Communicating State Transition Systems for Fine-Grained Concurrent Resources

Aleks Nanevski   Ruy Ley-Wild   **Ilya Sergey**   Germán Delbianco

institute IMDEA software   LOGICBLOX®

ESOP 2014

Good programs are compositional

Reasoning about programs should be compositional

Reasoning about concurrent programs should be compositional

Reasoning about concurrent programs combines reasoning about *resources* and *threads*

Adding more resources

$\{P\}\ \ C\ \ \{Q\}$

Adding more resources

$$R \vdash \{P\} \quad C \quad \{Q\}$$

Adding more resources

$$\frac{R \vdash \{P\}\ C\ \{Q\}}{R * S \vdash \{P * \Delta_s\} C \{Q * \Delta_s\}}$$

Adding more resources

$$\frac{R \vdash \{P\}\ \ C\ \ \{Q\}}{R * \boxed{S} \vdash \{P * \Delta_s\} C \{Q * \Delta_s\}}$$

Adding more resources

$$\frac{R \vdash \{P\}\ \ C\ \ \{Q\}}{R * \boxed{S} \vdash \{P * \boxed{\Delta_s}\}\, C\, \{Q * \boxed{\Delta_s}\}}$$

Adding more resources

$$\frac{R \vdash \{P\}\ \ C\ \ \{Q\}}{R * \boxed{S} \vdash \{P * \boxed{\Delta_s}\} C \{Q * \boxed{\Delta_s}\}}$$

R and S don’t overlap *at all*.

Adding more resources

$$\frac{R \vdash \{P\}\ \ C\ \ \{Q\}}{R * \boxed{S} \vdash \{P * \boxed{\Delta_s}\} C \{Q * \boxed{\Delta_s}\}}$$

R and S don’t overlap *at all*.

“frame rule”

Adding more resources

$$\frac{R \vdash \{P\}\ \ C\ \ \{Q\}}{R \bowtie S \vdash \{???\}\ C\ \{???\}}$$

R and S don’t overlap *at each moment*.

Adding more resources

$$\frac{R \vdash \{P\}\ \ C\ \ \{Q\}}{R \bowtie S \vdash \{???\}\ C\ \{???\}}$$

R and S don’t overlap *at each moment*.

Cannot *reuse* the proof of $R \vdash \{P\} C \{Q\}$.

Forking more threads

$\{P\}\ \ C\ \ \{Q\}$

Forking more threads

{P} C {Q}

{P} C {Q}

Forking more threads

$$c \,||\, c$$

Forking more threads

$$\{\mathcal{F}_{\mathrm{xy}}(\mathrm{P})\}$$

$$\mathrm{C} \;||\; \mathrm{C}$$

$$\{\mathcal{F}_{\mathrm{xy}}(\mathrm{Q})\}$$

Forking more threads

$\{\mathcal{F}_{xy}(P)\}$

$C \;||\; C$

$\{\mathcal{F}_{xy}(Q)\}$

Forking more threads

$\{\mathcal{F}_{xy}(P)\}$

$C \mid\mid C$

$\{\mathcal{F}_{xy}(Q)\}$

$\{P\}\ C\ \{Q\}$

Forking more threads

$$\{\mathcal{F}_{\mathrm{xyz}}(\mathrm{P})\}$$

$$\mathrm{C} \;||\; \mathrm{C} \;||\; \mathrm{C}$$

$$\{\mathcal{F}_{\mathrm{xyz}}(\mathrm{Q})\}$$

Forking more threads

$$\{\mathcal{F}_{\mathrm{xyz}}(\mathrm{P})\}$$

$$\mathrm{C} \;||\; \mathrm{C} \;||\; \mathrm{C}$$

$$\{\mathcal{F}_{\mathrm{xyz}}(\mathrm{Q})\}$$

Cannot *reuse* the proof for C || C.

Two dimensions of scalability

Two dimensions of scalability

Number of resources

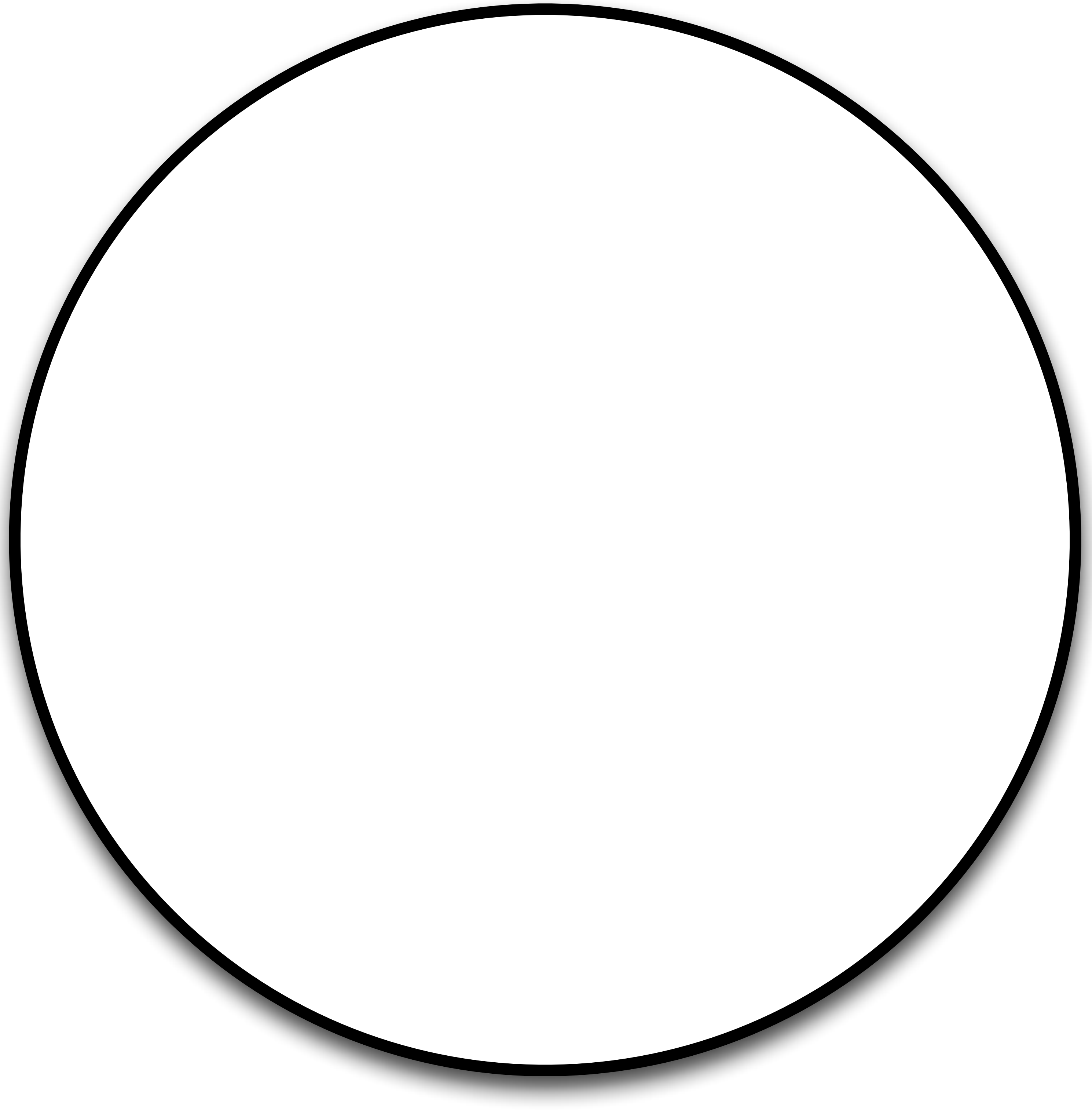
Structure and number of threads

This work

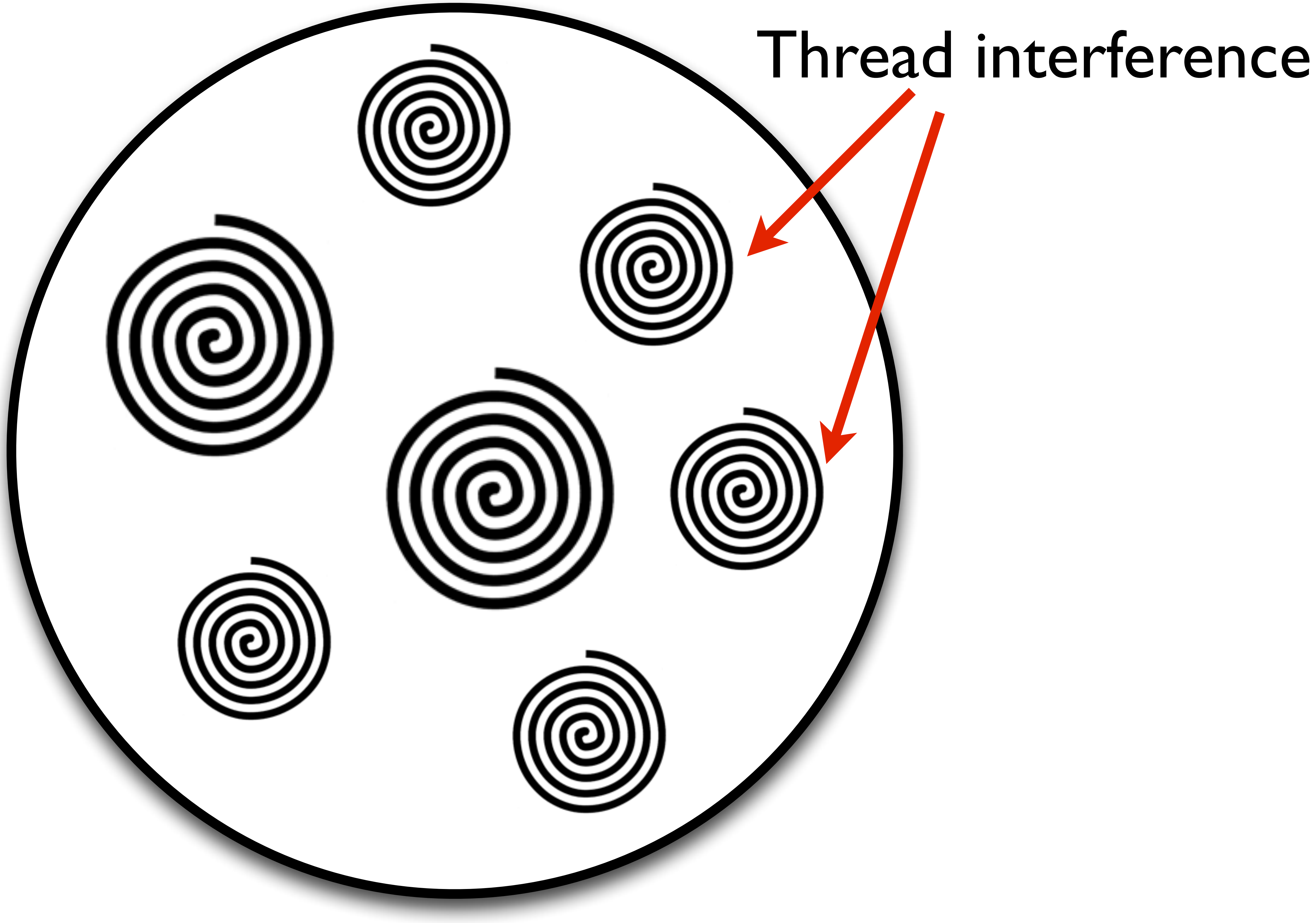
A model for
compositional reasoning
about shared-memory concurrency

(in both dimensions)

Shared Memory

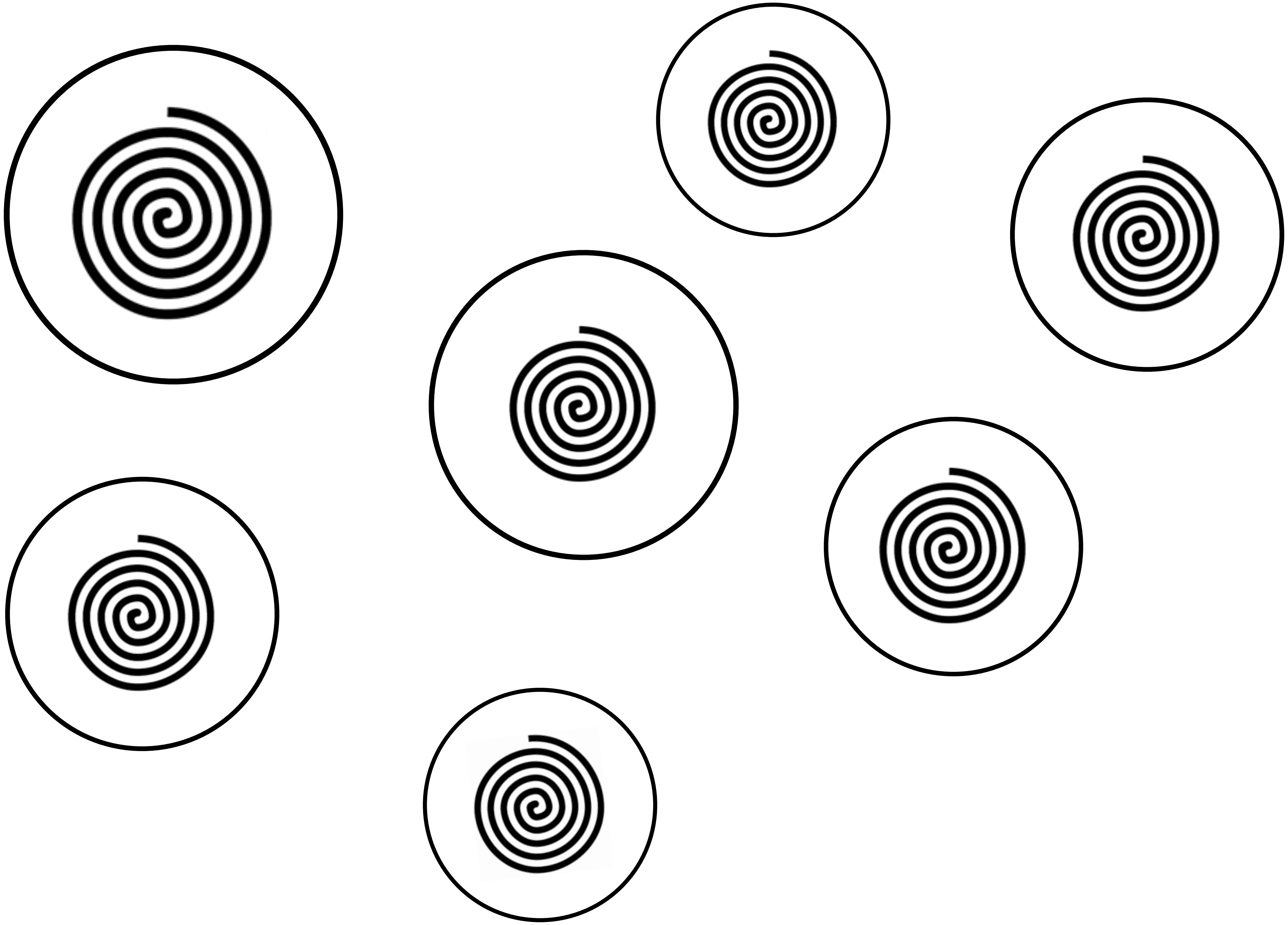


Shared Memory



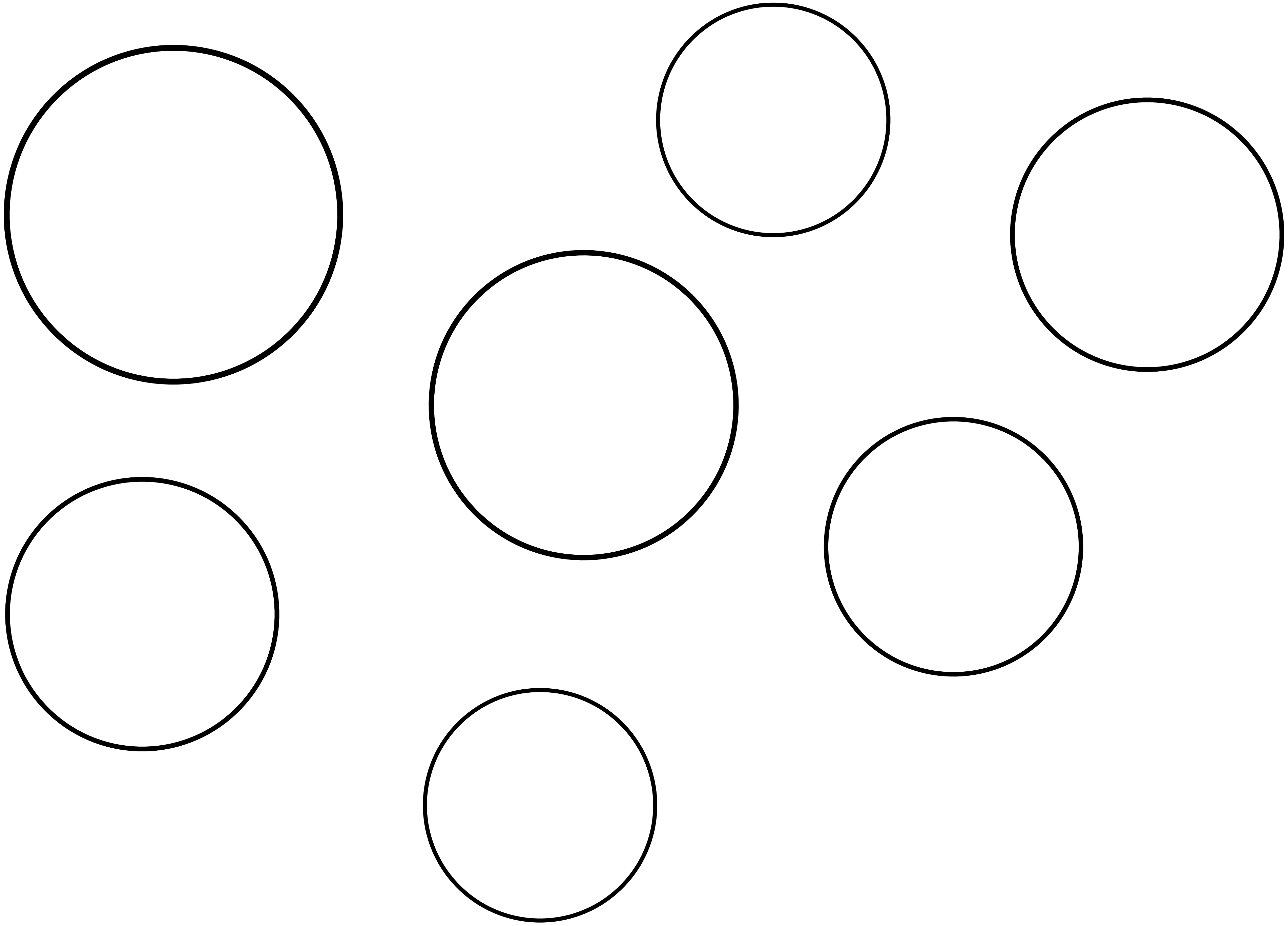
Thread interference

Disjoint Regions in Shared Memory



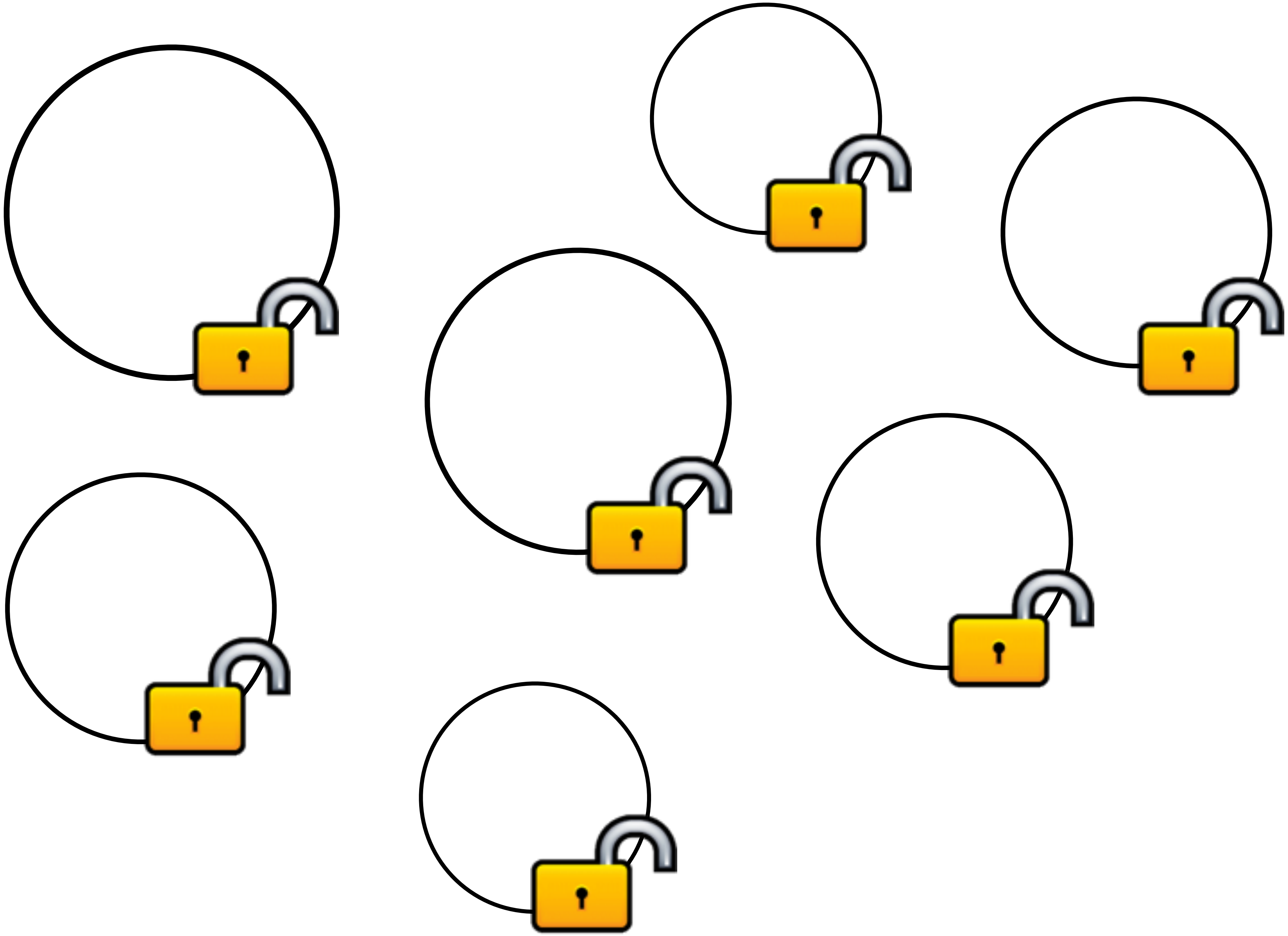
Critical Regions of Shared Memory

a.k.a Coarse-Grained Concurrency



Critical Regions of Shared Memory

a.k.a Coarse-Grained Concurrency



Critical Regions with Ownership Transfer

a.k.a Coarse-Grained Concurrency

Concurrent Separation Logic

O'Hearn [CONCUR'04], Brookes [CONCUR'04]

Critical Regions with Ownership Transfer

a.k.a Coarse-Grained Concurrency

Critical Regions with Ownership Transfer

a.k.a Coarse-Grained Concurrency

Critical Regions with Ownership Transfer

a.k.a Coarse-Grained Concurrency

Critical Regions with Ownership Transfer

a.k.a Coarse-Grained Concurrency

- Critical Regions — **State Transition Systems** (*Locked*, *Unlocked*);

  DinsdaleYoung-al:ECOOP'10, O'Hearn-al:PODC'10, Turon-al:POPL'13, Turon-al:ICFP'13, Svendsen-al:ESOP'13, Svendsen-Birkedal:ESOP'14, daRochaPinto-al:ECOOP'14…

Critical Regions with Ownership Transfer

a.k.a Coarse-Grained Concurrency

- Critical Regions — **State Transition Systems** (*Locked*, *Unlocked*);

  DinsdaleYoung-al:ECOOP'10, O'Hearn-al:PODC'10, Turon-al:POPL'13, Turon-al:ICFP'13, Svendsen-al:ESOP'13, Svendsen-Birkedal:ESOP'14, daRochaPinto-al:ECOOP'14…

- Ownership Transfer is a way to think of "somewhat overlapping" resources;

Critical Regions with Ownership Transfer

a.k.a Coarse-Grained Concurrency

- Critical Regions — **State Transition Systems** (*Locked*, *Unlocked*);

  DinsdaleYoung-al:ECOOP'10, O'Hearn-al:PODC'10, Turon-al:POPL'13, Turon-al:ICFP'13, Svendsen-al:ESOP'13, Svendsen-Birkedal:ESOP'14, daRochaPinto-al:ECOOP'14…

- Ownership Transfer is a way to think of "somewhat overlapping" resources;

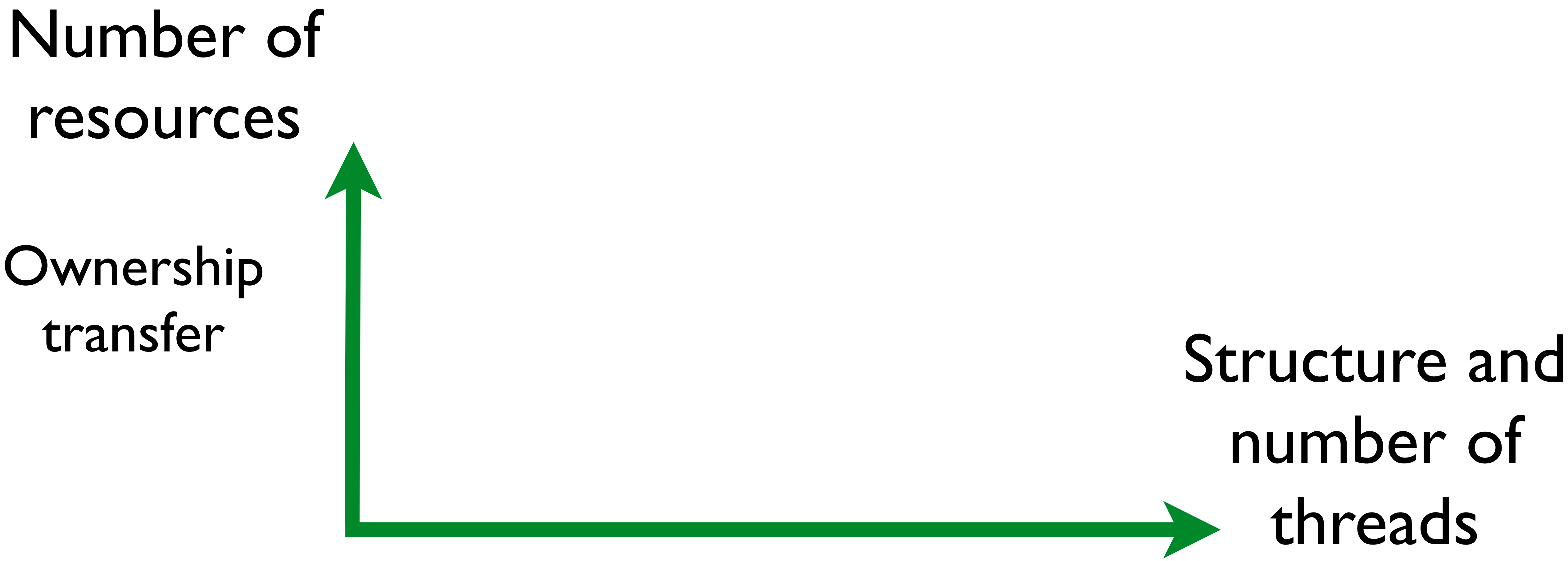
- Ownership Transfer — **Communication** between resources.
  [This work]

Two dimensions of scalability

Number of resources

Structure and number of threads

Two dimensions of scalability



Number of resources

Ownership transfer

Structure and number of threads

Two dimensions of scalability

Number of resources

~~Ownership transfer~~

Communication

Structure and number of threads

Two dimensions of scalability

Number of resources

~~Ownership transfer~~

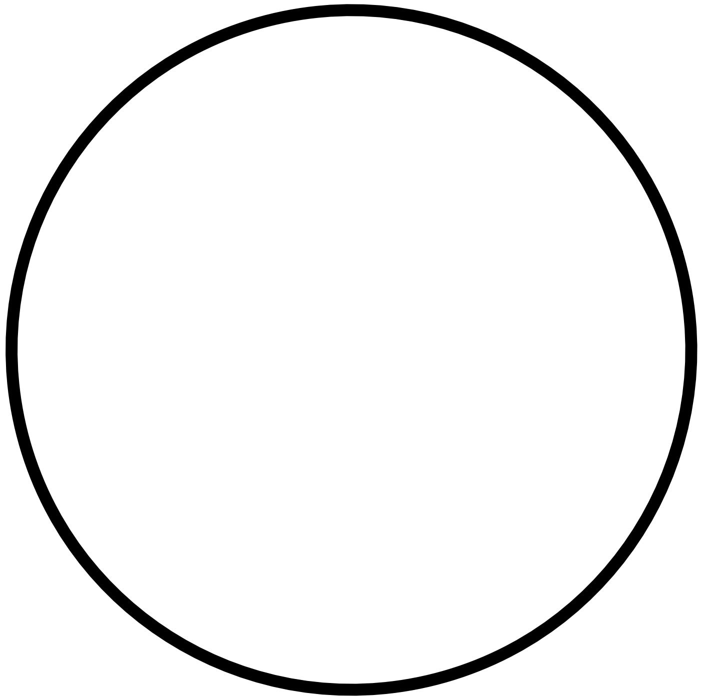
Communication

Structure and number of threads

???

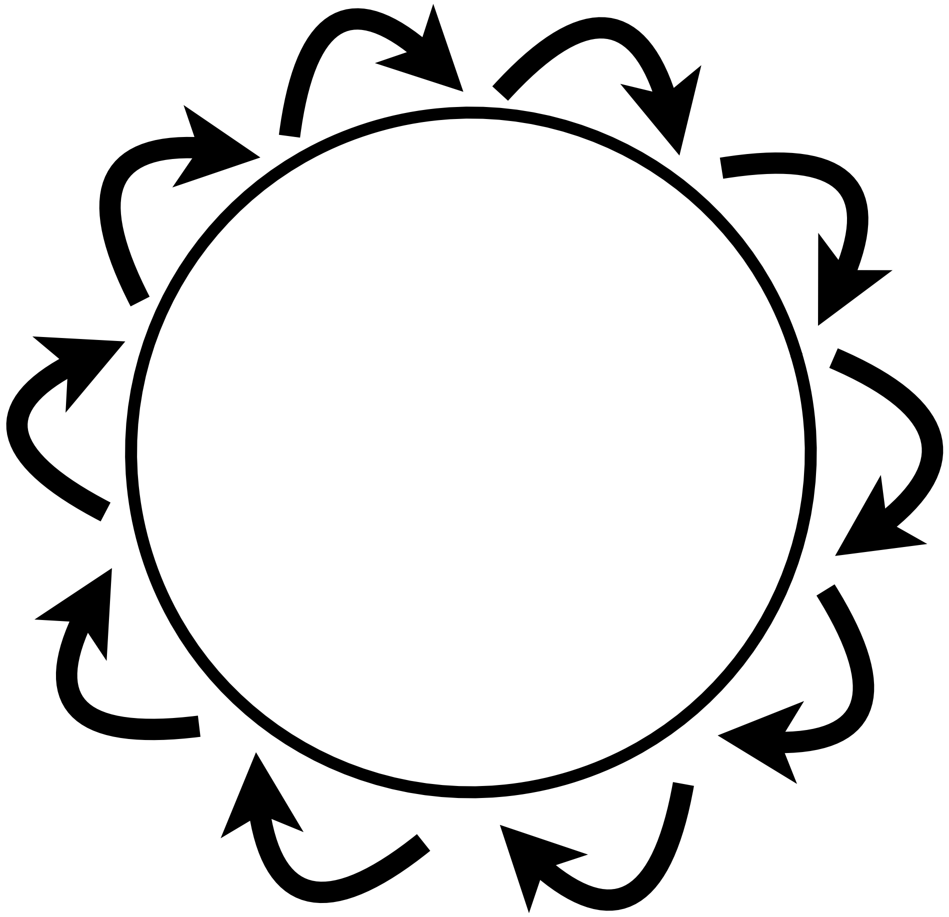
Resources with Arbitrary Transitions

a.k.a Fine-Grained Concurrency



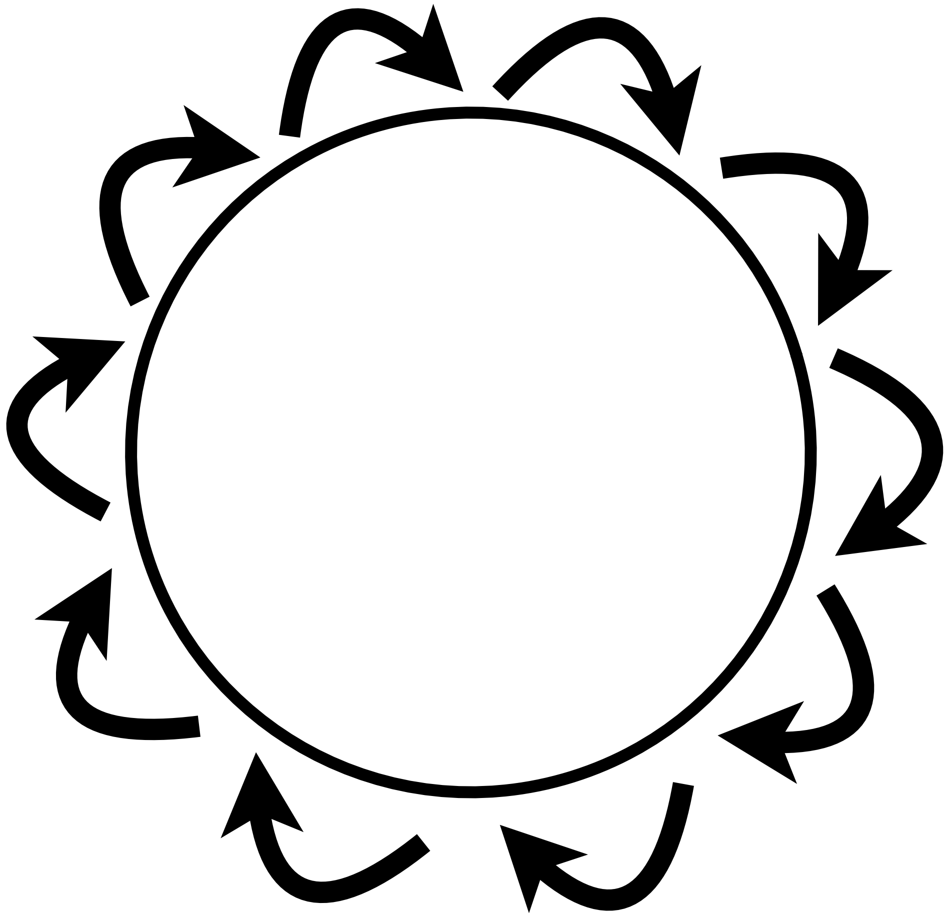
Resources with Arbitrary Transitions

a.k.a Fine-Grained Concurrency



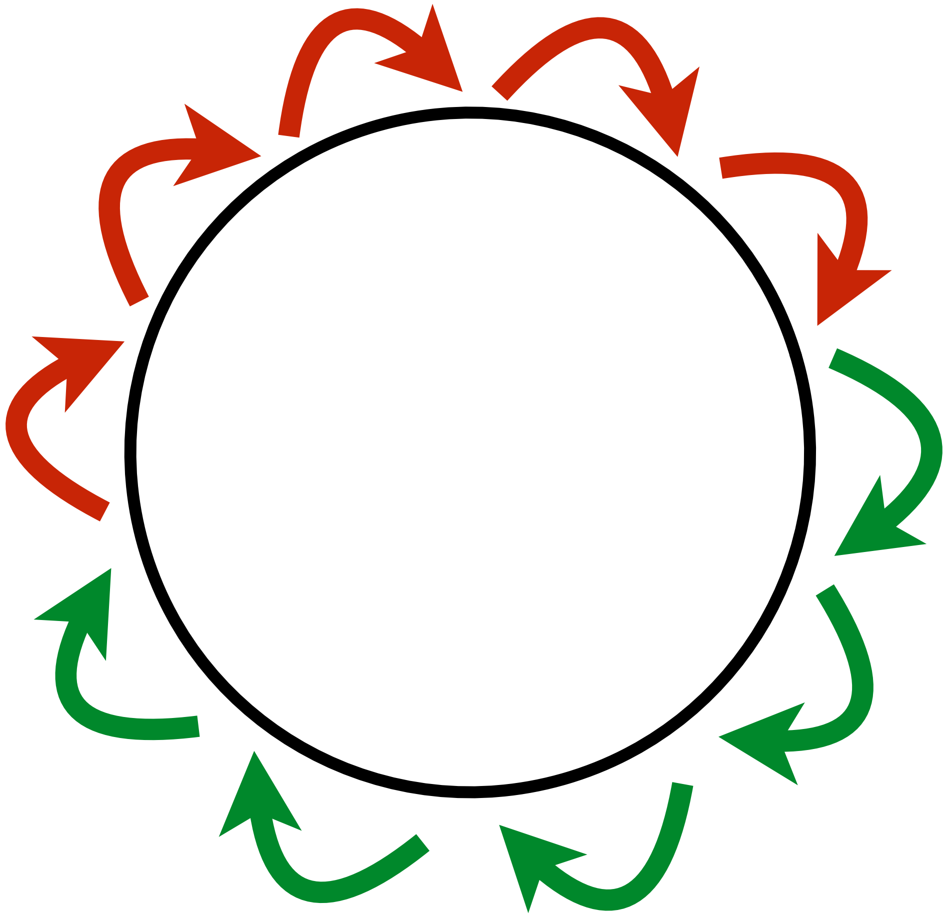
Resources with Arbitrary Transitions

a.k.a Fine-Grained Concurrency

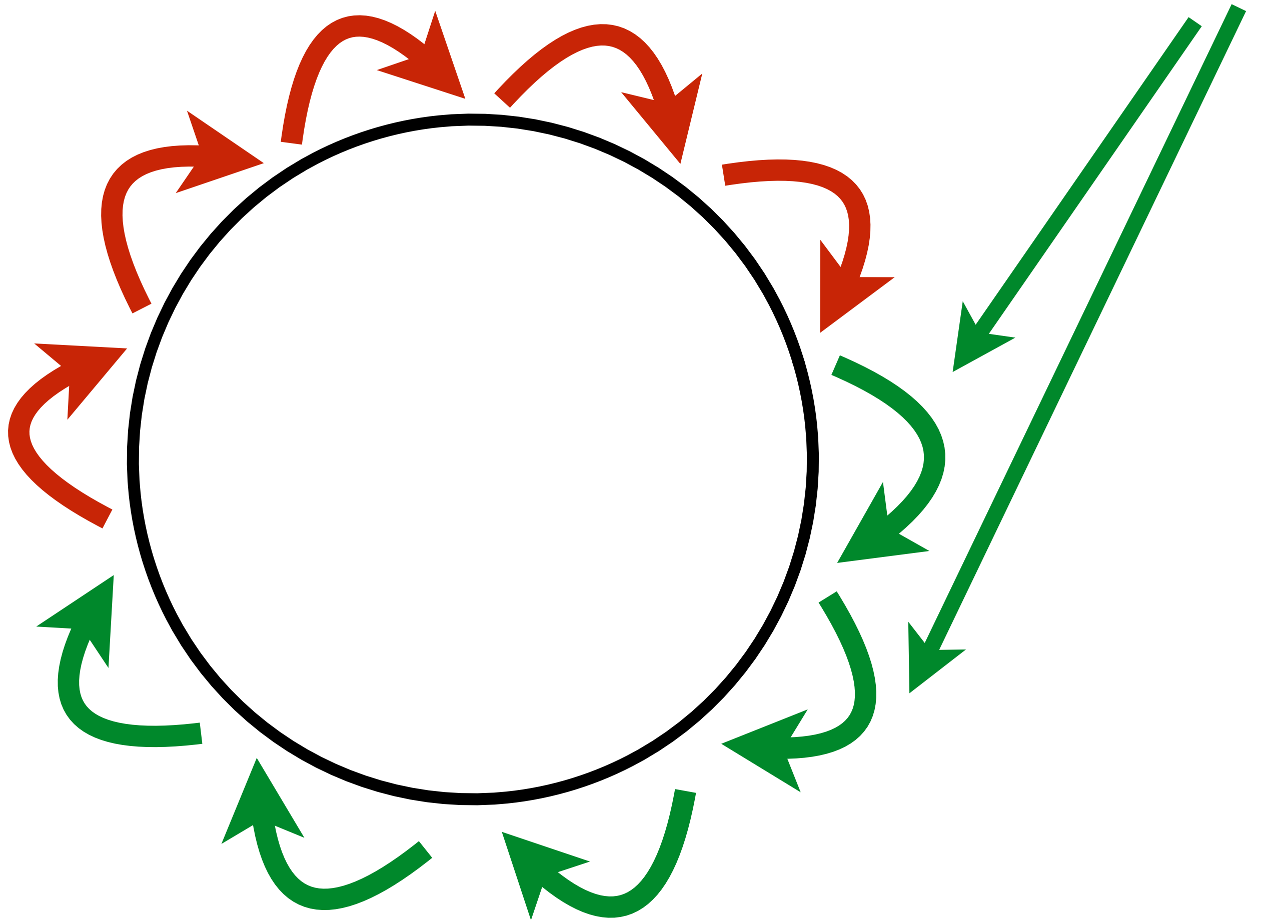


Need to decide **what** each thread is allowed to do!

Subjective Specifications for Arbitrary Transitions



Subjective Specifications for Arbitrary Transitions

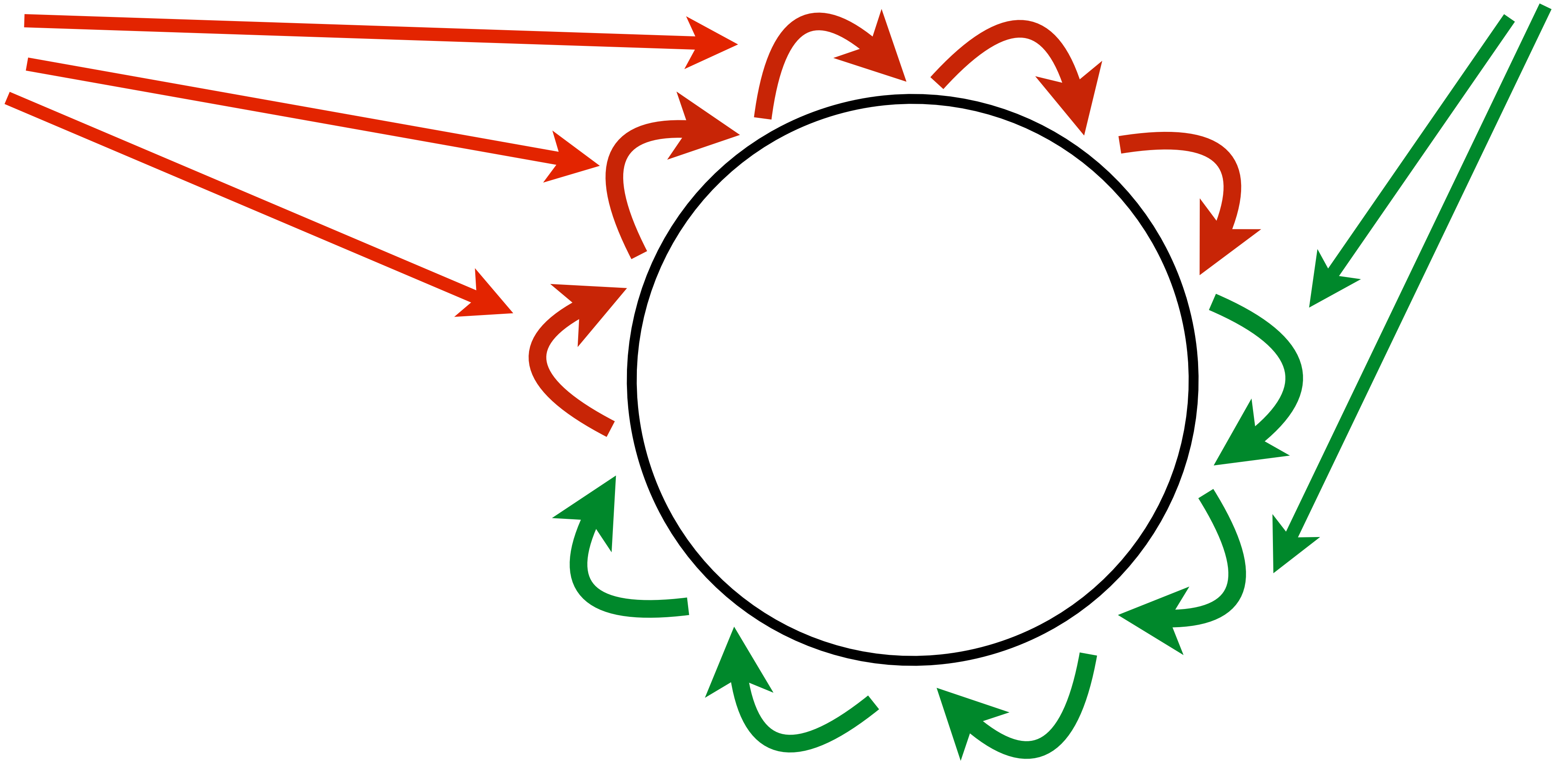


Transitions allowed to <u>myself</u>
(*Guarantee*)

Subjective Specifications for Arbitrary Transitions

Transitions allowed to the others
(*Rely*)

Transitions allowed to myself
(*Guarantee*)

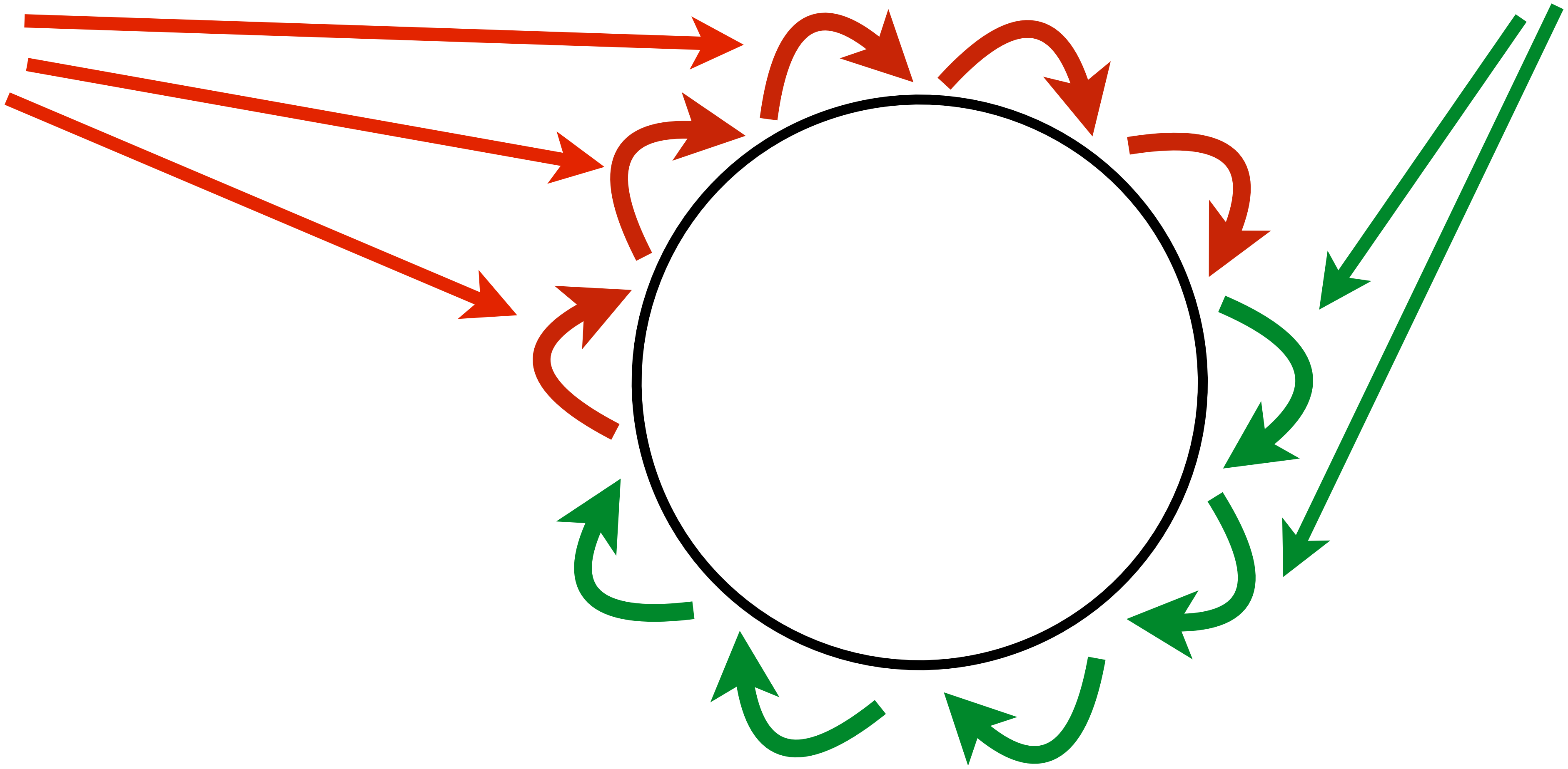


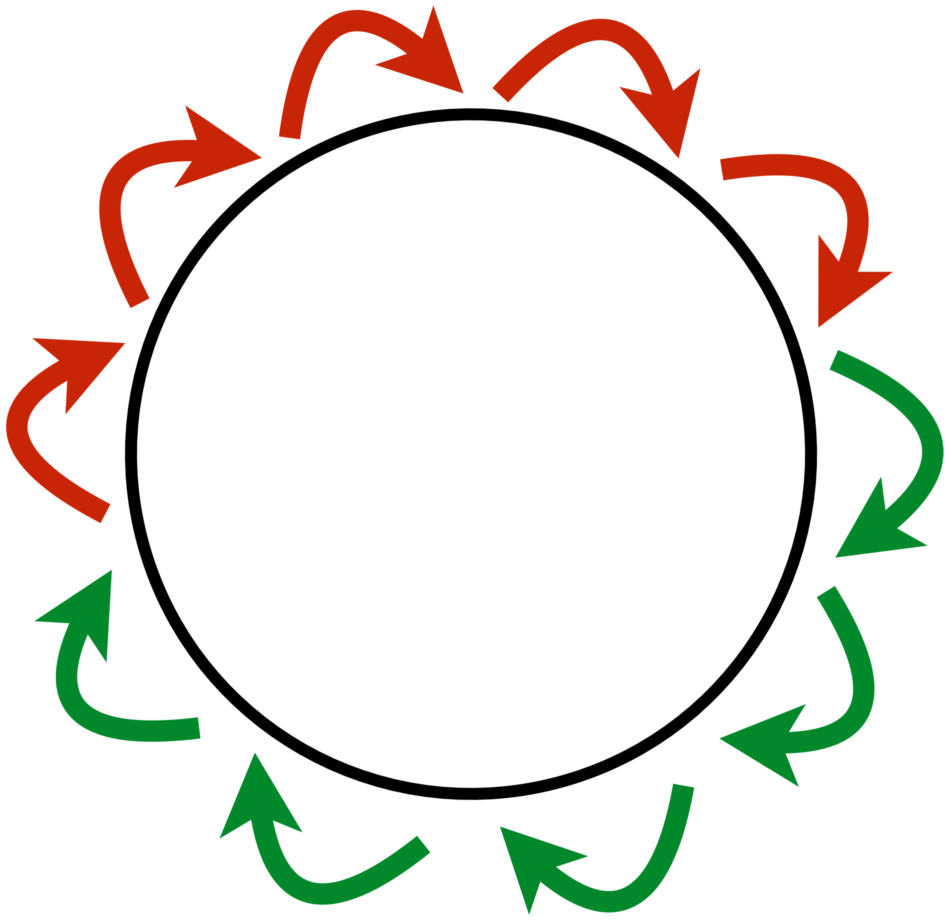
Subjective Specifications for Arbitrary Transitions

Rely-Guarantee Reasoning, Jones [TOPLAS83]

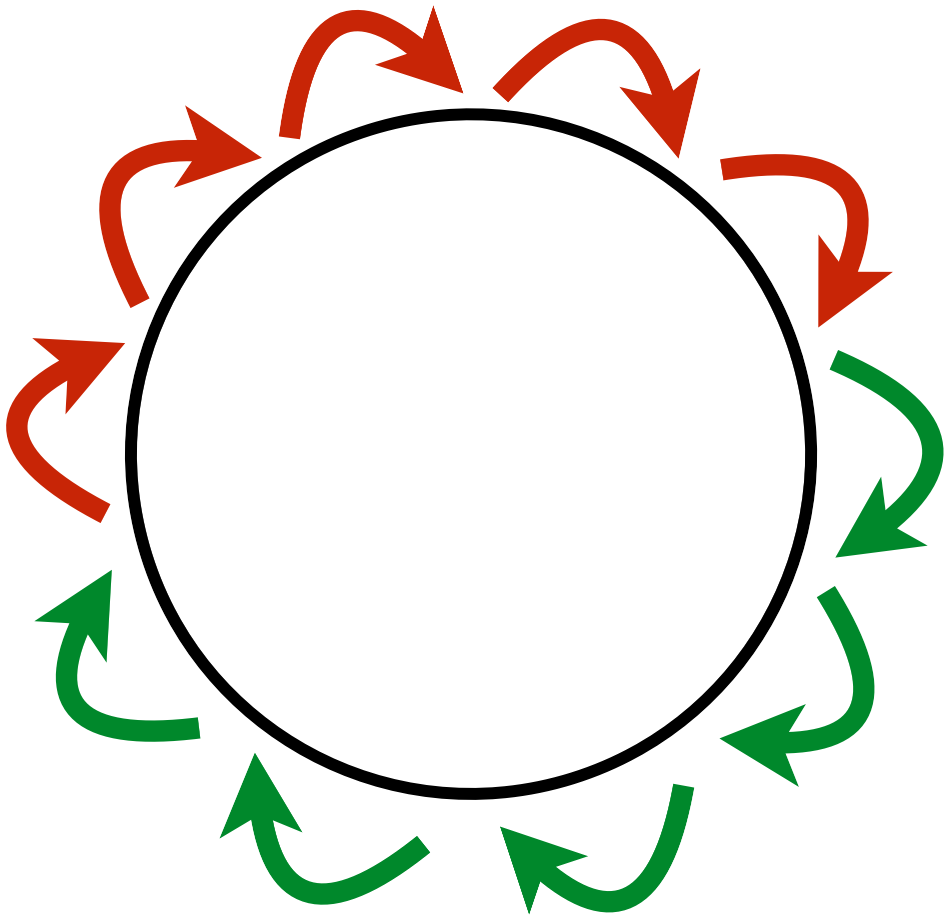
Transitions allowed to the <u>others</u>
(*Rely*)

Transitions allowed to <u>myself</u>
(*Guarantee*)



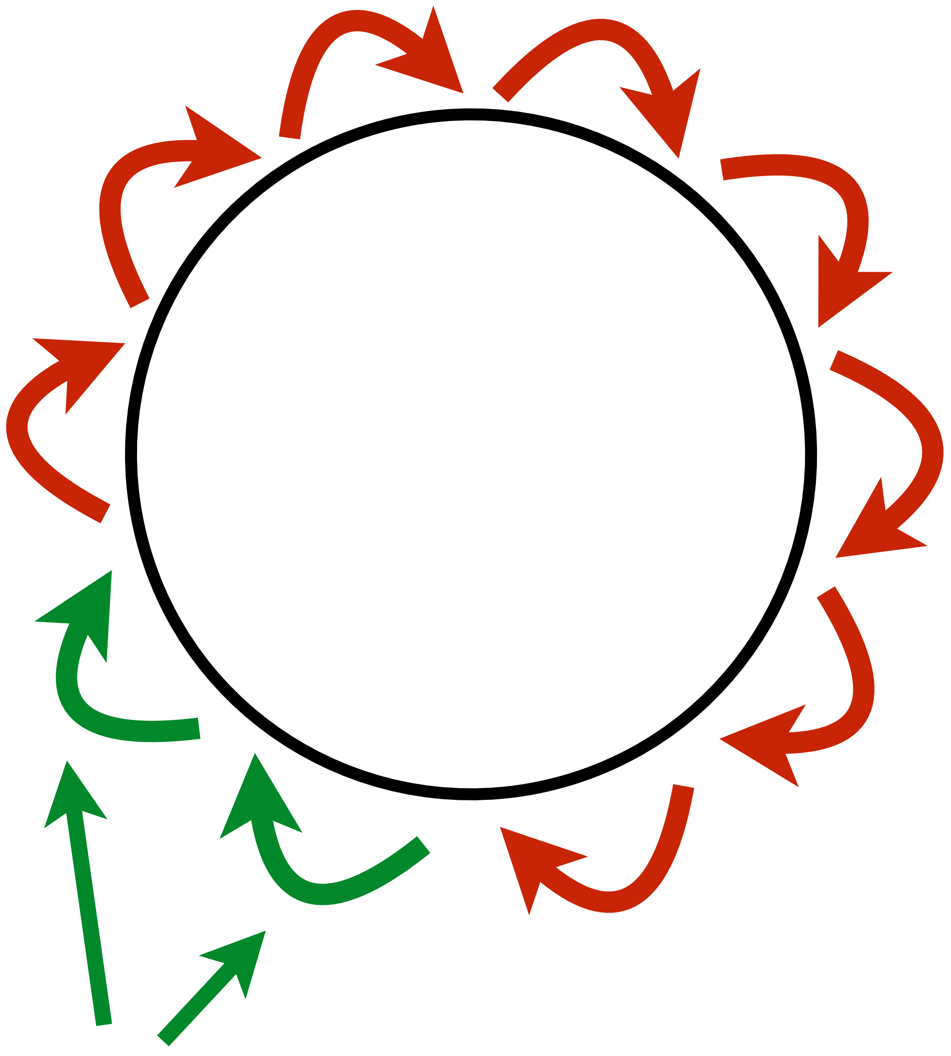


myself



myself

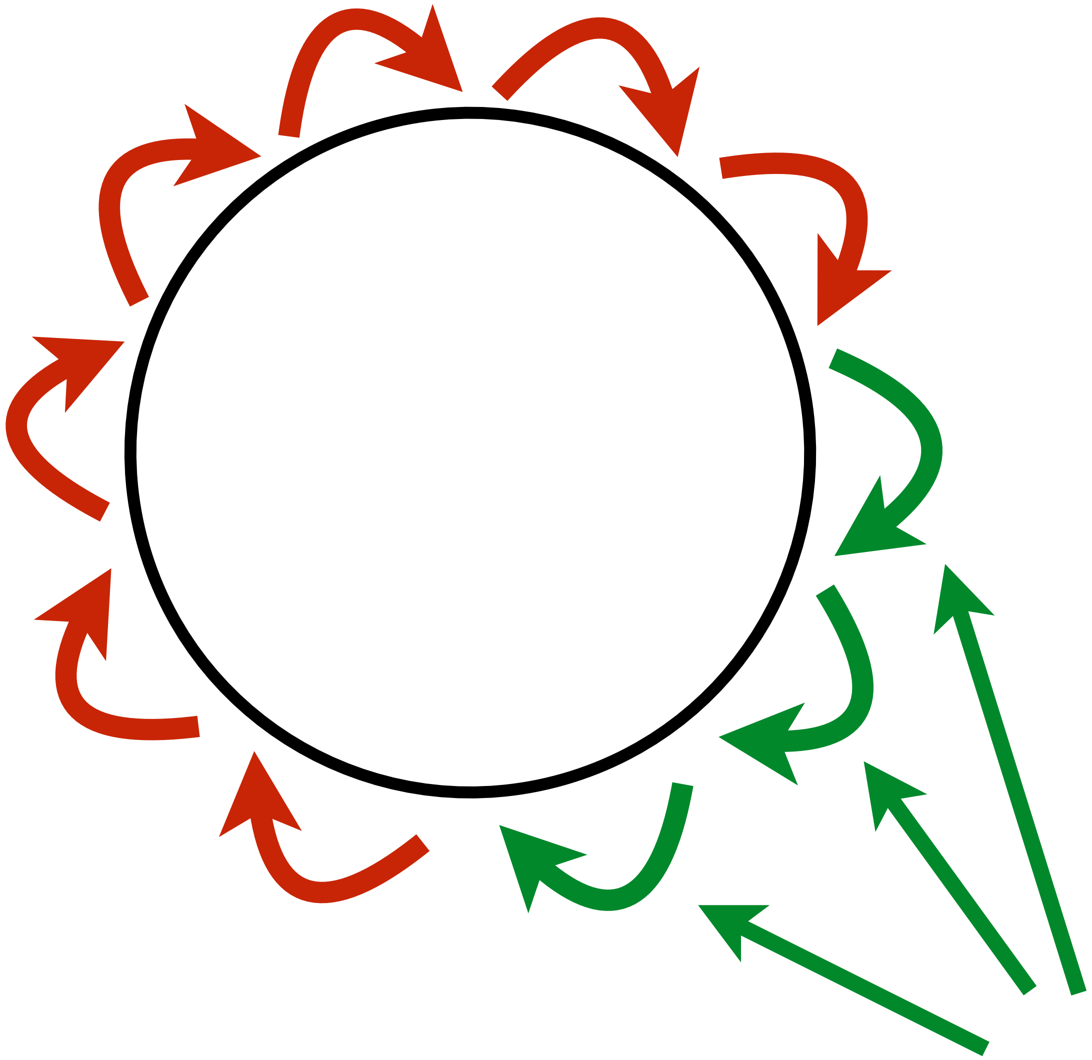
self(1) || self(2)



Transitions allowed to self(1)

myself

self(1) || self(2)



Transitions allowed to self(2)

myself

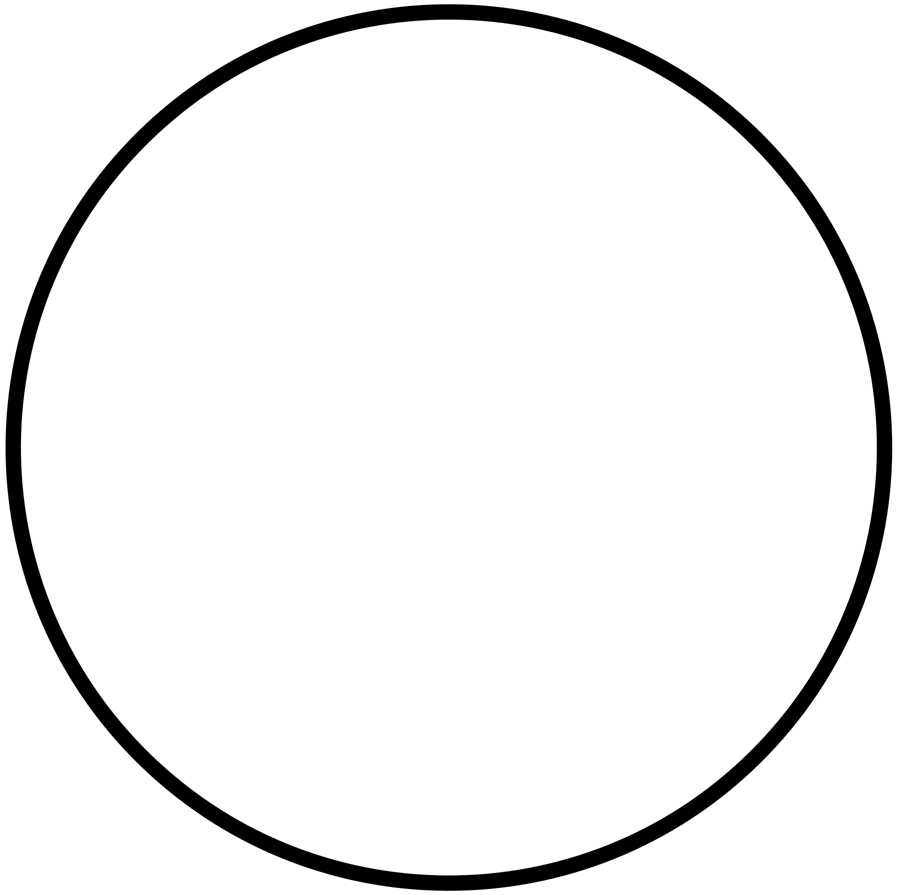
self(1) || self(2)

$$\frac{R \lor G_2, \boxed{G_1} \vdash \{p\}\ c_1\ \{q_1\} \qquad R \lor G_1, \boxed{G_2} \vdash \{p\}\ c_2\ \{q_2\}}{R, G_1 \lor G_2 \vdash \{p\}\ c_1 \parallel c_2\ \{q_1 \land q_2\}}\ \textsc{ParRG}$$

$$\frac{R \lor \boxed{G_2}, G_1 \vdash \{p\}\ c_1\ \{q_1\} \qquad R \lor \boxed{G_1}, G_2 \vdash \{p\}\ c_2\ \{q_2\}}{R, G_1 \lor G_2 \vdash \{p\}\ c_1 \parallel c_2\ \{q_1 \land q_2\}}\ \textsc{ParRG}$$

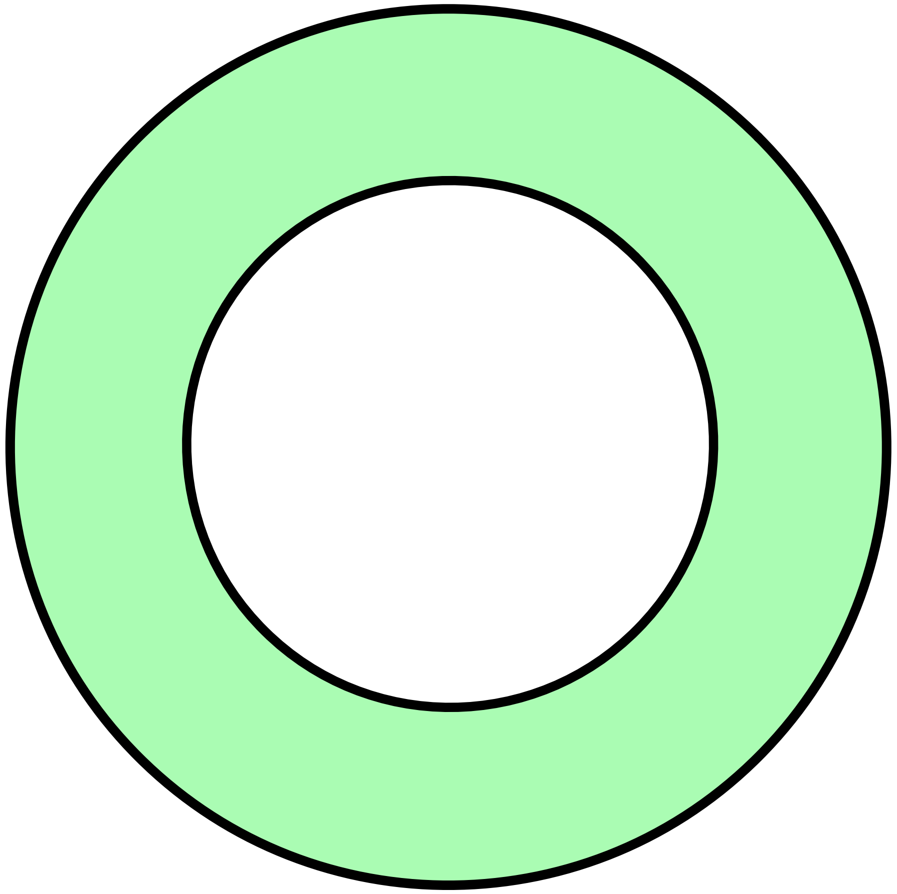
“Forking shuffle”

Reasoning about State



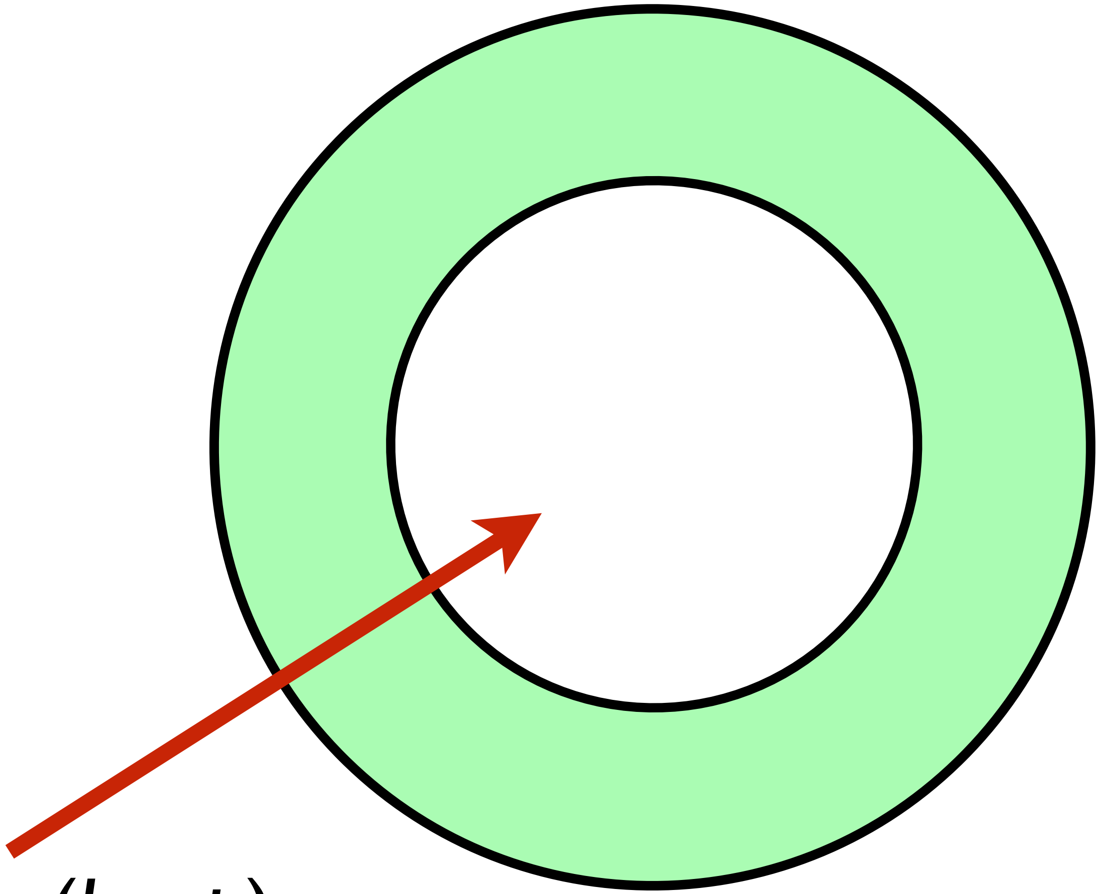
Auxiliary State

Hansen [CompSurv'73], Lauer[PhD'73], Owicki-Gries[CACM'76]



Auxiliary State

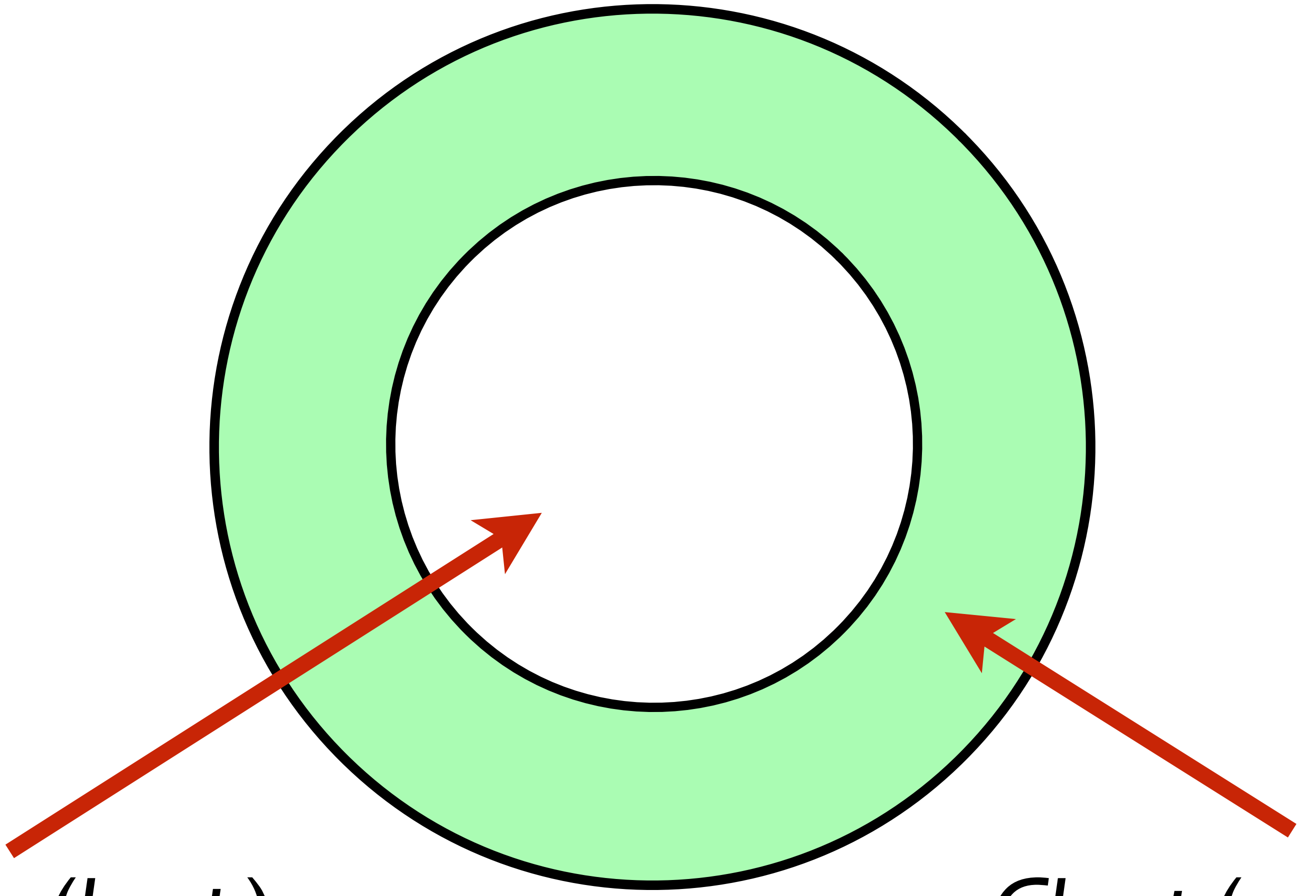
Hansen [CompSurv’73], Lauer[PhD’73], Owicki-Gries[CACM’76]



Real state (*heap*)

Auxiliary State

Hansen [CompSurv'73], Lauer[PhD'73], Owicki-Gries[CACM'76]

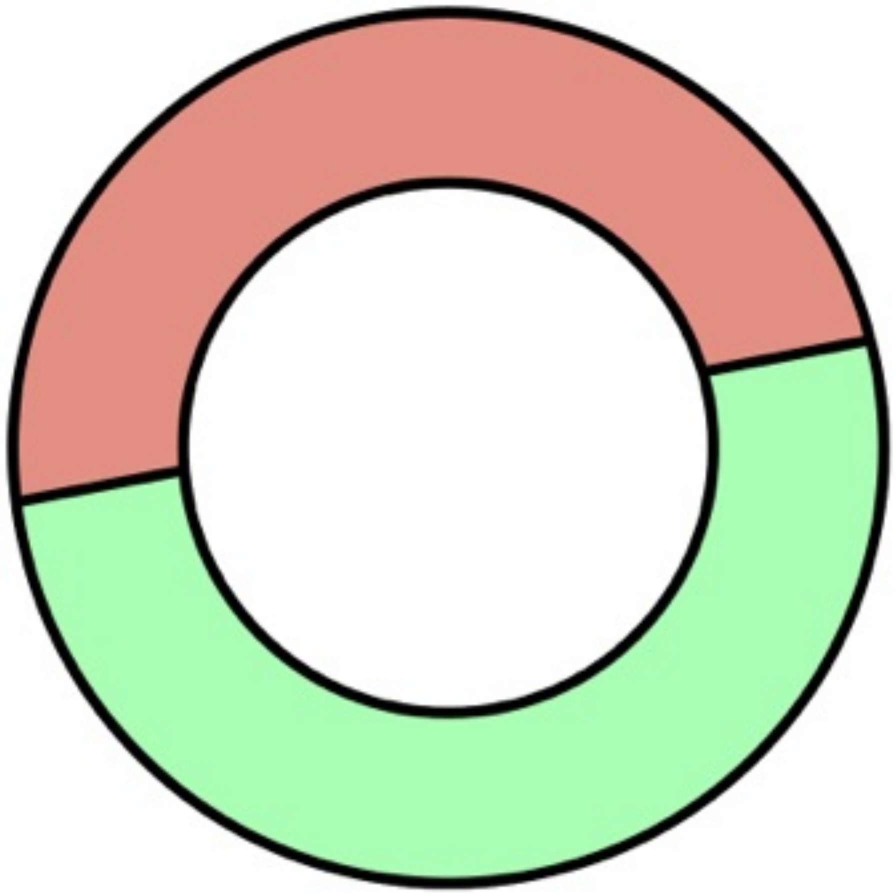


Real state (*heap*)

Ghost (*auxiliary*) state

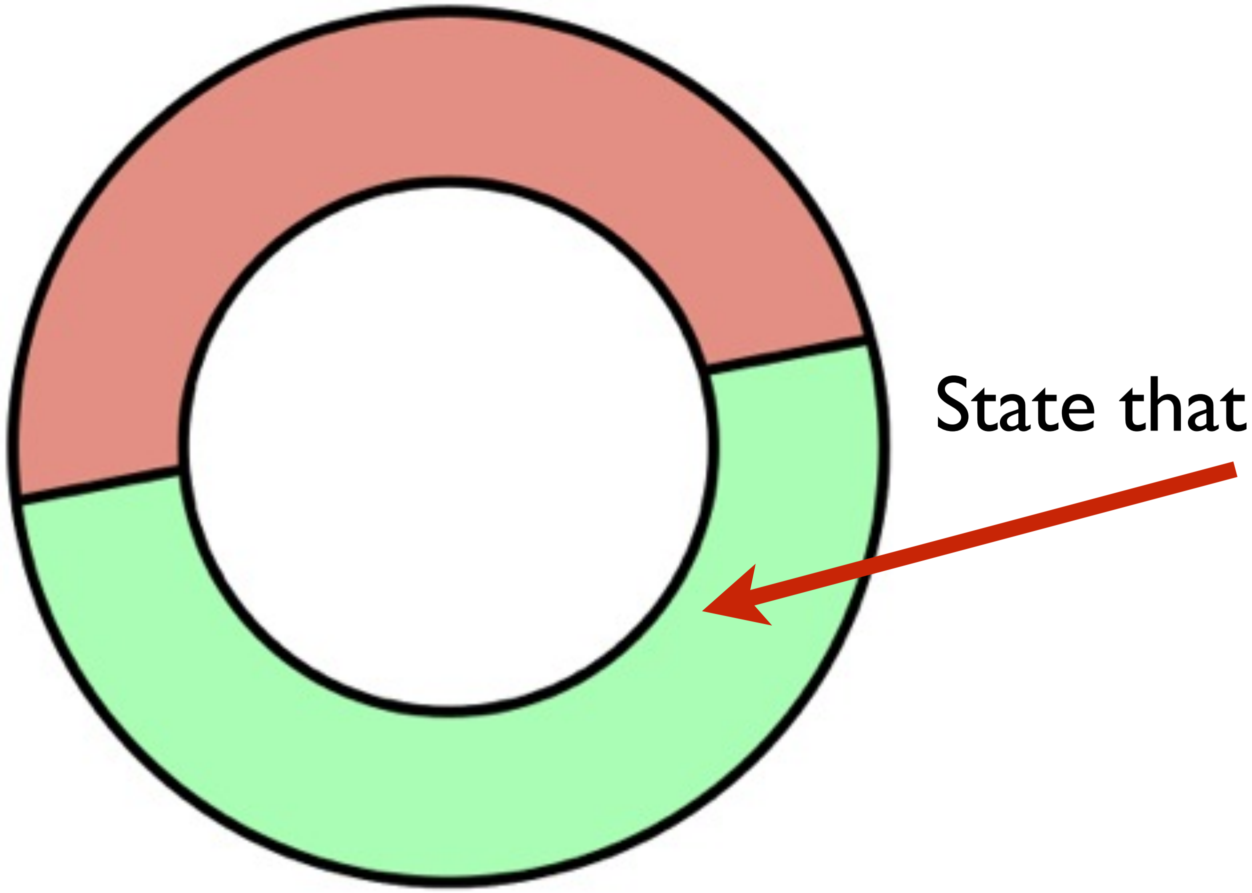
Subjective Auxiliary State

Subjective Concurrent Separation Logic,
LeyWild-Nanevski [POPL'13]



Subjective Auxiliary State

Subjective Concurrent Separation Logic,
LeyWild-Nanevski [POPL'13]

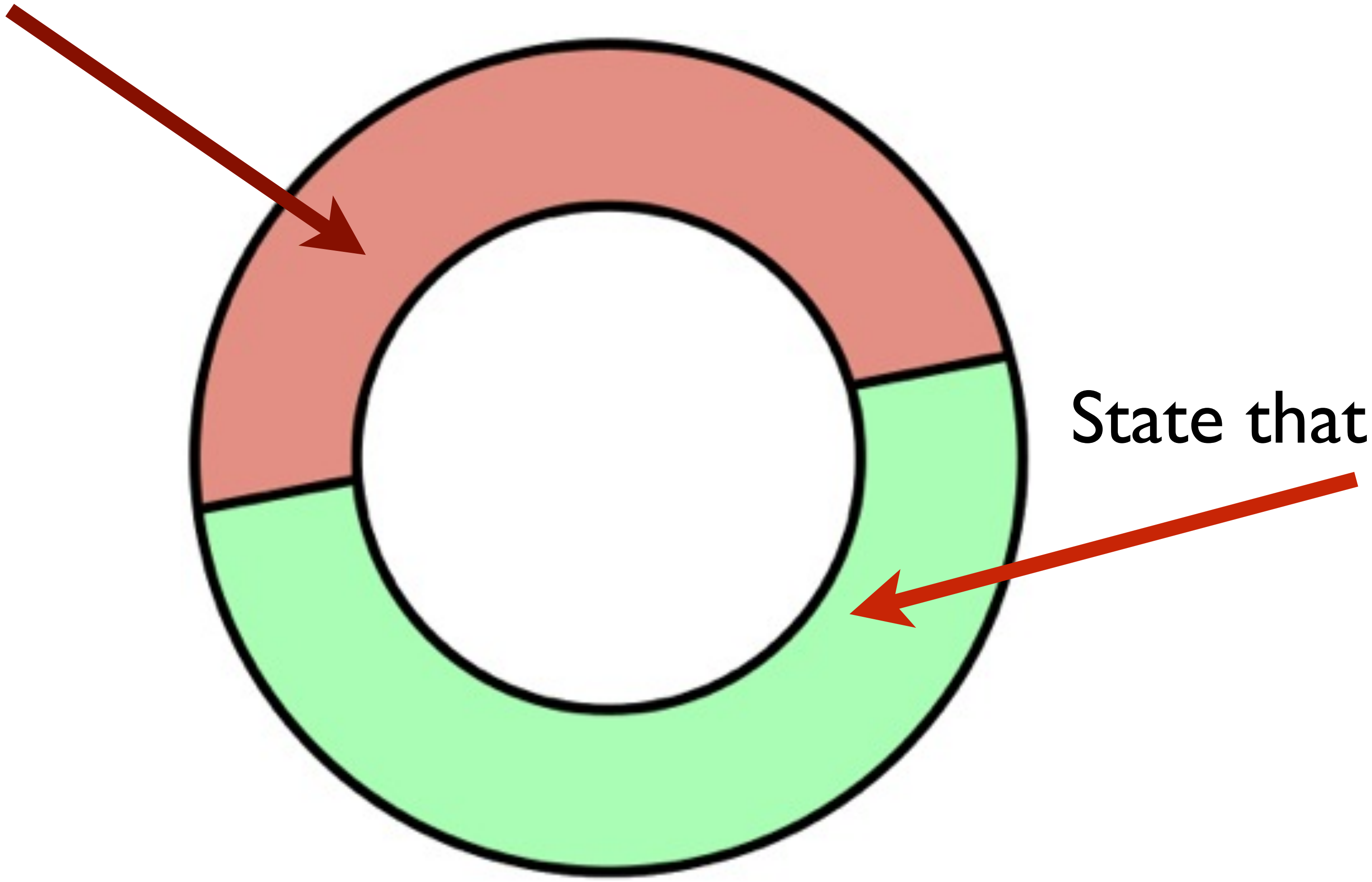


State that belongs to self

Subjective Auxiliary State

Subjective Concurrent Separation Logic,
LeyWild-Nanevski [POPL'13]

State that belongs
to the others

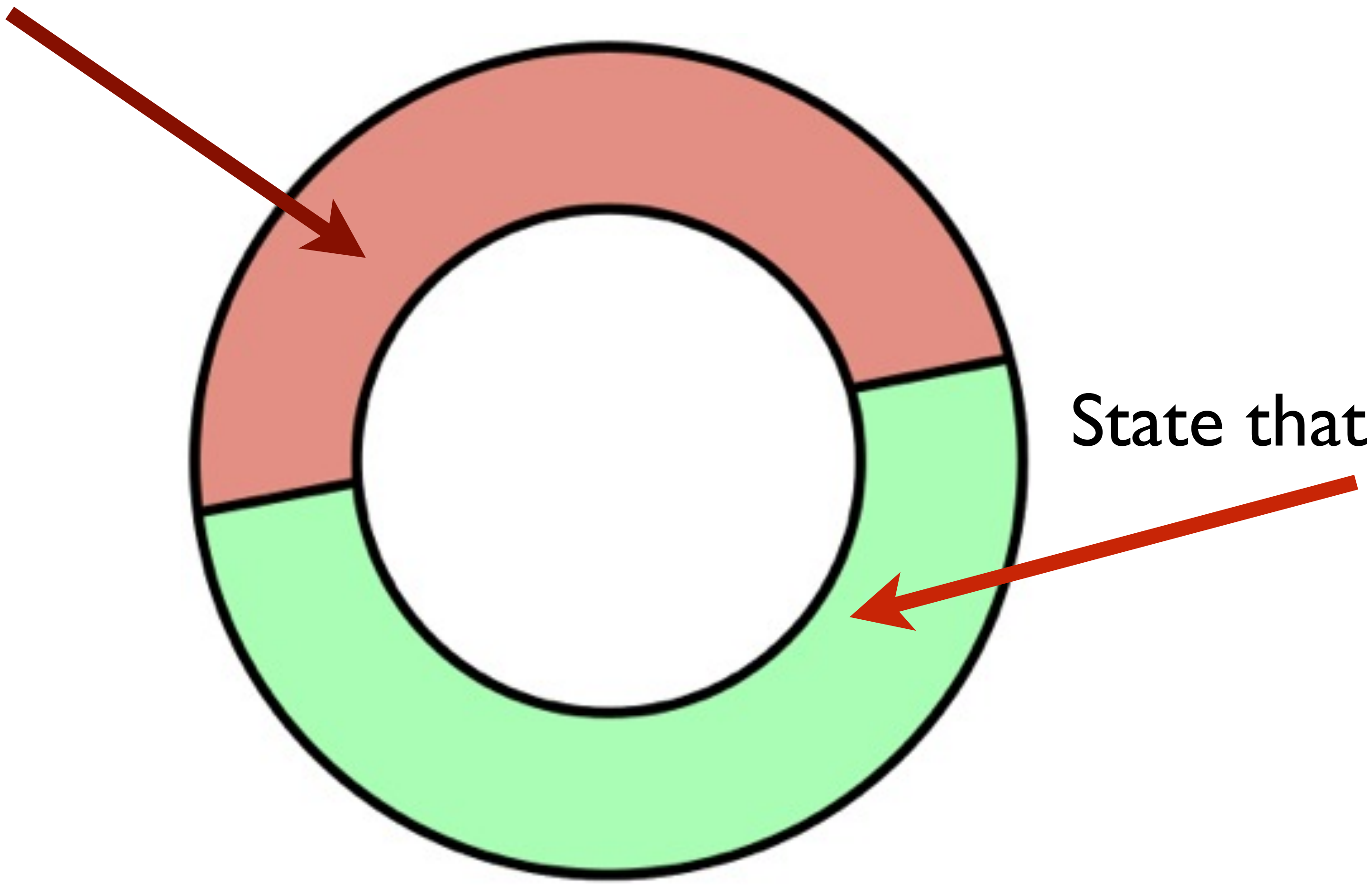


State that belongs to self

Subjective Auxiliary State

Subjective Concurrent Separation Logic,
LeyWild-Nanevski [POPL'13]

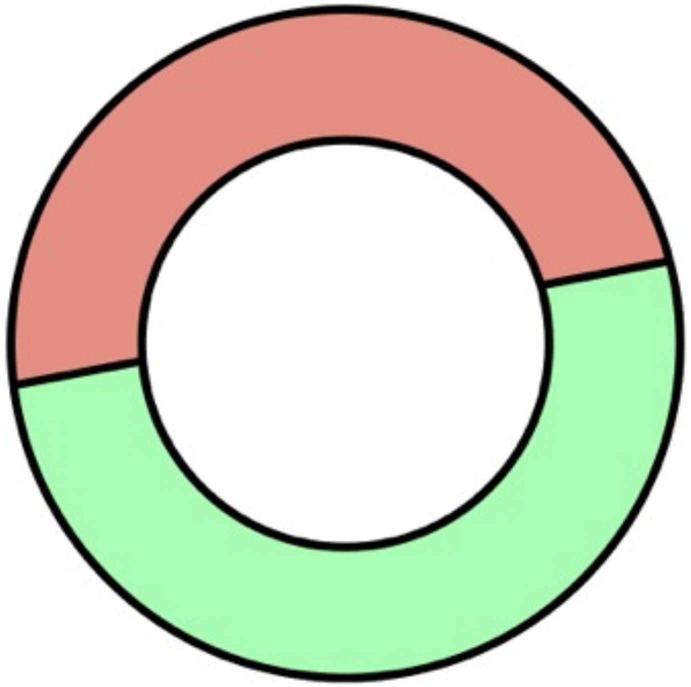
State that belongs
to the others



State that belongs to self

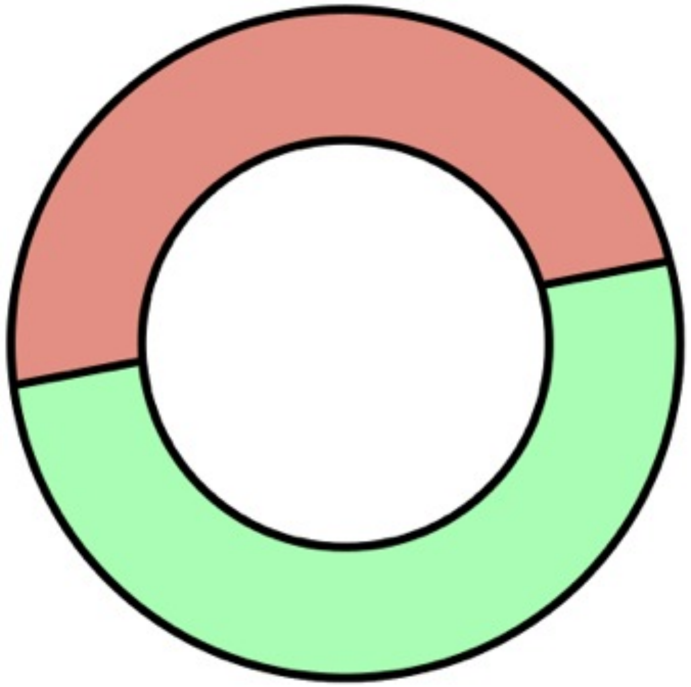
Self and Other states are elements of a *Partial Commutative Monoid* (PCM): $(\mathrm{S}, \mathbf{0}, \oplus)$.

Auxiliary State Split



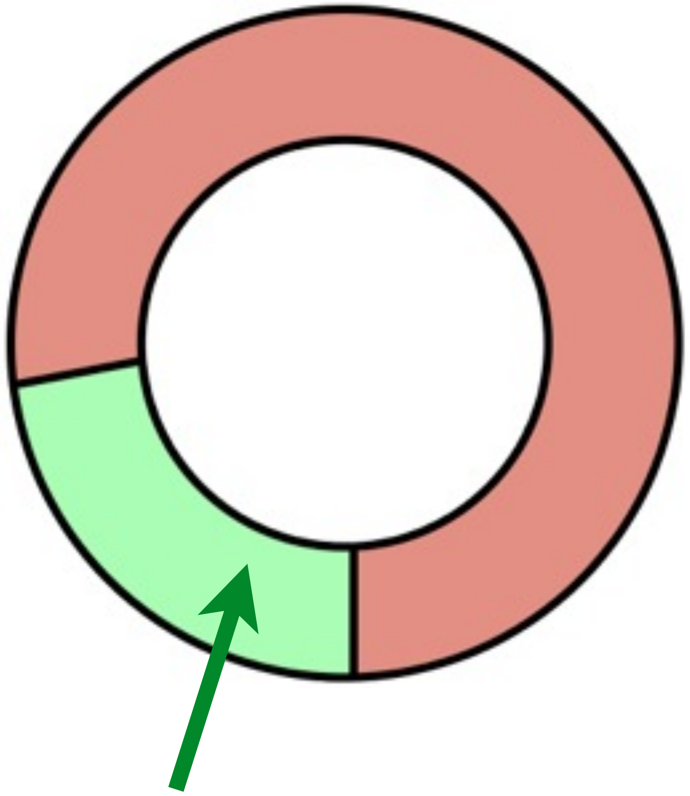
myself

Auxiliary State Split



myself

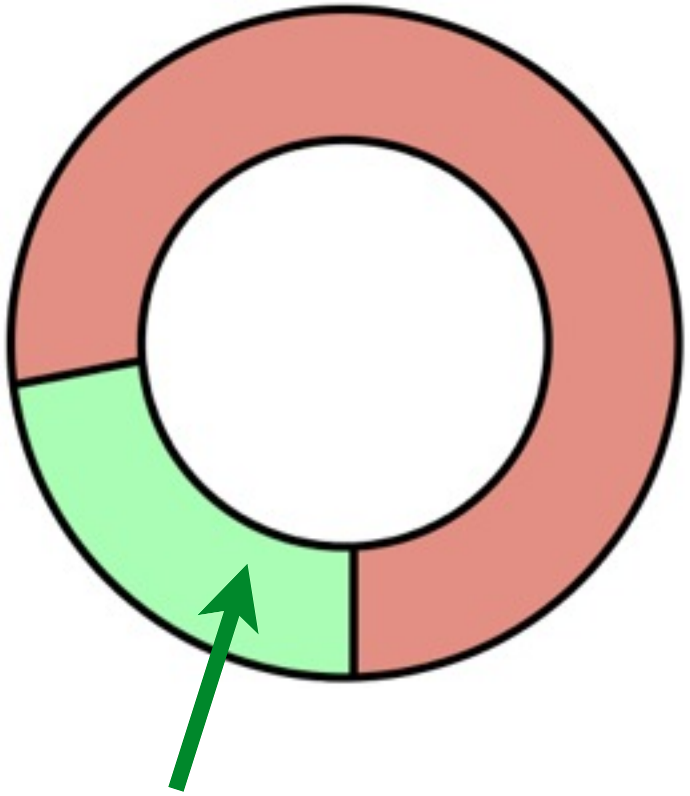
self(1) || self(2)



Ghost state that belongs to self(1)

myself

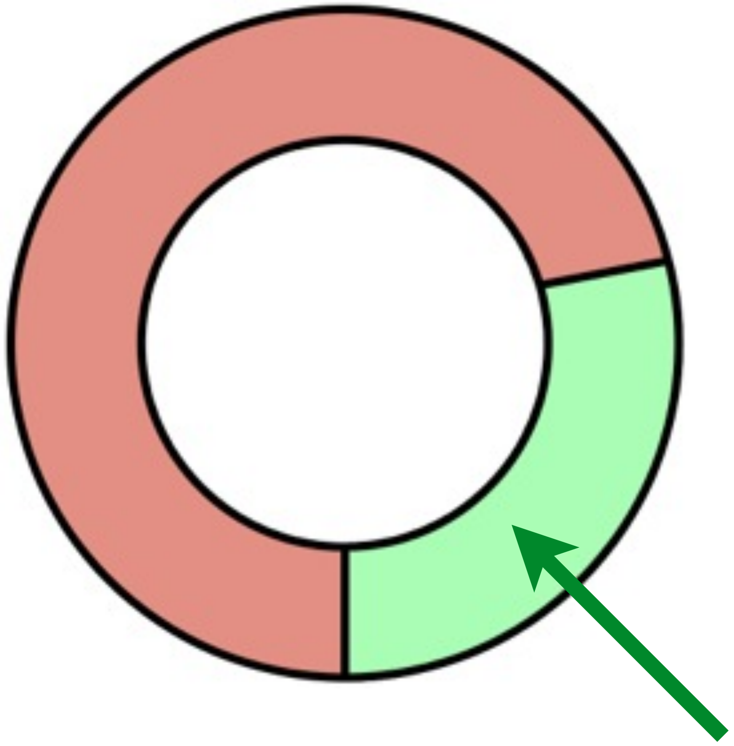
self(1) || self(2)



Ghost state that belongs to self(1)

myself

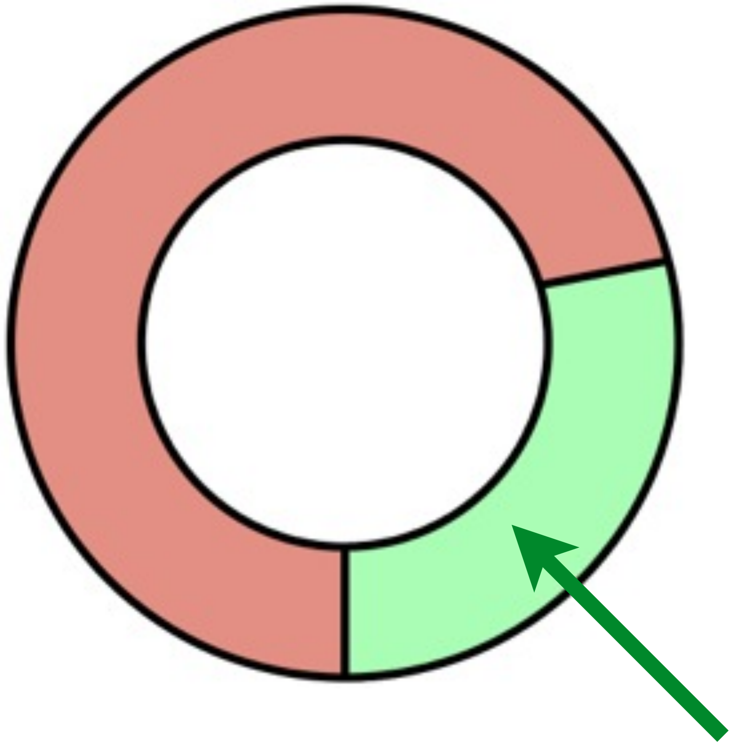
self(1) || self(2)



Ghost state that belongs to self(2)

myself

self(1) || self(2)



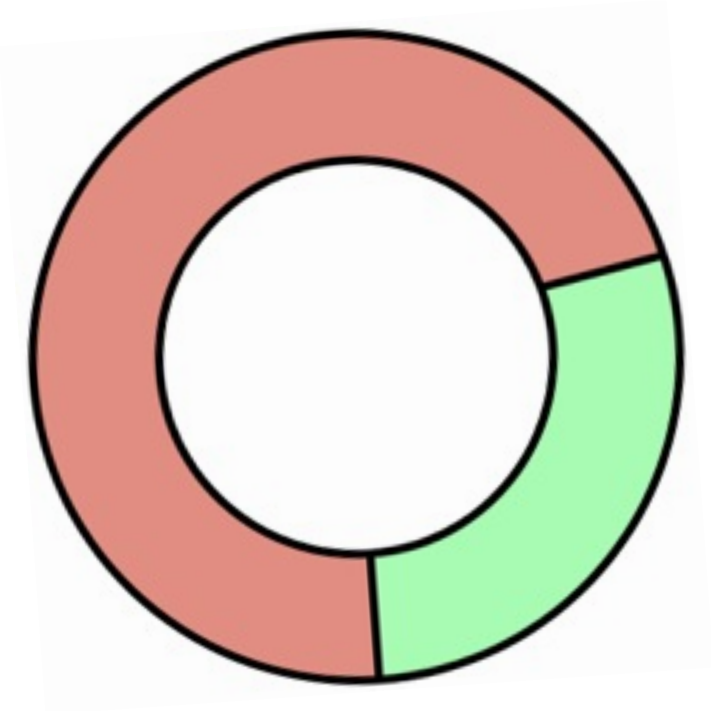
Ghost state that belongs to self(2)

myself

self(1) || self(2)

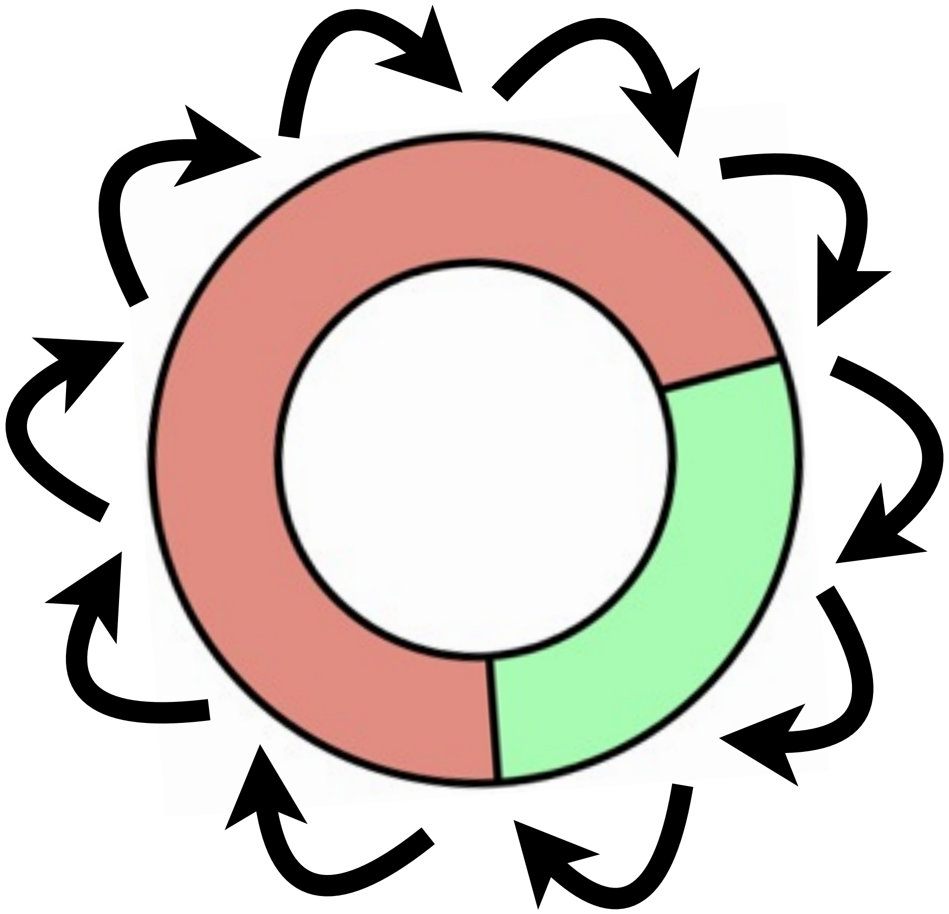
Subjective State for Fine-Grained Concurrency

[This work]



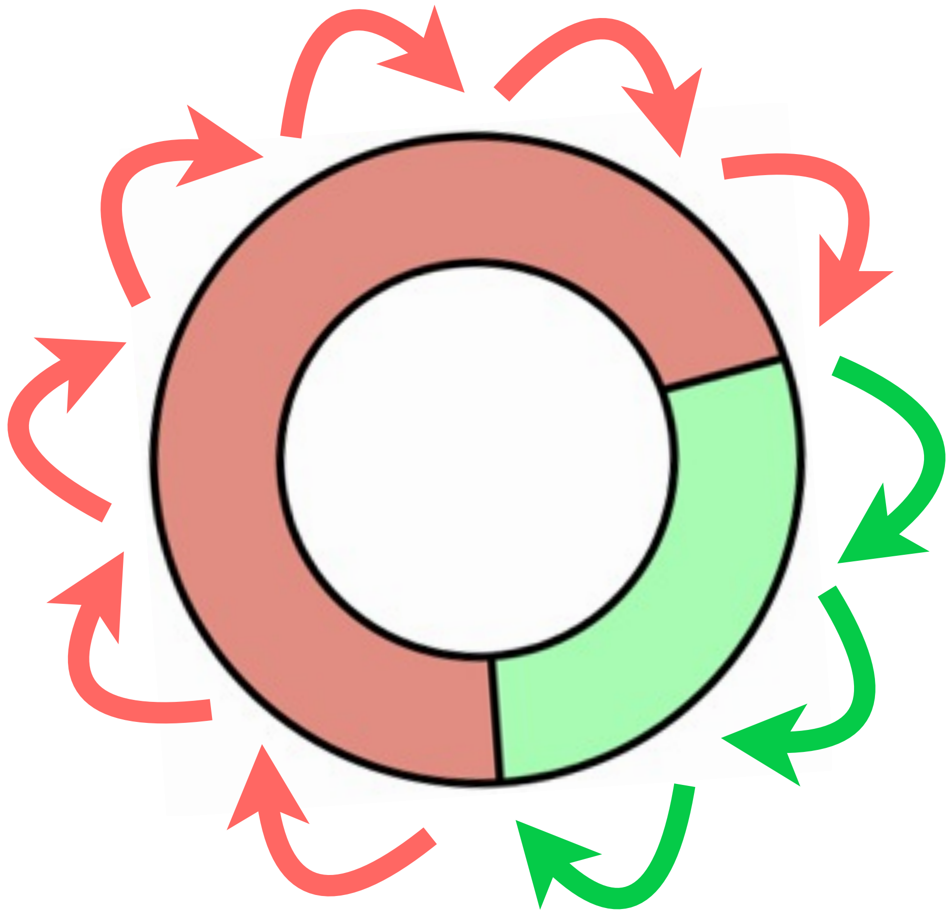
Subjective State for Fine-Grained Concurrency

[This work]



Auxiliary State Split determines Allowed Transitions

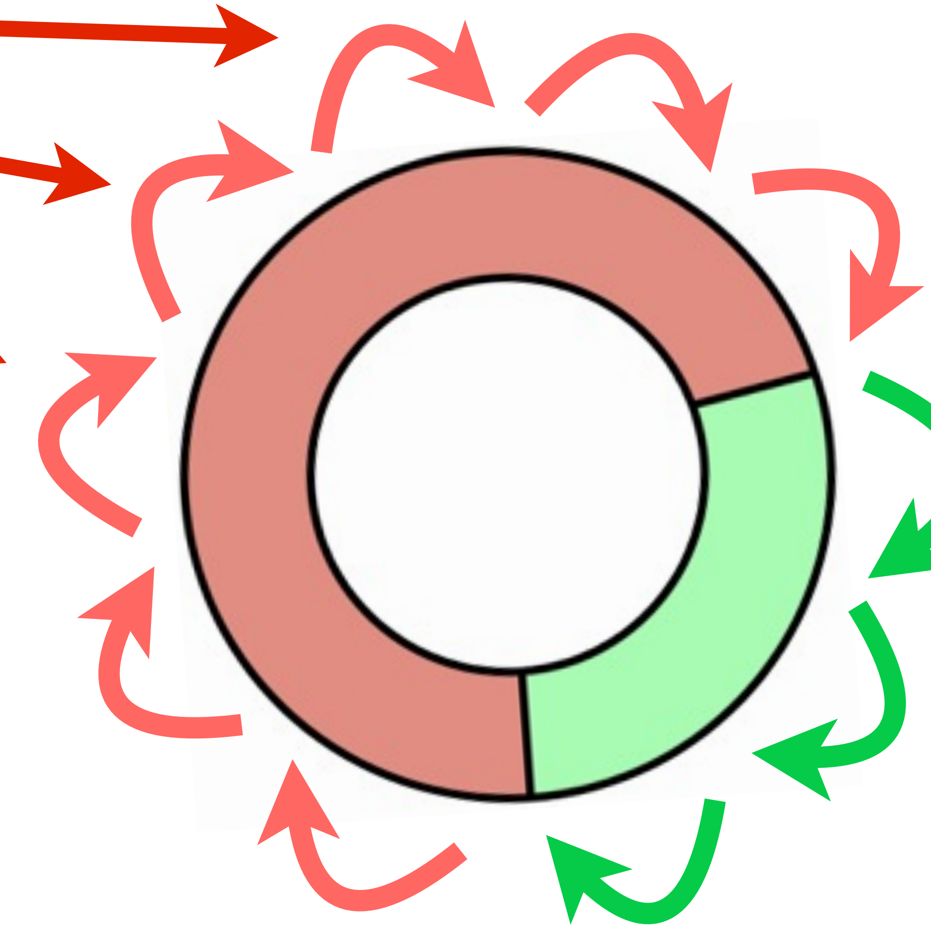
[This work]



Auxiliary State Split determines Allowed Transitions

[This work]

Transitions allowed to the others
(*Rely*)



Transitions allowed to myself
(*Guarantee*)

Subjective specifications

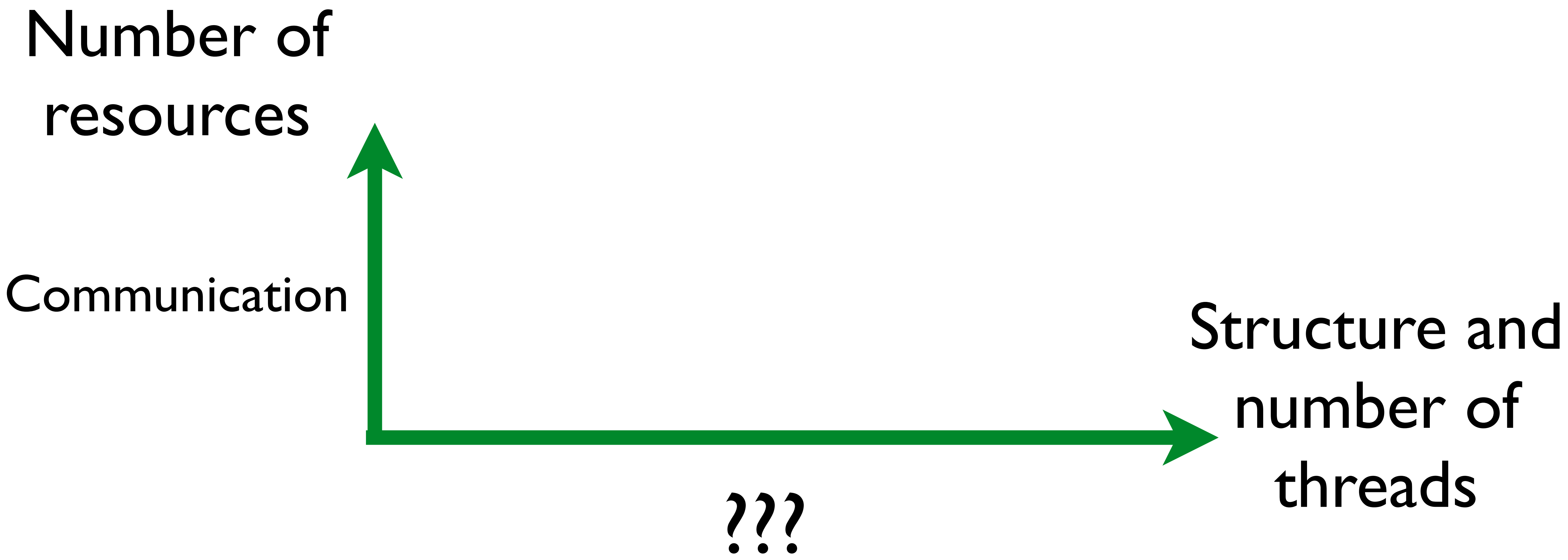
Subjective specifications

Prove for self,
abstract over the others

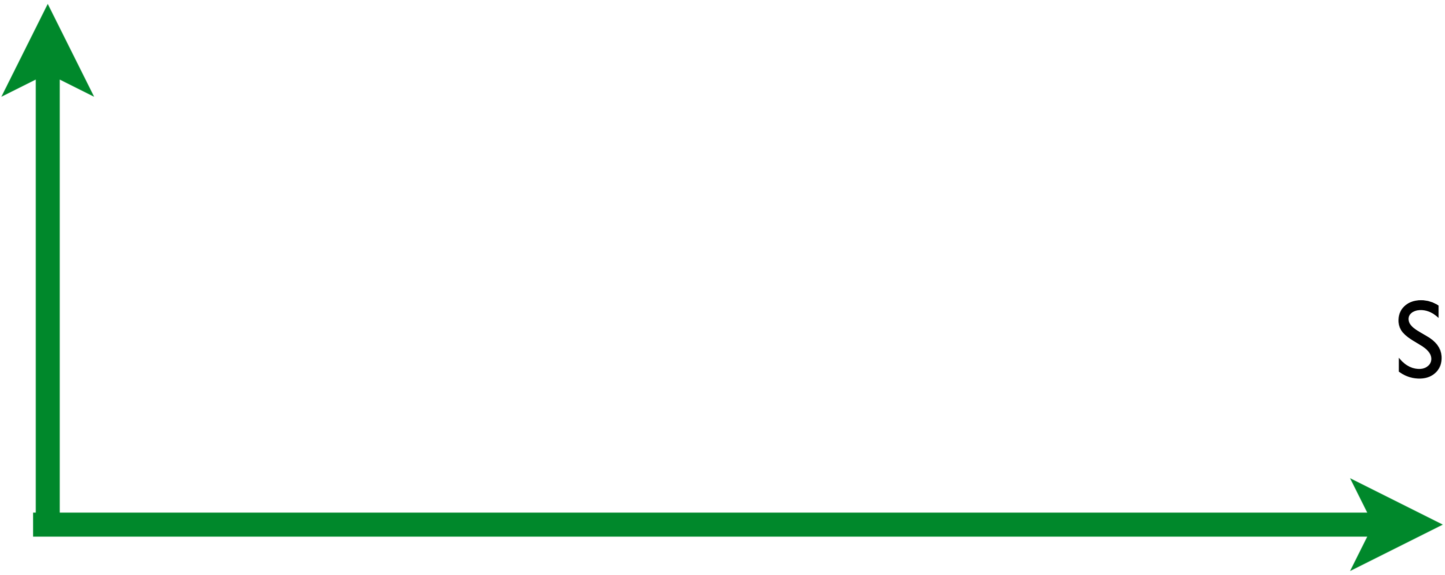
Others

Self

Two dimensions of scalability



Two dimensions of scalability



Number of resources

Communication

Structure and number of threads

Self/Other view on ghost resources

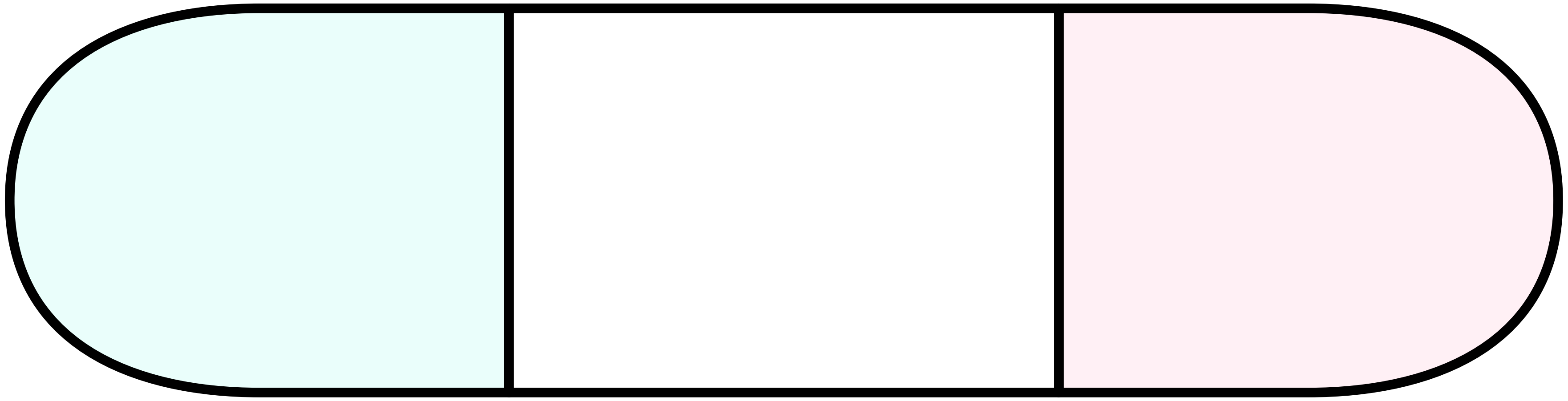
The Model

The Model

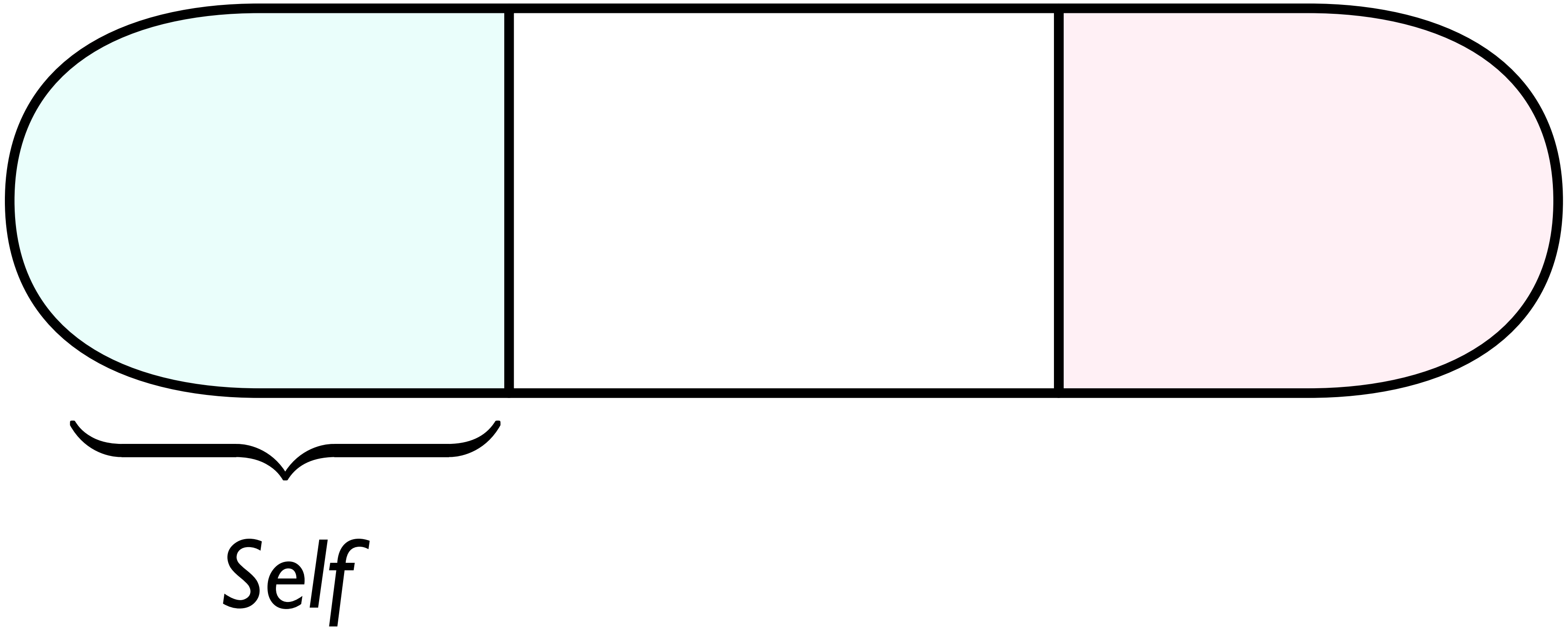
Communicating
Subjective
State-Transition Systems

Concurroids

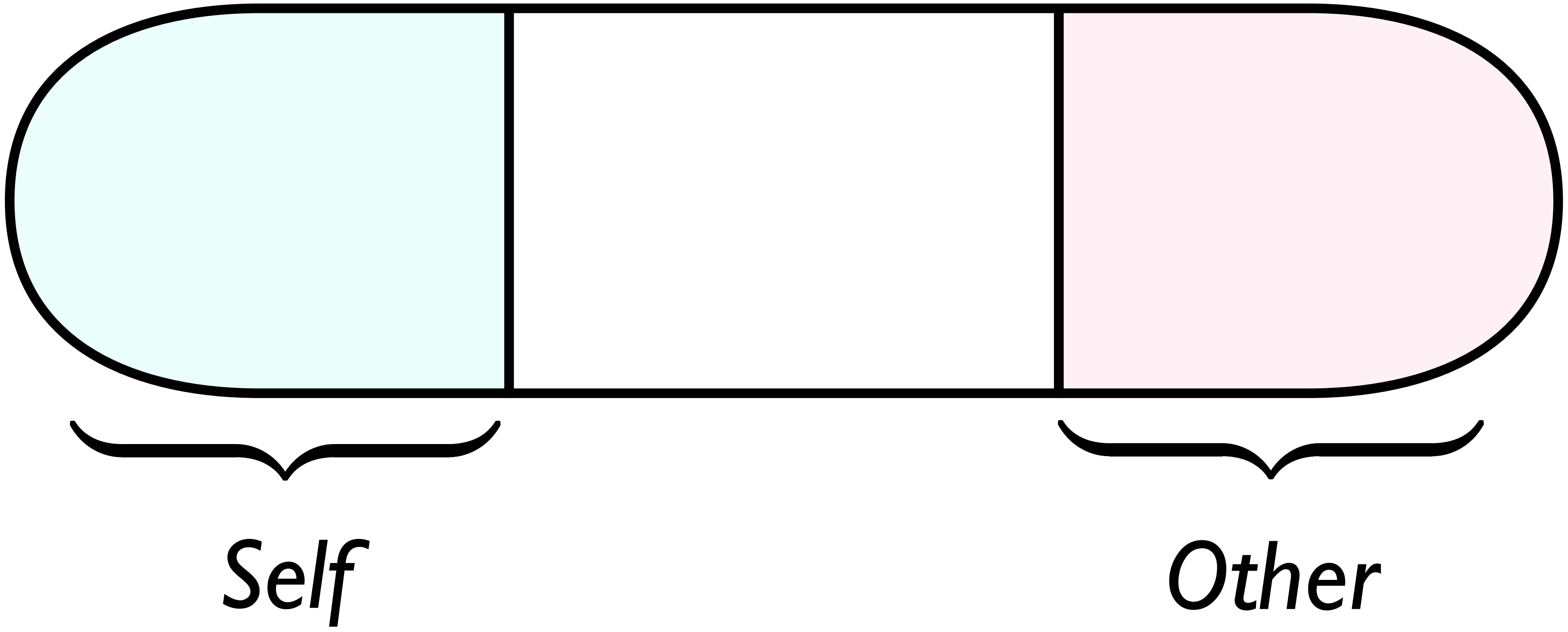
Concurroid States



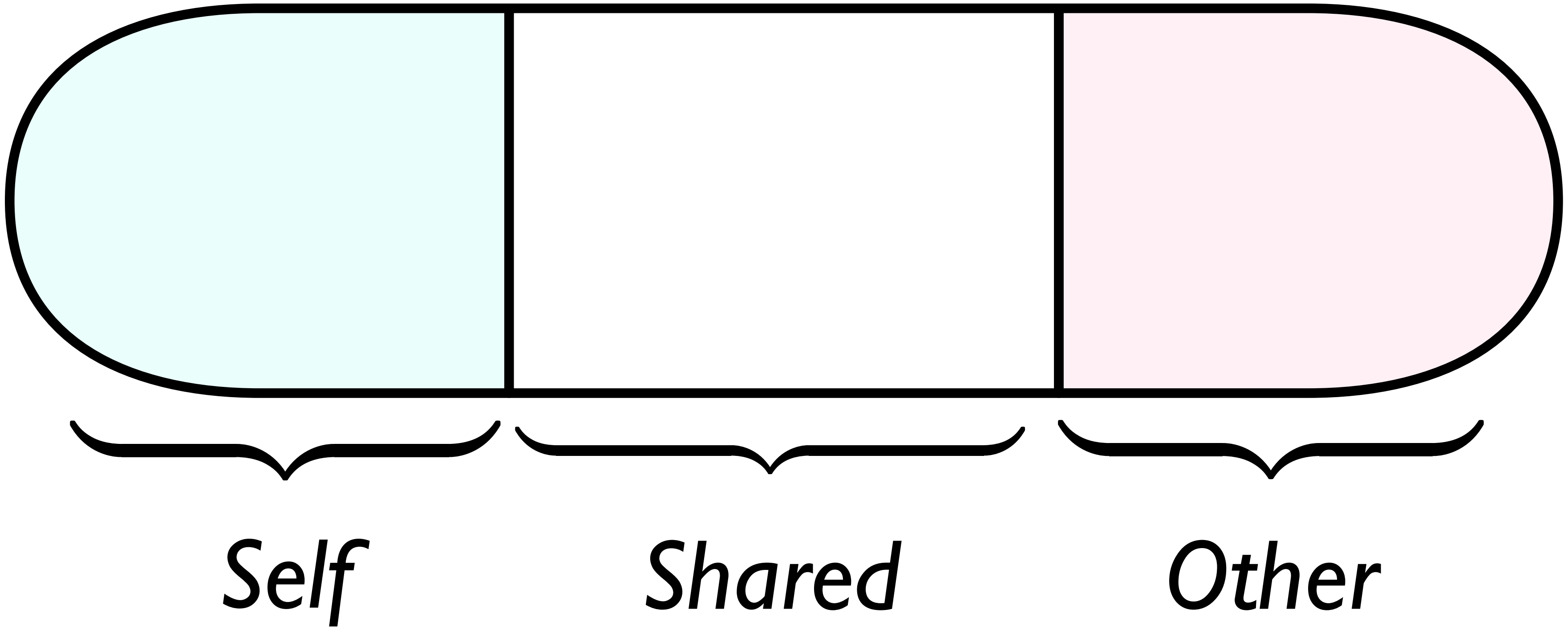
Concurroid States



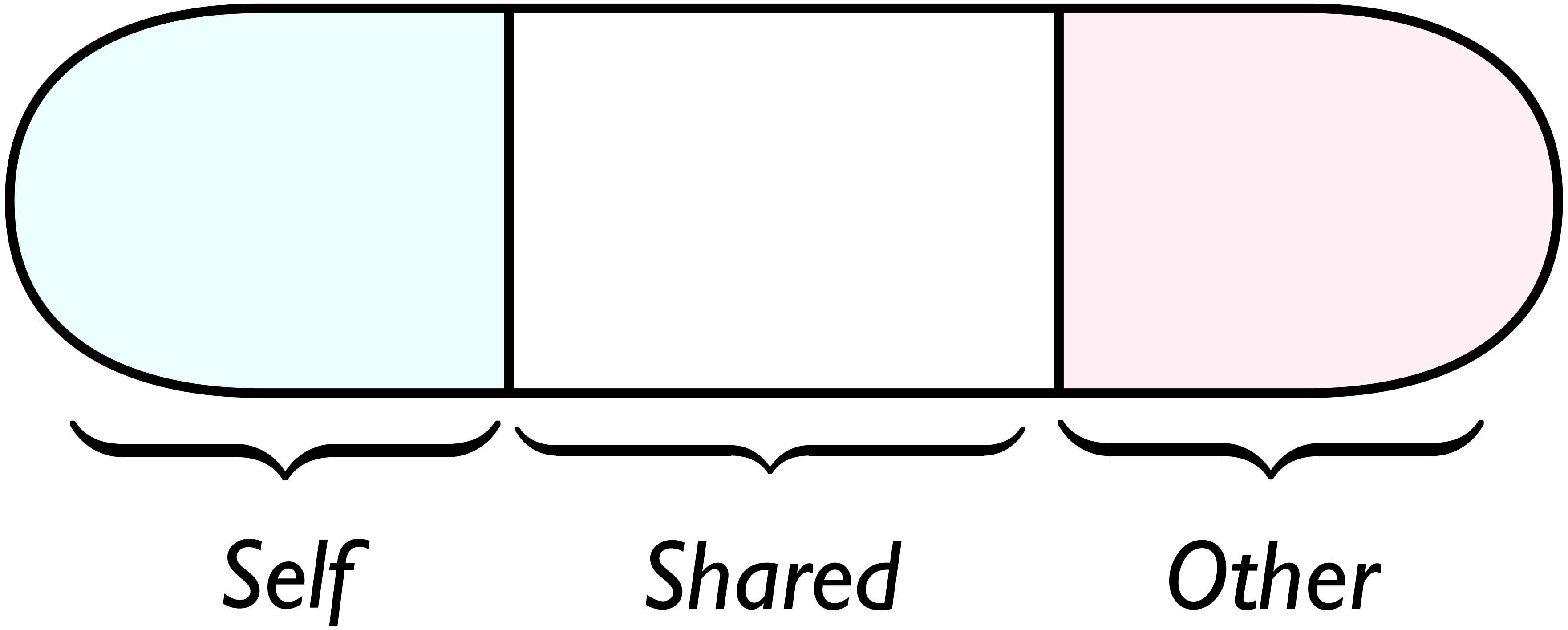
Concurroid States



Concurroid States



Concurroid States



- *Self* - (possibly ghost) state controlled by me;
- *Other* - (possibly ghost) state controlled by all others;
- *Shared* - state that belongs to the resource;
- Self and Other states are elements of a PCM.

Building a concurroid for Ticketed Lock

SERVING No.
27
n_1

SERVING No.
27
n_1

n_2

n_1

n_2

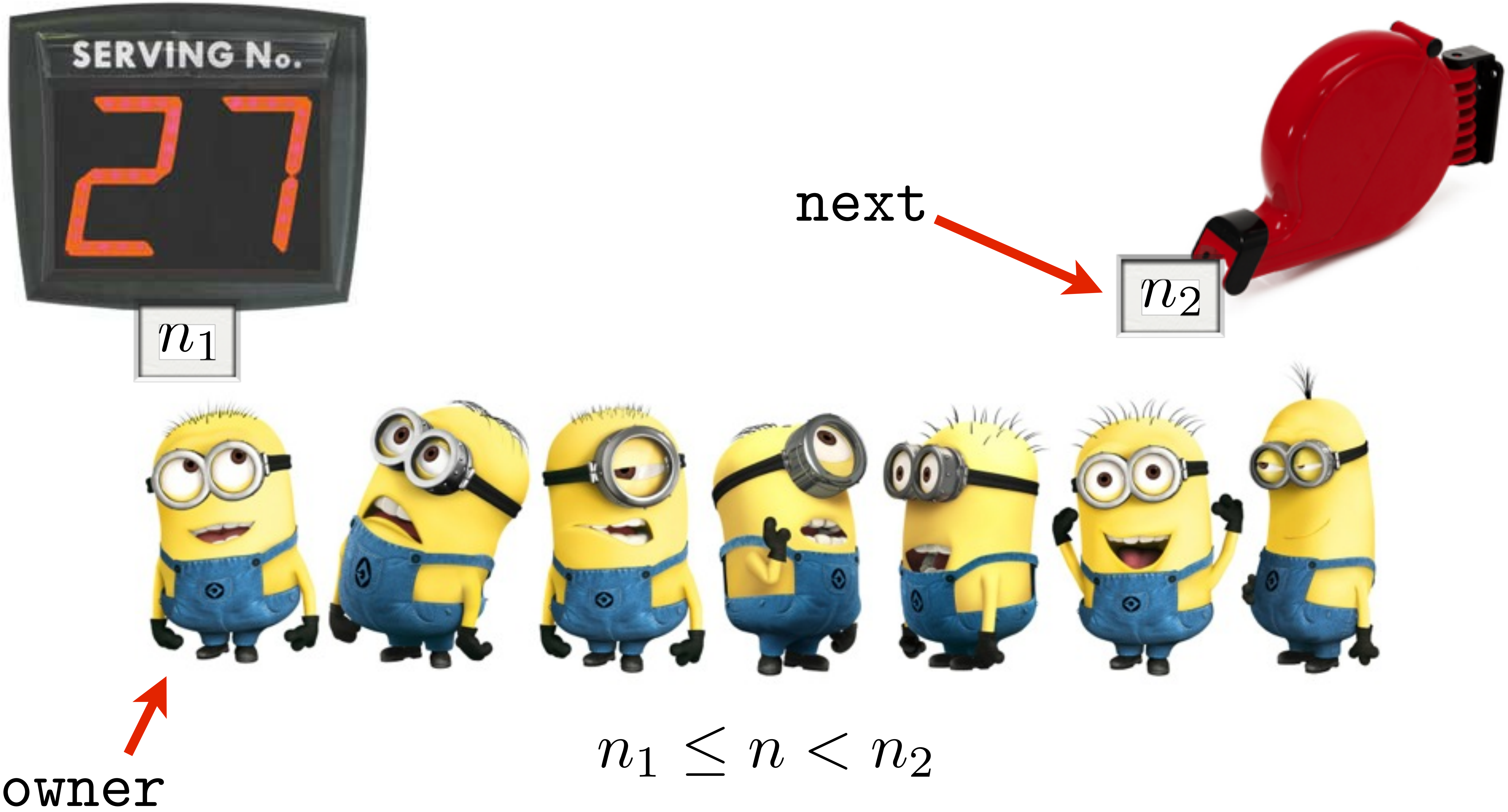
$$n_1 \leq n < n_2$$

n_1

n_2

owner

$$n_1 \leq n < n_2$$



n_1

next

n_2

owner

$n_1 \leq n < n_2$

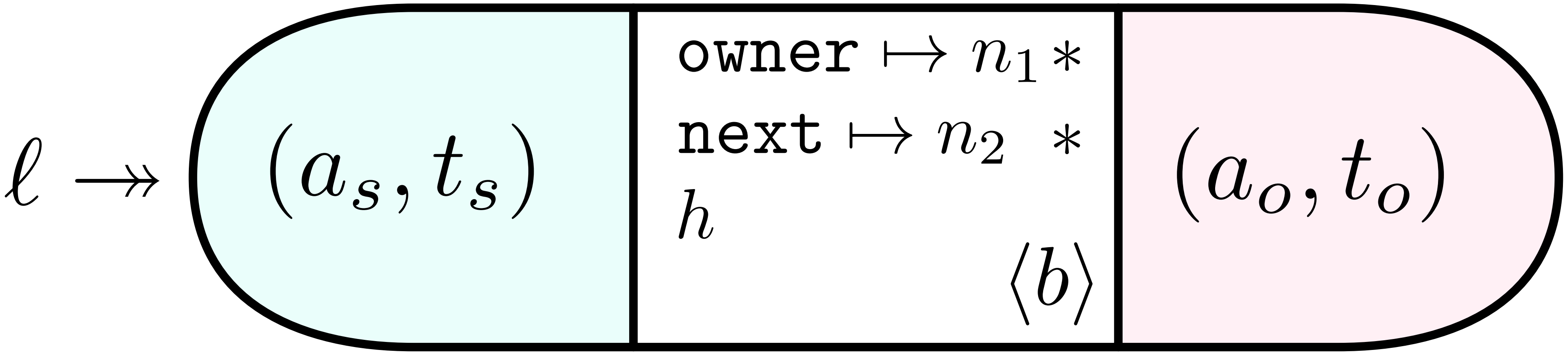
Reference Implementation

```
lock = {
  x := DRAW();
  while (!TRY(x)) SKIP;
}
```

```
unlock = {
  INCR_OWN();
}
```

```
DRAW()      = { return FETCH_AND_INCREMENT(next); }
TRY(n)      = { return (n == owner); }
INCR_OWN() = { owner := owner + 1; }
```

Ticketed Lock States



$\ell \twoheadrightarrow$

(a_s, t_s)

$\texttt{owner} \mapsto n_1 *$

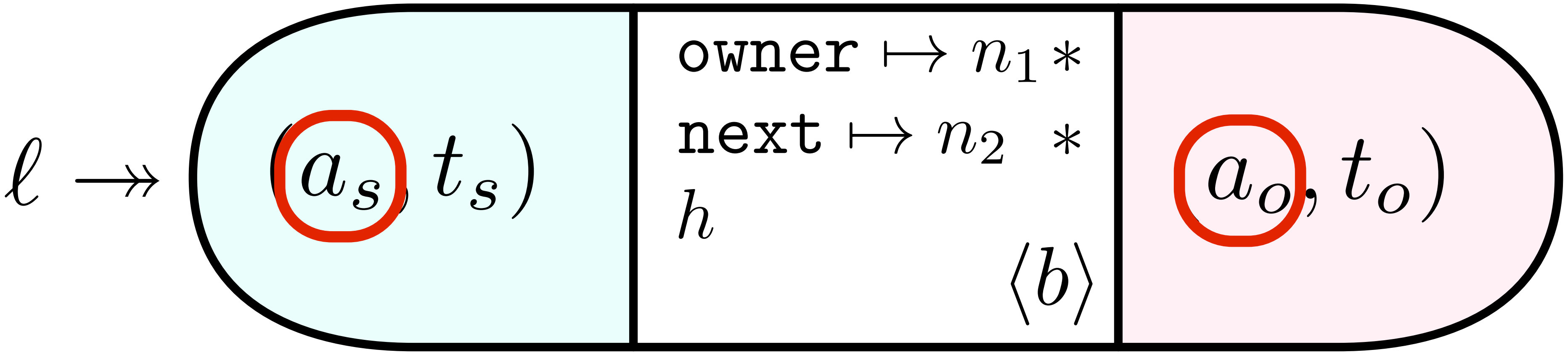
$\texttt{next} \mapsto n_2 \;*$

h

$\langle b \rangle$

(a_o, t_o)

Ticketed Lock States



$\ell \twoheadrightarrow$ (a_s, t_s)

$\texttt{owner} \mapsto n_1 *$

$\texttt{next} \mapsto n_2 \;\; *$

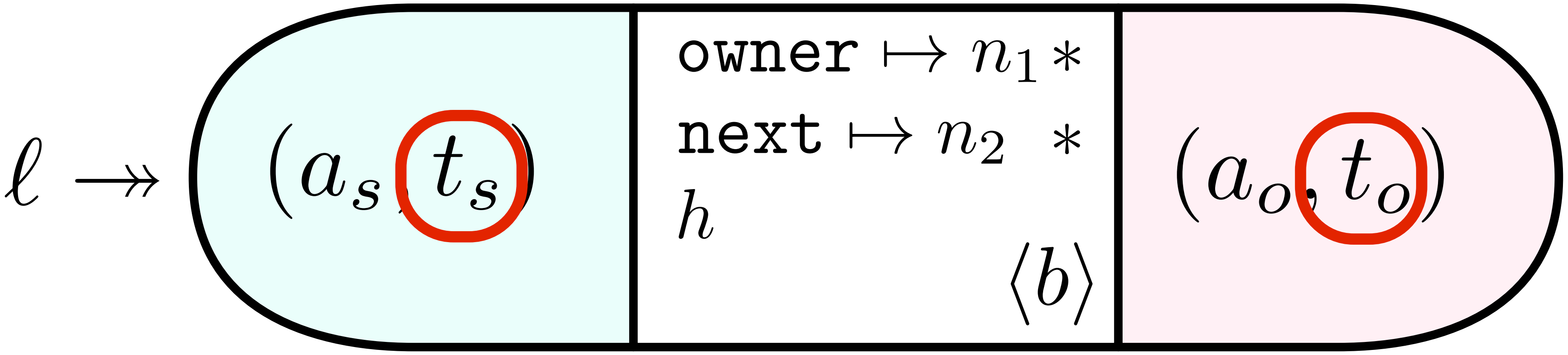
h

$\langle b \rangle$

(a_o, t_o)

- a_s, a_o - parameter ghost state controlled by <u>*self*</u>/<u>*other*;</u>

Ticketed Lock States



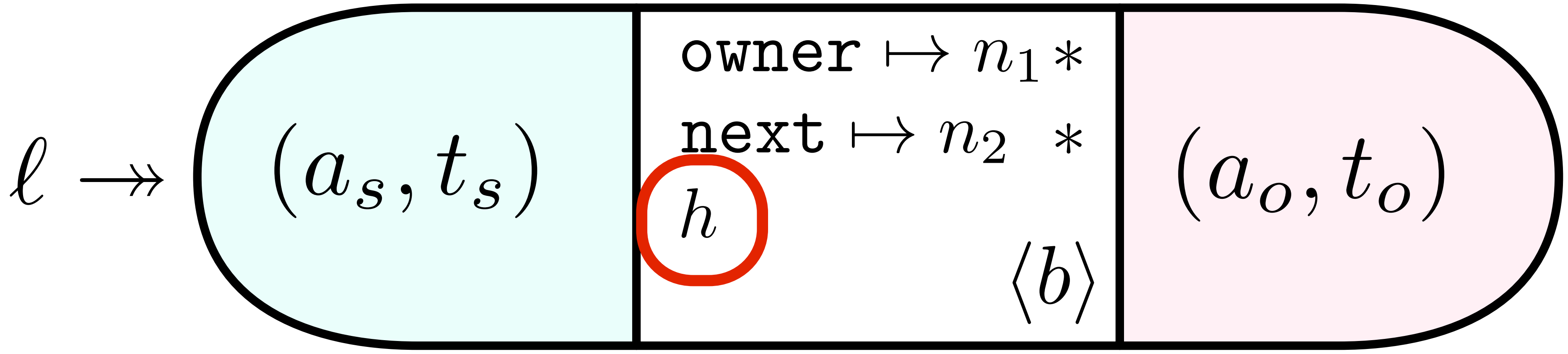
$\ell \twoheadrightarrow$ (a_s, t_s)

$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 *$
h
$\langle b \rangle$

(a_o, t_o)

- a_s, a_o - parameter ghost state controlled by *self*/*other*;
- t_s, t_o - tickets, owned by *self*/*other*;

Ticketed Lock States



$\ell \twoheadrightarrow (a_s, t_s)$

$\mathtt{owner} \mapsto n_1 *$

$\mathtt{next} \mapsto n_2 *$

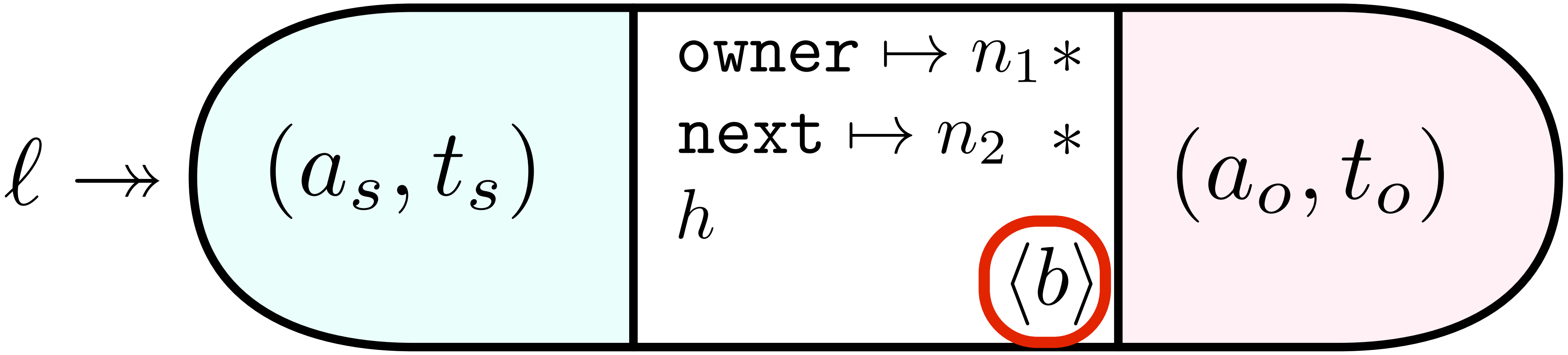
h

$\langle b \rangle$

(a_o, t_o)

- a_s, a_o - parameter ghost state controlled by *self*/*other*;
- t_s, t_o - tickets, owned by *self*/*other*;
- h - a heap protected by the lock, subject of ownership transfer;

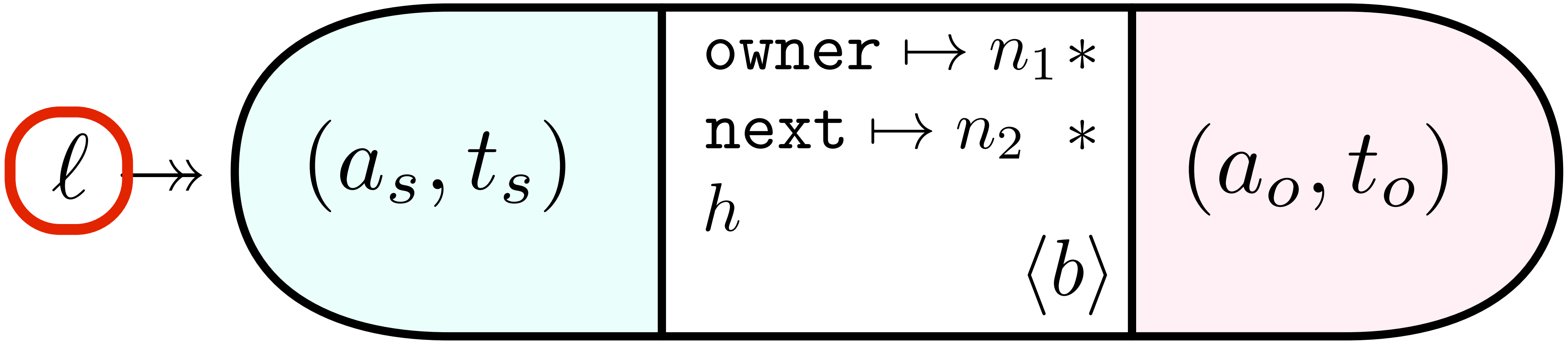
Ticketed Lock States



$\ell \twoheadrightarrow$ (a_s, t_s) $\mathtt{owner} \mapsto n_1 *$ $\mathtt{next} \mapsto n_2\ *$ h $\langle b \rangle$ (a_o, t_o)

- a_s, a_o - parameter ghost state controlled by *self*/*other*;
- t_s, t_o - tickets, owned by *self*/*other*;
- h - a heap protected by the lock, subject of ownership transfer;
- b - administrative flag to indicate locking;

Ticketed Lock States



$\ell \twoheadrightarrow$ (a_s, t_s)

$\mathtt{owner} \mapsto n_1 *$

$\mathtt{next} \mapsto n_2 \; *$

h

$\langle b \rangle$

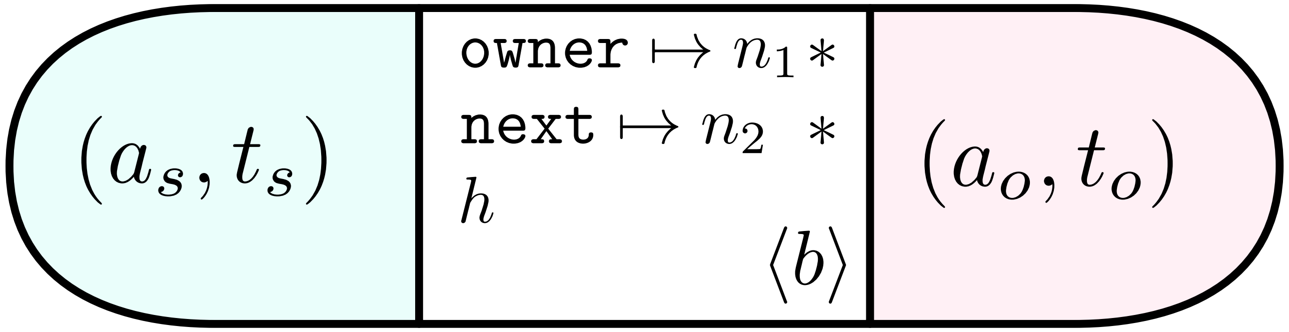
(a_o, t_o)

- a_s, a_o - parameter ghost state controlled by *self*/*other*;
- t_s, t_o - tickets, owned by *self*/*other*;
- h - a heap protected by the lock, subject of ownership transfer;
- b - administrative flag to indicate locking;
- ℓ - label to identify *this* particular instance of TLock concurroid.

Ticketed Lock Invariant

Ticketed Lock Invariant

$s = \ell \twoheadrightarrow$



(a_s, t_s)

$\texttt{owner} \mapsto n_1 *$

$\texttt{next} \mapsto n_2 \; *$

h

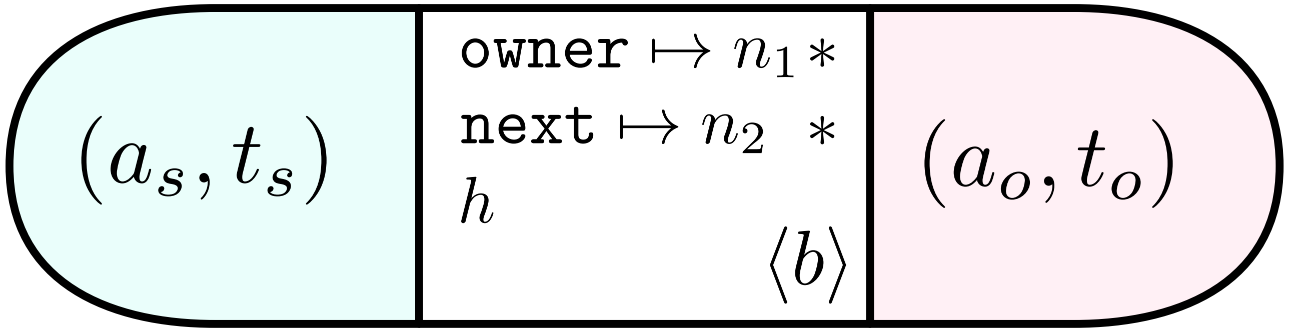
$\langle b \rangle$

(a_o, t_o)

$\wedge$

Ticketed Lock Invariant

$s = \ \ell \twoheadrightarrow$



(a_s, t_s)

$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 \ *$
h
$\langle b \rangle$

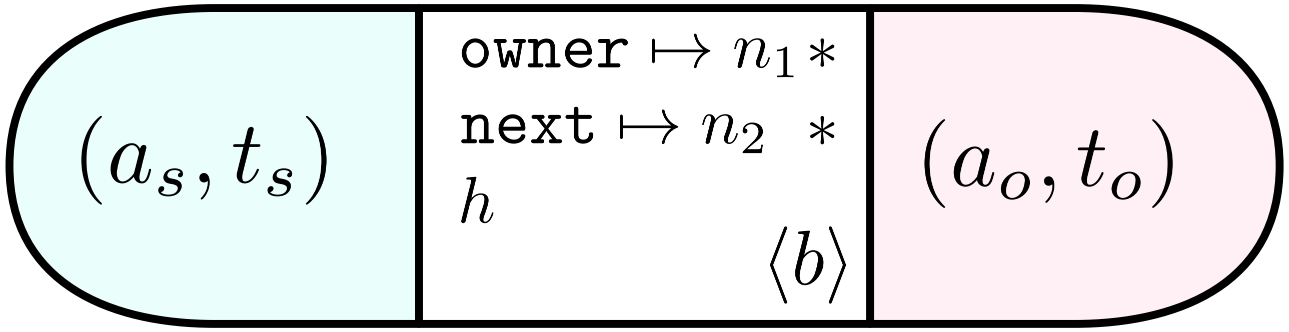
(a_o, t_o)

$\wedge$

$$\boxed{t_s \oplus t_o = \{n \mid n_1 \leq n < n_2\}} \ \wedge$$

All dispensed tickets

Ticketed Lock Invariant

$s = \ \ell \twoheadrightarrow$  $\wedge$

(a_s, t_s)

$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 \ *$
h
$\langle b \rangle$

(a_o, t_o)

All dispensed tickets

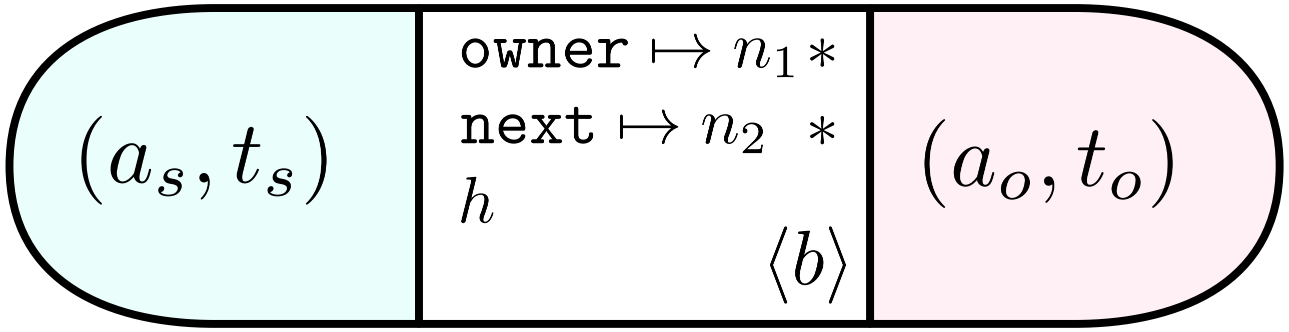
$$\boxed{t_s \oplus t_o = \{n \mid n_1 \leq n < n_2\}} \ \wedge$$

Locked

$$\left(\boxed{(n_1 \in (t_s \oplus t_o) \ \wedge \ b = \mathbf{true} \ \wedge \ h = \mathsf{emp})} \ \vee \right)$$

Ticketed Lock Invariant

$s = \ \ell \twoheadrightarrow$



(a_s, t_s)

$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 \ *$
h
$\langle b \rangle$

(a_o, t_o)

$\wedge$

All dispensed tickets

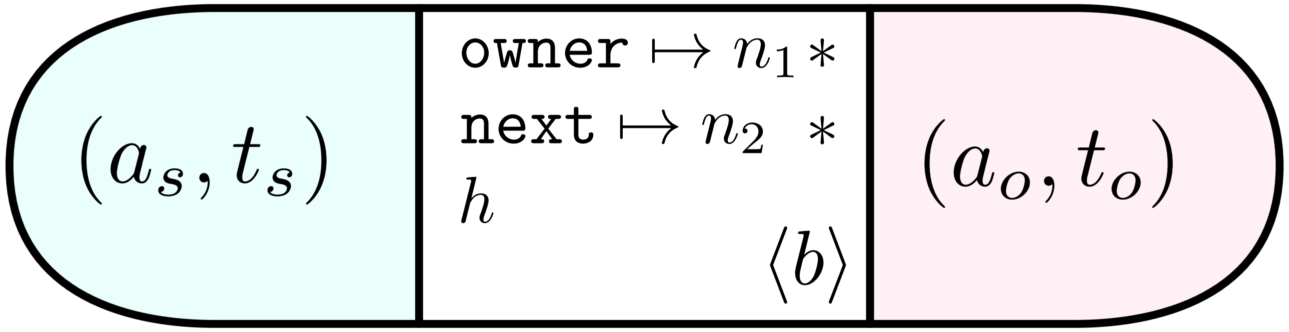
$$\boxed{t_s \oplus t_o = \{n \mid n_1 \leq n < n_2\}} \ \wedge$$

Locked

$$\left(\begin{array}{l} \boxed{(n_1 \in (t_s \oplus t_o) \ \wedge \ b = \textbf{true} \ \wedge \ h = \mathsf{emp})} \ \vee \\ \\ \textbf{if } n_1 < n2 \quad \textbf{then} \quad n_1 \in (t_s \oplus t_o) \ \wedge \ b = \textbf{false} \ \wedge \ I(a_s \oplus a_o)h \\ \\ \qquad\qquad\quad \textbf{else} \quad \boxed{n_1 = n_2 \ \wedge \ b = \textbf{false} \ \wedge \ I(a_s \oplus a_o)h} \end{array} \right)$$

Unlocked

Ticketed Lock Invariant

$s = \ell \twoheadrightarrow$  $\wedge$

(a_s, t_s)

$\texttt{owner} \mapsto n_1 *$

$\texttt{next} \mapsto n_2 \; *$

h

$\langle b \rangle$

(a_o, t_o)

All dispensed tickets

$$\boxed{t_s \oplus t_o = \{n \mid n_1 \leq n < n_2\}} \wedge$$

Locked

$$\left(\begin{array}{lll} \multicolumn{3}{l}{\boxed{(n_1 \in (t_s \oplus t_o) \;\wedge\; b = \mathbf{true} \;\wedge\; h = \mathsf{emp})} \vee} \\ & & \text{About to be served} \\ \mathbf{if}\ n_1 < n2 & \mathbf{then} & \boxed{n_1 \in (t_s \oplus t_o) \;\wedge\; b = \mathbf{false} \;\wedge\; I(a_s \oplus a_o)h} \\ & \mathbf{else} & \boxed{n_1 = n_2 \;\wedge\; b = \mathbf{false} \;\wedge\; I(a_s \oplus a_o)h} \end{array}\right)$$

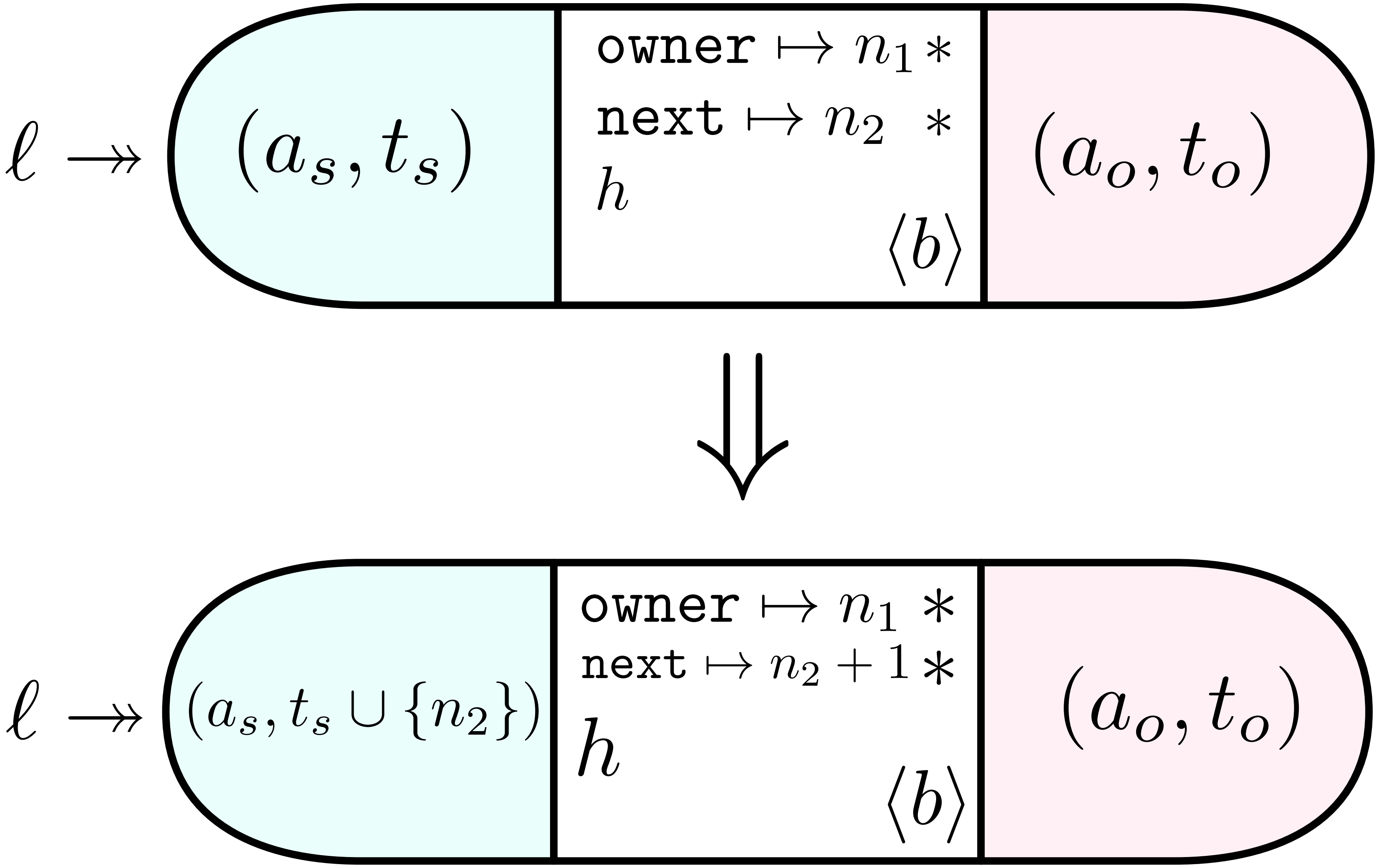
Unlocked

Transitions

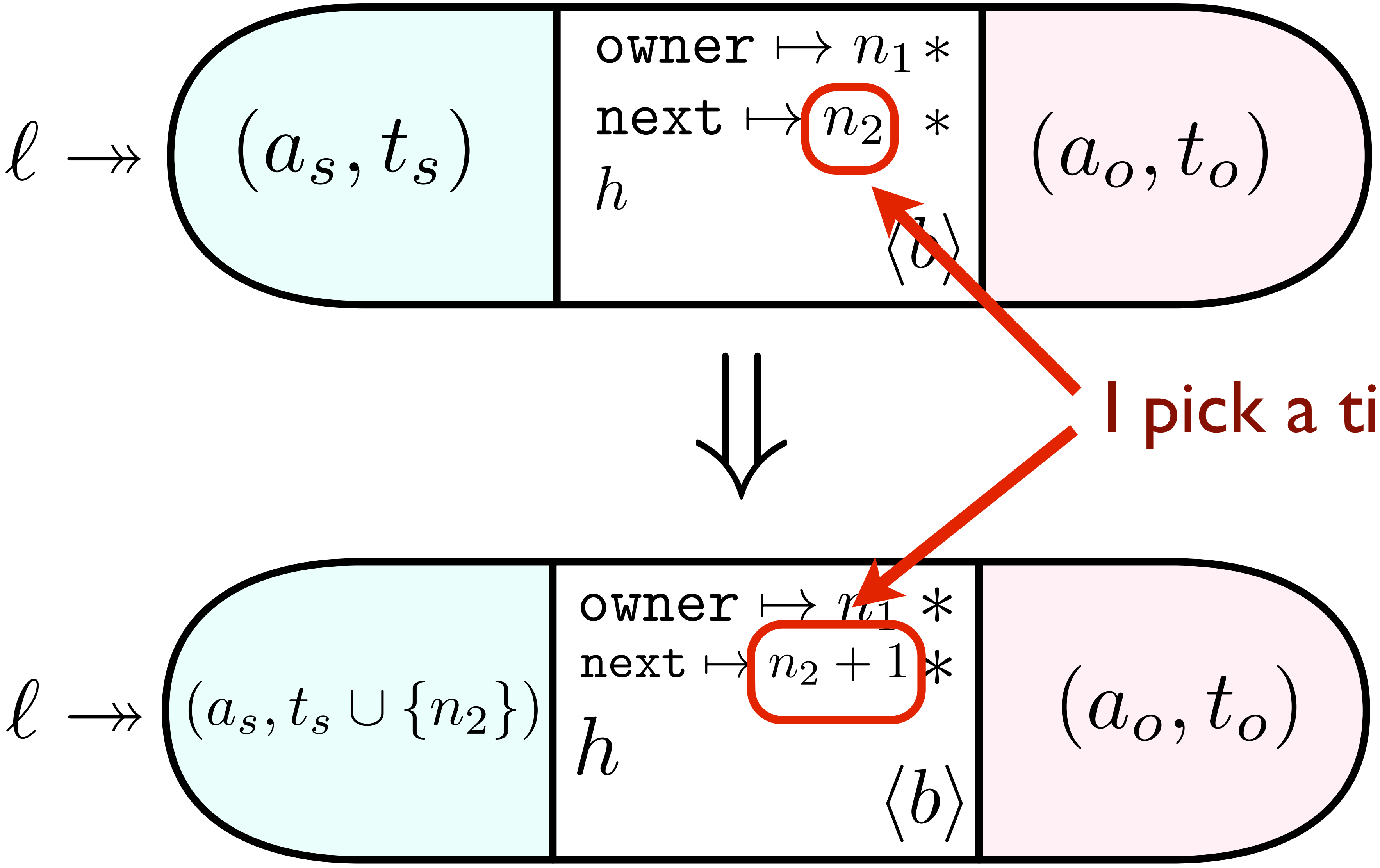
Internal Transitions

Intuition:

drawing a ticket from the dispenser



$\ell \twoheadrightarrow$
(a_s, t_s)
$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 *$
h
$\langle b \rangle$
(a_o, t_o)
$\Downarrow$
$\ell \twoheadrightarrow$
$(a_s, t_s \cup \{n_2\})$
$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 + 1 *$
h
$\langle b \rangle$
(a_o, t_o)

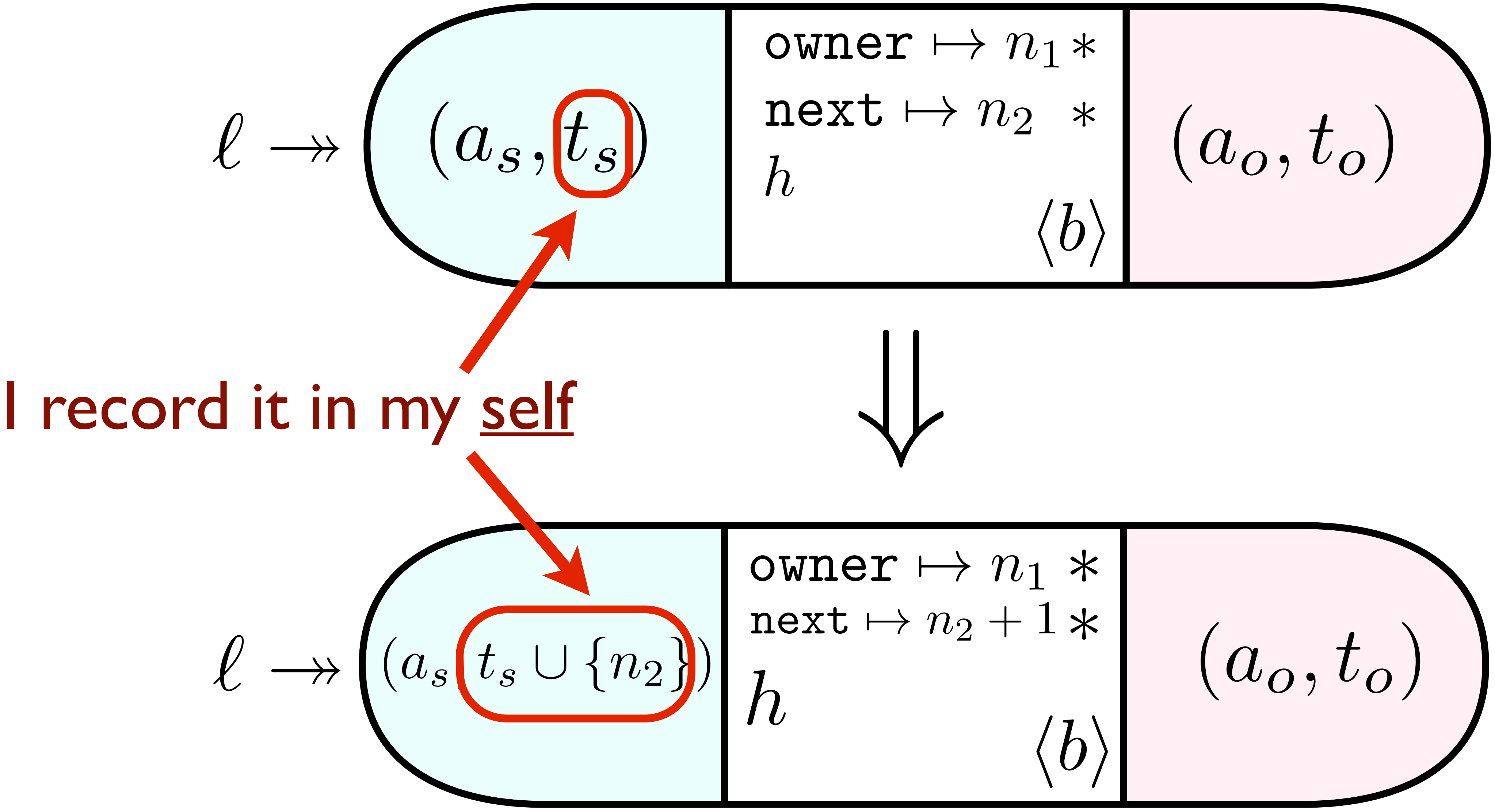


$\ell \twoheadrightarrow$ (a_s, t_s) | `owner` $\mapsto n_1 *$ `next` $\mapsto \boxed{n_2} *$ h $\langle b \rangle$ | (a_o, t_o)

$$\Downarrow$$

$\ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_2\})$ | `owner` $\mapsto n_1 *$ `next` $\mapsto \boxed{n_2 + 1} *$ h $\langle b \rangle$ | (a_o, t_o)

I pick a ticket



Communication

Communication

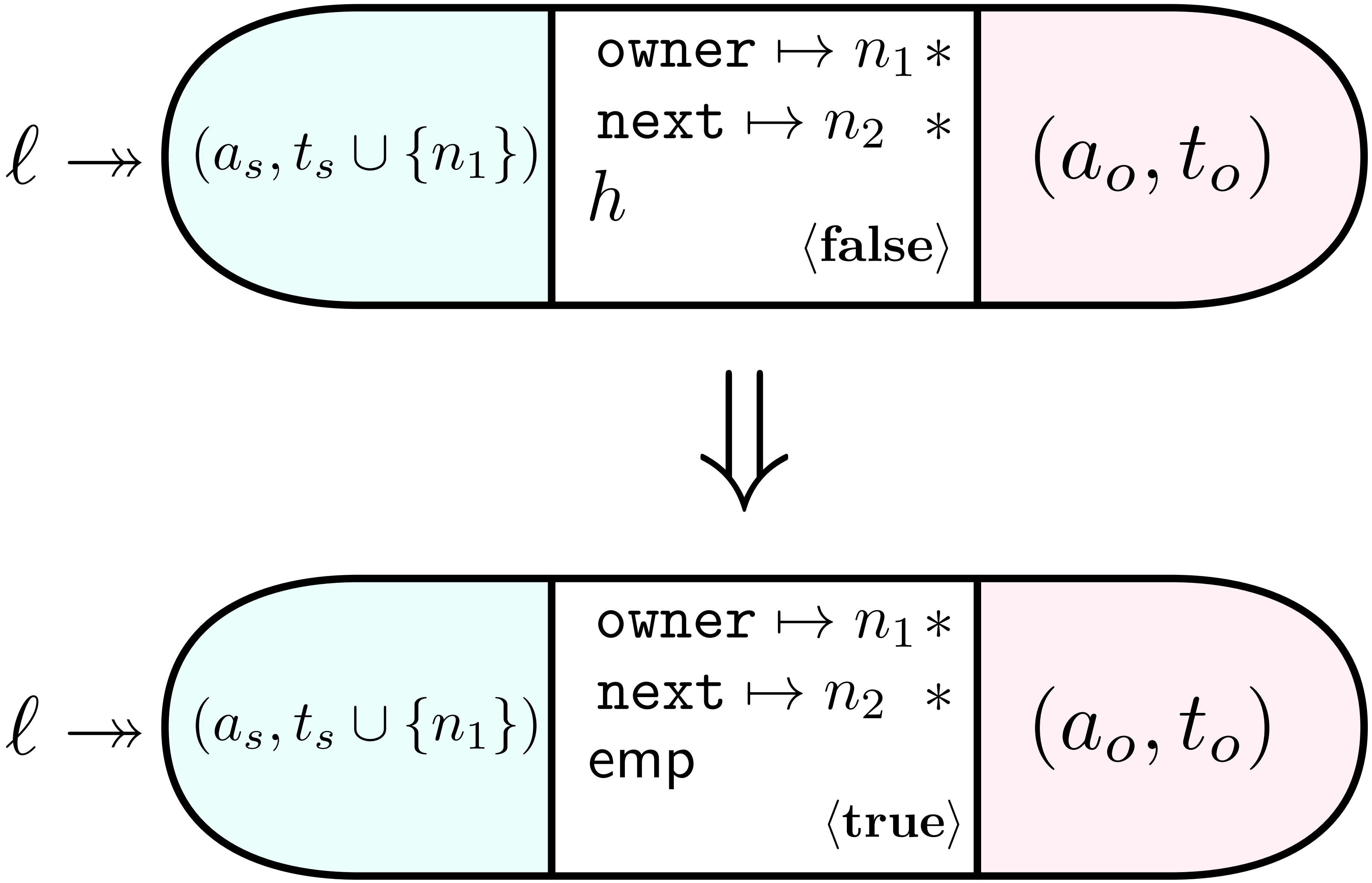
Acquire/Release transitions

(communication is via heap ownership transfer)

Release Transitions

Intuition:

the lock gives up ownership over the heap



$\ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$

$\texttt{owner} \mapsto n_1 *$

$\texttt{next} \mapsto n_2 \ *$

h

$\langle \textbf{false} \rangle$

(a_o, t_o)

$\Downarrow$

$\ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$

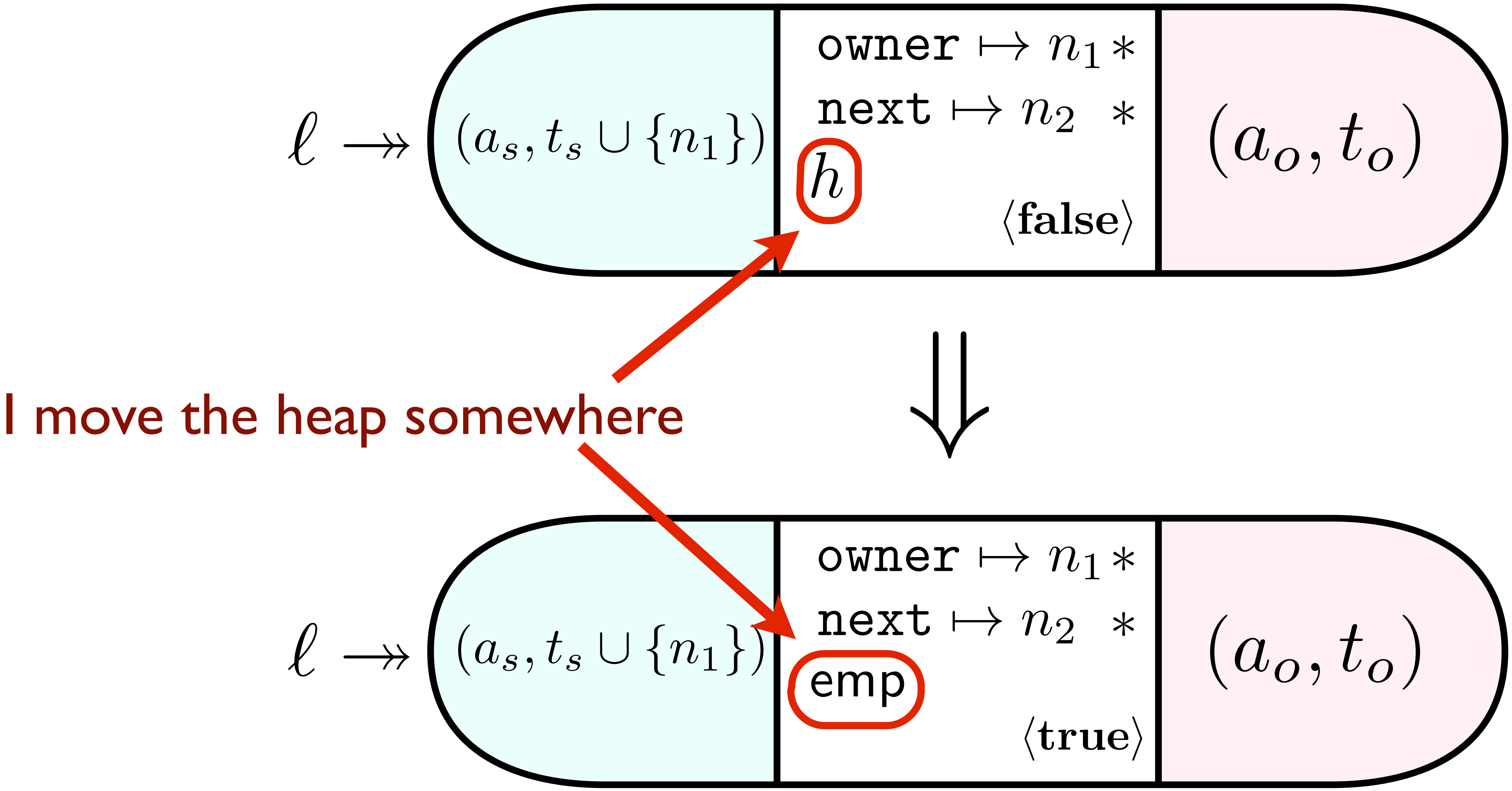
$\texttt{owner} \mapsto n_1 *$

$\texttt{next} \mapsto n_2 \ *$

emp

$\langle \textbf{true} \rangle$

(a_o, t_o)



$\ell \twoheadrightarrow (a_s, t_s \cup \{n_1\})$

$\texttt{owner} \mapsto n_1 *$

$\texttt{next} \mapsto n_2 *$

$\boxed{h}$

$\langle \textbf{false} \rangle$

(a_o, t_o)

$\Downarrow$

$\ell \twoheadrightarrow (a_s, t_s \cup \{n_1\})$

$\texttt{owner} \mapsto n_1 *$

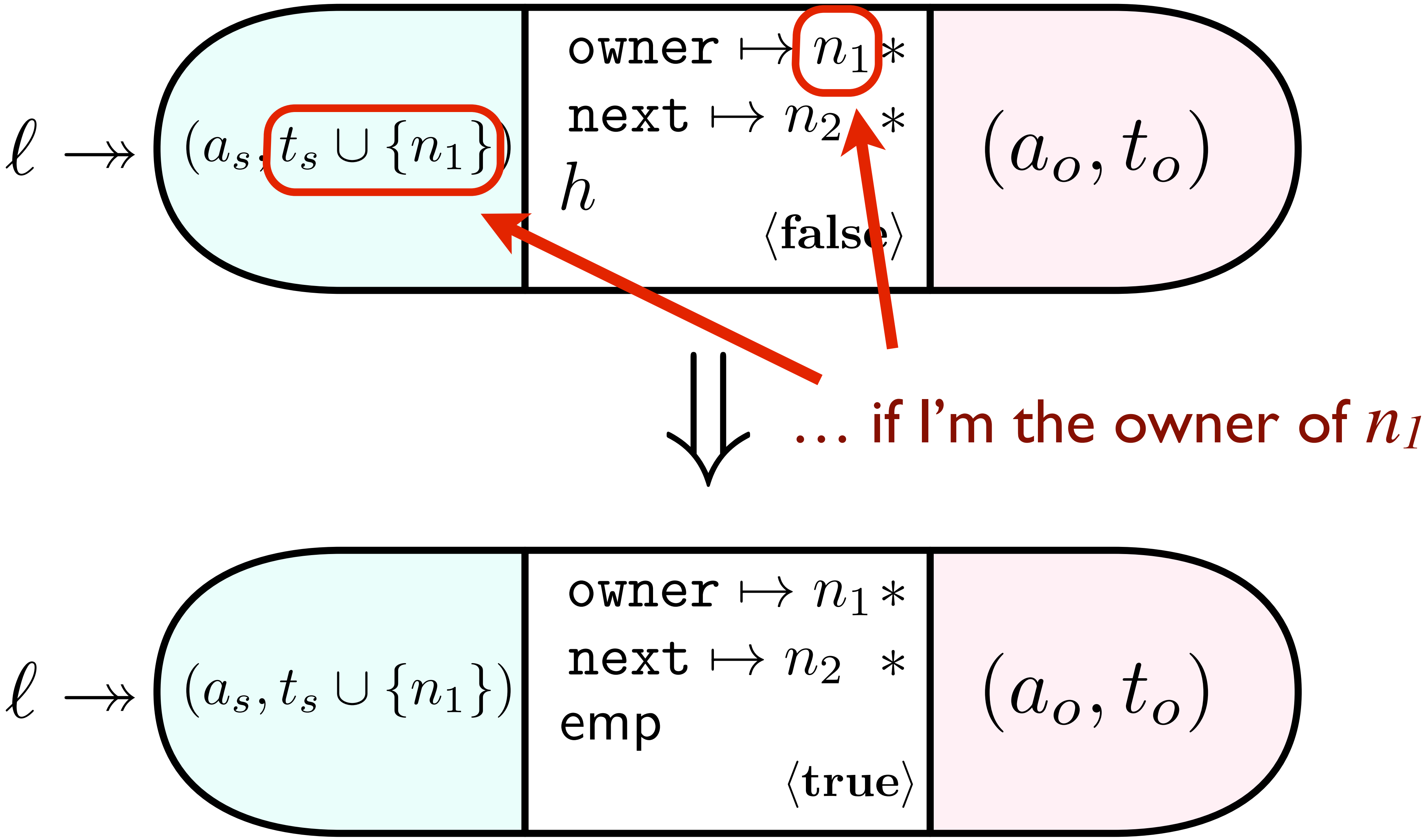
$\texttt{next} \mapsto n_2 *$

$\boxed{\mathsf{emp}}$

$\langle \textbf{true} \rangle$

(a_o, t_o)

I move the heap somewhere



$\ell \twoheadrightarrow (a_s, t_s \cup \{n_1\})$

$\texttt{owner} \mapsto n_1 \, *$

$\texttt{next} \mapsto n_2 \, *$

h

$\langle \mathbf{false} \rangle$

(a_o, t_o)

$\Downarrow$ … if I'm the owner of n_1

$\ell \twoheadrightarrow (a_s, t_s \cup \{n_1\})$

$\texttt{owner} \mapsto n_1 \, *$

$\texttt{next} \mapsto n_2 \, *$

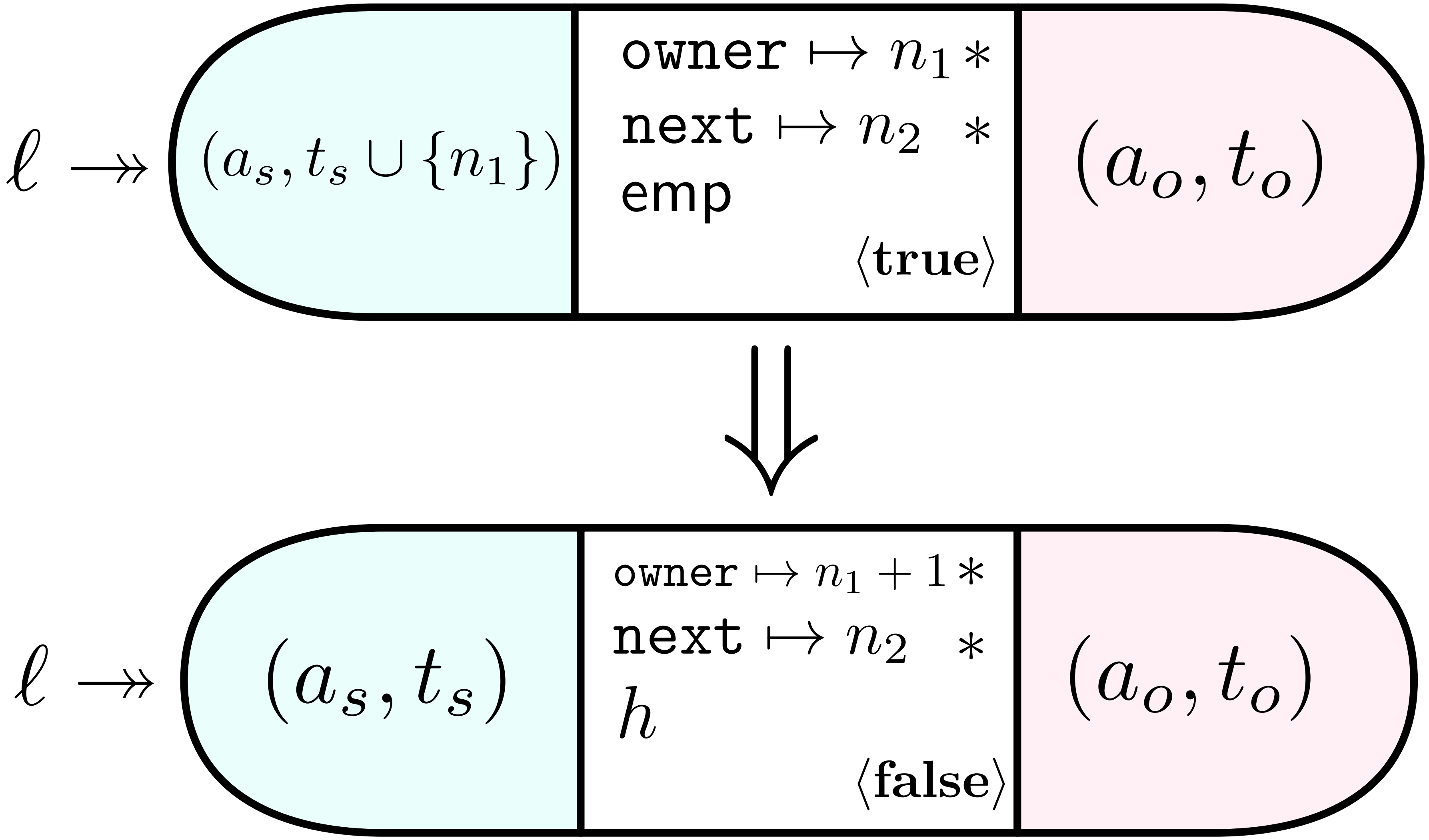
emp

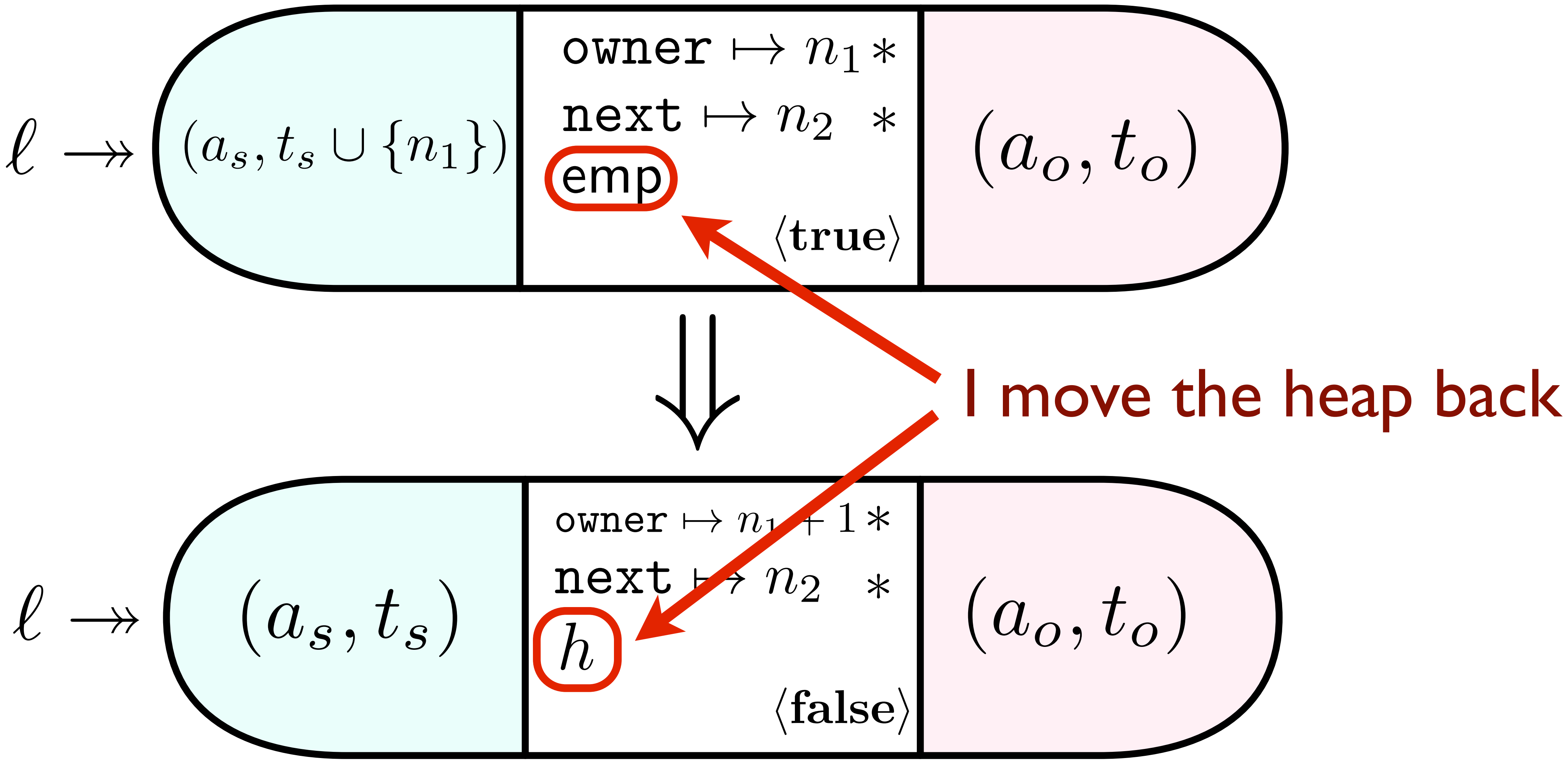
$\langle \mathbf{true} \rangle$

(a_o, t_o)

Acquire Transitions

Intuition:
the lock obtains back ownership over the heap and increments the service counter (`owner`)


$\ell \twoheadrightarrow$
$(a_s, t_s \cup \{n_1\})$
$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 \ *$
emp
$\langle \mathbf{true} \rangle$
(a_o, t_o)
$\Downarrow$
$\ell \twoheadrightarrow$
(a_s, t_s)
$\texttt{owner} \mapsto n_1 + 1 *$
$\texttt{next} \mapsto n_2 \ *$
h
$\langle \mathbf{false} \rangle$
(a_o, t_o)

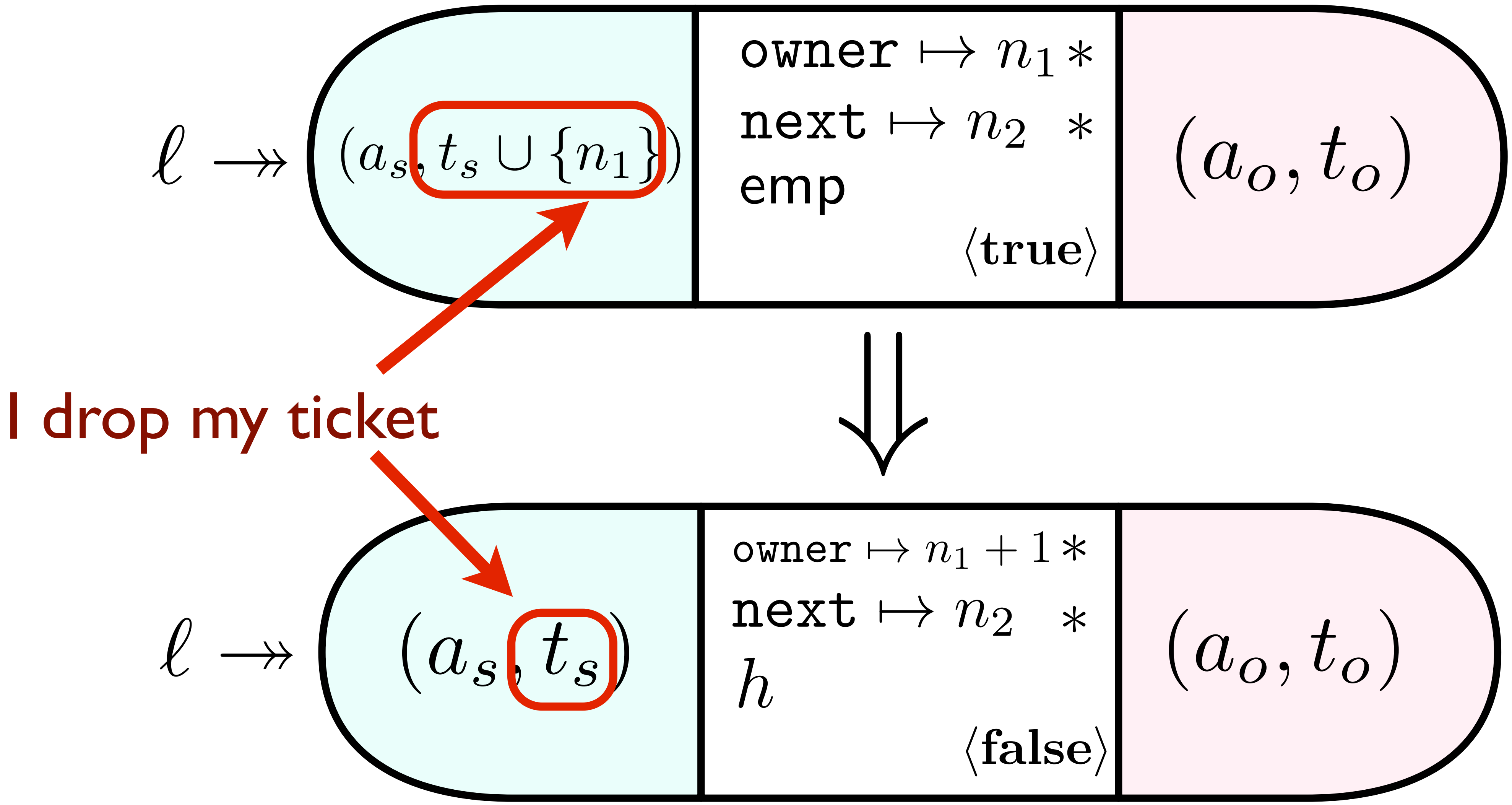


$\ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$ | $\texttt{owner} \mapsto n_1 *$ $\texttt{next} \mapsto n_2 *$ emp $\langle\mathbf{true}\rangle$ | (a_o, t_o)

$\Downarrow$

$\ell \twoheadrightarrow$ (a_s, t_s) | $\texttt{owner} \mapsto n_1 + 1 *$ $\texttt{next} \mapsto n_2 *$ h $\langle\mathbf{false}\rangle$ | (a_o, t_o)

I move the heap back



$\ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$

$\texttt{owner} \mapsto n_1 *$

$\texttt{next} \mapsto n_2 \quad *$

$\texttt{emp}$

$\langle \textbf{true} \rangle$

(a_o, t_o)

$\Downarrow$

$\ell \twoheadrightarrow$ (a_s, t_s)

$\texttt{owner} \mapsto n_1 + 1 *$

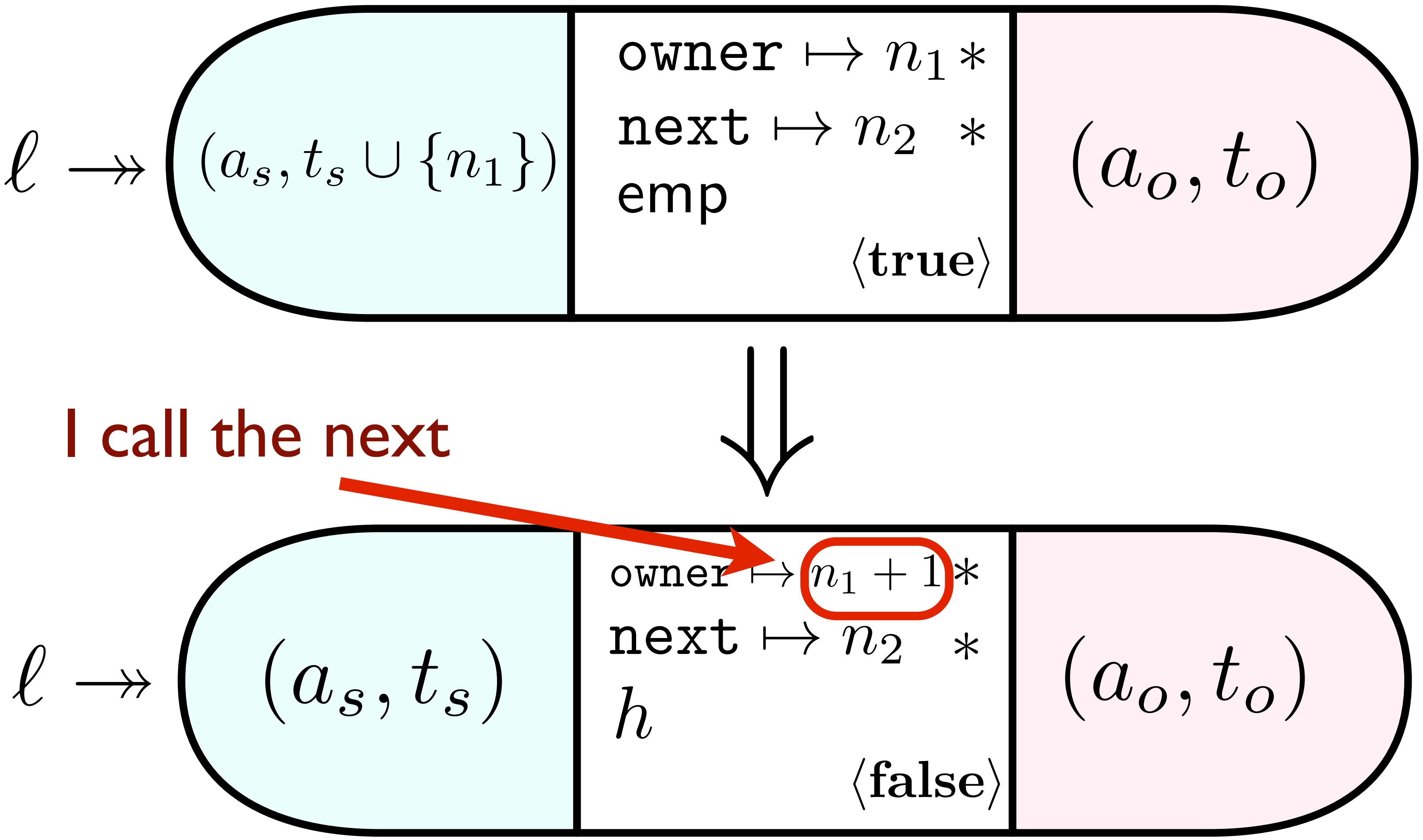
$\texttt{next} \mapsto n_2 \quad *$

h

$\langle \textbf{false} \rangle$

(a_o, t_o)

I drop my ticket



$\ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$ $\texttt{owner} \mapsto n_1 *$ $\texttt{next} \mapsto n_2 \; *$ $\textsf{emp}$ $\langle\textbf{true}\rangle$ (a_o, t_o)

$\Downarrow$

I call the next

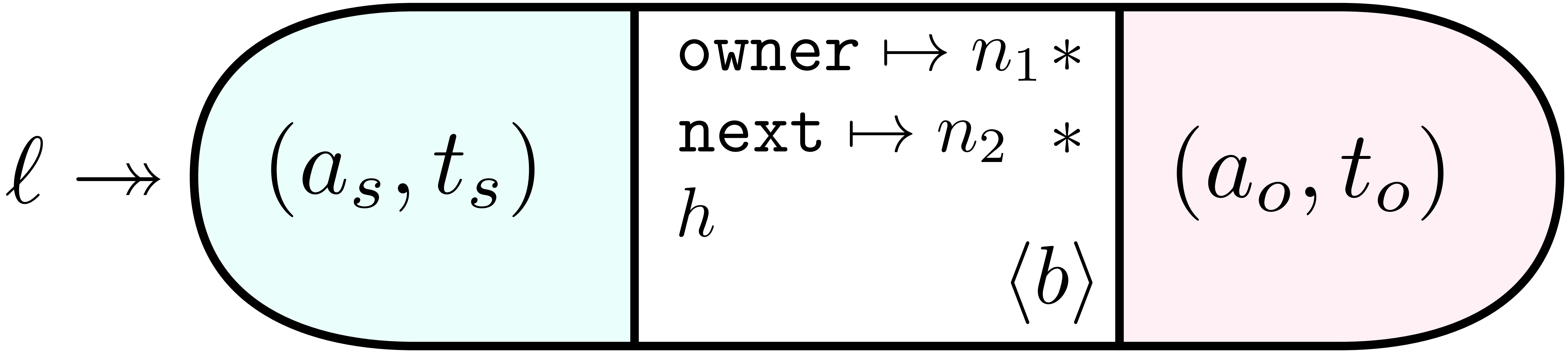
$\ell \twoheadrightarrow$ (a_s, t_s) $\texttt{owner} \mapsto \boxed{n_1 + 1} *$ $\texttt{next} \mapsto n_2 \; *$ h $\langle\textbf{false}\rangle$ (a_o, t_o)

Transitions **never** change the *other* part!

Transitions **never** change the *other* part!

Transitions = *Guarantee*

Transposing the Concurroid



$\ell \twoheadrightarrow$

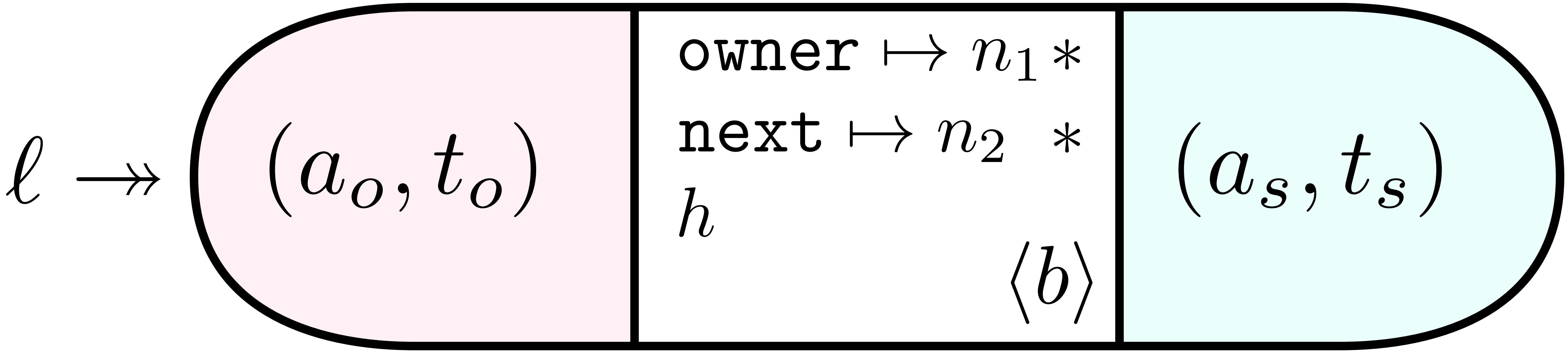
(a_s, t_s)

$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 \;\; *$
h

$\langle b \rangle$

(a_o, t_o)

Transposing the Concurroid



$\ell \twoheadrightarrow$

(a_o, t_o)

$\texttt{owner} \mapsto n_1 *$

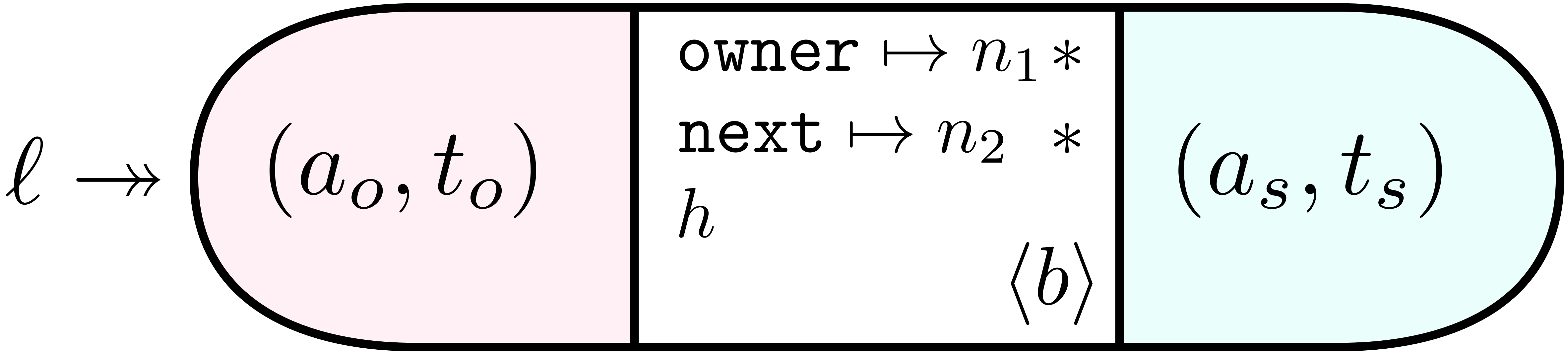
$\texttt{next} \mapsto n_2 \;\; *$

h

$\langle b \rangle$

(a_s, t_s)

Transposing the Concurroid



$\ell \twoheadrightarrow$

(a_o, t_o)

$\texttt{owner} \mapsto n_1 *$
$\texttt{next} \mapsto n_2 \ *$
h
$\langle b \rangle$

(a_s, t_s)

Transitions of transposed = *Rely*

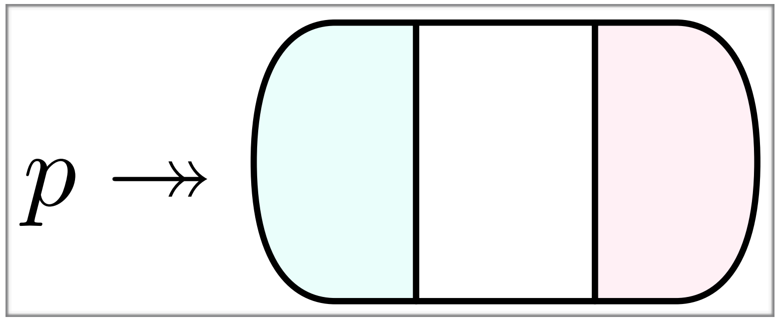
Composing Concurroids

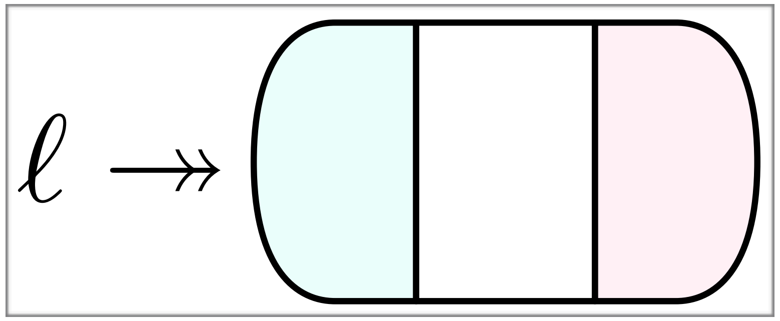
Intuition:

Connect communication channels with right polarity

Intuition:

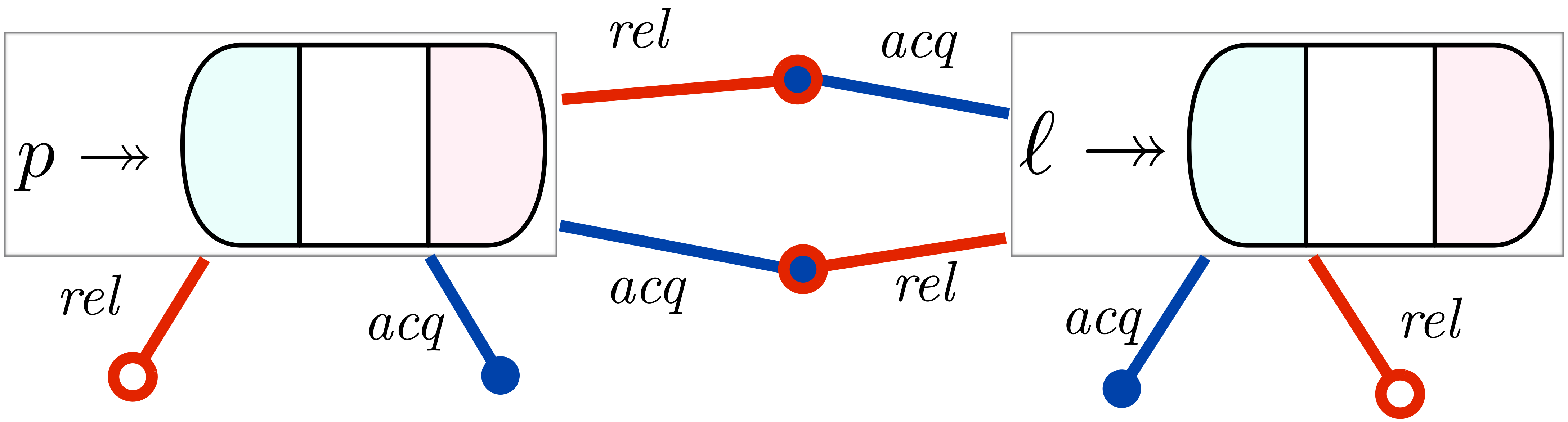
Connect communication channels with right polarity





Intuition:

Connect communication channels with right polarity



- Some channels might be left loose
- Some channels might be shut down
- *Same* channels might be connected several times

Entanglement Operators

$\bowtie, \rtimes, \ltimes, \times ...$

Connect two concurroids by connecting some of their acquire/release transitions.

Entanglement Operators

$\bowtie, \rtimes, \ltimes, \times$...

Connect two concurroids by connecting some of their acquire/release transitions.

Connected A/R transitions become *internal* for the entanglement.

Programming with Concurroids

Transitions are not yet commands!

Transitions are not yet commands!

They are just *specifications* of some *correct* behavior of a resource.

Concurroid-Aware Actions

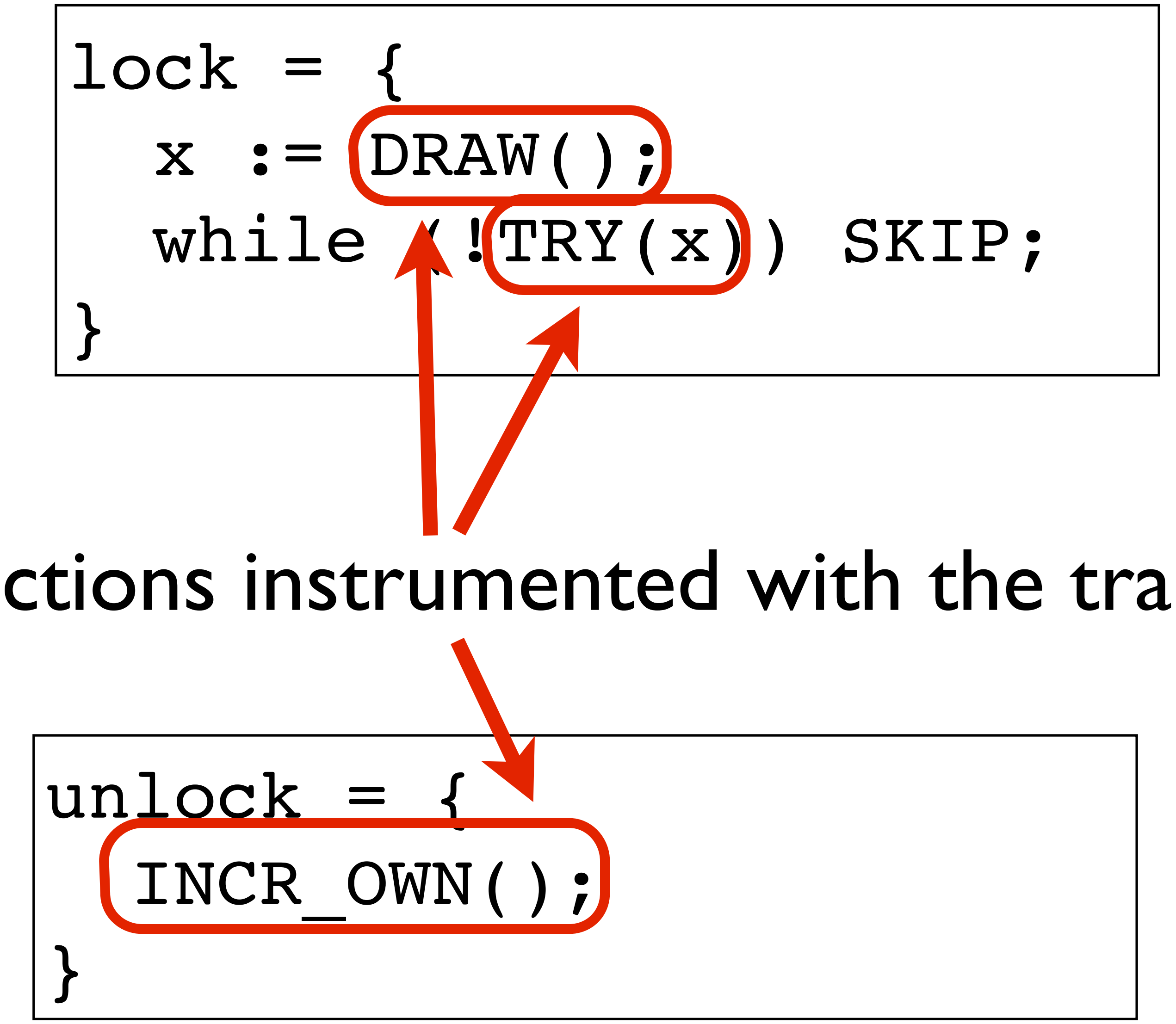
- Decorate machine commands with concurroid's *internal* transitions;
- Specify the result;
- Operational meaning: `READ`, `WRITE`, `SKIP` and various RMW-commands;
- All other command connectives are standard.

Recap: TLock Implementation

```
lock = {
  x := DRAW();
  while (!TRY(x)) SKIP;
}
```

```
unlock = {
  INCR_OWN();
}
```

Recap: TLock Implementation



```
lock = {
   x := DRAW();
   while (!TRY(x)) SKIP;
}
```

Atomic actions instrumented with the transition logic

```
unlock = {
  INCR_OWN();
}
```

Scaling along the two dimensions:

Proof Rules

Scaling along X: Parallel Composition

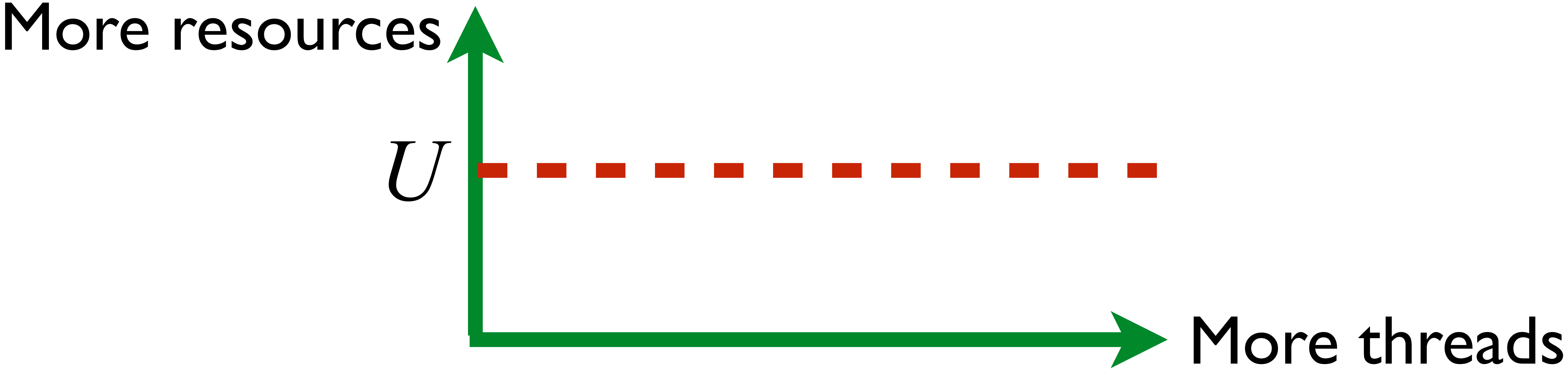
$$\frac{\{p_1\}\, C_1\, \{q_1\}\ @\ U \qquad \{p_2\}\, C_2\, \{q_2\}\ @\ U}{\{p_1 \circledast p_2\}\, C_1 \parallel C_2\, \{q_1 \circledast q_2\}\ @\ U}\ \textsc{Par}$$

where $\circledast$ accounts for adapting <u>self</u>/<u>other</u> view

Scaling along X: Parallel Composition

$$\frac{\{p_1\}\, C_1\, \{q_1\}\ @\ \boxed{U} \qquad \{p_2\}\, C_2\, \{q_2\}\ @\ \boxed{U}}{\{p_1 \circledast p_2\}\, C_1 \parallel C_2\, \{q_1 \circledast q_2\}\ @\ \boxed{U}}\ \textsc{Par}$$

where $\circledast$ accounts for adapting self/other view

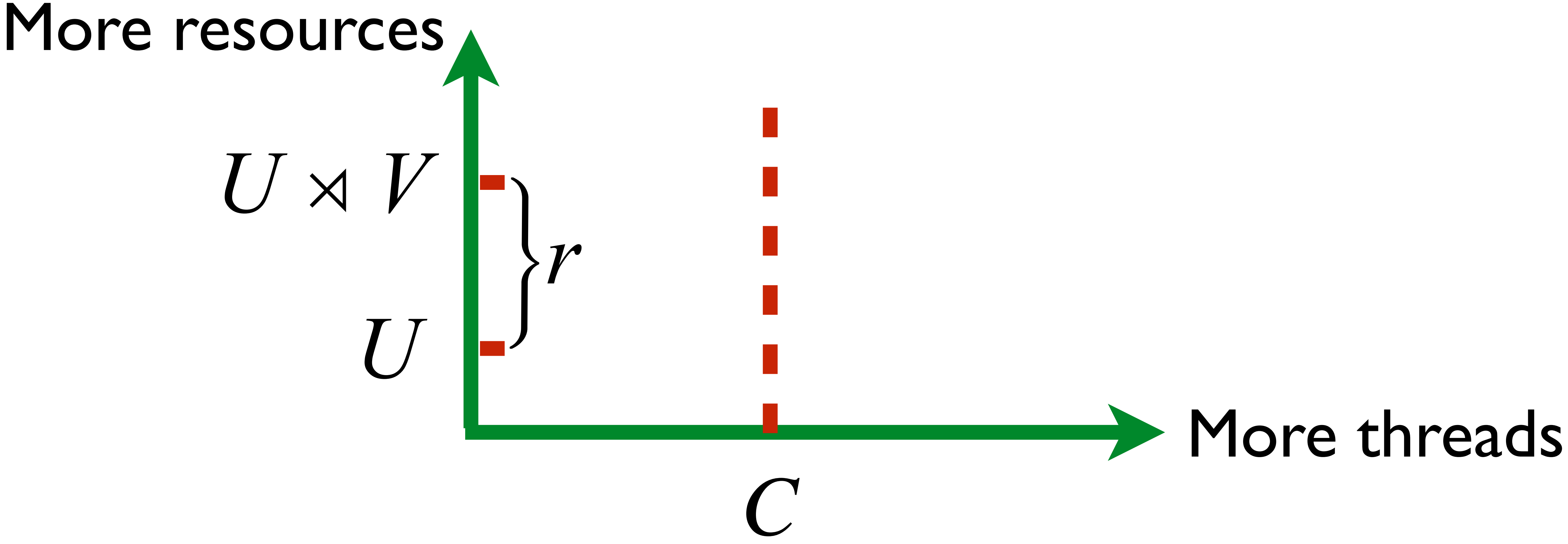


Scaling along Y: Injection

$$\frac{\{p\}\, C\, \{q\}\ @\ U \qquad\qquad r \text{ stable under } V}{\{p * r\}\, \mathsf{inject}_V\, C\, \{q * r\}\ @\ U \rtimes V}\ \textsc{Inject}$$

Scaling along Y: Injection

$$\frac{\{p\}\,C\,\{q\} \mathbin{@} U \qquad \boxed{r \text{ stable under } V}}{\{p * \boxed{r}\}\ \mathsf{inject}_V\ C\ \{q * \boxed{r}\} \mathbin{@} U \rtimes V}\ \textsc{Inject}$$



Not discussed in this talk

- Scoped creation/disposal of concurroids *(see the paper)*
- A concurroid for a spin-lock *(see the paper)*
- A concurroid model for readers/writers *(talk to me)*
- Abstract predicates (yes, we can do it, too) *(see the TR)*
- Denotational semantics of trees-of-traces *(see the TR)*
- Soundness of the logic *(check the TR or the Coq code)*

Implementation

- Implementation in Coq: metatheory, logic, proofs;
- Shallow embedding into the CIC (~15 KLOC);
- Higher-orderness and abstraction *for free;*
- Reasoning in HTT-style: Hoare specifications are types;
- Some automation is done for splitting the state among concurroids;
- Spin-lock and Ticketed lock are fully implemented.

To take away

- **State Transition Systems** are expressive behavioural specifications of shared resources;
- **Self/Other Dichotomy** is omnipresent when reasoning about shared-memory concurrency *(composing N threads)*;
- **Communication** is a way to describe state ownership transfer between resources *(composing N resources)*.

To take away

- **State Transition Systems** are expressive behavioural specifications of shared resources;

- **Self/Other Dichotomy** is omnipresent when reasoning about shared-memory concurrency *(composing N threads)*;

- **Communication** is a way to describe state ownership transfer between resources *(composing N resources)*.

Concurroids unify these concepts in one data structure.

To take away

- **State Transition Systems** are expressive behavioural specifications of shared resources;
- **Self/Other Dichotomy** is omnipresent when reasoning about shared-memory concurrency *(composing N threads)*;
- **Communication** is a way to describe state ownership transfer between resources *(composing N resources)*.

Concurroids unify these concepts in one data structure.

Thanks!

Q&A Slides

How the subjective split is defined?

$$
\begin{aligned}
w \models p \circledast q \quad & \text{iff valid } w \text{, and } w.\,s = s_1 \mathbin{\dot\cup} s_2 \text{, and} \\
& \quad [s_1 \mid w.\,j \mid s_2 \circ w.\,o] \models p \text{ and } [s_2 \mid w.\,j \mid s_1 \circ w.\,o] \models q
\end{aligned}
$$

How the subjective split is defined?

$$w \models p \circledast q \quad \text{iff valid } w\text{, and } w.\,s = s_1 \mathbin{\dot{\cup}} s_2\text{, and}$$
$$[\boxed{s_1} \mid w.\,j \mid \boxed{s_2 \circ w.\,o}] \models p \text{ and } [\boxed{s_2} \mid w.\,j \mid \boxed{s_1 \circ w.\,o}] \models q$$

“Forking shuffle” for the self/other components.

Why do you need the *explicit* other?

Why do you need the *explicit* other?

- Some programs are *easier* to specify and verify using the other:
  - E.g., in the *lock* module the other doesn't change if the lock is locked by self.
- Some programs are **much** easier to specify via the other:
  - Typically, optimistic, *non-effectful* programs (e.g., stack's *contains(x)*).
- other makes the *duality* between Rely and Guarantee explicit
  - and, in fact, the form of other is already present in R/G (it's just *Rely*)
- It's already in the model, so why not use it when it comes in handy?

Can't I just infer the other from some global/self knowledge?

Can’t I just infer the other from some global/self knowledge?

You can try. :)

But then you need to define your “*global*” to subtract the self from.

With other you don’t need to subtract.

Can’t we just use *Tokens* or *Fractional Permissions* instead of <u>other</u>?

Can’t we just use *Tokens* or *Fractional Permissions* instead of other?

Yes, you can.

Since both tokens and FP are just instances of PCM, you can, probably, instantiate self/other with any of them.

Can’t we just use *Tokens* or *Fractional Permissions* instead of other?

Yes, you can.

Since both tokens and FP are just instances of PCM, you can, probably, instantiate self/other with any of them.

But why bother? :)

Aren’t self/other just about ownership?

Aren’t self/other just about ownership?

No, they are not.

Aren’t self/other just about ownership?

No, they are not.

- Ownership assumes a holistic *“preservation law”* — ***everything is created in advance and owned by someone;***

Aren’t self/other just about ownership?

No, they are not.

- Ownership assumes a holistic *“preservation law”* — ***everything is created in advance and owned by someone;***
- Consider a Ticketed Lock example with ownership:
  - we need to account for *all currently used tickets;*
  - we need to account for *all disposed tickets*;
  - we need to account for *all not yet dispensed tickets*;
  - In our case we don’t bother about the last two.

Aren't self/other just about ownership?

No, they are not.

- Ownership assumes a holistic *"preservation law"* — ***everything is created in advance and owned by someone;***
- Consider a Ticketed Lock example with ownership:
  - we need to account for *all currently used tickets;*
  - we need to account for *all disposed tickets*;
  - we need to account for *all not yet dispensed tickets*;
  - In our case we don't bother about the last two.

Self/other dichotomy delivers more local reasoning ⇒

proofs are simpler!

Can you extract the verified program from your Coq implementation and run it?

Can you extract the verified program from your Coq implementation and run it?

Yes and no.

Can you extract the verified program from your Coq implementation and run it?

Yes and no.

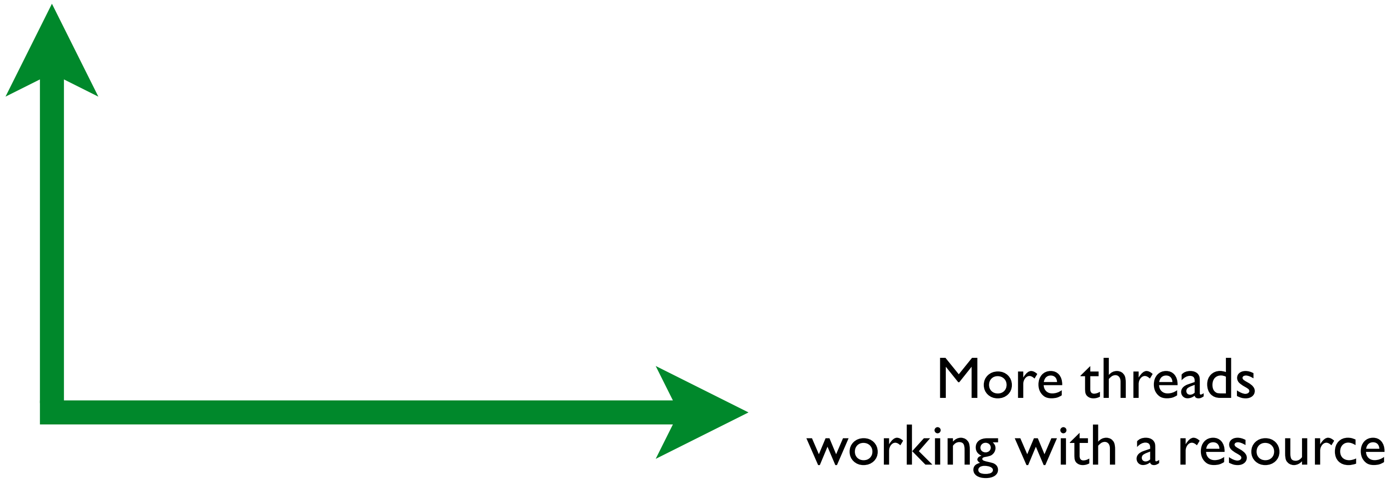
- Imperative programs are composed and verified (i.e., type-checked) by means of Coq;
- They cannot be run by means of Gallina's operational semantics;
- The reason for that is the necessity to reason about **while**-loops and potentially diverging programs;
- Think of our programs as of monadic values, which are *composed*, but not *run* yet.

Isn’t other just about framing?

Isn’t other just about framing?

Yes, in some sense it is.

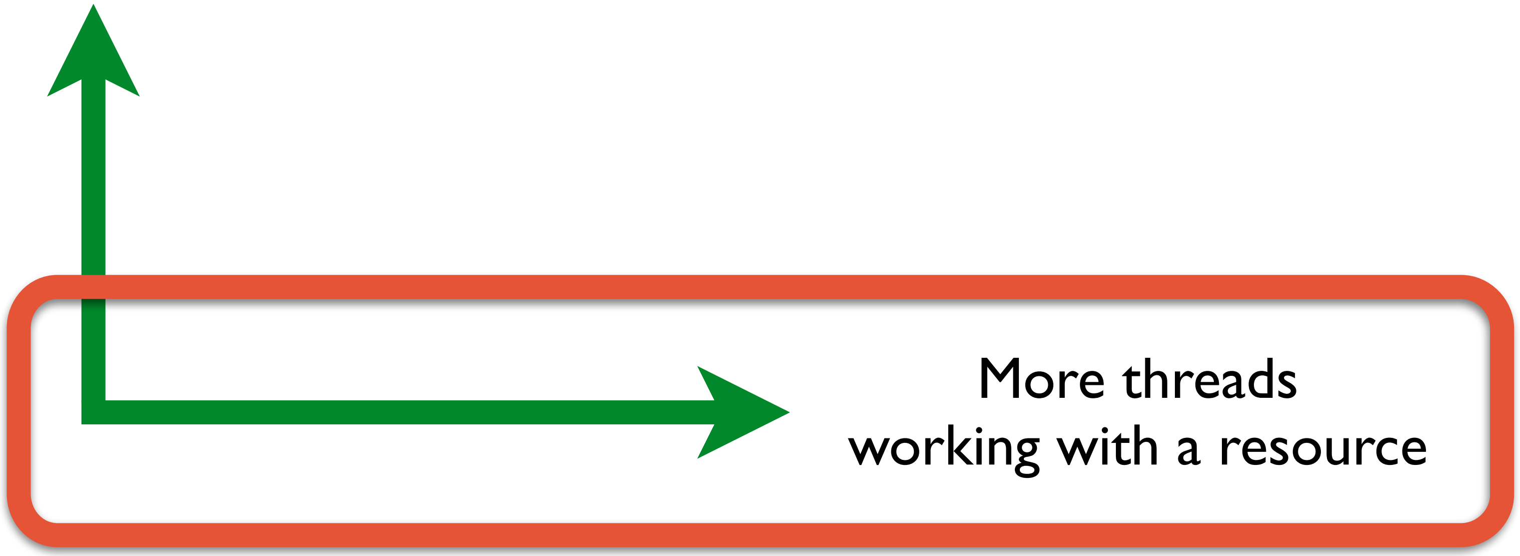
But just along just one axis of scalability.



Isn’t other just about framing?

Yes, in some sense it is.

But just along just one axis of scalability.



Other complements self for a particular resource.

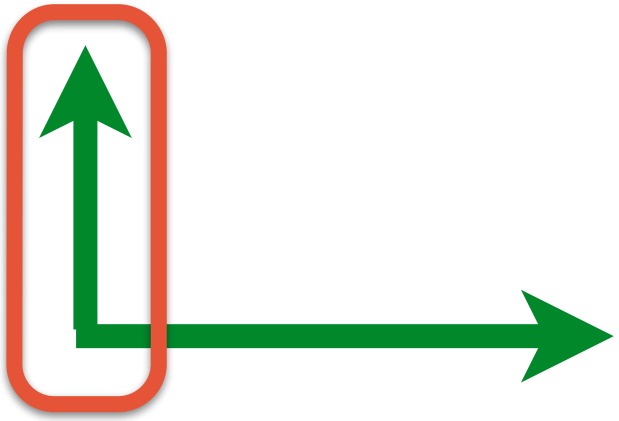
Why do you have two framing rules?

$$\frac{\Gamma \vdash \{p\}\, c : A\, \{q\} @ U \qquad r \text{ stable under } V}{\Gamma \vdash \{p * r\}\, \mathsf{inject}\ c : A\, \{q * r\} @ U \rtimes V}\ \textsc{Inject}$$

$$\frac{\{p_1\}\, C_1\, \{q_1\}\ @\ U \qquad \{p_2\}\, C_2\, \{q_2\}\ @\ U}{\{p_1 \circledast p_2\}\, C_1 \parallel C_2\, \{q_1 \circledast q_2\}\ @\ U}\ \textsc{Par}$$

Why do you have two framing rules?

$$\frac{\Gamma \vdash \{p\}\, c : A\, \{q\} @ U \qquad r \text{ stable under } V}{\Gamma \vdash \{p * r\}\, \textsf{inject}\ c : A\, \{q * r\} @ U \rtimes V}\ \textsc{Inject}$$

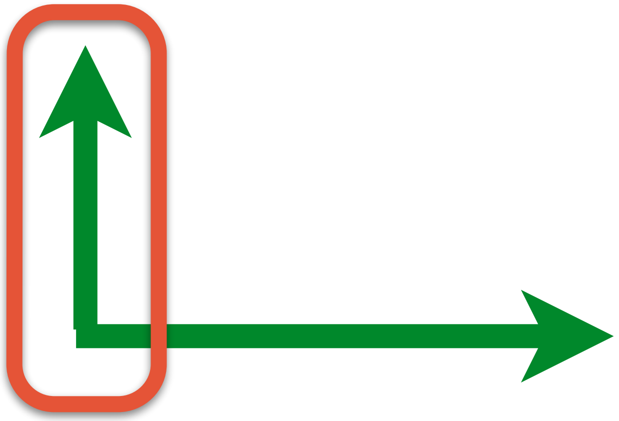


Framing with respect to the **other** resource V.

$$\frac{\{p_1\}\, C_1\, \{q_1\}\ @\ U \qquad \{p_2\}\, C_2\, \{q_2\}\ @\ U}{\{p_1 \circledast p_2\}\, C_1 \parallel C_2\, \{q_1 \circledast q_2\}\ @\ U}\ \textsc{Par}$$

Why do you have two framing rules?

$$\frac{\Gamma \vdash \{p\}\, c : A\, \{q\} @ U \qquad r \text{ stable under } V}{\Gamma \vdash \{p * r\}\, \mathsf{inject}\; c : A\, \{q * r\} @ U \rtimes V}\ \textsc{Inject}$$



Framing with respect to the **other** resource V.

$$\frac{\{p_1\}\, C_1\, \{q_1\} \ @\ U \qquad \{p_2\}\, C_2\, \{q_2\} \ @\ U}{\{p_1 \circledast p_2\}\, C_1 \parallel C_2\, \{q_1 \circledast q_2\} \ @\ U}\ \textsc{Par}$$

Framing — particular case of parallel composition on the **same** resource U.

“Framing” rules in CSL

O'Hearn [CONCUR'04]

$$\frac{\Gamma; \mathit{I1} \vdash \{Q\}\ C\ \{R\}}{\Gamma; \mathit{I1} \star \mathit{I2} \vdash \{Q\}\ C\ \{R\}}$$

Resource context weakening

$$\frac{\Gamma; I \vdash \{\mathit{Q1}\}\ \mathit{C1}\ \{\mathit{R1}\} \quad \Gamma; I \vdash \{\mathit{Q2}\}\ \mathit{C2}\ \{\mathit{R2}\}}{\Gamma; I \vdash \{\mathit{Q1} \star \mathit{Q2}\}\ \mathit{C1} \| \mathit{C2}\ \{\mathit{R1} \star \mathit{R2}\}}$$

Parallel composition

“Framing” rules in RGSep

Vafeiadis-Parkinson [CONCUR’07]

$$\frac{\begin{array}{c} R \subseteq R' \quad p \Rightarrow p' \\ \vdash C \;\mathbf{sat}\; (p', R', G', q') \quad G' \subseteq G \quad q' \Rightarrow q \end{array}}{\vdash C \;\mathbf{sat}\; (p, R, G, q)}$$

Rely/Guarantee weakening

$$\frac{\begin{array}{c} \vdash C_1 \;\mathbf{sat}\; (p_1, R \cup G_2, G_1, q_1) \\ \vdash C_2 \;\mathbf{sat}\; (p_2, R \cup G_1, G_2, q_2) \end{array}}{\vdash (C_1 \| C_2) \;\mathbf{sat}\; (p_1 * p_2, R, G_1 \cup G_2, q_1 * q_2)}$$

Parallel composition

Related Work

- [Owicki-Gries:CACM76] - reasoning about parallel composition is not compositional; *subjectivity fixes that;*
- [OHearn:CONCUR04] - only one type of resources - critical sections; *we allow one to define arbitrary resources;*
- [Feng-al:ESOP07,Vafeiadis-Parkinson:CONCUR07] - framing over Rely/Guarantee, but only one shared resource: *we allow multiple ones;*
- [Feng:POPL09] - introduced local Rely/Guarantee; *we improve on it by introducing a subjective state and explicitly identifying resources as STS, hence dialysing Guarantee and Rely;*
- [DinsdaleYoung-al:ECOOP10] - first introduced concurred protocols; *we avoid heavy use of permissions (for resources, actions, regions etc.) - self-state defines what a thread is allowed to do with a resource;*
- [Krishnaswami-al:ICFP12] - superficially substructural types; *that work doesn't target concurrency;*
- [DinsdaleYoung-al:POPL13] - general framework for concurrency logic; *we present a particular logic, not clear whether it's an instance of Views;*
- [Turon-al:POPL13,ICFP13] - CaReSL and reasoning about contextual refinement; *we don't address CR, our PCM-based self/other generalise Turon's tokens; we compose resources by communication;*
- [Svendsen-al:ESOP13,ESOP14] - use much richer semantic domain, *we are avoiding fractional permissions, using communication instead of view-shifts.*

Is entanglement associative?

Is entanglement associative?

Sort of.

Is entanglement associative?

Sort of.

$\times$ - "apart", doesn't connect channels, leaves all loose.

$\rtimes$ - connects all channels pair-wise, shuts channels of the right operand, leaves left one's loose

<u>*Lemma:*</u> $U \rtimes (V_1 \times V_2) = (U \rtimes V_1) \rtimes V_2$

Backup Slides

Subjective proofs

$$RI(\mathtt{lock}) \stackrel{\text{def}}{=} \mathbf{x} \mapsto (\mathbf{a_s} \oplus \mathbf{a_o})$$

```
lock;                 ||   lock;
 x := x + 1;          ||    x := x + 1;
 a_s := a_s + 1;      ||    a_s := a_s + 1;
unlock;               ||   unlock;
```

Subjective proofs

$$RI(\mathtt{lock}) \overset{\text{def}}{=} \mathbf{x} \mapsto (\mathbf{a_s} \oplus \mathbf{a_o})$$

$$\{\ \mathbf{a_s} \mapsto \mathbf{0}\ ,\ \mathbf{a_o} \mapsto \mathbf{n}\ \}$$

```
lock;                ||   lock;

 x  := x + 1;        ||    x  := x + 1;

 a_s := a_s + 1;     ||    a_s := a_s + 1;

unlock;              ||   unlock;
```

Subjective proofs

$$\mathit{RI}(\texttt{lock}) \overset{\text{def}}{=} \mathbf{x} \mapsto (\mathbf{a_s} \oplus \mathbf{a_o})$$

$$\{\ \mathbf{a_s} \mapsto 0 + 0\ ,\ \mathbf{a_o} \mapsto \mathbf{n}\ \}$$

```
lock;                ||   lock;
 x := x + 1;         ||    x := x + 1;
 a_s := a_s + 1;     ||    a_s := a_s + 1;
unlock;              ||   unlock;
```

Subjective proofs

$$RI(\texttt{lock}) \overset{\text{def}}{=} \mathbf{x} \mapsto (\mathbf{a}_s \oplus \mathbf{a}_o)$$

$$\{\ \mathbf{a}_s \mapsto 0 + 0\ ,\ \mathbf{a}_o \mapsto \mathbf{n}\ \}$$

$$\{\ \mathbf{a}_s \mapsto 0,\ \mathbf{a}_o \mapsto \mathbf{n} + 0\ \}$$

```
      lock;              ||      lock;

       x := x + 1;       ||       x := x + 1;

       a_s := a_s + 1;   ||       a_s := a_s + 1;

      unlock;            ||      unlock;
```

Subjective proofs

$$RI(\mathtt{lock}) \overset{\text{def}}{=} \mathtt{x} \mapsto (\mathtt{a_s} \oplus \mathtt{a_o})$$

$$\{\ \mathtt{a_s} \mapsto 0 + 0\ ,\ \mathtt{a_o} \mapsto \mathtt{n}\ \}$$

$\{\ \mathtt{a_s} \mapsto 0,\ \mathtt{a_o} \mapsto \mathtt{n} + 0\ \}$

```
lock;
 x := x + 1;
```

$\mathtt{a_s} := \mathtt{a_s} + 1;$

```
unlock;
```

$\|$

$\{\ \mathtt{a_s} \mapsto 0,\ \mathtt{a_o} \mapsto \mathtt{n} + 0\ \}$

```
lock;
 x := x + 1;
```

$\mathtt{a_s} := \mathtt{a_s} + 1;$

```
unlock;
```

Subjective proofs

$$\mathit{RI}(\texttt{lock}) \overset{\text{def}}{=} \mathbf{x} \mapsto (\mathbf{a_s} \oplus \mathbf{a_o})$$

$$\{\ \mathbf{a_s} \mapsto 0 + 0\ ,\ \mathbf{a_o} \mapsto \mathbf{n}\ \}$$

$$\{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \}$$

```
lock;

 x := x + 1;
```

$$\mathbf{a_s} := \mathbf{a_s} + 1;$$

```
unlock;
```

$$\{\ \mathbf{a_s} \mapsto 1,\ \mathbf{a_o} \mapsto \mathbf{n_1}\ \}$$

$$\|$$

$$\{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \}$$

```
lock;

 x := x + 1;
```

$$\mathbf{a_s} := \mathbf{a_s} + 1;$$

```
unlock;
```

Subjective proofs

$$RI(\texttt{lock}) \overset{\text{def}}{=} \mathbf{x} \mapsto (a_s \oplus a_o)$$

$$\{\; a_s \mapsto 0 + 0 \;,\; a_o \mapsto n \;\}$$

$$\begin{array}{l|l}
\{\; a_s \mapsto 0,\; a_o \mapsto n + 0 \;\} & \{\; a_s \mapsto 0,\; a_o \mapsto n + 0 \;\} \\
\quad \texttt{lock;} & \quad \texttt{lock;} \\
\quad\quad \texttt{x := x + 1;} & \quad\quad \texttt{x := x + 1;} \\
\quad\quad a_s \texttt{ := } a_s \texttt{ + 1;} & \quad\quad a_s \texttt{ := } a_s \texttt{ + 1;} \\
\quad \texttt{unlock;} & \quad \texttt{unlock;} \\
\{\; a_s \mapsto 1,\; a_o \mapsto n_1 \;\} & \{\; a_s \mapsto 1,\; a_o \mapsto n_2 \;\}
\end{array}$$

Subjective proofs

$$RI(\texttt{lock}) \overset{\text{def}}{=} \mathbf{x} \mapsto (\mathbf{a_s} \oplus \mathbf{a_o})$$

$$\{\ \mathbf{a_s} \mapsto 0 + 0\ ,\ \mathbf{a_o} \mapsto \mathbf{n}\ \}$$

$$\begin{array}{c||c}
\{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \} & \{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \} \\
\texttt{lock;} & \texttt{lock;} \\
\texttt{x := x + 1;} & \texttt{x := x + 1;} \\
\mathbf{a_s} := \mathbf{a_s} + 1; & \mathbf{a_s} := \mathbf{a_s} + 1; \\
\texttt{unlock;} & \texttt{unlock;} \\
\{\ \mathbf{a_s} \mapsto 1,\ \mathbf{a_o} \mapsto \mathbf{n_1}\ \} & \{\ \mathbf{a_s} \mapsto 1,\ \mathbf{a_o} \mapsto \mathbf{n_2}\ \}
\end{array}$$

$$\{\ \mathbf{a_s} \mapsto 1 + 1,\ \exists \mathbf{n'},\ \mathbf{a_o} \mapsto \mathbf{n'},\ \mathbf{n_1} = \mathbf{n} + 1,\ \mathbf{n_2} = \mathbf{n'} + 1\ \}$$

Subjective proofs

$RI(\mathtt{lock}) \overset{\text{def}}{=} \mathbf{x} \mapsto (\mathbf{a_s} \oplus \mathbf{a_o})$

$\{\ \mathbf{a_s} \mapsto 0 + 0\ ,\ \mathbf{a_o} \mapsto \mathbf{n}\ \}$

$\{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \}$

```
lock;

 x := x + 1;
```

$\mathtt{a_s := a_s + 1;}$

```
unlock;
```

$\{\ \mathbf{a_s} \mapsto 1,\ \mathbf{a_o} \mapsto \mathbf{n_1}\ \}$

||

$\{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \}$

```
lock;

 x := x + 1;
```

$\mathtt{a_s := a_s + 1;}$

```
unlock;
```

$\{\ \mathbf{a_s} \mapsto 1,\ \mathbf{a_o} \mapsto \mathbf{n_2}\ \}$

$\{\ \mathbf{a_s} \mapsto 2,\ \mathbf{a_o} \mapsto -\ \}$

Creating and disposing concurroids

Creating and disposing resources

CSL Resource Rule

O'Hearn [CONCUR'04]

$$\frac{\mathit{\Gamma}, r : I \vdash \{p\}\ c\ \{q\}}{\mathit{\Gamma} \vdash \{p * I\}\ \text{resource}\ r\ \text{in}\ c\ \{q * I\}}\ \textsc{ResourceCSL}$$

CSL Resource Rule

O'Hearn [CONCUR'04]

$$\frac{\Gamma, r : I \vdash \{p\}\ c\ \{q\}}{\Gamma \vdash \{p * \boxed{I}\}\ \text{resource}\ r\ \text{in}\ c\ \{q * \boxed{I}\}}\ \textsc{ResourceCSL}$$

CSL Resource Rule

O'Hearn [CONCUR'04]

$$\frac{\Gamma, \boxed{r : I} \vdash \{p\}\ c\ \{q\}}{\Gamma \vdash \{p * I\}\ \text{resource}\ r\ \text{in}\ c\ \{q * I\}}\ \textsc{ResourceCSL}$$

Allocating a Ticketed Lock

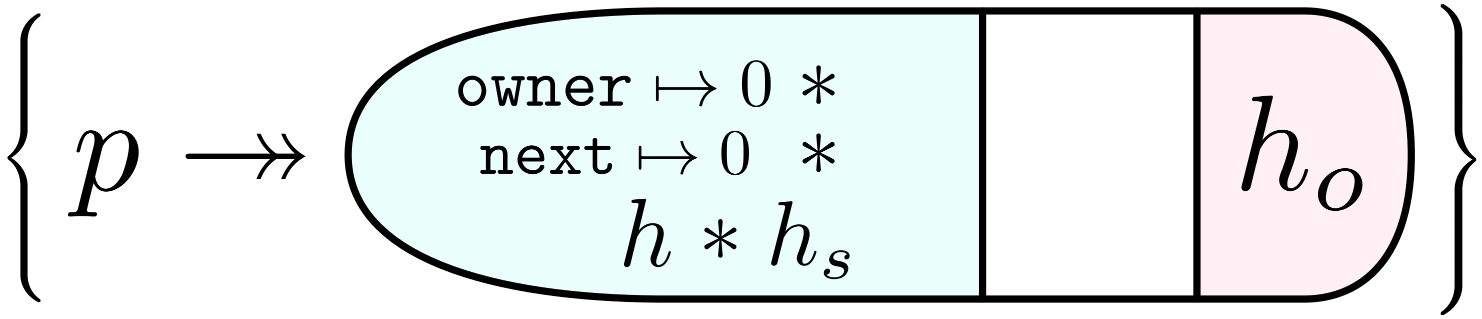
```
with_tlock(owner, next, body ) = {
    owner := 0;
    next  := 0;
    $hide_{coh_{(\mathrm{tlock}\ \ell(\texttt{owner},\texttt{next})),(a_S,\emptyset)}}$ {

        body;

     }
}
```

Allocating a Ticketed Lock

```
with_tlock(owner, next, body ) = {
    owner := 0;
    next  := 0;
    $hide_{coh_{(\mathtt{tlock}\ \ell(\mathtt{owner},\mathtt{next})),(a_s,\emptyset)}}$ {

        body;

    }
}
```

Scoped concurroid creation/disposal

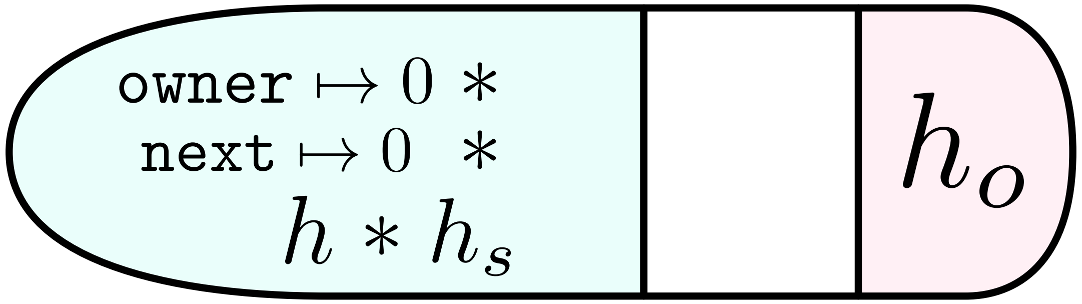
```
$hide_{coh_{(\mathrm{tlock}\ \ell(\texttt{owner,next})),(a_s,\emptyset)}}$ {

    body;

}
```



$\left\{ p \twoheadrightarrow \right.$ `owner` $\mapsto 0 \; *$ `next` $\mapsto 0 \; *$ $h * h_s$ $h_o \left. \right\}$

$$hide_{coh(\mathrm{tlock}\ \ell(\texttt{owner},\texttt{next})),(a_s,\emptyset)}\ \texttt{\{}$$

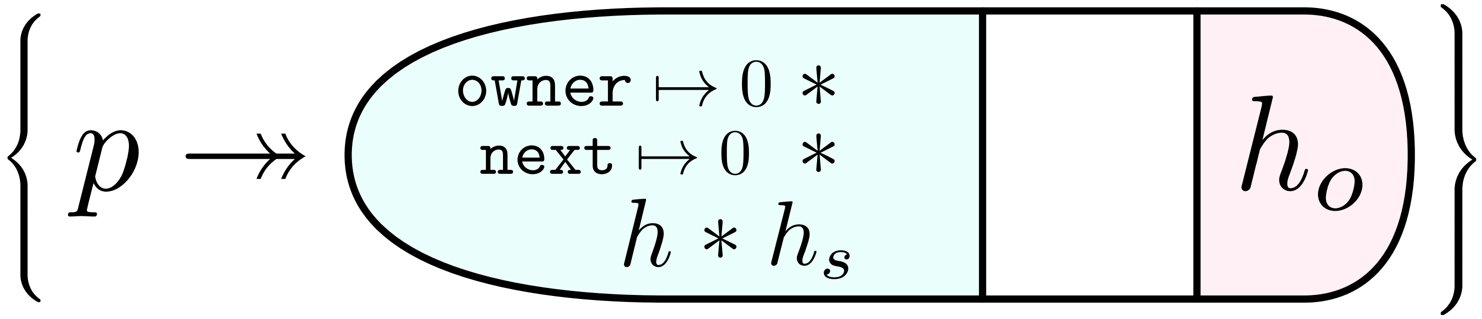
```
    body;

}
```

$\Big\{ p \twoheadrightarrow$  $\Big\}$

Concurroid spec

$$hide_{\boxed{coh_{(\texttt{tlock}\ \ell(\texttt{owner},\texttt{next}))}},(a_s,\emptyset)}\ \{$$

```
    body;

}
```



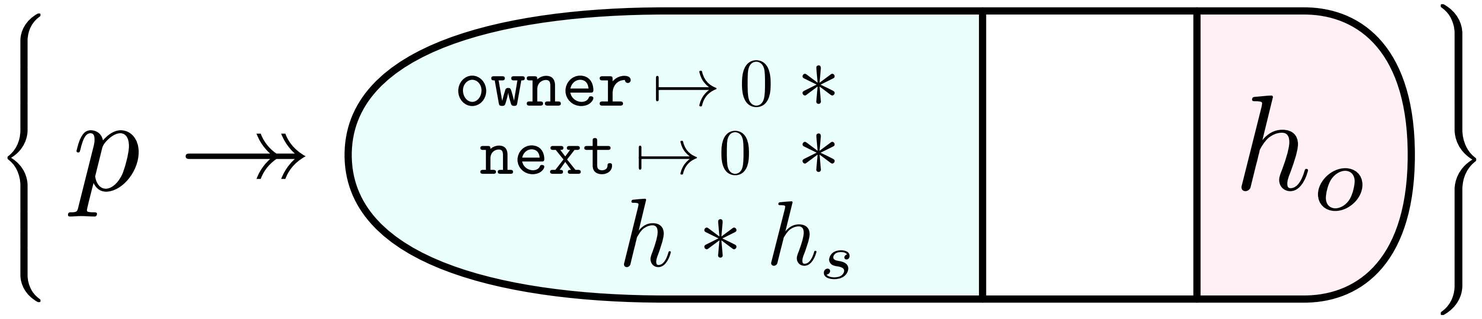
$$\left\{ p \twoheadrightarrow \quad \texttt{owner} \mapsto 0 \ast \quad \texttt{next} \mapsto 0 \ast \quad h \ast h_s \quad \Big| \quad \Big| \quad h_o \right\}$$

Concurroid spec

Initial “self” auxiliaries

$$hide_{\boxed{coh_{(\texttt{tlock}\ \ell(\texttt{owner},\texttt{next}))}}, \boxed{(a_s, \emptyset)}} \ \{$$

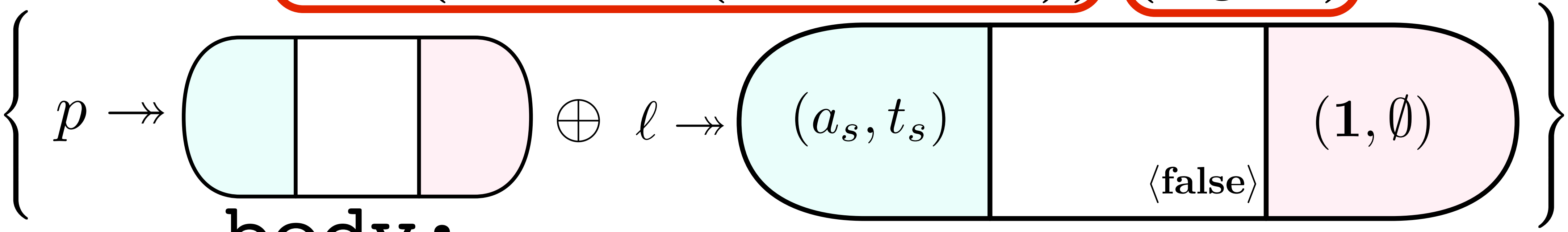
```
    body;

}
```



Concurroid spec

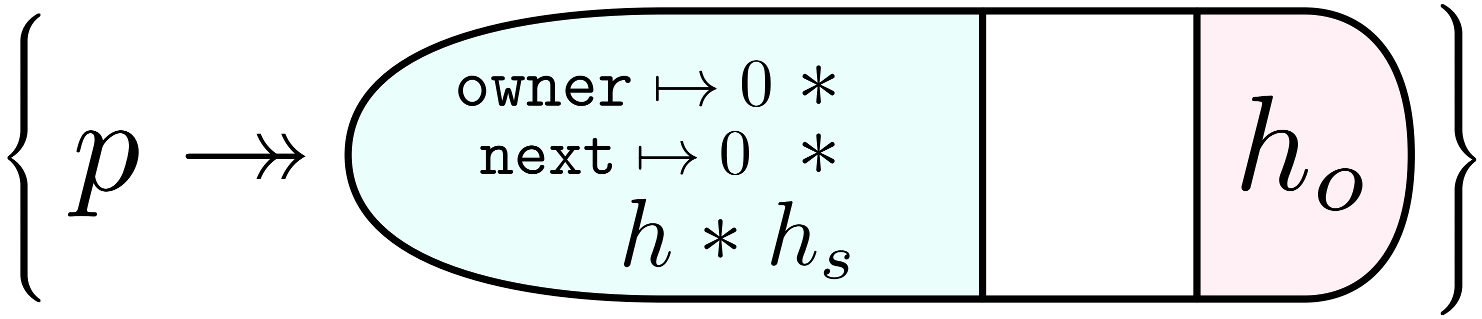
Initial “self” auxiliaries

$hide\ coh($tlock $\ell($owner,next$))$, $(a_s, \emptyset)$ {


$\{\; p \twoheadrightarrow$
$\oplus\ \ell \twoheadrightarrow$
(a_s, t_s)
$\langle$**false**$\rangle$
$(\mathbf{1}, \emptyset)\;\}$

```
body;
```

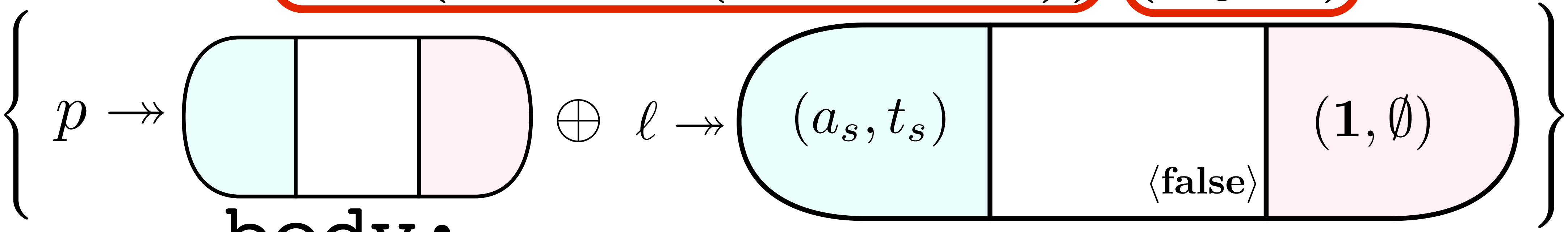
}



Concurroid spec

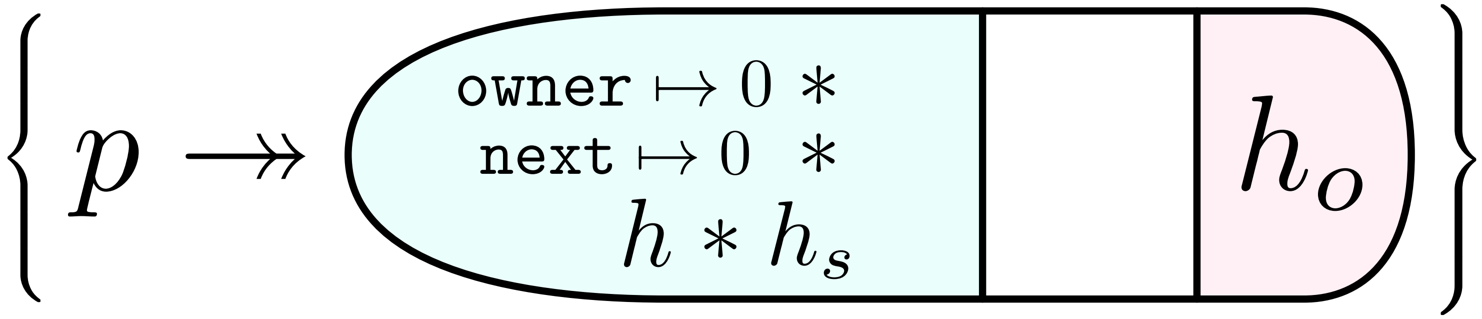
Initial "self" auxiliaries

$$\mathit{hide}\ \boxed{\mathit{coh}\,(\texttt{tlock}\ \ell(\texttt{owner},\texttt{next}))},\ \boxed{(a_s, \emptyset)}\ \{$$


$p \twoheadrightarrow$
$\oplus$
$\ell \twoheadrightarrow$
(a_s, t_s)
$\langle \textbf{false} \rangle$
$(\mathbf{1}, \emptyset)$

```
body;
```

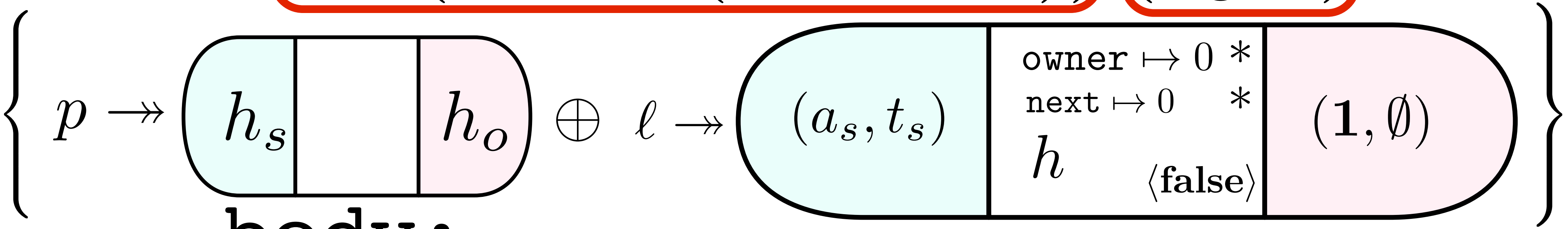
}



Concurroid spec

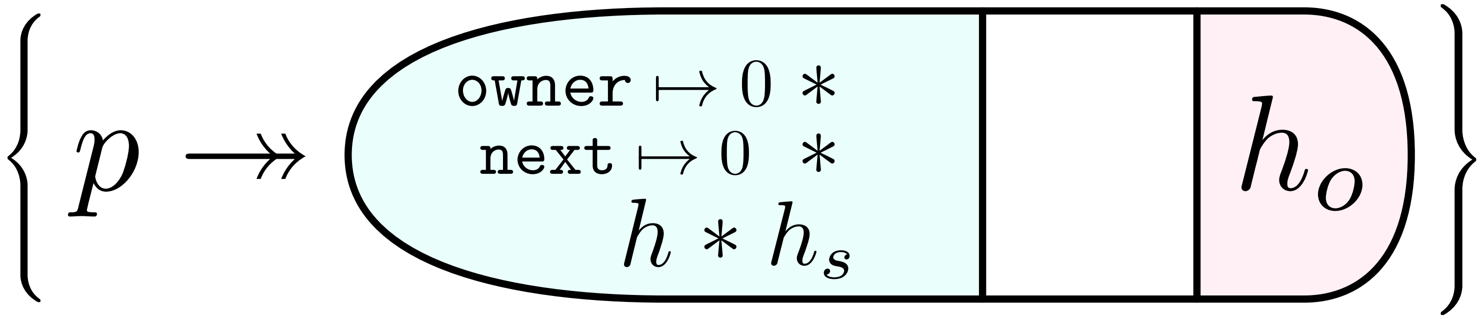
Initial “self” auxiliaries

$hide_{\boxed{coh\,(\texttt{tlock}\ \ell(\texttt{owner},\texttt{next}))},\ \boxed{(a_s, \emptyset)}}$ {


$\{ p \twoheadrightarrow$
h_s
h_o
$\oplus$
$\ell \twoheadrightarrow$
(a_s, t_s)
owner $\mapsto 0\ *$
next $\mapsto 0\ *$
h
$\langle \mathbf{false} \rangle$
$(\mathbf{1}, \emptyset) \}$

```
body;
```

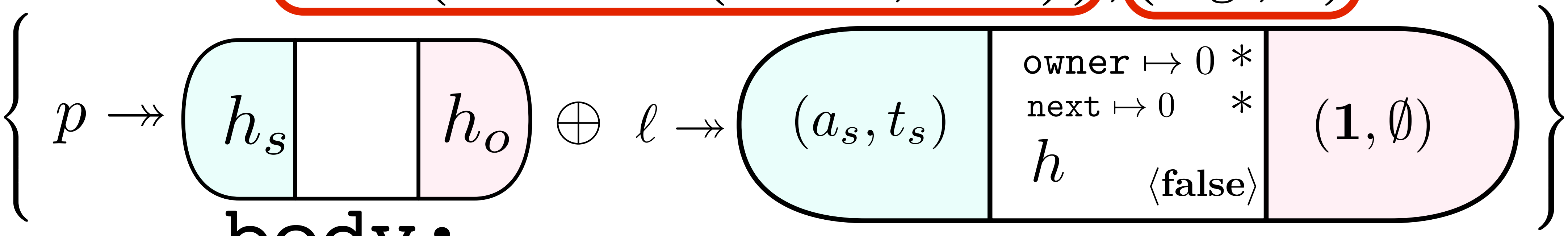
}



Concurroid spec

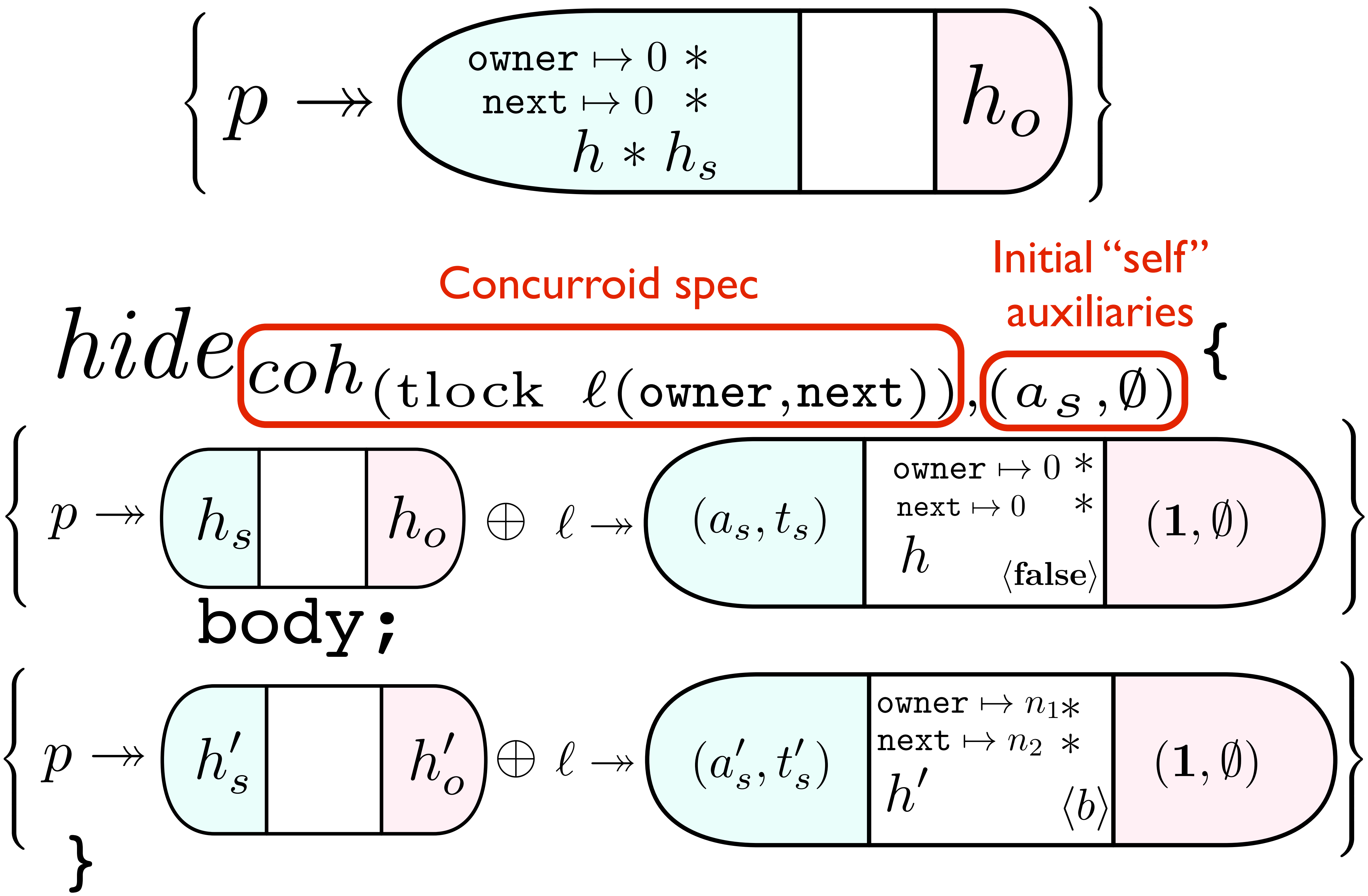
Initial "self" auxiliaries

$hide\ \boxed{coh\,(\texttt{tlock}\ \ \ell(\texttt{owner},\texttt{next}))},\boxed{(a_s,\emptyset)}$ {


$\{\, p \twoheadrightarrow$
h_s
h_o
$\oplus\ \ell \twoheadrightarrow$
(a_s, t_s)
owner $\mapsto 0\ *$
next $\mapsto 0\ *$
h
$\langle \textbf{false} \rangle$
$(\mathbf{1}, \emptyset)$
$\}$

`body;`

}



$\{ p \twoheadrightarrow$ owner $\mapsto 0 \, *$ next $\mapsto 0 \, *$ $h * h_s$ | | $h_o \}$

Concurroid spec

Initial “self” auxiliaries

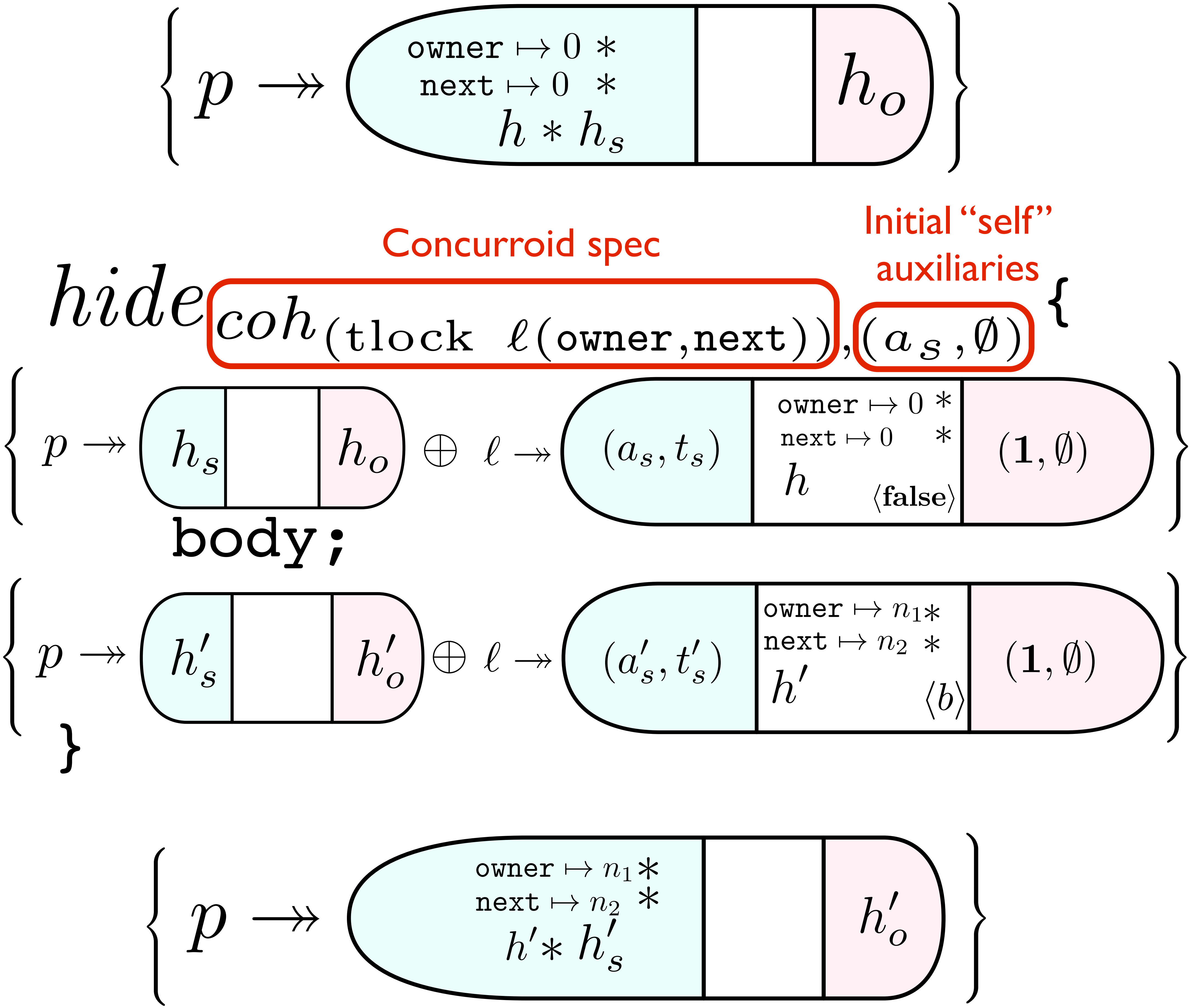
$hide_{coh(\texttt{tlock}\ \ell(\texttt{owner},\texttt{next})),(a_s,\emptyset)}$ {

$\{ p \twoheadrightarrow$ h_s | | h_o $\oplus\ \ell \twoheadrightarrow$ (a_s, t_s) | owner $\mapsto 0 \, *$ next $\mapsto 0 \, *$ h $\langle \textbf{false} \rangle$ | $(\mathbf{1}, \emptyset) \}$

body;

$\{ p \twoheadrightarrow$ h'_s | | h'_o $\oplus\ \ell \twoheadrightarrow$ (a'_s, t'_s) | owner $\mapsto n_1 *$ next $\mapsto n_2 \, *$ h' $\langle b \rangle$ | $(\mathbf{1}, \emptyset) \}$

}



$\{ p \twoheadrightarrow$ owner $\mapsto 0 *$ next $\mapsto 0 *$ $h * h_s$ | | $h_o \}$

Concurroid spec

Initial “self” auxiliaries

$hide \; coh$(tlock ℓ(owner,next)),$(a_s, \emptyset)$ {

$\{ p \twoheadrightarrow h_s \mid \mid h_o \oplus \ell \twoheadrightarrow (a_s, t_s) \mid$ owner $\mapsto 0 *$ next $\mapsto 0 *$ h $\langle \mathbf{false} \rangle$ $\mid (\mathbf{1}, \emptyset) \}$

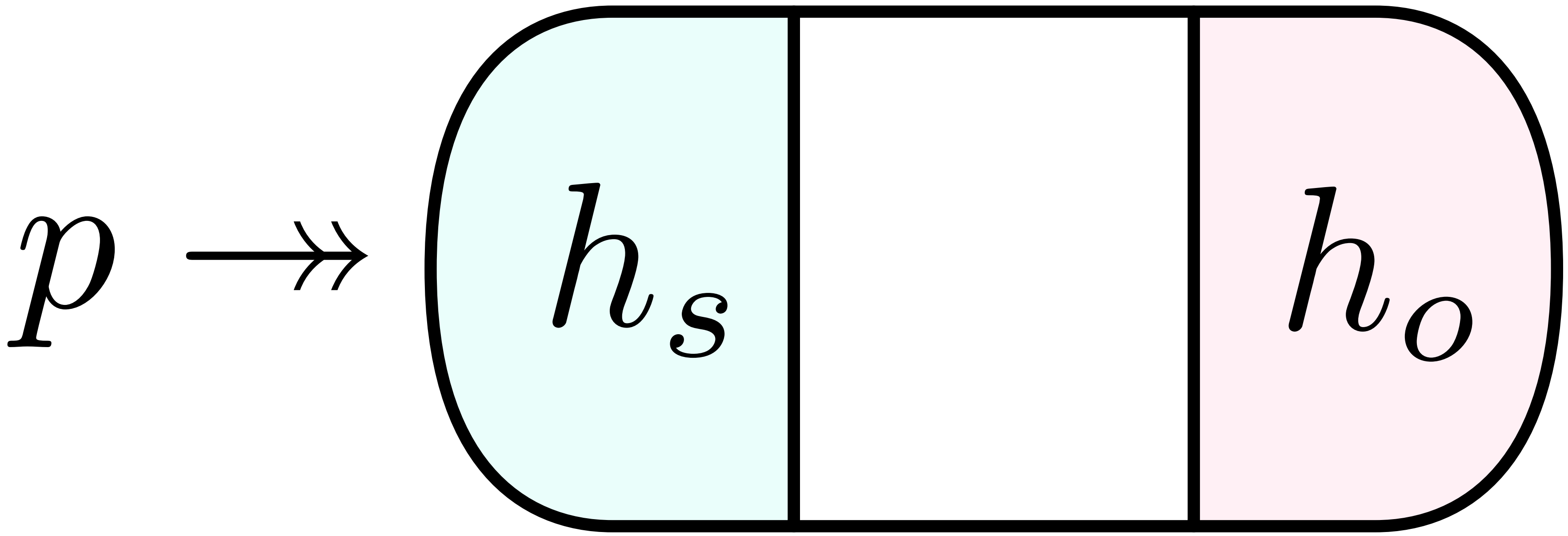
body;

$\{ p \twoheadrightarrow h'_s \mid \mid h'_o \oplus \ell \twoheadrightarrow (a'_s, t'_s) \mid$ owner $\mapsto n_1 *$ next $\mapsto n_2 *$ h' $\langle b \rangle$ $\mid (\mathbf{1}, \emptyset) \}$

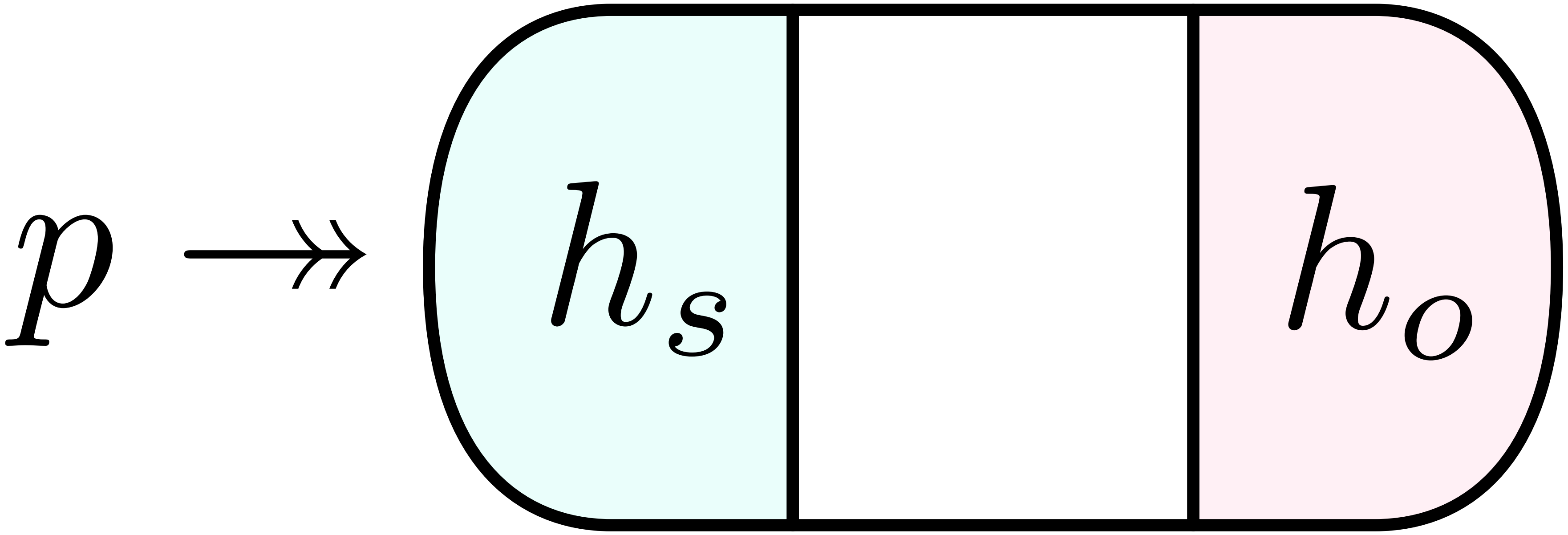
}

$\{ p \twoheadrightarrow$ owner $\mapsto n_1 *$ next $\mapsto n_2 *$ $h' * h'_s$ | | $h'_o \}$

Only One Basic Concurroid



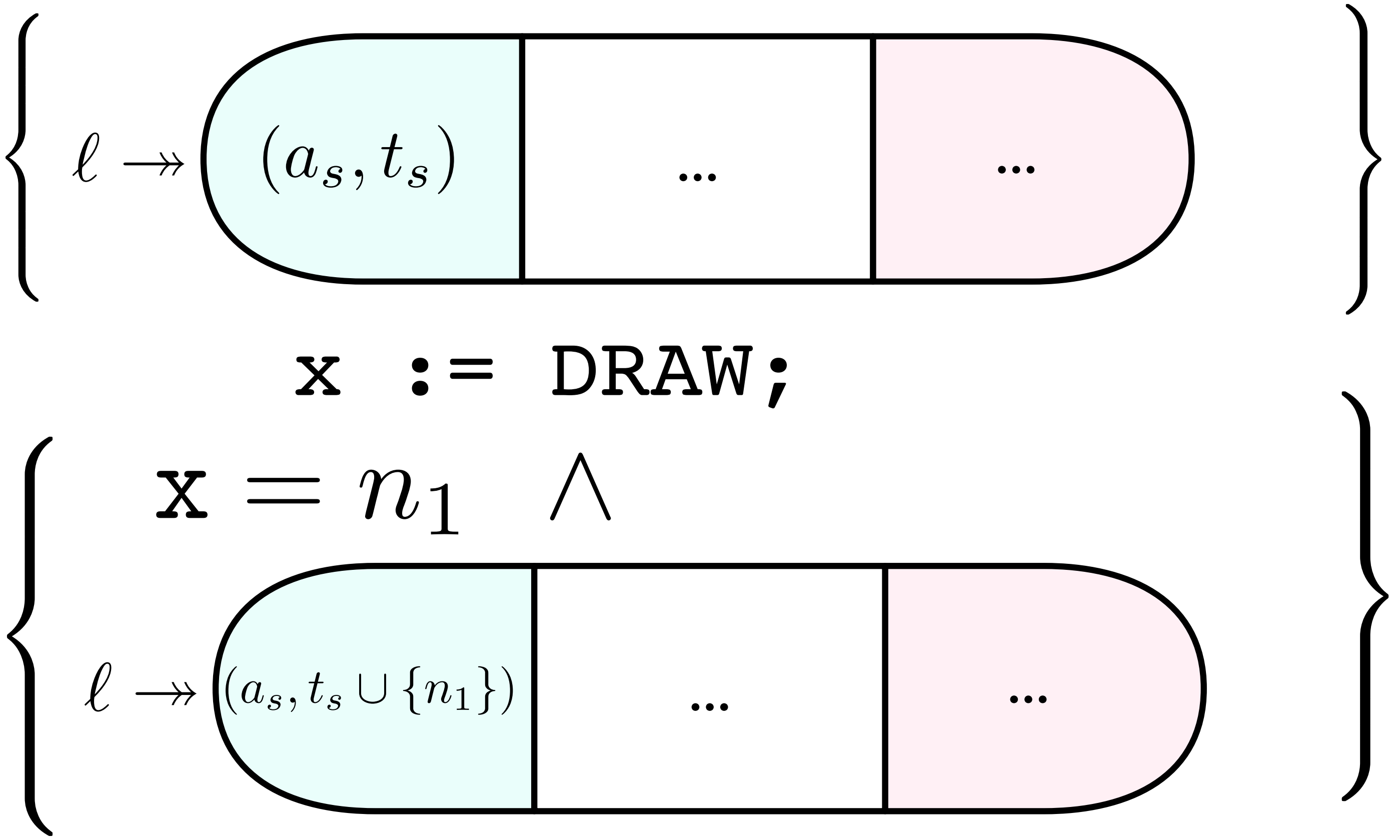
Only One Basic Concurroid



A concurroid of “*private heaps*”.

Framing with respect to concurroids.

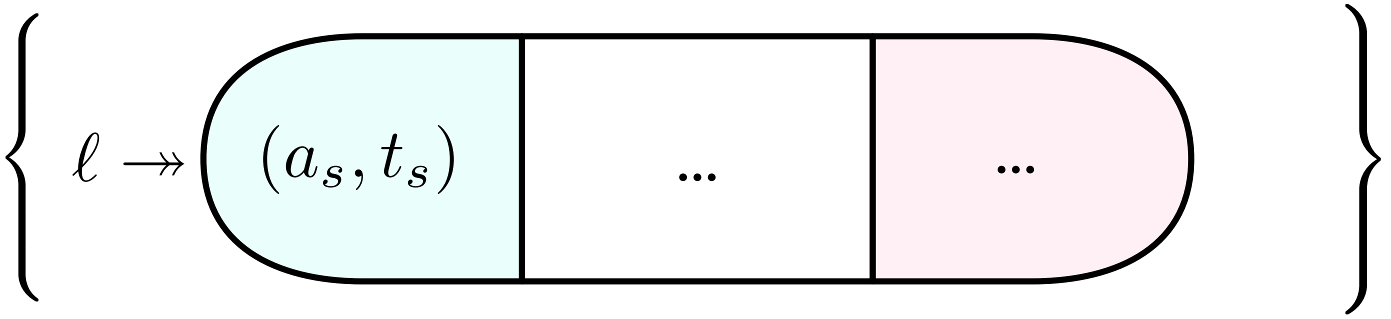
```
x := DRAW;
```



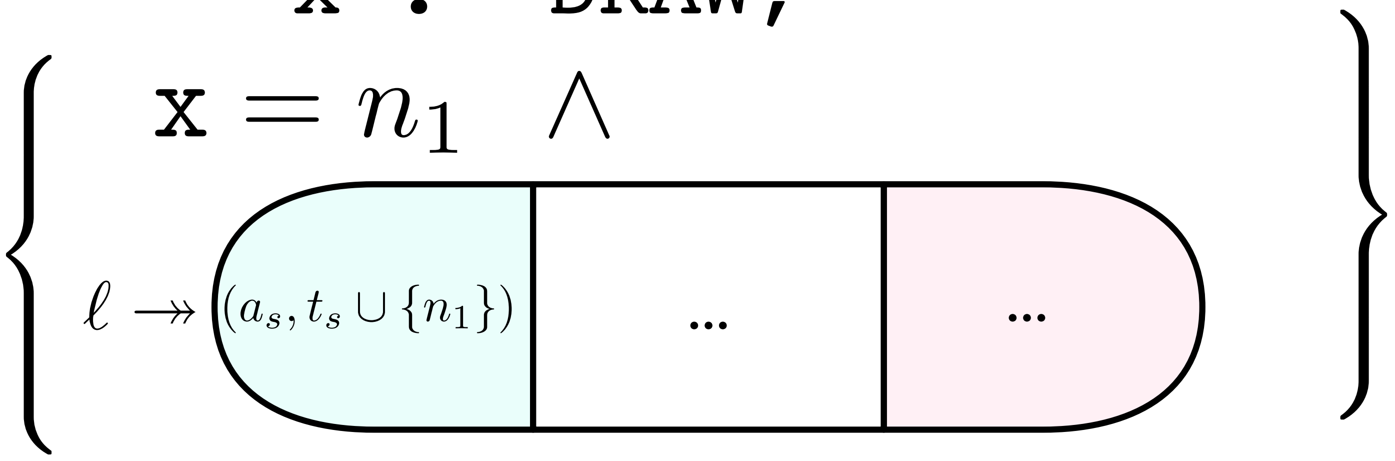
$\left\{ \ell \twoheadrightarrow (a_s, t_s) \quad ... \quad ... \right\}$

```
x := DRAW;
```

$\left\{ \mathtt{x} = n_1 \wedge \ell \twoheadrightarrow (a_s, t_s \cup \{n_1\}) \quad ... \quad ... \right\}$

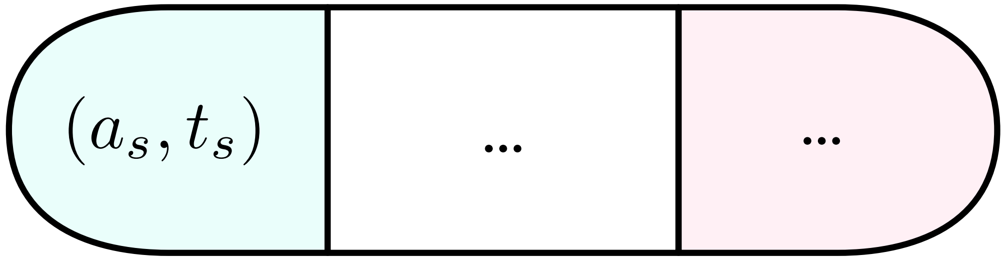


```
x := DRAW;
```


$\mathtt{x} = n_1 \quad \wedge$

```
lock = {
```

$\Big\{$ $\ell \twoheadrightarrow$

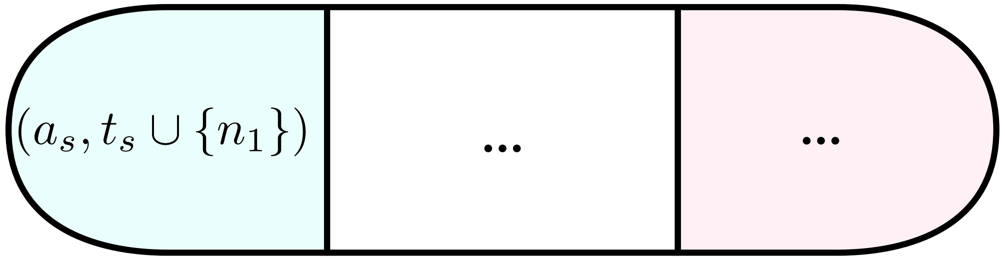


$\Big\}$

```
      x := DRAW;
```

$\Big\{$ $\mathtt{x} = n_1 \;\wedge$

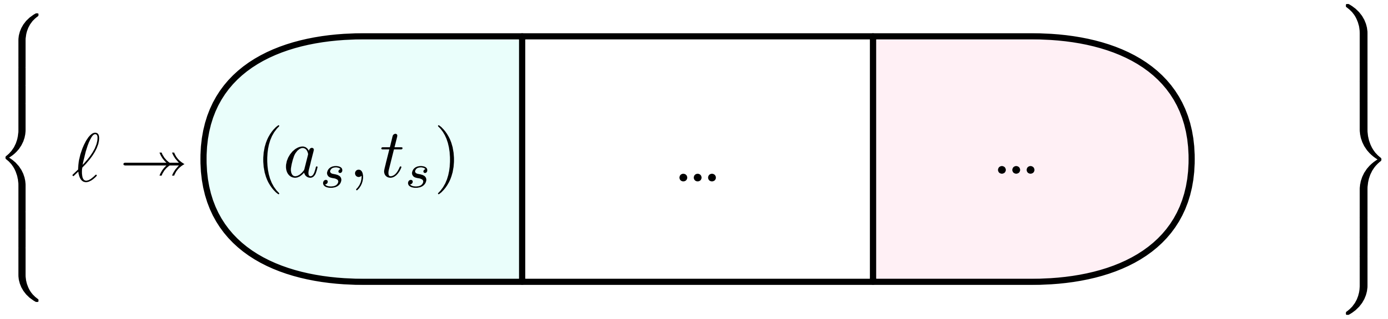
$\ell \twoheadrightarrow$



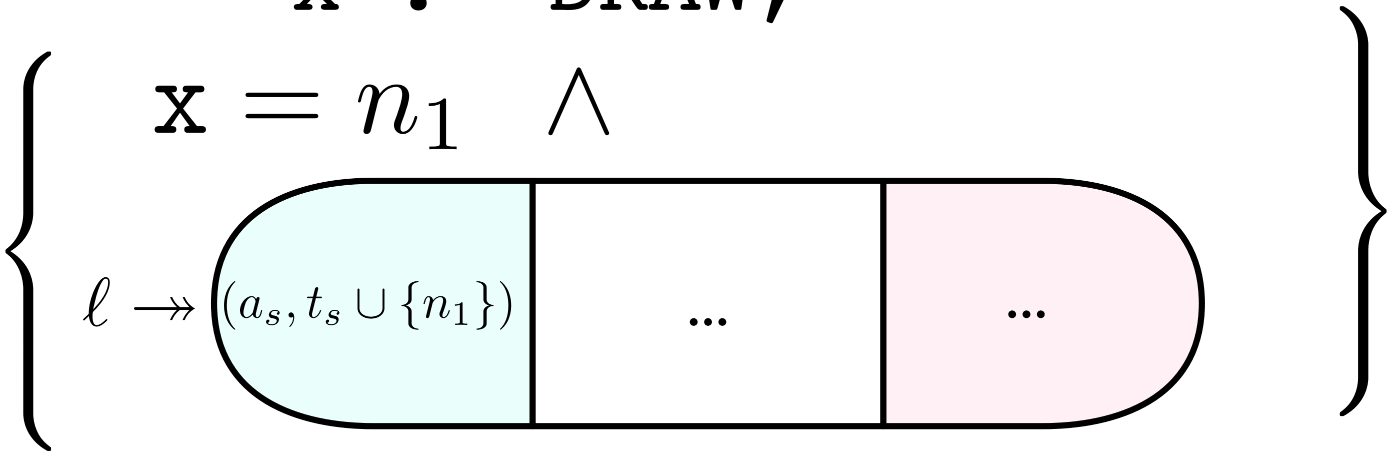
$\Big\}$

```
      while (!TRY(x)) SKIP;

}
```

```
lock = {
```

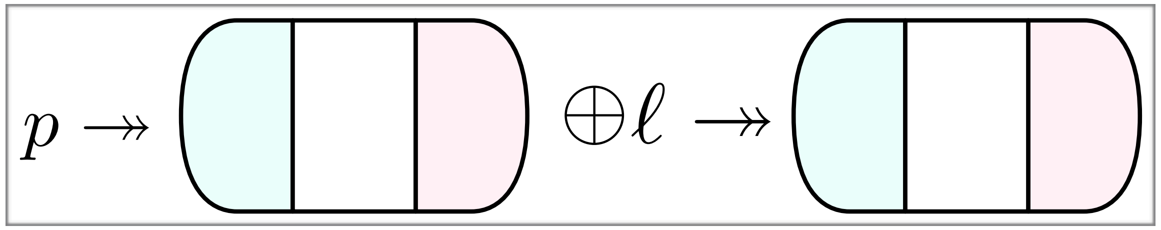


```
        x := DRAW;
```


$\mathtt{x} = n_1 \quad \wedge$
$\ell \twoheadrightarrow$
$(a_s, t_s \cup \{n_1\})$
...
...

```
        while (!TRY(x)) SKIP;
```

Defined in



```
}
```

Context Weakening!

Injection Rule

$$\frac{\{p\}\, C\, \{q\} \;@\; U \qquad r \text{ stable under } V}{\{p * r\}\, \mathsf{inject}_V\, C\, \{q * r\} \;@\; U \mathbin{\blacktriangleright\!\!\blacktriangleleft} V}\ \textsc{Inject}$$

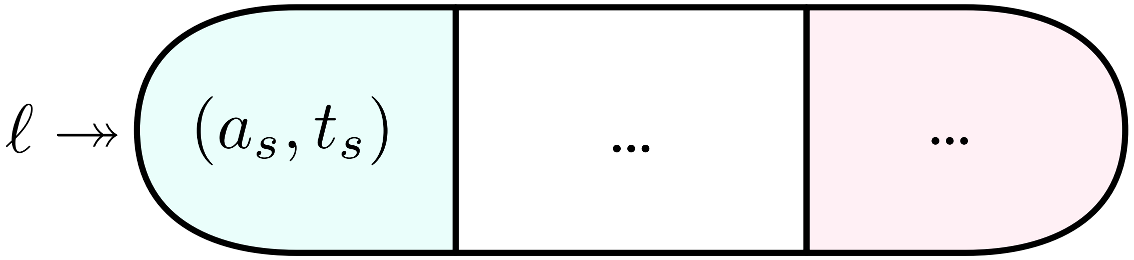
where $\mathbin{\blacktriangleright\!\!\blacktriangleleft} = \bowtie, \rtimes, \ltimes, \times \ldots$

Injection Rule

$$\frac{\{p\}\, C\, \{q\} \ @\ U \qquad \boxed{r \text{ stable under } V}}{\{p * \boxed{r}\}\ \mathsf{inject}_V\ C\ \{q * \boxed{r}\} \ @\ U \mathbin{\blacktriangleright\!\!\blacktriangleleft} V}\ \textsc{Inject}$$

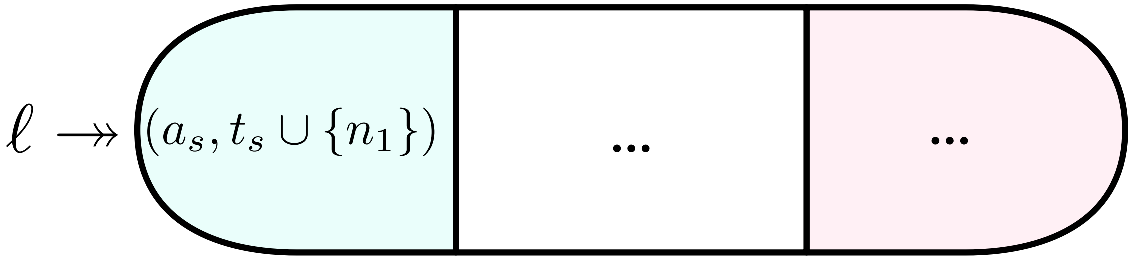
where $\blacktriangleright\!\!\blacktriangleleft \; = \; \bowtie, \rtimes, \ltimes, \times \ldots$

```
lock = {
```



```
x := DRAW ;
```

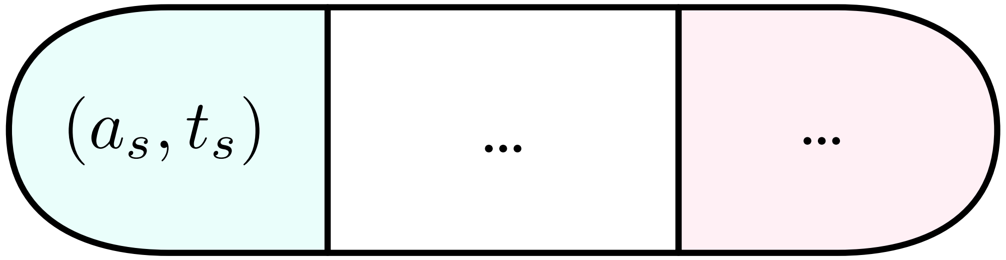
$\mathtt{x} = n_1 \;\; \wedge$



```
while (!TRY(x)) SKIP;
```

```
}
```

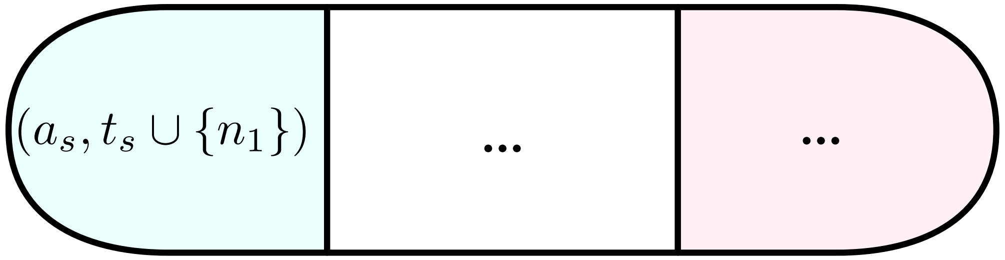
```
lock = {
```

$\Big\{$ $\ell \twoheadrightarrow$ 

$\Big\}$

x := $inject_p$(DRAW);

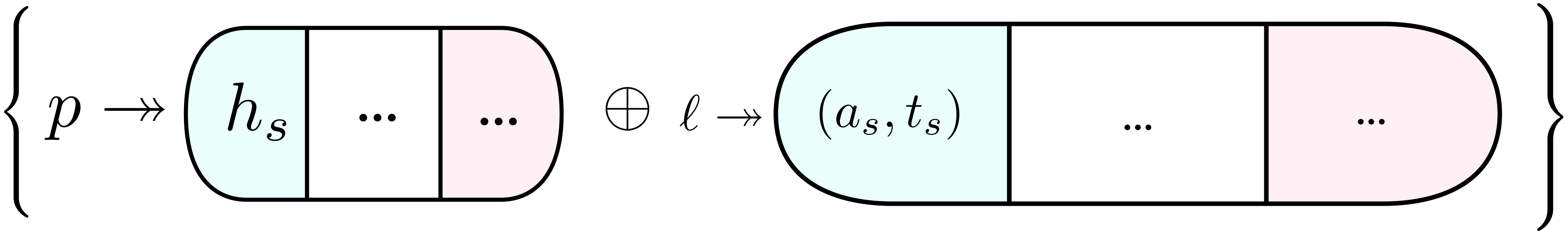
$\Big\{$ $\mathtt{x} = n_1 \;\; \wedge$

$\ell \twoheadrightarrow$ 

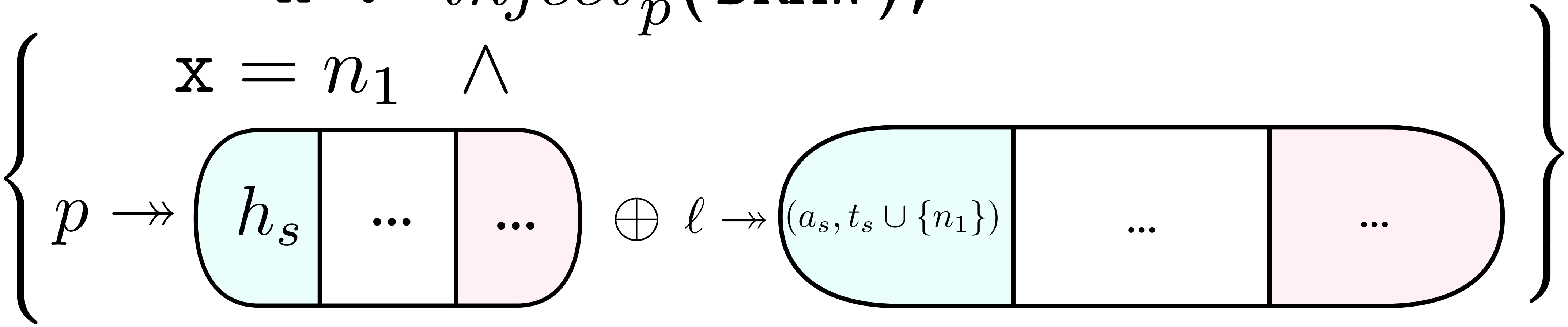
$\Big\}$

```
        while (!TRY(x)) SKIP;

}
```

```
lock = {
```

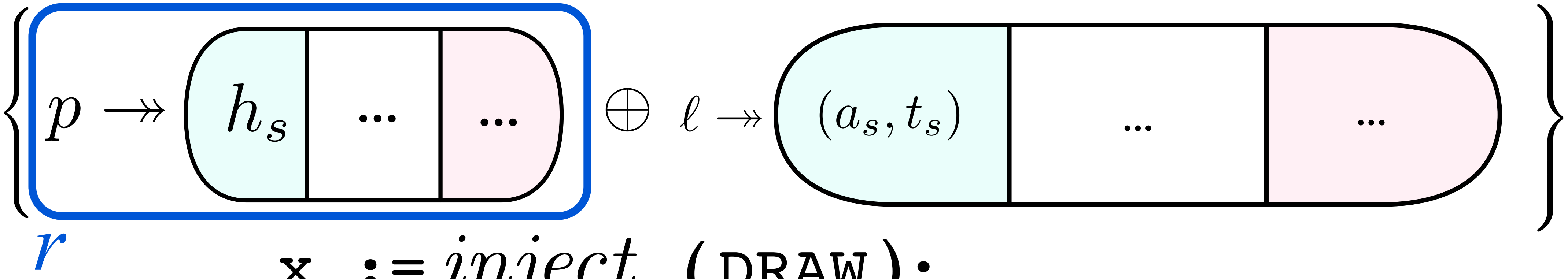


x := $inject_p$(DRAW);

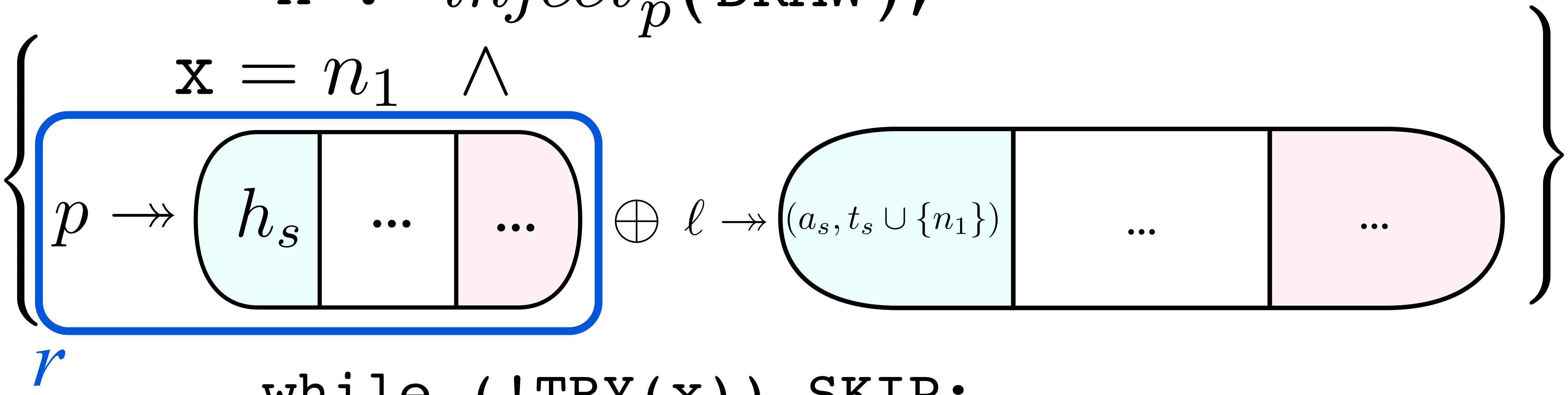

$\mathtt{x} = n_1 \quad \wedge$
$p \twoheadrightarrow$
h_s
...
...
$\oplus$
$\ell \twoheadrightarrow$
$(a_s, t_s \cup \{n_1\})$
...
...

```
        while (!TRY(x)) SKIP;

}
```

```
lock = {
```



```
        x := inject_p(DRAW);
```



$\mathtt{x} = n_1 \quad \wedge$
$p \twoheadrightarrow$
h_s
...
...
$\oplus$
$\ell \twoheadrightarrow$
$(a_s, t_s \cup \{n_1\})$
...
...
r

```
        while (!TRY(x)) SKIP;


}
```

On the role of hiding

- *Subjective state* allows one to give a lower bound to the joint contribution:

  "I know what is my contribution."

- *Hiding* (or *scoping*) allows one to provide an upper bound for the contribution:

  "When everyone is done, we can the auxiliaries are summed up."

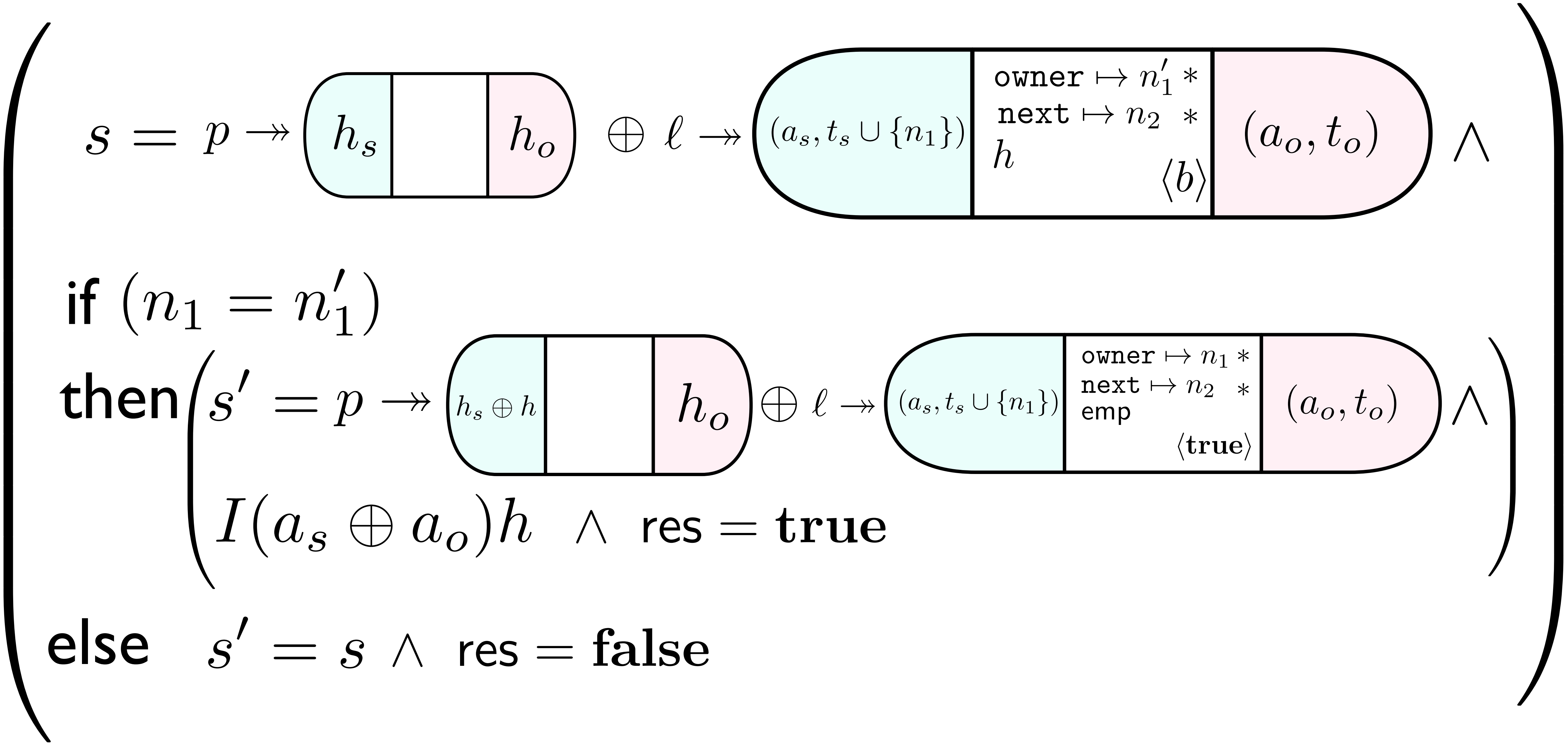
`TRY(`n_1`)` Action Specification

$\texttt{TRY}(\texttt{n}_1)$ Action Specification

$\texttt{TRY}(n_1)(s, s', \mathsf{res}) \triangleq$

$\texttt{TRY(n}_1\texttt{)}$ Action Specification

$$\texttt{TRY}(n_1)(s, s', \mathsf{res}) \triangleq$$



$$s = p \twoheadrightarrow \boxed{h_s \mid \quad \mid h_o} \oplus \ell \twoheadrightarrow \boxed{(a_s, t_s \cup \{n_1\}) \;\Big|\; \begin{array}{l} \texttt{owner} \mapsto n_1' \, * \\ \texttt{next} \mapsto n_2 \;\; * \\ h \qquad\qquad \langle b \rangle \end{array} \;\Big|\; (a_o, t_o)} \;\wedge$$

$$\textsf{if } (n_1 = n_1')$$

$$\textsf{then} \left(\begin{array}{l} s' = p \twoheadrightarrow \boxed{h_s \oplus h \mid \quad \mid h_o} \oplus \ell \twoheadrightarrow \boxed{(a_s, t_s \cup \{n_1\}) \;\Big|\; \begin{array}{l} \texttt{owner} \mapsto n_1 \, * \\ \texttt{next} \mapsto n_2 \;\; * \\ \mathsf{emp} \qquad \langle \mathbf{true} \rangle \end{array} \;\Big|\; (a_o, t_o)} \;\wedge \\ I(a_s \oplus a_o)h \;\wedge\; \mathsf{res} = \mathbf{true} \end{array} \right)$$

$$\textsf{else} \quad s' = s \;\wedge\; \mathsf{res} = \mathbf{false}$$

TRY(n₁) Action Specification

$\mathtt{TRY}(n_1)(s, s', \mathsf{res}) \triangleq$

$$s = p \twoheadrightarrow \boxed{h_s \;\vert\; \quad \;\vert\; h_o} \oplus \ell \twoheadrightarrow \boxed{(a_s, t_s \cup \{n_1\}) \;\vert\; \begin{array}{l} \mathtt{owner} \mapsto n_1' * \\ \mathtt{next} \mapsto n_2 \;\; * \\ h \\ \hfill \langle b \rangle \end{array} \;\vert\; (a_o, t_o)} \wedge$$

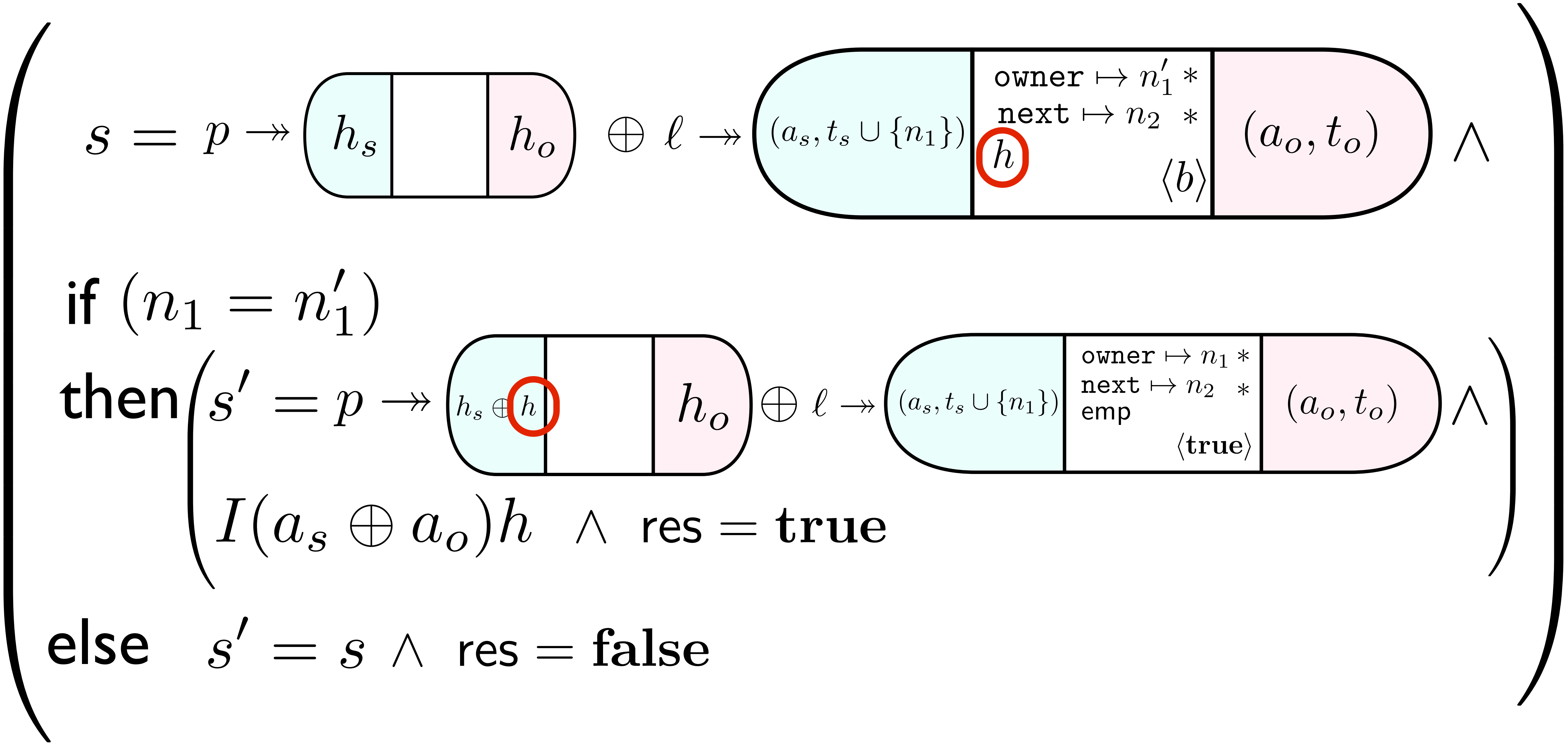
if $(n_1 = n_1')$

then $$\left(s' = p \twoheadrightarrow \boxed{h_s \oplus h \;\vert\; \quad \;\vert\; h_o} \oplus \ell \twoheadrightarrow \boxed{(a_s, t_s \cup \{n_1\}) \;\vert\; \begin{array}{l} \mathtt{owner} \mapsto n_1 * \\ \mathtt{next} \mapsto n_2 \;\; * \\ \mathsf{emp} \\ \hfill \langle \mathbf{true} \rangle \end{array} \;\vert\; (a_o, t_o)} \wedge \right.$$
$$\left. I(a_s \oplus a_o)h \;\wedge\; \mathsf{res} = \mathbf{true} \right)$$

else $s' = s \wedge \mathsf{res} = \mathbf{false}$

TRY(n₁) Action Specification

$\mathtt{TRY}(n_1)(s, s', \mathsf{res}) \triangleq$



$s = p \twoheadrightarrow$ h_s | | h_o $\oplus\ \ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$ | $\mathtt{owner} \mapsto n_1' \ *$ $\mathtt{next} \mapsto n_2 \ *$ h $\langle b \rangle$ | (a_o, t_o) $\wedge$

if $(n_1 = n_1')$

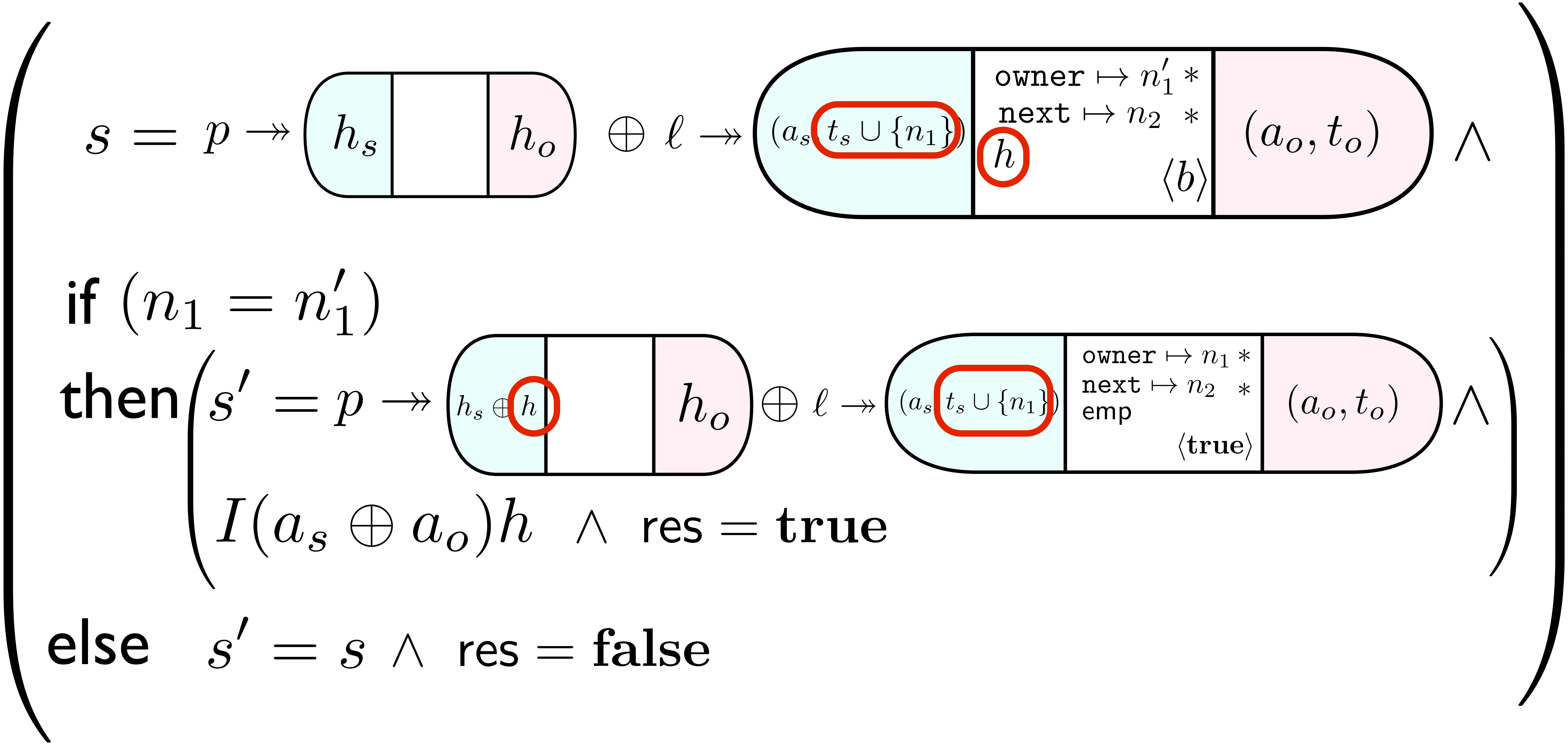
then $s' = p \twoheadrightarrow$ $h_s \oplus h$ | | h_o $\oplus\ \ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$ | $\mathtt{owner} \mapsto n_1 \ *$ $\mathtt{next} \mapsto n_2 \ *$ emp $\langle \mathbf{true} \rangle$ | (a_o, t_o) $\wedge$

$I(a_s \oplus a_o)h \ \wedge\ \mathsf{res} = \mathbf{true}$

else $s' = s \ \wedge\ \mathsf{res} = \mathbf{false}$

TRY(n₁) Action Specification

$\mathtt{TRY}(n_1)(s, s', \mathsf{res}) \triangleq$



$s = p \twoheadrightarrow$ h_s | | h_o $\oplus\ \ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$ | $\mathtt{owner} \mapsto n_1' \ast$ $\mathtt{next} \mapsto n_2\ \ast$ h $\langle b \rangle$ | (a_o, t_o) $\wedge$

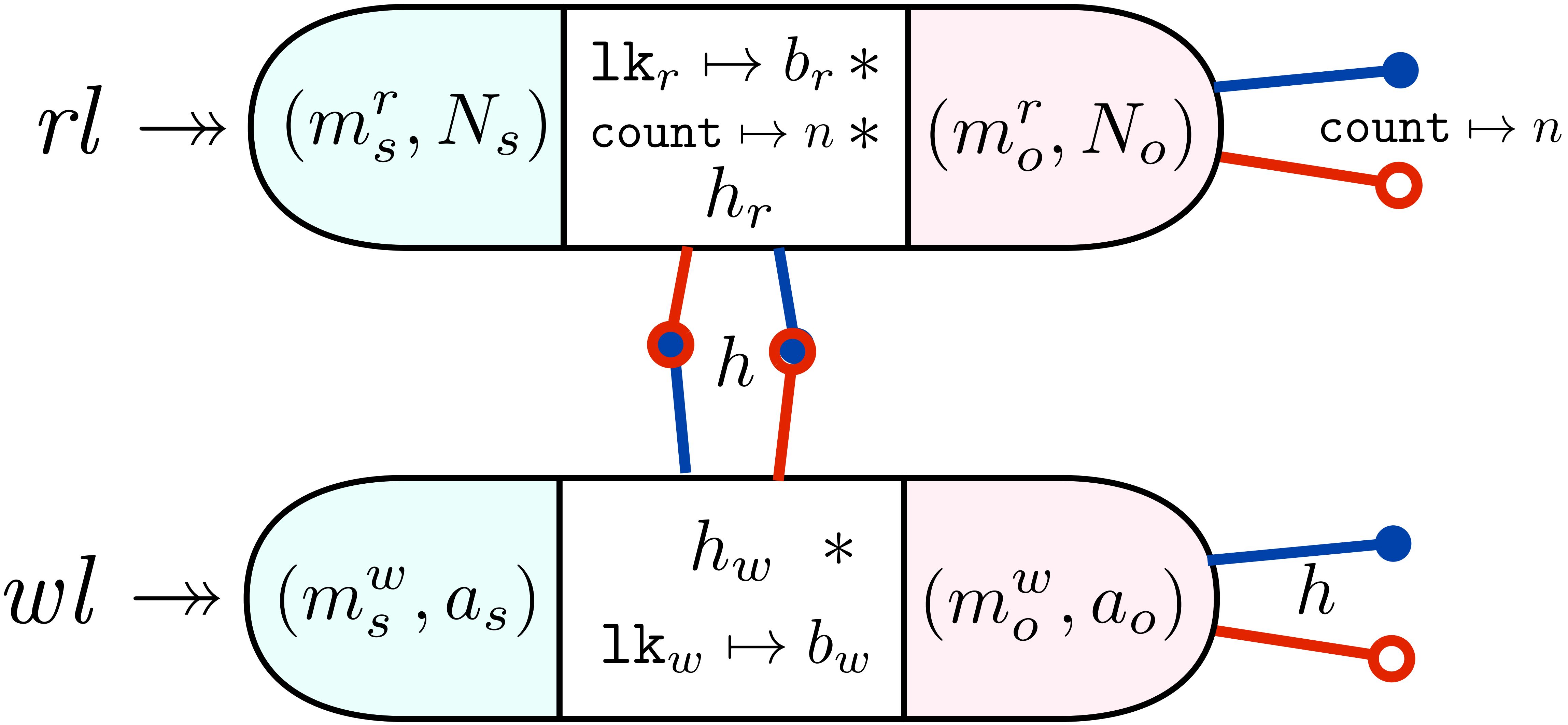
if $(n_1 = n_1')$

then $s' = p \twoheadrightarrow$ $h_s \oplus h$ | | h_o $\oplus\ \ell \twoheadrightarrow$ $(a_s, t_s \cup \{n_1\})$ | $\mathtt{owner} \mapsto n_1 \ast$ $\mathtt{next} \mapsto n_2\ \ast$ emp $\langle \mathbf{true} \rangle$ | (a_o, t_o) $\wedge$

$I(a_s \oplus a_o)h \ \wedge\ \mathsf{res} = \mathbf{true}$

else $s' = s\ \wedge\ \mathsf{res} = \mathbf{false}$

Readers-Writers



$rl \twoheadrightarrow$

(m^r_s, N_s)

$\texttt{lk}_r \mapsto b_r *$

$\texttt{count} \mapsto n *$

h_r

(m^r_o, N_o)

$\texttt{count} \mapsto n$

h

$wl \twoheadrightarrow$

(m^w_s, a_s)

$h_w \quad *$

$\texttt{lk}_w \mapsto b_w$

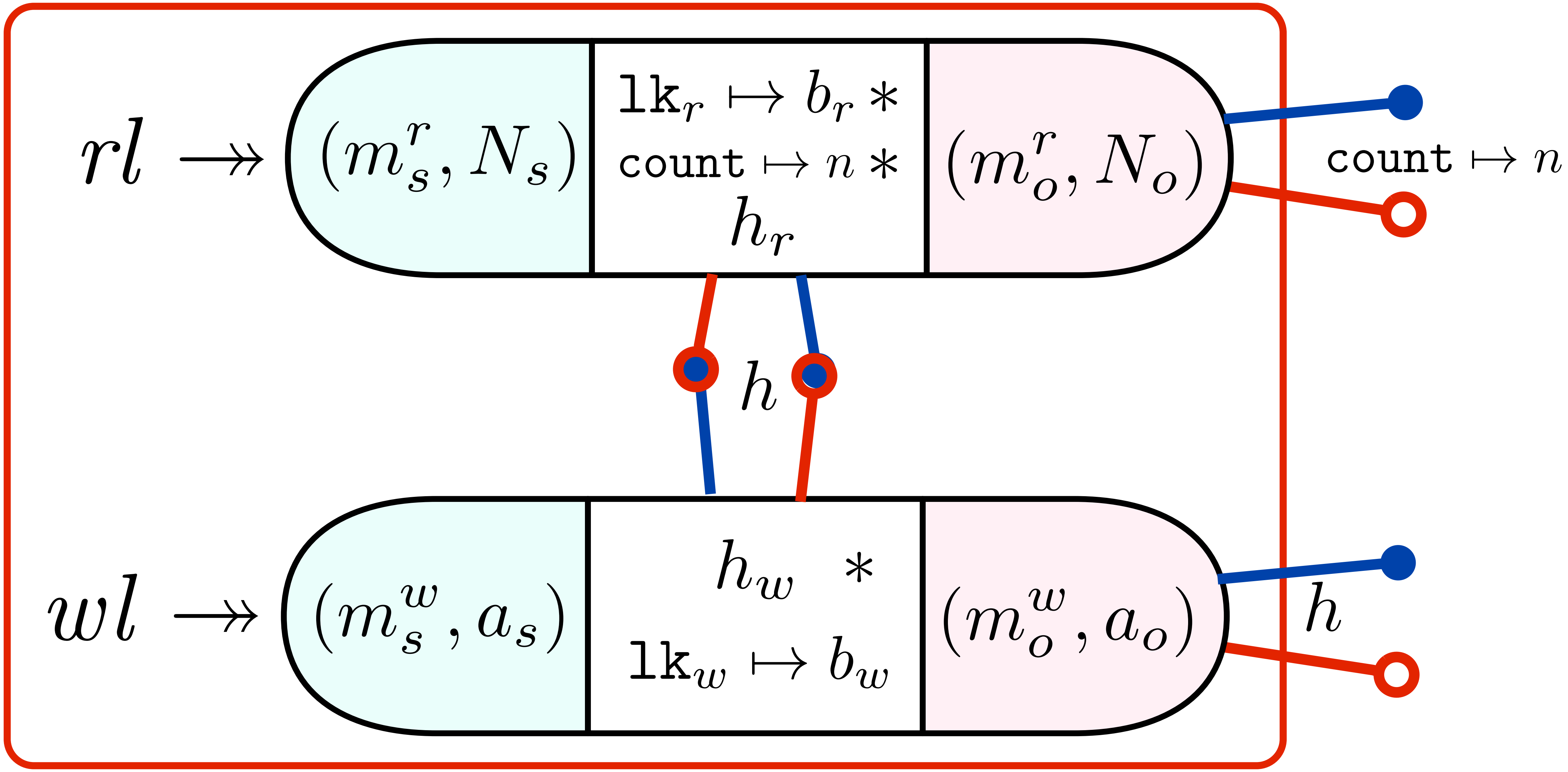
(m^w_o, a_o)

h

$$I_r(N_s \oplus N_o, h_r) \triangleq (N_s \oplus N_o = n) \ \wedge\ (N_s \oplus N_o = 0 \implies h_r = \mathsf{emp})$$

$$I_w(a_s \oplus a_o, h_w) \triangleq \ldots$$

Readers-Writers



$$I_r(N_s \oplus N_o, h_r) \triangleq (N_s \oplus N_o = n) \;\wedge\; (N_s \oplus N_o = 0 \implies h_r = \mathsf{emp})$$

$$I_w(a_s \oplus a_o, h_w) \triangleq \ldots$$