Verification of Chase-Lev work-stealing deque

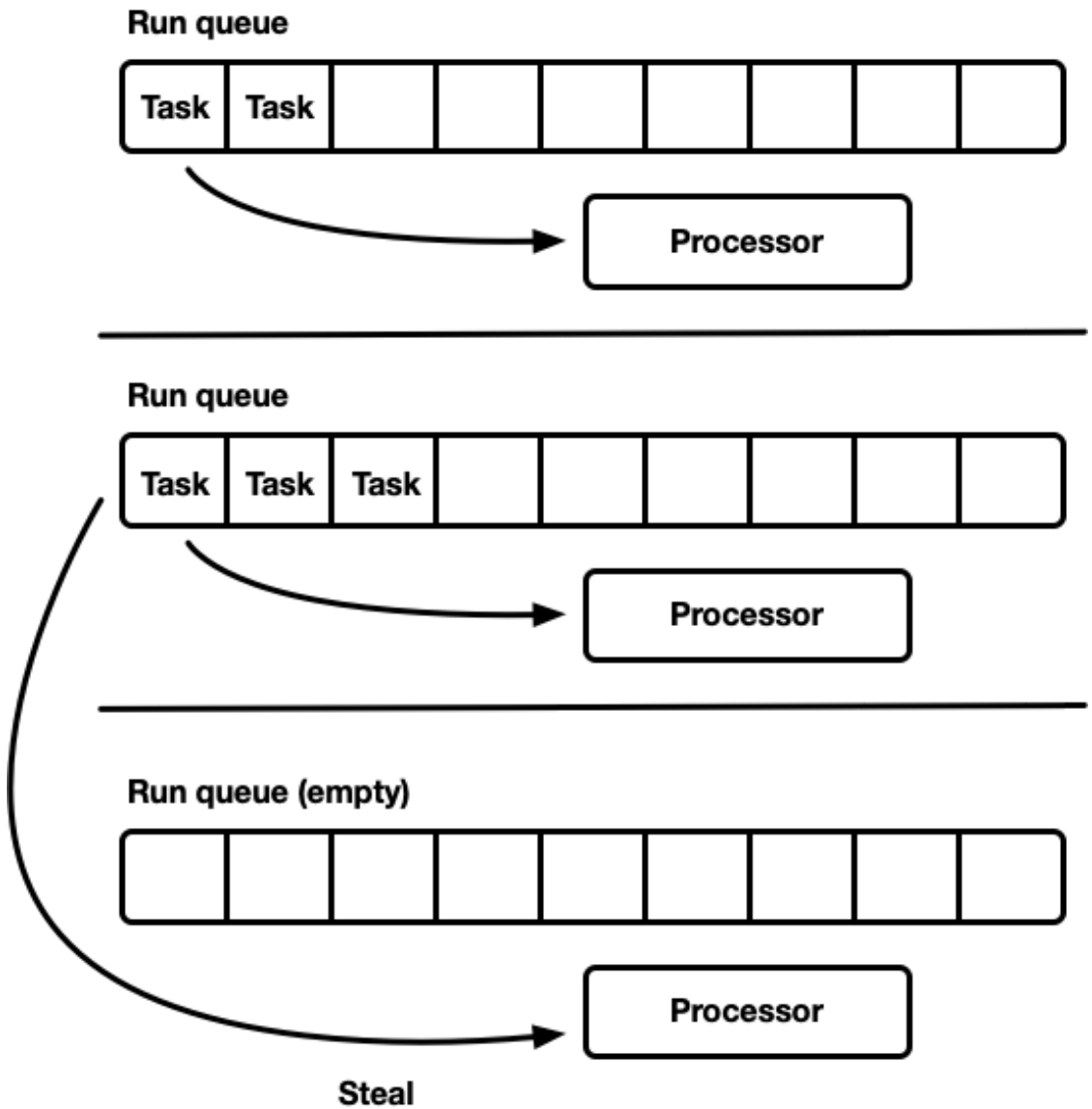
Clément Allain
François Pottier

May 1, 2023

Verification of a scheduler

```
let rec fib pool n =
  if n < 2 then 1 else
  let r1 = async pool (fun () -> fib_par (n - 1)) in
  let r2 = async pool (fun () -> fib_par (n - 2)) in
  await pool r1 + await pool r2
```

Work-stealing



Work-stealing algorithms

1. Frigo, Leiserson & Randall (1998)
   - at the core of Cilk 5
   - lock
2. Arora, Blumofe & Plaxton (2001)
   - no lock
   - one fixed size array (not circular), can overflow
3. Hendler, Lev & Shavit (2004)
   - no lock
   - list of small size arrays, no overflow
   - memory leak?
4. Chase & Lev (2005)
   - no lock
   - circular arrays, no overflow

Why is it interesting?

- demonstration of Iris on a (simplified) real-life concurrent data structure
- rich ghost state to enforce a subtle protocol
  - logical state $\neq$ physical state
  - external future-dependent linearization point
- use of prophecy variables (with memory)

The rest of this talk

- specification using logically atomic triples
- rough idea of how the data structure works
- why we need prophecy variables (with memory)

Specification — `chaselev_make`

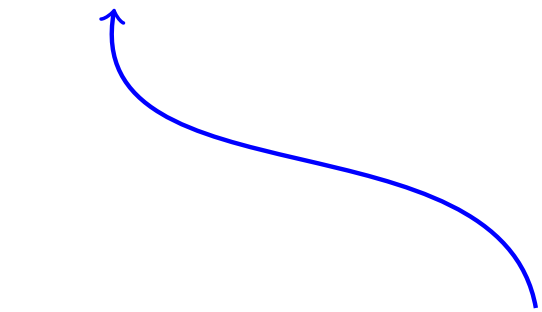
$$\frac{\{\,\text{True}\,\}}{\texttt{chaselev\_make ()}}$$

$$\left\{\, \lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-model}\ t\ []\ *\ \text{chaselev-owner}\ t \,\right\}$$

Specification — `chaselev_make`

$$\frac{\{\,\text{True}\,\}}{\texttt{chaselev\_make ()}}$$

$$\left\{\,\lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-model}\ t\ []\ *\ \text{chaselev-owner}\ t\,\right\}$$



enforces a protocol (using an Iris invariant)

Specification — `chaselev_make`

$$\frac{\{\,\text{True}\,\}}{\texttt{chaselev\_make ()}}$$

$$\left\{\, \lambda\, t \cdot \ \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-model}\ t\ []\ *\ \text{chaselev-owner}\ t \,\right\}$$

asserts the list of values that the deque (logically) contains

Specification — `chaselev_make`

$$\frac{\{\,\text{True}\,\}}{\texttt{chaselev\_make ()}}$$

$$\left\{\,\lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-model}\ t\ []\ *\ \text{chaselev-owner}\ t\,\right\}$$

gives the owner exclusive access to his end of the deque

Specification — `chaselev_push`

$$\left\{ \text{chaselev-inv } t\ \iota\ *\ \text{chaselev-owner } t \right\}$$

$$\left\langle \forall\, vs \cdot\ \text{chaselev-model } t\ vs \right\rangle$$

$$\texttt{chaselev\_push}\ t\ v,\ \uparrow \iota$$

$$\left\langle \exists \cdot\ \text{chaselev-model } t\ (vs +\!\!+ [v]) \right\rangle$$

$$\left\{ \lambda\, () \cdot\ \text{chaselev-owner } t \right\}$$

Specification — `chaselev_push`

Specification of a concurrent operation ($\simeq$ transaction):
standard triple + logically atomic triple

$$\frac{\frac{\frac{\frac{\{\, P \,\}}{\langle\, \forall\, \overline{x} \cdot\ P_{\text{lin}} \,\rangle}}{e,\ \mathcal{E}}}{\langle\, \exists\, \overline{y} \cdot\ Q_{\text{lin}} \,\rangle}}{\{\, \lambda\, res \cdot\ Q \,\}}$$

P : private precondition

Q : private postcondition

P_{lin} : public precondition

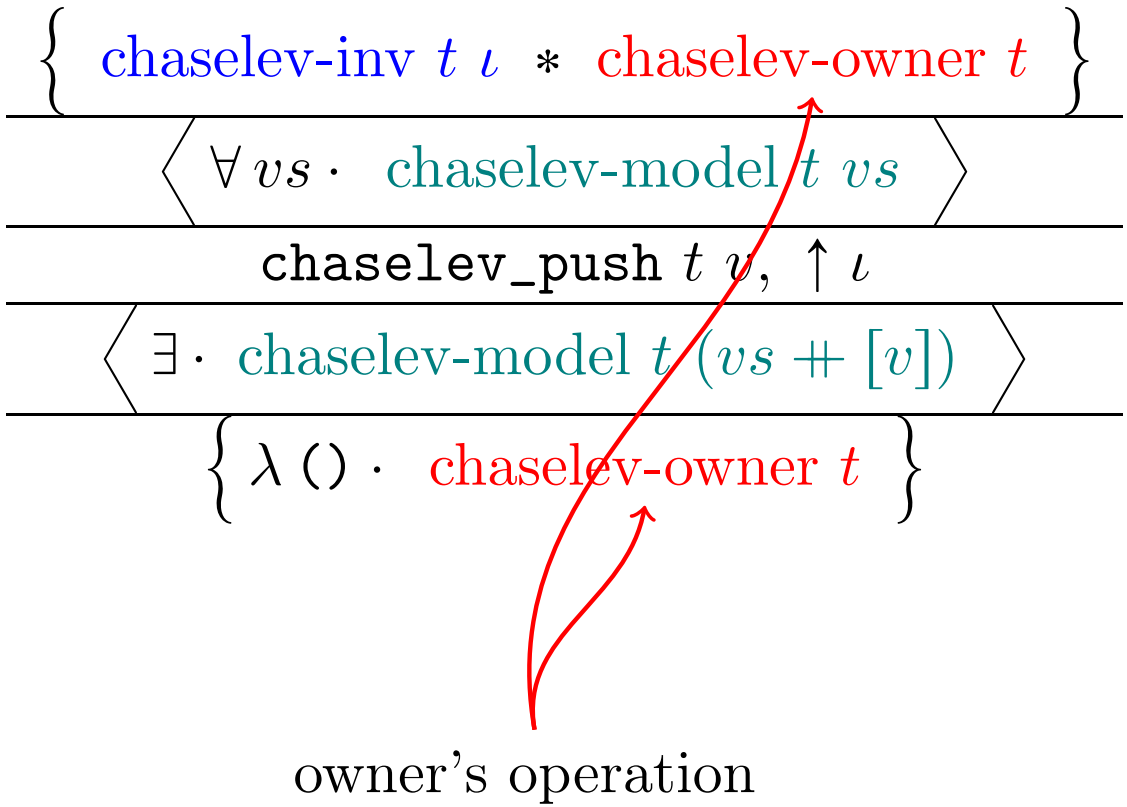
Q_{lin} : public postcondition

Specification — `chaselev_push`

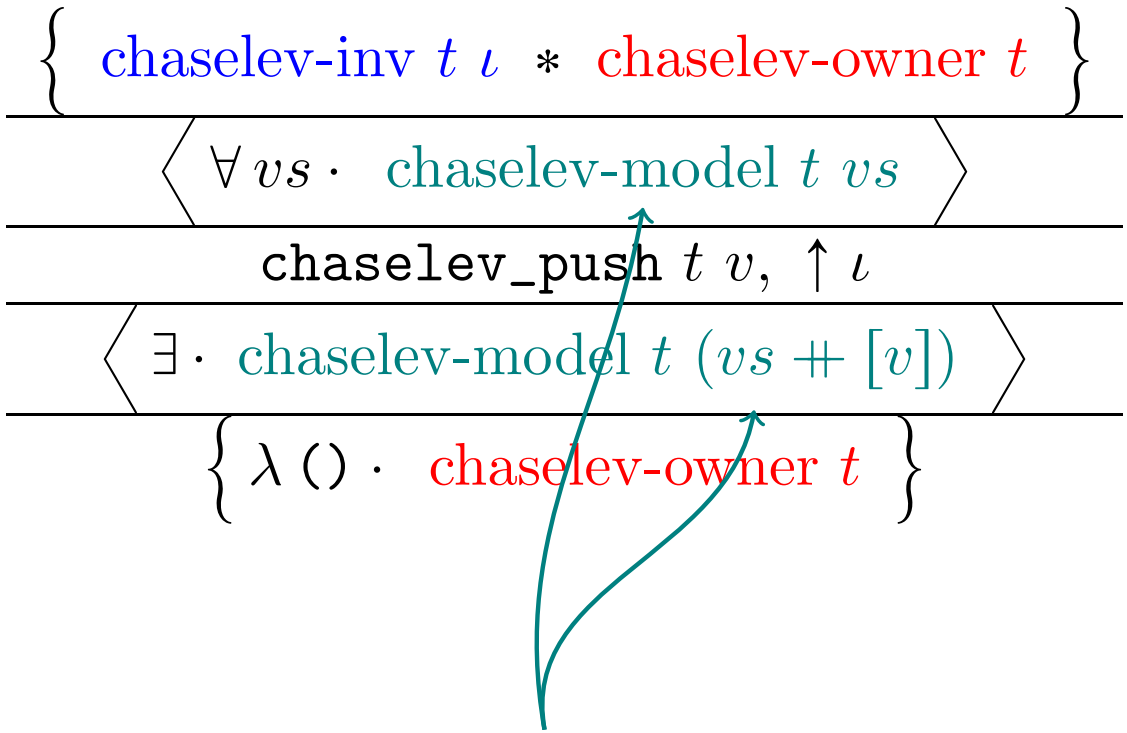
For a concurrent data structure:

$$\frac{\{\ \text{???-inv}\cdots * P\ \}}{\frac{\langle\ \forall\, \overline{x}\cdot\ \text{???-model}\cdots\ \rangle}{\frac{e,\ \mathcal{E}}{\frac{\langle\ \exists\, \overline{y}\cdot\ \text{???-model}\cdots\ \rangle}{\{\ \lambda\, res\cdot\ Q\ \}}}}}$$

Specification — `chaselev_push`



Specification — `chaselev_push`



v is atomically pushed at the owner's end

Specification — `chaselev_pop`

$$\left\{ \text{chaselev-inv } t\ \iota \ *\ \text{chaselev-owner } t \right\}$$

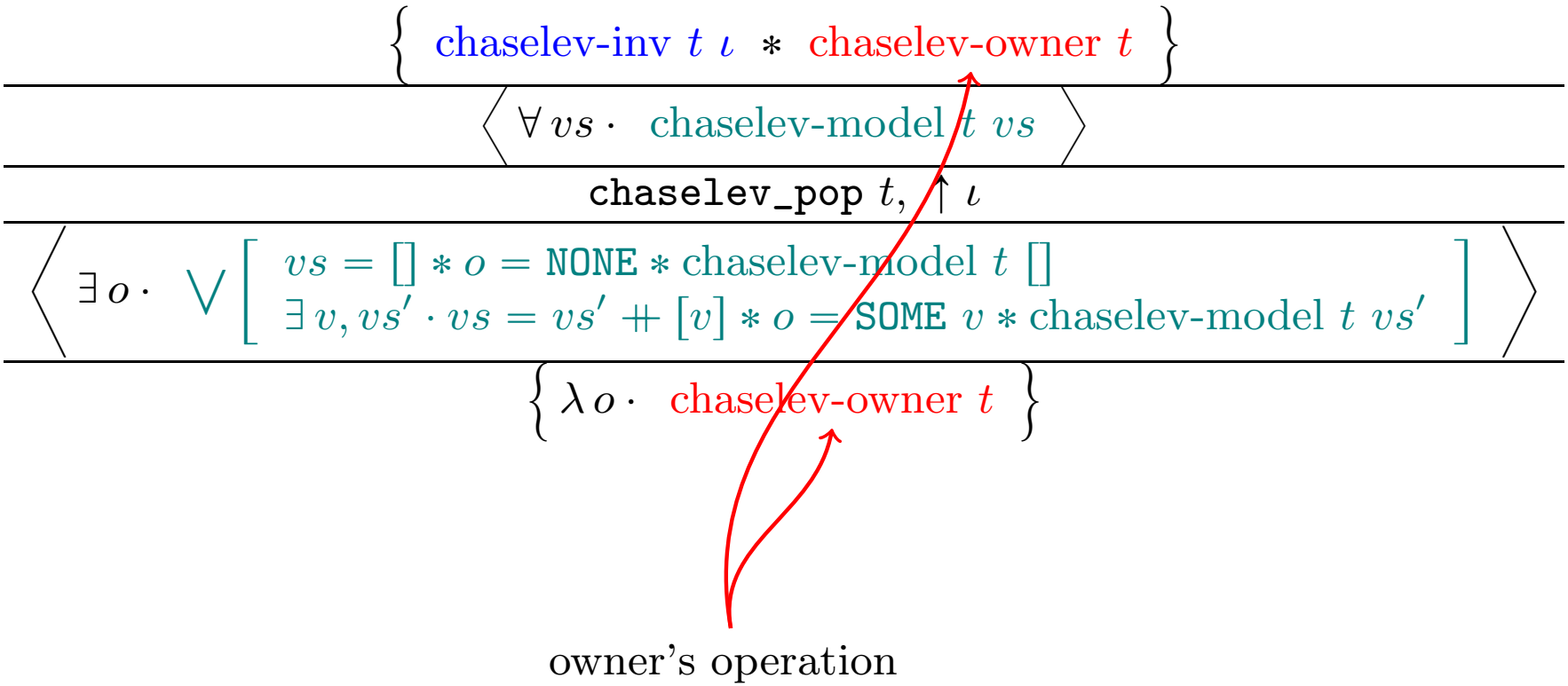
$$\left\langle \forall\, vs \cdot \ \text{chaselev-model } t\ vs \right\rangle$$

$$\texttt{chaselev\_pop}\ t,\ \uparrow \iota$$

$$\left\langle \exists\, o \cdot \ \bigvee \left[\begin{array}{l} vs = [] * o = \texttt{NONE} * \text{chaselev-model } t\ [] \\ \exists\, v, vs' \cdot vs = vs' +\!\!+ [v] * o = \texttt{SOME}\ v * \text{chaselev-model } t\ vs' \end{array} \right] \right\rangle$$

$$\left\{ \lambda\, o \cdot \ \text{chaselev-owner } t \right\}$$

Specification — `chaselev_pop`



Specification — `chaselev_pop`

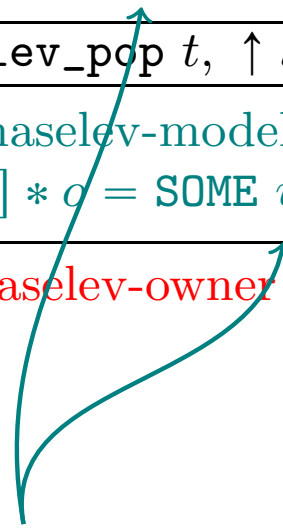
$$\left\{ \text{chaselev-inv } t\ \iota\ *\ \text{chaselev-owner } t \right\}$$

$$\left\langle \forall\, vs \cdot\ \text{chaselev-model } t\ vs \right\rangle$$

$$\texttt{chaselev\_pop}\ t,\ \uparrow \iota$$

$$\left\langle \exists\, o \cdot\ \bigvee \left[\begin{array}{l} vs = [] * o = \texttt{NONE} * \text{chaselev-model } t\ [] \\ \exists\, v, vs' \cdot vs = vs' +\!\!+ [v] * o = \texttt{SOME}\ v * \text{chaselev-model } t\ vs' \end{array} \right] \right\rangle$$

$$\left\{ \lambda\, o \cdot\ \text{chaselev-owner } t \right\}$$



either 1) the deque is seen empty
or 2) some value v is atomically popped at the owner's end

Specification — `chaselev_steal`

$$\left\{ \text{chaselev-inv } t\ \iota \right\}$$

$$\left\langle \forall\, vs \cdot \ \text{chaselev-model } t\ vs \right\rangle$$

$$\texttt{chaselev\_steal}\ t,\ \uparrow \iota$$

$$\left\langle \exists\, o \cdot \ \bigvee \left[\begin{array}{l} vs = [] * o = \texttt{NONE} * \text{chaselev-model } t\ [] \\ \exists\, v, vs' \cdot vs = v :: vs' * o = \texttt{SOME}\ v * \text{chaselev-model } t\ vs' \end{array} \right] \right\rangle$$

$$\{ \lambda\, o \cdot \ \text{True} \}$$

Specification — `chaselev_steal`

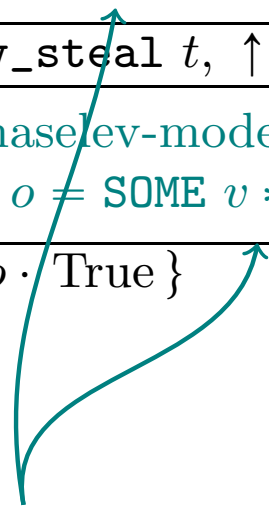
$$\left\{ \text{chaselev-inv } t\ \iota \right\}$$

$$\left\langle \forall\, vs \cdot \quad \text{chaselev-model } t\ vs \right\rangle$$

$$\texttt{chaselev\_steal}\ t,\ \uparrow \iota$$

$$\left\langle \exists\, o \cdot \quad \bigvee \left[\begin{array}{l} vs = [] * o = \texttt{NONE} * \text{chaselev-model } t\ [] \\ \exists\, v, vs' \cdot vs = v :: vs' * o = \texttt{SOME}\ v * \text{chaselev-model } t\ vs' \end{array} \right] \right\rangle$$

$$\{\, \lambda\, o \cdot \text{True} \,\}$$

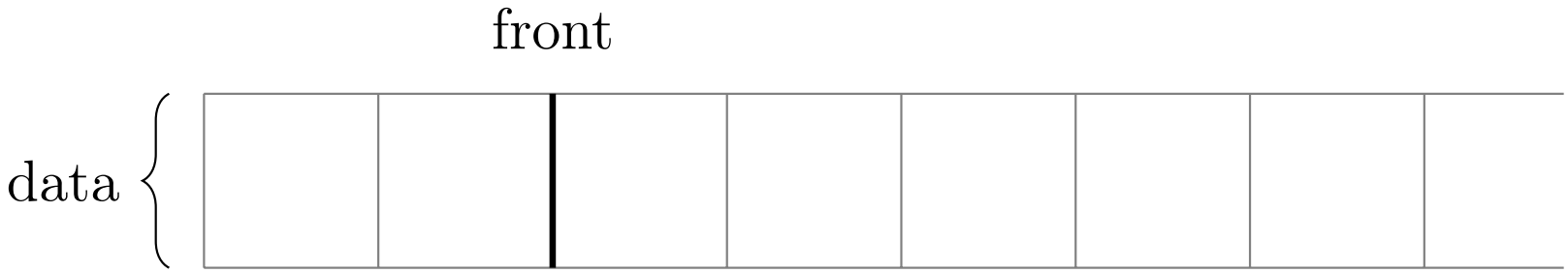


either 1) the deque is seen empty
or 2) some value v is atomically popped at the thieves' end

Physical state

data: infinite array storing all values

Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

Physical state

data

data: in

front: m

ChaselevFrontValid

γ.front $\bullet\ front_1$ $\quad$ γ.front $\circ\ front_2$

$$front2 \leqslant front1$$

ChaselevFrontUpdate

$front \leqslant front'$ $\quad$ γ.front $\bullet\ front$

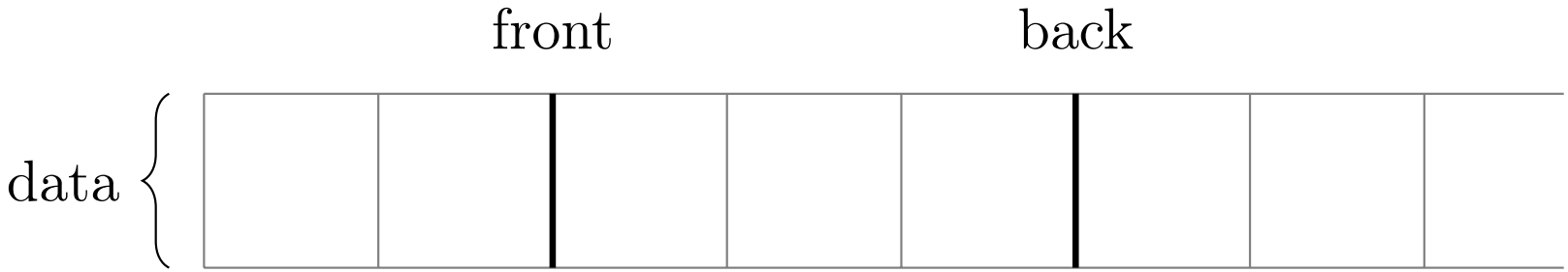
γ.front $\bullet\ front'$

ChaselevFrontFragGet

γ.front $\bullet\ front$

γ.front $\circ\ front$

Physical state

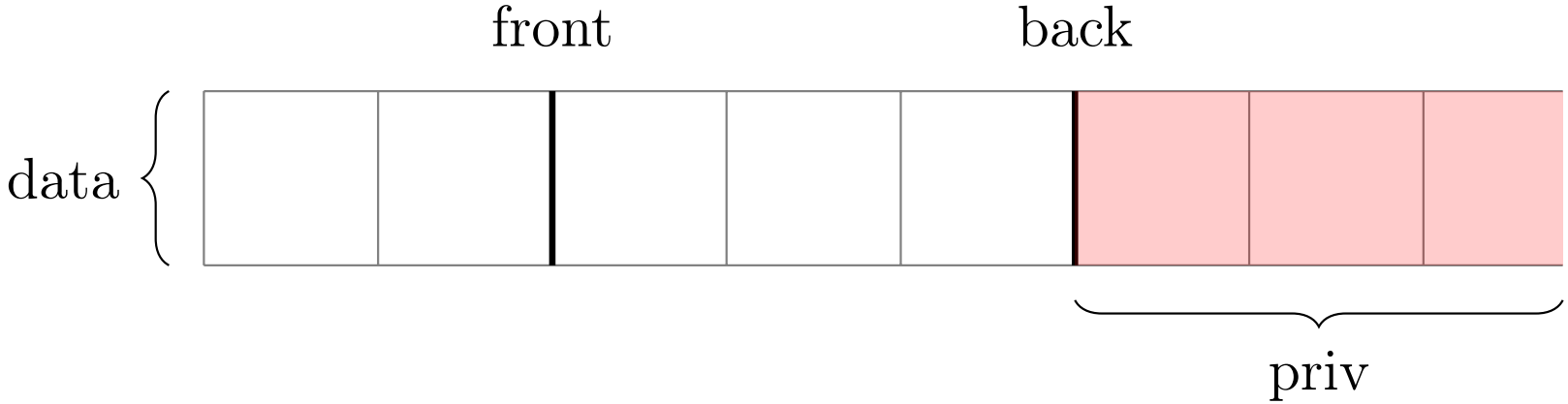


data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

Physical state



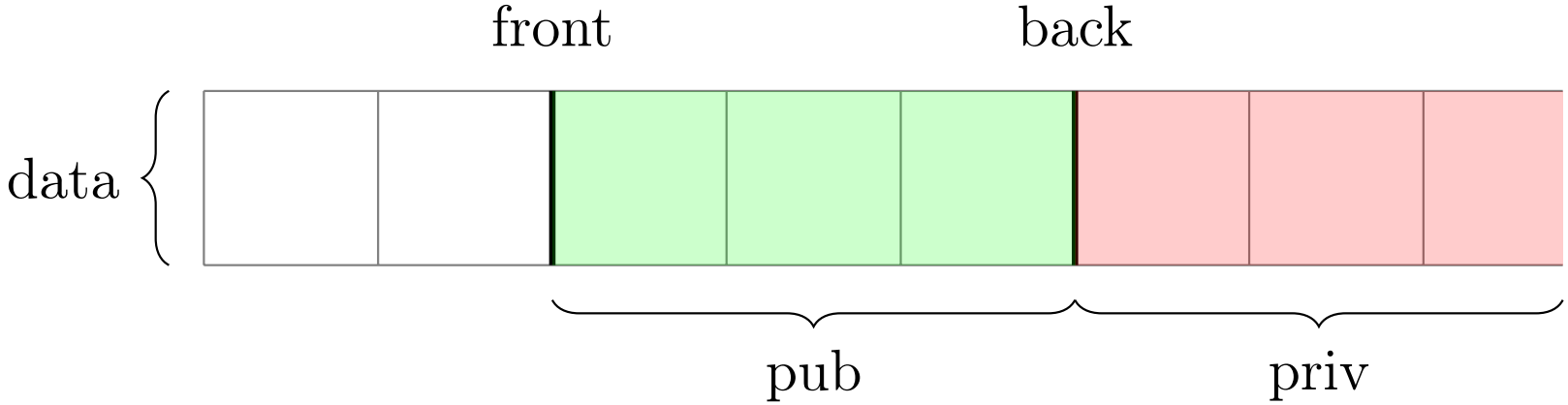
data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

priv: list of private values (controlled by owner)

Physical state



data: infinite array storing all values

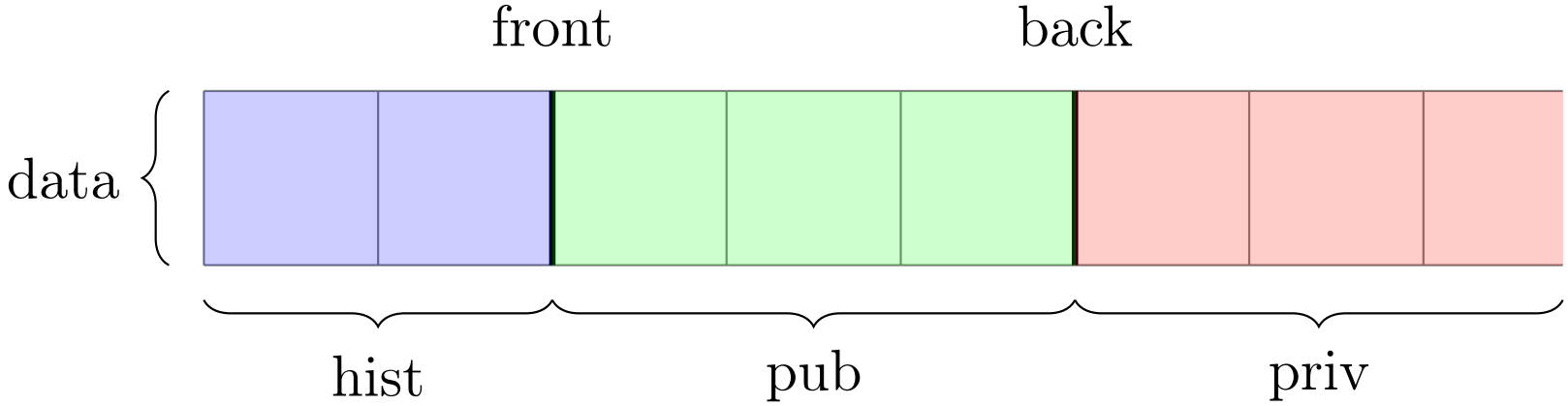
front: *monotone* index for thieves' end

back: index for owner's end

priv: list of private values (controlled by owner)

pub: list of public values (= model)

Physical state



data: infinite array storing all values

front: *monotone* index for thieves' end

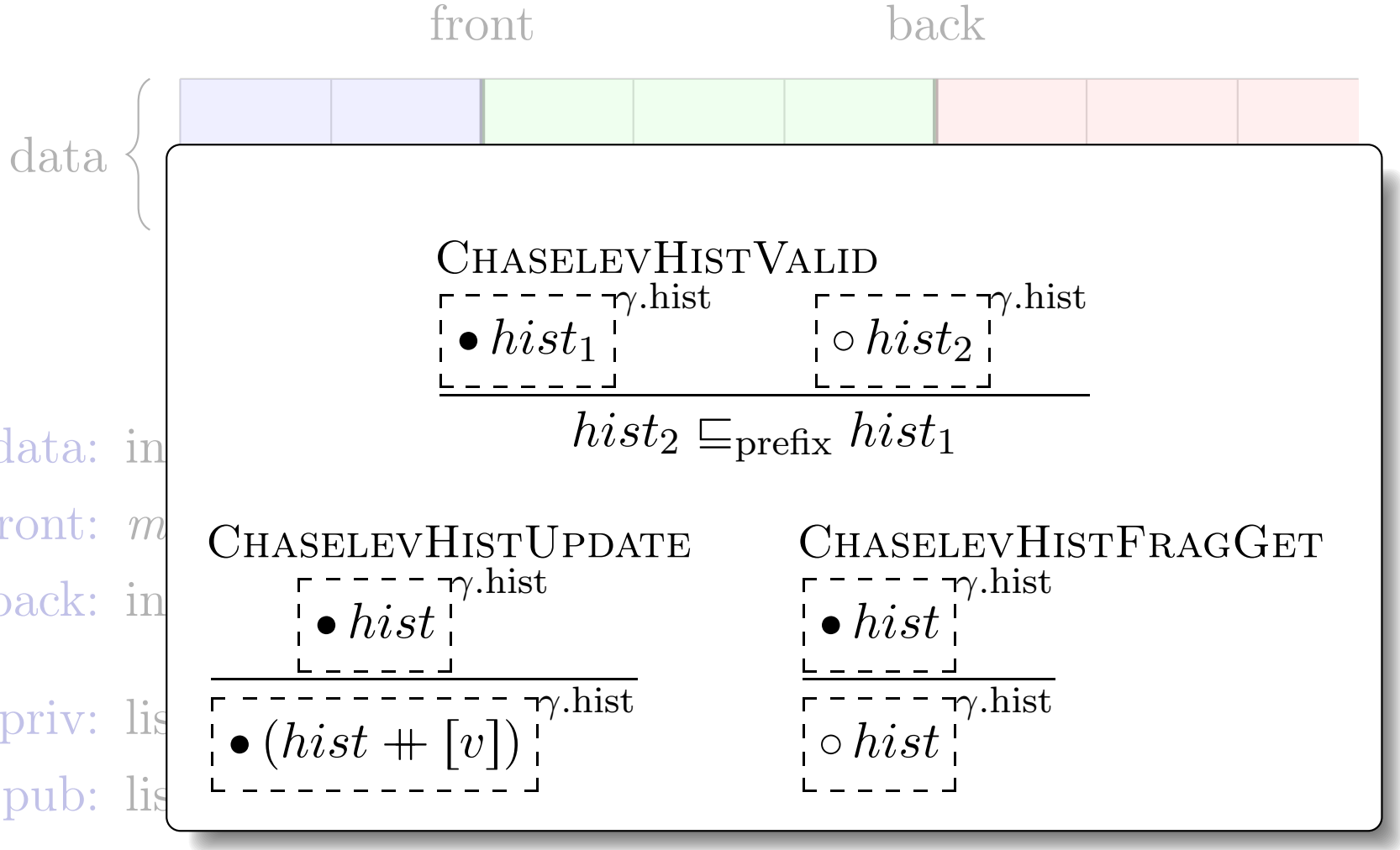
back: index for owner's end

priv: list of private values (controlled by owner)

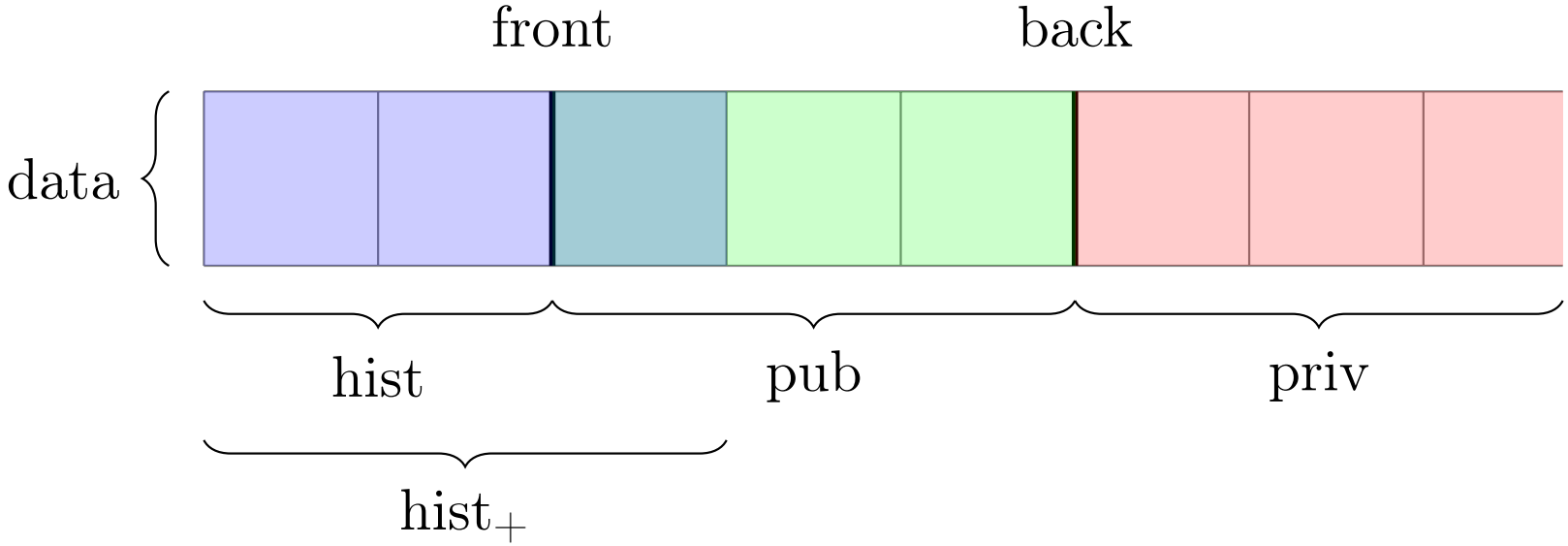
pub: list of public values (= model)

hist: *monotone* list of history values

Physical state



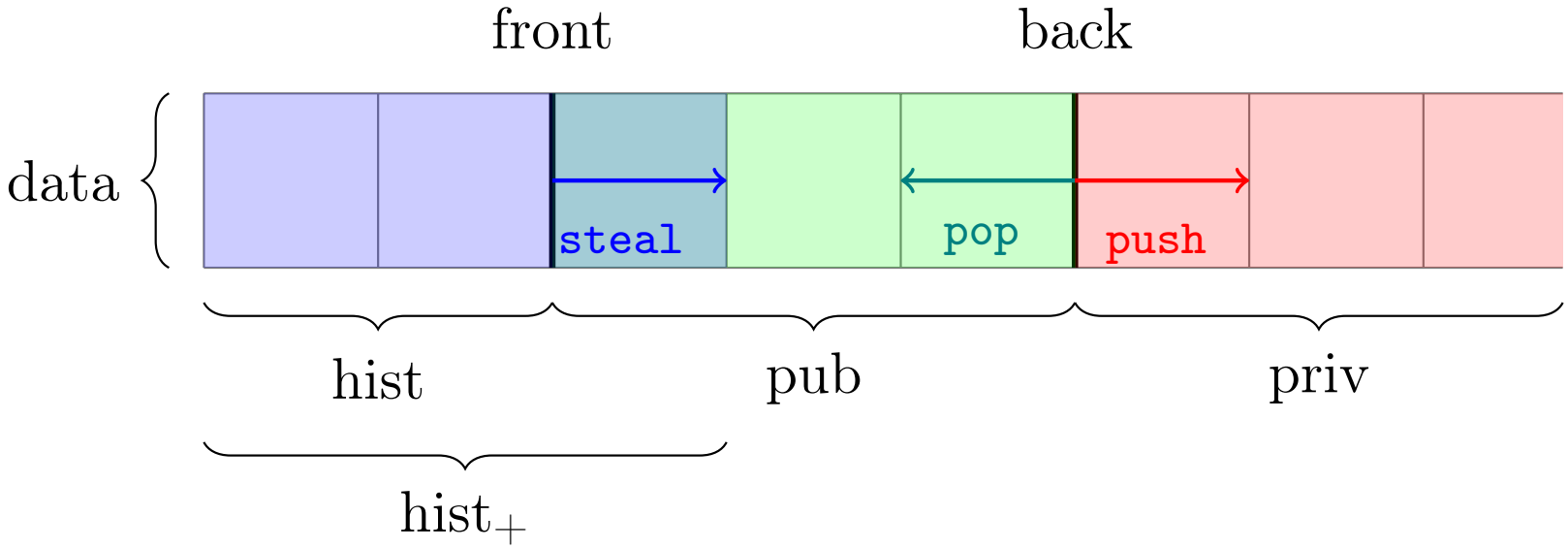
Physical state



data: infinite array storing all values
front: *monotone* index for thieves' end
back: index for owner's end

priv: list of private values (controlled by owner)
pub: list of public values (= model)
hist: *monotone* list of history values
hist$_+$: *monotone* list of extended history values

Physical state



data: infinite array storing all values
front: *monotone* index for thieves' end
back: index for owner's end

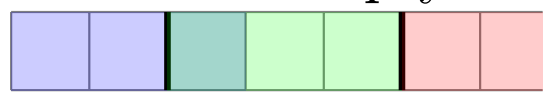
priv: list of private values (controlled by owner)
pub: list of public values (= model)
hist: *monotone* list of history values
hist_+: *monotone* list of extended history values

Logical state

① empty

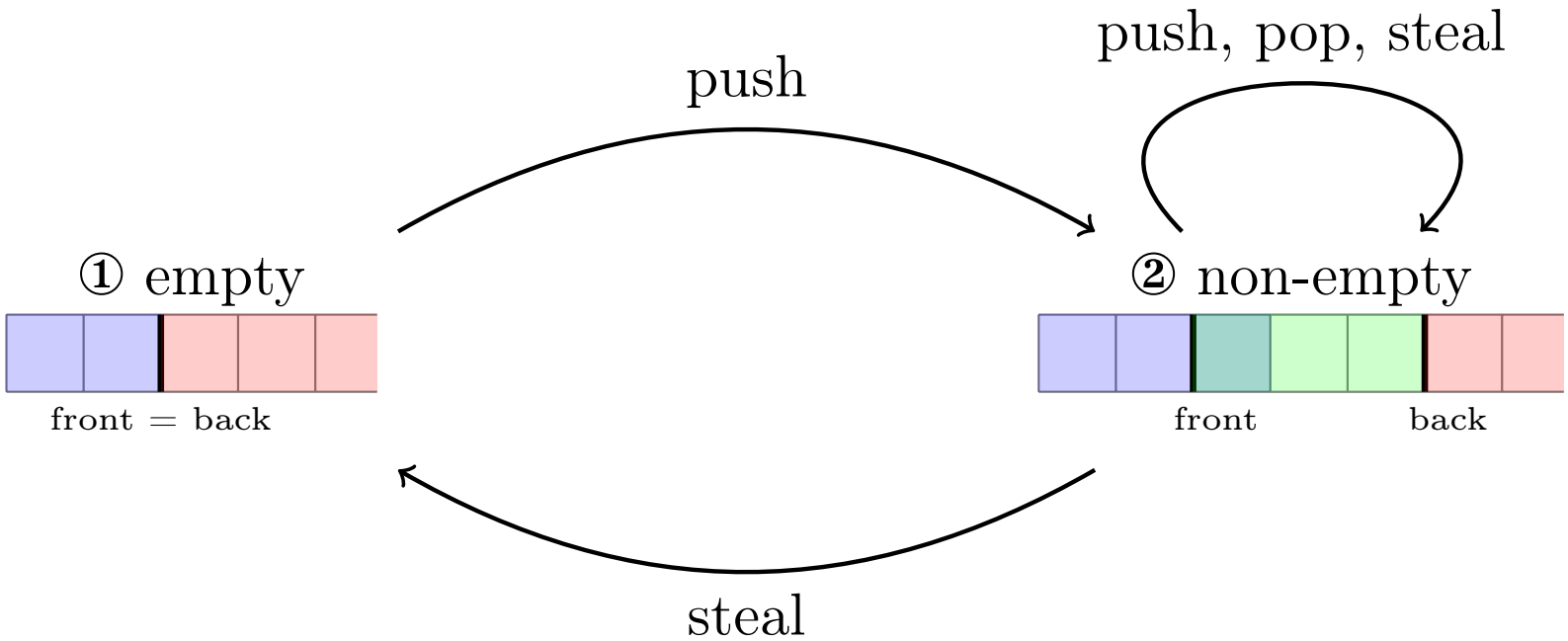
front = back

② non-empty

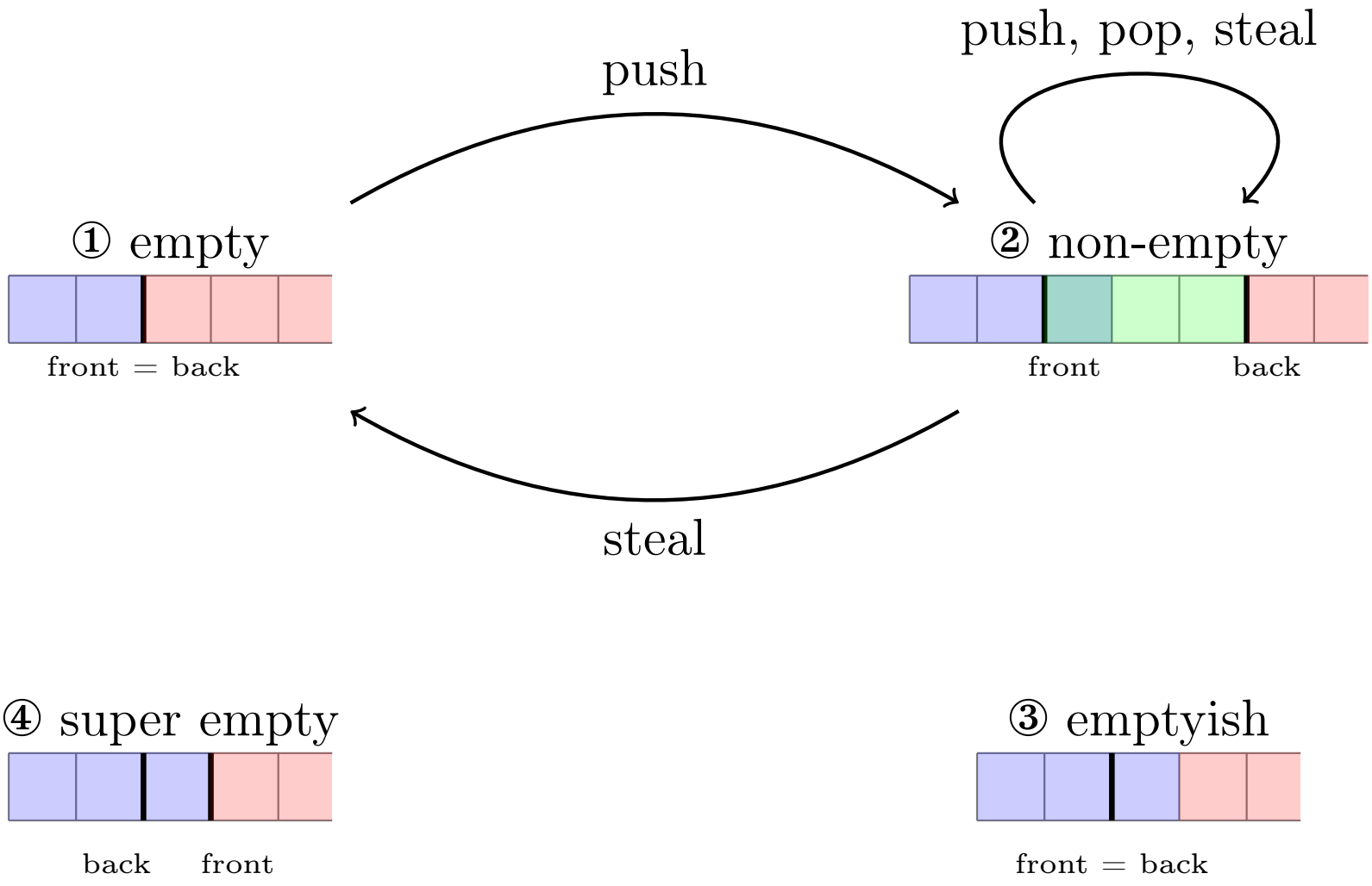


front back

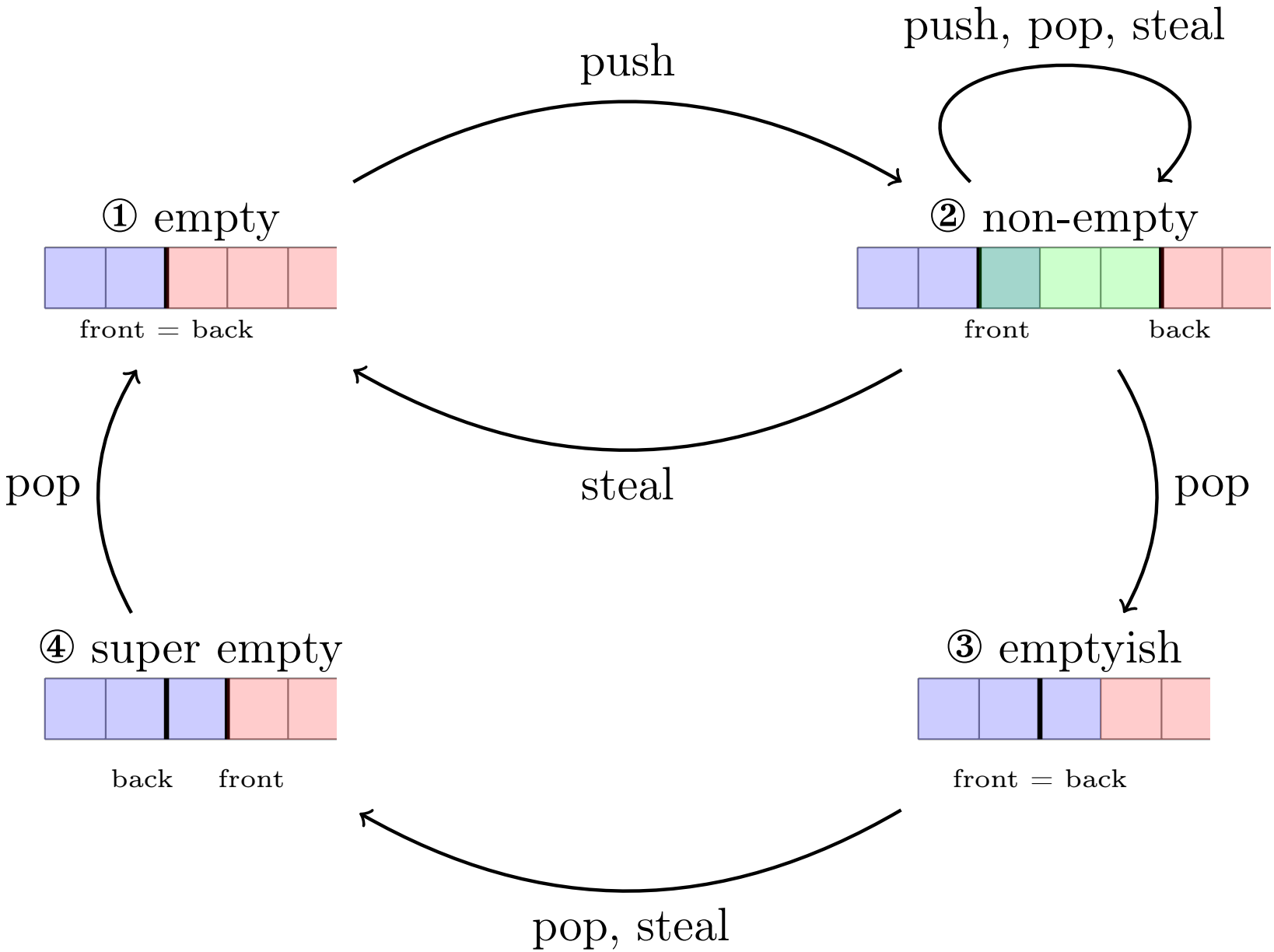
Logical state



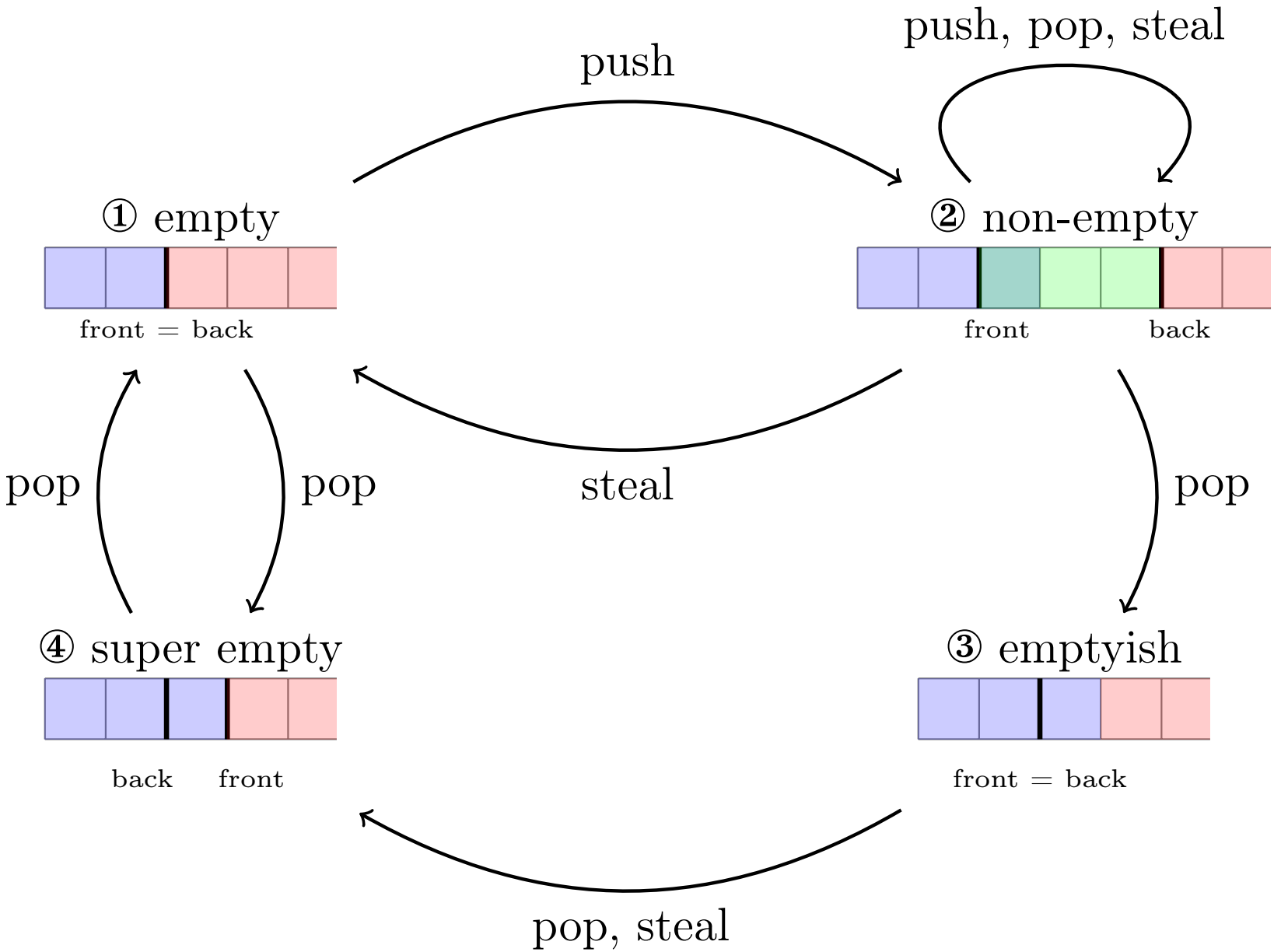
Logical state



Logical state



Logical state



Prophecy variable

$$\{\,\mathrm{True}\,\}\ \texttt{NewProph}\ \{\,\lambda\ p \cdot \exists\, \mathit{prophs} \cdot \mathrm{proph}\ p\ \mathit{prophs}\,\}$$

$$\frac{\begin{array}{c}\mathrm{atomic}\ e \\ \mathrm{proph}\ p\ \mathit{prophs} \\ \mathrm{WP}\ e \left\{ \begin{array}{l} \lambda w \cdot \forall\, \mathit{prophs}' \cdot \\ \mathit{prophs} = (w, v) :: \mathit{prophs}' \mathrel{-\!\!*} \\ \mathrm{proph}\ p\ \mathit{prophs}' \mathrel{-\!\!*} \\ \Phi\ w \end{array} \right\}\end{array}}{\mathrm{WP}\ \texttt{Resolve}\ e\ p\ v\ \{\,\Phi\,\}}$$

Prophecy variable in RDCSS

```
let rdcss rm rn m1 n1 n2 =
  let p = NewProph in
  let descr = ref (rm, m1, n1, n2, p) in
  ...


let complete descr rn =
  let (rm, m1, n1, n2, p) = !descr in
  let id = NewId in
  let m = !rm in
  let n_new = if m = m1 then n2 else n1 in
  Resolve (CmpXchg rn (inr descr) (inl n_new)) p id ;
  ()
```

Prophecy variable with memory

$$\{\,\mathrm{True}\,\}\ \texttt{NewProph}\ \{\,\lambda\ p \cdot \exists\,\gamma, \mathit{prophs} \cdot \mathrm{proph}\ p\ \gamma\ []\ \mathit{prophs}\,\}$$

$$\frac{\begin{array}{c}\mathrm{atomic}\ e \\ \mathrm{proph}\ p\ \gamma\ \mathit{past}\ \mathit{prophs} \\ \mathrm{WP}\ e \left\{ \begin{array}{l} \lambda w \cdot \forall\, \mathit{prophs}' \cdot \\ \mathit{prophs} = (w, v) :: \mathit{prophs}' \mathrel{-\!\!*} \\ \mathrm{proph}\ p\ \gamma\ (\mathit{past} +\!\!+ [(w, v)])\ \mathit{prophs}' \mathrel{-\!\!*} \\ \Phi\ w \end{array} \right\}\end{array}}{\mathrm{WP}\ \texttt{Resolve}\ e\ p\ v\ \{\,\Phi\,\}}$$

Thank you for your attention!

Implementation — `chaselev_make`

```
let chaselev_make _ =
  let t = AllocN 4 () in
  t.front <- 0 ;
  t.back <- 0 ;
  t.data <- inf_array_make () ;
  t.prophecy <- NewProph ;
  t
```

Implementation — `chaselev_push`

```
let chaselev_push t v =
  let back = !t.back in
  inf_array_set !t.data back v ;
  t.back <- back + 1
```

Implementation — chaselev_steal

```
let rec chaselev_steal t =
  let id = NewId in
  let front = !t.front in
  let back = !t.back in
  if front < back then (
    if Snd (
      Resolve (
        CmpXchg t.front front (front + 1)
       ) !t.prophecy (front, id)
    ) then (
      SOME (inf_array_get !t.data front)
    ) else (
      chaselev_steal t
    )
  ) else (
    NONE
  )
```

Implementation — chaselev_pop

```
let chaselev_pop t =
  let id = NewId in
  let back = !t.back - 1 in
  t.back <- back ;
  let front = !t.front in
  if back < front then (
    t.back <- front
  ) else (
    if front < back then (
      SOME (inf_array_get !t.data back)
    ) else (
      if Snd (
        Resolve (
          CmpXchg t.front front (front + 1)
        ) !t.prophecy (front, id)
      ) then (
        t.back <- front + 1 ;
        SOME (inf_array_get !t.data back)
      ) else (
        t.back <- front + 1 ;
        NONE
      )
```

Infinite array

$$\frac{\{\,\mathrm{True}\,\}}{\dfrac{\texttt{inf\_array\_make}\ v}{\{\,\lambda\ arr \cdot \text{inf-array-model}\ arr\ (\lambda\_ \cdot v)\,\}}}$$

$$\frac{\langle\,\forall\, vs \cdot\ \text{inf-array-model}\ arr\ vs * 0 \leqslant i\,\rangle}{\dfrac{\texttt{inf\_array\_get}\ arr\ i}{\langle\,\exists \cdot \lambda\,(vsi) \cdot\ \text{inf-array-model}\ arr\ vs\,\rangle}}$$

$$\frac{\langle\,\forall\, vs \cdot\ \text{inf-array-model}\ arr\ vs * 0 \leqslant i\,\rangle}{\dfrac{\texttt{inf\_array\_set}\ arr\ i\ v}{\langle\,\exists \cdot \lambda\ \_ \cdot\ \text{inf-array-model}\ arr\ vs[i \mapsto v]\,\rangle}}$$

Invariant

$$\text{chaselev-inv } t\ \iota \triangleq$$
$$\exists\, \ell, \gamma, \mathit{data}, p \cdot$$
$$* \left[\begin{array}{l} t = \ell * \text{meta } \ell\ \gamma \\ \ell.\text{data} \mapsto_{\square} \mathit{data} * \ell.\text{prophecy} \mapsto_{\square} p \\ \boxed{\text{chaselev-inv-inner } \ell\ \gamma\ \iota\ \mathit{data}\ p}^{\iota} \end{array}\right.$$

Invariant

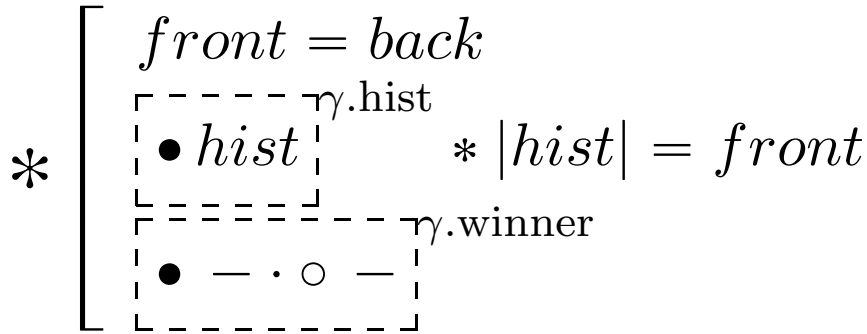
$$
\begin{aligned}
&\text{chaselev-inv-inner } \ell\ \gamma\ \iota\ \mathit{data}\ p \triangleq \\
&\quad \exists\, \mathit{front}, \mathit{back}, \mathit{hist}, \mathit{pub}, \mathit{priv}, \mathit{past}, \mathit{prophs} \cdot \\
&\quad\quad \ast \left[\begin{array}{l}
\ell.\text{front} \mapsto \mathit{front} * \ell.\text{back} \mapsto \mathit{back} \\
\boxed{\bullet\, (\mathit{back}, \mathit{priv})}^{\gamma.\text{ctl}} \\
\boxed{\bullet\, \mathit{front}}^{\gamma.\text{front}} \\
\text{inf-array-model } \mathit{data}\ (\mathit{hist} +\!\!+ \mathit{pub})\ \mathit{priv} \\
\boxed{\bullet\, \mathit{pub}}^{\gamma.\text{pub}} \quad * |\mathit{pub}| = (\mathit{back} - \mathit{front})_+ \\
\text{wise-prophet-model } p\ \gamma.\text{prophet}\ \mathit{past}\ \mathit{prophs} \\
\forall (\mathit{front}', \_) \in \mathit{past} \cdot \mathit{front}' < \mathit{front} \\
\text{chaselev-state } \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs}
\end{array}\right.
\end{aligned}
$$

State

$$\text{chaselev-state}\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs} \triangleq$$
$$\bigvee \left[\begin{array}{l} \text{chaselev-state}_1\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \\ \text{chaselev-state}_2\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs} \\ \text{chaselev-lock}\ \gamma * \bigvee \left[\begin{array}{l} \text{chaselev-state}_{3,1}\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{prophs} \\ \text{chaselev-state}_{3,2}\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \end{array} \right. \end{array} \right.$$

State

$$\text{chaselev-state}_1\ \gamma\ front\ back\ hist \triangleq$$



State

$$\text{chaselev-state}_2\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{pub}\ \mathit{prophs} \triangleq$$

$$
\begin{array}{l}
\mathit{front} < \mathit{back} \\
\boxed{\bullet\,(\mathit{hist} +\!\!+ [\mathit{pub}[0]])}^{\gamma.\text{hist}} \quad * \ |\mathit{hist}| = \mathit{front} \\
\\
\textbf{match}\ \text{filter}\ (\lambda(\mathit{front}', \_) \cdot \mathit{front}' = \mathit{front})\ \mathit{prophs}\ \textbf{with} \\
* \ \mid [] \Rightarrow \boxed{\bullet\ -\cdot\circ\ -}^{\gamma.\text{winner}} \\
\mid (\_, \mathit{id}) :: \_ \Rightarrow \\
\quad \bigvee \left[\begin{array}{l} \boxed{\bullet\ -\cdot\circ\ -}^{\gamma.\text{winner}} \\ \text{identifier}\ \mathit{id} * \exists\, \Phi \cdot \boxed{\bullet\,(\mathit{front}, \Phi)}^{\gamma.\text{winner}} \quad * \text{chaselev-au}\ \gamma\ \iota\ \Phi \end{array} \right.
\end{array}
$$

State

$$\text{chaselev-state}_{3,1}\ \gamma\ front\ back\ hist\ prophs \triangleq$$

$$
*\left[\begin{array}{l}
front = back \\
\boxed{\bullet\, hist}^{\gamma.\text{hist}} \quad * \left|hist\right| = front + 1 \\
\\
\quad \textbf{match}\ \text{filter}\ (\lambda(front', \_) \cdot front' = front)\ prophs\ \textbf{with} \\
\quad \mid [] \Rightarrow \boxed{\circ\,(front, -)}^{\gamma.\text{winner}} \\
\quad \mid \_ \Rightarrow \exists\, \Phi \cdot \boxed{\bullet\,(front, \Phi)}^{\gamma.\text{winner}} \quad * \Phi(\texttt{SOME}\ hist[front])
\end{array}\right.
$$

State

$$\text{chaselev-state}_{3,2}\ \gamma\ front\ back\ hist \triangleq$$