Distributed Consensus

CS4405 – Analysis of Concurrent and Distributed Programs

Burcu Kulahcioglu Ozkan

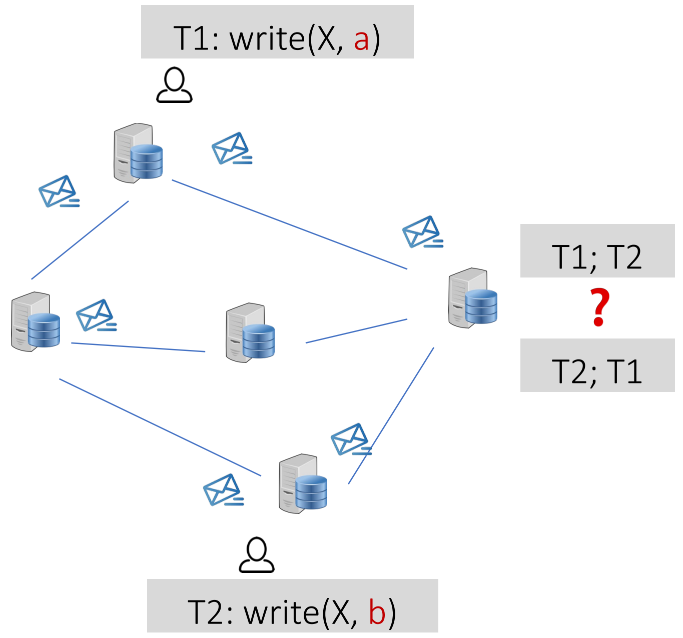
TU Delft

Distributed decision making

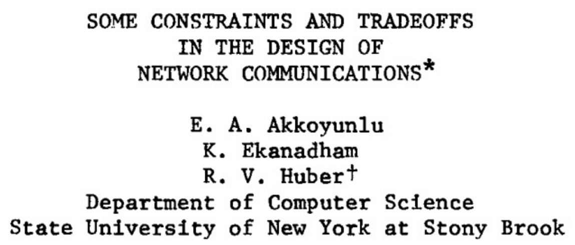
Consensus is a fundamental problem in distributed systems.

Providing agreement in the presence of process and network faults:

- Committing a transaction
- Synchronizing state machines
- Leader election
- Mutually exclusive access to a resource
- Atomic broadcasts



The 2-generals problem



SOME CONSTRAINTS AND TRADEOFFS
IN THE DESIGN OF
NETWORK COMMUNICATIONS*

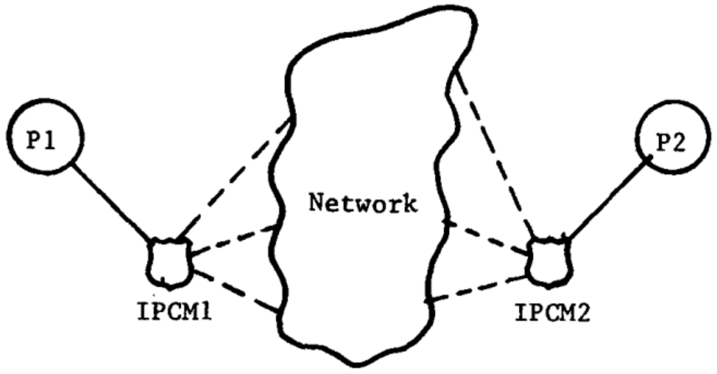
E. A. Akkoyunlu
K. Ekanadham
R. V. Huber†
Department of Computer Science
State University of New York at Stony Brook

SOSP, 1975

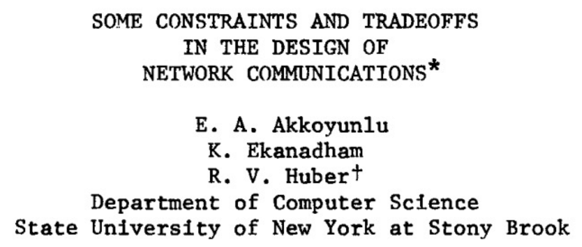
- A thought experiment to illustrate the pitfalls and design challenges of coordinating an action by communicating over an unreliable link

Can the generals guarantee a synchronized attack?

- 2 armies camped in opposing hills (A1 and A2)
- The are only able to communicate with messengers
- They need to decide on a time to attack
- Enemy (B) is camped between the two hills and can at any time intercept the messengers



The 2-generals problem (cont'd)


SOME CONSTRAINTS AND TRADEOFFS
IN THE DESIGN OF
NETWORK COMMUNICATIONS*

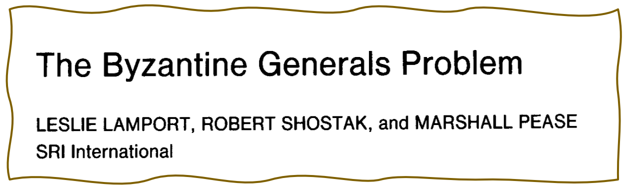
E. A. Akkoyunlu
K. Ekanadham
R. V. Huber†
Department of Computer Science
State University of New York at Stony Brook

SOSP, 1975

- It is impossible to make a reliable decision
  - Main problem: lack of common knowledge

- Approximate solutions: Accept the uncertainty of the communication channel and mitigate it to a sufficient degree
  - Pre-agree on timeouts
  - Send n labeled messages
  - Receiver calculates received messages within time window, then decides how many messages to send for ack.

Byzantine Generals problem

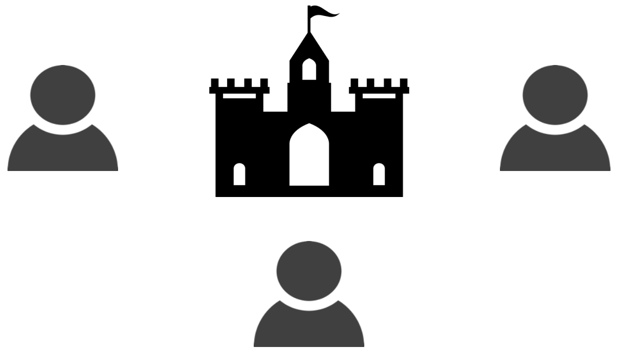

The Byzantine Generals Problem

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE
SRI International

TOPLAS, 1982

- Formulated by Lamport et al., the Byzantine generals problem shaped distributed systems research for the next 40 years.
  - Several divisions of the Byzantine army are camped outside an enemy city, each division commanded by its own general
  - The generals can communicate with each other only by messengers
  - They must decide upon a common plan of action: Attack or Retreat
  - There might be traitors (malicious or arbitrary behavior)

How can loyal generals agree on a plan?



Byzantine Generals solution: Fault tolerant consensus protocols

With only three Generals no solution can work in the presence of a single traitor

A consensus protocol defines a set of rules for message exchange and processing for distributed components to reach agreement

Fault tolerant consensus, e.g.,:

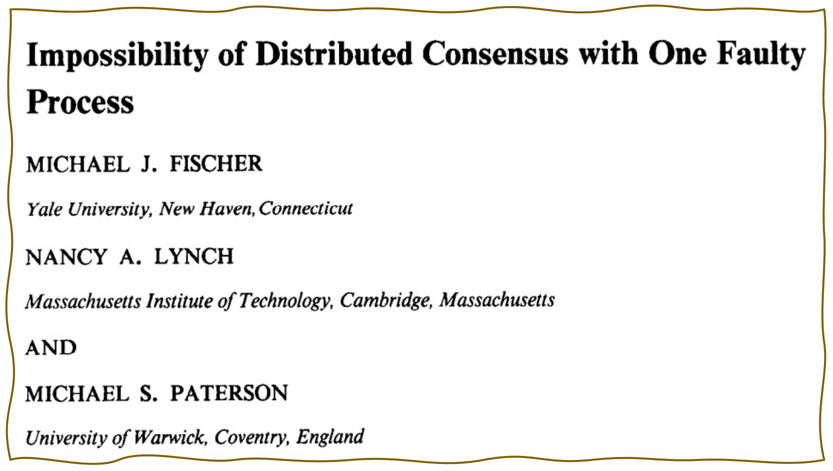
- PBFT - Byzantine fault tolerant consensus with at least 3f+1 nodes with f traitors
- Paxos, Raft - Crash fault tolerant consensus with at least 2f+1 nodes with f faulty nodes
  - Once a majority agrees on a value, that is consensus

Consensus protocols/algorithms

Given a set of processes each with an initial value, consensus requires:

- Termination: All non-faulty processes eventually decide on a value
- Integrity: No correct process decides twice
- Validity: The value that has been decided must have proposed by some (correct) process
- Agreement: No two (correct) processes decide differently

FLP Impossibility Result



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

JACM, 1985

- *"It is impossible to have a deterministic consensus algorithm that can satisfy agreement, validity, termination, and fault tolerance (of even a single process) in an asynchronous distributed system."*

- States that in an asynchronous system there is no algorithm

  that solves consensus in every possible run

  - For any protocol, there exists a configuration that is always bivalent (undecided)
  - The heart of the FLP result is the impossibility to distinguish slow vs crashed processes

- Consensus protocols in practice use randomization and partial synchrony (e.g., "failure detectors")

Paxos consensus algorithm

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

ACM Trans. Comput. Syst,1998

Paxos Made Simple

L. Lamport • Published 2001 • Education

The Paxos algorithm, when presented in plain English, is very simple.

- Proposed by Lamport in 1989
- Consensus in the existence of process crashes or network faults
  - Tolerates f crash faults in an execution with 2f+1
- Consensus is reached once majority agrees on a proposal

- Safety – always safe
- Liveness – very often live
  - Some value eventually chosen if fewer than half of processes fail
  - Conditions to prevent progress is extremely unlikely

Paxos algorithm: Process roles

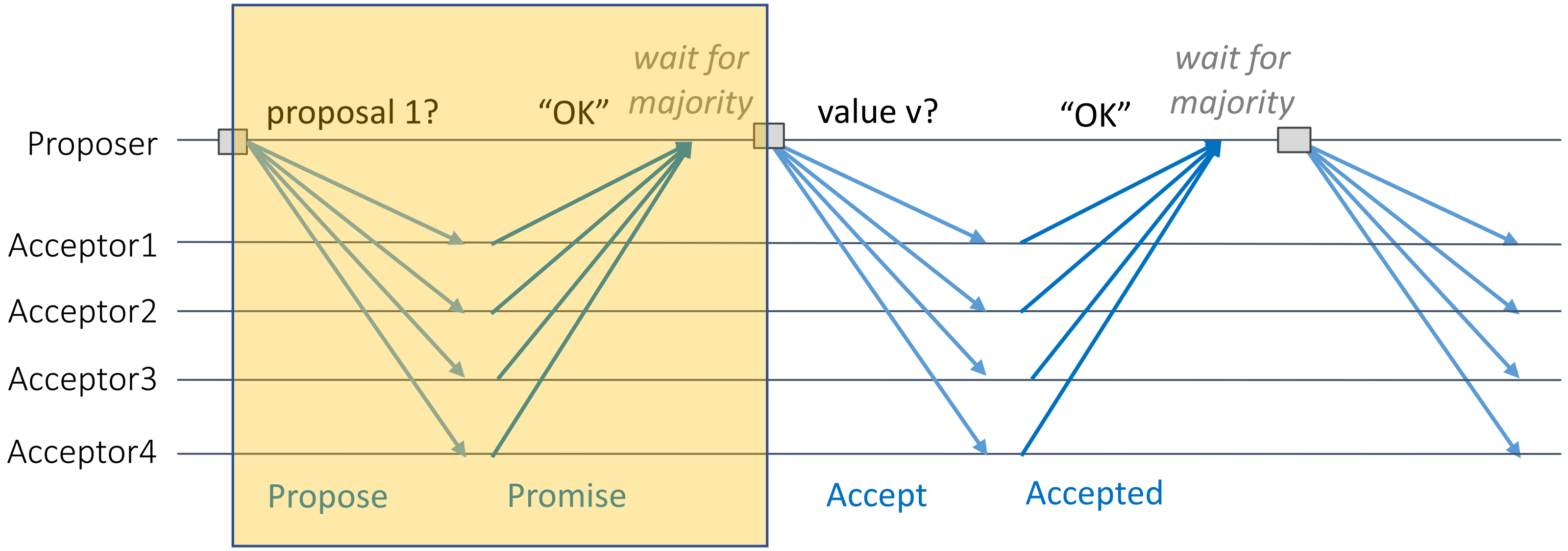
- Three roles:
  - **Proposer**: Chooses a value (or receives from a client) and sends it to a set of acceptors to collect votes
  - **Acceptor**: Vote to accept or reject the values proposed by the proposer. For fault tolerance, the algorithm requires only a majority of acceptor votes
  - **Learner**: They adopt the value when a large enough number of acceptors have accepted it.
- A process can play any/all roles
- Every proposal <n, v> consists of a value, proposed by the client, and a unique monotonically increasing proposal number n, aka “ballot” number
- Acceptance of the proposals by a majority of processes/servers provide fault tolerance.

Paxos consensus algorithm - Voting

Phase1 (Voting)

- (Prepare)A proposer selects a proposal number (ballot) n and sends a “prepare” request Prepare(n) to acceptors.
- (Promise) If *n* is higher than every previous proposal number received, then the Acceptor returns "Promise", to the Proposer, to ignore all future proposals having a number less than *n*.

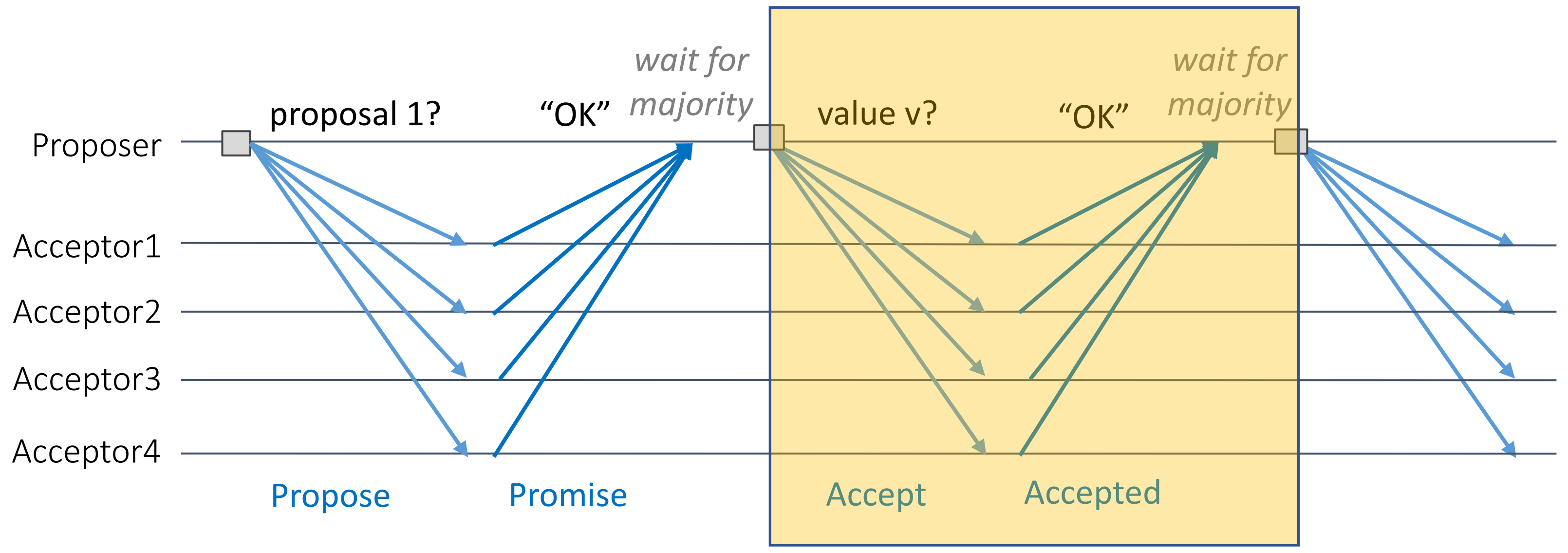
If the Acceptor *accepted* a proposal at some point in the past, it must include the previous proposal number, say *m*, and the corresponding accepted value, say *w*, in its response to the Proposer.



Paxos consensus algorithm - Replication

Phase2 (Replication)

- (Accept) If the proposer receives a response from a majority of acceptors, then it sends an accept request for a proposal numbered n with the highest-numbered proposal among the responses.
- (Accepted) If an acceptor receives an accept request for a proposal numbered n, it accepts the proposal unless it has already responded to a prepare request having a number greater or equal than the proposal number.

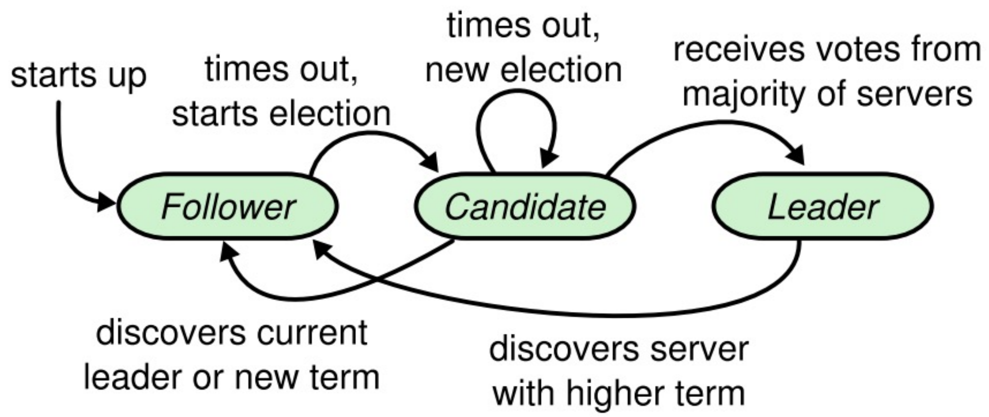


Raft consensus algorithm

In Search of an Understandable Consensus Algorithm
Diego Ongaro and John Ousterhout
Stanford University

USENIX, 2014

- Leader-based asymmetric model: A node in a system can only be in one of the three states at any point in time:
  - Leader, follower, or candidate
- Separates leader election and log replication states:


starts up
times out, starts election
times out, new election
receives votes from majority of servers
Follower
Candidate
Leader
discovers current leader or new term
discovers server with higher term

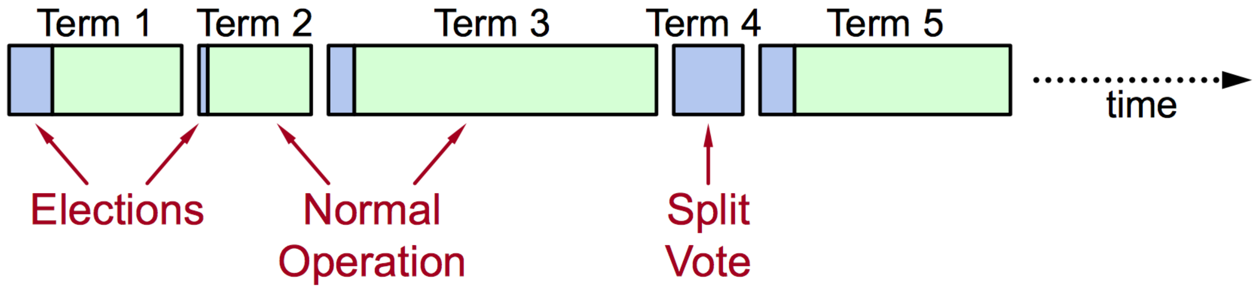
Raft consensus algorithm - Cluster states

Leader election

- Select one server to act as leader
- Detect crashes, choose new leader

- Log replication (normal operation)
  - Leader accepts commands from clients, appends to its log
  - Leader replicates its log to other servers (overwrites inconsistencies)

Raft ensures that: logs are always consistent and that only servers with up-to-date logs can become leader



Raft - Leader election

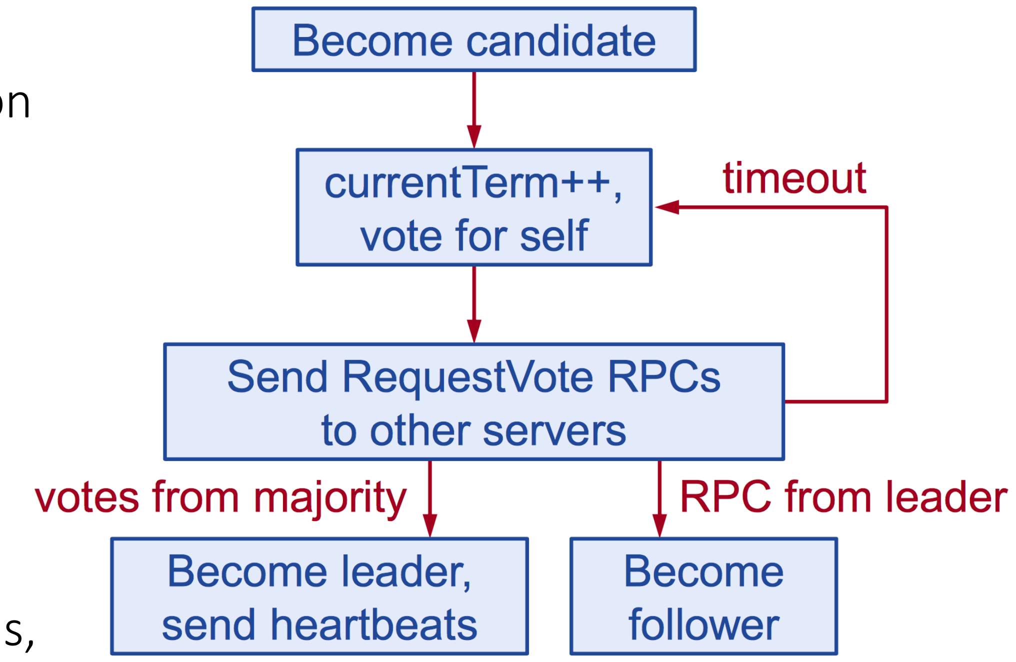
Raft defines the following server states:

- **Candidate**: Candidate for being a leader, asking for votes
- **Leader**: Accepts log entries from clients, replicates them on other servers
- **Follower**: Replicate the leader's state machine

Each server maintains current term value (no global view):

- Exchanged in every RPC
- Peer has later term? Update term, revert to follower

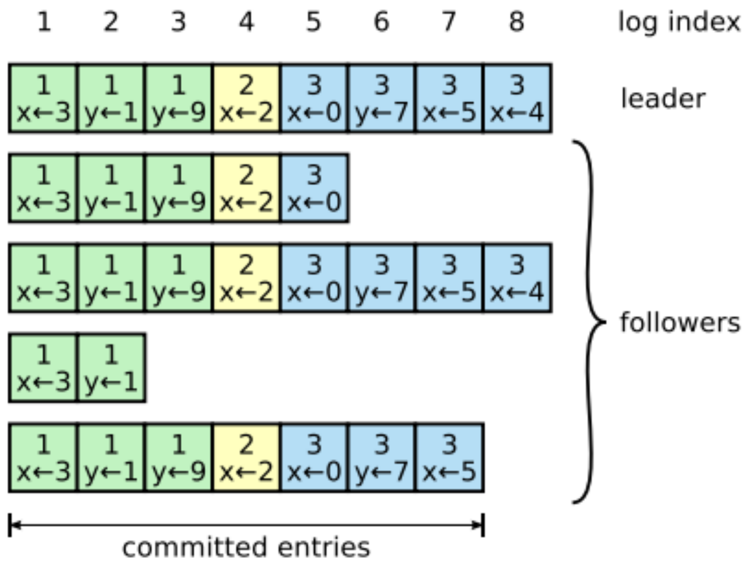
Raft selects at most one leader at each term. For some terms, the election may fail and result in no leader for that term.


Become candidate
currentTerm++, vote for self
timeout
Send RequestVote RPCs to other servers
votes from majority
RPC from leader
Become leader, send heartbeats
Become follower

The leader for any given term contains all of the entries committed in previous terms

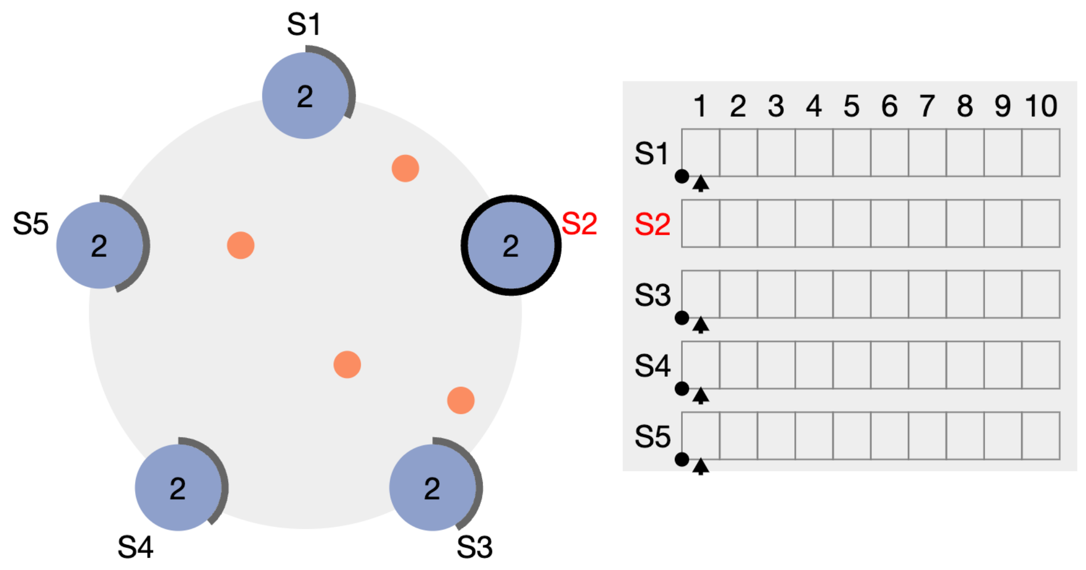
Raft – Log replication

- The leader accepts log entries from clients and replicates them across the servers:
  - Each log entry also has an integer index identifying its position in the log.
  - The leader sends *AppendEntries* message to append an entry to the log
  - A log entry is committed once the leader that created the entry has replicated it on a *majority* of the servers.



1 2 3 4 5 6 7 8
log index
leader
followers
1 x←3 | 1 y←1 | 1 y←9 | 2 x←2 | 3 x←0 | 3 y←7 | 3 x←5 | 3 x←4
1 x←3 | 1 y←1 | 1 y←9 | 2 x←2 | 3 x←0
1 x←3 | 1 y←1 | 1 y←9 | 2 x←2 | 3 x←0 | 3 y←7 | 3 x←5 | 3 x←4
1 x←3 | 1 y←1
1 x←3 | 1 y←1 | 1 y←9 | 2 x←2 | 3 x←0 | 3 y←7 | 3 x←5
committed entries

Raft visualization



https://raft.github.io/