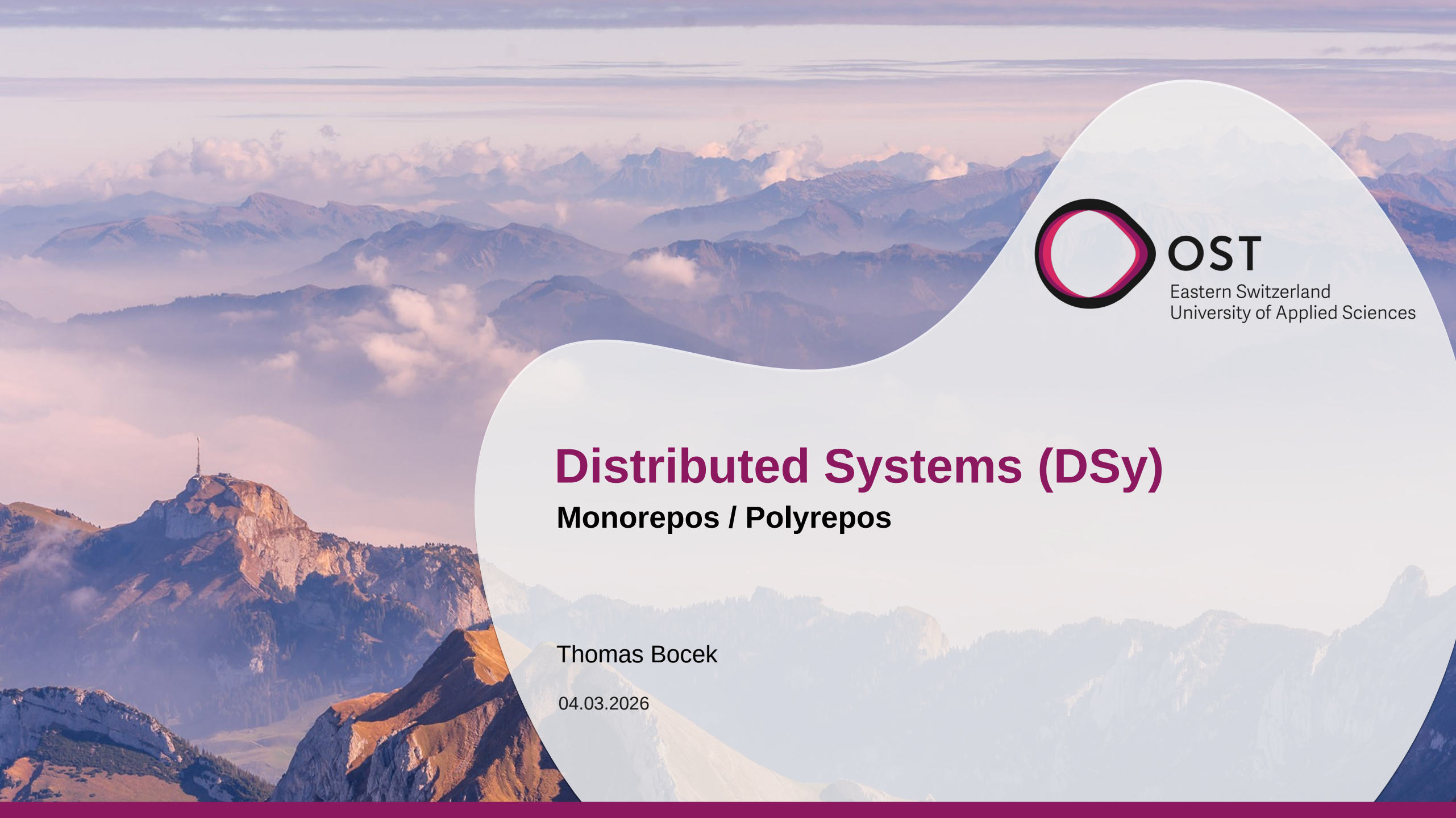


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Monorepos / Polyrepos

Thomas Bocek

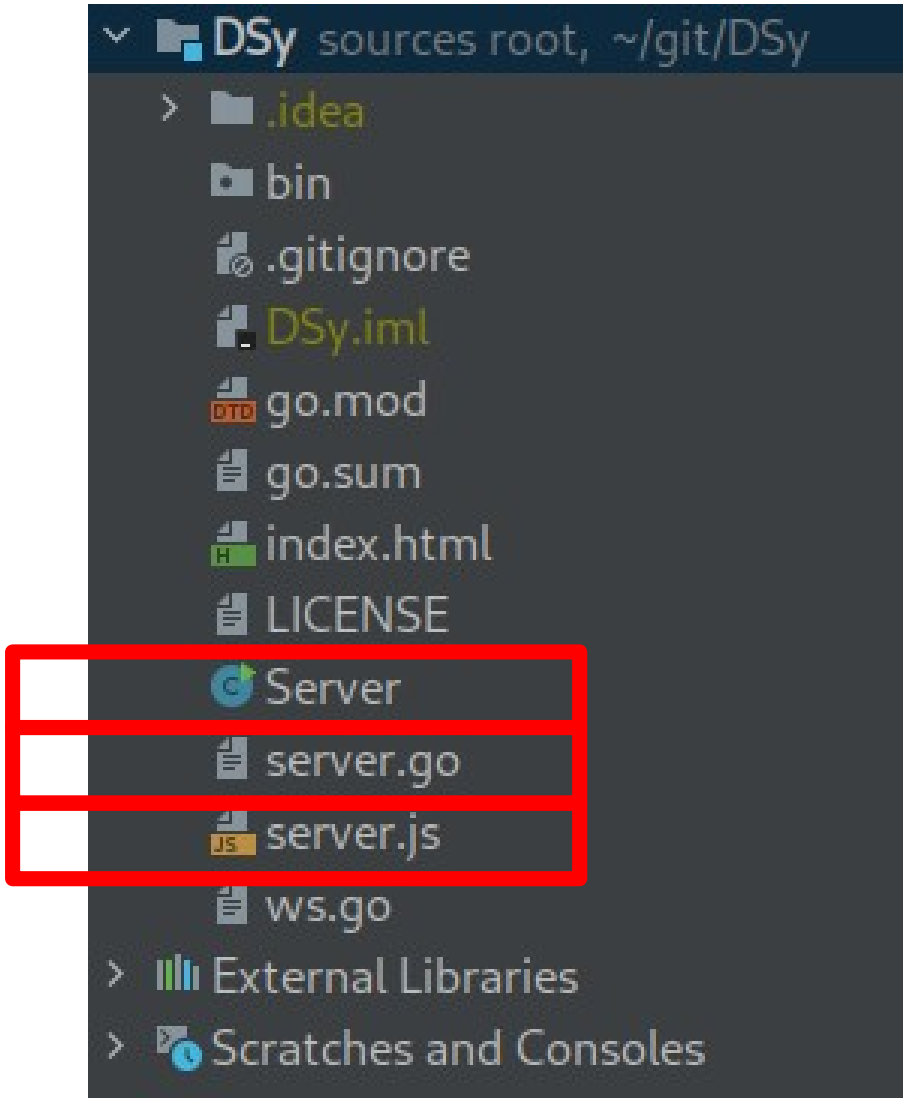
04.03.2026

Learning Goals

- Lecture 4 (Repositories)
  - What is a monorepo, what is a polyrepo?
  - When to use which type?
  - How do AI coding agents change the monorepo vs. polyrepo decision?

Project Setup

- Project setup the wrong way:
  - https://github.com/tbocek/DSy
  - Everything (Java, Golang, Javascript) flat in one directory – do not do this
    - Unless you want to show in the lecture how to different programming languages can communicate...
- But, keep it simple at the start, follow best practices [example]
  - Java: src/main/java – default in maven/gradle
  - Golang: keep it flat: “A basic Go package has all its code in the project’s root directory. [source]” [1, 2, 3]
  - Javascript/Typescript
    - Depends on type: backend / frontend / plugin
    - src, public, tests [1, 2, 3, 4]



OST

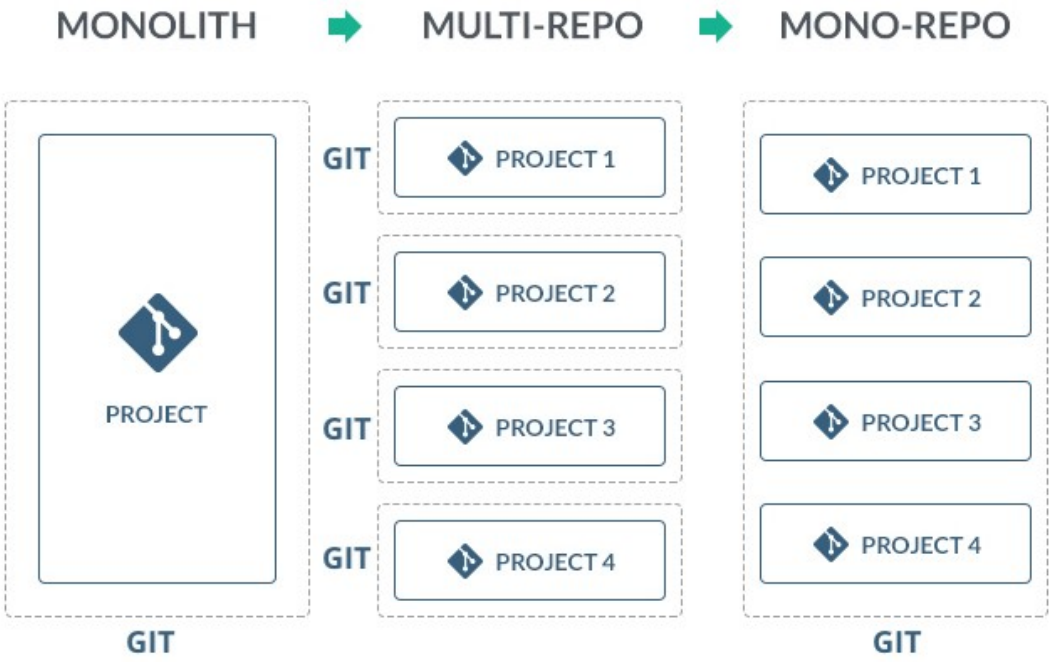
Project Setup

- General rules:
  1) Be consistent
  2) Other projects always prefer other structures
  3) A perfect structure does not exist
  4) Every project has a Readme

- Split up
  - Backend, frontend in separate repositories
  - ~1 technology per split for simple projects
    - Most likely you won't have a frontend mix of frontend technologies e.g., Angular with Vue
    - Sometimes you have a script directory, with different languages (bash, javascript)
  - Keep config out of code (secrets, API keys → .env, not committed)
  - More complex setups?
    - Multiple backends, multiple packages

Monorepo (OneRepo / UniRepo)

- One repository for all projects
  - 1 sub-directory with frontend, 1 sub-directory with backend, etc.
  - Tools (JS/TS): e.g., turborepo, Nx
  - Tools (generic): e.g., Bazel, Buck2, Pants
- Examples
  - Simform: started with monorepo, switched to mulirepo, now with hybrid approach
  - Google, Facebook, Twitter: use monorepos (others do not)
  - Flatfeestack: used hybrid approach (scripts, submodules), now monorepo (e.g.)
- Easier setup for AI agents

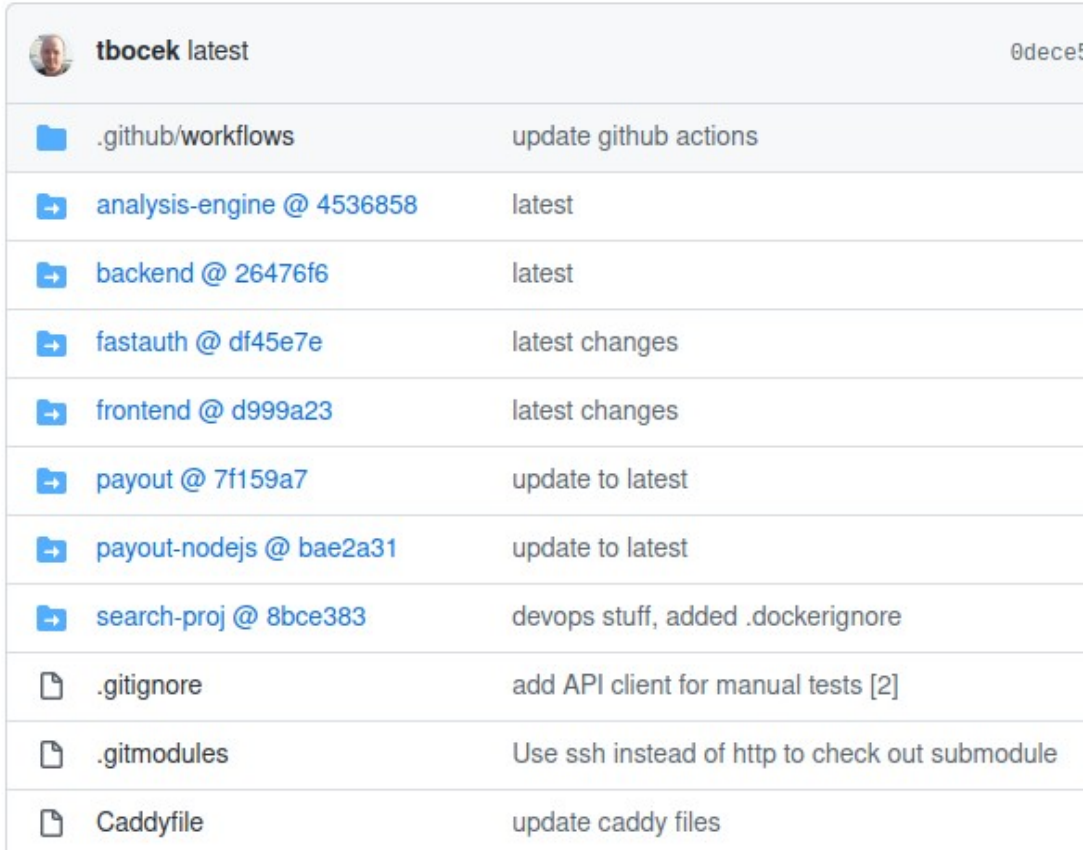


https://codefresh.io/continuous-integration/using-codefresh-with-mono-repos/

OST

Polyrepo (Manyrepo / Multirepo)

- Multiple repositories for a project
  - Frontend in a different repository than the backend
  - Example: https://github.com/flatfeestack
    - Wip...
    - Frontend: Svelte, pnpm
    - Backend: Golang
  - Other names: manyrepo or multirepo
- Sync via git submodules or via bash script
  - Submodules: can also be used as dependency management
- Sync with repo or via bash script



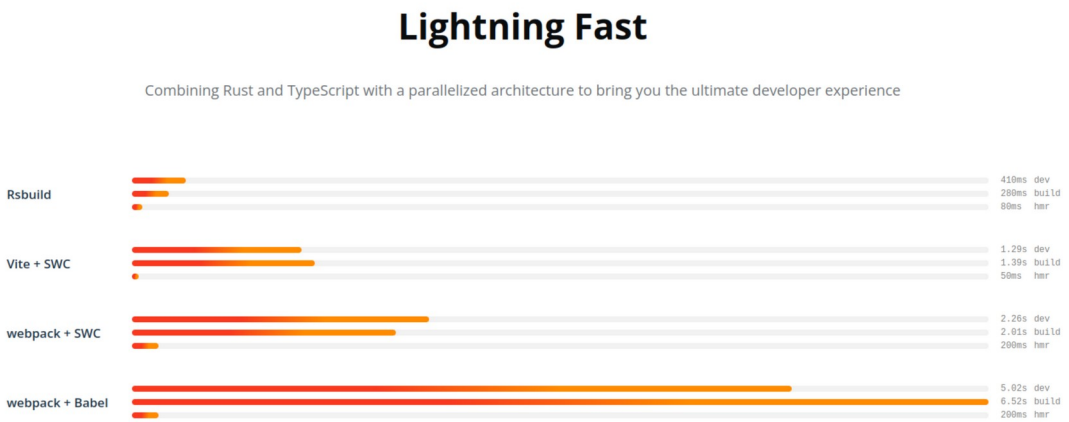
OST

Pro/Cons - Opinion

- **Monorepo**
  - Tight coupling of projects
    - E.g., generating openapi.yml from backend, generate types for frontend → simply copy, or tRPC
  - Everyone sees all code / commits
  - Encourages code sharing within organization
  - AI agents: full cross-project context (backend + frontend in one PR)
  - Atomic commits for cross-project changes
  - Scaling: large repos, specialized tooling
  - Risk: cross-package dependencies (one change can affect many services)
- **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
  - AI agents: possible (--add-dir), but extra setup and token cost
  - Synthetic Monorepo (Nx): bridges repo boundaries without moving code
  - Scaling: many projects, special coordination

Tools

- Overview: https://monorepo.tools
- Pnpm workspace
- Turbo (v2, Rust) / Nx – JS/TS, Gradle monorepo support, otherwise you need to do it manually, or write a script / make / just
  - Caching!
  - E.g., Bazel: (big projects) → define input/output → if no change in input, do not run command. Buck2: same concept
  - But e.g., Golang, Rust have extensive caching mechanisms
  - Caching with docker in upcoming lecture
- Many tools competing in the same space
- Complexity!
  - Is build speed slow, faster tools? Rsbuild, tsgo



OST