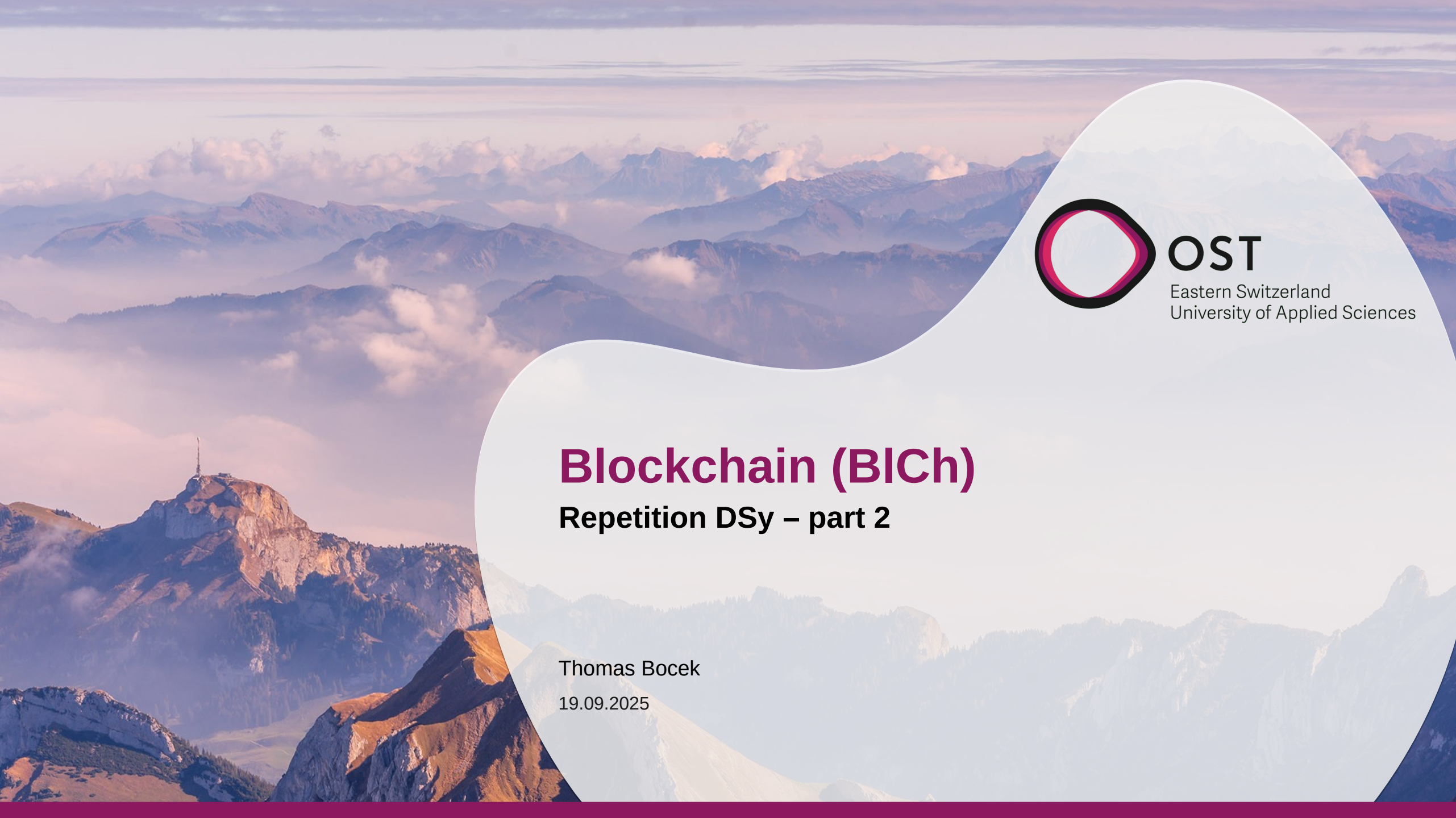


OST
Eastern Switzerland
University of Applied Sciences

# Blockchain (BlCh)

## Repetition DSy – part 2

Thomas Bocek

19.09.2025

# Lecture 7

# Distributed Systems Categorization

**“Controlled” Distributed Systems**

- 1 responsible organization
- Low churn
- Examples:
  - Amazon DynamoDB
  - Client/server
- “Secure environment”
- High availability
- Can be homogeneous / heterogeneous

**“Fully” Decentralized Systems**

- N responsible organizations
- High churn
- Examples:
  - BitTorrent
  - Blockchain
- “Hostile environment”
- Unpredictable availability
- Is heterogeneous



OST

# Distributed Systems Categorization

**“Controlled” Distributed Systems**

- Mechanisms that work well:
  - Consistent hashing (DynamoDB, Cassandra)
  - Master nodes, central coordinator
- Network is under control or client/server → no NAT issues

**“Fully” Decentralized Systems**

- Mechanisms that work well:
  - Consistent hashing (DHTs)
  - Flooding/broadcasting - Bitcoin
- NAT and direct connectivity huge problem

# Distributed Systems Categorization

## “Controlled” Distributed Systems

- Consistency
  - Leader election (Zookeeper, Paxos, Raft)
- Replication principles
  - More replicas: higher availability, higher reliability, higher performance, better scalability, but: requires maintaining consistency in replicas
- Transparency principles apply

## “Fully” Decentralized Systems

- Consistency
  - Weak consistency: DHTs
  - Nakamoto consensus (aka proof of work)
  - Proof of stake – Leader election, PBFT protocols - Is Bitcoin eventually consistent?
    - Some argue no, some argue it has even stronger guarantees
- Replication principles apply to fully decentralized systems as well
- Transparency principles apply



OST

# Distributed Systems Categorization

- Spring Term – Distributed Systems (DSy)
  - Tightly/loosely coupled
  - Heterogeneous systems
  - Small-scale systems
  - Distributed systems

  (we will also talk about blockchains in this lecture)

- Fall Term – Blockchain (BlCh)
  - Loosely coupled
  - Heterogeneous systems
  - Large-scale systems
  - Decentralized systems

  (we will also talk about distributed systems in this lecture, but DSy is highly recommended)

# Lecture 8

# Access Token / Refresh Token

- Access Token only short lifetime, e.g., 10min.
  - If public key / secret is known, the content in the token can be trusted, e.g., in the serivce
  - Can have userId, role, etc.
    - No need to query DB for those information, e.g.:

```
type TokenClaims struct {
     MailFrom string `json:"mail_from,omitempty"`
     MailTo   string `json:"mail_to,omitempty"`
     jwt.Claims
}
```

- Refresh Token longer lifetime, e.g., 6 month
  - A refresh token is used to get a new access token
  - IAM / Auth server creates access tokens

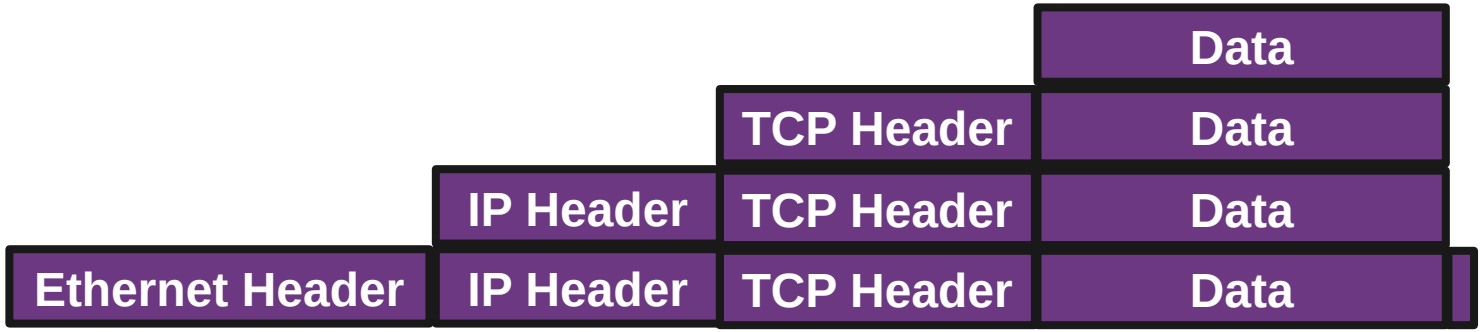
- Only access token, with long lifetime
  - If a user credential is revoked – how to inform every service?
- Only refresh token
  - Tightly coupled Service/Auth, every request to Service, Auth needs to be involved for every access
- Access + Refresh token
  - If a user credential is revoked, user has max. 10min more to access service
  - Auth only involved if access token is expired

# Lecture 9

# 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 from different vendors
  - Cisco, Dell, HP, Huawei, Juniper, Lenovo, Linksys, Netgear, MicroTik, Siemens, 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 |
| Pysical | |



# TCP/IP from an Application Developer View

- Server in golang (repo)
  - git clone https://github.com/tbocek/DSy
  - Download GoLand, or others
  - go run server.go → server
- Listening on TCP port 8081
  - Return string in uppercase
- Node.js version
  - Download WebStorm, or other
- Client:
  - nc localhost 8081

```
const net = require('net');
const server = new net.Server();
server.listen(8081, function() {
   console.log('Launching server...');
});

server.on('connection', function(socket) {
   socket.on('data', function(chunk) {
      console.log(`Data received from client: $
{chunk.toString()}`);

socket.write(chunk.toString().toUpperCase() +
"\n");
   });
});
```

```
package main
import ("bufio"
    "fmt"
    "net"
    "strings")
func main() {
    fmt.Println("Launching server...")
    ln, _ := net.Listen("tcp", ":8081") // listen
on all interfaces
    for {
        conn, _ := ln.Accept() // accept connectior
on port
        message, _ :=
bufio.NewReader(conn).ReadString('\n') //read line
        fmt.Print("Message Received:",
string(message))
        newMessage := strings.ToUpper(message)
//change to upper
        conn.Write([]byte(newMessage + "\n"))
//send upper string back
    }
}
```

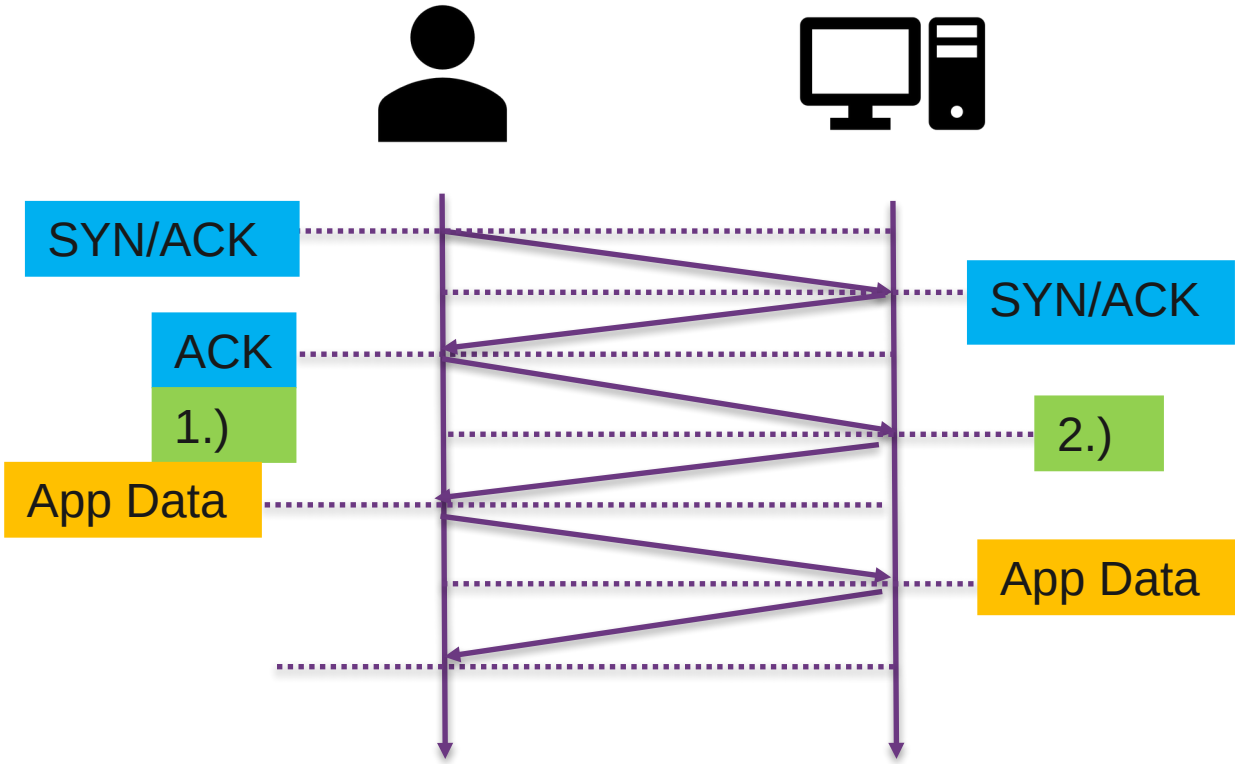


OST

```
PING sydney.edu.au (129.78.5.8) 56(84) bytes of data.
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=1 ttl=233 time=307 ms
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=2 ttl=233 time=305 ms
64 bytes from scilearn.sydney.edu.au (129.78.5.8): icmp_seq=3 ttl=233 time=305 ms
```

# Layer 4 – TCP + TLS

- Ping to Australia: 329ms
  - One way ~ 165ms
- TCP + TLS handshake:
  - 3RTT = 987ms! No data sent yet
- TLS 1.3, finished Aug 2018
  - 1 RTT instead of 2
    - 1.) Client Hello, Key Share
    - 2.) Server Hello, key Share, Verify Certificate, Finished
  - 0 RTT possible, for previous connections, loosing perfect forward secrecy
  - 90% of browsers used already support it

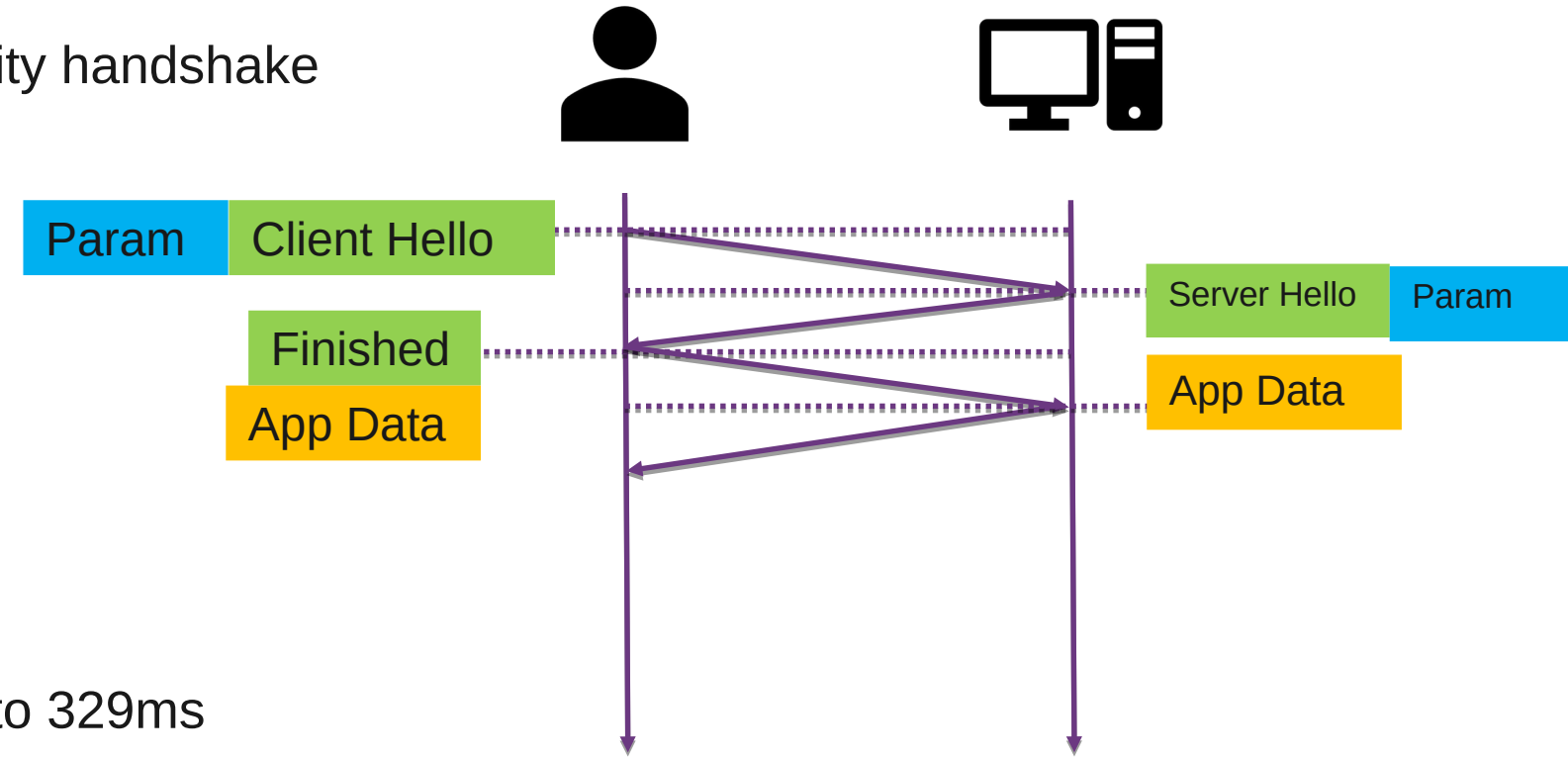





OST

# QUIC / HTTP/3

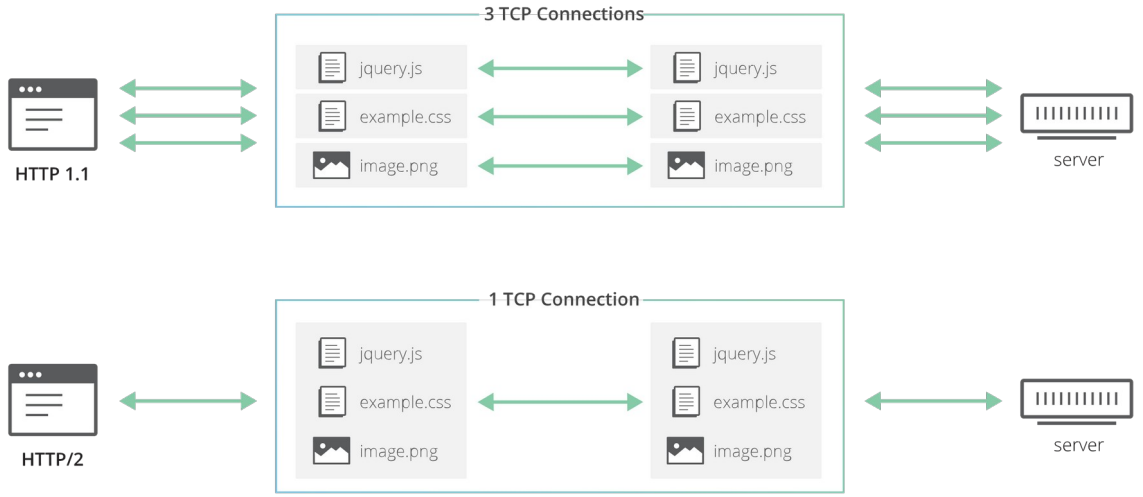
- QUIC: 1RTT connection + security handshake
  - For known connections: 0RTT
  - Built in security
  - “Google's 'QUIC' TCP alternative slow to excite anyone outside Google” [link] (9%, 25%, 75%)
    - Facebook
    - Cloudflare, state of HTTP
- Example Australia: from 987ms to 329ms



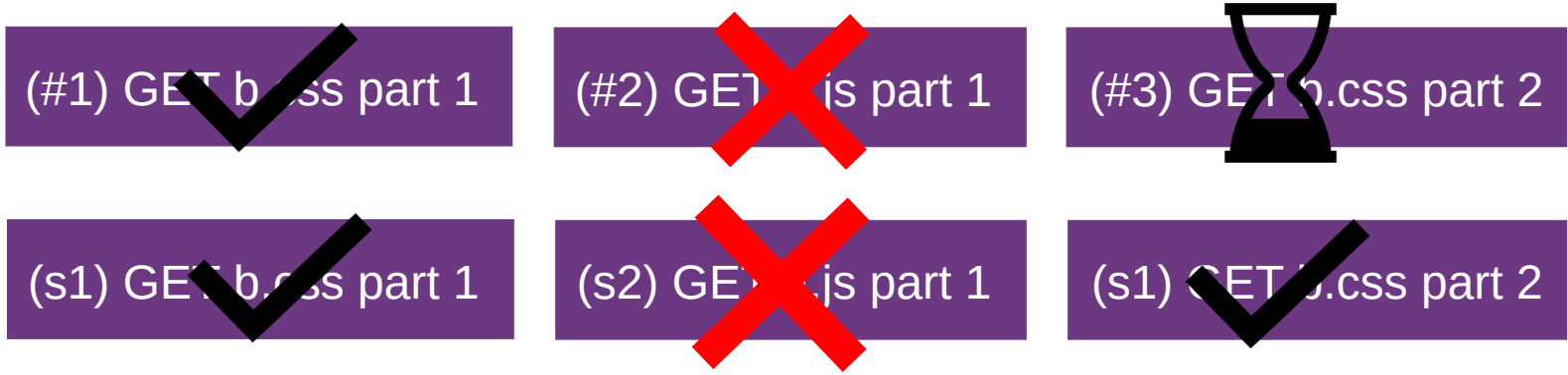

OST

# QUIC / HTTP3

- Multiplexing in HTTP/2
  - HTTP/1 → HTTP/2
- HTTP/2: Head-of-line blocking
  - One packet loss, TCP needs to be ordered
  - QUIC can multiplex requests: one stream does not affect others
- HTTP/3 is great, but…
  - NAT → SYN, ACK, FIN, conntrack knows when connection ends, not with QUIC, timeouts, new entries, many entries
  - HTTP header compression, referencing previous headers
  - Many TCP optimizations




source: https://blog.cloudflare.com/the-road-to-quic/



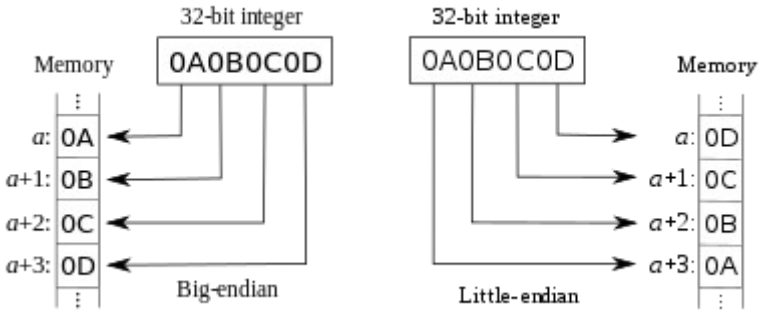
# Lecture 10

# Protocols

- Custom encoding/decoding
  - You control every aspect
  - You send more time on it

- Little-endian / Big-endian
  - sequential order where bytes are converted into numbers
- Networking, e.g. TCP headers: Big-endian
- Most CPUs e.g., x86: Little-endian, RISC-V: Bi-endianness

```
115     public static boolean decodeHeader(final ByteBuf buffer, final InetSocketAddress recipientSocket,
116             final InetSocketAddress senderSocket, final Message message) {
117         LOG.debug("Decode message. Recipient: {}, Sender:{}.", recipientSocket, senderSocket);
118         final int versionAndType = buffer.readInt();
119         message.version(versionAndType >>> 4);
120         message.type(Type.values()[(versionAndType & Utils.MASK_0F)]);
121         message.protocolType(ProtocolType.values()[versionAndType >>> 30]);
122         message.messageId(buffer.readInt());
123         final int command = buffer.readUnsignedByte();
124         message.command((byte) command);
125         final Number160 recipientID = Number160.decode(buffer);
126
127         //we only get the id for the recipient, the rest we already know
128         final PeerAddress recipient = PeerAddress.builder().peerId(recipientID).build();
129         message.recipient(recipient);
130
131
132         final int contentTypes = buffer.readInt();
133         message.hasContent(contentTypes != 0);
134         message.contentTypes(decodeContentTypes(contentTypes, message));
```




[source]



OST

# Application Protocol: HTTP

- HTTP (HyperText Transfer Protocol): foundation of data communication for www
- Started in 1989 by Tim Berners-Lee
  - HTTP/1.1 published in 1997
  - HTTP/2 published in 2015
    - More efficient, header compression, multiplexing
  - HTTP/3 published in 2022
- Request / response (resource)
- HTTP resources identified by URL
  - https://dsl.i.ost.ch/design/ost_logo.svg

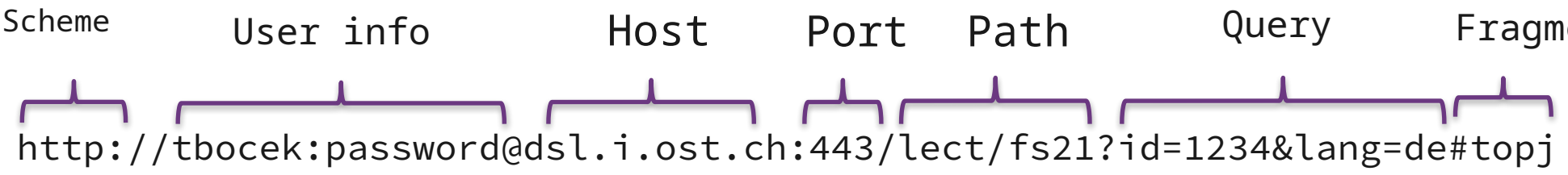
- Text-based protocol

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

- HTTP is a stateless protocol
  - Server maintains no state
- Browser sends a bit more…

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
```






OST

# Lecture 12

# Deployment Strategies

- Many strategies and variations [link, link, link]
- Rolling Deployment
  - New version is gradually deployed to replace the old version - without taking the entire system down at once
    + Minimal downtime, low risk
    - Complexity, longer deployment times
- Blue-Green Deployment
  - 2 environments, current prod (blue), current prod with new release (green). Test, then switch
    + Instant rollback, 0 downtime
    - 2 prod environments, keep data in sync
- Canary Releases
  - Canary in a coal mine - new version to a small group of users or servers first, if all goes well, more users
    + Risk reduction, user feedback
    - Complexity, inconsistencies
- Feature Toggle
  - Fine grained canary, set feature for specific users
    + More risk reduction, specific user feedback
    - Increase complexity of codebase, config management
- Big Bang
  - Deploy everything at once
    + Simple
    - High risk, limited rollback

# Latency Numbers Every Programmer Should Know

- Interactive [link] from 1990 - 2020
  - Network stays ~ 150ms
  - L1: 1ns / branch miss 3ns – example
- HDD / SSD / NVMe (Non-Volatile Memory Express) - comparison, 2

## NVMe Latency vs. Other Storage Protocols

| Feature | NVMe | SATA SSD | HDD |
|---|---|---|---|
| **Read Latency** | ~20 µs | ~100 µs | 2-5 ms |
| **Write Latency** | ~30 µs | ~200 µs | 5-10 ms |
| **Queue Depth** | 64K queues × 64K commands | 32 commands | 1 command |
| **Bandwidth** | Up to **16GB/s (PCIe 4.0)** | 600MB/s | ~150MB/s |

- Latency and throughput important
- Napkin Math [link]
  - Cost

| Operation | Latency | Throughput | 1 MiB | 1 GiB |
|---|---|---|---|---|
| Sequential Memory R/W (64 bytes) | 0.5 ns | | | |
| ├ Single Thread, No SIMD | | 10 GiB/s | 100 µs | 100 ms |
| ├ Single Thread, SIMD | | 20 GiB/s | 50 µs | 50 ms |
| ├ Threaded, No SIMD | | 30 GiB/s | 35 µs | 35 ms |
| ├ Threaded, SIMD | | 35 GiB/s | 30 µs | 30 ms |
| Network Same-Zone | | 10 GiB/s | 100 µs | 100 ms |
| ├ Inside VPC | | 10 GiB/s | 100 µs | 100 ms |
| ├ Outside VPC | | 3 GiB/s | 300 µs | 300 ms |
| Hashing, not crypto-safe (64 bytes) | 25 ns | 2 GiB/s | 500 µs | 500 ms |
| Random Memory R/W (64 bytes) | 50 ns | 1 GiB/s | 1 ms | 1s |
| Fast Serialization [8] [9] † | N/A | 1 GiB/s | 1 ms | 1s |
| Fast Deserialization [8] [9] † | N/A | 1 GiB/s | 1 ms | 1s |
| System Call | 500 ns | N/A | N/A | N/A |
| Hashing, crypto-safe (64 bytes) | 100 ns | 1 GiB/s | 1 ms | 1s |
| Sequential SSD read (8 KiB) | 1 µs | 4 GiB/s | 200 µs | 200 ms |
| Context Switch [1] [2] | 10 µs | N/A | N/A | N/A |
| Sequential SSD write, -fsync (8KiB) | 10 µs | 1 GiB/s | 1 ms | 1s |
| TCP Echo Server (32 KiB) | 10 µs | 4 GiB/s | 200 µs | 200 ms |
| Decompression [11] | N/A | 1 GiB/s | 1 ms | 1s |
| Compression [11] | N/A | 500 MiB/s | 2 ms | 2s |
| Sequential SSD write, +fsync (8KiB) | 1 ms | 10 MiB/s | 100 ms | 2 min |
| Sorting (64-bit integers) | N/A | 200 MiB/s | 5 ms | 5s |
| Sequential HDD Read (8 KiB) | 10 ms | 250 MiB/s | 2 ms | 2s |



OST

# Lecture 13

# 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
blk00000.dat  blk00002.dat  blk00004.dat  blk00006.dat  blk00008.dat
blk00001.dat  blk00003.dat  blk00005.dat  blk00007.dat  blk00009.dat
draft@home:/scratch/bitcoin/blocks$ head -c 300 blk00000.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$
```



OST

# Bitcoin - Introduction

- Not relying on trust, but on strong cryptography
- Weak anonymity (pseudonimity)
  - 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, …
- US, CH considered Bitcoin friendly, China (energy) not that much

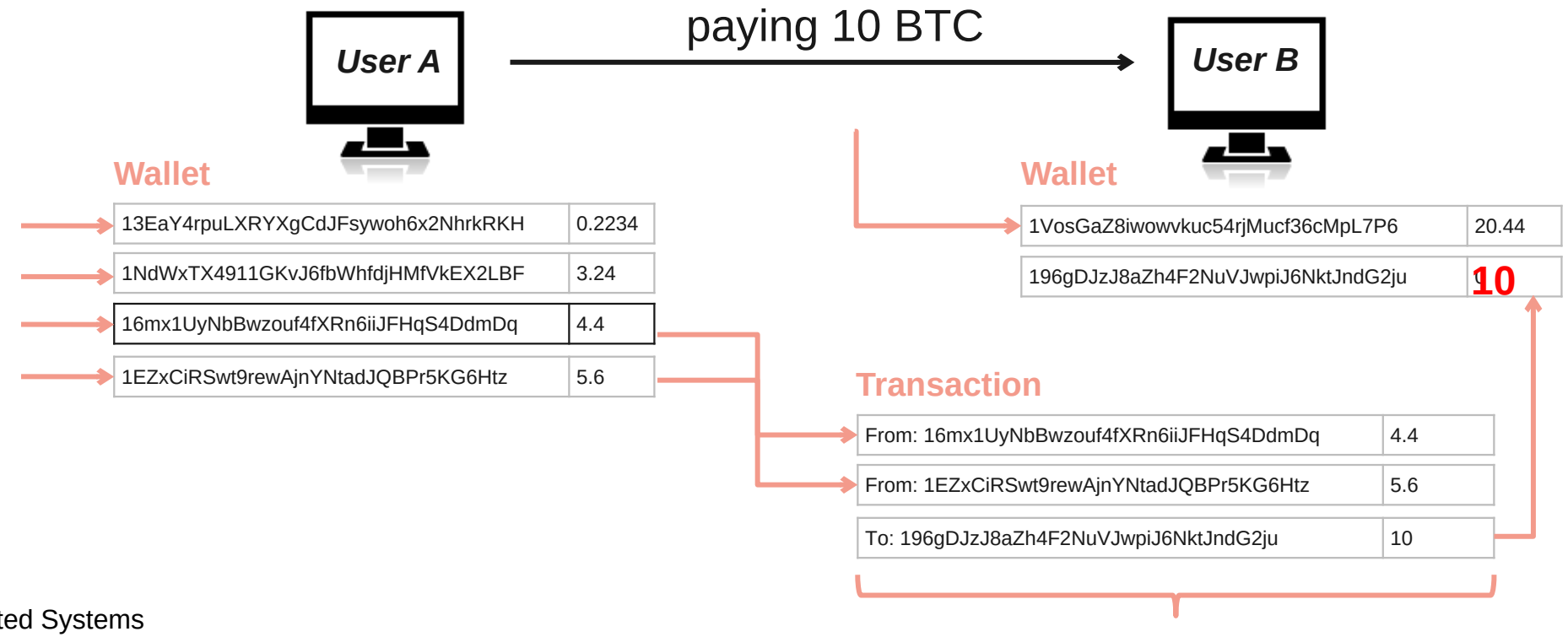
# Mechanism

- A wallet has public-private keys (wallet.dat)
  - Public key, ECDSA 256 bit → Bitcoin address (can receive bitcoins)
  - Simple address ~ base58(RIPEM160(Sha256(ecdsa public key)))
    - E.g. 1GCeaKuhDYnNLNR6LGmBtKhPqEJD4KeEtF
  - 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

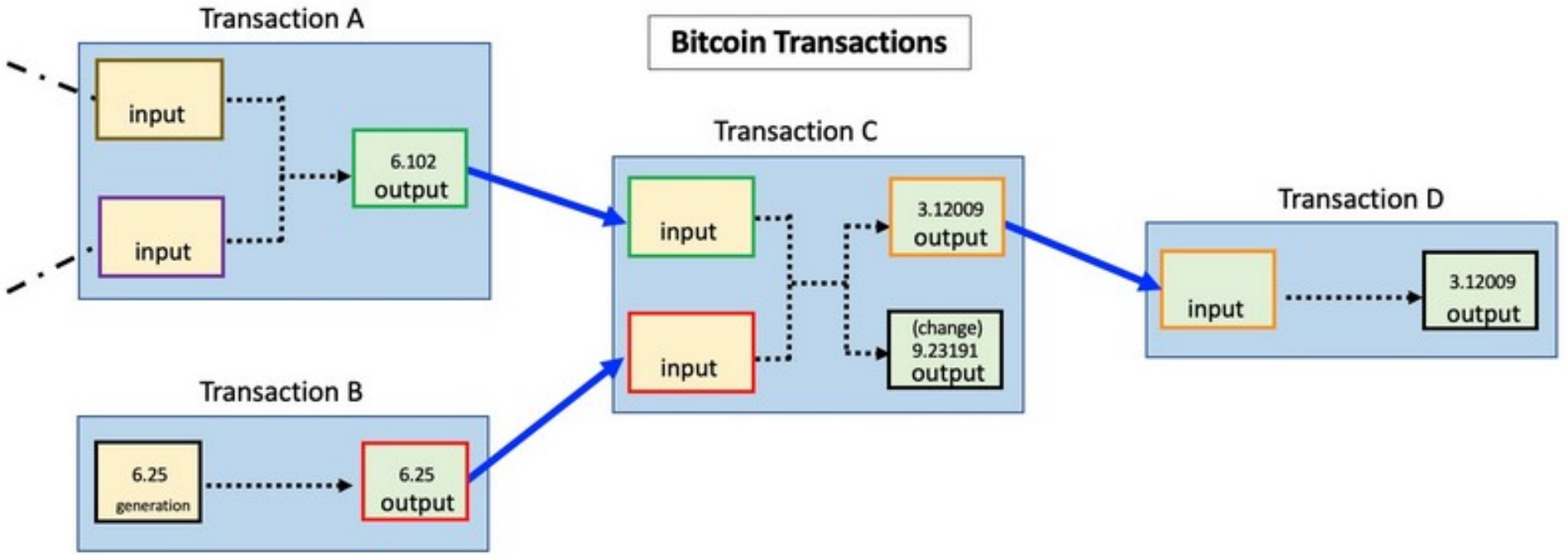
- Private key authorizes the transaction (“access“)
  - If keys are stolen, thief may use “your” coins
  - If keys are lost, coins are lost
  - In UTXO (unspent transaction output) systems, complete output is spent






OST

# Transactions



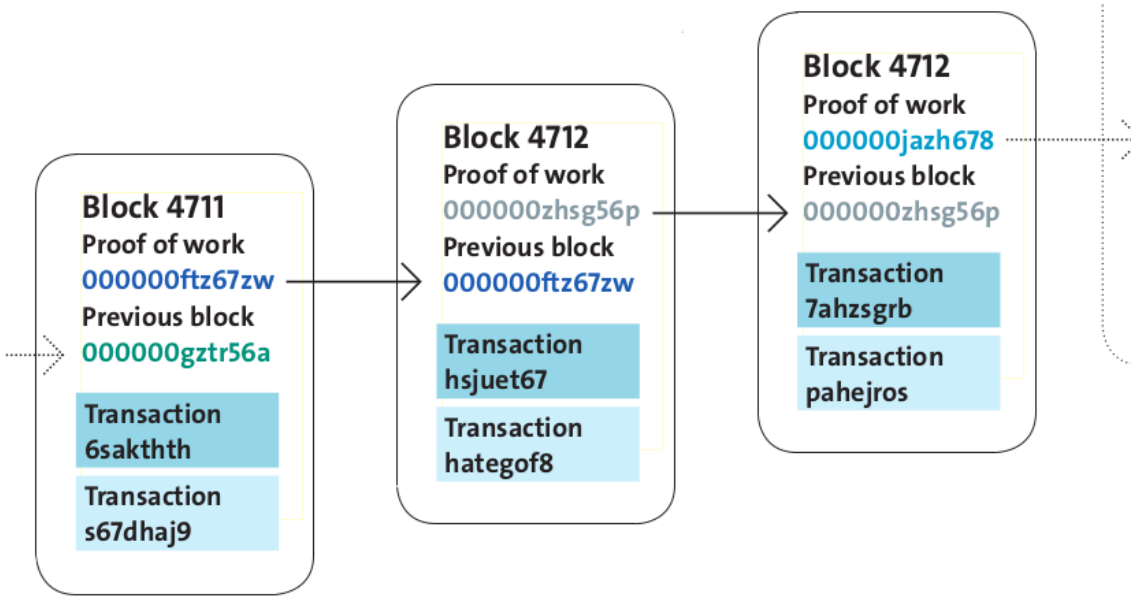

https://en.bitcoin.it/wiki/Transaction



OST

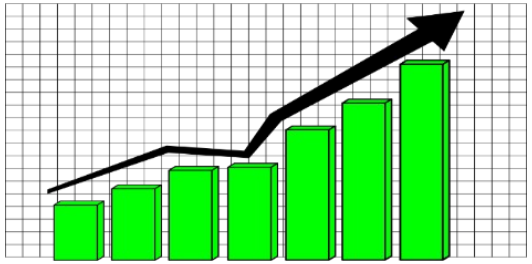
# Blockchain

- 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
    - ~1.2 satoshi per byte / 0.25USD (~200bytes tx)
  - Scalable
    - Hardware/storage gets faster

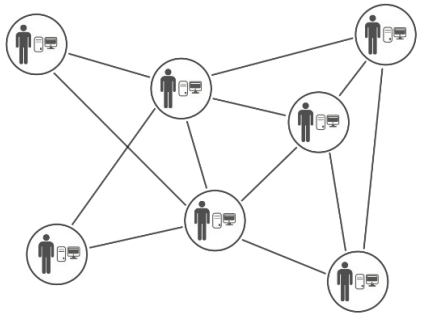

  - Anonymity
    - Preserving privacy



OST

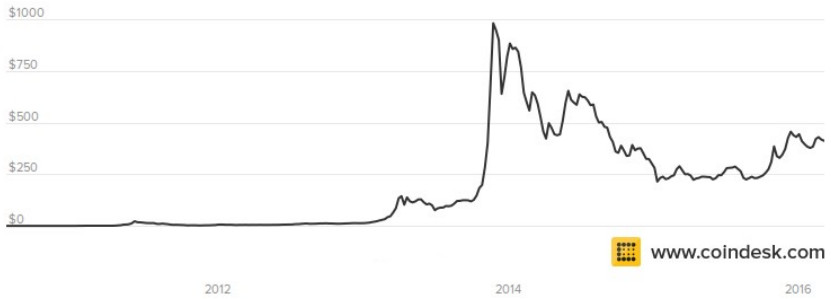
# Discussion (2)

- 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 elemements
    - Core developers
    - Mining farms [link]