

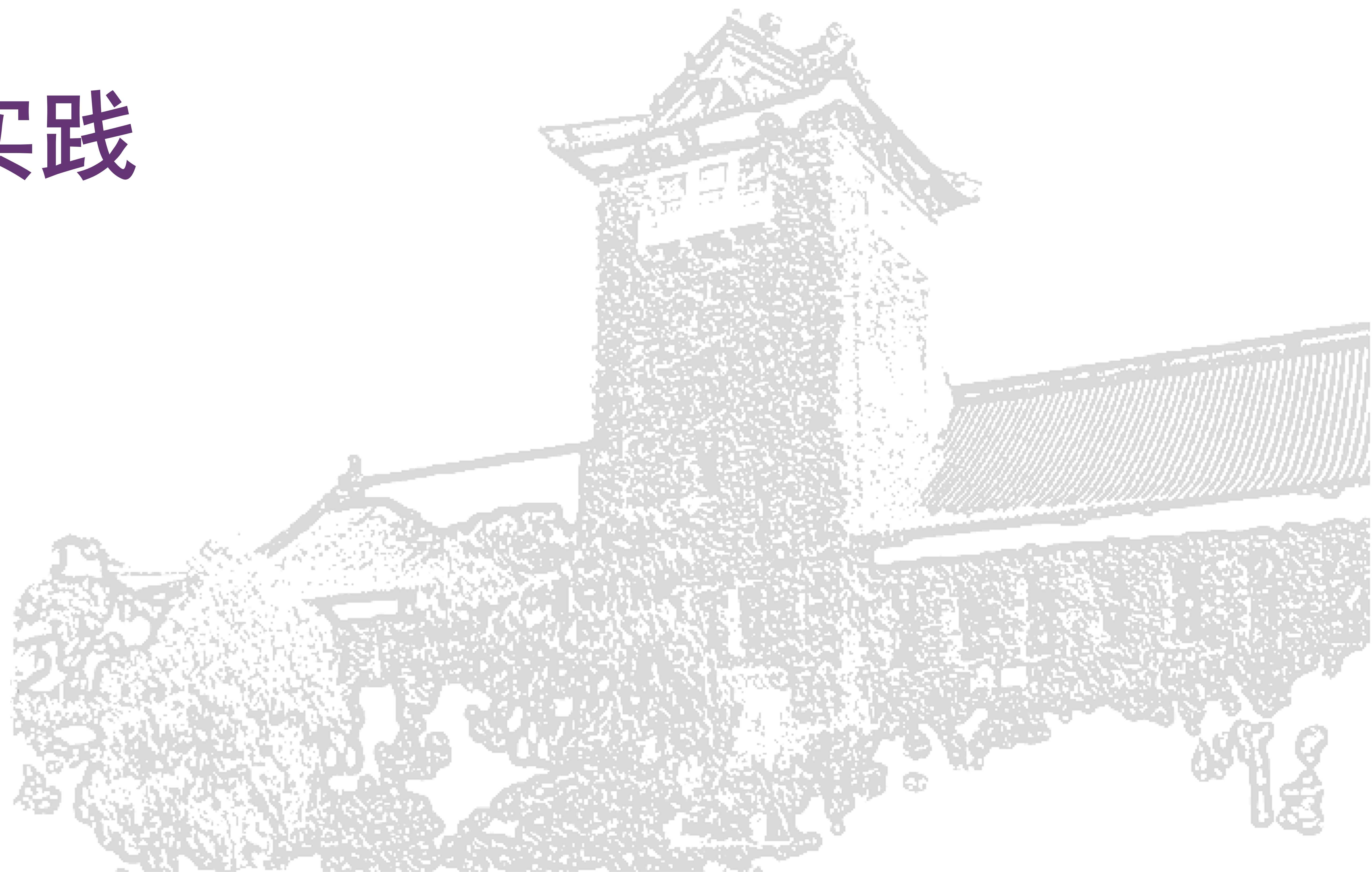


Monorepo 理念与实践

姜佳良

南京大学计算机学院

2025.8.23





目录

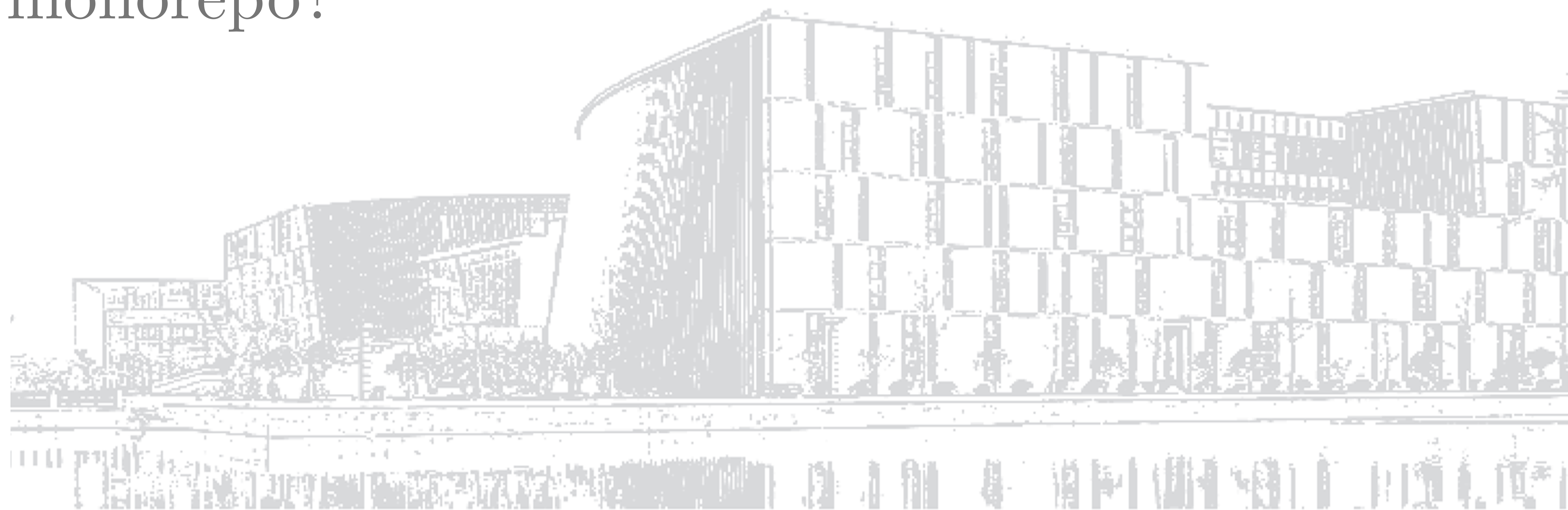
- Monorepo 理念
 - 什么是 Monorepo
 - 为什么选择 Monorepo
- Monorepo 现状
 - 开发方式
 - 基础设施
 - 当前挑战
- Monorepo 实践
 - rk8s 容器编排系统
 - mega 代码管理平台





Monorepo 理念

什么是 monorepo? 为什么选择 monorepo?

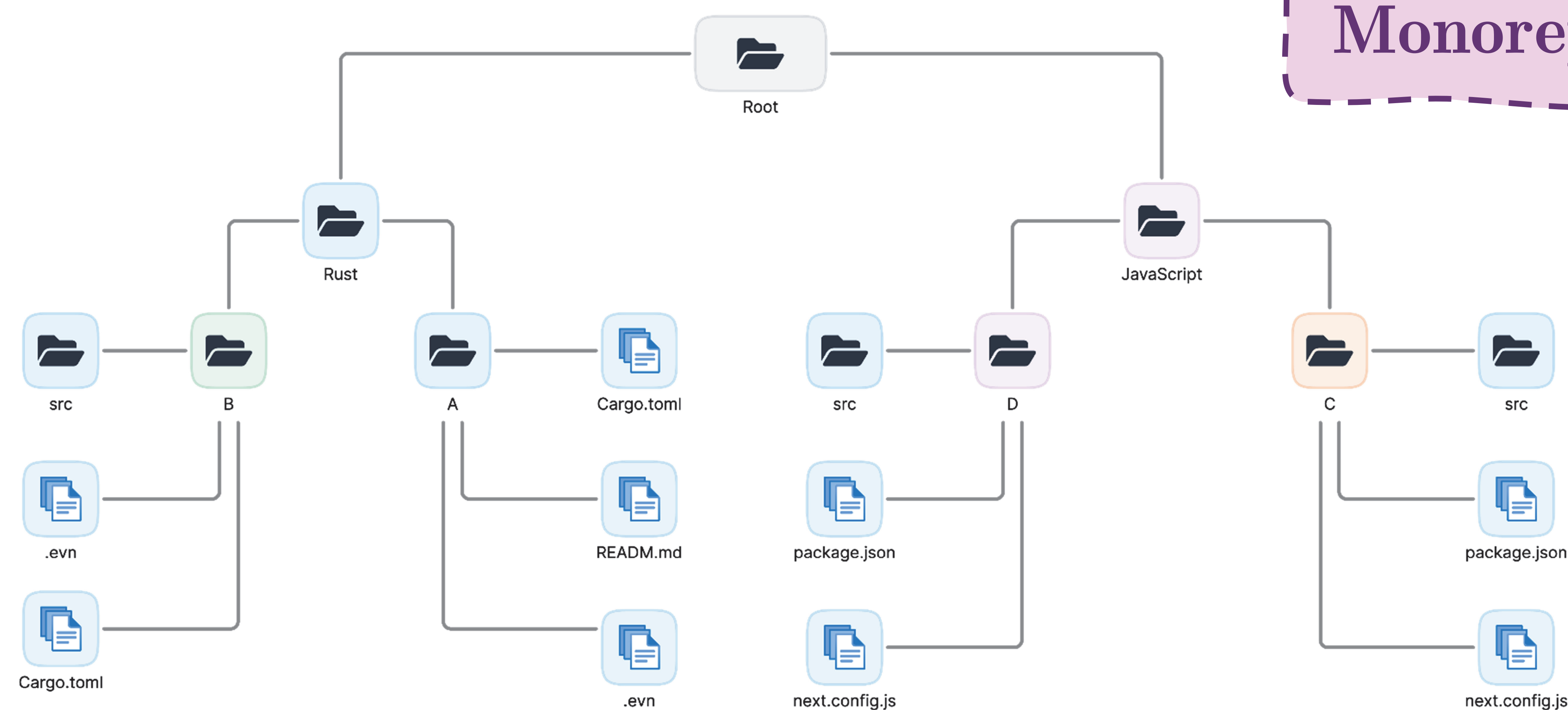




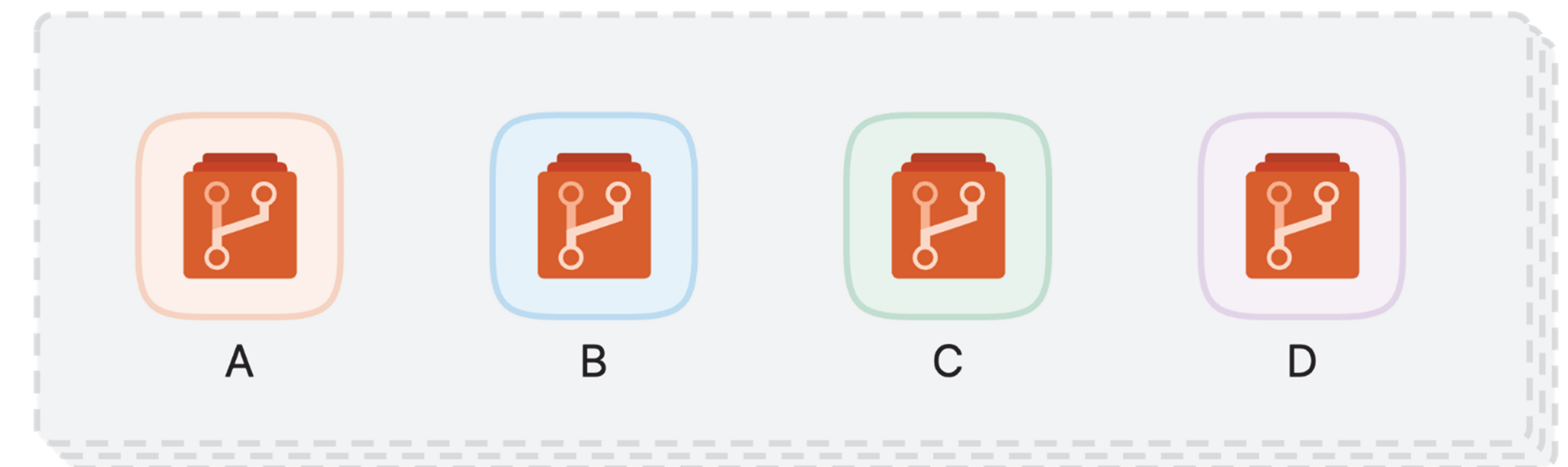
什么是 monorepo

What is monorepo?

- 单个仓库中管理多个项目的代码管理方式
- 统一代码库、共享依赖、跨项目协作



Monorepo VS Polyrepos



A single repository that contains all the source code for a project, group of related projects or whole company's codebase. Monorepos can simplify code sharing and synchronization, but they can have scalability issues.

A software development approach where each project or component is stored in a separate repository. Polyrepos allow for finer access control and independent development cycles, but they can struggle with dependency management.



为什么选择 monorepo

Why monorepo?

- 依赖管理更简单
 - 依赖版本统一升级
 - 公共组件集中管理
- 团队协作更高效
 - 原子提交
 - 大规模重构
 - 代码复用与共享
- 流水线与代码规范
 - 质量门禁
 - 静态分析
- 代码可见性
 - 清晰的树型结构
 - 灵活的访问权限
 - 统一的发版控制
- 混合语言支持



包管理器 & 依赖管理

Package Manger & Dependency Management

语言	C/C++	Python	Rust	Go	JavaScript
包管理器	vcpkg	pip, poetry, conda, uv ...	cargo	go mod	npm/pnpm, yarn
配置文件	CMakeLists.txt	requirements.txt	Cargo.toml	go.mod	package.json
解析方式	手动	扁平依赖 (pip)	依赖图解析 feature filter	树状嵌套	最小版本选择 (MVS)
原生支持	✗	⚠	✓	✓	✓
依赖隔离	无标准方案 (依靠构建系统)	venv/conda/uv/ poetry env	工作区/项目级 隔离	模块化	node_modules 目录隔离
中心仓库	无标准方案	PyPI	crates.io	github	npm registry



包管理器 & 依赖管理

Package Manger & Dependency Management

语言

C/C++

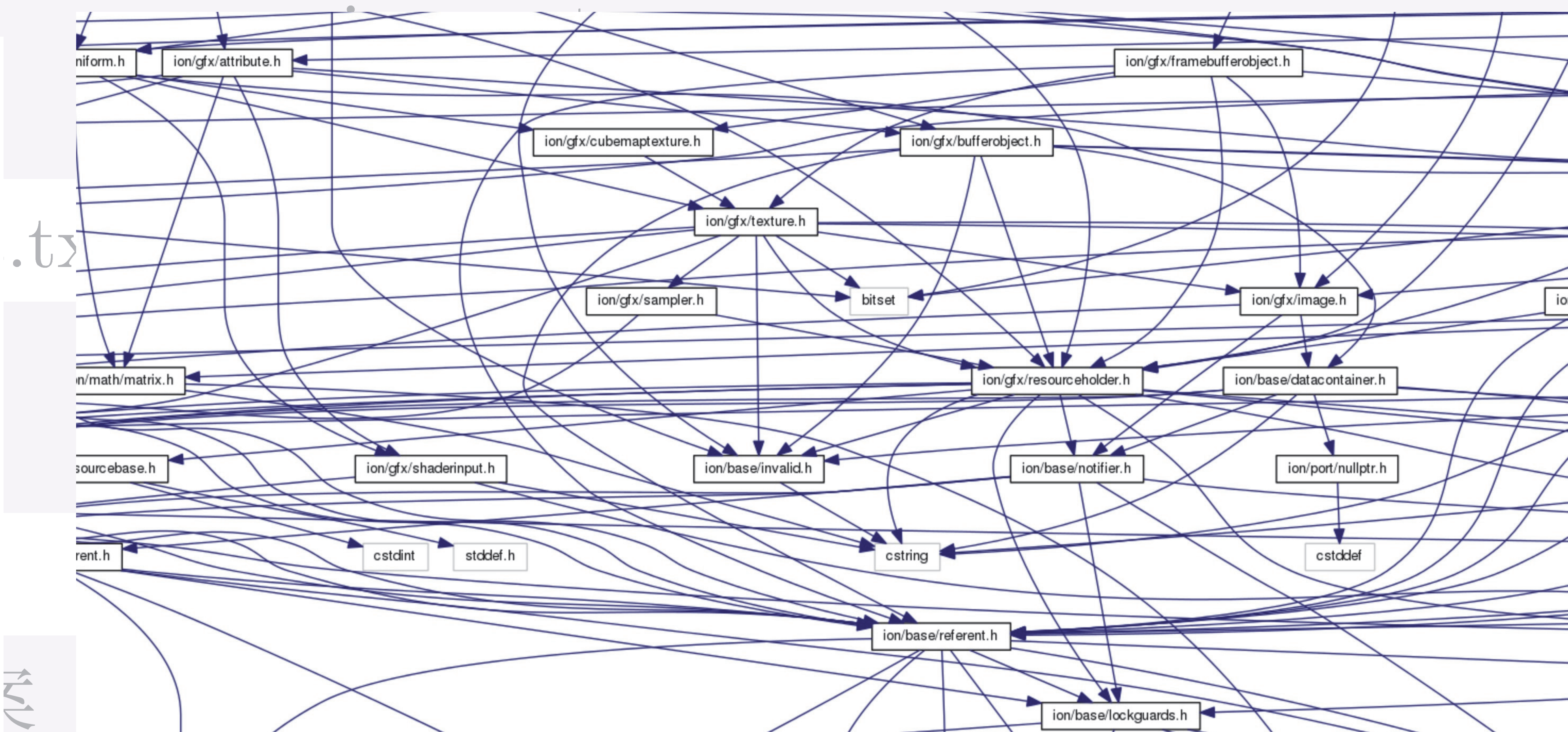
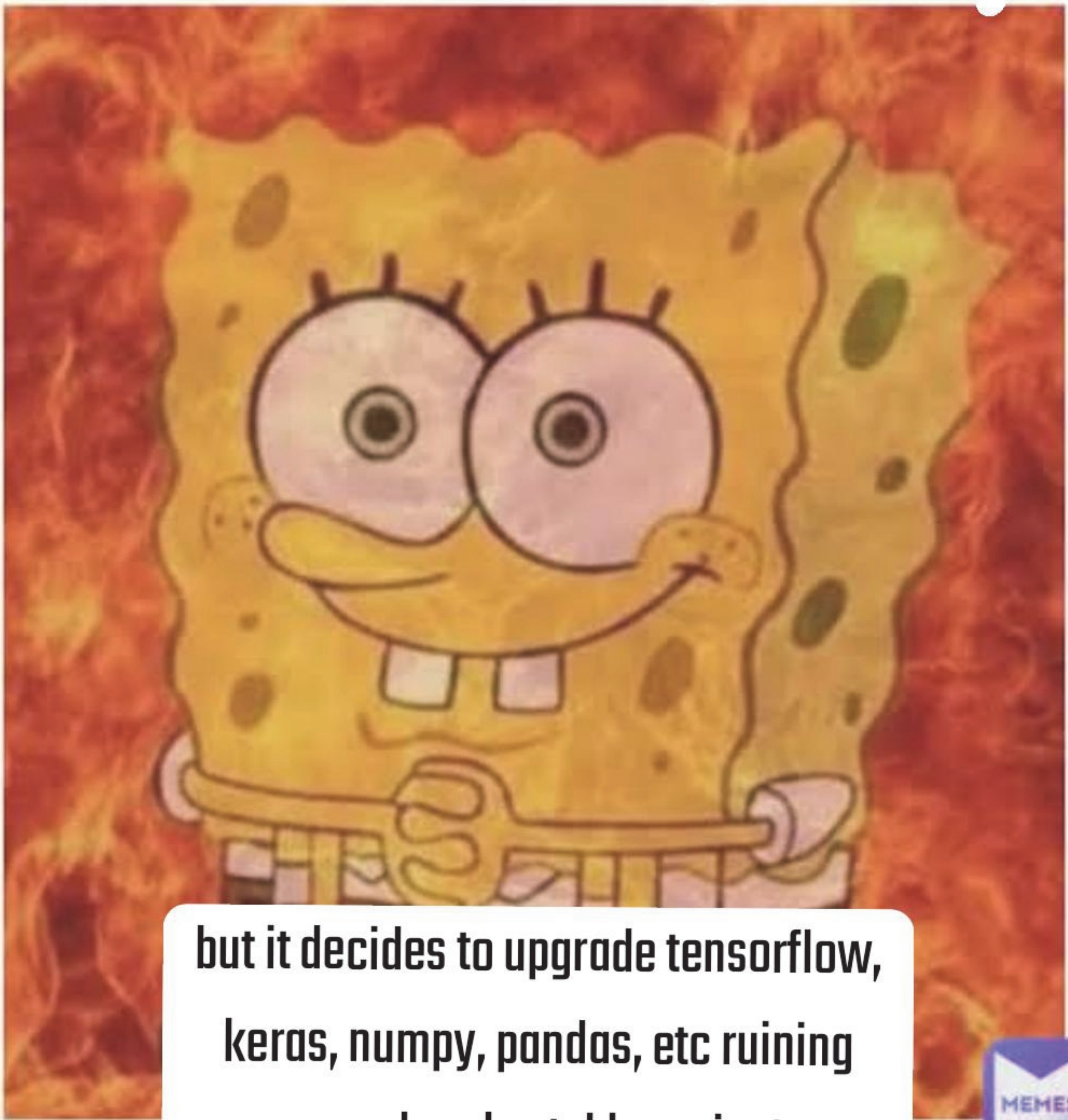
Python

Rust

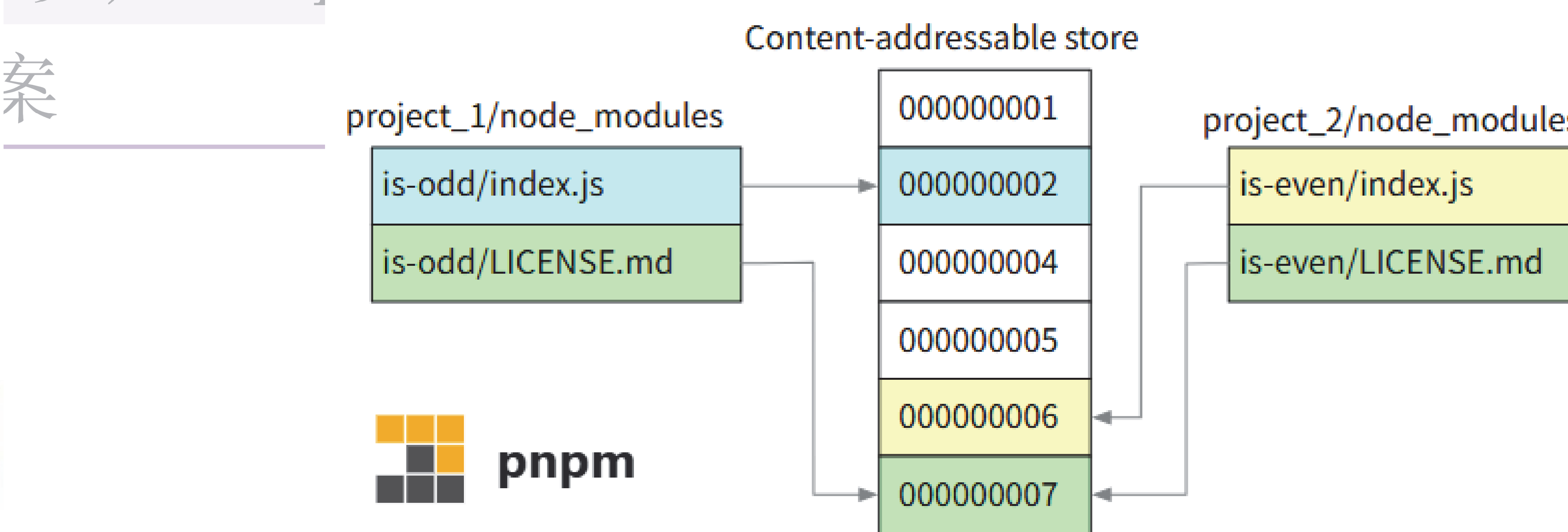
Go

JavaScript

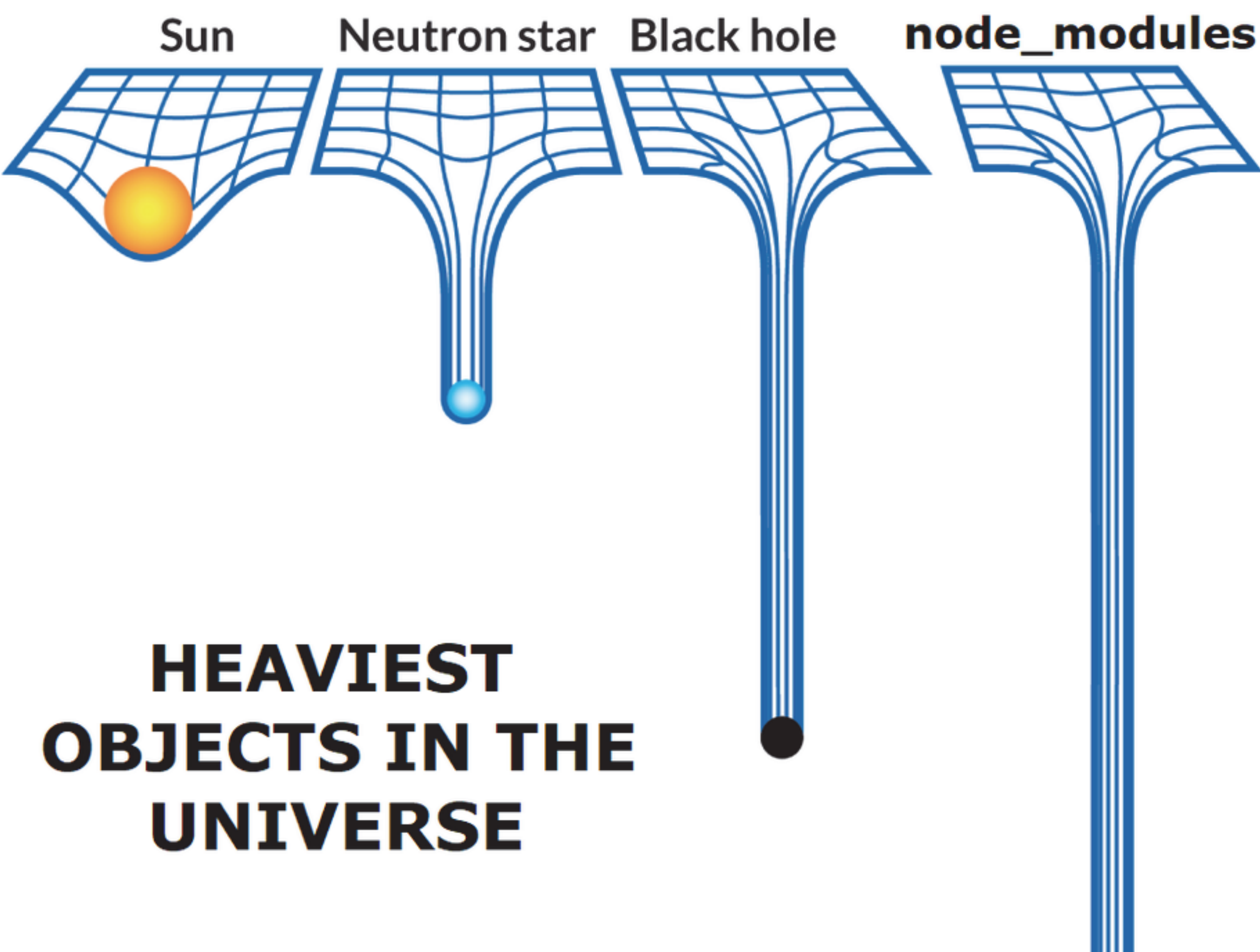
when you try to install
a new library in python



poetry env 隔离



npm/pnpm, yarn
package.json
最小版本选择 (MVS)



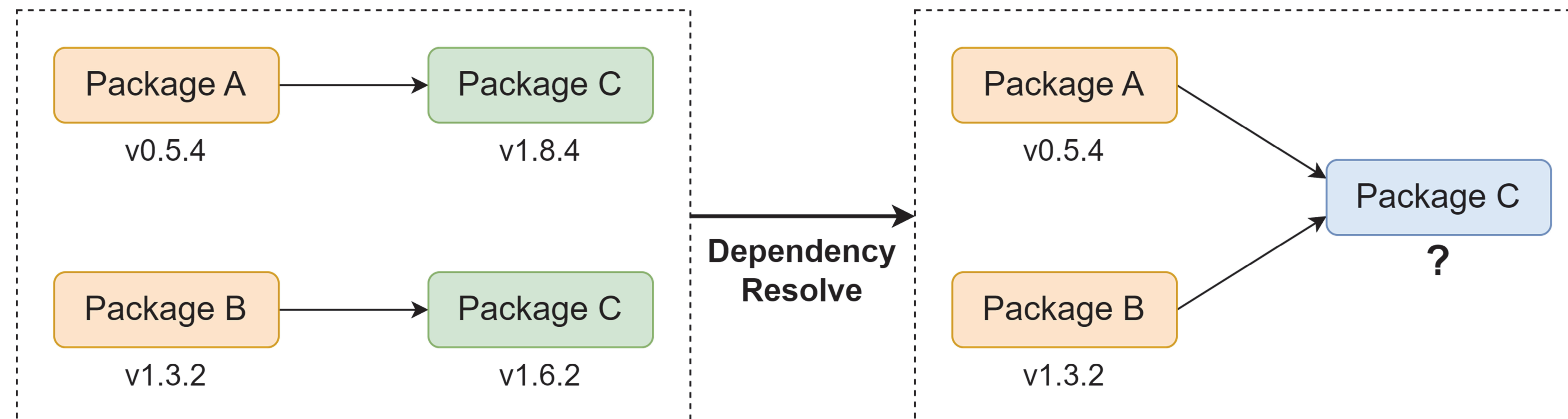
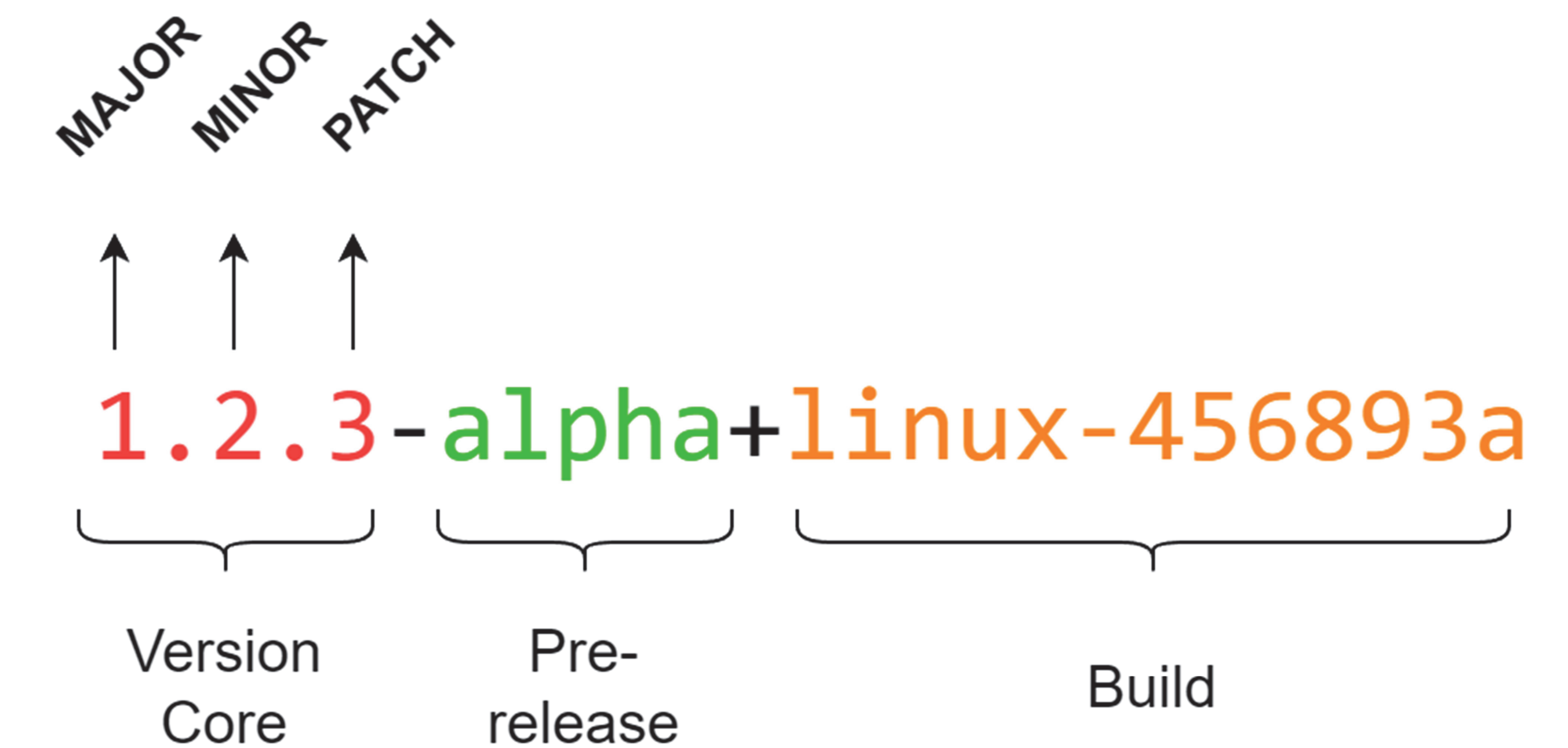
HEAVIEST
OBJECTS IN THE
UNIVERSE



语义化版本

Semantic Versioning

- 版本格式：主版本号.次版本号.修订号
 - 主版本号：当你做了不兼容的 API 修改，
 - 次版本号：当你做了向下兼容的功能性新增，
 - 修订号：当你做了向下兼容的问题修正。





混合编程

Hybrid Programming

- Native for performance, script for expansibility



Buck2



Kubernetes 2.0

- Foreign Function Interface

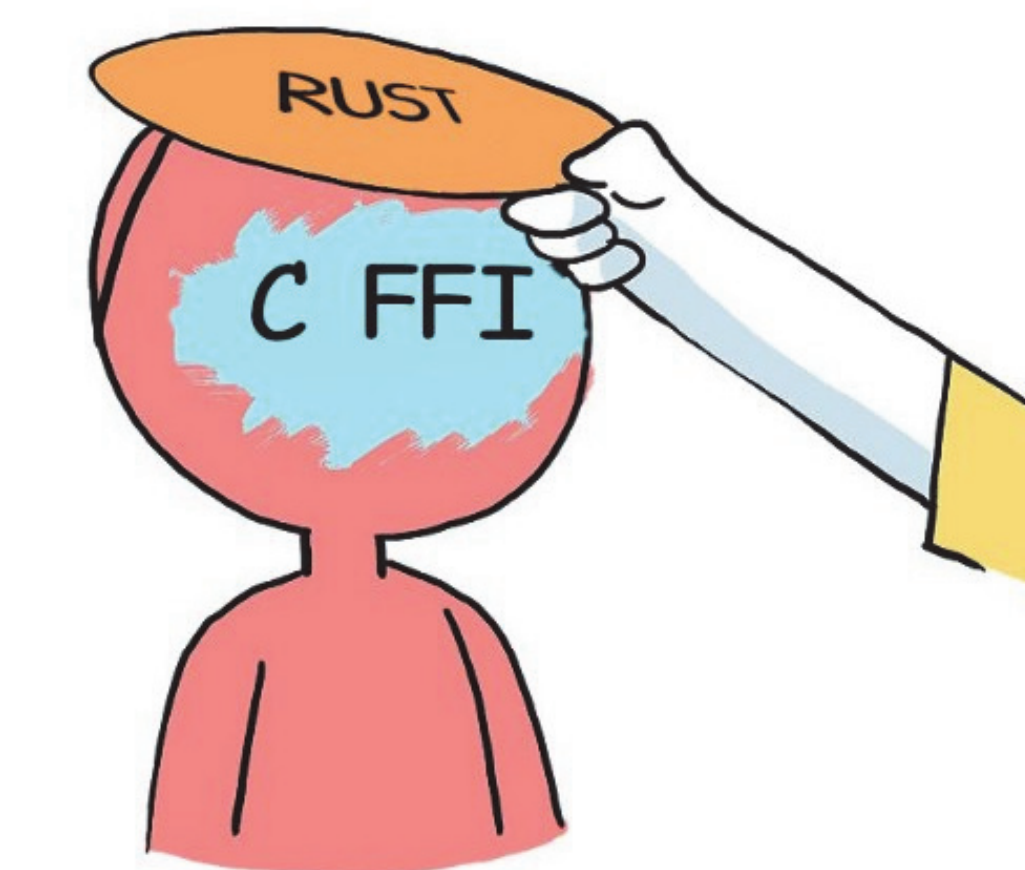
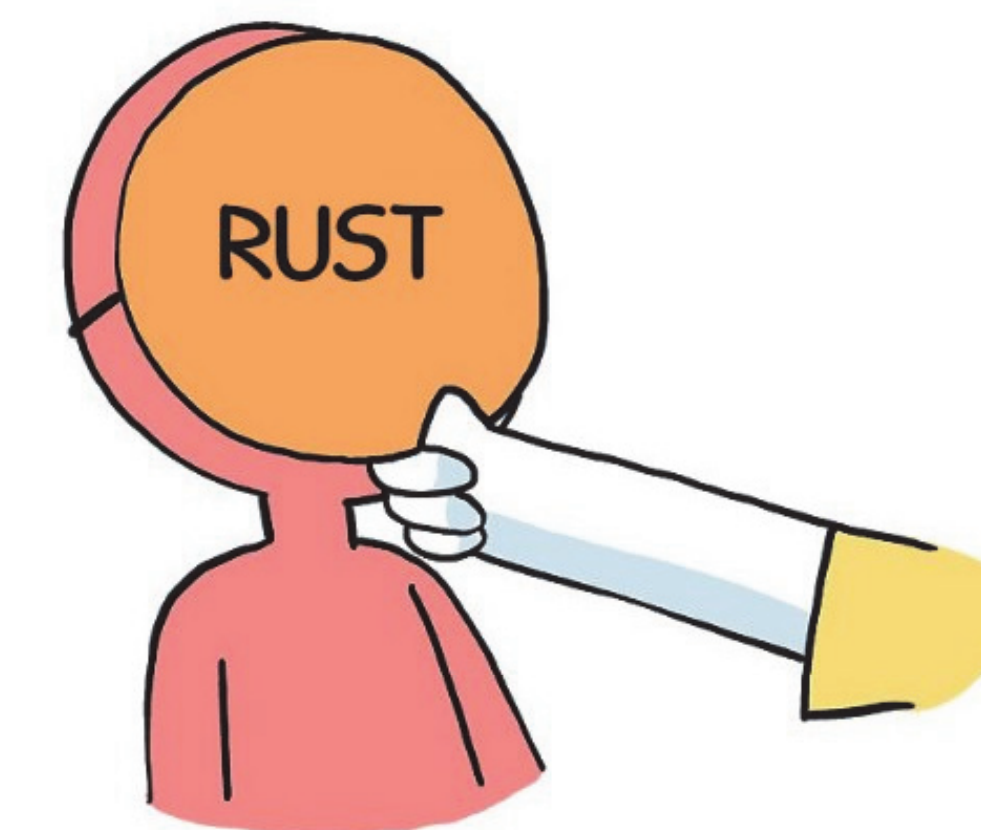
- Python
- Rust

- Compiler

- Cpython
- Rustc



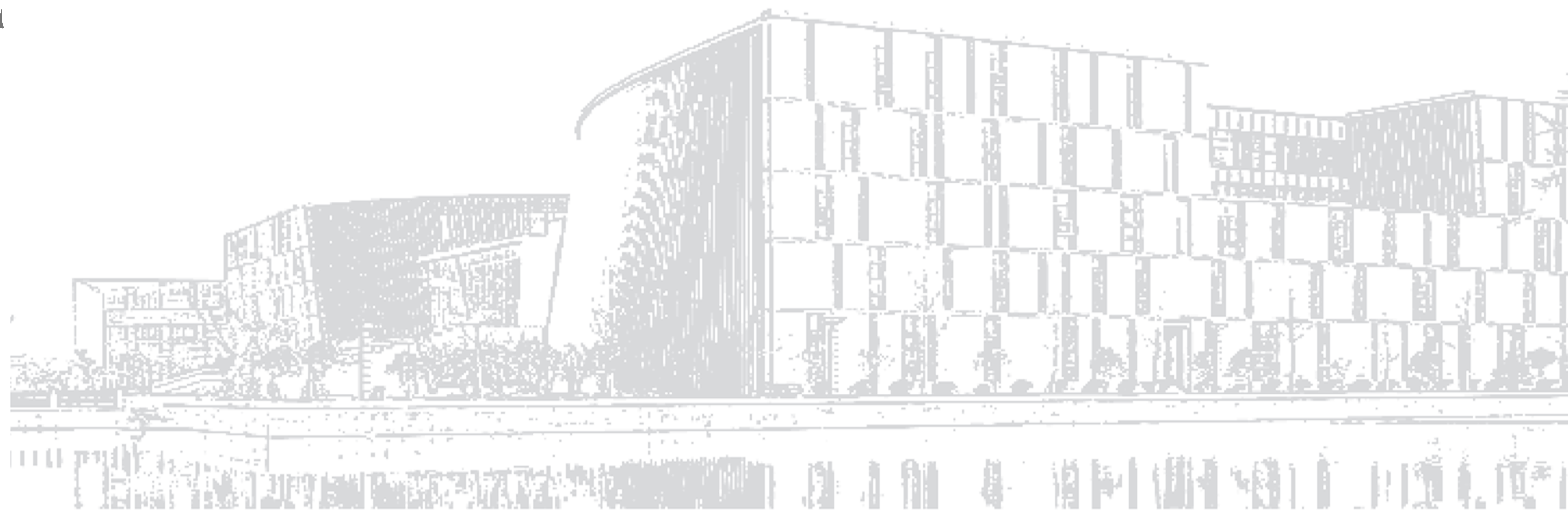
HEY RUST, WHY DO YOU ALWAYS WEAR THAT MASK?





Monorepo 现状

开发方式、基础设施与当前挑战





当前挑战

Current Challenges

- 仓库体积膨胀
 - 克隆速度慢、本地存储压力大
- 工具链要求高
 - 需要强大的构建系统支持
 - 需要增量构建、变更监控
- 短期成本上升
 - 权限与安全控制更复杂
 - 需要专人维护工具链与依赖库
 - 开发方式发生变化
- 文化背景差异



虚拟文件系统



大型构建系统



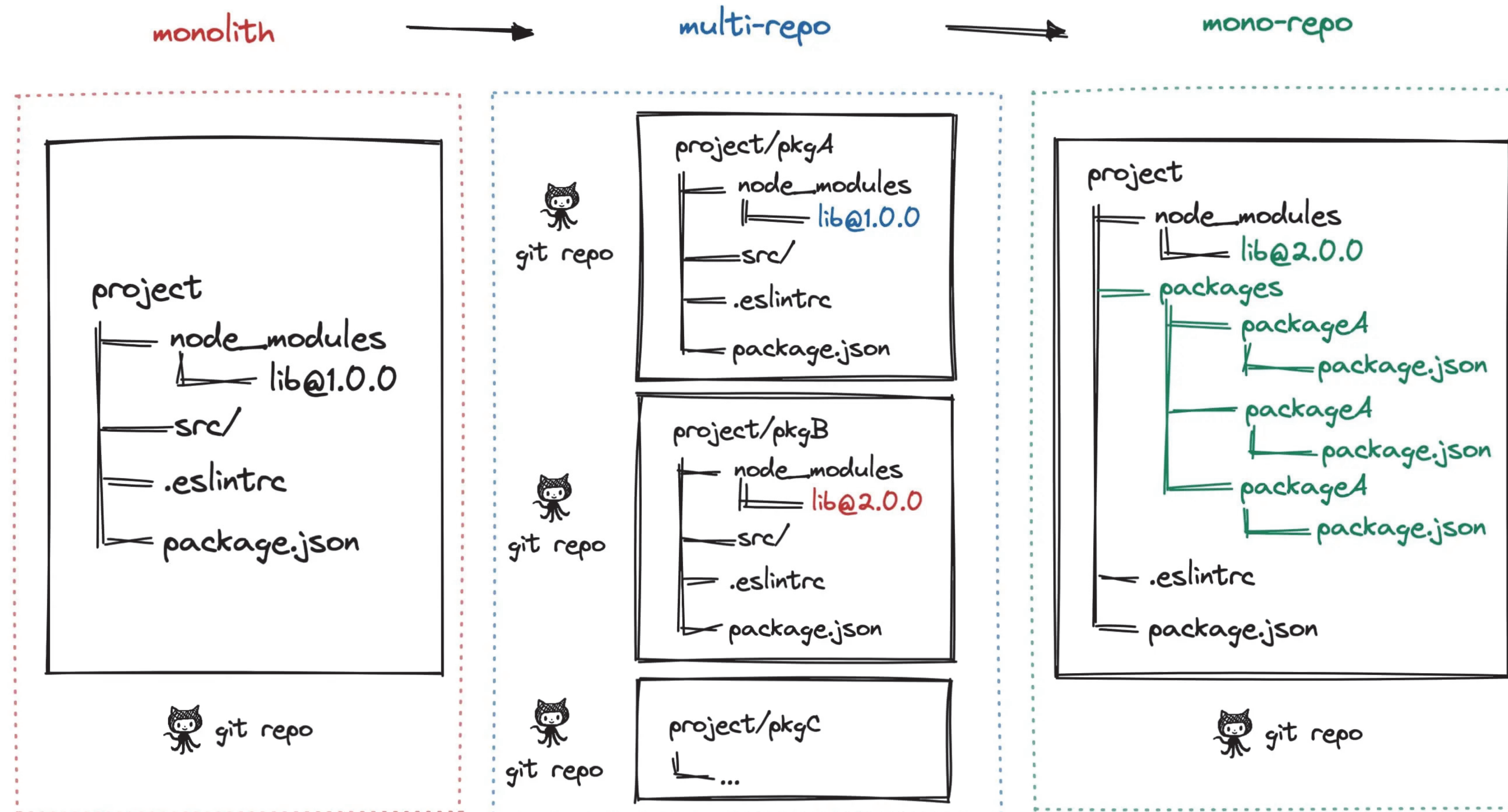
开发模式变革



开发方式的演进与变革

Evolution and Transformation of Development Practices

- 单仓库单项目 → 多仓库多项目 → 单仓库多项目

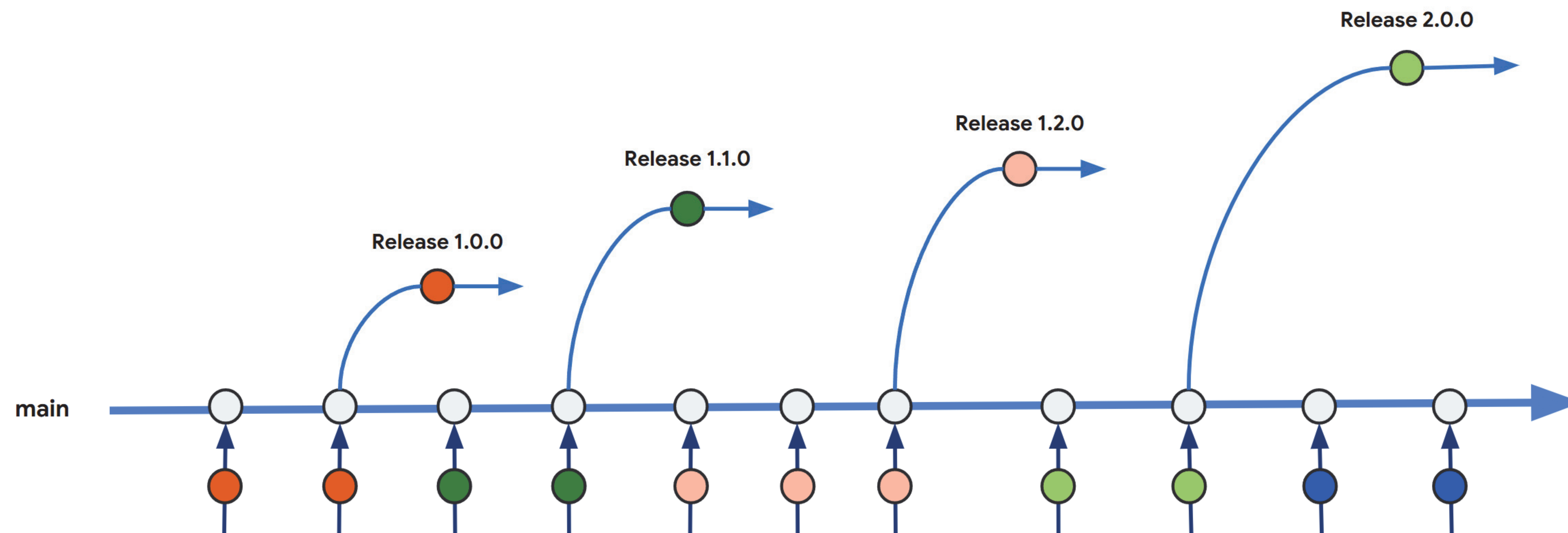




主干开发

Trunk-Based Development

- 在单一的“主干”（trunk）分支上协作开发，不再额外创建其他长期存在的开发分支
- 直接向主干分支提交代码，或者采用 Pull-Request 的方式，确保特性分支不会长期存在，但是产品独立存在

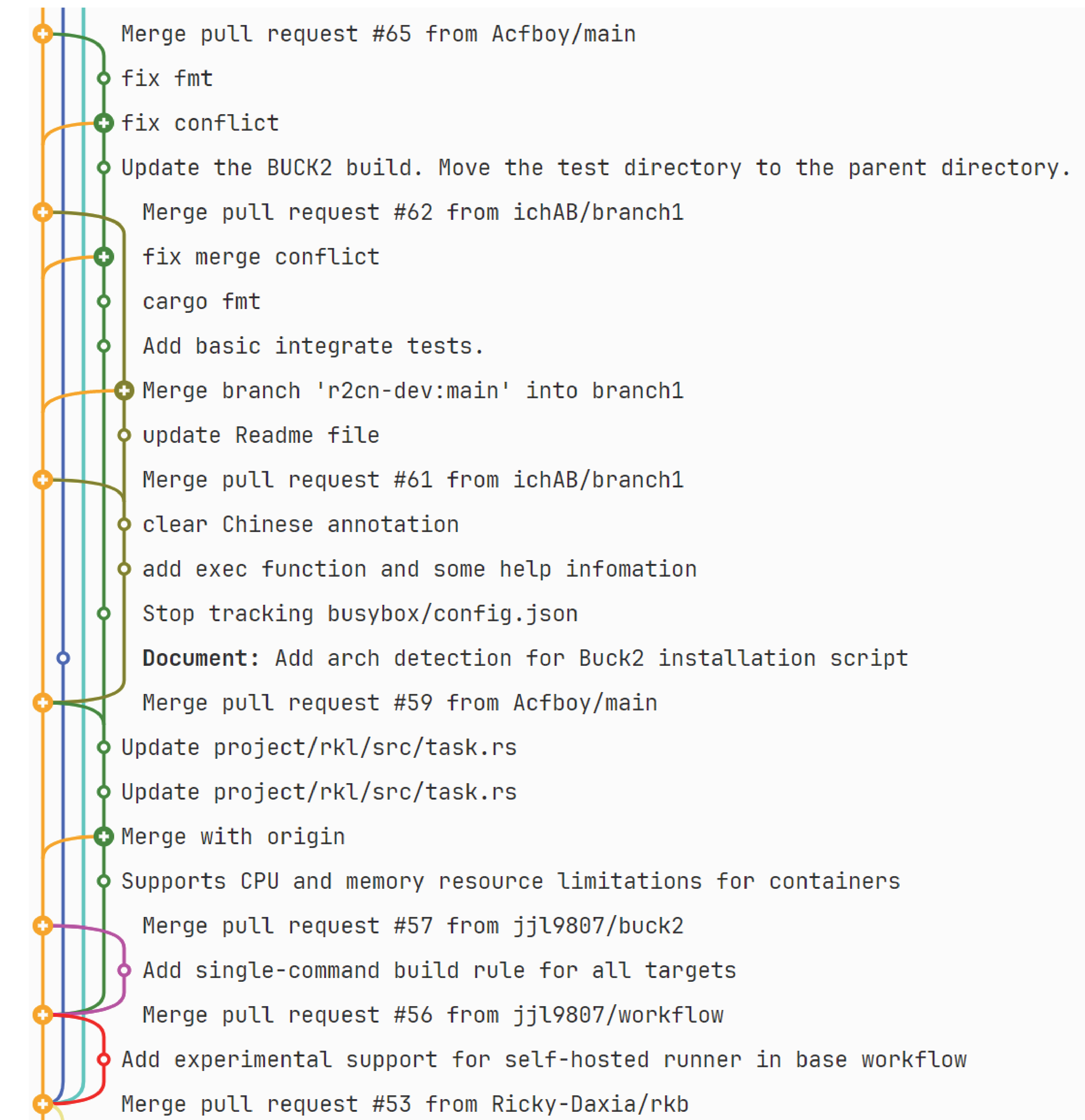
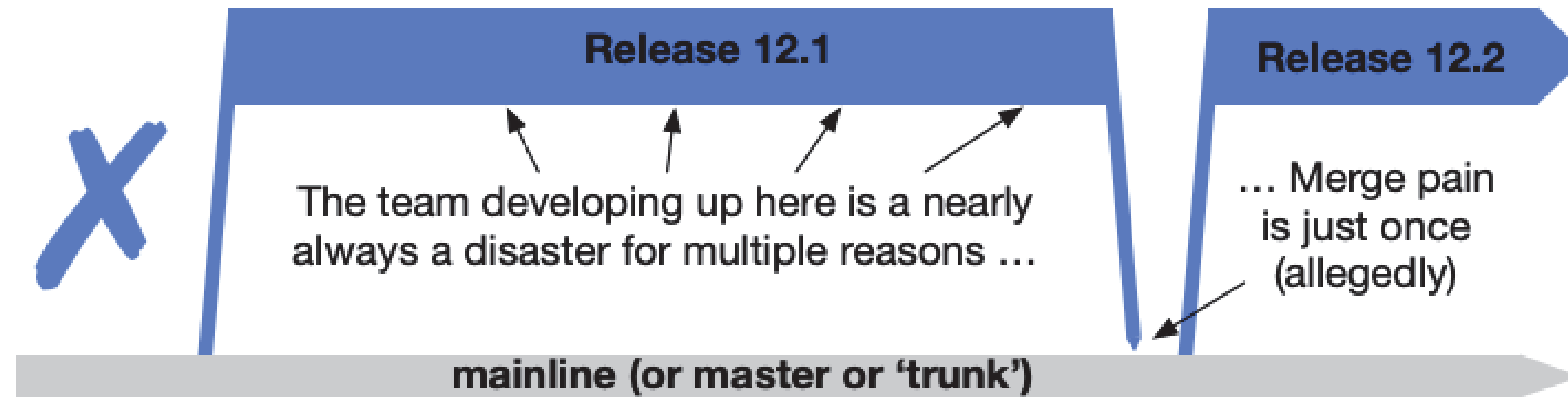




主干开发

Trunk-Based Development

- Shared branches off mainline/main/trunk are **bad** at any release cadence

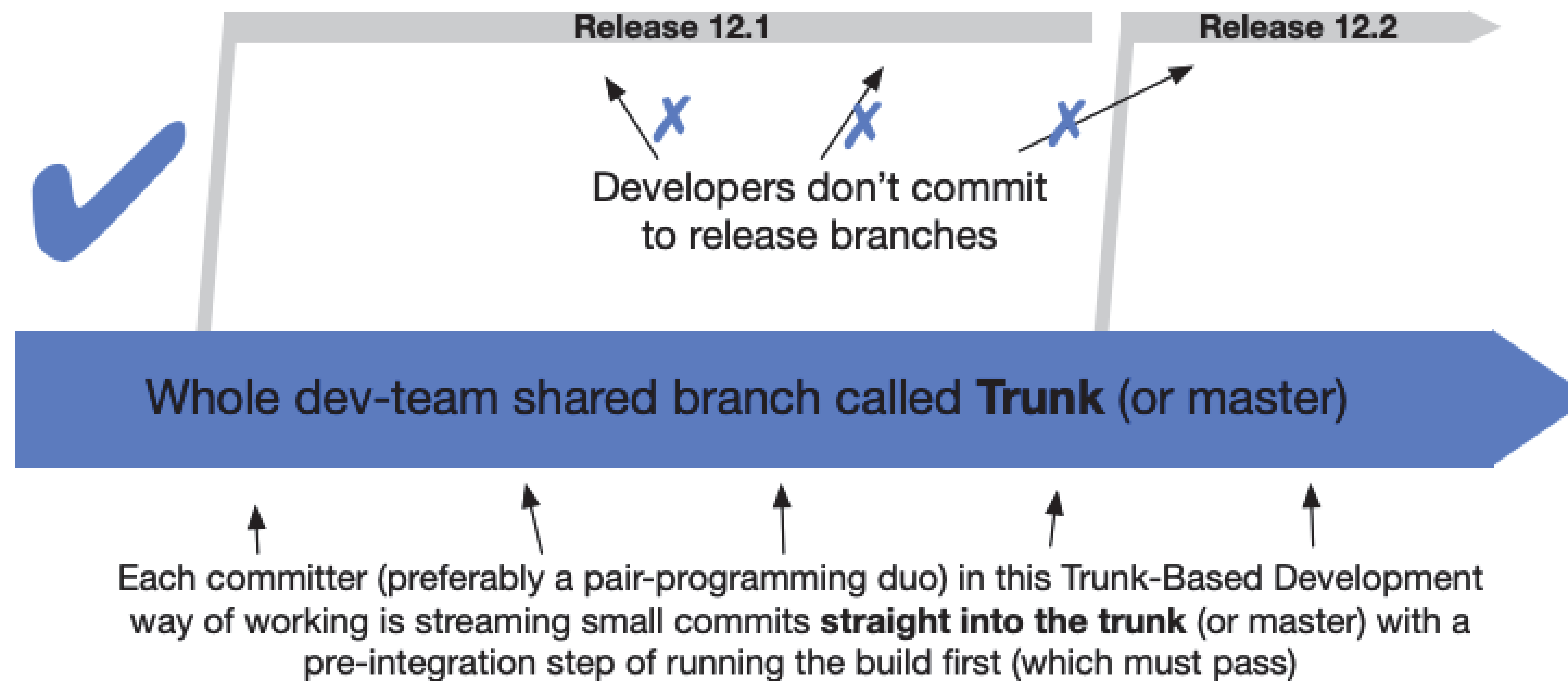




主干开发

Trunk-Based Development

- Trunk-Based Development For Smaller Teams

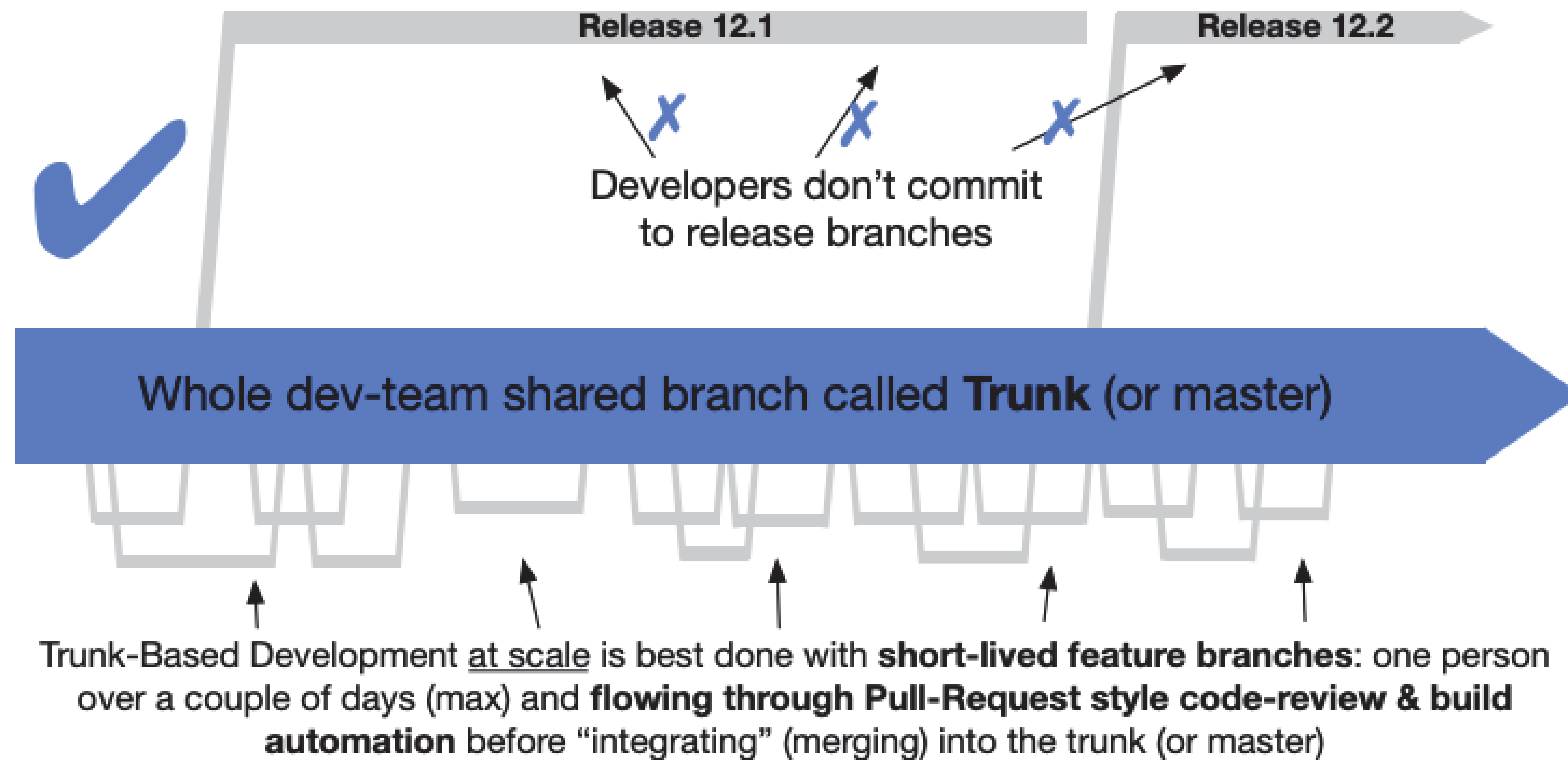




主干开发

Trunk-Based Development

- Scaled Trunk-Based Development

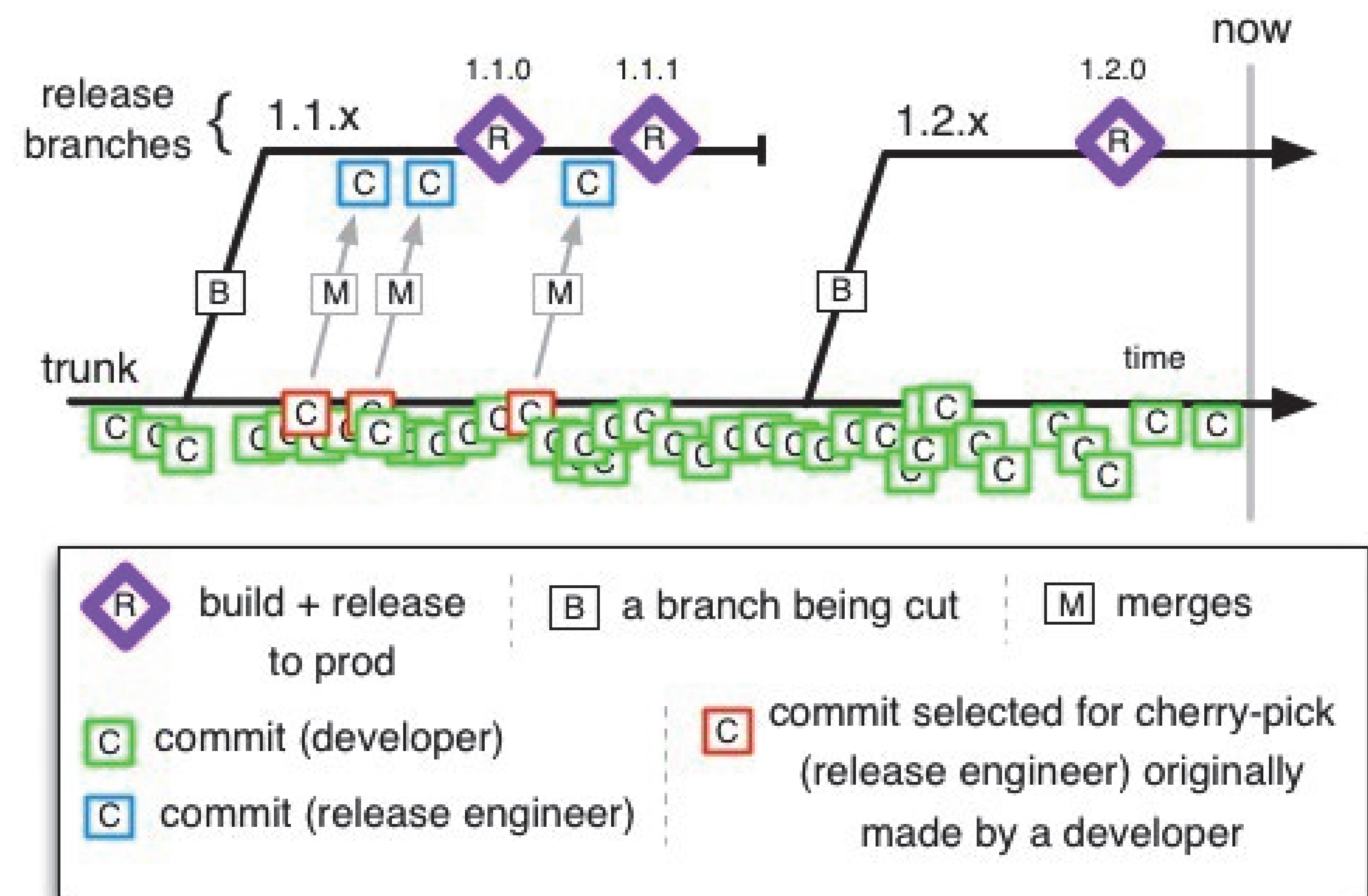




主干开发

Trunk-Based Development

- Fix forward
 - 如果主干没有其他提交，直接 hot fix
 - 否则
 - 从 release Tag 创建发布分支
 - 在主干上进行 Bug Fix 提交
 - 将 Bug Fix 提交 *Cherry Pick* 到发布分支
 - 为发布分支打上新的 Tag 并重新 release





基础设施

Infrastructure for monorepo

Google Piper

- January, 2015
 - 86TB, size of Content
 - 35 million, commit
 - 2 billion, lines of code
 - 9 million, source files
 - 1 billion, files
- Piper stores a single large repository and is implemented on top of standard Google infrastructure, originally Bigtable, now Spanner Piper is distributed over 10 Google data centers around the world, relying on the Paxos algorithm to guarantee consistency across replicas - “[Why Google Stores Billions of Lines of Code in a Single Repository](#)” on ACM
- CitC - Client in the Cloud
- Not open source

Facebook Sapling

- When used in conjunction with the Sapling server and virtual filesystem (not yet available publicly), Sapling scales to repositories with 10's of millions of files, commits, and branches. While inside Meta it is primarily used for our large monorepo, ... - “Sapling SCM” on [sapling-scm.com](#)
- Mononoke - Mononoke is a the server built for the Sapling
 - Production metadata SQL support - SQLite in OSS
 - Production blobstore storage backends - SQL (on sqlite) and File System backends in OSS
- open source

Microsoft Scalar

- Ships inside the fork of Git
- Helps Git scale to large repositories
 - Partial clone
 - Background prefetch
 - Sparse-checkout
 - File system monitor
 - Commit-graph
 - Multi-pack-index
 - Incremental repack
- Scalar is client tools with GVFS protocol
- open source, no open source GVFS server

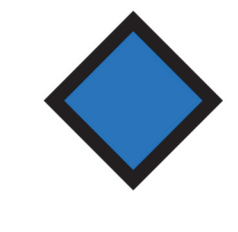


Only client open source



流水线系统 vs. 构建系统

Pipeline System vs. Build System

	构建系统	流水线系统
 主要职责	如何编译代码	如何自动化交付流程
 重点关注	将源代码转换为可执行文件	“构建-测试-部署-发布” 各阶段编排
 核心功能	依赖管理、增量编译、构建缓存	触发机制、阶段编排、并发执行
 常见工具	CMake, Bazel, Cargo, Buck2	Jenkins, GitHub Actions, GitLab CI

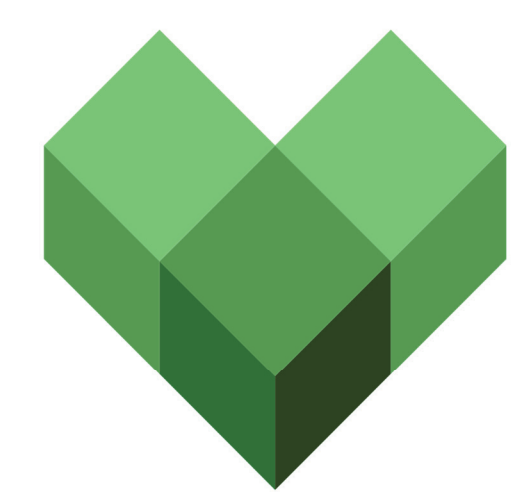
“Build systems answer *how* to compile.
Pipeline systems answer *when, where, and why* to deliver.”



大型构建系统：增量编译、分布式和并行化

Scaled Build Systems: Incremental Compilation, Distribution, and Parallelization

- 增量编译
 - 需要编程语言（编译器）原生支持
- 分布式编译
 - Mozilla sccache 支持分布式构建
 - 上下文同步与工具链依赖
 - 构建任务远程执行
- 并行化构建
 - Buck2 使用 DICE 引擎（Dynamic Incremental Computation Engine）管理和计算依赖图，支持并行化、增量化执行不同阶段的构建任务



Bazel



Buck



Buck2



大型构建系统：增量编译、分布式和并行化

Scaled Build Systems: Incremental Compilation, Distribution, and Parallelization

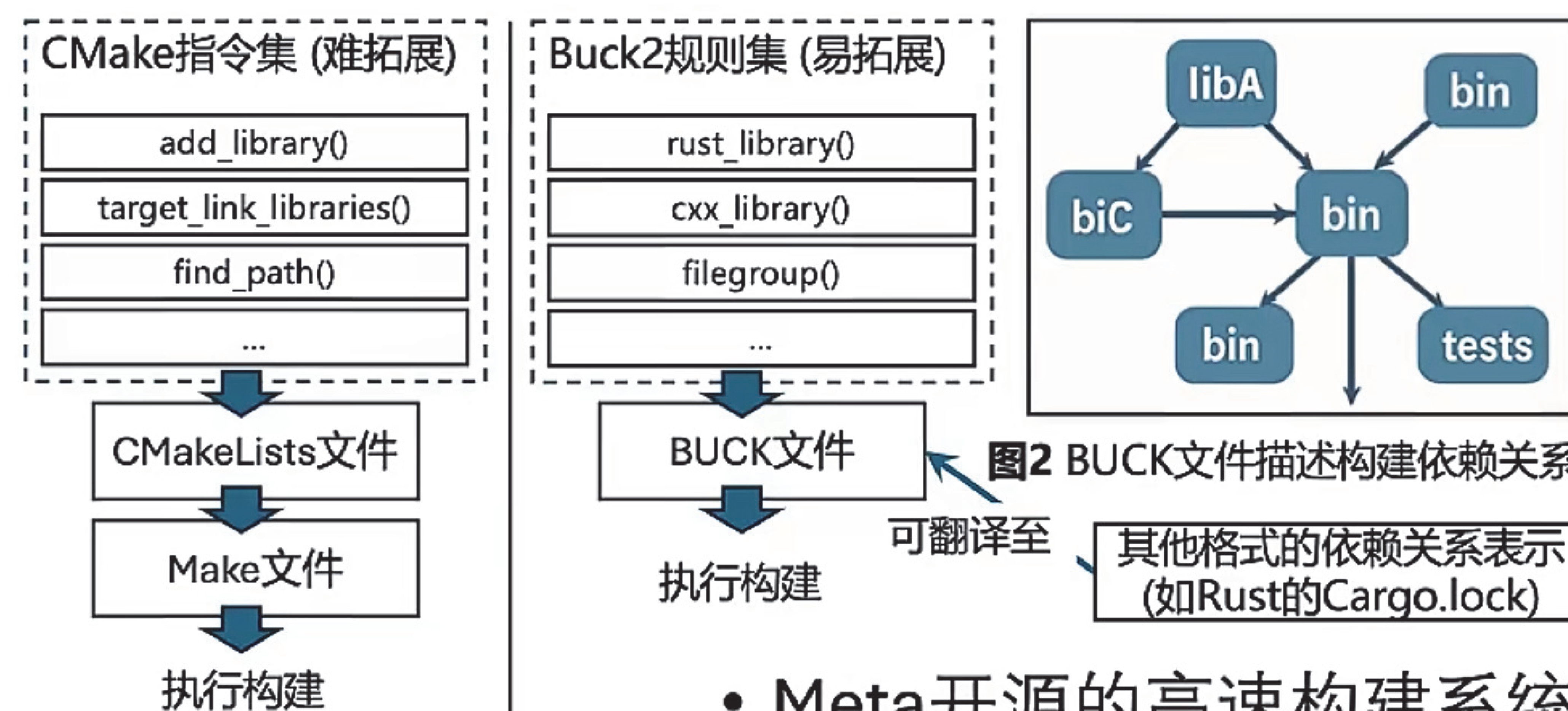
- 案例：rustc bootstrap with Buck2

- Buck2 简介

- 由 Meta 开发的大规模、高性能的构建系统
- 支持多语言、跨平台运行，通过 Starlark* 扩展功能
- BUCK 文件储存项目配置，使用 Starlark* 描述构建过程

*Starlark: 一种 Python 方言

Buck2简介



- Meta开源的高速构建系统
- 自动分析依赖并并行构建
- 支持多语言、分布式构建

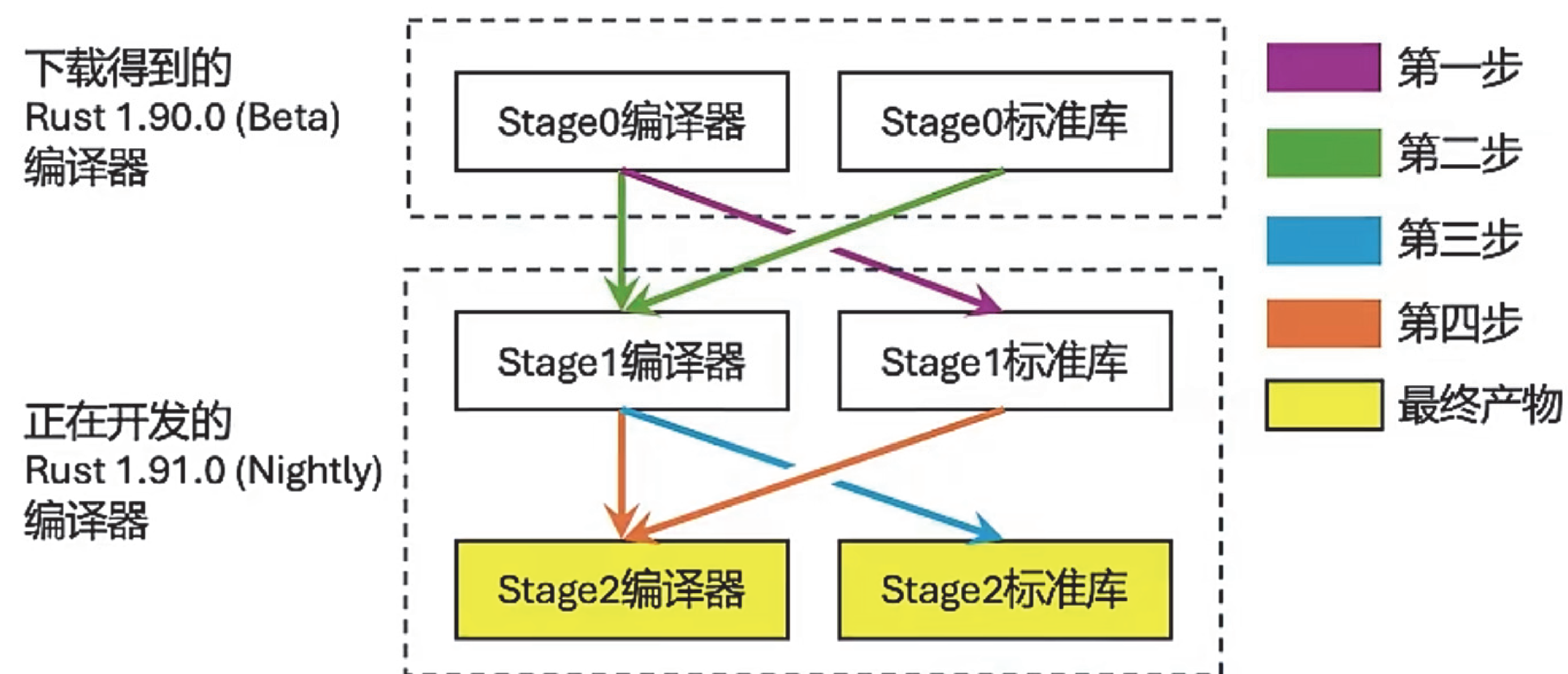


大型构建系统：增量编译、分布式和并行化

Scaled Build Systems: Incremental Compilation, Distribution, and Parallelization

- 案例：rustc bootstrap with Buck2

Rust 编译器是怎样炼成的？



- Rust 编译器由旧版编译器编译产生新版 (多阶段)
- 官方用 x.py 脚本编排 cargo 等指令完成上述流程
- 痛点：手工脚本，易出错、耗时较长

🔥 性能数据* 仅构建编译器

- x.py: 约 8 分钟
- Buck2: 约 6.5 分钟 (提速约 20%)

构建完整工具链

- x.py: 8 分钟 + 153 秒 = 约 10.5 分钟
- Buck2: 6.5 分钟 + 46 秒 = 约 7.3 分钟 (提速约 30%)

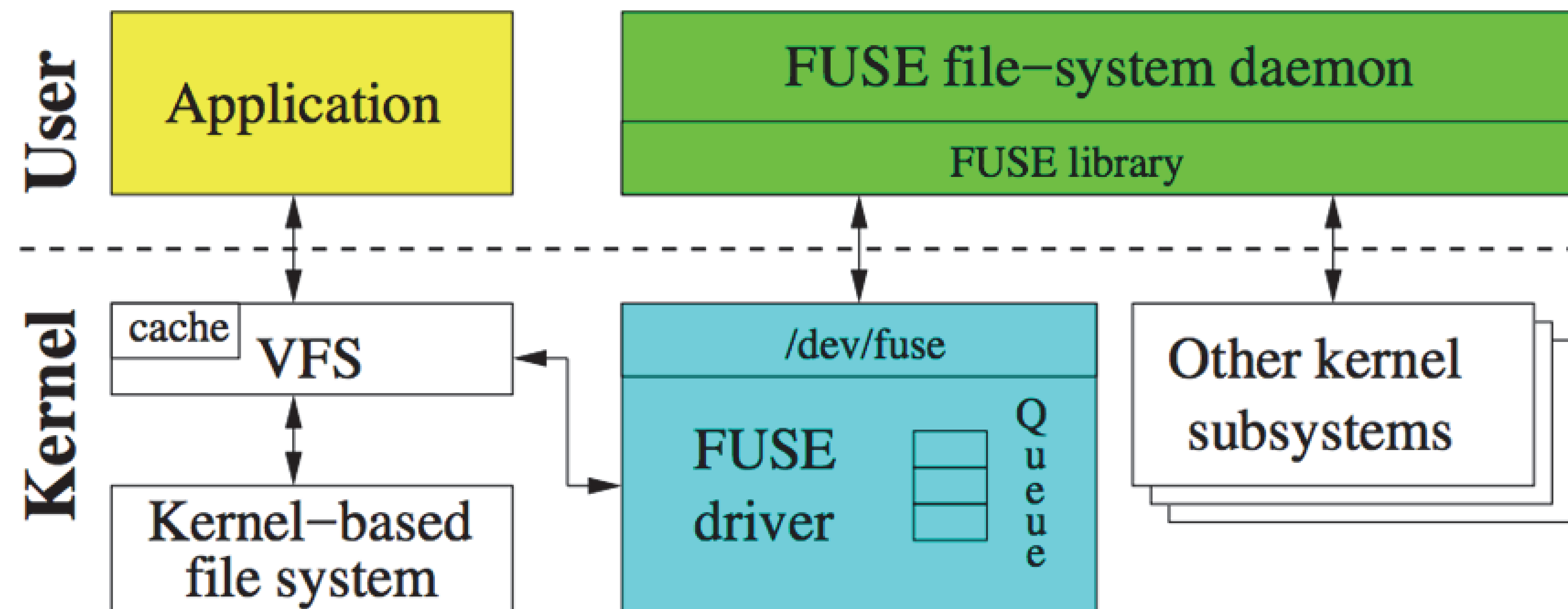
*实验环境：Dell PowerEdge R750xs (Intel Xeon® Gold 6326)



虚拟文件系统

Virtual File System for monorepo

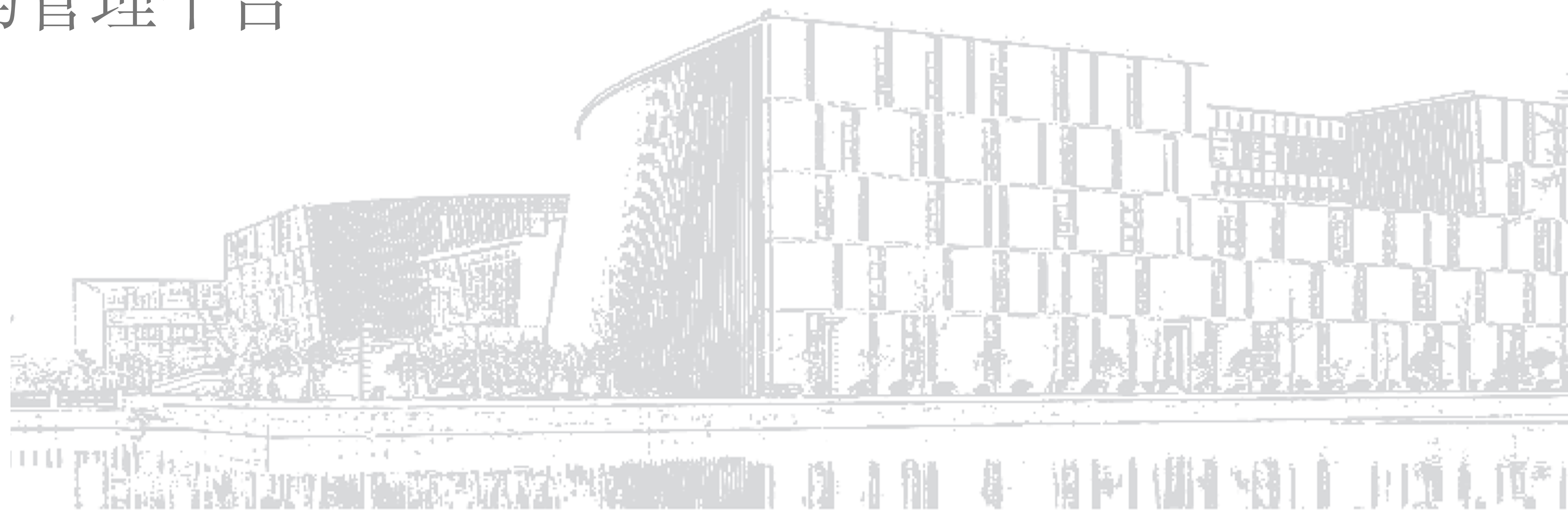
- 大规模仓库克隆/检出速度缓慢，本地存储压力大
- 虚拟文件系统通过按需加载文件加速大型代码仓库中的操作
 - 开发者通常只会使用仓库中的一小部分文件
 - 仅在需要访问时才将工作目录中的文件下载到本地
 - 代价是首次访问新文件时轻微的性能开销





Monorepo 实践

rk8s 容器编排系统 & mega 代码管理平台





rk8s: Rust 云原生项目

rk8s: A Rust Cloud-Native Project

- 使用 Rust 语言重构 kubernetes 的核心组件，打造一个更加安全、高性能、轻量级的云原生编排平台，并且
 - 精简架构：采用模块化设计，精简 Kubernetes 的复杂结构
 - 降低资源占用：无 GC 的特性减少内存占用和延迟抖动
 - 跨平台支持：针对 RISC-V 架构进行适配



项目目标

为云原生生态系统提供一个更加现代化、高性能的 Kubernetes 实现，特别适用于边缘计算、高性能计算和资源敏感的生产环境

<https://github.com/r2cn-dev/rk8s>

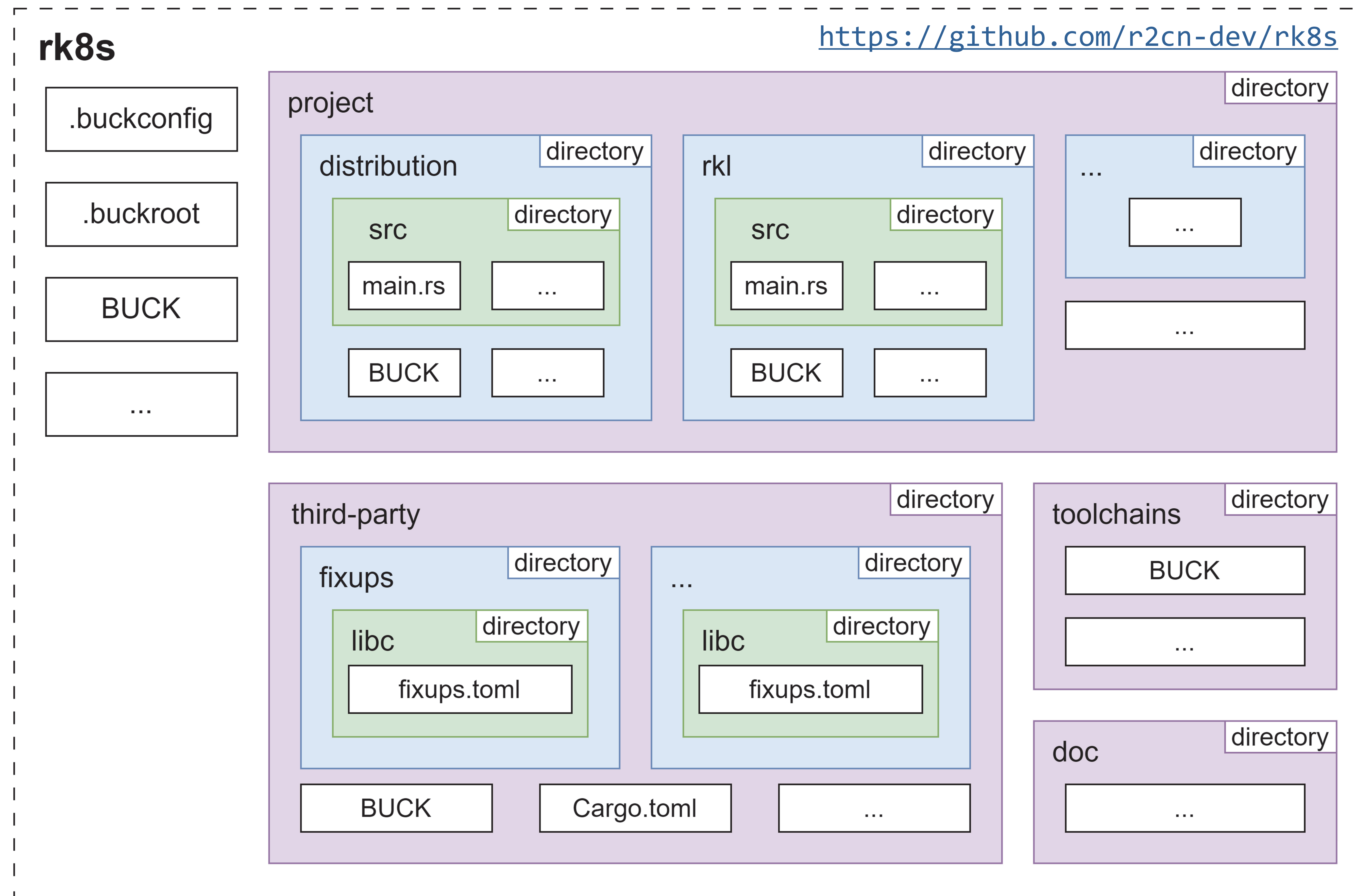




rk8s: Rust 云原生项目

rk8s: A Rust Cloud-Native Project

• 项目结构



```
j1-jiang@Debian:~/rk8s$ tree -L 3
.
├── BUCK
├── docs
│   └── ...
├── project
│   ├── Cargo.lock
│   ├── Cargo.toml
│   ├── distribution
│   ├── libbridge
│   ├── libcgroups
│   ├── libcni
│   ├── libcontainer
│   ├── libfuse-fs
│   ├── libipam
│   ├── libnetwork
│   ├── libscheduler
│   ├── rfuse3
│   ├── rkb
│   ├── rkl
│   ├── rks
│   └── test
│   └── ...
├── README.md
├── third-party
│   ├── rust
│   │   ├── crates
│   │   ├── cpp
│   │   └── packages
│   └── ...
└── ...
```




rk8s: Rust 云原生项目

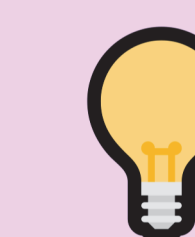
rk8s: A Rust Cloud-Native Project

• 基础设施

- 代码托管: Github
- 构建系统: Cargo/Buck2
- 测试方案: 单元测试 & 集成测试
- 开发模式: Trunk-Based Development (GitHub Flow)

🔍 **main** 📁 origin/buck2 📁 origin/main 📁 origin/workflow Support / in name field via custom dispatch_handler (#116)

- feat: implement memory-mapped file caching with moka and memmap2 (#115)
- test: implement the integration tests for RKL and add it to CI/CD (#113)
- add network for rks (#112)
- definition of node (#111)
- feat: add cache for file handles to improve performance (#101)
- [rfuse3] Fix typo in README (#100)
- fix(rkl/compose): refine the design of rkl's compose-style & fix some issue on v0.1.4 (#99)
- [libfuse-fs] Update libfuse-fs to v0.1.4, rfuse to v0.0.2. (#98)
- fix error in user_delete (#96)
- Docs: update the readme file of the root project and the RKL package (#91)
- feat: Basic scheduler framework (#92)



Squash and merge

提交信息 Support `/` in `name` field via custom dispatch_handler (#116)

* feat(registry): allow '/' in {name} via custom dispatch_handler

Signed-off-by: jl.jiang <jiangjl9807@gmail.com>

* docs(registry): update README

Signed-off-by: jl.jiang <jiangjl9807@gmail.com>

* fix(registry): configure URL via CLI instead of environment variable

Signed-off-by: jl.jiang <jiangjl9807@gmail.com>



rk8s: Rust 云原生项目

rk8s: A Rust Cloud-Native Project

- 构建配置
 - 显示定义依赖版本/支持多版本共存

```
# Cargo.toml
[package]
name = "distribution"
version = "0.1.0"
edition = "2024"
[dependencies]
async-trait = "0.1.86"
axum = "0.8.1"
dotenv = "0.15.0"
futures = "0.3.31"
hex = "0.4.3"
oci-spec = "0.7.1"
regex = "1.11.1"
serde = "1.0.217"
serde_json = "1.0.138"
sha2 = "0.10.8"
tokio = { version = "1.43.0", features = ["full"] }
tokio-util = { version = "0.7.13", features = ["io"] }
uuid = { version = "1.13.1", features = ["v4", "fast-rng", "macro-diagnostics"] }
```

```
# BUCK
rust_binary(
  name = "distribution",
  srcs = glob(
    ["src/**/*.rs"],
  ),
  crate_root = "src/main.rs",
  deps = [
    "//third-party:async-trait",
    "//third-party:axum",
    "//third-party:dotenv",
    "//third-party:futures",
    "//third-party:hex",
    "//third-party:oci-spec",
    "//third-party:regex",
    "//third-party:serde",
    "//third-party:serde_json",
    "//third-party:sha2",
    "//third-party:tokio",
    "//third-party:tokio-util",
    "//third-party:uuid",
  ],
)
```




Mega: 兼容 Git 的 Monorepo 引擎

Mega: Monorepo Engine with Git Compatible

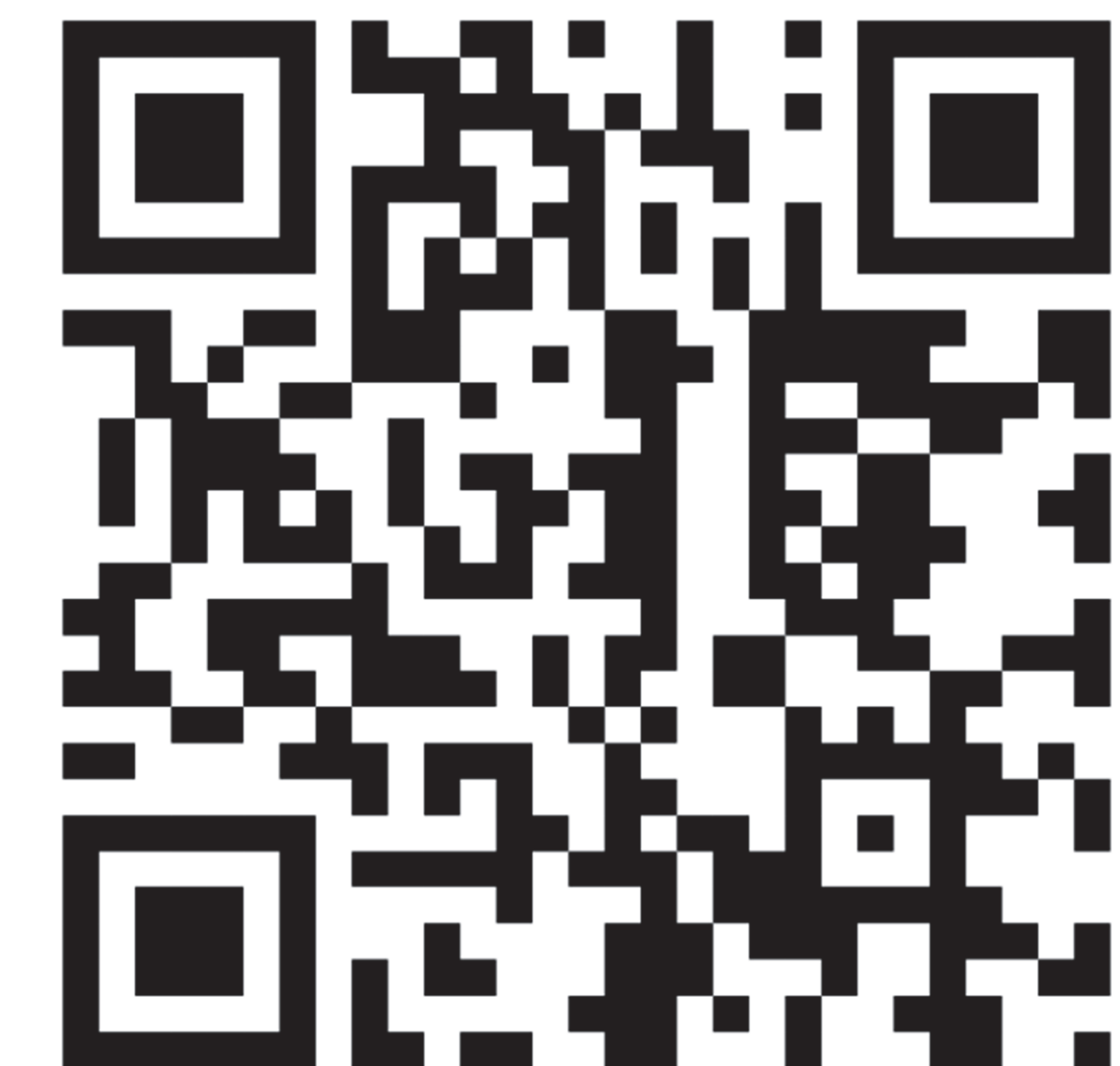
- Google Piper 非官方开源实现
 - 100% Git 兼容的读写路径与迁移能力
 - 集成 Buck2 作为平台默认的大型构建系统
 - 对“单仓/单体”场景的高效目录/对象管理
 - 面向大型组织的策略化权限与审计



项目目标

在兼容 Git 的前提下，提供“单仓管理 & 单体代码库治理 & 去中心化协作”的一体化解决方案

<https://github.com/web3infra-foundation/mega>





R2CN 开源实习计划

RISC-V Rust for Cloud Native

- 培养熟悉 RISC-V 架构和 Rust 编程语言的高校开发者
- 构建 RISC-V 架构下的云原生技术栈

[开源项目加入 R2CN](#)

RISC-V Rust for Cloud Native 开源实习计划

R2CN（RISC-V Rust for Cloud Native）开源实习计划由中国科学院软件研究所主办，目标是培养熟悉 RISC-V 架构和 Rust 编程语言的高校开发者，构建 RISC-V 架构下的云原生技术栈。

1. 完成测试任务

2. 注册实习计划

3. 查看帮助文档

<https://r2cn.dev/>





感谢观看！

