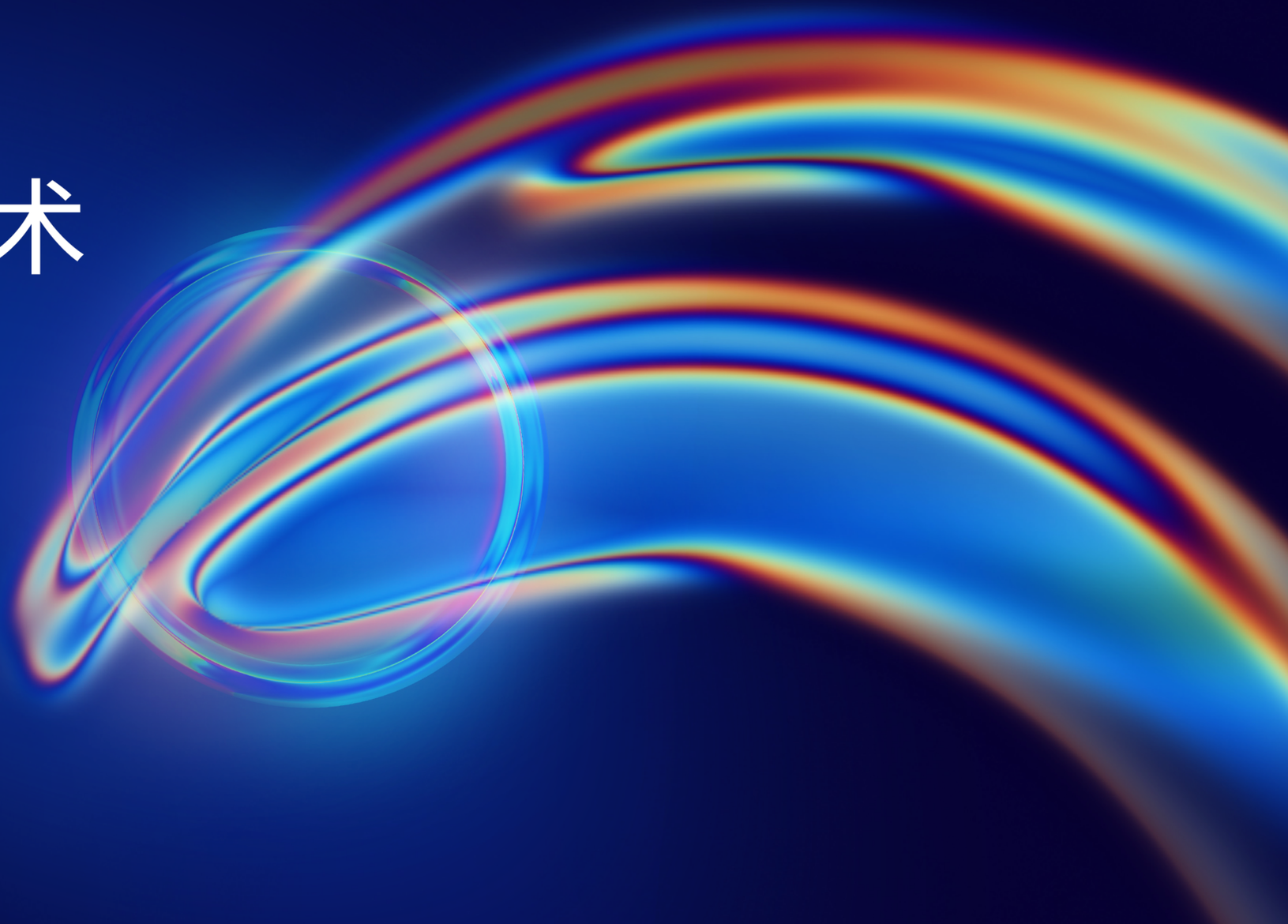
WOT | 51CTO

World Of Tech 2023

# WOT全球技术创新大会

2023

新技术 瞰未来



WOT | 51CTO

# 字节跳动的前端工程化实践

林宜丙 前端工程师

字节跳动-Web Infra-前端架构工程师，拥有多年前端工程化经验，致力于帮助前端工程师更好地管理和治理工程，目前负责工程治理方向的方案设计及其业务落地工作。

WOT | 51CTO

·PART ONE

前端趋势及其新挑战

·PART TWO

应对挑战的实践

·PART THREE

整体落地情况

·PART FOUR

总结与展望

# 什么是前端工程化?

**开发效率**
CLI/Bundler

**代码质量**
ESLint/Prettier

**代码复用性**
Modular Dev

**自动化流程**
CI/CD Pipeline

**团队协作**
Monorepo/Micro Fronend

**...**

# 前端趋势-工种趋势

支撑的业务

功能愈发复杂：VsCode, Figma, Doc

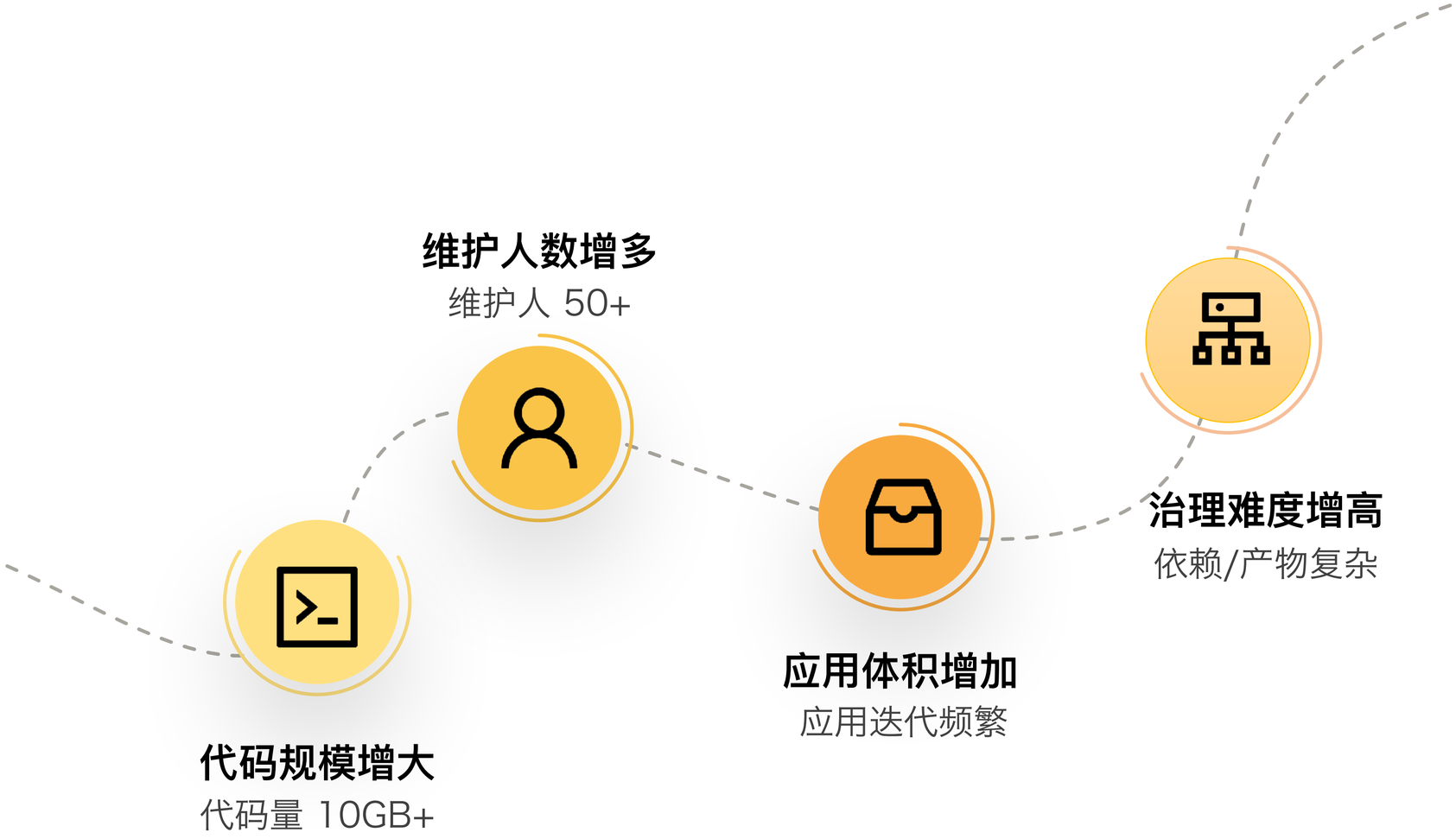
涉及的平台

技术栈变多样：Web, Node, Native

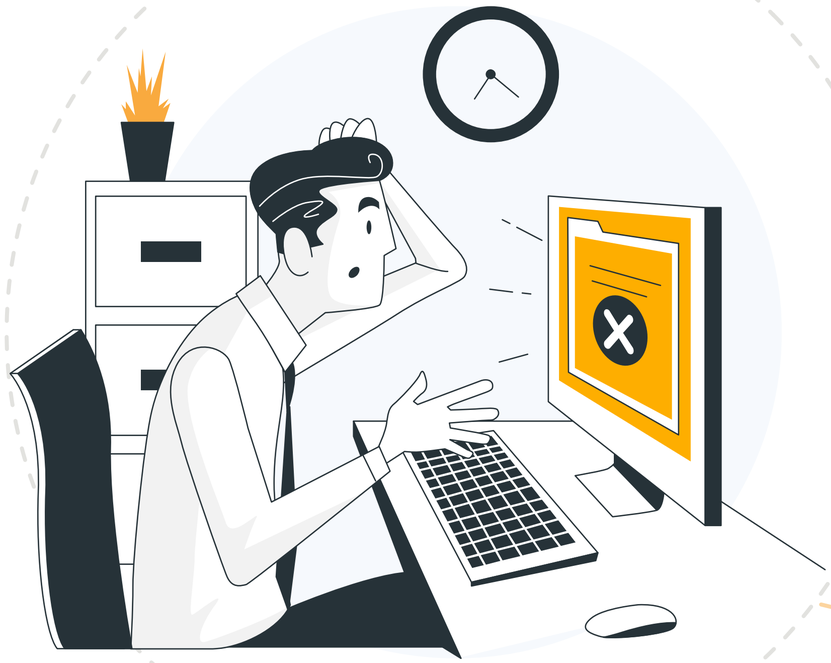
团队的规模

参与开发人数变多：10+, 30+, 50+

# 前端趋势-工程趋势



# 面临的新挑战



**多项目维护成本高**

项目基建重复、代码复用困难、工作流程割裂

**多人开发协作成本高**

相互依赖的流程、级联的依赖升级

**巨型应用构建速度慢**

构建耗时随着应用增大而变慢

**大型应用劣化速度快**

缺乏有效的防劣化手段

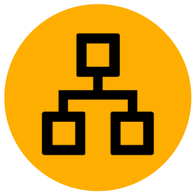
# 应对挑战的实践

**Monorepo**

降低多项目的维护成本

**Micro Frontend**

降低多人开发的协作成本



**Bundler & Build System**

加快巨型应用的构建速度

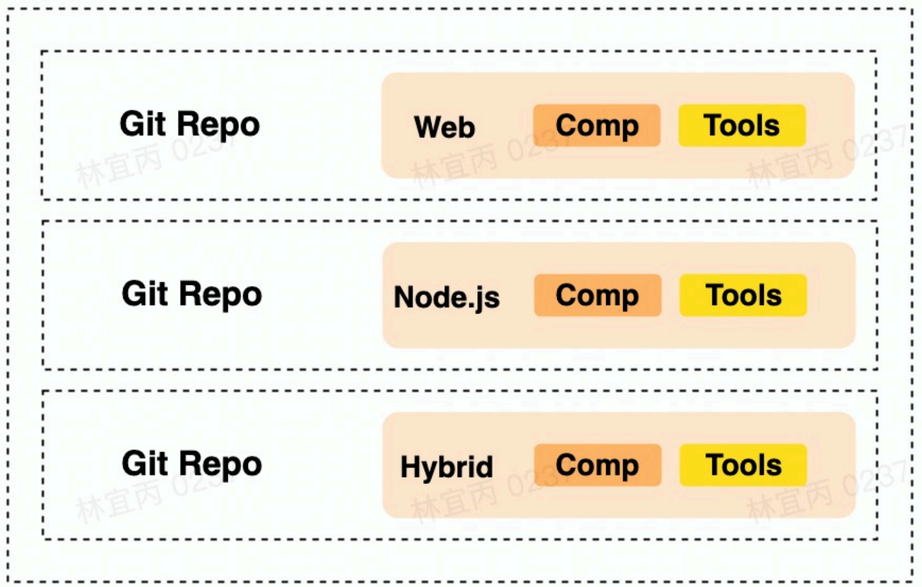
**Diagnostics Tool**

有效地防止应用劣化

# Monorepo 简介

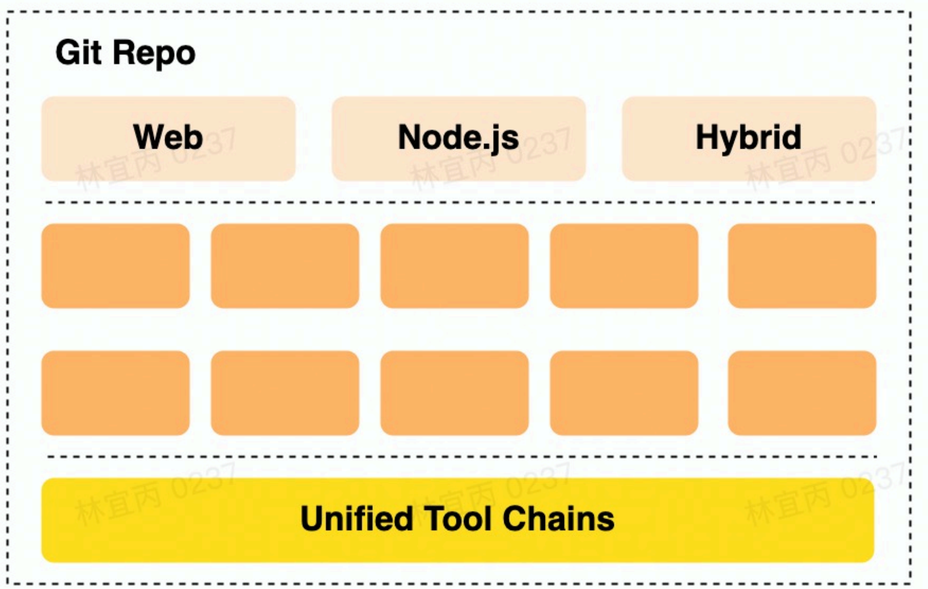
## Polyrepo

子项目分布到不同的仓库中管理




## Monorepo

子项目组织到一个仓库中统一管理




将多个项目组织良好地维护于单个仓库中

# 如何降低多项目的维护成本?

**复用基建**

让开发人员重新专注于应用本身

**代码共享**

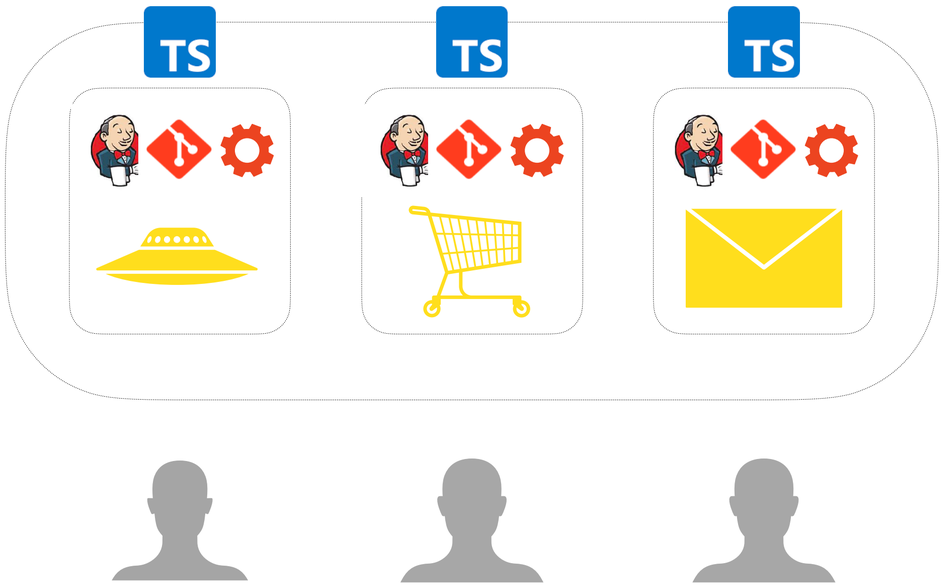
降低代码复用的成本

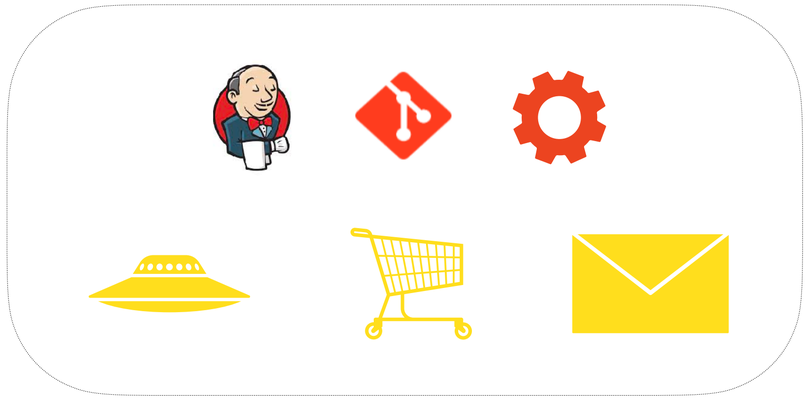
**原子提交**

使用自动化的多项目工作流

# Monorepo - 复用基建

Polyrepo VS Monorepo



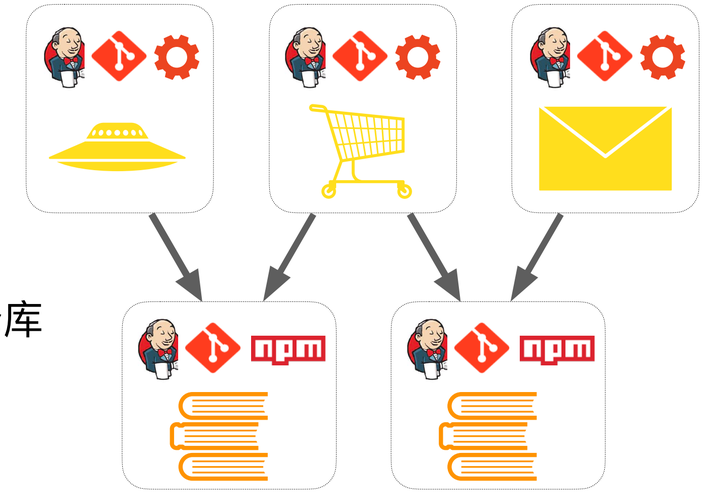
- 每个项目都需要有开发人员创建维护
- 基建重复，调整多个项目基建成本高

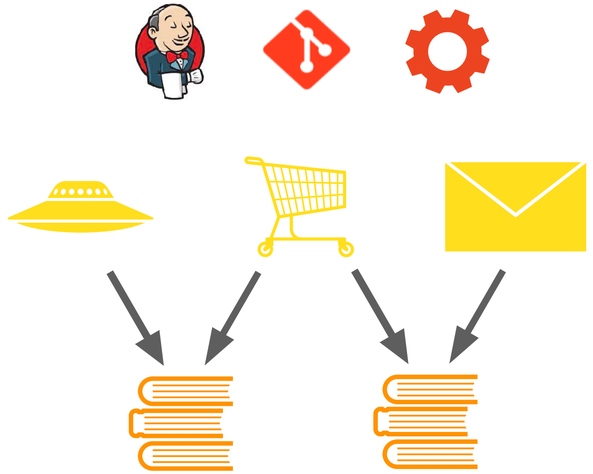
- 1-2 个开发人员负责所有项目的公共架构维护
- 基建复用，调整多个项目的基建成本低

# Monorepo - 代码共享

Polyrepo VS Monorepo

为每个模块设置单独的仓库



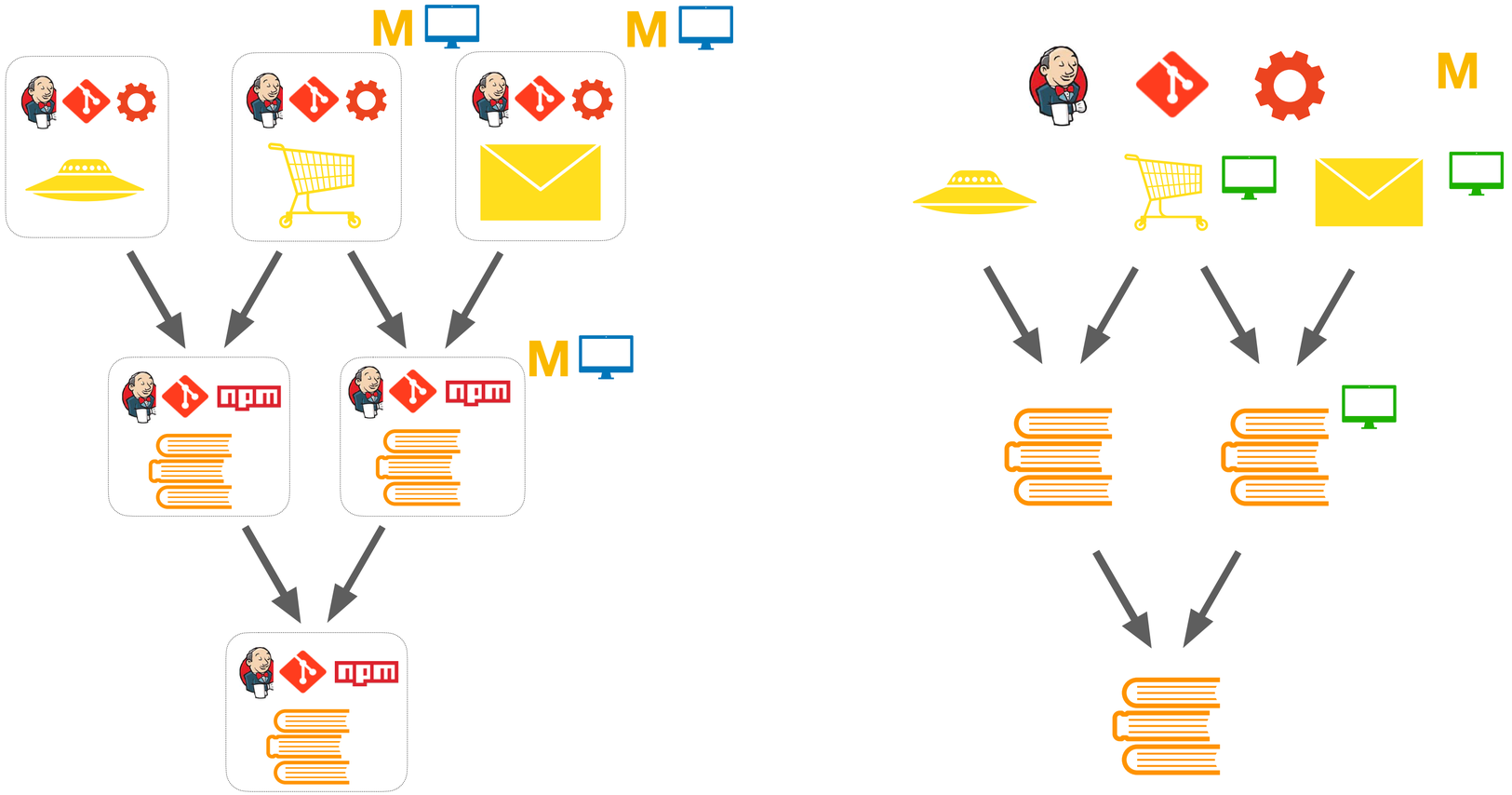
一键生成公用模块

- 公用模块的维护成本高，调试、升级繁琐
- 代码复用率低

- 便捷的代码共享，一键引入、调试公用模块
- 代码复用率高

# Monorepo - 原子提交

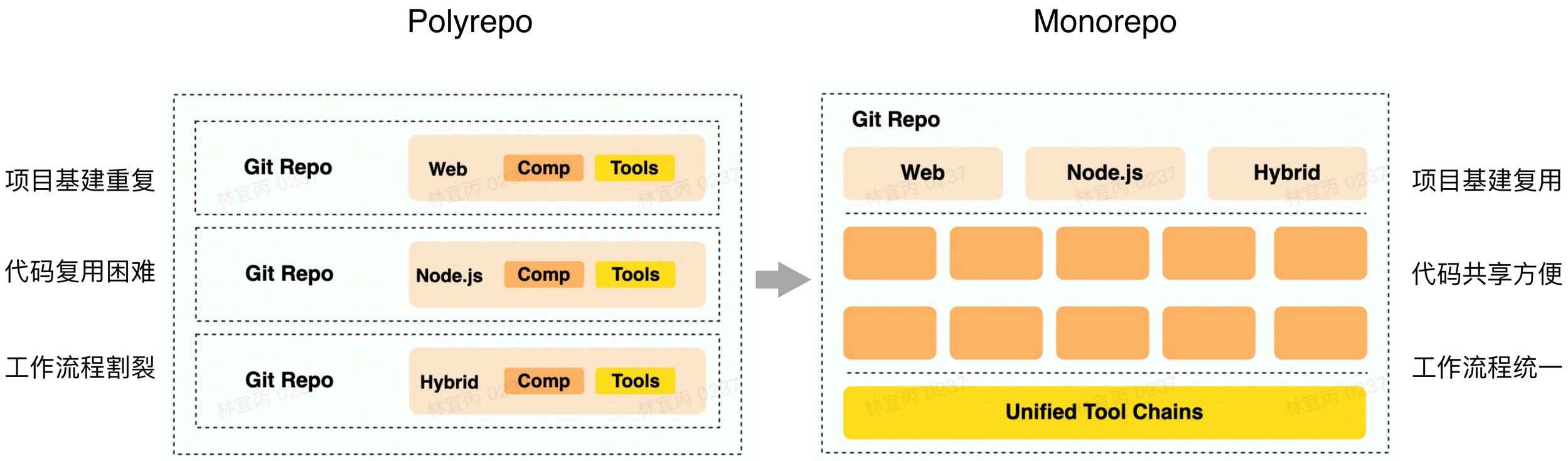
Polyrepo VS Monorepo




- 业务需求可能涉及多个项目
- 多项目工作流不连续

- 一次性调整并提交多个项目
- 自动、连续的 CI 流程

# Monorepo - 总结




通过建设 Monorepo 极大地降低了多项目的维护成本

# 如何加快巨型应用的构建速度?

**Bundler**

加快巨石应用的构建速度

**Build System**

加快 Monorepo 的构建速度

# Bundler 简介

处理文件之间的模块依赖关系图，并将其打包成静态资源

Webpack

Rollup

Vite

Parcel

Esbuild

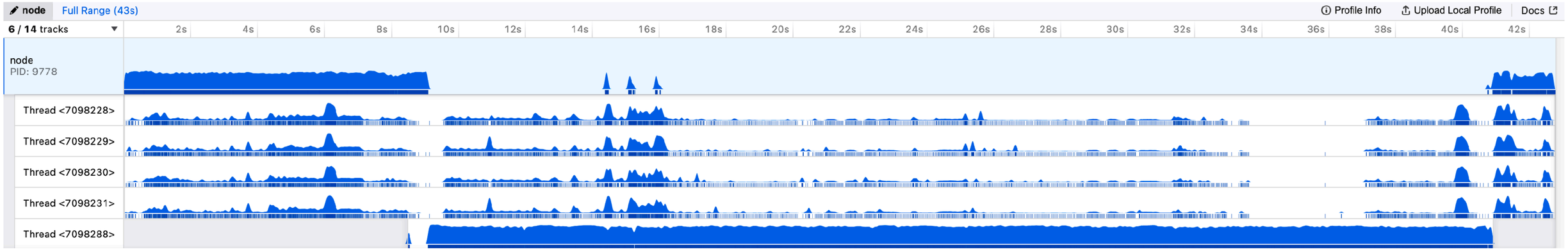
# Bundler 实践

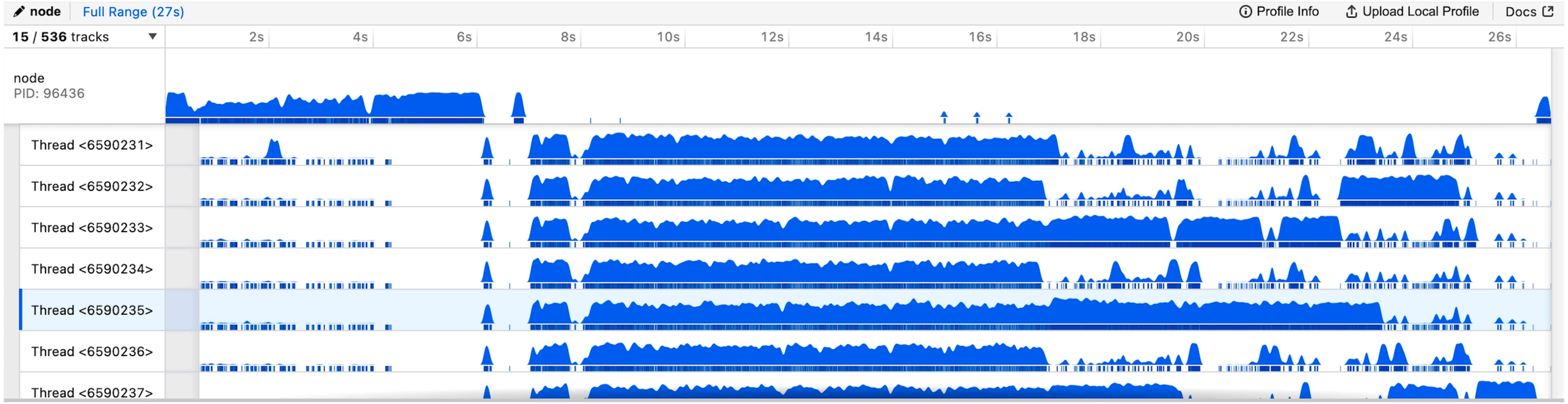
Rspack 是一个基于 Rust 的高性能构建引擎（Bundler），具备与 Webpack 生态系统的互操作性。

https://www.rspack.dev

# Bundler - Rust 语言实现

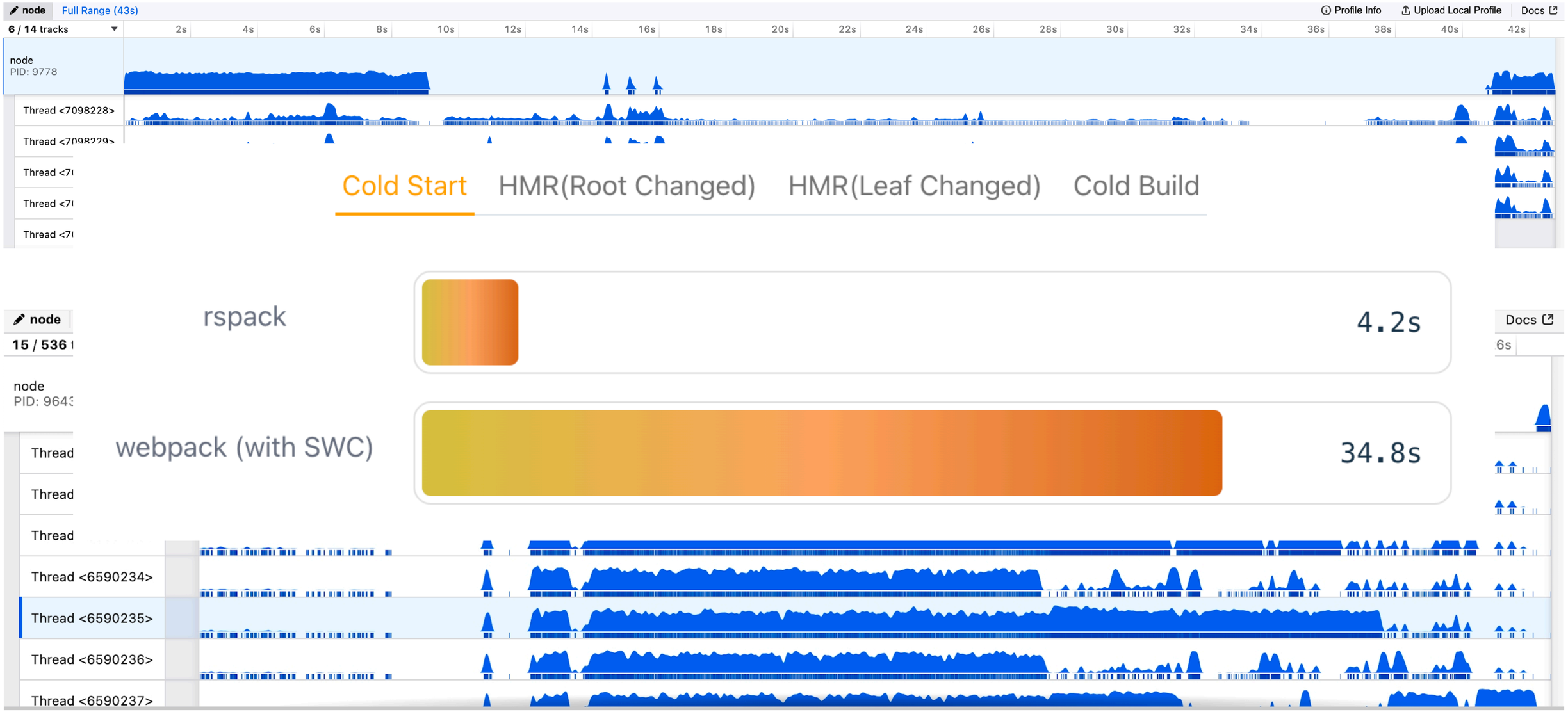
🦀 Rust 实现核心部分，充分利用并发特性



# Bundler - Rust 语言实现

Rust 实现核心部分，充分利用并发特性



# Bundler - Webpack 生态兼容

## 📦 与 Webpack 部分兼容

The subset of Webpack!

```ts
/** @type {import('@rspack/cli').Configuration} */
const config = {
  context: __dirname,
  entry: { main: "./src/index.tsx" },
  devServer: { port: 5555, webSocketServer: "sockjs", historyApiFallback: true },
  mode: prod ? "production" : "development",
  devtool: prod ? false : "source-map",
  cache: false,
  module: {…
  },
  resolve: { alias: { "@": path.resolve(__dirname, "src") } },
  output: { publicPath: "/", filename: "[name].[contenthash].js" },
  optimization: {
    realContentHash: true,
    splitChunks: { cacheGroups: { someVendor: { chunks: "all", minChunks: 2 } } }
  },
  plugins: [new HtmlPlugin({ title: "Arco Pro App" })],
  infrastructureLogging: { debug: false },
  builtins: { progress: {}, treeShaking: true, sideEffects: true, noEmitAssets: false }
};
```

- babel-loader
- style-loader
- css-loader
- postcss-loader
- sass-loader
- less-loader
- raw-loader
- file-loader
- url-loader
- svelte-loader
- @mdx-js/loader
- @svgr/webpack
- image-webpack-loader
- thread-loader
- source-map-loader
- node-loader
- vue-loader
- ……

- html-webpack-plugin => builtins.html
- react-refresh-webpack-plugin => builtins.react.refresh
- webpack.DefinePlugin => builtins.define
- webpack.ProvidePlugin => builtins.provide
- mini-css-extract-plugin => experiments.css
- tsconfig-paths-webpack-plugin => resolve.tsconfigPath
- copy-webpack-plugin => builtins.copy
- webpack-bundle-analyzer
- webpack-stats-plugin
- ……

https://www.rspack.dev

# Bundler - 提升效果

| | Webpack | Rspack | 提升 |
|---|---|---|---|
| Build | 380s | 28s | 13 倍 |
| Dev | 368s | 25s | 14 倍 |
| HMR | 21s | 1s | 21 倍 |

接入业务1

| | Webpack | Rspack | 提升 |
|---|---|---|---|
| Build | 508s | 59s | 9 倍 |
| Dev | 243s | 24s | 10 倍 |
| HMR | 18s | 1.4s | 13 倍 |

接入业务2

https://www.rspack.dev

# Build System 简介

处理 Monorepo 下的项目依赖关系图，并据此调度构建任务

Bazel

NX

Turborepo

Lage

# Build System 实践

## 任务并行能力

采用最大限度的并行任务加速

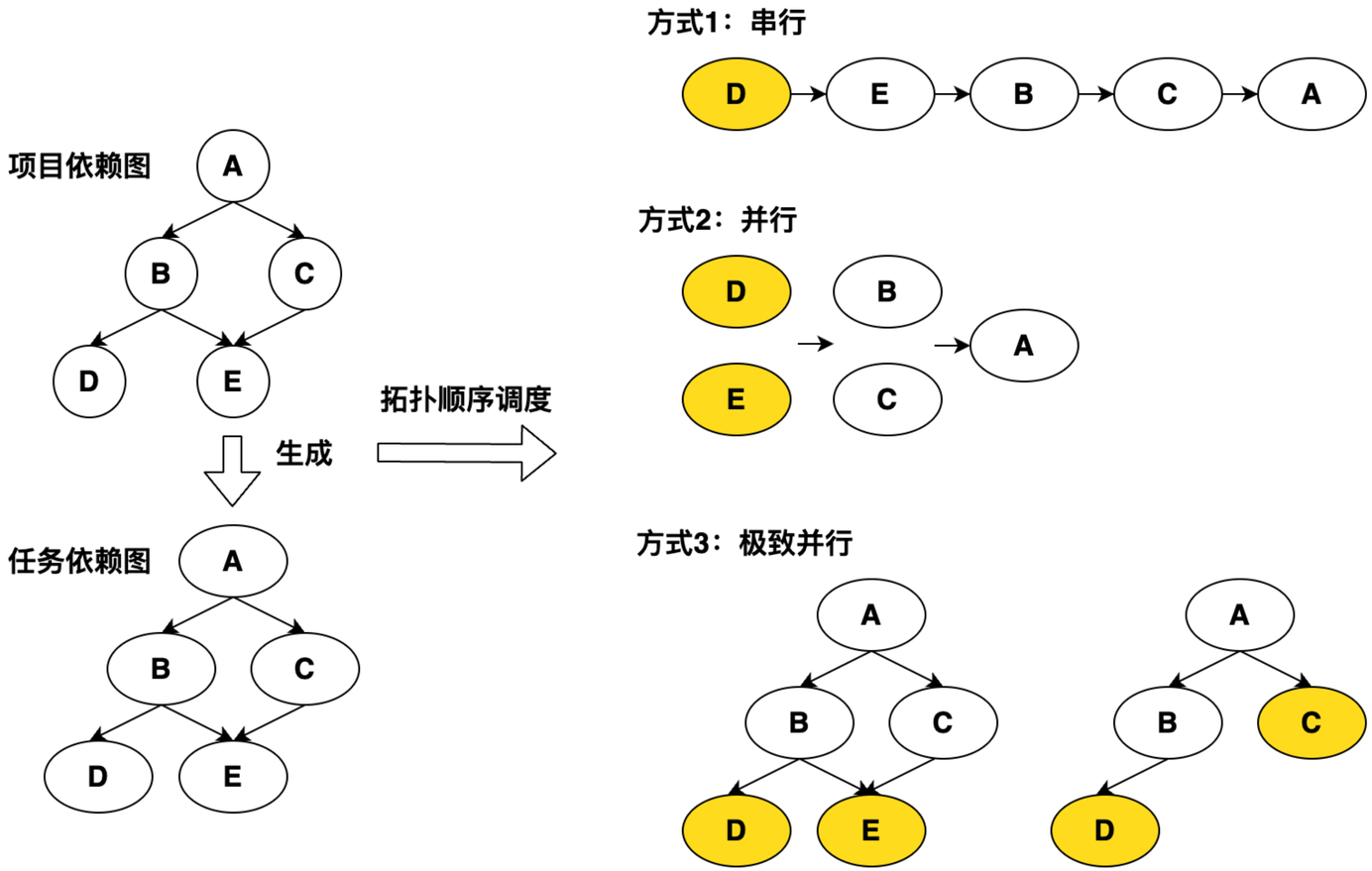
## 多级缓存能力

对构建实现了多级缓存

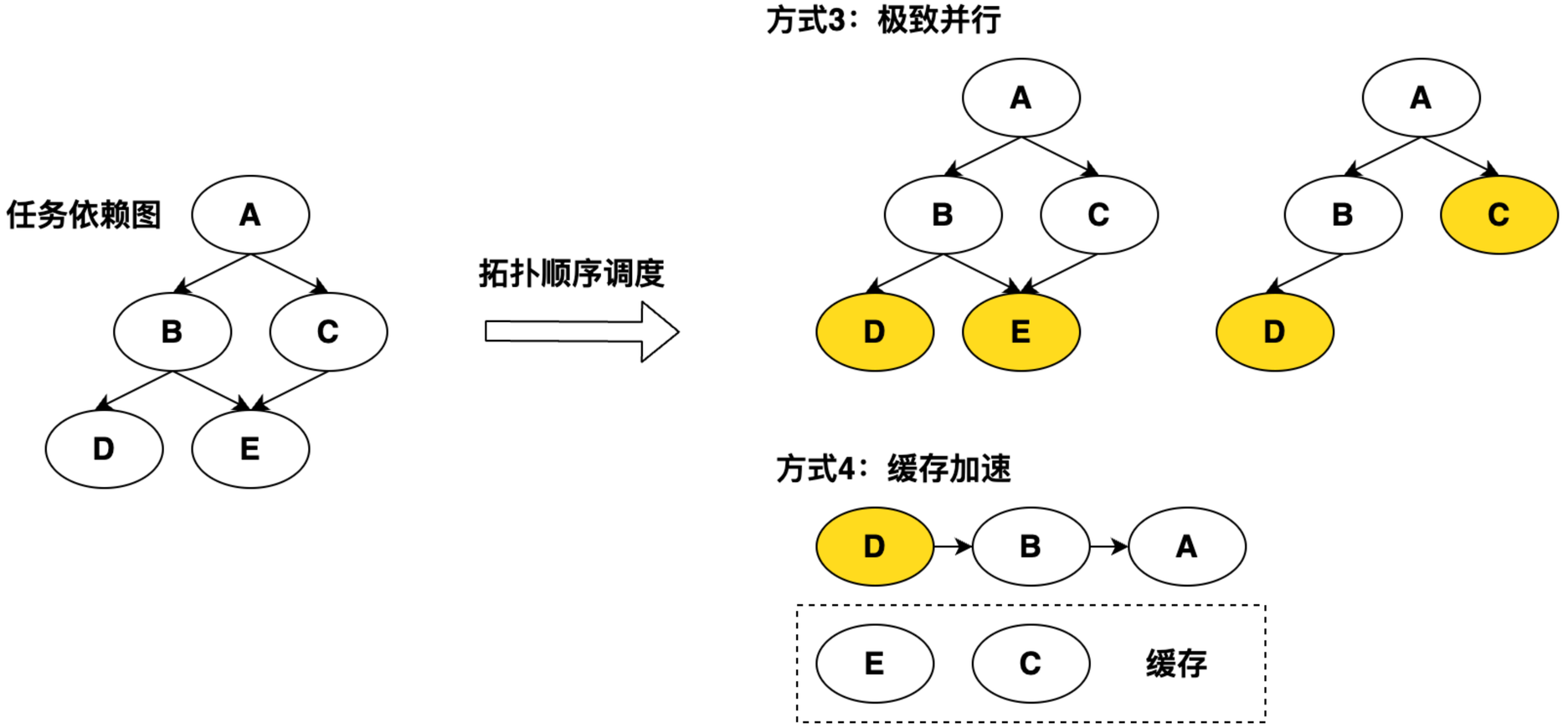
## 按需构建能力

根据代码更改的影响面来构建

# Build System - 任务并行能力

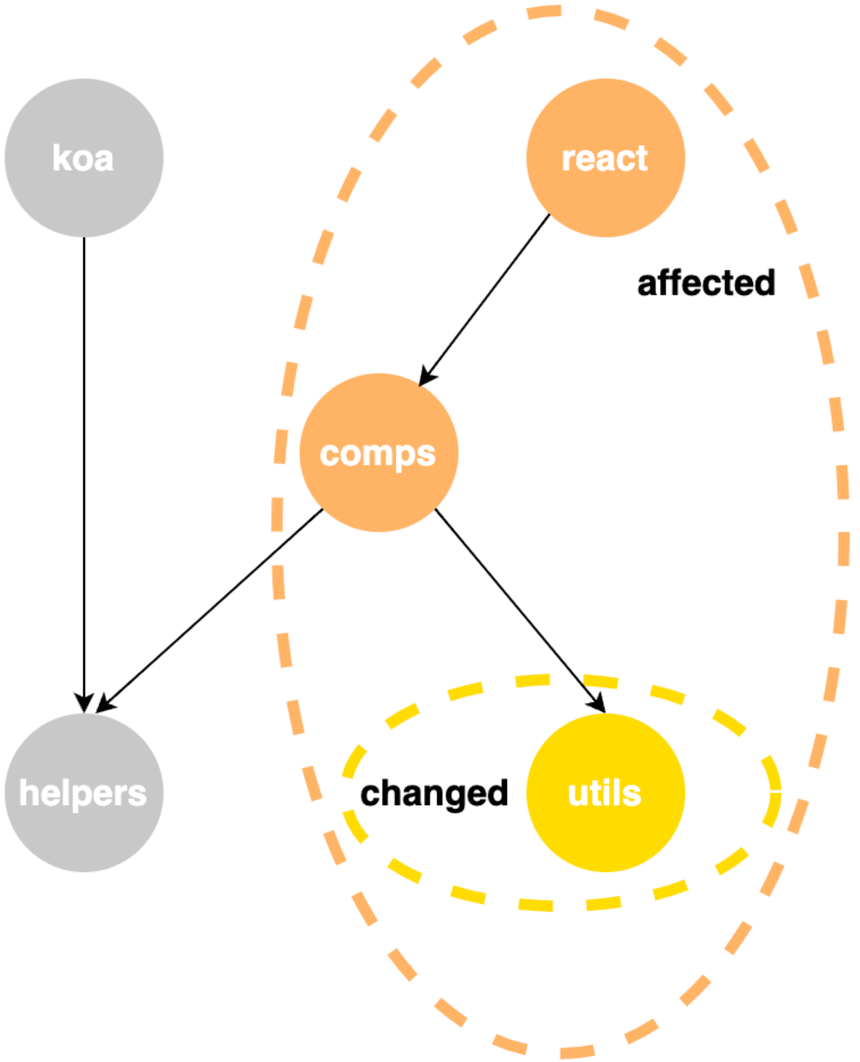


# Build System - 多级缓存能力

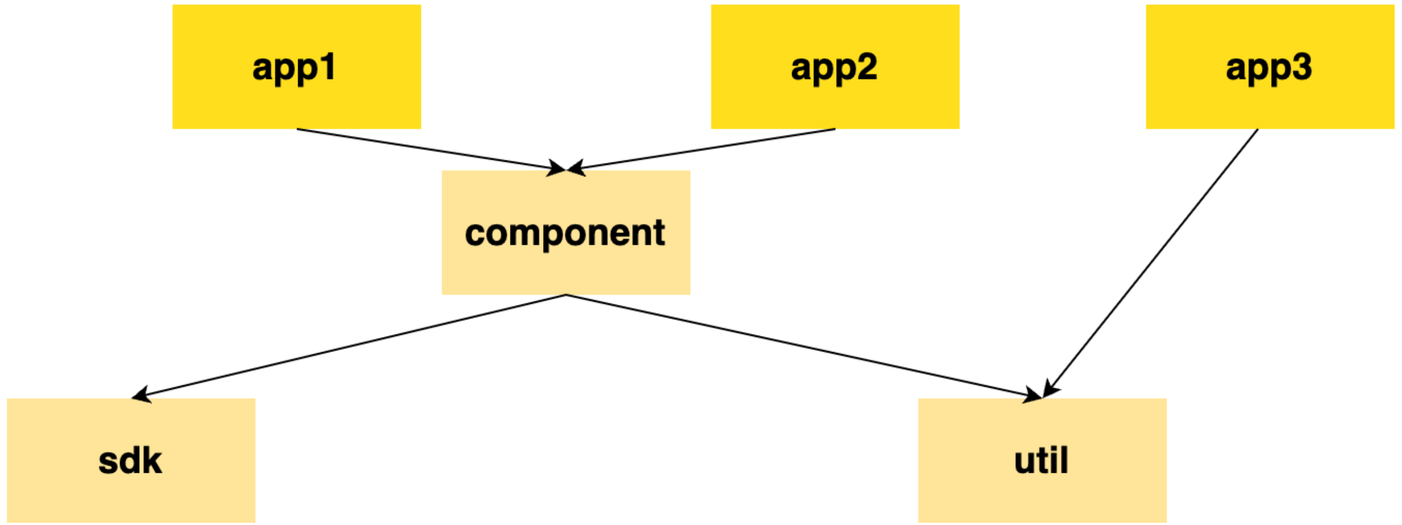


# Build System - 按需构建能力

Diff 代码



# Build System - 提升效果




| | App1 | App2 | App3 |
|---|---|---|---|
| 全量 | 17.72s | | |
| 按需 | 8.94s | | |
| 无缓存 | 16.94s | 16.13s | 10.77s |
| 有缓存 | 9.74s | 8.94s | 7.55s |

# Bundler & Build System - 总结

拉高应用上限

加快迭代速度

https://www.rspack.dev

通过建设 Bundler 和 Build System 极大地加速了巨型应用的构建速度

# Micro Frontend 简介

前端领域的应用分治解决方案

Qiankun

Garfish


Bit

# 如何降低多人开发的协作成本?

**减轻基座负担**

基座应用与业务解耦

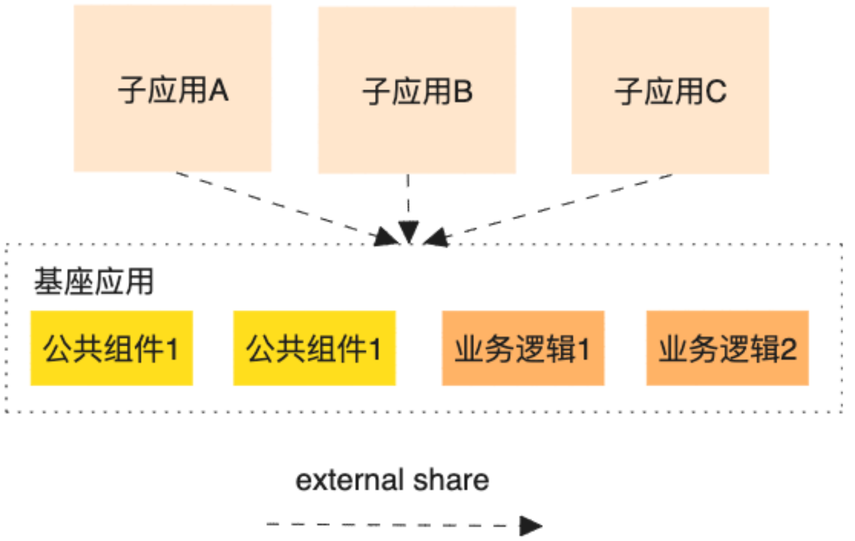
**细粒度的组合**

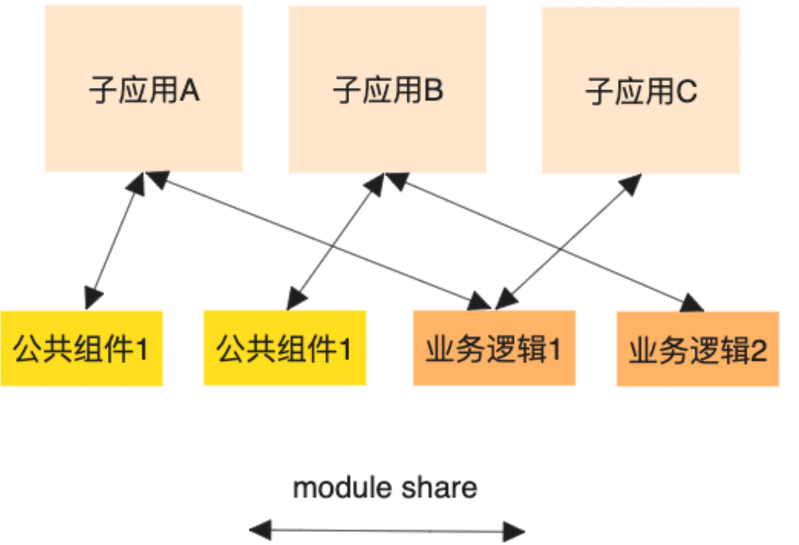
模块级别的独立开发、测试、部署

**模块协议标准**

模块中心、结合低码、灰度/AB

# Micro Frontend - 减轻基座负担



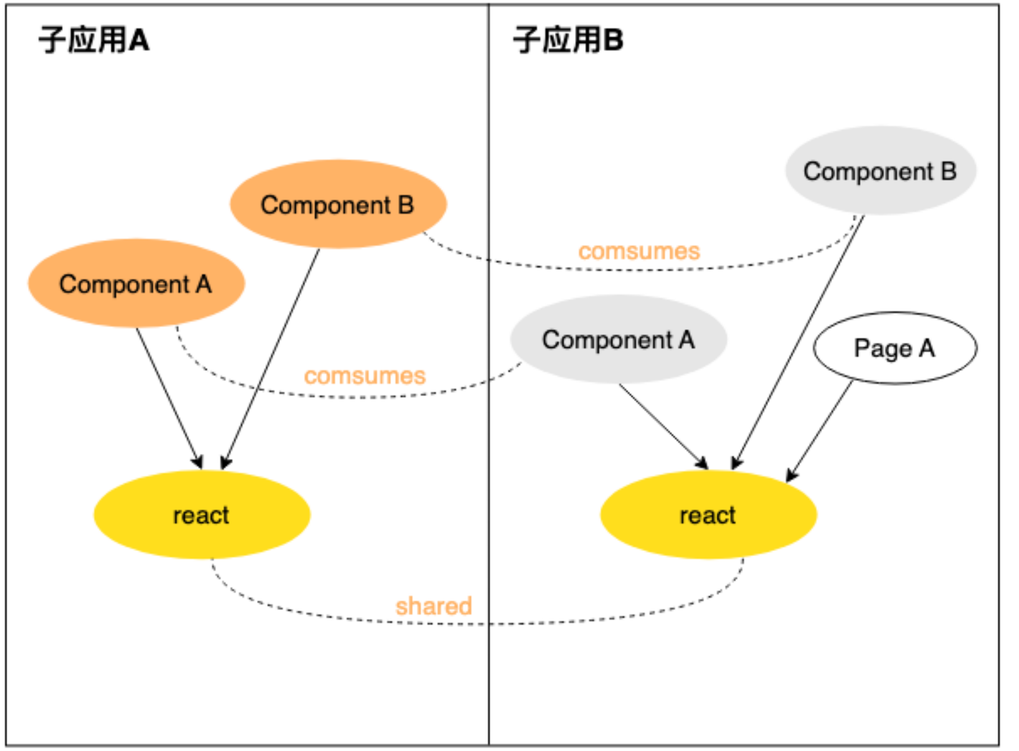




- 业务逻辑耦合到基座
- 故障影响面、失效的缓存
- 造成相互依赖

- 消除公共基座
- 业务逻辑通过消费机制
- 工具库通过共享机制

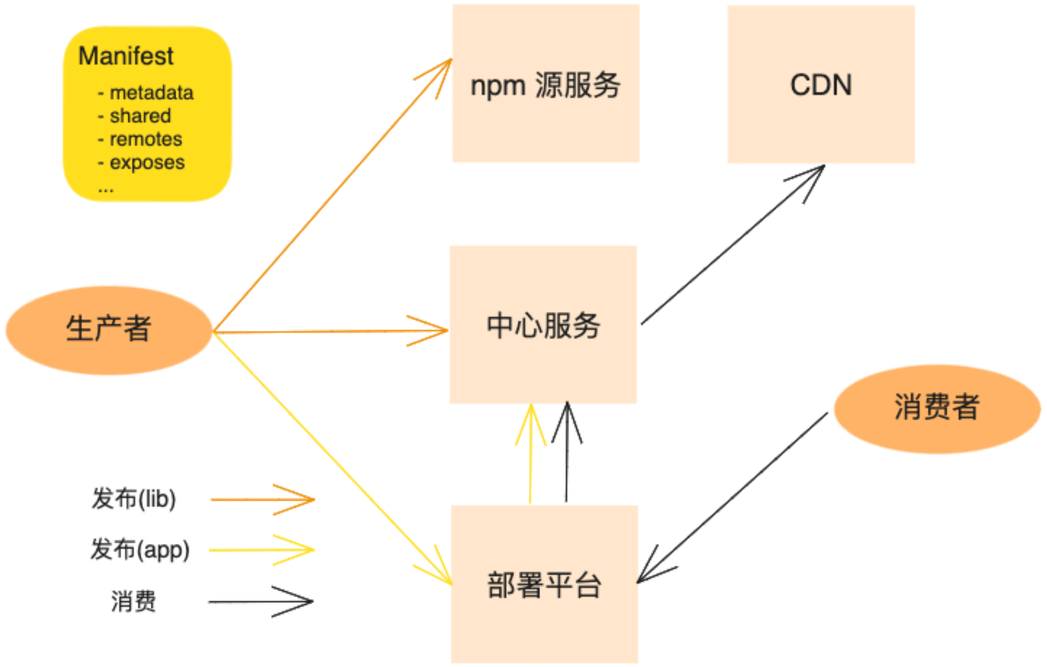
# Micro Frontend - 细粒度的组合

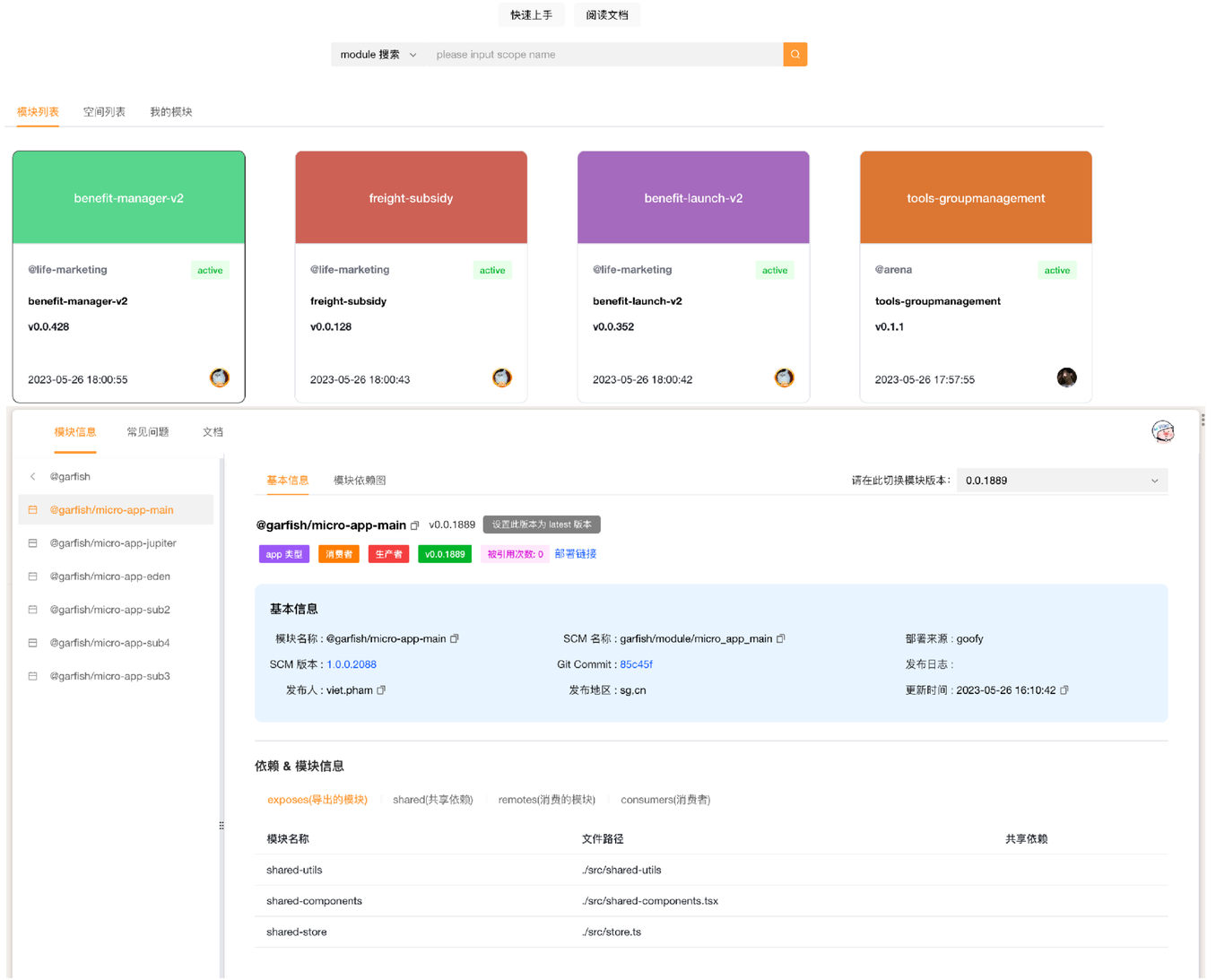

- 沙箱机制保证隔离性
- 制约子应用内外的人员协作




- 提供更细粒度的模块
- 跨子应用消费和共享

# Micro Frontend - 模块协议标准




- 流转于构建平台、部署平台、运行时

# Micro Frontend - 总结

| | SPA 架构 | 微前端架构 | 提升效果 |
|---|---|---|---|
| 构建耗时 | 17m43s | 8m | 50% |
| 部署耗时 | 10m22s | 2m | 80% |
| 月均发布 | 30.6次 | 77.6次 | 264% |

通过建设 Micro Frontend 降低了多人开发的协作成本

# Diagnostics Tool 简介

🩺 面向构建过程与构建产物的诊断与分析的一站式工具

Bundle Analyzer

Statoscope

Relative CI

# 如何有效地防止应用劣化?

**面向构建过程**

更细粒度和更丰富的分析能力

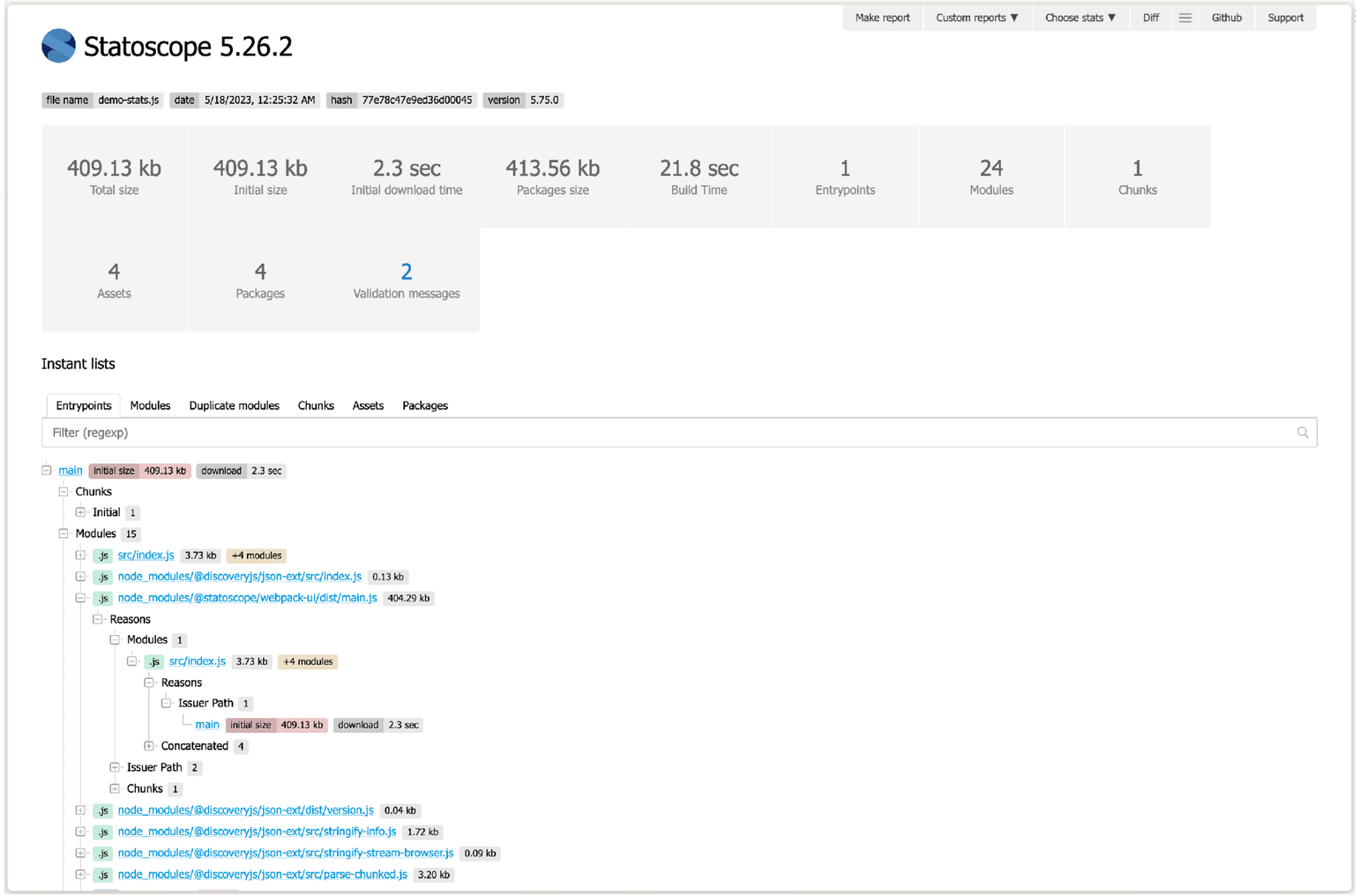
**可扩展的规则机制**

丰富的上下文，强大的扩展能力

**与核心研发流程结合**

在 CI 中让规则真正发生作用

# Diagnostics Tool - 面向构建过程

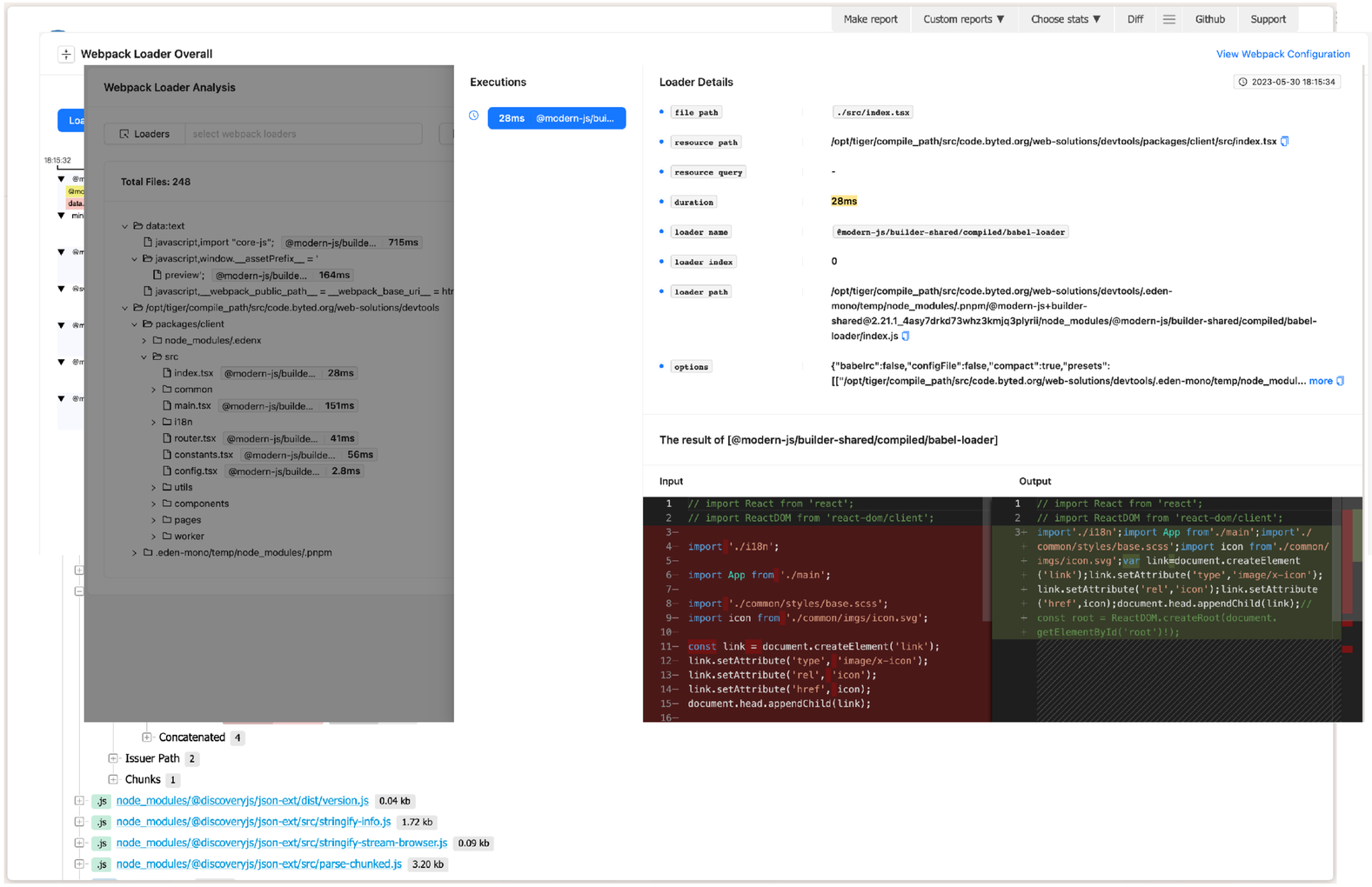



构建产物（stats.json）　　VS　　构建过程

# Diagnostics Tool - 面向构建过程

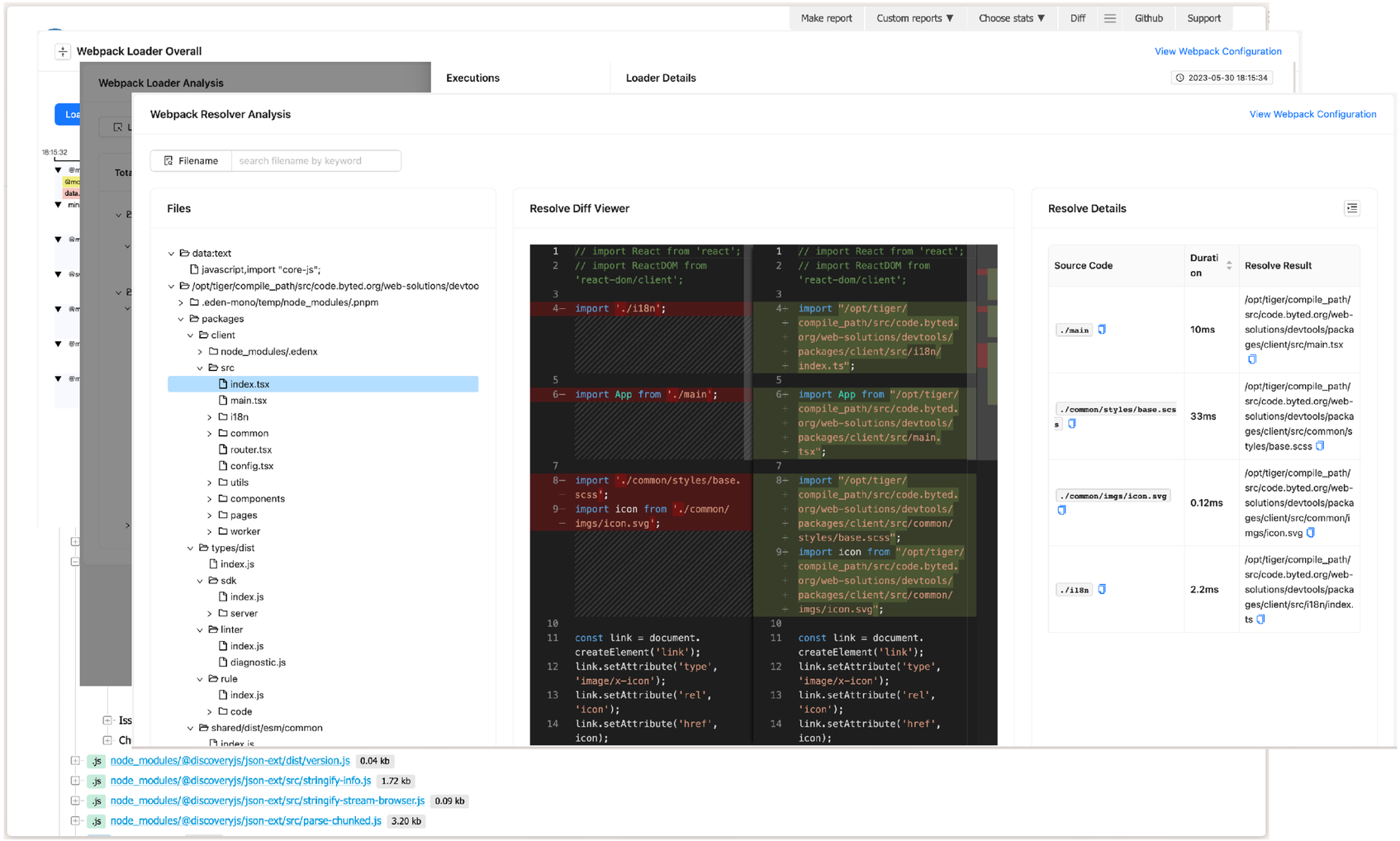

构建产物（stats.json）　　VS　　构建过程

# Diagnostics Tool - 面向构建过程



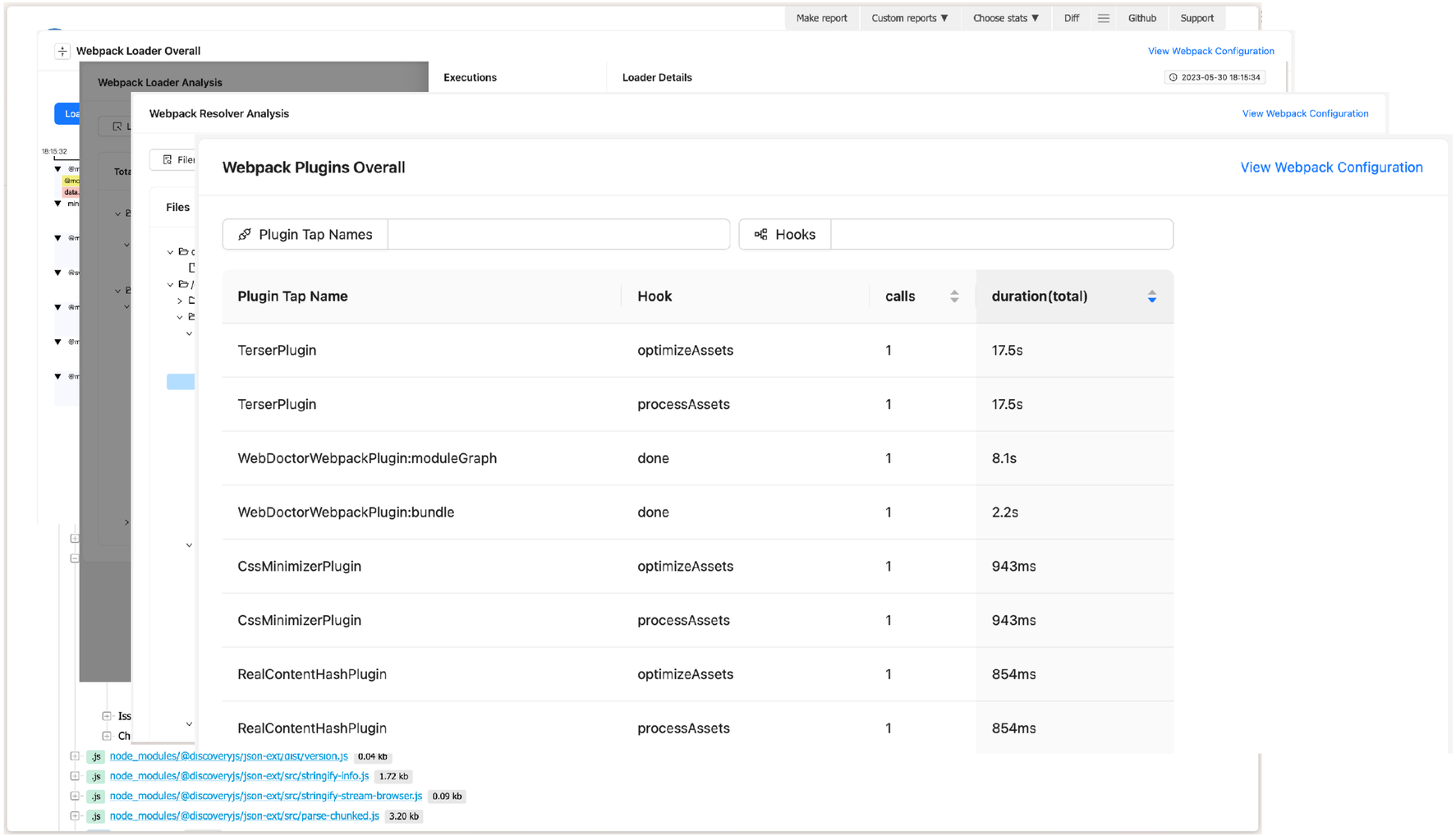

构建产物（stats.json） VS 构建过程

# Diagnostics Tool - 面向构建过程



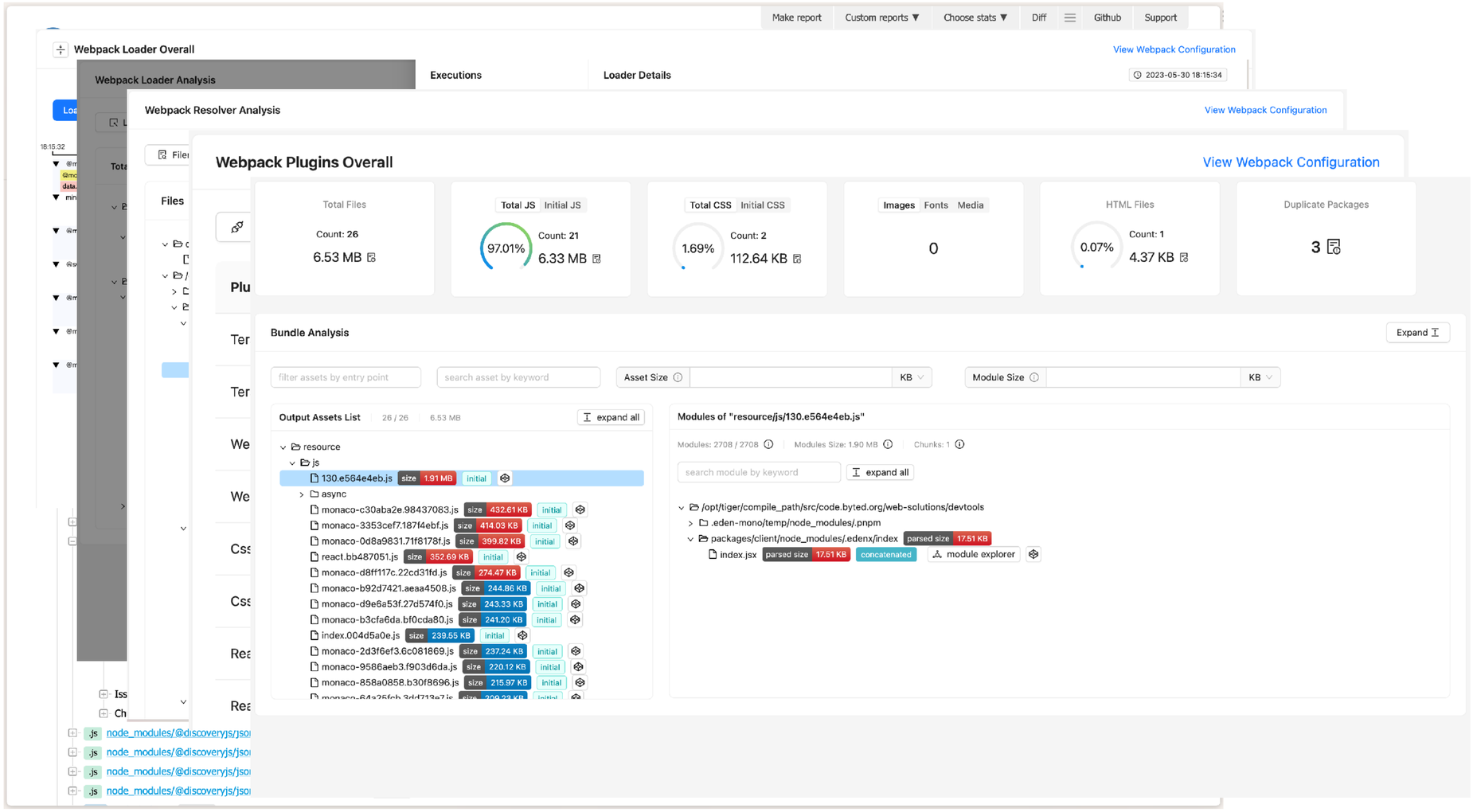

构建产物（stats.json）　　VS　　构建过程

# Diagnostics Tool - 面向构建过程




构建产物（stats.json） VS 构建过程

# Diagnostics Tool - 面向构建过程




```
{
    configs;
    errors;
    chunkGraph;
    moduleGraph;
    packageGraph;
    loader;
    resolver;
    plugin;
    code;
    sourceMap;
    ...
}
```

构建产物（stats.json）　　VS　　构建过程

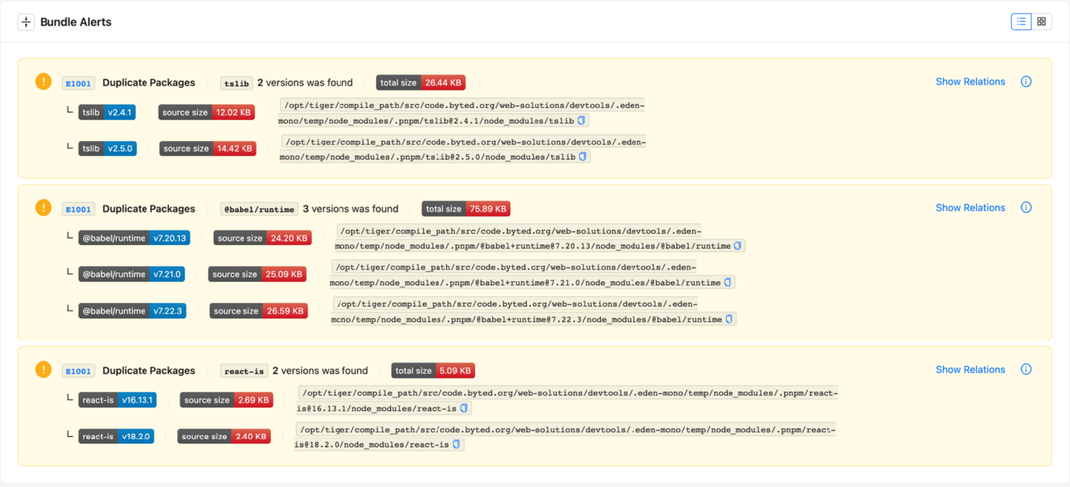
# Diagnostics Tool - 可扩展的规则机制

## 默认规则

Duplicate Packages

Default Import Check

Loader Performance Optimization




## 规则上下文

```
{
  configs;
  errors;
  chunkGraph;
  moduleGraph;
  packageGraph;
  codeAST;
  ...
}
```

## 扩展规则

依赖的引用方式检查

特定依赖的版本检查

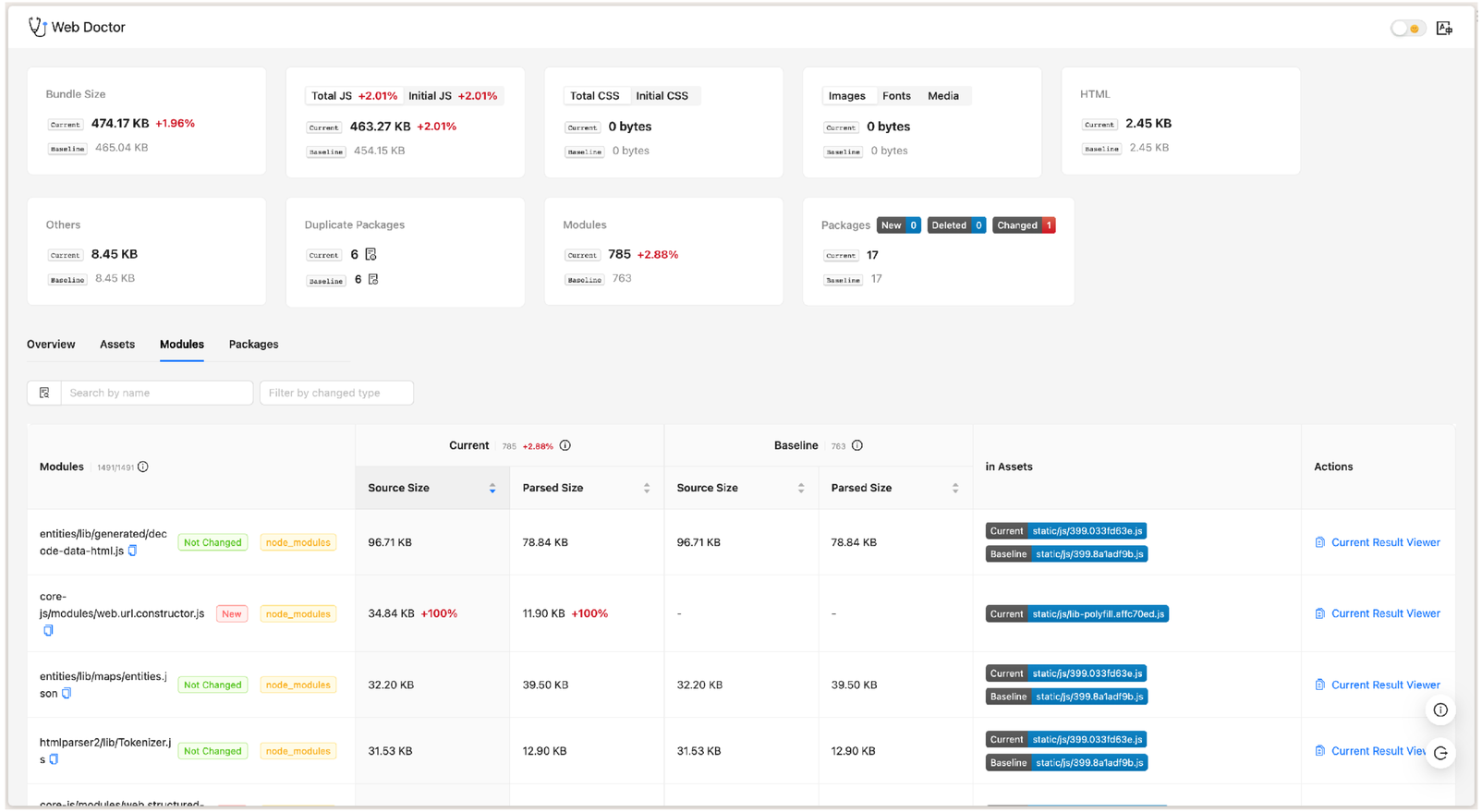
禁止使用特定的语句

# Diagnostics Tool - 与核心研发流程结合

**[CI] Report: Bundle Diff examples/dist/manifest.json** Bundle Size Diff Results: 1.96%

## Bundle Diff 点击查看详情

| Property | Size | Diff | BaseLine |
|---|---|---|---|
| ALL TOTAL Size | 474.17 KB | 1.96% —— 9.13 KB | 465.04 KB |
| JS TOTAL Size | 463.27 KB | 2.01% —— 9.13 KB | 454.15 KB |
| JS Initial Size | 463.27 KB | 2.01% —— 9.13 KB | 454.15 KB |
| CSS TOTAL Size | 0 bytes | 0% —— 0 bytes | 0 bytes |
| CSS Initial Size | 0 bytes | 0% —— 0 bytes | 0 bytes |
| IMGS TOTAL Size | 0 bytes | 0% —— 0 bytes | 0 bytes |
| HTML TOTAL Size | 2.45 KB | 0% —— 0 bytes | 2.45 KB |
| MEDIA TOTAL Size | 0 bytes | 0% —— 0 bytes | 0 bytes |
| FONTS TOTAL Size | 0 bytes | 0% —— 0 bytes | 0 bytes |
| OTHERS TOTAL Size | 8.45 KB | 0% —— 0 bytes | 8.45 KB |



# Diagnostics Tool - 总结

### 💯 业务典型收益

⏳ 发现异常 Loader 耗时，减少 50% 的编译耗时

📦 构建产物体积减少 14%，首屏加载速度提升 39%

🔍 推动二方包升级，重复包从 14 个减少到了 6 个

🚧 多次拦截包体积异常增大的 Merge Request

🩺 内部移动端框架自定义了典型的诊断规则 6 条

通过建设 Diagnostics Tool 有效地防止了应用劣化

# 整体落地情况

x个
接入工程

## Monorepo

降低多项目的维护成本

y个
月活用户

## Micro Frontend

降低多人开发的协作成本

z个
GitHub Star

## Bundler & Build System

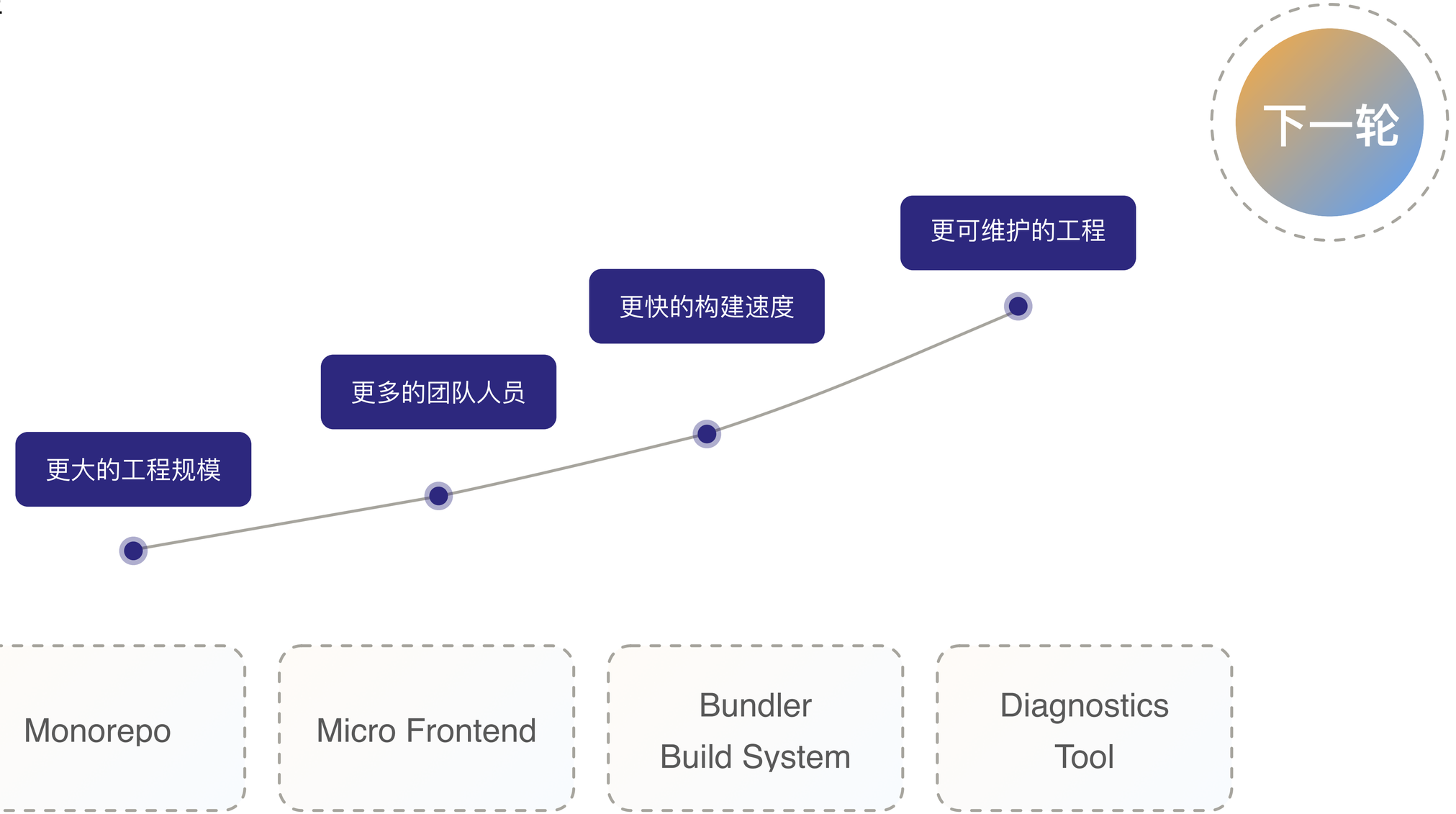
加快巨型应用的构建速度

n次
周下载量

## Diagnostics Tool

有效地防止应用劣化

# 总结与展望


下一轮
更可维护的工程
更快的构建速度
更多的团队人员
更大的工程规模
Monorepo
Micro Frontend
Bundler
Build System
Diagnostics
Tool

# 谢 谢 观 看

thanks