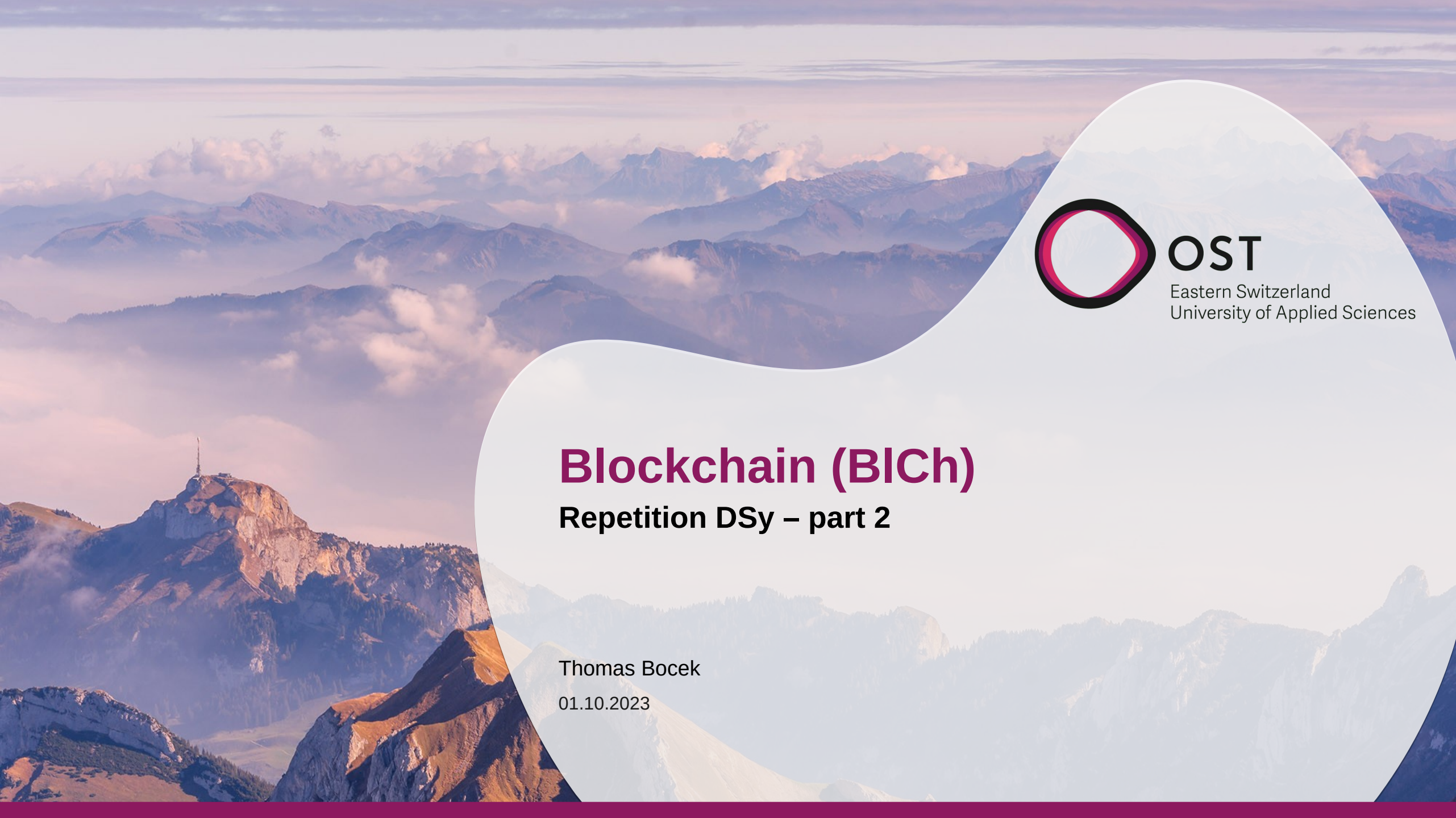


OST
Eastern Switzerland
University of Applied Sciences

Blockchain (BlCh)

Repetition DSy – part 2

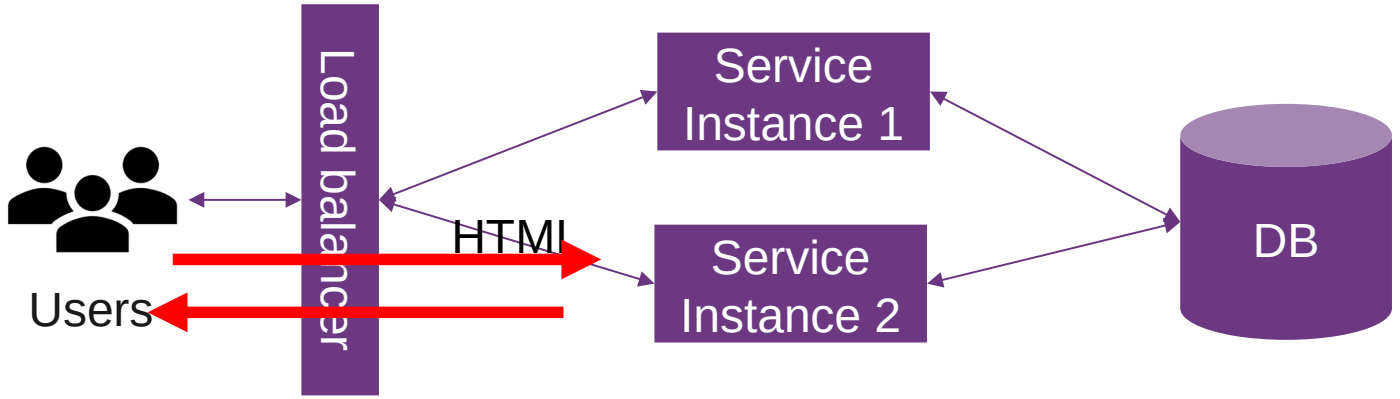
Thomas Bocek

01.10.2023

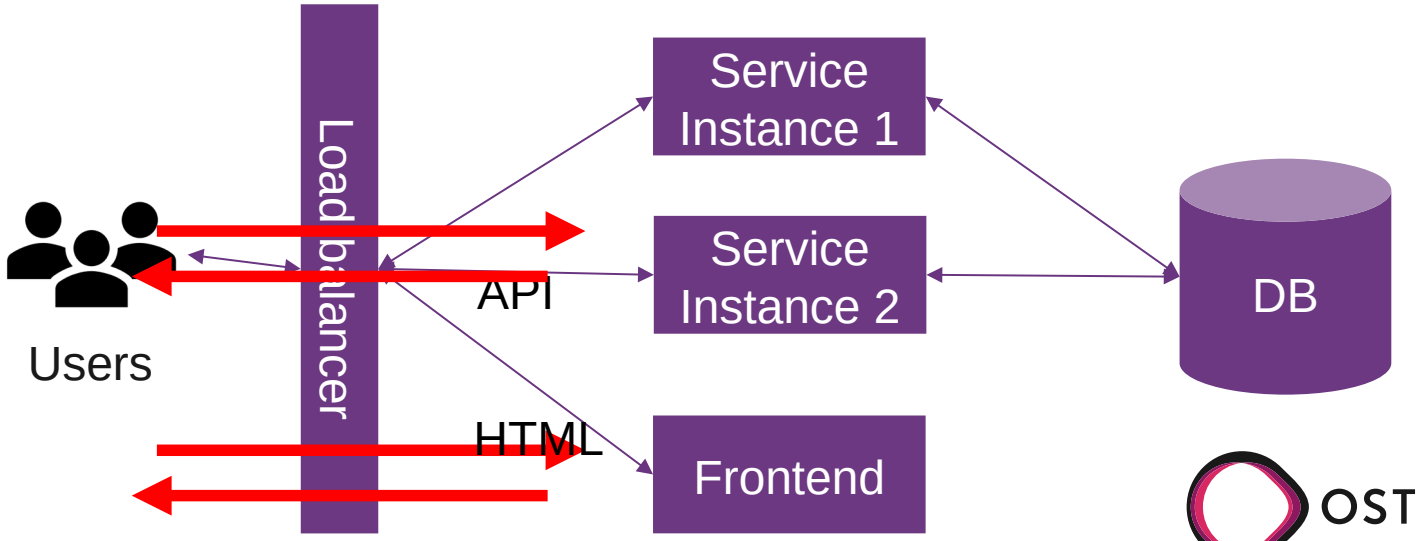
Lecture 5

Architecture

- Server side rendering (SSR)



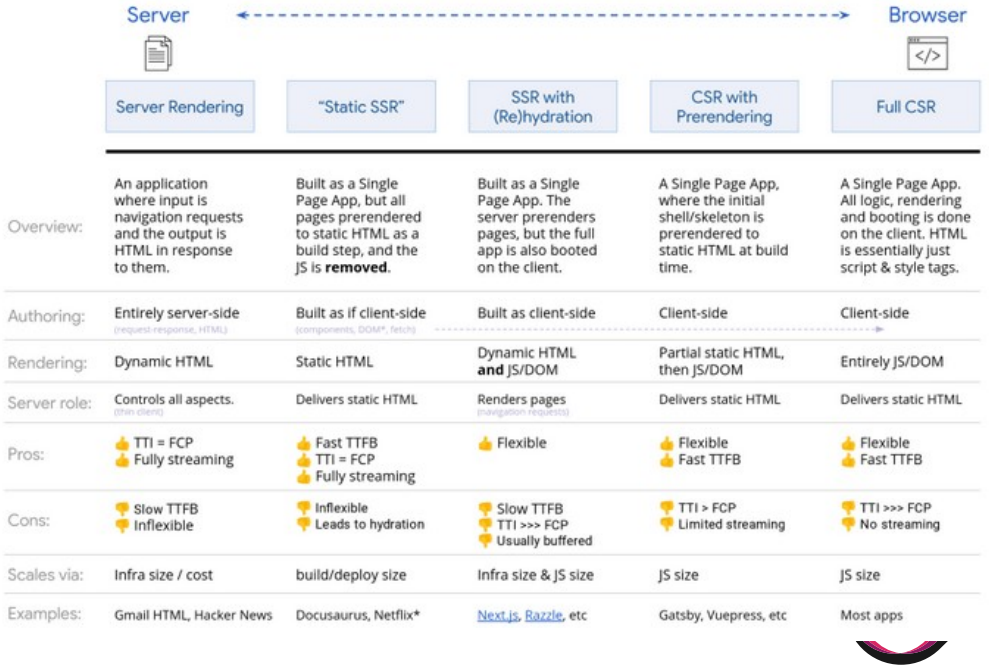
- Single page application (SPA), client side rending (CSR)



OST

Examples

- Static site rendering: dsl.i.ost.ch
  - Componets: nginx
  - Java daemon who reacts on file changes in a director. If markdown file changes → create HTML, copy it to nginx directory
- Server side rendering (e.g., handlebarsjs)
  - Simple example: ssr.go (no template)
  - Components: go-based server
- SPA
  - Components: node server, go server
- Hydration
  - Best of both worlds, but adds complexity, needs JavaScript in the backend
  - E.g., react: hydrate() instead of render() method – choices... source



Authentication

- Authentication
  - Single-factor authentication
    - E.g. password
  - Multi-factor authentication / 2FA
    - E.g. password and software token, SMS (15.03.2021)
- Password rules
  - Don’t use:
    - The name of a pet, child, family member, or significant other
    - Anniversary dates and birthdays
    - Birthplace
    - Name of a favorite holiday
    - Something related to a favorite sports team
    - The word "password“
  - Don’t’ reuse passwords, use password managers
- Don’t enter passwords on unencrypted sites
- Password length:
  password cracking with 5000$ in 2018 with hashcat
  - Hashtype: WPA/WPA2: 1190.5 kH/s

Pw length	Combinations	Time
6	11m	9s
7	656m	9m
8	38b	8h
9	$7 * 10^{15}$	186y
10	$4 * 10^{17}$	11ky
11	$2 * 10^{19}$	665ky
12	$1 * 10^{21}$	38my

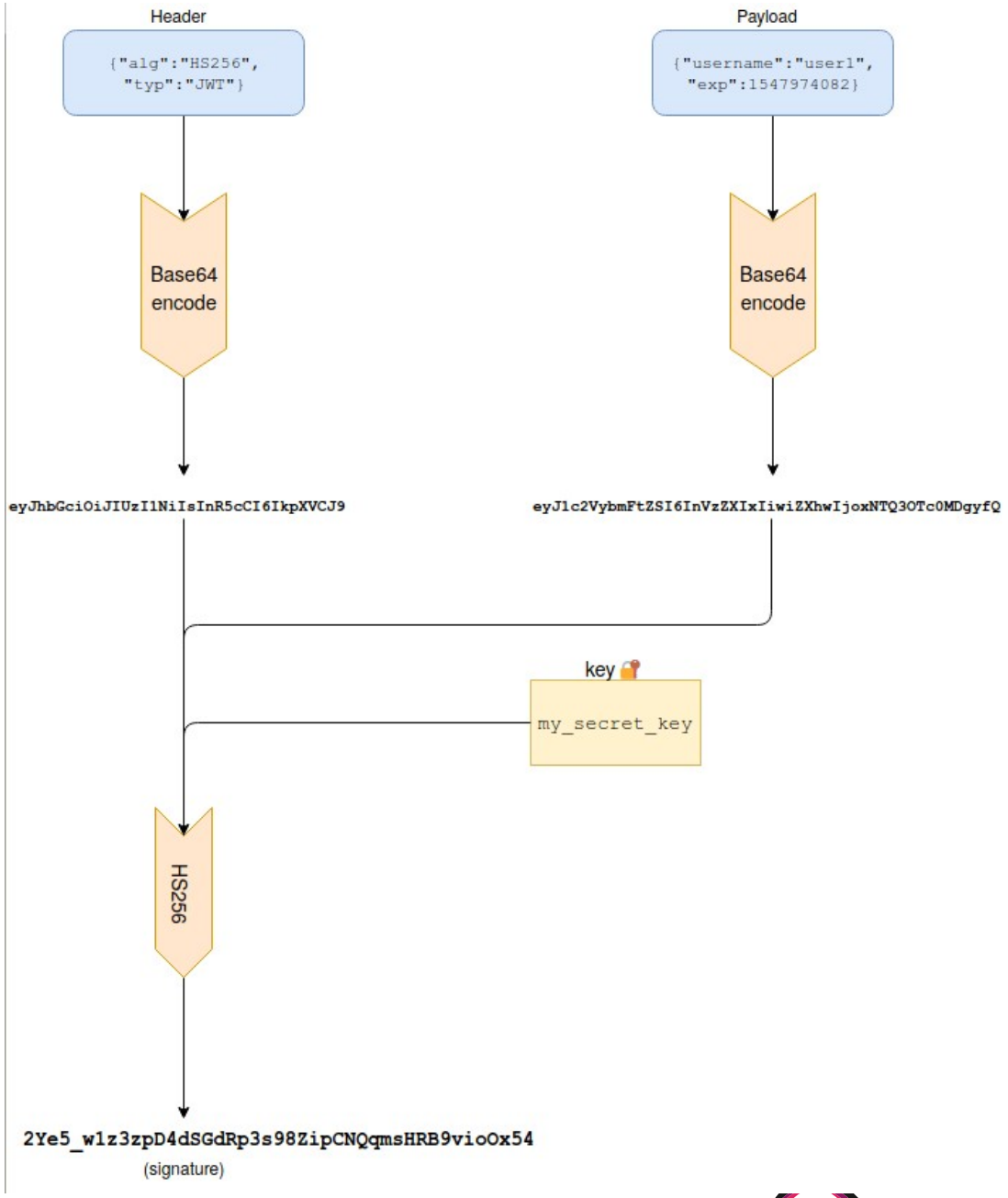
- Combinations depend on PW complexity

OST

Authentication

- JSON-based access tokens
  - Header: {"alg" : "HS256"}
  - Payload: {“sub" : "tom", "role" : "admin", “exp" : 1422779638}
- Signature (simple): keyed-hash message
  - ~hash(base64(header)+base64(payload) + secret token)
- Client can store user_token in
  - localStorage.setItem(“token", userToken);
- Example in golang with JWT
  - Tutorial: here and here

- OAuth - protocol for authorization 3rd party integration
  - Grant access on other websites without giving them the passwords



source

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

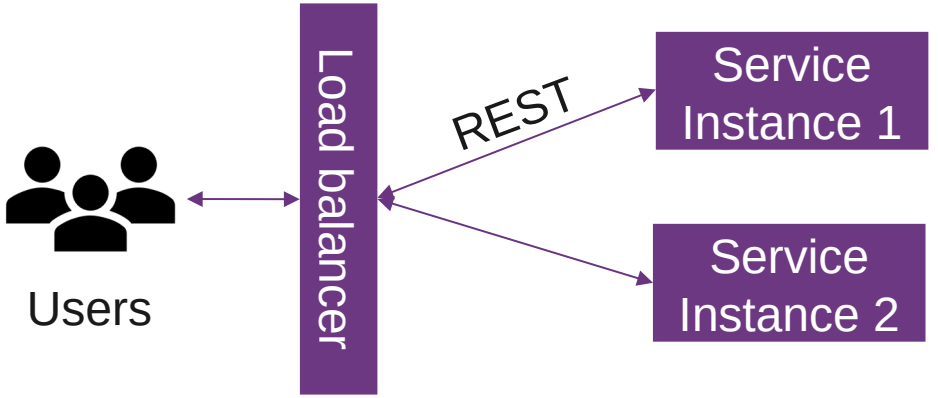
Lecture 6

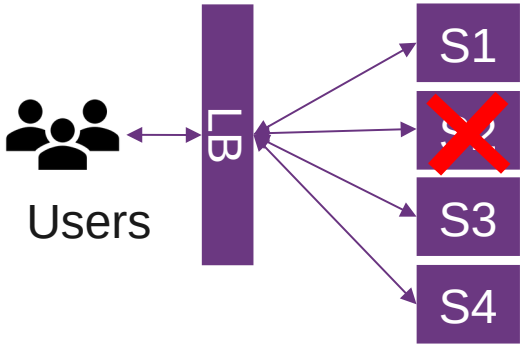
Load Balancing

- 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





OST

Caddy

- Configuration: dynamic
  - Static: Caddyfile
- One-liners:
  - Quick, local file server: caddy file-server
  - Reverse proxy: caddy reverse-proxy --from example.com --to localhost:9000

```
:7070
reverse_proxy 127.0.0.1:8081 127.0.0.1:8080 {
  unhealthy_status 5xx
  fail_duration 5s
}
```

- Open Source, software-based load balancer: https://github.com/caddyserver/caddy
  - “Caddy 2 is a powerful, enterprise-ready, open source web server with automatic HTTPS written in Go”
  - L7 load balancer
  - Reverse proxy
  - Static file server
  - HTTP/1.1, HTTP/2, and experimental HTTP/3
  - Caddy on docker hub

Dockerfile

- Example: caddy as LB, go as Service
  - docker-compose up --scale services=5

```
#docker-compose.yml
version: '3'
services:
  services:
    build: .
    ports:
      - "8080-8085:8080"
  lb:
    image: caddy
    ports:
      - "7070:7070"
    volumes:
      - ./Caddyfile:/etc/caddy/Caddyfile
```

```
#Caddyfile
:7070
reverse_proxy * {
  to http://dsy-services-1:8080
  to http://dsy-services-2:8080
  to http://dsy-services-3:8080
  to http://dsy-services-4:8080
  to http://dsy-services-5:8080

  lb_policy round_robin
  lb_try_duration 1s
  lb_try_interval 100ms
  fail_duration 10s
  unhealthy_latency 1s
}
```


OST

CORS

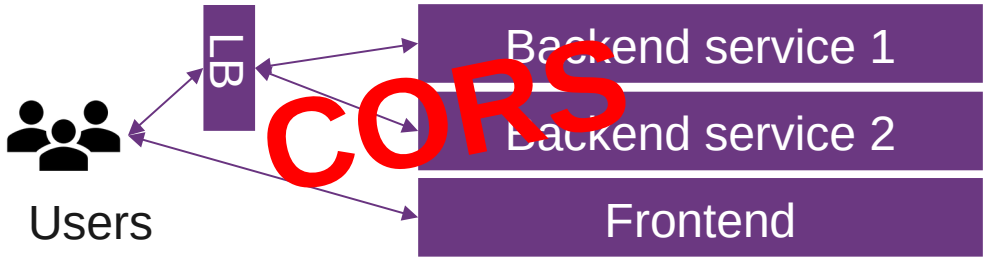
- CORS = Cross-Origin Resource Sharing
  - For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts (among others)
  - Mechanism to instruct browsers that runs a resource from origin A to run resources from origin B
- Solution
  - Use reverse proxy with builtin webserver, e.g., nginx, or user reverse proxy with external webserver.

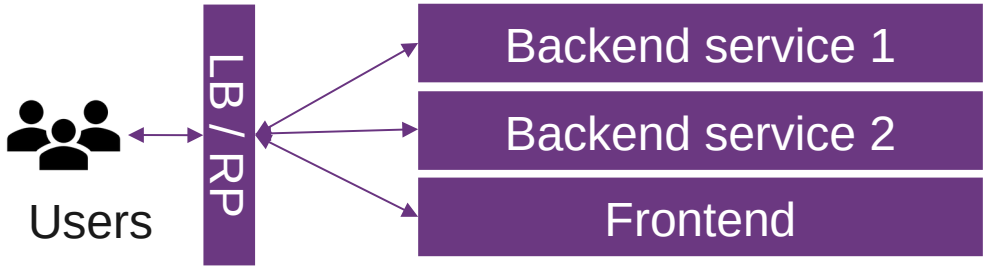
→ The client only sees the same origin for the API and the frontend assets

  - Access-Control-Allow-Origin: https://foo.example

→ For dev: Access-Control-Allow-Origin: *

  - w.Header().Set("Access-Control-Allow-Origin", "*")
- Reverse proxy

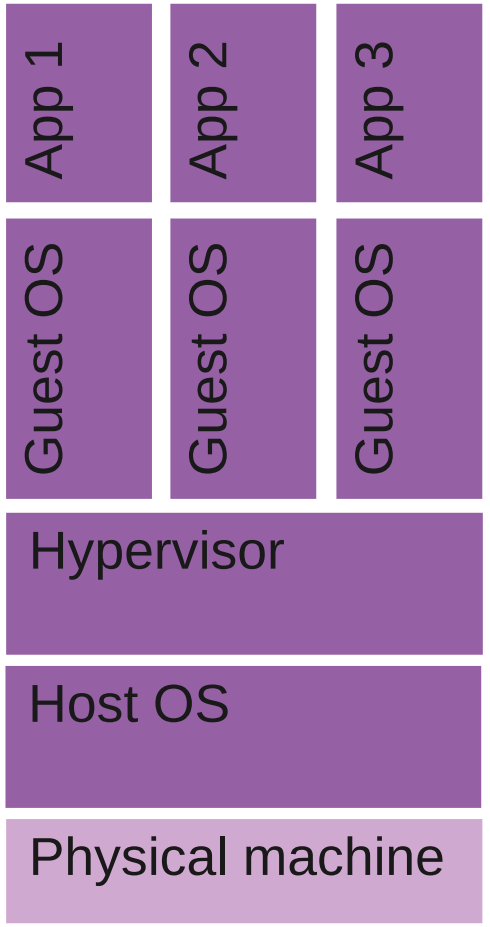




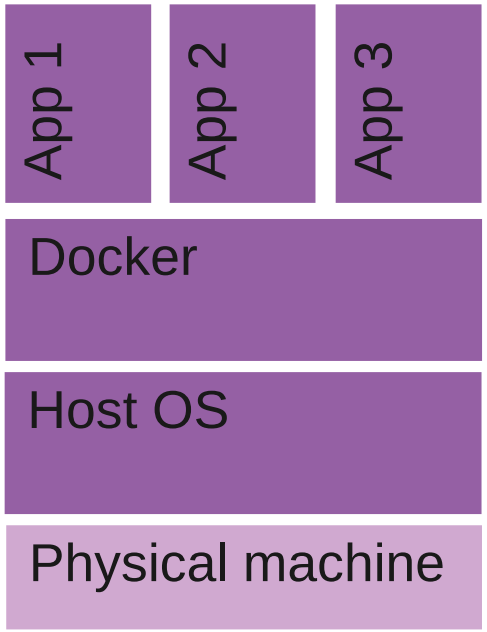
OST

Lecture 7

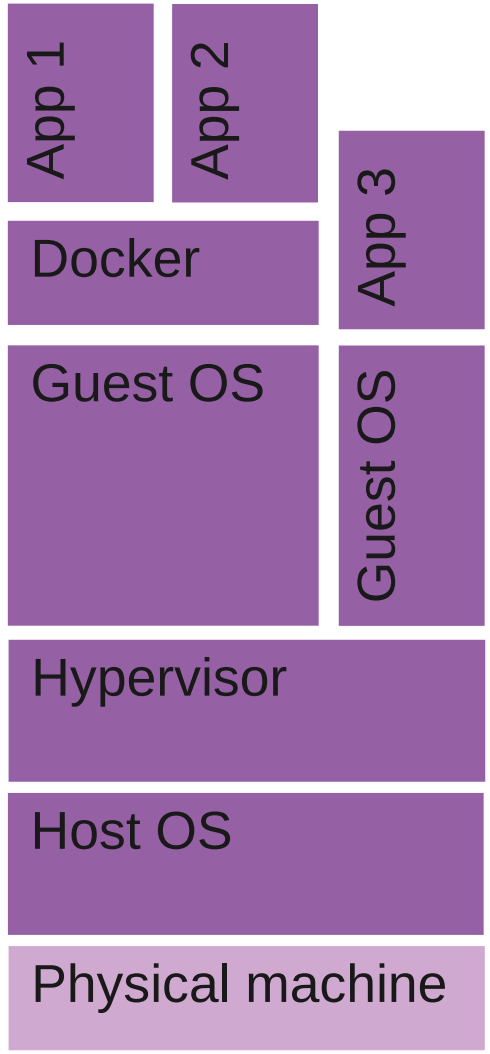
Introduction



- Virtual machines



- Container



- Both

OST

OverlayFS

- Example
  - The lower directory can be read-only or could be an overlay itself
  - The upper directory is normally writable
  - The workdir is used to prepare files as they are switched between the layers.

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o \
lowerdir=/tmp/lower,\
upperdir=/tmp/upper,\
workdir=/tmp/workdir \
none /tmp/overlay
```

- Read only
- How to remove data in read-only lowerdir
  - Mark as deleted in upperdir

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o
lowerdir=/tmp/lower1:/tmp/lower2 /tmp/overlay
```

```
cd /tmp
mkdir lower upper workdir overlay

sudo mount -t overlay -o \
lowerdir=/tmp/lower1:/tmp/lower2,\
upperdir=/tmp/upper,\
workdir=/tmp/workdir \
none /tmp/overlay
```


OST

Cgroups

- control groups: limits, isolates, prioritization of CPU, memory, disk I/O, network

```
ls /sys/fs/cgroup

sudo apt install cgroup-tools / yay -S libcgroup

cgcreate -g cpu:red
cgcreate -g cpu:blue

echo -n "20" > /sys/fs/cgroup/blue/cpu.weight
echo -n "80" > /sys/fs/cgroup/red/cpu.weight

cgexec -g cpu:blue bash
cgexec -g cpu:red bash

sha256sum /dev/urandom #does not work?
taskset -c 0 sha256sum /dev/urandom
```

- Install tools
- Create two groups
  - Assign 20% of CPU and 80% of CPU
- Execute bash → test CPU
- Resource control with docker

```
docker run \
--name=low_prio \
--cpuset-cpus=0 \
--cpu-shares=20 \
alpine sha256sum /dev/urandom

docker run \
--name=high_prio \
--cpuset-cpus=0 \
--cpu-shares=80 \
alpine sha256sum /dev/urandom
```


OST

Separate Networks

- Linux Network Namespaces
  - provide isolation of the system resources associated with networking [source]

```
ip netns add testnet
ip netns list
```

  - Create virtual ethernet connection

```
ip link add veth0 type veth peer name veth1 netns testnet
ip link list #?
ip netns exec testnet <cmd>
```

  - Configure network

```
ip addr add 10.1.1.1/24 dev veth0
ip netns exec testnet ip addr add 10.1.1.2/24 dev veth1
ip netns exec testnet ip link set dev veth1 up
```

- Run server

```
ip netns exec blue nc -l 8000
```

- Server can be contacted
- How to connect to outside?
  - E.g. layer 3

```
iptables -t nat -A POSTROUTING -s 10.1.1.0/24 -o enp9s0 -j MASQUERADE
iptables -A FORWARD -j ACCEPT #open up wide…
```

Lecture 8

Back in the old days…

- OTS: apt-get / yum / pacman install package, e.g., Apache – configure – run
- Custom SW: Java: war, provide custom /etc/init.d script with binary or script
- Problem:
  - It runs on my machine, who installs Java in the right version?
  - What happens on crashes?
  - Scaling?
  - HW defect?
  - Misconfiguration - access to complete PC?
- VMs / Containers help a lot
  - No access to complete PC, can scale, move to another machine, pre-install the right Java version
- So, how to deploy your app?
  - Ansible (Progress Chef, Puppet) - and more
    - Playbooks with ssh host list – your host needs to 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
      - Which to use? [link]
  - Kubernetes
    - Widespread

OST

Docker Swarm

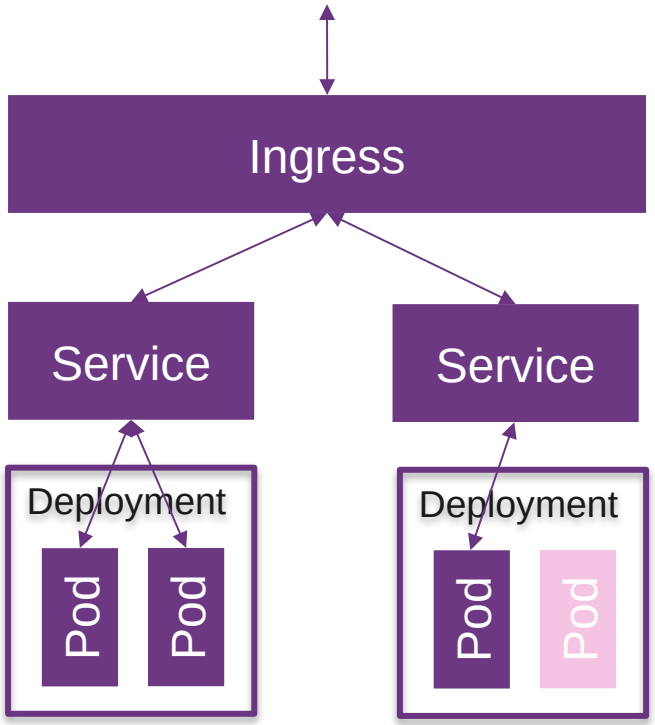
- Create service
  - docker service create --name registry --publish 5000:5000 registry:2
  - Where to find the docker image
- Check service
  - docker service ls
- Many options in docker-compose
  - docker stack deploy --compose-file docker-compose.yml

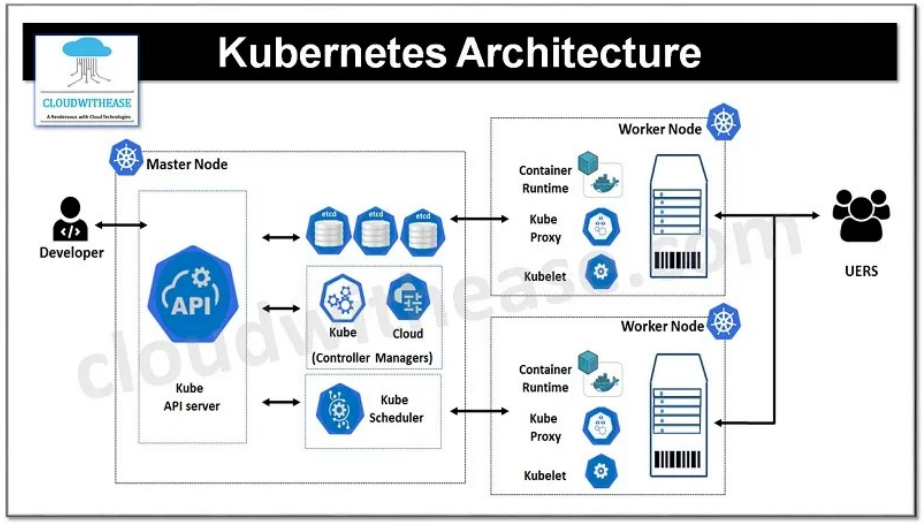
```
worker:
  image: gaiadocker/example-voting-app-worker:latest
  networks:
    voteapp:
      aliases:
        - workers
  depends_on:
    - db
    - redis
  # service deployment
  deploy:
    mode: replicated
    replicas: 2
    labels: [APP=VOTING]
    # service resource management
    resources:
      # Hard limit - Docker does not allow to allocate more
      limits:
        cpus: '0.25'
        memory: 512M
      # Soft limit - Docker makes best effort to return to it
      reservations:
        cpus: '0.25'
        memory: 256M
    # service restart policy
    restart_policy:
      condition: on-failure
      delay: 5s
      max_attempts: 3
      window: 120s
    # service update configuration
    update_config:
      parallelism: 1
      delay: 10s
      failure_action: continue
      monitor: 60s
      max_failure_ratio: 0.3
    # placement constraint - in this case on 'worker' nodes only
    placement:
      constraints: [node.role == worker]
```


OST

Kubernetes

- Getting Started with Kubernetes: Minikube, k3s
  - Minikube: Run a single-node Kubernetes cluster locally
  - kubectl: Command-line tool for managing a Kubernetes cluster
  - Kubernetes Dashboard: Web-based user interface for managing a cluster
- Deploy any containerized application
  - Use health endpoints
    - Liveness/Readiness
- Official documentation: https://kubernetes.io/docs
- Kubernetes tutorials: https://kubernetes.io/training
- Youtube course





Source: https://cloudwithease.com/what-is-kubernetes/

OST

Lecture 9

What is Mastodon?

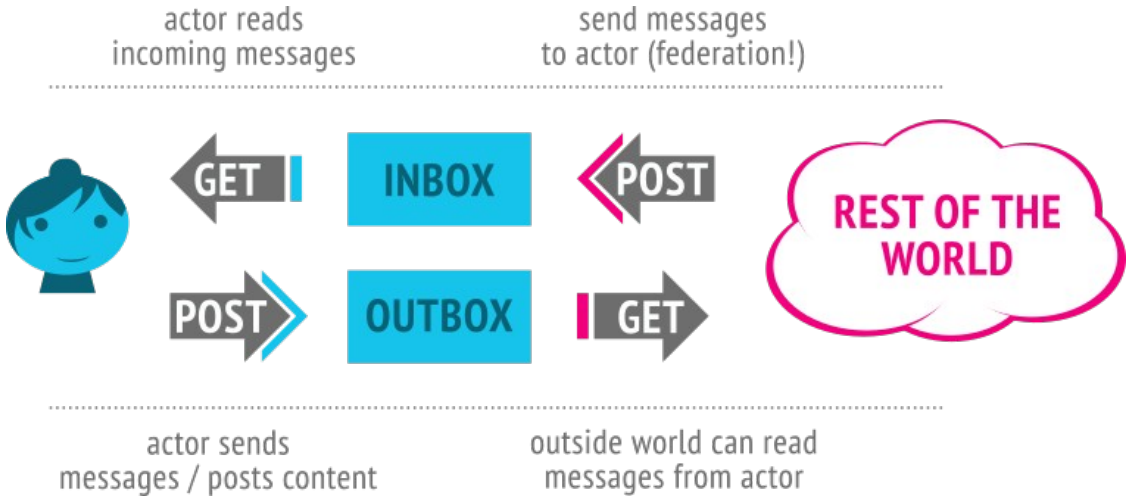
- “A mastodon (mastós 'breast' + odoús 'tooth') is any proboscidean belonging to the extinct genus Mammut” [link]
- A decentralized, open-source social network
  - “Mastodon is free and open-source software for running self-hosted social networking services” [link]
- Launched in 2016 by Eugen Rochko, now with Mastodan gGmbH [link]
  - Alternative to traditional social media platforms, offering greater privacy, user control, and an ad-free experience.

OST

Federation

- ActivityPub – decentralized communication protocol [link]
  - Open, decentralized protocol developed by the World Wide Web Consortium (W3C)
  - Enables communication and interaction between various social networks and applications
  - Uses URIs (Uniform Resource Identifiers) for uniquely identifying objects and users
  - Sends, receives, and processes activities such as Toots, Likes, and Follower relationships

- Inbox/Outbox

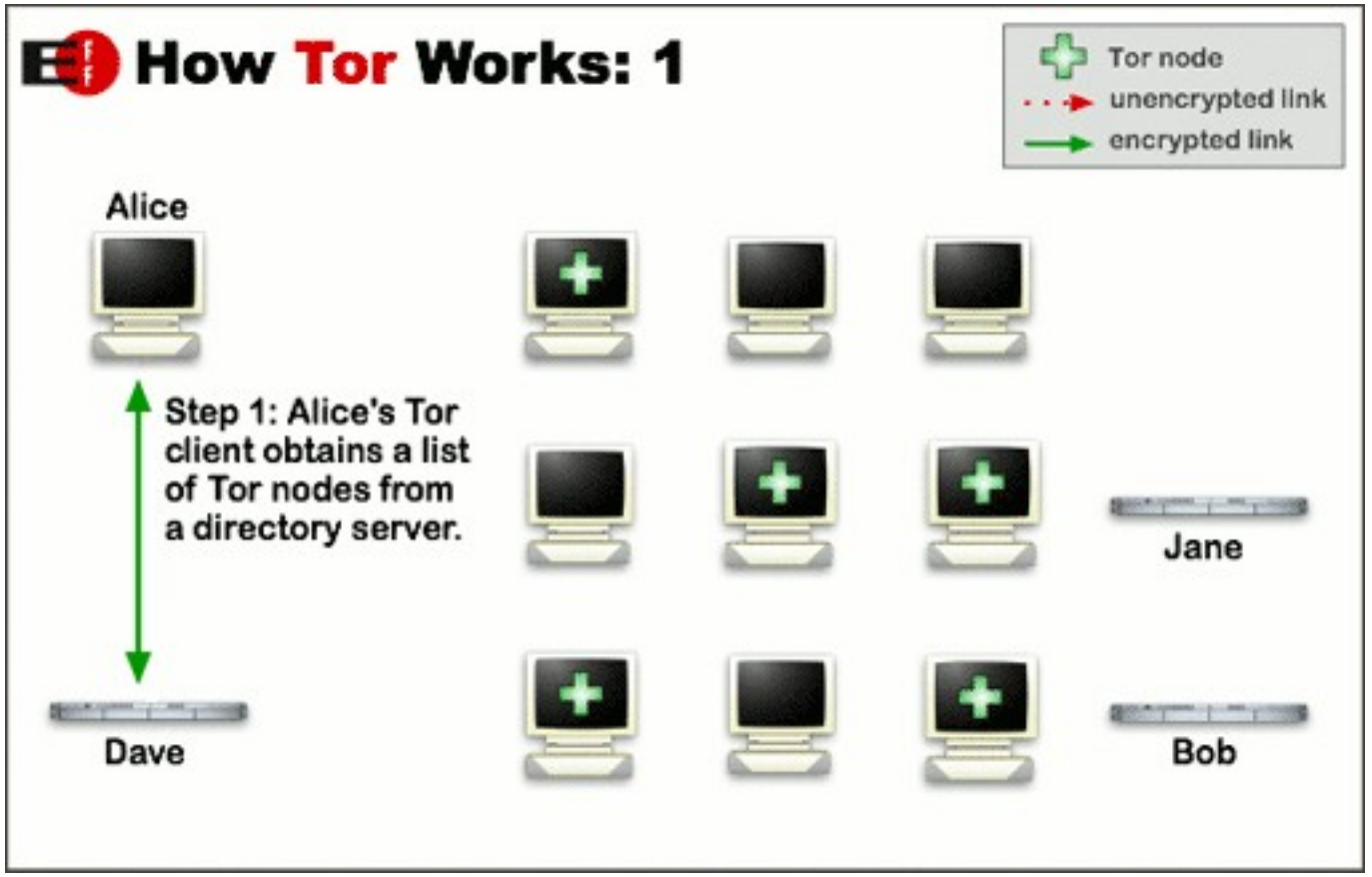


source

OST

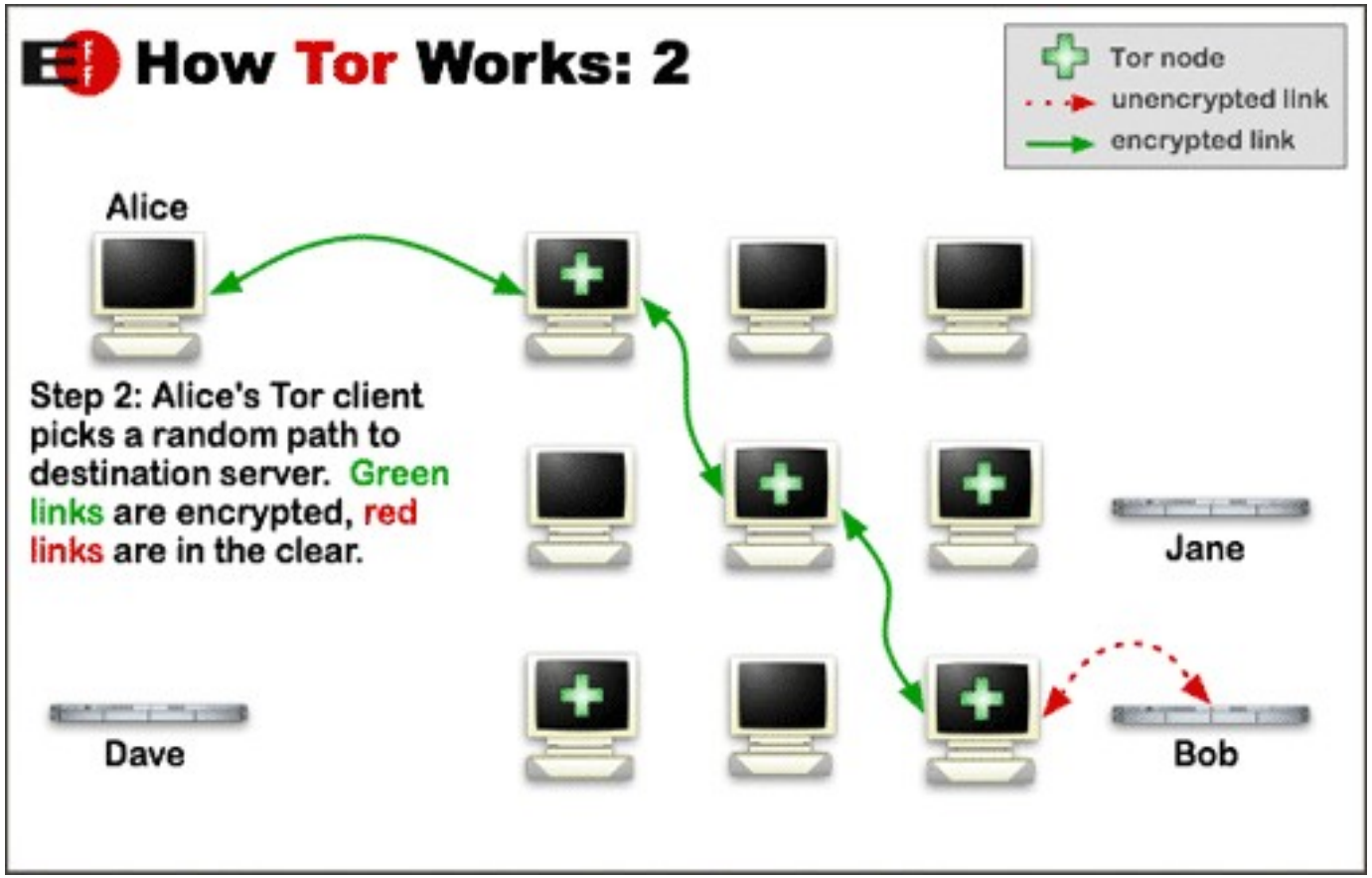
Tor

- How it works



Tor

- Alice to Bob



Tor

- Alice to Jane