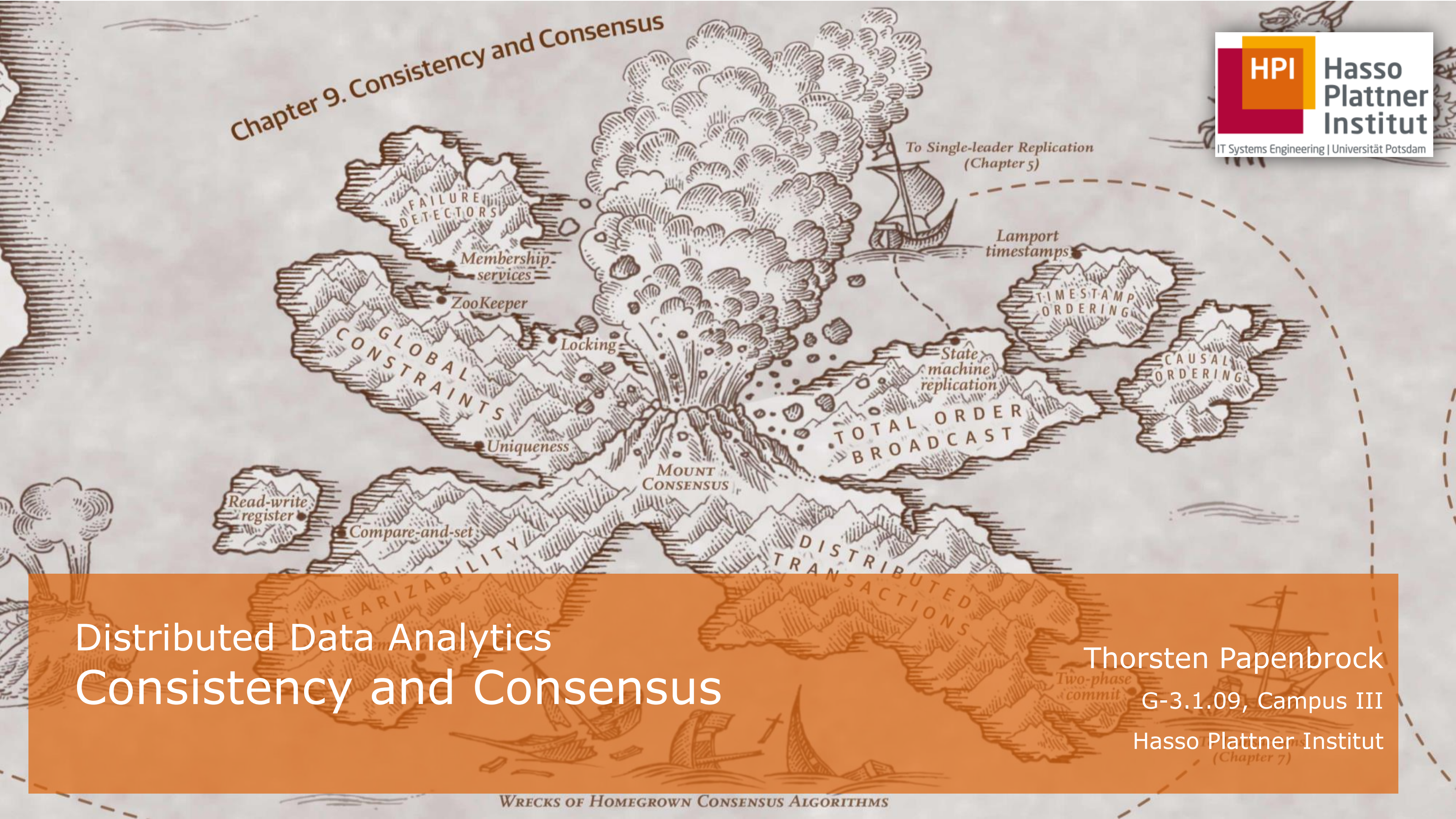


Distributed Data Analytics
Consistency and Consensus

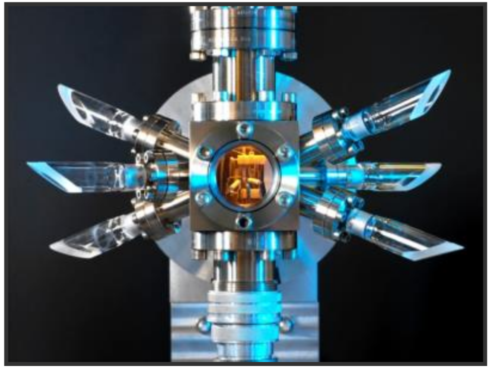
Thorsten Papenbrock
G-3.1.09, Campus III
Hasso Plattner Institut

The Situation

Unreliable Networks

A shark raiding an undersea cable

Unreliable Clocks



An atomic clock with minimum drift

Knowledge, Truth, Lies

Students communi-cating their knowledge

Unreliable Networks

- Messages can be lost, reordered, duplicated, and arbitrarily delayed

Unreliable Clocks

- Time is approximate at best, unsynchronized, and can pause

The Situation

Consensus

A decision carried by all group members although individuals might disagree; usually defined by the majority.

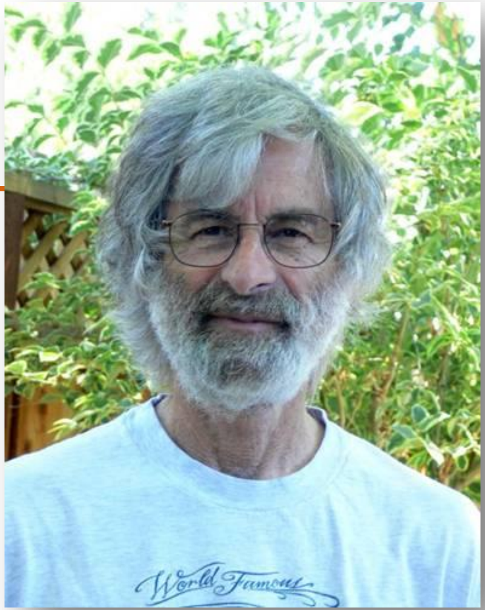
Challenge: find a consensus in spite of unreliable communication

Consistency and Consensus - OLTP Topics?

Why analytical applications might require consistency and consensus

- Non-static data:
  - Distributed analytical queries executed on operational data, i.e., non-warehouse data require a consistent view of the data
- Frameworks for distributed analytics:
  - Distributed analytical queries are usually broken apart so that (intermediate) results must be communicated consistently between the nodes
- Time-related analytics:
  - Distributed analytical queries analyzing volatile data streams or system behavior require a certain consensus on timing and/or ordering of events

Leslie Lamport



Person

Lamport not only defined the "Byzantine problem", he also proposed several solutions

a

ut

Basically serializable writes for distributed systems

Know for

- Byzantine fault tolerance
- Sequential consistency
- Lamport signature
- Atomic Register Hierarchy
- Lamport's bakery algorithm
- Paxos algorithm
- LaTeX

Popular method to construct digital signatures for arbitrary one-way crypto functions

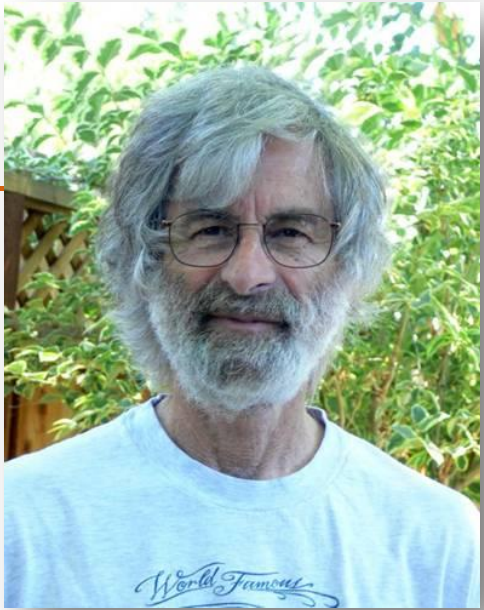
Approach of making register (record, key-value pair, …) appear atomic

Securing a critical section without shared mutexes (using thread IDs)

A fault-tolerant consensus algorithm (based on total order broadcast)

LaTeX !

Leslie Lamport



Person

- Pioneer in consistency and consensus methods for parallel and distributed systems

Known for

- Byzantine fault tolerance
- Sequential consistency
- Lamport signature
- Atomic Register Hierarchy
- Lamport's bakery algorithm
- Paxos algorithm
- LaTeX

Awards

For outstanding papers on the principles of distributed computing

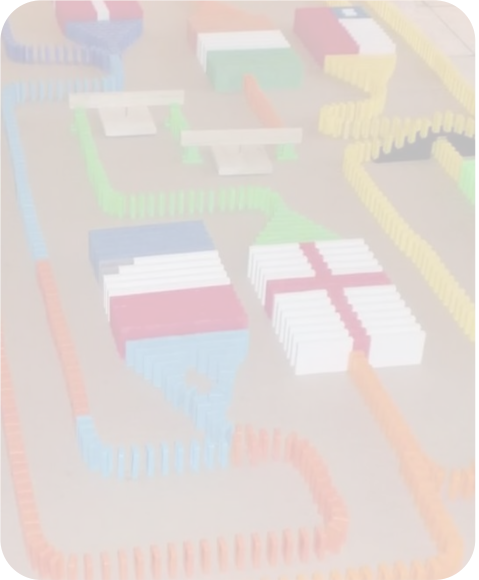
- Dijkstra Prize (2000, 2005, 2014)
- IEEE Emanuel R. Piore Award (2004)
- IEEE John von Neumann Medal (2008)
- ACM Turing Award (2013)
- ACM Fellow (2014)

"Nobel Prize of computing" (highest distinction in computer science)

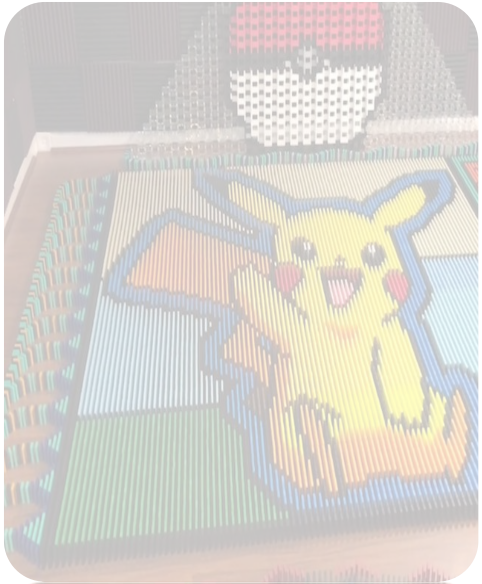
Consistency and Consensus

Linearizability

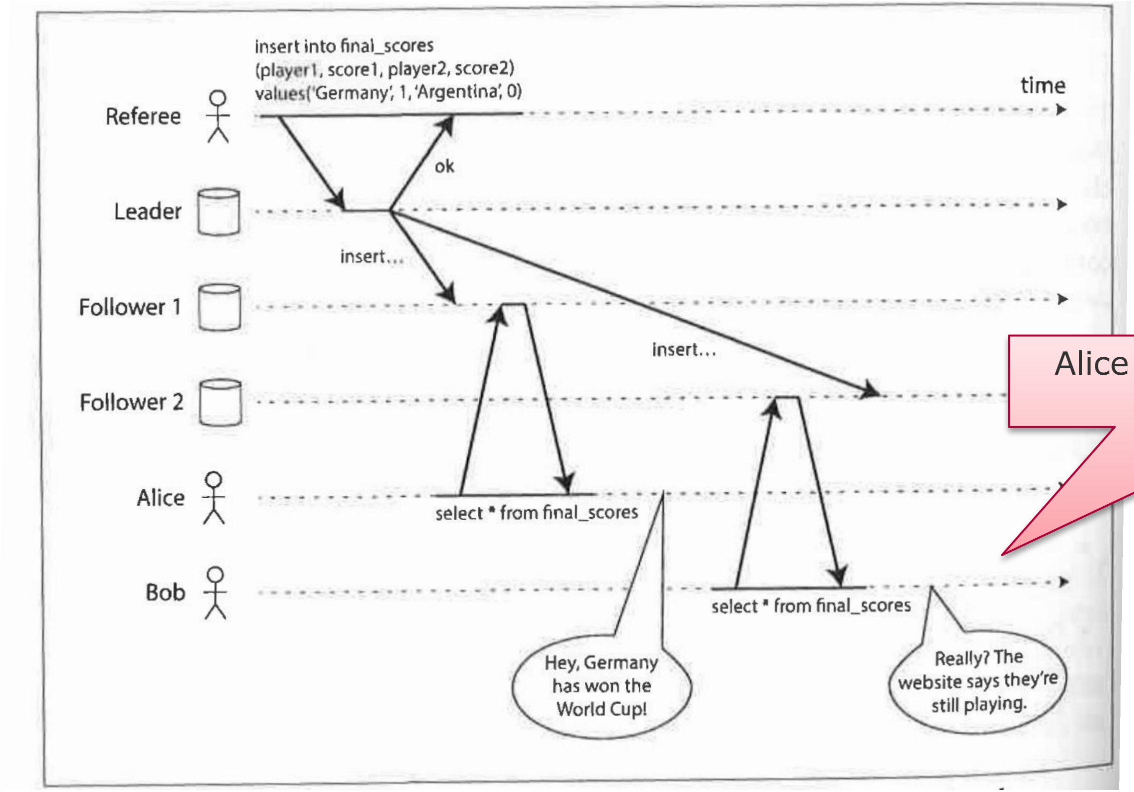
Ordering Guarantees



Consensus



The Problem



Alice and Bob disagree on a value (for some time)

Motivation

Locks and Leaders

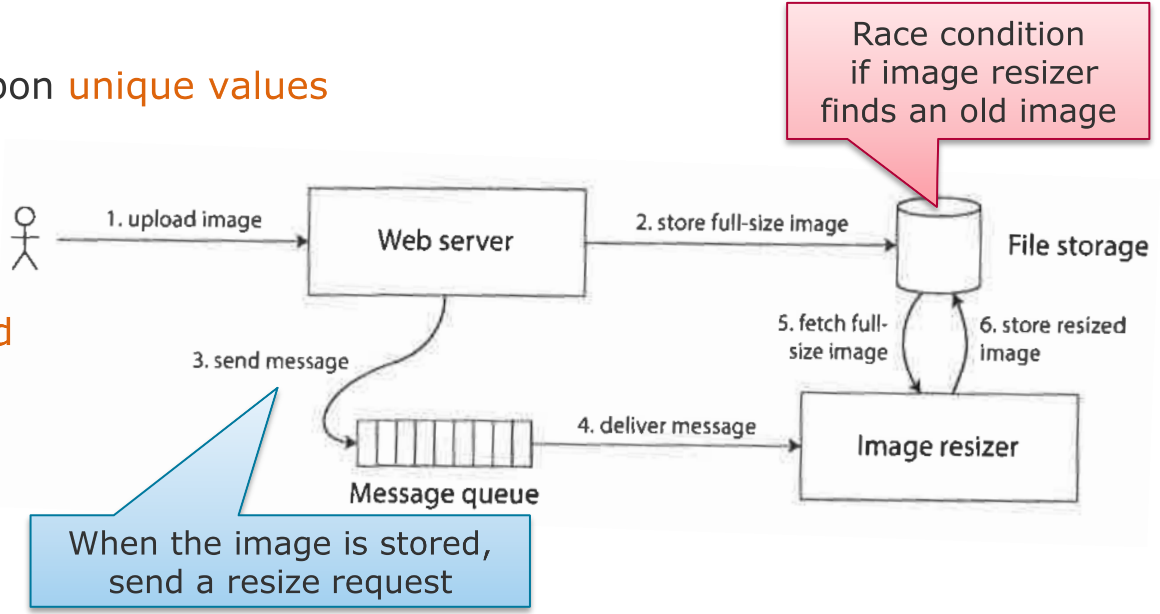
- System must agree upon lock- and leader-assignments
  - Otherwise: locks don’t work / split brain

Uniqueness constraints

- System must know and agree upon unique values
  - Otherwise: duplicate values

Cross-channel timing dependencies

- System must agree upon facts that are also communicated via side channels
  - Otherwise: inconsistent system behavior



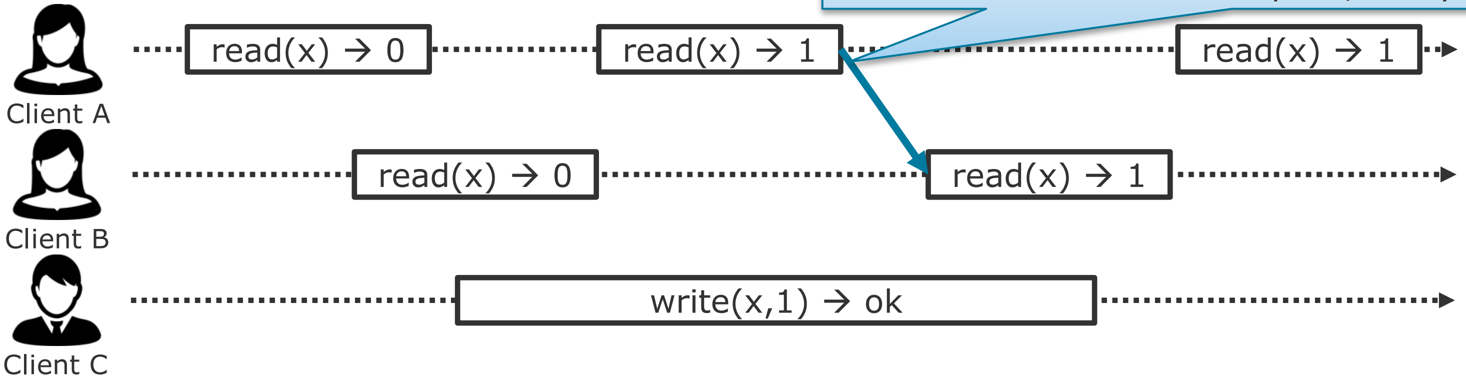
Definition

A linearizable system is 100% consistent w.r.t. the CAP theorem!

Linearizability

- A consistency guarantee of eventual consistent databases stating that a read operation should always return the most recent value of an object although replicas might have older values
- The databases appears as if there is only one copy of the data
- Also known as atomic consistency, strong consistency, immediate consistency, or external consistency

Values must not jump back in time:
If the value is on one replica, everyone should see it!



Linearizable vs. Serializable

Linearizability

- Guarantee for reads and writes to one register (record, key-value pair, …)
- Ensure that the database always returns the newest value from a set of redundant values
- Does not prevent phantom reads or write skew problems

Serializability

- Guarantee for reads and writes of transactions
- Ensure that concurrent transactions have the same effect as some serial execution of these transactions
- Does not ensure the newest values to be read (e.g. see Snapshot Isolation)

Implementation

Single-leader replication

- Run not only all writes but also all reads through the leader; redirect reads to only those replicas that confirmed relevant updates
  - ➢ Leader crashes, unavailability, re-elections, … might break linearizability

Multi-leader replication

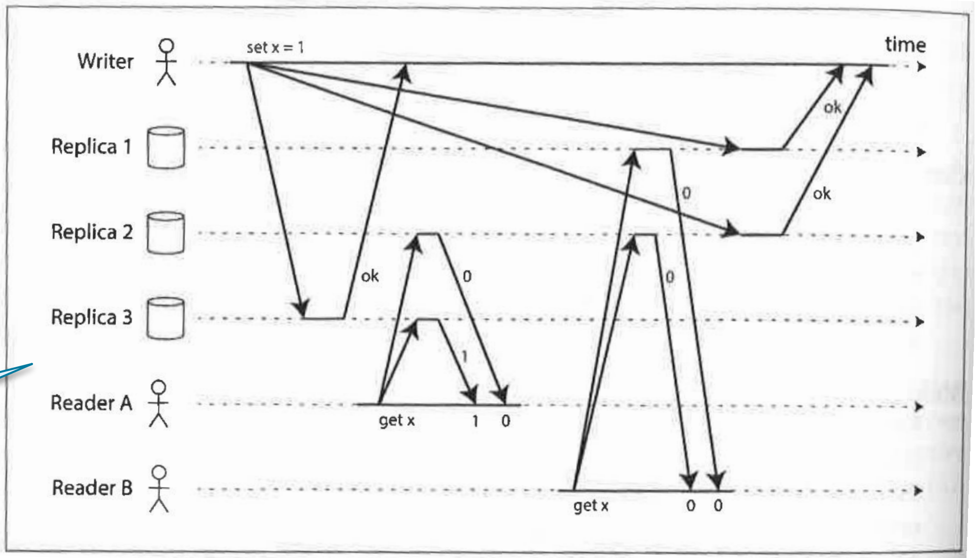
- Not linearizable!

Leaderless replication

This is done anyway

- Quorum read and writes (w + r > n)
  - Ensure new value gets found

Quorums alone do not ensure linearizability



Implementation

Therefore, distributed systems usually do not use linearability for all registers but only for critical, consensus relevant decision (e.g. role assignments)

Single-leader replication

- Run not only all writes but also all reads through the leader; redirect reads to only those replicas that confirmed relevant updates
  - Leader crashes, unavailability, re-elections, … might break linearizability

Multi-leader replication

- Not linearizable!

Linearizability is an expensive consistency guarantee that is dropped by most distributed systems in favor of performance

Leaderless replication

This is done anyway

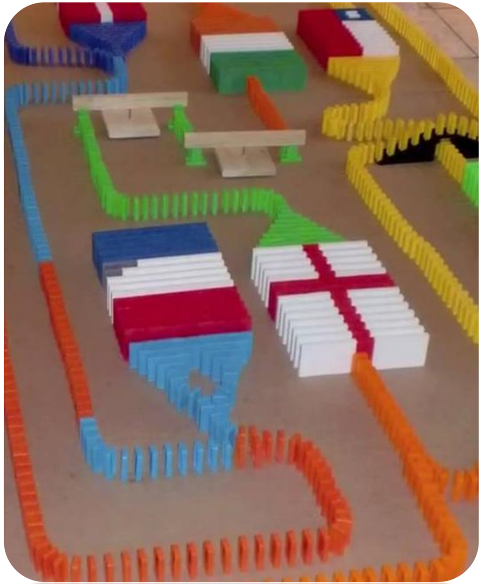
- Use three techniques:
  - Quorum read and writes (w + r > n)
    - Ensure new value gets found
  - Read-repair (write newest value of a read to all replicas with old value)
    - Help updating replicas before returning a value
  - Read before write (read quorum before writing new value)
    - Ensure your write does not conflict with other writes

In this way, other reads either return before you or they find the same result

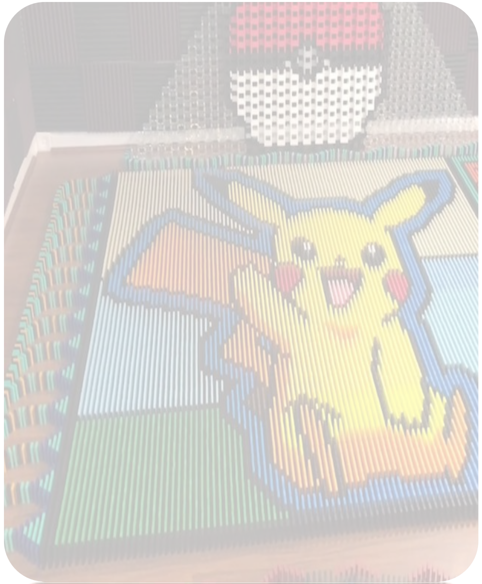
Consistency and Consensus

Linearizability

Ordering Guarantees



Consensus

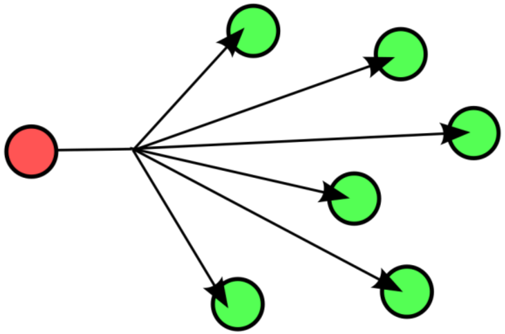


Total Order Broadcast

Total Order Broadcast

- A protocol for message exchange that guarantees:
  1. Reliable delivery:
     - No messages are lost
  2. Totally ordered messages:
     - Messages are delivered to all nodes in the same order
     - Order is not changed retroactively (in contrast to timestamp ordering)
- Any total order broadcast message is delivered (broadcasted) to all nodes
- Implemented in, for instance, ZooKeeper and etcd
- Enables:
  - Consistent, distributed log (ordered messages = log)
  - Lock service implementations for fencing tokens
  - Serializable transactions

Because messages are lost and re-ordered, the protocol must hide these issues!



Total Order Broadcast

Recall: we know how to implement linearizable storage (for single-leader or leaderless replication)

Total Order Broadcast

- Implementation:
  - Assume we have one linearizable register with an integer value supporting atomic increment-and-get (or compare-and-set) operations
  - [Sender] For every message send as total order broadcast:
    1. Increment-and-get the linearizable integer
    2. Attach the integer as sequence number to the message
    3. Send the message to all nodes (resending lost messages)
  - [Receiver] For every message received as total order broadcast:
    1. Check if sequence number is one greater than last received sequence number
    2. Process message if true; otherwise, wait for missing message
       - ➢ This is only possible because there are no sequence gaps!

Ordering Guarantees
Causal Ordering

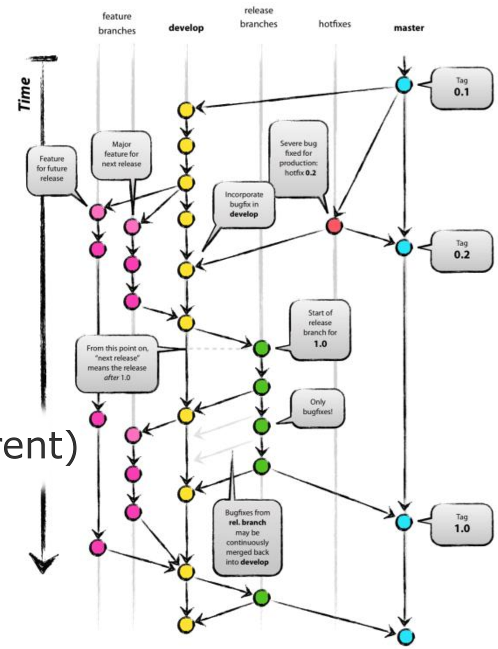
Thinking:
timelines that branch/merge;
events compare only along lines
➢ GIT

Linearizable (and Total Order Broadcast)

- Imposes a total order:
  - All events can be compared
  - For one object, only the newest event is relevant
- Implies causality:
  - A linear order is always also a causal order of the events
- Is expensive

Causal ordering

- Imposes a partial order:
  - Some events are comparable (causal), others are not (concurrent)
  - For many events some partial order is just fine:
    - Order of writes, transactions, side-channel messages, …
- Is cheaper



Causal Consistency

Causal ordering:

- Example: reads and writes in transactional systems
  - Reads and writes are causally unrelated unless they …
    - target the same object or
    - connect through transactions
- A system that guarantees causal ordering is causal consistent
- Snapshot Isolation
  - Reads old values for uncommitted transactions
    - ➢ causal consistency but no linearizability!
  - Ensures that causally related operations are ordered (unrelated operations still occur concurrently)
  - Is expensive, because it not only orders the events for the same object but also for an entire transaction!

Recall Multi-Version Concurrency Control (MVCC) from chapter Transactions

Sequence Number Ordering

One efficient way of tracking orders

Sequence Numbers and Timestamps

- Idea:
  - Label all events with a consecutive number
  - Events labeled w.r.t. comparable numbers are also comparable
- Sequence number:
  - Counter that increments with every event
- Timestamp:
  - Reading from a monotonic/logical clock
- Problem:
  - (Non-linearizable) sequence numbers and (potentially skewed) timestamps are not causally comparable across different nodes
  - See non-linearizable systems, such as multi-leader systems
    - ➢ Solution: Lamport timestamps!

Sequence Number Ordering

Leslie Lamport:
"Time, clocks, and the ordering of events in a distributed system", Communications of the ACM, volume 21, number 7, pages 558-565, 1978

One of the most cited papers in distributed computing!

Lamport timestamps

- Each node has a unique identifier and a counter for processed operations
- Lamport timestamp:
  - A pair (counter, identifier)
  - Globally unique for each event
  - Imposes a total order consistent with causality:
    - Order by counter
    - If counters are equal, use identifier as tie-breaker
- Achieving causal order consistency:
  - Nodes store their current counter c
  - Clients store the max counter m seen so far (send with each event)
  - Nodes increment their counter as c = max(c,m) + 1
    - ➢ Counter moves past some events that happened elsewhere

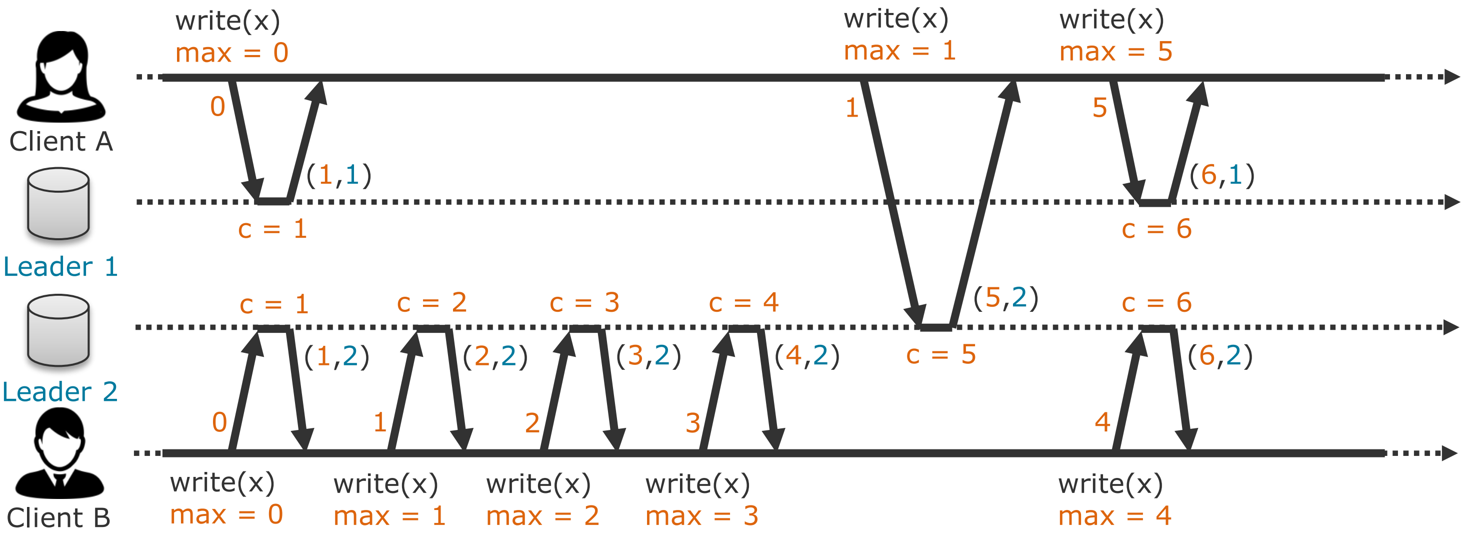
Sequence Number Ordering

Leslie Lamport: *"Time, clocks, and the ordering of events in a distributed system"*, Communications of the ACM, volume 21, number 7, pages 558-565, 1978

Lamport timestamps

- Example:

Although two leaders accept requests in parallel, the timestamps impose a global, causal order

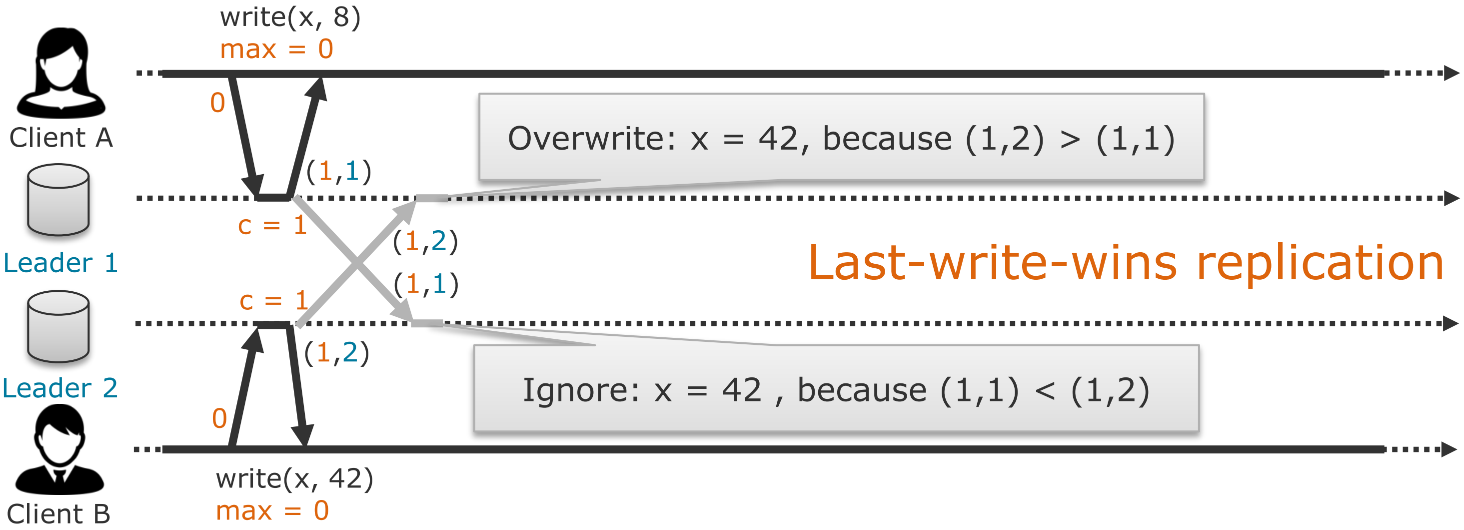


Sequence Number Ordering

Leslie Lamport:
"*Time, clocks, and the ordering of events in a distributed system*", Communications of the ACM, volume 21, number 7, pages 558-565, 1978

Lamport timestamps

- Example:



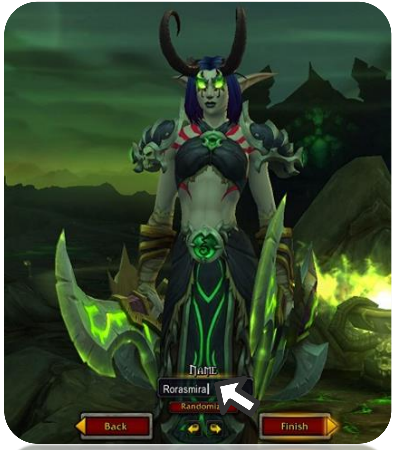
Ordering Guarantees
Sequence Number Ordering

Lamport timestamps

- About the order:
  - Does not capture a notion of time between events
  - Might differ from the real-world time order
  - Works to identify a winner after the fact
    (i.e., the most recent event after all events have been collected)

Usually not an issue

Not ok for locks/uniques/…



Examples for problems:

- Create a new user: assure name is unique before acknowledgement of user creation
- Acquire a role (e.g. leader): assure role is still free before acknowledgement of role assignment
- Buy a product: assure product is still in stock before acknowledgement of purchase

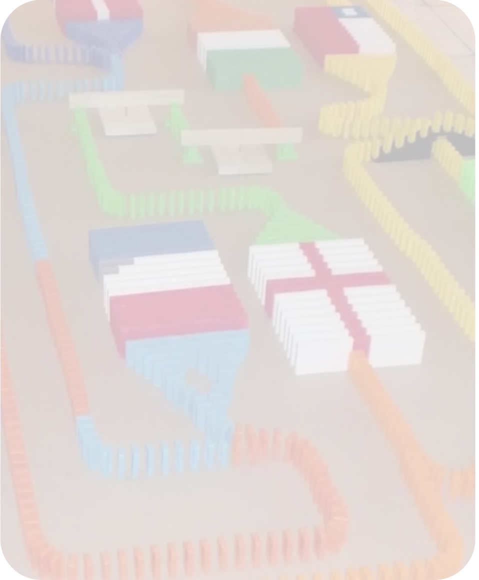
➢ Any form of locking!

Use linearizability / total order broadcast

Consistency and Consensus

Linearizability

Ordering Guarantees



Consensus

Consensus

Consensus

- A decision carried by all group members although individuals might disagree
- Usually defined by the majority
- Challenge:
  - Reach consensus in spite of unreliable communication
- Linearizability, total order broadcast, and consensus are equivalent problems:
  - If a distributed system supports one of them, the others can be achieved through the same protocol

We just did this for "linearizability → total order broadcast"

- Consensus properties:
  - Agreement: No two nodes decide differently
  - Integrity: No node decides twice
  - Validity: Nodes do not decide for a value that has not been proposed
  - Termination: Every non-crashed node makes a decision

i.e. no compromises!

Fault-Tolerant Consensus

Consensus via total order broadcast

- Total order broadcast implies a consensus about the order of messages
- Message order ⟺ several rounds of consensus:
  - Some nodes propose a message to be send next
  - Total order broadcast protocol decides for one message (= consensus)

No (majority) voting in this case

- Example: Locking
  - Multiple nodes want to acquire a lock and send their requests
  - Total order broadcast orders the requests and delivers them to all nodes
  - All nodes then learn from the sequence, which node in fact obtained the lock

i.e. the first node in the sequence

- Consensus properties hold for total order broadcasts:
  - Agreement: All nodes deliver the same order
  - Integrity: Messages are not duplicated
  - Validity: Messages are not corrupted or arbitrarily added
  - Termination: Messages are not lost

Fault-Tolerant Consensus

Consensus via total order broadcast

- Is the most common implementation approach for consensus protocols:
  - Viewstamped Replication [1,2]
  - Paxos [3,4,5]
  - Raft [6,7]
  - Zap [8,9]

[1] B. M. Oki and B. H. Liskov: "Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems," ACM Symposium on Principles of Distributed Computing (PODC), 1988.

[2] B. H. Liskov and J. Cowling: "Viewstamped Replication Revisited," Massachusetts Institute of Technology, Tech Report MIT-CSAIL-TR-2012-021, 2012.

[3] L. Lamport: "The Part-Time Parliament," ACM Transactions on Computer Systems, volume 16, number 2, pages 133–169, 1998.

[4] L. Lamport: "Paxos Made Simple," ACM SIGACT News, volume 32, number 4, pages 51–58, 2001.

[5] T. D. Chandra, R. Griesemer, and J. Redstone: "Paxos Made Live – An Engineering Perspective," ACM Symposium on Principles of Distributed Computing (PODC), 2007.

[6] D. Ongaro and J. K. Ousterhout: "In Search of an Understandable Consensus Algorithm (Extended Version)," USENIX Annual Technical Conference (ATC), 2014.

[7] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft: "Raft Refloated: Do We Have Consensus?," ACM SIGOPS Operating Systems Review, volume 49, number 1, pages 12–21, 2015.

[8] F. P. Junqueira, B. C. Reed, and M. Serafini: "Zab: High-Performance Broadcast for Primary-Backup Systems," IEEE International Conference on Dependable Systems and Networks (DSN), 2011.

[9] A. Medeiros: "ZooKeeper's Atomic Broadcast Protocol: Theory and Practice," Aalto University School of Science, 20, 2012.

Fault-Tolerant Consensus

The leader election problem

= "king", "proposer", …

- Consensus protocols (and linearizability and total order broadcast) usually rely on a leader
- [Problem 1] If the leader dies, a new leader must be elected
  - ➢ But how to get a consensus if the main protocol relies on a leader being present?
- [Solution 1] Actual voting:

  Here: a quorum-based voting protocol; see leaderless replication

  - Initiated when leader is determined dead (e.g. via φ accrual failure detector)
  - All nodes exchange their leader qualification (e.g. IDs, latencies, or resources) with w other nodes
    - Every node tries to identify who is the most qualified leader
    - ➢ The most qualified leader will then be known to w other nodes
  - Any node that "feels" like a leader asks r other nodes who their leader is
    - If none of the r nodes reports a more qualified leader, it is the leader

Recall that $r + w > n$ for n nodes to make vote stable

Fault-Tolerant Consensus

The leader election problem

- Consensus protocols (and linearizability and total order broadcast) usually rely on a leader
- [Problem 2] If the old leader comes back, it might still think it is the leader
  - ➢ How to prevent split brain issues?
- [Solution 2] Epoch numbers:
  - Whenever a leader voting is initiated, all nodes must increment an epoch number
  - An epoch number associates the validity of a leader election with a sequence
  - Before a leader is allowed to decide anything, it must collect votes from a quorum of r nodes (usually a majority)
  - Nodes agree to the quorum, if they do not know a leader with higher epoch
    - ➢ The leader must step down if any node disagrees

epoch number (Zap)
ballot number (Paxos)
term number (Raft)
view number (Viewstamed Replication)

Reliable consensus and leader election protocols are usually implemented in service discovery tools (e.g. ZooKeeper, etcd, Consul, …)

Consensus for Transaction Commits

Two-Phase Commit (2PC)

"Let's be ACID conform!"

- Goal:
  - Ensure that all nodes consistently commit or abort a transaction
  - Consensus = "all agree"
- Requirements:
  - One node that acts as a coordinator for a transaction (e.g. leader)
  - Coordinator must be able to generate unique IDs for transactions
- Steps: (coordinator view)
  - **Writing**: Send the data to all nodes
  - **Phase 1**: Upon global success, send prepare requests to all nodes
  - **Phase 2**: Upon global success, send commit request to all nodes
    - ➢ 2PC transaction commits are blocking operations

Consensus for Transaction Commits

See lecture "Database Systems II" by Prof. Naumann for more details and 3PC

Two-Phase Commit (2PC)

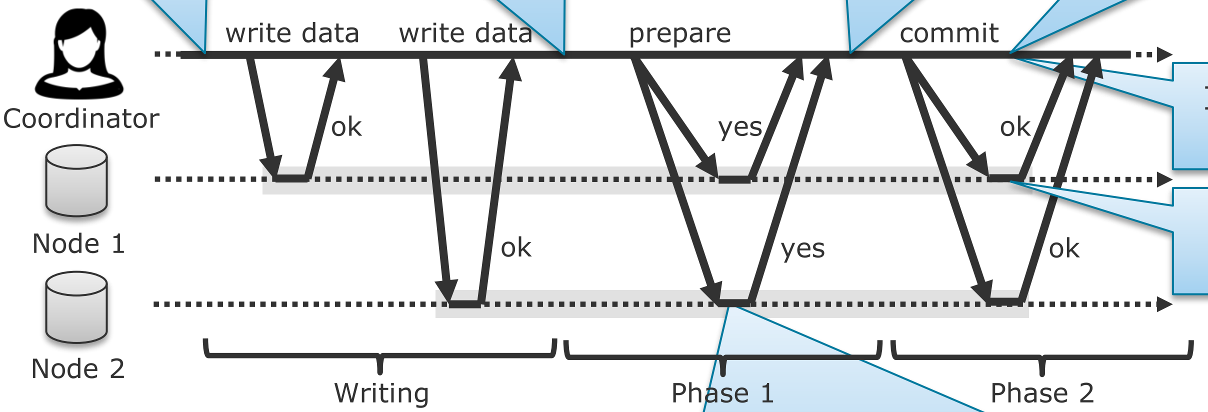
- Steps:

Obtain unique transaction ID

Whenever any response is missing/negative, abort transaction

Make a decision and append it to log on disk
➢ commit point

Keep sending commit messages until all nodes acknowledged



If coordinator crashes: recover and continue sending commits/aborts

If node crashes: recover (and query coordinator)

= locks held by transaction

Get ready to commit (append all writes to log on disk)
➢ crashes, power failures, exhausted memory, … are no excuses later on

Consensus for Transaction Commits

Two-Phase Commit (2PC)

- eXtended Architecture (XA):
  - Standard for implementing 2PC across multiple DBMSs
  - Implemented as C API with bindings to e.g. Java:
    - ➢ Java Transaction API (JTA) supported by various drivers for …
      - databases, i.e., Java Database Connectivity (JDBC) and
      - message brokers, i.e., Java Message Service (JMS)
  - Used in:
    - Databases: PostgreSQL, MySQL, DB2, SQL Server, Oracle, …
    - Message Broker: ActiveMQ, HornetQ, MSMQ, IBM MQ, …

Consensus for Transaction Commits

2PC is no good consensus protocol for **non-transactional votings**

Two-Phase Commit (2PC)

- Evaluation:
  - Expensive: e.g. 2PC is about 10 times slower than single-node transactions in MySQL
  - Blocking: locks are held for long times (indefinitely long if coordinator is lost)
- Extension:
  - Three-Phase-Commit (3PC):
    - Asynchronous, non-blocking transaction commits
    - Automatically choose another leader if the first one failed
      - ➢ Consensus voting inside a consensus protocol!
    - Complex and error prone (leader election = failover = risky)
      - ➢ Merely used in practical implementations

Consensus for Leaderless Cryptocurrencies

Bitcoin

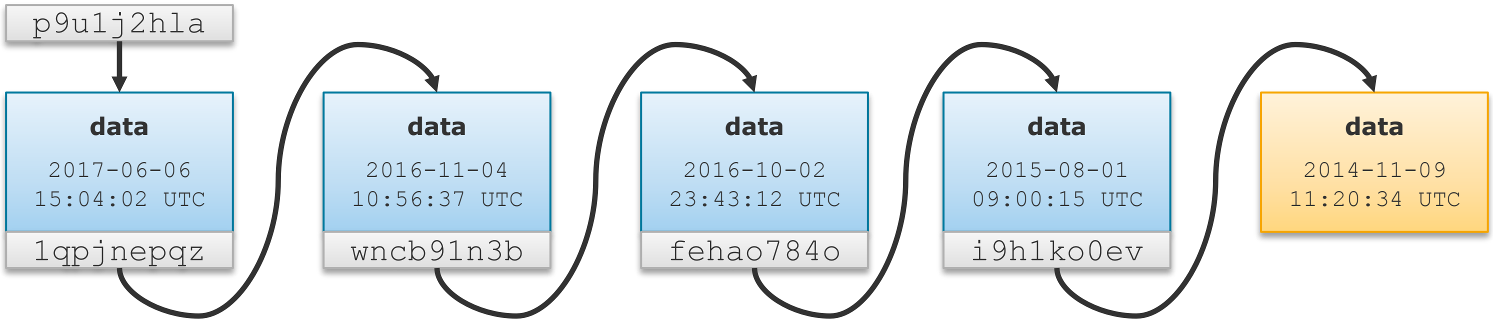
- A decentralized digital cryptocurrency based on an open distributed ledger
- Decentralized:
  - No dedicated authority that validates all transactions
  - ➢ Network validates transactions via consensus (!)
- Crypto:
  - Validated transactions are encrypted
  - Used to ensure consistency and prevent fraud (not to hide values)
- Open distributed ledger:
  - A data structure storing all transactions; replicated on different nodes
  - Nodes can append new transaction but cannot alter passed ones
  - Based on a clever encryption technique
    - ➢ Blockchain

High Byzantine fault tolerance

Consensus for Leaderless Cryptocurrencies

Blockchain

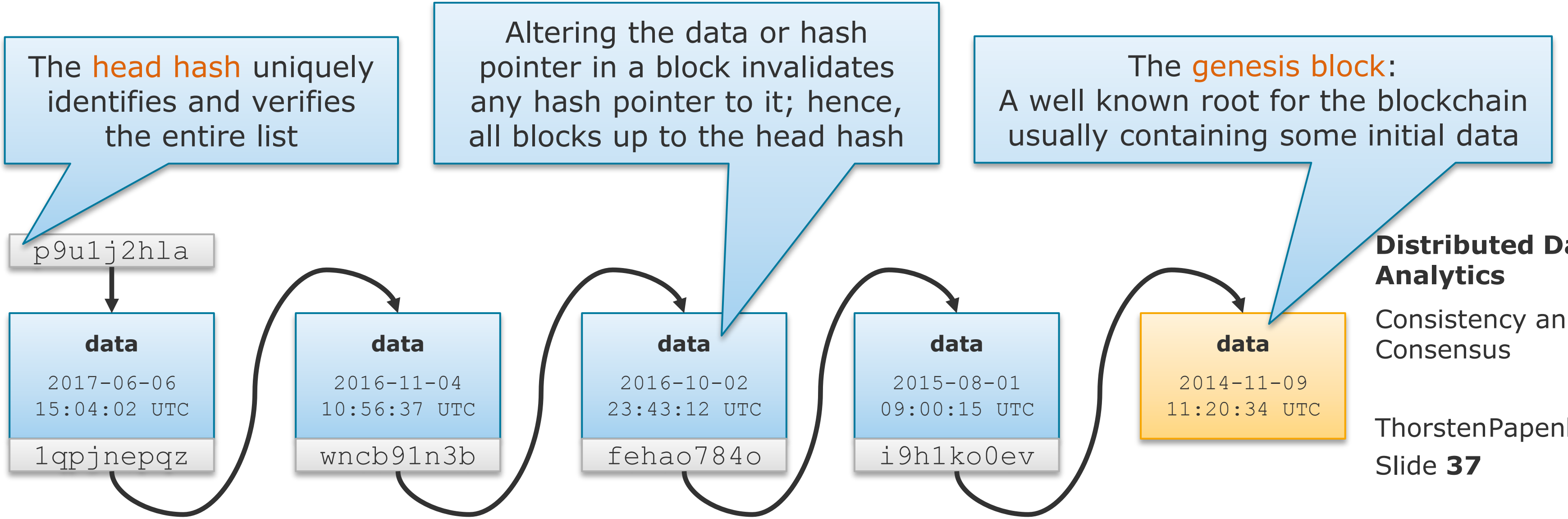
- A single linked list of blocks using hash pointer
- Block:
  - A container for data (transactions or log-entries, messages, measurements, contracts, …)
  - Also stores: timestamp of validation; hash pointer to previous block; nonce
- Hash pointer:
  - A pair of block-pointer (identify the block) and block-hash (verify block content)



Consensus for Leaderless Cryptocurrencies

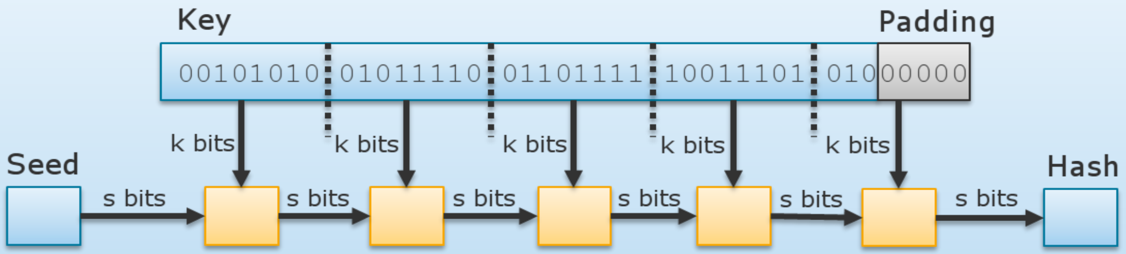
Blockchain

- The "trick":
  - The block-hashes encrypt the entire block with its hash pointer to the previous block

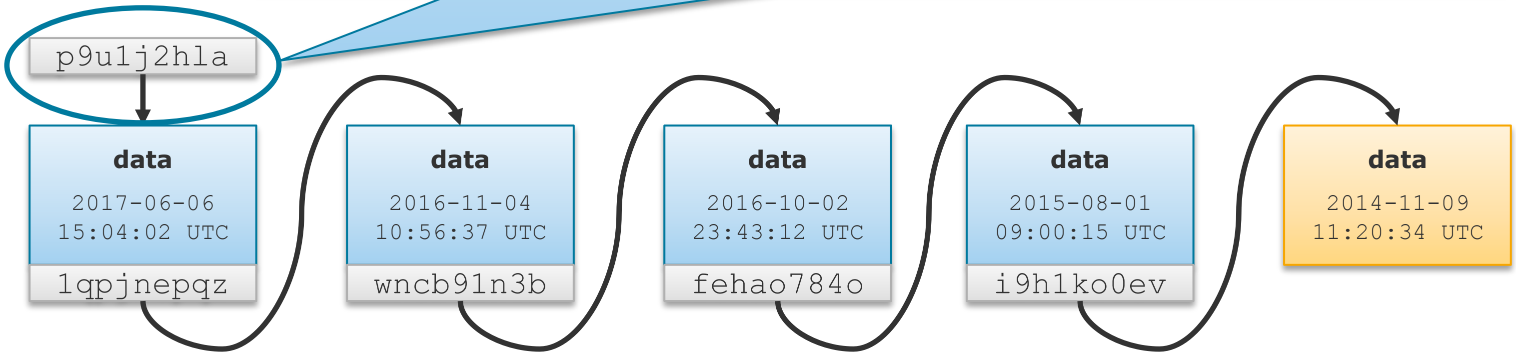


Consensus for Leaderless Cryptocurrencies

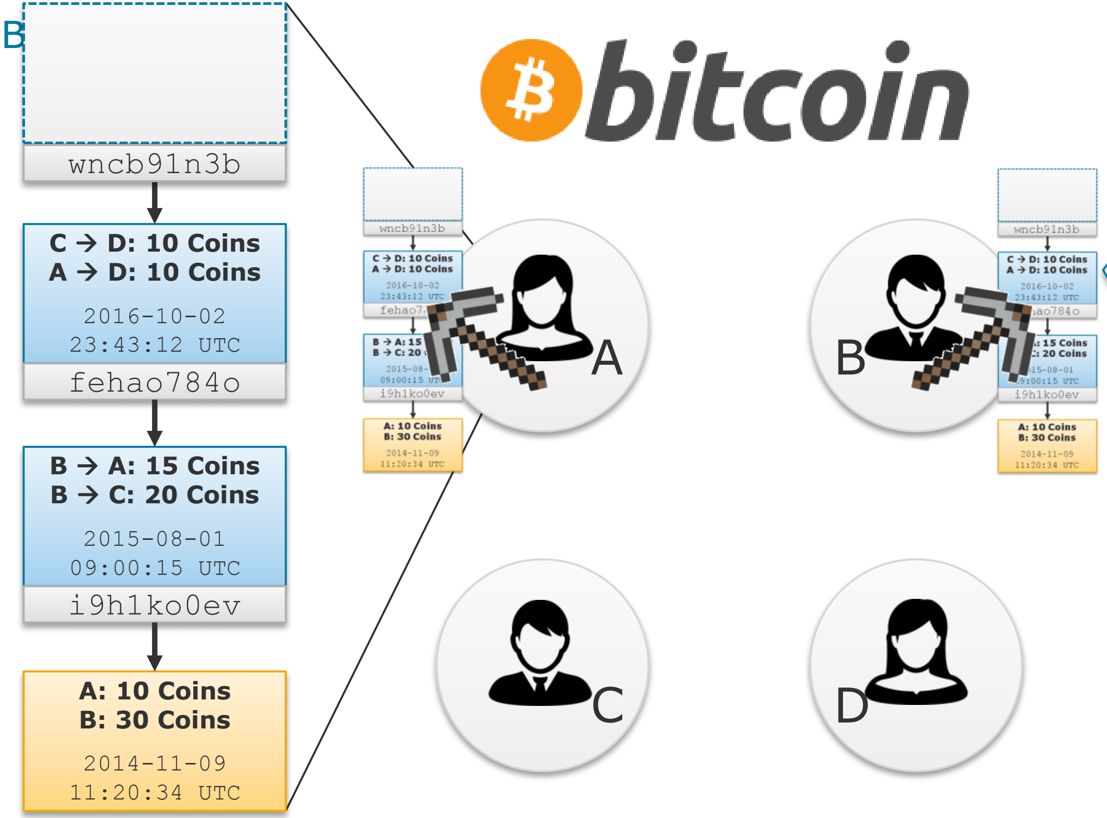
Blockchain



- Calculated via secure Merkle–Damgård hash function
- For instance, SHA-256 in bitcoin



Consensus for Leaderless Cryptocurrencies

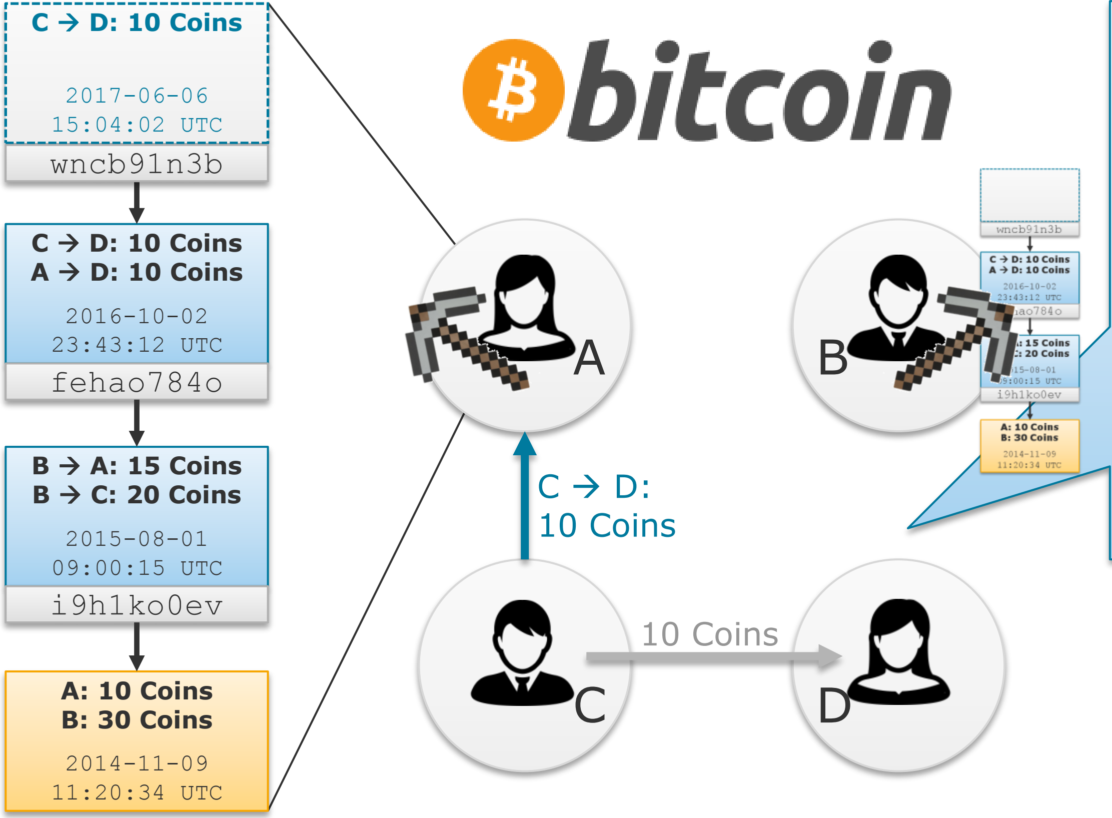


A cluster of nodes that participate in the bitcoin system

Some nodes take the role of mining nodes:

- Store a copy of the open ledger
- Collect and validate transactions
- Try to find a valid nonce

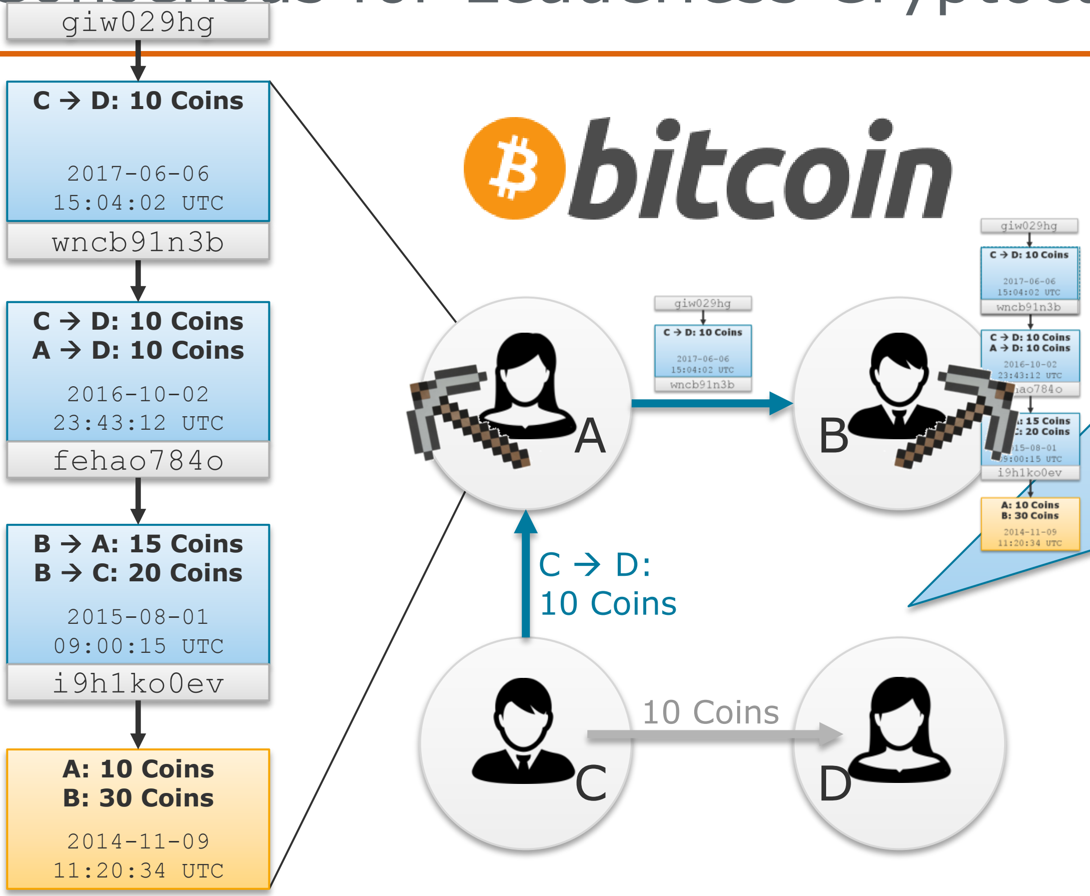
Consensus for Leaderless Cryptocurrencies



Algorithm:

- One node issues a new transaction by broadcasting it to some mining nodes
- Mining nodes:
  - validate the transaction using their open ledger copy
  - write the transaction into their current, non-closed block

Consensus for Leaderless Cryptocurrencies



Algorithm:

- One node issues a new transaction by broadcasting it to some mining nodes
- Mining nodes:
  - validate the transaction using their open ledger copy
  - write the transaction into their current, non-closed block
  - (if possible) close their block with a new hash pointer and broadcast the result

Consensus for Leaderless Cryptocurrencies

Bitcoin

- Mining:
  - To close a block, a miner calculates the hash for:
    data + current time + hash pointer to previous + nonce
  - If the hash fulfills a certain characteristic, e.g., a certain number of leading zeros, the mining was successful and the hash gets accepted

A random value that the miner changes with every hashing attempt

Costs time and electricity!

- ➢ Calculating acceptable hashes is expensive, as it requires many attempts
  - ➢ Miner get rewarded for finding hashes (with currency)
  - ➢ Rewriting, i.e., manipulating parts of the open ledger is expensive!
  - ➢ The deeper in the chain a block is placed, the more secure it is

Consensus for Leaderless Cryptocurrencies

Further reading:
Book: **Bitcoin and Cryptocurrency Technologies**
http://www.the-blockchain.com/docs/Princeton%20Bitcoin%20and%20Cryptocurrency%20Technologies%20Course.pdf

Bitcoin

- Consensus:
  - Blocks sealed with a valid, acceptable hash pointer are commonly agreed facts:
    - If a miner receives such a block it …
      1. tests the acceptance criterion and validates the hash history
      2. removes the agreed transactions from its working block
      3. appends the new block to its local open ledger copy
  - For contradicting blockchains, the longer chain wins
    - ➢ Contents of shorter chains must be re-evaluated and re-packed into new blocks

Disadvantage: Proof of works takes time and resources!

Consensus principle

A node earns the right to dictate consensus decisions by finding extremely rare hashes (= proof of work)

Chapter 9. Consistency and Consensus