Flexible Paxos

Revisiting quorum intersection using TLA+

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

Dr TLA+ Series
github.com/tlaplus/DrTLAPlus

Outline

Background - Revisiting the problem of distributed consensus and how Paxos addresses it.

Flexible Paxos - Introducing the revised quorum intersection requirement for Paxos.

Quorum systems - An overview of various alternatives to majorities.

TLA+ - Extending the TLA+ spec for Paxos to use Flexible Paxos and checking it.

Background

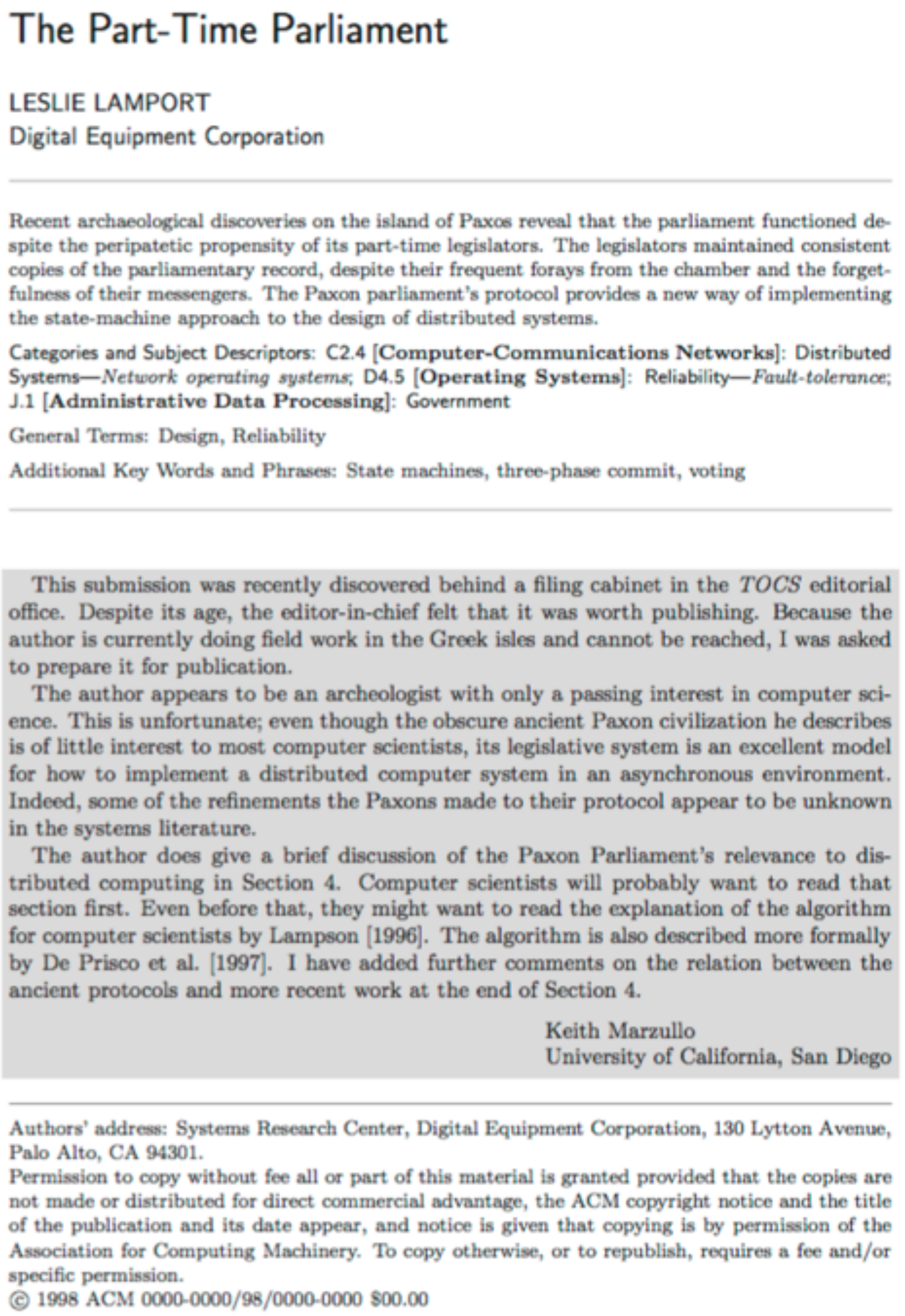
Revisiting the problem of distributed consensus and how Paxos addresses it.

Defining Consensus

Reaching agreement in an **asynchronous** distributed system in the face of **crash failures**.

More specifically:

- Compromising safety is never an option
- Full clock synchronization is not assumed
- Participants fail by crashing and messages may be lost but neither can act arbitrarily



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

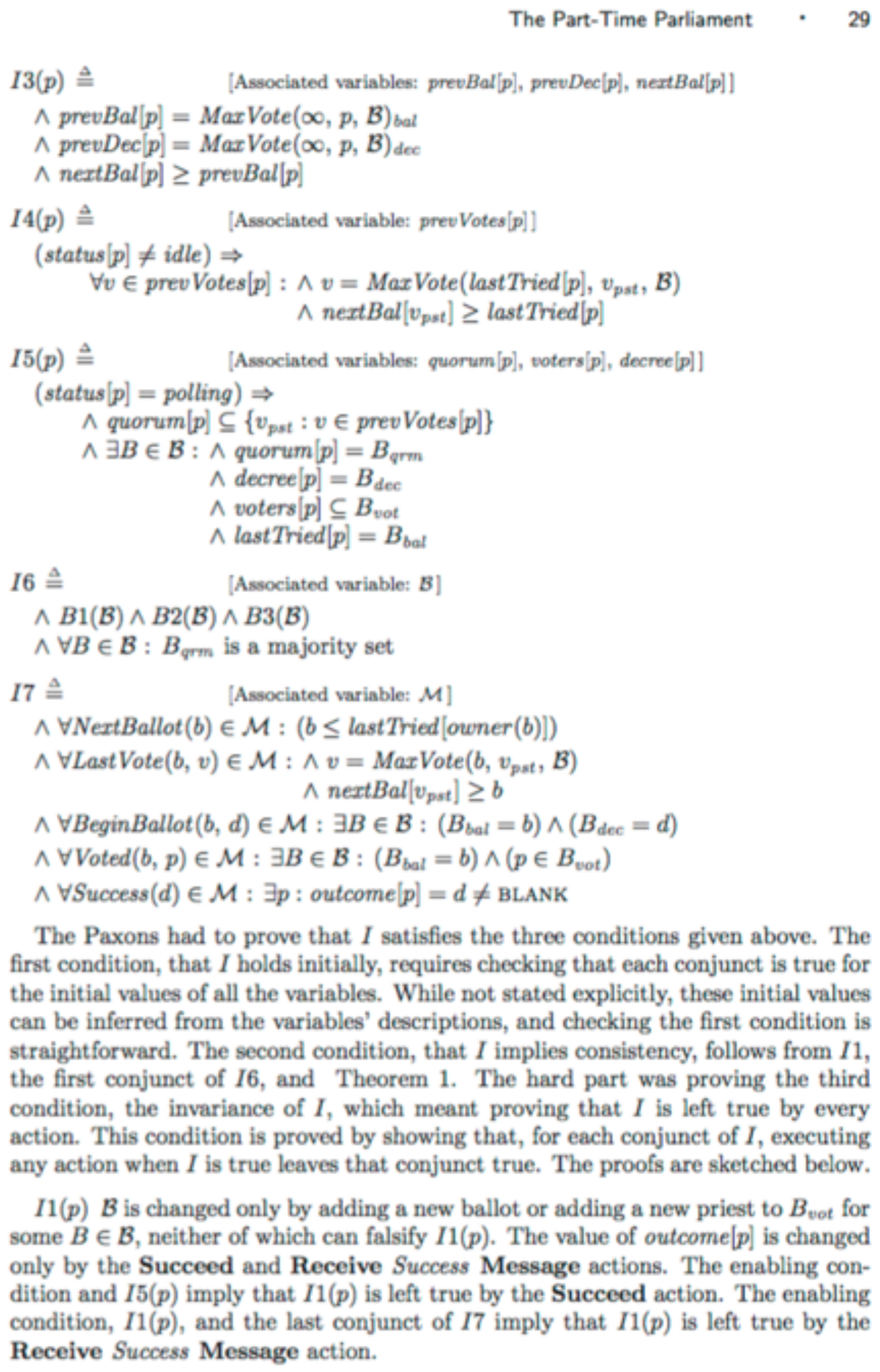
This submission was recently discovered behind a filing cabinet in the *TOCS* editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.

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

Authors' address: Systems Research Center, Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, CA 94301.



$I3(p) \triangleq$ [Associated variables: $prevBal[p]$, $prevDec[p]$, $nextBal[p]$]
$\wedge\ prevBal[p] = MaxVote(\infty, p, \mathcal{B})_{bal}$
$\wedge\ prevDec[p] = MaxVote(\infty, p, \mathcal{B})_{dec}$
$\wedge\ nextBal[p] \geq prevBal[p]$

$I4(p) \triangleq$ [Associated variable: $prevVotes[p]$]
$(status[p] \neq idle) \Rightarrow$
$\quad \forall v \in prevVotes[p] : \wedge\ v = MaxVote(lastTried[p], v_{pst}, \mathcal{B})$
$\quad\quad \wedge\ nextBal[v_{pst}] \geq lastTried[p]$

$I5(p) \triangleq$ [Associated variables: $quorum[p]$, $voters[p]$, $decree[p]$]
$(status[p] = polling) \Rightarrow$
$\quad \wedge\ quorum[p] \subseteq \{v_{pst} : v \in prevVotes[p]\}$
$\quad \wedge\ \exists B \in \mathcal{B} : \wedge\ quorum[p] = B_{qrm}$
$\quad\quad \wedge\ decree[p] = B_{dec}$
$\quad\quad \wedge\ voters[p] \subseteq B_{vot}$
$\quad\quad \wedge\ lastTried[p] = B_{bal}$

$I6 \triangleq$ [Associated variable: $\mathcal{B}$]
$\wedge\ B1(\mathcal{B}) \wedge B2(\mathcal{B}) \wedge B3(\mathcal{B})$
$\wedge\ \forall B \in \mathcal{B} : B_{qrm}$ is a majority set

$I7 \triangleq$ [Associated variable: $\mathcal{M}$]
$\wedge\ \forall NextBallot(b) \in \mathcal{M} : (b \leq lastTried[owner(b)])$
$\wedge\ \forall LastVote(b, v) \in \mathcal{M} : \wedge\ v = MaxVote(b, v_{pst}, \mathcal{B})$
$\quad\quad \wedge\ nextBal[v_{pst}] \geq b$
$\wedge\ \forall BeginBallot(b, d) \in \mathcal{M} : \exists B \in \mathcal{B} : (B_{bal} = b) \wedge (B_{dec} = d)$
$\wedge\ \forall Voted(b, p) \in \mathcal{M} : \exists B \in \mathcal{B} : (B_{bal} = b) \wedge (p \in B_{vot})$
$\wedge\ \forall Success(d) \in \mathcal{M} : \exists p : outcome[p] = d \neq$ BLANK

The Paxons had to prove that I satisfies the three conditions given above. The first condition, that I holds initially, requires checking that each conjunct is true for the initial values of all the variables. While not stated explicitly, these initial values can be inferred from the variables' descriptions, and checking the first condition is straightforward. The second condition, that I implies consistency, follows from $I1$, the first conjunct of $I6$, and Theorem 1. The hard part was proving the third condition, the invariance of I, which meant proving that I is left true by every action. This condition is proved by showing that, for each conjunct of I, executing any action when I is true leaves that conjunct true. The proofs are sketched below.

$I1(p)$ $\mathcal{B}$ is changed only by adding a new ballot or adding a new priest to B_{vot} for some $B \in \mathcal{B}$, neither of which can falsify $I1(p)$. The value of $outcome[p]$ is changed only by the **Succeed** and **Receive** *Success* **Message** actions. The enabling condition and $I5(p)$ imply that $I1(p)$ is left true by the **Succeed** action. The enabling condition, $I1(p)$, and the last conjunct of $I7$ imply that $I1(p)$ is left true by the **Receive** *Success* **Message** action.

[TOCS'98]

Consensus is not a solved problem

- **Consensus is not scalable** - often limited to 3, 5 or 7 participants.
- **Consensus is slow** - each decision requires at least majority agreement, more when failures occur.
- **Consensus is not available** - cannot handle majority failures or many network partitions.

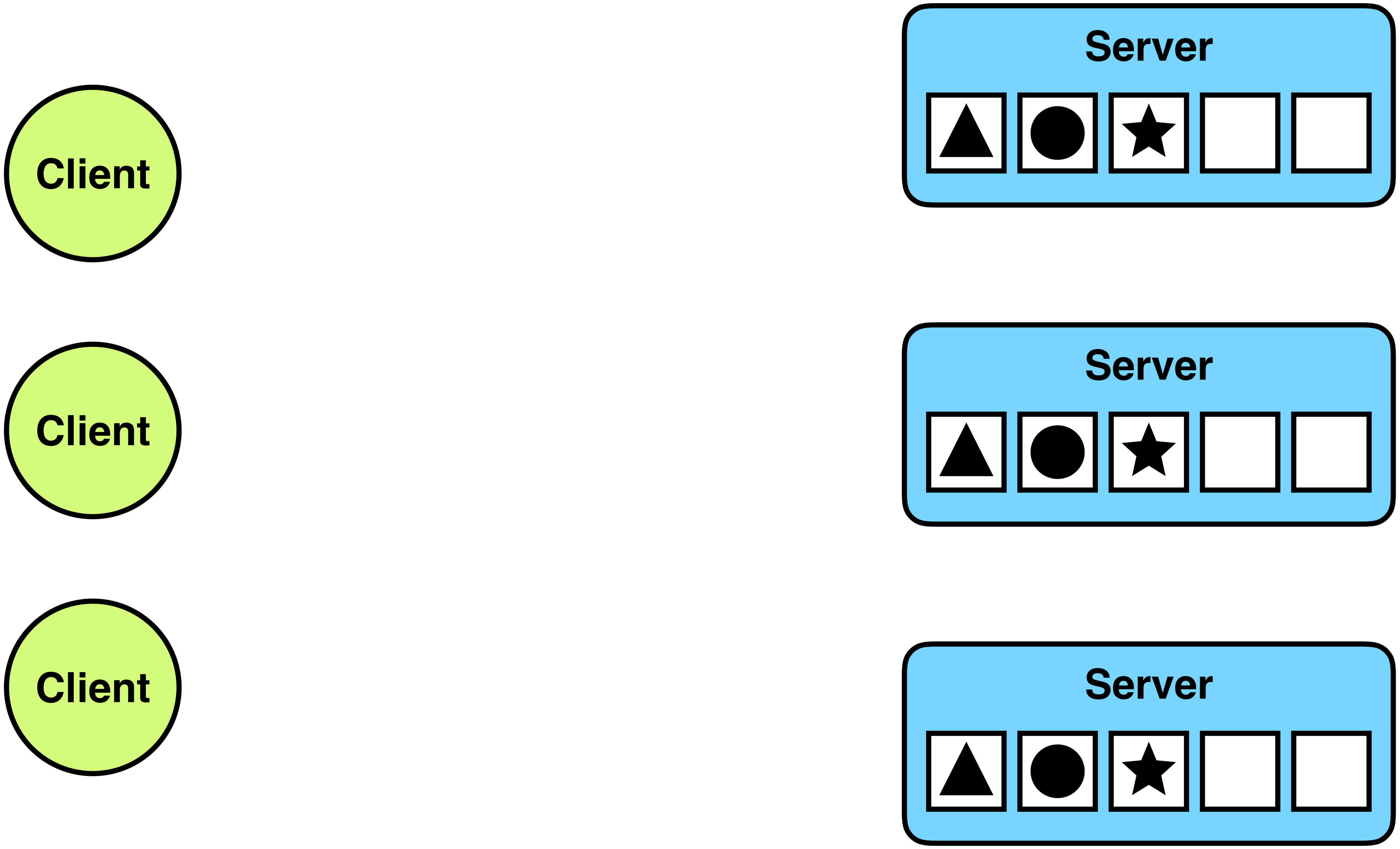
You pay a high price even if no failures occur. Consider by many as “best avoided”

Paxos 101

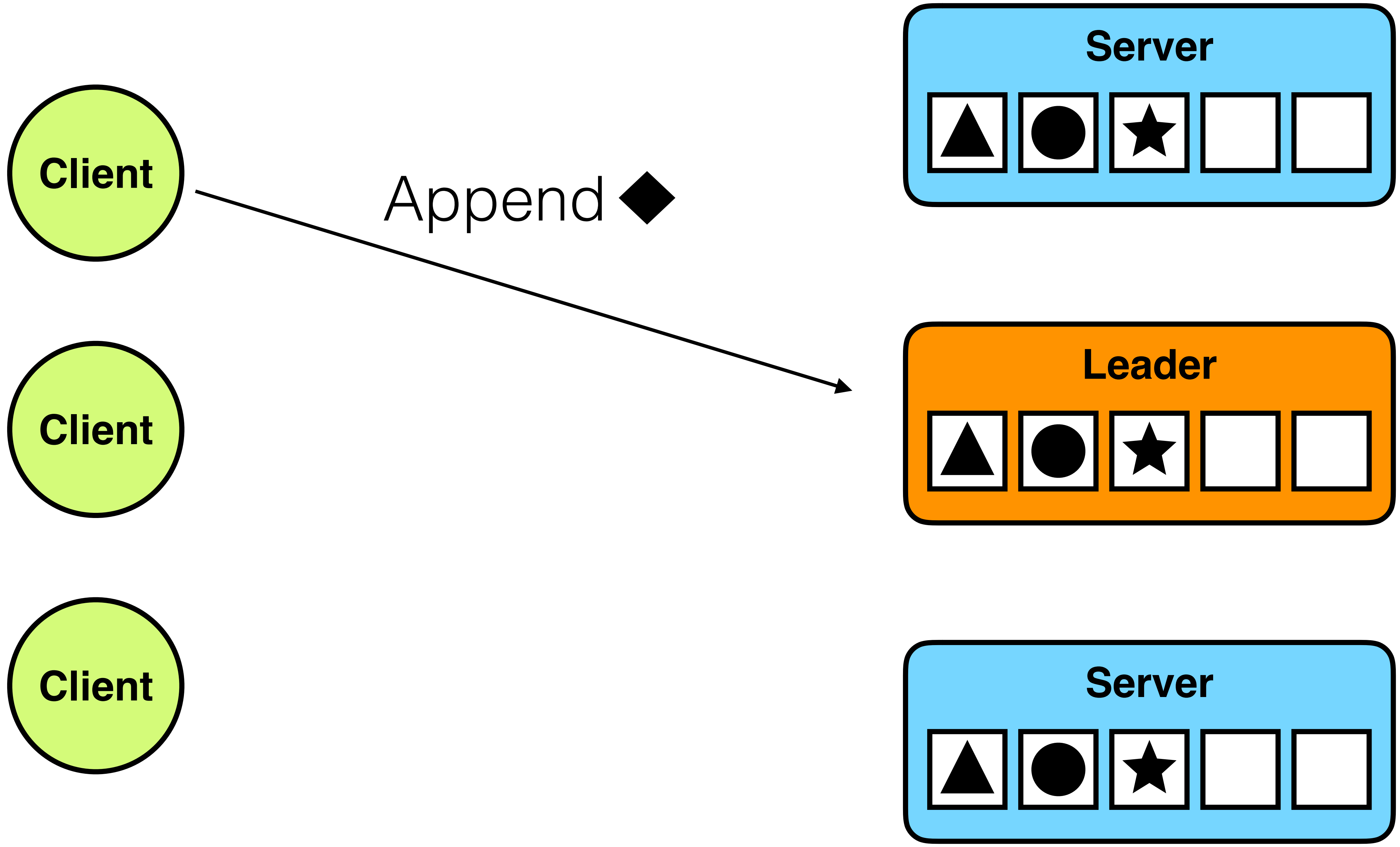
Not true to the original formation.

Focus on state machine replication and multi-value agreement.

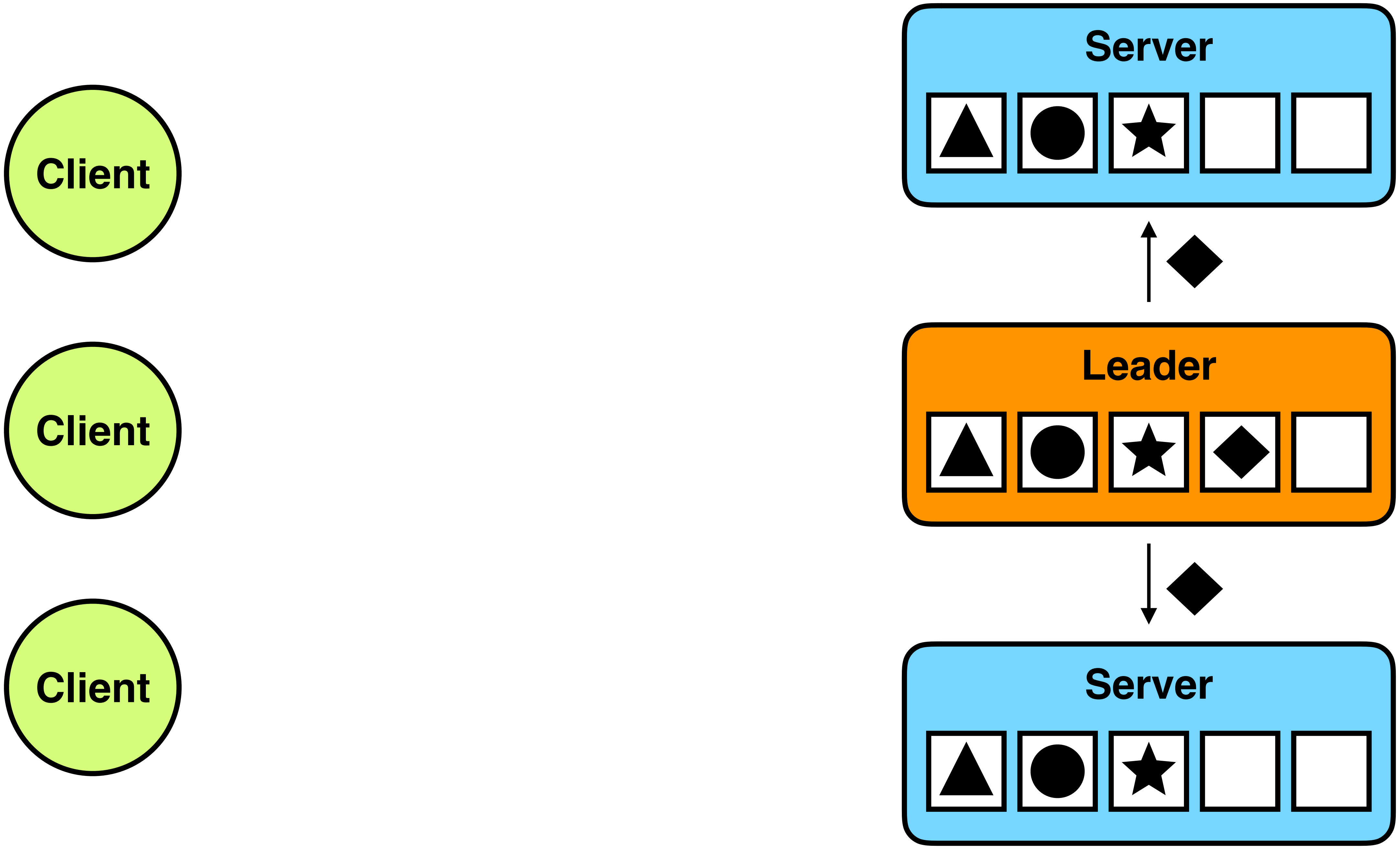
Replicate for availability



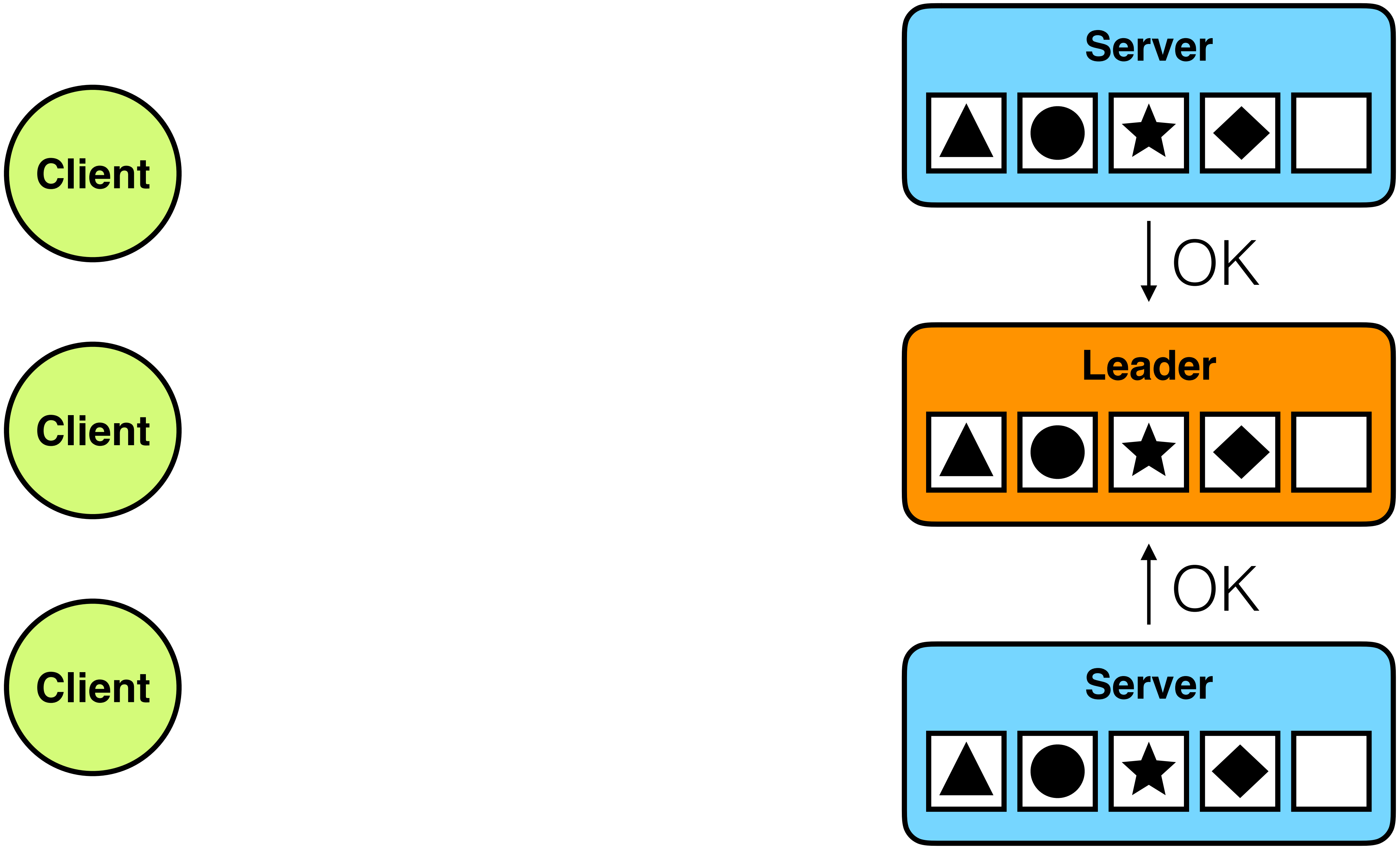
Replicate for availability



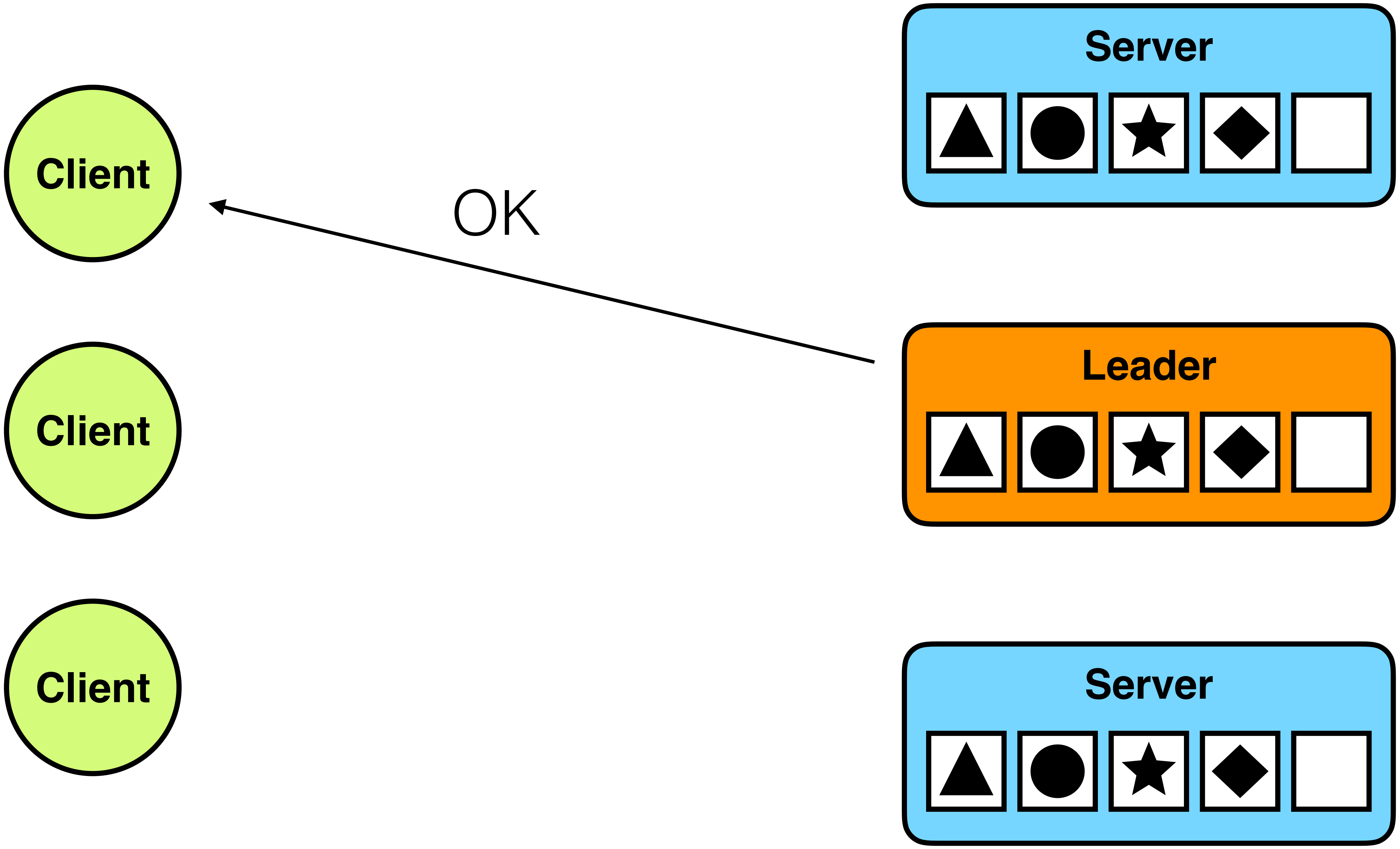
Replicate for availability



Replicate for availability



Replicate for availability



How many copies is enough?

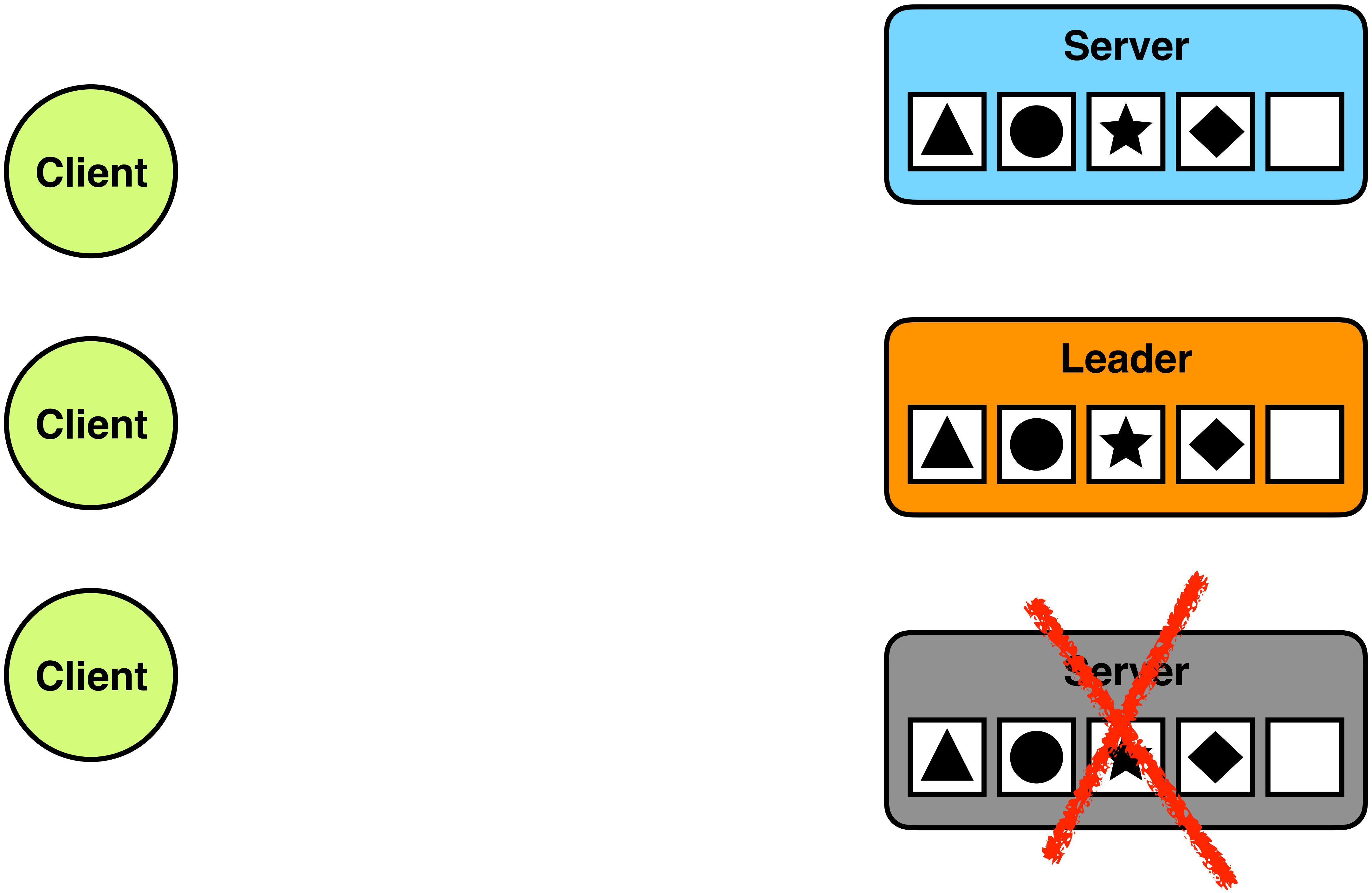
More copies,
More resilient

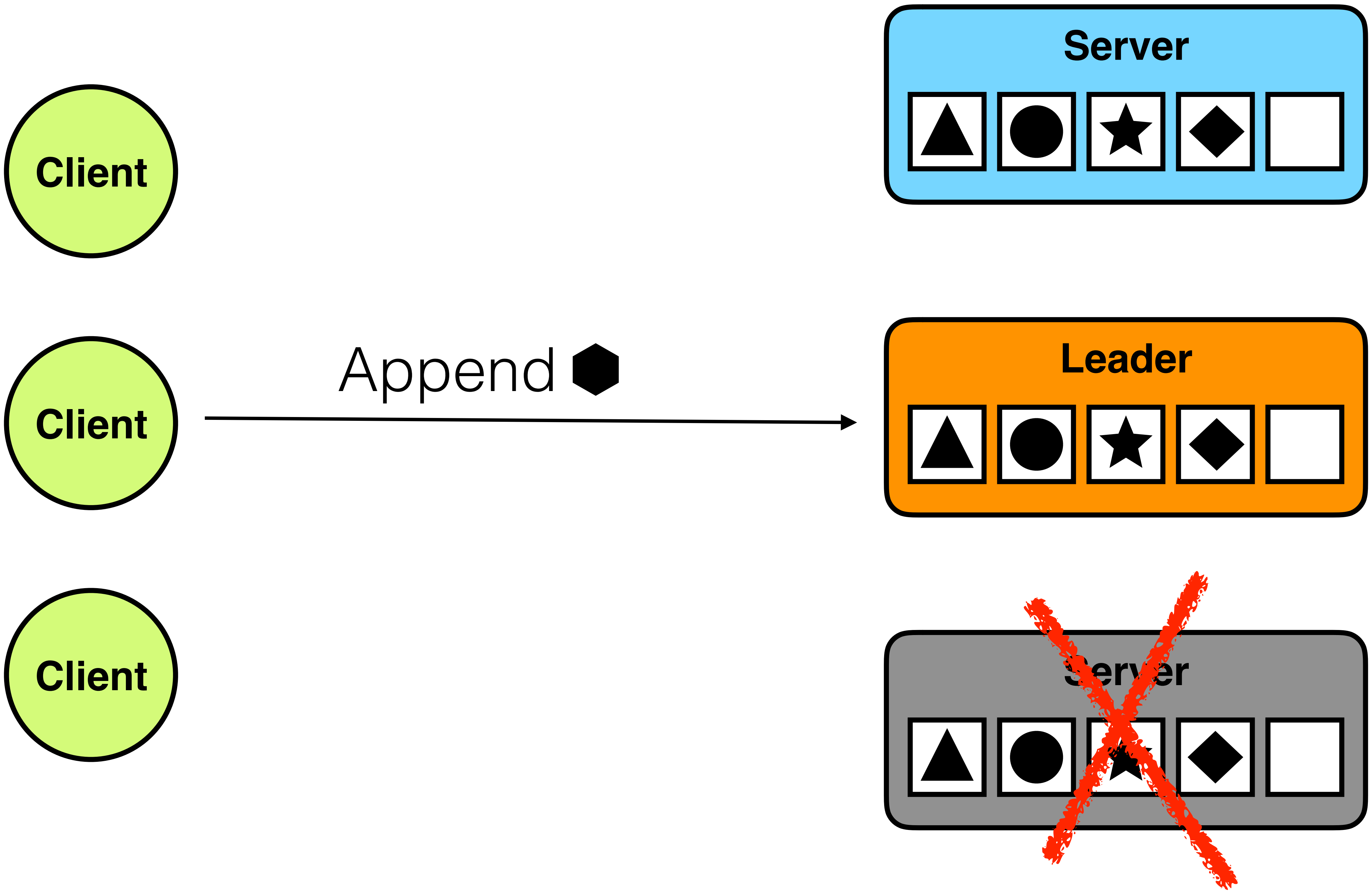
Fewer copies,
Faster replication

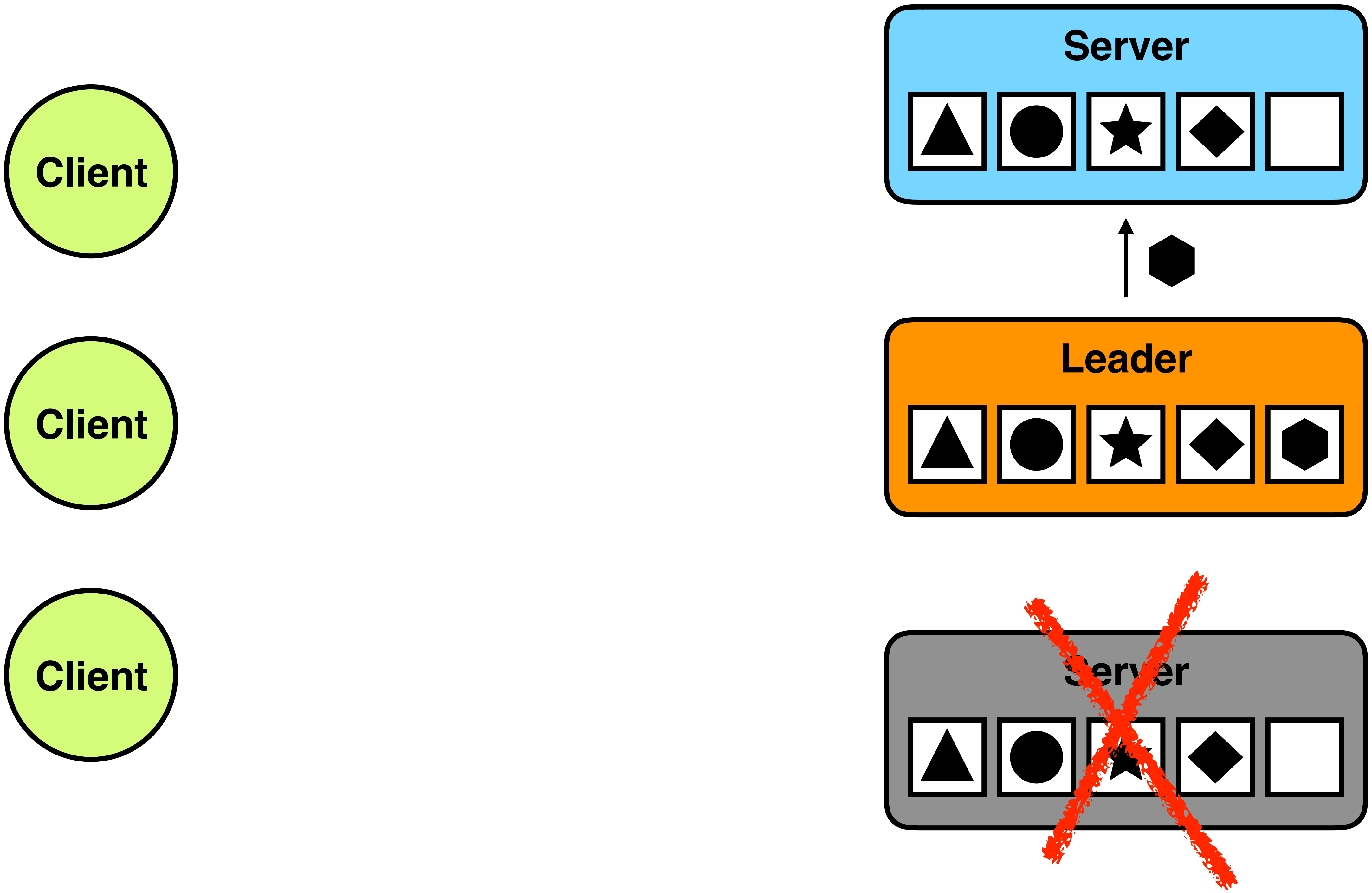
We refer to any group of nodes which is ‘enough’ as a **replication quorum.**

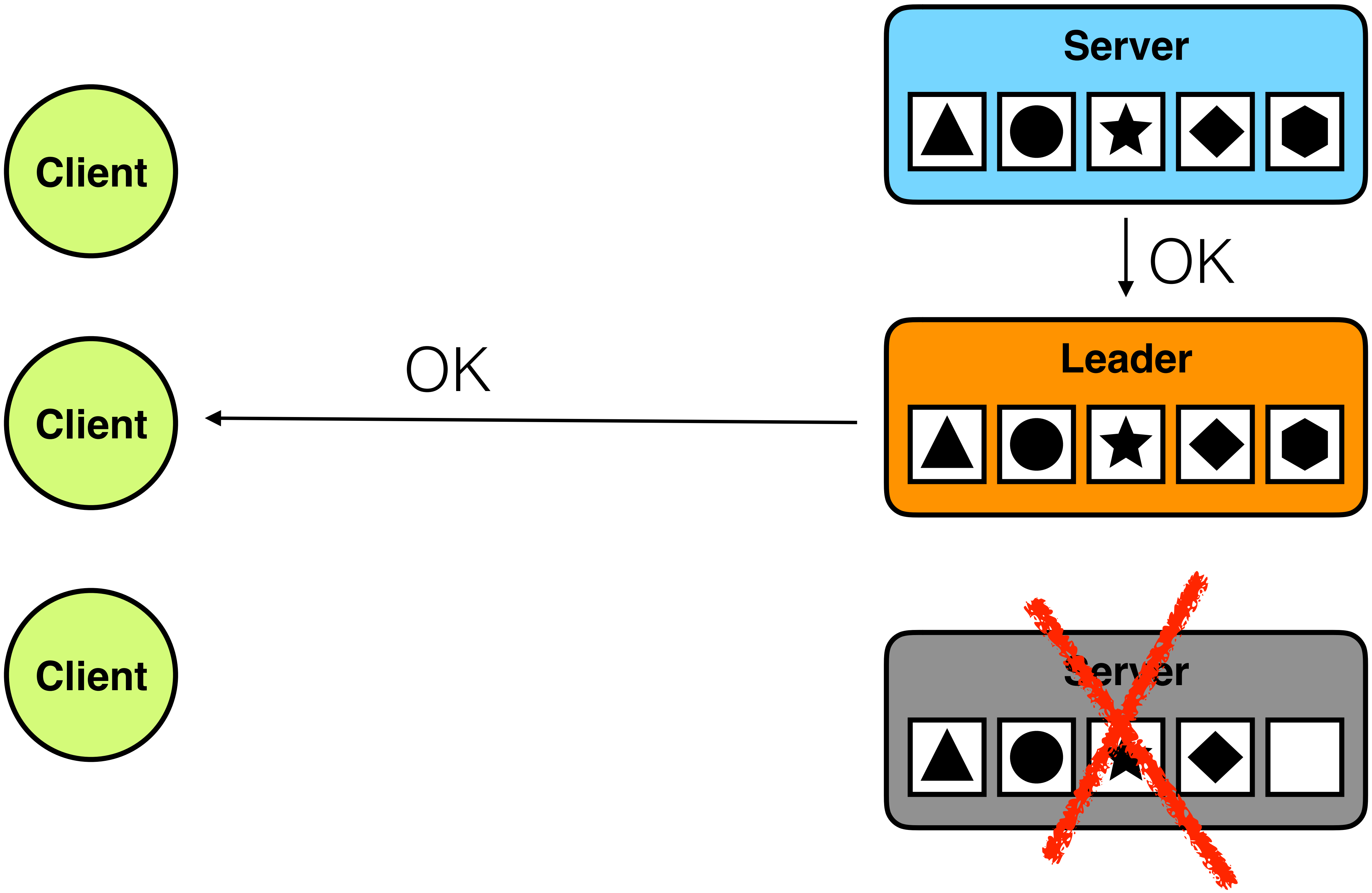
For now, we will use majorities.

Now what happens when a server fails?

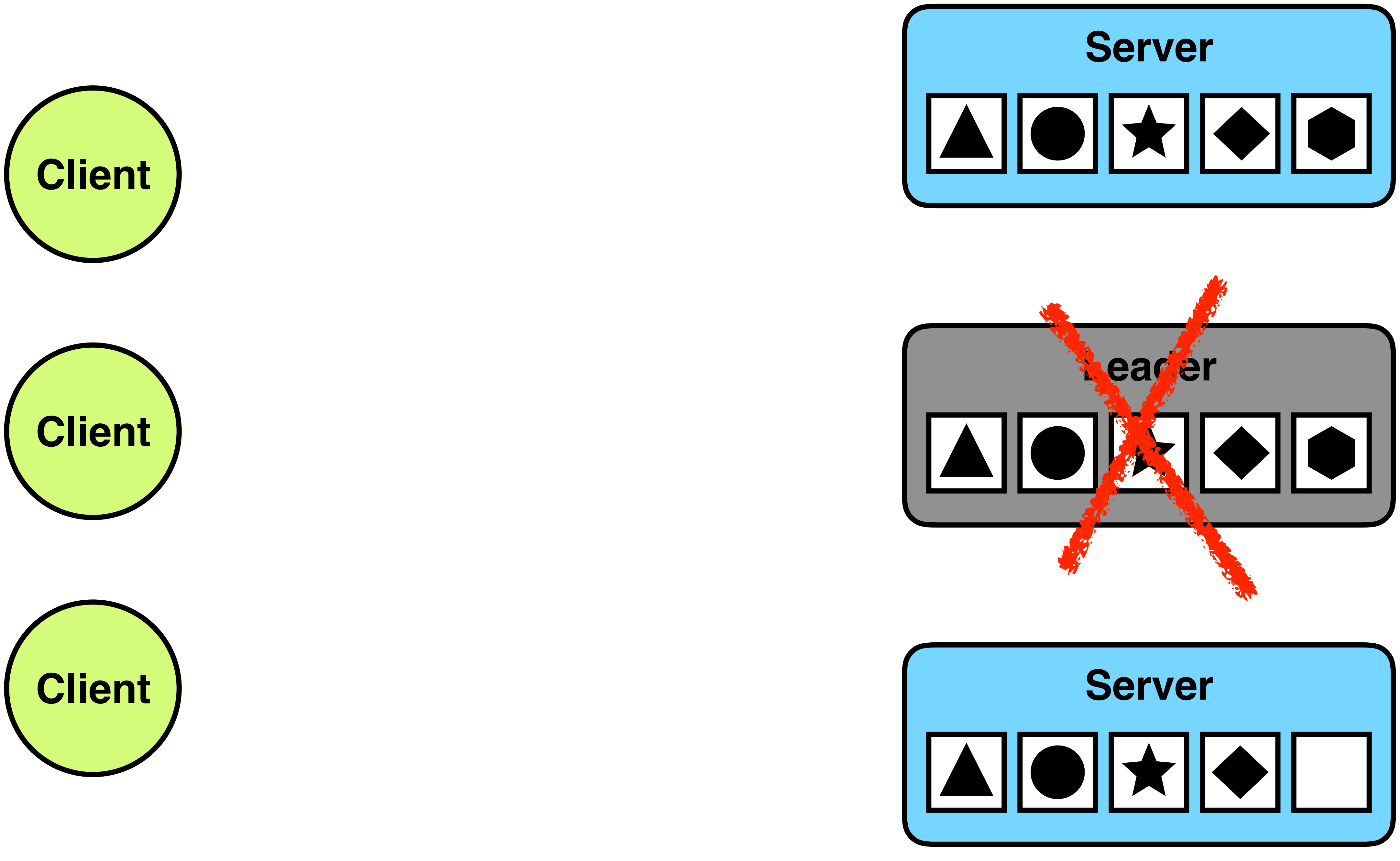

Client
Client
Client
Server
Leader
Server

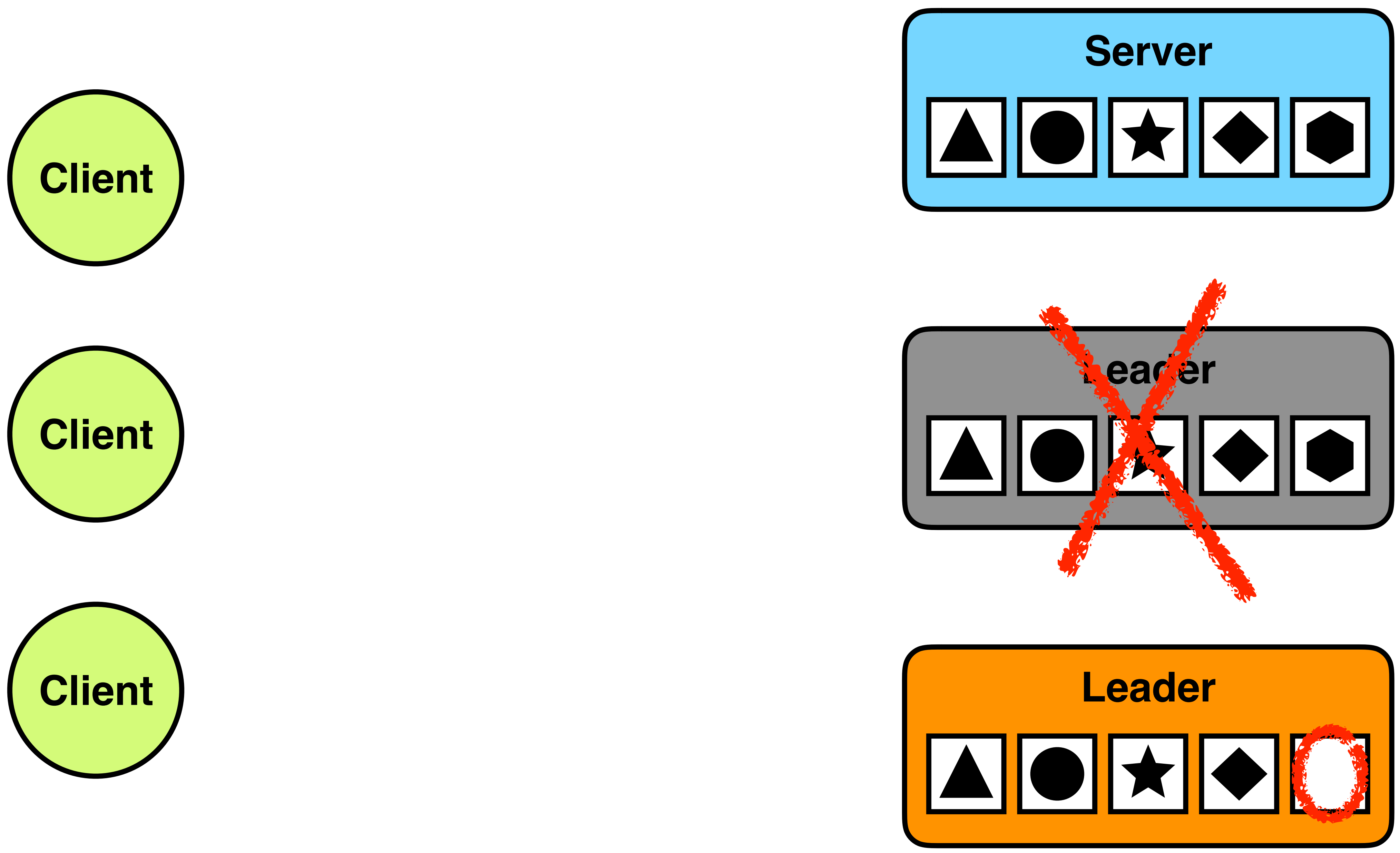

Client
Client
Client
Append ⬢
Server
Leader
Server


Client
Client
Client
Server
Leader
Server


Server
OK
Client
Client
OK
Leader
Client
Server

What happens when the leader fails?


Client
Client
Client
Server
Leader
Server


Client
Client
Client
Server
Leader
Leader

Recall that we never compromise safety

Thus the new leader has two jobs:

1. To stop the old leader. We cannot say for sure that they have failed.

2. To learn about which log entries have already been agreed and not overwrite them.

The order is important here!

How to stop past leaders?

They may come back to haunt us.

- Ask them to stop
- Use leases and wait for them to expire
- Ignore them by having nodes promise to stop replicating for them

How many promises are enough?

Leaders cannot wait forever.

We need a minimum of one node from each **replication quorum** to promise not to replicate new entries.

We refer to this as the **leader election quorum**.

For now, we will use majorities.

Promises are tricky to safely break

Let’s assume each leadership term has a unique term number.

Problem: We don’t always know who the past leaders are. We need a deterministic scheme for breaking promises.

Solution: Each node stores the promise as “ignore all smaller terms”.

Learn all committed entries

The new leader must **never overwrite** committed entries.

It must learn about the committed entries before it starts adding new entries without leaving holes.

Safely handle in-progress commands

Any in-progress entries already replicated on the leader election quorum nodes must be finished by the new leader.

If there are conflicting in-progress entries from different leaders, we choose the log entry with the highest view.

Context

This mechanism of views and promises is widely utilised. This core idea is commonly referred to as **Paxos**.

It's a recurring foundation in the literature:

Viewstamped Replication [PODC'88], Paxos [TOCS'98], Zab/Zookeeper [ATC'10] and Raft [ATC'14] and many more.

What does this give us?

Safety is always guaranteed.

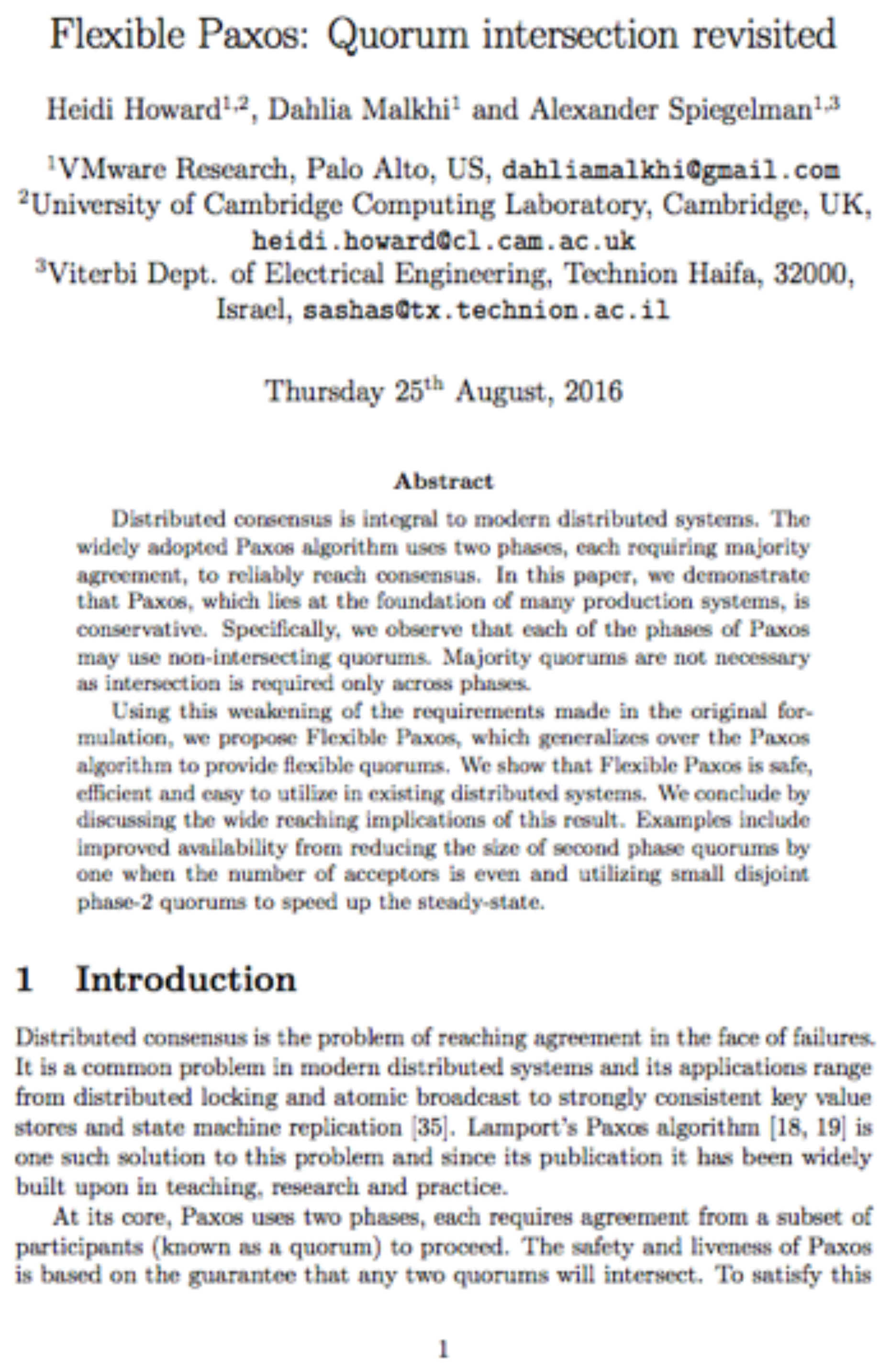
Progress is made provided that a quorum of members are up and communicating*.

*oversimplification

Flexible Paxos

Introducing the revised quorum intersection requirement for Paxos.

What's changed?



Flexible Paxos: Quorum intersection revisited

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

[1]VMware Research, Palo Alto, US, dahliamalkhi@gmail.com
[2]University of Cambridge Computing Laboratory, Cambridge, UK, heidi.howard@cl.cam.ac.uk
[3]Viterbi Dept. of Electrical Engineering, Technion Haifa, 32000, Israel, sashas@tx.technion.ac.il

Thursday 25th August, 2016

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 conclude by discussing the wide reaching implications of this result. Examples include improved availability from reducing the size of second phase quorums by one when the number of acceptors is even and utilizing small disjoint phase-2 quorums to speed up the steady-state.

1 Introduction

Distributed consensus is the problem of reaching agreement in the face of failures. It is a common problem in modern distributed systems and its applications range from distributed locking and atomic broadcast to strongly consistent key value stores and state machine replication [35]. Lamport's Paxos algorithm [18, 19] is one such solution to this problem and since its publication it has been widely built upon in teaching, research and practice.

At its core, Paxos uses two phases, each requires agreement from a subset of participants (known as a quorum) to proceed. The safety and liveness of Paxos is based on the guarantee that any two quorums will intersect. To satisfy this

1

[OPODIS'16]

Flexible Paxos

Traditionally, we have assumed that all quorums needed to intersect, regardless of whether the quorum was used for replication or leader election.

In practice, the only requirement is that leader election quorums intersect with replication quorums.

Implications

In theory:

- Helps us to understand Paxos
- Generalisation & weakening of the requirements
- Orthogonal to any particular algorithm built upon Paxos*

In practice:

- Many quorum schemes now become feasible

Quorum Systems

An overview of various alternatives to majorities.

Quorum Systems

Until now, majorities have been the only practical quorum system for consensus.

Great opportunity to borrow from commit protocols, databases & data replication.

What does this give us?

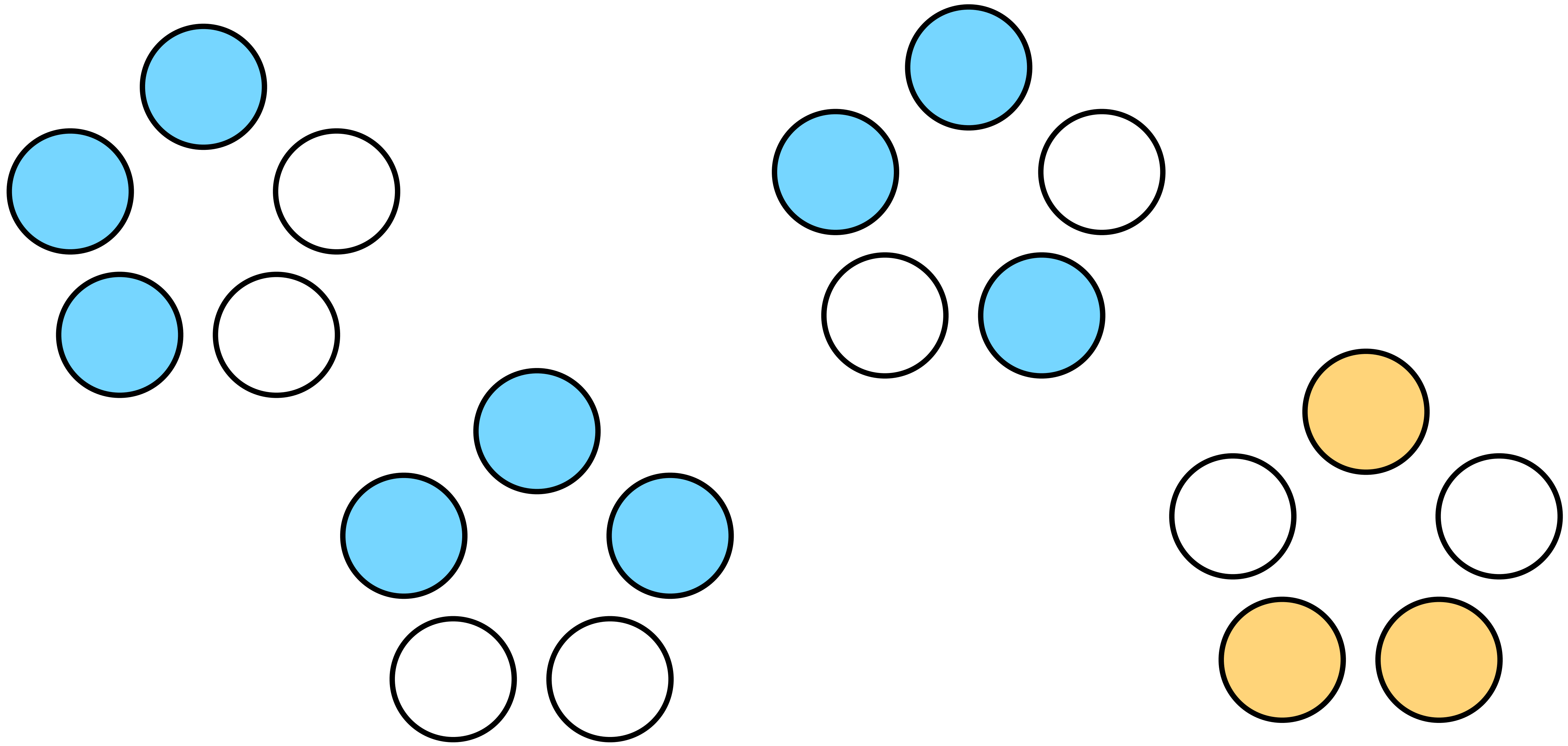
Safety is always guaranteed.

Progress is made provided that either:

1. leader is up and a replication quorum of members are up and communicating *(replication phase).*

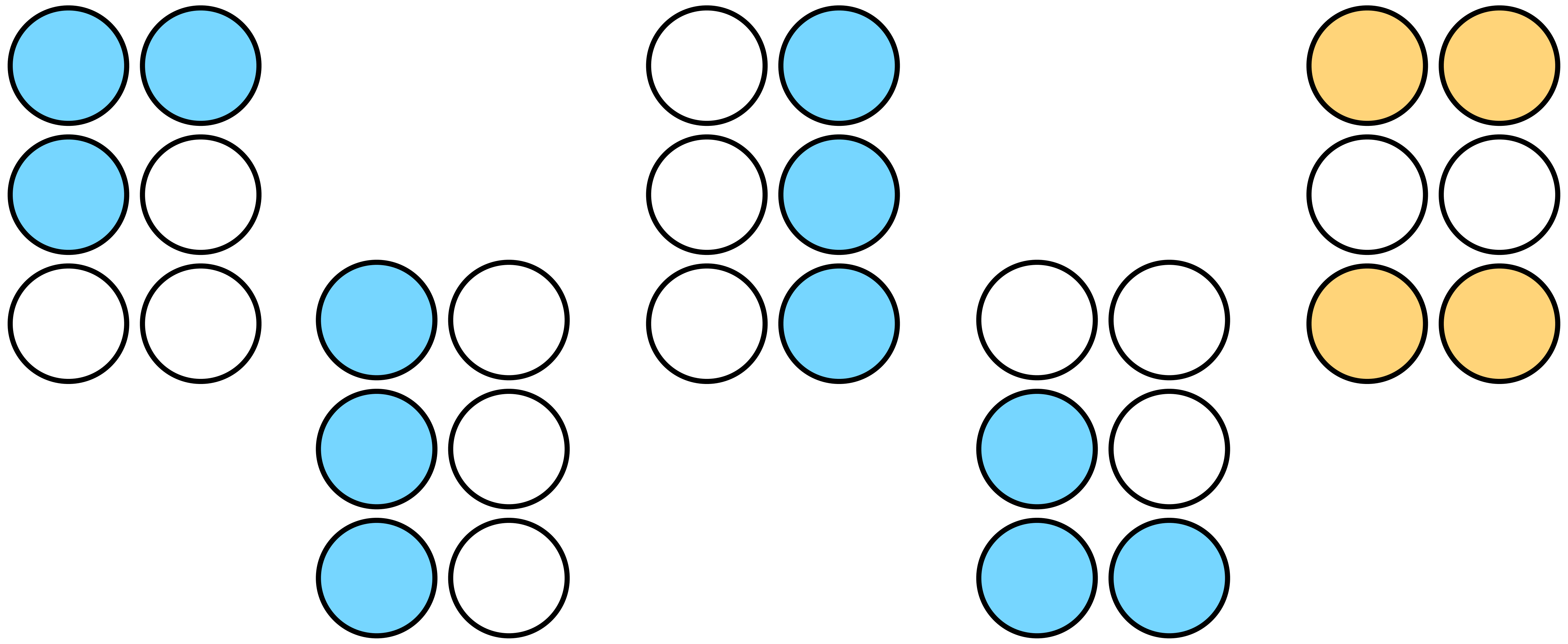
2. leader is down and a leader election quorum of members are up and communicating *(leader election phase).*

Strict Majorities



🔵 Replication quorum 🟡 Leader election quorum

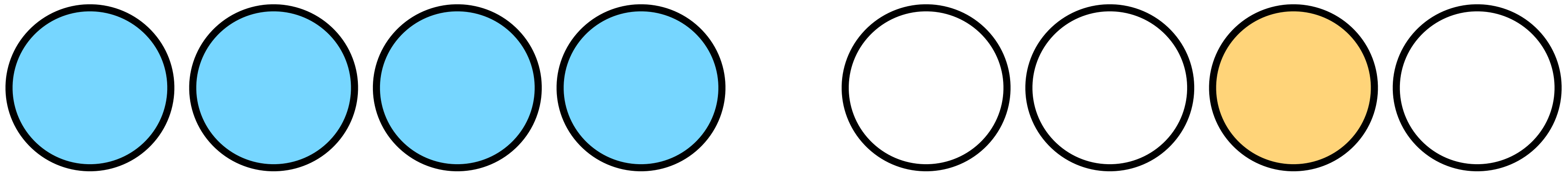
Non-strict Majorities



🟦 Replication quorum

🟨 Leader election quorum

All-In



🔵 Replication quorum

🟡 Leader election quorum

Counting quorums

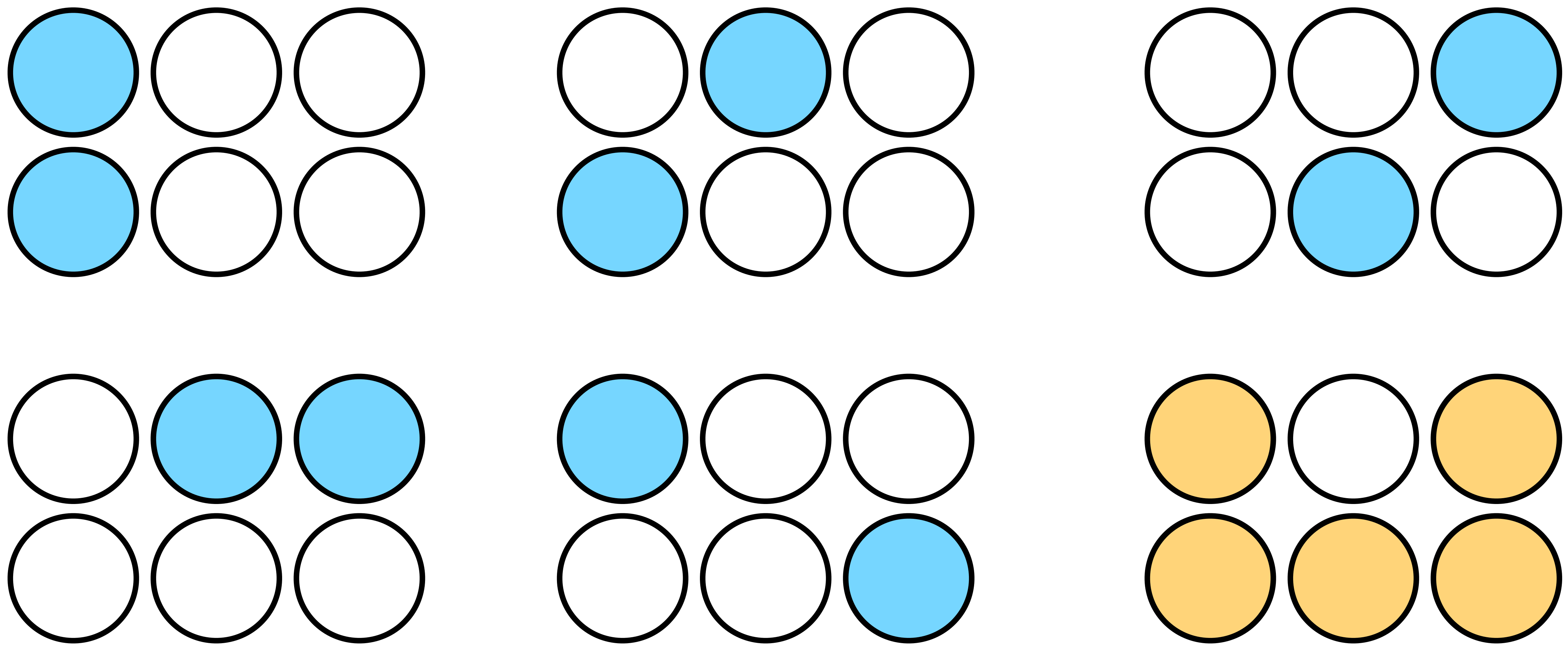
RQ + LQ = N + 1

RQ: Replication quorum size

LQ: Leader election quorum size

N: number of nodes

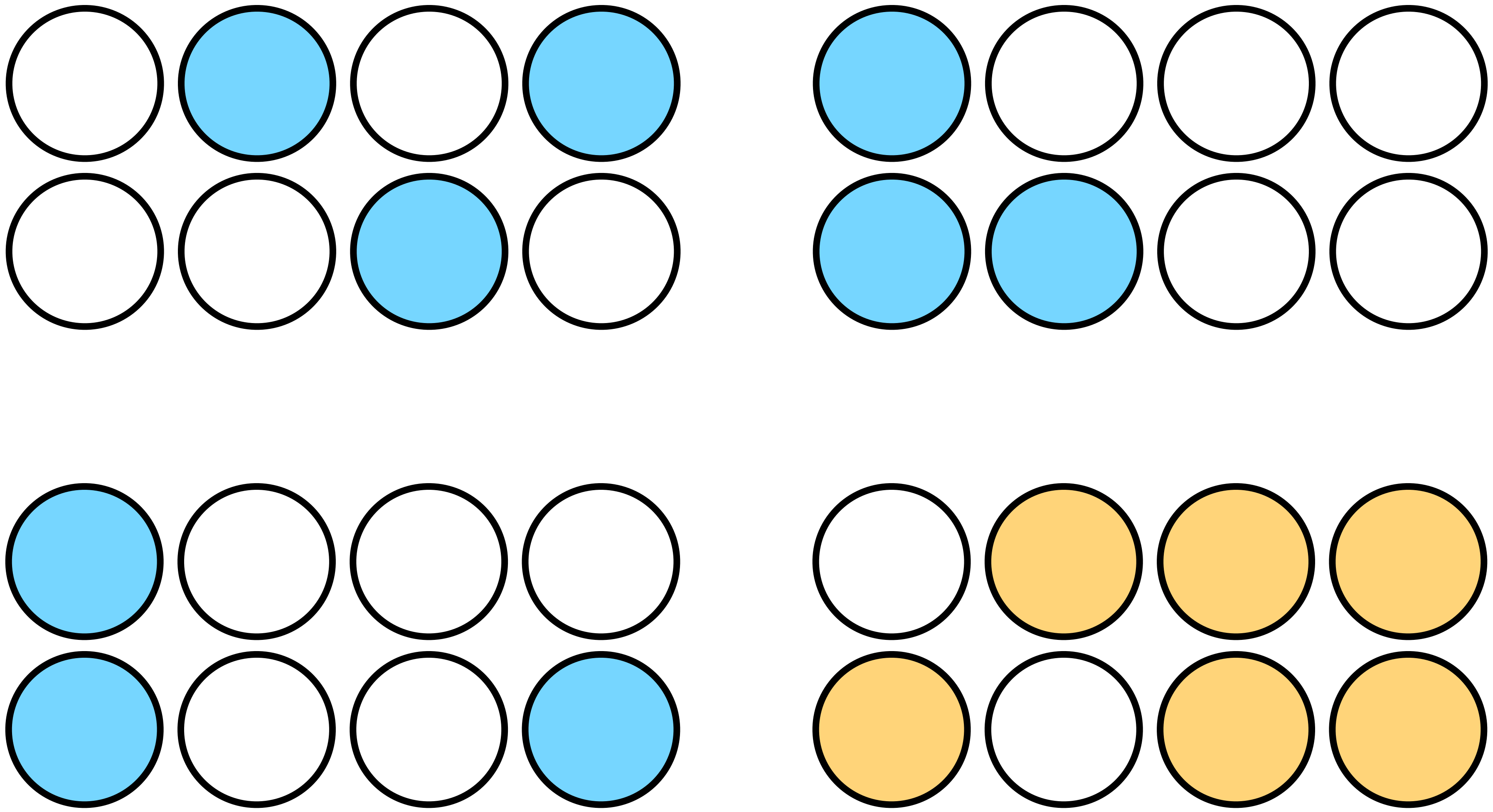
Replication quorum = 2/6

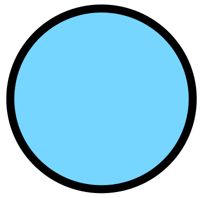


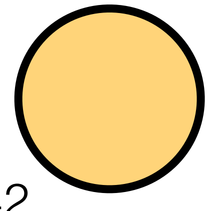
🔵 Replication quorum

🟡 Leader election quorum

Replication quorum = 3/8



 Replication quorum

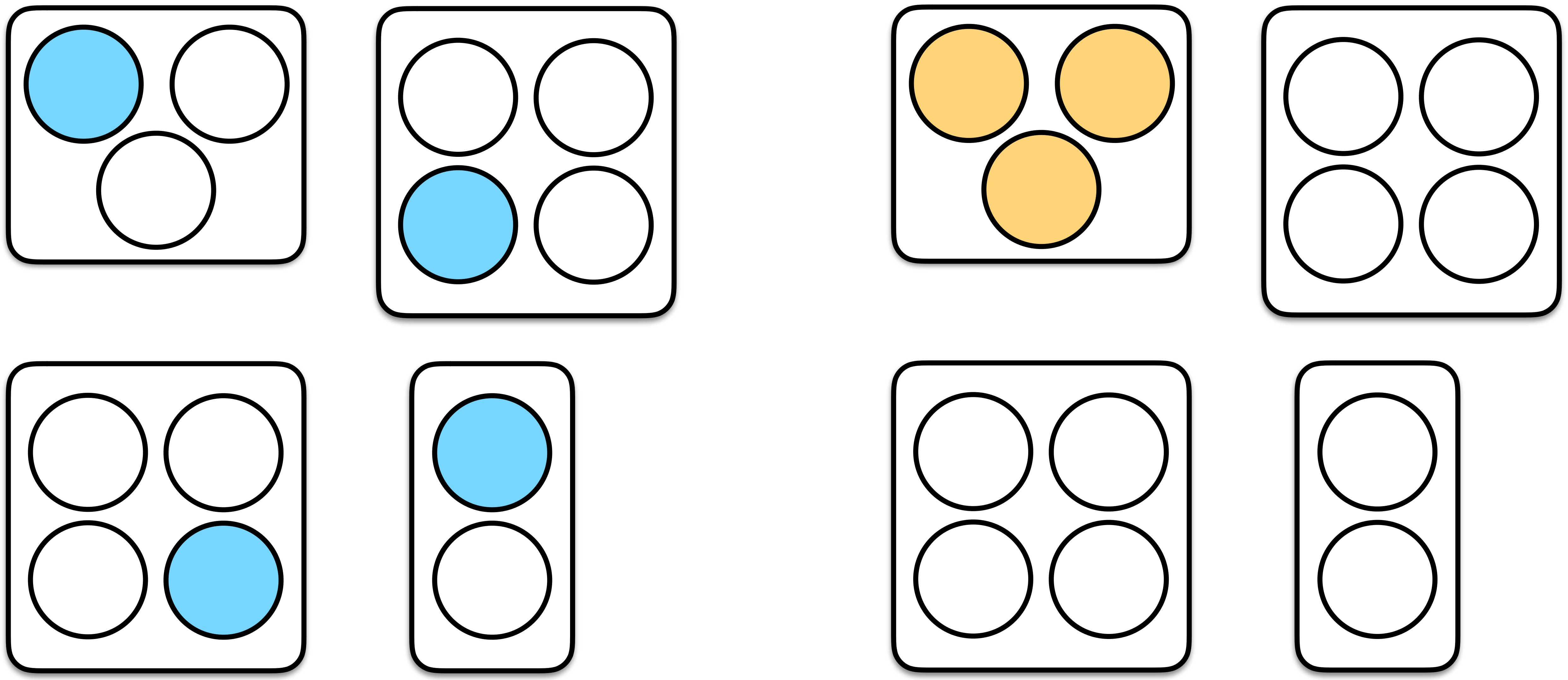
 Leader election quorum

Not all members were created equal

- Machines and the networks which connect them are heterogeneous

- Failures are not independent: members are more likely to simultaneously fail if they are located in the same host, rack or even datacenter

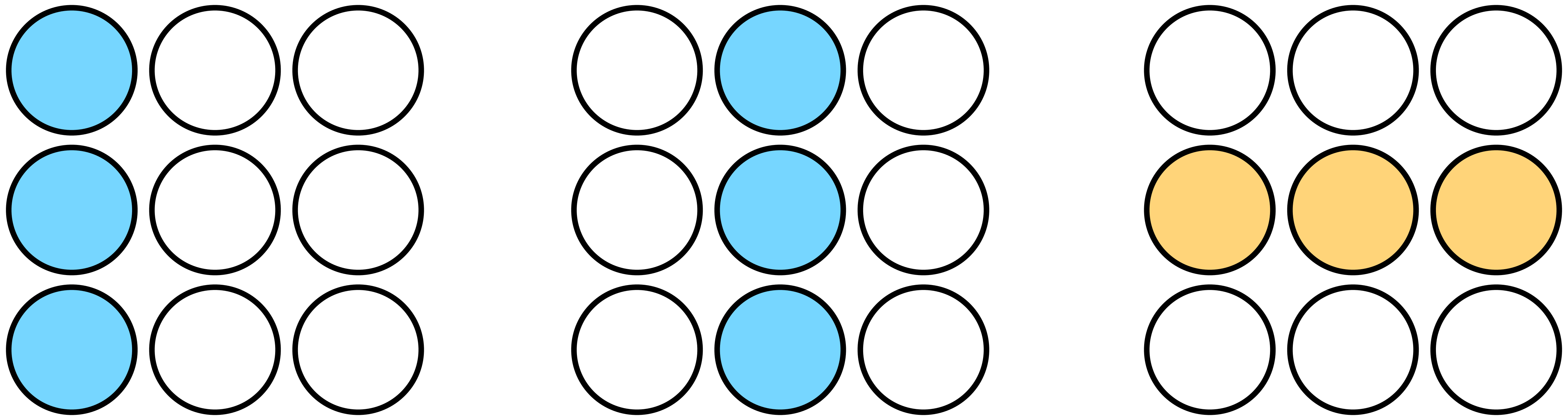
Groups



🔵 Replication quorum

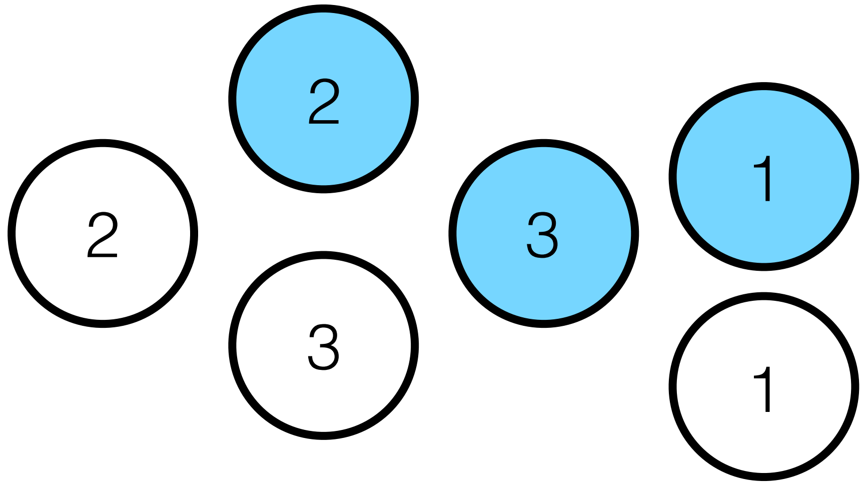
🟡 Leader election quorum

Grids

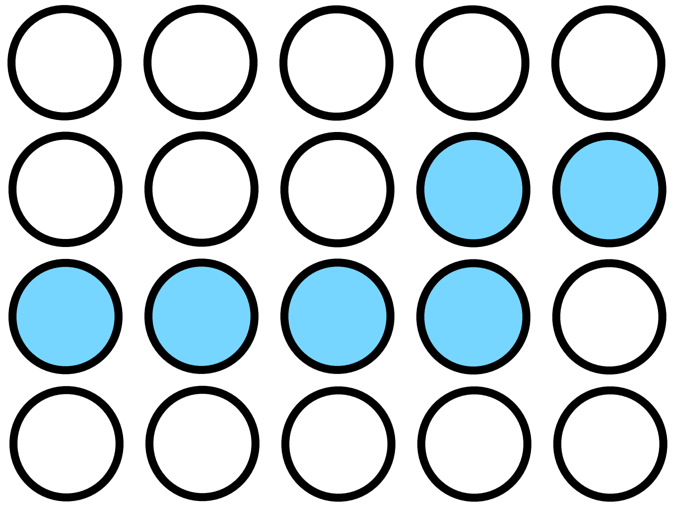


🟦 Replication quorum 🟨 Leader election quorum

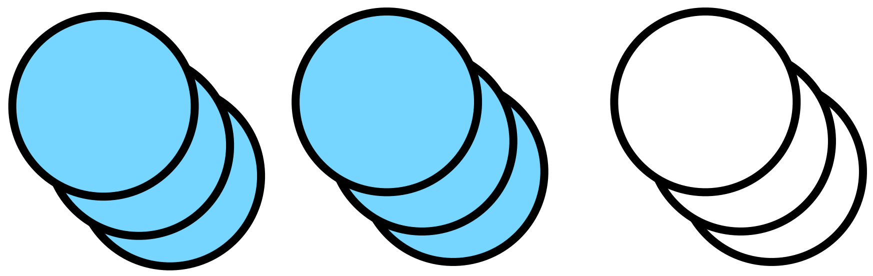
Any many more…



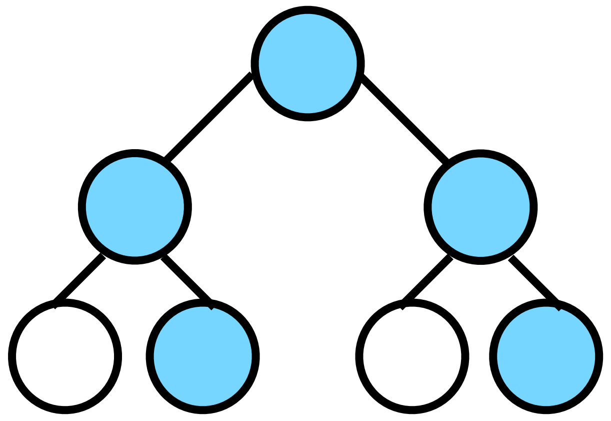
Weighed Voting



Grid Paths



Combined Schemes



Hierarchy

TLA+

Extending the TLA+ spec for Paxos to use Flexible Paxos and checking it*.

*not an expert

TLA+ Features

TLA+ key features:

1. A syntax for **specifying** the behavior of systems
2. **Model checking** for system specifications
3. Mechanical checking of **TLA+ proofs**

Aims

1. To modify the original TLA+ spec to weaken the quorum intersection requirements according to Flexible Paxos

2. To model check the revised specs with alternative quorums systems.

Flexible Paxos Specification

DEMO

What is the point of it all?

We have verified properties on a elegant formal specification of a distributed system.

When we implement these systems in languages such as Java, C++ & Go.

Could we bridge this divide?

Summary

- The only quorum intersection requirement is that leader election quorums must intersect with the replication quorums.
- Majorities are not the only option for safely reaching consensus and may not be best suited to practical systems.
- Don’t give up on strong consistency guarantees. Distributed consensus can be performant and scalable.
- TLA+ is a useful tool for reasoning about systems but it has limitations.

Questions?

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