



OST

Eastern Switzerland
University of Applied Sciences

Blockchain (BlCh)

Repetition DSy

Thomas Bocek

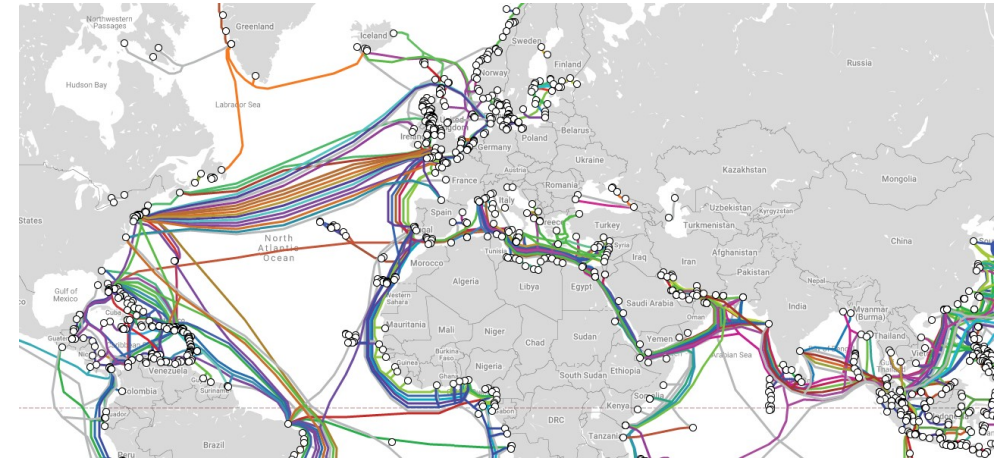
13.09.2026

Lecture 1 + 2

Introduccion / Motivation

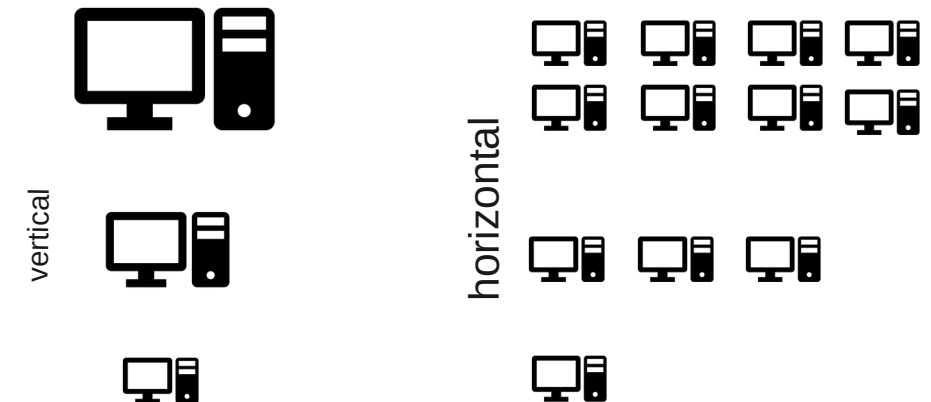
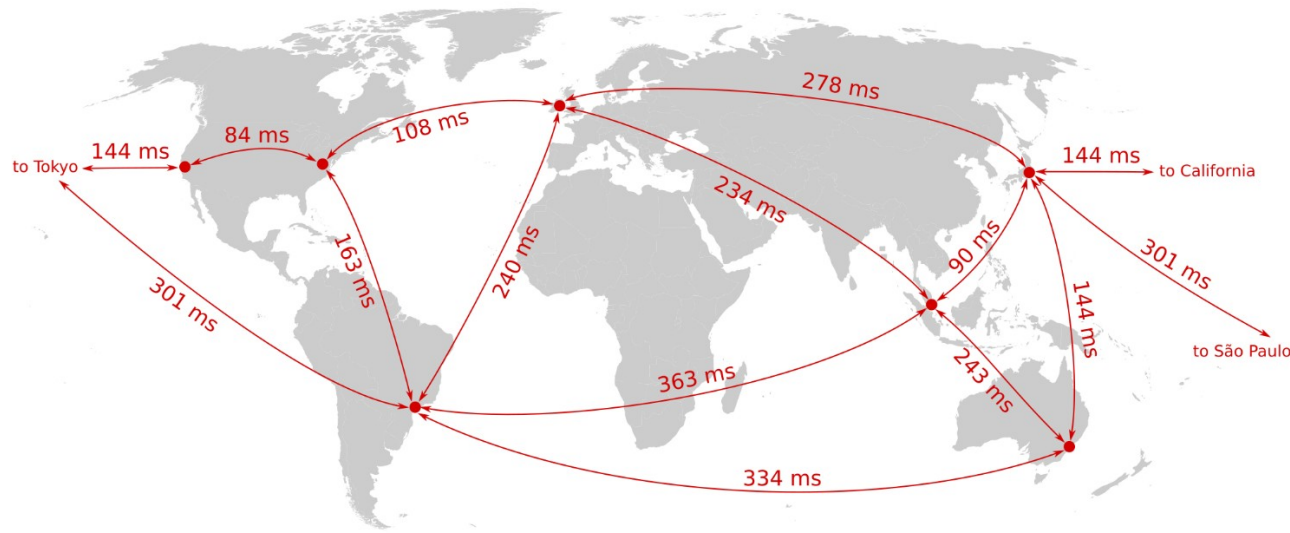
Distributed Systems Motivation

- Why Distributed Systems
 - **Scaling**
 - **Location**
 - **Fault-tolerance** (**bitflips**, **outages**)



Submarine Cable Map

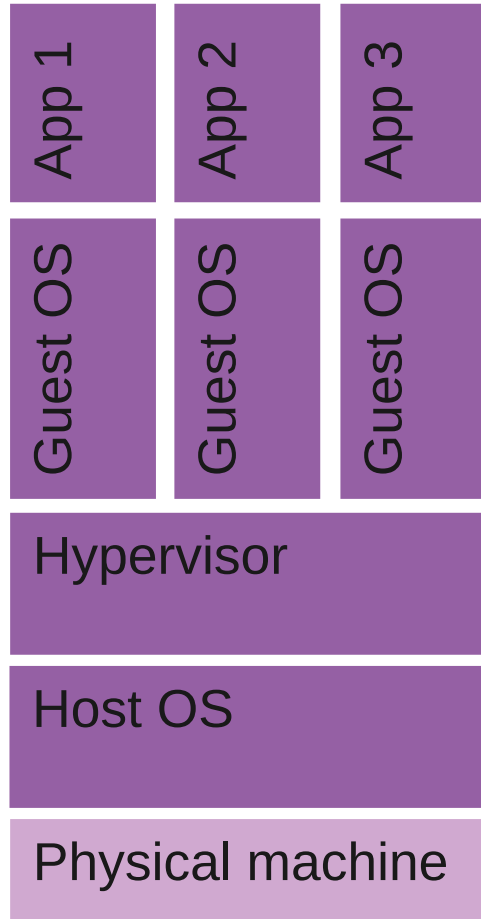
<https://www.inlandswitch.com/local-first.html>



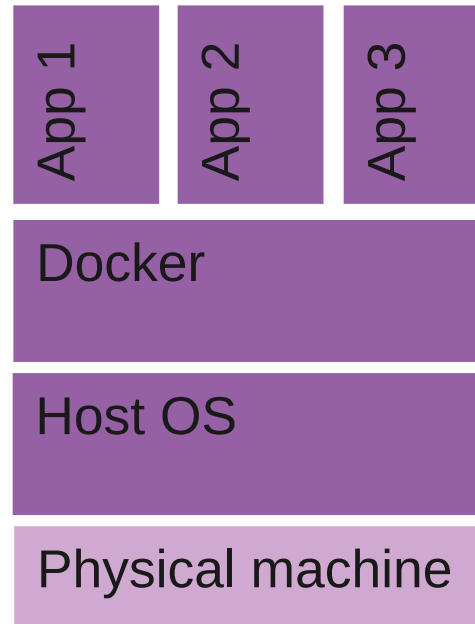
Lecture 3

Containers and VMs

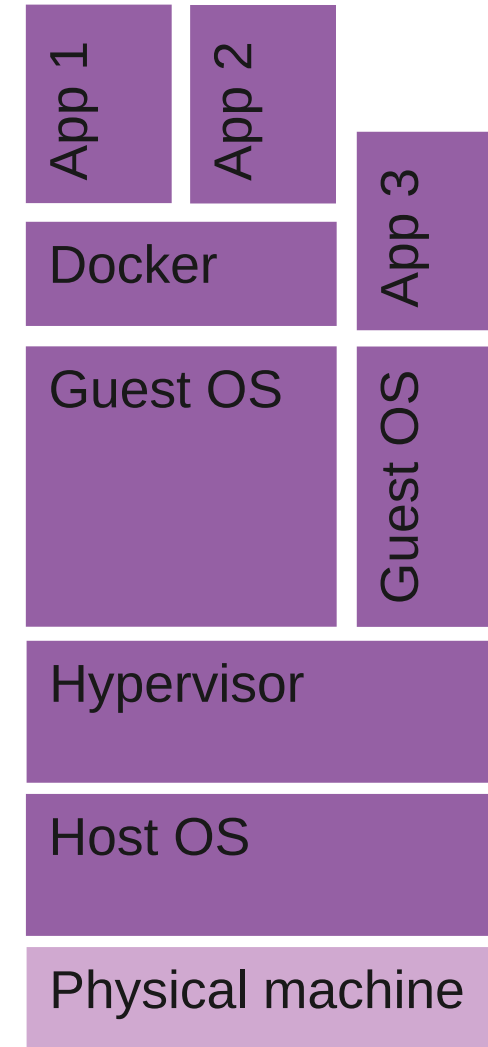
Introduction



- Virtual machines



- Container



- Both

Docker Examples

- Install docker [[ubuntu](#), [Mac](#), [Windows](#)]
 - `docker run hello-world`
 - Fetches the hello world example from [docker hub](#)
 - No version provided – latest
 - [Docker Hub](#): container image repository
 - Community / official
 - [Alpine](#)
 - `docker save hello-world -o test.tar`
 - `tar xf test.tar`
 - `tar xf cdccdf50922d90e847e097347de49119be0f17c18b4a2d98da9919fa5884479d/layer.tar`
 - `./hello`
- See your installed images
 - `docker images / docker images -a`
 - `docker rmi hello-world / docker rmi fce289e99eb9`
 - `docker ps -a`
 - `docker rm 913edc5c90c4`
- GUI: e.g., [Docker Desktop](#)

How to Debug? / Pitfalls

- What is going on in my container, why is it not running?
 - `docker exec -it <id> sh`
 - `nc / wget`
- Reach other container: ping cointainername
 - Docker-compose has own DNS and resolves to containernames
- Service bound to localhost?
 - This cannot run outside docker
- Check logfiles!
 - If you write your own application, handle cornercases at least with logging
- Example with netcat (nc)
 - `nc -l`
- `docker stats`
- Is docker application docker aware?
 - Docker memory limits are not hard limits, e.g., GC is not under pressure
 - If over limit, application restarts (if configured)
- Live-Code-Reloading: use volumes
- Check layers
- Performance: mulit-stage, `.dockerignore`

Lecture 4

Repos

Pro/Cons - Opinion

- **Monorepo**

- Tight coupling of projects
 - E.g., generating openapi.yml from backend, generate types for frontend → simply copy
- Everyone sees all code / commits
- Encourages code sharing within organization
- Scaling: large repos, specialized tooling

- **Polyrepo**

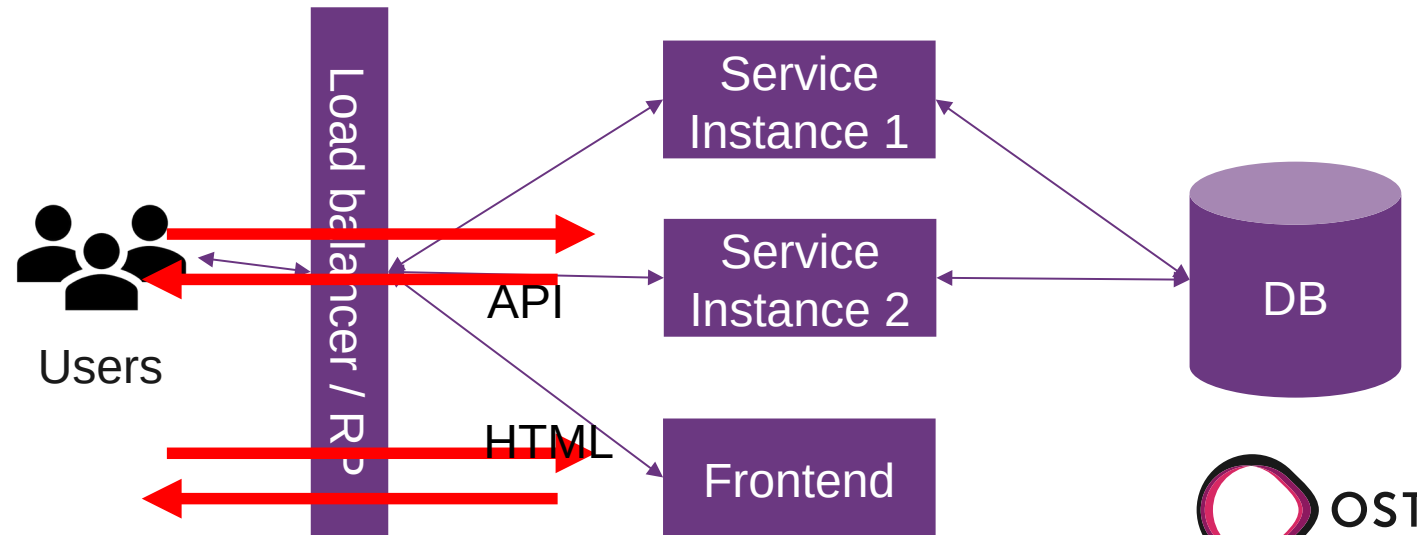
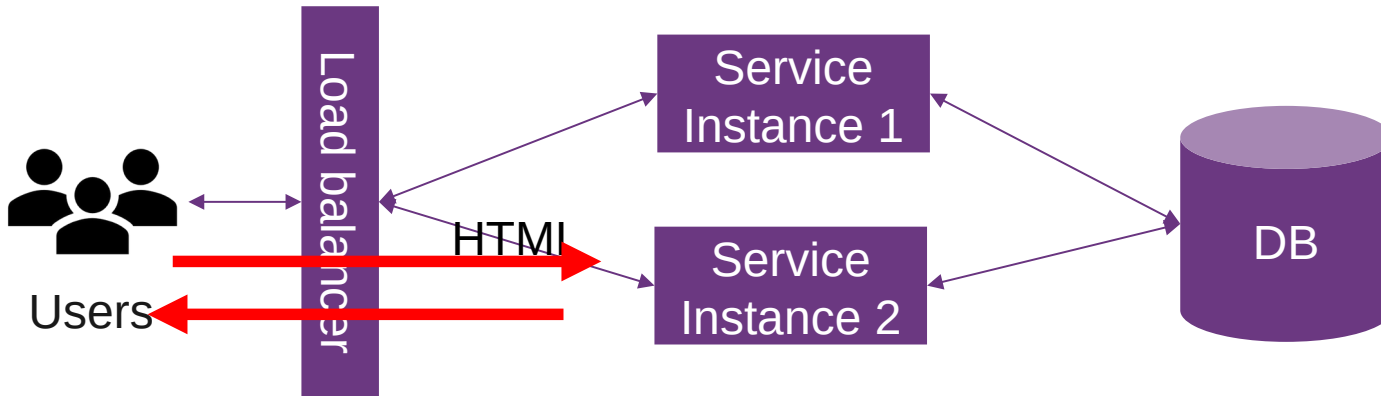
- Loose coupling of projects
 - If you want to generate openapi.yml, you need access from the backend repository to the frontend (e.g., curl+token)
- Fine grained access control
- Encourages code sharing across organizations
- Scaling: many projects, special coordination

Lecture 5

Architecture, CORS, Load balancing

Architecture Comparison

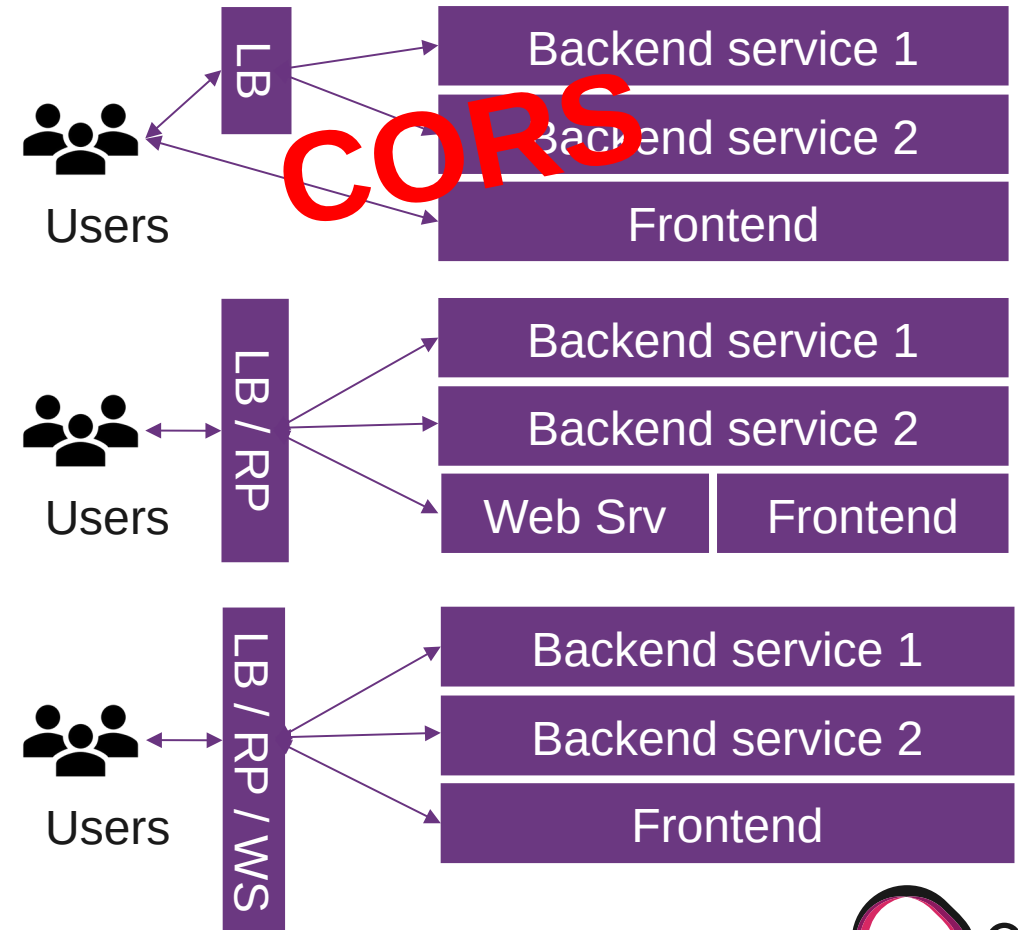
- Server side rendering (SSR)
- Single page application (SPA), client side rendering (CSR)



CORS

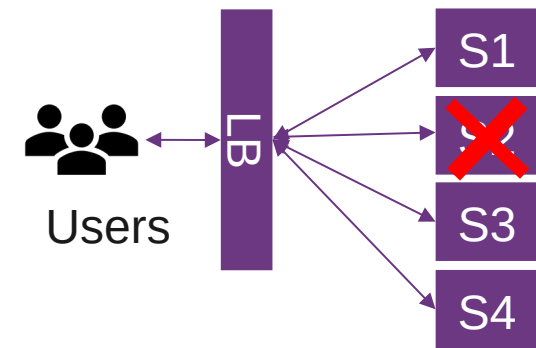
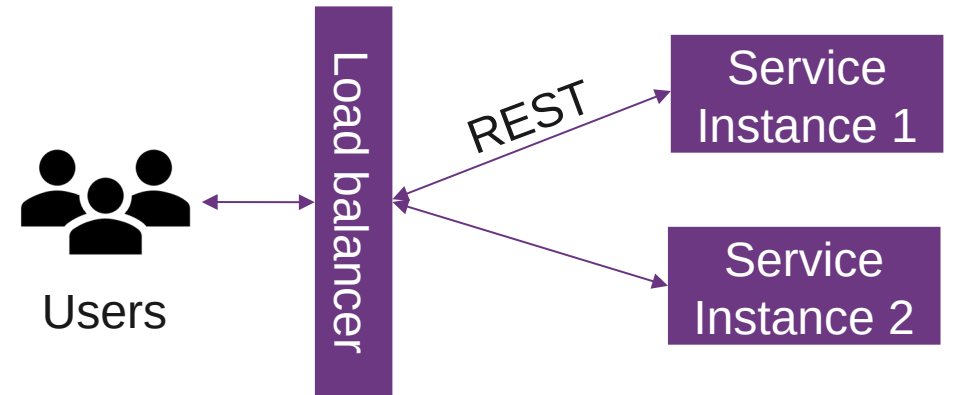
- **CORS** = Cross-Origin Resource Sharing
 - For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts (among others)
 - Mechanism to instruct browsers that runs a resource from origin A to run resources from origin B
- Solution: reverse proxy with webserver
 - Many solutions, caddy, vite/webpack dev server, nginx, mix
 - The client only sees the same origin for the API and the frontend assets
- “Workaround”: Access-Control-Allow-Origin: https://...
 - For dev: Access-Control-Allow-Origin: *
 - Pre-flight requests
 - Golang: `w.Header().Set("Access-Control-Allow-Origin", "*")`

- Reverse proxy



Load Balancing

- What is load balancing
 - Distribution of workloads across multiple computing resources
 - Workloads (requests)
 - Computing resources (machines)
 - Distributes client requests or network load efficiently across multiple servers [\[link\]](#)
 - E.g., service get popular, high load on service
- horizontal scaling
- Why load balancing
 - Ensures high availability and reliability by sending requests only to servers that are online
 - Provides the flexibility to add or subtract servers as demand dictates



Lecture 6

MCP, Authentication

MCP API

- tools/list

- JSON-RPC 2.0

```
→ {"jsonrpc":"2.0","id":1,"method":"tools/list"}  
← {"jsonrpc":"2.0","id":1,"result":{"tools":[  
  {"name":"measure_endpoint","description":"Measure  
  response time of a URL", "inputSchema"  
  {"type":"object","properties":  
    {"url":{"type":"string"}}}]}}}]}
```

- MCP client asks "what can you do?"
 - MCP server returns array of tools
 - Each tool: name, description, input JSON Schema

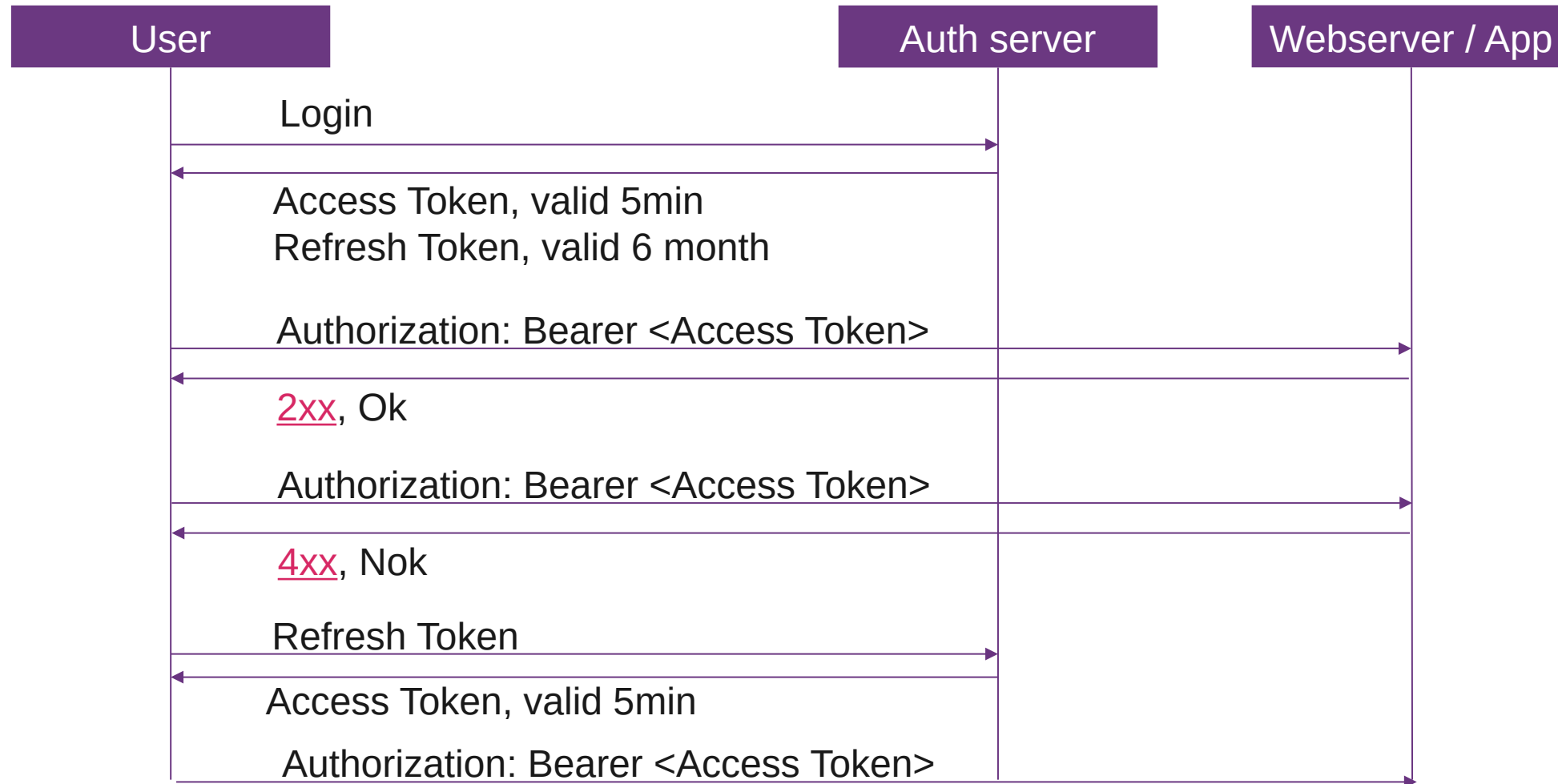
- tools/call

- JSON-RPC 2.0

```
→ {"jsonrpc":"2.0","id":2,"method":"tools/call",  
  "params":{"name":"measure_endpoint","arguments"  
    {"url":"api.example.com/health"}}}  
← {"jsonrpc":"2.0","id":2,"result":{"content"  
  [{"type":"text","text":"200 OK, 142ms"}]}}
```

- MCP client: "run this tool with these arguments"
 - MCP server executes, returns result
 - Result is an array of content blocks (text, image, etc.)

Access Token / Refresh Token

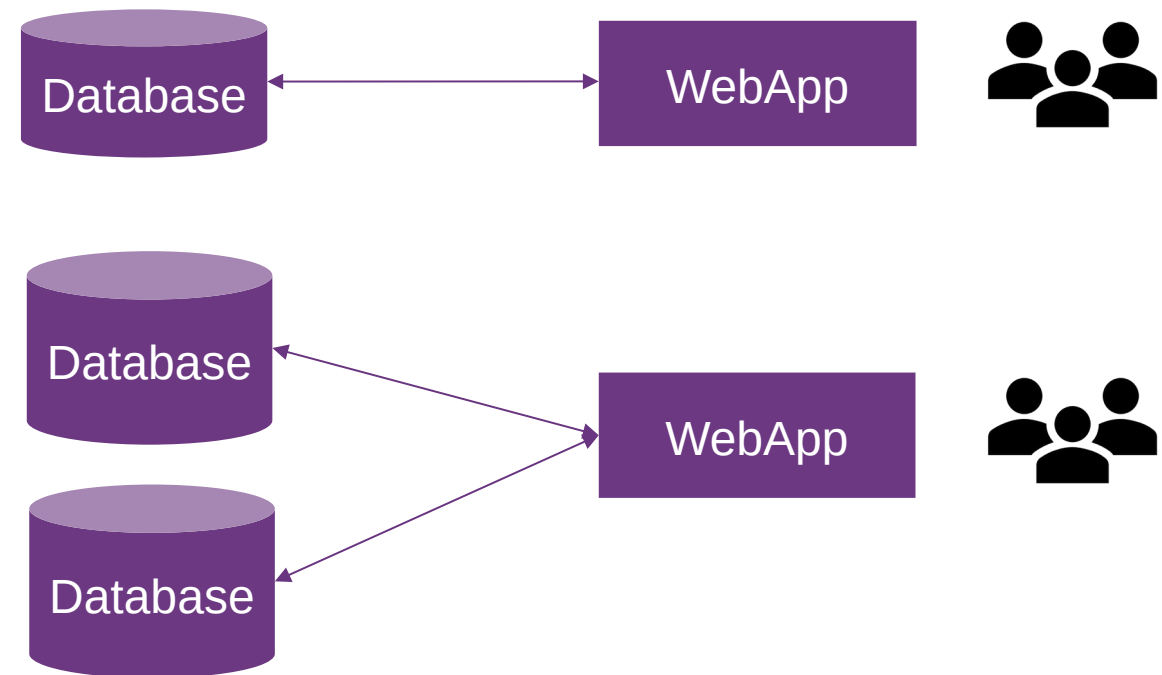


Lecture 7

Categorization

Distributed Systems Categorization

- It is useful to classify distributed systems as either tightly coupled, meaning that the processing elements, or nodes, have access to a common memory, and loosely coupled, meaning that they do not [[reference](#)]
- A homogeneous system is one in which all processors are of the same type; a heterogeneous system contains processors of different types
- Small-scale system: WebApp + database vs. large-scale with more than 2 machines
- A stateful system maintains state between requests (e.g., databases, session-based applications); a stateless system treats each request independently (e.g., REST APIs behind a load balancer)
- Decentralized vs. distributed
 - Decentralized ~ distributed in the technical sense, but not owned by one actor



Distributed Systems Categorization

- Spring Term – Distributed Systems (DSy)
 - Tightly/loosely coupled
 - Heterogeneous systems
 - Small-scale systems
 - Distributed systems
- Fall Term – Blockchain (BlCh)
 - Loosely coupled
 - Heterogeneous systems
 - Large-scale systems
 - Decentralized systems

(we will introduce blockchains in this lecture)

(we will recap distributed systems in this lecture, but DSy is highly recommended)

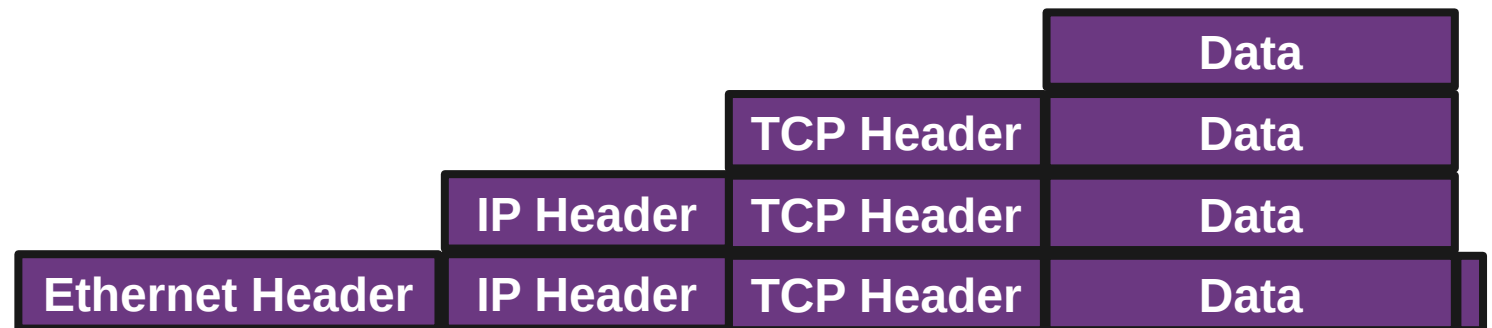
Lecture 8

Communication Protocols

Networking: Layers

- Networking: Each vendor had its own proprietary solution - not compatible with another solution
 - [IPX/SPX](#) – 1983, [AppleTalk](#) 1985, [DECnet](#) 1975, [XNS](#) 1977
- Nowadays most vendors build compatible networks hardware/software
 - Cisco, Dell, HP, Huawei, Juniper, Lenovo, Netgear, MicroTik, Ubiquiti, etc.
- Goal of layers: interoperability
 - 1984: ISO 7498 - The Basic Reference Model for Open Systems Interconnection

OSI model	"Internet model"
Application	Application
Presentation	
Session	
Transport	Transport
Network	Internet
Data link	Link
Physical	



Lecture 9

Application Protocols

Application Protocol: HTTP

- HTTP ([HyperText Transfer Protocol](#)): foundation of web data communication
- History: 1989 (Tim Berners-Lee)
 - HTTP/1.1 (1997) → HTTP/2 (2015, header compression, multiplexing) → HTTP/3 (2022, QUIC-based)
- Request/response model, resources identified by URL
- URL structure: scheme, user info, host, port, path, query, fragment

- HTTP1: text-based protocol

```
openssl s_client -connect dsl.i.ost.ch:443 -showcerts
... TLS handshake ...
GET / HTTP/1.1
```

- Stateless (server keeps no state)
- Browser sends additional headers (User-Agent, Accept, Accept-Encoding, etc.)

▼ Request Headers (359 B)

```
Host: dsl.hsr.ch
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate, br
DNT: 1
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0
TE: Trailers
```

Scheme User info Host Port Path Query Fragment



http://tbocek:password@dsl.i.ost.ch:443/lect/fs21?id=1234&lang=de#topj

Application Protocol: HTTP

- Response: header + status code + content body

- Header

```
▼ Response Headers (227 B)
HTTP/2 200 OK
server: cloudflare-nginx
content-type: text/html; charset=UTF-8
date: Mon, 02 Mar 2020 14:29:39 GMT
x-page-speed: 1.13.35.2-0
cache-control: max-age=0, no-cache
content-encoding: gzip
X-Firefox-Spdy: h2
```

- Status codes: 1xx (informational), 2xx (success – 200 OK), 3xx (redirection), 4xx (client error – 404 Not Found, 403 Forbidden), 5xx (server error), [\[list\]](#)

- Content

```
<!DOCTYPE html>
<html>
<head>
  <title>Distributed Systems and Ledgers Lab</title>
  <link rel="stylesheet" type="text/css"
href="design/layout.css"/>
...
```

- [Request methods](#): GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT, PATCH
- Web server [one-liner](#) - with netcat:
 - while true; do echo -e "HTTP/1.1 200 OK\r\n\r\n<h1>Hallo</h1>" | nc -vln -p 8080; done
- Every major browser has dev tools to show request / responses
 - Firefox, Chrome: ctrl+shift+i / F12
 - Used regularly

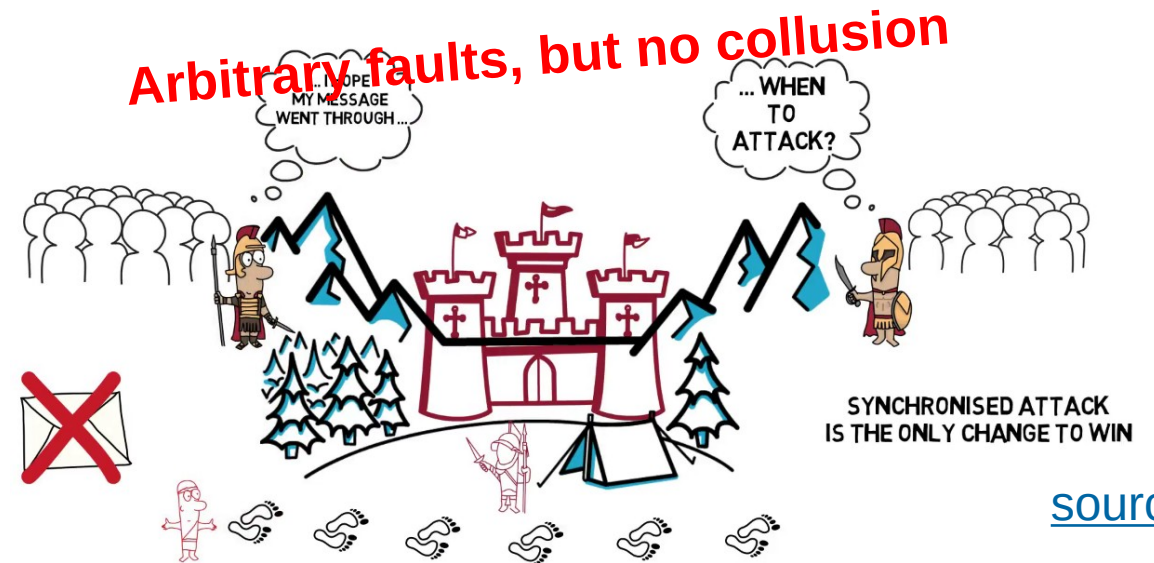
Lecture 11

Consensus, Deployment

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 to reach an advantage
- “Controlled” Distributed Systems: your own nodes, your control, no collusion
 - $2f + 1$ vs $3f + 1$
- Find consensus: Paxos, Raft, vDHT, Zookeeper

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



Practical Deployment

- Containerization as basis
 - [Ansible](#) ([Progress Chef](#), [Puppet](#)) - and [more](#)
 - Playbooks with ssh host list – your host should run the same OS (apt/yum)
 - Docker Swarm
 - Works with docker-compose.yml – with docker you package your application the same way on any platform - [simple](#)
 - Kubernetes
 - Widespread
 - Plain docker (compose) / podman
 - Simple

```
#!/usr/bin/env bash

export DOCKER_HOST="ssh://user@host"
docker compose up -d --build
```

- Ansible ([intro](#))
 - No agents running (unlike Progress Chef, Puppet)
 - Push-based system
 - ssh host list
 - Playbook
 - Run it: `ansible-playbook playbook.yml`
- More basic:
 - [pssh](#) / ssh

```
[webservers]
mwivmweb01
mwivmweb02
```

```
---
- name: Playbook
  hosts: webservers
  become: yes
  become_user: root
  tasks:
    - name: ensure apache is at the latest version
      yum:
        name: httpd
        state: latest
    - name: ensure apache is running
      service:
        name: httpd
        state: started
```

Lecture 12

Bitcoin

Introduction

- Bitcoin is an experimental digital currency
 - Bitcoin is fully peer-2-peer (no central entity)
 - 1st Bitcoin issued on January 3, 2009
 - Smallest unit: 0.00000001 BTC (1 satoshi)
- Key characteristics
 - **Maximum** of ~21 million BTC
 - Every transaction broadcast to all peers
 - Every peers knows all transactions (~660 GByte as of today)
 - Validation by proof-of-work (partial hash collision)
 - Difficult to fake proof-of-work
 - No double-spending
- The initiator is unknown so far



```
draft@home: /scratch/bitcoin/blocks
File Edit View Search Terminal Help
blk000000.dat blk000002.dat blk000004.dat blk000006.dat blk000008.dat
blk000001.dat blk000003.dat blk000005.dat blk000007.dat blk000009.dat
draft@home:/scratch/bitcoin/blocks$ head -c 300 blk000000.dat | hexdump -C
00000000 f9 be b4 d9 1d 01 00 00 01 00 00 00 00 00 00 00 | .....|
00000010 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000020 00 00 00 00 00 00 00 00 00 00 00 00 3b a3 ed fd | .....;...|
00000030 7a 7b 12 b2 7a c7 2c 3e 67 76 8f 61 7f c8 1b c3 | z{..z.,>gv.a...|
00000040 88 8a 51 32 3a 9f b8 aa 4b 1e 5e 4a 29 ab 5f 49 | ..Q2:...K.^J)..I|
00000050 ff ff 00 1d 1d ac 2b 7c 01 01 00 00 00 01 00 00 | .....+|.....|
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | .....|
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ff ff | .....|
00000080 ff ff 4d 04 ff ff 00 1d 01 04 45 54 68 65 20 54 | ..M.....EThe T|
00000090 69 6d 65 73 20 30 33 2f 4a 61 6e 2f 32 30 30 39 | imes 03/Jan/2009|
000000a0 20 43 68 61 6e 63 65 6c 6c 6f 72 20 6f 6e 20 62 | Chancellor on b|
000000b0 72 69 6e 6b 20 6f 66 20 73 65 63 6f 6e 64 20 62 | rink of second b|
000000c0 61 69 6c 6f 75 74 20 66 6f 72 20 62 61 6e 6b 73 | ailout for banks|
000000d0 ff ff ff ff 01 00 f2 05 2a 01 00 00 00 43 41 04 | .....*....CA.|
000000e0 67 8a fd b0 fe 55 48 27 19 67 f1 a6 71 30 b7 10 | g....UH'.g..q0..|
000000f0 5c d6 a8 28 e0 39 09 a6 79 62 e0 ea 1f 61 de b6 | \..(.9..yb....a..|
00000100 49 f6 bc 3f 4c ef 38 c4 f3 55 04 e5 1e c1 12 de | I..?L.8..U.....|
00000110 5c 38 4d f7 ba 0b 8d 57 8a 4c 70 2b 6b f1 1d 5f | \8M....W.Lp+k..._|
00000120 ac 00 00 00 00 f9 be b4 d9 d7 00 00 | .....|
0000012c
draft@home:/scratch/bitcoin/blocks$
```

Bitcoin - Introduction

- Not relying on trust, but on strong cryptography
- Weak anonymity (pseudonymity)
 - All peers know all transactions
 - **Clustering**: e.g. if a transaction has multiple input addresses, assume those addresses belong to the same wallet. (**example**)
- Not controlled by a single entity
 - Development community, no central bank – forks – Bitcoin Cash, SV
- **BIP**: Bitcoin Improvement Proposals
- Bitcoins can be exchange for real currencies
 - Several companies allow to exchange BTC for Dollar, Euro, ...

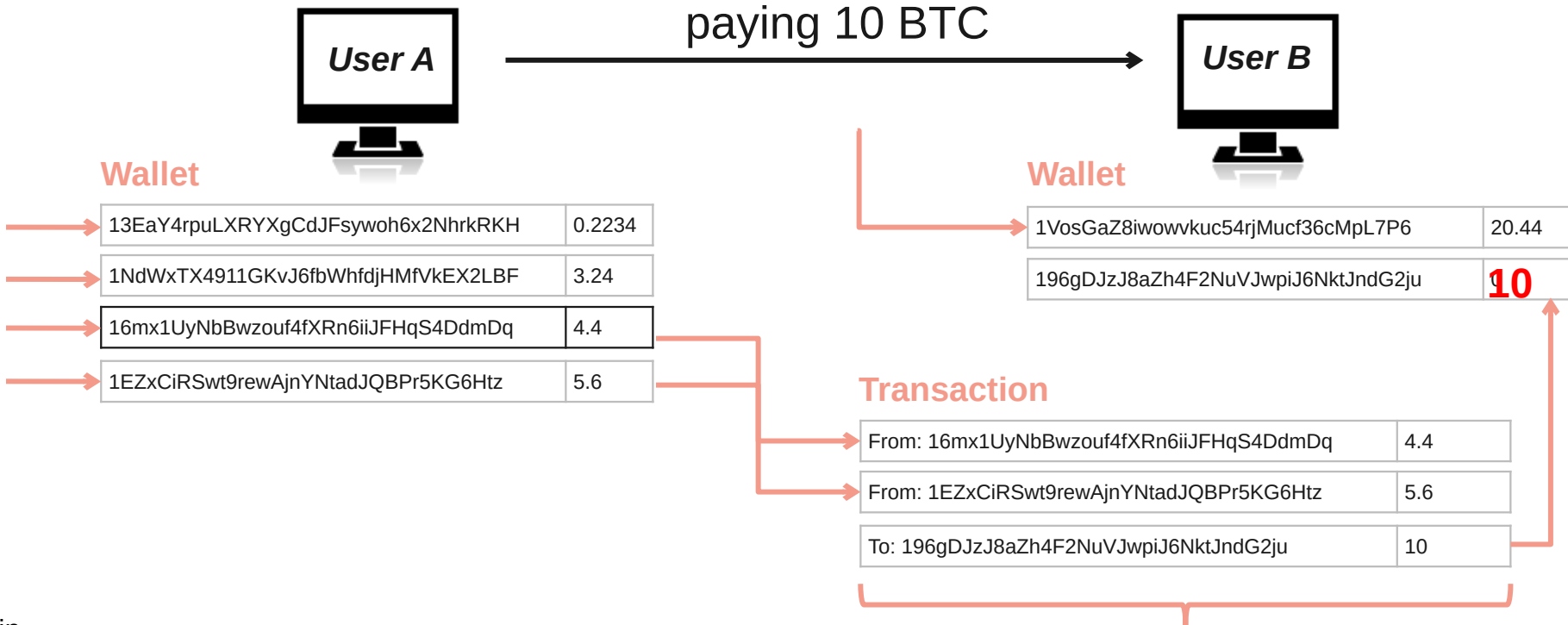
Mechanism



- A wallet has public-private keys (wallet.dat)
 - Public key, ECDSA 256 bit → Bitcoin address (can receive bitcoins)
 - Simple address ~ bech32("bc", 0x00, RIPEMD160(SHA256(compressed ECDSA public key)))
 - E.g. P2WPKH - bc1qw508d6qejxtdg4y5r3zarvary0c5xw7kv8f3t4
 - Private key used for signing transactions
- Transaction
 - Peer A wants to send BTC to peer B → creates transaction message
 - Transaction contains input / output
 - where the BTC came from and where it goes
 - Peer A broadcasts the transaction to all the peers in the network
 - Transaction stored in blocks → block is created / verified ~10min

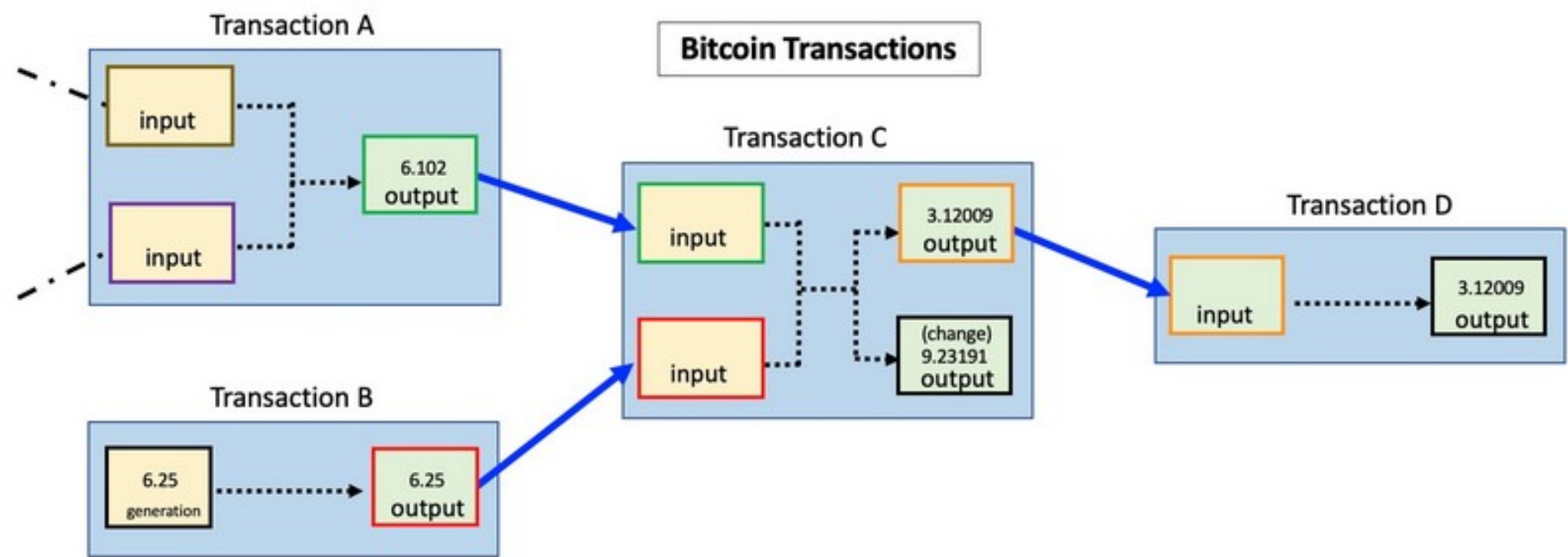
Key Bitcoin Operations

- Private key authorizes the transaction (“access”)
 - 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



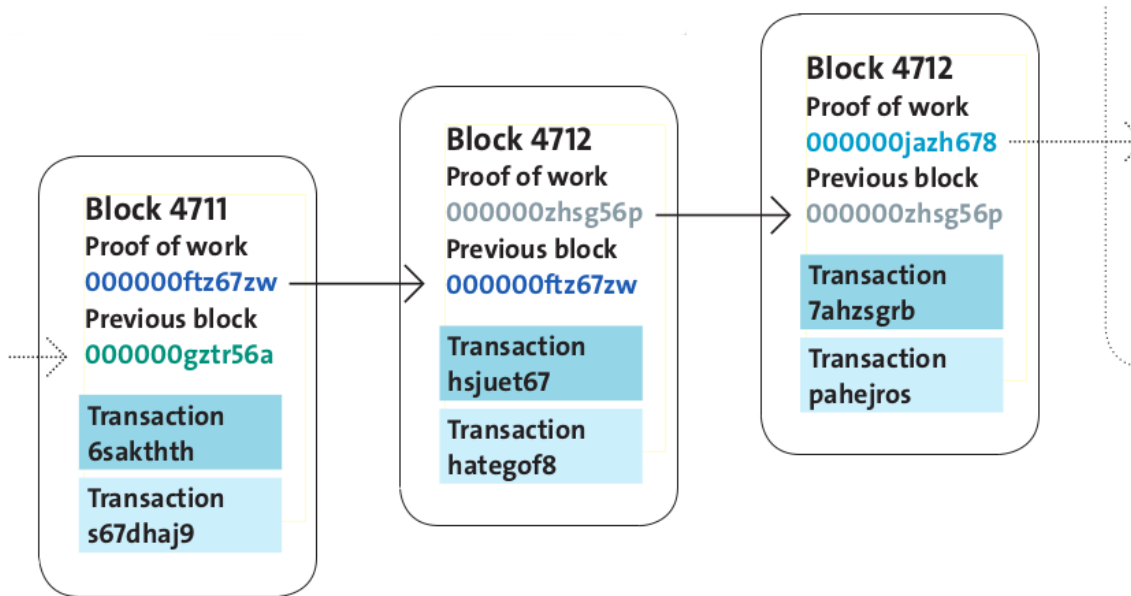
Sign with Private Key of User A

Transactions



<https://en.bitcoin.it/wiki/Transaction>

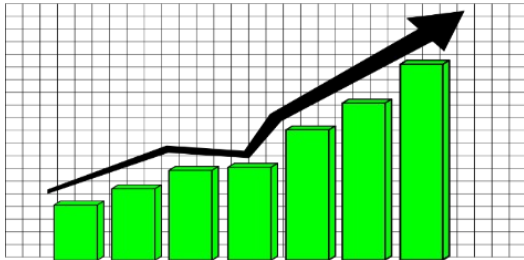
Blockchain



- Transactions are collected in blocks
 - New block created approximately every 10 min
- Blocks contain solved crypto puzzles
 - In the form of partial **hash collisions** (SHA256)
- A block has a pointer to previous **block** → **Blockchain**
- Creation of blocks is called mining (reward)
 - Mining / creating blocks → Miner get currently 3.125 BTC per creation
 - **adjustable difficulty** 6 blocks / h
 - Sometime in 2028 reward will be **1.5625**

Discussion (1)

- Disadvantages
 - Power consumption
 - ~ as much as Poland
 - Not scalable
 - Bitcoin with ~7 tps vs. VISA 57,000 tps (23.12)
[tps: transactions per sec]



- Anonymity
 - Can be used for illegal activities

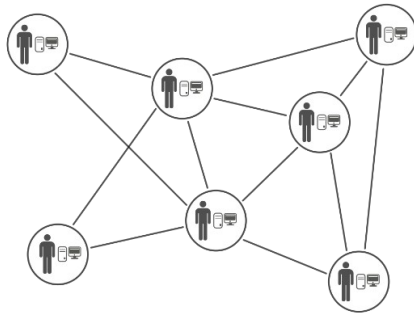
- Advantages
 - Low (fixed) tx fees
 - ~0.6 satoshi per byte / 0.07USD (~140bytes tx)
 - Scalable
 - Hardware/storage gets faster



- Anonymity
 - Preserving privacy

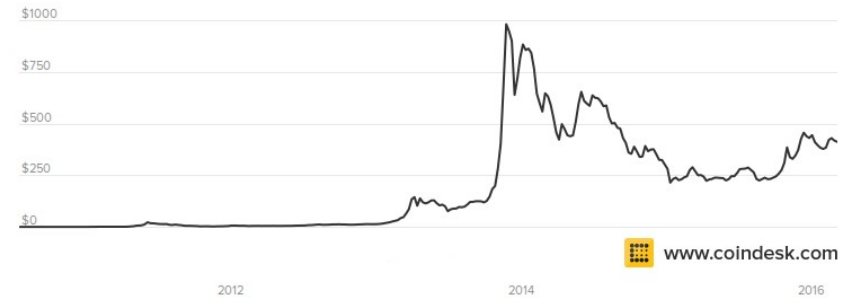
Discussion (2)

- Advantages
 - No major “crashes”
 - [Mt.Gox](#) / FTX was exchange site!
 - Decentralized
 - Open protocol
 - Forks



- Many other blockchain use cases
 - Smart contracts

- Disadvantages
 - Volatile exchange rate



- Central elements
 - Core developers
 - Mining farms [link]

