APICloud | IT大咖说 知识共享平台

Node全栈

阿里巴巴·前端技术专家

i5ting 狼叔

AI时代的移动技术革新

Era of AI: Innovations in Mobile Technologies

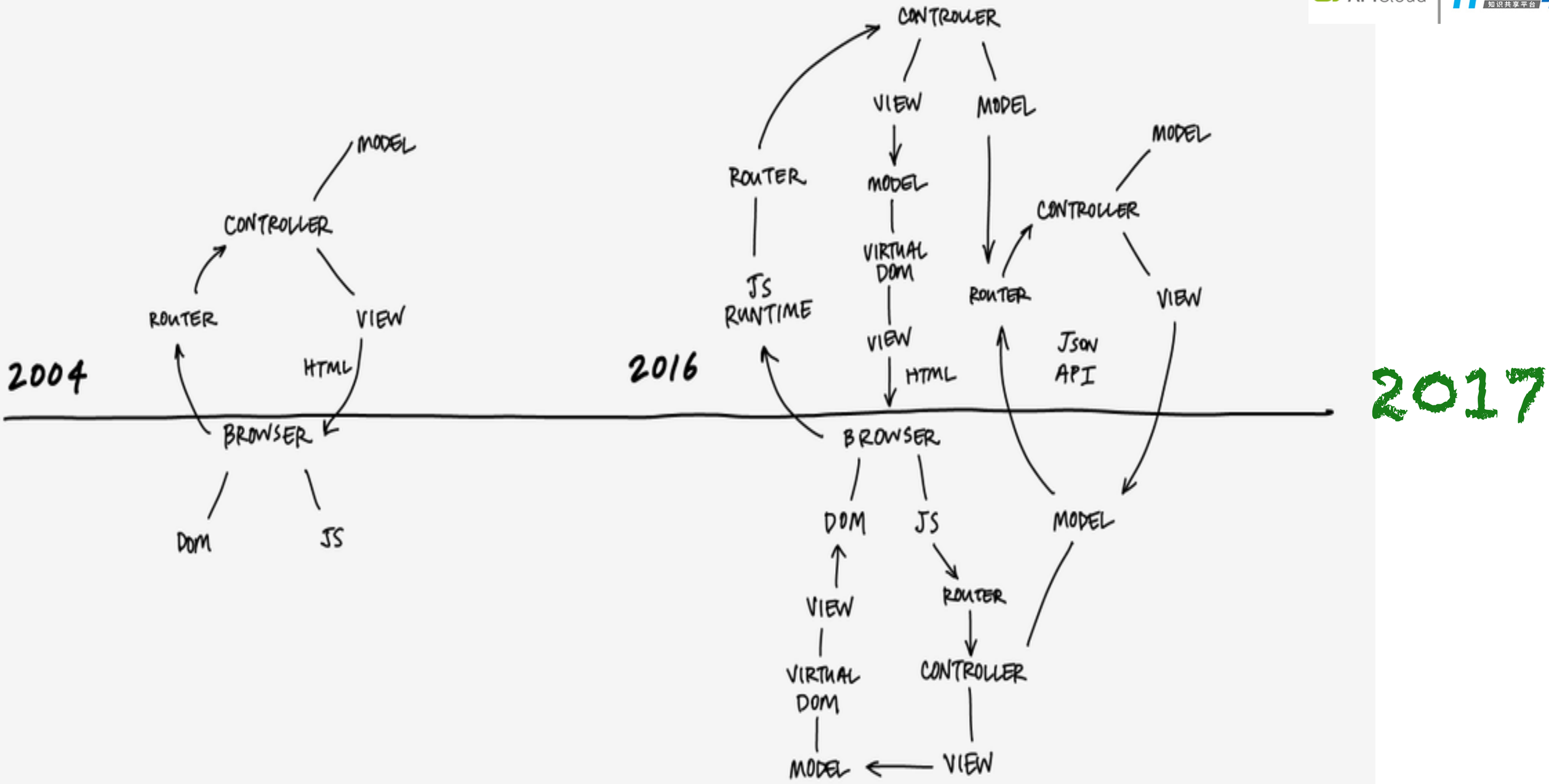
个人简介

他们叫我**狼叔**

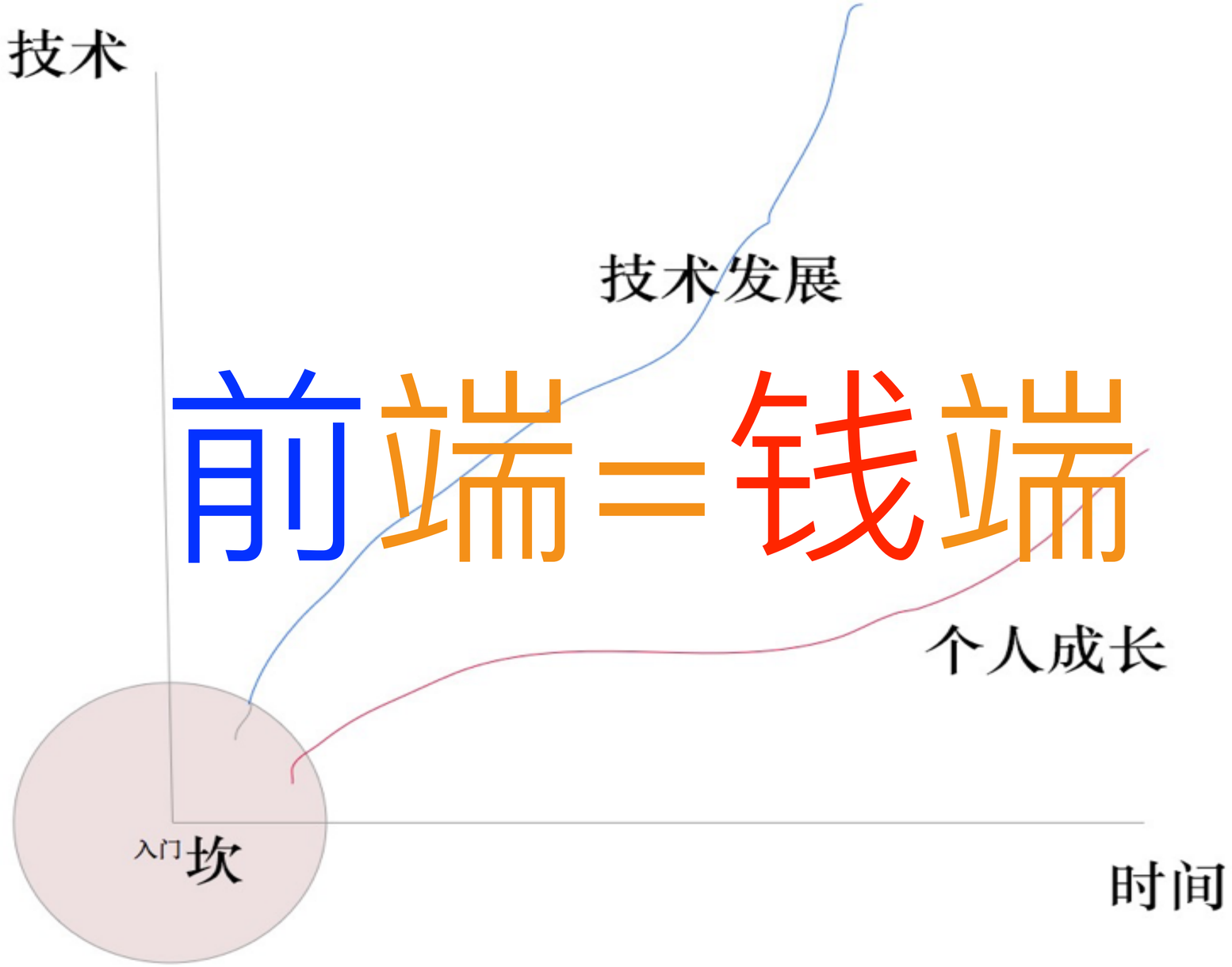
- 姓名：桑世龙
- 部门：阿里巴巴·大文娱事业部
- 简要介绍：嗷呜。。。

Node.js布道　StuQ明星讲师　被坑的CTO　晒娃狂魔

个人简介




预编译
依赖管理
npm, bower
资源处理
DataURL
APICloud
IT大咖说
知识共享平台
压缩
uglifyjs, jsmin， csso
React组件化、Vueis（未来趋势）
Backbone，
流行）
CSS预处理器
图片压缩
imagemin
js友好语言
coffee, babel， typescript
模块系统
CommonJs, AMD, ES6 Modules
基础）
模板引擎
jade, handlebars， nunjucks
模块加载器
require.js, jspm， sea， system.js
Cordova
NPM Scripts
Electron
Webpack
CSS预处理
PostCSS、less、sass
构建工具
grunt, gulp, make
平台工
构建系统
模块打包器
browserify, webpack
现代Web开发

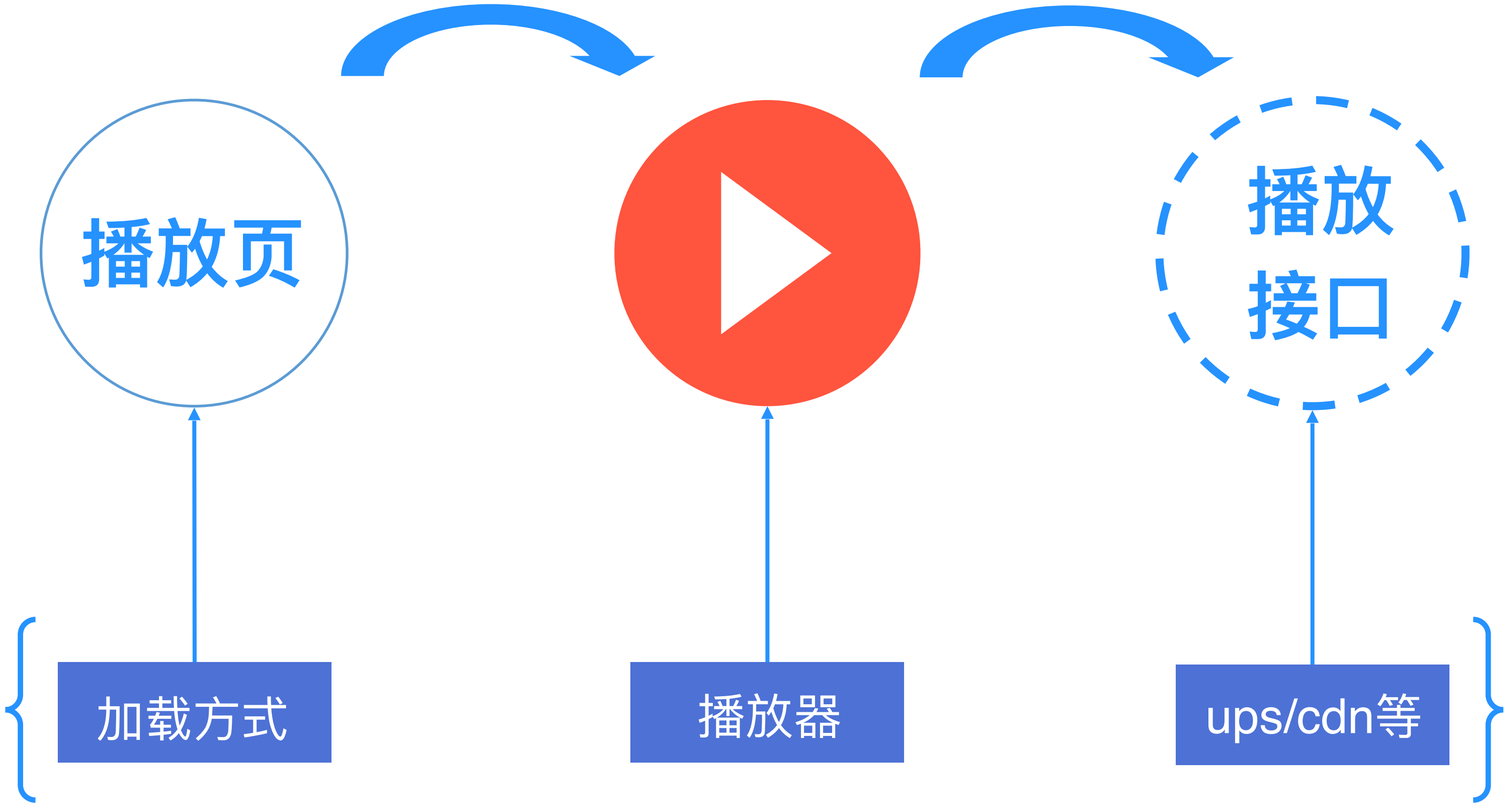

技术
技术发展
前端=钱端
个人成长
入门坎
时间

目录

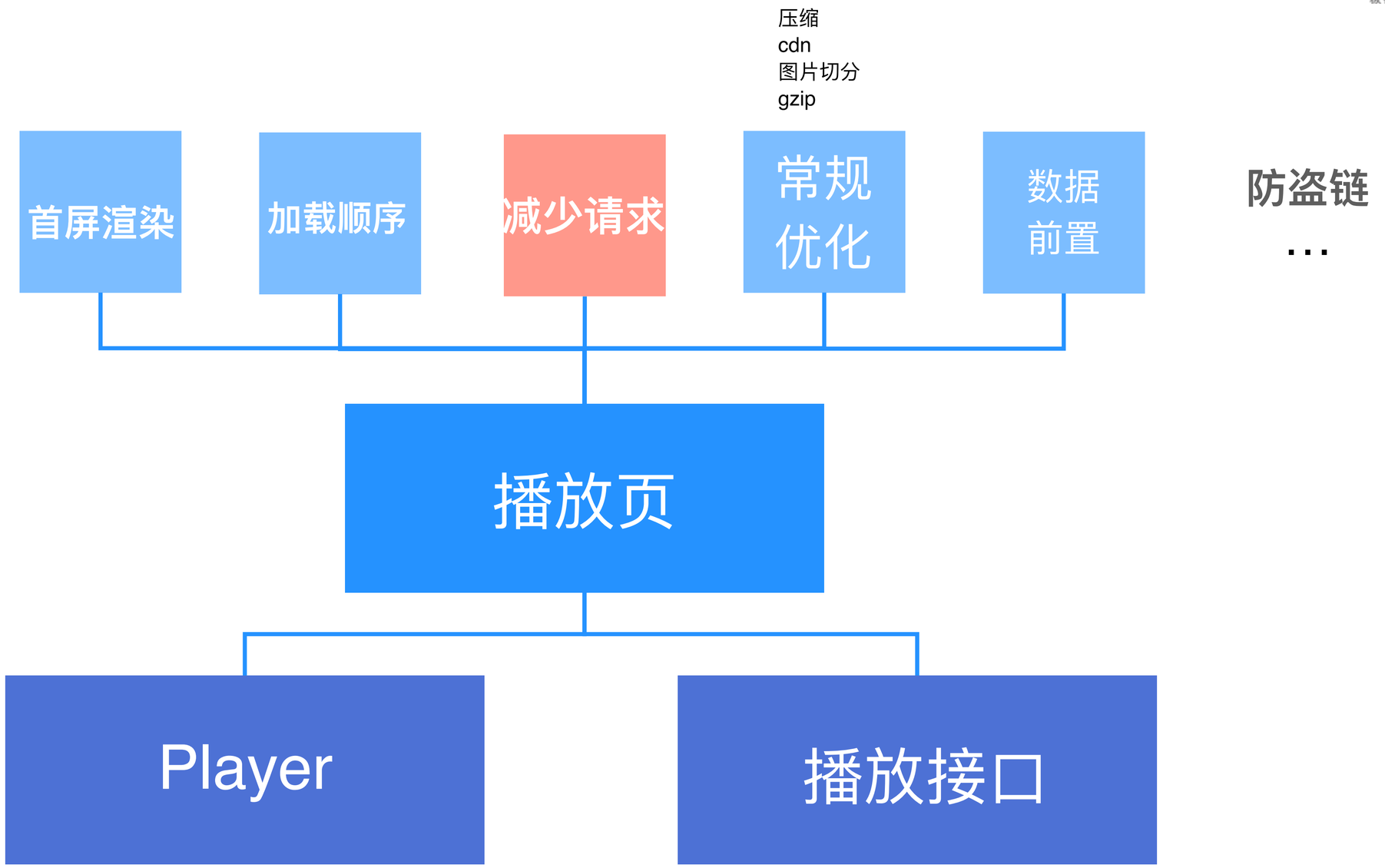
1、AI时代的三端分析

我在做什么

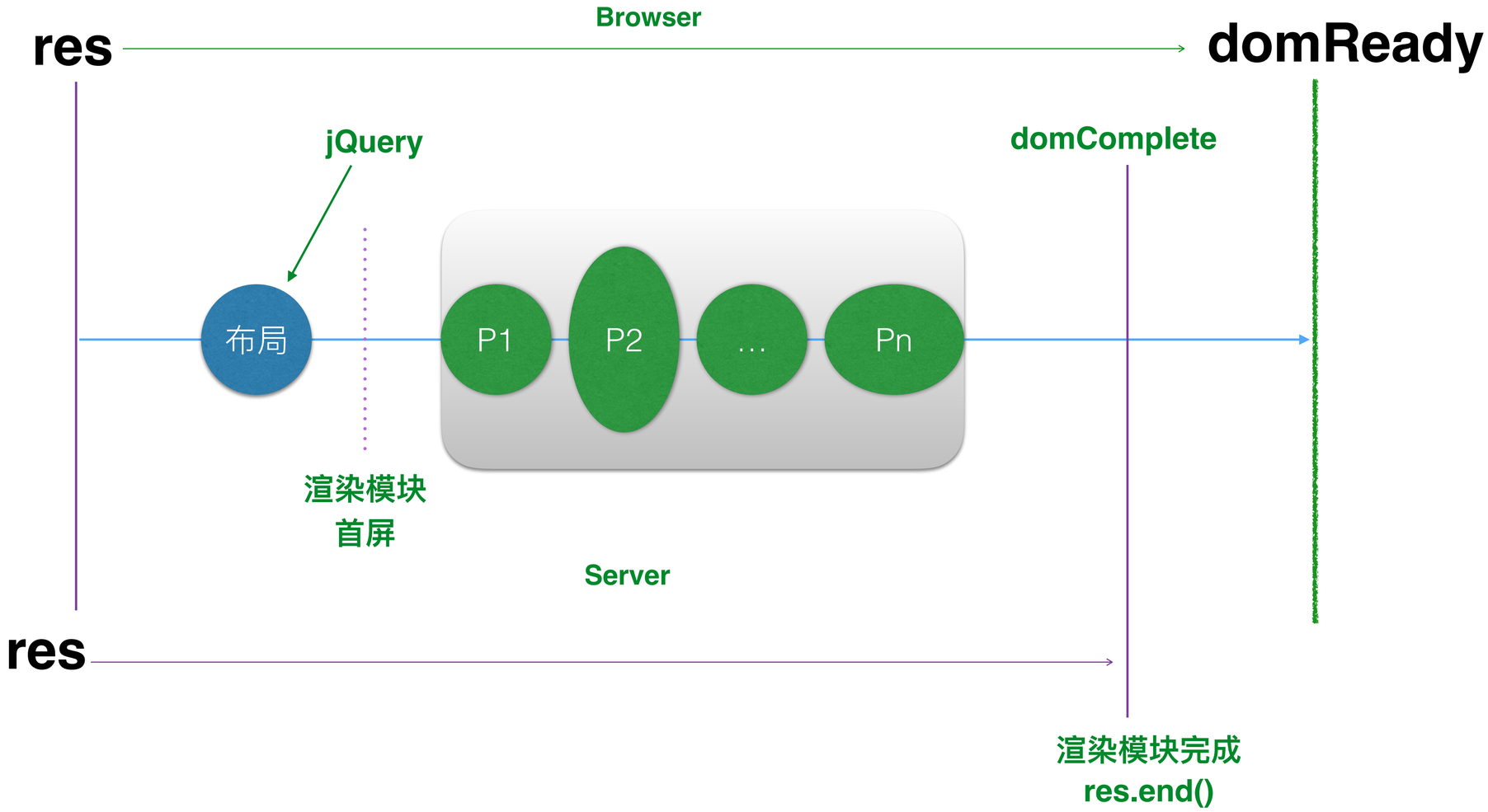
播放器是有前置和后置条件的，前置是播放页加载方式，后置是各种播放接口，非单一优化点



优化无止境



分块加载技术Bigpipe



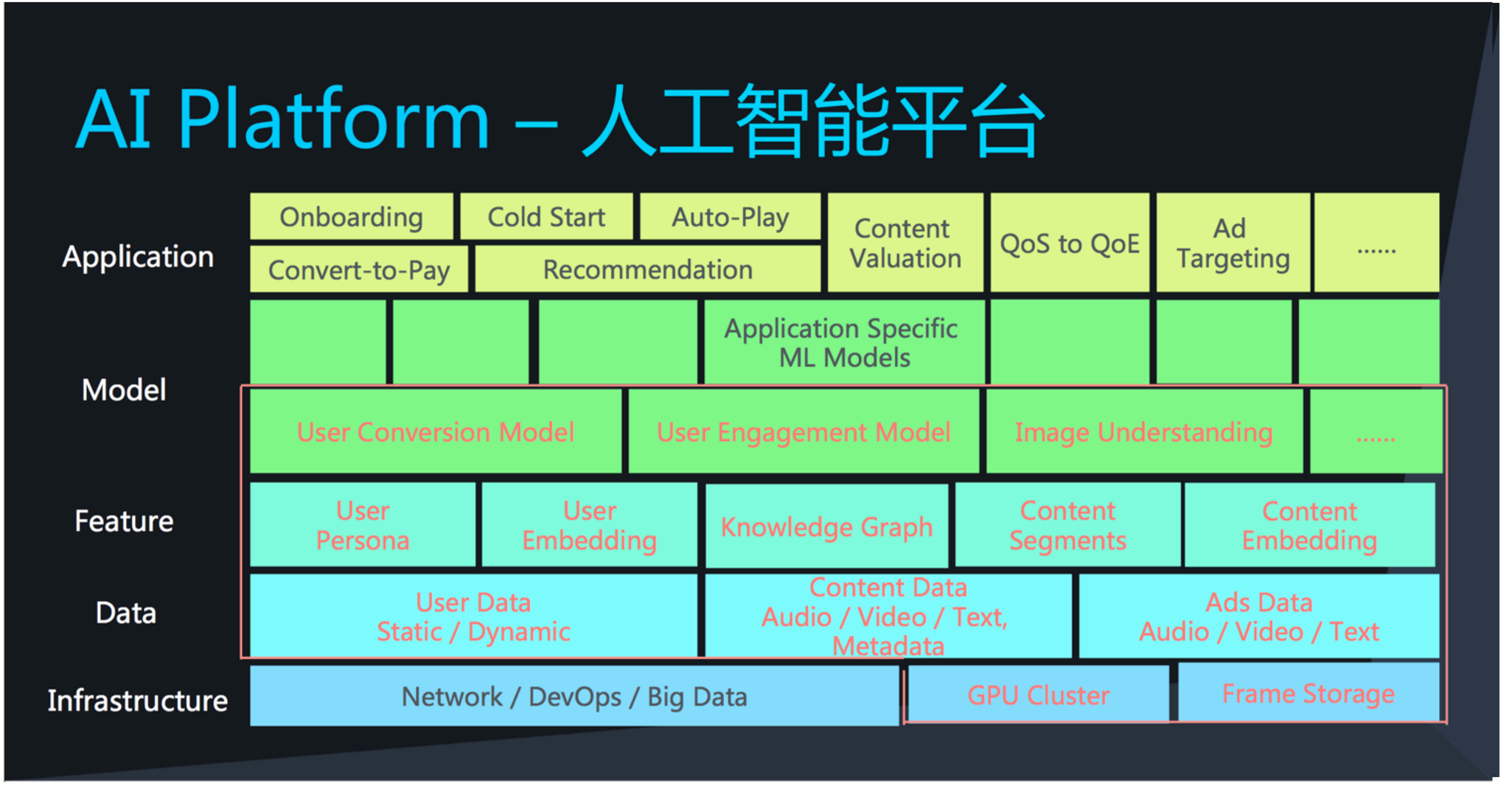
PC & H5还有前途吗?

移动互联网时代

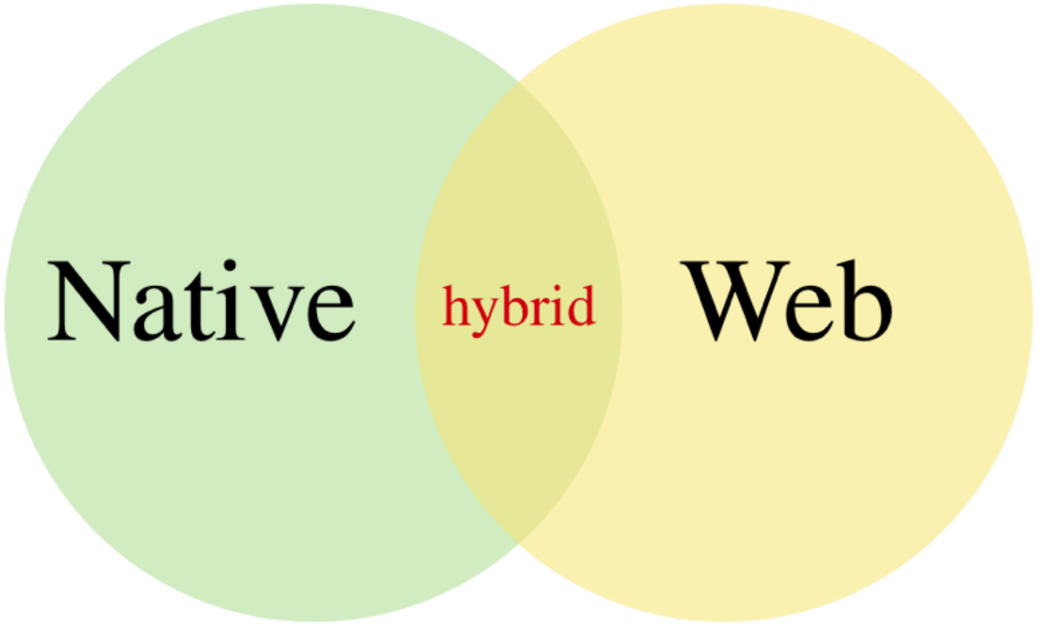
已死?

太远?

AI时代已经来了


AI Platform – 人工智能平台
Application
Onboarding
Cold Start
Auto-Play
Convert-to-Pay
Recommendation
Content Valuation
QoS to QoE
Ad Targeting
.....
Model
Application Specific ML Models
User Conversion Model
User Engagement Model
Image Understanding
.....
Feature
User Persona
User Embedding
Knowledge Graph
Content Segments
Content Embedding
Data
User Data Static / Dynamic
Content Data Audio / Video / Text, Metadata
Ads Data Audio / Video / Text
Infrastructure
Network / DevOps / Big Data
GPU Cluster
Frame Storage

AI时代的前端



从单兵，到组团

ai三端对齐
移动端
pc
pc client

从移动端的发展看未来

native < hybrid < rn/weex < h5

c/s架构到b/s架构，该对瘦客户端予以一定的支持

重点是开发变得无比复杂

Here is the advantage of the layout of the way

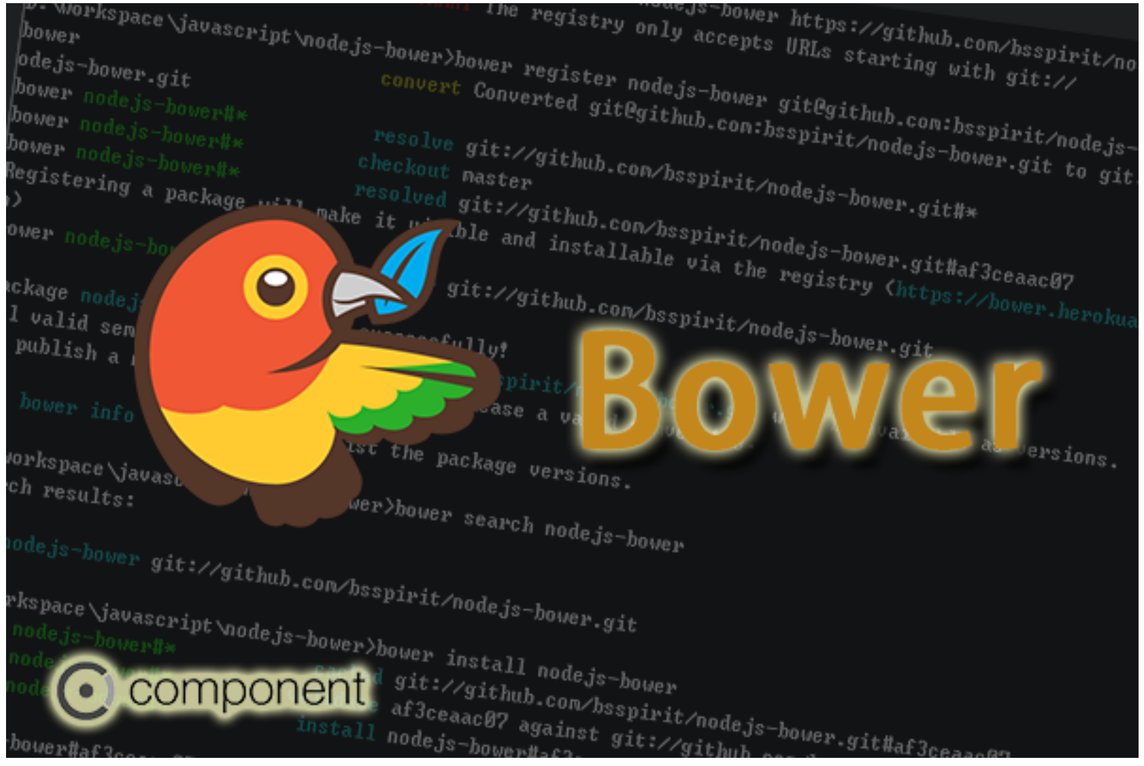
2、前端工程化构建与Webpack

依赖下载

- 下载某个库或插件
- 下载它的依赖，以及依赖的依赖
- 无穷尽…

依赖管理

- npm/bower



脚本加载

```
1 <script src="jquery.js"></script>
2 <script>$.uiBackCompat = false;</script>
3 <script src="jquery-ui.js"></script>
```

window　　eval　　iframe　　$.getScript()

模块系统

- 使用标准的模块系统来处理依赖和导出
- 每个文件是一个模块
- 使用模块加载器或打包器进行处理

AMD, CommonJS, ES6 Modules

CommonJS

- require("..."): Loads module, returns exports
  - require("./helpers.js"): ... by relative path, returns exports
  - require("jquery"): ... from dependencies manager folder
- exports or module.exports export object

```
var $ = require("jquery");

exports.doSomething = function() {
    return 42;
}
```

module
exports
require
global

AMD

```
define("module", ["dep1", "dep2"], function(d1, d2) {
  return someExportedValue;
});
require(["module", "../file"], function(module, file) { /* ... */ });
```

CommonJS 中逐渐分裂出了 AMD，专门用于浏览器的
因为浏览器没有io读写api

https://github.com/amdjs/amdjs-api/wiki/AMD

核心代码

Amd

- define
- require

```
void function() {
        var mapping = {}, cache = {};
        window.define = function(id, func) {
                mapping[id] = func
        };
        window.require = function(id) {
                if (!/\.js$/.test(id)) {
                        id += ".js"
                }
                if (cache[id]) {
                        return cache[id]
                } else {
                        return cache[id] = mapping[id]()
                }
        }
}();

define("test.js", function(exports) {
        exports = {};

        exports.var = 1;
        exports.say = function(){
                alert(this.var);
        }

        return exports;
});
```

cache

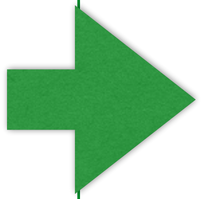
按需加载

以前的做法

- 使用多个<script>标签加载
- 手动管理顺序
- 手动管理加载哪些

```
<script src="/wreq.js"></script>
<script src="/js/entry.js" type="text/wreq"></script>

<div id="res">[]</div>
```

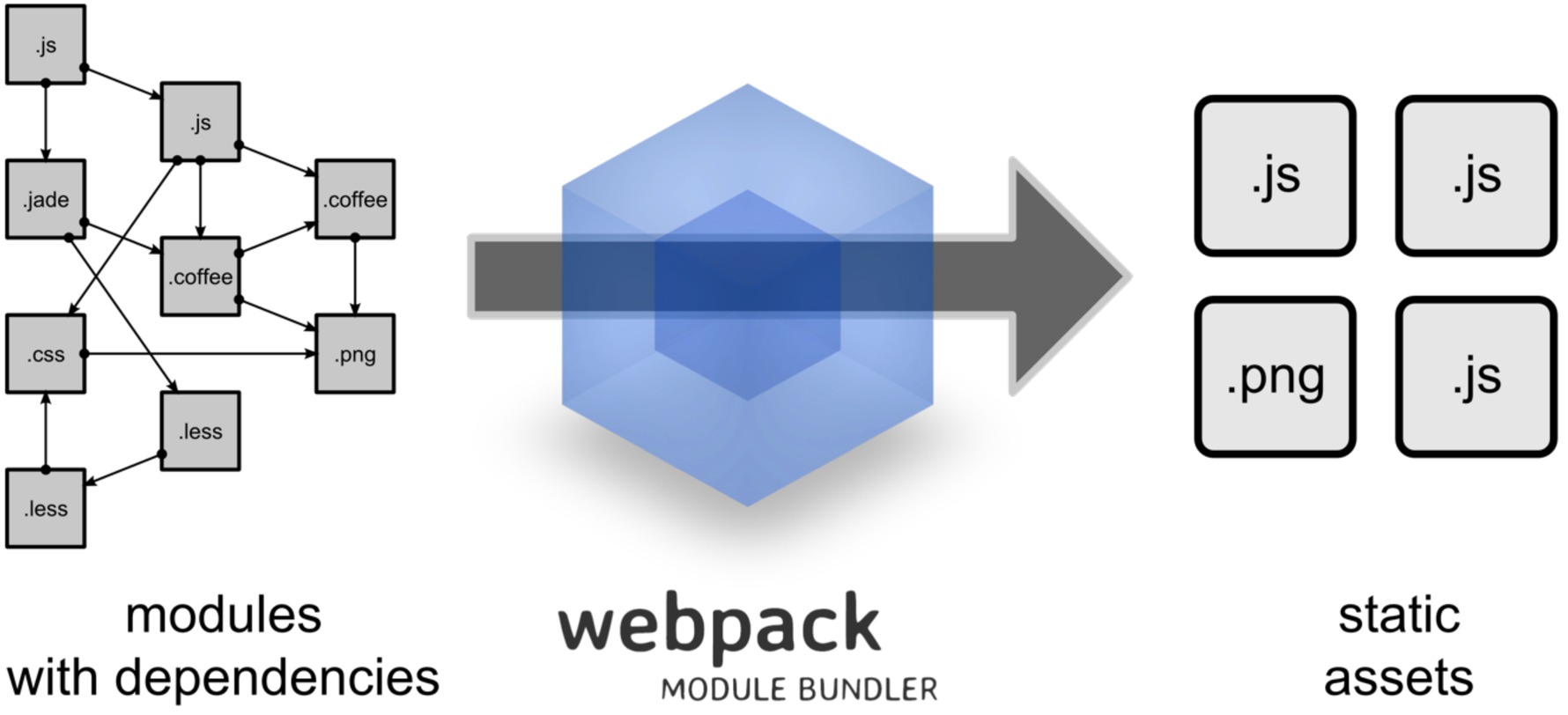
现在的做法



	开发环境	产品环境
	module loader	module bundler
依赖管理	• runs in the browser and loads modules when they are requested • easy to use • less optimized for production usage	• runs in preparation and bundles modules into static files • needs a preparation/build step • more optimized for production usage
模块系统	require.js systemjs	r.js\browserify\ webpackjspm

grunt/gulp

所有资源都是模块


.js
.js
.jade
.coffee
.coffee
.css
.png
.less
.less
modules
with dependencies
webpack
MODULE BUNDLER
.js
.js
.png
.js
static
assets

Webpack打包过程

- 从配置文件里找到entry point
- 解析模块系统
- 解决依赖
- 模块依赖处理（读取，解析，解决）
- 合并所有使用的模块
- 合并模块系统的运行时环境
- 产生打包后的文件

```
webpackJsonp([1], [
    /* 0 */
    /***/
    (function (module, exports, __webpack_require__) {

        __webpack_require__(1)

        __webpack_require__.e /* require.ensure */ (0).then((function (require) {
            __webpack_require__(6)
        }).bind(null, __webpack_require__)).catch(__webpack_require__.oe)

        /***/
    }),
    /* 1 */
    /***/
    (function (module, exports) {

        console.log('a')

        /***/
    })
], [0]);
```

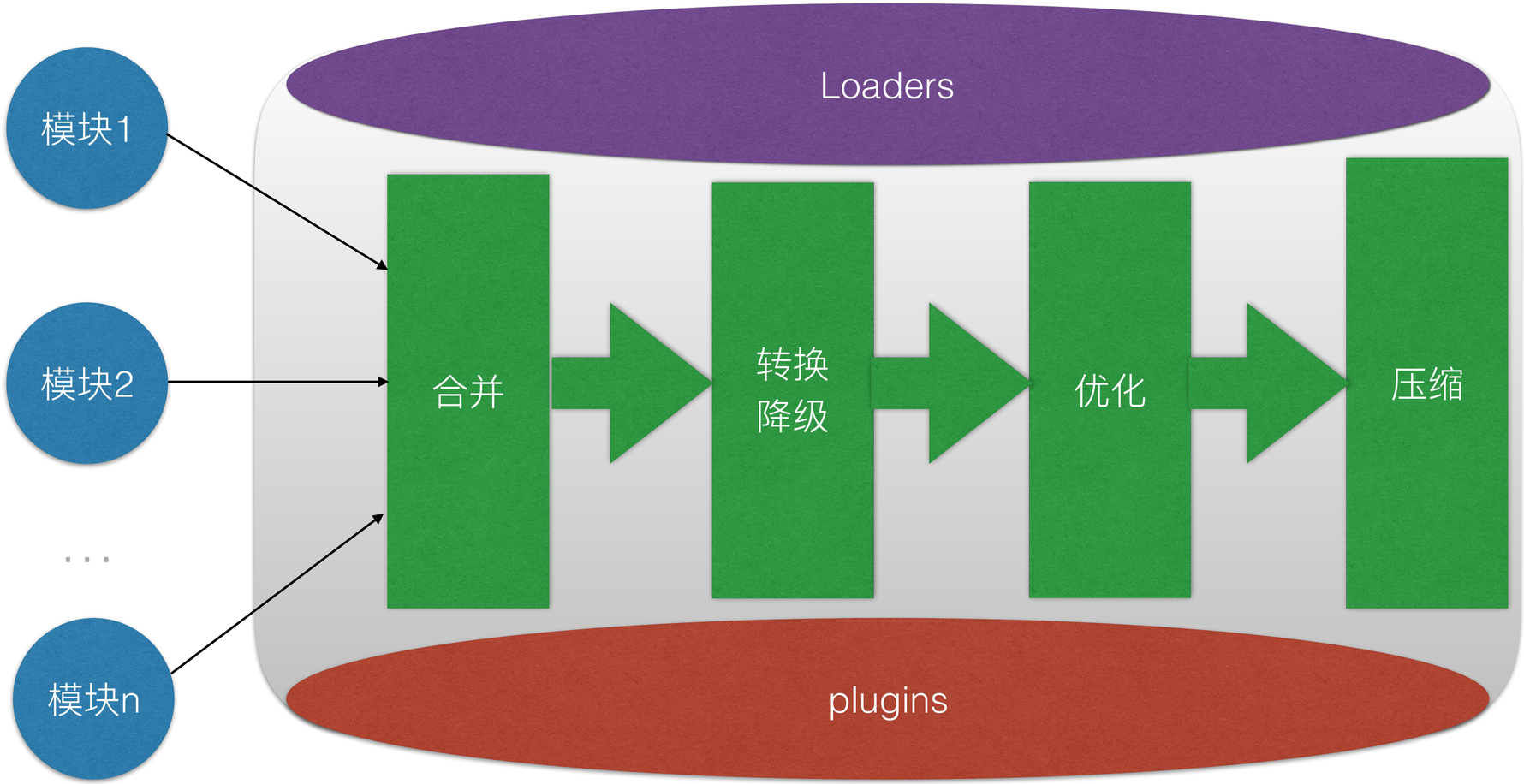
浏览器里解包过程

- 通过<script>加载webpack打包后的文件
- 加载模块运行时环境
- 加载entry point
- 读取依赖
- 解决依赖
- 执行（带有依赖的）entry point

演进过程

1. <script>混乱加载
2. 各种模块系统标准，commonjs/amd/es6 module
3. 模块加载器，requirejs/sea/systemjs
4. 模块打包器，webpack/r.js/jspm/browserfy
   A. 合并入口，对外暴露的只有entry point
   B. 提供浏览器运行环境（内置模块加载器）
   C. 优化（tree-shaking、DCE无用代码移除等）

时髦的开发



webpack

你只管写就好了

其他的webpack来

你能写好，就好了。。。

你能写，就好了。。

你能，就好了。。

你，就好了。。

打包器的演进

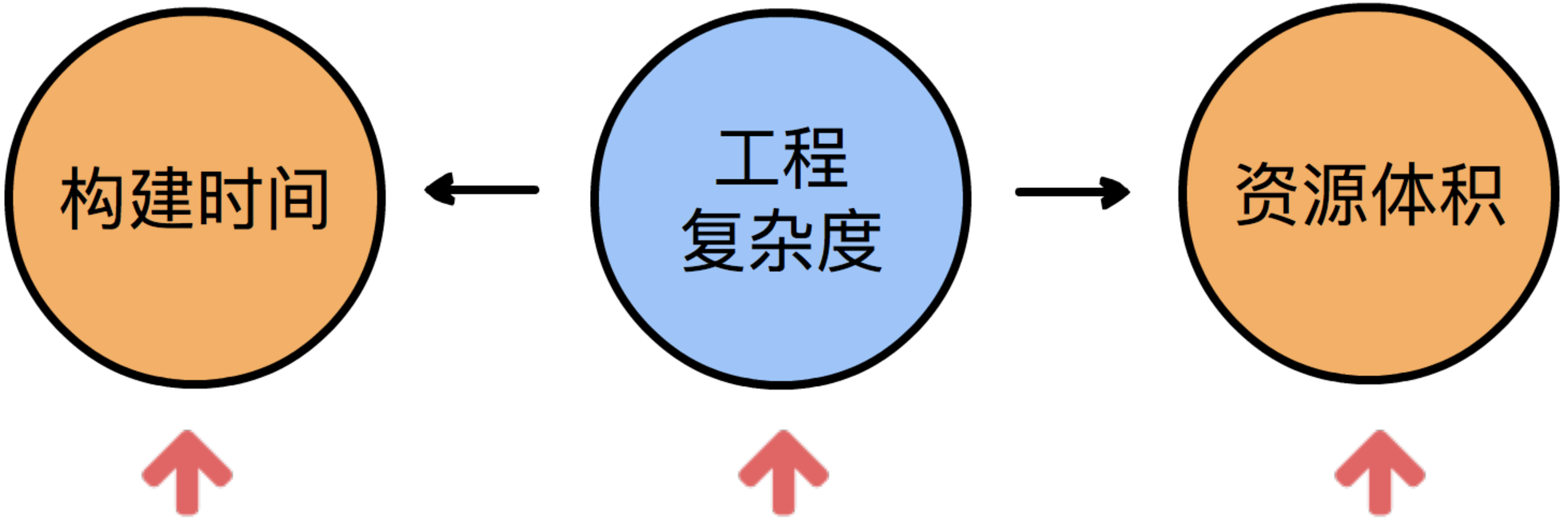
- browserify
- webpack 1
- rollup
- webpack 2
- Parcel

commonjs规范

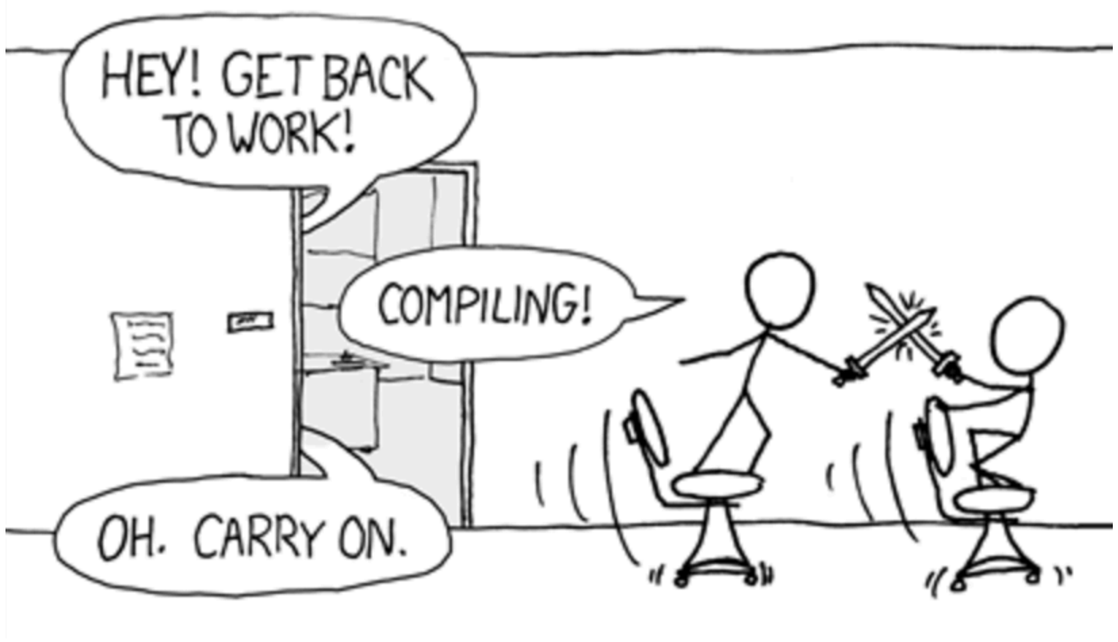
多种模块

利用es6模块能静态分析语法树的特性，只将需要的代码提取出来打包，能大大减小代码体积

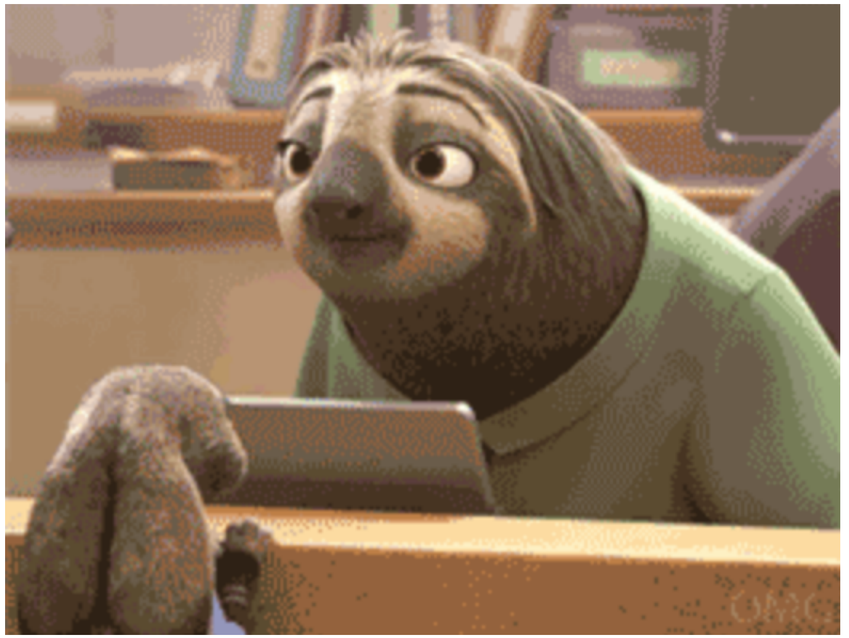
前端工程化



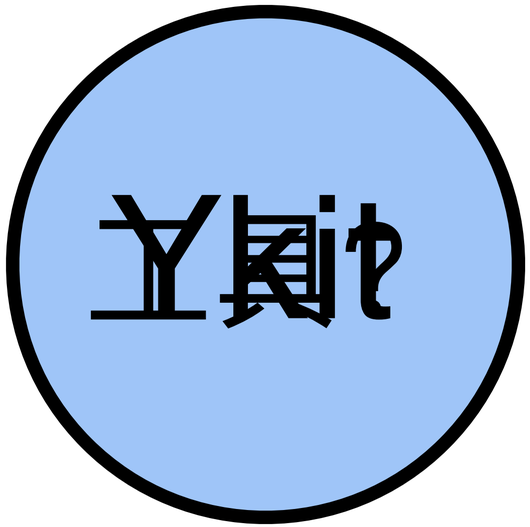
当下面临的问题



开发测试成本增加



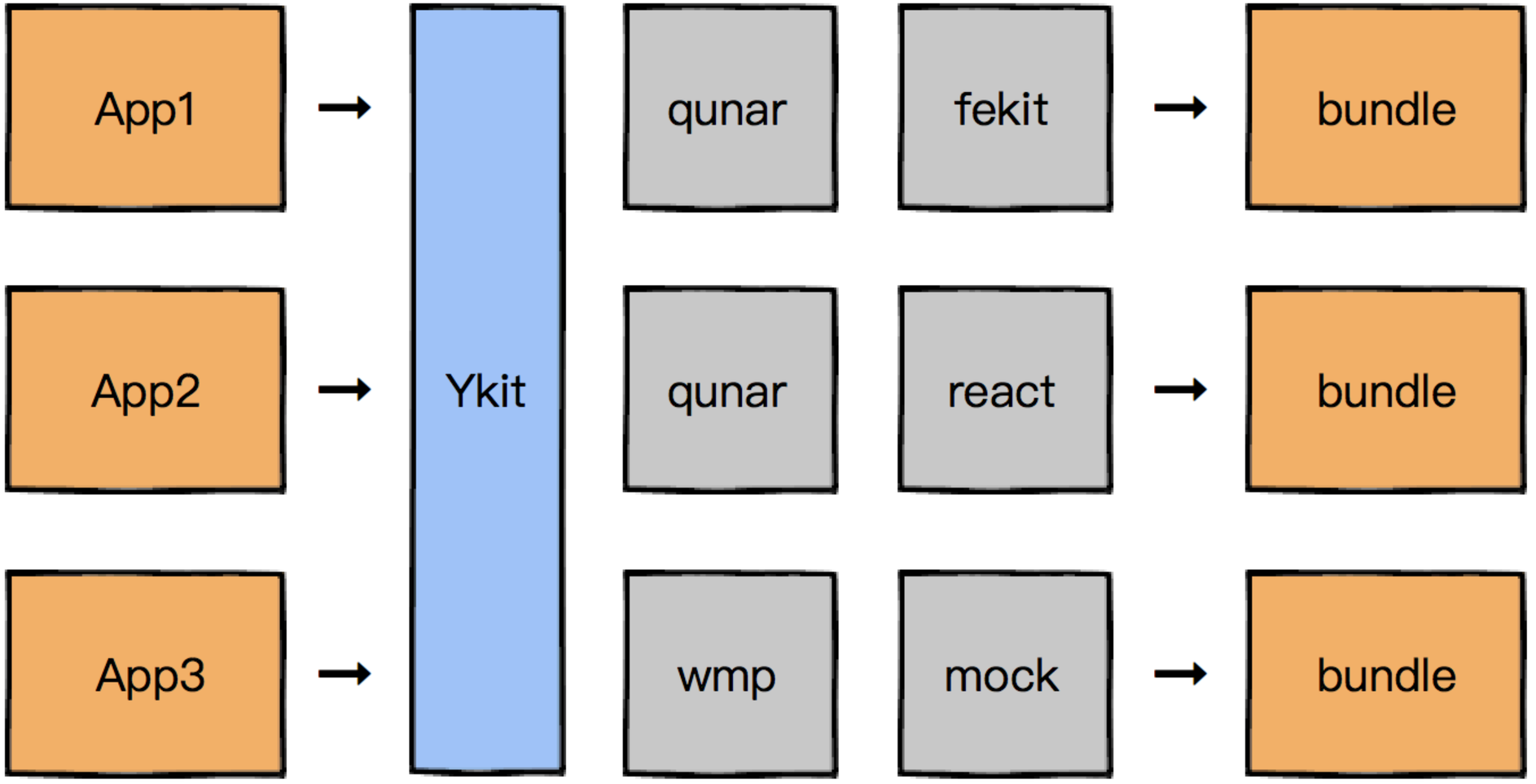
客户端性能受影响



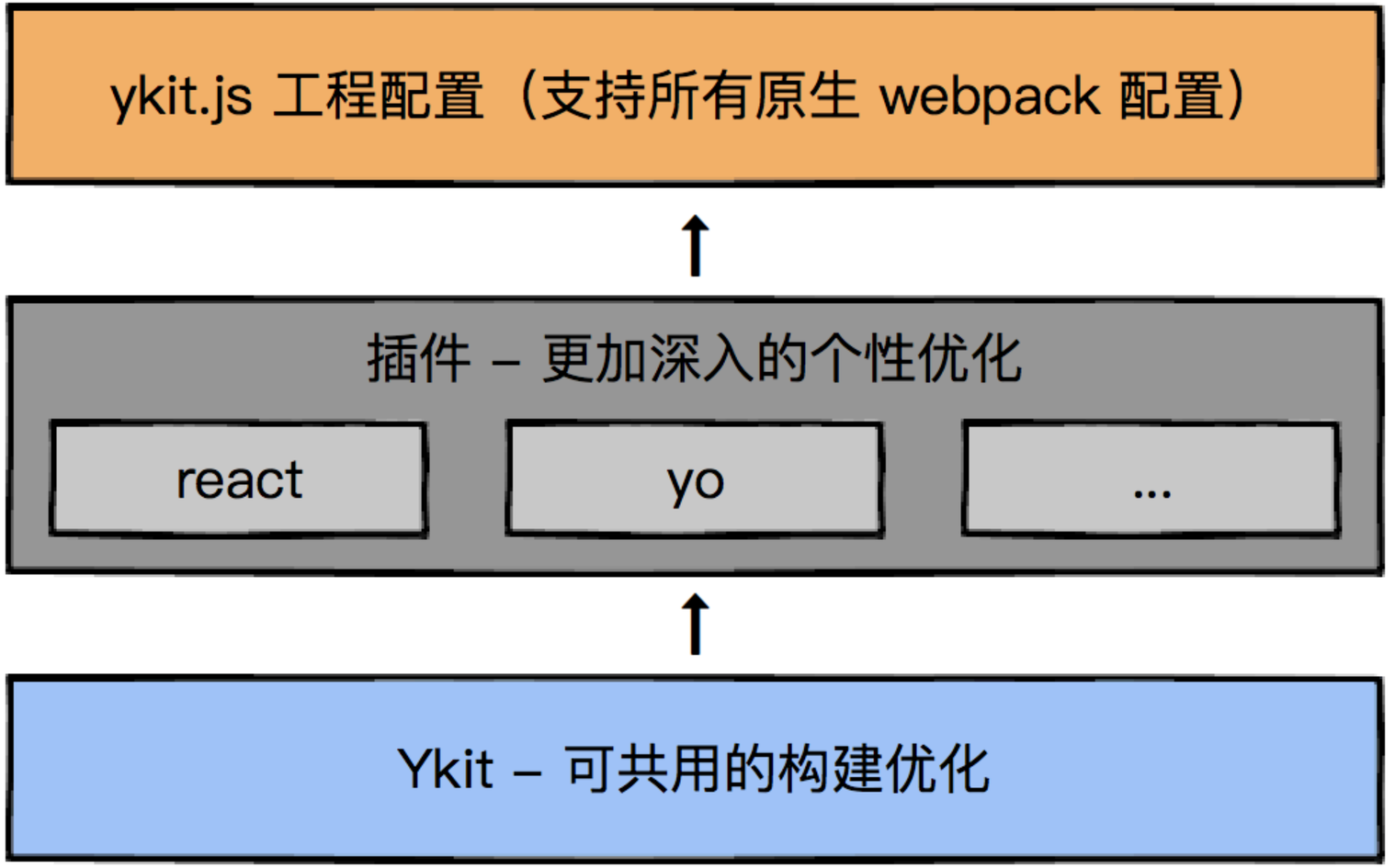
使业务同学
专注于研发

https://ykit.ymfe.org/

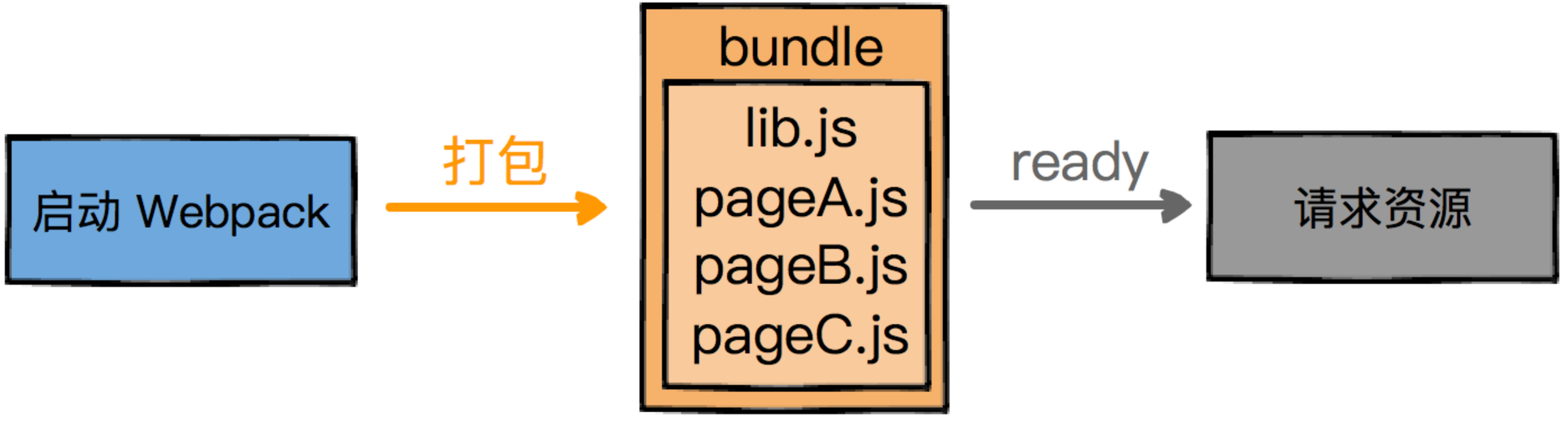
YKit工作原理


App1
App2
App3
Ykit
qunar
qunar
wmp
fekit
react
mock
bundle
bundle
bundle

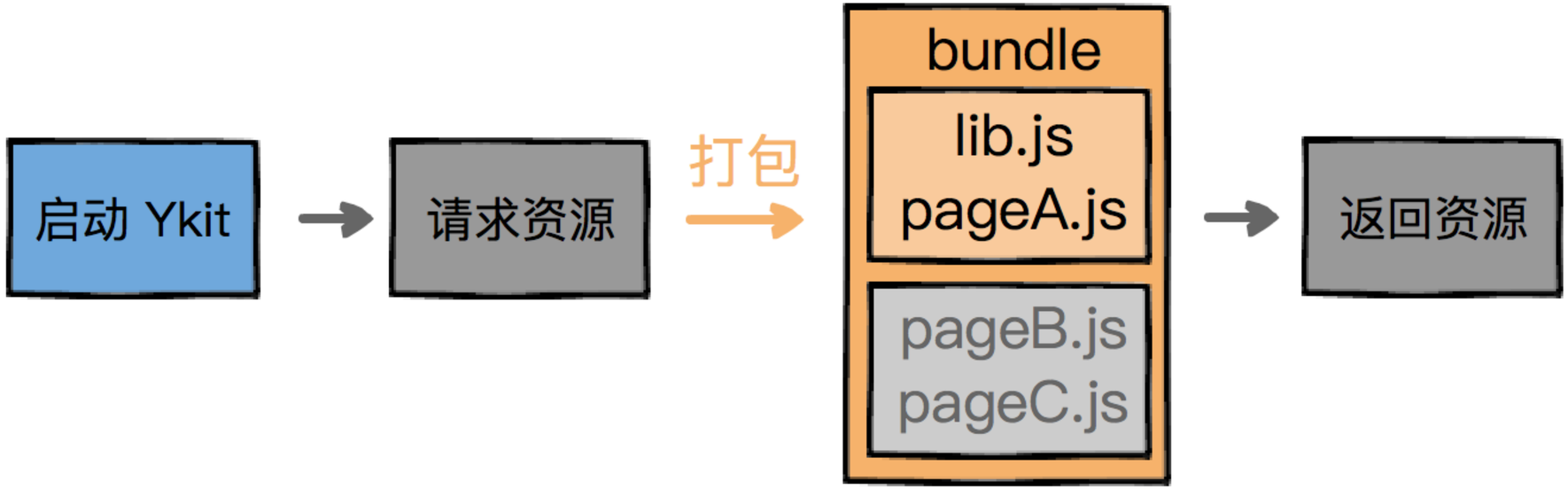
构建优化


ykit.js 工程配置（支持所有原生 webpack 配置）
插件 – 更加深入的个性优化
react
yo
...
Ykit – 可共用的构建优化

Webpack本地服务

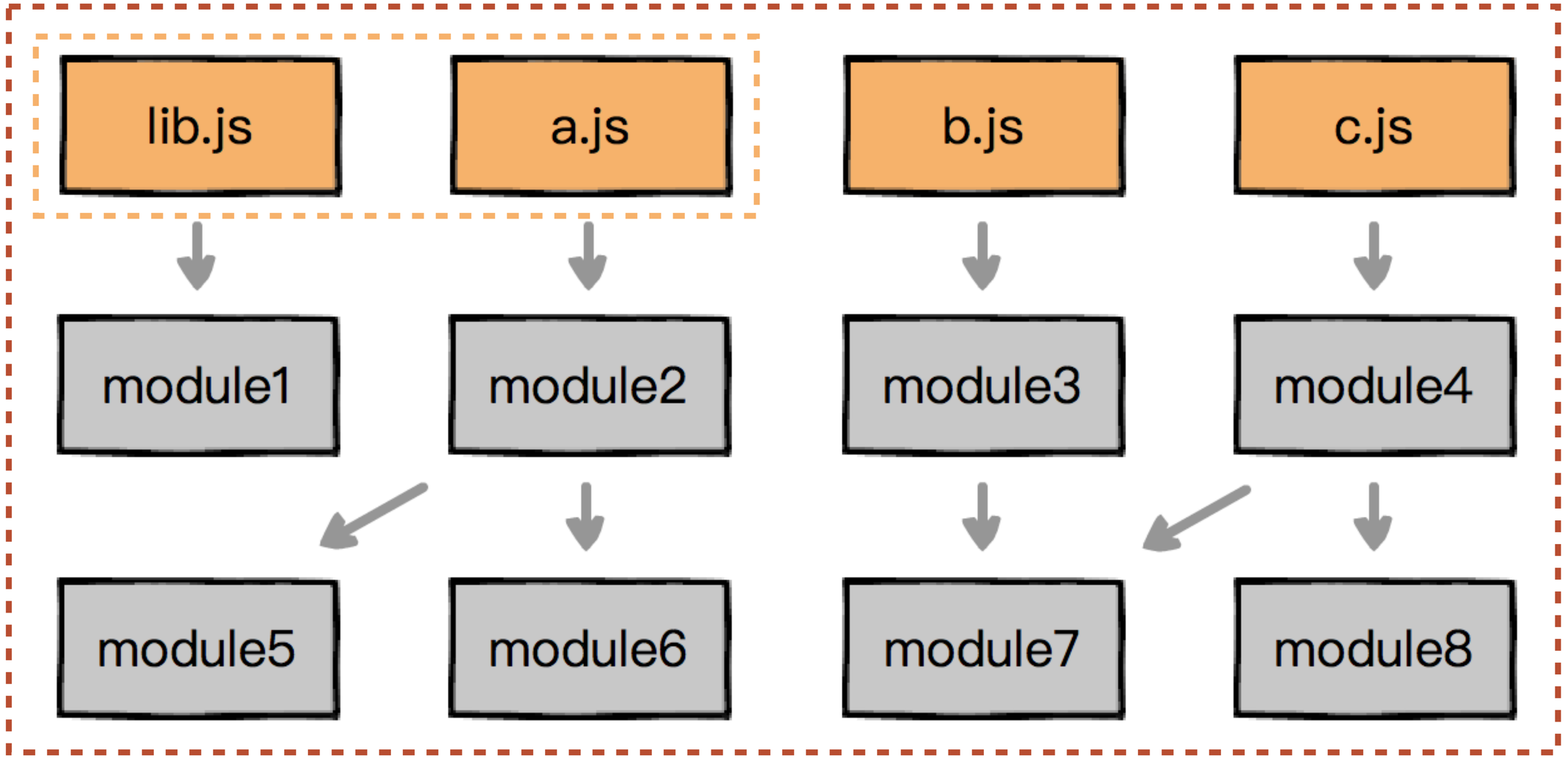


YKit本地服务

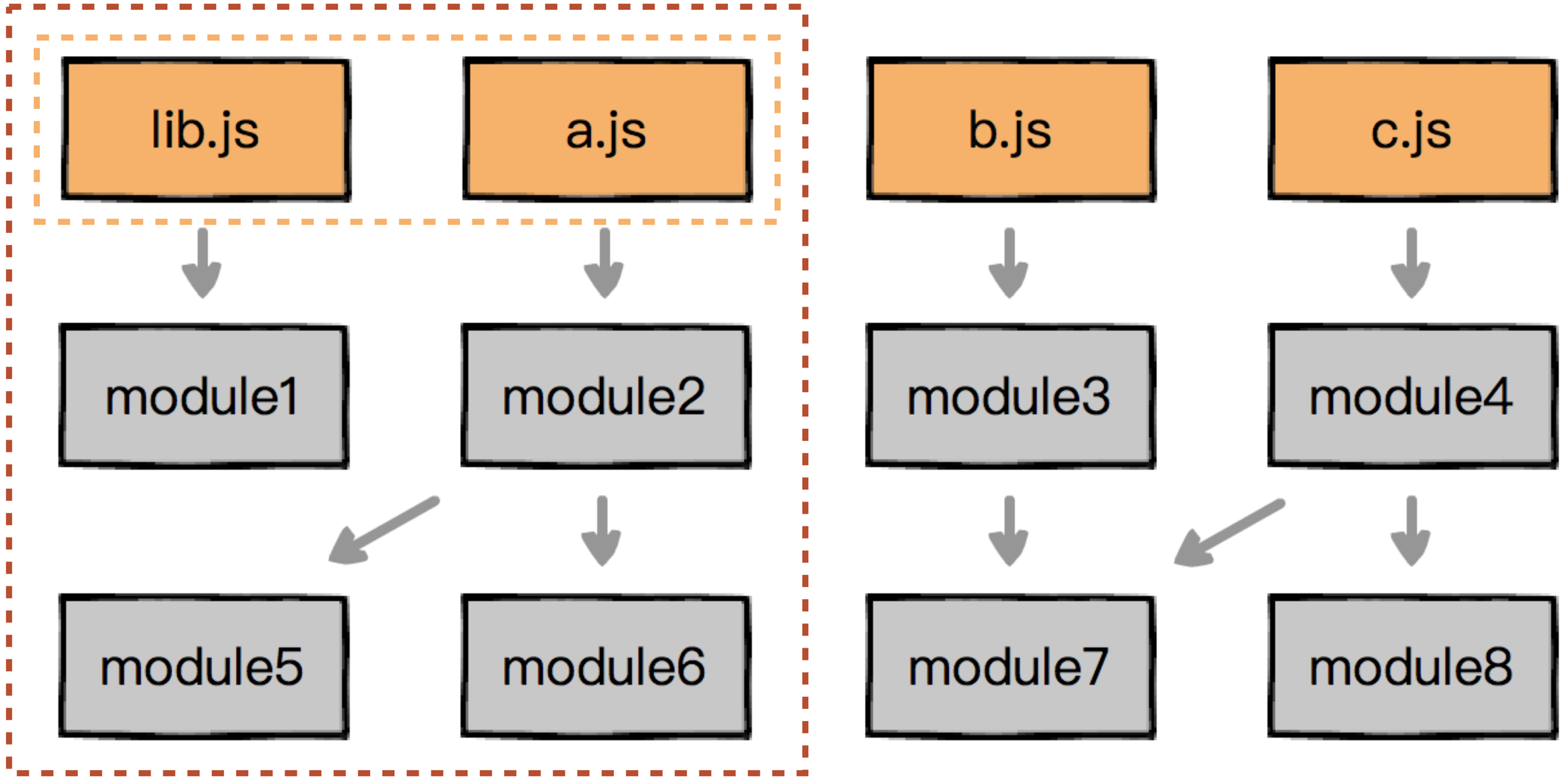


Ykit 会根据请求的资源进行入口过滤，仅打包页面所请求的资源

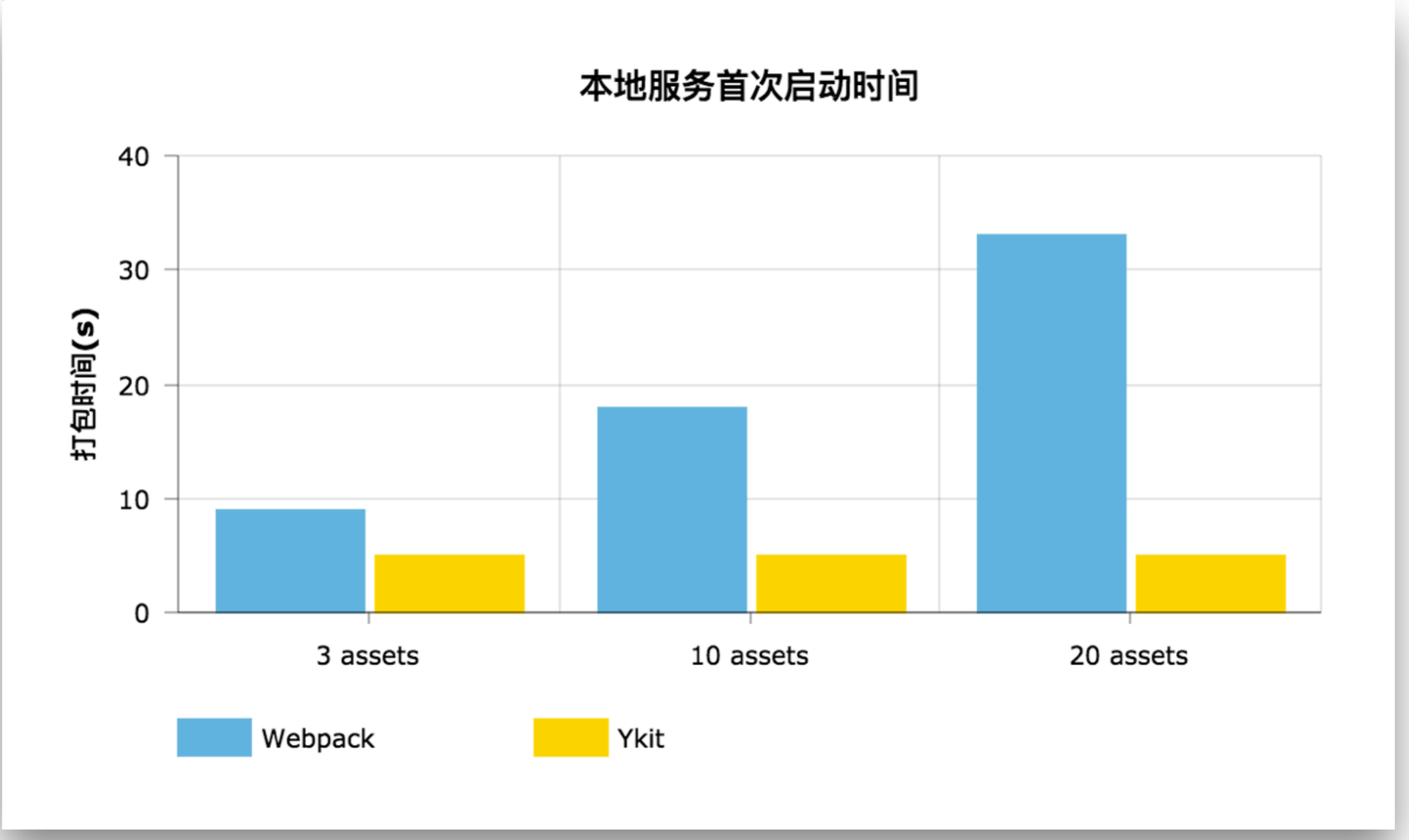
Webpack本地服务



YKit本地服务

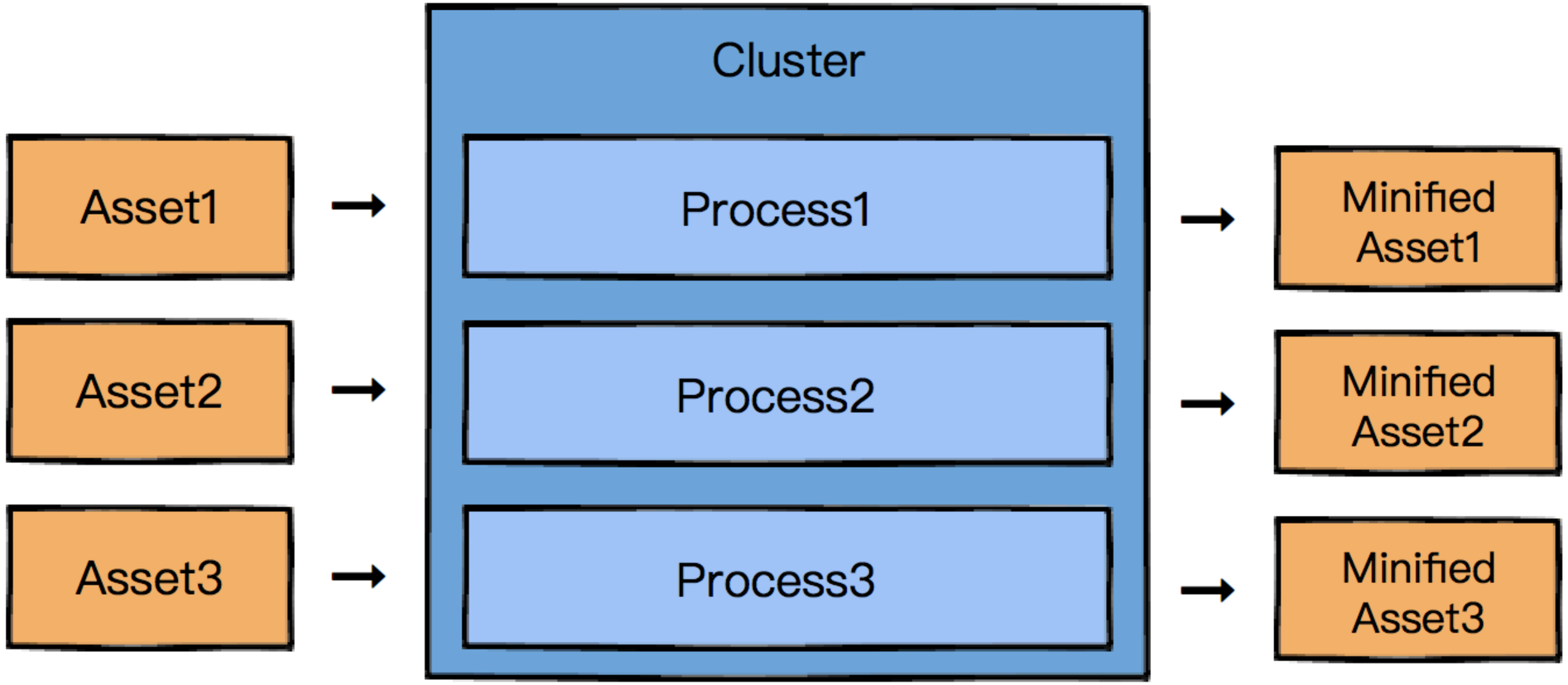

lib.js
a.js
b.js
c.js
module1
module2
module3
module4
module5
module6
module7
module8

效果对比

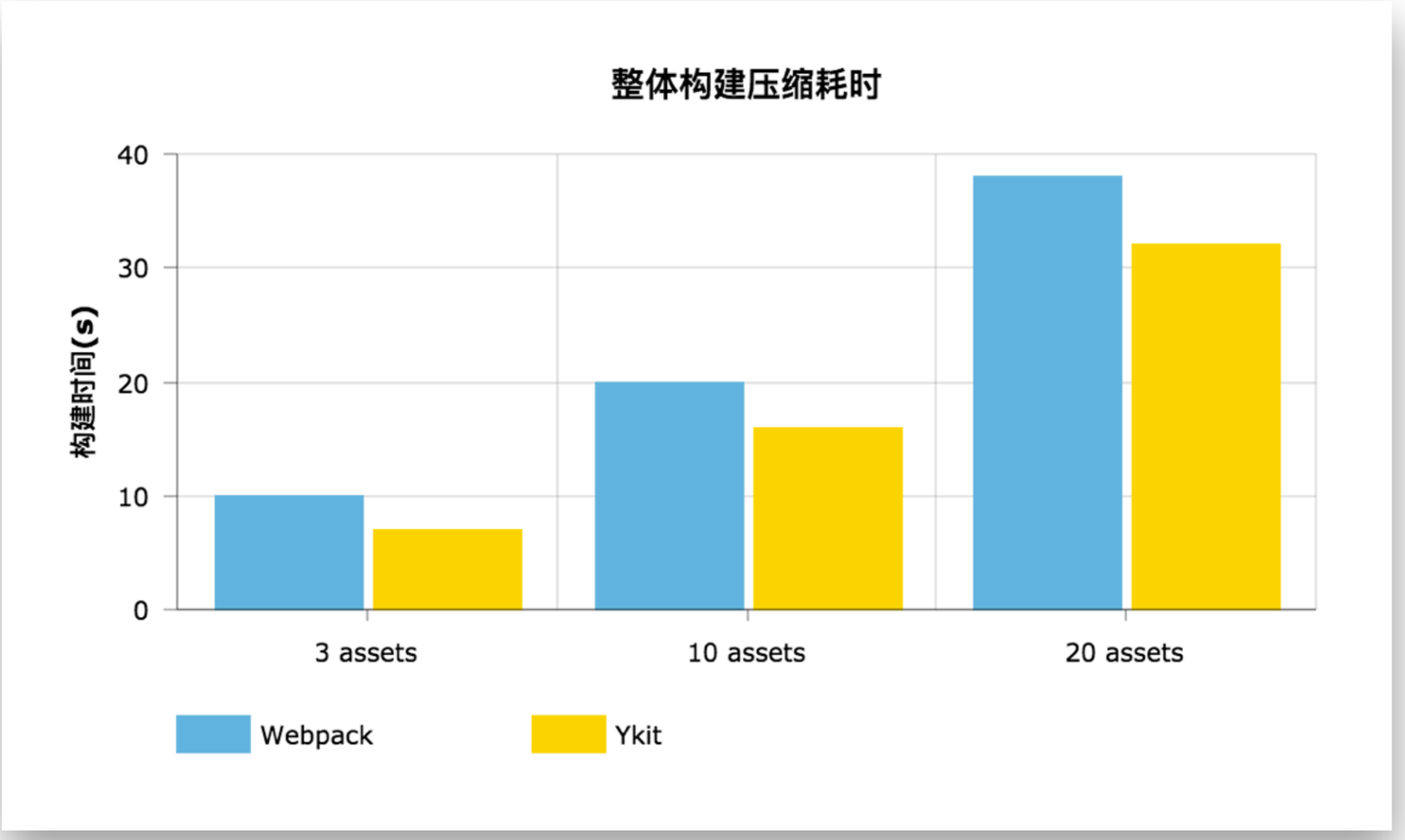

本地服务首次启动时间
打包时间(s)
40
30
20
10
0
3 assets
10 assets
20 assets
Webpack
Ykit

压缩优化

- 使用进程集群，一个进程压缩一个资源
- 防止 Webpack 内存溢出崩溃



效果对比


整体构建压缩耗时
构建时间(s)
40
30
20
10
0
3 assets
10 assets
20 assets
Webpack
Ykit

你觉得，复杂么？

3、更了不起的Node.js

一定有你不知道的应用场景

- 网站（如express/koa等）
- im即时聊天(socket.io)
- api（移动端，pc，h5）
- HTTP Proxy（淘宝、Qunar、腾讯、百度都有）
- 前端构建工具(grunt/gulp/bower/webpack/fis3…)
- 写操作系统（NodeOS）
- 跨平台打包工具（PC端的electron、nw.js，比如钉钉PC客户端、微信小程序IDE、微信客户端，移动的cordova，即老的Phonegap，还有更加有名的一站式开发框架ionicframework）
- 命令行工具（比如cordova、shell.js）
- 反向代理（比如anyproxy，node-http-proxy）
- 编辑器Atom、VSCode等

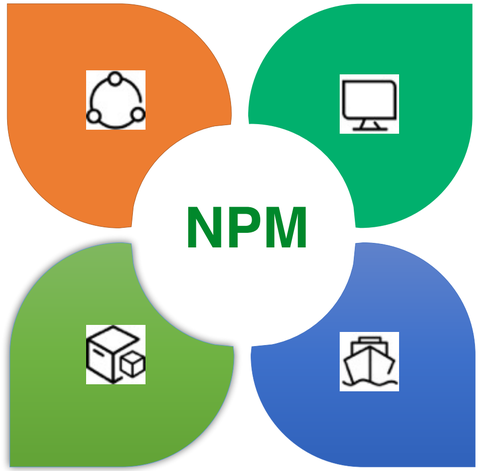
更了不起的Node.js

跨平台 01

前端（web+h5）
移动端（hybrid）
PC端

Node后端 02

核心特性、Web应用、Api
rpc、测试、部署、最佳实践
微服务、厂商支持



03 前端

react\vue\angular
应用实践
架构

04 工具

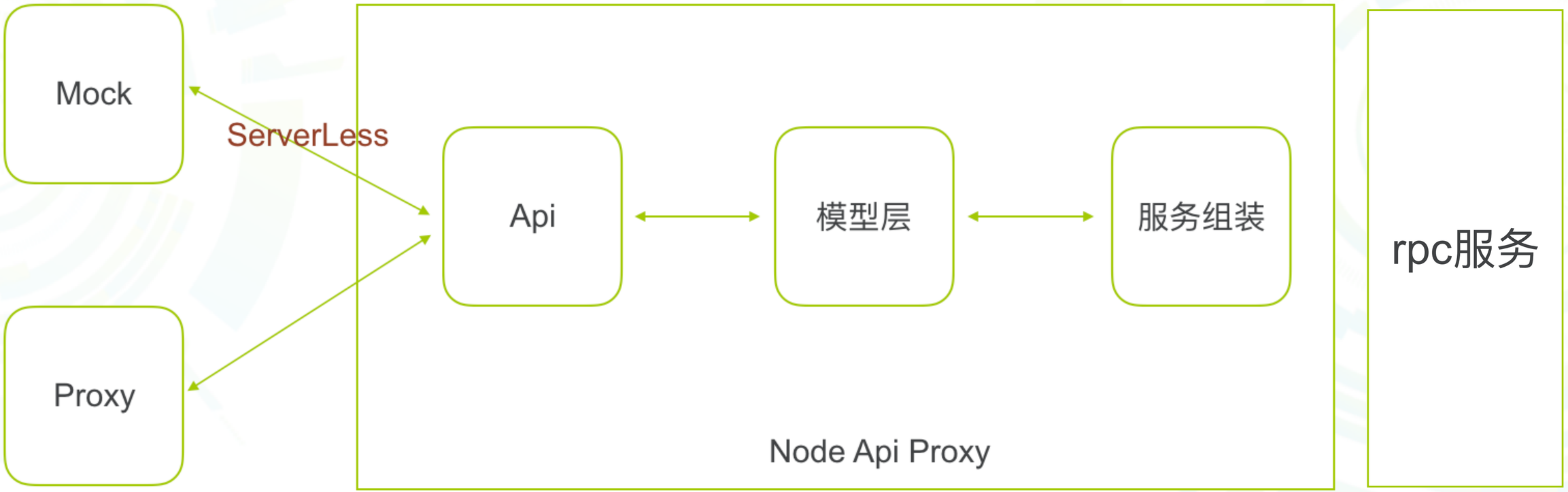
各种预编译、构建工具
webpack/gulp、工程化，
hack技巧、npm等

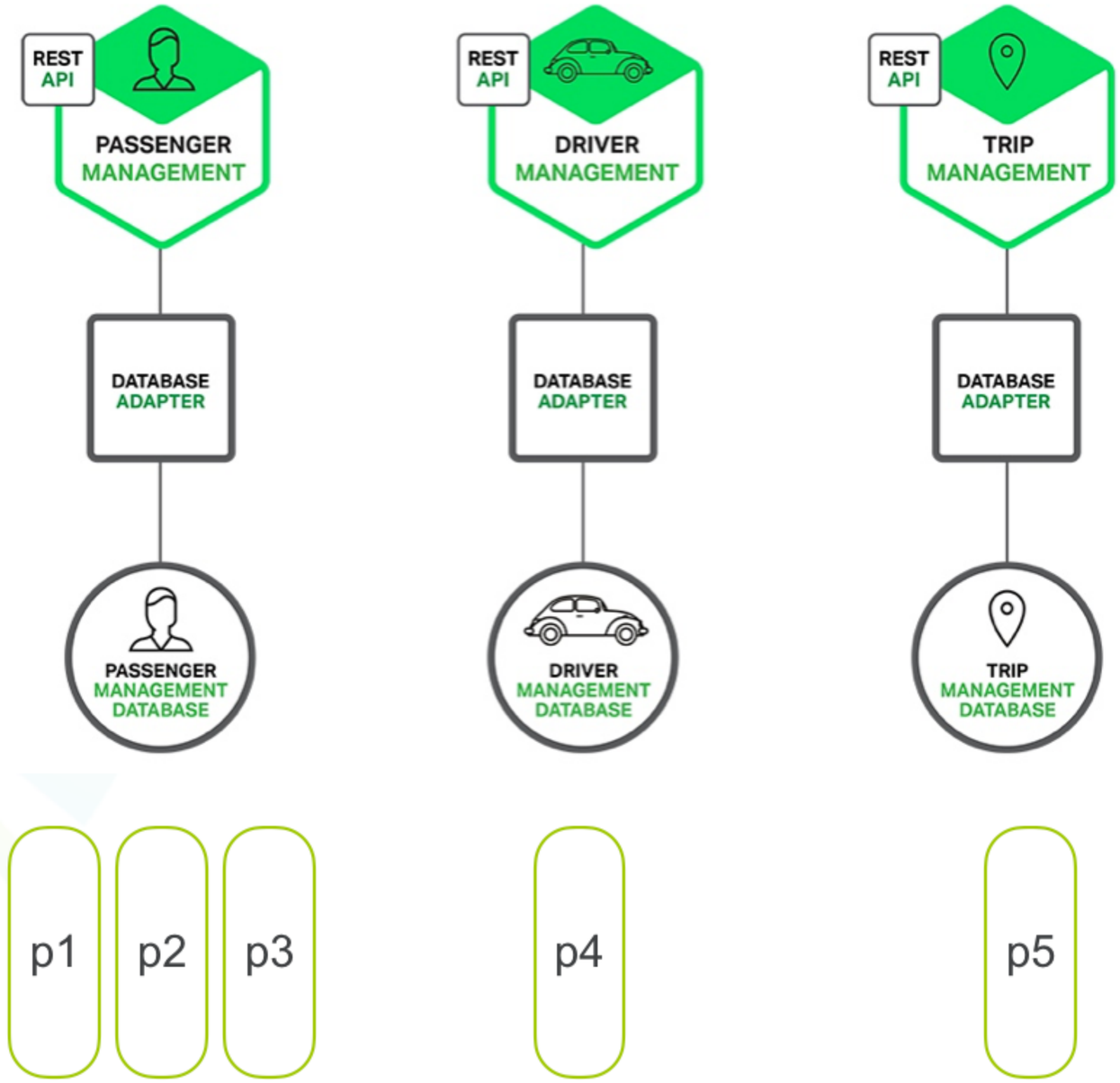
做合适的事儿

大前端 -> Api -> oltp(node | java | go)
-> olap（dw && bi）
-> data mining -> ai

利益与时间

贯穿开发全过程

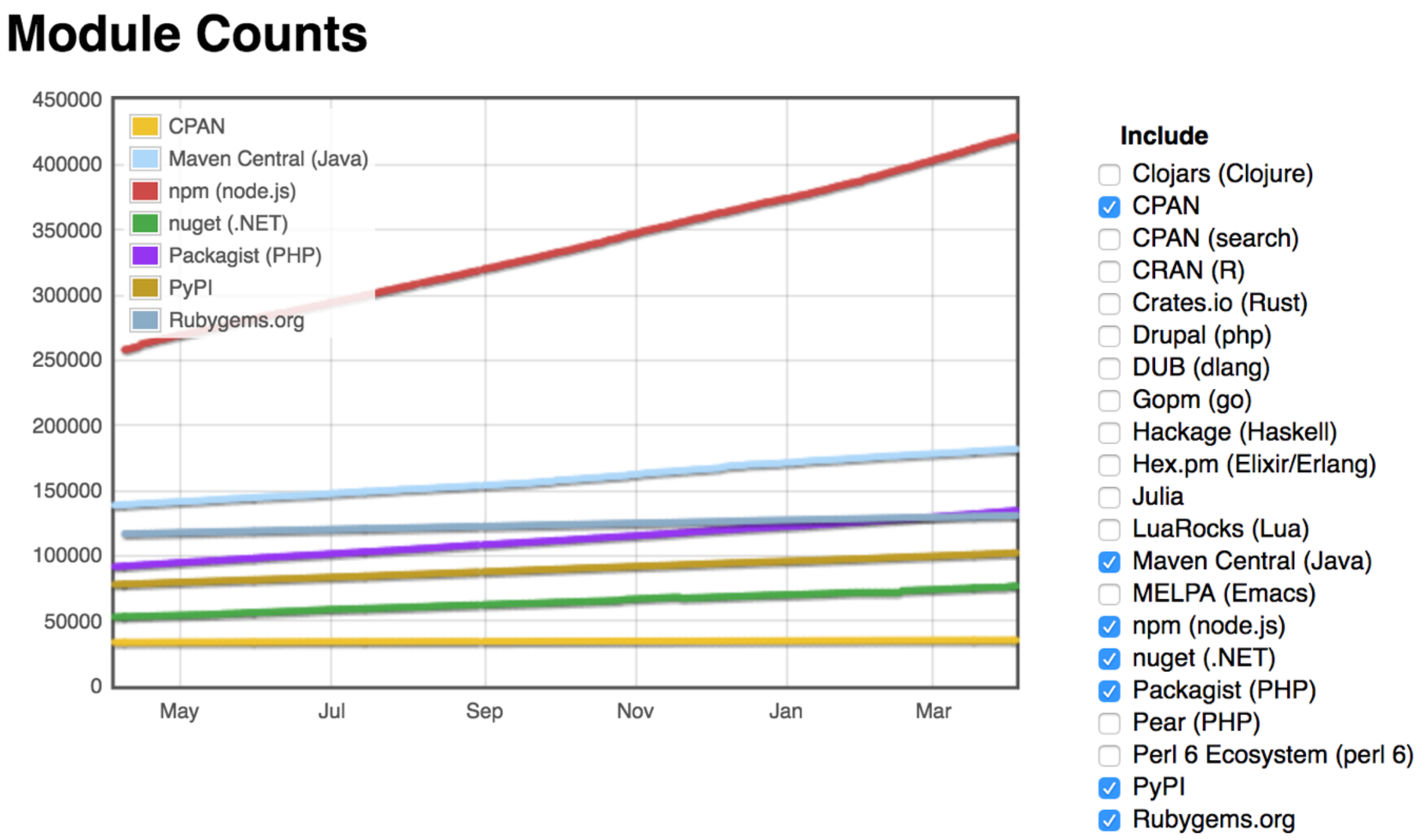




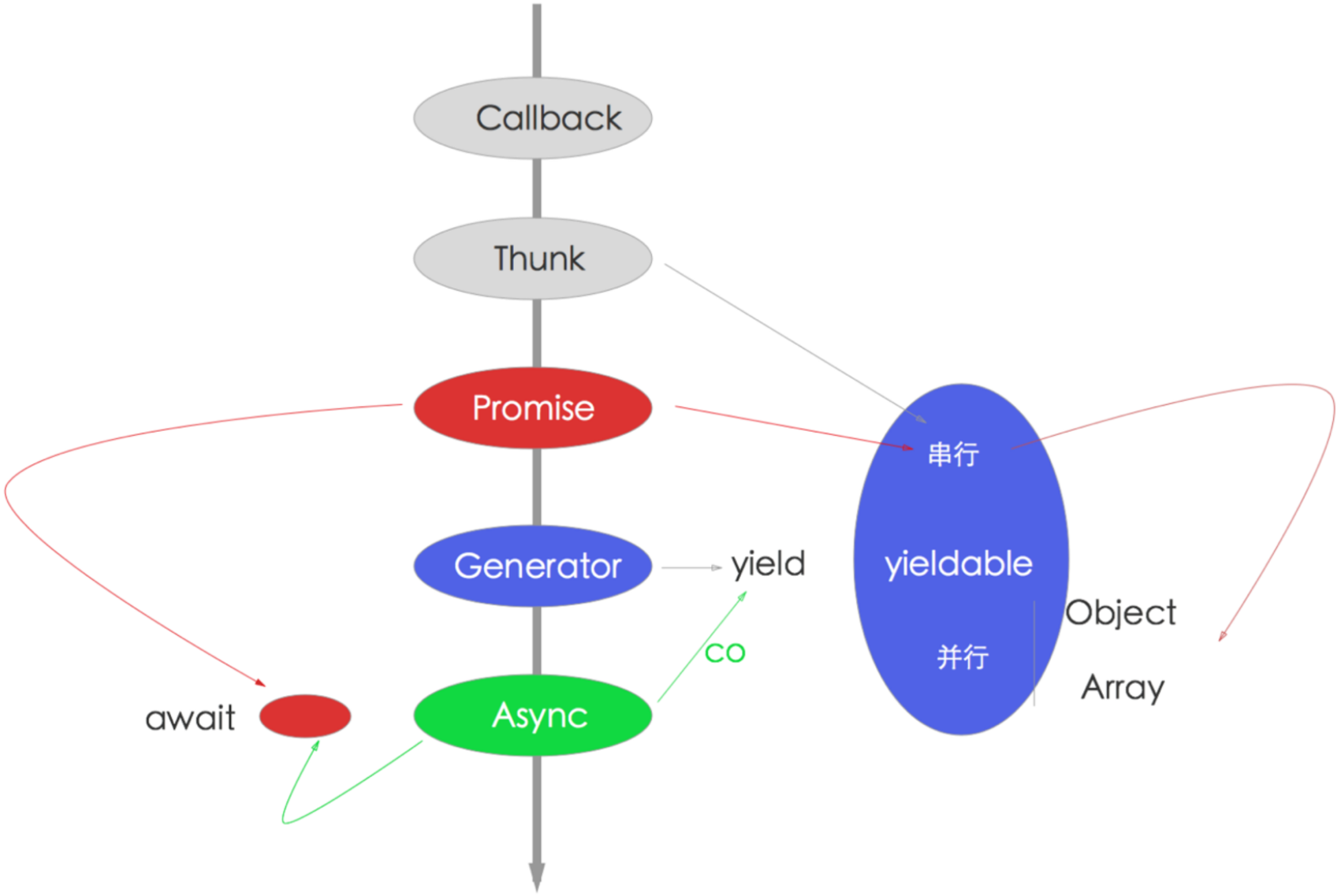
页面即服务

- 页面不会特别复杂，更容易把控
- 页面崩溃对整体来说影响绩效
- 页（必须配合devops）

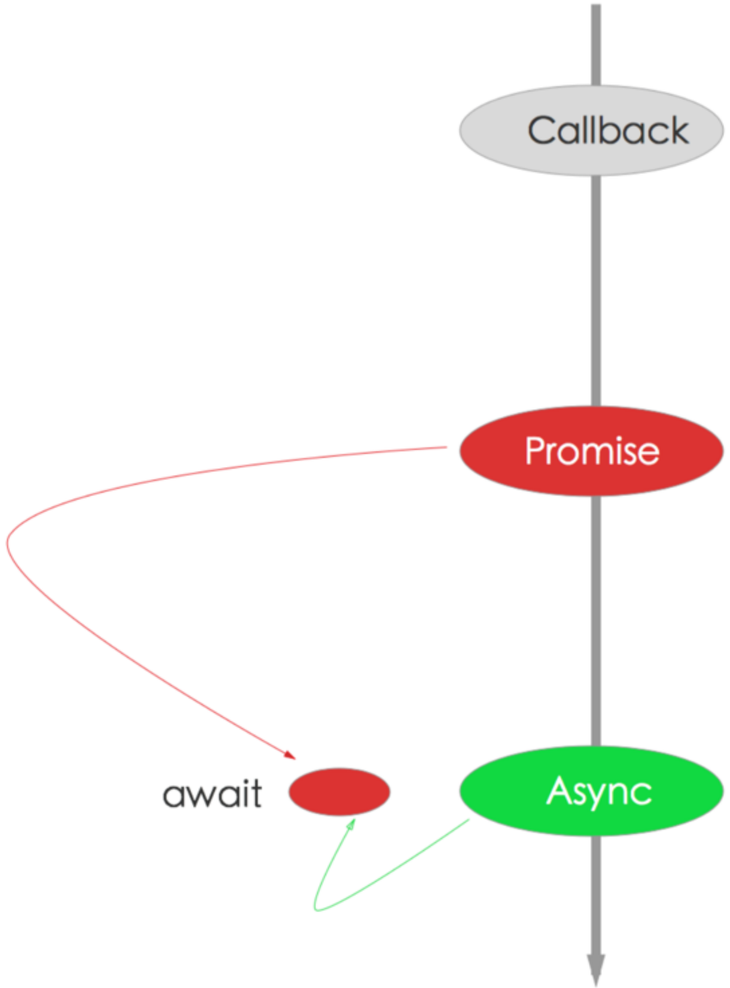
npm上45w+的模块，叫生态



异步流程控制


Callback
Thunk
Promise
Generator
yield
Async
co
await
串行
yieldable
并行
Object
Array

学习重点



同步代码

```
exports.list = async (ctx, next) => {
  console.log(ctx.method + ' /students => list, query: ' +
  try {
    let students = await Student.getAllAsync();

    await ctx.render('students/index', {
      students : students
    })
  } catch (err) {
    return ctx.api_error(err);
  }
};
```

调试

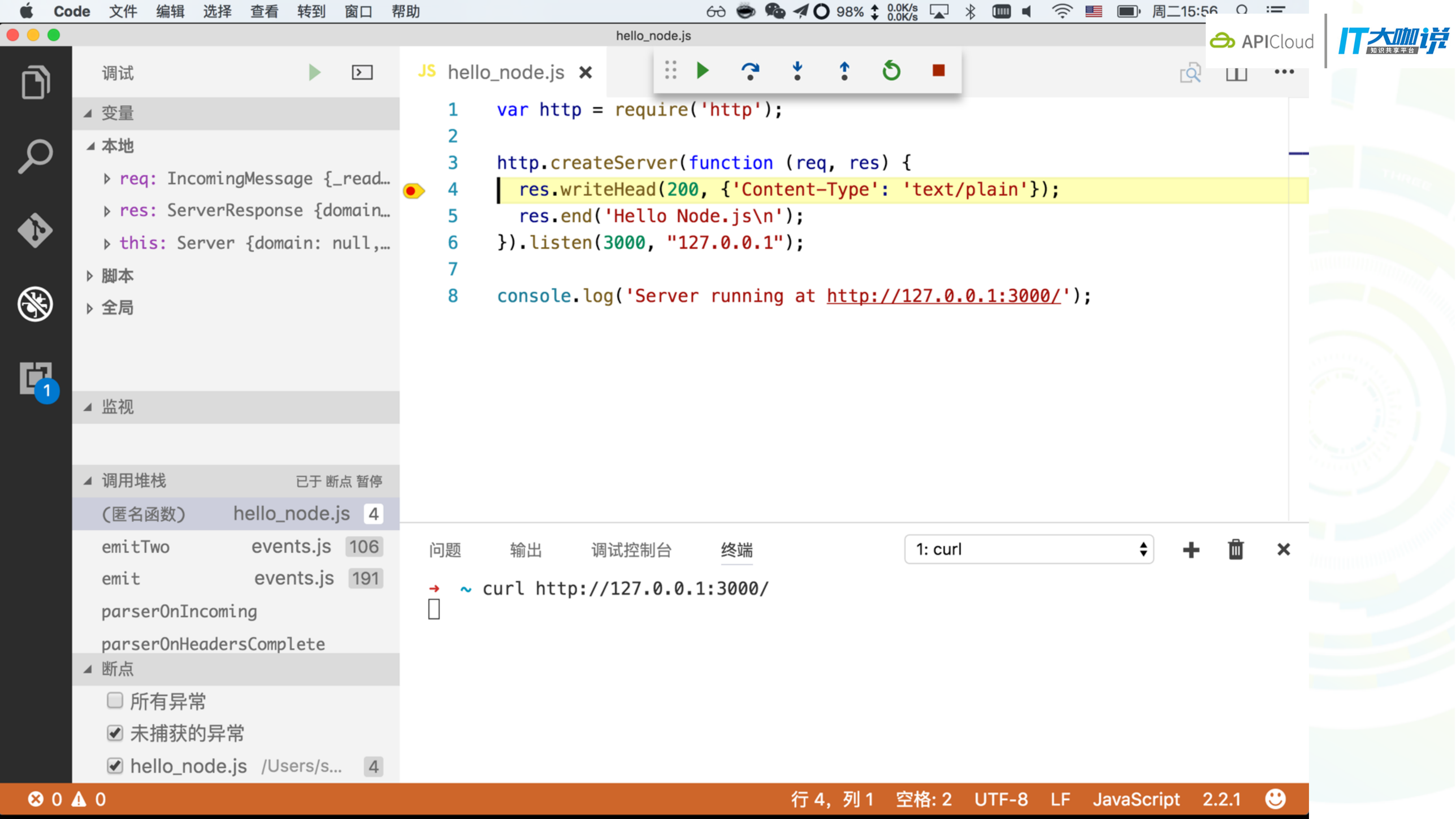
VSCode

单个文件调试

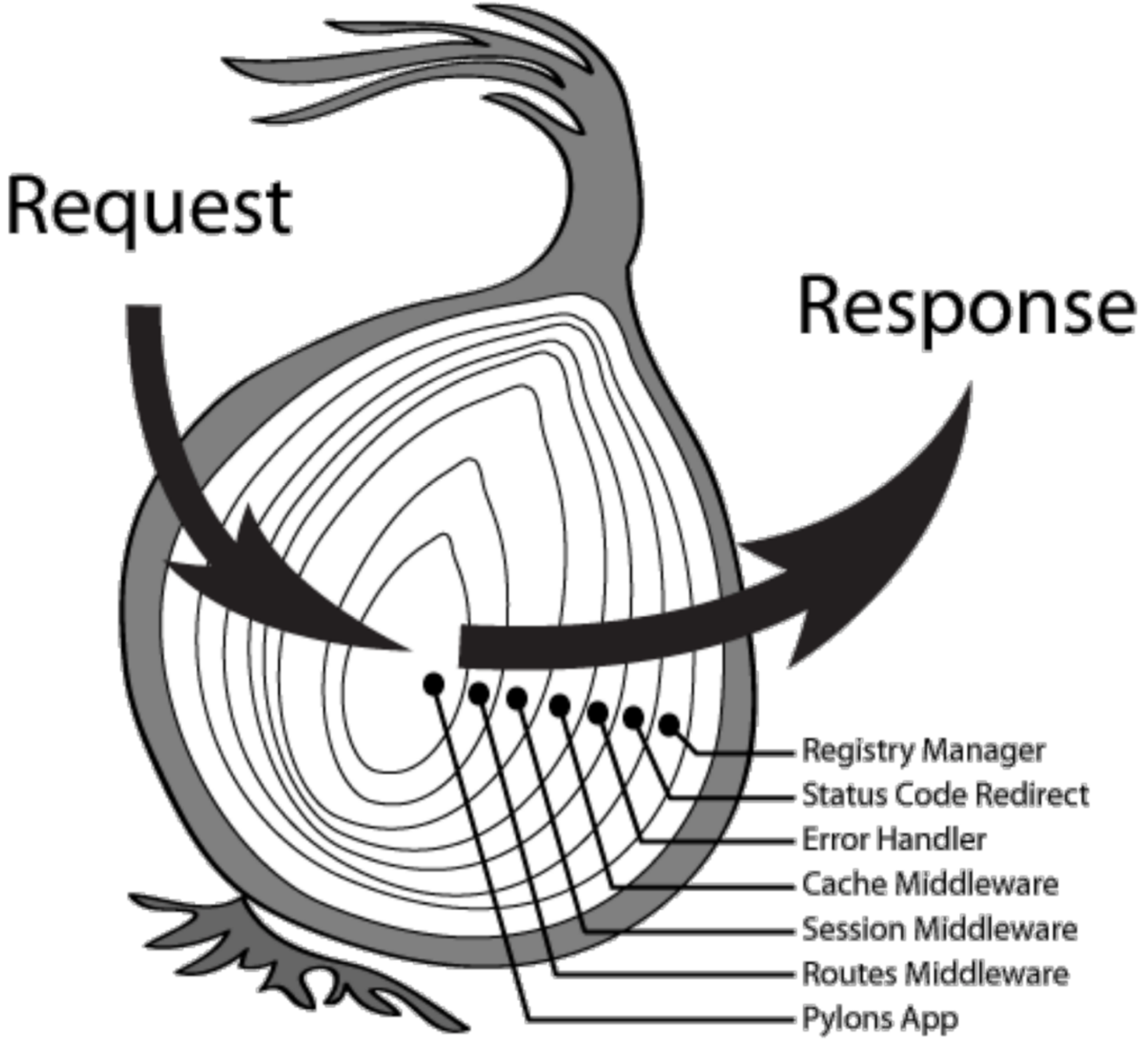
固定文件调试

远程调试

跨进程调试


Code 文件 编辑 选择 查看 转到 窗口 帮助
hello_node.js
调试
变量
本地
req: IncomingMessage {_read…
res: ServerResponse {domain…
this: Server {domain: null,…
脚本
全局
监视
调用堆栈 已于 断点 暂停
(匿名函数) hello_node.js 4
emitTwo events.js 106
emit events.js 191
parserOnIncoming
parserOnHeadersComplete
断点
所有异常
未捕获的异常
hello_node.js /Users/s… 4
hello_node.js
1 var http = require('http');
2
3 http.createServer(function (req, res) {
4 res.writeHead(200, {'Content-Type': 'text/plain'});
5 res.end('Hello Node.js\n');
6 }).listen(3000, "127.0.0.1");
7
8 console.log('Server running at http://127.0.0.1:3000/');
问题 输出 调试控制台 终端
1: curl
→ ~ curl http://127.0.0.1:3000/
0 0
行 4，列 1 空格: 2 UTF-8 LF JavaScript 2.2.1
APICloud
IT大咖说

框架名称		点评
Express	简	
Derby.js && Meteor	同	看上去更简
Sails、Total	面	的一个标志
MEAN.js	面	蹭了热点
Hapi和Restfy	面	可独立分类
ThinkJS	面	路，对于
Koa	考	



Code 文件 编辑 选择 查看 转到 调试 窗口 帮助 周六15:52

index.js — koa2-demo

资源管理器

打开的编辑器
- 左侧
  - index.js routes
  - app.js
- 右侧
  - app.js
  - index.js routes

KOA2-DEMO
- bin
- public
- routes
  - index.js
  - users.js
- views
- app.js
- package.json

index.js

```
const Koa = require('koa')
const app = new Koa()
const views = require('koa-views')
const json = require('koa-json')
const onerror = require('koa-onerror')
const bodyparser = require('koa-bodyparser')
const logger = require('koa-logger')

const index = require('./routes/index')
const users = require('./routes/users')

// error handler
onerror(app)

// middlewares
app.use(bodyparser({
  enableTypes:['json', 'form', 'text']
}))
app.use(json())
app.use(logger())
app.use(require('koa-static')(__dirname + '/

app.use(views(__dirname + '/views', {
  extension: 'pug'
}))

// logger
app.use(async (ctx, next) => {
  const start = new Date()
  await next()
  const ms = new Date() - start
  console.log(`${ctx.method} ${ctx.url} - ${
```

app.js

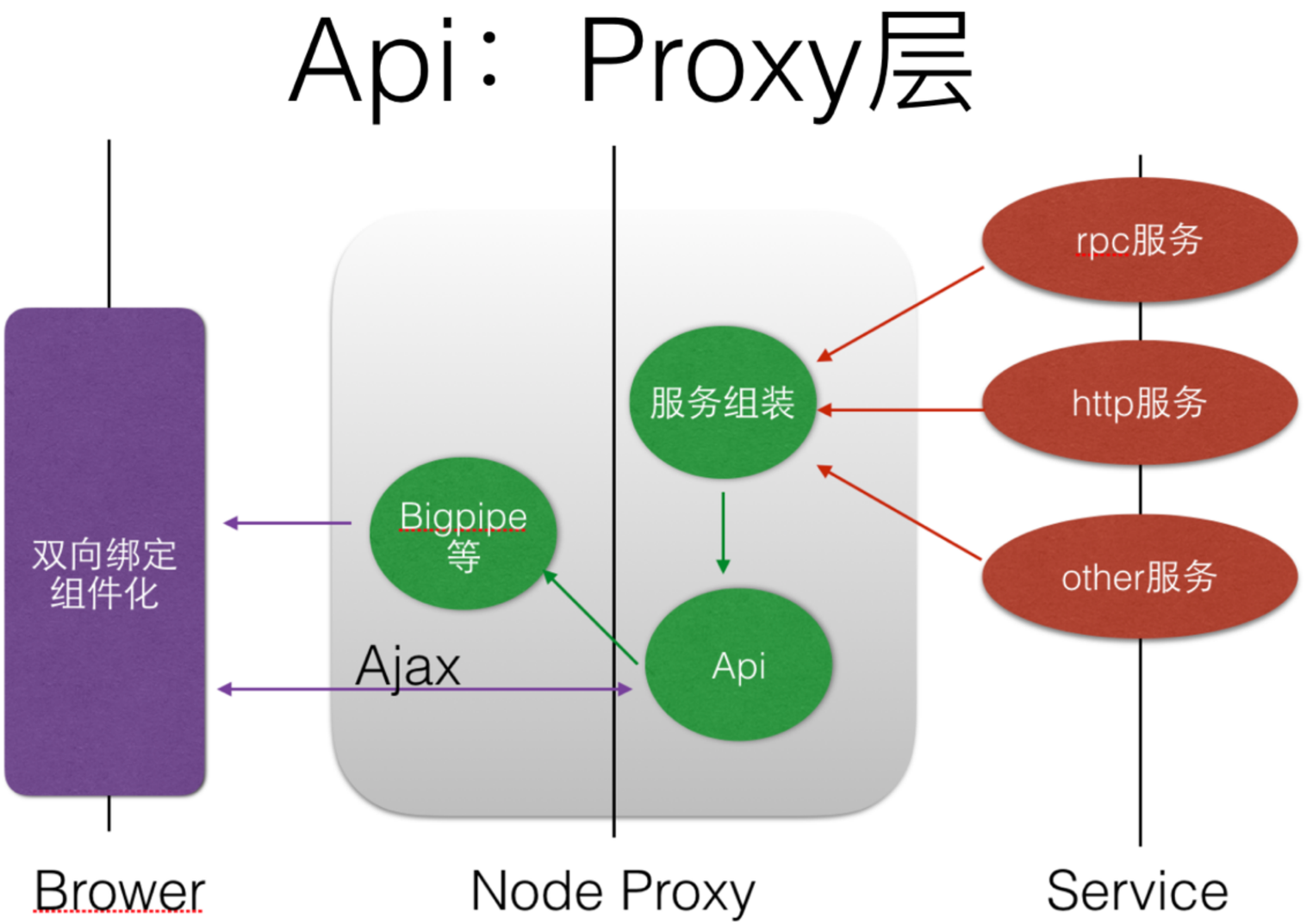
```
const router = require('koa-router')()

router.get('/', async (ctx, next) => {
  await ctx.render('index', {
    title: 'Hello Koa 2!'
  })
})

router.get('/string', async (ctx, next) => {
  ctx.body = 'koa2 string'
})

router.get('/json', async (ctx, next) => {
  ctx.body = {
    title: 'koa2 json'
  }
})

module.exports = router
```

0 0 行 1，列 1 空格: 2 UTF-8 LF JavaScript

4、我眼中Node全栈

假设前面有2辆车，你必须撞一个，你选哪个？

有了Node更美好


Api：Proxy层
双向绑定
组件化
Bigpipe
等
Ajax
服务组装
Api
rpc服务
http服务
other服务
Brower
Node Proxy
Service

成为更好的自己，将终身学习进行到底

减少沟通成本

可攻可受

享受快乐

闲时要有吃紧的心思，忙里要有偷闲的乐趣

你会的越多解决越容易

少抱怨，多思考，未来更美好

谢谢观看
THANKS