



OST

Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Consensus

Thomas Bocek

01.05.2026

Learning Goals

- Lecture 11 (Consistency / Consensus)
 - Why does scaling lead to consensus problems?
 - Fault models: crash faults vs. Byzantine faults
 - Consensus algorithms: Paxos, Raft
 - Consistency in DHTs: vDHT
 - CRDTs: conflict-free data structures
 - Tradeoffs between approaches

Distributed Systems Categorization - L07S11

“Controlled” Distributed Systems

- Consistency
 - Leader election (Zookeeper/Zab, Paxos, Raft)
- Replication principles
 - More replicas: higher availability, higher reliability, higher performance, better scalability, but: requires maintaining consistency in replicas
- Transparency principles apply

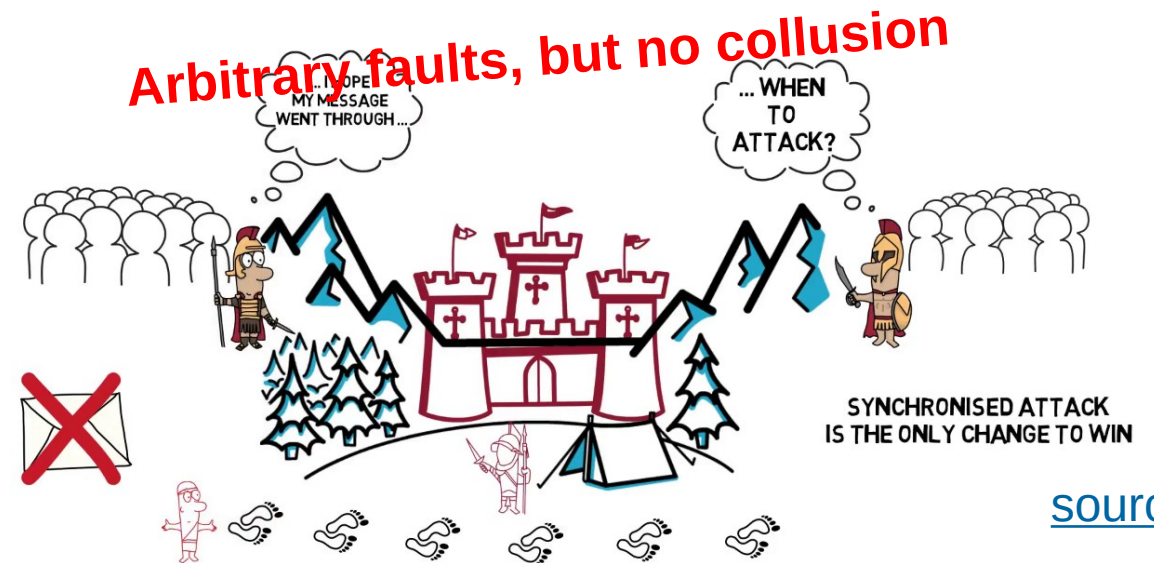
“Fully” Decentralized Systems

- Consistency
 - Weak consistency: DHTs
 - Nakamoto consensus (aka proof of work, [link])
 - Proof of stake – Leader election, PBFT protocols
 - Is Bitcoin eventually consistent?
 - Some argue no, some argue it has even stronger guarantees [link]
- Replication principles apply to fully decentralized systems as well
- Transparency principles apply

Consensus

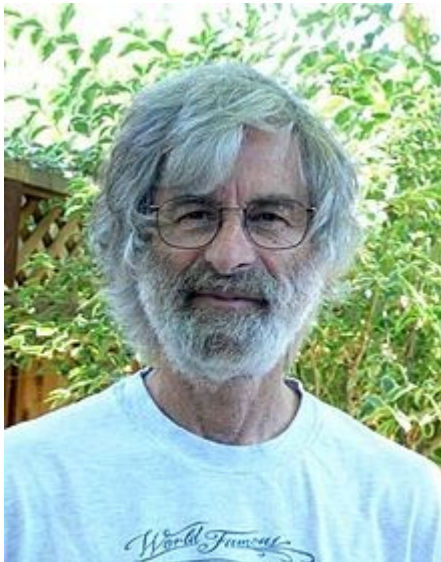
- Definition: Consensus decision-making is a group decision-making process in which group members develop, and agree to support a decision in the best interest of the whole.
- A **Byzantine fault** is an arbitrary fault that occurs during the execution of an algorithm by a distributed system
 - Not only crash, but lie to reach an advantage
- “Controlled” Distributed Systems: your own nodes, your control, no collusion
 - $2f + 1$ vs $3f + 1$
- Find consensus: Paxos, Raft, vDHT, Zookeeper

- Often: consensus defines leader
 - Leader creates block, adds data, creates version
- How to find a leader?



Paxos History

- [Leslie Lamport](#) discovered the Paxos algorithm in late 1980s
 - Attempt to prove that there was no such algorithm which can tolerate the failure of any number of its processes
 - Until he realized that he created working protocol



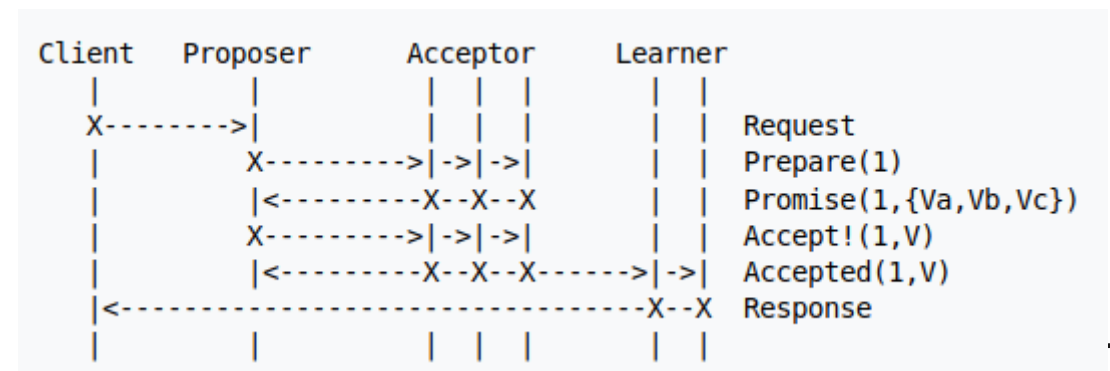
source

Distributed Systems

- Wrote paper and submitted it to Transactions on Computer Systems (TOCS) in 1990
 - Reviewer: was mildly interesting, but needs significant improvement
 - Leslie Lamport: “so I did nothing with the paper”
- People started to using Paxos to solve problems in distributed systems
- Resubmitted in 1998 to TOCS
 - Accepted without any major changes
- Paxos paper won an ACM SIGOPS Hall of Fame Award in 2012
- [Received Turing award in 2013](#), also due to Paxos
 - “Turing Award is generally recognized as the highest distinction in computer science and the “Nobel Prize of computing”“ [\[link\]](#)”

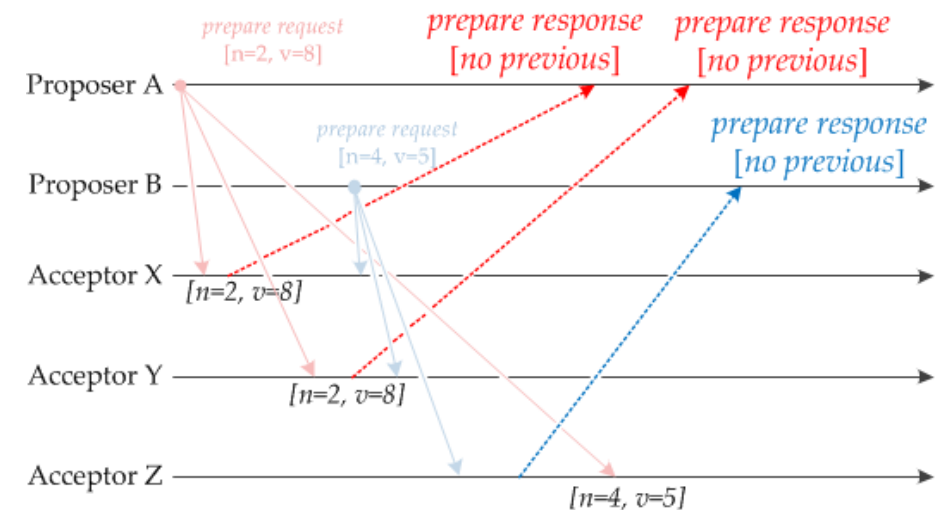
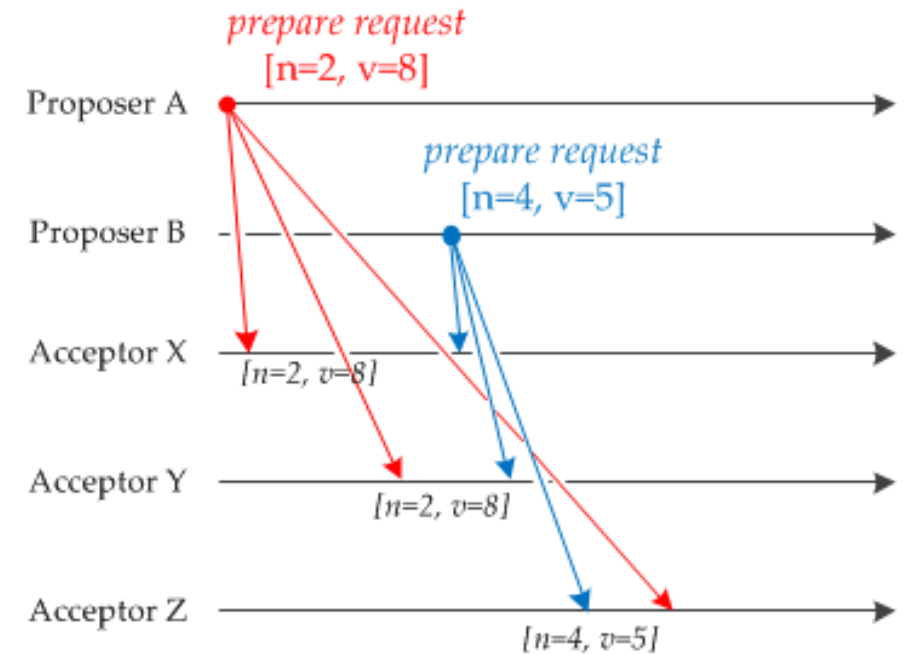
Paxos Consensus

- Paxos: considered difficult to understand
 - “This website explains the infamously difficult to understand Paxos consensus protocol” [link]
 - “Paxos, a really beautiful protocol for distributed consensus” [link]
 - “Paxos is an algorithm whose entire behaviour is subtly difficult to grasp” [link]
- Problem: want reliable computing,
 - But: have unreliable components
- Due to unreliable components, run multiple components, i.e, multiple servers
 - Replica: inconsistent state
- Paxos guarantees that nodes will only ever choose a single value, but does not guarantee that a value will be chosen if a majority of nodes are unavailable
- Roles: proposer, acceptor, and learner
 - Proposer proposes a value that it wants agreement upon
 - Acceptor gets proposal, makes promises, sends result to learners
 - Learners: majority of acceptors must choose the same value
- 2 phases: prepare/promise phase, accept phase



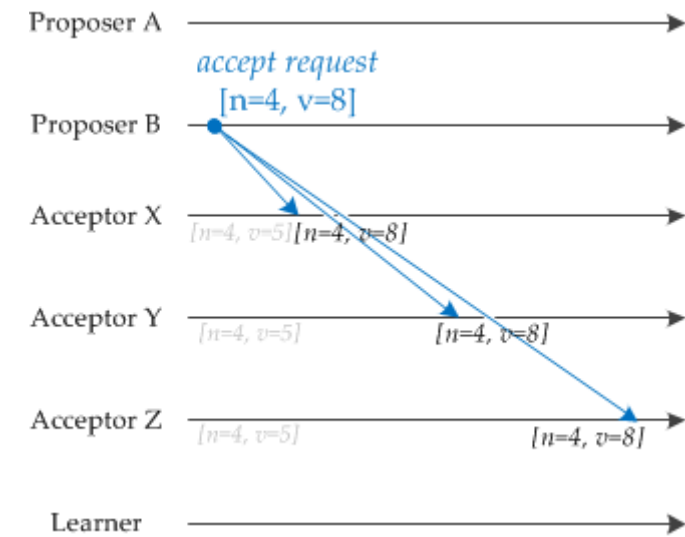
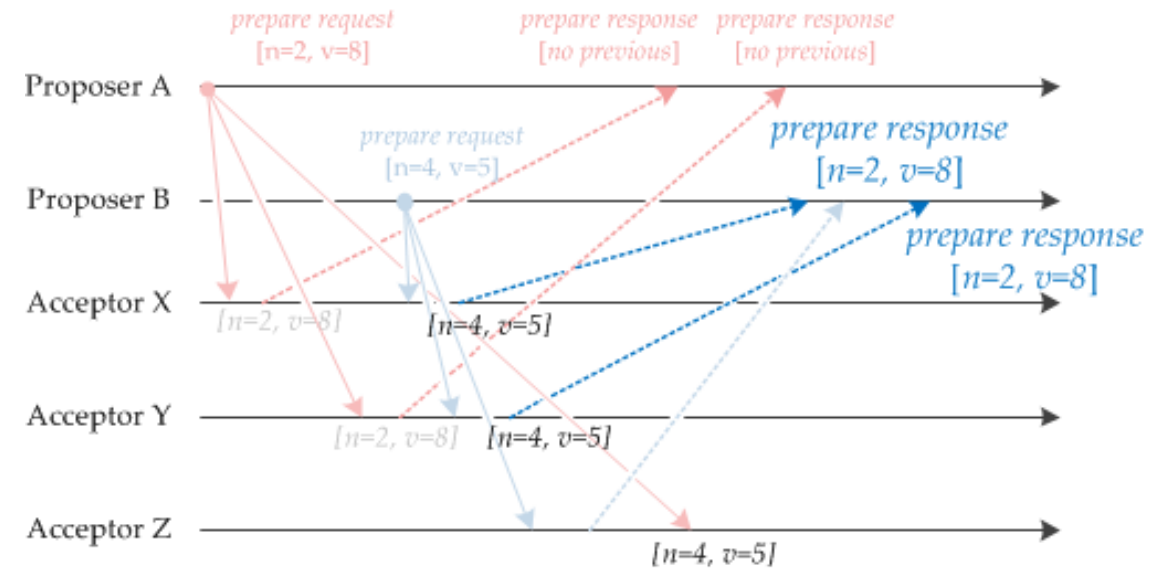
Paxos Example

- Proposer: prepare and accept requests, proposal number and value (n, v) [link]
 - Acceptor: already seen higher proposal number: ignore
 - Acceptor: seen lower proposal number: send back highest accepted n,v.
- Proposer A and proposer B
 - Acceptor Z receives B before A
- If acceptor receiving a prepare request for the first time
 - Acceptor responds with a prepare response
 - Promises never to accept another proposal with a lower proposal number



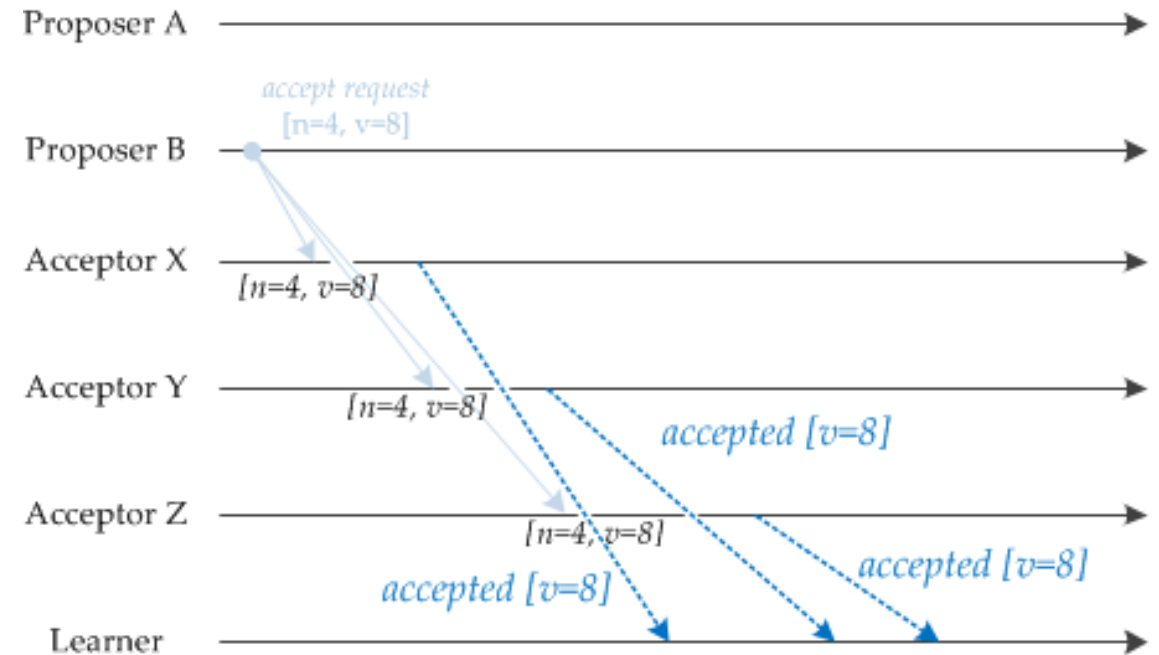
Paxos Consensus

- Acceptor Z receives proposer A's request, and acceptors X and Y receive proposer B's request.
 - Only accept requests with higher number, Acceptor Z not sending response (or negative response) to B.
- Proposer B sends an accept request to each acceptor containing the proposal number it previously used ($n=4$) and the value associated with the highest proposal number among the prepare response messages it received ($v=8$)
- Proposer A sends its accept request before proposer B, but acceptor ignores them



Paxos Consensus

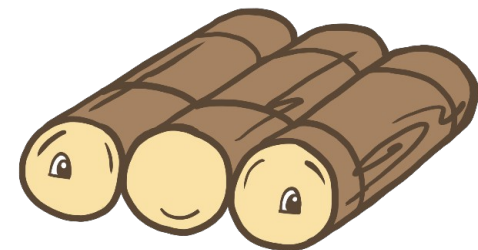
- If an acceptor receives an accept request for a higher or equal proposal number than it has already seen, it accepts and sends a notification to every learner node.
 - A value is chosen by the Paxos algorithm when a learner discovers that a majority of acceptors have accepted a value
 - Once a value is chosen by Paxos, communication with other proposers cannot change value
 - If another proposer, sends a higher proposal number than has previously been seen, with a different value (e.g., $n=6, v=7$), each acceptor responds with the previous highest proposal ($n=4, v=8$)
 - This requires the proposer to send an accept request containing $[n=6, v=8]$, which confirms the value that has already been chosen



- Leader election: the leader is the node that has its data chosen by the Paxos instance [[link](#)] - [youtube](#)

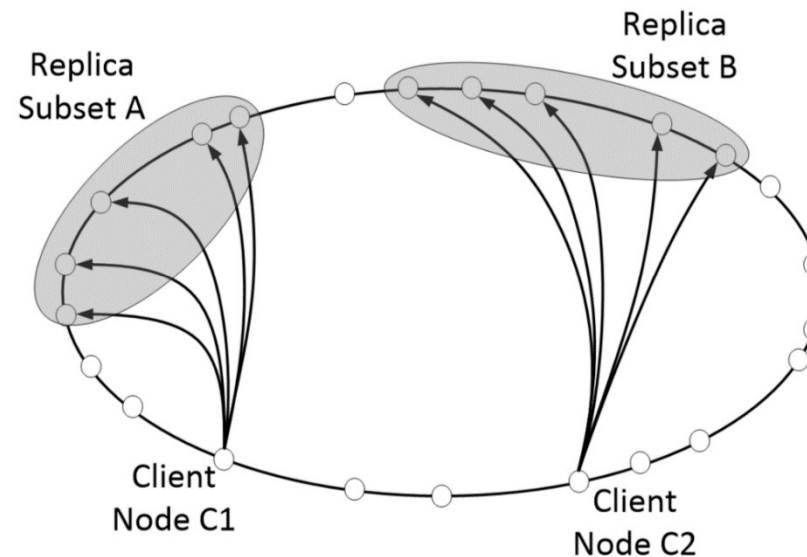
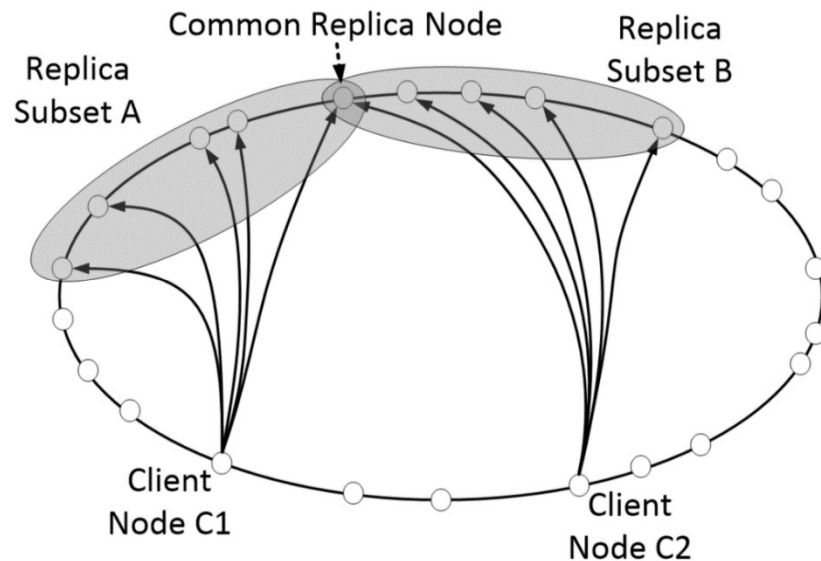
Raft (multi paxos)

- “this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems.” [\[link\]](#)
- **RAFT**: Reliable, Replicated, Redundant, And Fault-Tolerant
- Follower, Candidate, Leader [\[link\]](#)
 - Raft implements leadership election,
 - Once a leader has been elected, all decision-making within the protocol will then be driven only by the leader
 - Only one leader can exist at a single time
- Each follower has a timeout (typically between 150 and 300 ms) in which it expects the heartbeat from the leader.
 - The system is only available when a leader has been elected and is alive
 - Otherwise, a new leader will be elected and the system will remain unavailable for the duration of the vote
 - Starts election by increasing term counter, voting for itself, and sending a message to all other servers requesting their vote
 - If a higher term is received, become follower, if not, leader



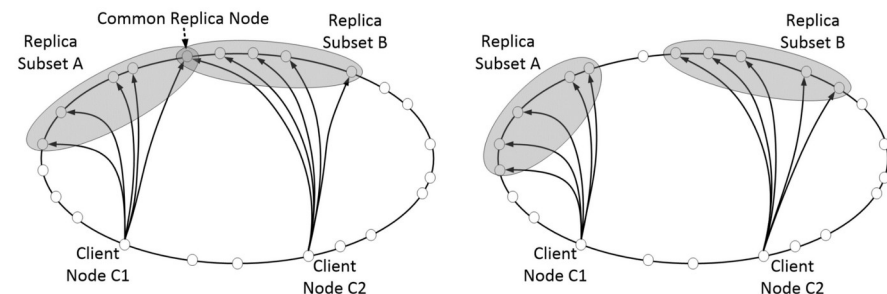
Consistency

- Consistency in DHTs – vDHT, similarities to Paxos
 - Number = versions, for doing updates
 - Simplified roles (peer)
 - No leader election, works well with churn (not heavy churn)
- **CoW**, software transactional memory (**STM**) → for consistent updates. Works for light churn



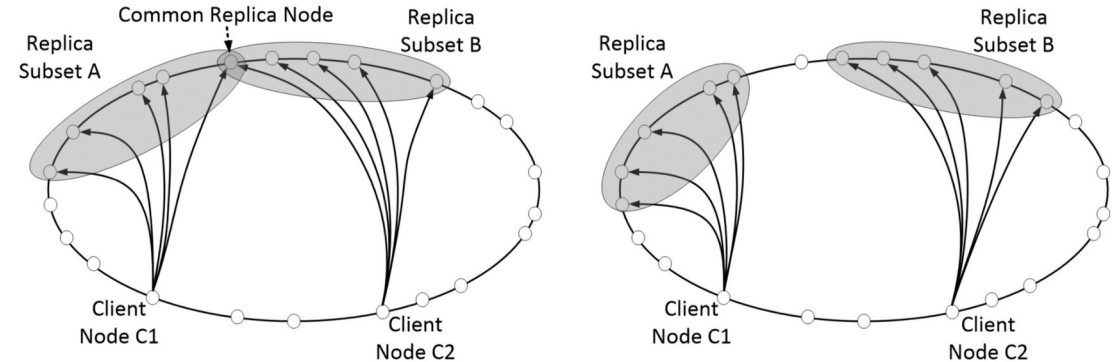
Consistency

- vDHT Basics
 - No locking, no timestamps
 - Every update – new version
 - 1. get() latest version, check if all replica peers have latest version, if not wait and try again
 - - may add delay
 - + wait until update is completed
 - 2. put() prepared with data and short TTL, if status is OK on all replica peers, go ahead, otherwise, remove the data and go to step 1.
 - Peer marks the value as prepared, other put() fail on that key. If nothing happens, TTL
 - Value linked to previous version(s) (hash)
 - 3. put() confirmed, don't send the data, just remove the prepared flag and reset TTL
- In case of heavy churn, API user needs to resolve
 - Get latest version may return fork
 - Abort or resolve (join) manually



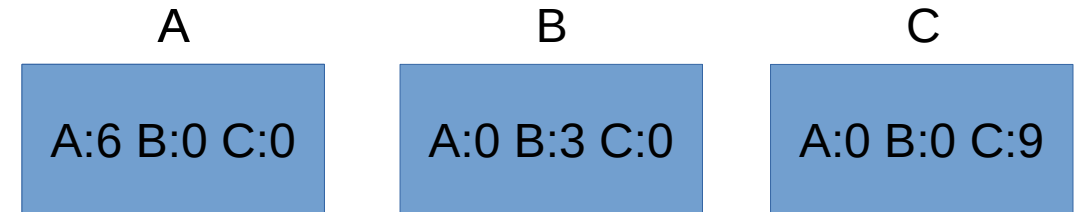
CRDT

- CRDT vs. Paxos/Raft: no leader, no majority needed
 - Available during partitions, but limited to specific data structures
- vDHT \approx operation-based CRDT, but delegates conflict resolution (forks) to application
 - \sim CRDT (operation-based CRDTs)
 - Conflict-free replicated data type (CRDT)
 - \sim git but with no merge conflicts
- CRDT must be
 - Commutative $x \bullet y = y \bullet x$
 - Associative $(x \bullet y) \bullet z = x \bullet (y \bullet z)$
 - Idempotent $x \bullet x = x$



- **CRDT Counter (G-Counter)**

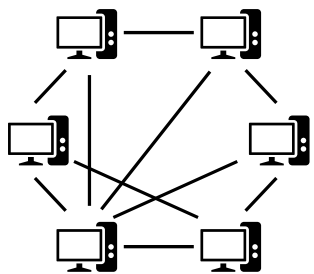
- For each machine 1 array position for counter



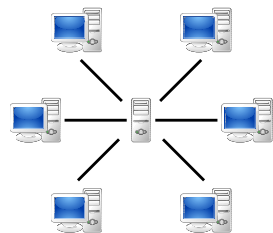
- Merge: max of each counter (A:6 B:3 C:9)
 - Old data A:6 B:2 C:9 merge-max / A:5 B:3 C:2
A:6 B:3 C:9
 - Commutative, associative, idempotent

CRDT

- **Example:** collaborative text editing
 - Google Docs: operational transformation
- Operational transformation
 - Requires central server
- CRDT without central server: P2P
 - You need to find suitable algorithms
 - Collaborative text editing: merge sets
 - Assign for letter an increasing value from 0-1



Distributed Systems



source

H	e	l	o
0	1	2	3

edit

H	e	l	l	o
0	1	2	3	4

Insert "l" at pos 3

H	e	l	o
0	1	2	3

edit

H	e	l	o	!
0	1	2	3	4

Insert "!" at pos 4

Server

H	e	l	l	o	!
0	1	2	3	4	5

Insert "!" at pos 5

H	e	l	l	o	!
0	1	2	3	4	5

Insert "l" at pos 3



OST

CRDT

- Example: collaborative text editing
 - Edit is 0.7: l, 0.9: !
 - Merge with 0.2: H, 0.4: e, 0.6: l, 0.8 o
 - Collision? Find strategy that matches use-case
- Easy!
 - But interleaving – find algorithm that handles interleaving
- CRDT: existing algorithms exist
 - G-Counter, PN-Counter, G-Set, 2P-Set, ...
 - [Yjs](#): CRDT library

H	e	l	o
0.2	0.4	0.6	0.8

H	e	l	l	o	!
0.2	0.4	0.6	0.7	0.8	0.9

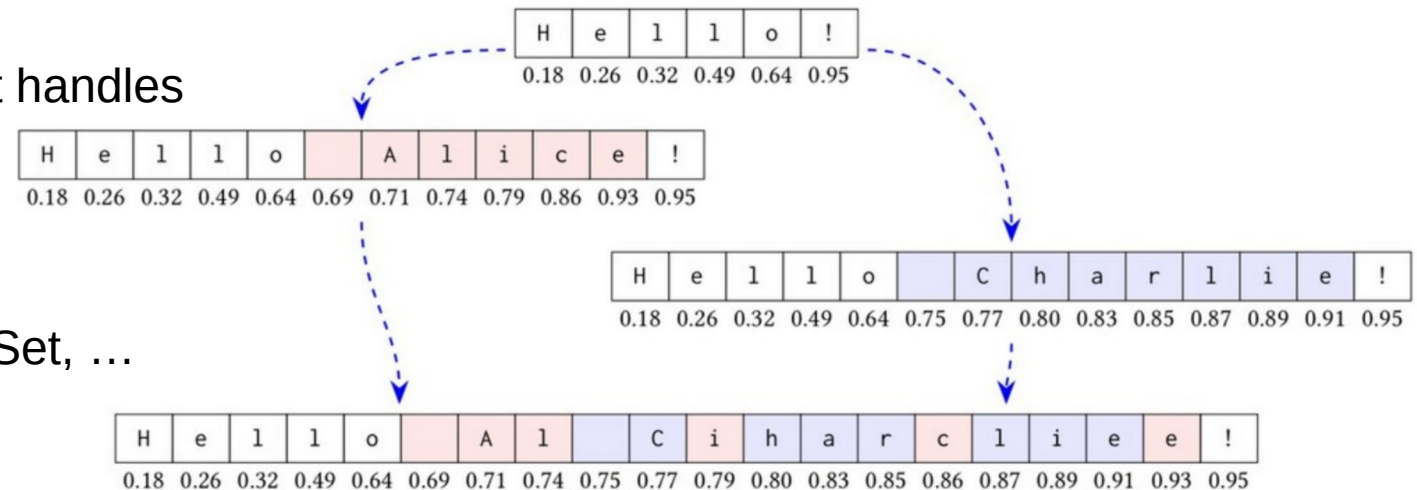


Figure 3: Interleaving due to character positions taken from a dense identifier set, e.g. the rational numbers \mathbb{Q} .