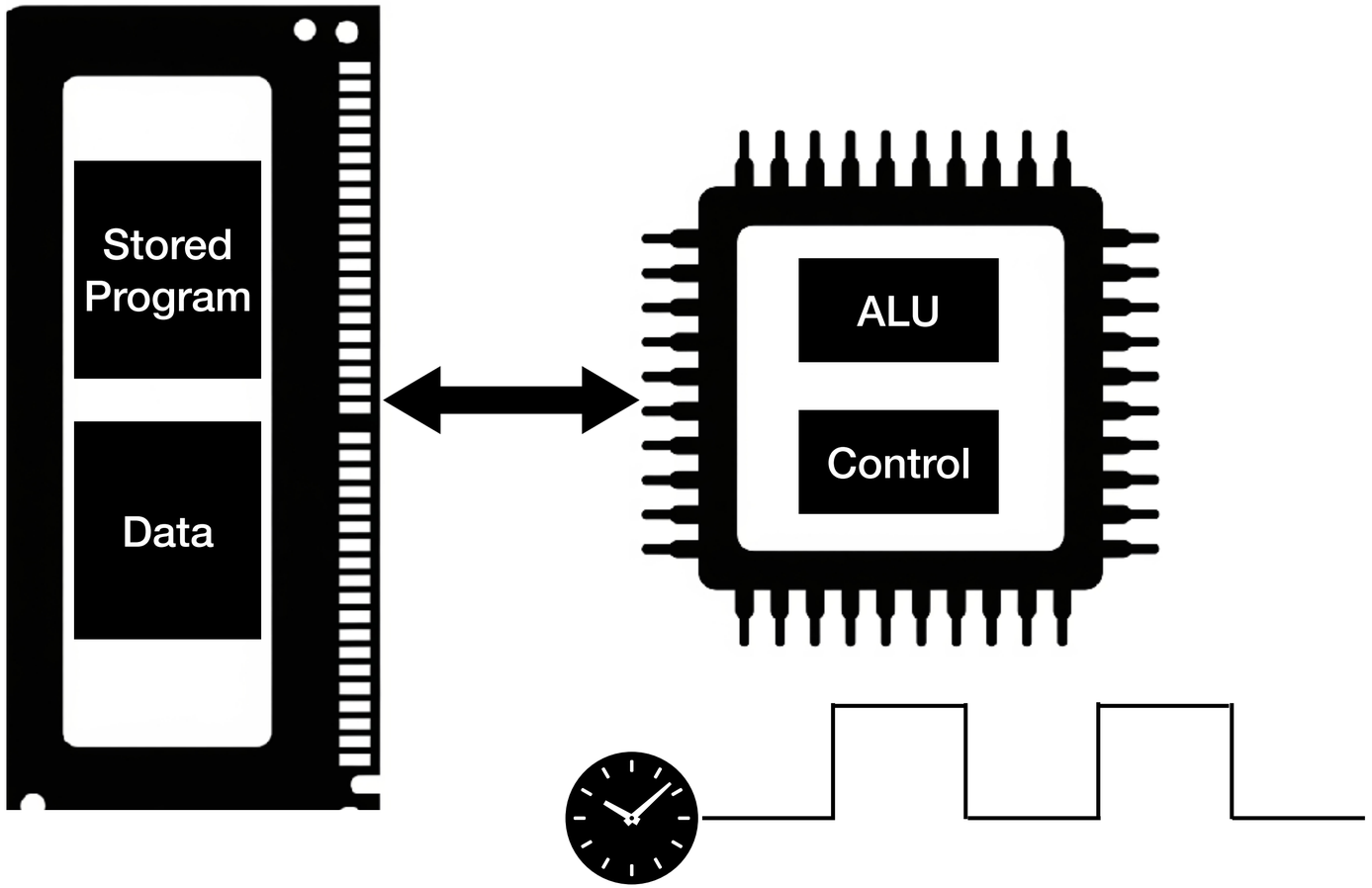
Affordable Formal Verification for Practical Distributed Systems.

Gowtham Kaki
University of Colorado Boulder

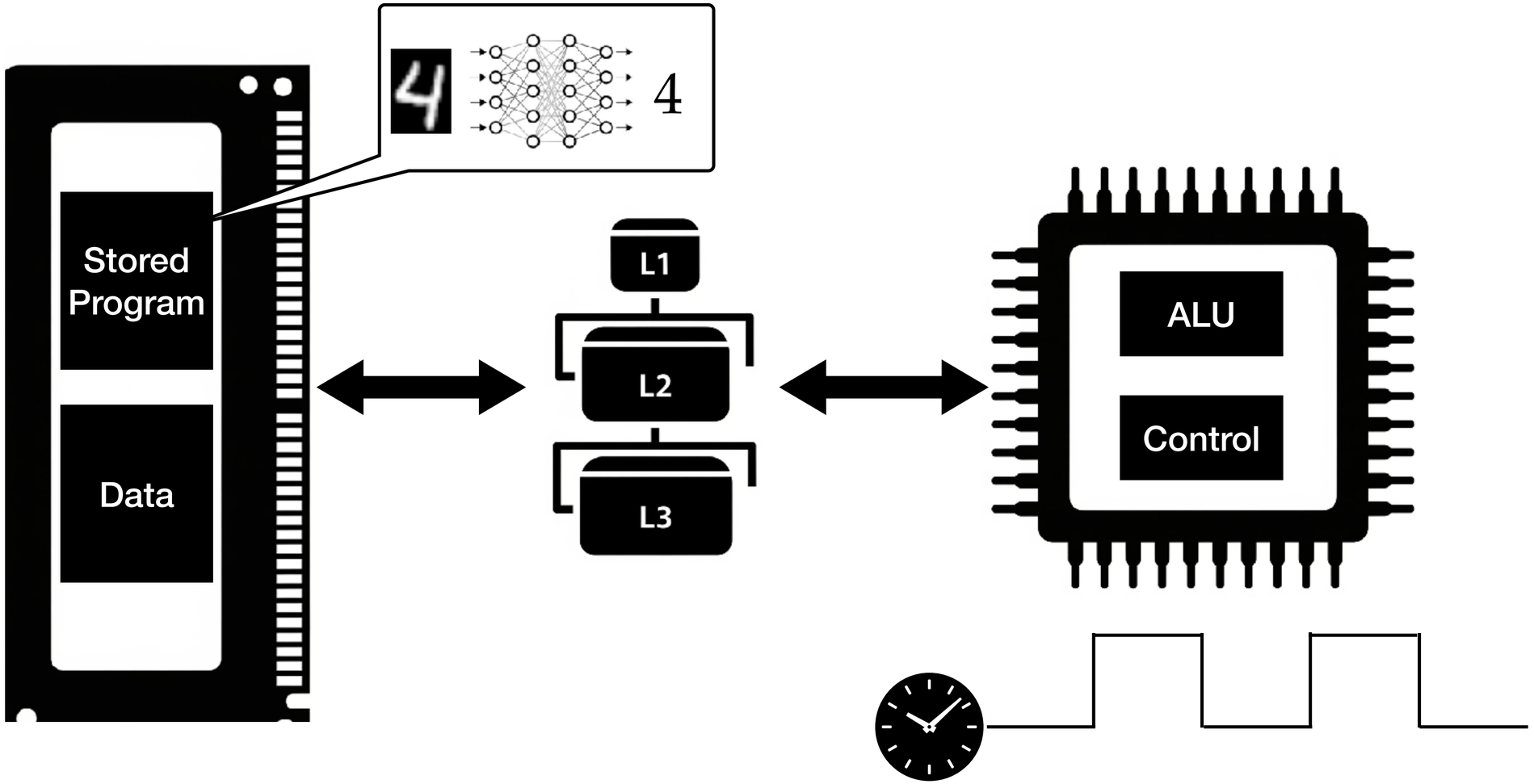
CU Programming Languages
& Verification

A von Neumann Computer

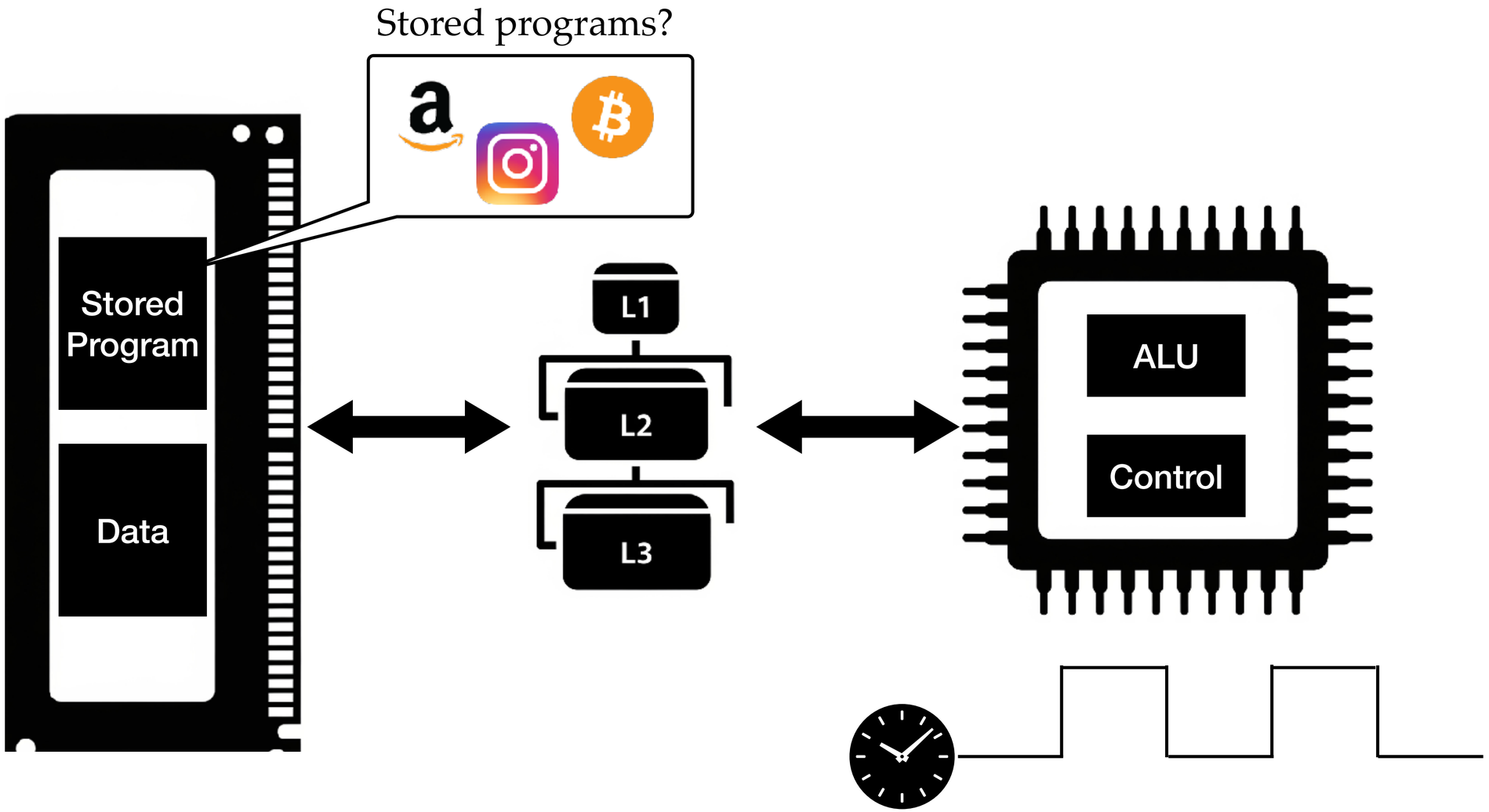
"A stored-program computer"



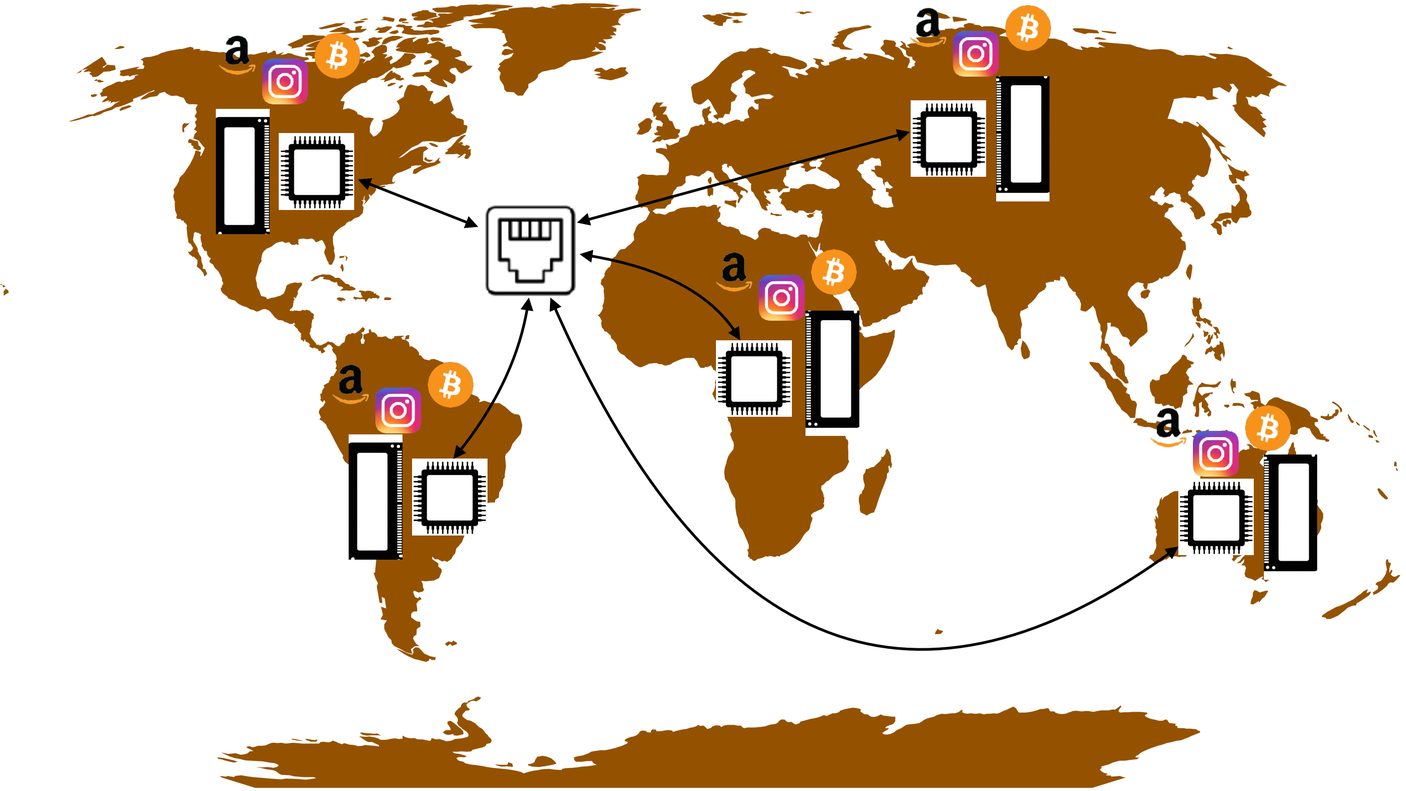
A von Neumann Computer



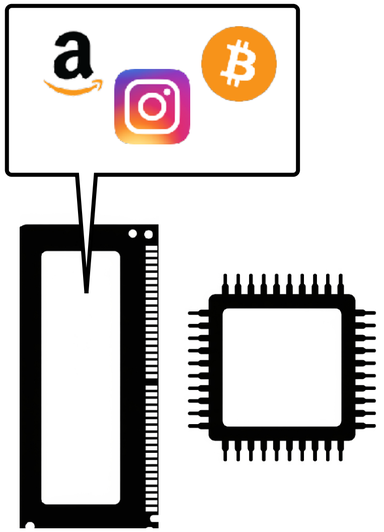
A von Neumann Computer



A Distributed System

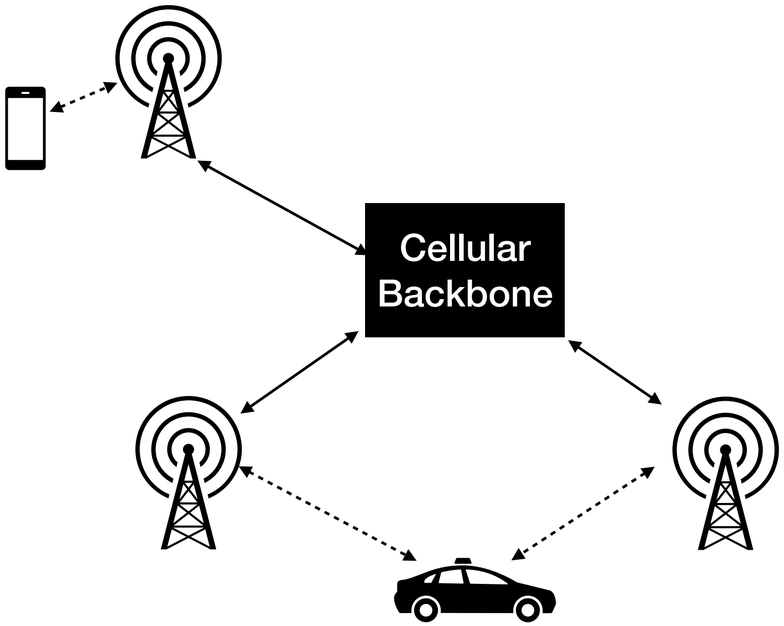


$\not\propto$



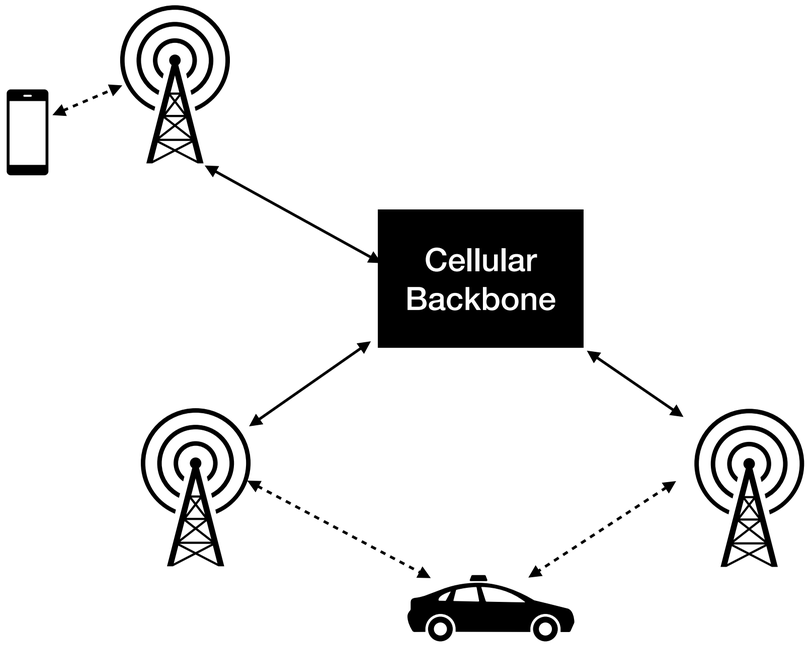
- The speed of light is too low: ~6in per clock cycle
  - $\Longrightarrow$ Control and data flow is *not* instantaneous.
- Internet is a shared network
  - $\Longrightarrow$ Performance is *not* predictable.
  - $\Longrightarrow$ System is *not* isolated.

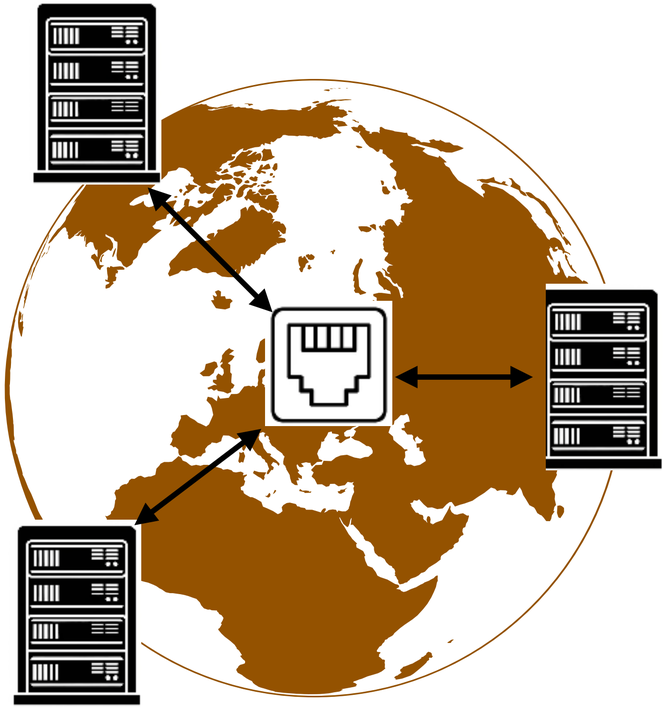
Also Distributed Systems

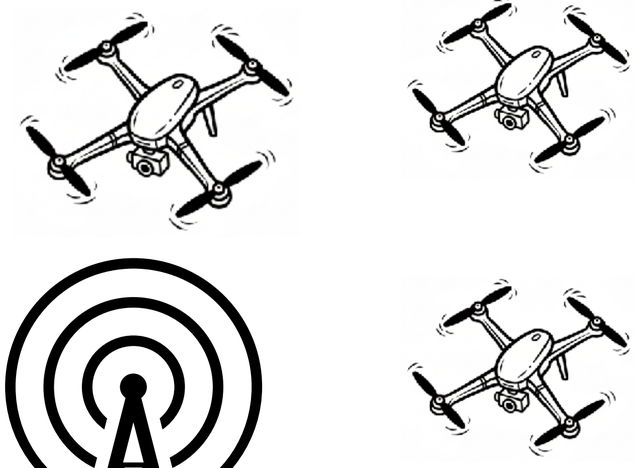


5G Cellular Network

A fleet of unmanned aerial vehicles (UAVs)







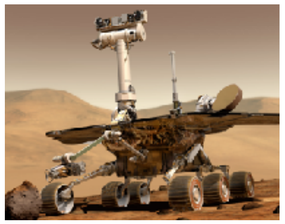
How do we program massive distributed systems to ensure fail-safe operation and resilience against adversarial attacks?

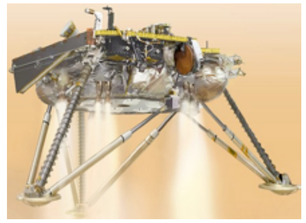
- A question motivating Programming Language and Formal Methods research.

Formal Methods: Recent Success Stories

- "*As AWS has grown, formal methods have become increasingly valuable*" — Brooker and Desai, ACM Queue, 02/04/2025.
- "*Project Everest produced provably correct and secure software deployed in a range of production systems*" — Project Everest Collaborators, Microsoft.

- "*.. DARPA successfully applied the HACMS tools to other military systems, including satellites and self-driving convoy trucks. Each successful project has validated the advantages of applying a formal methods approach to software security*" — DARPA HACMS Project Report.
- "*DARPA touts 'formal methods' for nipping cyber disasters in the bud*" — Defense News, 02/21/2025.





- Kornfeld et al, Verification and Validation of the Mars Science Laboratory/Curiosity Rover Entry, Descent, and Landing System, JSR Vol. 51, 2014.

- "*Formal verification is one of the recommended techniques for improving smart contract security*" — Ethereum Official Documentation.

Formal Methods: Impediments to Widespread Adoption

Poor scalability

- Formal verification is mostly confined to abstract models.
- Verifying real distributed systems requires enormous computational resources and great manual effort.

Insufficient generality

- Formal methods research is almost entirely focused on safety properties.
- Emerging internet-native distributed systems are most concerned about privacy and security.

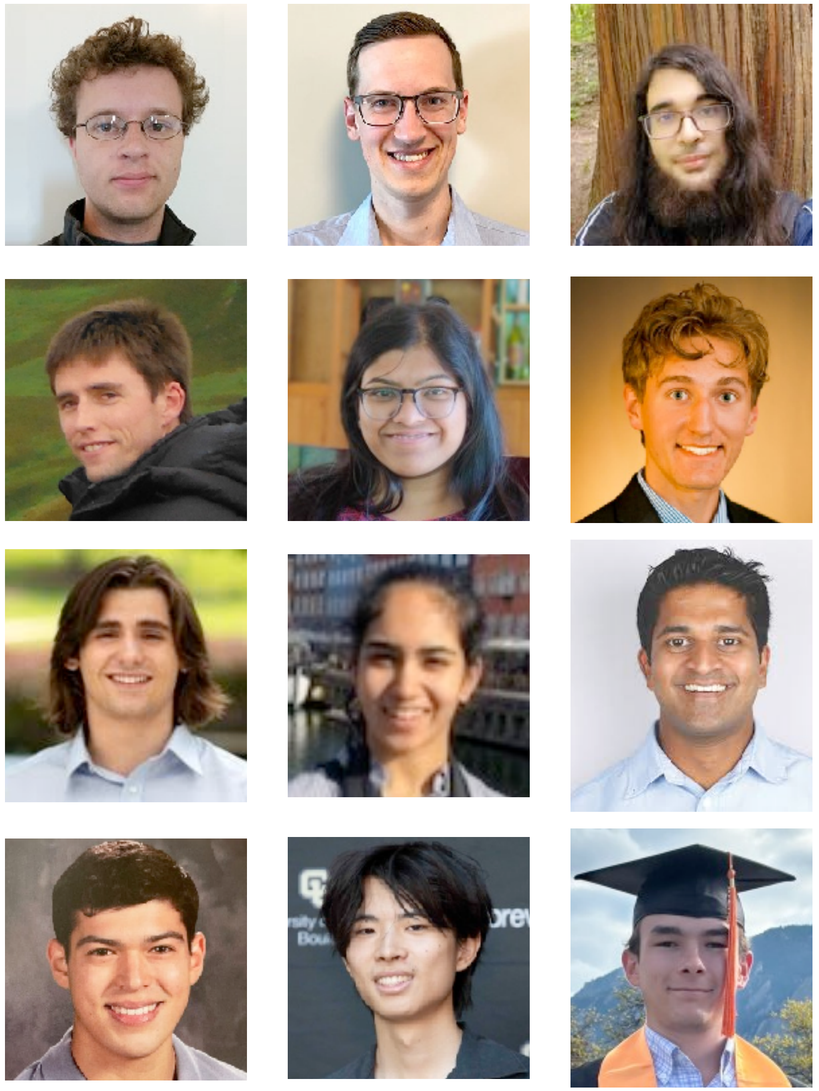
Steep Learning Curve

- Writing specifications, invariants, and proofs requires training in formal logic.
- Formal methods need to work with what's available: unit tests, assertions, and natural language documentation.

My Research

Programming abstractions and automated reasoning techniques to bring affordable formal verification to practical distributed systems.

CUPLV
Programming Languages and Verification
University of Colorado Boulder



This Talk

Poor scalability

- Formal verification is currently
- 1. Improve Scalability:
- from *models* to *implementations*
- requires enormous computational resources and great manual effort.

Insufficient generality

- Formal methods research is
- 2. Improve Generality:
- from *safety* to *security & privacy*
- distributed systems are most concerned about privacy and security.

Steep Learning Curve

- Writing specifications,
- 3. Improve Usability:
- from *experts* to *programmers*
- with what's available: unit tests, assertions, and natural language documentation.

1. Improving Scalability: From *models* to *implementations*

Contributors

Nicholas Lewchenko

Prasanth Prahladan

Evan Chang

Tools

Quark

https://github.com/cuplv/Quark

Super-V

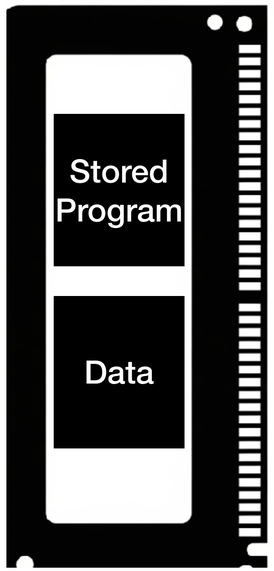
https://github.com/cuplv/super-v

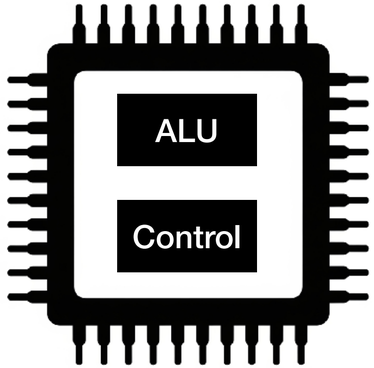
Publications

- Nicholas Lewchenko, Gowtham Kaki, and Bor-Yuh Evan Chang. **Bolt-On Strong Consistency: Specification, Implementation, and Verification**. Proceedings of the ACM on Programming Languages (PACMPL), issue **OOPSLA1 2025**, Article No.: 137, Pages 1604 - 1631.
- Gowtham Kaki, Prasanth Prahladan, and Nicholas Lewchenko. **RunTime-Assisted Convergence in Replicated Data Types**. Proceedings of 43rd ACM SIGPLAN Programming Language Design and Implementation (**PLDI 2022)**, Pages 364 - 378.
- Nicholas Lewchenko and Gowtham Kaki. **Distributed Consensus Algorithms as Replicated State Applications**. ACM SIGOPS Principles and Practice of Consistency for Distributed Data (**PaPoC 2023**).
- Gowtham Kaki, Prasanth Prahladan, and Nicholas Lewchenko. **Bolt-On Convergence in Mergeable Replicated Data Types**. ACM SIGOPS Principles and Practice of Consistency for Distributed Data (**PaPoC 2022**).

State Machine Abstraction

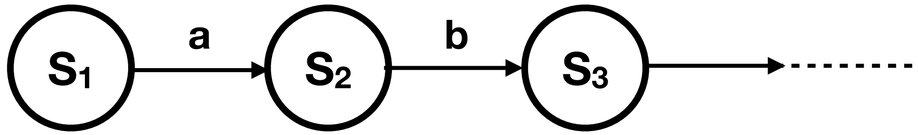
Micro-architectural abstraction





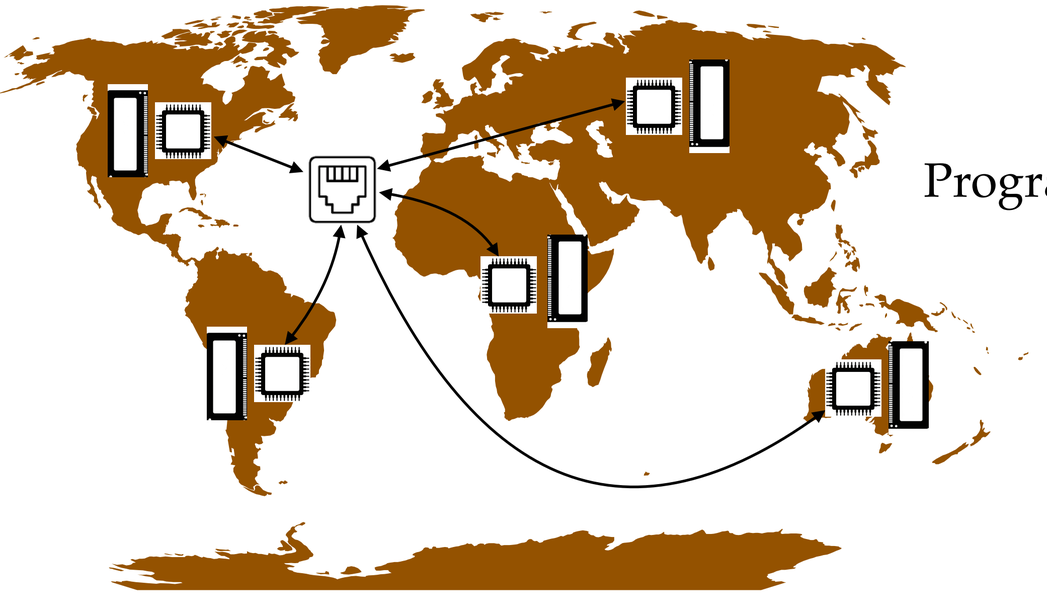
Programming Language

Programming Abstraction



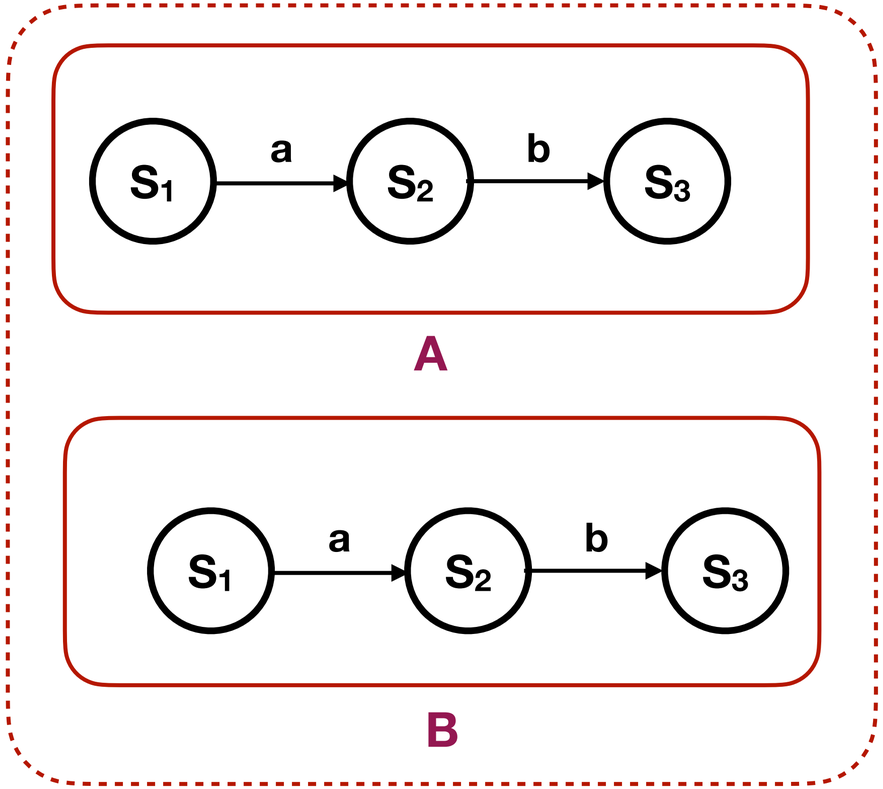
Replicated State Machine (RSM) Abstraction

Micro-architectural abstraction

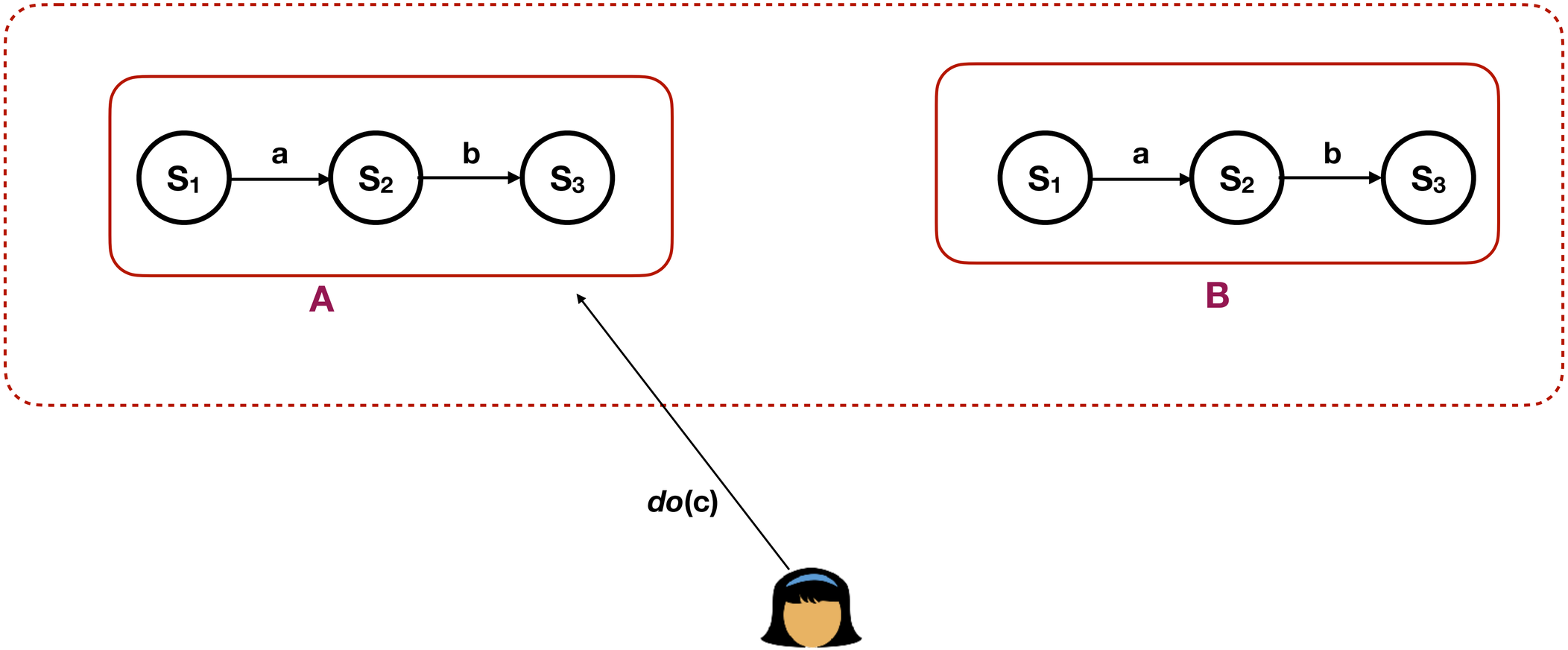


Programming Language

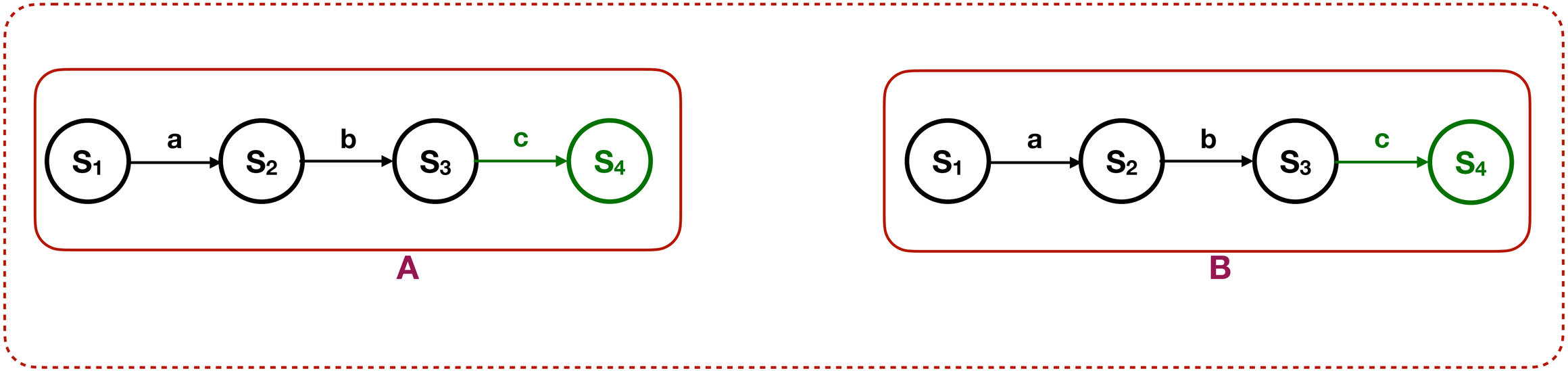
Programming Abstraction



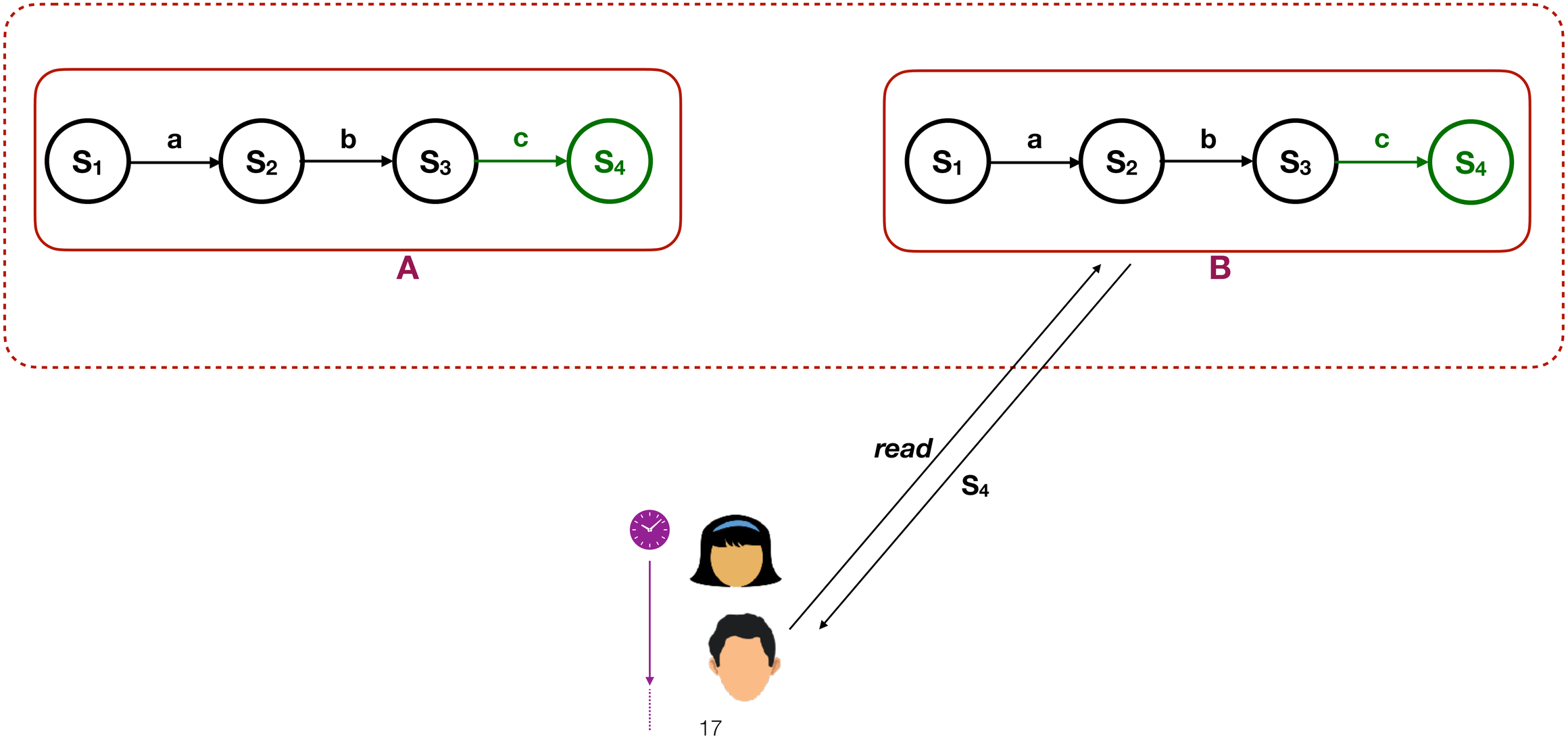
Replicated State Machines



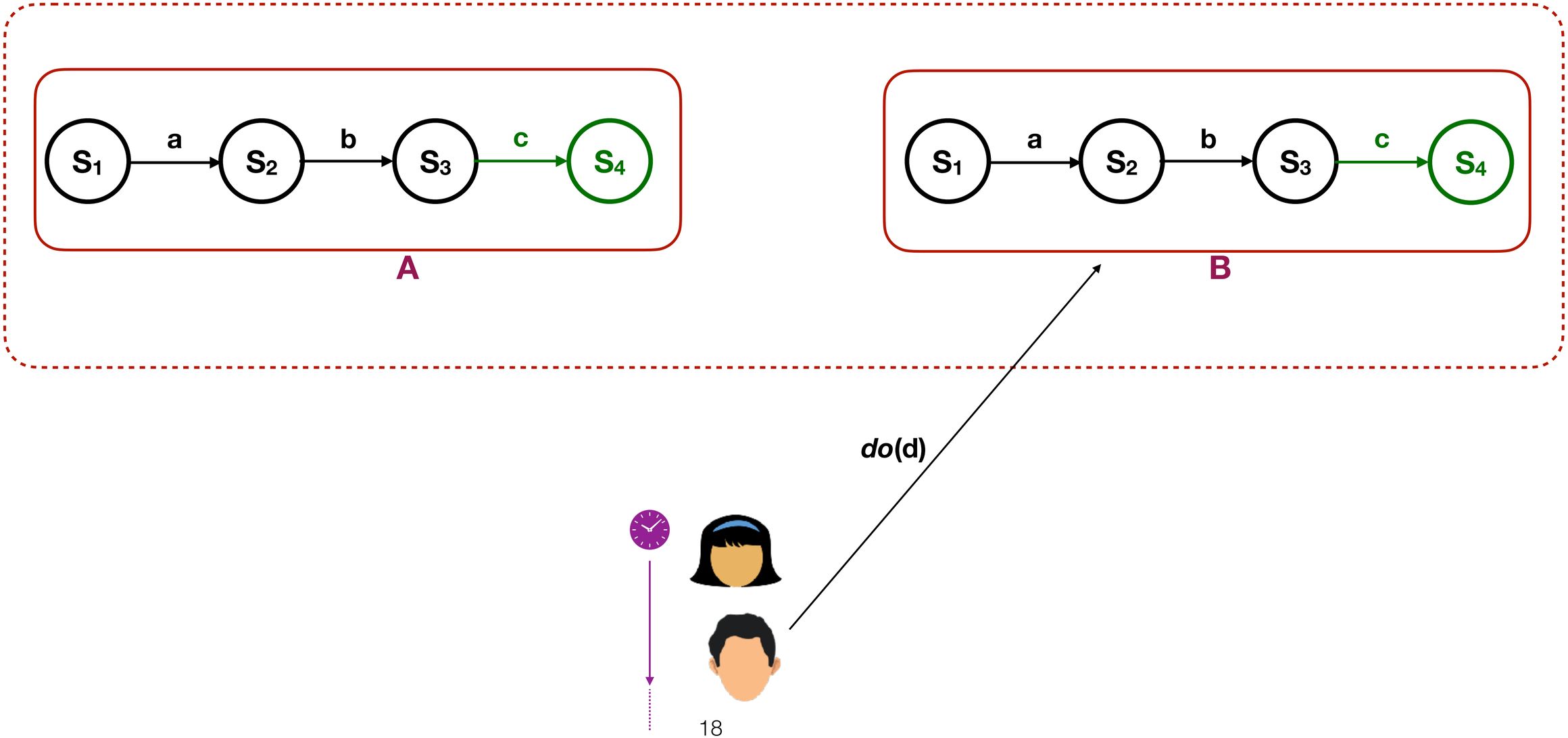
Replicated State Machines



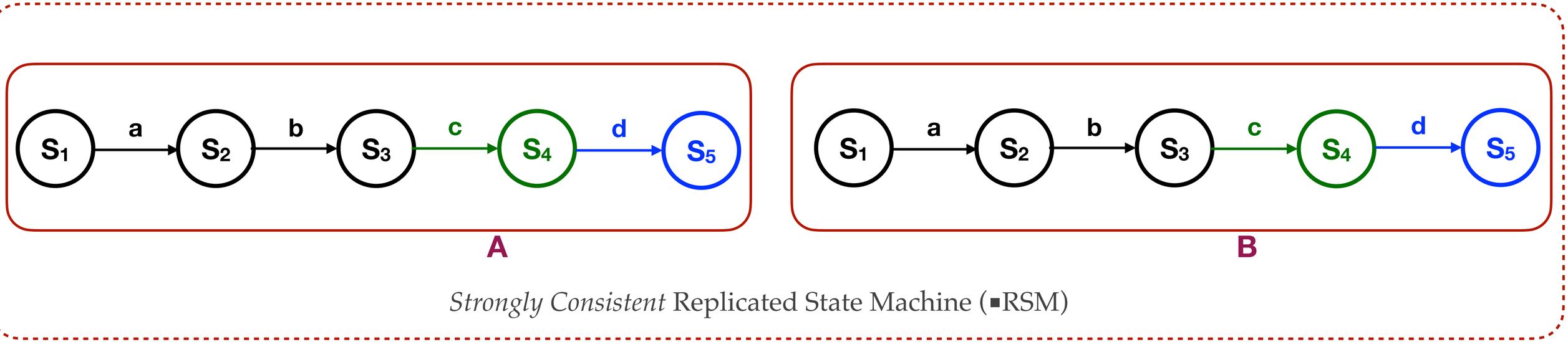
Replicated State Machines



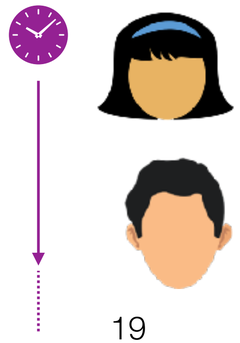
Replicated State Machines



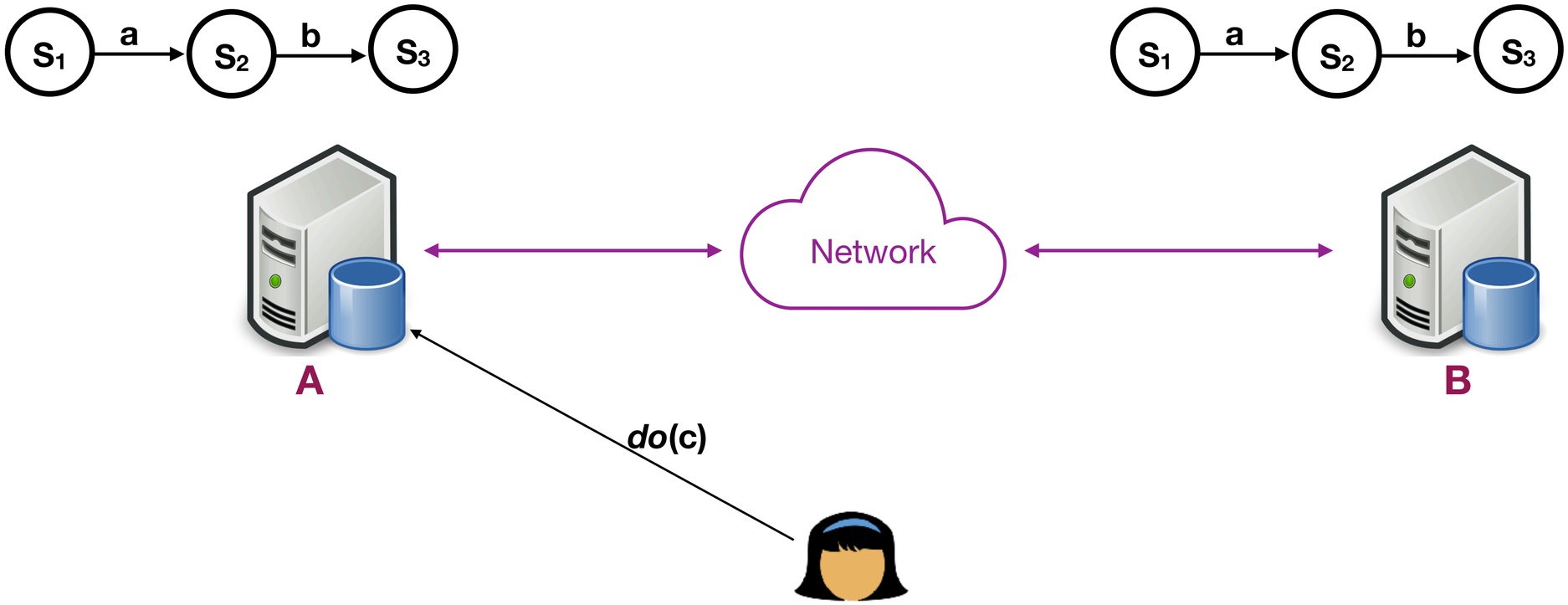
Replicated State Machines



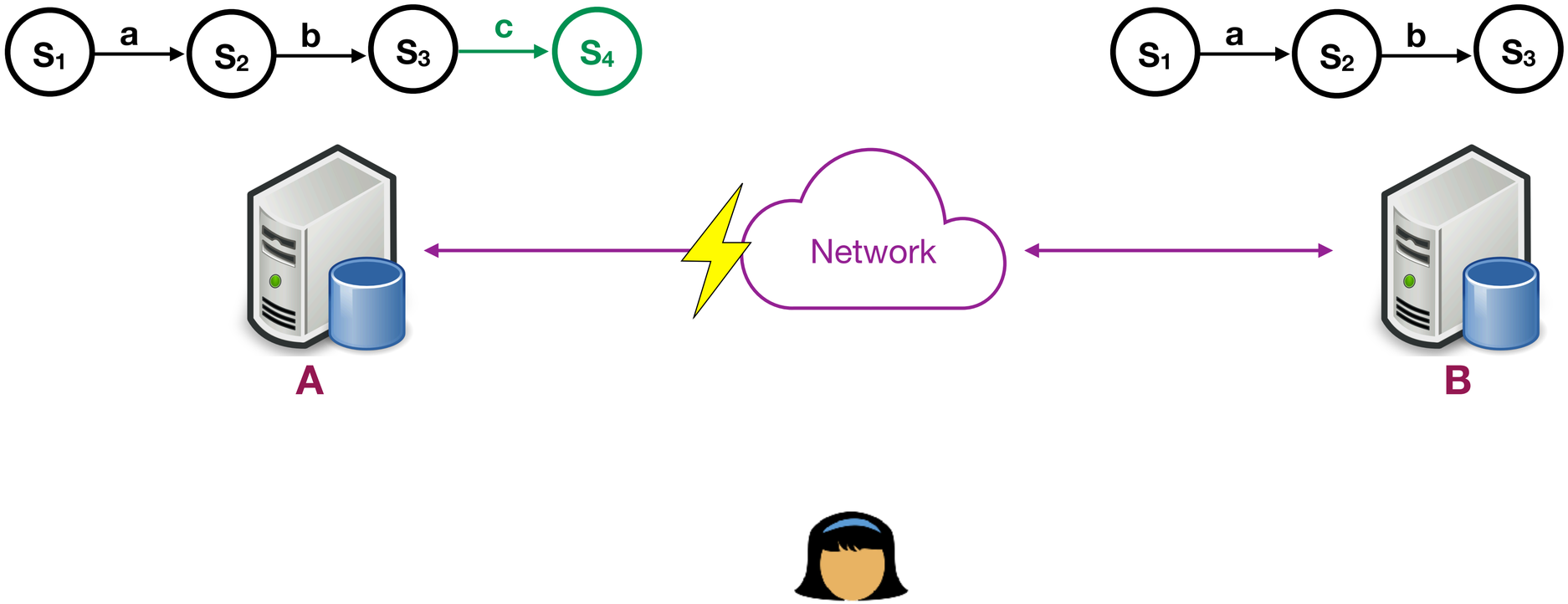
Strongly Consistent Replicated State Machine (▪RSM)



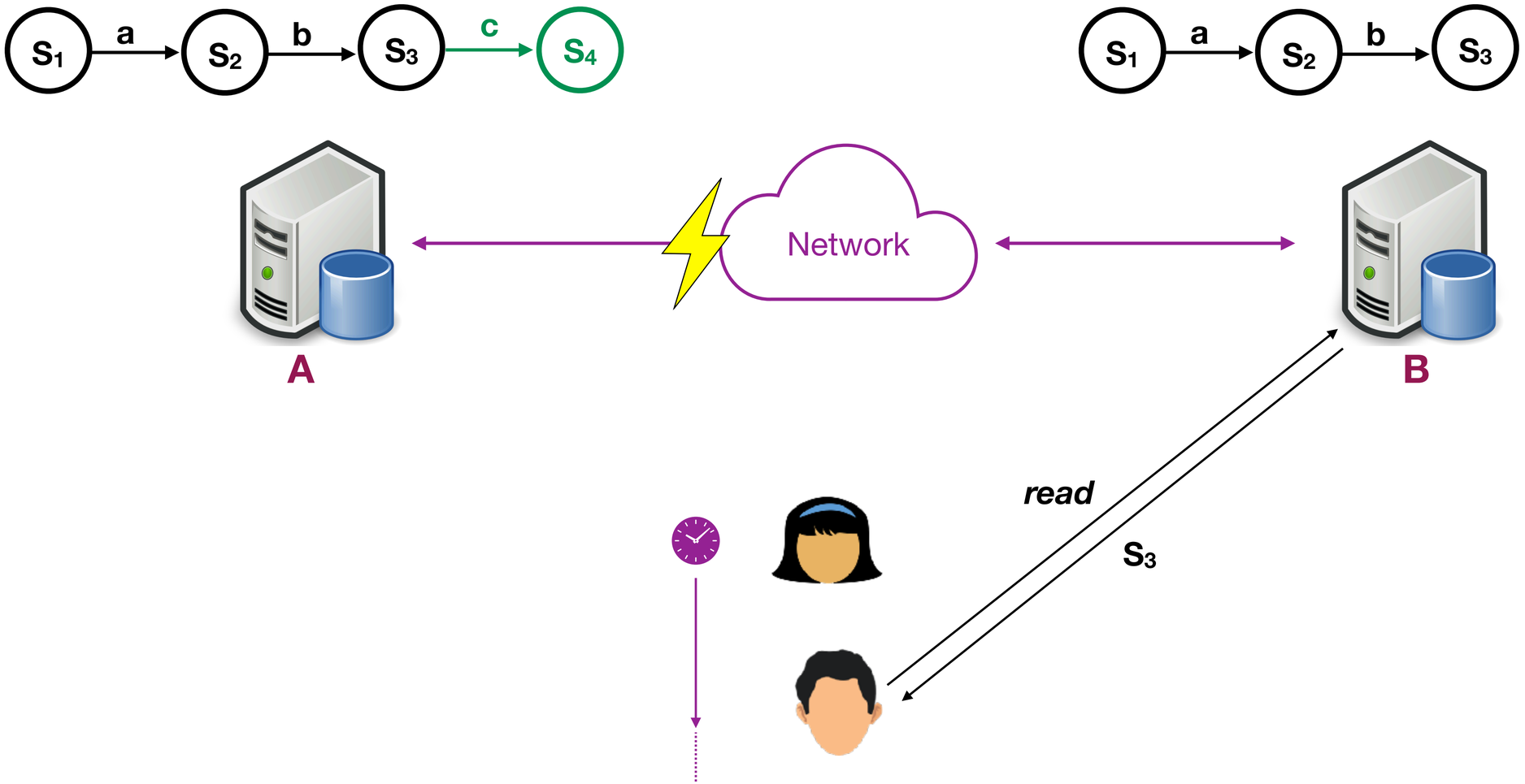
Distributed Systems In Practice



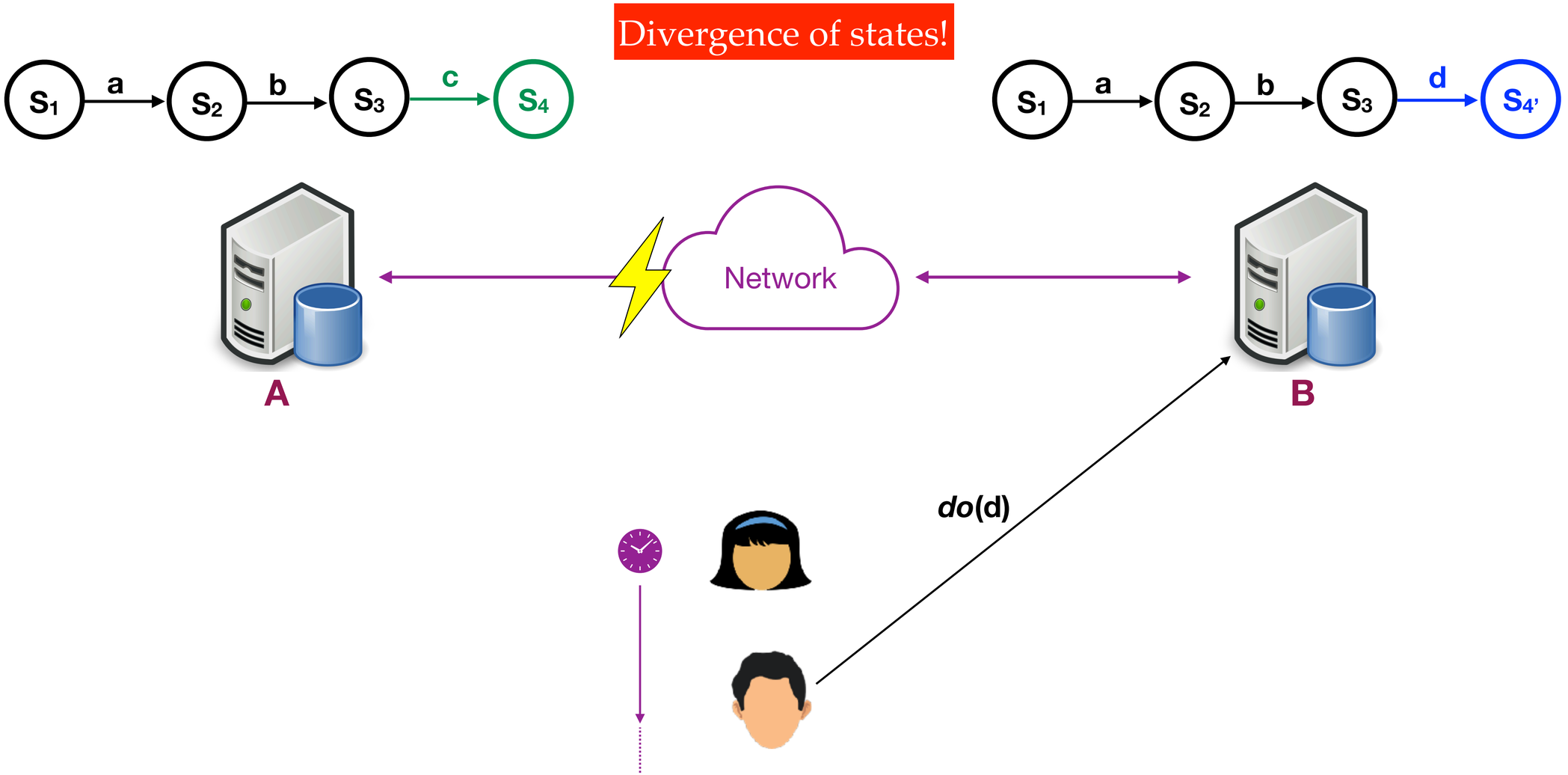
Distributed Systems In Practice



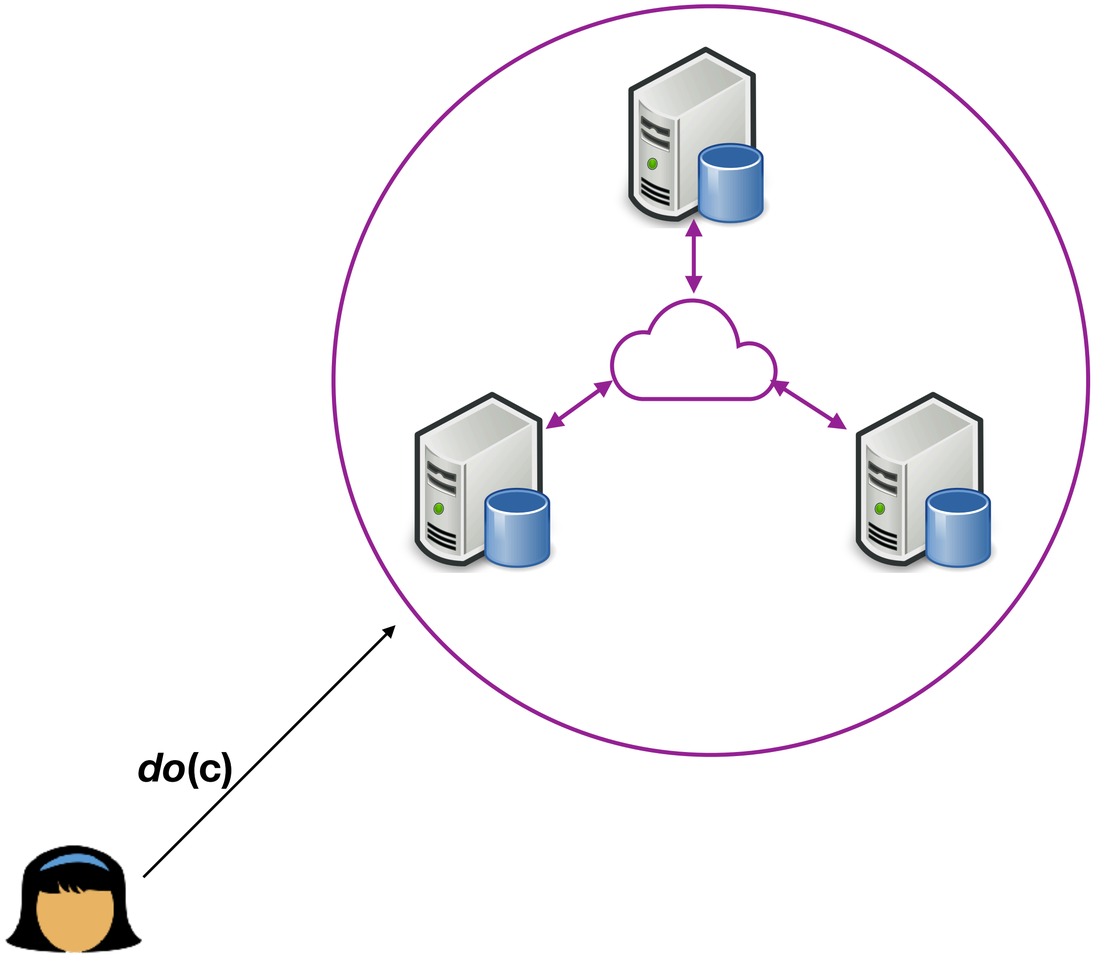
Distributed Systems In Practice

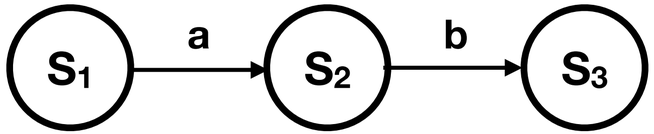


Distributed Systems In Practice

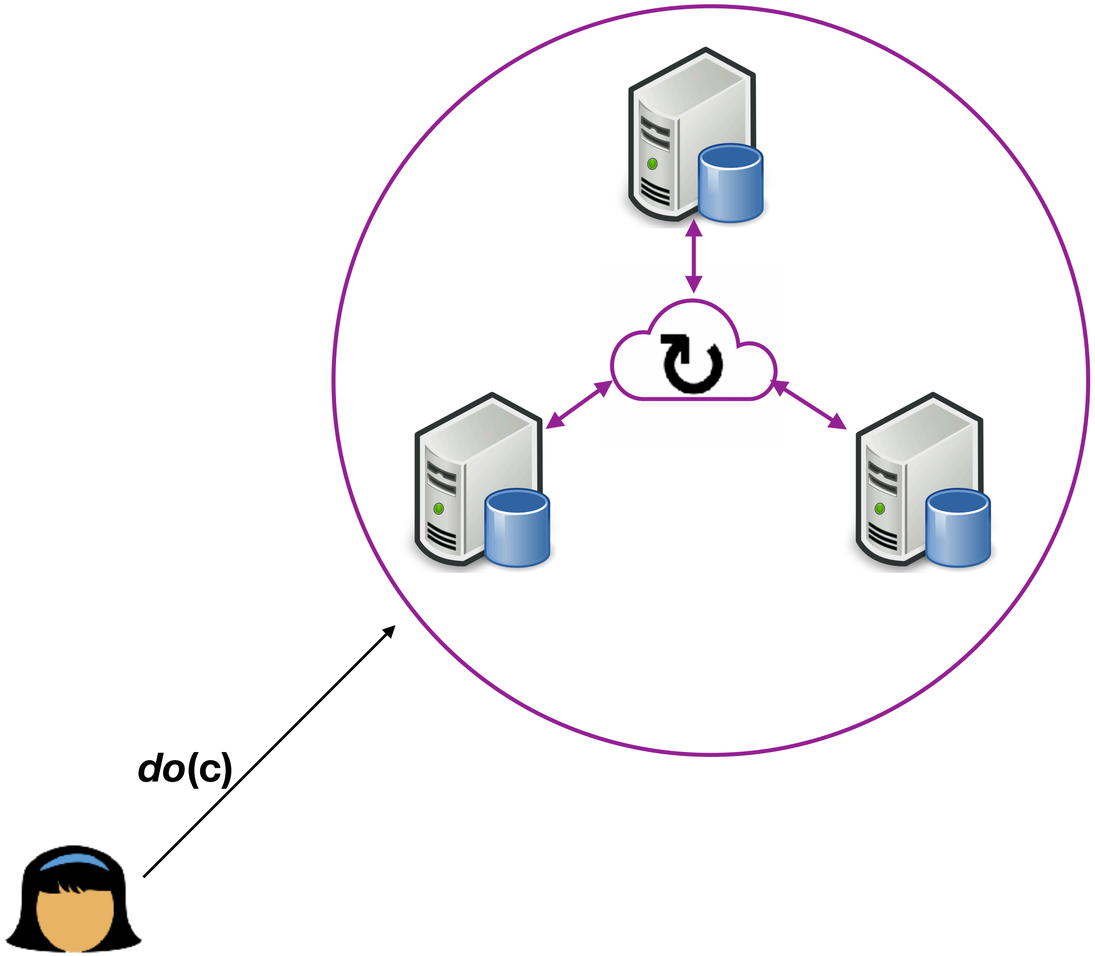


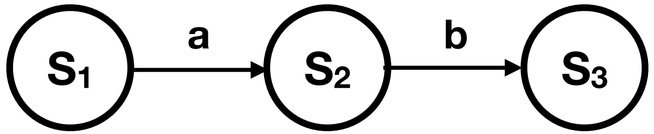
Distributed System + Consensus = ■RSM



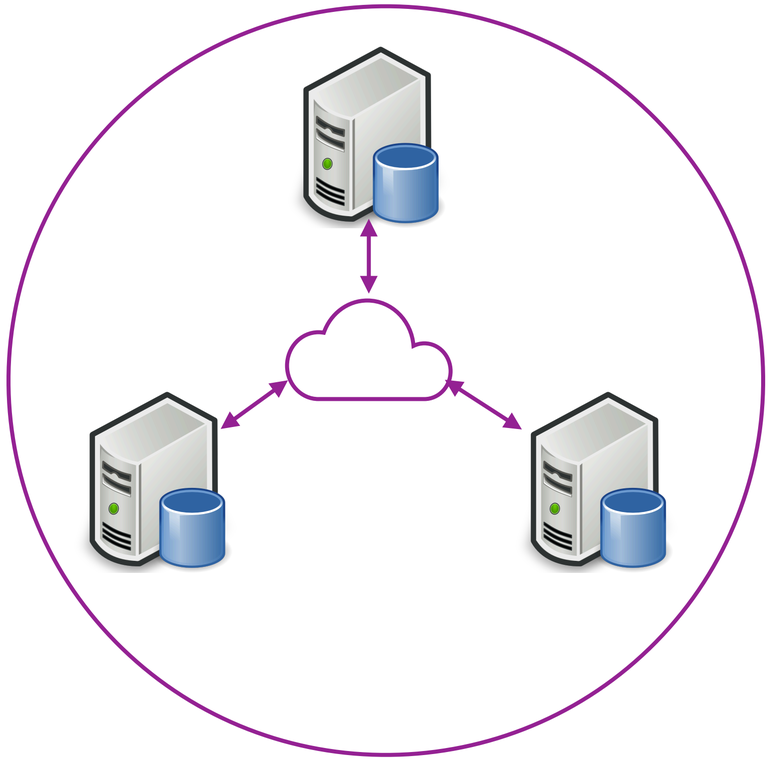


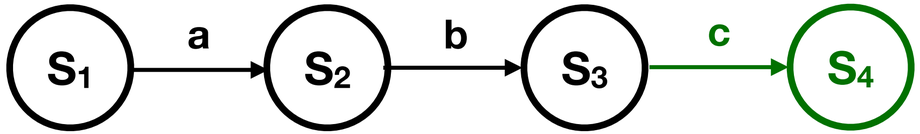
Distributed System + Consensus = ■RSM



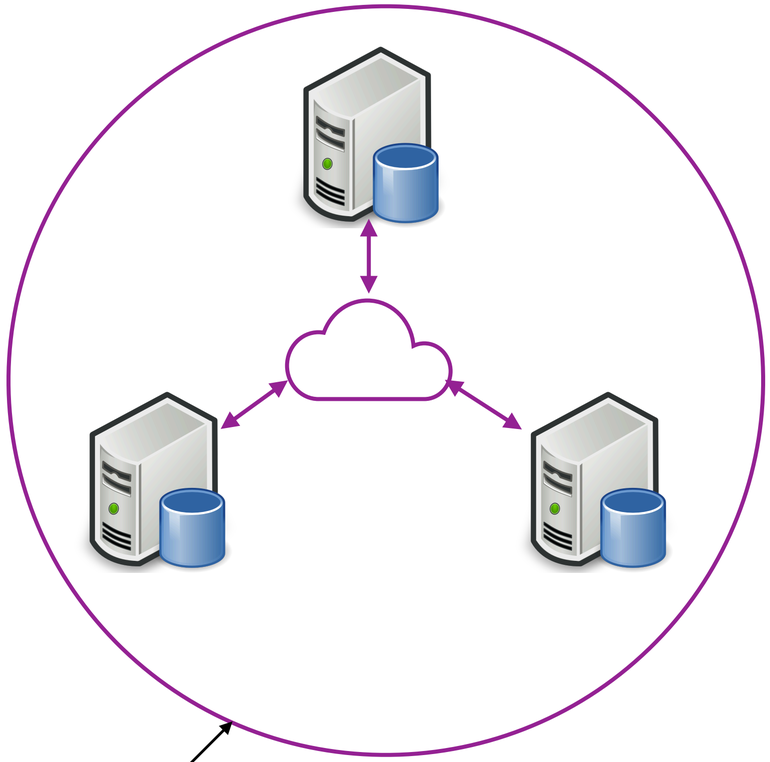


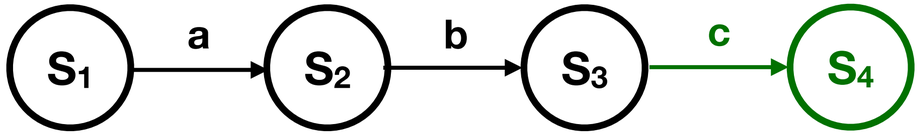
Distributed System + Consensus = ■RSM

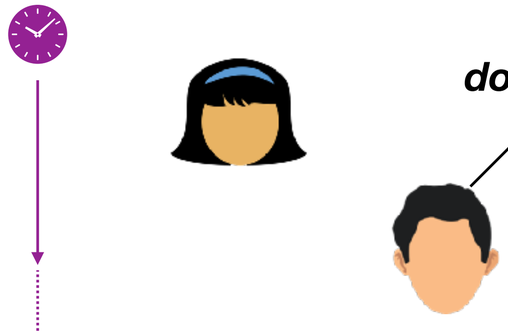




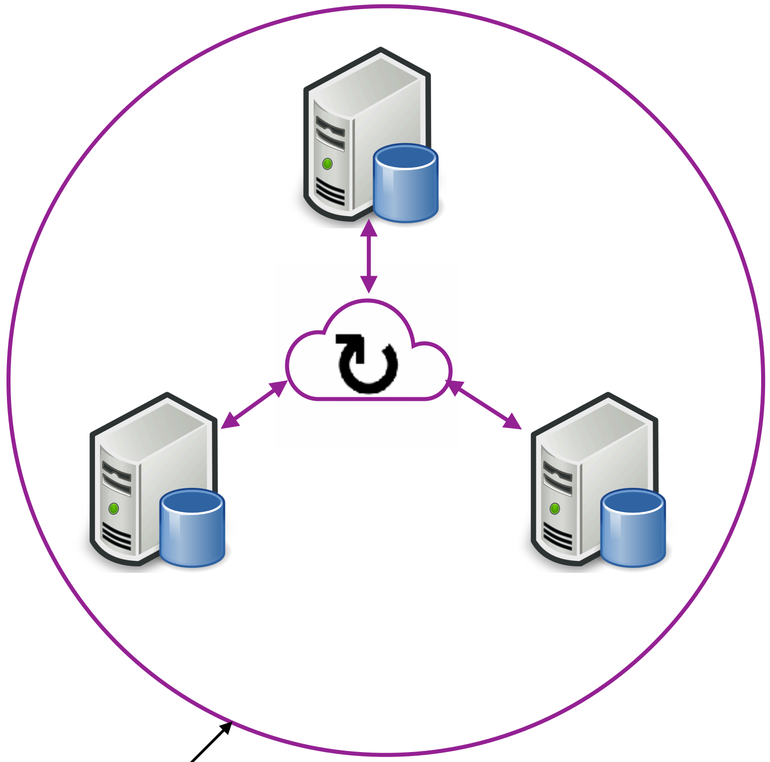
Distributed System + Consensus = ■RSM

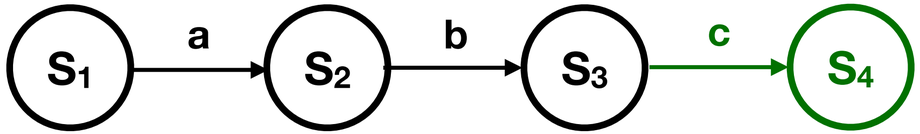


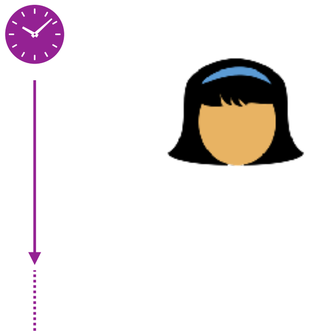




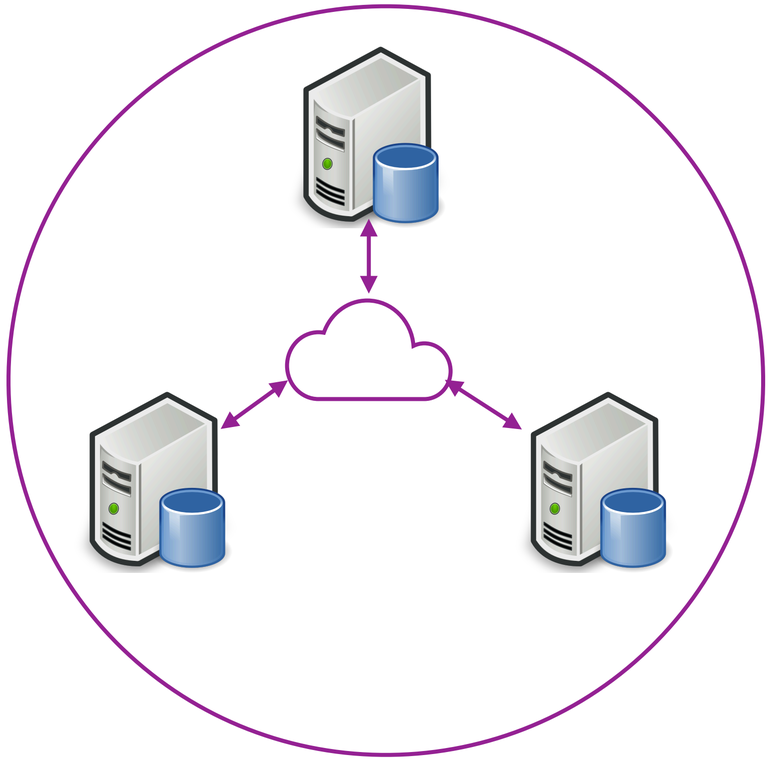
Distributed System + Consensus = ■RSM

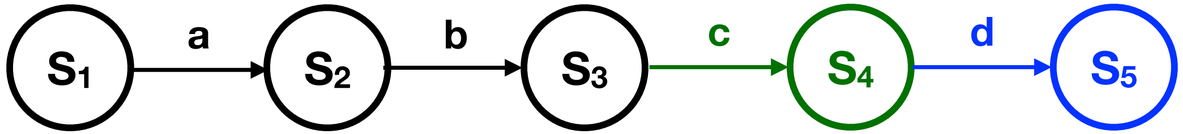


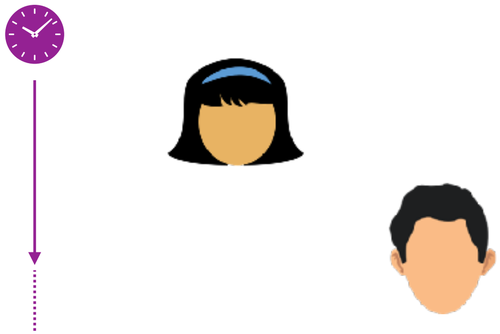




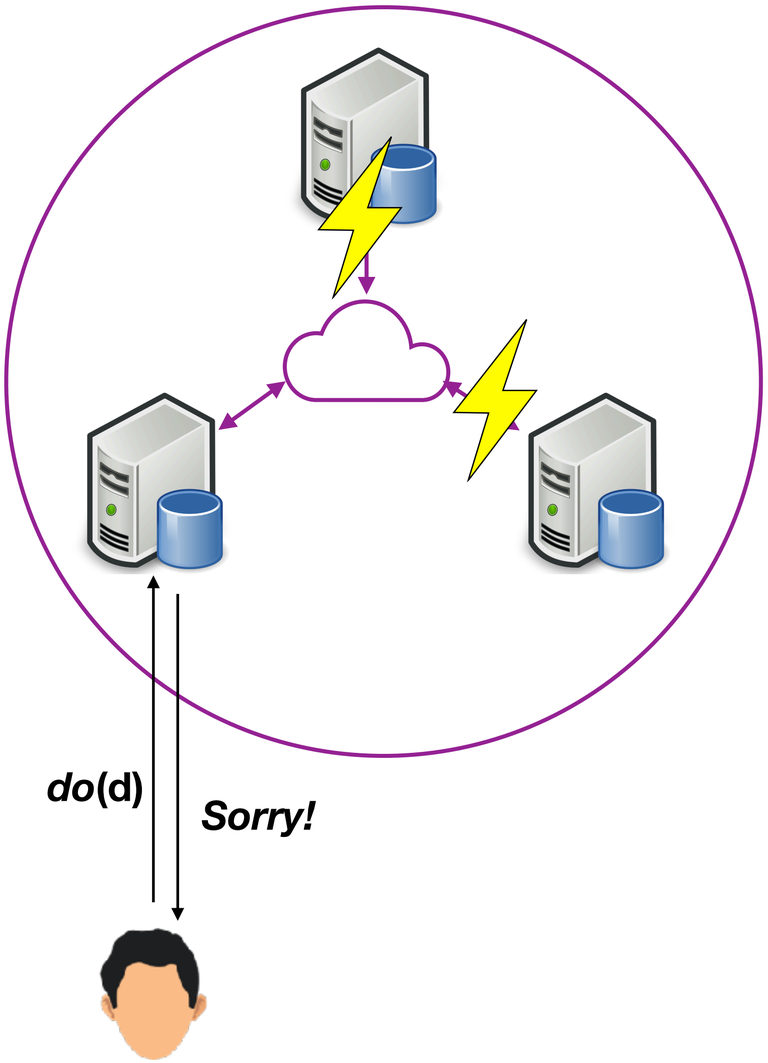
Distributed System + Consensus = ■RSM







Distributed System + Consensus = ■RSM



Consensus Protocols

- Paxos [Lamport 1998]
  - Multi-Paxos [Lamport 2001]
  - Disk Paxos [Gafni et al 2002]
  - Fast Paxos [Lamport 2006]
  - Stoppable Paxos [Malkhi et al 2008]
  - …
- Raft [Ongaro et al 2014]
  - Muti-Region Raft [Cockroach DB, TikV DB]
  - FlexiRaft [Yadav et al 2023]
- Viewstamped Replication [Oki et al 1988]
- Zookeeper [Junqueira et al 2011]
- Chubby [2006]
- …

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable. – Leslie Lamport (Turing award 2013).

- Negatively affect the *availability* (CAP theorem).
- Extremely complex ⟹ Error-prone implementations

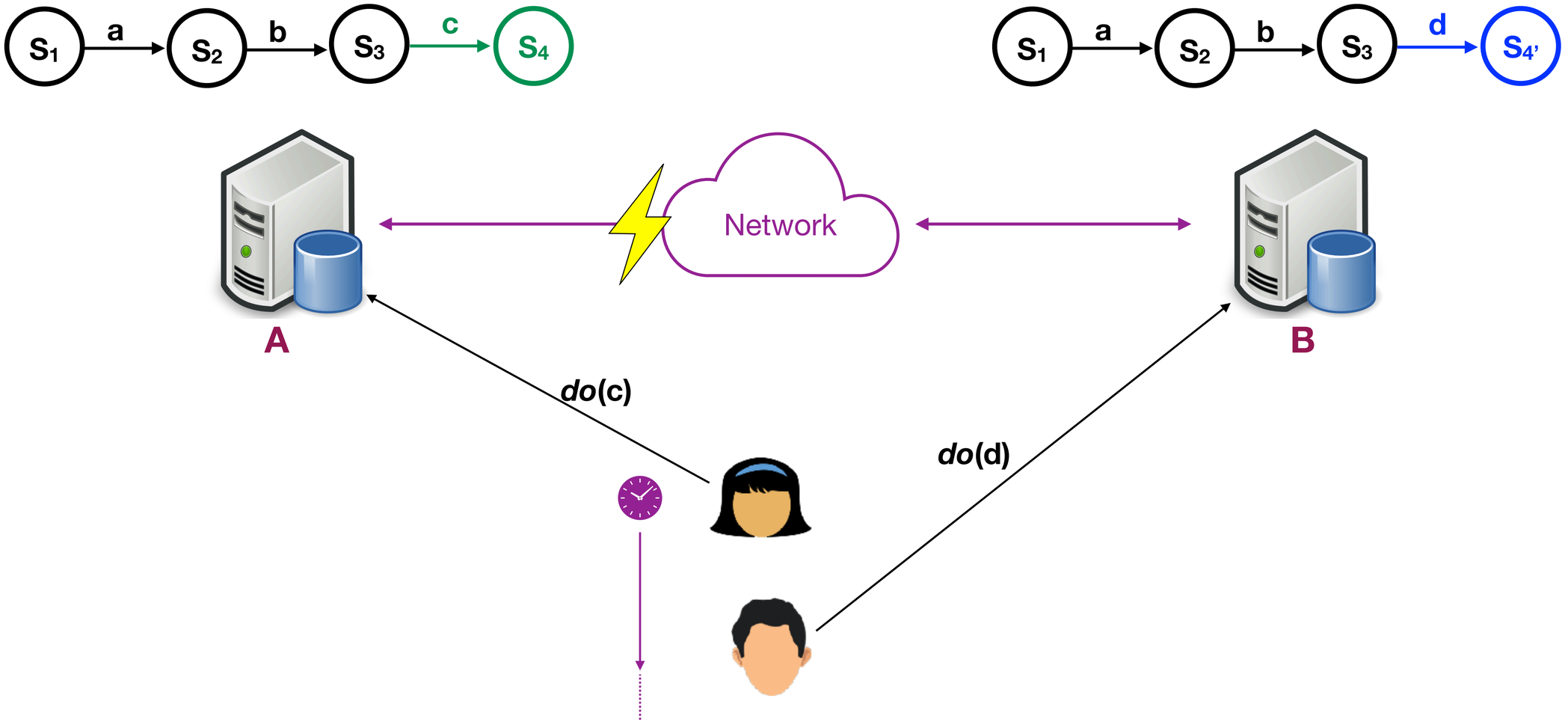
Key Questions

Can we avoid strong consistency / consensus in practice?

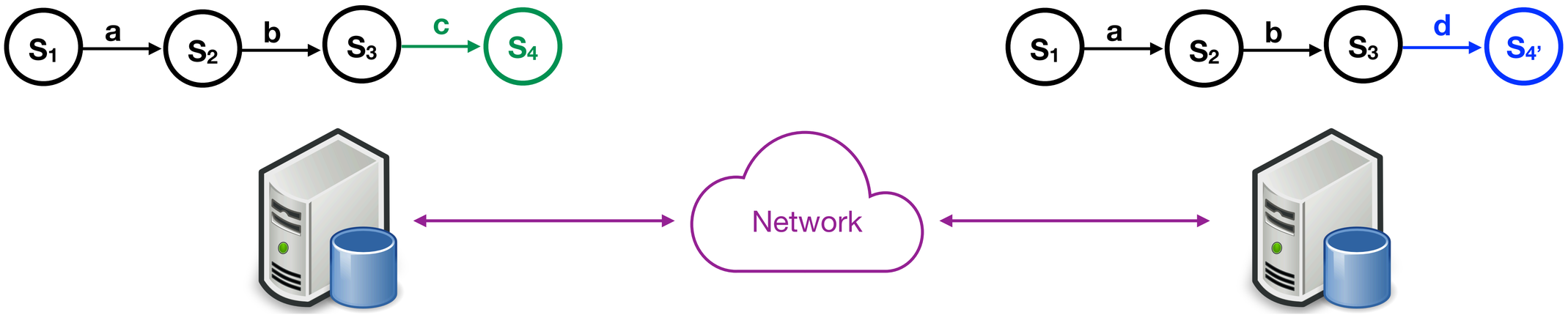
- Negatively affect the *availability* (CAP theorem).
- Extremely complex $\Longrightarrow$ Error-prone implementations

If not, can we make formal verification scale to implementations?

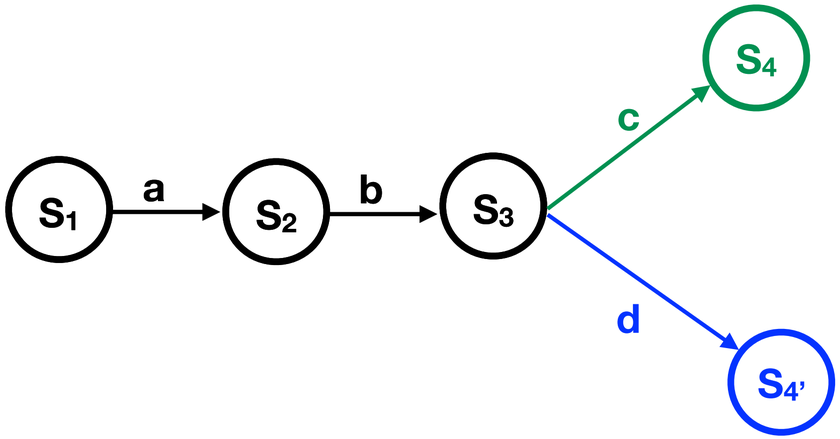
Divergence re-visited



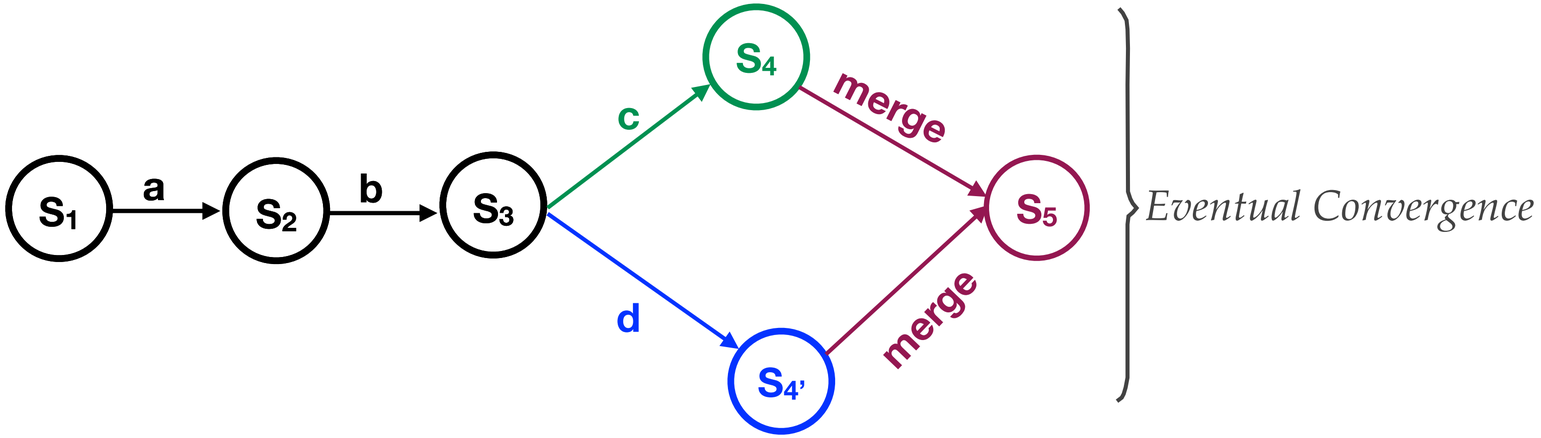
Divergence re-visited



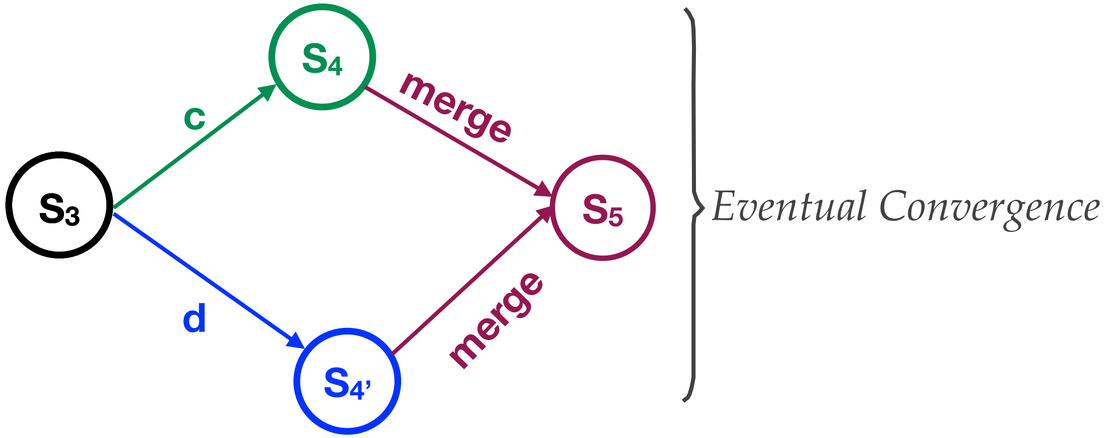
Divergence: Zoomed In

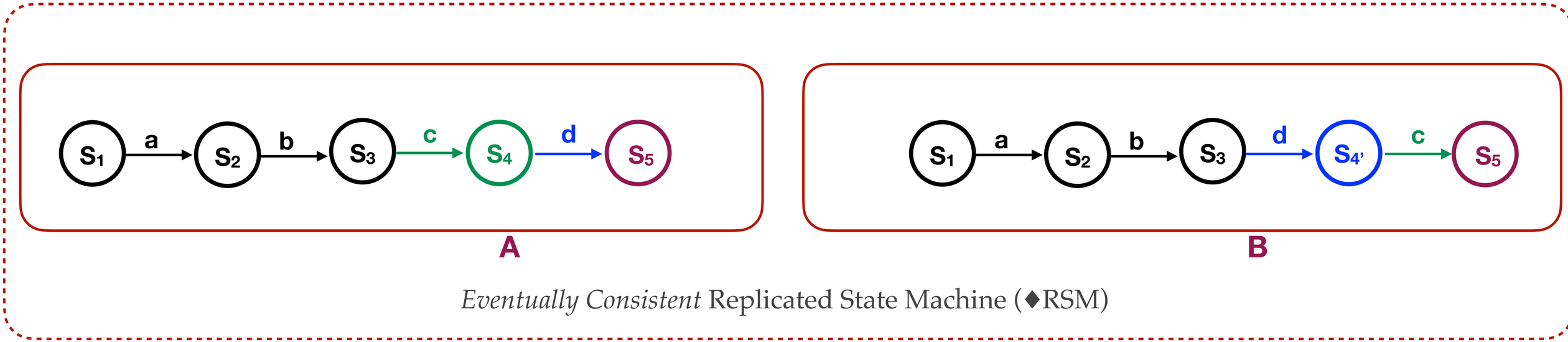


Merging Divergent States



Merging Divergent States

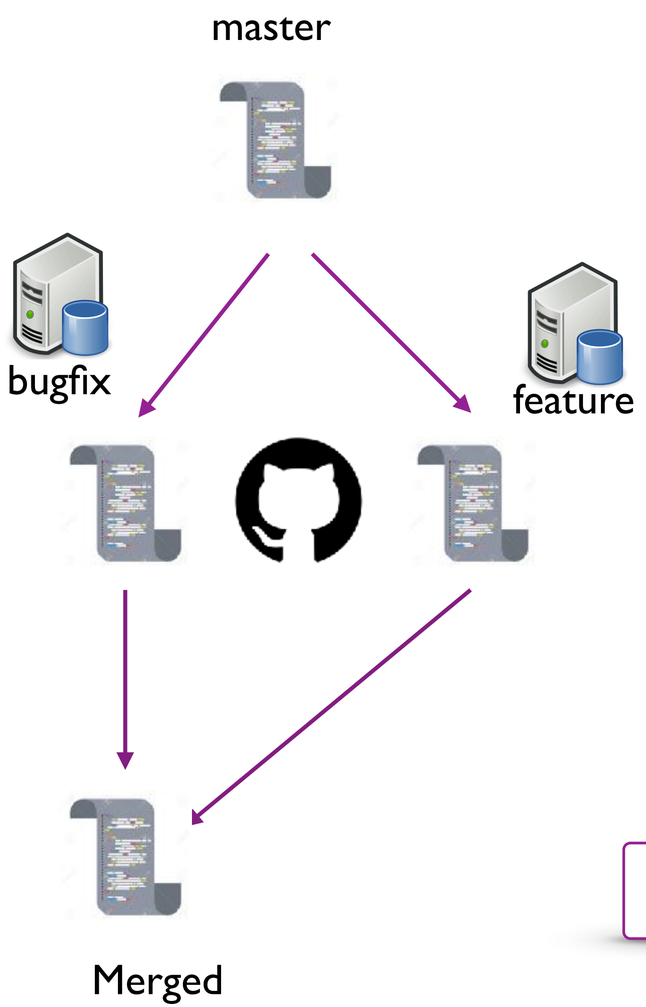




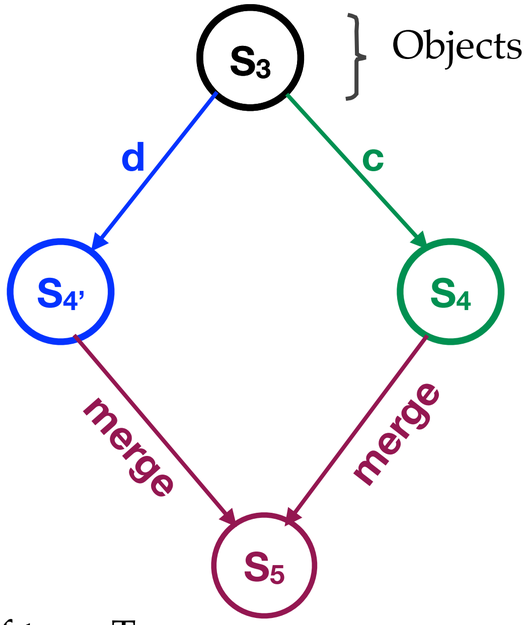
Eventually Consistent Replicated State Machine (♦RSM)

♦RSM in Practice

A familiar example: Git



In general:



To merge objects of type T

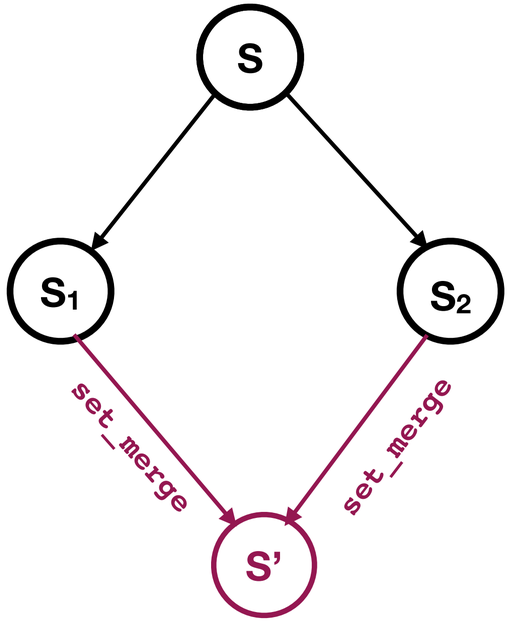
$$\overbrace{\texttt{merge}}(S_3, \underbrace{S_{4'}, S_4}) = S_5$$

Lowest Common Ancestor (LCA)

Concurrent versions

Merged version

A merge function for Set data type



$$\texttt{set\_merge}(s, s_1, s_2) = (s \cap s_1 \cap s_2) \cup (s_1 - s) \cup (s_2 - s)$$

- Merged set contains:
  - Elements in S that are not deleted in S_1 and S_2
  - Elements newly added in S_1
  - Elements newly added in S_2

 +  = 

Stdlib Set data type + `set_merge` = Mergeable Replicated Data Type (MRDT)

Generality & Practicality of MRDTs

Gowtham Kaki, Swarn Priya, KC Sivaramakrishnan, and Suresh Jagannathan. **Mergeable Replicated Data Types**, OOPSLA 2019.

Table 1. Characteristic relations for various data types

Data Type	Characteristic Relations
Binary Heap	Membership (R_{mem}), Ancestor ($R_{ans} \subseteq R_{mem} \times R_{mem}$)
Priority Queue	Membership (R_{mem})
Set	Membership (R_{mem})
Grap	
Func	
List	
Bina	
Bina	

Y **Hacker News** new | threads | past | comments | ask | show

Mergeable replicated data types – Part I (acolyer.org)

104 points by telotortium 75 days ago | hide | past | web | un-favorite | 21 comments

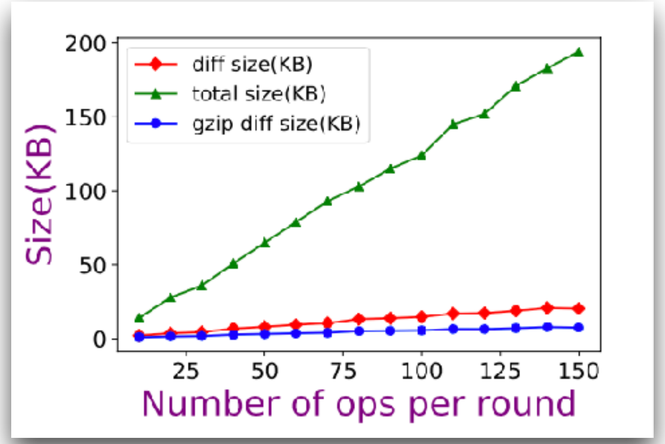
Sens
data
them

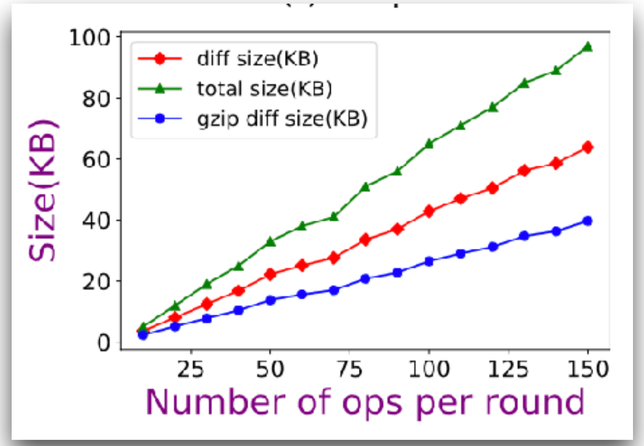
Mergeable replicated data types – Part I

NOVEMBER 25, 2019

Mergeable replicated data types Kaki et al., *OOPSLA'19*

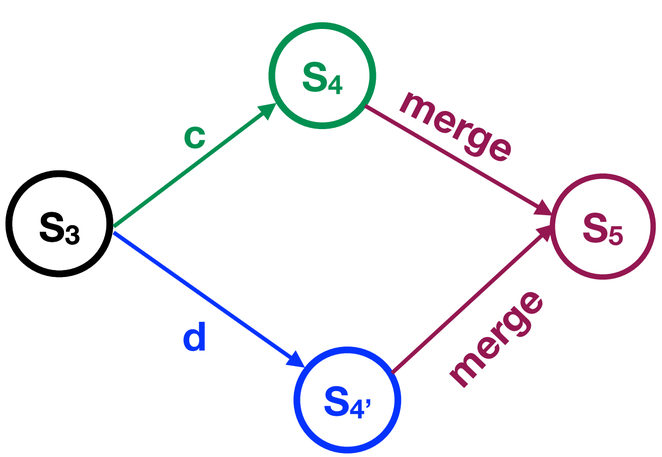
This paper was published at OOPSLA, but perhaps it's amongst the distributed systems community that I expect there to be the greatest interest.





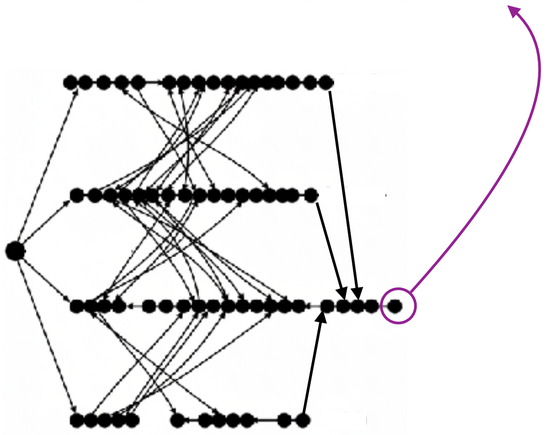
Network overhead imposed by state merges can be kept low by efficiently computing *diffs* (à la Git)

Theoretical Guarantees of MRDTs

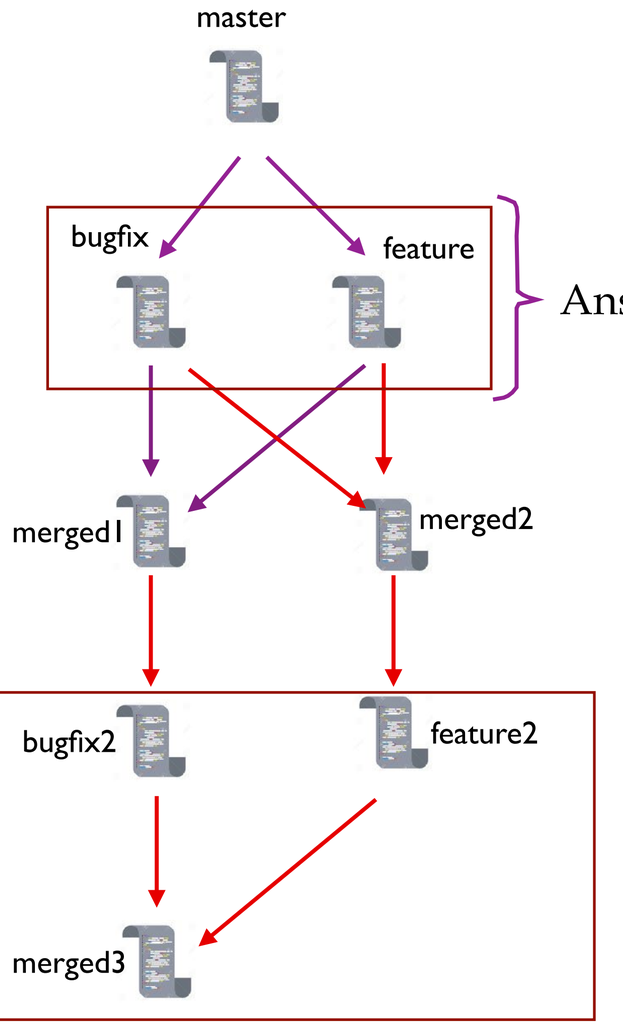


States have to converge.

But, do they converge in practice?



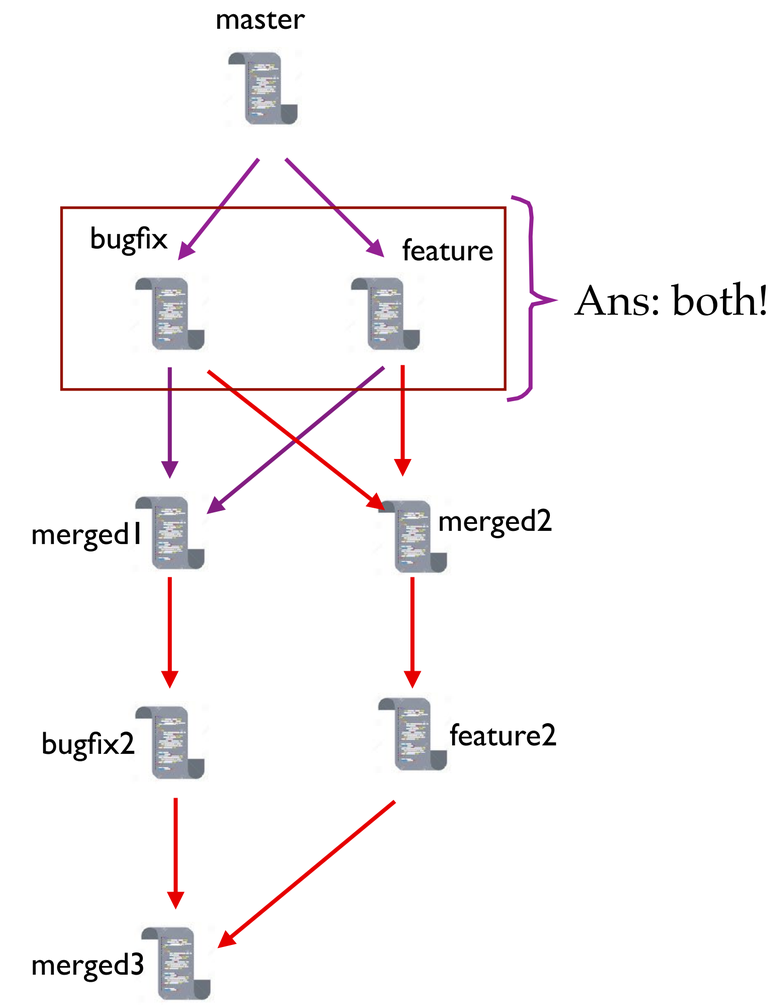
A fly in the ointment:



Ans: both!

What is the LCA for this merge?

Ill-formed Version Graphs



Ans: both!

What is the LCA for this merge?

From my blog (https://tycon.github.io/git-inconsistencies.html)

Vagaries of Git Merge

Git's merge algorithm seems to have inexplicable semantics leading to some interesting cases. I describe a couple of examples below. The bottom line is that:

1. Git merge is inconsistent: Merge result seems to depend not just on *what* versions were merged, but also *how* they were merged.

2. Git merge is unintuitive: The result of a successful (auto) merge may not necessarily be what we expect.

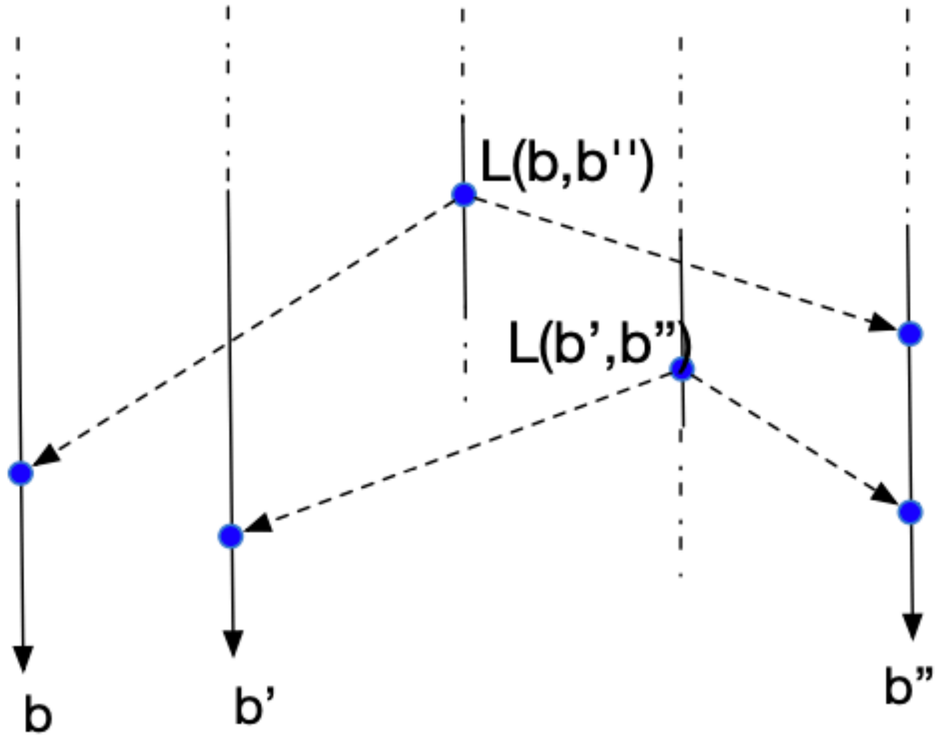
Git's recursive merging strategy™

What do we do? Options:

- Constrain merge function s.t. Git-style recursive merging is safe, i.e., guarantees convergence.
- Constrain version graphs s.t. ill-formed graphs never manifest.

Constraining Version Graphs

Key idea: Enforce a semi-lattice structure



For every branch b'' in the system, the LCA of b'' with merging branches b and b' should be causally related, i.e:

$L(b,b'') \rightarrow^* L(b',b'') \ \vee \ L(b',b'') \rightarrow^* L(b,b'')$.

Constraining Version Graphs

RunTime-Assisted Convergence in Replicated Data Types

Gowtham Kaki
University of Colorado Boulder
Boulder, USA
gowtham.kaki@colorado.edu

Prasanth Prahladan
University of Colorado Boulder
Boulder, USA
prasanth.prahladan@colorado.edu

Nicholas V. Lewchenko
University of Colorado Boulder
Boulder, USA
nile1033@colorado.edu

[PLDI 2022]

A fault-tolerant distributed algorithm to orchestrate *well-formed* distributed executions of MRDTs.

$$\boxed{(G, N, C, H, L) \longrightarrow (G', N', C', H', L')}$$

$$\frac{b \in dom(H) \quad v \notin V \quad i \notin codom(C)}{((V,E), N, C, H, L) \longrightarrow (V \cup \{v\},\ E \cup \{H(b) \xrightarrow{c} v\},\ N[v \mapsto n],\ C[v \mapsto \{i\} \cup C(H(b))],\ H[b \mapsto v], L)} \text{ [Commit]}$$

$$\frac{b \in dom(H) \quad b' \notin dom(H) \quad v \notin V}{\begin{array}{l}((V,E), N, C, H, L) \longrightarrow (V \cup \{v\},\ E \cup \{H(b) \xrightarrow{f} v\},\ N[v \mapsto N(H(b))],\ C[v \mapsto C(H(b))], \\ \qquad H[b' \mapsto v],\ L[(b', b) \mapsto H(b)][\{(b', b'') \mapsto L(b, b'') \mid b'' \neq b\}])\end{array}} \text{ [Fork]}$$

$$\frac{\begin{array}{c}b, b' \in dom(H) \quad C(H(b)) \supset C(L(b,b')) \quad C(H(b')) \supset C(L(b,b')) \\ \forall (b'' \in dom(H)).\ L(b,b'') \rightarrow^* L(b',b'') \ \vee\ L(b',b'') \rightarrow^* L(b,b'') \\ n = \text{merge}(N(L(b,b')),\ N(H(b)),\ N(H(b'))) \quad v \notin V\end{array}}{\begin{array}{l}((V,E), N, C, H, L) \longrightarrow (V \cup \{v\},\ E \cup \{H(b) \xrightarrow{m} v, H(b') \xrightarrow{m} v\}, \\ \qquad N[v \mapsto n],\ C[v \mapsto C(H(b)) \cup C(H(b'))],\ H[b \mapsto v], \\ \qquad L[(b,b') \mapsto H(b')][\{(b,b'') \mapsto L(b',b'') \mid L(b,b'') \rightarrow^* L(b',b'')\}])\end{array}} \text{ [MergeFwd]}$$

$$\frac{\begin{array}{c}b, b' \in dom(H) \quad C(H(b)) = C(L(b,b')) \quad C(H(b')) \supset C(L(b,b')) \\ \forall (b'' \in dom(H)).\ L(b,b'') \rightarrow^* L(b',b'') \ \vee\ L(b',b'') \rightarrow^* L(b,b'') \quad v \notin V\end{array}}{\begin{array}{l}((V,E), N, C, H, L) \longrightarrow (V \cup \{v\},\ E \cup \{H(b) \xrightarrow{m} v, H(b') \xrightarrow{m} v\}, \\ \qquad N[v \mapsto N(H(b'))],\ C[v \mapsto C(H(b'))],\ H[b \mapsto v], \\ \qquad L[(b,b') \mapsto H(b')][\{(b,b'') \mapsto L(b',b'') \mid L(b,b'') \rightarrow^* L(b',b'')\}])\end{array}} \text{ [FastFwd]}$$

Theorem 3.7 (Convergence). *In every reachable state* $\Delta = ((V,E), N, C, H, L)$ *of the abstract machine: For all distinct* $b_1, b_2 \in dom(H)$, *and* $v_1, v_2 \in V$ *such that* $v_1 = H(b_1)$ *and* $v_2 = H(b_2)$, *the following is true:* $C(v_1) = C(v_2) \Rightarrow N(v_1) = N(v_2)$.

MACHINE CHECKED

Theorem 3.8 (Progress). *Every reachable state* $\Delta = ((V,E), N, C, H, L)$ *of the abstract machine is either:*

- *A "quiescent" state, where:* $\forall b_1, b_2 \in dom(H).\ C(H(b_1)) = C(H(b_2))$, *Or*
- *An "unstuck" state, where there exist* $b_1, b_2 \in dom(H)$ *that satisfy the pre-conditions of* Merge *or* FastFwd *rules, i.e.,* b_1 *and* b_2 *are mergeable.*

Quark

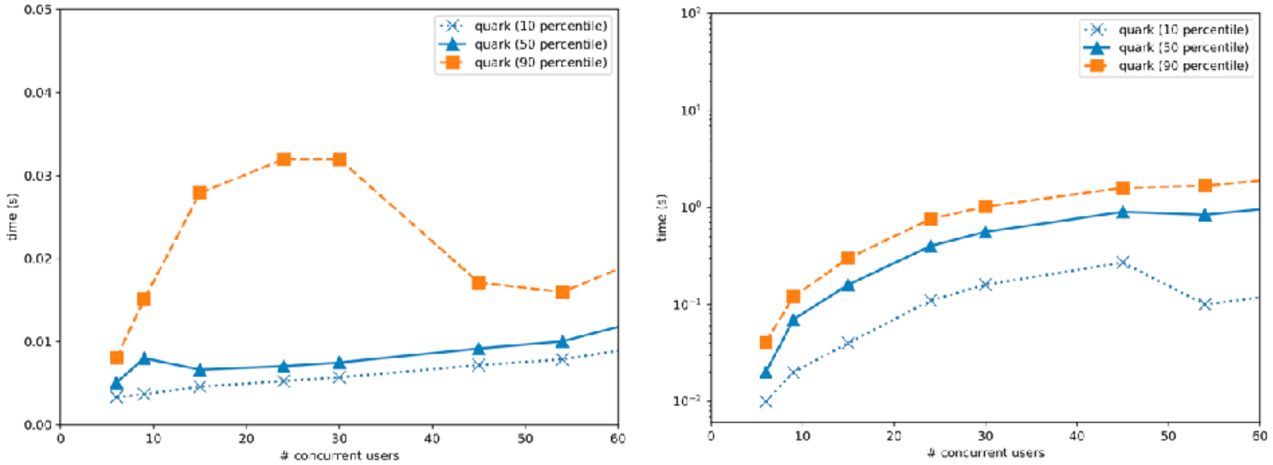
RunTime-Assisted Convergence in Replicated Data Types

Gowtham Kaki
University of Colorado Boulder
Boulder, USA
gowtham.kaki@colorado.edu

Prasanth Prahladan
University of Colorado Boulder
Boulder, USA
prasanth.prahladan@colorado.edu

Nicholas V. Lewchenko
University of Colorado Boulder
Boulder, USA
nile1033@colorado.edu

[PLDI 2022]



Implemented in a distributed runtime called:

Quark

https://github.com/cuplv/Quark

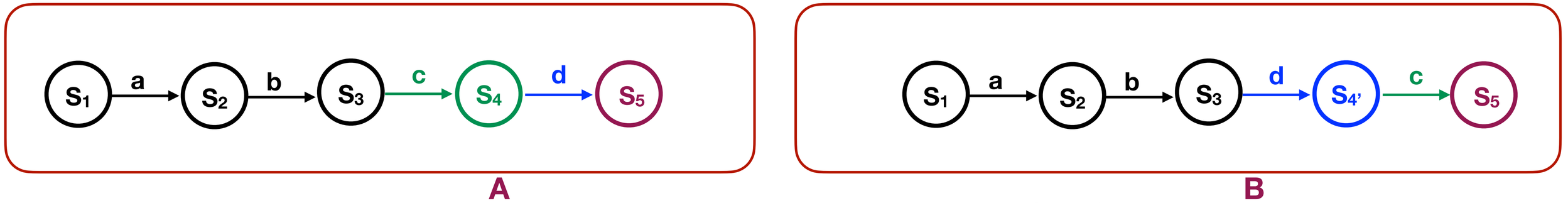
Key Takeaway

For any data type T, Quark lets the programmer implement a T ♦RSM that is:

- Fault-tolerant and scalable, and
- Guaranteed to converge without needing a type-specific convergence proof.

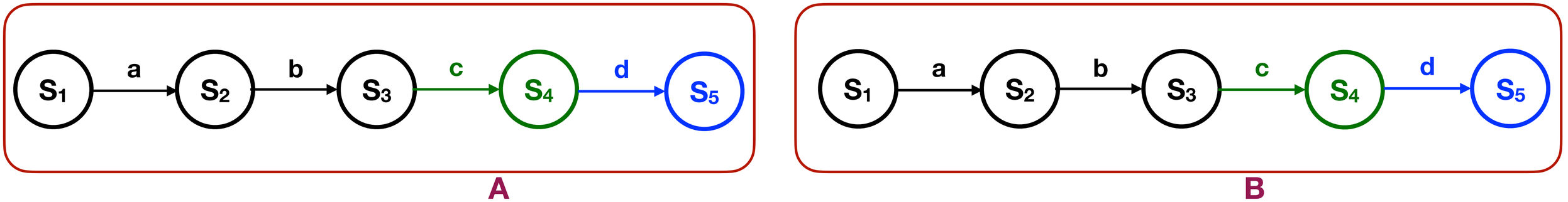
But, Convergence ≠ Consensus

Convergence is about *eventually* agreeing



Eventually Consistent Replicated State Machine (♦RSM)

Consensus is about *never* disagreeing



Strongly Consistent Replicated State Machine (■RSM)

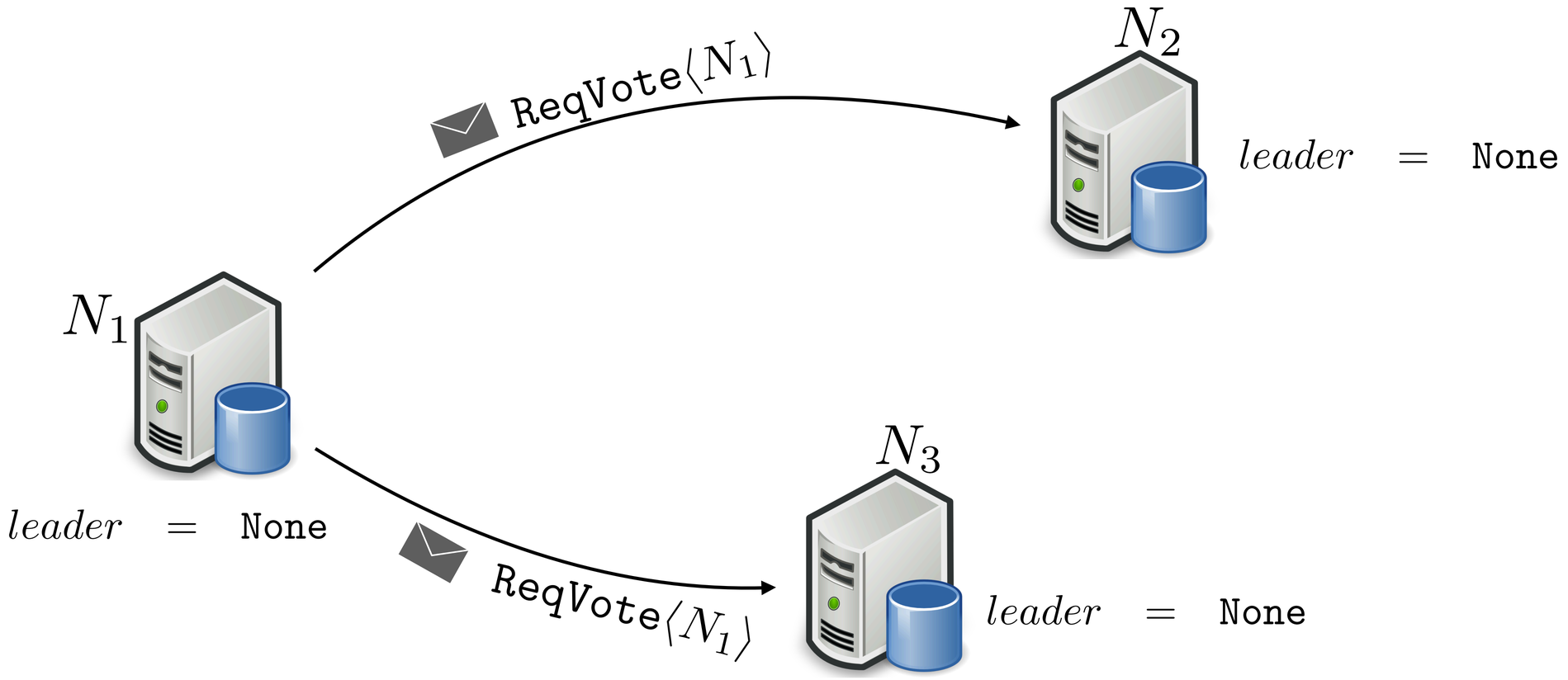
Key Questions

Can we avoid strong consistency / consensus in practice?

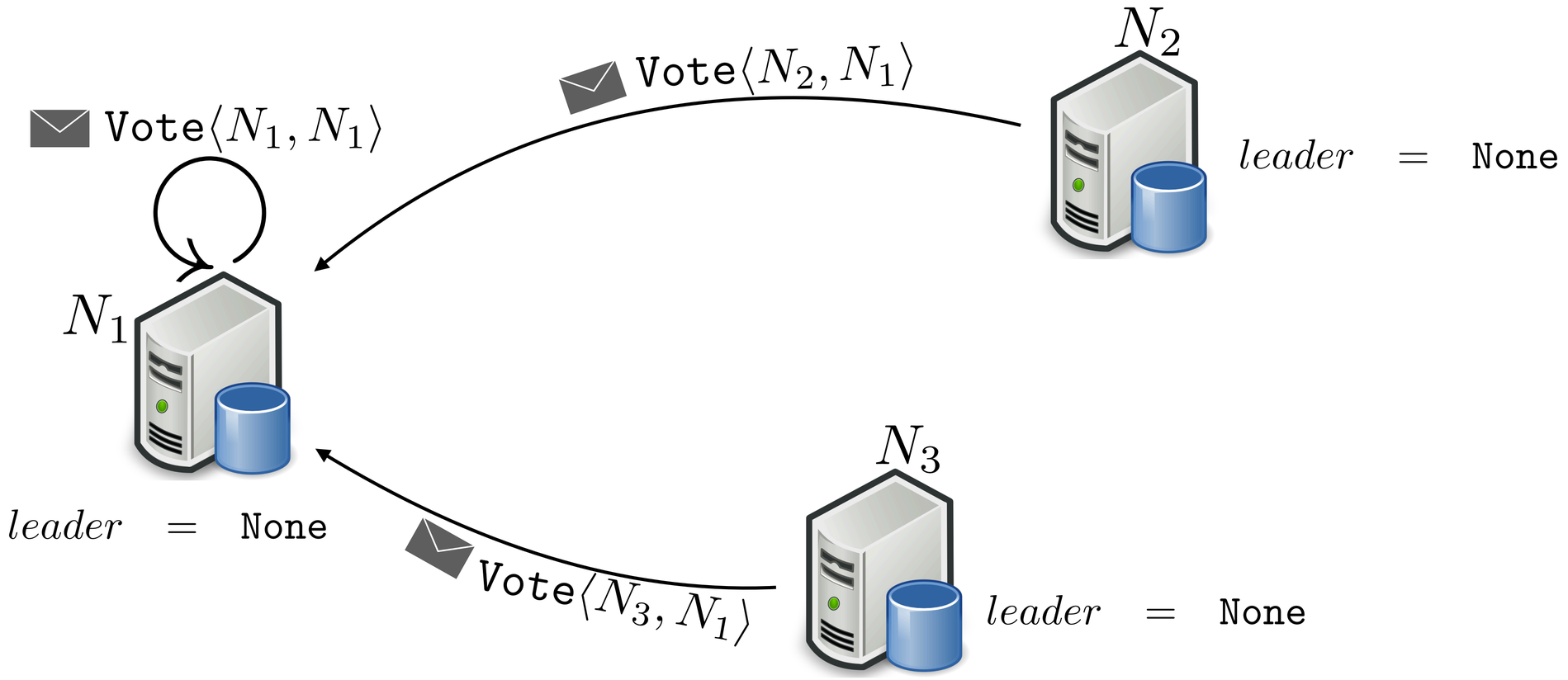
- Negatively affect the *availability* (CAP theorem).
- Extremely complex $\Longrightarrow$ Error-prone implementations

If not, can we make formal verification scale to implementations?

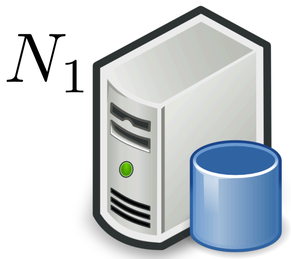
A Simple Consensus Protocol: Leader Election



Leader Election

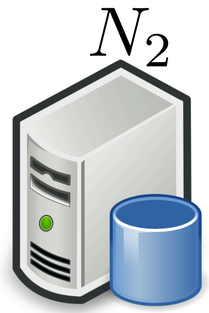


Leader Election

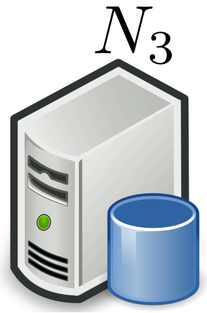


$$
\begin{array}{lcl}
leader & = & \texttt{None} \\
votes & = & \{N_1, N_2, N_3\} \\
hasVoted & = & true
\end{array}
$$

Quorum

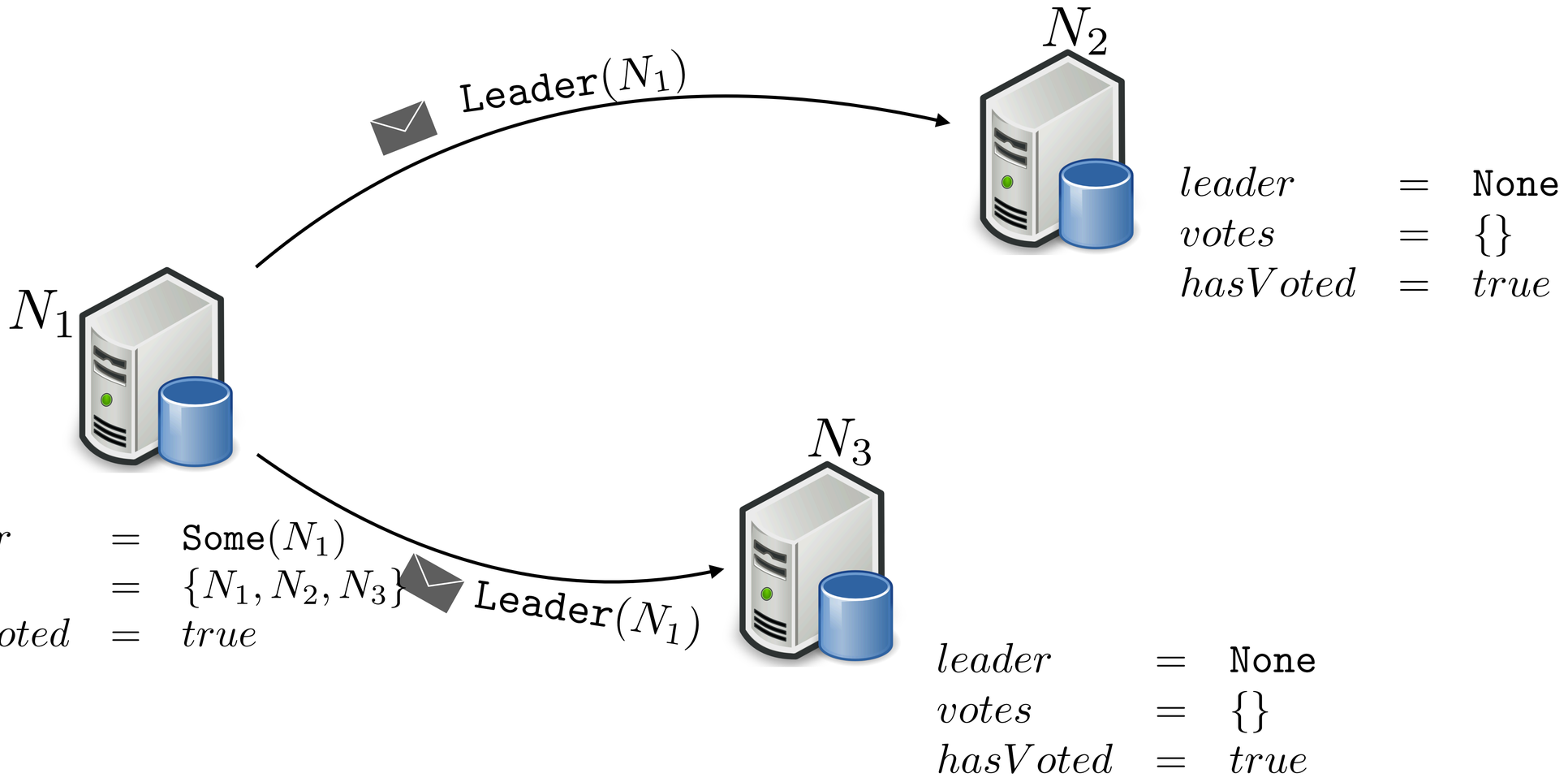


$$
\begin{array}{lcl}
leader & = & \texttt{None} \\
votes & = & \{\} \\
hasVoted & = & true
\end{array}
$$

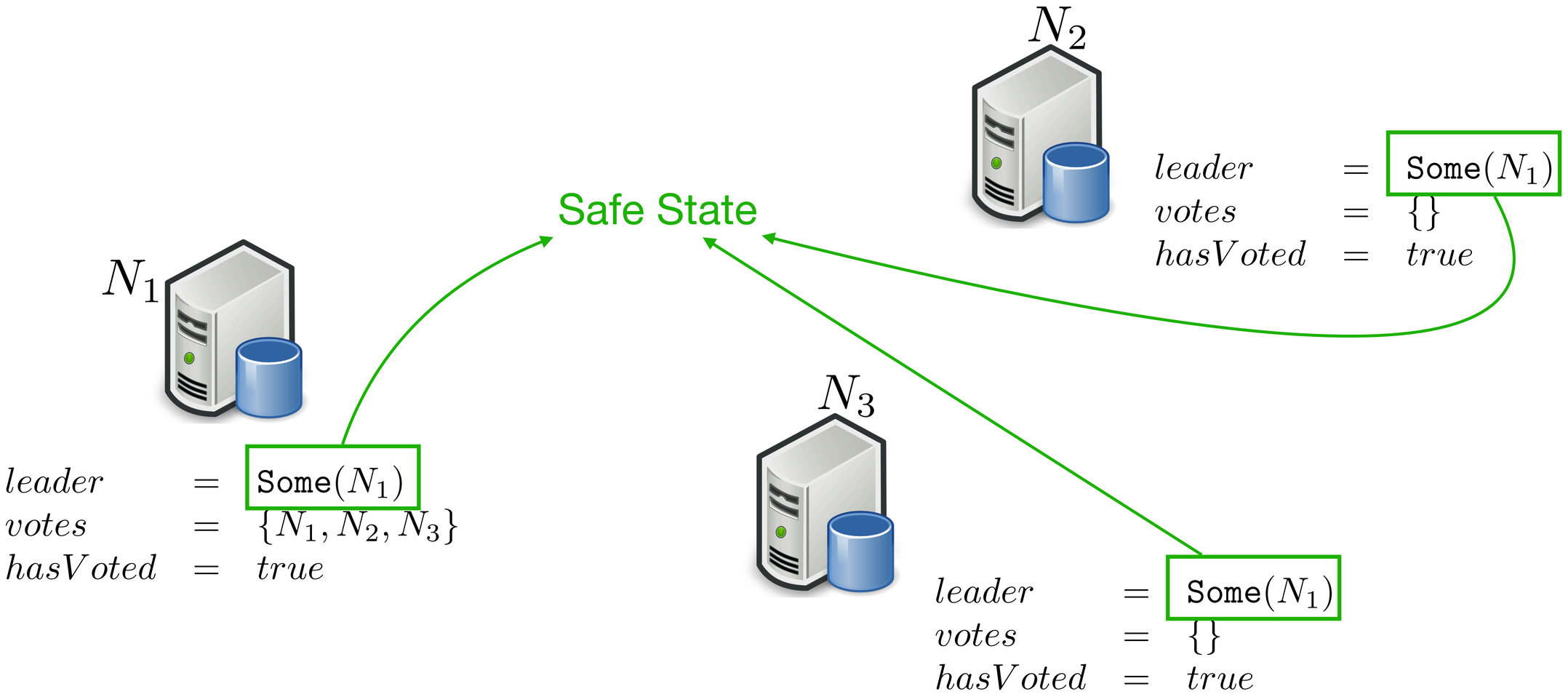


$$
\begin{array}{lcl}
leader & = & \texttt{None} \\
votes & = & \{\} \\
hasVoted & = & true
\end{array}
$$

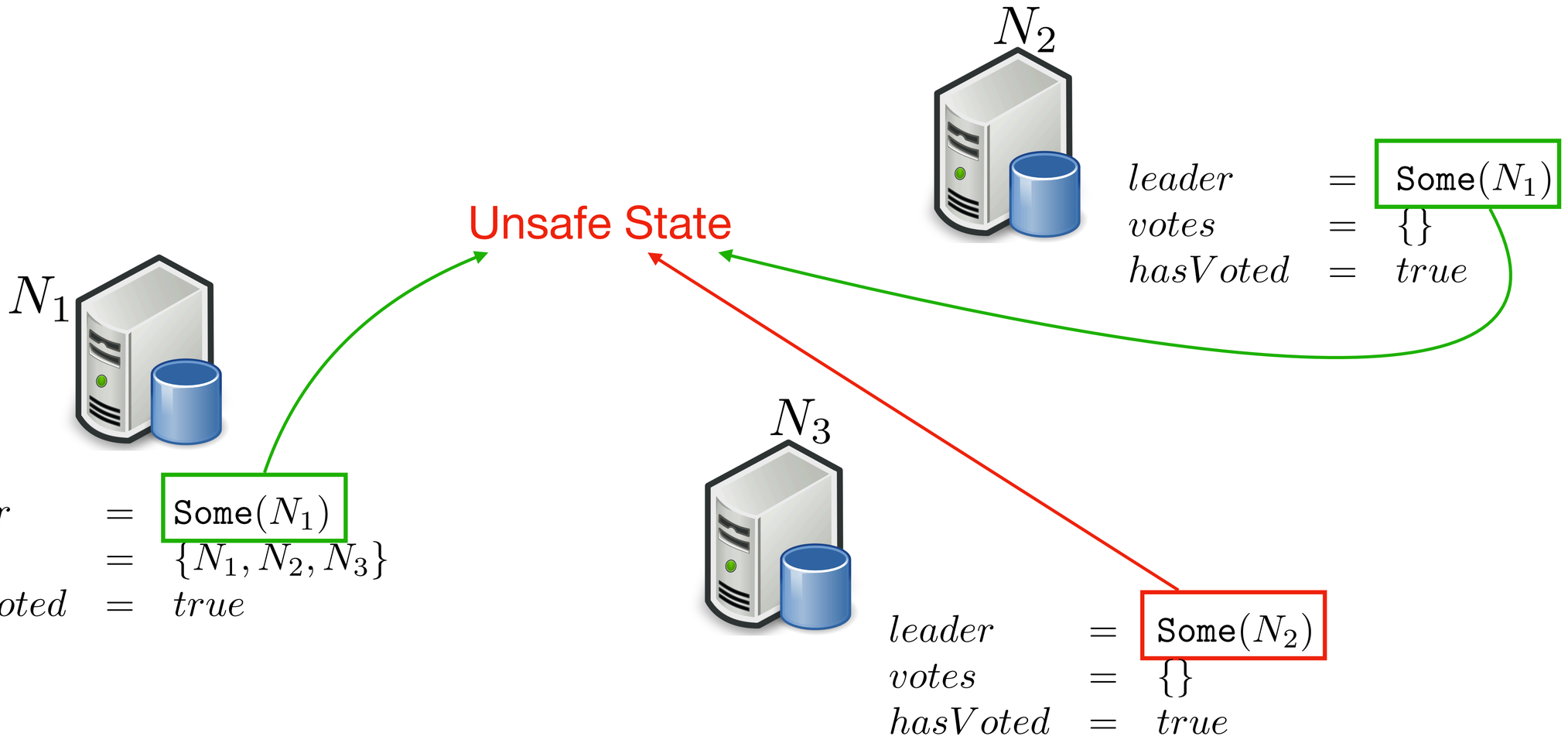
Leader Election



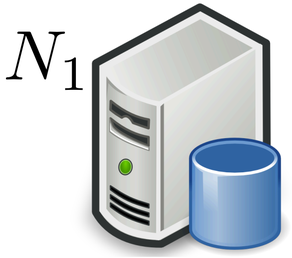
Leader Election



Leader Election

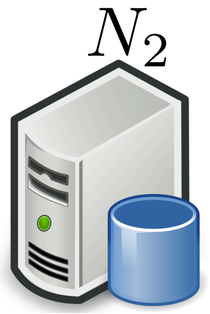


Leader Election: Safety Spec



$$\begin{array}{lcl} leader & = & \boxed{\texttt{Some}(N_1)} \\ votes & = & \{N_1, N_2, N_3\} \\ hasVoted & = & true \end{array}$$

$$\begin{array}{lcl} leader & = & \boxed{\texttt{Some}(N_2)} \\ votes & = & \{\} \\ hasVoted & = & true \end{array}$$



$$\begin{array}{lcl} leader & = & \boxed{\texttt{Some}(N_1)} \\ votes & = & \{\} \\ hasVoted & = & true \end{array}$$

$$\begin{array}{rcl} \phi & = & \forall (N_1, N_2, N_3, N_4).\ leader[N_1] = \texttt{Some}(N_2) \\ & & \quad \wedge\ leader[N_3] = \texttt{Some}(N_4) \ \Rightarrow\ N_2 = N_4 \end{array}$$

Proving Safety of Leader Election: Inductive Invariant

$$\mathbb{I} = \bigwedge_{i=1}^{6} \varphi_i$$

$$\varphi_1 \quad = \quad \forall N.\ \texttt{Leader}\langle N \rangle \in msgs \ \Rightarrow\ votes[N] \in Quorum$$

$$\varphi_2 \quad = \quad \forall N_1, N_2.\ N_1 \in votes[N_2] \ \Rightarrow\ \texttt{Vote}\langle N_1, N_2 \rangle \in msgs$$

$$\varphi_3 \quad = \quad \forall N_1, N_2.\ \texttt{Vote}\langle N_1, N_2 \rangle \in msgs \ \Rightarrow\ hasVoted[N_1] = true$$

$$\varphi_4 \quad = \quad \forall N_1, N_2, N_3.\ \texttt{Vote}\langle N_1, N_2 \rangle \in msgs \ \wedge\ \texttt{Vote}\langle N_1, N_3 \rangle \in msgs \ \Rightarrow\ N_2 = N_3$$

$$\varphi_5 \quad = \quad \forall N_1, N_2.\ leader[N_1] = N_2 \ \Rightarrow\ \texttt{Leader}\langle N_2 \rangle \in msgs$$

$$\varphi_6 \quad = \quad \forall N_1, N_2.\ \texttt{Leader}\langle N_1 \rangle \in msgs \ \wedge\ \texttt{Leader}\langle N_2 \rangle \in msgs \ \Rightarrow\ N_1 = N_2$$

1. A Leader message exists only if the sender received a quorum of vote

2. A node counts a vote only if the corresponding Vote message exists

3. If a node has voted then its *hasVoted* flag is set

4. A node only ever casts one vote

5. A node sets a leader only if the corresponding Leader message exists

6. A Leader message is only ever broadcast for one node

Inductive Invariant $\Longrightarrow$ Undecidability

$$\varphi_2 \quad = \quad \forall N_1, N_2.\ N_1 \in votes[N_2] \ \Rightarrow\ \texttt{Vote}\langle N_1, N_2\rangle \in msgs$$

Is actually written as:

$$\exists m.\ m \in msgs \ \wedge\ \texttt{typeOf}(m) = \texttt{Vote} \ \wedge\ \texttt{sender}(m) = N_1 \ \wedge\ \texttt{receiver}(m) = N_2$$

$\forall^*\exists^*$ **Quantifier alternation is undecidable.**

Undecidability ⟹ Manual Proofs

Problem: Too Much Work!

- Hawblitzel et al. 2015 *IronFleet: Proving Practical Distributed Systems Correct.*
  "…developing IronFleet and applying it to two real systems required approximately 3.7 person-years"
- Wilcox et al. 2015 *Verdi: A Framework for Implementing and Formally Verifying Distributed Systems.*
- Woos et al. 2016 *Planning for Change in a Formal Verification of the Raft Consensus Protocol.*
  "…required iteratively discovering and proving 90 system invariants"
  "…530 lines of code and 50,000 lines of proof."
- Padon et al. 2017 *Paxos made EPR: Decidable Reasoning about Distributed Protocols.*
- Taube et al. 2018 *Modularity for Decidability of Deductive Verification with Applications to Distributed Systems.*
  "…took approximately 3 person-months …300 [lines] of invariants and ghost code."
- Gleissenthall et al. 2019. *Pretend Synchrony: Synchronous Verification of Asynchronous Distributed Systems.*

How do we avoid this?

Network is the root of complexity

$$\phi \;=\; \forall(N_1, N_2, N_3, N_4).\; leader[N_1] = \texttt{Some}(N_2) \wedge leader[N_3] = \texttt{Some}(N_4) \;\Rightarrow\; N_2 = N_4$$

$$\varphi_1 \;=\; \forall N.\; \texttt{Leader}\langle N\rangle \in msgs \;\Rightarrow\; votes[N] \in Quorum$$

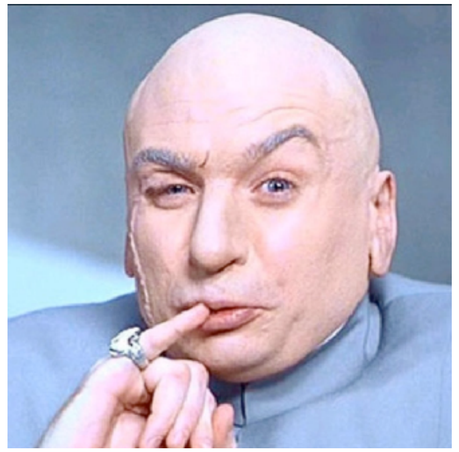
$$\varphi_2 \;=\; \forall N_1, N_2.\; N_1 \in votes[N_2] \;\Rightarrow\; \texttt{Vote}\langle N_1, N_2\rangle \in msgs$$

$$\varphi_3 \;=\; \forall N_1, N_2.\; \texttt{Vote}\langle N_1, N_2\rangle \in msgs \;\Rightarrow\; hasVoted[N_1] = true$$

$$\varphi_4 \;=\; \forall N_1, N_2, N_3.\; \texttt{Vote}\langle N_1, N_2\rangle \in msgs \;\wedge\; \texttt{Vote}\langle N_1, N_3\rangle \in msgs \;\Rightarrow\; N_2 = N_3$$

$$\varphi_5 \;=\; \forall N_1, N_2.\; leader[N_1] = N_2 \;\Rightarrow\; \texttt{Leader}\langle N_2\rangle \in msgs$$

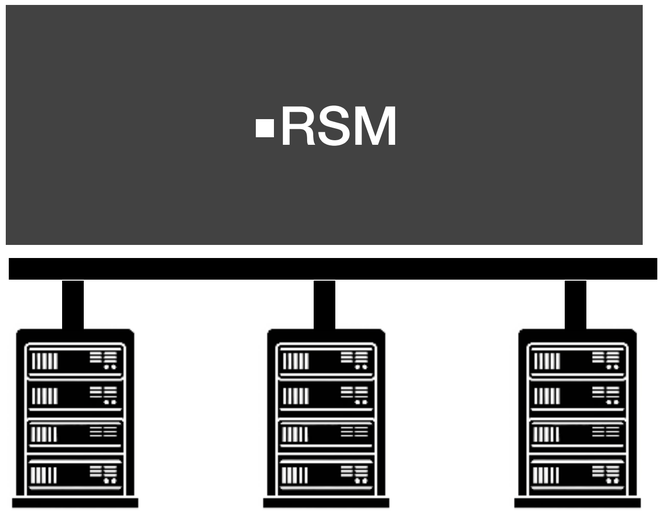
$$\varphi_6 \;=\; \forall N_1, N_2.\; \texttt{Leader}\langle N_1\rangle \in msgs \;\wedge\; \texttt{Leader}\langle N_2\rangle \in msgs \;\Rightarrow\; N_1 = N_2$$



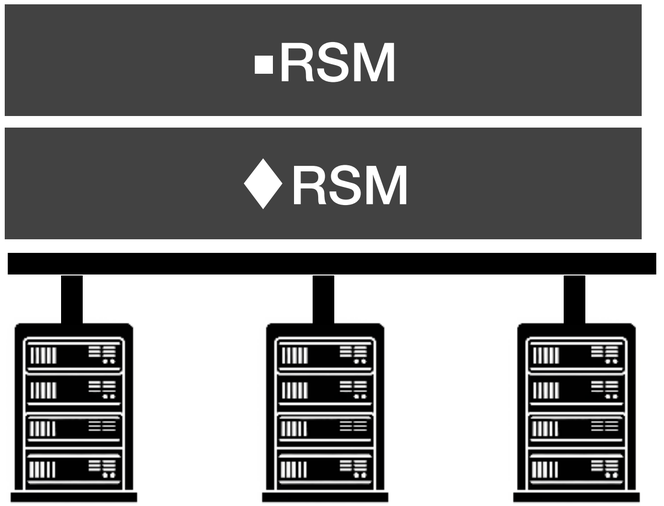
Network is the root of complexity

$msgs$

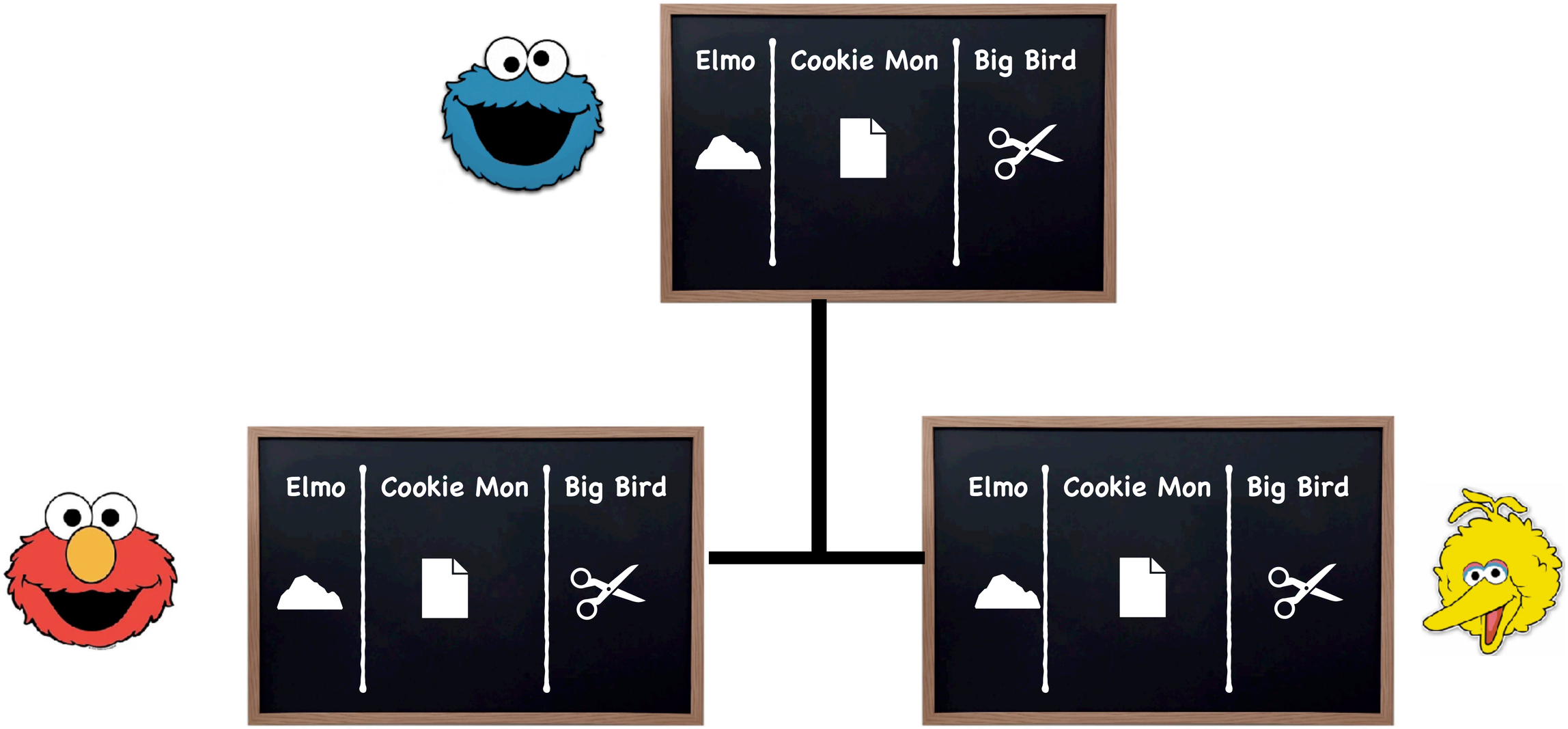
Instead of the conventional *monolithic* approach…



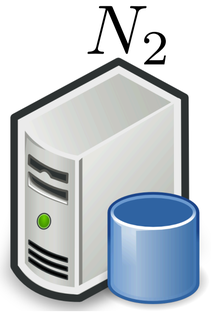
Let's try a *modular* approach of building ▪RSMs



Working with♦RSM



Leader Election on ♦RSM

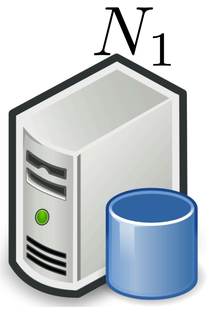


VoteMap	
N_1	$\perp$
N_2	$\perp$
N_3	$\perp$

$leader = \texttt{None}$

VoteMap	
N_1	$\perp$
N_2	$\perp$
N_3	$\perp$

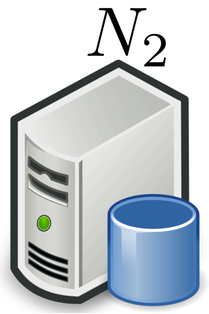
$leader = \texttt{None}$



VoteMap	
N_1	$\perp$
N_2	$\perp$
N_3	$\perp$

$leader = \texttt{None}$

Leader Election on ♦RSM

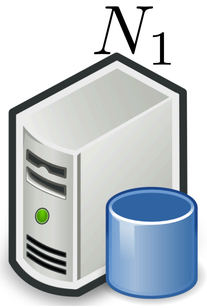


VoteMap	
N_1	$\perp$
N_2	N_1
N_3	$\perp$

$leader = \texttt{None}$

VoteMap	
N_1	$\perp$
N_2	$\perp$
N_3	N_3

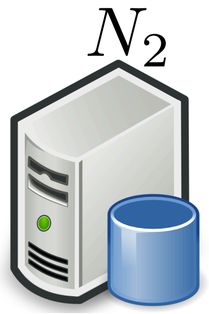
$leader = \texttt{None}$



VoteMap	
N_1	N_1
N_2	$\perp$
N_3	$\perp$

$leader = \texttt{None}$

Leader Election on ♦RSM

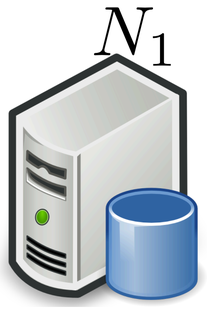


VoteMap	
N_1	N_1
N_2	N_1
N_3	N_3

$leader = \texttt{None}$

VoteMap	
N_1	N_1
N_2	N_1
N_3	N_3

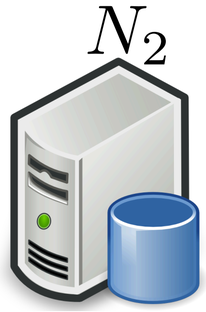
$leader = \texttt{None}$



VoteMap	
N_1	N_1
N_2	N_1
N_3	N_3

$leader = \texttt{None}$

Leader Election on ♦RSM



N_2

VoteMap	
N_1	N_1
N_2	N_1
N_3	N_3

$leader = \texttt{Some}(N_1)$

N_3

VoteMap	
N_1	N_1
N_2	N_1
N_3	N_3

$leader = \texttt{Some}(N_1)$

Safe State

VoteMap	
N_1	N_1
N_2	N_1
N_3	N_3

$leader = \texttt{Some}(N_1)$

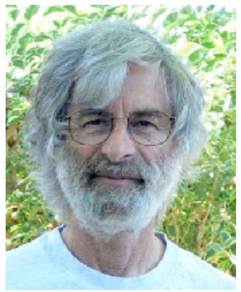
Generalized Leader Election on ♦RSM

VoteMap	
From	To

VoteMap		
Round	From	To

ProposalMap		
Round	Proposer	Proposal

Paxos!



Paxos Made Simple

Leslie Lamport

01 Nov 2001

Paxos

MS ○ Marc Shapiro -- at work <marc.sh... Wednesday, May 21, 2025 at 3:52 PM

To: ⊗ Gowtham Kaki

[External email - use caution]

Thanks for your talk today. It was inspiring. Thanks to you, I finally understood Paxos! Your explanation was so simple.

Marc

A Proof System for ♦RSM-based Consensus

THEOREM 4.3 (SUP VERIFICATION). *A replicated update system* RS *has stable update precondition* Φ^s *if the following verification conditions hold:*

Initiation Safety

$$\forall N \in \mathtt{Nid}_{\mathsf{RS}}.\quad \forall E \in \mathtt{Effect}_{\mathsf{RS}}.\quad \forall S \in \mathtt{AState}_{\mathsf{RS}}.\quad \forall V_1, V_2 \in \mathtt{Val}_{\mathsf{RS}}.\quad \forall f^a \in \mathtt{Action}.$$
$$f^a(N, V_1, S) = \langle V_2, E\rangle \implies \langle N, E, S\rangle \in \Phi^s$$

Racing-State Stability

$$\forall N_1, N_2 \in \mathtt{Nid}_{\mathsf{RS}}.\quad \forall E_1, E_2 \in \mathtt{Effect}_{\mathsf{RS}}.\quad \forall S, S' \in \mathtt{AState}_{\mathsf{RS}}.$$
$$\langle N_1, E_1, S\rangle \in \Phi^s \wedge \langle N_2, E_2, S\rangle \in \Phi^s \wedge f^h_{\mathsf{RS}}(E_2, S) = S' \implies \langle N_1, E_1, S'\rangle \in \Phi^s$$

Racing-State Commutativity

$$\forall N_1, N_2 \in \mathtt{Nid}_{\mathsf{RS}}.\quad \forall E_1, E_2 \in \mathtt{Effect}_{\mathsf{RS}}.\quad \forall S, S_1', S_1'', S_2', S_2'' \in \mathtt{AState}_{\mathsf{RS}}.$$
$$(\langle N_1, E_1, S\rangle \in \Phi^s \wedge \langle N_2, E_2, S\rangle \in \Phi^s \wedge f^h_{\mathsf{RS}}(E_1, S) = S_1'$$
$$\wedge f^h_{\mathsf{RS}}(E_2, S_1') = S_1'' \wedge f^h_{\mathsf{RS}}(E_2, S) = S_2' \wedge f^h_{\mathsf{RS}}(E_1, S_1') = S_2'') \implies S_1'' = S_2''$$

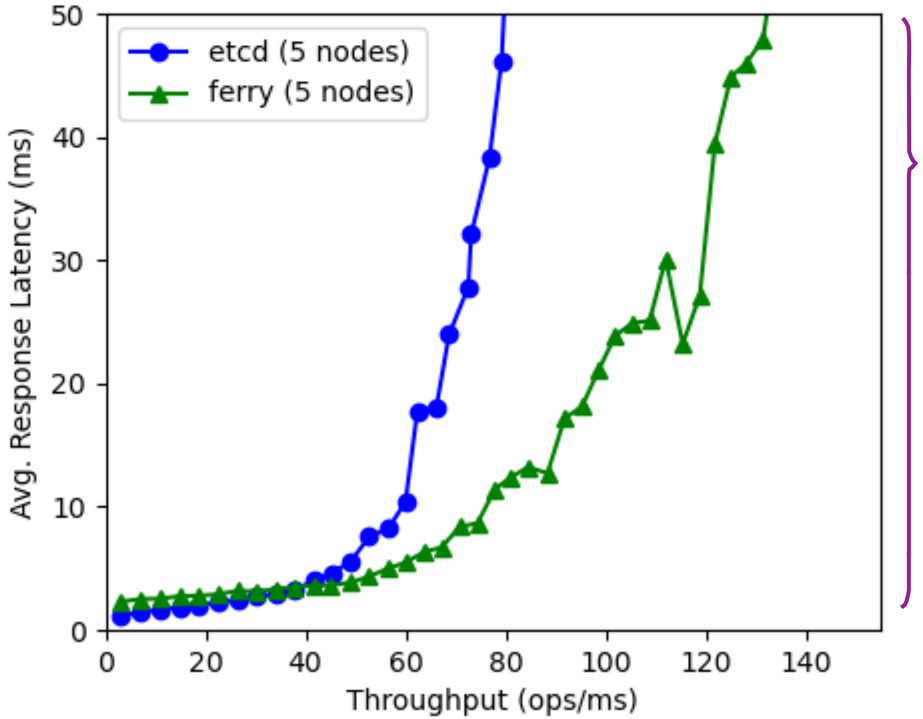
Uses Strong Update Preconditions instead of Inductive Invariants

$\mathbb{P}$ vs $\mathbb{I}$

THEOREM 4.8 (DECIDABILITY FRIENDLY). *The Def. 4.6 conditions are decidable as long as the relations* R^a, R^h, R^d, *and* Φ^s, *taken together, are in the Extended EPR logic fragment.*

Super-V

- A Haskell library to implement and formally verify distributed protocols.
- Case study: Raft log replication protocol.
  - Implementation: ~400 LOC.
  - Specification (**P**): ~110 LOC. No quantifier alternation. Extended EPR.
- Automatically verified using CVC5 in under 50mins on Intel Core I5 laptop.



- **Performs similarly or better than `etcd-raft`!**

Bolt-On Strong Consistency: Specification, Implementation, and Verification

NICHOLAS V. LEWCHENKO, University of Colorado Boulder, USA
GOWTHAM KAKI, University of Colorado Boulder, USA
BOR-YUH EVAN CHANG*, University of Colorado Boulder, USA and Amazon, USA

[OOPSLA 2025]

2. Improving Generality: From *safety* to *security*

Contributors


Kirby Linvill


Eric Wustrow


Evan Chang

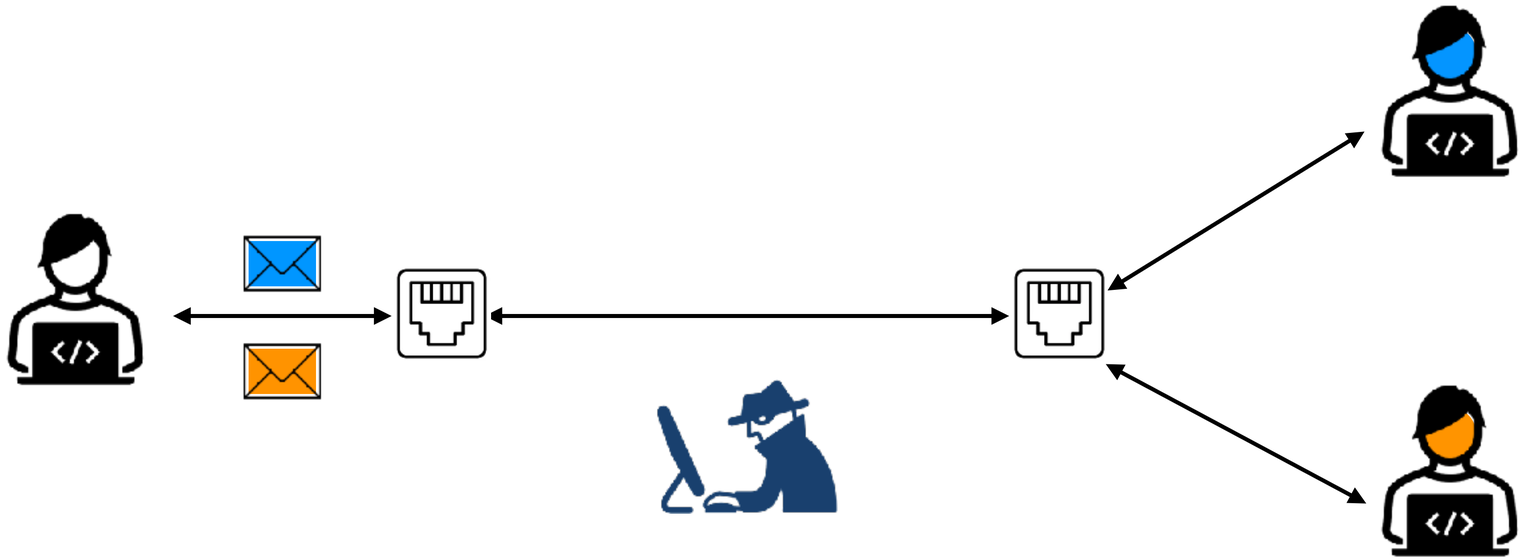
Tools

https://github.com/cuplv/Waldo

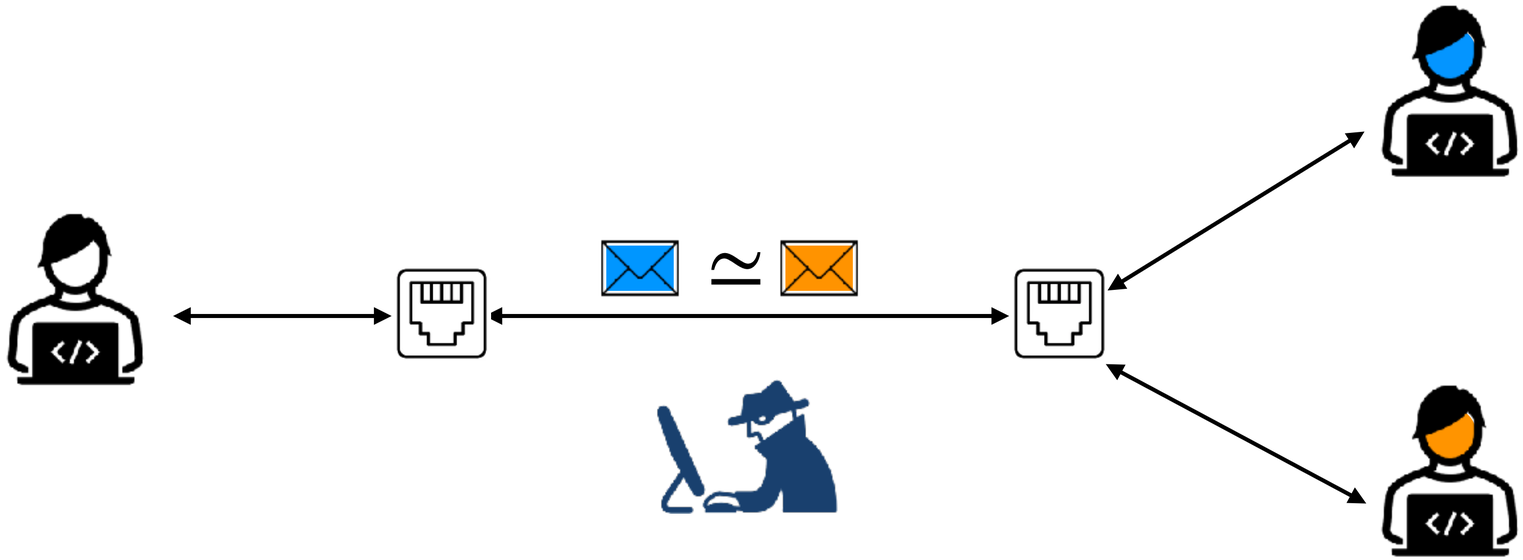
Publications

- Kirby Linvill, Gowtham Kaki, and Eric Wustrow. **Verifying Indistinguishability of Privacy-Preserving Protocols.** Proceedings of the ACM on Programming Languages (PACMPL), issue **OOPSLA2 2023**, Article No.: 273, Pages 1442 - 1469.
- Kirby Linvill, Gowtham Kaki, and Bor-Yuh Evan Chang. **A Bisimulation Proof Framework for Symbolic Active Indistinguishability.** Pre-print.

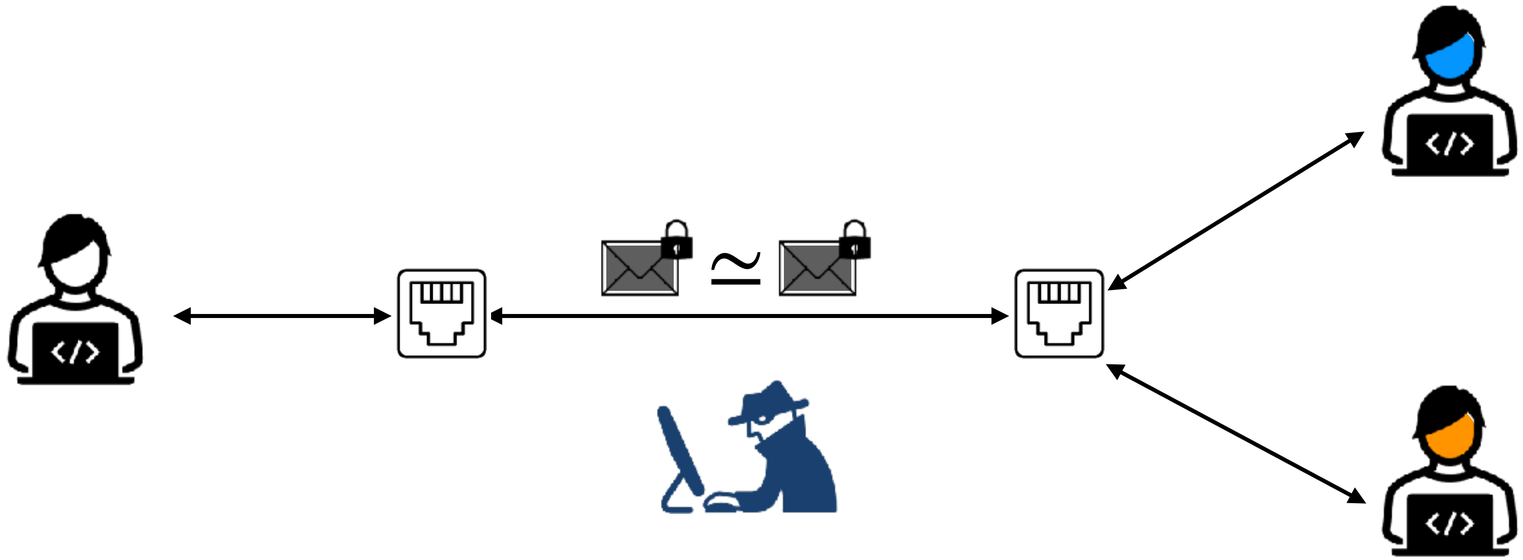
Privacy on Public Networks



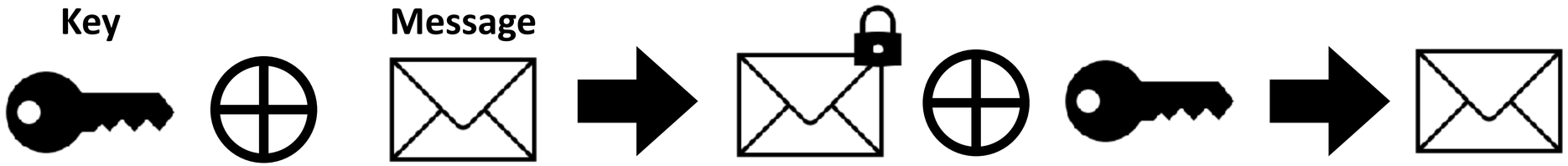
Privacy on Public Networks



Privacy on Public Networks



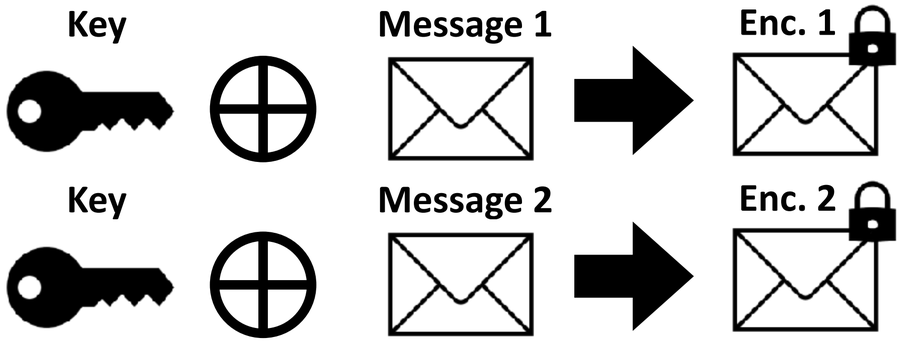
One-Time Pad (OTP) Encryption

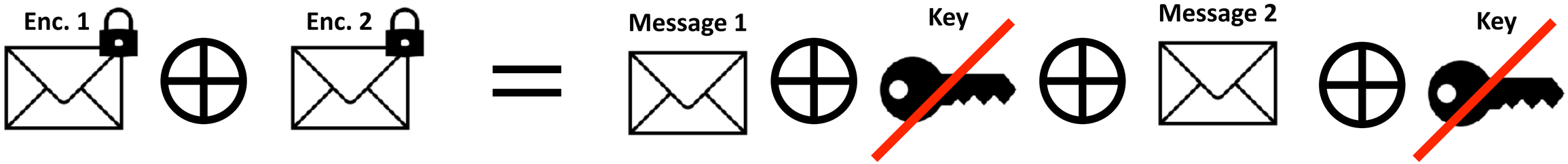


1. Generate a key such that length(key) = length(message)
2. Exclusive Or (XOR) the key and message to encrypt
3. XOR the key and message to decrypt

Improper One-Time Pad (OTP) Encryption

What if we re-use the key?

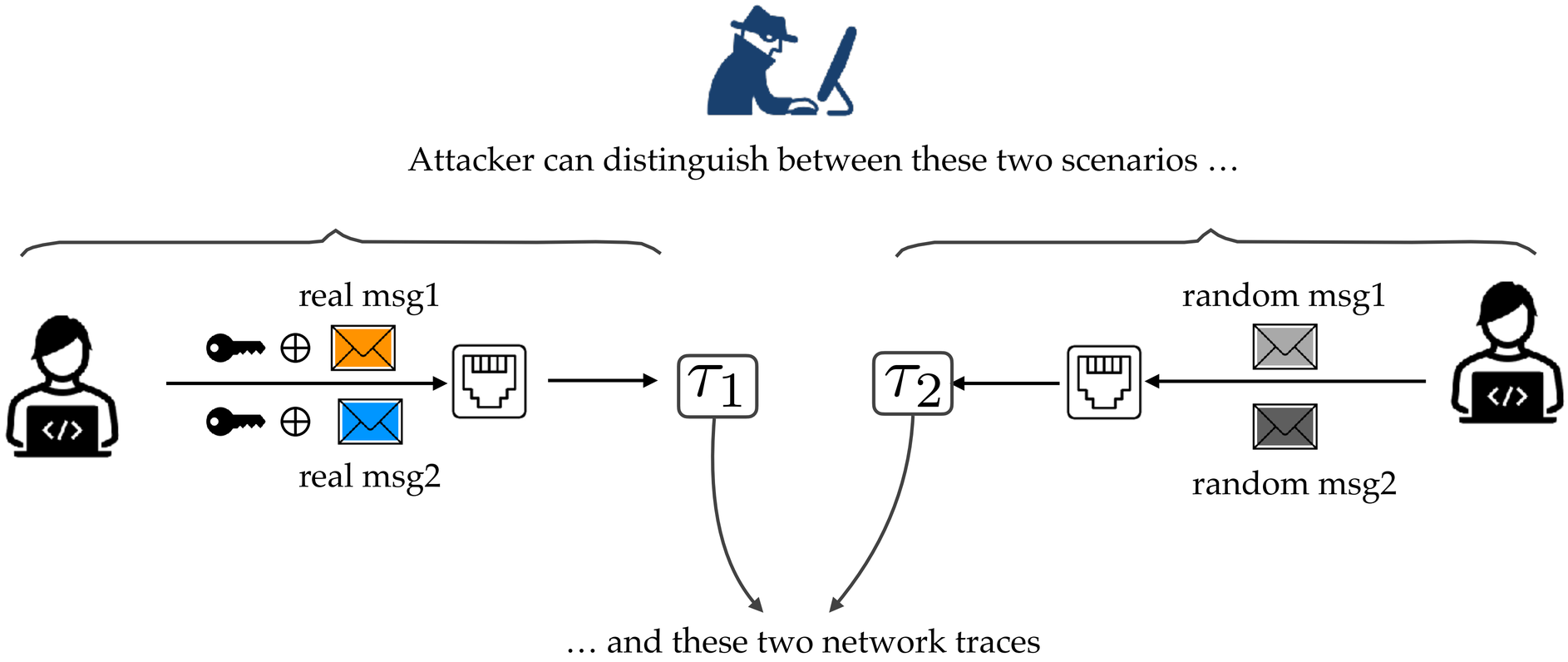




Since $x \oplus x = 0$, $x \oplus 0 = x$, and $\oplus$ is commutative

This leaks which bits of Message 1 and Message 2 differ

Improper One-Time Pad (OTP) Encryption



Symbolic Model of Cryptography

An axiomatization of cryptography to enable symbolic reasoning

- Plaintexts and ciphertexts are uninterpreted symbols.
- Encryption and decyption are uninterpreted functions related by:

$$\forall m.\forall k_1, k_2.\ \texttt{decrypt}(\texttt{encrypt}(m, k_1), k_2) = m \ \Leftrightarrow\ k_1 = k_2$$

- Decryption is the only operation defined on ciphertexts.
- The only way to decrypt a ciphertext is by knowing the key, in which case the corresponding plaintext can be *deduced*.
- A pair of symbols, S_1 and S_2, are *distinguishable* iff there exists a deduction possible on one but not the other. Otherwise they are *indistinguishable*. Likewise with traces of symbols.

In the symbolic model, OTP is a perfect encryption primitive.

$\Longrightarrow$

We can "prove" in the symbolic model that τ_1 and τ_2 are indistinguishable

Probabilistic Model of Cryptography

Verifying Indistinguishability of Privacy-Preserving Protocols

KIRBY LINVILL, University of Colorado Boulder, USA
GOWTHAM KAKI, University of Colorado Boulder, USA
ERIC WUSTROW, University of Colorado Boulder, USA

[OOPSLA 2023]

- Introduces probabilistic model of cryptography.
- Crypto primitives are modeled as functions that sample bitstrings from pre-defined distributions.
- E.g., a perfect block cipher with block size L can be modeled as:

$$c \xleftarrow{\$} \{0,1\}^L$$

- We can now reason about the (symbolic) probability of generating a trace.
- Two encrypted network traces, τ_1 and τ_2, are indistinguishable iff they are equally likely to have been generated by a protocol execution.
- Alternatively, two protocol executions, π_1 and π_2, are indistinguishable iff they are equally likely to generate any given encrypted trace τ.

$$\underbrace{\mathtt{Pr}[\pi_1 \Downarrow \tau] \;=\; \mathtt{Pr}[\pi_2 \Downarrow \tau]}$$

We adapted Probabilistic Relational Hoare Logic (PRHL; Barthe et al, POPL 2009) to prove verification conditions of this form …

Probabilistic Model of Cryptography

Verifying Indistinguishability of Privacy-Preserving Protocols

KIRBY LINVILL, University of Colorado Boulder, USA
GOWTHAM KAKI, University of Colorado Boulder, USA
ERIC WUSTROW, University of Colorado Boulder, USA

[OOPSLA 2023]

Waldo

https://github.com/cuplv/Waldo

$$\underbrace{\mathtt{Pr}[\pi_1 \Downarrow \tau] \;=\; \mathtt{Pr}[\pi_2 \Downarrow \tau]}$$

We adapted Probabilistic Relational Hoare Logic (PRHL; Barthe *et al*, POPL 2009) to prove verification conditions of this form …

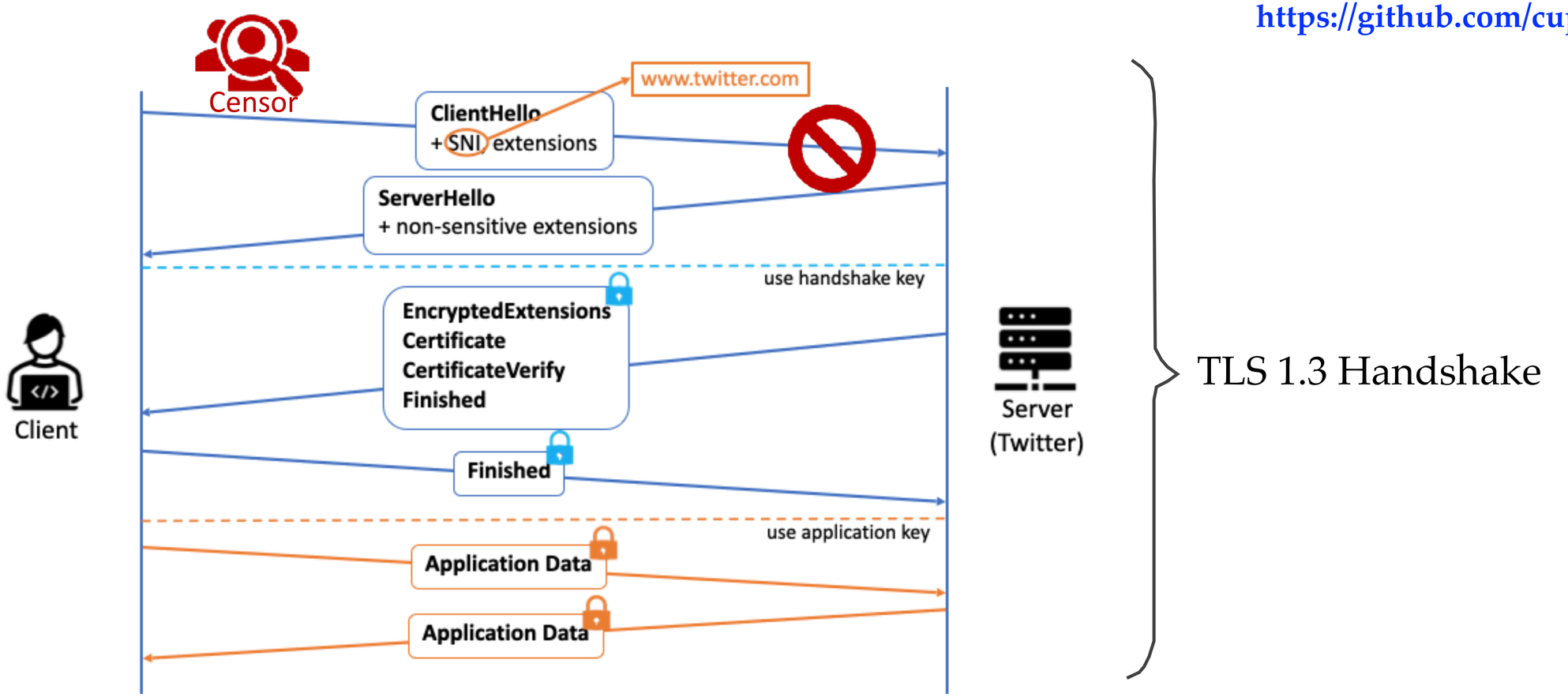
… and built an SMT-aided proof automation framework in F*.

Waldo

- A framework to verify privacy guarantees of communication protocols
- Practical impact: verification of TLS ECH Extension

Waldo

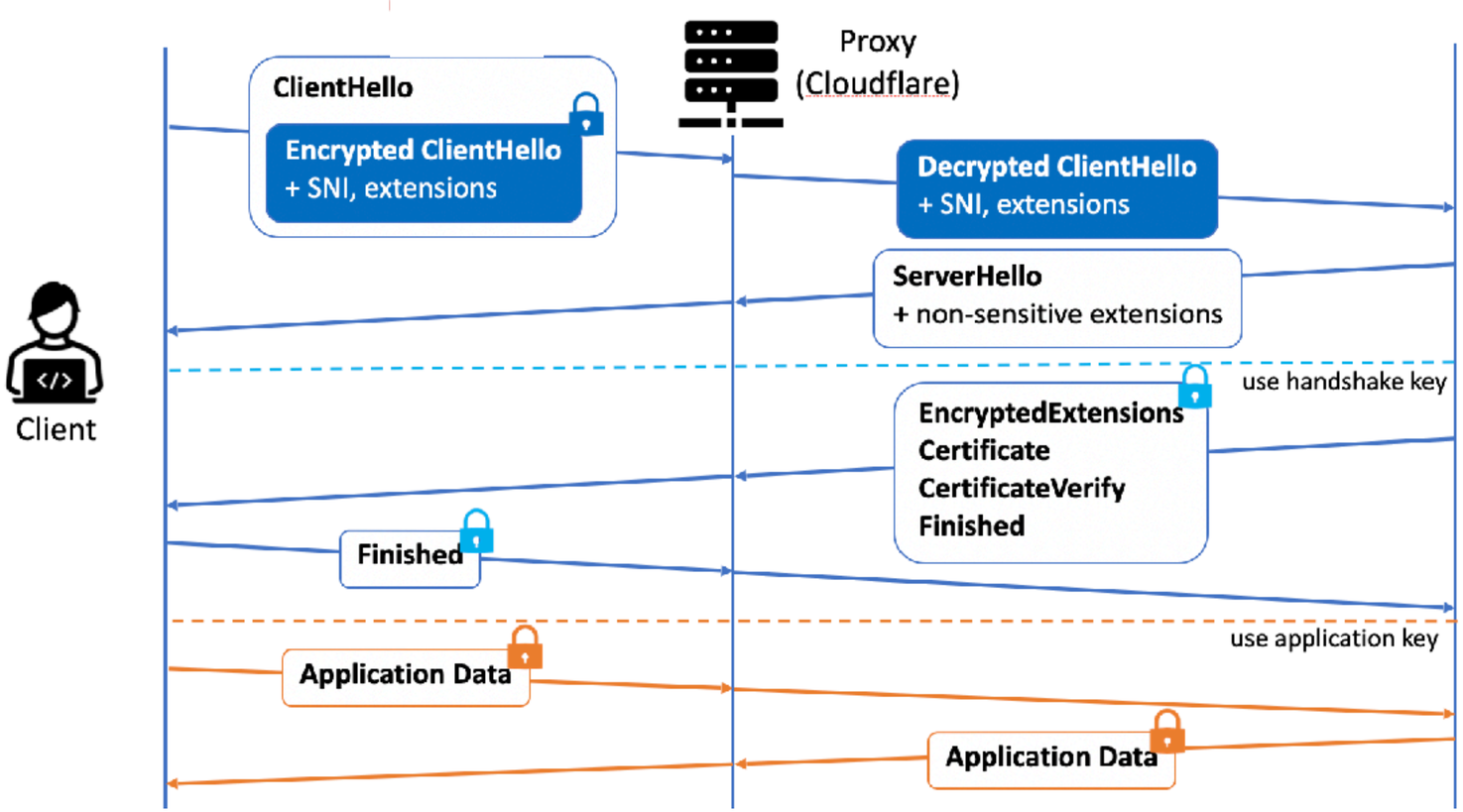
https://github.com/cuplv/Waldo



Waldo

- A framework to verify privacy guarantees of communication protocols
- Practical impact: verification of TLS ECH Extension

https://github.com/cuplv/Waldo



Encrypted Client Hello (ECH) extension to TLS

3. Improving Usability: From *experts* to *programmers*

Contributors


Nick Lewchenko


Christian Fontenot


Shawn Meier


Evan Chang

Tools

Raven

Historia

DisProve

https://github.com/cuplv/ravencheck
https://github.com/cuplv/Historia
https://github.com/cuplv/disprove

Publications

- Nicholas Lewchenko, Gowtham Kaki, and Bor-Yuh Evan Chang. **Effectively-Propositional Higher-Order Functional Programming.** Under editorial review for ACM on Programming Languages (PACMPL), issue **OOPSLA1 2026**.
- Shawn Meier, Sergio Mover, Gowtham Kaki, and Bor-Yuh Evan Chang. **Historia: Refuting Callback Reachability with Message-History Logics.** Proceedings of the ACM on Programming Languages, Volume 7, Issue **OOPSLA2 2023**. Article No.: 289, Pages 1905 - 193.
- Christian Fontenot, Gowtham Kaki, and Bor-Yuh Evan Chang. **Dissprove: Goal-Directed Verification of Parametrized Affine Actor Systems**. Under submission to POPL 2026.

Artifacts of Formal Verification

- Specifications
- Invariants
- Proofs

Artifacts of Formal Verification

- Specifications
- Invariants
- Proofs

More often than not:

- Intuitively known to programmers
- Can be recovered from the `asserts`

Encoding in a decidable logic guarantees proof automation.

- Notoriously hard to discern.
- Quantified invariant inference is an open problem.

Is it possible to circumvent the need for invariants?

- An (inductive) invariant characterizes every *reachable* state of the system.
- What if we reason about *unreachable* states instead?

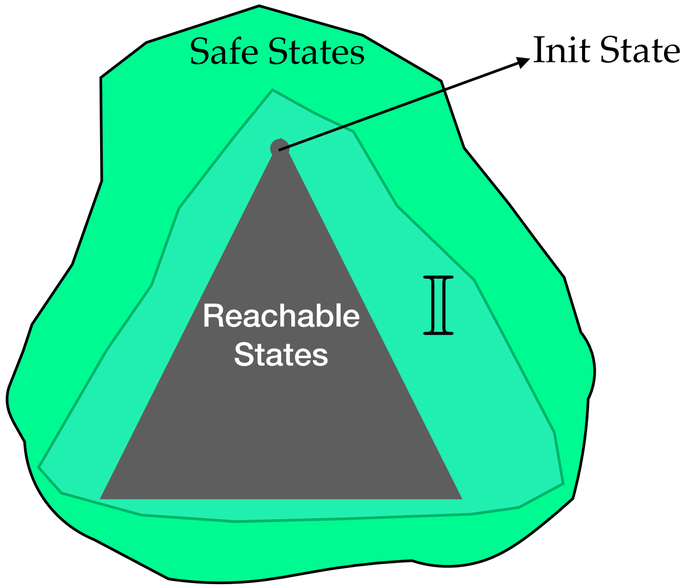
```
event eInitialize :
  (Id: int, nodes: map[int, Node]);
event eVote : int;
event eLeader : int;

machine Node {
  var my_id: int;
  var my_votes: int;
  var my_leader: int;

  on eInitialize do (int) {
    ...
    my_leader = -1;
  }
  ...
  on eLeader do (id: int) {
    assert(my_leader == -1 ||
           my_leader == id);
    my_leader = id;
  }
}
```

An implementation of Leader Election protocol in P language (Desai et al, PLDI'13)

Goal-Directed Verification

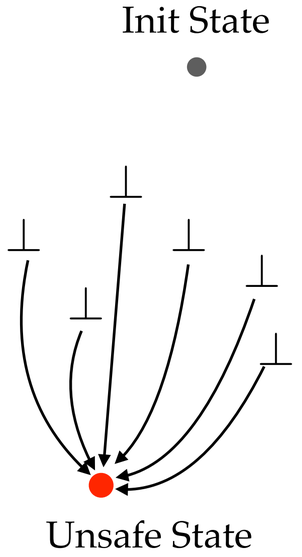
- An (inductive) invariant characterizes every *reachable* state of the system.
- What if we reason about *unreachable* states instead?

Regular (Forwards) Verification



All states reachable from the initial state are safe

Goal-Directed Verification



Unsafe state is unreachable from the init state

Goal-Directed Verification in Practice

Historia: Refuting Callback Reachability with Message-History Logics

SHAWN MEIER, University of Colorado Boulder, USA
SERGIO MOVER, LIX, École Polytechnique, CNRS, Institut Polytechnique de Paris, France
GOWTHAM KAKI, University of Colorado Boulder, USA
BOR-YUH EVAN CHANG', University of Colorado Boulder & Amazon, USA

Historia

[OOPSLA 2023]

https://github.com/cuplv/Historia

- A goal-directed formal verification engine for Android apps.
- Proved the absence of a specific class of callback ordering-driven bugs in 47 open-source apps with 2M lines of code *without* requiring explicit inductive invariants.

Dissprove: Goal-Directed Verification of Parametrized Affine Actor Systems

ANONYMOUS AUTHOR(S)

[Under submission to POPL 2026]

DisProve

https://github.com/cuplv/disprove

- Adapts goal-directed formal verification to distributed protocols.
- Proved the safety of leader election protocol with *unbounded* number of participants *without* needing inductive invariants or update preconditions!
- Guaranteed to terminate on distributed algorithms composed of affine actors.

Key bottleneck: computational complexity!

Focus of future work.

Conclusion

- Programming distributed systems is hard due to the sheer combinatorial complexity of many things that can go wrong.
- Formal verification can help ensure fail-safe operation and resilience to adversarial attacks.
- There exist several impediments to widespread adoption of formal methods in distributed systems development. Three main ones are poor scalability, insufficient generality, and steep learning curve.
- My research addresses these impediments by developing novel programming abstractions and automated reasoning techniques. The goal is to make provably-safe distributed systems development accessible to mainstream application developers.
- Research results have been published in open-access journals and implemented in several open-source tools.

Acknowledgements

NSF
Computer Science
COLLEGE OF ENGINEERING AND APPLIED SCIENCE
Research & Innovation Office
Engineering Education and AI-Augmented Learning IRT
COLLEGE OF ENGINEERING AND APPLIED SCIENCE
Google