

liquid information flow control

Nadia Polikarpova

Chalmers Security Seminar

join work with Deian Stefan, Shachar Itzhaky, Jean Yang, Travis Hance, Armando Solar-Lezama

Facebook Security Breach Exposes Accounts of 50 Million Users



One of the challenges for Facebook's chief executive Mark Zuckerberg is convincing users that the company handles their data responsibly. Josh Edelson/Agence France-Presse — Getty Images

By **Mike Isaac** and **Sheera Frenkel**

Sept. 28, 2018



SAN FRANCISCO — Facebook, already facing scrutiny over how it handles the private information of its users, said on Friday that an attack on its computer network had exposed the personal information of nearly 50 million users.



PRIVACY AND SECURITY

This Is Almost Certainly James Comey's Twitter Account



Ashley Feinberg

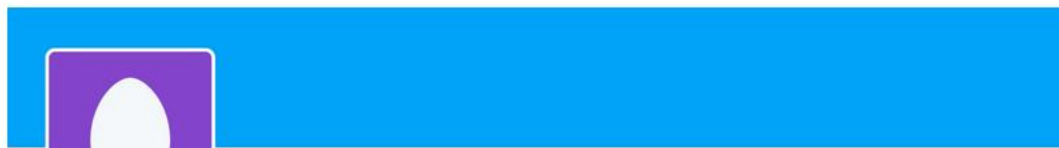
3/30/17 3:29PM • Filed to: JAMES COMEY ▾



640



239

**Reinhold Niebuhr**

@projectoxide7

Joined February 2014

Tweet to Reinhold Niebuhr

FOLLOWING
27FOLLOWERS
1LIKES
39

Follow

Likes



Reuters Top News @Reuters · Mar 22

'Rotten apples': Even South Africa's government says there are problems with some immigration system officials:



3 57 46

Who to follow · Refresh · View all



Daniel D'Addario @DDPD_

Follow



Blake Hounshell @blake...

Follow



David Frum @davidfrum

Follow

Find friends

United States Trends · Change

#NationalDoctorsDay

7,591 Tweets

#ThursdayThoughts

@JoyceCarolOates is Tweeting about this

#NeverAskAWoman

12.3K Tweets

Mike Pence

39K Tweets

#RepealH82

LOST rights across South Florida



Recent Video

**Pink Snow in the Italian Alps Is a Cute Sign of Environmental Catastrophe**

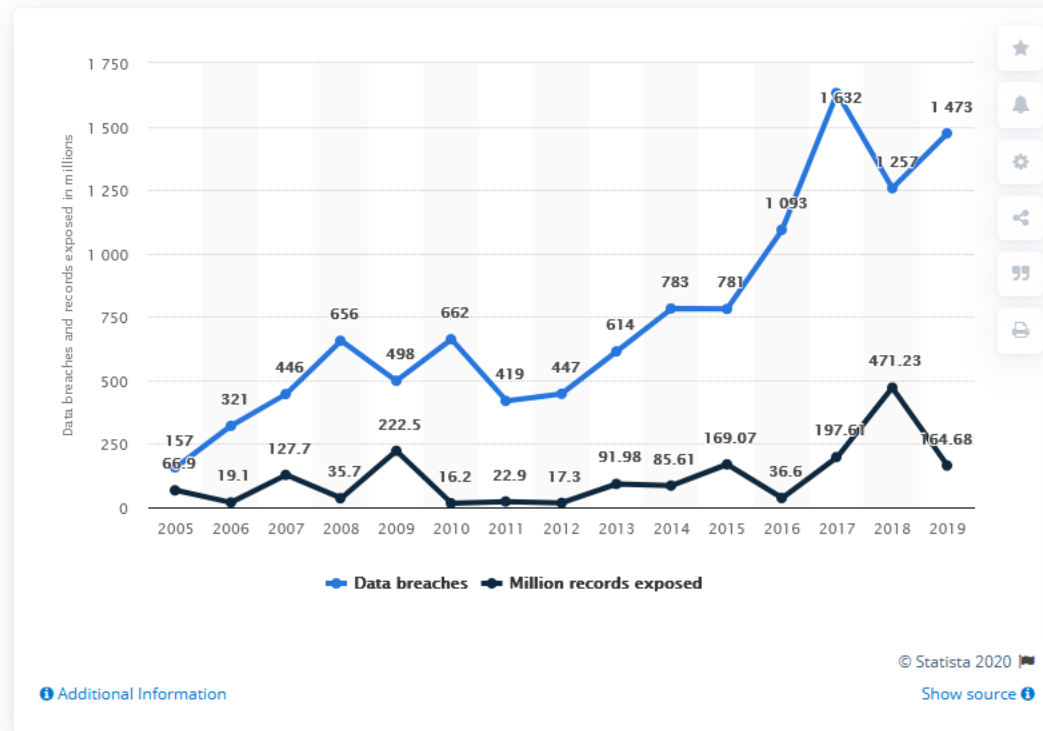
Yessenia Funes | 23

Digital security and its discontents—from Hillary Clinton's emails to ransomware to Tor hacks—is in many ways one of the chief concerns of the contemporary FBI. So it makes sense that the bureau's director, James Comey, would dip his toe into the digital torrent with a Twitter account. It also makes sense, given Comey's high profile, that he would want that Twitter account to

[Internet](#) > [Cyber Crime](#)

Annual number of data breaches and exposed records in the United States from 2005 to 2019

(in millions)



DOWNLOAD



Source

- [Show sources information](#)
- [Show publisher information](#)

Release date

January 2020

Region

United States

Survey time period

2005 to 2019

Special properties

sensitive records exposed; excluding non-sensitive records exposed

Join
the expert
webinar

Technology Market Outlook

How to
actively shape
tech trends
worldwide.



Sign up

statista

Cyber crime: number of breaches and records exposed 2005-2019

Published by [J. Clement](#), Mar 10, 2020



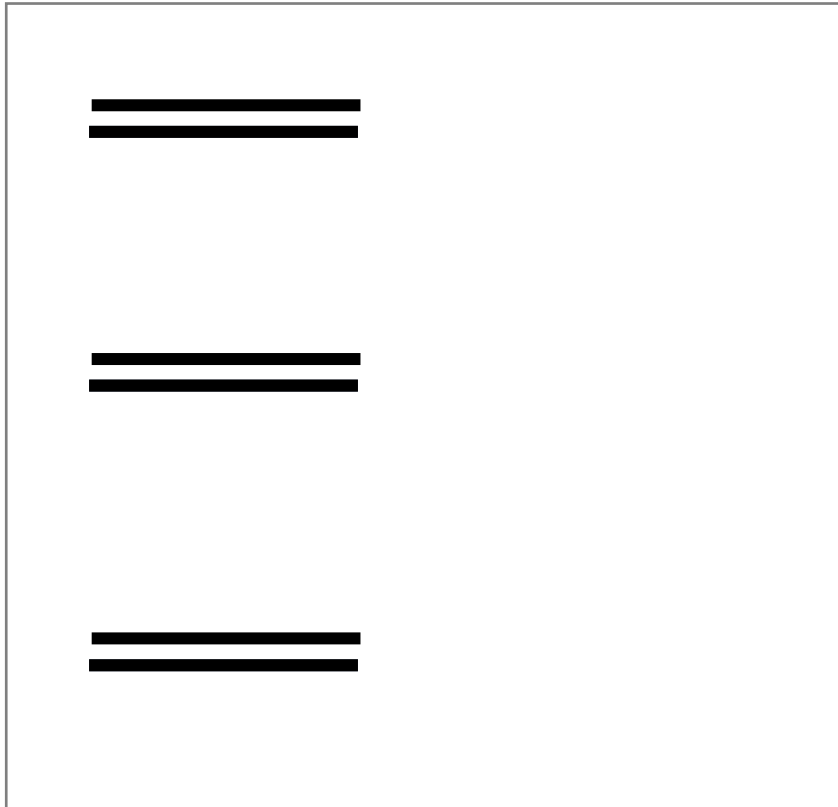
In 2019, the number of data breaches in the United States amounted to 1,473 with over 164.68 million sensitive records exposed.

Data breaches and exposed records - additional information

Data breaches have gained attention with the increasing use of digital files and companies and users large reliance on digital data. Even though data breaches happened before digitalization of information – for instance, looking at one's hard copy of medical files without authorization could be

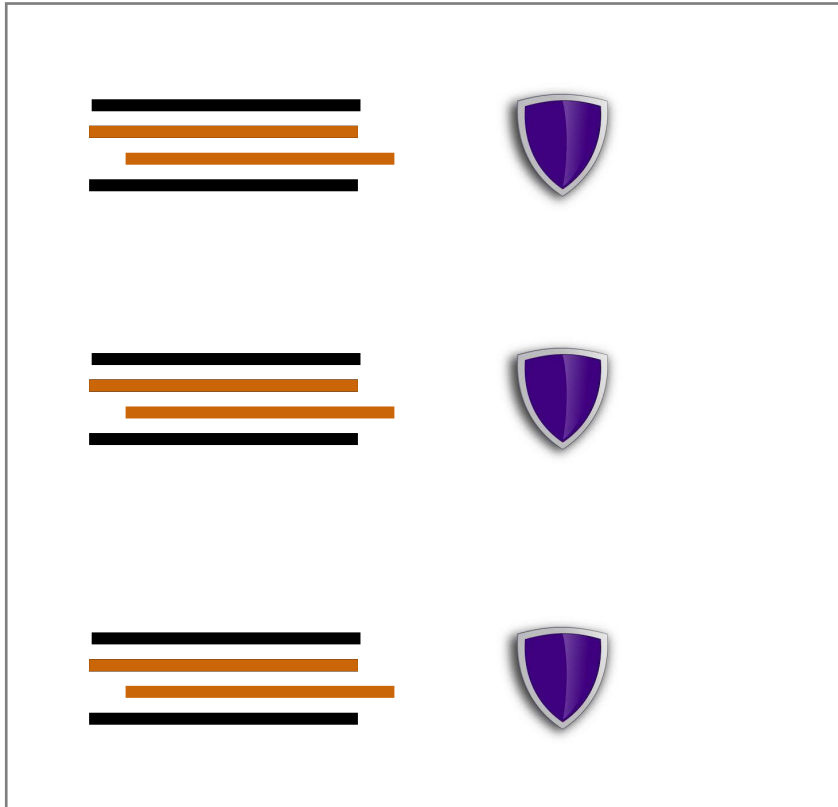
traditional policy enforcement

application code



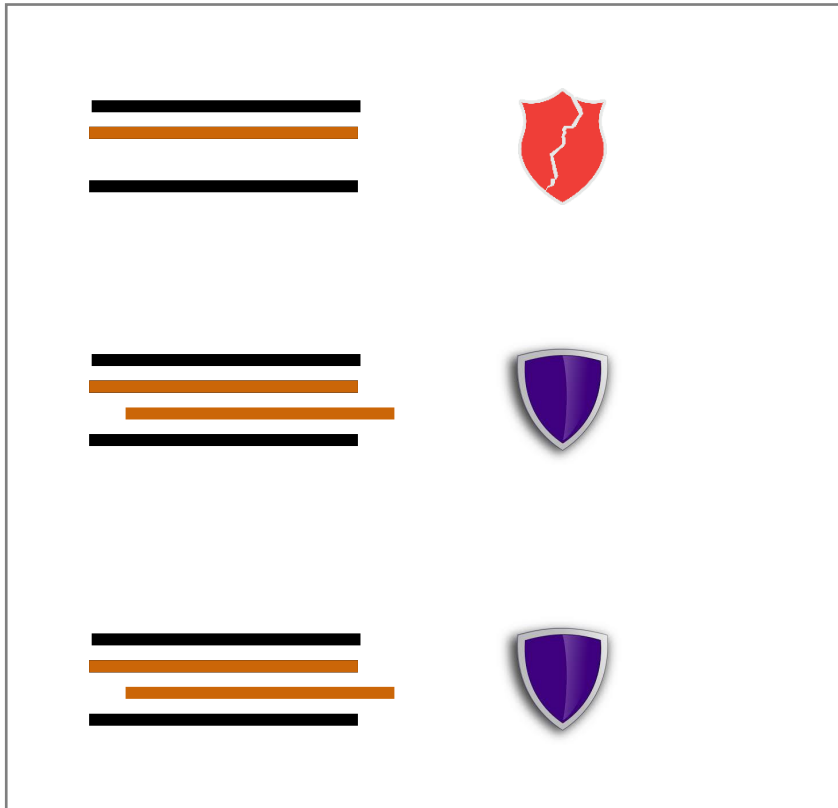
traditional policy enforcement

application code



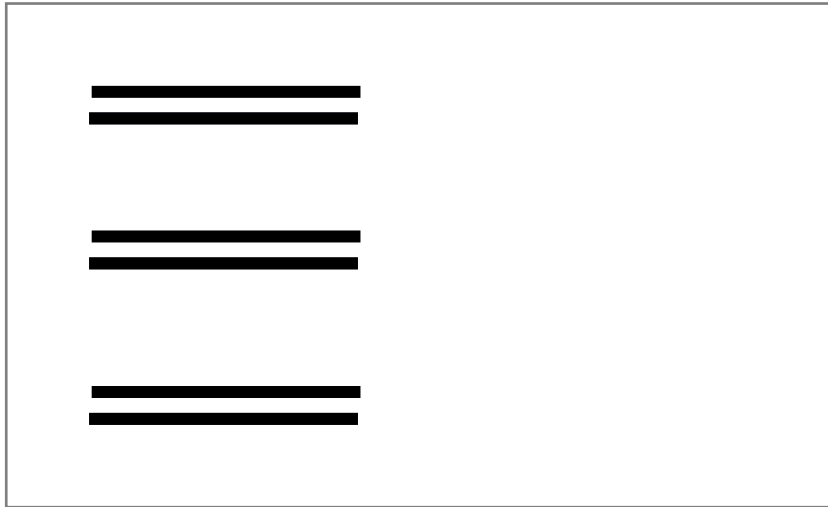
traditional policy enforcement

application code

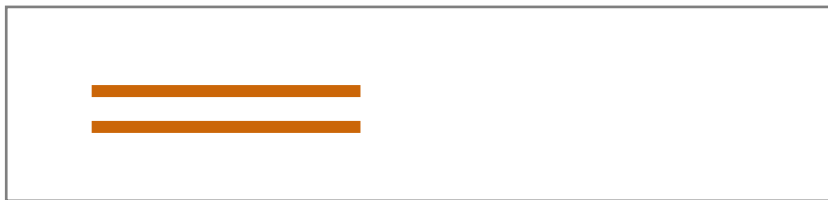


IFC* frameworks

application code

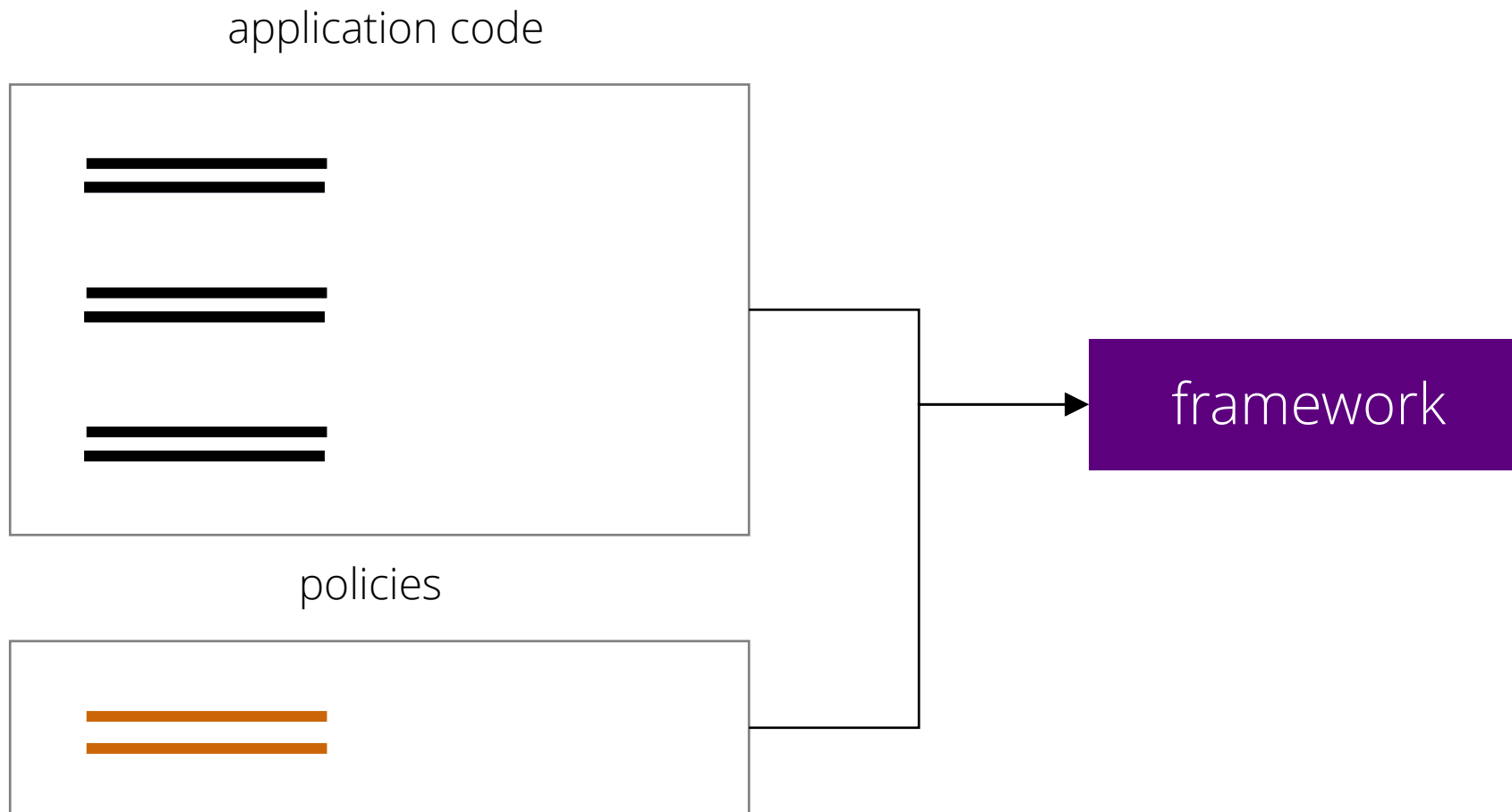


policies



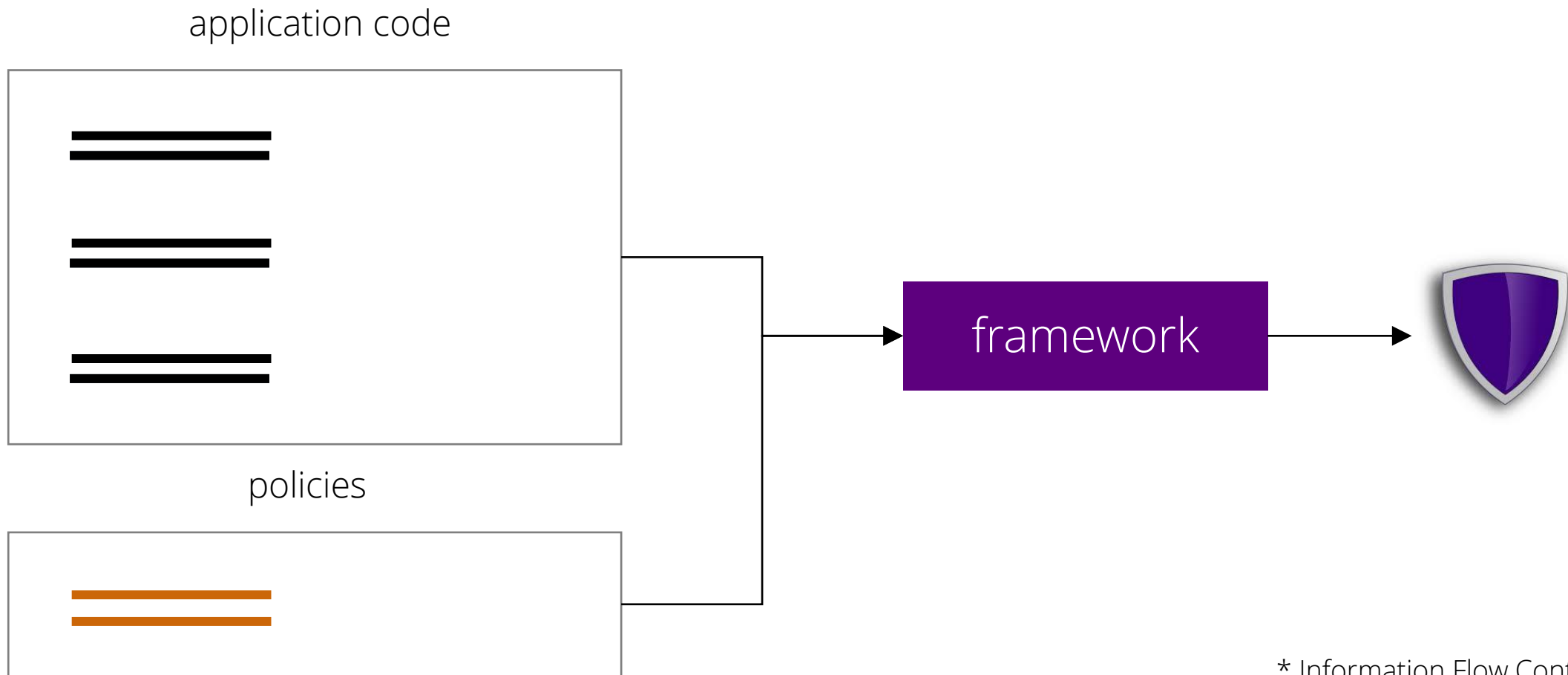
* Information Flow Control

IFC* frameworks



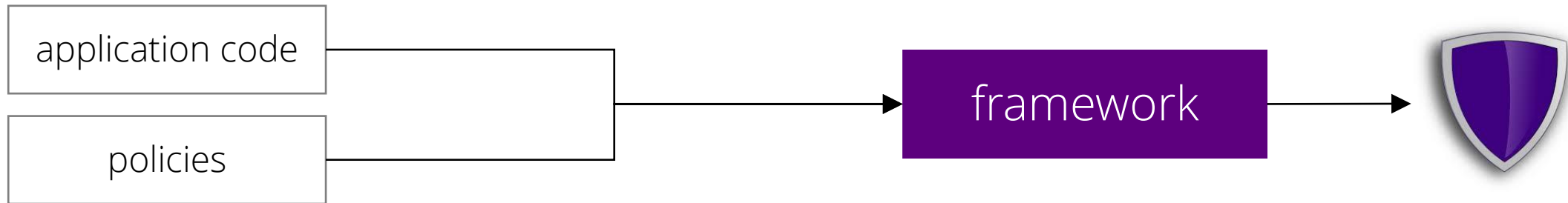
* Information Flow Control

IFC* frameworks

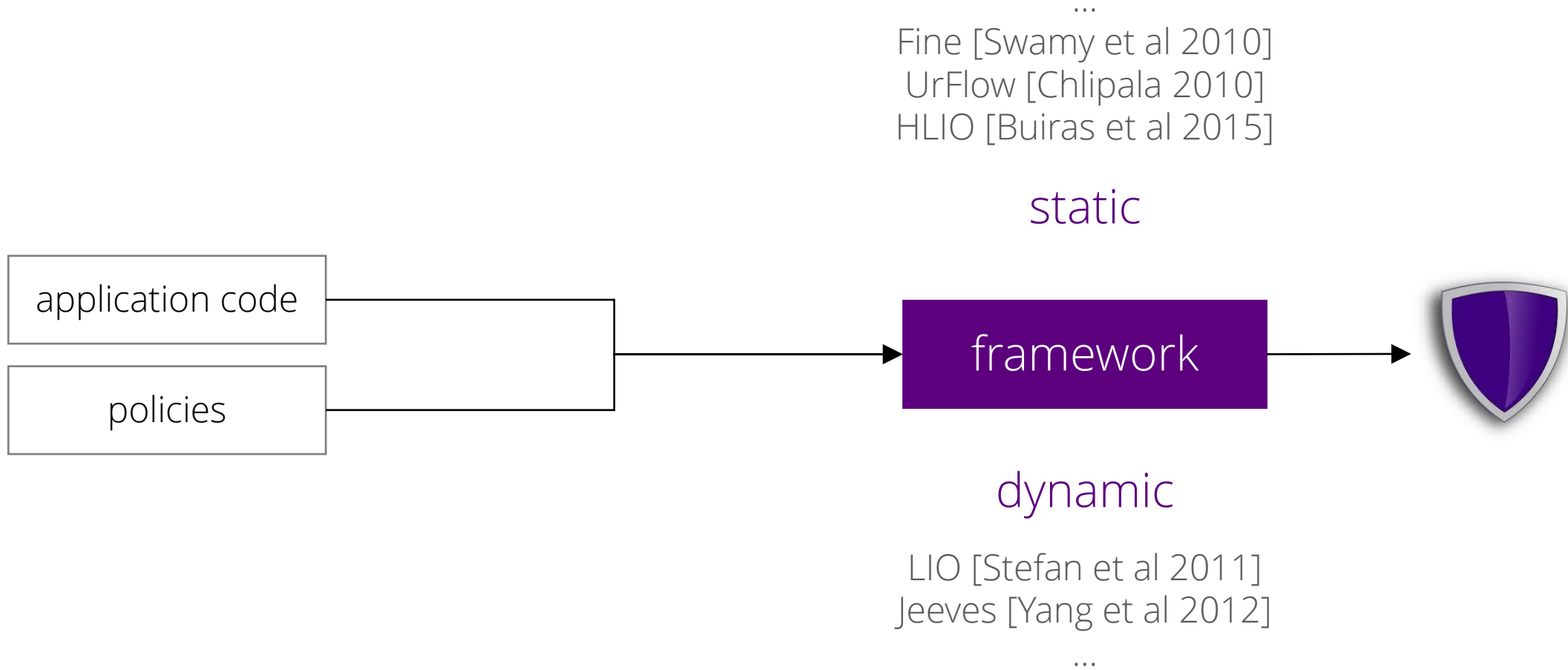


* Information Flow Control

existing IFC frameworks

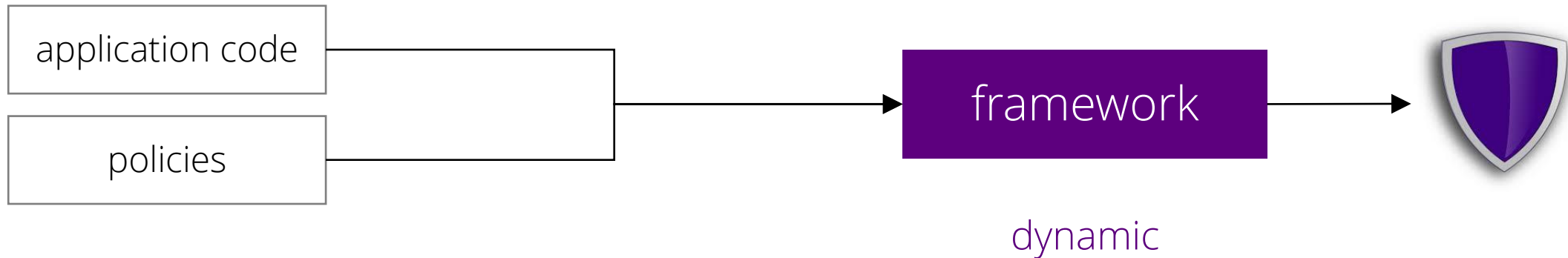


existing IFC frameworks



existing IFC frameworks

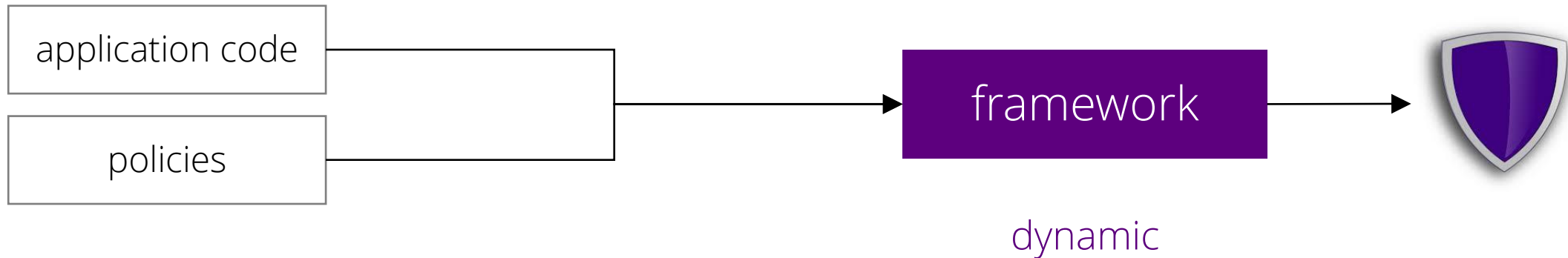
- + predictable behavior
- + no run-time overhead



- crashes / unpredictable behavior
 - run-time overhead

existing IFC frameworks

- + predictable behavior
- + no run-time overhead
- rich policies require proof hints



- crashes / unpredictable behavior
 - run-time overhead
- + support for rich policies

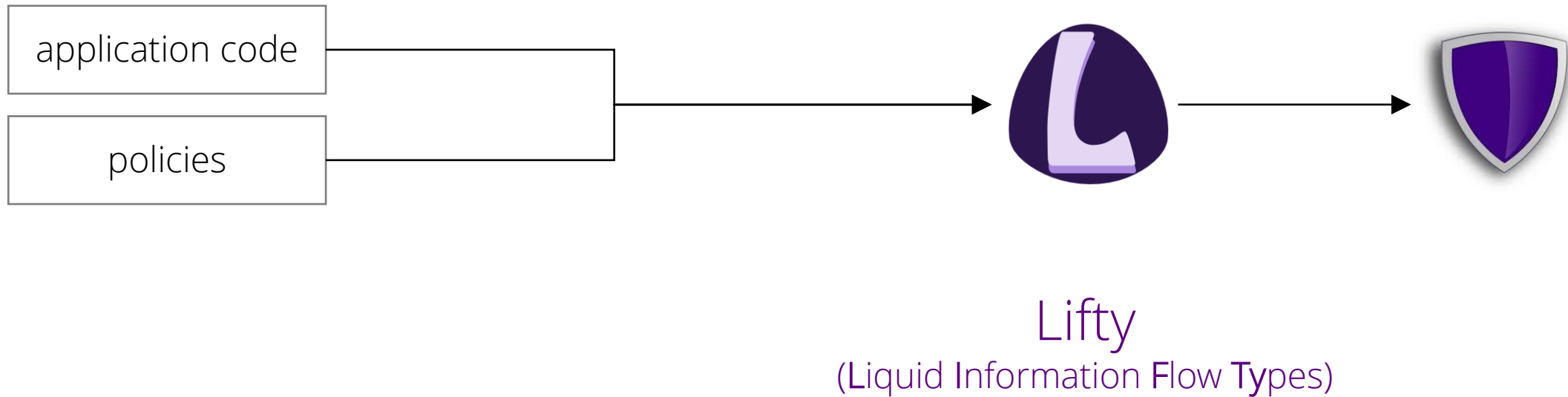
our solution: lifty

+ predictable behavior
+ no run-time overhead



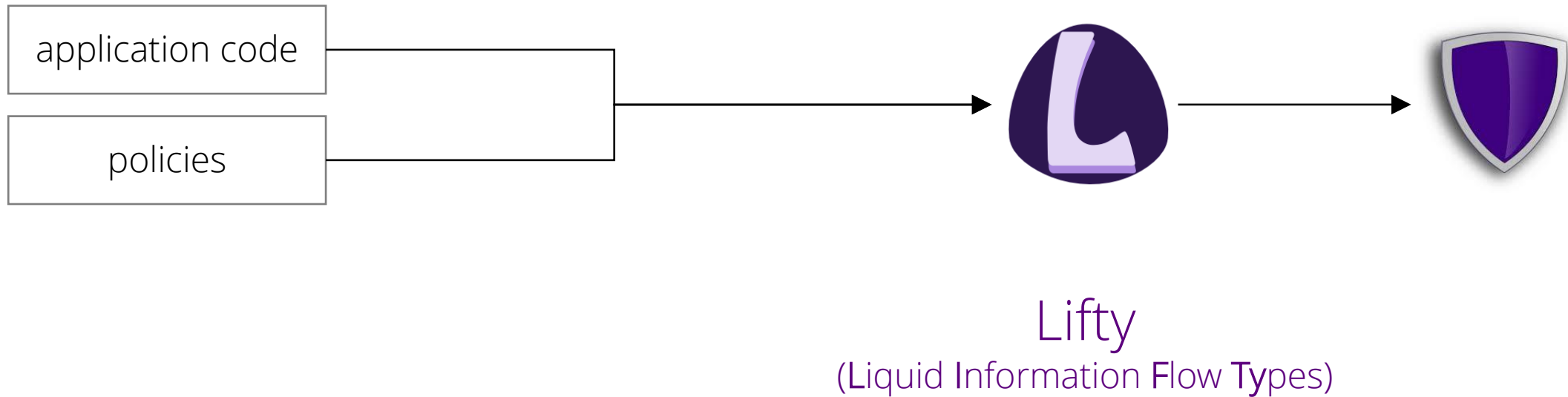
our solution: **lifty**

+ predictable behavior
+ no run-time overhead



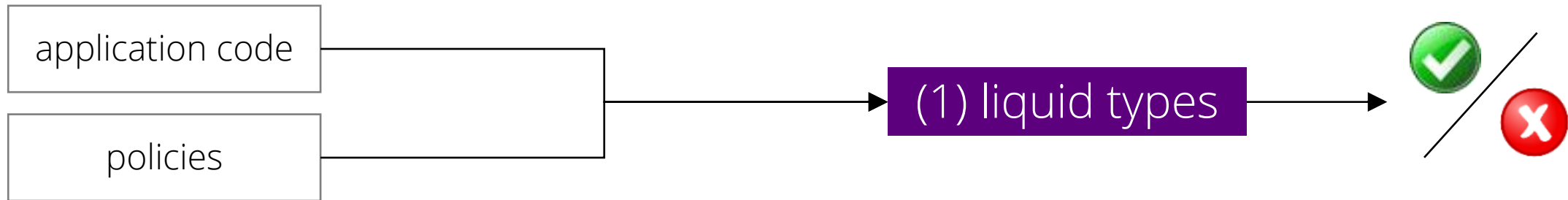
our solution: **lifty**

- + predictable behavior
- + no run-time overhead
- + automatic checking of rich policies



our solution: lofty

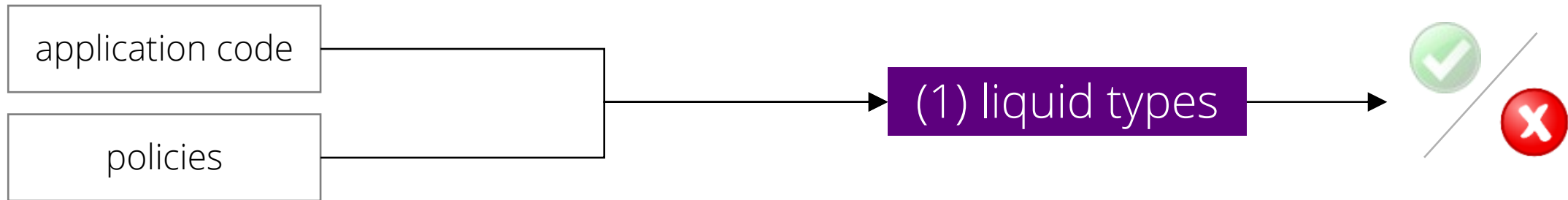
- + predictable behavior
- + no run-time overhead
- + automatic checking of rich policies



Lifty
(Liquid Information Flow Types)

our solution: lofty

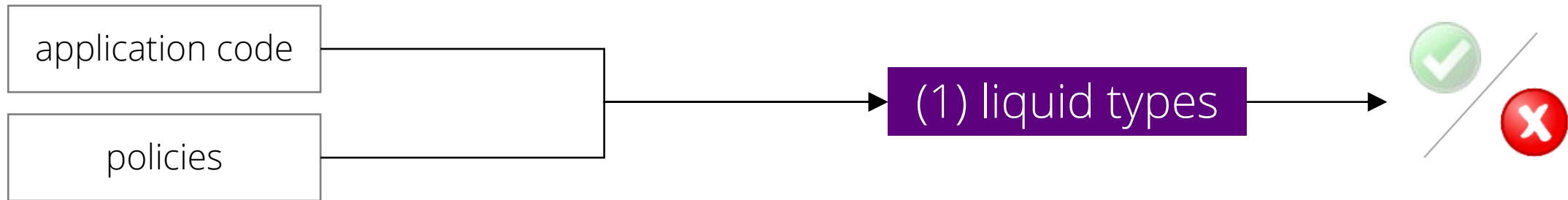
- + predictable behavior
- + no run-time overhead
- + automatic checking of rich policies



Lifty
(Liquid Information Flow Types)

our solution: lifty

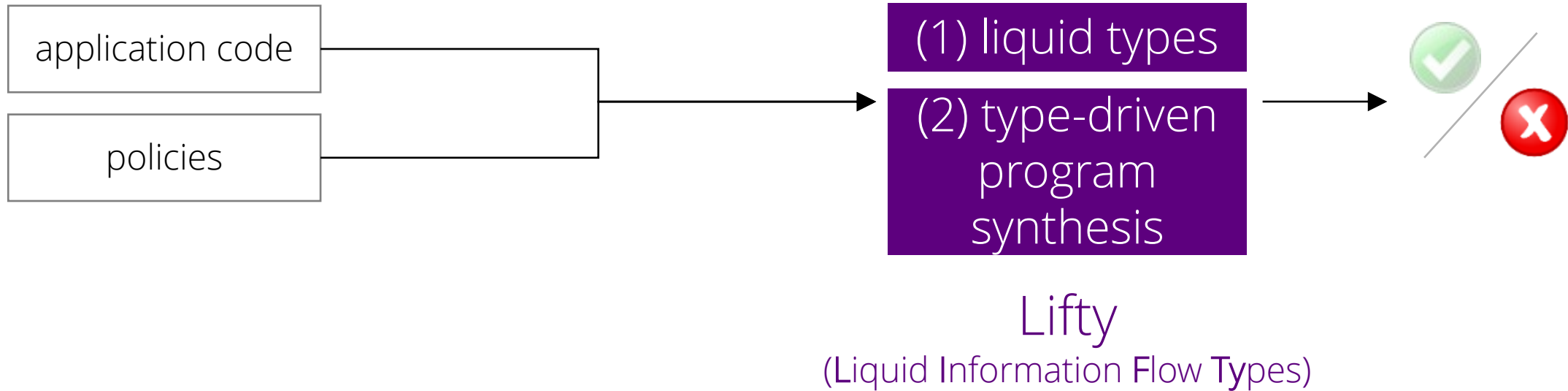
- + predictable behavior
- + no run-time overhead
- + automatic checking of rich policies
- + leak repair



Lifty
(Liquid Information Flow Types)

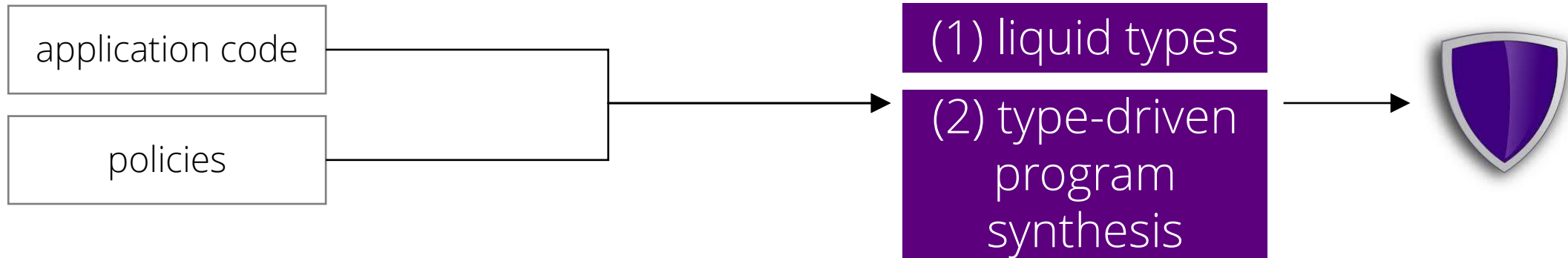
our solution: **lifty**

- + predictable behavior
- + no run-time overhead
- + automatic checking of rich policies
- + leak repair



our solution: **lifty**

- + predictable behavior
- + no run-time overhead
- + automatic checking of rich policies
- + leak repair



Lifty
(Liquid Information Flow Types)

1. IFC with liquid types

1. IFC with liquid types
2. programming in Lifty

1. IFC with liquid types
2. programming in Lifty
3. leak repair

1. IFC with liquid types
2. programming in Lifty
3. leak repair

liquid types*

Int

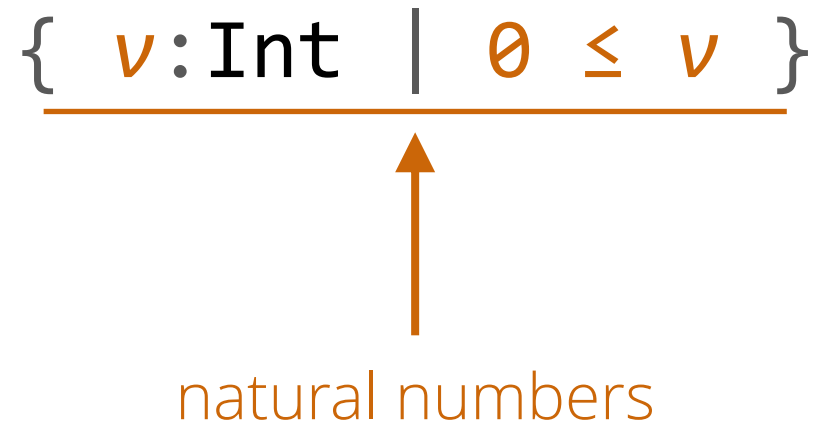
* [Rondon, Kawaguchi, Jhala. 2008]

liquid types*

$$\{ \textcolor{brown}{v} : \text{Int} \mid \frac{\textcolor{brown}{0} \leq \textcolor{brown}{v}}{\text{refinement}} \}$$

* [Rondon, Kawaguchi, Jhala. 2008]

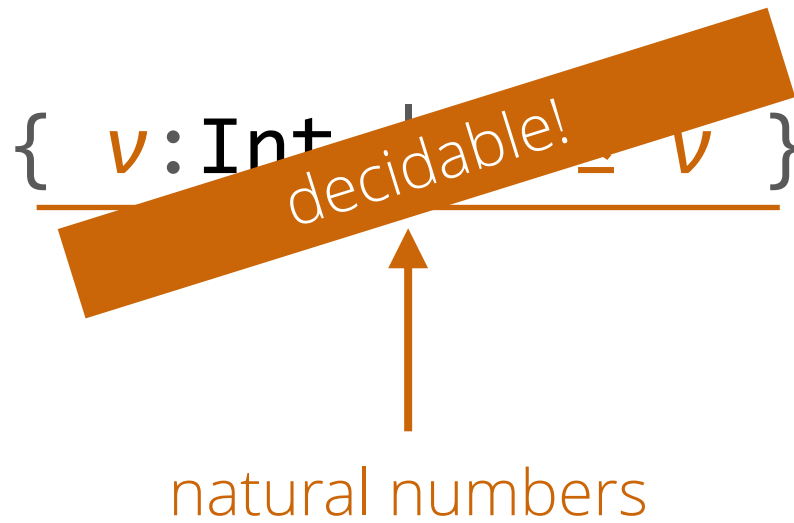
liquid types*

$$\{ \textcolor{brown}{v} : \text{Int} \mid \textcolor{brown}{0} \leq \textcolor{brown}{v} \}$$


natural numbers

* [Rondon, Kawaguchi, Jhala. 2008]

liquid types*



* [Rondon, Kawaguchi, Jhala. 2008]

security monads*

TIO Int

* [Russo, Claessen, Hughes, 2008], [Stefan et al 2011], ...

security monads*

TIO Int



sensitive computation that returns int

* [Russo, Claessen, Hughes, 2008], [Stefan et al 2011], ...

TIO = security monads + liquid types

TIO Int $\langle u = \text{alice} \rangle \langle u = \text{bob} \rangle$

TIO = security monads + liquid types

TIO Int $\frac{\langle u = \text{alice} \rangle}{\text{input label}}$ $\frac{\langle u = \text{bob} \rangle}{\text{output label}}$

TIO = security monads + liquid types

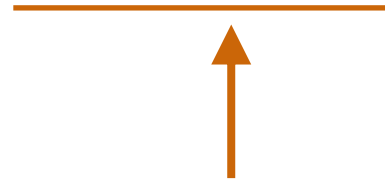
TIO Int <u = alice> <u = bob>



may read data restricted to alice

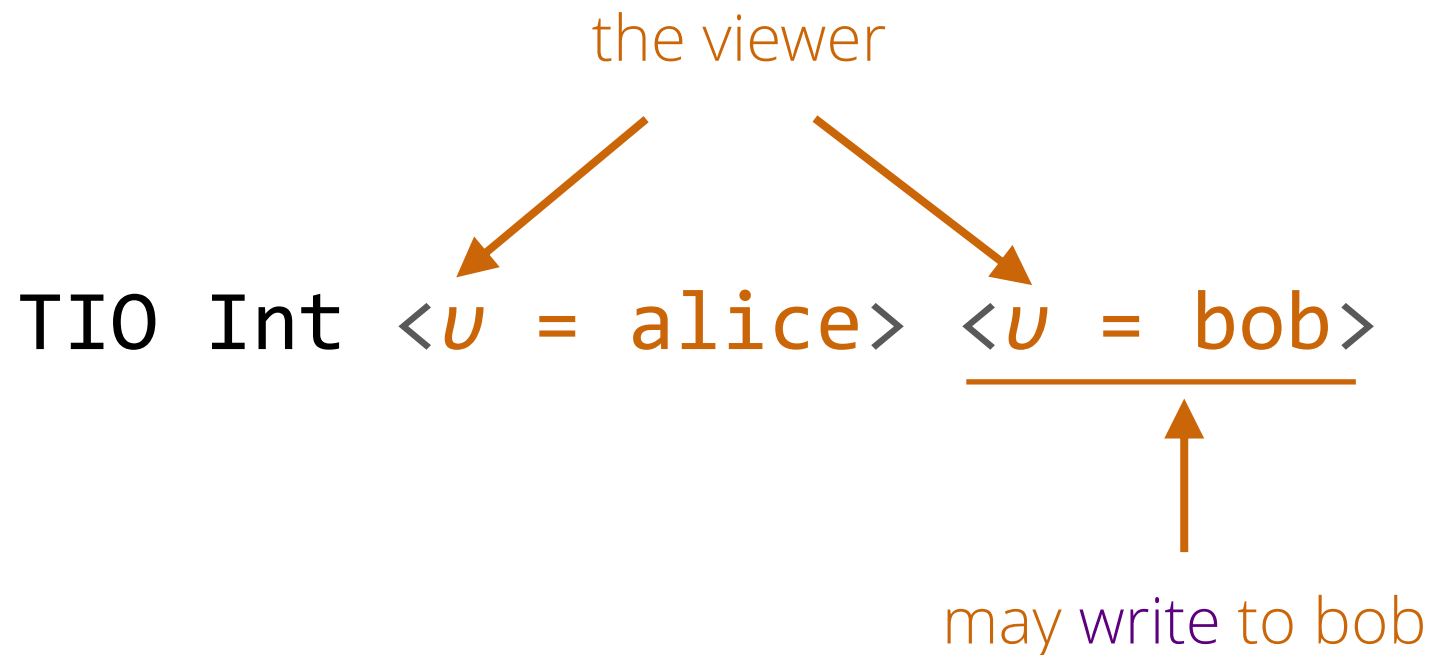
TIO = security monads + liquid types

TIO Int $\langle u = \text{alice} \rangle$ $\frac{\langle u = \text{bob} \rangle}{\text{may write to bob}}$



The diagram illustrates a state transition in a TIO (Typed Intermediate Object) monad. It shows two states: $\langle u = \text{alice} \rangle$ and $\langle u = \text{bob} \rangle$. An orange arrow points from the text "may write to bob" to the transition line between the two states, indicating the action that causes the state change.

TIO = security monads + liquid types



1. IFC with liquid types

- the TIO monad
- building TIO computations
- type checking

2. programming in Lifty

3. leak repair

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

```
getSharedKey :: TIO String <u ∈ [alice, bob]> <false>
```

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

getSharedKey :: TIO String <u ∈ [alice, bob]> <false>

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

`getSharedKey :: TIO String <u ∈ [alice, bob]> <false>`

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

```
getSharedKey :: TIO String <u ∈ [alice, bob]> <false>
```

```
print :: rec:User → String → TIO Unit <true> <u = rec>
```

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

`getSharedKey :: TIO String <u ∈ [alice, bob]> <false>`

`print :: rec:User → String → TIO Unit <true> <u = rec>`

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

```
getSharedKey :: TIO String <u ∈ [alice, bob]> <false>
```

```
print :: rec:User → String → TIO Unit <true> <u = rec>
```

atomic TIO actions

model sources and sinks of sensitive data, e.g.:

```
getSharedKey :: TIO String <u ∈ [alice, bob]> <false>
```

```
print :: rec:User → String → TIO Unit <true> <u = rec>
```



the TIO API

the TIO API

-- Return a pure value

`return :: a → TIO a`

the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

-- Sequence two sensitive computations

`bind :: TIO a →
 (a → TIO b) →
 TIO b`

the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

-- Sequence two sensitive computations

`bind :: TIO a →
 (a → TIO b) →
 TIO b`



the TIO API

-- Return a pure value

return :: a → TIO a <true> <false>

-- Sequence two sensitive computations

bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b



the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

-- Sequence two sensitive computations

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b`



the TIO API

-- Return a pure value

```
return :: a → TIO a <true> <false>
```

-- Sequence two sensitive computations

```
bind :: TIO a <i> <p> →  
      (a → TIO b <j> <i ∧ p>) →  
      TIO b
```



the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

-- Sequence two sensitive computations

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b`



the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

-- Sequence two sensitive computations

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b <i ∧ j>`



the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

-- Sequence two sensitive computations

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b <i ∧ j> <o ∨ (i ∧ p)>`



the TIO API

-- Return a pure value

`return :: a → TIO a <true> <false>`

-- Sequence two sensitive computations

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b <i ∧ j> <o ∨ (i ∧ p)>`

-- Safely downgrade a Boolean

`downgrade :: ... -- later in this talk`

the TIO API

`return :: a → TIO a <true> <false>`

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b <i ∧ j> <o ∨ (i ∧ p)>`

`downgrade :: ...`

`liftM :: ...`

`filterM :: ...`

`mapM :: ...`

`sortM :: ...`

the TIO API

`return :: a → TIO a <true> <false>`

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b <i ∧ j> <o ∨ (i ∧ p)>`

`downgrade :: ...`

`liftM :: ...`

`filterM :: ...`

`mapM :: ...`

`sortM :: ...`

primitive operations
(trusted)

the TIO API

`return :: a → TIO a <true> <false>`

`bind :: TIO a <i> <o> →
 (a → TIO b <j> <i ∧ p>) →
 TIO b <i ∧ j> <o ∨ (i ∧ p)>`

`downgrade :: ...`

`liftM :: ...`

`filterM :: ...`

`mapM :: ...`

`sortM :: ...`

primitive operations
(trusted)

derived operations
(verified)

demo

<http://comcom.csail.mit.edu/demos/#lifty-chalmers-toy>

1. IFC with liquid types

- the TIO monad
- building TIO computations
- type checking

2. programming in Lifty

3. leak repair

example (safe)

```
do
  k ← getSharedKey
  print alice k
```

example (safe)

how do we verify that this code is safe?

```
do
  k ← getSharedKey
  print alice k
```

example (safe)

how do we verify that this code is safe?

```
do  
  k ← getSharedKey  
  print alice k
```

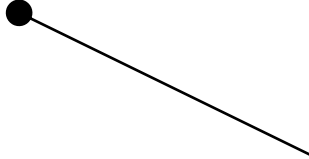
key insight 1:
use liquid type inference

example (safe)

```
bind getSharedKey (\k . print alice k)
```

example (safe)

TIO String $\langle u \in [\text{alice}, \text{bob}] \rangle \langle \text{false} \rangle$



bind getSharedKey ($\backslash k$. print alice k)

example (safe)

TIO String $\langle v \in [\text{alice}, \text{bob}] \rangle \langle \text{false} \rangle$ $\text{rec}:\text{User} \rightarrow \text{String} \rightarrow \text{TIO Unit} \langle \text{true} \rangle \langle v = \text{rec} \rangle$



bind getSharedKey (\k . print alice k)

example (safe)

TIO String $\langle v \in [\text{alice}, \text{bob}] \rangle \langle \text{false} \rangle$

$\text{rec}:\text{User} \rightarrow \text{String} \rightarrow \text{TIO Unit} \langle \text{true} \rangle \langle v = \text{rec} \rangle$

bind getSharedKey ($\backslash k . \text{print alice } k$)

$\forall a \text{ } i \text{ } o \text{ } j \text{ } p .$

TIO a $\langle i \rangle \langle o \rangle \rightarrow$

$(a \rightarrow \text{TIO } b \langle j \rangle \langle i \wedge p \rangle) \rightarrow$

TIO b $\langle i \wedge j \rangle \langle o \vee (i \wedge p) \rangle$

step 1: instantiate bind

TIO String $\langle u \in [\text{alice}, \text{bob}] \rangle \langle \text{false} \rangle$

```
rec:User → String → TIO Unit <true> <v = rec>
```

```
bind getSharedKey (\k . print alice k)
```

$$\begin{aligned} \text{TIO String } \langle I \rangle \langle O \rangle &\rightarrow \\ (\text{String} \rightarrow \text{TIO Unit } \langle J \rangle \langle I \wedge P \rangle) &\rightarrow \\ \text{TIO Unit } \langle I \wedge J \rangle \langle O \vee (I \wedge P) \rangle & \end{aligned}$$

step 1: instantiate bind

TIO String $\langle u \in [\text{alice}, \text{bob}] \rangle \langle \text{false} \rangle$

```
rec:User → String → TIO Unit <true> <v = rec>
```

```
bind getSharedKey (\k . print alice k)
```

$$\begin{aligned} \text{TIO String } \langle I \rangle \langle O \rangle &\rightarrow \\ (\text{String} \rightarrow \text{TIO Unit } \langle J \rangle \langle I \wedge P \rangle) &\rightarrow \\ \text{TIO Unit } \langle I \wedge J \rangle \langle O \vee (I \wedge P) \rangle & \end{aligned}$$

Horn clauses:

step 2: generate Horn clauses

TIO String $\langle u \in [\text{alice}, \text{bob}] \rangle \langle \text{false} \rangle$

```
rec:User → String → TIO Unit <true> <v = rec>
```

```
bind getSharedKey (\k . print alice k)
```

TIO String $\langle I \rangle \langle O \rangle \rightarrow$
 (String $\rightarrow$ TIO Unit $\langle J \rangle \langle I \wedge P \rangle$) $\rightarrow$
 TIO Unit $\langle I \wedge J \rangle \langle O \vee (I \wedge P) \rangle$

Horn clauses:

$I \Rightarrow v \in [\text{alice}, \text{bob}]$
false $\Rightarrow 0$

step 2: generate Horn clauses



Horn clauses:

$I \Rightarrow \nu \in [\text{alice}, \text{bob}]$

$\text{false} \Rightarrow 0$

$J \Rightarrow \text{true}$

$\nu = \text{alice} \Rightarrow I \wedge P$

step 2: generate Horn clauses



step 3: solve Horn clauses

Horn clauses:

$$I \Rightarrow v \in [\text{alice}, \text{bob}]$$

$$v = \text{alice} \Rightarrow I \wedge P$$

step 3: solve Horn clauses

Horn clauses:

query $I \Rightarrow u \in [\text{alice}, \text{bob}]$

rule $u = \text{alice} \Rightarrow I \wedge P$

step 3: solve Horn clauses

Horn clauses:

query $I \Rightarrow v \in [\text{alice}, \text{bob}]$

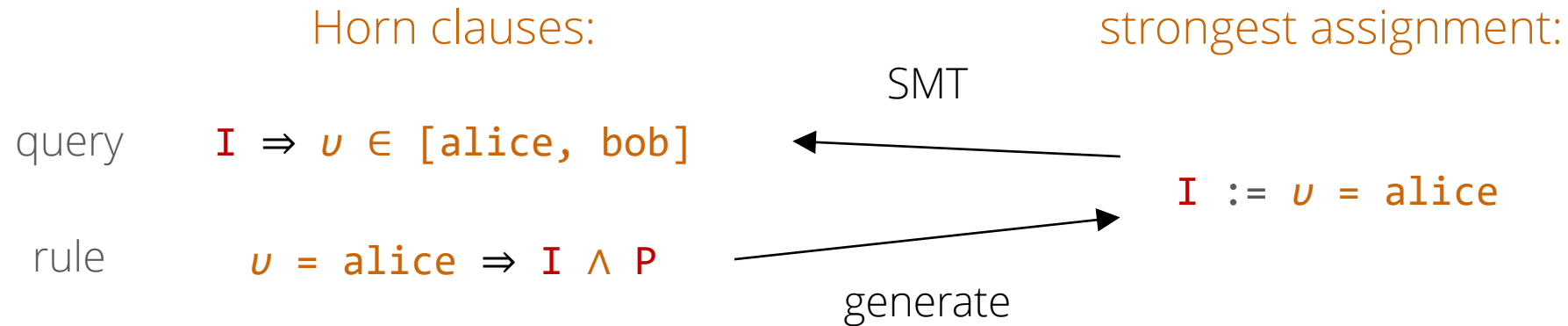
rule $v = \text{alice} \Rightarrow I \wedge P$

strongest assignment:

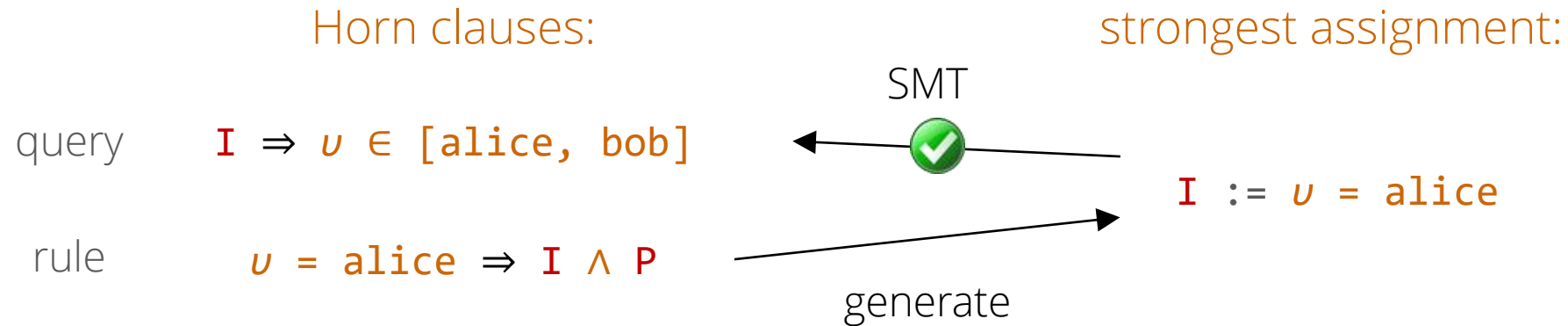
$I := v = \text{alice}$

generate

step 3: solve Horn clauses

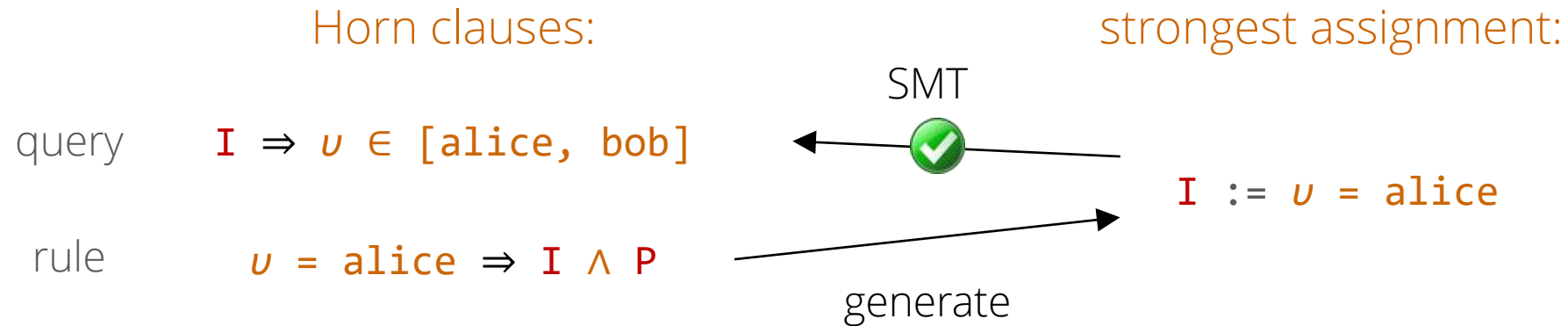


step 3: solve Horn clauses



step 3: solve Horn clauses

```
bind getSharedKey (\k . print alice k)
```



example (unsafe)

```
bind getSharedKey (\k . print alice k)
```

example (unsafe)

```
\client . bind getSharedKey (\k . print client k)
```

example (unsafe)

```
\client . bind getSharedKey (\k . print client k)
```

Horn clauses:

query $I \Rightarrow u \in [\text{alice}, \text{bob}]$

rule $\boxed{u = \text{client}} \Rightarrow I \wedge P$

example (unsafe)

```
\client . bind getSharedKey (\k . print client k)
```

Horn clauses:

query $I \Rightarrow v \in [\text{alice}, \text{bob}]$

rule $\boxed{v = \text{client}} \Rightarrow I \wedge P$

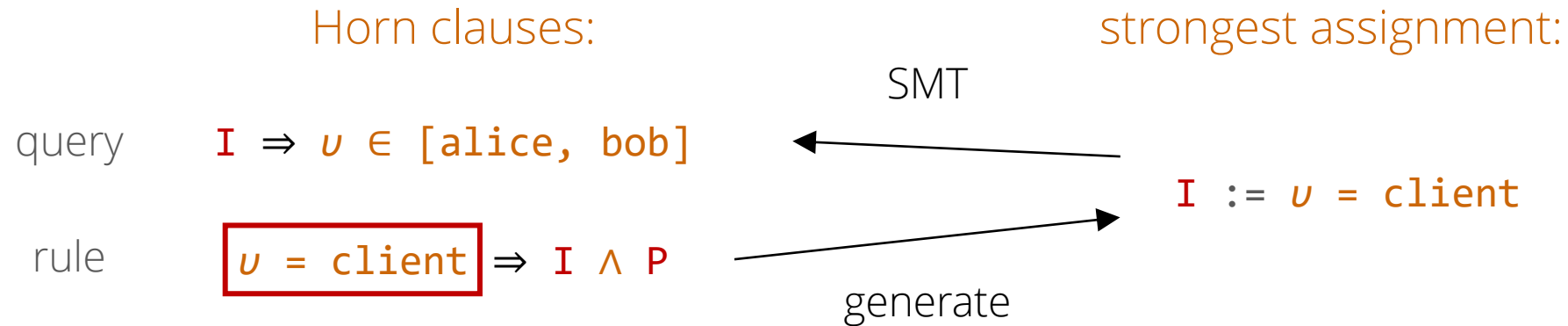
strongest assignment:

$I := v = \text{client}$

generate

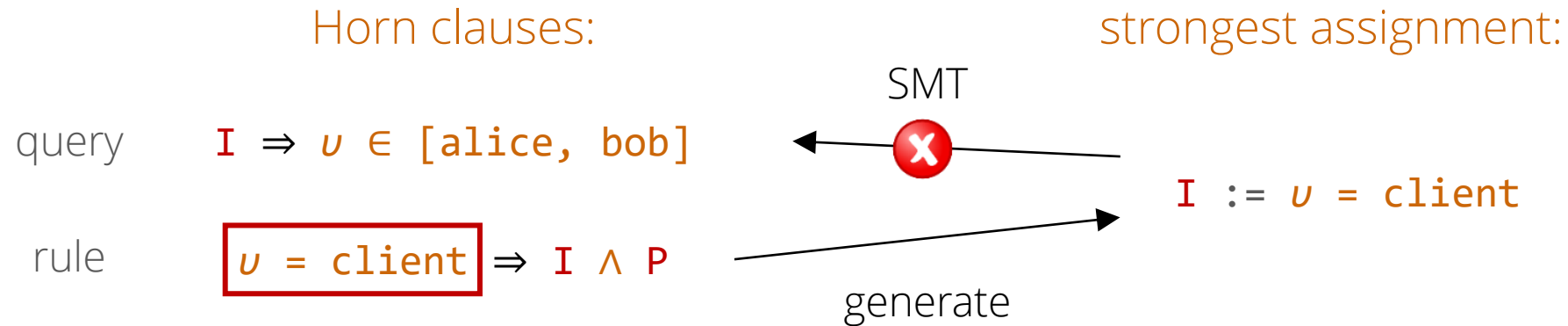
example (unsafe)

```
\client . bind getSharedKey (\k . print client k)
```



example (unsafe)

```
\client . bind getSharedKey (\k . print client k)
```



example (unsafe)

```
\client . bind getSharedKey (\k . print client k)
```



Horn clauses:

query $I \Rightarrow v \in [\text{alice}, \text{bob}]$

rule $\boxed{v = \text{client}} \Rightarrow I \wedge P$

SMT



strongest assignment:

$I := v = \text{client}$

generate

1. IFC with liquid types
2. programming in Lifty
3. leak repair

running example: conference manager

running example: conference manager

#	title*	score	decision
1	Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive μ -types	1.8	accept
2	Naive Gradual Π -Calculus with Strong Normalization	0.8	reject
3	Abstract Impredicative Type Theory with Ownership Types	2.0	accept
4	Higher Type Theory with Primitive Recursion	1.5	accept
5	Liquid Information Flow Control	c	c
6	Standard Higher Calculus of Inductive Constructions with Full Type Inference	0.5	reject

*titles generated by <http://type.systems/>

running example: conference manager

#	title*	score	decision
1	Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive μ -types	1.8	accept
2	Naive Gradual Π -Calculus with Strong Normalization	0.8	reject
3	Abstract Impredicative Type Theory with Ω	0.9	accept
4	Higher Type	1.5	accept
5	Liquid Information Flow Control	c	c
6	Standard Higher Calculus of Inductive Constructions with Full Type Inference	0.5	reject

policy: if conflicted, can't see score until notification

*titles generated by <http://type.systems/>

running example: conference manager

#	title*	score	decision
1	Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive μ -types	1.8	accept
2	Naive Gradual Π -Calculus with Strong Normalization	0.8	reject
3	Abstract Impredicative Type Theory with Ownership Types	2.0	accept
4	Higher Type Theory with Primitive Recursion	1.5	accept
5	Liquid Information Flow Control	c	c
6	Standard Higher Calculus of Inductive Constructions with Full Type Inference	0.5	reject

*titles generated by <http://type.systems/>

running example: conference manager

#	title*	score ▼	decision
3	Abstract Impredicative Type Theory with Ownership Types	2.0	accept
5	Liquid Information Flow Control	c	c
1	Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursive μ -types	1.8	accept
4	Higher Type Theory with Primitive Recursion	1.5	accept
2	Naive Gradual Π -Calculus with Strong Normalization	0.8	reject
6	Standard Higher Calculus of Inductive Constructions with Full Type Inference	0.5	reject

*titles generated by <http://type.systems/>

running example: conference manager

#	title*	score ▼	decision
3	Abstract Impredicative Type Theory with Ownership Types	2.0	accept
5	Liquid Information Flow Control	c	c
1	Non-standard Paraconsistent Calculus of Coinductive Constructions with Isorecursion	1.8	accept
4	Higher Type	1.5	accept
2	Naive Gradual Π -Calculus with Strong Normalization	0.8	reject
6	Standard Higher Calculus of Inductive Constructions with Full Type Inference	0.5	reject

leaks score through order of papers!

*titles generated by <http://type.systems/>

demo

<http://comcom.csail.mit.edu/demos/#lifty-chalmers-demo>

solution:

<http://comcom.csail.mit.edu/demos/#lifty-chalmers-demo-solution>

1. IFC with liquid types

2. programming in Lifty

- conference manager
- self-referential policies

3. leak repair

demo

<http://comcom.csail.mit.edu/demos/#lifty-chalmers-demo-self-ref>

security guarantee

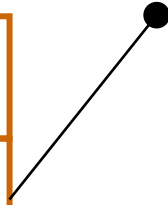
conflicts: Andrei, Alejandro

conflicts: Nadia , Andrei, Alejandro

security guarantee

Nadia can see

conflicts: Andrei, Alejandro



conflicts: Nadia , Andrei, Alejandro

security guarantee

Nadia can see



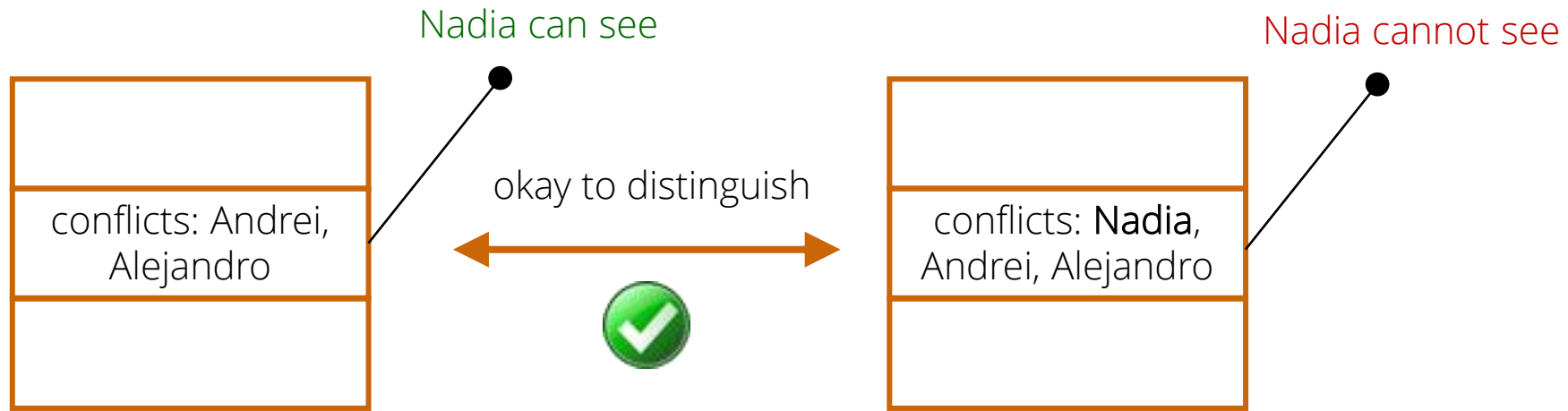
conflicts: Andrei, Alejandro

Nadia cannot see

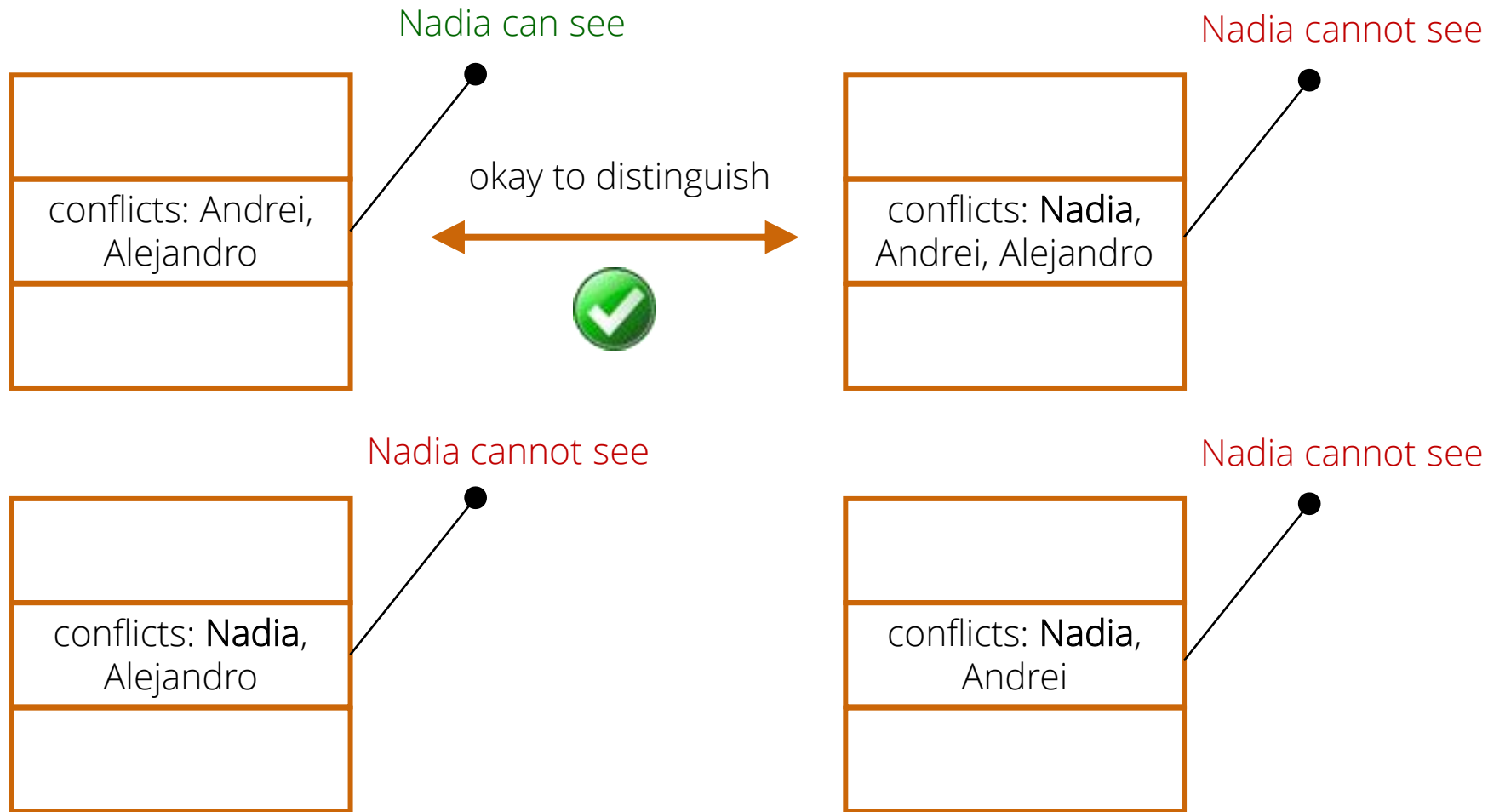


conflicts: Nadia , Andrei, Alejandro

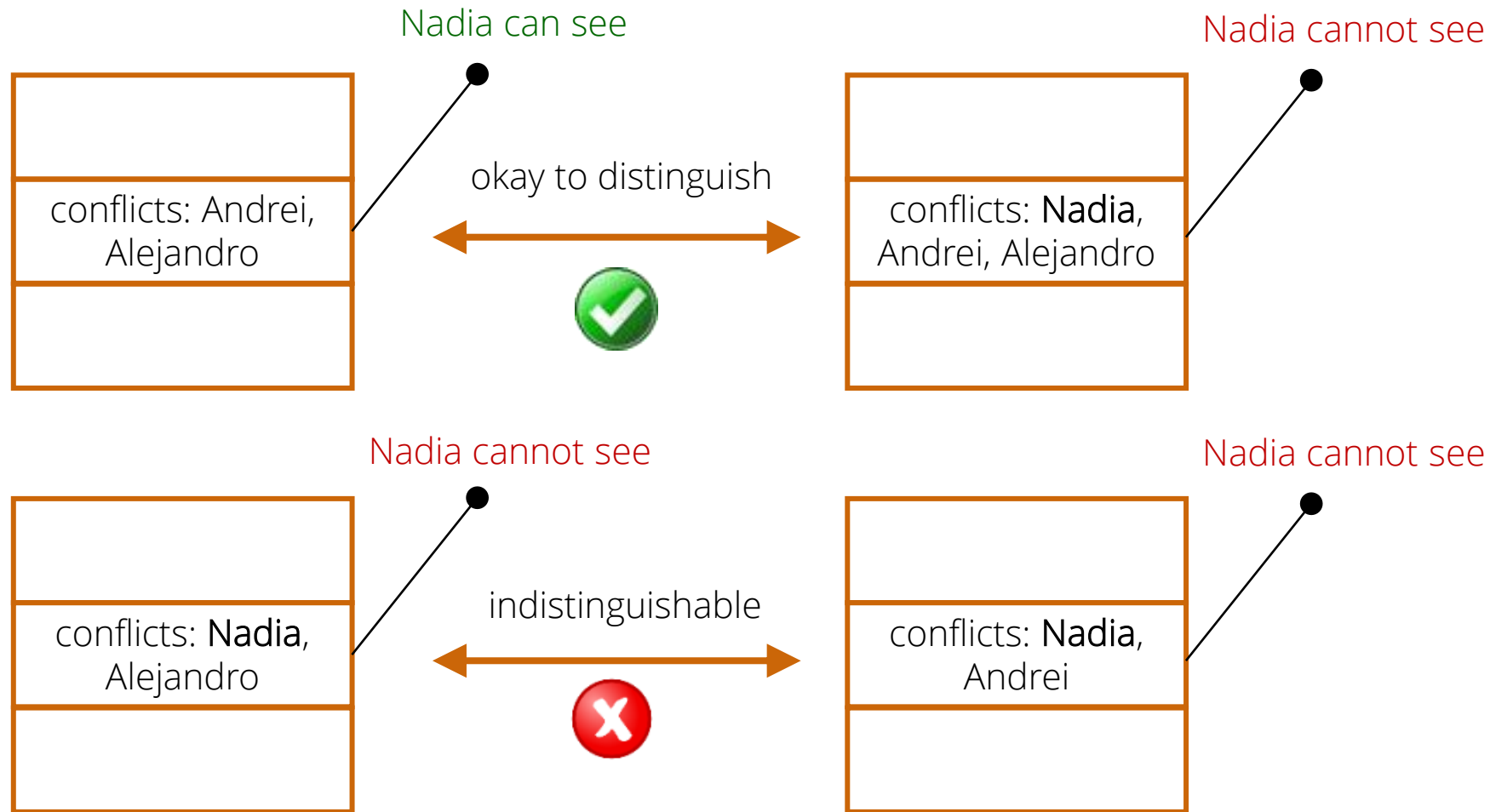
security guarantee



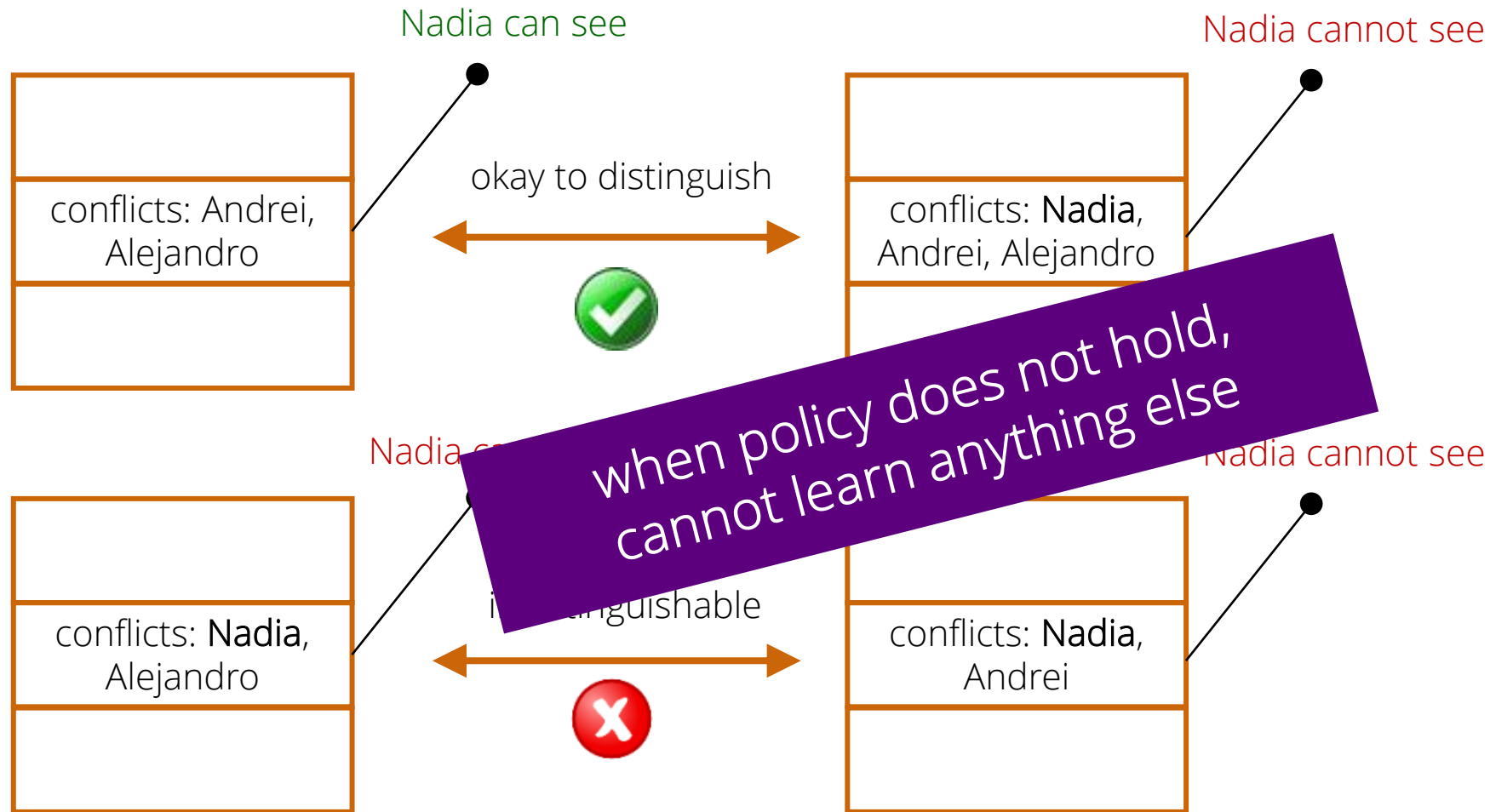
security guarantee



security guarantee



security guarantee

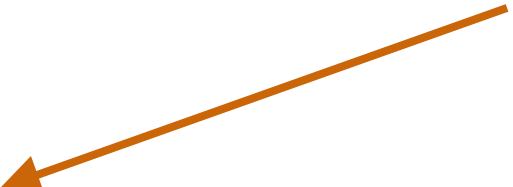


safe downgrading

```
-- Safely downgrade a Boolean  
downgrade :: TIO Bool <i ∧ p> <o> →  
           TIO Bool <i> <o>
```

safe downgrading

"removes" p from input label



-- *Safely downgrade a Boolean*

```
downgrade :: TIO Bool <i  $\wedge$  p> <o>  $\rightarrow$   
           TIO Bool <i> <o>
```

safe downgrading

“removes” p from input label

-- *Safely downgrade a Boolean*

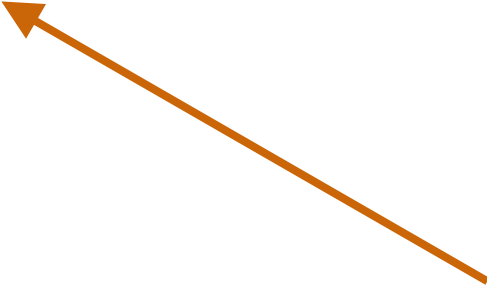
downgrade :: TIO { Bool | $v \Rightarrow p$ } $\langle i \wedge p \rangle \langle o \rangle \rightarrow$
 TIO Bool $\langle i \rangle \langle o \rangle$

safe downgrading

“removes” p from input label

-- *Safely downgrade a Boolean*

downgrade :: TIO { Bool | $v \Rightarrow p$ } $\langle i \wedge p \rangle \langle o \rangle \rightarrow$
 TIO Bool $\langle i \rangle \langle o \rangle$



but can only “leak” the constant False!

demo

<http://comcom.csail.mit.edu/demos/#lifty-chalmers-demo-final>

1. IFC with liquid types
2. programming in Lifty
- 3. leak repair**

example

```
do
  x ← getPaperScore ds pid
  print client (show x)
```

example

how does Lifty repair this leak?

```
do
  x ← getPaperScore ds pid
  print client (show x)
```

example

how does Lifty repair this leak?

do

```
x ← getPaperScore ds pid  
print client (show x)
```

key insight 2:
use type errors to localize leaks!

example

```
bind (getPaperScore ds pid)  
      (\x . print client (show x))
```

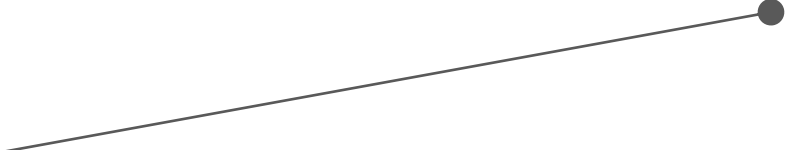
step 1: liquid type checking

```
bind (getPaperScore ds pid)  
    (\x . print client (show x))
```

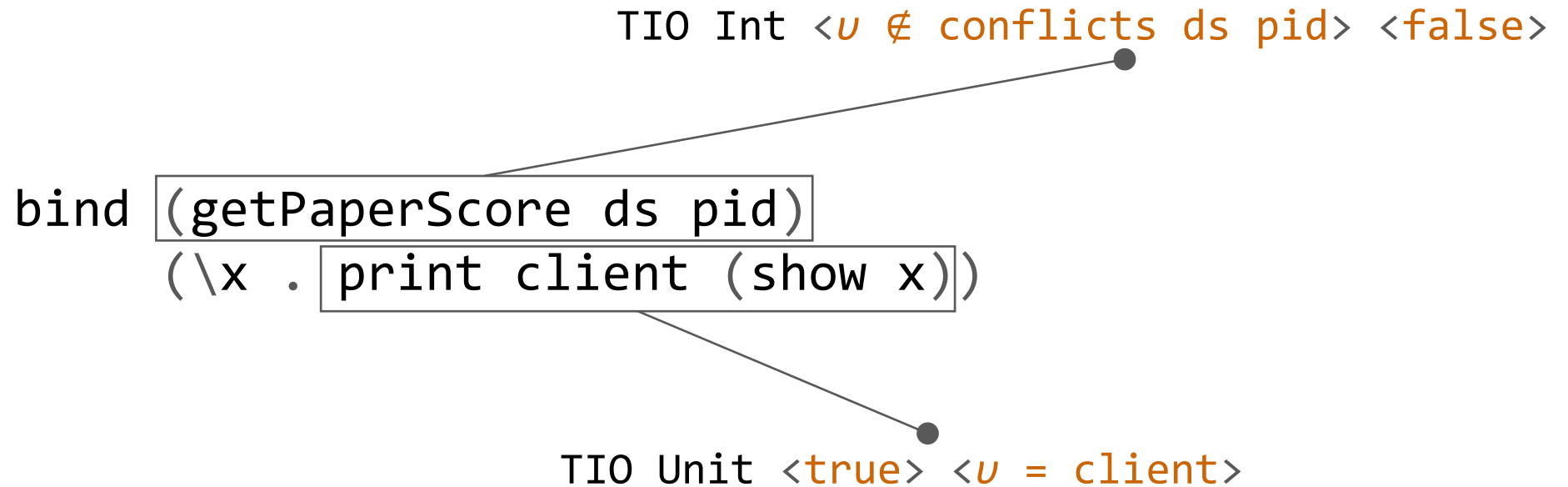
step 1: liquid type checking

TIO Int <*u* *∉ conflicts* ds pid> <false>

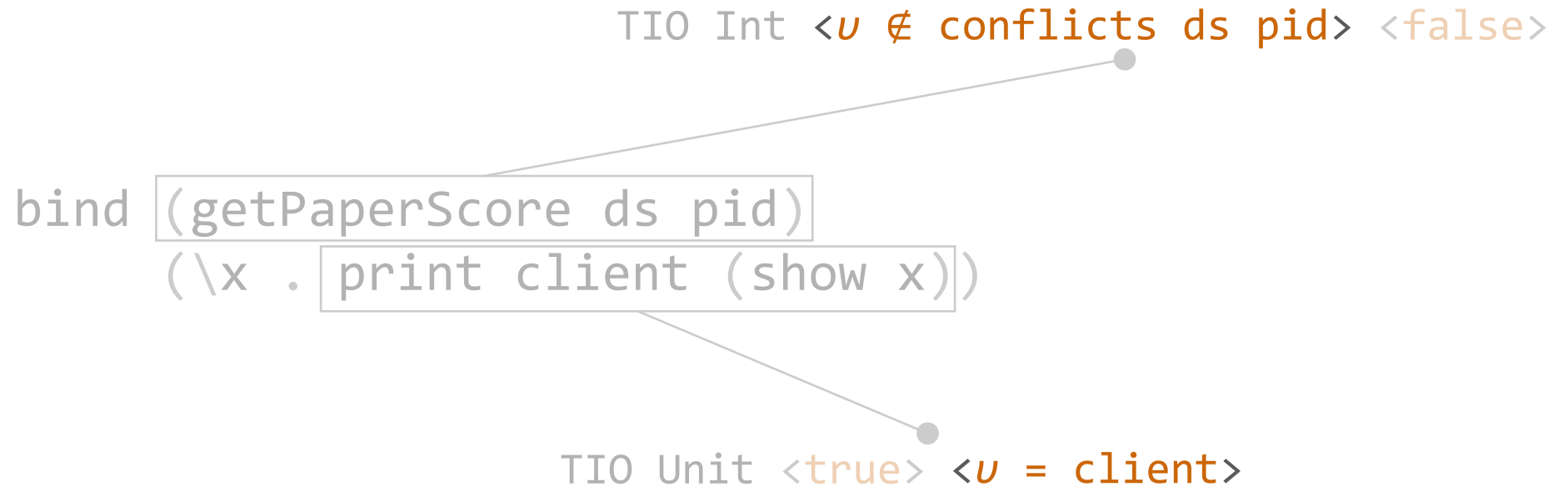
bind (getPaperScore ds pid)
(\x . print client (show x))



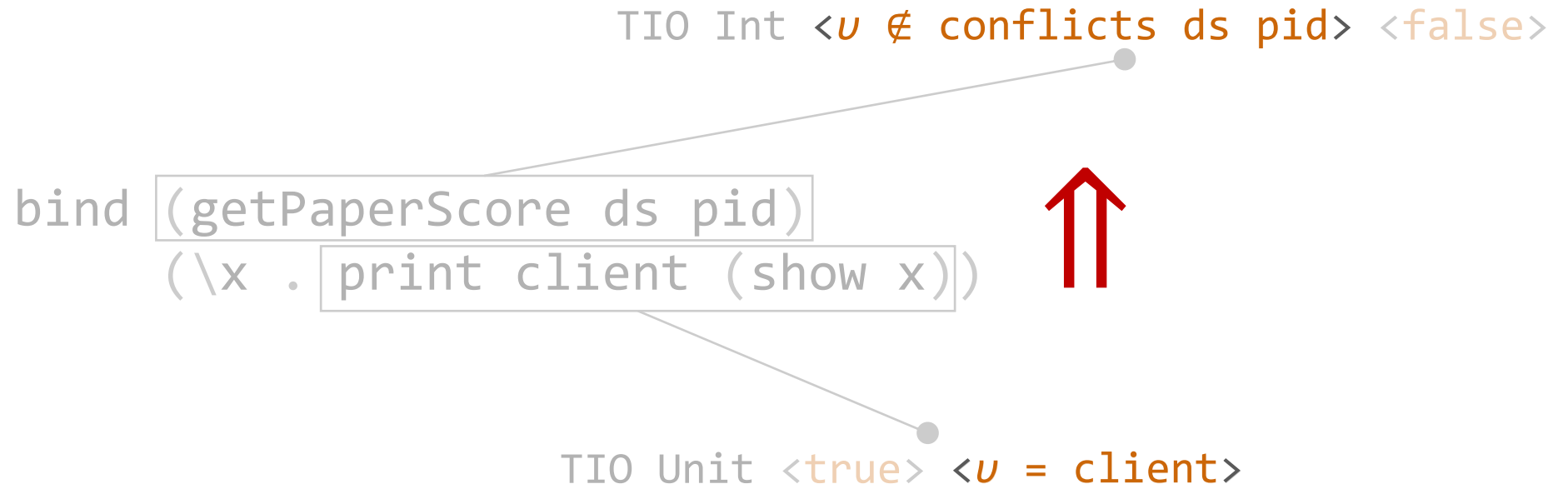
step 1: liquid type checking



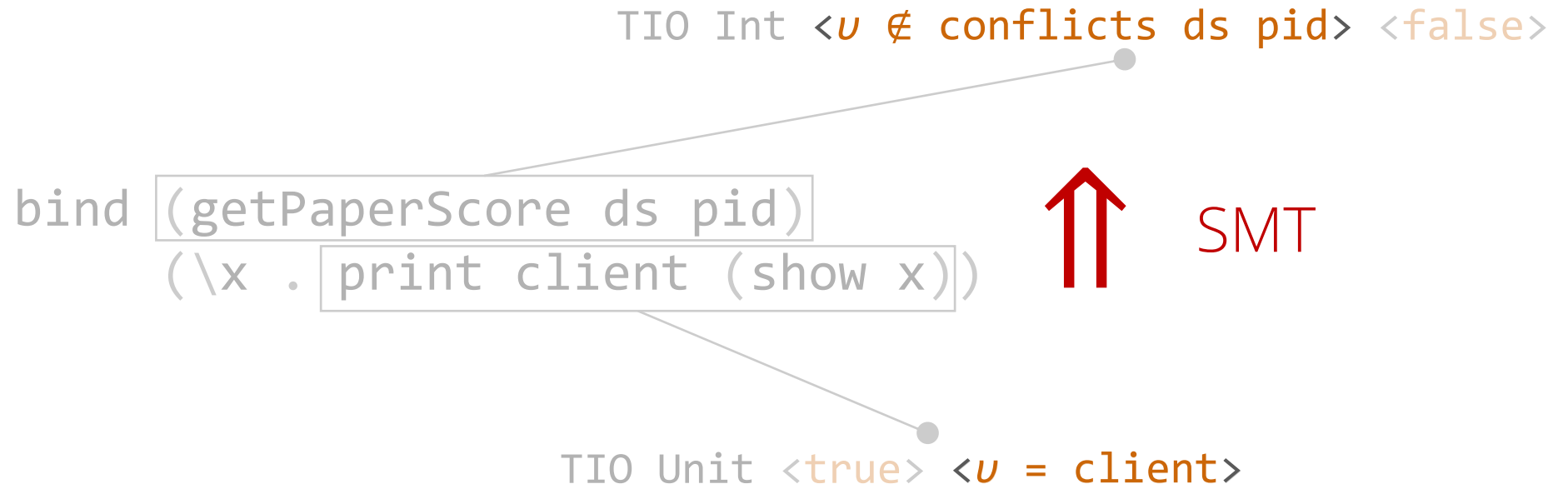
step 1: liquid type checking



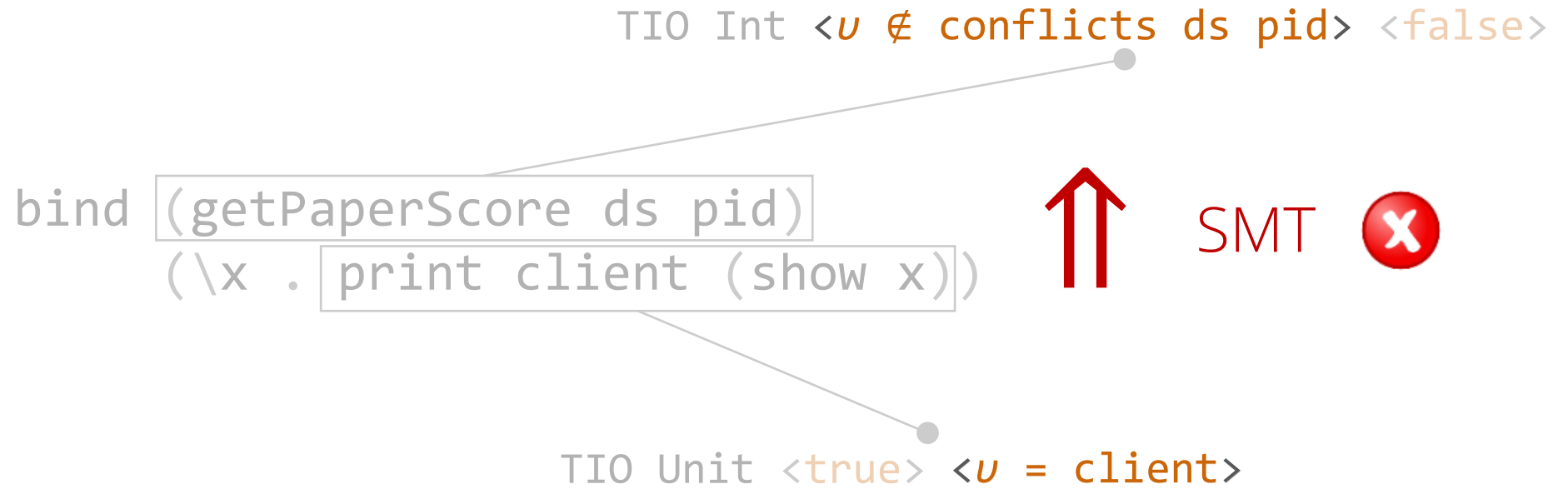
step 1: liquid type checking



step 1: liquid type checking



step 1: liquid type checking



step 1: liquid type checking

type error

```
bind (getPaperScore ds pid)  
(\x . print client (show x))
```

TIO Int $\langle u \notin \text{conflicts } ds \text{ pid} \rangle \langle \text{false} \rangle$

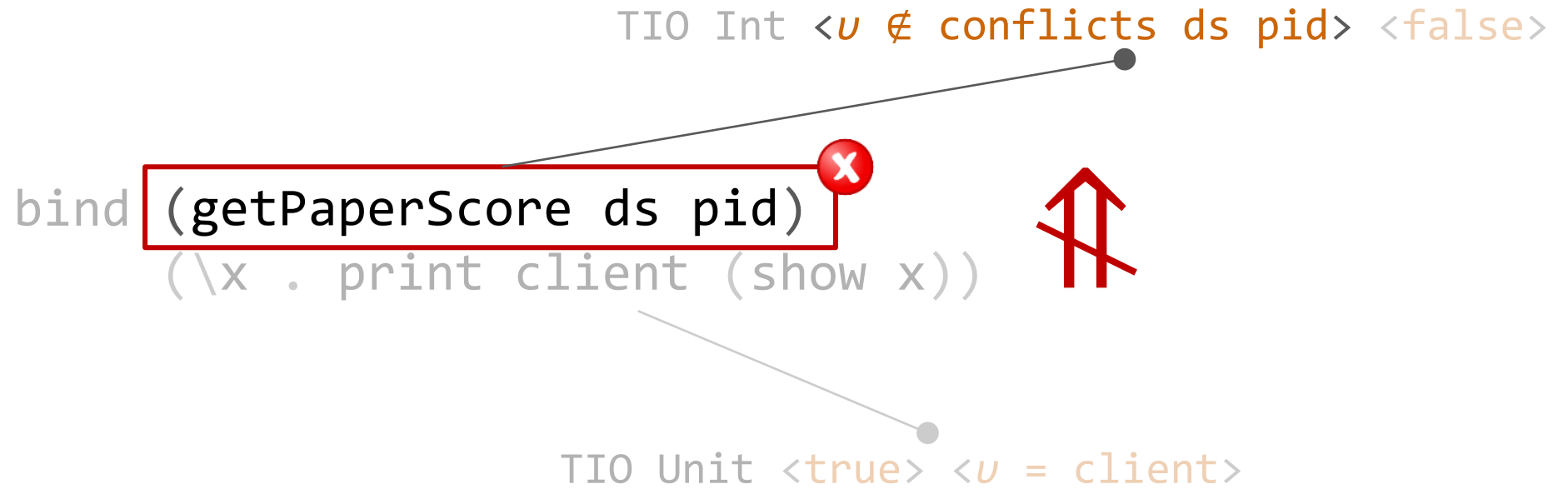


SMT



TIO Unit $\langle \text{true} \rangle \langle u = \text{client} \rangle$

step 2: error localization



step 3: infer local specification

```
if ??  
  then getPaperScore ds pid  
  else ??
```

```
bind (getPaperScore ds pid)  
  (\x . print client (show x))
```

TIO Unit <true> <u = client>

step 3: infer local specification

```
if ??      -- policy holds  
then getPaperScore ds pid  
else ??
```

```
bind (getPaperScore ds pid)  
(\x . print client (show x))
```

TIO Unit <true> <*u* = client>

step 3: infer local specification

```
if ??      -- policy holds
then getPaperScore ds pid
else ??      -- default
```

```
bind (getPaperScore ds pid)
      (\x . print client (show x))
```

TIO Unit <true> <*u* = client>

step 3: infer local specification

```
if ??      -- policy holds
then getPaperScore ds pid
else ??    -- default
```

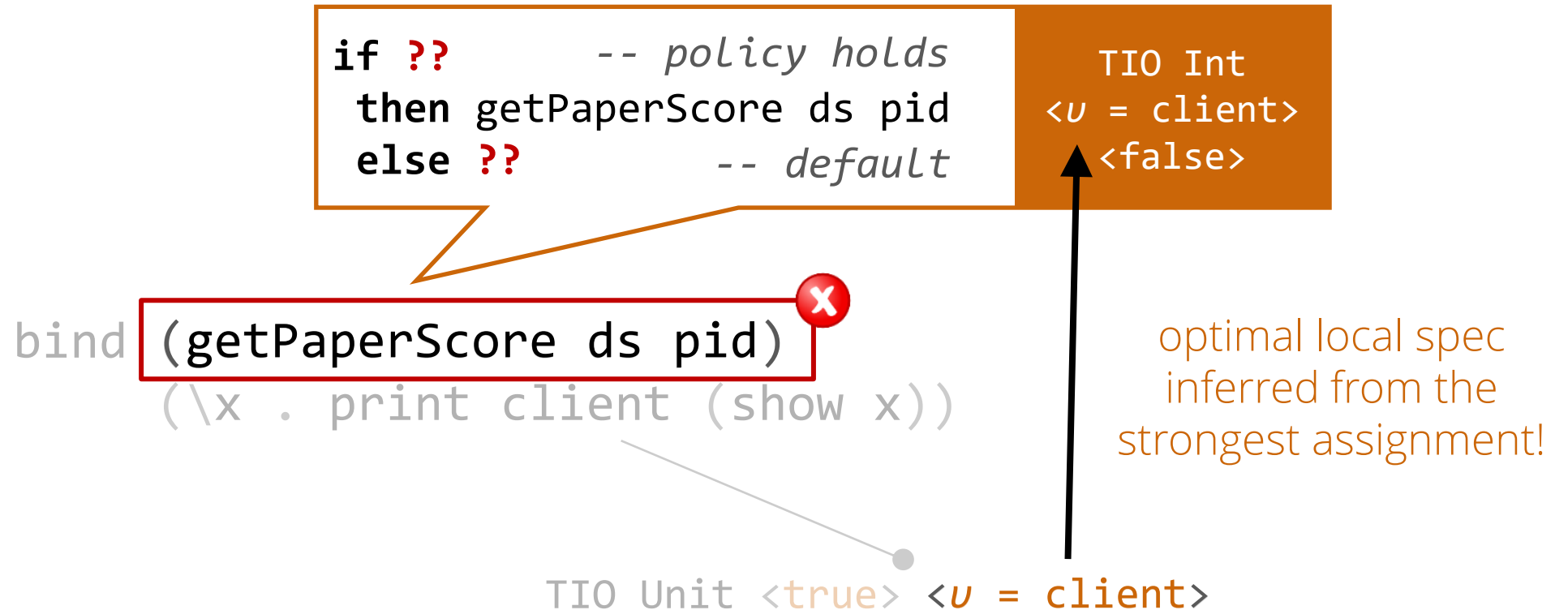
```
TIO Int
<u = client>
<false>
```

```
bind (getPaperScore ds pid)
      (\x . print client (show x))
```



TIO Unit <true> <u = client>

step 3: infer local specification



step 4: type-driven synthesis*

```
if ??  
  then getPaperScore ds pid  
  else ??
```

TIO Int
<u = client>
<false>

```
bind (getPaperScore ds pid)  
  (\x . print client (show x))
```

* [Polikarpova, Kuraj, Solar-Lezama. 2016]

step 4: type-driven synthesis*

```
do
  cs ← getPaperConflicts ds pid
  if not (elem client cs)
  then getPaperScore ds pid
  else ??
```

TIO Int
<v = client>
<false>

```
bind (getPaperScore ds pid)
      (\x . print client (show x))
```

* [Polikarpova, Kuraj, Solar-Lezama. 2016]

step 4: type-driven synthesis*

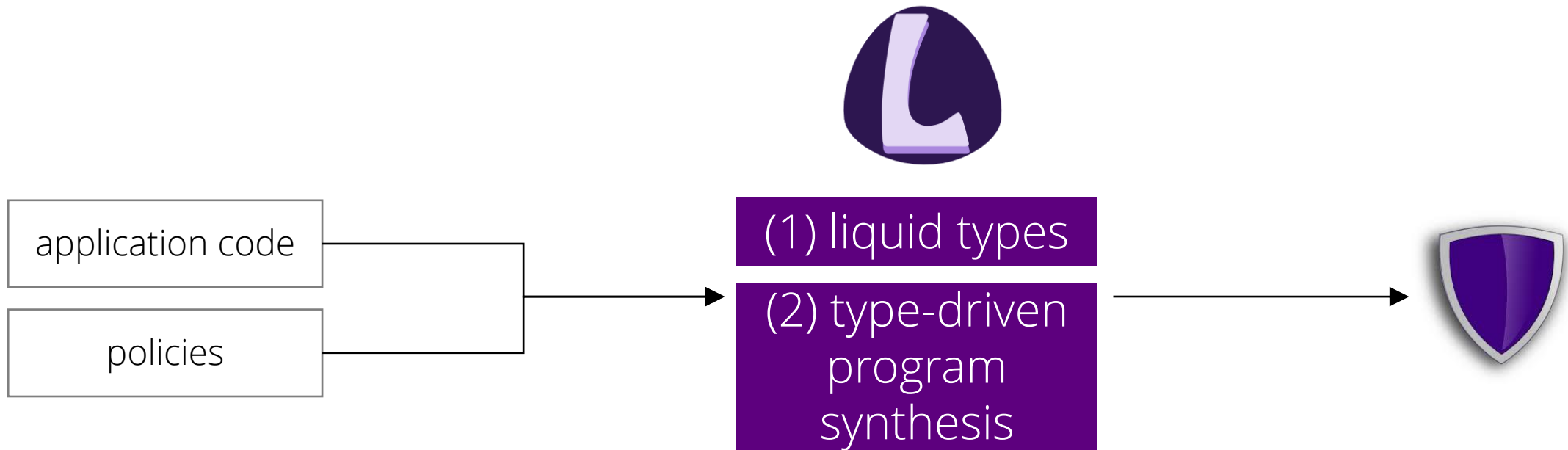
```
do
  cs ← getPaperConflicts ds pid
  if not (elem client cs)
    then getPaperScore ds pid
    else return 0
```

TIO Int
<v = client>
<false>

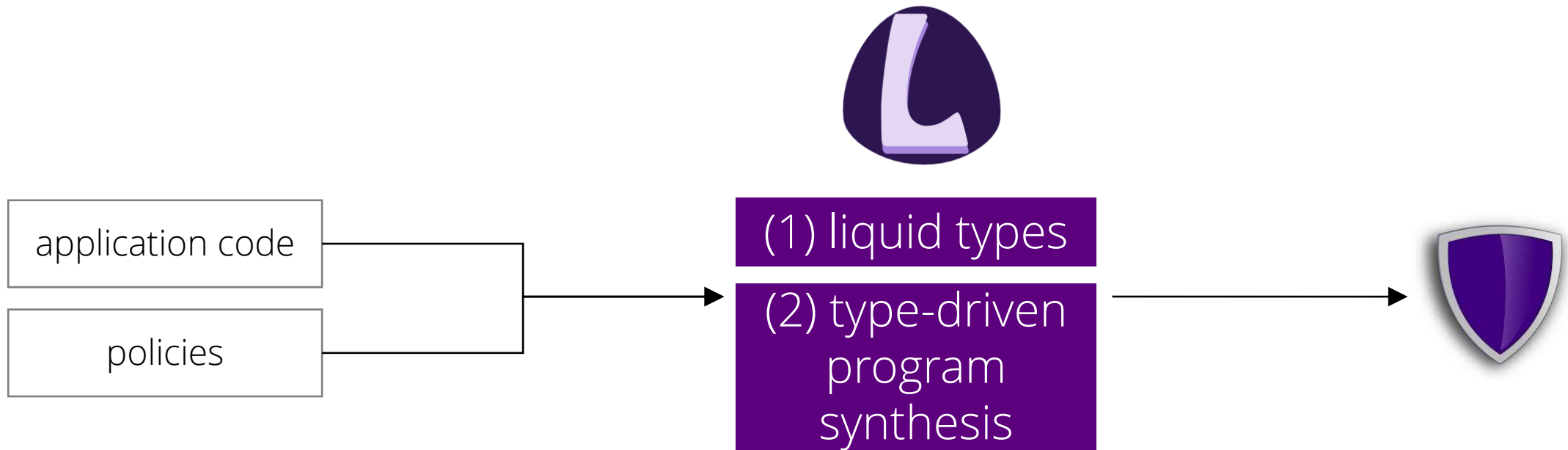
```
bind (getPaperScore ds pid)
  (\x . print client (show x))
```

* [Polikarpova, Kuraj, Solar-Lezama. 2016]

lifty



lifty



<https://cseweb.ucsd.edu/~npolikarpova/lifty/>

ongoing work

STORM

(Security-Typed ORM)

ongoing work

STORM

(Security-Typed ORM)



ongoing work

STORM

(Security-Typed ORM)



+

ORM

ongoing work

STORM

(Security-Typed ORM)



+

ORM

+



ongoing work

STORM

(Security-Typed ORM)



+

ORM

+



= secure web framework for Haskell