

# Decentralizing Distributed Consensus with Faster Paxos

[Distributed consensus revised]

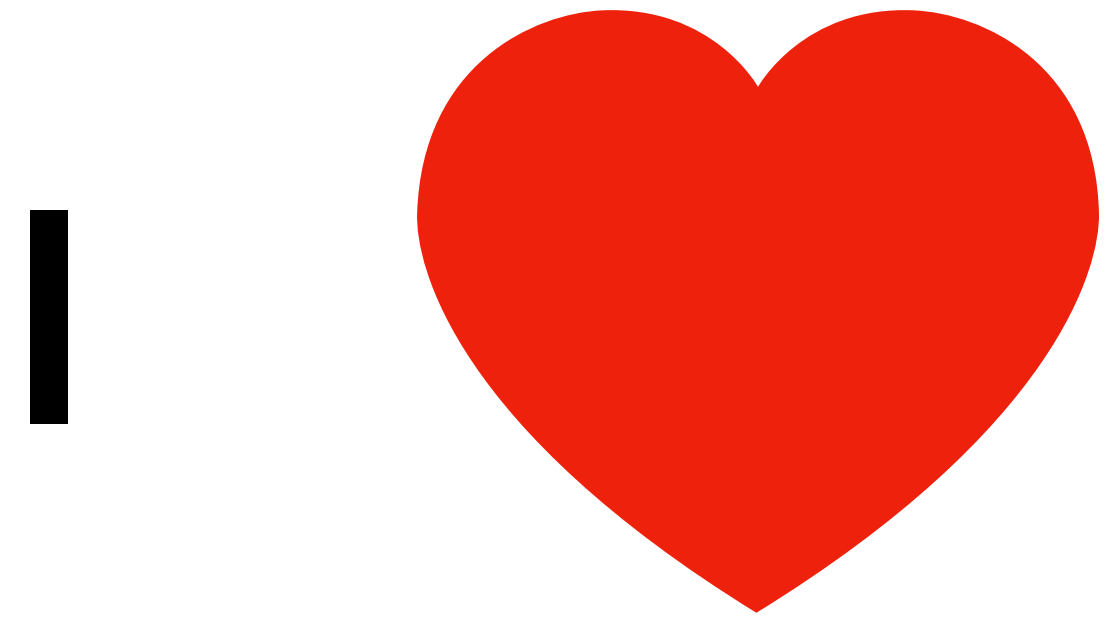
Heidi Howard @ Cambridge University  
[heidi.howard@cl.cam.ac.uk](mailto:heidi.howard@cl.cam.ac.uk)

# TL;DR

Linearizability is not as expensive as you might think.

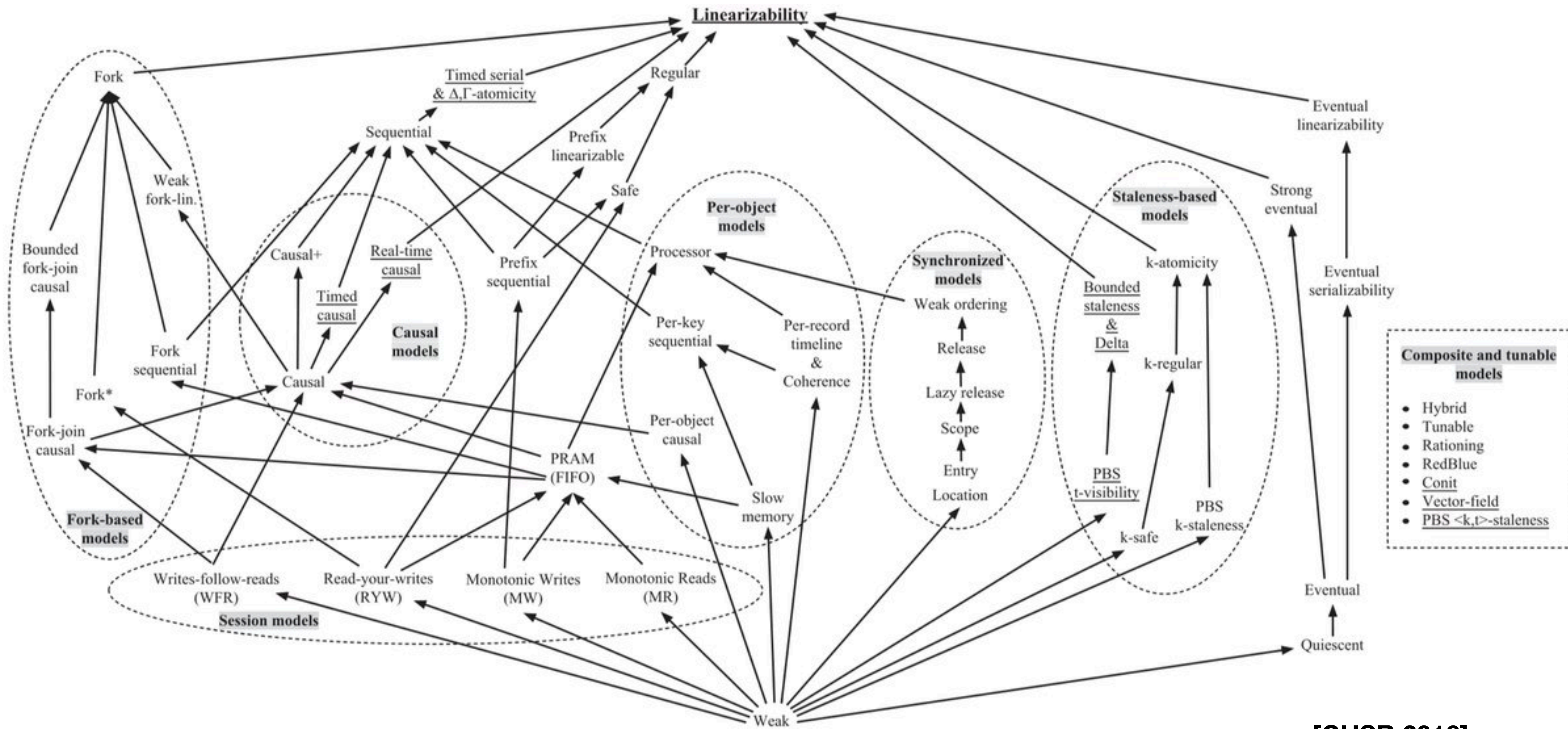
Paxos and friends offer a one-size-fits-all solution to distributed consensus which may not perform best for every application.

Faster Paxos demonstrates that decentralised consensus algorithms can perform better than we previously believed.



I

**Linearizability**



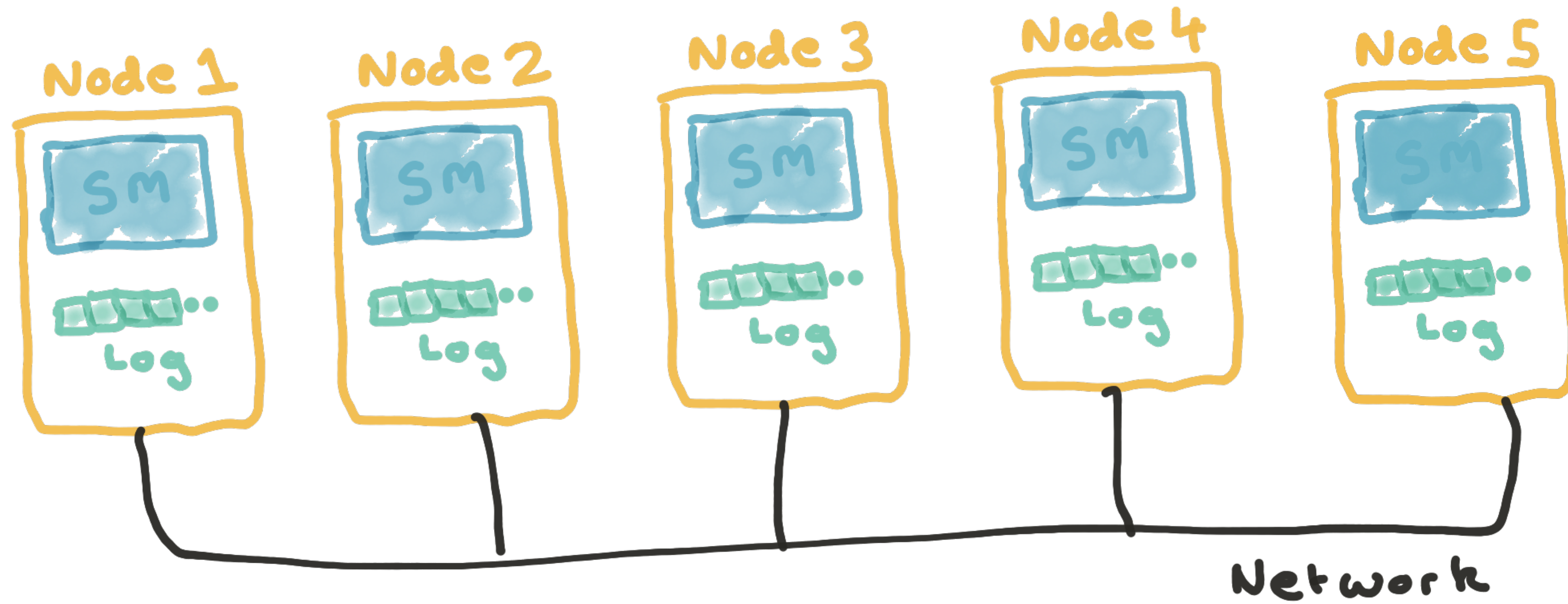
[CUSR 2016]

**I**  **Linearizability**

**But I don't trust** 

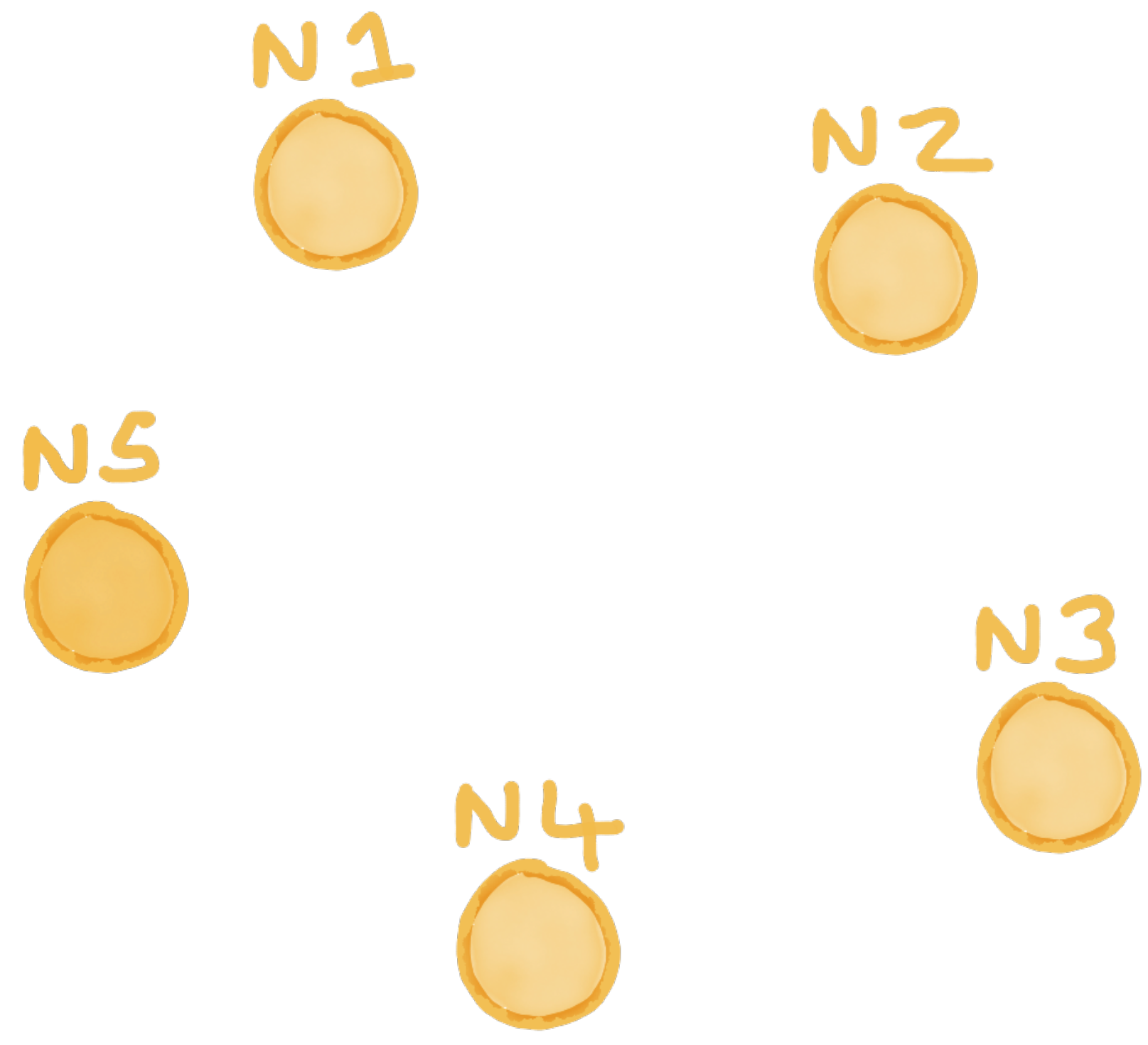


# State Machine Replication



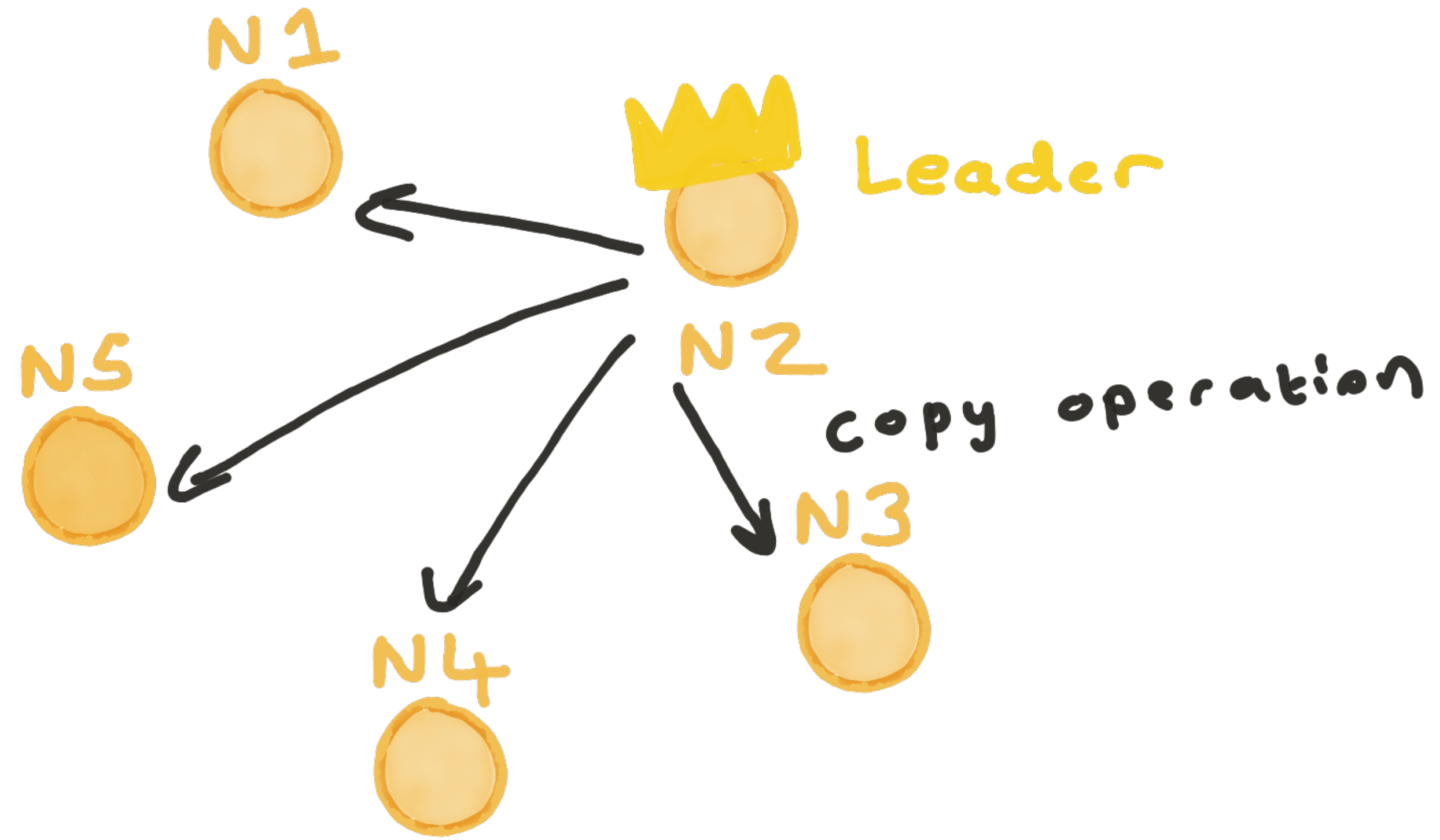
# Multi-Paxos

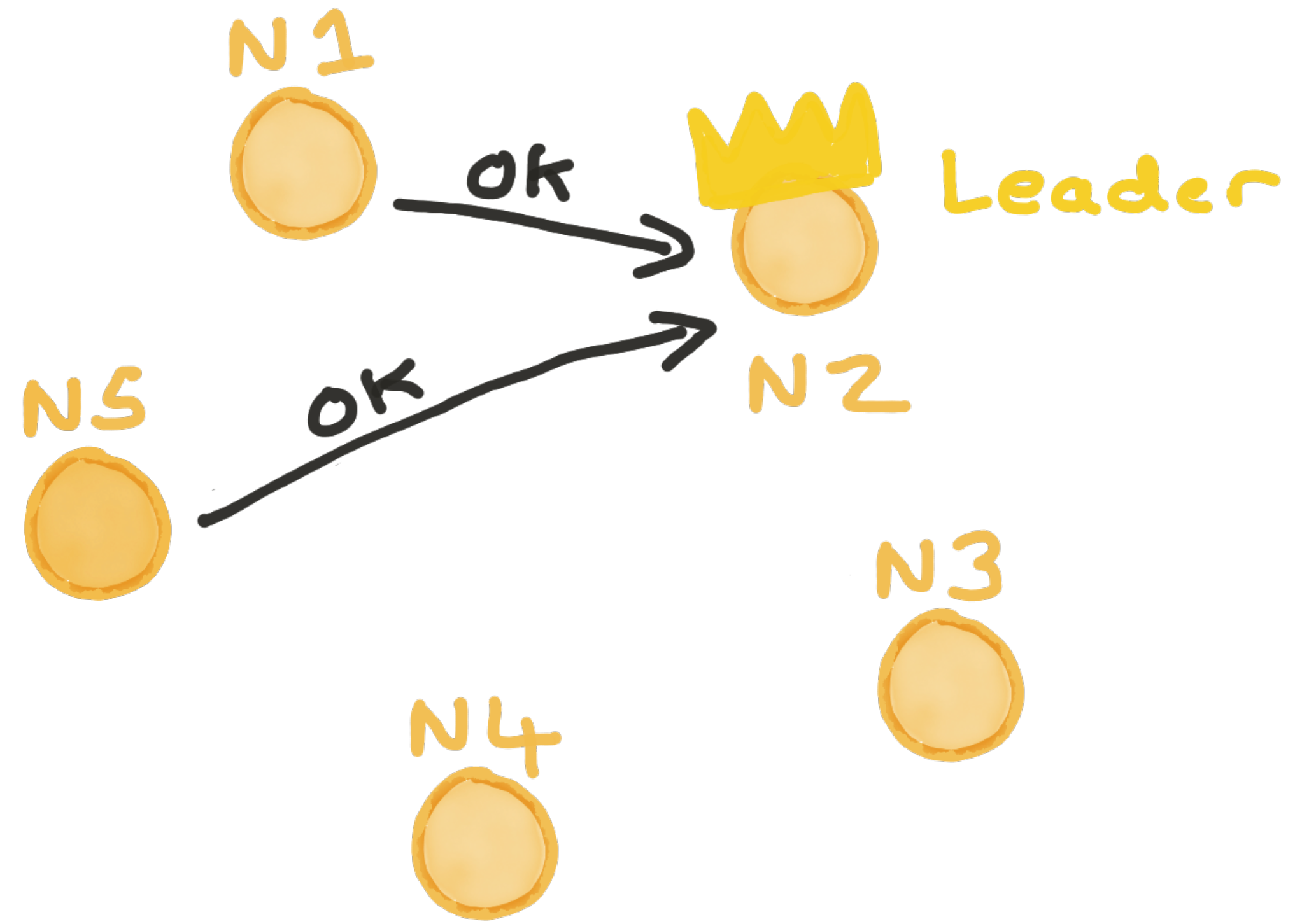
1. Select a node to be leader.
2. Nodes send operations to leader, the leader orders them and replicates them to the other nodes.
3. Once an operation has been replicated to a majority of nodes then it can be applied to the state machines.
4. If the leader fails, it is replaced by another node. This process requires agreement from a majority of nodes.

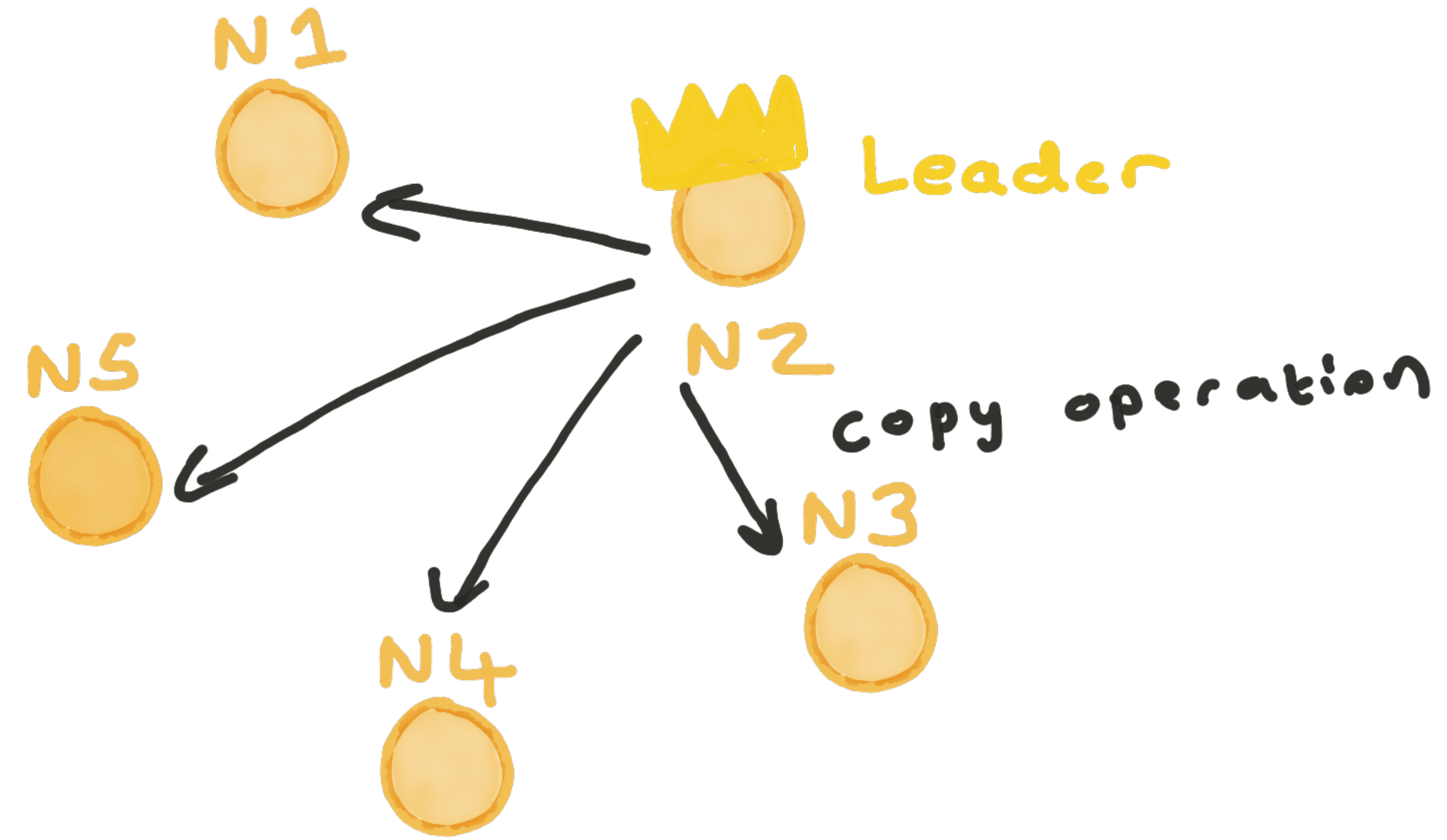


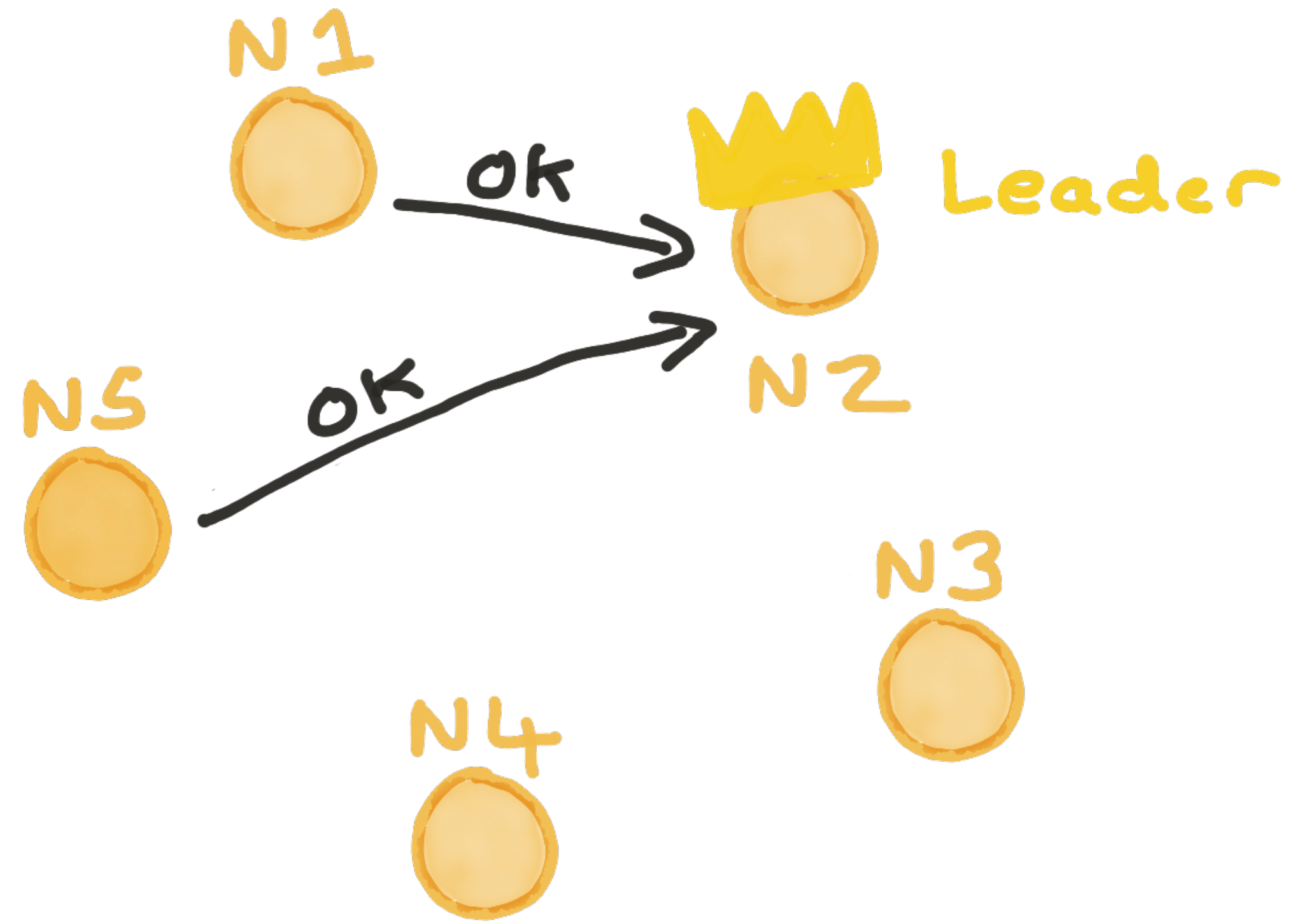


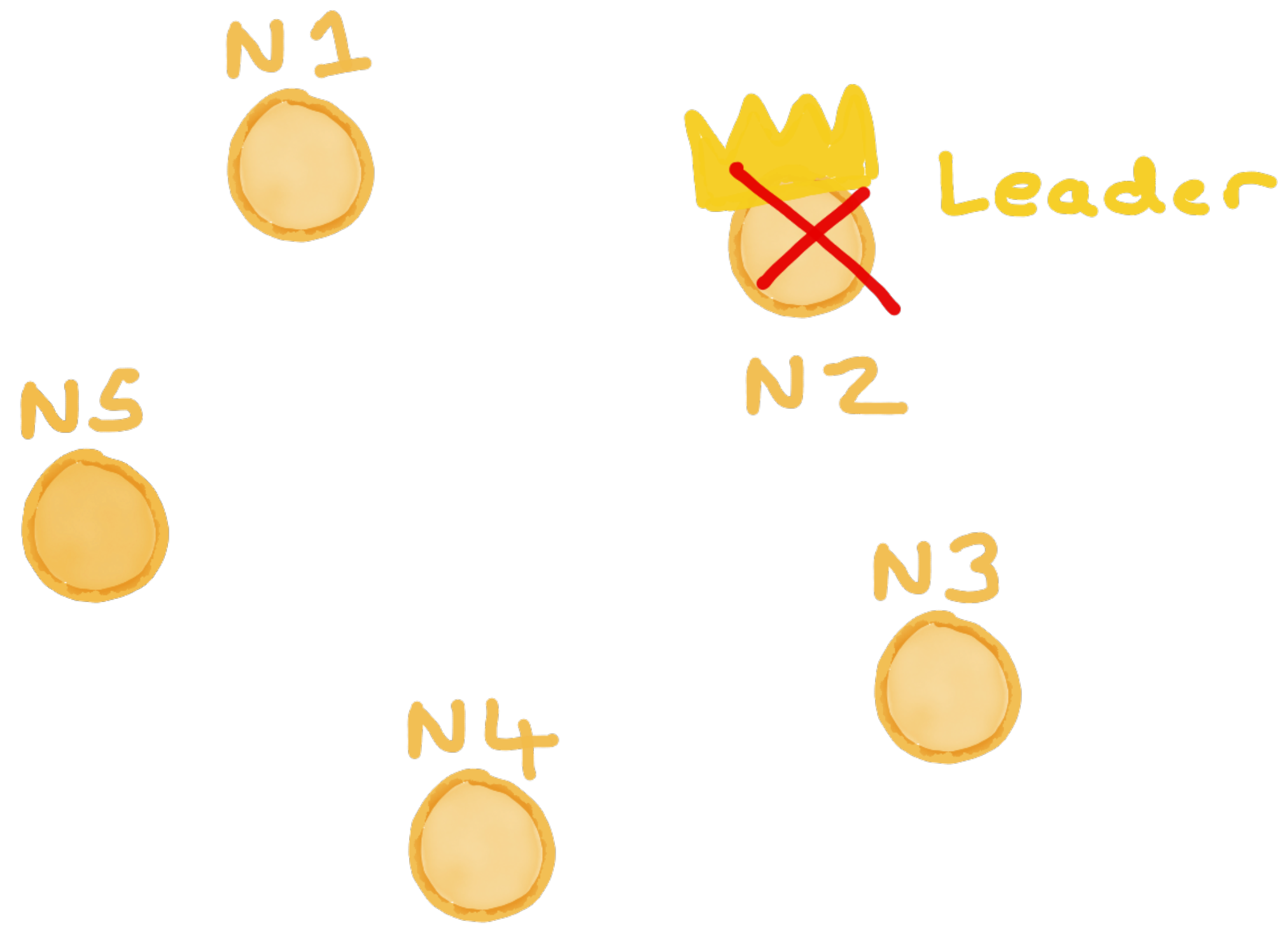




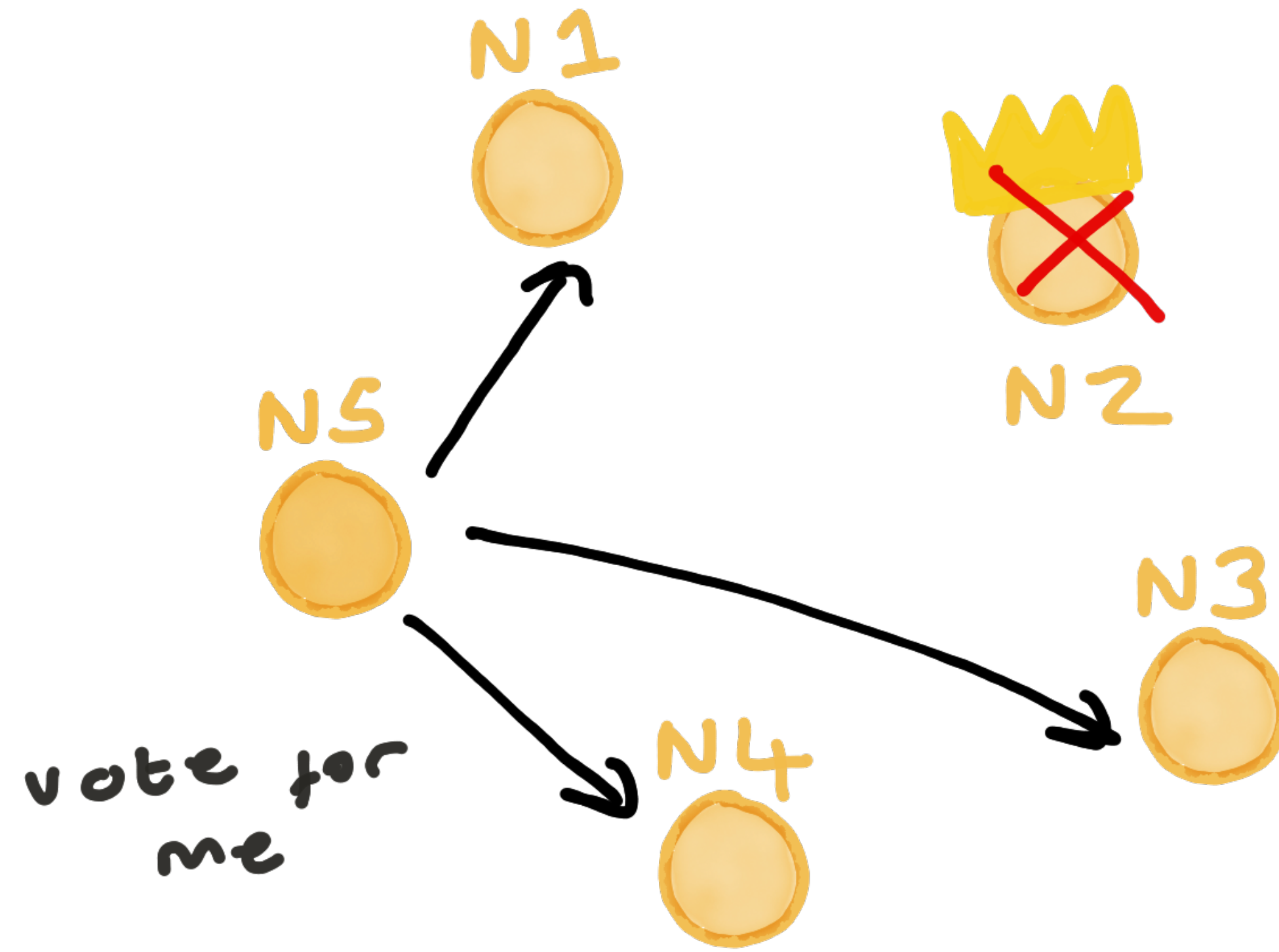


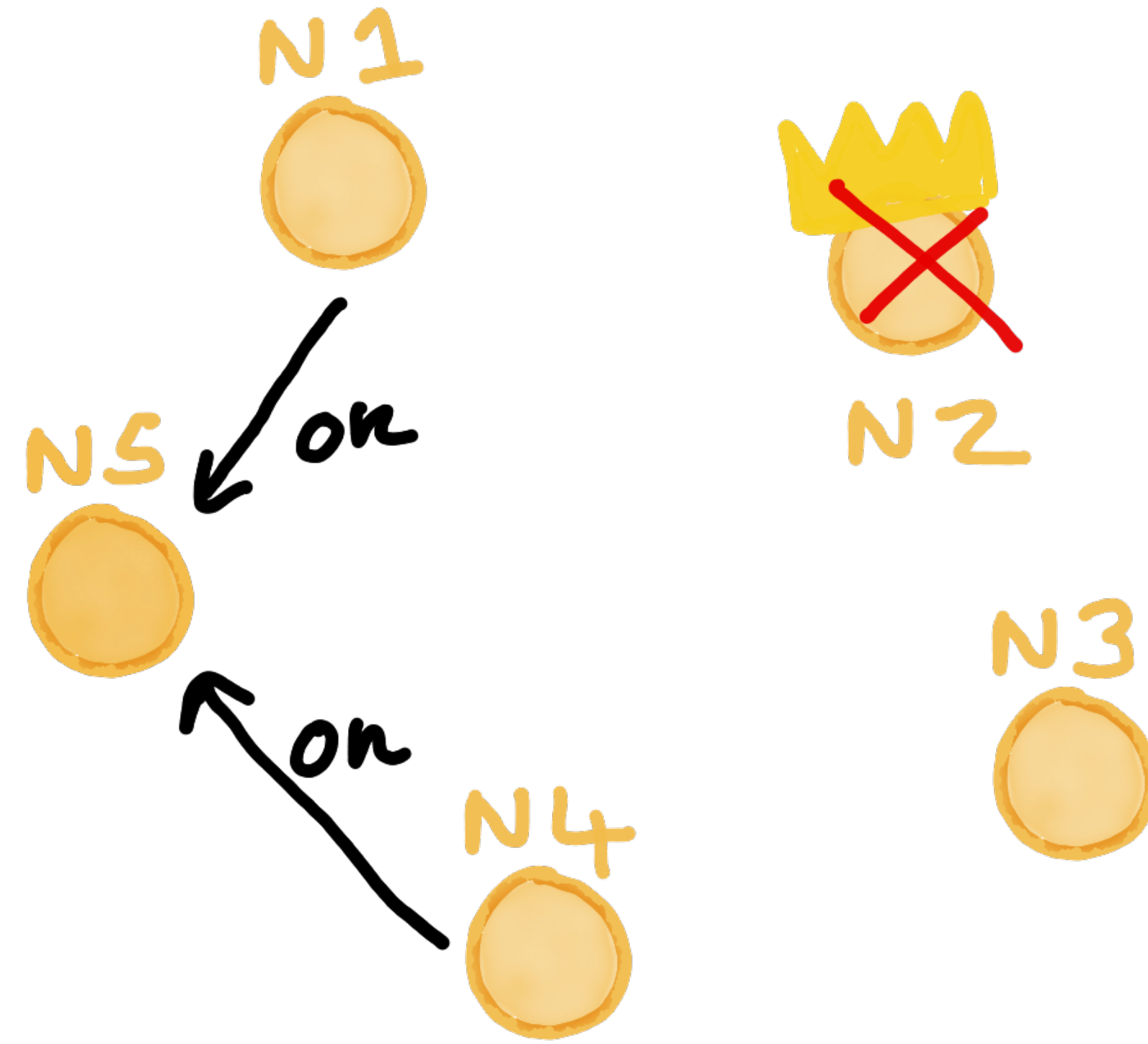


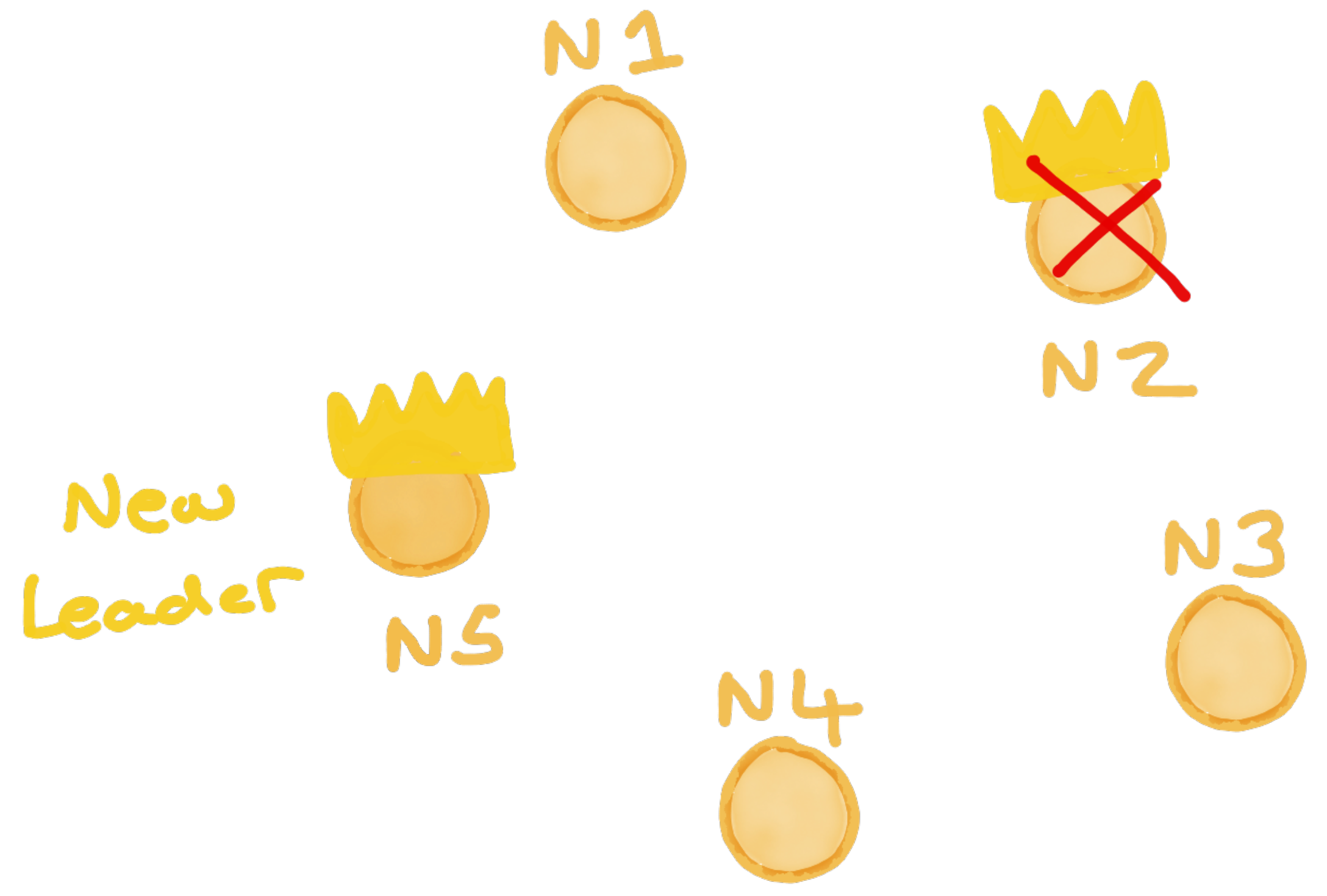




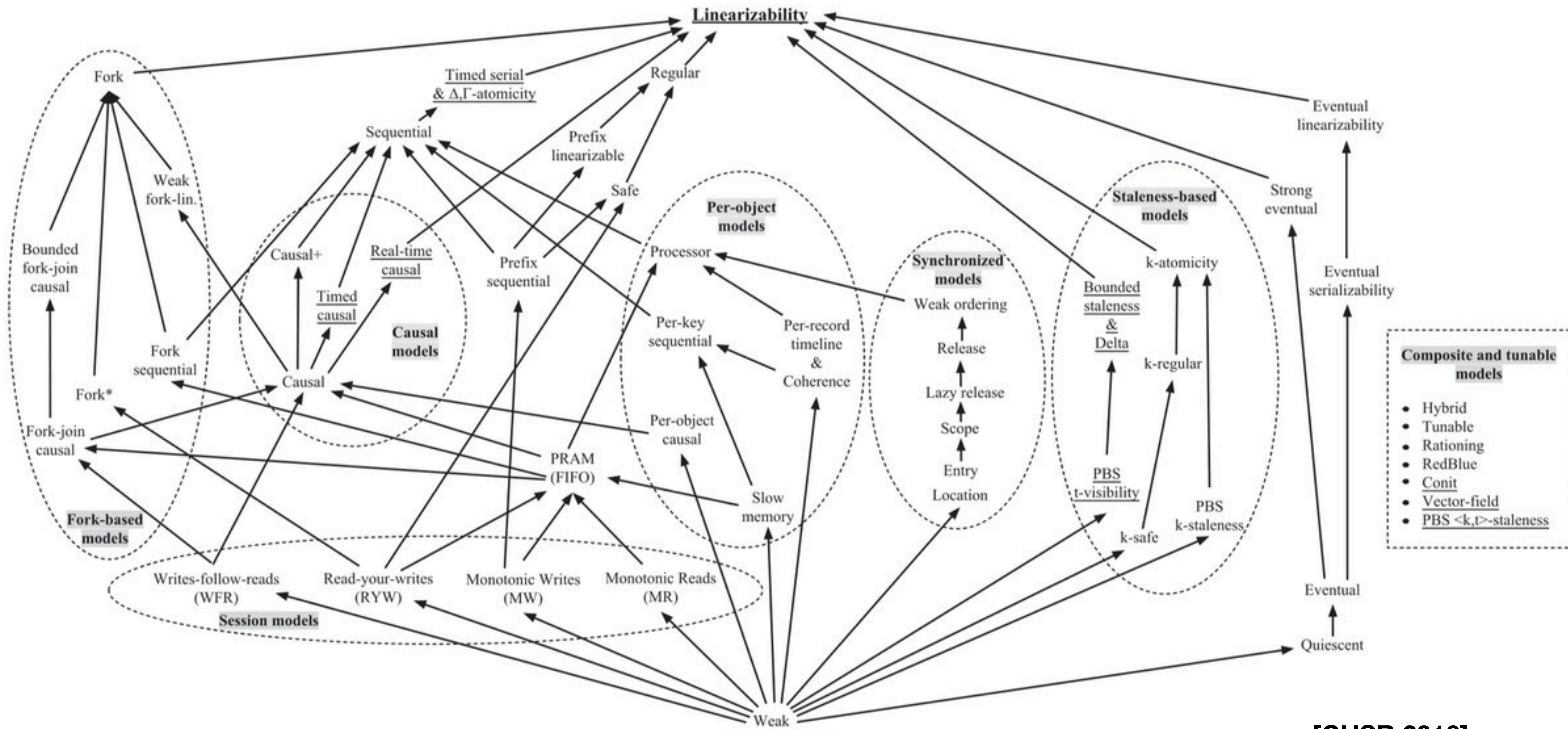












- Composite and tunable models**
- Hybrid
  - Tunable
  - Rationing
  - RedBlue
  - Conit
  - Vector-field
  - $\text{PBS} \langle k, t \rangle$ -staleness

[CUSR 2016]

# Distributed yet highly centralised

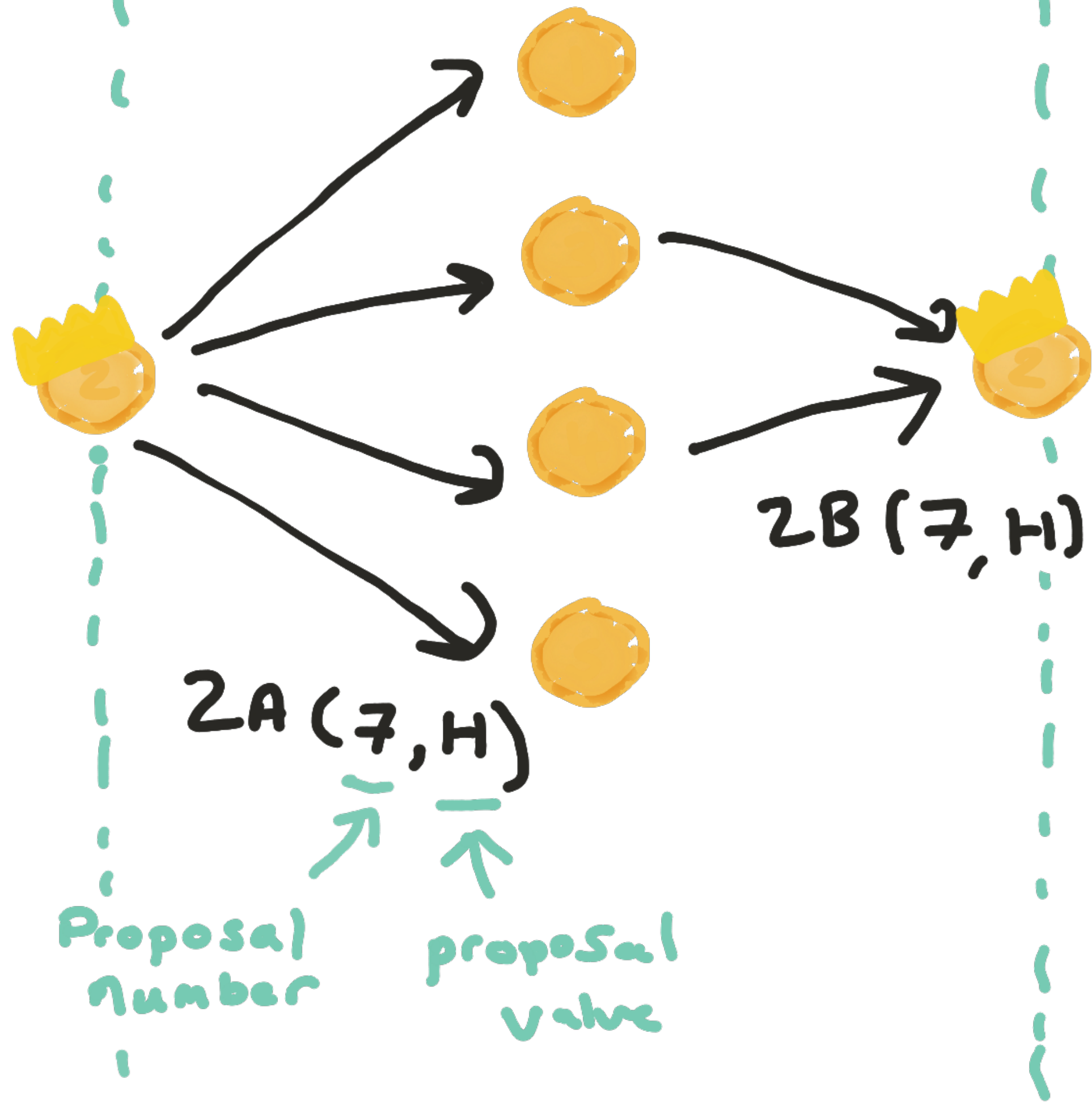
The leader is a single point of serialisation.

However, it also:

- Limits throughput as it becomes a bottleneck
- Increases latency (3 steps instead of 2)

Alternatives have been proposed but have seen little adoption.

# Paxos - Phase 2





# Fast Paxos

Fast Paxos is similar to Paxos, except that some proposal numbers are *fast*.

These use quorums of  $3/4$  of nodes instead of  $1/2$  of nodes.

However, any node can send a value directly to the other nodes, bypassing the leader.

Distrib. Comput. (2006) 19:79–103  
DOI 10.1007/s00446-006-0005-x

ORIGINAL ARTICLE

## Fast Paxos

Leslie Lamport

Received: 20 August 2005 / Accepted: 5 April 2006 / Published online: 8 July 2006  
© Springer-Verlag 2006

**Abstract** As used in practice, traditional consensus algorithms require three message delays before any process can learn the chosen value. Fast Paxos is an extension of the classic Paxos algorithm that allows the value to be learned in two message delays. How and why the algorithm works are explained informally, and a TLA<sup>+</sup> specification of the algorithm appears as an appendix.

**Keywords** Consensus · Fault tolerance · Distributed algorithms · Paxos

### 1 Introduction

The consensus problem requires a set of processes to choose a single value. This paper considers the consensus problem in an asynchronous message-passing system subject to non-Byzantine faults. A solution to this problem must never allow two different values to be chosen despite any number of failures, and it must eventually choose a value if enough processes are nonfaulty and can communicate with one another.

In the traditional statement of the consensus problem, each process proposes a value and the chosen value must be one of those proposed values. It is not hard to see that any solution requires at least two message delays before any process learns what value has been chosen [3]. A number of algorithms achieve this delay in the best case. The classic Paxos algorithm [7,9] is popular because it

achieves the optimal delay in the normal case when used in practical systems [12].

The apparently optimal number of message delays required by traditional consensus algorithms is illusory – an artifact of the traditional problem statement in which values are chosen by the same processes that propose them. In many applications, values are not proposed by the same processes that choose the value. For example, in a client/server system, the clients propose the next command to be executed and the servers choose one proposed command. When a traditional consensus algorithm is used in such a system, three message delays are required between when a client proposes a command and when some process learns which command has been chosen.

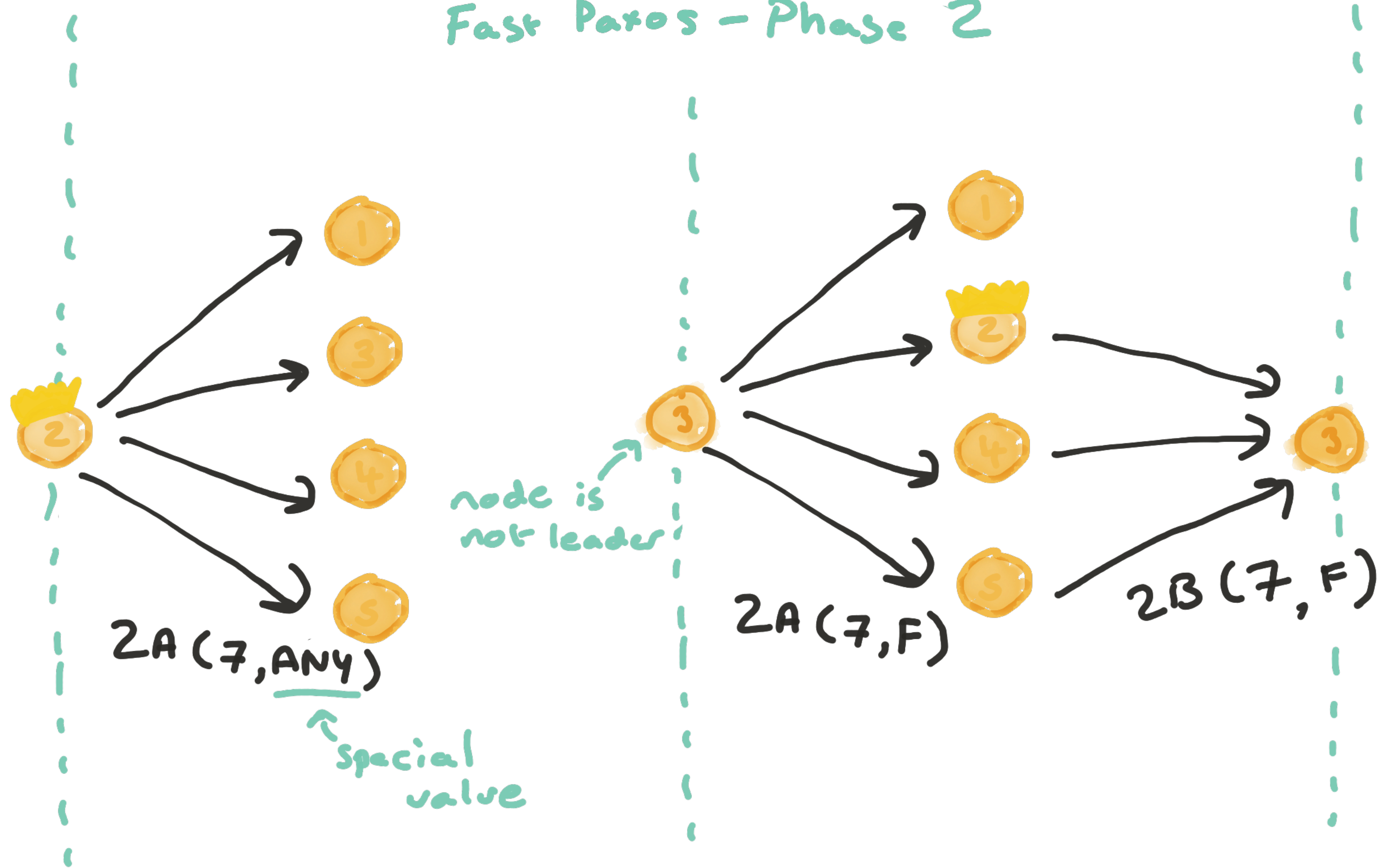
A fast consensus algorithm is one in which a process can learn the chosen value within two message delays of when it is proposed, even if values are proposed and chosen by different sets of processes. It has been shown that no general consensus algorithm can guarantee learning within two message delays if competing proposals collide – that is, if two different values are proposed concurrently [11]. A fast consensus algorithm therefore cannot always be fast in the event of collision.

Fast Paxos is a fast consensus algorithm that is a variant of classic Paxos. In the normal case, learning occurs in two message delays when there is no collision and can be guaranteed to occur in three message delays even with a collision. Moreover, it can achieve any desired degree of fault tolerance using the smallest possible number of processes.

The basic idea behind Fast Paxos also underlies an earlier algorithm of Brasileiro et al. [1]. However, they considered only the traditional consensus problem, so they failed to realize that their algorithm could be easily

L. Lamport (✉)  
Microsoft Research,  
1065 La Avenida,  
Mountain View, CA 94043, USA

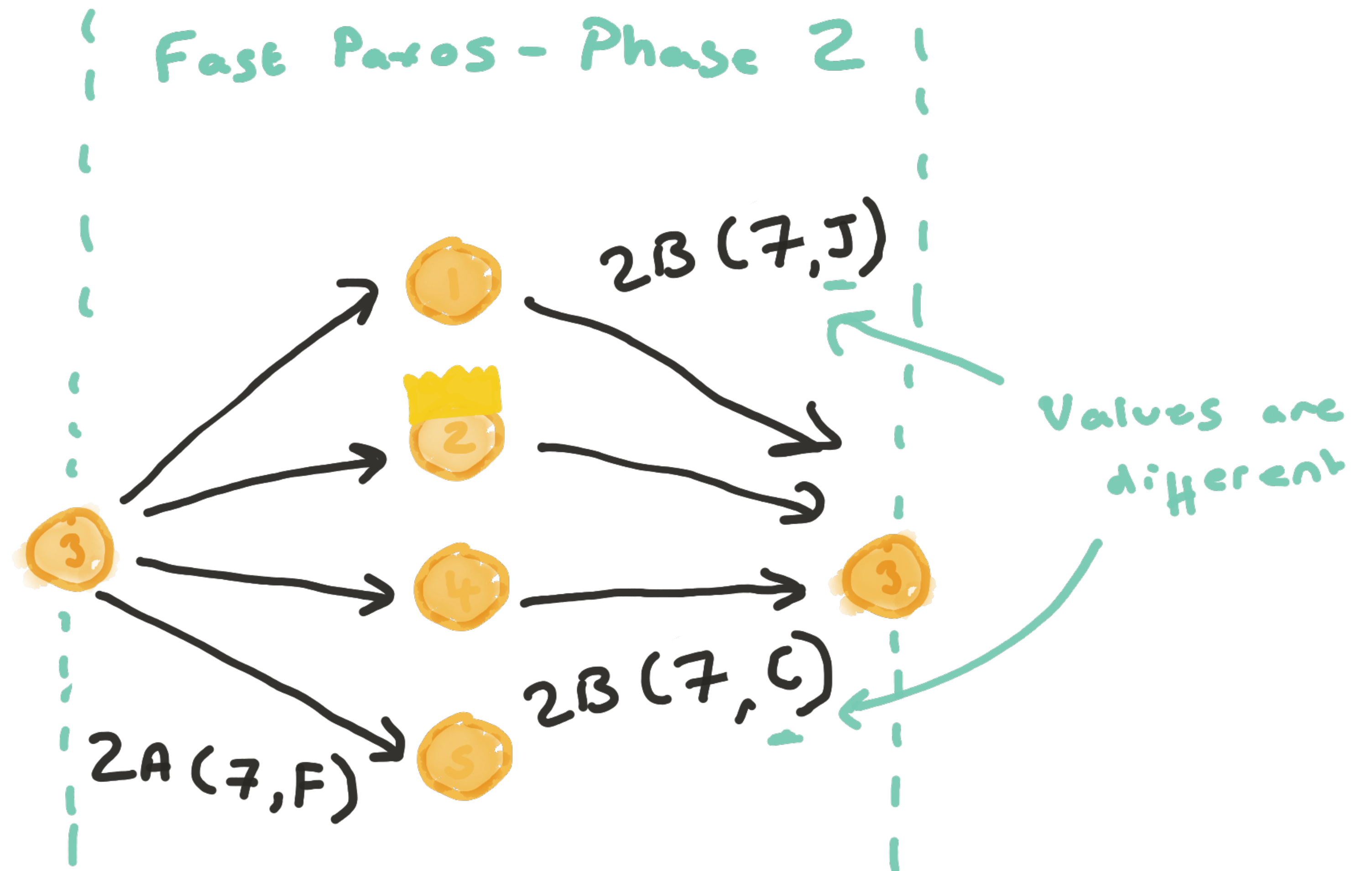
# Fast Paxos - Phase 2



# New algorithm, new problems

Fast Paxos solves our issue with leaders but it introduces the issue of conflicts.

Recovering from conflicts is expensive.

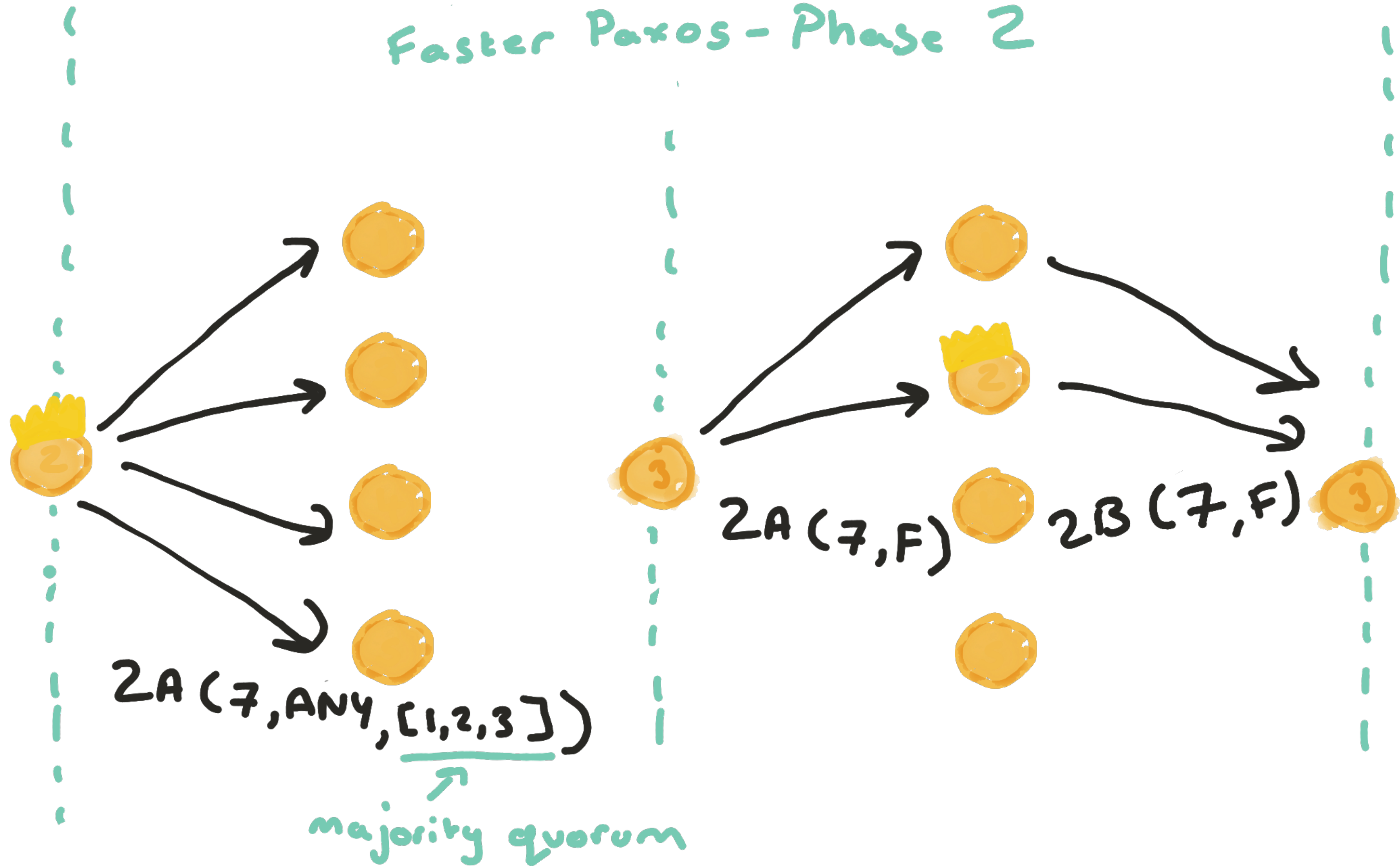


# Introducing Faster Paxos

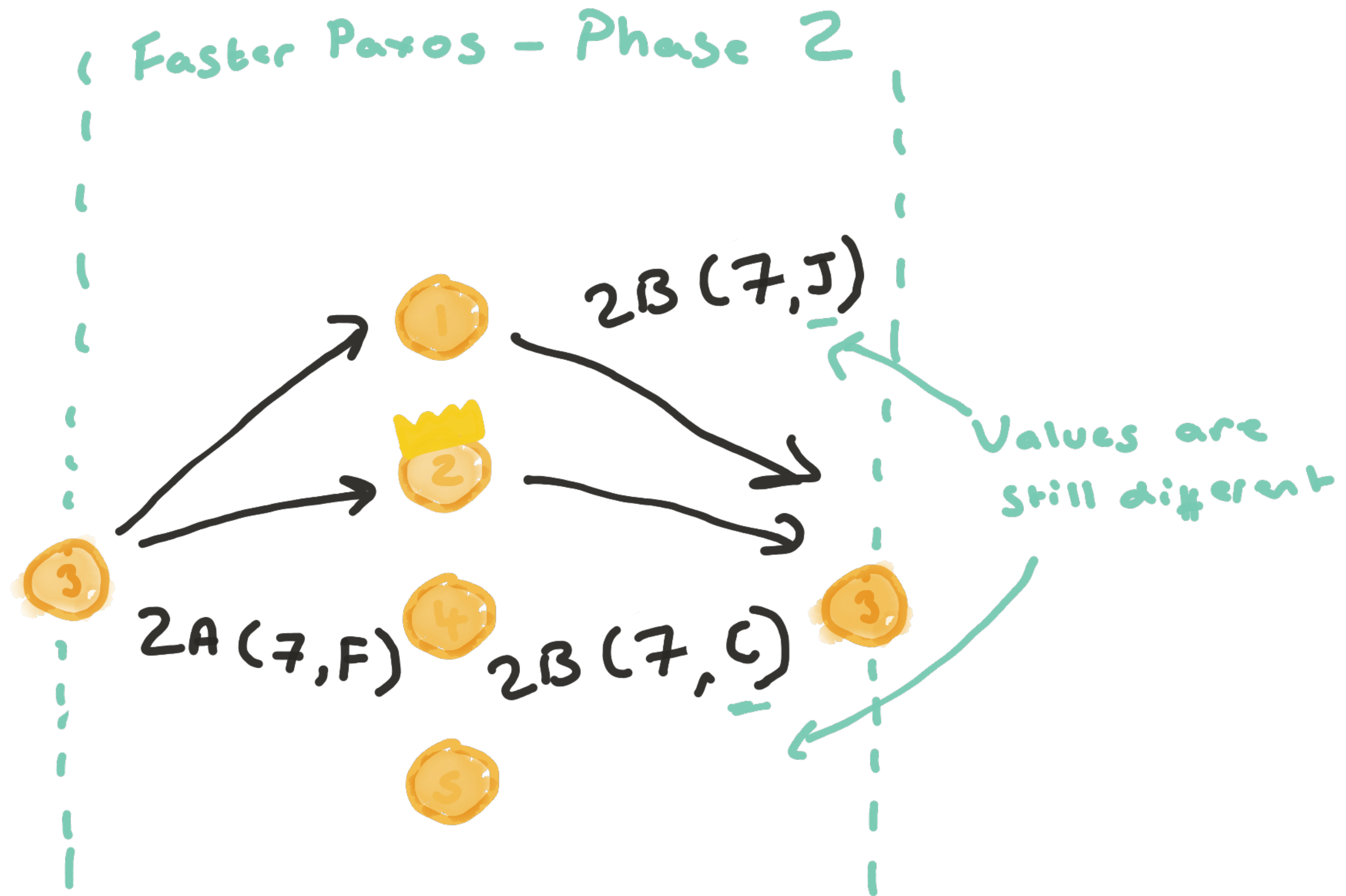
Faster Paxos is similar to Paxos, except that a leader can choose to allow any node to send values directly to a chosen majority quorum of nodes.



# Faster Paxos - Phase 2



# Faster Paxos - Conflicts





# Requirements of consensus

The aim of distributed consensus is to decide a single value.

**Validity** - The decided value must have been proposed by some node.

**Agreement** - All nodes must learn the same decided value.

**Termination** - Eventually, all nodes must learn the decided value.

Is this what we actually want?

# Requirements of faster consensus

The aim of distributed consensus is to decide non-empty ordered set of values.

**Validity** - Each decided value must have been proposed by some node.

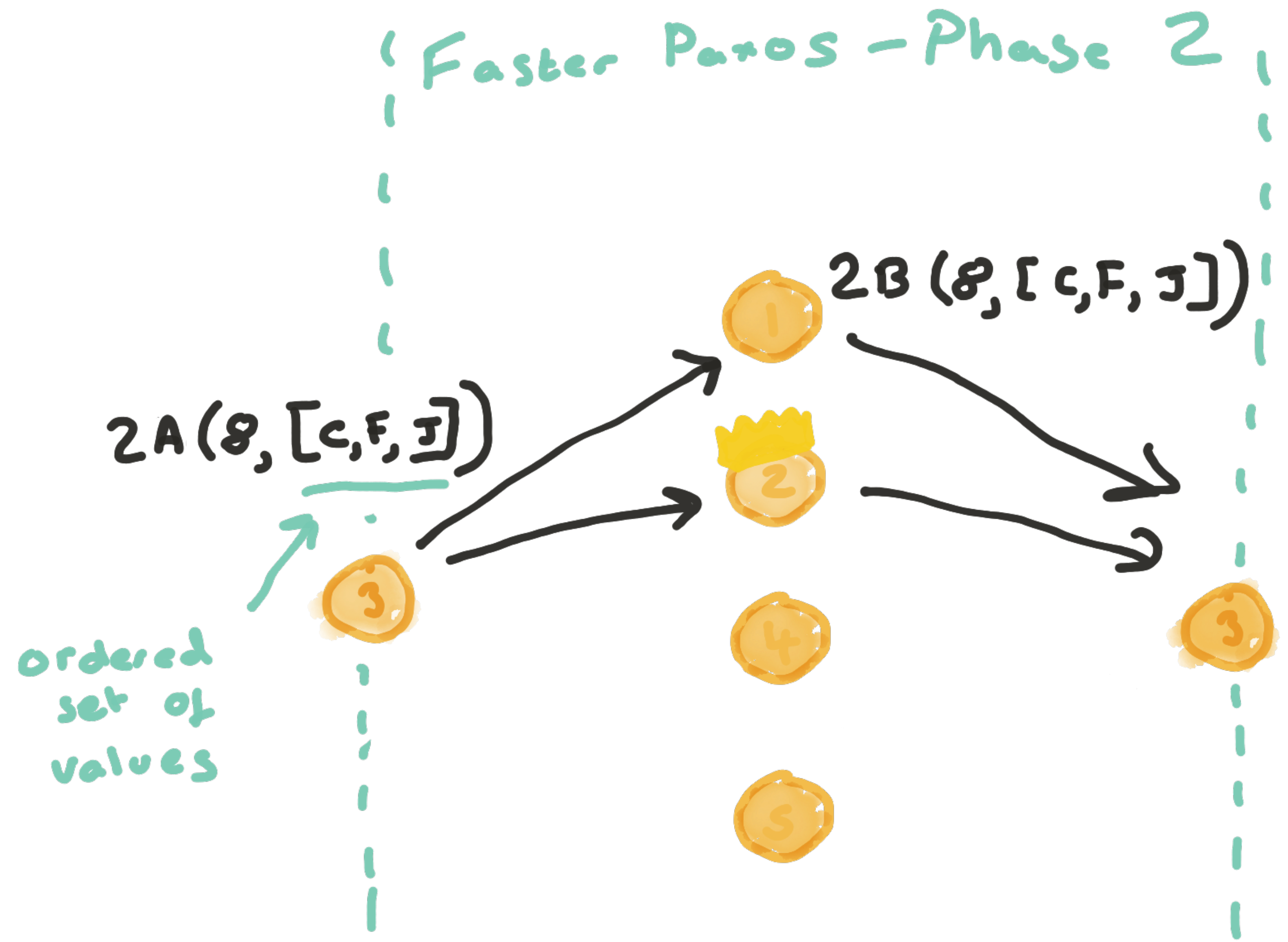
**Agreement** - All nodes must learn the same ordered set of decided values.

**Termination** - Eventually, all nodes must learn the set of decided values.

# Faster Paxos - Conflict resolution

When a conflict is detected, each node retries with the set of values it has seen.

Each node will see the same values so next time the proposals will not conflict.



# Summary

Faster Paxos solves consensus with optimal latency in the absence of conflicts/failures. Any conflict are resolved in 2 additional steps.

Faster Paxos however would not be suit to systems where:

- Performance is not a concern.
- Node failures and/or network partitions are very common.
- One node is the source of most options.

# Faster Paxos is one of many options

## Flexible Paxos: Quorum Intersection Revisited

Heidi Howard<sup>1</sup>, Dahlia Malkhi<sup>2</sup>, and Alexander Spiegelman<sup>3</sup>

- 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.

1998 ACM Subject Classification C.2.4 Distributed Systems

Keywords and phrases Paxos, Distributed Consensus, Quorums

Digital Object Identifier 10.4230/LIPIcs.OPODS.2016.25

### 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 [36]. Lamport's Paxos algorithm [19, 20] 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 requirement, quorums are typically composed of any majority from a fixed set of participants, although other quorum schemes have been proposed.

In practice, we usually wish to reach agreement over a sequence of values, known as Multi-Paxos [20]. We use the first phase of Paxos to establish one participant as a *leader* and the second phase of Paxos to propose a series of values. To commit a value, the leader must

© Heidi Howard, Dahlia Malkhi, and Alexander Spiegelman; licensed under Creative Commons License CC-BY  
20th International Conference on Principles of Distributed Systems (OPODS 2016).  
Editors: Panagiota Fatourou, Ernesto Jiménez, and Fernando Pedone; Article No. 25; pp. 25:1–25:14  
Leibniz International Proceedings in Informatics  
LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

## Technical Report

UCAM-CL-TR-935  
ISSN 1476-2986

Number 935



## Distributed consensus revised

Heidi Howard

April 2019

15 JJ Thomson Avenue  
Cambridge CB3 0FD  
United Kingdom  
phone +44 1223 763500  
<https://www.cl.cam.ac.uk/>

## A Generalised Solution to Distributed Consensus

Heidi Howard, Richard Mortier  
University of Cambridge  
first.last@cl.cam.ac.uk

### Abstract

Distributed consensus, the ability to reach agreement in the face of failures and asynchrony, is a fundamental primitive for constructing reliable distributed systems from unreliable components. The Paxos algorithm is synonymous with distributed consensus, yet it performs poorly in practice and is famously difficult to understand. In this paper, we re-examine the foundations of distributed consensus. We derive an abstract solution to consensus, which utilises immutable state for intuitive reasoning about safety. We prove that our abstract solution generalises over Paxos as well as the Fast Paxos and Flexible Paxos algorithms. The surprising result of this analysis is a substantial weakening to the quorum requirements of these widely studied algorithms.

### 1 Introduction

We depend upon distributed systems, yet the computers and networks that make up these systems are asynchronous and unreliable. The longstanding problem of distributed consensus formalises how to reliably reach agreement in such systems. When solved, we become able to construct strongly consistent distributed systems from unreliable components [13, 21, 4, 17]. Lamport's Paxos algorithm [14] is widely deployed in production to solve distributed consensus [5, 6], and experience with it has led to extensive research to improve its performance and our understanding but, despite its popularity, both remain problematic.

Paxos performs poorly in practice because its use of majorities means that each decision requires a round trip to many participants, thus placing substantial load on each participant and the network connecting them. As a result, systems are typically limited in practice to just three or five participants. Furthermore, Paxos is usually implemented in the form of *Multi-Paxos*, which establishes one participant as the *master*, introducing a performance bottleneck and increasing latency as all decisions are forwarded via the master. Given these limitations, many production systems often opt to sacrifice strong consistency guarantees in favour of performance and high availability [7, 3, 18]. Whilst compromise is inevitable in practical distributed systems [10], Paxos offers just one point in the space of possible trade-offs. In response, this paper aims to improve performance by offering a generalised solution allowing engineers the flexibility to choose their own trade-offs according to the needs of their particular application and deployment environment.

Paxos is also notoriously difficult to understand, leading to much follow up work, explaining the algorithm in simpler terms [20, 15, 19, 23] and filling the gaps in the original description, necessary for constructing practical systems [6, 2]. In recent years, immutability has been increasingly widely utilised in distributed systems to tame complexity [1]. Examples such as append-only log stores [1, 8] and CRDTs [22] have inspired us to apply immutability to the problem of consensus.

1

Flexible Paxos:  
Quorum intersection  
revisited, 2016

Distributed consensus  
revised, 2018

A generalised solution  
to distributed  
consensus, 2019

Q & A

Heidi Howard @ Cambridge University  
[heidi.howard@cl.cam.ac.uk](mailto:heidi.howard@cl.cam.ac.uk)