21st Formal Methods in Computer-Aided Design
FMCAD 2021

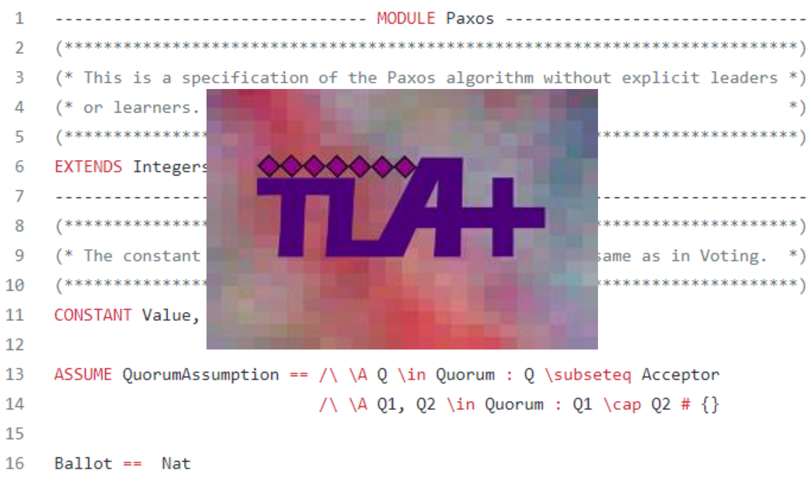
Towards an Automatic Proof of Lamport’s Paxos

Aman Goel and Karem Sakallah

University of Michigan, Ann Arbor

amangoel@umich.edu

Distributed Protocol ≡ Architectural Blueprint

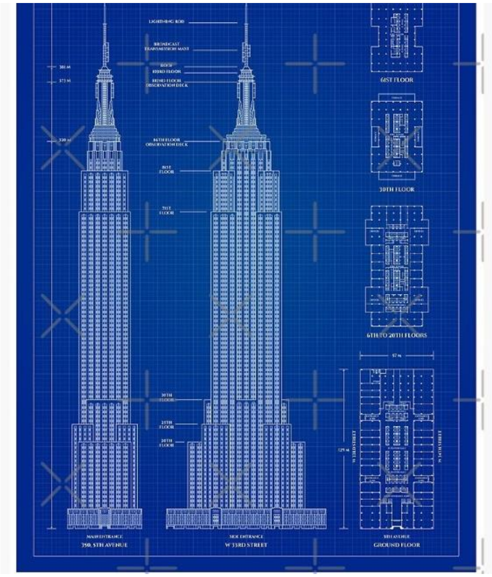


```
------------------------------- MODULE Paxos -------------------------------
(***************************************************************************)
(* This is a specification of the Paxos algorithm without explicit leaders *)
(* or learners.                                                           *)
(*************                                         ********************)
EXTENDS Integers
-------------                                         ---------------------
(*************                                         ********************)
(* The constant                                   same as in Voting.   *)
(*************                                         ********************)
CONSTANT Value,

ASSUME QuorumAssumption == /\ \A Q \in Quorum : Q \subseteq Acceptor
                           /\ \A Q1, Q2 \in Quorum : Q1 \cap Q2 # {}

Ballot ==  Nat
```

Amazon EC2

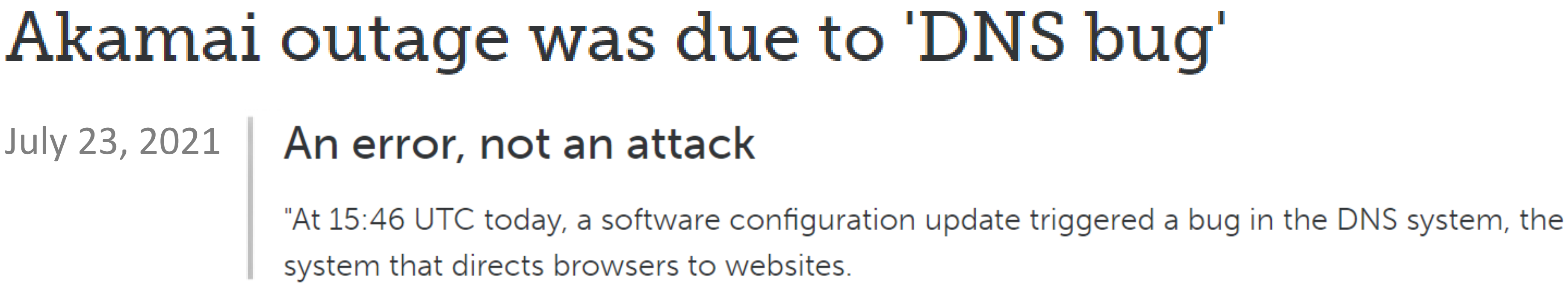
Google Cloud Spanner

Apache CASSANDRA



Why Verify?

DatacenterDynamics



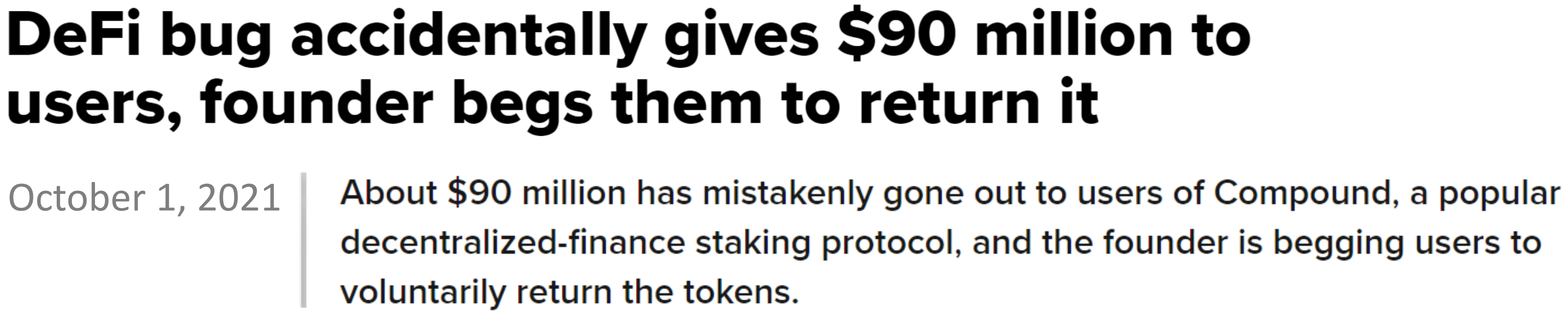
Akamai outage was due to 'DNS bug'

July 23, 2021

An error, not an attack

"At 15:46 UTC today, a software configuration update triggered a bug in the DNS system, the system that directs browsers to websites.

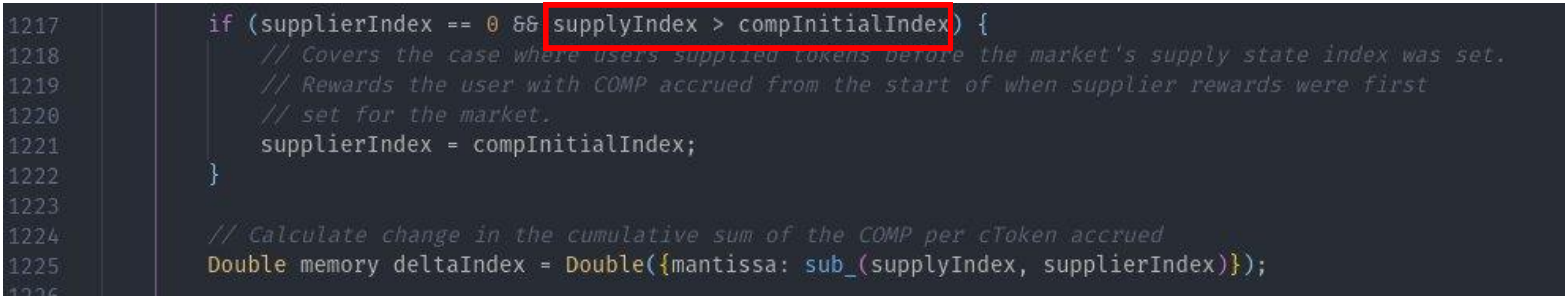
CNBC



DeFi bug accidentally gives $90 million to users, founder begs them to return it

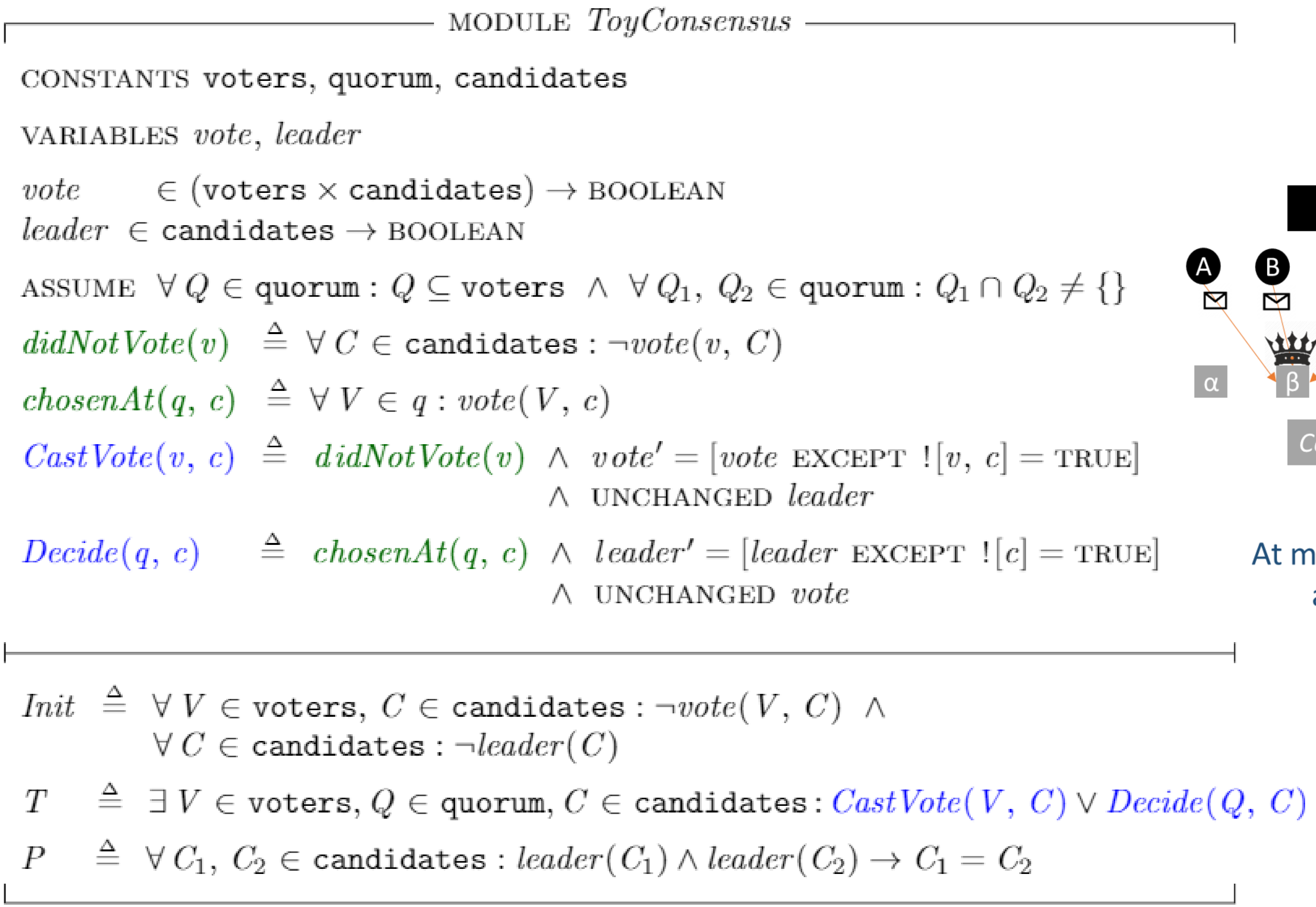
October 1, 2021

About $90 million has mistakenly gone out to users of Compound, a popular decentralized-finance staking protocol, and the founder is begging users to voluntarily return the tokens.



```
if (supplierIndex == 0 && supplyIndex > compInitialIndex) {
    // Covers the case where users supplied tokens before the market's supply state index was set.
    // Rewards the user with COMP accrued from the start of when supplier rewards were first
    // set for the market.
    supplierIndex = compInitialIndex;
}

// Calculate change in the cumulative sum of the COMP per cToken accrued
Double memory deltaIndex = Double({mantissa: sub_(supplyIndex, supplierIndex)});
```

ToyConsensus Protocol[1] in TLA+



MODULE *ToyConsensus*

Domains

$\text{CONSTANTS } \texttt{voters}, \texttt{quorum}, \texttt{candidates}$

State variables

$\text{VARIABLES } vote, leader$

$vote \in (\texttt{voters} \times \texttt{candidates}) \rightarrow \text{BOOLEAN}$

$leader \in \texttt{candidates} \rightarrow \text{BOOLEAN}$

Global axiom

$\text{ASSUME } \forall Q \in \texttt{quorum} : Q \subseteq \texttt{voters} \land \forall Q_1, Q_2 \in \texttt{quorum} : Q_1 \cap Q_2 \neq \{\}$

Definitions

$didNotVote(v) \triangleq \forall C \in \texttt{candidates} : \neg vote(v, C)$

$chosenAt(q, c) \triangleq \forall V \in q : vote(V, c)$

Actions

$CastVote(v, c) \triangleq didNotVote(v) \land vote' = [vote \text{ EXCEPT } ![v, c] = \text{TRUE}]$
$\land \text{ UNCHANGED } leader$

$Decide(q, c) \triangleq chosenAt(q, c) \land leader' = [leader \text{ EXCEPT } ![c] = \text{TRUE}]$
$\land \text{ UNCHANGED } vote$

Initial States

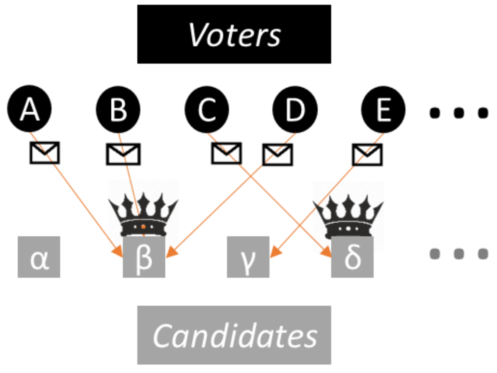
$Init \triangleq \forall V \in \texttt{voters}, C \in \texttt{candidates} : \neg vote(V, C) \land$
$\forall C \in \texttt{candidates} : \neg leader(C)$

Transition Relation

$T \triangleq \exists V \in \texttt{voters}, Q \in \texttt{quorum}, C \in \texttt{candidates} : CastVote(V, C) \lor Decide(Q, C)$

Safety Property

$P \triangleq \forall C_1, C_2 \in \texttt{candidates} : leader(C_1) \land leader(C_2) \rightarrow C_1 = C_2$

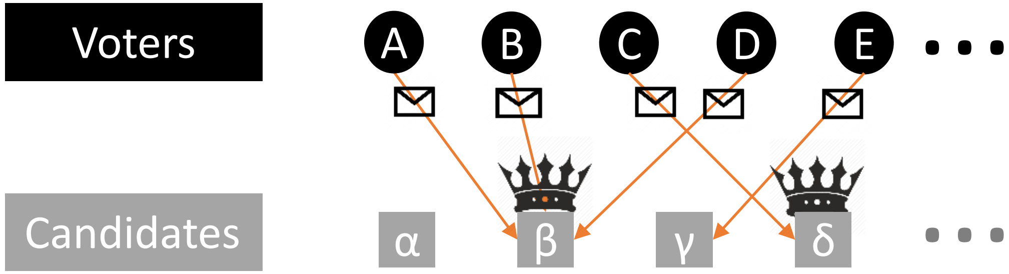


❌

At most one leader at all times

[1] Ken McMillan, "Toy Consensus in the Ivy language." *https://github.com/microsoft/ivy/blob/master/examples/ivy/toy_consensus.ivy*

Verifying Distributed Protocols



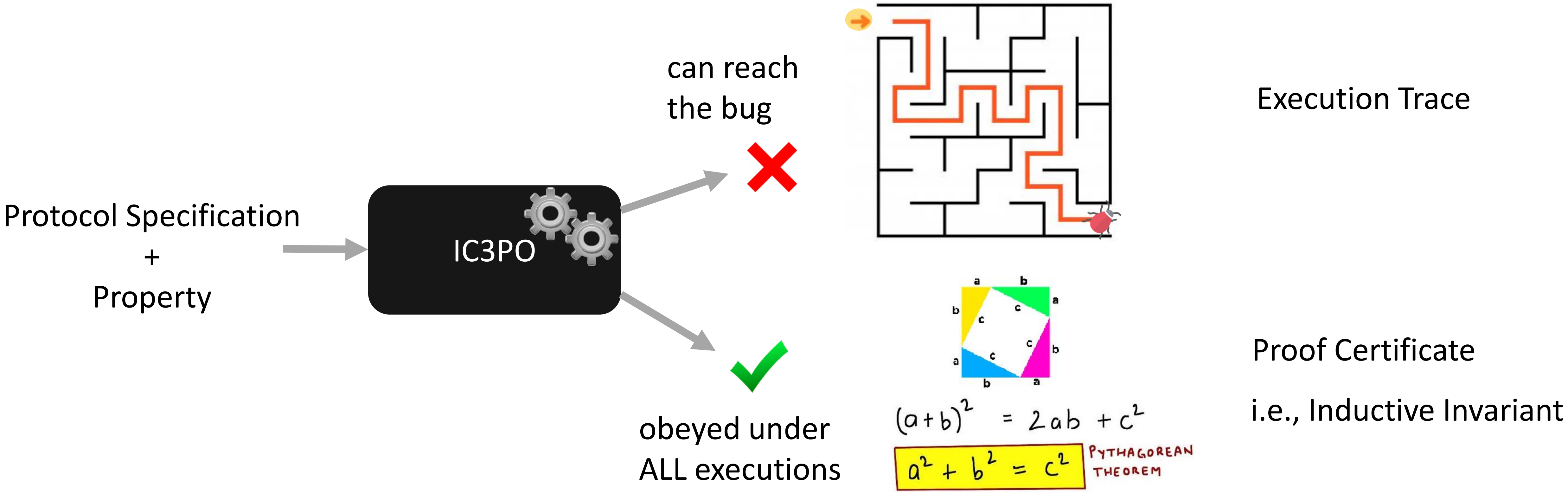
❌

At most one leader
at all times

Challenges 😢

- **Infinite** State Space
- Reasoning is **Hard/Undecidable**
- **Not** Scalable

IC3PO: IC3 for Proving Protocol Properties



Protocol Specification
+
Property

IC3PO

can reach the bug ✗

Execution Trace

✓ obeyed under ALL executions

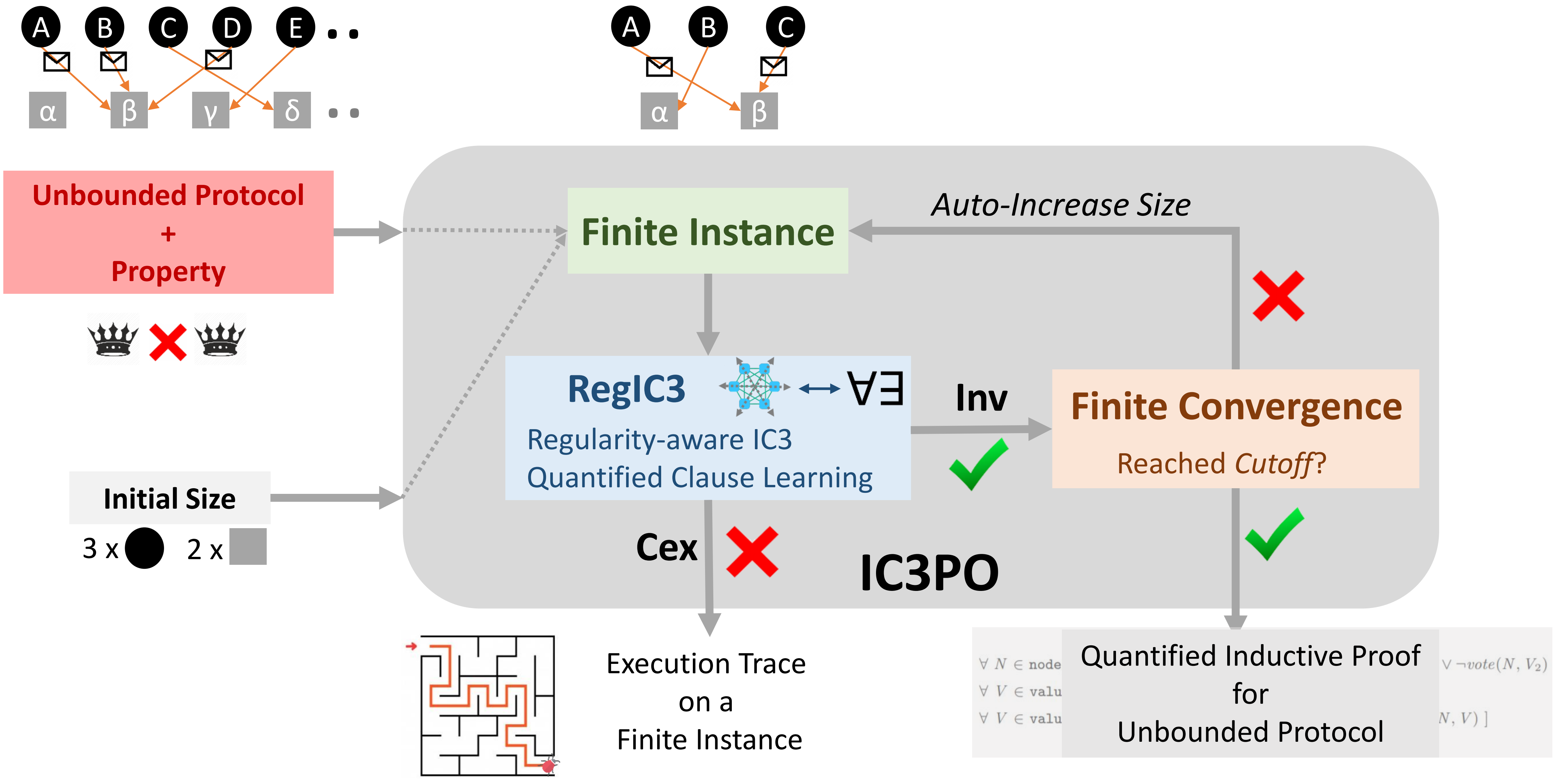
Proof Certificate

i.e., Inductive Invariant

IC3PO’s Key Ingredients

Finite-Domain Model Checking	 Leslie Lamport <tlaplus.ll@gmail.com>: Apr 15 09:45AM -0700 While large sets can cause performance problems, it's rare for an algorithm to be correct for a set of 3 elements and not for a set of 1000 elements.
Spatial Regularity	*Symmetry Boosting* using Protocol’s Domain Symmetries
Temporal Regularity	*Range Boosting* over Totally-ordered Domains
Regularity ↔ Quantification	*Compact Quantified* Clause Learning
Hierarchical Structure	*Hierarchical Strengthening* for High Scalability

IC3PO: IC3 for Proving Protocol Properties



Unbounded Protocol + Property

Initial Size

Finite Instance

Auto-Increase Size

RegIC3 ⟷ ∀∃

Regularity-aware IC3

Quantified Clause Learning

Inv ✓

Finite Convergence

Reached *Cutoff*?

Cex ✗

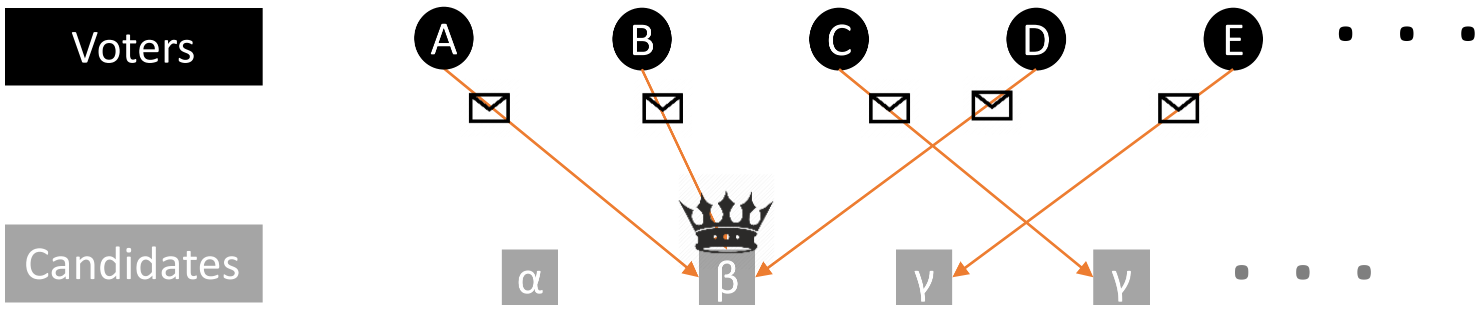
IC3PO

Execution Trace on a Finite Instance

Quantified Inductive Proof for Unbounded Protocol

Finite-Domain Model Checking

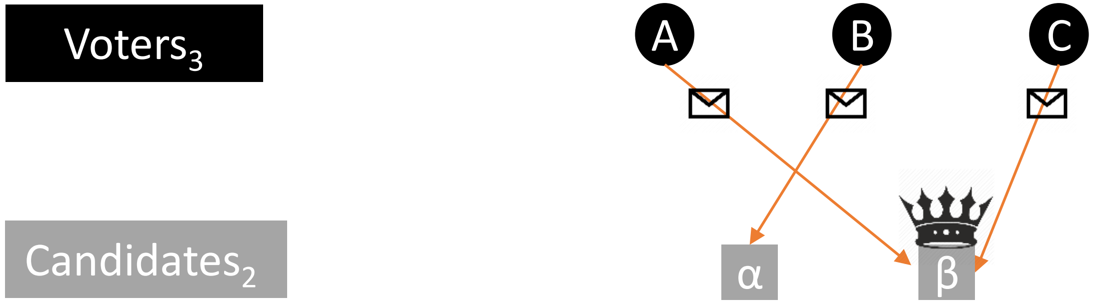
Unbounded Protocol



State-space size = unbounded

3 x ● 2 x ■

Finite Instance



State-space size = 2^8

Challenges 😢

Infinite State Space

Reasoning is **Hard/Undecidable**

Not Scalable

Benefits 😊

Finite State Space

Always **Decidable**

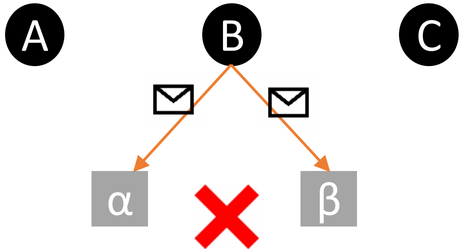
Fast reasoning with SMT solvers

Symmetry Boosting for Symmetric Domains

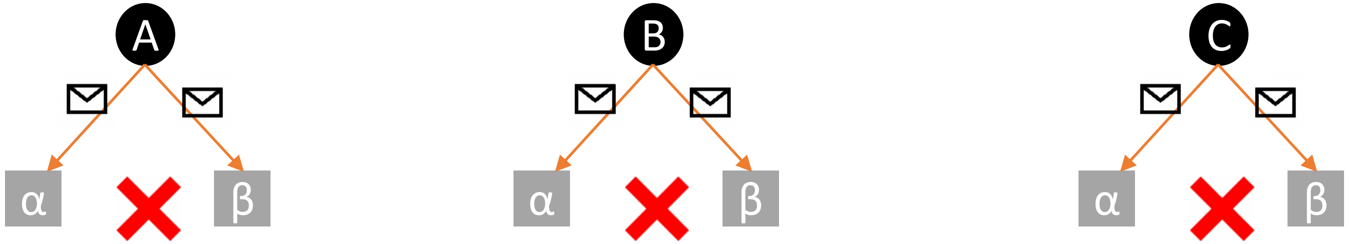
Finite Instance

Voters$_3$

Candidates$_2$



All *voters* are ***symmetrically-equivalent***

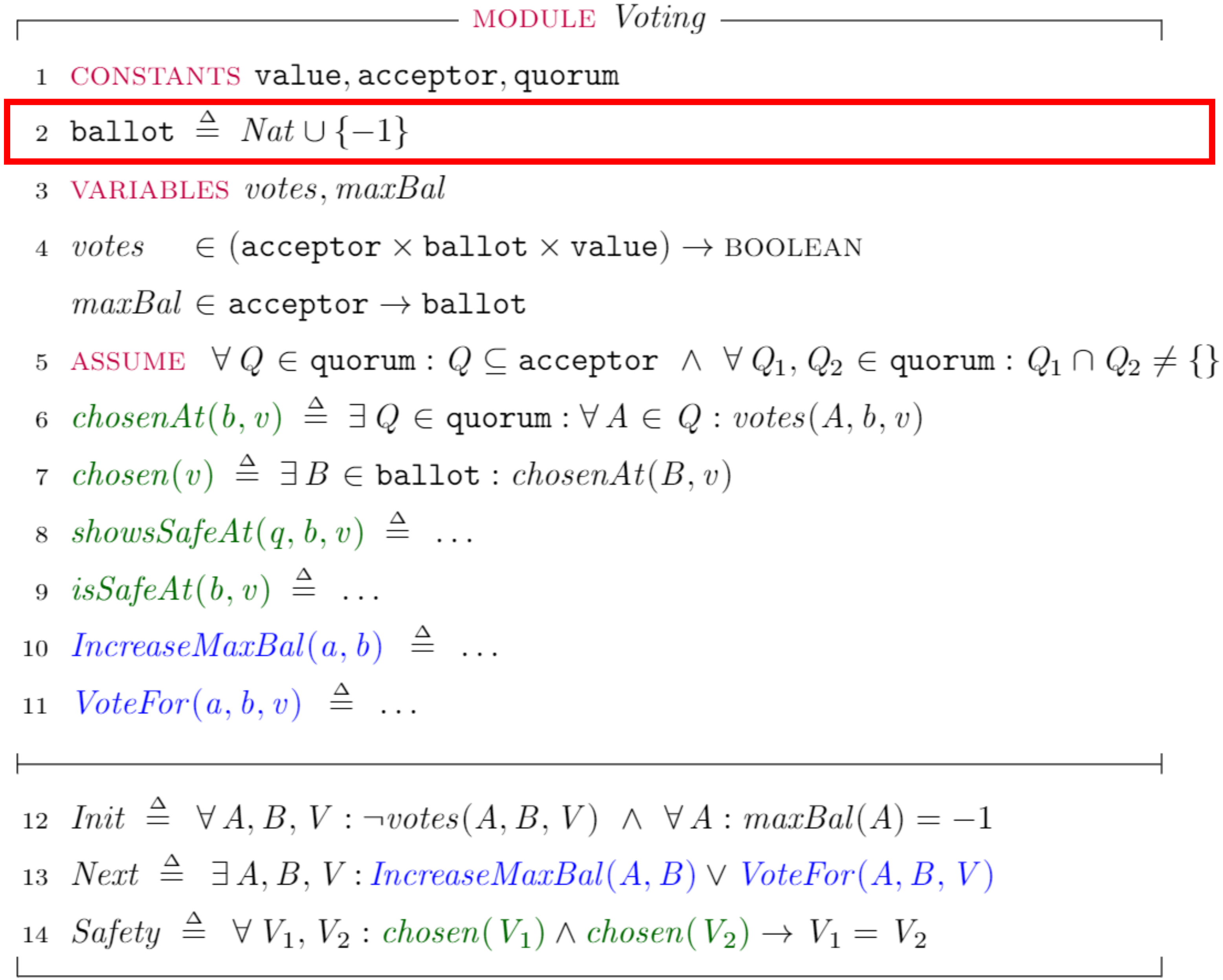


- All domain elements can be **permuted arbitrarily**
- Learn all *symmetrically-equivalent* clauses **without any additional reasoning**
- Compact **quantified clauses**

Relating Symmetry with Quantification

Form	Clause	Boosted Clause
$\forall$	$\text{clause}_1 = \neg \mathit{vote}(A, \alpha) \vee \neg \mathit{vote}(A, \beta)$	$\text{Quantified}(\text{clause}_1) = \forall\, X \in \text{Voters}_3\text{: } \neg \mathit{vote}(X, \alpha) \vee \neg \mathit{vote}(X, \beta)$
$\exists$	$\text{clause}_2 = \mathit{vote}(A, \alpha) \vee \mathit{vote}(B, \alpha) \vee \mathit{vote}(C, \alpha)$	$\text{Quantified}(\text{clause}_2) = \exists\, Y \in \text{Voters}_3\text{: } \mathit{vote}(Y, \alpha)$
$\forall\ \exists$	$\text{clause}_3 = \neg \mathit{vote}(A, \alpha) \vee \mathit{vote}(B, \alpha) \vee \mathit{vote}(C, \alpha)$	$\text{Quantified}(\text{clause}_2) = \forall\, X \in \text{Voters}_3\text{: } \exists\, Y \in \text{Voters}_3\text{:}$ $\neg \mathit{vote}(X, \alpha) \vee [\ (X \neq Y) \wedge \mathit{vote}(Y, \alpha)\ ]$

Voting Protocol[1] in TLA+

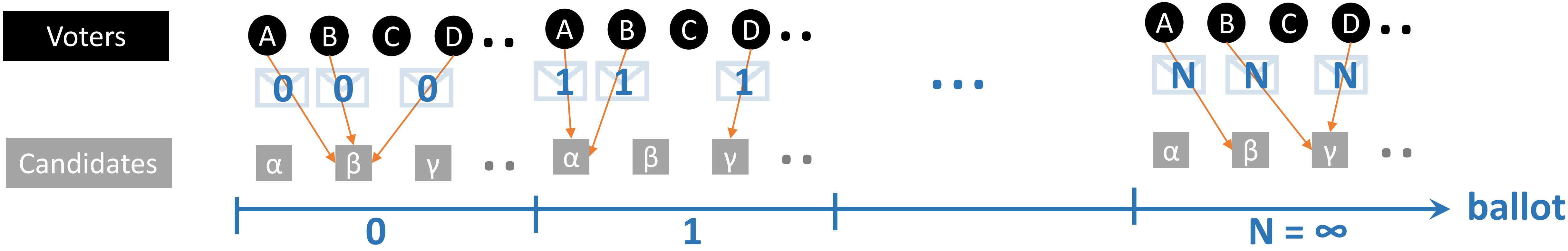


```
──────────────────── MODULE Voting ────────────────────
CONSTANTS value, acceptor, quorum
ballot ≜ Nat ∪ {−1}
VARIABLES votes, maxBal
votes  ∈ (acceptor × ballot × value) → BOOLEAN
maxBal ∈ acceptor → ballot
ASSUME ∀ Q ∈ quorum : Q ⊆ acceptor ∧ ∀ Q1, Q2 ∈ quorum : Q1 ∩ Q2 ≠ {}
chosenAt(b, v) ≜ ∃ Q ∈ quorum : ∀ A ∈ Q : votes(A, b, v)
chosen(v) ≜ ∃ B ∈ ballot : chosenAt(B, v)
showsSafeAt(q, b, v) ≜ ...
isSafeAt(b, v) ≜ ...
IncreaseMaxBal(a, b) ≜ ...
VoteFor(a, b, v) ≜ ...
───────────────────────────────────────────────────────
Init ≜ ∀ A, B, V : ¬votes(A, B, V) ∧ ∀ A : maxBal(A) = −1
Next ≜ ∃ A, B, V : IncreaseMaxBal(A, B) ∨ VoteFor(A, B, V)
Safety ≜ ∀ V1, V2 : chosen(V1) ∧ chosen(V2) → V1 = V2
───────────────────────────────────────────────────────
```

[1] Leslie Lamport, "The Voting protocol." *https://github.com/tlaplus/Examples/blob/master/specifications/PaxosHowToWinATuringAward/Voting.tla*

Totally-Ordered Domains

Unbounded Protocol



Challenges 😢

Cannot be permuted arbitrarily

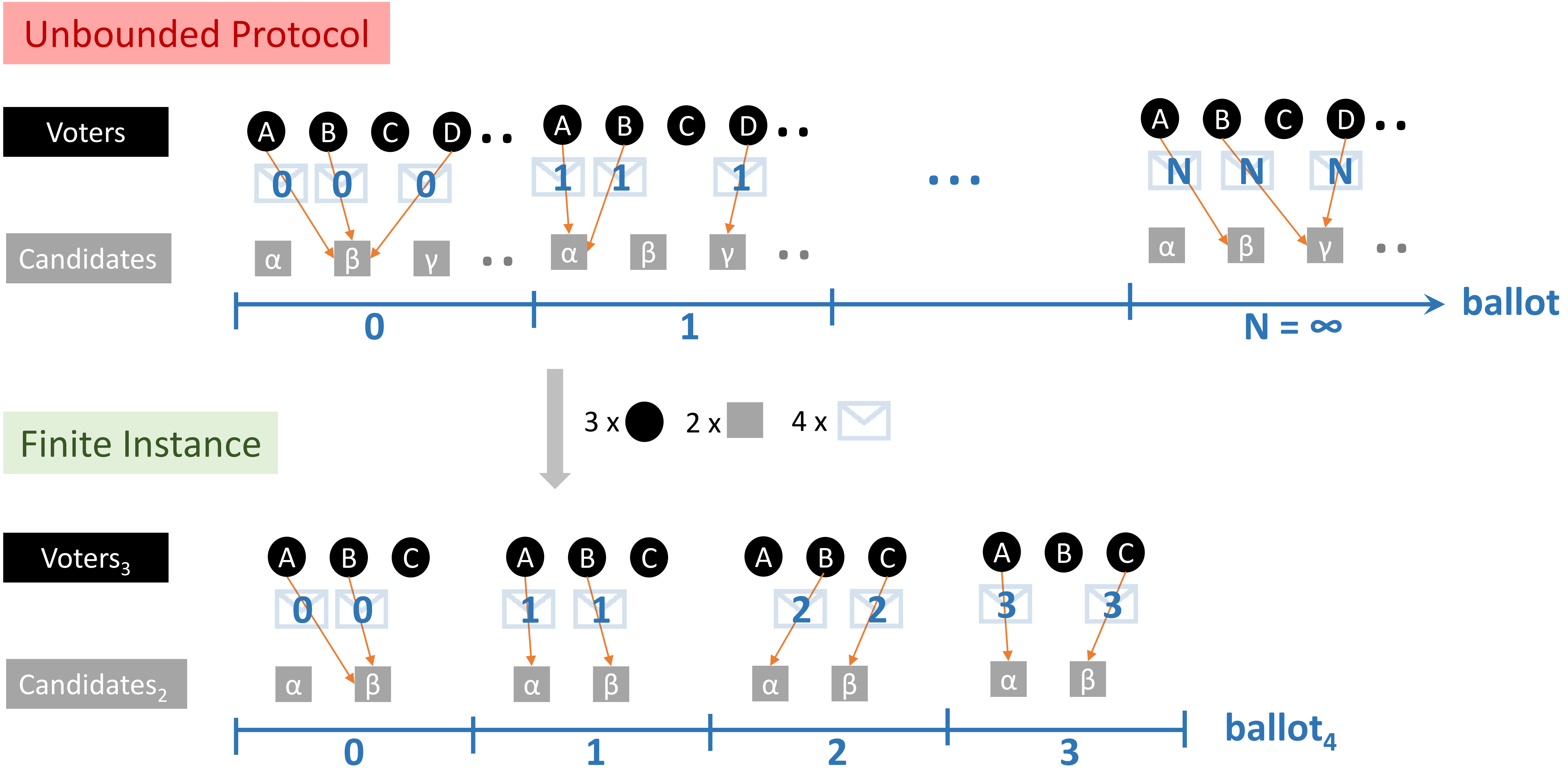
Unsafe combinations due to special elements

Solution 😊

Respect the **total order**

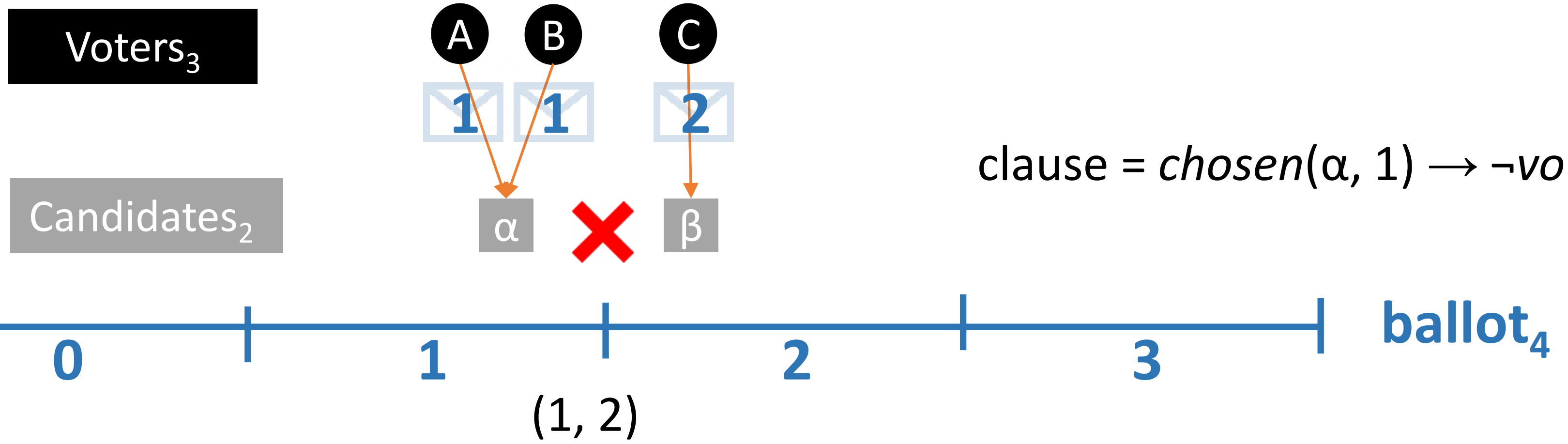
Respect **reachability constraints**

Finite-Domain Model Checking



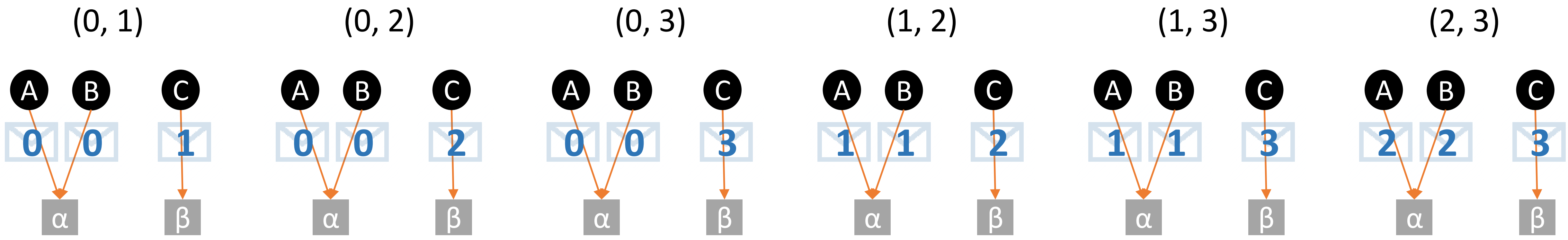
Boosting for Totally-Ordered Domains

Finite Instance

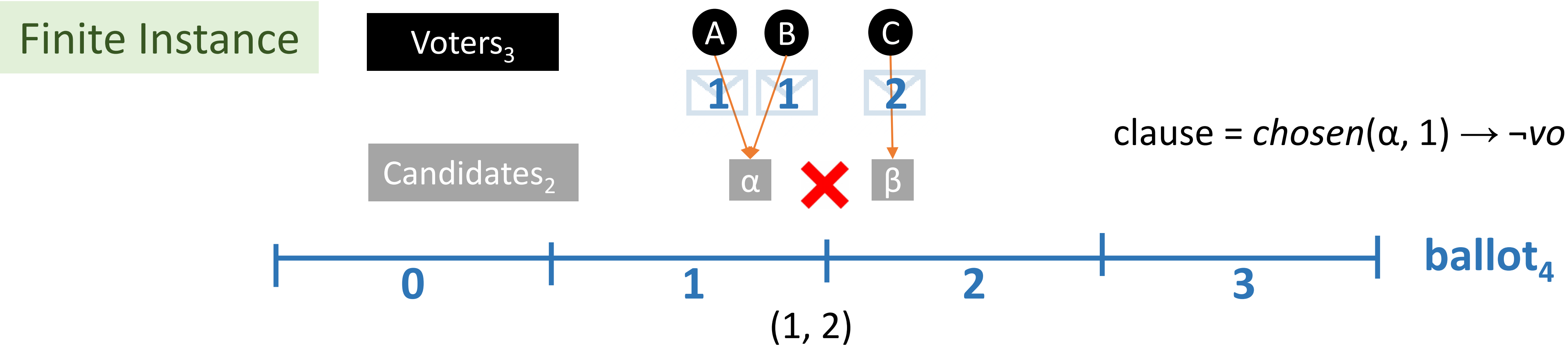


clause = *chosen*(α, 1) ⟶ ¬*vote*(C, β, 2)

Respect the **total order**, i.e., only consider *ordered* permutations

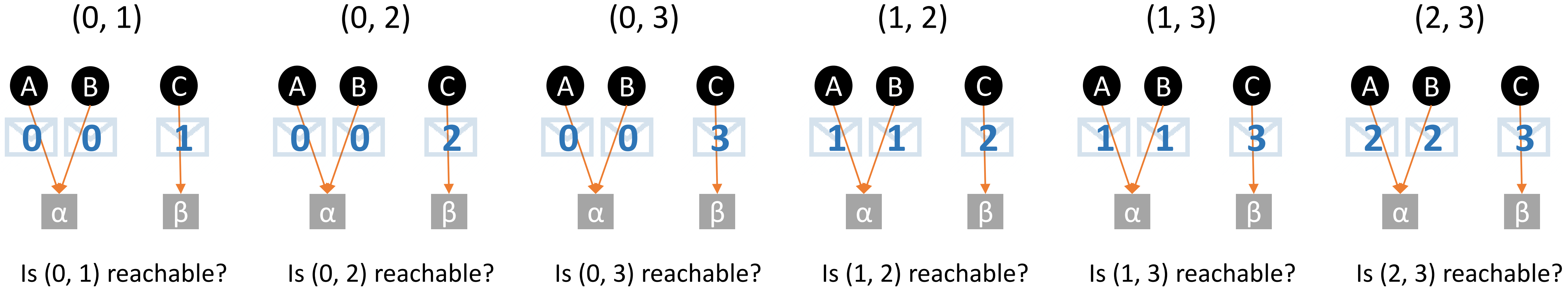


Boosting for Totally-Ordered Domains



clause = *chosen*(α, 1) ⟶ ¬*vote*(C, β, 2)

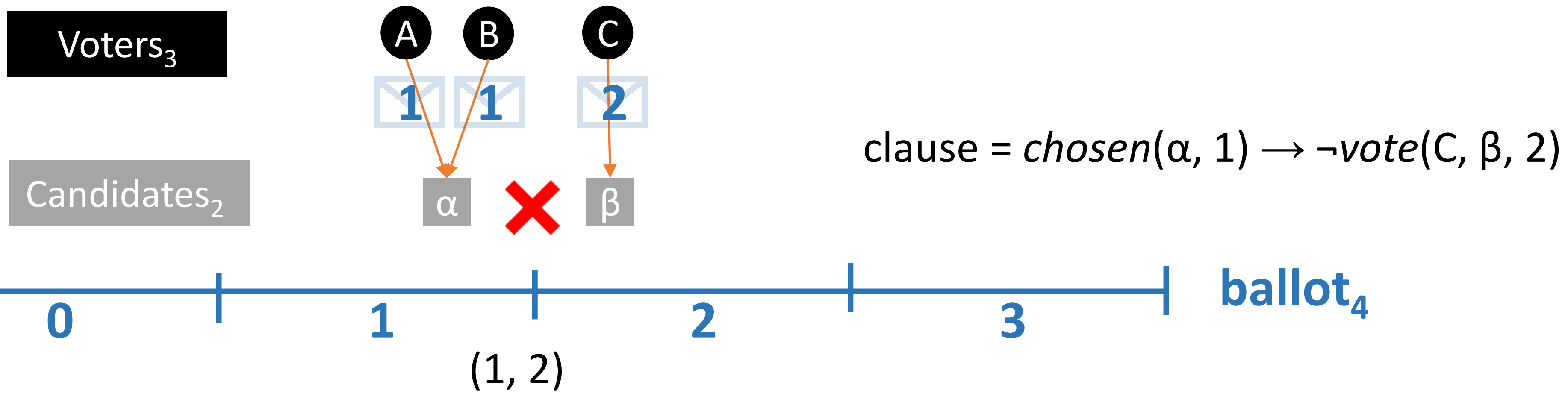
Respect **reachability constraints**, i.e., check unreachability with additional SMT queries



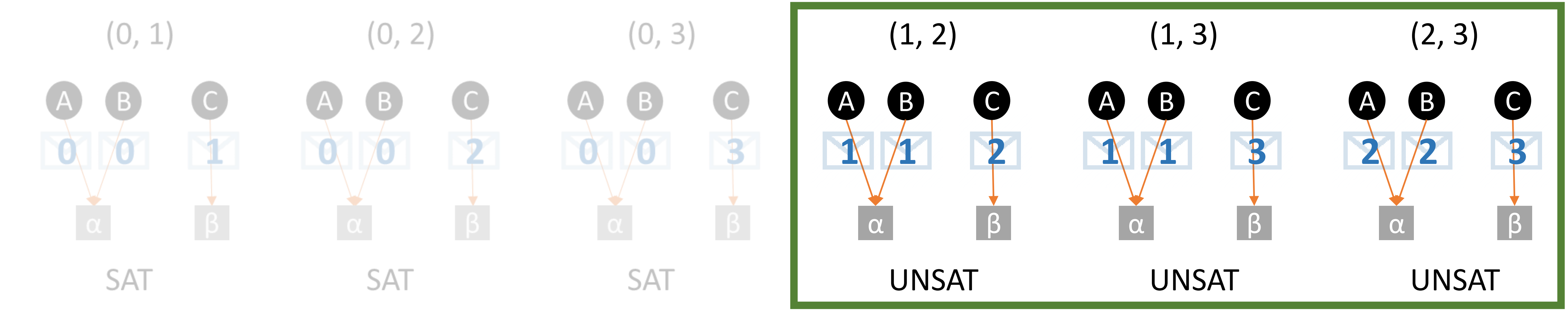
Is (0, 1) reachable? Is (0, 2) reachable? Is (0, 3) reachable? Is (1, 2) reachable? Is (1, 3) reachable? Is (2, 3) reachable?

Range Boosting for Totally-Ordered Domains

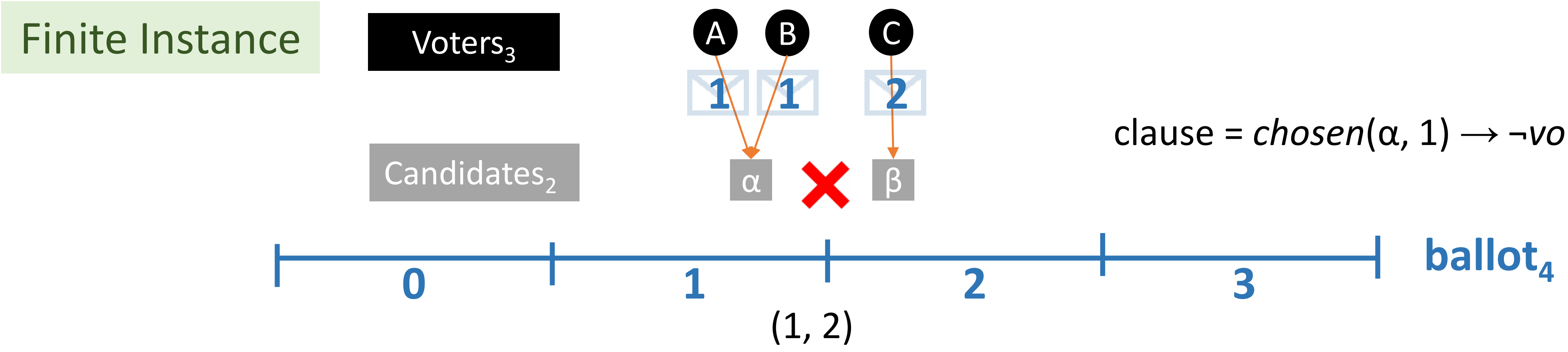
Finite Instance



clause = *chosen*(α, 1) ⟶ ¬*vote*(C, β, 2)



Range Boosting for Totally-Ordered Domains



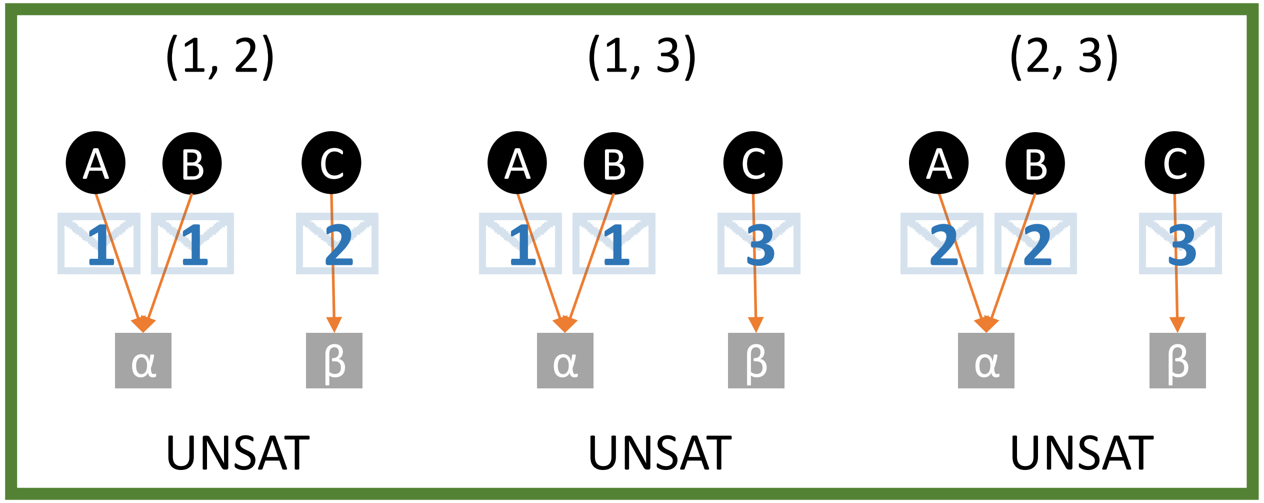
clause = *chosen*(α, 1) ⟶ ¬*vote*(C, β, 2)

Safe Orbit(clause) =

[*chosen*(α, 1) ⟶ ¬*vote*(C, β, 2)] ∧

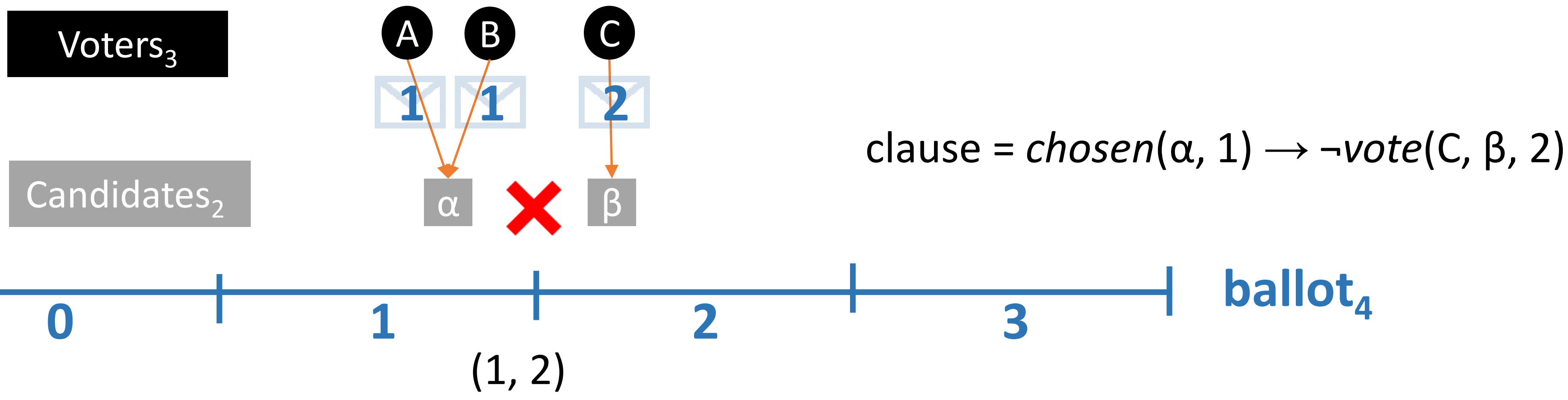
[*chosen*(α, 1) ⟶ ¬*vote*(C, β, 3)] ∧

[*chosen*(α, 2) ⟶ ¬*vote*(C, β, 3)]



Range Boosting for Totally-Ordered Domains

Finite Instance



clause = $\mathit{chosen}(\alpha, 1) \longrightarrow \neg \mathit{vote}(C, \beta, 2)$

Encode unreachable combinations as a **quantified range constraint**

Safe Orbit(clause) =

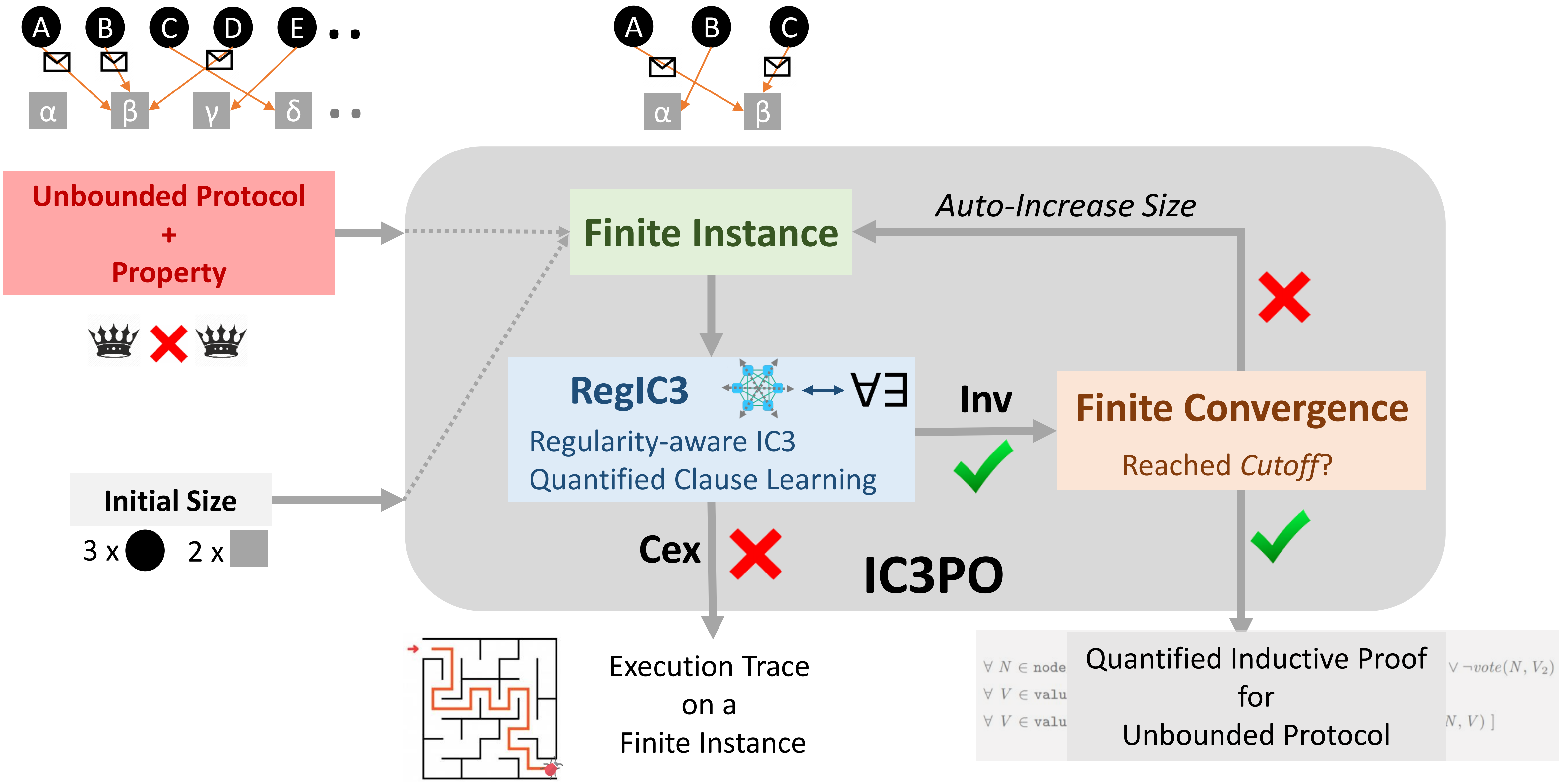
$$[\ \mathit{chosen}(\alpha, 1) \longrightarrow \neg \mathit{vote}(C, \beta, 2)\ ]\ \wedge$$
$$[\ \mathit{chosen}(\alpha, 1) \longrightarrow \neg \mathit{vote}(C, \beta, 3)\ ]\ \wedge$$
$$[\ \mathit{chosen}(\alpha, 2) \longrightarrow \neg \mathit{vote}(C, \beta, 3)\ ]$$

$$\equiv$$

Quantified(clause) =

$$\forall\ X, Y \in \mathrm{ballot}_4 :$$
$$(0 < X < Y) \longrightarrow [\ \mathit{chosen}(\alpha, X) \longrightarrow \neg \mathit{vote}(C, \beta, Y)\ ]$$

IC3PO: IC3 for Proving Protocol Properties



Unbounded Protocol + Property

Initial Size

3 x ● 2 x ■

Finite Instance

Auto-Increase Size

RegIC3 ⟷ ∀∃

Regularity-aware IC3

Quantified Clause Learning

Inv ✓

Finite Convergence

Reached *Cutoff*?

Cex ✗

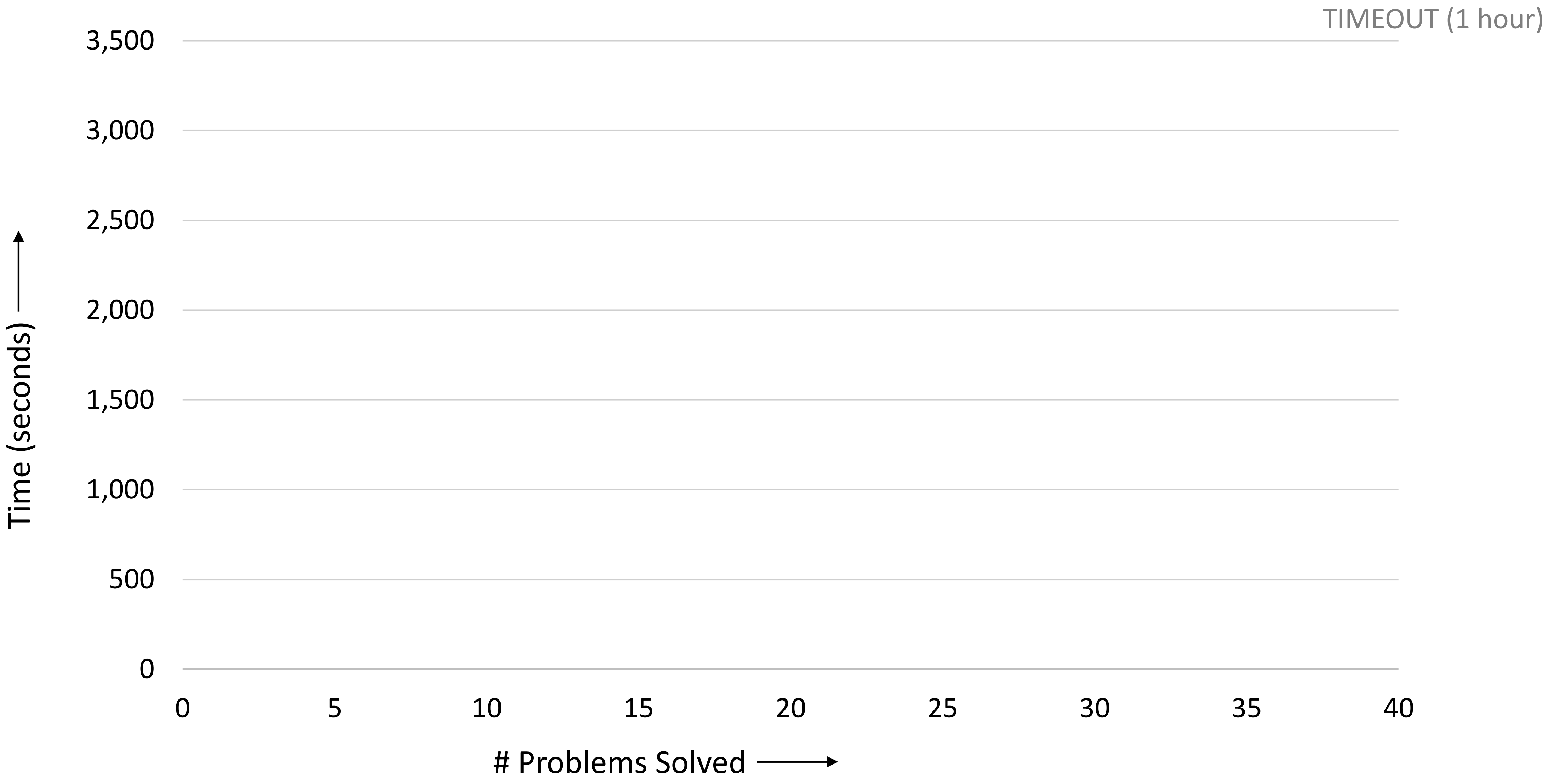
IC3PO

Execution Trace on a Finite Instance

Quantified Inductive Proof for Unbounded Protocol

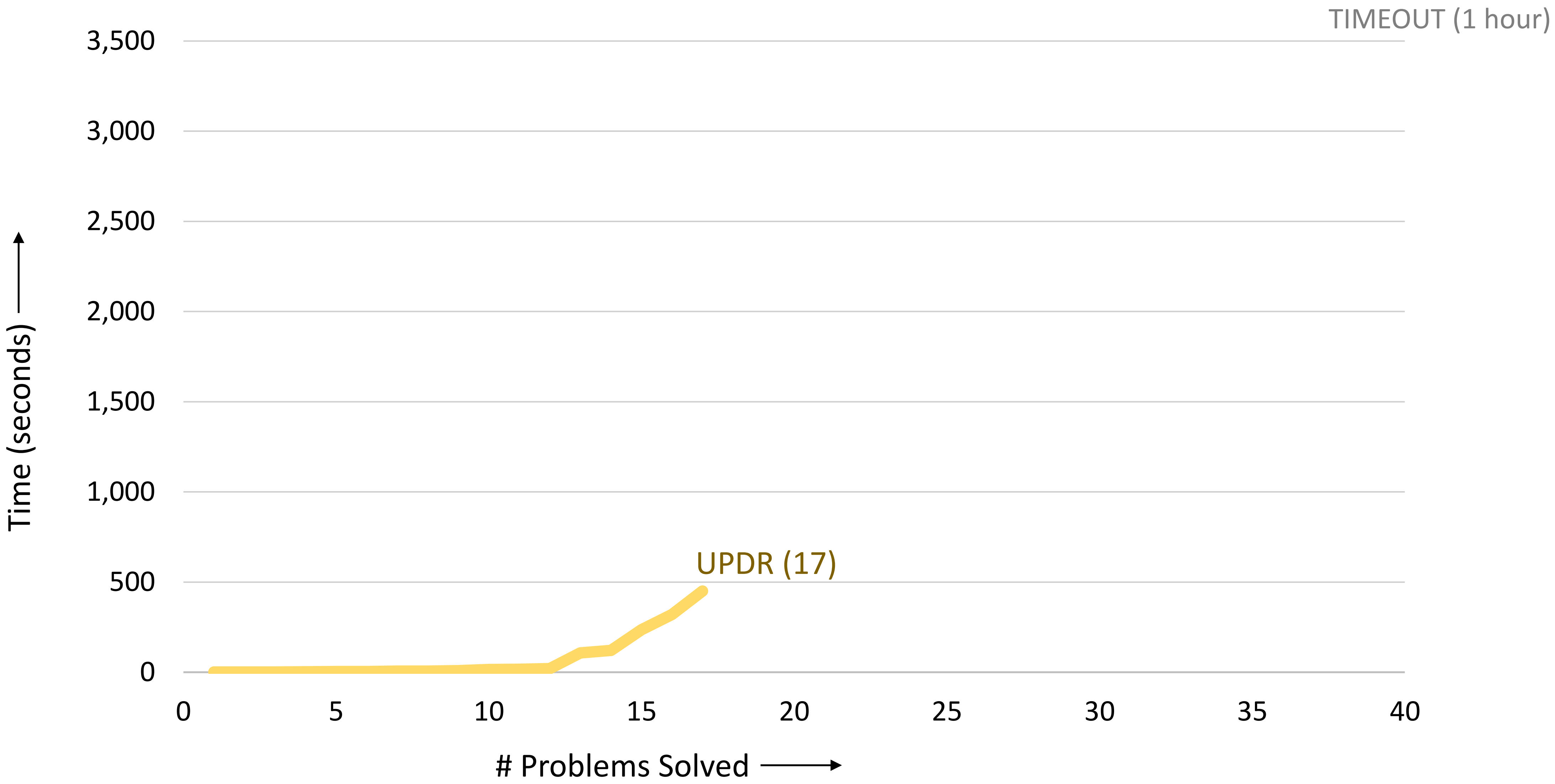
Evaluation

IC3PO	DISTAI	SWISS	FOL-IC3	I4	UPDR
NFM’21	*OSDI’21*	*NSDI’21*	*PLDI’20*	*SOSP’19*	*JACM’17*



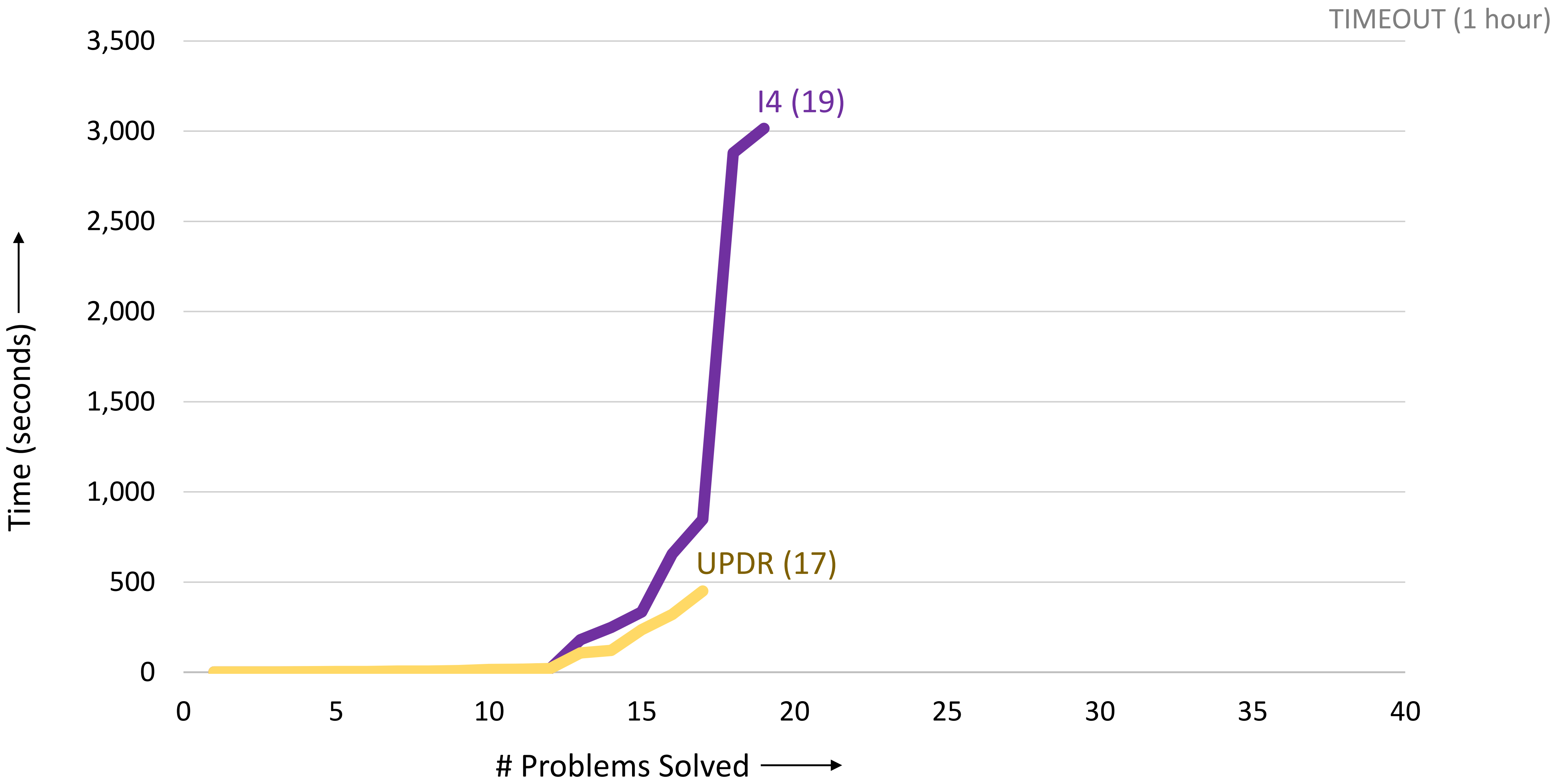
Evaluation

IC3PO	DISTAI	SWISS	FOL-IC3	I4	UPDR
NFM’21	*OSDI’21*	*NSDI’21*	*PLDI’20*	*SOSP’19*	*JACM’17*



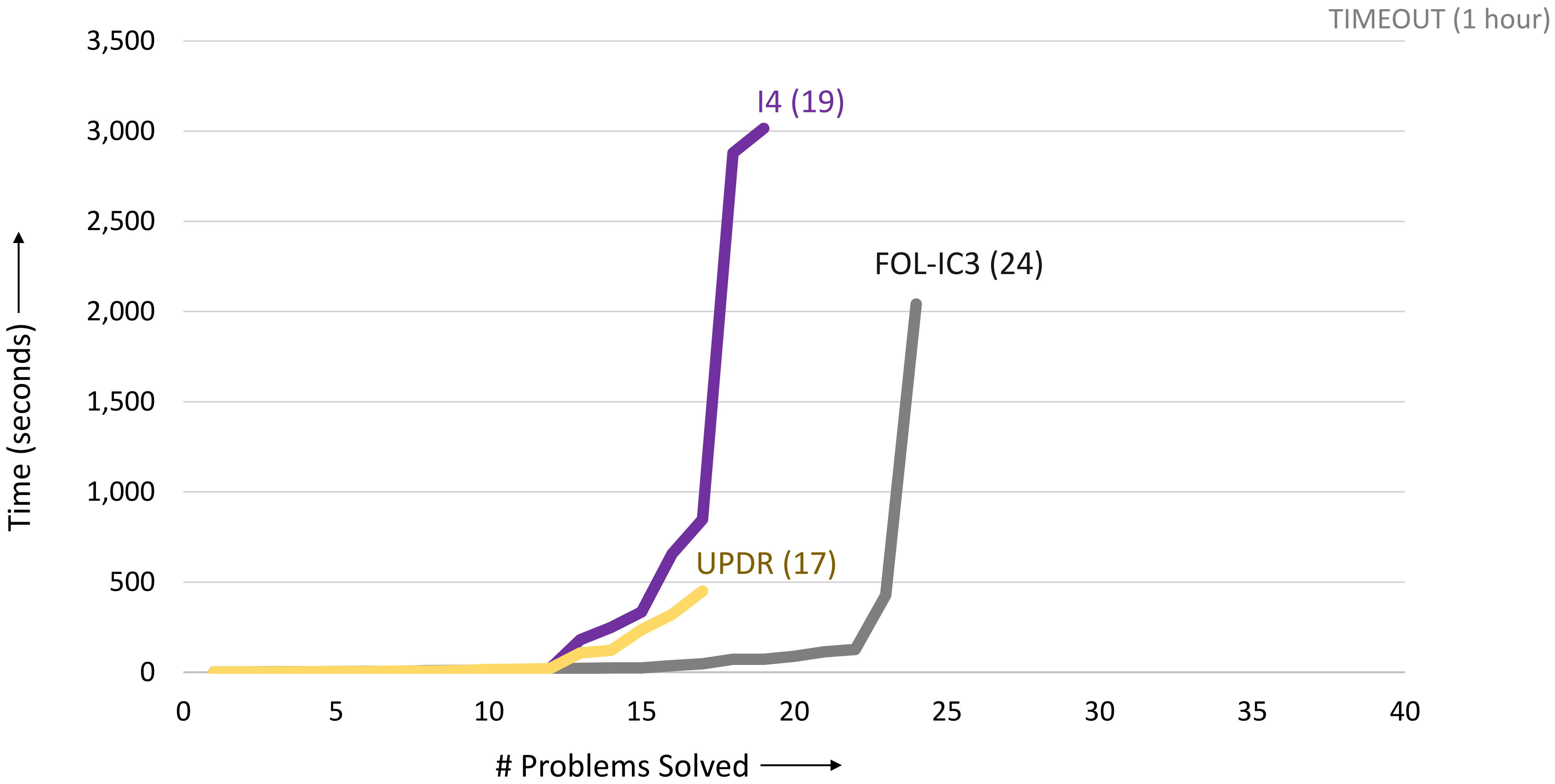
Evaluation

IC3PO	DISTAI	SWISS	FOL-IC3	I4	UPDR
NFM'21	*OSDI'21*	*NSDI'21*	*PLDI'20*	*SOSP'19*	*JACM'17*



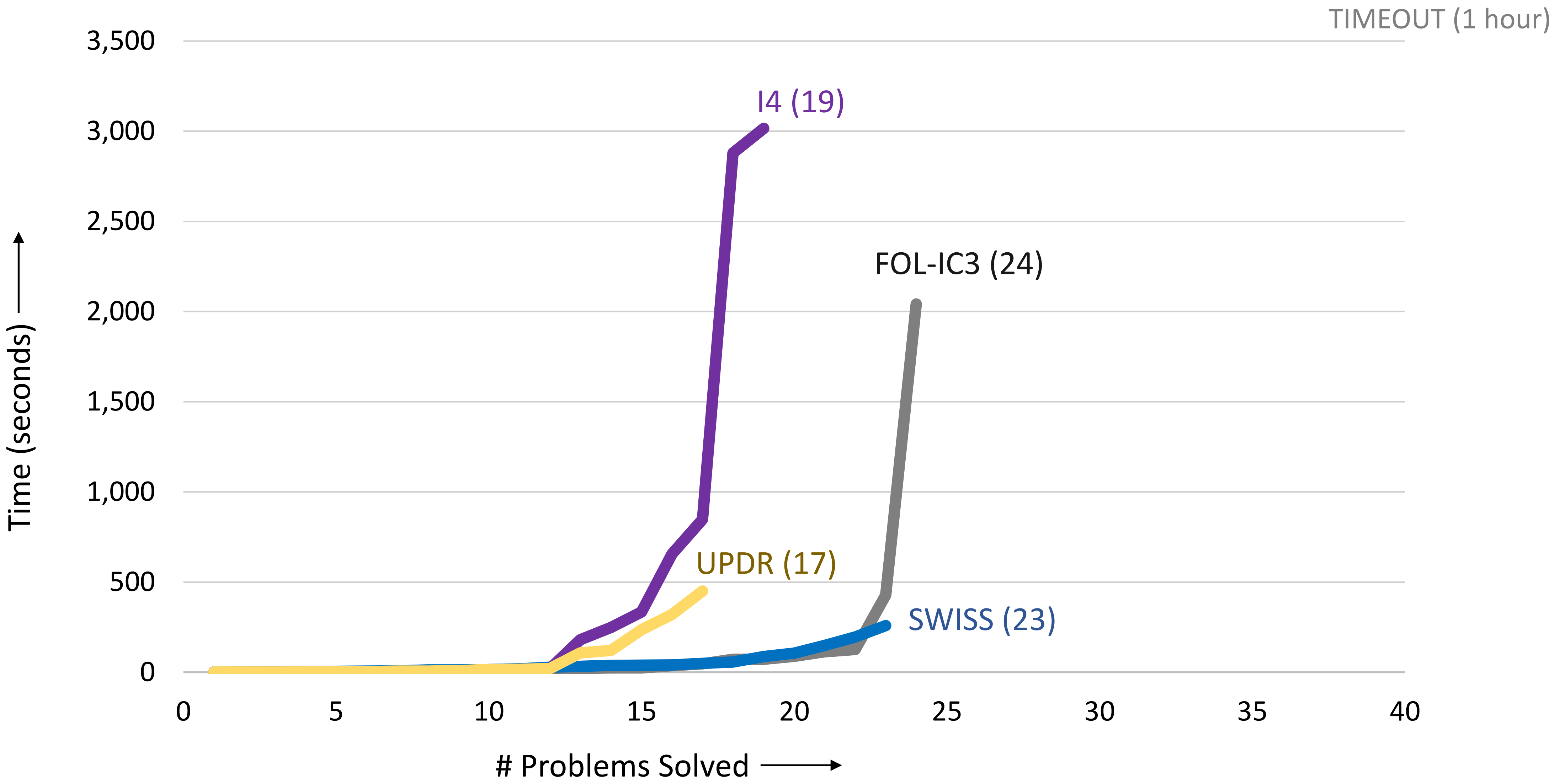
Evaluation

IC3PO	DISTAI	SWISS	FOL-IC3	I4	UPDR
NFM’21	*OSDI’21*	*NSDI’21*	*PLDI’20*	*SOSP’19*	*JACM’17*



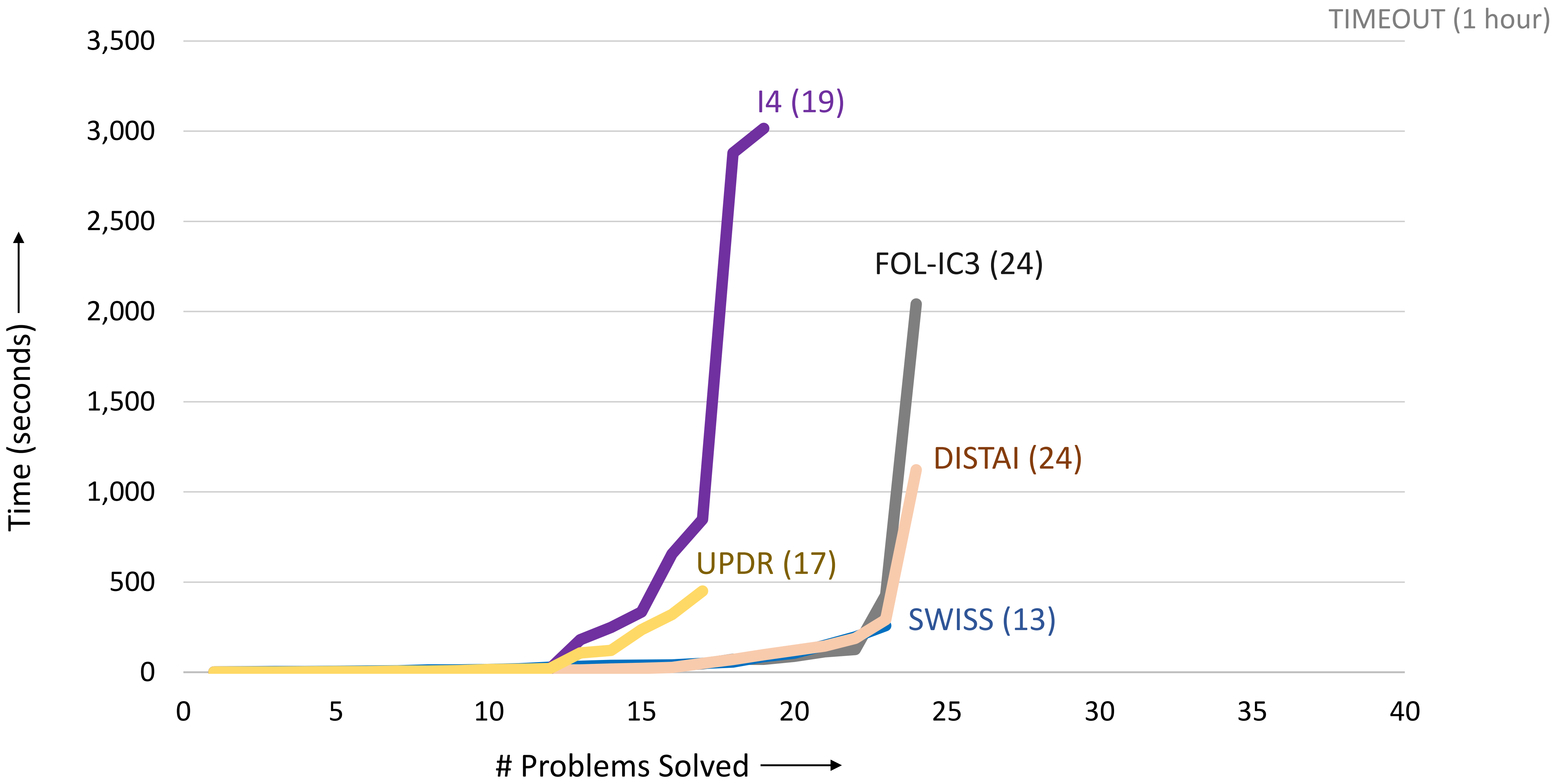
Evaluation

IC3PO	DISTAI	SWISS	FOL-IC3	I4	UPDR
NFM’21	*OSDI’21*	*NSDI’21*	*PLDI’20*	*SOSP’19*	*JACM’17*



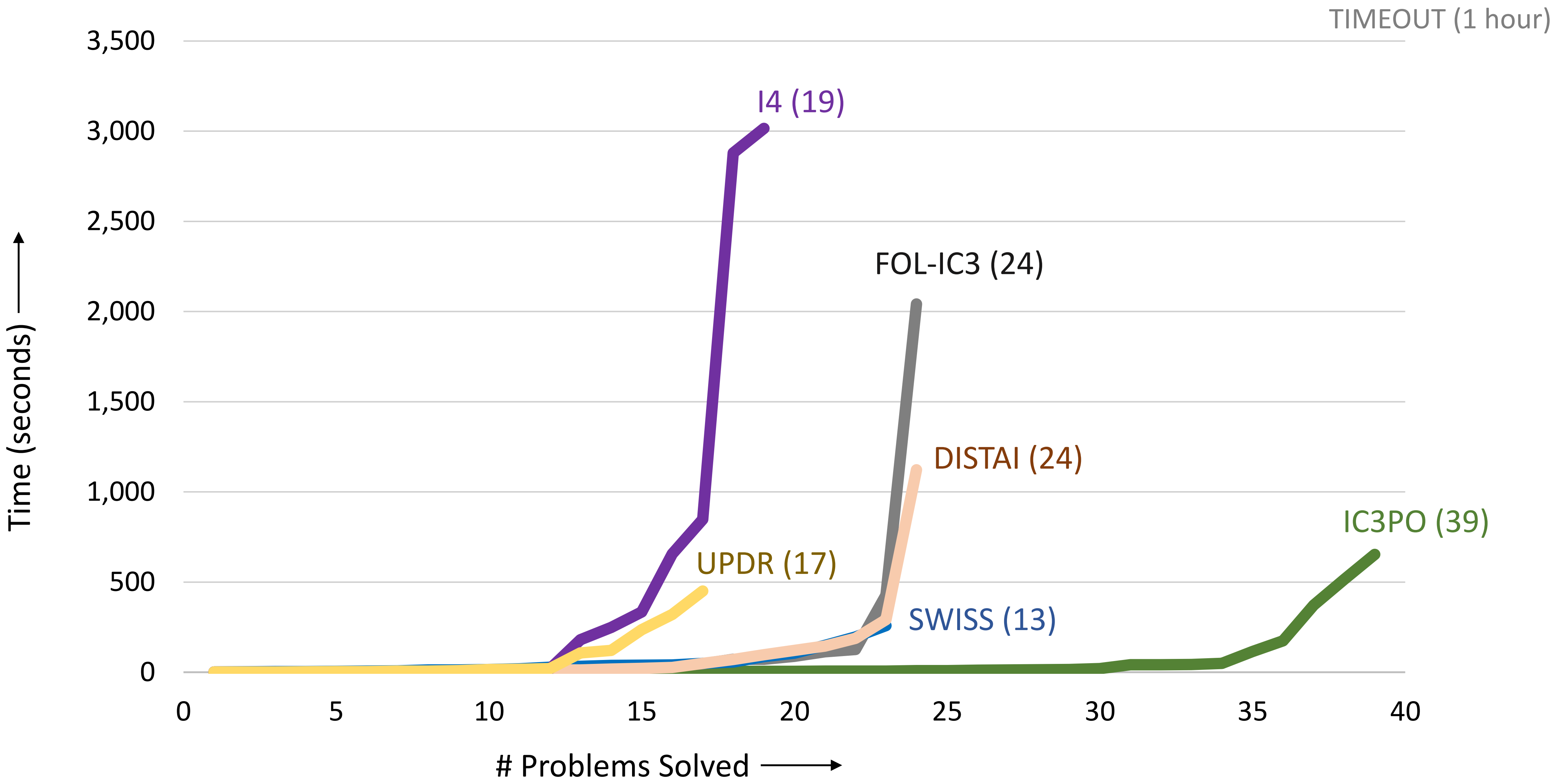
Evaluation

IC3PO	DISTAI	SWISS	FOL-IC3	I4	UPDR
NFM’21	*OSDI’21*	*NSDI’21*	*PLDI’20*	*SOSP’19*	*JACM’17*



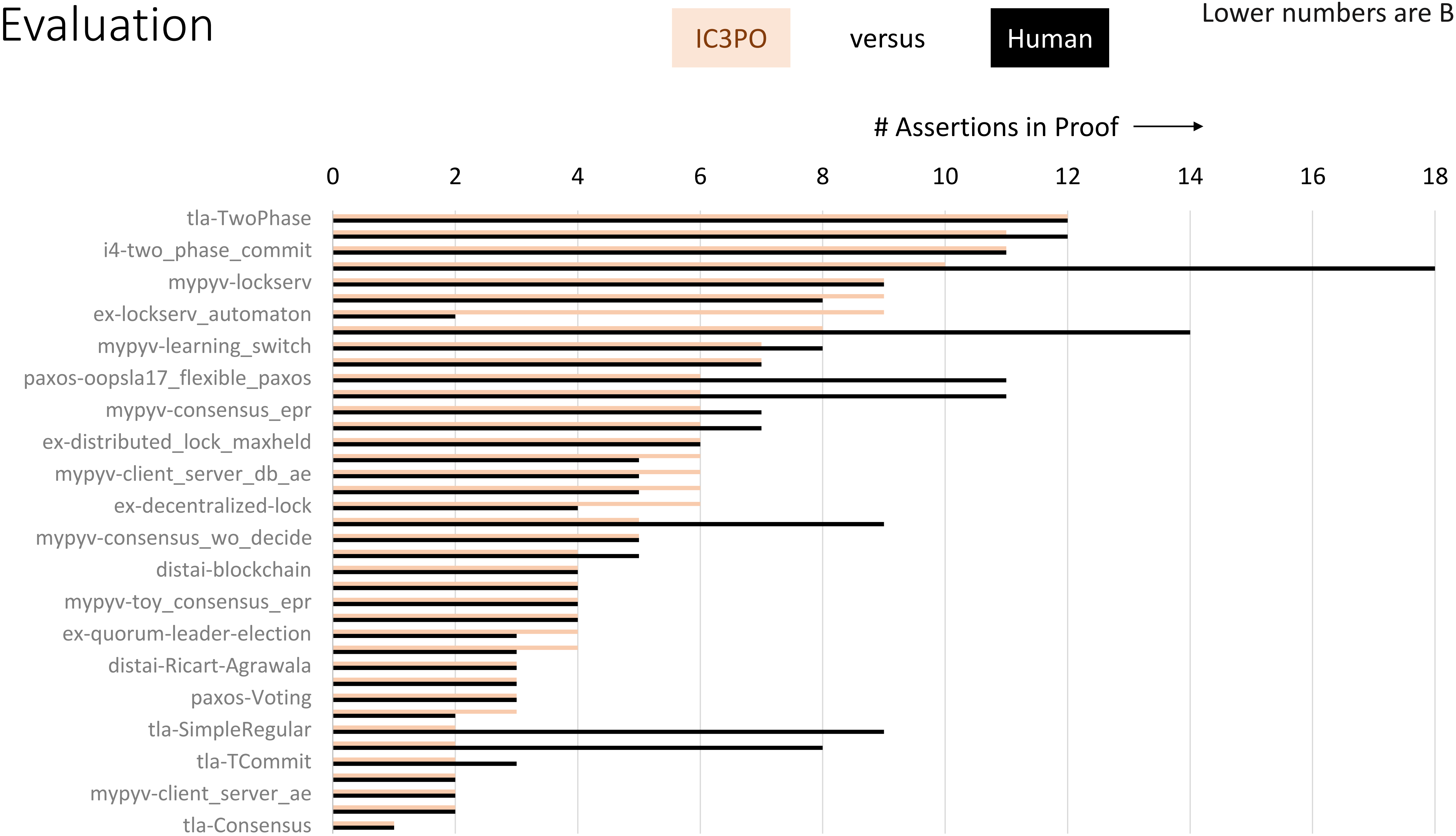
Evaluation

IC3PO	DISTAI	SWISS	FOL-IC3	I4	UPDR
NFM'21	*OSDI'21*	*NSDI'21*	*PLDI'20*	*SOSP'19*	*JACM'17*

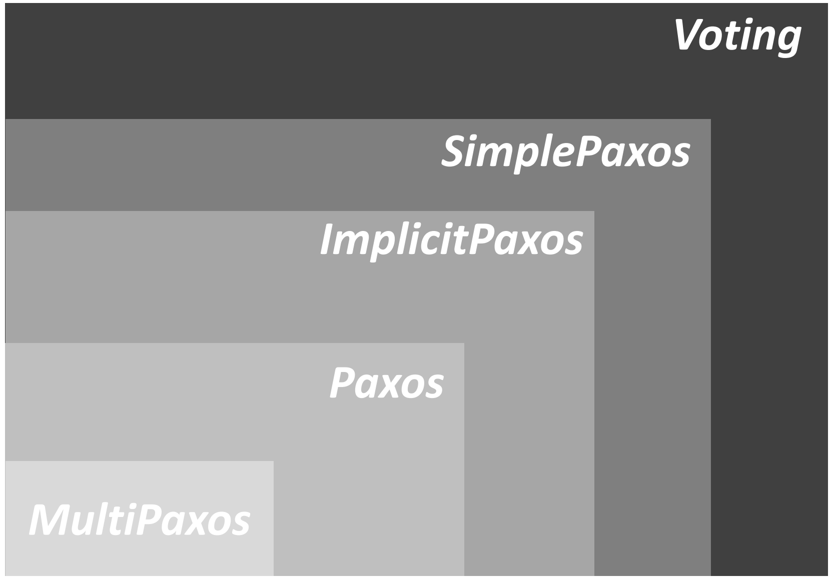


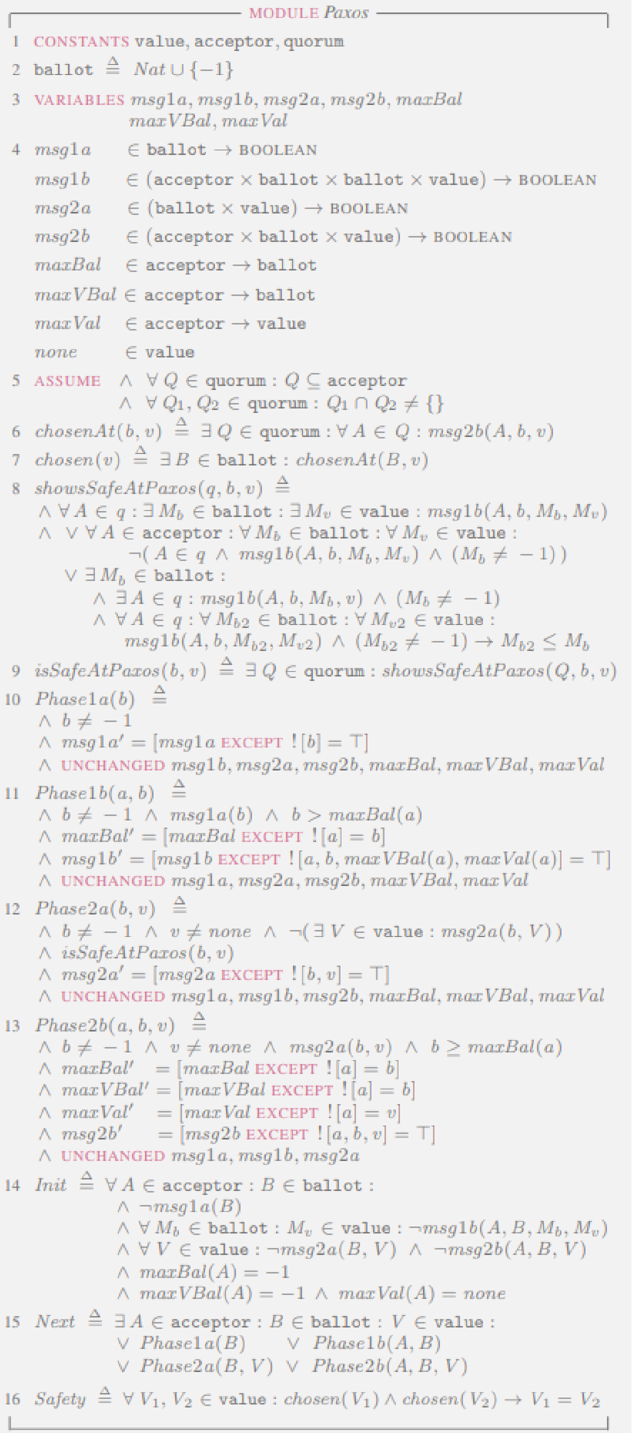
Evaluation

Lower numbers are Better



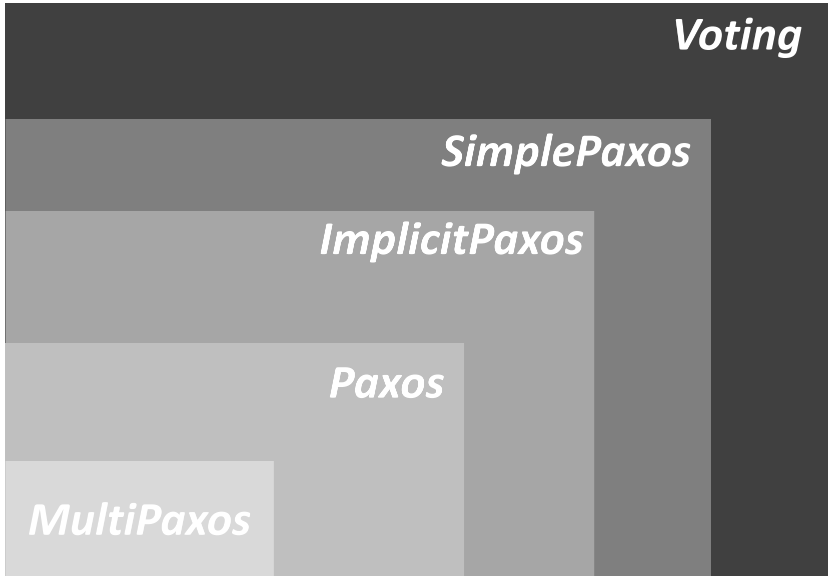
Proving Paxos Automatically





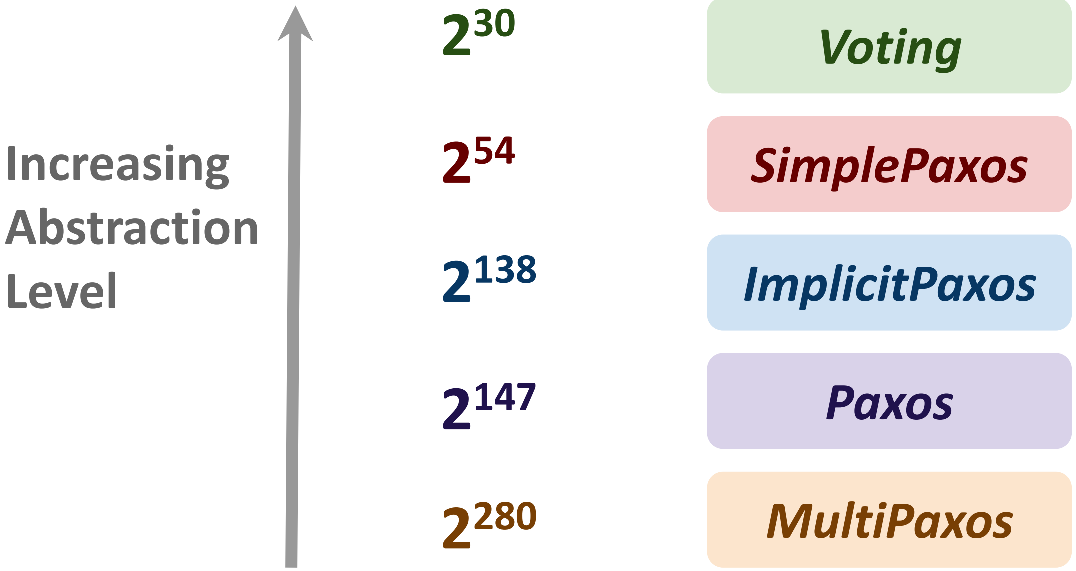
```
                         MODULE Paxos
1  CONSTANTS value, acceptor, quorum
2  ballot ≜ Nat ∪ {−1}
3  VARIABLES msg1a, msg1b, msg2a, msg2b, maxBal
             maxVBal, maxVal
4  msg1a    ∈ ballot → BOOLEAN
   msg1b    ∈ (acceptor × ballot × ballot × value) → BOOLEAN
   msg2a    ∈ (ballot × value) → BOOLEAN
   msg2b    ∈ (acceptor × ballot × value) → BOOLEAN
   maxBal   ∈ acceptor → ballot
   maxVBal  ∈ acceptor → ballot
   maxVal   ∈ acceptor → value
   none     ∈ value
5  ASSUME  ∧ ∀ Q ∈ quorum : Q ⊆ acceptor
           ∧ ∀ Q₁, Q₂ ∈ quorum : Q₁ ∩ Q₂ ≠ {}
6  chosenAt(b, v) ≜ ∃ Q ∈ quorum : ∀ A ∈ Q : msg2b(A, b, v)
7  chosen(v) ≜ ∃ B ∈ ballot : chosenAt(B, v)
8  showsSafeAtPaxos(q, b, v) ≜
    ∧ ∀ A ∈ q : ∃ M_b ∈ ballot : ∃ M_v ∈ value : msg1b(A, b, M_b, M_v)
    ∧  ∨ ∀ A ∈ acceptor : ∀ M_b ∈ ballot : ∀ M_v ∈ value :
              ¬( A ∈ q ∧ msg1b(A, b, M_b, M_v) ∧ (M_b ≠ −1))
       ∨ ∃ M_b ∈ ballot :
           ∧ ∃ A ∈ q : msg1b(A, b, M_b, v) ∧ (M_b ≠ −1)
           ∧ ∀ A ∈ q : ∀ M_b2 ∈ ballot : ∀ M_v2 ∈ value :
              msg1b(A, b, M_b2, M_v2) ∧ (M_b2 ≠ −1) → M_b2 ≤ M_b
9  isSafeAtPaxos(b, v) ≜ ∃ Q ∈ quorum : showsSafeAtPaxos(Q, b, v)
10 Phase1a(b) ≜
    ∧ b ≠ −1
    ∧ msg1a′ = [msg1a EXCEPT ![b] = ⊤]
    ∧ UNCHANGED msg1b, msg2a, msg2b, maxBal, maxVBal, maxVal
11 Phase1b(a, b) ≜
    ∧ b ≠ −1 ∧ msg1a(b) ∧ b > maxBal(a)
    ∧ maxBal′ = [maxBal EXCEPT ![a] = b]
    ∧ msg1b′ = [msg1b EXCEPT ![a, b, maxVBal(a), maxVal(a)] = ⊤]
    ∧ UNCHANGED msg1a, msg2a, msg2b, maxVBal, maxVal
12 Phase2a(b, v) ≜
    ∧ b ≠ −1 ∧ v ≠ none ∧ ¬(∃ V ∈ value : msg2a(b, V))
    ∧ isSafeAtPaxos(b, v)
    ∧ msg2a′ = [msg2a EXCEPT ![b, v] = ⊤]
    ∧ UNCHANGED msg1a, msg1b, msg2b, maxBal, maxVBal, maxVal
13 Phase2b(a, b, v) ≜
    ∧ b ≠ −1 ∧ v ≠ none ∧ msg2a(b, v) ∧ b ≥ maxBal(a)
    ∧ maxBal′  = [maxBal EXCEPT ![a] = b]
    ∧ maxVBal′ = [maxVBal EXCEPT ![a] = b]
    ∧ maxVal′  = [maxVal EXCEPT ![a] = v]
    ∧ msg2b′   = [msg2b EXCEPT ![a, b, v] = ⊤]
    ∧ UNCHANGED msg1a, msg1b, msg2a
14 Init ≜ ∀ A ∈ acceptor : B ∈ ballot :
             ∧ ¬msg1a(B)
             ∧ ∀ M_b ∈ ballot : M_v ∈ value : ¬msg1b(A, B, M_b, M_v)
             ∧ ∀ V ∈ value : ¬msg2a(B, V) ∧ ¬msg2b(A, B, V)
             ∧ maxBal(A) = −1
             ∧ maxVBal(A) = −1 ∧ maxVal(A) = none
15 Next ≜ ∃ A ∈ acceptor : B ∈ ballot : V ∈ value :
             ∨ Phase1a(B)      ∨ Phase1b(A, B)
             ∨ Phase2a(B, V)   ∨ Phase2b(A, B, V)
16 Safety ≜ ∀ V₁, V₂ ∈ value : chosen(V₁) ∧ chosen(V₂) → V₁ = V₂
```

Hierarchical Structure



State-space Size

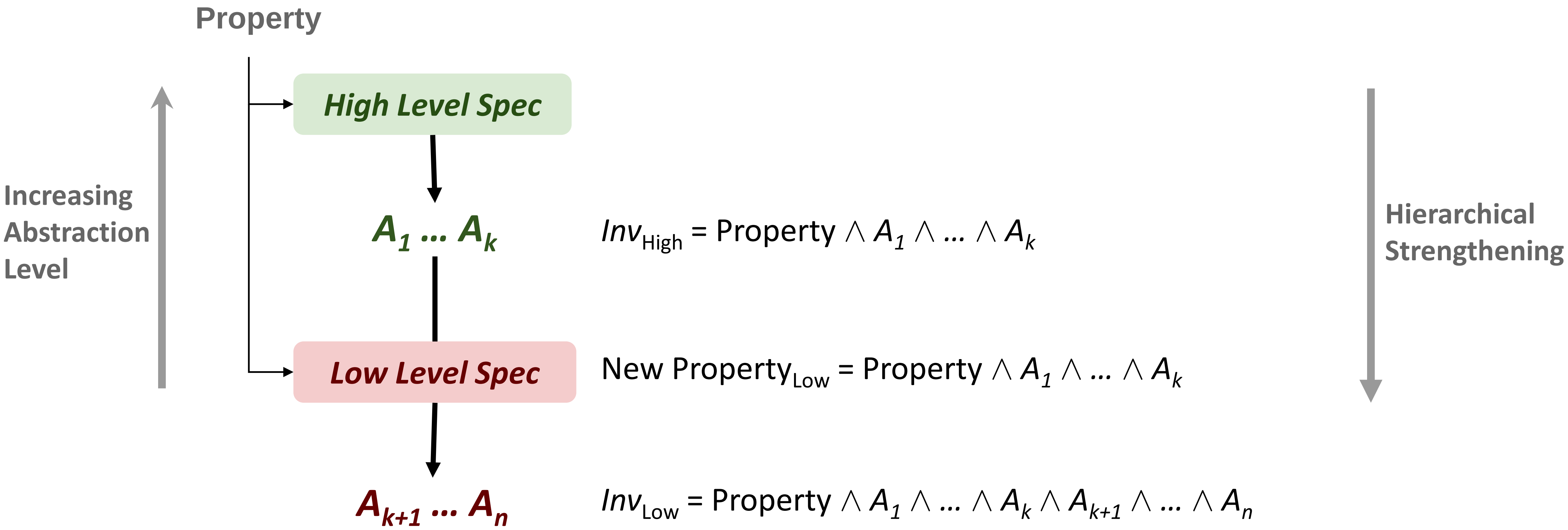
(2 values, 3 acceptors, 3 quorums, 4 ballots)



2^{30}	Voting
2^{54}	SimplePaxos
2^{138}	ImplicitPaxos
2^{147}	Paxos
2^{280}	MultiPaxos

Use Hierarchical Structure to counter Complexity

Hierarchical Strengthening

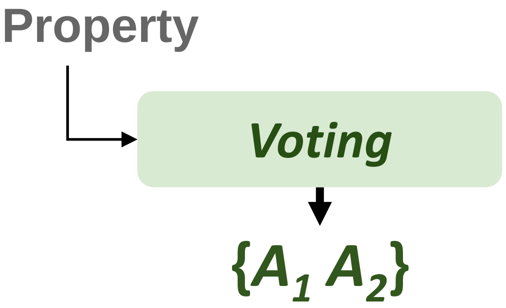


$Inv_{\text{High}} = \text{Property} \wedge A_1 \wedge \dots \wedge A_k$

$\text{New Property}_{\text{Low}} = \text{Property} \wedge A_1 \wedge \dots \wedge A_k$

$Inv_{\text{Low}} = \text{Property} \wedge A_1 \wedge \dots \wedge A_k \wedge A_{k+1} \wedge \dots \wedge A_n$

Proving Paxos Automatically



Input Strengthening Assertions

none

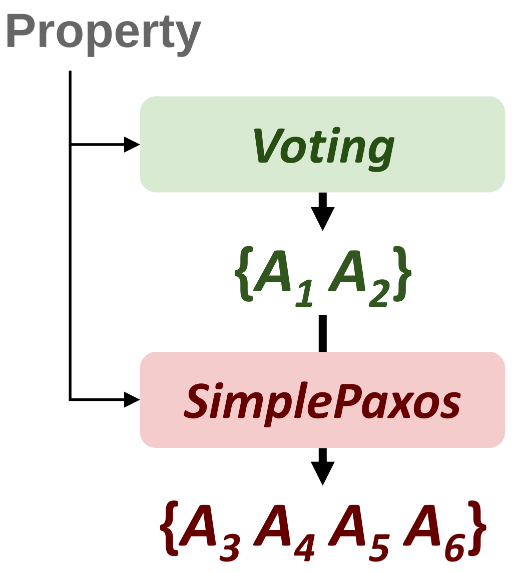
A_1 = ∀ A ∈ **acceptor**, B ∈ **ballot**, V ∈ **value**:
$\quad$ *votes*(A, B, V) ⟶ *isSafeAt*(B, V)

A_2 = ∀ A ∈ **acceptor**, B ∈ **ballot**, V_1, V_2 ∈ **value**:
$\quad$ *chosenAt*(B, V_1) ∧ *votes*(A, B, V_2) ⟶ ($V_1 = V_2$)

A_1: If an acceptor voted for value *V* in ballot number *B*, then *V* is safe at *B*.

A_2: If value *V* is chosen at ballot *B*, then no acceptor can vote for a value different than *V* in *B*.

Proving Paxos Automatically



Property

Voting

$\{A_1\ A_2\}$

SimplePaxos

$\{A_3\ A_4\ A_5\ A_6\}$

Input Strengthening Assertions

none

$A_1\ A_2$

$A_3 = \forall$ B ∈ **ballot**, V ∈ **value**:
$\quad msg2a(B, V) \rightarrow isSafeAt(B, V)$

$A_4 = \forall$ B ∈ **ballot**, V_1, V_2 ∈ **value**:
$\quad msg2a(B, V_1) \wedge msg2a(B, V_2) \rightarrow (V_1 = V_2)$

$A_5 = \forall$ A ∈ **acceptor**, B ∈ **ballot**, V ∈ **value**:
$\quad msg2b(A, B, V) \rightarrow msg2a(B, V)$

$A_6 = \forall$ A ∈ **acceptor**, B ∈ **ballot**:
$\quad msg1a(A, B) \rightarrow maxBal(A) \geq B$

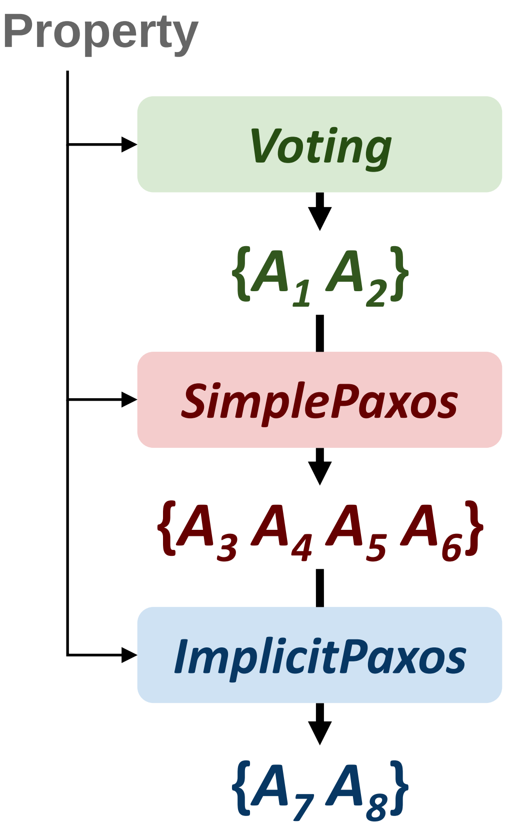
A_3: If ballot *B* leader sends a *2a* message for value *V*, then *V* is safe at *B*.

A_4: A ballot leader can send *2a* messages only for a unique value.

A_5: If an acceptor voted for a value in ballot *B*, then there is a *2a* message for that value at *B*.

A_6: If an acceptor has sent a *1b* message at a ballot *B*, then its *maxBal* is at least as high as *B*.

Proving Paxos Automatically



Input Strengthening Assertions

none

$A_1\ A_2$

$A_1 \ldots A_6$

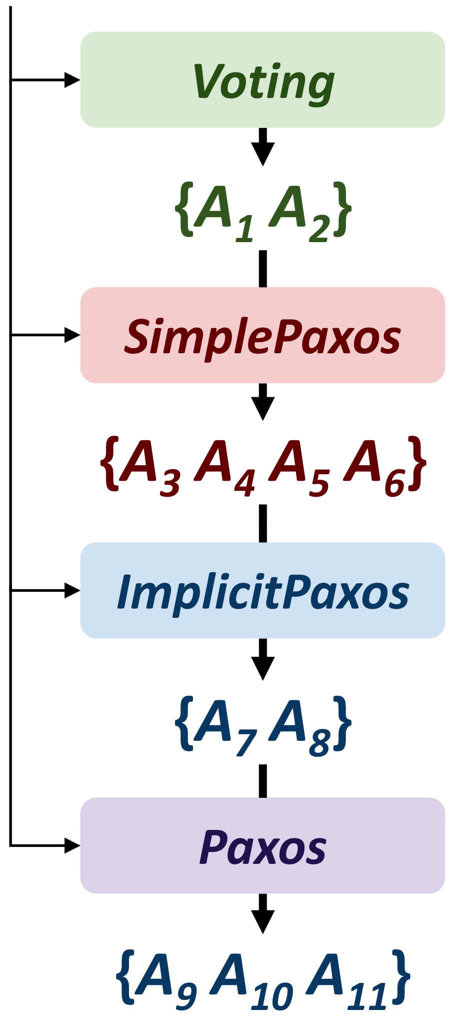
$$\begin{aligned} A_7 = {} & \forall A \in \textbf{acceptor}, B, B_{max} \in \textbf{ballot}, V_{max} \in \textbf{value}: \\ & (B > -1) \wedge (B_{max} > -1) \wedge msg1b(A, B, B_{max}, V_{max}) \\ & \qquad \rightarrow msg2b(A, B_{max}, V_{max}) \end{aligned}$$

$$\begin{aligned} A_8 = {} & \forall A \in \textbf{acceptor}, B, B_{mid}, B_{max} \in \textbf{ballot}, V, V_{max} \in \textbf{value}: \\ & (B > B_{mid} > B_{max}) \wedge msg1b(A, B, B_{max}, V_{max}) \\ & \qquad \rightarrow \neg\, msg2b(A, B_{mid}, V) \end{aligned}$$

A_7: If an acceptor issued a *1b* message at ballot *B* with the maximum vote (B_{max}, V_{max}), and both *B* and B_{max} are higher than -1, then the acceptor has voted for value V_{max} in ballot B_{max}.

A_8: If an acceptor issued a *1b* message at ballot B with the maximum vote (B_{max}, V_{max}), then the acceptor cannot have voted in any ballot number strictly between B_{max} and *B*.

Proving Paxos Automatically



Property	Input Strengthening Assertions
Voting	none
$\{A_1\ A_2\}$	
SimplePaxos	$A_1\ A_2$
$\{A_3\ A_4\ A_5\ A_6\}$	
ImplicitPaxos	$A_1 \dots A_6$
$\{A_7\ A_8\}$	
Paxos	$A_1 \dots A_8$
$\{A_9\ A_{10}\ A_{11}\}$	

$A_9 = \forall$ A ϵ **acceptor**: *maxVBal*(A) $\leq$ *maxBal*(A)

$A_{10} = \forall$ A ϵ **acceptor**, B ϵ **ballot**, V ϵ **value**:
msg2b(A, B, V) $\longrightarrow$ *maxVBal*(A) $\geq$ B

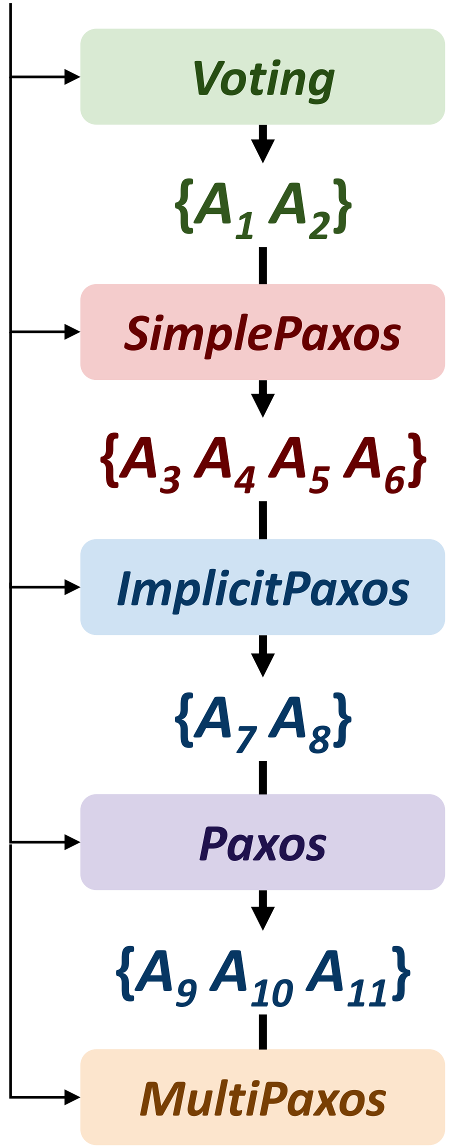
$A_{11} = \forall$ A ϵ **acceptor**:
maxVBal(A) $\geq$ -1 $\longrightarrow$ *msg2b*(A, *maxVBal*(A), *maxVal*(A))

A_9: *maxVBal* of an acceptor is less than or equal to its *maxBal*.

A_{10}: If an acceptor voted in a ballot *B*, then its *maxVBal* is at least as high as *B*.

A_{11}: If acceptor *A* has its *maxVBal* higher than −1, then *A* has already cast a vote (*maxVBal(A), maxVal(A)*).

Proving Paxos Automatically



Property	Input Strengthening Assertions
Voting	none
$\{A_1\ A_2\}$	
SimplePaxos	$A_1\ A_2$
$\{A_3\ A_4\ A_5\ A_6\}$	
ImplicitPaxos	$A_1 \ldots A_6$
$\{A_7\ A_8\}$	
Paxos	$A_1 \ldots A_8$
$\{A_9\ A_{10}\ A_{11}\}$	
MultiPaxos	$A_1 \ldots A_{11}$

✓ Proved!

Proving Paxos Automatically

A_1: If an acceptor voted for value *V* in ballot *B*, then *V* is safe at *B*.

A_2: If value *V* is chosen at ballot *B*, then no acceptor can vote for a value different than *V* in *B*.

A_3: If ballot *B* leader sends a *2a* message for value *V*, then *V* is safe at *B*.

A_4: A ballot leader can send *2a* messages only for a unique value.

A_5: If an acceptor voted for a value in ballot *B*, then there is a *2a* message for that value at *B*.

A_6: If an acceptor has sent a *1b* message at a ballot *B*, then its *maxBal* is at least as high as *B*.

A_7: If an acceptor issued a *1b* message at ballot *B* with the maximum vote (B_{max}, V_{max}), and both *B* and B_{max} are higher than −1, then the acceptor has voted for value V_{max} in ballot B_{max}.

A_8: If an acceptor issued a *1b* message at ballot B with the maximum vote (B_{max}, V_{max}), then the acceptor cannot have voted in any ballot number strictly between B_{max} and *B*.

A_9: *maxVBal* of an acceptor is less than or equal to its *maxBal*.

A_{10}: If an acceptor voted in a ballot *B*, then its *maxVBal* is at least as high as *B*.

A_{11}: If acceptor *A* has its *maxVBal* higher than −1, then *A* has already cast a vote (*maxVBal(A), maxVal(A)*).

Summary

Automatically Verify Distributed Protocols

Finite-Domain Model Checking	No Undecidability Issues
Spatial & Temporal Regularity	Boost Clause Learning
Regularity ↔ Quantification	Compact Quantified Inductive Invariants
Hierarchical Strengthening	High Scalability

Provable Correctness & Assurance

Independently-Checkable Proofs/Traces

IC3PO

IC3 for **P**r**o**ving **P**r**o**tocol **P**r**o**perties

GitHub github.com/aman-goel/ic3po

arXiv arxiv.org/abs/2108.08796