OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems Advanced & Blockchain

Repetition VSS

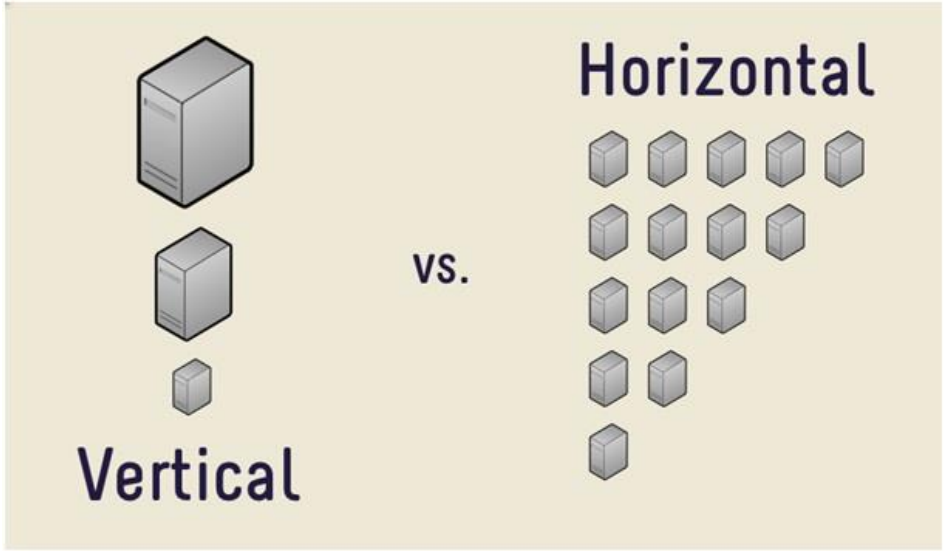
Thomas Bocek

14 September 2020

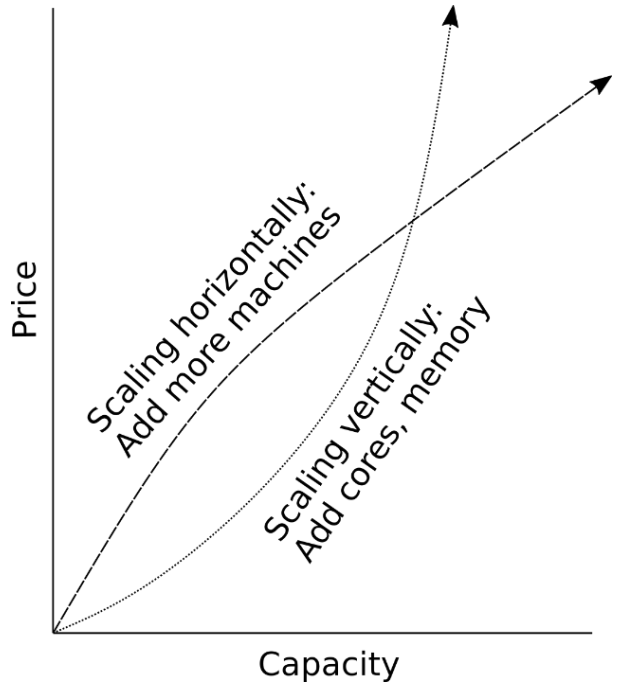
Lecture 1

Distributed Systems Motivation

- Why Distributed Systems
  - Scaling
    - Vertical (scale up), more memory, faster CPU
    - Horizontal (scale out), more machines
    - Apple has 75’000 Apache Cassandra nodes storing 10 petabytes of data in 2015



- Economics
  - Initially scaling vertically is cheaper, until you max out HW
  - Current x86 max: 64 cores (AMD)



OST

Distributed Systems Motivation

Horizontal Scaling

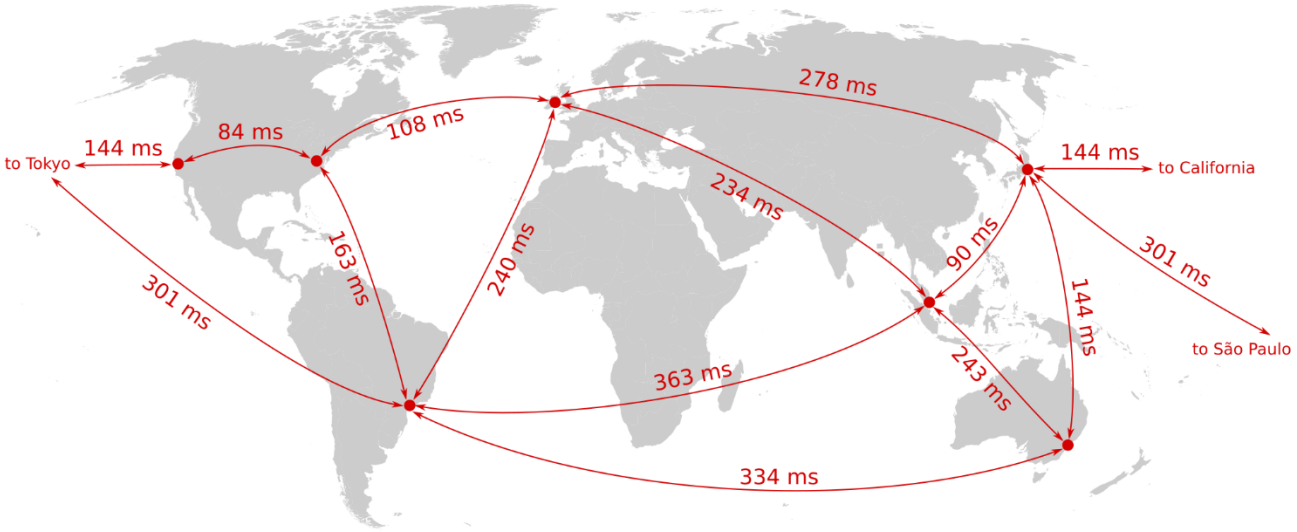
- + Lower cost with massive scale
- + Easier to add fault-tolerance
- + Higher availability
- - Adaption of software required
- - More complex system, more components involved
- - More power consumption

Vertical Scaling

- + Lower cost with small scale
- + No adaption of software required
- + Less administrative effort
- + Less power consumption
- - Risk of HW failure causing outage
- - More difficult to add fault-tolerance
- - HW limits for scaling

Distributed Systems Motivation

- Why Distributed Systems
  - Location
    - Everything gets faster, latency stays
    - Physically bounded by the speed of light



https://www.inkandswitch.com/local-first.html

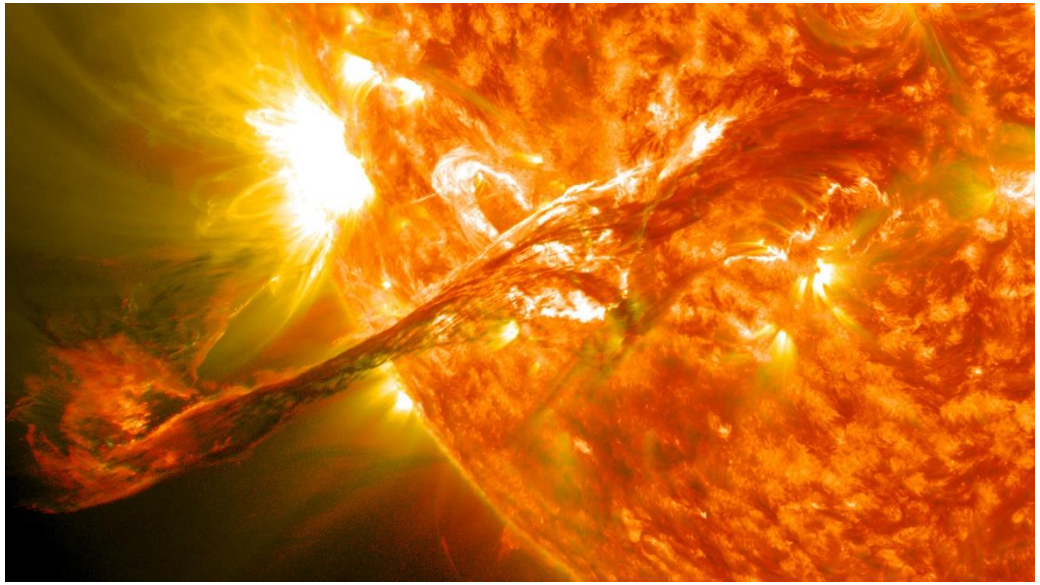
- New protocols can decrease RTT
  - Upcoming lecture
- Place services closer to user
  - Sometimes latency of 278ms is unacceptable
  - Gaming / Esports:
    - Human reaction time 200ms
    - Total from keypress to display:
      - Thinkpad 13 ChromeOS: 70ms
      - Lenovo X1 carbon 2016: 150ms
    - TV output lag ~15-30ms (random TV)
    - Keyboard 15-60ms
- CDN: Content delivery network
  - Place your images, sites, scripts close to your users

OST

Distributed Systems Motivation

- Why Distributed Systems
  - Fault-tolerance
    - Any hardware will crash eventually
- Random bit flips in memory
  - 1990: “Computers typically experience about one cosmic-ray-induced error per 256 megabytes of RAM per month”
  - Google study 2009: 1-5 bit errors per 8GB of RAM per hour
  - 2007: 44 reported memory errors (41 ECC and 3 double bit) on ~1300 nodes during a period of about 3 month
- Source
  - Cosmic rays
    - Solar flares, Coronal mass ejection, Solar proton events, Background radiation

- Cosmic rays may be blamed for an electronic voting error in Belgium (2003)
  - Bit flip in electronic voting machine
  - Added 4096 extra votes to one candidate



OST

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

Transparency in distributed systems

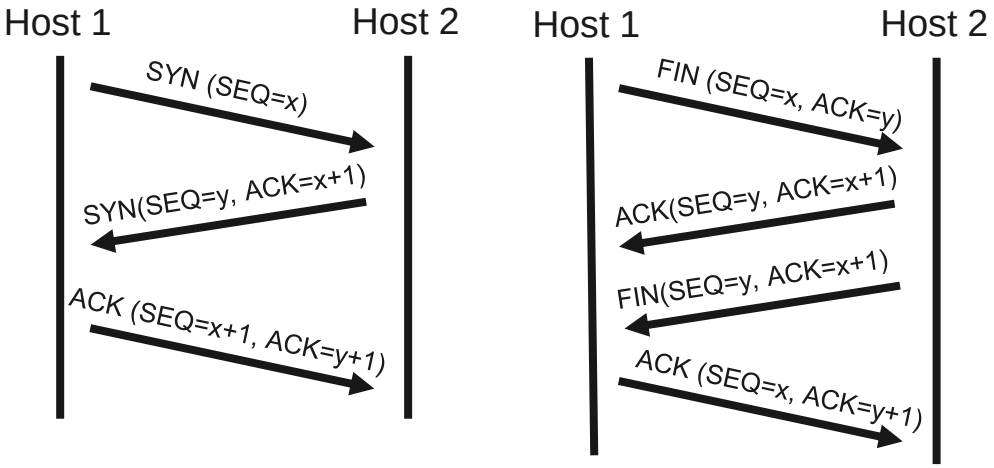
- Distributed system should hide its distributed nature
  - Location transparency – users should not be aware of the physical location
  - Access transparency - users should access resources in a single, uniform way
  - Migration, relocation transparency – users should not be aware, that resource have moved
  - Replication transparency – users should not be aware about replicas, it should appear as a single resource
  - Concurrent transparency – users should not be aware of other users
  - Failure transparency – users should be aware of recovery mechanisms
  - Security transparency – users should be minimally aware of security mechanisms
- More/other transparencies here, here, here
  - Depends on the context

OST

Lecture 2

Layer 4 - TCP

- Connection establishment
  - SYN, SYN-ACK, ACK (three way)
  - Initiates TCP session: initial sequence number is ~random
- Connection termination
  - FIN, ACK + FIN, ACK (three/four way)
  - 3-way handshake, when host 1 sends a FIN and host 2 replies with a FIN & ACK
- Sequences and ACKs
  - Identification each byte of data
  - Order of the bytes → reconstruction
  - Detecting lost data: RTO, DupACK:

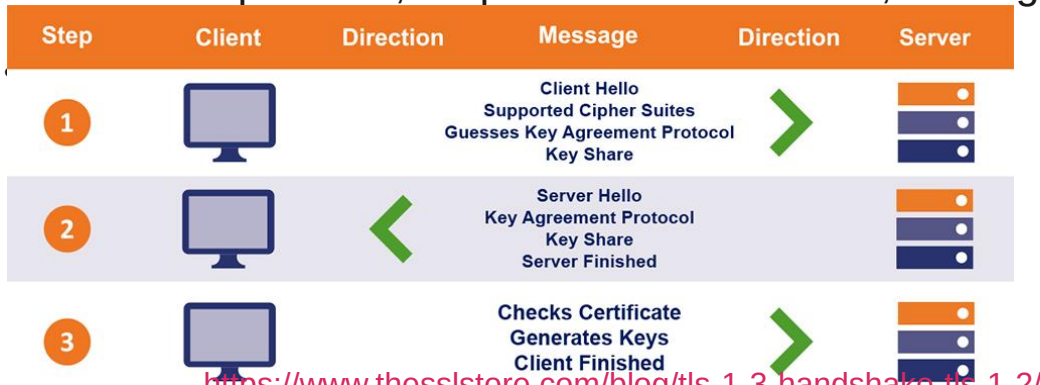


- Retransmission timeout
  - If no ACK is received aftert timout (e.g. 2xRTT), resend.
- Duplicate cumulative acknowledgements, selective ACK
  - ACKs for last consecutive packets
  - 3 times same ACK → retransmit missing packets (fast retransmit)

OST

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
  - 0 RTT possible, for previous connections, loosing perfect for



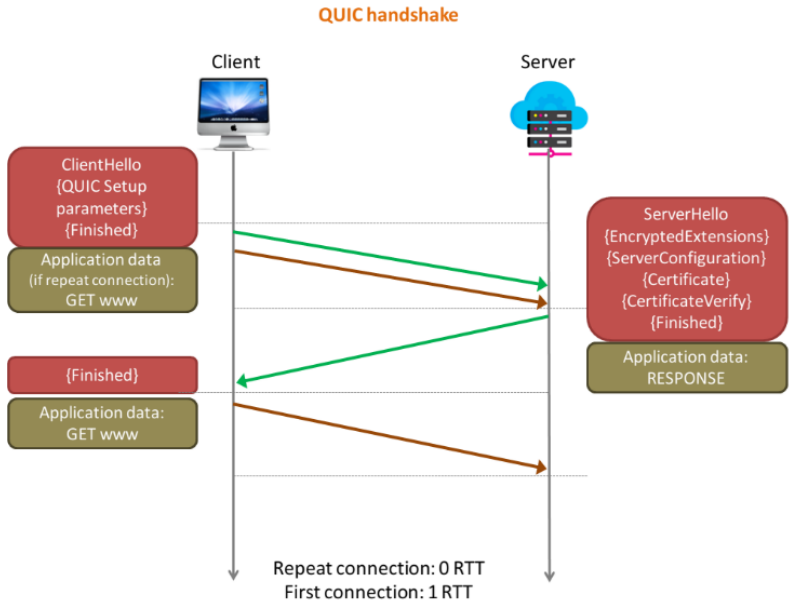
https://www.thesslstore.com/blog/tls-1-3-handshake-tls-1-2/

```
draft@schleppi: ping www.curtin.edu.au
File  Edit  View  Search  Terminal  Help
.com (52.64.39.142): icmp_seq=4 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=5 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=6 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=7 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=8 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=9 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=10 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=11 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=12 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=13 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=14 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=15 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=16 ttl=42 time=329 ms
64 bytes from ec2-52-64-39-142.ap-southeast-2.compute.amazonaws
.com (52.64.39.142): icmp_seq=17 ttl=42 time=329 ms
```

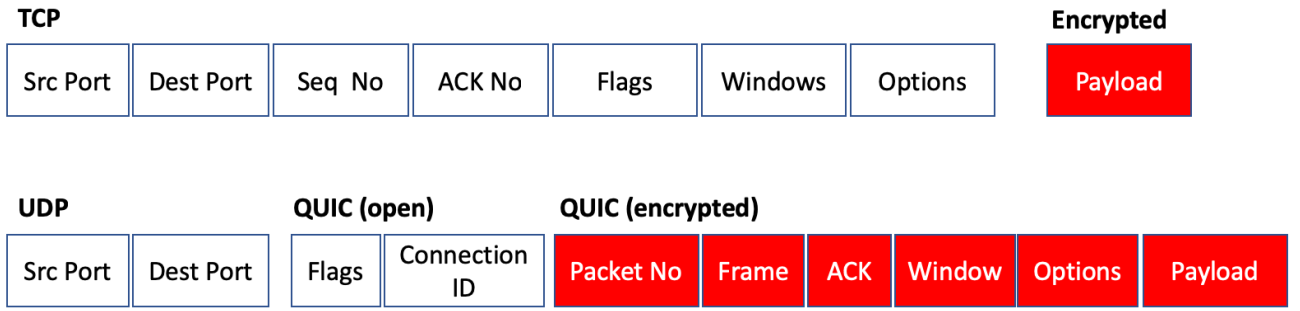

OST

QUIC

- QUIC: 1RTT (chrome example)
  - For known connections: 0RTT
  - Built in security
  - “between 2.6 per cent and 9.1 per cent of traffic (3%)” - end-user 0%




```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1|T T|X X X X|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0..160)             ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Source Connection ID (0..160)               ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```



Lecture 3

JSON example

- JSON + REST
  - Human-readable text to transmit data
  - Often used for web apps
- 187 bytes

```
func main() {
  fmt.Println("Connecting...")
  req, _ := http.NewRequest("POST", "http://localhost:7000",
    strings.NewReader(`{"code": 5,"message": "Anybody there?"}`))
  req.Header.Set("Content-Type", "application/json")
  client := &http.Client{}
  resp, err := client.Do(req)
  if err != nil {
    panic(err)
  }
  defer resp.Body.Close()
  fmt.Printf("wrote request")
}
```

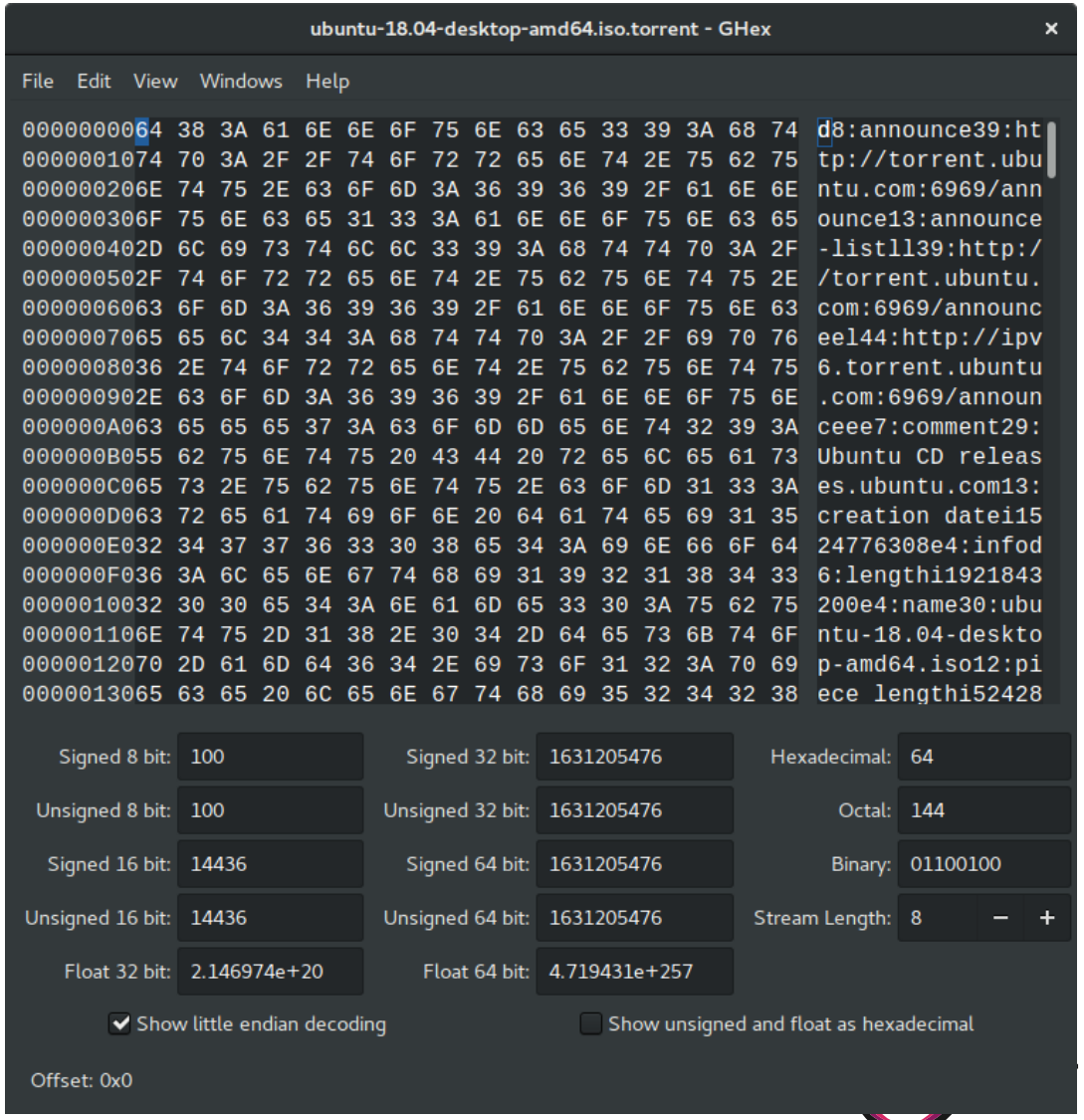
- Parsing overhead, JSON slower than binary protocol - benchmarks

```
[
    {
        "id": "bitcoin",
        "name": "Bitcoin",
        "symbol": "BTC",
        "rank": "1",
        "price_usd": "9324.08",
        "price_btc": "1.0",
        "24h_volume_usd": "9039300000.0",
        "market_cap_usd": "158560288125",
        "available_supply": "17005462.0",
        "total_supply": "17005462.0",
        "max_supply": "21000000.0",
        "percent_change_1h": "0.46",
        "percent_change_24h": "-0.27",
        "percent_change_7d": "4.5",
        "last_updated": "1525011874"
    }, ...
]
```


OST

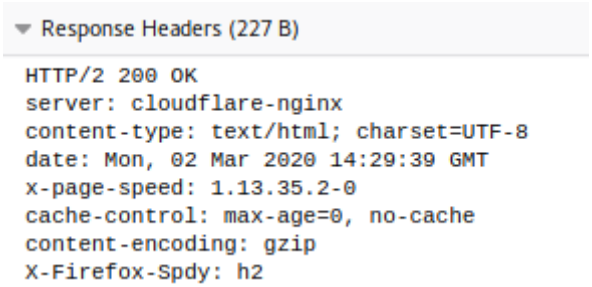
Protocols Bencoding and Others

- Benconding
  - Integers: i42e
  - Byte string: 4:test
  - List: l4:testi42ee
  - Map/dictionary: d4:test3:hsr3:tomi42ee
- Use: BitTorrent
- Cap'n Proto , FlatBuffers
  - Do not serialize, just copy, little-endian
- Apache Arrow
  - Do not serialize, copy, and optimally layout for memory access
- … and many others
- Benchmarks, benchmarks, benchmarks



Application Protocol: HTTP

- Response
  - Header



  - Status Code: 200
    - List: from 1xx (information response), 2xx (success) – 200 OK, 3xx (redirection), 4xx (client error), 404 Not Found, 403 Forbidden (access slides outside HSR), 5xx (server errors)
  - Content

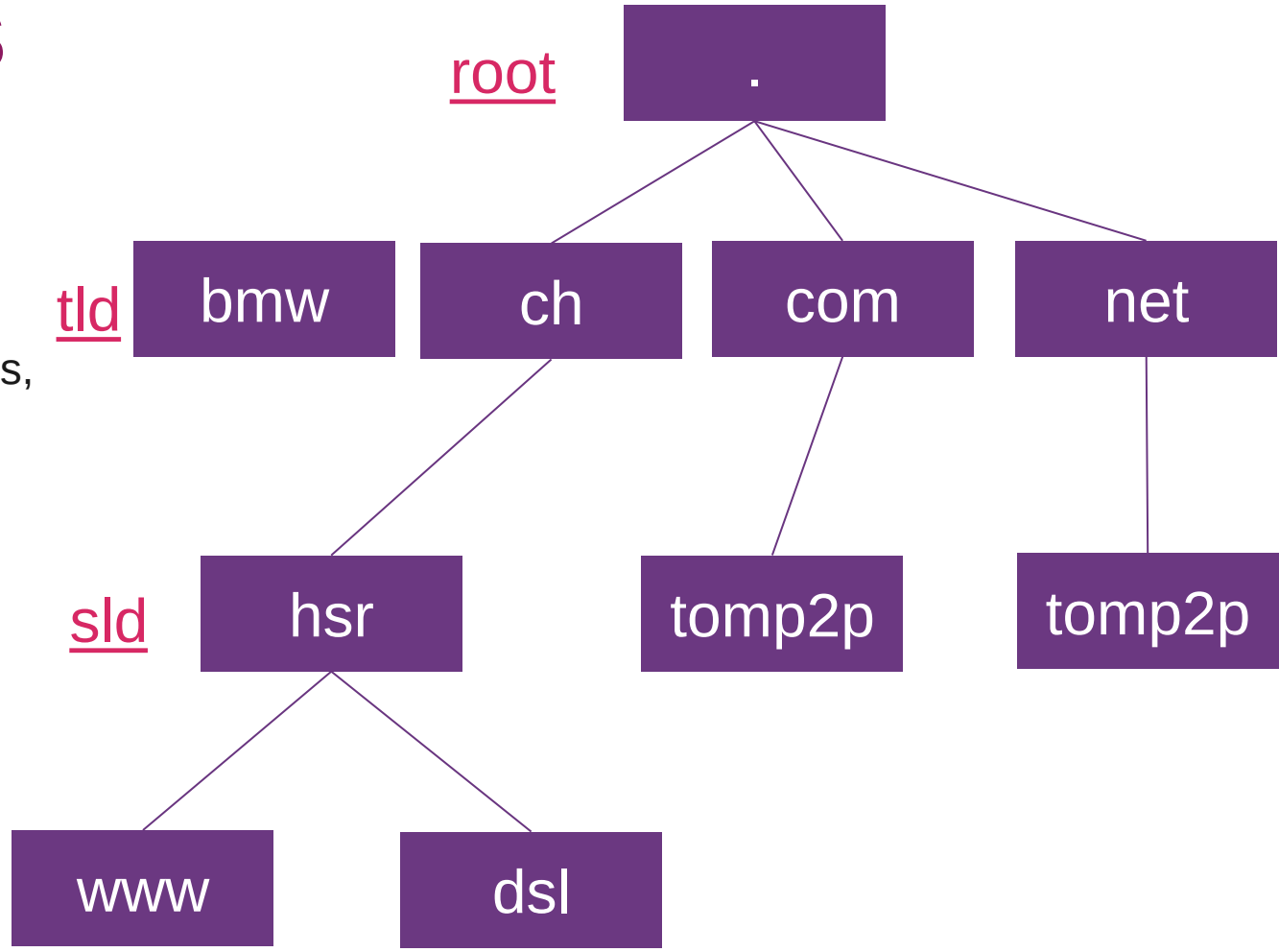
```
<!DOCTYPE html>
<html>
<head>
    <title>Distributed Systems and Ledgers Lab</title>
    <link rel="stylesheet" type="text/css" href="design/layout.css"/>
…
```

- HTTP is a stateless protocol
  - Server maintains no state
- 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 Hallo'; } | nc -l 8080 -q 1; done
- Every Webbrowser has dev tools to show request / responses
  - Firefox, Chrome: ctrl+shift+i / F12
  - Used regularly

OST

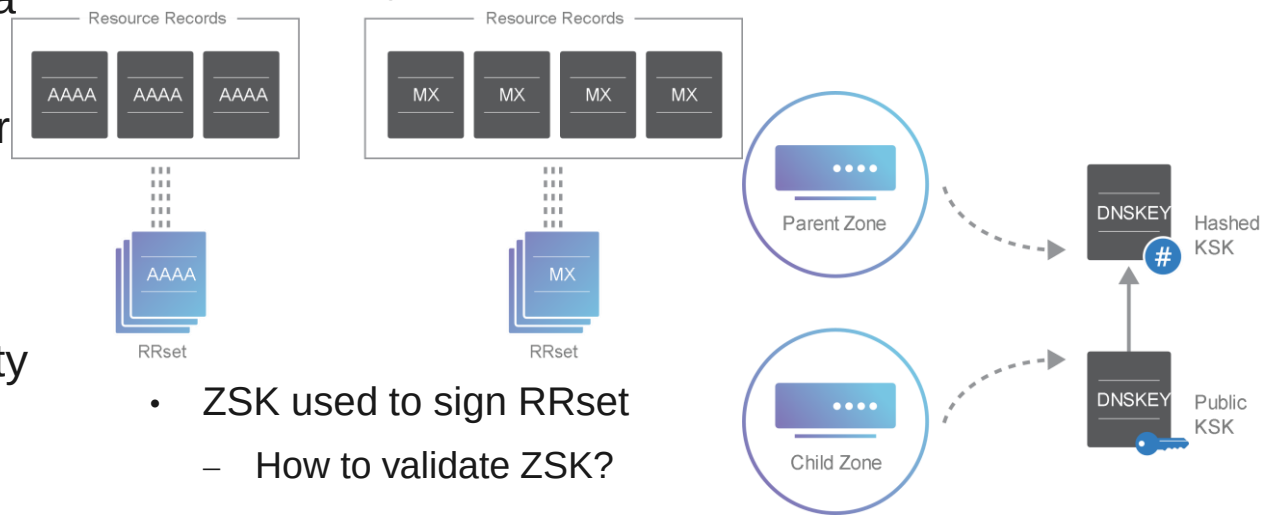
Application Protocol: DNS

- Translates human readable domain names to IP addresses “phonebook of the Internet”
  - Delegate authority over sub-domains to other name servers
- Lots of new TLD: .zuerich, .bmw, .americanexpress, .youtube, .ϑϑ
  - No special characters: ASCII (no UTF)
  - Punycode: bücher.tld → xn--bcher-kva.tld
- Hierarchical and decentralized naming system for computers
  - E.g., dsl.hsr.ch
  - Uses UDP, port 53
  - Designed in 1983: unencrypted, unsigned
- Before DNS: exchange of hosts.txt
  - Does not scale



Application Protocol: DNS

- To run your own DynDNS service: TSIG
  - Enables DNS queries to authenticate updates to a DNS database
  - Uses shared secret and cryptographic hashing for authentication
- DNSSEC (security extension)
  - Authenticated and data integrity, **not** confidentiality
  - Can be used to bootstrap other security systems
    - Certificates, SSH fingerprints, IPSec pub keys
  - KSK: key signing keys to sign other DNSKEY records
  - ZSK: zone signing keys to sign records
    - Example: dig DNSKEY tomp2p.net
- New record types: RRSIG, DNSKEY, DS, …
  - RRSIG, sign all resource sets



  - ZSK used to sign RRset
    - How to validate ZSK?
  - KSK used to sign ZSK pub key
    - With 2 keys, its easier to change ZSK
  - We now know that entries are signed. KSK signes ZSK, ZSK signs Rrset
    - dig any tomp2p.net
  - DS (delegation signer) record in the parent zone

https://www.cloudflare.com/dns/dnssec/how-dnssec-works/

The DNS war

DoH

- provides confidentiality of lookups in transit
- Uses standard HTTP/2, on the standard port (443)
  - Cannot distinguish between traffic/DNS
- Trivially deployed , DNS response are served like simple web pages
- Performance: TCP+TLS handshake → 2/3 RTT
  - But: Cloudflare is close to you
- Difficult upgrade path for clients: per-application installation
- Browsers can perform DNS queries using Javascript

DoT

- provides confidentiality of lookups in transit
- DNS over TLS, separate port (853)
  - Can be blocked
- Widely supported by serving software (Bind, PowerDNS, Unbound) and public resolvers (Cloudflare, Quad9, Google)
- Performance: TCP+TLS handshake → 2/3 RTT
  - But: ISP is close to you
- Easy upgrade path for clients: clients can test if the configured resolver supports DoT on port 853, fall back to DoU53 otherwise)

Lecture 4

3 Types of load balancing: Hardware, Cloud-based, Software load balancer

- Hardware load balancer
  - Application delivery controller (ADC)
    - Also a load balancer, remove load from web servers
  - HW-LB use proprietary software, which often uses specialized processors
    - Less generic, more performance
    - Some use open-source SW, e.g., Haproxy
  - E.g., Barracuda, F5, Cisco
  - Only if you control your datacenter

- Software load balancer
  - L2/L3: Seesaw
  - L4: LoadMaster, HAProxy (desc), ZEVENET, Neutrino, Balance (C), Nginx, Gobetween, Traefik
  - L7: Envoy (C++), LoadMaster, HAProxy (C), ZEVENET, Neutrino (Java/Scala), Nginx (C), Traefik (golang), Gobetween (golang), Eureka (Java) – services register at Eureka

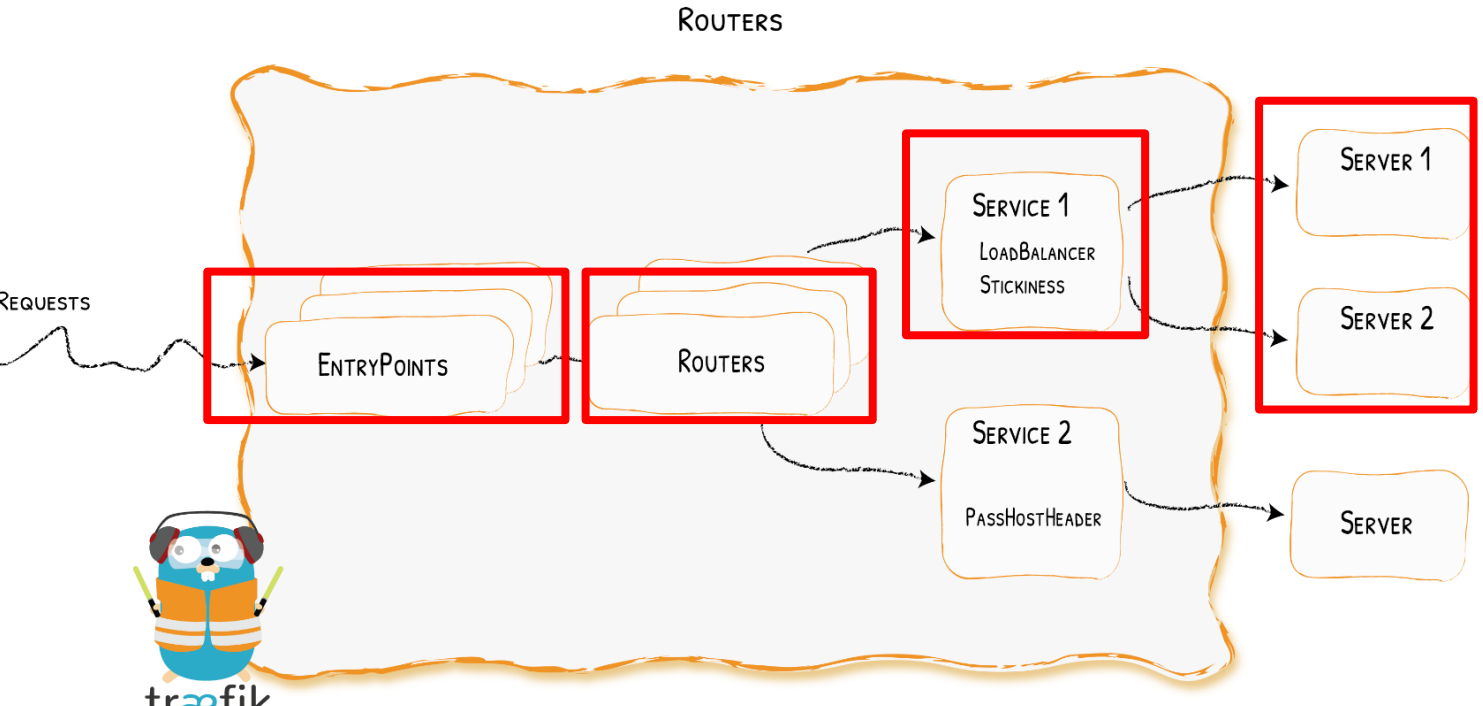
- SW vs. SW / SW vs. HW
  - strong opinions, funny opinions, other opinion, but: “We encourage users to benchmark Envoy in their own environments with a configuration similar to what they plan on using in production [source]”

OST

Traefik

- Run it: ./traefik
  - Now lets configure
- Redirect 8888 to access dashboard
  - http://127.0.0.1:8888/dashboard/

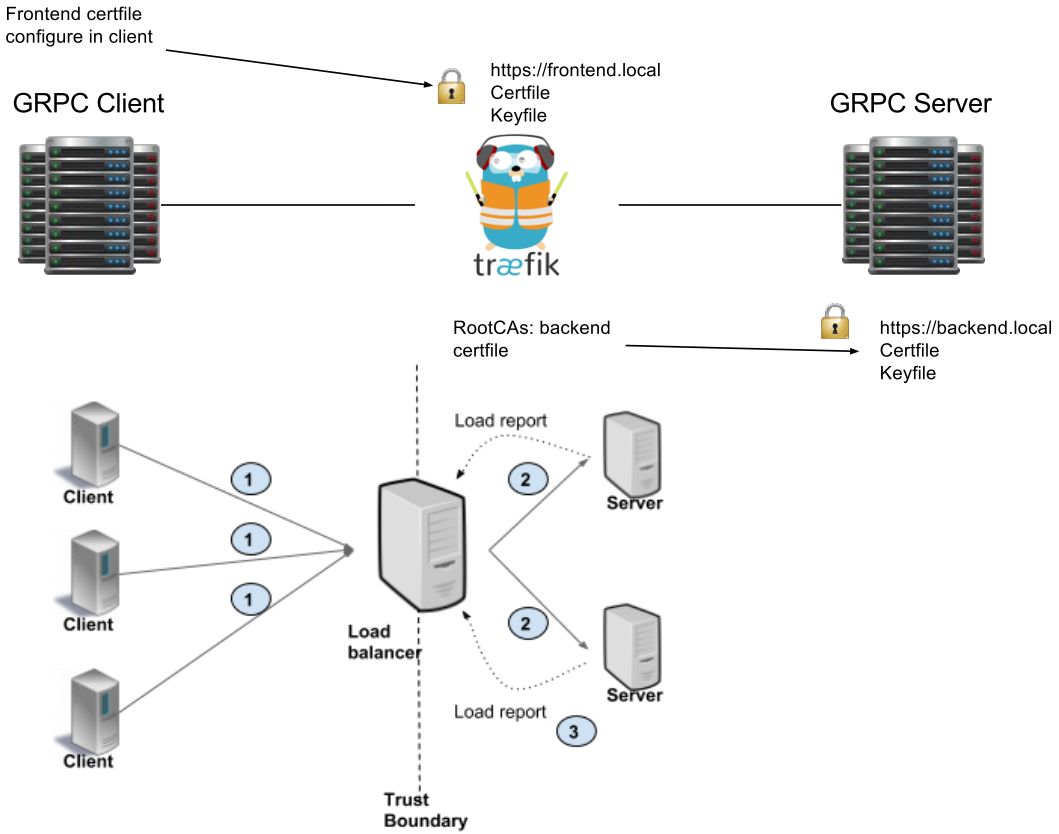
```
[entryPoints.web]
address = ":80“

[api]
dashboard = true

[providers.file]
filename = "dynamic_load.toml“

[log]
#filePath = "traefik.log“
level = "INFO“

[accessLog]
```




```
[http.routers.dashboard]
rule = "PathPrefix(`/api`) || PathPrefix(`/dashboard`)“
entrypoints = ["web"]
service = "api@internal“
middlewares = ["auth"]

[http.middlewares.auth.basicAuth]
users = ["test:$apr1$H6uskkkW$IgXLP6ewTrSuBkTrqE8wj/"]

[http.routers.coinservice]
rule = "PathPrefix(`/`)“
entrypoints = ["web"]
service = "coinservice“

[[http.services.coinservice.loadBalancer.servers]]
url = "http://127.0.0.1:8080"
[[http.services.coinservice.loadBalancer.servers]]
url = "http://127.0.0.1:8081"
```

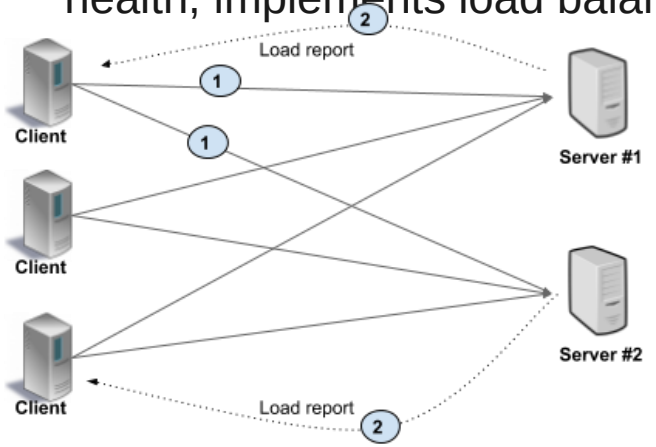
gRPC Load Balancing

- Proxy gRPC with Traefik



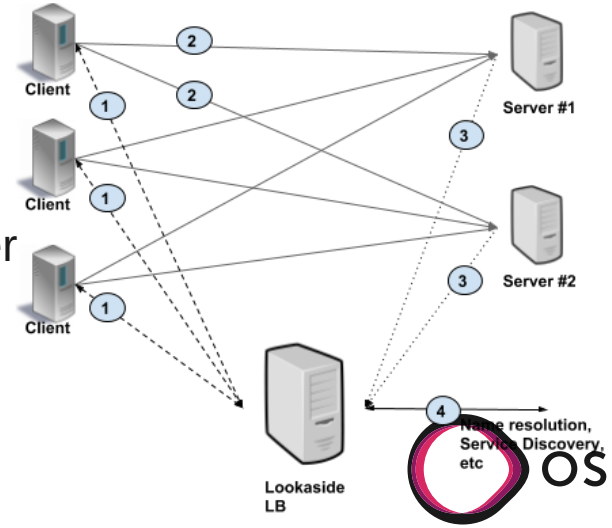
https://grpc.io/blog/grpc-load-balancing/

- P2P/ client side
  - Complex client, keeps track of server load and health, implements load balancing algo



- Look-aside
  - Mix of P2P/proxy
  - Ask proxy for server

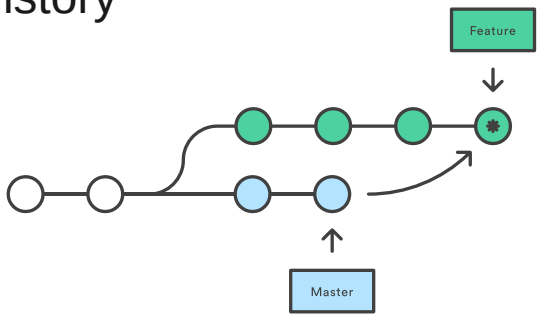
    → connect directly



Distributed Version Control

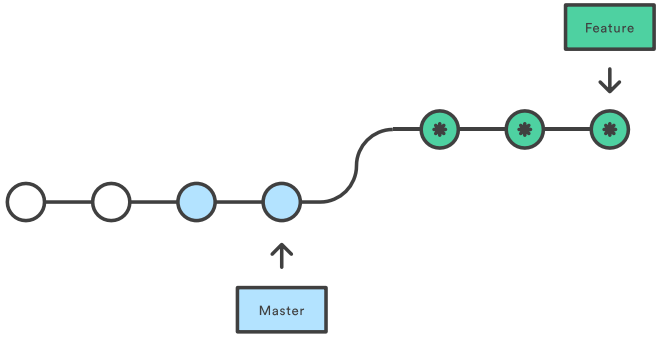
- Merge conflict
  - Edit file and fix it
  - git add *
  - git commit
- Rebase
  - Merge preserves history whereas rebase rewrites it
  - Rebasing is better to streamline a complex history
- Squash
  - git checkout master
  - git merge –squash feature
  - git commit
- Cherry picking
  - git cherry-pick cc5507b071

Merging feature branch into master



Merge Commit

Rebasing the feature branch onto master



Brand New Commit

OST