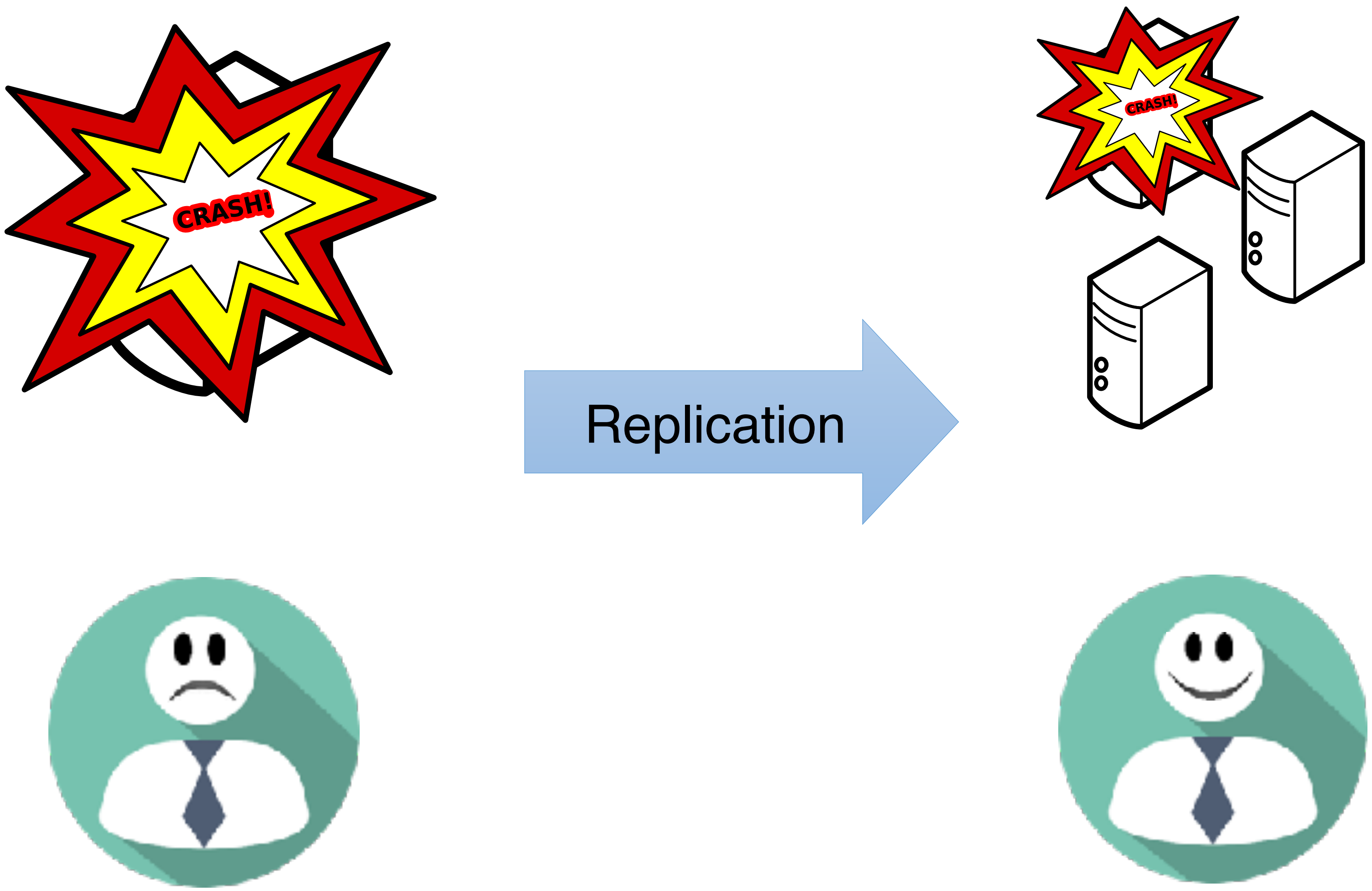
Programming abstractions for automating the verification of replicated state machine

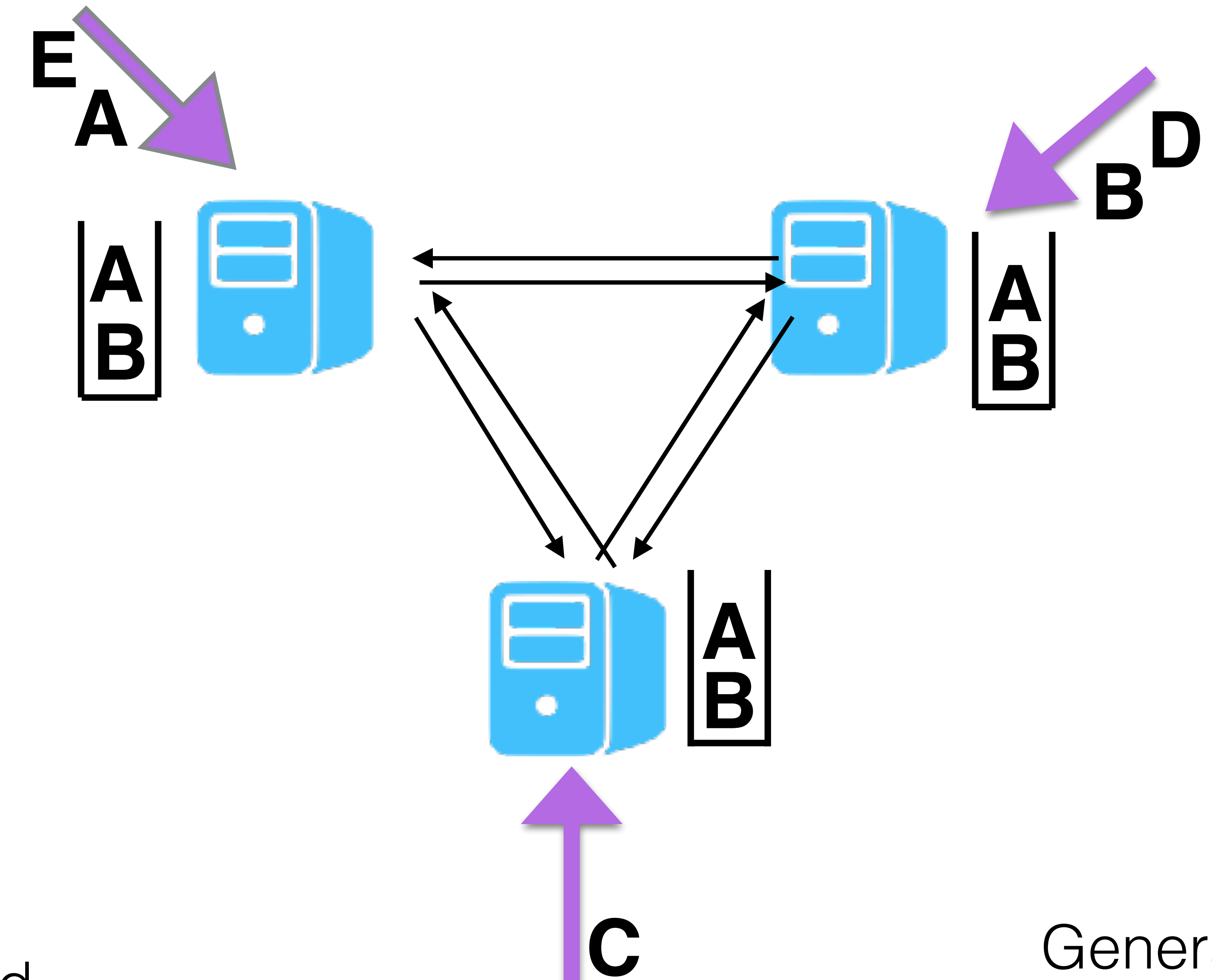
Cezara Drăgoi,   INRIA Paris/ENS

joint work with
Tom Henzinger, IST Austria
Josef Widder,   TU Wien
Damien Zufferey, MPI Kaiserslautern

Replication



Replicated state machine



Zab
Viewstamped

Generalized Paxos

Goal

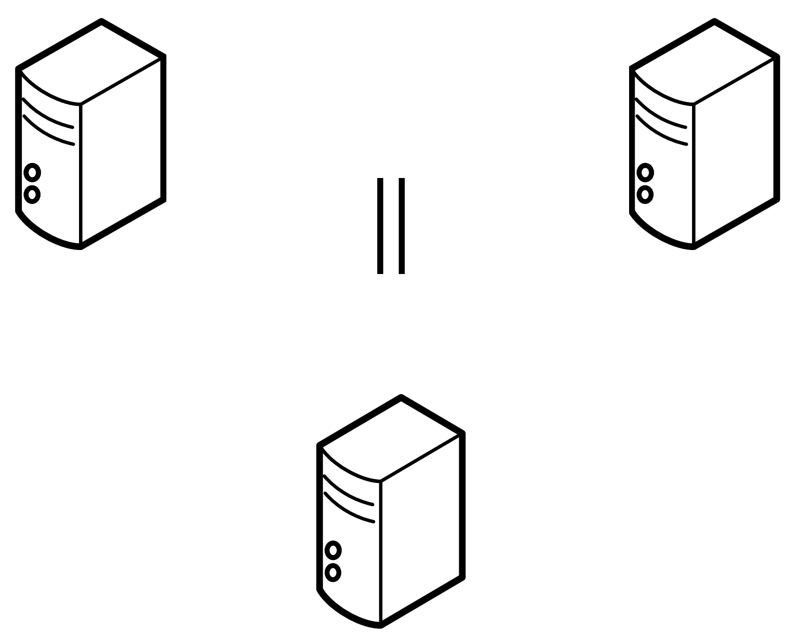
Design **automated verification** for **implementations** of replicated state machine

Difficulties

- **Impossibility result** [FLP'85]: "Consensus cannot be reached in asynchronous networks in the presence of at least one faulty process"
  - protocols have been developed for **different network assumptions:** Different degrees of synchrony and faults

Difficulties: Complex control structure

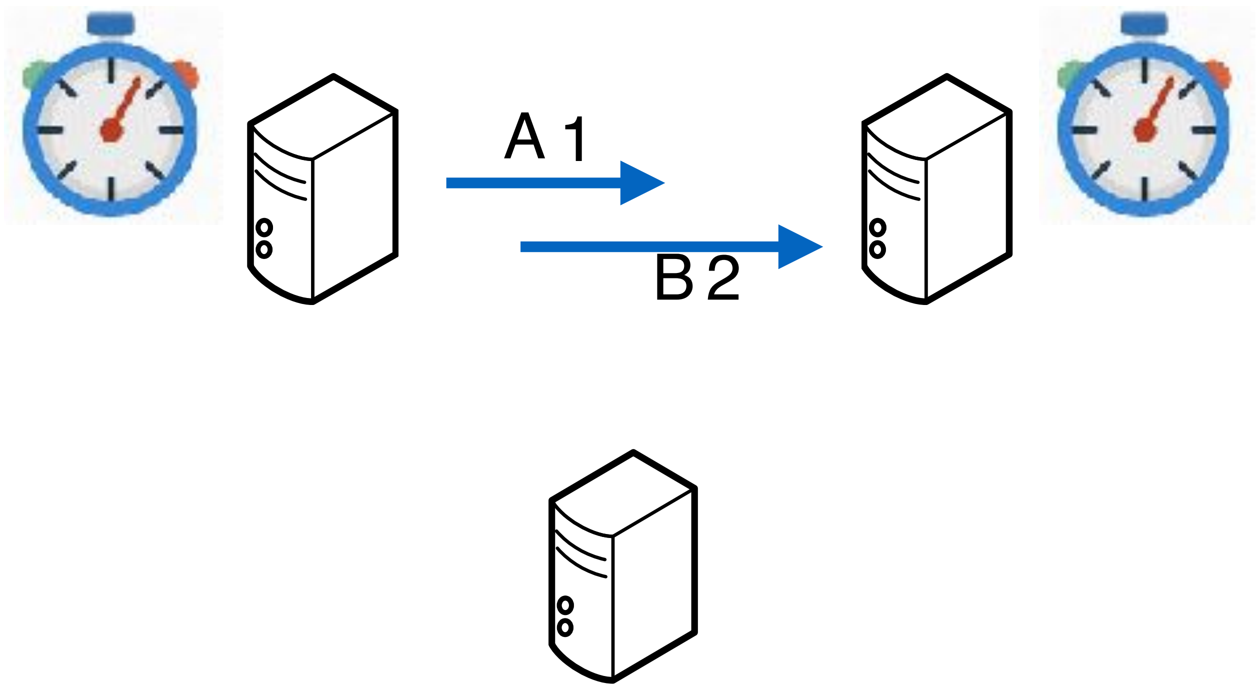
- asynchronous parallel composition



Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passing: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message ?

what if two messages are received in the reverse sending order ?

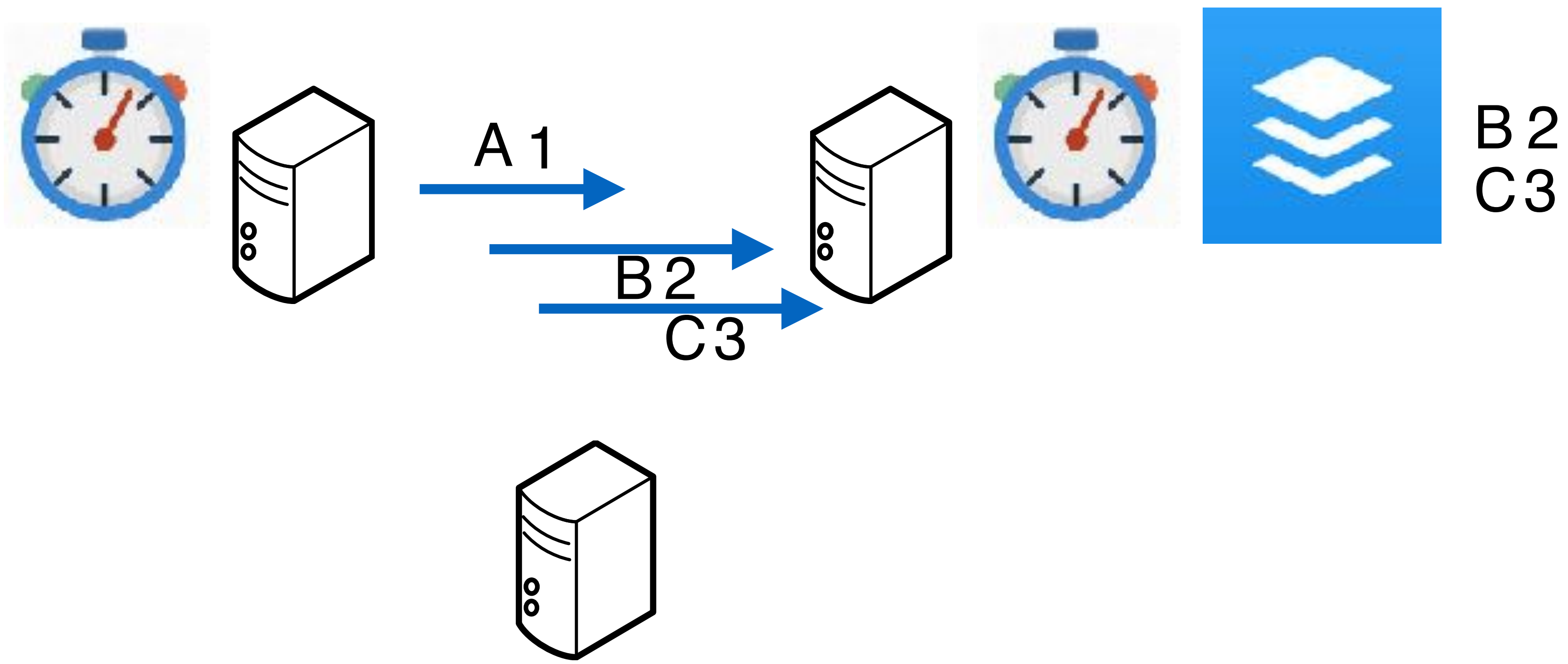


Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passings: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message ?

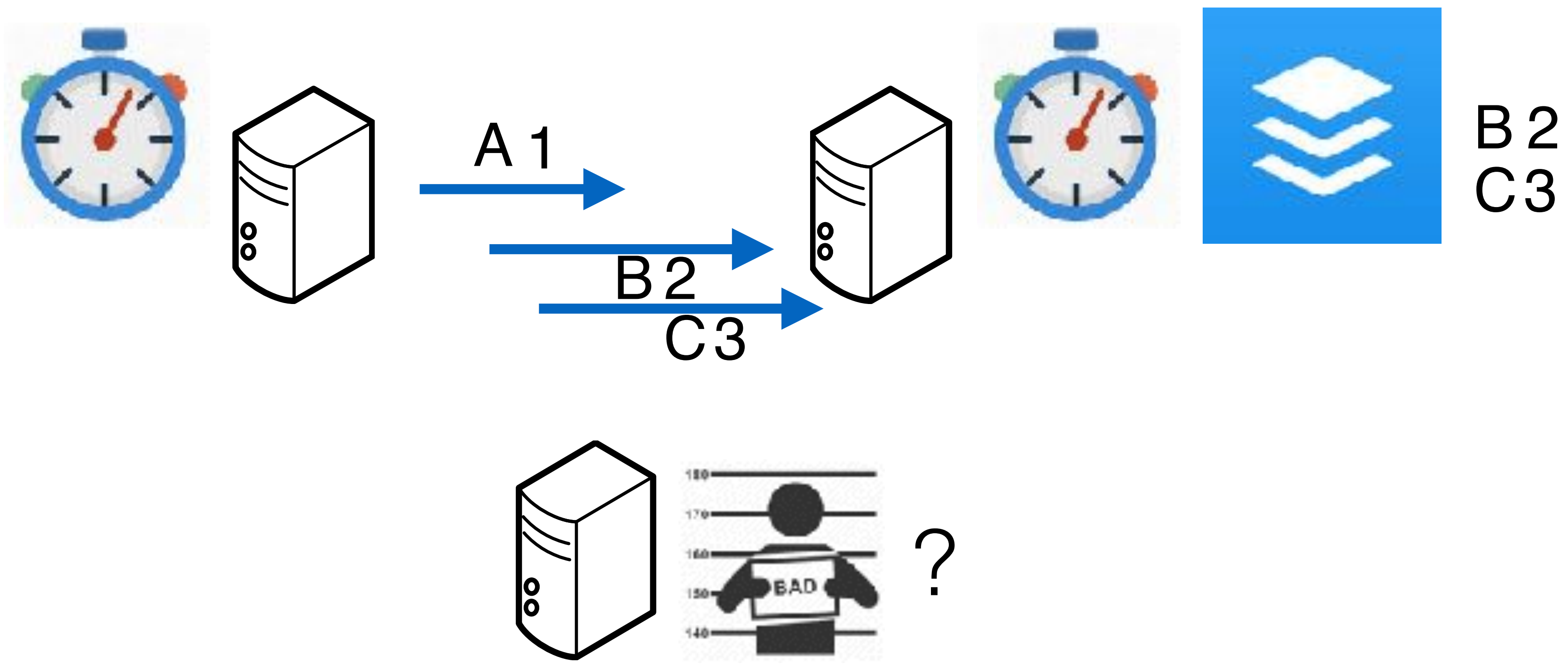
what if two messages are received in the reverse order ?

- unbounded buffers: A process receives a bunch of messages that momentarily it does not need.



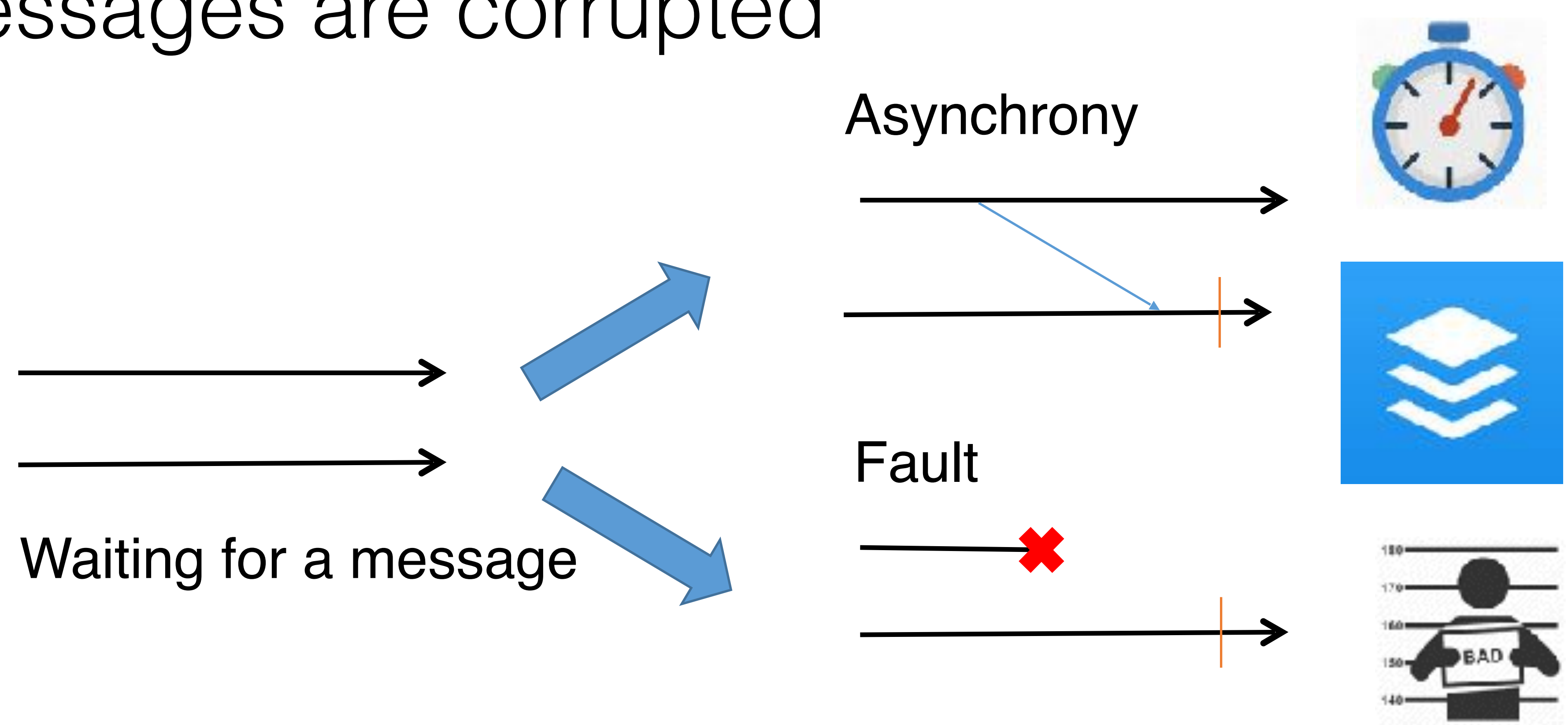
Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passings: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message ? what if two messages are received in the reverse order ?
- unbounded buffers: A process receives a bunch of messages that momentarily it does not need.
- faults: message drop, process-crash, messages are corrupted



Difficulties: Complex control structure

- asynchronous parallel composition
- communication via message passings: peer-to-peer, broadcast
- timer constraints: how long does a process wait for a message ? what if two messages are received in the reverse order ?
- unbounded buffers: A process receives a bunch of messages that momentarily it does not need.
- faults: message drop, process-crash, messages are corrupted



State of the art in verification of RSM

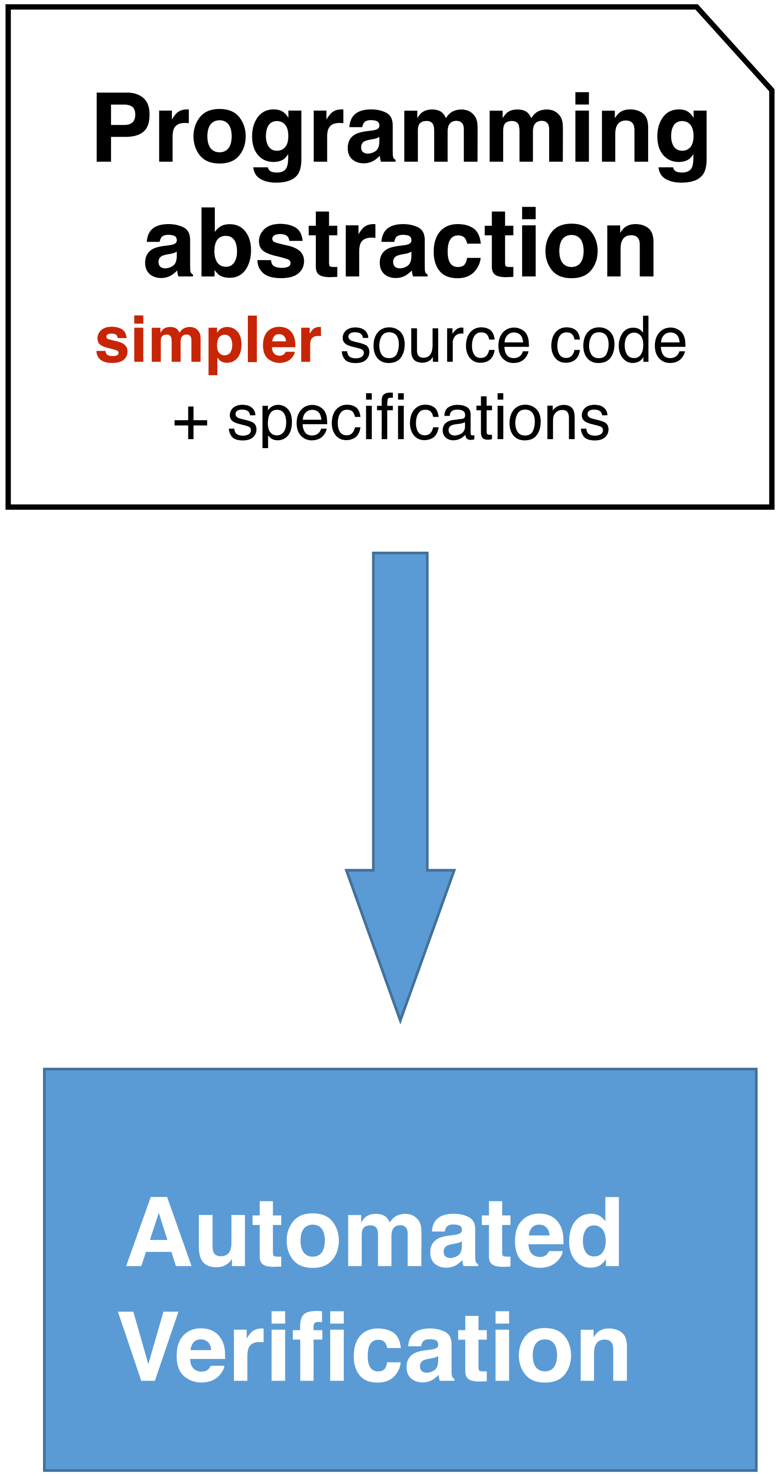
Mechanized verification:

- Verdi, EventML(2012) (only safety)
- IronFleet(2013)
- TLA+

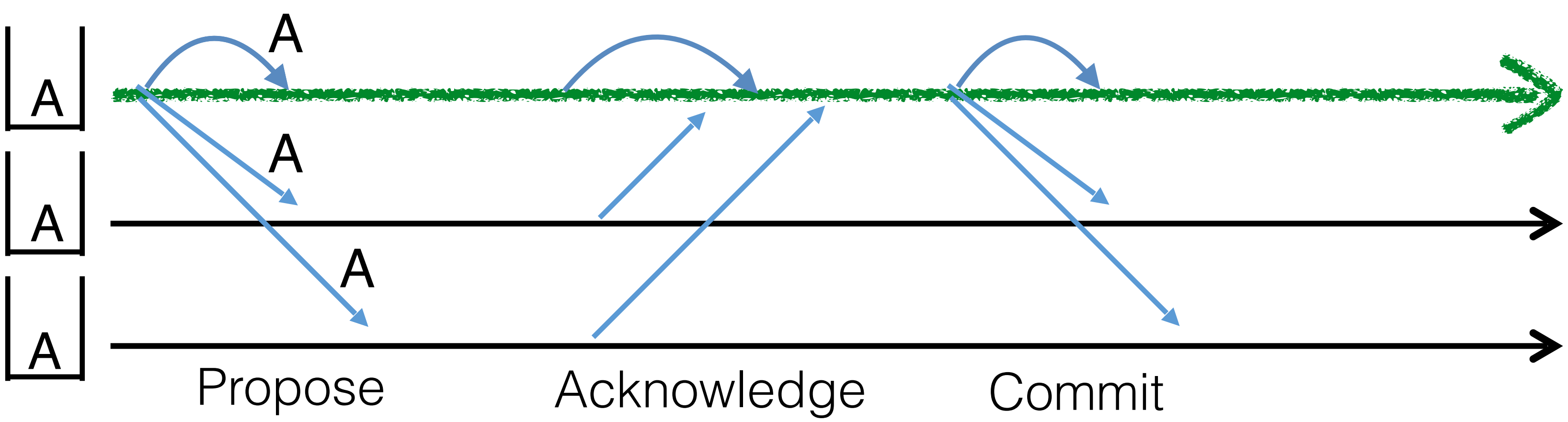
Automated verification:

- model checking of simple algorithms or protocols using minor coordination
- Ivy(2016)
- Psync

Goal


Programming abstraction
simpler source code
+ specifications
Automated Verification

Example

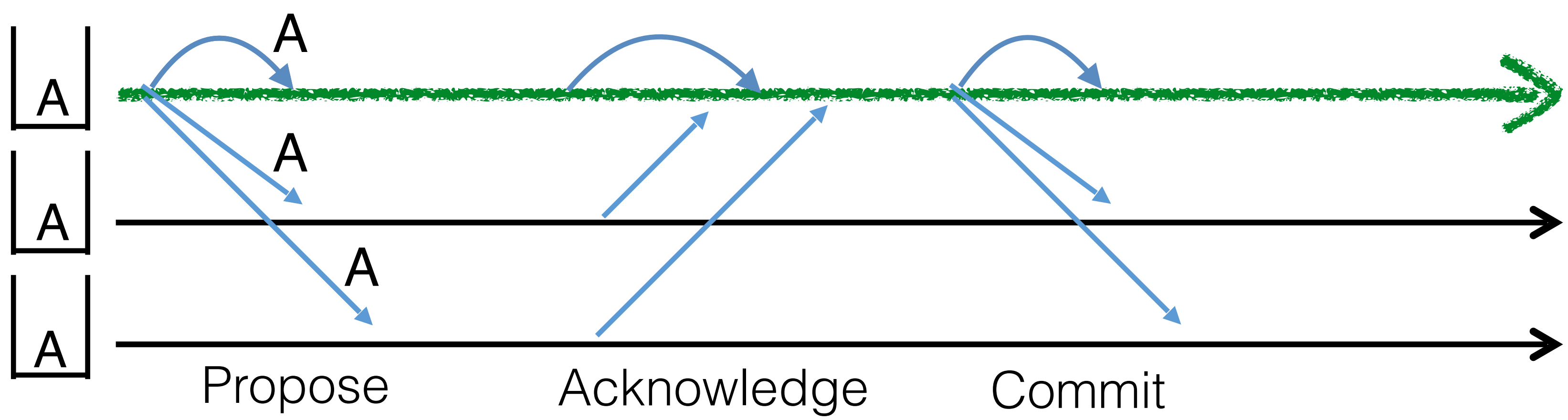


Assumptions: synchronous communication and no faults

there is a bound **b** on the message delay

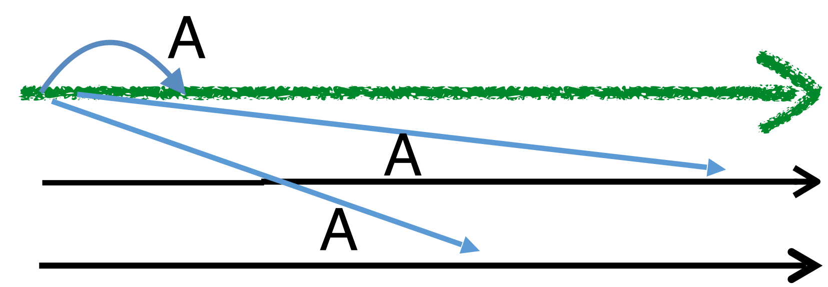
Assumptions: the leader receives the messages from all processes and every process hears from the leader

Example: Communication



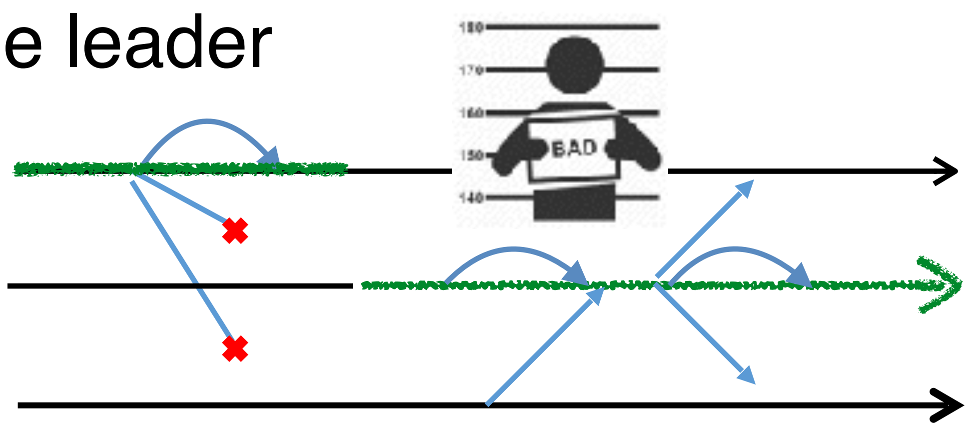
every process hears from the leader

Wait longer



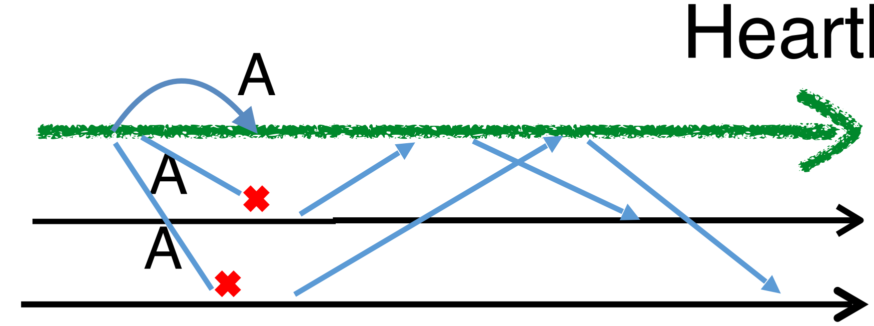
Propose/Commit

Change leader



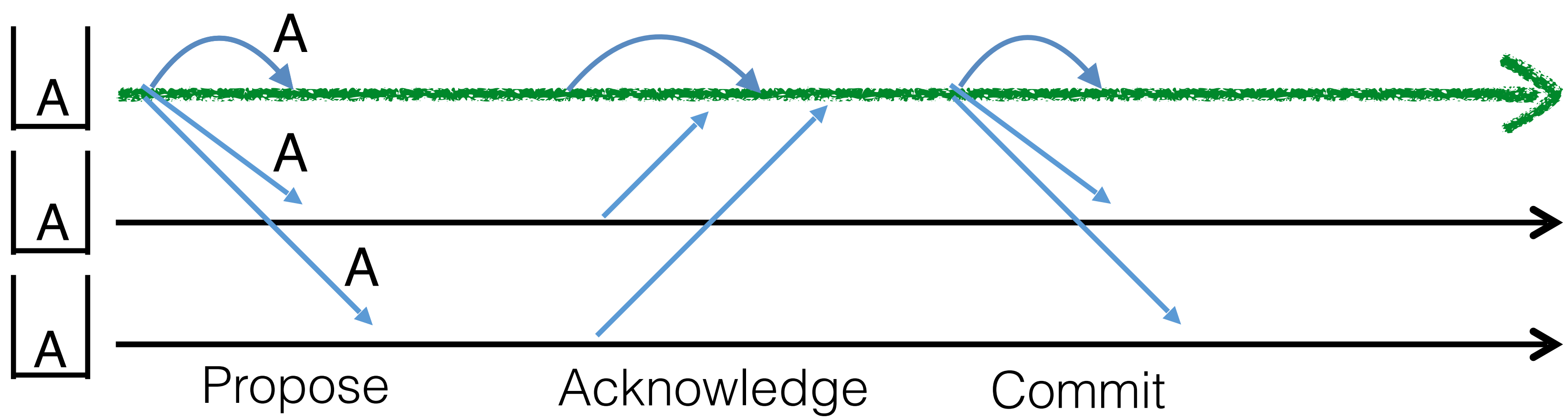
Propose/Commit

Heartbeats



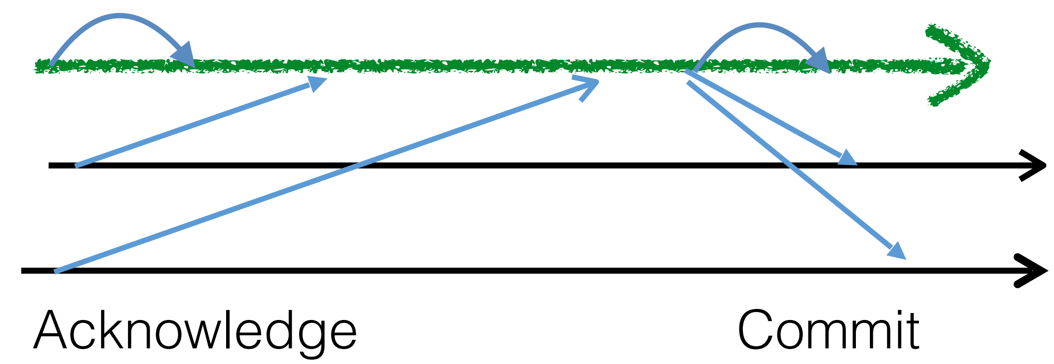
Propose/Commit

Example: Communication

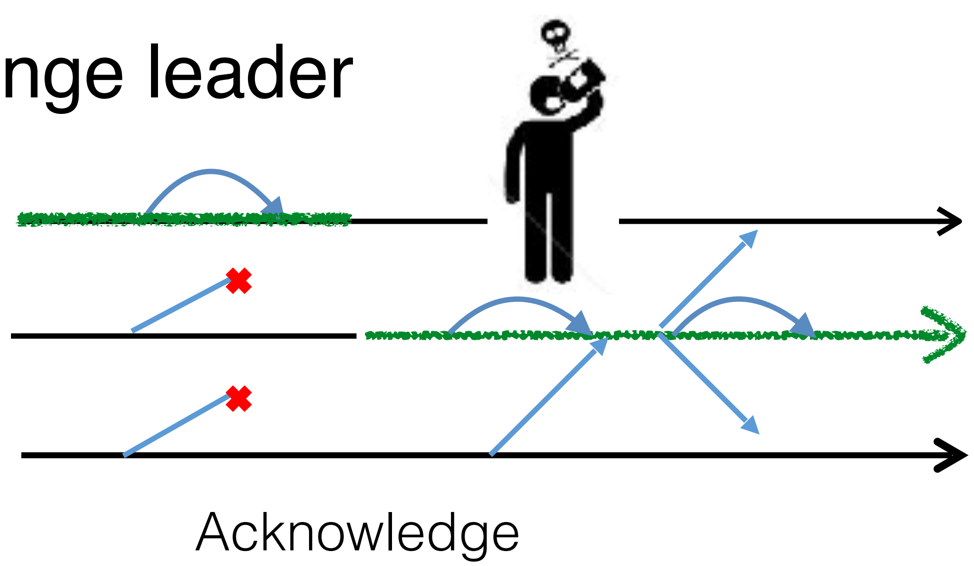


the leader
hears from all processes

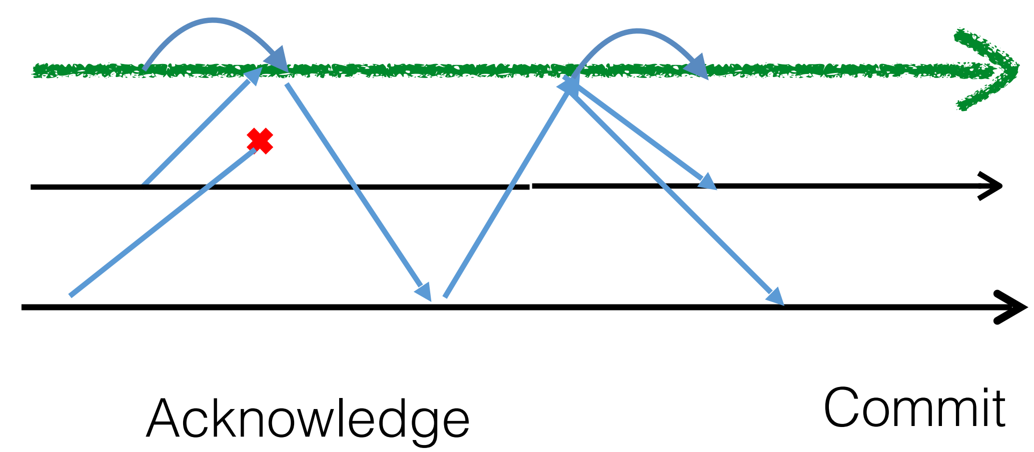
Wait longer



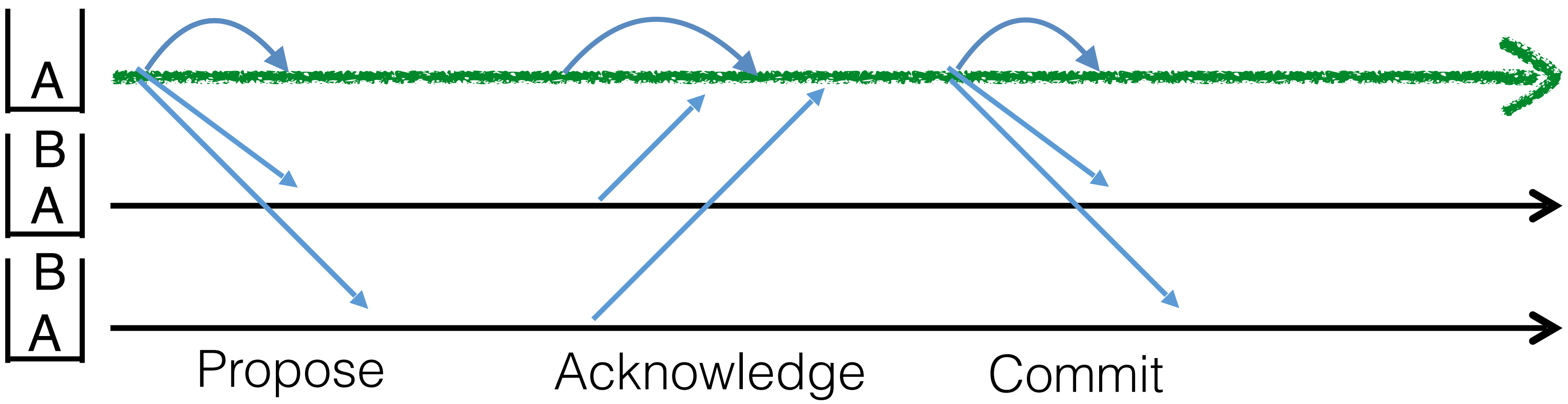
Change leader



Heartbeats



Example



We would like a round based model that separates the ***network properties*** from the <u>algorithmic computation</u>

the leader hears from all processes,
every process hears from the leader

<u>compute the minimum value,</u>
<u>a commun prefix, etc.</u>

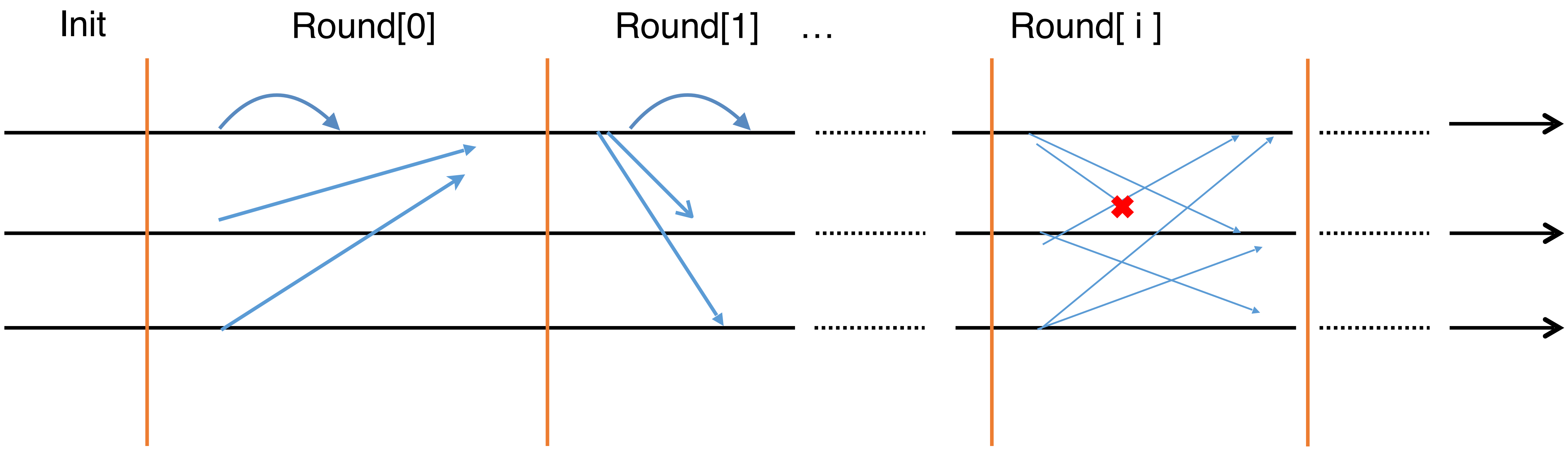
Programming model

1. Separates the ***network properties*** from the <u>algorithmic computation</u>

2. Faults and asynchrony simulated by an adversarial environment

3. We want a **simple programming abstraction** that offers a synchronous image of the system and compiles into **executable asynchronous code**.

Communication close rounds[Erlad,Francez’83]

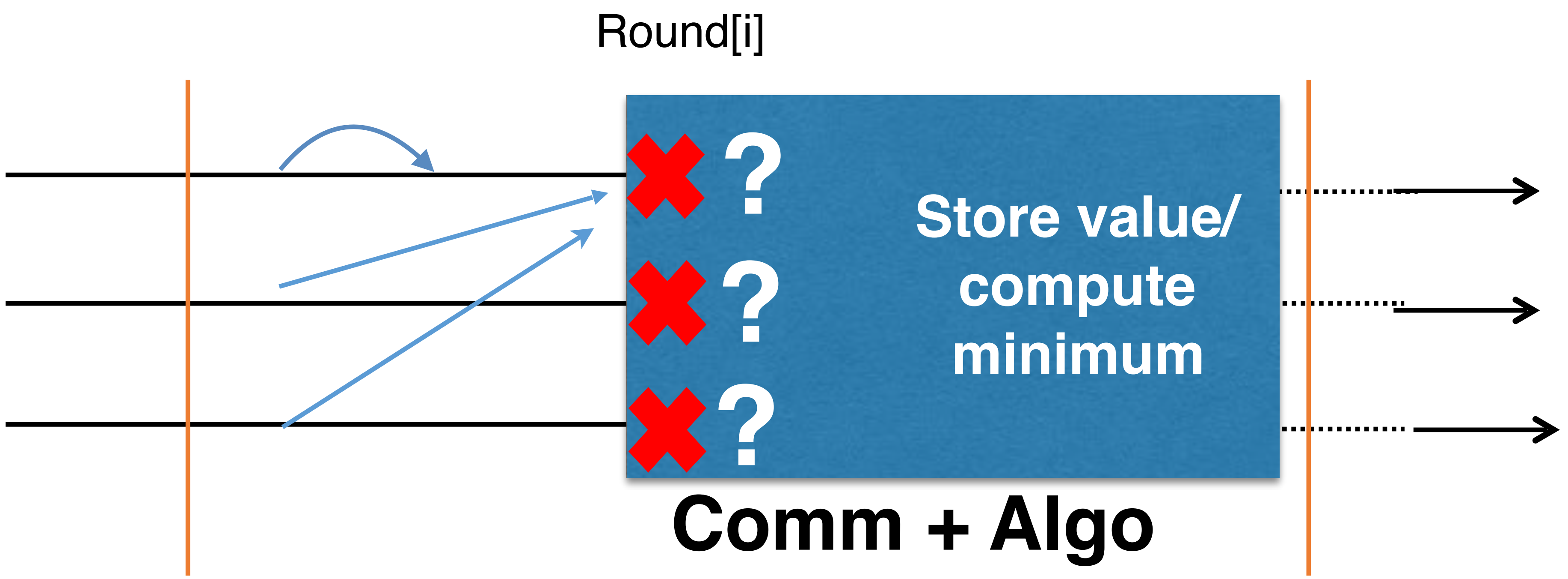


A **round** is a computation step that

- defines *the interaction between the processes* that participate in that step**,**
- processes synchronize at the boundary between rounds,
- it gives a logical unit of time.

Communication close = two rounds do not communicate with each other, i.e., messages are scoped in the round.

Communication close rounds[Erlad,Francez'83]

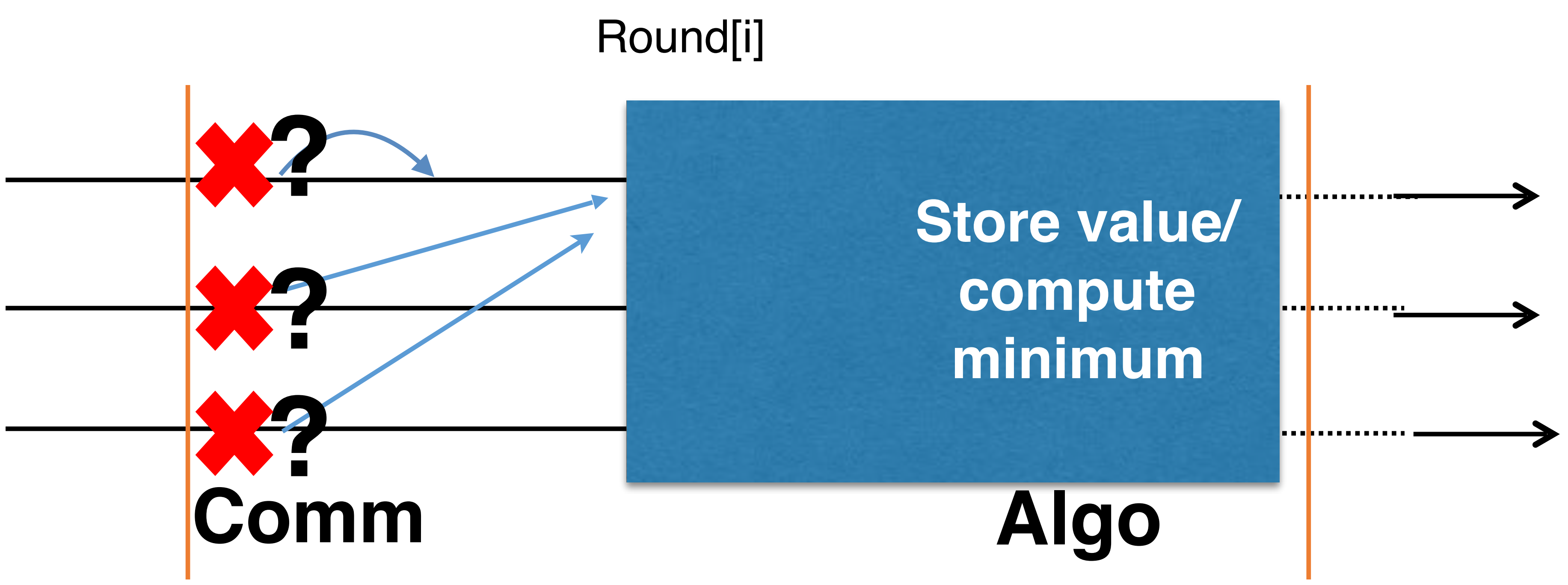


A **round** is a computation step that

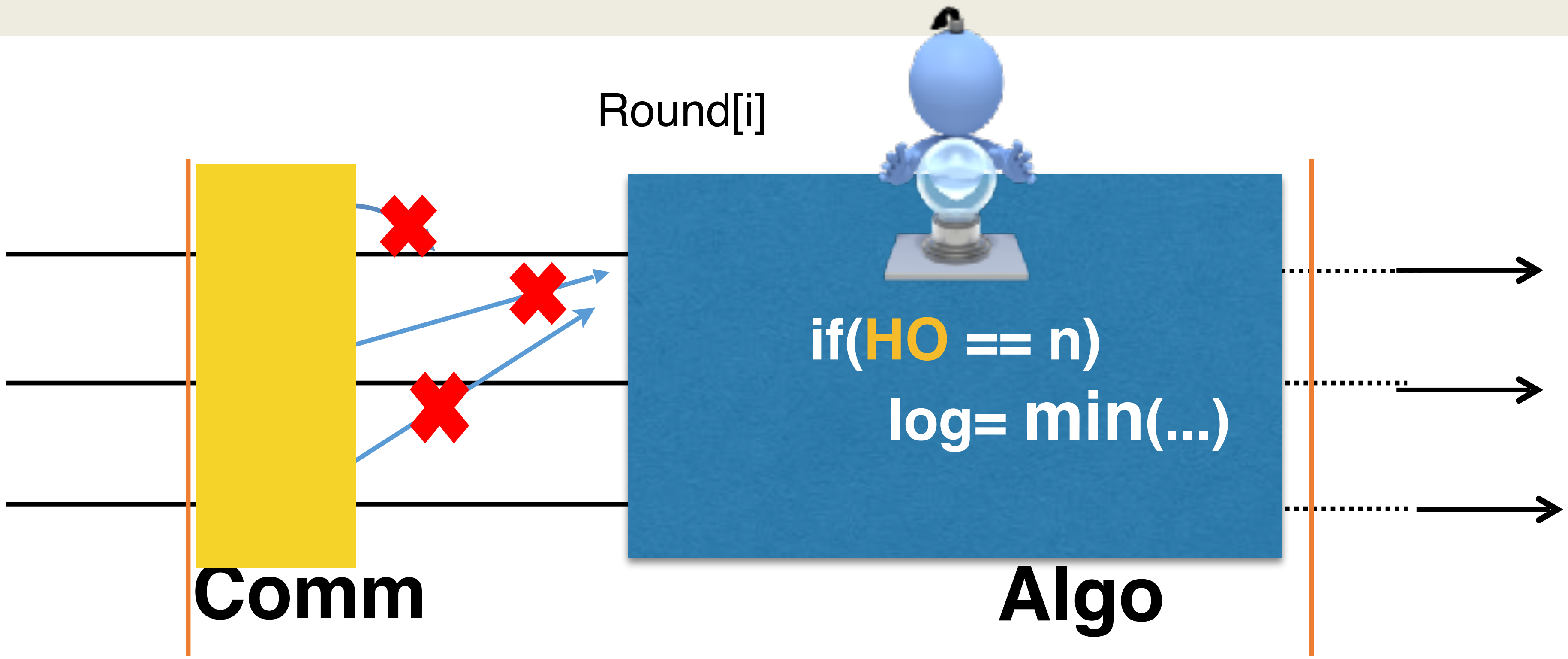
- defines *the interaction between the processes* that participate in that step**,**
- processes synchronize at the boundary between rounds,
- it gives a logical unit of time.

Communication close = two rounds do not communicate with each other, i.e., messages are scoped in the round.

HO-model [Charrone-Bost, Schiper'09]



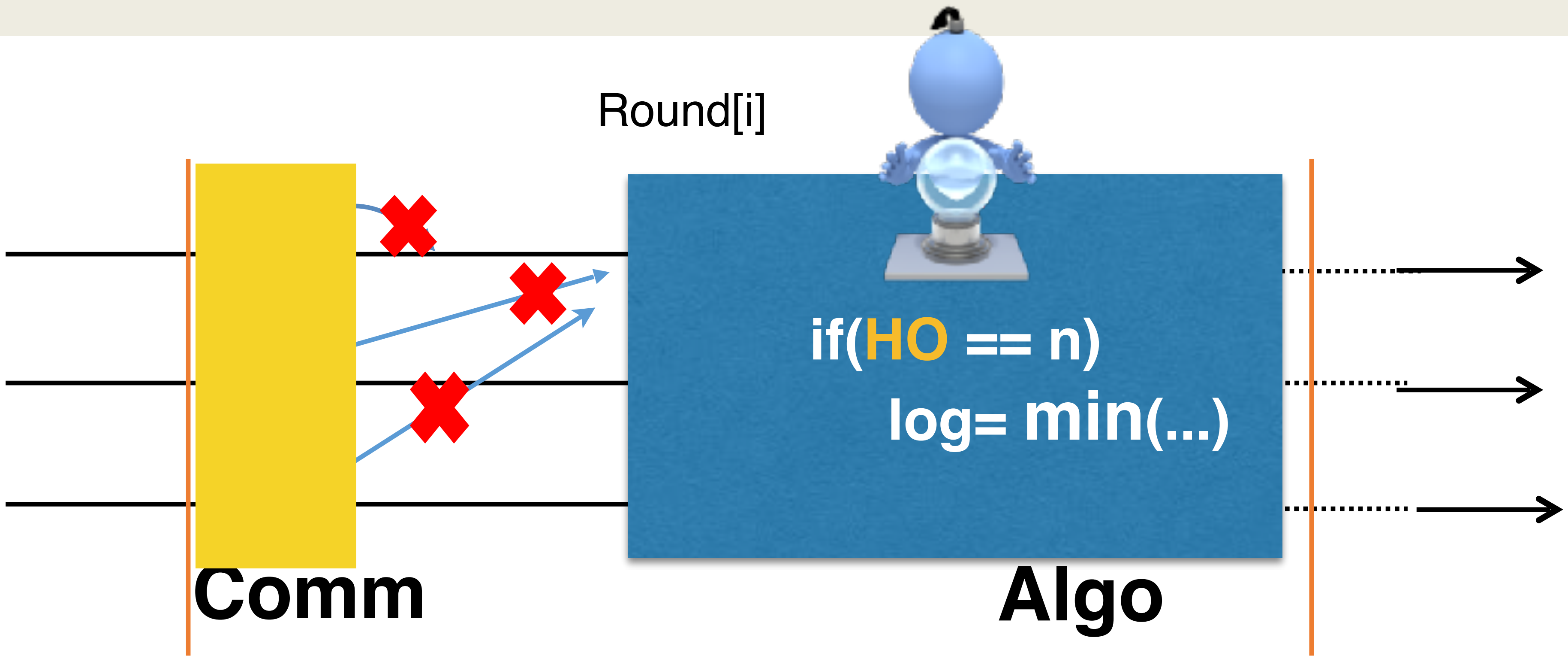
HO-model



Each process p has a variable, ***HO(p)=*** the set of processes p hears-from
HO(p) is non-deterministically chosen by an adversarial environment.

1. Determines the delivered messages

HO-model

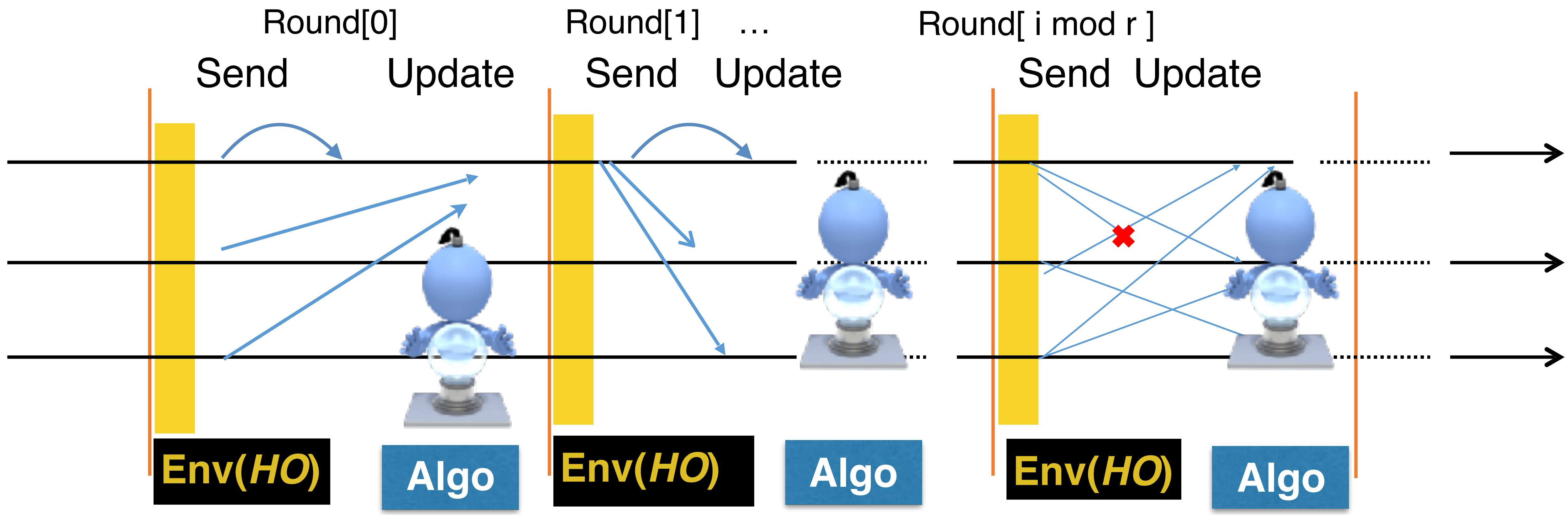


Each process p has a variable, ***HO(p)=*** the set of processes p hears-from
HO(p) is non-deterministically chosen by an adversarial environment.

1. Determines the delivered messages
2. The network assumptions are stated over the HO variables

$\square$ (∀p. p \in HO(leader) ∧ leader \in HO(p))

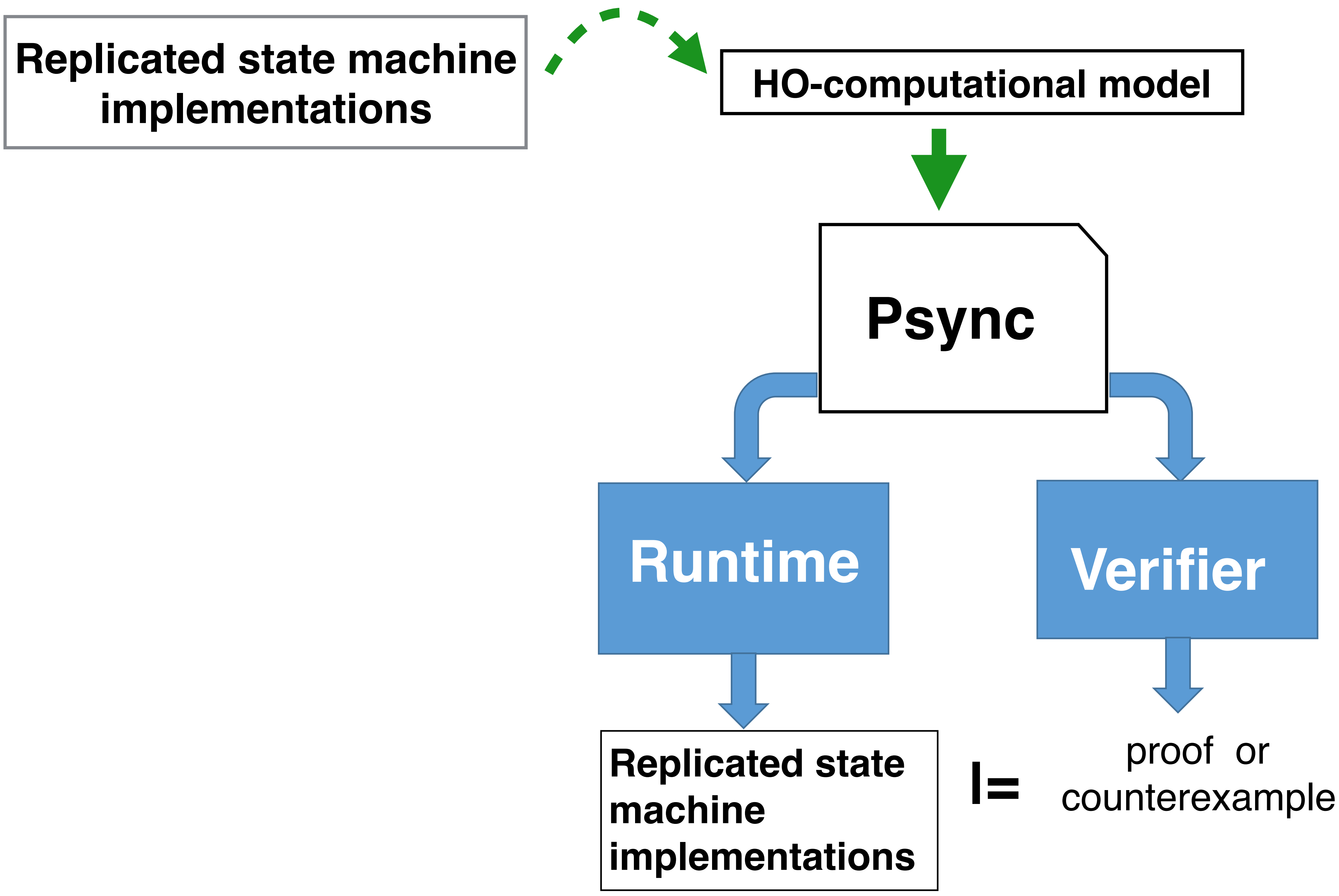
HO-model



$\Box\Diamond$(∀p. p \in HO(Leader) ∧ Leader \in HO(p))

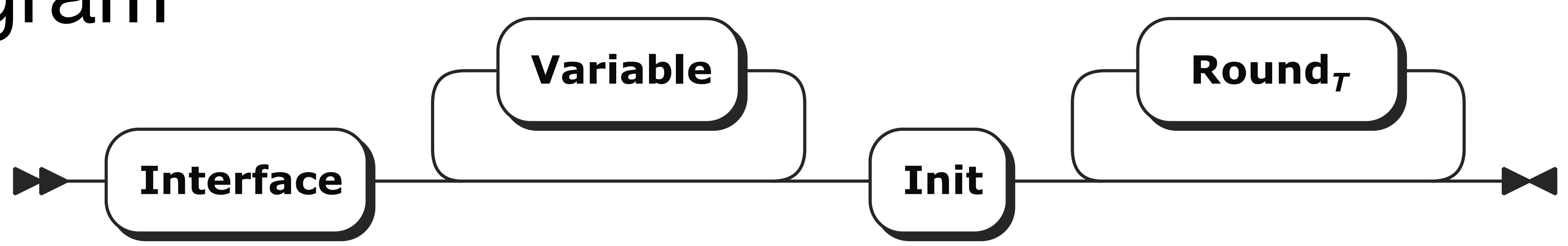
$\Box$ (∀p. p \in HO(leader) ∧ leader \in HO(p))

Approach


Replicated state machine implementations
HO-computational model
Psync
Runtime
Verifier
Replicated state machine implementations
|=
proof or counterexample

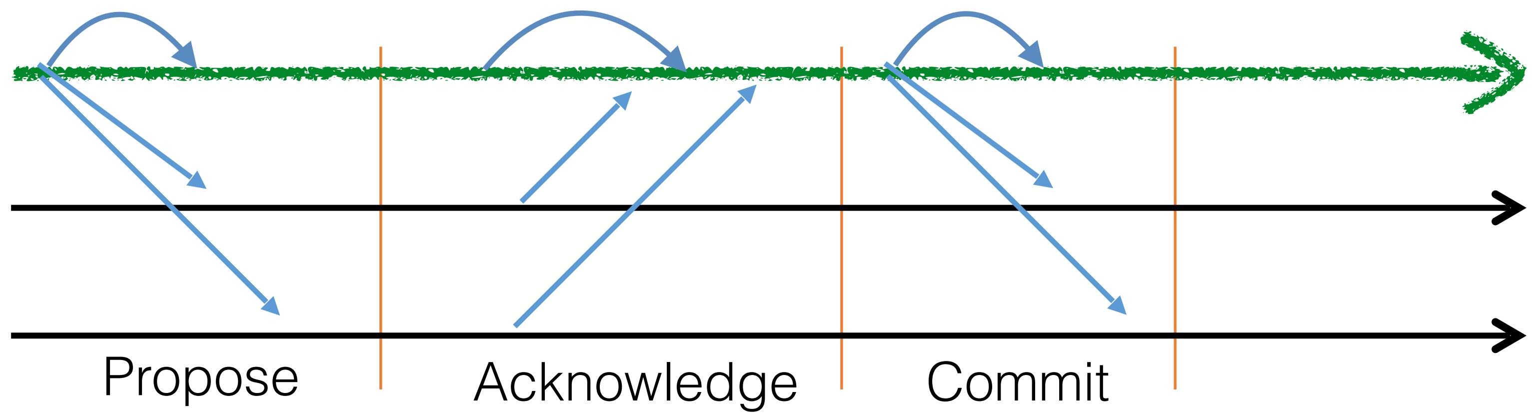
PSync Program Structure [popl'16]

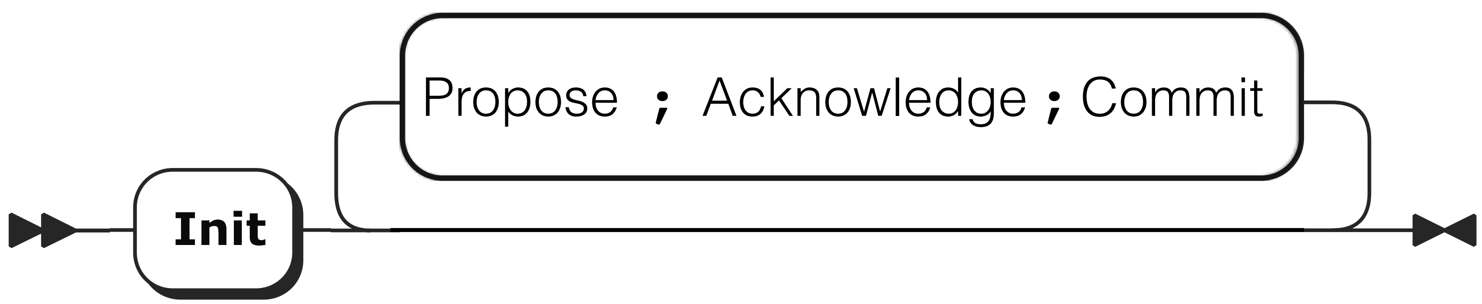
- Program



- Round$_T$

Example



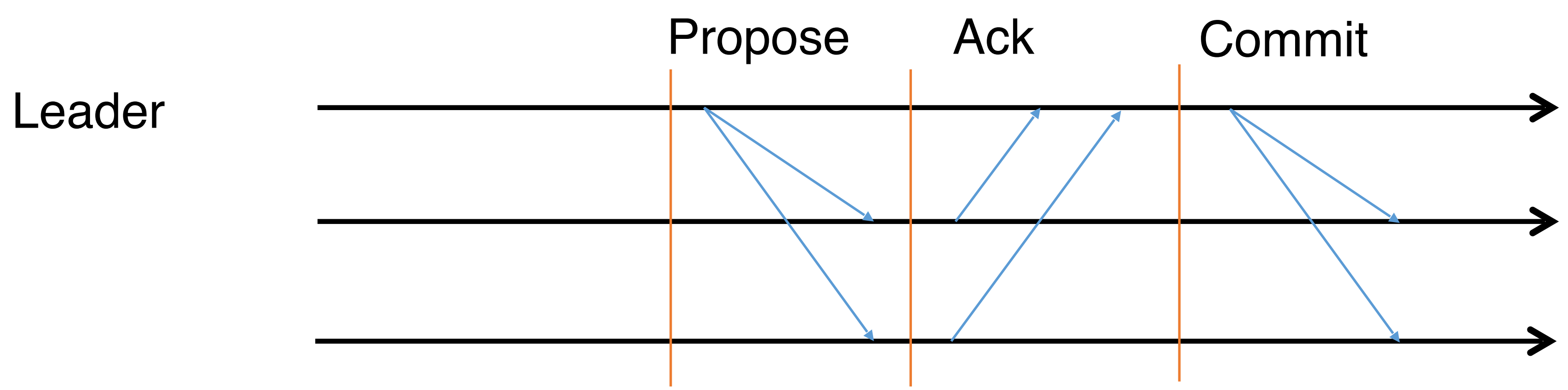


Communication Predicates:

Zab, Viewstamped $\Box(\forall p.\ p \in HO(\mathrm{Leader}) \land \mathrm{Leader} \in HO(p))$

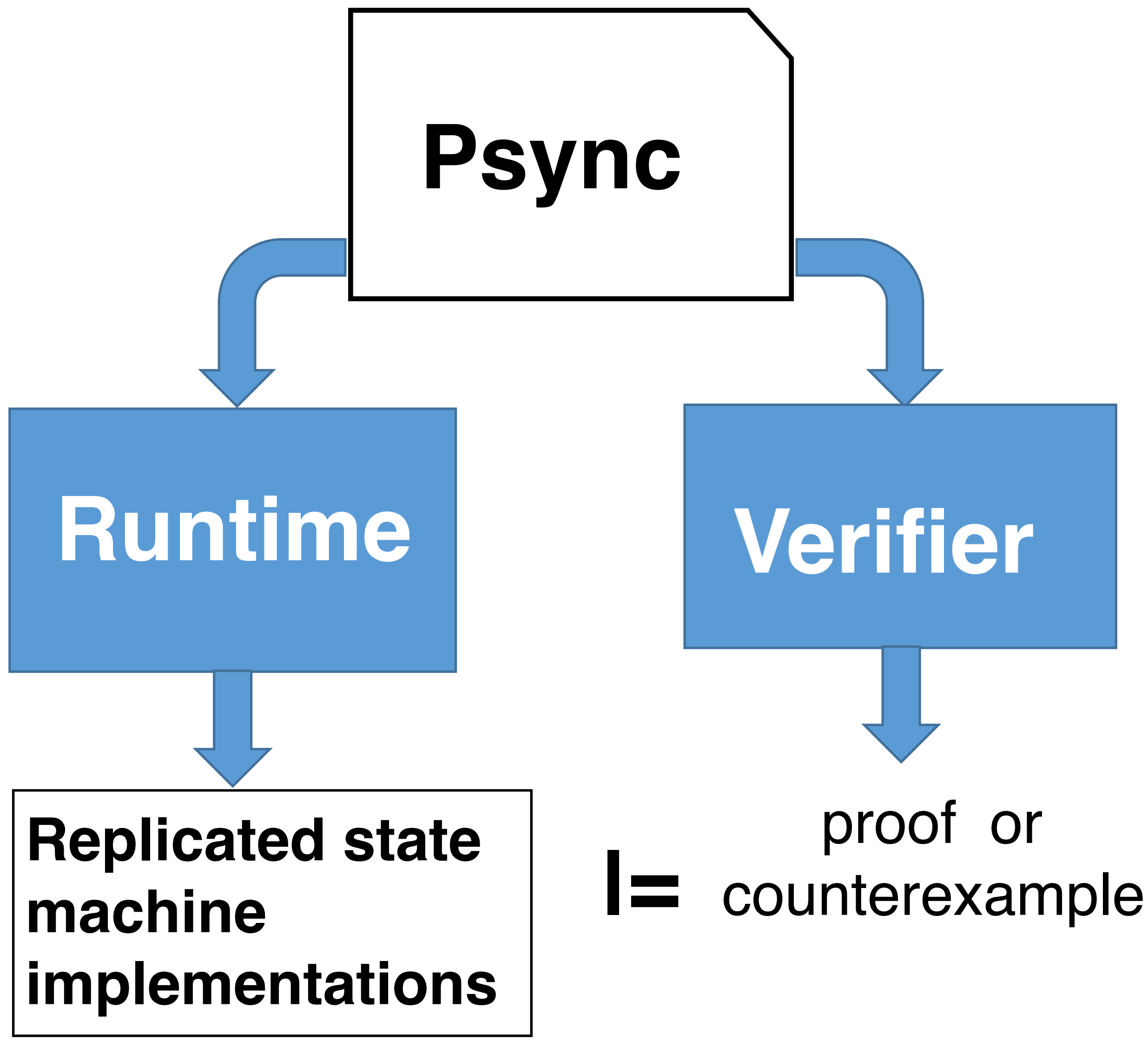
Paxos $\Box\Diamond(\forall p.\ p \in HO(\mathrm{Leader}) \land \mathrm{Leader} \in HO(p))$

Example: Last Voting Algorithm

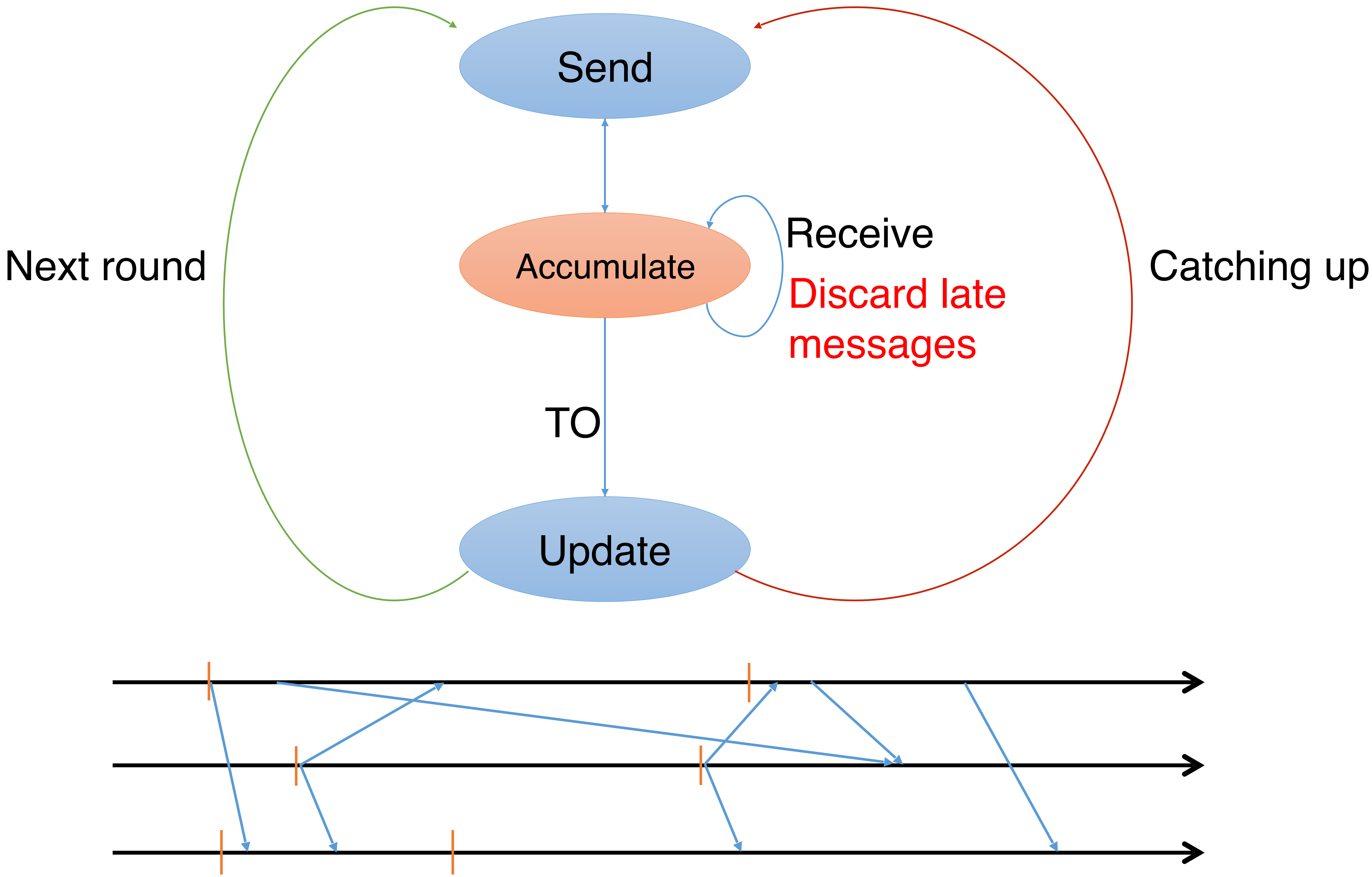



```
new Round[(Int,Time)]{
  def send(): Map[ProcessID, (Int,Time)] = Map( leader -> (x, ts) )
  def update(mailbox: Map[ProcessID, (Int,Time)]) {
    if (id == leader && mailbox.size > n/2) {
      vote = mailbox.maxBy(_._2._2)._2._1 // value with maximal ts
      commit = true
    }
} }
```

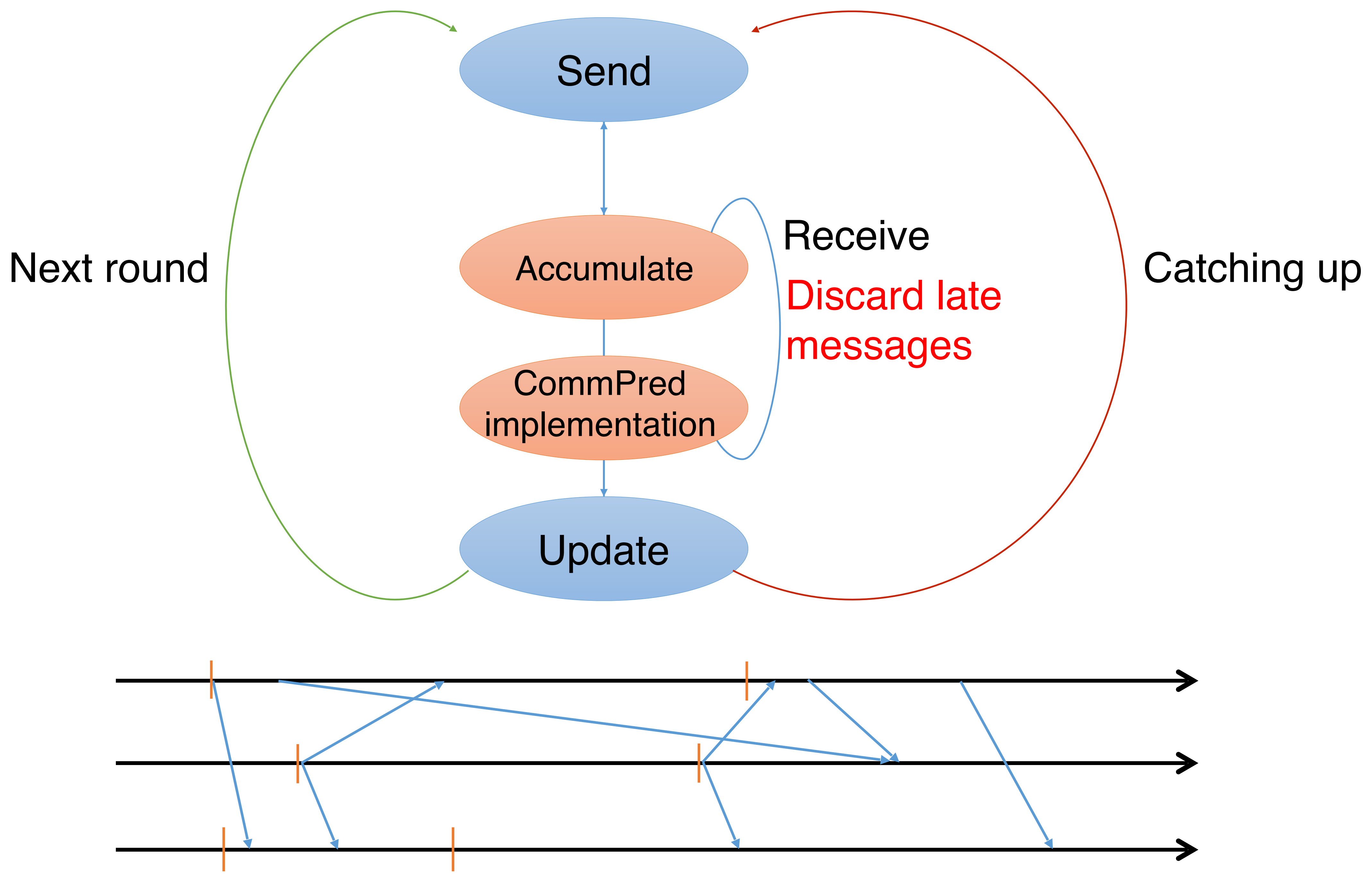
Outline


Psync
Runtime
Verifier
Replicated state machine implementations
|=
proof or
counterexample

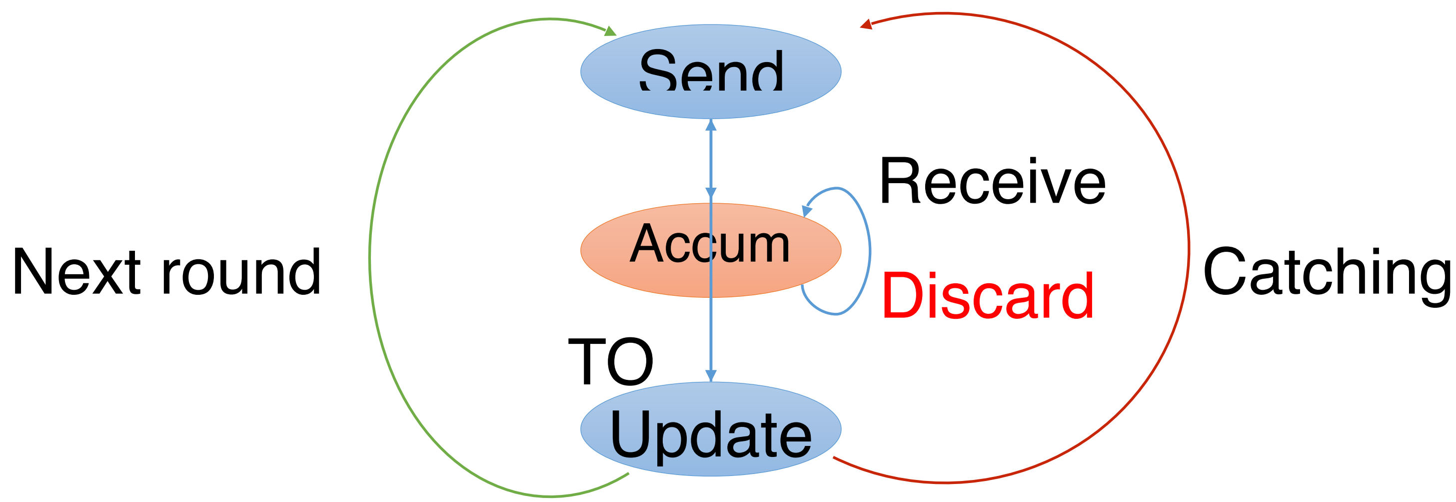
Runtime: Round switch

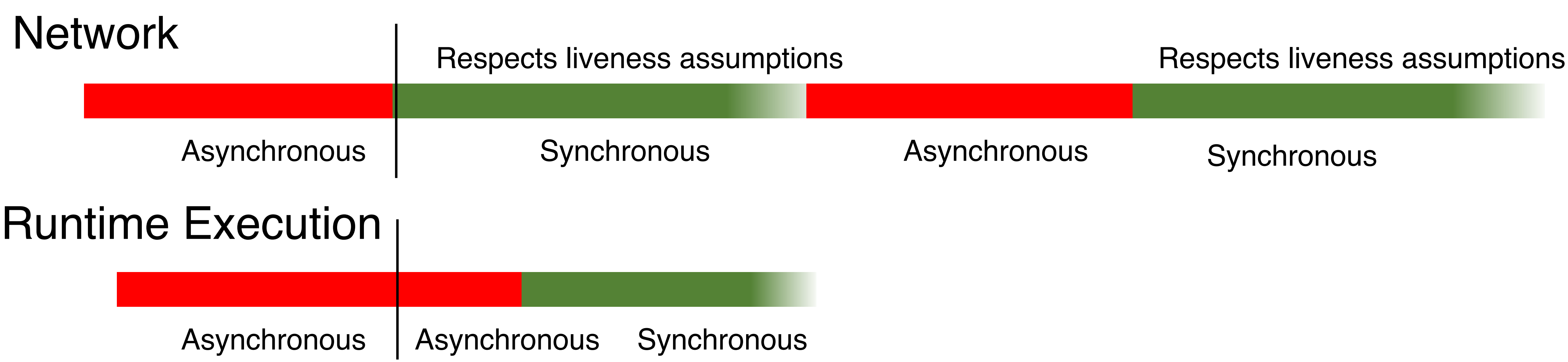


Runtime: Round switch

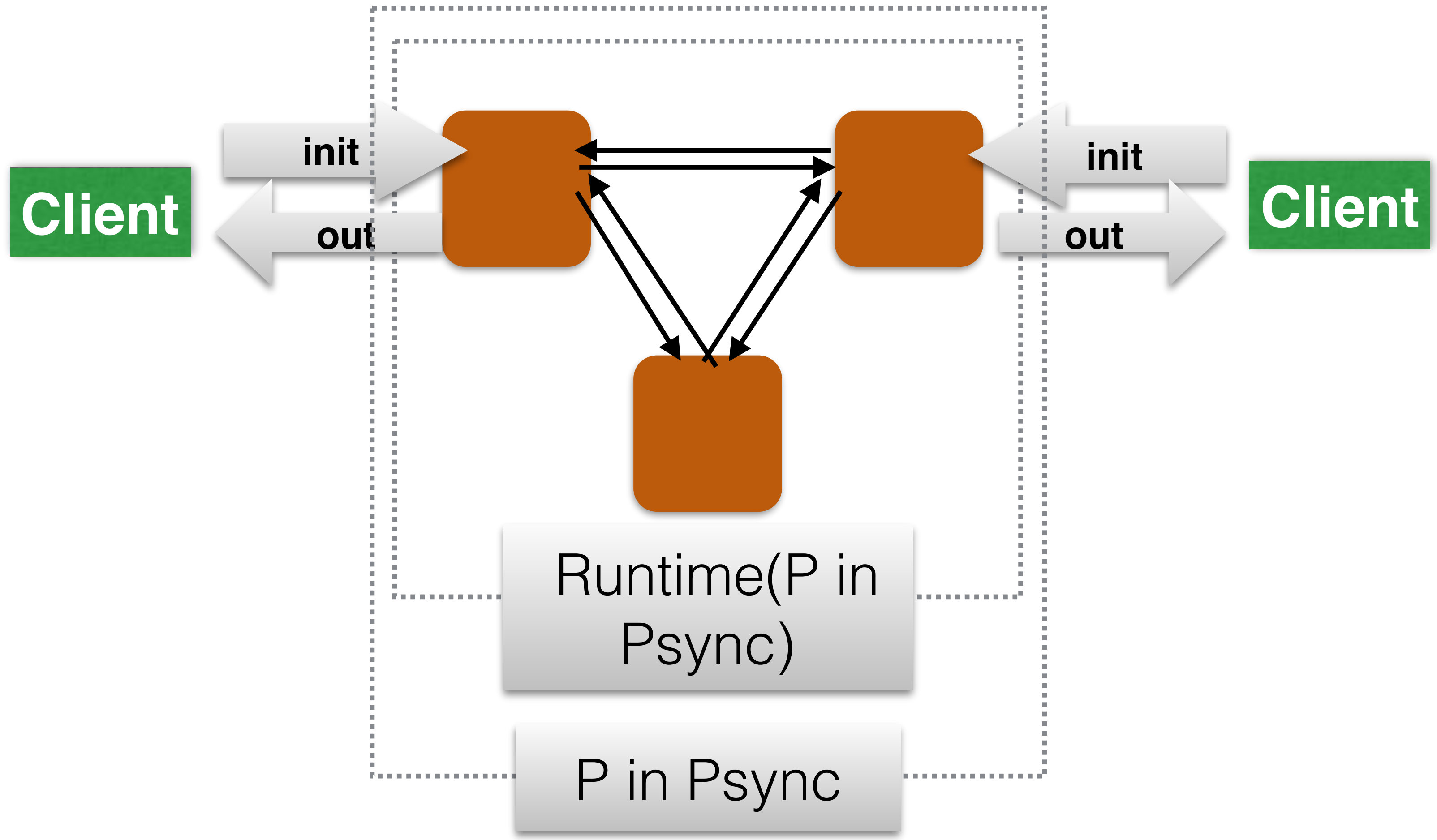


Partial synchrony





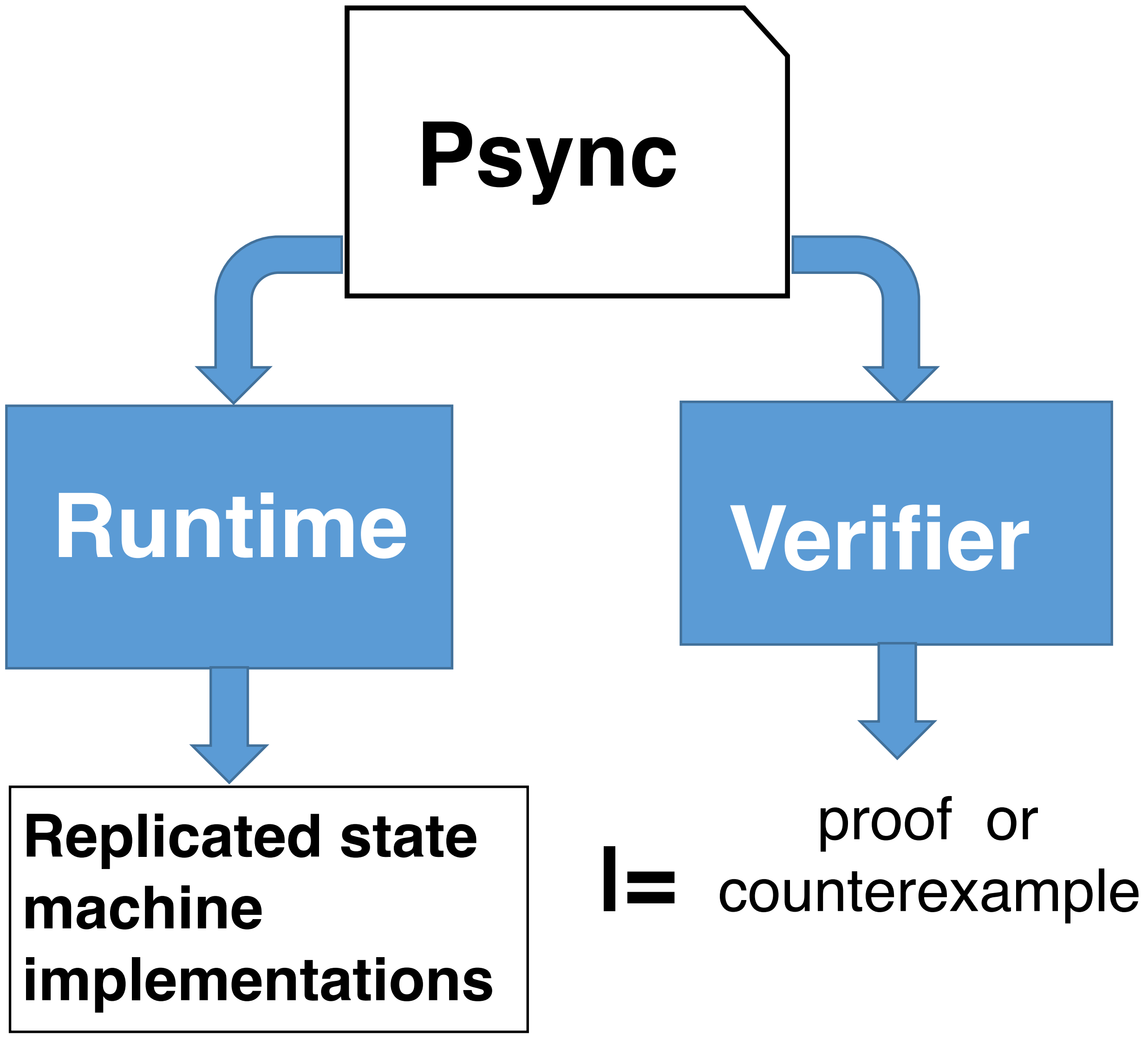
Runtime Correctness



For any Psync program P, the runtime semantics of P observationally refines the HO semantics of P, if the client is commutative.

$$\text{Clients} \parallel \text{Runtime}(P) \subseteq \text{Clients} \parallel \text{HO}(P)$$

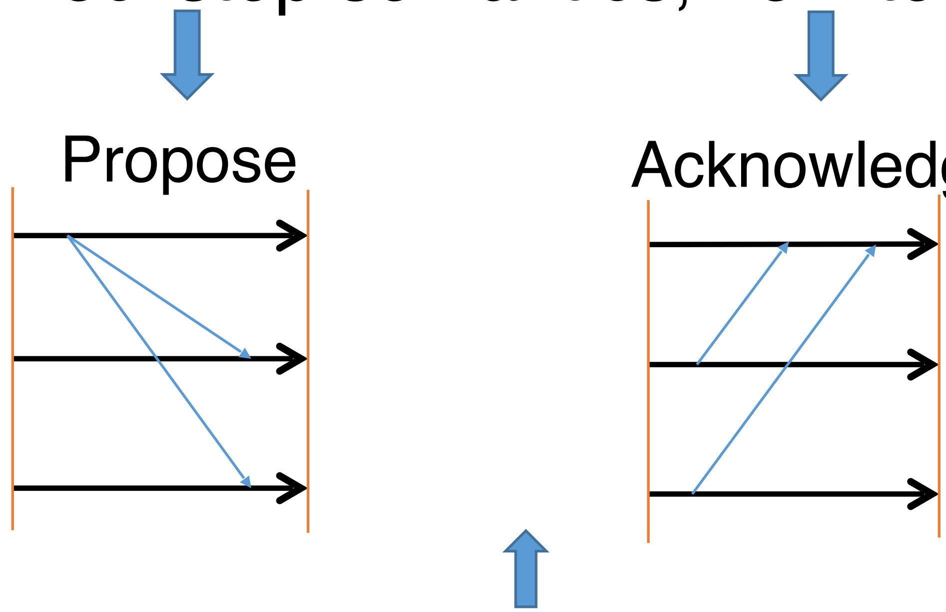
Outline



Psync
Runtime
Verifier
Replicated state machine implementations
|=
proof or
counterexample

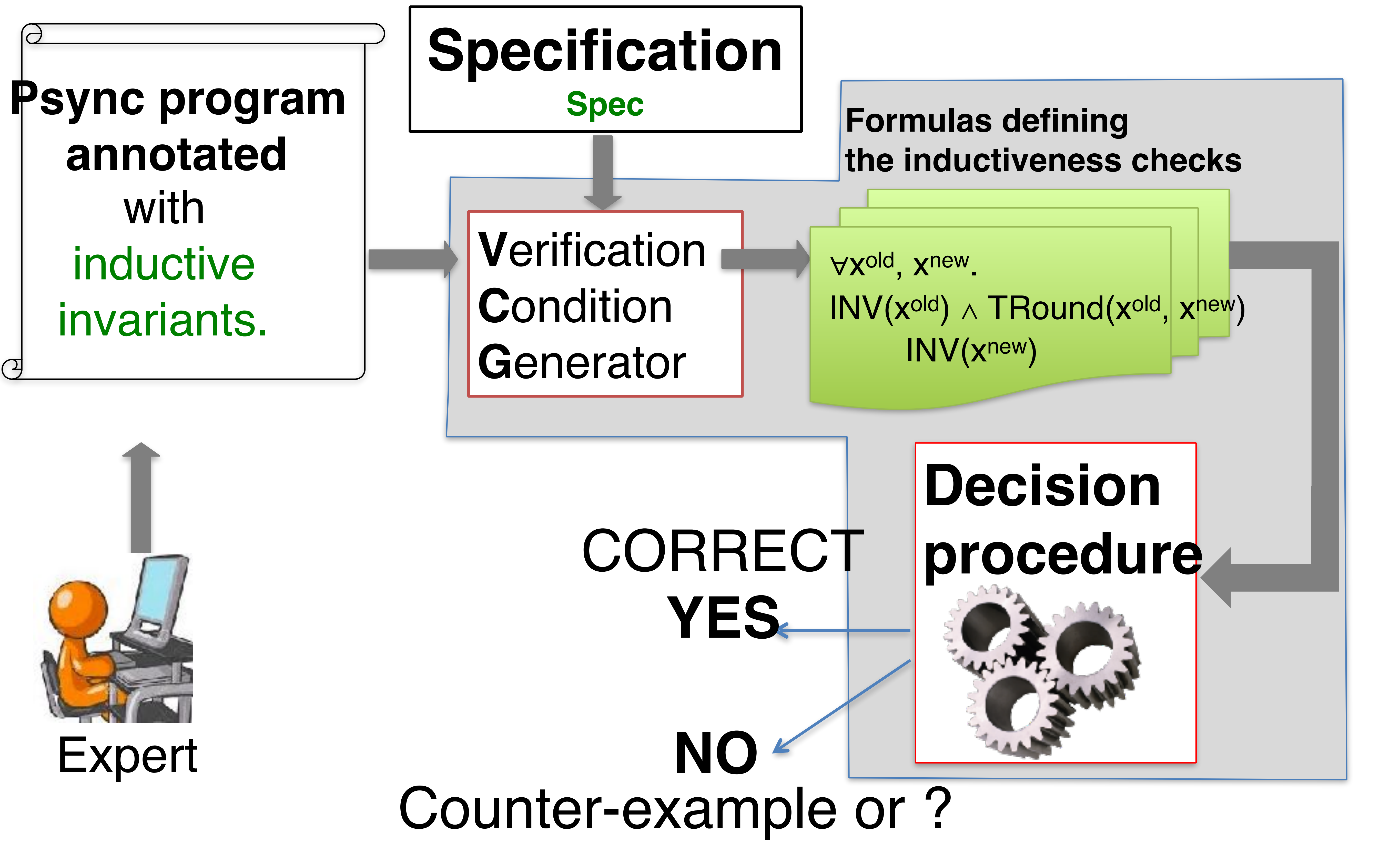
Psync: Benefits for Verification

Reason about rounds in isolation.
Lockstep semantics, no interleaving.

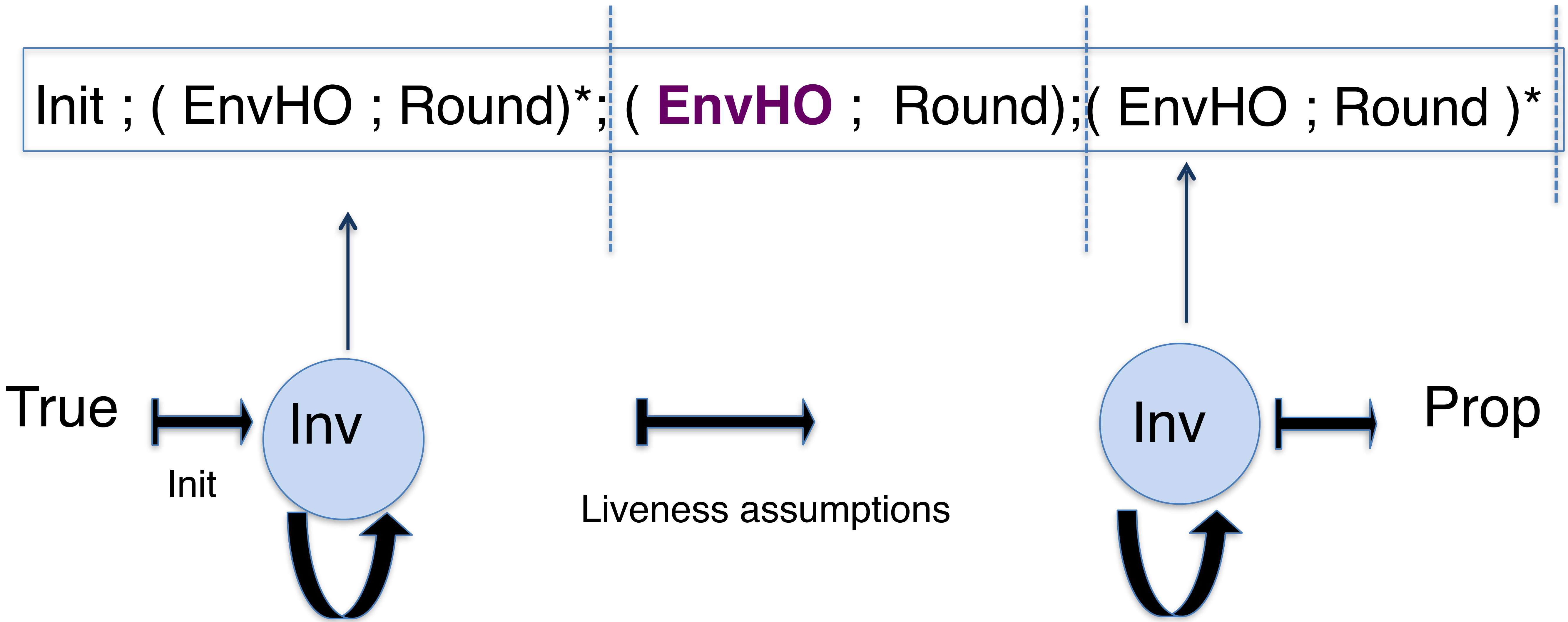


Simple invariants that reason at the round boundaries,
no messages are in flight, only the local states matter.

Psync: semi-automated verification



Hoare-style Verification [vmcai'14]



Safety: Inductive invariant checking
Liveness: Variant functions

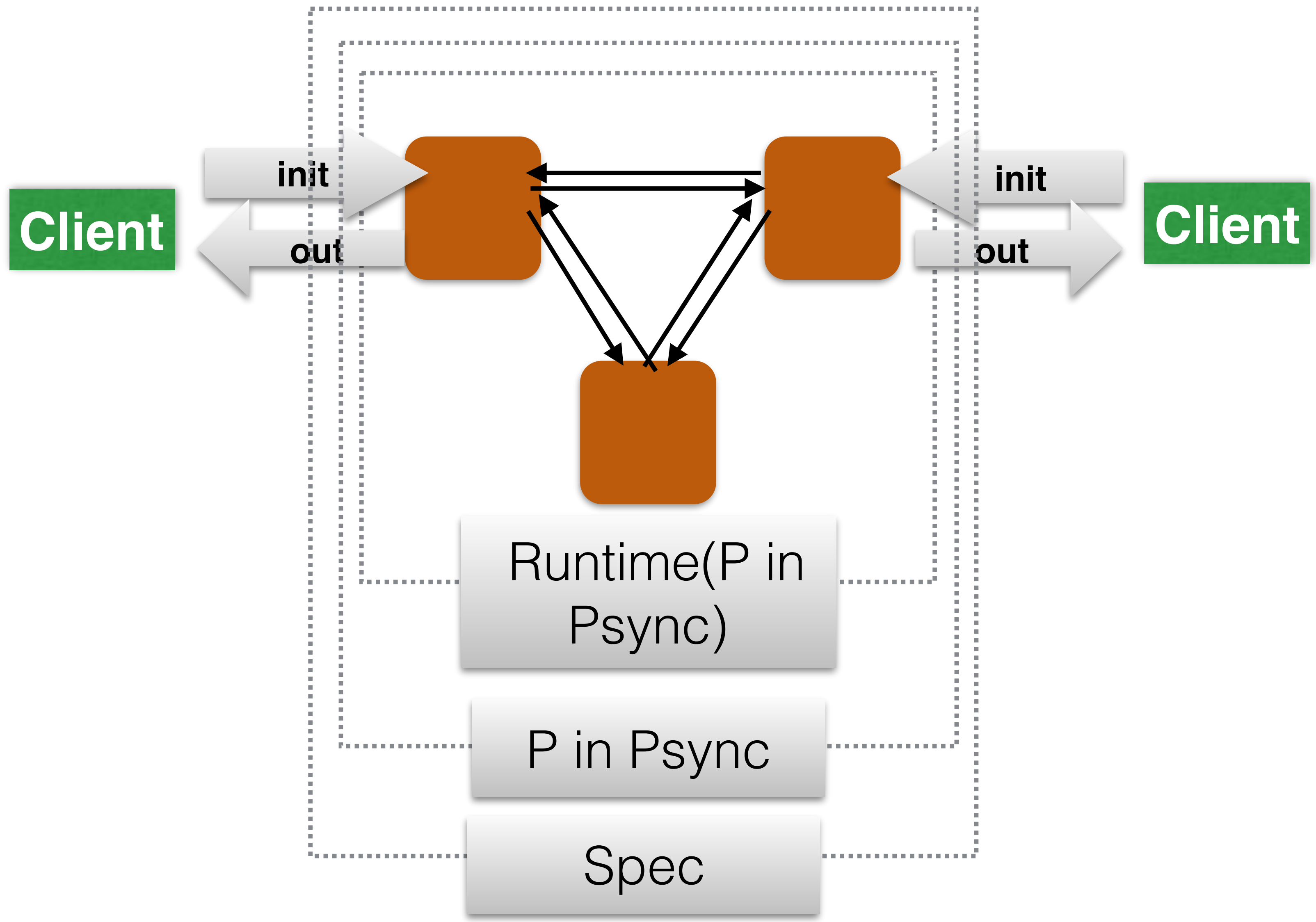
Invariant for Agreement

$$
\begin{aligned}
& \forall i. \neg decided(i) \wedge \neg ready(i) \\
\vee \quad & \exists\, v, t, A.\; A = \{\, i.\; ts(i) > t \} \wedge |A| > n/2 \\
& \quad \wedge \quad \forall i. i \in A \Rightarrow x(i) = v \\
& \quad \wedge \quad \forall i. decided(i) \Rightarrow x(i) = v \\
& \quad \wedge \quad \forall i. commit(i) \vee ready(i) \Rightarrow vote(i) = v \\
& \quad \wedge \quad t \leq \Phi \\
& \quad \wedge \quad \forall i.\; ts(i) = \Phi \Rightarrow commit(coord)
\end{aligned}
$$

Specification logic

- is able to express:
  - -- properties of sets of processes in the network
  - -- cardinality constraints
  - -- properties of the data values stored by each process
  - -- $\exists\forall$ quantifier alternation
- captures the transition relation of algorithms in the HO-model
- has a semi-decision procedure for checking entailment
- has a decidable satisfiability problem for a fragment Cl_{dec}

Psync: verification correctness



Given a specification S closed under indistinguishability, if a Psync program P satisfies S then the asynchronous semantics of P refines S.

$$\text{Clients} \parallel \text{Runtime}(P) \subseteq \text{Clients} \parallel \text{Spec}(P)$$

Do Algorithms use Rounds ?

Algorithm	LOC	Use rounds	Asynchronous
One third rule	*52*	✓	✓
Last Voting (Paxos)	*89*	✓	✓
Flood min consensus	*24*	✓	✗
Ben-Or randomized consensus	*56*	✓	✓
K-set agreement	*42*	✓	✓
K-set agreement early stopping	*33*	✓	✗
Lattice agreement	*34*	✗	✓
	54	✓	✓
Two phases commit	*53*	✓	✗
Eager reliable broadcast	*36*	✗	✗
Chandra Toueg	*121*	✓	✓
Generalized Paxos	*152*	✓	✓
ViewStampted	*91*	✗	✓

Code Size (Easy to Implement)

Paxos in	LOC	Executable	Verification
PSync	*89*	✓	*Semi-*
DistAlgo	*43*	✓	✗
Distal	*157*	✓	✗
Overlog	*107*	✓	✗
TLA+	*53*	✗	*Interactive*
IO Automata	*142*	✗	*Interactive*
EventML	*1729 N*	✓	*Interactive*
Verdi (Raft)	*520*	✓	*Interactive*

Performance and Verification

Implementation	Year	Language	Throughput (x 1000 req./s)
Last Voting in PSync	*2015*	*Scala*	*170*
Egalitarian Paxos	*2013*	*Go*	*450*
Paxos in Distal	*2013*	*Scala*	*150*
JPaxos / SPaxos	*2012*	*Java*	*75 / 300*
Paxos for system builder	*2008*	*C*	*40*

Verification of	# Invariants (LOC)	# VCs	Solving time in
One third rule	*4 (23)*	*27*	*5*
Last Voting	*8 (35)*	*45*	*16*

Conclusions

PSync uses a simple programming abstraction:
  Communication-closed rounds
  Separates the algorithm from the network requirements
  Asynchrony and faults are modelled by an adversary that drops messages

Runtime:
  Asynchronous semantics refines the lockstep semantics
  Preserves strong consistency
  Can be implemented efficiently

Automated verification becomes possible