

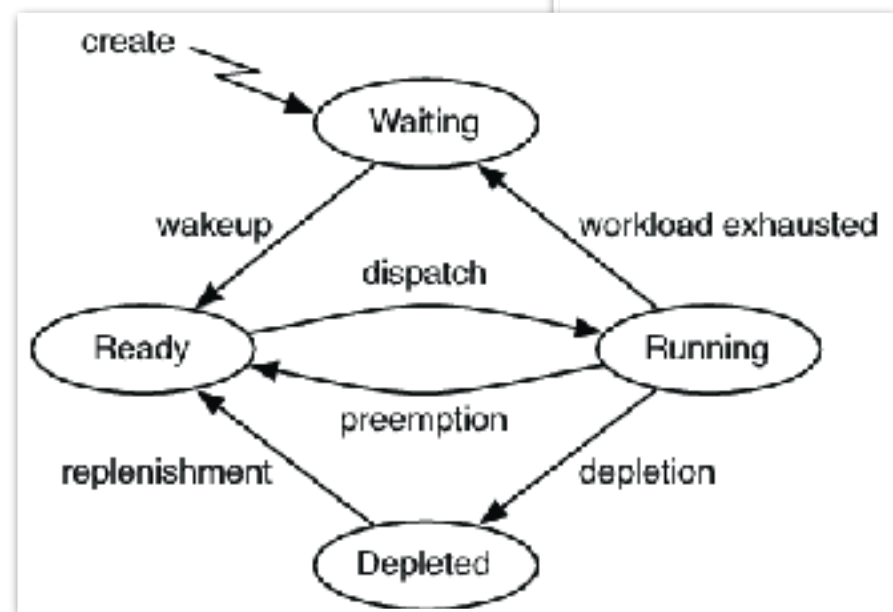
# Trillium

## History-Sensitive Refinement in Separation Logic

3 May, 2022

**Simon Oddershede Gregersen**

joint work with Amin Timany, Léo Stefanescu, Léon Gondelman, Abel Nieto, and Lars Birkedal



```

TLA+ Toolbox
File Edit Window T.C Model Checker TLA Proof Manager Help
Firewall.tla
C:\Users\chelsea\TDL\Firewall.tla
TLA Module
1 ----- MODULE Firewall -----
2 EXTENDS Integers
3 CONSTANTS Address, /* The set of all addresses
4 Port, /* The set of all ports
5 Protocol /* The set of all protocols
6
7 AddressRange == /* The set of all address ranges
8 Address \X Address
9
10 InAddressRange[r \in AddressRange, a \in Address] ==
11 /\ r[1] <= a
12 /\ a <= r[2]
13
14 PortRange == /* The set of all port ranges
15 Port \X Port
16
17 InPortRange[r \in PortRange, p \in Port] ==
18 /\ r[1] <= p
19 /\ p <= r[2]
20
21 Packet == /* The set of all packets
22 [sourceAddress : Address,
23 sourcePort : Port,
24 destAddress : Address,
25 destPort : Ports,
26 protocol : Protocol]
27
28 Firewall == /* The set of all firewalls
29 [Packet -> BOOLEAN]
30
31 Rule == /* The set of all firewall rules
32 [remoteAddress : AddressRange,
33 remotePort : PortRange,
34 localAddress : AddressRange,
35 localPort : PortRange,
36 protocol : SUBSET Protocol,
37 allow : SOULSAN]
38
39 Ruleset == /* The set of all firewall rulesets
40 SUBSET Rule
41
42 Allowed[rset \in Ruleset, p \in
43 LET matches == {rule \in rset
44 /\ InAddressRange[rule.r, p.sourceAddress]
45 /\ InPortRange[rule.r, p.sourcePort]
46 /\ InAddressRange[rule.l, p.destAddress]
47 /\ InPortRange[rule.l, p.destPort]
48 /\ p.protocol \in rule.p
49 /\ matches /= {}
50 IN /\ rule \in matches :
51 =====
52
53 // a small example spin model
54 // Peterson's solution to the mutual exclusion problem (1981)
55
56 bool turn, flag[2]; // the shared variables, booleans
57 byte ncrit; // nr of procs in critical section
58
59 active [2] proctype user() // two processes
60 {
61   assert(_pid == 0 || _pid == 1);
62   again:
63     flag[_pid] = 1;
64     turn = _pid;
65     (flag[1 - _pid] == 0 || turn == 1 - _pid);
66
67     ncrit++;
68     assert(ncrit == 1); // critical section
69     ncrit--;
70
71     flag[_pid] = 0;
72     goto again;
73 }
74 // analysis:
75 // ↓ spin -run peterson.pn1
  
```

TLA+ Parser Errors  
Line 25, col 16 to line 25, col 20 of module Firewall  
Unknown operator: 'Ports'.

The type "2a" message sent by this action therefore tells every acceptor a that, when it receives the message, all the enabling conditions of VoteFor(a, b, v) but the first,  $maxBal[a] \leq b$ , are satisfied.

$$Phase2a(b, v) \triangleq$$

$$\wedge \neg \exists m \in msgs : m.type = "2a" \wedge m.bal = b$$

$$\wedge \exists Q \in Quorum :$$

$$LET Q1b \triangleq \{m \in msgs : m.type = "1b" \wedge m.acc \in Q \wedge m.bal = b\}$$

$$Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\}$$

$$IN \wedge \forall a \in Q : \exists m \in Q1b : m.acc = a$$

$$\wedge \forall Q1bv = \{ \}$$

$$\forall \exists m \in Q1bv :$$

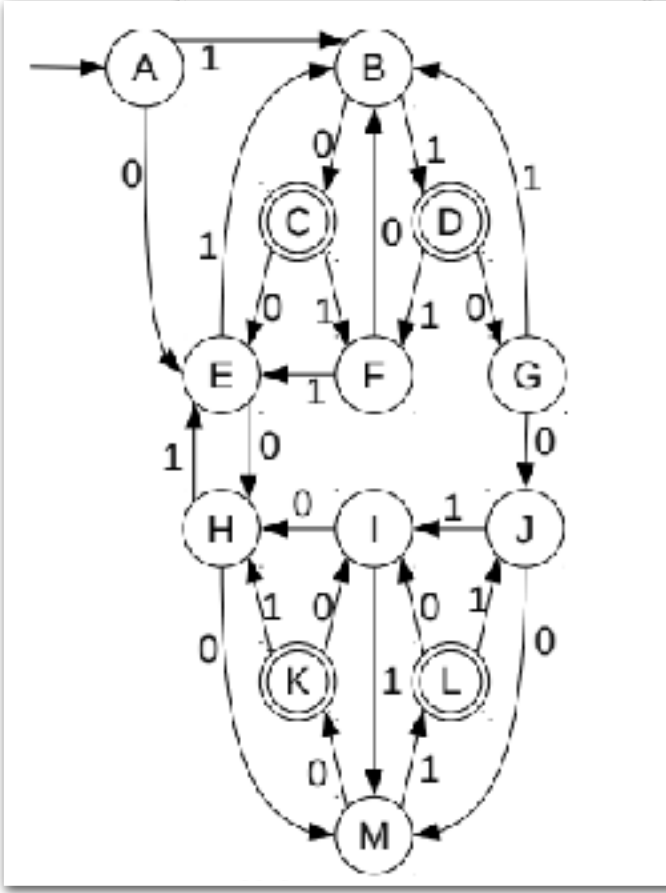
$$\wedge m.mval = v$$

$$\wedge \forall mm \in Q1bv : m.mbal \geq mm.mbal$$

$$\wedge Send(type \mapsto "2a", bal \mapsto b, val \mapsto v)$$

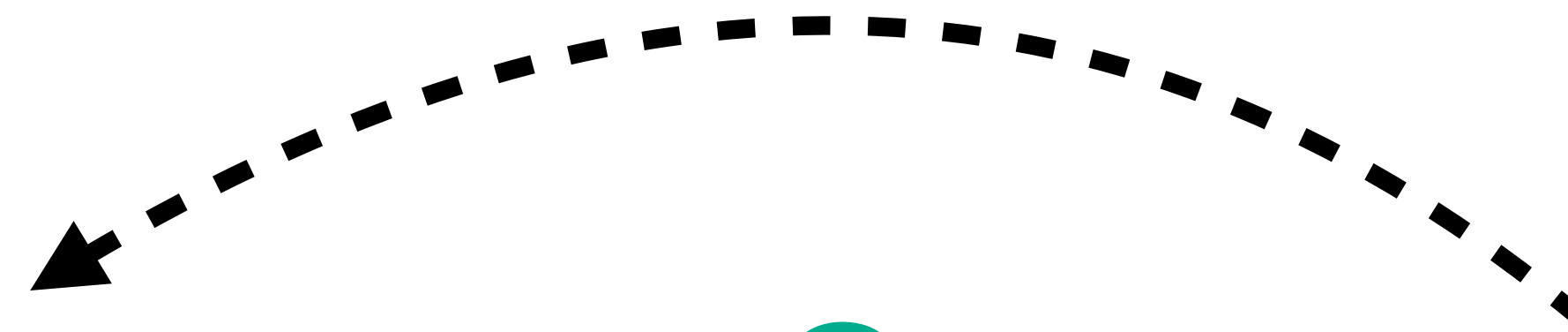
$$\wedge UNCHANGED (maxBal, maxVBal, maxVal)$$

The Phase2b(a) action describes what acceptor a does when it receives a phase 2a message m, which is sent by the leader of ballot m.bal asking acceptors to vote for m.val in that ballot. Acceptor a acts on that request, voting for m.val in ballot number m.bal, iff  $m.bal \geq maxBal[a]$ , which means that a has not participated in any ballot numbered greater than m.bal. Thus, this enabling condition of the Phase2b(a) action together with the receipt of the phase 2a message m enables the VoteFor(a, m.bal, m.val) action of module Voting is enabled and can be executed. message updates  $maxBal[a]$ ,  $maxVBal[a]$ , and  $maxVal[a]$  so their values mean claimed to mean in the comments preceding the variable declarations.



Models, **not** implementations!

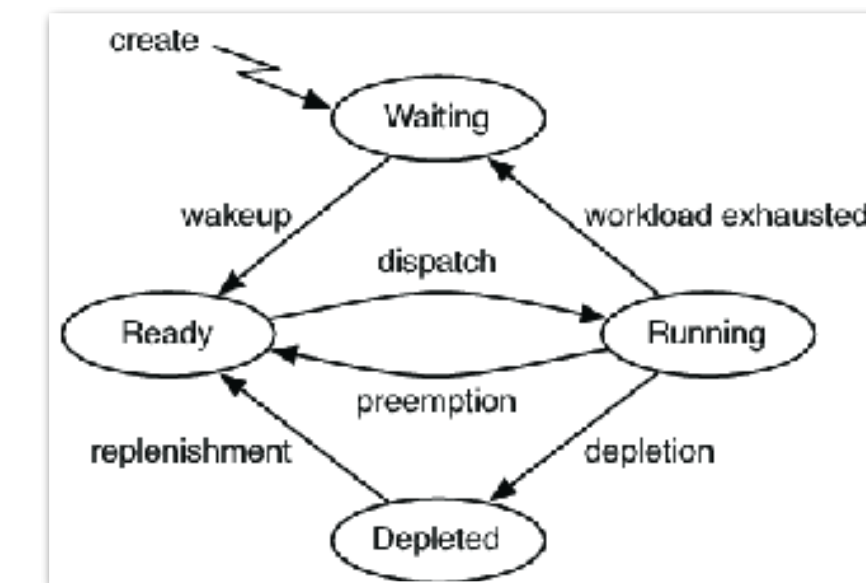
Transport properties?



```
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 (n, sndr) = receivefrom skt in
match acceptor_deser n 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 (n1, sndr1) = receivefrom skt in
let (_, v1) = client_deser n1 in
let (n2, _) = wait_receivefrom skt
(fun (_, sndr2), sndr2 <=> sndr1) in
let (_, v2) = client_deser n2 in
assert (v1 = v2); v1.
```



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

... using Iris, obviously

# Outline

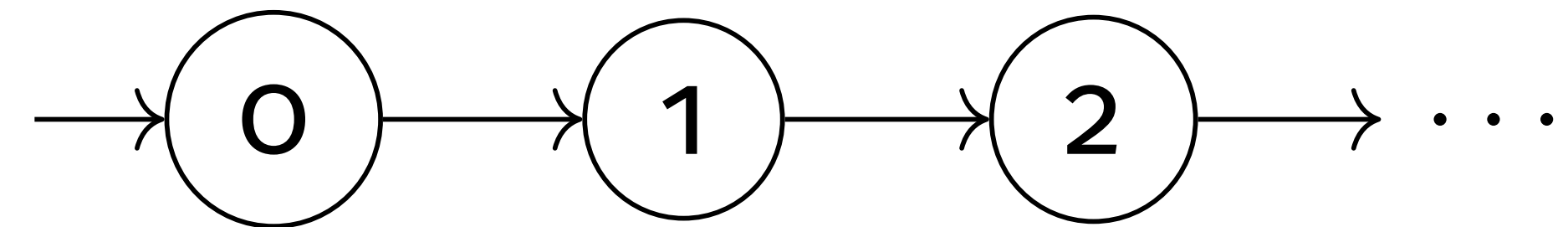
- ▶ The Trillium methodology
- ▶ **Case study:** Single-decree Paxos using a TLA+ model
- ▶ **Case study:** Fair termination of a concurrent program

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



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

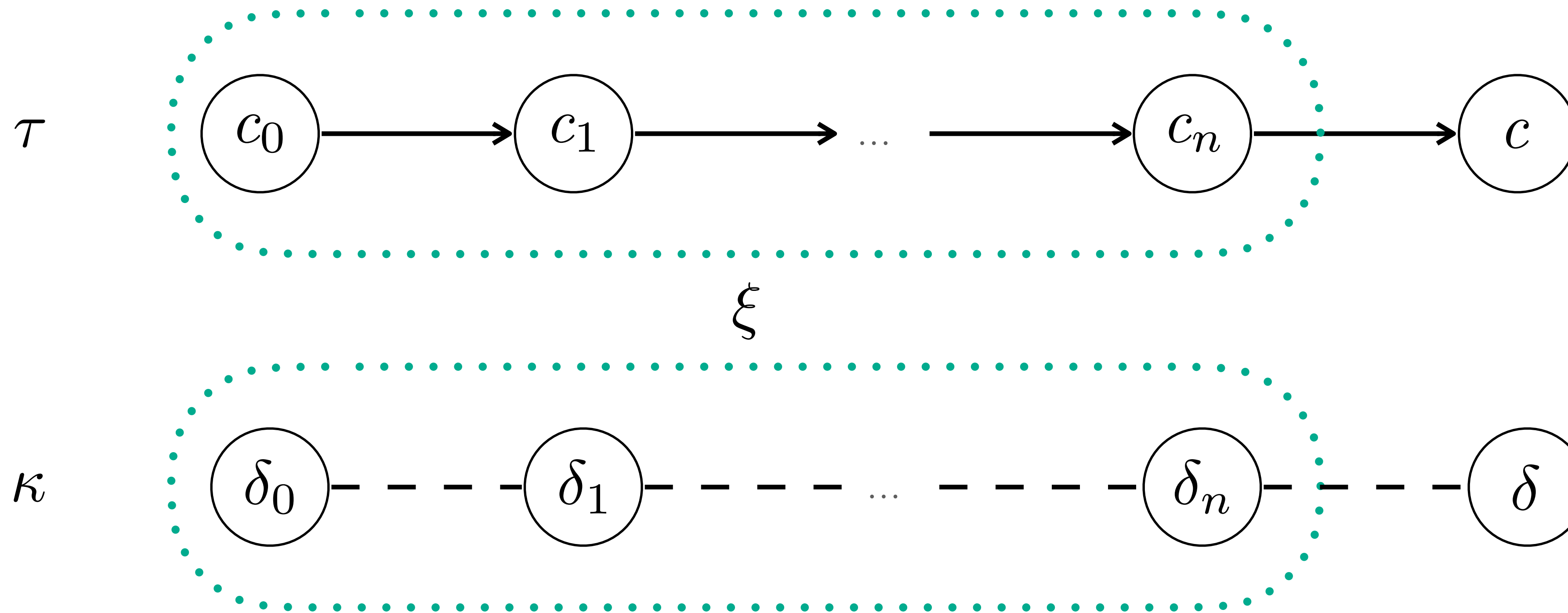
# Definition

Given **relation btw. traces**  $\xi$   
**execution trace**  $\mathcal{T}$  (non-empty sequence of configurations)  
**model trace**  $\kappa$  (non-empty sequence of model states)

$\mathcal{T}$  is a **history-sensitive refinement** of  $\kappa$  under  $\xi$  whenever

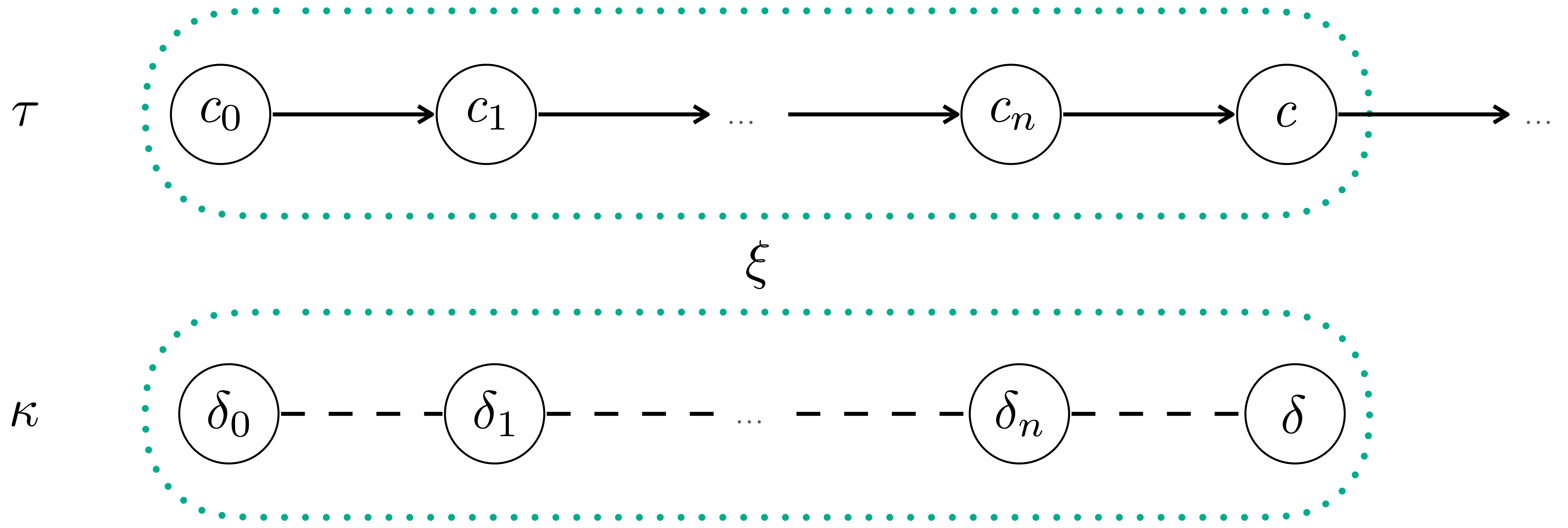
$$\mathcal{T} \lesssim_{\xi} \kappa \triangleq \xi(\mathcal{T}, \kappa) \wedge \forall c. \text{last}(\mathcal{T}) \rightarrow c \Rightarrow \exists \delta. \mathcal{T} \cdot c \lesssim_{\xi} \kappa \cdot \delta$$

holds coinductively.



$$\tau \underset{\xi}{\sim} \kappa \triangleq \xi(\tau, \kappa) \wedge \forall c. \text{last}(\tau) \rightarrow c \Rightarrow \exists \delta. \tau \cdot c \underset{\xi}{\sim} \kappa \cdot \delta$$





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




# Running Example

For our running example, we pick

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

where

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

stepping relation of the STS  


which reduces refinement to a notion of simulation.

# Trillium

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

$$\text{wp}^{\mathcal{M}} e \{Q\}$$

$$\text{Model}(\delta : \mathcal{M})$$

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

# Trillium

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

$$\frac{\{P\} e \{Q\}^{\mathcal{M}} \quad \delta \rightarrow_{\mathcal{M}} \delta' \quad \text{Atomic}(e) \quad e \notin \text{Val}}{\{P * \text{Model}(\delta)\} e \{Q * \text{Model}(\delta')\}^{\mathcal{M}}}$$

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

using the usual encoding of Hoare triples.

# Running Example

We show

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

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**,  $\sigma$  a **state**,  $\delta$  a **model state** and  $\xi$  a **finitary** trace relation.  
Suppose

$$\text{Tr} \Rightarrow_{\top} \mathcal{S}((e, \sigma), \delta) * \text{wp}_{\top}^{\mathcal{M}} e \{ \Phi \} * \text{AlwaysHolds}(\xi)$$

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

$$\text{AlwaysHolds}(\xi) \triangleq \forall \tau, \kappa. (\dots) \rightarrow * \text{Tr} \Rightarrow^{\emptyset} \xi(\tau, \kappa)$$

# 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  $\mathcal{S} \in \mathcal{P}(\text{PaxosMessage})$  is the set of sent messages
- ▶ Transitions, e.g.,

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

**THEOREM 3.1 (CONSISTENCY, SDP MODEL).** *Let  $\iota_{\text{SDP}} = (\emptyset, \lambda_{\_}. \text{None}, \lambda_{\_}. \text{None})$ . If  $\iota_{\text{SDP}} \rightarrow_{\text{SDP}}^* (\mathcal{S}, \mathcal{B}, \mathcal{V})$  and both  $\text{Chosen}(\mathcal{S}, v_1)$  and  $\text{Chosen}(\mathcal{S}, v_2)$  hold then  $v_1 = v_2$ .*



# Paxos Specs

$$\{I_{SDP} * \text{MaxBal}_o(a, \text{None}) * \text{MaxBal}_o(a, \text{None}) * \dots\} \langle ip; \text{acceptor } L a \rangle \{\text{False}\}$$

$$\{I_{SDP} * \text{pending}(b) * \dots\} \langle ip; \text{proposer } A \text{ skt } b v \rangle \{\text{True}\}$$

where

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

resolves underspecified aspect of the model

maps model messages to sent messages in the implementation

$$\frac{\exists v'. \text{msg2a}(b, v') \in \mathcal{S} \quad \text{Quorum}(Q) \quad \text{ShowsSafeAt}(\mathcal{S}, Q, b, v)}{\mathcal{S}, \mathcal{B}, \mathcal{V} \rightarrow_{SDP} \mathcal{S} \cup \{\text{msg2a}(b, v)\}, \mathcal{B}, \mathcal{V}}$$

# Paxos Refinement

Pick

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

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  $\sigma$ , if  $(e; \emptyset) \rightarrow^* (T; \sigma)$  and both  $\text{ChosenI}(\text{messages}(\sigma), v_1)$  and  $\text{ChosenI}(\text{messages}(\sigma), v_2)$  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.

```
{ISDP * ...} ⟨ip; client a⟩ {...}
```

```
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.

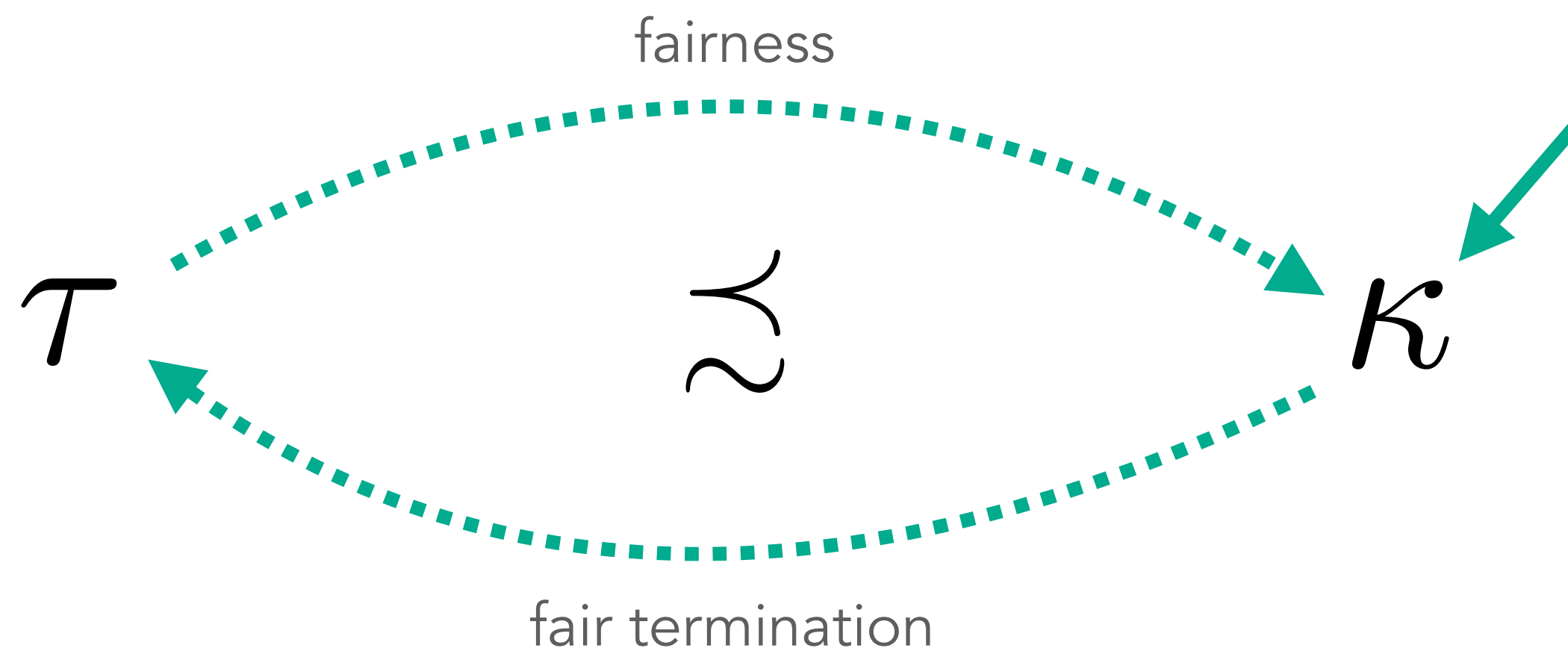
A program is **fairly terminating** if all its **fair traces** are finite.

**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  $\tau$  there exists a model trace  $\kappa$  such that*

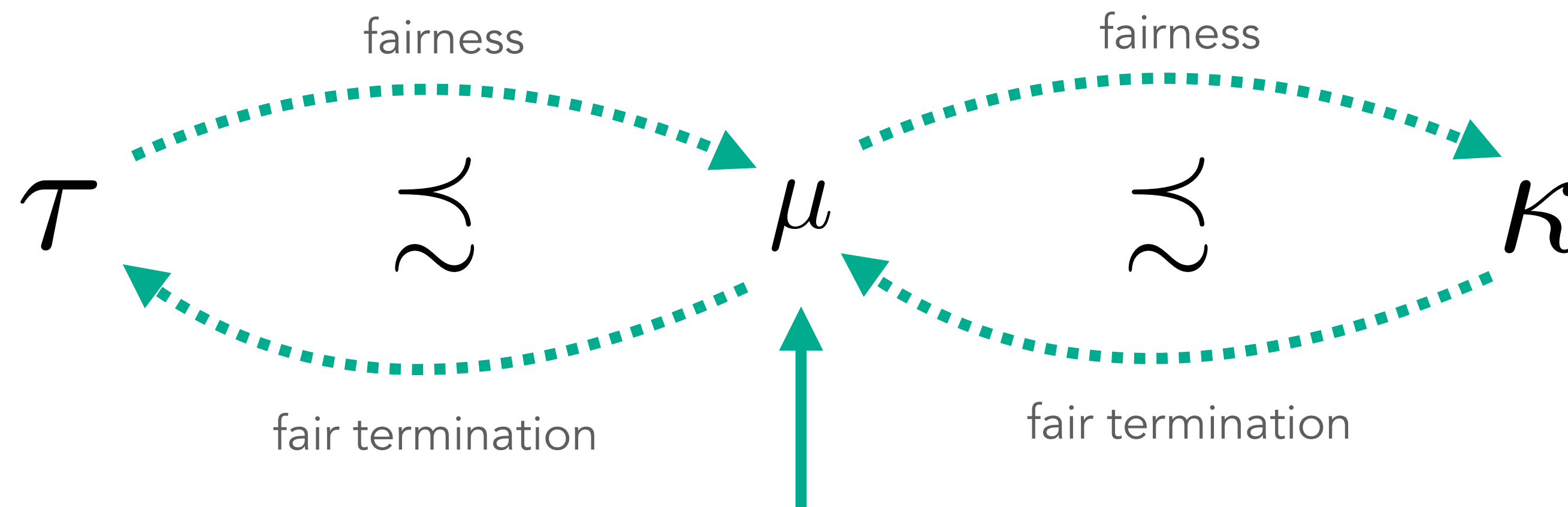


a particular kind of model with 'roles' that allows us to talk about model traces being 'fair'

# Fair termination

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

*For all program traces  $\tau$  there exists a model trace  $\kappa$  such that*



a lifted notion of model with fuel to  
make sure threads don't 'starve' roles



# 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

# Thank you

# Semantics of the Weakest Precondition

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

$$S : \text{Trace}(\text{Cfg}) \times \text{Trace}(A_{\mathcal{M}}) \rightarrow \text{iProp}$$

and define

$$\begin{aligned} \text{wp}_{\varepsilon}^{\mathcal{M}} e \{ \Phi \} \triangleq & (e \in \text{Val} * \Vdash_{\varepsilon} \Phi(e)) \vee \\ & \left( e \notin \text{Val} * \forall \tau, \tau', \kappa, \sigma, K, T_1, T_2. \right. \\ & \quad \text{valid}(\tau) * \tau = (\tau' \cdot (T_1 \dashv\vdash K[e] \dashv\vdash T_2, \sigma)) * S(\tau, \kappa) * \varepsilon \Vdash^{\emptyset} \\ & \quad \text{reducible}(e, \sigma) * \\ & \quad \left( \forall e_2, \sigma_2, \vec{e}_f. (e, \sigma) \rightarrow (e_2, \sigma_2, \vec{e}_f) * \triangleright^{\emptyset} \Vdash^{\varepsilon} \right. \\ & \quad \quad \left. \exists \delta. S(\tau \cdot (T_1 \dashv\vdash K[e_2] \dashv\vdash T_2 \dashv\vdash \vec{e}_f, \sigma'), \kappa \cdot \delta) * \right. \\ & \quad \quad \left. \left. \text{wp}_{\varepsilon}^{\mathcal{M}} e_2 \{ \Phi \} * \bigstar_{e' \in \vec{e}_f} \text{wp}_{\varepsilon}^{\mathcal{M}} e' \{ \text{True} \} \right) \right) \end{aligned}$$

# Remark

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

$$\left\{ \boxed{\exists n. \ell \mapsto n * \boxed{n : \text{MONONAT}}^\gamma} \right\} \text{inc } \{\dots\}$$

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 ()
```

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

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

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

$$\exists a', b', v'. m' = \text{msg1b}(a', b, \text{Some}(b', v')) \wedge b_0 \geq b'$$

$$\text{ShowsSafeAt}(\mathcal{S}, Q, b, v) \triangleq \text{HavePromised}(\mathcal{S}, Q, b) \wedge (Q1bv(\mathcal{S}, Q, b) = \emptyset \vee \text{IsMaxVote}(\mathcal{S}, Q, b, v))$$

SDP-PHASE1A

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

SDP-PHASE1B

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

SDP-PHASE2A

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

SDP-PHASE2B

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

