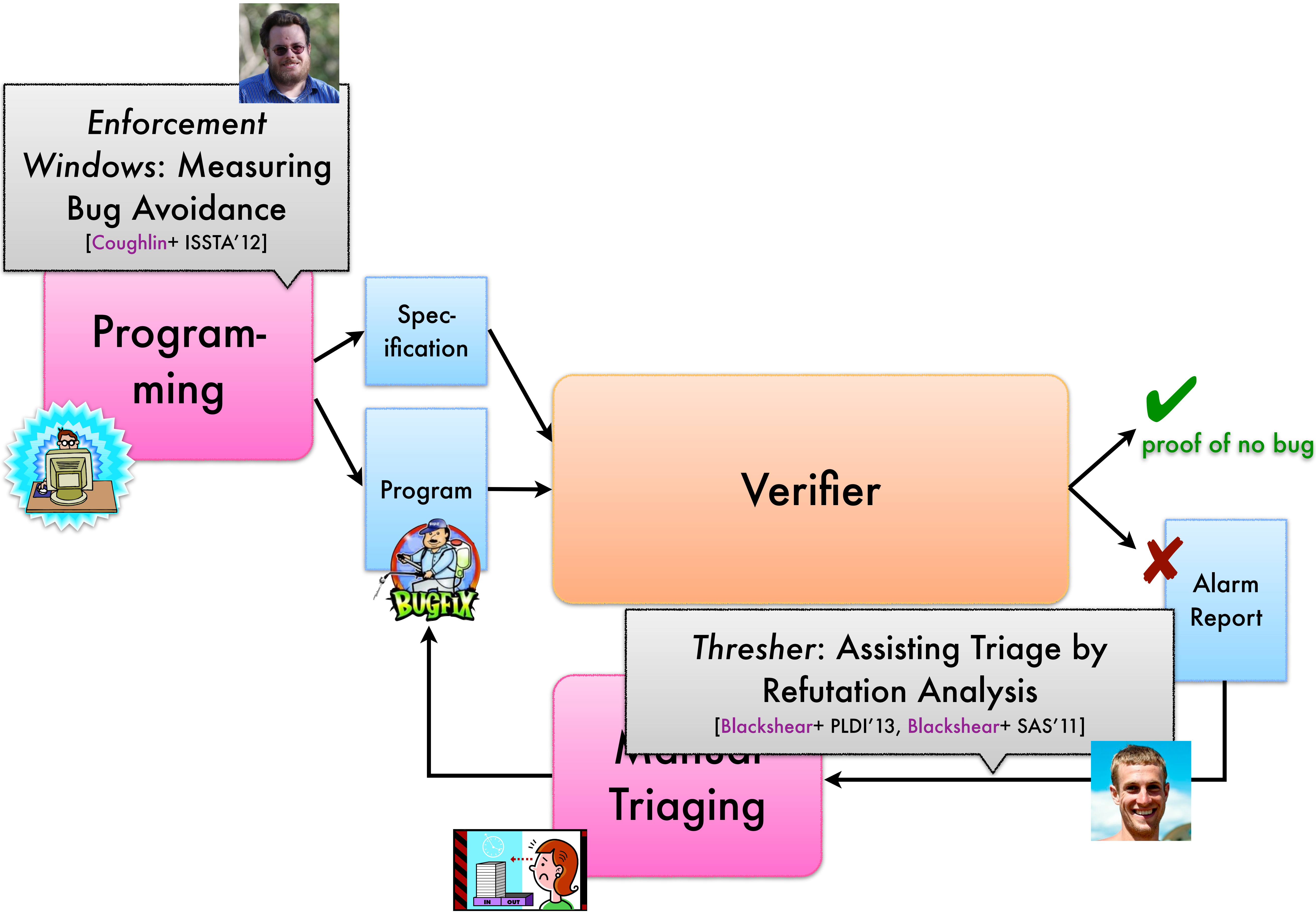
Lab: **Program analysis** in the **whole** bug mitigation **process**



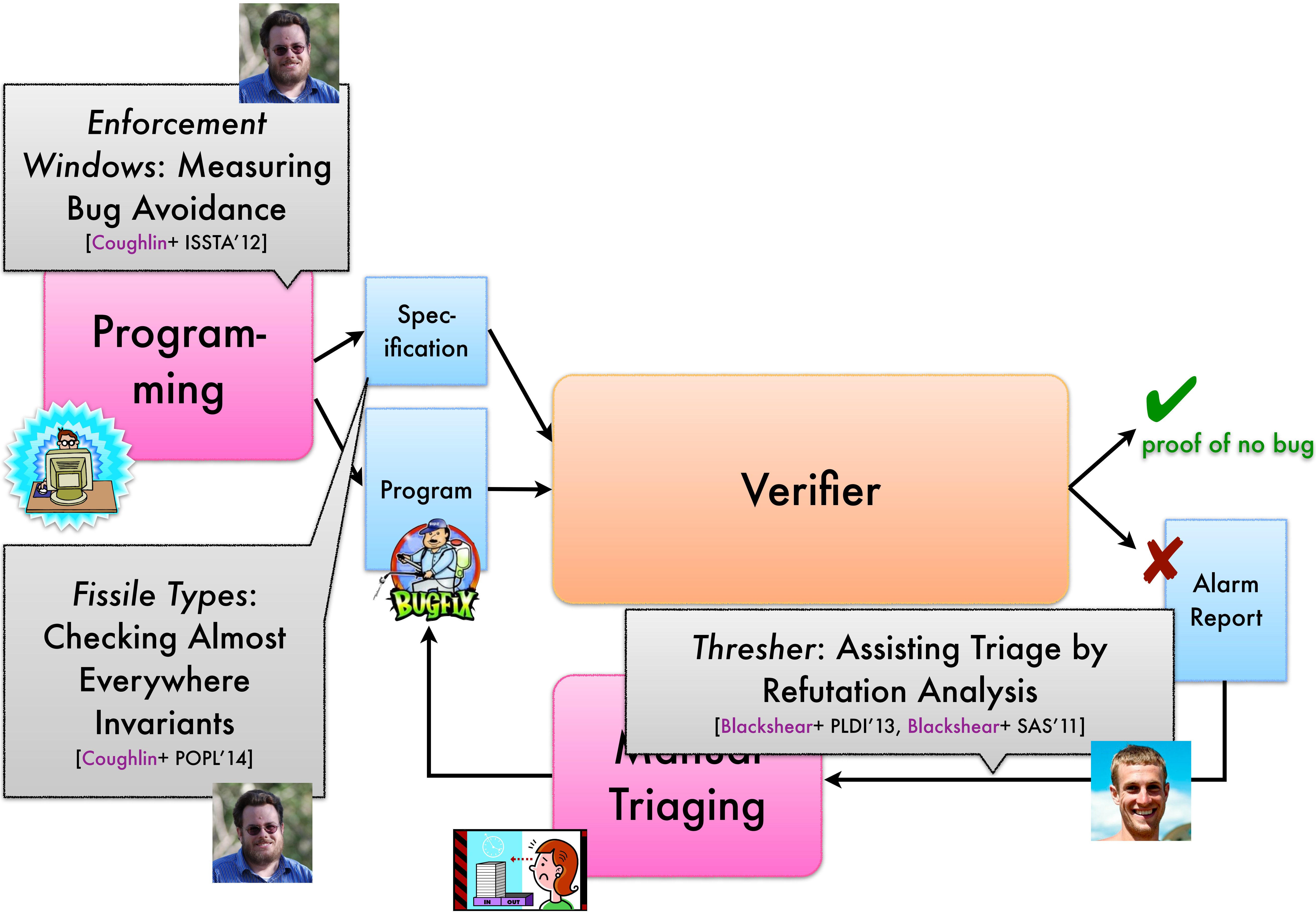
Lab: **Program analysis** in the **whole** bug mitigation **process**



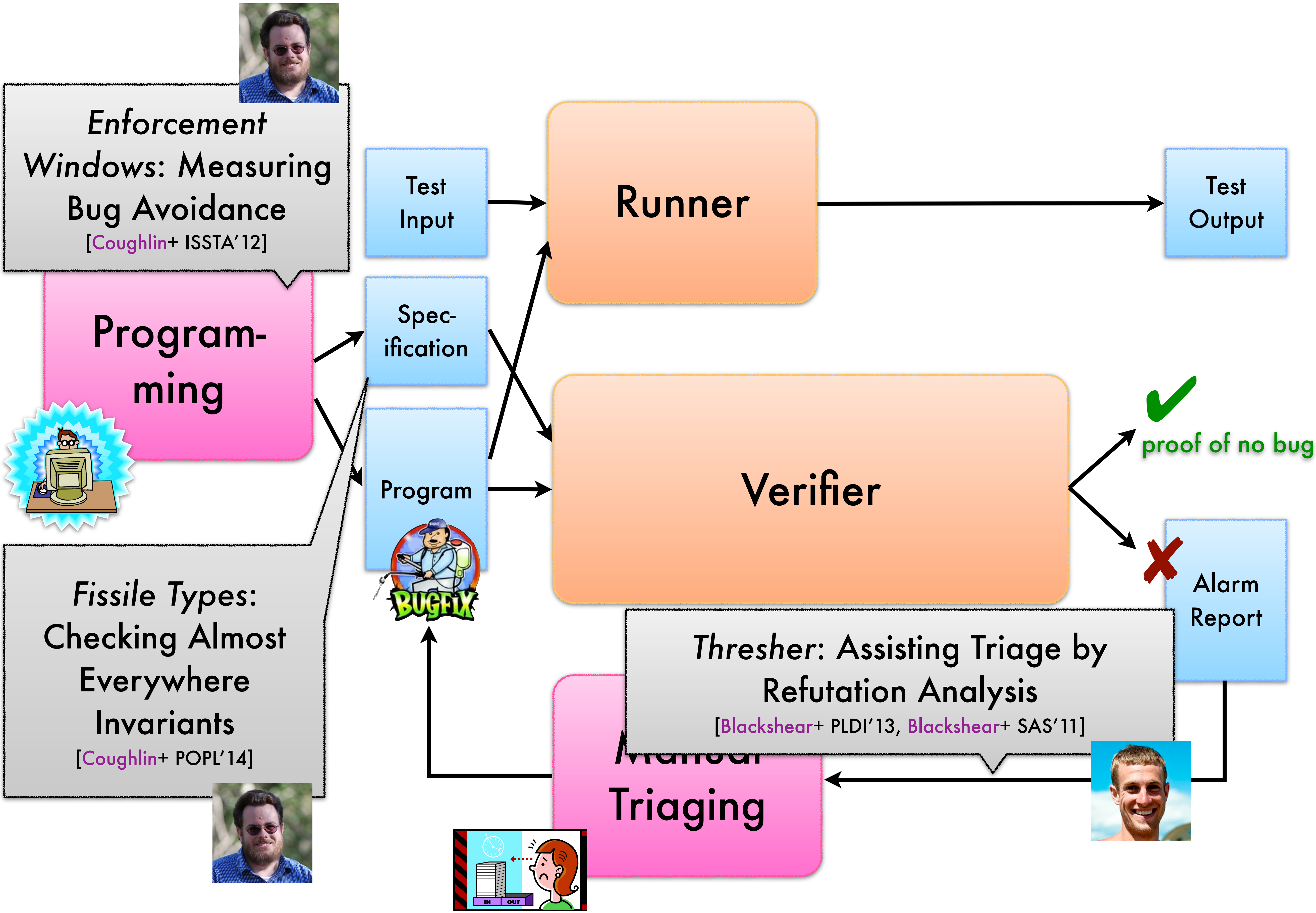
Lab: **Program analysis** in the **whole** bug mitigation **process**



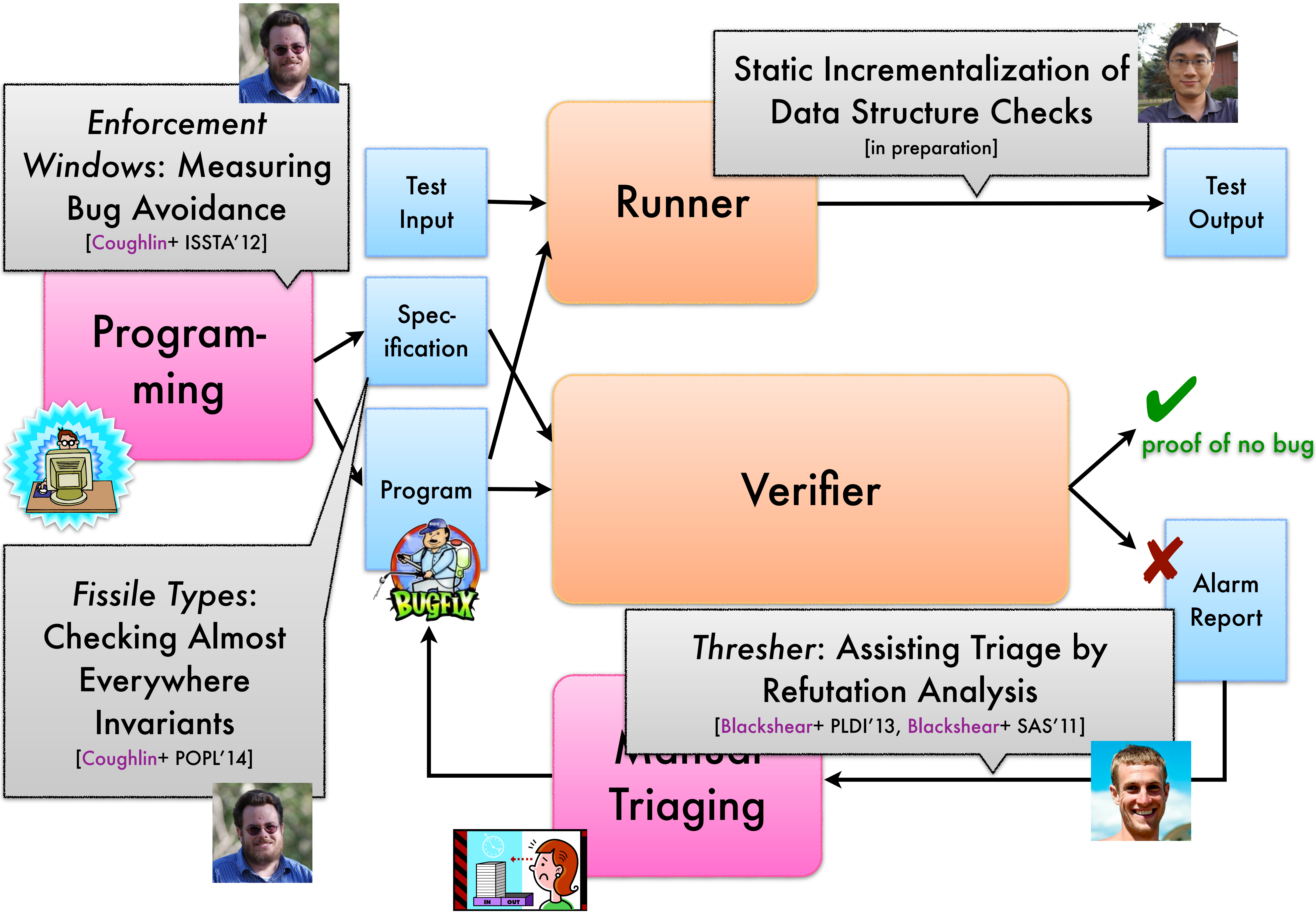
Lab: **Program analysis** in the **whole** bug mitigation **process**



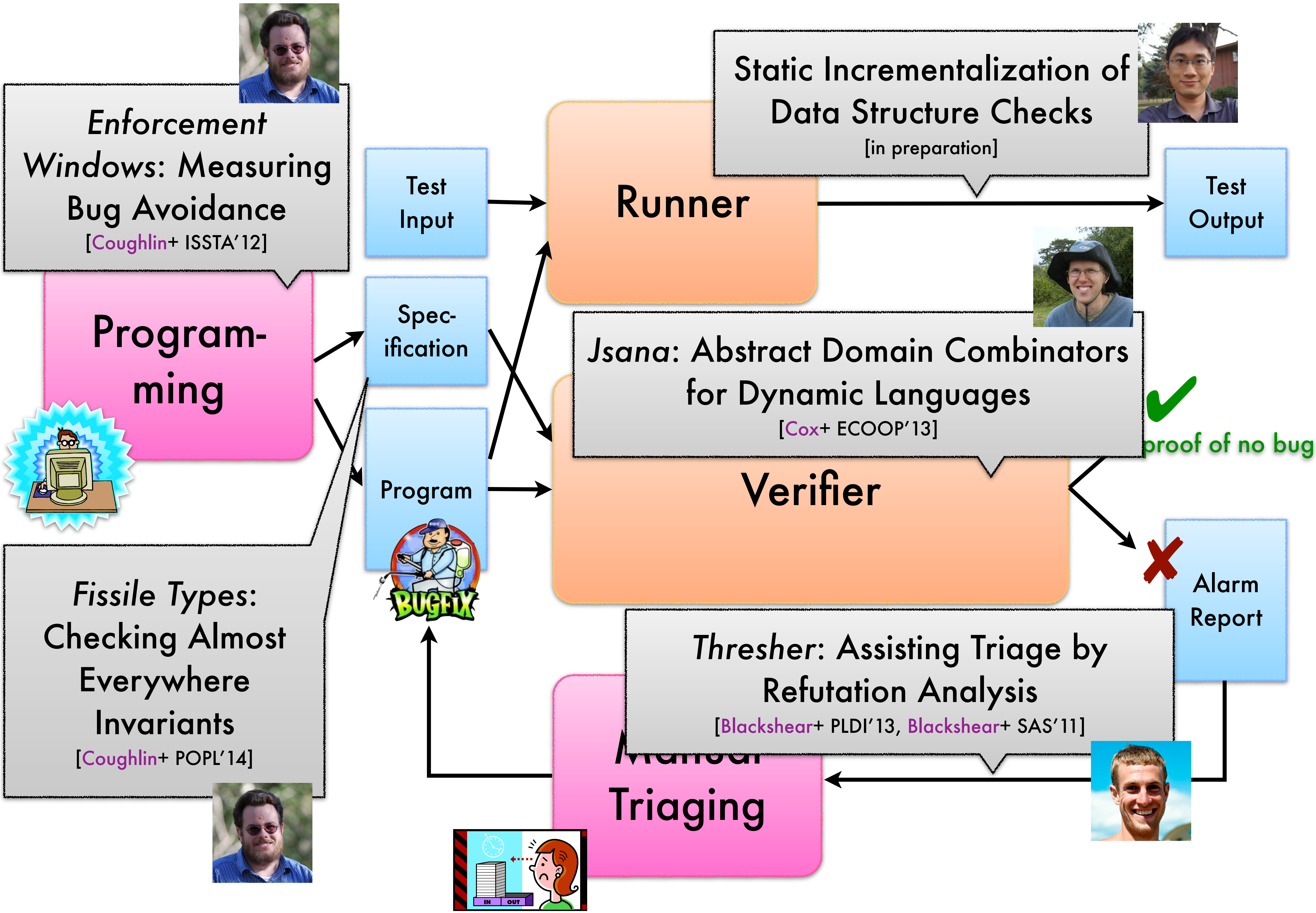
Lab: **Program analysis** in the **whole** bug mitigation **process**



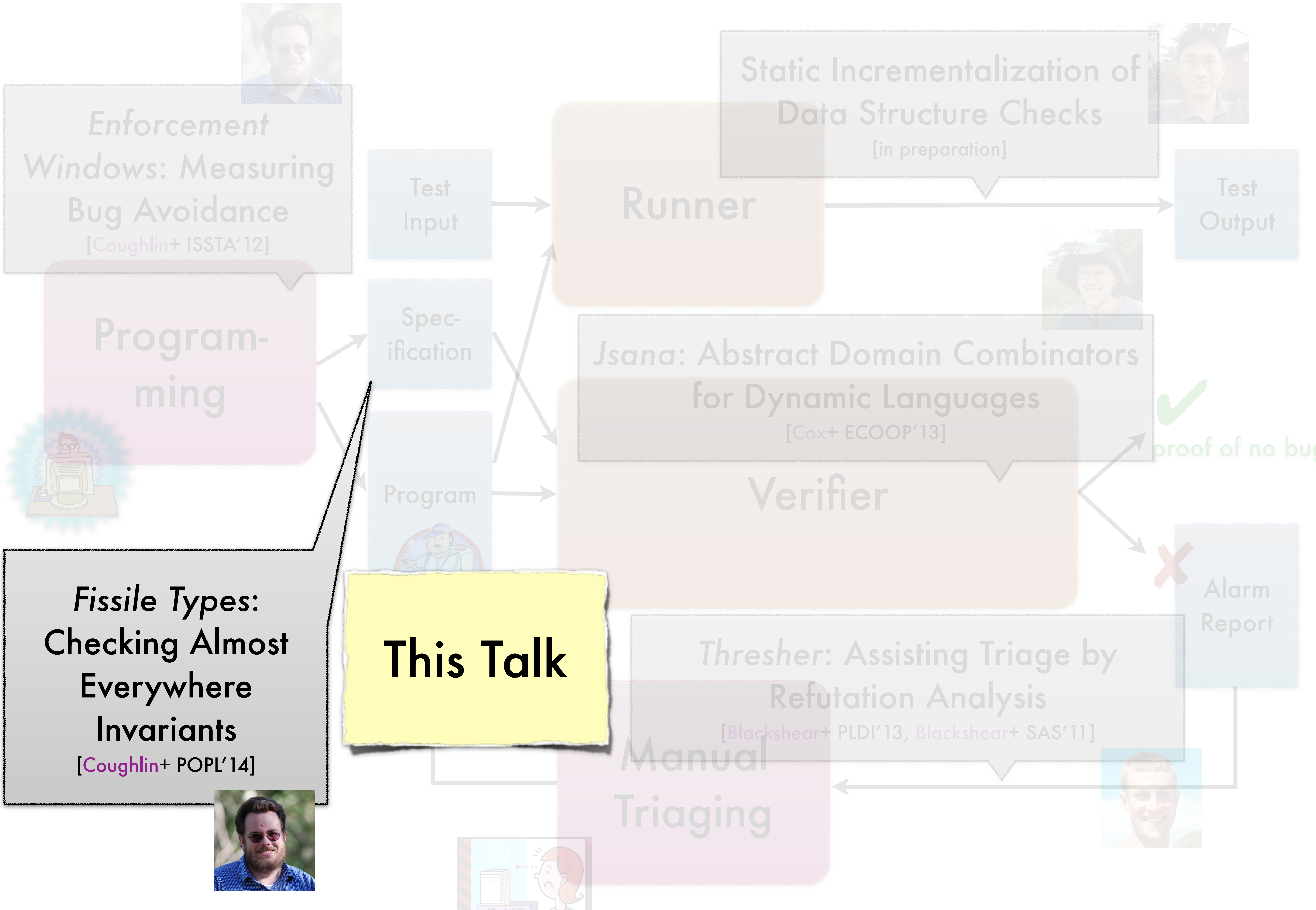
Lab: **Program analysis** in the **whole** bug mitigation **process**



Lab: **Program analysis** in the **whole** bug mitigation **process**



Lab: **Program analysis** in the **whole** bug mitigation **process**



Fissile Type Analysis: Modular Checking of Almost Everywhere Invariants

Devin Coughlin
University of Colorado Boulder

Bor-Yuh Evan Chang
University of Colorado Boulder

University of Maryland, College Park
February 26, 2014

How to **type check** a program that is **almost** well-typed?

In this talk

Example property of interest:
safety of **reflective method calls**

Type system:
dependent refinement types

Reflective method call dispatches based on **runtime string value**

```
class Callback
 var sel : Str
 var obj : Obj

 def call()
  this.obj.[this.sel]()
```

Reflective method call dispatches based on **runtime string value**

```
class Callback
 var sel : Str
 var obj : Obj

 def call()
  this.obj.[this.sel]()
```

Calls method with **name** (selector) stored in `sel` on object stored in `obj`

Reflective method call dispatches based on **runtime string value**

```
class Callback
  var sel : Str
  var obj : Obj

  def call()
    this.obj.[this.sel]()
```

Calls method with **name** (selector) stored in `sel` on object stored in `obj`

If `sel` held string "`notifyDidClick`" would call `notifyDidClick()` on `obj`.

Reflective method call dispatches based on **runtime string value**

```
class Callback
 var sel : Str
 var obj : Obj

 def call()
  this.obj.[this.sel]()
```

Calls method with **name** (selector) stored in **`sel`** on object stored in **`obj`**

Run time error if **`obj`** does not **respond to `sel`** — i.e., method does not exist

Method Reflection and **the Great Divide**

Method Reflection and **the Great Divide**

```
object.[string]()
```

Method Reflection and **the Great Divide**

reflective method call: dispatch based on **run-time value** (in string)

```
object.[string]()
```

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

```
object.[string]()
```

"static" folks

"web 2.0" developers

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

`object.[string]()`

"static" folks

"web 2.0" developers

"Static" folks, like type system designers, **worry**.

What gets called? What if **`object`** has **no method** named by **`string`**?

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

```
object.[string]()
```

"static" folks

"web 2.0" developers

"Static" folks, like type system designers, **worry**.

What gets called? What if **`object`** has **no method** named by **`string`**?

"Web 2.0" developers think it's **cool**.

I can flexible and compact code, so I will take it over **static safety**.

Method Reflection and the Great Divide

reflective method call: dispatch based on **run-time value** (in string)

```
object.[string]()
```

“static” folks

“web 2.0” developers

“Sta
desi

Wha
has
str

ty.

Bridge the divide to support both first-class **reflective** method call and **static checking** of reflection safety

Ensure reflection safety with **dependent refinement type** expressing required **relationship**

```
class Callback
  var sel : Str
  var obj : Obj

  def call()
    this.obj.[this.sel]()
```

Ensure reflection safety with **dependent refinement type** expressing required **relationship**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def call()
  this.obj.[this.sel]()
```

obj must "respond to" sel

Ensure reflection safety with **dependent refinement type** expressing required **relationship**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def call()
  this.obj.[this.sel]()
```

`obj` must "respond to" `sel`

Shorthand for $\text{obj} :: \{\nu : \mathsf{Obj} \mid \nu \ \mathsf{r2}\ \text{sel}\}$

Ensure reflection safety with **dependent refinement type** expressing required **relationship**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def call()
  this.obj.[this.sel]()
```

`obj` must "respond to" `sel`

Shorthand for $\text{obj} :: \{\nu : \text{Obj} \mid \nu \text{ r2 sel}\}$

Guarantees no `MethodNotFound` error in `call()`

Similar relationship for **array bounds** safety

```
class Iterator
 var idx : Int
 var buf : Obj[] | indexedBy idx

 def get(): Obj
  return this.buf[this.idx]
```

Similar relationship for **array bounds** safety

idx must be a valid index into buf

```
class Iterator
  var idx : Int
  var buf : Obj[] | indexedBy idx

  def get(): Obj
    return this.buf[this.idx]
```

Similar relationship for **array bounds** safety

```
class Iterator
  var idx : Int
  var buf : Obj[] | indexedBy idx

  def get(): Obj
    return this.buf[this.idx]
```

idx must be a valid index into **buf**

Guarantees **no** “ArrayOutOfBounds”

Similar relationship for **array bounds** safety

`idx` must be a valid index into `buf`

```
class Iterator
 var idx : Int
 var buf : Obj[] | indexedBy idx

 def get(): Obj
  return this.buf[this.idx]
```

These kinds of **relationships** are important to **many safety properties**

Updating relationship causes **type error**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def update(s : Str, o : Obj | r2 s)
  this.sel = s
  this.obj = o

 def call()
  this.obj.[this.sel]()
```

Updating relationship causes **type error**

Field type says: `obj` must always respond to `sel`

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def update(s : Str, o : Obj | r2 s)
  this.sel = s
  this.obj = o

 def call()
  this.obj.[this.sel]()
```

Updating relationship causes **type error**

Field type says: `obj` must always respond to `sel`

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel
```

`o` guaranteed to respond to `s`

```
 def update(s : Str, o : Obj | r2 s)
  this.sel = s
  this.obj = o

 def call()
  this.obj.[this.sel]()
```

Updating relationship causes **type error**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def update(s : Str, o : Obj | r2 s)
  this.sel = s
  this.obj = o

 def call()
  this.obj.[this.sel]()
```

Field type says: `obj` must **always** respond to `sel`

`o` guaranteed to respond to `s`

Type error: **old** `obj` may not respond to **new** `sel`

Updating relationship causes **type error**

Field type says: `obj` must **always** respond to `sel`

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel
```

`o` guaranteed to respond to `s`

```
 def update(s : Str, o : Obj | r2 s)
  this.sel = s
  this.obj = o
```

Type error: **old** `obj` may not respond to **new** `sel`

```
 def call()
  this.obj.[this.sel]()
```

False alarm: no runtime error

Two styles of **reasoning** to determine **false alarm**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def update(s : Str, o : Obj| r2 s )
  this.sel = s
  this.obj = o

 def call()
  this.obj.[this.sel]()
```

Two styles of **reasoning** to determine **false alarm**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def update(s : Str, o : Obj| r2 s )
  this.sel = s
  this.obj = o

 def call()
  this.obj.[this.sel]()
```

Reasoning by global **invariant**: call safe if **relationship holds**

Two styles of **reasoning** to determine **false alarm**

```
class Callback
 var sel : Str
 var obj : Obj | r2 sel

 def update(s : Str, o : Obj| r2 s)
  this.sel = s
  this.obj = o

 def call()
  this.obj.[this.sel]()
```

Reasoning by global **invariant**: call safe if **relationship holds**

Two styles of **reasoning** to determine **false alarm**

```
class Callback
  var sel : Str
  var obj : Obj

  def update(s : Str, o : Obj| r2 s)
    this.sel = s
    this.obj = o

  def call()
    this.obj.[this.sel]()
```

Reasoning about **effects** of **imperative updates**

Reasoning by global **invariant**: call safe if **relationship holds**

Two styles of **reasoning** to determine **false alarm**

```
class Callback
  var sel : Str
  var obj : Obj

  def update(s : Str, o : Obj | r2 s)
    this.sel = s
    this.obj = o

  def call()
    this.obj.[this.sel]()
```

Reasoning about **effects of imperative updates**

Relationship violated

Reasoning by global **invariant**: call safe if **relationship holds**

Two styles of **reasoning** to determine **false alarm**

```
class Callback
  var sel : Str
  var obj : Obj

  def update(s : Str, o : Obj | r2 s)
    this.sel = s
    this.obj = o

  def call()
    this.obj.[this.sel]()
```

Reasoning about **effects of imperative updates**

Relationship violated

Relationship restored

Reasoning by global **invariant**: call safe if **relationship holds**

Idea: **Selectively alternate** between **reasoning styles** in verification

Fissile Type Analysis **combines** two styles of **reasoning**

Fissile Type Analysis **combines** two styles of **reasoning**

Automated **reasoning**
about **global**
invariants

Fissile Type Analysis **combines** two styles of **reasoning**

Automated **reasoning** about **global invariants**

$$\Gamma \vdash \cdots$$

Flow-Insensitive Type Systems

Fissile Type Analysis **combines** two styles of **reasoning**

Automated **reasoning** about **global invariants**

$\Gamma \vdash \cdots$

Flow-Insensitive Type Systems

Automated **reasoning** about **execution**

Fissile Type Analysis **combines** two styles of **reasoning**

Automated **reasoning** about **global invariants**

$\Gamma \vdash \cdots$

Flow-Insensitive Type Systems

Automated **reasoning** about **execution**

$\gamma(\cdot) = \cdots$

Abstract Interpretation/ Flow Analysis

Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis

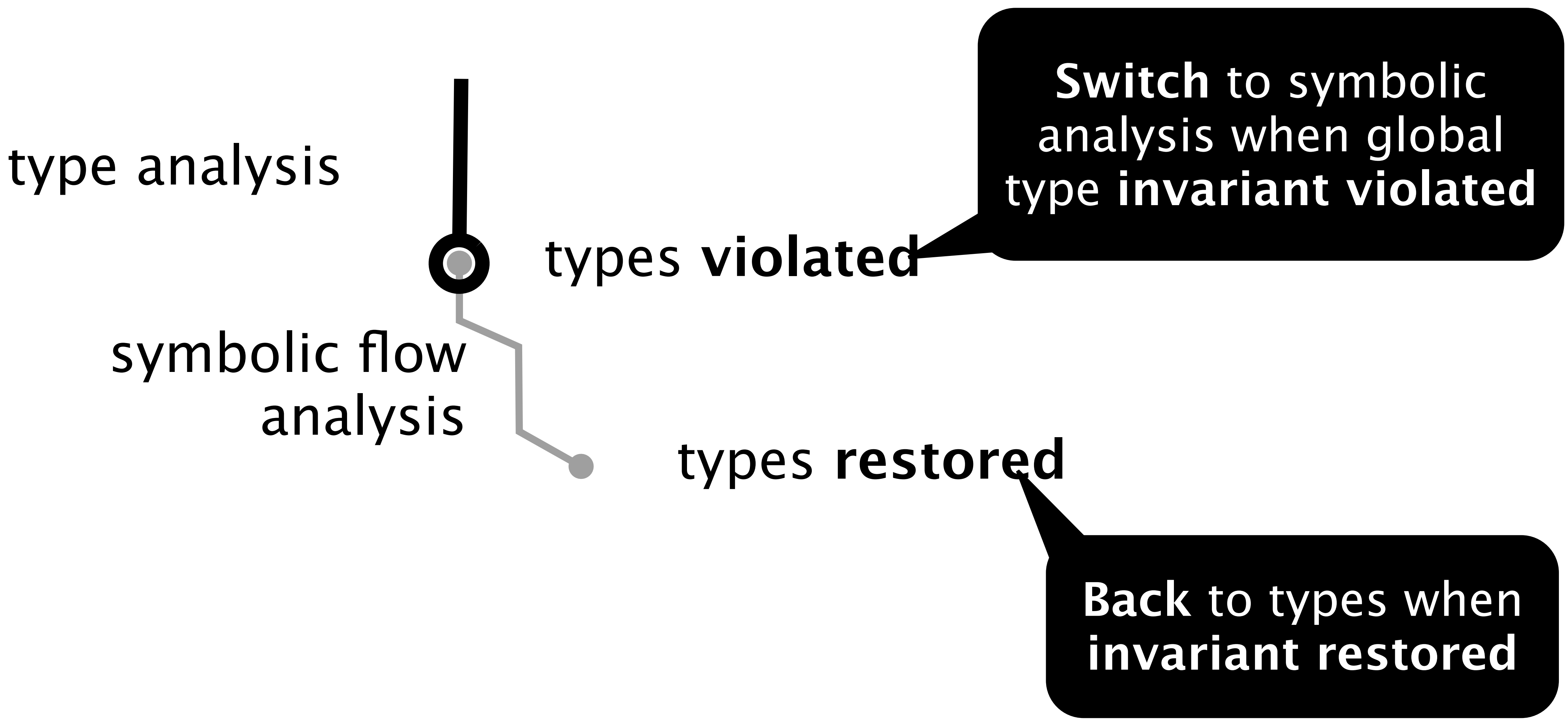
Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis

type analysis

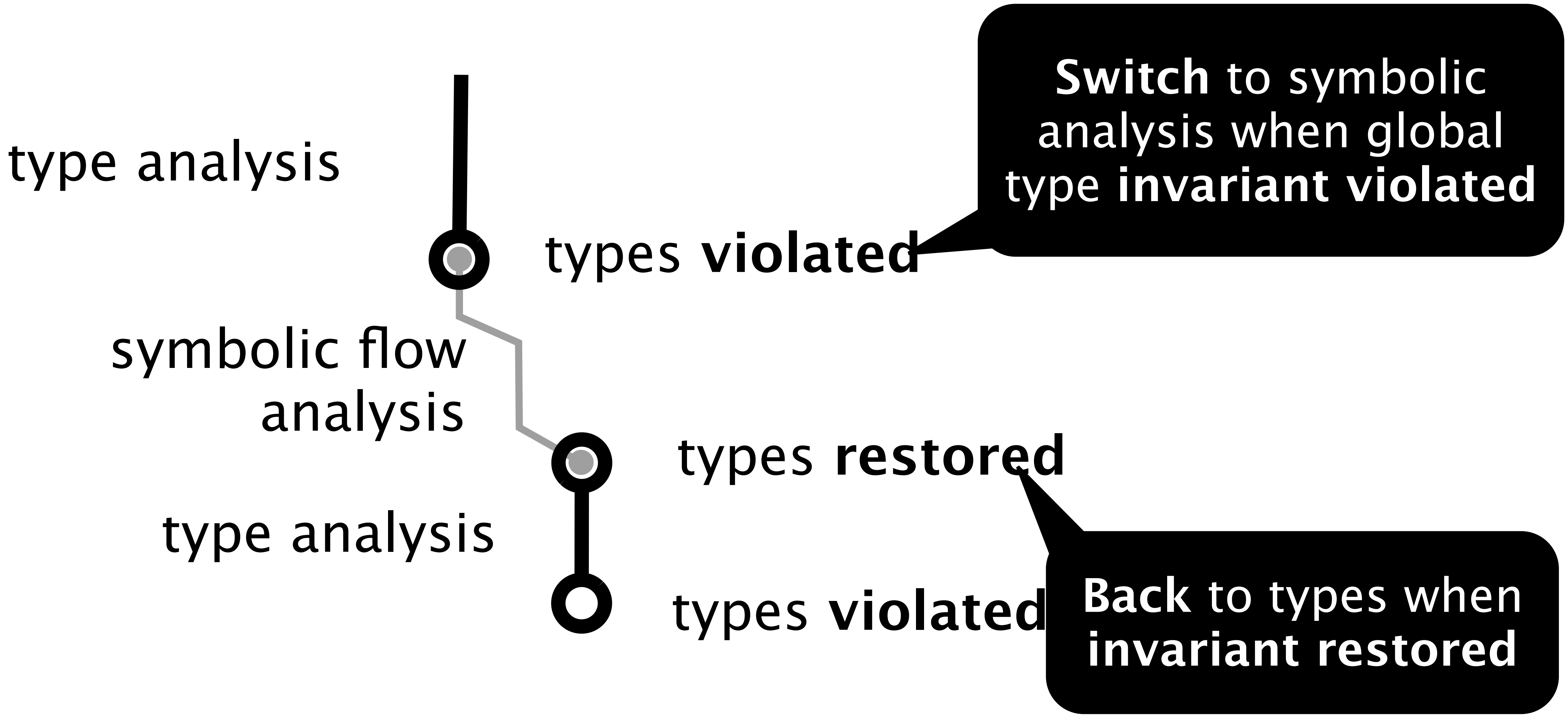
types **violated**

Switch to symbolic analysis when global type **invariant violated**

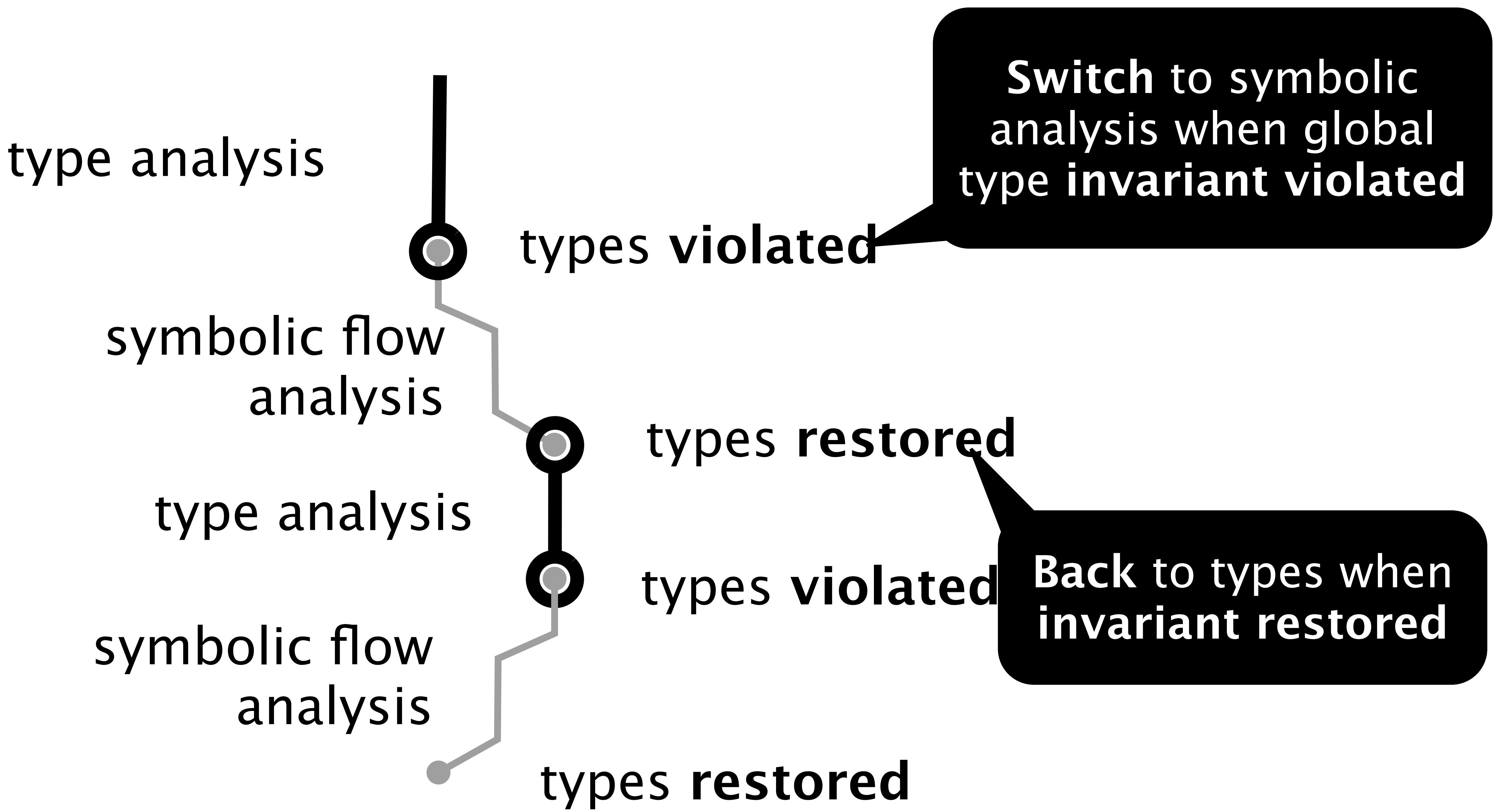
Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis



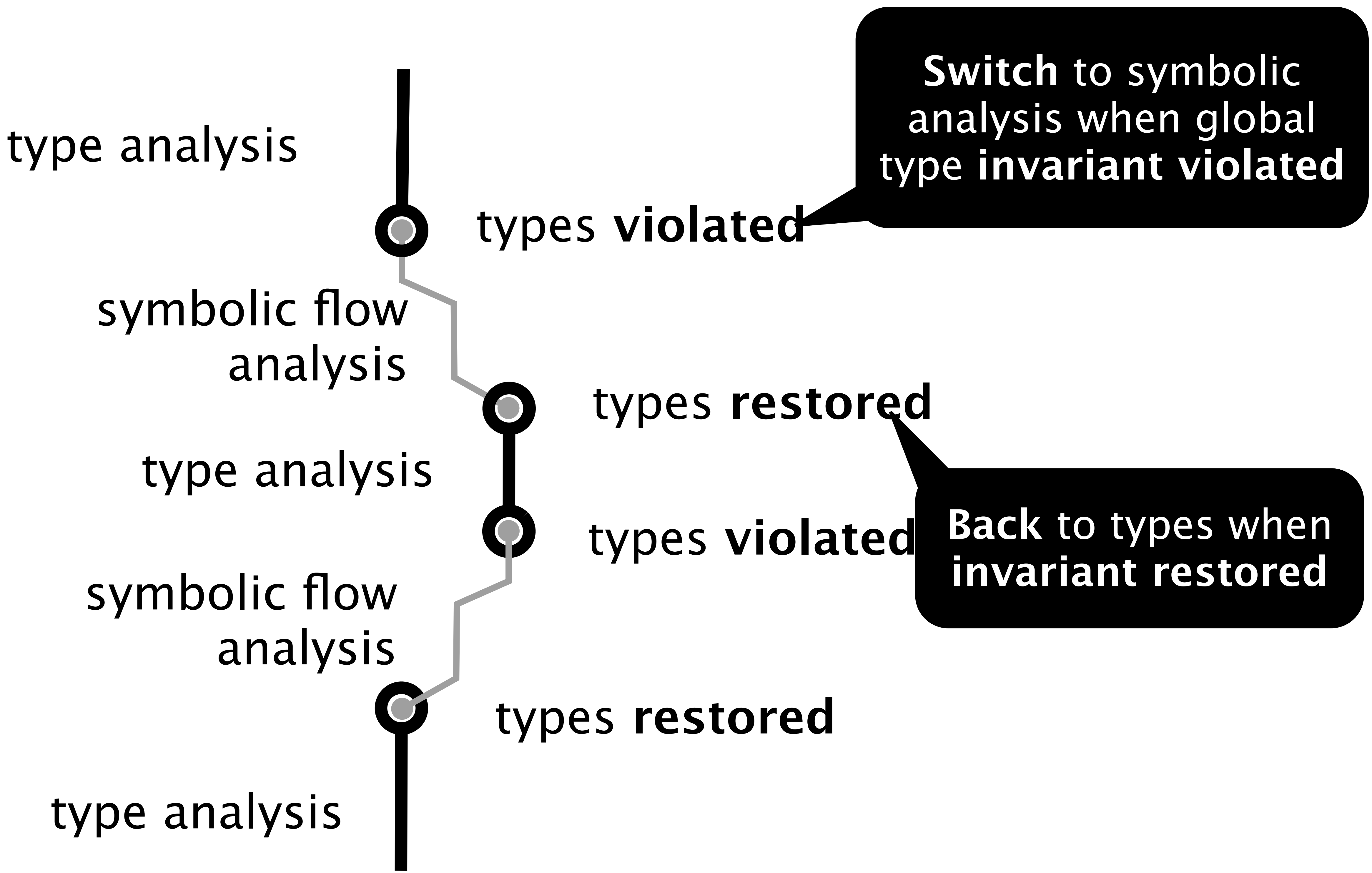
Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis



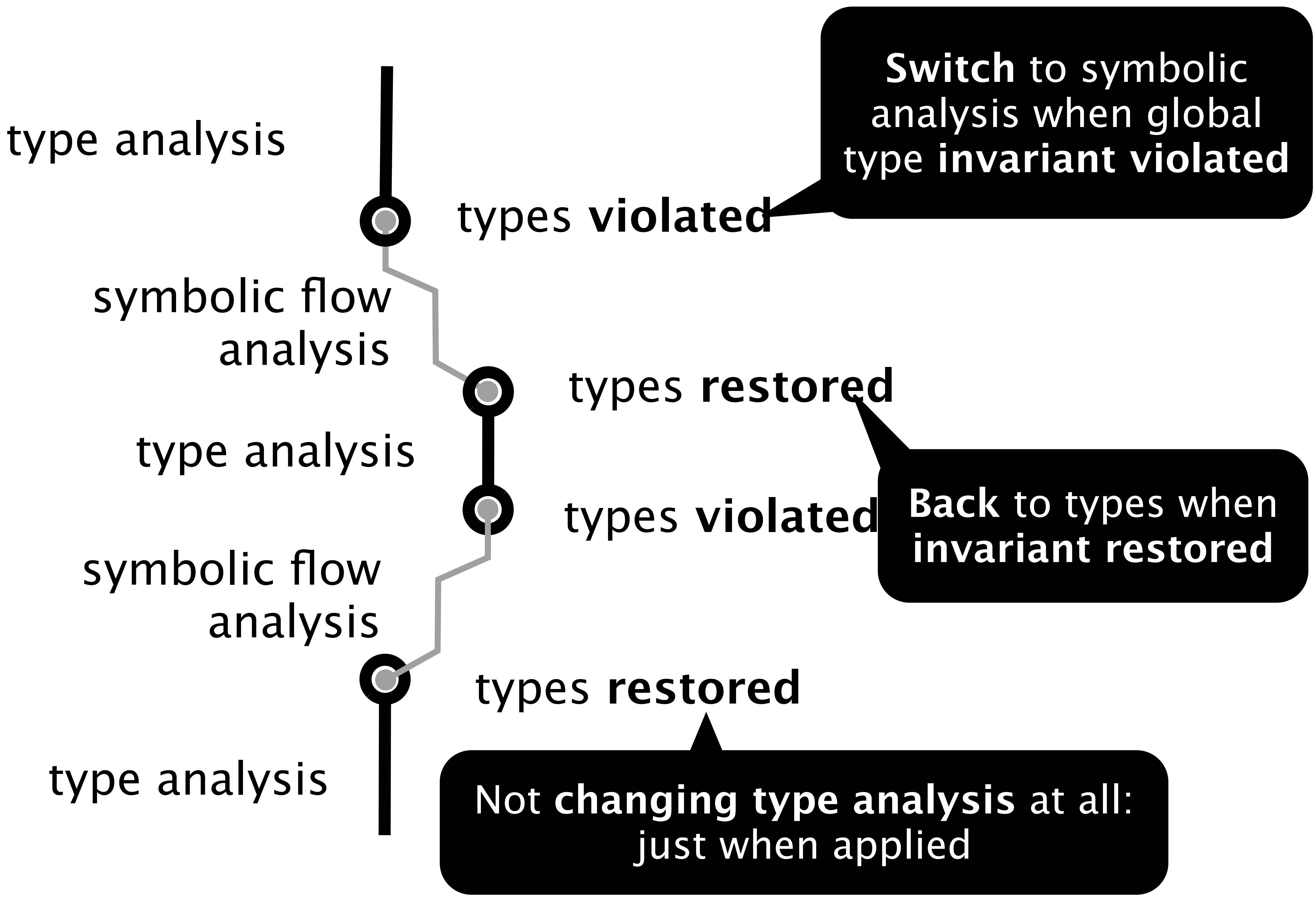
Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis



Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis

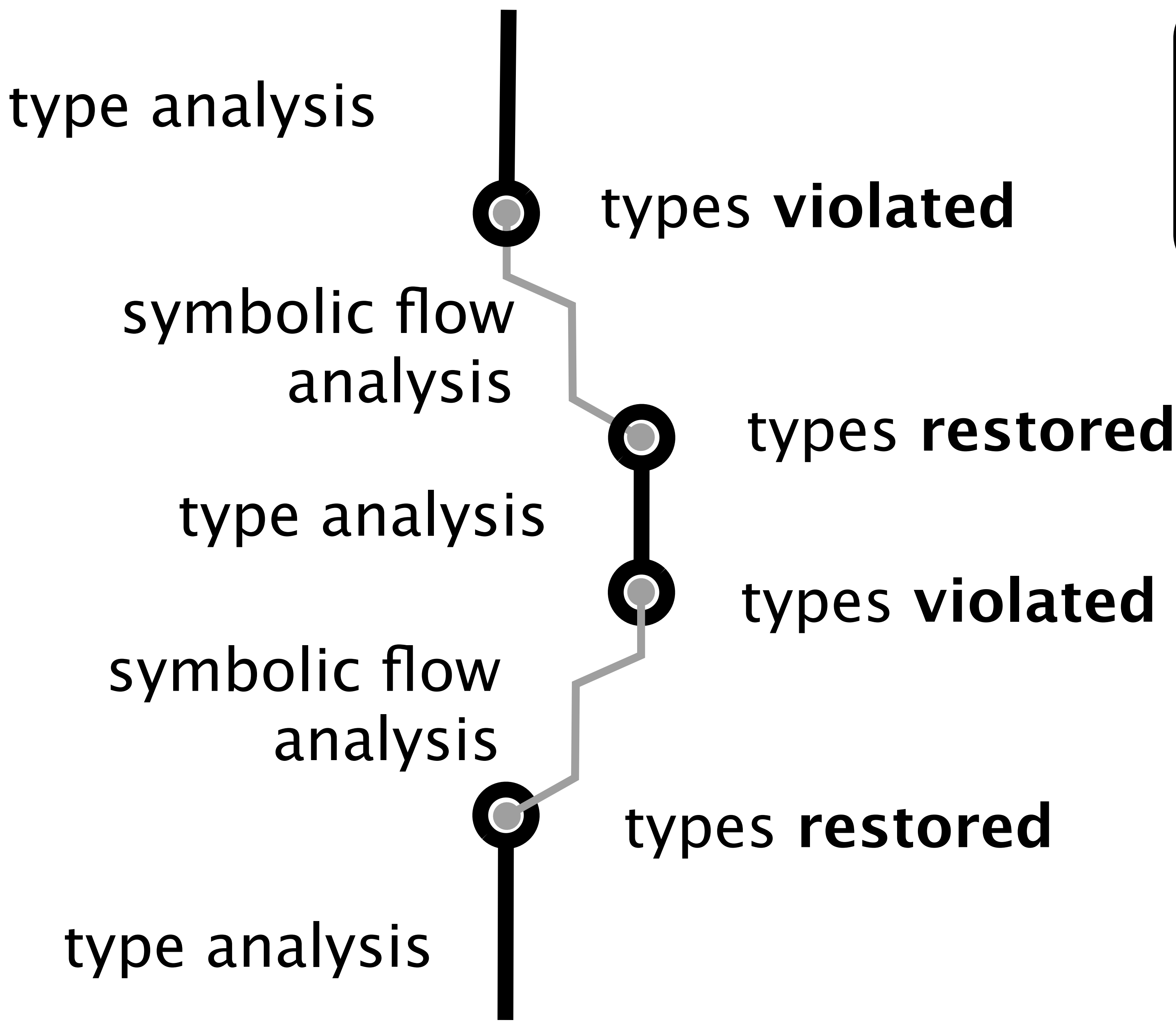


Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis

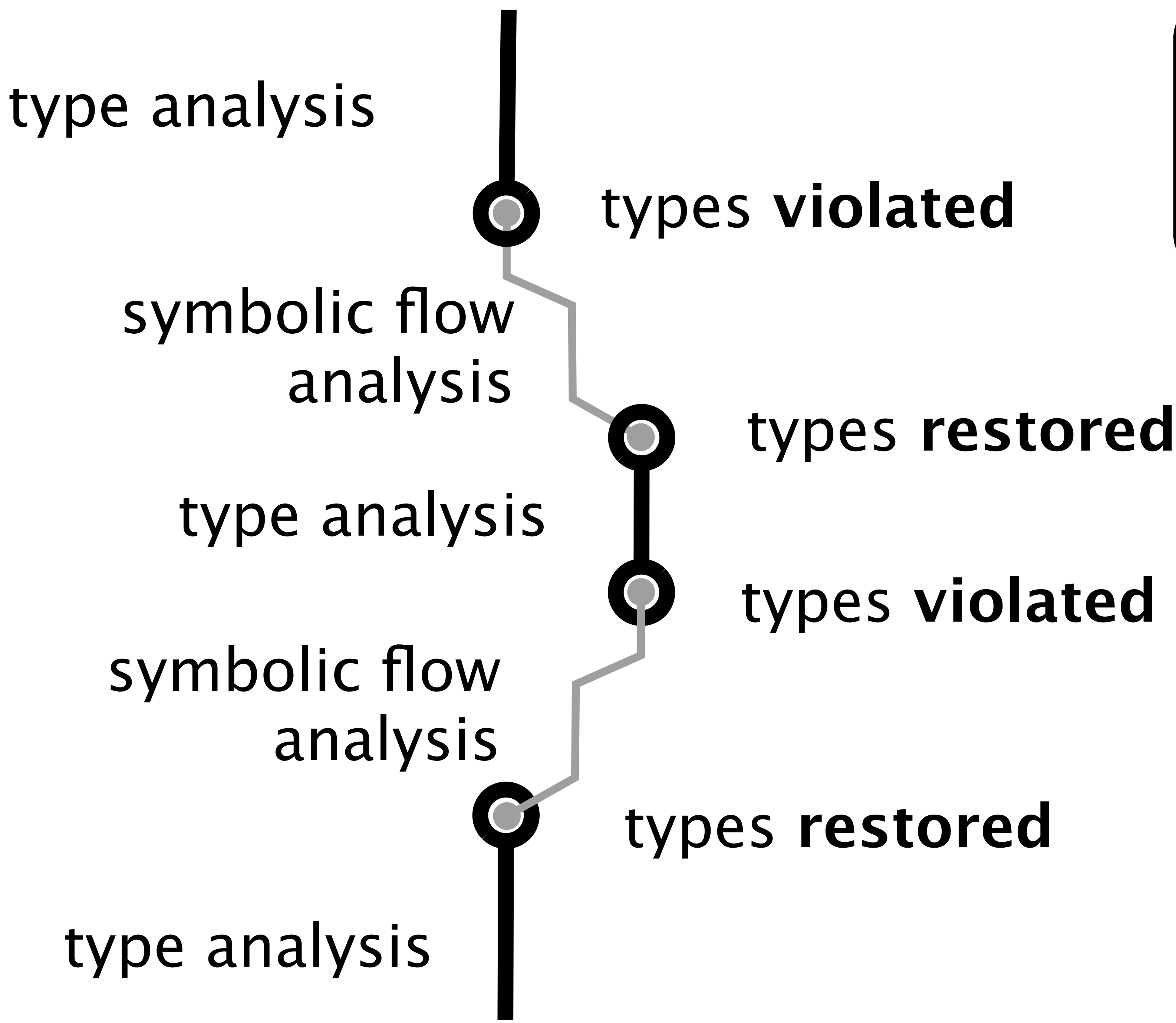


Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis

Effective when **global** type **invariant holds most of the time**



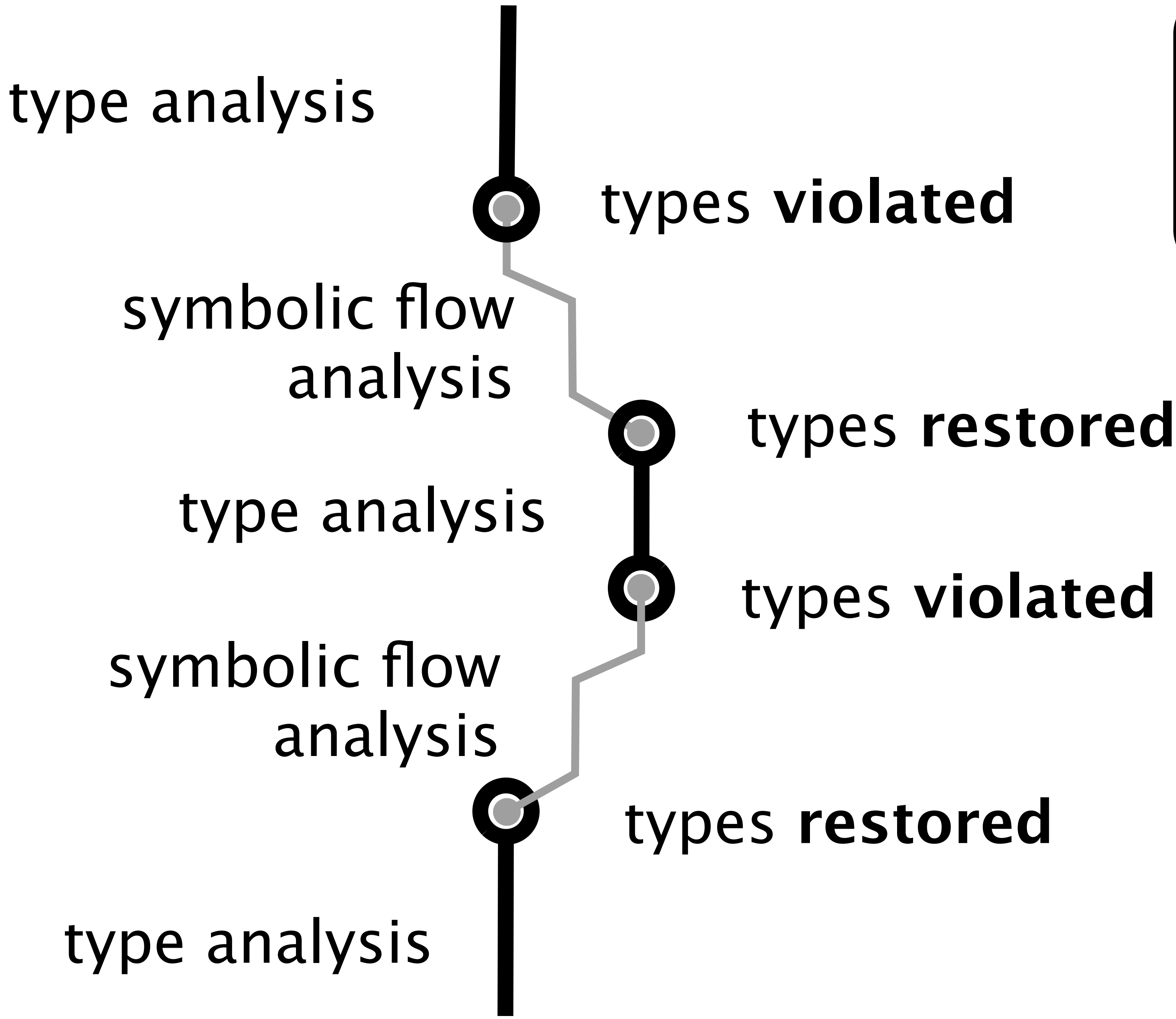
Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis



Effective when **global** type **invariant holds most of the time**

- **Relationship updates**

Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis

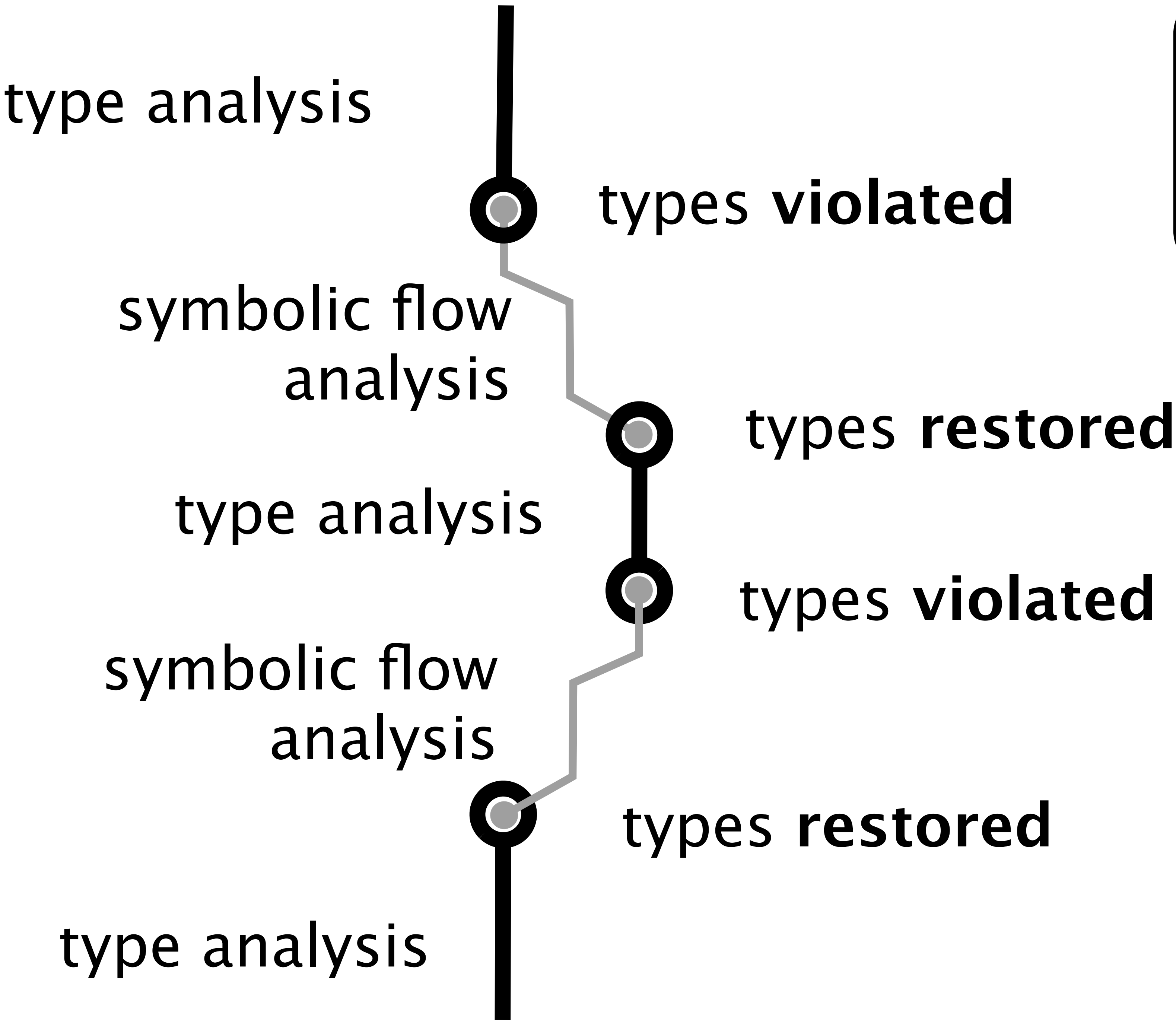


Effective when **global** type **invariant holds most of the time**

- **Relationship updates**
- **Occurrence typing**

Verification of **almost-everywhere invariants** with **intertwined** type and flow analysis

Effective when **global** type **invariant holds most of the time**



- **Relationship updates**
- **Occurrence typing**
- **Tagged unions**

Play to the **strengths** of each intertwined **analysis**

Play to the **strengths** of each intertwined **analysis**

Flow-Insensitive Types

- Easy to **specify global** invariants
- **Fast**
- Natural for **modular** reasoning
- Good **error reporting**

Play to the **strengths** of each intertwined **analysis**

Flow-Insensitive Types

Easy to **specify global** invariants
Fast
Natural for **modular** reasoning
Good **error reporting**

Symbolic Flow Analysis

Natural for **local** reasoning about **heap mutation**
Precise
Can be disjunctive/path-sensitive

Play to the **strengths** of each intertwined **analysis**

Flow-Insensitive Types

Easy to **specify global** invariants
Fast
Natural for **modular** reasoning
Good **error reporting**

flow-sensitive typing?
ownership types?
alias types?
permissions?
effects?

Symbolic Flow Analysis

Natural for **local** reasoning about **heap mutation**
Precise
Can be disjunctive/path-sensitive

Play to the **strengths** of each intertwined **analysis**

Flow-Insensitive Types

Easy to **specify global** invariants
Fast
Natural for **modular** reasoning
Good **error reporting**

flow-sensitive typing?
ownership types?
alias types?
permissions?
effects?

Goal: keep **types** as **simple as possible**

Symbolic Flow Analysis

Natural for **local** reasoning about **heap mutation**
Precise
Can be disjunctive/path-sensitive

Play to the **strengths** of each intertwined **analysis**

Flow-Insensitive Types

Easy to **specify global** invariants
Fast
Natural for **modular** reasoning
Good **error reporting**

flow-sensitive typing?
ownership types?
alias types?
permissions?
effects?

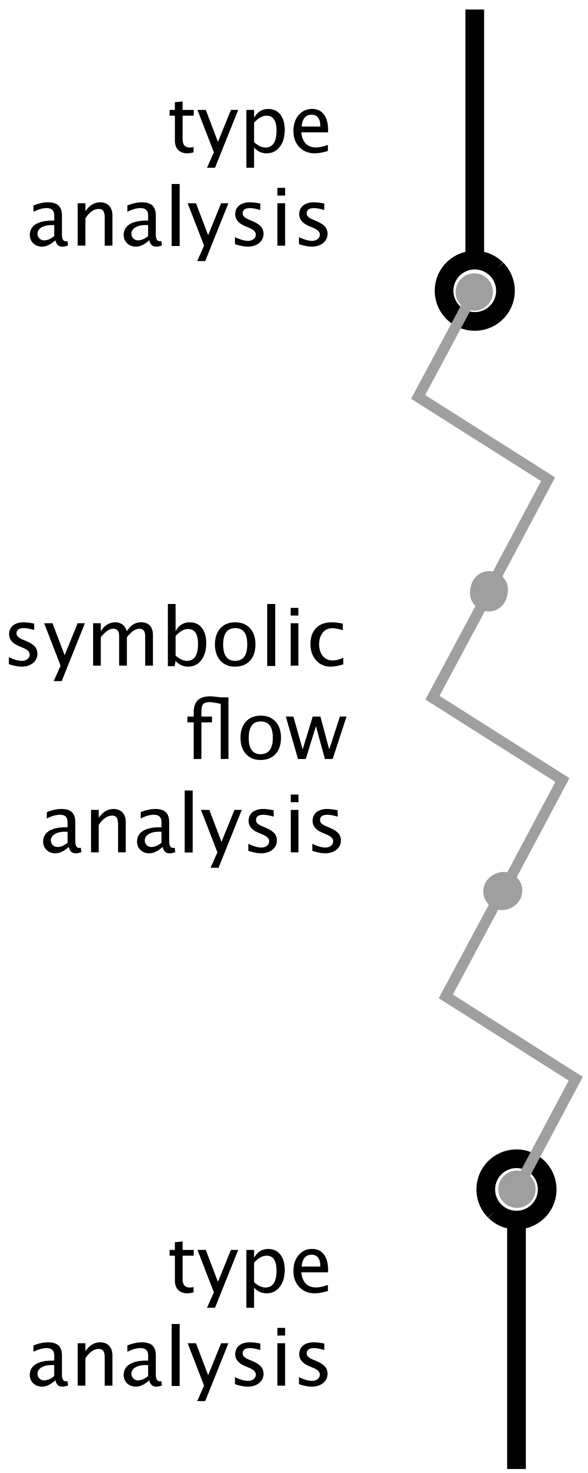
Goal: keep **types** as **simple as possible**

Symbolic Flow Analysis

Natural for **local** reasoning about **heap mutation**
Precise
Can be disjunctive/path-
s

Complexity lies in **handoff** between analyses and in **symbolic analysis**

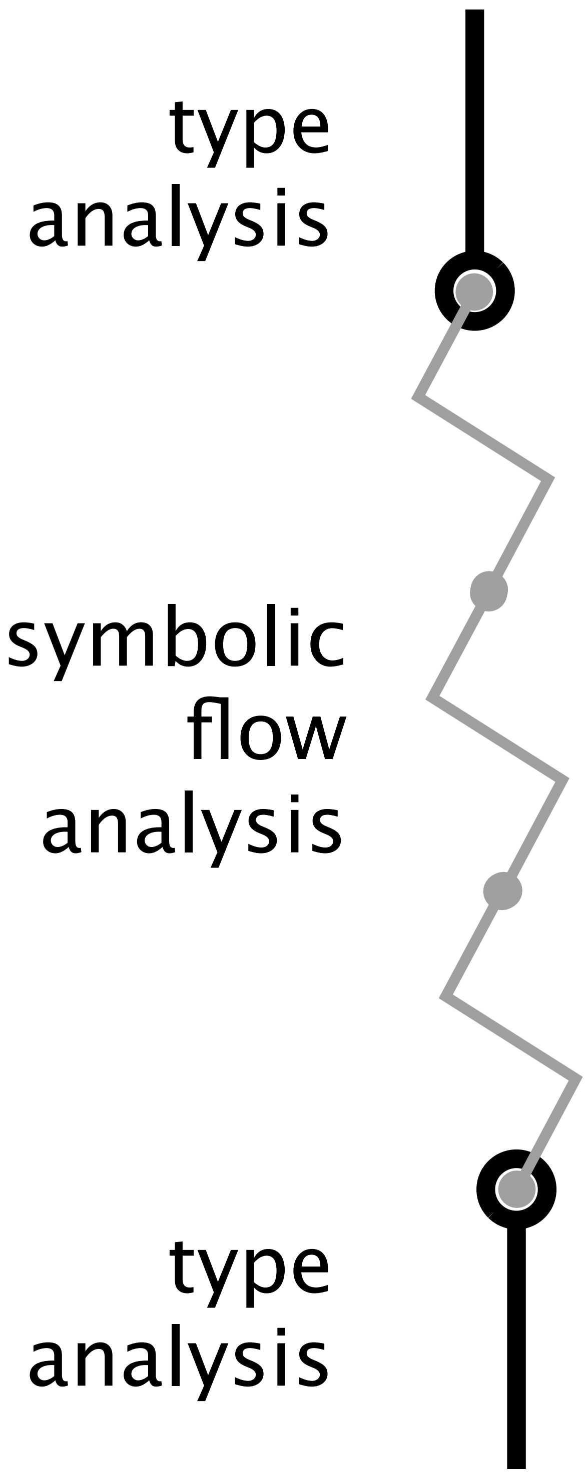
Key Contributions



1. **Translate** type invariant into symbolic state via **“symbolization”** of type environment

2. **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**

Key Contributions



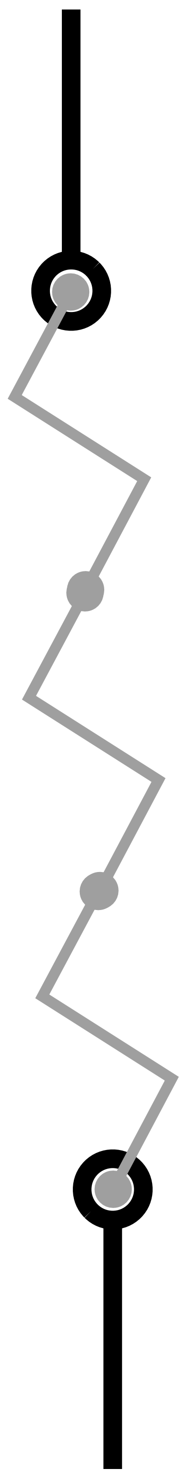
1. **Translate** type invariant into symbolic state via **“symbolization”** of type

Reason precisely only **when** type invariant violated

2. **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**

Key Contributions

type
analysis



symbolic
flow
analysis

type
analysis

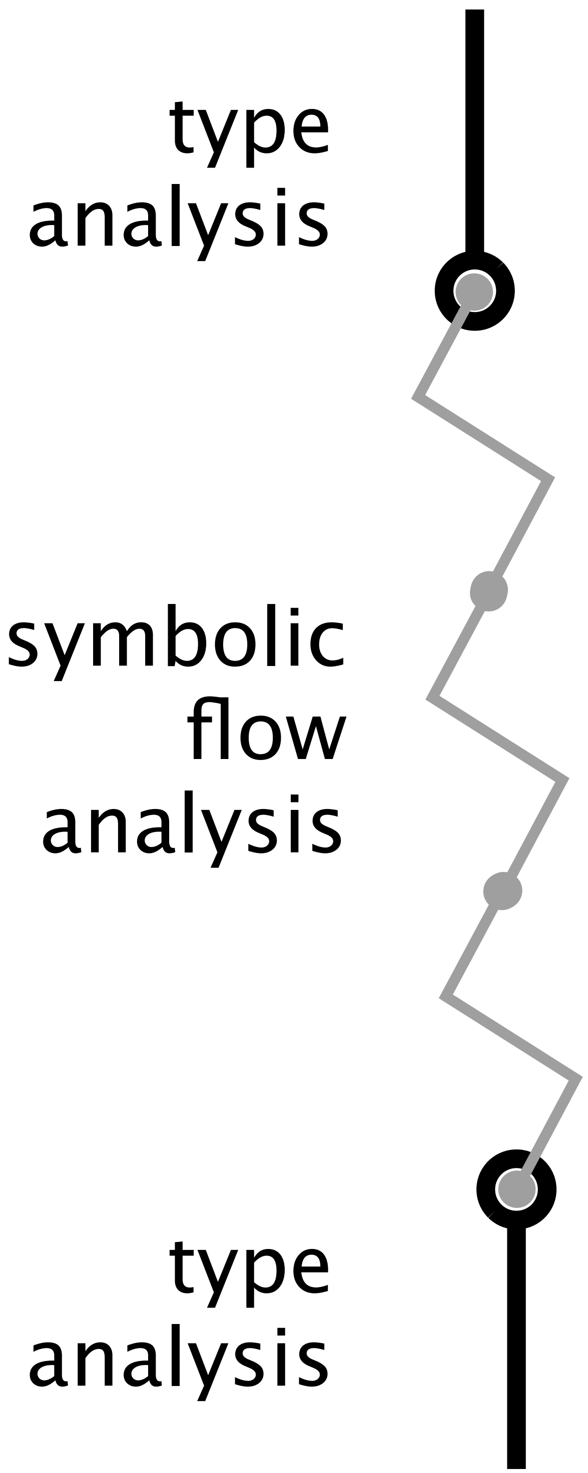
1 **Translate** type invariant into symbolic state via **“symbolization”** of type

Reason precisely only **when** type invariant violated

2 **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and

Reason precisely only for locations **where** type invariant violated

Key Contributions



1 **Translate** type invariant into symbolic state via **“symbolization”** of type

Reason precisely only **when** type invariant violated

2 **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and

Reason precisely only for locations **where** type invariant violated

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s ✗
  this.obj = o
```

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Type environment

Maps local **variables** to dependent **types**

$$\Gamma \quad \begin{array}{l} s : \mathsf{Str} \\ o : \mathsf{Obj} \mid \mathsf{r2}\ s \\ \mathrm{this} : \mathsf{Callback} \end{array}$$

Refinements refer to **variables**

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Type environment

Maps local **variables** to dependent **types**

$$\Gamma \quad \begin{array}{l} \mathrm{s} : \mathsf{Str} \\ \mathrm{o} : \mathsf{Obj} \mid \mathsf{r2}\ \mathrm{s} \\ \mathrm{this} : \mathsf{Callback} \end{array}$$

Refinements refer to **variables**

$$\xrightarrow{\textbf{symbolize}}$$

Symbolic state

$$\widetilde{E} \quad \begin{array}{l} \mathrm{s} : \widetilde{\mathrm{s}} \\ \mathrm{o} : \widetilde{\mathrm{o}} \\ \mathrm{this} : \widetilde{\mathrm{t}} \end{array}$$

$$\widetilde{\Gamma} \quad \begin{array}{l} \widetilde{\mathrm{s}} : \mathsf{Str} \\ \widetilde{\mathrm{o}} : \mathsf{Obj} \mid \mathsf{r2}\ \widetilde{\mathrm{s}} \\ \widetilde{\mathrm{t}} : \mathsf{Callback} \end{array}$$

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Type environment

Maps local **variables** to dependent **types**

$$\Gamma \quad \begin{array}{l} \mathrm{s} : \mathsf{Str} \\ \mathrm{o} : \mathsf{Obj} \mid \mathsf{r2}\ \mathrm{s} \\ \mathrm{this} : \mathsf{Callback} \end{array}$$

Refinements refer to **variables**

Maps local variables to symbolic **values**

$$\xrightarrow{\textbf{symbolize}}$$

Symbolic state

$$\widetilde{\mathrm{E}} \quad \begin{array}{l} \mathrm{s} : \widetilde{\mathrm{s}} \\ \mathrm{o} : \widetilde{\mathrm{o}} \\ \mathrm{this} : \widetilde{\mathrm{t}} \end{array}$$

$$\widetilde{\Gamma} \quad \begin{array}{l} \widetilde{\mathrm{s}} : \mathsf{Str} \\ \widetilde{\mathrm{o}} : \mathsf{Obj} \mid \mathsf{r2}\ \widetilde{\mathrm{s}} \\ \widetilde{\mathrm{t}} : \mathsf{Callback} \end{array}$$

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Type environment

Maps local **variables** to dependent **types**

Γ
s : Str
o : Obj | r2 s
this : Callback

Refinements refer to **variables**

symbolize

Symbolic state

Maps local variables to symbolic **values**

$\widetilde{E}$
$s : \widetilde{s}$
$o : \widetilde{o}$
$this : \widetilde{t}$

$\widetilde{\Gamma}$
$\widetilde{s}$: Str
$\widetilde{o}$: Obj | r2 $\widetilde{s}$
$\widetilde{t}$: Callback

Maps symbolic values to dependent types **lifted** to symbolic values (**symbolic facts**)

Symbolization splits a **type environment** into **facts** about **values** and **storage** for those values

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Type environment

Maps local **variables** to dependent **types**

$$\Gamma \quad \begin{array}{l} \mathrm{s} : \textsf{Str} \\ \mathrm{o} : \textsf{Obj} \mid \textsf{r2}\ \mathrm{s} \\ \mathrm{this} : \textsf{Callback} \end{array}$$

Refinements refer to **variables**

symbolize $\longrightarrow$

Symbolic state

Maps local variables to symbolic **values**

$$\widetilde{\mathrm{E}} \quad \begin{array}{l} \mathrm{s} : \widetilde{\mathrm{s}} \\ \mathrm{o} : \widetilde{\mathrm{o}} \\ \mathrm{this} : \widetilde{\mathrm{t}} \end{array}$$

Refinements refer to **values**

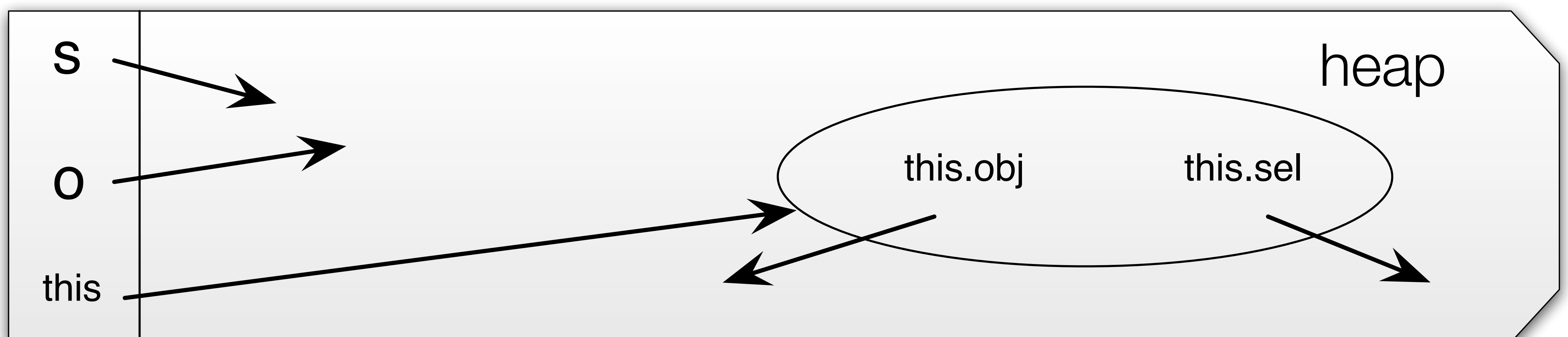
$$\widetilde{\Gamma} \quad \begin{array}{l} \widetilde{\mathrm{s}} : \textsf{Str} \\ \widetilde{\mathrm{o}} : \textsf{Obj} \mid \textsf{r2}\ \widetilde{\mathrm{s}} \\ \widetilde{\mathrm{t}} : \textsf{Callback} \end{array}$$

Maps symbolic values to dependent types **lifted** to symbolic values (**symbolic facts**)

Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \begin{array}{l} \mathrm{s} : \mathsf{Str} \\ \mathrm{o} : \mathsf{Obj} \mid \mathsf{r2}\ \mathrm{s} \\ \mathrm{this} : \mathsf{Callback} \end{array}$$

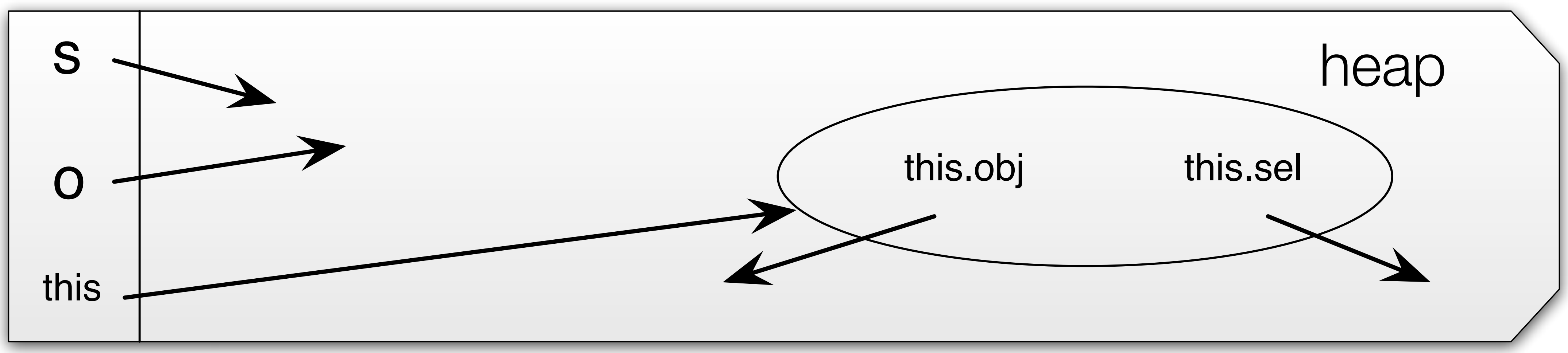


Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

A type environment constrains local variables

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \quad \begin{array}{l} s : \mathsf{Str} \\ o : \mathsf{Obj} \mid \mathsf{r2}\ s \\ \mathrm{this} : \mathsf{Callback} \end{array}$$



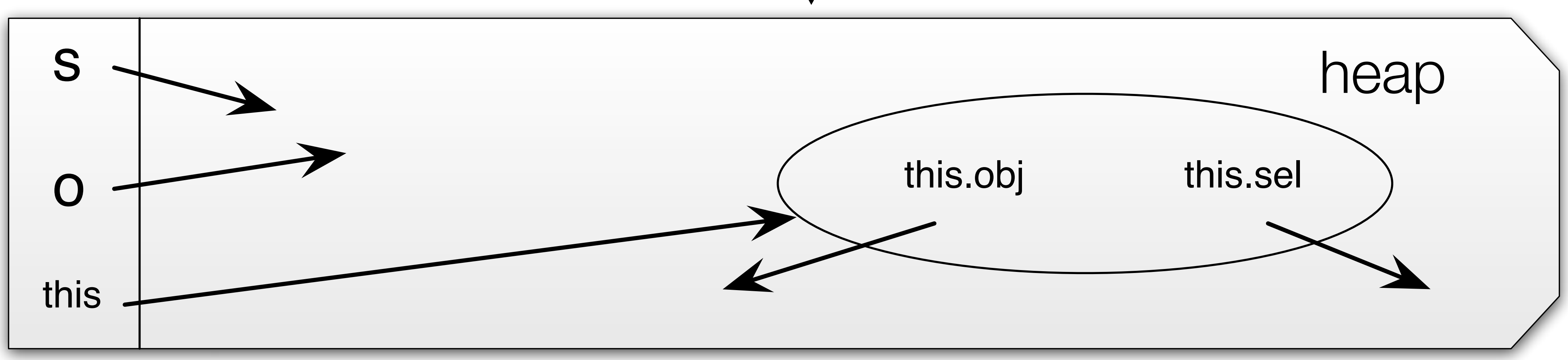
Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

A **type environment constrains local variables**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \begin{array}{l} \mathrm{s} : \mathsf{Str} \\ \mathrm{o} : \mathsf{Obj} \mid \mathsf{r2}\ \mathrm{s} \\ \mathrm{this} : \mathsf{Callback} \end{array}$$

But also constrains the reachable heap to be **type-consistent**: fields must **conform** to **declared** types



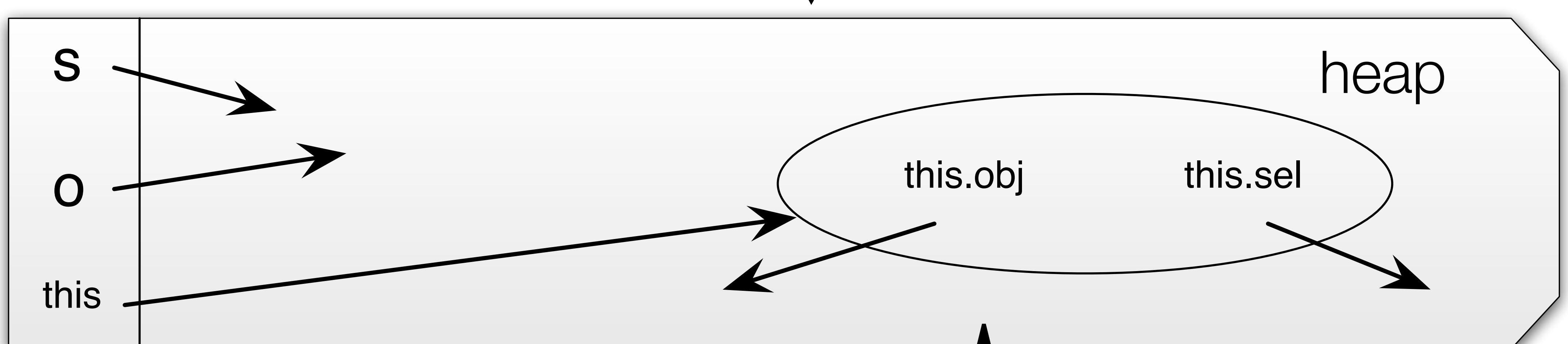
Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

A **type environment** constrains local variables

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \begin{array}{l} s : \mathsf{Str} \\ o : \mathsf{Obj} \mid \mathsf{r2}\ s \\ \mathrm{this} : \mathsf{Callback} \end{array}$$

But also constrains the reachable heap to be **type-consistent**: fields must **conform** to **declared** types

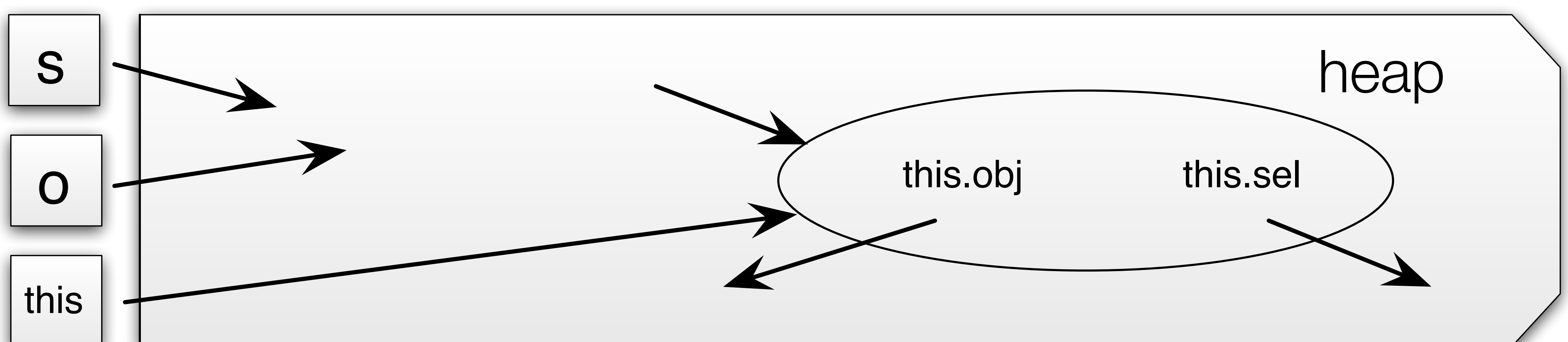


This picture captures the **fully type-consistent concrete** state

Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \begin{array}{l} s : \mathsf{Str} \\ o : \mathsf{Obj} \mid \mathsf{r2}\ s \\ \mathrm{this} : \mathsf{Callback} \end{array} \xrightarrow{\textbf{symbolize}} \widetilde{E} \quad \widetilde{\Gamma}$$

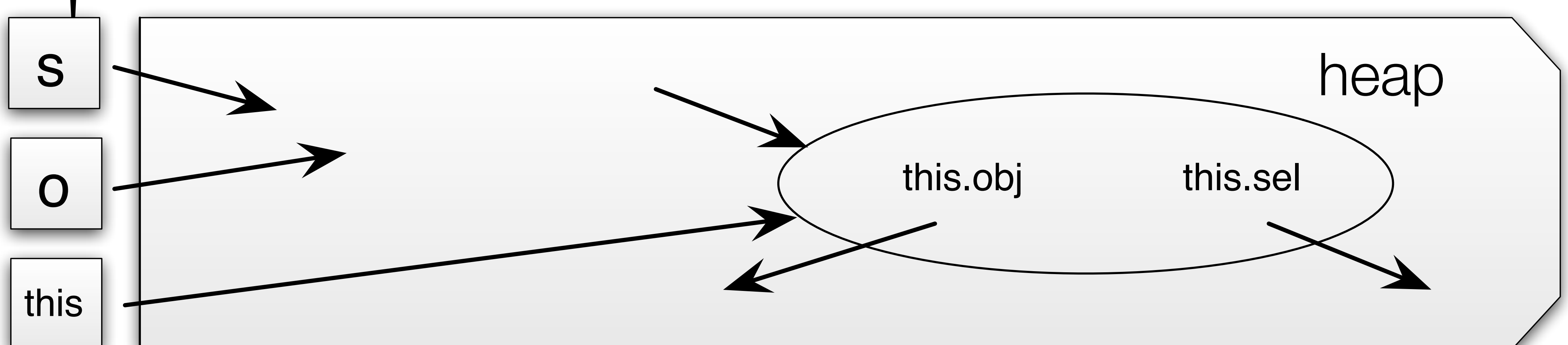


Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

Symbolic environment
allows, e.g., int in s

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \quad \begin{array}{l} \mathrm{s} : \mathsf{Str} \\ \mathrm{o} : \mathsf{Obj} \mid \mathsf{r2}\ \mathrm{s} \\ \mathrm{this} : \mathsf{Callback} \end{array} \xrightarrow{\textbf{symbolize}} \widetilde{\mathrm{E}} \quad \widetilde{\Gamma}$$

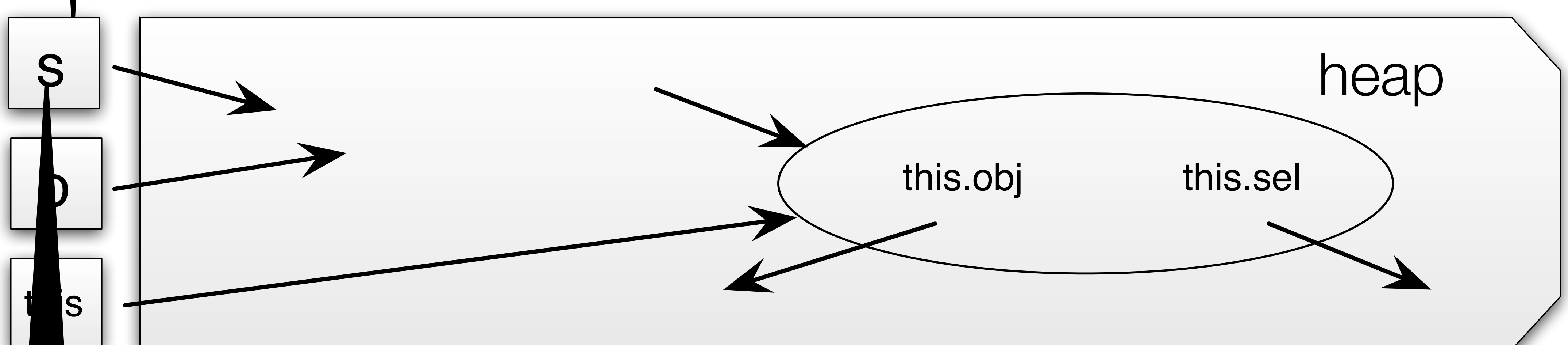


Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

Symbolic environment allows, e.g., int in s

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \quad \begin{array}{l} s : \mathrm{Str} \\ o : \mathrm{Obj} \mid \mathrm{r2}\ s \\ \mathrm{this} : \mathrm{Callback} \end{array} \quad \xrightarrow{\textbf{symbolize}} \quad \widetilde{E} \quad \widetilde{\Gamma}$$



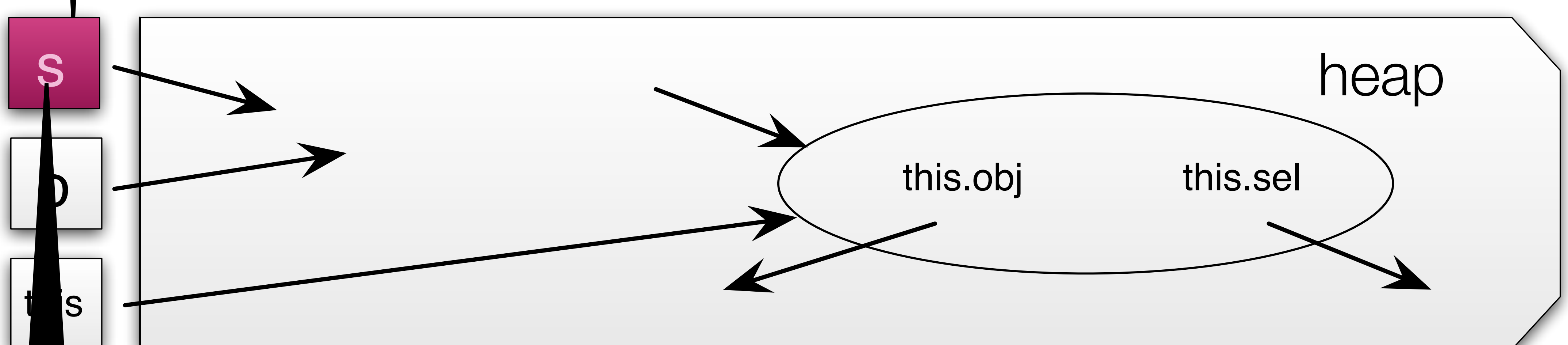
Immediately type-inconsistent: value stored **without dereferences violates** a type constraint

Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

Symbolic environment
allows, e.g., int in s

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\Gamma \quad \begin{array}{l} s : \mathsf{Str} \\ o : \mathsf{Obj} \mid \mathsf{r2}\ s \\ \mathit{this} : \mathsf{Callback} \end{array} \quad \xrightarrow{\textbf{symbolize}} \quad \widetilde{E} \quad \widetilde{\Gamma}$$



Immediately type-inconsistent: value stored **without dereferences violates** a type constraint

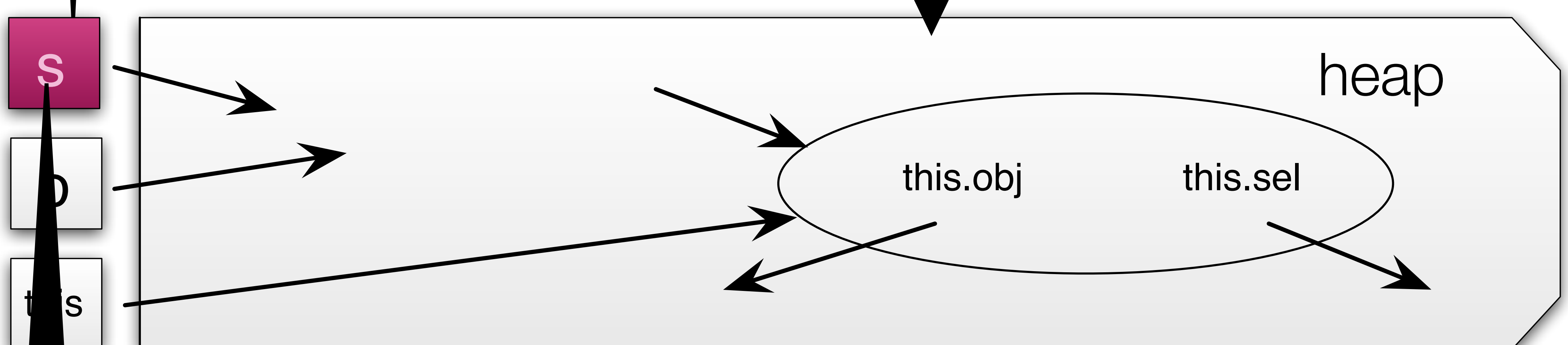
Symbolization allows **local variables** to hold values **inconsistent** with **declared types**

Symbolic environment allows, e.g., int in s

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Γ s : Str
o : Obj | r2 s
this : Callback

Grey indicates storage that is **not immediately type-inconsistent**



Immediately type-inconsistent: value stored **without dereferences violates** a type constraint

Symbolization **unpacks local cells**, but **symbolic facts** about **values** still **constrain the heap**

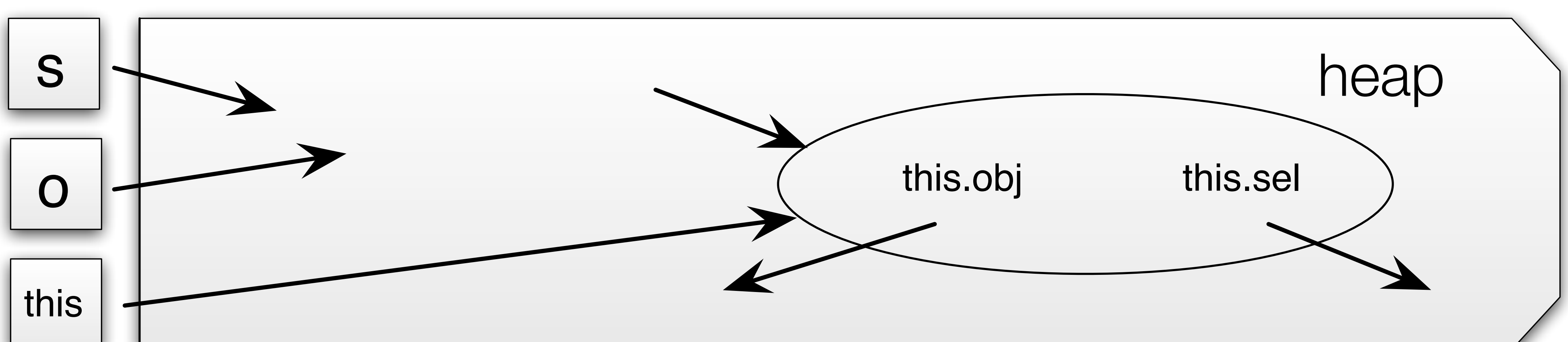
Type environment

$$\Gamma \quad \begin{array}{l} s : \mathsf{Str} \\ o : \mathsf{Obj} \mid \mathsf{r2}\ s \\ this : \mathsf{Callback} \end{array}$$

$$\xrightarrow{\textbf{symbolize}}$$

Symbolic fact map

$$\widetilde{\Gamma} \quad \begin{array}{l} \widetilde{s} : \mathsf{Str} \\ \widetilde{o} : \mathsf{Obj} \mid \mathsf{r2}\ \widetilde{s} \\ \widetilde{t} : \mathsf{Callback} \end{array}$$



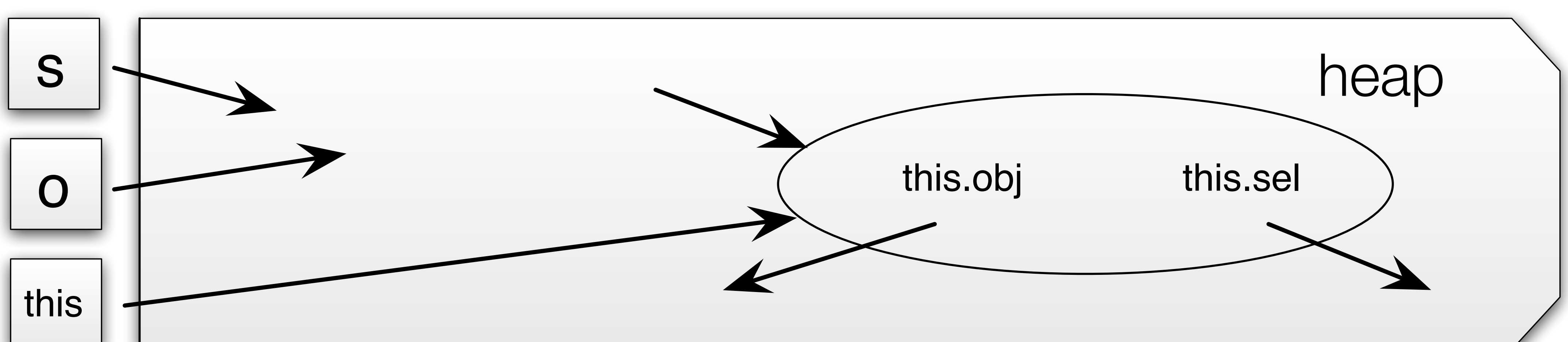
Symbolization **unpacks local cells**, but **symbolic facts** about **values** still **constrain the heap**

Type environment

Base types same on both sides

Symbolic fact map

$$\Gamma \quad \begin{array}{l} s : \mathsf{Str} \\ o : \mathsf{Obj} \mid \mathsf{r2}\ s \\ \mathrm{this} : \mathsf{Callback} \end{array} \quad \xrightarrow{\textbf{symbolize}} \quad \widetilde{\Gamma} \quad \begin{array}{l} \tilde{s} : \mathsf{Str} \\ \tilde{o} : \mathsf{Obj} \mid \mathsf{r2}\ \tilde{s} \\ \tilde{t} : \mathsf{Callback} \end{array}$$



Symbolization **unpacks local cells**, but **symbolic facts** about **values** still **constrain the heap**

Type environment

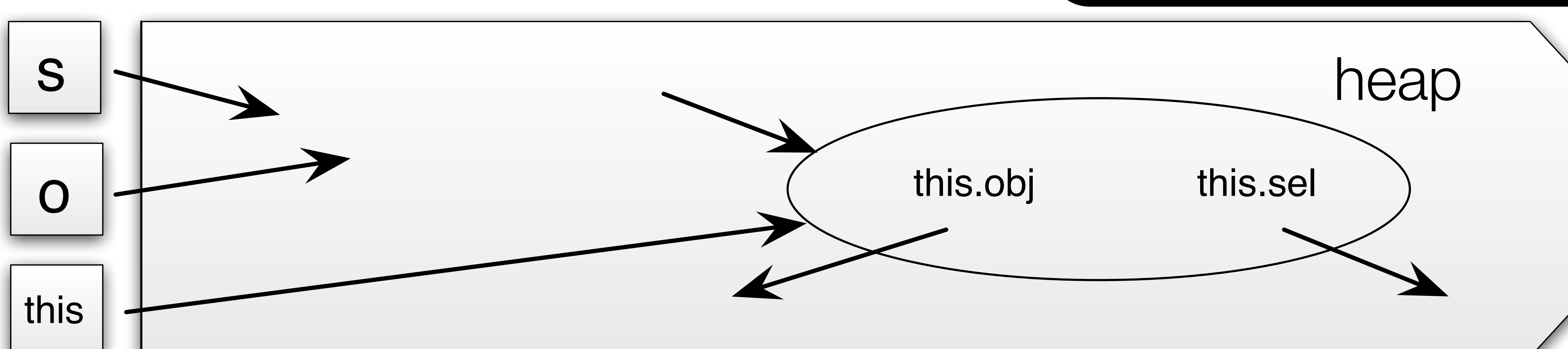
Base types same on both sides

Symbolic fact map

$$\Gamma \quad \begin{array}{l} \mathrm{s} : \mathsf{Str} \\ \mathrm{o} : \mathsf{Obj} \mid \mathsf{r2}\ \mathrm{s} \\ \mathrm{this} : \mathsf{Callback} \end{array} \quad \xrightarrow{\textbf{symbolize}} \quad \widetilde{\Gamma} \quad \begin{array}{l} \widetilde{\mathrm{s}} : \mathsf{Str} \\ \widetilde{\mathrm{o}} : \mathsf{Obj} \mid \mathsf{r2}\ \widetilde{\mathrm{s}} \\ \widetilde{\mathrm{t}} : \mathsf{Callback} \end{array}$$

$$\mathsf{Callback} \triangleq \{\mathrm{sel} : \mathsf{Str}, \mathrm{obj} : \mathsf{Obj} \mid \mathsf{r2}\ \mathrm{sel}\}$$

Base type field refinements still refer to **fields**



Summarize heap locations that are **not immediately type-inconsistent** with **okheap**

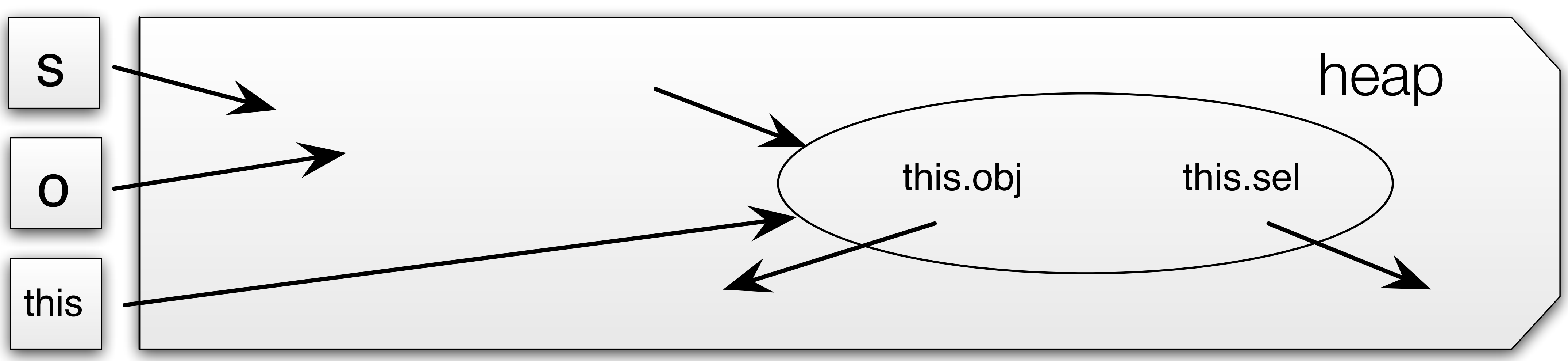
Symbolic Heap

$$\widetilde{H}$$

okheap

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Concrete State



Summarize heap locations that are **not immediately type-inconsistent** with **okheap**

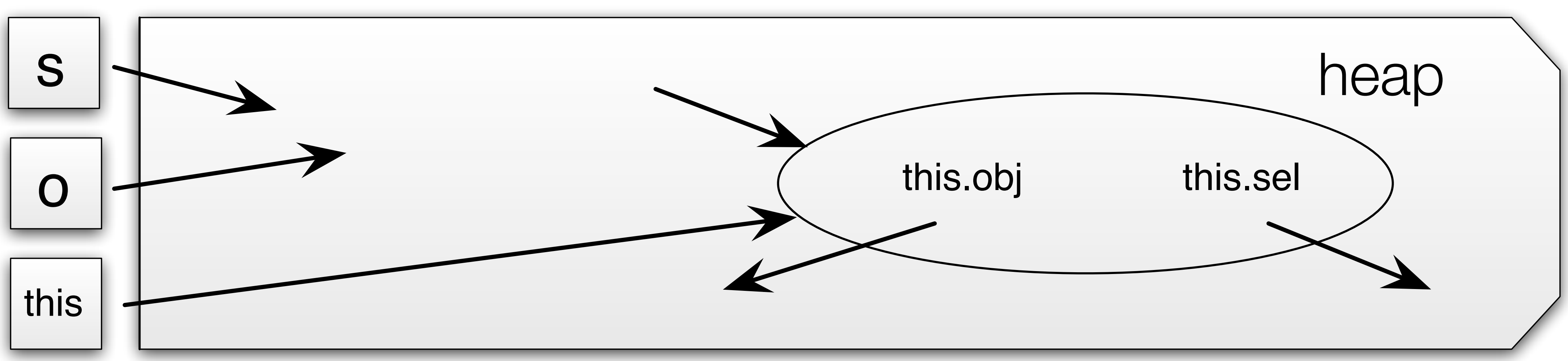
Symbolic Heap

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$\widetilde{H}$ okheap

Formula literal: **concretization** includes every subheap that is **not immediately type inconsistent**

Concrete State



Summarize heap locations that are **not immediately type-inconsistent** with **okheap**

Symbolic Heap

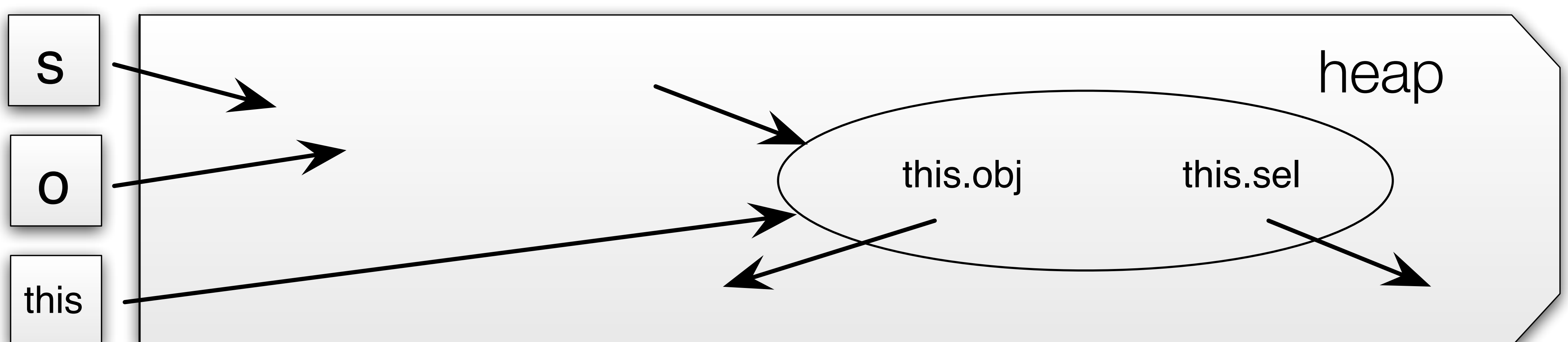
$\widetilde{H}$ okheap

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Formula literal: **concretization** includes every subheap that is **not immediately type inconsistent**

Describes storage **without explicitly enumerating it**

Concrete State



Summarize heap locations that are **not immediately type-inconsistent** with **okheap**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic Heap

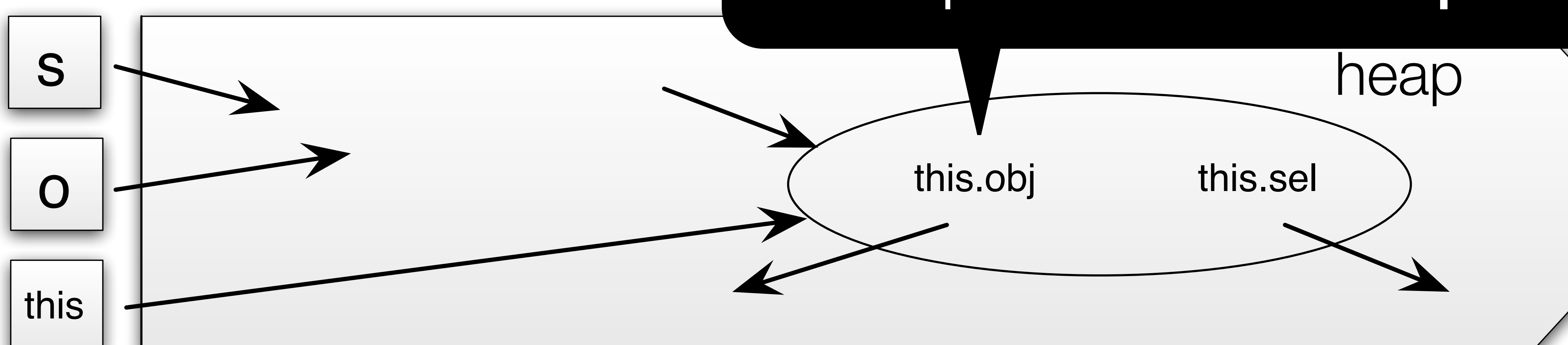
$\widetilde{H}$ okheap

Formula literal: **concretization** includes every subheap that is **not immediately type inconsistent**

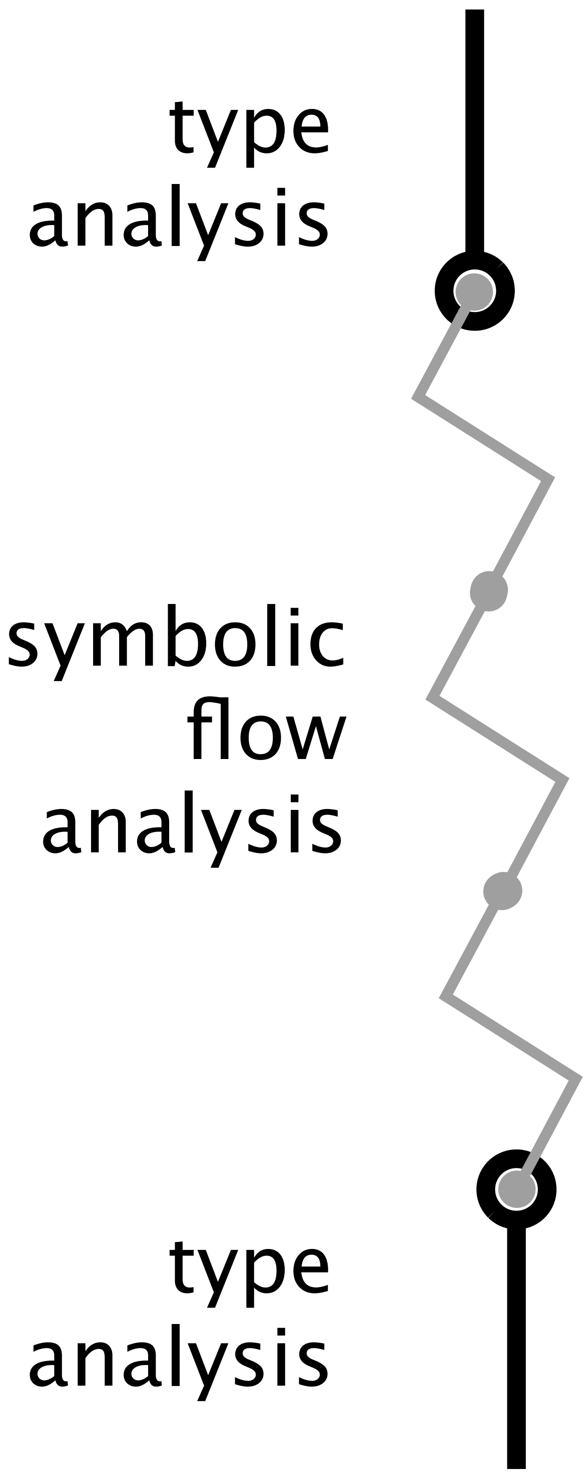
Describes storage **without explicitly enumerating it**

Concrete State

Immediately after switch, type **invariants still hold** so **okheap** represents **entire heap**



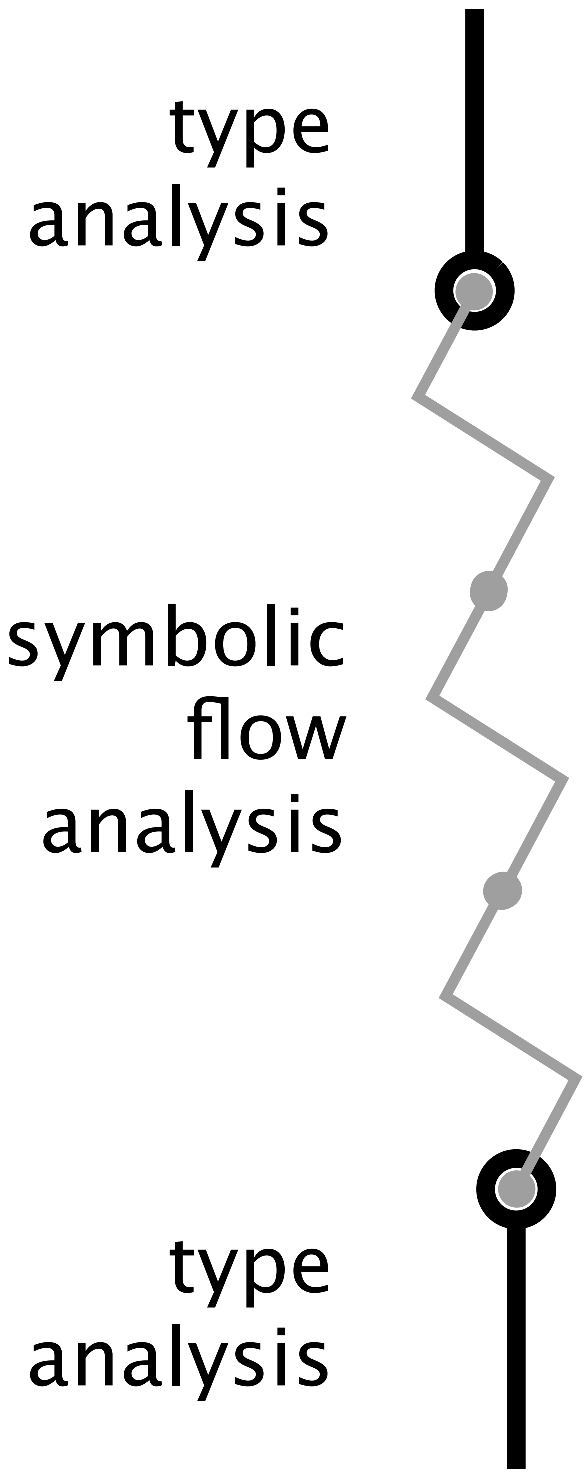
Key Contributions



1. **Translate** type invariant into symbolic state via **“symbolization”** of type environment

2. **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**

Key Contributions



1. **Translate** type invariant into symbolic state via **"symbolization"** of type environment

2. **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**

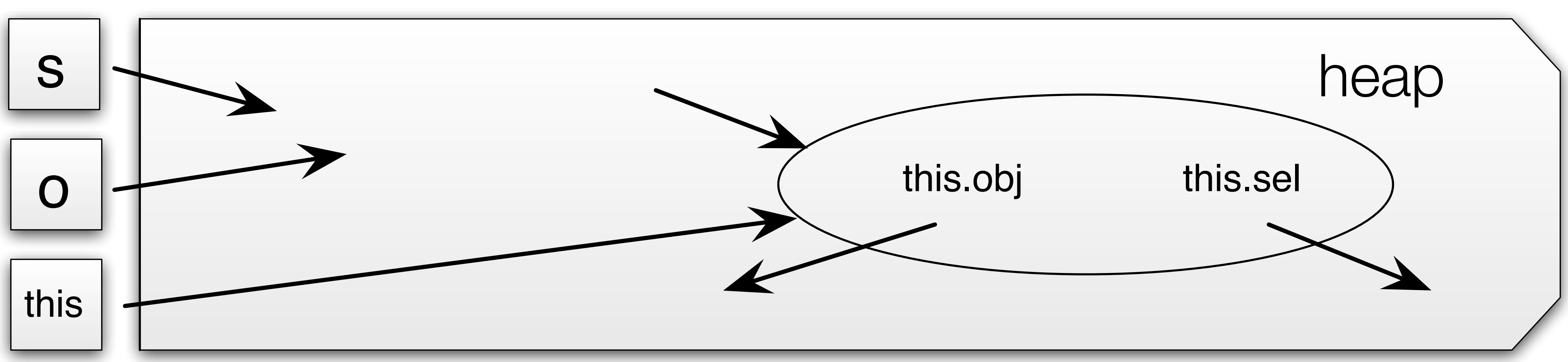
Leverage **heap type invariant** via **type-consistent materialization**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$$\widetilde{H}$$ okheap

Concrete State



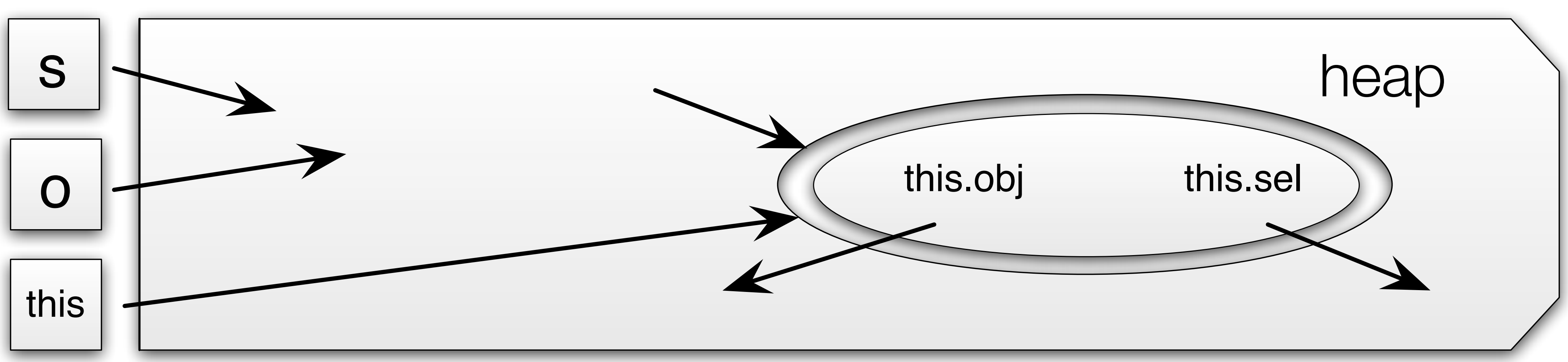
Leverage **heap type invariant** via **type-consistent materialization**

Materialize onto standard **separation-logic** explicit heap

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\widetilde{H} \quad \text{okheap} \; * \; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \widetilde{\text{sel}} * \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Concrete State



Leverage **heap type invariant** via **type-consistent materialization**

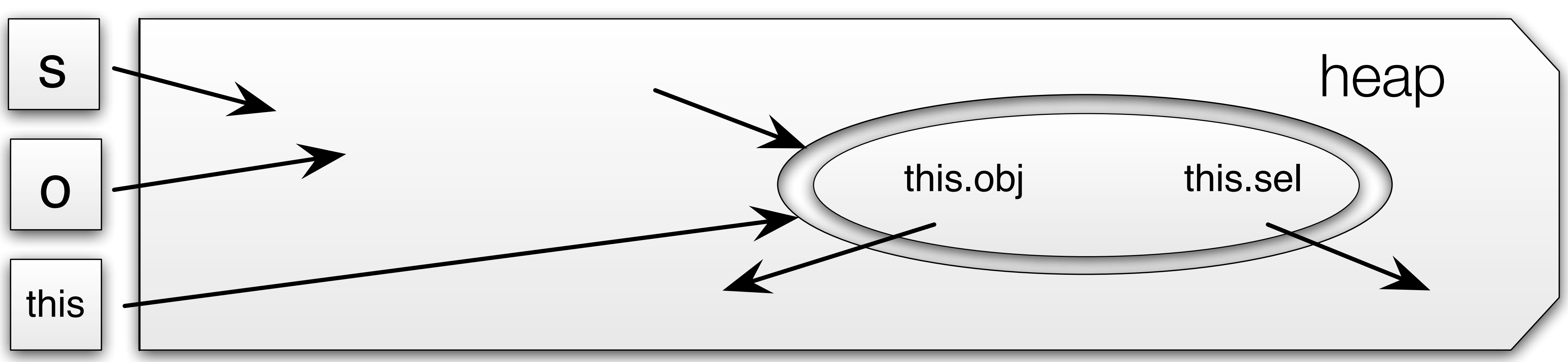
Materialize onto standard **separation-logic** explicit heap

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\widetilde{H} \qquad \text{okheap} \;*\; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \widetilde{\text{sel}} * \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Must-alias and **disalias** guarantee requires **case split** on **materialization**

Concrete State



Leverage **heap type invariant** via **type-consistent materialization**

Materialize onto standard **separation-logic** explicit heap

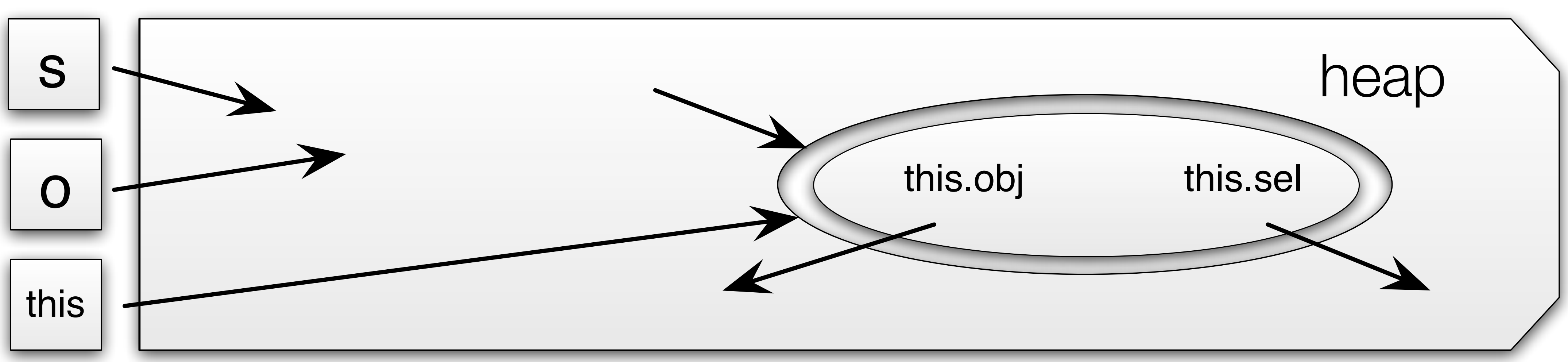
```
def update(
  this.sel
  this.obj
```

Materialized storage guaranteed to be **not immediately type-inconsistent**

$$\widetilde{H} \quad \text{okheap} \; * \; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \widetilde{\text{sel}} * \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Must-alias and **disalias** guarantee requires **case split** on **materialization**

Concrete State



Leverage **heap type invariant** via **type-consistent materialization**

Materialize onto standard **separation-logic** explicit heap

```
def update(
  this.sel
  this.obj
```

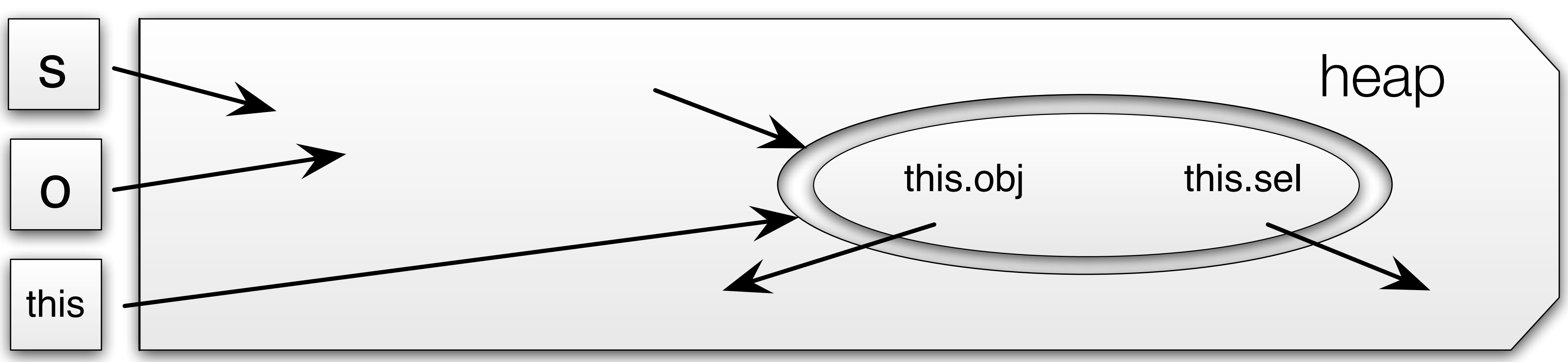
Materialized storage guaranteed to be **not immediately type-inconsistent**

$$\widetilde{H} \quad \text{okheap} \; * \; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \widetilde{\text{sel}} * \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Must-alias and **disalias** guarantee requires **case split** on **materialization**

Value stored in `obj` **responds to** value stored in `sel`

Concrete State



Leverage **heap type invariant** via **type-consistent materialization**

```
def update(
  this.sel :
  this.obj :
```

Materialize onto standard **separation-logic** explicit heap

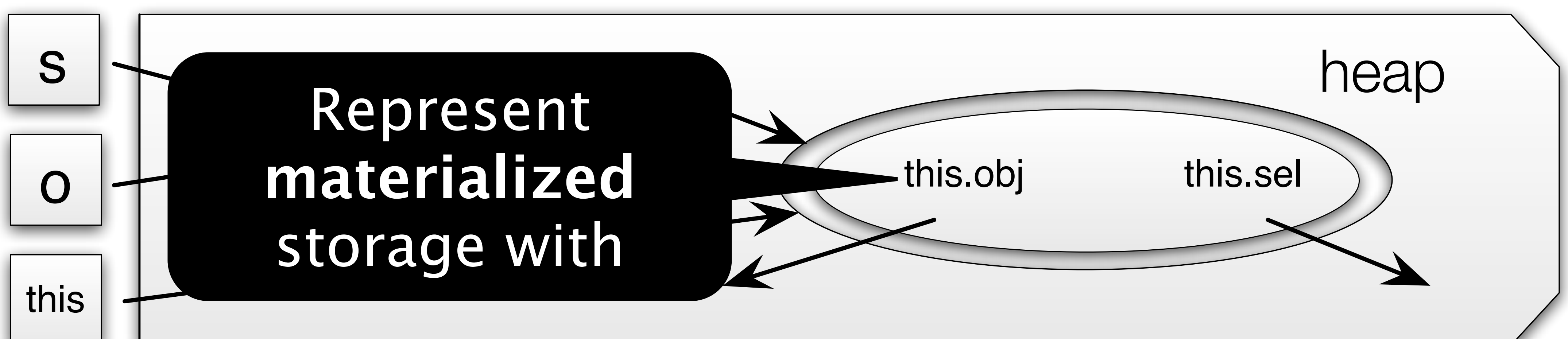
Materialized storage guaranteed to be **not immediately type-inconsistent**

$$\widetilde{H} \qquad \text{okheap} \;*\; \widetilde{\mathrm{this}} \mapsto \{\mathrm{sel} \mapsto \widetilde{\mathrm{sel}} * \mathrm{obj} \mapsto \widetilde{\mathrm{obj}}\}$$

Must-alias and **disalias** guarantee requires **case split** on **materialization**

Value stored in `obj` **responds to** value stored in `sel`

Concrete State



Represent **materialized** storage with

Leverage **heap type invariant** via **type-consistent materialization**

```
def update(
  this.sel
  this.obj
```

Materialize onto standard **separation-logic** explicit heap

Materialized storage guaranteed to be **not immediately type-inconsistent**

$$\widetilde{H} \qquad \text{okheap} \; * \; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \widetilde{\text{sel}} * \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Must-alias and **disalias** guarantee requires **case split** on **materialization**

Value stored in `obj` **responds to** value stored in `sel`

Concrete State

Analysis can **assume** that **type invariant initially holds on** all **materialized storage**

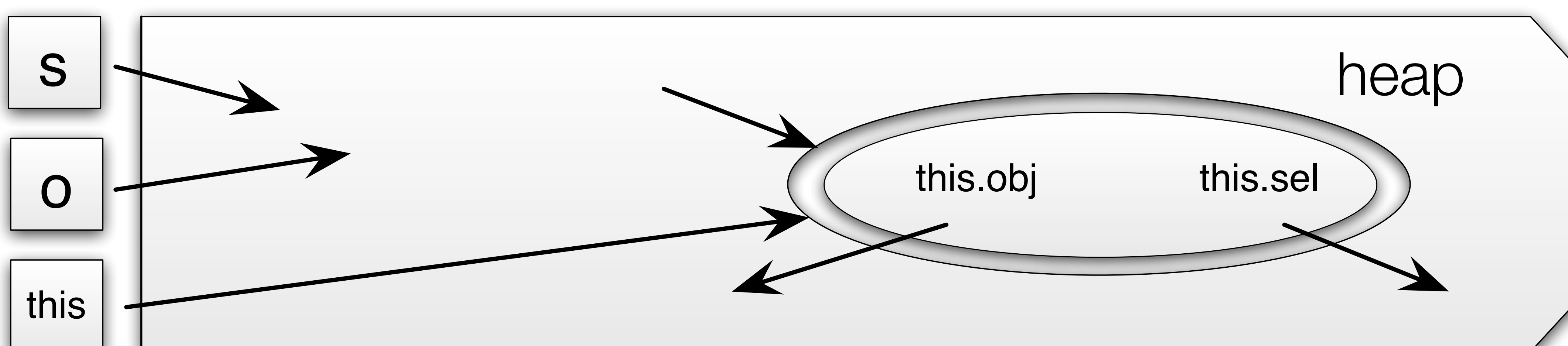
Strong updates on materialized storage to detect **invariant restoration**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$$\widetilde{H} \qquad \text{okheap} \; * \; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \widetilde{\text{sel}} * \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Concrete State



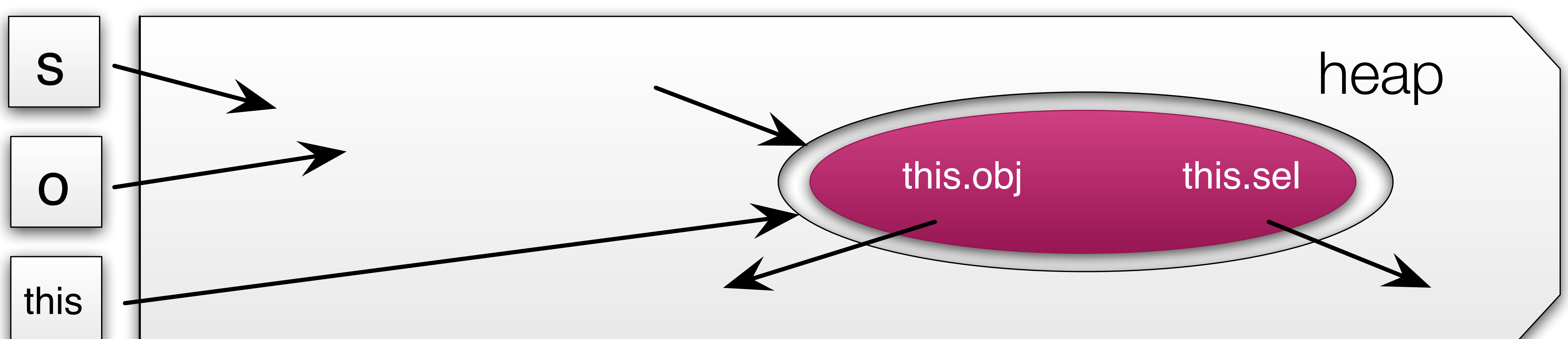
Strong updates on materialized storage to detect **invariant restoration**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$$\widetilde{H} \quad \text{okheap} \; * \; \widetilde{\mathrm{this}} \mapsto \{\mathrm{sel} \mapsto \tilde{\mathrm{s}} * \mathrm{obj} \mapsto \widetilde{\mathrm{obj}}\}$$

Concrete State



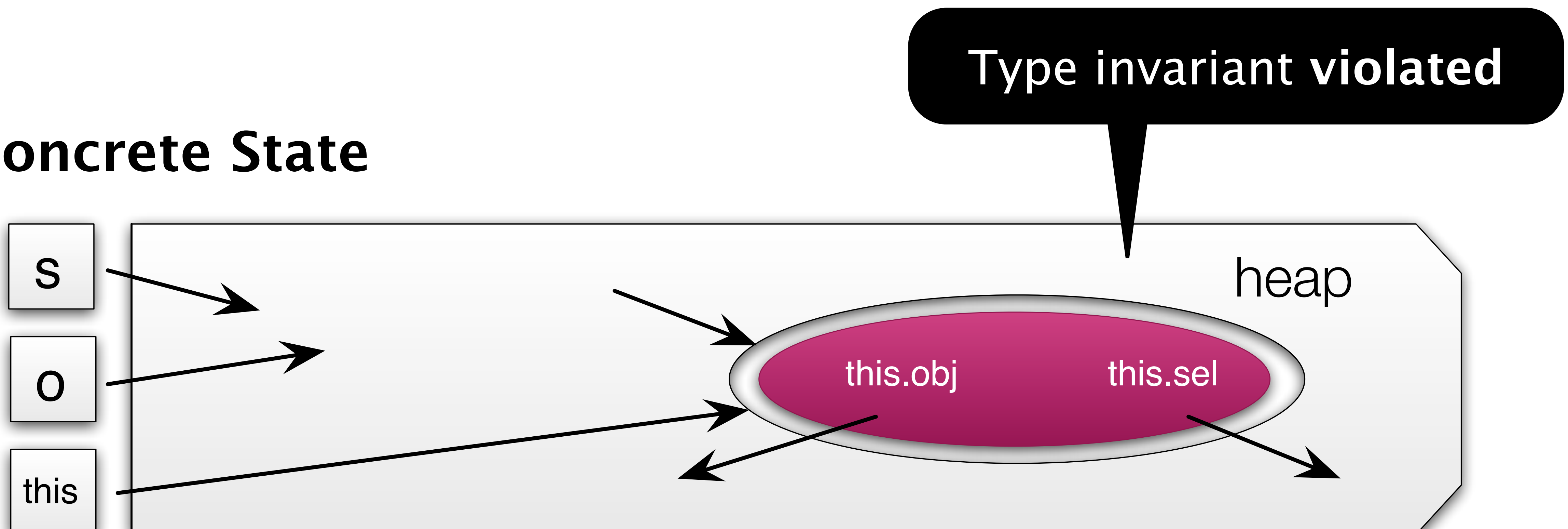
Strong updates on materialized storage to detect **invariant restoration**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$$\widetilde{H} \quad \text{okheap} * \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \tilde{\text{s}} * \text{obj} \mapsto \widetilde{\text{obj}}\}$$

Concrete State



Strong updates on materialized storage to detect **invariant restoration**

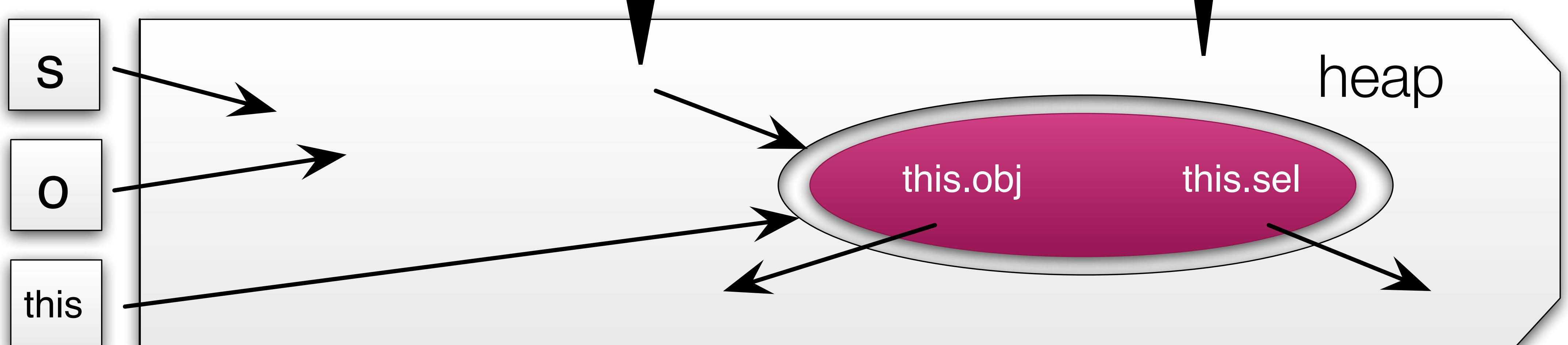
```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$$\widetilde{H} \quad \text{okheap} \; * \; \widetilde{\mathrm{this}} \mapsto \{\mathrm{sel} \mapsto \tilde{\mathrm{s}} * \mathrm{obj} \mapsto \widetilde{\mathrm{obj}}\}$$

Surprising: can **soundly** permit **pointers in** and **out** of the region that is **not immediately type-inconsistent**

Type invariant **violated**



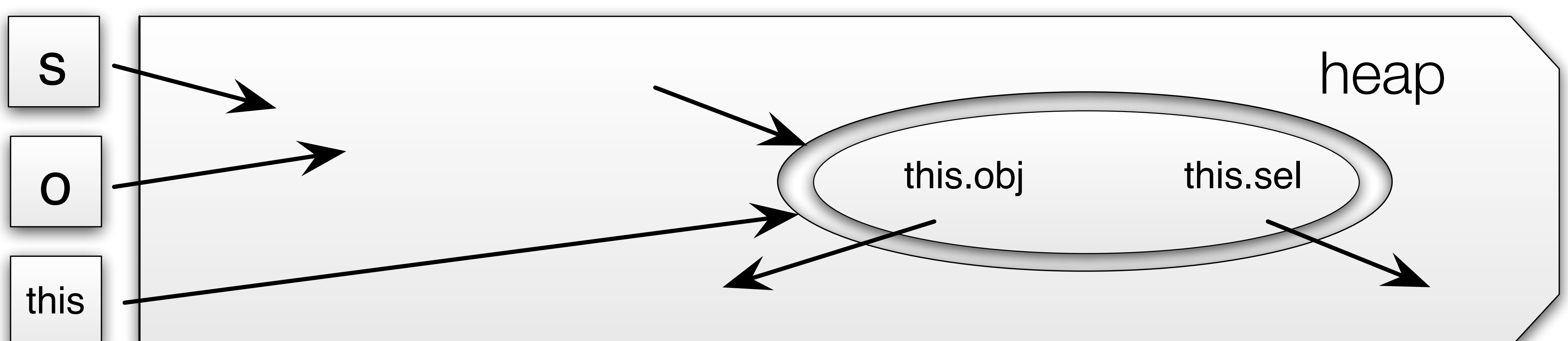
Strong updates on materialized storage to detect **invariant restoration**

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$$\widetilde{H} \qquad \text{okheap} \; * \; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \; \tilde{\text{s}} \; * \text{obj} \mapsto \; \tilde{\text{o}} \; \}$$

Concrete State



Strong updates on materialized storage to detect **invariant restoration**

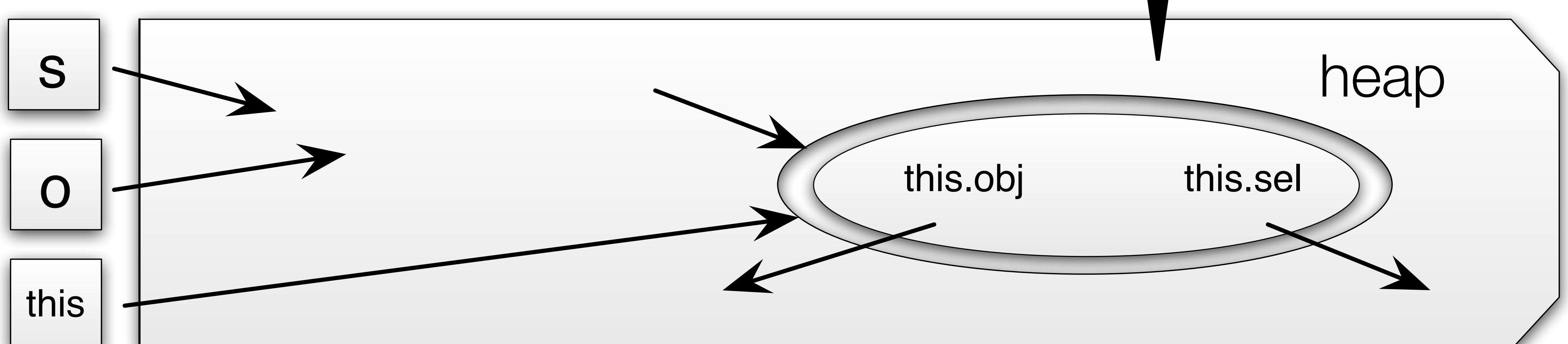
```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$$\widetilde{H} \qquad \text{okheap} \; * \; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \; \tilde{\text{s}} \; * \text{obj} \mapsto \; \tilde{\text{o}} \; \}$$

No longer immediately **type-inconsistent**

Concrete State



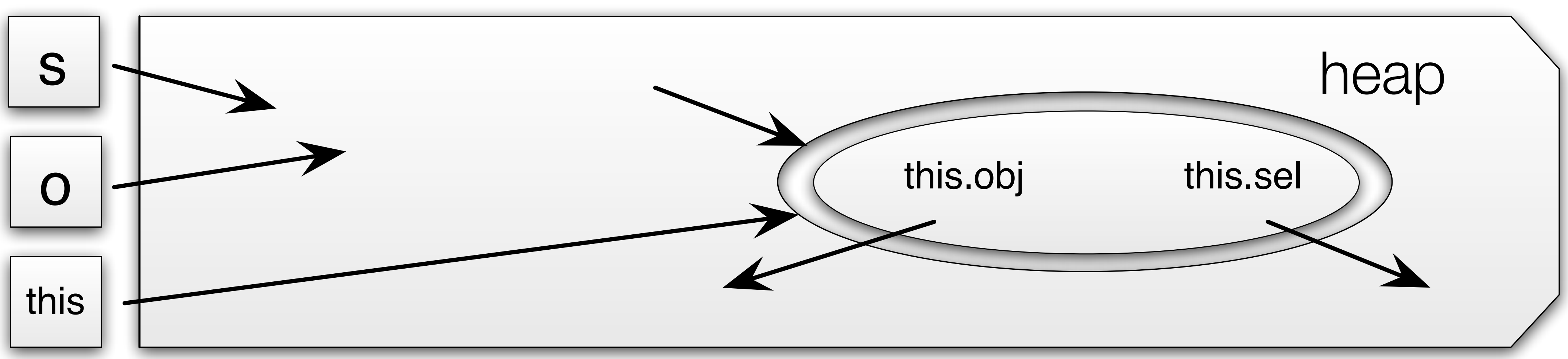
Safely **summarize** storage that is not immediately type inconsistent

Symbolic State

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$$\widetilde{H} \qquad \text{okheap} \;*\; \widetilde{\text{this}} \mapsto \{\text{sel} \mapsto \tilde{\text{s}} * \text{obj} \mapsto \tilde{\text{o}}\}$$

Concrete State



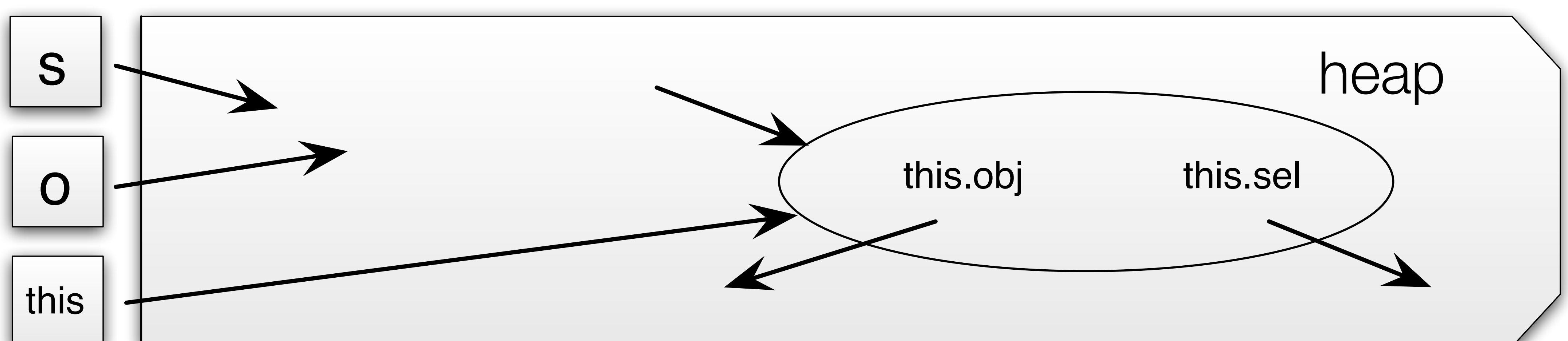
Safely **summarize** storage that is not immediately type inconsistent

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$\widetilde{H}$ okheap

Concrete State



Safely **summarize** storage that is not immediately type inconsistent

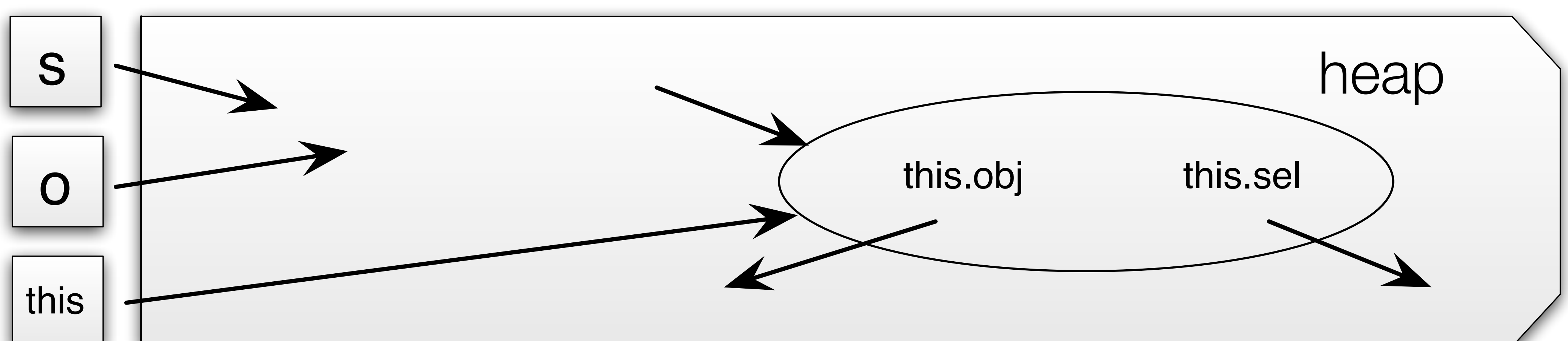
Symbolic State

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

$\widetilde{H}$ okheap

Only need to reason **precisely** about **part of heap where invariant broken,** so helps **manage alias explosion**

Concrete State



Safely **summarize** storage that is not immediately type inconsistent

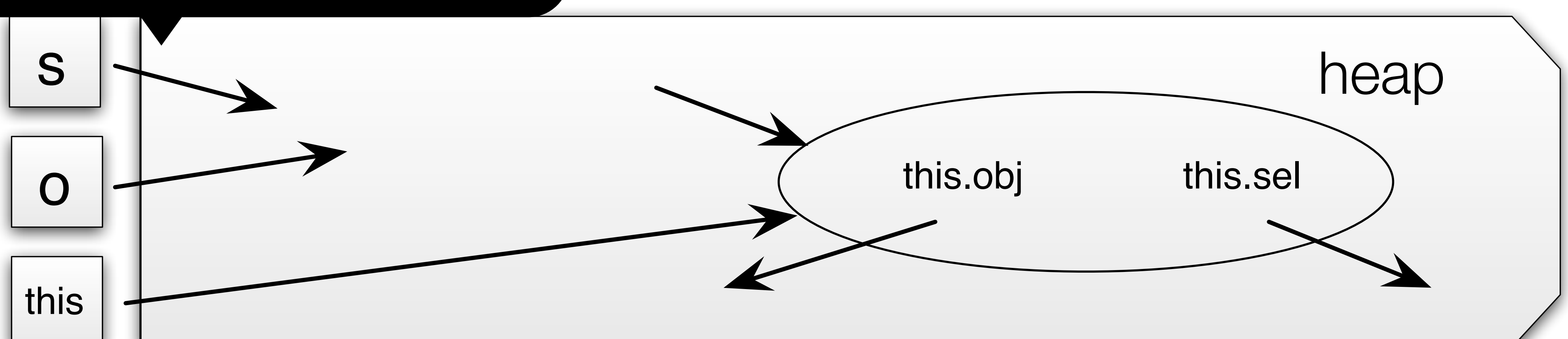
```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

Symbolic State

$\widetilde{H}$ okheap

Only need to reason **precisely** about **part of heap where invariant broken,** so helps **manage alias explosion**

Entire heap is type consistent so safe to **return to type checking**



Safely **summarize** storage that is not immediately type inconsistent

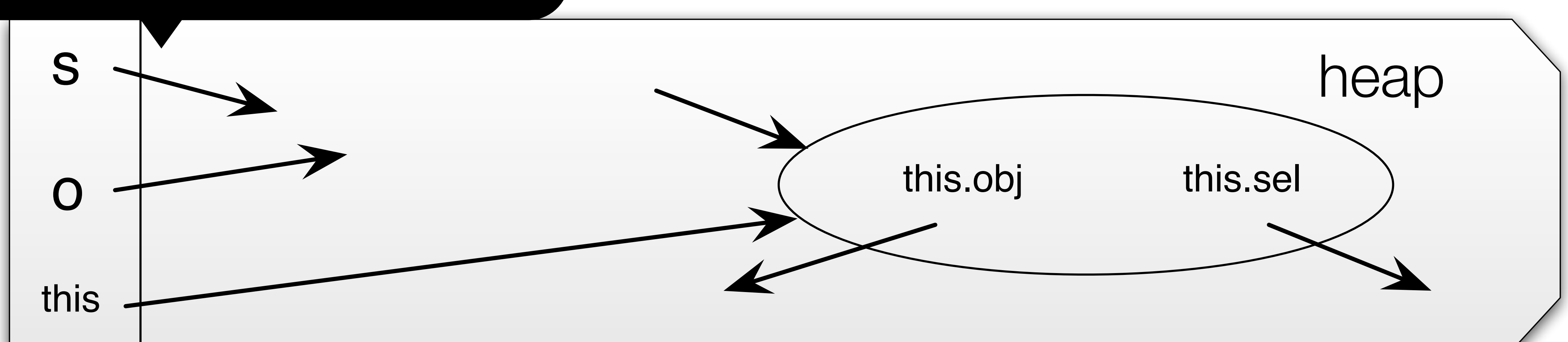
Symbolic State

```
def update(s:Str, o:Obj | r2 s)
  this.sel = s
  this.obj = o
```

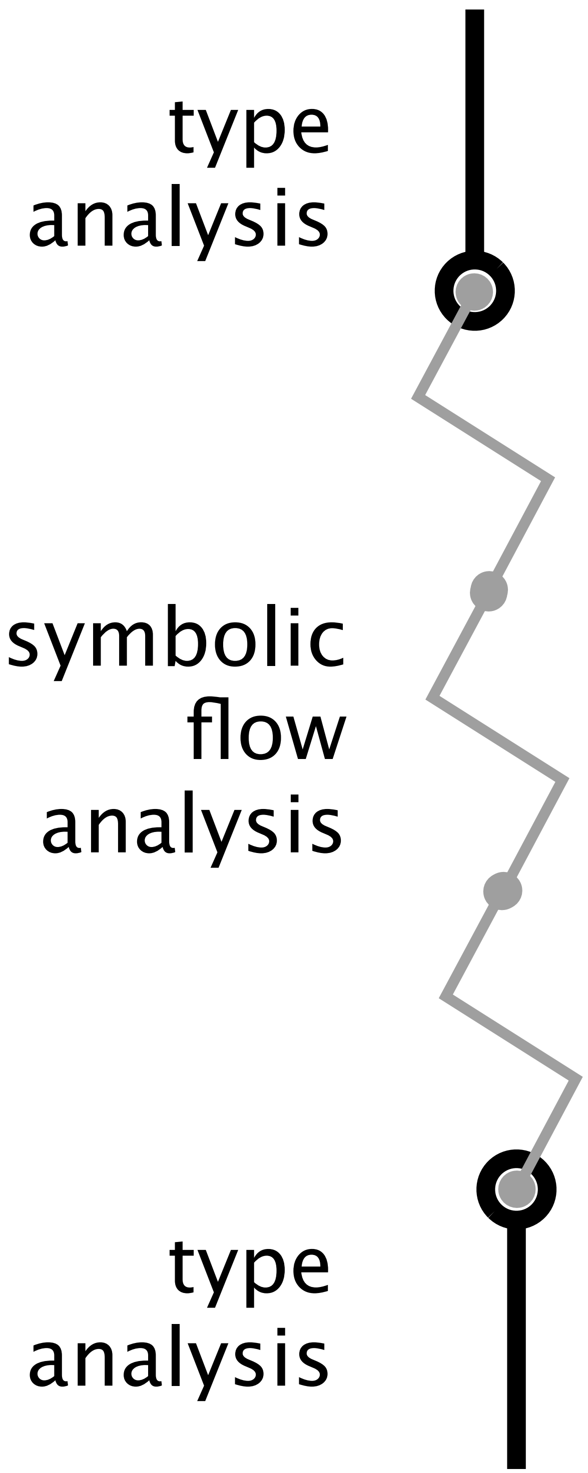
$\widetilde{H}$ okheap

Only need to reason **precisely** about **part of heap where invariant broken,** so helps **manage alias explosion**

Entire heap is type consistent so safe to **return to type checking**



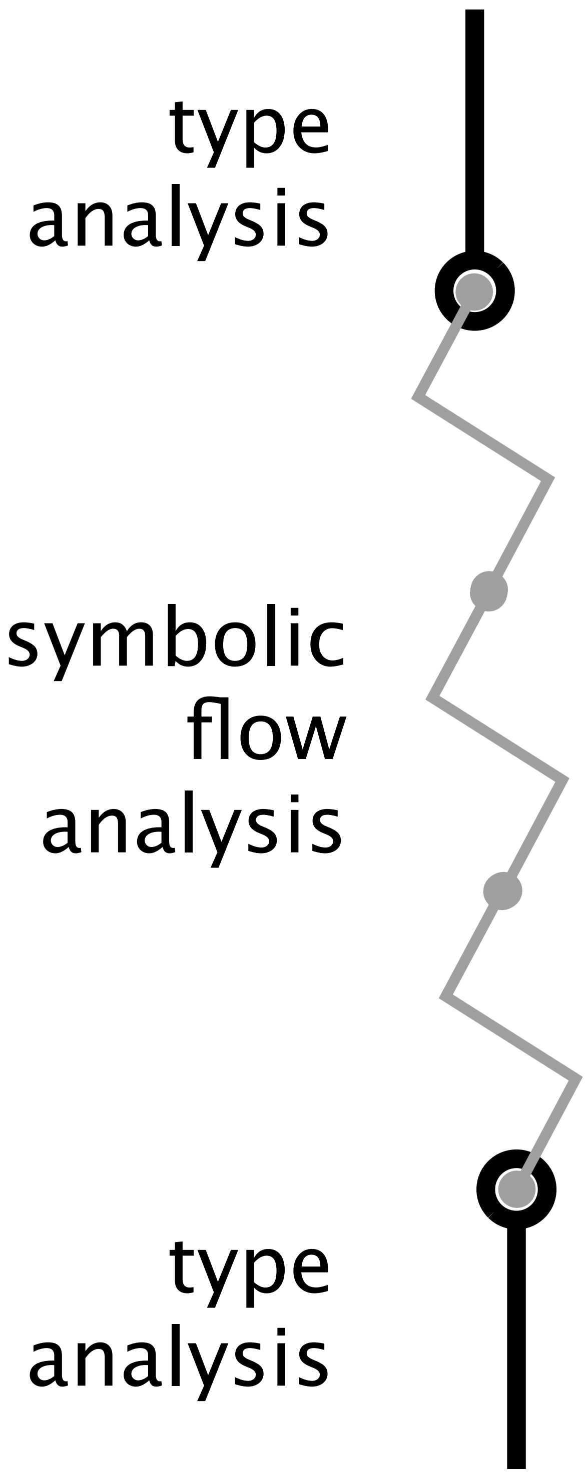
Key Contributions



1. **Translate** type invariant into symbolic state via **“symbolization”** of type environment

2. **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**

Key Contributions



1. **Translate** type invariant into symbolic state via **"symbolization"** of type environment

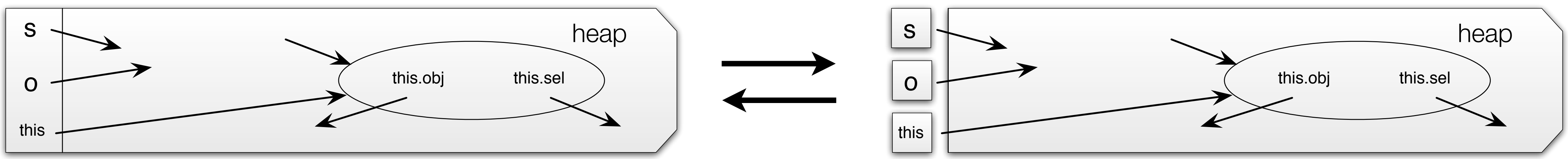
2. **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**

Fissile Type Analysis is **sound**

Fissile Type Analysis is **sound**

Theorem (Soundness of Handoff).

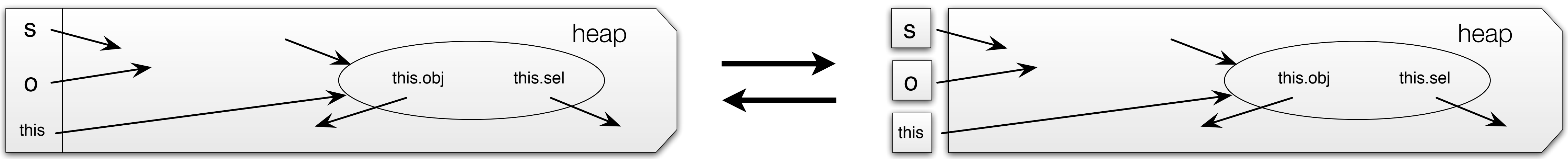
The **entire** state is **type-consistent** iff all locations are **not immediately type-inconsistent**.



Fissile Type Analysis is **sound**

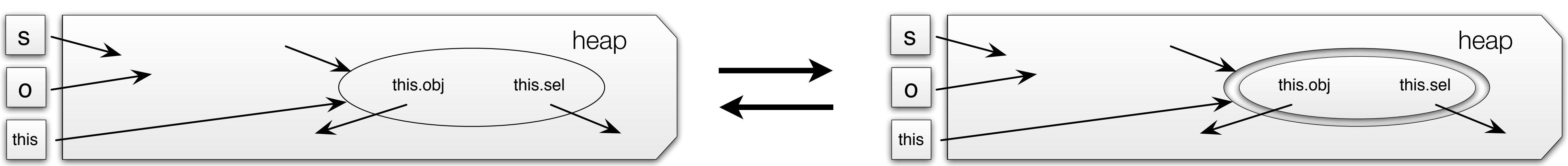
Theorem (Soundness of Handoff).

The **entire** state is **type-consistent** iff all locations are **not immediately type-inconsistent**.



Theorem (Soundness of Materialization/Summarization).

Storage that is **not immediately type-inconsistent** can be safely materialized and summarized into **okheap**.



Evaluation

Analysis mechanics: How often is **symbolic reasoning** required?

Precision: What is improvement over **flow-insensitive checking** alone?

Cost: What is the cost of analysis in **running time**?

Case Study: Reflection in Objective-C

Prototype analysis implementation

- Plugin for **clang** static analyzer in C++

9 **Objective-C** benchmarks

- 6 **libraries** and 3 **applications**
- **1,000** to **176,000** lines of code

Manual **type annotations**

- 76 r2 annotations on **system libraries**
- 136 annotations on **benchmark code**

Case Study: Reflection in Objective-C

Prototype analysis implementation

- Plugin for **clang** static analyzer in C++

9 **Objective-C** benchmarks

- 6 **libraries** and 3 **applicatio**
- **1,000** to **176,000** lines of cod

Including **Skim, Adium,** and **OmniGraffle**

Manual **type annotations**

- 76 r2 annotations on **system libraries**
- 136 annotations on **benchmark code**

Analysis **mechanics**

	size		
benchmark	**(loc)**	**symbolic sections**	**maximum materializations**
OAuth	1248	7	1
SCRecorder	2716	2	2
ZipKit	3301	0	0
Sparkle	5289	3	1
ASIHTTPRequest	14620	59	2
OmniFrameworks	160769	7	1
Vienna	37327	28	2
Skim	60211	0	0
Adium	176629	16	1
combined	461080	125	2

Analysis **mechanics**

	size		
benchmark	**(loc)**	**symbolic sections**	**maximum materializations**
OAuth	1248	7	1
	2716	2	2
	3301	0	0
	5289	3	1
ASIHTTPRequest	14620	59	2
OmniFrameworks	160769	7	1
Vienna	37327	28	2
Skim	60211	0	0
Adium	176629	16	1
combined	461080	125	2

Number of **successful** switches to **symbolic analysis** and **back**

Analysis **mechanics**

Number of **successful** switches to **symbolic analysis** and **back**

benchmark	**size (loc)**	**symbolic sections**	**maximum materializations**
OAuth	1248	7	1
[illegible]	2716	2	2
[illegible]	3301	0	0
[illegible]	5289	3	1
ASIHTTPRequest	14620	59	2
OmniFrameworks	160769	7	1
Vienna	37327	28	2
Skim	60211	0	0
Adium	176629	16	1
combined	461080	125	2

A **significant** number of **switches**:
Approach successfully handles when **developers break** and **restore** global **invariants**

Analysis **mechanics**

	size		
benchmark	**(loc)**	**symbolic [illegible]**	**maximum materializations**
[illegible]	[illegible]	7	1
[illegible]	[illegible]	2	2
[illegible]	[illegible]	0	0
[illegible]	[illegible]	3	1
[illegible]	[illegible]	59	2
OMNIFRAMEWORKS	160769	7	1
VIENNA	37327	28	2
SKIM	60211	0	0
ADIUM	176629	16	1
combined	461080	125	2

Maximum number of **simultaneous** materialized storage locations

A **significant** number of **switches**:
Approach successfully handles when **developers break** and **restore** global **invariants**

Analysis **mechanics**

	size		
benchmark	**(loc)**	**symbolic sectio[illegible]**	**maximum materializations**
[illegible]	[illegible]	7	1
[illegible]	[illegible]	2	2
[illegible]	[illegible]	0	0
[illegible]	[illegible]	3	1
[illegible]	[illegible]	59	2
OmniFrameworks	160769	7	1
Vienna	37327	28	2
Skim	60211	0	0
Adium	176629	16	1
combined	461080	125	2

Maximum number of **simultaneous** materialized storage locations

A **significant** number of **switches**:
Approach successfully handles when **developers break** and **restore** global **invariants**

At most **2 simultaneous materializations**:
Aliasing case splits will **not blow up**

Analysis **mechanics**

	size		
benchmark	**(loc)**	**symbolic sections**	**maximum materializations**
OAuth	1248	7	1
SCRecorder	2716	2	2
ZipKit	3301	0	0
Sparkle	5289	3	1
ASIHTTPRequest	14620	59	2
OmniFrameworks	160769	7	1
Vienna	37327	28	2
Skim	60211	0	0
Adium	176629	16	1
combined	461080	125	2

A **significant** number of **switches**:
Approach successfully handles when **developers break** and **restore** global **invariants**

At most **2 simultaneous materializations**:
Aliasing case splits will **not blow up**

Analysis **mechanics**

	size		
benchmark	**(loc)**	**symbolic sections**	**maximum materializations**
OAuth	1248	7	1
SCRecorder	2716	2	2
ZipKit	3301	0	0
Sparkle	5289	3	1
ASIHTTPRequest	14620	59	2
OmniFrameworks	160769	7	1
Vienna	37327	28	2
Skim	60211	0	0
Adium	176629	16	1
		125	2

Approaches limited to **one-at-a-time materialization not sufficient**

A **significant** number of **switches**:
Approach successfully handles when **developers break** and **restore** global **invariants**

At most **2 simultaneous materializations**:
Aliasing case splits will **not blow up**

Precision

	size		false alarms	
benchmark	(loc)	reflective call sites	flow-insensitive	almost-everywhere
OAuth	1248	7	7	2 (–71%)
SCRecorder	2716	12	2	0 (–100%)
ZipKit	3301	28	0	0 (–)
Sparkle	5289	40	4	1 (–75%)
ASIHTTPRequest	14620	68	50	10 (–80%)
OmniFrameworks	160769	192	82	74 (–10%)
Vienna	37327	186	59	38 (–36%)
Skim	60211	207	43	43 (–0%)
Adium	176629	587	87	70 (–20%)
combined	461080	1327	334	238 (–29%)

Precision

	size		false alarms	
benchmark	**(loc)**	**reflective call sites**	**flow-insensitive**	**almost-everywhere**
OAuth	1248	7	7	2 (–71%)
SCRecorder	2716	12	2	0 (–100%)
ZipKit	3301	28	0	0 (–)
Sparkle	5289	40	4	1 (–75%)
ASIHTTPRequest	14620	68	50	10 (–80%)
OmniFrameworks	160769	192	82	74 (–10%)
Vienna	37327	186	59	38 (–36%)
Skim	60211	207	43	43 (–0%)
Adium	176629	587	87	70 (–20%)
combined	461080	1327	334	238 (–29%)

Baseline: standard, **flow-insensitive** type analysis – no switching

Precision

	size		false alarms	
benchmark	**(loc)**	**reflective call sites**	**flow-insensitive**	**almost-everywhere**
OAUTH	1248	7	7	2 (–71%)
SCRECORDER	2716	12	2	0 (–100%)
ZIPKIT	3301	28	0	0 (–)
SPARKLE	5289	40	4	1 (–75%)
ASIHTTPREQUEST	14620	68	50	10 (–80%)
OMNIFRAMEWORKS	160769	192	82	74 (–10%)
VIENNA	37327	186	59	38 (–36%)
SKIM	60211	207	43	43 (–0%)
ADIUM	176629	587	87	70 (–20%)
combined	461080	1327	334	238 (–29%)

Baseline: standard, **flow-insensitive** type analysis – no switching

Precision

	size		false alarms	
benchmark	**(loc)**	**reflective call sites**	**flow-insensitiv e**	**almost-everywhere**
OAuth	1248	7	7	2 (–71%)
SCRecorder	2716	12	2	0 (–100%)
ZipKit	3301	28	0	0 (–)
Sparkle	5289	40	4	1 (–75%)
ASIHTTPRequest	14620	68	50	10 (–80%)
OmniFrameworks	160769	192	82	74 (–10%)
Vienna	37327	186	59	38 (–36%)
Skim	60211	207	43	43 (–0%)
Adium	176629	587	87	70 (–20%)
combined	461080	1327	334	238 (–29%)

Baseline: standard, **flow-insensitive** type analysis – no
sw

Almost everywhere techniques show
29% improvement in false alarms

Precision

	size		false alarms	
benchmark	(loc)	reflective call sites	flow-insensitive	almost-everywhere
OAuth	1248	7	7	2 (–71%)
	2716	12	2	0 (–100%)
	3301	28	0	0 (–)
	5289	40	4	1 (–75%)
	14620	68	50	10 (–80%)
	160769	192	82	74 (–10%)
Vienna	37327	186	59	38 (–36%)
Skim	60211	207	43	43 (–0%)
Adium	176629	587	87	70 (–20%)
combined	461080	1327	334	238 (–29%)

Also found a real reflection **bug** in Vienna, which we reported and which was **fixed**

Baseline: standard, **flow-insensitive** type analysis – no sw

Almost everywhere techniques show **29% improvement** in false alarms

Cost: Analysis **time**

	size	analysis time	
benchmark	**(loc)**	**Time**	**Rate (kloc/s)**
OAuth	1248	0.24s	5.3
SCRecorder	2716	0.28s	10.8
ZipKit	3301	0.10s	33.0
Sparkle	5289	0.67s	7.9
ASIHTTPRequest	14620	0.50s	27.2
OmniFrameworks	160769	4.25s	37.8
Vienna	37327	2.79s	13.4
Skim	60211	2.49s	24.1
Adium	176629	8.79s	20.1
combined	461080	20.09s	23.0

Cost: Analysis **time**

	size	analysis time	
benchmark	**(loc)**	**Time**	**Rate (kloc/s)**
OAuth	1248	0.24s	5.3
SCRecorder	[illegible]	0.28s	10.8
ZipKit	3301	0.10s	33.0
[illegible]	5289	0.67s	7.9
[illegible]	14620	0.50s	27.2
[illegible]	160769	4.25s	37.8
[illegible]	37327	2.79s	13.4
Skim	60211	2.49s	24.1
Adium	176629	8.79s	20.1
combined	461080	20.09s	23.0

Includes analysis time but **not parsing, base type checking**

Cost: Analysis **time**

	size	analysis time	
	(loc)	Time	Rate (kloc/s)
[illegible]	1248	0.24s	5.3
[illegible]	[illegible]	0.28s	10.8
[illegible]	3301	0.10s	33.0
[illegible]	5289	0.67s	7.9
[illegible]	14620	0.50s	27.2
[illegible]	160769	4.25s	37.8
[illegible]	37327	2.79s	13.4
[illegible]	60211	2.49s	24.1
ADIUM	176629	8.79s	20.1
combined	461080	20.09s	23.0

Does **not** include **system headers**

Includes analysis time but **not parsing, base type checking**

Cost: Analysis **time**

benchmark	size (loc)	analysis time: Time	analysis time: Rate (kloc/s)
OAuth	1248	0.24s	5.3
SCRecorder	2716	0.28s	10.8
ZipKit	3301	0.10s	33.0
Sparkle	5289	0.67s	7.9
ASIHTTPRequest	14620	0.50s	27.2
OmniFrameworks	160769	4.25s	37.8
Vienna	37327	2.79s	13.4
Skim	60211	2.49s	24.1
Adium	176629	8.79s	20.1
combined	461080	20.09s	23.0

Fast: 5 to 38 kloc/s with most time spent **analyzing system headers**

Cost: Analysis **time**

	size	analysis time	
benchmark	**(loc)**	**Time**	**Rate (kloc/s)**
OAuth	1248	0.24s	5.3
SCRecorder	2716	0.28s	10.8
ZipKit	3301	0.10s	33.0
Sparkle	5289	0.67s	7.9
ASIHTTPRequest	14620	0.50s	27.2
OmniFrameworks	160769	4.25s	37.8
Vienna	37327	2.79s	13.4
Skim	60211	2.49s	24.1
Adium	176629	8.79s	20.1
combined	461080	20.09s	23.0

Fast: 5 to 38 kloc/s with most time spent **analyzing system headers**

Interactive speeds

Cost: Analysis **time**

benchmark	size (loc)	analysis time: Time	analysis time: Rate (kloc/s)
OAuth	1248	0.24s	5.3
SCRecorder	2716	0.28s	10.8
ZipKit	3301	0.10s	33.0
Sparkle	5289	0.67s	7.9
ASIHTTPRequest	14620	0.50s	27.2
OmniFrameworks	160769	4.25s	37.8
Vienna	37327	2.79s	13.4
Skim	60211	2.49s	24.1
Adium	176629	8.79s	20.1
combined	461080	20.09s	23.0

Fast: 5 to 38 kloc/s with most time spent **analyzing system headers**

Higher rate for projects with **larger translation units**

Cost: Analysis **time**

	size	analysis time	
benchmark	**(loc)**	**Time**	**Rate (kloc/s)**
OAuth	1248	0.24s	5.3
SCRecorder	2716	0.28s	10.8
ZipKit	3301	0.10s	33.0
Sparkle	5289	0.67s	7.9
ASIHTTPRequest	14620	0.50s	27.2
OmniFrameworks	160769	4.25s	37.8
Vienna	37327	2.79s	13.4
Skim	60211	2.49s	24.1
Adium	176629	8.79s	20.1
combined	461080	20.09s	23.0

Fast: 5 to 38 kloc/s with most time spent

Maintains key benefit of flow-insensitive analyses: **speed**

tion

Summary

- Check **almost everywhere** heap invariants with **intertwined type** and **symbolic flow analysis**
  - **Translate** type environment into symbolic state with **symbolization**
  - **Leverage** heap type invariant during symbolic analysis via **type-consistent materialization** and **summarization**
- Approach is **very fast** and **scales to large programs**

Fissile Type Analysis yields significant **precision improvement** at **little cost** in performance

Fissile Type Analysis yields significant **precision improvement** at **little cost** in performance

Why?

Fissile Type Analysis yields significant **precision improvement** at **little cost** in performance

Why?

Because **almost-everywhere** invariants hold **almost everywhere**

Manual annotation **burden**

	size			false alarms				analysis time	
benchmark	**(loc)**	**reflective call sites**	**annotation count**	**flow-insens.**	**almost-everwhere**	**symbolic sections**	**max. mat-erializations**	**Time**	**Rate (kloc/s)**
OAUTH	1248	7	5	7	2 (–71%)	7	1	0.30s	4.1
SCRECORDER	2716	12	9	2	0 (–100%)	2	2	0.28s	9.8
ZIPKIT	3301	28	0	0	0 (–)	0	0	0.10s	33.8
SPARKLE	5289	40	0	4	1 (–75%)	3	1	0.78s	6.8
ASIHTTPREQUEST	14620	68	2	50	10 (–80%)	59	2	0.15s	90.2
OMNIFRAMEWORKS	160769	192	49	82	74 (–10%)	7	1	4.61s	34.9
VIENNA	37327	186	24	59	38 (–36%)	28	2	2.57s	14.5
SKIM	60211	207	7	43	43 (–0%)	0	0	2.55s	23.6
ADIUM	176629	587	40	87	70 (–20%)	16	1	7.50s	23.6
combined	461080	1327	136	334	238 (–29%)	125	2	18.83	24.5

Manual annotation **burden**

	size			false alarms				analysis time	
benchmark	**(loc)**	**reflective call sites**	**annotation count**	**flow-insens.**	**almost-everwhere**	**symbolic sections**	**max. mat-erializations**	**Time**	**Rate (kloc/s)**
OAuth	1248	7	5	7	2 (−71%)	7	1	0.30s	4.1
SCRecorder	2716	12	9	2	0 (−100%)	2	2	0.28s	9.8
ZipKit	3301	28	0	0	0 (−)	0	0	0.10s	33.8
Sparkle	5289	40	0	4	1 (−75%)	3	1	0.78s	6.8
ASIHTTPRequest	14620	68	2	50	10 (−80%)	59	2	0.15s	90.2
OmniFrameworks	160769	192	[illegible]	82	74 (−10%)	7	1	4.61s	34.9
Vienna	37327	186	24	59	38 (−36%)	28	2	2.57s	14.5
Skim	60211	207	7	43	43 (−0%)	0	0	2.55s	23.6
Adium	176629	587	40	87	70 (−20%)	16	1	7.50s	23.6
combined	461080	1327	136	334	238 (−29%)	125	2	18.83	24.5

essentially zero for **clients** of reflection

Manual annotation **burden**

	size			false alarms				analysis time	
benchmark	**(loc)**	**reflective call sites**	**annotation count**	**flow-insens.**	**almost-everwhere**	**symbolic sections**	**max. mat-erializations**	**Time**	**Rate (kloc/s)**
OAUTH	1248	7	5	7	2 (–71%)	7	1	0.30s	4.1
SCRECORDER	2716	12	9	2	0 (–100%)	2	2	0.28s	9.8
ZIPKIT	3301	28	0	0	0 (–)	0	0	0.10s	33.8
SPARKLE	5289	40	0	4	1 (–75%)	3	1	0.78s	6.8
ASIHTTPREQUEST	14620	68	2	50	10 (–80%)	59	2	0.15s	90.2
OMNIFRAMEWORKS	160769	192	49	82	74 (–10%)	7	1	4.61s	34.9
VIENNA	37327	186	24	59	38 (–36%)	28	2	2.57s	14.5
SKIM	60211	207	7	43	43 (–0%)	0	0	2.55s	23.6
ADIUM	176629	587	40	87	70 (–20%)	16	1	7.50s	23.6
combined	461080	1327	136	334	238 (–29%)	125	2	18.83	24.5

essentially zero for **clients** of reflection

higher for frameworks exporting **reflective interfaces**

Manual annotation **burden**

	size			false alarms				analysis time	
benchmark	**(loc)**	**reflective call sites**	**annotation count**	**flow-insens.**	**almost-everwhere**	**symbolic sections**	**max. mat-erializations**	**Time**	**Rate (kloc/s)**
OAuth	1248	7	5	7	2 (–71%)	7	1	0.30s	4.1
SCRecorder	2716	12	9	2	0 (–100%)	2	2	0.28s	9.8
ZipKit	3301	28	0	0	0 (–)	0	0	0.10s	33.8
Sparkle	5289	40	0	4	1 (–75%)	3	1	0.78s	6.8
ASIHTTPRequest	14620	68	2	50	10 (–80%)	59	2	0.15s	90.2
OmniFrameworks	160769	192	49	82	74 (–10%)	7	1	4.61s	34.9
Vienna	37327	186	24	59	38 (–36%)	28	2	2.57s	14.5
Skim	60211	207	7	43	43 (–0%)	0	0	2.55s	23.6
Adium	176629	587	40	87	70 (–20%)	16	1	7.50s	23.6
combined	461080	1327	[illegible]	334	238 (–29%)	125	2	18.83	24.5

essentially zero for **clients** of reflection

higher for frameworks exporting **reflective interfaces**

in-between for **applications** and large frameworks (which do both)

application code

```
class MyButton {
  var cb : Callback = ...

  def setState(s : Str)
    var m = "draw" + s
    cb.update(self, m)
  end

  def draw()
      cb.call()
  end

  def drawUp() ... end
  def drawDown() ... end
}
```

library code

```
class Callback {
  var sel : Str = ...
  var obj : Obj = ...

  def update(s : Str,
             o : Obj)
    this.sel = s
    this.obj = o
  end

  def call()
    this.obj.[this.sel]()
  end
}
```

Idiomatic reflection decouples **callbacks** and avoids **boilerplate**

application
code

```
class MyButton {
  var cb : Callback = ...

  def setState(s : Str)
    var m = "draw" + s
    cb.update(self, m)
  end

  def draw()
      cb.call()
  end

  def drawUp() ... end
  def drawDown() ... end
}
```

library
code

```
class Callback {
  var sel : Str = ...
  var obj : Obj = ...

  def update(s : Str,
             o : Obj)
    this.sel = s
    this.obj = o
  end

  def call()
    this.obj.[this.sel]()
  end
}
```

Idiomatic reflection decouples **callbacks** and avoids **boilerplate**

application code

```
class MyButton {
  var cb : Callback = ...

  def setState(s : Str)
    var m = "draw" + s
    cb.update(self, m)
  end

  def draw()
    cb.call()
  end

  def drawUp() ... end
  def drawDown() ... end
}
```

library code

```
class Callback {
  var sel : Str = ...
  var obj : Obj = ...

  def update(s : Str,
             o : Obj)
    this.sel = s
    this.obj = o
  end

  def call()
    this.obj.[this.sel]()
  end
}
```

Idiomatic reflection decouples **callbacks** and avoids **boilerplate**

application code

```
class MyButton {
  var cb : Callback = ...

  def setState(s : Str)
    var m = "draw" + s
    cb.update(self, m)
  end

  def draw()
      cb.call()
  end

  def drawUp() ... end
  def drawDown() ... end
}
```

library code

```
class Callback {
  var sel : Str = ...
  var obj : Obj = ...

  def update(s : Str,
             o : Obj)
    this.sel = s
    this.obj = o
  end

  def call()
    this.obj.[this.sel]()
  end
}
```

Idiomatic reflection decouples **callbacks** and avoids **boilerplate**

application code

```
class MyButton {
  var cb : Callback = ...

  def setState(s : Str)
    var m = "draw" + s
    cb.update(self, m)
  end

  def draw()
      cb.call()
  end

  def drawUp() ... end
  def drawDown() ... end
}
```

library code

```
class Callback {
  var sel : Str = ...
  var obj : Obj = ...

  def update(s : Str,
             o : Obj)
    this.sel = s
    this.obj = o
  end

  def call()
    this.obj.[this.sel]()
  end
}
```

Idiomatic reflection decouples **callbacks** and avoids **boilerplate**