MAKING THE IMPOSSIBLE POSSIBLE (AGAIN)

Ten Years of Flexible Paxos

Heidi Howard
Researcher @ Microsoft & University of Cambridge

Flexible Paxos: Quorum Intersection Revisited

Heidi Howard[1], Dahlia Malkhi[2], and Alexander Spiegelman[3]

1 **VMware Research, Palo Alto, CA, USA; and**
University of Cambridge Computing Laboratory, Cambridge, UK
heidi.howard@cl.cam.ac.uk

2 **VMware Research, Palo Alto, CA, USA**
dahliamalkhi@gmail.com

3 **VMware Research, Palo Alto, CA, USA; and**
Viterbi Dept. of Electrical Engineering, Technion Haifa, Haifa, Israel
sashas@tx.technion.ac.il

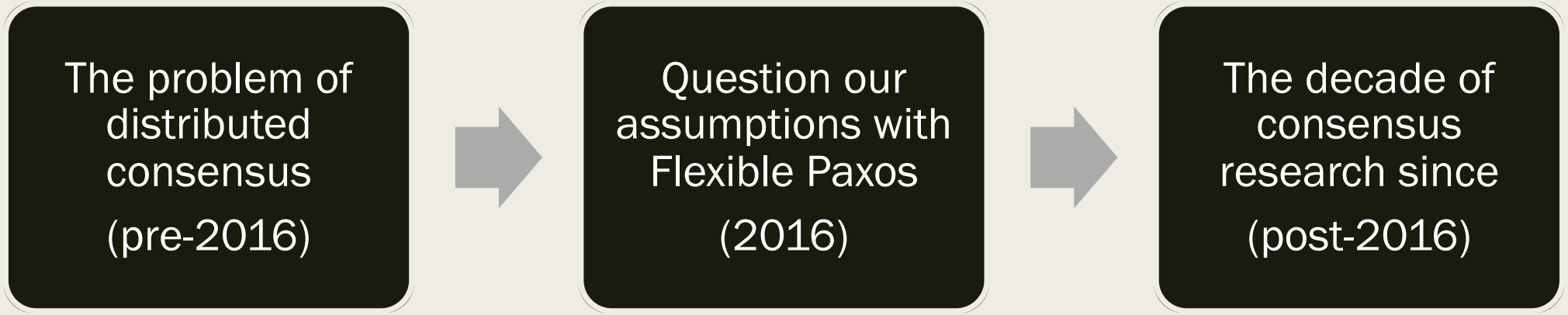
Abstract

Distributed consensus is integral to modern distributed systems. The widely adopted Paxos algorithm uses two phases, each requiring majority agreement, to reliably reach consensus. In this paper, we demonstrate that Paxos, which lies at the foundation of many production systems, is conservative. Specifically, we observe that each of the phases of Paxos may use non-intersecting quorums. Majority quorums are not necessary as intersection is required only across phases.

Using this weakening of the requirements made in the original formulation, we propose Flexible Paxos, which generalizes over the Paxos algorithm to provide flexible quorums. We show that Flexible Paxos is safe, efficient and easy to utilize in existing distributed systems. We discuss far reaching implications of this result. For example, improved availability results from reducing the size of second phase quorums by one when the system size is even, while keeping majority quorums in the first phase. Another example is improved throughput of replication by using much smaller phase 2 quorums, while increasing the leader election (phase 1) quorums. Finally, non intersecting quorums in either first or second phases may enhance the efficiency of both.

[OPODIS 2016]

Today's talk


The problem of distributed consensus
(pre-2016)
Question our assumptions with Flexible Paxos
(2016)
The decade of consensus research since
(post-2016)

THE PROBLEM OF DISTRIBUTED CONSENSUS

There is a particular kind of suffering reserved for engineers who must make multiple computers agree on something.

It's not the clean suffering of a hard mathematical proof or the dramatic suffering of a production outage at 3 AM (though consensus algorithms cause plenty of both).

It's the slow, grinding suffering of reading a paper that says "the protocol is straightforward" and then spending six months discovering all the ways it isn't.

The Agony of Consensus Algorithms

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

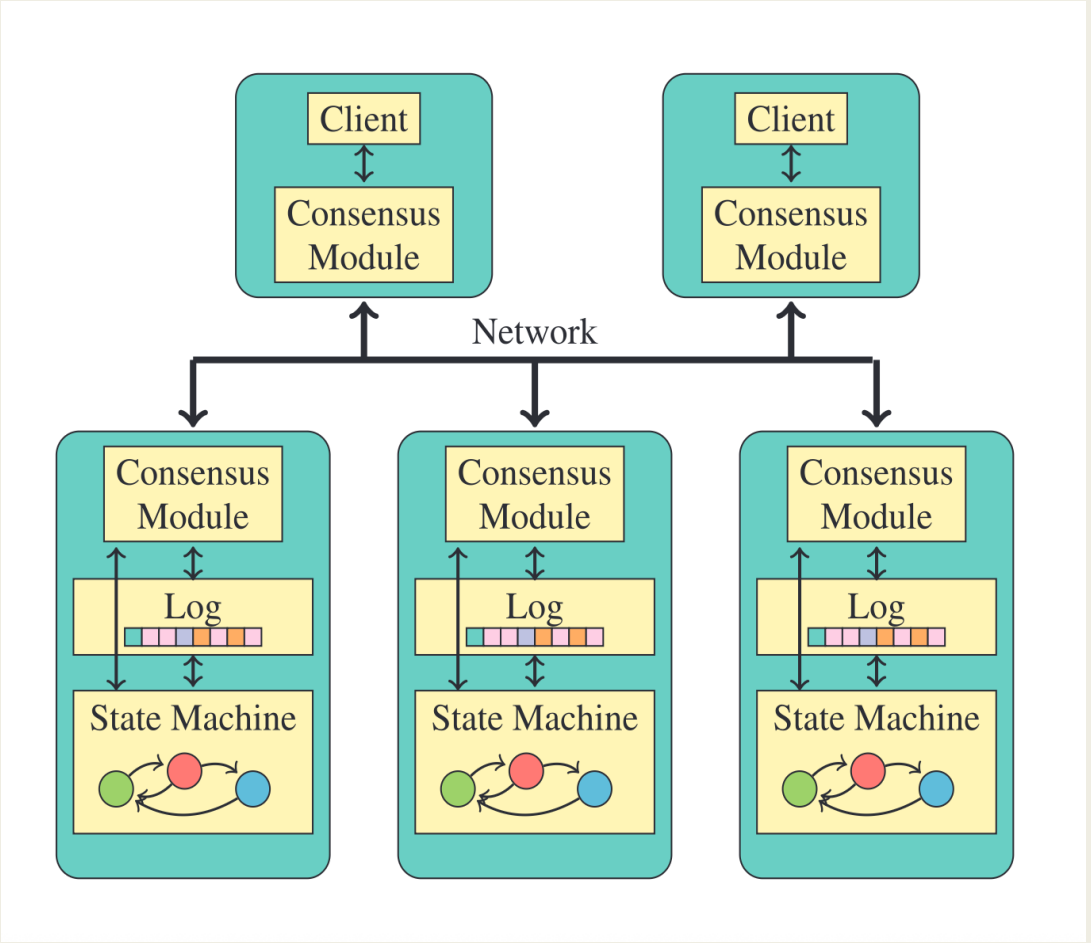
[JACM 1985]

Defining distributed consensus

- Definitions vary depending on who you ask (and when)
- Reaching agreement over a sequence of values in an asynchronous distributed system in the face of crash failures.
  - *Compromising safety is never an option*
  - *Servers fail by crashing*
  - *Network messages can be lost or delayed*
  - *Clock synchronization is not assumed*
  - *Servers will follow the protocol (they cannot act arbitrarily)*

Why bother with this *particular kind of suffering* ?

- One reason is to implement state machine replication (SMR)
- Take a deterministic state machine (i.e. key-value store) and replicate it across a set of servers
- Distributed consensus is then used to agree an append-only log of operations for the state machines
- Distributed consensus ensures each state machine applies the same operations (in the same order!)



The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators. The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems.

Categories and Subject Descriptors: C2.4 [**Computer-Communications Networks**]: Distributed Systems—*Network operating systems*; D4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; J.1 [**Administrative Data Processing**]: Government

General Terms: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

This submission was recently discovered behind a filing cabinet in the *TOCS* editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.

The author appears to be an archeologist with only a passing interest in computer science. This is unfortunate; even though the obscure ancient Paxon civilization he describes is of little interest to most computer scientists, its legislative system is an excellent model for how to implement a distributed computer system in an asynchronous environment. Indeed, some of the refinements the Paxons made to their protocol appear to be unknown in the systems literature.

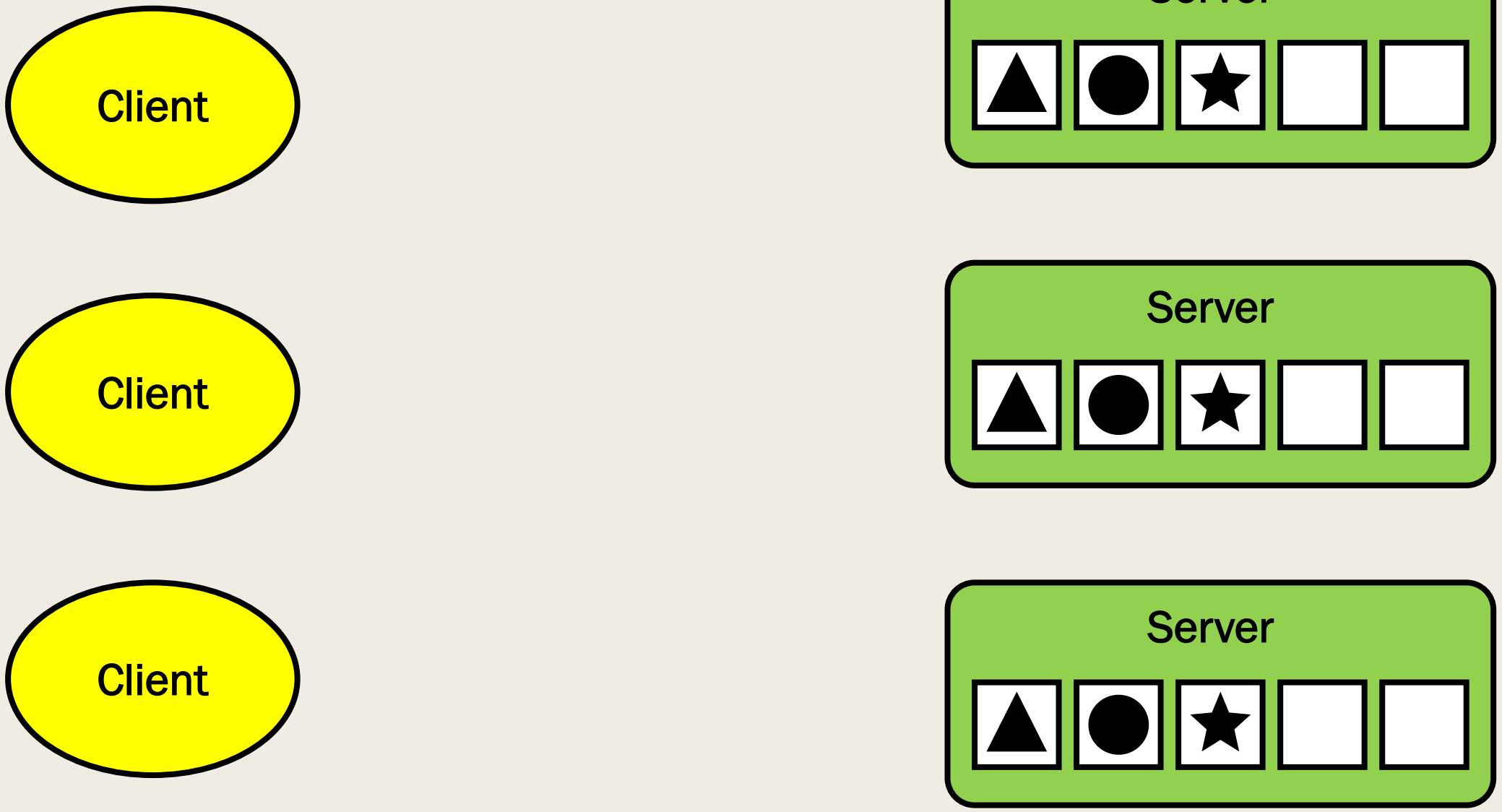
The author does give a brief discussion of the Paxon Parliament's relevance to distributed computing in Section 4. Computer scientists will probably want to read that section first. Even before that, they might want to read the explanation of the algorithm for computer scientists by Lampson [1996]. The algorithm is also described more formally by De Prisco et al. [1997]. I have added further comments on the relation between the ancient protocols and more recent work at the end of Section 4.

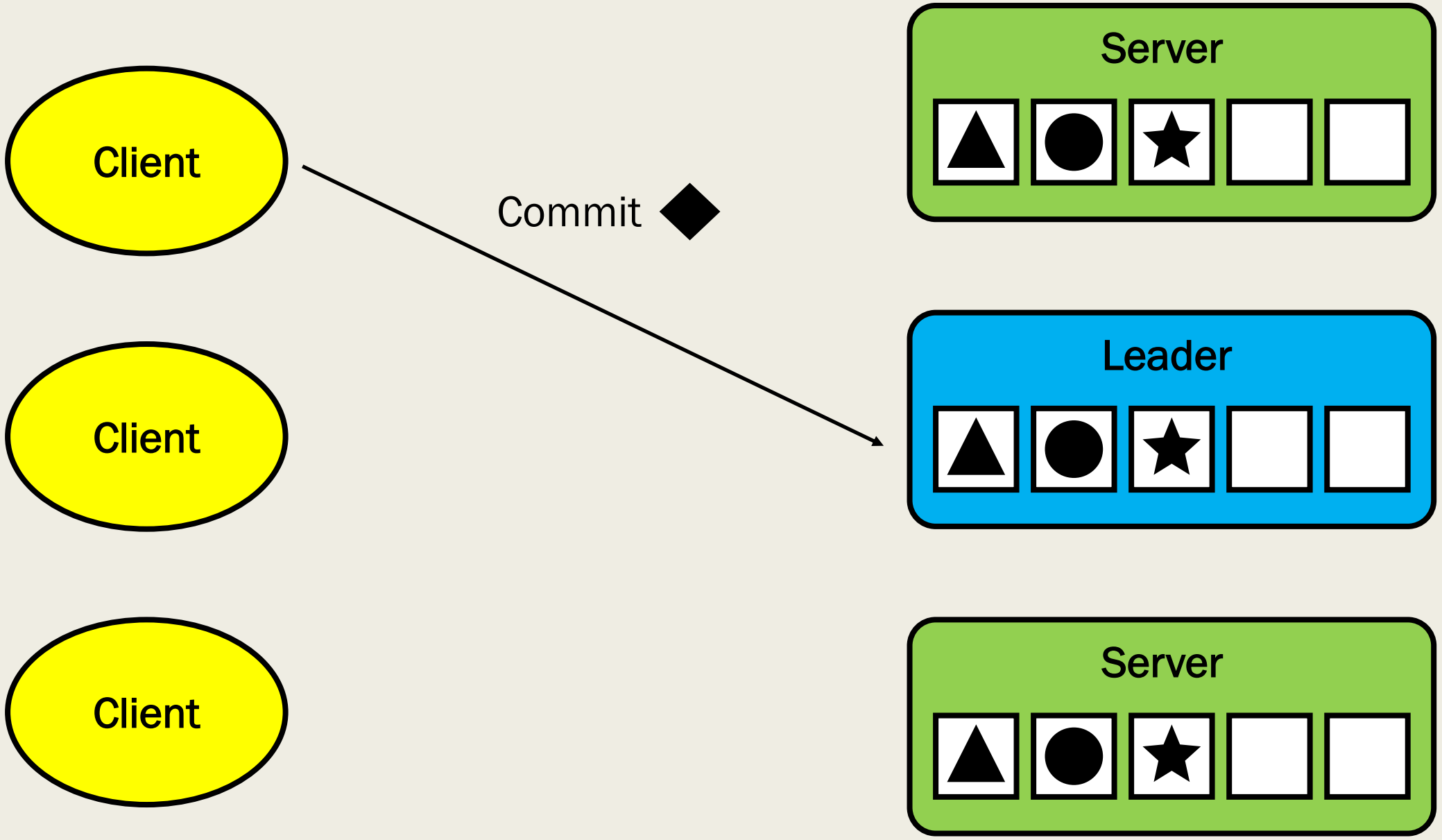
Keith Marzullo
University of California, San Diego

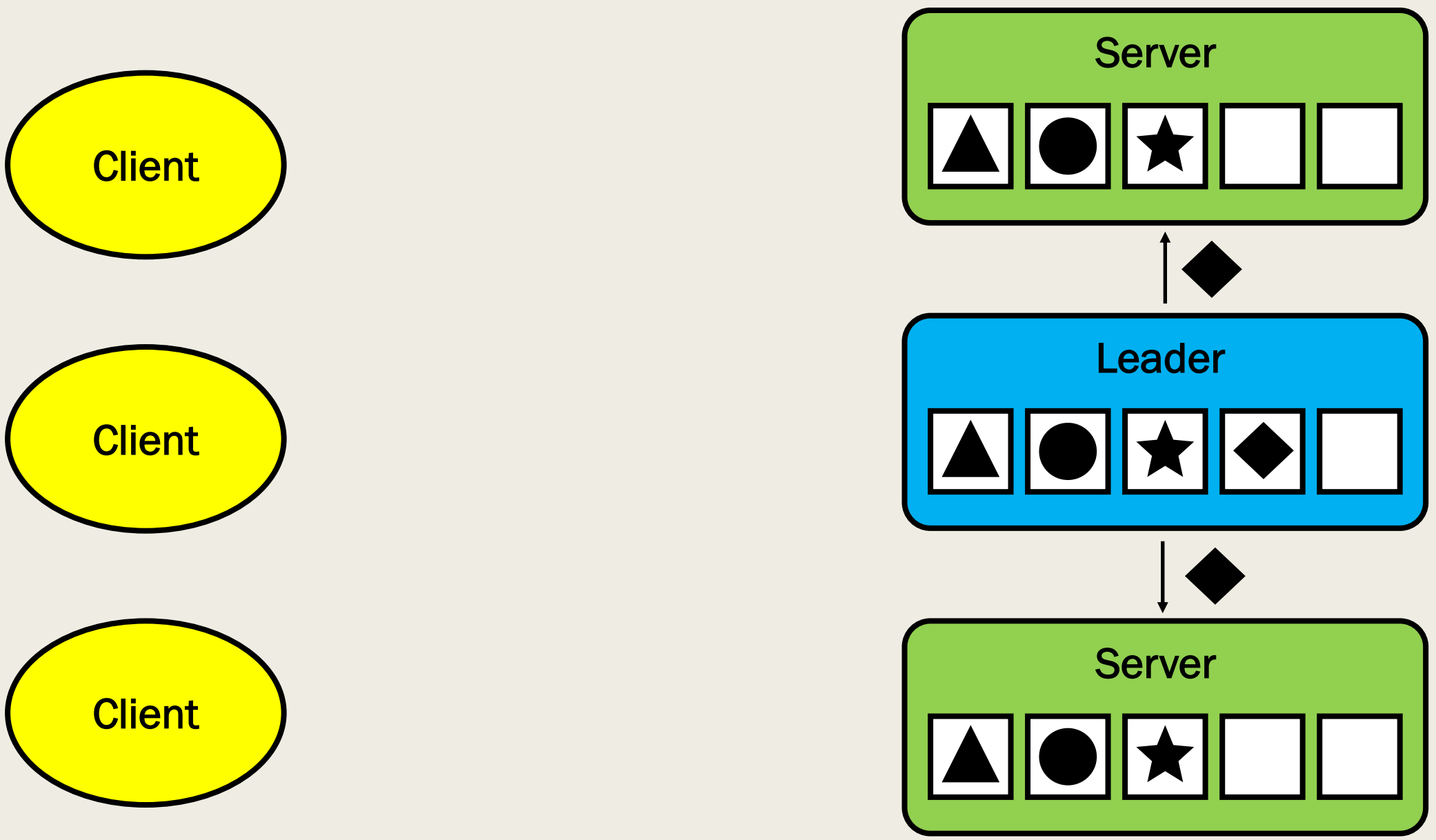
[TOCS 1998]

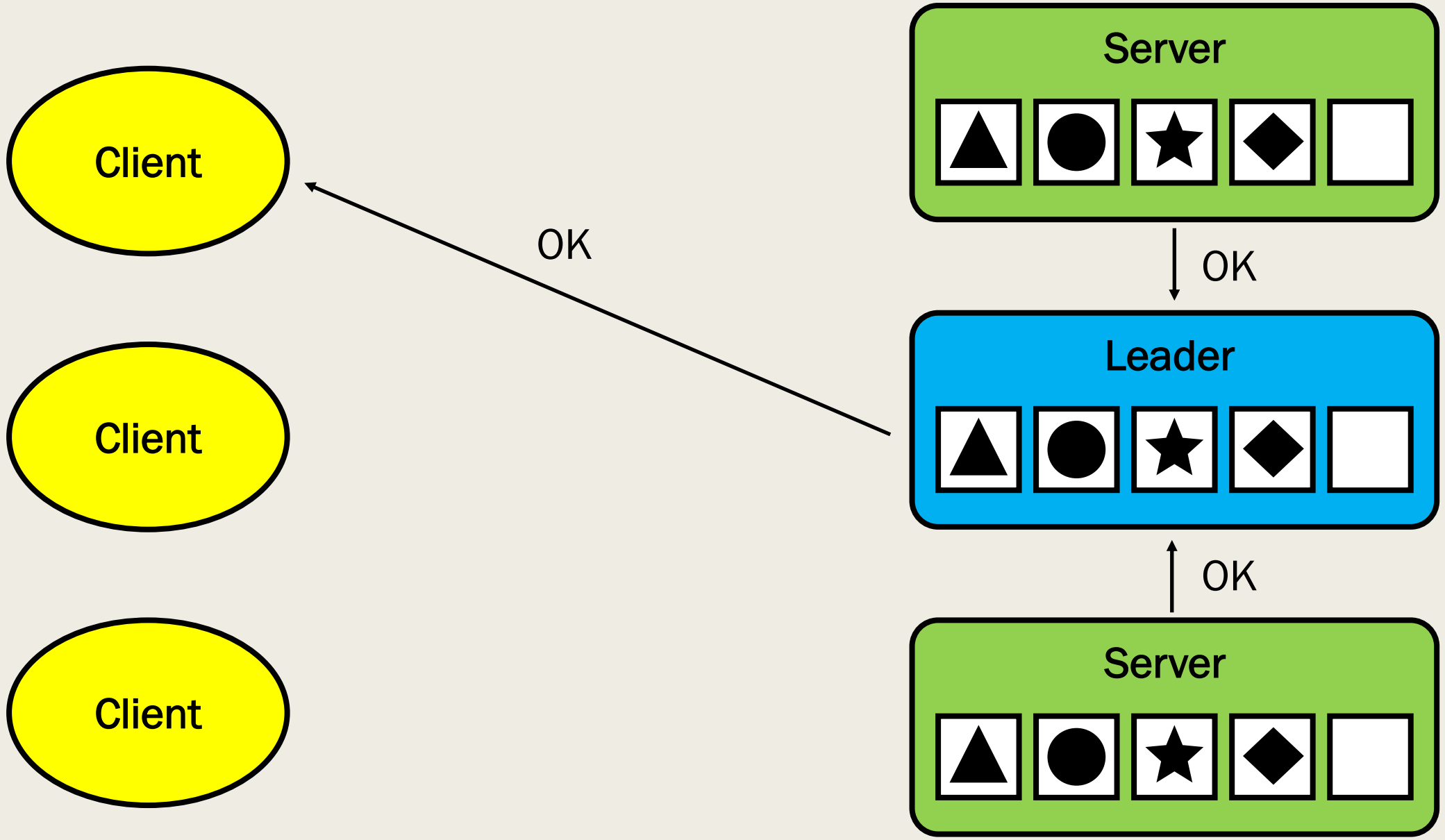
Paxos - Replication phase

- One of the servers is elected to be *leader*
- Clients submit operations to the leader, the leader appends them to their own log and tell the other servers to do the same
- Once the leader learns that enough servers have appended an operation to their logs, the leader considers it committed. The leader applies it to their state machine and replies to the client


Client
Client
Client
Server
Server
Server


Client
Client
Client
Commit
Server
Leader
Server


Client
Client
Client
Server
Leader
Server


Server
Client
OK
OK
Leader
Client
OK
Server
Client

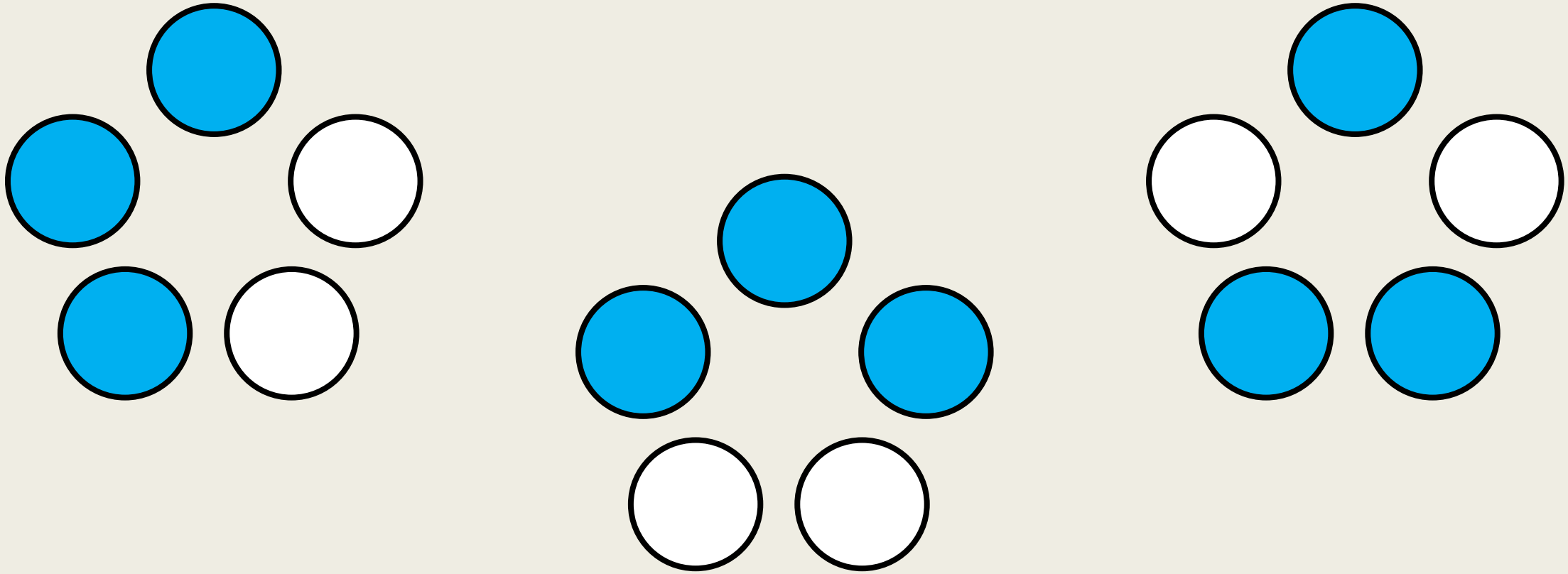
Paxos – Leader election phase

- If the leader fails, then another server is elected to become the leader by getting enough votes from other servers
- Each leader runs for a *term*. When one server votes for another it promises never to vote for a leader with a smaller term
- Newly elected leaders update their logs with the operations from the greatest terms to ensure they are up to date

How many servers are **enough**?

- There needs to be **enough** servers with copies of a command before it is considered committed and new leaders need **enough** votes
- We refer to any set of servers that is enough as a quorum and require that quorums intersect. Majority quorums can be used to achieve this
- Why? So that a leader will always learn about previously committed operations

Majority quorums



QUESTIONING OUR ASSUMPTIONS WITH FLEXIBLE PAXOS

In 2016, Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman published a paper that made the distributed systems community collectively facepalm. The paper, "Flexible Paxos: Quorum Intersection Revisited," presented an insight so simple, so obvious in retrospect, that you have to wonder how it escaped notice for nearly three decades...

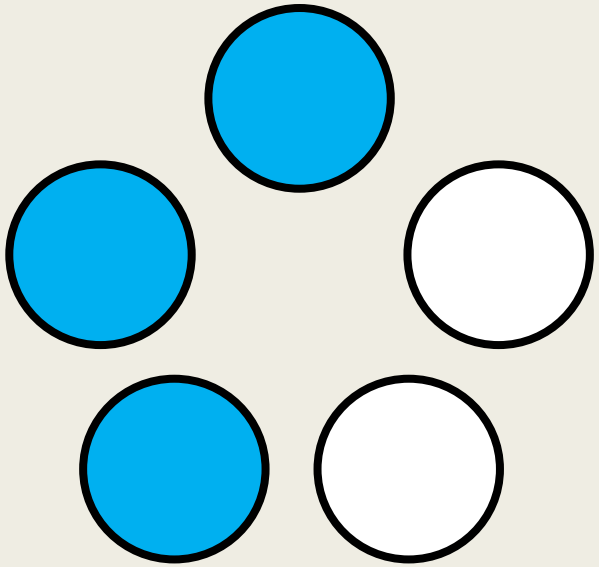
How this remained unformalized for decades is a question that says something uncomfortable about how we read our own foundational papers.

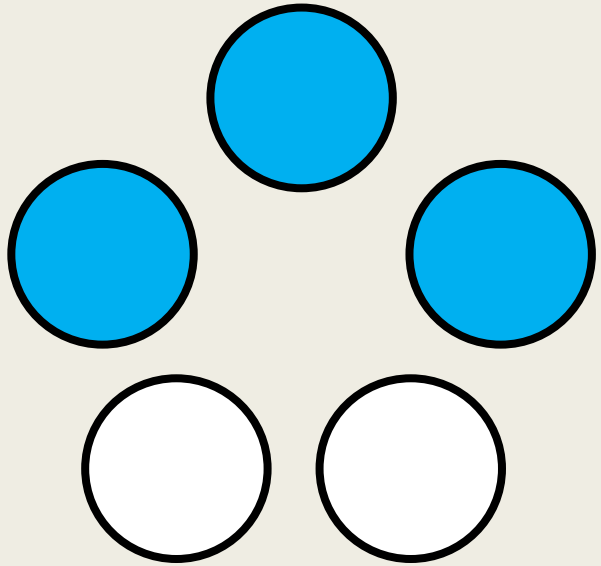
The Agony of Consensus Algorithms

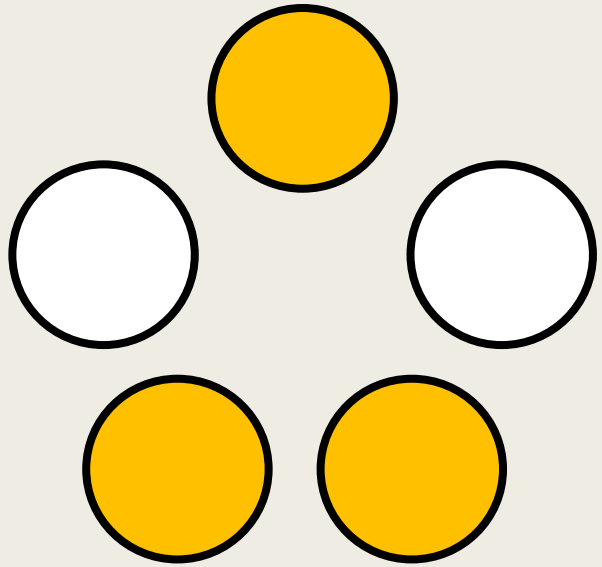
The insight that was hiding in plain sight

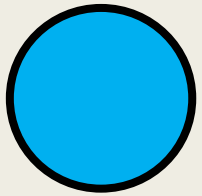
- Let’s distinguish between the quorums we use for replication and leader election
- Traditionally, we have assumed that all quorums needed to intersect, regardless of whether the quorum was used for replication or leader election.
- In reality, the only requirement is that leader election quorums intersect with replication quorums.
- We do not require intersection between quorums used for replication (or even leader election*)

Strict Majorities

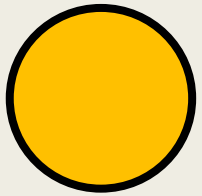






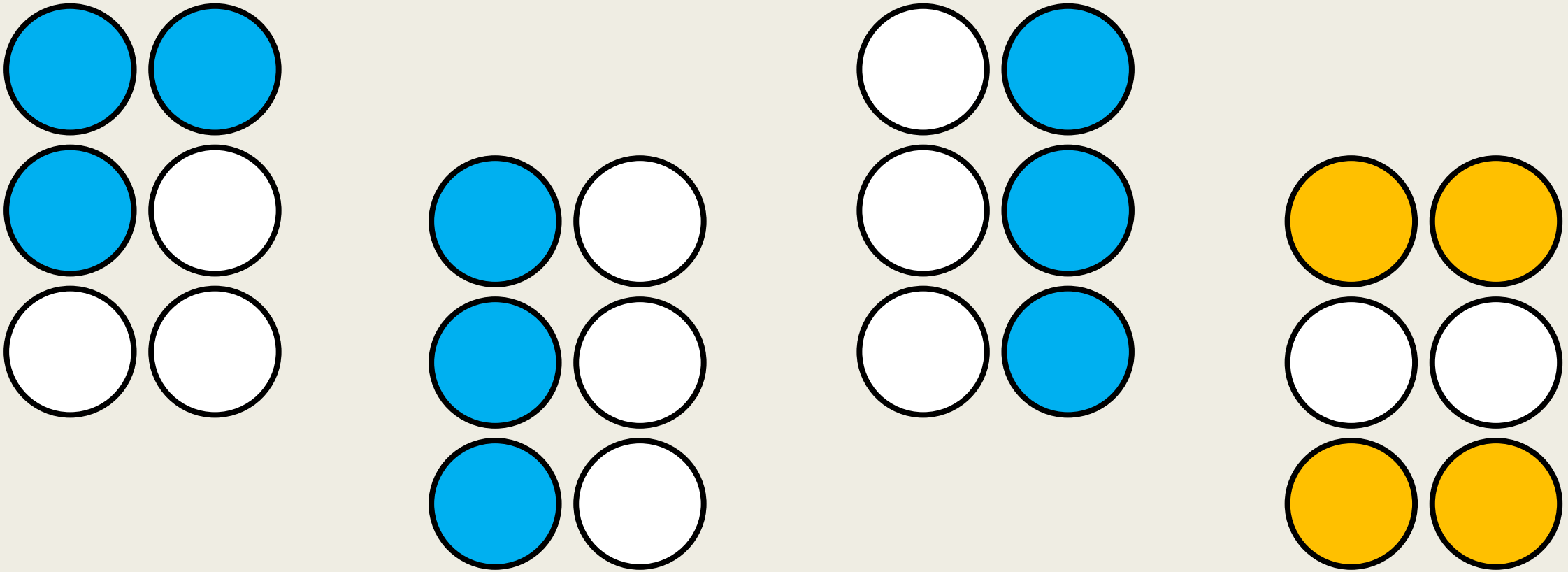


Replication quorum



Leader election quorum

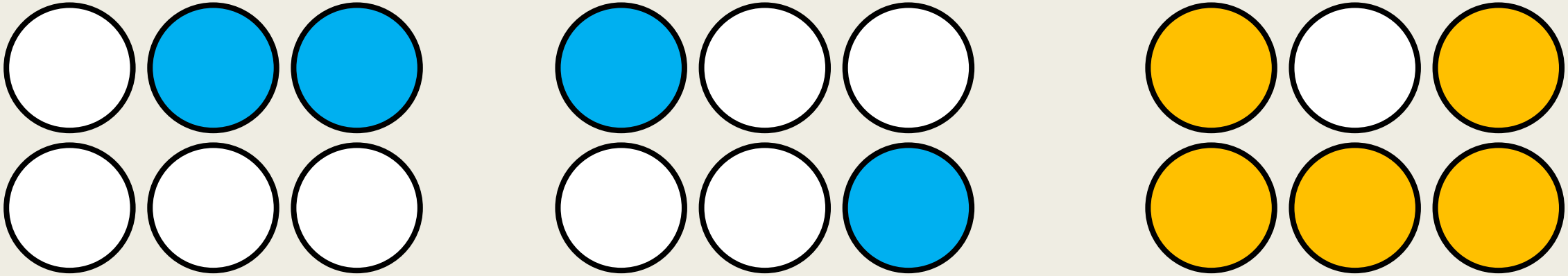
Non-Strict Majorities



Replication quorum

Leader election quorum

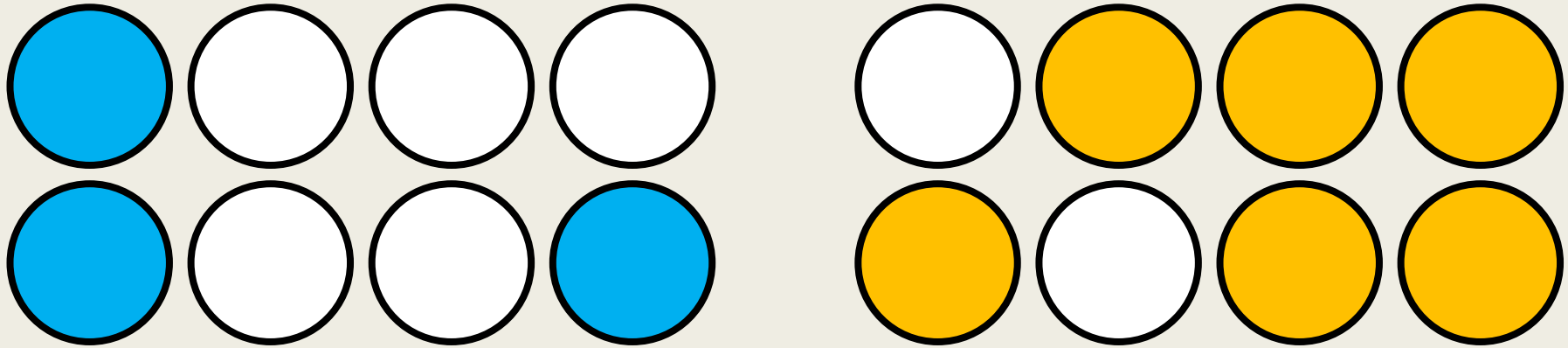
Counting quorums

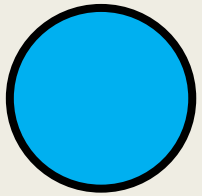


Replication quorum

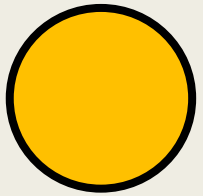
Leader election quorum

Counting quorums



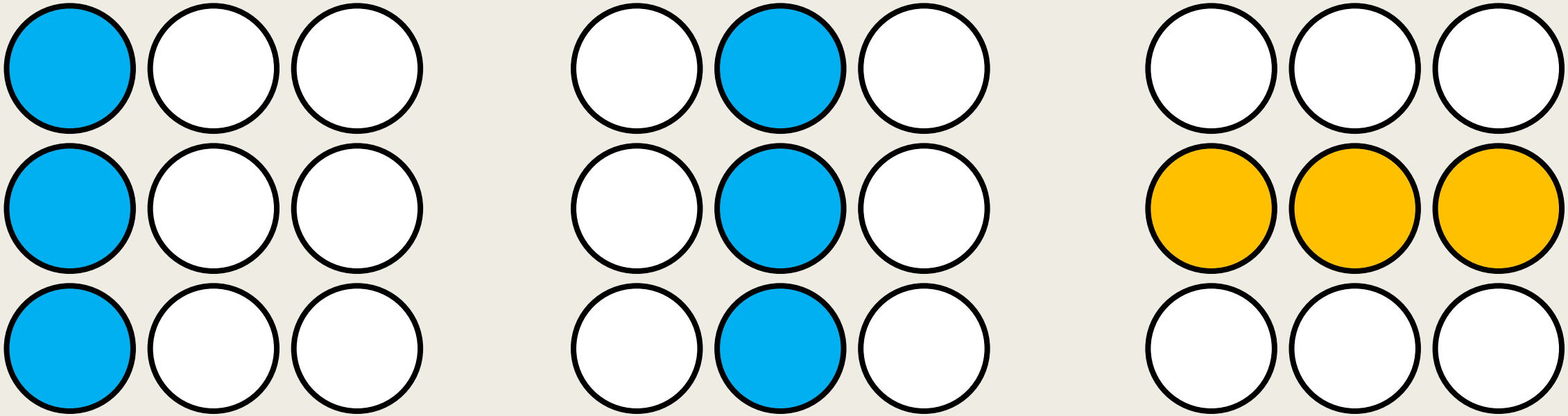


Replication quorum



Leader election quorum

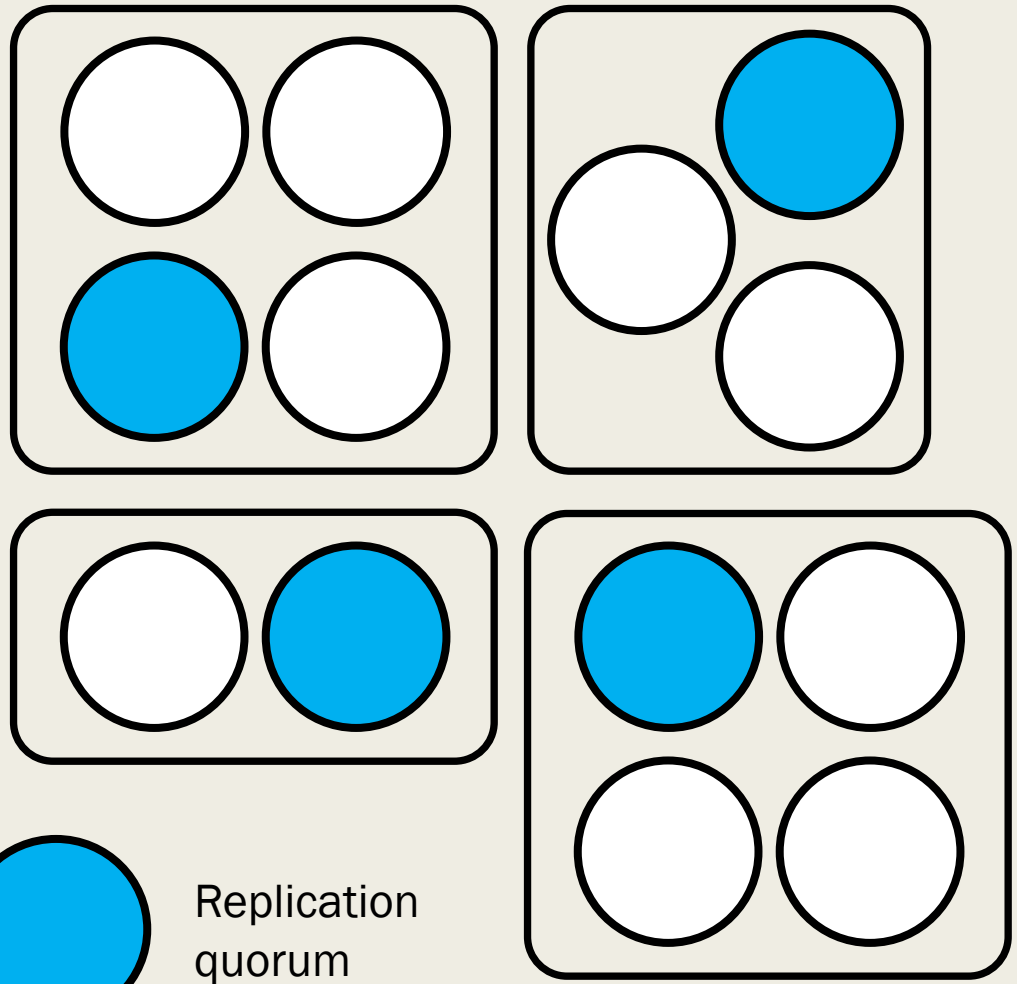
Grid quorums



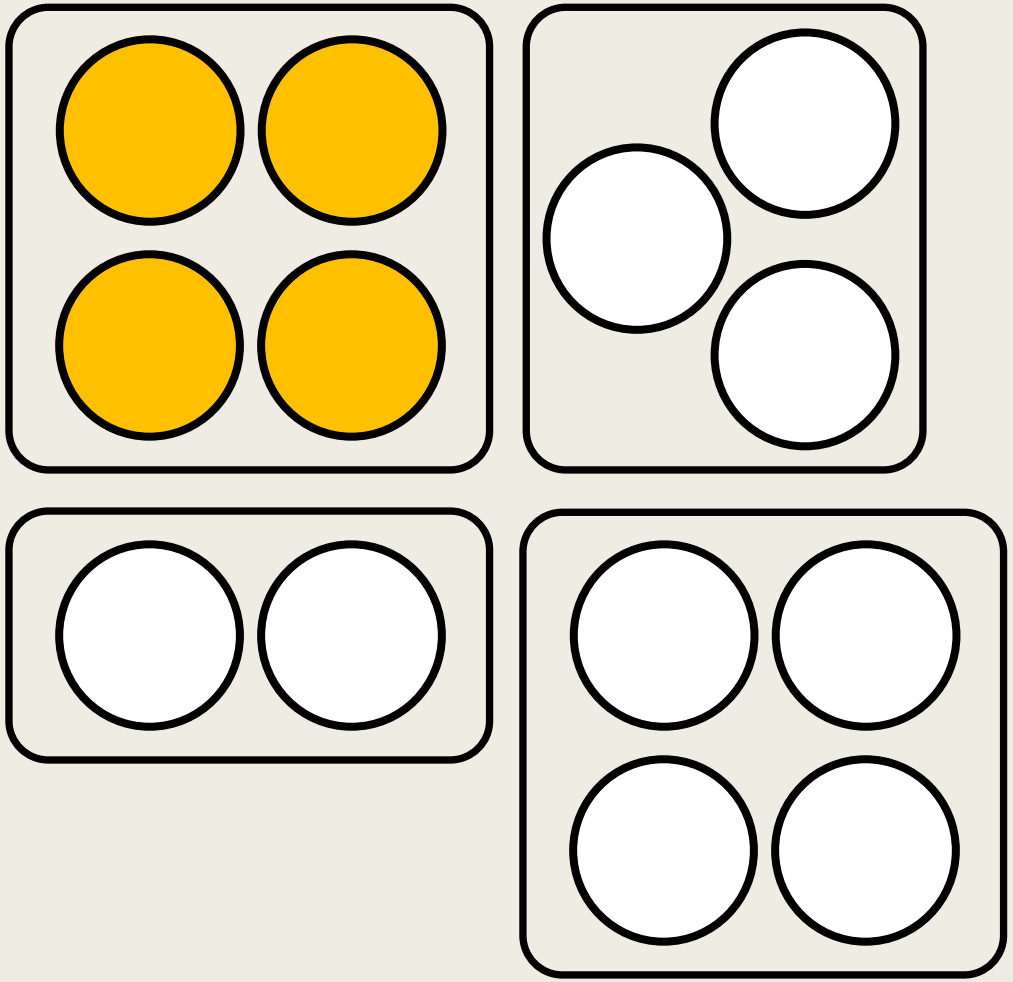
Replication quorum

Leader election quorum

Group quorums

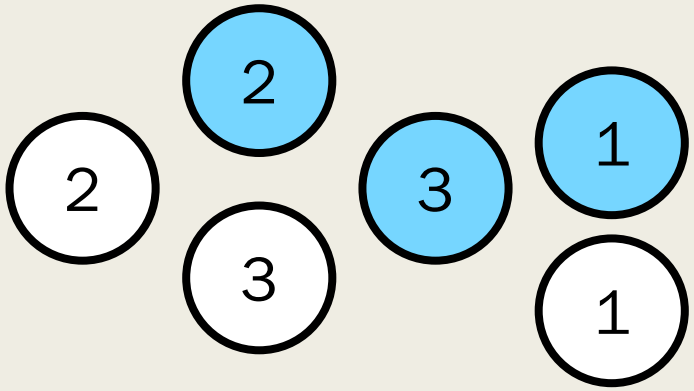


Replication quorum

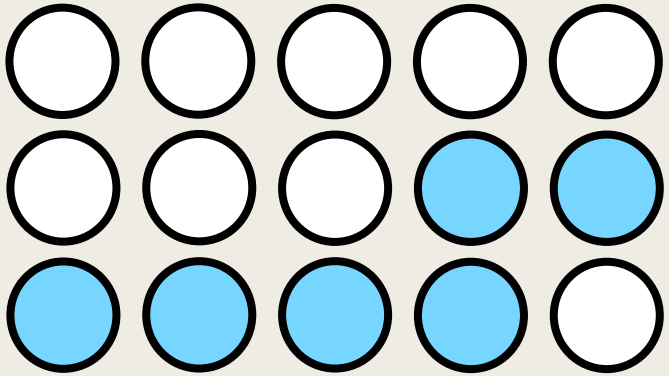


Leader election quorum

And many more quorum systems

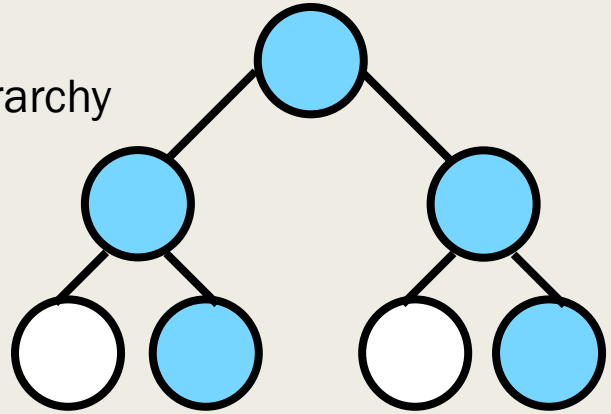


Weighed Voting

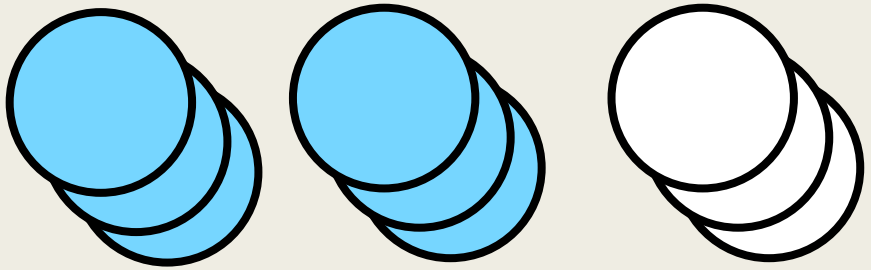


Grid Paths

Hierarchy



Combined Schemes



Flexible Paxos teaches us something important about distributed systems research: sometimes the biggest insights are the simplest. The quorum intersection requirement was always the fundamental invariant. Majority quorums were always just one way to achieve it. But the field was so focused on building on top of the majority assumption that nobody thought to question it.

This should make us wonder: what other simplifying assumptions in our foundational protocols are unnecessarily restrictive? What other degrees of freedom are we leaving on the table because “that is how it has always been done”?

In the meantime, the next time someone tells you that consensus requires a majority, you can smile and say, “Well, actually…” Just be prepared for the blank stares. Some insights, no matter how simple, take decades to sink in.

The Agony of Consensus Algorithms

THE DECADE SINCE

Even servers no longer considered always harmful

- Sometimes we want (or need) to deploy consensus systems over an even number of servers
- This is unwise with simple majority quorums
  - *Four servers were worse than three*
  - *Both tolerate one server failure but there's one more server which can fail plus the overhead of a larger replication quorum*
- With Flexible Paxos, we can switch to using exactly half the number of servers for replication quorums

Distributed consensus is not just Paxos

- Paxos was never the only solution to consensus
- Consensus algorithms include Viewstamped Replication, Zab, Raft, EPaxos and countless others
  - *Talk to James Cowling (or anyone from the TigerBeetle team)*
- Different communities use different abstractions and terminology but the fundamentals are similar
- We can relax their quorum assumptions too (caveats apply)
  - *Leaderless consensus algorithms famously use many servers and larger quorums due to even stronger quorum interception requirements. These can be relaxed too*

Fast Flexible Paxos: Relaxing Quorum Intersection for Fast Paxos

Heidi Howard
University of Cambridge
heidi.howard@cl.cam.ac.uk

Aleksey Charapko
University of New Hampshire
aleksey.charapko@unh.edu

Richard Mortier
University of Cambridge
richard.mortier@cl.cam.ac.uk

ABSTRACT

Paxos, the de facto standard approach to solving distributed consensus, operates in two phases, each of which requires an intersecting quorum of nodes. Multi-Paxos reduces this to one phase by electing a leader but this leader is also a performance bottleneck. Fast Paxos bypasses the leader but has stronger quorum intersection requirements.

In this paper we observe that Fast Paxos' intersection requirements can be safely relaxed, reducing to just one additional intersection requirement between phase-1 quorums and any pair of fast round phase-2 quorums. We thus find that the quorums used with Fast Paxos are larger than necessary, allowing alternative quorum systems to obtain new tradeoffs between performance and fault-tolerance.

performance by adjusting quorums depending on the phase of the algorithm [1, 4, 8, 9, 28, 35].

Paxos is usually implemented using Multi-Paxos [18, 19], an optimization that elects one node to be a *leader*. This single leader can then achieve distributed consensus in just one phase, but unfortunately also becomes a performance bottleneck.

Seeking to improve performance, a new family of *leaderless* consensus algorithms emerged, starting with Fast Paxos [23], which forms the basis for subsequent algorithms including Generalized Paxos [22] and Egalitarian Paxos [26]. Paxos uses the idea of *rounds* in which at most one value can be *proposed*. Fast Paxos introduced the notion of *fast rounds* where multiple values can be safely proposed in the same round. However, such fast rounds require stronger quorum intersection than classical rounds. Specifically, Paxos only requires that any two quorums intersect whereas Fast

[ICDCN 2021]

All people are created equal, the same is not true of servers

- Paxos and friends treat all servers the same
  - *Assuming they are equally quick to reply and fail independently*
- However, ideally servers are geo-graphically distributed
  - *Servers in the same AZ are more likely to fail together*
  - *Servers in different AZs are likely to be slower to respond*
- Let's make our consensus algorithms aware of server locality
- We could use grid/group quorums but we can do much more

WPaxos: Wide Area Network Flexible Consensus

Ailidani Ailijiang, Aleksey Charapko, Murat Demirbas and Tevfik Kosar
Department of Computer Science and Engineering
University at Buffalo, SUNY
Email: {ailidani,acharapk,demirbas,tkosar}@buffalo.edu

Abstract—WPaxos is a multileader Paxos protocol that provides low-latency and high-throughput consensus across wide-area network (WAN) deployments. WPaxos uses multileaders, and partitions the object-space among these multileaders. Unlike statically partitioned multiple Paxos deployments, WPaxos is able to adapt to the changing access locality through object stealing. Multiple concurrent leaders coinciding in different zones steal ownership of objects from each other using phase-1 of Paxos, and then use phase-2 to commit update-requests on these objects locally until they are stolen by other leaders. To achieve fast phase-2 commits, WPaxos adopts the flexible quorums idea in a novel manner, and appoints phase-2 acceptors to be close to their respective leaders. We implemented WPaxos and evaluated it on WAN deployments across 5 AWS regions. The dynamic partitioning of the object-space and emphasis on zone-local commits allow WPaxos to significantly outperform both partitioned Paxos deployments and leaderless Paxos approaches.

[PDS 2020]

DPaxos: Managing Data Closer to Users for Low-Latency and Mobile Applications

Faisal Nawab
University of California, Santa Cruz
Santa Cruz, CA
fnawab@ucsc.edu

Divyakant Agrawal
Amr El Abbadi
University of California, Santa Barbara
Santa Barbara, CA
[agrawal,amr]@cs.ucsb.edu

ABSTRACT

In this paper, we propose *Dynamic Paxos* (DPaxos), a Paxos-based consensus protocol to manage access to partitioned data across globally-distributed datacenters and edge nodes. DPaxos is intended to implement a State Machine Replication component in data management systems for the edge. DPaxos targets the unique opportunities of utilizing edge computing resources to support emerging applications with stringent mobility and real-time requirements such as Augmented and Virtual Reality and vehicular applications. The main objective of DPaxos is to reduce the latency of serving user requests, recovering from failures, and reacting to mobility. DPaxos achieves these objectives by a few proposed changes to the traditional Paxos protocol. Most notably, DPaxos proposes a *dynamic* allocation of quorums (*i.e.*, groups of nodes) that are needed for Paxos Leader Election. Leader Election quorums in DPaxos are smaller than traditional Paxos and *expand* only in the presence of conflicts.

delivery networks utilize edge locations. However, rather than edge locations being used for data caching only, they will also host data management components that will allow manipulation and querying of local edge partitions. In this paper, we focus on transaction processing as the data management task to be supported by the edge data management components. The model and focus of this paper aims to serve web and cloud applications, such as online shops, social networks, and collaborative applications.

In this paper, we propose *Dynamic Paxos* (DPaxos), a Paxos-based consensus protocol [21, 22]. DPaxos is intended to be used as the State Machine Replication (SMR) component in data management systems for the edge. In such systems, we envision utilizing edge nodes around the world by partitioning the data and placing each data partition at the edge node closest to its corresponding users. DPaxos, as a SMR component, manages access to data partitions in the form of consistent and fault-tolerant transactions and commands. DPaxos targets harnessing the benefits of deploying on

[SIGMOD 2018]

Consensus protocol can scale

- Conventual wisdom is that distributed consensus is inherently slow so best avoided whenever possible
- Consensus does not scale with the number of servers
- In fact, adding more servers makes consensus systems slower!
- However, this need not be the case

Scaling Replicated State Machines with Compartmentalization

Michael Whittaker
UC Berkeley
mjwhittaker@berkeley.edu

Ailidani Ailijiang
Microsoft
aiailiji@microsoft.com

Aleksey Charapko
University of New Hampshire
aleksey.charapko@unh.edu

Murat Demirbas
University at Buffalo
demirbas@buffalo.edu

Neil Giridharan
UC Berkeley
giridhn@berkeley.edu

Joseph M. Hellerstein
UC Berkeley
hellerstein@berkeley.edu

Heidi Howard
University of Cambridge
hh360@cst.cam.ac.uk

Ion Stoica
UC Berkeley
istoica@berkeley.edu

Adriana Szekeres
VMWare
aszekeres@vmware.com

ABSTRACT

State machine replication protocols, like MultiPaxos and Raft, are a critical component of many distributed systems and databases. However, these protocols offer relatively low throughput due to several bottlenecked components. Numerous existing protocols fix different bottlenecks in isolation but fall short of a complete solution. When you fix one bottleneck, another arises. In this paper, we introduce compartmentalization, the first comprehensive technique to eliminate state machine replication bottlenecks. Compartmentalization involves decoupling individual bottlenecks into distinct components and scaling these components independently. Compartmentalization has two key strengths. First, compartmentalization leads to strong performance. In this paper, we demonstrate how to compartmentalize MultiPaxos to increase its throughput by 6× on a write-only workload and 16× on a mixed read-write workload. Unlike other approaches, we achieve this performance without the need for specialized hardware. Second, compartmentalization is a technique, not a protocol. Industry practitioners can apply compartmentalization to their protocols incrementally without having to adopt a completely new protocol.

many state machine replication protocols, a single node has multiple responsibilities. For example, a Raft [29] leader acts as a batcher, a sequencer, a broadcaster, *and* a state machine replica. These overloaded nodes are often a throughput bottleneck, which can be disastrous for systems that rely on state machine replication.

Many databases, for example, rely on state machine replication to replicate large data partitions of tens of gigabytes [2, 34]. These databases require high-throughput state machine replication to handle all the requests in a partition. However, in such systems, it is not uncommon to exceed the throughput budget of a partition. For example, Cosmos DB will split a partition if it experiences high throughput despite being under the storage limit. The split, aside from costing resources, may have additional adverse effects on applications, as Cosmos DB provides strongly consistent transactions only within the partition. Eliminating state machine replication bottlenecks can help avoid such unnecessary partition splits and improve performance, consistency, and resource utilization.

Researchers have studied how to eliminate throughput bottlenecks, often by inventing new state machine replication protocols that eliminate a *single* throughput bottleneck [5, 6, 9, 12, 17, 22, 23, 25, 26, 37, 44]. However, eliminating a *single* bottleneck is not

[VLDB 2021]

Optimizing Distributed Protocols with Query Rewrites

DAVID C. Y. CHU, University of California, Berkeley, USA

RITHVIK PANCHAPAKESAN, University of California, Berkeley, USA

SHADAJ LADDAD, University of California, Berkeley, USA

LUCKY E. KATAHANAS, Sutter Hill Ventures, USA

CHRIS LIU, University of California, Berkeley, USA

KAUSHIK SHIVAKUMAR, University of California, Berkeley, USA

NATACHA CROOKS, University of California, Berkeley, USA

JOSEPH M. HELLERSTEIN, University of California, Berkeley, USA and Sutter Hill Ventures, USA

HEIDI HOWARD, Azure Research, Microsoft, UK

Distributed protocols such as 2PC and Paxos lie at the core of many systems in the cloud, but standard implementations do not scale. New scalable distributed protocols are developed through careful analysis and rewrites, but this process is ad hoc and error-prone. This paper presents an approach for scaling *any* distributed protocol by applying rule-driven rewrites, borrowing from query optimization. Distributed protocol rewrites entail a new burden: reasoning about spatiotemporal correctness. We leverage order-insensitivity and data dependency analysis to systematically identify correct coordination-free scaling opportunities. We apply this analysis to create preconditions and mechanisms for coordination-free decoupling and partitioning, two fundamental vertical and horizontal scaling techniques. Manual rule-driven applications of decoupling and partitioning improve the throughput of 2PC by 5× and Paxos by 3×, and match state-of-the-art throughput in recent work. These results point the way toward automated optimizers for distributed protocols based on correct-by-construction rewrite rules.

[SIGMOD 2024]

Servers don’t need to be trusted

- These consensus protocols assume all servers follow the algorithm perfectly
  - *This is particularly tricky in the cloud computing era*
- Byzantine consensus can tolerate a threshold of malicious servers but its expensive
- *Confidential computing* allows us servers to remotely prove that they running the correct consensus code
- This is the basis of the Confidential Consortium Framework

Confidential Consortium Framework: Secure Multiparty Applications with Confidentiality, Integrity, and High Availability

Heidi Howard*
Azure Research, Microsoft

Fritz Alder†
imec-DistriNet, KU Leuven
Belgium

Edward Ashton
Azure Research, Microsoft

Amaury Chamayou
Azure Research, Microsoft

Sylvan Clebsch
Azure Research, Microsoft

Manuel Costa
Azure Research, Microsoft

Antoine Delignat-Lavaud
Azure Research, Microsoft

Cédric Fournet
Azure Research, Microsoft

Andrew Jeffery†
University of Cambridge
UK

Matthew Kerner
Microsoft

Fotios Kounelis†
Imperial College London
UK

Markus A. Kuppe
Microsoft Research

Julien Maffre
Azure Research, Microsoft

Mark Russinovich
Microsoft

Christoph M. Wintersteiger
Azure Research, Microsoft

ABSTRACT

Confidentiality, integrity protection, and high availability, abbreviated to CIA, are essential properties for trustworthy data systems. The rise of cloud computing and the growing demand for multiparty applications however means that building modern CIA systems is more challenging than ever. In response, we present the Confidential Consortium Framework (CCF), a general-purpose foundation for developing secure stateful CIA applications. CCF combines centralized compute with decentralized trust, supporting deployment on untrusted cloud infrastructure and transparent governance by mutually untrusted parties.

CCF leverages hardware-based trusted execution environments for remotely verifiable confidentiality and code integrity. This is coupled with state machine replication backed by an auditable immutable ledger for data integrity and high availability. CCF enables each service to bring its own application logic, custom multiparty governance model, and deployment scenario, decoupling the operators of nodes from the consortium that governs them. CCF is open-source and available now at https://github.com/microsoft/CCF.

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/microsoft/CCF.

1 INTRODUCTION & MOTIVATION

Together data confidentiality, integrity protection, and high availability form the CIA triad, a classic formulation in information security [128]. We begin by examining each property, with a particular focus on running trustworthy multiparty applications on untrusted infrastructure.

Data confidentiality. Organizations are responsible for ensuring the privacy of personal data. Increasingly this responsibility is codified in law, and the penalties for failing to fulfill this obligation can be substantial, for instance up to 4% of turnover in the case of GDPR [96]. Even where data is not personal, companies may wish to keep data confidential to protect intellectual property, for competitive advantage, or to maintain systems security, for instance when storing secrets. Encryption at rest and in-flight are well-established approaches to achieving confidentiality, but confidentiality during execution is more challenging. Moreover

[VLDB 2023]

Formal methods are becoming viable

- There's understandable fear about experimenting with the fundamentals of distributed consensus
  - *Reasoning about distributed systems is complex and error prone*
  - *Many feel it's best left to the likes of Barbara Liskov and Leslie Lamport*
- Formal methods, from model checking and formal specification to fully verified implementations, are becoming increasing viable
  - *Talk to Hillel Wayne*
- For instance, Leslie proved a formal specification of Paxos but there's no guarantee that a real-world systems faithfully implements this spec

Smart Casual Verification of the Confidential Consortium Framework

Heidi Howard[*], *Markus A. Kuppe*[*], *Edward Ashton*[*], *Amaury Chamayou*[*], *Natacha Crooks*[*†]
Azure Research, Microsoft[*] *UC Berkeley*[†]

Abstract

The Confidential Consortium Framework (CCF) is an open-source platform for developing trustworthy and reliable cloud applications. CCF powers Microsoft's Azure Confidential Ledger service and as such it is vital to build confidence in the correctness of CCF's design and implementation. This paper reports our experiences applying *smart casual verification* to validate the correctness of CCF's novel distributed protocols, focusing on its unique distributed consensus protocol and its custom client consistency model. We use the term smart casual verification to describe our hybrid approach, which combines the rigor of formal specification and model checking with the pragmatism of automated testing, in our case binding the formal specification in TLA$^+$ to the C++ implementation. While traditional formal methods approaches require substantial buy-in and are often one-off efforts by domain experts, we have integrated our smart casual verification approach into CCF's CI pipeline, allowing contributors to continuously validate CCF as it evolves. We describe the challenges we faced in applying smart casual verification to a complex existing codebase and how we overcame them to find six subtle bugs in the design and implementation before they could impact production.

for integrity-protection with SQL Server [3,63] and Azure Immutable Blob Storage [35]. Open source applications of CCF range from code transparency [17,58] and mediation of multi-party data sharing [43], to decentralized identity [59], privacy-preserving ad auctions [62], and confidential storage [32,56].

This paper summarizes our experience with verifying the correctness guarantees of the distributed protocols in CCF, a large-scale, production distributed system, using TLA$^+$ [45, 47]. We have five primary requirements that guided our approach to verification:

(1) Verify high-level distributed safety properties. We aim to check the correctness of our custom distributed consensus protocol, including its dynamic reconfiguration logic. CCF's consensus logic, though based on Raft [74] has been sufficiently modified such that it is now based on an unproven algorithm. This is a common problem: the Chubby authors [8] similarly note that the additional requirements imposed by a real-world system over a vanilla Paxos [46] implementation require significant changes. Moreover, distributed consensus protocols are notoriously hard to get right, and even well-known protocols, including PBFT [9], Egalitarian Paxos [66], and Zyzzyva [39], as well as the previously mentioned Raft and Chubby, have been found to contain subtle bugs [1, 2, 4, 55, 72, 82, 90]. The

[NSDI 2025]

Thank you!

- Consensus algorithms are fundamental to building distributed systems with strong guarantees
- We have long established ways of solving consensus but we are not limited to them
- Let’s continue to challenge our assumptions and rethink the foundations
- We can (and should!) build better consensus algorithms
- Now is the time we ask more of our distributed systems