Very Large, Distributed Data Volumes

Trustworthy Data Systems

Theodoros Chondrogiannis

October 30, 2025

Overview

- Shared databases
- Blockchain technology
- BlockchainDB

Overview

- Shared databases
- Blockchain technology
- BlockchainDB

Shared Databases

Shipping

orderkey	status

custkey	orderkey	product

Customer

Supplier

Shared Databases

Shipping

orderkey	status
100	new (WF)

put(WF, {Sjokoklem, 100})

custkey	orderkey	product
WF	100	Sjokoklem

Customer

Supplier

Shared Databases

Shipping

orderkey	status
100	new (WF)

get(WF) → ({100, Sjokoklem})

custkey	orderkey	product
WF	100	Sjokoklem

Customer

Supplier

Shared Databases

Shipping

orderkey	status
100	new (WF) ready (Lindt)

custkey	orderkey	product
WF	100	Sjokoklem

put(100, ‘ready’)

Customer

Supplier

Shared Databases

Shipping

get(100) → (‘ready’)

orderkey	status
100	new (WF) ready (Lindt)

custkey	orderkey	product
WF	100	Sjokoklem

Customer

Supplier

Shared Databases

Shipping

put(100) → (‘delivered’)

orderkey	status
100	new (WF) ready (Lindt) delivered (FedEx)

custkey	orderkey	product
WF	100	Sjokoklem

Customer

Supplier

Shared Databases

Shipping

Customer

Supplier

Shared Databases

Shipping

orderkey	status
100	new (WF) ready (Lindt) **lost (FedEx)**

edit(100, ‘lost’)

custkey	orderkey	product
WF	100	Sjokoklem

Customer

Supplier

Shared Databases

Shipping

orderkey	status
100	new (WF) ready (Lindt) **lost (FedEx)**

custkey	orderkey	product
WF	100	Sjokoklem

get(100) → (‘lost’)

Customer

Supplier

Shared Databases

- In cases of an error or malicious intent, an investigation is necessary
- However, every company keeps separate logs
  - Everyone can claim their side is the correct one
    - WholeFoods can claim FedEx lost the order by showing the database
    - FedEx can claim the order was delivered
- In practice, there is no way to find out who tells the truth
- A ledger that keeps track of all changes is needed
  - That’s were blockchain comes into play

Overview

- Shared databases
- Blockchain technology
- BlockchainDB

History: Bitcoin

- Bitcoin: A Peer-to-Peer Electronic Cash System
  - Cryptocurrency and payment system
  - Blockchain is the infrastructure

Satoshi Nakamoto (pseudo)
Oct. 31, 2008 (Halloween)

- Since then
  - Many blockchains: Etherum in 2013, Ripple in 2014, etc.
  - Increasing use for high-risk investment
    - Initial Coin Offerings
  - But also in fraudulent or illegal activities !
    - Scam, purchase on the dark web, money laundering, tax evasion, …
  - Warnings from market authorities and beginning of regulation

Blockchain Definition

- A distributed ledger
  - Shared by all participants
    - Replicated
  - Decentralized
  - Append-only
    - No update, no delete
  - Distributed transaction validation
    - Consensus
  - Unfalsifiable, verifiable

Blockchain Promises

- Increased trust in value exchange
  - Trust the data, not the participants
- No single point of failure
  - Increased security
- Efficient, consistent transactions between participants
  - Faster and cheaper than relying on a long chain of intermediaries, with incompatible systems and rules

Public versus Private Blockchain

- Public blockchain
  - Open P2P network
    - Participants can join and leave without notification
  - Anonymous, untrusted participants
  - Large-scale distributed ledger
- Private blockchain
  - Closed permissioned network
  - Identified, trusted participants
  - Regulated control
  - Small to medium-scale distributed ledger

Blockchain Concepts

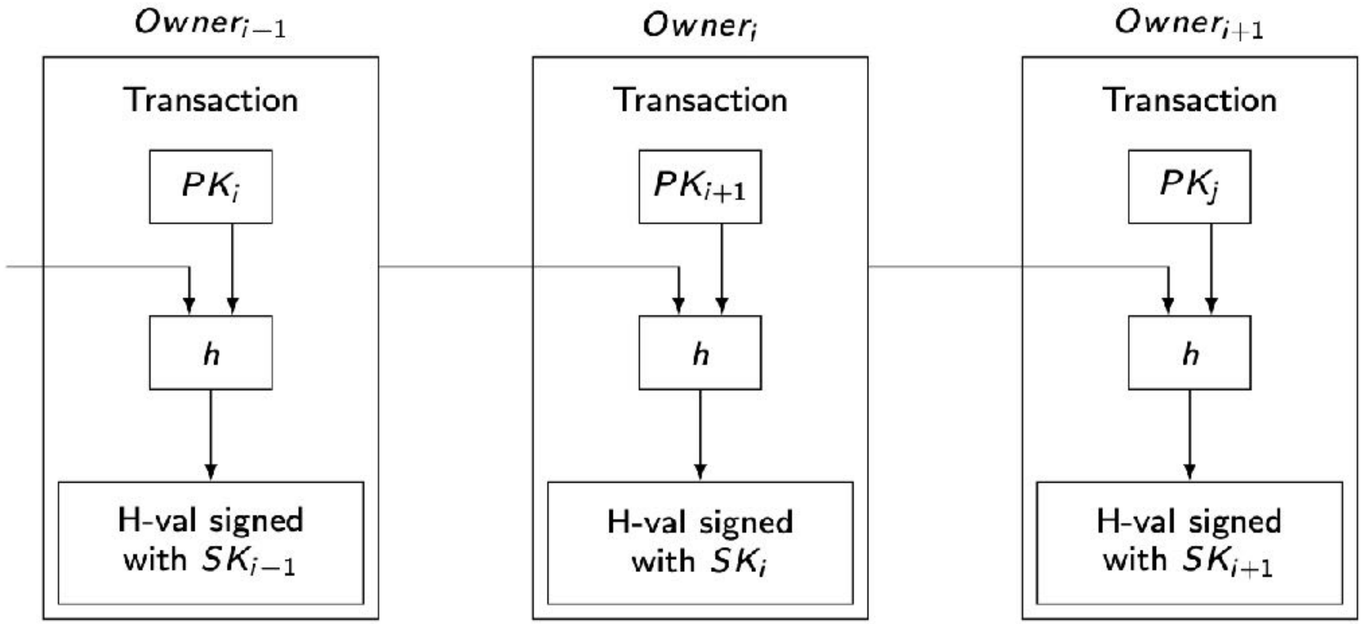
- Blockchain
  - An *immutable* distributed database, i.e. a log of blocks, which are linked and replicated on *full nodes*
- A block
  - Digital container for transactions, contracts, property titles, etc.
  - Transactions are secured using public key encryption
  - The code of each new block is built on that of the preceding block
    - Guarantees that it cannot be changed or tampered
- The blockchain is viewed by all participants
  - Privacy: users are pseudonymized

Blockchain Protocol (Nakamoto 2008)

0. Initialization (of a *full node*)
   – Synchronization with the network to obtain the blockchain
1. Two users agree on a transaction
   – Information exchange: wallet addresses, public keys, …
2. Grouping with other transactions in a block and validation of the block (and of the transactions)
3. Addition of the validated block in the blockchain and replication in the P2P network
4. Transaction confirmation

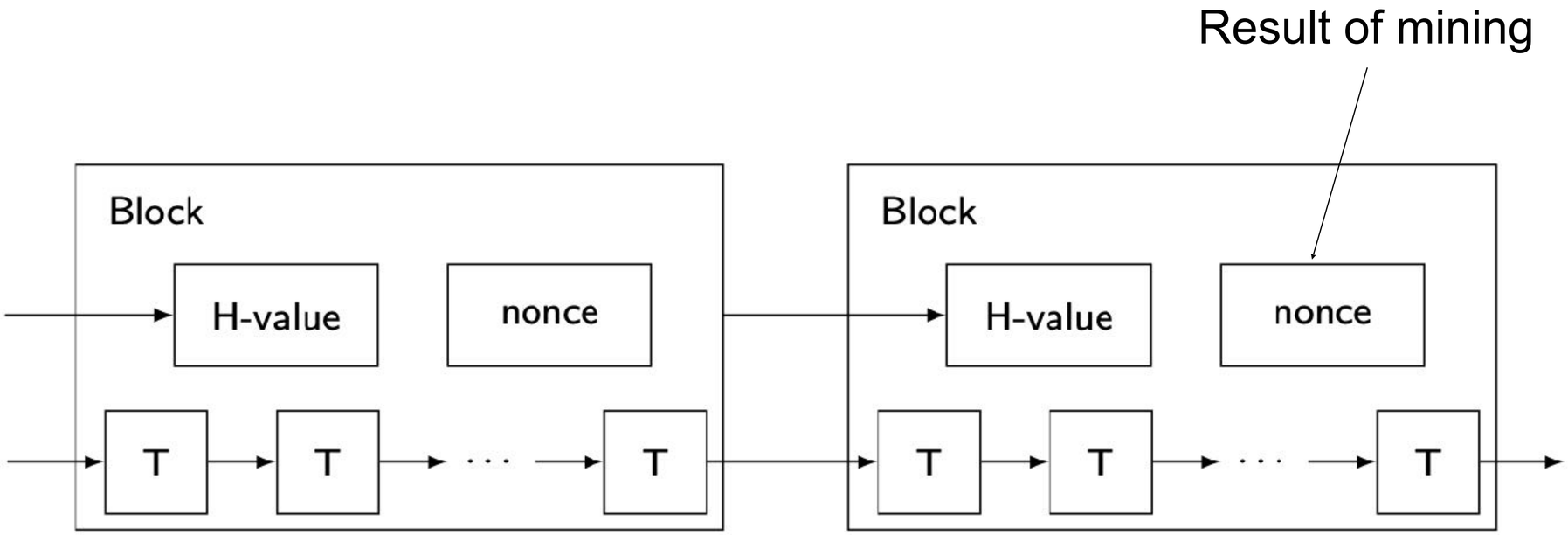
Transaction

- The owner signs the transaction by
  1. Creating a hash value of
     - The previous transaction
     - And the public key (PK) of the next owner
  2. Signing it with its secret key (SK)



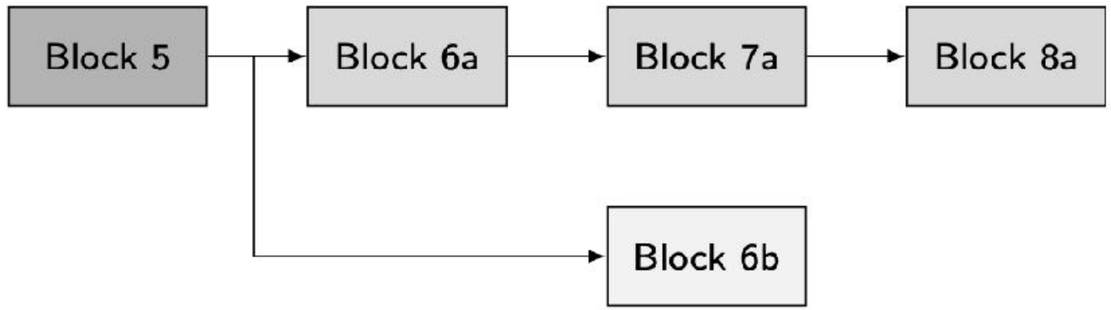
Block Management

- Transactions are placed into blocks, validated (by checking inputs/outputs, etc.) and linked by their addresses
  - Size of a bitcoin block = 1 Megabyte



Validation by the Network

- Each block is validated by network nodes, the *miners*, by a consensus protocol
- Problem: accidental fork
  - As different blocks are validated in parallel, one node can see several candidate chains at any time
  - Solution: longest chain rule



Intentional Fork

- Main reasons
  - To add new features to the blockchain (protocol changes) => new software
  - To reverse the effects of hacking or catastrophic bugs
- Soft versus hard fork
  - Soft fork: backward compatible
    - The old software recognizes blocks created with new rules as valid
    - Makes it easy for attackers
  - Hard fork
    - The old software recognizes blocks created with new rules as invalid

Consensus Protocol: *mining*

- To validate a block, miner nodes compete (as in a lottery) to produce a *nonce* (number used once)
  - One of the first competing solutions is selected, e.g. the one that includes the largest number of transactions
  - The winner miner is paid, e.g. 12.5 bitcoins today (originally 50)
  - This increases the money supply

Consensus Protocol: *mining*

- Mining is designed to be difficult
  - The more mining power the network has, the harder it is to compute the nonce
  - This allows controlling the injection of new blocks ("inflation") in the system, on avg. 1 block every 10mn
  - Advantages powerful nodes

Mining Difficulty : Proof of Work (PoW)

- PoW
  - A piece of data that is difficult to calculate but easy to verify
  - First proposed to prevent DoS attacks
- Hashcash PoW
  - Computed by each miner to produce the nonce
  - Goal: produce a value v s.t. $h(f(block, v)) < T$ where
    - h is the SHA-256 hash function
    - T is a target value shared by all nodes that reflects the network size
    - f is a function that combines v with information in the block
    - v is a 256-bit number starting with n zero bits

The 51% Attack

- Also called Goldfinger attack
  - Enables the attacker to invalidate valid transactions and double spend funds
- How
  - By holding more than 50% of the total computing power for mining
- Solution: monitoring by the community
  - In January 2014, Ghash.io reached 42%, then dropped to 9% after the Bitcoin community alert

Transaction Confirmation

- A provisionally validated transaction in a candidate block ensures that it has been verified and is viable
- Each new block accepted in the chain after the validation of the transaction is considered as a confirmation
  - A transaction is considered mature after 6 confirmations (1 hour on average)
  - New bitcoins (mining products) are only valid after 120 confirmations, to avoid the 51% attack

Public Blockchain Limitations

- Complexity and low scalability
  - Difficult evolution of operating rules
  - Increasing chain size
  - Low number of transactions per second (TPS)
    - 5-7 TPS for Bitcoin versus 25K TPS for VISA
  - Unpredictable duration of transactions, from minutes to days

Public Blockchain Limitations

- Cost
  - High energy consumption
  - Favors concentration of miners
- Users are pseudonymized, not anonymized
  - Making a transaction with a user reveals all its other transactions
- Lack of control and regulation
  - Hard for states to watch and tax transactions

Blockchain 2.0

- Evolution of Paradigm
- Beyond Bitcoin and other cryptocurrencies
  - Recording and exchange of assets without powerful intermediaries
  - Example: smart contracts
- Positioning in the internet
  - TCP/IP: the communication protocol
  - Blockchain: the value-exchange protocol?

Blockchain 2.0 Technology

- Programmable blockchain, e.g. Etherum
  - Allows application developpers to build APIs on the Blockchain protocol
    - APIs to allocate digital resources (bandwidth, storage, etc.) to the connected devices, e.g. FileCoin
    - Micropayment APIs tailored to the type of transaction (e.g. tipping a blog versus tipping a car share driver)
- Private blockchain
  - Efficient transaction validation since participants are trusted
    - No need to produce a PoW
  - Efficient management, e.g. in the cloud

Blockchain 2.0 Apps

- Critical characteristics of the applications
  - Asset and value are exchanged (transactions)
  - Multiple participants, unknown to each other
  - Trust is critical
- Top use cases
  - Financial services, micropayments
  - Digital rights using smart contracts
  - Digital identity
  - Supply chain management
  - Internet of Things (IoT)

Overview

- Shared databases
- Blockchain technology
- BlockchainDB

Shared Databases

Shipping

Customer

Supplier

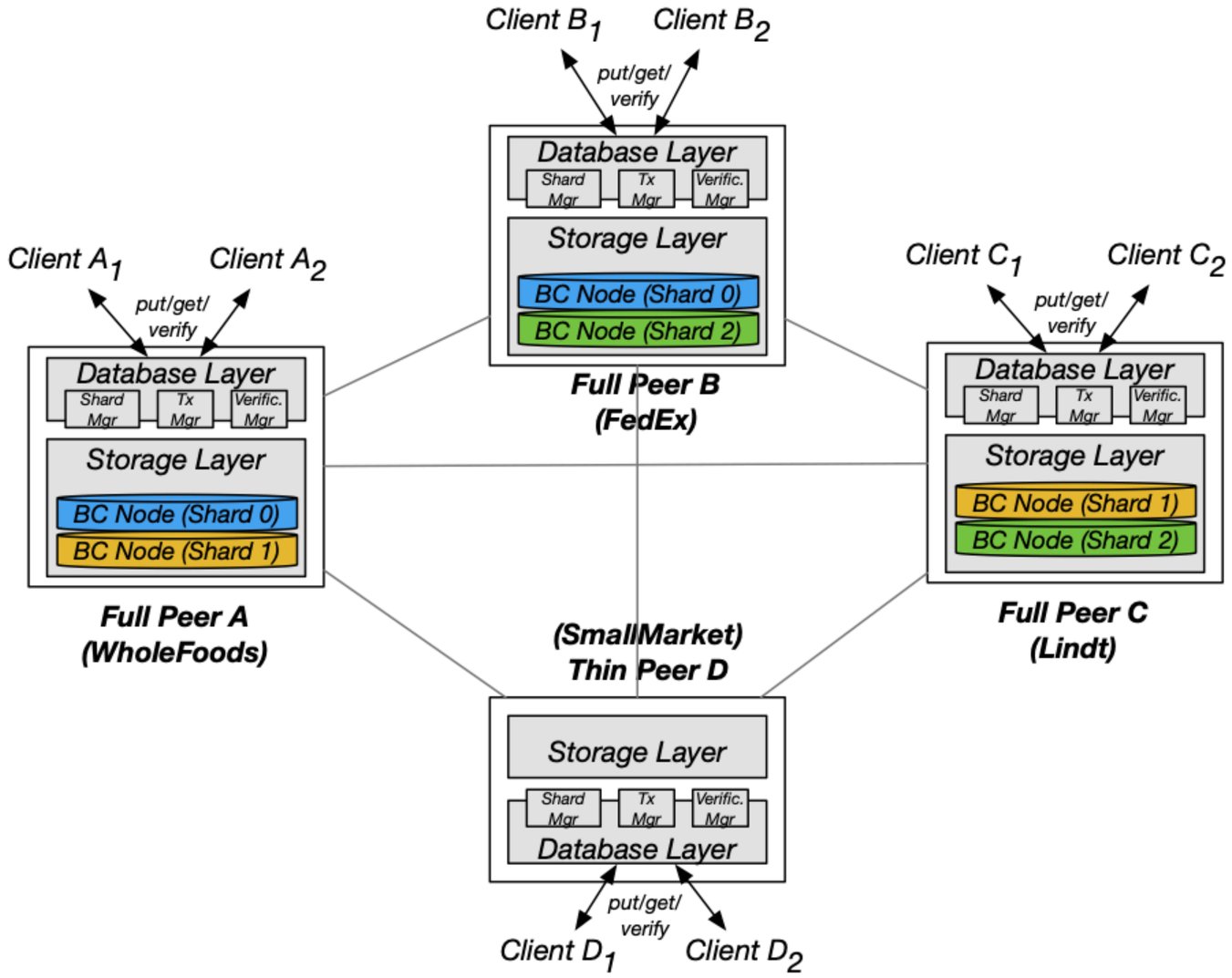
Blockchain Guarantees

- The data in each peer is stored in a tamper-proof log
  - Any data modification of the log by any potentially malicious party could be detected since the cryptographic hashes used in the blockchain would not be valid anymore
- All parties read only from their local copy of the database
  - spurious reads (that are a problem if data needs to be read form a remote party) can be avoided
- Problem: performance

BlockchainDB

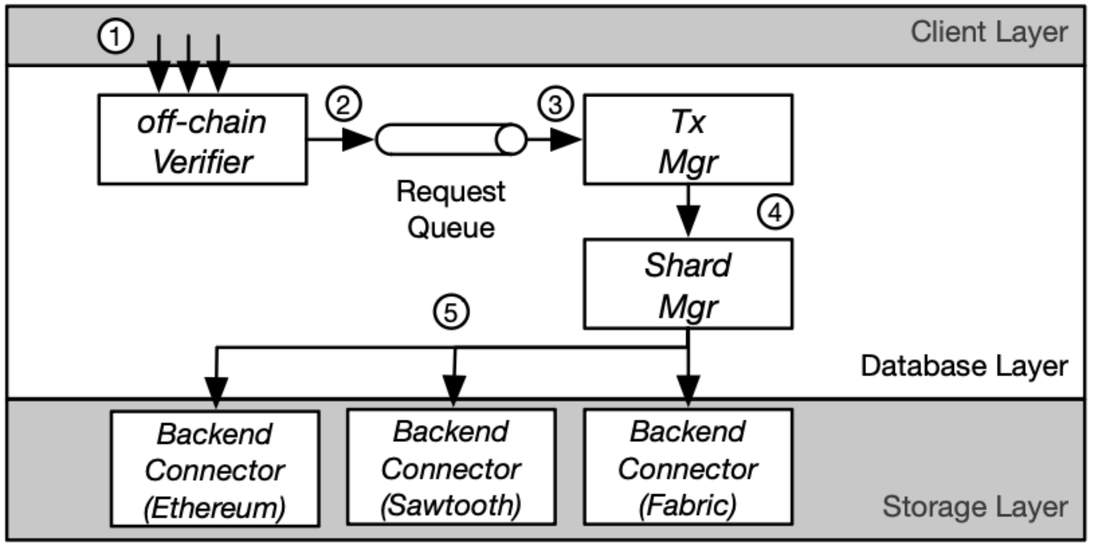
- Main idea
  - Implement a database layer on top of an existing blockchain
  - Simple-to-use abstraction with a put/get
  - stores all data in its storage layer that relies on blockchains
- Goals
  - Provide all the benefits of the blockchain along with high performance
- How does BlockchainDB achieve these goals
  - Partitioning and partial replication
  - Query interface and consistency

BlockchainDB Architecture



BlockchainDB Components

- Client
  - Clients interact with a shared table via their own BlockchainDB peer
- Database layer
  - Mainly responsible to execute the put/get calls from the clients
- Storage layer
  - A persistent auditable storage backend based on existing blockchain systems



Trust Assumptions

- Clients that connect to a BlockchainDB peer trust their local database and storage layer
- A BlockchainDB peer is allowed to perform verification on behalf of all locally connected clients
- A BlockchainDB peer can trust the data that is written to or read from a local shard without verification
- If the majority of the peers that keep a copy of a shard is not malicious, then a client can trust all puts/gets once verified

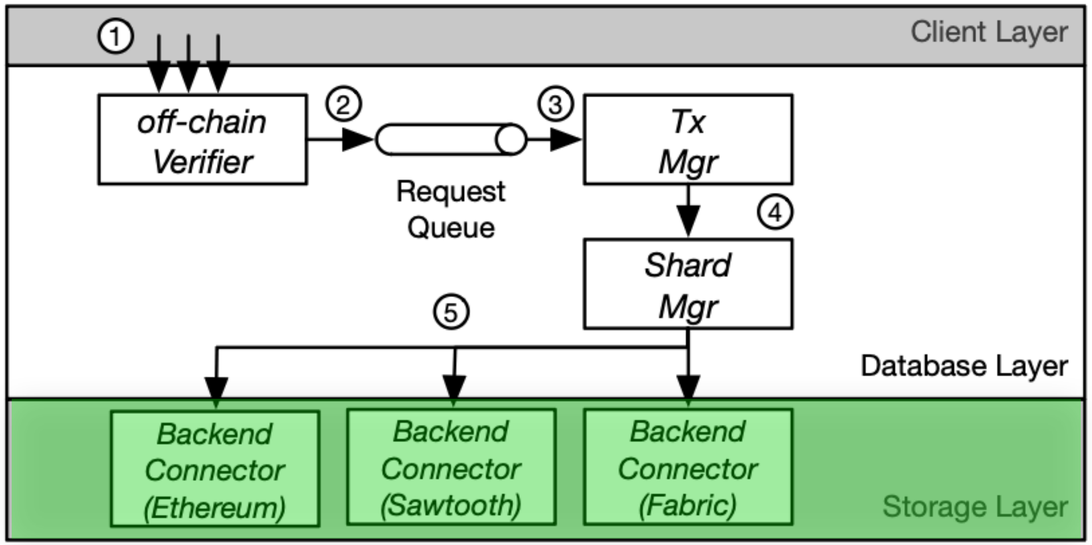
Potential Threats

- The drop of a put operation that needs to write data to a re mote peer
- Spurious/fake data returned for any get operation that needs to read data from a remote peer

These attacks are handles by the verification process

Storage Layer

- Storage interface
- Backend connector

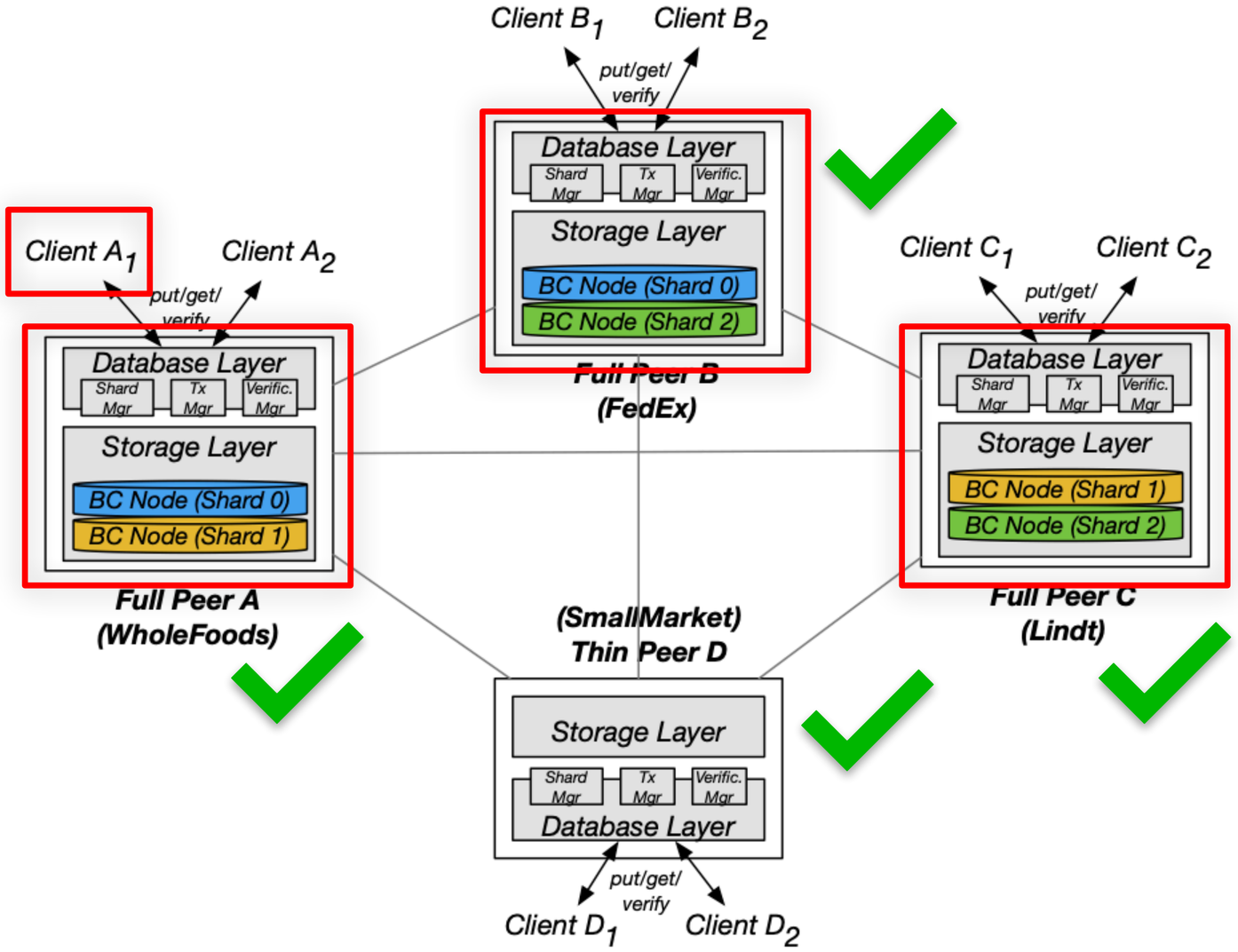


Shared Tables

- A new table is defined using a key/value data model
  - the key is the primary key
  - the value represents the payload of a tuple
- Sharding information
  - Contains values for the parameters such as the number of shards or the replication factor and the allocation

Creating a New Shared Table

1. A client proposes the creation of a table
2. The peer that receives the request handles the process
3. Shards of the new table are distributed to peers
4. All peers confirm that they agree with the table



NTNU

Storage Interface

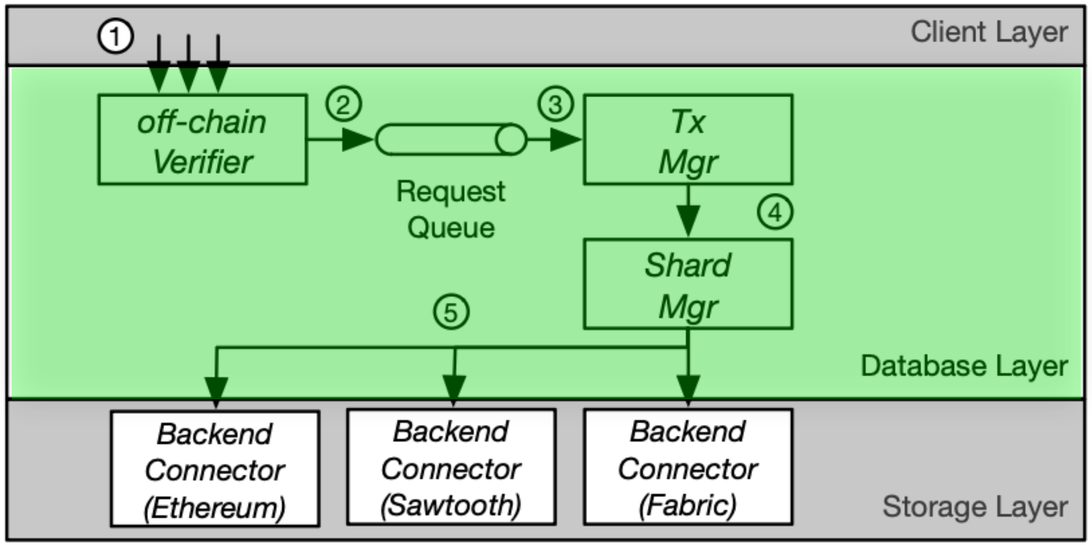
- Main methods
  - $read(s, k) \rightarrow v$
    - Reads a value v for a given shard s and a key k
  - $write\text{-}async(s, k, v) \rightarrow tx\text{-}id$
    - Write a value v with key k into shard s
  - $check\text{-}tx\text{-}status(s, tx\text{-}id)) \rightarrow TX\text{-}STATUS$
    - Check the status of a write-async operation, i.e., $COMMITED, ABORTED, PENDING$
  - $get\text{-}writeset(s, e)) \rightarrow ws$
    - Returns all writes that were executed on shard s in epoch e

Backend Connector

- Methods
  - *read-blockchain*
  - *write-blockchain*
- The backend connector implements the first three methods of the interface and handles the off-the-self blockchain

Database Layer

- Components
  - Shard manager
    - Distributes data to peers
  - Transaction manager
    - Facilitates consistency
  - Off-chain verifier

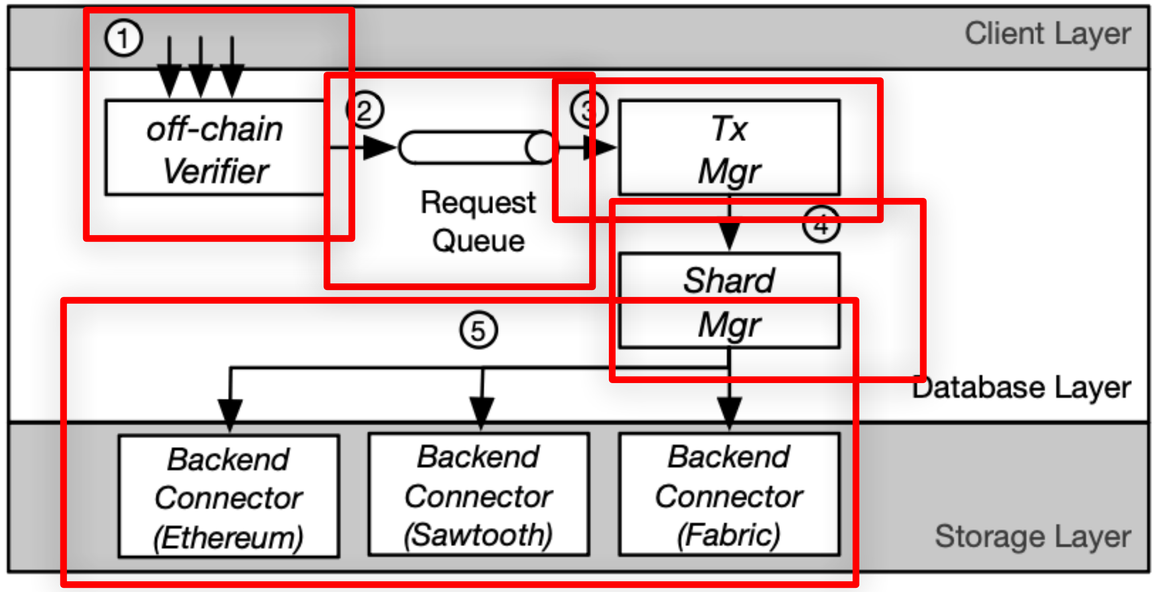


Query Interface

- Methods
  - $get(t, k) \rightarrow v$
    - Returns all attributes of the row in table t that has the key k
  - $put(t, k, v)$
    - Inserts a new row in table t that has key k and value v
  - $verify() \rightarrow bool$
    - For $put(t, k, v)$, it means whether or not the put was actually committed in the storage layer
    - For a $get(t, k)$ it is checked whether or not the returned value was correct or a spurious value.

Query Execution

1. A request arrives from the client in the database layer and is received by the off-chain verifier
2. The verifier then forwards the request to an internal queue
3. The transaction manager polls the request and processes it
4. A put/get-request is created and is forwarded to the ShardManager
5. The backend connector accesses the blockchain that stores the data



Consistency

- The property that a database transaction must only change data in allowed ways, ensuring all data is valid, accurate, and uniform according to all defined rules
- Consistency model
  - Strict consistency
  - Sequential consistency
  - Eventual consistency
  - … and more

Consistency

- The property that a database transaction must only change data in allowed ways, ensuring all data is valid, accurate, and uniform according to all defined rules
- Consistency model
  - Strict consistency
  - **Sequential consistency**
  - **Eventual consistency**
  - … and more

Consistency

- Sequential consistency
  - Implies that operations appear to take place in some total order, and that that order is consistent with the order of operations on each individual process
  - Implementation in BlockchainDB
    - Monitor all pending writes in the database layer
    - Lazy writing
    - The blockchain network provides a global order of all writes

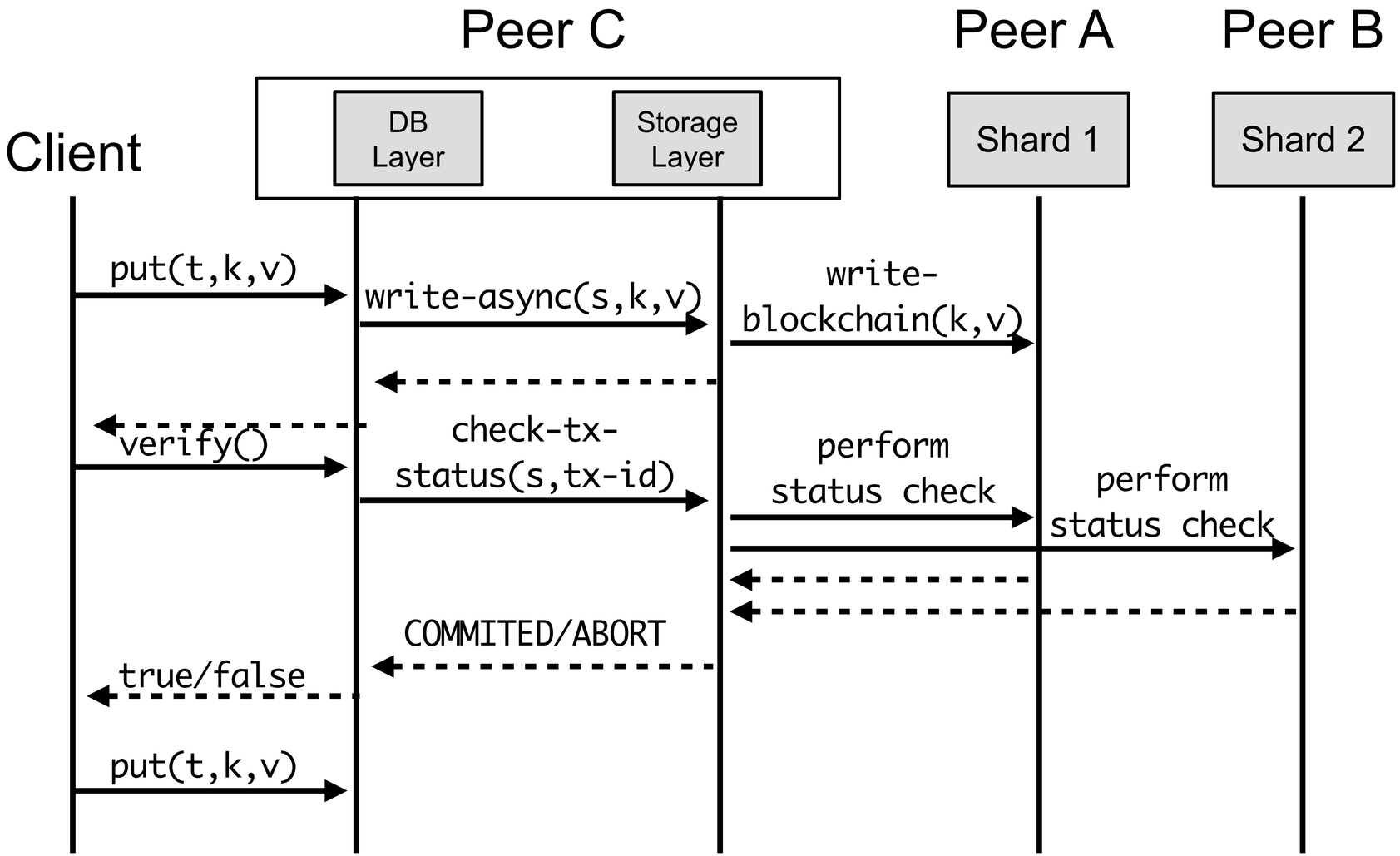
Consistency

- Eventual consistency
  - Ensures that if no new updates are made to a given data item, eventually all read accesses to that item will return the last value
  - Implementation in BlockchainDB
    - Execute each incoming read operation of a client immediately
  - Staleness
    - Pending-write queue might grow quickly
    - BlockchainDB might return stale values
  - Solution: set a bound on the size of the pending-write queue
    - This limit is user-defined and provides a trade-off between staleness and latency

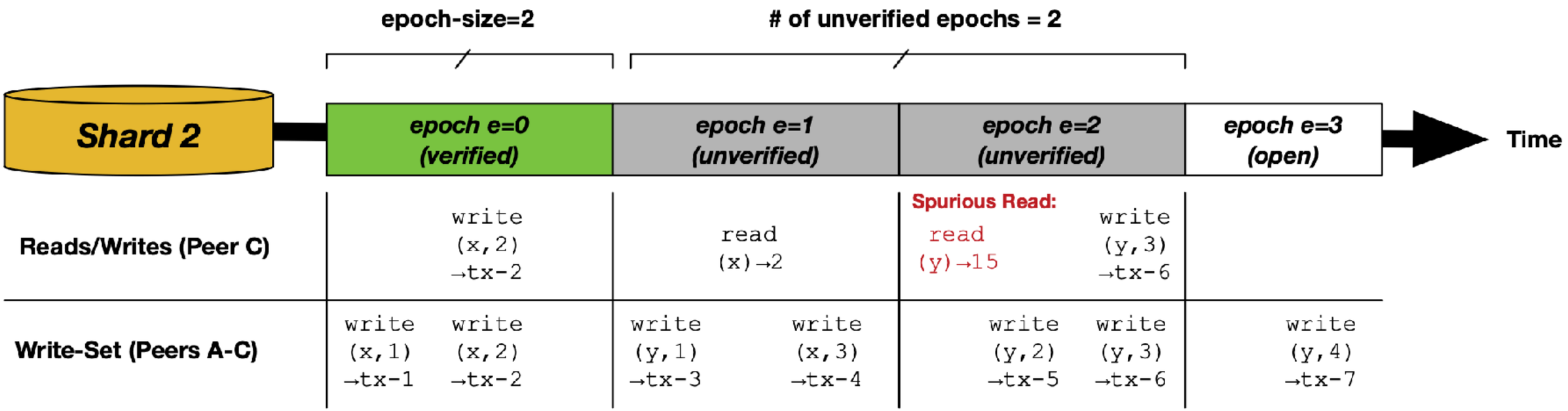
Off-chain Verification

- Main goal
  - Peers can detect other potentially misbehaving peers
    - The storage layer of a remote peer could potentially drop puts or return a spurious value for a read operation
- Online verification
  - Every operation issued by a client is subsequently verified if the client calls the *verify* operation
- Offline verification
  - Defers verification
  - Executed per shard in batches called *epochs*

On-line Verification



Off-line Verification



END OF LECTURE