

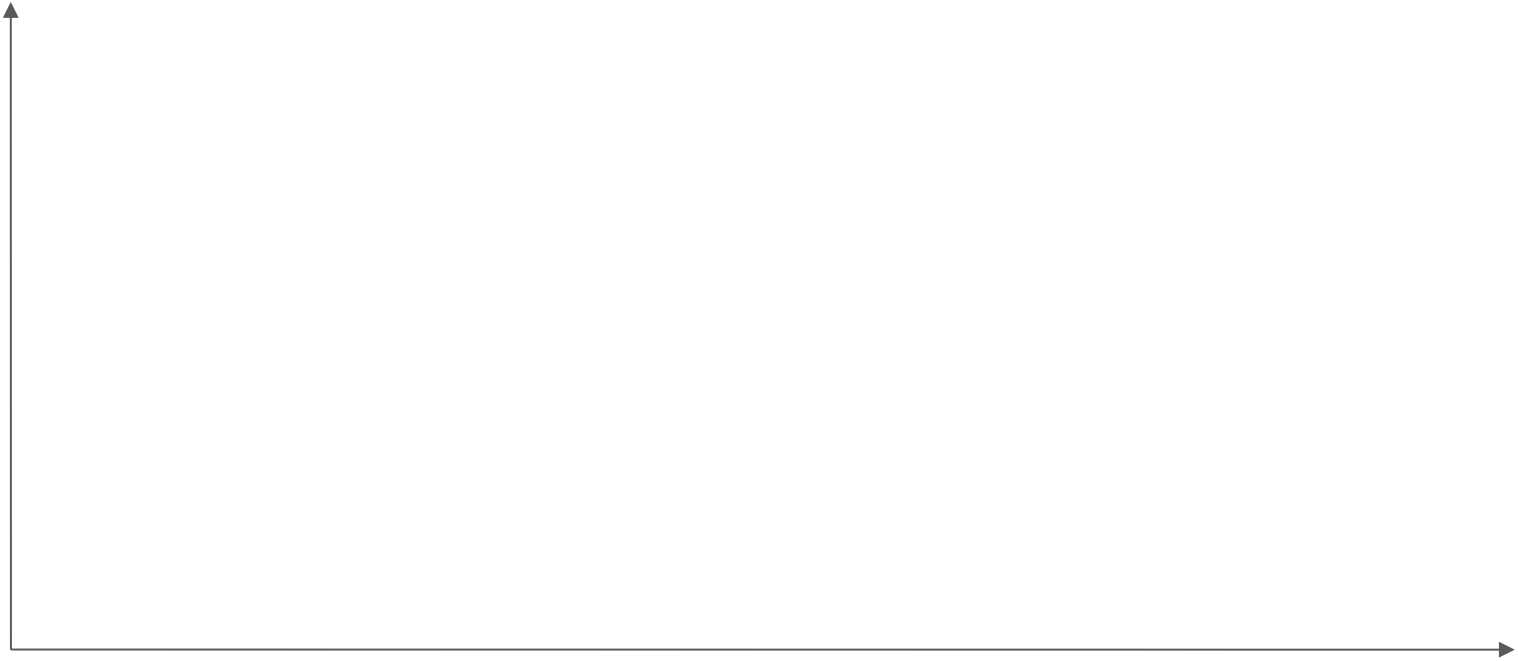
Formal Foundations of the Viper Verification Infrastructure

Thibault Dardinier

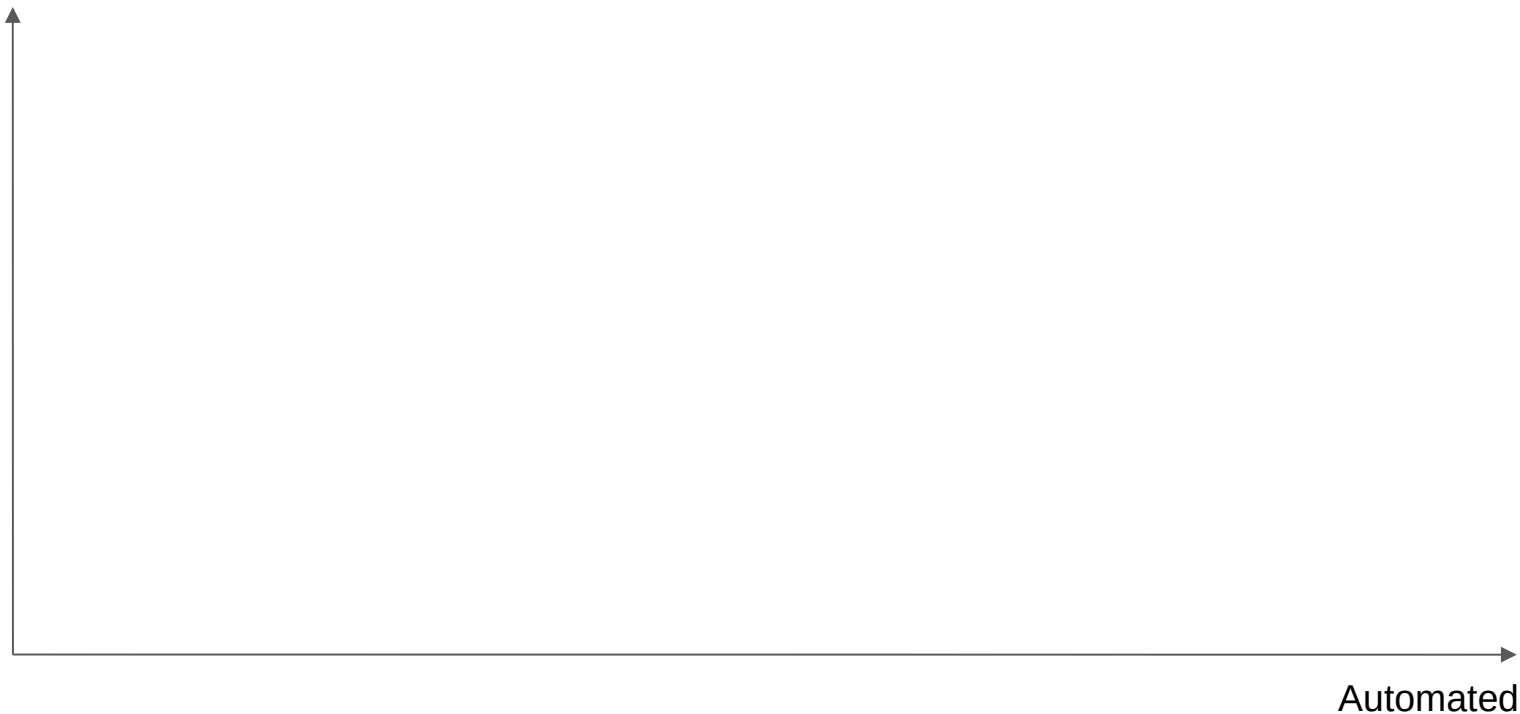
ETH zürich

Joint work with Gaurav Parthasarathy, Alex Summers, Peter Müller

Program Verifiers Based on Separation Logic (SL)



Program Verifiers Based on Separation Logic (SL)



Program Verifiers Based on Separation Logic (SL)

Foundational



Automated



Program Verifiers Based on Separation Logic (SL)



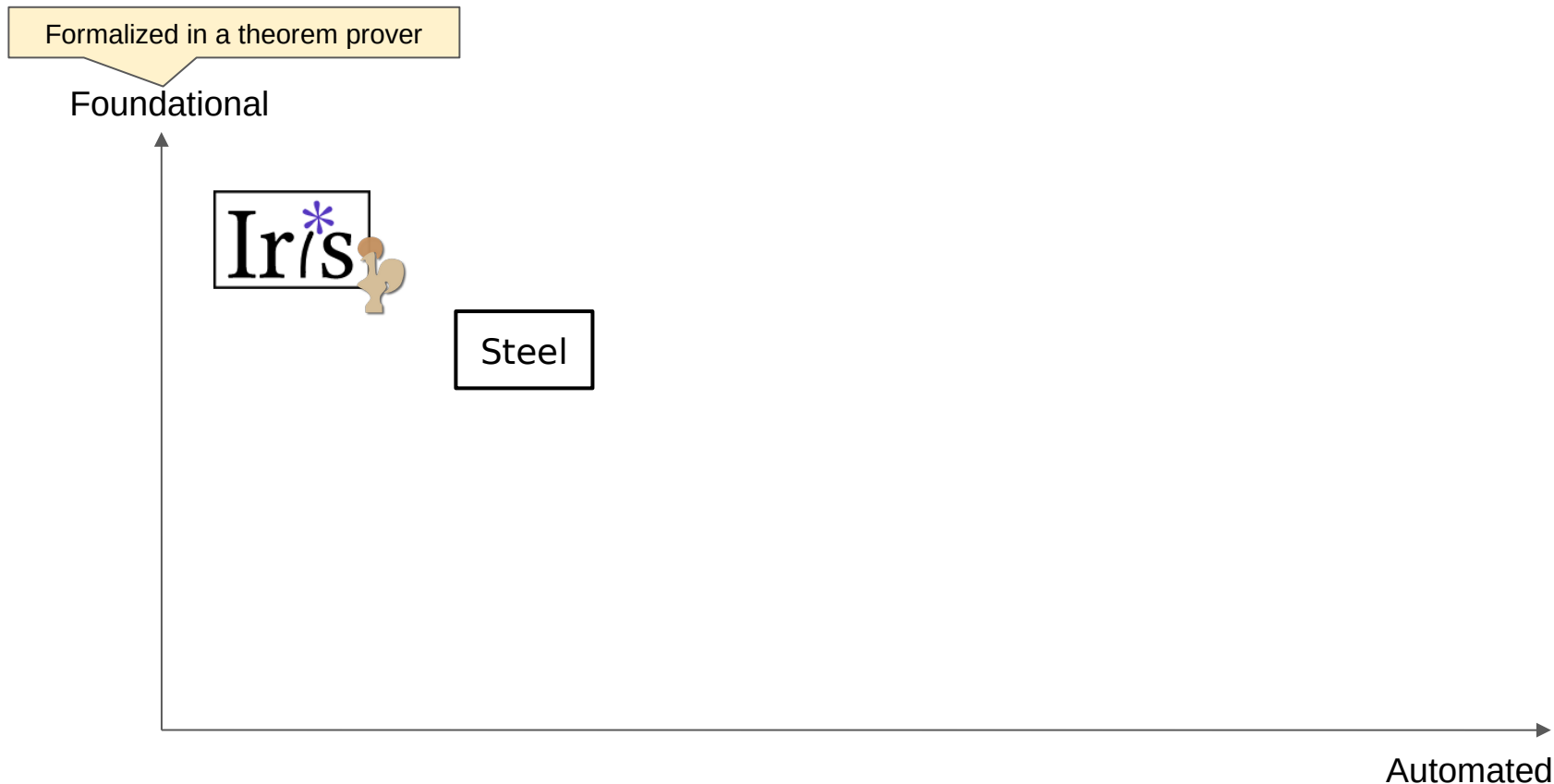
Program Verifiers Based on Separation Logic (SL)



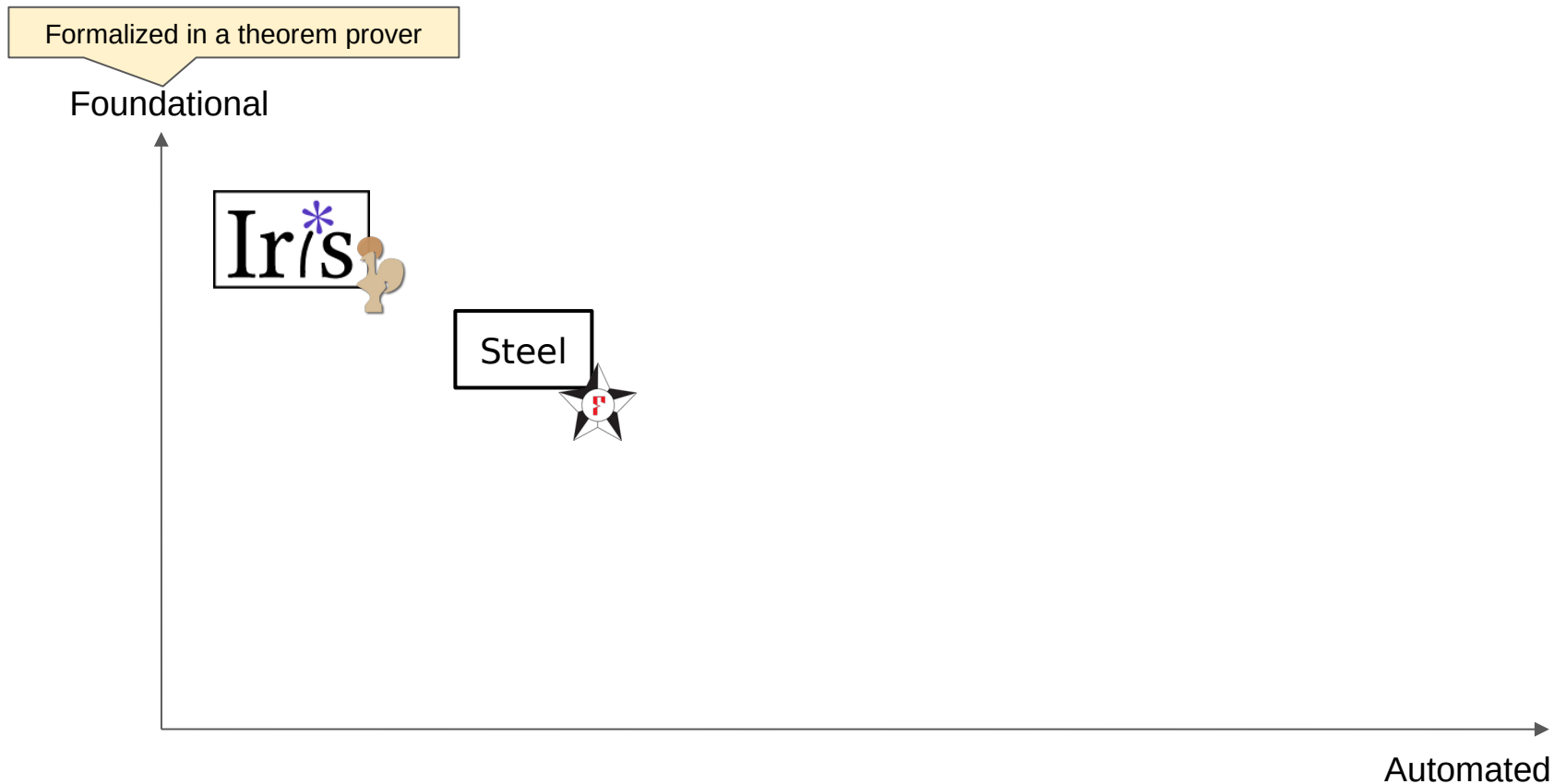
Program Verifiers Based on Separation Logic (SL)



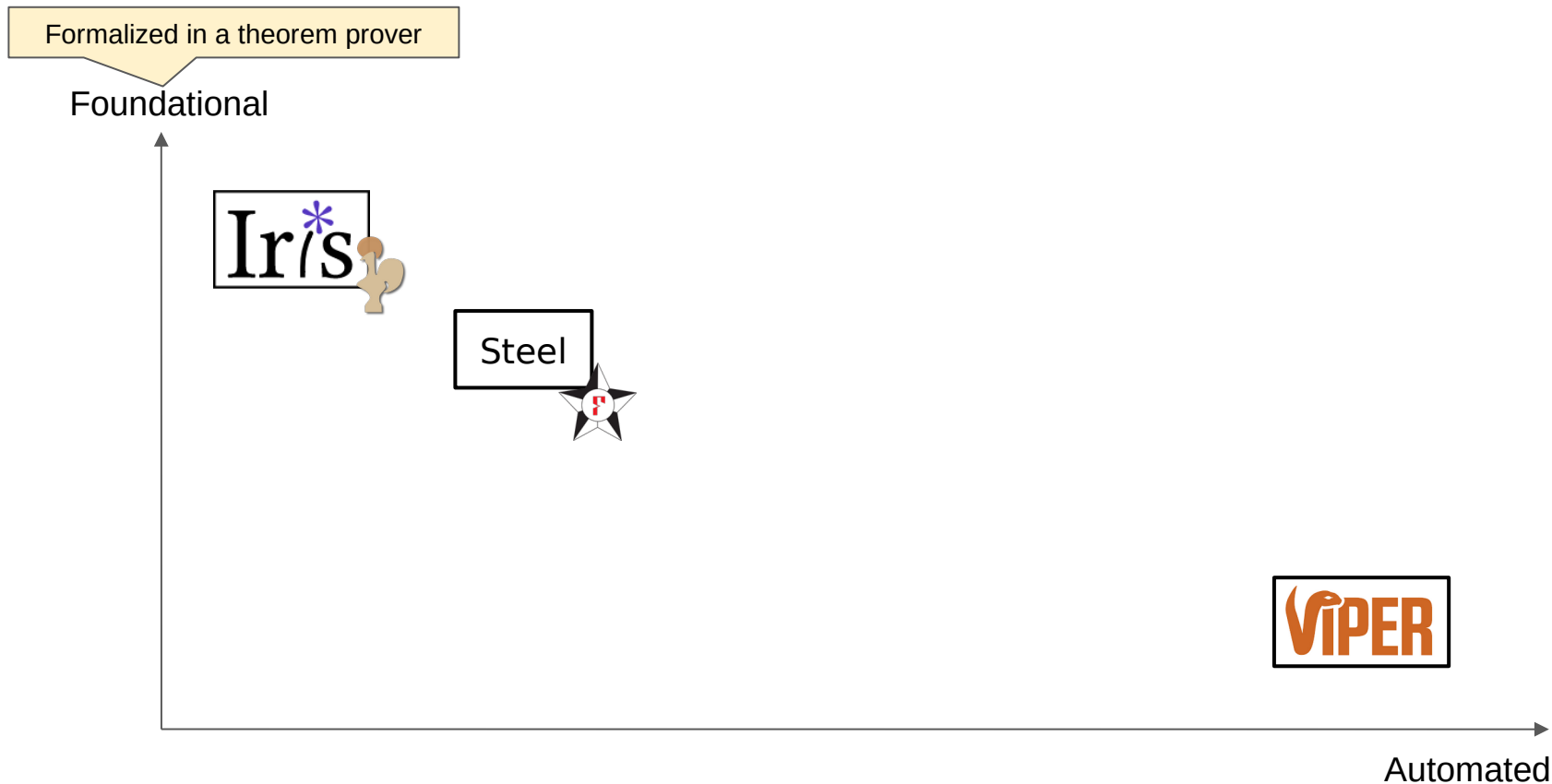
Program Verifiers Based on Separation Logic (SL)



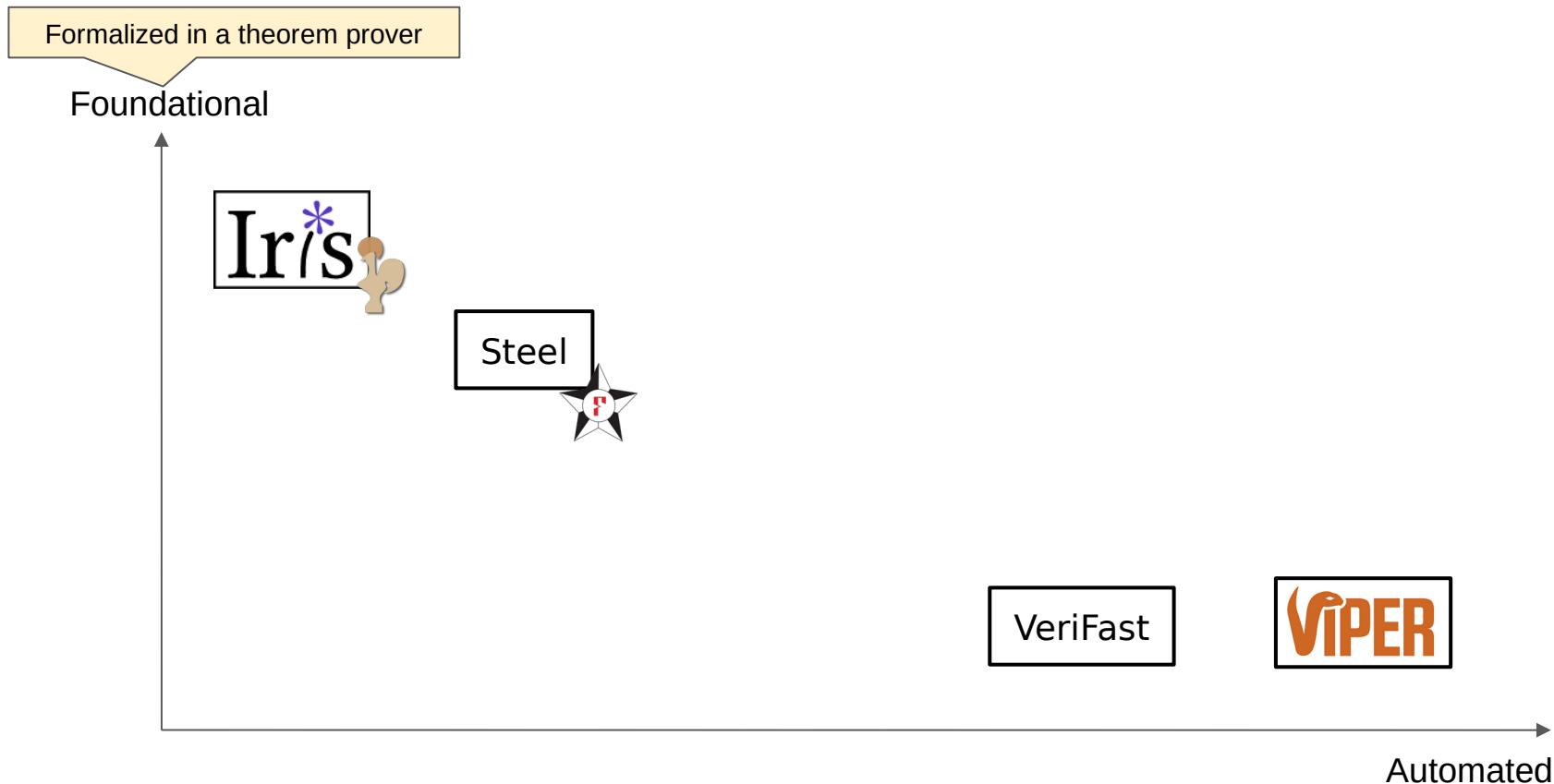
Program Verifiers Based on Separation Logic (SL)



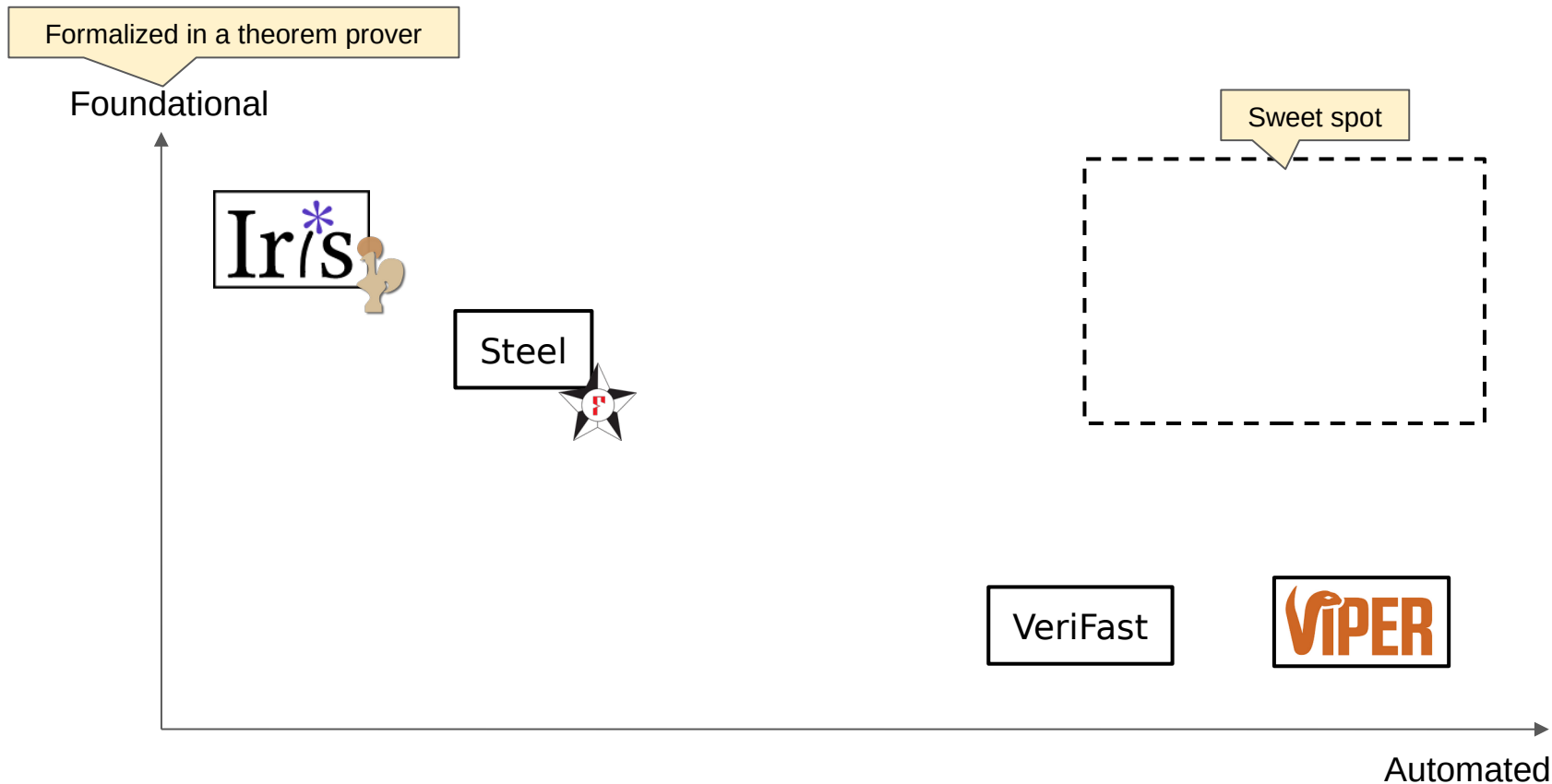
Program Verifiers Based on Separation Logic (SL)



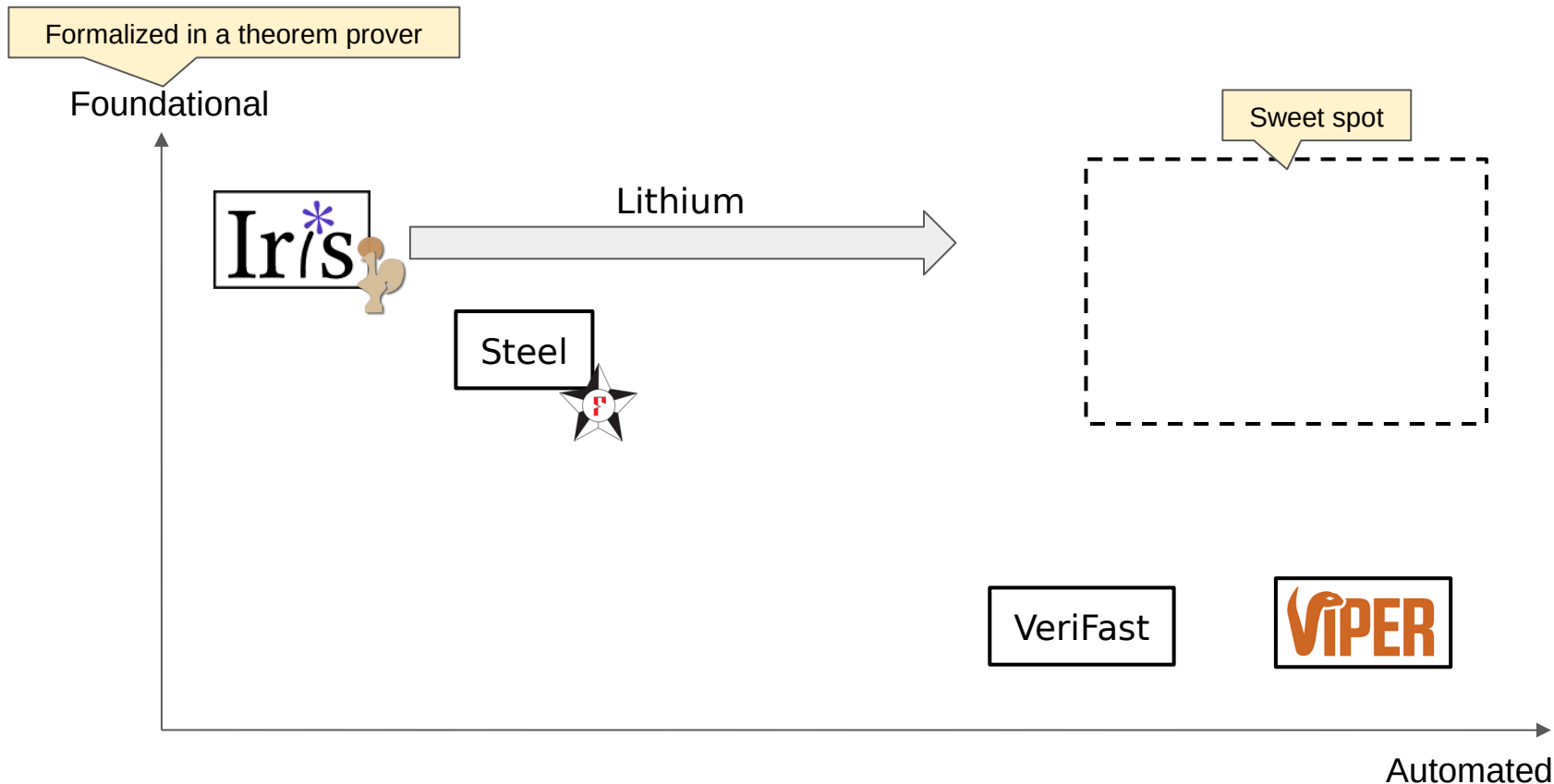
Program Verifiers Based on Separation Logic (SL)



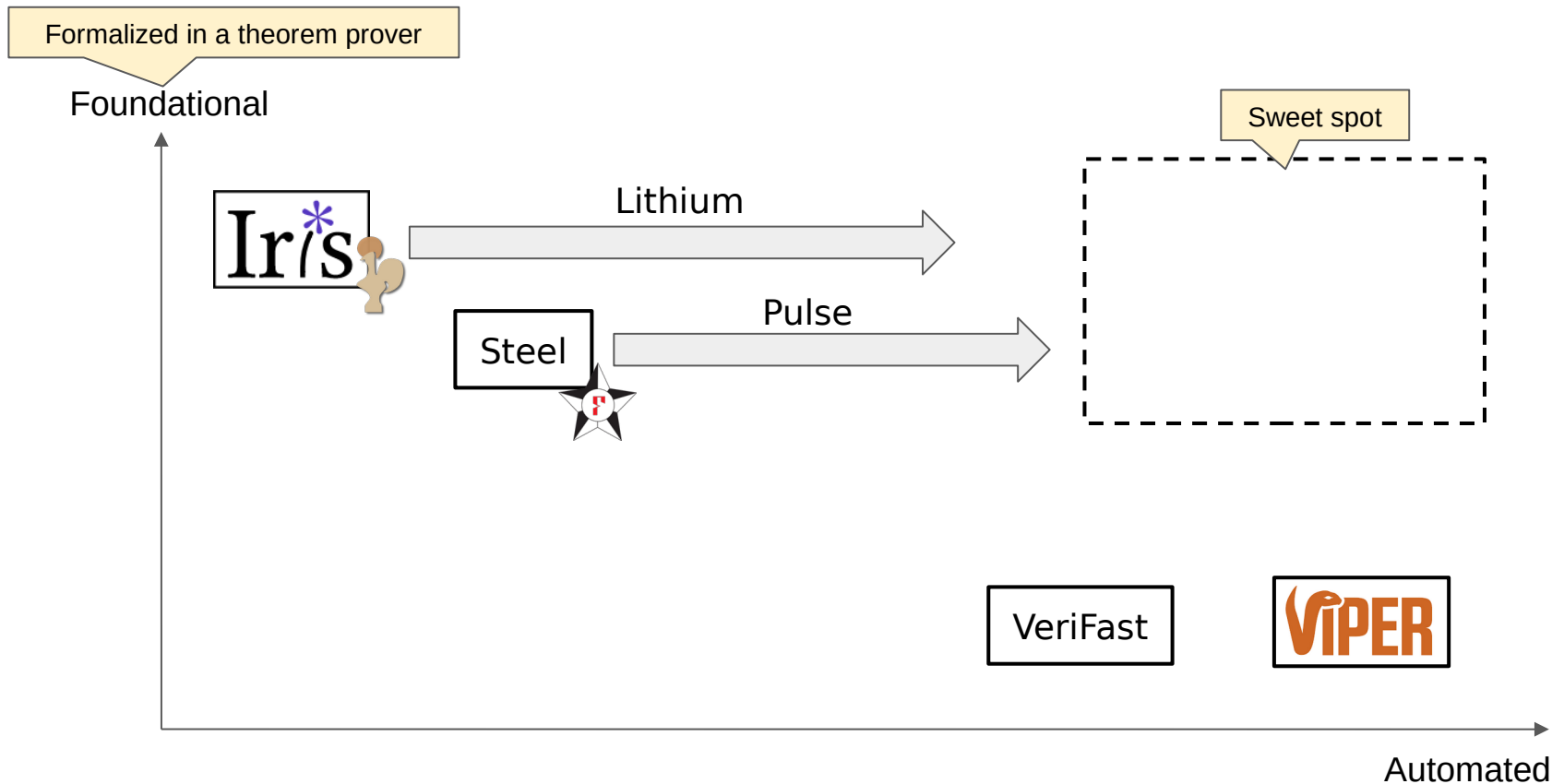
Program Verifiers Based on Separation Logic (SL)



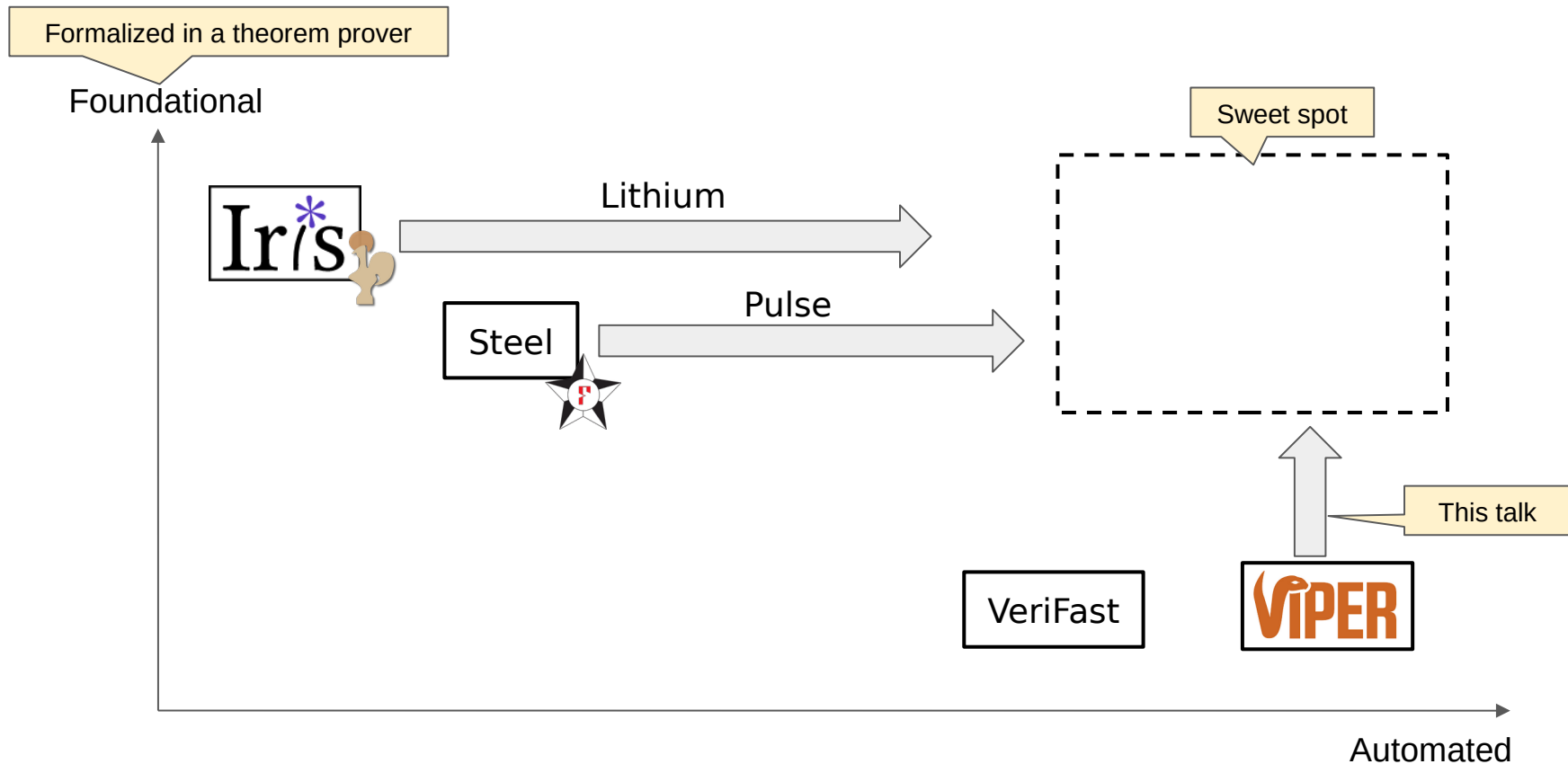
Program Verifiers Based on Separation Logic (SL)



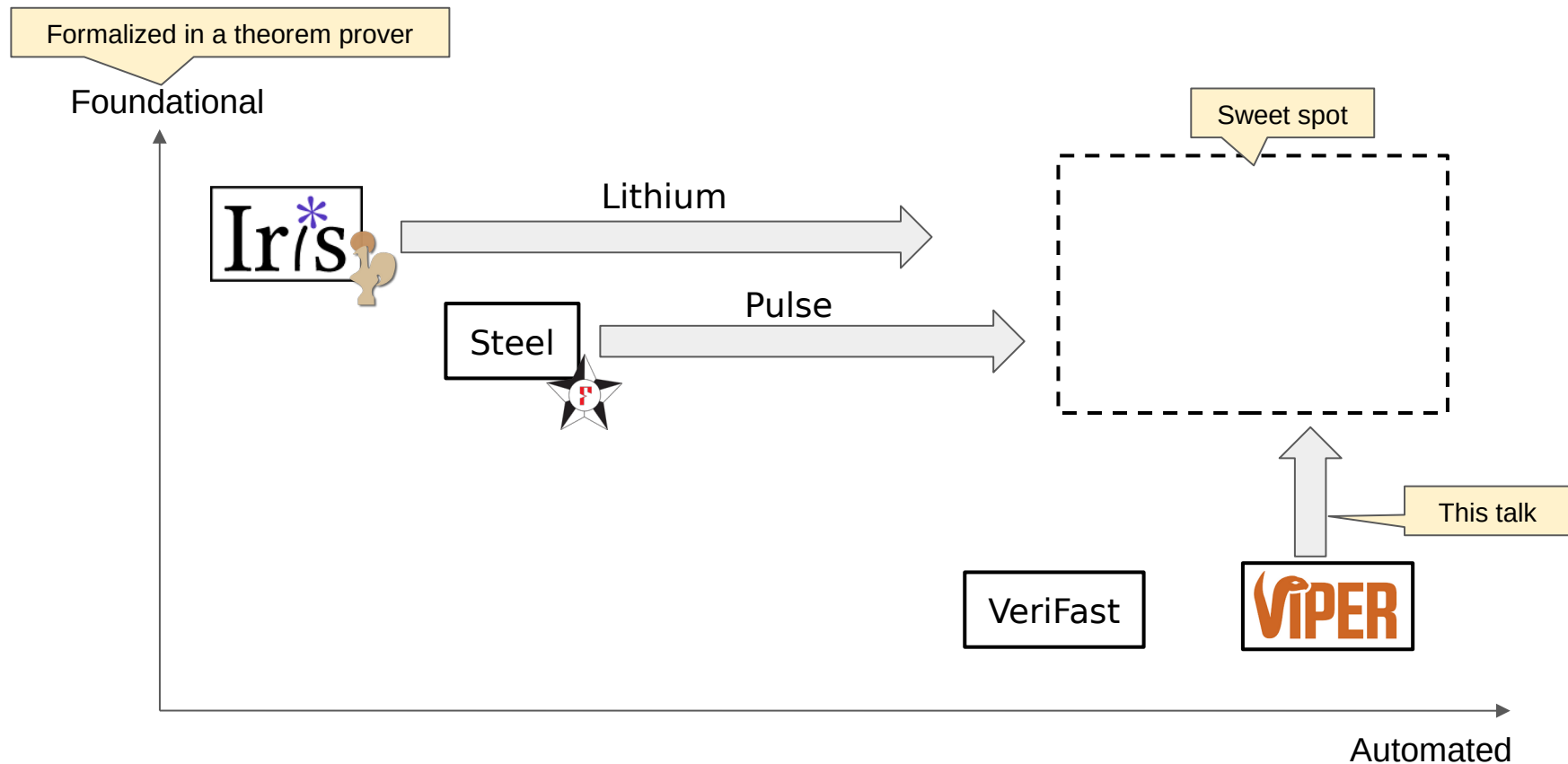
Program Verifiers Based on Separation Logic (SL)



Program Verifiers Based on Separation Logic (SL)



Program Verifiers Based on Separation Logic (SL)



*highly non-exhaustive (missing GRASShopper, Gillian, VST, Diaframe ...)

Outline of the Talk

Outline of the Talk

1. Overview of Viper

Outline of the Talk

1. Overview of Viper
2. Inhale and Exhale: An Operational View of Separation Logic

Outline of the Talk

1. Overview of Viper
2. Inhale and Exhale: An Operational View of Separation Logic
3. Toward a Foundational Viper

Outline of the Talk

1. Overview of Viper

2. Inhale and Exhale: An Operational View of Separation Logic

3. Toward a Foundational Viper

Demo

```
1 field x: Int
2 field y: Int
3
4 method main(point: Ref)
5   requires acc(point.x) && acc(point.y)
6   // point.x |-> _ * point.y |-> _
7   {
8     point.x := 5
9     point.y := 7
10    add(point)
11    assert point.x == 5
12    assert point.y == 12
13  }
14
15 method add(p: Ref)
16   requires acc(p.x, 1/2) && acc(p.y)
17   ensures acc(p.x, 1/2) && acc(p.y)
18   // ensures p.y == old(p.x + p.y)
19   {
20     p.y := p.x + p.y
21   }
```

The Viper Verification Infrastructure

The Viper Verification Infrastructure

front-end
program



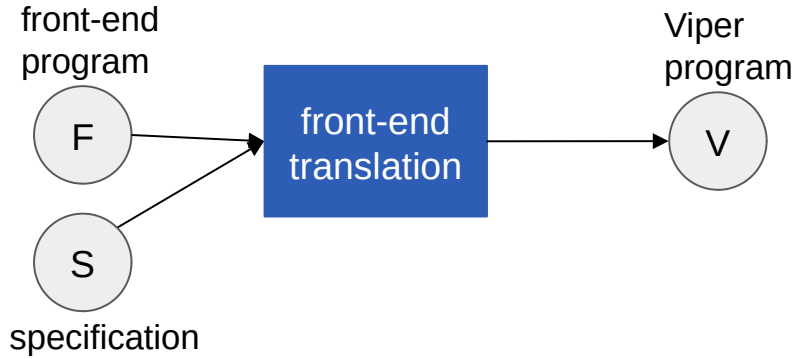
The Viper Verification Infrastructure

front-end
program

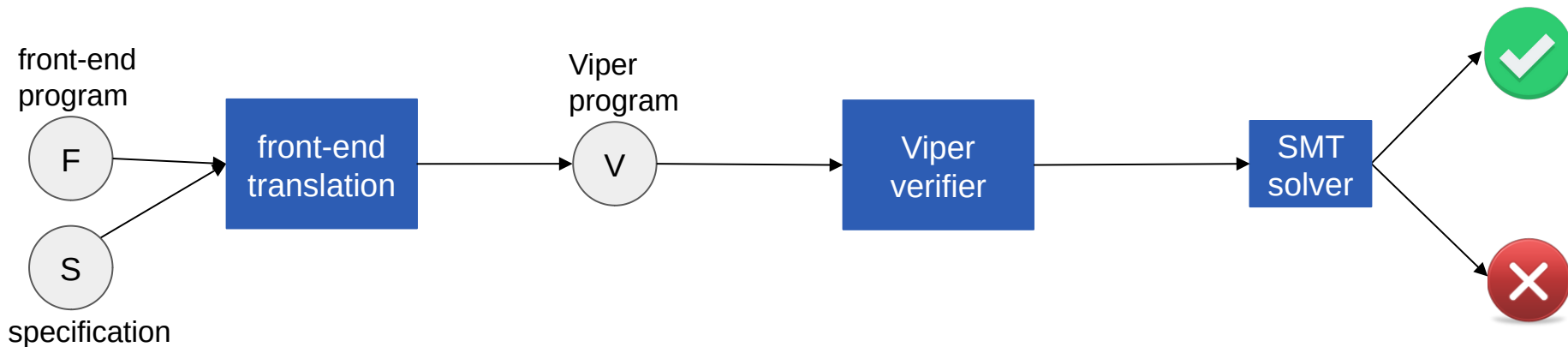


specification

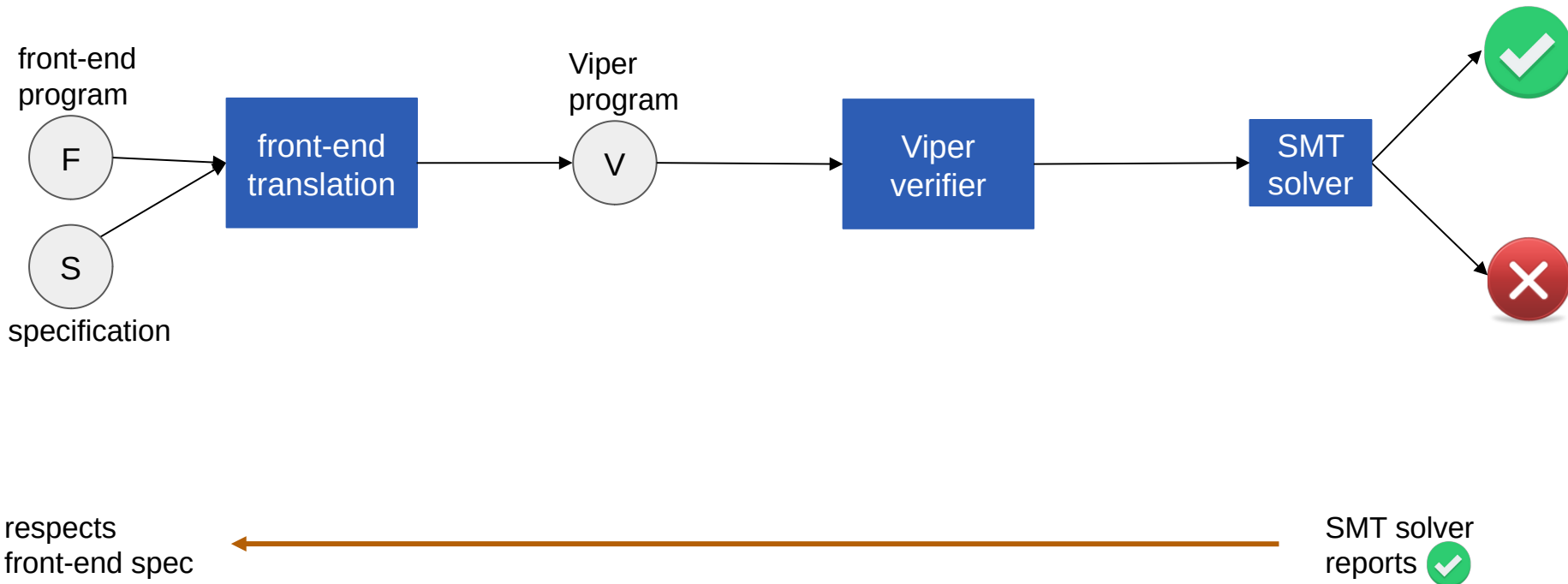
The Viper Verification Infrastructure



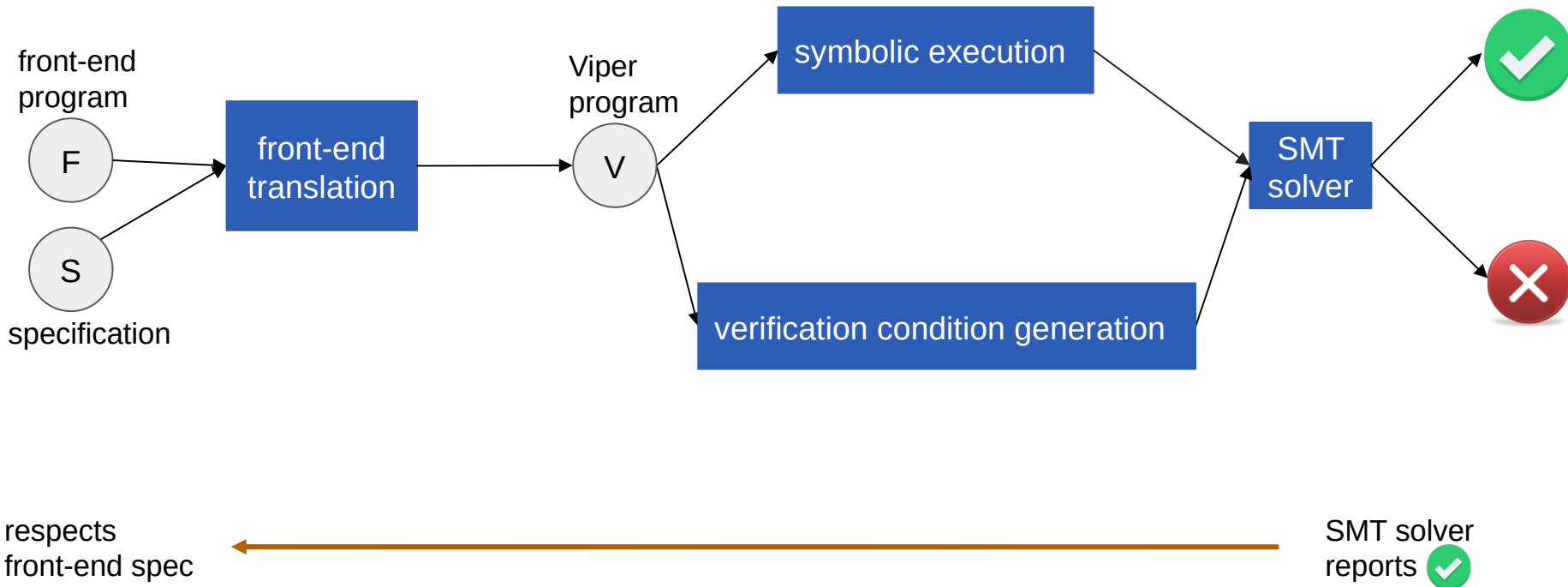
The Viper Verification Infrastructure



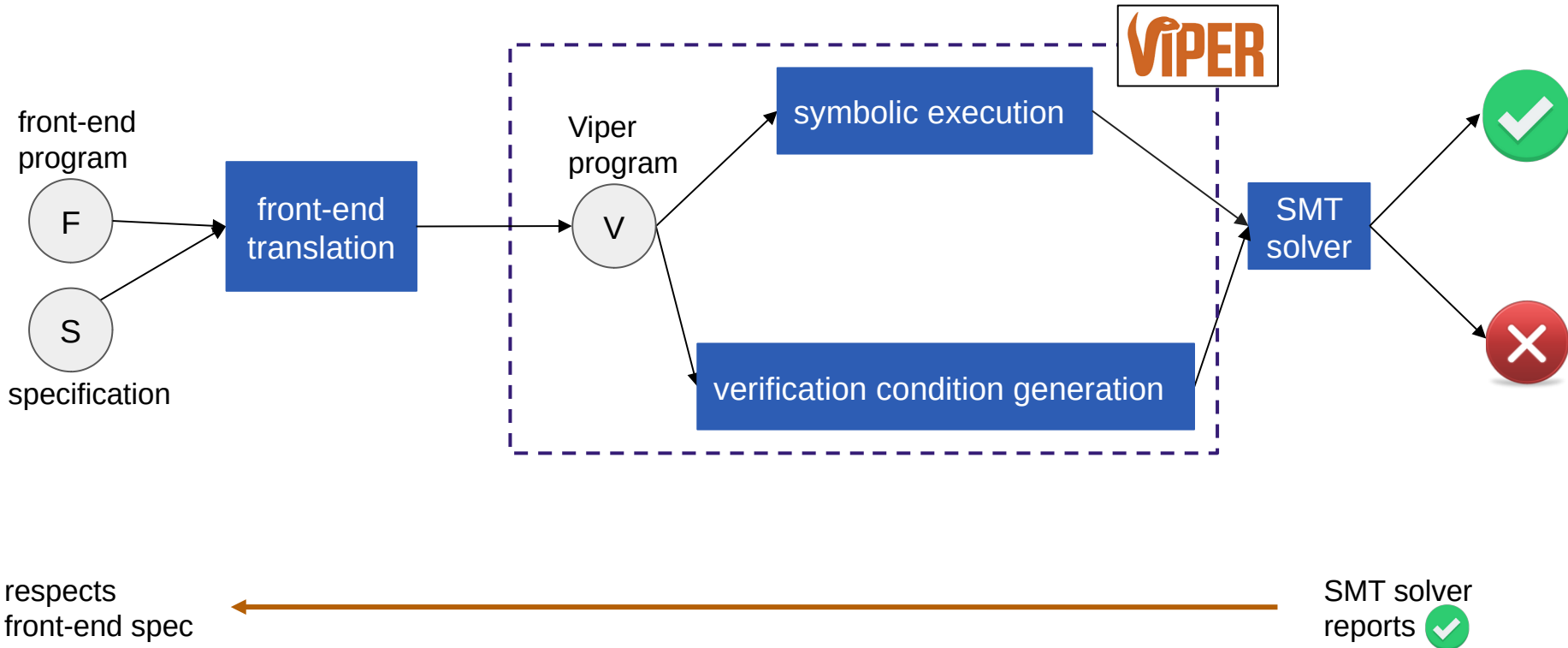
The Viper Verification Infrastructure



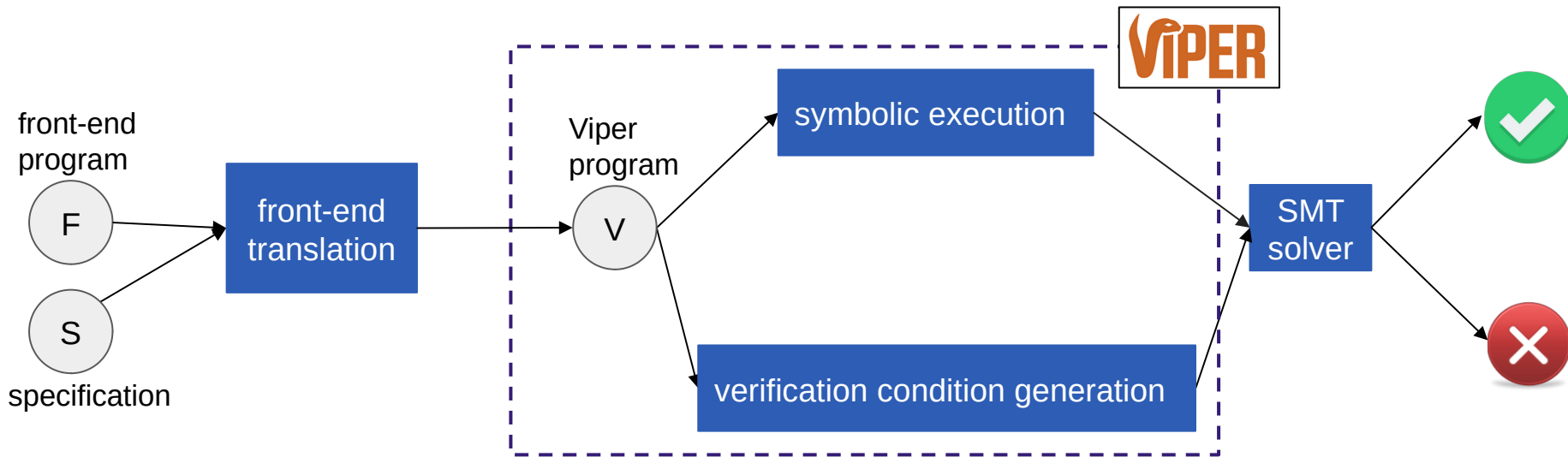
The Viper Verification Infrastructure



The Viper Verification Infrastructure



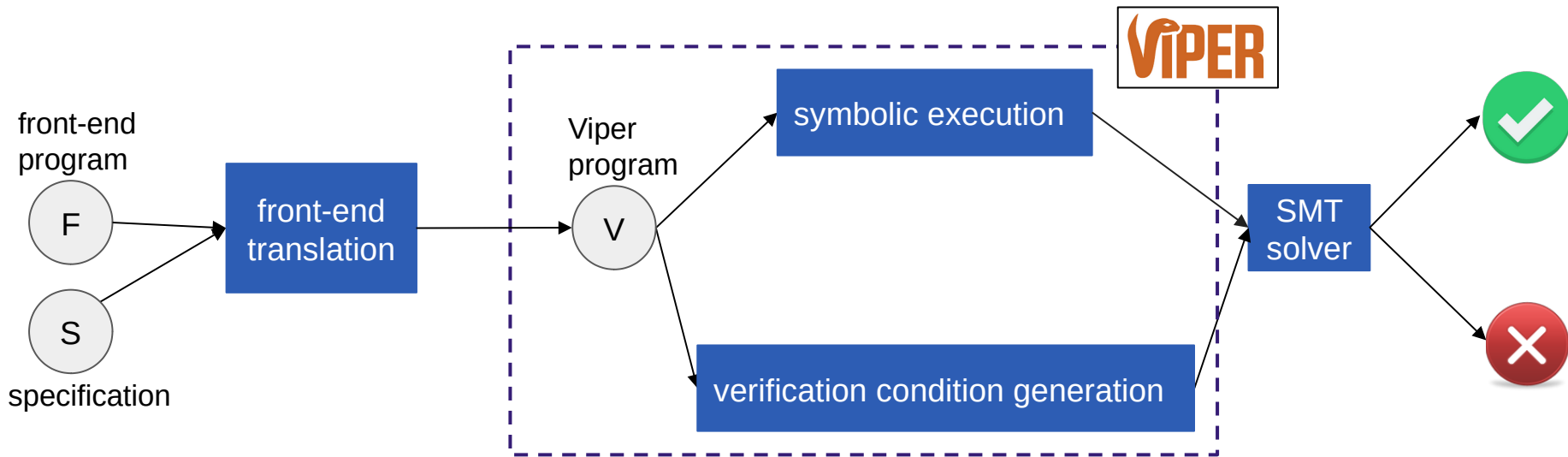
The Viper Verification Infrastructure



Program verifiers built on top of Viper

<i>Prusti</i> (Rust)	<i>Gobra</i> (Go)	<i>Nagini</i> (Python)
<i>VerCors</i> (Java, C, OpenCL, OpenMP)	Smart contracts	RSL, FSL, FSL++
Secure information flow	Gradual verification	...

The Viper Verification Infrastructure



Program verifiers built on top of Viper

Verification of the SCION Internet architecture
(existing router implementation ~5k LOC)

<i>Prusti</i> (Rust)	<i>Gobra</i> (Go)	<i>Nagini</i> (Python)
<i>VerCors</i> (Java, C, OpenCL, OpenMP)	Smart contracts	RSL, FSL, FSL++
Secure information flow	Gradual verification	...

Overview of the Viper Language

Program Code

Assertion Language

Verification Features

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

Verification Features

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

- Fractional permissions

Verification Features

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

- Fractional permissions
- Inductive predicates

Verification Features

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

- Fractional permissions
- Inductive predicates
- Iterated separating conjunction

Verification Features

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

- Fractional permissions
- Inductive predicates
- Iterated separating conjunction
- Magic wands

Verification Features

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

- Fractional permissions
- Inductive predicates
- Iterated separating conjunction
- Magic wands
- ...

Verification Features

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

- Fractional permissions
- Inductive predicates
- Iterated separating conjunction
- Magic wands
- ...

Verification Features

- Standard contract features
- Inhale and exhale
- ...

Mathematical Background

Overview of the Viper Language

Program Code

- Sequential, imperative language
- Standard control structures
- Basic type system
- Built-in heap

Assertion Language

- Fractional permissions
- Inductive predicates
- Iterated separating conjunction
- Magic wands
- ...

Verification Features

- Standard contract features
- Inhale and exhale
- ...

Mathematical Background

- Predefined and user-defined datatypes
- Uninterpreted functions
- Axioms

Outline of the Talk

1. Overview of Viper

2. Inhale and Exhale: An Operational View of Separation Logic

3. Toward a Foundational Viper

Verification Primitives: Inhale and Exhale

Verification Primitives: Inhale and Exhale

inhale A

exhale A

Verification Primitives: Inhale and Exhale



Verification Primitives: Inhale and Exhale



Adds resources specified by A to the current context

Verification Primitives: Inhale and Exhale



Adds resources specified by A to the current context

Removes resources specified by A from the current context

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically		
Operationally		

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{ inhale } A \{P * A\}$	
Operationally		

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{ inhale } A \{P * A\}$	
Operationally		

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	
Operationally		

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$
Operationally		

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$
Operationally		

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally		

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally		

Acting on a separation
logic state
(e.g., $Loc \mapsto (0, 1] \times Val$)

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{ inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{ exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally	- All logical constraints in A are assumed - All resources required by A are obtained	

Acting on a separation
logic state
(e.g., $\text{Loc} \mapsto (0, 1] \times \text{Val}$)

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{ inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{ exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally	• All logical constraints in A are assumed • All resources required by A are obtained	• All logical constraints in A are asserted • All resources required by A are removed

Acting on a separation
logic state
(e.g., $Loc \mapsto (0, 1] \times Val$)

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally	• All logical constraints in A are assumed • All resources required by A are obtained	• All logical constraints in A are asserted • All resources required by A are removed
Separation logic analogue of	assume A	assert A

Verification Primitives: Inhale and Exhale

Sometimes called
produce

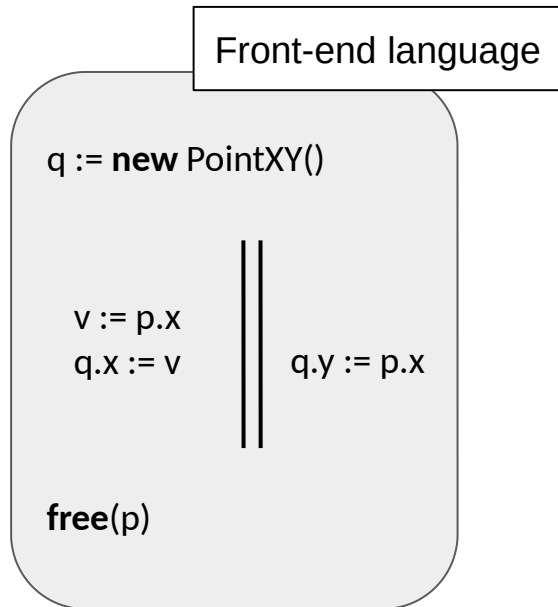
inhale A

Sometimes called
consume

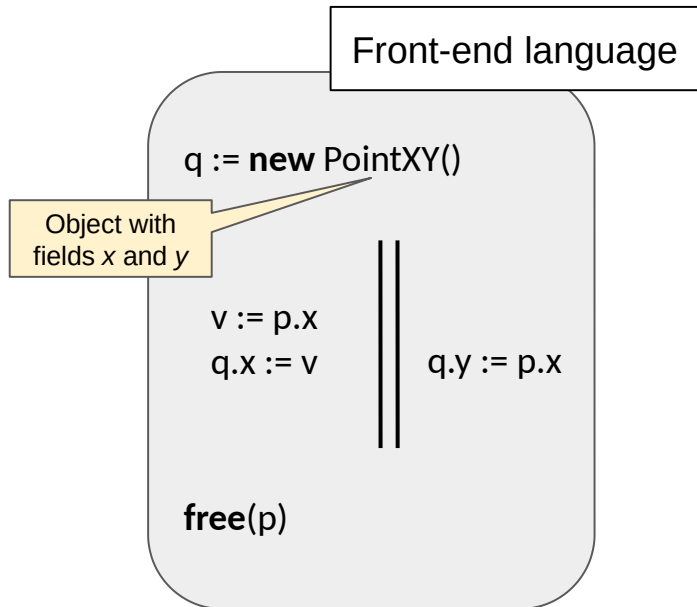
exhale A

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A - * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally	- All logical constraints in A are assumed - All resources required by A are obtained	- All logical constraints in A are asserted - All resources required by A are removed
Separation logic analogue of	assume A	assert A

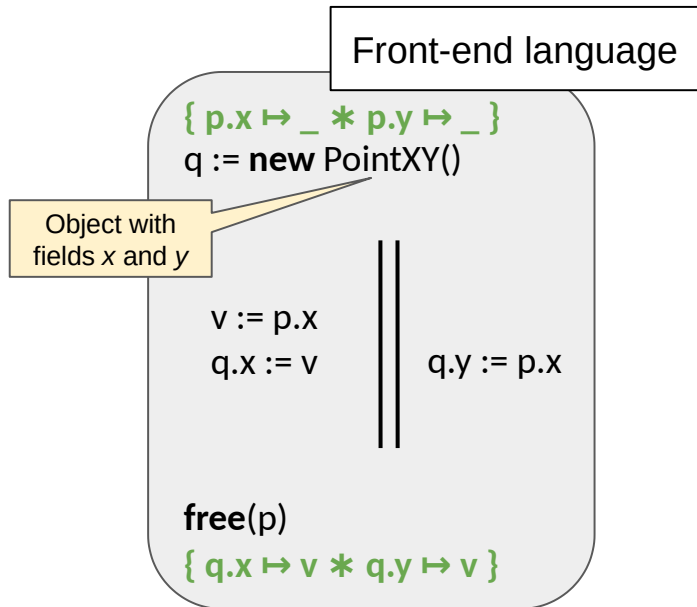
Example: Verifying a Parallel Program (1/2)



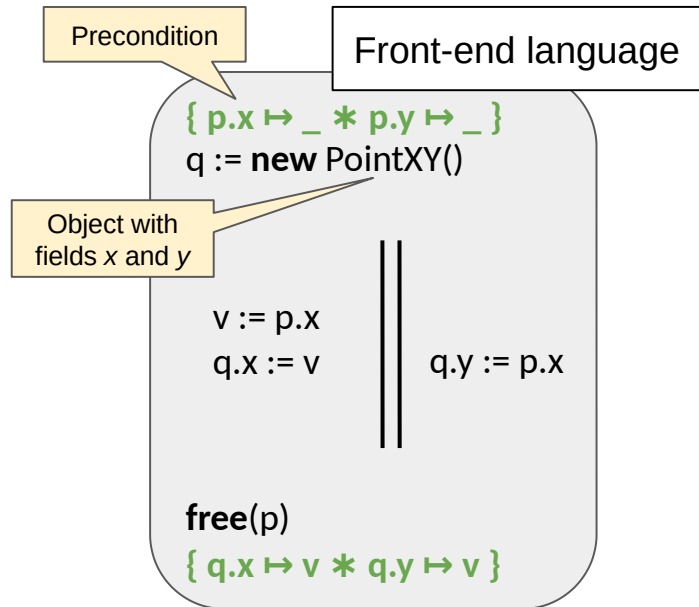
Example: Verifying a Parallel Program (1/2)



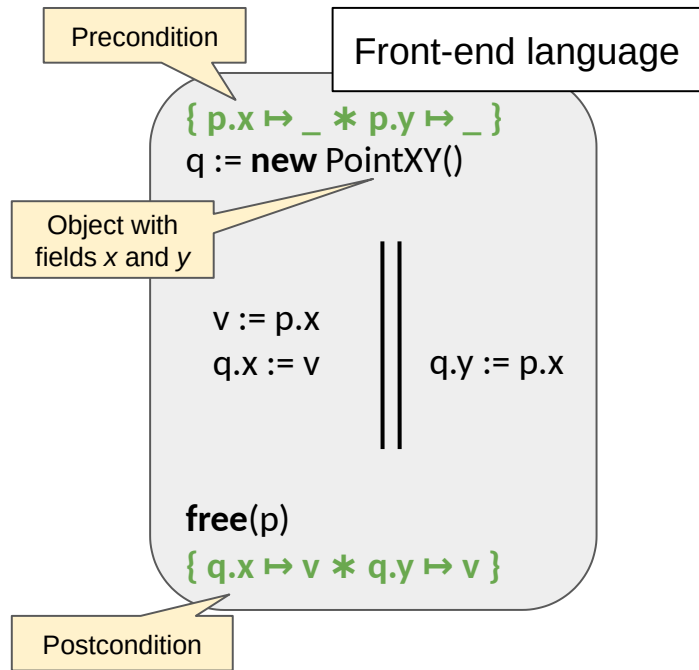
Example: Verifying a Parallel Program (1/2)



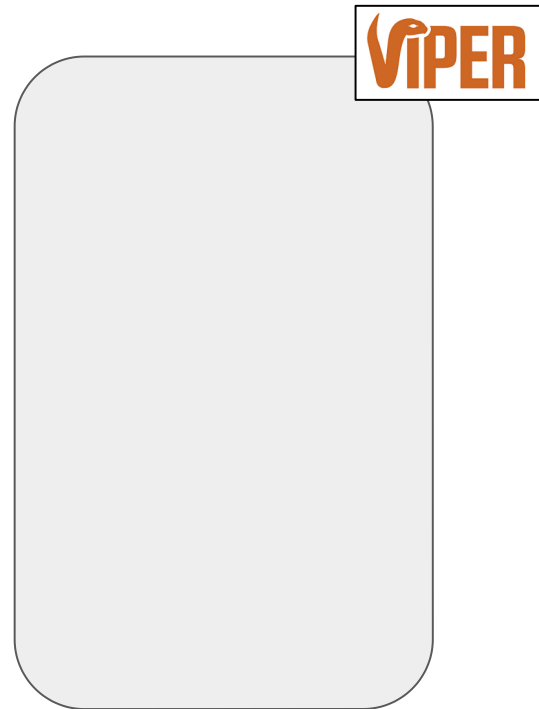
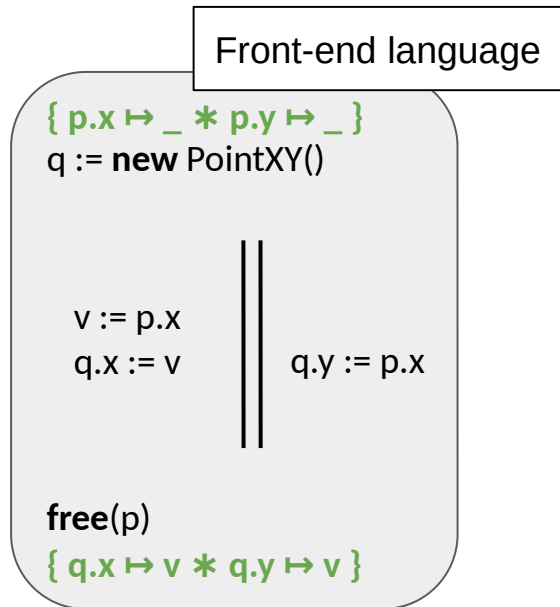
Example: Verifying a Parallel Program (1/2)



Example: Verifying a Parallel Program (1/2)



Example: Verifying a Parallel Program (1/2)



Example: Verifying a Parallel Program (1/2)

Front-end language

$\{ p.x \mapsto _ * p.y \mapsto _ \}$

$q := \text{new PointXY}()$

$v := p.x$

$q.x := v$

$q.y := p.x$

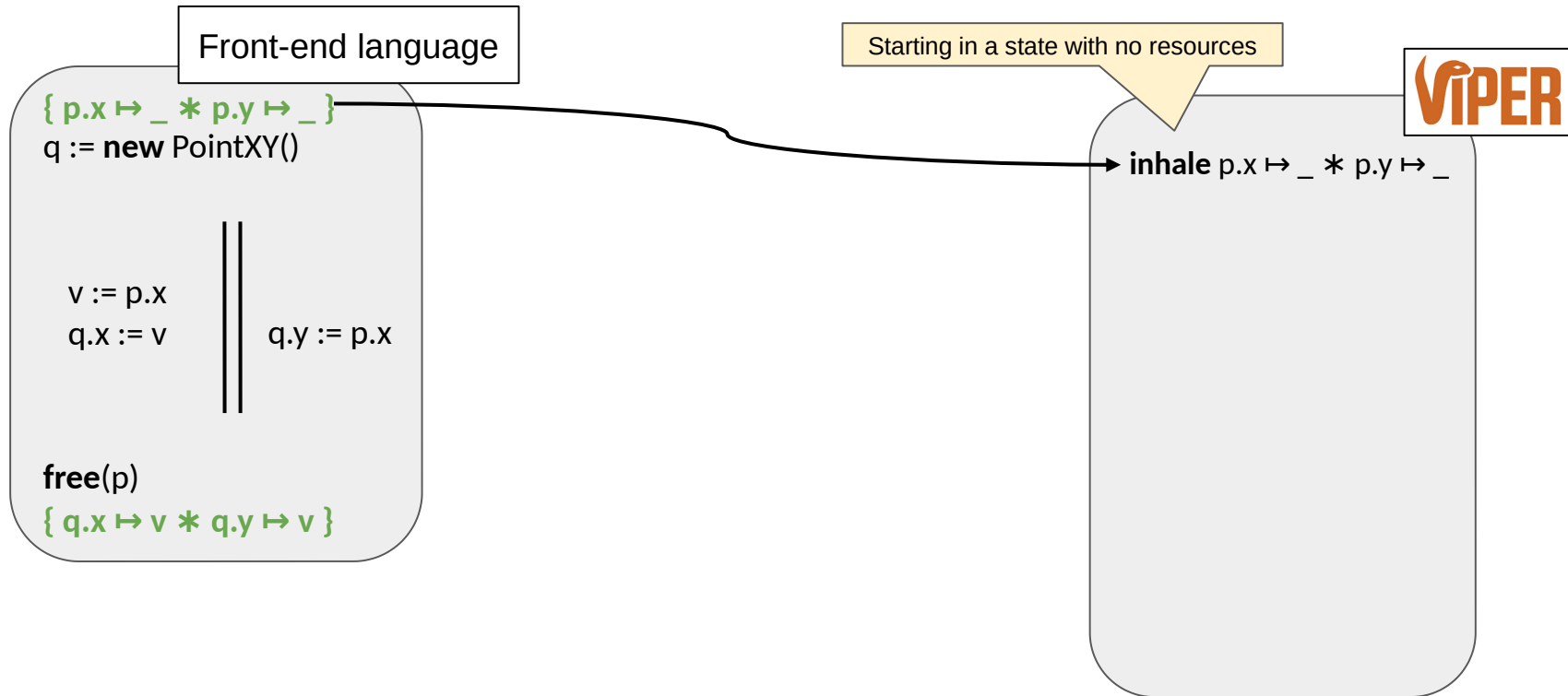
$\text{free}(p)$

$\{ q.x \mapsto v * q.y \mapsto v \}$

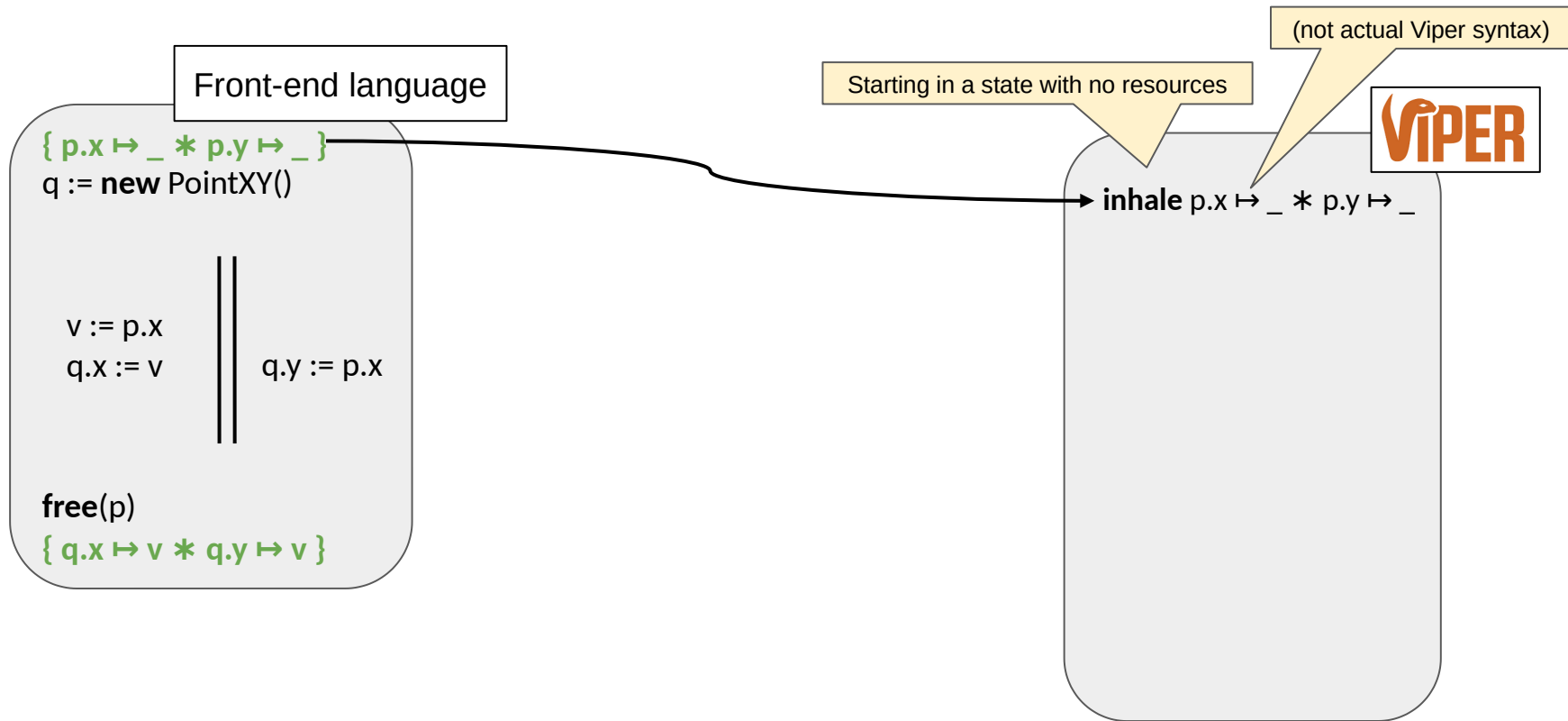
Starting in a state with no resources

VIPER

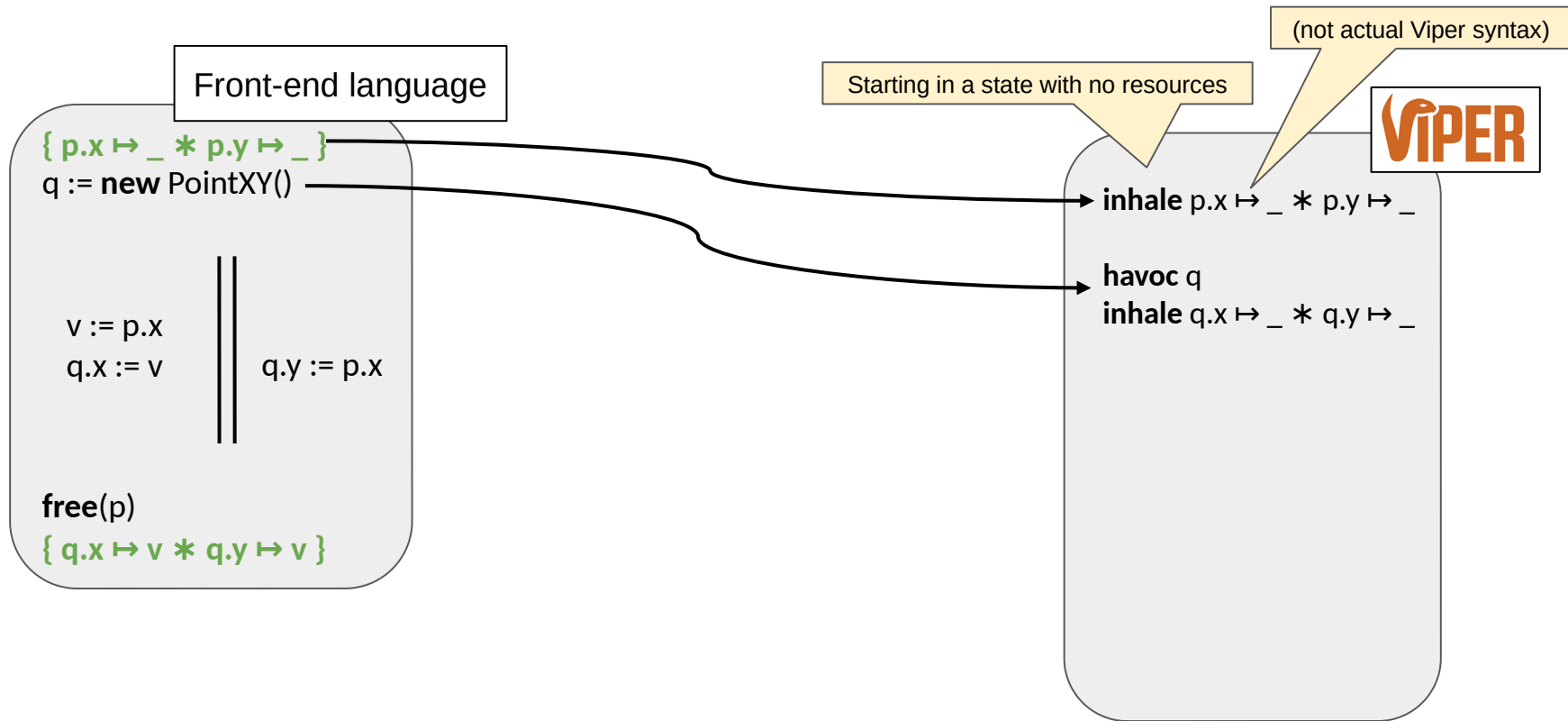
Example: Verifying a Parallel Program (1/2)



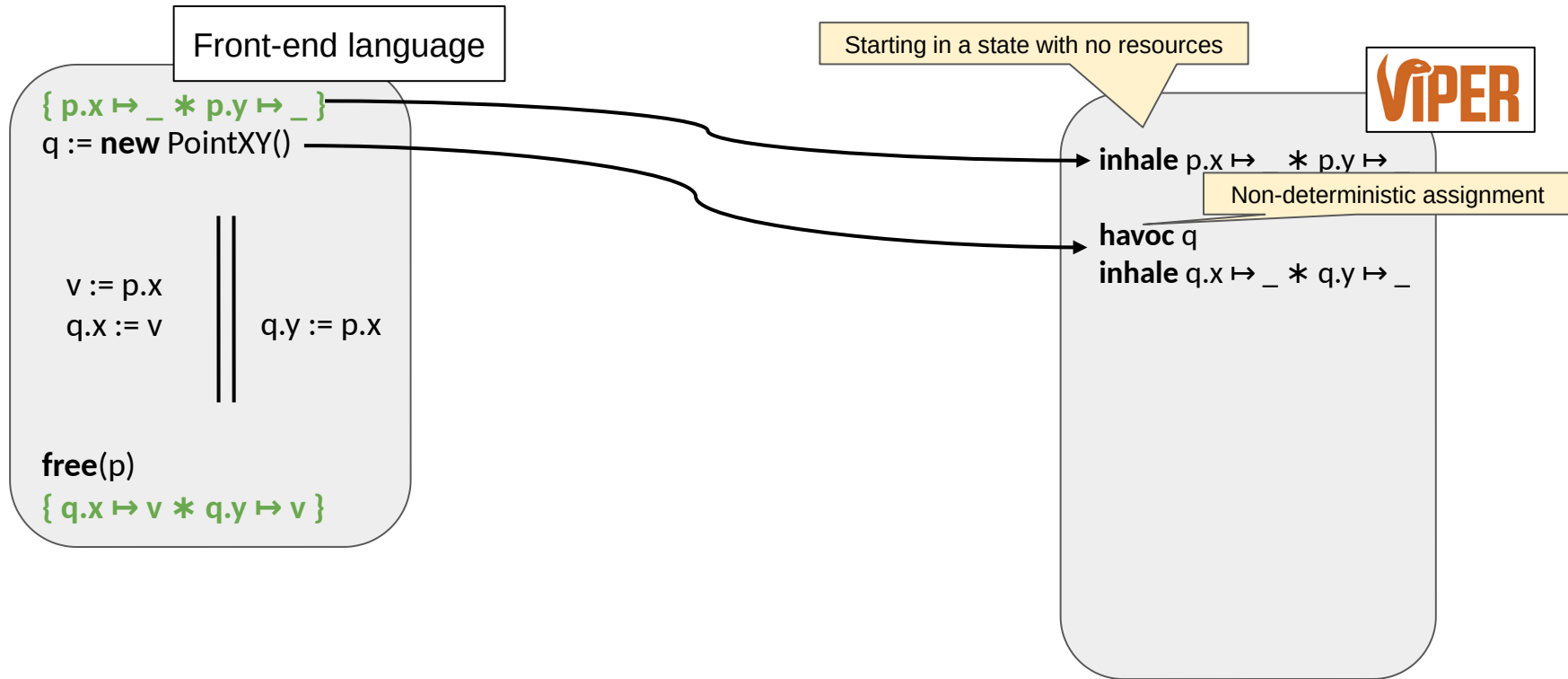
Example: Verifying a Parallel Program (1/2)



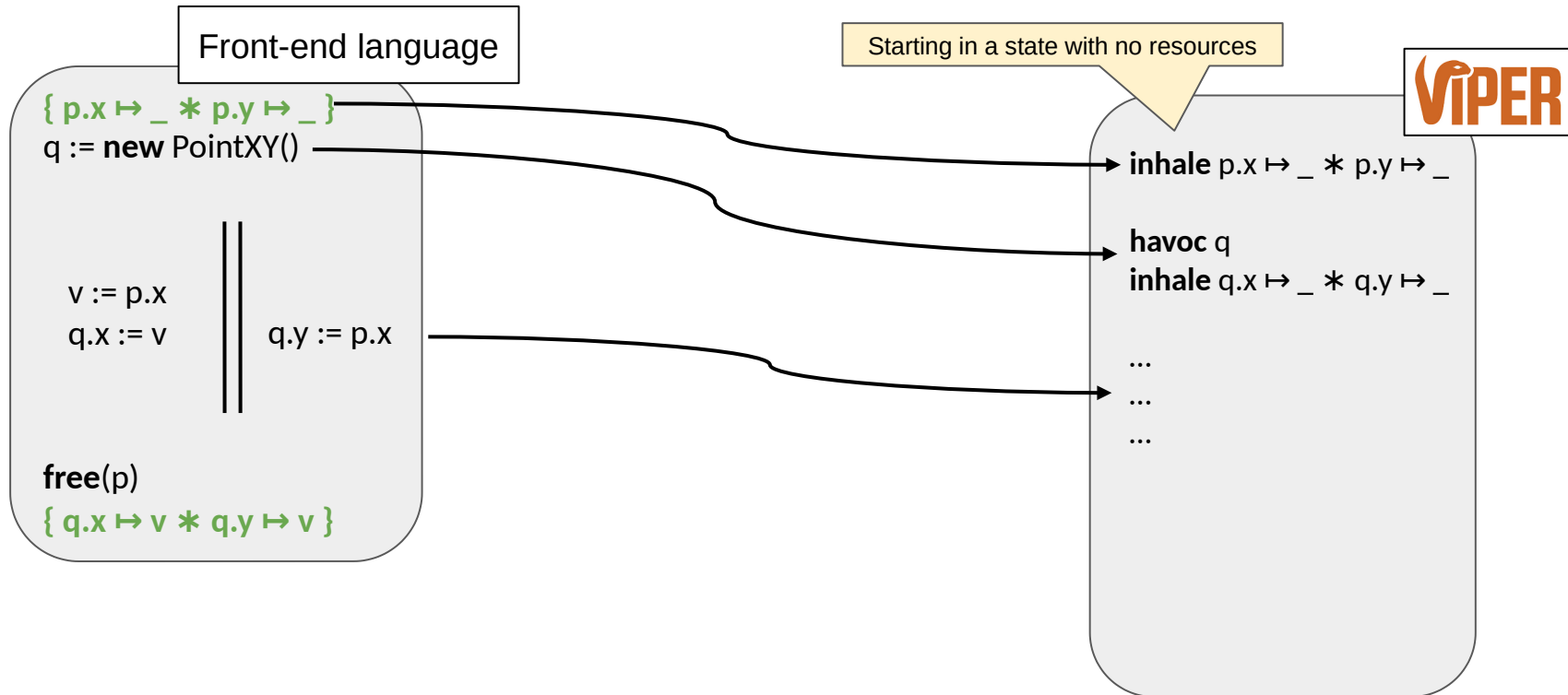
Example: Verifying a Parallel Program (1/2)



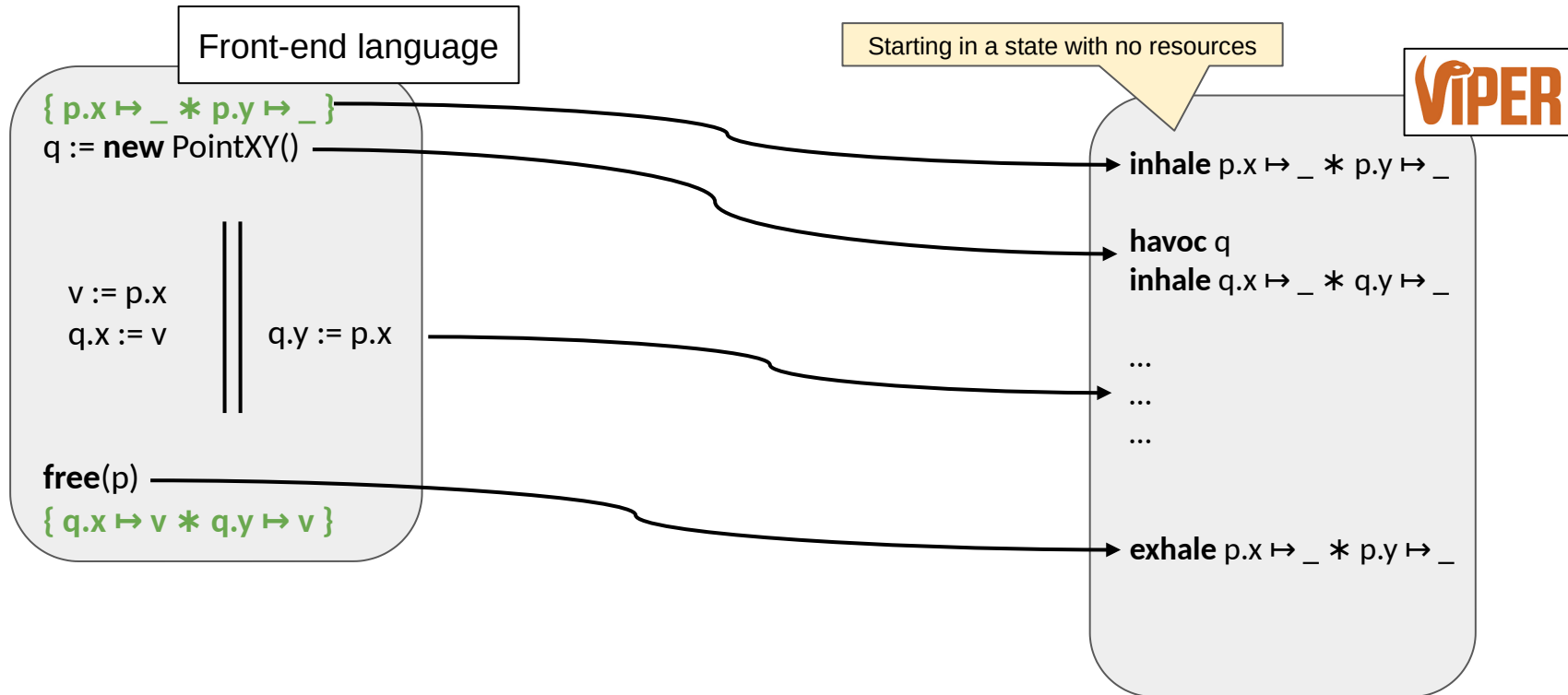
Example: Verifying a Parallel Program (1/2)



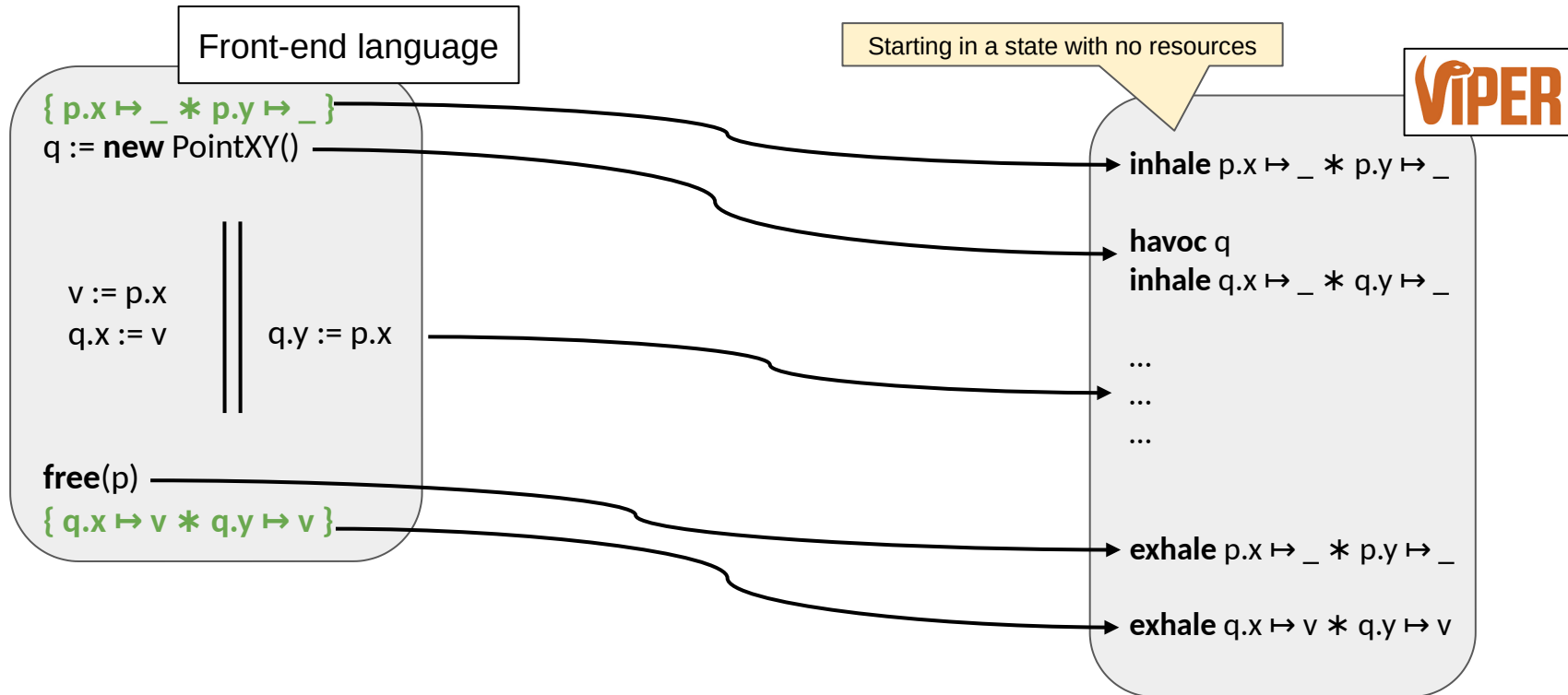
Example: Verifying a Parallel Program (1/2)



Example: Verifying a Parallel Program (1/2)



Example: Verifying a Parallel Program (1/2)



Example: Verifying a Parallel Program (2/2)

Front-end language

```
{ p.x ↦ _ * p.y ↦ _ }
```

```
q := new PointXY()
```

```
v := p.x
```

```
q.x := v
```

```
q.y := p.x
```

```
free(p)
```

```
{ q.x ↦ v * q.y ↦ v }
```

VIPER

```
inhale p.x ↦ _ * p.y ↦ _
```

```
havoc q
```

```
inhale q.x ↦ _ * q.y ↦ _
```

```
...
```

```
...
```

```
...
```

```
exhale p.x ↦ _ * p.y ↦ _
```

```
exhale q.x ↦ v * q.y ↦ v
```

Example: Verifying a Parallel Program (2/2)

Front-end language

```
{ p.x ↦ _ * p.y ↦ _ }
```

```
q := new PointXY()
```

```
v := p.x
```

```
q.x := v
```

```
q.y := p.x
```

```
free(p)
```

```
{ q.x ↦ v * q.y ↦ v }
```

VIPER

```
inhale p.x ↦ _ * p.y ↦ _
```

```
havoc q
```

```
inhale q.x ↦ _ * q.y ↦ _
```

```
...
```

```
...
```

```
...
```

```
exhale p.x ↦ _ * p.y ↦ _
```

```
exhale q.x ↦ v * q.y ↦ v
```

Example: Verifying a Parallel Program (2/2)

Parallel rule

$$\frac{\vdash \{P_1\} C_1 \{Q_1\} \quad \vdash \{P_2\} C_2 \{Q_2\} \quad \dots}{\vdash \{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}}$$

Front-end language

$\{ p.x \mapsto _ * p.y \mapsto _ \}$

$q := \text{new PointXY}()$

$v := p.x$

$q.x := v$

$q.y := p.x$

$\text{free}(p)$

$\{ q.x \mapsto v * q.y \mapsto v \}$



inhale $p.x \mapsto _ * p.y \mapsto _$

havoc q

inhale $q.x \mapsto _ * q.y \mapsto _$

...

...

...

exhale $p.x \mapsto _ * p.y \mapsto _$

exhale $q.x \mapsto v * q.y \mapsto v$

Example: Verifying a Parallel Program (2/2)

Parallel rule

$$\frac{\vdash \{P_1\} C_1 \{Q_1\} \quad \vdash \{P_2\} C_2 \{Q_2\} \quad \dots}{\vdash \{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}}$$

Front-end language

```
{ p.x ↦ _ * p.y ↦ _ }  
q := new PointXY()
```

$\{P_1\}$		$\{P_2\}$
v := p.x		
q.x := v		q.y := p.x
$\{Q_1\}$		$\{Q_2\}$

```
free(p)  
{ q.x ↦ v * q.y ↦ v }
```



```
inhale p.x ↦ _ * p.y ↦ _
```

```
havoc q  
inhale q.x ↦ _ * q.y ↦ _
```

```
...  
...  
...
```

```
exhale p.x ↦ _ * p.y ↦ _
```

```
exhale q.x ↦ v * q.y ↦ v
```

Example: Verifying a Parallel Program (2/2)

Parallel rule

$$\frac{\vdash \{P_1\} C_1 \{Q_1\} \quad \vdash \{P_2\} C_2 \{Q_2\} \quad \dots}{\vdash \{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}}$$

Front-end language

$\{ p.x \mapsto _ * p.y \mapsto _ \}$

$q := \text{new PointXY}()$

$\{ P_1 \}$

$v := p.x$

$q.x := v$

$\{ Q_1 \}$

$\{ P_2 \}$

$q.y := p.x$

$\{ Q_2 \}$

$\text{free}(p)$

$\{ q.x \mapsto v * q.y \mapsto v \}$

VIPER

inhale $p.x \mapsto _ * p.y \mapsto _$

havoc q

inhale $q.x \mapsto _ * q.y \mapsto _$

...

...

...

exhale $p.x \mapsto _ * p.y \mapsto _$

exhale $q.x \mapsto v * q.y \mapsto v$

$$P_1 \triangleq (q.x \mapsto _ * p.x \xrightarrow{1/2} _) \quad P_2 \triangleq (q.y \mapsto _ * p.x \xrightarrow{1/2} _)$$

$$Q_1 \triangleq (q.x \mapsto v * p.x \xrightarrow{1/2} v) \quad Q_2 \triangleq (\exists k. q.y \mapsto k * p.x \xrightarrow{1/2} k)$$

Example: Verifying a Parallel Program (2/2)

Parallel rule

$$\frac{\vdash \{P_1\} C_1 \{Q_1\} \quad \vdash \{P_2\} C_2 \{Q_2\} \quad \dots}{\vdash \{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}}$$

Front-end language

$\{p.x \mapsto _ * p.y \mapsto _ \}$

$q := \text{new PointXY}()$

$\{P_1\}$

$v := p.x$

$q.x := v$

$\{Q_1\}$

$\{P_2\}$

$q.y := p.x$

$\{Q_2\}$

$\text{free}(p)$

$\{q.x \mapsto v * q.y \mapsto v\}$

VIPER

inhale $p.x \mapsto _ * p.y \mapsto _$

havoc q

inhale $q.x \mapsto _ * q.y \mapsto _$

...

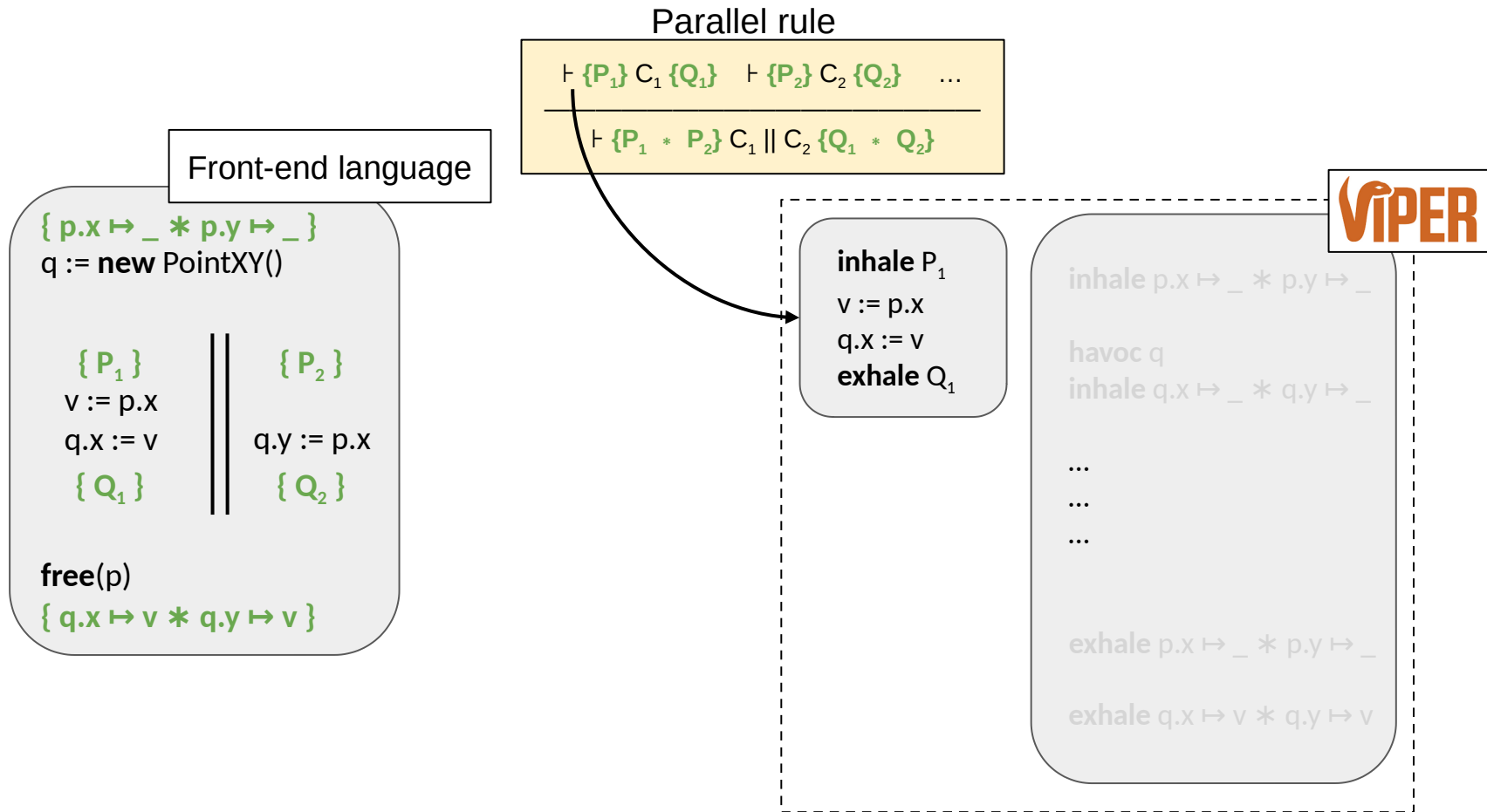
...

...

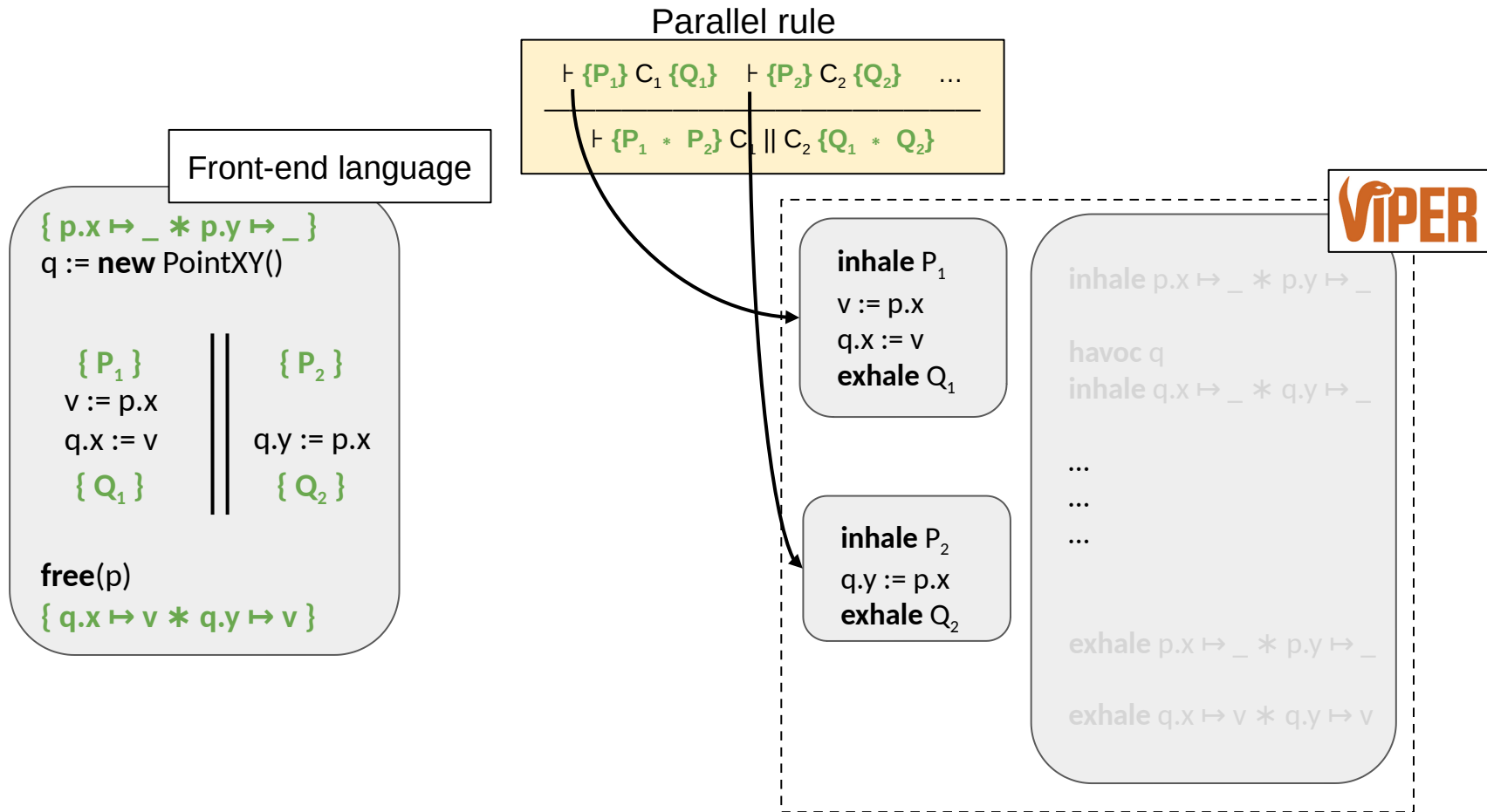
exhale $p.x \mapsto _ * p.y \mapsto _$

exhale $q.x \mapsto v * q.y \mapsto v$

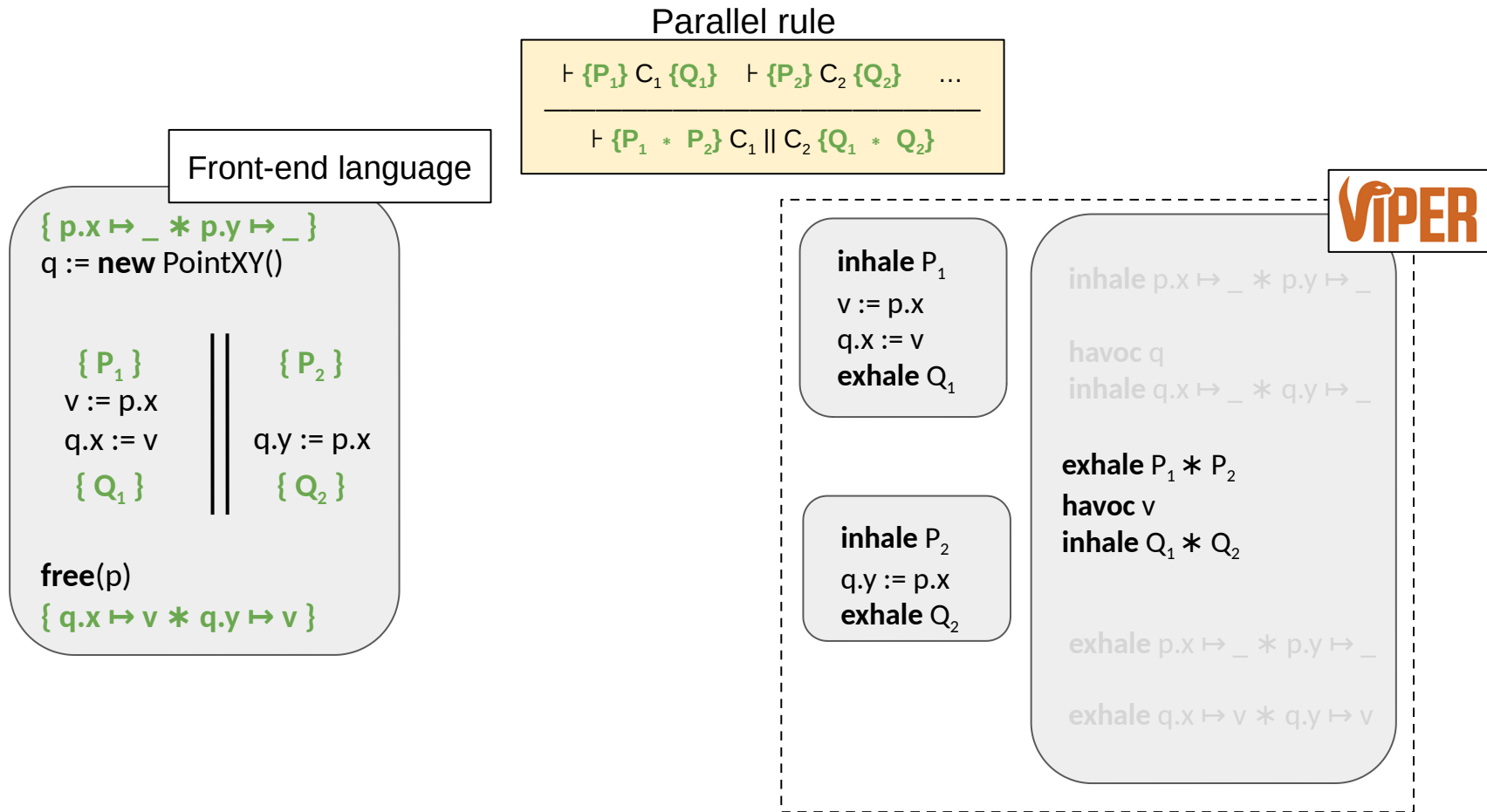
Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)

Parallel rule

$$\frac{\vdash \{P_1\} C_1 \{Q_1\} \quad \vdash \{P_2\} C_2 \{Q_2\} \quad \dots}{\vdash \{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}}$$

Front-end language

$\{p.x \mapsto _ * p.y \mapsto _ \}$

$q := \text{new PointXY}()$

$\{P_1\}$

$v := p.x$

$q.x := v$

$\{Q_1\}$

$\{P_2\}$

$q.y := p.x$

$\{Q_2\}$

$\text{free}(p)$

$\{q.x \mapsto v * q.y \mapsto v\}$

inhale P_1

$v := p.x$

$q.x := v$

exhale Q_1

inhale P_2

$q.y := p.x$

exhale Q_2

inhale $p.x \mapsto _ * p.y \mapsto _$

havoc q

inhale $q.x \mapsto _ * q.y \mapsto _$

exhale $P_1 * P_2$

havoc v

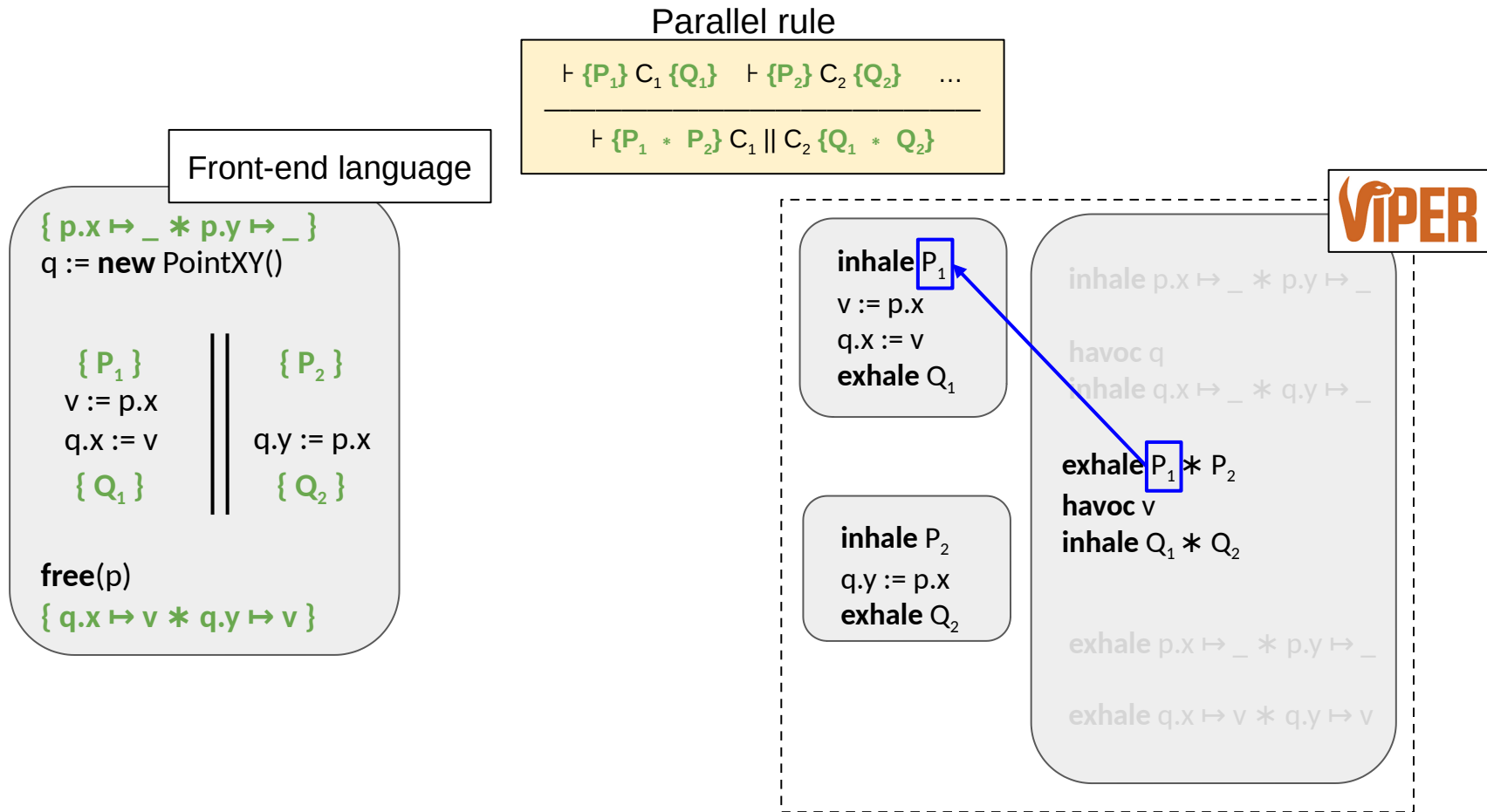
inhale $Q_1 * Q_2$

exhale $p.x \mapsto _ * p.y \mapsto _$

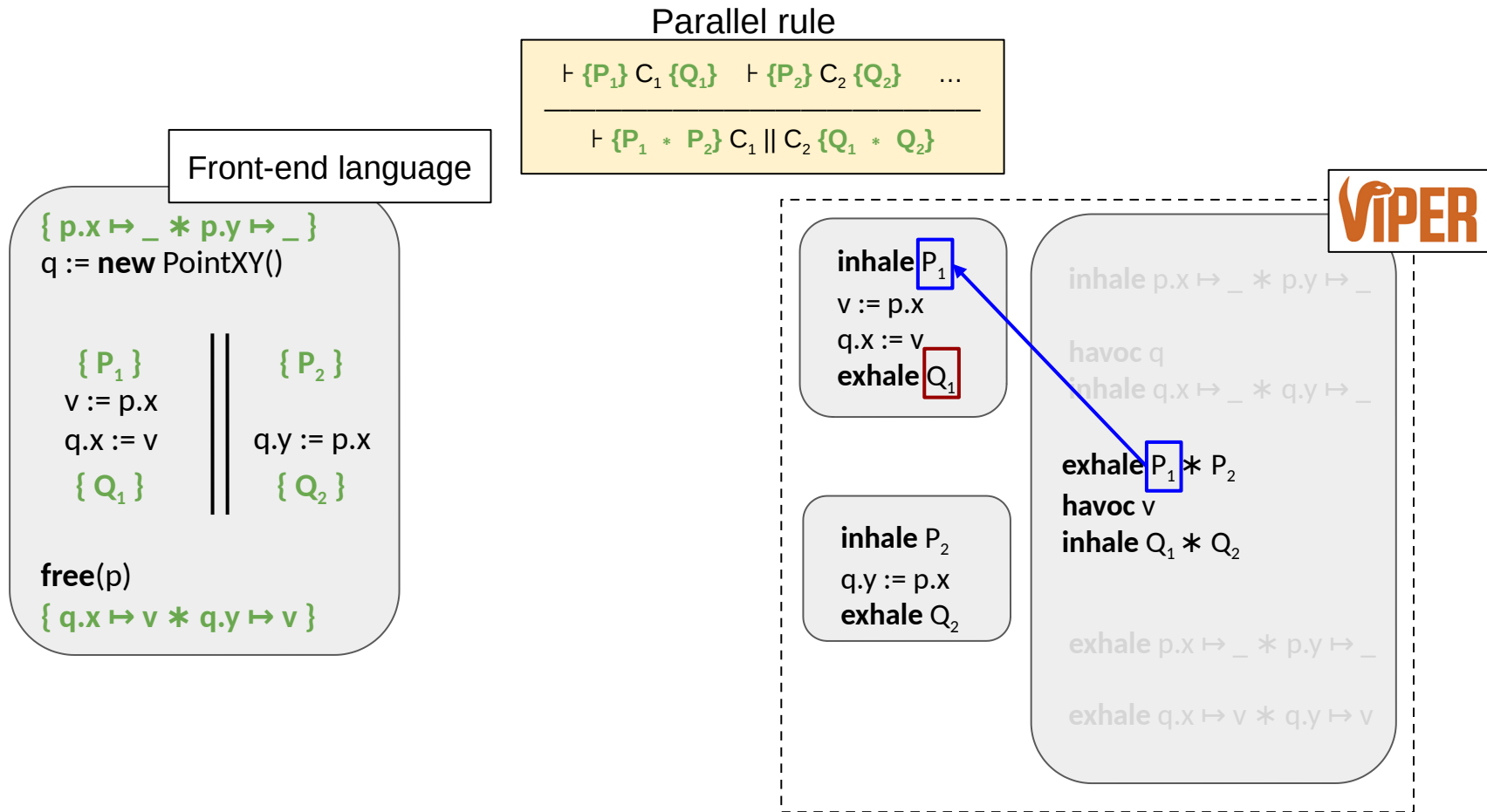
exhale $q.x \mapsto v * q.y \mapsto v$



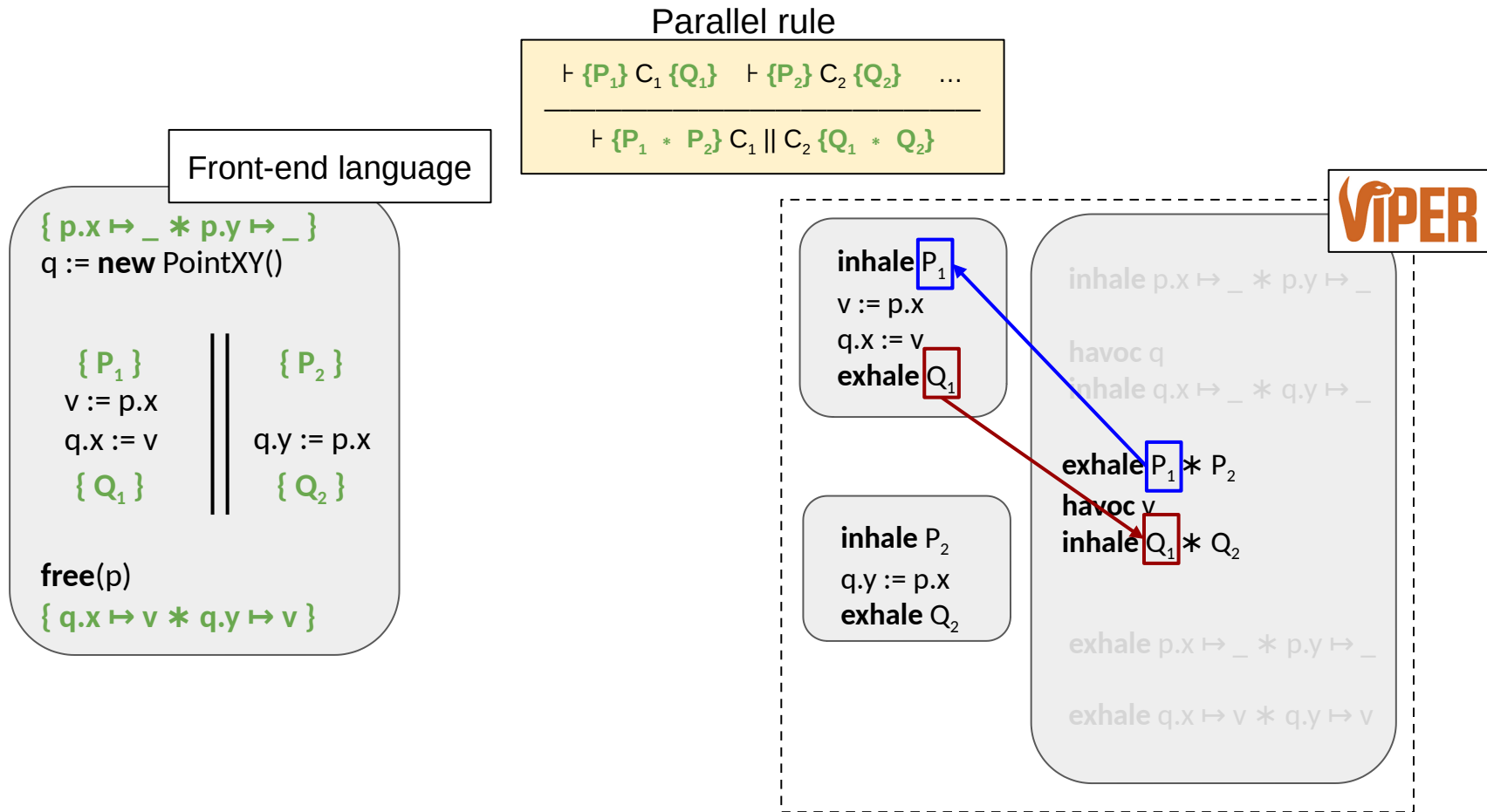
Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)

Parallel rule

$$\frac{\vdash \{P_1\} C_1 \{Q_1\} \quad \vdash \{P_2\} C_2 \{Q_2\} \quad \dots}{\vdash \{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}}$$

Front-end language

$\{p.x \mapsto _ * p.y \mapsto _ \}$

$q := \text{new PointXY}()$

$\{P_1\}$

$v := p.x$

$q.x := v$

$\{Q_1\}$

$\{P_2\}$

$q.y := p.x$

$\{Q_2\}$

$\text{free}(p)$

$\{q.x \mapsto v * q.y \mapsto v\}$

inhale P_1

$v := p.x$

$q.x := v$

exhale Q_1

inhale P_2

$q.y := p.x$

exhale Q_2

inhale $p.x \mapsto _ * p.y \mapsto _$

havoc q

inhale $q.x \mapsto _ * q.y \mapsto _$

exhale $P_1 * P_2$

havoc v

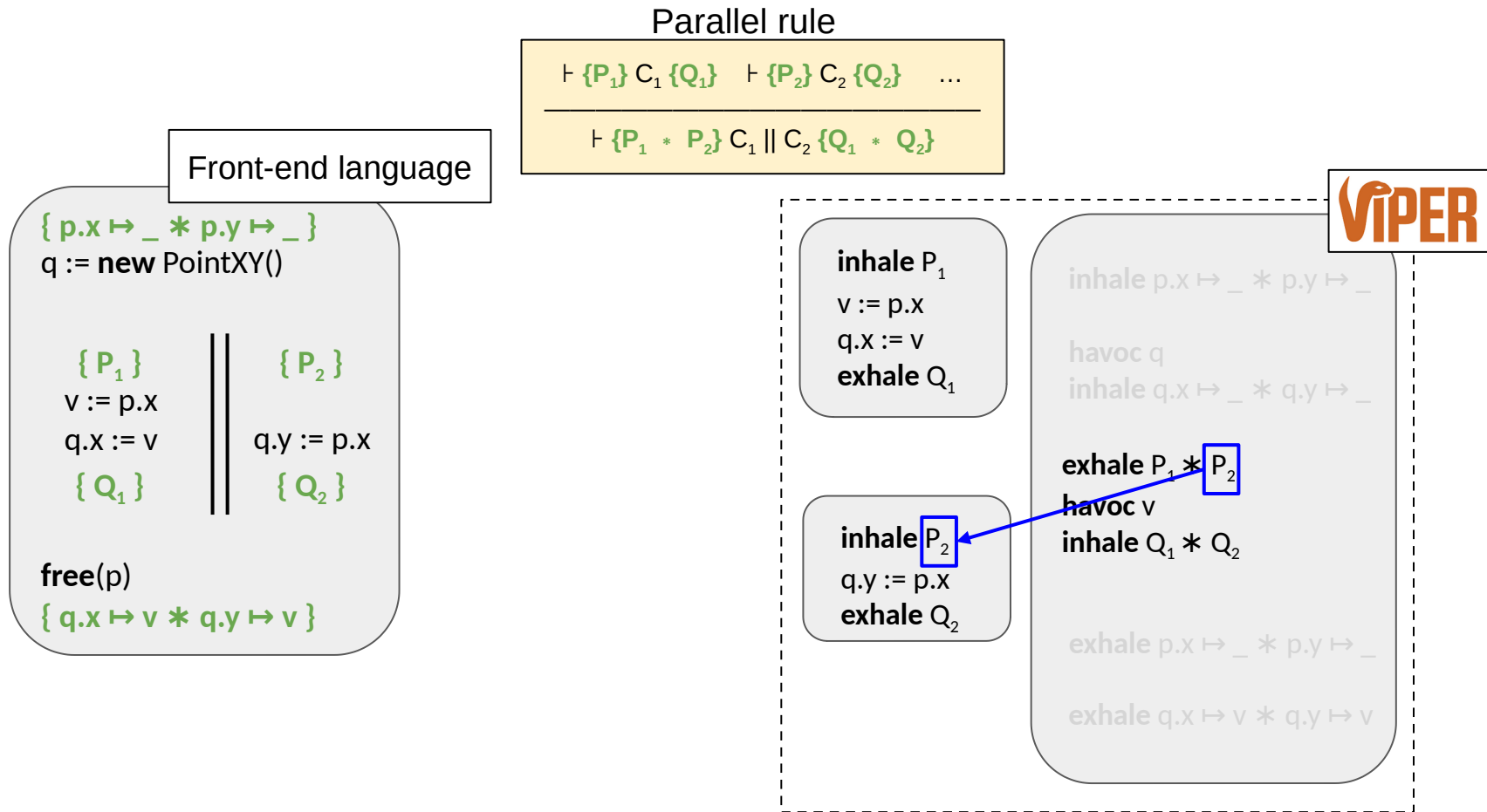
inhale $Q_1 * Q_2$

exhale $p.x \mapsto _ * p.y \mapsto _$

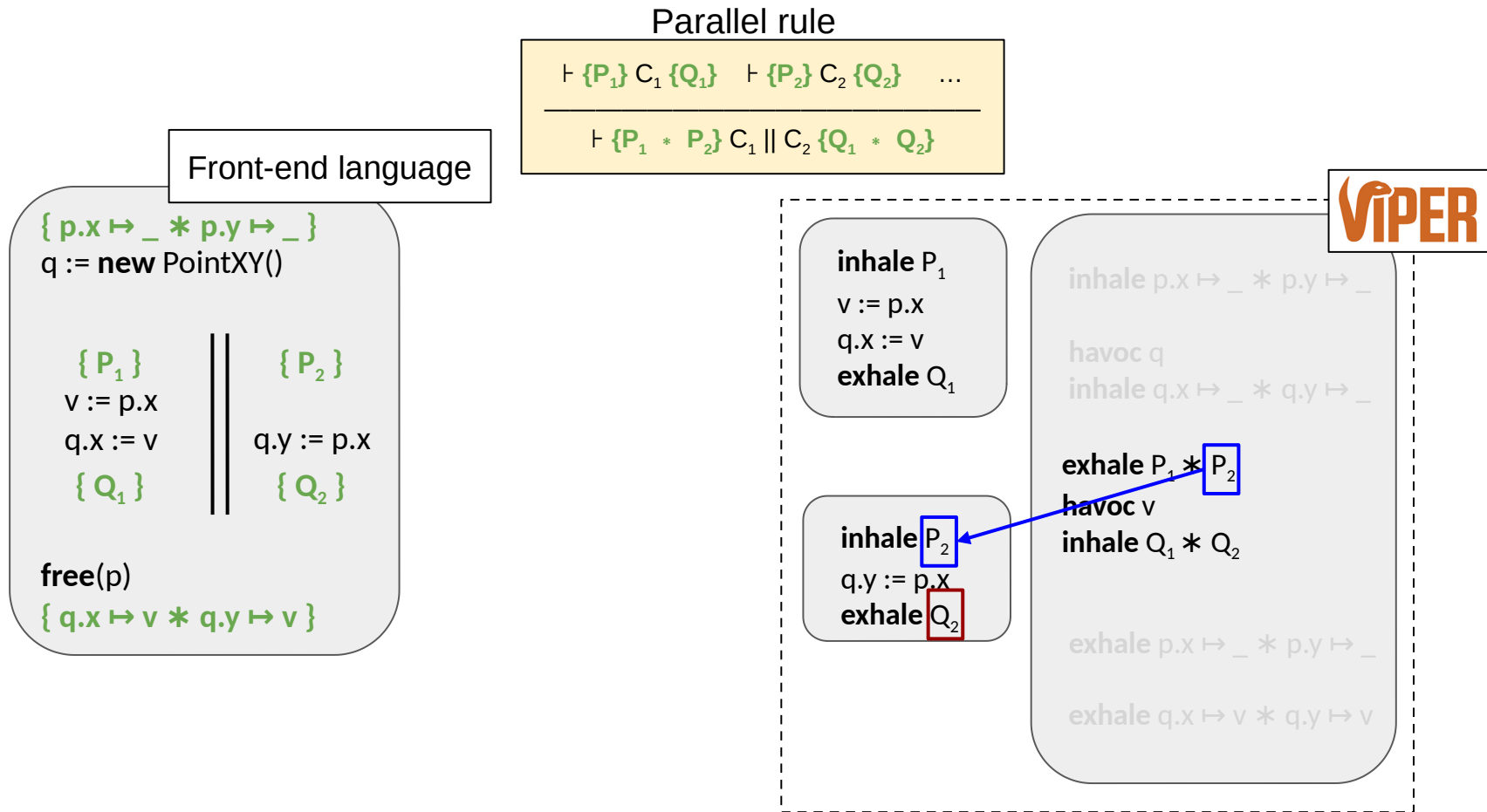
exhale $q.x \mapsto v * q.y \mapsto v$



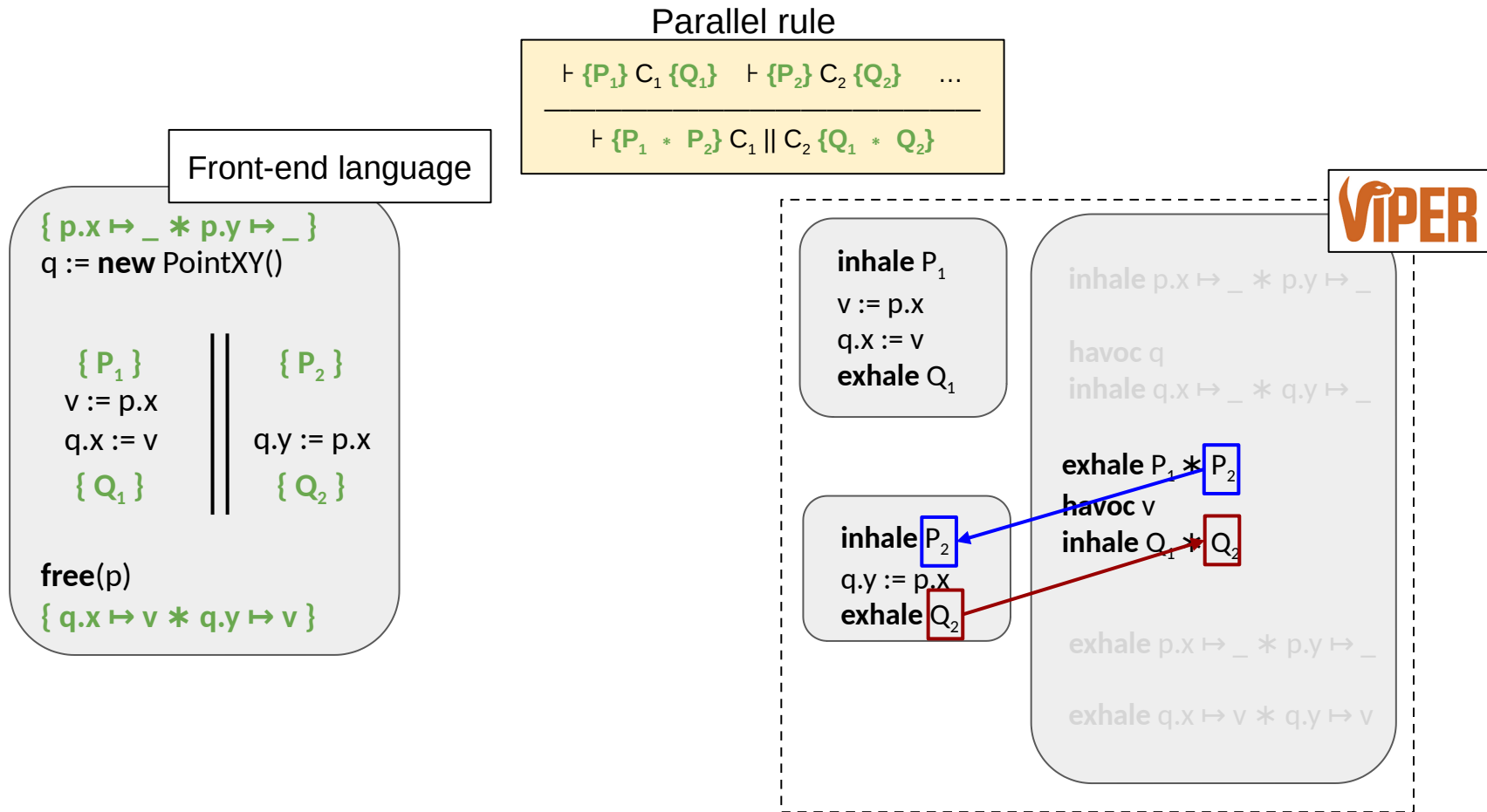
Example: Verifying a Parallel Program (2/2)



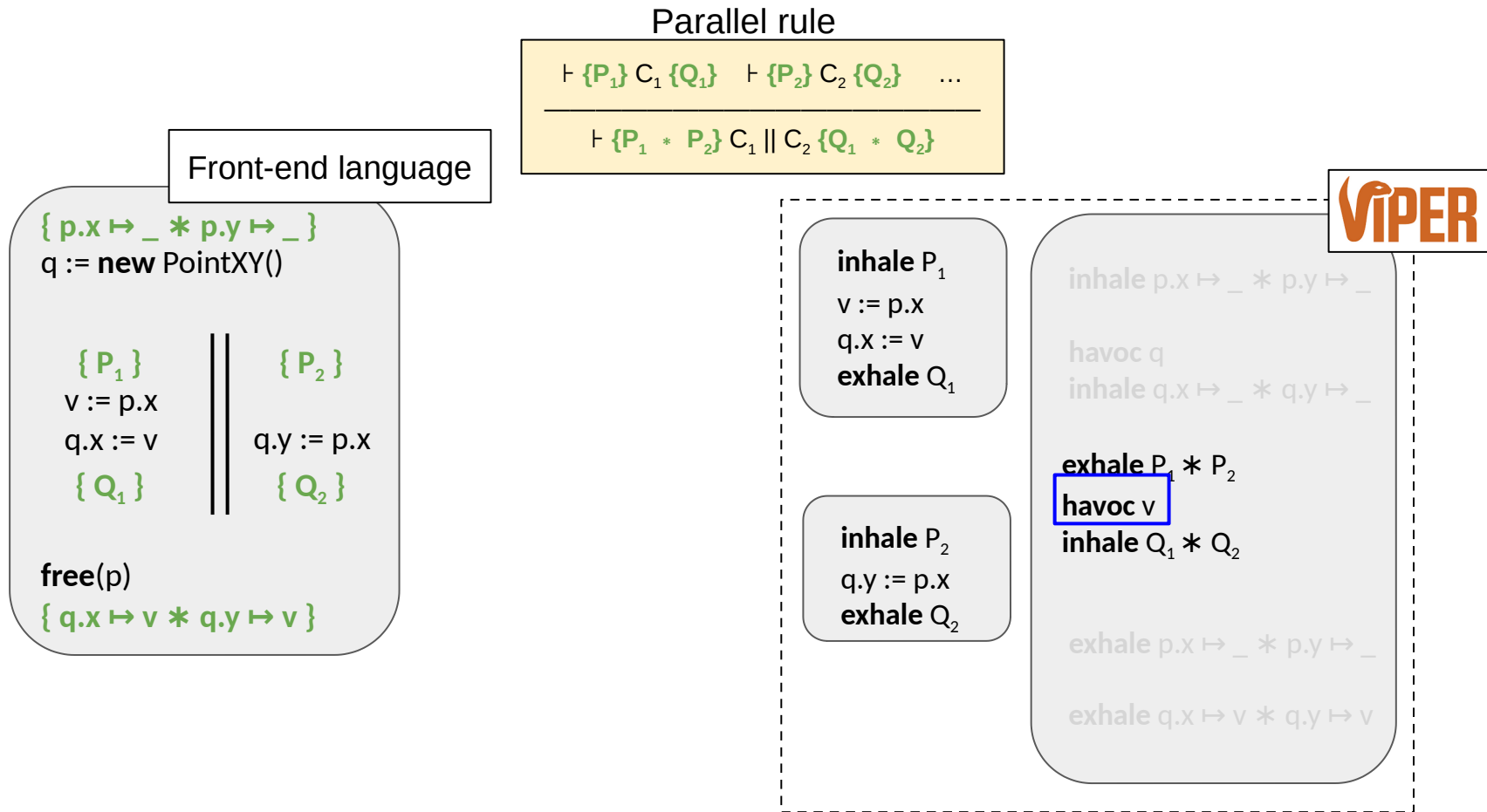
Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)



Example: Verifying a Parallel Program (2/2)

Parallel rule

$$\frac{\vdash \{P_1\} C_1 \{Q_1\} \quad \vdash \{P_2\} C_2 \{Q_2\} \quad \dots}{\vdash \{P_1 * P_2\} C_1 \parallel C_2 \{Q_1 * Q_2\}}$$

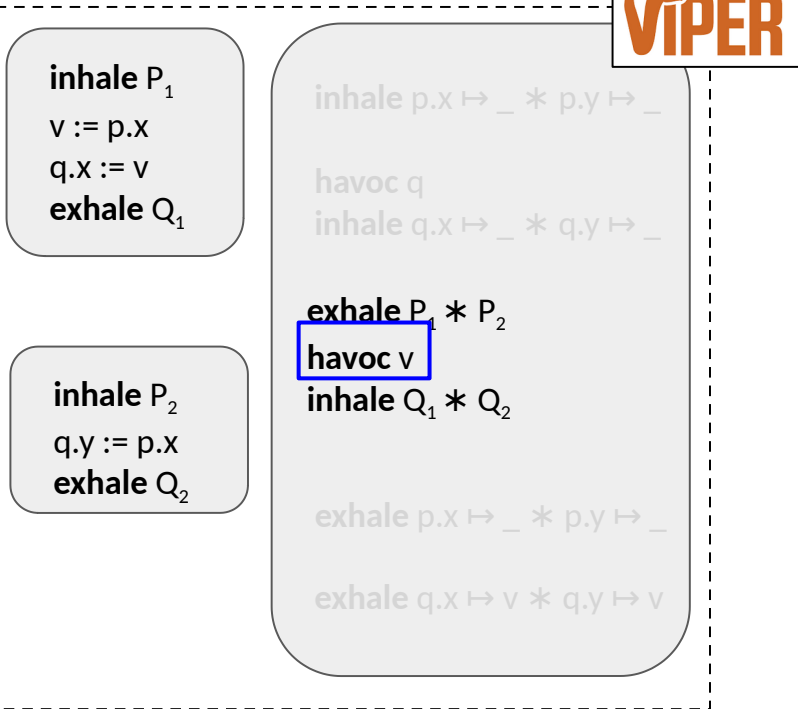
Front-end language

```

{ p.x ↦ _ * p.y ↦ _ }
q := new PointXY()

{ P1 }           { P2 }
v := p.x         q.y := p.x
q.x := v         { Q2 }
{ Q1 }           { Q2 }

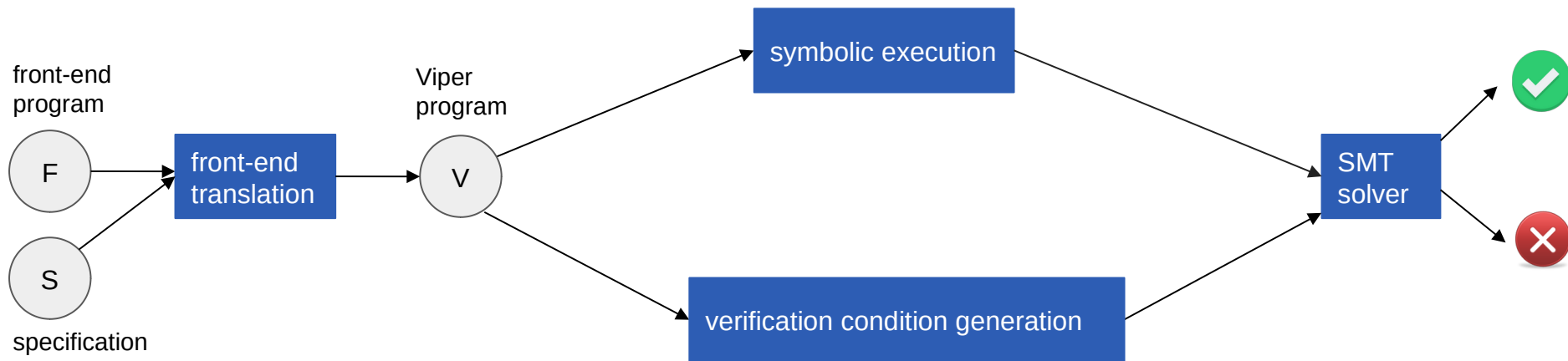
free(p)
{ q.x ↦ v * q.y ↦ v }
    
```



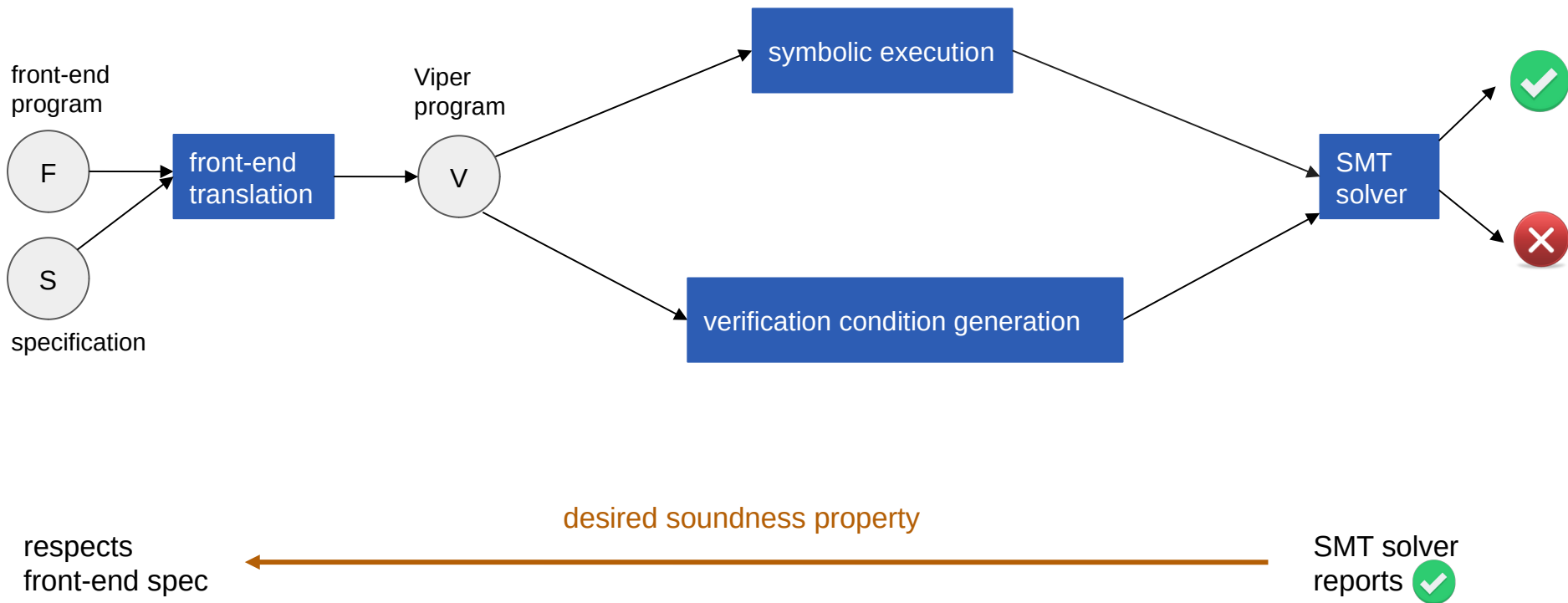
Outline of the Talk

1. Overview of Viper
2. Inhale and Exhale: An Operational View of Separation Logic
- 3. Toward a Foundational Viper**

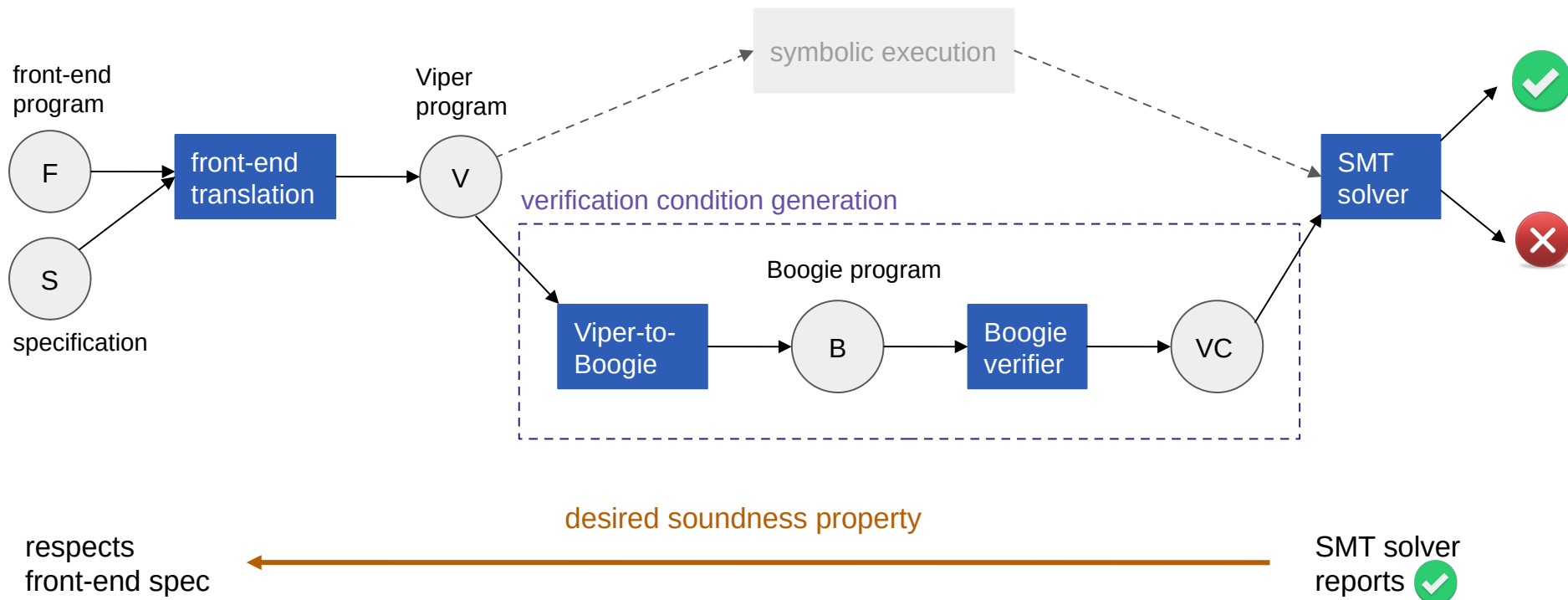
Soundness



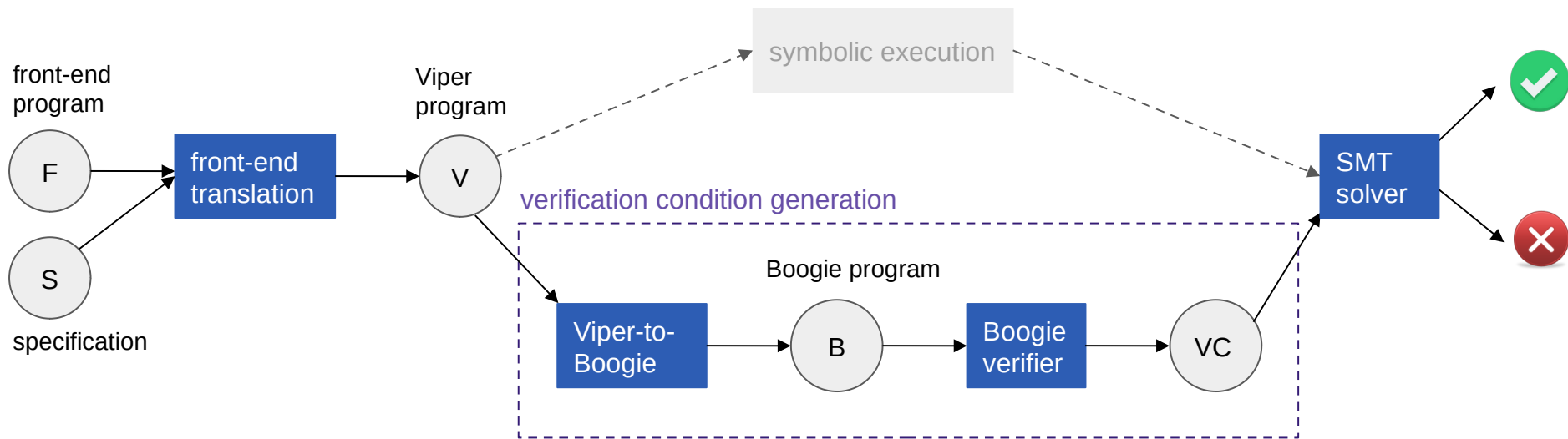
Soundness



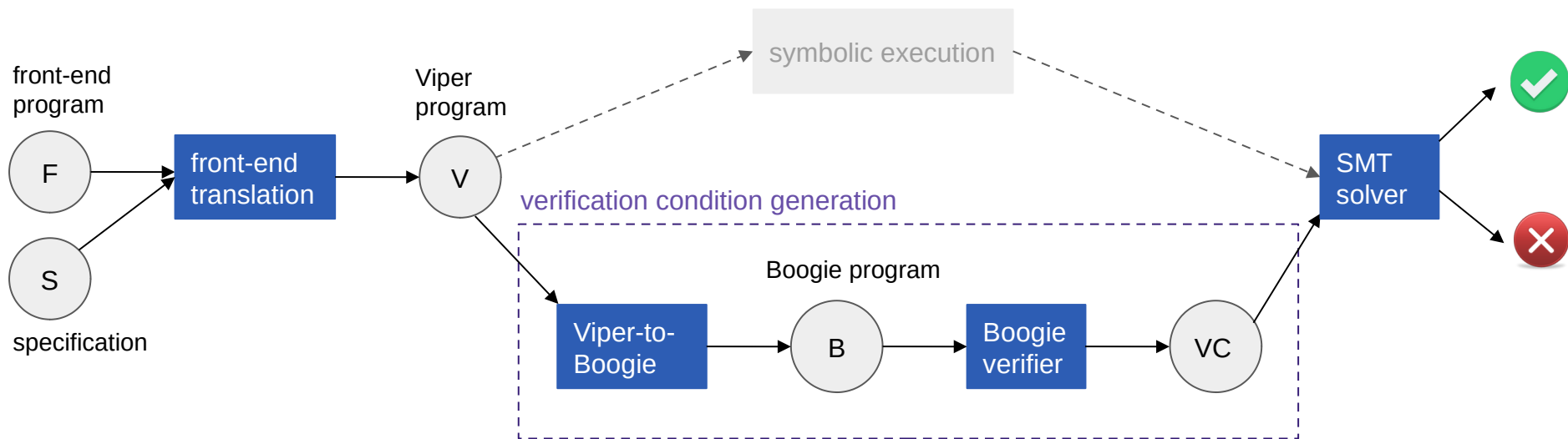
Soundness



Soundness: Proof Strategy

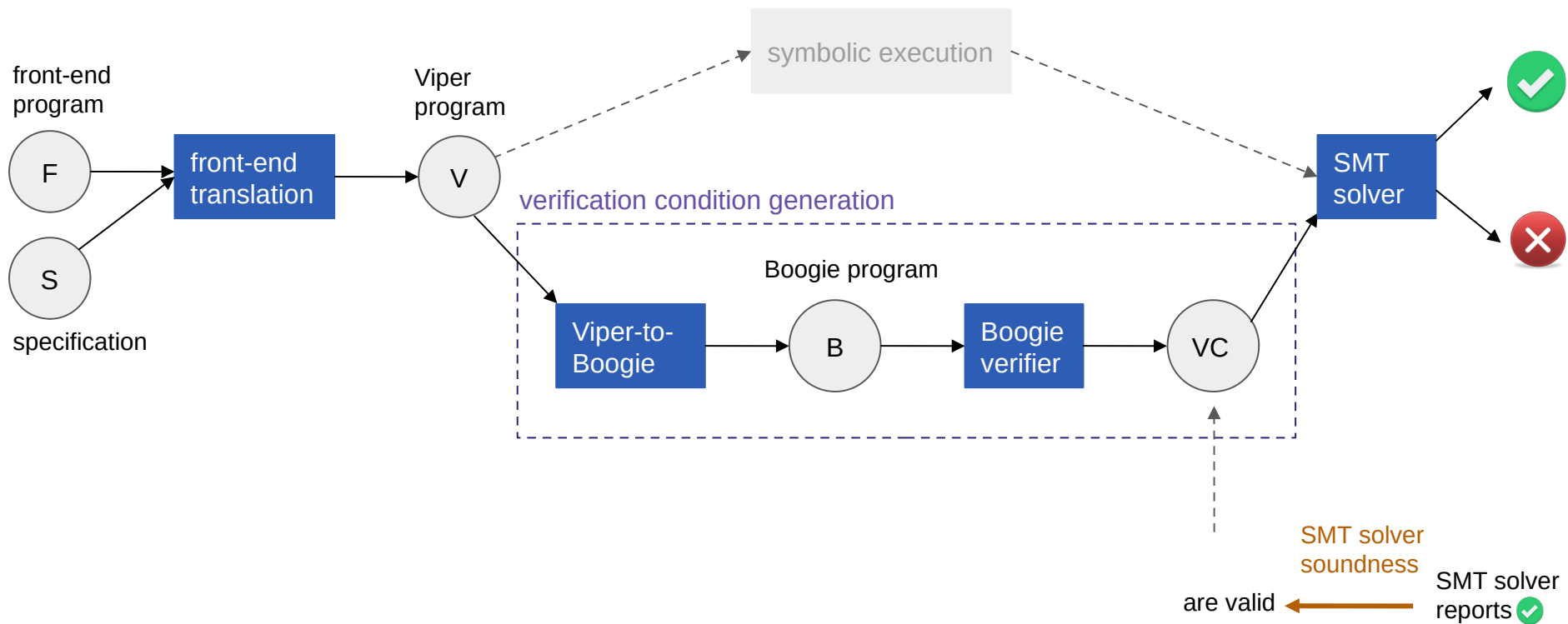


Soundness: Proof Strategy

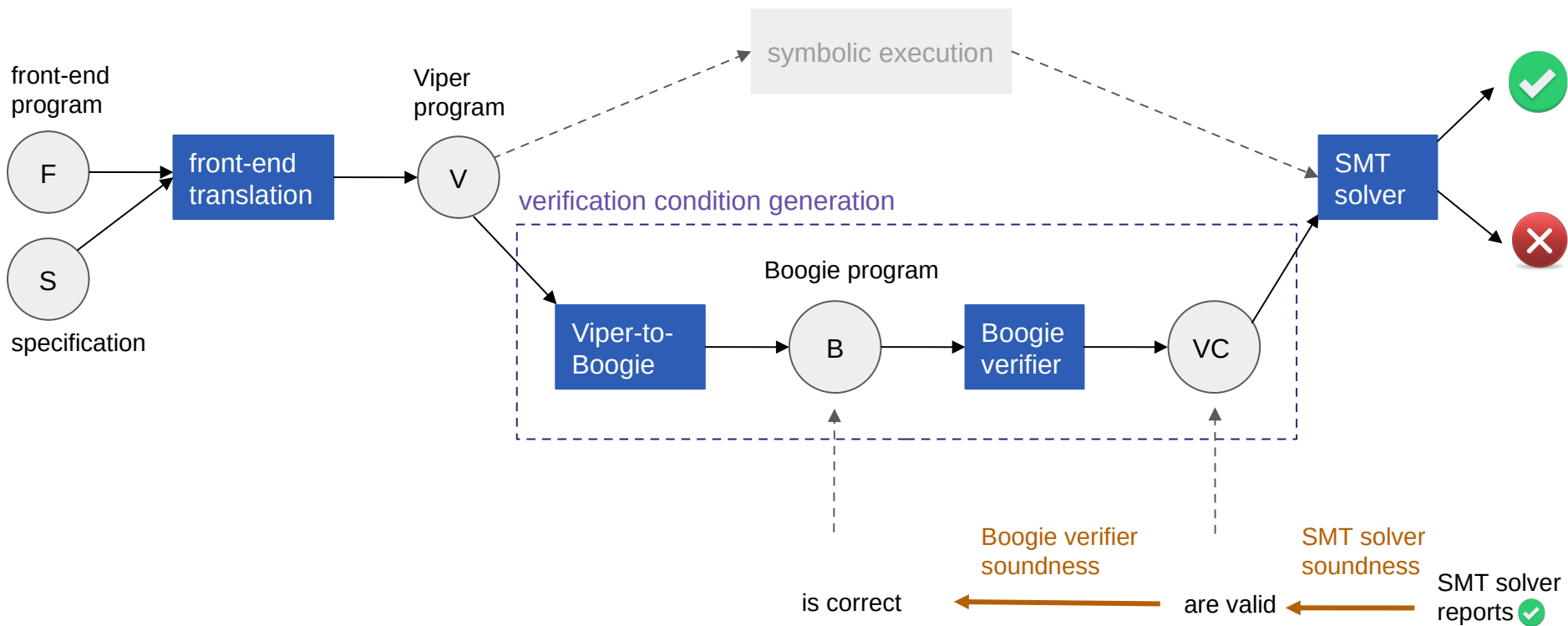


SMT solver reports ✓

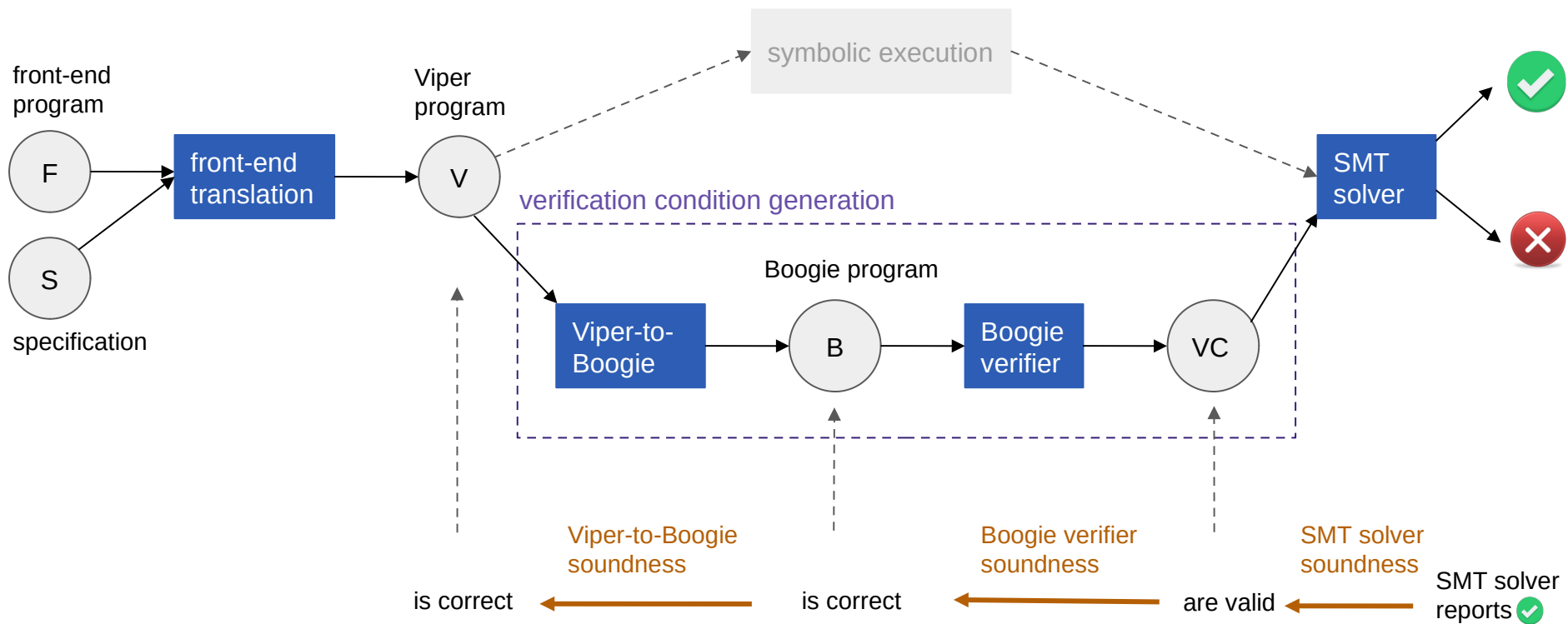
Soundness: Proof Strategy



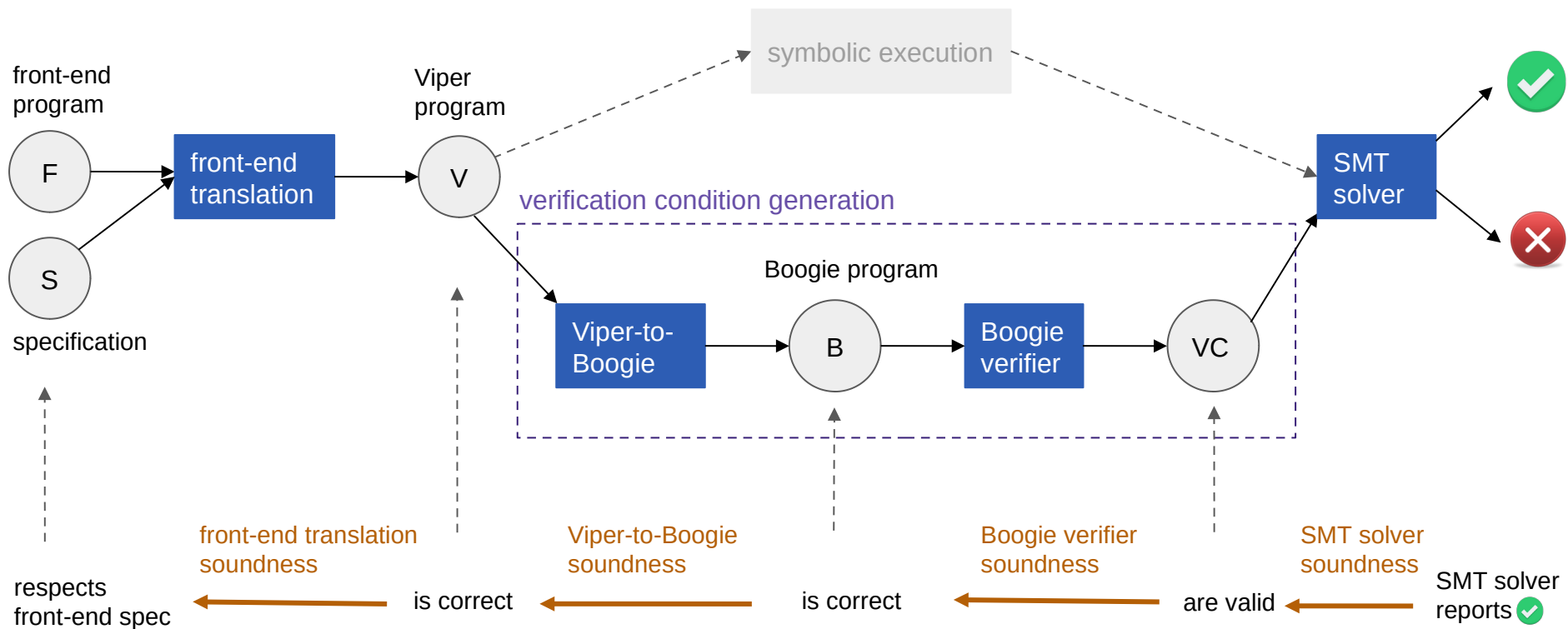
Soundness: Proof Strategy



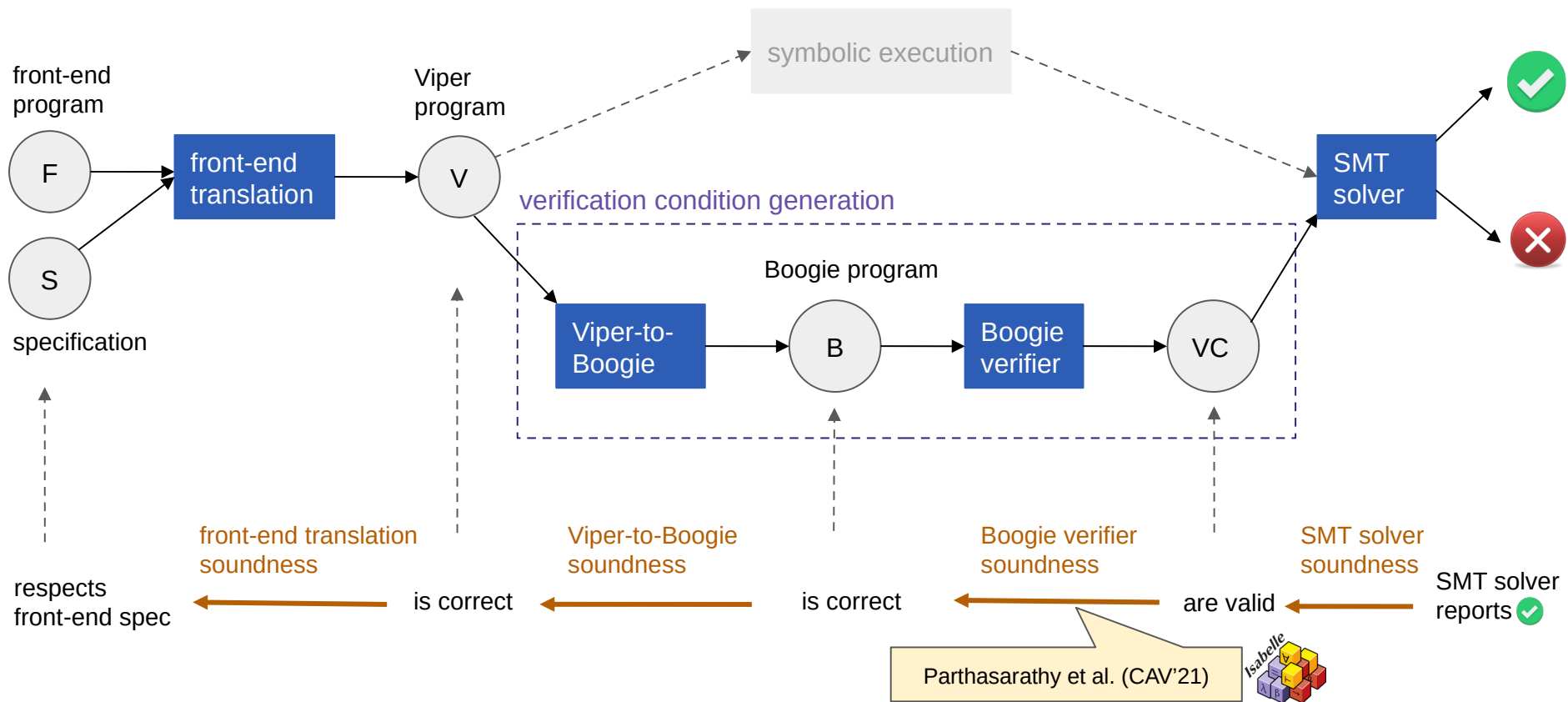
Soundness: Proof Strategy



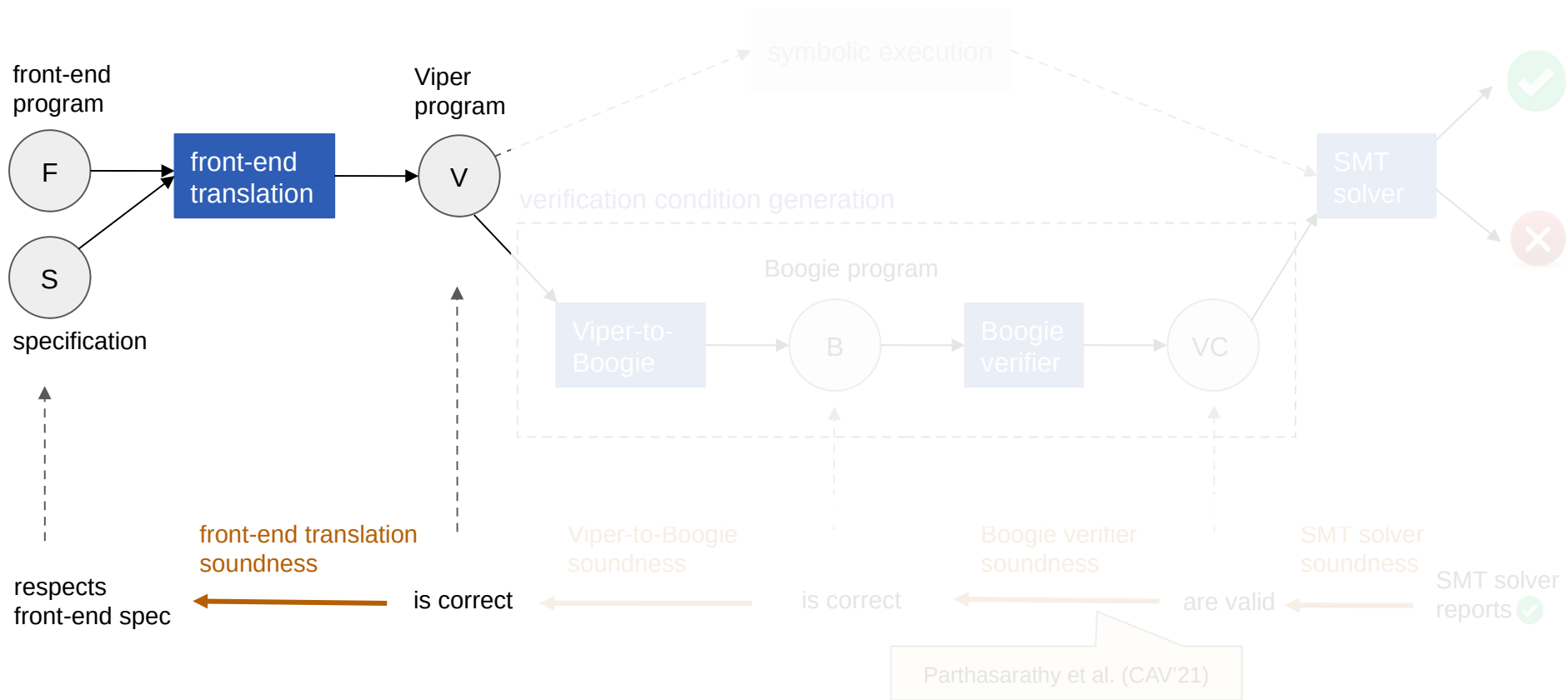
Soundness: Proof Strategy



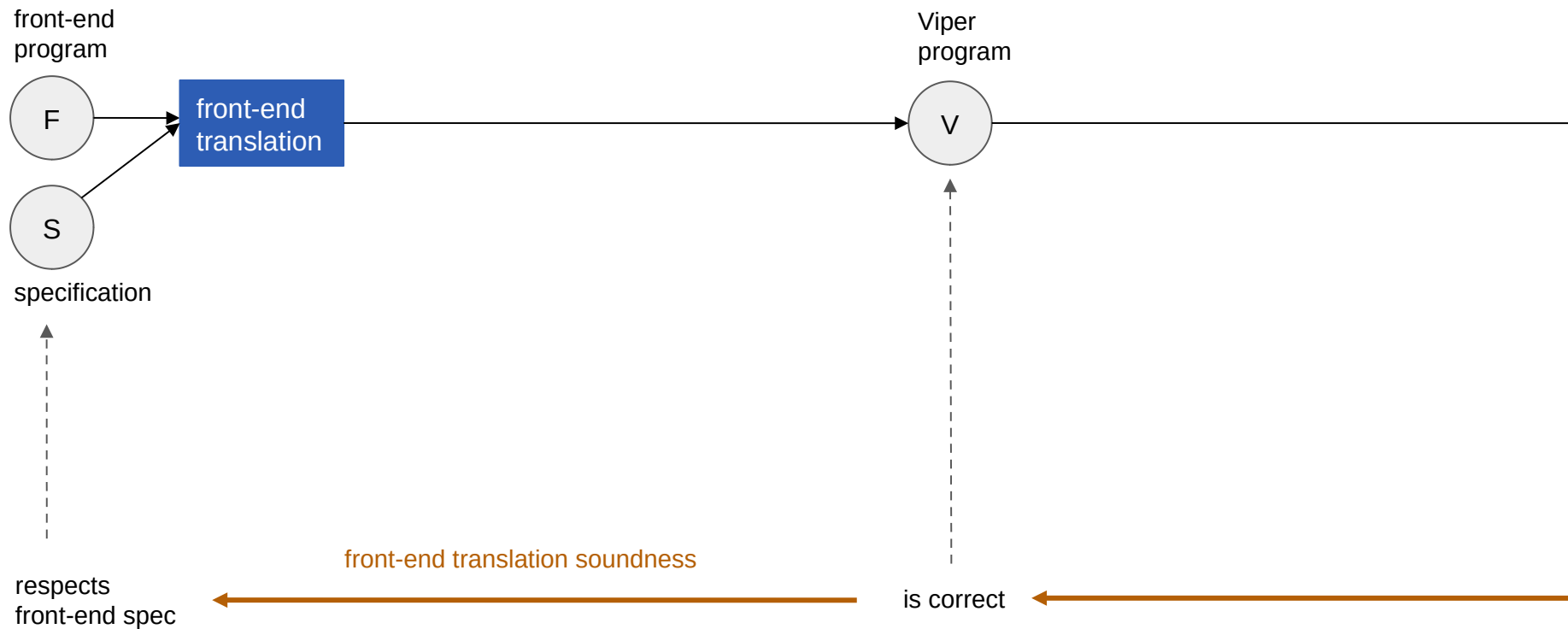
Soundness: Proof Strategy



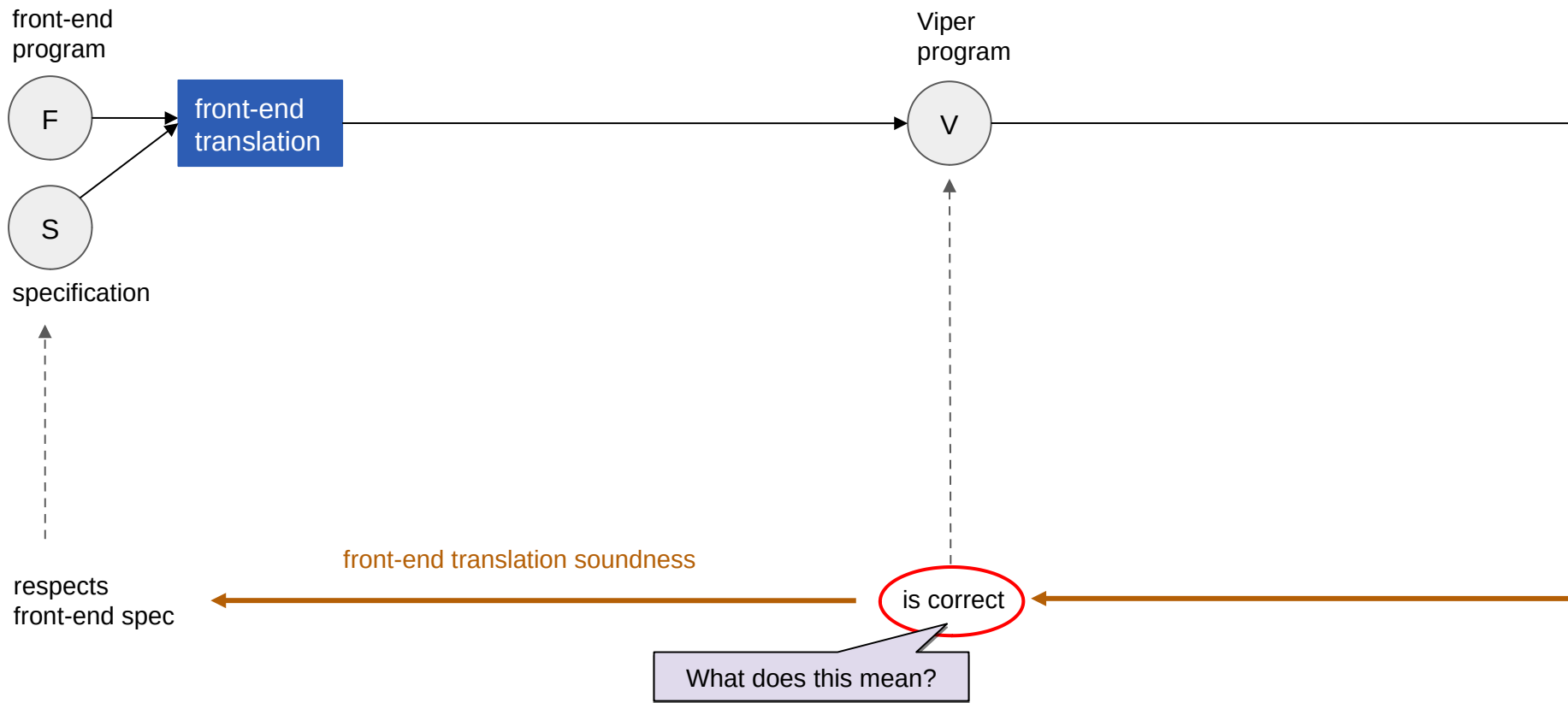
Soundness: Proof Strategy



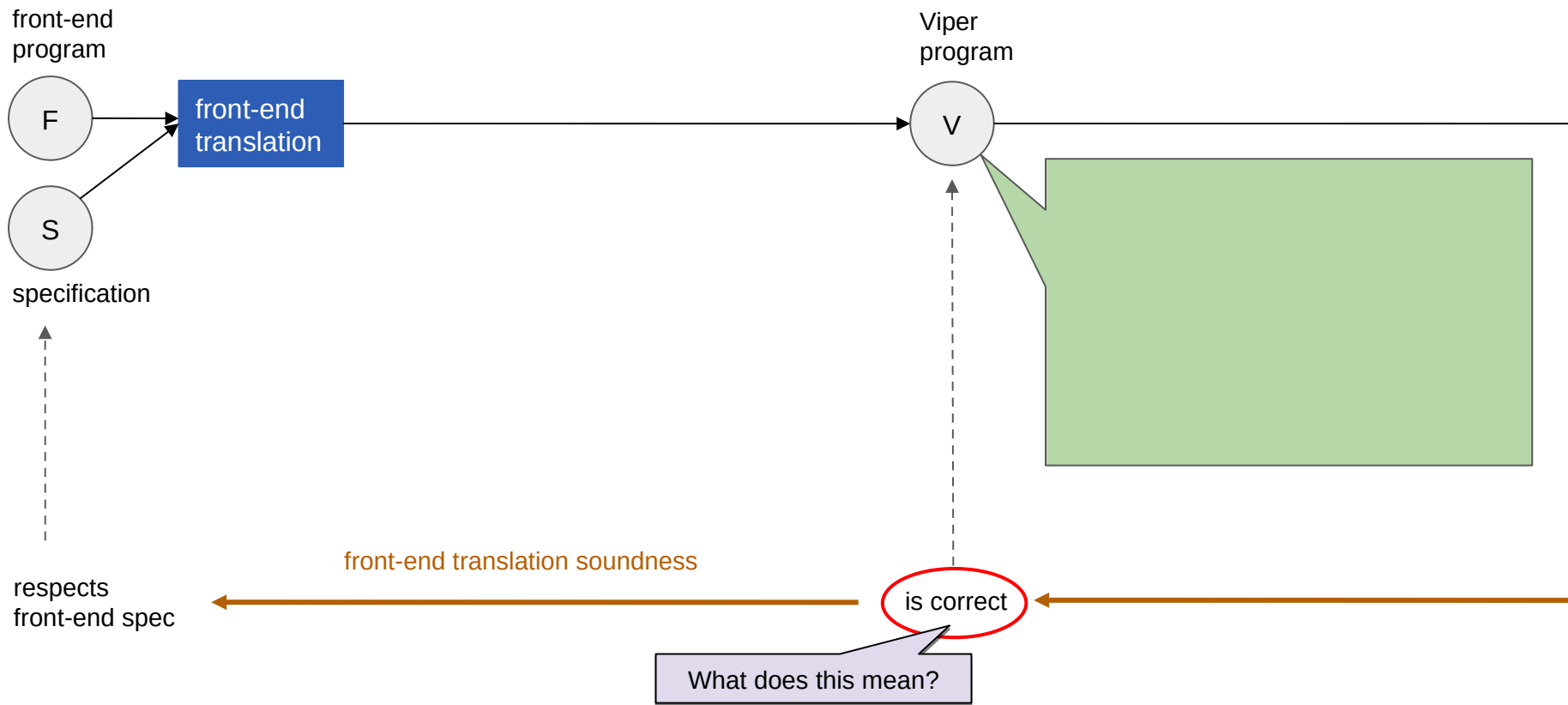
Viper's Formal Foundations



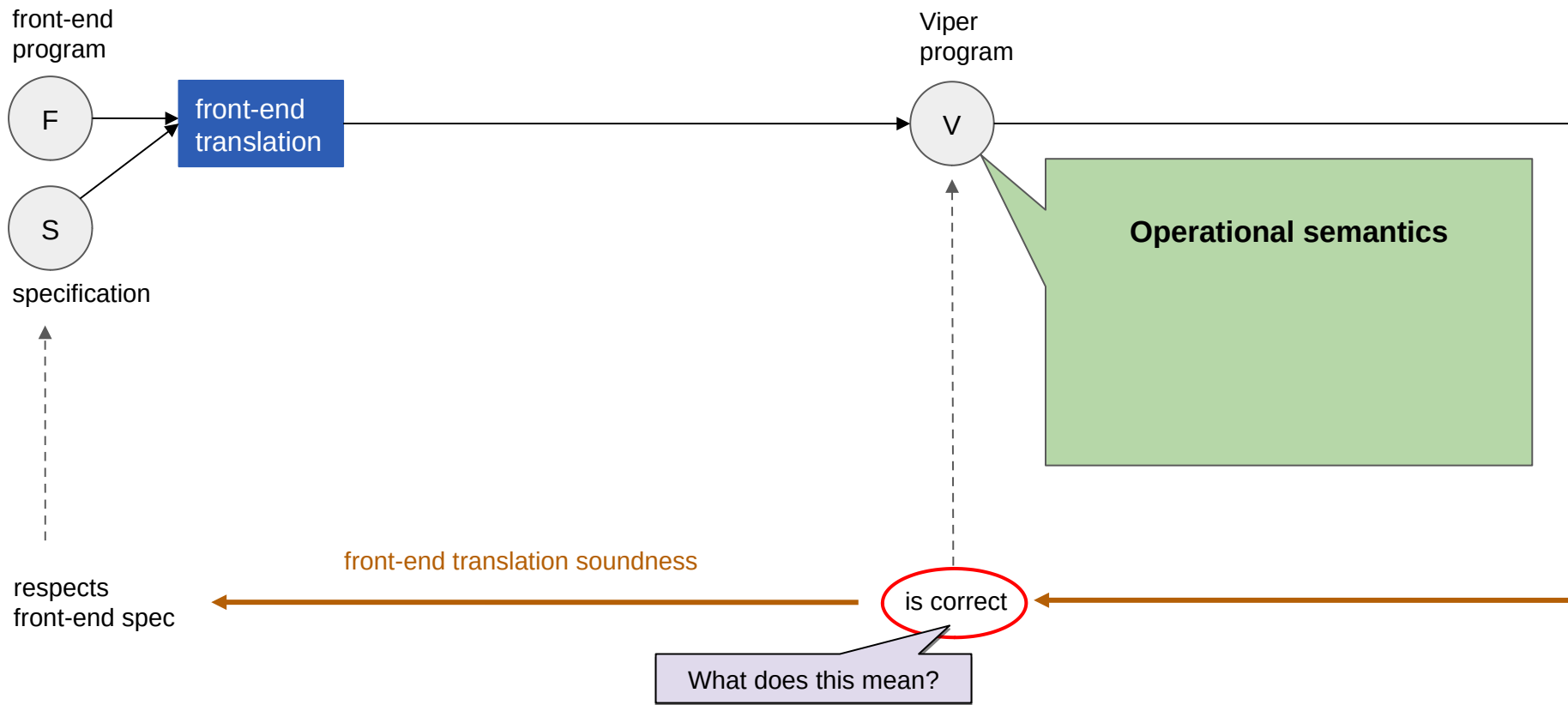
Viper's Formal Foundations



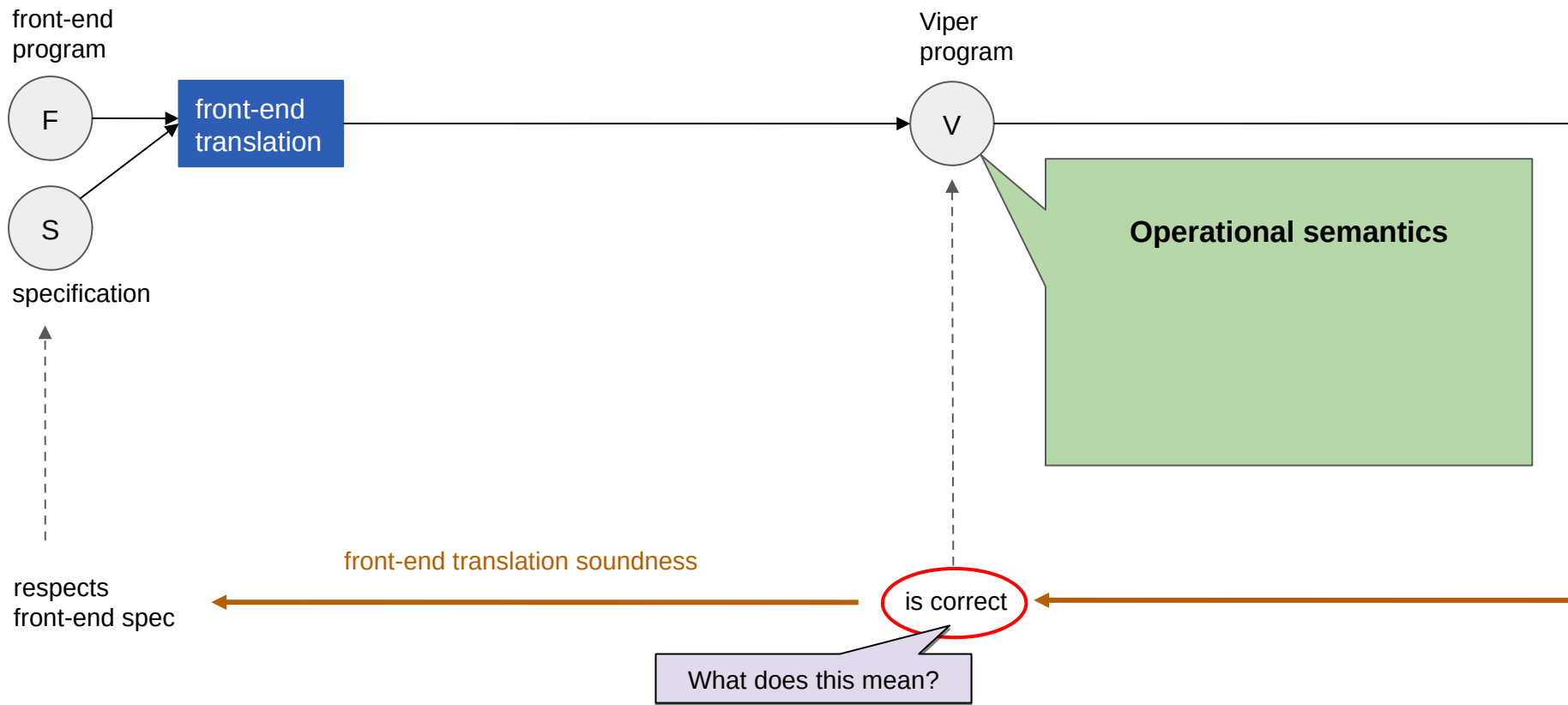
Viper's Formal Foundations



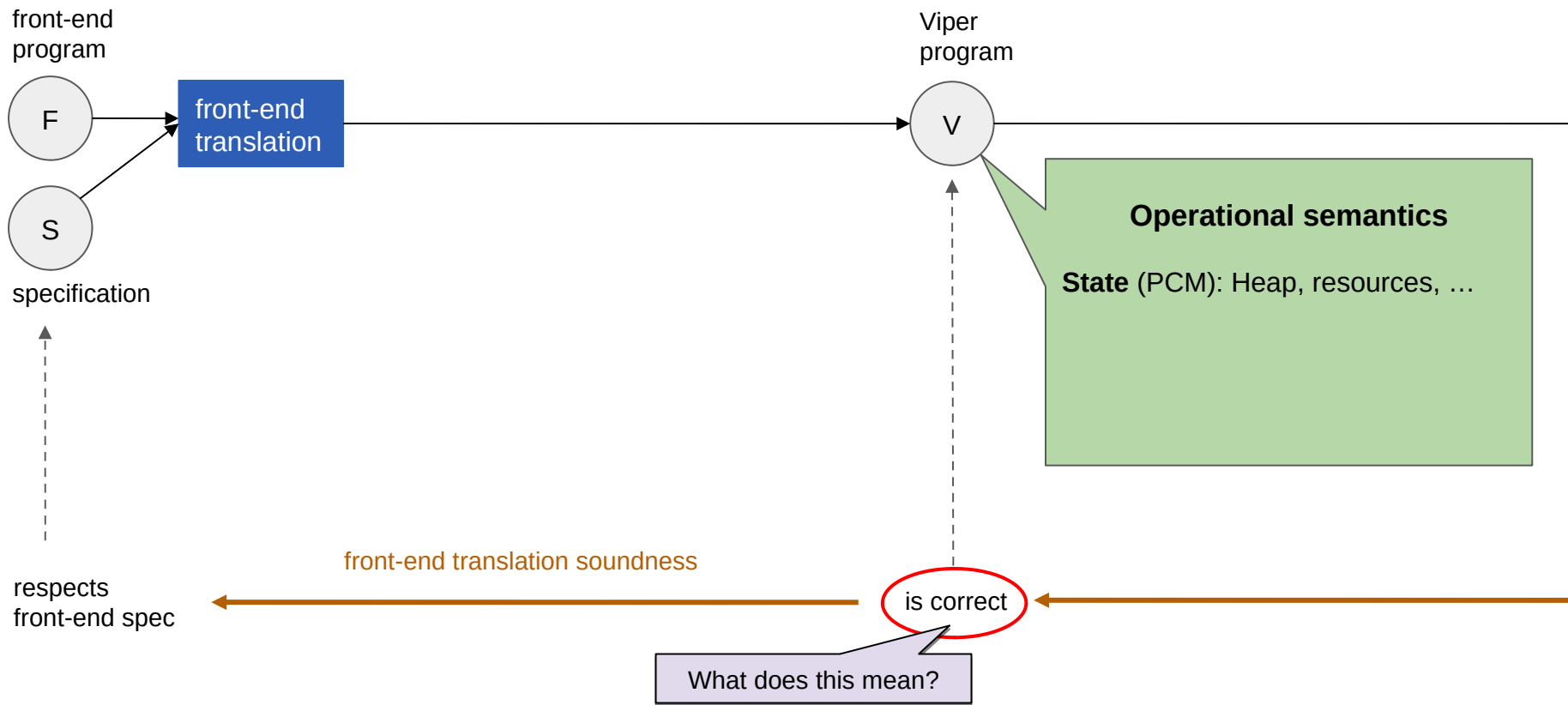
Viper's Formal Foundations



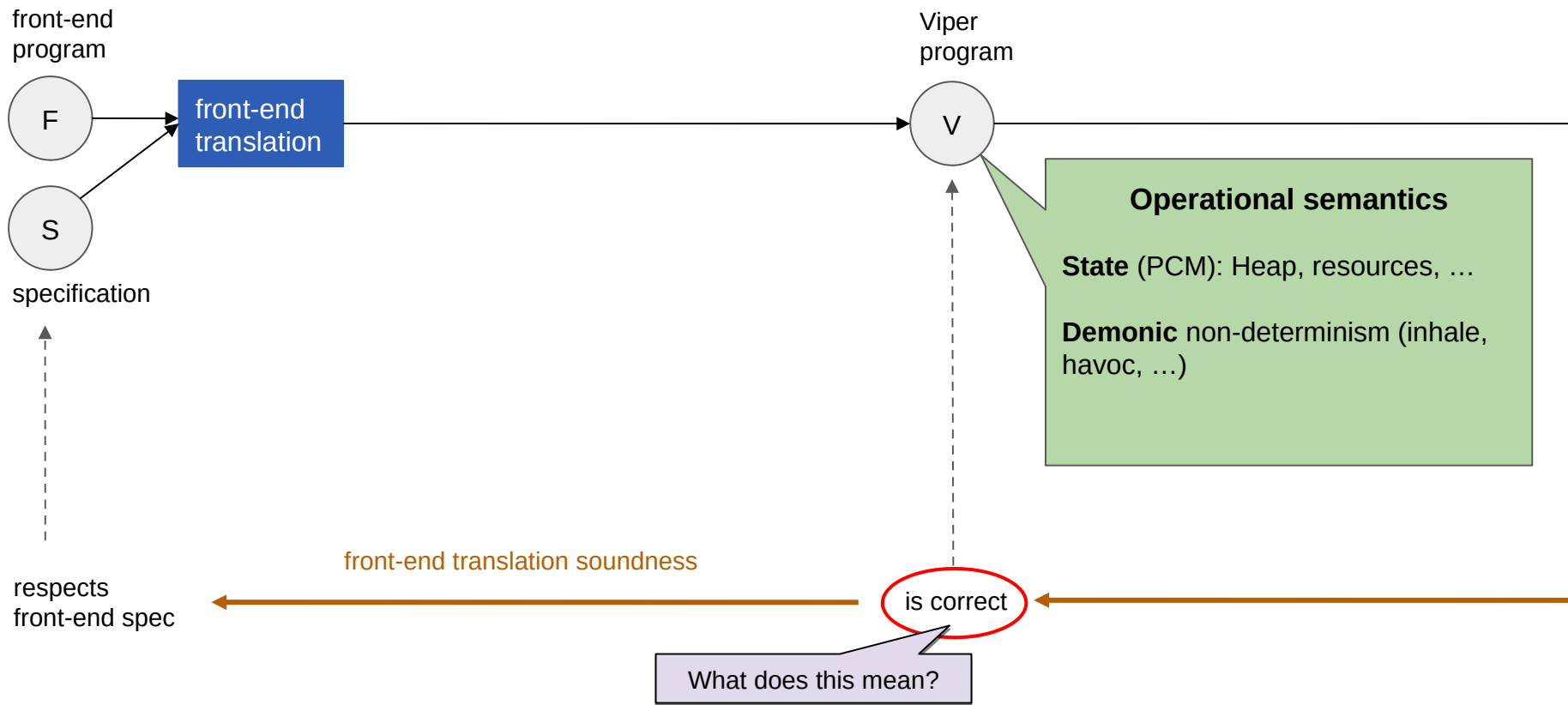
Viper's Formal Foundations



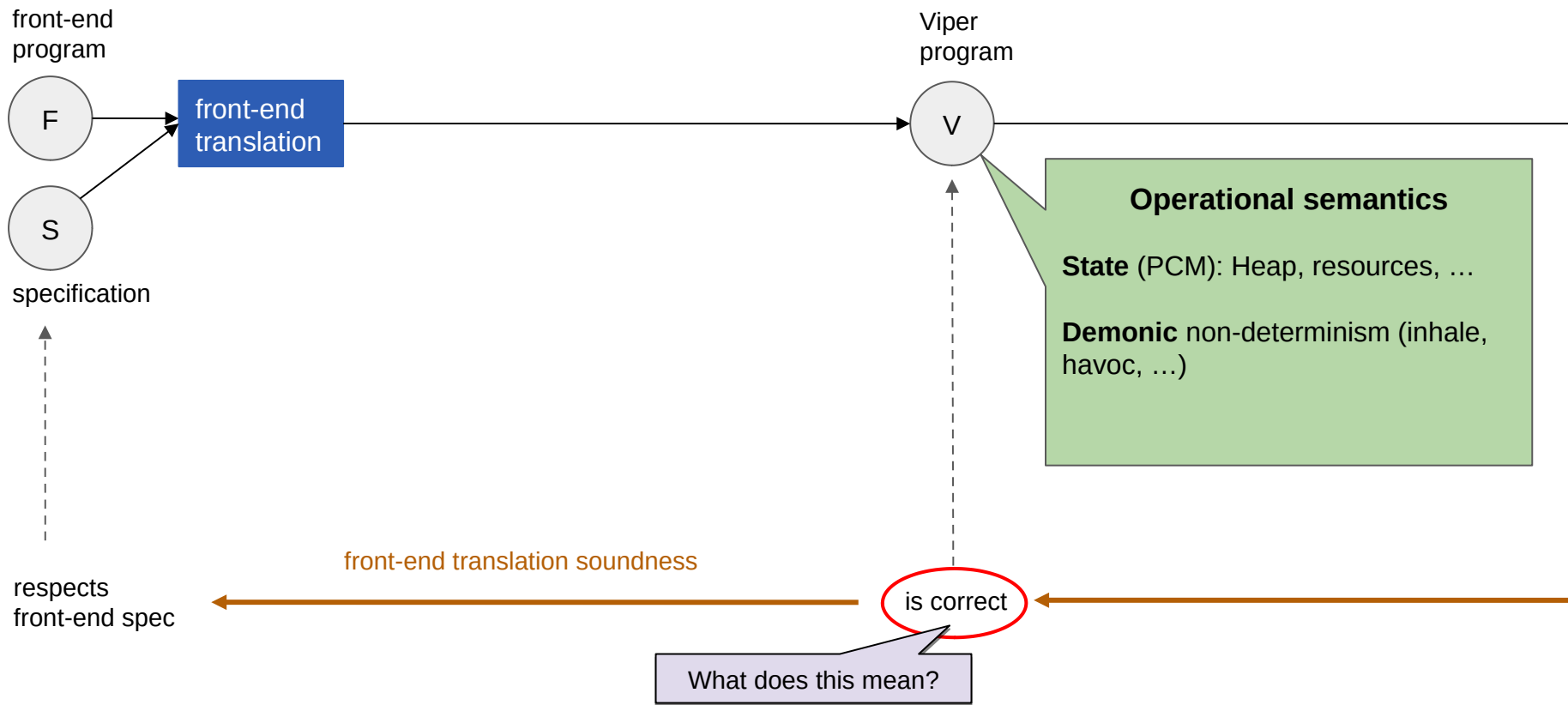
Viper's Formal Foundations



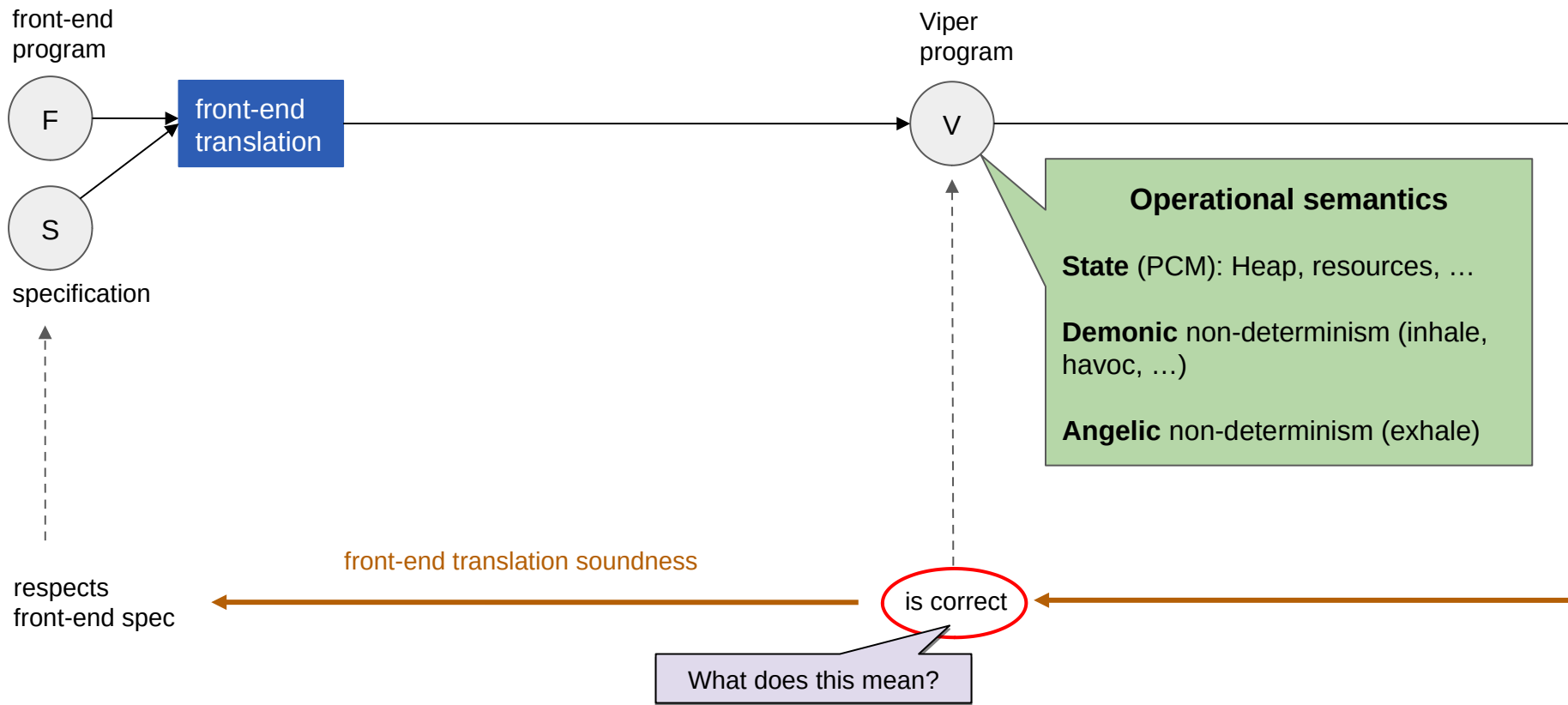
Viper's Formal Foundations



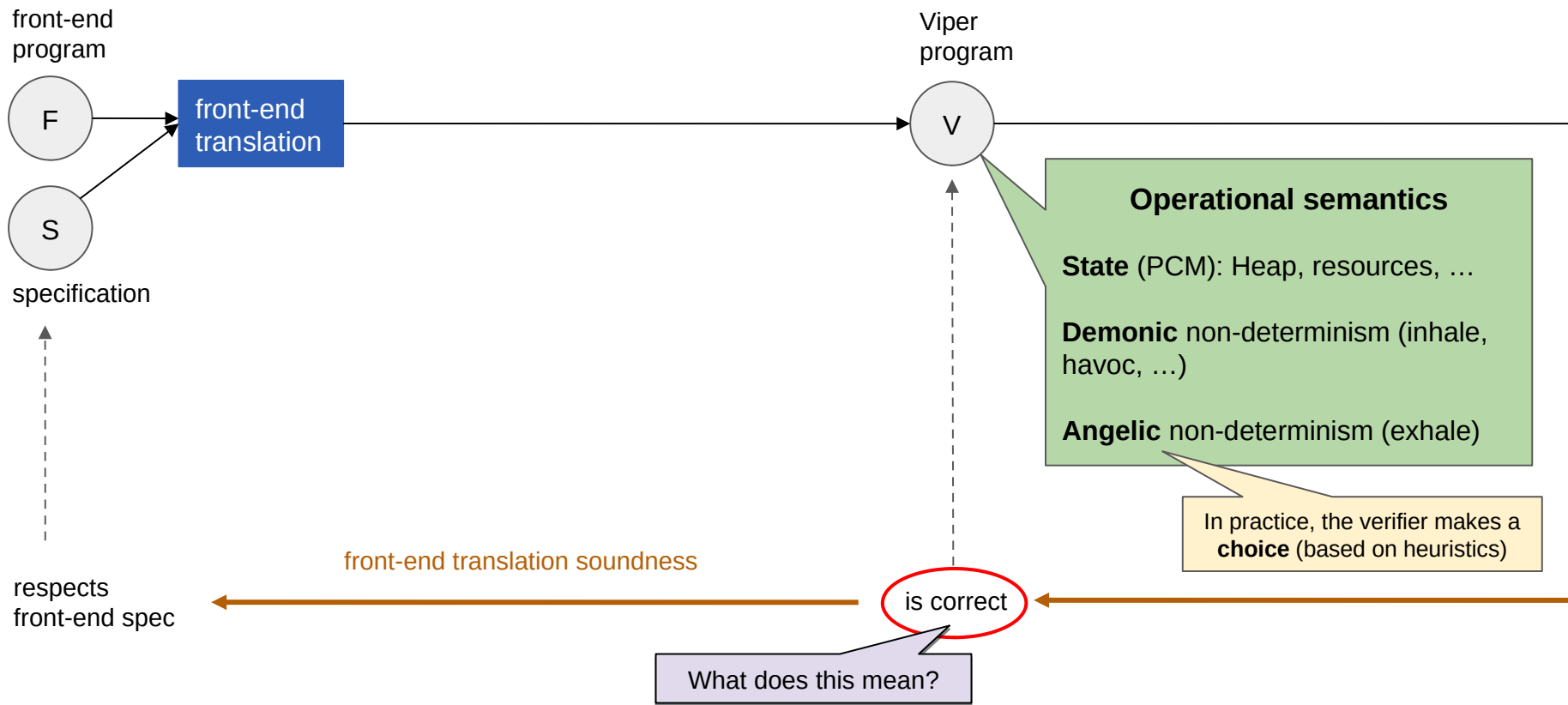
Viper's Formal Foundations



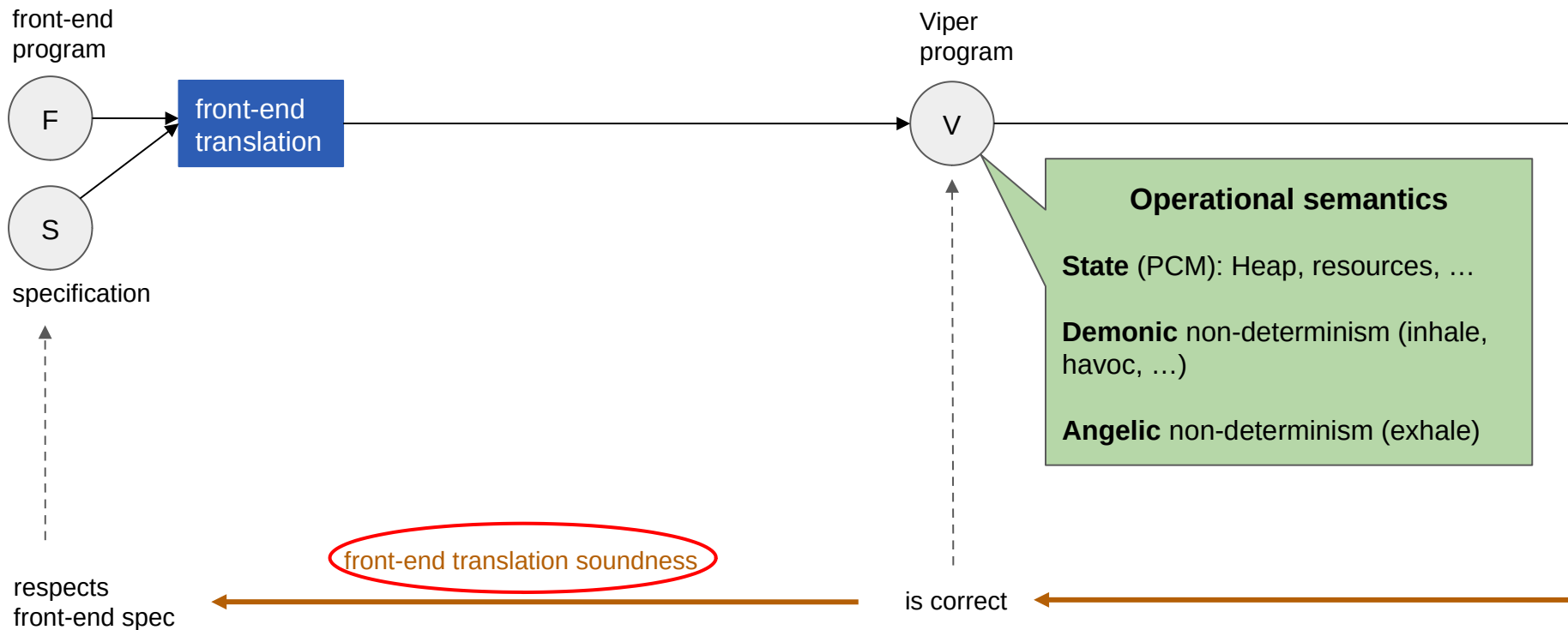
Viper's Formal Foundations



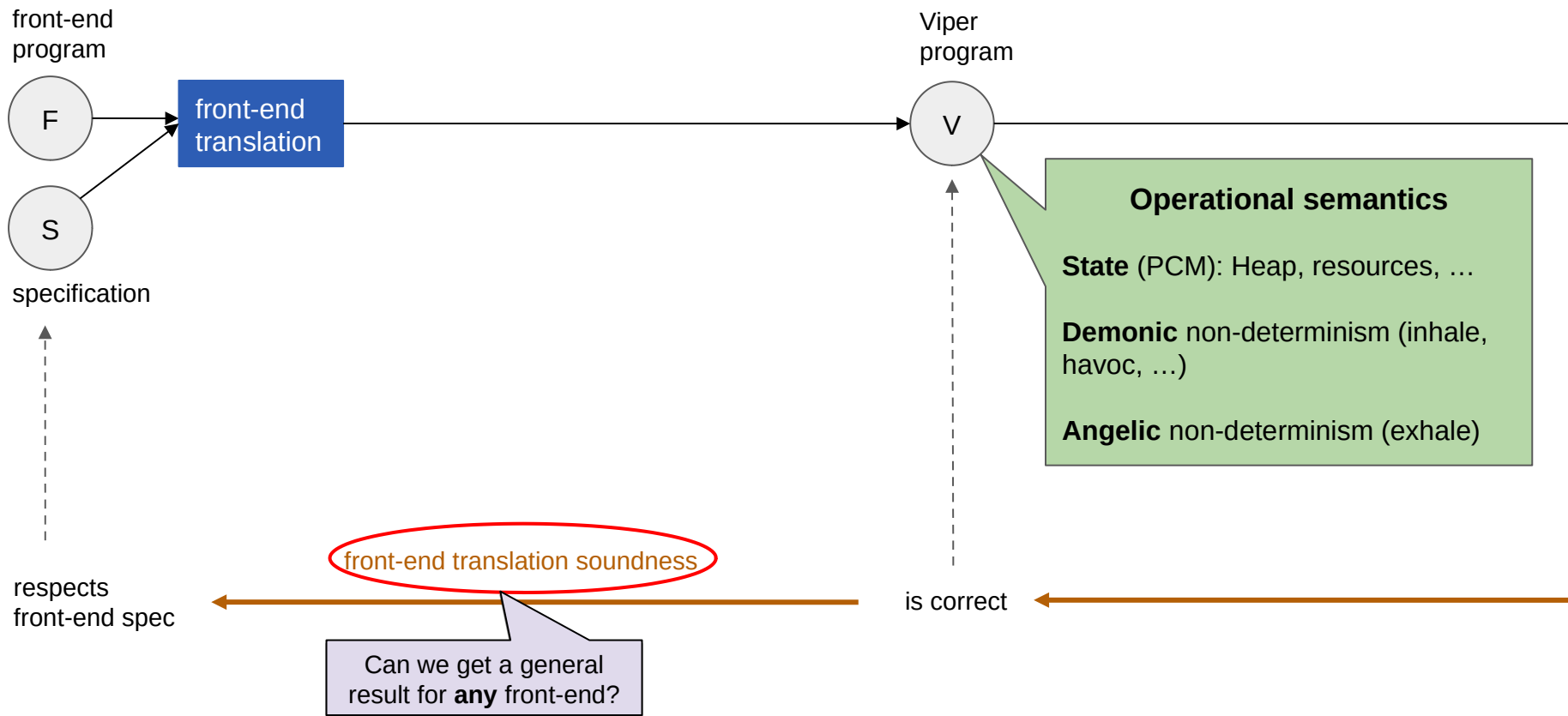
Viper's Formal Foundations



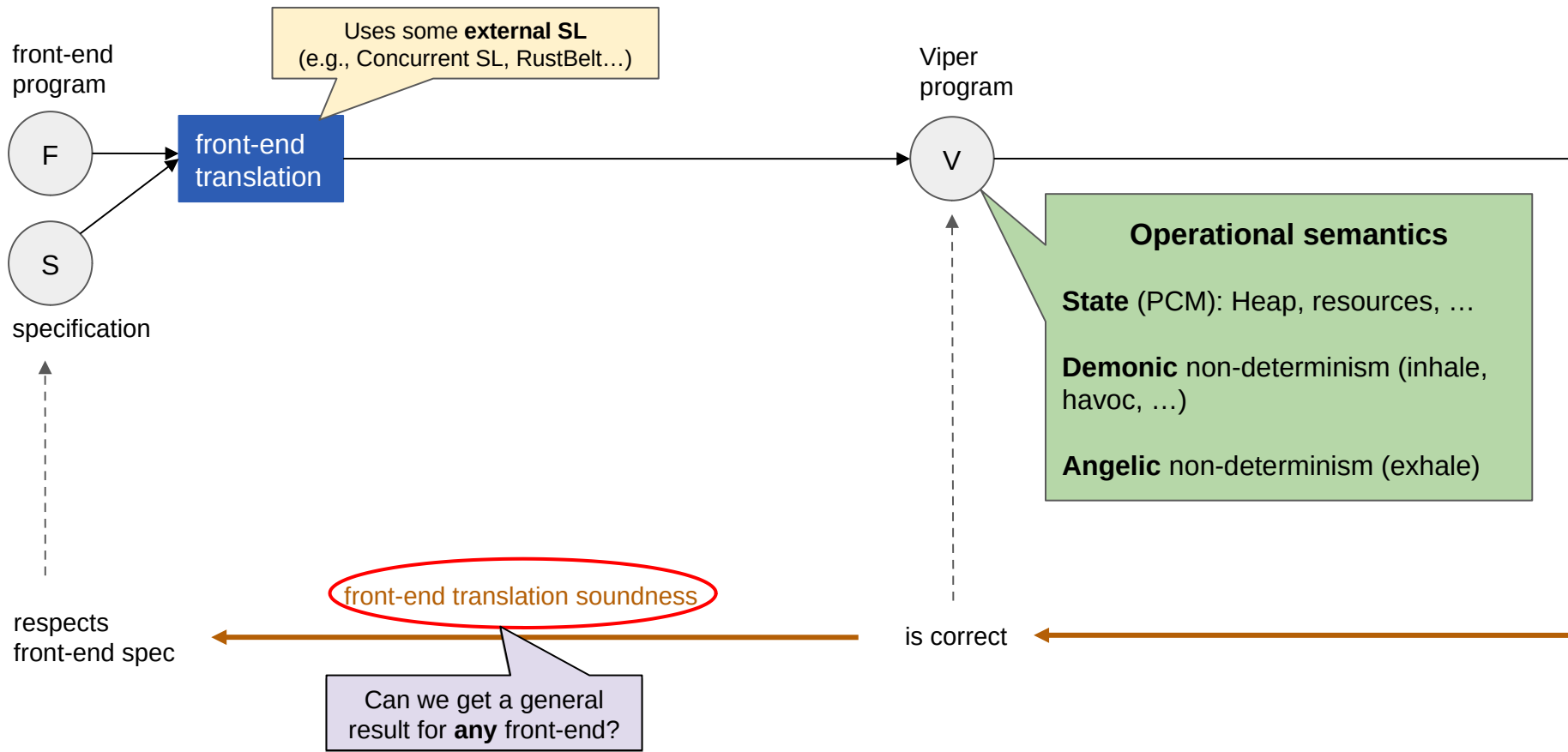
Viper's Formal Foundations



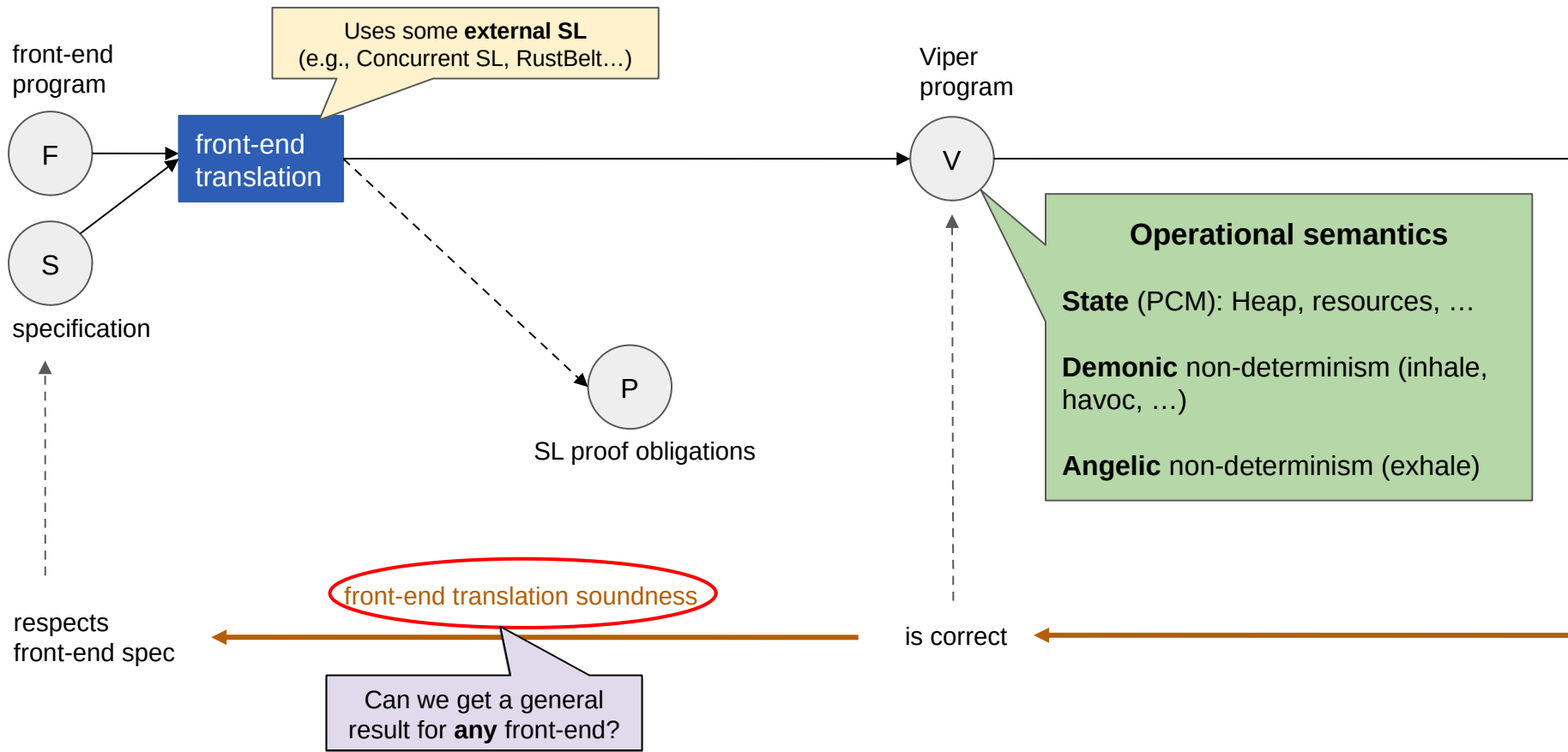
Viper's Formal Foundations



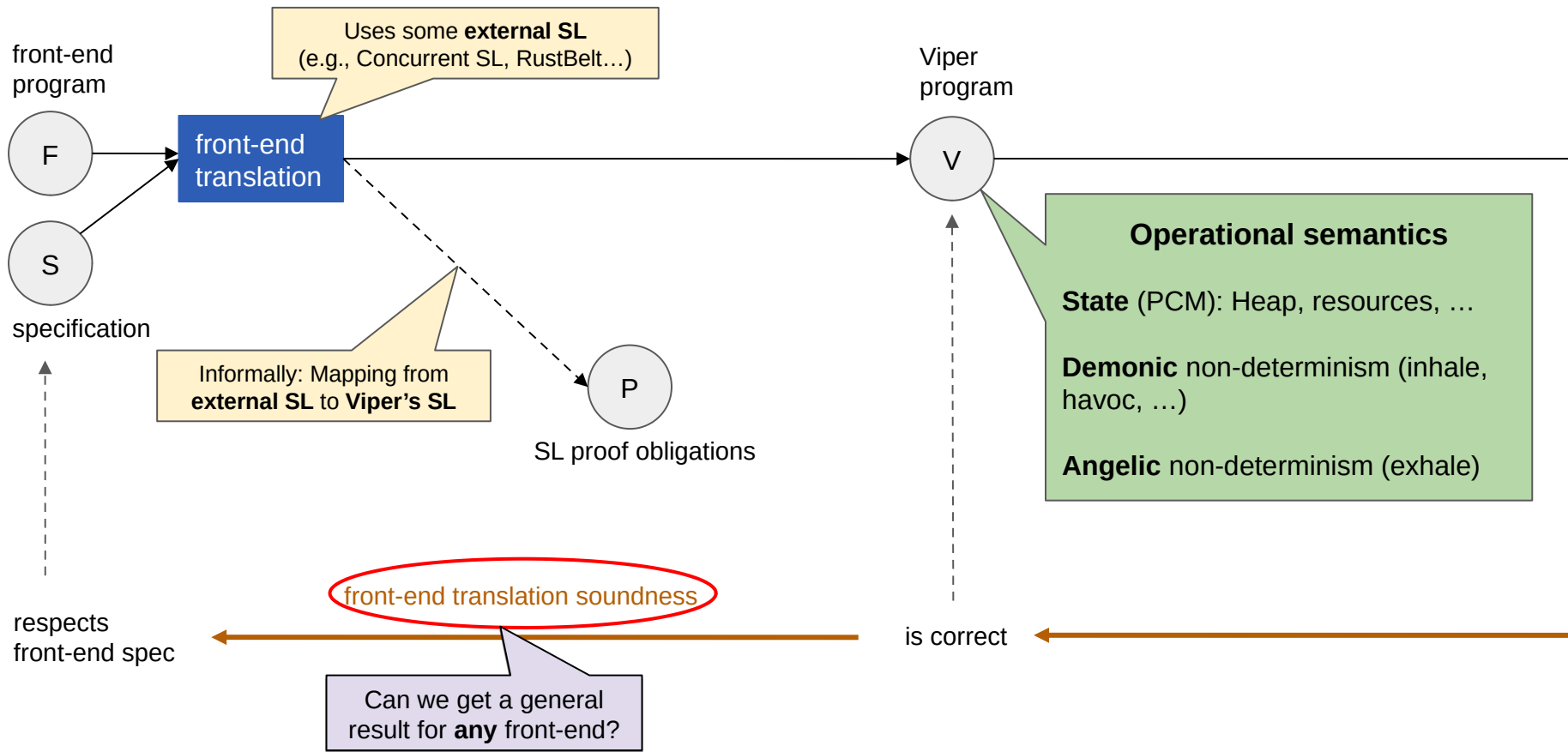
Viper's Formal Foundations



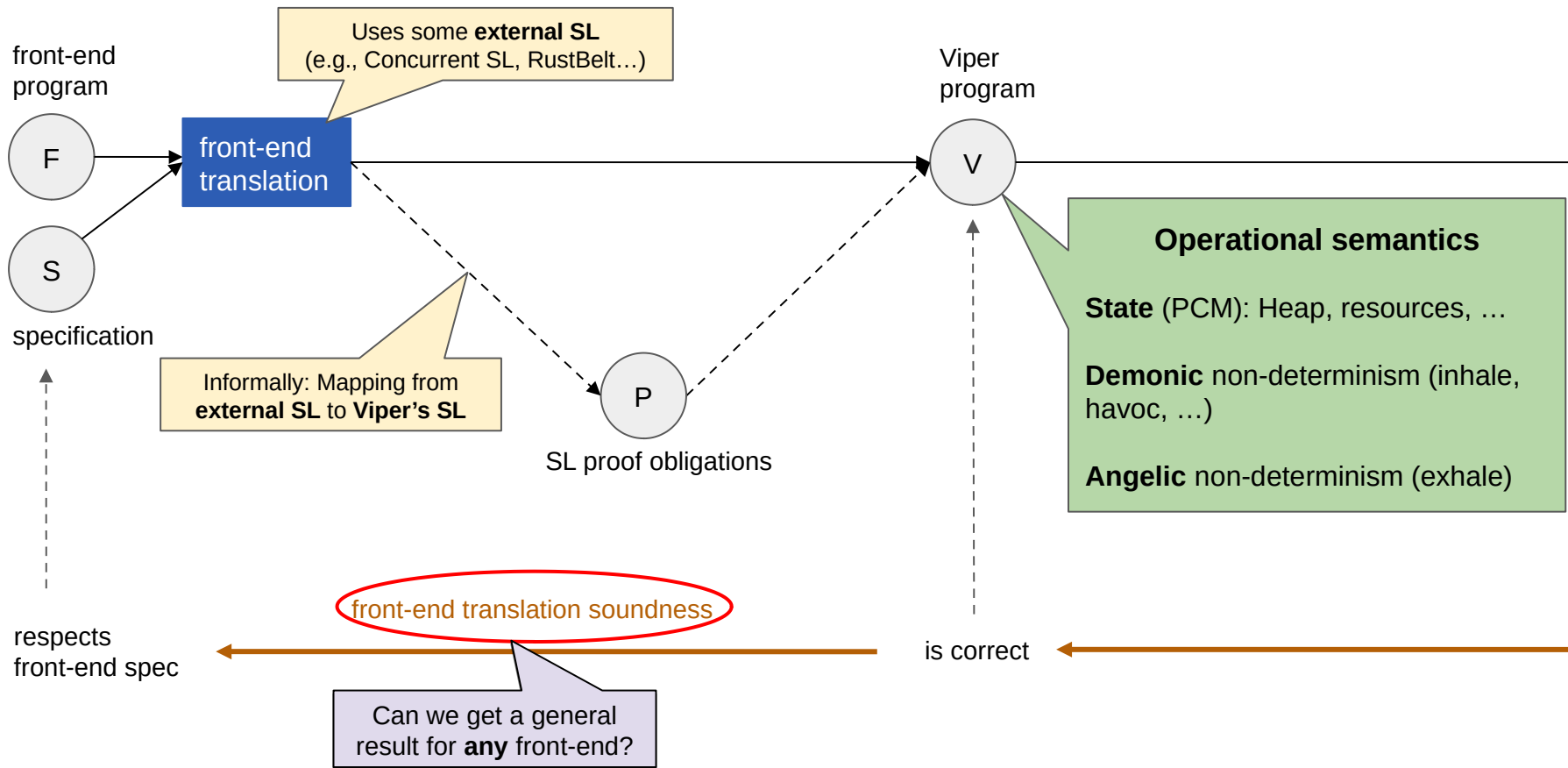
Viper's Formal Foundations



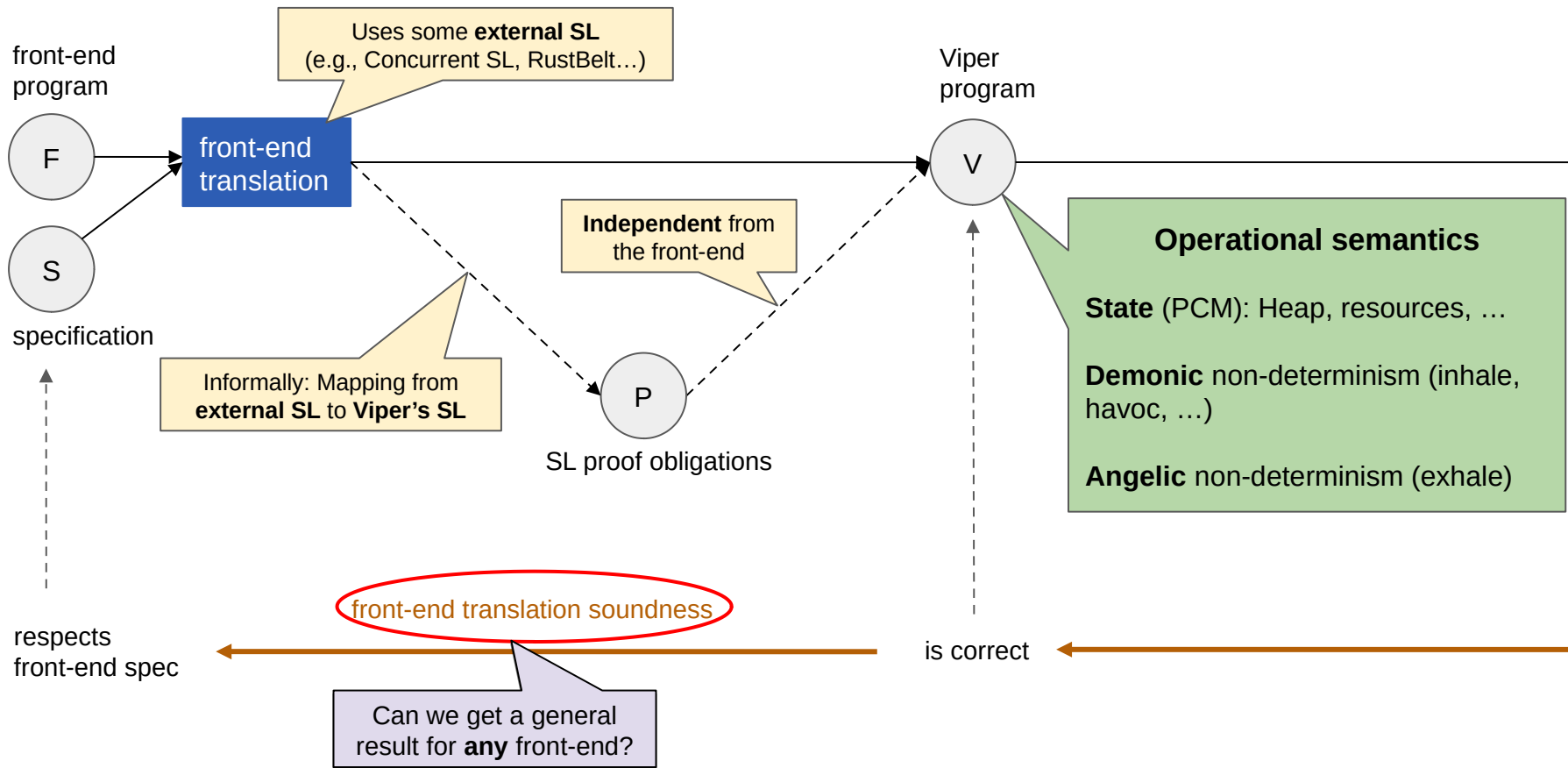
Viper's Formal Foundations



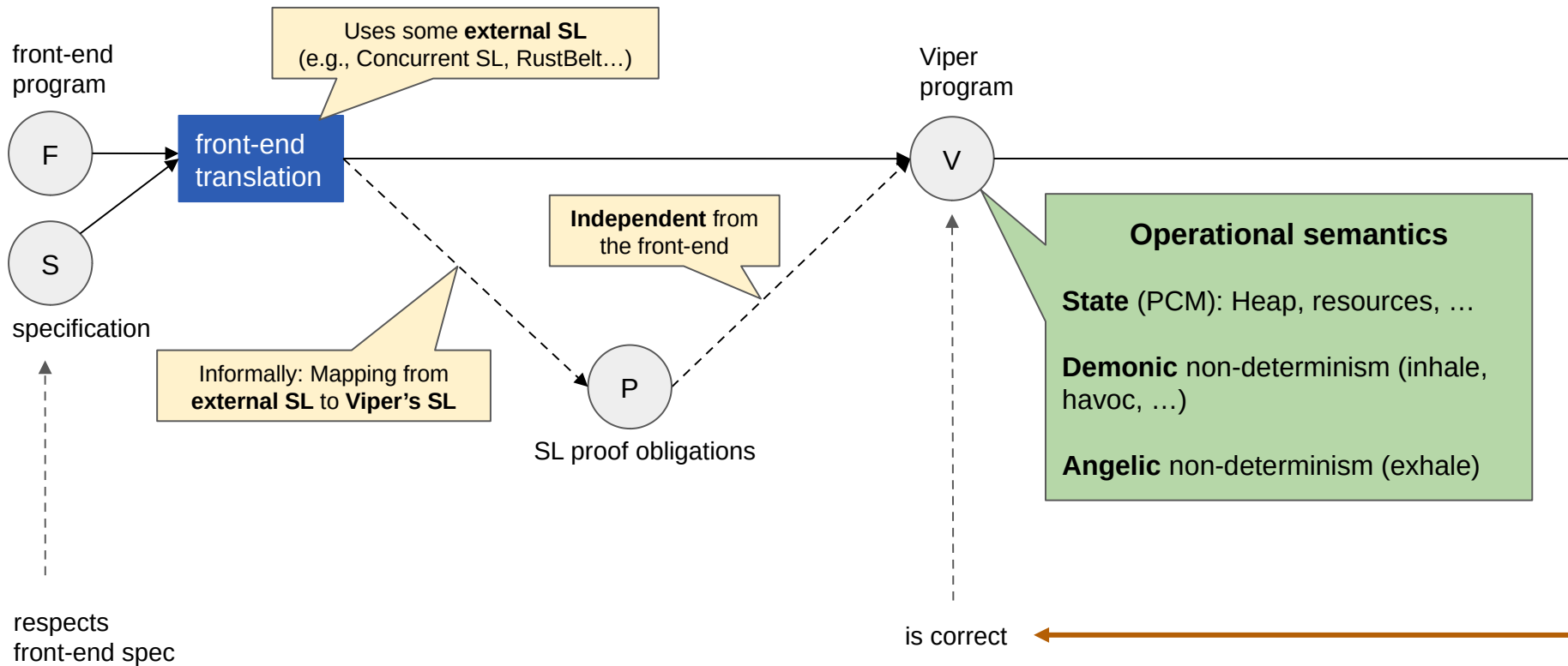
Viper's Formal Foundations



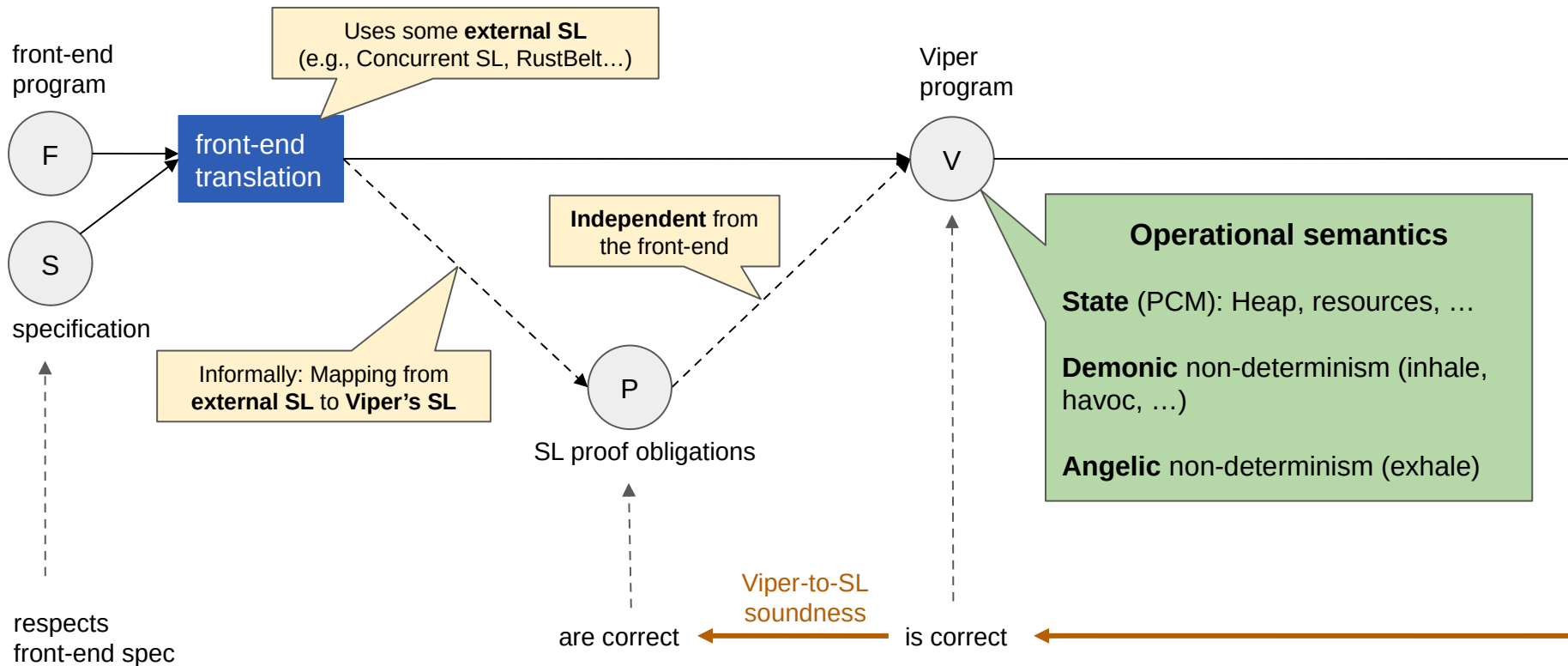
Viper's Formal Foundations



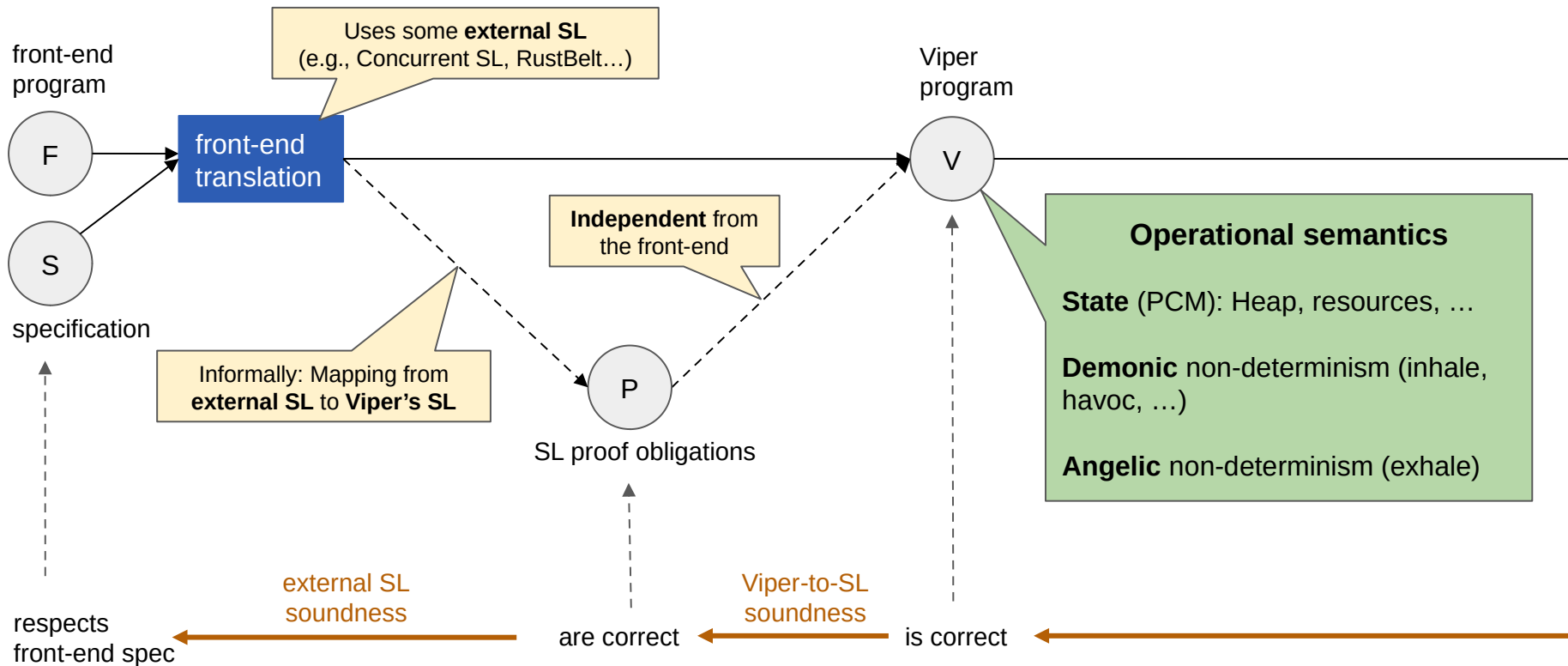
Viper's Formal Foundations



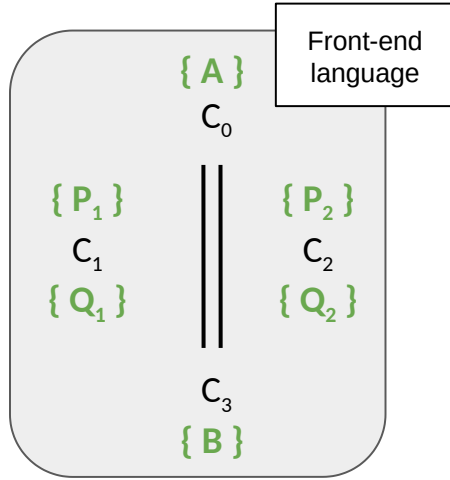
Viper's Formal Foundations



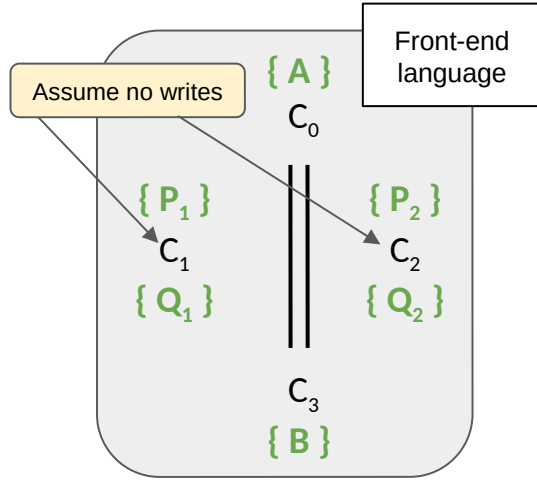
Viper's Formal Foundations



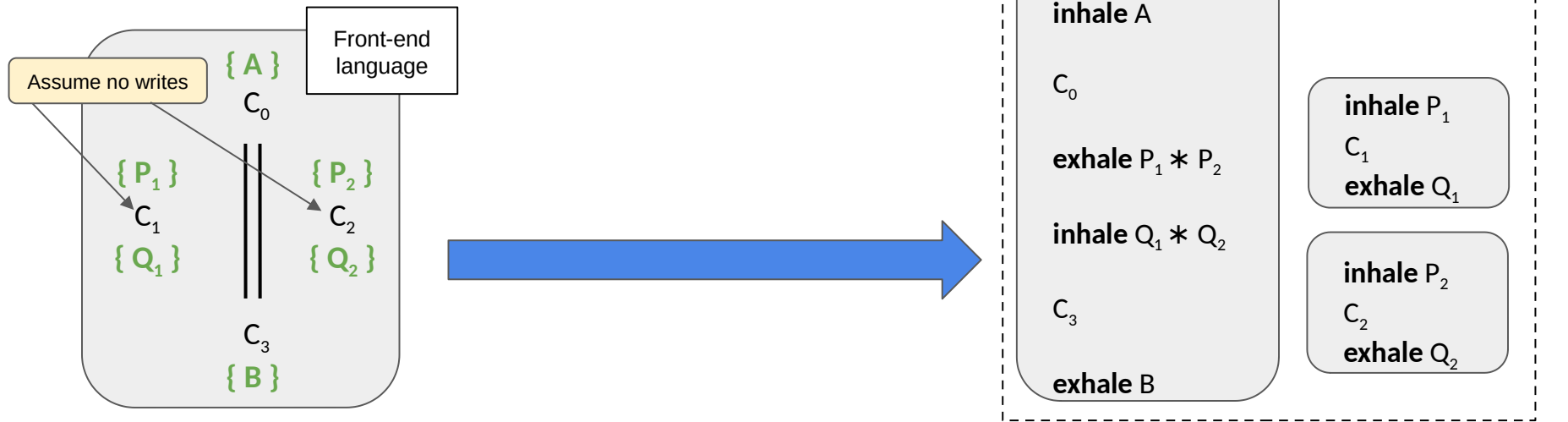
Viper-to-SL: Parallel Program



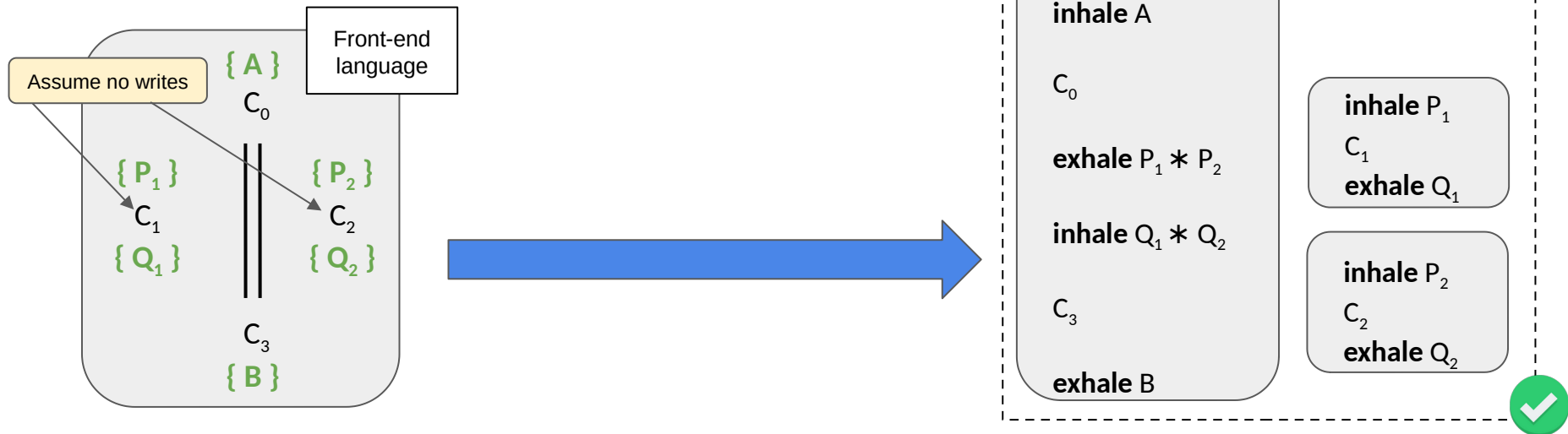
Viper-to-SL: Parallel Program



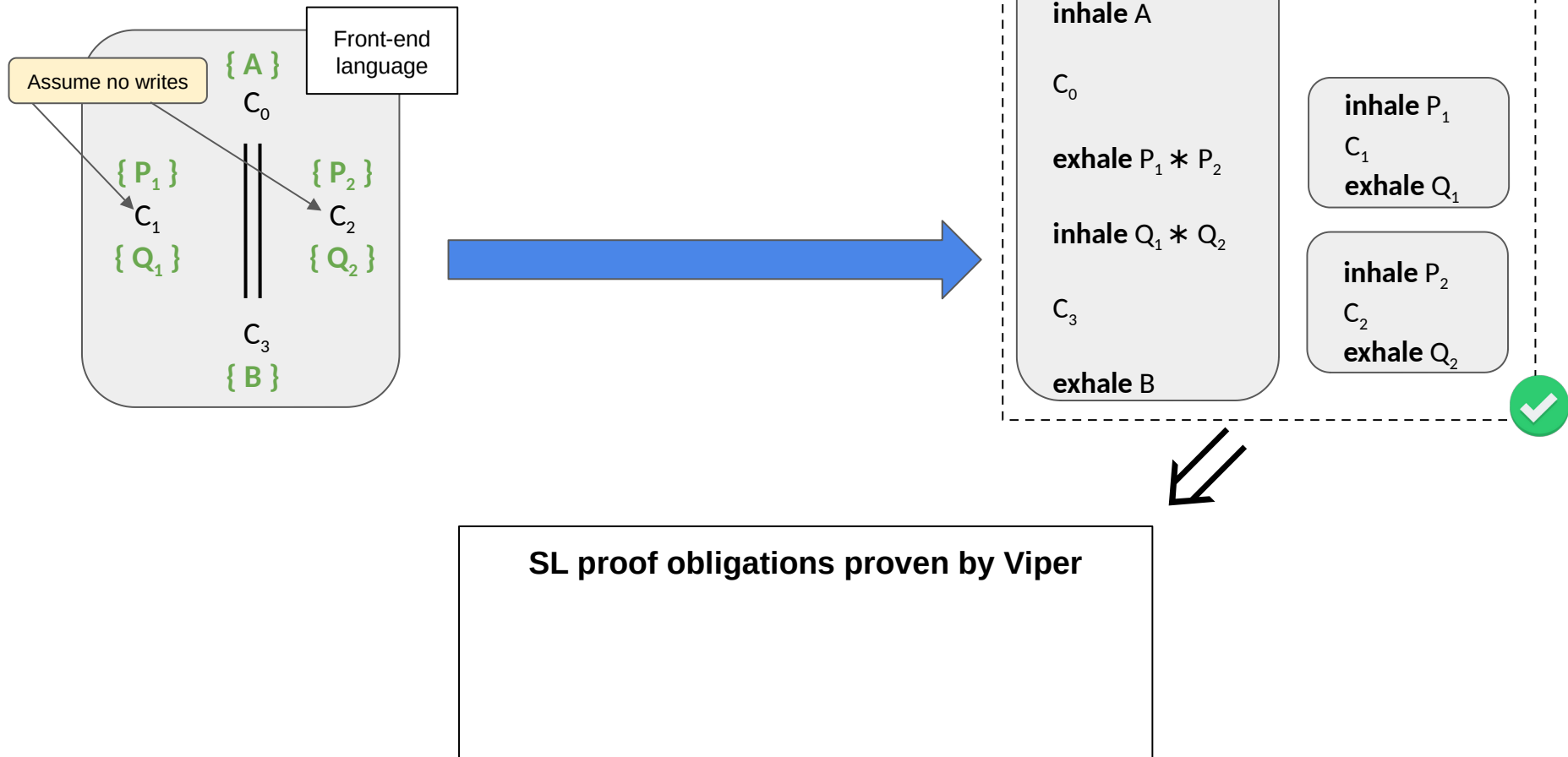
Viper-to-SL: Parallel Program



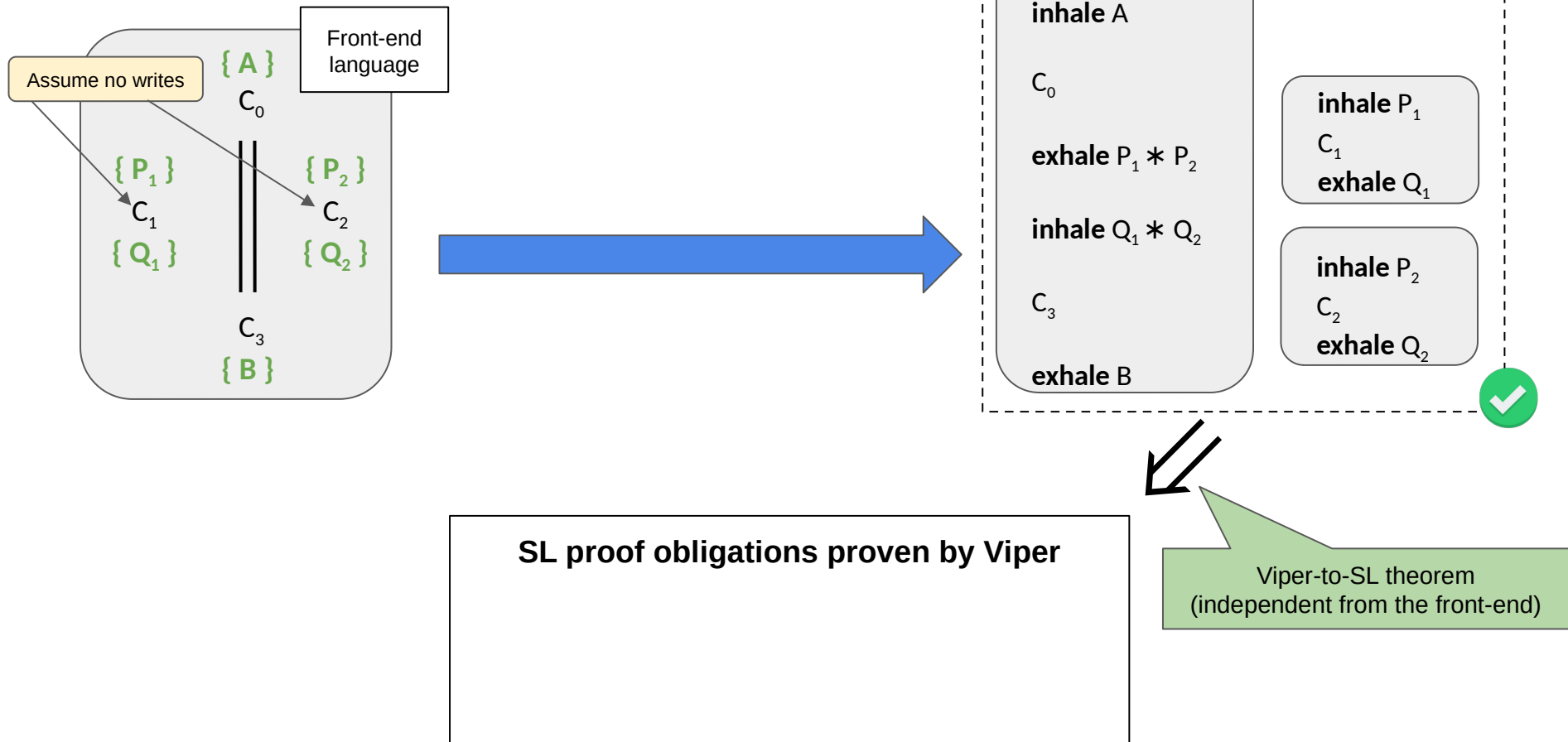
Viper-to-SL: Parallel Program



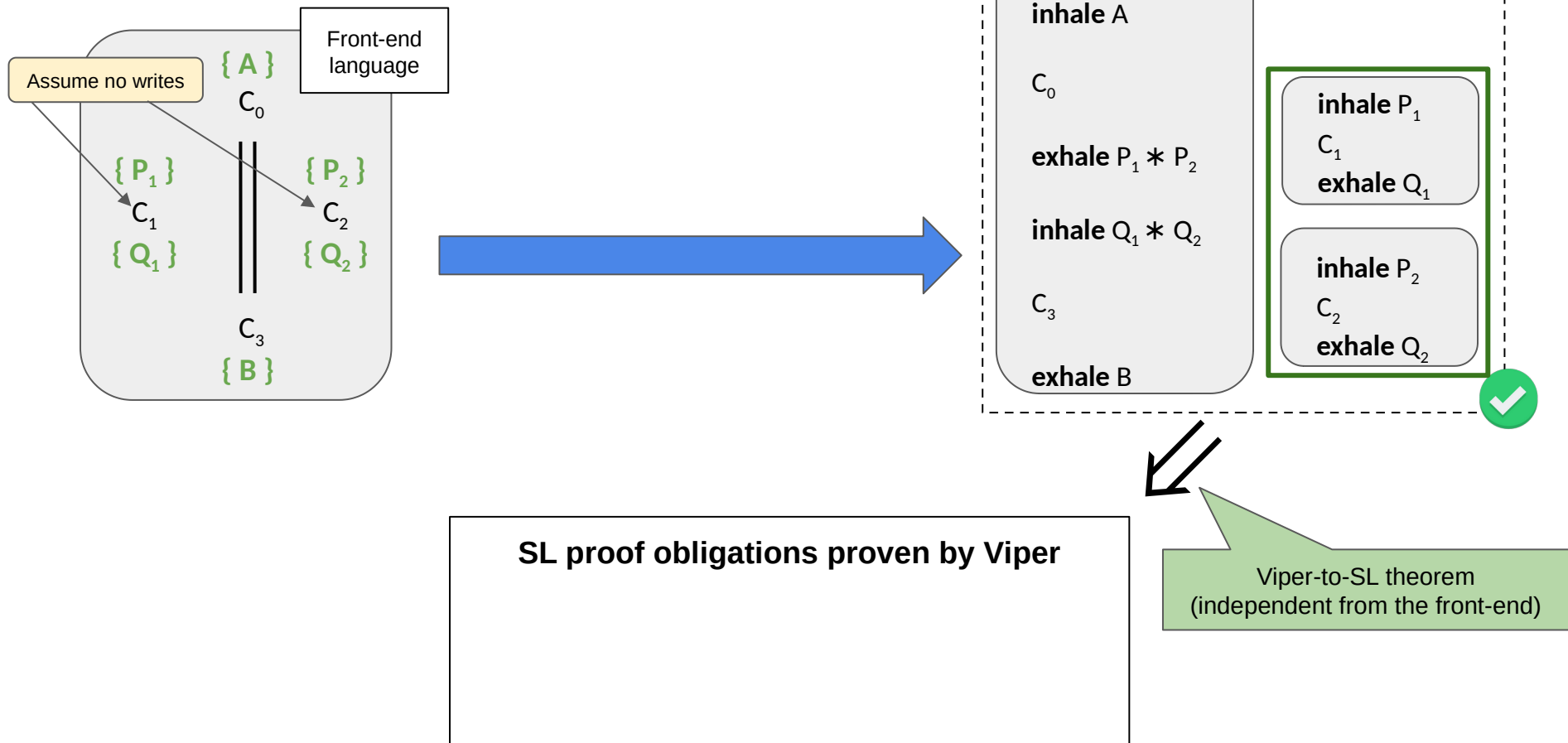
Viper-to-SL: Parallel Program



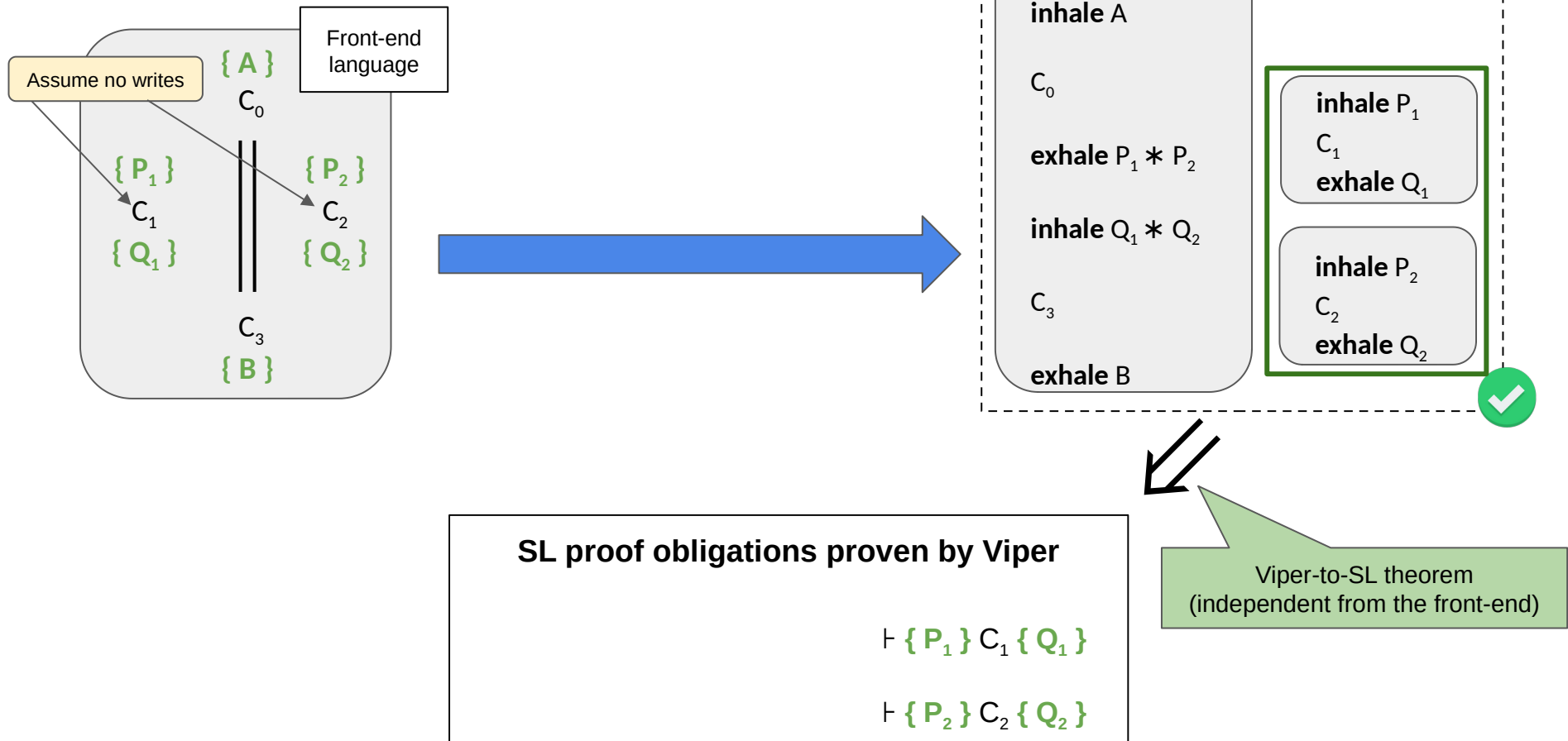
Viper-to-SL: Parallel Program



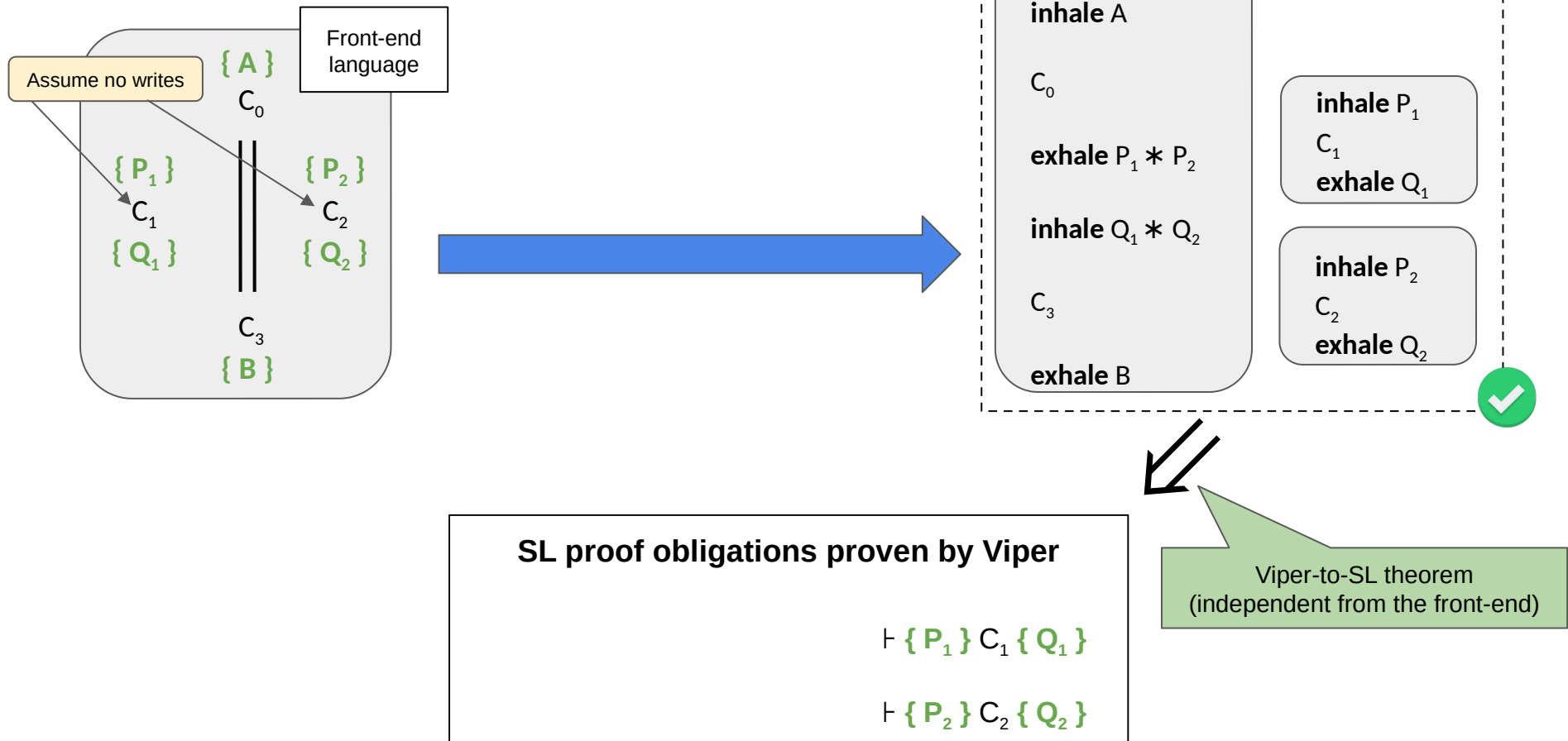
Viper-to-SL: Parallel Program



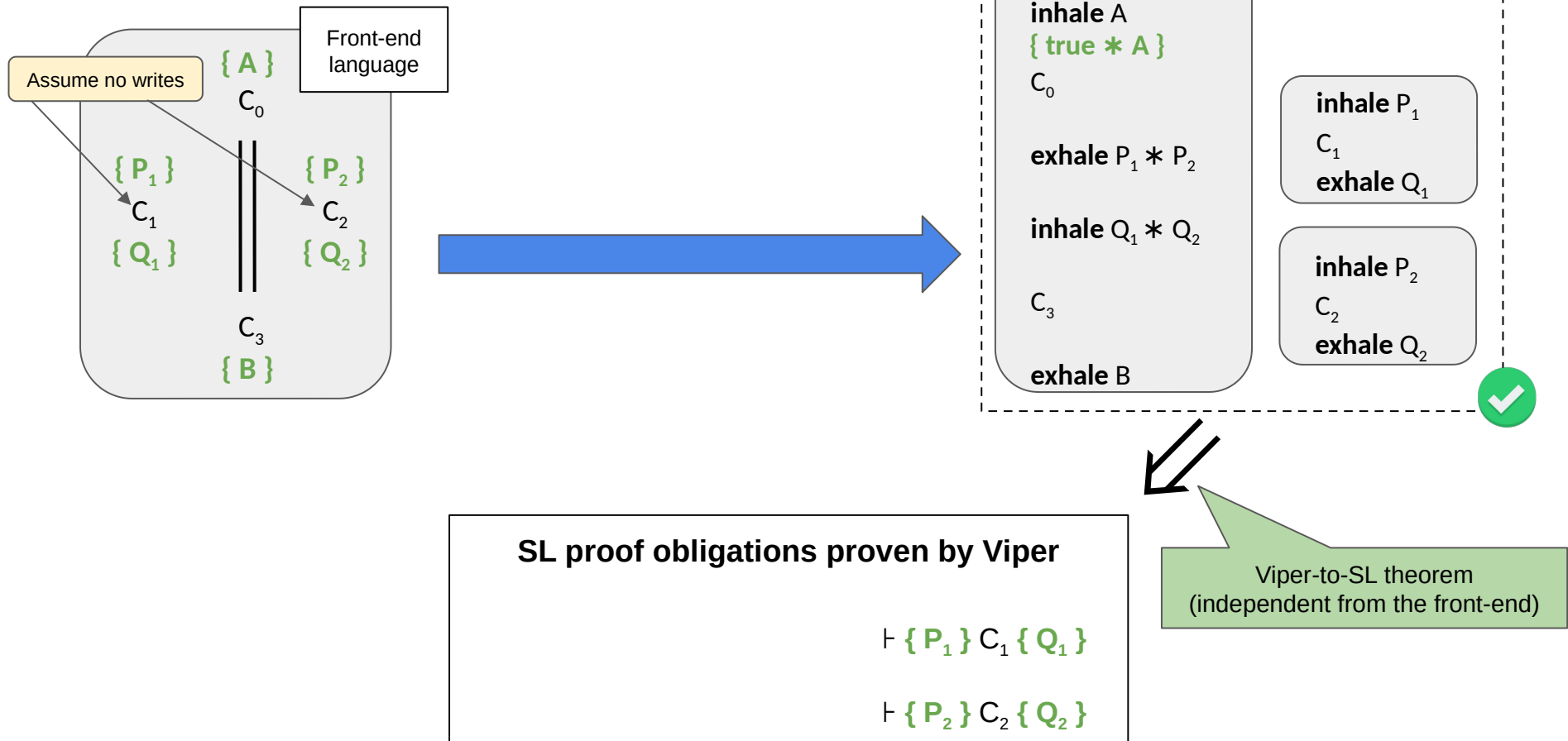
Viper-to-SL: Parallel Program



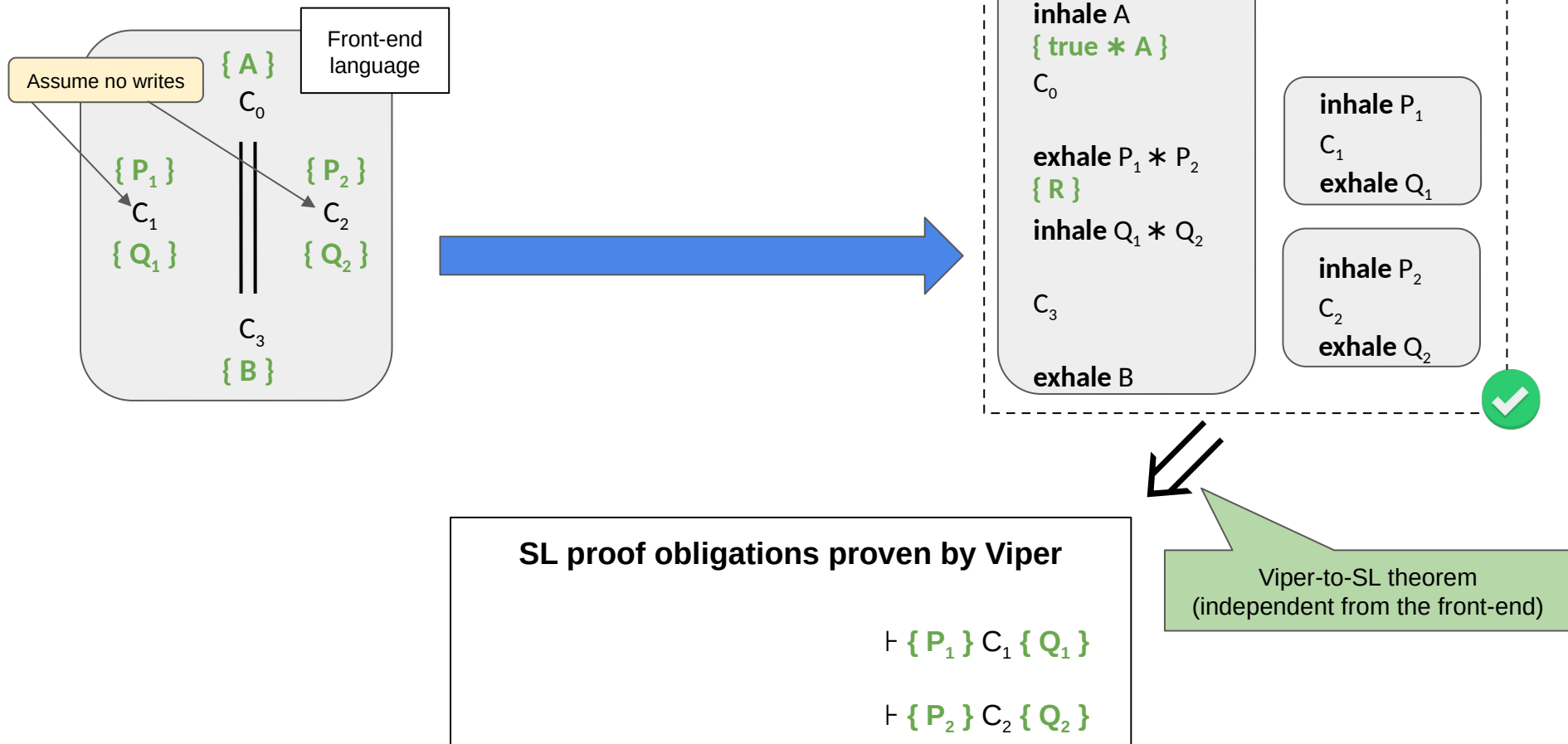
Viper-to-SL: Parallel Program



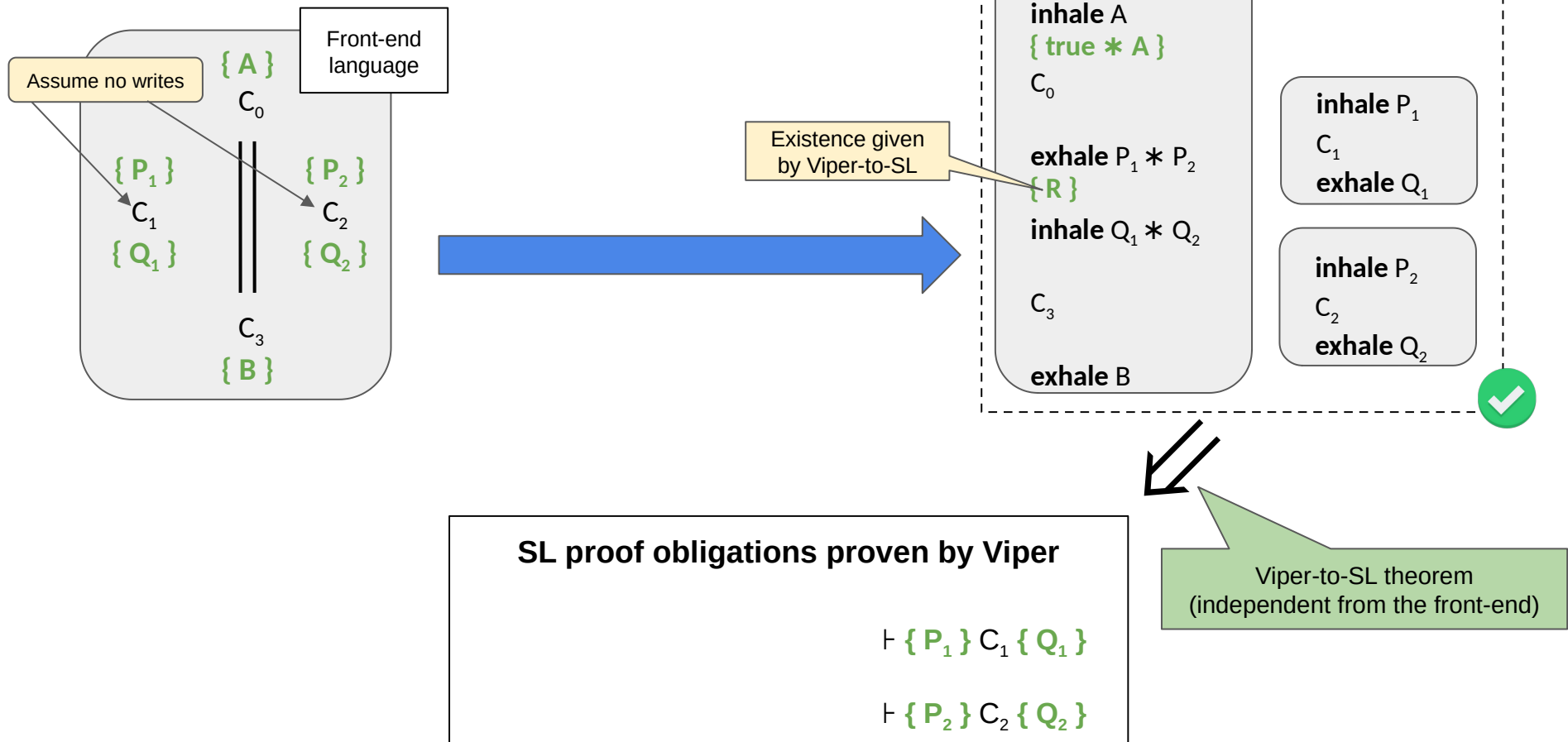
Viper-to-SL: Parallel Program



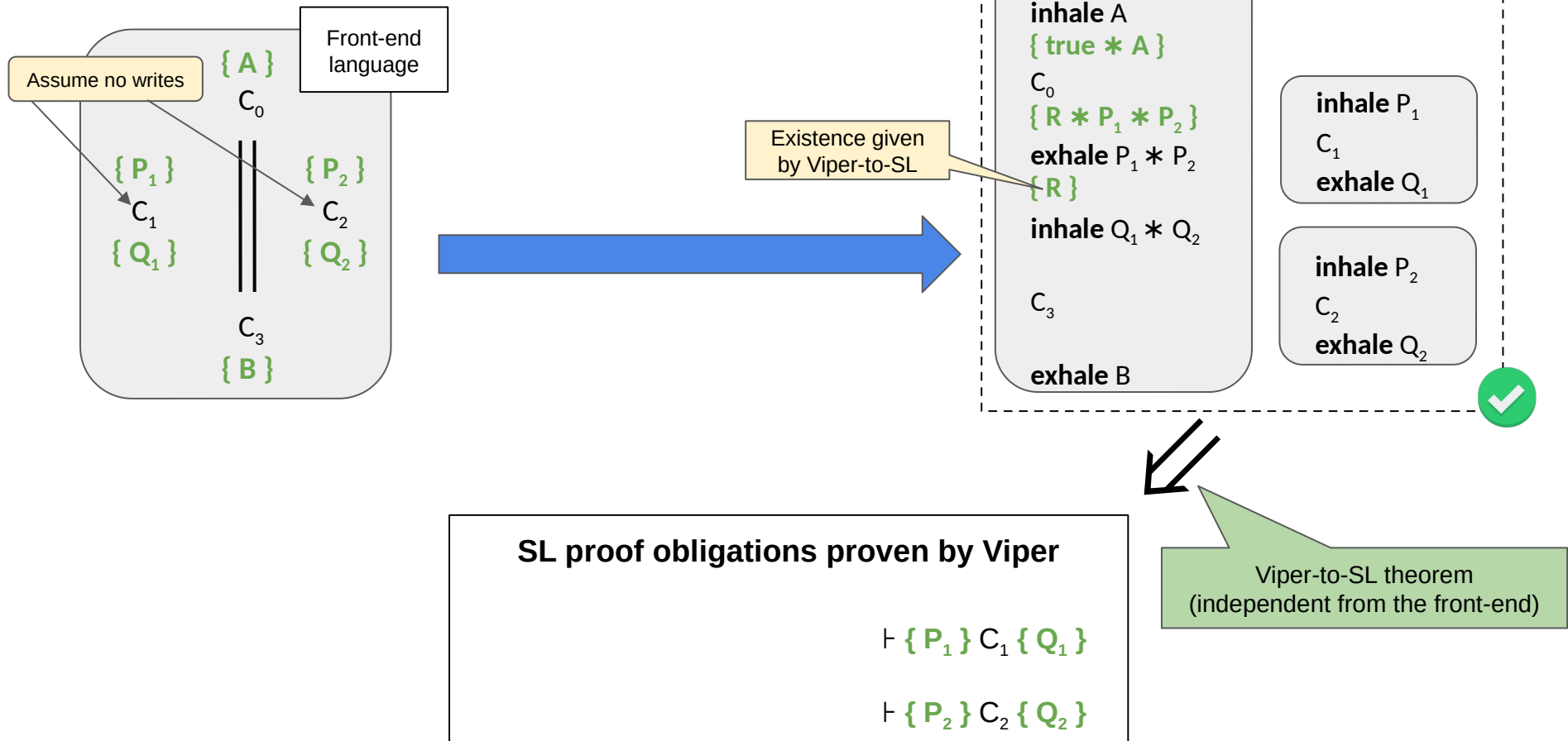
Viper-to-SL: Parallel Program



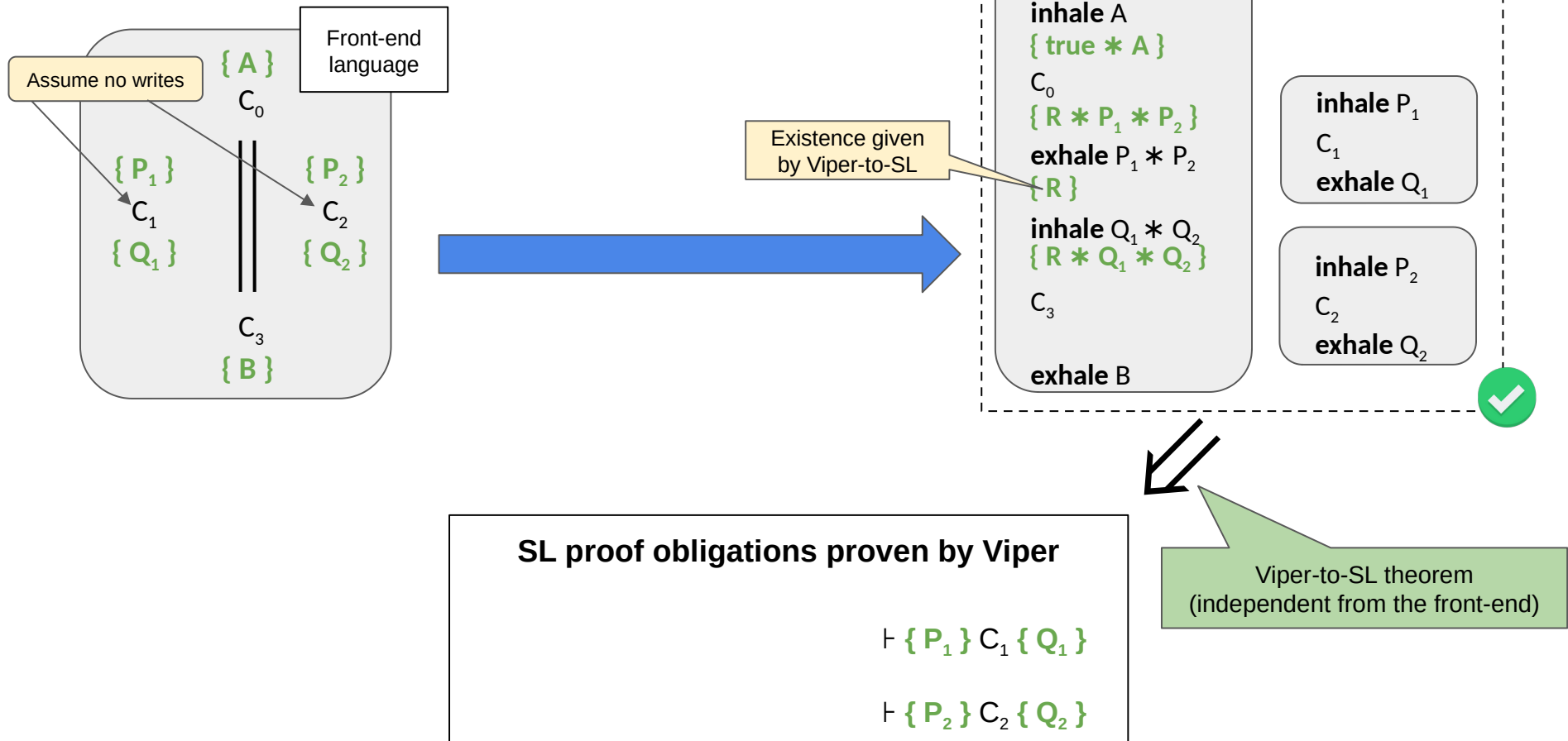
Viper-to-SL: Parallel Program



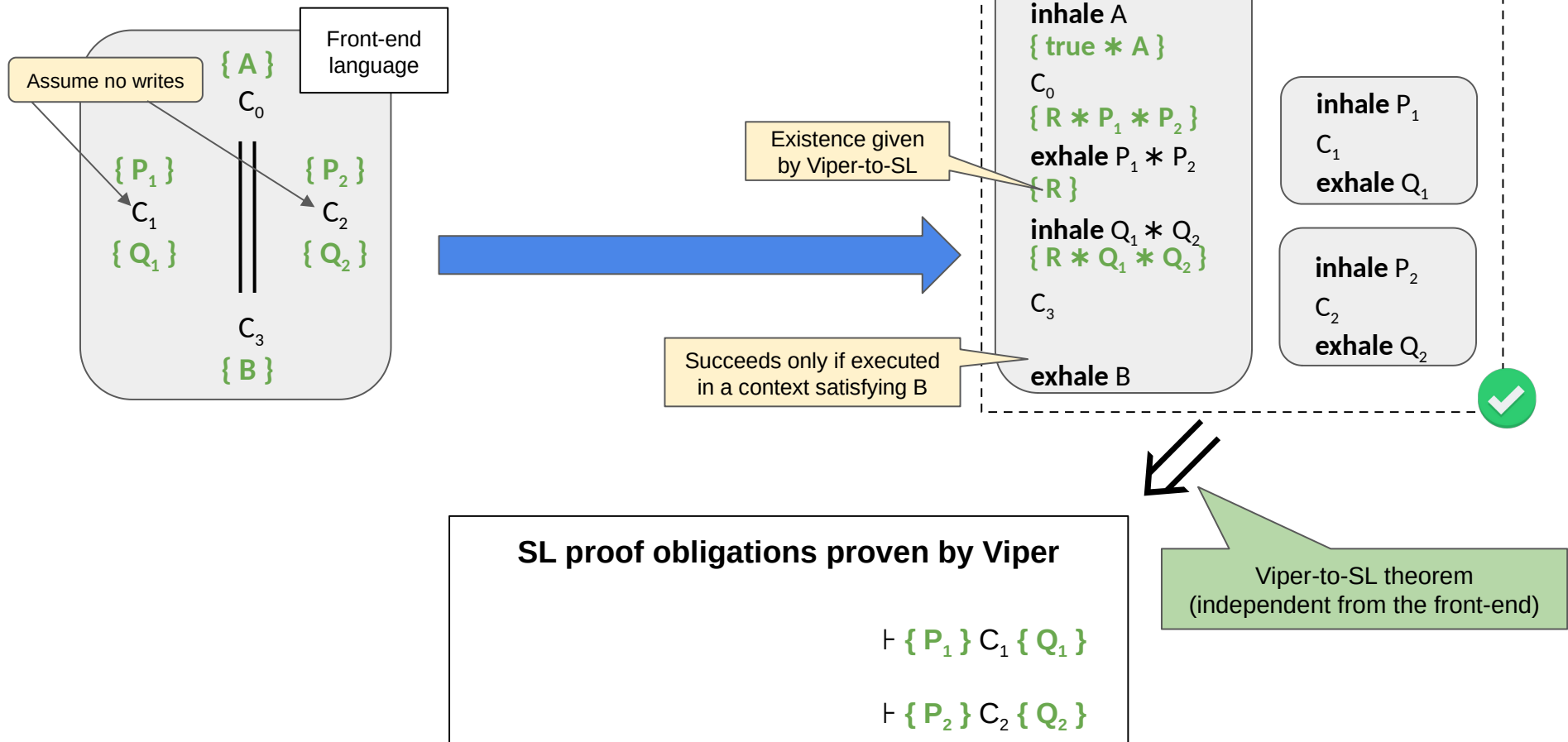
Viper-to-SL: Parallel Program



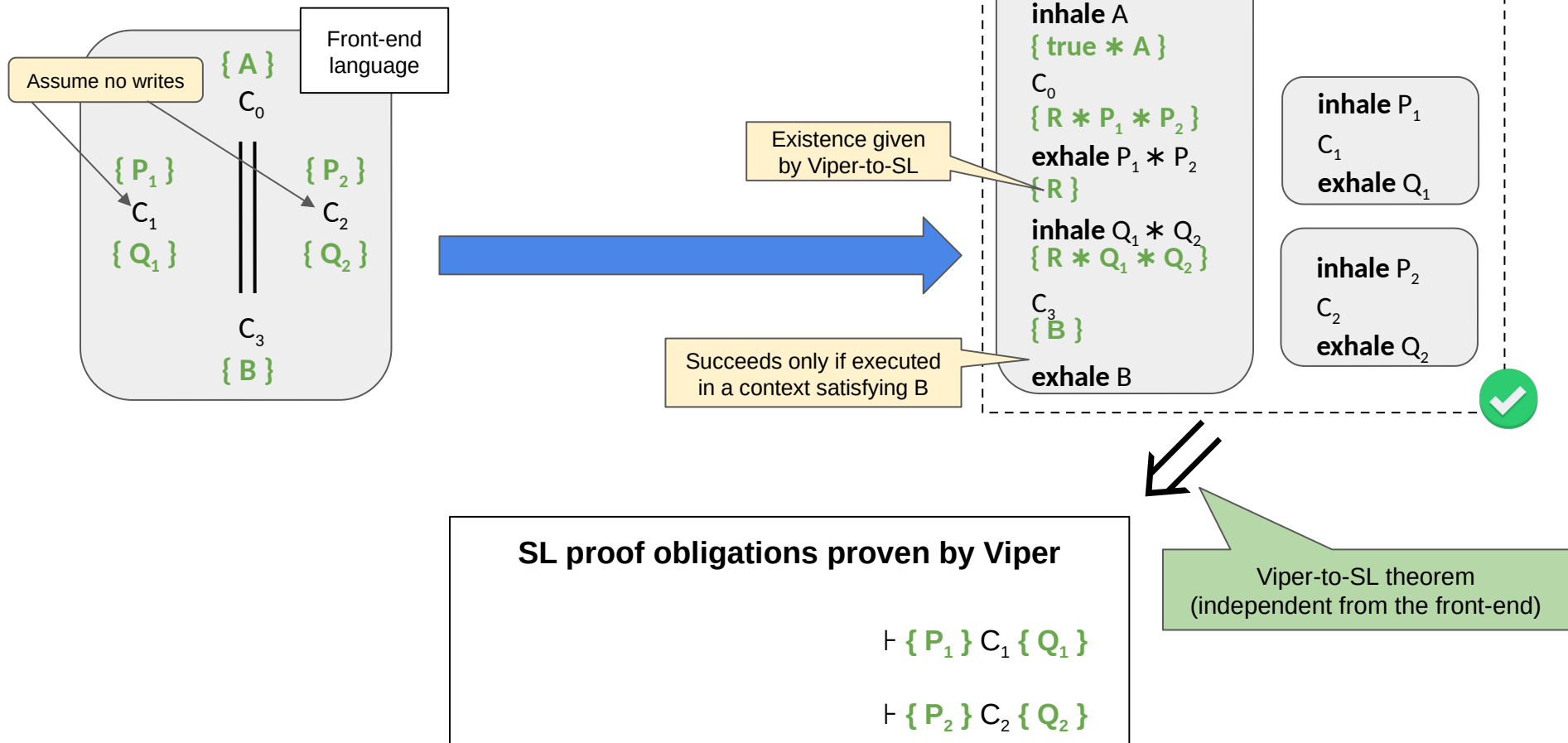
Viper-to-SL: Parallel Program



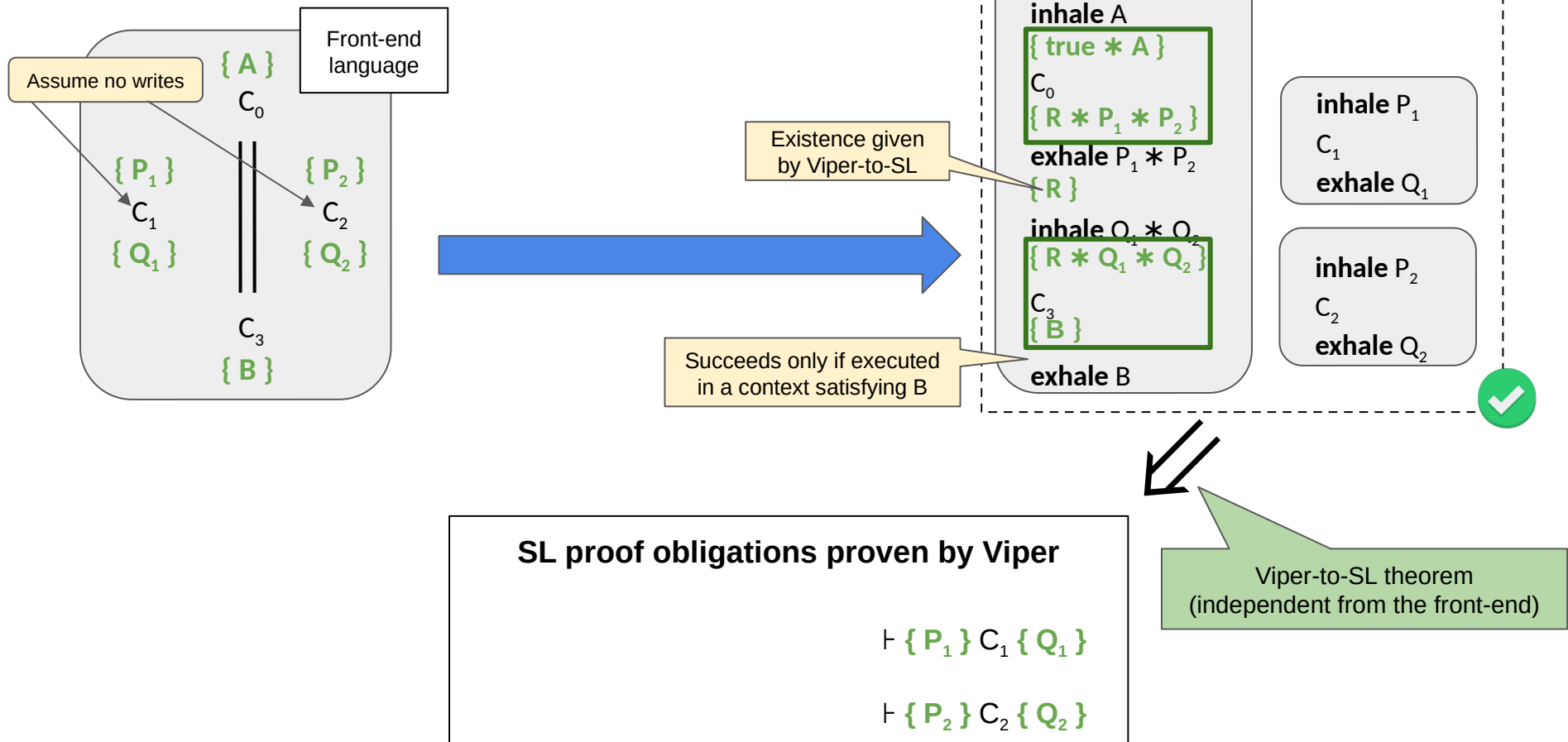
Viper-to-SL: Parallel Program



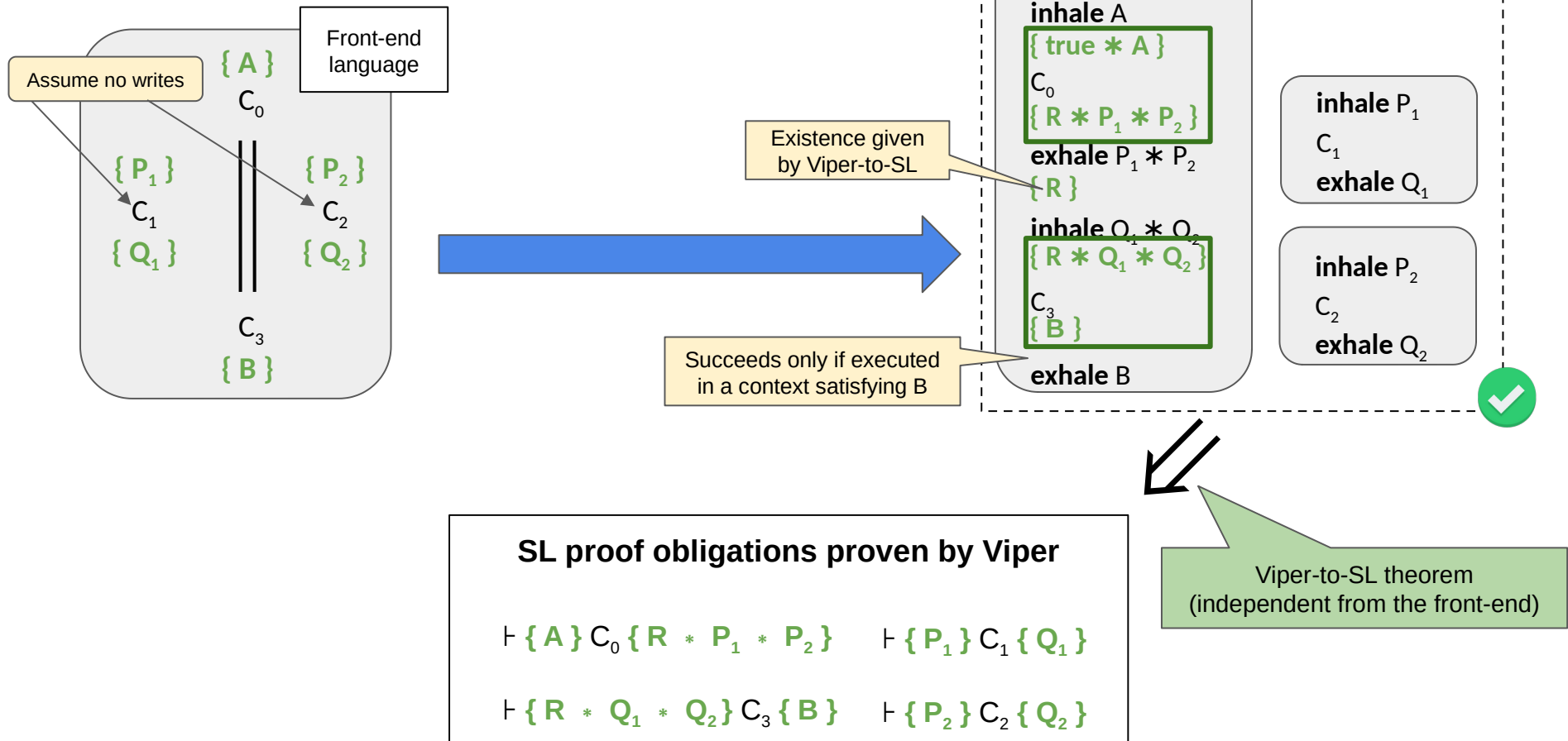
Viper-to-SL: Parallel Program



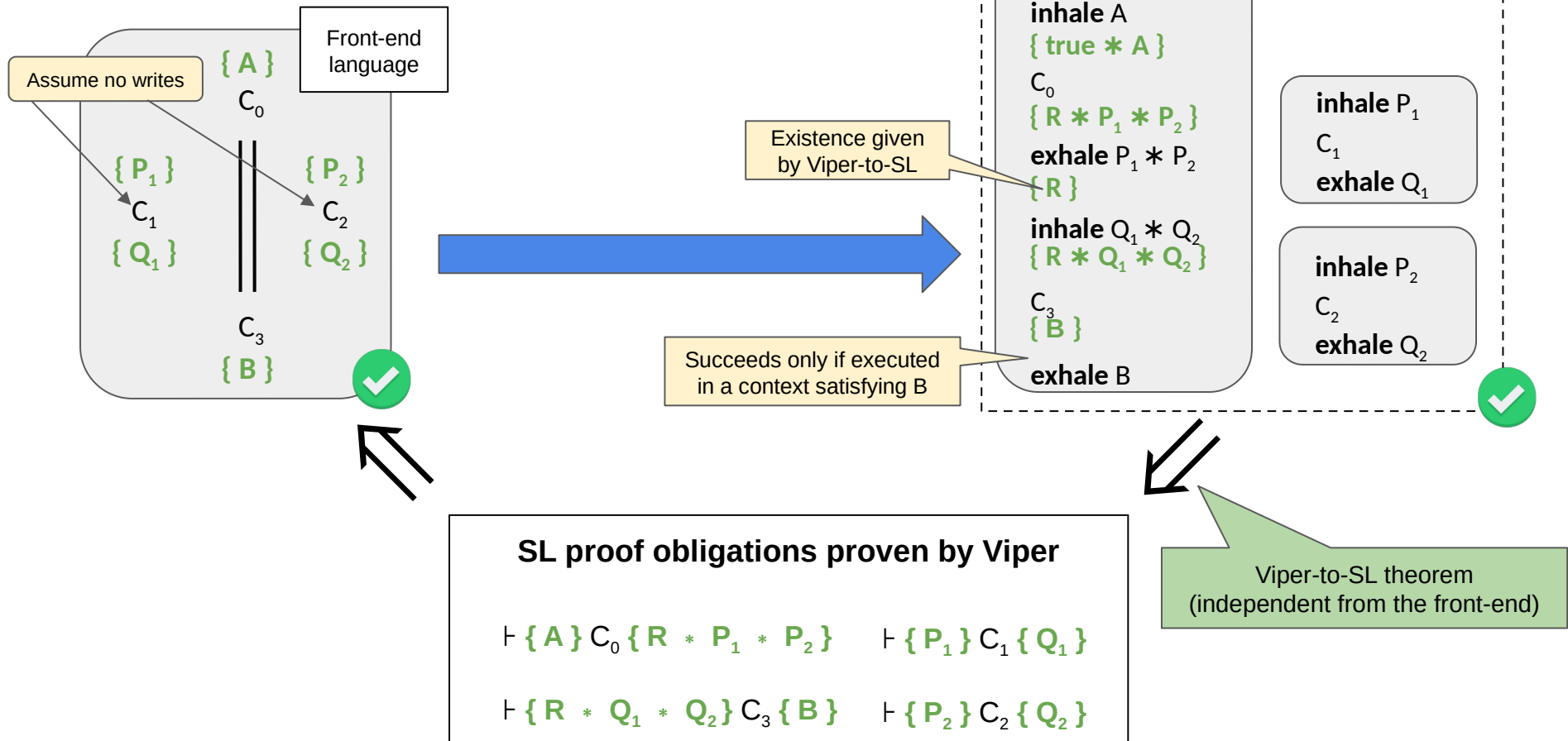
Viper-to-SL: Parallel Program



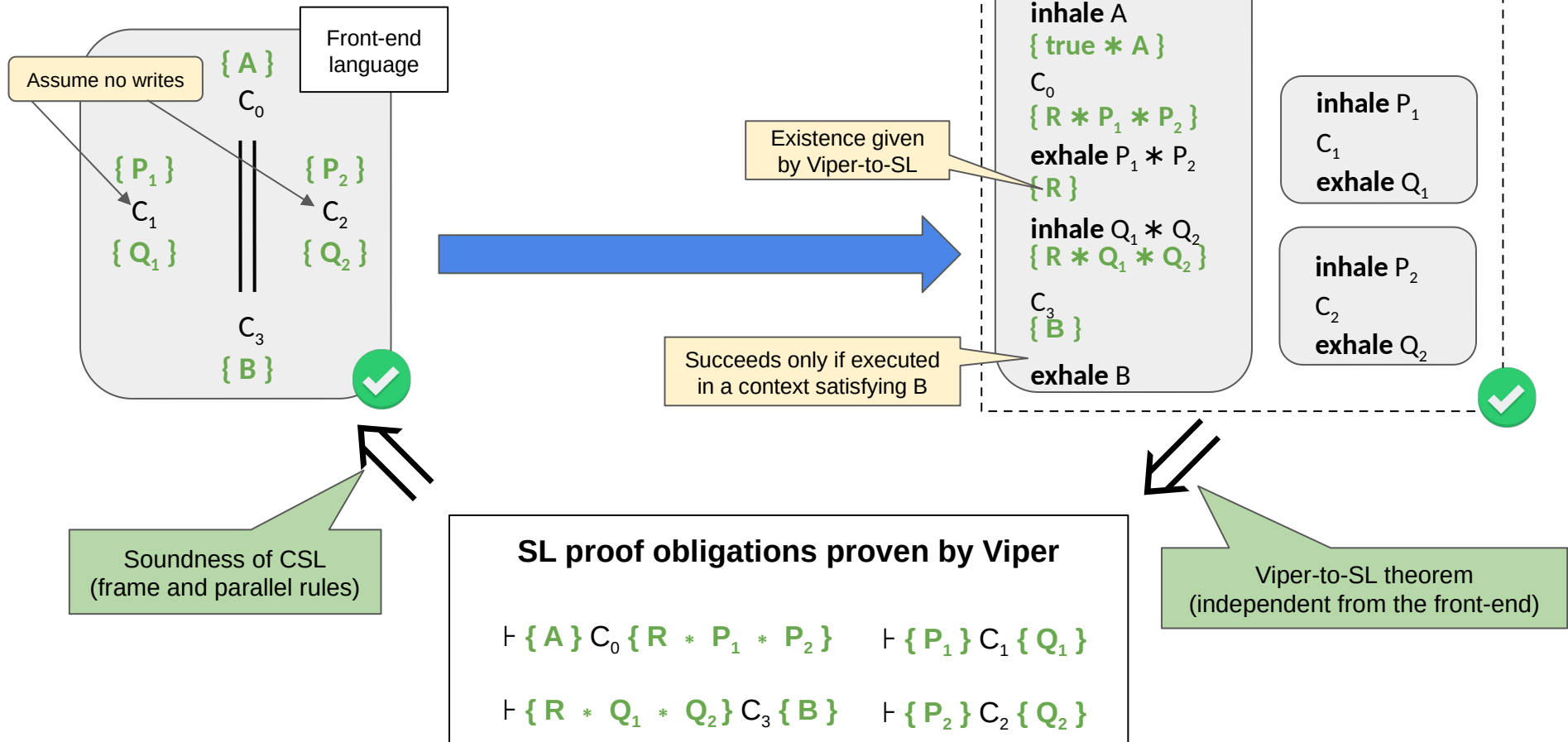
Viper-to-SL: Parallel Program



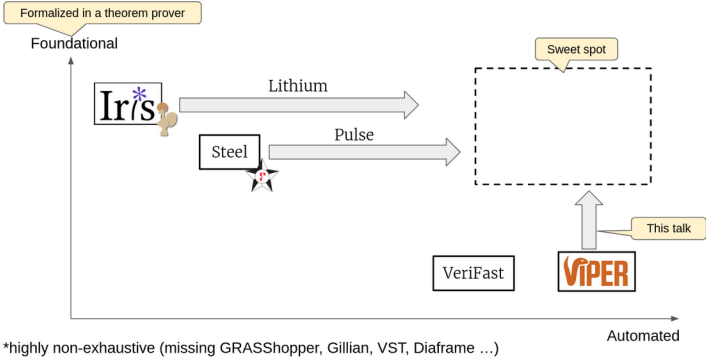
Viper-to-SL: Parallel Program



Viper-to-SL: Parallel Program

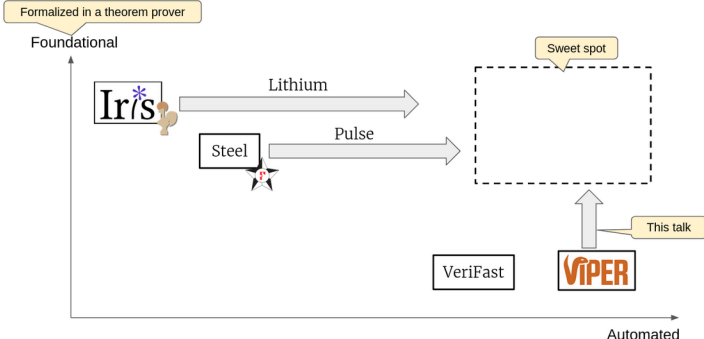


Program Verifiers Based on Separation Logic (SL)



*highly non-exhaustive (missing GRASShopper, Gillian, VST, Diaframe ...)

Program Verifiers Based on Separation Logic (SL)



*highly non-exhaustive (missing GRASShopper, Gillian, VST, Diaframe ...)

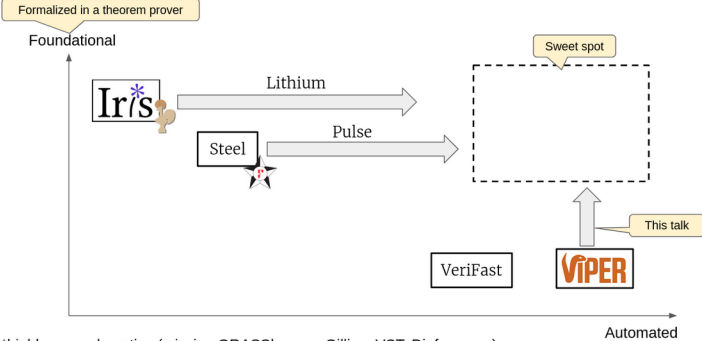
2

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A -* Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally	- All resources required by A are obtained - All logical constraints are assumed	- All resources required by A are removed - All logical constraints are asserted
Separation logic analogue of	assume A	assert A

10

Program Verifiers Based on Separation Logic (SL)



*highly non-exhaustive (missing GRASShopper, Gillian, VST, Diaframe ...)

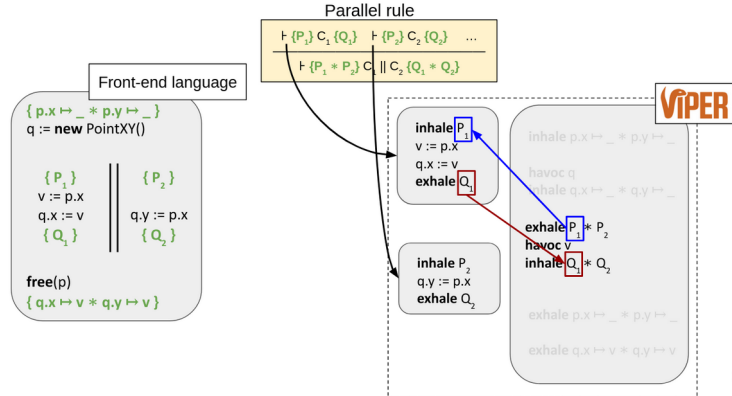
2

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally	- All resources required by A are obtained - All logical constraints are assumed	- All resources required by A are removed - All logical constraints are asserted
Separation logic analogue of	assume A	assert A

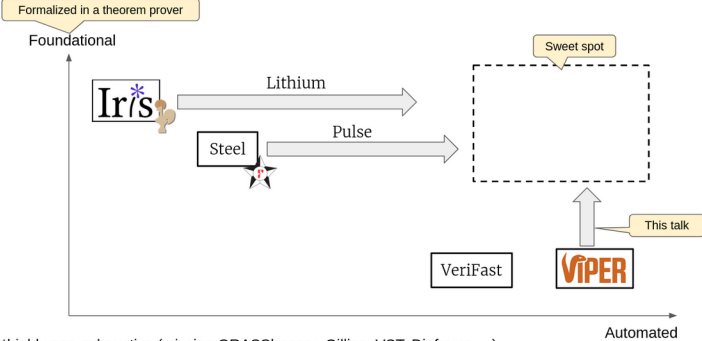
10

Example: Verifying a Parallel Program (2/2)



12

Program Verifiers Based on Separation Logic (SL)



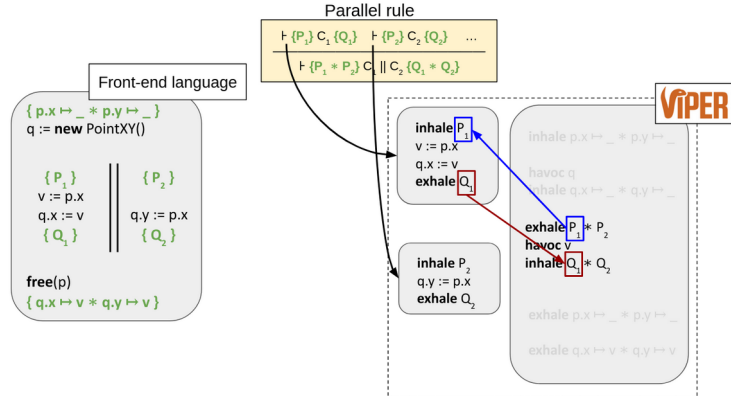
2

Verification Primitives: Inhale and Exhale

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{inhale } A \{P * A\}$ weakestPre(inhale A, Q) = $A * Q$	$\vdash \{P * A\} \text{exhale } A \{P\}$ weakestPre(exhale A, Q) = $A * Q$
Operationally	- All resources required by A are obtained - All logical constraints are assumed	- All resources required by A are removed - All logical constraints are asserted
Separation logic analogue of	assume A	assert A

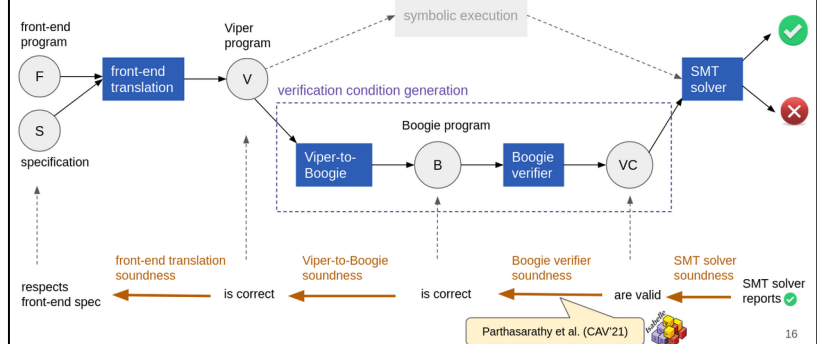
10

Example: Verifying a Parallel Program (2/2)



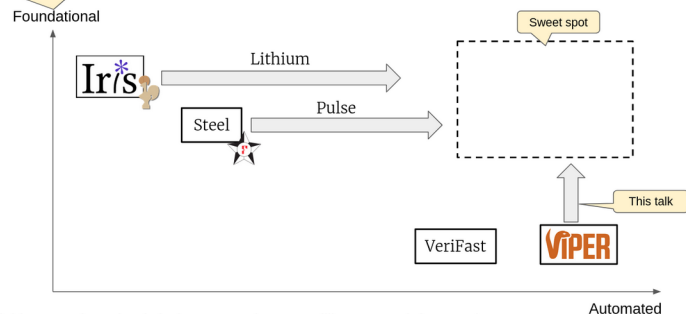
12

Soundness: Proof Strategy



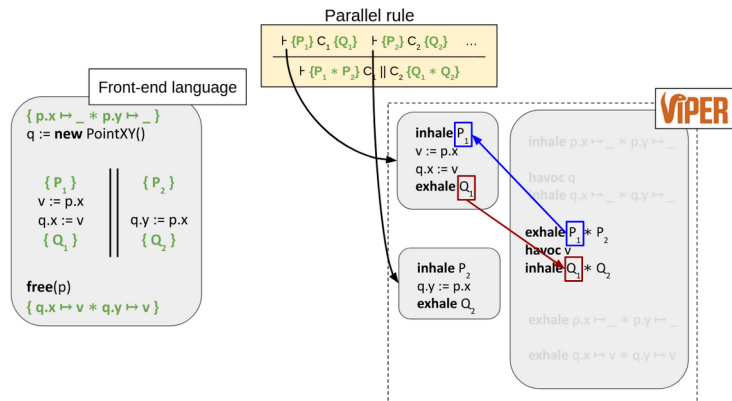
16

Formalized in a theorem prover



*highly non-exhaustive (missing GRASShopper, Gillian, VST, Diaframe ...)

12



10

	inhale A	exhale A
Intuitive meaning	Adds resources specified by A to the current context	Removes resources specified by A from the current context
Logically	$\vdash \{P\} \text{ inhale } A \{P * A\}$ $\text{weakestPre}(\text{inhale } A, Q) = A * Q$	$\vdash \{P * A\} \text{ exhale } A \{P\}$ $\text{weakestPre}(\text{exhale } A, Q) = A * Q$
Operationally	- All resources required by A are obtained - All logical constraints are assumed	- All resources required by A are removed - All logical constraints are asserted
Separation logic analogue of	assume A	assert A

16

