HS18

DISTRIBUTED SYSTEMS ADVANCED & BLOCKCHAIN

Raft, Paxos, vDHT, Consensus without Byzantine Failure

T. Bocek

thomas.bocek@hsr.ch

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

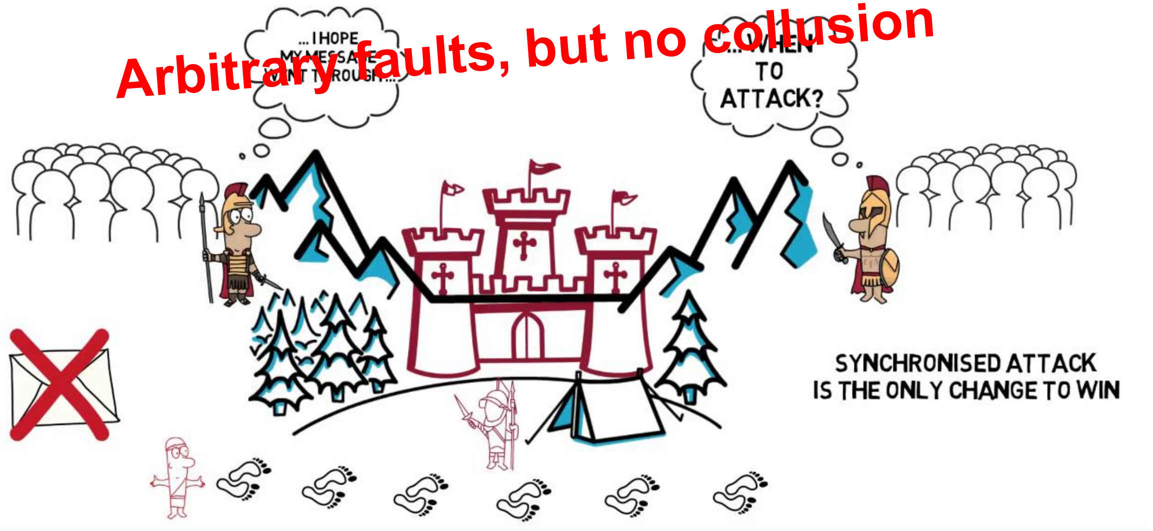
FHO Fachhochschule Ostschweiz

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 or even collude to reach an advantage
- **Your own nodes, your control, no collusion**
- **Other forms of consensus**
  - Paxos
  - Raft
  - vDHT

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

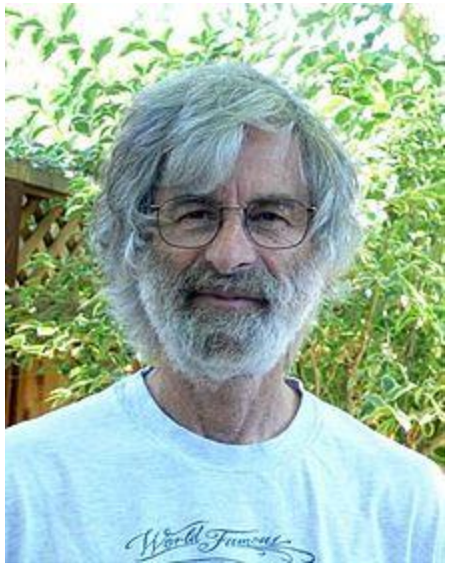
Arbitrary faults, but no collusion



source

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

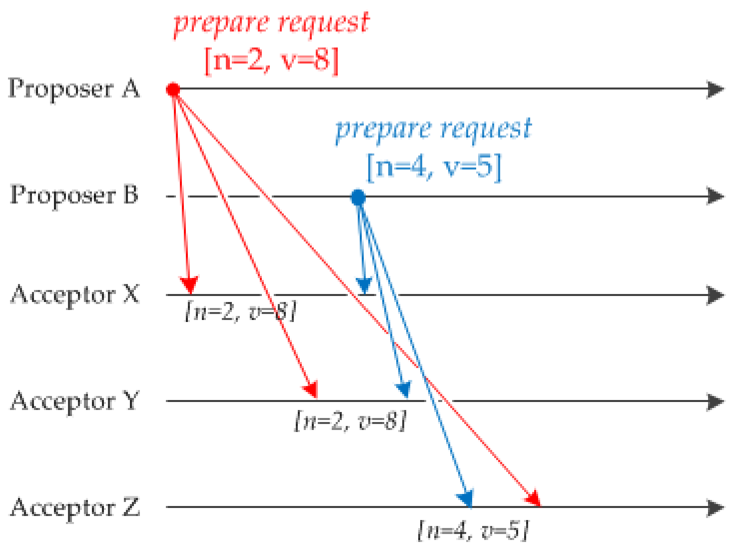
- **Wrote paper and submitted it to Transactions on Computer Systems (TOCS) in 1990**
  - Reviewer: was mildly interesting, but needs significant improvement
  - Leslie Lamport: "so I did nothing with the paper"
- **People started to using Paxos to solve problems in distributed systems**
- **Resubmitted 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""

Paxos consensus

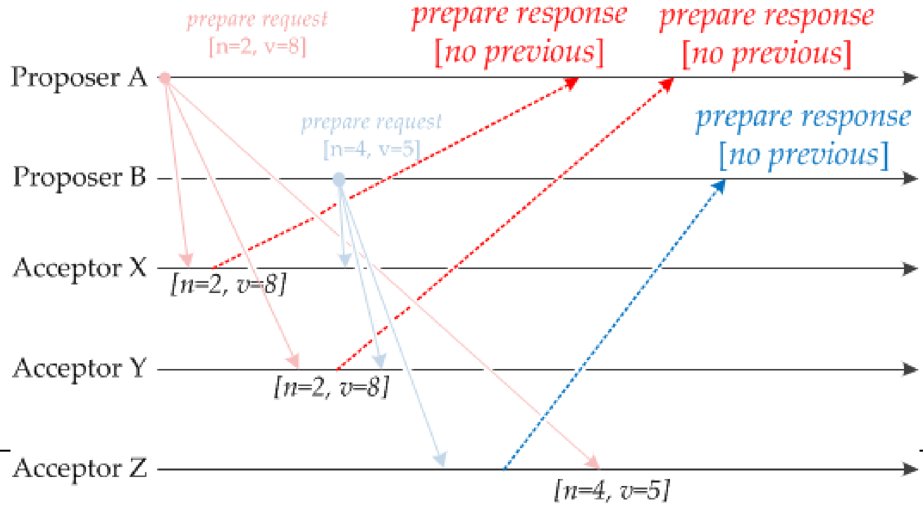
- **Paxos majority considered difficult to understand**
  - “This website explains the infamously difficult to understand Paxos consensus protocol”
  - “Paxos, a really beautiful protocol for distributed consensus”
  - “Paxos is an algorithm whose entire behaviour is subtly difficult to grasp”
- **Problem: 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**

```
Client   Proposer      Acceptor     Learner
   |         |          |  |  |       |  |
   X-------->|          |  |  |       |  |  Request
   |         X--------->|->|->|       |  |  Prepare(1)
   |         |<---------X--X--X       |  |  Promise(1,{Va,Vb,Vc})
   |         X--------->|->|->|       |  |  Accept!(1,V)
   |         |<---------X--X--X------>|->|  Accepted(1,V)
   |<---------------------------------X--X  Response
   |         |          |  |  |       |  |
```

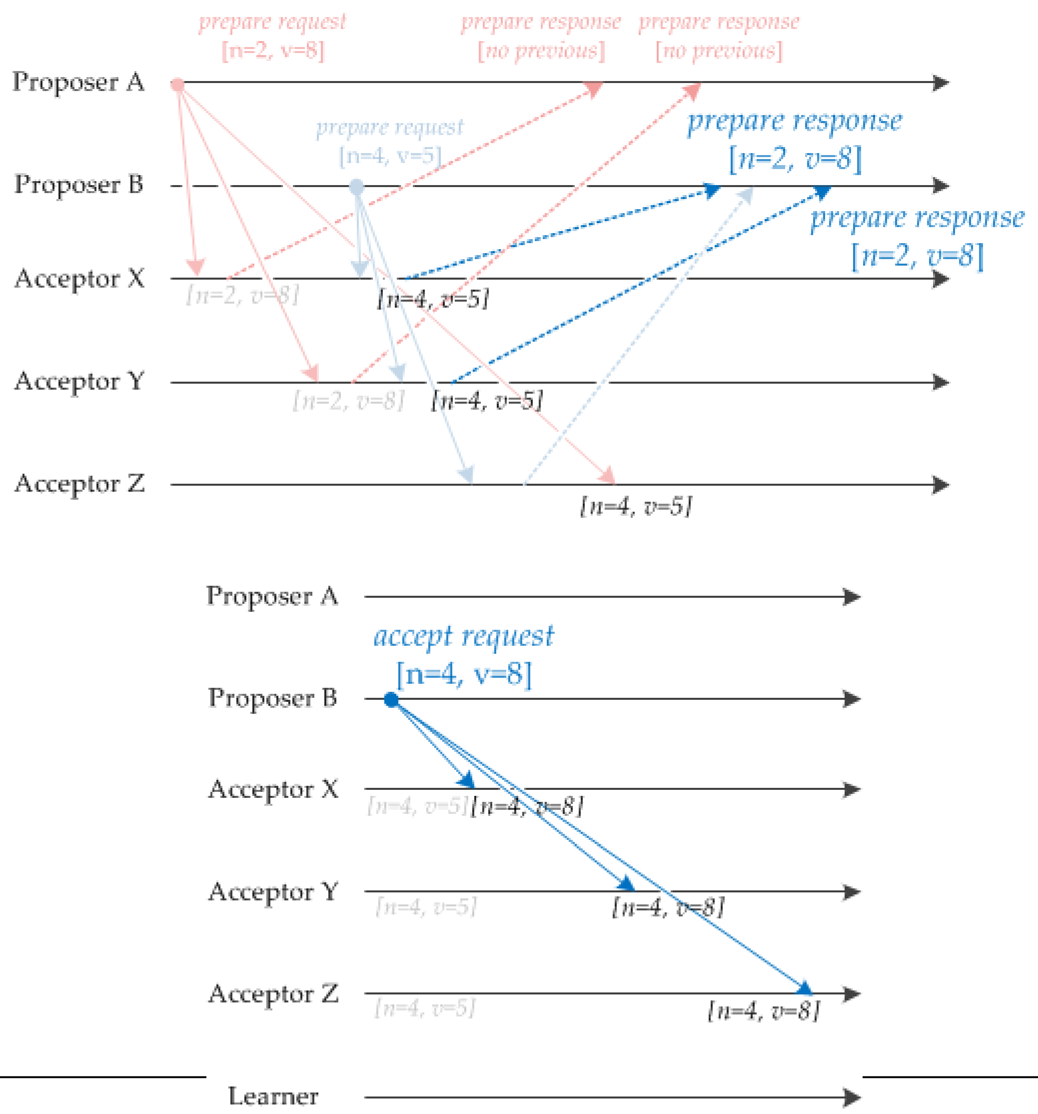
- **Proposer: prepare and accept requests, proposal number and value (n, v)**
  - Only accept if n is the highest proposal number for a prepare request that acceptor has seen so far.
- **Accept phase: proposer adopts value from highest proposal**



- **Proposer A and proposer B**
  - Acceptor Z receives B before A
- **If acceptor receiving a prepare request has not seen another proposal, acceptor responds with a prepare response which promises never to accept another proposal with a lower proposal number**

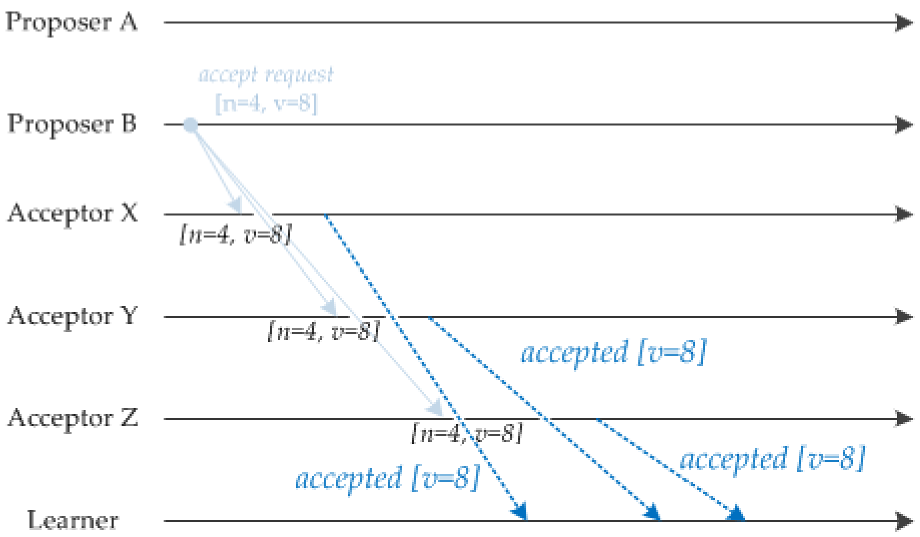


- **Acceptor Z receives proposer A's request, and acceptors X and Y receive proposer B's request.**
  - Only accept requests with higher number, Acceptor Z not sending response (or negative response) to B.
- **Proposer B sends an accept request to each acceptor containing the proposal number it previously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



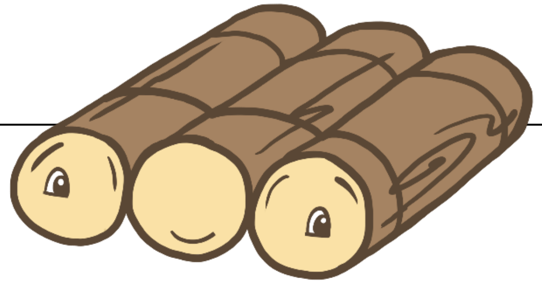
Paxos consensus

- **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
  - If some minority of acceptors not set a value, this ensures to reach eventually consensus on v=8



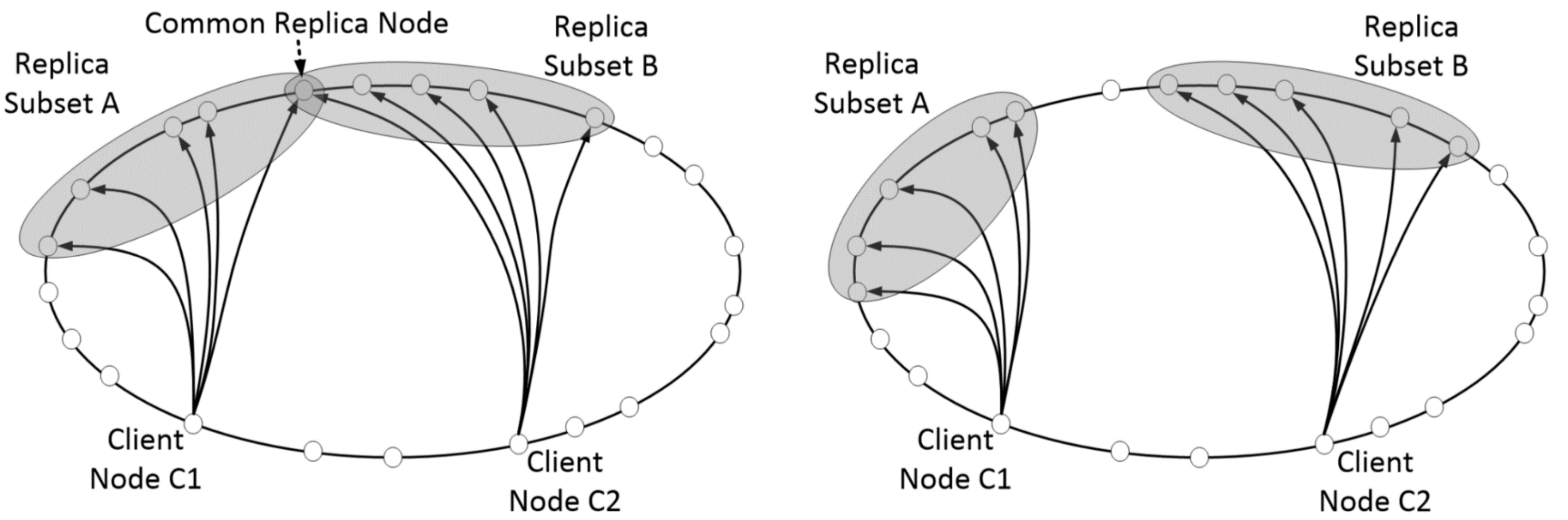
Raft (multi paxos)

- **“this makes Raft more understandable than Paxos and also provides a better foundation for building practical systems. ”**
- **Multi-Paxos: If leaders/acceptors are stable, skip prepare/promise**
- **RAFT: Reliable,Replicated,Redundant, And Fault-Tolerant**
- **Follower, Candidate, Leader**
  - Leader election: Raft implements strong leadership, similar to Zookeeper
  - 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
  - Send value only once (replication outside of consistency mechanism),
  - Simplified roles
  - 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 (replication time may have an influence)
  - 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.
      - Data can be either send now or with the confirm, if sent now we are optimistic
      - 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)

URL: b.socrative.com/student/

Room: HSR

Questions?

Name