Communicating State Transition Systems for Fine-Grained Concurrent Resources

Aleksandar Nanevski   Ruy Ley-Wild   Ilya Sergey   Germán Delbianco

institute IMDEA software   LOGICBLOX®

HOPE 2013

Reasoning about shared-memory concurrency

How to model shared-memory concurrency

Two views at shared-memory concurrency

Coarse-Grained Concurrency

Locks (or CCRs) are given
as a primitive for synchronization.

Fine-Grained Concurrency

Synchronization is implemented via atomic *Read-Modify-Write* commands.

Two powerful tools for reasoning

Concurrent Separation Logic

O'Hearn [CONCUR'07], Brookes [CONCUR'04]

Rely Guarantee Reasoning

Jones [TOPLAS'83]

The essence of CSL

The essence of CSL

- The protocol for interference is fixed:
  Conditional Critical Regions with Resource Invariants

The essence of CSL

- The protocol for interference is <u>fixed</u>:
  Conditional Critical Regions with Resource Invariants
- *Interference* doesn't matter: CCR handle it

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

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

“resource creation”

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

$$\frac{\mathit{\Gamma} \vdash \{p_1\}\ c_1\ \{q_1\} \qquad \mathit{\Gamma} \vdash \{p_2\}\ c_2\ \{q_2\}}{\mathit{\Gamma} \vdash \{p_1 * p_2\}\ c_1 \parallel c_2\ \{q_1 * q_2\}}\ \textsc{ParCSL}$$

All interference
is handled here

$$\frac{\boxed{\Gamma} \vdash \{p_1\}\ c_1\ \{q_1\} \qquad \boxed{\Gamma} \vdash \{p_2\}\ c_2\ \{q_2\}}{\boxed{\Gamma} \vdash \{p_1 * p_2\}\ c_1 \parallel c_2\ \{q_1 * q_2\}}\ \textsc{ParCSL}$$

The essence of R/G

The essence of R/G

- One can define <u>arbitrary protocols</u>
  for process interference via *Guarantee relation.*

The essence of R/G

- One can define arbitrary protocols
  for process interference via *Guarantee relation.*
- Interference matters!
  Atomic operations should be given
  specifications stable wrt *Rely relation.*

“Forking/shuffling” parallel composition

$$\frac{R \vee G_2, G_1 \vdash \{p\}\ c_1\ \{q_1\} \qquad R \vee G_1, G_2 \vdash \{p\}\ c_2\ \{q_2\}}{R, G_1 \vee G_2 \vdash \{p\}\ c_1 \parallel c_2\ \{q_1 \wedge q_2\}}\ \textsc{ParRG}$$

"Forking/shuffling" parallel composition

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

“Forking/shuffling” parallel composition

$$\frac{R \vee G_2, G_1 \vdash \{p\}\; c_1\; \{q_1\} \qquad R \vee G_1, G_2 \vdash \{p\}\; c_2\; \{q_2\}}{R, G_1 \vee G_2 \vdash \{p\}\; c_1 \parallel c_2\; \{q_1 \wedge q_2\}}\ \textsc{ParRG}$$

Taking the best of two worlds

Our Approach

Fine-Grained Resources

Resources

Fine-Grained

Resources

State Invariants

Fine-Grained

Resources

Fine-Grained

State Invariants

Transitions

Resources

Fine-Grained

Composition

Forking/shuffling

State Invariants

Transitions

Resources

Fine-Grained

Composition

Forking/shuffling

State Invariants

Transitions

Communication

Resources	State Invariants
Fine-Grained	Transitions
Composition	Communication
Forking/shuffling	Subjectivity

State Transition Systems

Communication

Subjectivity

State Transition Systems

Communication

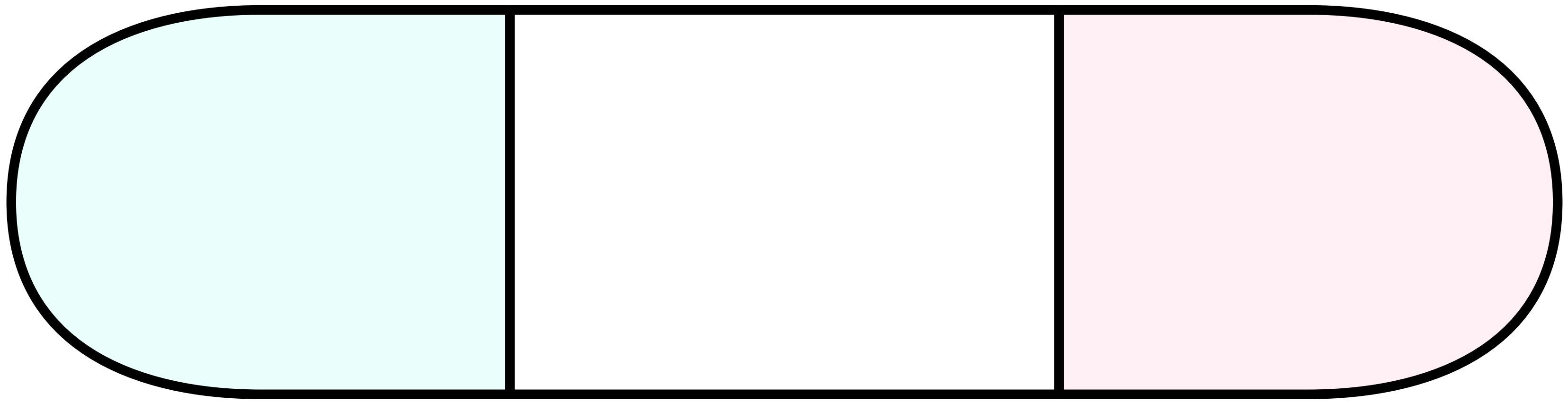
$\boxed{\text{Subjectivity}}$

(Ley-Wild and Nanevski, POPL 2013)

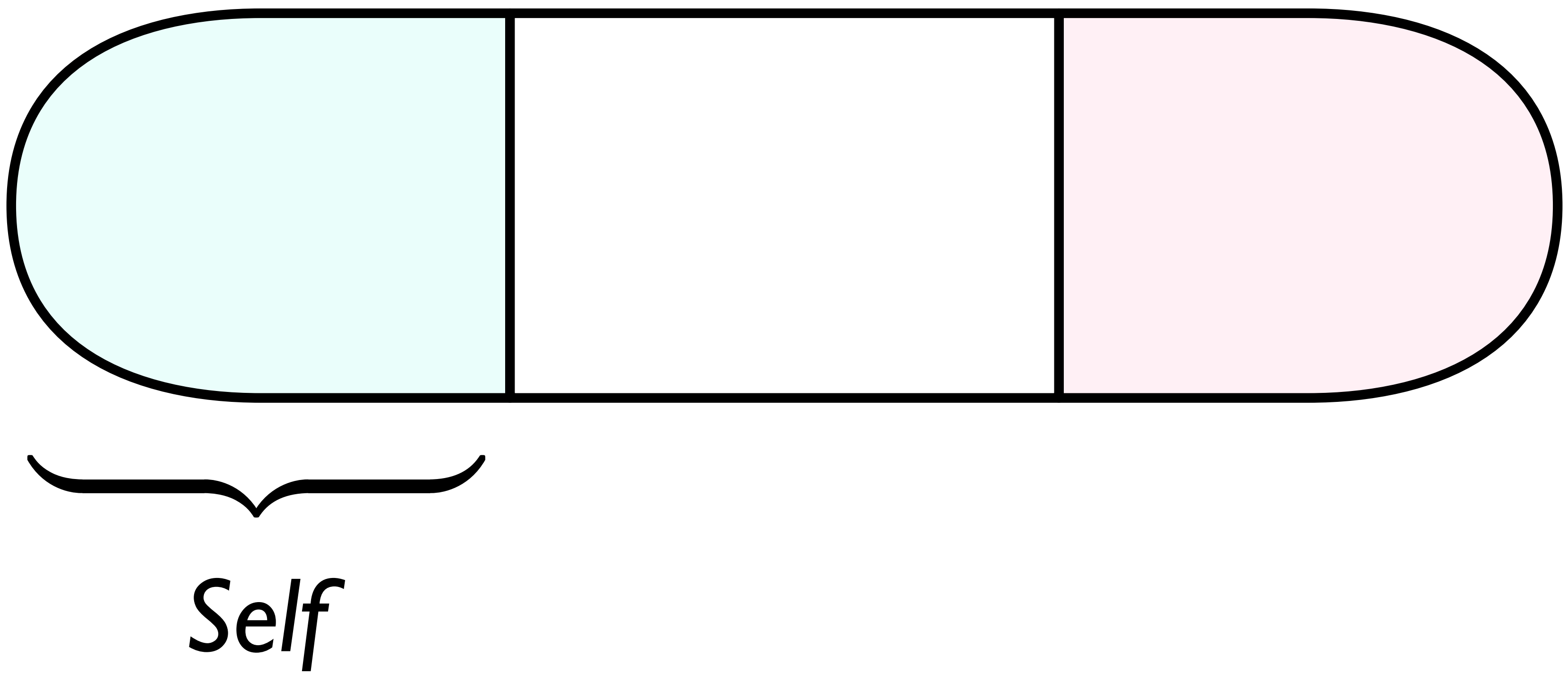
Subjective Communicating State-Transition Systems

Concurroids

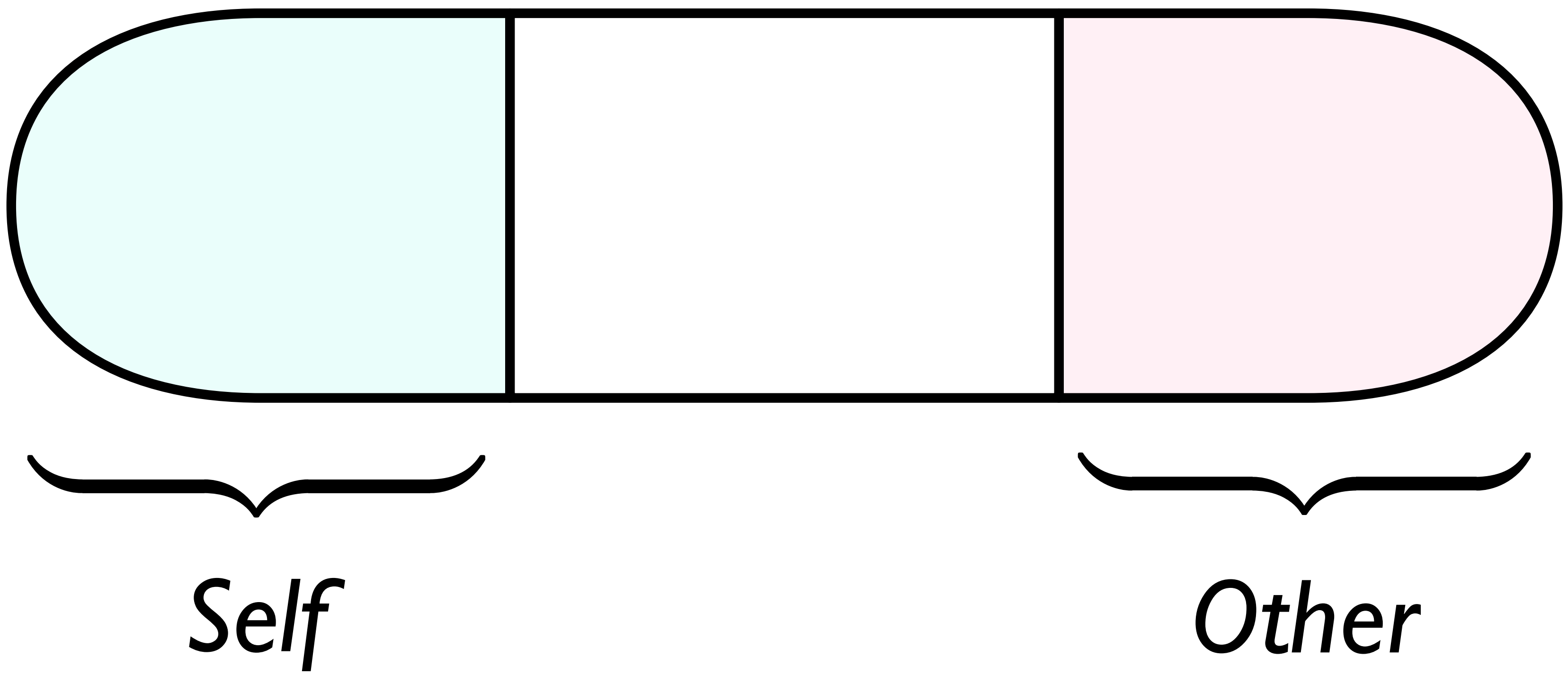
Concurroid States



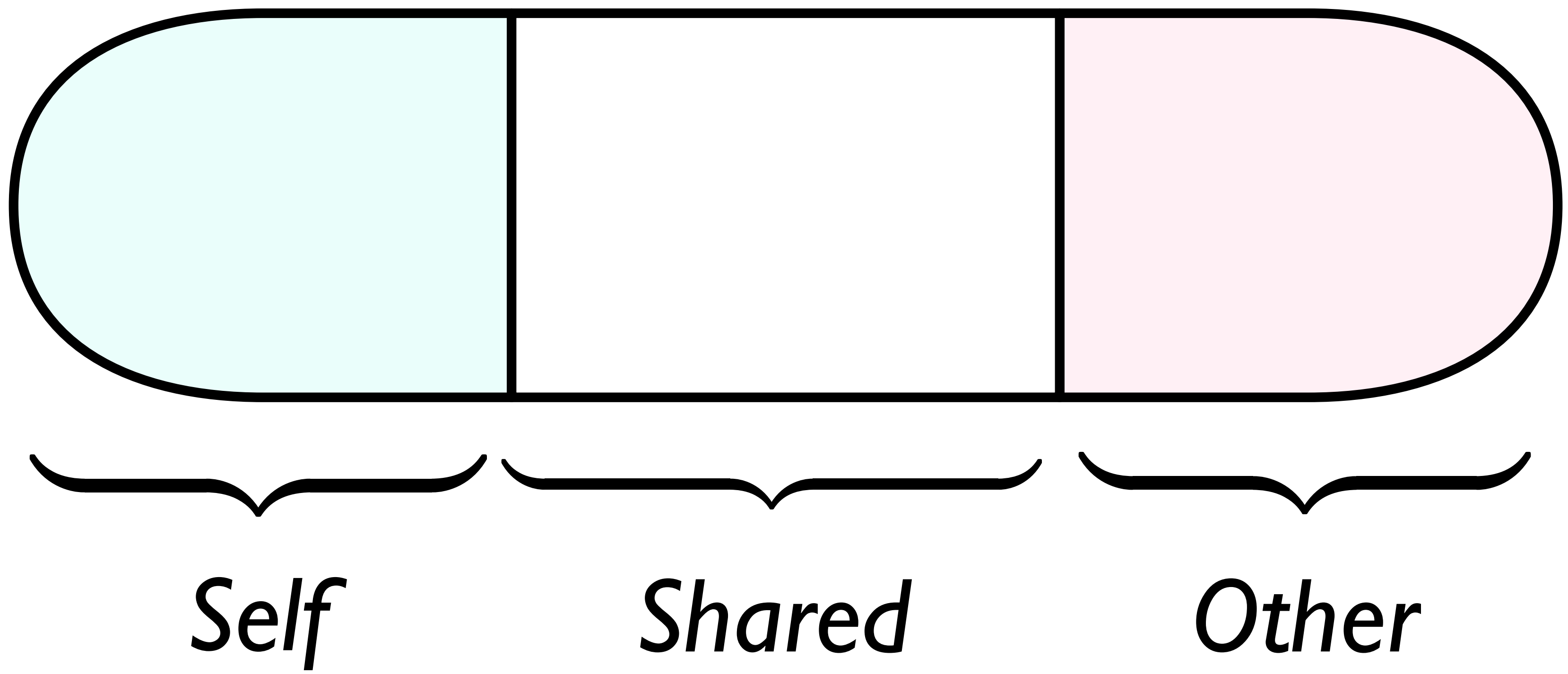
Concurroid States



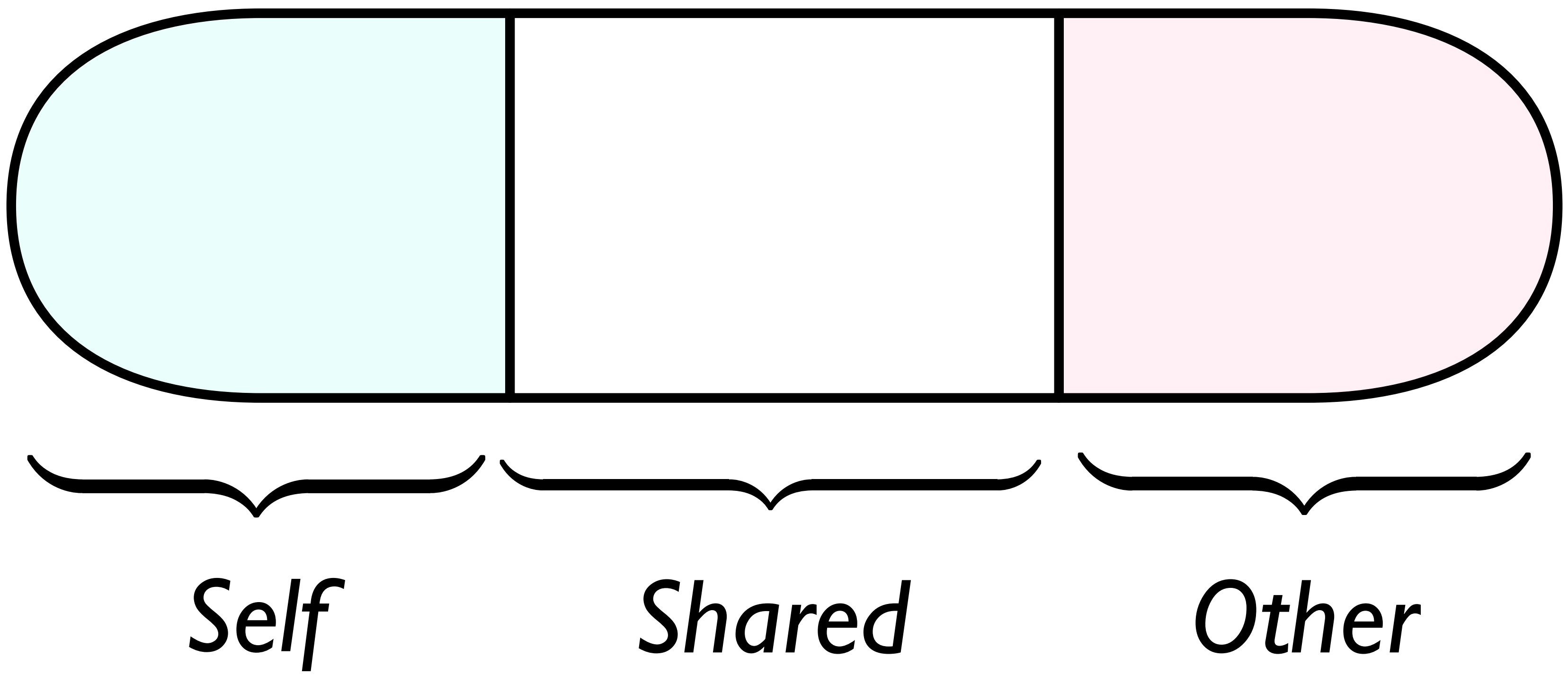
Concurroid States



Concurroid States

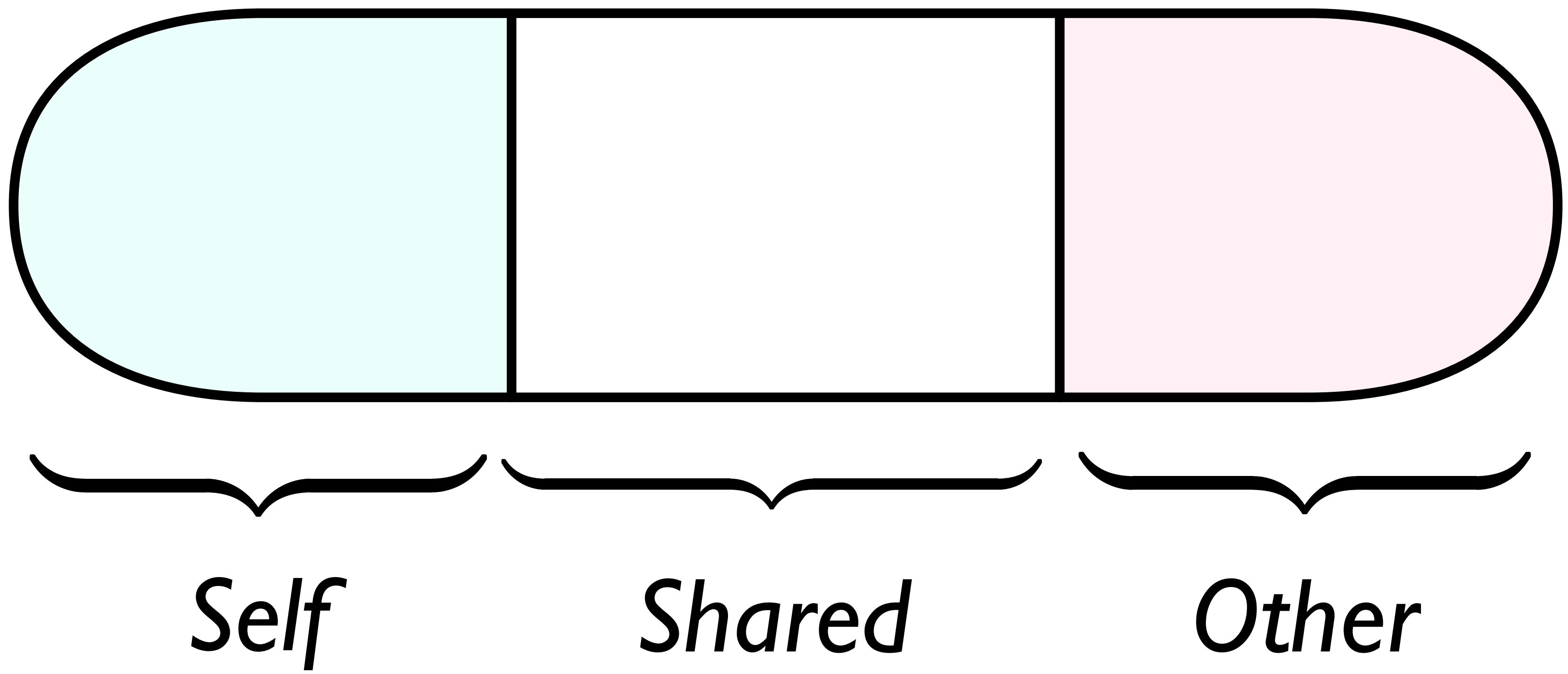


Concurroid States



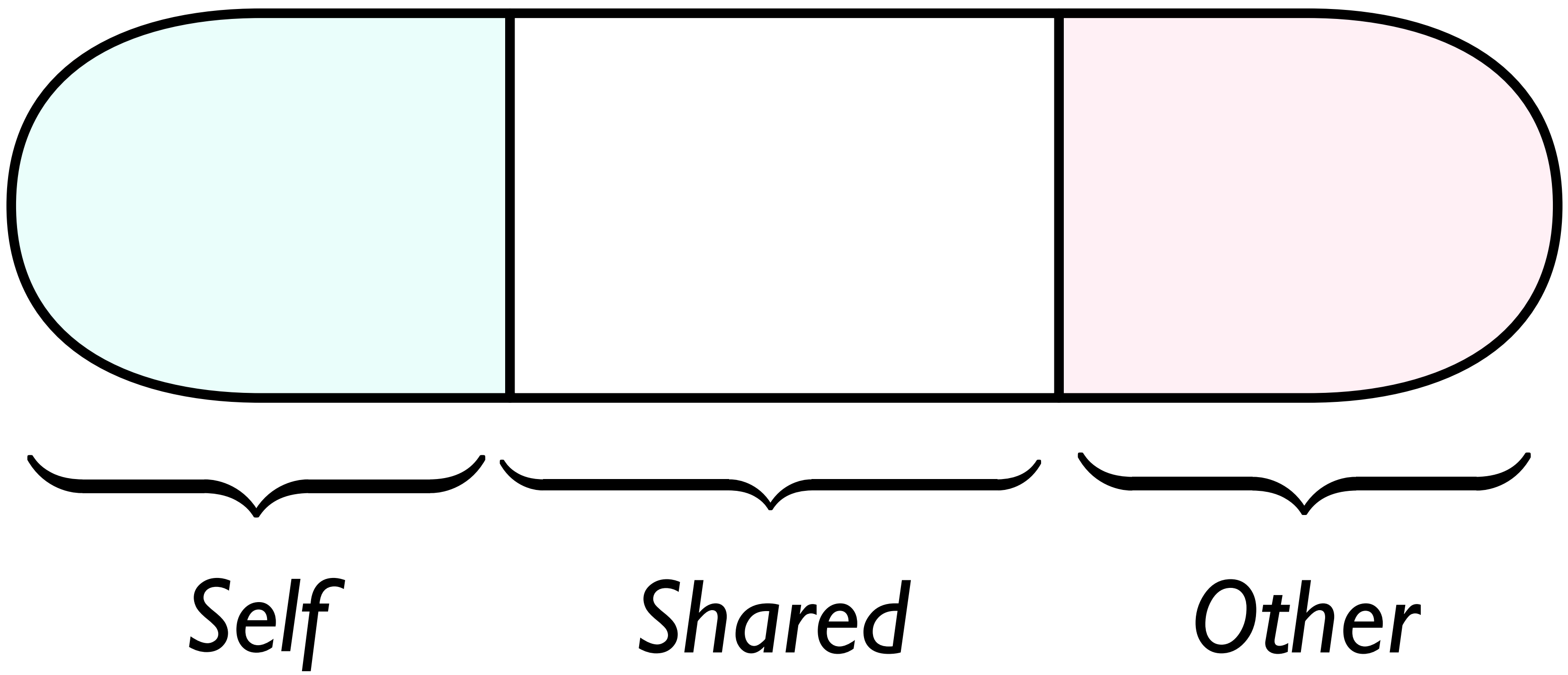
- *Self* - owned by <u>me</u>

Concurroid States



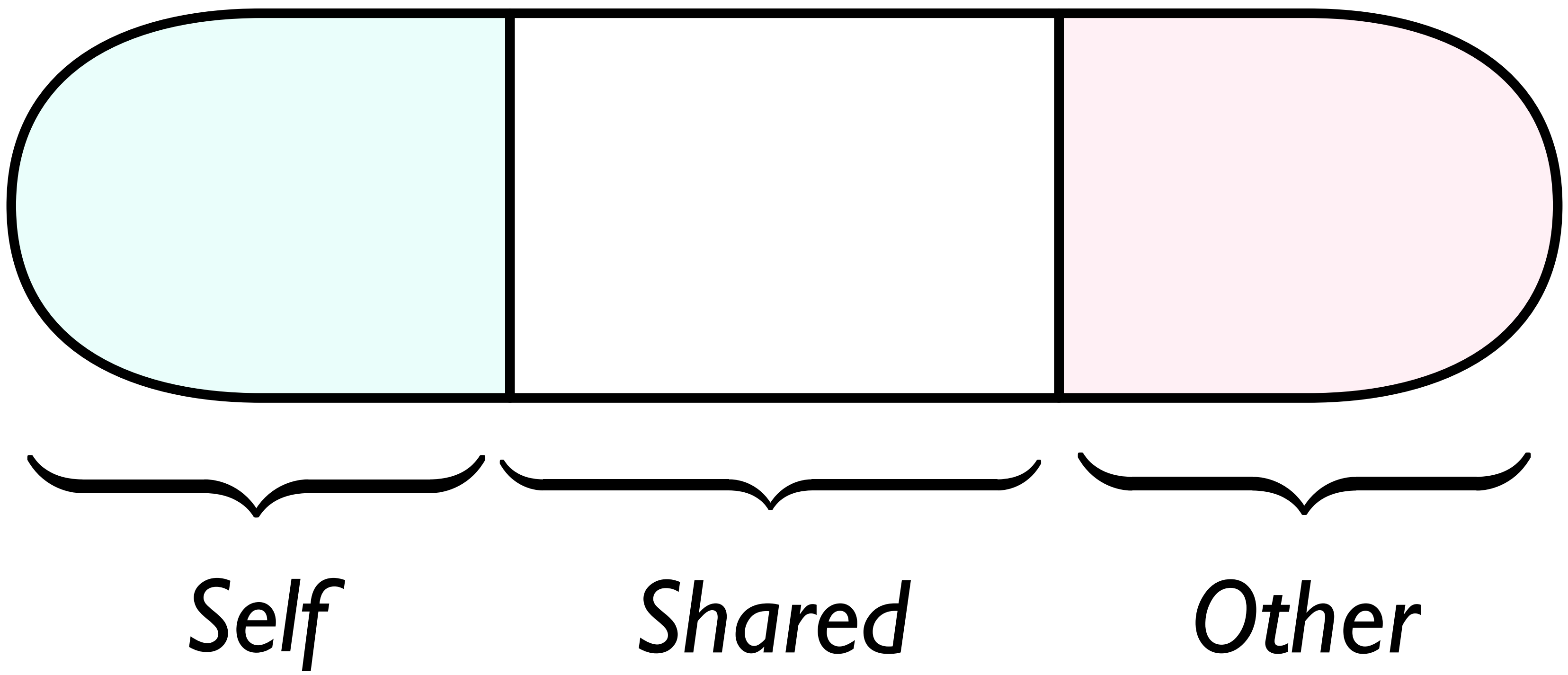
- *Self* - owned by me
- *Other* - owned by all others

Concurroid States



- *Self* - owned by <u>me</u>
- *Other* - owned by <u>all others</u>
- *Shared* - owned by <u>the resource</u>

Concurroid States



- *Self* - owned by <u>me</u>
- *Other* - owned by <u>all others</u>
- *Shared* - owned by <u>the resource</u>
- Self and Other are elements of a *Partial Commutative Monoid* (PCM): $(\mathrm{S}, \mathbf{0}, \oplus)$.

Building a concurroid for Ticketed Lock


SERVING No.
27
n_1

n_1

n_2

n_1

$$n_1 \leq n < n_2$$

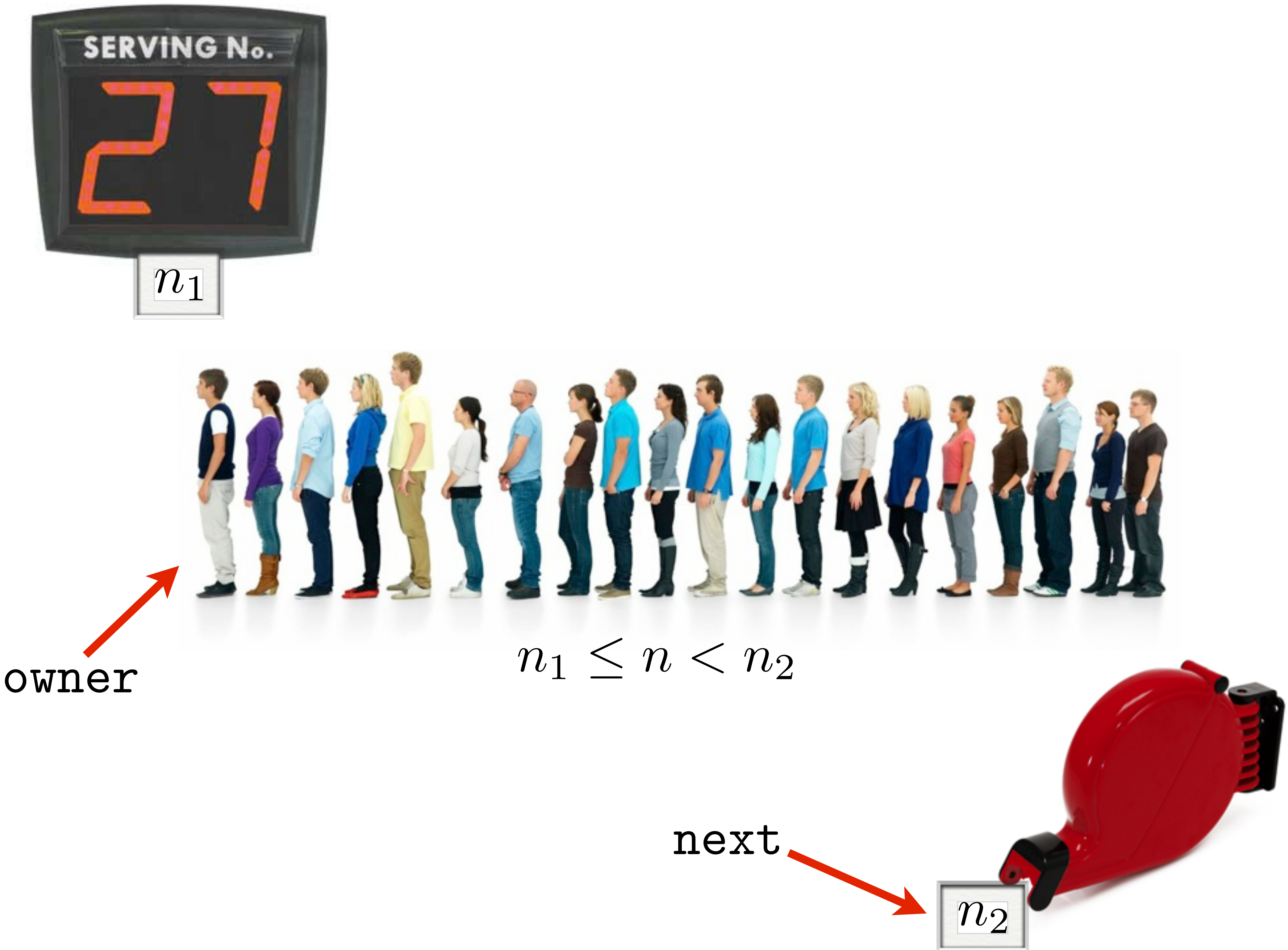
n_2

n_1

owner

$$n_1 \leq n < n_2$$

n_2


SERVING No.
27
n_1
owner
$n_1 \leq n < n_2$
next
n_2

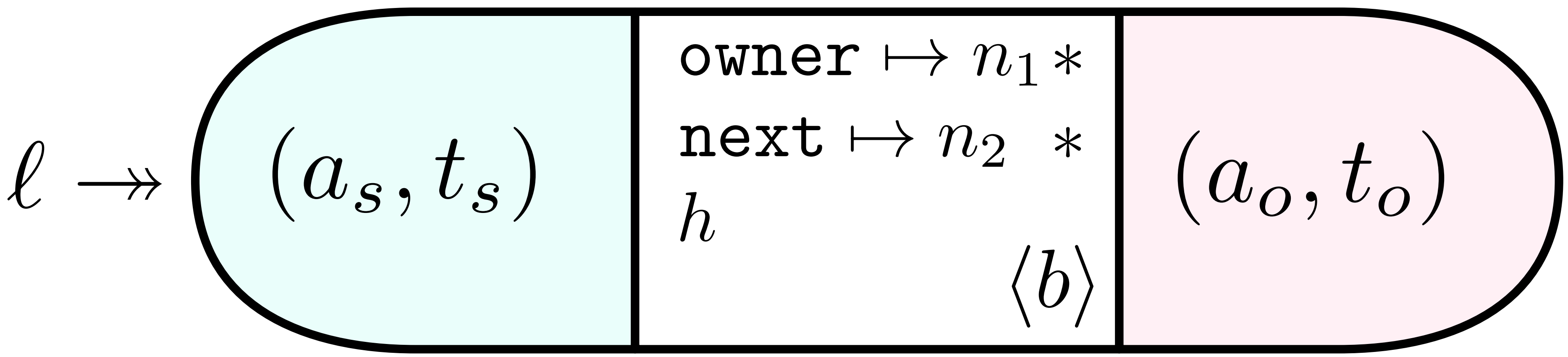
Reference Implementation

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

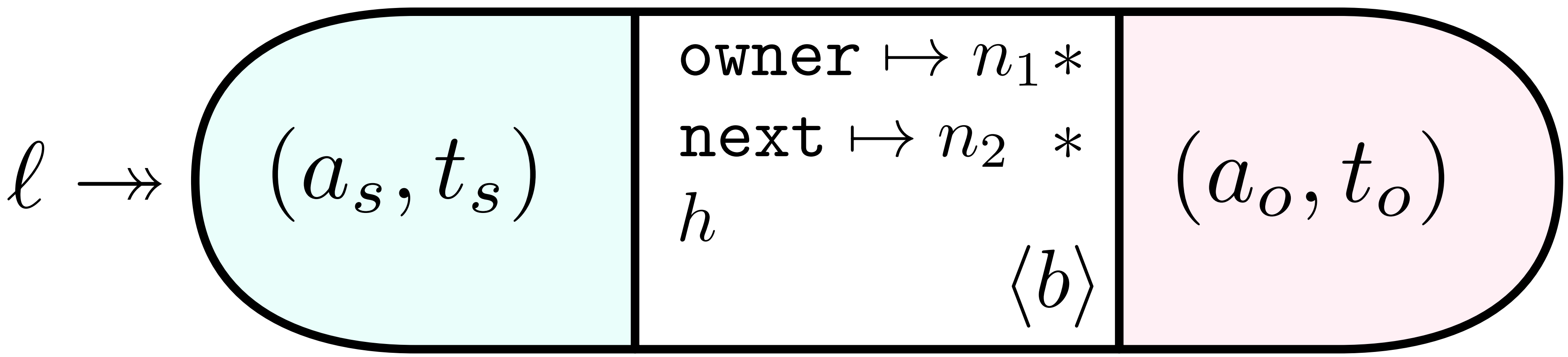
```
unlock = {
  INCR_OWN;
}
```

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

Ticketed Lock States

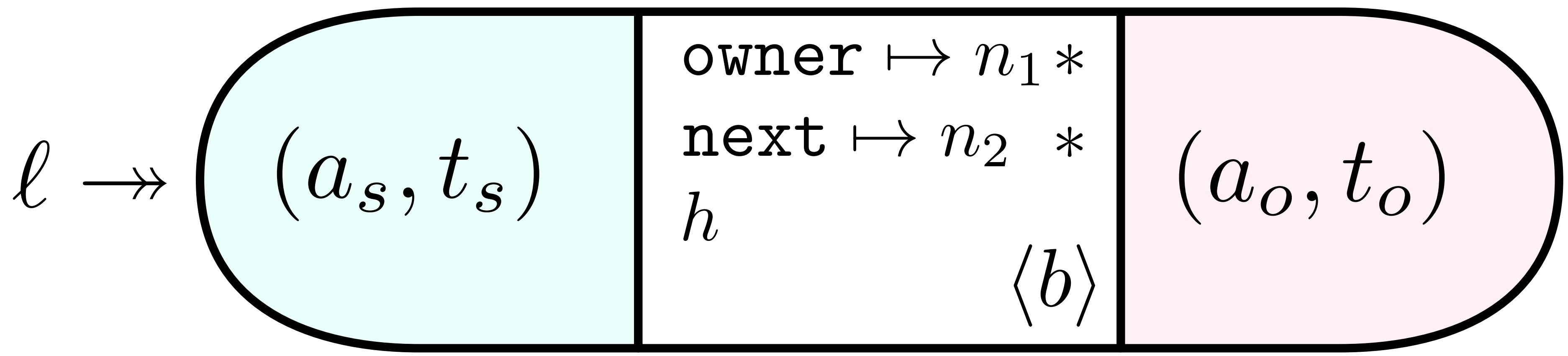


Ticketed Lock States



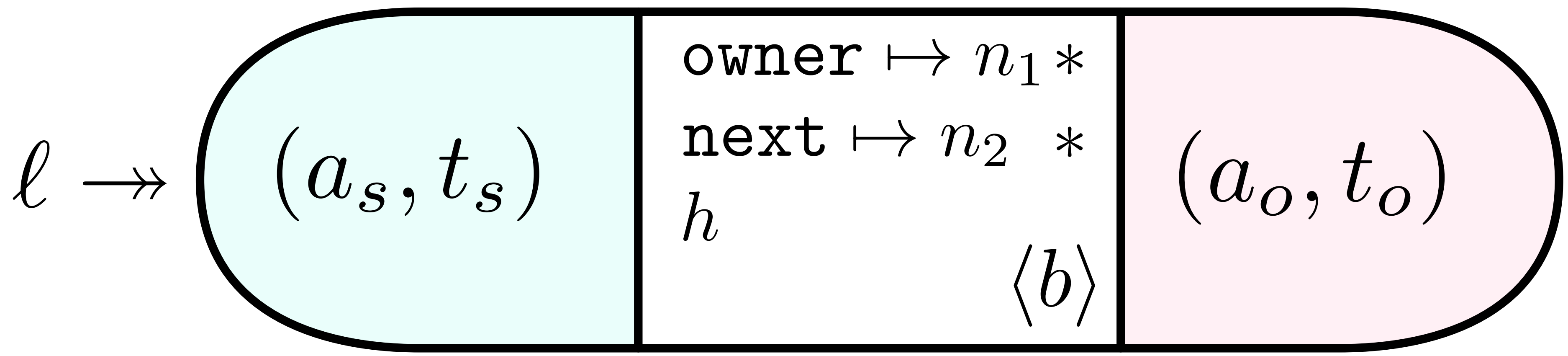
- a_s, a_o - auxiliaries controlled by self/other

Ticketed Lock States



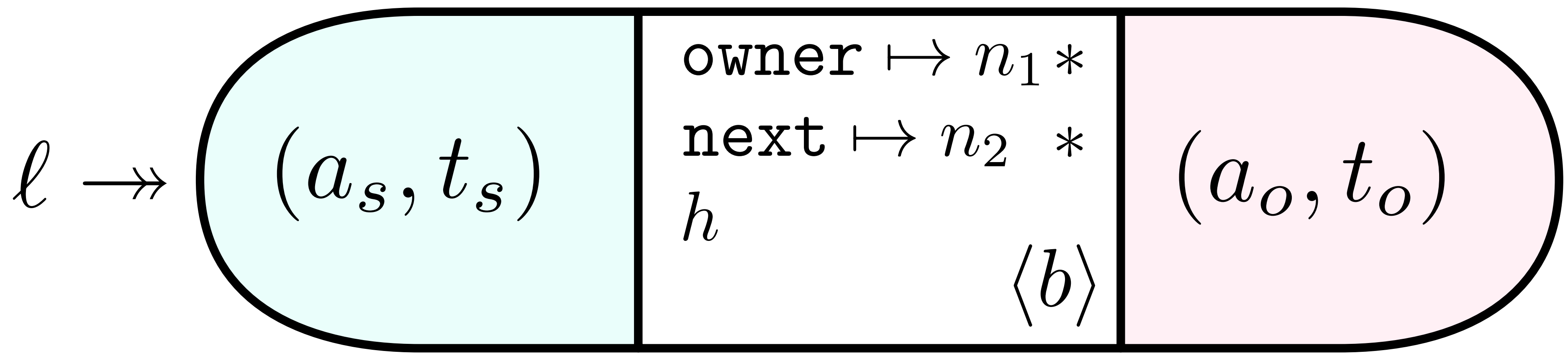
- a_s, a_o - auxiliaries controlled by self/other
- t_s - tickets, owned by *self*

Ticketed Lock States



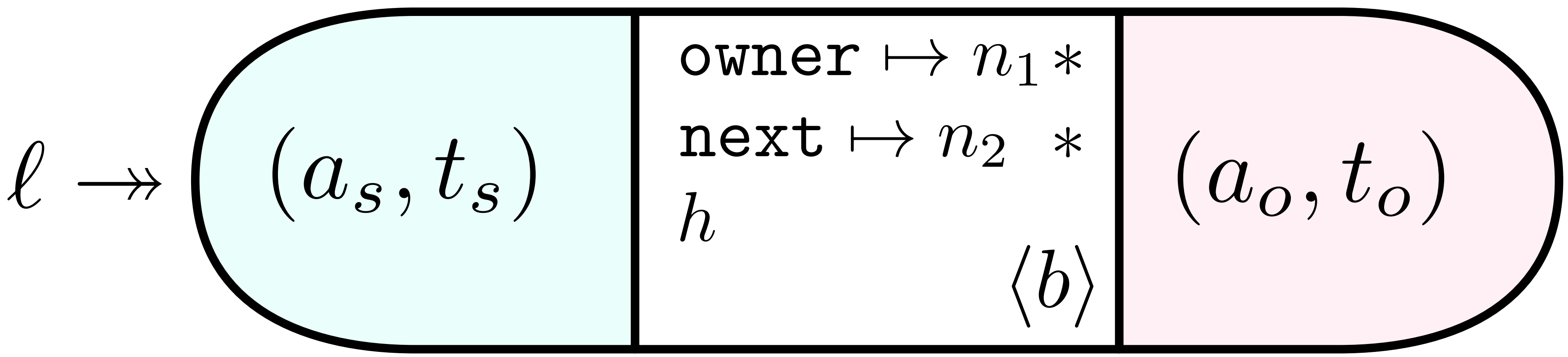
- a_s, a_o - auxiliaries controlled by self/other
- t_s - tickets, owned by *self*
- t_o - tickets, owned by *other* threads

Ticketed Lock States



- a_s, a_o - auxiliaries controlled by self/other
- t_s - tickets, owned by *self*
- t_o - tickets, owned by *other* threads
- b - administrative flag to indicate locking

Ticketed Lock States



- a_s, a_o - auxiliaries controlled by self/other
- t_s - tickets, owned by *self*
- t_o - tickets, owned by *other* threads
- b - administrative flag to indicate locking
- ℓ - label to identify *this* particular instance of TLock concurroid

Ticketed Lock Invariant

Ticketed Lock Invariant

$$s = \ \ell \twoheadrightarrow \ (a_s, t_s) \ \Big| \ \begin{array}{l} \texttt{owner} \mapsto n_1 * \\ \texttt{next} \mapsto n_2 \ * \\ h \\ \hfill \langle b \rangle \end{array} \ \Big| \ (a_o, t_o) \quad \wedge$$

Ticketed Lock Invariant

$$s = \ell \twoheadrightarrow \left((a_s, t_s) \;\middle|\; \begin{array}{l} \texttt{owner} \mapsto n_1 * \\ \texttt{next} \mapsto n_2 * \\ h \hfill \langle b \rangle \end{array} \;\middle|\; (a_o, t_o) \right) \wedge$$

All dispensed tickets

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

Ticketed Lock Invariant

$$s = \ \ell \twoheadrightarrow \left(\begin{array}{c|c|c} (a_s, t_s) & \begin{array}{l} \texttt{owner} \mapsto n_1 * \\ \texttt{next} \mapsto n_2 \ * \\ h \\ \hfill \langle b \rangle \end{array} & (a_o, t_o) \end{array} \right) \ \wedge$$

All dispensed tickets

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

$$\left(\qquad \right)$$

Ticketed Lock Invariant

$$s = \ \ell \twoheadrightarrow \left[(a_s, t_s) \ \middle|\ \begin{array}{l} \texttt{owner} \mapsto n_1 * \\ \texttt{next} \mapsto n_2 \ * \\ h \\ \hfill \langle b \rangle \end{array} \ \middle|\ (a_o, t_o) \right] \quad \wedge$$

All dispensed tickets

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

Locked

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

Ticketed Lock Invariant

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

All dispensed tickets

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

Locked

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

Ticketed Lock Invariant

$$s = \ \ell \twoheadrightarrow \left((a_s, t_s) \ \middle|\ \begin{array}{l} \texttt{owner} \mapsto n_1 * \\ \texttt{next} \mapsto n_2 \ * \\ h \\ \hfill \langle b \rangle \end{array} \ \middle|\ (a_o, t_o) \right) \quad \wedge$$

All dispensed tickets

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

Locked

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

Unlocked

Ticketed Lock Invariant

$$s = \ \ell \twoheadrightarrow \left((a_s, t_s) \;\middle|\; \begin{array}{l} \texttt{owner} \mapsto n_1 * \\ \texttt{next} \mapsto n_2 \ * \\ h \\ \hfill \langle b \rangle \end{array} \;\middle|\; (a_o, t_o) \right) \quad \wedge$$

All dispensed tickets

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

Locked

$$\left(\begin{array}{ll} \multicolumn{2}{l}{\boxed{(n_1 \in (t_s \oplus t_o) \ \wedge \ b = \mathbf{true} \ \wedge \ h = \mathsf{emp})} \ \vee} \\ & \hfill \text{Transit} \\ \textbf{if } n_1 < n2 \quad \textbf{then} & \boxed{n_1 \in (t_s \oplus t_o) \ \wedge \ b = \mathbf{false} \ \wedge \ I(a_s \oplus a_o)h} \\ \hfill \textbf{else} & \boxed{n_1 = n_2 \ \wedge \ b = \mathbf{false} \ \wedge \ I(a_s \oplus a_o)h} \end{array} \right)$$

Unlocked

Transitions

Internal Transitions

Internal Transitions

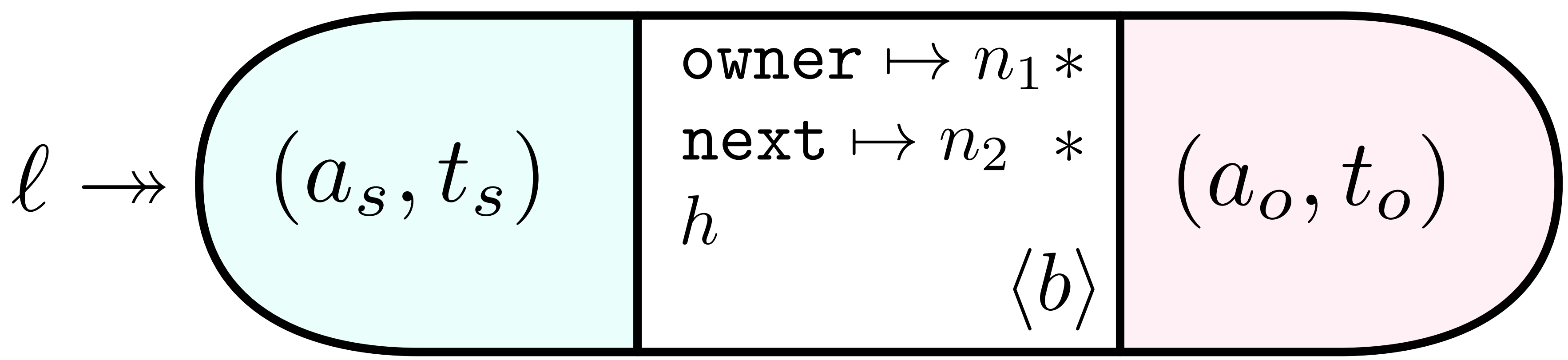
Intuition:

drawing a ticket from the dispenser

Internal Transitions

Intuition:

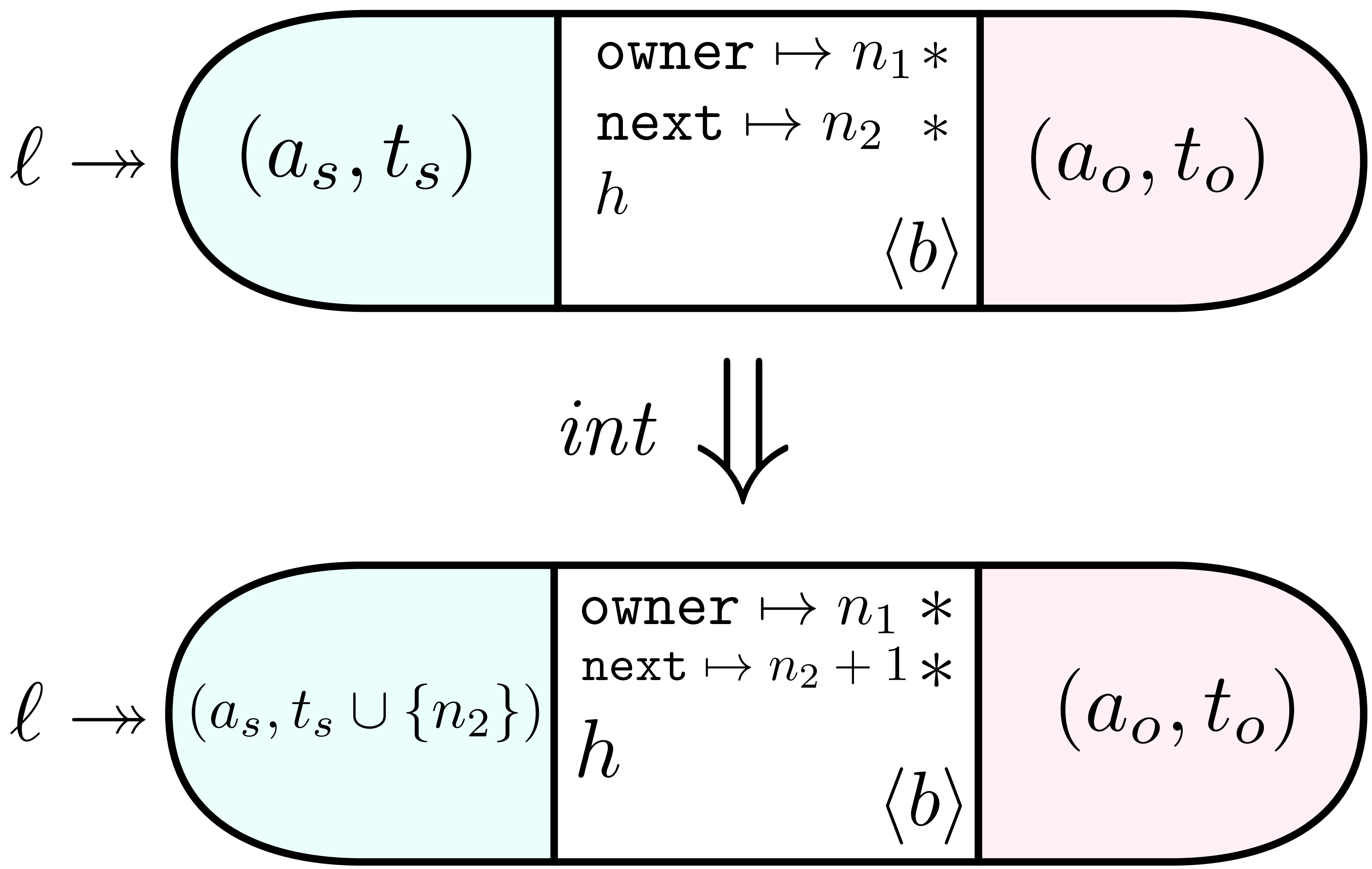
drawing a ticket from the dispenser



Internal Transitions

<u>*Intuition:*</u>

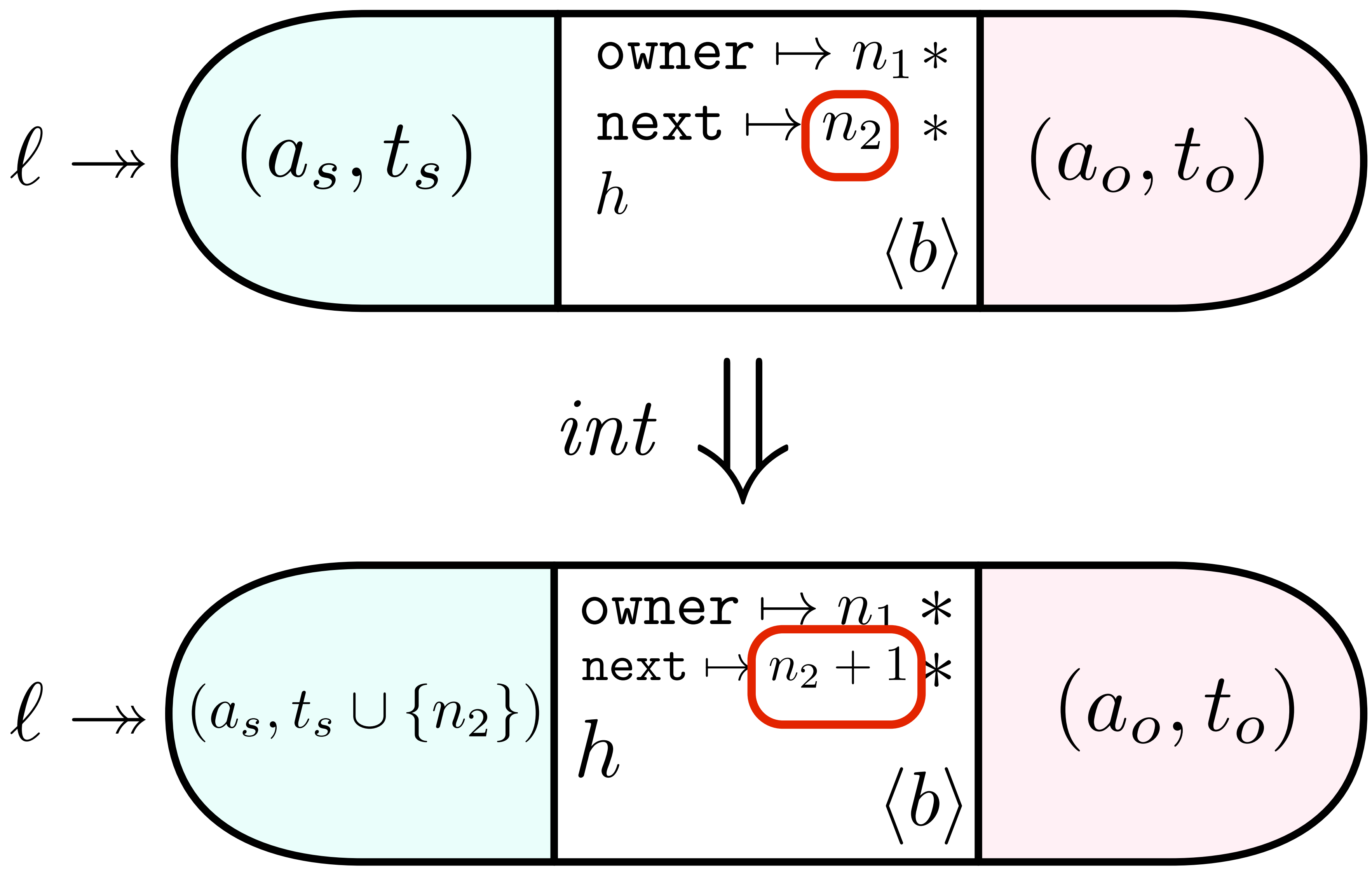
drawing a ticket from the dispenser



Internal Transitions

Intuition:

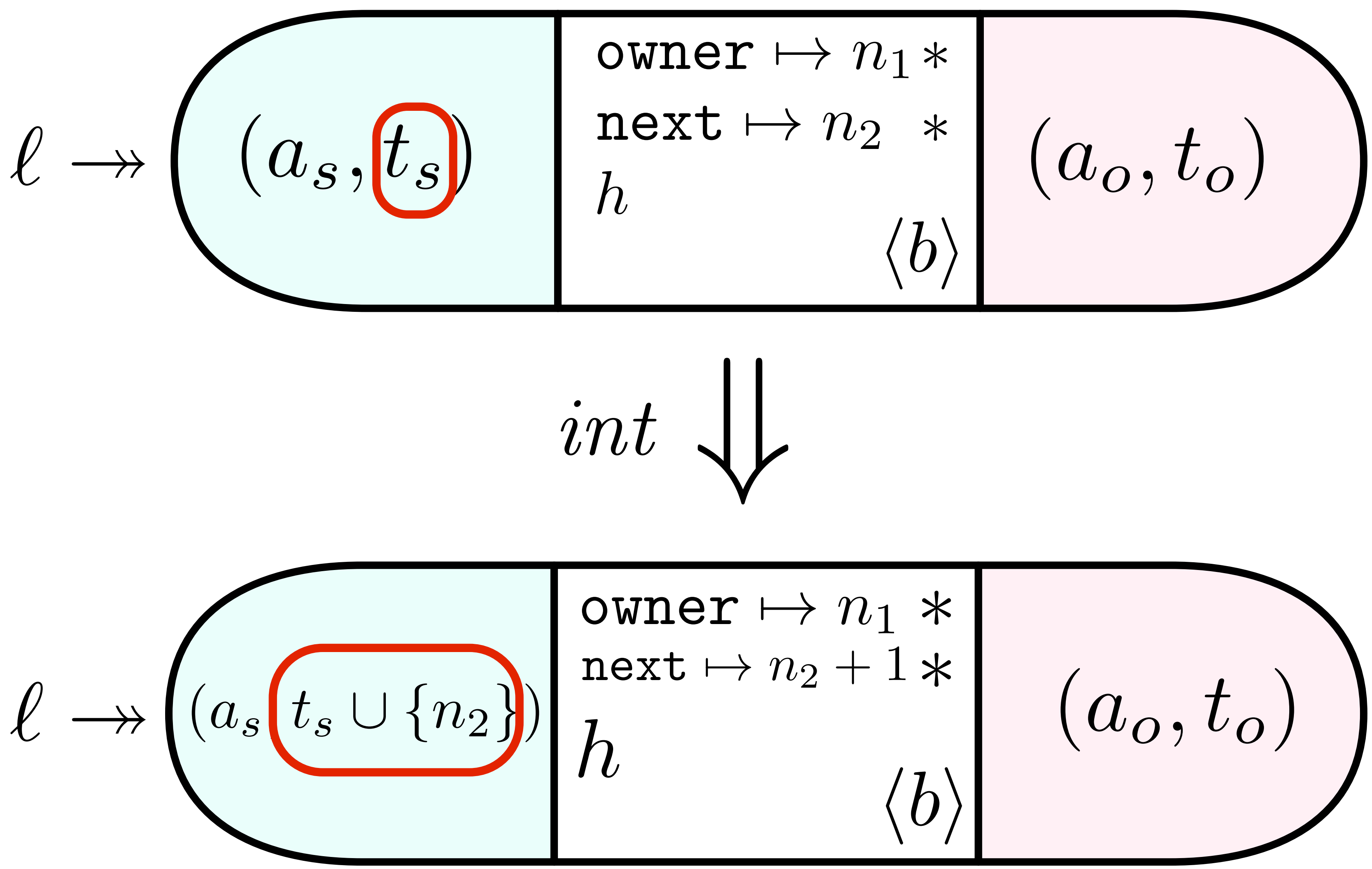
drawing a ticket from the dispenser



Internal Transitions

Intuition:

drawing a ticket from the dispenser



Communication

Intuition:

Channels with different polarity

Intuition:

Channels with different polarity

Implementation:

Acquire/Release transitions
(communication is via heap ownership transfer)

Acquire Transitions

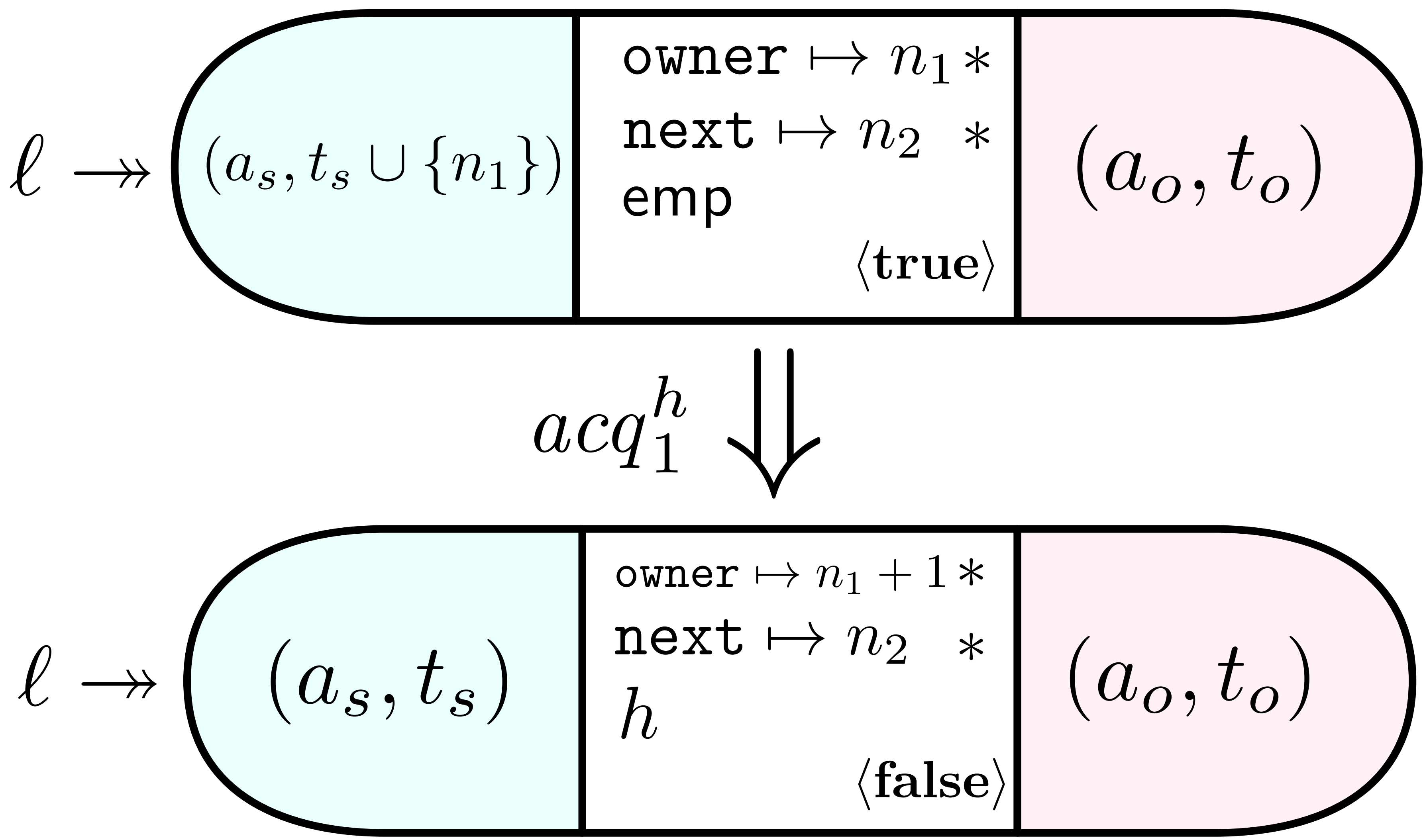
Intuition:

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

Acquire Transitions

Intuition:

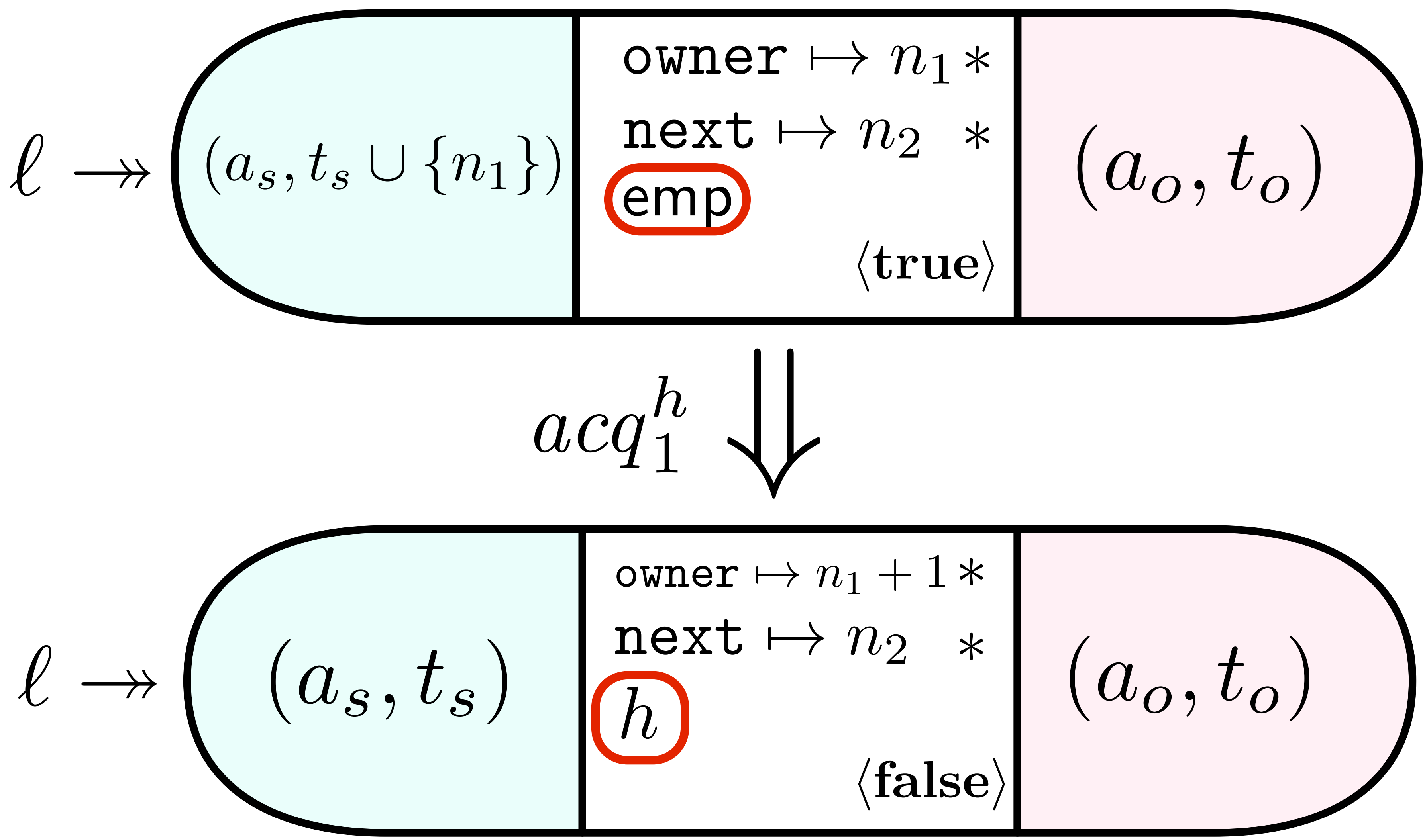
the lock obtains back ownership over the heap and increments the service counter (`owner`)



Acquire Transitions

<u>Intuition:</u>

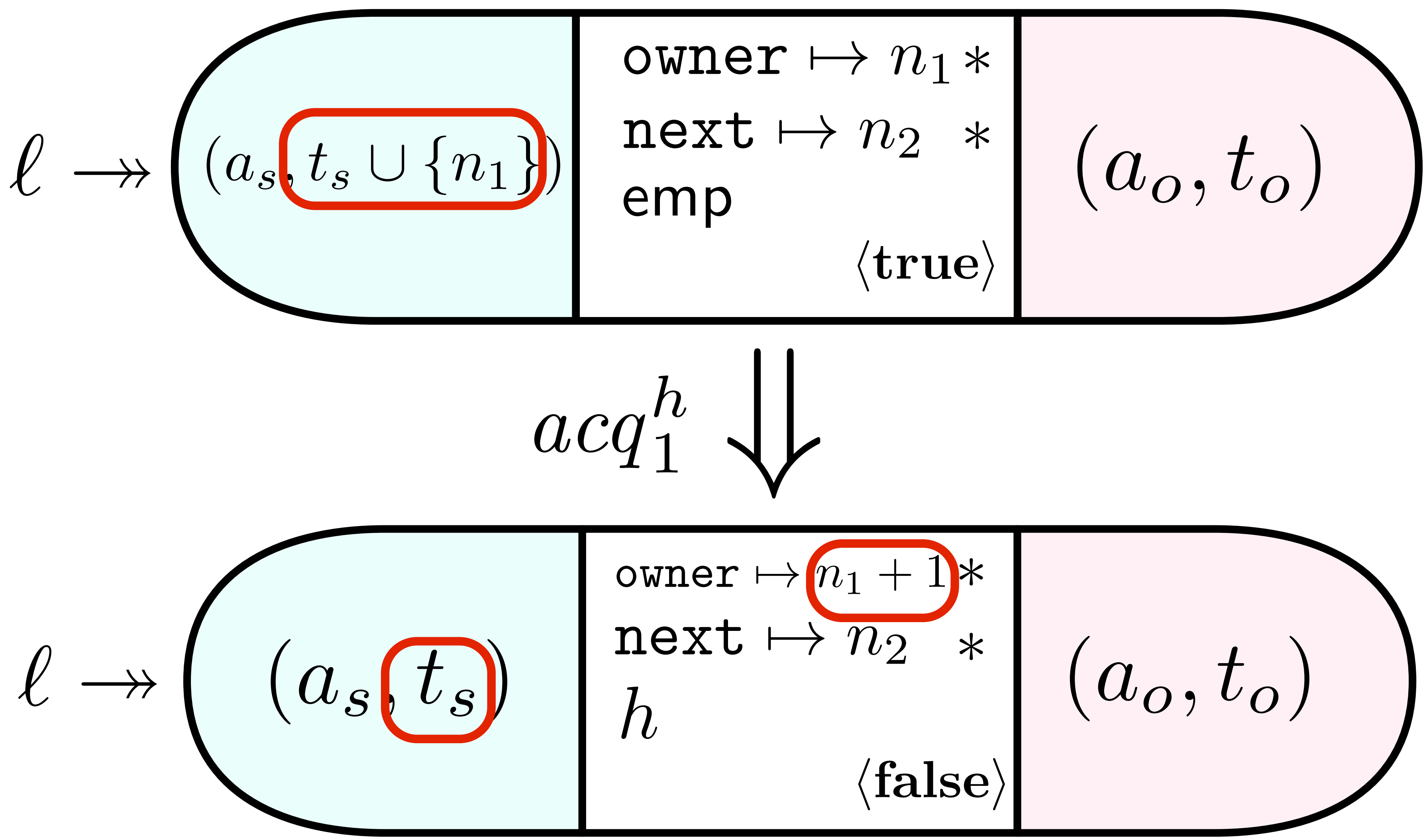
the lock obtains back ownership over the heap and increments the service counter (`owner`)



Acquire Transitions

Intuition:

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



Release Transitions

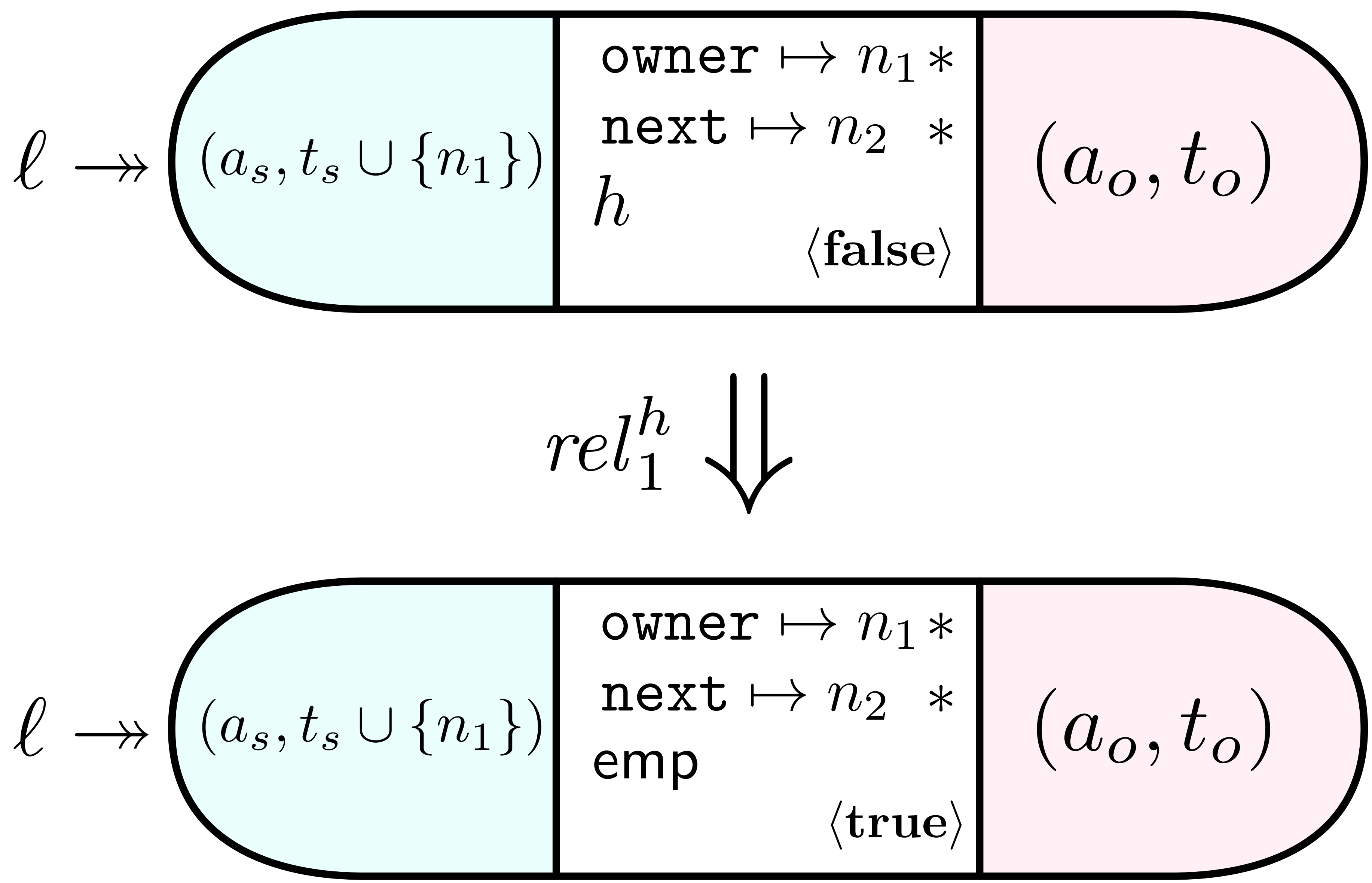
Intuition:

the lock gave up ownership over the heap

Release Transitions

Intuition:

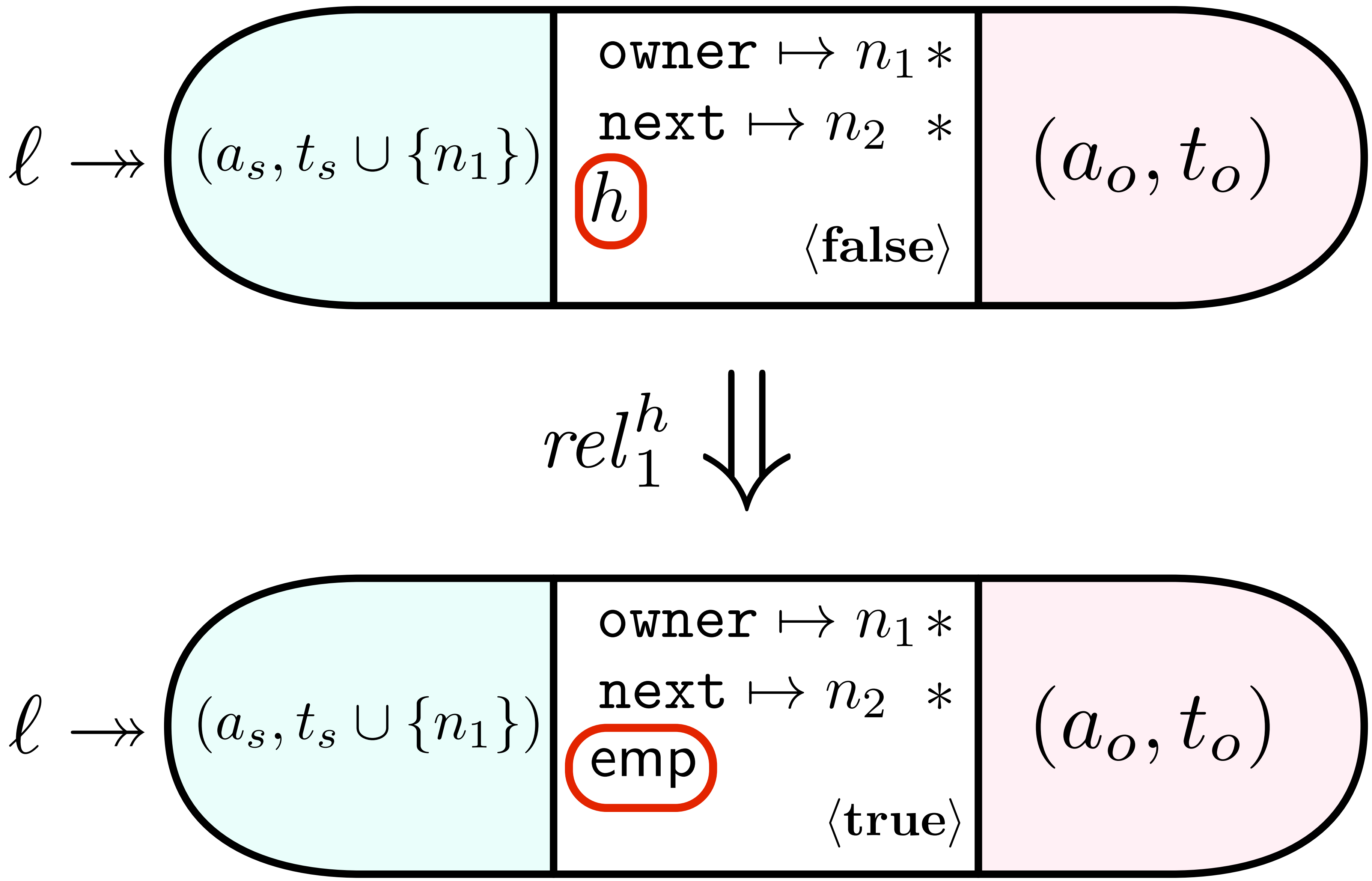
the lock gave up ownership over the heap



Release Transitions

Intuition:

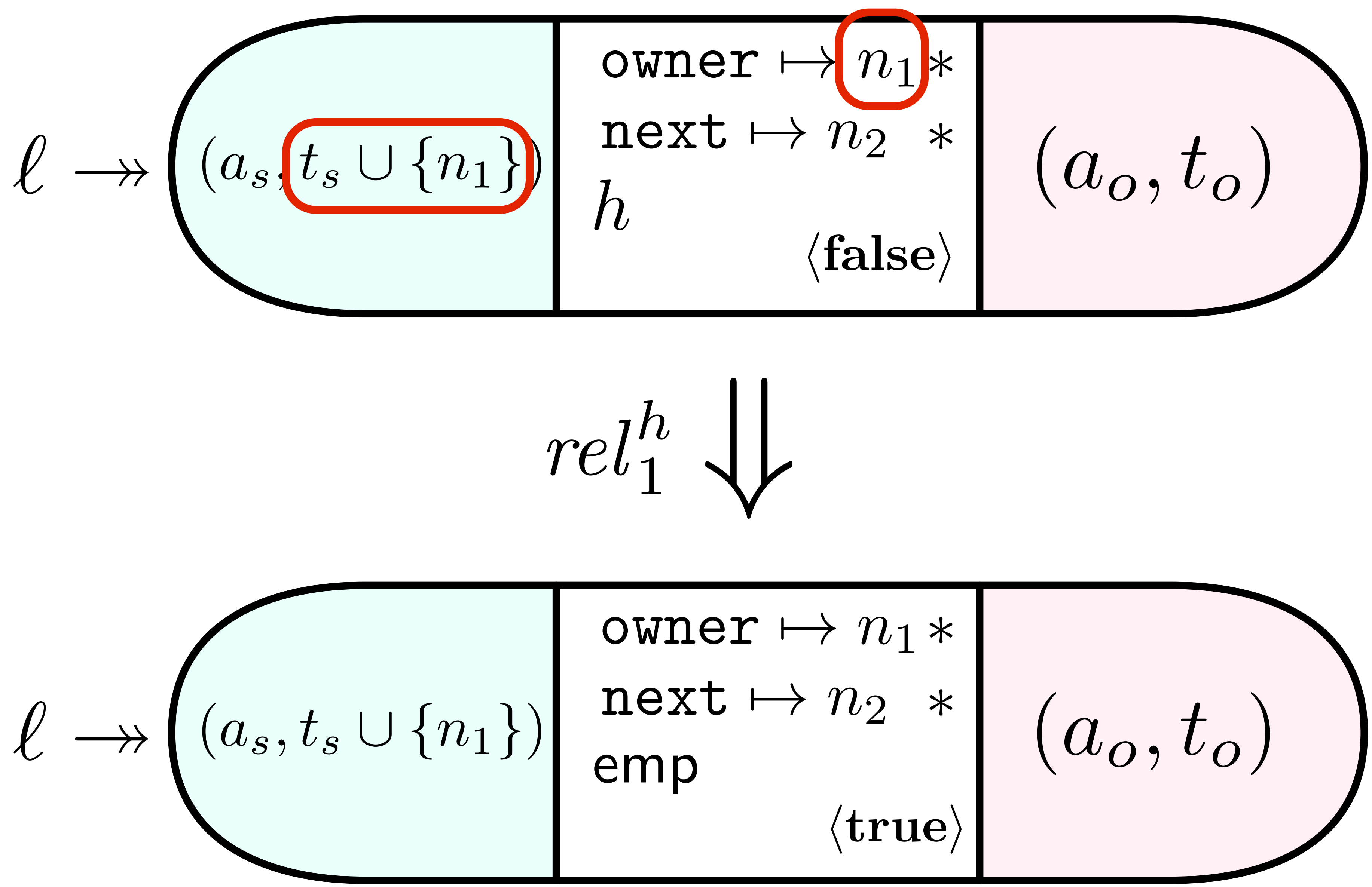
the lock gave up ownership over the heap



Release Transitions

<u>*Intuition:*</u>

the lock gave up ownership over the heap

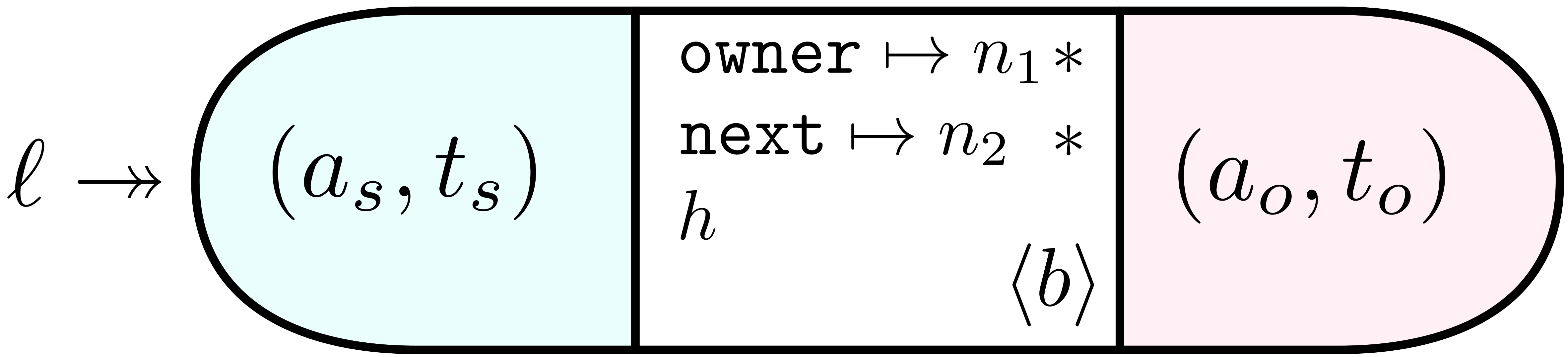


Transitions don't change the *other* part!

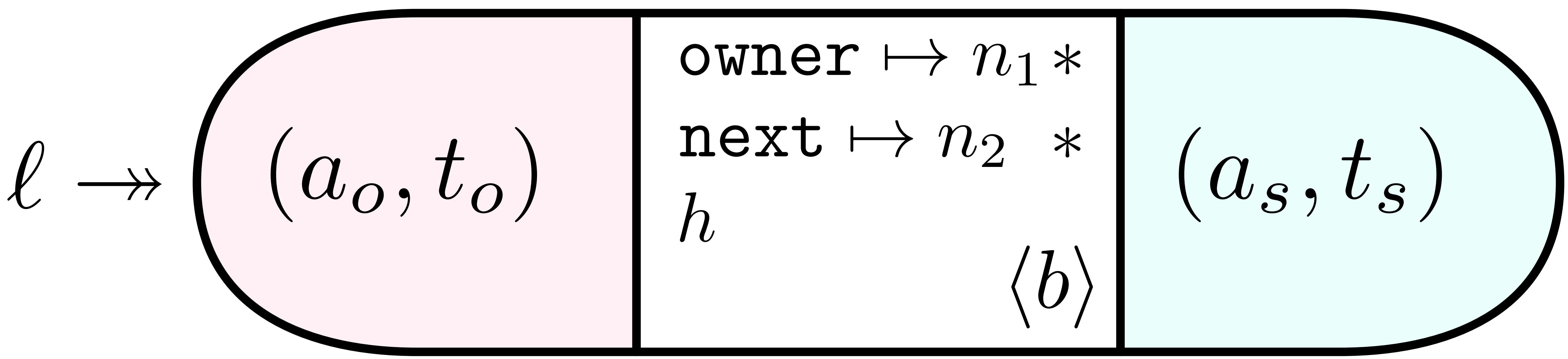
Transitions don’t change the *other* part!

$\boxed{\text{Transitions} = \underline{\textit{Guarantee}}}$

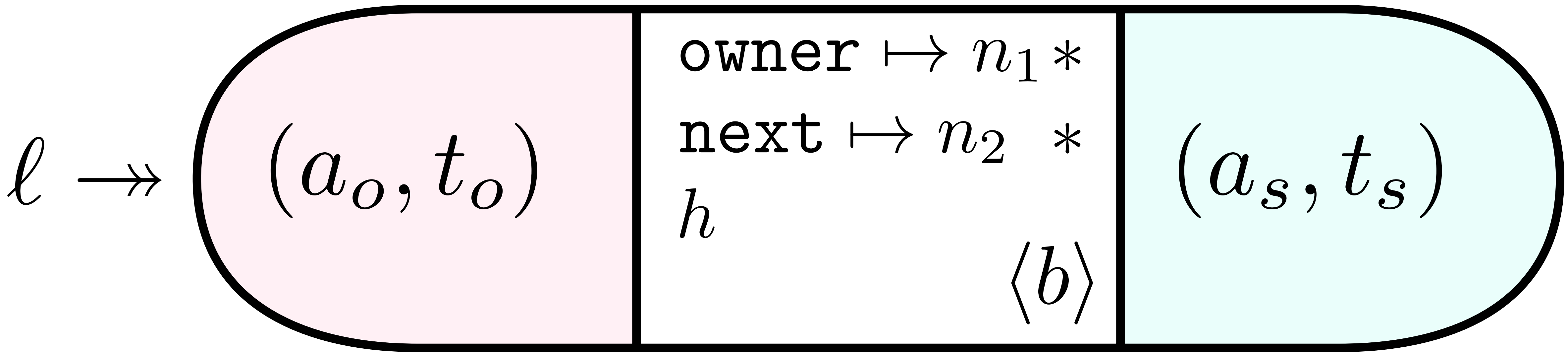
Transposing the Concurroid



Transposing the Concurroid

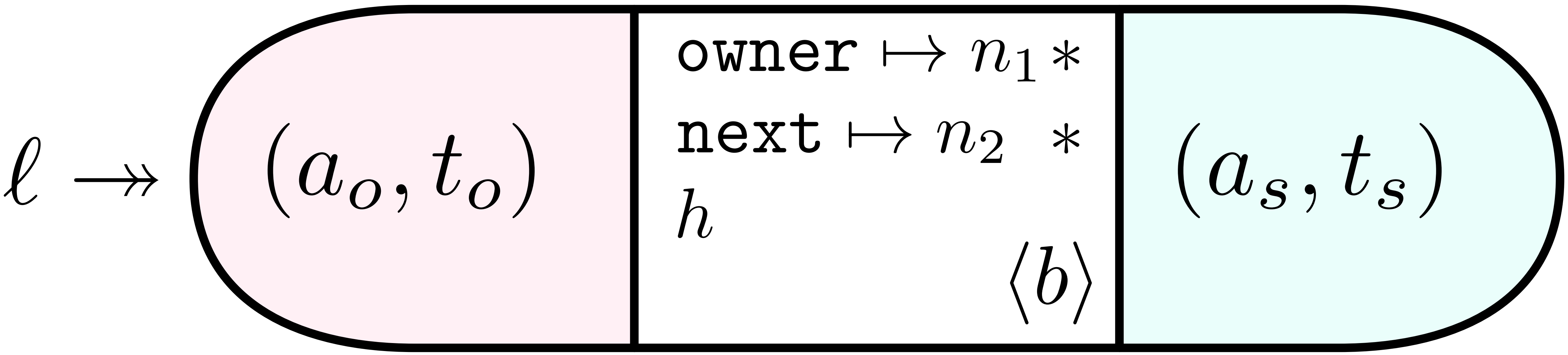


Transposing the Concurroid



Transitions of transposed = ***Rely***

Transposing the Concurroid



$\boxed{\text{Transitions of transposed} = \underline{\textit{Rely}}}$

reminiscent to <u>tokens</u> by Turon *et al.* [POPL'13, ICFP'13]

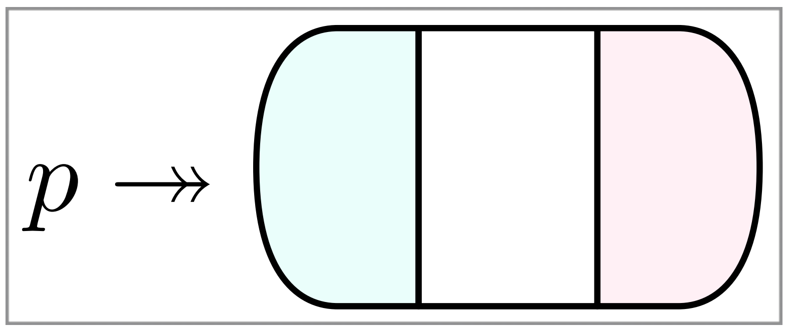
Composing Concurroids

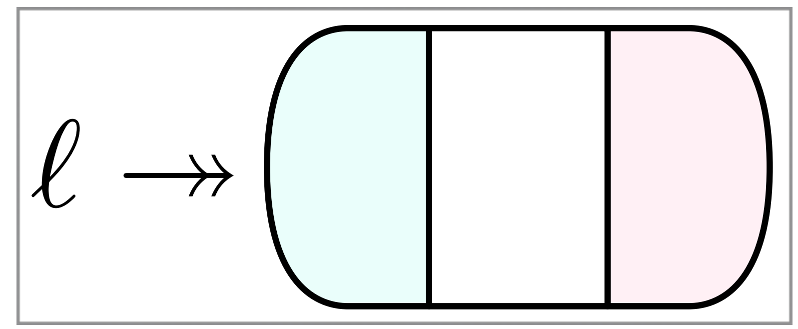
Intuition:

Connect communication channels with right polarity

Intuition:

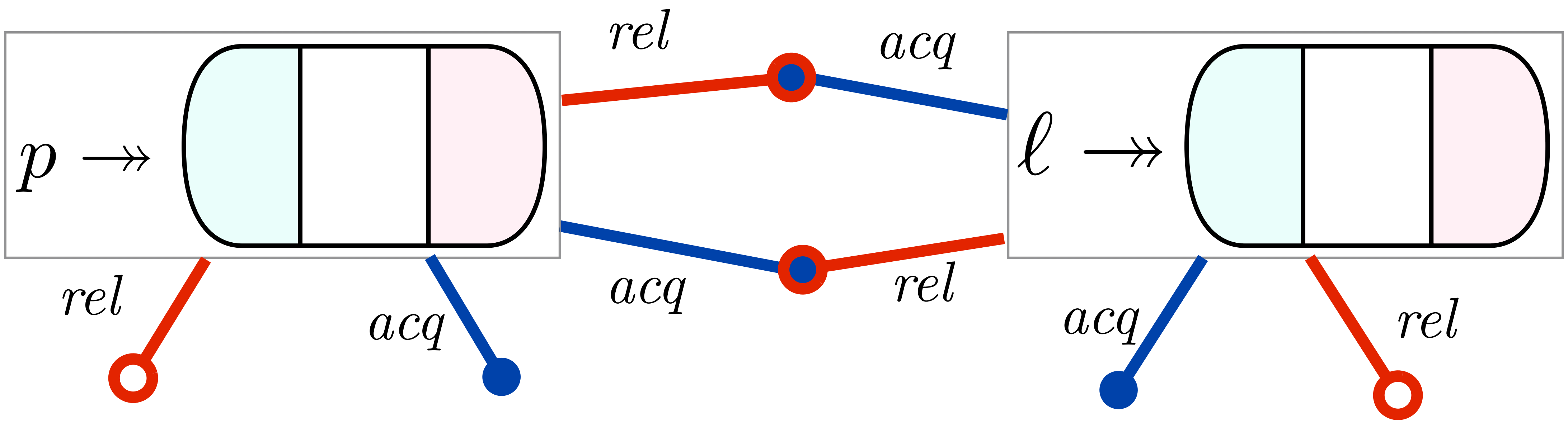
Connect communication channels with right polarity





<u>Intuition:</u>

Connect communication channels with right polarity



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

Entanglement Operators

$$\bowtie, \rtimes, \ltimes, \times \ldots$$

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

Entanglement Operators

$$\bowtie, \rtimes, \ltimes, \times \ldots$$

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

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

Useful Entanglement Operators

$\times$ - “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

Useful Entanglement Operators

$\times$ - "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>: $U \rtimes (V_1 \times V_2) = (U \rtimes V_1) \rtimes V_2$

Programming with Concurroids

Transitions are not yet commands!

Transitions are not yet commands!

They only describe
some *correct* behavior.

Atomic Actions

- Defined as subsets of *internal transitions*
- Specify the result
- Operational meaning:
  `READ`, `WRITE`, `SKIP` and various RMW-commands
- *Synchronize* ownership transfer and manipulation with auxiliaries

Recap: TLock Implementation

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

```
unlock = {
  INCR_OWN;
}
```

Recap: TLock Implementation

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

```
unlock = {
  INCR_OWN;
}
```

`TRY(n`$_1$`)` Action Specification

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

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

$\mathtt{TRY(n_1)}$ Action Specification

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

$$\left(\begin{array}{l}
s = p \twoheadrightarrow \boxed{h_s \;\middle|\; \quad \;\middle|\; h_o} \oplus \ell \twoheadrightarrow \boxed{(a_s, t_s \cup \{n_1\}) \;\middle|\; \begin{array}{l} \texttt{owner} \mapsto n_1' * \\ \texttt{next} \mapsto n_2 * \\ h \qquad \langle b \rangle \end{array} \;\middle|\; (a_o, t_o)} \wedge \\[2ex]
\textbf{if } (n_1 = n_1') \\
\textbf{then} \left(\begin{array}{l} s' = p \twoheadrightarrow \boxed{h_s \oplus h \;\middle|\; \quad \;\middle|\; h_o} \oplus \ell \twoheadrightarrow \boxed{(a_s, t_s \cup \{n_1\}) \;\middle|\; \begin{array}{l} \texttt{owner} \mapsto n_1 * \\ \texttt{next} \mapsto n_2 * \\ \mathsf{emp} \qquad \langle \mathbf{true} \rangle \end{array} \;\middle|\; (a_o, t_o)} \wedge \\ I(a_s \oplus a_o)h \;\wedge\; \mathsf{res} = \mathbf{true} \end{array} \right) \\[2ex]
\textbf{else} \quad s' = s \;\wedge\; \mathsf{res} = \mathbf{false}
\end{array} \right)$$

$\mathtt{TRY(n_1)}$ Action Specification

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

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

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

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

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

if $(n_1 = n_1')$

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

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

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

$\mathtt{TRY(n_1)}$ Action Specification

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

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

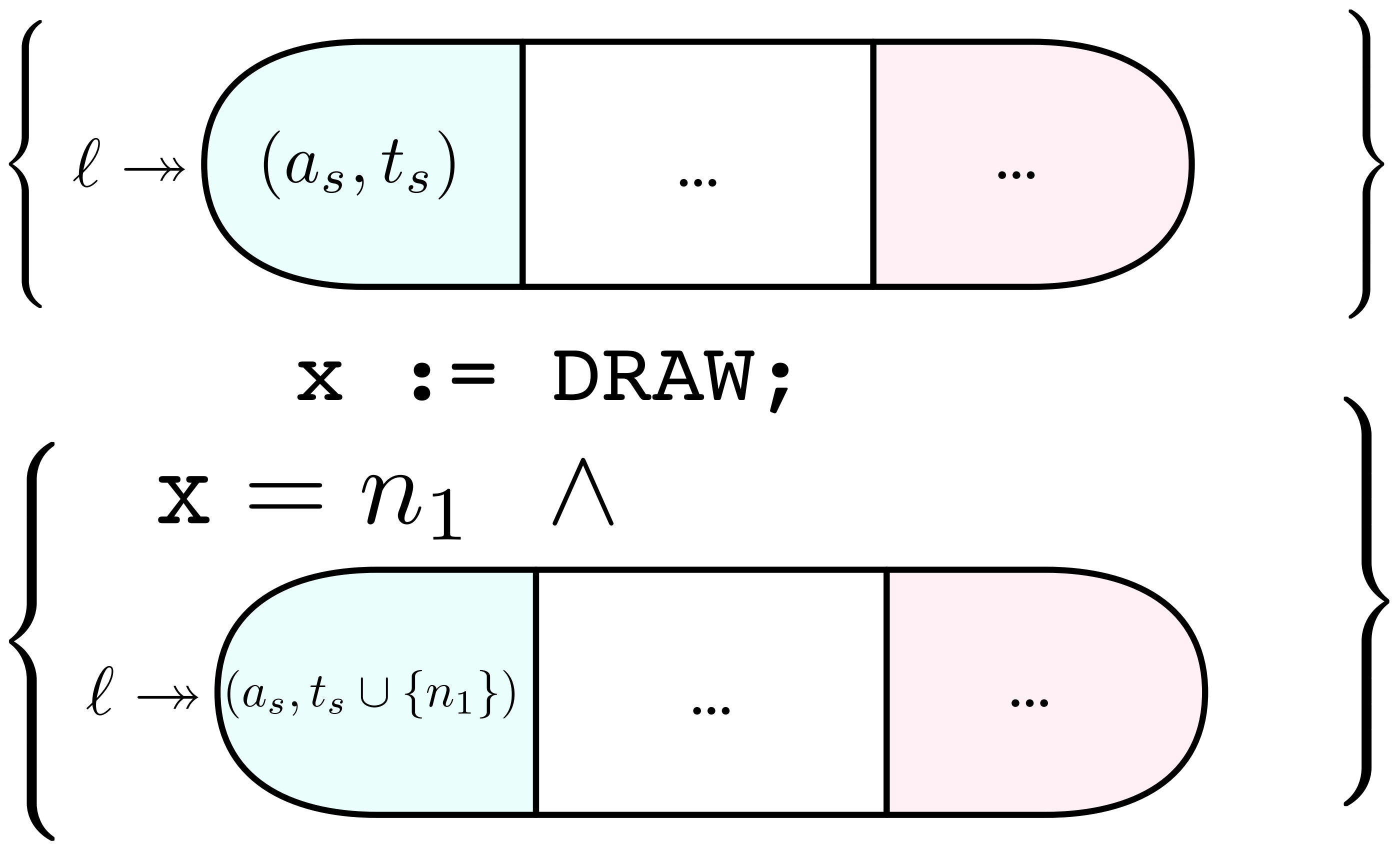
if $(n_1 = n_1')$

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

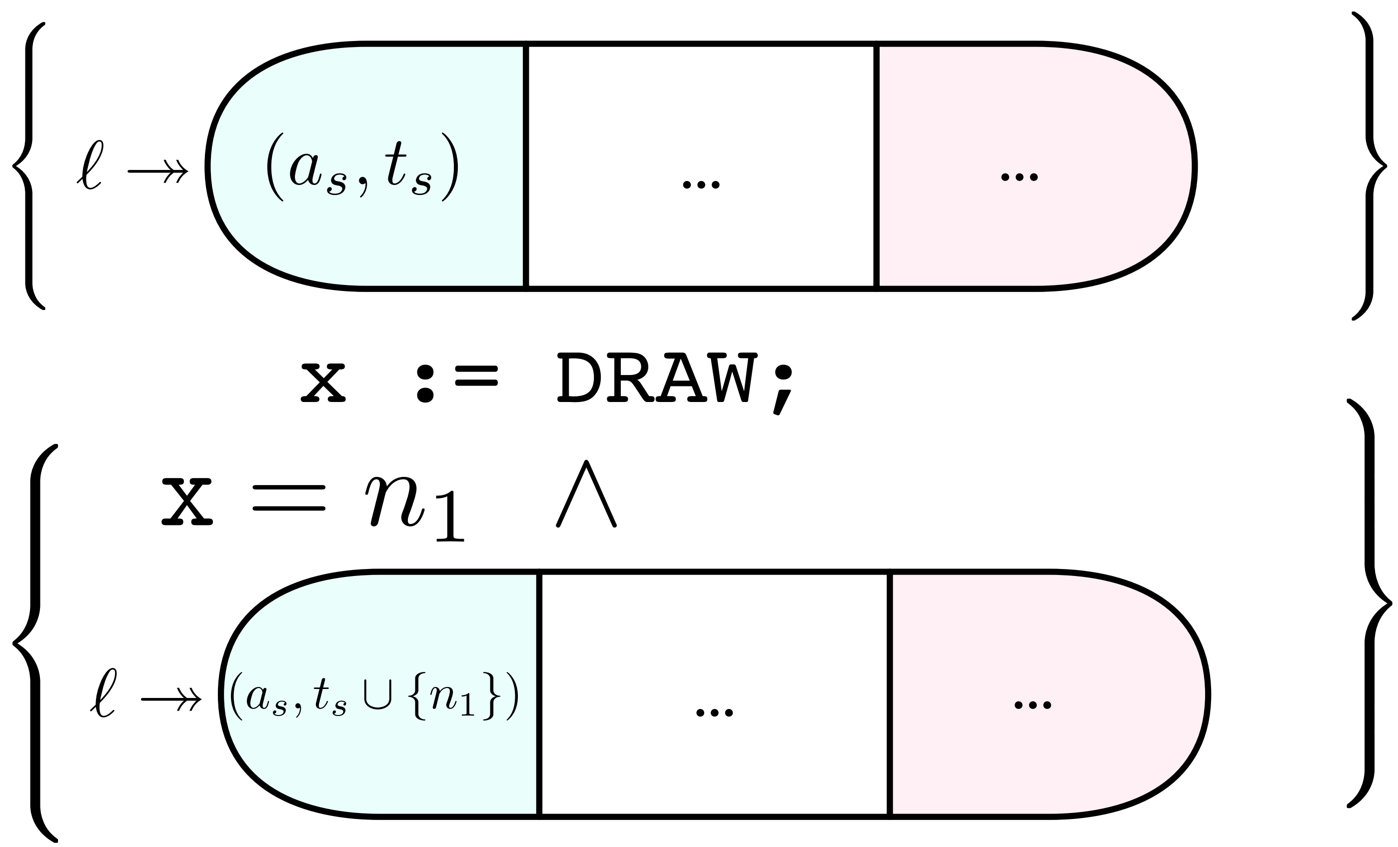
else $s' = s \land \mathsf{res} = \mathbf{false}$

What about modular reasoning?

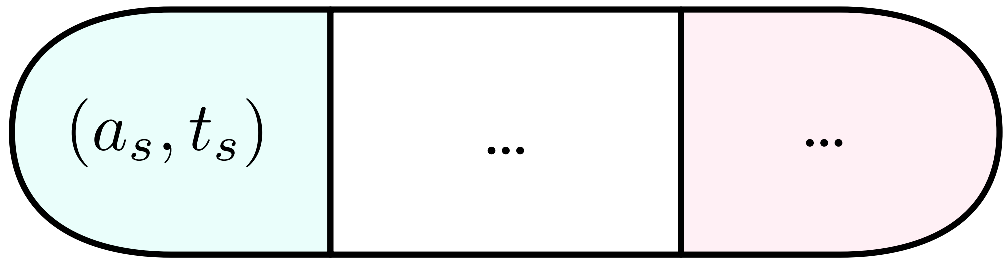
```
x := DRAW;
```



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


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

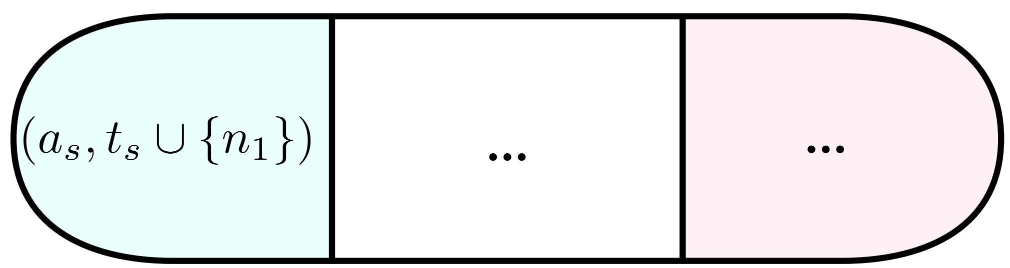
`lock = {`

$\Big\{ \ell \twoheadrightarrow$ 

$\Big\}$

`x := DRAW;`

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

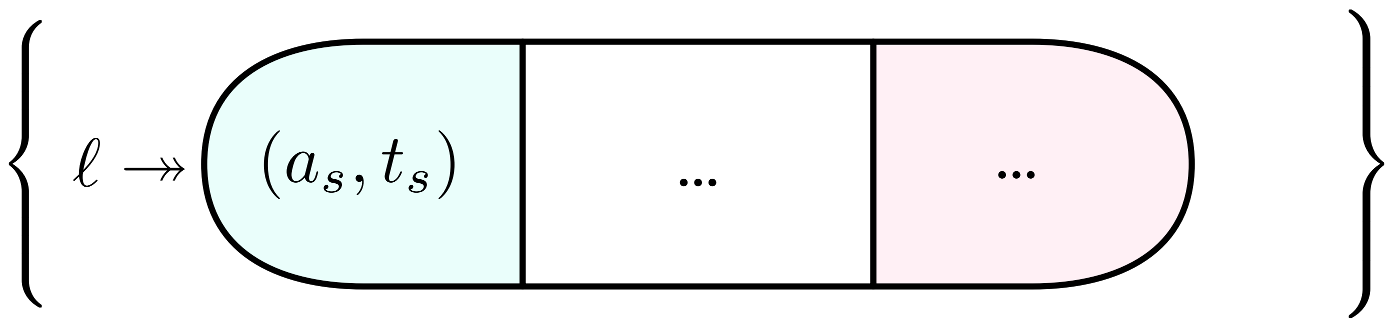
$\ell \twoheadrightarrow$ 

$\Big\}$

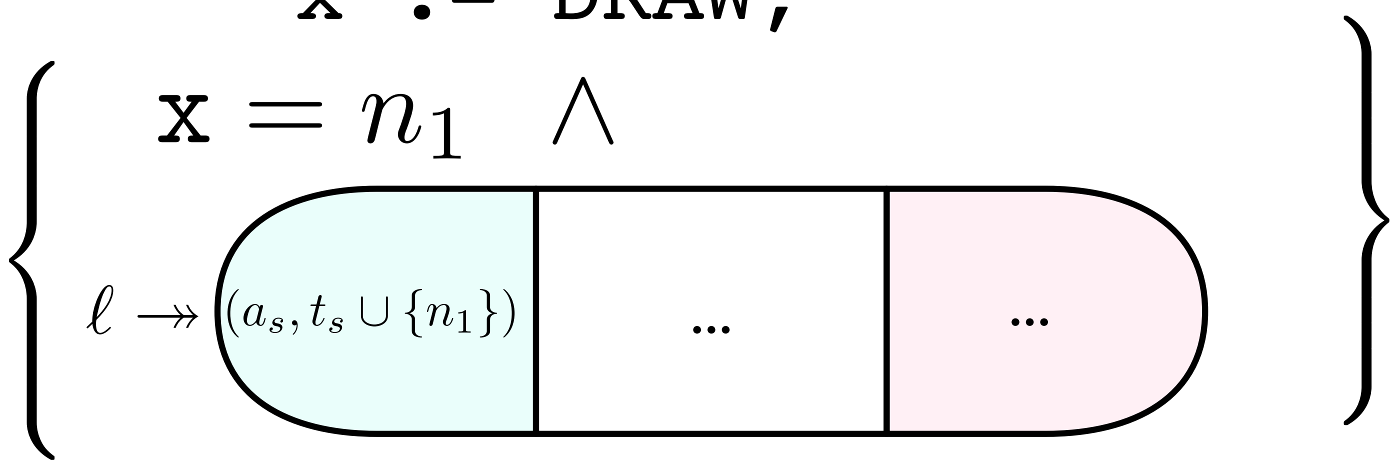
`while (!TRY(x)) SKIP;`

`}`

```
lock = {
```

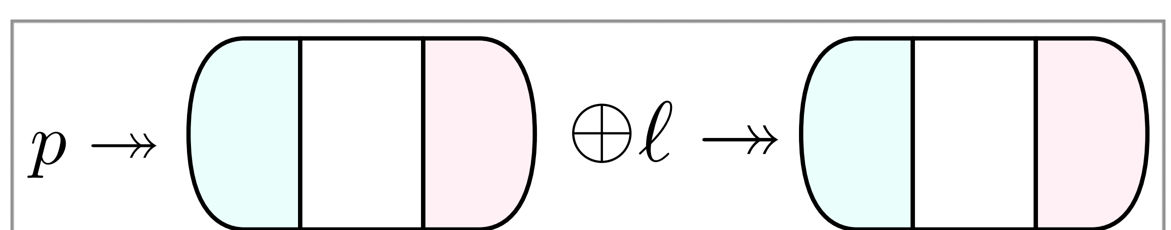



```
x := DRAW;
```



$\texttt{while } \underbrace{\texttt{(!TRY(x))}} \texttt{ SKIP;}$

Defined in




```
}
```

Context Weakening!

Injection Rule

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

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

Injection Rule

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

where $\bowtie\; = \;\bowtie, \rtimes, \ltimes, \times \ldots$

```
lock = {
```

$$\left\{\quad \ell \twoheadrightarrow \big[\,(a_s, t_s) \;\big|\; \ldots \;\big|\; \ldots\,\big] \quad\right\}$$

```
x := DRAW ;
```

$$\left\{\quad \mathtt{x} = n_1 \;\wedge\; \ell \twoheadrightarrow \big[\,(a_s, t_s \cup \{n_1\}) \;\big|\; \ldots \;\big|\; \ldots\,\big] \quad\right\}$$

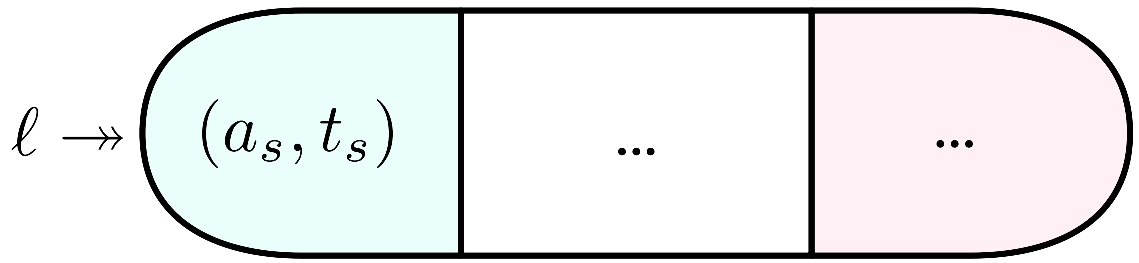
```
while (!TRY(x)) SKIP;



}
```

`lock = {`

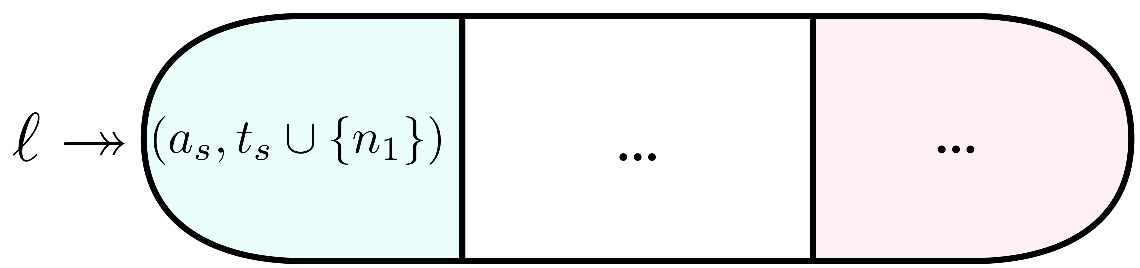
$\{$



$\}$

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

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

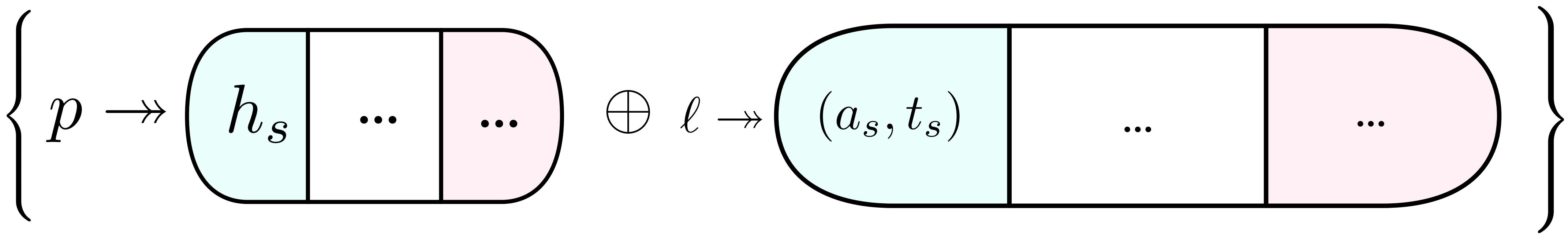


$\}$

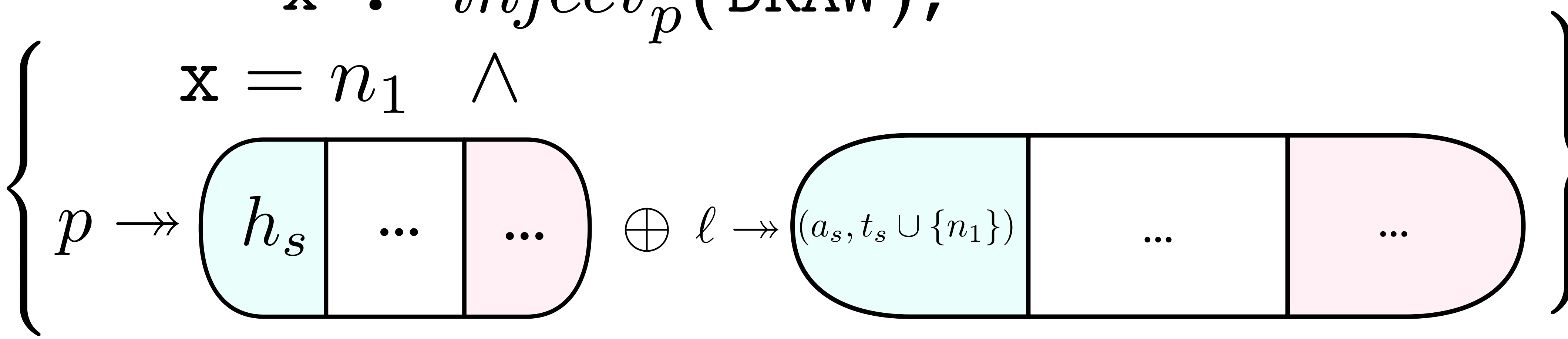
`while (!TRY(x)) SKIP;`

`}`

`lock = {`



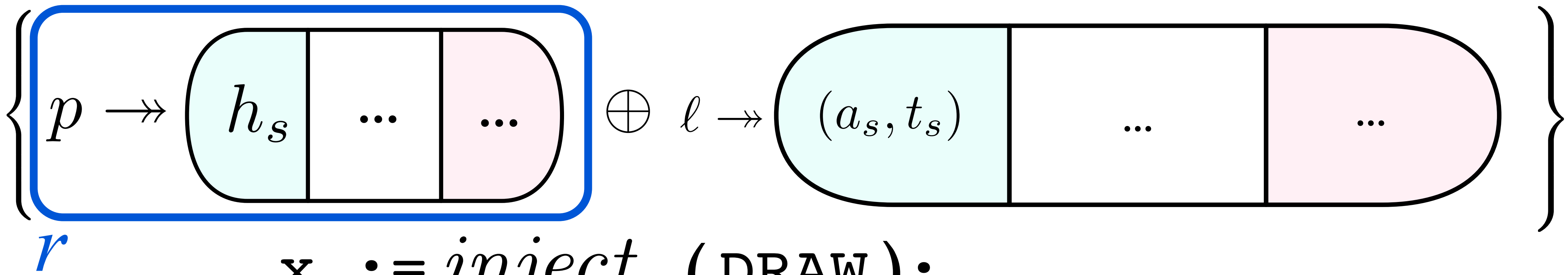
`x :=` $inject_p$`(DRAW);`



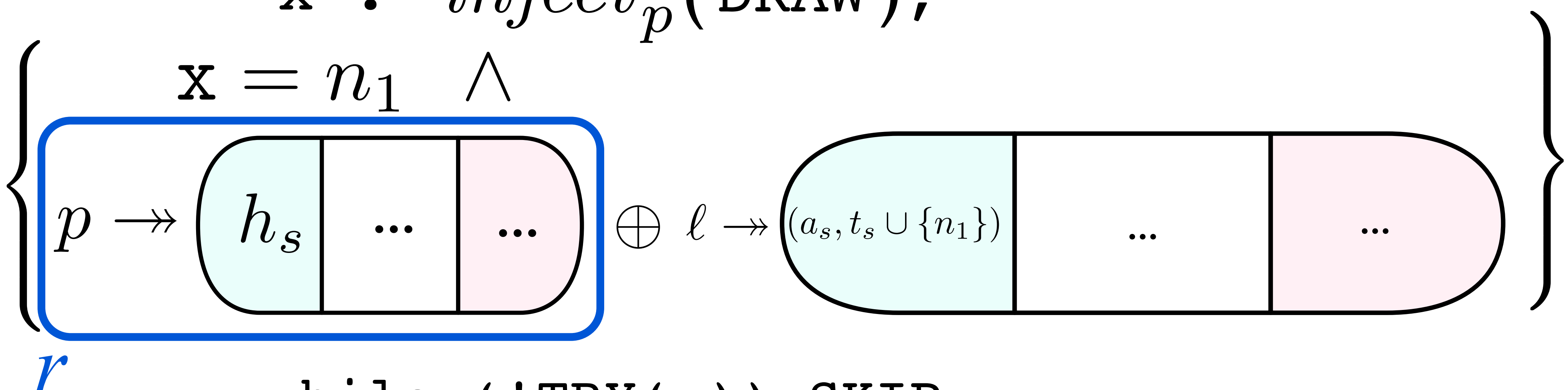

```
while (!TRY(x)) SKIP;
```

`}`

```
lock = {
```



$\texttt{x} := inject_p(\texttt{DRAW});$




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

}
```

Creating and disposing concurroids

Creating and disposing resources

CSL Resource Rule

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

CSL Resource Rule

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

CSL Resource Rule

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

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;
```

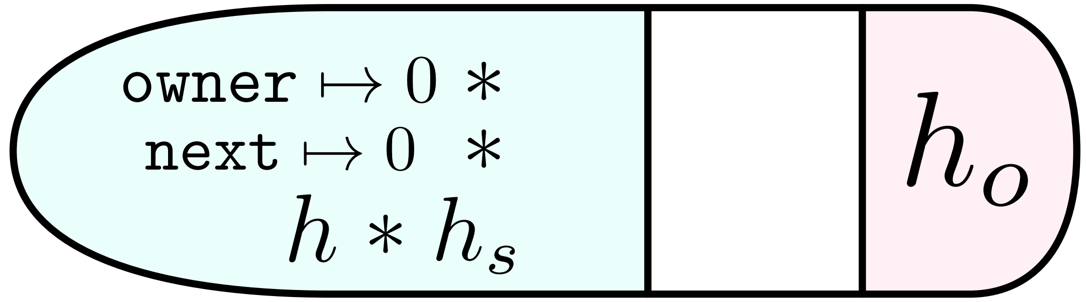
$hide_{coh_{(\mathtt{tlock}\ \ell(\mathtt{owner},\mathtt{next})),(a_s,\emptyset)}}$ `{`

```
        body;
    }
}
```

Scoped concurroid creation/disposal

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

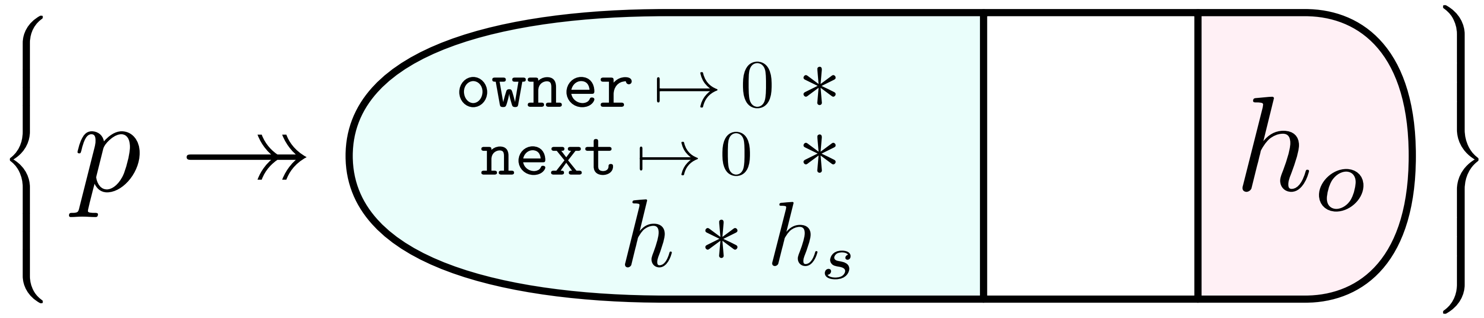
```
    body;
}
```

$\Big\{ p \twoheadrightarrow$ 

$\Big\}$

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

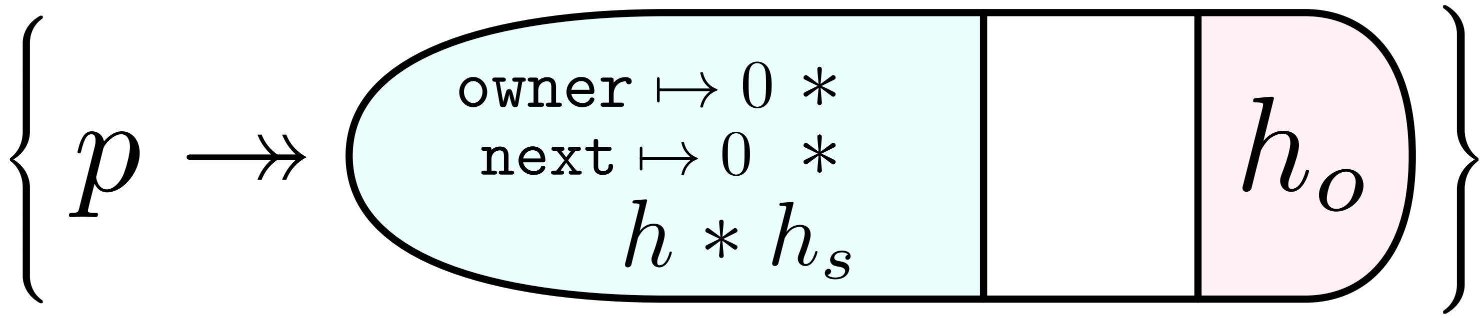
```
    body;
}
```



Concurroid spec

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

```
    body;
}
```



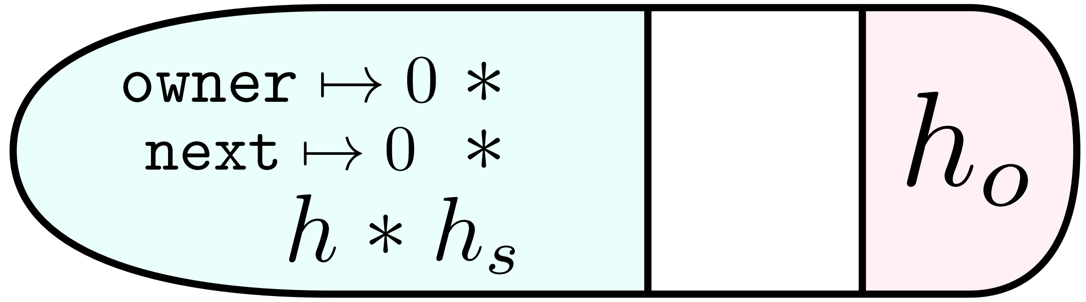
Concurroid spec

Initial "self" auxiliaries

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

```
    body;

}
```

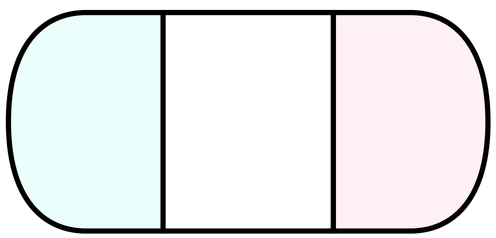
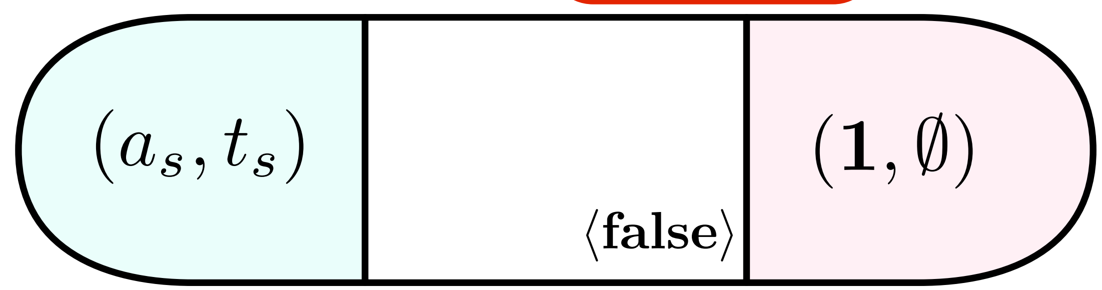
$\{\, p \twoheadrightarrow$ 

$\,\}$

Concurroid spec

Initial "self" auxiliaries

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

$\{\, p \twoheadrightarrow$  $\oplus\ \ell \twoheadrightarrow$ 

$\,\}$

```
body;
```

}

$$\left\{ p \twoheadrightarrow \left[\ \begin{matrix}\texttt{owner} \mapsto 0\ *\\ \texttt{next} \mapsto 0\ *\\ h * h_s\end{matrix}\ \middle|\ \quad \middle|\ h_o \right] \right\}$$

Concurroid spec

Initial "self" auxiliaries

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

$$\left\{ p \twoheadrightarrow \left[\ \quad \middle|\ \quad \middle|\ \quad \right] \oplus \ell \twoheadrightarrow \left[\ (a_s, t_s)\ \middle|\ \langle\mathbf{false}\rangle\ \middle|\ (\mathbf{1}, \emptyset) \right] \right\}$$

body;

}

$$\left\{ p \twoheadrightarrow \fbox{$\begin{array}{c} \texttt{owner} \mapsto 0 \; * \\ \texttt{next} \mapsto 0 \; * \\ h * h_s \end{array}$} \fbox{$\quad$} \fbox{$h_o$} \right\}$$

Concurroid spec

Initial "self" auxiliaries

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

$$\left\{ p \twoheadrightarrow \fbox{h_s} \fbox{$\quad$} \fbox{h_o} \oplus \ell \twoheadrightarrow \fbox{(a_s, t_s)} \fbox{$\begin{array}{l} \texttt{owner} \mapsto 0 \; * \\ \texttt{next} \mapsto 0 \; * \\ h \quad \langle\textbf{false}\rangle \end{array}$} \fbox{$(\mathbf{1}, \emptyset)$} \right\}$$

```
body;
```

}

$\left\{ p \twoheadrightarrow \left[\begin{array}{c|c|c} \texttt{owner} \mapsto 0 \ * & & \\ \texttt{next} \mapsto 0 \ * & & h_o \\ h * h_s & & \end{array}\right] \right\}$

Concurroid spec

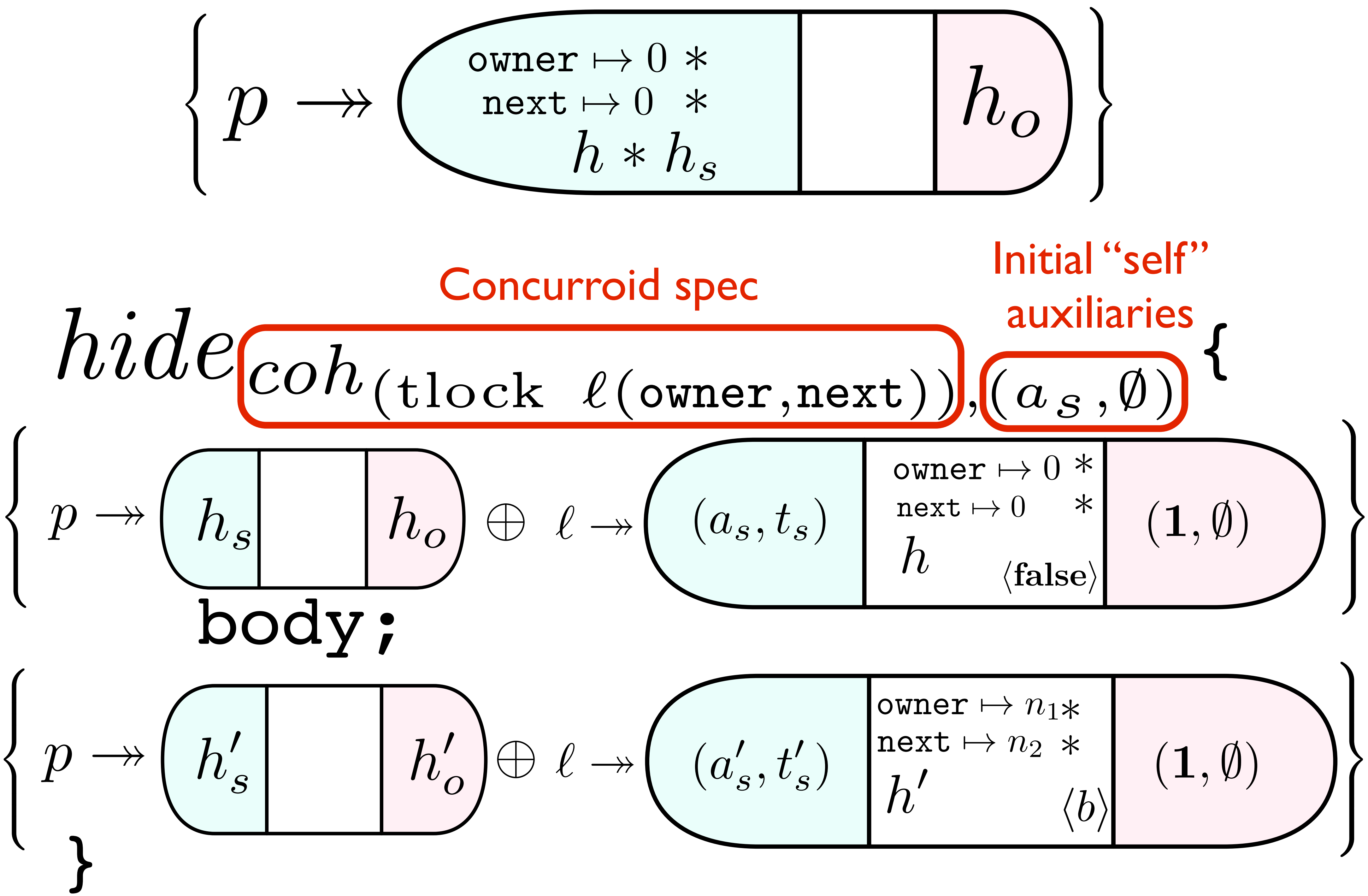
Initial "self" auxiliaries

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

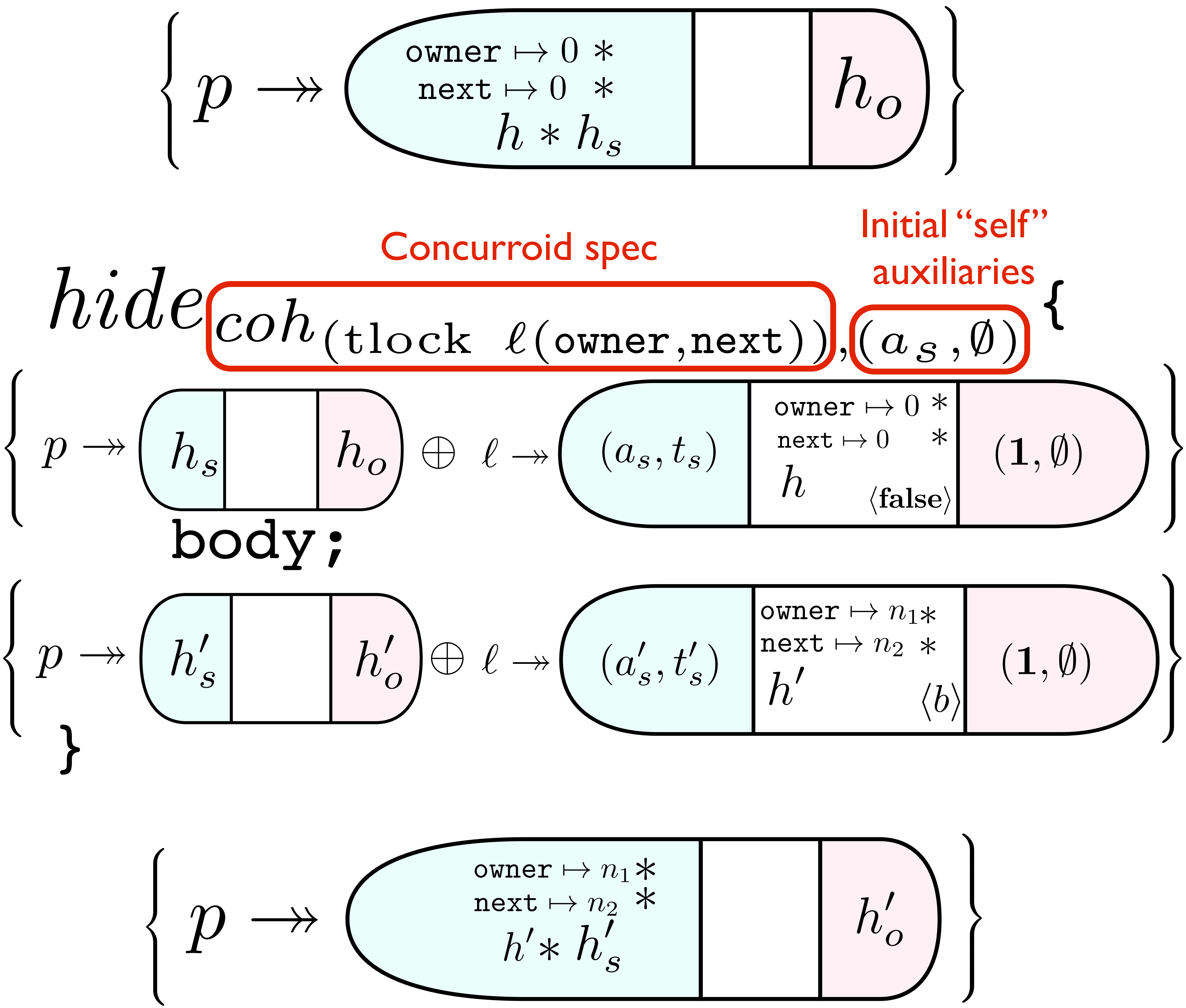
$\left\{ p \twoheadrightarrow \left[\begin{array}{c|c|c} h_s & & h_o \end{array}\right] \oplus \ \ell \twoheadrightarrow \left[\begin{array}{c|c|c} & \texttt{owner} \mapsto 0 \ * & \\ (a_s, t_s) & \texttt{next} \mapsto 0 \ \ * & (\mathbf{1}, \emptyset) \\ & h \quad \langle\mathbf{false}\rangle & \end{array}\right] \right\}$

```
body;
```

}

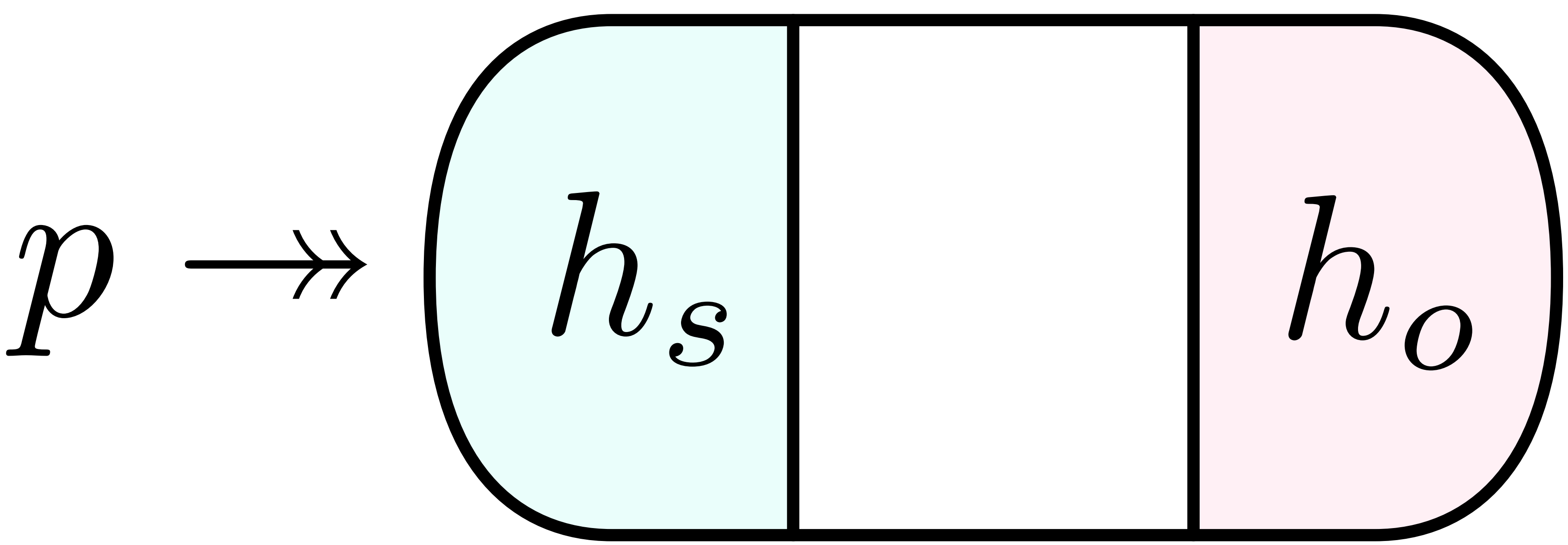


{ p ↠ owner ↦ 0 * next ↦ 0 * h * h_s | | h_o }
Concurroid spec
Initial "self" auxiliaries
hide coh(tlock ℓ(owner,next)), (a_s, ∅) {
{ p ↠ h_s | | h_o ⊕ ℓ ↠ (a_s, t_s) | owner ↦ 0 * next ↦ 0 * h ⟨false⟩ | (1, ∅) }
body;
{ p ↠ h'_s | | h'_o ⊕ ℓ ↠ (a'_s, t'_s) | owner ↦ n_1 * next ↦ n_2 * h' ⟨b⟩ | (1, ∅) }
}



owner ↦ 0 * next ↦ 0 * h * h_s
h_o
Concurroid spec
Initial "self" auxiliaries
hide coh(tlock ℓ(owner,next)), (a_s, ∅) {
p ↠ h_s h_o ⊕ ℓ ↠ (a_s, t_s) owner ↦ 0 * next ↦ 0 * h ⟨false⟩ (1, ∅)
body;
p ↠ h'_s h'_o ⊕ ℓ ↠ (a'_s, t'_s) owner ↦ n_1 * next ↦ n_2 * h' ⟨b⟩ (1, ∅)
}
p ↠ owner ↦ n_1 * next ↦ n_2 * h' * h'_s h'_o

Only One Basic Concurroid



Parallel Composition

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

Parallel Composition

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

"Fork-shuffling" is handled by subjectivity: R/G are encoded by $\boldsymbol{U}$.

Not discussed today

- A concurroid for CAS-based lock
- A concurroid model for readers/writers
- Allocation
- Non-scoped locks
- Soundness theorem and its proof
- Abstract predicates (yes, we can do it, too)

Implementation

- Implementation in Coq (metatheory, logic, proofs): shallow embedding into the CIC
- Higher-orderness and abstraction *for free*
- Reasoning in HTT-style: specifications are monadic types
- Some automation is done for splitting the state among concurroids
- CAS-lock and Ticketed lock are fully implemented

To take away

To take away

Goal 1: Model fine-grained concurrent resources with arbitrary protocols

To take away

Goal 1: Model fine-grained concurrent resources with arbitrary protocols

Goal 2: Combine simplicity and modularity of Concurrent Separation Logic with the power of Rely-Guarantee reasoning

To take away

Goal 1: Model fine-grained concurrent resources with arbitrary protocols

Goal 2: Combine simplicity and modularity of Concurrent Separation Logic with the power of Rely-Guarantee reasoning

Proposal:

Concurroids

"fine-grained resources"

To take away

Goal 1: Model fine-grained concurrent resources with arbitrary protocols

Goal 2: Combine simplicity and modularity of Concurrent Separation Logic with the power of Rely-Guarantee reasoning

Proposal:

Concurroids

"fine-grained resources"

- State Transition Systems
- Communication
- Subjectivity

To take away

Goal 1: Model fine-grained concurrent resources with arbitrary protocols

Goal 2: Combine simplicity and modularity of Concurrent Separation Logic with the power of Rely-Guarantee reasoning

Proposal:

Concurroids

"fine-grained resources"

- State Transition Systems
- Communication
- Subjectivity

Thanks!

Backup Slides

Taking the best of two worlds

CSL + FG protocols

- CSL + Permissions
  Bornat et al. [POPL'05]
- Auxiliaries for FG
  Parkinson et al. [POPL'07]
- Storable locks
  Gotsman et al. [APLAS'07]
- Concurrent Abstract Predicates (CAP)
  Dindsdale-Young et al. [ECOOP'10]
- Higher-Order CAP
  Svendsen et al. [ESOP'13]
- Impredicative CAP
  Svendsen and Birkedal [HOPE'13]
- ...

RG + Resource composition

- Separate Assume-Guarantee (SAGL)
  Feng et al. [ESOP'07]
- RGSep
  Vafeiadis and Parkinson [CONCUR'07]
- Local RG
  Feng [POPL'09]
- Deny-Guarantee
  Dods et al. [ESOP'09]
- ...

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

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

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

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

`lock;`	$\Vert$	`lock;`
`x := x + 1;`		`x := x + 1;`
$\mathbf{a_s}$ `:=` $\mathbf{a_s}$ `+ 1;`		$\mathbf{a_s}$ `:=` $\mathbf{a_s}$ `+ 1;`
`unlock;`		`unlock;`

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

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

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

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

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

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

`lock;`	`lock;`
`x := x + 1;`	`x := x + 1;`
$\mathbf{a_s}$ `:=` $\mathbf{a_s}$ `+ 1;`	$\mathbf{a_s}$ `:=` $\mathbf{a_s}$ `+ 1;`
`unlock;`	`unlock;`

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

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

$\{\ \mathbf{a}_s \mapsto 0,\ \mathbf{a}_o \mapsto \mathbf{n} + 0\ \}$	$\Vert$	$\{\ \mathbf{a}_s \mapsto 0,\ \mathbf{a}_o \mapsto \mathbf{n} + 0\ \}$
`lock;`		`lock;`
`x := x + 1;`		`x := x + 1;`
$\mathbf{a}_s$ `:=` $\mathbf{a}_s$ `+ 1;`		$\mathbf{a}_s$ `:=` $\mathbf{a}_s$ `+ 1;`
`unlock;`		`unlock;`

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

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

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

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

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

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

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

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

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

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

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

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

||

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

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

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

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

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

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

$\{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \}$	$\{\ \mathbf{a_s} \mapsto 0,\ \mathbf{a_o} \mapsto \mathbf{n} + 0\ \}$
`lock;`	`lock;`
`x := x + 1;`	`x := x + 1;`
$\mathbf{a_s}$ `:=` $\mathbf{a_s}$ `+ 1;`	$\mathbf{a_s}$ `:=` $\mathbf{a_s}$ `+ 1;`
`unlock;`	`unlock;`
$\{\ \mathbf{a_s} \mapsto 1,\ \mathbf{a_o} \mapsto \mathbf{n_1}\ \}$	$\{\ \mathbf{a_s} \mapsto 1,\ \mathbf{a_o} \mapsto \mathbf{n_2}\ \}$

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

Verifying Programs with Atomic Actions

Taming Stability

TRY(n) Action Specification

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

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

if $(n_1 = n_1')$

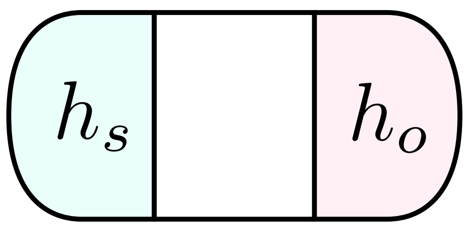
then $s' = p \twoheadrightarrow \boxed{h_s \oplus h \mid \quad \mid h_o} \oplus \ell \twoheadrightarrow \boxed{(a_s, t_s \cup \{n_1\}) \mid \mathtt{owner} \mapsto n_1 * \mathtt{next} \mapsto n_2 * \mathsf{emp} \; \langle \mathbf{true} \rangle \mid (a_o, t_o)} \;\wedge$

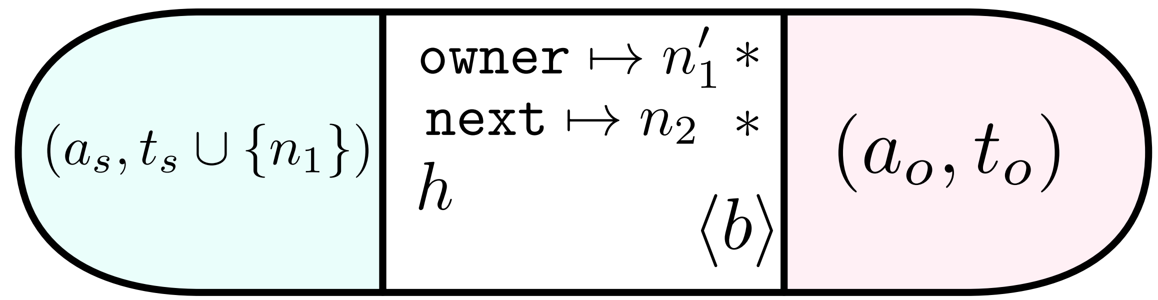
$I(a_s \oplus a_o)h \;\wedge\; \mathsf{res} = \mathbf{true}$

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

TRY(n) Action Specification

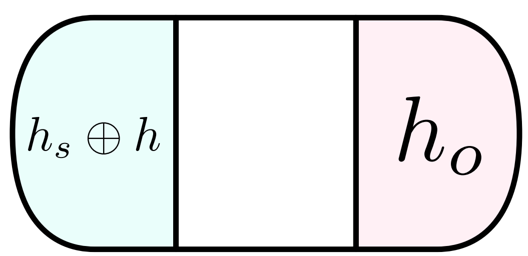
$\mathtt{TRY}(n_1)(s, s', \mathsf{res}) \triangleq$

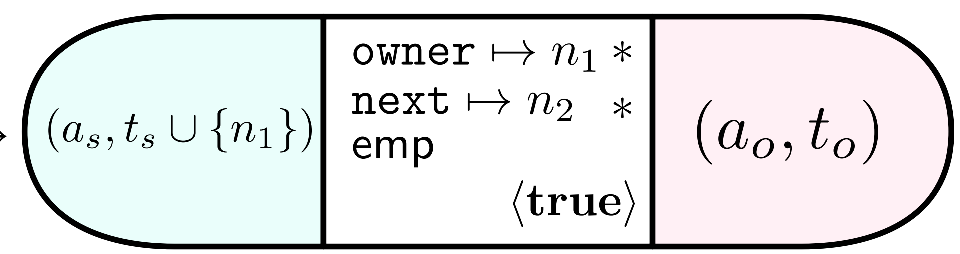
$s = p \twoheadrightarrow$ 

$\oplus\ \ell \twoheadrightarrow$ 

$\wedge$

if $(n_1 = n_1')$

then $\Big(s' = p \twoheadrightarrow$ 

$\oplus\ \ell \twoheadrightarrow$ 

$\wedge$

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

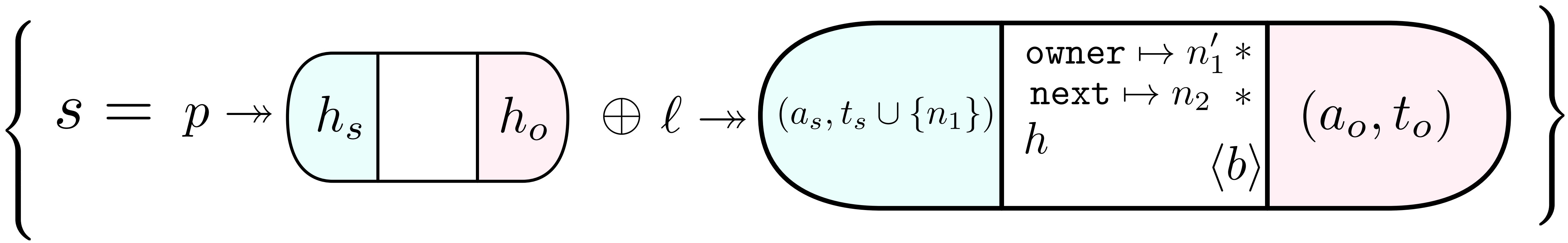
else $s' = s \ \wedge\ \mathsf{res} = \mathbf{false}$

Not stable!

Stable Hoare-style Rule for `TRY(n)`

$$\boxed{\mathtt{TRY}(n_1)}$$

Stable Hoare-style Rule for `TRY(n)`



$$\boxed{\texttt{TRY}(n_1)}$$

Stable Hoare-style Rule for `TRY(n)`

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

$$\boxed{\texttt{TRY}(n_1)}$$

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

Stable Hoare-style Rule for `TRY(n)`

$$\left\{ s = p \twoheadrightarrow \left[\begin{array}{c|c|c} h_s & & h_o \end{array}\right] \oplus \ell \twoheadrightarrow \left[\begin{array}{c|l|c} (a_s, t_s \cup \{n_1\}) & \begin{array}{l} \texttt{owner} \mapsto n_1' \ * \\ \texttt{next} \mapsto n_2 \ * \\ h \\ \hfill \langle b \rangle \end{array} & (a_o, t_o) \end{array}\right] \right\}$$

$$\boxed{\texttt{TRY}(n_1)}$$

$$\left\{ \begin{array}{l}
\text{if } \ (n_1 = n_1') \\
\text{then } \left(\begin{array}{l} \exists h_o', n_2', t_o', \\
\quad s' = p \twoheadrightarrow \left[\begin{array}{c|c|c} h_s \oplus h & & h_o' \end{array}\right] \oplus \ell \twoheadrightarrow \left[\begin{array}{c|l|c} (a_s, t_s \cup \{n_1\}) & \begin{array}{l} \texttt{owner} \mapsto n_1 \ * \\ \texttt{next} \mapsto n_2' \ * \\ \mathsf{emp} \\ \hfill \langle \mathbf{true} \rangle \end{array} & (a_o, t_o') \end{array}\right] \wedge \\
\quad I(a_s \oplus a_o)h \ \wedge \ \mathsf{res} = \mathbf{true} \ \wedge \ coh(s') \end{array} \right) \\
\text{else } \left(\begin{array}{l} \exists h_o', n_1', n_2', t_o', b', h', a_o', t_o', \\
s' = p \twoheadrightarrow \left[\begin{array}{c|c|c} h_s & & h_o' \end{array}\right] \oplus \ell \twoheadrightarrow \left[\begin{array}{c|l|c} (a_s, t_s \cup \{n_1\}) & \begin{array}{l} \texttt{owner} \mapsto n_1' \ * \\ \texttt{next} \mapsto n_2' \ * \\ h' \\ \hfill \langle b' \rangle \end{array} & (a_o', t_o') \end{array}\right] \wedge coh(s') \end{array} \right)
\end{array} \right\}$$

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."

Some General Coherence Properties

- Heap-consistency:

$h_{self} \oplus h_{other} \oplus h_{shared}$ is defined

- Self-other consistency:

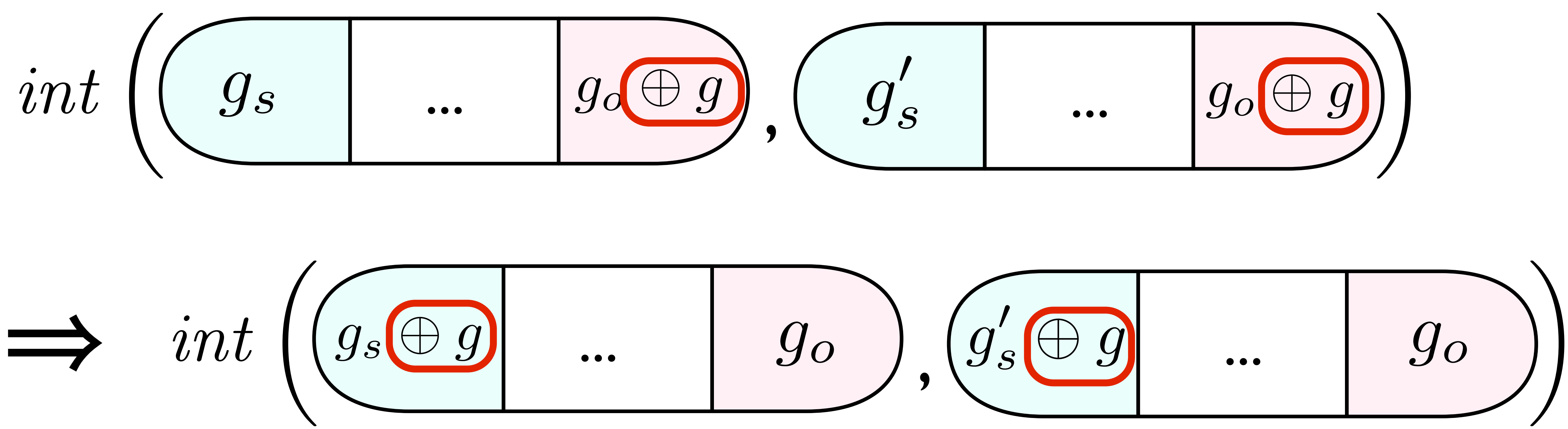
$(a_s, t_s) \oplus (a_o, t_o)$ is defined

- Fork-join consistency:

$$coh\left(\left(\begin{array}{c|c|c} g_s \oplus g & \ldots & g_o \end{array}\right)\right) \iff coh\left(\left(\begin{array}{c|c|c} g_s & \ldots & g_o \oplus g \end{array}\right)\right)$$

Internal Transition Properties

- Coherence-consistency: $int(s_1, s_2) \Rightarrow coh(s_1) \wedge coh(s_2)$
- Reflexivity
- Heap footprint preservation:
  $int(s_1, s_2) \Rightarrow \mathrm{dom}(heap(s_1)) = \mathrm{dom}(heap(s_2))$
- *Self*-locality: $int(s_1, s_2) \Rightarrow other(s_1) = other(s_2)$
- Fork-join consistency:



Acquire/Release Properties

- Coherence-consistency:

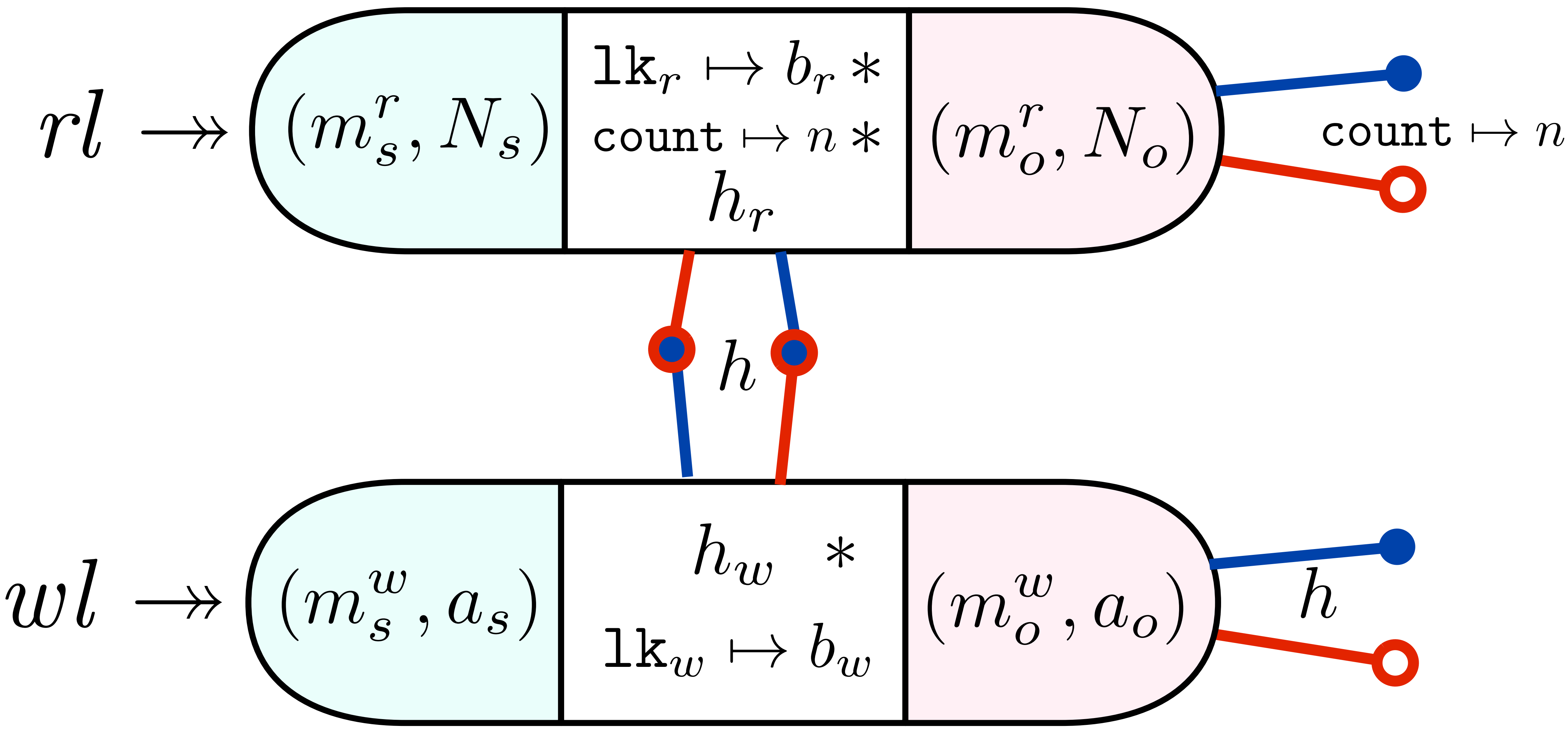
$$acq(s_1, s_2) \Rightarrow coh(s_1) \wedge coh(s_2)$$

- *Self*-locality:

$$acq(s_1, s_2) \Rightarrow other(s_1) = other(s_2)$$

- Fork-join consistency

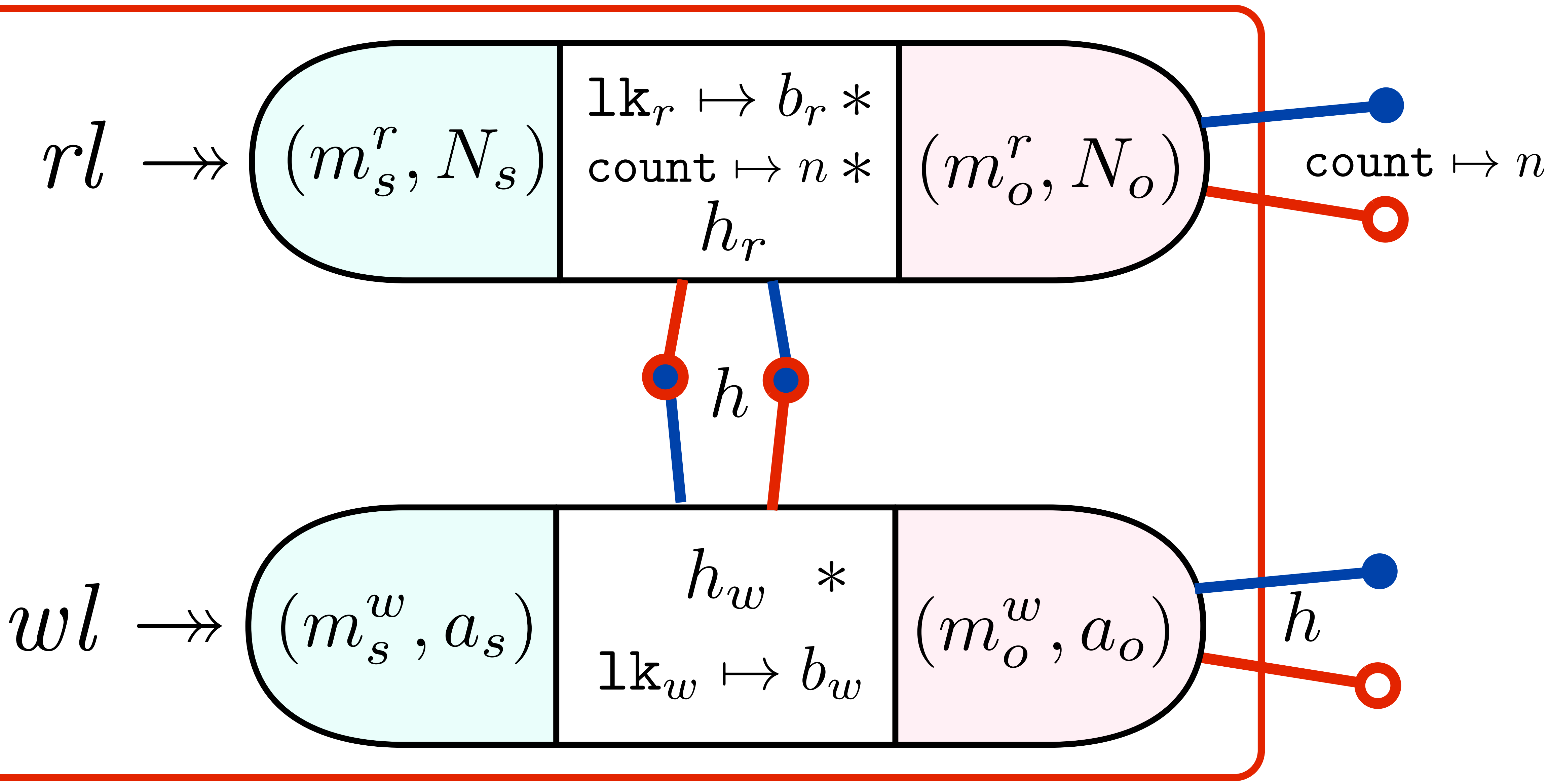
Readers-Writers



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

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

Readers-Writers



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

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