Getting Veil

- Website: **veil.dev**
- Online environment: **try.veil.dev**
- Repository: github.com/verse-lab/veil
  Branch **leanlang-demo**
  ```
  lake exe cache get; lake build
  ```

Multi-Modal Verification of Distributed Systems in Lean

Ilya Sergey

ilyasergey.net

A Framework for Automated and Interactive Verification of Distributed Protocols

Distributed Protocols

Define how multiple parties (nodes) collaborate with each other to achieve a common goal 🤝

Operating Systems — R. Stockton Gaines, Editor

Time, Clocks, and the Ordering of Events in a Distributed System

Leslie Lamport
Massachusetts Computer Associates, Inc.

Operating Systems — R. Stockton Gaines, Editor

An Optimal Algorithm for Mutual Exclusion in Computer Networks

Glenn Ricart
National Institutes of Health

Ashok K. Agrawala
University of Maryland

Minister

Anne, are you prepared to commit to this relationship?

Henry

Anne

Two-Phase Commit depiction from Leslie Lamport's video course on TLA+

Distributed Protocols

Some nodes might *fail* 💥

Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems

Brian M. Oki
Barbara H. Liskov

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge, MA 02139

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

There Is More Consensus in Egalitarian Parliaments

Iulian Moraru, David G. Andersen, Michael Kaminsky
Carnegie Mellon University and Intel Labs

In Search of an Understandable Consensus Algorithm

Diego Ongaro and John Ousterhout, *Stanford U*

https://www.usenix.org/conference/atc14/technical-sessions/pres

Just Say NO to Paxos Overhead: Replacing Consensus with Network Ordering

Jialin Li Ellis Michael Naveen Kr. Sharma Adriana Szekeres Dan R. K. Ports
University of Washington
{lijl, emichael, naveenks, aaasz, drkp}@cs.washington.edu

Distributed Protocols

Some nodes might behave *maliciously* 😈

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

Practical Byzantine Fault Tolerance

Miguel Castro and Barbara Liskov

The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus

DAVID MAZIÈRES, Stellar Development Foundation

HotStuff: BFT Consensus with Linearity and Responsiveness

Maofan Yin, Cornell University, VMware Research
Dahlia Malkhi, VMware Research
Michael K. Reiter, UNC-Chapel Hill
Guy Golan Gueta, VMware Research
Ittai Abraham, VMware Research

All You Need is DAG

Eleftherios Kokoris-Kogias
IST Austria and Novi Research

Alexander Spiegelman
Novi Research

Bullshark: DAG BFT Protocols Made Practical

Alexander Spiegelman
sasha.spiegelman@gmail.com

Neil Giridharan
giridhn@berkeley.edu

Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback

Rati Gelashvili, *Novi Research*
Lefteris Kokoris-Kogias, *Novi Research & IST Austria*
Alberto Sonnino, *Novi Research*
Alexander Spiegelman, *Novi Research*
Zhuolun Xiang*, *University of Illinois at Urbana-Champaign*

A Secure Sharding Protocol For Open Blockchains

Loi Luu, National University of Singapore, loiluu@comp.nus.edu.sg
Viswesh Narayanan, National University of Singapore, visweshn@comp.nus.edu.sg
Chaodong Zheng, National University of Singapore, chaodong.zheng@comp.nus.edu.sg
Kunal Baweja, National University of Singapore, bawejaku@comp.nus.edu.sg
Seth Gilbert, National University of Singapore, seth.gilbert@comp.nus.edu.sg
Prateek Saxena, National University of Singapore, prateeks@comp.nus.edu.sg

Frameworks for Verifying Distributed Protocols

Proving the Correctness of Disk Paxos in Isabelle/HOL

2005

IronFleet: Proving Practical Distributed Systems Correct

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch,
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill

Ivy: Safety Verification by Interactive Generalization

Oded Padon
Tel Aviv University, Israel

Kenneth L. McMillan
Microsoft Research, USA

Aurojit Panda
UC Berkeley, USA

Verdi: A Framework for Implementing and Formally Verifying Distributed Systems

Programming and Proving with Distributed Protocols

ILYA SERGEY, University College London, UK
JAMES R. WILCOX, University of Washington, USA
ZACHARY TATLOCK, University of Washington, USA

Velisarios: Byzantine Fault-Tolerant Protocols Powered by Coq *

Aneris: A Mechanised Logic for Modular Reasoning about Distributed Systems

Grove: a Separation-Logic Library for Verifying Distributed Systems

, Amin Timany*, Marit Edna Ohlenbusch,

LiDO: Linearizable Byzantine Distributed Objects with Refinement-Based Liveness Proofs

LONGFEI QIU, Yale University, USA
YOONSEUNG KIM, Yale University, USA
JI-YONG SHIN, Northeastern University, USA
JIEUNG KIM, Inha University, South Korea
WOLF HONORÉ*, Yale University, USA
ZHONG SHAO, Yale University, USA

Compositional Verification of Composite Byzantine Protocols

Qiyuan Zhao
National University of Singapore
Singapore, Singapore
qiyuanz@comp.nus.edu.sg

George Pîrlea
National University of Singapore
Singapore, Singapore
gpirlea@comp.nus.edu.sg

Karolina Grzeszkiewicz
Yale-NUS College
Singapore, Singapore
karolina.grzeszkiewicz@u.yale-nus.edu.sg

Seth Gilbert
National University of Singapore
Singapore, Singapore
seth.gilbert@comp.nus.edu.sg

Ilya Sergey
National University of Singapore
Singapore, Singapore
ilya@nus.edu.sg

Frameworks for Verifying Distributed Protocols

Proving the Correctness of Disk Paxos in Isabelle/HOL

2005

IronFleet: Proving Practical Distributed Systems Correct

Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch,
Bryan Parno, Michael L. Roberts, Srinath Setty, Brian Zill

Verdi: A Framework for Formally Verifying Dis

Programming and Proving wi

ILYA SERGEY, University College London, UK
JAMES R. WILCOX, University of Washington,
ZACHARY TATLOCK, University of Washingto

Grove: a Separation-Logic L

LiDO: Linearizable Byzantine Dist
Refinement-Based Liveness Proof

LONGFEI QIU, Yale University, USA
YOONSEUNG KIM, Yale University, USA
JI-YONG SHIN, Northeastern University, USA
JIEUNG KIM, Inha University, South Korea
WOLF HONORÉ*, Yale University, USA
ZHONG SHAO, Yale University, USA

n by Interactive Generalization

enneth L. McMillan
crosoft Research, USA

Aurojit Panda
UC Berkeley, USA

ne Fault-Tolerant Protocols
ered by Coq *

nised Logic for Modular
t Distributed Systems

hin Timany*, Marit Edna Ohlenbusch,

Composite Byzantine Protocols

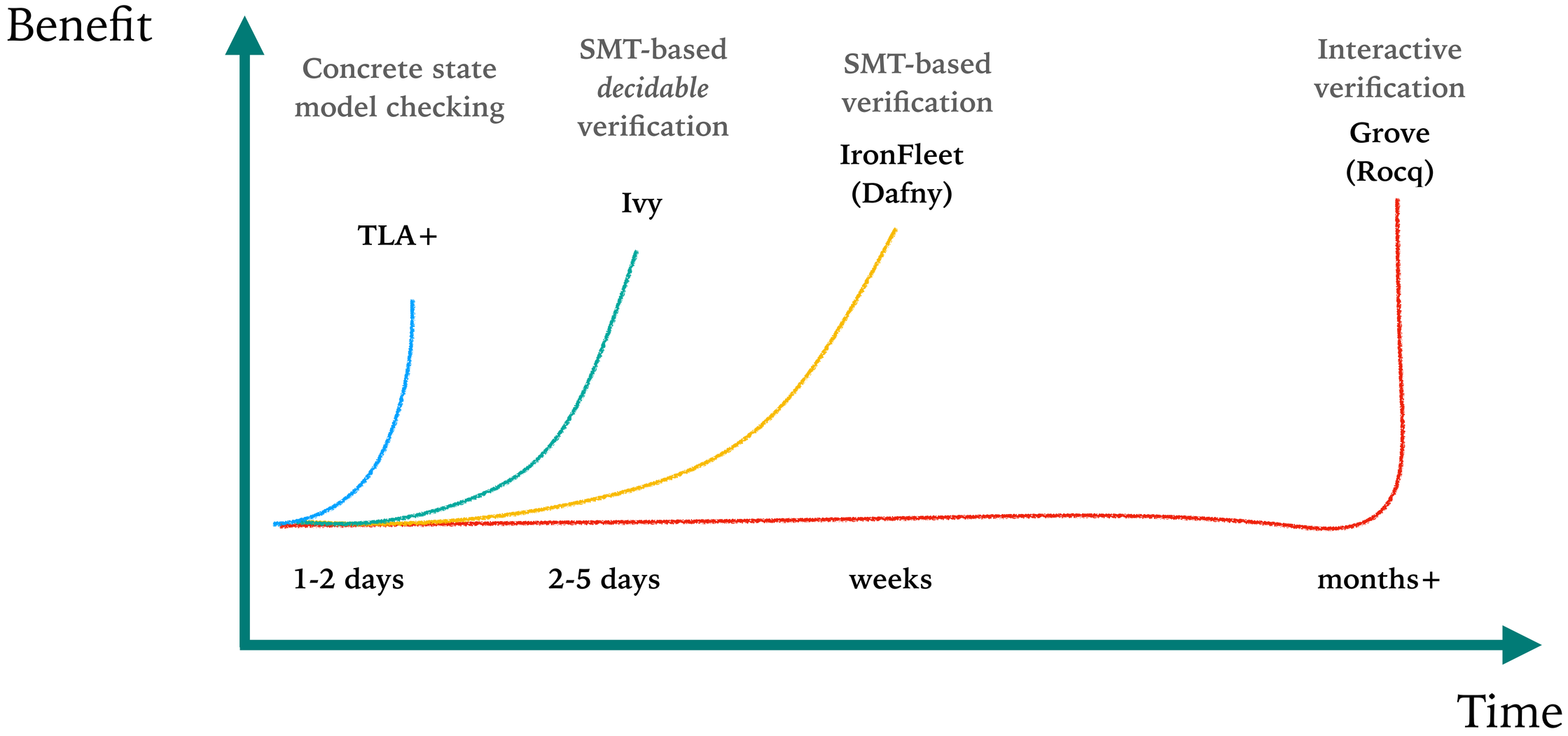
ge Pîrlea
National University of Singapore
Singapore, Singapore
gpirlea@comp.nus.edu.sg

Karolina Grzeszkiewicz
Yale-NUS College
Singapore, Singapore
karolina.grzeszkiewicz@u.yale-nus.edu.sg

Seth Gilbert
National University of Singapore
Singapore, Singapore
seth.gilbert@comp.nus.edu.sg

Ilya Sergey
National University of Singapore
Singapore, Singapore
ilya@nus.edu.sg

Approaches for Verifying Distributed Protocols



When Your Tool is Not Sufficient

You either:

- use a **combination** of tools, or

- add an **interactive escape hatch** to your automated tool

Pîrlea's Law

Every verifier expands until it contains
an ad hoc, bug-ridden, unusable
implementation of half of
an interactive theorem prover.

Those verifiers which cannot so expand
are replaced by ones which can.

Lessons from Building an Auto-Active Verifier in Lean

George Pîrlea
National University of Singapore
Singapore
gpirlea@comp.nus.edu.sg

Vladimir Gladshtein
National University of Singapore
Singapore
vgladsht@comp.nus.edu.sg

Qiyuan Zhao
National University of Singapore
Singapore
qiyuanz@comp.nus.edu.sg

Ilya Sergey
National University of Singapore
Singapore
ilya@nus.edu.sg

see also:
Greenspun's tenth rule

Veil

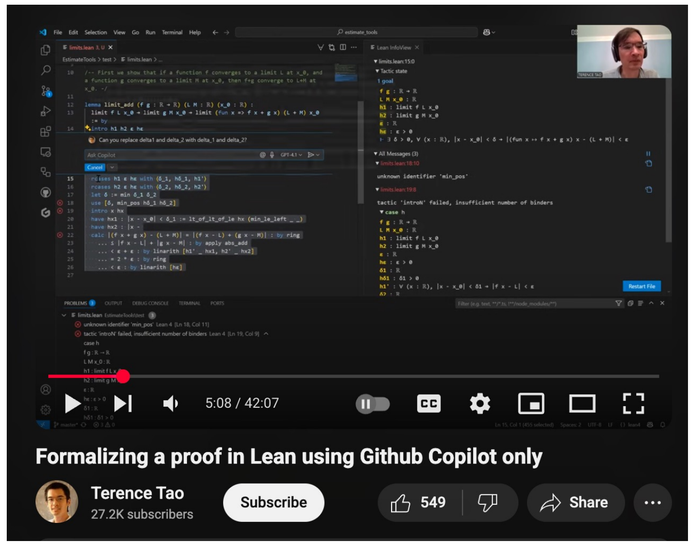
We just built the whole verifier in Lean!

What is Lean

- Lean is a **programming language** and **proof assistant**
- Lean and its tooling are **implemented in Lean**. Lean is very extensible.
- Batteries included: LSP, Parser, Macro System, Type Checker, Tactic Framework, Proof automation, Compiler, Build System, Documentation Authoring Tool.
- Lean has a **small trusted core**, proofs can be exported and independently checked.
- The Lean FRO is a nonprofit dedicated to developing Lean.

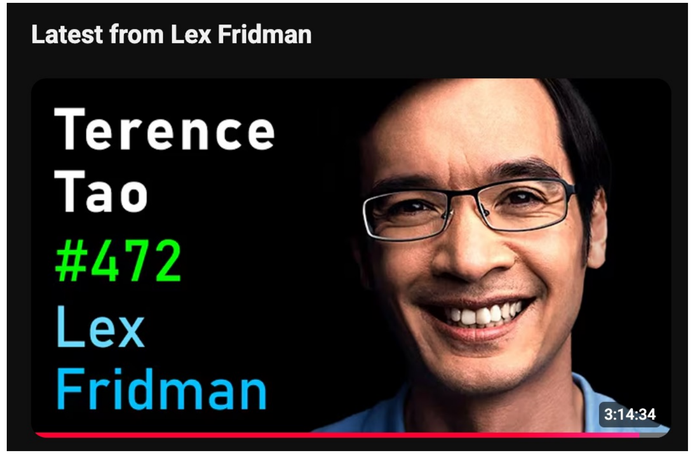
Lean is Taking Mathematics by Storm

*"**Lean enables large-scale collaboration** by allowing mathematicians to break down complex proofs into smaller, verifiable components. This formalization process ensures the correctness of proofs and facilitates contributions from a broader community. **With Lean, we are beginning to see how AI can accelerate the formalization of mathematics, opening up new possibilities for research.**" — Terence Tao*



Fermat's Last Theorem – Kevin Buzzard

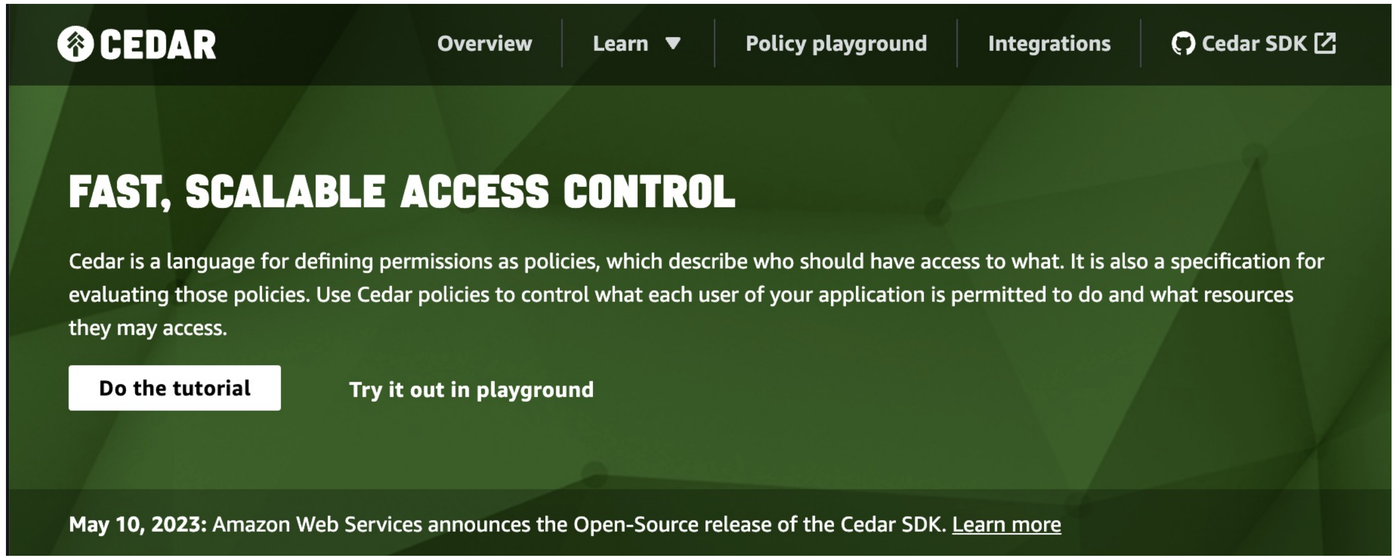
Carleson's Theorem – Floris van Doorn



slide credit: Leo de Moura

Cedar

https://www.cedarpolicy.com/

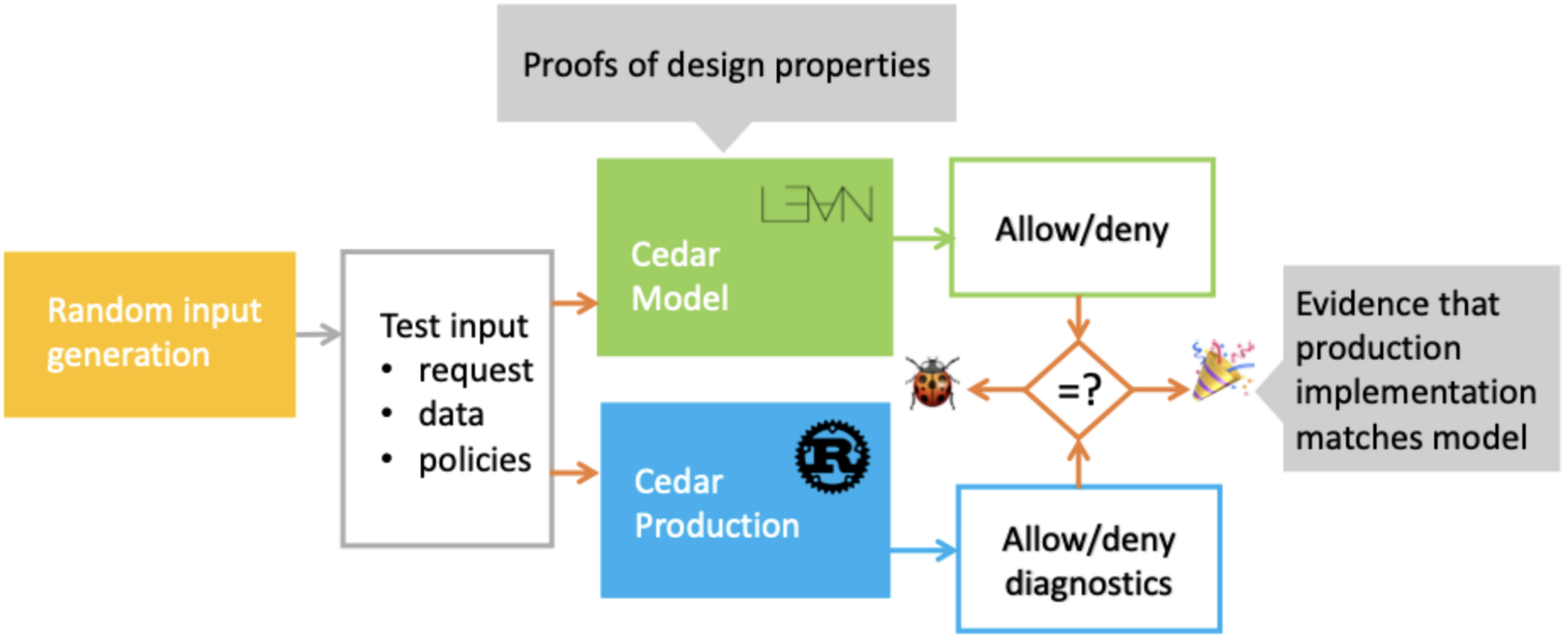


https://github.com/cedar-policy/cedar-spec

```
def isAuthorized (req : Request) (entities : Entities) (policies : Policies) : Response :=
  let forbids := satisfiedPolicies .forbid policies req entities
  let permits := satisfiedPolicies .permit policies req entities
  let erroringPolicies := errorPolicies policies req entities
  if forbids.isEmpty && !permits.isEmpty
  then { decision := .allow, determiningPolicies := permits, erroringPolicies }
  else { decision := .deny,  determiningPolicies := forbids, erroringPolicies }
```


slide credit: Leo de Moura

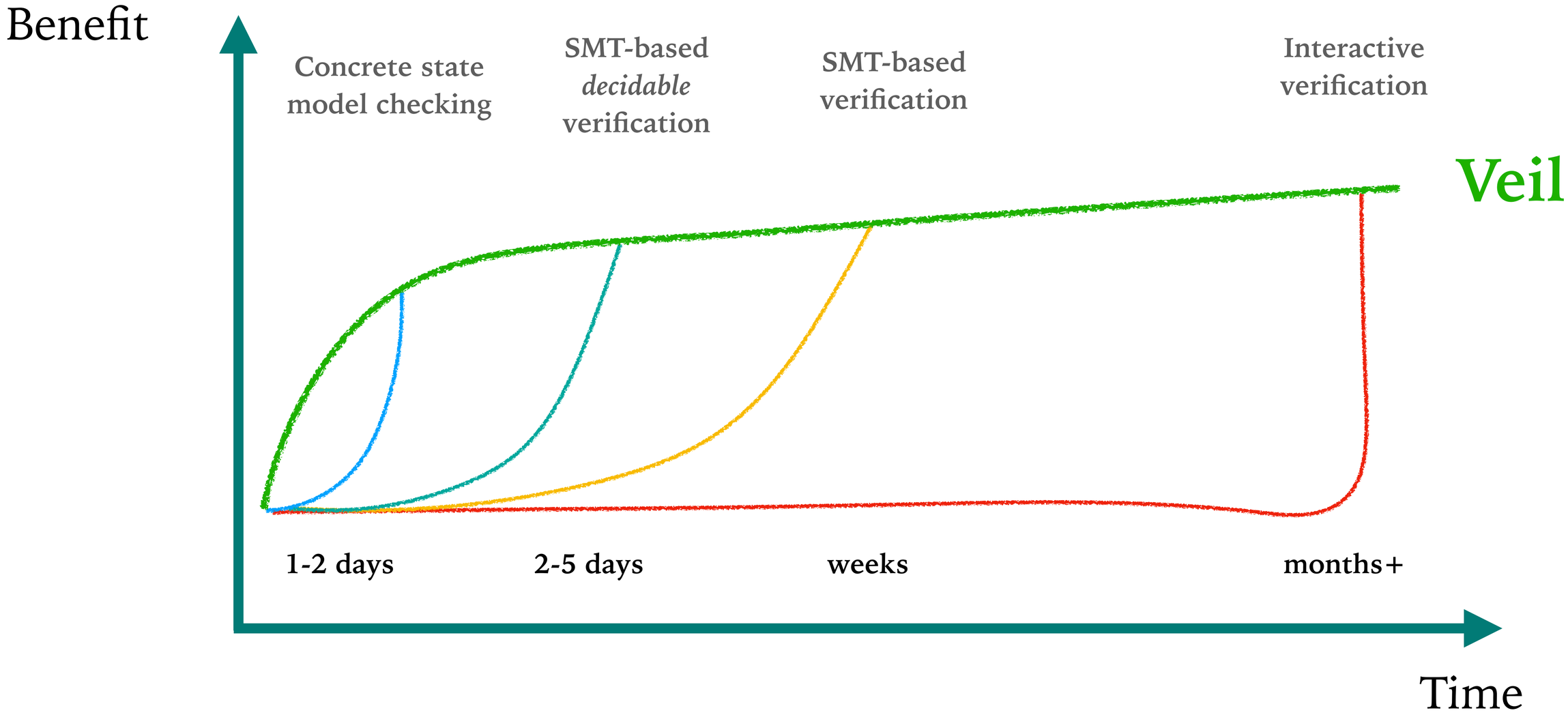
Cedar



*"**Lean is the core verification technology behind Cedar**, the open-source authorization language that powers cloud services like Amazon Verified Permissions and AWS Verified Access. Our team rigorously formalizes and verifies core components of Cedar using Lean's proof assistant, and we leverage **Lean's lightning-fast runtime** to continuously test our production Rust code against the Lean formalization. Lean's efficiency, extensive libraries, and vibrant community **enable us to develop and maintain Cedar at scale**, while ensuring the key correctness and security properties that our users depend on." — Emina Torlak, Senior Principal Applied Scientist, AWS*

slide credit: Leo de Moura

The Future of Verifying Distributed Protocols



Transition Systems

Transition Systems

- Formal model of computer systems, with two components:
  - **States**, some of which are *initial states*
  - **Transitions** between states
- **Reachability:** transitive closure of the transition relation wrt. initial states
- **Invariant:** state property that holds on all reachable states
- **Inductive invariant:** invariant that is preserved by every transition

Safety and Liveness

- **Safety**: “Bad thing will not happen.”
- **Liveness**: “Good thing will eventually happen.”

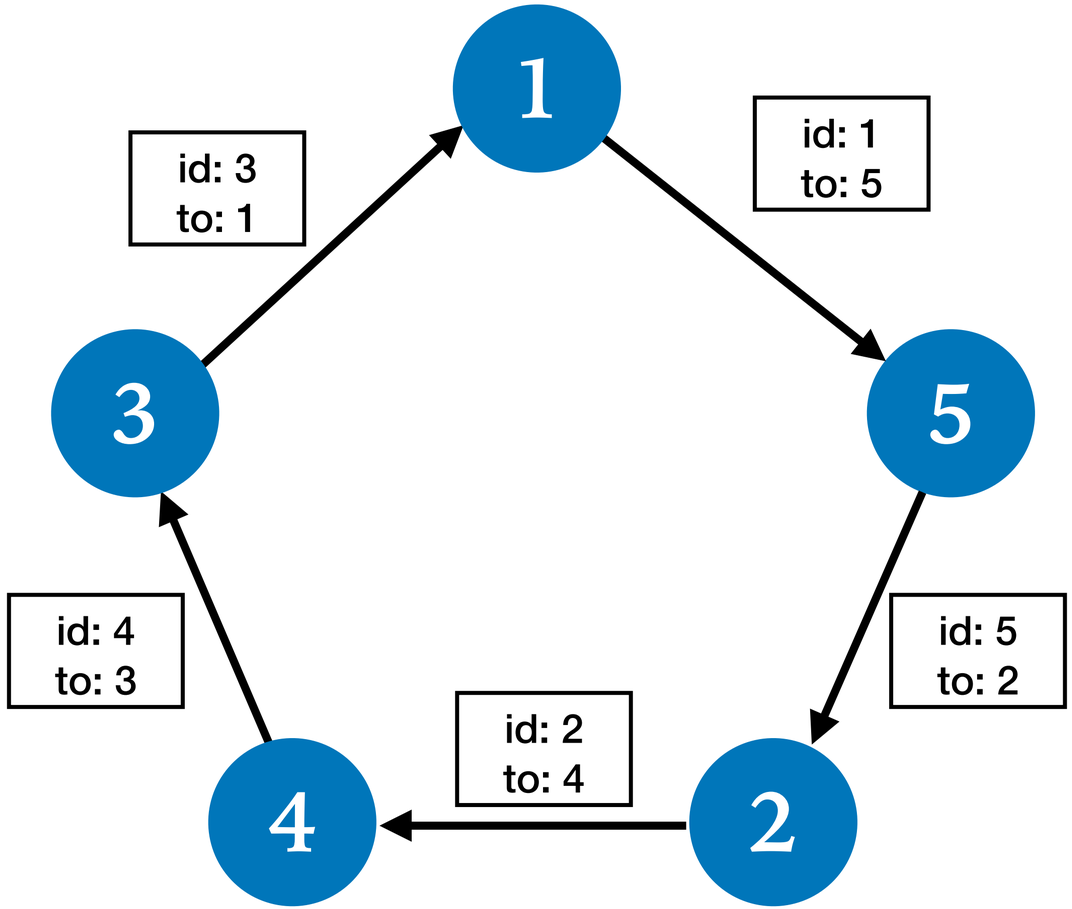
Safety bugs can be reliably discovered with explicit state model checking.

Modelling Distributed Protocols

- Distributed protocols are commonly modelled as transition systems
  - Multiple nodes
  - Each transition is *atomically* performed by *one* node
  - Nodes communicate by sending and receiving messages

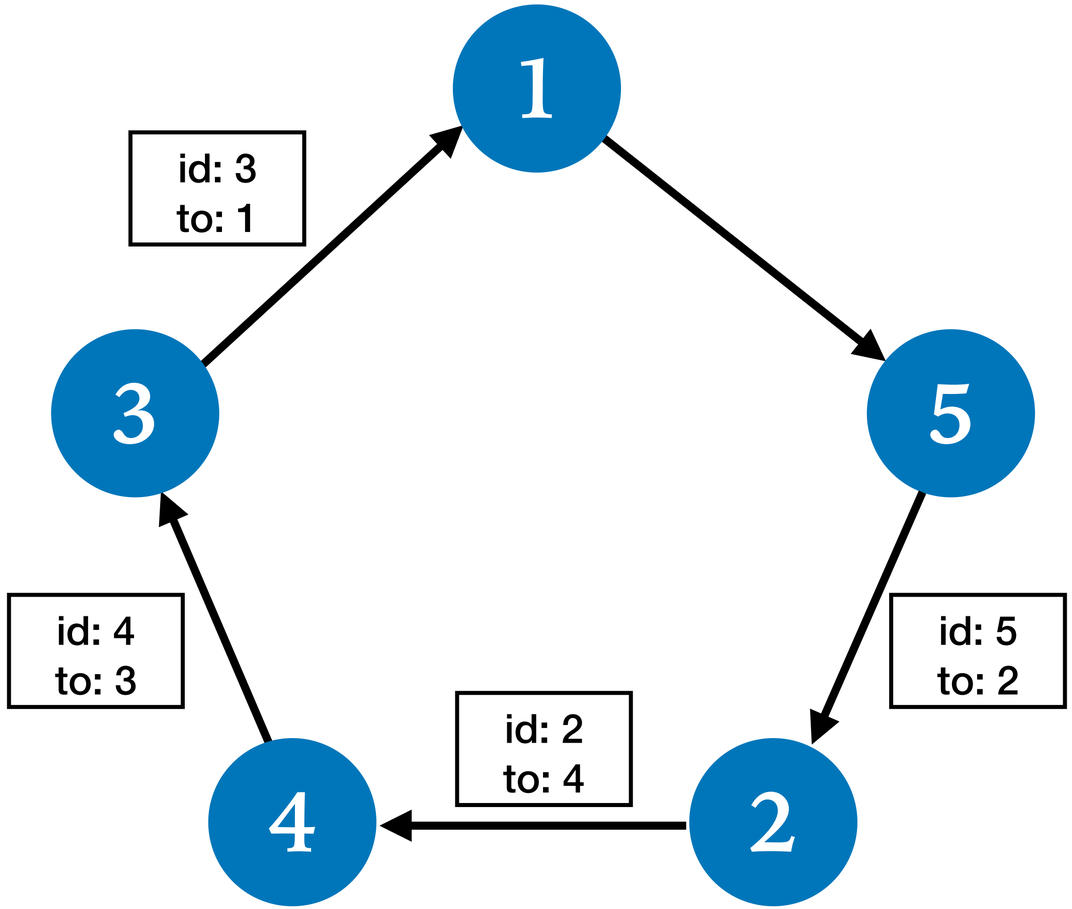
Leader Election in a Ring

Ring Leader Election



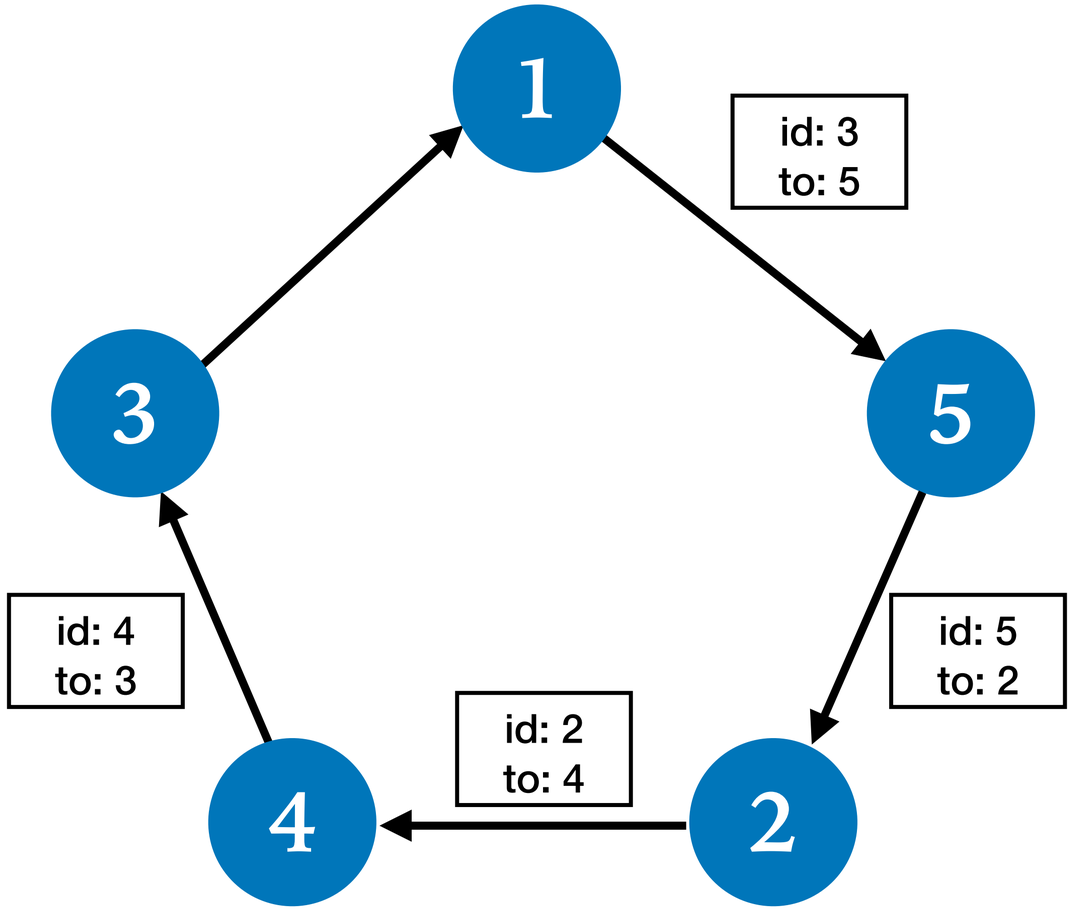
Node sending its own ID to its successor

Ring Leader Election



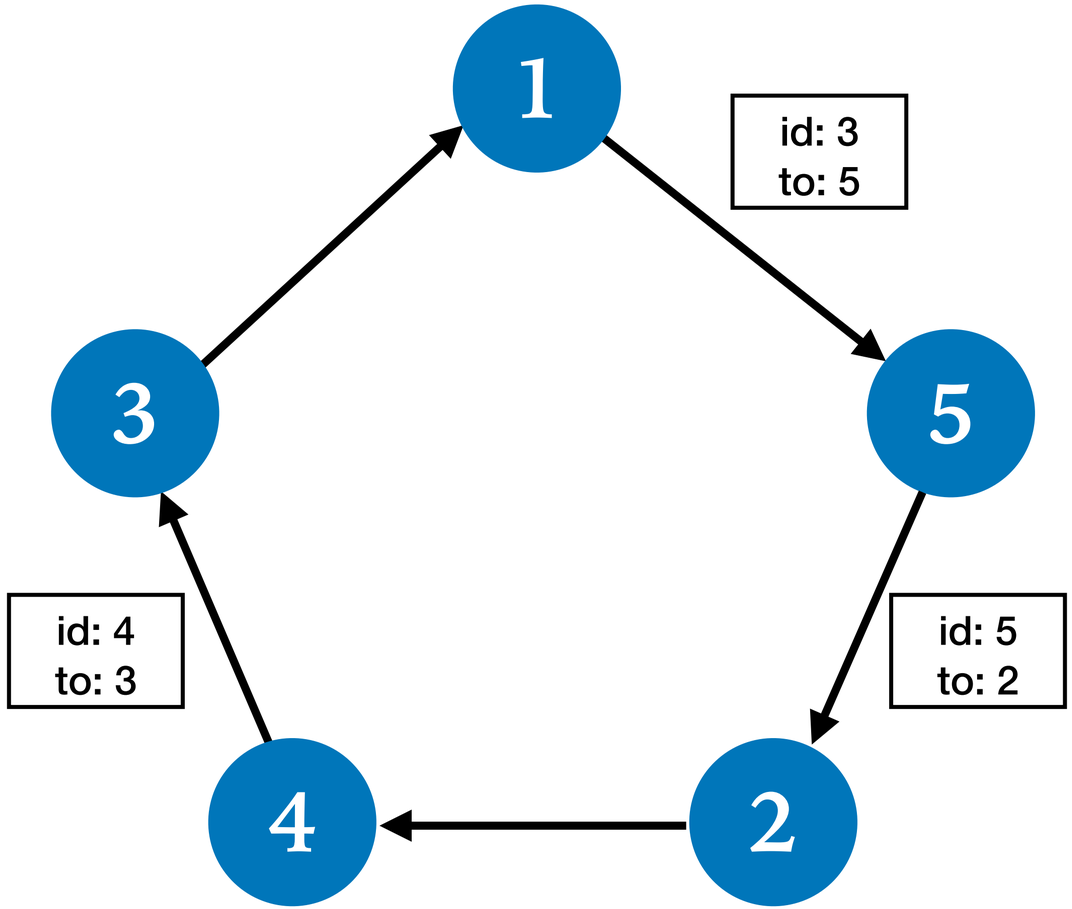
Node discarding message
whose ID is smaller than its own

Ring Leader Election

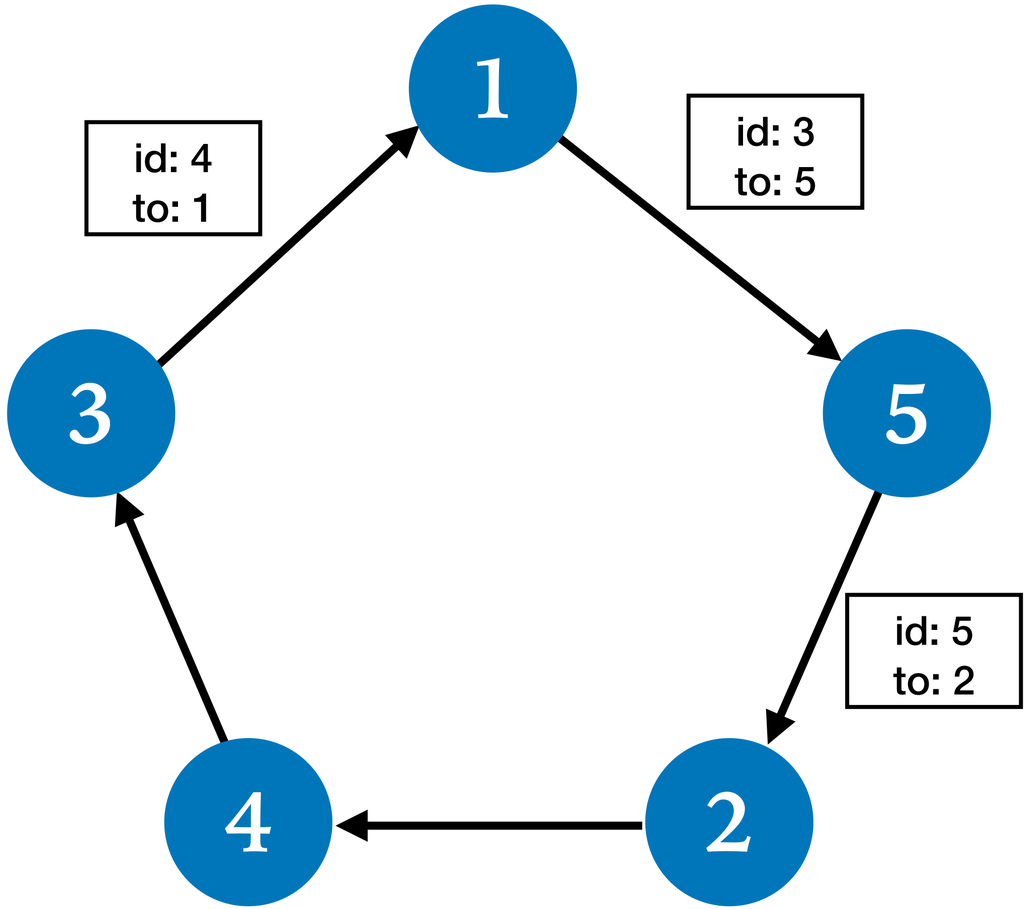


Node forwarding message
whose ID is larger than its own

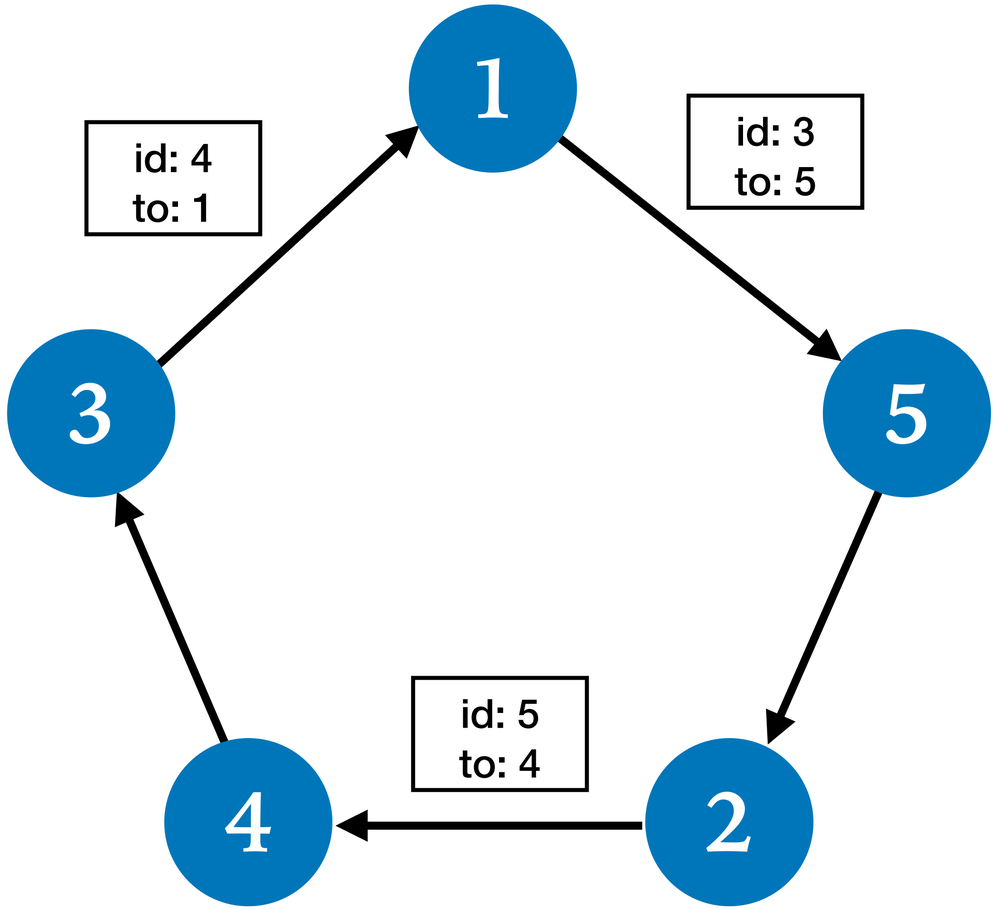
Ring Leader Election



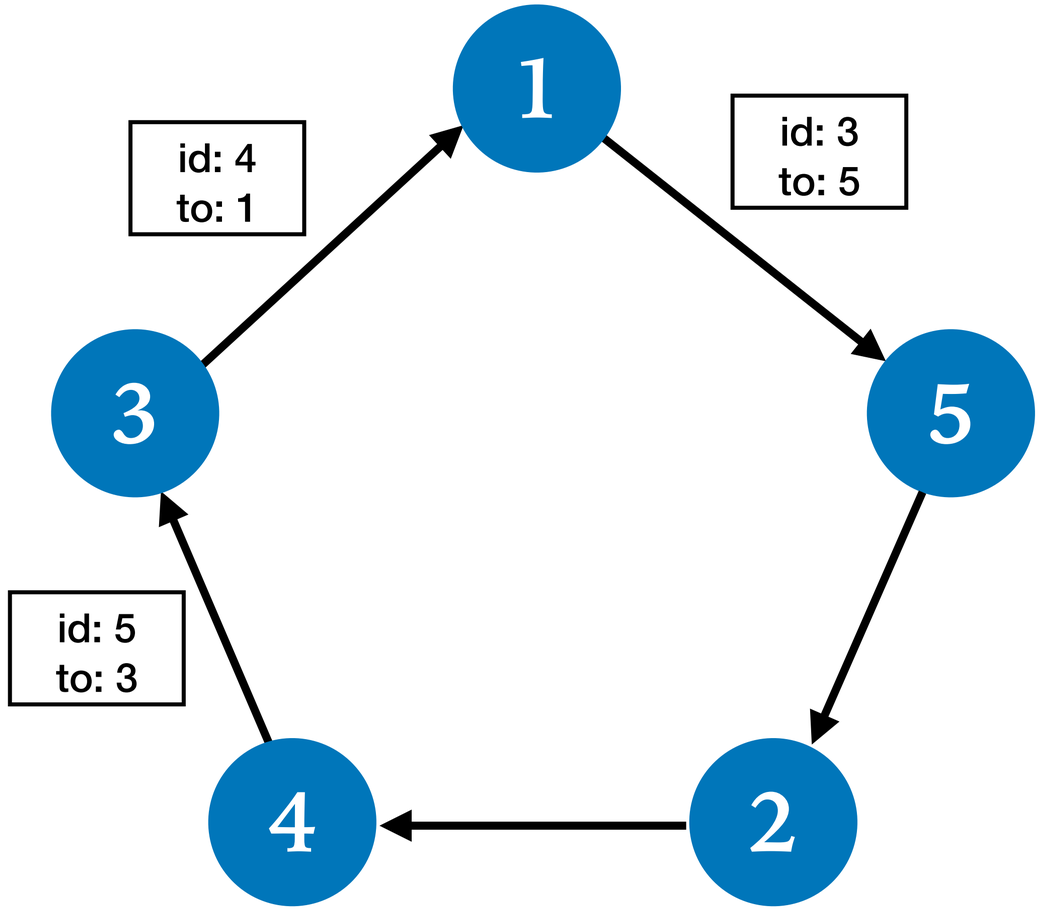
Ring Leader Election



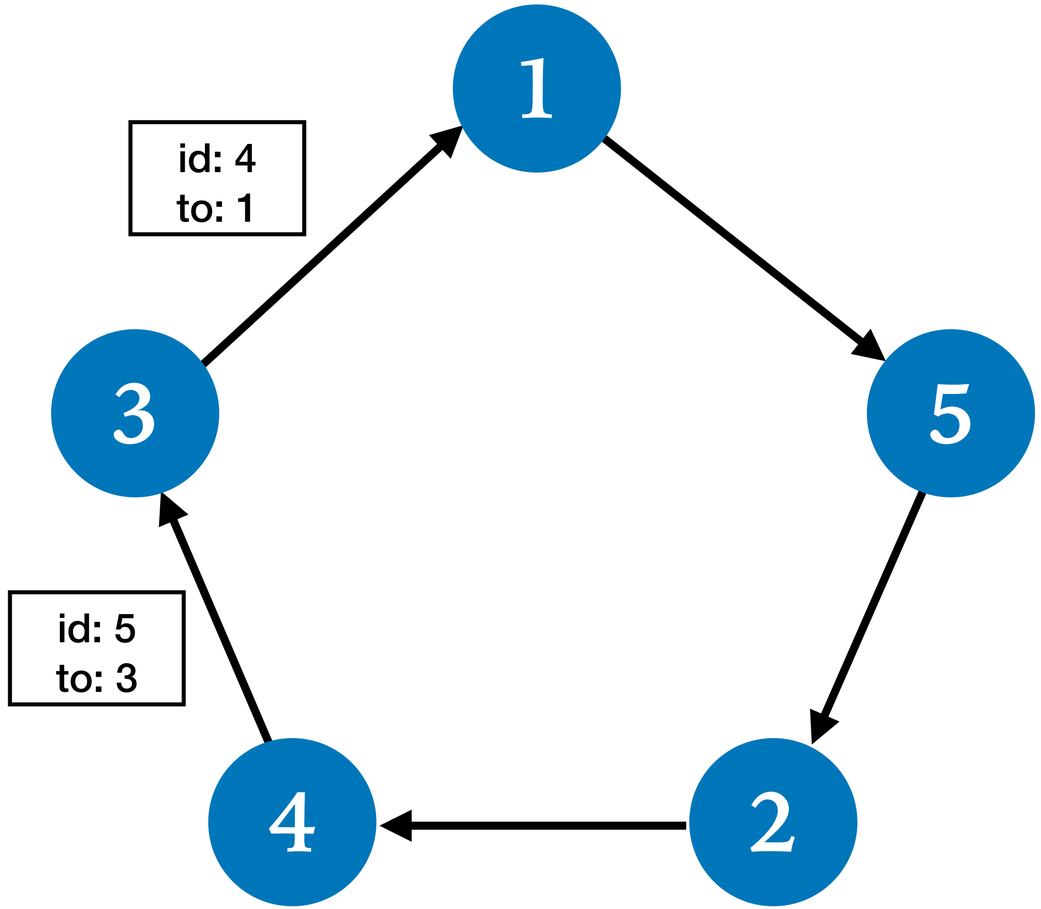
Ring Leader Election



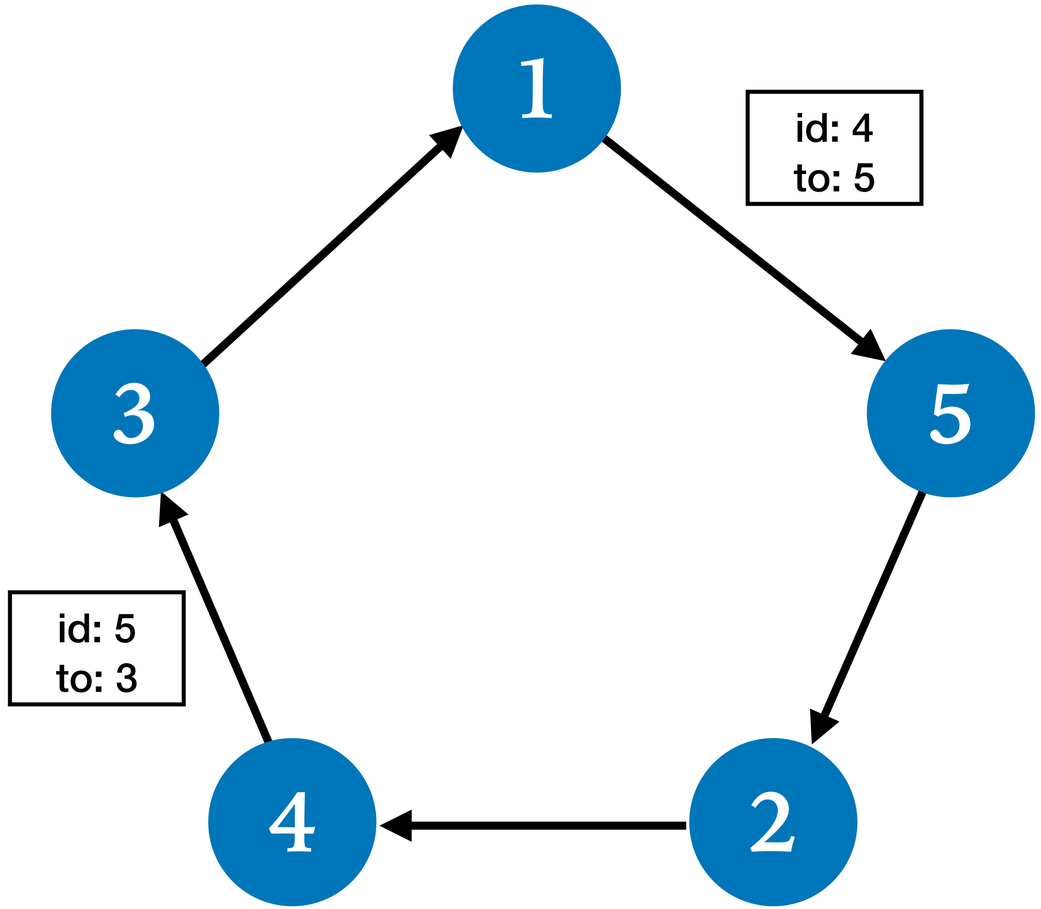
Ring Leader Election



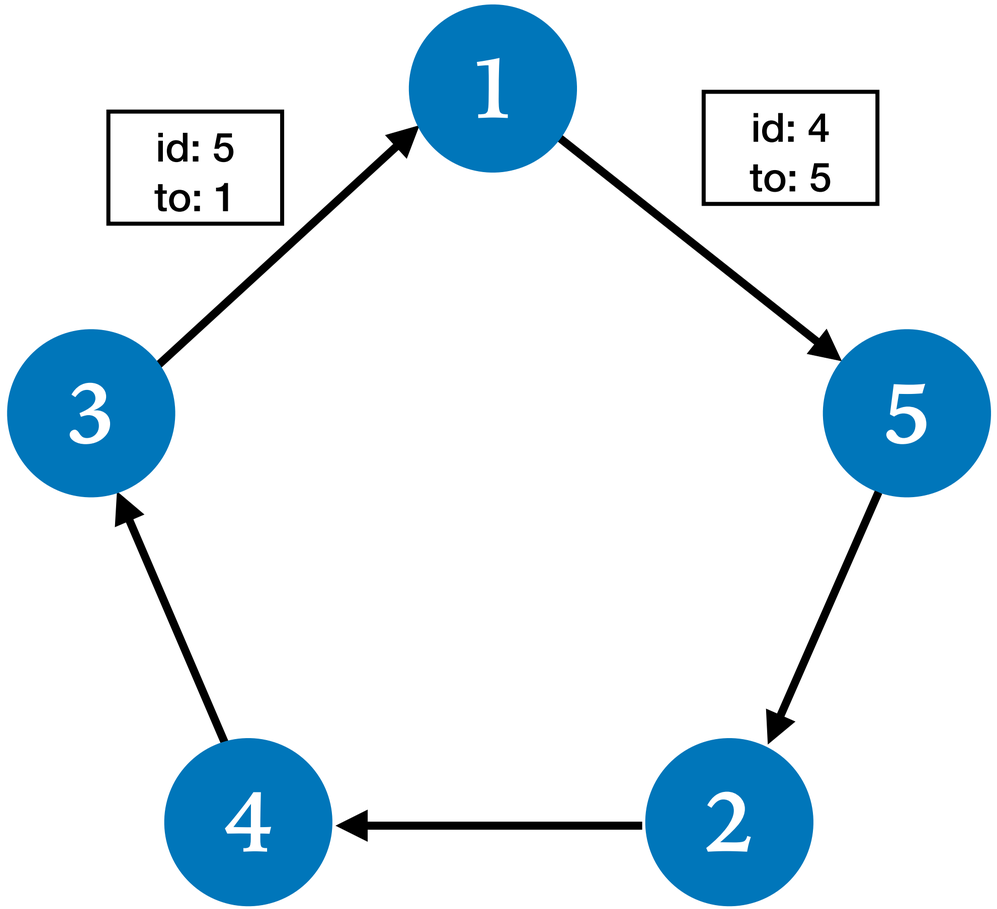
Ring Leader Election



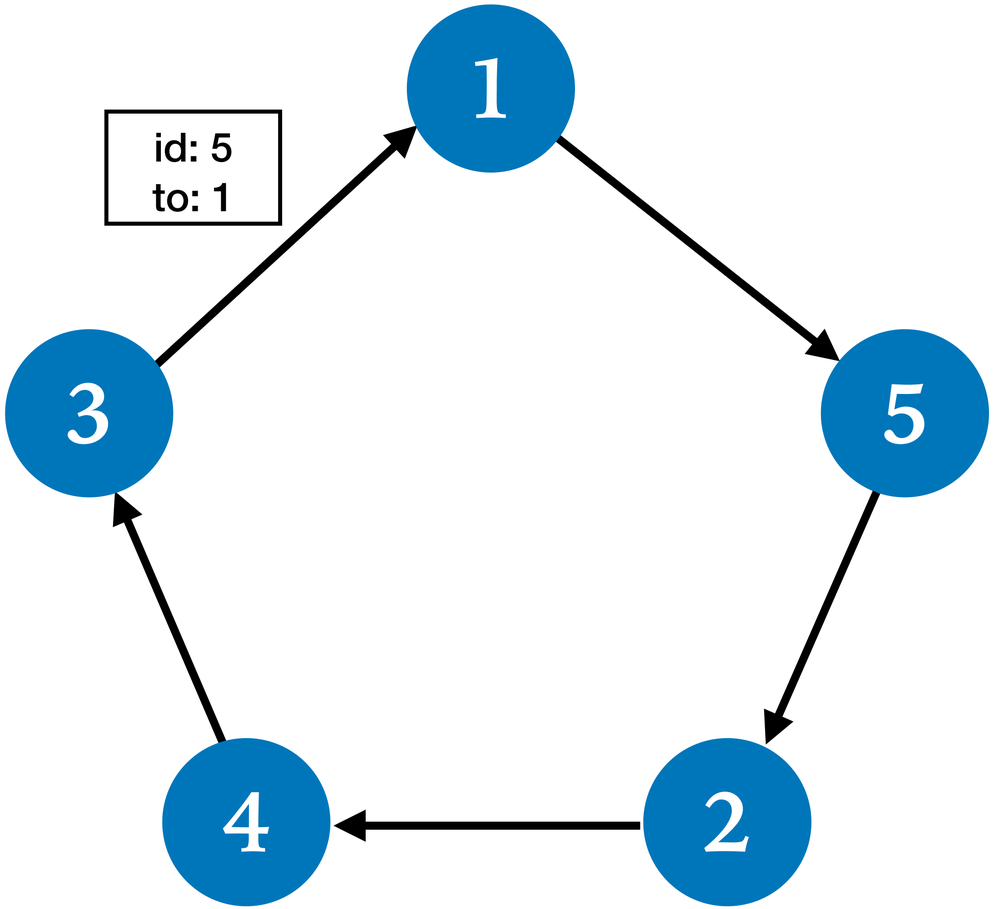
Ring Leader Election



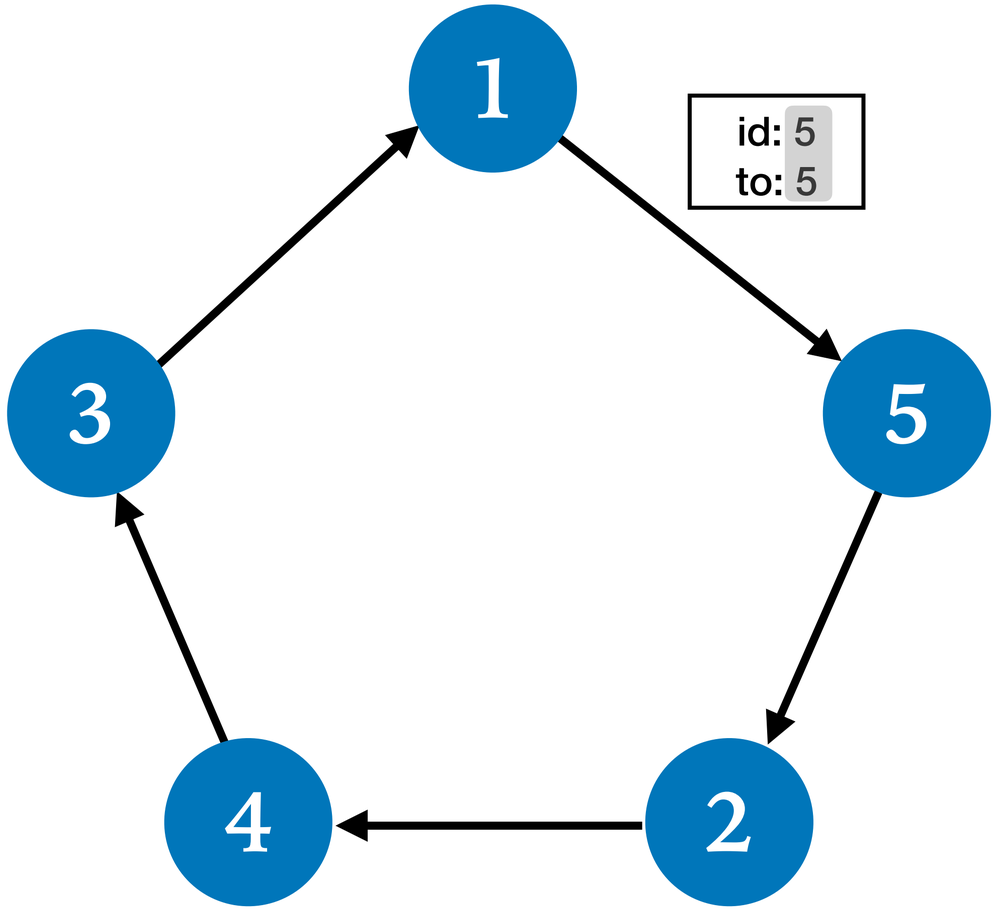
Ring Leader Election



Ring Leader Election

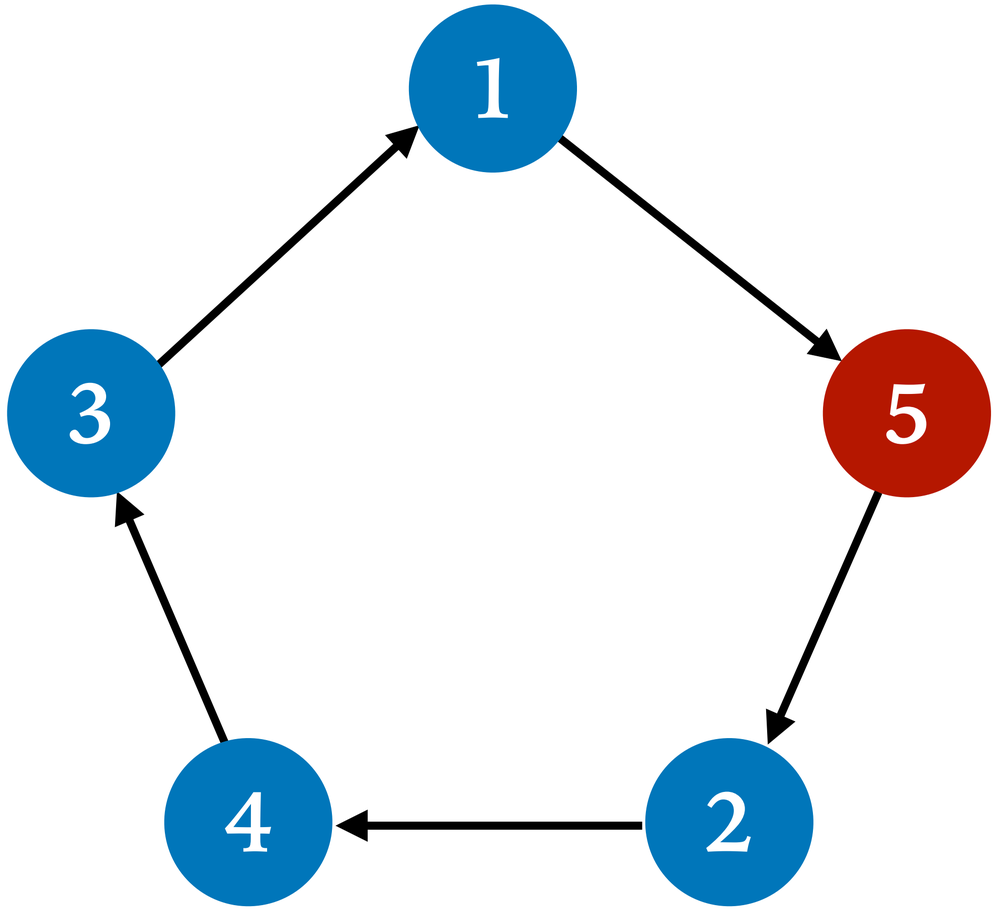


Ring Leader Election



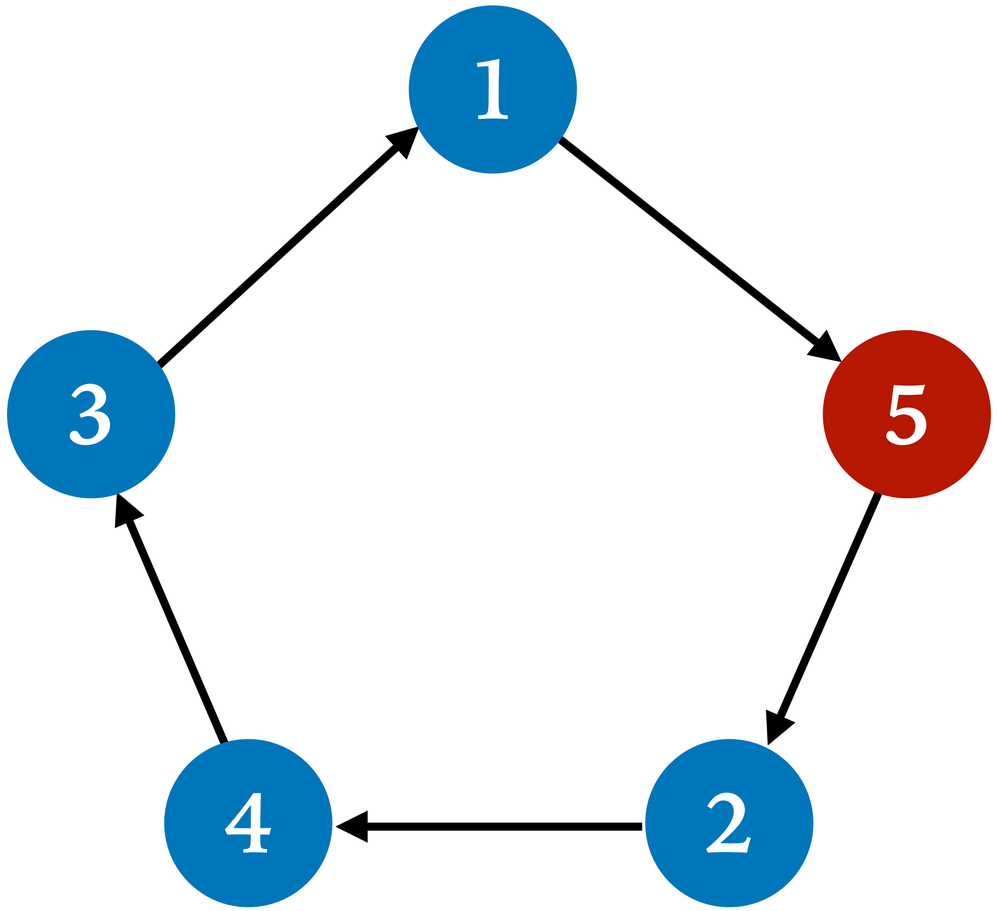
Node declaring itself as leader on receiving a message with its own ID

Ring Leader Election



leader elected

Safety Property to Verify



There is at most one leader.

Demo 1

(Examples/Tutorial/RingNat.lean in the repository, or
Ring (Concrete) in the online playground)

Questions?

Intermezzo: Die Hard 3

In *Die Hard 3*, McClane (Bruce Willis) and Zeus (Samuel L. Jackson) must defuse a bomb at a park fountain by placing exactly 4 gallons of water on a scale, using only a 5-gallon jug and a 3-gallon jug with no other measuring tools.

https://www.youtube.com/watch?v=2vdF6NASMiE

Symbolic Reasoning

Symbolic Reasoning

- Given a system **S** and a safety property **P**, we want to *prove* that all reachable states of **S** satisfy **P**
  - **Symbolic testing**: sound, but not complete → fully automated
  - **Symbolic verification**: sound and complete → semi-automated
    - Requires *inductive invariants* to be provided

Symbolic Representations

- Key idea:
  - Represent the set of reachable states of the system as a formula: ***States***
  - Represent the set of safe states as another formula: ***Safety***
  - Prove ***States*** → ***Safety***

Under and Over-Approximation

- Two options for representing reachable states:
  - **Under-approximation**: only consider states reachable in k steps
  - **Over-approximation**: states described by an inductive invariant

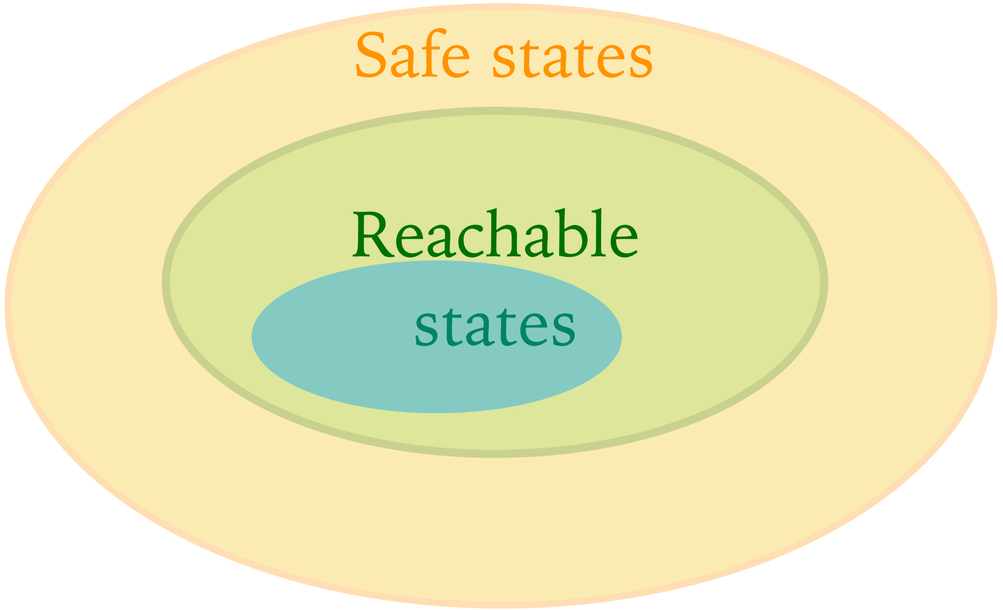
Symbolic Testing

- Symbolic bounded model checking (BMC)
  - $\exists\ s_0\ s_1\ \ldots\ s_k.\ \mathit{Init}\ s_0 \wedge \mathit{Tr}\ s_0\ s_1 \wedge \ldots \wedge \mathit{Tr}\ s_{k-1}\ s_k \wedge P\ s_k$
  - $\forall\ s_0\ s_1\ \ldots\ s_k.\ \mathit{Init}\ s_0 \wedge \mathit{Tr}\ s_0\ s_1 \wedge \ldots \wedge \mathit{Tr}\ s_{k-1}\ s_k \Rightarrow P\ s_k$

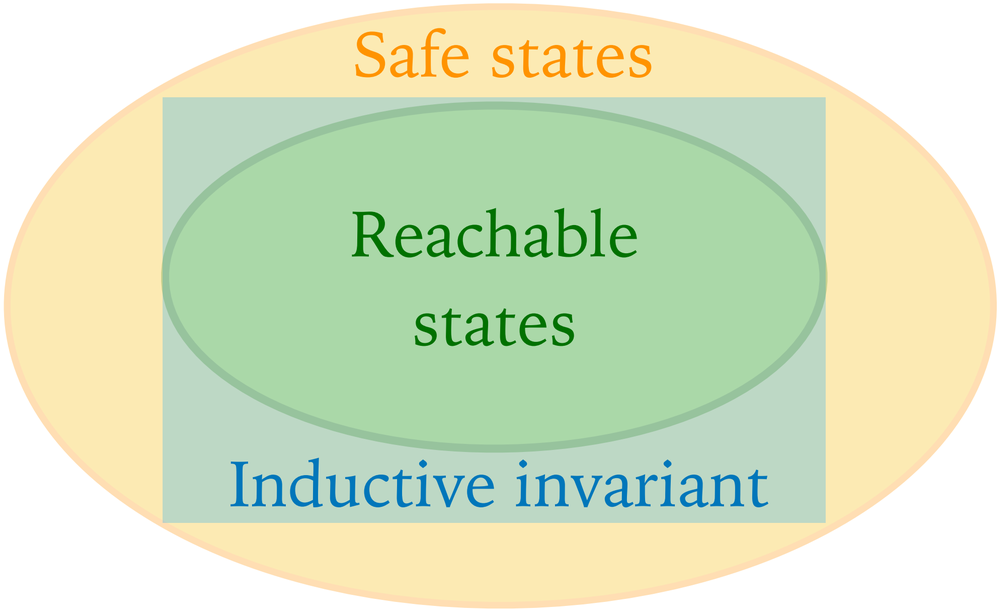
Symbolic Verification

- Inductive invariant that over-approximates the set of reachable states
  - `∀ s. Init s ⇒ Inv s` **Initiation**
  - `∀ s s'. Inv s ∧ Tr s s' ⇒ Inv s'` **Consecution**
  - `∀ s. Inv s ⇒ Safety s` **Safety**
- Corollary: all reachable states satisfy `Inv` and thus `Safety`

Symbolic Testing (under-approximation)



Symbolic Verification (over-approximation)



Abstraction

- Full implementation details are often a hindrance to symbolic reasoning
  - It's helpful to *hide* implementation details behind *abstractions*
- Key idea: operate on an *abstract* state space that is easier to describe and symbolically reason about
  - E.g., rather than talk about natural numbers, talk about uninterpreted sorts with a total order
  - Come up with domain-specific axiomatisations

Decidable Abstractions

- SMT solvers use First-Order Logic for symbolic reasoning
- Some well-studied fragments of FOL are *decidable*
  - These fragments are expressive enough to encode complex distributed protocols (see the entire line of work on Ivy by *Padon et al.*)
- Veil comes with a small library of decidable abstractions for common design patterns in distributed protocols

Demo 2

(Examples/Tutorial/RingDec.lean in the repository, or
Ring (Decidable) in the online playground)

Summary of the Demo

- A protocol state in Veil consists of uninterpreted *types* and *relations*
- *Actions* update the relations; they are *guarded* with `require` clauses
- Reachability of states can be *symbolically tested* using BMC
- Invariants are verified to *hold under all actions*, via SMT or interactively

System Specification

initial state

```
after_init {
  leader N := False
  pending M N := False
}
```

actions

```
action send (n next : node) = {
  require n ≠ next ∧ ∀ Z,
          ((Z ≠ n ∧ Z ≠ next) → btw n next Z)
  pending n next := True
}

action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
}
```

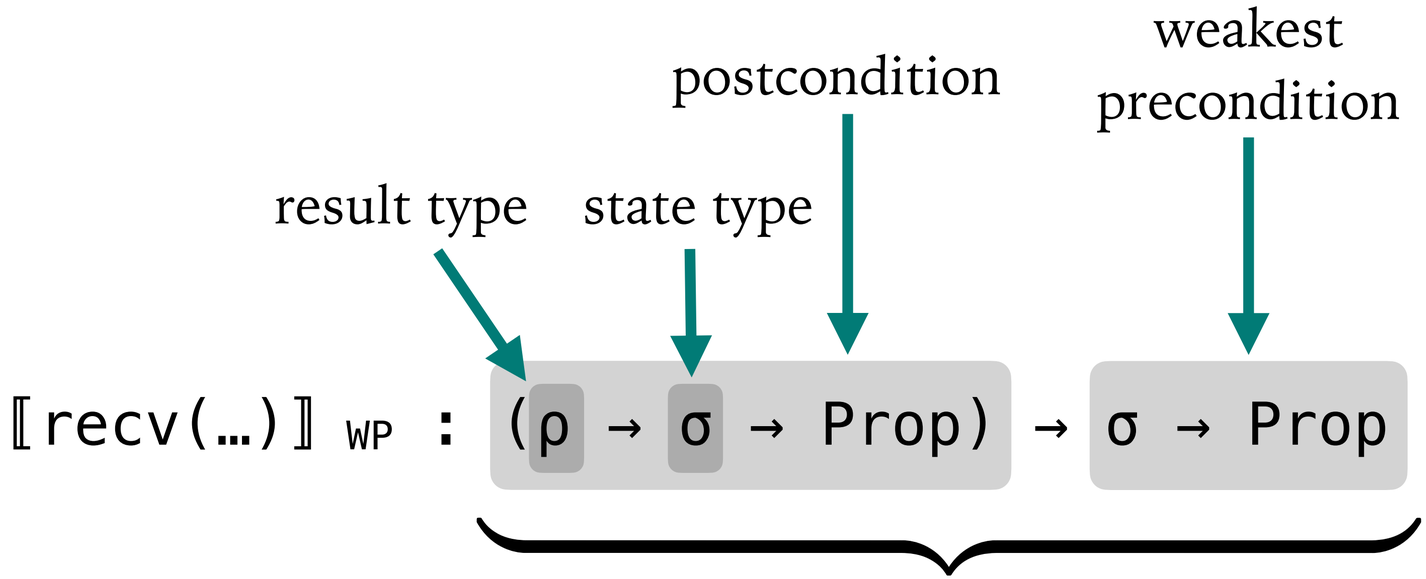
safety properties

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2

invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

Semantics of Actions

```
action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
}
```



Verification Conditions for Safety

```
safety [single_leader] leader L1 ∧ leader L2 → L1 = L2
```

```
invariant [leader_greatest] leader L → le N L
invariant [self_msg_only_if_greatest] pending L L → le N L
invariant [no_bypass] pending S D ∧ btw S N D → le N S
```

```
⟦act⟧ WP : (ρ → σ → Prop) → σ → Prop
```

```
after_init {
  leader N := False
  pending M N := False
}
```

```
Inv = λ (r: ρ) (s: σ). Inv s
```

1. ∀ s. ⟦after_init⟧ WP **Inv** s
2. ∀ act s. **Inv** s ⇒ ⟦act⟧ WP **Inv** s
3. ∀ s. Inv s ⇒ Safety s

Generating Weakest Preconditions Automatically

$$[\![\texttt{act}]\!]_{\texttt{WP}} : \underbrace{(\rho \to \sigma \to \texttt{Prop}) \to \sigma \to \texttt{Prop}}$$

weakest precondition transformer (WP σ ρ)

```
def WP.pure (r : ρ) : WP σ ρ := fun s post => post r s

def WP.bind (c : WP σ ρ) (next : ρ → WP σ ρ) : WP σ ρ :=
                        fun s post => c (fun r s' => next r post s') s
```

a.k.a. a Dijkstra monad

Verifying higher-order programs with the Dijkstra monad, Swamy et al., PLDI'13

Generating Weakest Preconditions Automatically

```
def WP.pure (r : ρ) : WP σ ρ := fun s post => post r s

def WP.bind (c : WP σ ρ) (next : ρ → WP σ ρ) : WP σ ρ :=
                              fun s post => c (fun r s' => next r post s') s
```

```
action recv (id n next : node) = {
  require isNext n next
  require pending id n
  pending id n := *
  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
}
```

Generating Weakest Preconditions Automatically

```
def WP.pure (r : ρ) : WP σ ρ := fun s post => post r s

def WP.bind (c : WP σ ρ) (next : ρ → WP σ ρ) : WP σ ρ :=
  fun s post => c (fun r s' => next r post s') s
```

```
action recv (id n next : node) = do
  require isNext n next
  require pending id n

  let newPending ← fresh
  pending := newPending

  if (id = n) then
    leader n := True
  else
    if (le n id) then
      pending id next := True
```

```
def fresh (τ : Type) : WP σ τ := fun s post => ∀ (t : τ), post t s
```

Dolev-Strong FloodSet Consensus

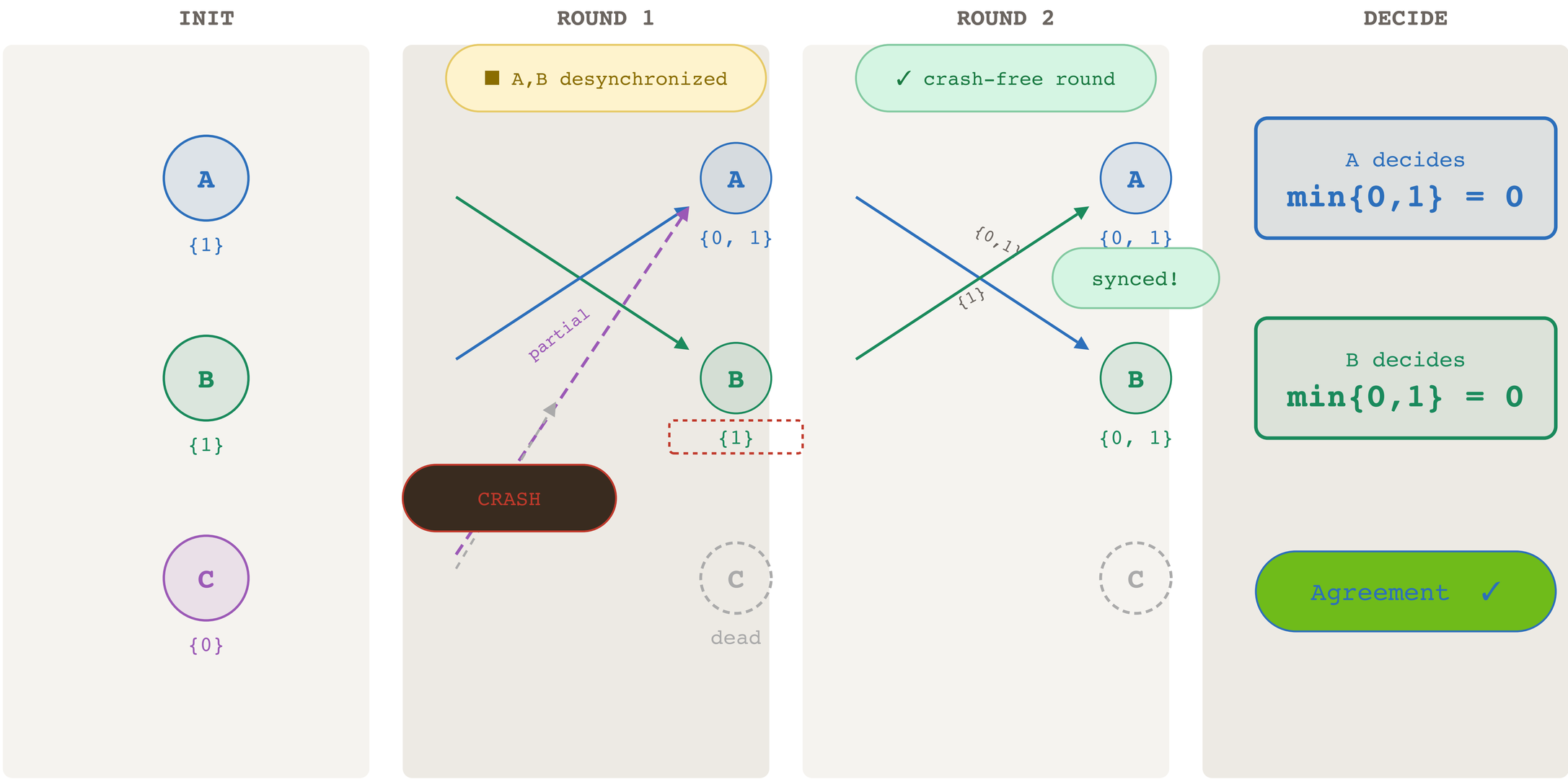
How FloodSet Works

Synchronous crash-fault consensus via value flooding

- **Synchrony:** communication proceeds in lock-step rounds with guaranteed delivery before each round ends.
- **Crash faults:** at most t of n nodes may crash; a crashing node may deliver to an arbitrary subset of peers.
- **Flooding:** over t+1 rounds, every alive node broadcasts its entire seen set to all others.
- **Pigeonhole:** with t crashes over t+1 rounds, at least one round is crash-free, synchronizing all alive nodes.
- **Agreement** after t+1 rounds every alive node decides min(seen); identical seen sets guarantee the same decision.
- **Validity** every seen value traces back to some node's proposal, so the decision is always genuine.

FloodSet Consensus Protocol

Example: 3 nodes, 2 values, t = 1 2 rounds | C proposes 0, crashes in round 1



Non-Trivial Behaviours

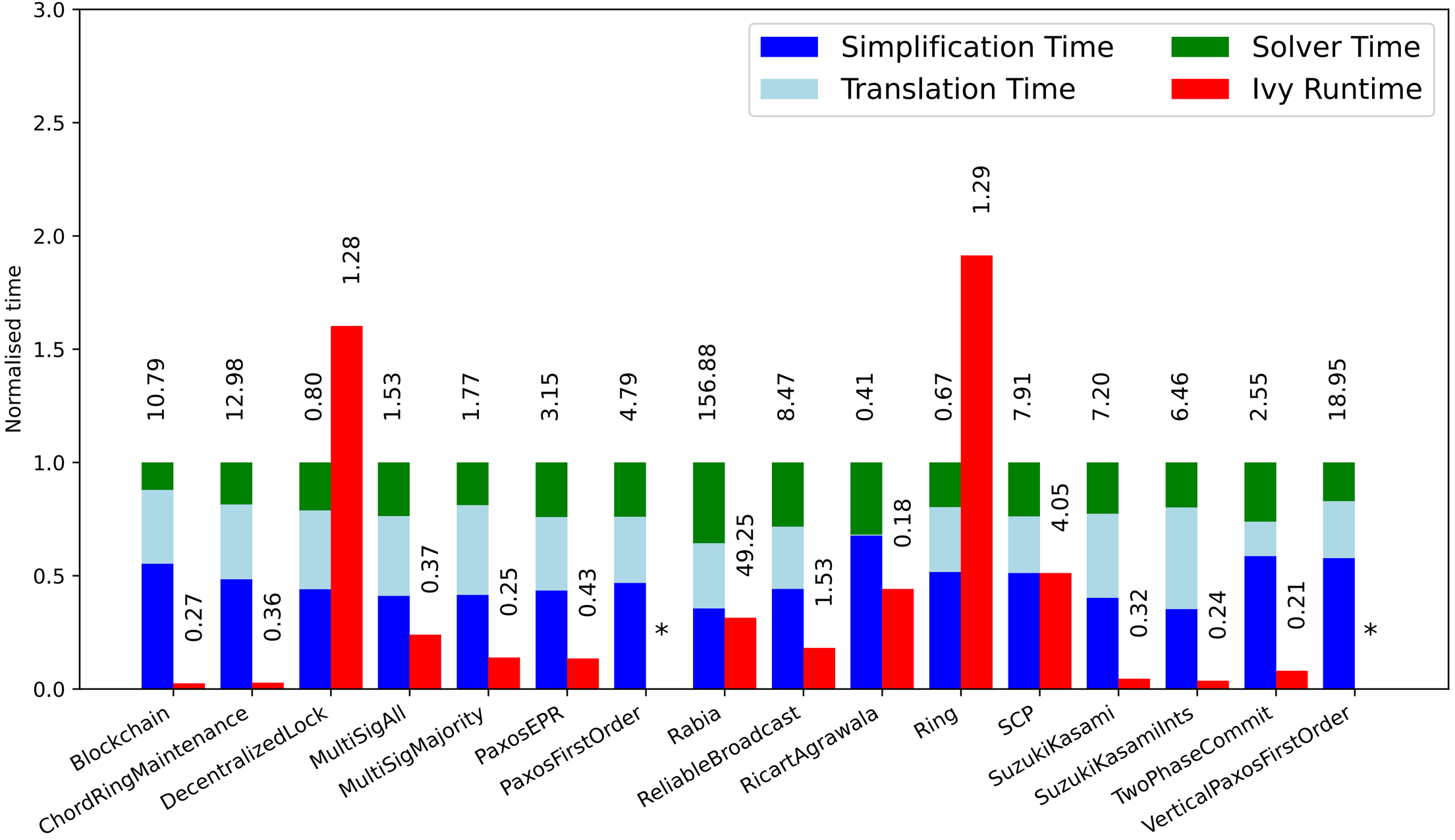
Subtle edge cases that make FloodSet verification interesting

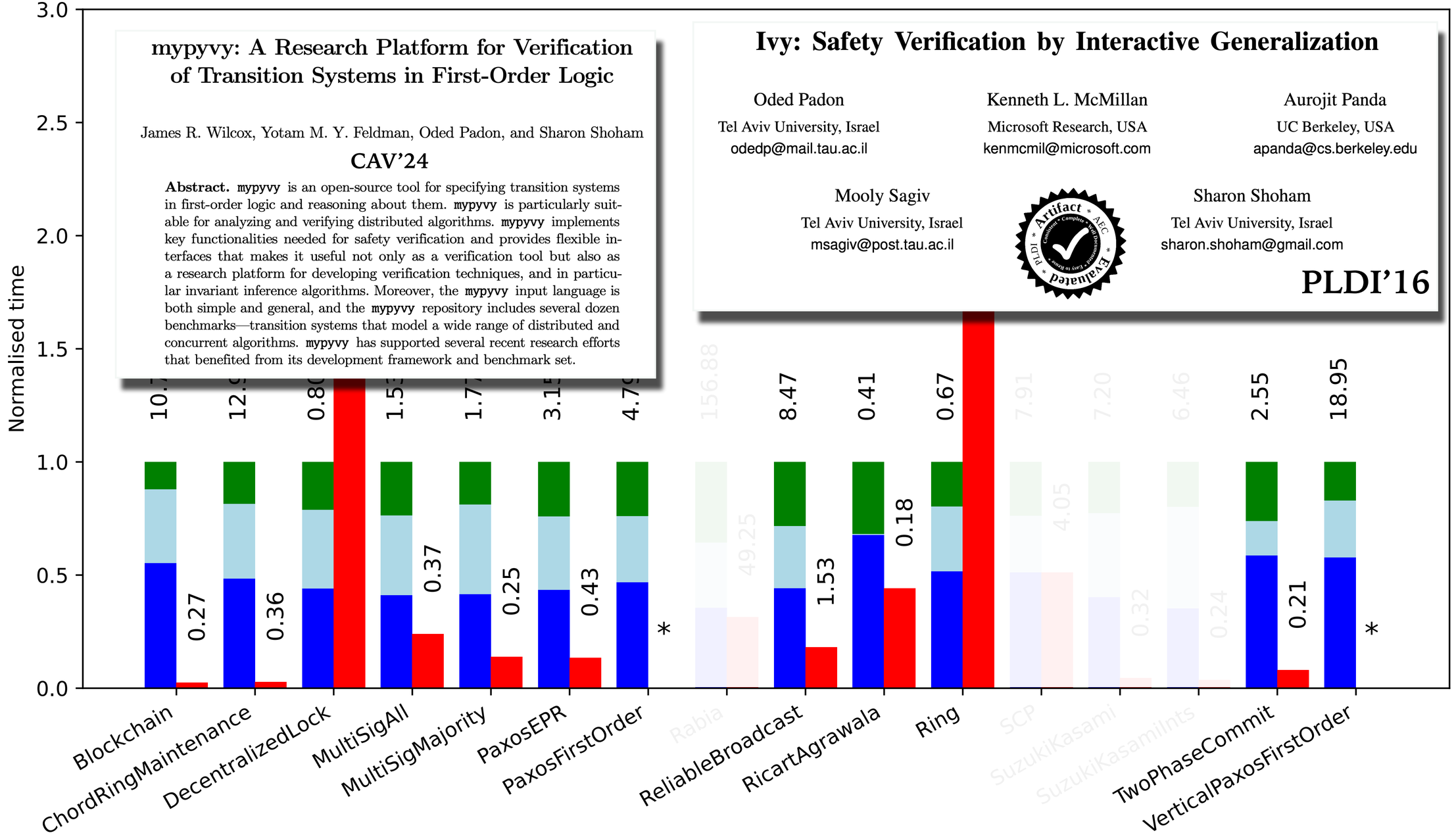
- **Partial delivery:** A crashing node may send its values to some peers but not others, causing temporary asymmetric seen sets among alive nodes.
- **Delayed healing:** A crash in round r can desynchronize alive nodes, but the next crash-free round repairs the asymmetry — the healing may happen several rounds later.
- **Multiple crashes, same round:** Several nodes can crash in a single round (up to the budget t), each with independent partial-delivery sets — the adversary chooses who receives what.
- **Crash-free round isn't the last:** Synchronization can occur in any round, not just the final one; subsequent crash-free rounds simply preserve the already-identical seen sets.

Demo 3

(Examples/Synchronous/FloodSet.lean)

More Case Studies


Simplification Time
Translation Time
Solver Time
Ivy Runtime
Normalised time
3.0
2.5
2.0
1.5
1.0
0.5
0.0
10.79
0.27
12.98
0.36
0.80
1.28
1.53
0.37
1.77
0.25
3.15
0.43
4.79
*
156.88
49.25
8.47
1.53
0.41
0.18
0.67
1.29
7.91
4.05
7.20
0.32
6.46
0.24
2.55
0.21
18.95
*
Blockchain
ChordRingMaintenance
DecentralizedLock
MultiSigAll
MultiSigMajority
PaxosEPR
PaxosFirstOrder
Rabia
ReliableBroadcast
RicartAgrawala
Ring
SCP
SuzukiKasami
SuzukiKasamiInts
TwoPhaseCommit
VerticalPaxosFirstOrder



mypyvy: A Research Platform for Verification of Transition Systems in First-Order Logic

James R. Wilcox, Yotam M. Y. Feldman, Oded Padon, and Sharon Shoham

CAV'24

Abstract. `mypyvy` is an open-source tool for specifying transition systems in first-order logic and reasoning about them. `mypyvy` is particularly suitable for analyzing and verifying distributed algorithms. `mypyvy` implements key functionalities needed for safety verification and provides flexible interfaces that makes it useful not only as a verification tool but also as a research platform for developing verification techniques, and in particular invariant inference algorithms. Moreover, the `mypyvy` input language is both simple and general, and the `mypyvy` repository includes several dozen benchmarks—transition systems that model a wide range of distributed and concurrent algorithms. `mypyvy` has supported several recent research efforts that benefited from its development framework and benchmark set.

Ivy: Safety Verification by Interactive Generalization

Oded Padon
Tel Aviv University, Israel
odedp@mail.tau.ac.il

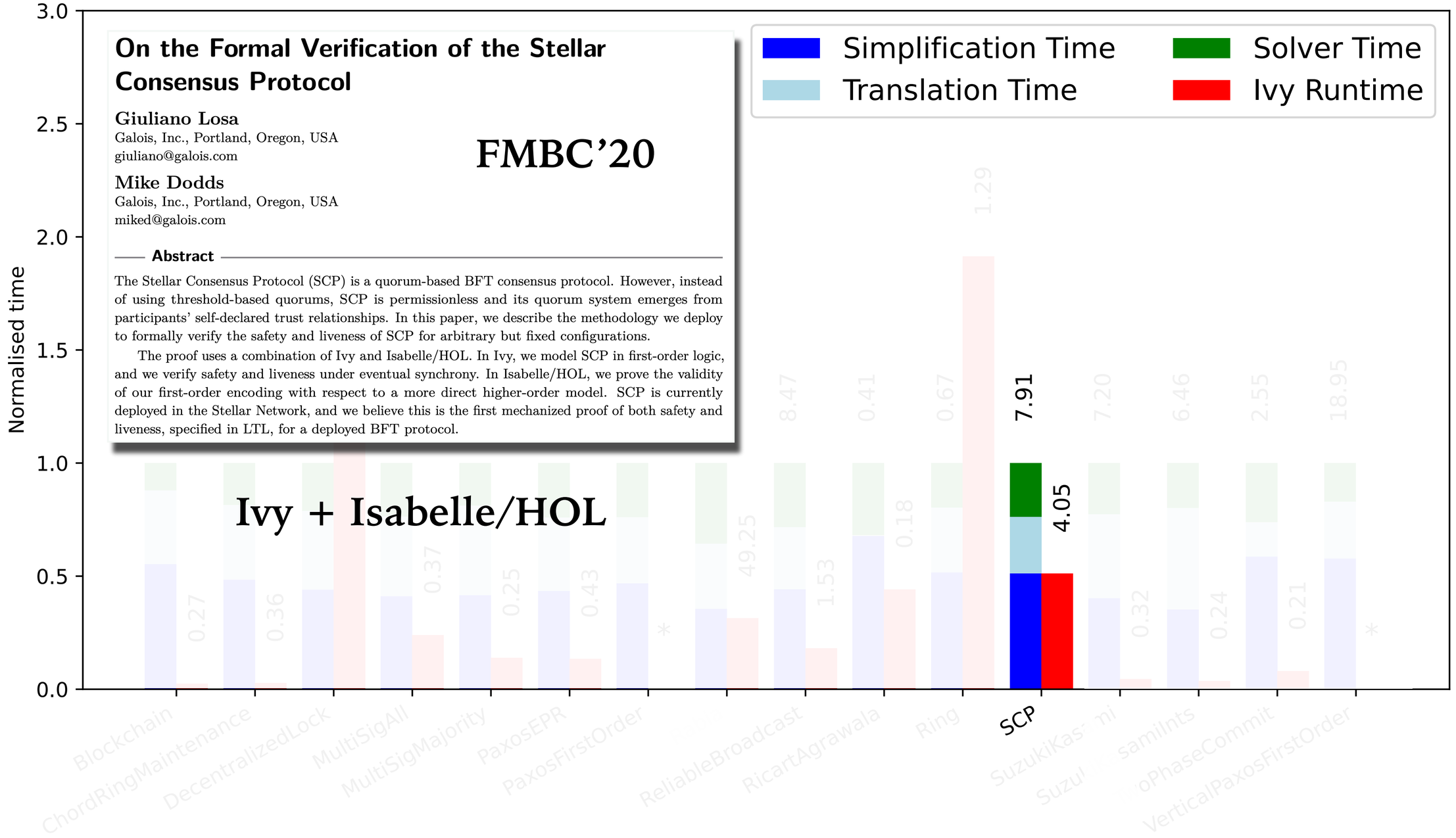
Kenneth L. McMillan
Microsoft Research, USA
kenmcmil@microsoft.com

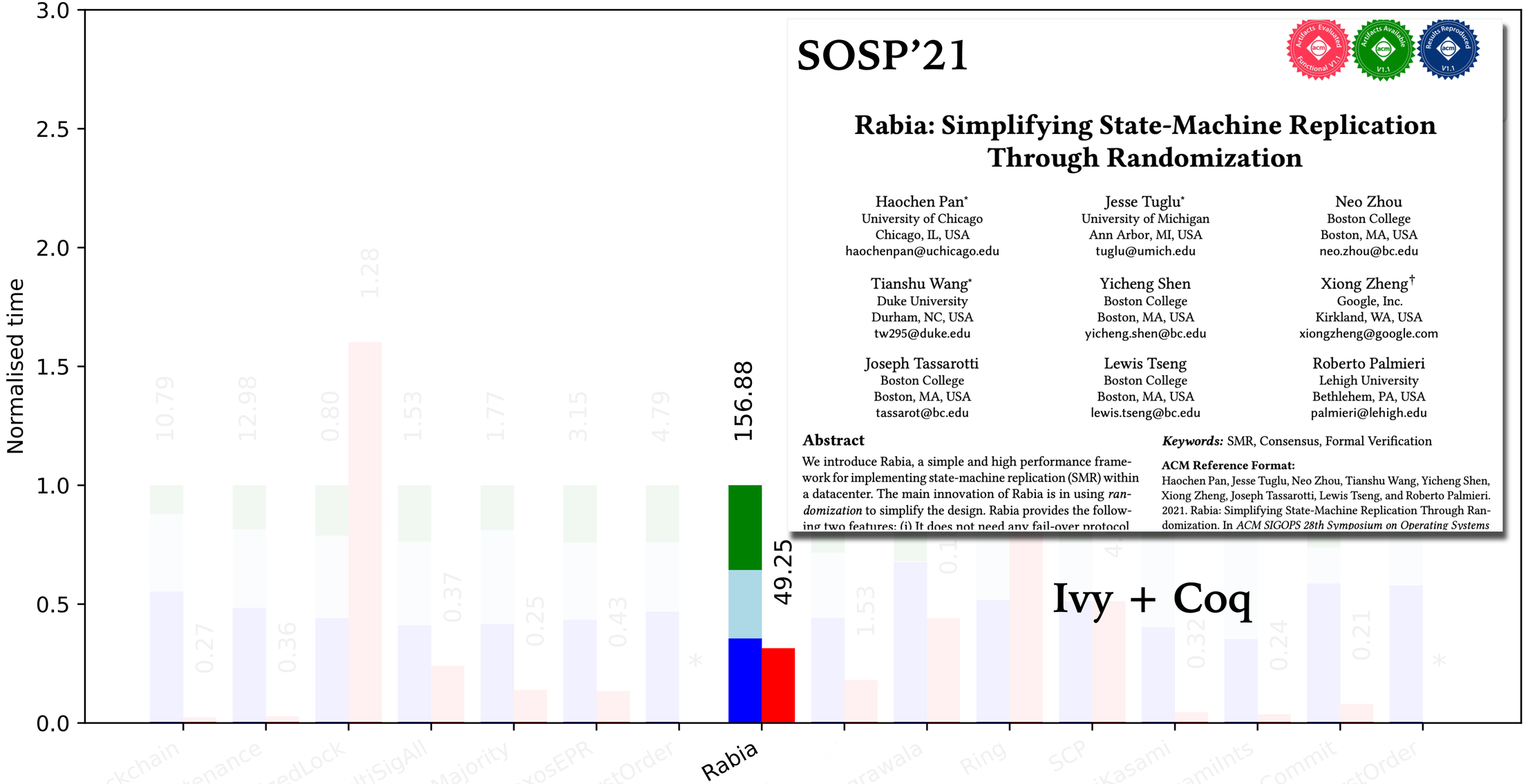
Aurojit Panda
UC Berkeley, USA
apanda@cs.berkeley.edu

Mooly Sagiv
Tel Aviv University, Israel
msagiv@post.tau.ac.il

Sharon Shoham
Tel Aviv University, Israel
sharon.shoham@gmail.com

PLDI'16


On the Formal Verification of the Stellar Consensus Protocol
Giuliano Losa
Galois, Inc., Portland, Oregon, USA
giuliano@galois.com
Mike Dodds
Galois, Inc., Portland, Oregon, USA
miked@galois.com
FMBC'20
Abstract
The Stellar Consensus Protocol (SCP) is a quorum-based BFT consensus protocol. However, instead of using threshold-based quorums, SCP is permissionless and its quorum system emerges from participants' self-declared trust relationships. In this paper, we describe the methodology we deploy to formally verify the safety and liveness of SCP for arbitrary but fixed configurations.
The proof uses a combination of Ivy and Isabelle/HOL. In Ivy, we model SCP in first-order logic, and we verify safety and liveness under eventual synchrony. In Isabelle/HOL, we prove the validity of our first-order encoding with respect to a more direct higher-order model. SCP is currently deployed in the Stellar Network, and we believe this is the first mechanized proof of both safety and liveness, specified in LTL, for a deployed BFT protocol.
Ivy + Isabelle/HOL
Simplification Time
Translation Time
Solver Time
Ivy Runtime
Normalised time
3.0
2.5
2.0
1.5
1.0
0.5
0.0
7.91
4.05
SCP
Blockchain
ChordRingMaintenance
DecentralizedLock
MultiSigAll
MultiSigMajority
PaxosEPR
PaxosFirstOrder
ReliableBroadcast
RicartAgrawala
Ring
SuzukiKas
Suzul
samiInts
oPhaseCommit
VerticalPaxosFirstOrder



SOSP'21

Rabia: Simplifying State-Machine Replication Through Randomization

Haochen Pan*
University of Chicago
Chicago, IL, USA
haochenpan@uchicago.edu

Jesse Tuglu*
University of Michigan
Ann Arbor, MI, USA
tuglu@umich.edu

Neo Zhou
Boston College
Boston, MA, USA
neo.zhou@bc.edu

Tianshu Wang*
Duke University
Durham, NC, USA
tw295@duke.edu

Yicheng Shen
Boston College
Boston, MA, USA
yicheng.shen@bc.edu

Xiong Zheng†
Google, Inc.
Kirkland, WA, USA
xiongzheng@google.com

Joseph Tassarotti
Boston College
Boston, MA, USA
tassarot@bc.edu

Lewis Tseng
Boston College
Boston, MA, USA
lewis.tseng@bc.edu

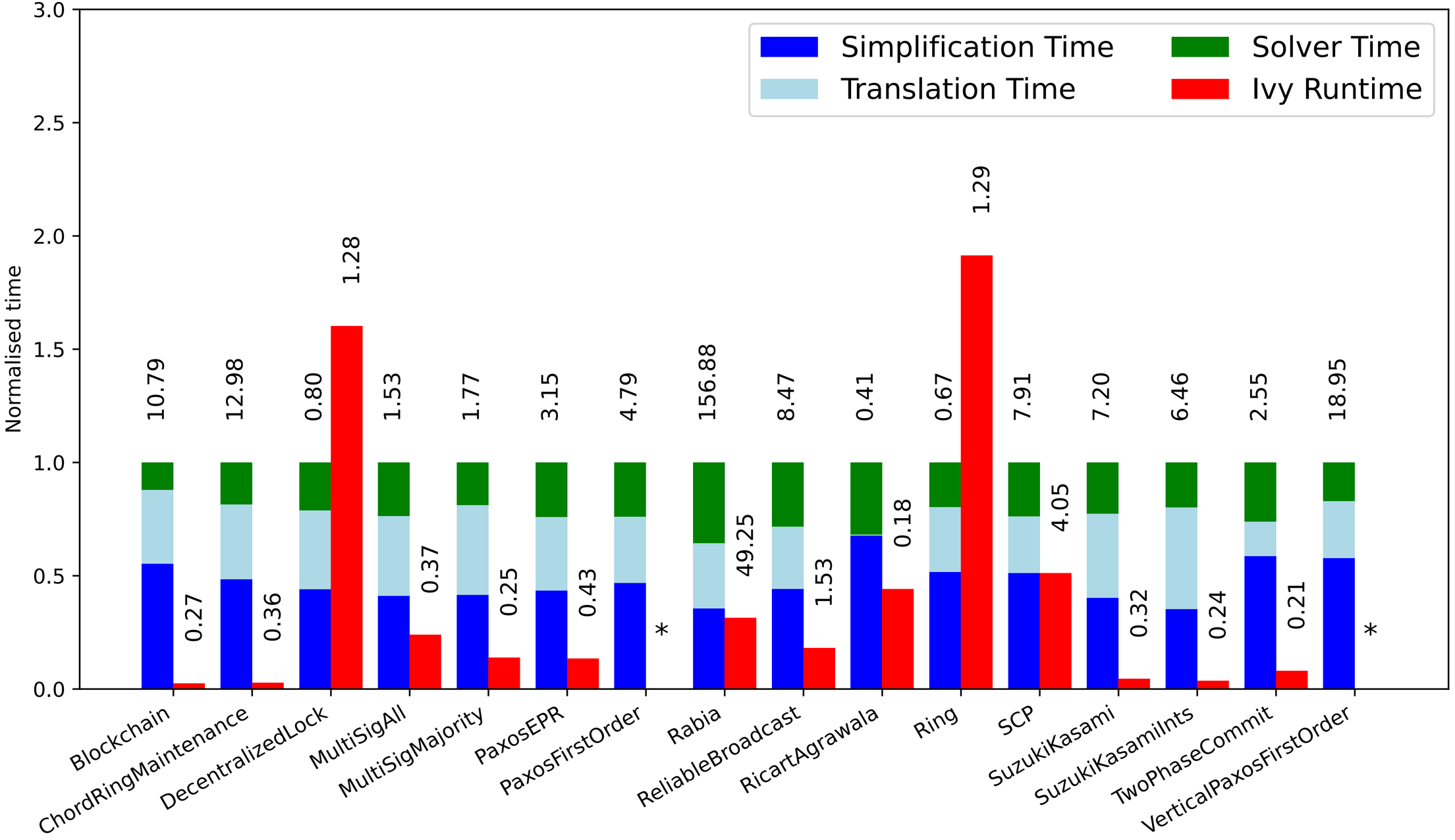
Roberto Palmieri
Lehigh University
Bethlehem, PA, USA
palmieri@lehigh.edu

Abstract

We introduce Rabia, a simple and high performance framework for implementing state-machine replication (SMR) within a datacenter. The main innovation of Rabia is in using *randomization* to simplify the design. Rabia provides the following two features: (i) It does not need any fail-over protocol

Keywords: SMR, Consensus, Formal Verification

ACM Reference Format:
Haochen Pan, Jesse Tuglu, Neo Zhou, Tianshu Wang, Yicheng Shen, Xiong Zheng, Joseph Tassarotti, Lewis Tseng, and Roberto Palmieri. 2021. Rabia: Simplifying State-Machine Replication Through Randomization. In *ACM SIGOPS 28th Symposium on Operating Systems*


Simplification Time
Translation Time
Solver Time
Ivy Runtime
Normalised time
3.0
2.5
2.0
1.5
1.0
0.5
0.0
10.79
0.27
12.98
0.36
0.80
1.28
1.53
0.37
1.77
0.25
3.15
0.43
4.79
*
156.88
49.25
8.47
1.53
0.41
0.18
0.67
1.29
7.91
4.05
7.20
0.32
6.46
0.24
2.55
0.21
18.95
*
Blockchain
ChordRingMaintenance
DecentralizedLock
MultiSigAll
MultiSigMajority
PaxosEPR
PaxosFirstOrder
Rabia
ReliableBroadcast
RicartAgrawala
Ring
SCP
SuzukiKasami
SuzukiKasamiInts
TwoPhaseCommit
VerticalPaxosFirstOrder

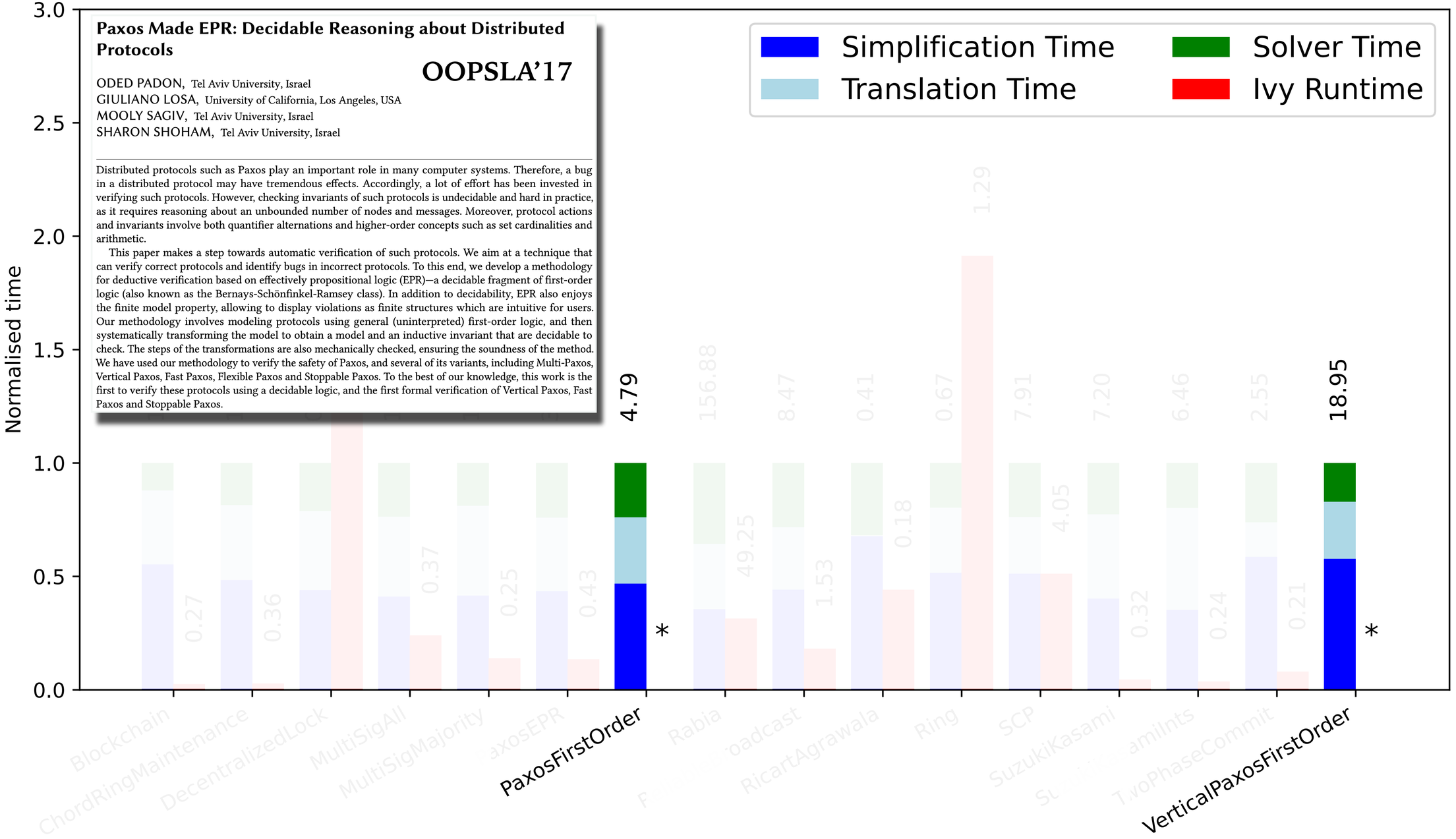
Paxos Made EPR: Decidable Reasoning about Distributed Protocols

OOPSLA'17

ODED PADON, Tel Aviv University, Israel
GIULIANO LOSA, University of California, Los Angeles, USA
MOOLY SAGIV, Tel Aviv University, Israel
SHARON SHOHAM, Tel Aviv University, Israel

Distributed protocols such as Paxos play an important role in many computer systems. Therefore, a bug in a distributed protocol may have tremendous effects. Accordingly, a lot of effort has been invested in verifying such protocols. However, checking invariants of such protocols is undecidable and hard in practice, as it requires reasoning about an unbounded number of nodes and messages. Moreover, protocol actions and invariants involve both quantifier alternations and higher-order concepts such as set cardinalities and arithmetic.

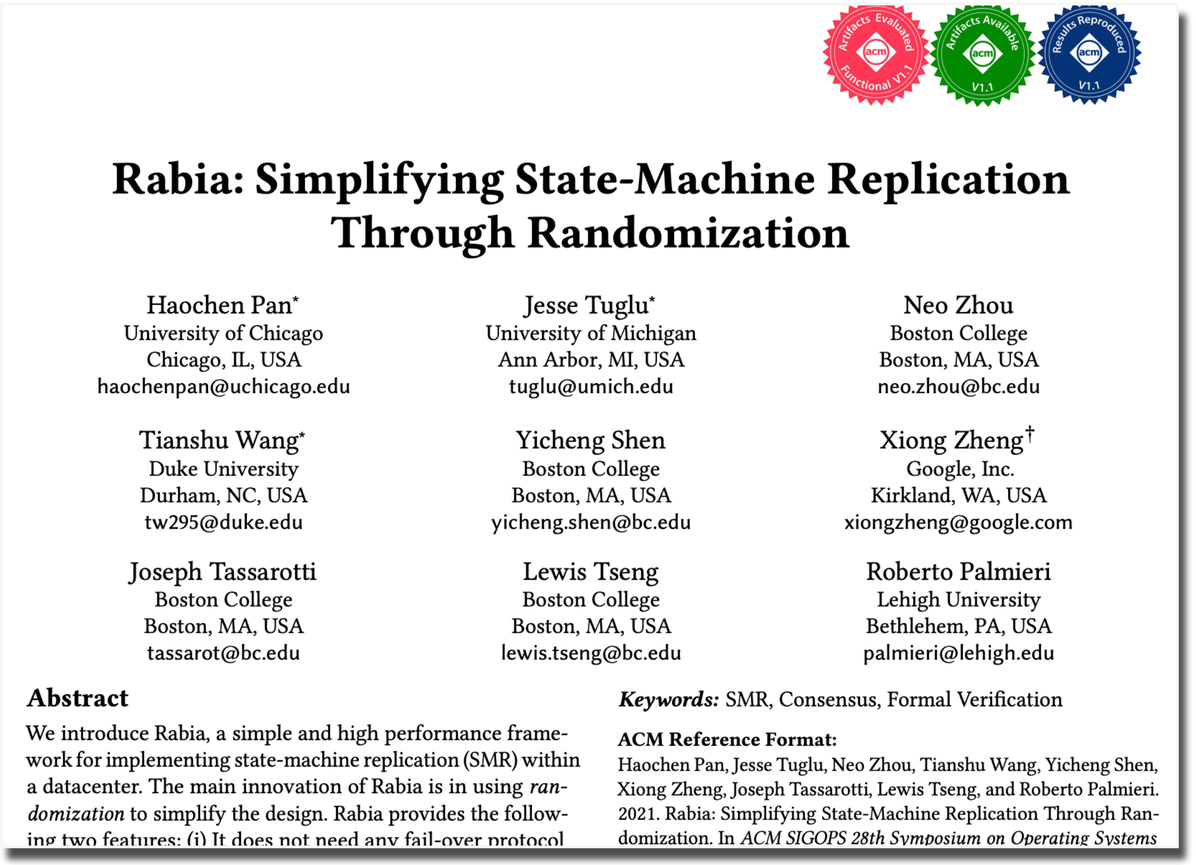
This paper makes a step towards automatic verification of such protocols. We aim at a technique that can verify correct protocols and identify bugs in incorrect protocols. To this end, we develop a methodology for deductive verification based on effectively propositional logic (EPR)—a decidable fragment of first-order logic (also known as the Bernays-Schönfinkel-Ramsey class). In addition to decidability, EPR also enjoys the finite model property, allowing to display violations as finite structures which are intuitive for users. Our methodology involves modeling protocols using general (uninterpreted) first-order logic, and then systematically transforming the model to obtain a model and an inductive invariant that are decidable to check. The steps of the transformations are also mechanically checked, ensuring the soundness of the method. We have used our methodology to verify the safety of Paxos, and several of its variants, including Multi-Paxos, Vertical Paxos, Fast Paxos, Flexible Paxos and Stoppable Paxos. To the best of our knowledge, this work is the first to verify these protocols using a decidable logic, and the first formal verification of Vertical Paxos, Fast Paxos and Stoppable Paxos.



On the Virtues of Multi-Modal Verification

Case Study: The Rabia Protocol

- Original formalisation: Ivy + Rocq
- **Ivy** for protocol modelling and automated invariant checking
- **Rocq** for interactive proofs for some additional invariants



Rabia: Simplifying State-Machine Replication Through Randomization

Haochen Pan*
University of Chicago
Chicago, IL, USA
haochenpan@uchicago.edu

Jesse Tuglu*
University of Michigan
Ann Arbor, MI, USA
tuglu@umich.edu

Neo Zhou
Boston College
Boston, MA, USA
neo.zhou@bc.edu

Tianshu Wang*
Duke University
Durham, NC, USA
tw295@duke.edu

Yicheng Shen
Boston College
Boston, MA, USA
yicheng.shen@bc.edu

Xiong Zheng†
Google, Inc.
Kirkland, WA, USA
xiongzheng@google.com

Joseph Tassarotti
Boston College
Boston, MA, USA
tassarot@bc.edu

Lewis Tseng
Boston College
Boston, MA, USA
lewis.tseng@bc.edu

Roberto Palmieri
Lehigh University
Bethlehem, PA, USA
palmieri@lehigh.edu

Abstract

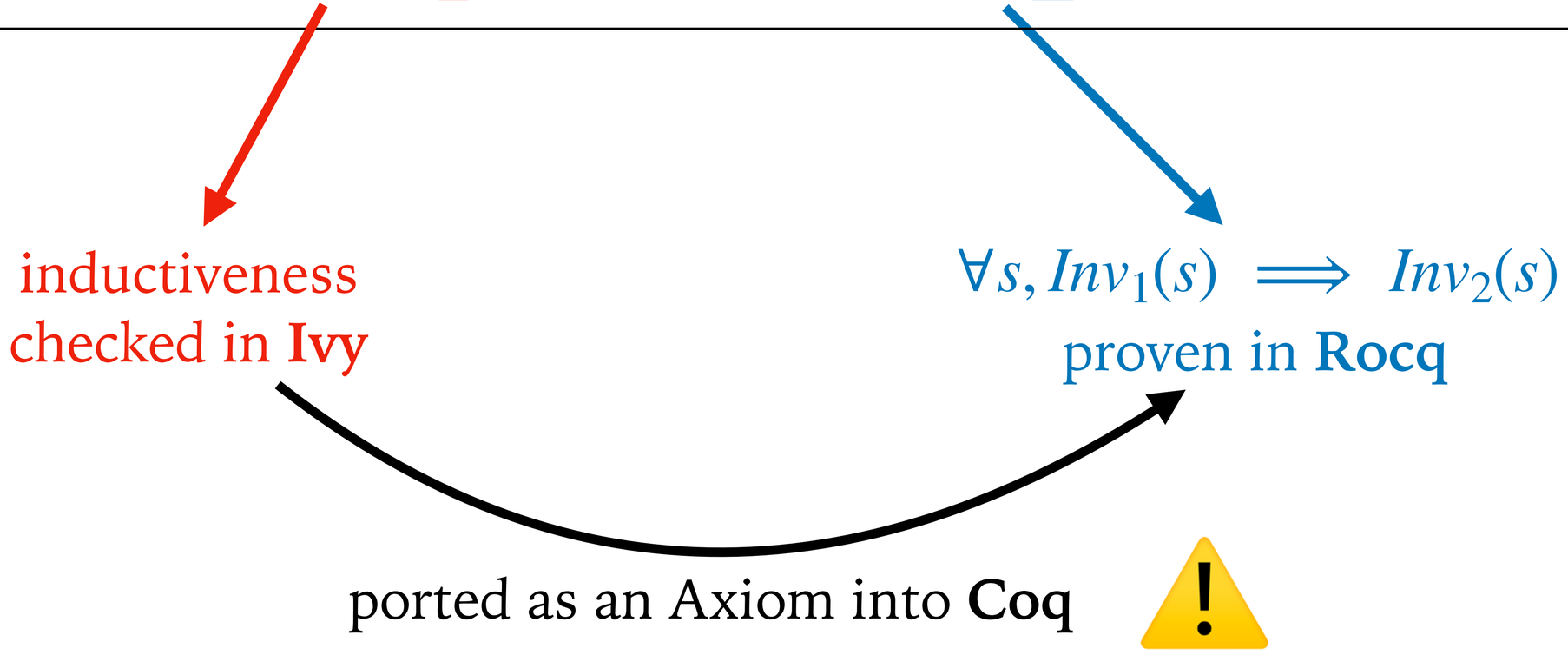
We introduce Rabia, a simple and high performance framework for implementing state-machine replication (SMR) within a datacenter. The main innovation of Rabia is in using *randomization* to simplify the design. Rabia provides the following two features: (i) It does not need any fail-over protocol

Keywords: SMR, Consensus, Formal Verification

ACM Reference Format:
Haochen Pan, Jesse Tuglu, Neo Zhou, Tianshu Wang, Yicheng Shen, Xiong Zheng, Joseph Tassarotti, Lewis Tseng, and Roberto Palmieri. 2021. Rabia: Simplifying State-Machine Replication Through Randomization. In *ACM SIGOPS 28th Symposium on Operating Systems*

Case Study: The Rabia Protocol

Claim: $Inv_1 \wedge Inv_2$ is an invariant.



Case Study: The Rabia Protocol

- Moving facts between the provers introduced a discrepancy

Ivy

```
specification {
  conjecture [started_pred] (ind_defs.started(Psucc) & succ(P, Psucc) -> ind_defs.started(Psucc))
}
```

Rocq

```
Definition started_pred σ :=
  (∀ P, started σ (succ P) → started σ P).
Axiom started_pred_invariant : invariant started_pred.
```

- In **Veil**, when proving Inv_2 , the proof of Inv_1 (from `#check_invariants`) is reused; this allowed us to catch and fix this discrepancy.

Reflection

- Different approaches for building assurance in distributed protocols in one unified framework, embedded in Lean 4
- Veil is still immature, but we are working to make it the best verifier for distributed protocols. We are constantly shipping improvements and we plan to add:
  - better support for non-FOL specifications
  - integration with *grind* and other proof automation
  - liveness properties
  - a module system, enabling compositional testing & proving of protocols
  - support for refinement testing and refinement proofs
- Vision: Veil as one-stop shop for all your protocol reasoning needs

To Take Away

- **Veil** is a Lean library for *automated/interactive* verification of distributed protocols
- Combines *symbolic* proofs and TLC-style *concrete-state* model checker
- *Foundational*: different VC generators are proven sound wrt. concrete semantics
- Acceptable performance for FOL, *seamless integration* with HO specifications

veil.dev proofsandintuitions.net

Veil: A Framework for Automated and Interactive Verification of Transition Systems, CAV'25

Lessons from Building an Auto-Active Verifier in Lean, Dafny'26

Thanks!