1 Introduction

1.1 Distributed Systems Definition

A distributed system in its simplest definition is a group of computers working together as to appear as a single computer to the user.

1.2 Why Distributed Systems

- Scaling
  - Vertical: more memory, faster CPU
  - Horizontal: more machines
- Economics
  - Initially scaling vertically is cheaper until max HW
  - Current x86 max: 64 cores
- Location
  - Everything gets faster, latency stays
  - Physically bounded by the speed of light
  - New Protocols can decrease RT
  - Place services closer to user
- Fault tolerance
  - Every hardware will crash eventually

1.3 Scaling

1.3.1 Horizontal

Pros

- Lower cost with massive scale
- Easier to add fault-tolerance
- Higher availability

Cons

- Adaption of software required
- More complex system, more components involved

1.3.2 Vertical

Moore's Law: Nr. of transistors doubles every 2 years.
Nielsen's Law: High-end user's connection speed grows by 50% per year.
Kryder's Law: Disk density doubling every 13 month.
Bandwidth grows slower than computer power

Pros

- Lower cost with small scale
- No adaption of software required
- Less administrative effort

Cons

- HW limits for scaling
- Risk of HW failure causing outage
- More difficult to add fault tolerance

1.4 Distributed Systems Categorization

Tightly Coupled

- Processing Elements have access to a common memory

Loosely Coupled (this lecture)

- Processing Elements have NO access to a common memory

Homogenous System

- All processors are of the same type

Heterogeneous System (this lecture)

- Processors of different types

Small Scale

- WebApp + database

Large Scale (this lecture)

- More than 2 machines

Decentralized

- Distributed in the technical sense, but not owned by one actor

1.4.1 Controlled Distributed Systems

- 1 responsible organization
- Low churn
- Secure environment
- High availability
- Homogenous / Heterogeneous
- Examples: Amazon DynamoDB, Client/Server

Mechanisms that work well:

- Consistent hashing
- Master nodes, central coordinator

Network is under control or client/server

- no NAT issues

Consistency

- Leader election (Zookeeper, Paxos, Raft)

Replication principles

- More replicas: higher availability / reliability / performance / scalability
- Requires maintaining consistency in replicas

Transparency principles apply

1.4.2 Fully Decentralized Systems

- N responsible organizations
- High churn
- Hostile environment
- Unpredictable availability
- Heterogeneous
- Examples: BitTorrent, Blockchain

Mechanisms that work well:

- Consistent hashing (DHTs)
- Flooding/broadcasting - Bitcoin

NAT and direct connectivity huge problem
Consistency

- Weak consistency: DHTs
- Proof of work

Replication / Transparency principles apply

1.4.3 CAP theorem

A distributed data store cannot simultaneously be consistent, available and partition tolerant.

- **Consistency:** Every node has the same consistent state
- **Availability:** Every non-failing node always returns a response
- **Partition Tolerant:** The system continues to be consistent even when network partitions

Examples:

- Network partition: AP or CP
- Blockchain: CP or AP
- Cassandra (Apple): AP, can be configured CP

1.5 Transparency in DS

Distributed system should hide its distributed nature

- Location: users should not be aware of the physical location
- Access: users should access resources in a single, uniform way
- Migration, relocation: users should not be aware, that resources have moved
- Replication: Users should not be aware about replicas, it should appear as a single resource
- Concurrency: users should not be aware of other users
- Failure: Users should be aware of recovery mechanisms
- Security: Users should be minimally aware of security mechanisms

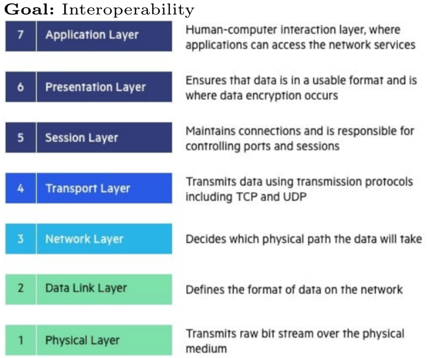
1.6 Fallacies of Distributed Computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogenous

2 Protocols

2.1 Networking Layers

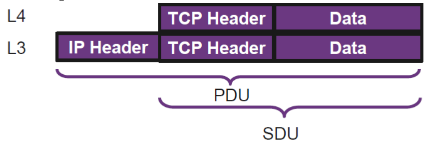
Goal: Interoperability



2.1.1 Layer Abstraction

- Protocols enable an entity to interact with an entity at the same layer in another host
- **Service definitions:** provide functionality to an (N)-layer by an (N-1) layer
- Layer N exchange protocol data units (PDUs) with layer N protocol
- Each PDU contains a header and payload, the service data unit (SDU)

Example PDU of L3:



2.2 Layer 4 - Transport

2.2.1 TCP

- Reliable
- Ordered
- Window - capacity of receiver
- Checksum - 16bit
- TCP overhead: 20 bytes
- Tries to correct errors

Connection establishment

- SYN, SYN-ACK, ACK
- Initiates TCP session: initial sequence number is random

Connection termination

- FIN, ACK + FIN, ACK
- 3-way handshake

Sequences and ACKs

- Identification each byte of data
- Order of the bytes: reconstruction
- Detecting lost data: RTO, DupACK

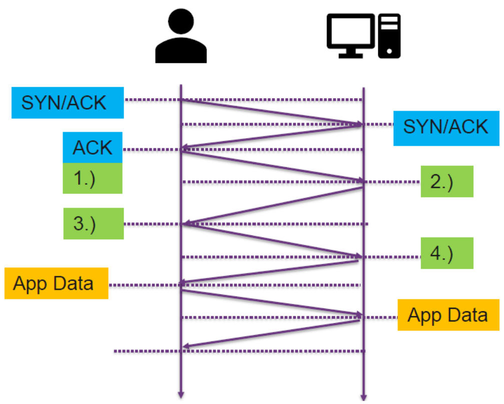
Retransmission timeout

- If no ACK is received after timeout

Flow control

- Sender is not overwhelming a receiver
- Back pressure
- Sliding window
- Congestion control
  - Slow-start
  - Congestion avoidance

2.2.2 TCP + TLS



TLS <1.3

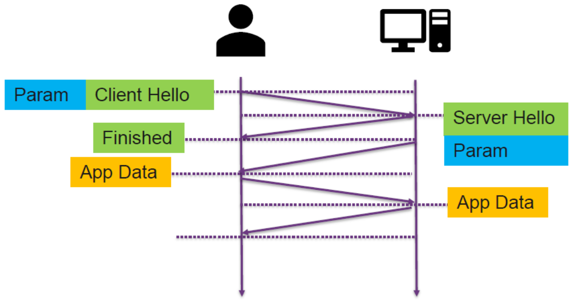
1. client hello - lists crypto information, TLS version, ciphers/keys
2. server hello - chosen cipher, session ID, random bytes, digital certificate
3. Key exchange using random bytes, now client + server can calculate secret key
4. finished - encrypted message

3RTT until first byte
TLS 1.3

- 1 RTT instead of 2
  1. Client Hello, Key share
  2. Server Hello, Key share, Verify Certificate, Finished
- 0 RTT possible for previous connections (no perfect forward secrecy)

2.2.3 QUIC

- 1 RTT (0 RTT for know connections)
- Built in security



- Multiplexing in HTTP/2
- QUIC can multiplex requests: one stream does not affect others

2.2.4 UDP

- Used for DNS, streaming
- Simple connectionless communication model
- No guarantee
  - Delivery
  - Ordering
  - Duplicate protection

2.2.5 SCTP - Stream Control Transmission Protocol

- Message based
- Allows data to be divided into multiple streams
- Syn cookies: Four-way handshake with a signed cookie
- Multi-homing multiple IP addresses of endpoints

3 Virtualization

Creation of a virtual machine that acts like a real computer with an operating system.
Host: machine where the virtualization SW runs.
Guest: VM
Hypervisor: runs VM

- Type 1: bare-metal e.g. Xen
- Type 2: hosted e.g. VirtualBox

Needs to be the same architecture

- Otherwise emulation needed

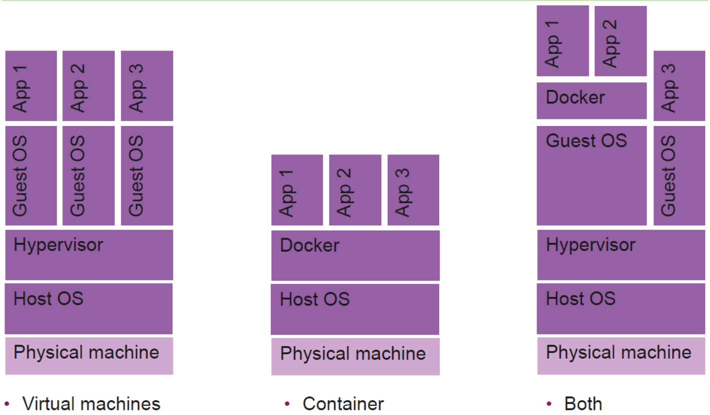
Virtual desktop infrastructure (VDI)

- Interact with a VM over a network

Containers

- Isolated user-space instances
- Share the OS

3.1 VM vs. Container



- Containers are more agile than VMs
- Containers enable hybrid and multi-cloud adoption
- Integrate Containers with your existing IT processes
- Containers safe on VM licensing

3.1.1 Container

\+ Reduced IT management resources
\+ Reduced size of snapshots
\+ Quicker spinning up apps
+/- Available memory is shared
+/- Process-based isolation (share same kernel)
Use case: complex application setup, with container less complex config

3.1.2 Virtual Machine

\+ App can access all OS resources
\+ Live migrations
+/- Pre allocates memory
+/- Full isolation
Use case: better hardware utilization / resource sharing

3.2 Docker

- Containerization platform
- Software delivery framework
- Packages software into containers
- Provides OS-level virtualization
- Containers are isolated from each other

Docker Compose

- Deploy multiple containers

graphicx

4 Loadbalancing

Distribution of workloads across multiple computing resources.
Horizontal scaling: Distributes client requests or network load efficiently across multiple servers.

4.1 Why?

- Ensures high availability and reliability
- Sending requests only to servers that are online
- Provides flexibility to add/subtract servers on demand

4.2 Types of Load Balancers

4.2.1 Hardware Load Balancer

- Use proprietary software, which often uses specialized processes
  - Less generic, more performance
  - Some use open-source SW
- Only if you control your datacenter
- E.g. loadbalancer.org, F5, Cisco

4.2.2 Software Load Balancer

- L2/L3: Seesaw
- L4: Traefik, Nginx, LoadMaster, etc.
- L7: Traefik, Envoy, Neutrino, Envoy, etc.

DNS Load Balancing

- Round-robin DNS
  - Very easy to set up
  - Static
  - Caching with no fast changes
- Split horizon DNS
  - Different DNS information
  - Depending on source of the DNS request
- Anycast
  - Difficult and time consuming
  - Return the IP with lowest latency

4.2.3 Cloud-based Load Balancer

- Pay for use
- Many offerings
  - AWS, Google Cloud, Cloudflare, DigitalOcean, Azure

4.3 Load Balancing Algorithms (L4/L7)

- Round robin: loop sequentially
- Weighted round robin: some server are more powerful
- Least connections
- Least time: fastest response time + fewest connections
- Least pending requests
- Agent-based: service reports on its load
- Hash: Distributes based on a key
- Random

Conclusion

- Easiest: Round-robin
- Stateless: don't store anything in the service
- Health checks: Tell LB if you are low on resources
- L7 LB is more resource intensive than L4 LB

4.4 Traefik

- Open Source, SW based
- L4/L7
- Golang, single binary
- Authentication
- Experimental HTTP/3 support

4.5 Caddy

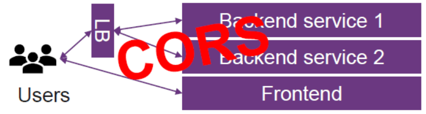
- Open Source, SW based
- L7
- Reverse proxy
- Static file server
- Experimental HTTP/3 support

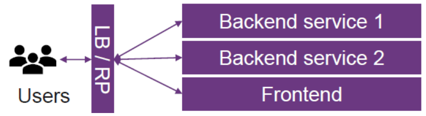
4.6 NGINX

- Free + commercial version
- HTTP / Mail / reverse proxy
- No active health checks (commercial)
- No sticky sessions (commercial)

4.7 CORS

For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts.
Solution:
Use reverse proxy with builtin webserver e.g. nginx, or use reverse proxy with external webserver.
The client only sees the same origin for the API and the frontend assets.





Access-Control-Allow-Origin
graphicx

5 Authentication

Confidentiality: Protects transmitted data against eavesdropper.
Integrity: Provides protection against modification.
Availability: Data needs to be available when needed.
Non-repudiation: No one can deny an action.
Identification: Username connects to a person.
Authentication: Verifying a claim of identity with:

- Something you know
- Something you have
- Something you are

Authorization: Determines what resources a user can access.

5.1 Software Token

TOTP: Time-based One-time Password

- Often used as 2nd factor
- Based on keyed-hash message authentication code

5.2 Basic Auth

- Load balancer
- Services (keep state!)
- Only with HTTPS
- Can be encoded in URL: *user:pw@domain*
- Server will reply with header: *WWW-Authenticate*

5.3 Digest Auth

- Based on Basic Auth
- Also available in traefik
- Hash + nonce, against replay attacks

Advantages

- PW not in clear text (MD5), can be SHA-256
- Nonce for replay protection for client/server

Disadvantages

- Browser L&F
- Cannot use scrypt or bcrypt to store PWs

5.4 Create SSL CA certificates for server

- Create CA
- Create certificate
- Add nginx security in your local network

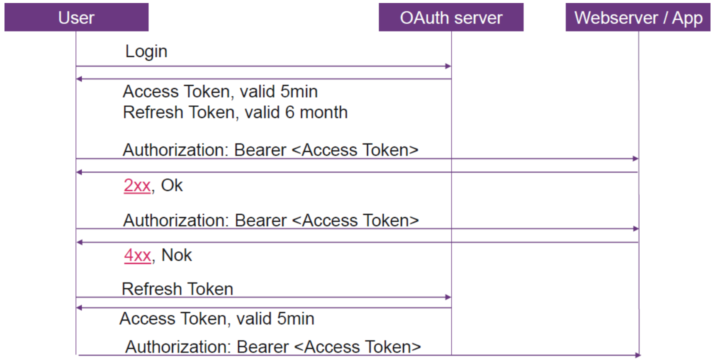
5.5 Session-based authentication

- Sticky session required
- Authenticate in Service Instance

5.6 JWT

- Stateless
- All server instances know a secret token / public key
- When user logs in, server send back token
- Client sends: Authorization: Bearer ¡token¿
- Client can store token in local storage

5.6.1 Access Token / Refresh Token



Access Token
- Short lifetime (10min)

Refresh Token
- Used to get a new access token
- IAM / Auth server creates access tokens

6 Application Protocols

Designing custom protocols
- Needs more time to develop / test
+ Can be more efficient (space/performance)

Protocol generators: (ASN1, ProtoBuf)
+ IDL (interface description language) generates code
+ Standard
- Has more overhead

6.1 ASN1
- Define data structures
- Can be serialized and deserialized
- Generic binary protocol

6.2 Protocol Buffers (ProtoBuf)
- Data serialization system from Google
- Design goals: smaller and faster than XML
- Use: nearly all inter-machine communication at Google
- Contains only numbers, not field names

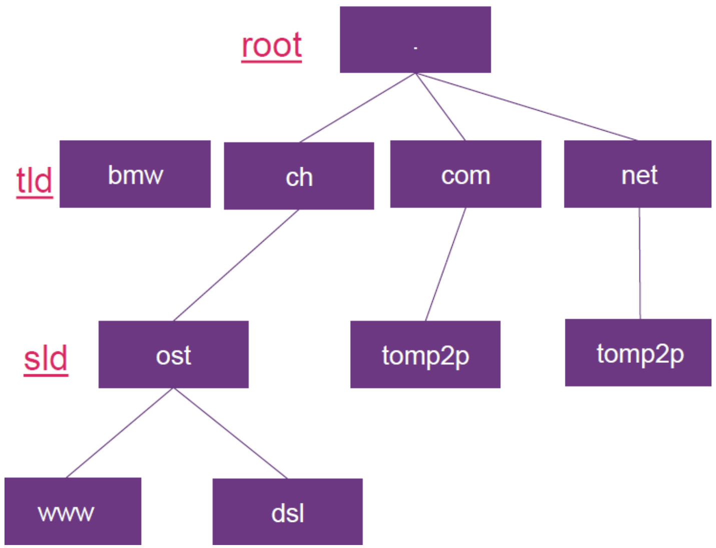
6.3 gRPC
- Uses HTTP/2 for transport
- Uses ProtoBuf
- Features
  - Authentication
  - Bidirectional Streaming
  - Flow Control
  - Blocking / Nonblocking bindings
  - Cancellation and timeouts
- Define services and messages

6.4 JSON + REST
- Human readable text to transmit data
- Often used for web apps
- Parsing overhead
- JSON slower than binary protocol

6.5 HTTP
- Text based protocol
- Request / Response
- Request Methods
  - GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH
- Stateless
- HTTP resources identified by URL

```
Scheme   User info      Host     Port   Path     Query         Fragment
http://tbocek:password@dsl.hsr.ch:443/lect/fs21?id=1234&lang=de#topj
```

6.6 DNS
- Translates human readable domain to IP
- Phonebook of the Internet
- Hierarchical and decentralized naming system
- Caching/forwarding DNS
- Recursive servers
- Authoritative servers
- Restriction to 13 root servers (512 byte packet limit)



Type of records:

- SOA: Start of Authority
- NS
- MX
- A/AAAA
- TXT
- PTR

DNSSEC

- Authenticated and data integrity, NOT confidentiality
- Can be used to bootstrap other security systems
- KSK: key singing keys to sign ZSK
- ZSK: zone signing keys to sign records

6.6.1 The DNS war

DoH (DNS over HTTP)

- Provides confidentiality of lookups in transit
- Uses standard HTTP/2 on Port 443
- Trivially deployed, DNS response served like web pages
- Performance: TCP+TLS handshake = 2/3 RTT
- Difficult upgrade path for clients
- Browsers can perform DNS queries using JavaScript

DoT (DNS over TLS)

- Provides confidentiality of lookups in transit
- DNS over TLS, separate port 853, can be blocked
- Widely supported by serving software and public resolvers
- Performance: TCP+TLS handshake = 2/3 RTT
- Easy upgrade path for clients

6.7 Let's Encrypt

- Non-profit CA
- Provides certificates for TLS
- No Identity Checks
- Certificates are valid for 90 days
- Automated renewal
  - ACME protocol: challenge response

7 Git

müssti eigentlich klar si..
trotzdem no paar commands:

- Branching
  - git checkout -b feature
  - git checkout **branch-name**
    - Switch between branches
  - git branch -a
    - Show branches
  - git branch -d **branch-name**
    - Remove local branch
  - git push -d origin feature
    - Remove remote branch
  - git push origin feature
    - Push it remote, before, everything was local
- Merging
  - git merge feature
- Get changes from remote
  - git pull / git pull --rebase
- Go back
  - git log
  - git reset –hard cc5507b071
  - git revert cc5507b071..HEAD
    - Revert last commits
  - Remove passwords, data: git-filter-branch, bfg
- Tags (release in github specific)
  - git tag tagname cc5507b071
  - git checktout tagname
  - git show tagname
  - git push origin tagname
    - To sync it with remote

8 Bitcoin / Blockchain

Bitcoin
- Experimental digital currency
- Fully P2P, no central entity
- Maximum of 21 million BTC
- Every transaction broadcast to all peers
- Validation by proof-of-work (partial hash collision)

8.1 Bitcoint Introduction
- Not relying on trust, but strong cryptography
- Weak anonymity (pseudonimity)
  - All peers know all transactions
  - Clustering: if a transaction has multiple input addresses, assume those addresses belong to the same wallet
- Not controlled by a single entity
- **BIP:** Bitcoin Improvement Proposals
- Can be exchanged for real currencies

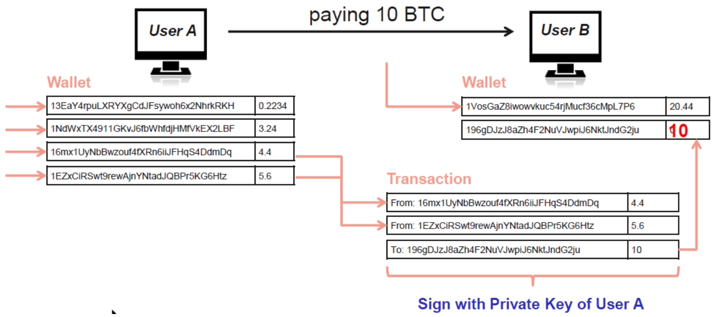
8.1.1 Mechanism

Wallet
- Has public-private keys (wallet.dat)
- Public key, ECDSA 256 bit = bitcoin address
- Simple address = base58(RIPEM160(Sha256(ecdsa public key)))
- Private key used for signing transactions

Transaction
- Peer A wants to send BTC to B, creates transaction message
- Transaction contains input / output
  - source / destination of BTC
- Peer A broadcasts the transaction to all peers
- Transaction stored in blocks, block created / verified in ~ 10min

8.1.2 Key Bitcoin Operations

Private Key authorizes the transaction
- If keys are stolen, thief may use your coins
- If keys are lost, coins are lost
- In UTXO (unspent transaction output) systems, complete output is spent



Avoiding Double spending
- Transactions in blocks are confirmed
- Guessing value that results in zero bits
- Chained proofs of work

8.2 Blockchain

- Transactions are collected in blocks
- New Block every 10min
- Blocks contain solved crypto puzzles
  - Partial hash collisions
- A block has a pointer to a previous block
- Creation of blocks is called mining

8.2.1 Mining

- Creating valid blocks
- Different level of confirmations
  - 3-6 block conf. is considered secure
- Dangerous if someone has more than 50% computing power
  - could exclude and modify the ordering of transactions

8.3 Discussion

Advantages

- Low (fixed) tx fees
- Scalable (faster HW/Storage)
- Anonymity: No privacy concerns
- No major crashes
- Decentralized
- Many other blockchain use cases
  - Smart contracts

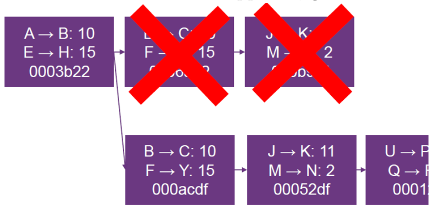
Disadvantages

- Power consumption (as much as Netherlands)
- Not scalable: 5 transactions per sec
- Anonymity: used for illegal activities

8.4 51% Attack

If a majority of CPU power is controlled by honest nodes, the honest chain will grow the fastest and outpace any competing chains.
PoW: Majority of hashing power
PoS: Majority of coins
Double spend / rollback transactions

- X is an exchange
- Mine secretly, Y is your address
- X arrived - payout (1 block conf.)
- You mine faster, broadcast secret chain
- Tx F →X: 15 never happened, goes to Y



8.5 Bitcoin vs. Ethereum

Bitcoin

- Implementing new features slow
- Bitcoin Script limited
- Pros and Cons - no silver bullet

Ethereum

- Generalized blockchain
- Protocols designed from scratch
- Mining reward (block every 14s): ∼ 2 ETH

8.6 Ethereum

8.6.1 Blocktime and Gas

- Gas Price set by Miner
- Miner decides which transaction at which gas price to include
  - Market for transaction
- Gas price too low, longer waiting time until TX will be included
- Block time: 14-15s
- Smart Contracts are turing complete
  - Every instruction needs to be paid for
- Gas price / Gas limit by miners
  - If you run out of gas, state is reverted, ETH gone

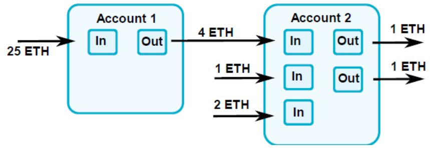
8.7 Smart Contract (ETH)

- For now, proof of work
- Every contract is run on every full Ethereum node
  - Result on every node is the same
  - Global computer, always running, always correct
- Account-based
  - External accounts: controlled by private keys
  - Contract accounts: never executed on their own
    - controlled by code
    - All action fired from externally controlled accounts

8.7.1 Account-based vs. UTXO-based

Account-based

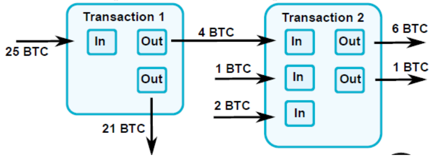
- Global state stores a list of accounts with balances and code
- Transaction is valid if the sending account has enough balance
- If the receiving account has code, the code runs, and state may be changed
  - Signature must match sending account



- Large space savings
  - One input - one output - one signature
- Simplicity
  - Easier to code and understand
  - Easier for smart contracts (stateful scripting language)

UTXO-based

- Every reference input must be valid and not yet spent
- Total value of the inputs must equal or exceed the total value of the outputs
  - you always spend all outputs
- Transaction must have a signature matching the owner of the input for every input
  - Script determines if input is valid



- Higher degree of privacy
  - New address for each TX
  - No replay attacks - no nonce required
  - Allows transactions to be processed in parallel

8.7.2 Security Considerations

1. Reentrancy
2. Access Control
3. Arithmetic Issues
4. Unchecked Return Values for low level calls
5. Denial of Service
6. Bad Randomness
7. Front Running
8. Time Manipulation
9. Short Address Attack

8.8 Solidity

mümmer das chöne?

9 Deployment

9.1 Kubernetes K8s

- Container orchestration (docker)
- Automated deployment, scaling
- Started by Google, now with CNCF

Why Kubernetes?

- Containers can crash
- Machine that runs container can crash (out of memory)
- Kubernetes manages the lifecycle of containers

9.1.1 Design Principles

- Configuration is declarative (YAML/JSON)
- Abstraction layer for distribution system
  - Provides interface to interact with containers
- Immutable containers
  - Don't store state in containers
  - Kubernetes replaces container if health check fails

Pod: one (or more close connected) container (long running)

- Job: short running
- Volume: directory accessible to all containers in a Pod

Deployment: define scale, HW limits
Service: single entry point (internal)
Ingress: expose end points / external access
Namespaces: run multiple projects on one cluster, separate with namespaces
StatefulSets: when running a DB
DaemonSets: placement of pods
Minikube, k3s

- Kubernetes master / server / control plane
- Kubernetes worker / nodes / agent / compute machine

9.2 Docker Swarm

- Group multiple docker nodes
- Deploy with docker-compose.yaml (deploy:)
- Similar base architecture: swarm manager / swarm nodes
- Built into docker
  - *docker swarm*: manage swarm
  - *docker node*: manage nodes
- Scheduler is responsible for placement of containers to nodes

Docker Swarm vs. Kubernetes

- Kubernetes is used more often
- Kubernetes supports higher demands with more complexity
- Docker Swarm offers a simple solution, quick to get started with