Invariant Generation for Parametrized Systems using Self-Reflection

Alejandro Sánchez[1] Sriram Sankaranarayanan[2]

César Sánchez[1,3] Bor-Yuh Evan Chang[2]

[1]IMDEA Software Institute, Spain

[2]University of Colorado, USA

[3]Spanish Council for Scientific Research (CSIC), Spain

Rich Model Toolkit COST Action Meeting, Malta, 17 June 2013

Motivation

Motivation

P_1 P_2 $\cdots$ P_N

- Consider a **finite** but **unbounded** set of processes

Motivation

$$P_1 \quad \| \quad P_2 \quad \| \quad \cdots \quad \| \quad P_N$$

- Consider a **finite** but **unbounded** set of processes
- Running the **same program** in **paralallel**

Motivation

$$P_1 \quad \| \quad P_2 \quad \| \quad \cdots \quad \| \quad P_N$$

- Consider a **finite** but **unbounded** set of processes
- Running the **same program** in **paralallel**
- Classical scenarios include:
  - Device drivers
  - Distributed algorithms
  - Concurrent datastructures
  - Robotic swarms
  - Biological Systems

Motivation

GOAL : Automatic Infer Parametrized Invariants

- Consider a **finite** but **unbounded** set of processes
- Running the **same program** in **paralallel**
- Classical scenarios include:
  - Device drivers
  - Distributed algorithms
  - Concurrent datastructures
  - Robotic swarms
  - Biological Systems

Our Contribution

Our Contribution

- An **automatic abstract-interpretation-based framework**
  - Able to reason over **parametrized programs**
  - Capable of infering **indexed invariants**

Our Contribution

- An **automatic abstract-interpretation-based framework**
  - Able to reason over **parametrized programs**
  - Capable of infering **indexed invariants**

... and even more important ...

- **Adapting** existing invariant **synthesis techniques**
- **Leveraging** off-the-shelf **sequential invariant generators**

Our Contribution

- An **automatic abstract-interpretation-based framework**
  - Able to reason over **parametrized programs**
  - Capable of infering **indexed invariants**

... and even more important ...

- **Adapting** existing invariant **synthesis techniques**
- **Leveraging** off-the-shelf **sequential invariant generators**

In this talk

- Introduce the notion of **Reflective Abstraction**
- Report first **empirical** evaluation

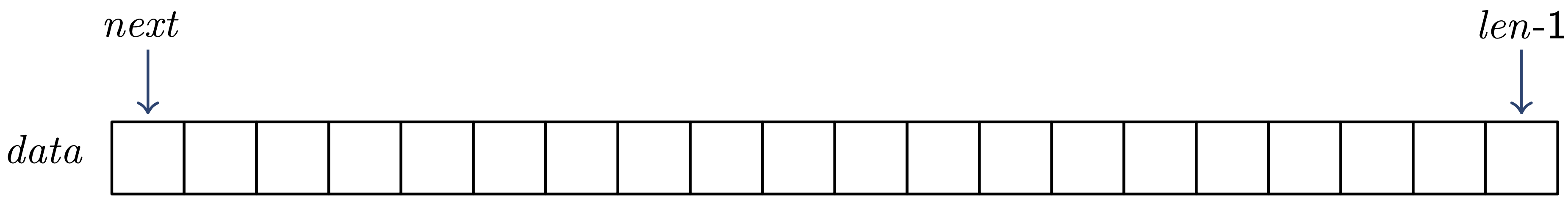
Motivating Example: WORKSTEALING

Motivating Example: WORKSTEALING

```
global   Int len > 0;   Array[len] data;   Int next = 0;

procedure WORKSTEALING
local   Int c = 0;   Int end = 0;
if next + 4 ≤ len{c := next;   end := next + 5;   next := next + 5; }
while c < end{
    data[c] := process data[c];
    c := c + 1;
}
```

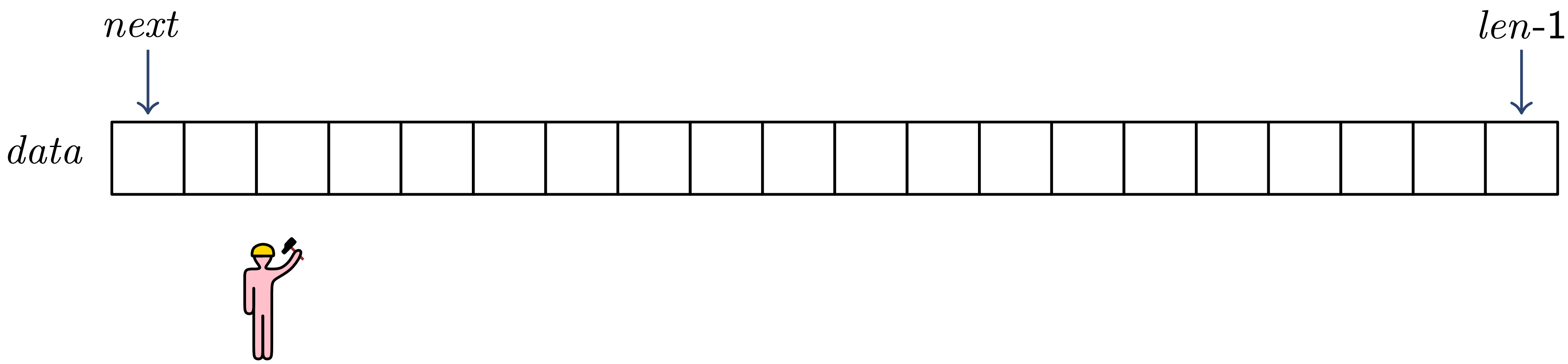
Motivating Example: WORKSTEALING

```
global   Int len > 0;   Array[len] data;   Int next = 0;

procedure WORKSTEALING
local   Int c = 0;   Int end = 0;
if next + 4 ≤ len{c := next;   end := next + 5;   next := next + 5;}
while c < end{
    data[c] := process data[c];
    c := c + 1;
}
```



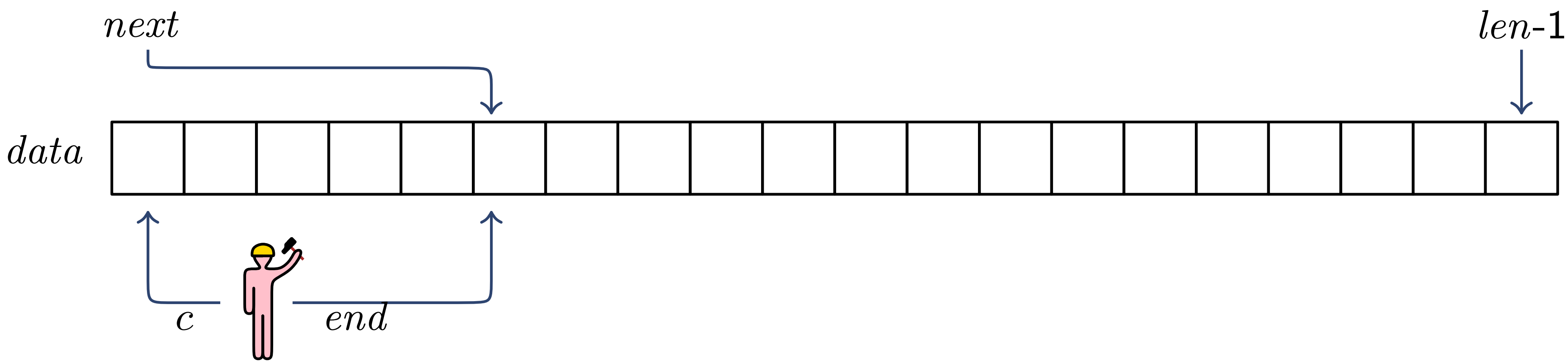
Motivating Example: WORKSTEALING

```
global   Int len > 0;   Array[len] data;   Int next = 0;

procedure WORKSTEALING
local   Int c = 0;   Int end = 0;
if next + 4 ≤ len{c := next;   end := next + 5;   next := next + 5; }
while c < end{
    data[c] := process data[c];
    c := c + 1;
}
```



Motivating Example: Workstealing

```
global    Int len > 0;   Array[len] data;   Int next = 0;

procedure Workstealing
local   Int c = 0;   Int end = 0;
if next + 4 ≤ len {c := next;   end := next + 5;   next := next + 5; }
while c < end {
    data[c] := process data[c];
    c := c + 1;
}
```



Motivating Example: Workstealing

global $\quad Int\ len > 0; \quad Array[len]\ data; \quad Int\ next = 0;$

procedure Workstealing

local $\quad Int\ c = 0; \quad Int\ end = 0;$

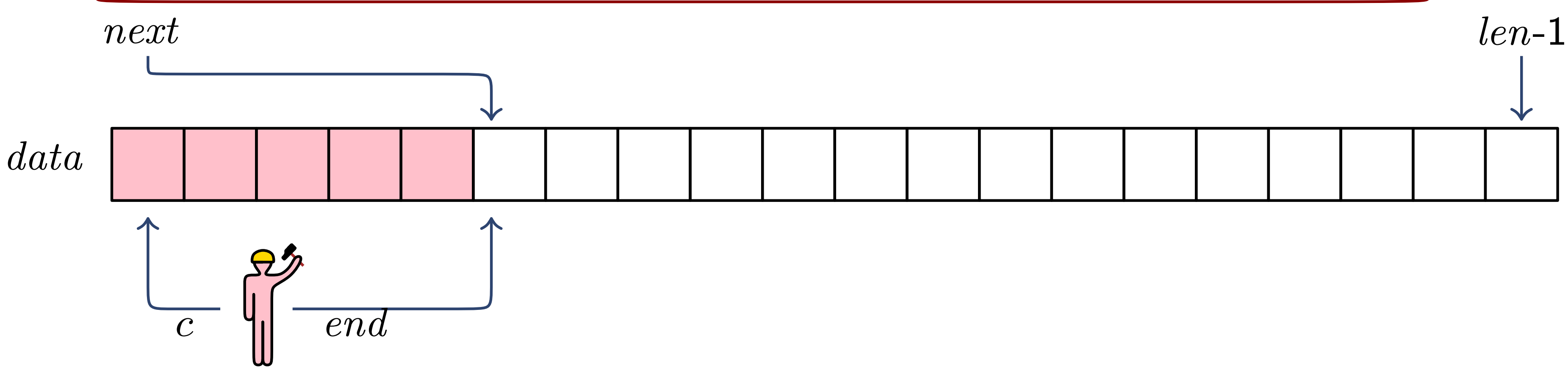
if $next + 4 \leq len \{ c := next; \quad end := next + 5; \quad next := next + 5; \}$

while $c < end \{$

$\quad data[c] := process\, data[c];$

$\quad c := c + 1;$

$\}$



Motivating Example: Workstealing

global $\quad Int\ len > 0; \quad Array[len]\ data; \quad Int\ next = 0;$

procedure Workstealing

local $\quad Int\ c = 0; \quad Int\ end = 0;$

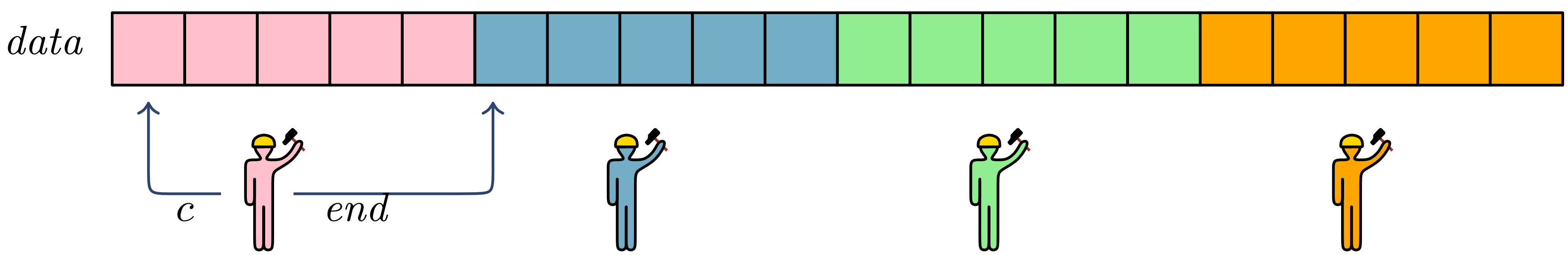
if $next + 4 \leq len \{ c := next; \quad end := next + 5; \quad next := next + 5; \}$

while $c < end \{$

$\quad data[c] := process\, data[c];$

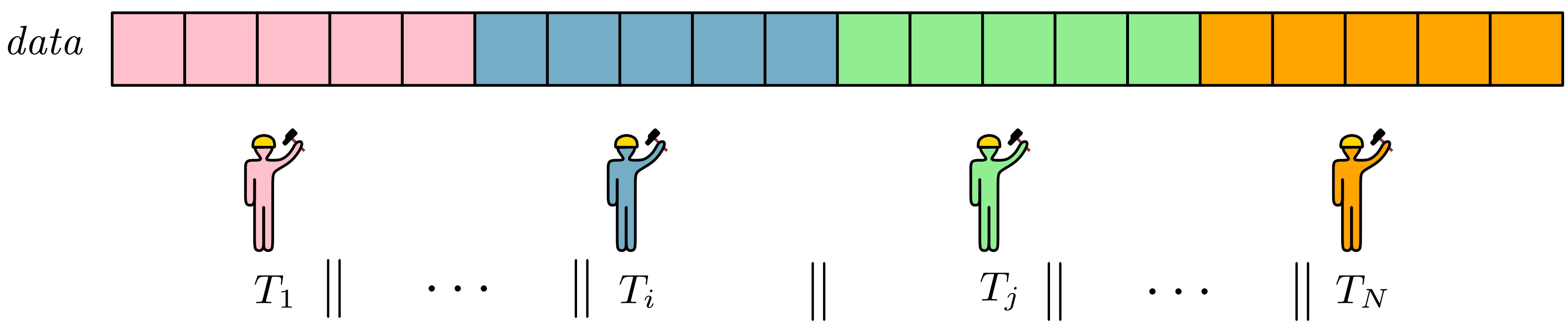
$\quad c := c + 1;$

$\}$



Motivating Example: WORKSTEALING

```
global   Int len > 0;   Array[len] data;   Int next = 0;

procedure WORKSTEALING
local   Int c = 0;   Int end = 0;
if next + 4 ≤ len{c := next;   end := next + 5;   next := next + 5; }
while c < end{
    data[c] := process data[c];
    c := c + 1;
}
```



Motivating Example: WORKSTEALING

global $Int\ len > 0; \quad Array[len]\ data; \quad Int\ next = 0;$

procedure WORKSTEALING

local $Int\ c = 0; \quad Int\ end = 0;$

if $next + 4 \leq len \{ c := next; \quad end := next + 5; \quad next := next + 5; \}$

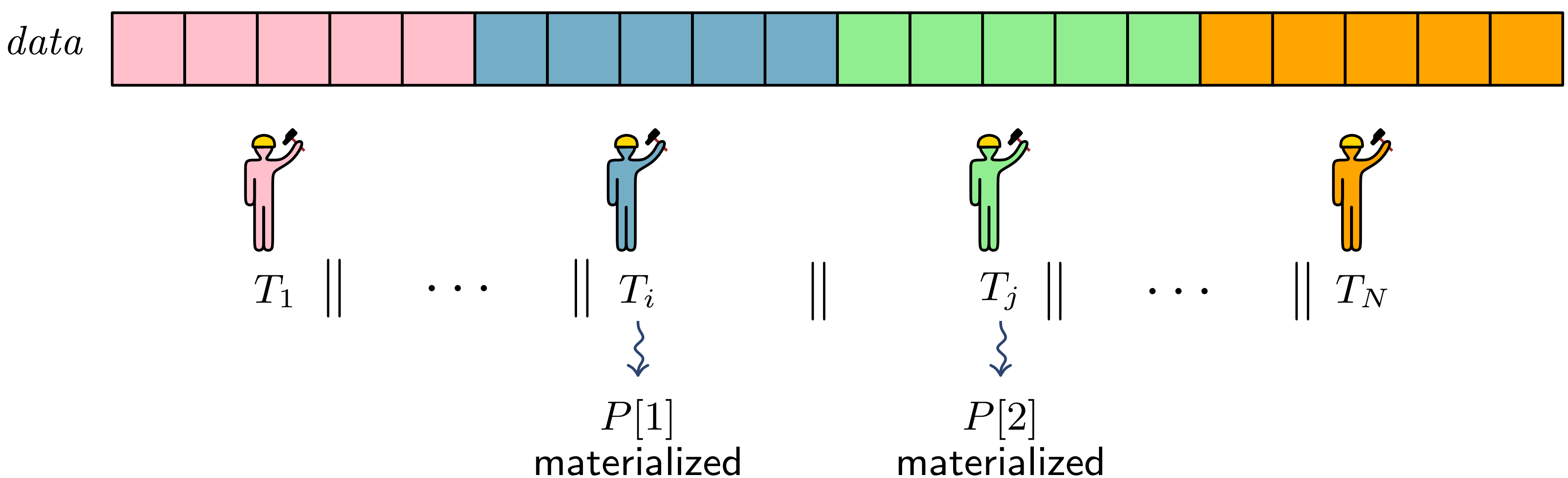
while $c < end \{$

$\quad data[c] := process\, data[c];$

$\quad c := c + 1;$

$\}$

As example, for a 2-indexed invariant

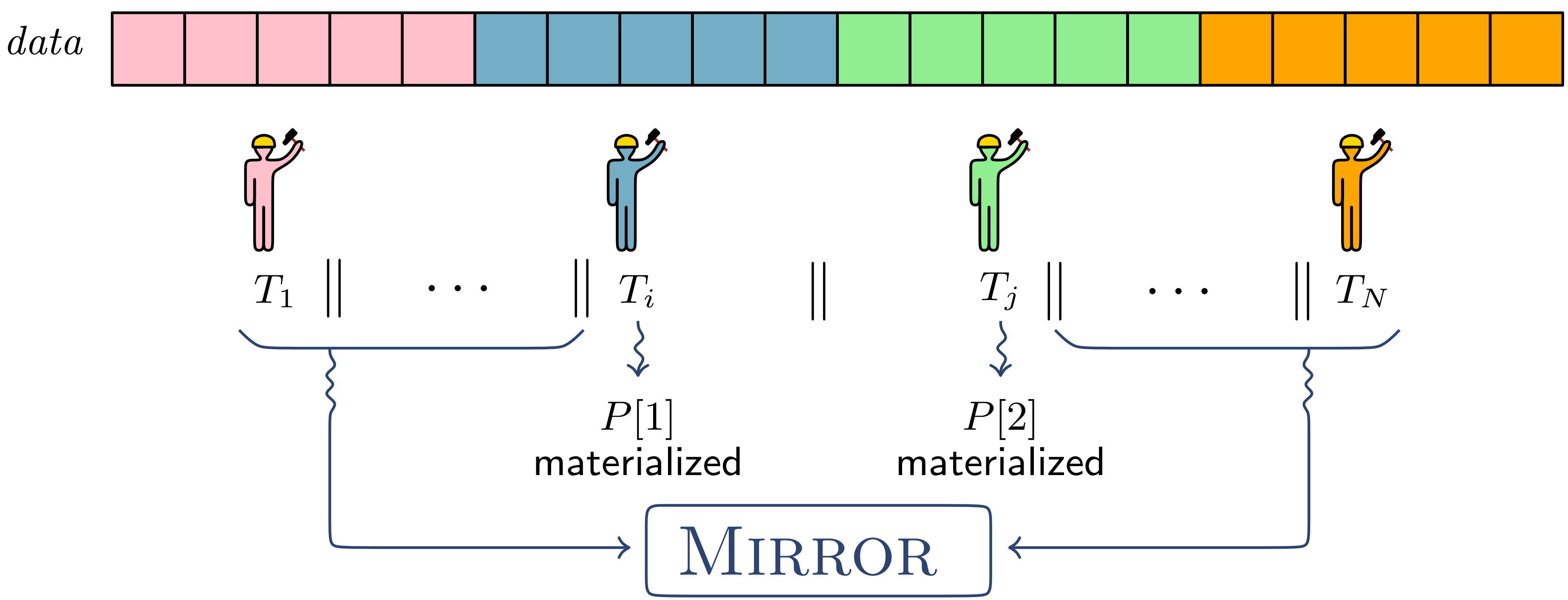


Motivating Example: Workstealing

global $Int\ len > 0;\quad Array[len]\ data;\quad Int\ next = 0;$

procedure Workstealing
local $Int\ c = 0;\quad Int\ end = 0;$
if $next + 4 \leq len\{c := next;\quad end := next + 5;\quad next := next + 5;\}$
while $c < end\{$
  $data[c] := process\,data[c];$
  $c := c + 1;$
$\}$

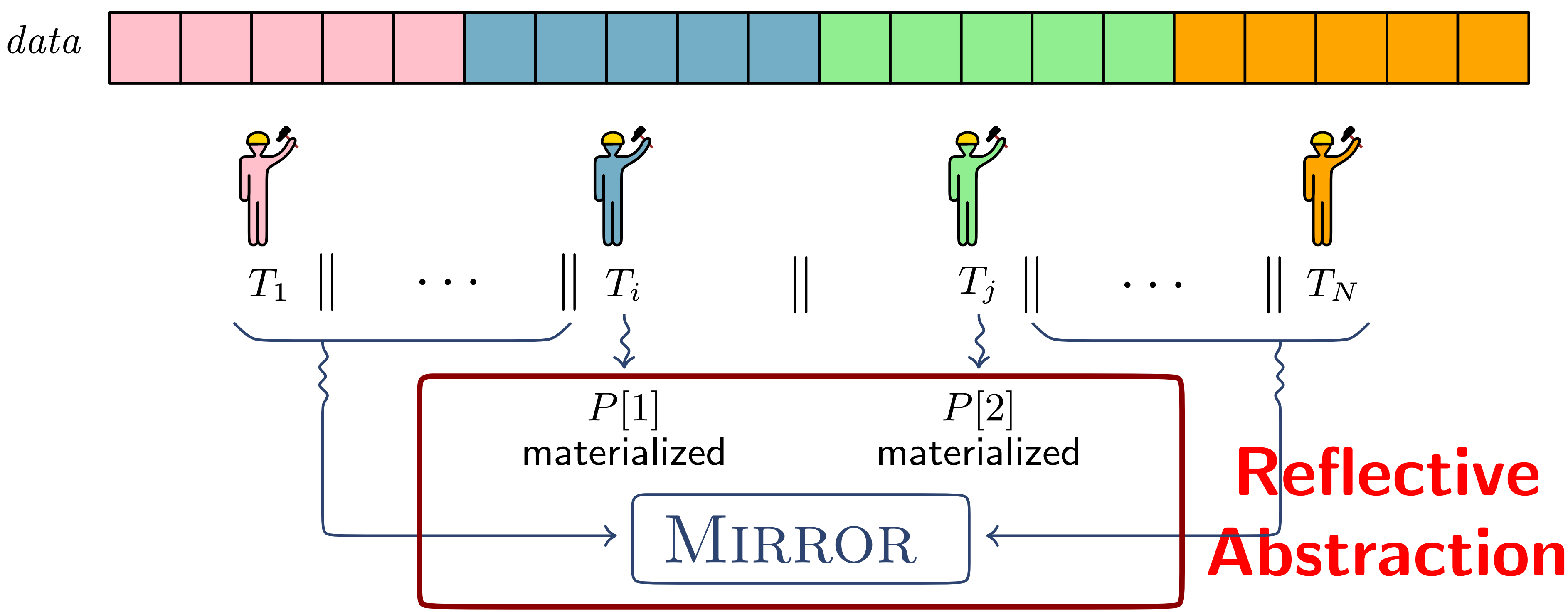
As example, for a 2-indexed invariant



Motivating Example: WORKSTEALING

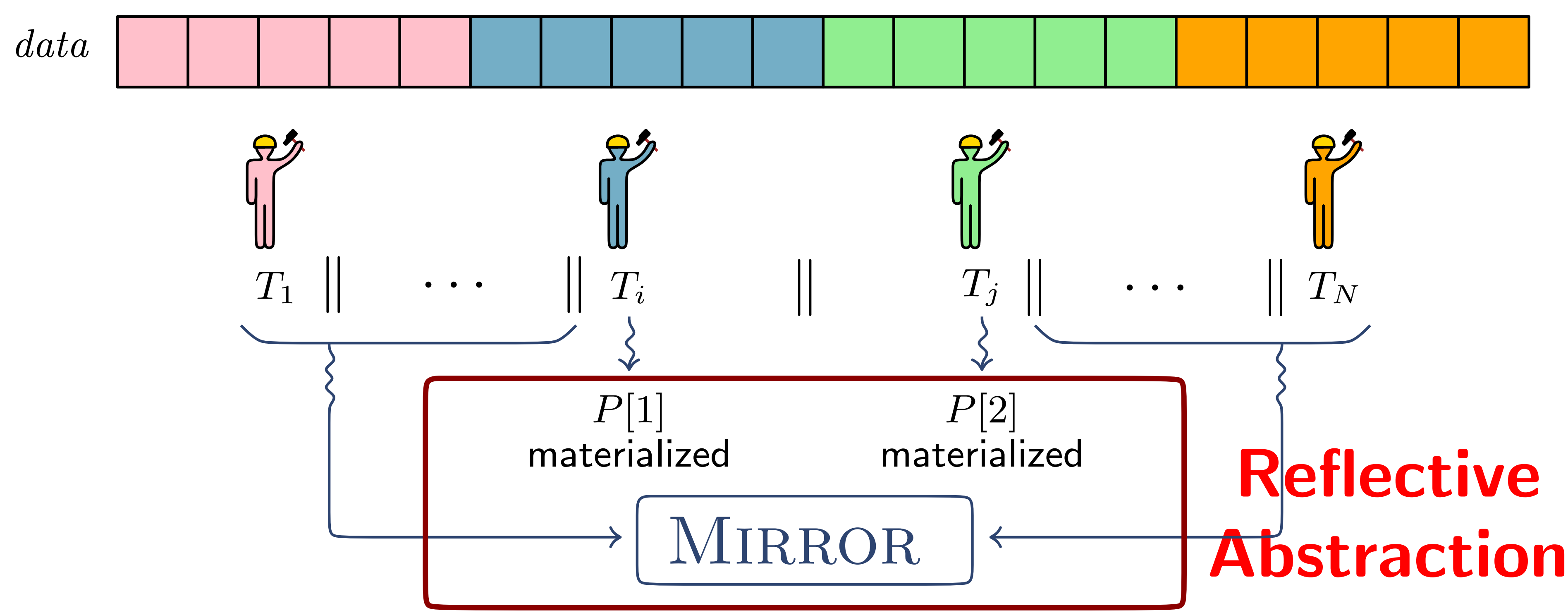
```
global   Int len > 0;   Array[len] data;   Int next = 0;

procedure WORKSTEALING
local   Int c = 0;   Int end = 0;
if next + 4 ≤ len{c := next;   end := next + 5;   next := next + 5; }
while c < end{
    data[c] := process data[c];
    c := c + 1;
}
```

As example, for a 2-indexed invariant



Motivating Example: WORKSTEALING

Invariant examples			# Mat. Threads
Ψ_0	:	$next$ mod $5 = 0$	0
Ψ_1	:	$(\forall\, i) \bullet 0 \leq c[i] < len$	1
Ψ_2	:	$(\forall\, i_1, i_2) \bullet c[i_1] \neq c[i_2]$	2



Reflective Invariants in a Nutshell

Reflective Invariants in a Nutshell

Materialized

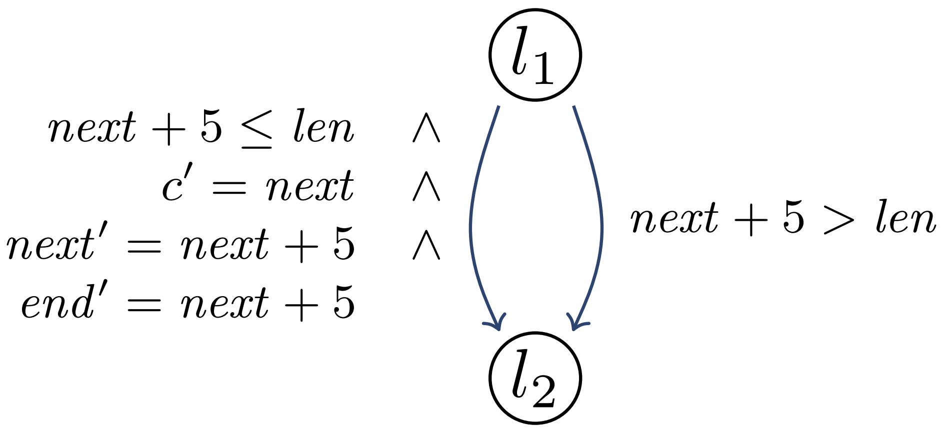
Reflective Invariants in a Nutshell

Materialized

$$\begin{aligned} next + 5 \leq len \quad &\wedge \\ c' = next \quad &\wedge \\ next' = next + 5 \quad &\wedge \\ end' = next + 5 \quad & \end{aligned}$$

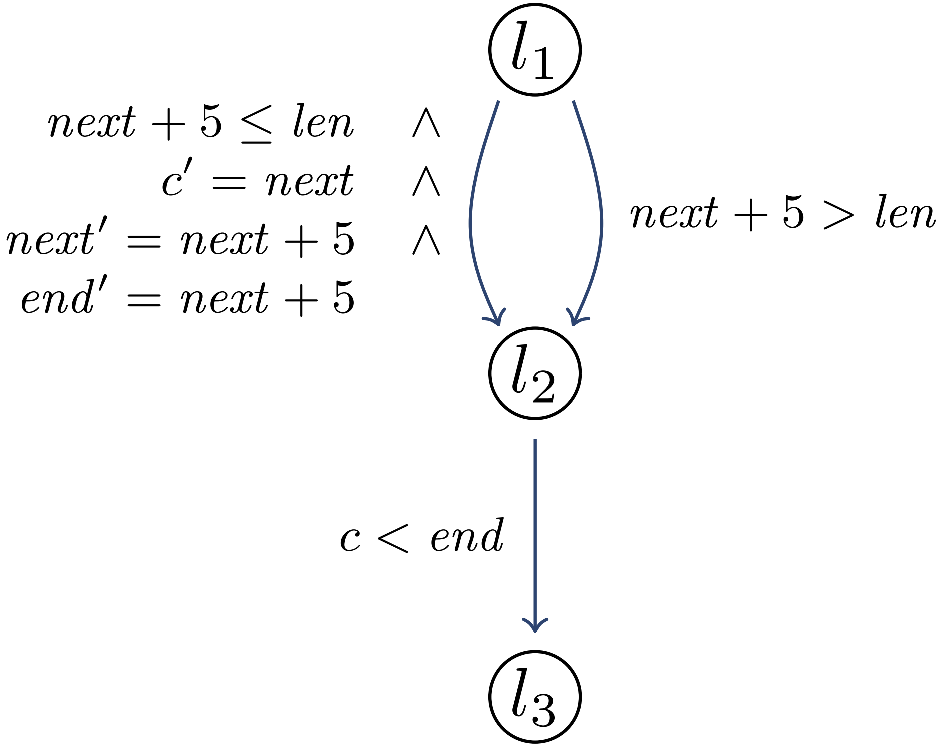
Reflective Invariants in a Nutshell

Materialized



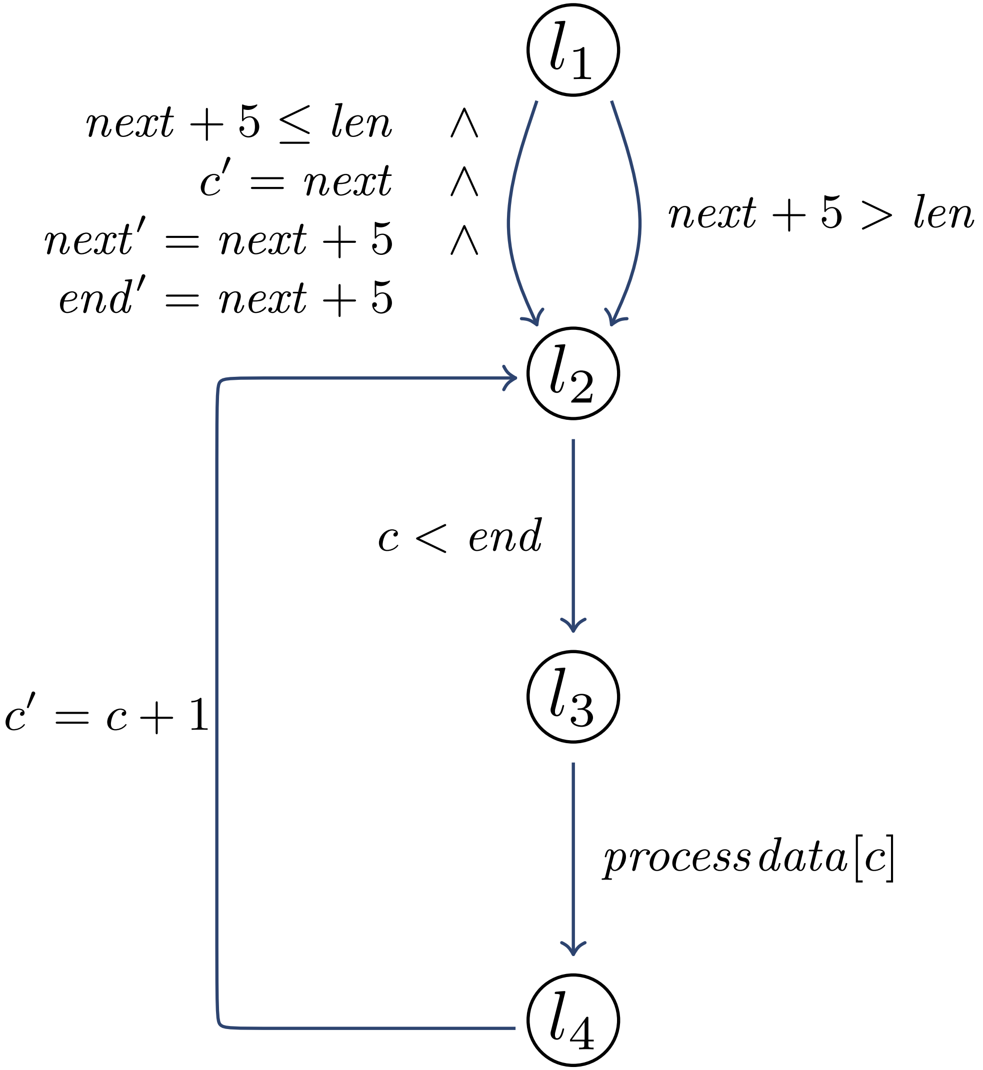
Reflective Invariants in a Nutshell

Materialized



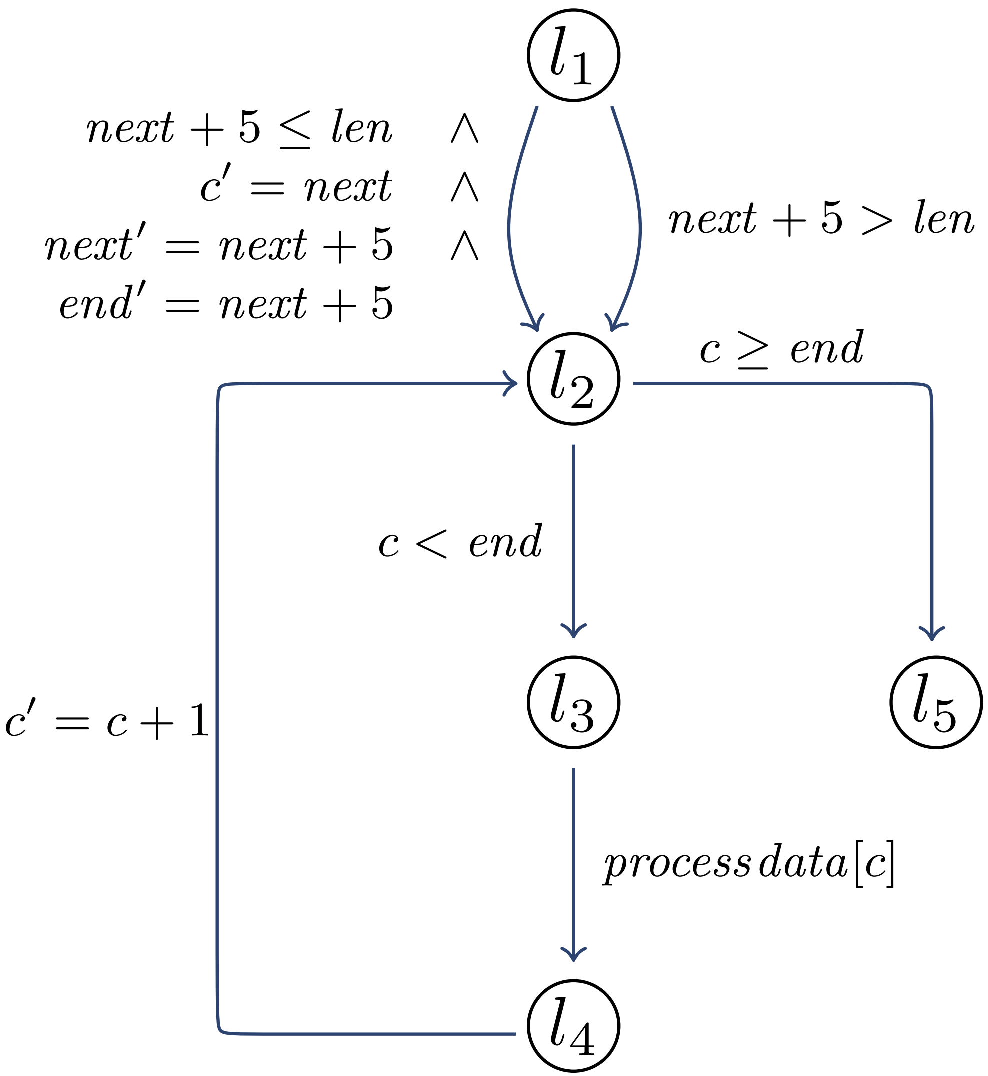
Reflective Invariants in a Nutshell

Materialized



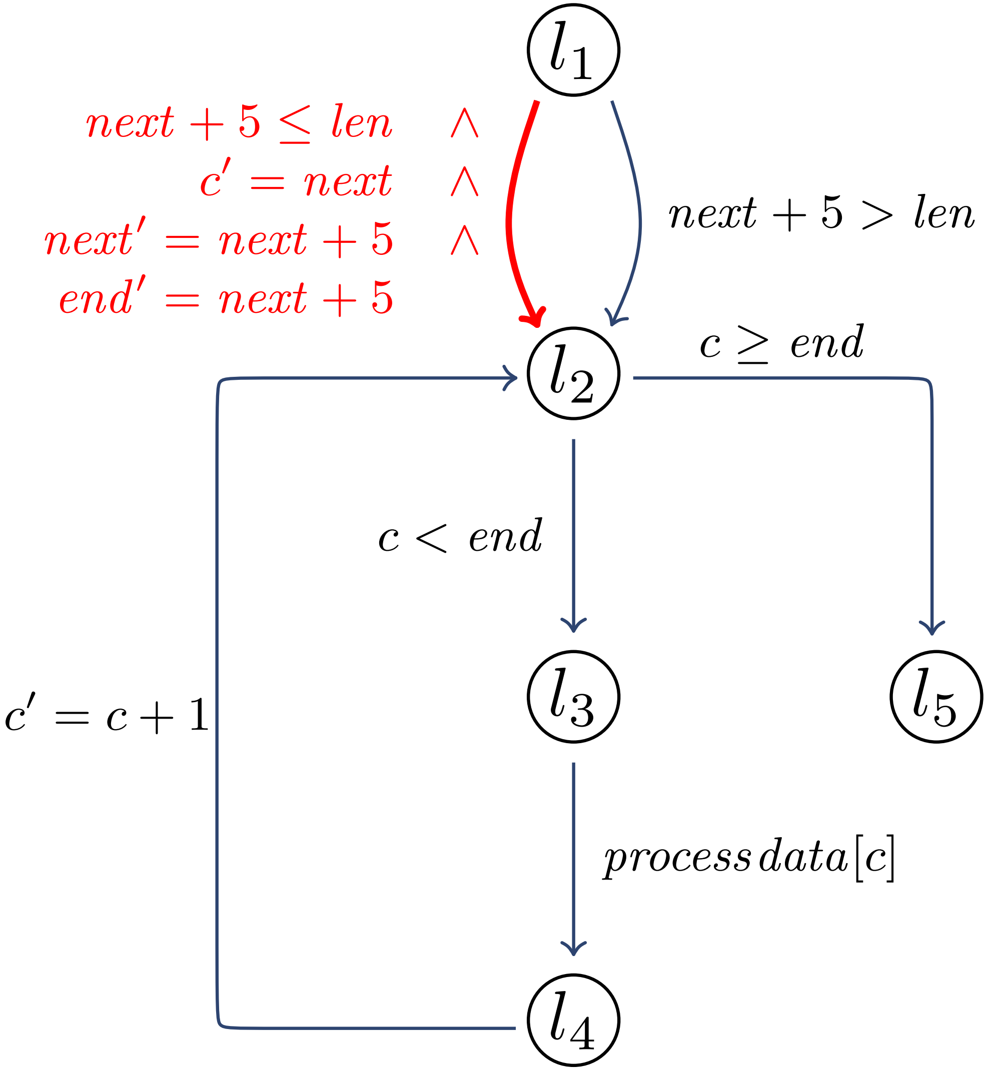
Reflective Invariants in a Nutshell

Materialized



Reflective Invariants in a Nutshell

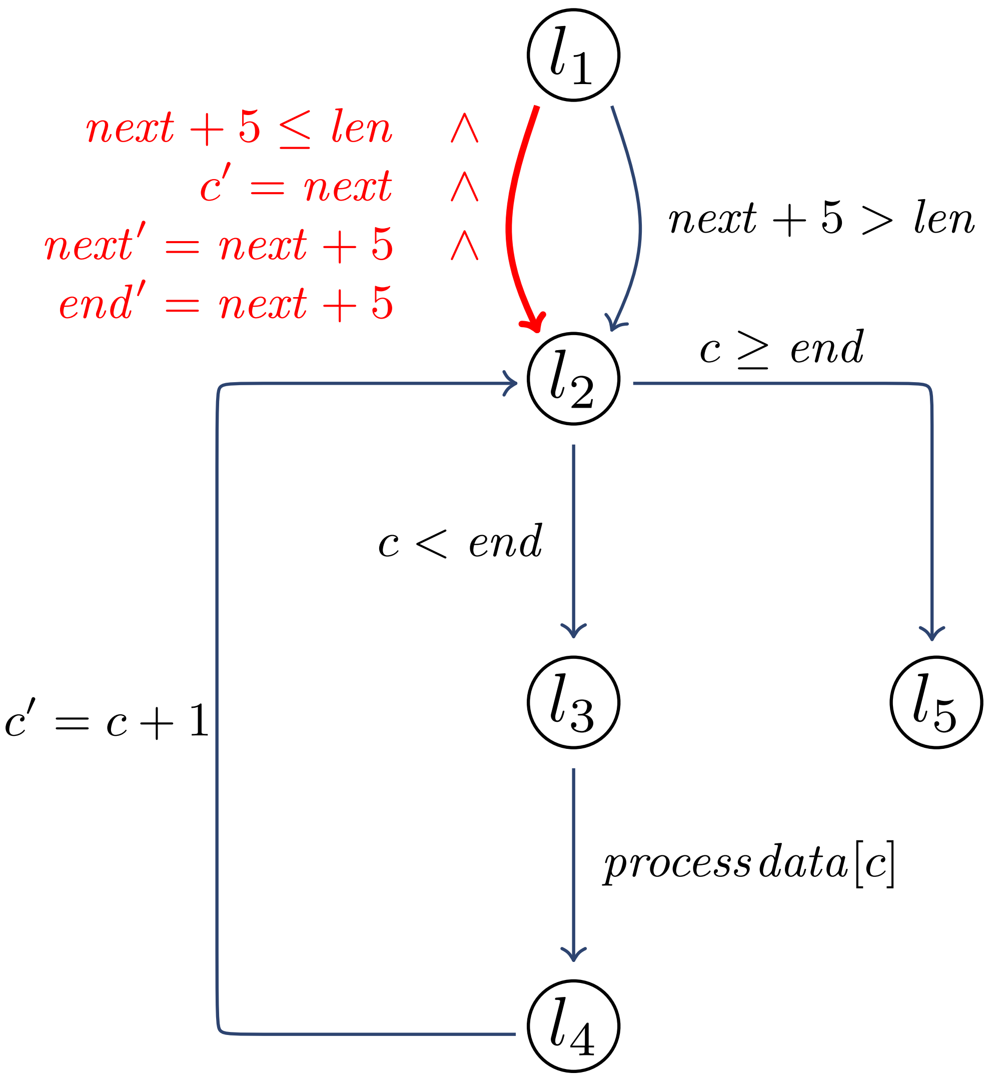
Materialized



Transition modifying shared variable $next$

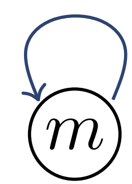
Reflective Invariants in a Nutshell

Materialized



Mirror

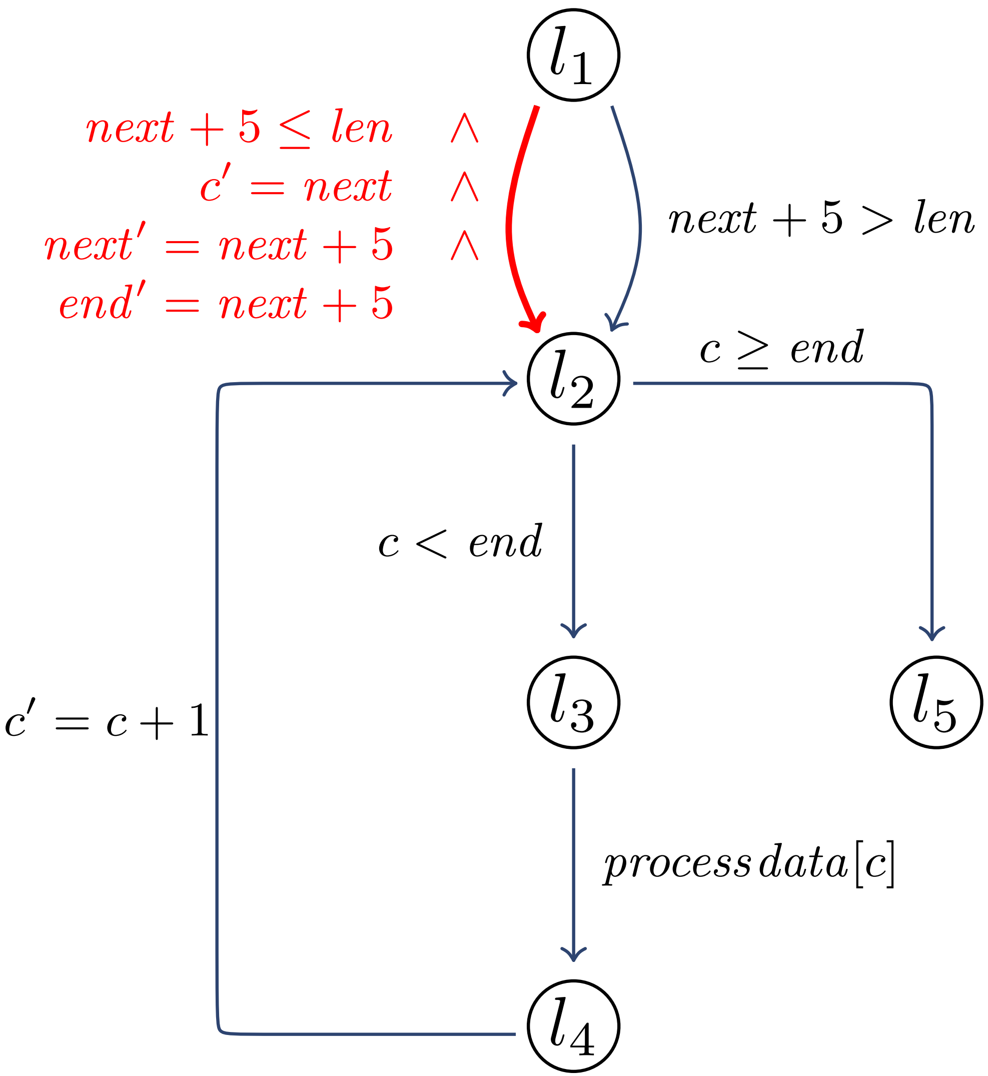
$$\left(\begin{array}{cc} Inv@l_1 & \wedge \\ next + 10 \leq len & \wedge \\ c' = next & \wedge \\ next' = next + 10 & \wedge \\ end' = next + 10 & \end{array} \right)$$



Transition modifying shared variable $next$

Reflective Invariants in a Nutshell

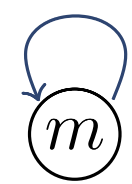
Materialized



Mirror

$$\left(\exists \begin{array}{c} c, end, \\ c', end' \end{array} \right)$$

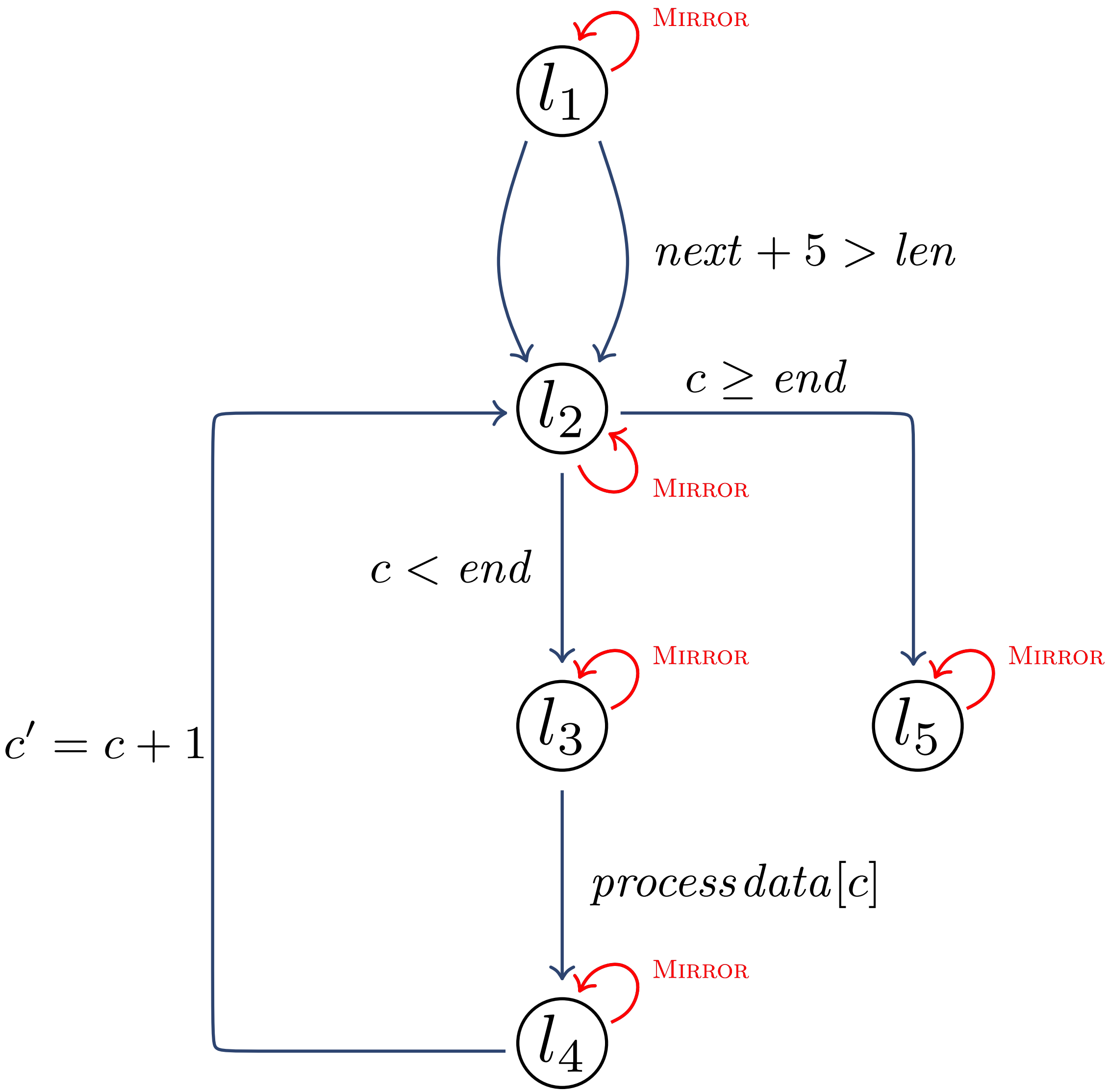
$$\left(\begin{array}{cc} Inv@l_1 & \wedge \\ next + 10 \leq len & \wedge \\ c' = next & \wedge \\ next' = next + 10 & \wedge \\ end' = next + 10 & \end{array} \right)$$



Transition modifying shared variable $next$

Reflective Invariants in a Nutshell

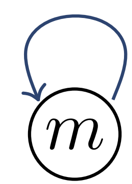
Materialized



Mirror

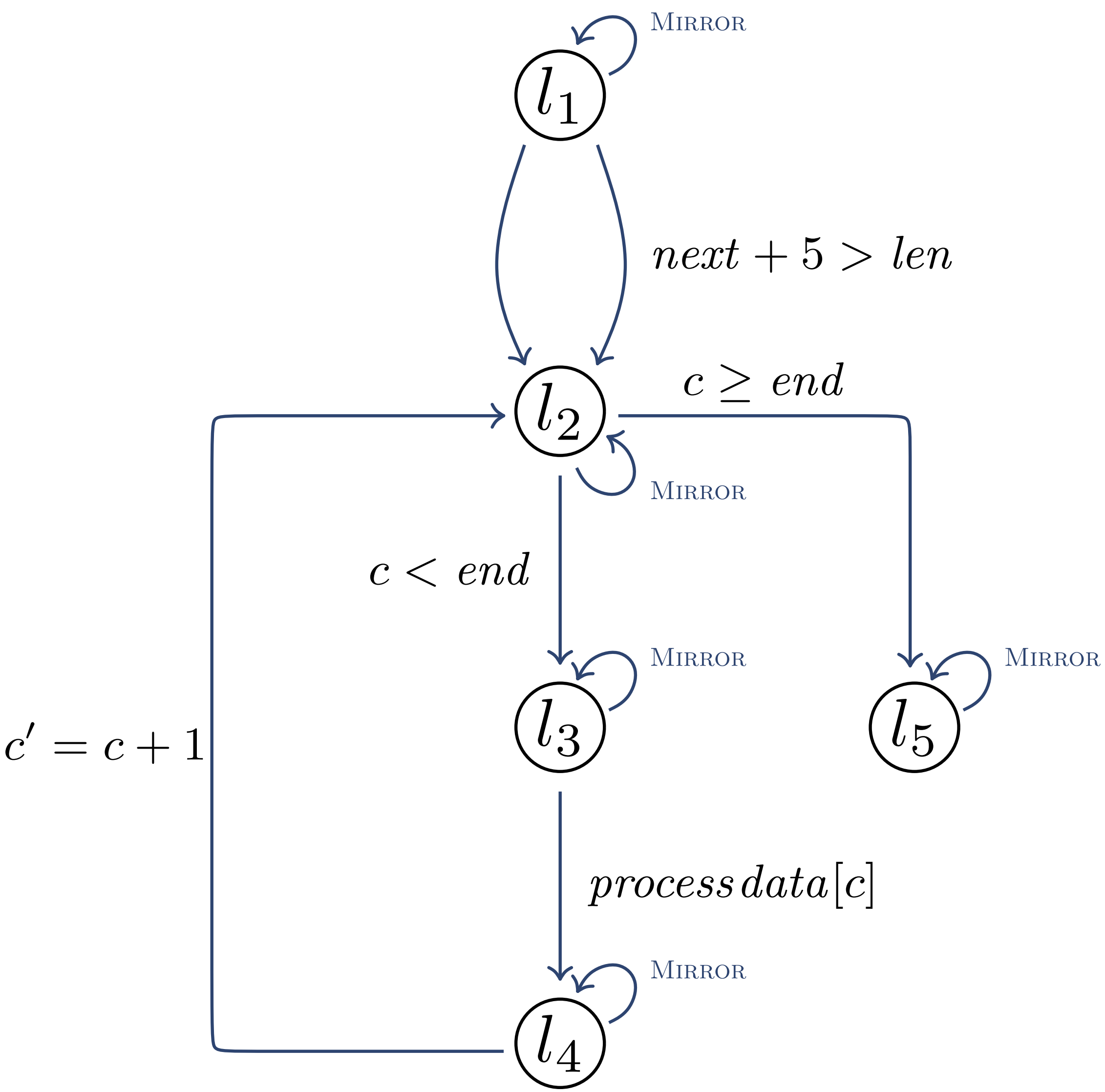
$$\left(\exists \begin{array}{c} c, end, \\ c', end' \end{array} \right)$$

$$\left(\begin{array}{cc} \mathit{Inv}@l_1 & \wedge \\ next + 10 \leq len & \wedge \\ c' = next & \wedge \\ next' = next + 10 & \wedge \\ end' = next + 10 & \end{array} \right)$$



Reflective Invariants in a Nutshell

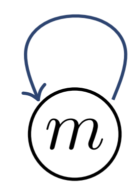
Materialized



Mirror

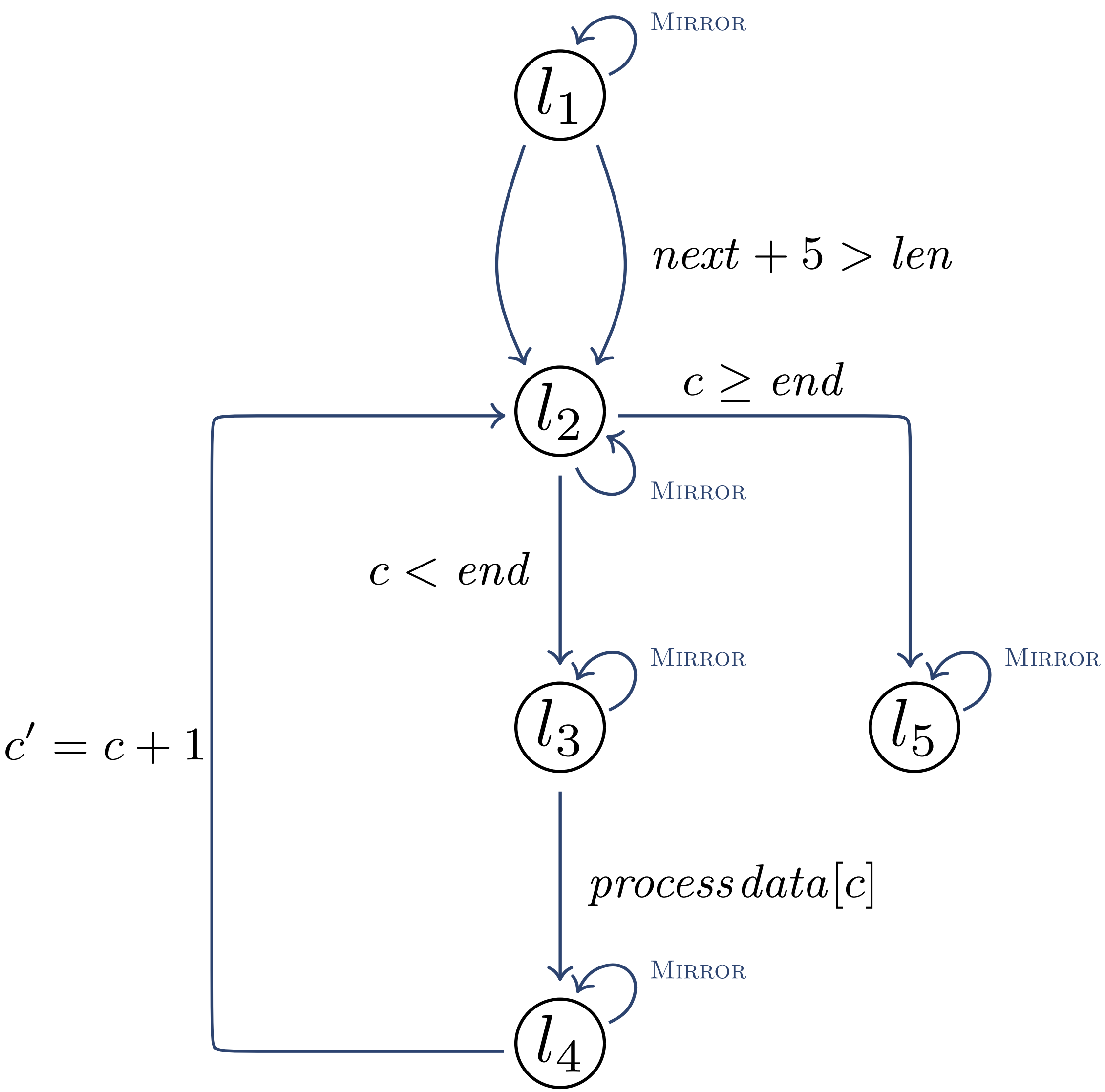
$$\left(\exists \begin{array}{c} c, end, \\ c', end' \end{array} \right)$$

$$\left(\begin{array}{cc} \textsf{Inv}@l_1 & \wedge \\ next + 10 \leq len & \wedge \\ c' = next & \wedge \\ next' = next + 10 & \wedge \\ end' = next + 10 & \end{array} \right)$$



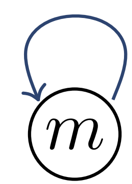
Reflective Invariants in a Nutshell

Materialized



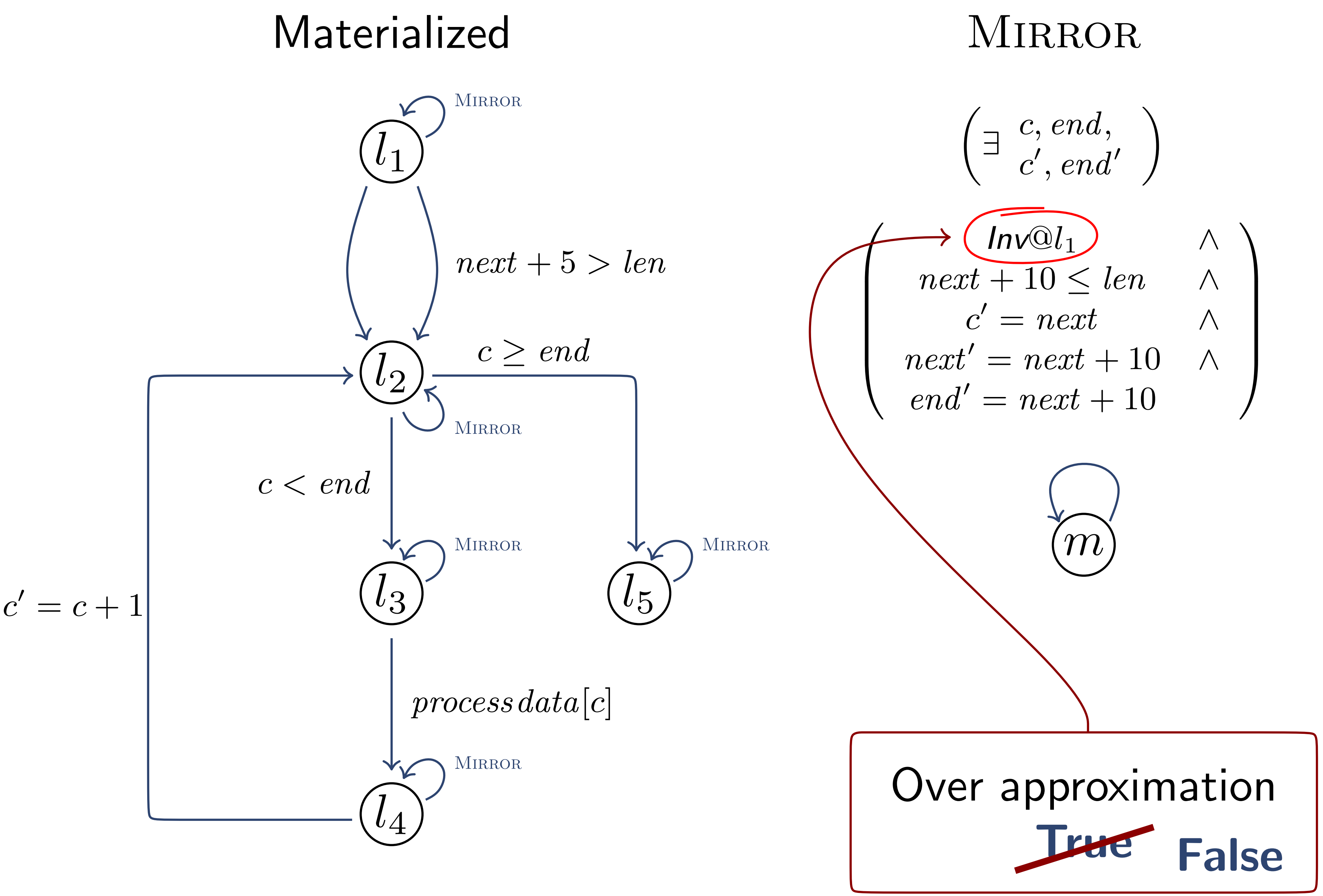
MIRROR

$$\left(\exists \begin{array}{c} c, end, \\ c', end' \end{array}\right) \left(\begin{array}{cc} Inv@l_1 & \wedge \\ next + 10 \leq len & \wedge \\ c' = next & \wedge \\ next' = next + 10 & \wedge \\ end' = next + 10 & \end{array}\right)$$



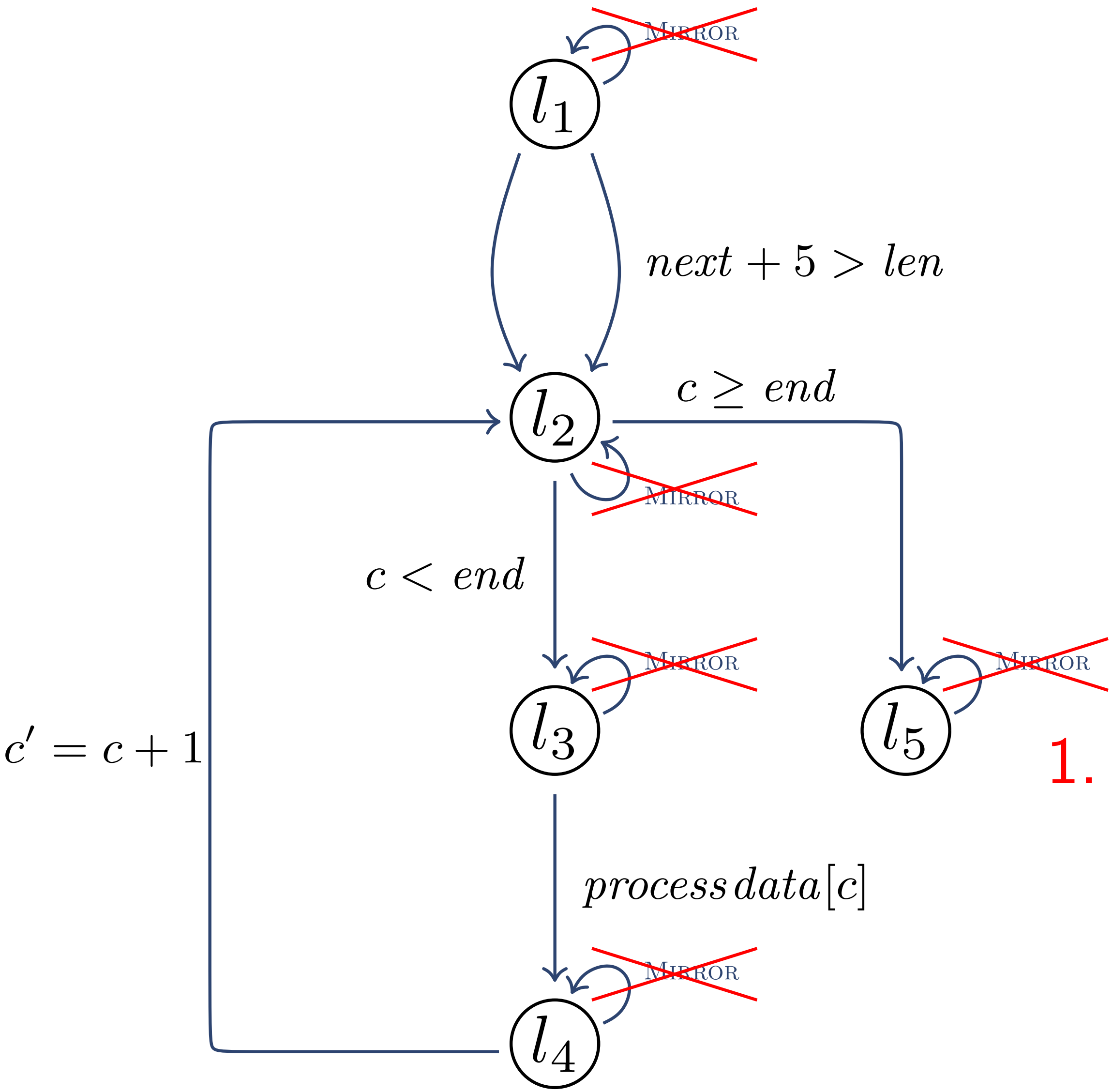
Over approximation
True

Reflective Invariants in a Nutshell



Reflective Invariants in a Nutshell

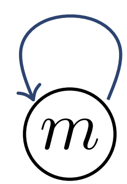
Materialized



Mirror

$$\left(\exists \begin{array}{c} c, end, \\ c', end' \end{array}\right)$$

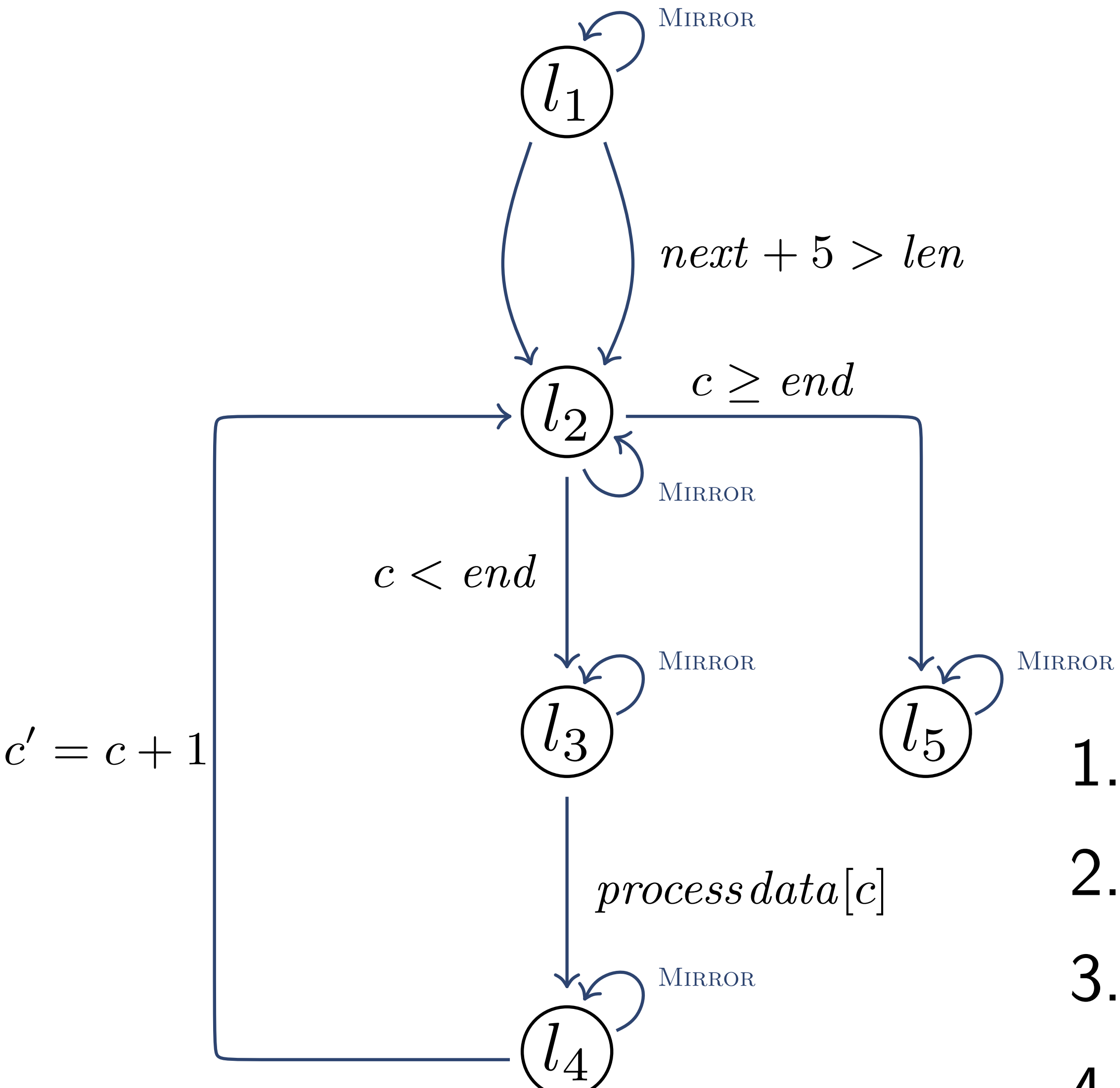
$$\left(\begin{array}{cc} Inv@l_1 & \wedge \\ next + 10 \leq len & \wedge \\ c' = next & \wedge \\ next' = next + 10 & \wedge \\ end' = next + 10 & \end{array}\right)$$



1. Begin, $\forall l \in Loc$, $Inv_1(l) = false$

Reflective Invariants in a Nutshell

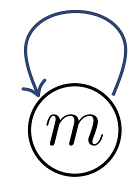
Materialized



MIRROR

$$\left(\exists \begin{array}{l} c, end, \\ c', end' \end{array}\right)$$

$$\left(\begin{array}{cc} Inv@l_1 & \wedge \\ next + 10 \leq len & \wedge \\ c' = next & \wedge \\ next' = next + 10 & \wedge \\ end' = next + 10 & \end{array}\right)$$



1. Begin, $\forall l \in Loc$, $Inv_1(l) = false$
2. Compute Σ_j using Inv_j
3. Obtain $Inv_{j+1} = AbsInt(\Sigma_j)$
4. Stop when $\forall l \in Loc$, $Inv_{j+1}(l) \sqsubseteq Inv_j(l)$

Reflective Abstraction and Inductive Invariants

Reflective Abstraction and Inductive Invariants

- We define as usual:
  - Parametrized Transition System: $\Pi = \langle G, X, \mathit{Trs}, l_0, \Theta \rangle$
  - Transition: $\tau = \langle l_{src}, l_{dst}, \rho \rangle$

- A **reflective abstraction** over approximates all parametrized instances of Π by a finite transition system

Reflective Abstraction and Inductive Invariants

- We define as usual:
  - Parametrized Transition System: $\Pi = \langle G, X, \mathit{Trs}, l_0, \Theta \rangle$
  - Transition: $\tau = \langle l_{\mathit{src}}, l_{\mathit{dst}}, \rho \rangle$
- A **reflective abstraction** over approximates all parametrized instances of Π by a finite transition system

---

Let $\Gamma[X]$ be some fixed first-order language of assertions

- An **assertion map** $\eta : \mathit{Loc} \rightharpoonup_{\mathit{fin}} \Gamma[X]$ is **inductive** when:
  1. $\Theta \to \eta(l_0)$
  2. for any $\tau : \langle l_{\mathit{src}}, l_{\mathit{dst}}, \rho \rangle$, $\tau(\eta(l_{\mathit{src}})) \to \eta(l_{\mathit{dst}})$

Reflective Abstraction and Inductive Invariants

- We define as usual:
  - Parametrized Transition System: $\Pi = \langle G, X, \mathit{Trs}, l_0, \Theta \rangle$
  - Transition: $\tau = \langle l_{\mathit{src}}, l_{\mathit{dst}}, \rho \rangle$
- A **reflective abstraction** over approximates all parametrized instances of Π by a finite transition system

---

Let $\Gamma[X]$ be some fixed first-order language of assertions

- An **assertion map** $\eta : \mathit{Loc} \rightharpoonup_{\mathit{fin}} \Gamma[X]$ is **inductive** when:

  1. $\Theta \to \eta(l_0)$
  2. for any $\tau : \langle l_{\mathit{src}}, l_{\mathit{dst}}, \rho \rangle$, $\tau(\eta(l_{\mathit{src}})) \to \eta(l_{\mathit{dst}})$

In formal verification, to show Ψ,
we seek an inductive assertion map η s.t.
$\forall l \in \mathit{Loc}, \quad \eta(l) \models \Psi$

Reflective Abstraction and Inductive Invariants

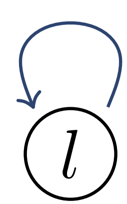
Given $\Pi = \langle G, X, \mathit{Loc}, \mathit{Trs}, l_0, \Theta \rangle$ and assertion map η, we define:

$$\textsc{Reflect}_{\Pi}(\eta)$$

as a sequential transition system with:

$$\begin{array}{rcl} \text{variables} & : & G \cup X \\ \text{locations} & : & \mathit{Loc} \\ \text{transitions} & : & \mathit{Trs} \cup \left\{ \textsc{Mirror}(\tau, \eta, l) \mid \tau \in \mathit{Trs} \text{ and } l \in \mathit{Loc} \right\} \end{array}$$

$$\mathit{pres}(X) \wedge (\exists Y, Y') \big(\eta(l_{\mathsf{src}})[G, Y] \wedge \rho[G, Y, G', Y'] \big)$$



Theorem: Reflection Soundness

Let η be an inductive assertion map for $\textsc{Reflect}_{\Pi}(\eta)$, then for each location l of Π,

$\eta(l)$ **is a 1-index invariant**

Theorem: Reflection Soundness

Let η be an inductive assertion map for $\text{Reflect}_{\Pi}(\eta)$, then for each location l of Π,

$\eta(l)$ is a 1-index invariant

- What about k-index invariants?

$$\text{Start from } \underbrace{\Pi \times \cdots \times \Pi}_{k}$$

To obtain an assertion map $\quad \eta_k : \mathit{Loc} \times \cdots \times \mathit{Loc} \rightharpoonup_{\mathit{fin}} \Gamma[X]$

Reflective Abstract Interpretation

- We **iteratively** generate invariants for **parametrized systems**
- As fix point of a **monotone operator** over an **abstract domain**
- Applying **abstract interpretation** on reflective abstractions

$$\widehat{\eta}^* = \mathsf{lfp}\, \widehat{\mathscr{F}}(\bot, \Sigma)$$

Reflective Abstract Interpretation

- We consider **3 schemas**:
  - **Lazy**:
  $$\widehat{\eta}^{*}_{\textsc{Lazy}} = \mathsf{lfp}\, \widehat{\mathscr{G}}_{L,\Pi}(\bot)$$
  $$\widehat{\mathscr{G}}_{L,\Pi}(\widehat{\eta}) \stackrel{\mathsf{def}}{=} \mathsf{lfp}\, \widehat{\mathscr{F}}\,(\bot, \textsc{Reflect}_{\Pi}(\gamma \circ \widehat{\eta}))$$
  - **Eager**:
  $$\widehat{\eta}^{*}_{\textsc{Eager}} = \mathsf{lfp}\, \widehat{\mathscr{G}}_{E,\Pi}(\bot)$$
  $$\widehat{\mathscr{G}}_{E,\Pi}(\widehat{\eta}) \stackrel{\mathsf{def}}{=} \widehat{\mathscr{F}}\,(\widehat{\eta}, \textsc{Reflect}_{\Pi}(\gamma \circ \widehat{\eta}))$$
  - **Eager+**:
  $$\widehat{\eta}^{*}_{\textsc{Eager+}} = \mathsf{lfp}\, \widehat{\mathscr{G}}_{E+,\Pi}(\bot)$$
  $$\widehat{\mathscr{G}}_{E,\Pi}(\widehat{\eta}) \stackrel{\mathsf{def}}{=} \mathsf{lfp}\, \widehat{\mathscr{F}}(\bot, \textsc{Reflect}_{\Pi}(\gamma \circ \widehat{\eta}^{*}_{\textsc{Eager}}))$$

Reflective Abstraction vs. Interference Abstraction

Reflective Abstraction vs. Interference Abstraction

- Consider the following **example**:

global $Int\ g \geq 0;$

Thread P $\{$
local $Int\ x = 0;$
1: $\langle$**await**$(g > 0);\ x := g;\ g := 0;\rangle$
2: $x := x + 1$
3: $g := x$
4: $\}$

Reflective Abstraction vs. Interference Abstraction

- Consider the following **example**:

global $\quad Int\ g \geq 0;$

Thread P $\{$
local $\quad Int\ x = 0;$
1: $\quad \langle$**await**$(g > 0);\ x := g;\ g := 0; \rangle$
2: $\quad x := x + 1$
3: $\quad g := x$
4: $\}$

- For the MIRROR, considering transition 3, we have:

$$(\exists x) \quad \eta(l_3) \wedge g' = x$$

Reflective Abstraction vs. Interference Abstraction

- Consider the following **example**:

```
        global  Int g ≥ 0;

        Thread P {
          local  Int x = 0;
1:           ⟨await(g > 0); x := g; g := 0;⟩
2:           x := x + 1
3:           g := x
4:      }
```

- For the MIRROR, considering transition 3, we have:

$$(\exists x) \quad \underbrace{\eta(l_3)} \land g' = x$$

Interference Abstraction

Reflective Abstraction

Reflective Abstraction vs. Interference Abstraction

- Consider the following **example**:

$$
\begin{array}{ll}
 & \textbf{global} \quad Int\ g \geq 0; \\
 & \\
 & \textbf{Thread P } \{ \\
 & \quad \textbf{local} \quad Int\ x = 0; \\
1: & \qquad \langle \textbf{await}(g > 0);\ x := g;\ g := 0; \rangle \\
2: & \qquad x := x + 1 \\
3: & \qquad g := x \\
4: & \}
\end{array}
$$

- For the MIRROR, considering transition 3, we have:

$$(\exists x) \quad \underbrace{\eta(l_3)} \wedge g' = x$$

Interference Abstraction

over approximates from **True**

fails to derive $g \geq 0$
(a non- deterministic update to g)

Reflective Abstraction

Reflective Abstraction vs. Interference Abstraction

- Consider the following **example**:

```
        global    Int g ≥ 0;

        Thread P {
          local   Int x = 0;
1:          ⟨await(g > 0); x := g; g := 0;⟩
2:          x := x + 1
3:          g := x
4:      }
```

- For the MIRROR, considering transition 3, we have:

$$(\exists x) \quad \underbrace{\eta(l_3)} \wedge g' = x$$

Interference Abstraction	**Reflective Abstraction**
over approximates from **True**	over approximates from **False**
⇝	⇝
fails to derive $g \geq 0$ (a non- deterministic update to g)	incrementally infers invs over x (derives $g \geq 0$)

Empirical Evaluation

- We study these **examples** (2-75 locs., 6-24 vars., 4-49 trans.):
  - Simple software barrier
  - Centralized software barrier
  - WORKSTEALING
  - Generalized dinning philosophers
  - Robot swarm moving on a $m \times n$ grid

Empirical Evaluation

- We study these **examples** (2-75 locs., 6-24 vars., 4-49 trans.):
  - Simple software barrier
  - Centralized software barrier
  - WORKSTEALING
  - Generalized dinning philosophers
  - Robot swarm moving on a $m \times n$ grid

- We **compare 4 schemes**:
  - Lazy
  - Eager
  - Eager+
  - Interference

Empirical Evaluation

- Results:

ID	Dom	Prps	Lazy			Eager			Eager+			Interf.		
			Time	Wid*	Prp	Time	Wid	Prp	Time	Wid	Prp	Time	Wid	Prp
Tbar	I	4	0.1	2	0	0.1	5	0	0.1	5	0	0.1	4	0
	P		0.2	4	4	0.1	5	4	0.1	5	4	0.1	4	4
	O		0.8	3	3	0.1	5	3	0.1	5	3	0.1	4	3
Wsteal	I	5	0.3	6	2	0.1	5	1	0.1	5	1	0.1	4	0
	P		2.4	6	1	0.1	7	1	0.2	7	3	0.1	7	5
	O		8.2	6	4	7.5	6	4	0.2	6	4	6.2	5	4
Cbar	I	9	0.9	3	4	0.1	7	0	0.1	8	0	0.1	7	0
	P		TO		0	1.7	11	4	2.7	12	5	1.1	10	6
	O		TO		0	7.5	9	6	11.3	9	6	6.2	8	4
Phil	I	14	1.9	4	2	0.1	8	2	0.1	8	2	0.1	7	0
	P		11.8	6	14	1.1	11	8	1.8	11	8	6.3	13	14
	O		TO		0	25	12	4	40	12	4	20	12	4
Rb(2,2)	I	16	31.3	8	4	0.4	10	4	0.4	11	4	0.2	10	0
	P		TO		0	9.3	22	3	15	23	3	5.8	15	4
	O		TO		0	142	25	3	225	26	3	105	18	3
Rb(2,3)	I	18	133	8	6	0.7	10	6	0.9	11	6	0.5	10	0
	P		TO		0	23	22	5	36.8	23	5	16	15	5
	O		TO		0	404	25	5	629	26	5	320	18	5
Rb(3,3)	I	23	1141	8	9	1.6	10	9	2.1	11	9	0.9	10	0
	P		TO		0	68.2	22	8	111.5	23	8	52	15	8
	O		TO		0	1414	25	8	2139	26	8	1168	18	8
Rb(4,4)	I	29	TO		0	6.7	11	16	9.4	11	16	3.2	11	0
	P		TO		0	49	23	15	396	23	15	303	15	15
	O		TO		0	TO		0	TO		0	TO		0

Empirical Evaluation

- Results:

ID	Dom	Prps	Lazy			Eager			Eager+			Interf.		
			Time	Wid*	Prp	Time	Wid	Prp	Time	Wid	Prp	Time	Wid	Prp
Tbar	I	4	0.1	2	0	0.1	5	0	0.1	5	0	0.1	4	0
	P		0.2	4	4	0.1	5	4	0.1	5	4	0.1	4	4
	O		0.8	3	3	0.1	5	3	0.1	5	3	0.1	4	3
Wsteal	I	5	0.3	6	2	0.1	5	1	0.1	5	1	0.1	4	0
	P		2.4	6	1	0.1	7	1	0.2	7	3	0.1	7	5
	O		8.2	6	4	7.5	6	4	0.2	6	4	6.2	5	4
Cbar	I	9	0.9	3	4	0.1	7	0	0.1	8	0	0.1	7	0
	P		TO		0	1.7	11	4	2.7	12	5	1.1	10	6
	O		TO		0	7.5	9	6	11.3	9	6	6.2	8	4
Phil	I	14	1.9	4	2	0.1	8	2	0.1	8	2	0.1	7	0
	P		11.8	6	14	1.1	11	8	1.8	11	8	6.3	13	14
	O		TO		0	25	12	4	40	12	4	20	12	4
Rb(2,2)	I	16	31.3	8	4	0.4	10	4	0.4	11	4	0.2	10	0
	P		TO		0	9.3	22	3	15	23	3	5.8	15	4
	O		TO		0	142	25	3	225	26	3	105	18	3
Rb(2,3)	I	18	133	8	6	0.7	10	6	0.9	11	6	0.5	10	0
	P		TO		0	23	22	5	36.8	23	5	16	15	5
	O		TO		0	404	25	5	629	26	5	320	18	5
Rb(3,3)	I	23	1141	8	9	1.6	10	9	2.1	11	9	0.9	10	0
	P		TO		0	68.2	22	8	111.5	23	8	52	15	8
	O		TO		0	1414	25	8	2139	26	8	1168	18	8
Rb(4,4)	I	29	TO		0	6.7	11	16	9.4	11	16	3.2	11	0
	P		TO		0	49	23	15	396	23	15	303	15	15
	O		TO		0	TO		0	TO		0	TO		0

Still **interference** abstraction is the **fastest**

Empirical Evaluation

- Results:

ID	Dom	Prps	Lazy			Eager			Eager+			Interf.		
			Time	Wid*	Prp	Time	Wid	Prp	Time	Wid	Prp	Time	Wid	Prp
Tbar	I	4	0.1	2	0	0.1	5	0	0.1	5	0	0.1	4	0
	P		0.2	4	4	0.1	5	4	0.1	5	4	0.1	4	4
	O		0.8	3	3	0.1	5	3	0.1	5	3	0.1	4	3
Wsteal	I	5	0.3	6	2	0.1	5	1	0.1	5	1	0.1	4	0
	P		2.4	6	1	0.1	7	1	0.2	7	3	0.1	7	5
	O		8.2	6	4	7.5	6	4	0.2	6	4	6.2	5	4
Cbar	I	9	0.9	3	4	0.1	7	0	0.1	8	0	0.1	7	0
	P		TO		0	1.7	11	4	2.7	12	5	1.1	10	6
	O		TO		0	7.5	9	6	11.3	9	6	6.2	8	4
Phil	I	14	1.9	4	2	0.1	8	2	0.1	8	2	0.1	7	0
	P		11.8	6	14	1.1	11	8	1.8	11	8	6.3	13	14
	O		TO		0	25	12	4	40	12	4	20	12	4
Rb(2,2)	I	16	31.3	8	4	0.4	10	4	0.4	11	4	0.2	10	0
	P		TO		0	9.3	22	3	15	23	3	5.8	15	4
	O		TO		0	142	25	3	225	26	3	105	18	3
Rb(2,3)	I	18	133	8	6	0.7	10	6	0.9	11	6	0.5	10	0
	P		TO		0	23	22	5	36.8	23	5	16	15	5
	O		TO		0	404	25	5	629	26	5	320	18	5
Rb(3,3)	I	23	1141	8	9	1.6	10	9	2.1	11	9	0.9	10	0
	P		TO		0	68.2	22	8	111.5	23	8	52	15	8
	O		TO		0	1414	25	8	2139	26	8	1168	18	8
Rb(4,4)	I	29	TO		0	6.7	11	16	9.4	11	16	3.2	11	0
	P		TO		0	49	23	15	396	23	15	303	15	15
	O		TO		0	TO		0	TO		0	TO		0

Lazy abstraction was the **slowest** (with many TO)

Empirical Evaluation

- Results:

ID	Dom	Time(Lazy)	Time(Eager)	Time(Eager+)	Time(Interf)
Wsteal	P	2.4	0.1	0.2	0.1
	O	8.2	7.5	0.2	6.2
Cbar	P	TO	1.7	2.7	1.1
	O	TO	7.5	11.3	6.2
Phil	P	11.8	1.1	1.8	6.3
	O	TO	25	40	20
Rb(3,3)	P	TO	68.2	111.5	52
	O	TO	1414	2139	1168

Polyhedra in general results faster than **octagons**

(possibly due to Apron library)

Empirical Evaluation

- Results:

ID	Dom	Prps	Prp(Lazy)	Prp(Eager)	Prp(Eager+)	Prp(Interf)
Wsteal	I	5	2	1	1	0
	P		1	1	3	5
Cbar	I	9	4	0	0	0
	P		0	4	5	6
Phil	I	14	2	2	2	0
	P		14	8	8	14
Rb(4,4)	I	29	0	16	16	0
	P		0	15	15	15

Interference infers **more properties in polyhedra**,

Empirical Evaluation

- Results:

ID	Dom	Prps	Prp(Lazy)	Prp(Eager)	Prp(Eager+)	Prp(Interf)
Wsteal	I	5	2	1	1	0
	P		1	1	3	5
Cbar	I	9	4	0	0	0
	P		0	4	5	6
Phil	I	14	2	2	2	0
	P		14	8	8	14
Rb(4,4)	I	29	0	16	16	0
	P		0	15	15	15

Interference infers **more properties in polyhedra**, but **less in intervals**

Empirical Evaluation

- Relative strengths of inferred invariants

ID	Dom	L:E	L:E+	L:In	E:In	E+:In	E:E+
Tbar	I	−	−	+	+	+	=
	P	=	=	+	+	+	=
	O	=	=	+	+	+	=
Wsteal	I	+	+	+	+	+	=
	P	+	≠	≠	≠	≠	−
	O	=	=	+	+	+	=
Cbar	I	≠	≠	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	+	+	=
Phil	I	+	+	+	+	+	=
	P	+	+	+	−	−	=
	O	TO	TO	TO	+	+	=
Rb(2,2)	I	+	+	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(2,3)	I	+	+	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(3,3)	I	+	+	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(4,4)	I	TO	TO	TO	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	TO	TO	TO

Empirical Evaluation

- Relative strengths of inferred invariants

ID	Dom	L:E	L:E+	L:In	E:In	E+:In	E:E+
Tbar	I	−	−	+	+	+	=
	P	=	=	+	+	+	=
	O	=	=	+	+	+	=
Wsteal	I	+	+	+	+	+	=
	P	+	≠	≠	≠	≠	−
	O	=	=	+	+	+	=
Cbar	I	≠	≠	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	+	+	=
Phil	I	+	+	+	+	+	=
	P	+	+	+	−	−	=
	O	TO	TO	TO	+	+	=
Rb(2,2)	I	+	+	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(2,3)	I	+	+	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(3,3)	I	+	+	+	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(4,4)	I	TO	TO	TO	+	+	=
	P	TO	TO	TO	≠	≠	−
	O	TO	TO	TO	TO	TO	TO

Lazy, eager and eager+ prove stronger invariants for **interval domain** compared to *interference*

Empirical Evaluation

- Relative strengths of inferred invariants

ID	Dom	L:E	L:E+	L:In	E:In	E+:In	E:E+
Tbar	I	−	−	+	+	+	=
	P	=	=	+	+	+	=
	O	=	=	+	+	+	=
Wsteal	I	+	+	+	+	+	=
	P	+	≠	≠	≠	≠	−
	O	=	=	+	+	+	=
Cbar	I	≠	≠	+	+	+	=
	P	TO	TO	**TO**	≠	≠	−
	O	TO	TO	TO	+	+	=
Phil	I	+	+	+	+	+	=
	P	+	+	+	−	−	=
	O	TO	TO	TO	+	+	=
Rb(2,2)	I	+	+	+	+	+	=
	P	TO	TO	**TO**	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(2,3)	I	+	+	+	+	+	=
	P	TO	TO	**TO**	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(3,3)	I	+	+	+	+	+	=
	P	TO	TO	**TO**	≠	≠	−
	O	TO	TO	TO	≠	+	−
Rb(4,4)	I	TO	TO	TO	+	+	=
	P	TO	TO	**TO**	≠	≠	−
	O	TO	TO	TO	TO	TO	TO

Trend is **reversed** for **polyhedra domain**

Conclusions

- We presented a new technique: **Reflective Abstraction**
- Helpful to **infer k-indexed invariants** on **parametrized systems**
- Enables leveraging **off-the-shelf** invariant generators
- We studied three variants of **reflective abstraction**, analyzing their relative strength in inferring invariants
- Possible future directions, study the possibility of **additional structure** on the summarized threads