EINDHOVEN UNIVERSITY OF TECHNOLOGY

Consensus and Blockchains

Architectures of Distributed Systems, 2019

Mike Holenderski

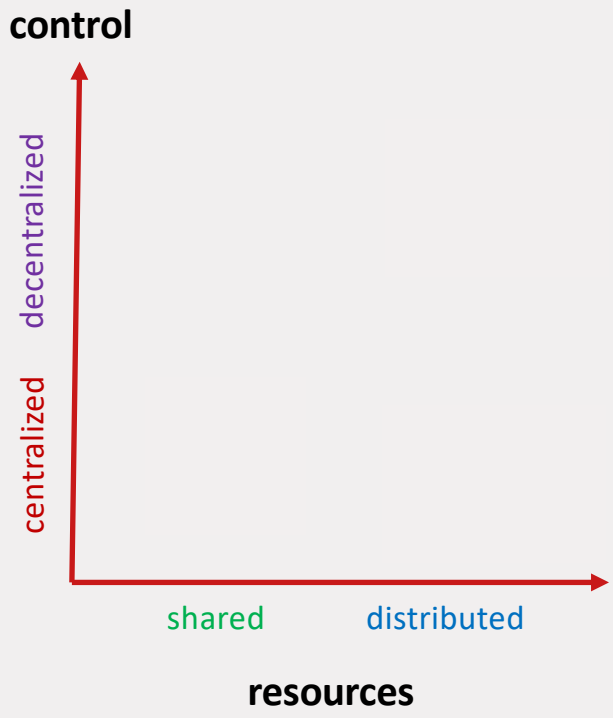
Decentralized vs. Distributed systems

Decentralized

- Partitioning of *control*
- Decisions are taken autonomously by peers
- No central authority

Distributed

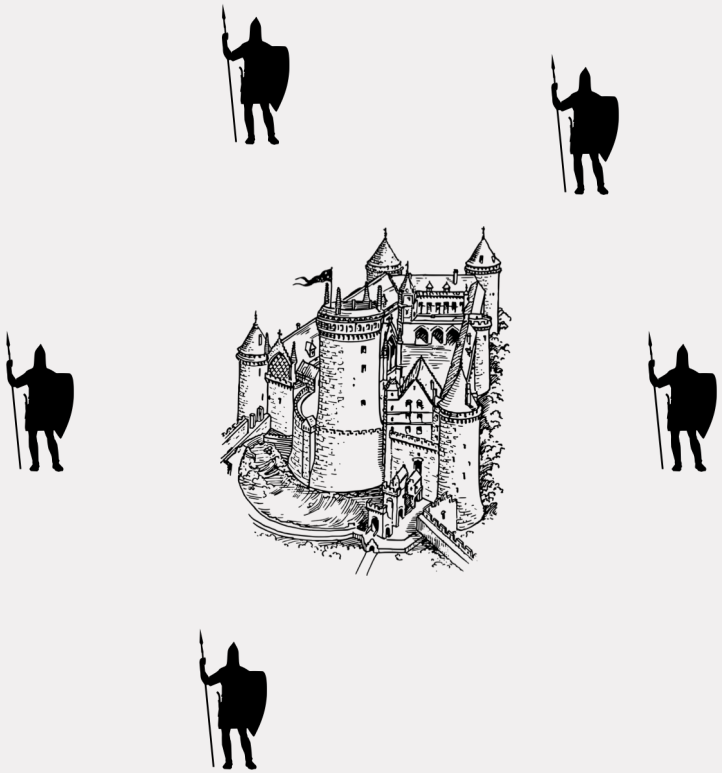
- Partitioning or *resources* (CPU, memory, …)
- Resources are located at different network nodes
- Computation is coordinated via message passing



Byzantine agreement problem

"The Byzantine Generals Problem"
by Lamport, Shostak, Pease (1982)

- Group of Byzantine generals encircle a city
- Must reach **consensus** if to attack or retreat (otherwise will be defeated)
- Communicate via **messengers**
- Messengers are **unreliable**
- Some generals may be **traitors**



Consensus problem

Given:

- variable x
- set of n nodes:
  - each node i has an initial value X_i for x
  - subset of f faulty nodes and $n - f$ correct nodes
- nodes communicate by sending messages

have the correct nodes agree on a common value X for x

Fault models (for nodes)

Crash-stop

- A node executes correctly up to some point when it stops

Byzantine

- Nodes can behave arbitrarily

(Many other fault modes possible)

Consensus guarantees

Agreement:
If two correct nodes decide, then they decide on the same value.

Validity:
If a correct nodes decides on X, than X was proposed by some node.

Termination:
All correct nodes eventually decide on a value.

Consensus applications

- Selecting a leader (node)
- Committing or aborting a transaction
- Establishing a total order of operations

Impossibility of Distributed Consensus

by Fischer, Lynch, Paterson (1983)

Assuming:

- Asynchronous model
  - nodes run at arbitrary relative speed
  - messages always delivered, but with arbitrary delay and arbitrary order
  - no global clock
- Deterministic algorithm
- Single faulty node (crash-stop)

... consensus is not guaranteed!

Solutions

Relax some of the assumptions, e.g.:

- Synchronous communication (e.g. bounded message delays)
- Randomized algorithms

Usually require each node to talk with every other node

- Require at least $O(n^2)$ messages

Practical Byzantine Fault Tolerance (pBFT)

by Castro, Liskov (1999)

Assumptions:

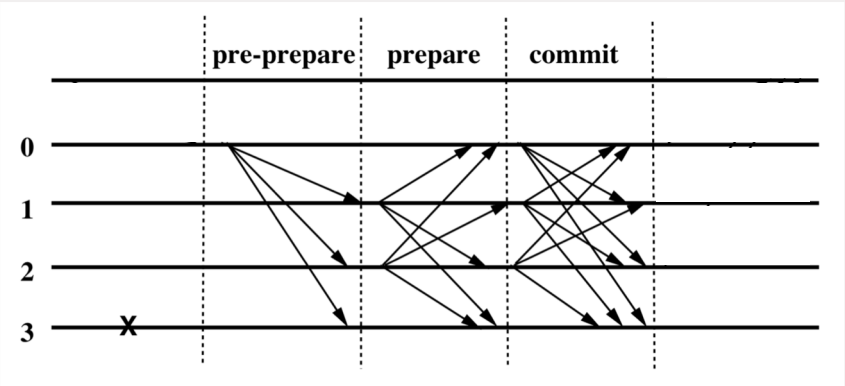
- Asynchronous model *with bounded message delay*
- At most $f = \left\lfloor \frac{n-1}{3} \right\rfloor$ faulty nodes (Byzantine)

Practical Byzantine Fault Tolerance (pBFT)

- 1 *proposer* node for value of x
- $n - 1$ *validator* nodes

Algorithm (3-phase commit):

1. *Proposer* node broadcasts a PRE-PREPARE with the value of x to the *validator* nodes
2. Upon receiving PRE-PREPARE messages, validator checks the value and broadcasts PREPARE
3. Upon receiving $2f + 1$ PREPARE messages, node broadcasts COMMIT
4. Upon receiving $2f + 1$ COMMIT messages, node sets value to X



Practical Byzantine Fault Tolerance (pBFT)

Weaknesses:

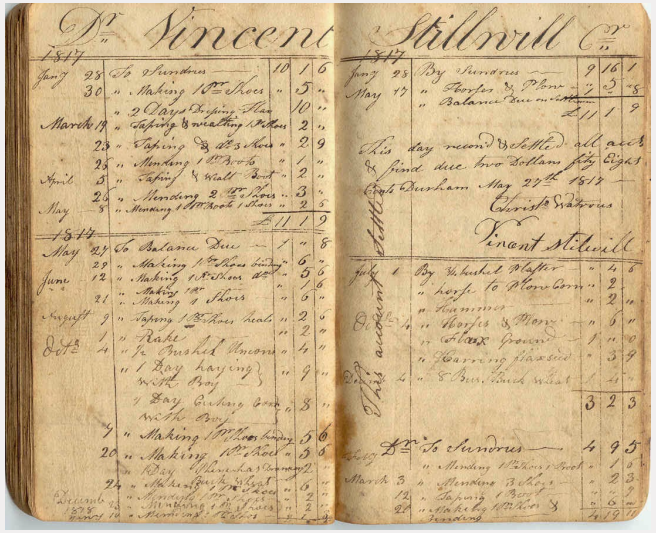
- Requires $O(n^2)$ messages to be sent
- Susceptible to Sybil attacks
  - Single malicious entity operating multiple nodes

(Bitcoin) Blockchain

by Satoshi Nakamoto (2008)

Main idea (for leader election):

- Leader must have certain expertise
- Proving expertise is difficult
- Verifying the proof is easy
- In effect only one (or very few) person will be able to prove their expertise at a time
  - Reduce the number of messages to O(n)!



Original application: distributed ledger

Crypto primitives

Hashing

- Allows to verify integrity of data

Signing

- Allows to authenticate the signer

(Bitcoin) blockchain building blocks

- Accounts and transactions
- Ledger
- Mining
- Incentives
- Merkle trees

Accounts and transactions

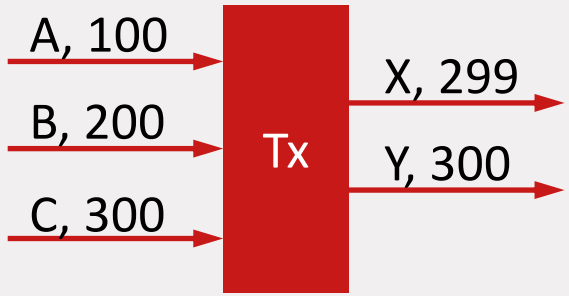
Account:

- Pair: (Private/public key pair, current value)
- Only the private key holder can sign transactions from the account

Transaction:

- Transfer of value from input to output accounts (A,B,C,X,Y are public keys)
- Transaction cost:

$$\Delta = \sum inputs - \sum outputs$$

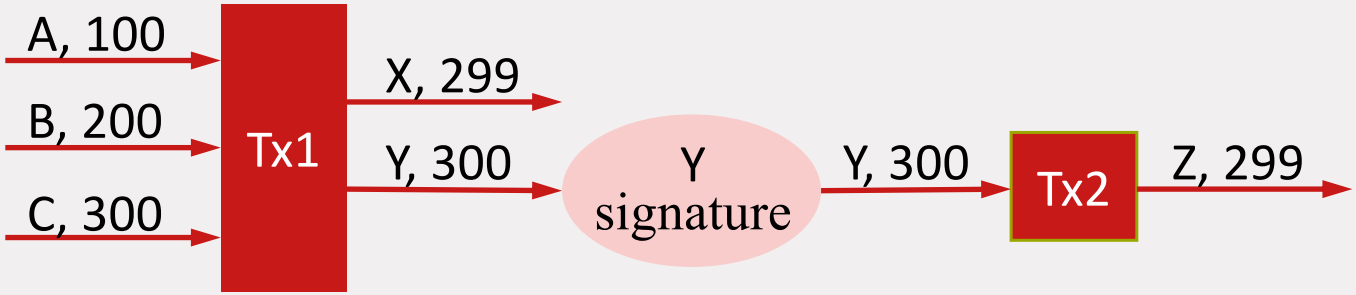


$$\sum inputs \geq \sum outputs$$

Accounts and transactions

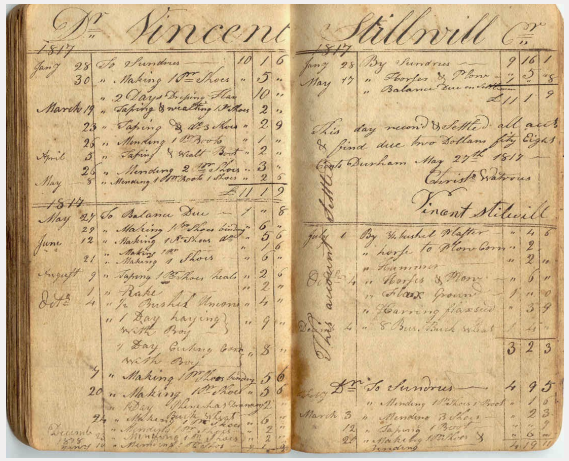
Transaction graph:

- Transactions form a graph (to prevent double spending)
- Leaves = account balances (existing Bitcoins)



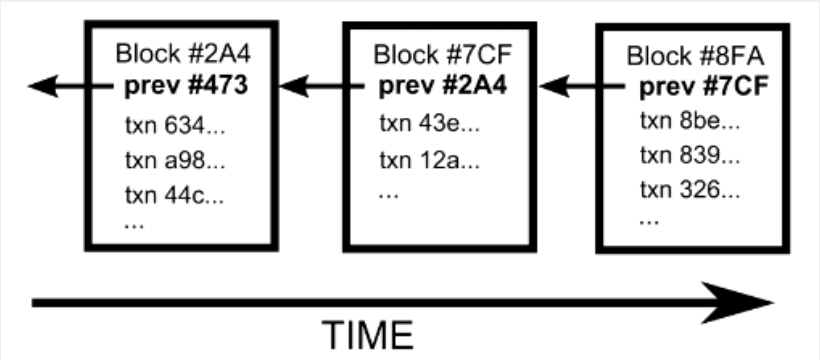
Ledger

- Transactions are combined into blocks
- Blocks are appended in a sequence



Block:

- Header (80 bytes) with metadata
  - timestamp
  - **back hash (hash of the previous block)**
  - …
- Payload (up to 1 MB) with transactions



Verify integrity of blockchain:

- Check if previous block hashes to the back hash

Mining

Client nodes

- Create transactions and broadcast through the network

Miner nodes

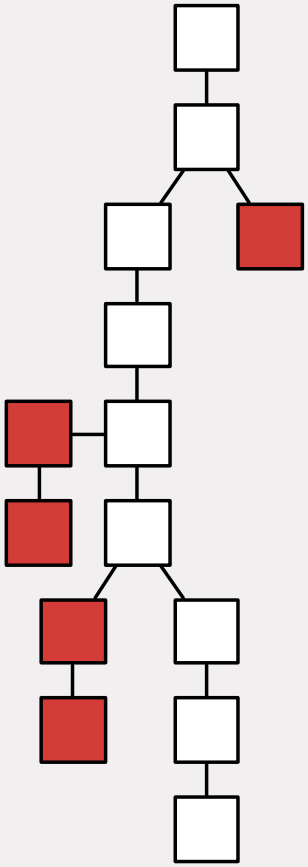
- Collect transactions, to form a new block
- Solve a puzzle to append the block to the blockchain:
  - Find a *nonce* (a 32 bit number) such that *hash*(*block*+*nonce*) > a globally known 256 bit number
  - Nonce is called **proof of work (PoW)**
- Broadcast the solved block to the network

Full nodes

- Collect mined blocks and verify them

Resolving forks

- Miners have a local view on the network
  - Receive transactions from neighboring client nodes
  - Receive solved blocks from neighboring miners
  - Network latency
- Miner append solved blocks to their local blockchain copy



Problem:

- Chain may **fork** when blocks are solved at similar times

Solution:

- Append solved blocks to the longest path
- Keep forks to predefined depth (6 in Bitcoin)
- Adapt PoW difficulty to average 10 minute block creation time

Incentive

- A miner is rewarded 12.5 BTC (currently around $100 000)

- Only paid if block is valid
  - Transactions properly formed, consistent (no double spending), …
  - **Motivates miners to keep integrity of the blockchain**

- Only the reward in the global chain (i.e. longest fork) counts
  - Requires a “51%” attack (note that the actual percentage is lower) to have invalid blocks accepted (e.g. blocks with double spending)
  - **Deters malicious players from cheating**

Proof of Work

Role of Proof of Work:

1. Limit communication overhead to $O(n)$
2. Maintain integrity of the blockchain

An optimization

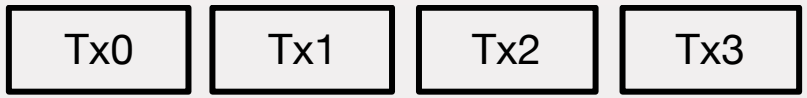
Problem:

- It is expensive to process all transactions during validation

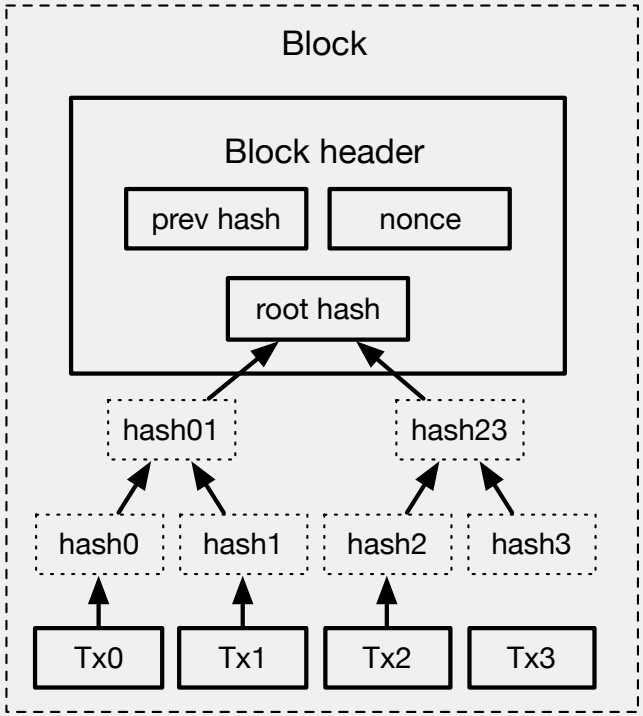
Solution:

- Merkle trees

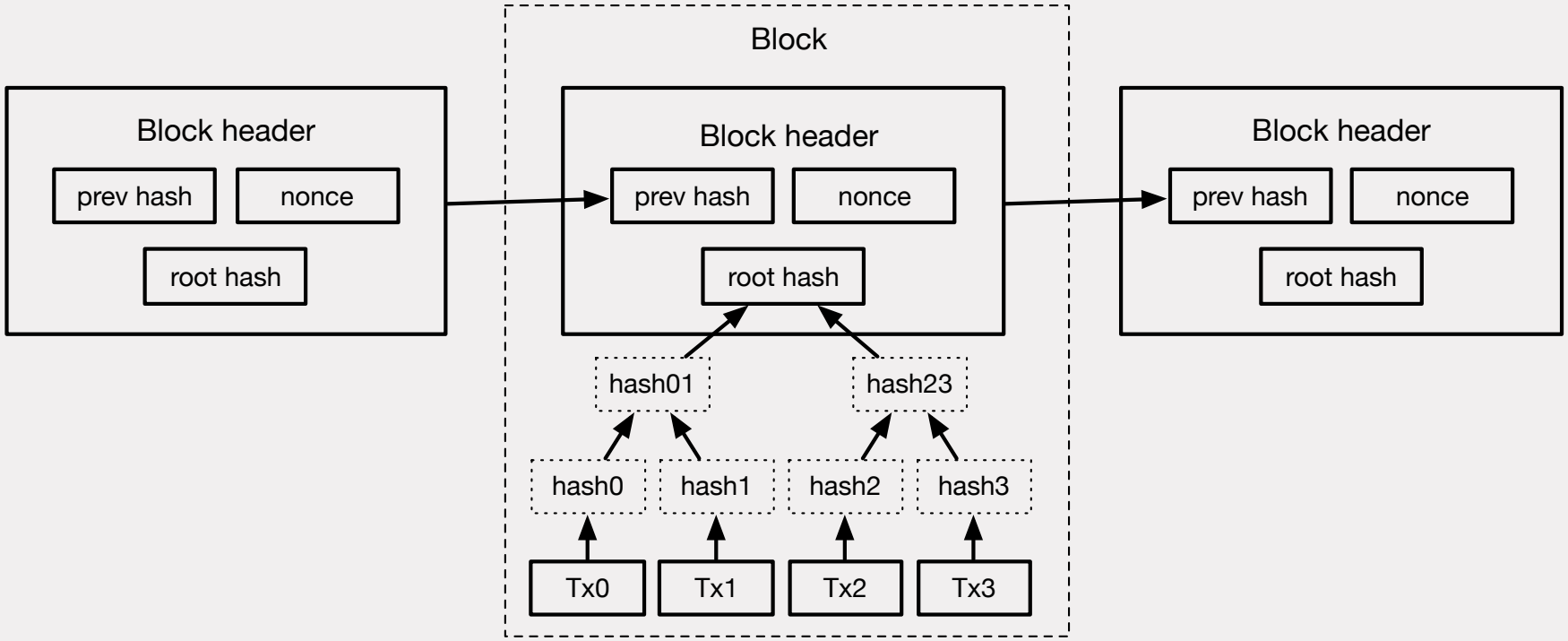
Merkle trees



Merkle trees



Merkle trees



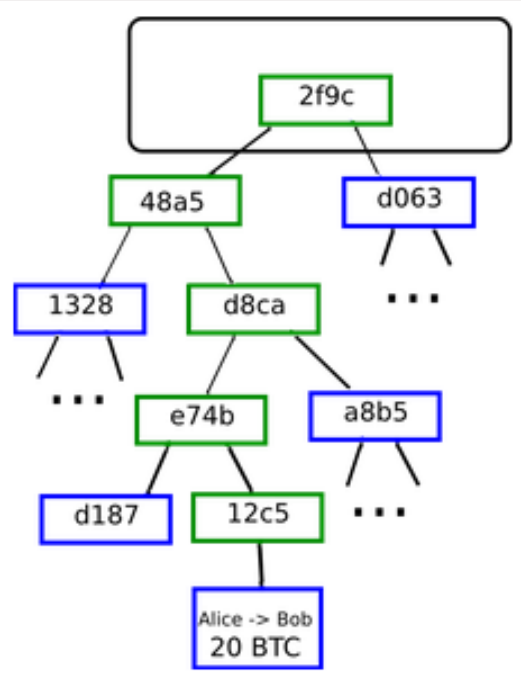
Merkle trees

Optimization trick

- Instead of back-hashing a whole block, back-hash only the header
- About 10,000 times speedup in validating back-hashes

Allow light client nodes

- Store only headers
- Authenticate transactions by getting Merkle proof from a full node



Blockchain novelty:

technology

+

economic incentive

Smart contracts

Smart contract generalizes a financial transaction

- Allow for “any” computation (execution limits to prevent Denial-of-Service attacks)
- Code stored on the blockchain
- Triggered by blockchain transactions
- Interacts with the blockchain's state (reads and writes data stored on the blockchain)

Examples

- Ethereum (permissionless, PoW)
- Hyperledger (permissioned, pBFT, Paxos, Proof of Authority)
- Libra (permissioned, pBFT)

Issues

Proof of work (PoW) is costly

- computing hardware, **energy**, …

Database replication is costly

- all miners and full nodes must keep a copy of the whole blockchain

Execution of smart contracts is costly

- contracts must be executed by all miners and full nodes

Large latency

- Bitcoin 10 minutes, Ethereum 15 seconds between blocks

Low throughput

- Due to 10 minute block create time + block size limit
- Bitcoin 7, Ethereum 25 transactions per second

Solutions

- PoW replaced by PoS (Proof of Stake)
- Database sharding (partitioning) instead of replication
- Payment channels
- …

References

- Lamport, Shostak, Pease, “The Byzantine Generals Problem”, ACM Transactions on Programming Languages and Systems, 1982
- Fisher, Lynch, Paterson, “Impossibility of Distributed Consensus with One Faulty Process”, Journal of the ACM, 1985
- Lamport, “The part-time parliament”, ACM Transactions on Computer Systems, 1998
- Lamport, “Paxos Made Simple”, 2001
- Castro, Liskov, “Practical Byzantine Fault Tolerance”, Symposium on Operating Systems Design and Implementation, 1999
- Satoshi Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System”, 2008