Verification of Chase-Lev work-stealing deque

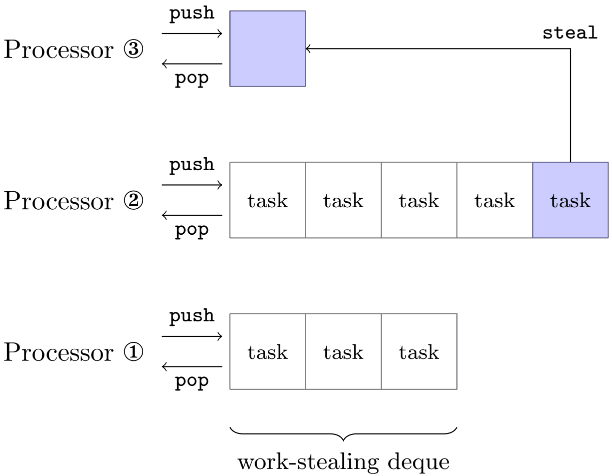
Clément Allain
François Pottier

May 14, 2023

Context: scheduler for task-based parallelism

- Cilk (C, C++)
- Threading Building Blocks (C++)
- Taskflow (C++)
- Tokio (Rust)
- Goroutines (Go)
- Domainslib (OCaml 5)

Work-stealing



Chase-Lev work-stealing deque

1. *The Implementation of the Cilk-5 Multithreaded Language.* Frigo, Leiserson & Randall (1998).
   - lock
2. *Thread Scheduling for Multiprogrammed Multiprocessors.* Arora, Blumofe & Plaxton (1998).
   - non-blocking
   - one fixed size array, potential overflow
3. *A dynamic-sized nonblocking work stealing deque.* Hendler, Lev, Moir, & Shavit (2004).
   - non-blocking
   - list of small arrays, no overflow
4. *Dynamic circular work-stealing deque.* Chase & Lev (2005).
   - lockfree
   - 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
- Nontrivial use of prophecy variables.

The rest of this talk

- Specification using logically atomic triples.
- Rough idea of how the data structure works.
- Why we need prophecy variables.

Specification

Physical state

Logical state

Prophecy variables

Specification — `chaselev_make`

$$\frac{\{\,\mathrm{True}\,\}}{\dfrac{\texttt{chaselev\_make ()}}{\left\{\,\lambda\, t \cdot\; \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-model}\ t\ []\ *\ \text{chaselev-owner}\ t\,\right\}}}$$

Specification — `chaselev_make`

$$\frac{\{\,\text{True}\,\}}{\dfrac{\texttt{chaselev\_make ()}}{\left\{\lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-model}\ t\ []\ \ast\ \text{chaselev-owner}\ t\right\}}}$$

t is an instance of Chase-Lev deque.
Enforces a protocol (using an Iris invariant).

Specification — `chaselev_make`

$$\dfrac{\{\,\text{True}\,\}}{\dfrac{\texttt{chaselev\_make ()}}{\left\{\lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-model}\ t\ []\ *\ \text{chaselev-owner}\ t\right\}}}$$

Asserts the list of values that t logically contains.

Specification — `chaselev_make`

$$\frac{\{\,\mathrm{True}\,\}}{\dfrac{\texttt{chaselev\_make ()}}{\left\{\,\lambda\, t \cdot\ \text{chaselev-inv}\ t\ \iota\ *\ \text{chaselev-model}\ t\ []\ *\ \underbrace{\text{chaselev-owner}\ t}_{\text{Gives the owner exclusive access to his end of } t.}\,\right\}}}$$

Specification — `chaselev_push`

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

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

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

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

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

Specification — `chaselev_push`

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

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

P : private precondition

Q : private postcondition

P_{lin} : public precondition

Q_{lin} : public postcondition

Specification — `chaselev_push`

For a concurrent data structure:

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

Specification — `chaselev_push`

$$
\begin{array}{c}
\left\{ \text{chaselev-inv } t\ \iota \ * \ \text{chaselev-owner } t \right\} \\
\hline
\left\langle \forall\, vs \cdot \ \text{chaselev-model } t\ vs \right\rangle \\
\hline
\texttt{chaselev\_push}\ t\ v,\ \uparrow \iota \\
\hline
\left\langle \exists \cdot \ \text{chaselev-model } t\ (vs \mathbin{+\!\!+} [v]) \right\rangle \\
\hline
\left\{ \lambda\, \texttt{()} \cdot \ \text{chaselev-owner } t \right\}
\end{array}
$$

t is an instance of Chase-Lev deque.

Specification — `chaselev_push`

$$
\begin{array}{c}
\left\{ \text{chaselev-inv}\ t\ \iota \ * \ \text{chaselev-owner}\ t \right\} \\
\hline
\left\langle \forall\, vs \cdot \ \text{chaselev-model}\ t\ vs \right\rangle \\
\hline
\texttt{chaselev\_push}\ t\ v,\ \uparrow \iota \\
\hline
\left\langle \exists \cdot \ \text{chaselev-model}\ t\ (vs +\!\!+ [v]) \right\rangle \\
\hline
\left\{ \lambda\, \texttt{()} \cdot \ \text{chaselev-owner}\ t \right\}
\end{array}
$$

This operation is reserved to the owner of t.

Specification — `chaselev_push`

$$
\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ \ast\ \text{chaselev-owner}\ t\ \right\} \\
\hline
\left\langle\ \forall\, vs \cdot\ \text{chaselev-model}\ t\ vs\ \right\rangle \\
\hline
\texttt{chaselev\_push}\ t\ v,\ \uparrow\iota \\
\hline
\left\langle\ \exists \cdot\ \text{chaselev-model}\ t\ (vs +\!\!+ [v])\ \right\rangle \\
\hline
\left\{\ \lambda\,\texttt{()} \cdot\ \text{chaselev-owner}\ t\ \right\}
\end{array}
$$

v is atomically pushed at the owner's end of t.

Specification — `chaselev_pop`

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

Specification — `chaselev_pop`

$$\begin{array}{c}
\left\{ \text{chaselev-inv}\ t\ \iota \ast \text{chaselev-owner}\ t \right\} \\ \hline
\left\langle \forall vs \cdot \text{chaselev-model}\ t\ vs \right\rangle \\ \hline
\texttt{chaselev\_pop}\ t,\ \uparrow \iota \\ \hline
\left\langle \exists o \cdot \bigvee \begin{bmatrix} vs = [] \ast o = \texttt{NONE} \ast \text{chaselev-model}\ t\ [] \\ \exists v, vs' \cdot vs = vs' +\!\!+ [v] \ast o = \texttt{SOME}\ v \ast \text{chaselev-model}\ t\ vs' \end{bmatrix} \right\rangle \\ \hline
\left\{ \lambda o \cdot \text{chaselev-owner}\ t \right\}
\end{array}$$

t is an instance of Chase-Lev deque.

Specification — `chaselev_pop`

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

This operation is reserved to the owner of t.

Specification — `chaselev_pop`

$$\begin{array}{c}
\left\{ \text{chaselev-inv } t\ \iota \ * \ \text{chaselev-owner } t \right\} \\ \hline
\left\langle \forall vs \cdot \ \text{chaselev-model } t\ vs \right\rangle \\ \hline
\texttt{chaselev\_pop } t,\ \uparrow \iota \\ \hline
\left\langle \exists o \cdot \ \bigvee \begin{bmatrix} vs = [] * o = \texttt{NONE} * \text{chaselev-model } t\ [] \\ \exists v, vs' \cdot vs = vs' \mathbin{+\!\!+} [v] * o = \texttt{SOME } v * \text{chaselev-model } t\ vs' \end{bmatrix} \right\rangle \\ \hline
\left\{ \lambda o \cdot \ \text{chaselev-owner } t \right\}
\end{array}$$

Either 1) t is seen empty
or 2) some value v is atomically popped at the owner's end of t.

Specification — `chaselev_steal`

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

Specification — `chaselev_steal`

$$\begin{array}{c}
\left\{\ \text{chaselev-inv}\ t\ \iota\ \right\} \\ \hline
\left\langle \forall\, vs \cdot\ \ \text{chaselev-model}\ t\ vs \right\rangle \\ \hline
\texttt{chaselev\_steal}\ t,\ \uparrow \iota \\ \hline
\left\langle \exists\, o \cdot\ \ \bigvee \left[\begin{array}{l} vs = [] * o = \texttt{NONE} * \text{chaselev-model}\ t\ [] \\ \exists\, v, vs' \cdot vs = v :: vs' * o = \texttt{SOME}\ v * \text{chaselev-model}\ t\ vs' \end{array} \right] \right\rangle \\ \hline
\{\, \lambda\, o \cdot\ \text{True}\, \}
\end{array}$$

t is an instance of Chase-Lev deque.

Specification — `chaselev_steal`

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

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

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

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

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

Either 1) t is seen empty
or 2) some value v is atomically popped at the thieves' end of t.

Specification

Physical state

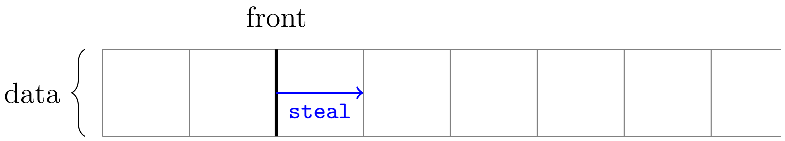
Logical state

Prophecy variables

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

ChaselevFrontLbGet

$$\frac{\boxed{\bullet\ front}^{\gamma.\mathrm{front}}}{\boxed{\circ\ front}^{\gamma.\mathrm{front}}}$$

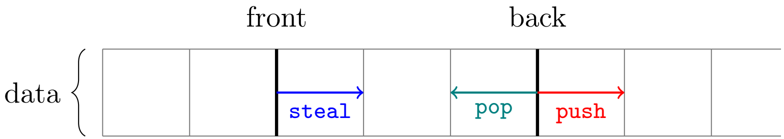
ChaselevFrontValid

$$\frac{\boxed{\bullet\ front_1}^{\gamma.\mathrm{front}} \qquad \boxed{\circ\ front_2}^{\gamma.\mathrm{front}}}{front_2 \leqslant front_1}$$

ChaselevFrontUpdate

$$\frac{front \leqslant front' \qquad \boxed{\bullet\ front}^{\gamma.\mathrm{front}}}{\boxed{\bullet\ front'}^{\gamma.\mathrm{front}}}$$

Physical state

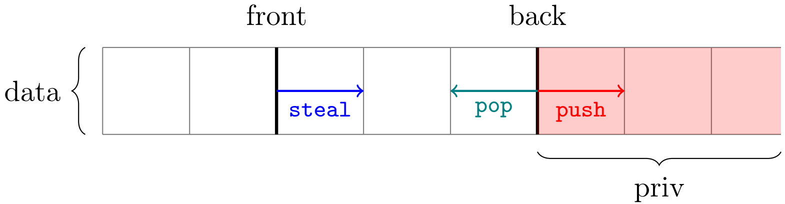


data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

Physical state



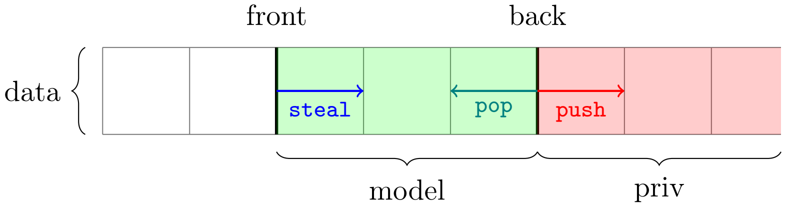
data: infinite array storing all values

front: *monotone* index for thieves' end

back: index for owner's end

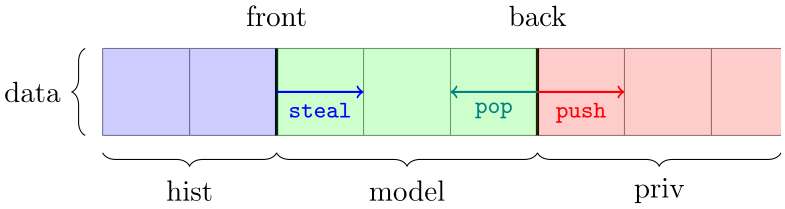
priv: list of private values (controlled by owner)

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)
model:	list of contained 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)

model: list of contained values

hist: *monotone* list of history values

Physical state

data

data: in

front: m

back: in

priv: lis

model: lis

hist: m

ChaselevHistLbGet

$$\frac{\boxed{\bullet\, \mathit{hist}}^{\gamma.\text{hist}}}{\boxed{\circ\, \mathit{hist}}^{\gamma.\text{hist}}}$$

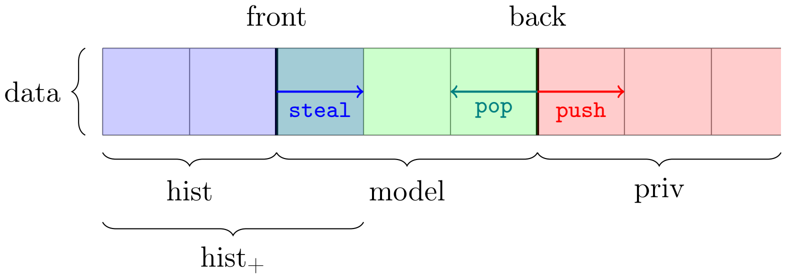
ChaselevHistValid

$$\frac{\boxed{\bullet\, \mathit{hist}_1}^{\gamma.\text{hist}} \qquad \boxed{\circ\, \mathit{hist}_2}^{\gamma.\text{hist}}}{\mathit{hist}_2 \sqsubseteq_{\text{prefix}} \mathit{hist}_1}$$

ChaselevHistUpdate

$$\frac{\boxed{\bullet\, \mathit{hist}}^{\gamma.\text{hist}}}{\boxed{\bullet\, (\mathit{hist} + [v])}^{\gamma.\text{hist}}}$$

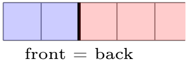
Physical state



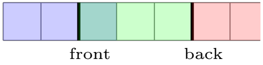
data:	infinite array storing all values
front:	*monotone* index for thieves' end
back:	index for owner's end
priv:	list of private values (controlled by owner)
model:	list of contained values
hist:	*monotone* list of history values
hist_+:	*monotone* list of extended history values

Logical state

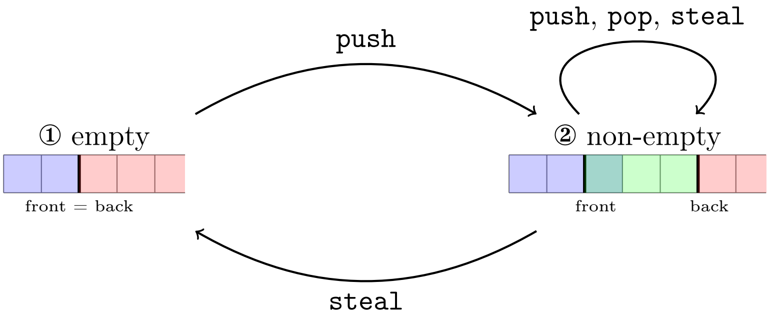
① empty



② non-empty

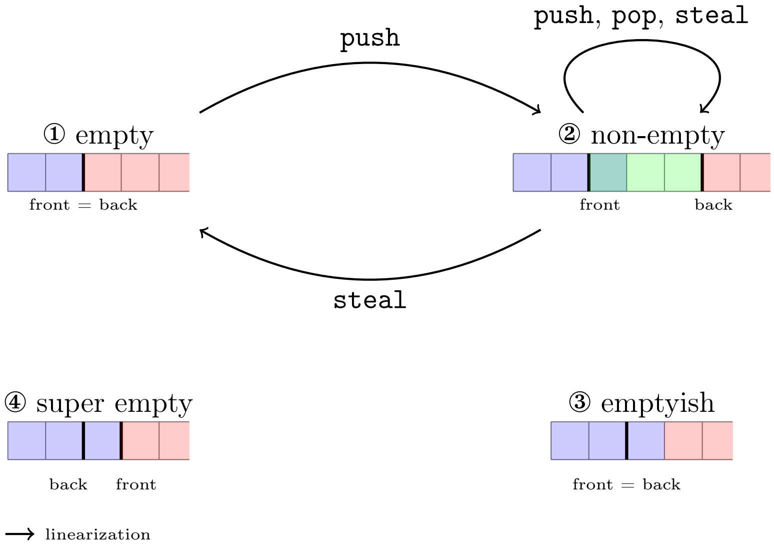


Logical state

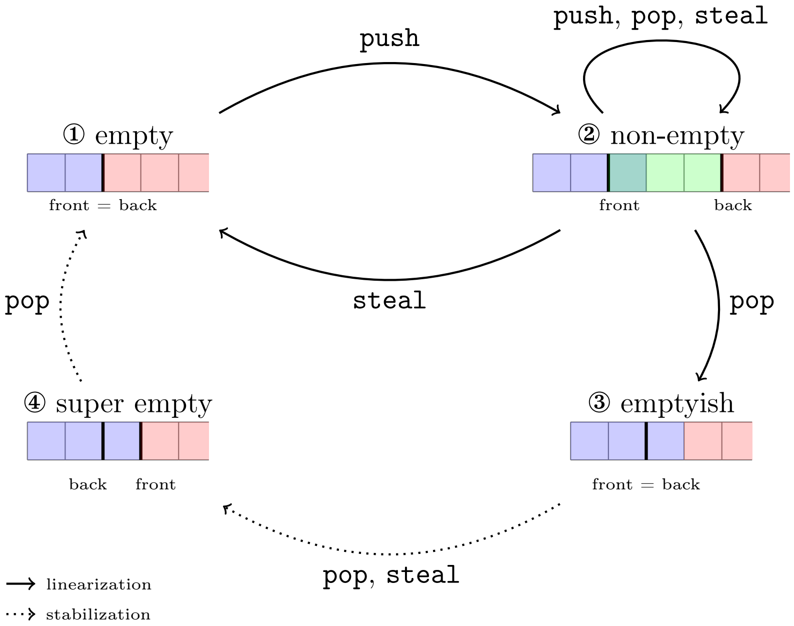


⟶ linearization

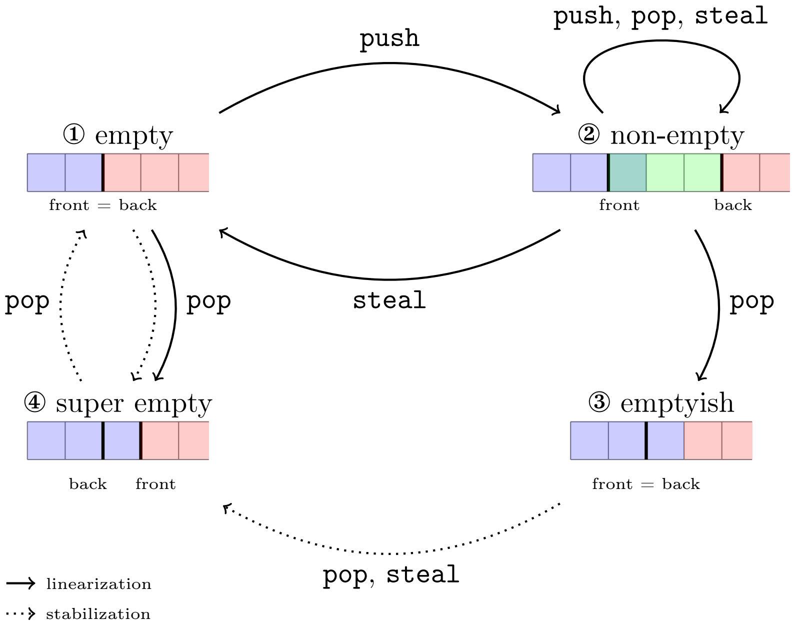
Logical state



Logical state



Logical state



Specification

Physical state

Logical state

Prophecy variables

Prophecy variables

The future is ours: prophecy variables in separation logic.
Jung, Lepigre, Parthasarathy, Rapoport, Timany, Dreyer & Jacobs (2020).

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

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

Back to *The future is ours* (Jung *et al.*)

```
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 variables with memory

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

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

Prophecy variables with memory

$$\frac{\text{proph } p\ \gamma\ \mathit{past}\ \mathit{prophs}}{\text{proph-lb } \gamma\ \mathit{prophs}}\ \textsc{ProphecyLbGet}$$

$$\frac{\text{proph } p\ \gamma\ \mathit{past}\ \mathit{prophs}_1 \qquad \text{proph-lb } \gamma\ \mathit{prophs}_2}{\exists\, \mathit{past}_1, \mathit{past}_2 \cdot \bigwedge \left[\begin{array}{l} \mathit{past} = \mathit{past}_1 +\!\!+\ \mathit{past}_2 \\ \mathit{past}_2\ +\!\!+\ \mathit{prophs}_1 = \mathit{prophs}_2 \end{array} \right.}\ \textsc{ProphecyValid}$$

Conclusion

- Coq mechanization is available on `github` :
  `https://github.com/clef-men/caml5_alienation`

- Simplified Chase-Lev deque (one infinite array) ✓
  Real-life Chase-Lev deque (multiple circular arrays) ✗

- Proof looks more complex than the sketch. In particular, transitions between logical states are not really formalized.

- We plan to verify more primitives (Domainslib, Taskflow) based on Chase-Lev deque. This is thanks to modularity of IRIS specifications.

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

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

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

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

Invariant

$$\text{chaselev-inv } t \; \iota \triangleq$$

$$\exists\, \ell, \gamma, \mathit{data}, p \cdot$$

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

Invariant

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

State

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

$$\bigvee \left[\begin{array}{l} \text{chaselev-state}_1\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \\ \text{chaselev-state}_2\ \gamma\ \iota\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{model}\ \mathit{prophs} \\ \text{chaselev-lock}\ \gamma * \bigvee \left[\begin{array}{l} \text{chaselev-state}_3\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{prophs} \\ \text{chaselev-state}_4\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \end{array} \right. \end{array} \right.$$

State 1 (empty)

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

$$\mathop{\ast}\left[\begin{array}{l} \mathit{front} = \mathit{back} \\ \boxed{\bullet\, \mathit{hist}}^{\gamma.\text{hist}} \ast |\mathit{hist}| = \mathit{front} \\ \boxed{\bullet\ - \cdot \circ\ -}^{\gamma.\text{winner}} \end{array}\right.$$

State 2 (non-empty)

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

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

State 3 (emptyish)

$$\text{chaselev-state}_3\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist}\ \mathit{prophs} \triangleq$$

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

State 4 (super empty)

$$\text{chaselev-state}_4\ \gamma\ \mathit{front}\ \mathit{back}\ \mathit{hist} \triangleq$$
$$\ast \left[\begin{array}{l} \mathit{front} = \mathit{back} + 1 \\ \boxed{\bullet\, \mathit{hist}}^{\gamma.\text{hist}} \quad \ast\ |\mathit{hist}| = \mathit{front} \\ \boxed{\bullet\ - \cdot \circ\ -}^{\gamma.\text{winner}} \end{array}\right.$$