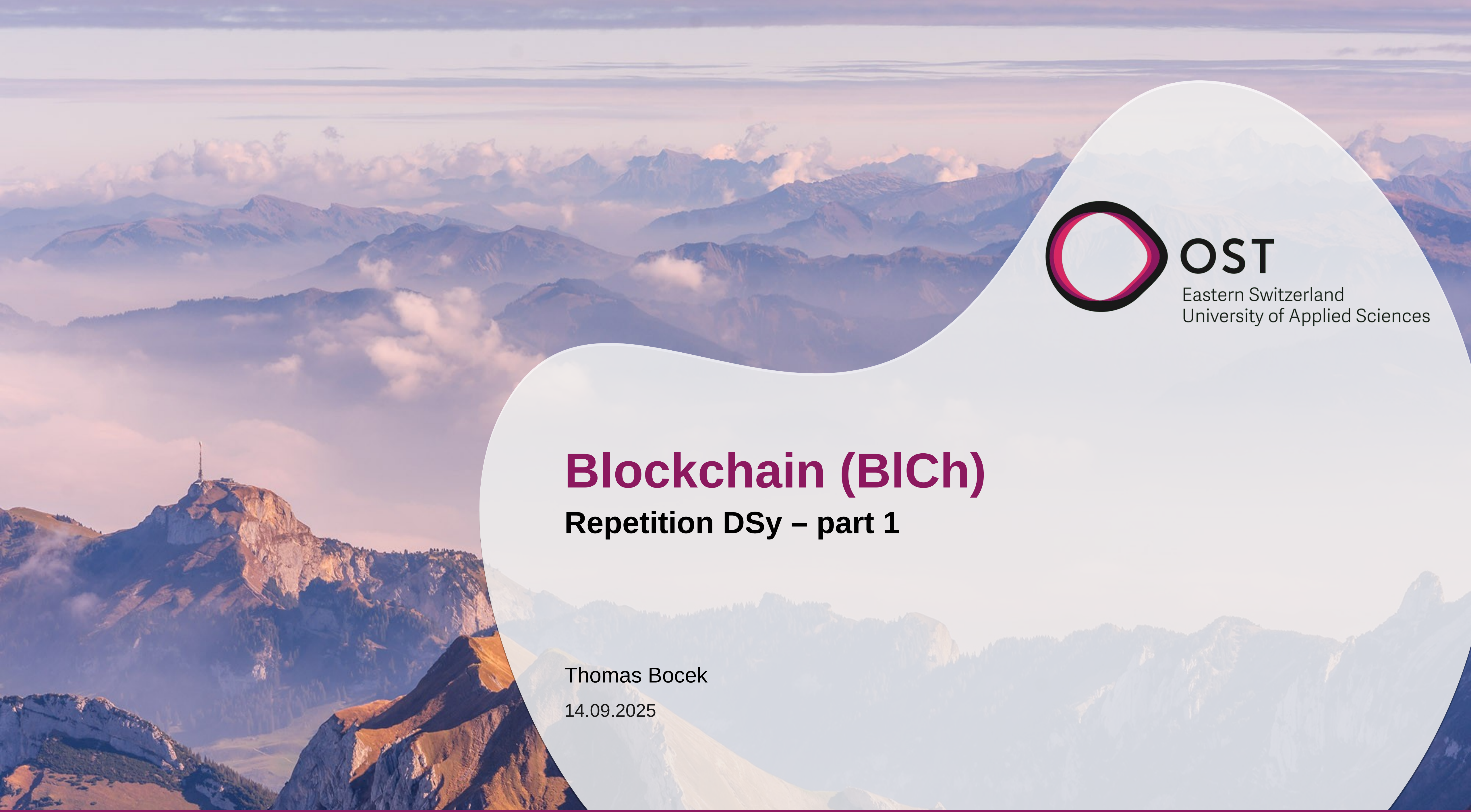


OST
Eastern Switzerland
University of Applied Sciences

Blockchain (BlCh)

Repetition DSy – part 1

Thomas Bocek

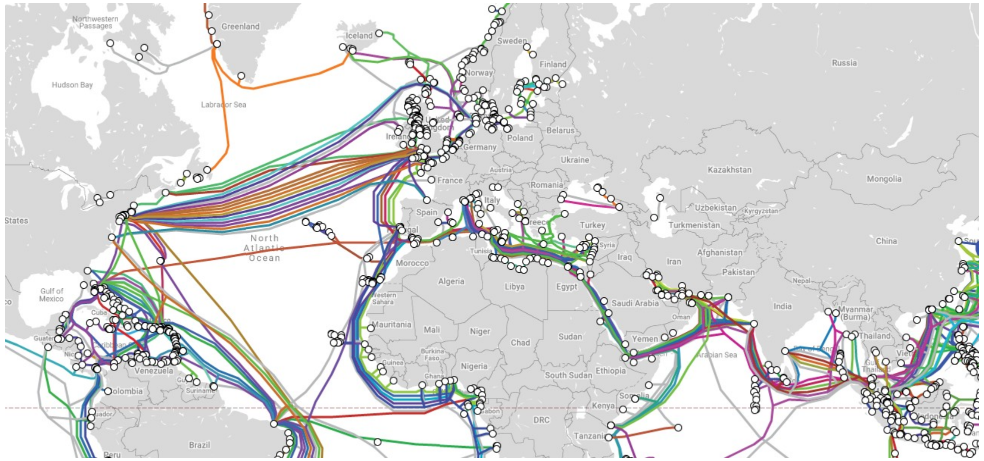
14.09.2025

Lecture 1 + 2

Introducion / Motivation

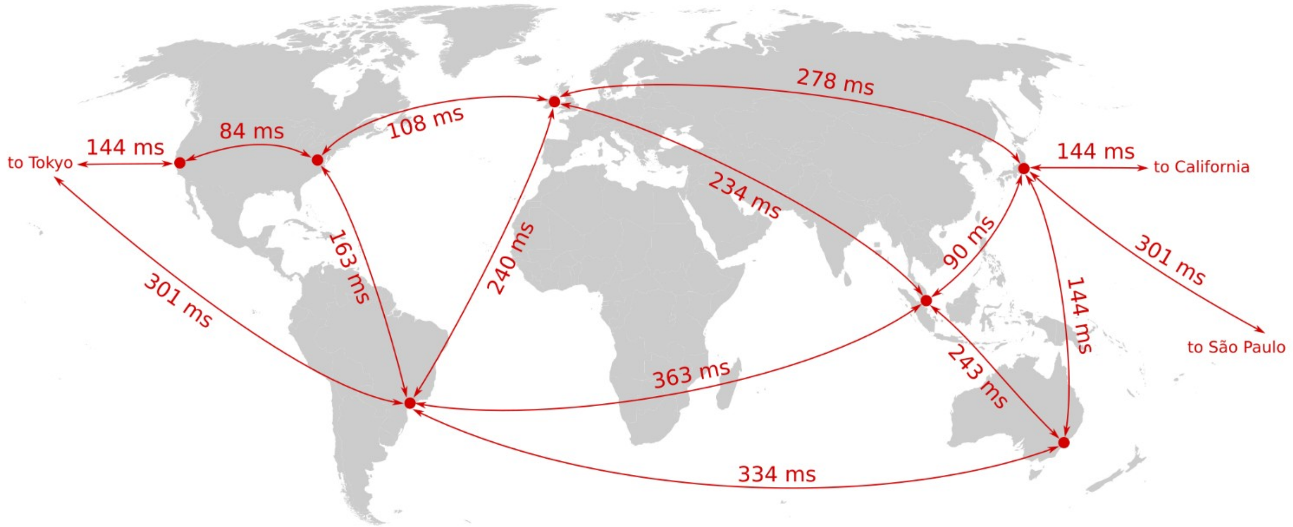
Distributed Systems Motivation

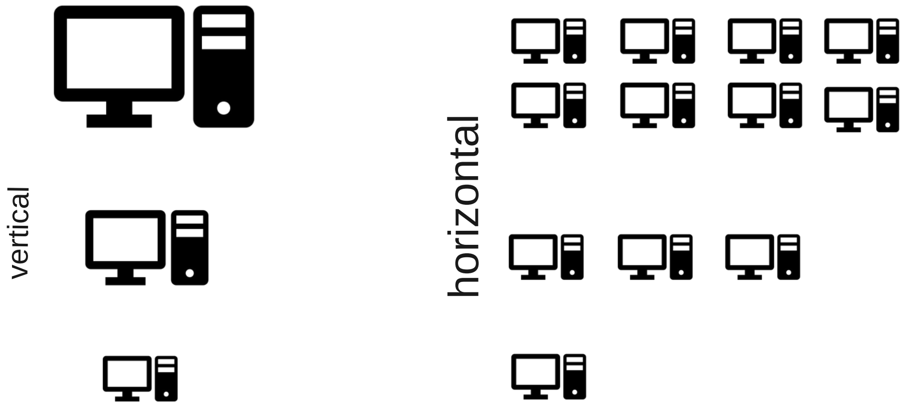
- Why Distributed Systems
  - Scaling
  - Location
  - Fault-tolerance (bitflips, outages)



Submarine Cable Map

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





Lecture 3

Monorepos / Polyrepos, Containers and VMs

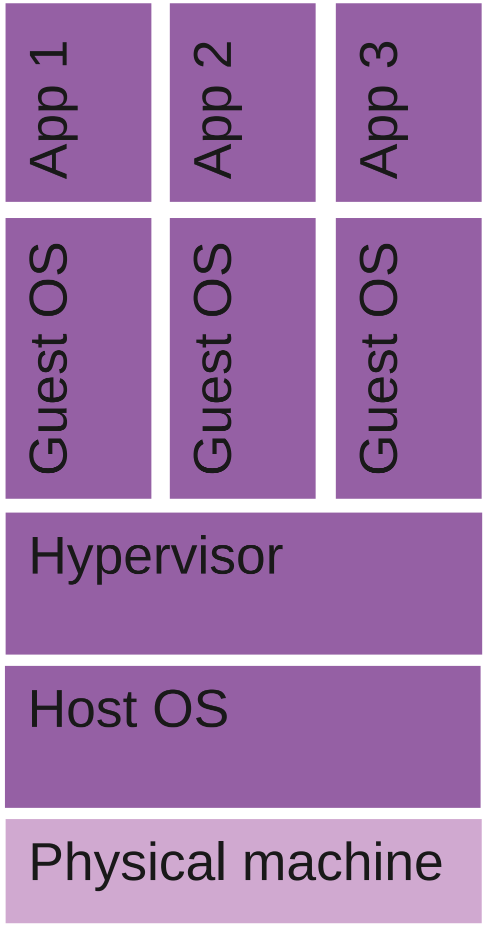
Pro/Cons - Opinion

- **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

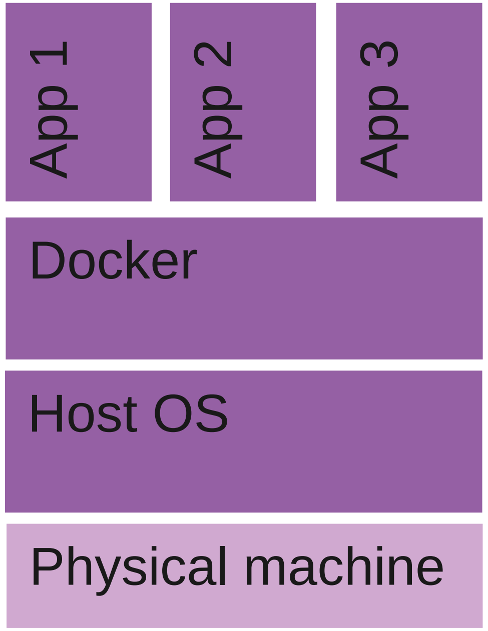
- **Polyrepo**
  - Loose coupling of projects
    - If you want to generate openapi.yml, you need access from the backend repository to the frontend (e.g., curl+token)
  - Fine grained access control
  - Encourages code sharing across organizations
  - Scaling: many projects, special coordination

Key Differences

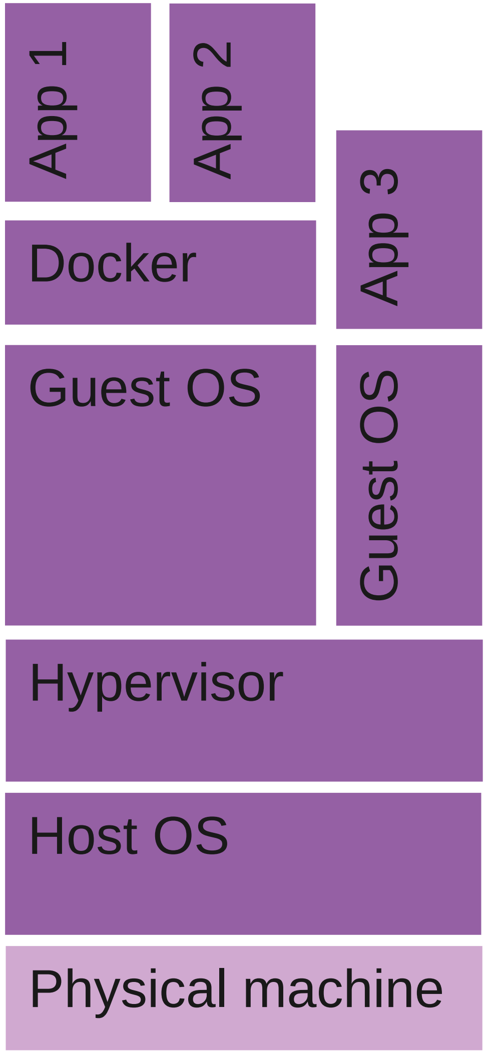
Introduction



- Virtual machines



- Container



- Both

Lecture 4

Docker, Debugging Containers

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
- Peformance: mulit-stage, .dockerignore

Lecture 5

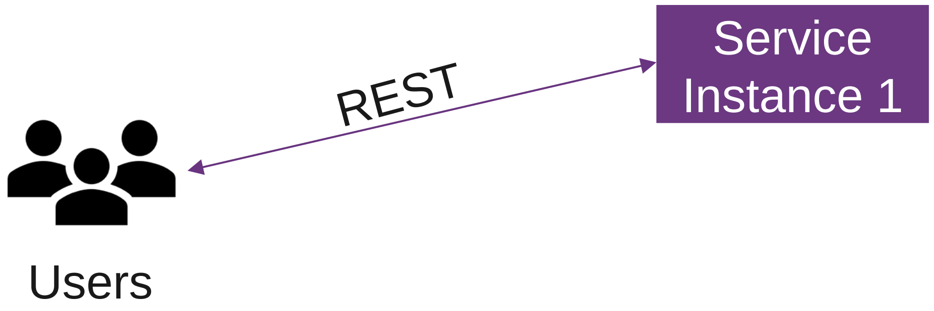
Load Balancing

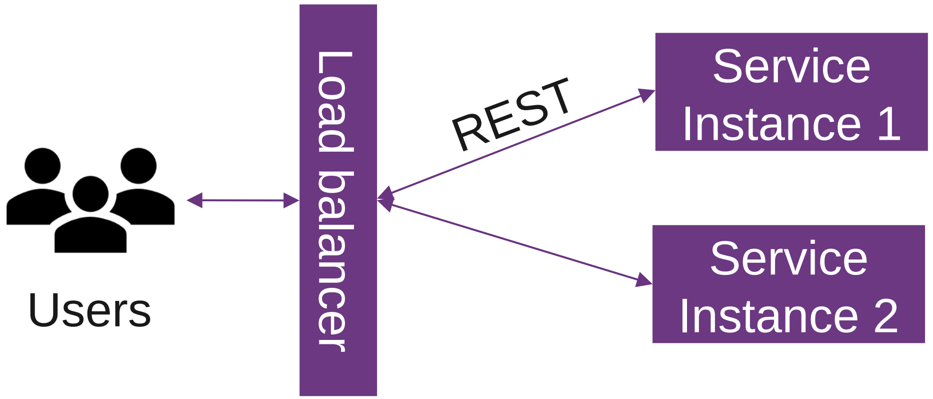
Load Balancing

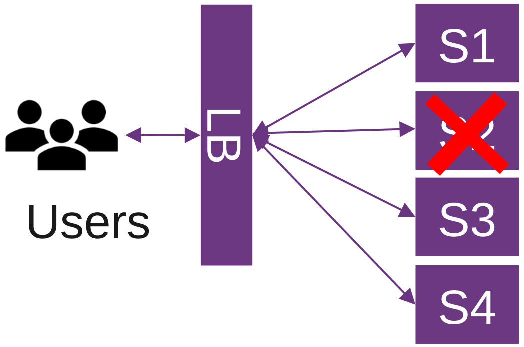
- What is load balancing
  - Distribution of workloads across multiple computing resources
    - Workloads (requests)
    - Computing resources (machines)
  - Distributes client requests or network load efficiently across multiple servers [link]
    - E.g., service get popular, high load on service

→ 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







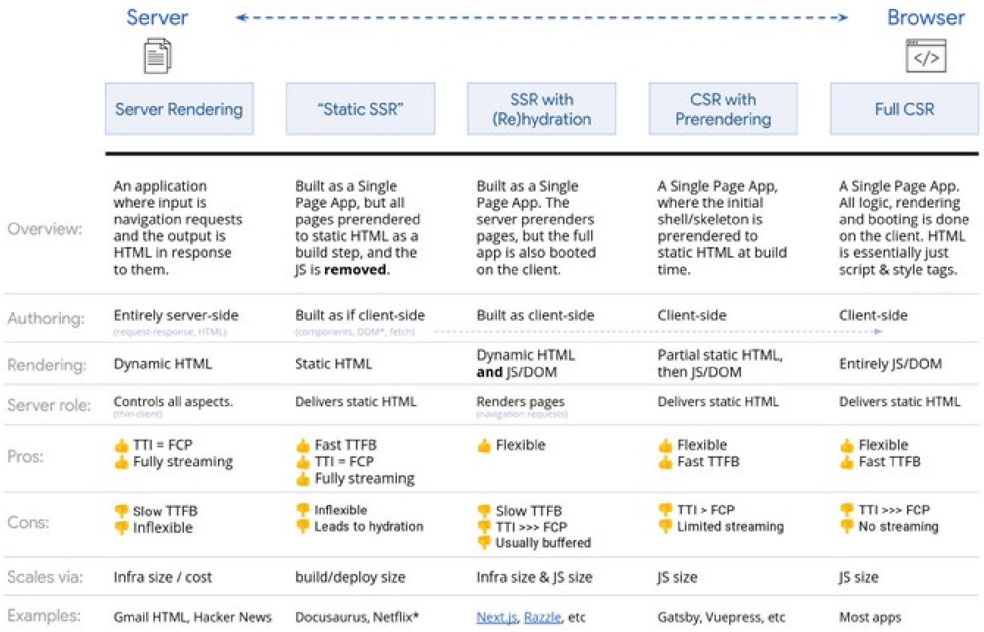
Lecture 6

Web Architectures

Examples

- Static site generation: 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
  - Overview: source



Server ←→ Browser

	Server Rendering	"Static SSR"	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is **removed**.	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side (request-response, HTML)	Built as if client-side (components, DOM*, fetch)	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML **and** JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. (thin client)	Delivers static HTML	Renders pages (navigation requests)	Delivers static HTML	Delivers static HTML
Pros:	👍 TTI = FCP 👍 Fully streaming	👍 Fast TTFB 👍 TTI = FCP 👍 Fully streaming	👍 Flexible	👍 Flexible 👍 Fast TTFB	👍 Flexible 👍 Fast TTFB
Cons:	👎 Slow TTFB 👎 Inflexible	👎 Inflexible 👎 Leads to hydration	👎 Slow TTFB 👎 TTI >>> FCP 👎 Usually buffered	👎 TTI > FCP 👎 Limited streaming	👎 TTI >>> FCP 👎 No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	Next.js, Razzle, etc	Gatsby, Vuepress, etc	Most apps

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.

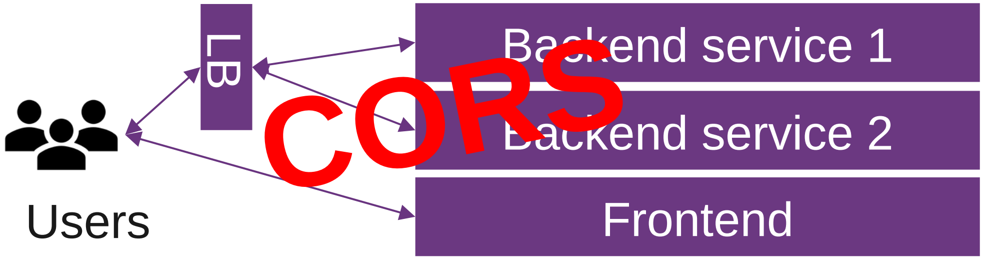
→ The client only sees the same origin for the API and the frontend assets

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

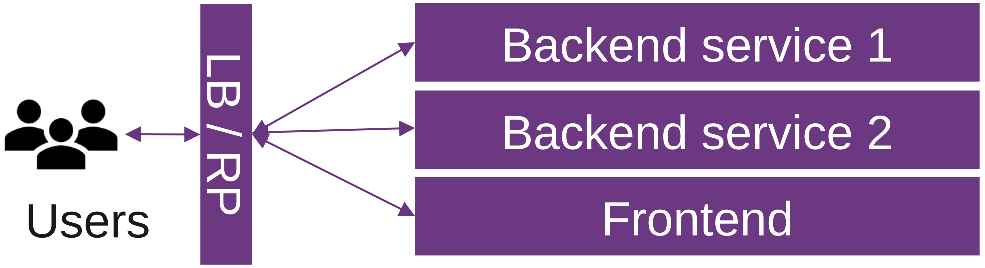
→ For dev: Access-Control-Allow-Origin: *

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

- Reverse proxy



LB
Backend service 1
Backend service 2
Frontend
CORS
Users



LB / RP
Backend service 1
Backend service 2
Frontend
Users