Temporal Deductive Verification of Parametrized Systems

César Sánchez

IMDEA Software Institute

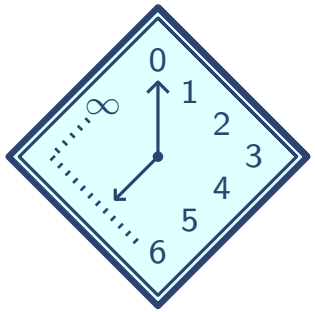
Joint work with Alejandro Sanchez

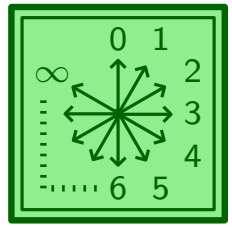
2nd Int'l Workshop on Parametrized Verification Madrid, 4 Sep, 2015

The Problem

The Problem

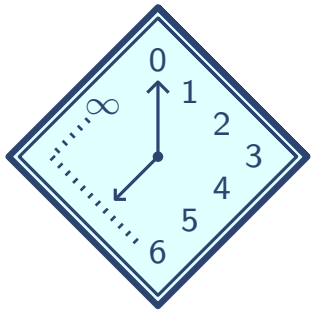
Temporal Properties

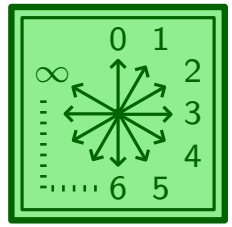




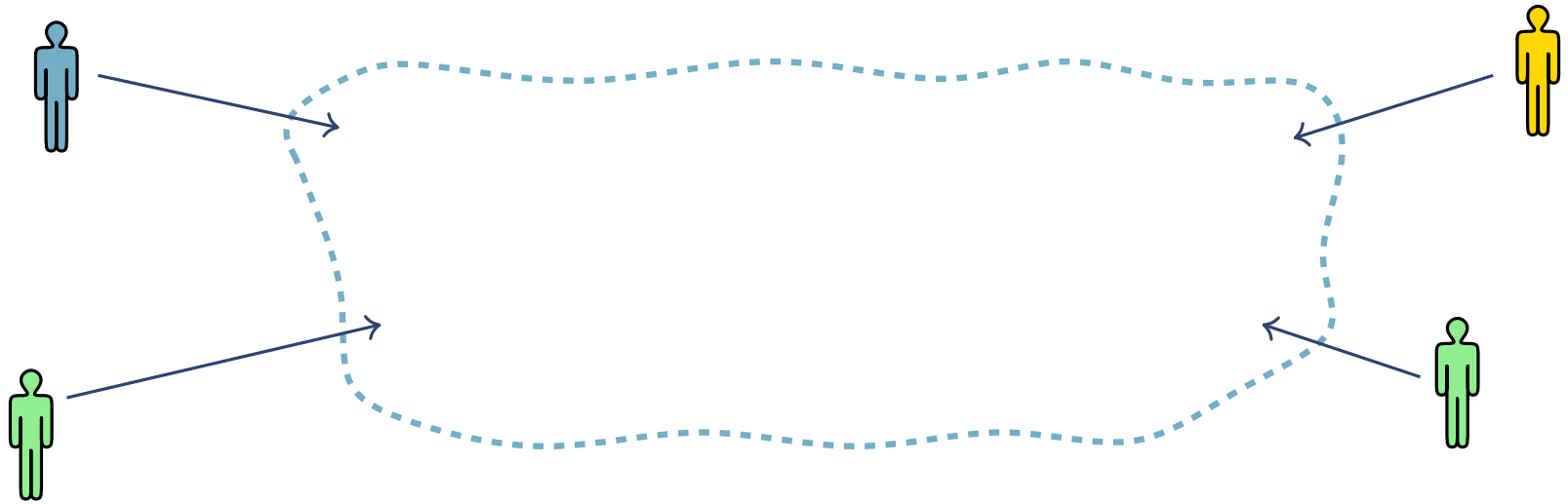
The Problem

Temporal Properties



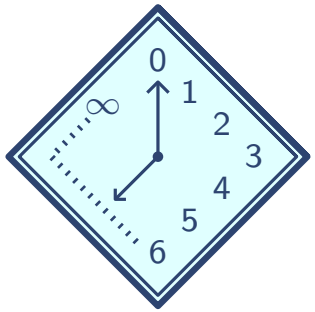


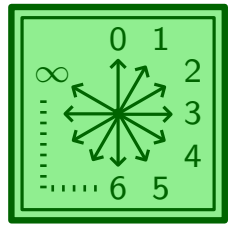
Concurrent Datatypes



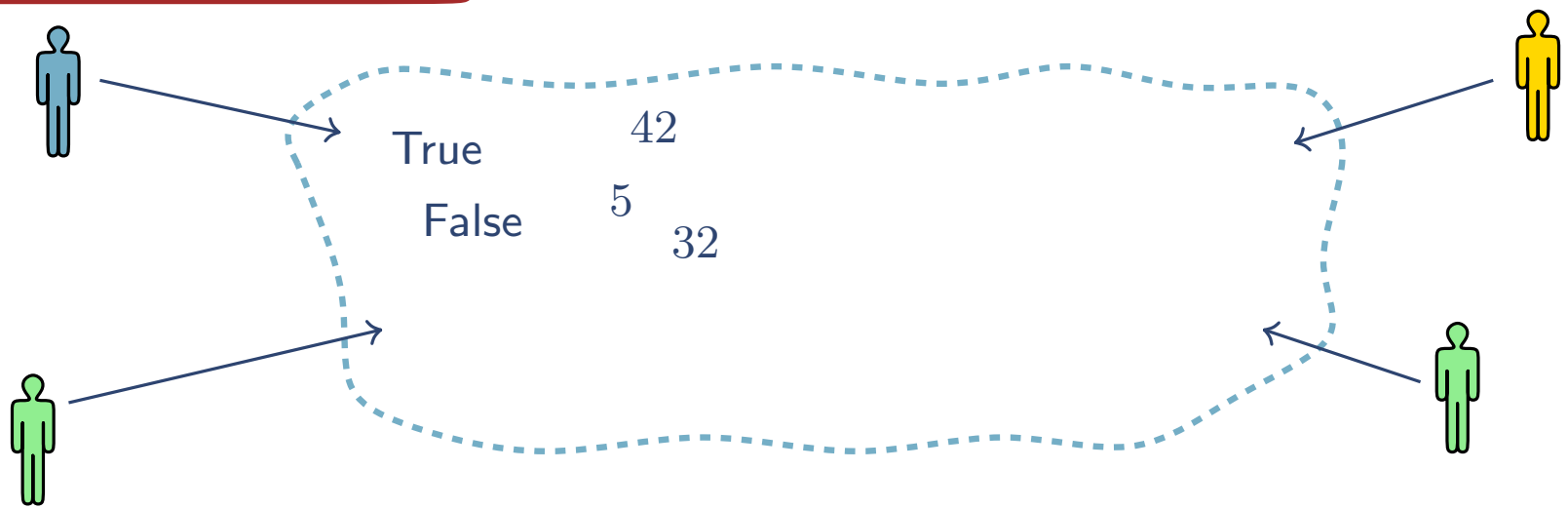
The Problem

Temporal Properties



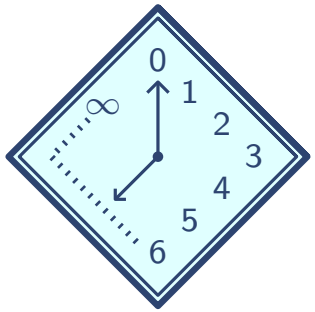


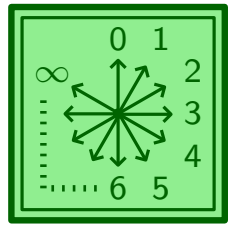
Concurrent Datatypes



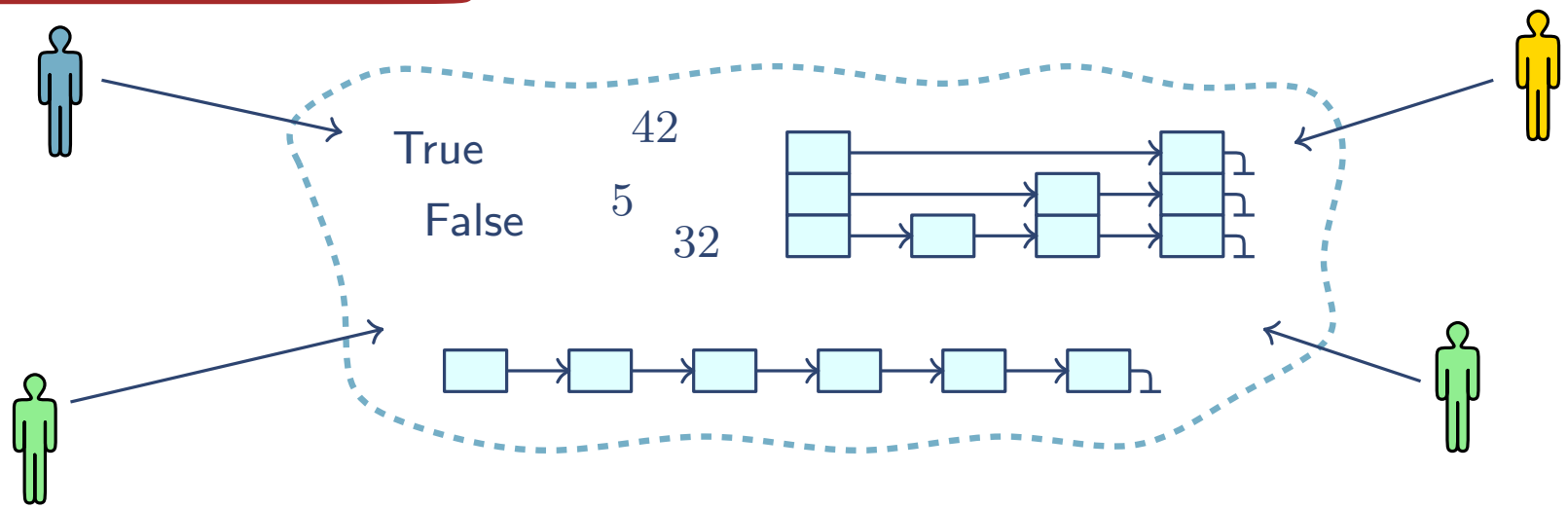
The Problem

Temporal Properties



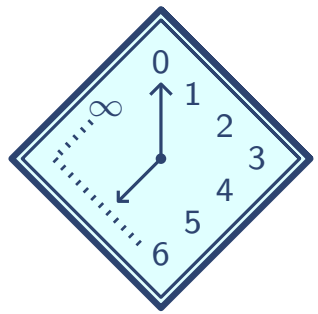


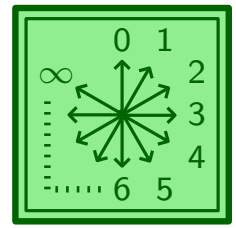
Concurrent Datatypes



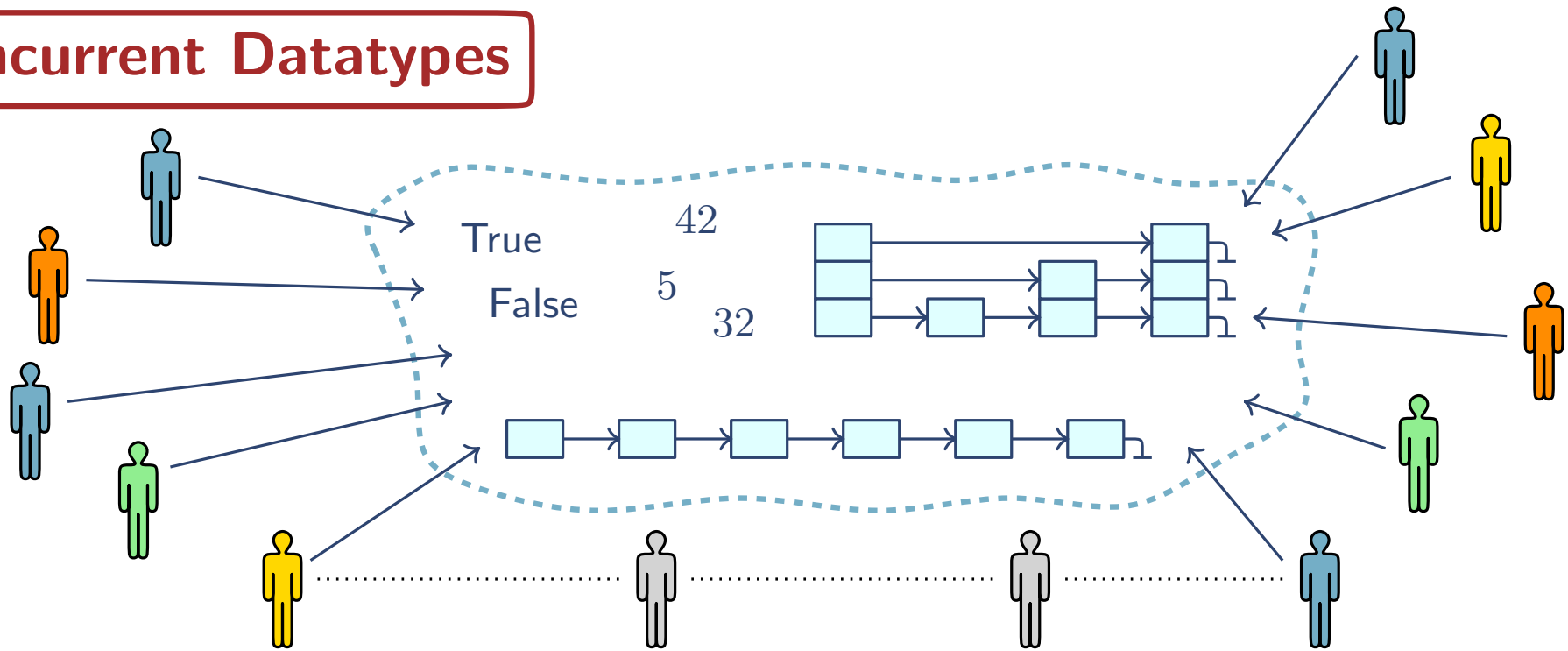
The Problem

Temporal Properties





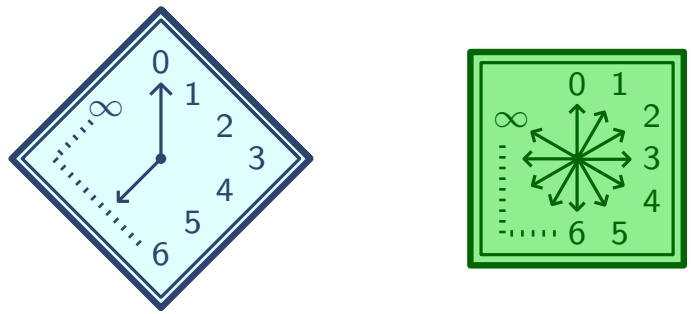
Concurrent Datatypes



Parametrized Verification

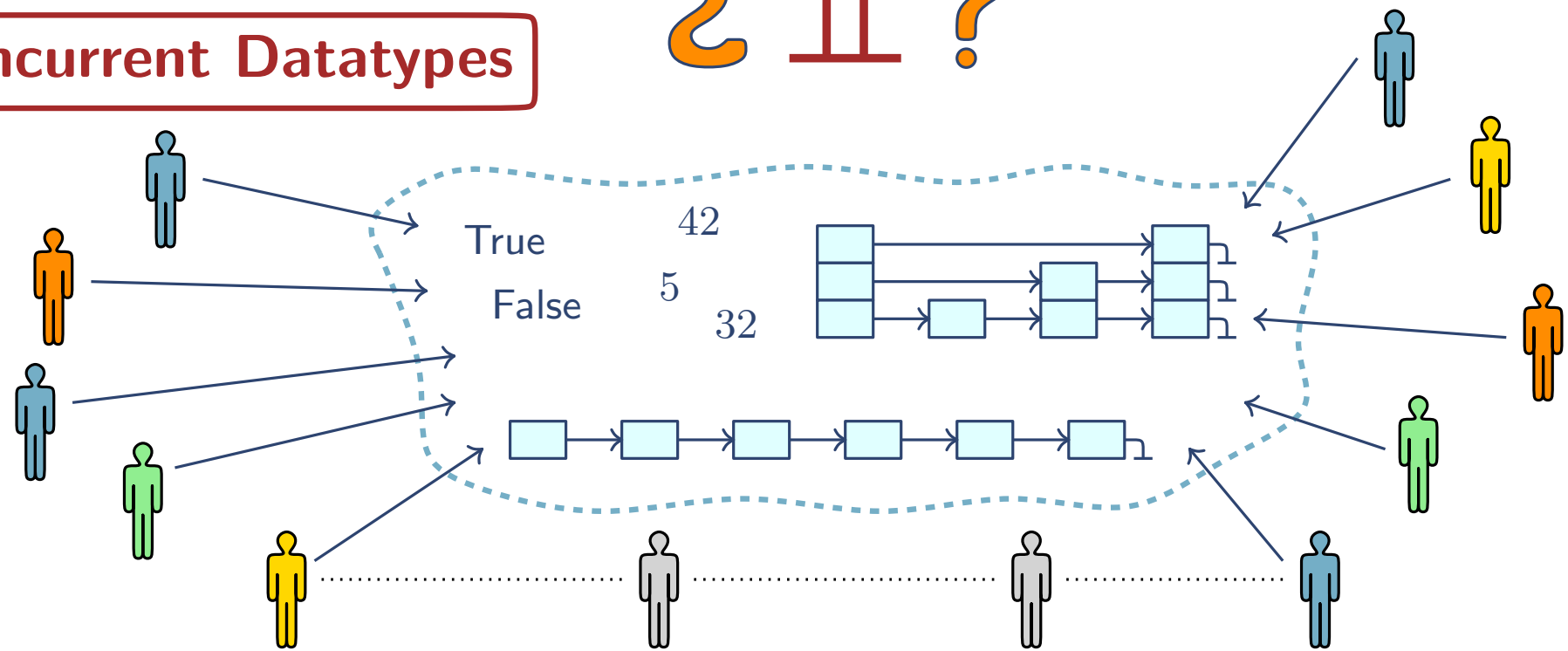
The Problem

Temporal Properties



¿ ⫫ ?

Concurrent Datatypes

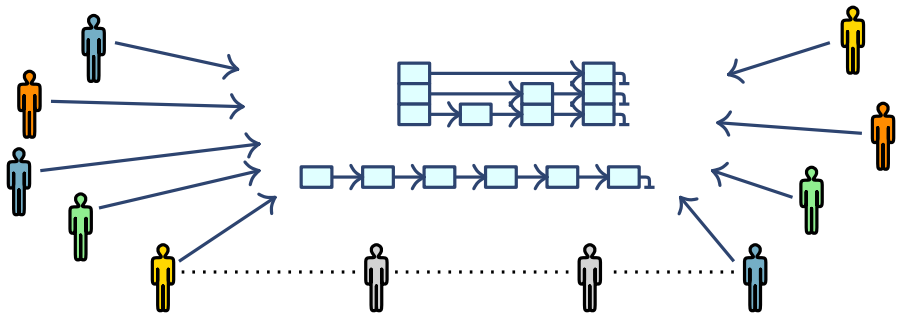


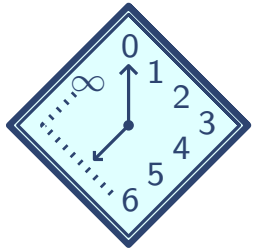
Parametrized Verification

Motivation

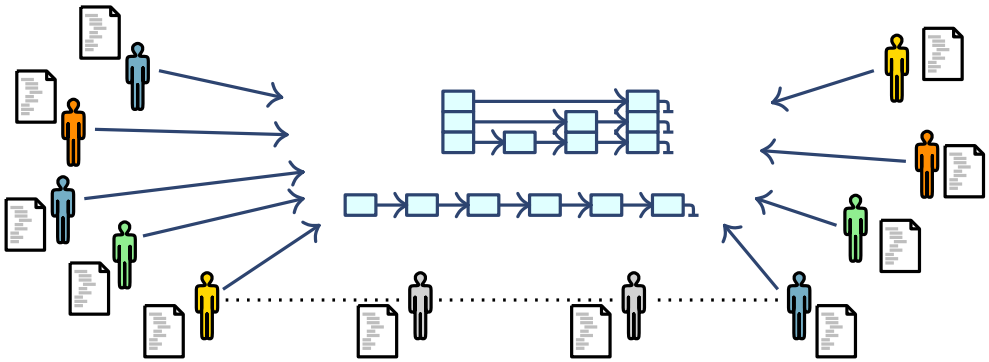
- Goal: **Formal Verification** (proving system correct)
- First target: concurrent data structures
- **Complex data types**: lists, trees, skiplists, hash-maps, etc
- **Unstructured fine-grained** synchronization methods
- **Automation** preferable but not mandatory
- Need to tackle **lock-based** and **lock-free** synchronization
- **Liveness** is as interesting as **safety** properties
- Parametrization to enable verification for **all system instances**

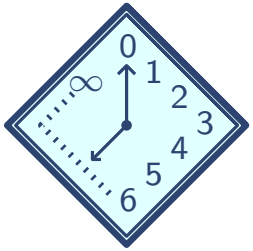
Our Approach



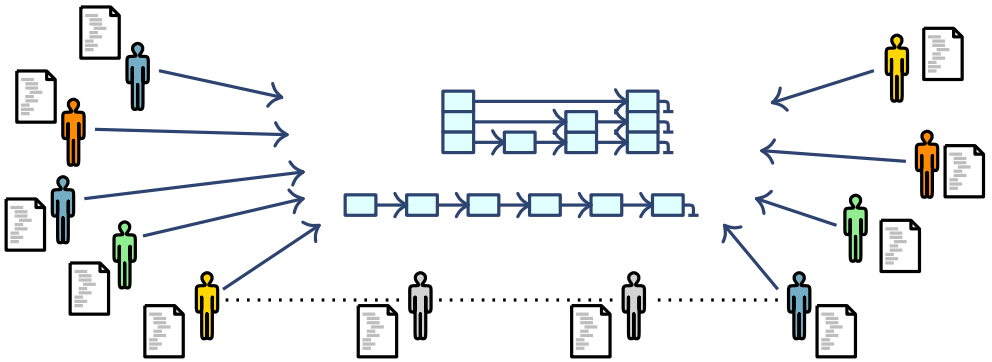


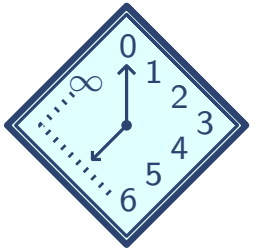
Our Approach





Our Approach



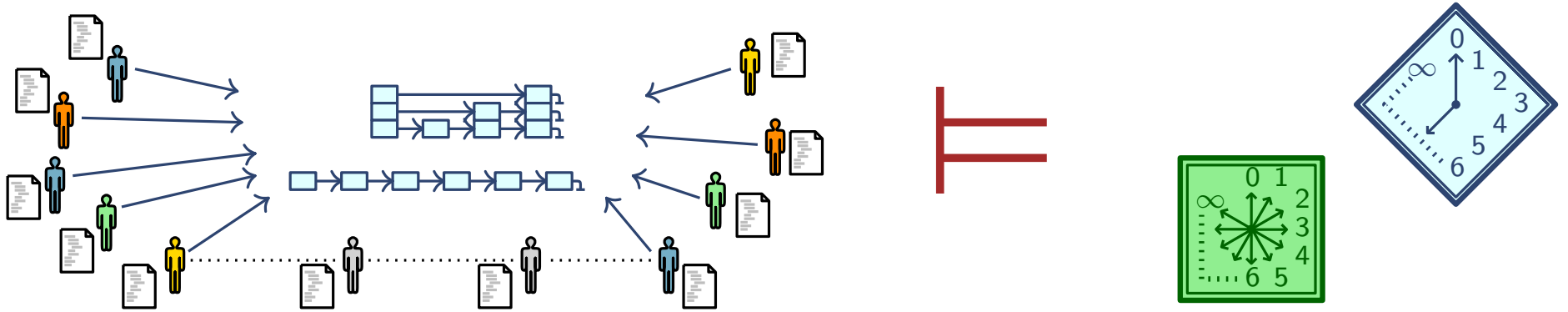


Safety

Liveness

Parametrized Deductive Methods

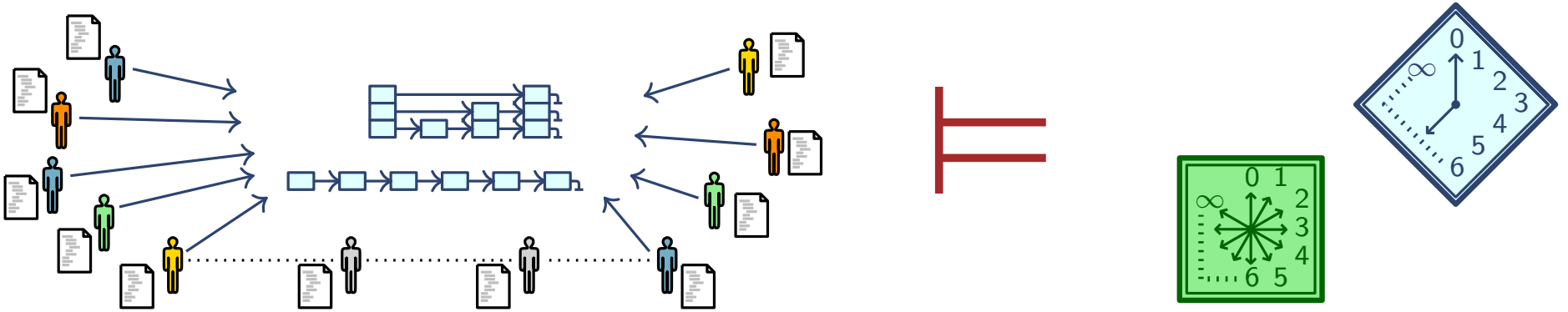
Our Approach



Parametrized Deductive Methods

Safety	Liveness
Parametrized Invariance	**Parametrized Verification Diagrams**

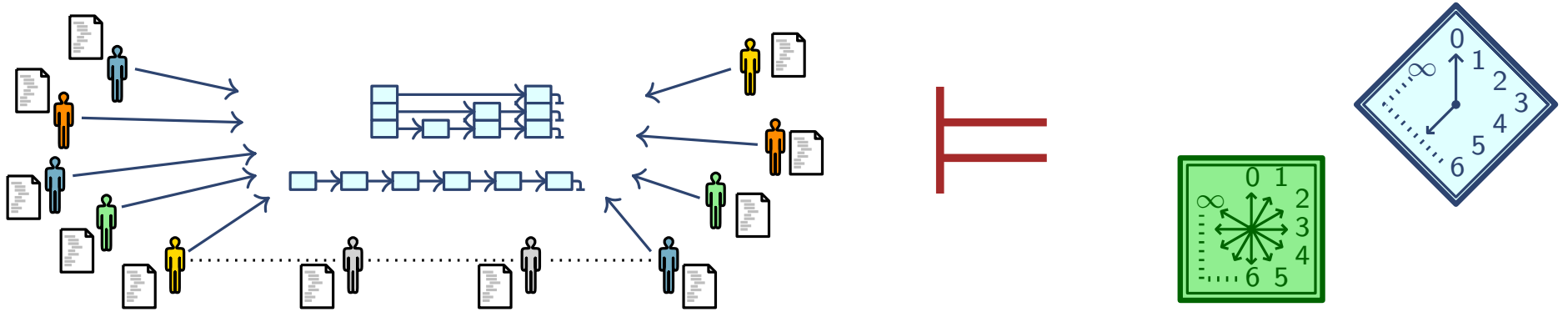
Our Approach



Parametrized Deductive Methods

Parametrized Invariant Generation	
Safety	Liveness
Parametrized Invariance	Parametrized Verification Diagrams

Our Approach



Parametrized Deductive Methods

Parametrized Invariant Generation

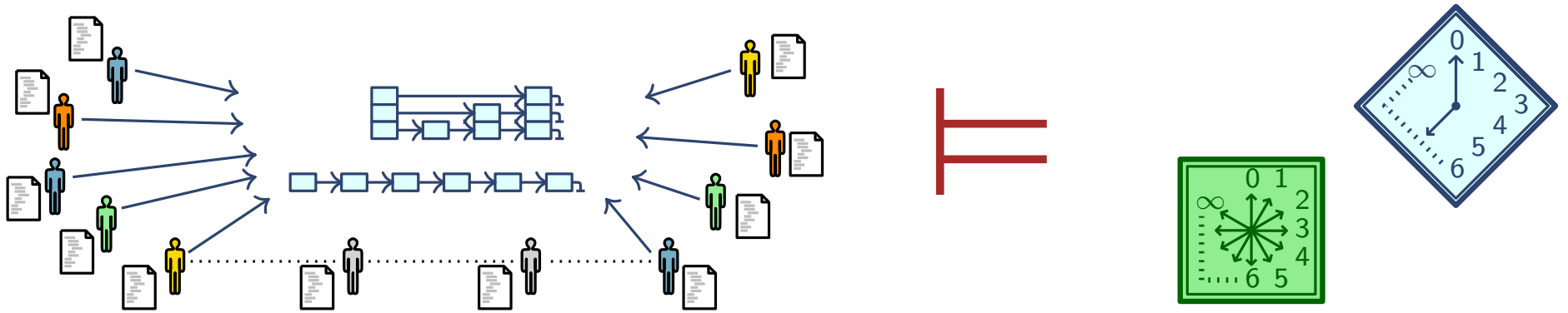
Safety	Liveness
Parametrized Invariance	**Parametrized Verification Diagrams**

Finite collection of VC

Decision Procedures

Locations	Numeric	Lists	Queues	Stacks	Skiplists

Our Approach



Parametrized Deductive Methods

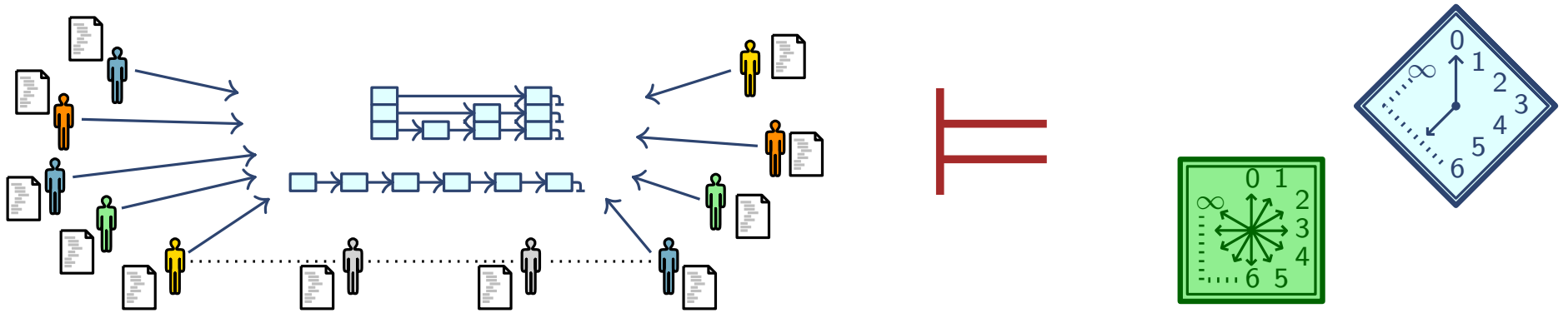
Parametrized Invariant Generation	
Safety	Liveness
Parametrized Invariance	Parametrized Verification Diagrams

Finite collection of VC

Decision Procedures

Locations	Numeric	Lists	Queues	Stacks	Skiplists
		TL3			TSL_K TSL

Our Approach



Parametrized Deductive Methods

Parametrized Invariant Generation

Safety	Liveness
Parametrized Invariance	**Parametrized Verification Diagrams**

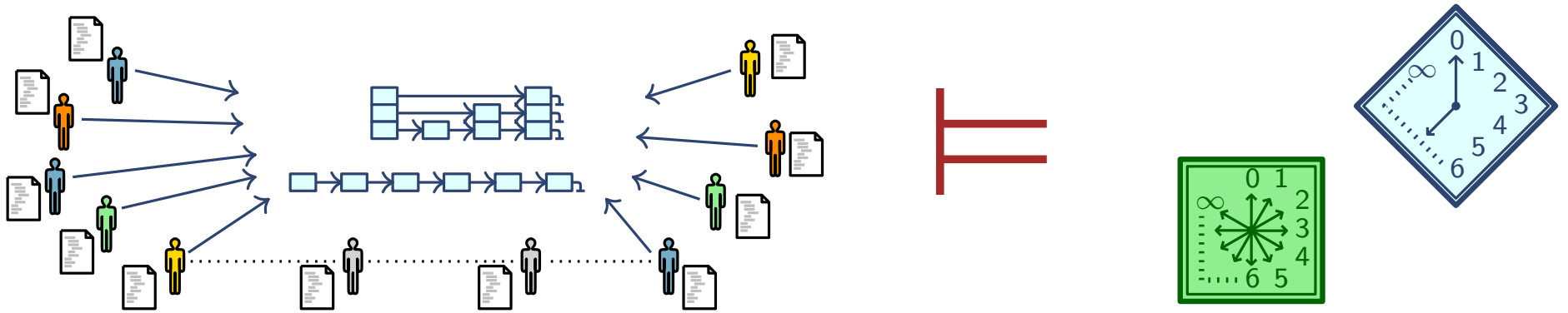
Finite collection of VC

Decision Procedures

Locations	Numeric	Lists	Queues	Stacks	Skiplists
		TL3			**TSL_K TSL**

SMT Solvers (Z3, Yices, CVC4,...)

Our Approach



Parametrized Deductive Methods

Parametrized Invariant Generation

Safety	Liveness
Parametrized Invariance	**Parametrized Verification Diagrams**

Finite collection of VC

Decision Procedures

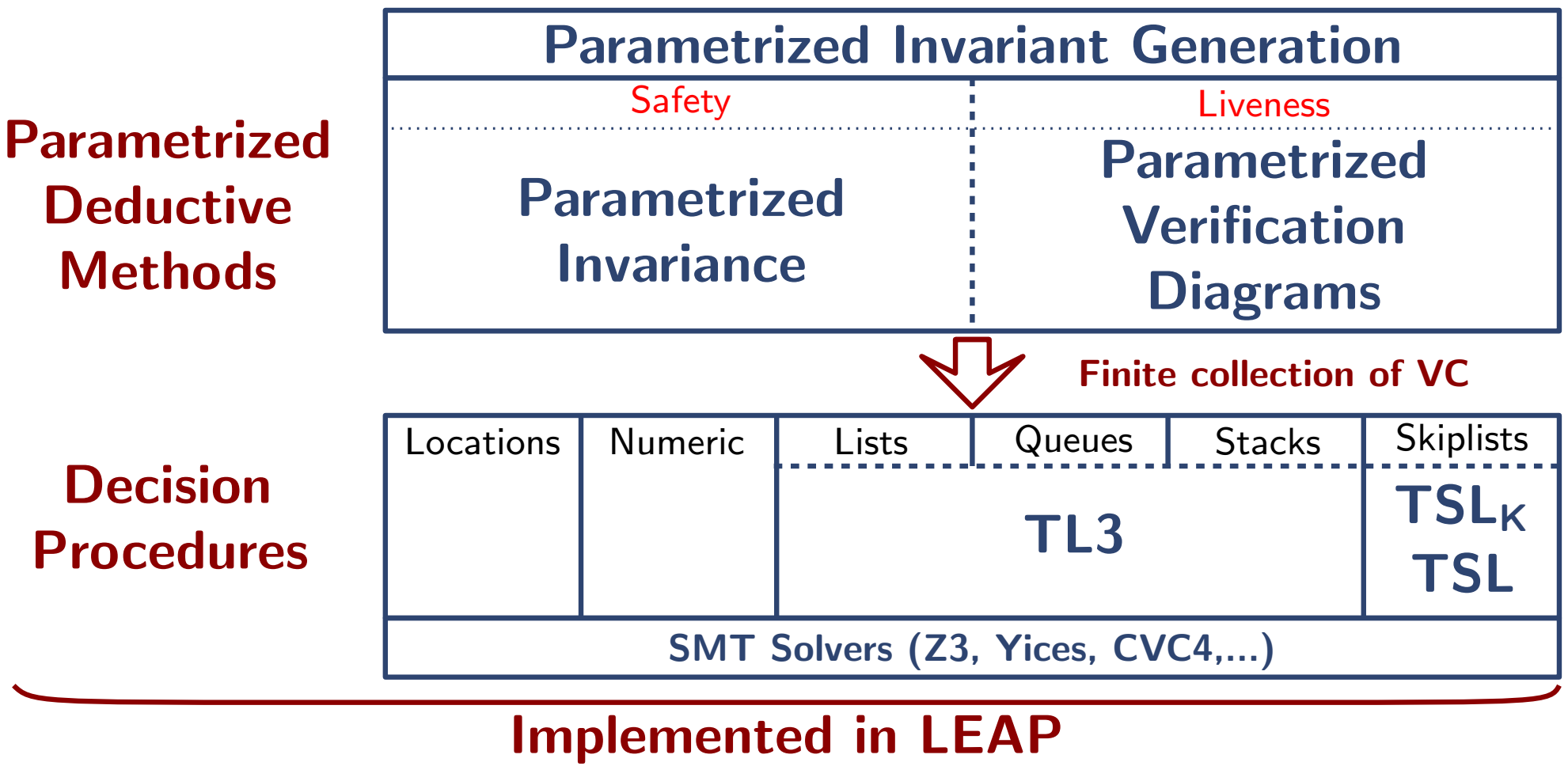
Locations	Numeric	Lists	Queues	Stacks	Skiplists
		TL3			**TSL_K TSL**

SMT Solvers (Z3, Yices, CVC4,...)

Implemented in LEAP

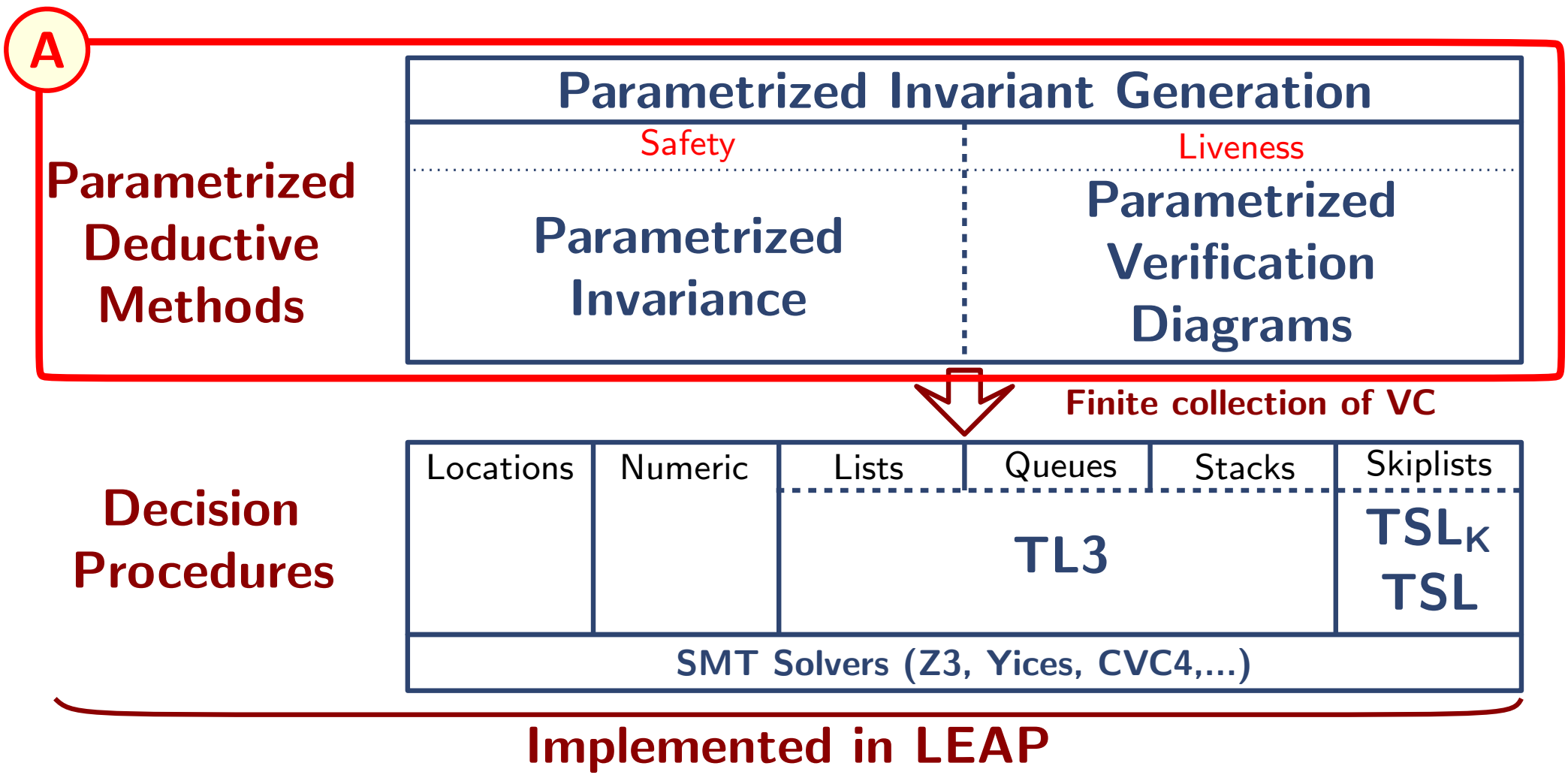
Our Contributions

A deductive verification framework for parametrized concurrent systems



Our Contributions

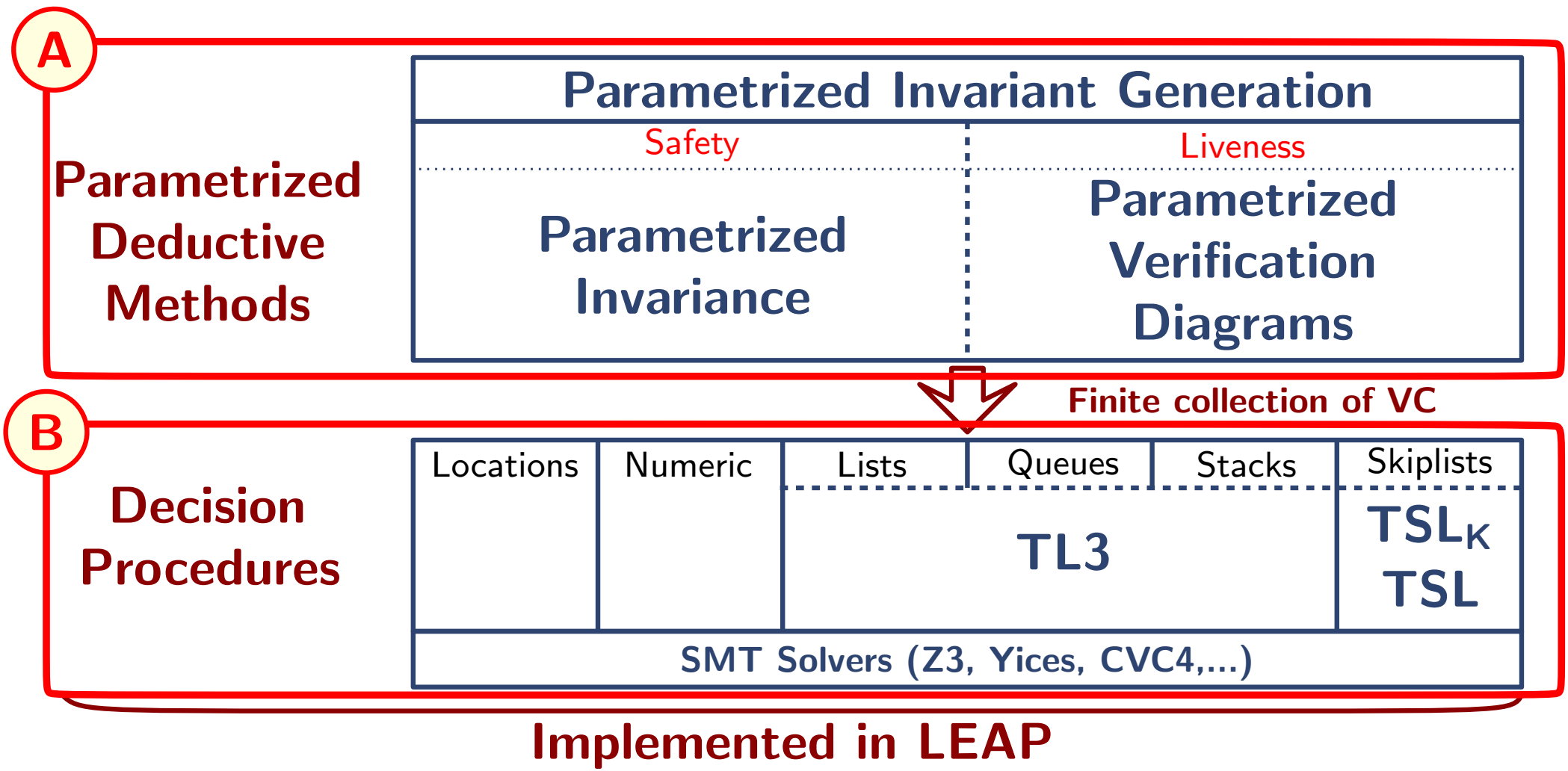
A Deductive Verification Techniques for Parametrized Systems



Our Contributions

A Deductive Verification Techniques for Parametrized Systems

B Decision Procedures for Complex Data Structures

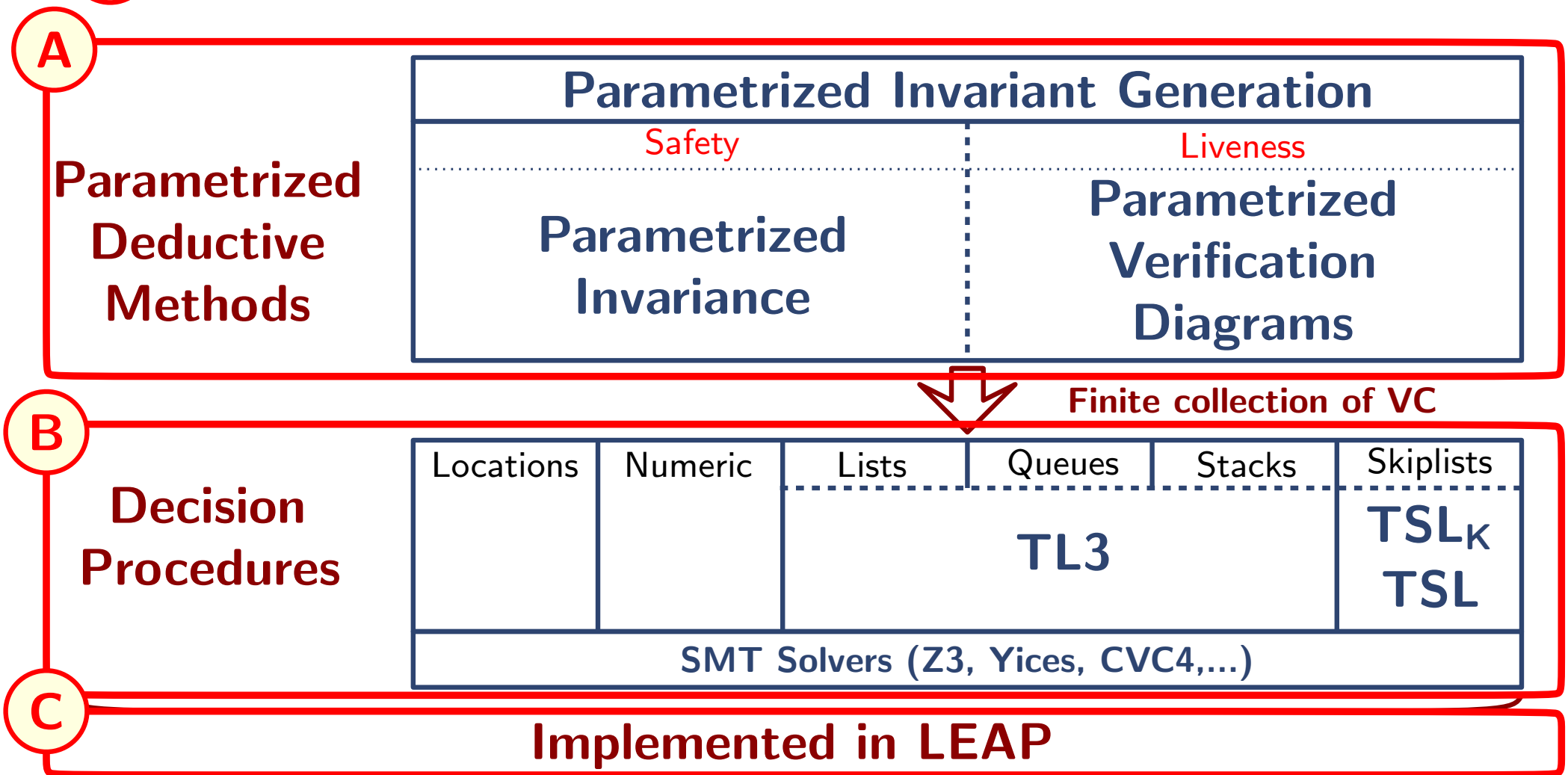


Our Contributions

A Deductive Verification Techniques for Parametrized Systems

B Decision Procedures for Complex Data Structures

C Implementation and Evaluation of our Framework



Our Contributions

A **Deductive Verification Techniques for Parametrized Systems**

B **Decision Procedures for Complex Data Structures**

C **Implementation and Evaluation of our Framework**

Our Contributions

A Deductive Verification Techniques for Parametrized Systems

1 Parametrized Invariance

Deductive proof rules for concurrent parametrized invariants

B Decision Procedures for Complex Data Structures

C Implementation and Evaluation of our Framework

Our Contributions

A Deductive Verification Techniques for Parametrized Systems

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

B Decision Procedures for Complex Data Structures

C Implementation and Evaluation of our Framework

Our Contributions

A Deductive Verification Techniques for Parametrized Systems

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

C Implementation and Evaluation of our Framework

Our Contributions

A Deductive Verification Techniques for Parametrized Systems

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

3 Invariant Generation using Self-reflection
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

4 TL3: A Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

5 TSL_K: A Decidable Family for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

6 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

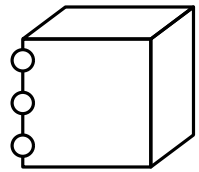
Verification of Concurrent Data-structures

Our verification approach

Verification of Concurrent Data-structures

Our verification approach

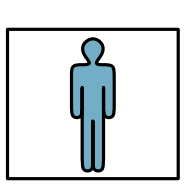
Concurrent DataStructure

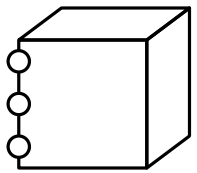


Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure

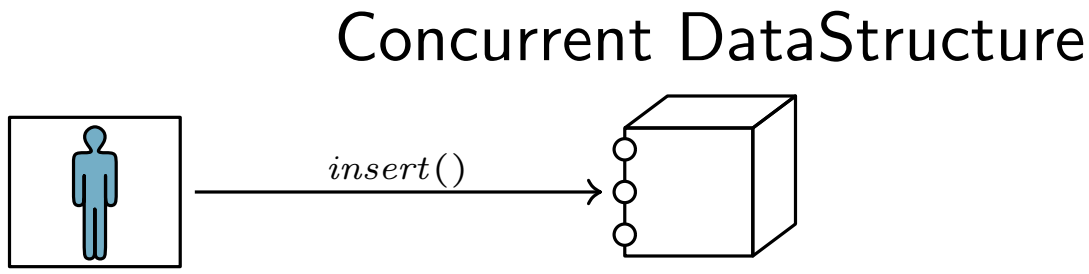




Most General Client

Verification of Concurrent Data-structures

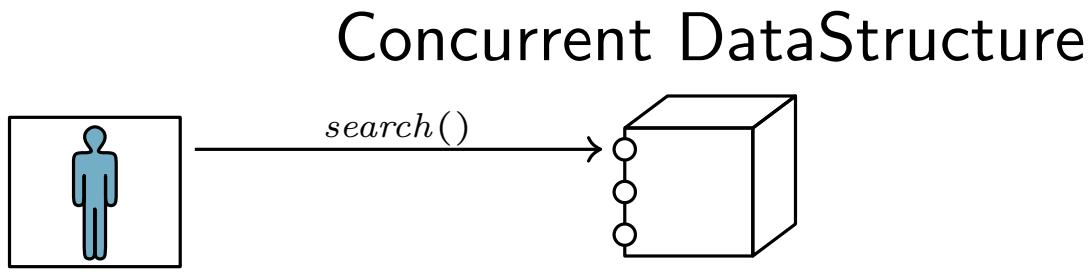
Our verification approach



Most General Client

Verification of Concurrent Data-structures

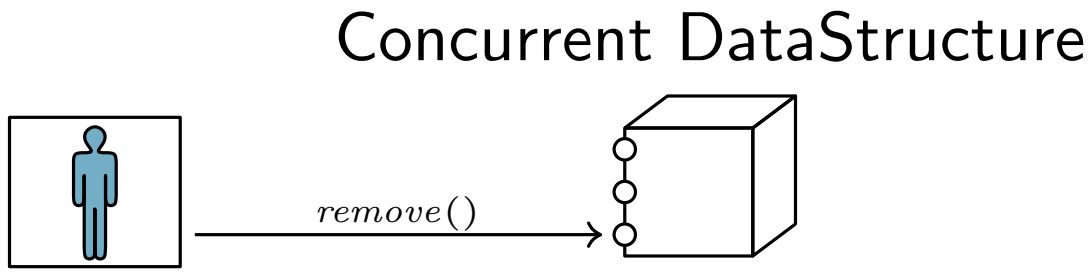
Our verification approach



Most General Client

Verification of Concurrent Data-structures

Our verification approach

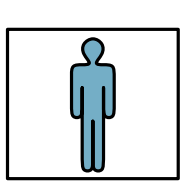


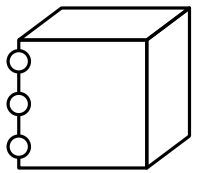
Most General Client

Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



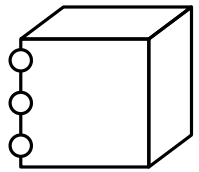


Most General Client

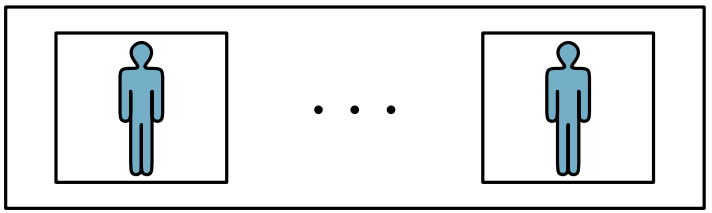
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



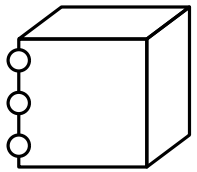
Most General Client



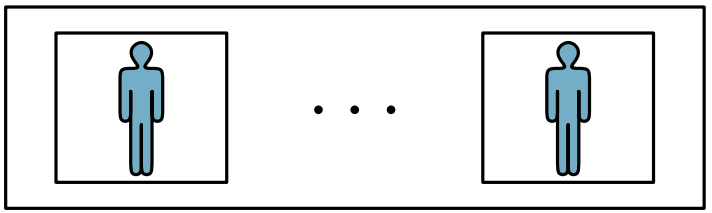
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client

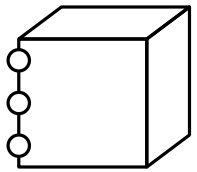


$$P[N] : P(1)||\cdots||P(N)$$

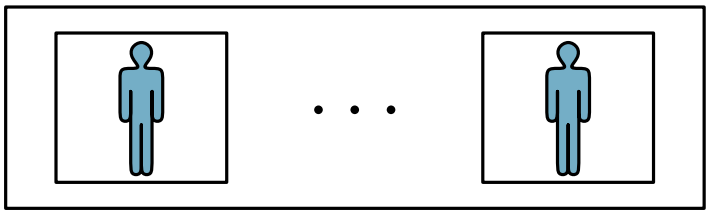
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)|| \cdots ||P(N)$

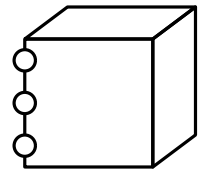
Property

φ

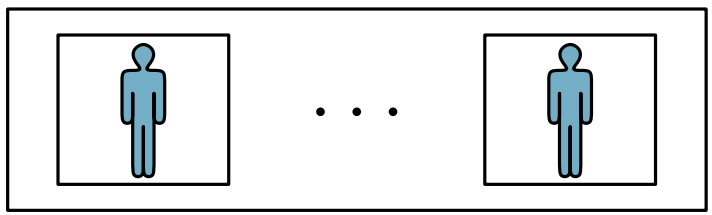
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure

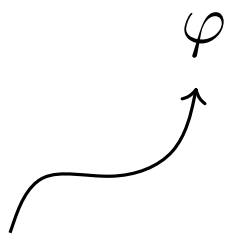


Most General Client



$P[N] : P(1)||\cdots||P(N)$

Property

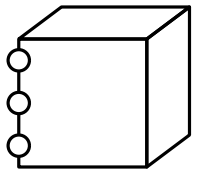


LTL ($\Box$,$\Diamond$,$\mathcal{U}$,. . .)

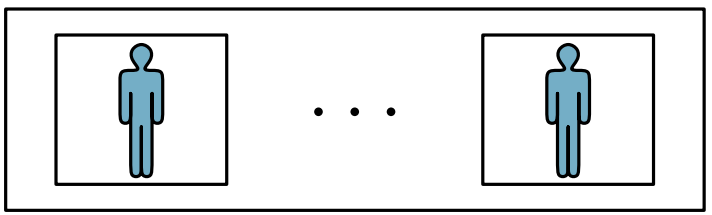
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)||\cdots||P(N)$

Safety
Property

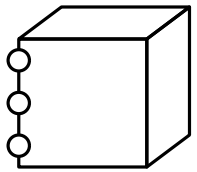
φ

LTL ($\Box$,$\Diamond$,$\mathcal{U}$,. . .)

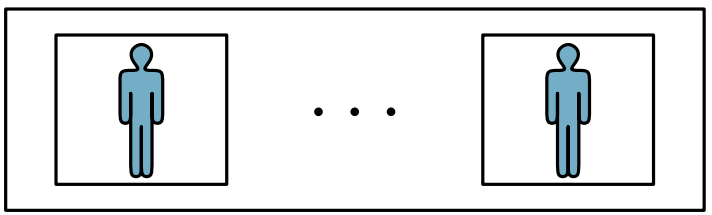
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)||\cdots||P(N)$

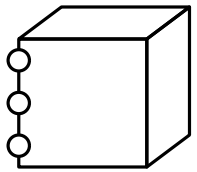
Safety Property

$\Box p$

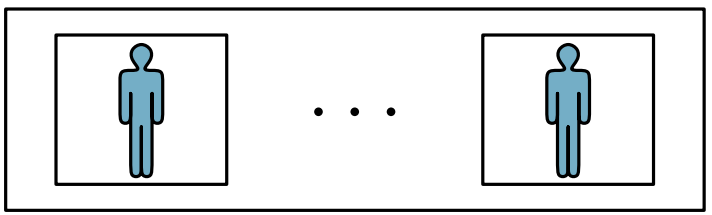
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)||\cdots||P(N)$

$\models ?$

Safety Property

$\Box p$

Verification of Concurrent Data-structures

Safety Properties

$$P[N] : P(1) || \cdots || P(N) \quad \overset{?}{\models} \quad \Box p$$

Verification of Concurrent Data-structures

Safety Properties

$$P[N] : P(1) || \cdots || P(N) \quad \overset{?}{\models} \quad \Box p$$

means:

forall N

$$P[N] \models \Box p$$

Verification of Concurrent Data-structures

Safety Properties

$$P[N] : P(1) || \cdots || P(N) \quad \overset{?}{\models} \quad \Box p$$

means:

forall N

$$P[N] \models \Box p\,(k_1, k_2)$$

Verification of Concurrent Data-structures

Safety Properties

$$P[N] : P(1) || \cdots || P(N) \quad \models^{?} \quad \Box p$$

means:

forall N

$$P[N] \models \Box p\,(k_1, k_2)$$

Parametrized Property

Verification of Concurrent Data-structures

Safety Properties

$$P[N] : P(1) || \cdots || P(N) \quad \overset{?}{\models} \quad \Box p$$

means:

$$\text{forall } N \;.\; \text{forall } k_1, k_2 : [N]$$
$$P[N] \models \Box p\,(k_1, k_2)$$

Verification of Concurrent Data-structures

Safety Properties

$$P[N] : P(1) || \cdots || P(N) \quad \models^{?} \quad \Box p$$

means:

forall N . forall $k_1, k_2 : [N]$

$$P[N] \models \Box p\,(k_1, k_2)$$

Uniform Verification Problem

Non-parametrized General Invariance Rule

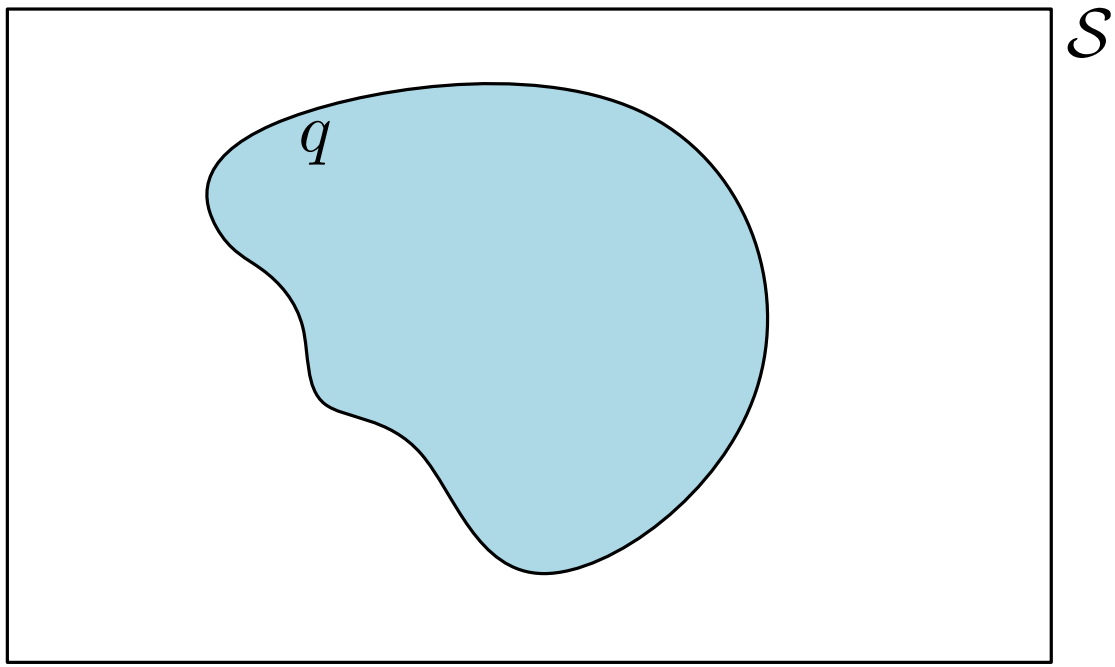
To show that P satisfies $\Box p$, find q:

$$\frac{\begin{array}{lcl} \text{I1.} & \Theta \rightarrow q & \\ \text{I2.} & q \wedge \tau \rightarrow q' & \text{for all } \tau \\ \text{I3.} & q \rightarrow p & \end{array}}{\Box p}$$

Non-parametrized General Invariance Rule

To show that P satisfies $\Box p$, find q:

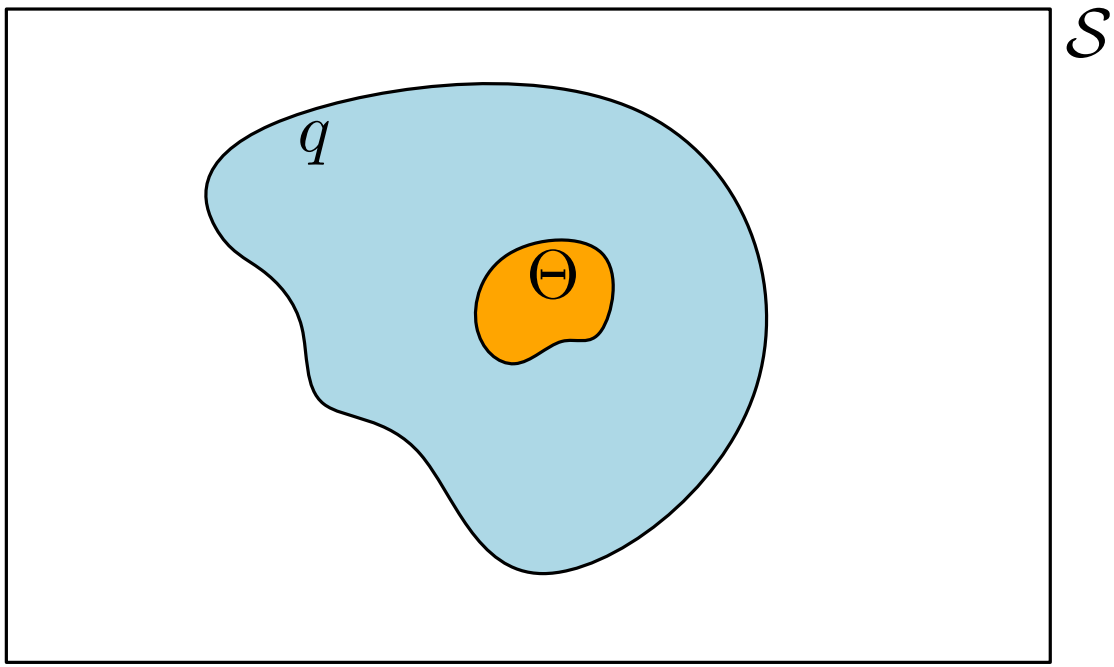
$$\frac{\begin{array}{lcl} \text{I1.} & \Theta \rightarrow q & \\ \text{I2.} & q \wedge \tau \rightarrow q' & \text{for all } \tau \\ \text{I3.} & q \rightarrow p & \end{array}}{\Box p}$$



Non-parametrized General Invariance Rule

To show that P satisfies $\Box p$, find q:

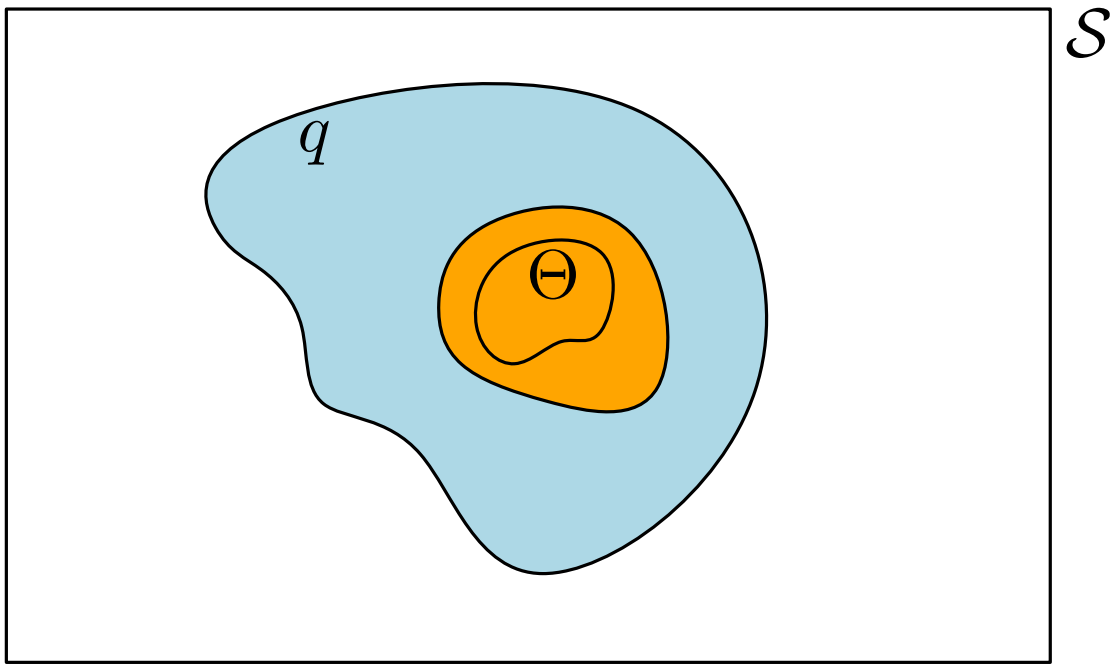
$$\frac{\begin{array}{lcl} \text{I1.} & \Theta \rightarrow q & \\ \text{I2.} & q \wedge \tau \rightarrow q' & \text{for all } \tau \\ \text{I3.} & q \rightarrow p & \end{array}}{\Box p}$$



Non-parametrized General Invariance Rule

To show that P satisfies $\Box p$, find q:

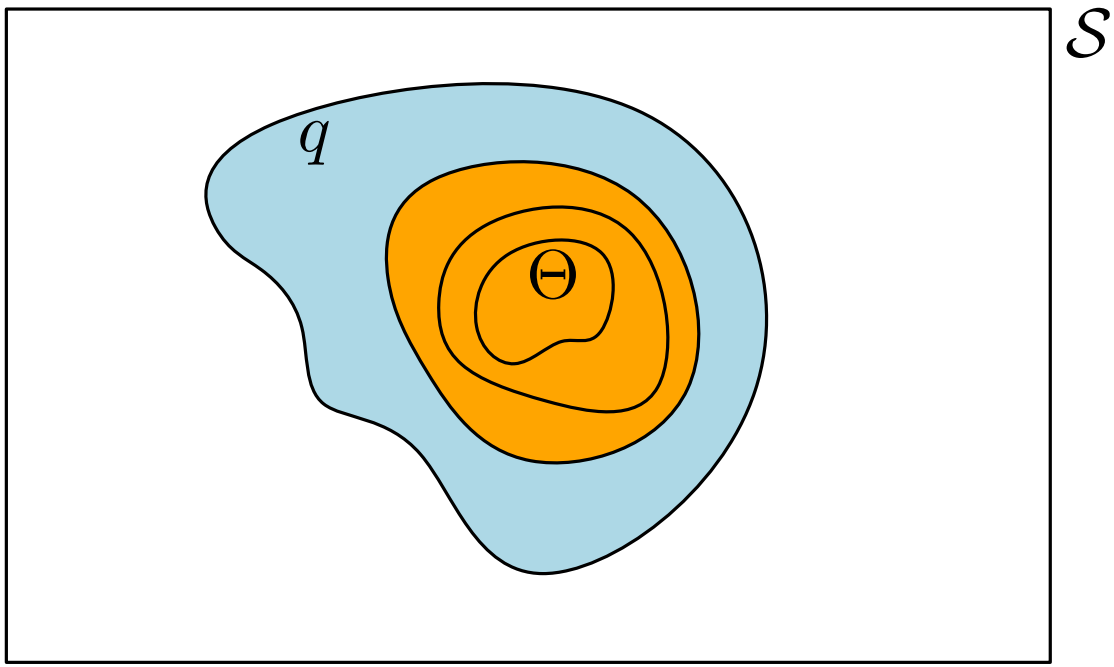
$$\frac{\begin{array}{lcl} \text{I1.} & \Theta \rightarrow q & \\ \text{I2.} & q \wedge \tau \rightarrow q' & \text{for all } \tau \\ \text{I3.} & q \rightarrow p & \end{array}}{\Box p}$$



Non-parametrized General Invariance Rule

To show that P satisfies $\Box p$, find q:

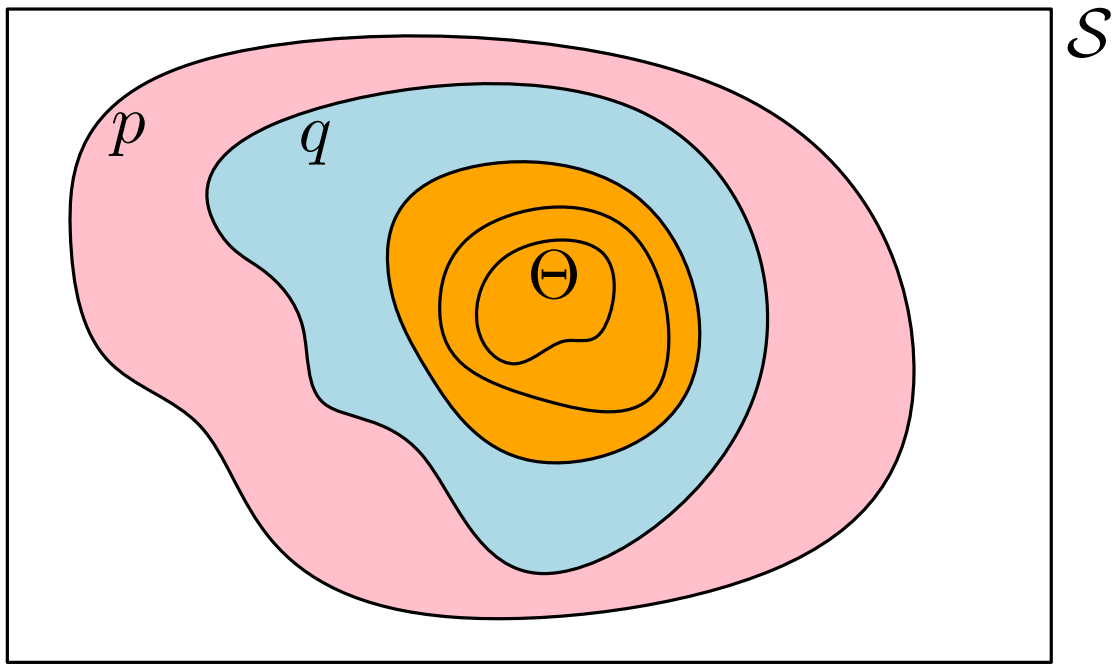
$$\frac{\begin{array}{lcl} \text{I1.} & \Theta \rightarrow q & \\ \text{I2.} & q \wedge \tau \rightarrow q' & \text{for all } \tau \\ \text{I3.} & q \rightarrow p & \end{array}}{\Box p}$$



Non-parametrized General Invariance Rule

To show that P satisfies $\Box p$, find q:

$$\frac{\begin{array}{lcl} \text{I1.} & \Theta \rightarrow q & \\ \text{I2.} & q \wedge \tau \rightarrow q' & \text{for all } \tau \\ \text{I3.} & q \rightarrow p & \end{array}}{\Box p}$$



Non-parametrized General Invariance Rule

To show that P satisfies $\Box p$, find q:

$$\frac{\begin{array}{lcl} \text{I1.} & \Theta \rightarrow q & \\ \text{I2.} & q \wedge \tau \rightarrow q' & \text{for all } \tau \\ \text{I3.} & q \rightarrow p & \end{array}}{\Box p}$$

Why this rule does not work for parametrized systems?

Motivating Example: Mutual Exclusion Protocol

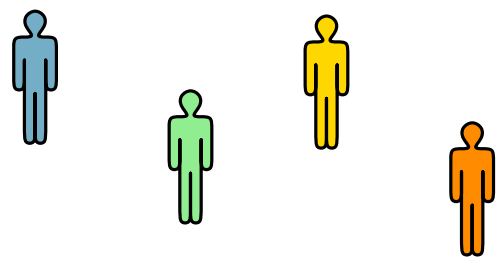
Motivating Example: Mutual Exclusion Protocol

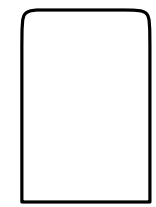
global
  $Int\ tick := 0$
  $Set\langle Int\rangle\ bag := \emptyset$

procedure SetMutex
  $Int\ ticket := 0$
begin
1: **while** $true$ **do**
2: **noncritical**
3: $\left\langle \begin{array}{l} ticket := tick++ \\ bag.add(ticket) \end{array} \right\rangle$
4: **await** $(bag.min == ticket)$
5: **critical**
6: $bag.remove(ticket)$
7: **end while**
end procedure

Motivating Example: Mutual Exclusion Protocol

```
   global
     Int tick := 0
     Set⟨Int⟩ bag := ∅

   procedure SetMutex
     Int ticket := 0
   begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
       ⟨ bag.add(ticket)
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
   end procedure
```



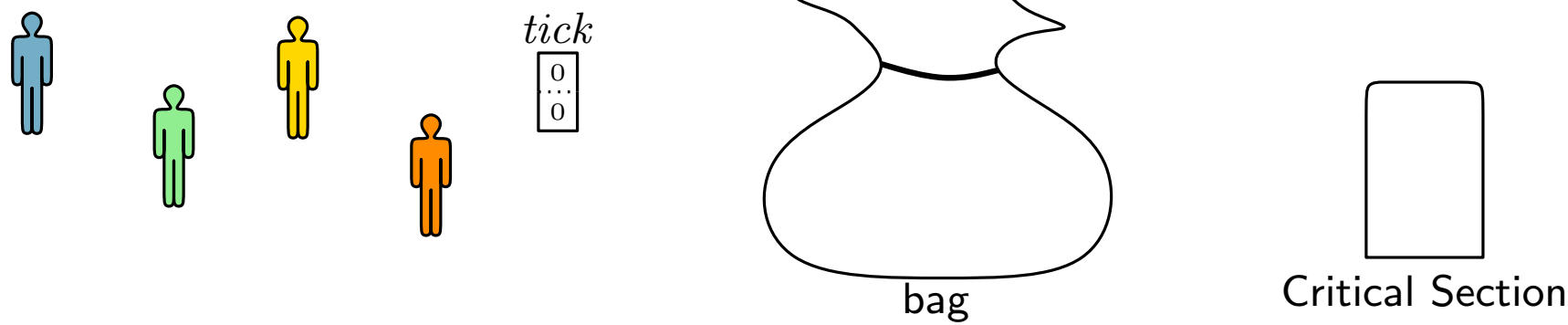


Critical Section

Motivating Example: Mutual Exclusion Protocol

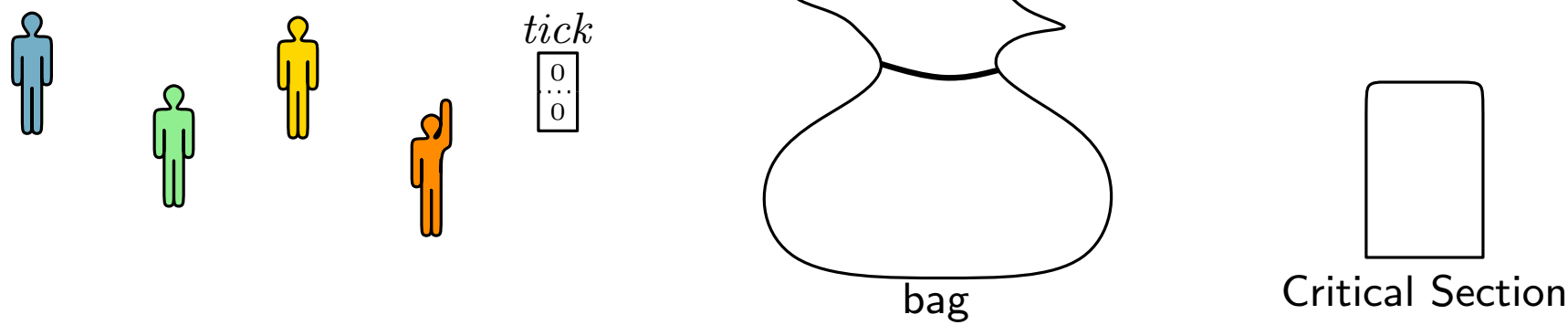
global
$Int\ tick := 0$
$Set\langle Int\rangle\ bag := \emptyset$

procedure SetMutex
$Int\ ticket := 0$
begin
1: **while** $true$ **do**
2: **noncritical**
3: $\left\langle \begin{array}{l} ticket := tick++ \\ bag.add(ticket) \end{array} \right\rangle$
4: **await** $(bag.min == ticket)$
5: **critical**
6: $bag.remove(ticket)$
7: **end while**
end procedure



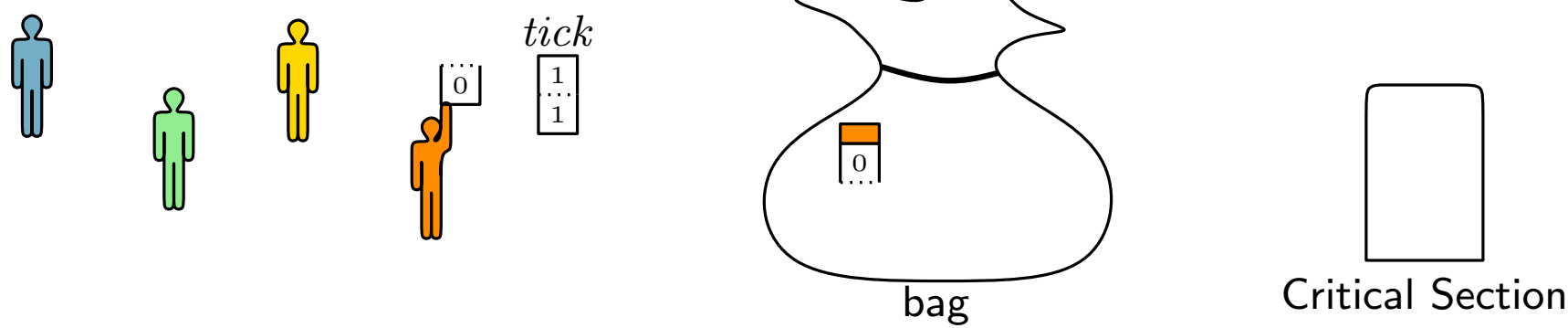
Motivating Example: Mutual Exclusion Protocol

```
global
  Int tick := 0
  Set⟨Int⟩ bag := ∅

procedure SetMutex
  Int ticket := 0
begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
       ⟩ bag.add(ticket)
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
end procedure
```



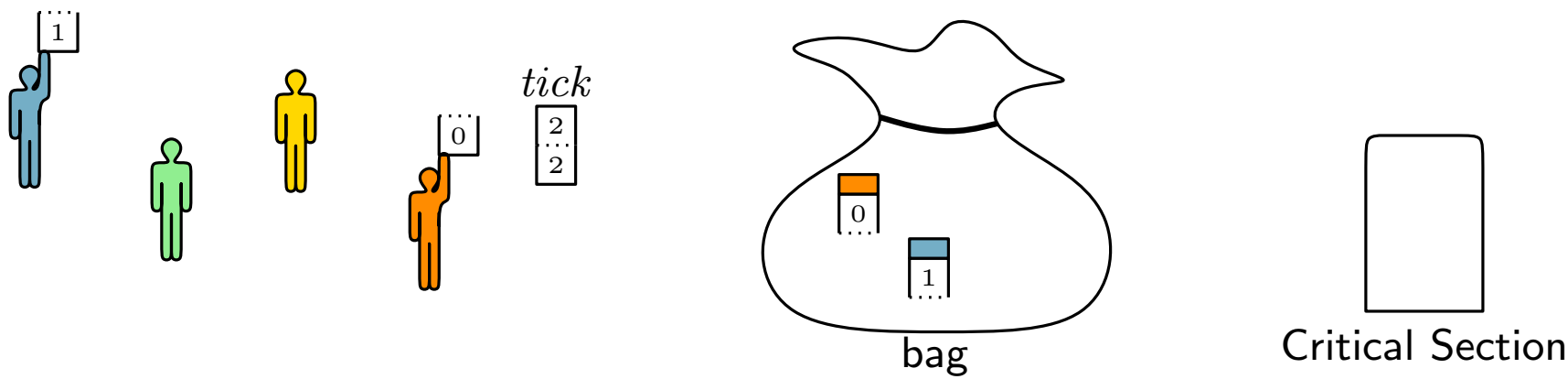
Motivating Example: Mutual Exclusion Protocol

```
global
  Int tick := 0
  Set⟨Int⟩ bag := ∅

procedure SetMutex
  Int ticket := 0
begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
       bag.add(ticket) ⟩
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
end procedure
```



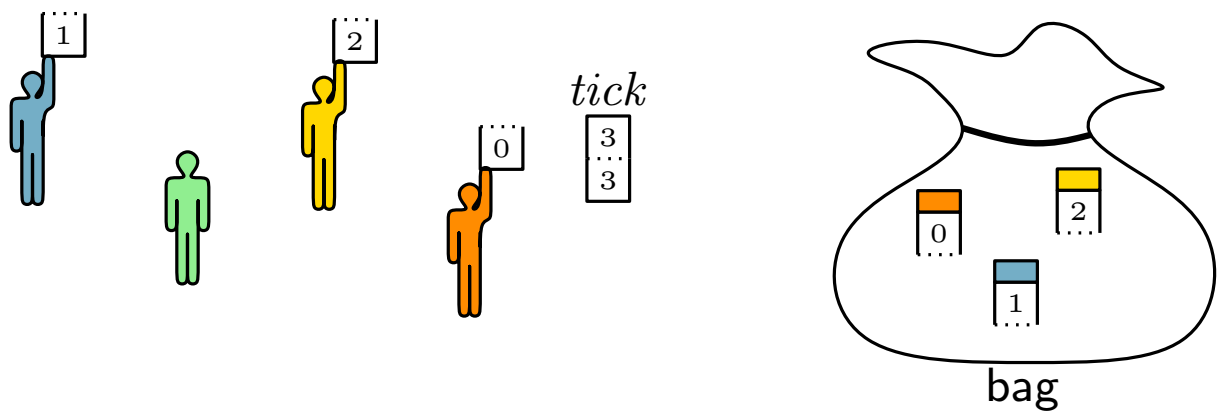
Motivating Example: Mutual Exclusion Protocol

```
global
  Int tick := 0
  Set⟨Int⟩ bag := ∅

procedure SetMutex
  Int ticket := 0
begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
       bag.add(ticket) ⟩
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
end procedure
```



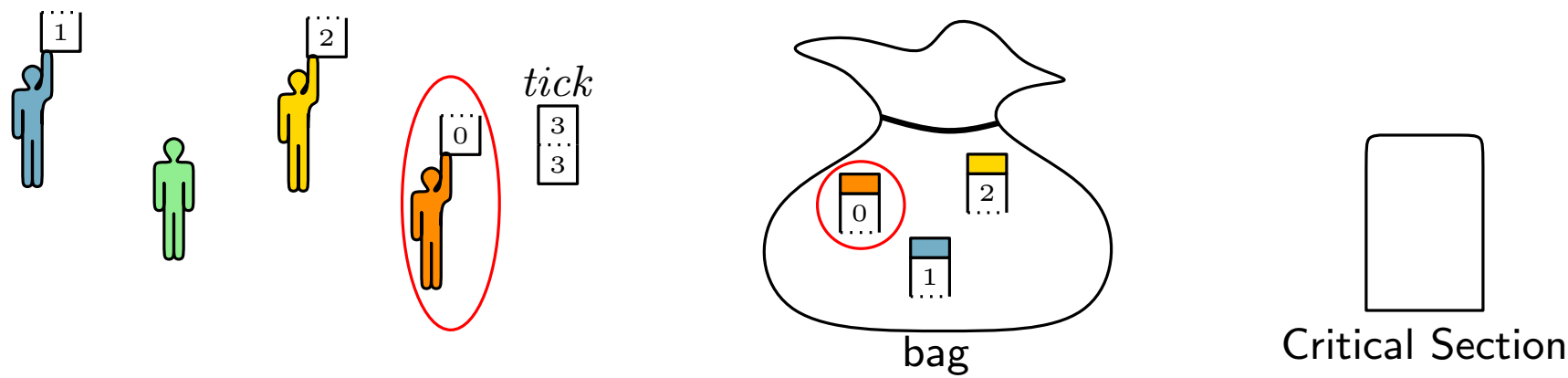
Motivating Example: Mutual Exclusion Protocol

```
global
  Int tick := 0
  Set⟨Int⟩ bag := ∅

procedure SetMutex
  Int ticket := 0
begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
       bag.add(ticket) ⟩
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
end procedure
```



Motivating Example: Mutual Exclusion Protocol

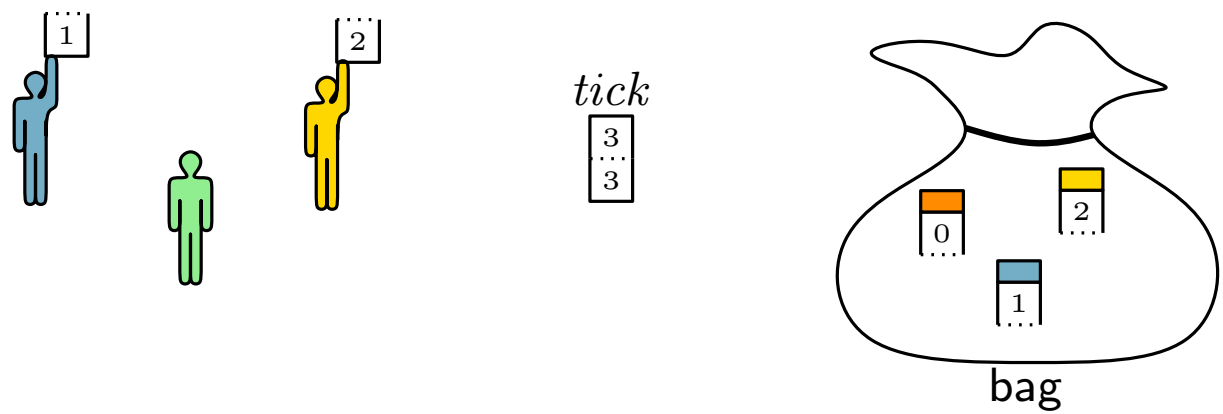
```
global
  Int tick := 0
  Set⟨Int⟩ bag := ∅

procedure SetMutex
  Int ticket := 0
begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
        bag.add(ticket) ⟩
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
end procedure
```



Motivating Example: Mutual Exclusion Protocol

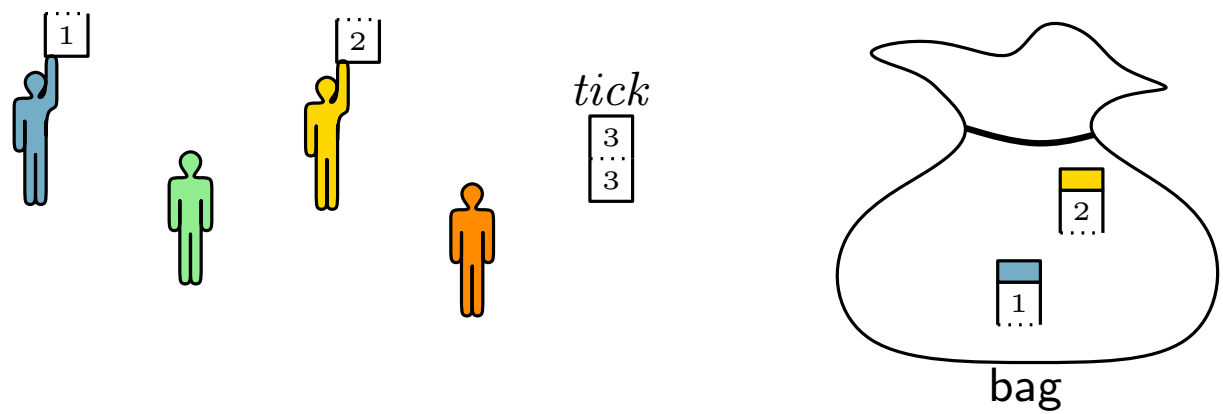
global
  $Int\ tick := 0$
  $Set\langle Int\rangle\ bag := \emptyset$

procedure SetMutex
  $Int\ ticket := 0$
begin
1: **while** *true* **do**
2: **noncritical**
3: $\left\langle \begin{array}{l} ticket := tick++ \\ bag.add(ticket) \end{array} \right\rangle$
4: **await** $(bag.min == ticket)$
5: **critical**
6: $bag.remove(ticket)$
7: **end while**
end procedure

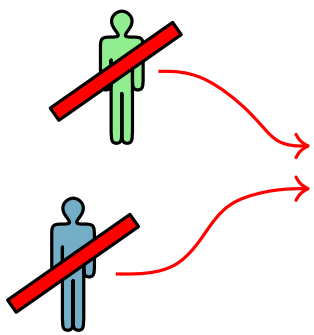


Motivating Example: Mutual Exclusion Protocol

```
global
  Int tick := 0
  Set⟨Int⟩ bag := ∅

procedure SetMutex
  Int ticket := 0
begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
        bag.add(ticket) ⟩
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
end procedure
```



Motivating Example: Mutual Exclusion Protocol



```
   global
     Int tick := 0
     Set⟨Int⟩ bag := ∅

   procedure SetMutex
     Int ticket := 0
   begin
1: while true do
2:     noncritical
3:     ⟨ ticket := tick++
       ⟨ bag.add(ticket)
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket)
7: end while
   end procedure
```

$$\mathsf{mutex}(i,j) \mathrel{\hat{=}} \Box\left[i \neq j \rightarrow \neg(critical(i) \wedge critical(j))\right]$$

General Deductive Verification [Manna-Pnueli '95]

1.	Θ	$\rightarrow$	mutex	
2.	mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex	
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex	
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1

General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex	
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1

l_1

General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex	
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$	for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

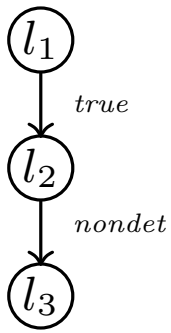
$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1



General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex	
2.	mutex $\wedge$ τ		$\rightarrow$	mutex$'$	for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

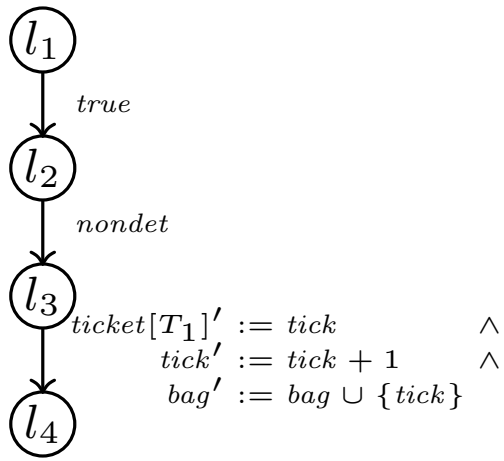
$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1



General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex	
2.	mutex $\wedge$ τ		$\rightarrow$	mutex$'$	for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

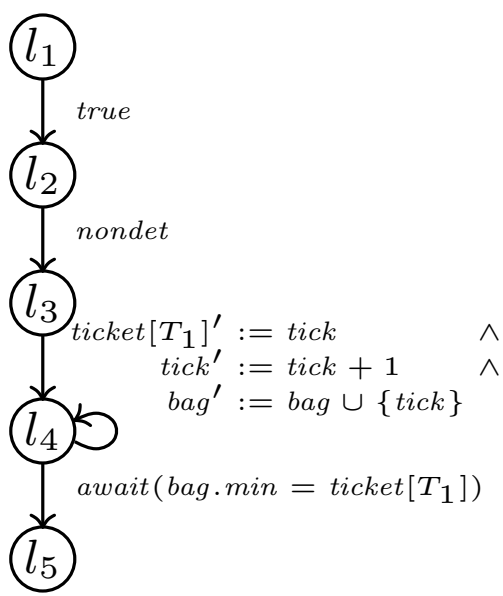
$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:

T_1



General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$ for each τ

2 Threads T_1, T_2

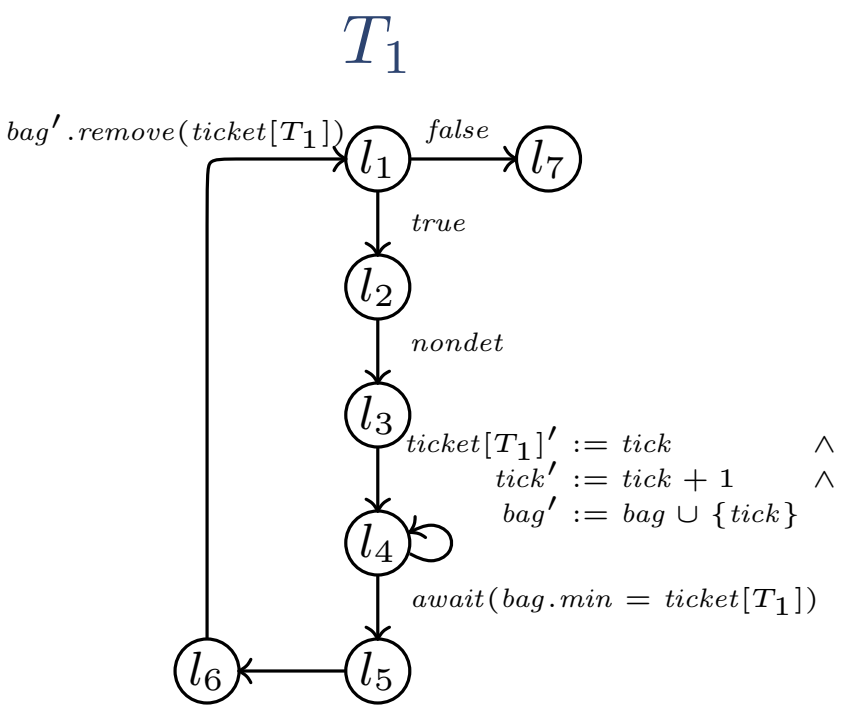
- Initiation: **1 VC**

$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution:



General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$ for each τ

2 Threads T_1, T_2

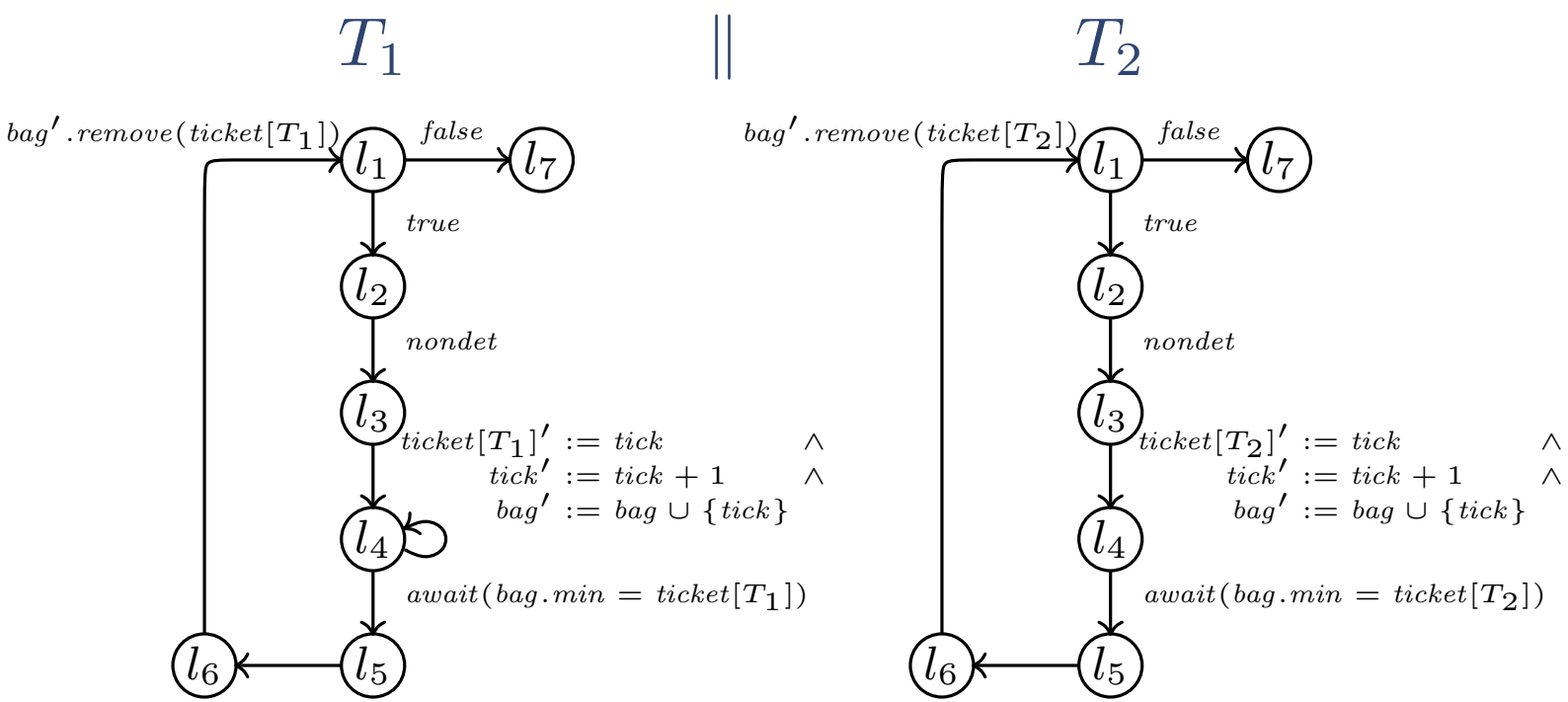
- Initiation: **1 VC**

$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution: **16 VC**



General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$ for each τ

2 Threads T_1, T_2

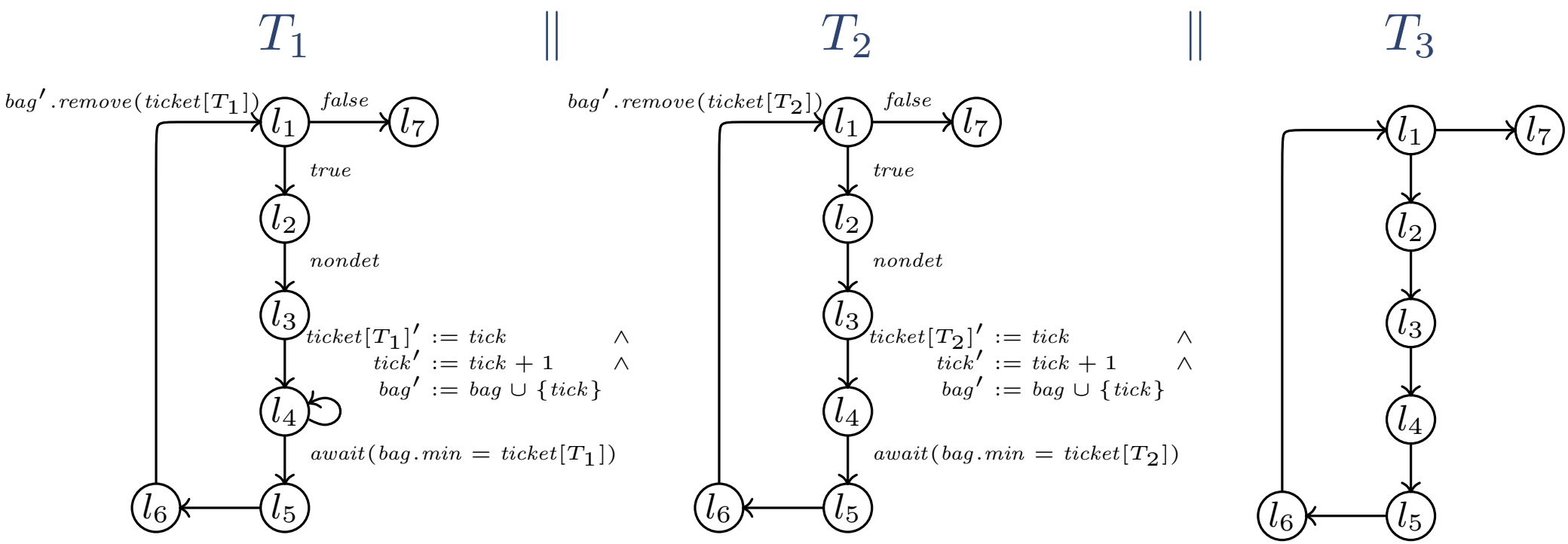
- Initiation: **1 VC**

$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

- Consecution: **16 VC**



General Deductive Verification [Manna-Pnueli '95]

1.		Θ	$\rightarrow$	mutex
2.		mutex $\wedge\ \tau$	$\rightarrow$	mutex$'$ for each τ

2 Threads T_1, T_2

- Initiation: **1 VC**

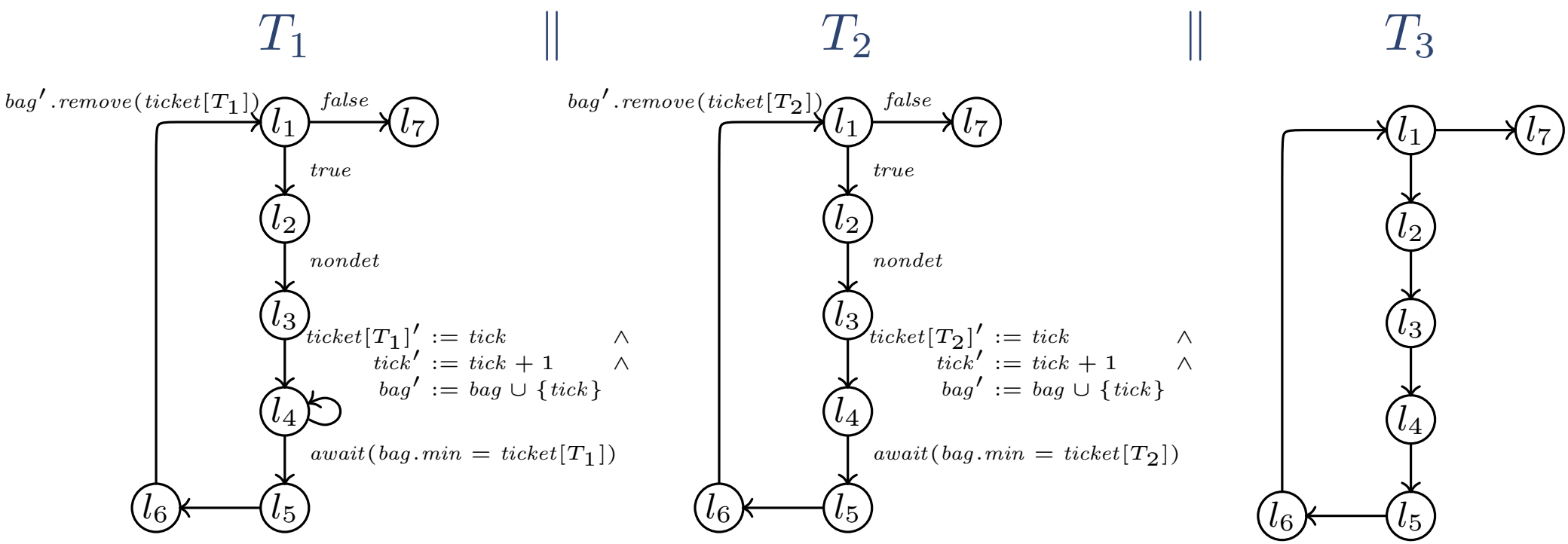
$\Theta_G : tick = 0 \wedge bag = \emptyset$

$\Theta_{T_1} : ticket[T_1] = 0 \wedge pc[T_1] = 1$

$\Theta_{T_2} : ticket[T_2] = 0 \wedge pc[T_2] = 1$

$\Theta_{T_3} : ticket[T_3] = 0 \wedge pc[T_3] = 1$

- Consecution: ~~**16 VC**~~ **24 VC**



Parametrized Invariance

Problem

Unbounded number of **verification conditions**

Parametrized Invariance

Problem

Unbounded number of **verification conditions**

Our solution

- **Specialized** invariance proof rules
- **Finite and bounded** number of verification conditions

Parametrized Invariance exploits the **similarities** of **fully symetric systems**

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$: **Initiation**

$$
\begin{array}{lrcll}
(\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \\
\hline
 & \Box\varphi & & &
\end{array}
$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

Self-consecution

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

Other-consecution

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\square\varphi}$$

- For our example: mutex(i, j)

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

- For our example: mutex(i, j) **#VC : 1**

$$(\ \mathsf{I}\ ) \qquad \Theta(i,j) \rightarrow \text{mutex}$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

- For our example: mutex(i, j) **#VC : 1 + 16**

$$\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta(i,j) & \rightarrow & \text{mutex} & \\ (\mathsf{SC}) & \text{mutex} \wedge \tau^{(i)} & \rightarrow & \text{mutex}' & \text{forall } \tau \\ & \text{mutex} \wedge \tau^{(j)} & \rightarrow & \text{mutex}' & \text{forall } \tau \end{array}$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

- For our example: mutex(i, j) **#VC : 1 + 16 + 8 = 25**

$$\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta(i,j) & \rightarrow & \text{mutex} & \\ (\mathsf{SC}) & \text{mutex} \wedge \tau^{(i)} & \rightarrow & \text{mutex}' & \text{forall } \tau \\ & \text{mutex} \wedge \tau^{(j)} & \rightarrow & \text{mutex}' & \text{forall } \tau \\ (\mathsf{OC}) & \text{mutex} \wedge k \neq i \wedge k \neq j \wedge \tau^{(k)} & \rightarrow & \text{mutex}' & \text{forall } \tau, \text{fresh } k \end{array}$$

Parametrized Invariance (p-inv)

- **Bounded** number of VC, based on **program** and **specification**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

Independently on #threads in the system

- For our example: mutex(i, j) **#VC : 1 + 16 + 8 = 25**

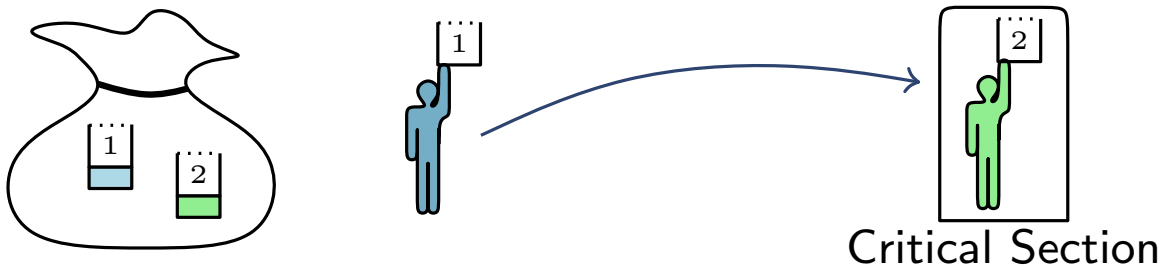
$$\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta(i,j) & \rightarrow & \text{mutex} & \\ (\mathsf{SC}) & \text{mutex} \wedge \tau^{(i)} & \rightarrow & \text{mutex}' & \text{forall } \tau \\ & \text{mutex} \wedge \tau^{(j)} & \rightarrow & \text{mutex}' & \text{forall } \tau \\ (\mathsf{OC}) & \text{mutex} \wedge k \neq i \wedge k \neq j \wedge \tau^{(k)} & \rightarrow & \text{mutex}' & \text{forall } \tau, \text{fresh } k \end{array}$$

Parametrized Invariance (p-inv) is not enough

- Lets try to prove mutex using P-INV...

Parametrized Invariance (p-inv) is not enough

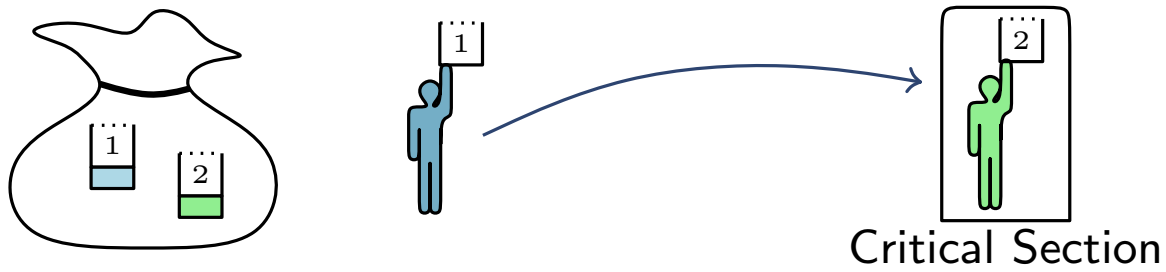
- Lets try to prove mutex using P-INV...
- ... transition 4 (i.e., **await** $(bag.min == ticket)$) **fails**



Because mutex does not encode that the
thread in the critical section owns the minimum ticket

Parametrized Invariance (p-inv) is not enough

- Lets try to prove mutex using P-INV...
- ... transition 4 (i.e., **await** $(bag.min == ticket)$) **fails**



Because mutex does not encode that the
thread in the critical section owns the minimum ticket

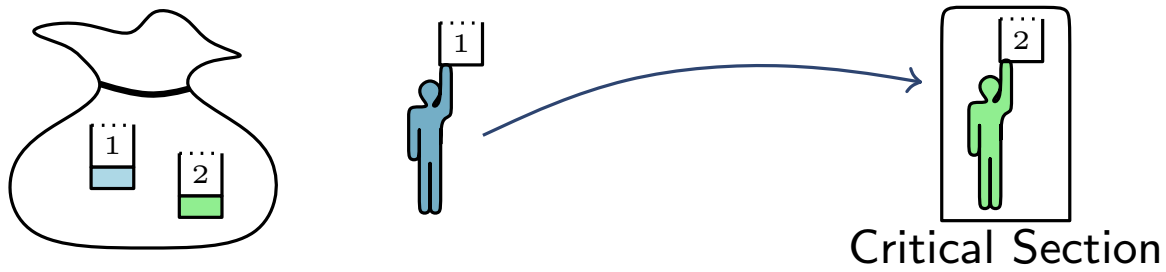
- Extra support is required

$$\text{minticket}(i) \;\hat{=}\; \Box\big[critical(i) \rightarrow min(bag) = ticket(i)\big]$$

$$\text{notsame}(i,j) \;\hat{=}\; \Box\big[i \neq j \wedge active(i) \wedge active(j) \rightarrow ticket(i) \neq ticket(j)\big]$$

Parametrized Invariance (p-inv) is not enough

- Lets try to prove mutex using P-INV...
- ... transition 4 (i.e., **await** $(bag.min == ticket)$) **fails**



Because mutex does not encode that the
thread in the critical section owns the minimum ticket

- Extra support is required

$$\text{minticket}(i) \;\hat{=}\; \Box\big[critical(i) \rightarrow min(bag) = ticket(i)\big]$$

$$\text{notsame}(i,j) \;\hat{=}\; \Box\big[i \neq j \wedge active(i) \wedge active(j) \rightarrow ticket(i) \neq ticket(j)\big]$$

We now require a **new rule** for **invariant support**

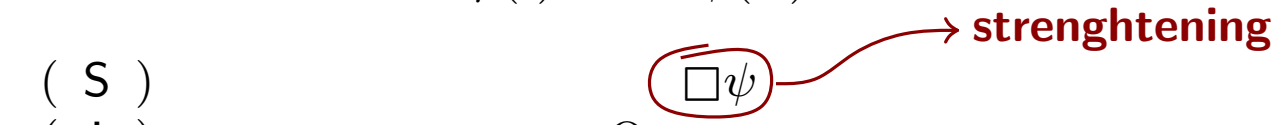
Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{lllllll}
(\ \mathsf{S}\ ) & & & & \Box\psi & & \\
(\ \mathsf{I}\ ) & & & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & & \rightarrow & \varphi' & \text{forall } \tau\text{, fresh } k \\
\hline
 & & & \Box\varphi & & &
\end{array}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:



$$
\begin{array}{lllcll}
(\ \mathsf{S}\ ) & & & \Box\psi & & \\
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau\text{, fresh } k \\
\hline
 & & \Box\varphi & & &
\end{array}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{lllcll}
(\ \mathsf{S}\ ) & & & & \Box\psi & \text{strenghtening} \\
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi & \text{initiation} \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{self-consecution} \quad \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{other-consecution} \quad \text{forall } \tau, \text{ fresh } k \\
\hline
 & & \Box\varphi & & &
\end{array}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$\frac{\begin{array}{llrcll}
(\ \mathsf{S}\ ) & & & \Box\psi & & \\
(\ \mathsf{I}\ ) & & \Theta & \to & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \to & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \to & \varphi' & \text{forall } \tau, \text{ fresh } k
\end{array}}{\Box\varphi}$$

Instantiate the assumptions for **self** and **others**

$\psi \rhd (A \to B)$ whether $\left[\left(\bigwedge_{\sigma \in S} \psi_\sigma \wedge A\right) \to B\right]$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$\frac{\begin{array}{llrcll}
(\ \mathsf{S}\ ) & & & \Box\psi & & \\
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau\text{, fresh } k
\end{array}}{\Box\varphi}$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$\frac{\begin{array}{llrcll}
(\ \mathsf{S}\ ) & & & \Box\psi & & \\
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau\text{, fresh } k
\end{array}}{\Box\varphi}$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

(S) $\Box$minticket $\wedge$ $\Box$notsame

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

(S)			$\Box\psi$		
(I)		Θ	$\rightarrow$	φ	
(SC)	$\psi, \varphi \rhd$	$\tau^{(i)}$	$\rightarrow$	φ'	forall τ
(OC)	$\psi, \varphi \rhd$	$k \neq i \wedge \tau^{(k)}$	$\rightarrow$	φ'	forall τ, fresh k
		$\Box\varphi$			

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

(S) $\Box$minticket $\wedge$ $\Box$notsame

(I) $\Theta(i, j) \rightarrow$ mutex

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{lllll}
(\ \mathsf{S}\ ) & & \Box\psi & & \\
(\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) \quad \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) \quad \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k \\
\hline
 & \Box\varphi & & &
\end{array}
$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

$$
\begin{array}{ll}
(\ \mathsf{S}\ ) & \Box\mathsf{minticket} \wedge \Box\mathsf{notsame} \\
(\ \mathsf{I}\ ) & \Theta(i,j) \quad \rightarrow \quad \mathsf{mutex} \\
(\mathsf{SC}) & \left[\bigwedge_{\sigma \in \{t_1,\ldots,t_5\} \rightarrow \{i,j\}} \left[\begin{array}{l} \mathsf{minticket}(t_1) \ \wedge \\ \mathsf{notsame}(t_2, t_3) \wedge \\ \mathsf{mutex}(t_4, t_5) \end{array} \right]_\sigma \wedge \tau^{(i)} \right] \rightarrow \mathsf{mutex}' \quad \forall\tau
\end{array}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$\frac{\begin{array}{llrcll}
(\ \mathsf{S}\ ) & & & \Box\psi & & \\
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k
\end{array}}{\Box\varphi}$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

$$\begin{array}{lll}
(\ \mathsf{S}\ ) & \Box\mathsf{minticket} \wedge \Box\mathsf{notsame} & \\
(\ \mathsf{I}\ ) & \Theta(i,j) \rightarrow \mathsf{mutex} & \\
(\mathsf{SC}) & \left[\bigwedge_{\sigma\in\{t_1,\ldots,t_5\}\rightharpoonup\{i,j\}} \left[\begin{array}{l}\mathsf{minticket}(t_1)\ \wedge \\ \mathsf{notsame}(t_2,t_3)\wedge \\ \mathsf{mutex}(t_4,t_5)\end{array}\right]_\sigma \wedge \tau^{(i)}\right] \rightarrow \mathsf{mutex}' \quad \forall\tau & \\
 & \left[\bigwedge_{\sigma\in\{t_1,\ldots,t_5\}\rightharpoonup\{i,j\}} \left[\begin{array}{l}\mathsf{minticket}(t_1)\ \wedge \\ \mathsf{notsame}(t_2,t_3)\wedge \\ \mathsf{mutex}(t_4,t_5)\end{array}\right]_\sigma \wedge \tau^{(j)}\right] \rightarrow \mathsf{mutex}' \quad \forall\tau &
\end{array}$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$
\begin{array}{lllllll}
(\ \mathsf{S}\ ) & & & \Box\psi & & & \\
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi & & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau & \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{ fresh } k & \\
\hline
 & & \Box\varphi & & & &
\end{array}
$$

Instantiate the assumptions for **self** and **others**

- **Example:** minticket and notsame to support mutex(i, j)

$$
\begin{array}{lllll}
(\ \mathsf{S}\ ) & \Box\text{minticket} \wedge \Box\text{notsame} & & & \\
(\ \mathsf{I}\ ) & & \Theta(i,j) & \rightarrow & \text{mutex} \\
(\mathsf{SC}) & \left[\bigwedge_{\sigma \in \{t_1,\ldots,t_5\} \rightarrow \{i,j\}} \left[\begin{array}{l} \text{minticket}(t_1) \ \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5) \end{array} \right]_\sigma \wedge \tau^{(i)} \right] & \rightarrow & \text{mutex}' & \forall\tau \\
 & \left[\bigwedge_{\sigma \in \{t_1,\ldots,t_5\} \rightarrow \{i,j\}} \left[\begin{array}{l} \text{minticket}(t_1) \ \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5) \end{array} \right]_\sigma \wedge \tau^{(j)} \right] & \rightarrow & \text{mutex}' & \forall\tau \\
(\mathsf{OC}) & \left[\bigwedge_{\sigma \in \{t_1,\ldots t_5\} \rightarrow \{i,j,k\}} \left[\begin{array}{l} \text{minticket}(t_1) \ \wedge \\ \text{notsame}(t_2,t_3) \wedge \\ \text{mutex}(t_4,t_5) \end{array} \right]_\sigma \wedge k \neq i \wedge k \neq j \wedge \tau^{(k)} \right] & \rightarrow & \text{mutex}' & \forall\tau
\end{array}
$$

Parametrized Invariance with Support (sp-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi(i)$. Find $\psi(\overline{w})$ with:

$$\frac{\begin{array}{llrcll}
(\ \mathsf{S}\ ) & & & \Box\psi & & \\
(\ \mathsf{I}\ ) & & \Theta & \to & \varphi & \\
(\mathsf{SC}) & \psi, \varphi \rhd & \tau^{(i)} & \to & \varphi' & \text{forall } \tau \\
(\mathsf{OC}) & \psi, \varphi \rhd & k \neq i \wedge \tau^{(k)} & \to & \varphi' & \text{forall } \tau, \text{ fresh } k
\end{array}}{\Box\varphi}$$

Instantiate the assumptions for **self** and **others**

Problem

Sometimes we have **invariant circular dependency**

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

$$
\begin{array}{lllll}
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi_i \wedge \varphi_j \\
(\ \mathsf{SC_i}\ ) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_i' \qquad \text{forall } \tau, \text{ forall } t \in \overline{v} \\
(\ \mathsf{SC_j}\ ) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_j' \qquad \text{forall } \tau, \text{ forall } t \in \overline{w} \\
(\mathsf{OC_i}) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_i' \qquad \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
(\mathsf{OC_j}) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_j' \qquad \text{forall } \tau, \text{ fresh } k \notin \overline{w} \\
\hline
 & & \Box\varphi_i \wedge \Box\varphi_j & &
\end{array}
$$

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

initiation

(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	
(SC$_\text{i}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
(SC$_\text{j}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
(OC$_\text{i}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
(OC$_\text{j}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$

$\Box\varphi_i \wedge \Box\varphi_j$

Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

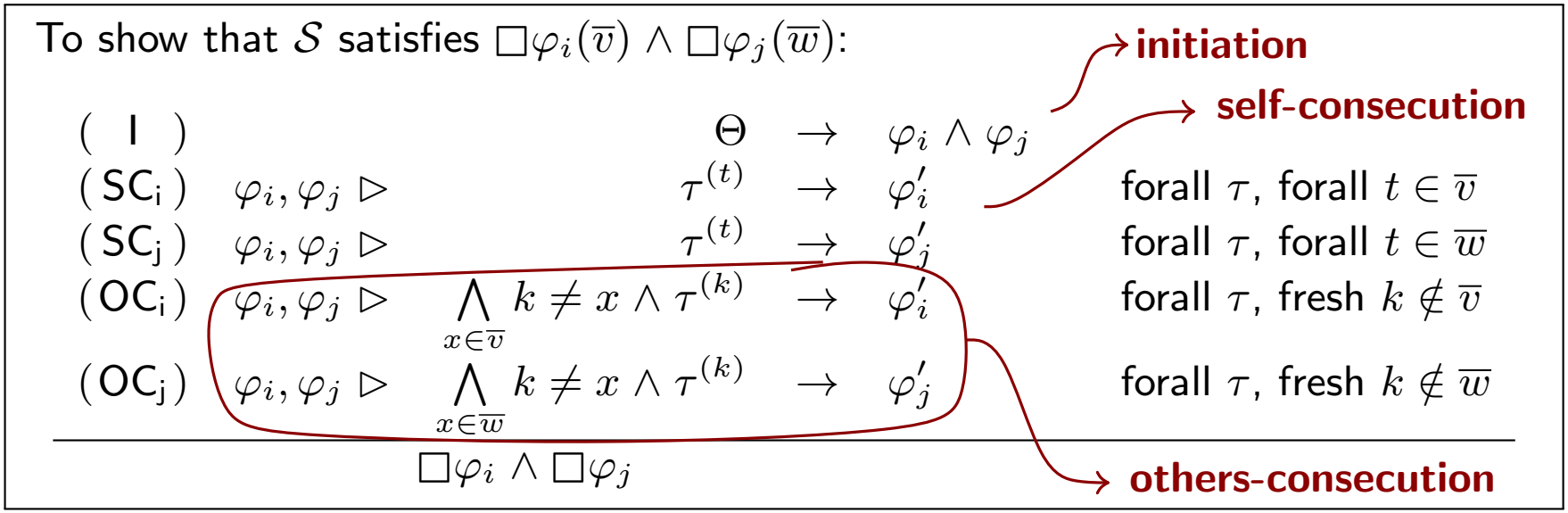
initiation

self-consecution

$$
\begin{array}{lllll}
(\ \mathsf{I}\ ) & & \Theta & \rightarrow & \varphi_i \wedge \varphi_j \\
(\ \mathsf{SC_i}\ ) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_i' \qquad \text{forall } \tau, \text{ forall } t \in \overline{v} \\
(\ \mathsf{SC_j}\ ) & \varphi_i, \varphi_j \rhd & \tau^{(t)} & \rightarrow & \varphi_j' \qquad \text{forall } \tau, \text{ forall } t \in \overline{w} \\
(\mathsf{OC_i}) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_i' \qquad \text{forall } \tau, \text{ fresh } k \notin \overline{v} \\
(\mathsf{OC_j}) & \varphi_i, \varphi_j \rhd & \bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)} & \rightarrow & \varphi_j' \qquad \text{forall } \tau, \text{ fresh } k \notin \overline{w}
\end{array}
$$

$$\Box\varphi_i \wedge \Box\varphi_j$$

Parametrized Invariance with Graph Support (g-inv)



Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\Box\varphi_i(\overline{v}) \wedge \Box\varphi_j(\overline{w})$:

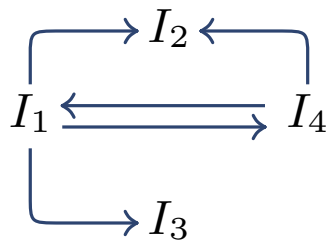
(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	
(SC$_\text{i}$)	$\varphi_i, \varphi_j \triangleright$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
(SC$_\text{j}$)	$\varphi_i, \varphi_j \triangleright$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
(OC$_\text{i}$)	$\varphi_i, \varphi_j \triangleright$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
(OC$_\text{j}$)	$\varphi_i, \varphi_j \triangleright$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$
		$\Box\varphi_i \wedge \Box\varphi_j$			

initiation

self-consecution

others-consecution

- A **generalization** of G-INV is a **proof graph**



Parametrized Invariance with Graph Support (g-inv)

To show that $\mathcal{S}$ satisfies $\square\varphi_i(\overline{v}) \wedge \square\varphi_j(\overline{w})$:

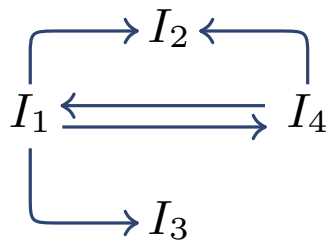
(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	
($\mathsf{SC_i}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
($\mathsf{SC_j}$)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
($\mathsf{OC_i}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
($\mathsf{OC_j}$)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$
		$\square\varphi_i \wedge \square\varphi_j$			

initiation

self-consecution

others-consecution

- A **generalization** of G-INV is a **proof graph**



Theorem

Every node is invariant if every node is either:

- an inductive invariant, or
- has an incident edge and all VCs are valid

Parametrized Invariance with Graph Support (g-inv)

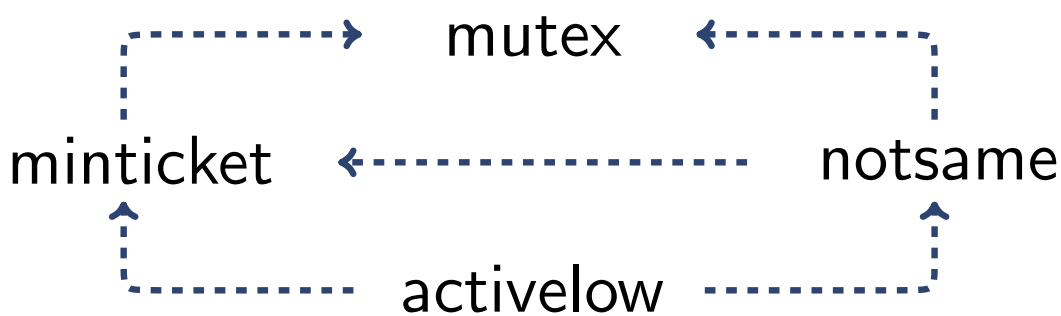
To show that $\mathcal{S}$ satisfies $\square\varphi_i(\overline{v}) \wedge \square\varphi_j(\overline{w})$:

(I)		Θ	$\rightarrow$	$\varphi_i \wedge \varphi_j$	
(SC_i)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_i'	forall τ, forall $t \in \overline{v}$
(SC_j)	$\varphi_i, \varphi_j \rhd$	$\tau^{(t)}$	$\rightarrow$	φ_j'	forall τ, forall $t \in \overline{w}$
(OC_i)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{v}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_i'	forall τ, fresh $k \notin \overline{v}$
(OC_j)	$\varphi_i, \varphi_j \rhd$	$\bigwedge_{x \in \overline{w}} k \neq x \wedge \tau^{(k)}$	$\rightarrow$	φ_j'	forall τ, fresh $k \notin \overline{w}$
		$\square\varphi_i \wedge \square\varphi_j$			

initiation

self-consecution

others-consecution



Theorem

Every node is invariant if every node is either:

- an inductive invariant, or
- has an incident edge and all VCs are valid

Our Contributions

That's for safety... what about **liveness**?

Our Contributions

A Deductive Verification Techniques for Parametrized Systems

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

3 Invariant Generation with Self-refelction
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

4 TL3: A Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

5 TSL_K: A Decidable Family for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

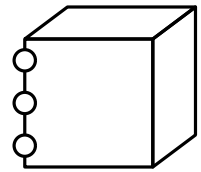
6 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

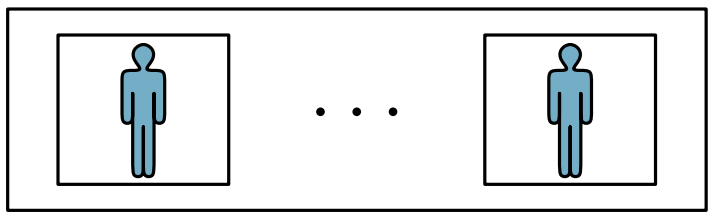
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure

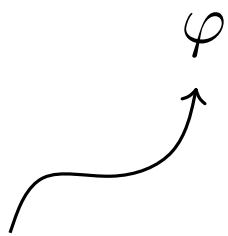


Most General Client



$P[N] : P(1)||\cdots||P(N)$

Property

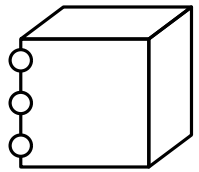


LTL ($\Box$,$\Diamond$,$\mathcal{U}$,. . .)

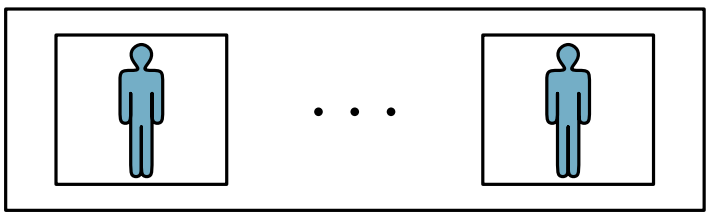
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure

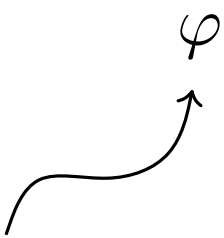


Most General Client



$P[N] : P(1)||\cdots||P(N)$

Liveness
Property

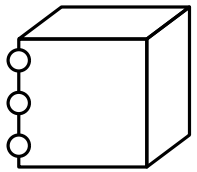


LTL ($\Box$,$\Diamond$,$\mathcal{U}$,. . .)

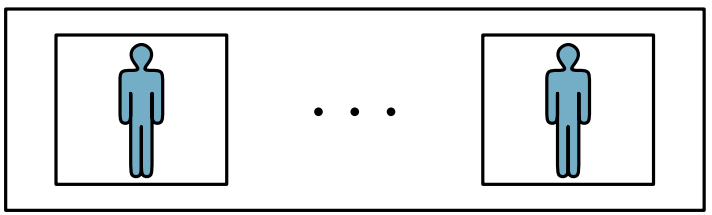
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)||\cdots||P(N)$

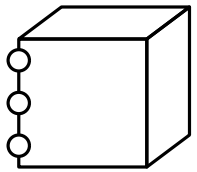
Liveness Property

$\varphi^{(k)}$

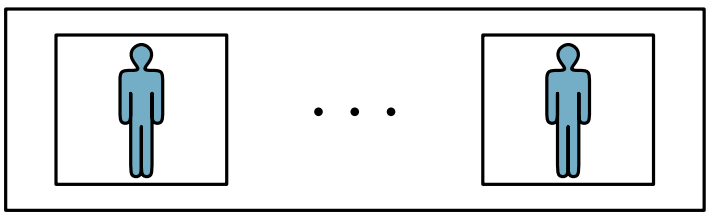
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)||\cdots||P(N)$

$\models$?

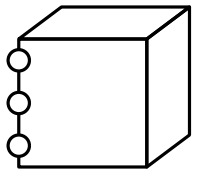
Liveness Property

$\varphi^{(k)}$

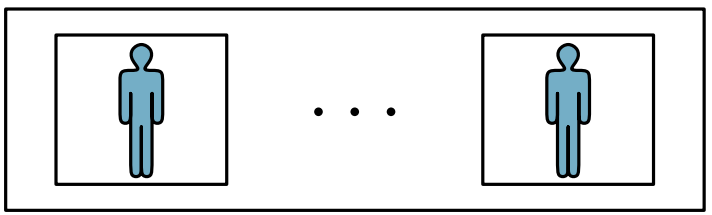
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)|| \cdots ||P(N)$

Diagram

$\mathcal{D}$

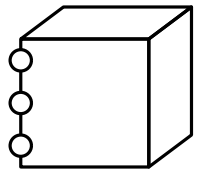
Liveness Property

$\varphi^{(k)}$

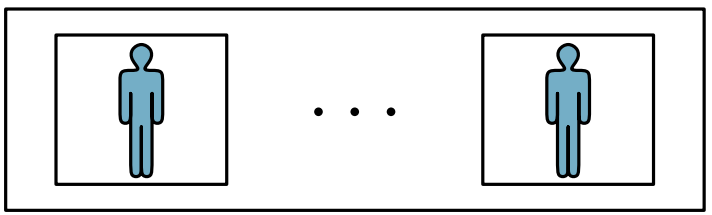
Verification of Concurrent Data-structures

Our verification approach

Concurrent DataStructure



Most General Client



$P[N] : P(1)||\cdots||P(N)$

Diagram

$\mathcal{D}$

Liveness Property

$\varphi^{(k)}$

$\models$ Verification Conditions:

- Initiation
- Consecution
- Acceptance
- Fairness

$\models$ Satisfaction (Model Checking)

The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

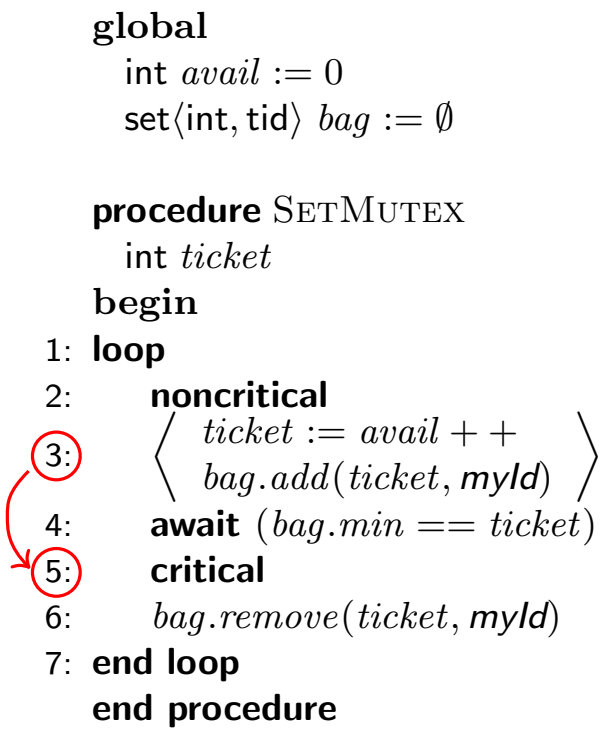
global
  int $avail := 0$
  set⟨int, tid⟩ $bag := \emptyset$

procedure SetMutex
  int $ticket$
begin
1: **loop**
2: **noncritical**
3: $\left\langle \begin{array}{l} ticket := avail++ \\ bag.add(ticket, myId) \end{array} \right\rangle$
4: **await** $(bag.min == ticket)$
5: **critical**
6: $bag.remove(ticket, myId)$
7: **end loop**
end procedure

The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$



The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

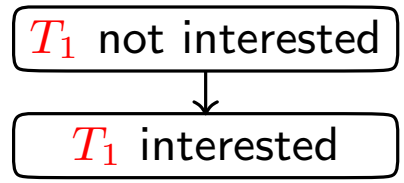
T_1 not interested

```
   global
     int avail := 0
     set⟨int, tid⟩ bag := ∅

   procedure SetMutex
     int ticket
   begin
1: loop
2:     noncritical
3:     ⟨ ticket := avail++
       ⟨ bag.add(ticket, myId)
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket, myId)
7: end loop
   end procedure
```

The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

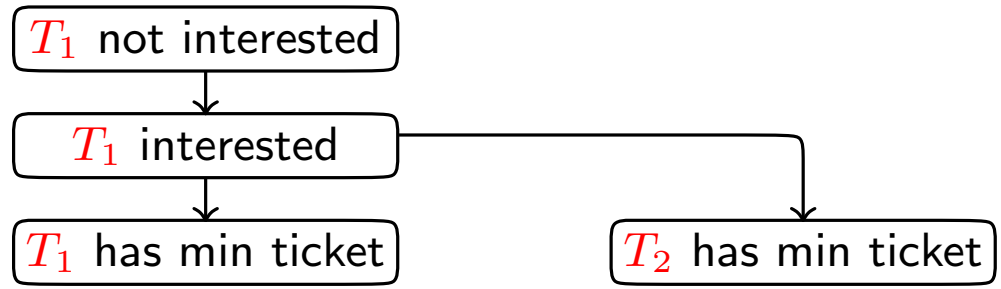



```
    global
      int avail := 0
      set⟨int, tid⟩ bag := ∅

    procedure SetMutex
      int ticket
    begin
1:  loop
2:    noncritical
3:    ⟨ ticket := avail++
      ⟩ bag.add(ticket, myId)
4:    await (bag.min == ticket)
5:    critical
6:    bag.remove(ticket, myId)
7:  end loop
    end procedure
```

The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$



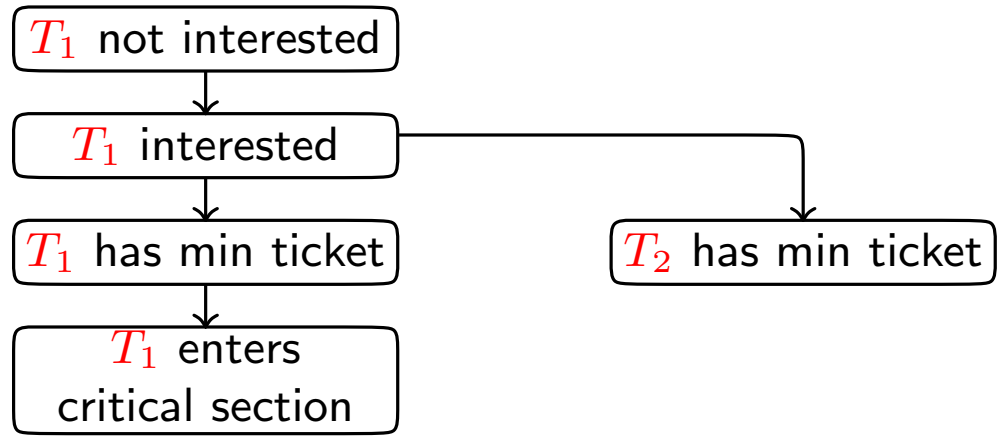

```
    global
      int avail := 0
      set⟨int, tid⟩ bag := ∅

    procedure SetMutex
      int ticket
    begin
1:  loop
2:    noncritical
3:    ⟨ ticket := avail++
      ⟩ bag.add(ticket, myId)
4:    await (bag.min == ticket)
5:    critical
6:    bag.remove(ticket, myId)
7:  end loop
    end procedure
```

The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

```
   global
     int avail := 0
     set⟨int, tid⟩ bag := ∅

   procedure SetMutex
     int ticket
   begin
1: loop
2:     noncritical
3:     ⟨ ticket := avail++
       ⟨ bag.add(ticket, myId)
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket, myId)
7: end loop
   end procedure
```

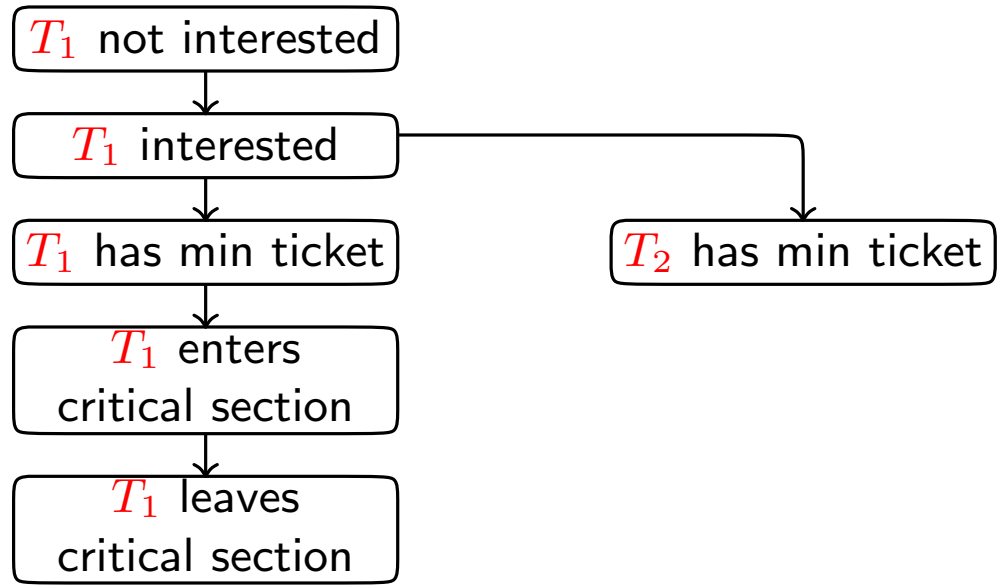


The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

```
    global
      int avail := 0
      set⟨int, tid⟩ bag := ∅

    procedure SetMutex
      int ticket
    begin
1:  loop
2:    noncritical
3:    ⟨ ticket := avail++
      ⟨ bag.add(ticket, myId)
4:    await (bag.min == ticket)
5:    critical
6:    bag.remove(ticket, myId)
7:  end loop
    end procedure
```

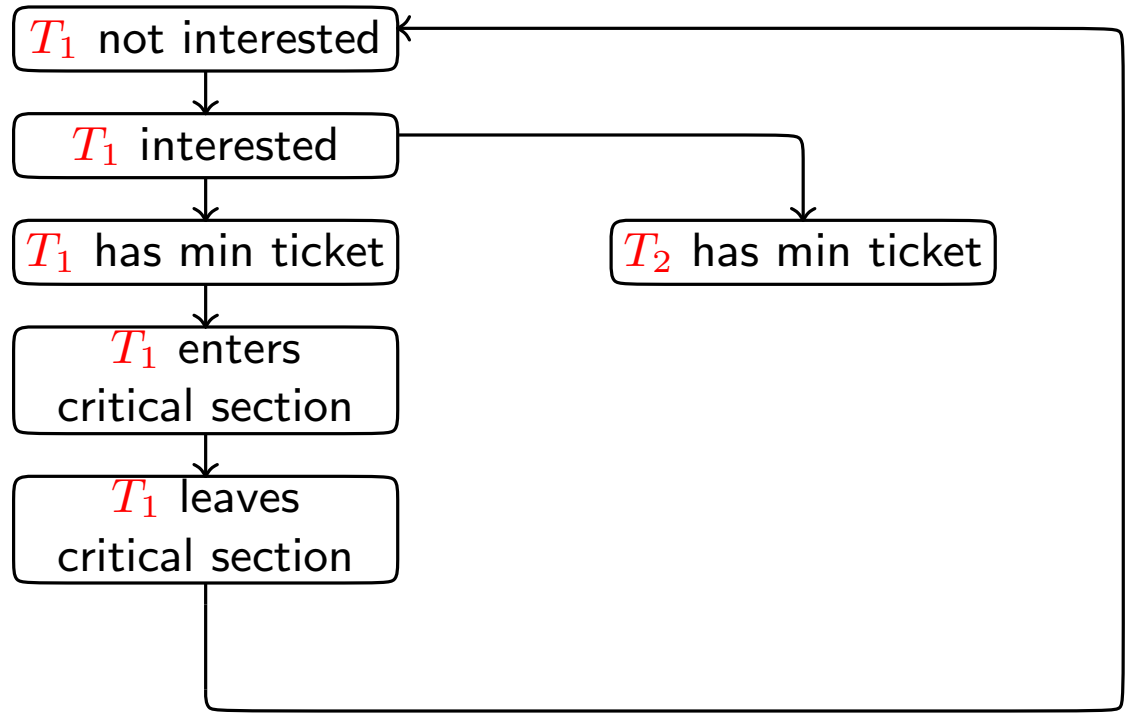


The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

```
global
  int avail := 0
  set⟨int, tid⟩ bag := ∅

procedure SetMutex
  int ticket
begin
1: loop
2:    noncritical
3:    ⟨ ticket := avail++
      ⟩ bag.add(ticket, myId)
4:    await (bag.min == ticket)
5:    critical
6:    bag.remove(ticket, myId)
7: end loop
end procedure
```

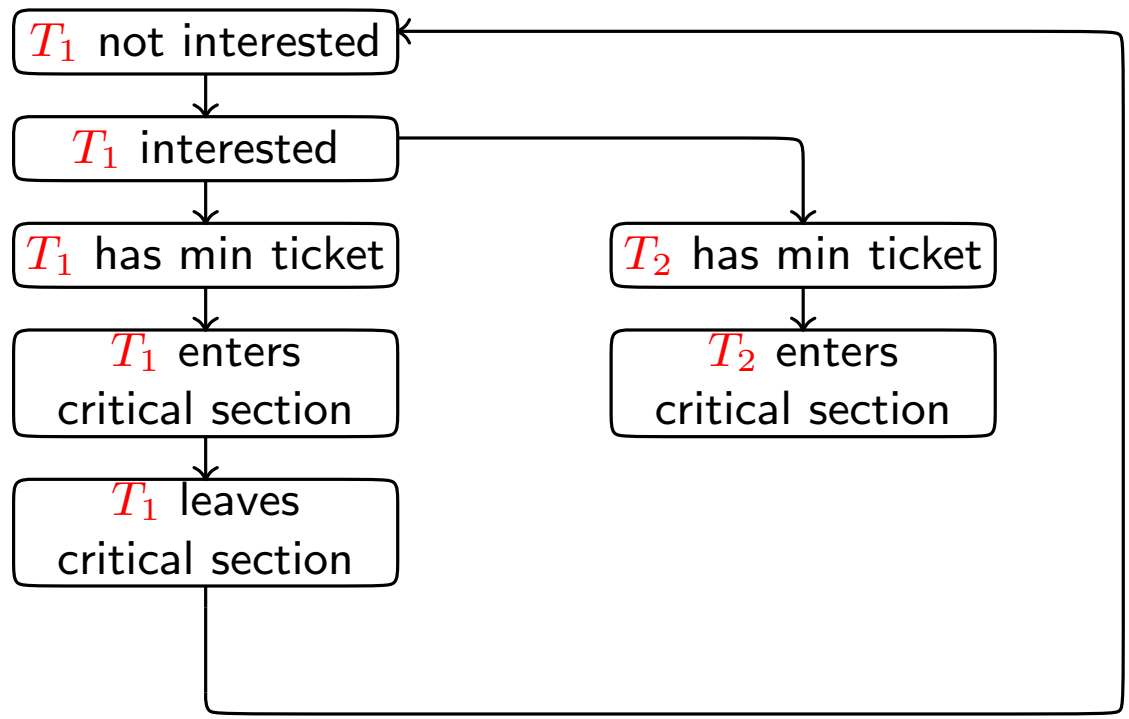


The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

```
   global
     int avail := 0
     set⟨int, tid⟩ bag := ∅

   procedure SetMutex
     int ticket
   begin
1: loop
2:     noncritical
3:     ⟨ ticket := avail++
       ⟩ bag.add(ticket, myId)
4:     await (bag.min == ticket)
5:     critical
6:     bag.remove(ticket, myId)
7: end loop
   end procedure
```

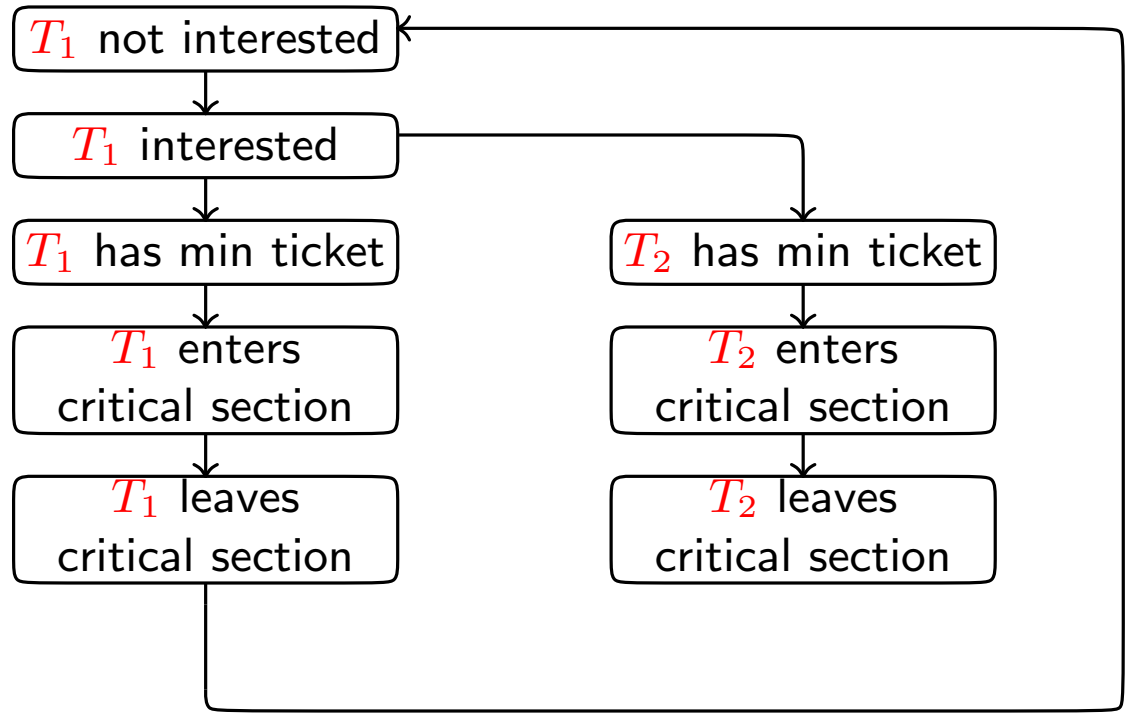


The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

```
global
  int avail := 0
  set⟨int, tid⟩ bag := ∅

procedure SetMutex
  int ticket
begin
1: loop
2:   noncritical
3:   ⟨ ticket := avail++
     ⟩ bag.add(ticket, myId)
4:   await (bag.min == ticket)
5:   critical
6:   bag.remove(ticket, myId)
7: end loop
end procedure
```

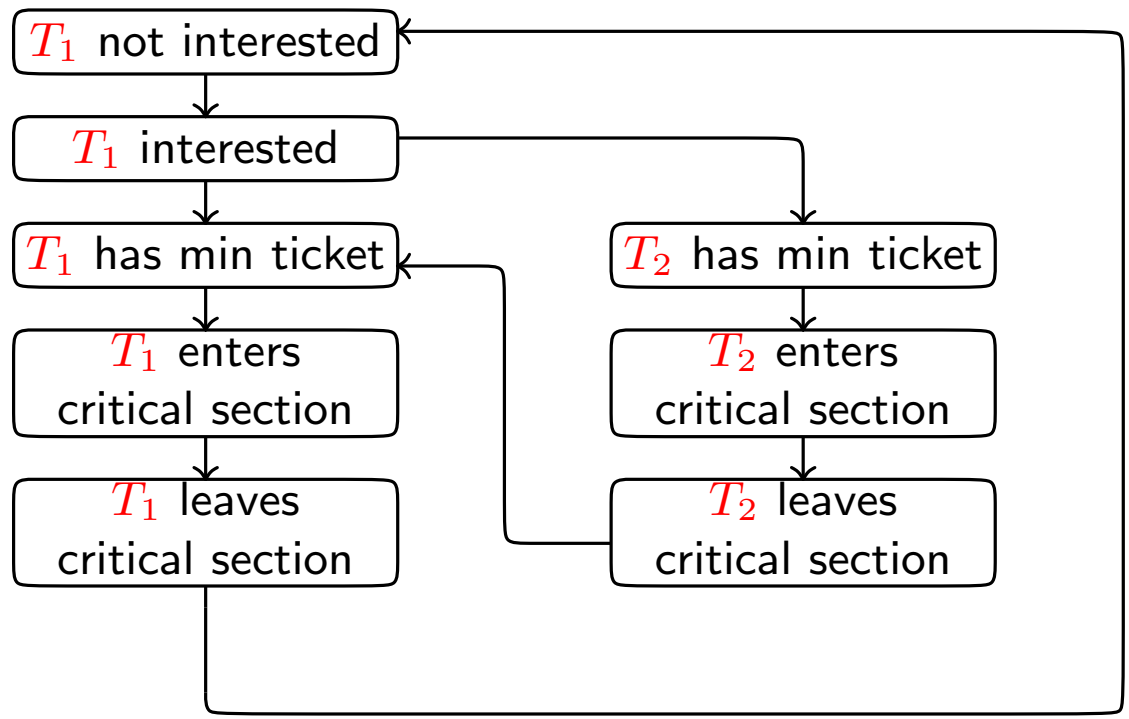


The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$

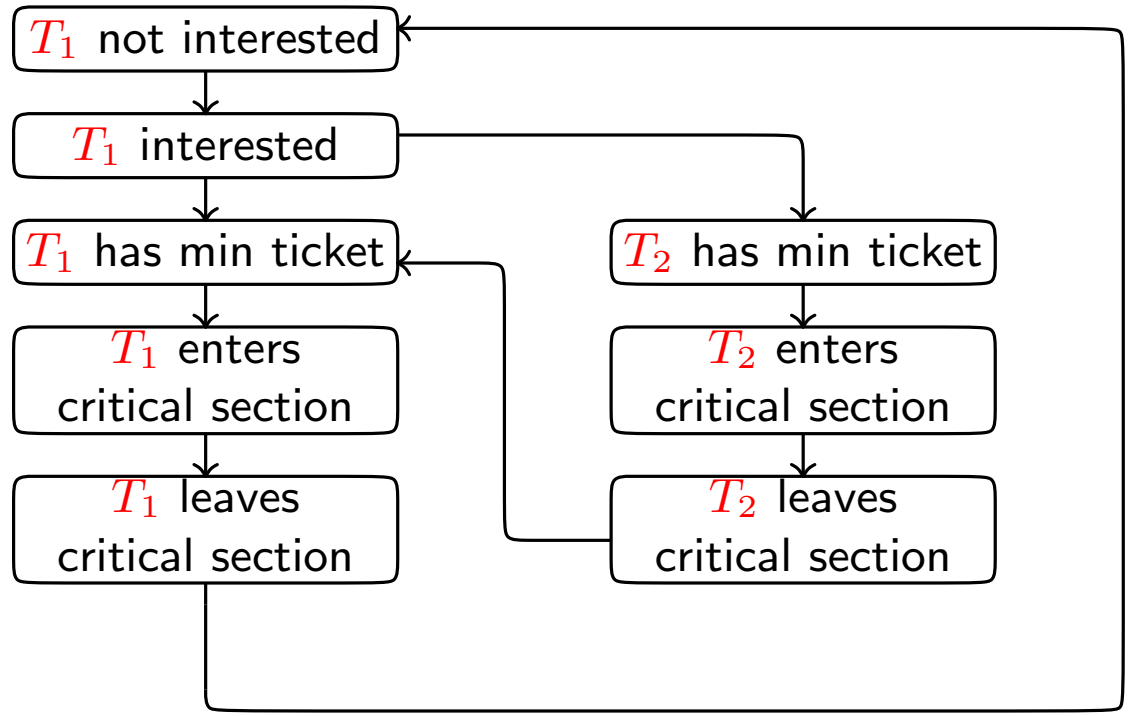
```
global
  int avail := 0
  set⟨int, tid⟩ bag := ∅

procedure SetMutex
  int ticket
begin
1: loop
2:   noncritical
3:   ⟨ ticket := avail++
     ⟩ bag.add(ticket, myId)
4:   await (bag.min == ticket)
5:   critical
6:   bag.remove(ticket, myId)
7: end loop
end procedure
```



The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)$

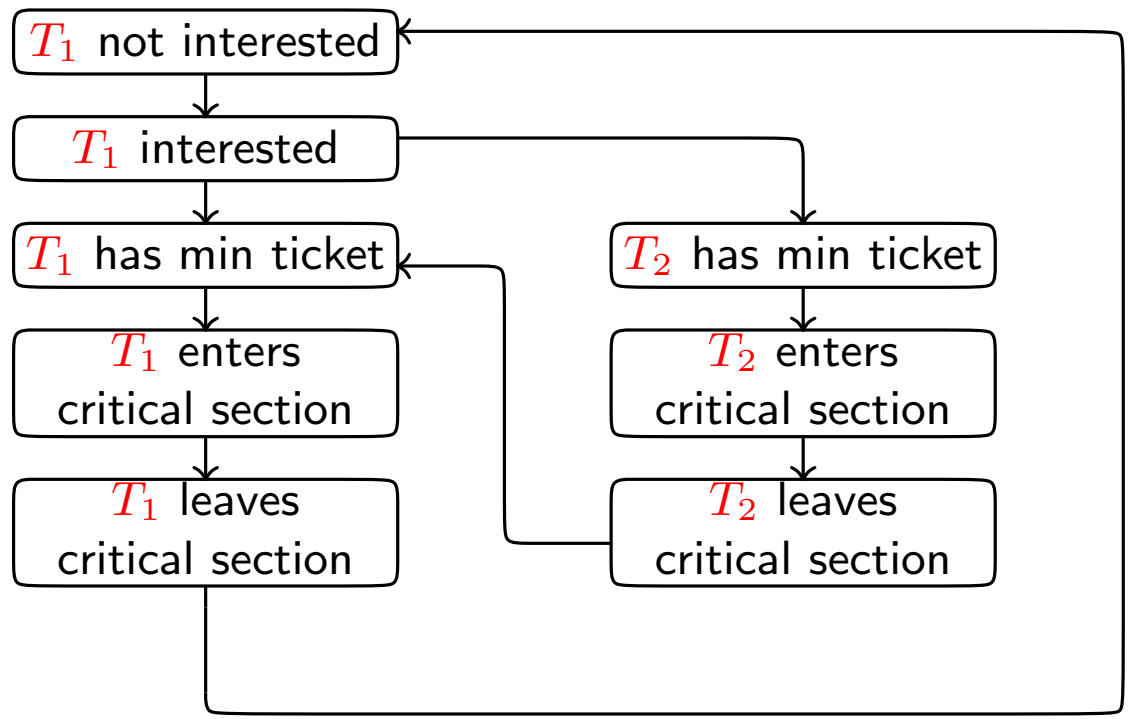
Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$



The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)||\textsc{SetMutex}(T_3)$

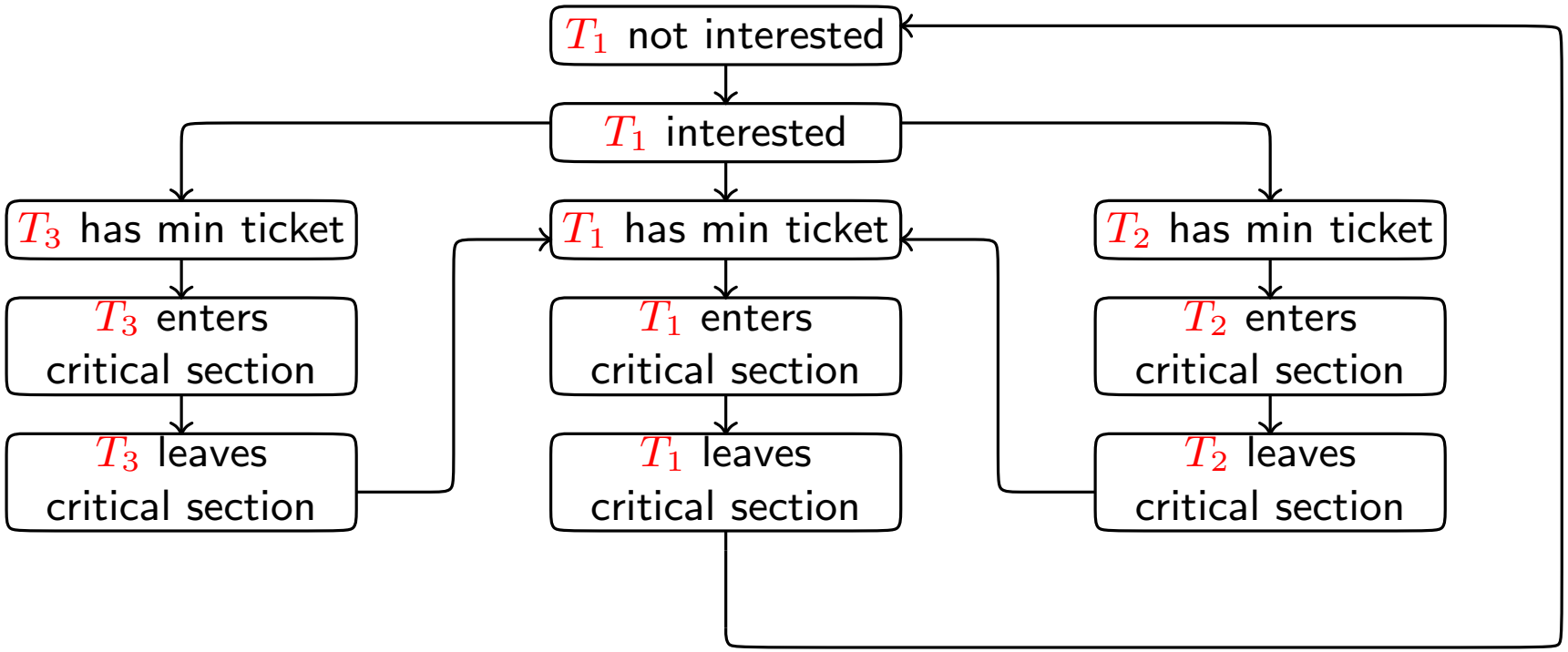
Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$



The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)||\textsc{SetMutex}(T_3)$

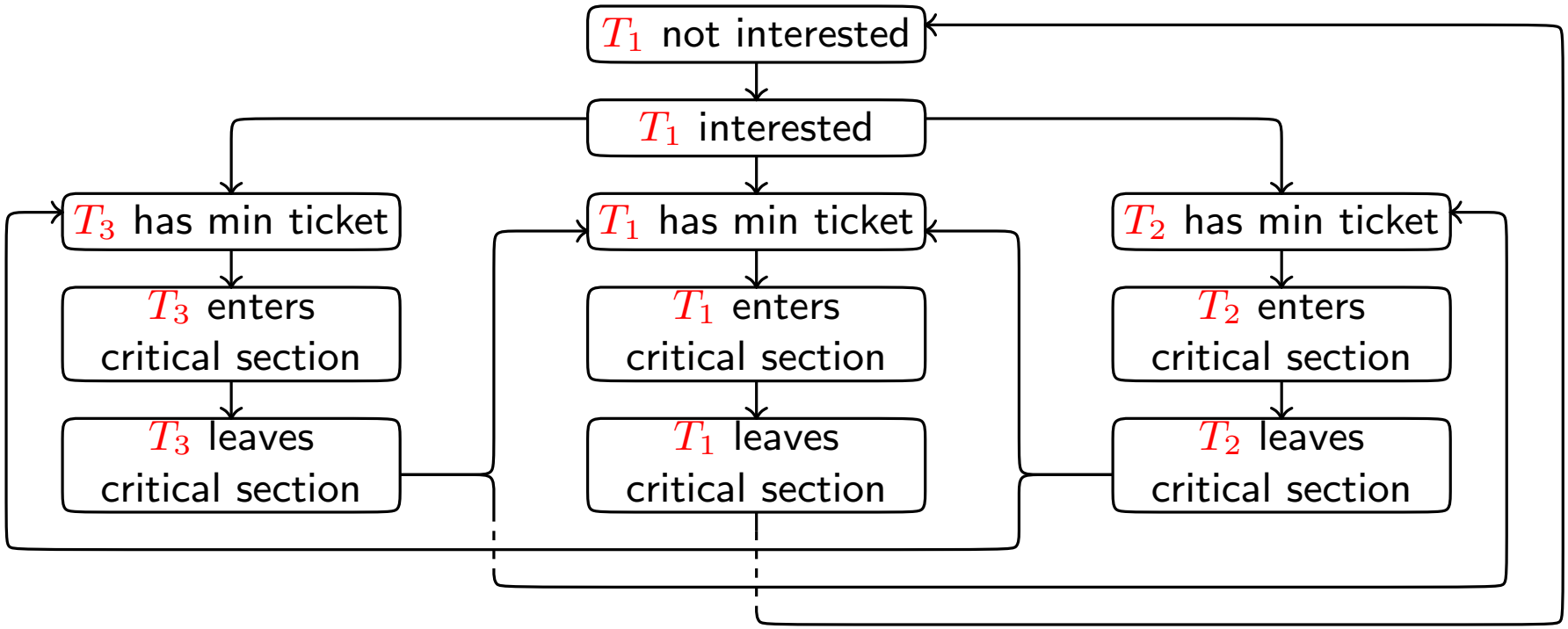
Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$



The Need of Parametrized Diagrams

System: $\textsc{SetMutex}(T_1)||\textsc{SetMutex}(T_2)||\textsc{SetMutex}(T_3)$

Spec: $\Box\left(pc_{T_1} = 3 \rightarrow \Diamond pc_{T_1} = 5\right)$



Solution: Parametrized Verification Diagrams

Problem

- **Not a single diagram** for arbitrary number of threads
- **Unbounded** number of **verification conditions**

Solution: Parametrized Verification Diagrams

Problem

- **Not a single diagram** for arbitrary number of threads
- **Unbounded** number of **verification conditions**

Our solution

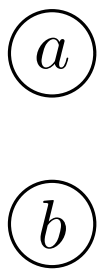
- **Unique diagram** for arbitrary number of threads
- **Finite and bounded** number of verification conditions

Parametrized Verification Diagrams exploit **symmetry**

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD

Nodes

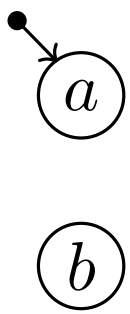


Parametrized VD

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD

Initial Node

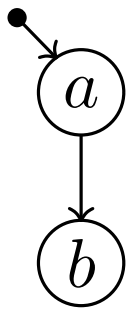


Parametrized VD

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD

Edges

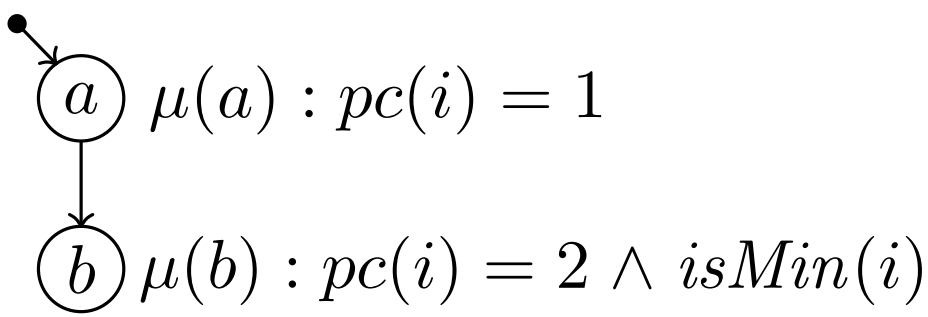


Parametrized VD

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD

Node labeling

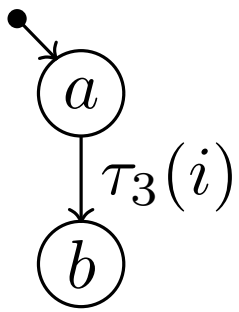


Parametrized VD

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD

Edge labeling

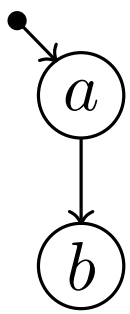


Parametrized VD

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD

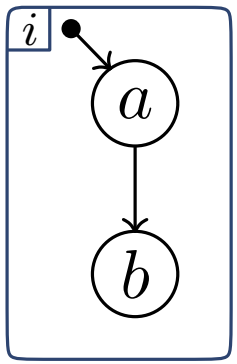
Ranking functions



Parametrized VD

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

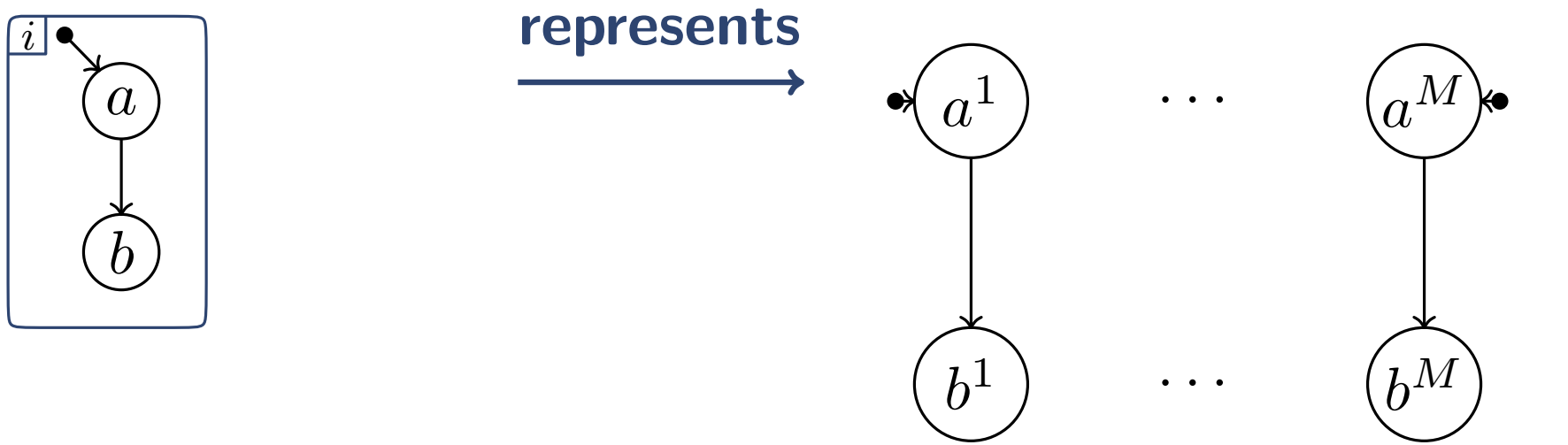


Parametrized VD

Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

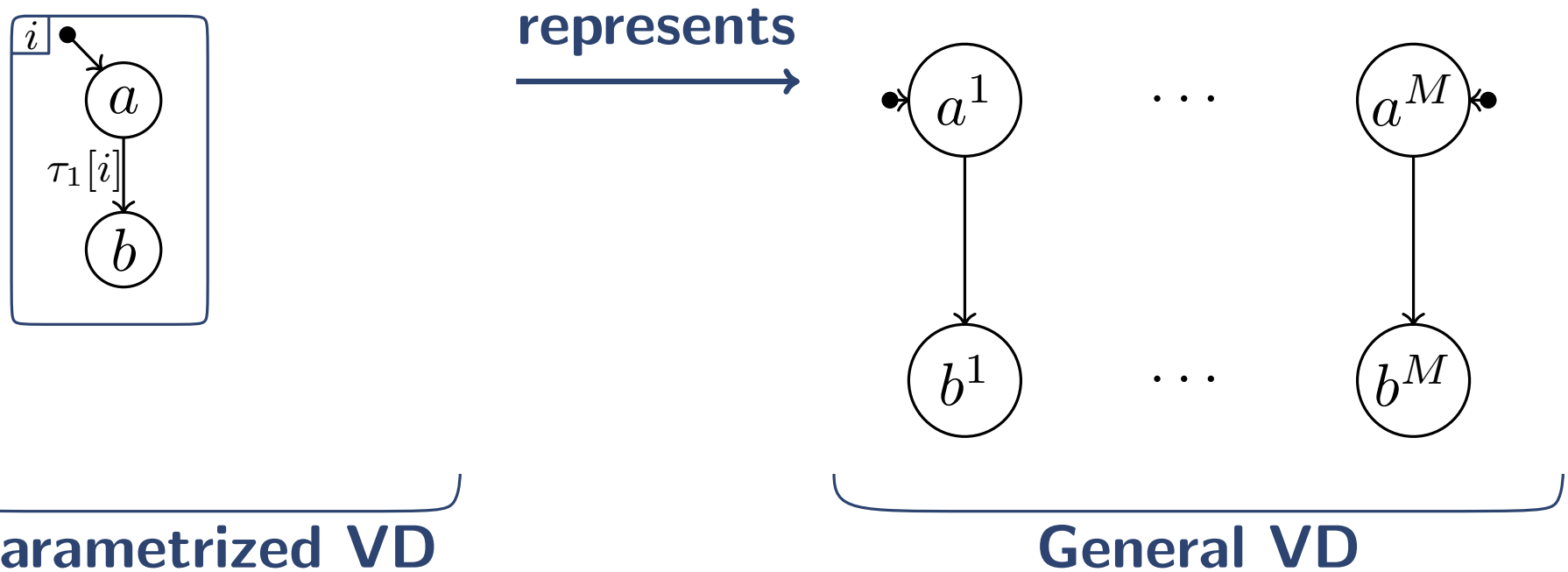
For M threads



Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

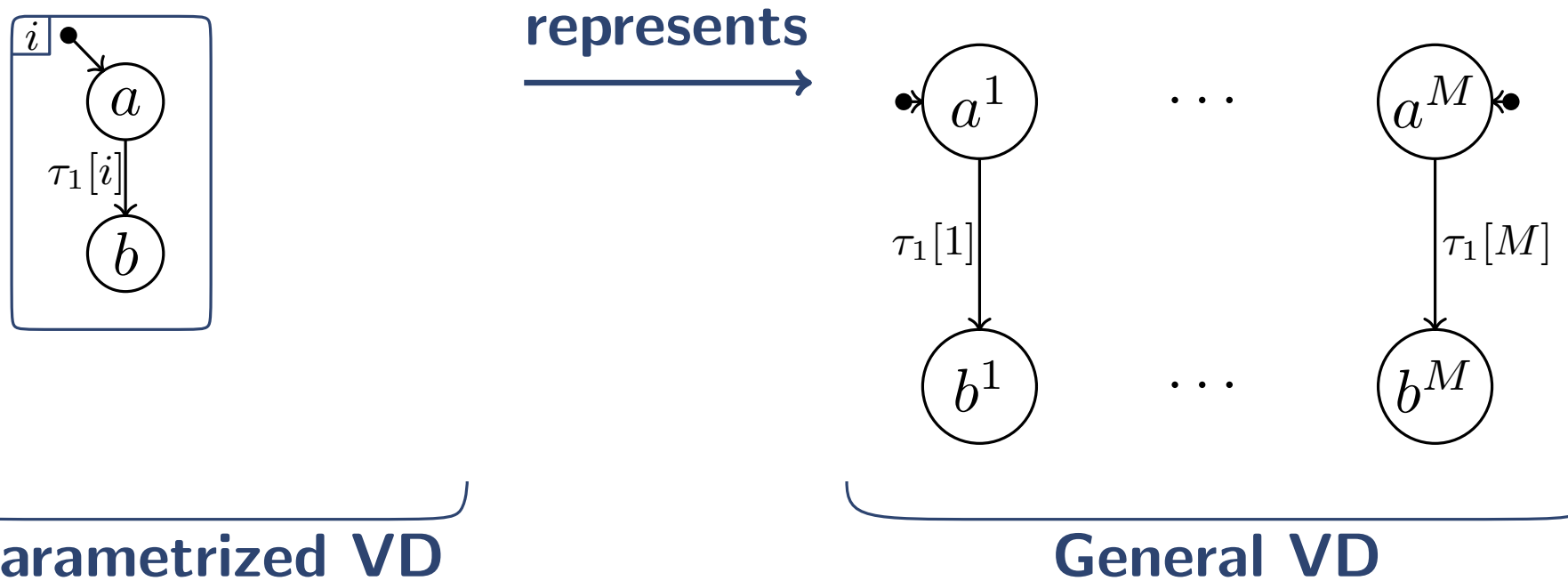
For M threads



Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

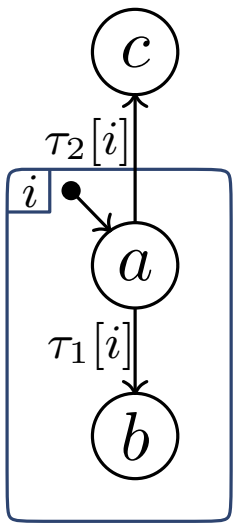
For M threads



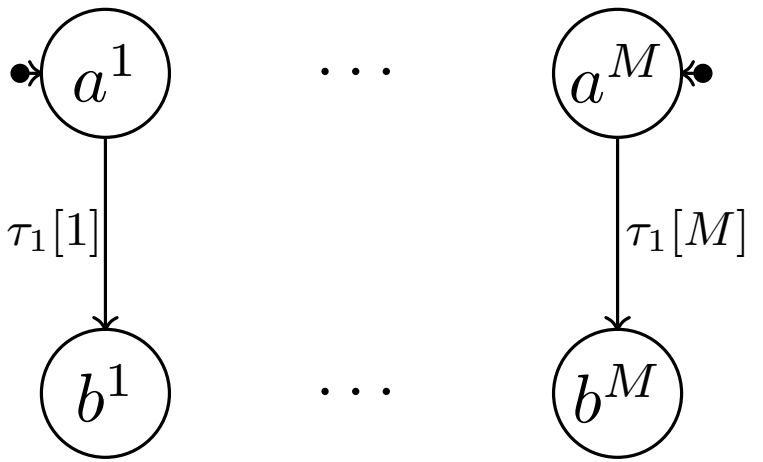
Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

For M threads



represents

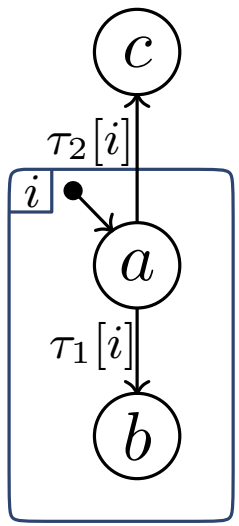


Parametrized VD **General VD**

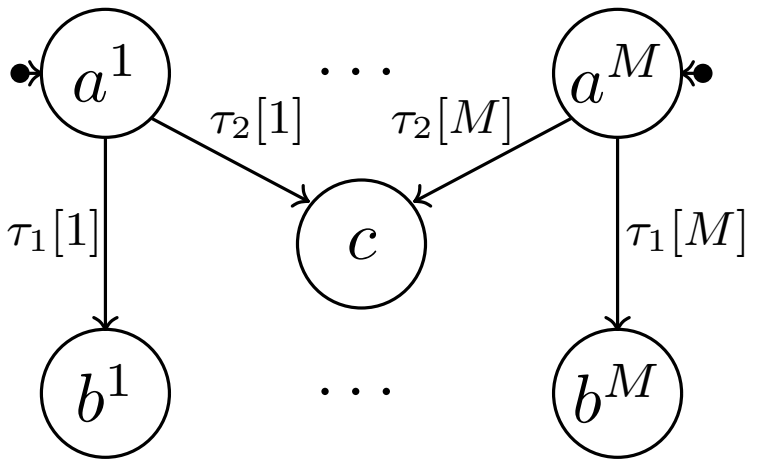
Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

For M threads



represents



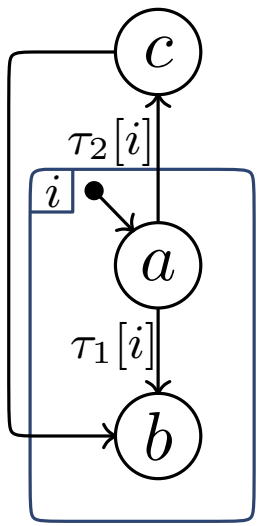
Parametrized VD

General VD

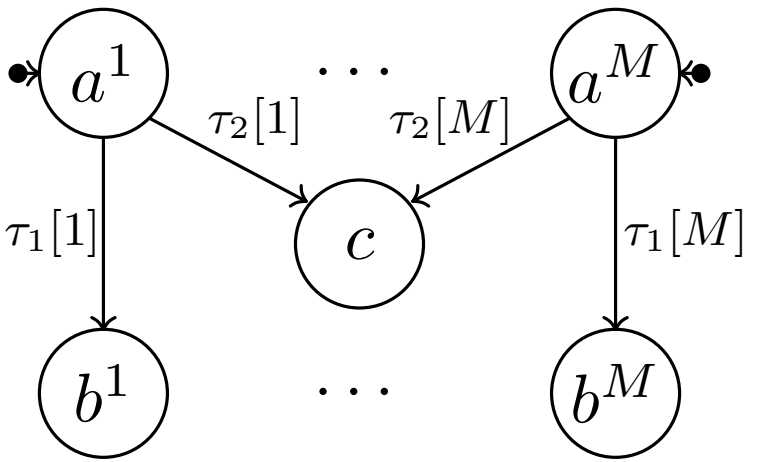
Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

For M threads



represents



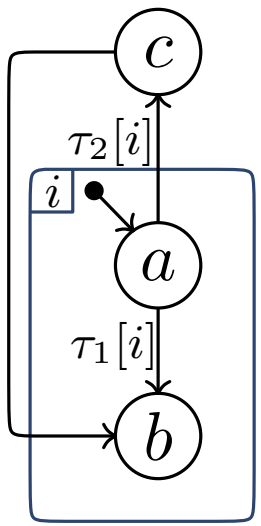
Parametrized VD

General VD

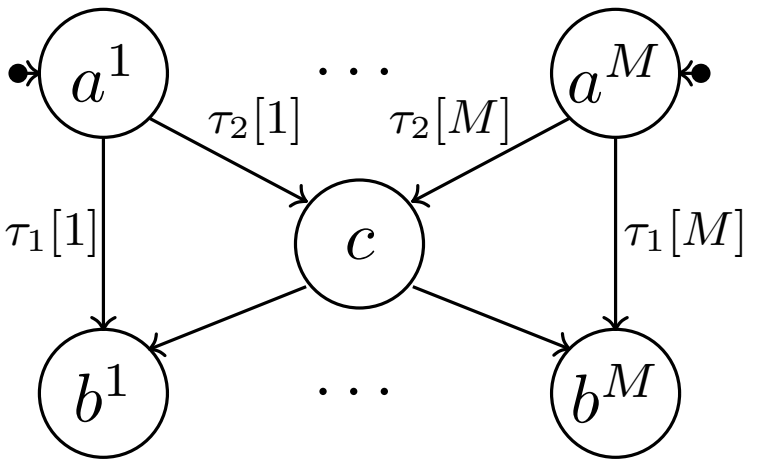
Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

For M threads



represents



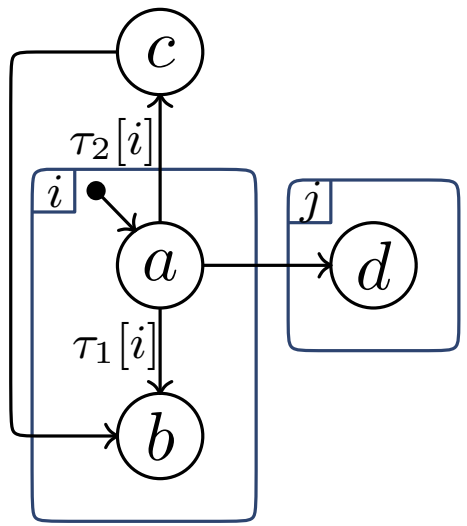
Parametrized VD

General VD

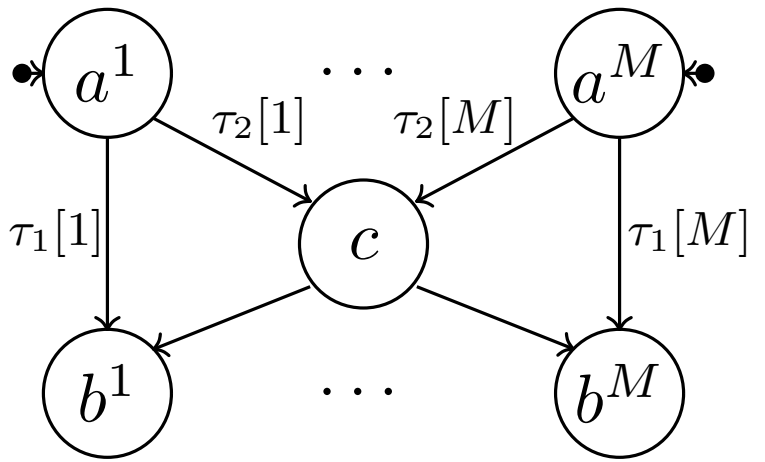
Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

For M threads



represents

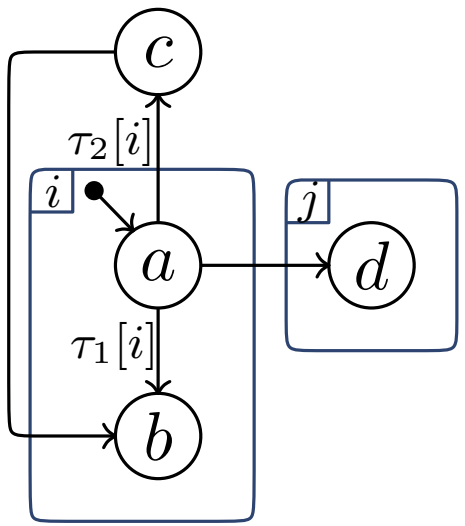


Parametrized VD **General VD**

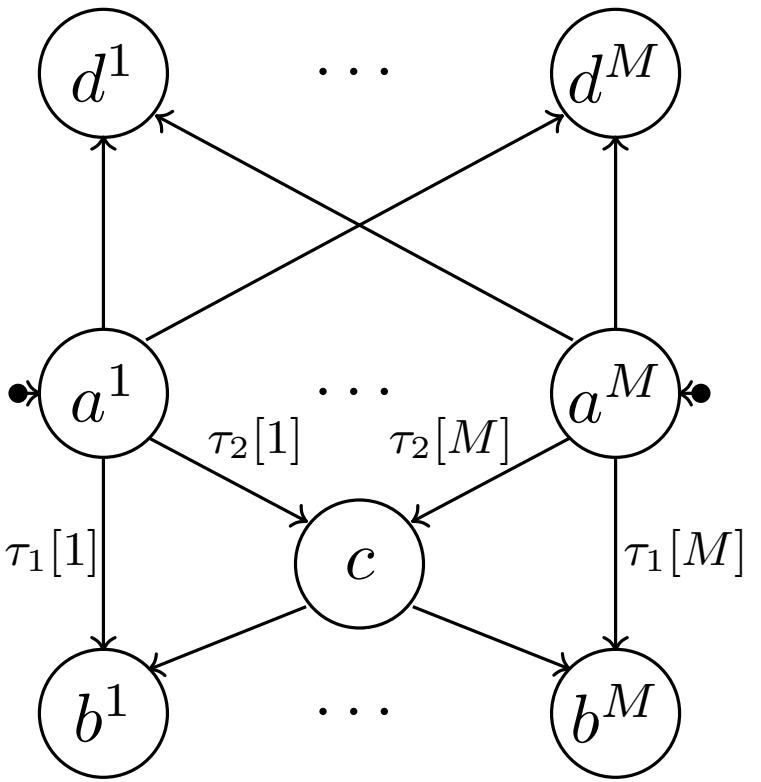
Parametrized Verification Diagrams: Sketch

- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

For M threads



represents



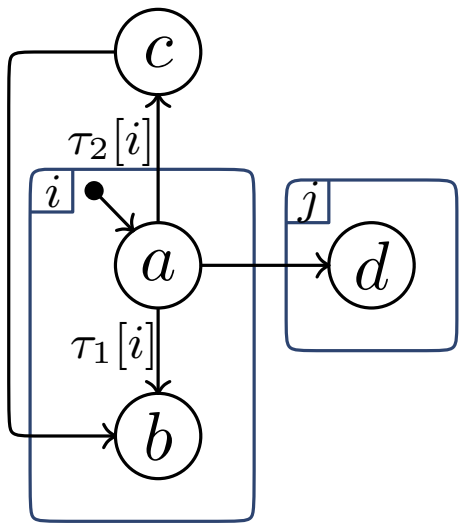
Parametrized VD **General VD**

Parametrized Verification Diagrams: Sketch

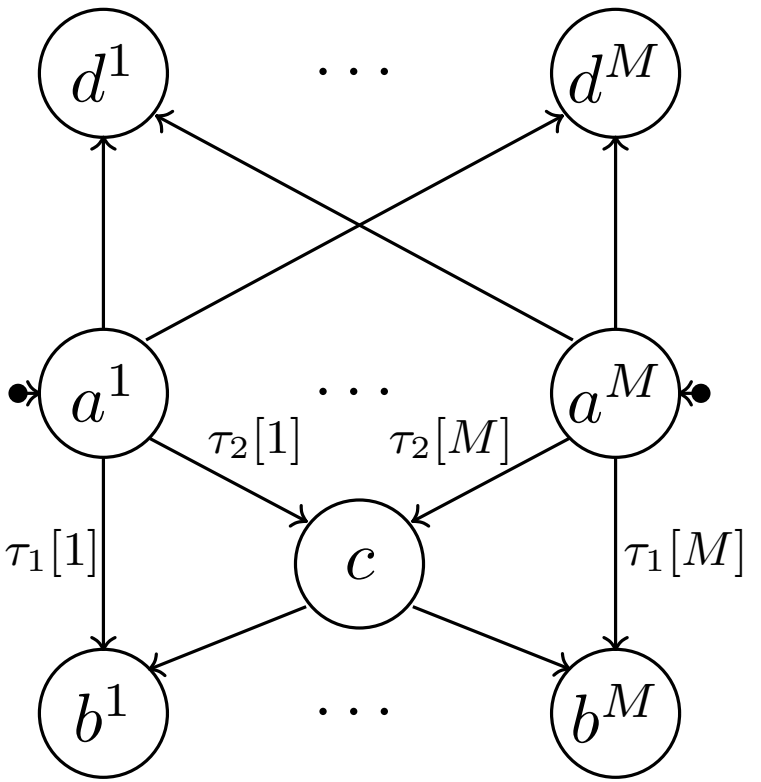
- PVDs are **an extension** of GVD
- Includes de notion of **boxes**

A PVD **abstracts all instantiations** of a parametric system

For M threads

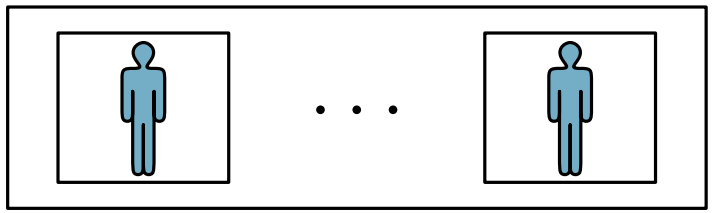


represents



Parametrized VD **General VD**

Verification Conditions for Parametrized Diagrams



$P[N] : P(1)|| \cdots ||P(N)$

Diagram Property

$\models \quad \mathcal{D} \quad \models \quad \varphi^{(k)}$

Verification Conditions:

- Initiation
- Consecution
- Acceptance
- Fairness

Verification Conditions for Parametrized Diagrams

- **Initiation** $\Theta \rightarrow \mu(N_0)$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\bigvee_{n \to m} \mu(n) \wedge \tau(i) \;\rightarrow\; \mu(m')$$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\bigvee_{n \to m} \mu(n) \wedge \tau(i) \;\rightarrow\; \mu(m')$$

Self-Consecution

$$\bigvee_{n \to m} \mu(n) \wedge \tau(i) \;\rightarrow\; \mu(m') \qquad \text{for all } i \in Voc(n, m)$$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\bigvee_{n \to m} \mu(n) \wedge \tau(i) \;\rightarrow\; \mu(m')$$

Self-Consecution

$$\bigvee_{n \to m} \mu(n) \wedge \tau(i) \;\rightarrow\; \mu(m') \qquad \text{for all } i \in Voc(n, m)$$

Others-Consecution

$$\bigvee_{n \to m} \mu(n) \wedge \tau(j) \wedge j \neq i \;\rightarrow\; \mu(m') \qquad \text{for fresh } j \notin Voc(n, m)$$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution:** For every $n \in N$ and $\tau \in \mathcal{T}$,

$$\bigvee_{n \to m} \mu(n) \wedge \tau(i) \;\rightarrow\; \mu(m')$$

How many VCs?

$|\mathcal{T}| \cdot |\mathcal{V}|$

Self-Consecution

$$\bigvee_{n \to m} \mu(n) \wedge \tau(i) \;\rightarrow\; \mu(m')$$

for all $i \in Voc(n, m)$

$|\mathcal{T}|$

Others-Consecution

$$\bigvee_{n \to m} \mu(n) \wedge \tau(j) \wedge j \neq i \;\rightarrow\; \mu(m')$$

for fresh $j \notin Voc(n, m)$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution**
- **Acceptance:** For every (B, G, δ) and all edges $n \to_e m$:

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution**
- **Acceptance:** For every (B, G, δ) and all edges $n \to_e m$:

Self-Acceptance (for all $i \in Voc(n, m)$)

$$(\mu(n) \wedge \tau(i) \wedge \mu(m')) \to \delta(n) > \delta(m) \qquad \text{if } e \in B$$

$$(\mu(n) \wedge \tau(i) \wedge \mu(m')) \to \delta(n) \geq \delta(m) \qquad \text{if } e \in E \setminus (B \cup G)$$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution**
- **Acceptance:** For every (B, G, δ) and all edges $n \to_e m$:

Self-Acceptance (for all $i \in Voc(n, m)$)

$$(\mu(n) \wedge \tau(i) \wedge \mu(m')) \to \delta(n) > \delta(m) \quad \text{if } e \in B$$

$$(\mu(n) \wedge \tau(i) \wedge \mu(m')) \to \delta(n) \geq \delta(m) \quad \text{if } e \in E \setminus (B \cup G)$$

Others-Acceptance (for fresh $i \notin Voc(n, m)$)

$$(\mu(n) \wedge \tau(j) \wedge i \neq j \wedge \mu(m')) \to \delta(n) > \delta(m) \quad \text{if } e \in B$$

$$(\mu(n) \wedge \tau(j) \wedge i \neq j \wedge \mu(m')) \to \delta(n) \geq \delta(m) \quad \text{if } e \in E \setminus (B \cup G)$$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution**
- **Acceptance**
- **Fairness** For each $n \rightarrow_e m$ with $\eta(e) = \tau(i)$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution**
- **Acceptance**
- **Fairness** For each $n \rightarrow_e m$ with $\eta(e) = \tau(i)$

$$\mu(n) \rightarrow En(\tau(i))$$

$$\mu(n) \wedge \tau(i) \rightarrow \mu(m')$$

Verification Conditions for Parametrized Diagrams

- **Initiation**
- **Consecution**
- **Acceptance**
- **Fairness**
- **Satisfaction**

$$\mu(n) \to f(n)$$

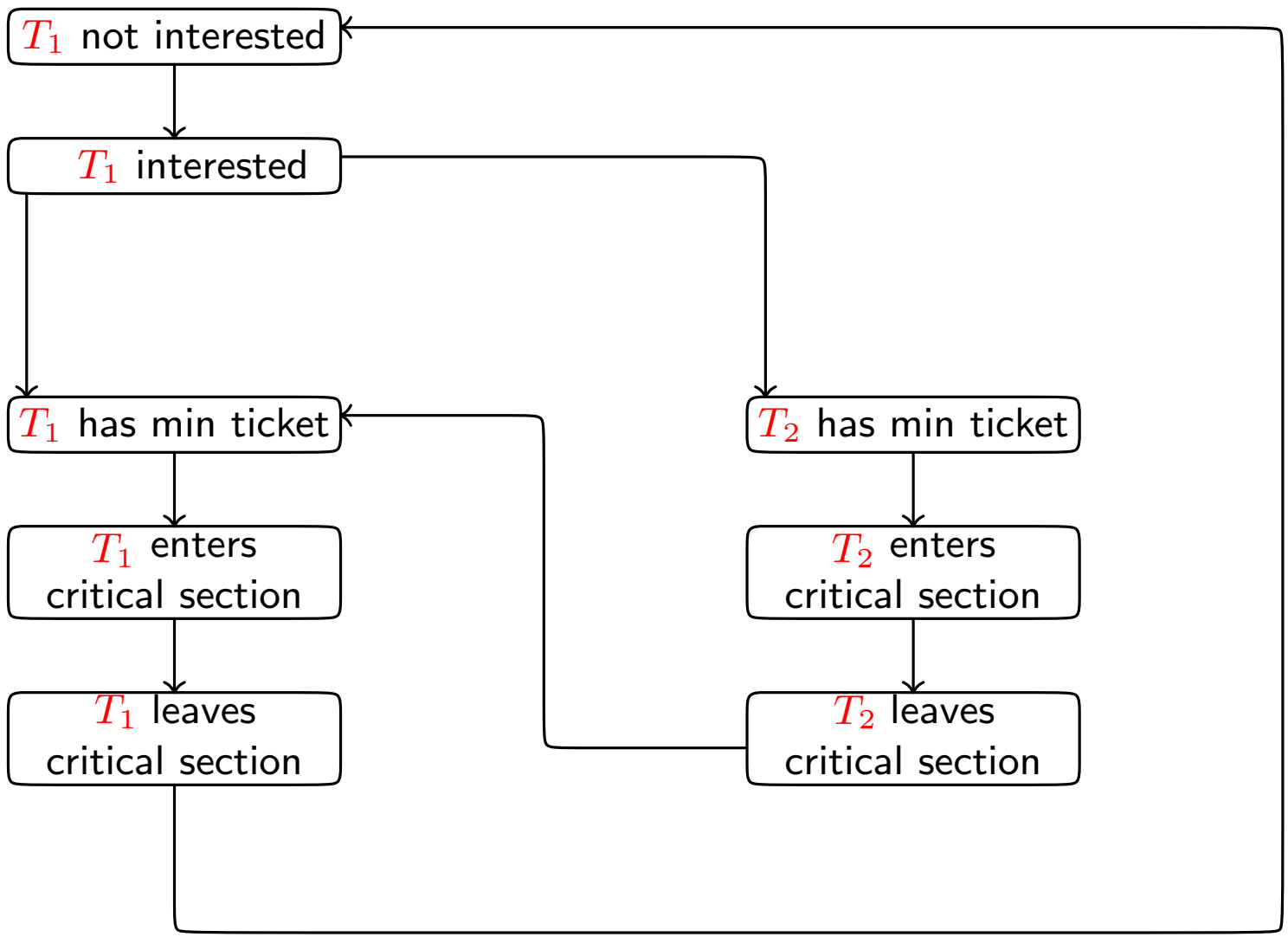
$$\mathsf{ModelCheck}(\mathcal{D} \models \varphi)$$

Mutual Exclusion Algorithm (revisited)

$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$

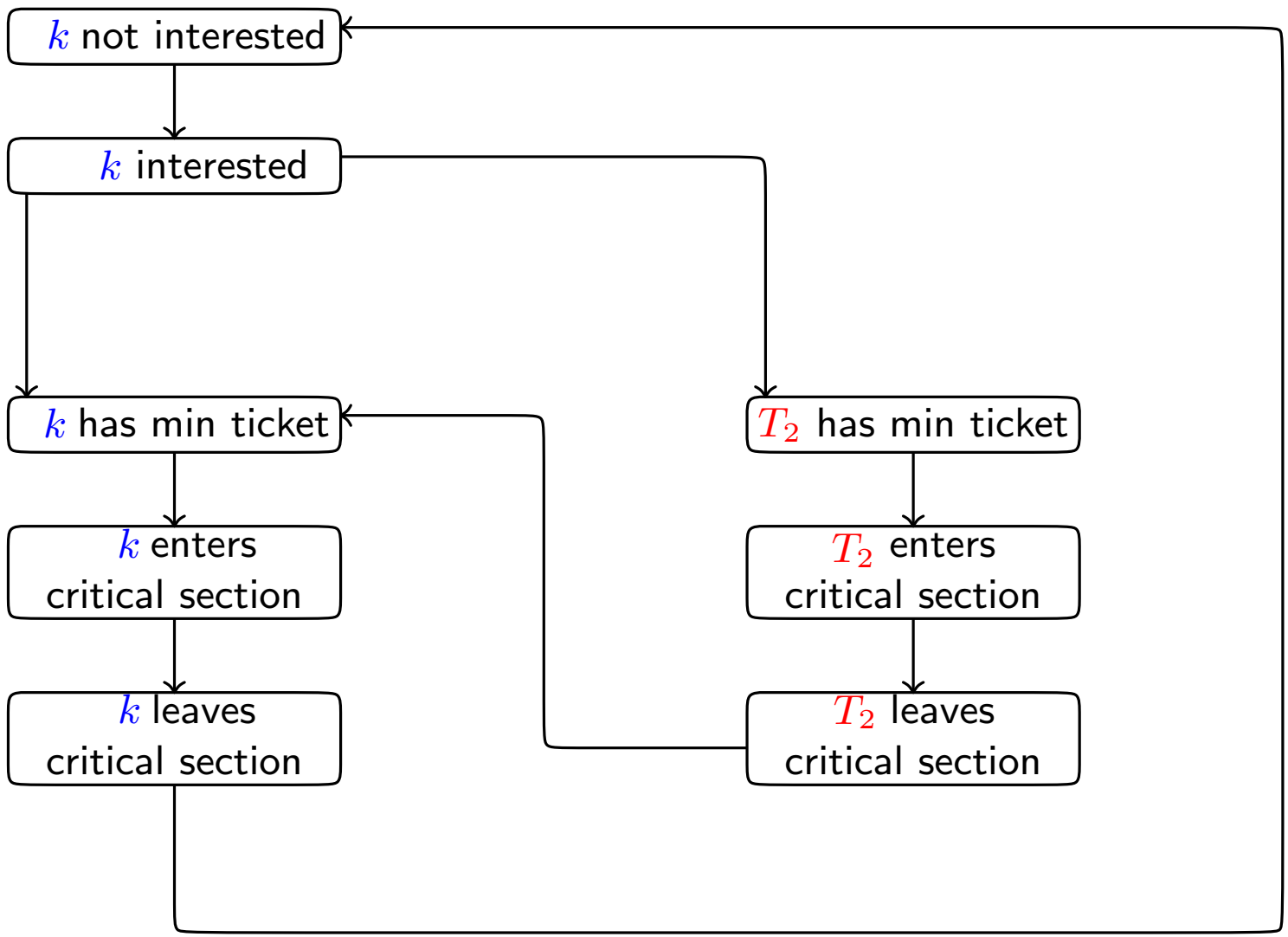
Mutual Exclusion Algorithm (revisited)

$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$



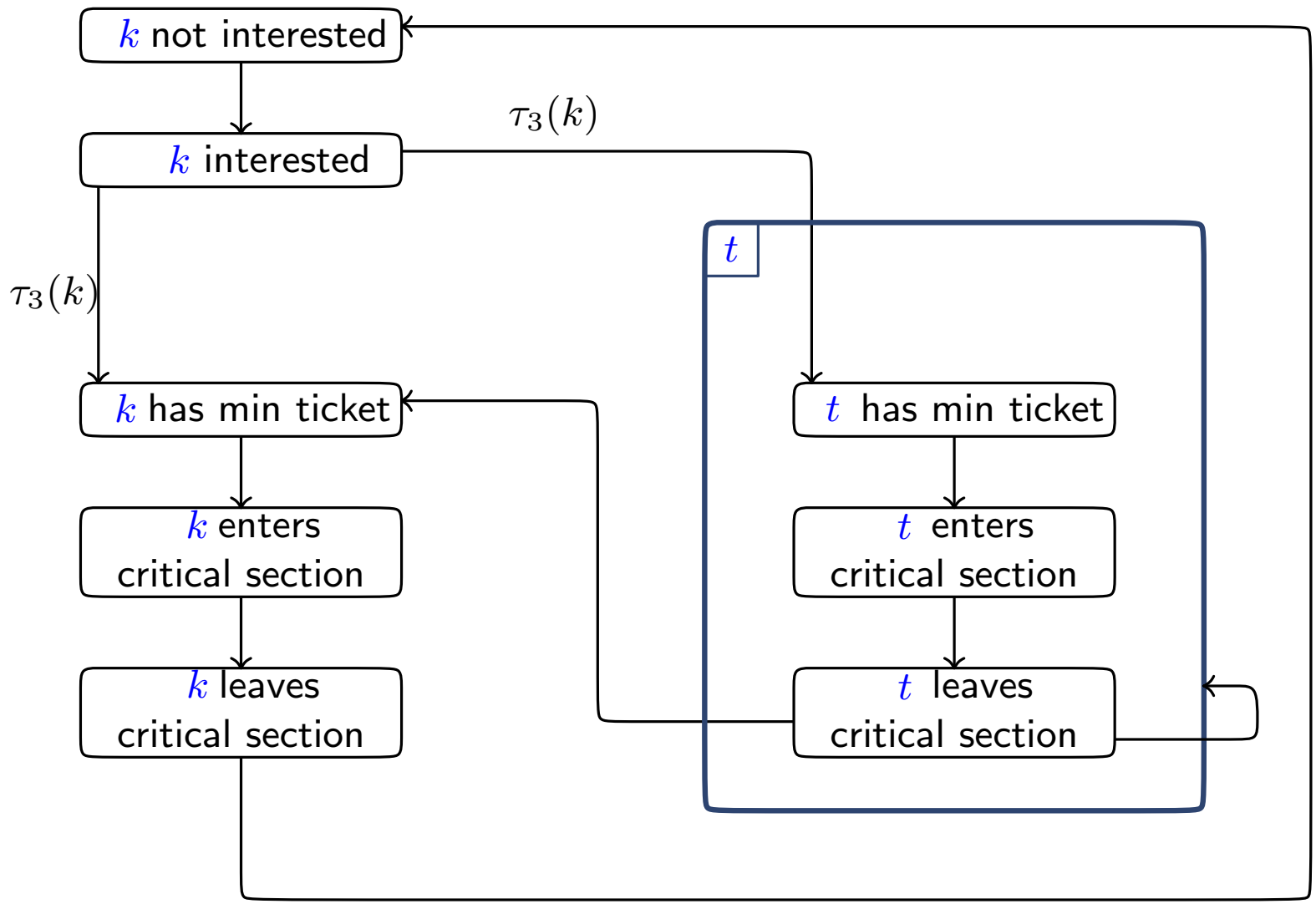
Mutual Exclusion Algorithm (revisited)

$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$



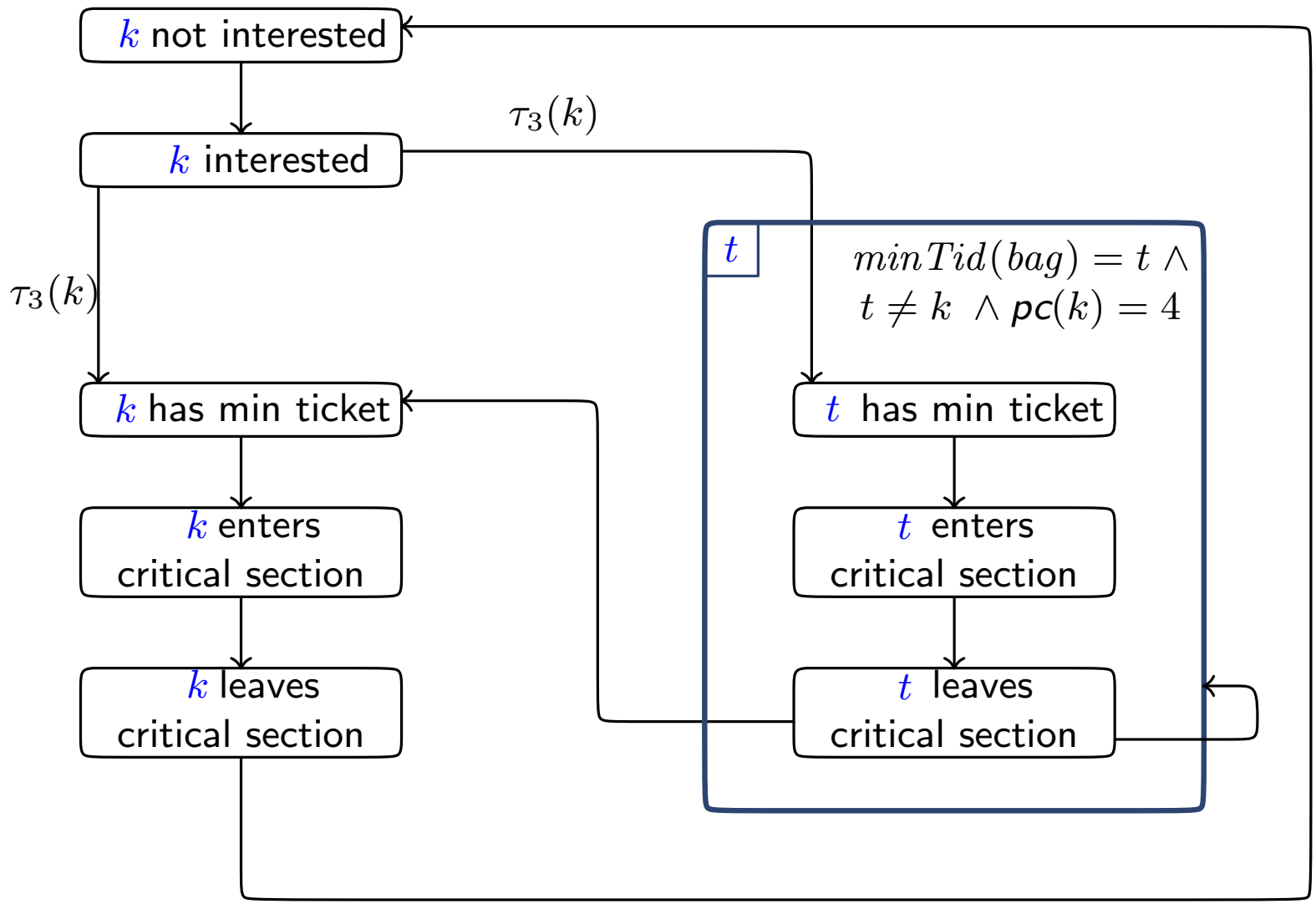
Mutual Exclusion Algorithm (revisited)

$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$



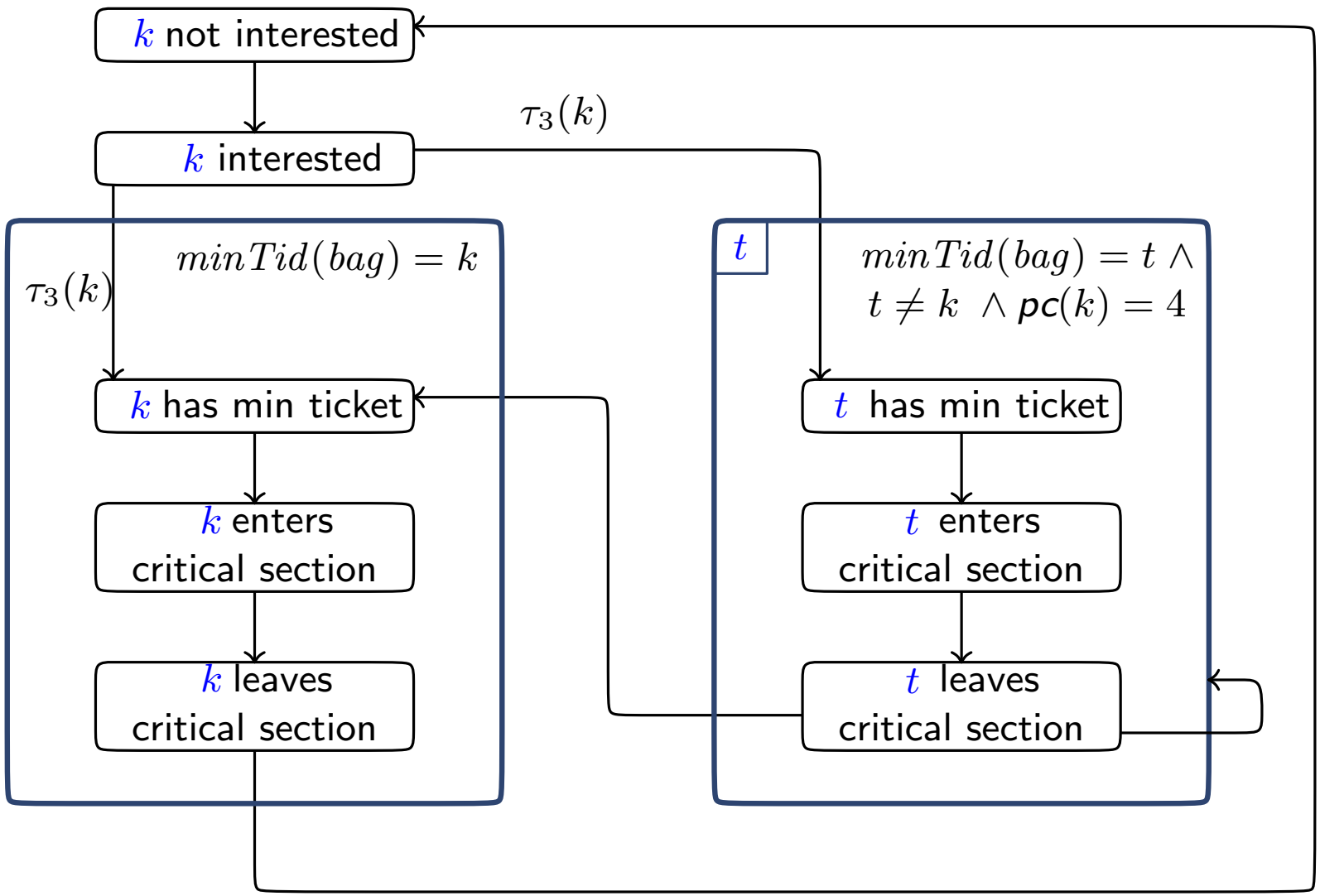
Mutual Exclusion Algorithm (revisited)

$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$



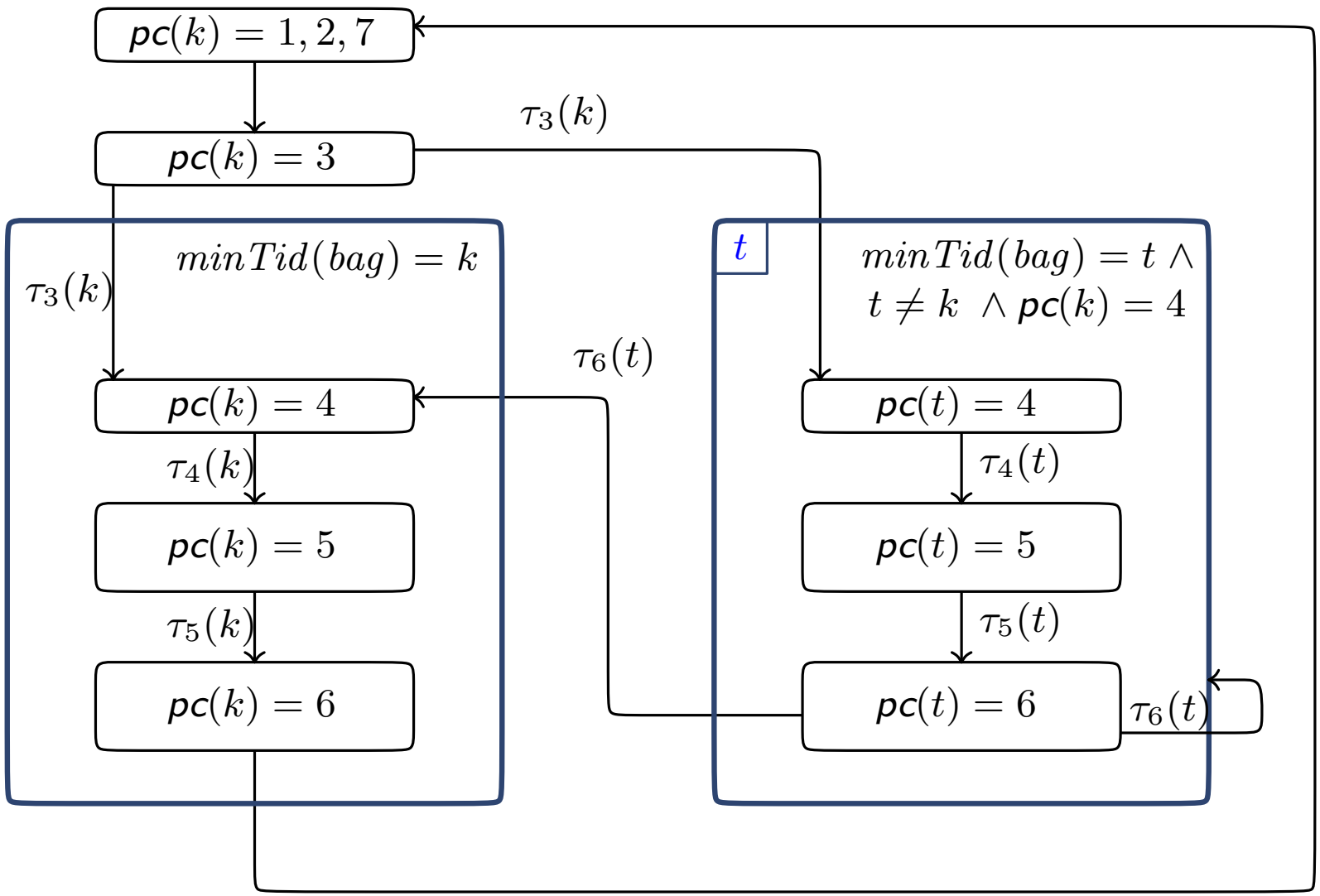
Mutual Exclusion Algorithm (revisited)

$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$



Mutual Exclusion Algorithm (revisited)

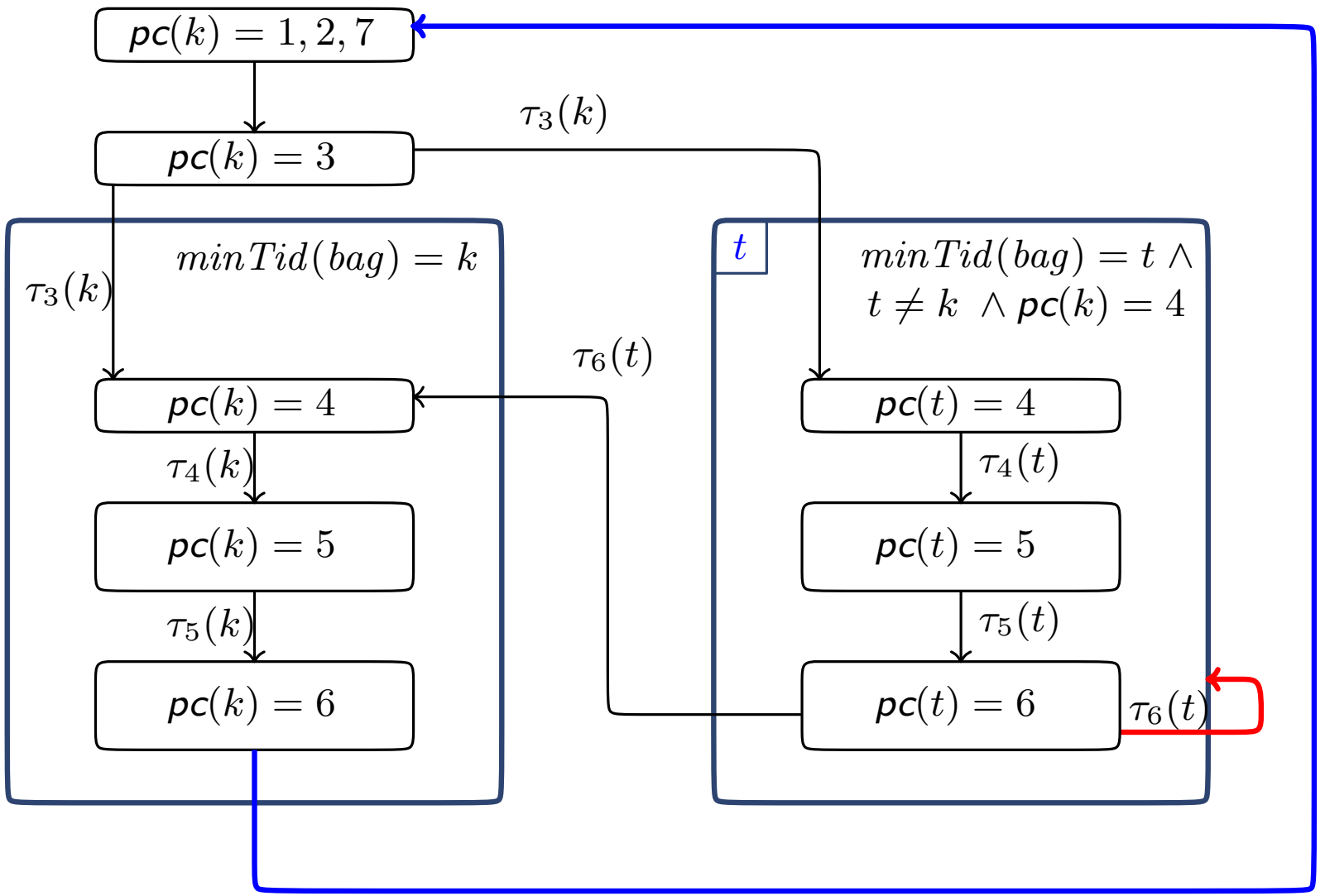
$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$



Mutual Exclusion Algorithm (revisited)

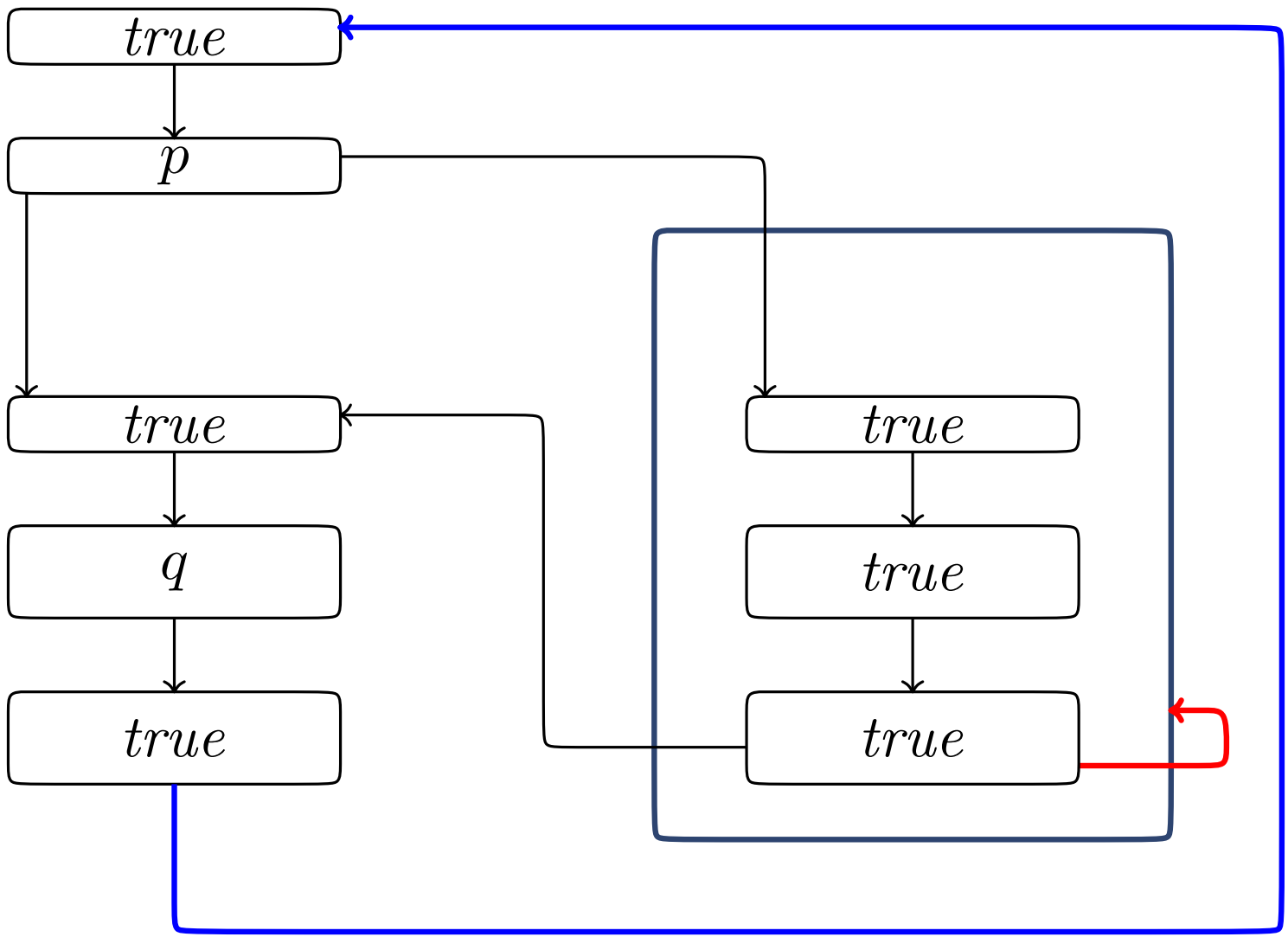
$$\varphi(k) \;=\; \Box(pc(k) = 3 \rightarrow \Diamond pc(k) = 5)$$

B, G $\quad \delta : lower(bag, ticket(k))$



Mutual Exclusion Algorithm (revisited)

$$\varphi(k) = \Box(\quad p \quad \rightarrow \Diamond \quad q \quad)$$



Our Contributions

A Deductive Verification Techniques for Parametrized Systems

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

3 Invariant Generation using Self-reflection
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

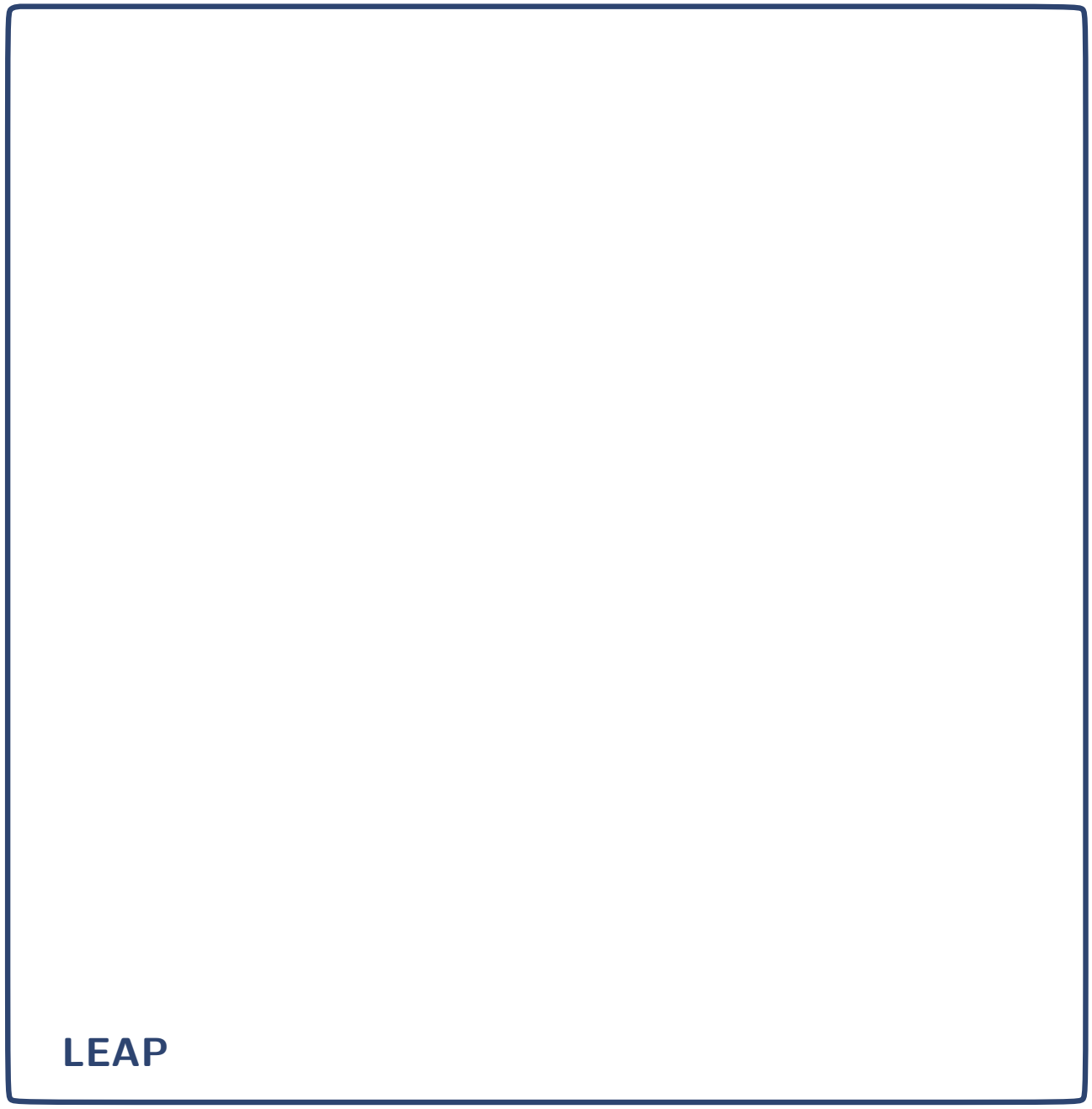
4 TL3: A Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

5 TSL_K: A Decidable Family for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

6 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

LEAP: Structure

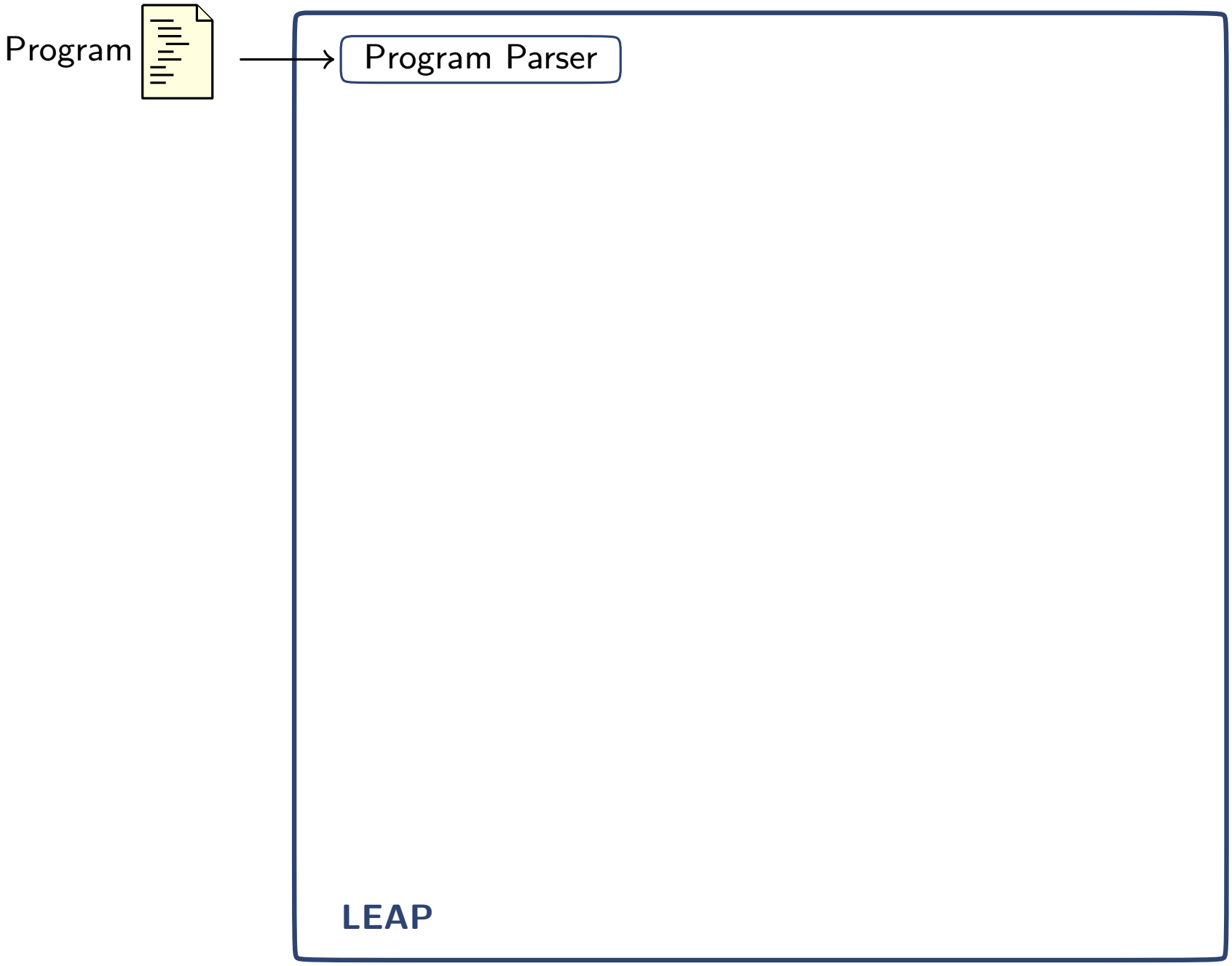


LEAP: Structure

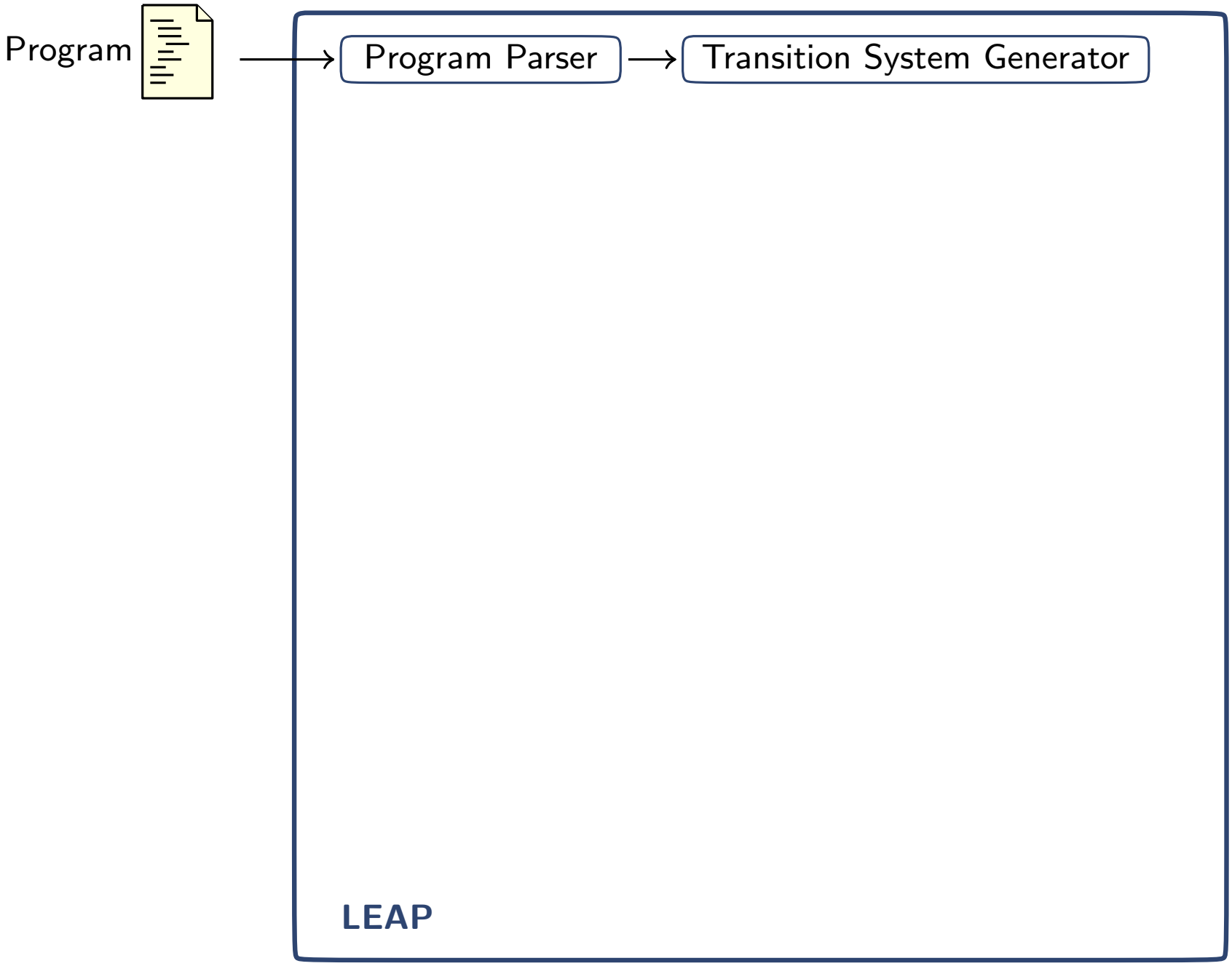
Program 

LEAP

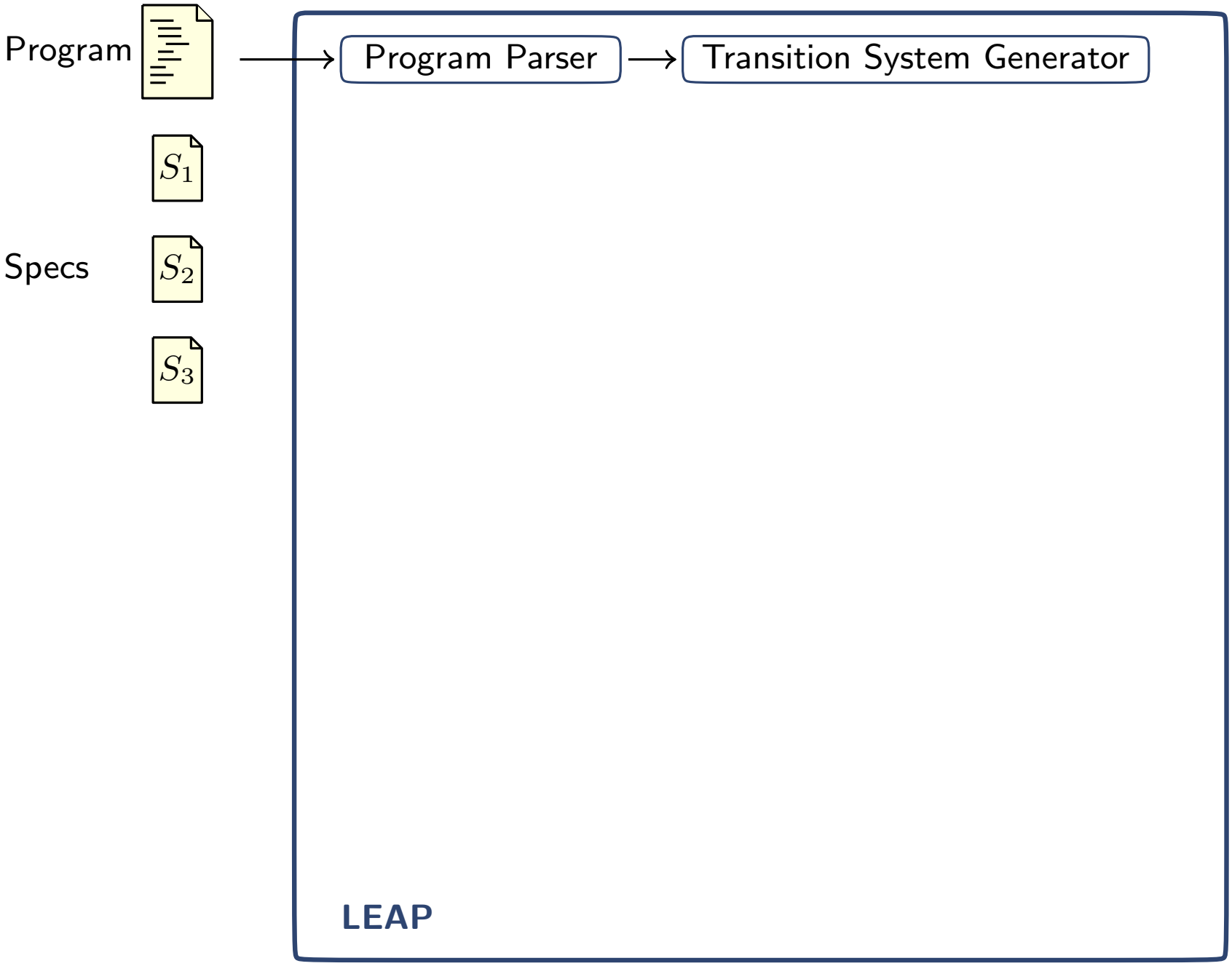
LEAP: Structure



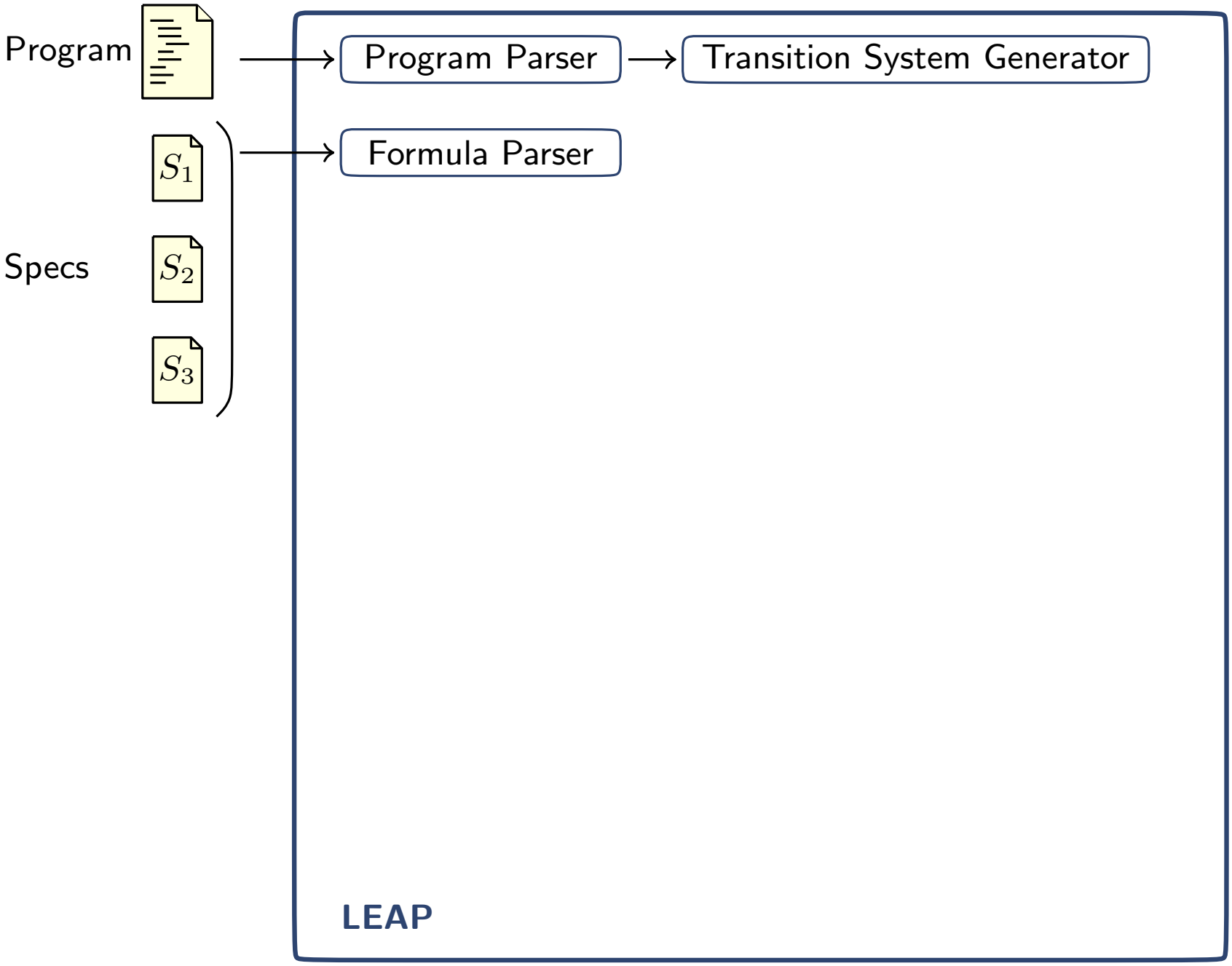
LEAP: Structure



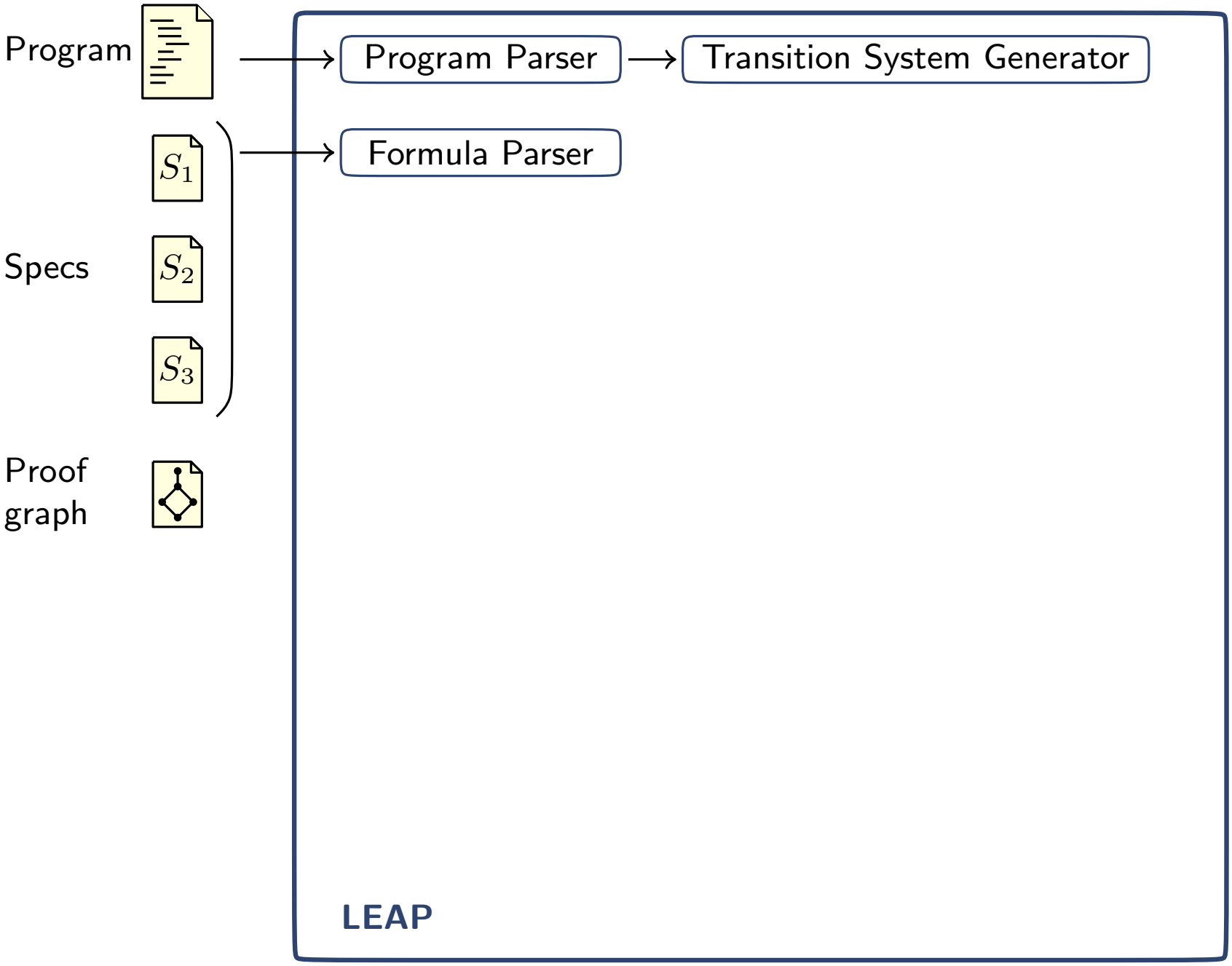
LEAP: Structure



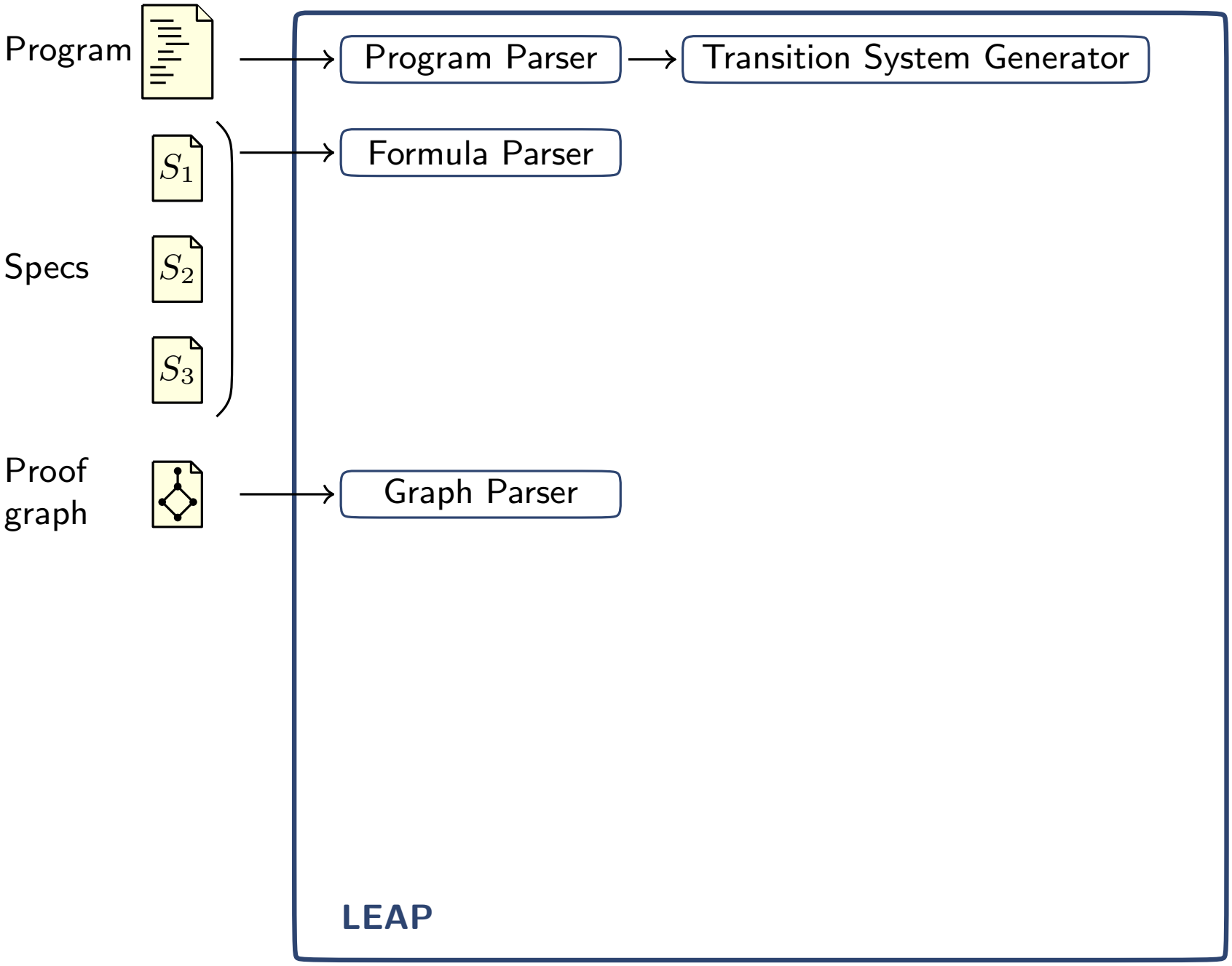
LEAP: Structure



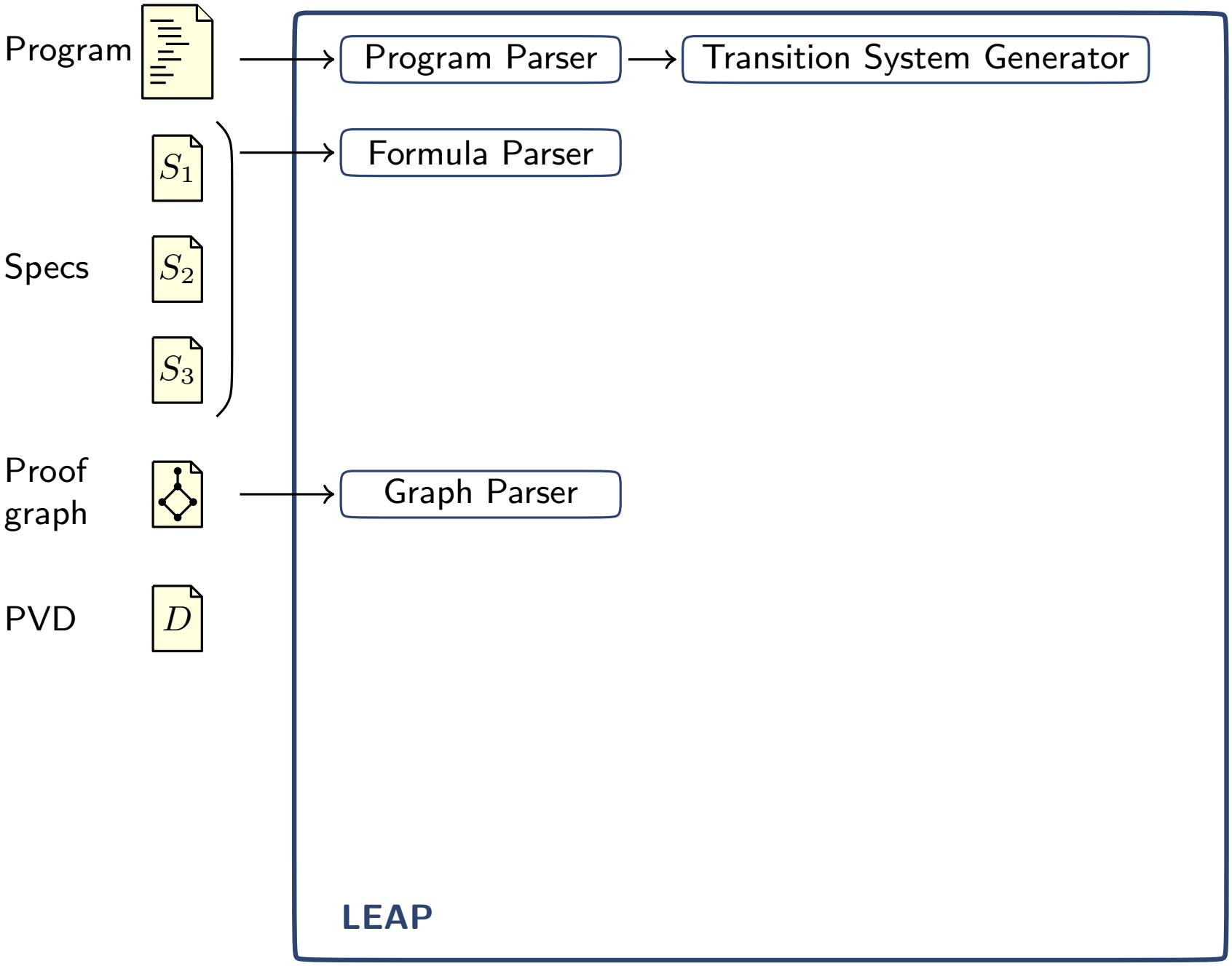
LEAP: Structure



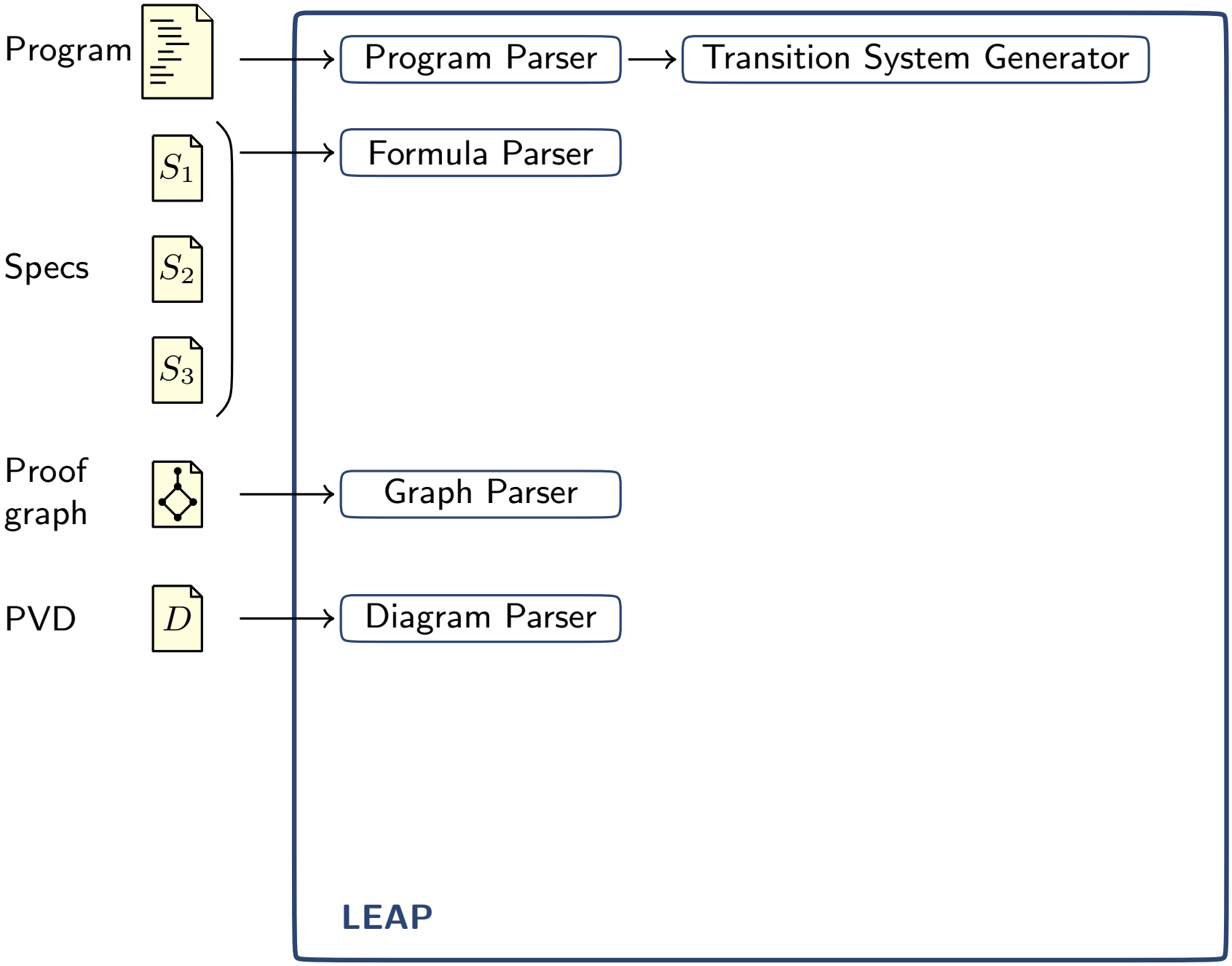
LEAP: Structure



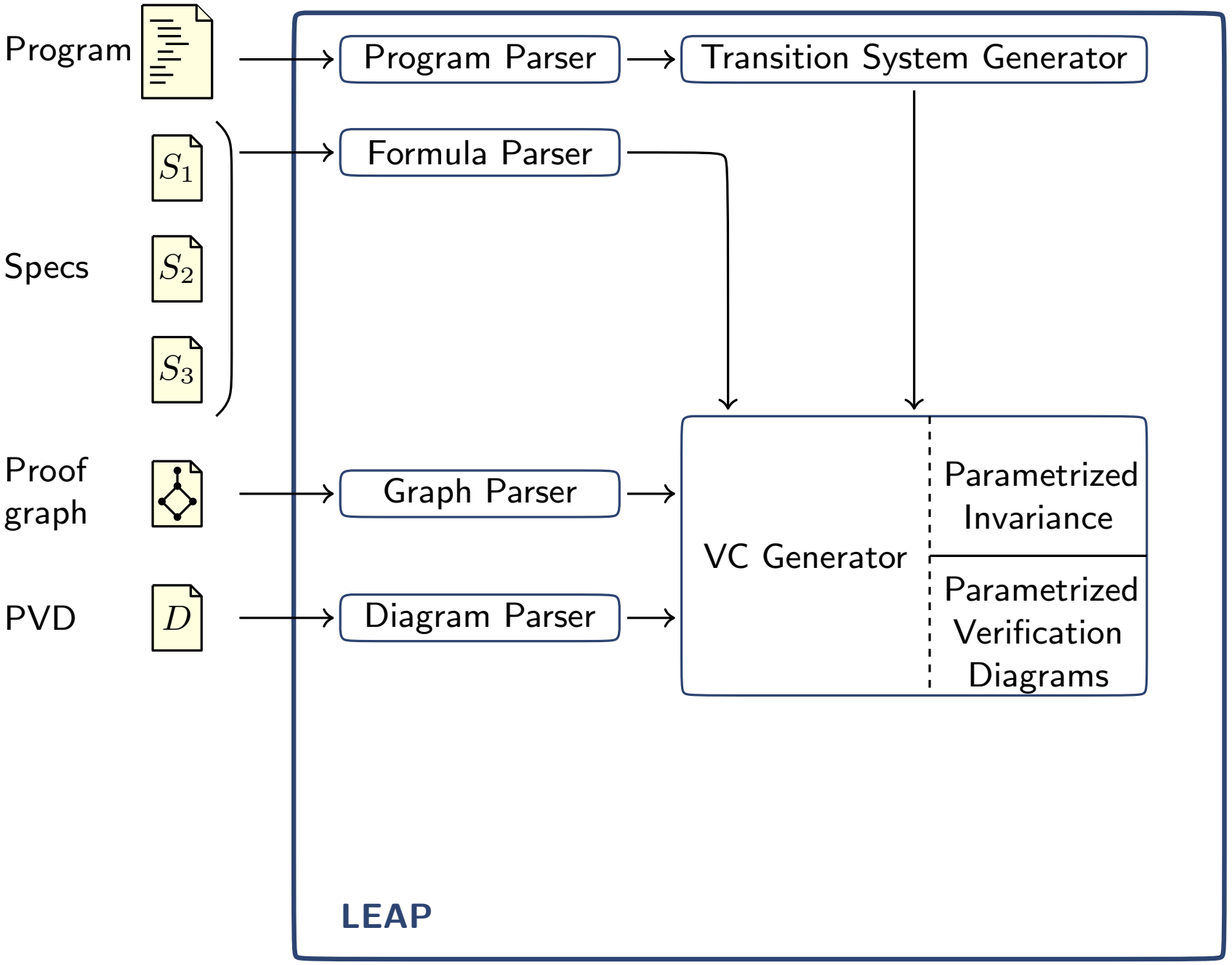
LEAP: Structure



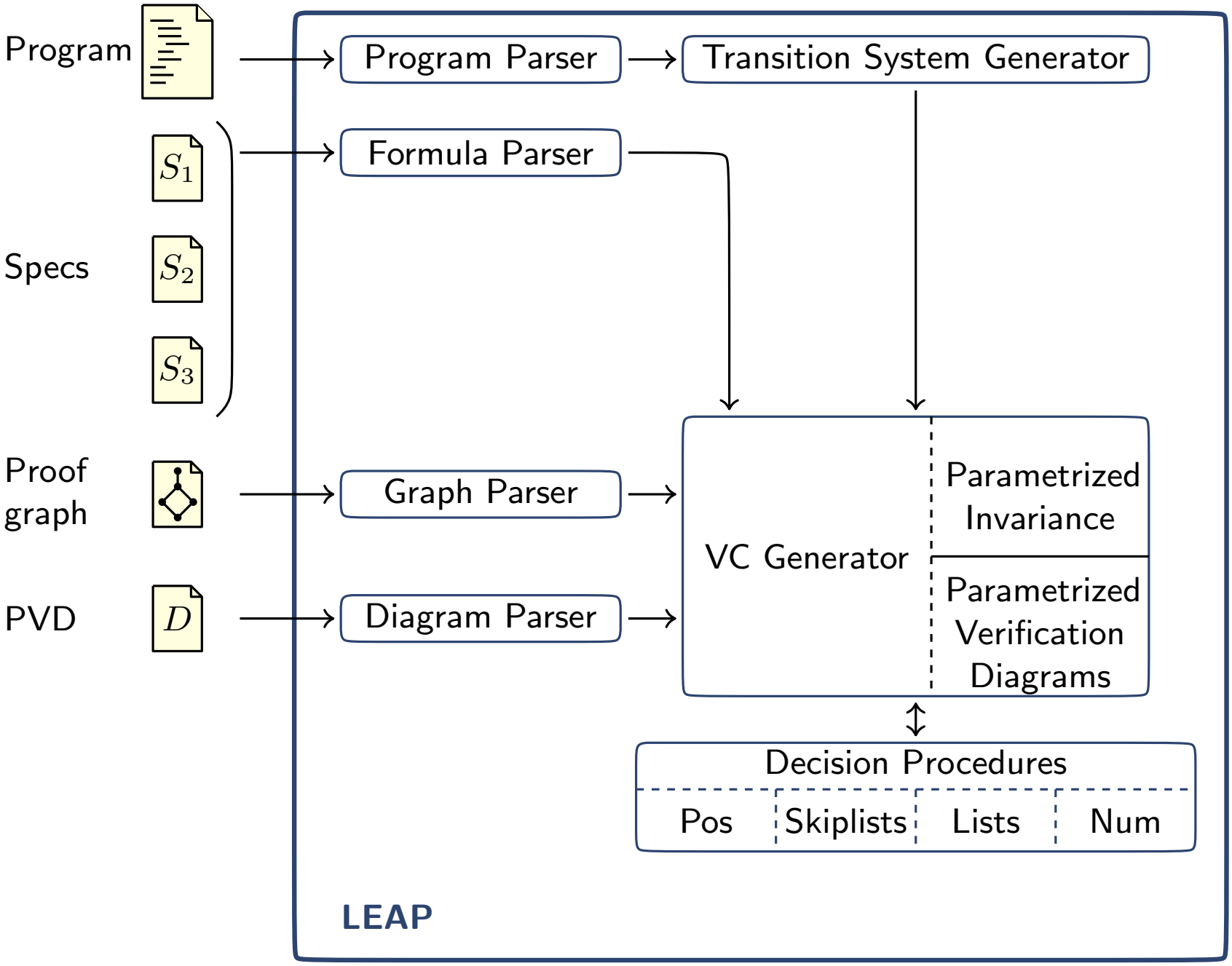
LEAP: Structure



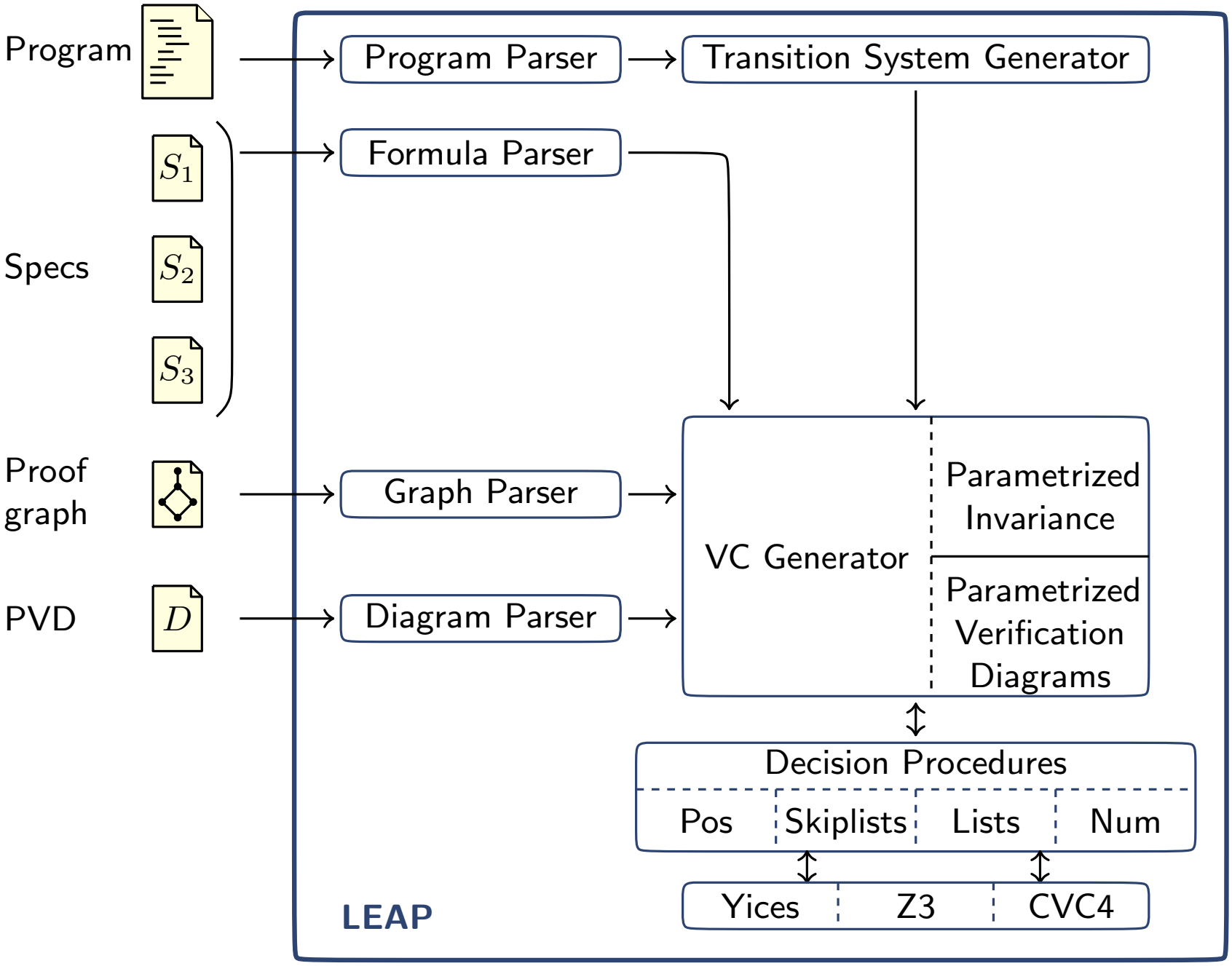
LEAP: Structure



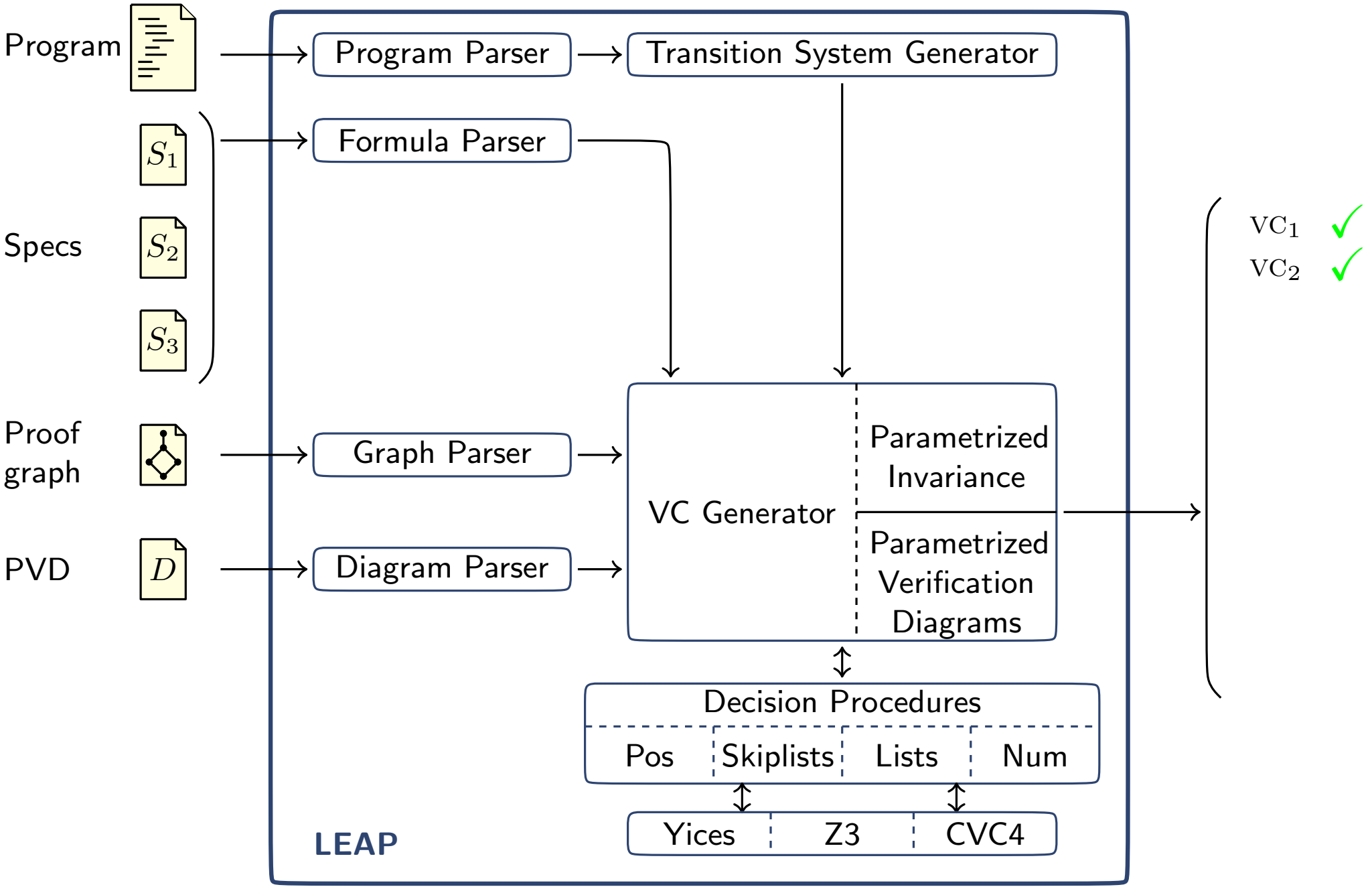
LEAP: Structure



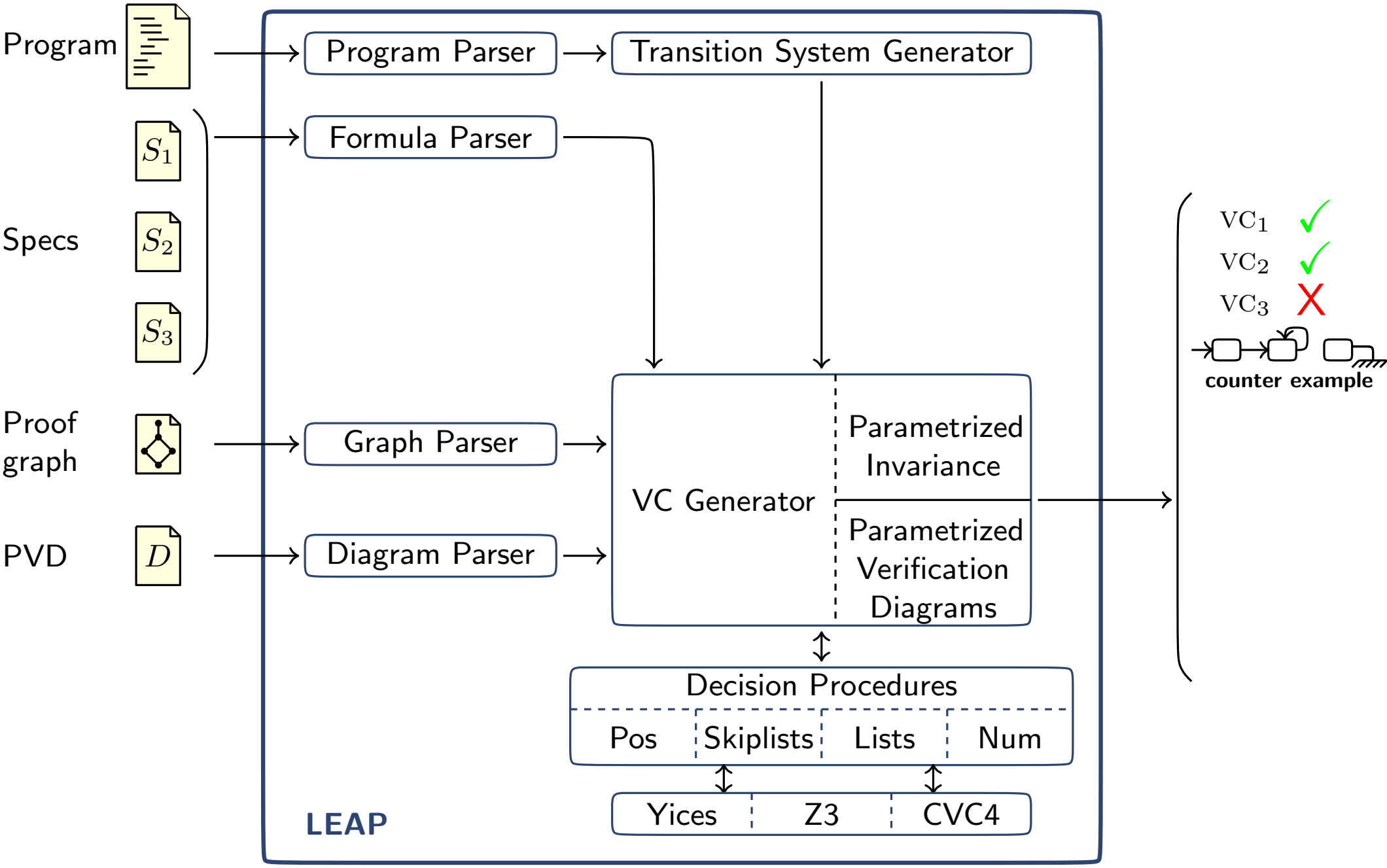
LEAP: Structure



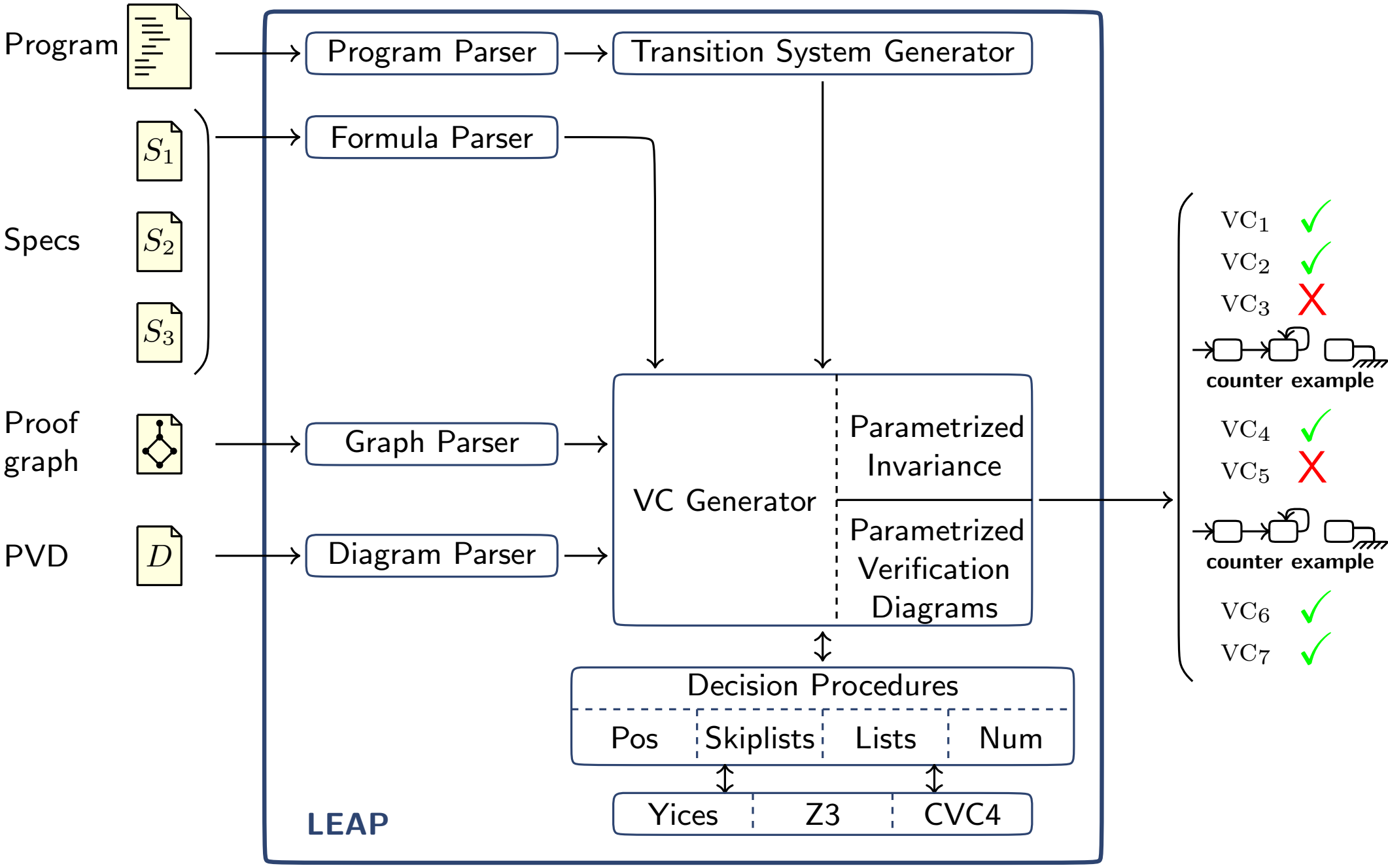
LEAP: Structure



LEAP: Structure



LEAP: Structure



Verification using LEAP

We use **LEAP** to verify temporal, structural and functional properties

- Mutual exclusion protocols
- Concurrent lock-coupling lists
- Concurrent lock-based queues
- Lock-free stacks
- Lock-free queues
- Skiplists with 2,3,4,5,.. levels
- Skiplists with unbounded levels (including KDE implementation)

LEAP and examples **available online** at

`software.imdea.org/leap`

LEAP: Some Experimental Results (safety)

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLin	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchIns	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRem	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Some Experimental Results (safety)

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLin	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchIns	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRem	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Some Experimental Results (safety)

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLin	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchIns	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRem	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Some Experimental Results (safety)

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLin	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchIns	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRem	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Some Experimental Results (safety)

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLin	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchIns	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRem	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP: Some Experimental Results (safety)

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1	121	95	26	∞	22.58	18.17	0.18	0.23
6	disj	2	181	177	4	121.74	0.19	0.01	0.01	0.12
7	funSchLin	1	121	97	24	∞	6.29	3.04	0.05	0.08
8	funSchIns	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRem	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11			6.75
21	mutex	2	28	26	2	0.32	0		1	0.01
22	minticket	1	19	18	1	0.04	0		1	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

LEAP analysis time remains insignificant

LEAP: Some Experimental Results (safety)

		formula		#solved vc		Brute	Heurist.	DP time(s.)		LEAP
		idx	#vc	pos	dp	time(s.)	time(s.)	slowest	average	time(s.)
1	list	0	61	38	23	∞	18.67	11.90	0.30	0.20
2	order	1	121	62	59	998.35	1.12	0.03	0.01	0.47
3	lock	1	121	76	45	778.15	0.47	0.02	0.01	0.18
4	next	1	121	60	61	∞	2.11	0.61	0.01	0.59
5	region	1						18.17	0.18	0.23
6	disj	2						0.01	0.01	0.12
7	funSchLin	1						3.04	0.05	0.08
8	funSchIns	1	121	93	28	∞	4.15	1.91	0.03	0.08
9	funSchRem	1	121	93	28	∞	5.40	2.60	0.04	0.10
10	funSearch	1	208	198	10	∞	3.54	1.57	0.01	0.34
11	funInsert	1	208	200	8	∞	0.50	0.01	0.01	0.22
12	funRemove	1	208	200	8	∞	1.41	0.95	0.01	0.24
13	skiplist_3	0	154	92	62	∞	1221.97	776.45	15.27	0.45
14	region_3	0	124	97	27	∞	27.50	17.36	0.34	0.58
15	next_3	0	84	65	19	∞	0.67	0.09	0.01	0.20
16	order_3	0	84	59	25	∞	9.66	7.80	0.10	1.31
17	skiplist	0	560	532	28	∞	19.79	5.40	0.24	0.15
18	region	0	1583	1527	56	∞	44.28	22.66	0.54	1.35
19	next	0	1899	1869	30	∞	3.19	0.32	0.02	1.59
20	order	0	2531	2474	57	∞	11.19	2.35	0.84	6.75
21	mutex	2	28	26	2	0.32	0.01	0.01	0.01	0.01
22	minticket	1	19	18	1	0.04	0.01	0.01	0.01	0.01
23	notsame	2	28	26	2	0.13	0.03	0.01	0.01	0.01
24	mutexS	2	28	26	2	0.44	0.04	0.01	0.01	0.01
25	minticketS	1	19	18	1	0.31	0.01	0.01	0.01	0.01
26	notsameS	2	28	26	2	0.14	0.02	0.01	0.01	0.01

Decision procedures perform well... but still room for improvements

LEAP: Some Experimental Results (liveness)

MinTicket (progress):

	#VC	#solved VC		single VC time(s.)		DP	LEAP
		pos	num	slowest	average	time(s)	time(s)
Initiation	1	0	1	0.01	0.01	0.01	0.01
Consecution	153	144	9	2.66	0.03	4.22	0.06
Acceptance	195	132	63	1.46	0.08	15.28	0.05
Fairness	24	20	4	0.03	0.01	0.10	0.02

Concurrent List (termination):

	#VC	#solved VC		single VC time(s.)		DP	LEAP
		pos	TLL	slowest	average	time(s)	time(s)
Initiation	1	0	1	0.01	0.01	0.01	0.01
Consecution	1550	1343	207	3.80	0.05	78.12	3.42
Acceptance	5404	4352	1052	191.61	0.12	647.04	1.61
Fairness	48	20	28	0.42	0.16	7.82	0.14

Published Results

A Deductive Verification Techniques for Parametrized Systems

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

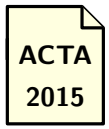
6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

Published Results

A Deductive Verification Techniques for Parametrized Systems



1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

Published Results

A Deductive Verification Techniques for Parametrized Systems

ACTA 2015

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

TIME 2014

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

Published Results

A Deductive Verification Techniques for Parametrized Systems

ACTA 2015

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

TIME 2014

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

SAS 2012

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

Published Results

A Deductive Verification Techniques for Parametrized Systems

ACTA 2015

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

TIME 2014

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

SAS 2012

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

ICFEM 2010

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

Published Results

A Deductive Verification Techniques for Parametrized Systems

ACTA 2015

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

TIME 2014

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

SAS 2012

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

ICFEM 2010

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

NFM 2011

6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

Published Results

A Deductive Verification Techniques for Parametrized Systems

ACTA 2015

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

TIME 2014

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

SAS 2012

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

ICFEM 2010

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

NFM 2011

6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

ATVA 2014

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

Published Results

A Deductive Verification Techniques for Parametrized Systems

ACTA 2015

1 Parametrized Invariance
Deductive proof rules for concurrent parametrized invariants

TIME 2014

2 Parametrized Verification Diagrams
Diagram based verification for concurrent parametrized liveness properties

SAS 2012

3 Invariant Generation using Abstract Interpretation
Automatic parametrized invariant generation using off-the-shelf sequential absint

B Decision Procedures for Complex Data Structures

ICFEM 2010

5 TL3: Decidable Theory for Concurrent Lists
A Theory and decision procedure for concurrent data structures of the shape of a list

NFM 2011

6 TSL_K: Decidable Theories for Concurrent Bounded Skiplists
Theories and decision procedures for concurrent skiplists of at most K levels

ATVA 2014

7 TSL: A Decidable Theory for Skiplists with Arbitrary Levels
Theory and decision procedure for skiplists with unbounded many levels

C Implementation and Evaluation of our Framework

CAV 2014

Conclusions

- A novel **deductive framework** for parametrized verification
- Designed for **concurrent data structures**
- Suitable for **safety** and **liveness** temporal properties
- Constructed **specialized theories and decision procedures**
- We studied **automatic generation** of parametrized invariants
- **Implemented a tool** with our verification framework
- We **verified various programs** and concurrent data structures

Future Work and Open Questions

- **Relax symmetry** and study **process topologies**
- Better **invariant generation**: for more data domains
- Beyond interleaving semantics:
  Weak memory models and **distributed algorithms**
- More **decision procedures** for more datatypes
- Implement a generic **Nelson-Oppen** for faster decision procedures
- Use **theorem provers** in combination with SMT solvers, and generate full formal proofs
- Add support for **real world programming languages**

Future Work

- **Relax symmetry** and study **process topologies**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

Future Work

- **Relax symmetry** and study **process topologies**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

Future Work

- **Relax symmetry** and study **process topologies**

To show that $\mathcal{S}$ satisfies $\varphi(i)$:

$$\frac{\begin{array}{lrcll} (\ \mathsf{I}\ ) & \Theta & \rightarrow & \varphi & \\ (\mathsf{SC}) & \varphi \wedge \tau^{(i)} & \rightarrow & \varphi' & \text{forall } \tau \\ (\mathsf{OC}) & \varphi \wedge k \neq i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}}{\Box\varphi}$$

$$\begin{array}{lrcll} (\mathsf{OC}_<) & \varphi \wedge k < i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \\ (\mathsf{OC}_>) & \varphi \wedge k > i \wedge \tau^{(k)} & \rightarrow & \varphi' & \text{forall } \tau, \text{fresh } k \end{array}$$

Thanks!