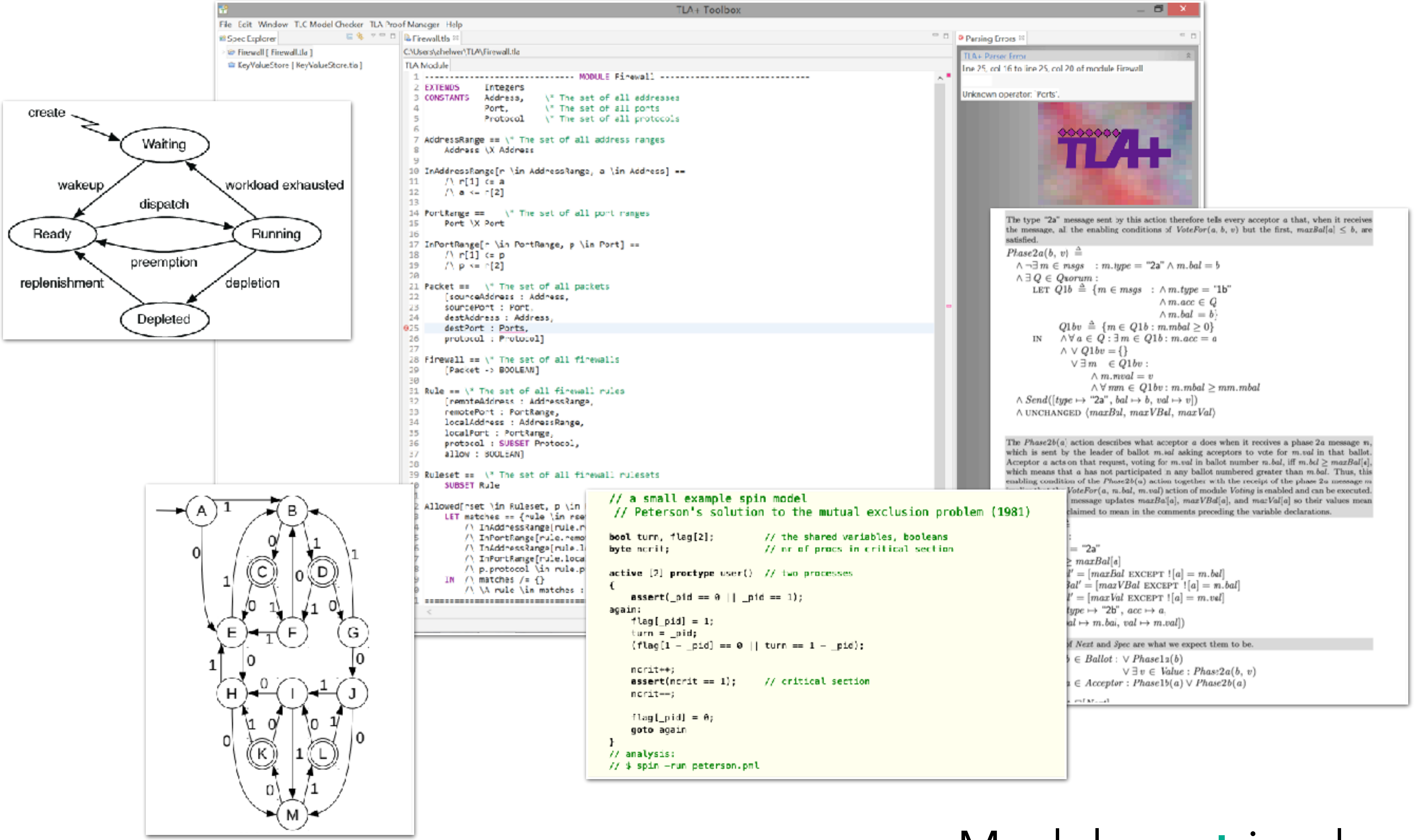
AARHUS UNIVERSITY
CENTER FOR BASIC RESEARCH IN PROGRAM VERIFICATION

Trillium

History-Sensitive Refinement in Separation Logic

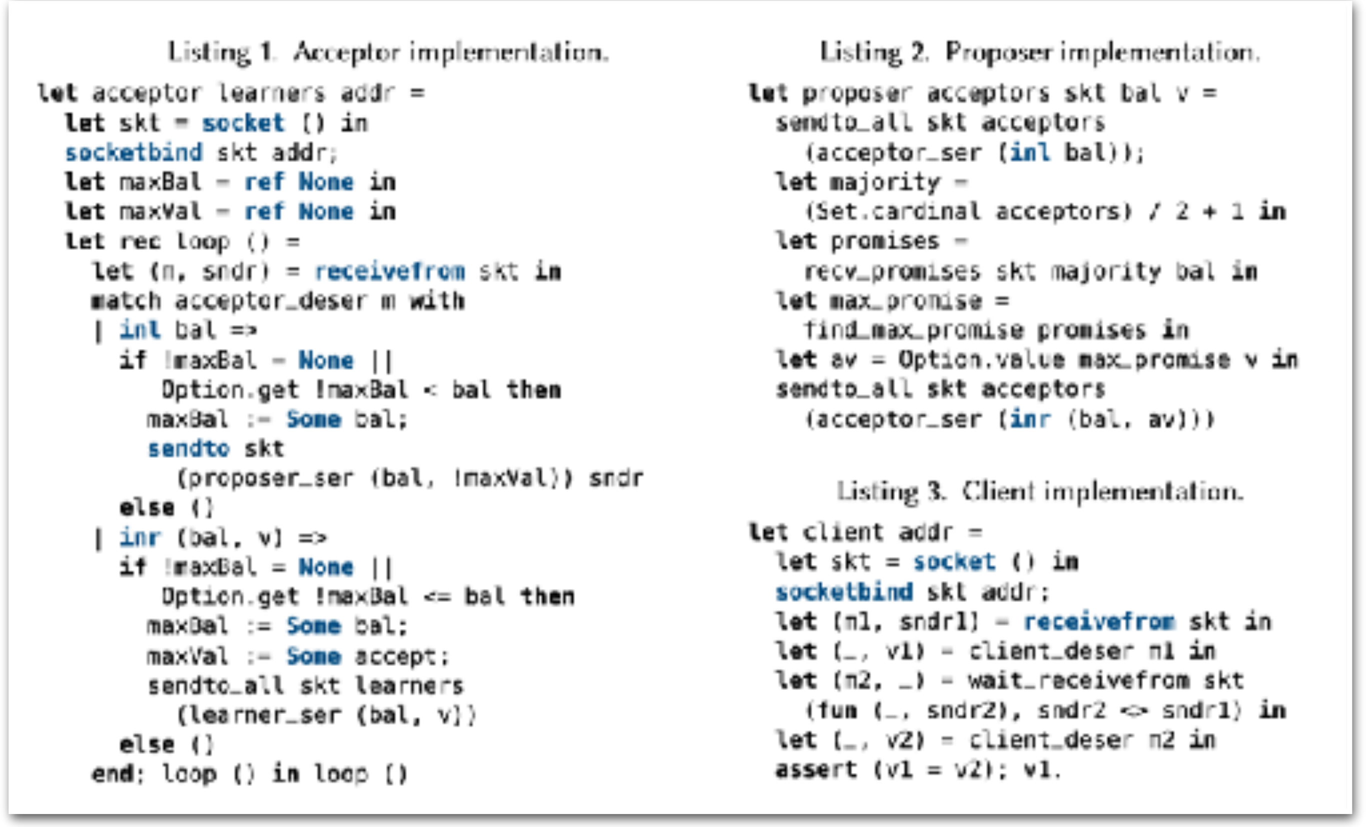
3 May, 2022

Simon Oddershede Gregersen
joint work with Amin Timany, Léo Stefanesco, Léon Gondelman, Abel Nieto, and Lars Birkedal



Models, **not** implementations!

Transport properties?



Listing 1. Acceptor implementation.

```
let acceptor learners addr =
  let skt = socket () in
  socketbind skt addr;
  let maxBal = ref None in
  let maxVal = ref None in
  let rec loop () =
    let (m, sndr) = receivefrom skt in
    match acceptor_deser m with
    | inl bal =>
      if !maxBal = None ||
         Option.get !maxBal < bal then
        maxBal := Some bal;
        sendto skt
          (proposer_ser (bal, !maxVal)) sndr
      else ()
    | inr (bal, v) =>
      if !maxBal = None ||
         Option.get !maxBal <= bal then
        maxBal := Some bal;
        maxVal := Some accept;
        sendto_all skt learners
          (learner_ser (bal, v))
      else ()
    end; loop () in loop ()
```

Listing 2. Proposer implementation.

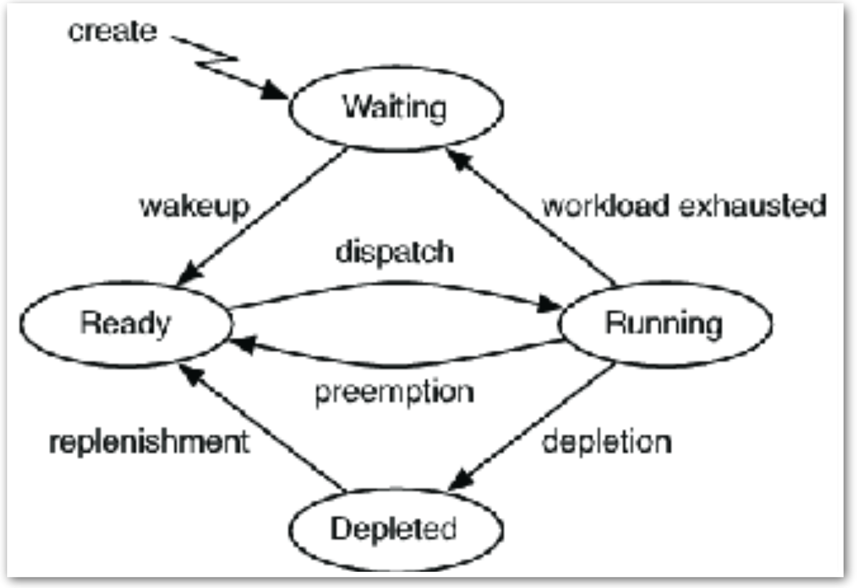
```
let proposer acceptors skt bal v =
  sendto_all skt acceptors
    (acceptor_ser (inl bal));
  let majority =
    (Set.cardinal acceptors) / 2 + 1 in
  let promises =
    recv_promises skt majority bal in
  let max_promise =
    find_max_promise promises in
  let av = Option.value max_promise v in
  sendto_all skt acceptors
    (acceptor_ser (inr (bal, av)))
```

Listing 3. Client implementation.

```
let client addr =
  let skt = socket () in
  socketbind skt addr;
  let (m1, sndr1) = receivefrom skt in
  let (_, v1) = client_deser m1 in
  let (m2, _) = wait_receivefrom skt
    (fun (_, sndr2), sndr2 <> sndr1) in
  let (_, v2) = client_deser m2 in
  assert (v1 = v2); v1.
```

?

$\preceq$



How do we connect **implementations** to more abstract **models**?

… using Iris, obviously

Outline

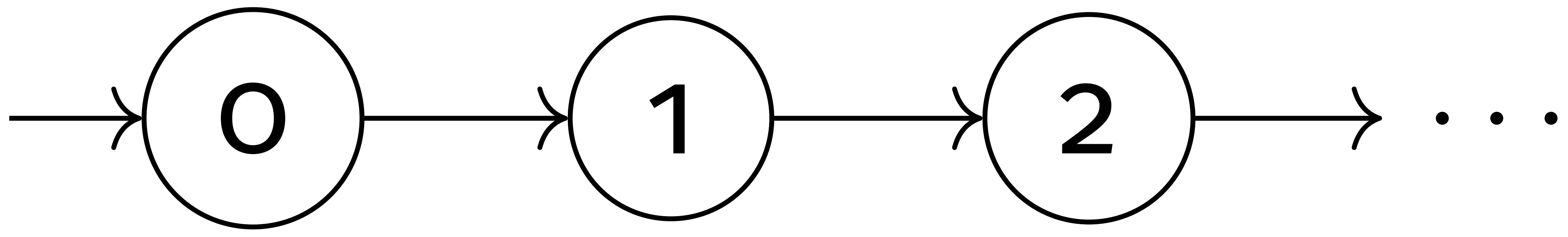
- The Trillium methodology
- **Case study**: Single-decree Paxos using a TLA+ model
- **Case study**: Fair termination of a concurrent program

We also show **eventual consistency** of a CRDT; see the paper for more details

Running Example

```
let rec inc_loop () =
  let n = !ℓ in
  cas(ℓ, n, n + 1);
  inc_loop ()
in
  inc_loop () || inc_loop ()
```

inc



$\mathcal{M}_{\text{inc}}$

Definition

Given **relation btw. traces** ξ

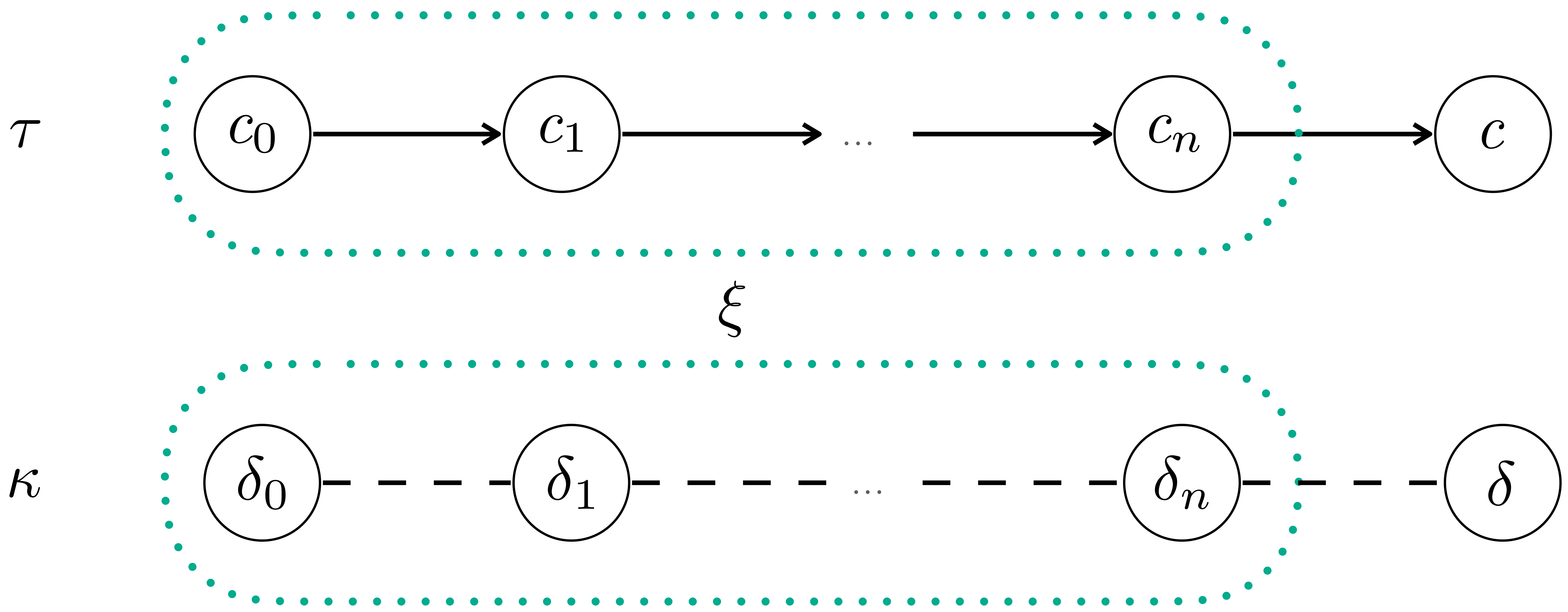
execution trace τ (non-empty sequence of configurations)

model trace κ (non-empty sequence of model states)

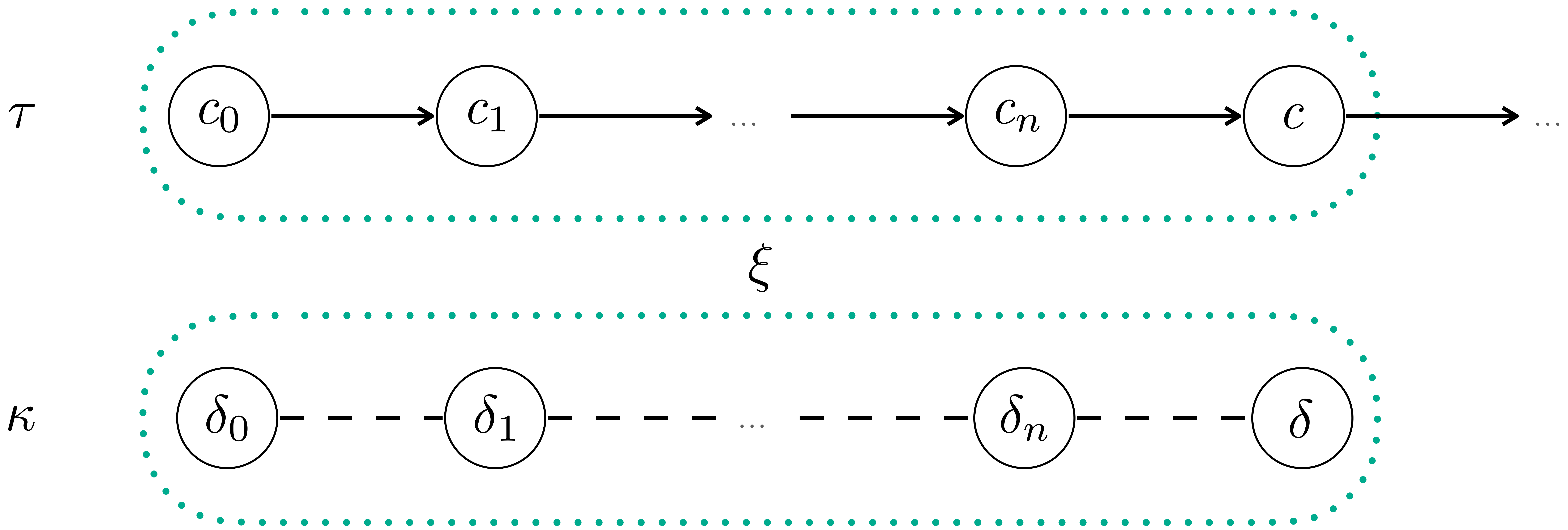
τ is a history-sensitive refinement of κ under ξ whenever

$$\tau \precsim_{\xi} \kappa \triangleq \xi(\tau, \kappa) \wedge \forall c.\, \mathit{last}(\tau) \to c \Rightarrow \exists \delta.\, \tau \cdot c \precsim_{\xi} \kappa \cdot \delta$$

holds coinductively.



$$\tau \precsim_\xi \kappa \triangleq \xi(\tau, \kappa) \wedge \forall c.\, last(\tau) \to c \Rightarrow \exists \delta.\, \tau \cdot c \precsim_\xi \kappa \cdot \delta$$



$$\tau \precsim_{\xi} \kappa \triangleq \xi(\tau, \kappa) \wedge \forall c.\, last(\tau) \to c \Rightarrow \exists \delta.\, \tau \cdot c \precsim_{\xi} \kappa \cdot \delta$$

Running Example

For our running example, we pick

$$\xi_{inc}(\tau, \kappa) \quad \triangleq \quad heap(last(\tau))(\ell) = last(\kappa) \wedge stuttering(\kappa)$$

where

stepping relation of the STS

$$stuttering(\kappa) = \begin{cases} last(\kappa') = \delta \vee last(\kappa') \rightharpoonup_{\mathcal{M}_{\mathsf{inc}}} \delta & \text{if } \kappa = \kappa' \cdot \delta \\ \text{True} & \text{otherwise} \end{cases}$$

which reduces refinement to a notion of simulation.

Trillium

On top of the standard Iris base logic, we introduce two new connectives

$$\mathsf{wp}^{\mathcal{M}}\, e\, \{Q\} \qquad\qquad \mathsf{Model}(\delta : \mathcal{M})$$

where $\mathcal{M} = (A_{\mathcal{M}}, \rightharpoonup_{\mathcal{M}})$ is some STS.

Trillium

The weakest precondition theory satisfies all the usual rules **and**

$$\frac{\{P\}\, e\, \{Q\}^{\mathcal{M}} \qquad \delta \rightharpoonup_{\mathcal{M}} \delta' \qquad \mathsf{Atomic}(e) \qquad e \notin \mathit{Val}}{\{P * \mathsf{Model}(\delta)\}\, e\, \{Q * \mathsf{Model}(\delta')\}^{\mathcal{M}}}$$

ensures that we relate a program step with a single model step

using the usual encoding of Hoare triples.

Running Example

We show

$$\left\{\boxed{\exists n.\, \ell \mapsto n * \mathsf{Model}(n)}\right\} \; \texttt{inc} \; \{\mathsf{False}\}^{\mathcal{M}_{\mathsf{inc}}}$$

which implies the refinement relation.

Theorem (Adequacy)

The set $\{\delta \mid \xi(\tau \cdot c, \kappa \cdot \delta)\}$ is finite

Let e be a **program**, σ a **state,** δ a **model state** and ξ a **finitary** trace relation. Suppose

$$\Rrightarrow_{\top} S((e,\sigma),\delta) * \mathsf{wp}_{\top}^{\mathcal{M}}\; e\;\{\Phi\} * \mathit{AlwaysHolds}(\xi)$$

then e is safe and $(e,\sigma) \precsim_{\xi} \delta$ holds in the metalogic, where

$$\mathit{AlwaysHolds}(\xi) \triangleq \forall \tau, \kappa.\ (\ldots) \twoheadrightarrow {}^{\top}\Rrightarrow^{\emptyset}\ \xi(\tau,\kappa)$$

Paxos by Refinement

1. **Instantiate Trillium** with **AnerisLang**, recovering the **Aneris** logic.
2. **Find a suitable model**: we pick Lamport's TLA+ specification, manually translate it into Coq, and prove it correct.
3. **Show node-local specs** for each 'role' (proposer, acceptor, learner) under a suitable invariant; compose spec for a distributed system
4. **Prove consensus** for the implementation by combining the refinement with the model correctness theorem

Paxos TLA+ Model

- States $(\mathcal{S}, \mathcal{B}, \mathcal{V})$ where $\mathcal{S} \in \mathcal{P}(\mathsf{PaxosMessage})$ is the set of sent messages
- Transitions, e.g.,

$$\frac{\mathsf{msg1a}(b) \in \mathcal{S} \qquad b > \mathcal{B}(a) \qquad \mathcal{V}(a) = o}{\mathcal{S}, \mathcal{B}, \mathcal{V} \rightharpoonup_{\mathsf{SDP}} \mathcal{S} \cup \{\mathsf{msg1b}(a, b, o)\}, \mathcal{B}[a \mapsto \mathsf{Some}(b)], \mathcal{V}}$$

Theorem 3.1 (Consistency, SDP model). *Let* $\iota_{\mathsf{SDP}} = (\emptyset, \lambda\_.\, None, \lambda\_.\, None)$. *If* $\iota_{\mathsf{SDP}} \rightharpoonup^*_{\mathsf{SDP}} (\mathcal{S}, \mathcal{B}, \mathcal{V})$ *and both* $Chosen(\mathcal{S}, v_1)$ *and* $Chosen(\mathcal{S}, v_2)$ *hold then* $v_1 = v_2$.

Paxos Specs

$$\{I_{\mathsf{SDP}} * \mathsf{MaxBal}_{\circ}(a, \mathsf{None}) * \mathsf{MaxBal}_{\circ}(a, \mathsf{None}) * \ldots\} \langle ip; \texttt{acceptor}\ L\ a \rangle \{\mathsf{False}\}$$

$$\{I_{\mathsf{SDP}} * \textit{pending}(b) * \ldots\} \langle ip; \texttt{proposer}\ A\ skt\ b\ v \rangle \{\mathsf{True}\}$$

where

$$I_{\mathsf{SDP}} \triangleq \boxed{\exists \mathcal{S}, \mathcal{B}, \mathcal{V}.\, \mathsf{Model}(\mathcal{S}, \mathcal{B}, \mathcal{V}) * \mathsf{Msgs}_{\bullet}(\mathcal{S}) * \mathsf{MaxBal}_{\bullet}(\mathcal{B}) * \mathsf{MaxVal}_{\bullet}(\mathcal{V}) * \mathit{BalCoh}(\mathcal{S}) * \mathit{MsgCoh}(\mathcal{S})}$$

resolves underspecified aspect of the model

maps model messages to sent messages in the implementation

$$\frac{\nexists v'.\, \mathsf{msg2a}(b, v') \in \mathcal{S} \qquad \mathit{Quorum}(Q) \qquad \mathit{ShowsSafeAt}(\mathcal{S}, Q, b, v)}{\mathcal{S}, \mathcal{B}, \mathcal{V} \rightharpoonup_{\mathsf{SDP}} \mathcal{S} \cup \{\mathsf{msg2a}(b, v)\}, \mathcal{B}, \mathcal{V}}$$

Paxos Refinement

Pick

$$\xi_{\mathsf{SDP}}(\tau, \kappa) \triangleq \exists \mathcal{S}.\, last(\kappa) = (\mathcal{S}, \_, \_) \wedge messages(last(\tau)) \sim \mathcal{S} \wedge stuttering(\kappa)$$

and combine the refinement with the model consensus theorem to conclude

Corollary 3.2. *Let e be a distributed system obtained by composing n proposers, m acceptors, and k learners. For any T and σ, if $(e; \emptyset) \longrightarrow^* (T; \sigma)$ and both $ChosenI(messages(\sigma), v_1)$ and $ChosenI(messages(\sigma), v_2)$ hold then $v_1 = v_2$.*

Safety of Clients

The model is embedded as a resource in the logic so we can **also** exploit properties of the model **while** proving specifications.

$$\{I_{\mathsf{SDP}} * \ldots\}\ \langle ip; \texttt{client}\ a\rangle\ \{\ldots\}$$

```
let client addr =
  // ...
  let (_, v1) = client_deser m1 in
  let (_, v2) = client_deser m2 in
  assert (v1 = v2); v1.
```

Fair termination

Termination of **every** execution is too strong a notion for most **concurrent programs**.

Most concurrent programs only terminate if the scheduler is **fair.**

```
let rec yes b n = if cas b 1 0 then n := !n-1;
  if !n > 0 then yes b n

let rec no b m = if cas b 0 1 then m := !m-1;
  if !m > 0 then no b m

let start k = let b = ref 0 in
  (yes b (ref k) || no b (ref k))
```

Fair termination

A program trace is **fair** if its finite, or if its infinite and every **reducible** thread **eventually** takes a step.

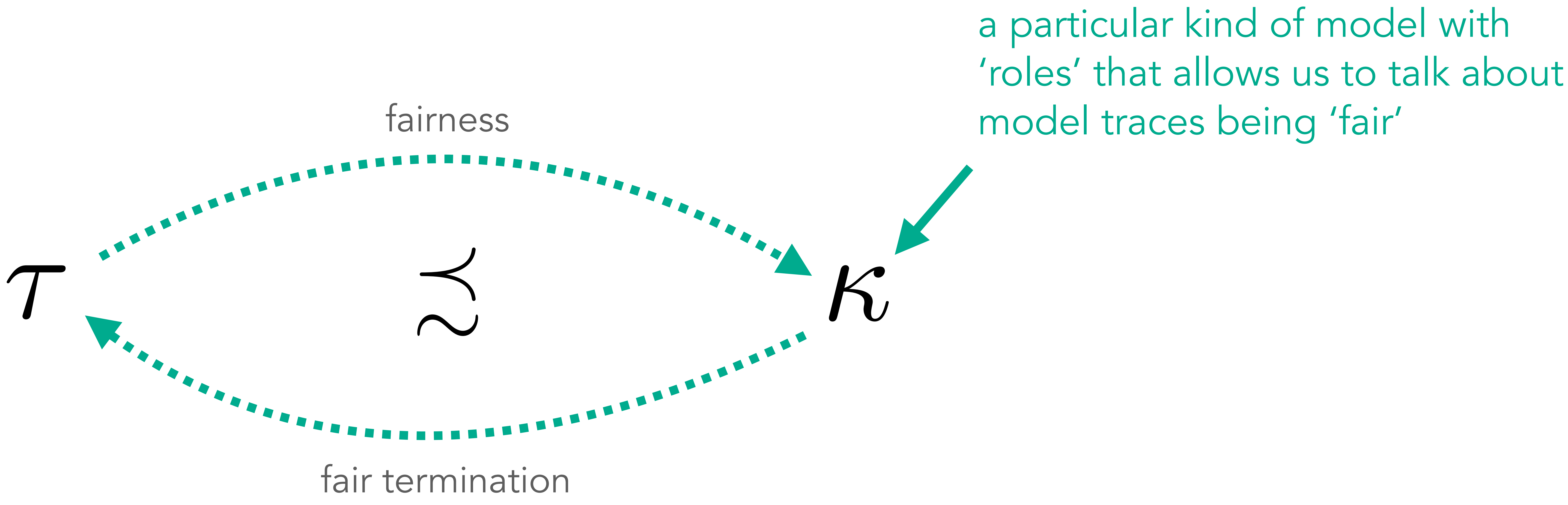
A program is **fairly terminating** if all its **fair traces** are finite.

But termination is a liveness property???

Fair termination

We prove fair termination by constructing a **fairness-preserving** and **termination-preserving** refinement:

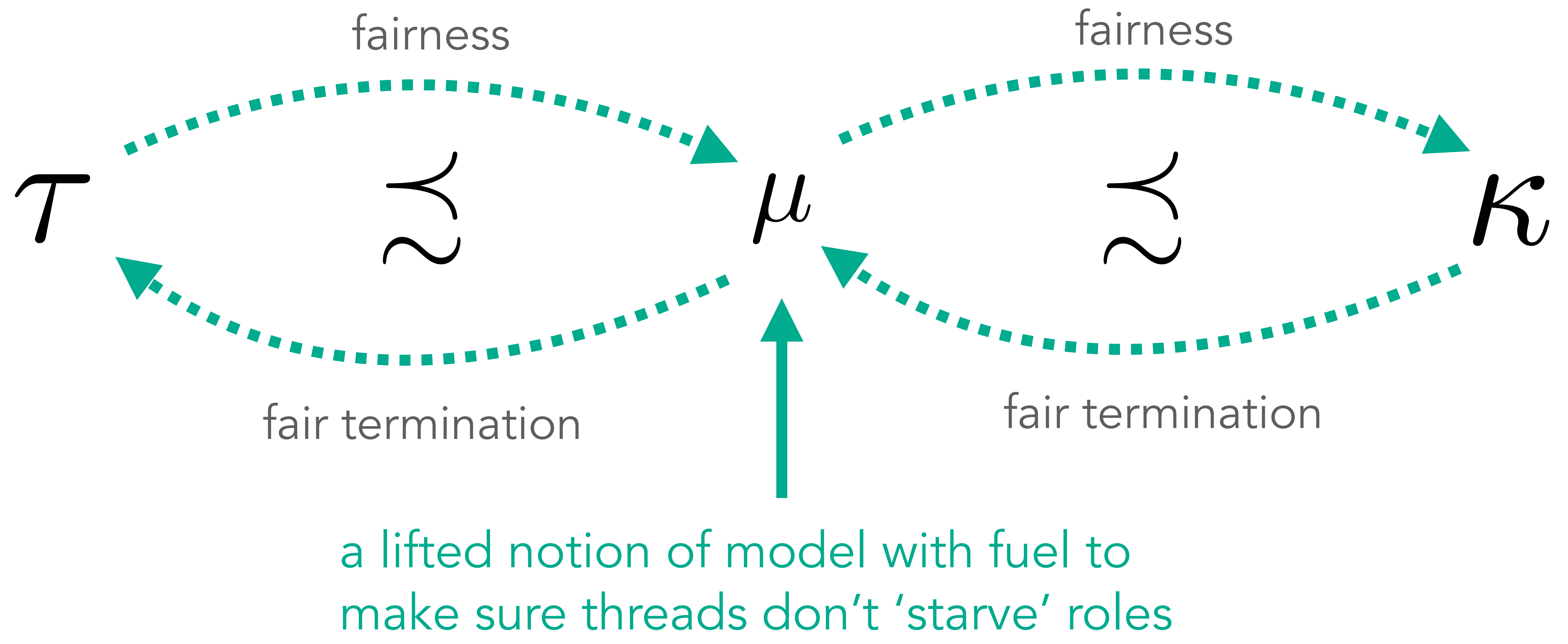
For all program traces τ *there exists a model trace* κ *such that*



Fair termination

We prove fair termination by constructing a **fairness-preserving** and **termination-preserving** refinement:

For all program traces τ there exists a model trace κ such that



Summary

- **Trillium**: a framework for showing **history-sensitive refinement** of programs and abstract models
- **Safety** and **liveness** properties of models can be transported to the implementation
- Instantiation with **AnerisLang** and **HeapLang**:
  - **Consensus** of single-decree Paxos
  - **Eventual consistency** of a CRDT
  - **Fair termination** of a concurrent program

AARHUS UNIVERSITY

CENTER FOR BASIC RESEARCH IN PROGRAM VERIFICATION

Thank you

Semantics of the Weakest Precondition

We generalise the notion of state interpretation to **trace interpretation**

$$S : \mathit{Trace}(\mathit{Cfg}) \times \mathit{Trace}(A_{\mathcal{M}}) \to \mathit{iProp}$$

and define

$$\begin{aligned}
\mathsf{wp}^{\mathcal{M}}_{\mathcal{E}}\, e\, \{\Phi\} \triangleq\ & (e \in \mathit{Val} * \Rrightarrow_{\mathcal{E}} \Phi(e)) \vee \\
& \Big(e \notin \mathit{Val} * \forall \tau, \tau', \kappa, \sigma, K, T_1, T_2. \\
& \quad \mathit{valid}(\tau) * \tau = (\tau' \cdot (T_1 \mathbin{+\!\!+} K[e] \mathbin{+\!\!+} T_2, \sigma)) * S(\tau, \kappa) \twoheadrightarrow {}^{\mathcal{E}}\Rrightarrow^{\emptyset} \\
& \qquad \mathit{reducible}(e, \sigma) * \\
& \qquad \Big(\forall e_2, \sigma_2, \vec{e_f}.\, (e, \sigma) \to (e_2, \sigma_2, \vec{e_f}) \twoheadrightarrow \triangleright\, {}^{\emptyset}\Rrightarrow^{\mathcal{E}} \\
& \qquad\quad \exists \delta.\, S(\tau \cdot (T_1 \mathbin{+\!\!+} K[e_2] \mathbin{+\!\!+} T_2 \mathbin{+\!\!+} \vec{e_f}, \sigma'), \kappa \cdot \delta) * \\
& \qquad\quad \mathsf{wp}^{\mathcal{M}}_{\mathcal{E}}\, e_2\, \{\Phi\} * \bigast_{e' \in \vec{e_f}} \mathsf{wp}^{\mathcal{M}}_{\mathcal{E}}\, e'\, \{\mathsf{True}\}\Big)\Big)
\end{aligned}$$

Remark

The standard Iris WP doesn't allow us to prove this kind of refinement.
We could prove, e.g.,

$$\left\{\boxed{\exists n.\, \ell \mapsto n * \left[\!\left[n : \textsc{MonoNat} \right]\!\right]^{\gamma}}\right\} \texttt{inc} \{\ldots\}$$

but this spec would also be satisfied by, e.g.,

```
let rec inc_loop () =
  let n = !ℓ in
  cas(ℓ, n, n + 2);
  inc_loop ()
in
  inc_loop () || inc_loop ()
```

$$Q1bv(\mathcal{S}, Q, b) \triangleq \{m \in \mathcal{S} \mid \exists a, v.\, m = \mathsf{msg1b}(a, b, \mathsf{Some}(v)) \land a \in Q\}$$

$$HavePromised(\mathcal{S}, Q, b) \triangleq \forall a \in Q.\, \exists m \in \mathcal{S}, o.\, m = \mathsf{msg1b}(a, b, o)$$

$$\begin{aligned} IsMaxVote(\mathcal{S}, Q, b, v) \triangleq\ & \exists m_0 \in Q1bv(\mathcal{S}, Q, b), a_0, b_0.\, m = \mathsf{msg1b}(a_0, b, \mathsf{Some}(b_0, v)) \land \\ & \forall m' \in Q1bv(\mathcal{S}, Q, b). \\ & \quad \exists a', b', v'.\, m' = \mathsf{msg1b}(a', b, \mathsf{Some}(b', v')) \land b_0 \geq b' \end{aligned}$$

$$ShowsSafeAt(\mathcal{S}, Q, b, v) \triangleq HavePromised(\mathcal{S}, Q, b) \land (Q1bv(\mathcal{S}, Q, b) = \emptyset \lor IsMaxVote(\mathcal{S}, Q, b, v))$$

SDP-Phase1a

$$\frac{}{\mathcal{S}, \mathcal{B}, \mathcal{V} \rightharpoonup_{\mathsf{SDP}} \mathcal{S} \cup \{\mathsf{msg1a}(b)\}, \mathcal{B}, \mathcal{V}}$$

SDP-Phase1b

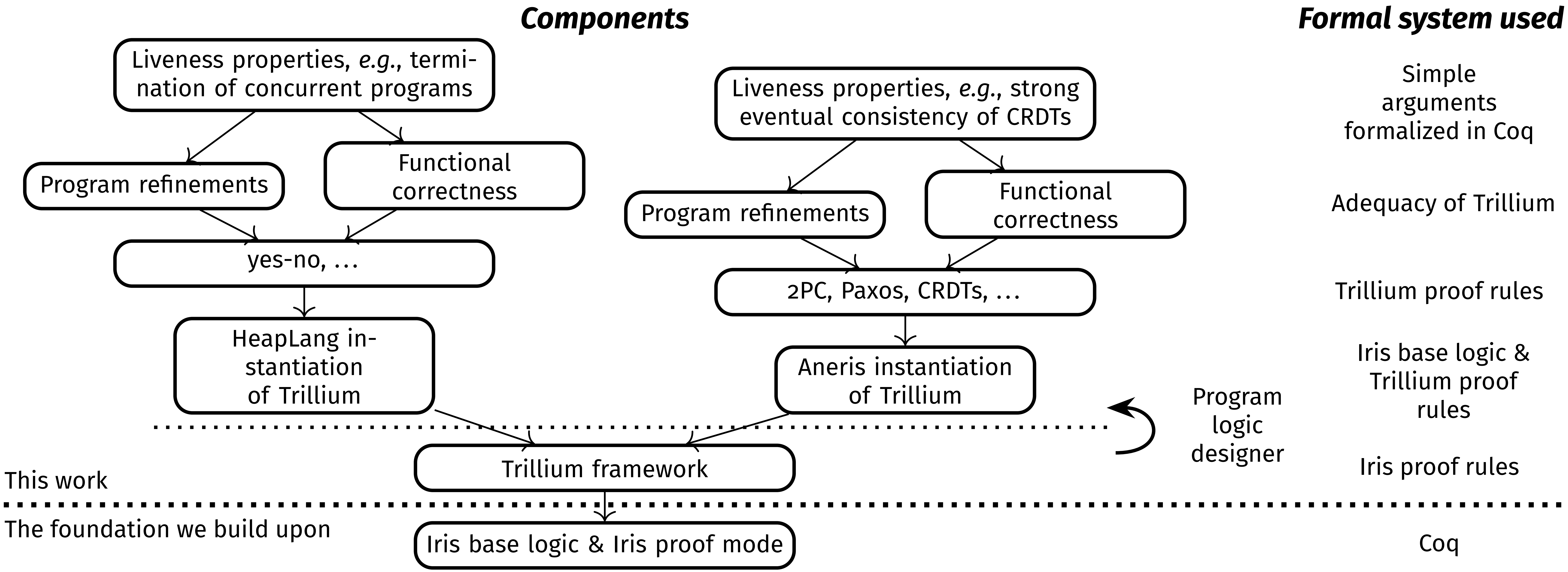
$$\frac{\mathsf{msg1a}(b) \in \mathcal{S} \qquad b > \mathcal{B}(a) \qquad \mathcal{V}(a) = o}{\mathcal{S}, \mathcal{B}, \mathcal{V} \rightharpoonup_{\mathsf{SDP}} \mathcal{S} \cup \{\mathsf{msg1b}(a, b, o)\}, \mathcal{B}[a \mapsto \mathsf{Some}(b)], \mathcal{V}}$$

SDP-Phase2a

$$\frac{\nexists v'.\, \mathsf{msg2a}(b, v') \in \mathcal{S} \qquad Quorum(Q) \qquad ShowsSafeAt(\mathcal{S}, Q, b, v)}{\mathcal{S}, \mathcal{B}, \mathcal{V} \rightharpoonup_{\mathsf{SDP}} \mathcal{S} \cup \{\mathsf{msg2a}(b, v)\}, \mathcal{B}, \mathcal{V}}$$

SDP-Phase2b

$$\frac{\mathsf{msg2a}(b, v) \in \mathcal{S} \qquad b \geq \mathcal{B}(a)}{\mathcal{S}, \mathcal{B}, \mathcal{V} \rightharpoonup_{\mathsf{SDP}} \mathcal{S} \cup \{\mathsf{msg2b}(a, b, v)\}, \mathcal{B}[a \mapsto \mathsf{Some}(b)], \mathcal{V}[a \mapsto \mathsf{Some}(b, v)]}$$



Components
Formal system used
Liveness properties, e.g., termination of concurrent programs
Liveness properties, e.g., strong eventual consistency of CRDTs
Simple arguments formalized in Coq
Program refinements
Functional correctness
Program refinements
Functional correctness
Adequacy of Trillium
yes-no, ...
2PC, Paxos, CRDTs, ...
Trillium proof rules
HeapLang instantiation of Trillium
Aneris instantiation of Trillium
Iris base logic & Trillium proof rules
Program logic designer
Trillium framework
Iris proof rules
This work
The foundation we build upon
Iris base logic & Iris proof mode
Coq