理解现代 Web 开发

Understanding Modern Web Development

dexteryy @ Flipboard

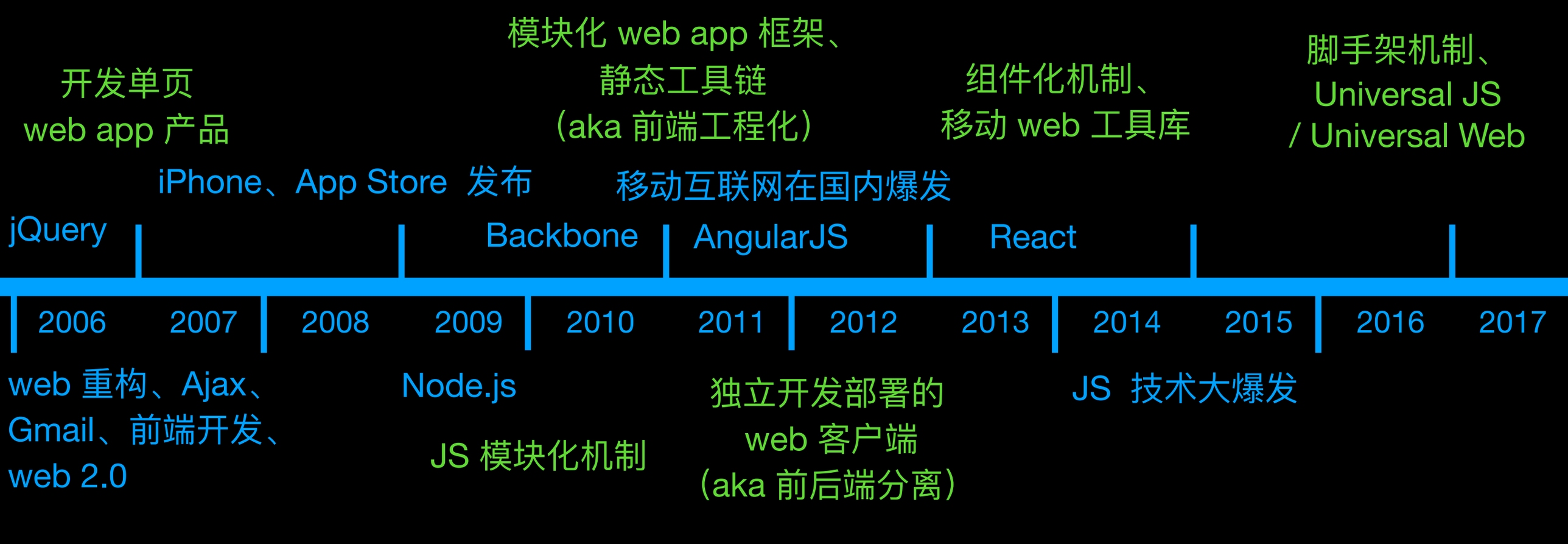
Agenda

1. 介绍
2. 背景
3. 问题和方法
4. 导读
5. 实践

介绍我自己

dexteryy / 杨扬

- 11 年前是 TRPG 游戏网站站长、美漫汉化组创始人和 web 设计师
- 此后一直是 JS hacker 和 web app 开发者（开发 + 设计 + 产品 + 运营）
- 先后在土豆、豆瓣、Flipboard 开发 JS 基础设施和创新产品
- 努力推动 JS 技术和 web 技术的演进：



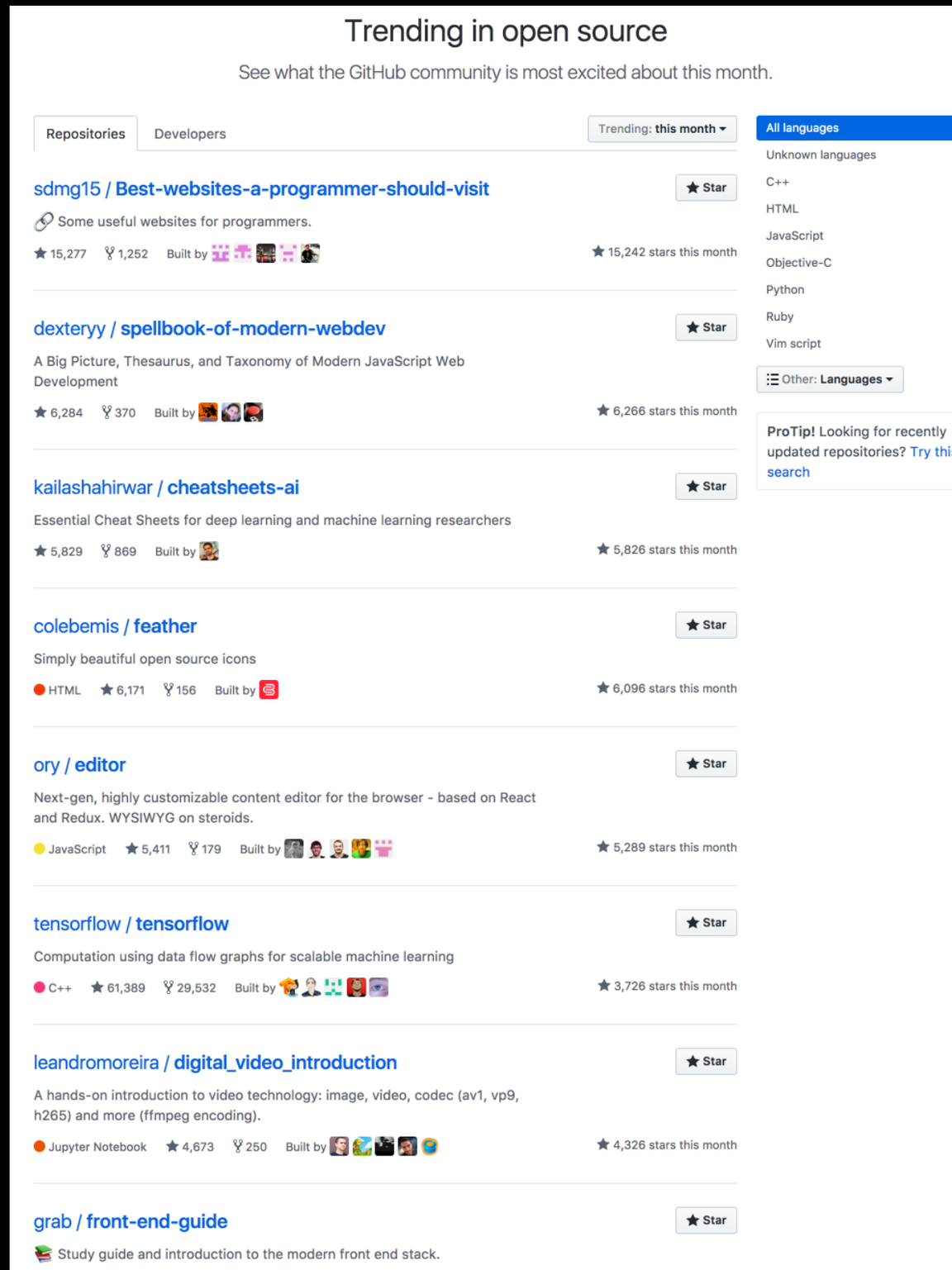
Spellbook of Modern Web Dev

现代 web 开发者的魔法书

https://github.com/dexteryy/spellbook-of-modern-webdev

社区反馈

- 在 Github 发布四天 4000 颗星
- Github Trending：
  - 第一天：（全语言类）日榜第一
  - 第二天：（全语言类）周榜第二
  - 第三天：（全语言类）月榜第六
  - 一周后：（全语言类）周榜第一月榜第二



社区反馈

推荐：

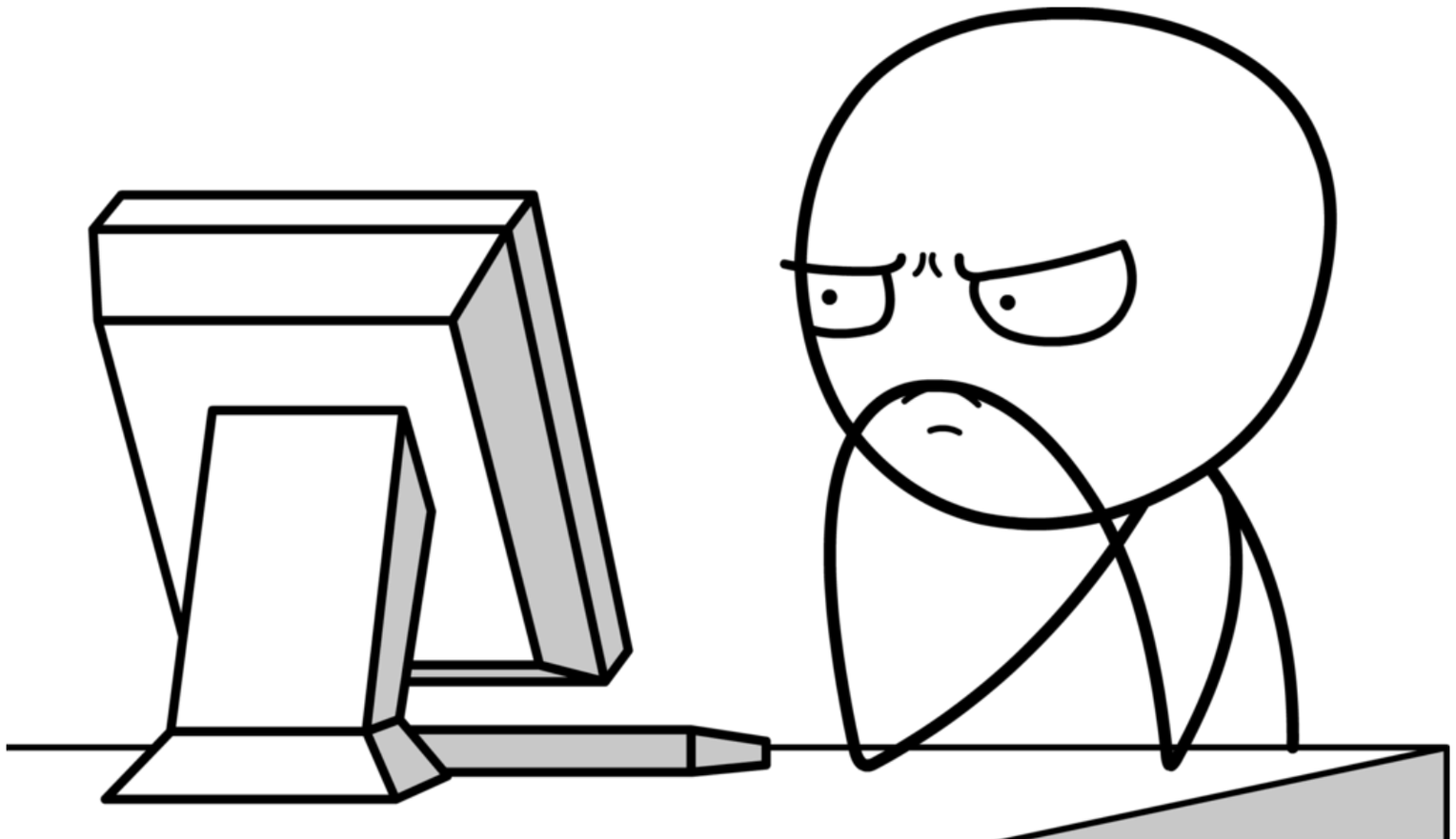
- JS 之父（Brendan Eich）
- 德国阮一峰（Dr. Axel Rauschmayer）

社区反馈

推荐：

- Smashing Magazine、CSS-Tricks、Changelog、O’Reilly Media、……
- freeCodeCamp 约稿（至今还没写过任何文章作介绍和推广）

但是，我有一个感受不知当讲不当讲...



『与世隔绝』的国内社区

来自国内开发者的反馈很少很少...

satóru 2017-06-13 11:36:13

少见的不靠人海战术挤进 trending 的中国仓库，哈哈

『与世隔绝』的国内社区

Cloud Services (Global)

- Compute
  - FaaS / Serverless / WebHook
    - AWS Lambda / Google Cloud Functions
    - webtask / hook.io
    - Graphcool Functions
    - Amazon API Gateway
  - PaaS
    - See *Tooling > Workflow* > *Deployment* > *DevOps*
  - CaaS
    - Amazon ECS / Google Container Engine
- Storage
  - Object Storage
    - Amazon S3 / Google Cloud Storage
    - imgix
  - DBaaS
    - In-Memory Key-Value NoSQL - Amazon ElastiCac
      - Redis - Compose / Redise Cloud / Heroku Re
    - Document NoSQL - Amazon DynamoDB / Google
      - MongoDB - Compose / mLab / MongoDB Atla
      - CouchDB - Couchbase / Cloudant
    - Wide Column NoSQL - Google Bigtable

Cloud Services (China)

> The evil twins inside the Great Firewall of China

- Compute
  - FaaS / Serverless / WebHook
    - 阿里云-函数计算 / 腾讯云-无服务器云函数 SCF
    - 阿里云-API 网关
  - PaaS
    - See *Tooling > Workflow* > *Deployment* > *DevOps*
  - CaaS
    - 阿里云-容器服务 / 腾讯云-容器服务 CCS / DaoClou
- Storage
  - Object Storage
    - 阿里云-对象存储 OSS / 腾讯云-对象存储 COS
  - DBaaS
    - In-Memory Key-Value NoSQL
      - Redis - 阿里云-云数据库 Redis 版 / 腾讯云-云
    - Document NoSQL
      - MongoDB - 阿里云-云数据库 MongoDB版 / 腾
    - Wide Column NoSQL - 阿里云-表格存储 OTS
      - HBase - 阿里云-云数据库 HBase 版 / 腾讯云-
    - SQL
      - PostgreSQL - 阿里云-云数据库 PostgreSQL 版

『与世隔绝』的国内社区

README-zh-CN.md #85

Edit | New issue

Closed **wadekun** opened this issue 8 days ago · 2 comments

wadekun commented 8 days ago

0 First Issue in this repo

e... maybe need a README-zh-CN.md for developer that english is poor , just like me...🙂

👎 1

dexteryy commented 8 days ago • edited — Owner

This document consists of two kinds of content elements: the plain-text category name and the link.

The category names are basically some proper nouns, technical terms and buzzwords which are also used in non-English-speaking regions (like China). The links all point to English-language articles or web pages which are almost impossible to be translated by this project. And It doesn't make much sense to only translate titles of these links.

So I don't think an README-zh-CN.md would be meaningful.

How to keep all translations in sync is also a problem. #7

👍 1

Assignees

No one—assign yourself

Labels

None yet

Milestone

No milestone

Notifications

Unsubscribe

You're receiving notifications because you commented.

3 participants

Lock conversation

『与世隔绝』的国内社区

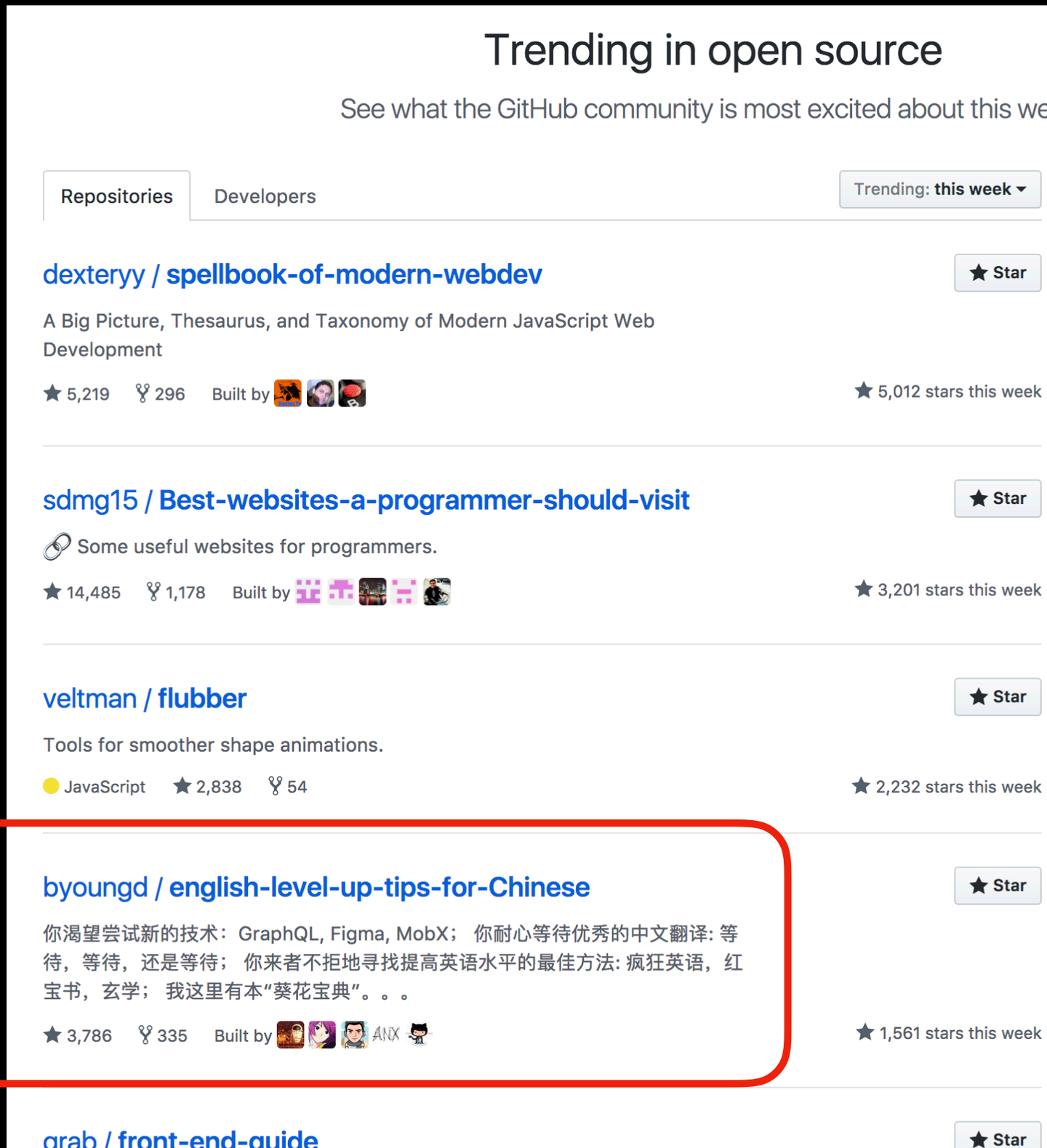
在柏林的几天我和许多认识不认识的人交谈下来，最大的感受就是……中国的开发者在国际社区几乎没有什么存在感。……都表示好奇为什么中国的程序员好像活在另一个宇宙里。……但是点进去一看都在用中文讨论 issue 和写 commit，注释里可能还有很多中文，于是就默默点叉了。……

国内对“自己创建的开源项目”一般更上心，但是据我观察好像这样的自己创建开源项目很少会有国际社区的人参与进来，一个主要的原因是这些项目大部分第一工作语言都是中文，让大部分不会中文的人望而却步，所以最后依然是两个平行世界。

—— Node.js CTC 成员、alinode 工程师张秋怡《柏林纪行》

『与世隔绝』的国内社区

- 语言隔阂才是最高的『墙』
- 翻译是双刃剑



『与世隔绝』的国内社区

- 语言隔阂才是最高的『墙』
- 翻译是双刃剑
- 融入全球社区
  - 二语习得（SLA）：
    有效的海量输入
    日常的频繁使用
  - 阅读原版文章/文档就是
    最好的学习方法

Next Generation JS

- ES6+ Features
  - Overview of ECMAScript 6 features
  - Babel REPL / ES6 New Features Comparison
- Intro to ES6+
  - Dr. Axel Rauschmayer's blog
    - Exploring ES6
    - Exploring ES2016 and ES2017
  - ES6 In Depth
  - Nicholas C. Zakas's Understanding ECMAScript 6
- Re-intro to JS
  - Articles
    - MDN
      - A re-introduction to JavaScript (JS tutorial)
      - Equality comparisons and sameness, Data types and data structures, Closu
        prototype chain
      - Concurrency model and Event Loop, Memory Management
    - Dmitry Soshnikov
      - JavaScript. The Core
      - ECMA-262-3 in detail
        - Execution Contexts, Variable object, This, Scope chain, Functions, Clos
        - OOP: The general theory, ECMAScript implementation,
      - ECMA-262-5 in detail
        - Properties and Property Descriptors, Strict Mode
        - Lexical environments: Common Theory, ECMAScript implementation
      - Notes
        - Equality operators, Default values of parameters
    - Dmitri Pavlutin
      - equality operator, undefined
      - variables hoisting, variables lifecycle
      - declare functions, 'this' keyword
      - three dots
      - array creation, object literals
      - well-known symbols

为什么称作『现代 web 开发』，

而不是『全栈』？

背景

1. 『前端』/『后端』/『全栈』的黑历史
2. 时代变革大幅加速了趋势
3. 新分工和『现代 web 开发』

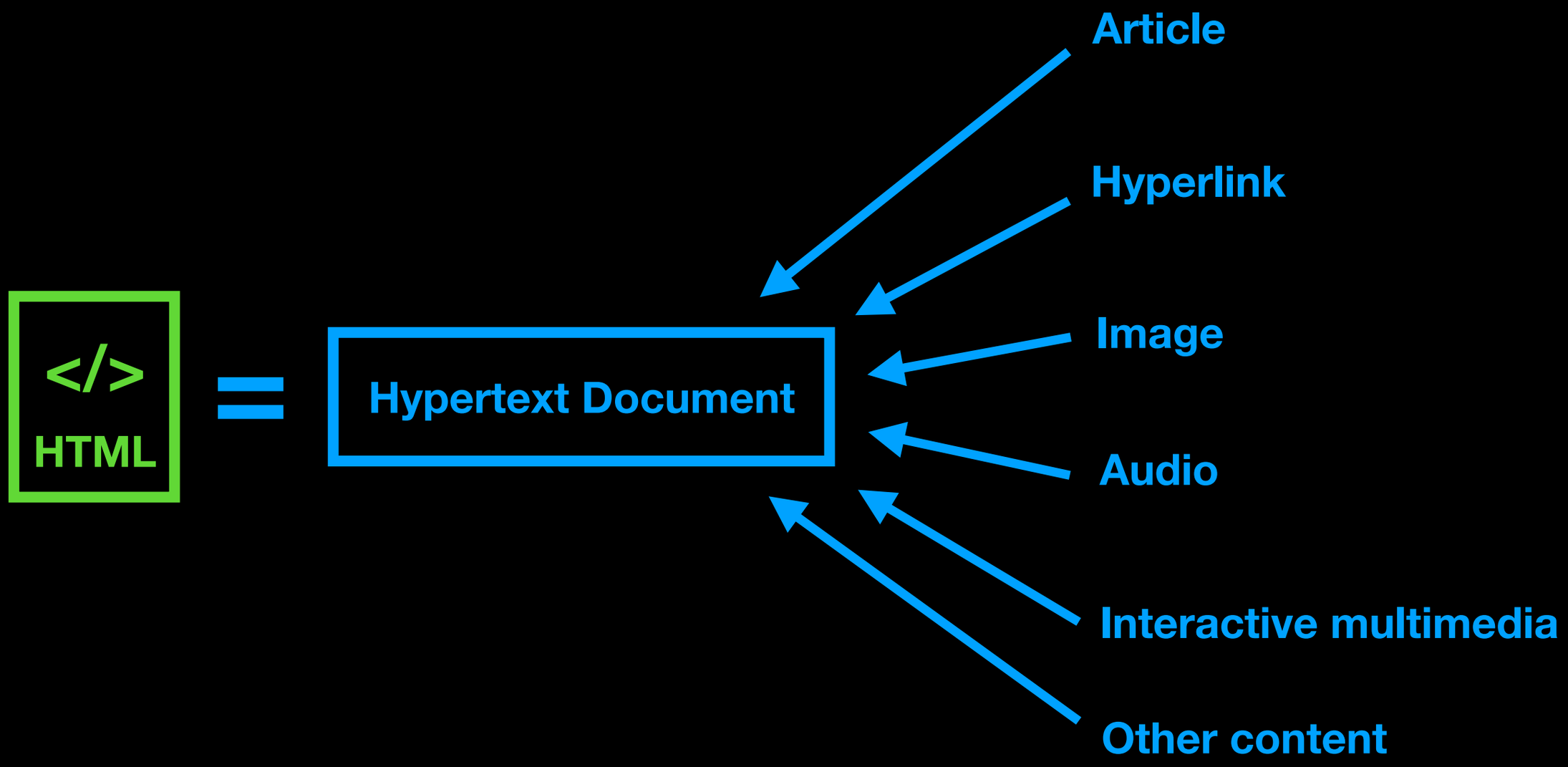
『前端』/『后端』/『全栈』的黑历史

1. 『后端』的来历：

2. 『前端』的来历：

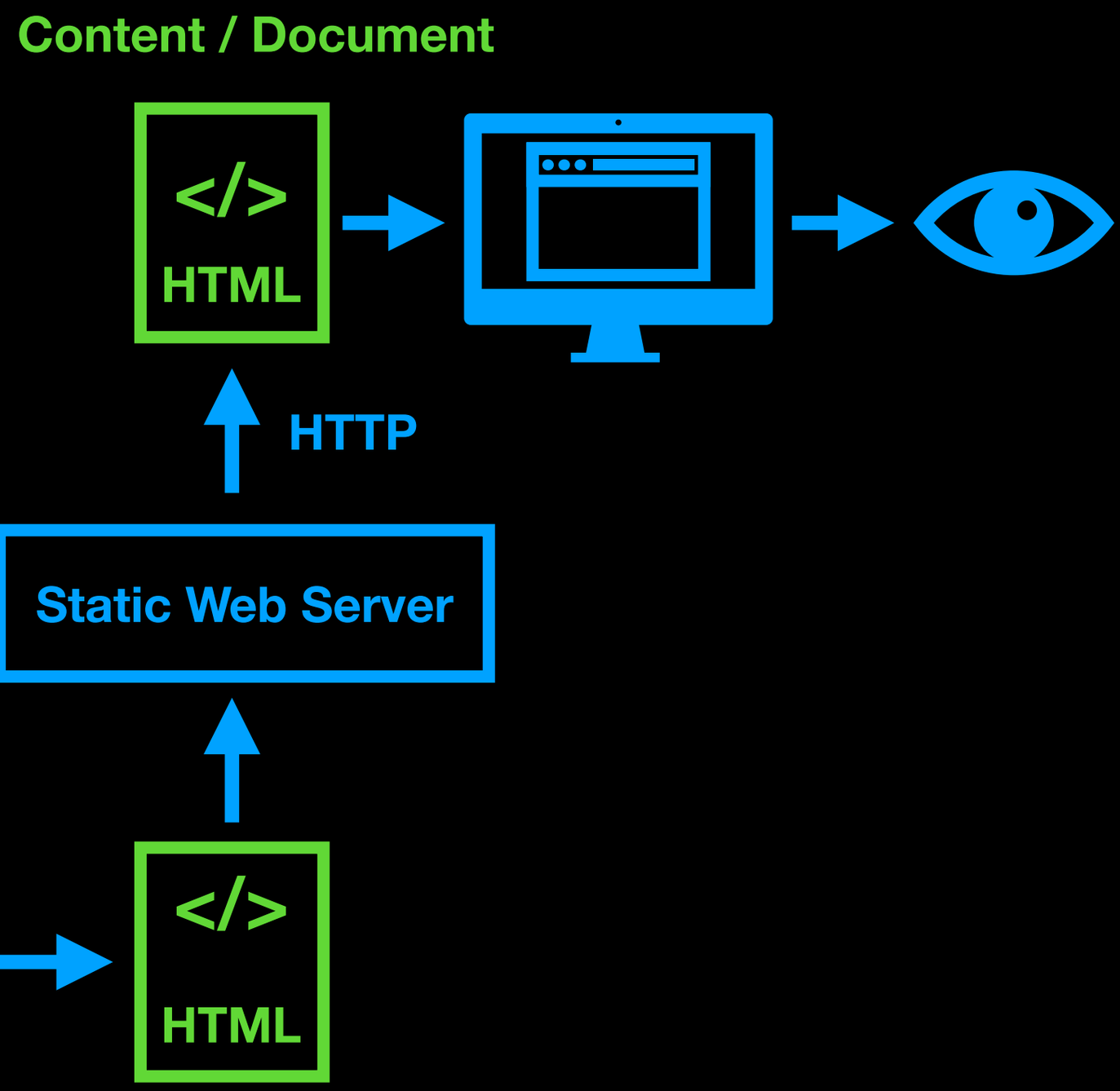
3. 『全栈』的来历：

『后端』的来历：早期 web 开发 = 内容开发



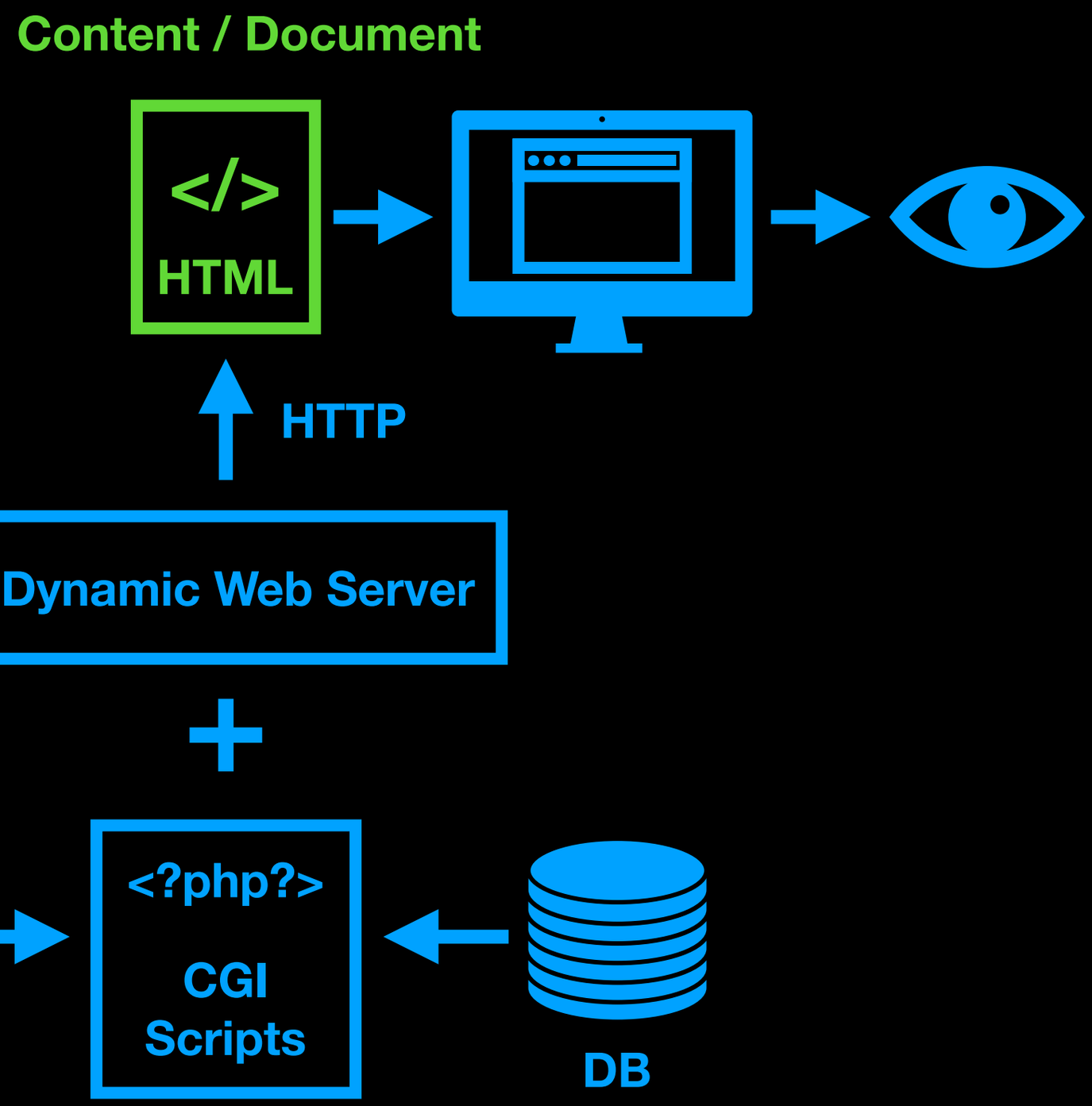
『后端』的来历：早期 web 开发 = 内容开发

- 内容的手工编写



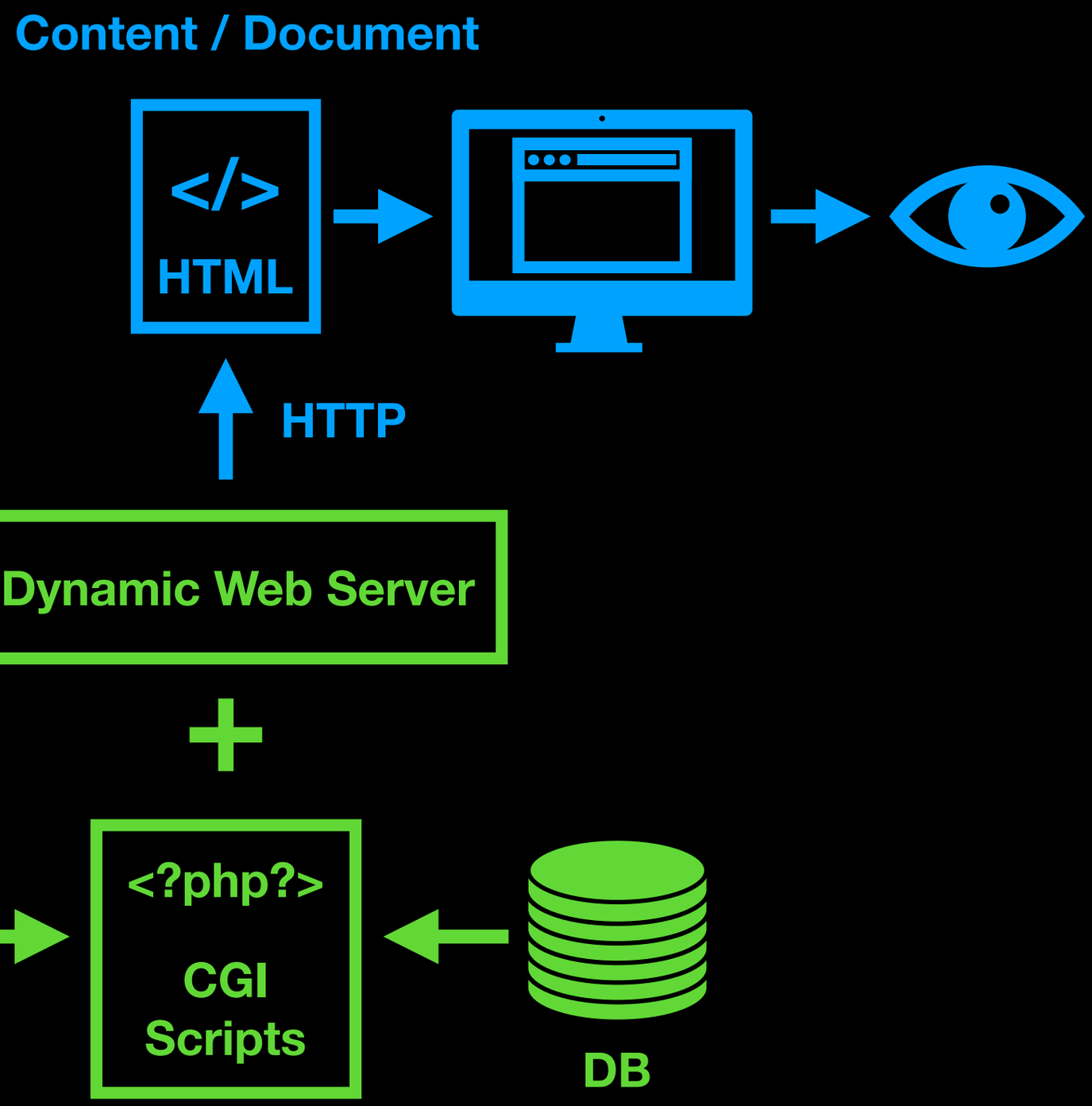
『后端』的来历：早期 web 开发 = 内容开发

- 内容的手工编写
- 内容的动态生成



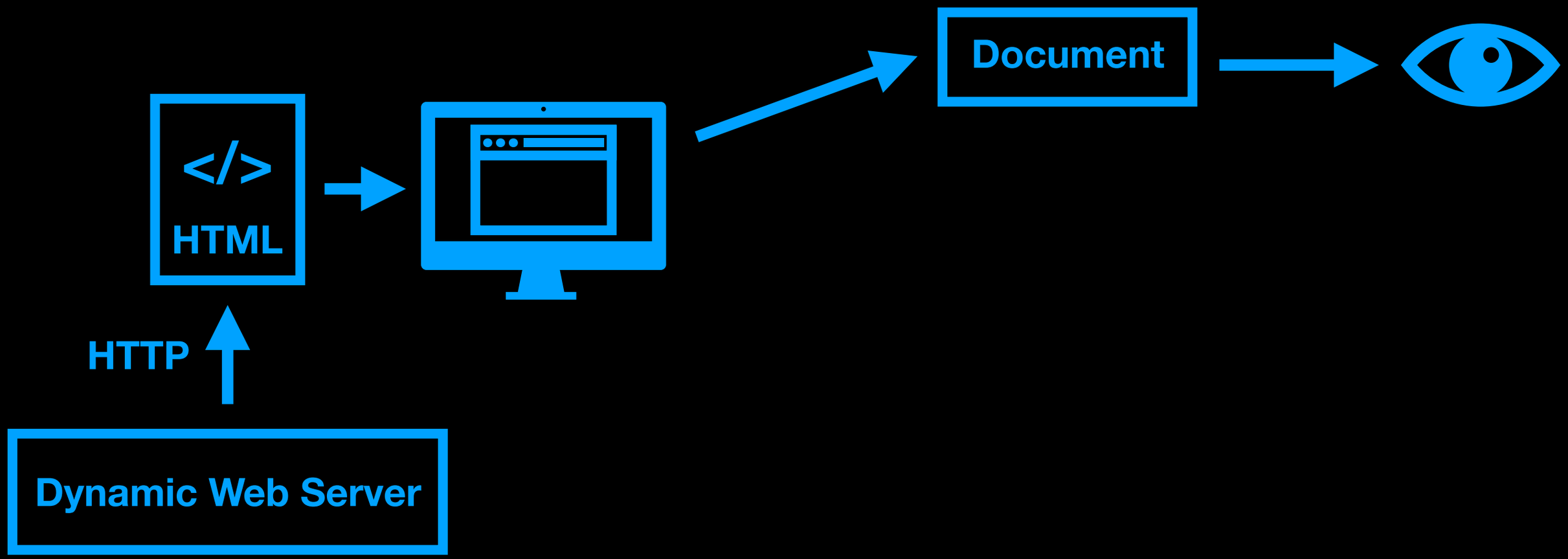
『后端』的来历：早期 web 开发 = 内容开发

- 内容的手工编写
- 内容的动态生成
- 只有『后端』



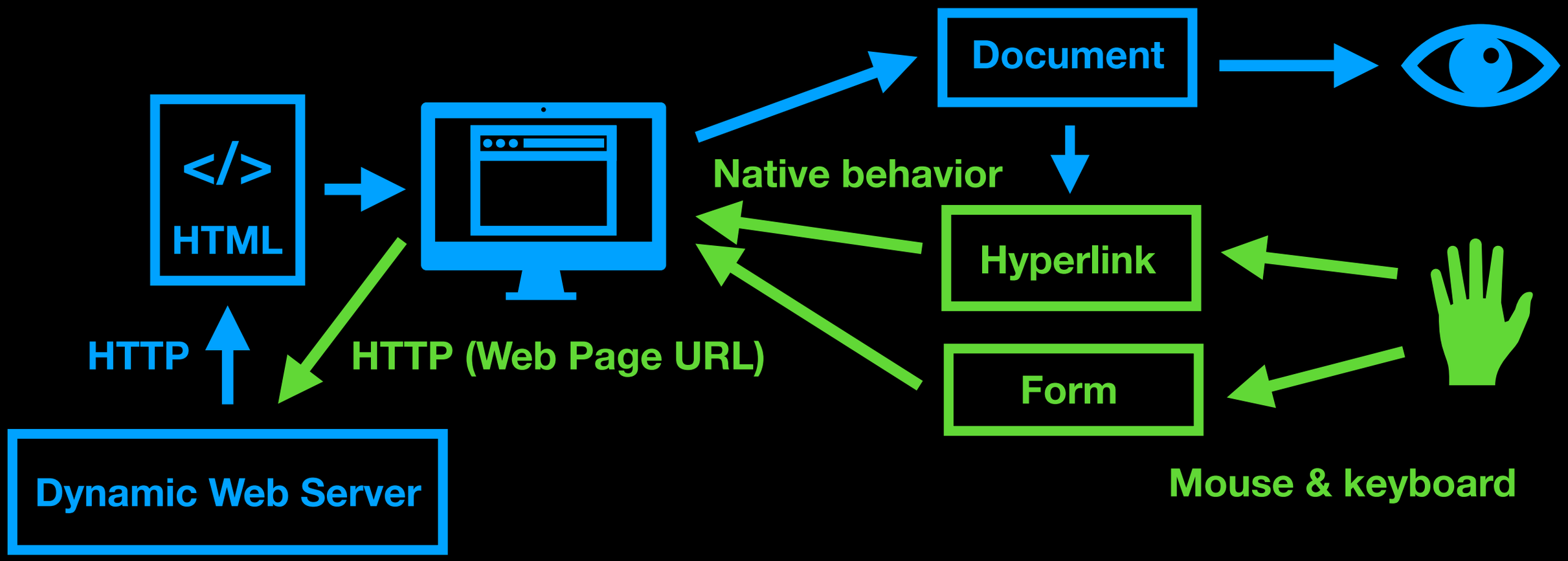
『前端』的来历：传统 web 应用开发

早期 web 应用 = 内容 + 页面级交互



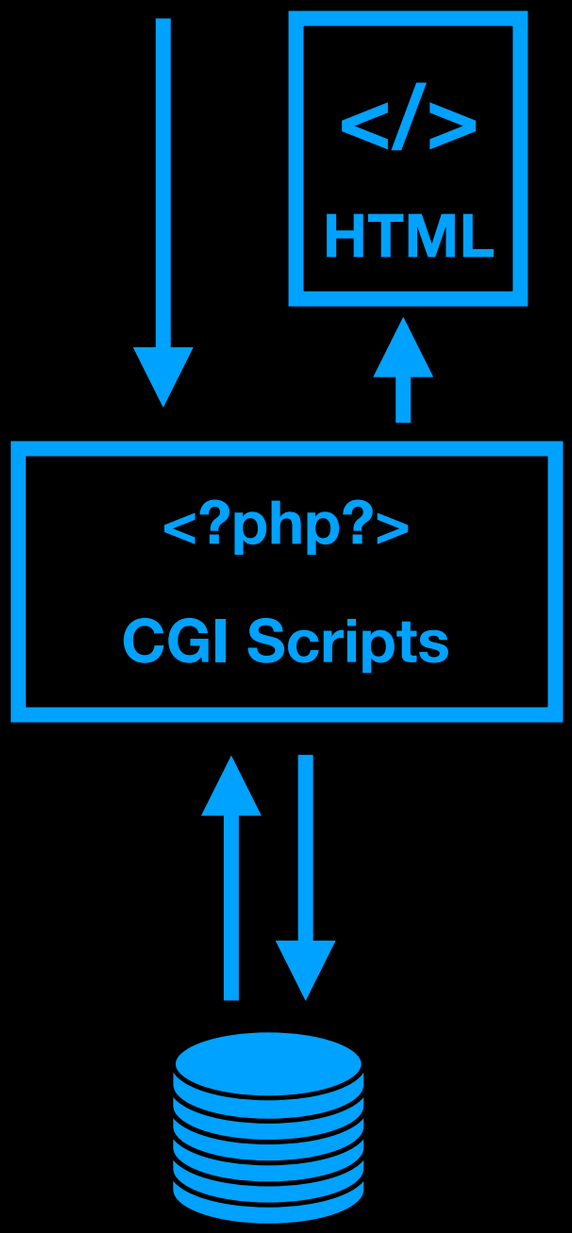
『前端』的来历：传统 web 应用开发

早期 web 应用 = 内容 + 页面级交互



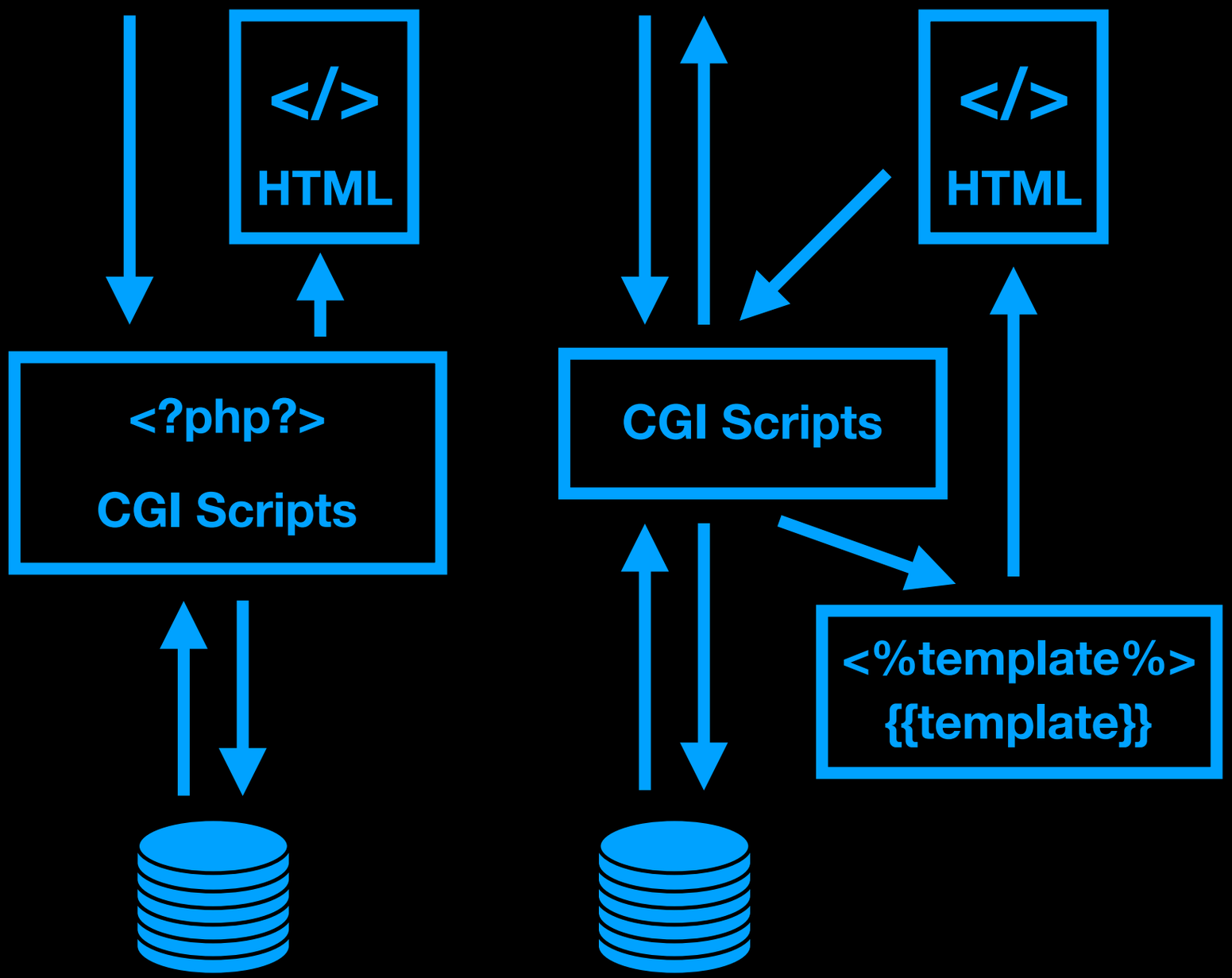
『前端』的来历：传统 web 应用开发

CGI -> PHP -> 纯模板语言 -> 服务器端 MVC



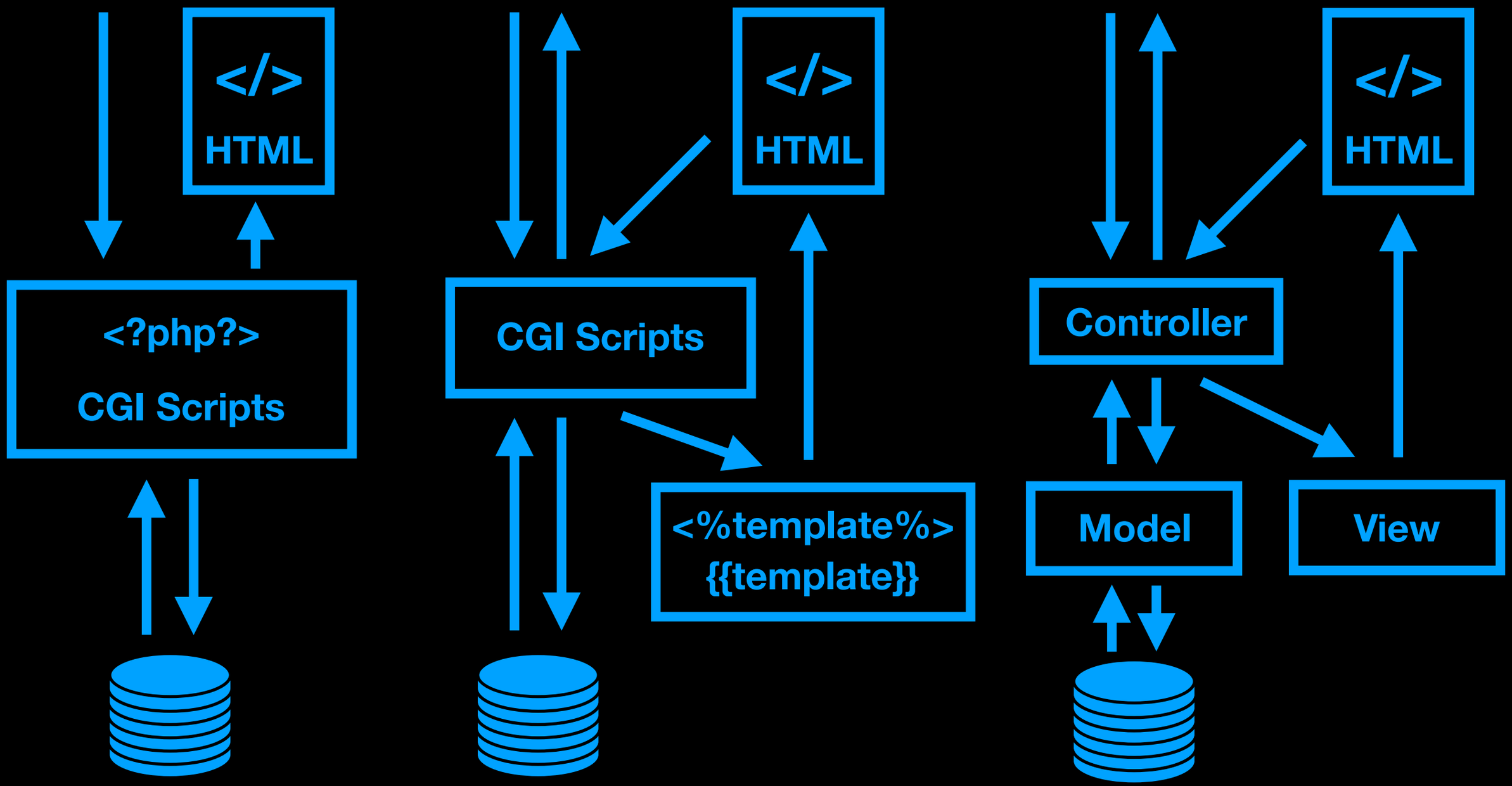
『前端』的来历：传统 web 应用开发

CGI -> PHP -> 纯模板语言 -> 服务器端 MVC



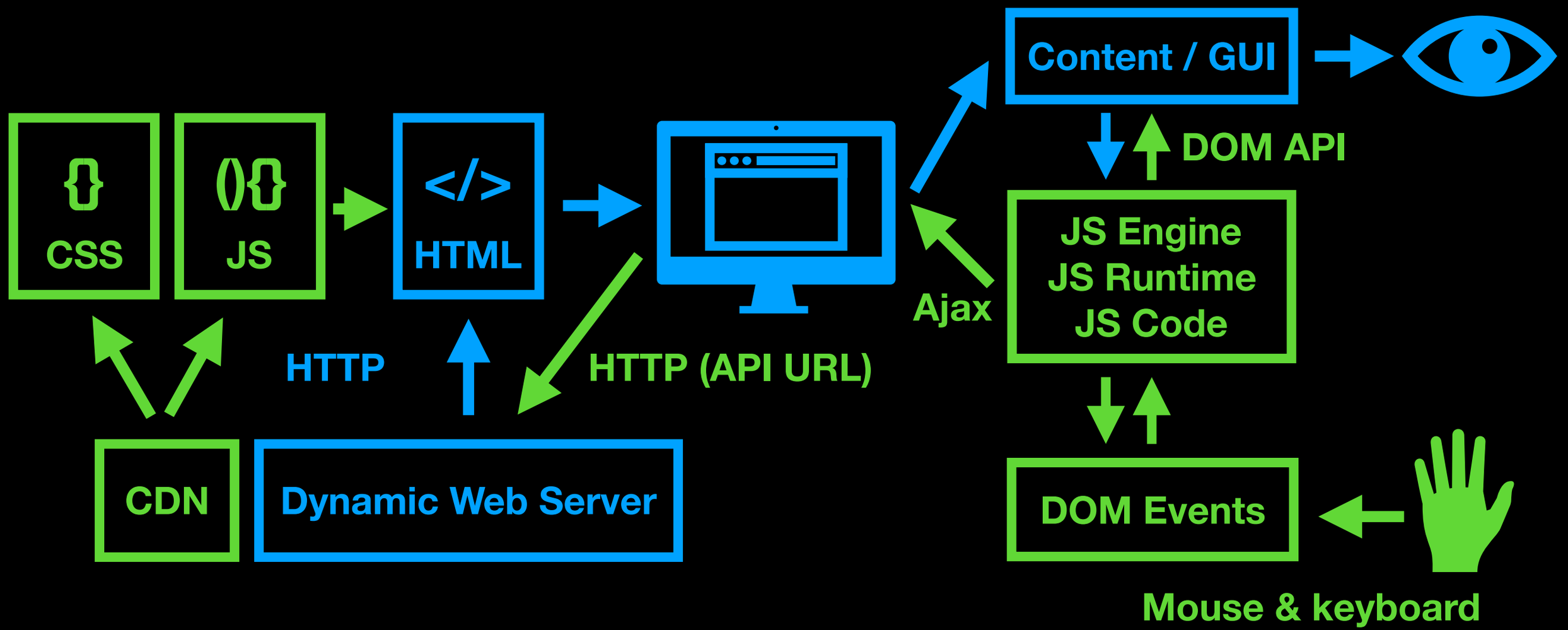
『前端』的来历：传统 web 应用开发

CGI -> PHP -> 纯模板语言 -> 服务器端 MVC



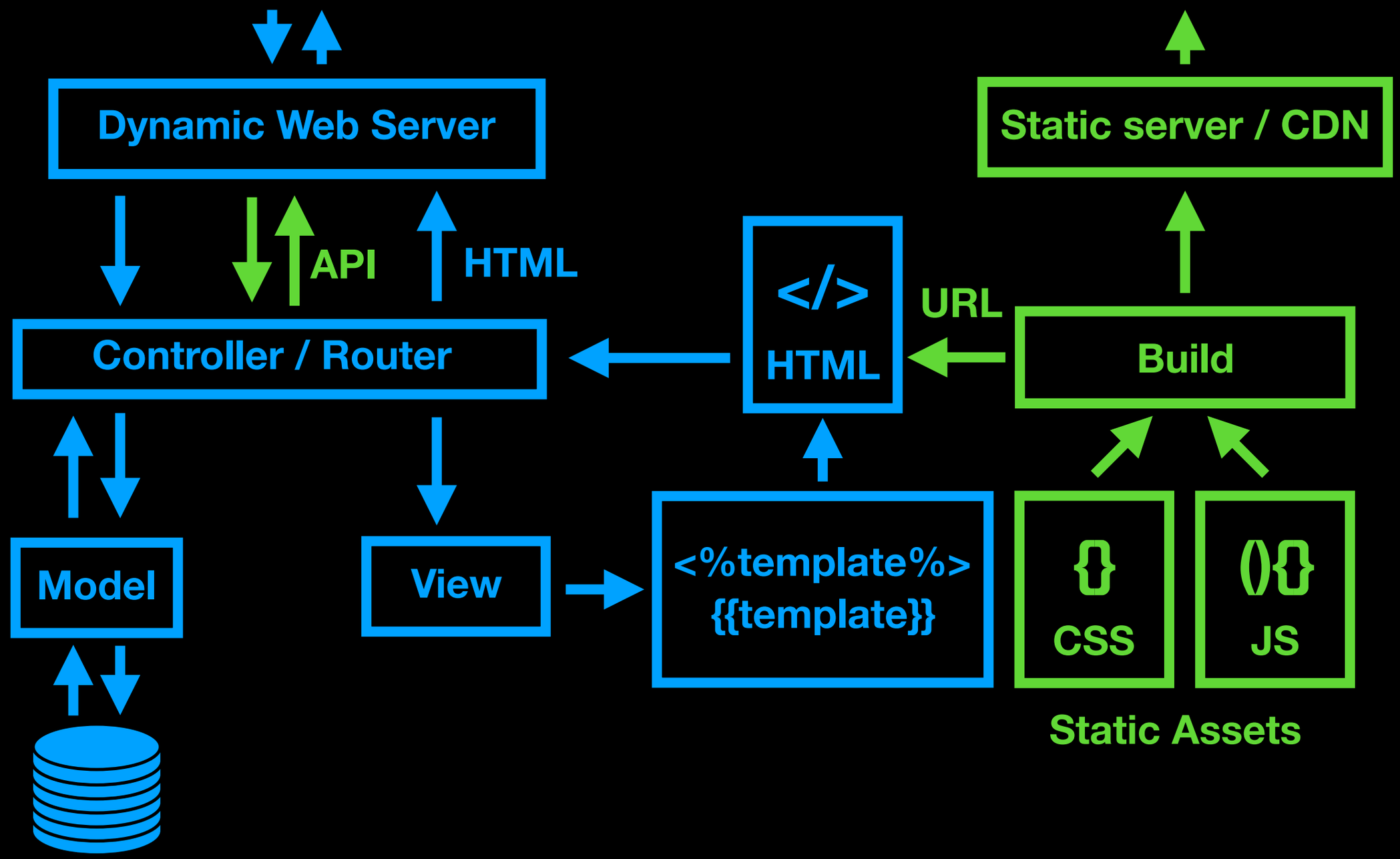
『前端』的来历：传统 web 应用开发

富 web 应用 = 内容/GUI +
API 级交互（Ajax） + 『DHTML』交互（DOM API）



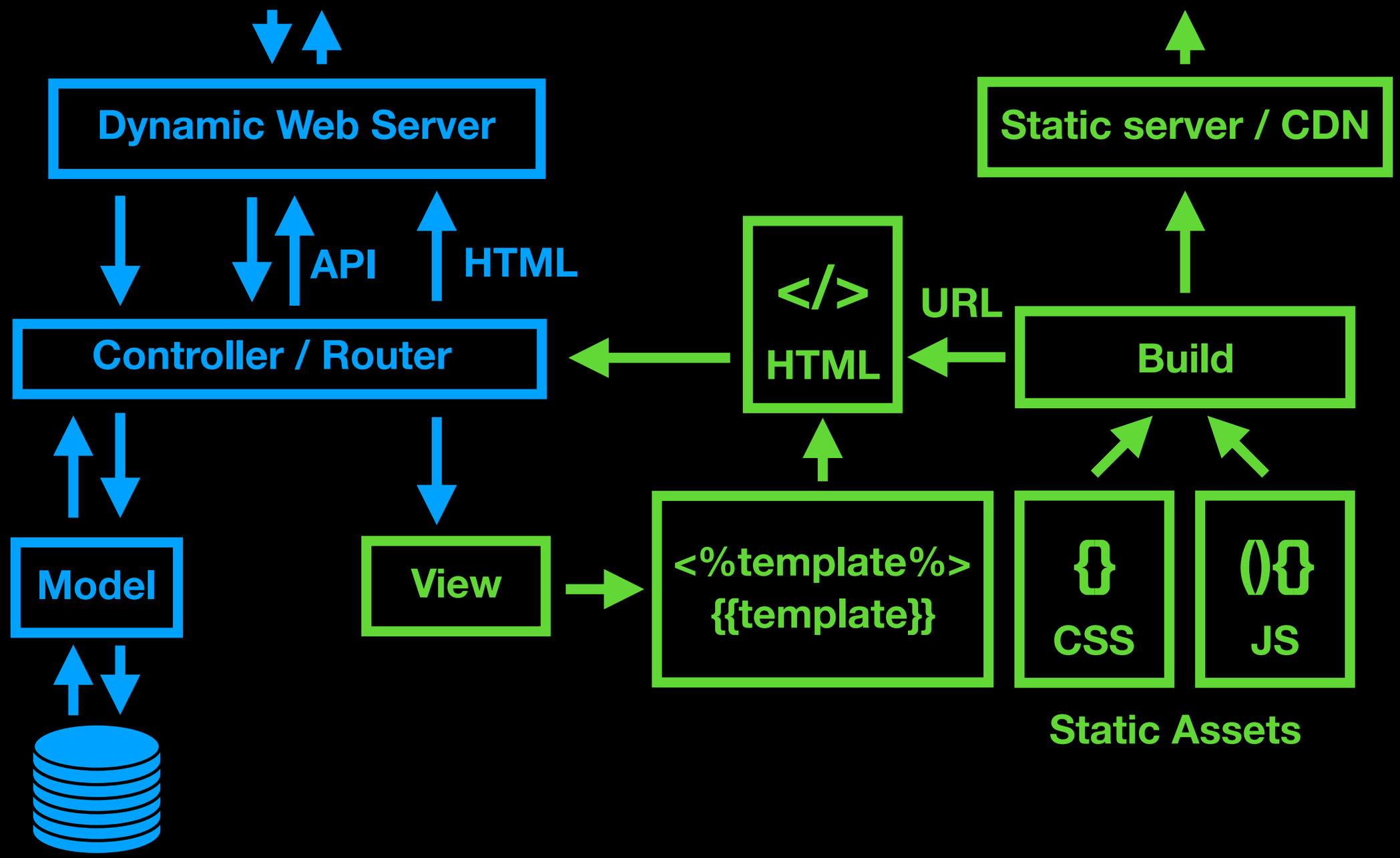
『前端』的来历：传统 web 应用开发

服务器端 web 框架



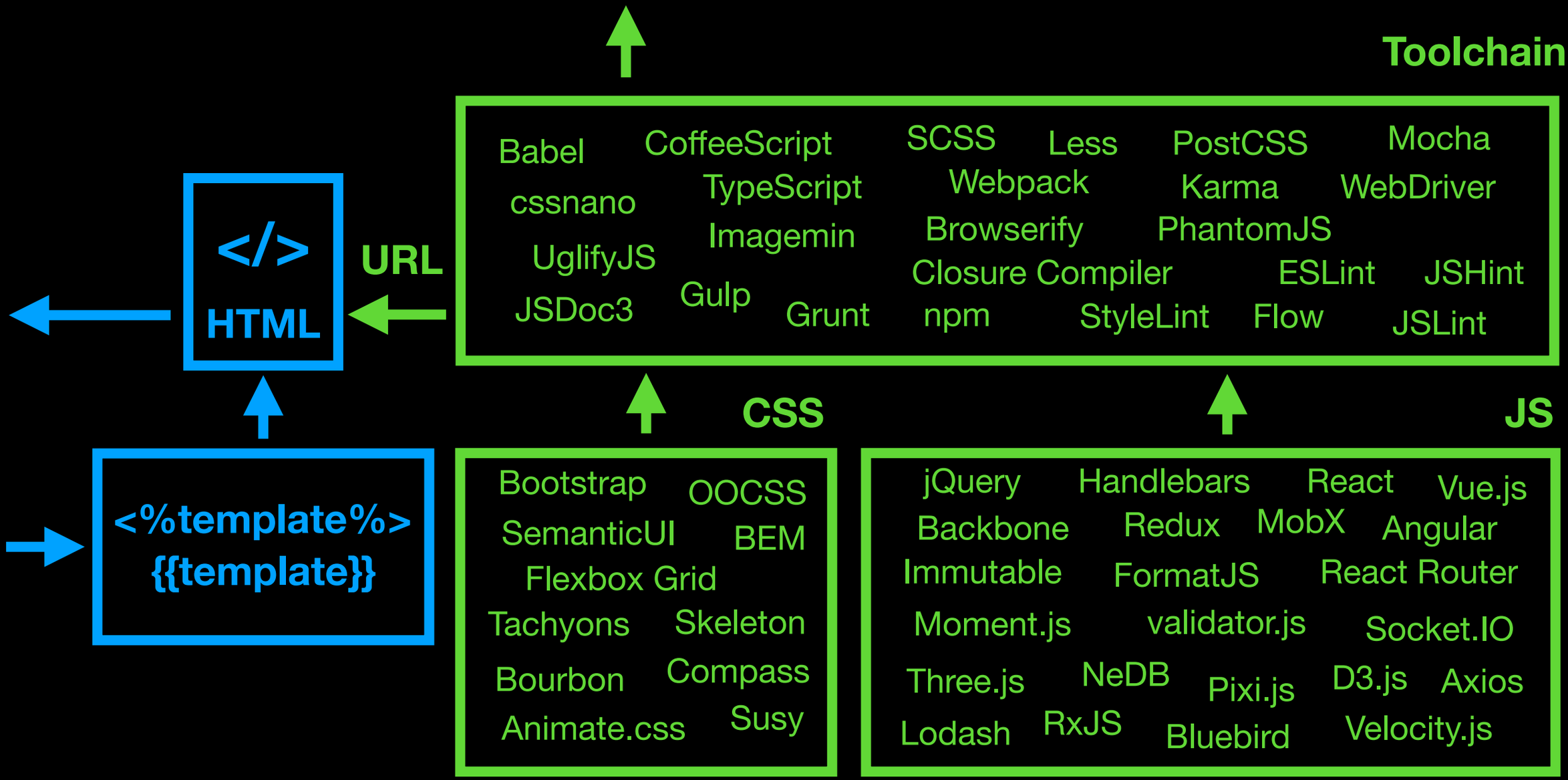
『前端』的来历：传统 web 应用开发

『前端』= 服务器端 web 框架中的视图层



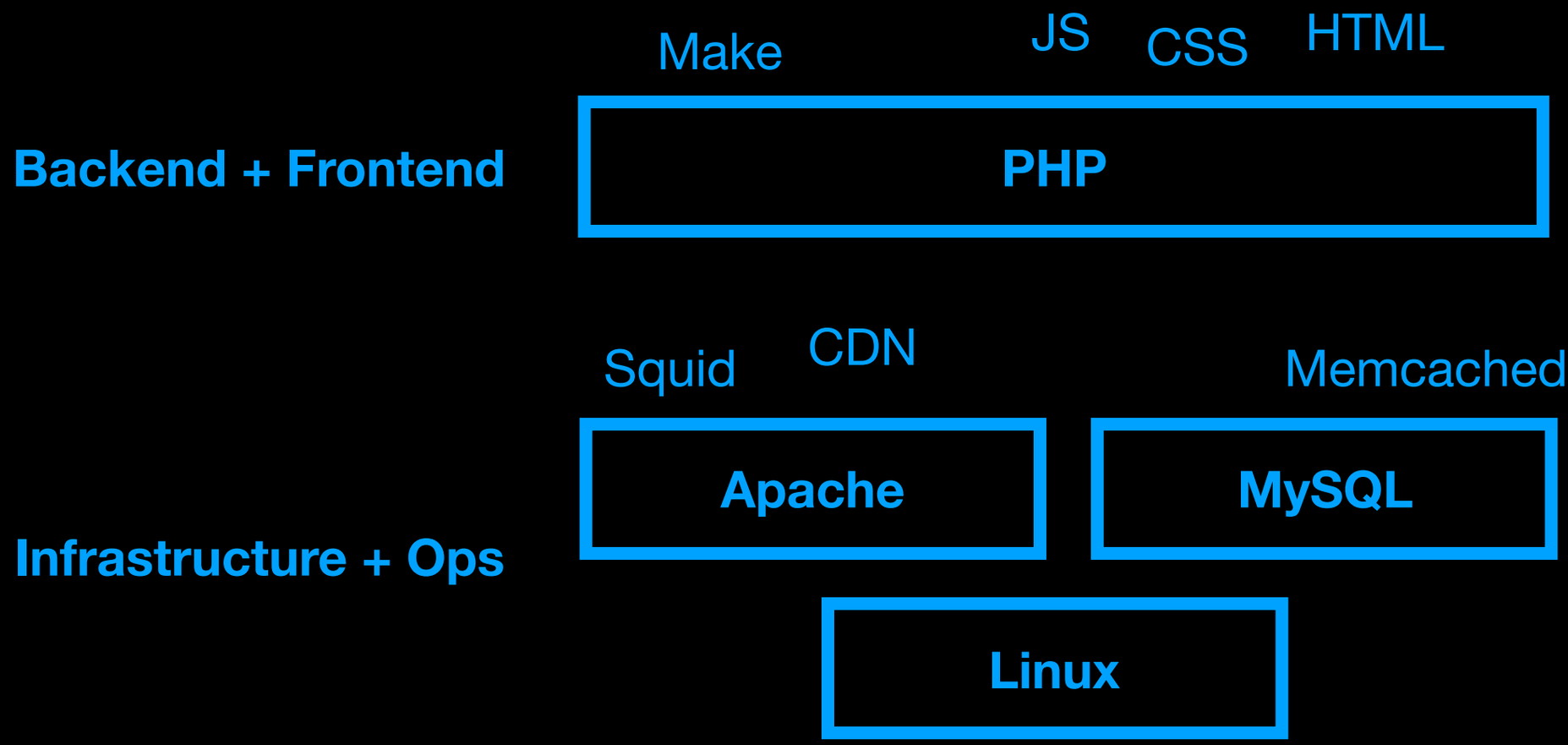
『前端』的来历：传统 web 应用开发

用户需求和业务逻辑从服务器端向浏览器端转移，
『前端』层越来越厚、与 web 框架的其他部分越来越割裂



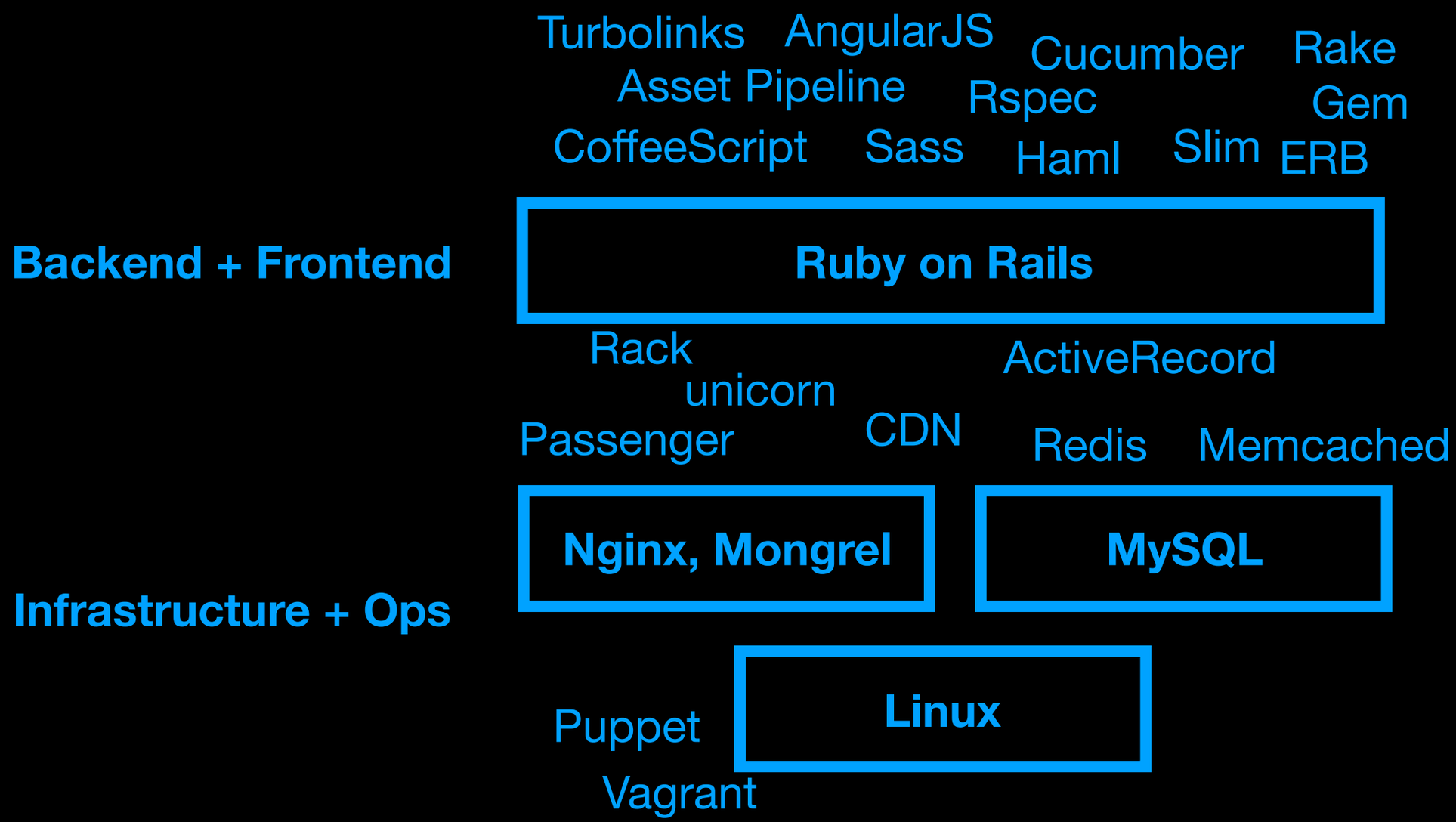
『全栈』的来历：LAMP 和 RoR

LAMP 时代的 PHP 全能战士



『全栈』的来历：LAMP 和 RoR

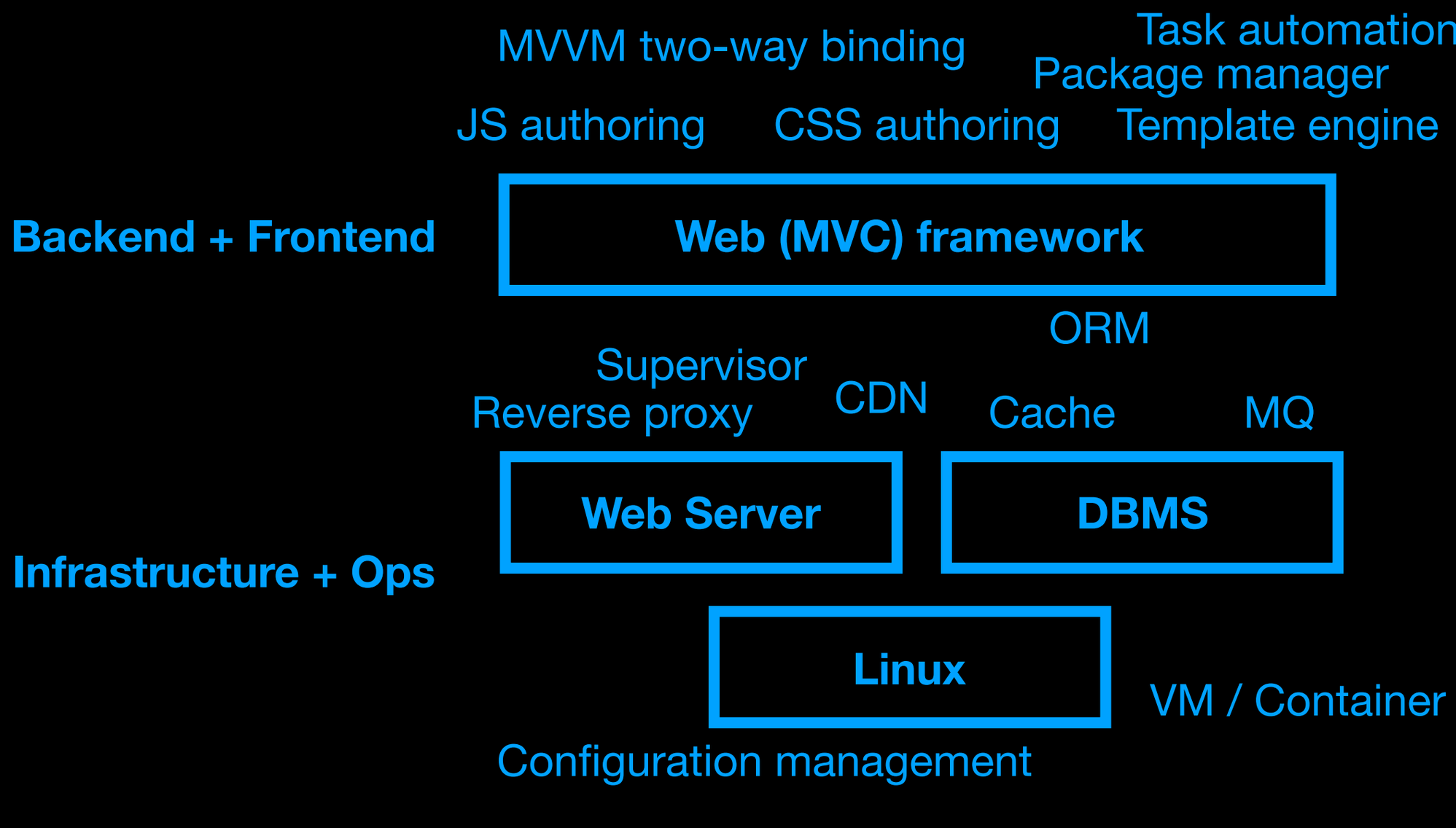
RoR 时代的 Ruby 全栈工程师



『全栈』的来历：LAMP 和 RoR

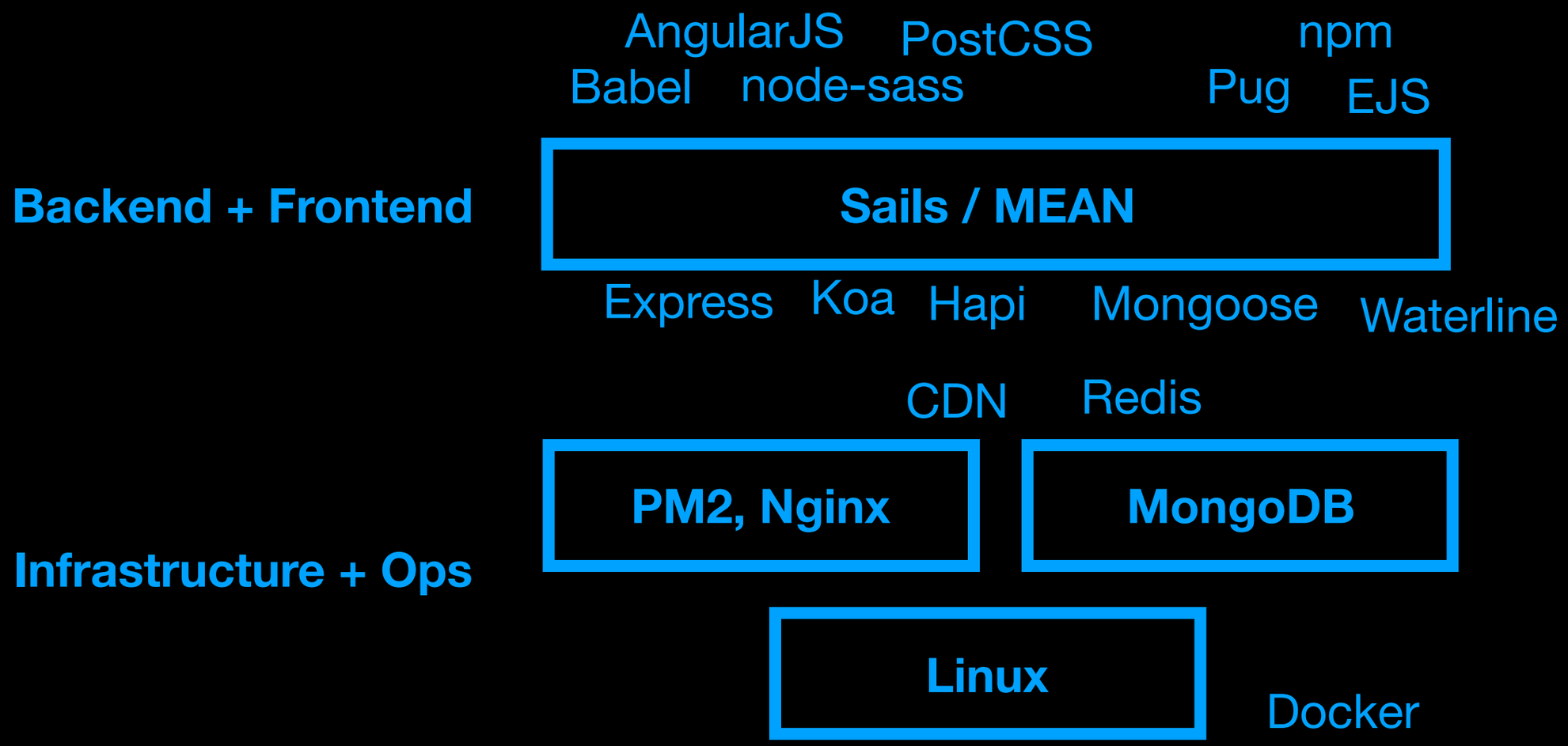
『全栈』=

『薄』前端 + 『薄』后端 + 『非分布式』基础设施和运维



『全栈』的来历：LAMP 和 RoR

Node.js web (MVC) framework 仍然是这种『全栈』



『前端』/『后端』/『全栈』的黑历史

1. 『后端』的来历：早期 web 开发 = 内容开发
2. 『前端』的来历：传统 web 应用开发
3. 『全栈』的来历：LAMP 和 RoR

背景

1. 『前端』/『后端』/『全栈』的黑历史
2. 时代变革导致趋势大幅加速
3. 新分工和『现代 web 开发』

时代变革导致趋势大幅加速

1. 移动时代
2. AI 时代
3. XXX 时代的前夜

移动时代：从内容到服务

传统互联网

- 使用场景固定且局限
- 『内容』为主
- 『服务』局限于特定领域

E-commerce SNS Blog
Email IM Forum
Download

移动时代：从内容到服务

传统互联网

- 使用场景固定且局限
- 『内容』为主
- 『服务』局限于特定领域

移动互联网

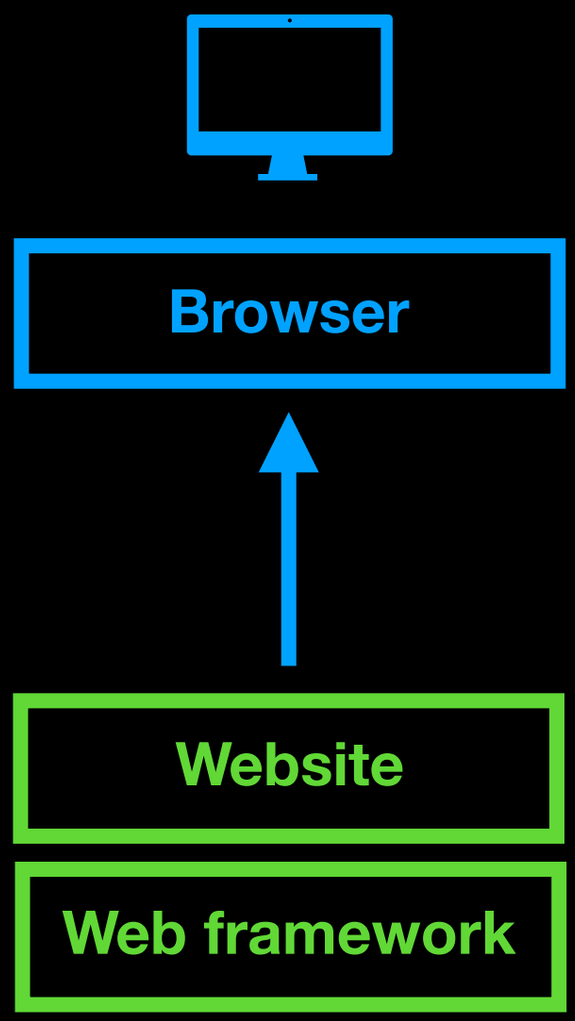
- 使用场景触达每个角落
- 更多事物被连接到云端
- 海量『服务』

移动时代：从内容到服务

- 客户端需求复杂化，大量应用流行，对用户体验的期望提高
- 客户端渲染成为『刚需』
- 客户端程序不得不具备完整的生命周期、分层架构和技术栈

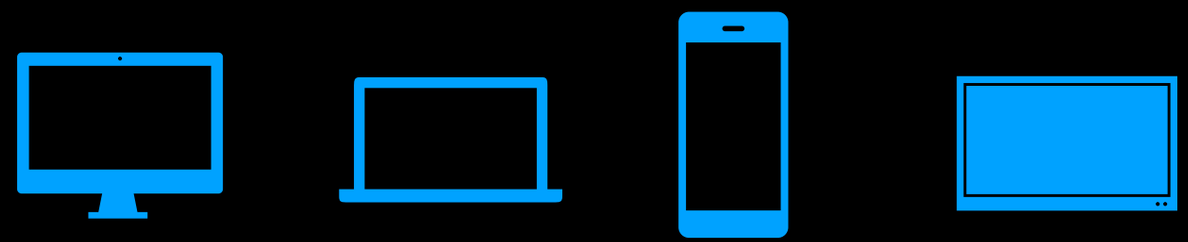
移动时代：从单站到多端

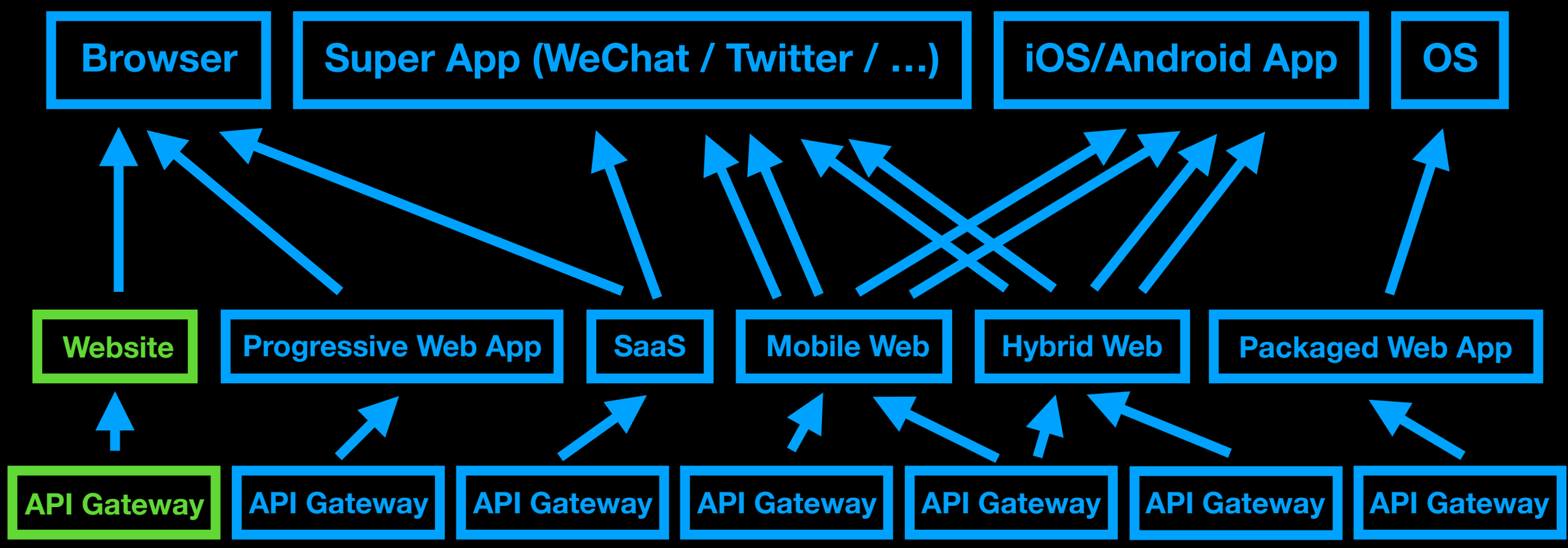
- 多设备：网站不再是唯一客户端
- 多平台（超级 app 平台化）：单一网站的解体



移动时代：从单站到多端

- 多设备：网站不再是唯一客户端
- 多平台（超级 app 平台化）：单一网站的解体





移动时代：从单站到多端

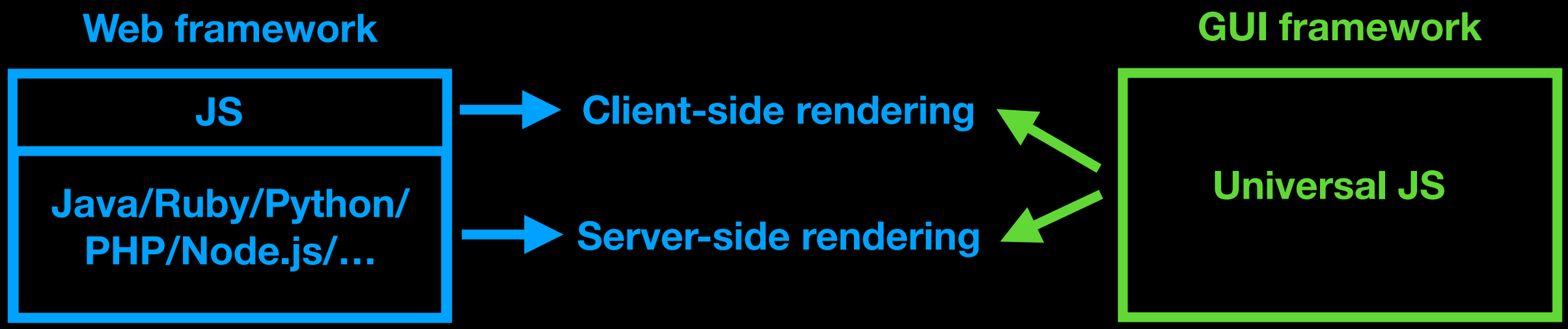
- Client Agnostic: 服务器端只输出 API，剥离视图/『前端』
- web 客户端变成独立开发和部署的程序/项目，不再是服务器端 web 程序/项目中的『前端』层
- 每个客户端都倾向于拥有专门为自己量身打造、可被自己掌控的 API Gateway

移动时代：资源限制

- 网络环境多变：精简网络请求，Offline First
- 设备资源有限（电池、内存...）

移动时代：资源限制

- 网络环境多变：精简网络请求，Offline First
- 设备资源有限（电池、内存...）
- 服务器端渲染成为『刚需』
- Isomorphic/Universal JS：服务器端渲染是 web 客户端的组成部分，是应用的一种部署方式，而不是『后端』



AI 时代：从主机到云、从增删改查到大数据

- 分布式计算，分布式存储
- 服务器端需求的复杂化，微服务流行

AI 时代：从主机到云、从增删查改到大数据

- 单一『后端』开始变厚、解体、分层
- 面向数据和分布式架构的『层』远远厚过面向应用『层』

APIs for clients

API gateway
Microservice

API gateway
Microservice

Microservice
Microservice

API gateway
Microservice

Big Data / AI

Web scraping
Log analysis
Microservice
Ad hoc analysis
Microservice
Interactive computing
Microservice
ETL
MapReduce
DAG
SQL
Regression Analysis
NLP
TF-IDF
Word2Vec
Collaborative Filtering
GAN
ML
SVM
Decision Tree
Deep learning
CNN
RNN
LSTM

Distributed infrastructure

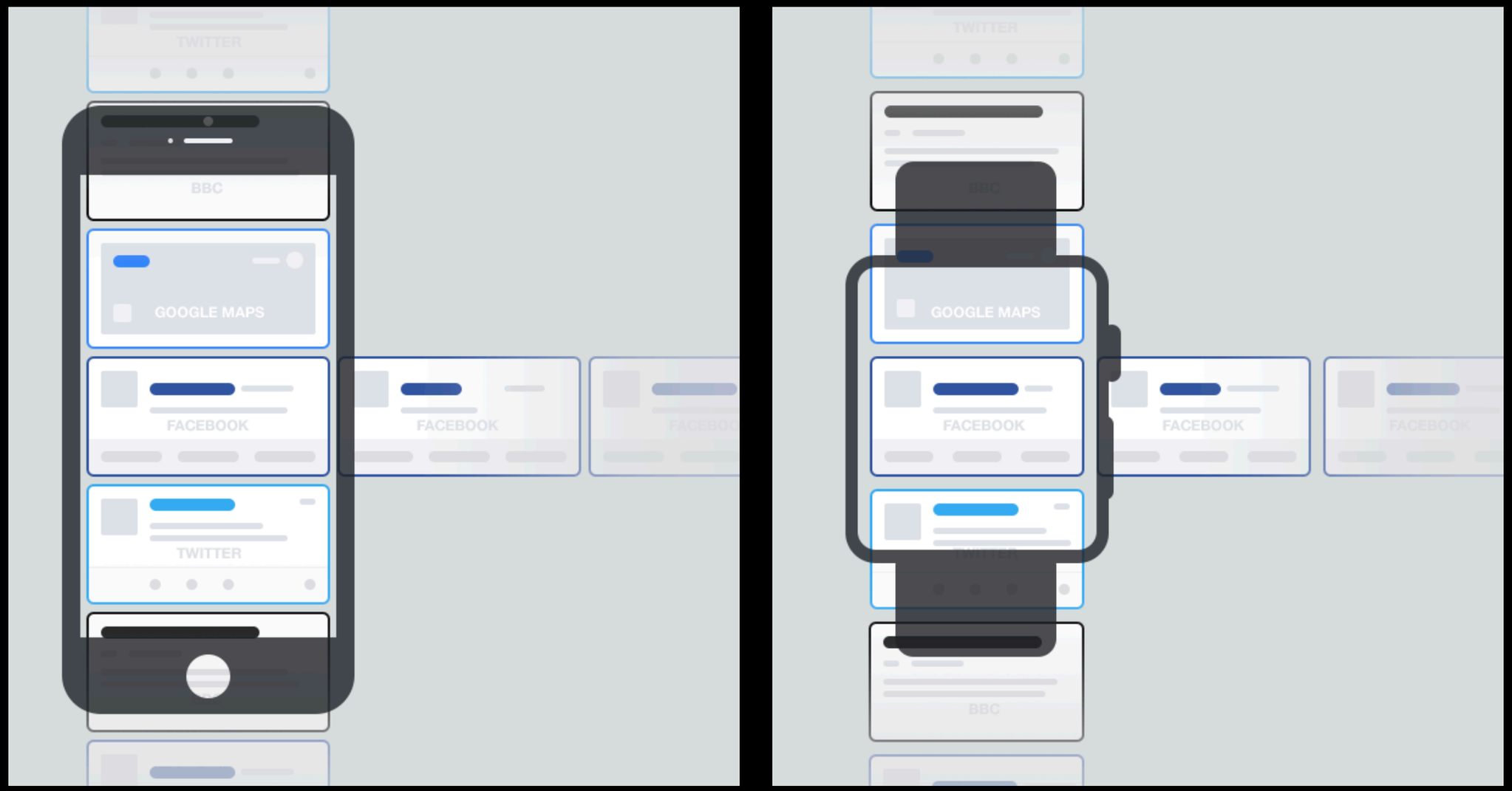
Service discovery
Container clusters
Service registry
Batch processing
Load balancing
Stream processing
VPC
Object storage
Resource scheduling
Message queue
Warehouse
NoSQL
Analytic DB
NewSQL
High availability
Distributed File System
Resiliency

XXX 时代的前夜：新人机界面

超越触屏： Voice UI、AR/MR、VR

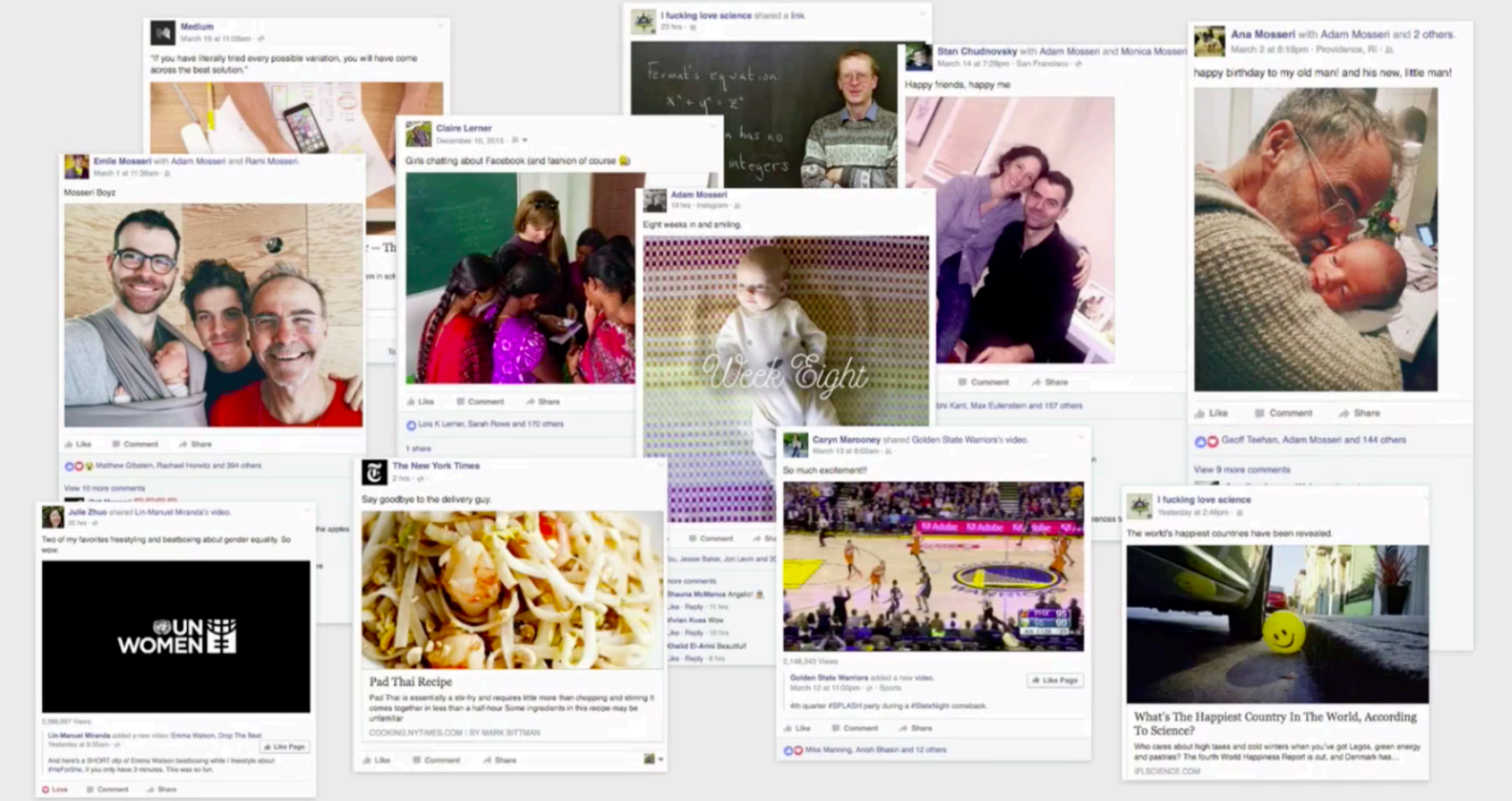
XXX 时代的前夜：新人机界面

万能人机界面：信息流、卡片



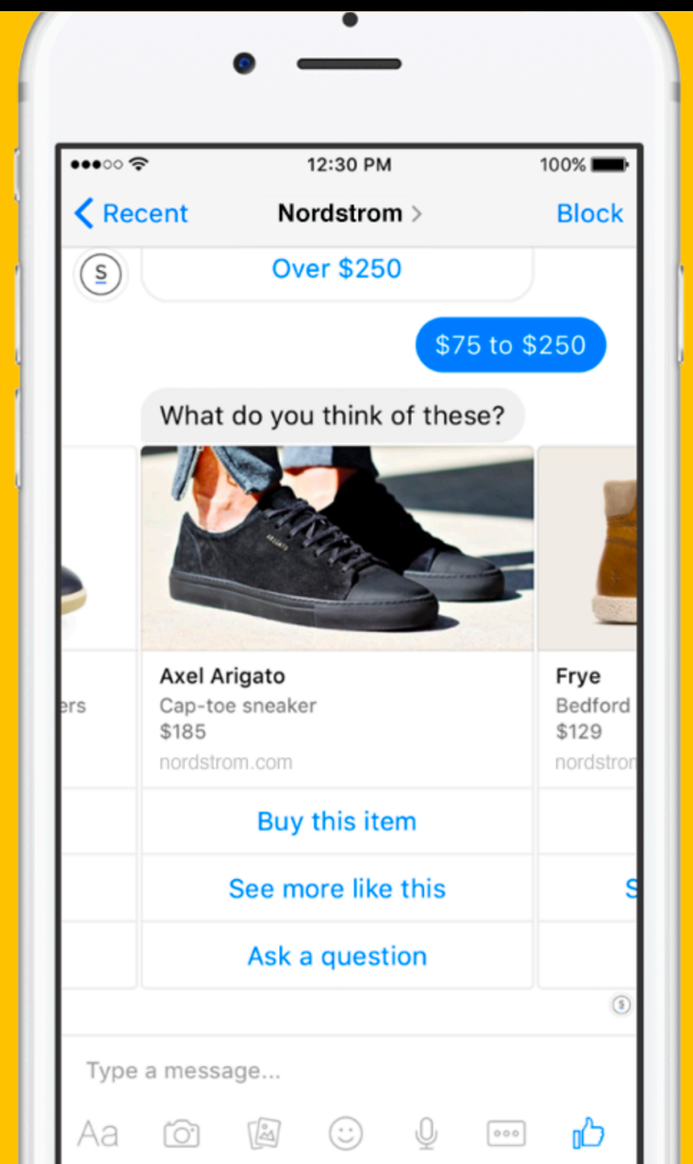
XXX 时代的前夜：新人机界面

万能人机界面：信息流、卡片



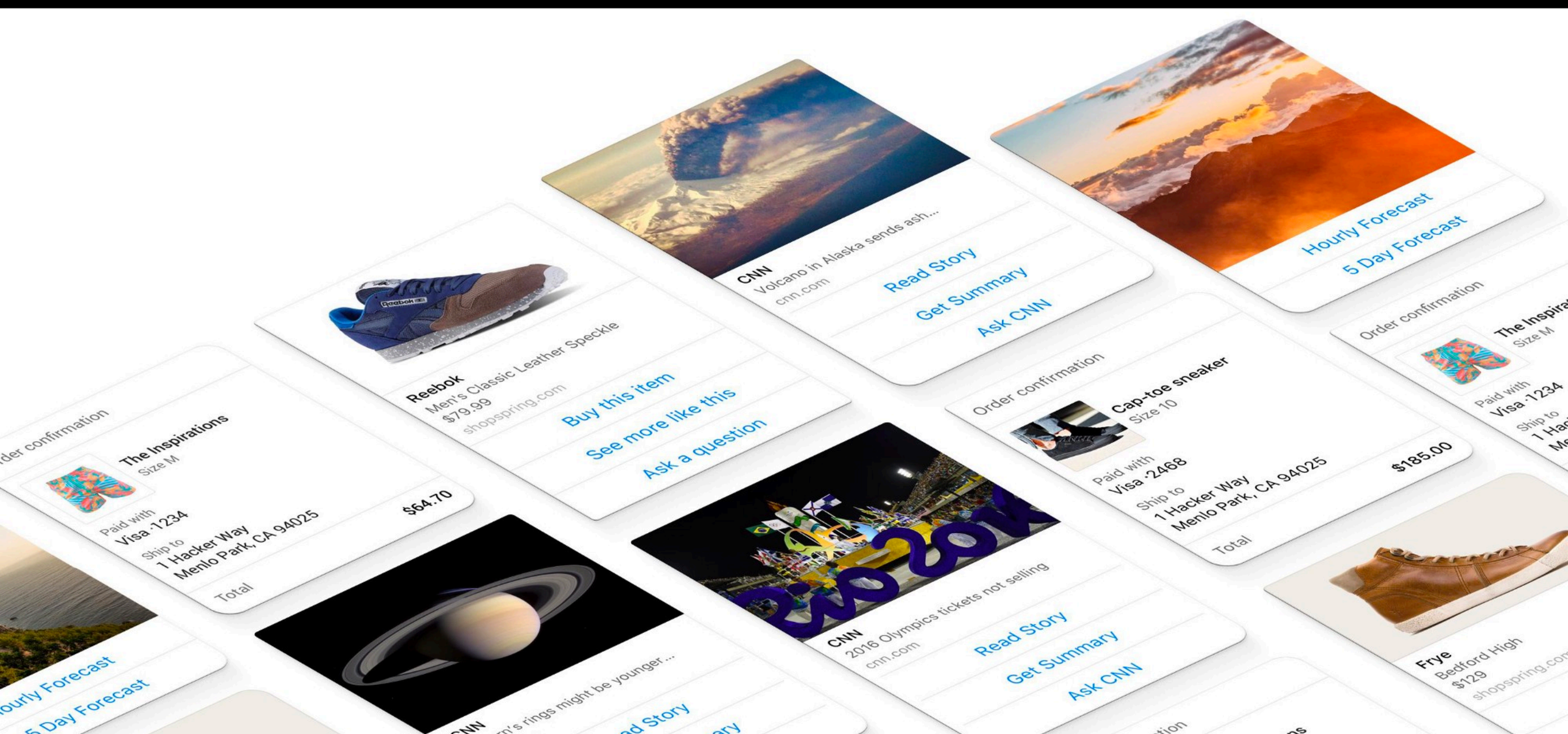
XXX 时代的前夜：新人机界面

万能人机界面：信息流、卡片



XXX 时代的前夜：新人机界面

万能人机界面：信息流、卡片



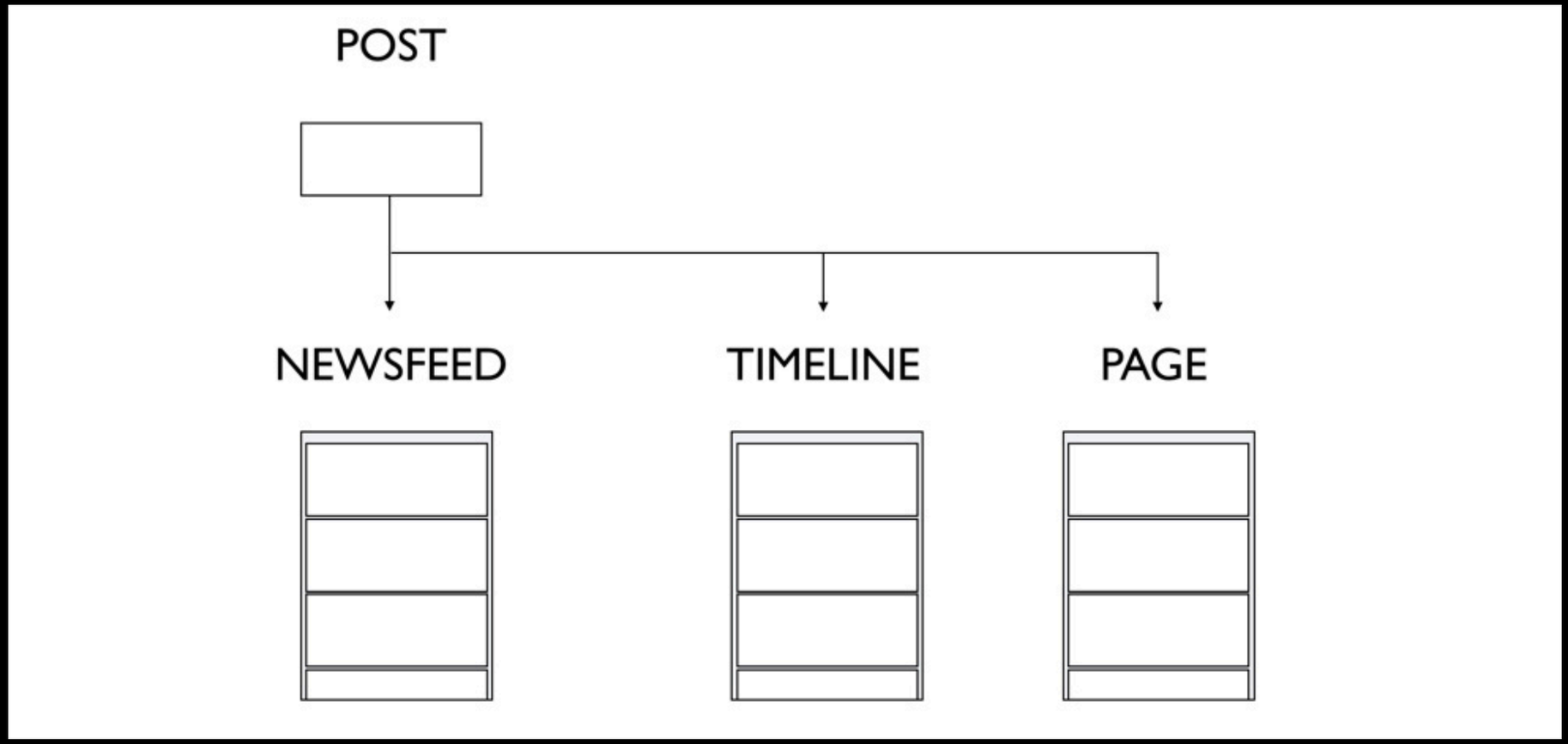
XXX 时代的前夜：新人机界面

万能人机界面：信息流、卡片

XXX 时代的前夜：从单一应用到离散系统

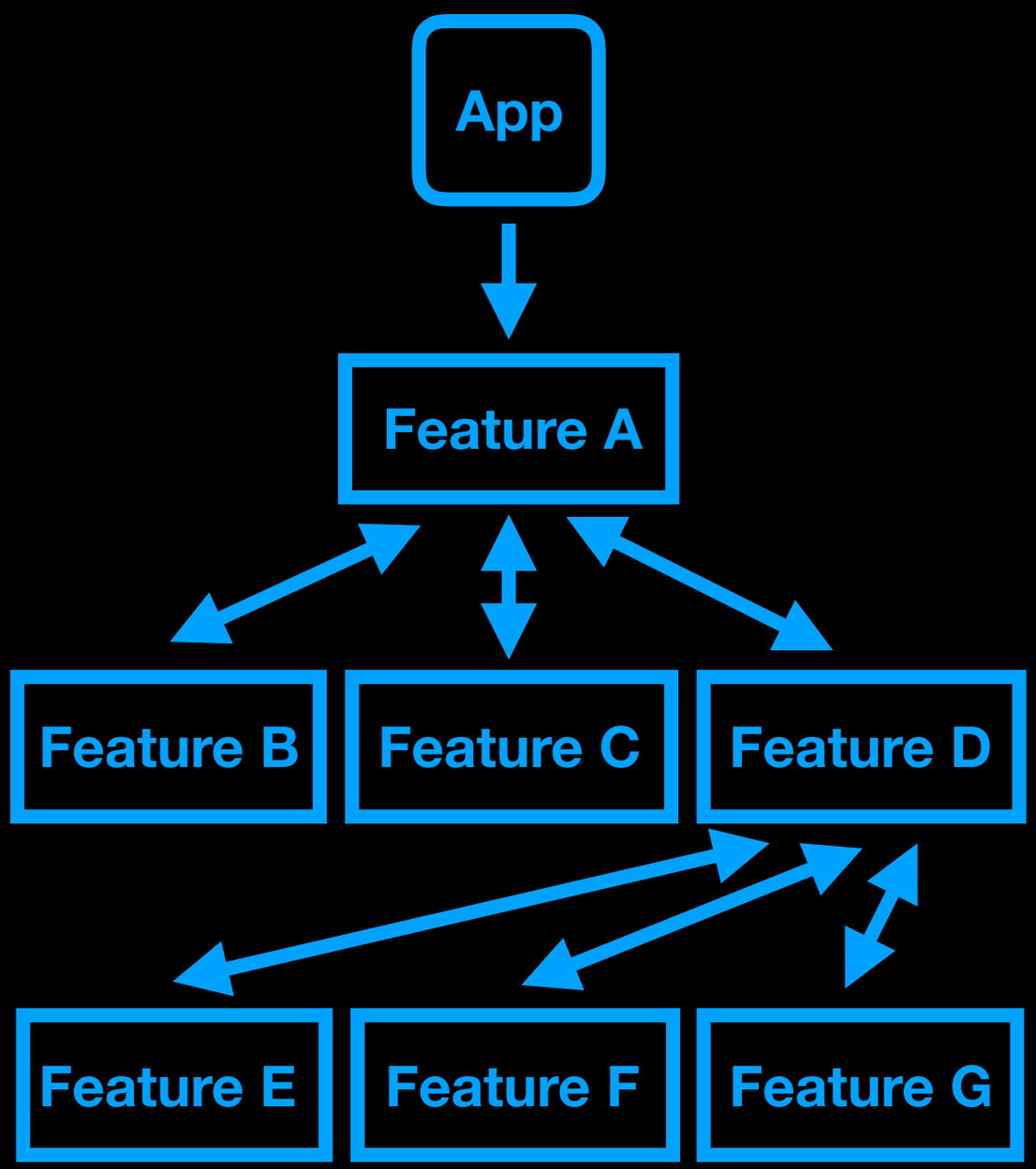
“Facebook is not a blue and white website. It’s a system of connected things which exist independently and in aggregated views.”

– Paul Adams (Creating systems not destinations)



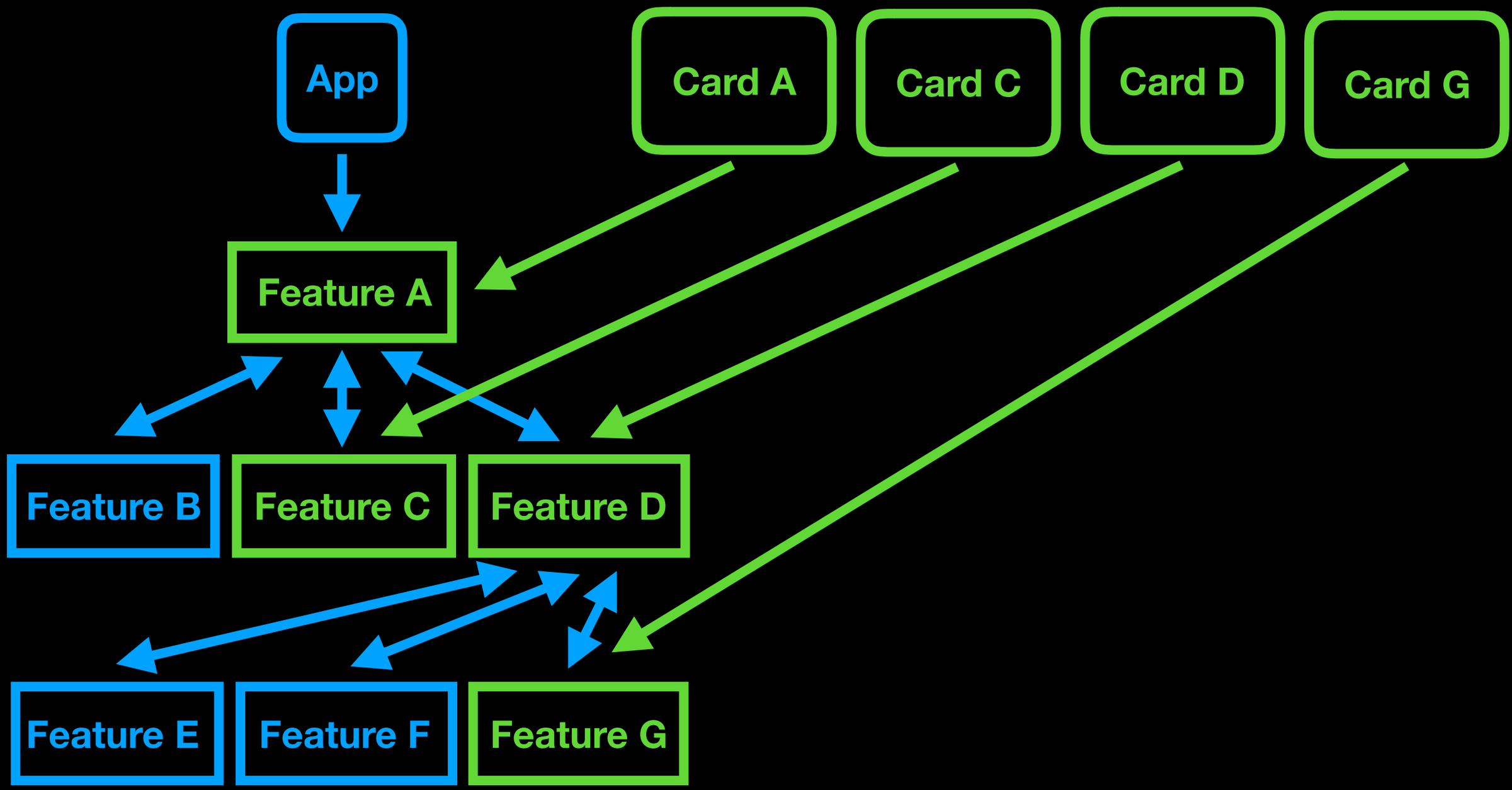
XXX 时代的前夜：从单一应用到离散系统

单一应用解体，入口扁平化



XXX 时代的前夜：从单一应用到离散系统

单一应用解体，入口扁平化



时代变革导致趋势大幅加速

1. 移动时代：从内容到服务、从单站到多端、资源限制

2. AI 时代：从主机到云、从增删查改到大数据

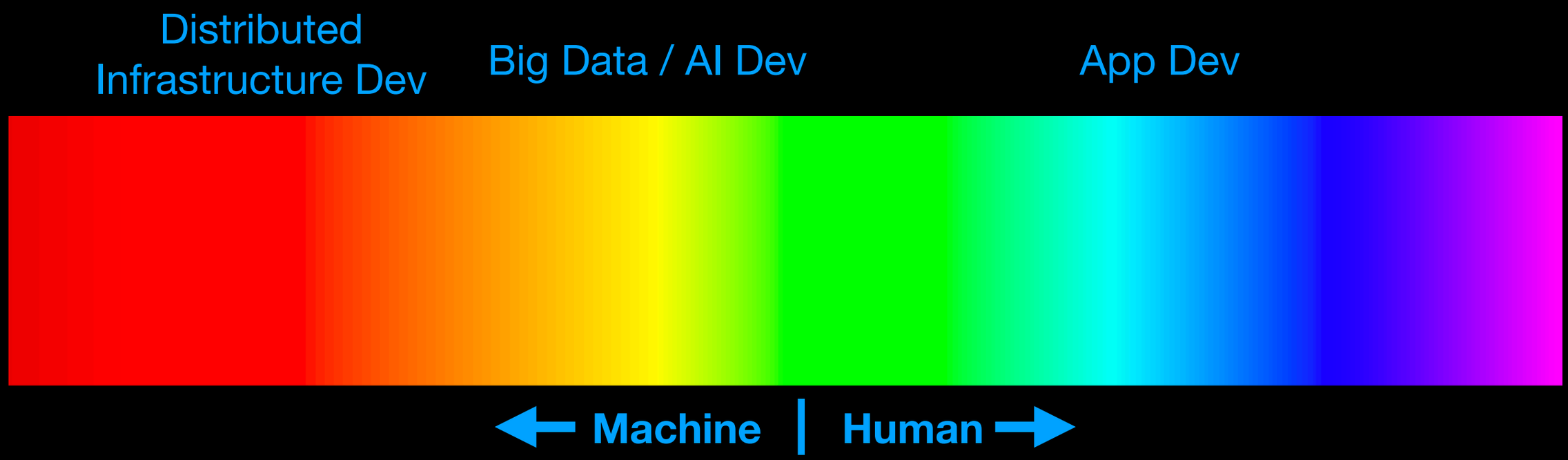
3. XXX 时代的前夜：新人机界面、从单一应用到离散系统

背景

1. 『前端』/『后端』/『全栈』的黑历史
2. 时代变革大幅加速了趋势
3. 新分工和『现代 web 开发』

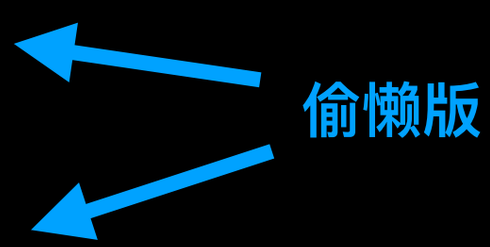
新分工和『现代 web 开发』

- 面向人类：应用开发者
- 面向数据：大数据/AI开发者
- 面向机器：分布式基础设施开发者（向公有云企业集中）



新分工和『现代 web 开发』

- 应用开发者：现代 web 开发技术（Universal JS、Open Web 平台、原生扩展）、原生平台开发技术
- 大数据/AI开发者：Python、JVM
- 分布式基础设施开发者：Go、JVM



新分工和『现代 web 开发』

『现代 web 开发』的特征

- 以客户端为主体：服务器端部分很薄，大量成熟业务逻辑、数据能力、运维能力和基础设施被 API 化、云服务化（由第三方服务商和团队内的专业开发者提供）
- 不只有客户端：服务器端渲染 + API Gateway + 部分面向应用的微服务
- 不只有浏览器：超级 app 平台、各种基于 Web Runtime / JS Runtime 的 Hybrid 技术
- JS everywhere

新分工和『现代 web 开发』

『现代 web 开发』的方向

- 主流软件开发技术：日臻完善的工具链、工程环境和生态系统
- More APIs, More Power：HTML5 API（交互、访问、离线、HTTP/TCP/UDP、图像、显卡、…）、 Node.js（底层网络、操作系统、…）、Hybrid API（系统深度集成）
- 新形态 web 应用：传统 web 和传统应用软件的融合


JAVASCRIPT
LIGHTYEAR
JAVASCRIPT EVERYWHERE

JavaScript Fatigue!

问题和方法

1. JavaScript Fatigue 的根源
2. JavaScript Fatigue 的解决
3. Awesome list
4. Spellbook 的方法

JavaScript Fatigue 的根源

- 多样性
- 需求多
- 成本低

JavaScript Fatigue 的根源：多样性

- 开发者基数庞大，来自各种技术背景，从事各种行业面对各种需求
- 每年增长 100%，明年超过 Java，任何时候都有 50% 社区成员今年才开始写 JS

Background

Web designer

Ruby developer

PHP developer

Flash/AS3 developer

C++ developer

JVM developer

Microsoft developer

Requirements

SaaS developer

Realtime web app developer

Mobile web developer

Hybrid app developer

Creative developer

Data visualization developer

Network developer

CLI developer

JavaScript Fatigue 的根源：多样性

- 开发者基数庞大，来自各种技术背景，从事各种行业面对各种需求
- 每年增长 100%，明年超过 Java，任何时候都有 50% 社区成员今年才开始写 JS

Melting Pot

VS

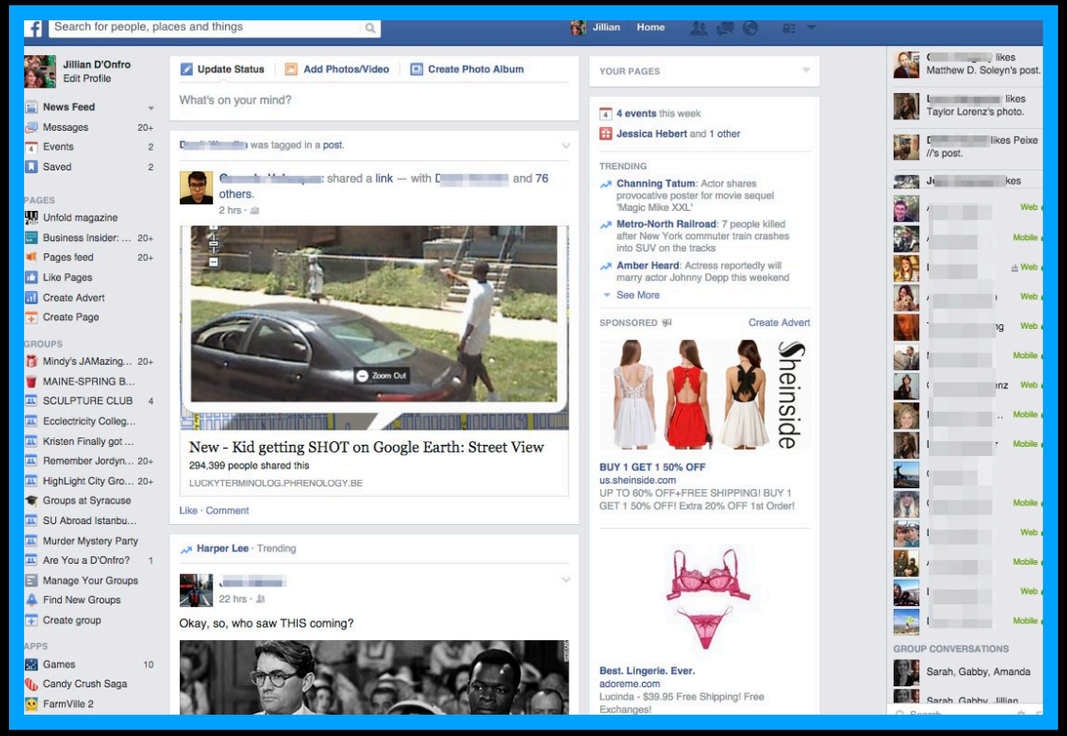
Salad bowl

JavaScript Fatigue 的根源：需求多

- 面对最前沿的产品需求，需求的多样性和复杂性不断增长，远远超过 HTML/CSS/JS 时代
- JS 覆盖整个软件生命周期，承担更多责任，在软件质量、开发效率等方面都面临更高要求

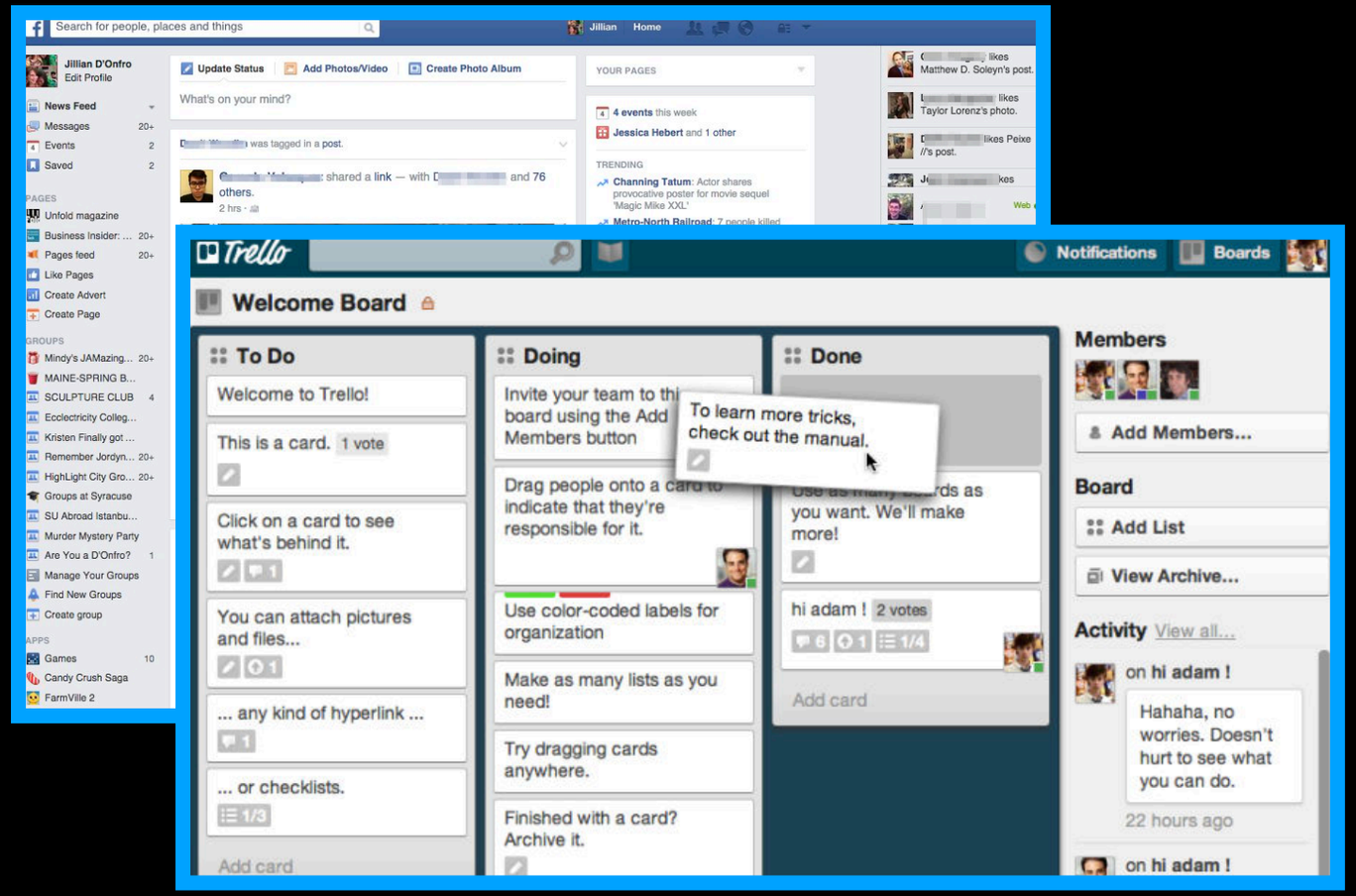
JavaScript Fatigue 的根源：需求多

- 面对最前沿的产品需求，需求的多样性和复杂性不断增长，远远超过 HTML/CSS/JS 时代
- JS 覆盖整个软件生命周期，承担更多责任，在软件质量、开发效率等方面都面临更高要求



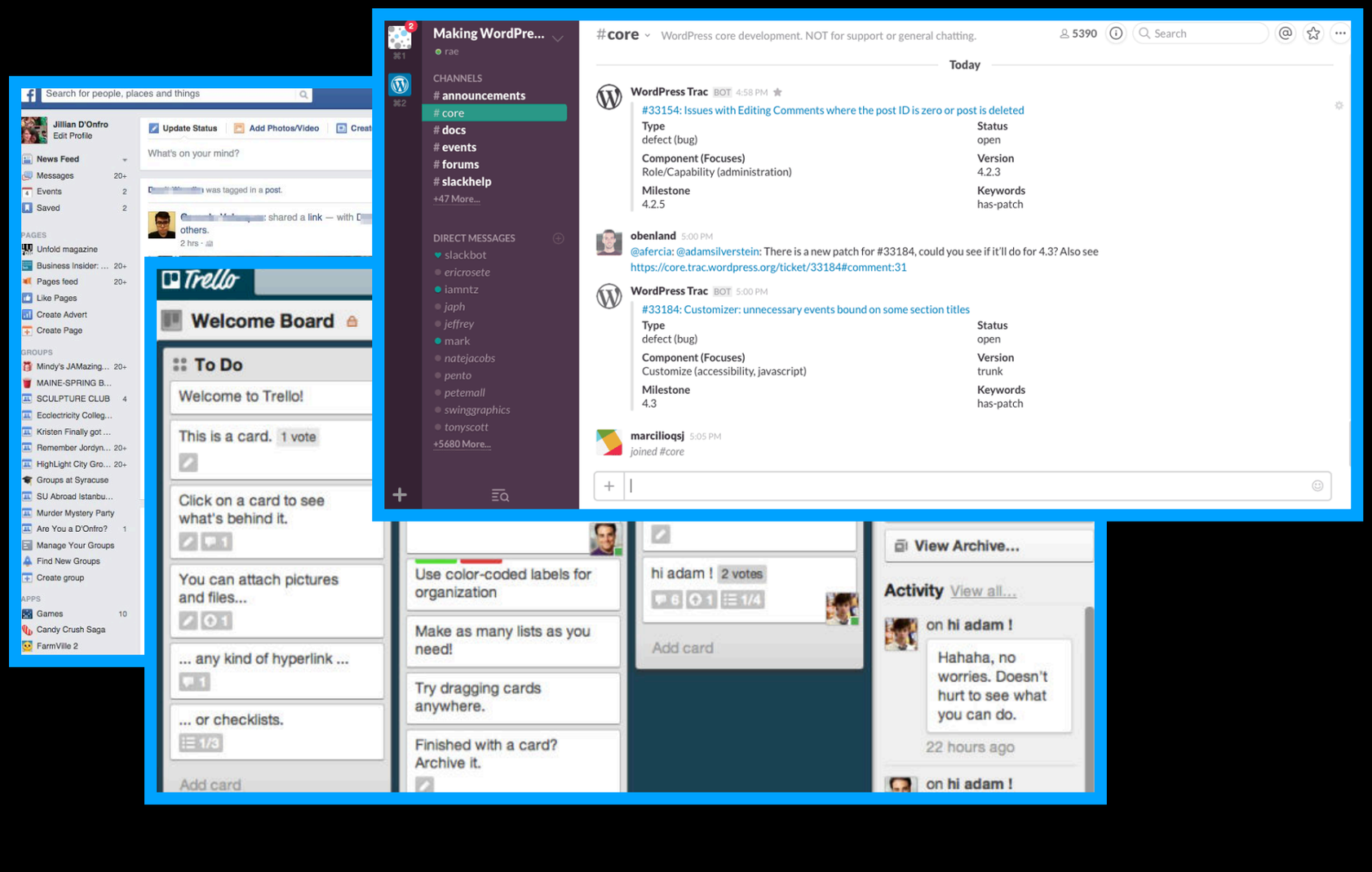
JavaScript Fatigue 的根源：需求多

- 面对最前沿的产品需求，需求的多样性和复杂性不断增长，远远超过 HTML/CSS/JS 时代
- JS 覆盖整个软件生命周期，承担更多责任，在软件质量、开发效率等方面都面临更高要求



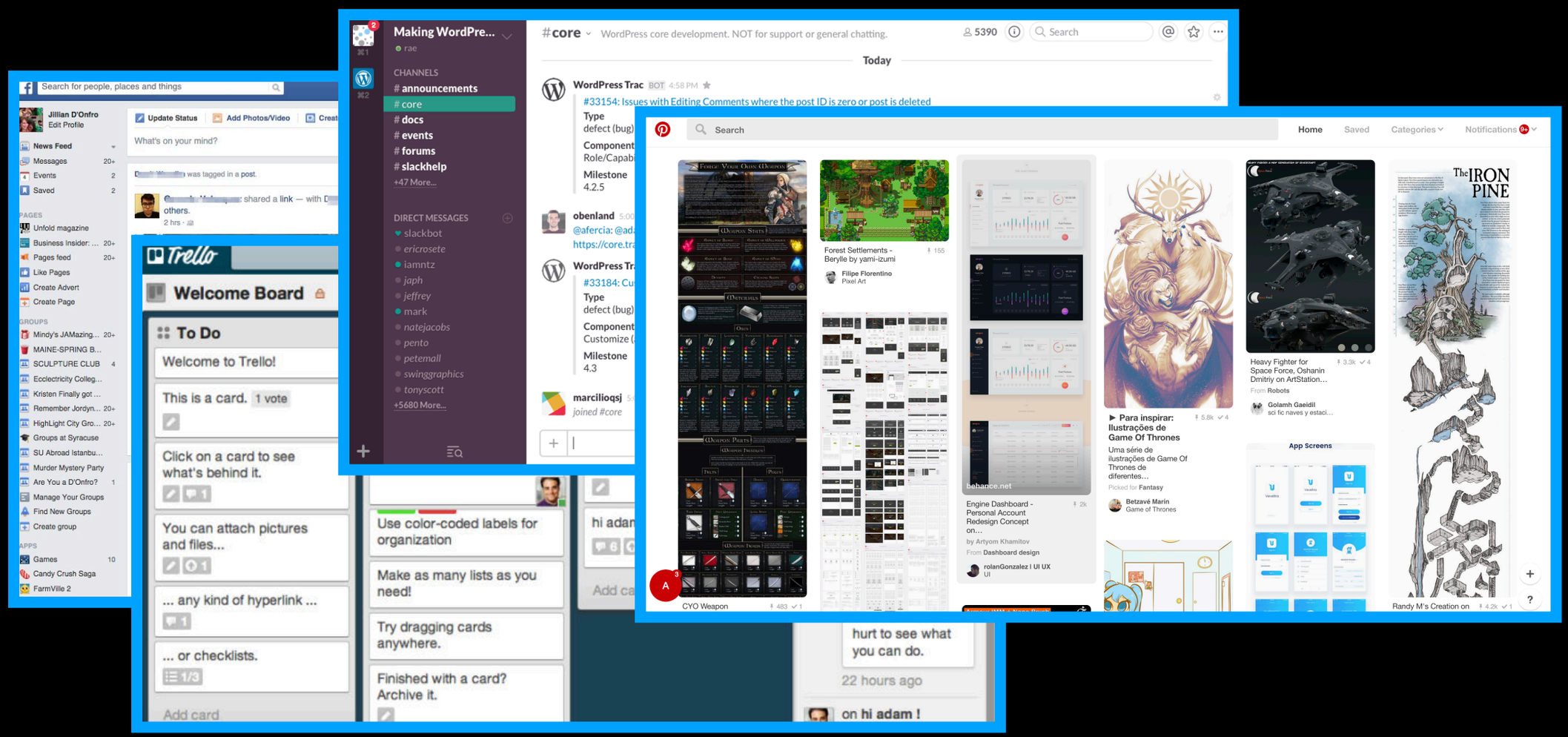
JavaScript Fatigue 的根源：需求多

- 面对最前沿的产品需求，需求的多样性和复杂性不断增长，远远超过 HTML/CSS/JS 时代
- JS 覆盖整个软件生命周期，承担更多责任，在软件质量、开发效率等方面都面临更高要求



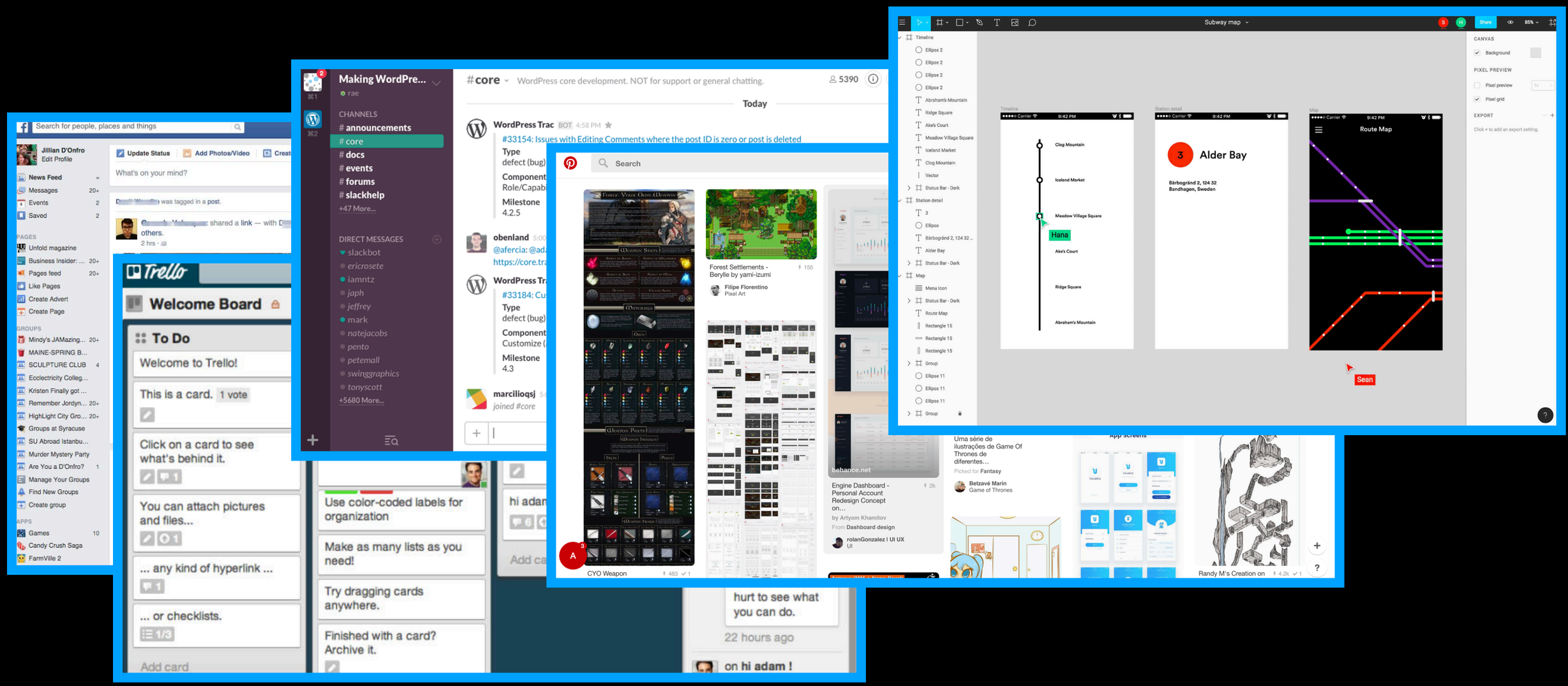
JavaScript Fatigue 的根源：需求多

- 面对最前沿的产品需求，需求的多样性和复杂性不断增长，远远超过 HTML/CSS/JS 时代
- JS 覆盖整个软件生命周期，承担更多责任，在软件质量、开发效率等方面都面临更高要求



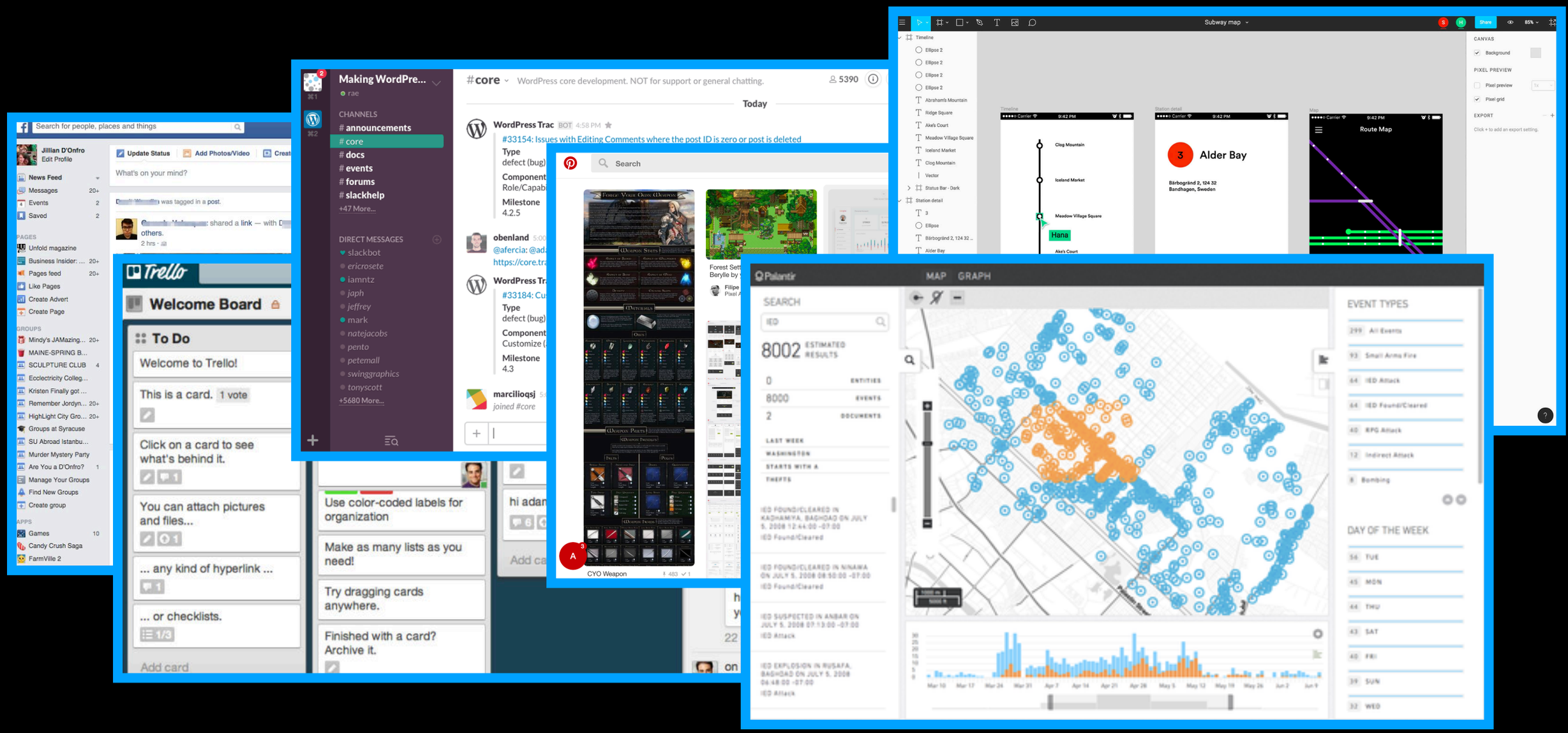
JavaScript Fatigue 的根源：需求多

- 面对最前沿的产品需求，需求的多样性和复杂性不断增长，远远超过 HTML/CSS/JS 时代
- JS 覆盖整个软件生命周期，承担更多责任，在软件质量、开发效率等方面都面临更高要求



JavaScript Fatigue 的根源：需求多

- 面对最前沿的产品需求，需求的多样性和复杂性不断增长，远远超过 HTML/CSS/JS 时代
- JS 覆盖整个软件生命周期，承担更多责任，在软件质量、开发效率等方面都面临更高要求



JavaScript Fatigue 的根源：成本低

- JS 是抽象层级最高的语言之一
- JS 使用的 API 是抽象层级最高的 API
- JS 的初始设计和后续发展带来强大灵活的抽象能力和实现能力

- 实现新技术、新方案的成本低
- 应用新技术、新方案的成本低

JavaScript Fatigue 的解决

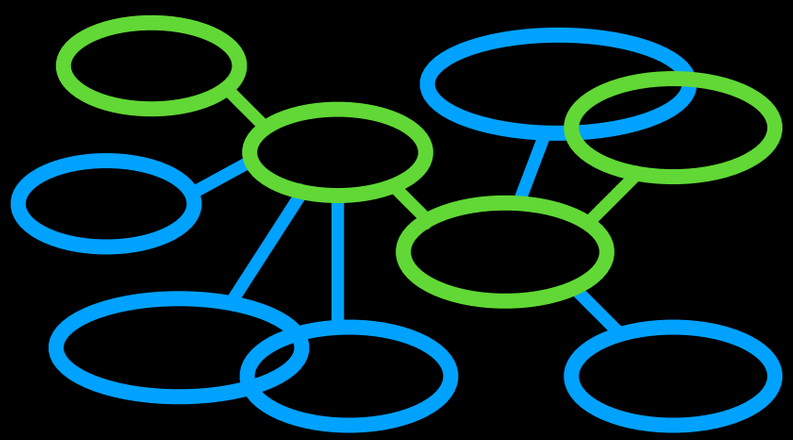
- 提供整体图景
- 普及维护低成本

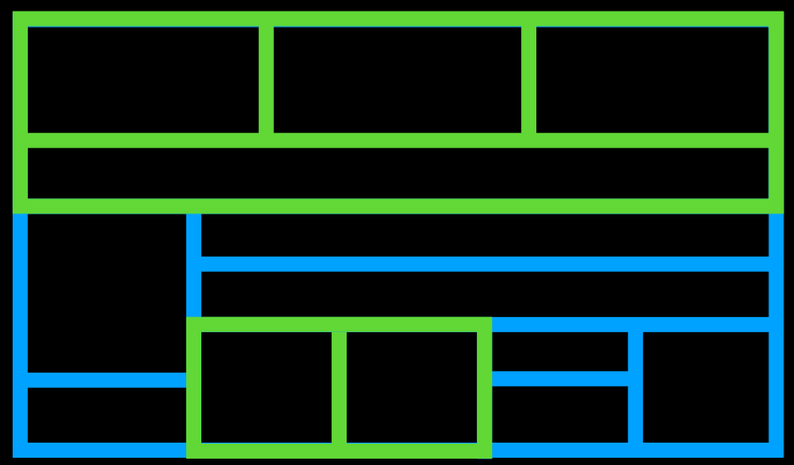
JavaScript Fatigue 的解决：提供整体图景

“The cure for JavaScript fatigue is not to learn all the things.”

– Eric Elliott (Why I’m Thankful for JS Fatigue)

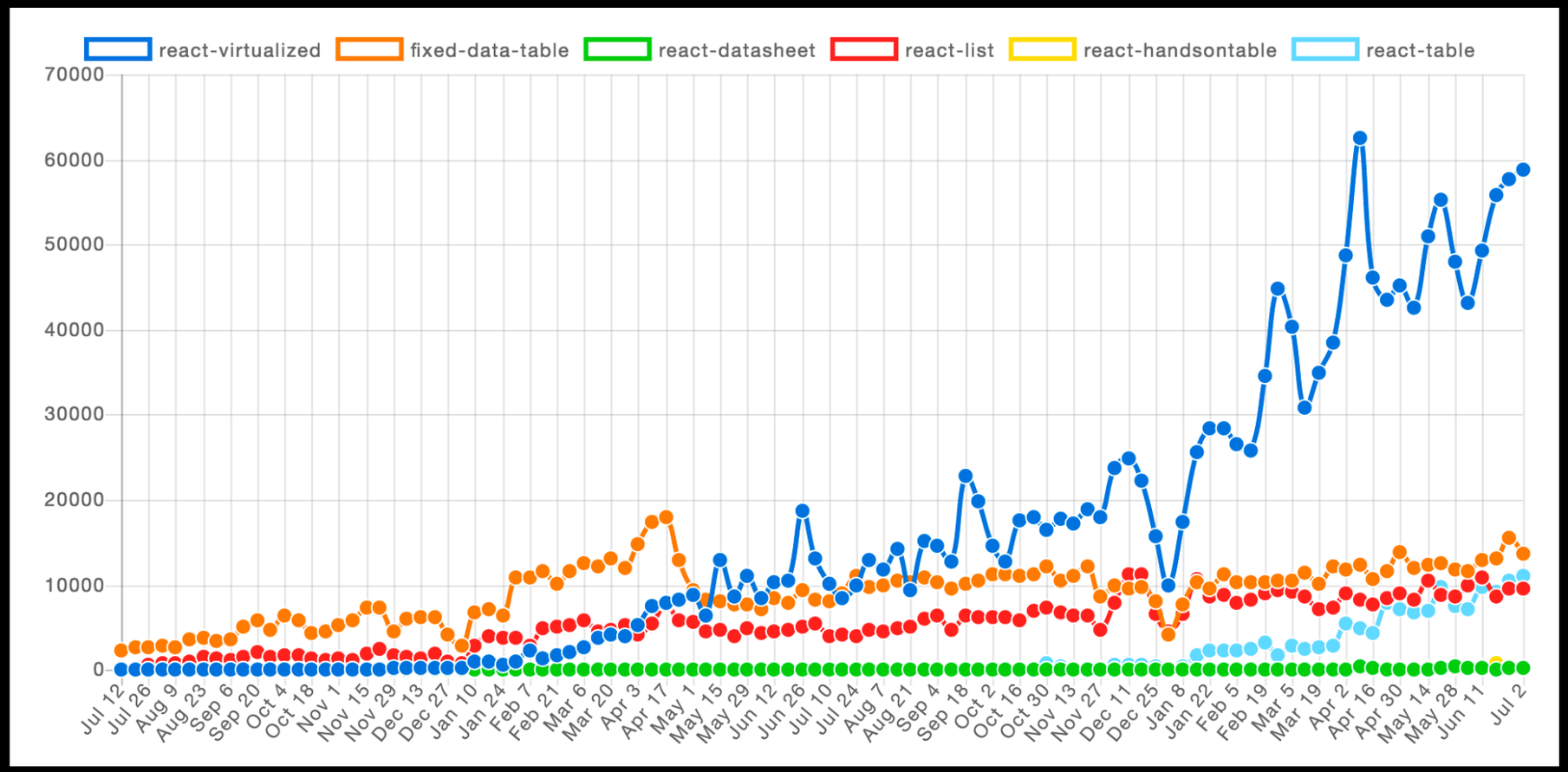
- 问题域和抽象层：不用学所有东西 != 不利用那些东西
- 提供整体图景，有助于对分支和层级的分辨、理解、选择和利用





JavaScript Fatigue 的解决：普及维护低成本

- 普及主流方案：开源生态相当于个人和中小团队的『基础平台部门』



http://www.npmtrends.com/react-virtualized-vs-fixed-data-table-vs-react-datasheet-vs-react-list-vs-react-handsontable-vs-react-table

JavaScript Fatigue 的解决：普及维护低成本

- 普及主流方案：开源生态相当于个人和中小团队的『基础平台部门』
- 填补缺失的中间层

> “The Javascript pendulum has swung from restrictive, monolithic frameworks to modular, boilerplate-hindered libraries.”
>
> *– Eric Clemmons (JS Fatigue)*

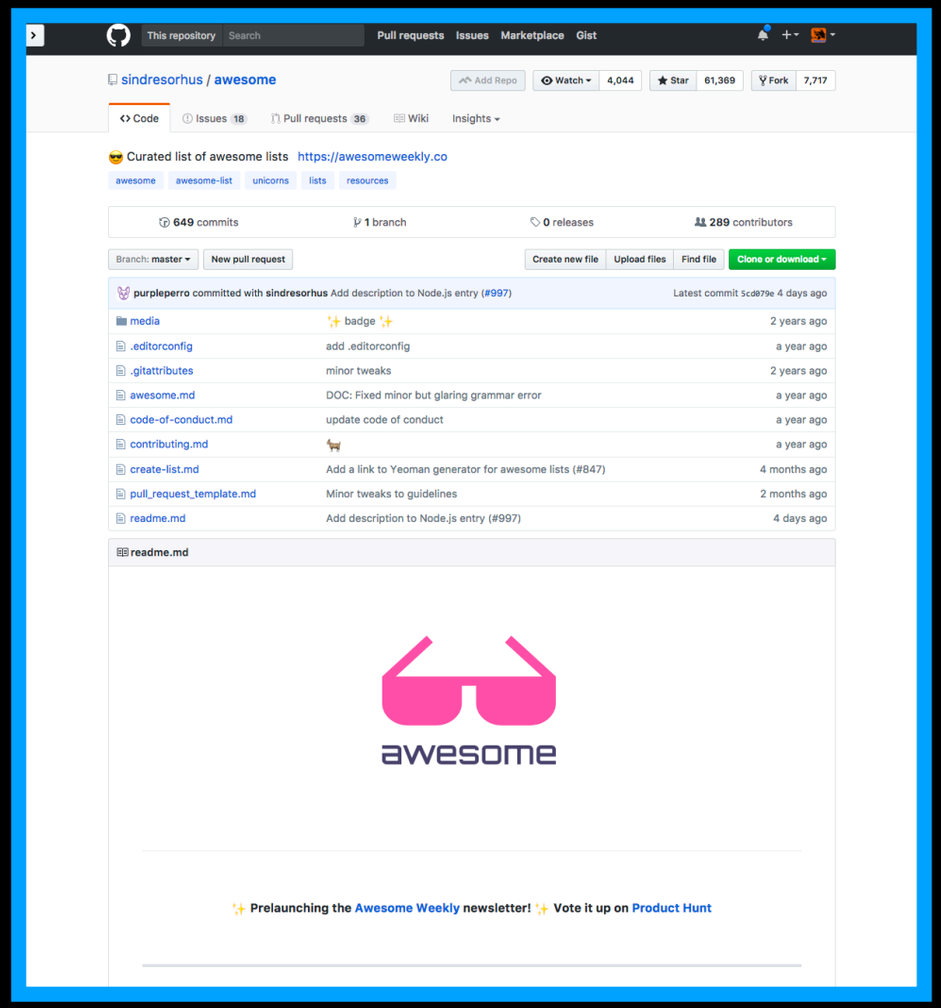
JavaScript Fatigue 的解决

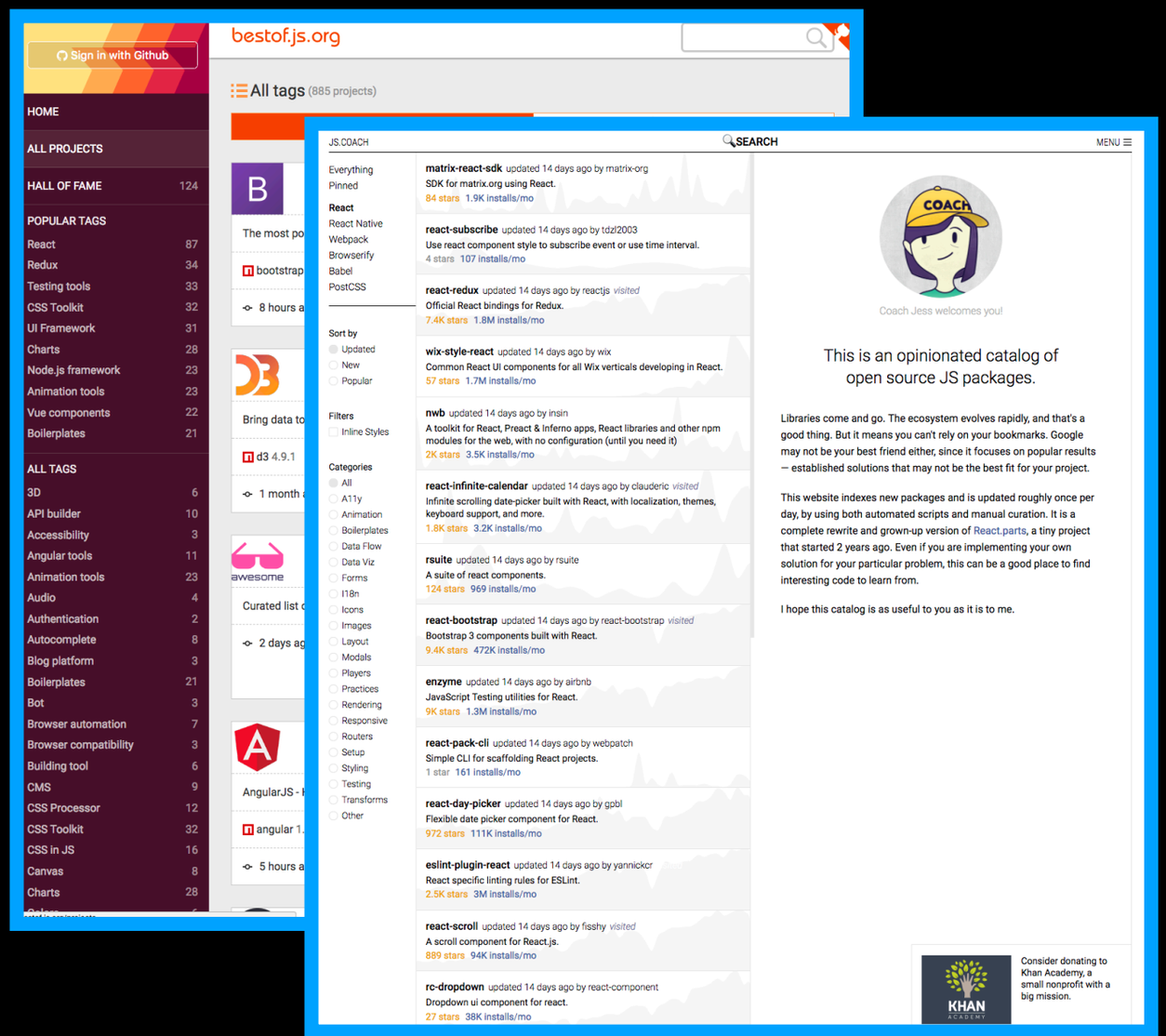
- 提供整体图景
- 普及维护低成本

- 多样性
- 需求多
- 成本低

Awesome list

- 基于专家意见
- 基于统计数据和爬虫





Awesome list 的价值

- 信息爆炸的时代，信息的列表正在成为最重要的信息

- AI 时代，很多解决方案需要人力和机器的结合

- 对『按需搜索』和『时效性文章』的补充

Awesome list 的缺陷

- 『收集』会越『集』越多，列表膨胀，『curation』失效
- 每个局部都可以独立膨胀，列表本身越来越多，彼此割裂

Frontend Development awesome

Manually curated collection of resources for frontend web developers.

You are viewing a browseable version, split by category in many small files. There is also a really huge file with every single resource on one page. Proceed to the totally gigantic file if you are into this kind of thing.

This is the current version, which receives ongoing updates. If you want the good old bookmarks, please use the tag v.1.0. Keep in mind, that the old version has many outdated links.

frontend directory donate PayPal Beerpay donate Flattr chat on gitter follow twitter

Appearance

The outward or visible aspect of a website.

- **Animation**: The process of creating motion and shape change.
- **Typography**: The style, arrangement, or appearance of typeset matter.
- **Visualization**: Placing data in a visual context.

Architecture

High level structure of the frontend code and the discipline of creating such structures.

- **Algorithms**: A self-contained step-by-step set of operations to be performed. Algorithms perform calculation, data processing, and/or automated reasoning tasks.
- **Design Patterns**: Best practices that the programmer can use to solve common problems when designing an application or system.
- **Designs**: Ready to use and well documented structures and frameworks for frontend development.
- **Event-Driven Programming**: Event-driven programming is a programming paradigm in which the flow of the program is determined by events such as user actions, sensor outputs, or messages from other programs/threads.
- **Functional Programming**: Functional programming is a programming paradigm, that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data.
- **Functional Reactive Programming (FRP)**: FRP is a programming paradigm for asynchronous dataflow programming using the building blocks of functional programming.

Compatibility

Ability of a product to work with different input/output devices and rendering software. Including printers, email

- **redux-transduce** - Transducer utilities for Redux
- **redux-actions** - Flux Standard Action utilities for Redux
- **redux-promise** - FSA-compliant promise middleware for Redux
- **redux-sync-promise** - Middleware for writing asynchronous actions in synchronous style
- **redux-simple-promise** - FSA-compliant promise middleware for Redux with simple behaviour with minimal boilerplate declarations
- **redux-async** - FSA-compliant promise property middleware for Redux
- **redux-async-queue** - Manage queues of thunk actions
- **redux-async-initial-state** - Set initial Redux state asynchronously
- **redux-await** - Manage async redux actions sanely
- **redux-rx** - RxJS utilities for Redux
- **reselect** - Selector library for Redux like in NuclearJS
- **react-redux** - React bindings for Redux
- **redux-react-router** - Redux bindings for React Router – keep your router state inside your Redux store
- **redux-promise-middleware** - Redux middleware for resolving and rejecting promises
- **redux-thunk** - Thunk middleware for Redux
- **redux-batched-updates** - Batch React updates that occur as a result of Redux dispatches, to prevent cascading renders.
- **redux-combine-actions** - Redux middleware that allows you to easy combine actions and dispatch them sequentially
- **redux-catch-promise** - Extended replacement of redux-thunk middleware to supporting async-await functions and implement server-side rendering for React components with async state
- **redux-delegator** - Compose redux reducers in a structured way
- **routex** - Simple router for Redux universal applications. Can be used with React too
- **redux-persist-store** - Persist and rehydrate a redux store
- **adrenaline** - React bindings for Redux with Relay in mind
- **redux-localstorage** - Store enhancer that syncs (a subset) of your Redux store state to localstorage.
- **redux-storage** - Persistence layer for redux with flexible backends
- **redux-pouchdb** - sync store state to pouchdb
- **redux-vstack-router** - Helpers to bind vstack-router to redux
- **redux-create-store**
- **redux-batched-subscribe** - Batch calls to subscribe handlers with a custom function, including debouncing or React batched updates

Awesome list 的缺陷

- 『收集』会越『集』越多，列表膨胀，『curation』失效
- 每个局部都可以独立膨胀，列表本身越来越多，彼此割裂
- Awesome list of Awesome lists

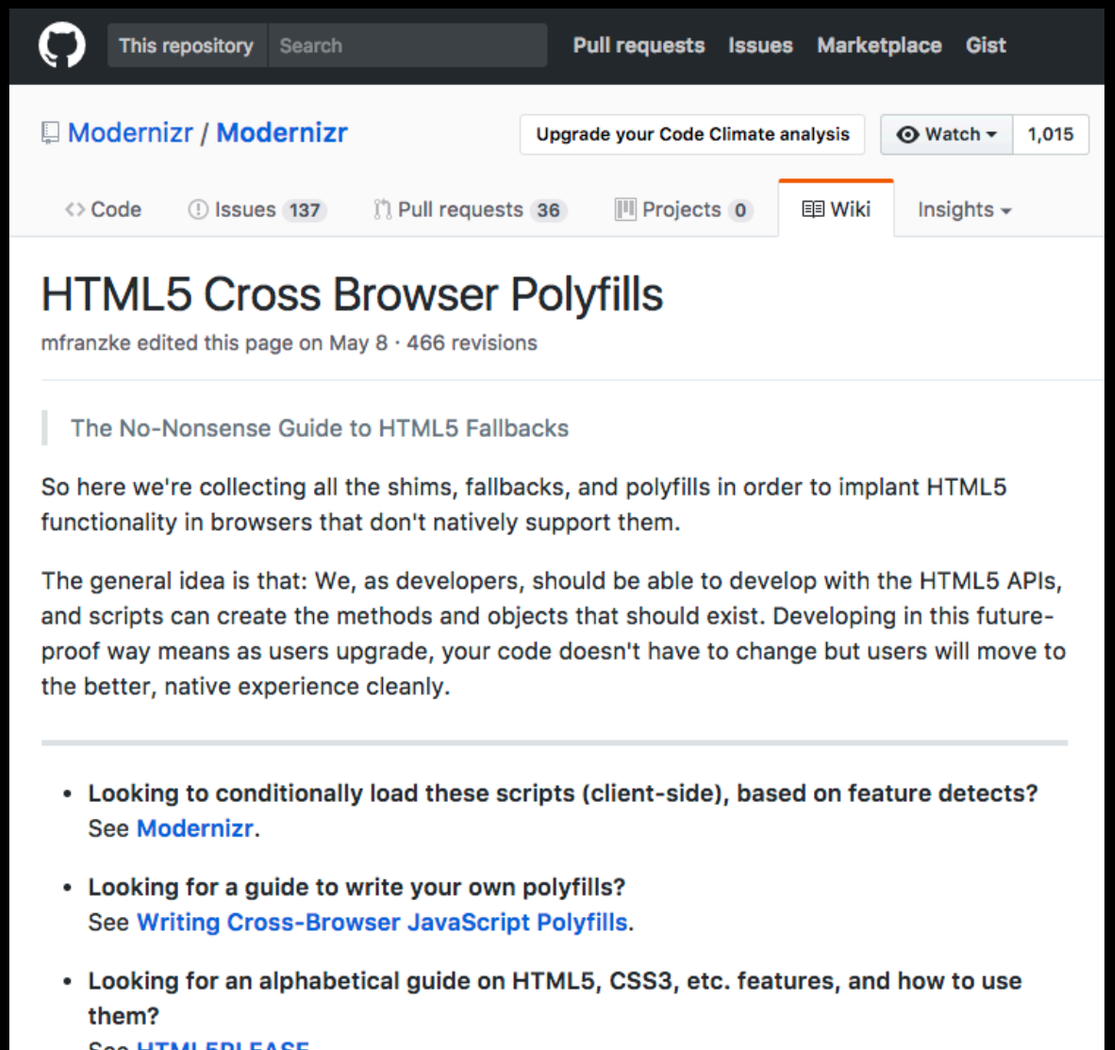
awesome-awesome-awesome

You need it, Luke.

- https://github.com/sindresorhus/awesome - above 30k stars
- https://github.com/bayandin/awesome-awesomeness - above 10k stars
- https://github.com/oyvindrobertsen/awesome-awesome
- https://github.com/fleveque/awesome-awesomes
- https://github.com/erichs/awesome-awesome
- https://github.com/emijrp/awesome-awesome

Awesome list 的缺陷

- 『收集』会越『集』越多，列表膨胀，『curation』失效
- 每个局部都可以独立膨胀，列表本身越来越多，彼此割裂
- Awesome Awesomes (Awesome list of Awesome lists)
- 无效的、过时的『curation』太多



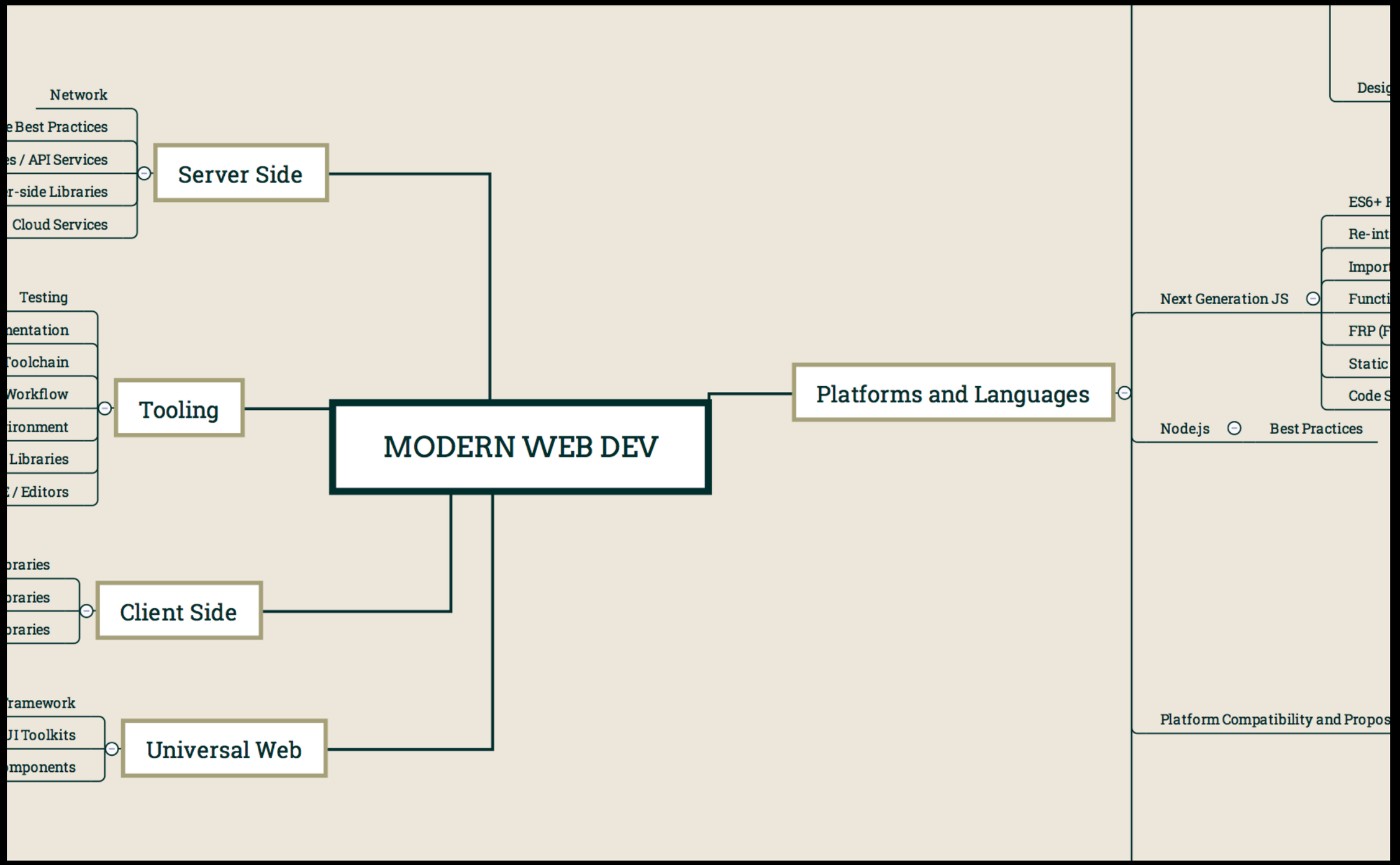
Awesome list 的缺陷

- 『收集』会越『集』越多，列表膨胀，『curation』失效
- 每个局部都可以独立膨胀，列表本身越来越多，彼此割裂
- Awesome Awesomes (Awesome list of Awesome lists)
- 无效的、过时的『curation』太多
- 描述/评论冗长重复抓不住重点
- 缺少『关系』和『结构』

Spellbook 的方法

- 整体视角
- 分类和嵌套
- 控制数量
- 数据分析

Spellbook 的方法：整体视角



Spellbook 的方法：分类和嵌套

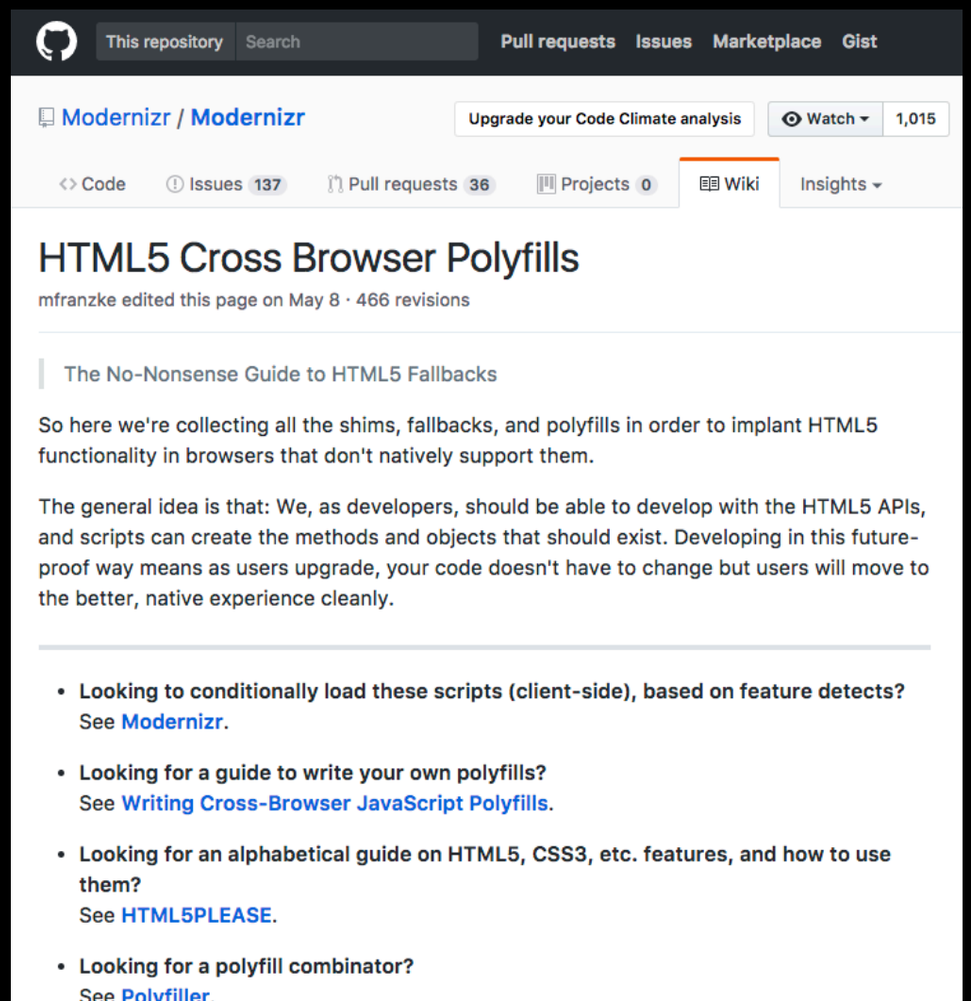
- 以『类别』作为组织单位
- 深度嵌套到非常细粒度的『类别』
- 从一维的列表变成二维的树状结构
- 知识的『关系』是一种最重要的知识

Cloud Services (Global)

- Compute
  - FaaS / Serverless / WebHook
    - AWS Lambda / Google Cloud Functions
    - webtask / hook.io
    - Graphcool Functions
    - Amazon API Gateway
  - PaaS
    - See *Tooling > Workflow* > *Deployment* > *DevOps* > *PaaS*
  - CaaS
    - Amazon ECS / Google Container Engine
- Storage
  - Object Storage
    - Amazon S3 / Google Cloud Storage
    - imgix
  - DBaaS
    - In-Memory Key-Value NoSQL - Amazon ElastiCache
      - Redis - Compose / Redise Cloud / Heroku Redis
    - Document NoSQL - Amazon DynamoDB / Google Cloud Datastore
      - MongoDB - Compose / mLab / MongoDB Atlas
      - CouchDB - Couchbase / Cloudant
    - Wide Column NoSQL - Google Bigtable
    - SQL - Amazon RDS / Google Cloud SQL
      - PostgreSQL - Compose / Heroku Postgres
      - MySQL - Compose
    - NewSQL - Google Cloud Spanner
    - Queue - Amazon SQS / Amazon Kinesis / Google Cloud Pub/Sub
      - Kafka - Heroku Kafka
      - RabbitMQ - Compose
    - Analytics - Amazon CloudSearch
      - Elasticsearch - Amazon Elasticsearch Service / Elastic Cloud / Bonsai
    - Warehouse - Amazon Redshift / Google BigQuery
- BaaS
  - CRUD
    - Realtime
      - Firebase Realtime Database
    - GraphQL
      - Graphcool / Scaphold
    - CMS
      - WordPress.com REST API / Contentful / DatoCMS / GraphCMS / Baasic
  - Auth

Spellbook 的方法：控制数量

- 最细粒度的『类别』提供一个或少数几个链接（替换关系、补充关系）
  - 不过时
  - 使用最多



VS

Cross-browser / Polyfill Libraries

- Appearance
  - Responsive Web Design
    - Media Queries - Enquire.js
    - Responsive Image - Picturefill
    - Viewport Units Buggyfill
  - Web Typography
    - `@font-face` - Font Face Observer
  - Web Animation API
    - Web Animations Polyfill
  - Web Components
    - webcomponents.js (v1 spec polyfills) / Polymer
- Interaction
  - Keyboard - Mousetrap
  - `scroll-behavior: smooth;` - Smoothscroll Polyfill
  - PointerEvent - PEP / React Pointable
  - ResizeObserver Polyfill
- Access
  - Web Notifications API - Push.js / Notify.js
  - Clipboard API - Clipboard.js / copy-to-clipboard
  - Fullscreen API - Screenfull
  - Page Visibility API - Visibility.js
  - `<iframe>` - iframe-resizer
- Network
  - XHR - window.fetch Polyfill
  - Server-Sent Events - EventSource Polyfill
  - WebSocket - Socket.IO-client / Engine.IO-client / SockJS-client
- Performance
  - `document-write` - PostScribe
  - User Timing API - marky
- Offline
  - Service Work - sw-toolbox
  - File / FileReader API
    - FileSaver.js
    - blob-util
  - IndexedDB
    - LocalStorage API - localForage
- Media
  - `<video>` - Video.js
  - `<audio>` - Howler.js
  - Web Audio API - Waud.js, Tone.js

>> Return to Table of Contents

Spellbook 的方法：数据分析

- npm 下载数据的权重高于 github 仓库数据（star、fork...）
- 对影响下载数据的因素给予不同权重
- 对列表数据的抓取、分析和跟踪

导读

1. 语言和平台
2. Universal Web
3. 客户端（省略）
4. 服务器端
5. 工具

#table-of-contents

导读：语言和平台

- JS 的三大优势
- CSS 的最大趋势
- Web 开源生态

JS 的三大优势

- API
- 多范式
- 混合运行

JS 的三大优势：API

- 独家 API：Open Web 平台（DOM API、HTML5 API）
  #open-web-platform #html5-features #css-features #next-generation-js
- 胶水 API：Node.js API、Hybrid API、……
  #nodejs #platform-compatibility-and-proposal-status
- 提案和工作组：W3C WG、WICG、WHATWG、ECMA TC39、Node.js CTC
  #platform-compatibility-and-proposal-status

JS 的三大优势：多范式

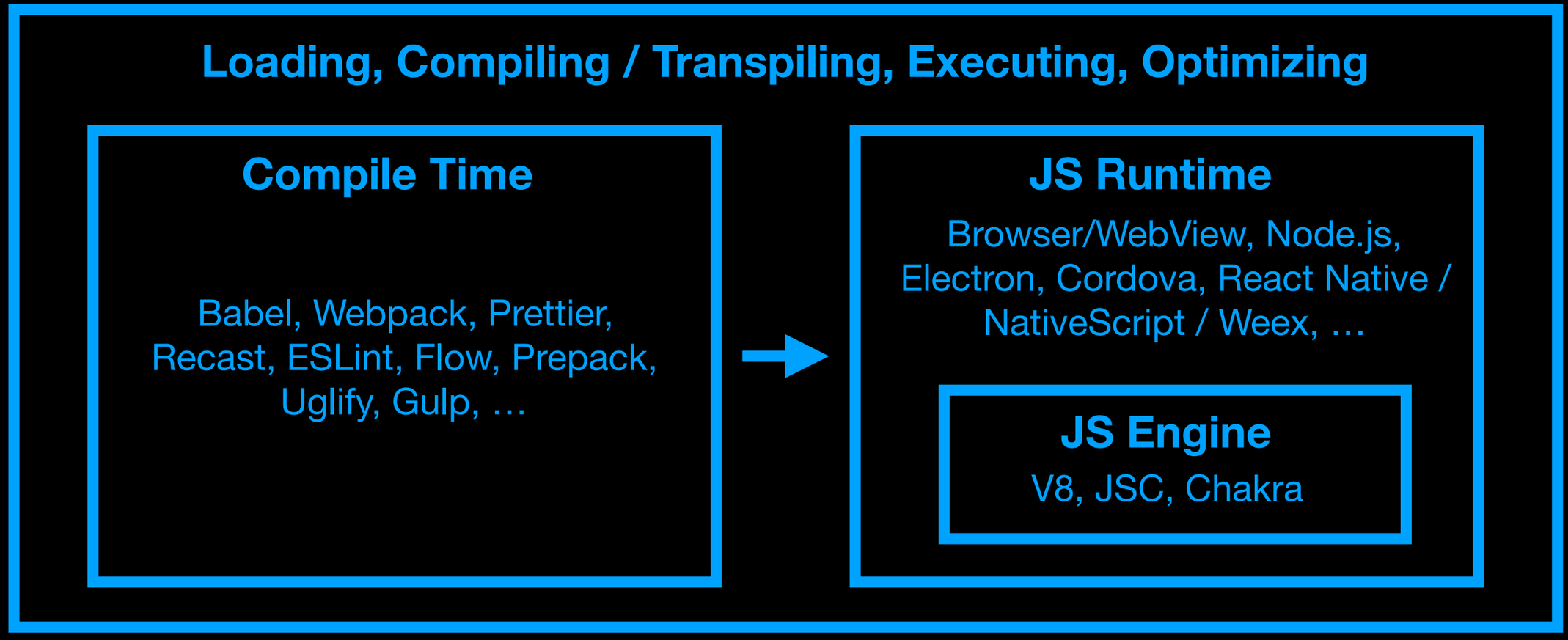
- 异步范式的 DNA
  #next-generation-js #universal-utility-libraries
- 第一个大规模应用、进入主流的 FP 语言
  #next-generation-js #universal-utility-libraries
- Prototypes vs Classes, Composition vs Inheritance
  #next-generation-js
- Static Typing
  #next-generation-js #toolchain

“OOP to me means only messaging.”

– Alan Kay (prototypes vs classes)

JS 的三大优势：混合运行

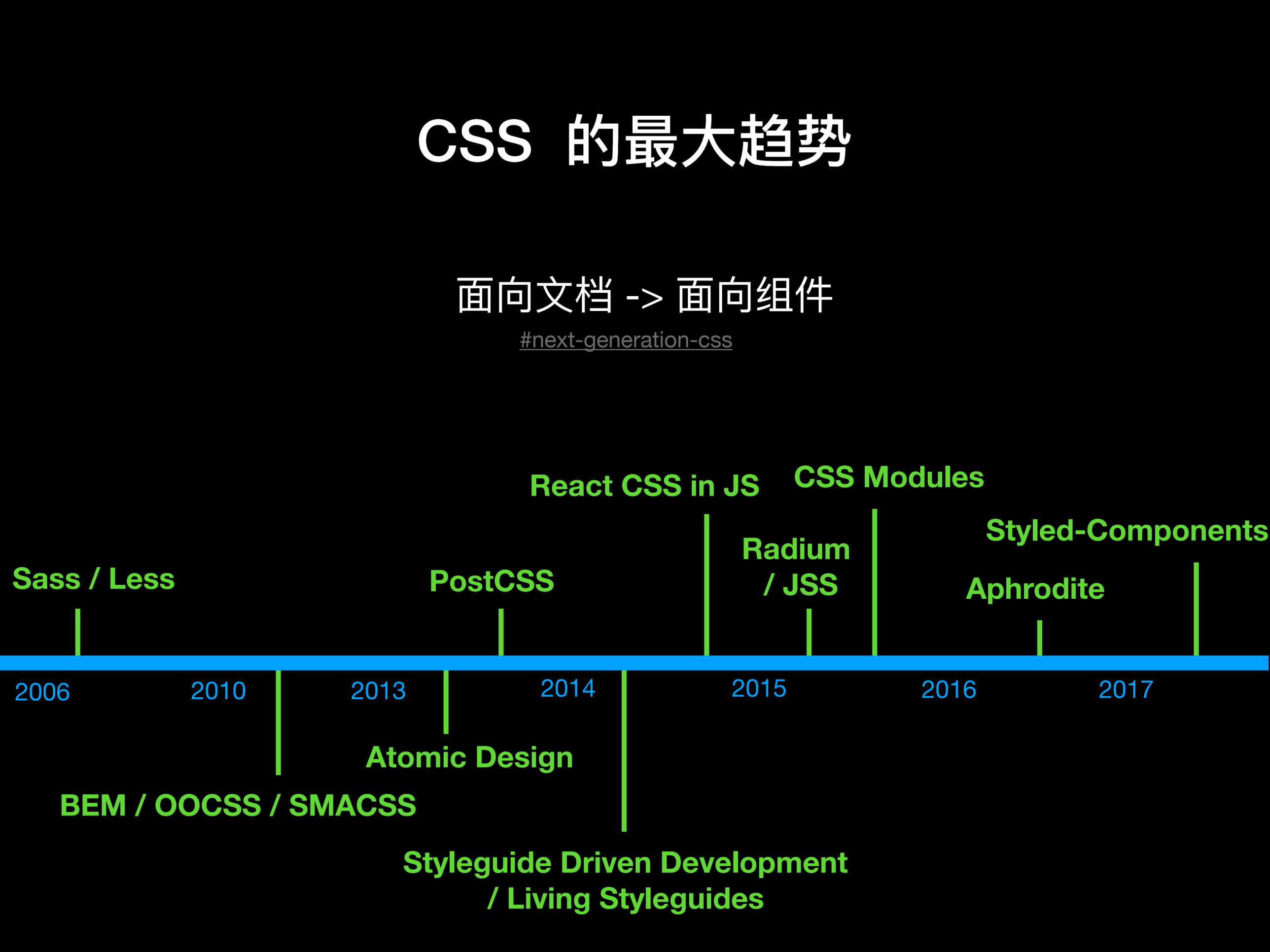
- 引擎 + 运行时 + 静态工具链
  #platform-compatibility-and-proposal-status　#toolchain
- 编译时与运行时、静态与动态的融合



CSS 的最大趋势

面向文档 -> 面向组件

#next-generation-css



Web 开源生态

- web 开源生态 = npm 生态
  #npm-ecosystem
- 五大流派
  #npm-ecosystem #standalone-ui-components #ui-toolkits

All-in-one

jQuery, AngularJS, Meteor, Sails, …

Composable

Express, React, Redux, Babel, PostCSS, …

Small Modules / Standalone Components

qs, uuid, EventEmitter3, winston, React Select, React DnD, …

Hyper Modular / One-line Modules

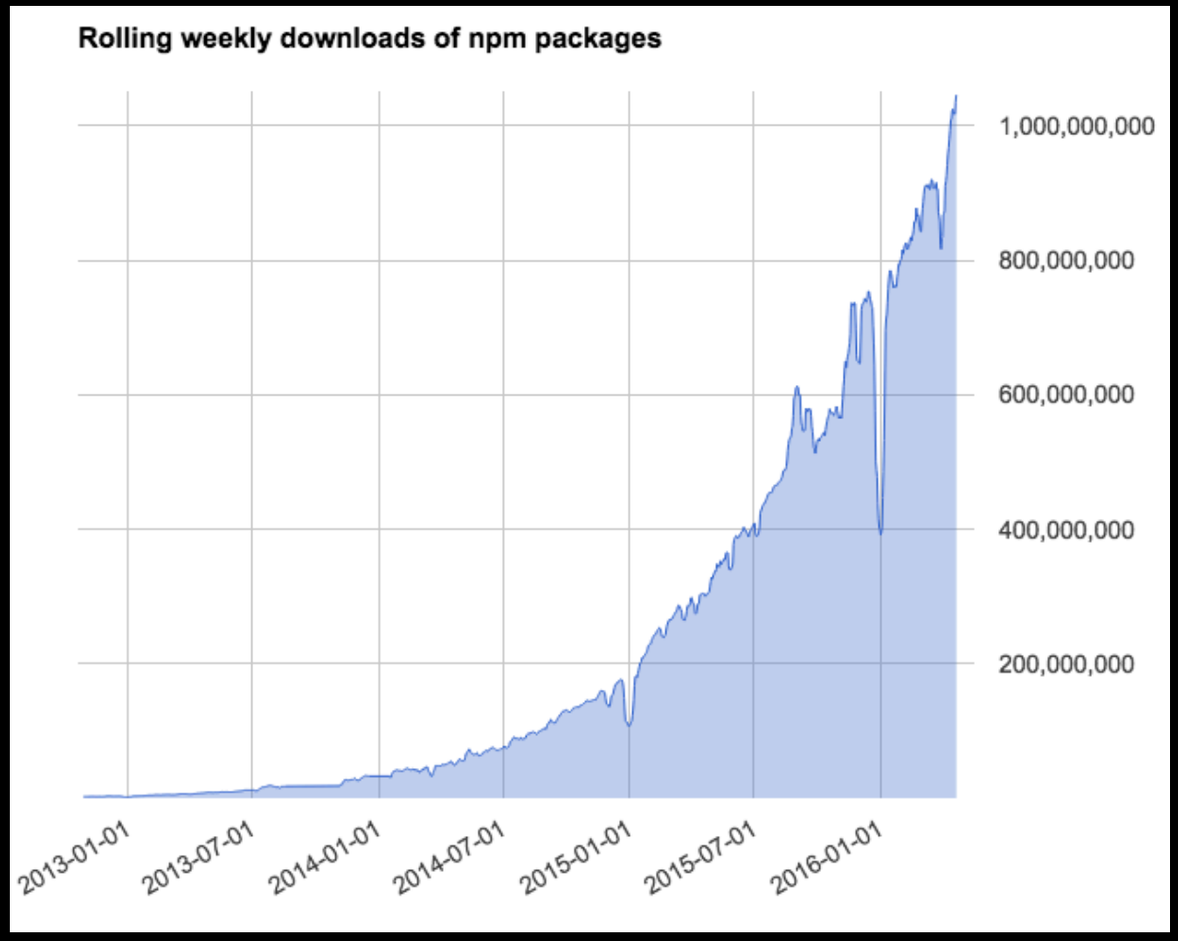
is-electron-renderer, left-pad, …

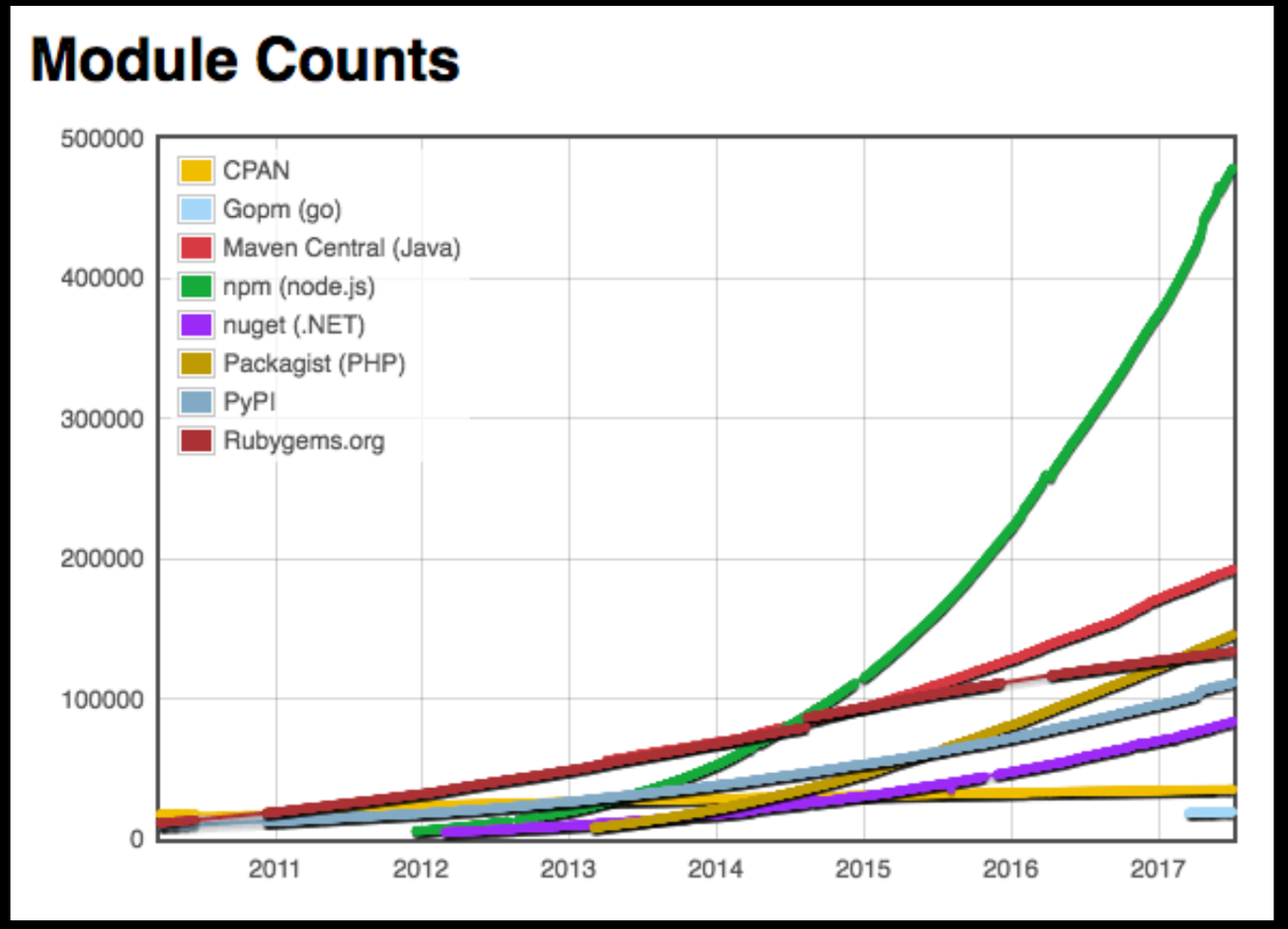
Toolkit

Lodash, Moment, Bluebird, fs-extra, Material-UI, …

Web 开源生态

- web 开源生态 = npm 生态
  #npm-ecosystem
- 五大流派
  #npm-ecosystem #standalone-ui-components #ui-toolkits
- Universal JS
  #npm-ecosystem #universal-utility-libraries





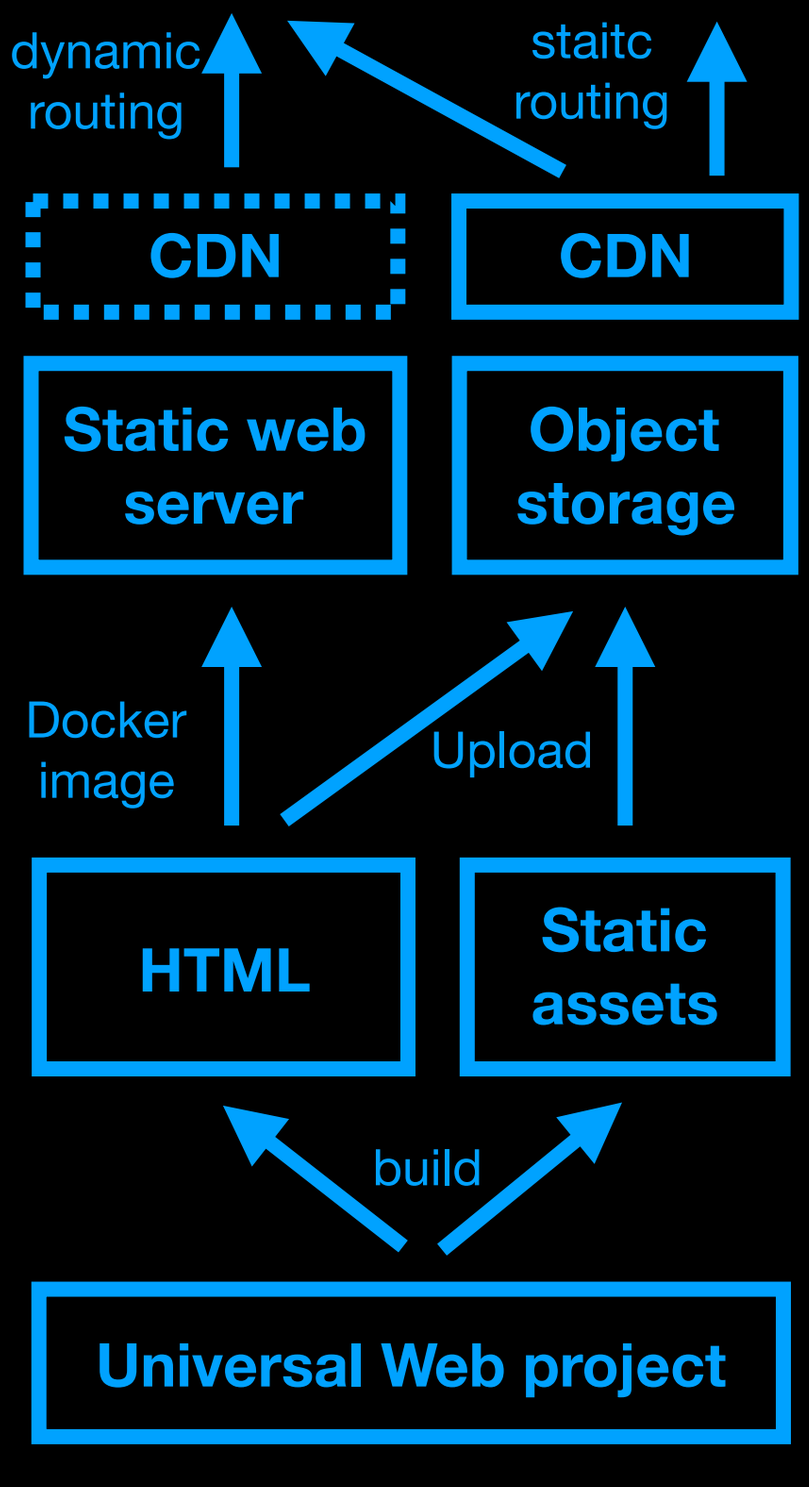
导读：Universal Web

#universal-web-apps--web-pages #gui-framework

- 三种部署模式
- 从 HTML 到 JS
- 从 OOP 到 FP

三种部署模式

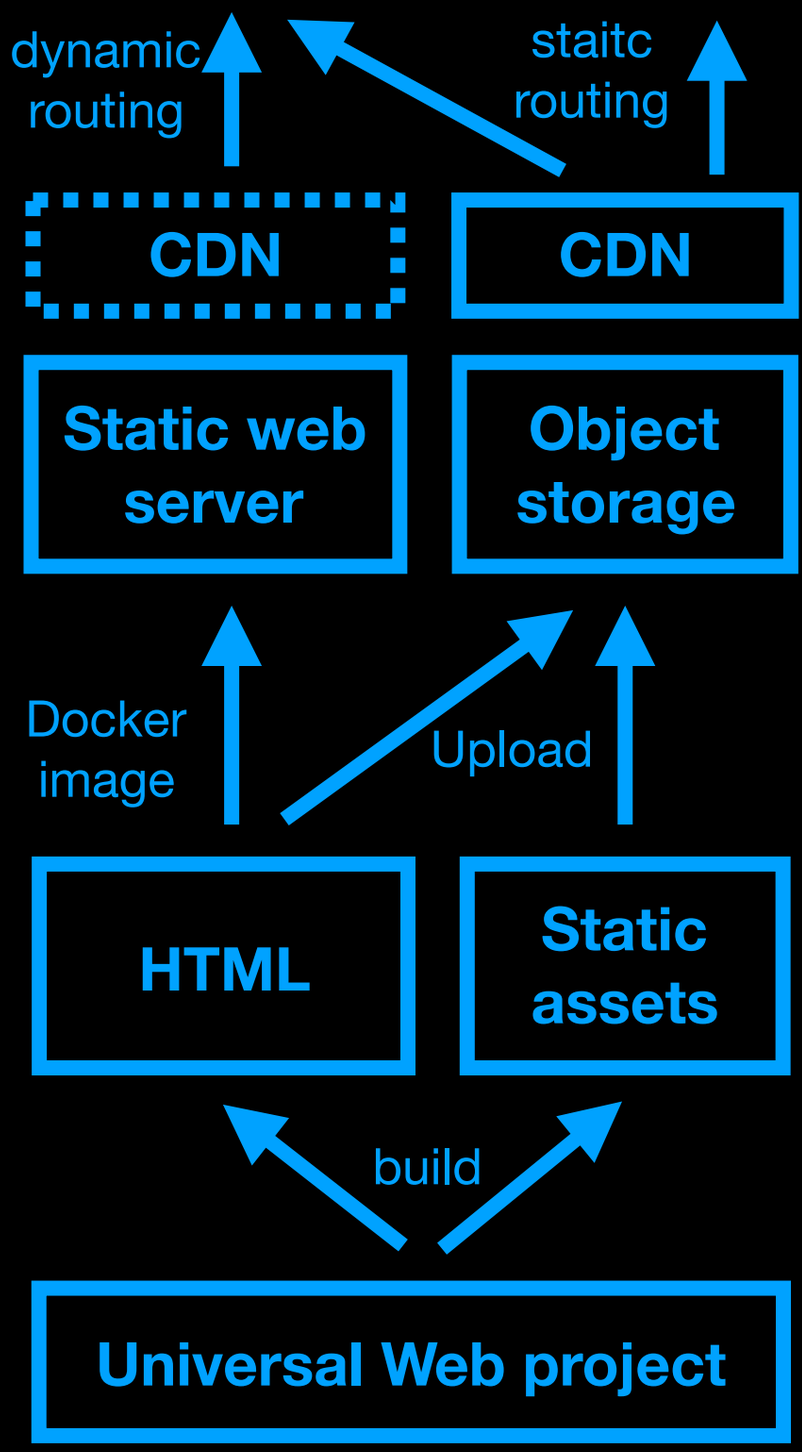
#workflow



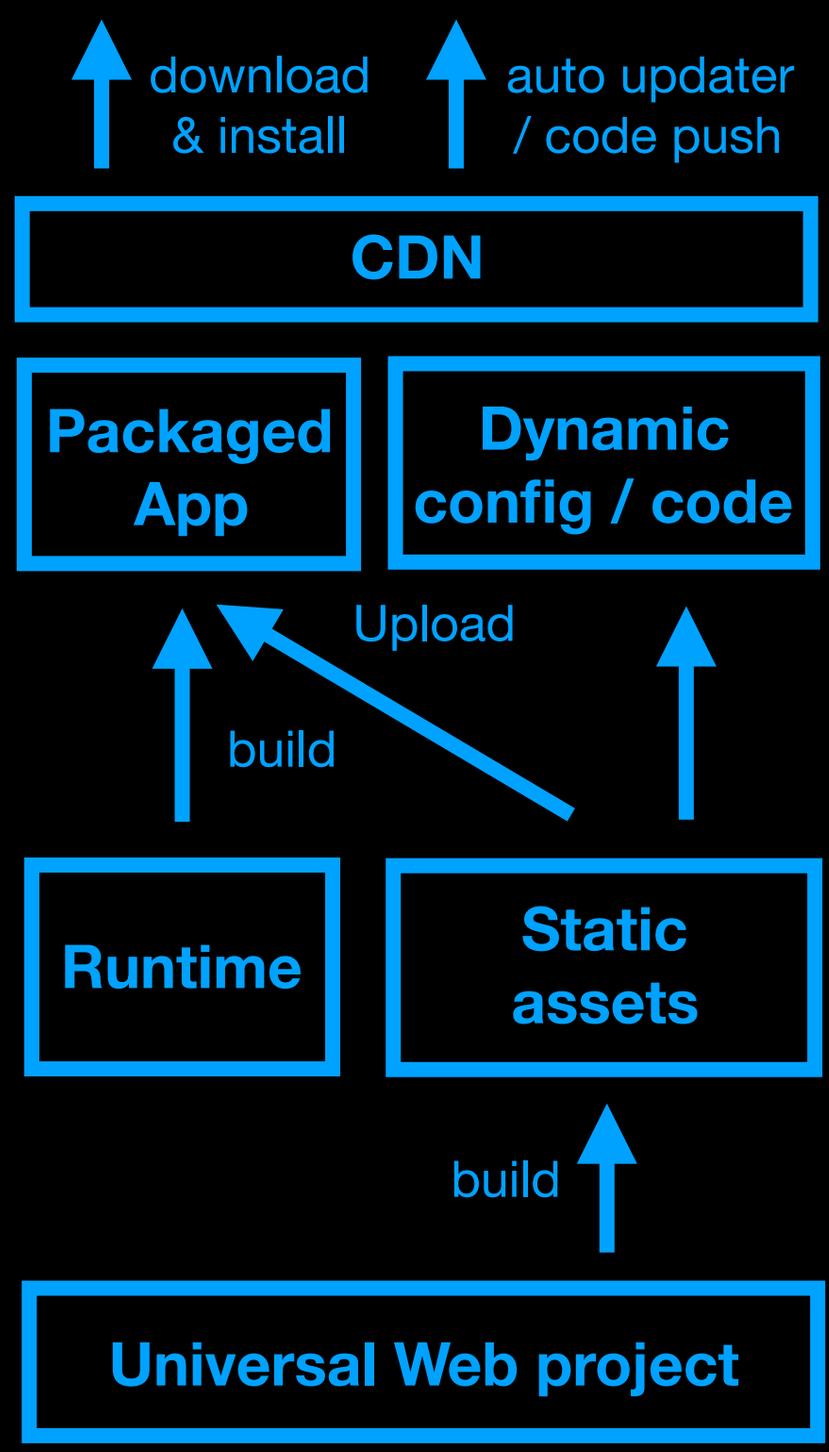
Static Web

三种部署模式

#workflow



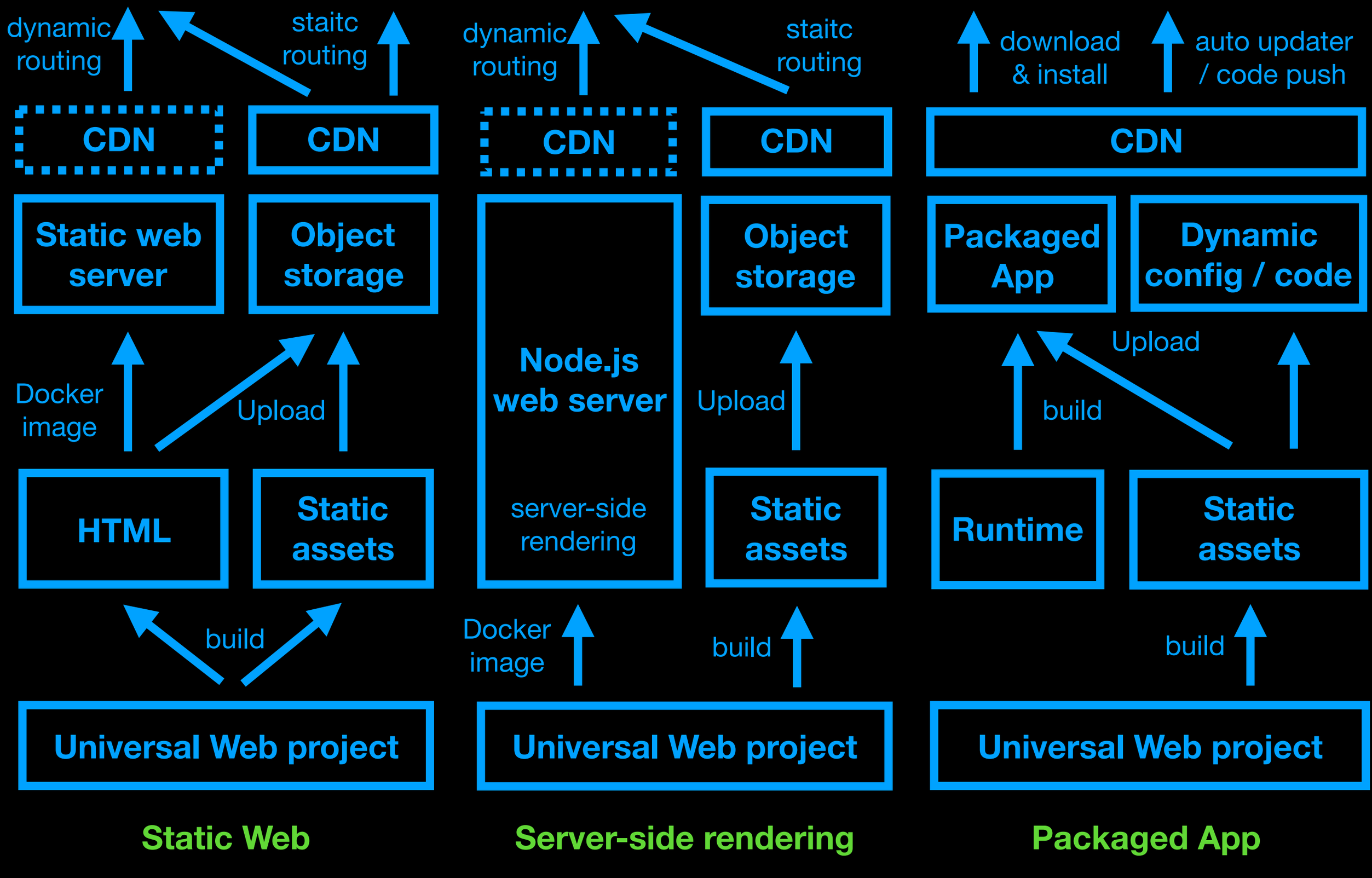
Static Web



Packaged App

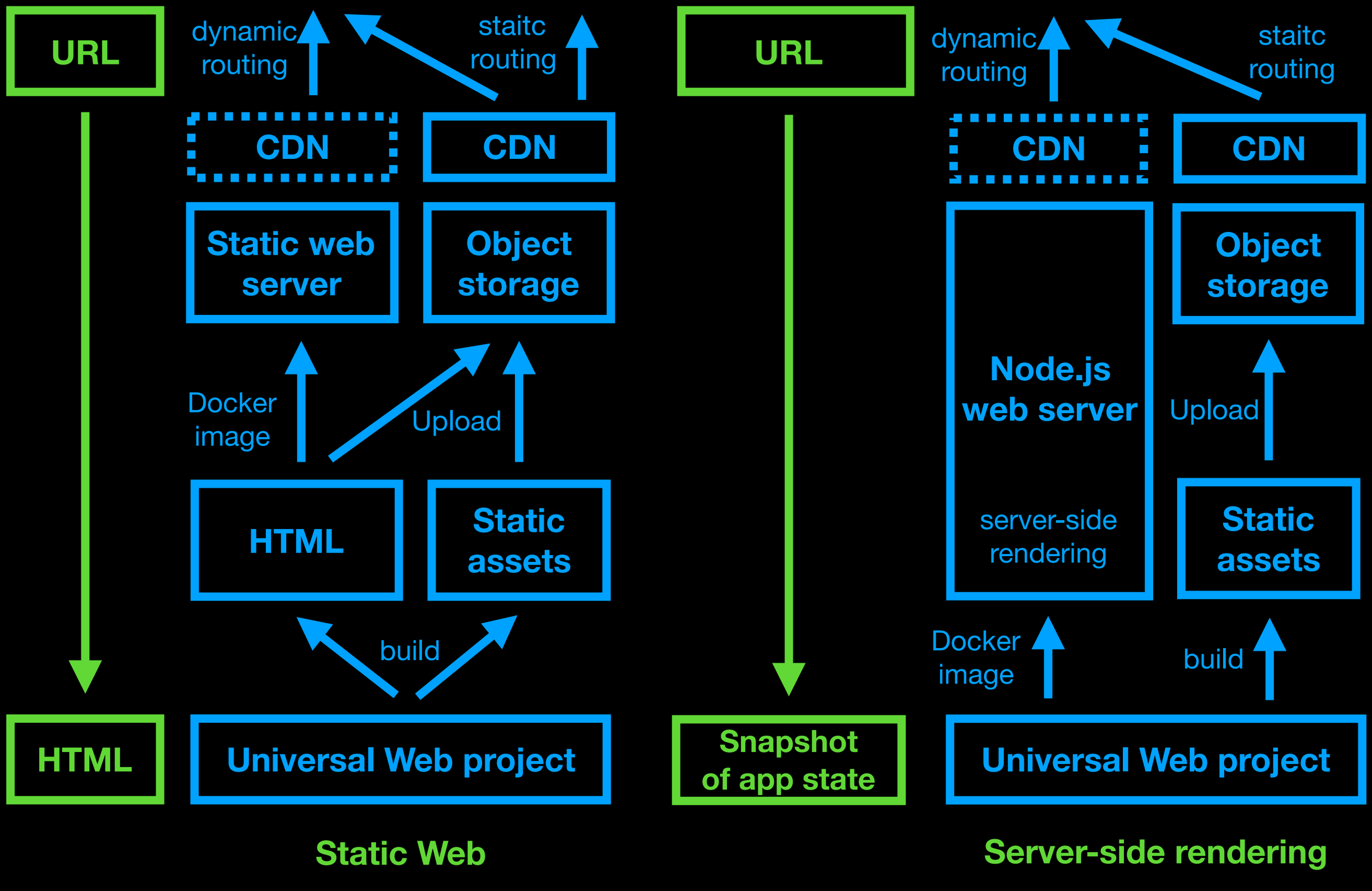
三种部署模式

#workflow



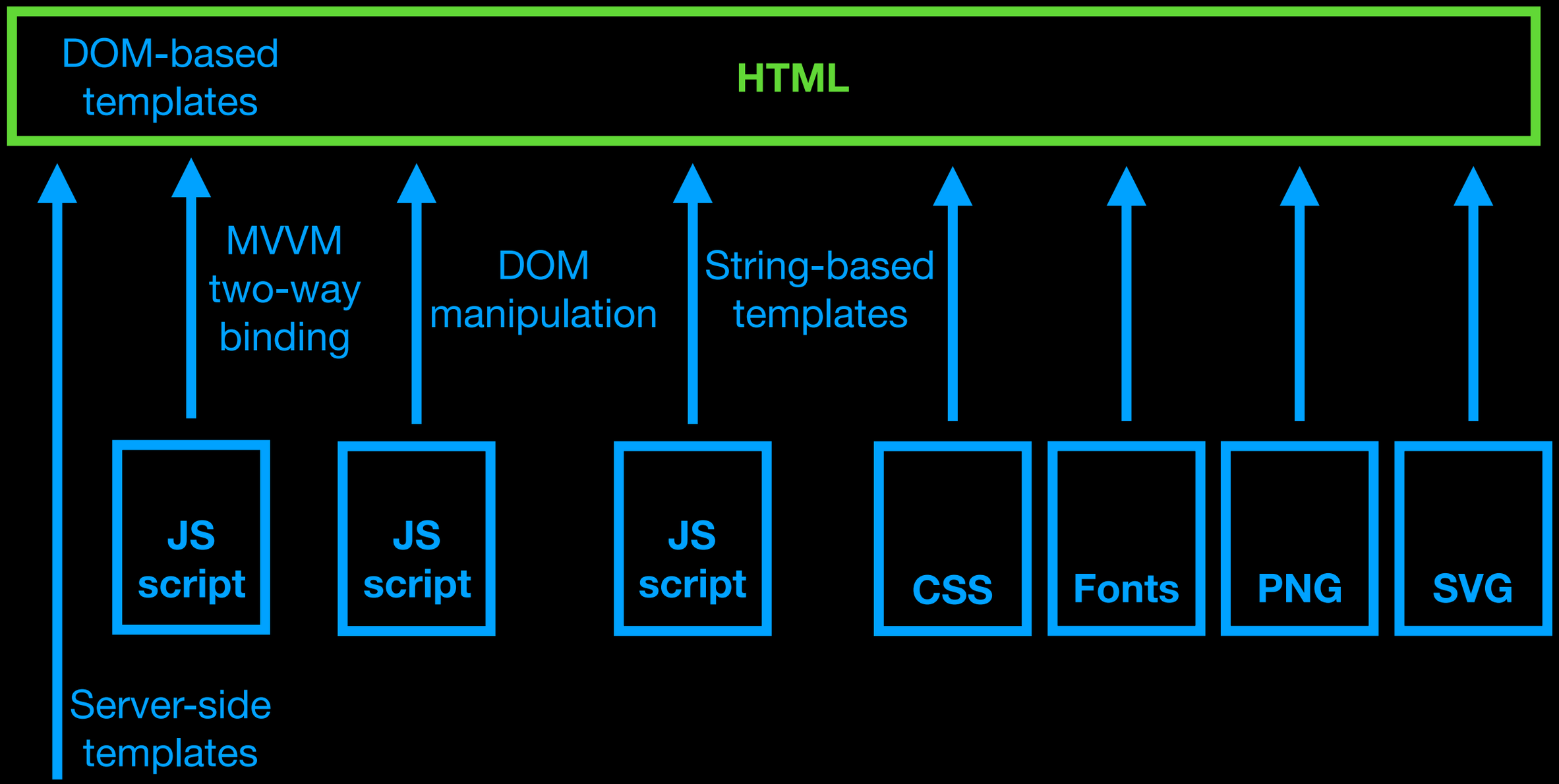
三种部署模式

#workflow



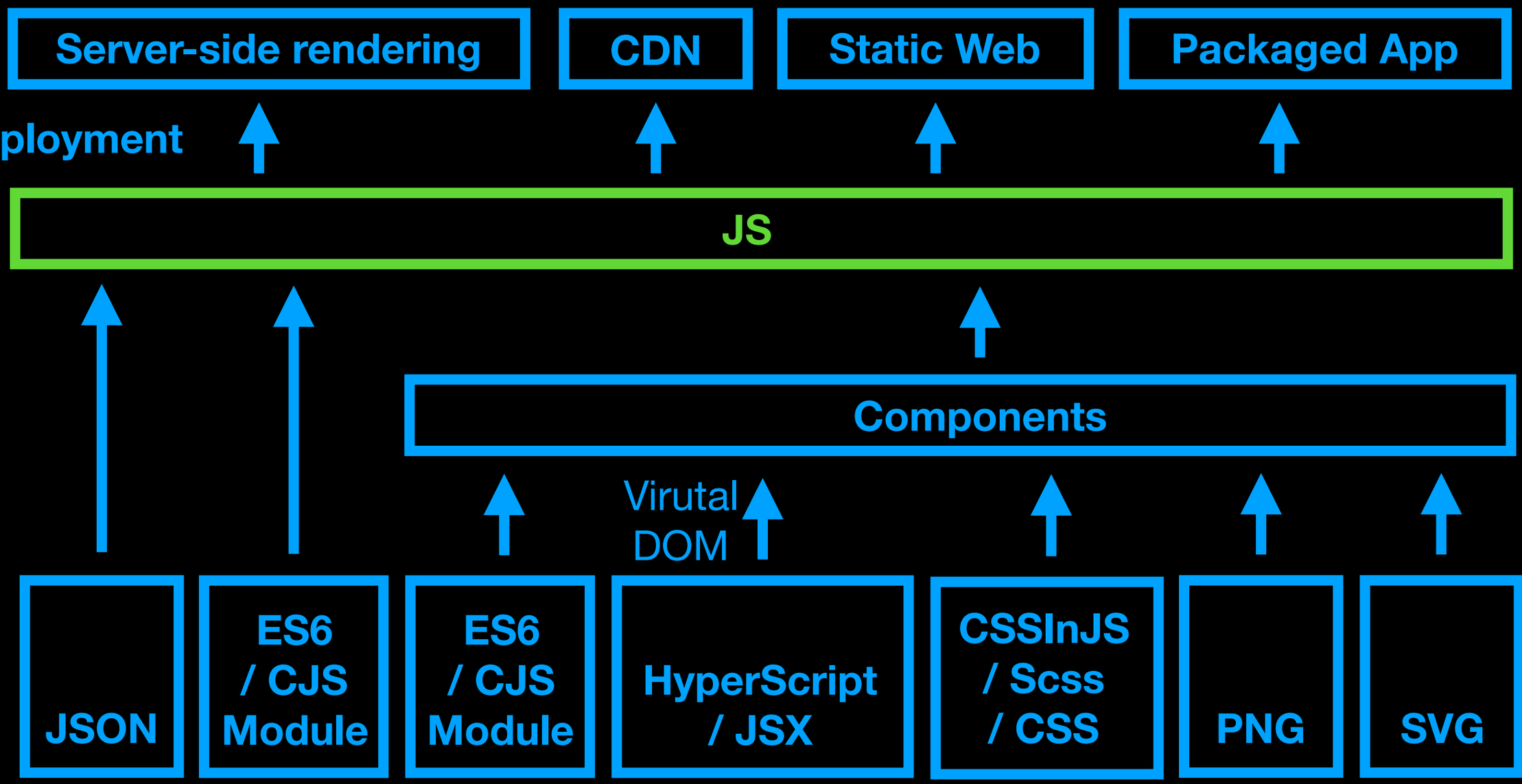
从 HTML 到 JS

传统 web 开发：以 HTML 为中心



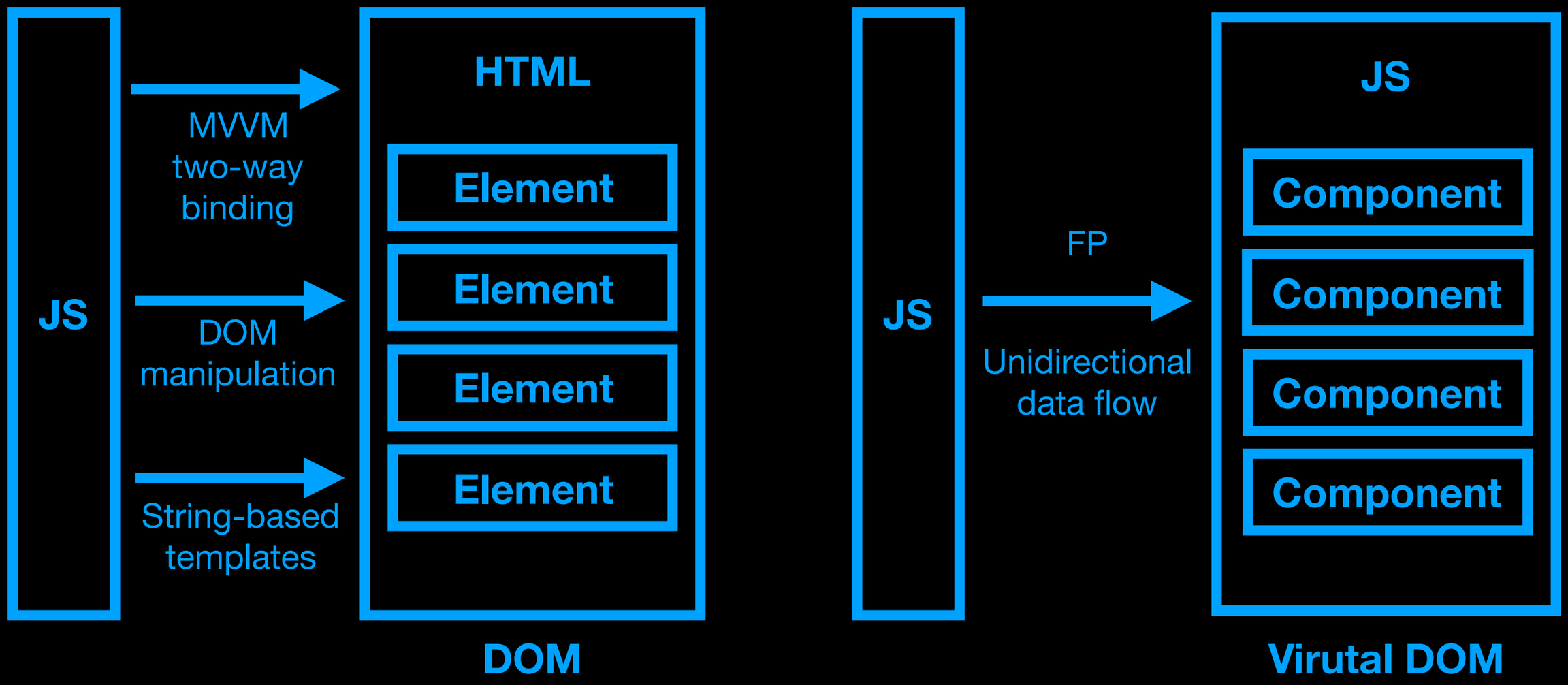
从 HTML 到 JS

（以 React 为代表的）现代 web 开发：以 JS 为中心



从 HTML 到 JS

面向文档元素 -> 面向组件

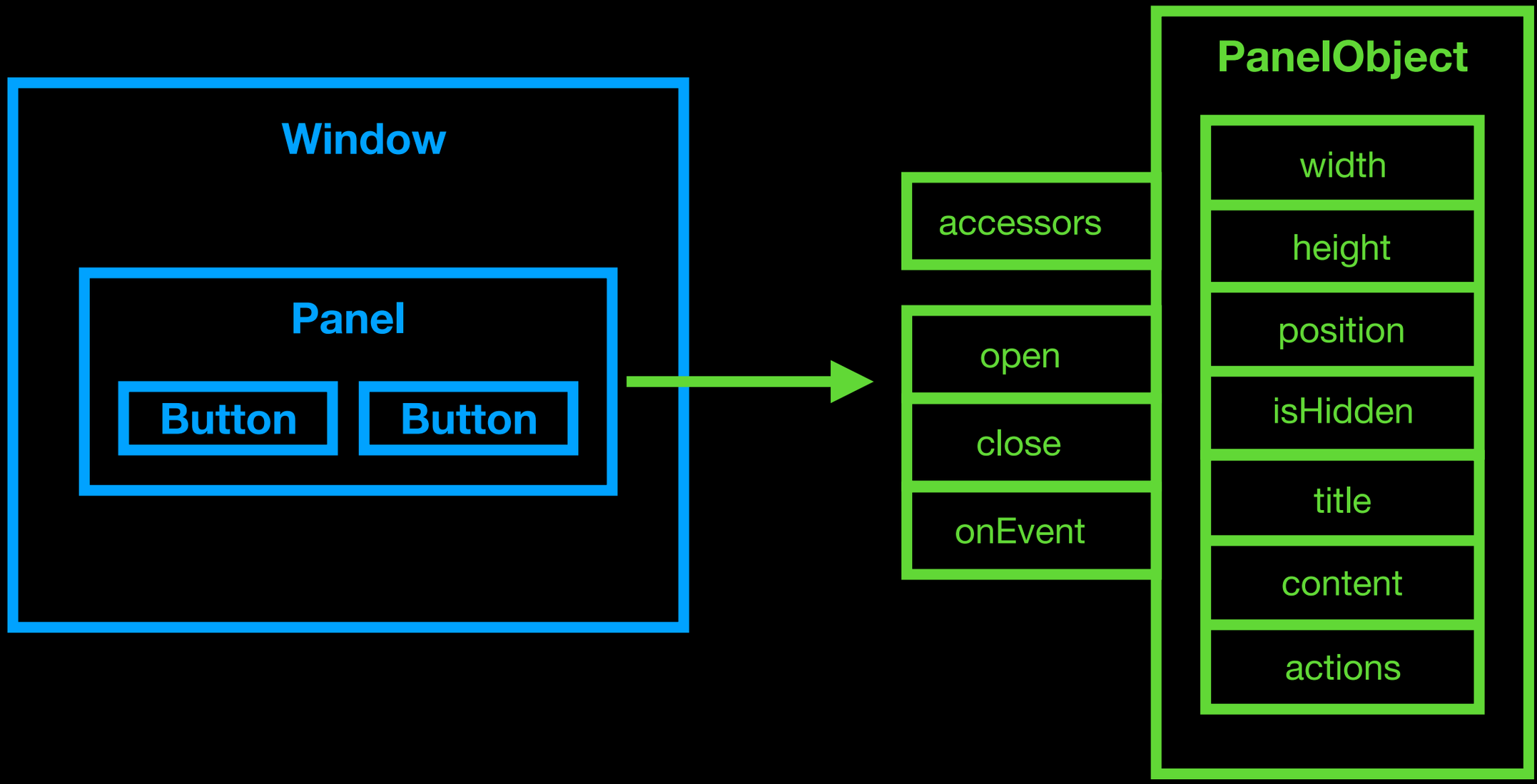


当 Universal JS 遇到 Virutal DOM

- Server-side rendering
- React Native
- Enzyme
- React Sketch.app
- …

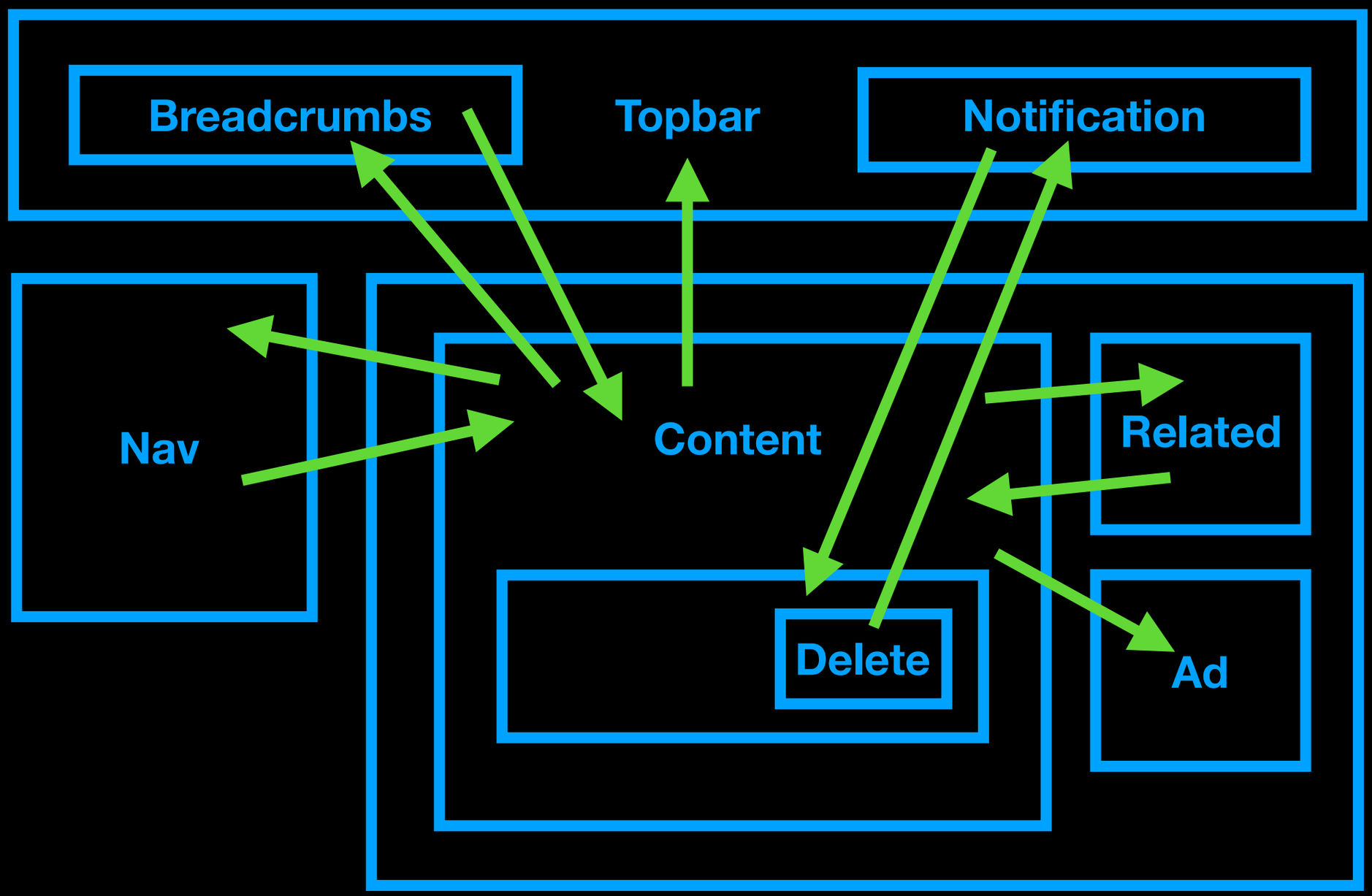
从 OOP 到 FP

传统 GUI 开发由 OOP 主导



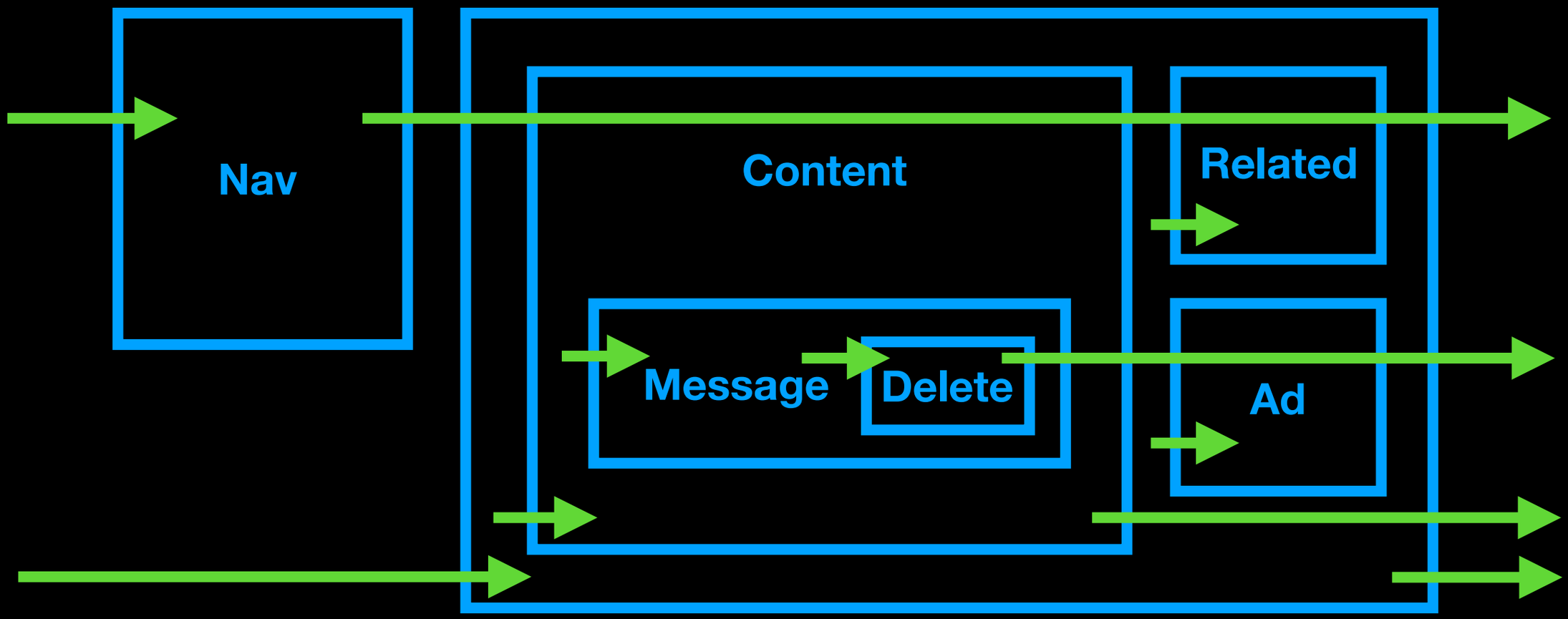
从 OOP 到 FP

对象间消息传递 -> 单向数据流



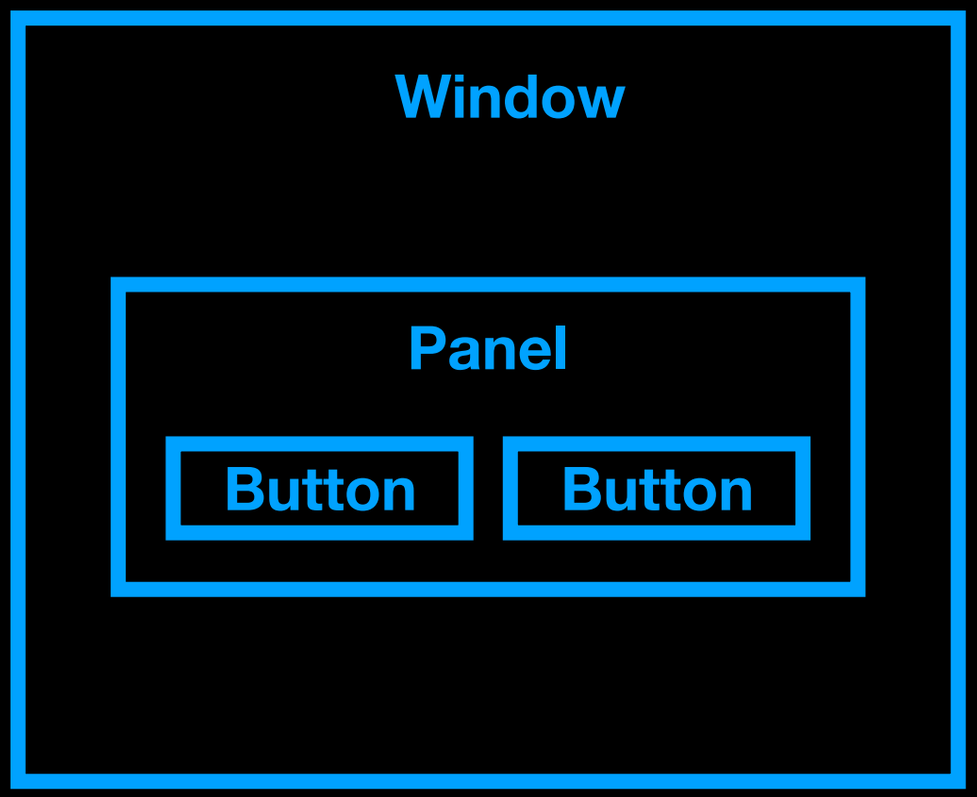
从 OOP 到 FP

对象间消息传递 -> 单向数据流



从 OOP 到 FP

函数式 GUI 编程



导读：服务器端

- 基础设施的服务化
- 业务逻辑的服务化
- 数据智能的服务化
- Microservice / Serverless
  + API Gateway

基础设施的服务化：存储

- Object Storage：万用服务
  #cloud-services-global　#cloud-services-china

基础设施的服务化：存储

- Object Storage：万用服务
- DBaaS

#cloud-services-global　#cloud-services-china

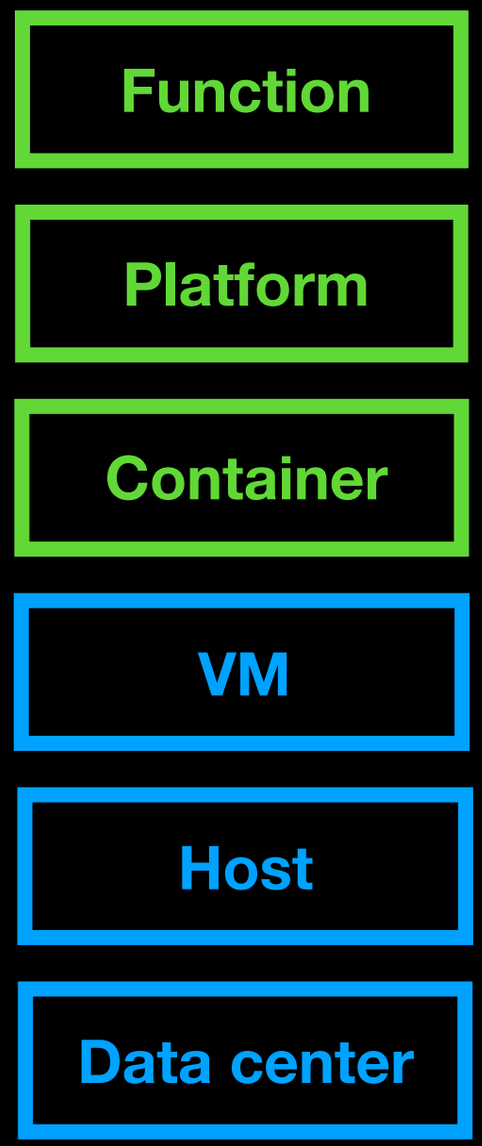
- In-Memory Key-Value NoSQL - Amazon ElastiCache
  - Redis - Compose / Redise Cloud / Heroku Redis
- Document NoSQL - Amazon DynamoDB / Google Cloud Datastore
  - MongoDB - Compose / mLab / MongoDB Atlas
  - CouchDB - Couchbase / Cloudant
- Wide Column NoSQL - Google Bigtable
- SQL - Amazon RDS / Google Cloud SQL
  - PostgreSQL - Compose / Heroku Postgres
  - MySQL - Compose
- NewSQL - Google Cloud Spanner
- Queue - Amazon SQS / Amazon Kinesis / Google Cloud Pub/Sub
  - Kafka - Heroku Kafka
  - RabbitMQ - Compose
- Analytics - Amazon CloudSearch
  - Elasticsearch - Amazon Elasticsearch Service / Elastic Cloud / Bonsai
- Warehouse - Amazon Redshift / Google BigQuery

- In-Memory Key-Value NoSQL
  - Redis - 阿里云-云数据库 Redis 版 / 腾讯云-云存储 Redis
- Document NoSQL
  - MongoDB - 阿里云-云数据库 MongoDB版 / 腾讯云-文档数
- Wide Column NoSQL - 阿里云-表格存储 OTS
  - HBase - 阿里云-云数据库 HBase 版 / 腾讯云-列式数据库 H
- SQL
  - PostgreSQL - 阿里云-云数据库 PostgreSQL 版 / 腾讯云-云
  - MySQL - 阿里云-云数据库 MySQL 版 / 腾讯云-云数据库 C
- Queue - 阿里云-消息服务 MNS / 腾讯云-消息服务 CMQ
  - Kafka - 腾讯云-消息服务 CKAFKA
- Analytics - 阿里云-开放搜索 OpenSearch / 腾讯云搜 TCS
- Warehouse - 阿里云-MaxCompute (ODPS) / 腾讯云-大数据处理

基础设施的服务化：计算

#cloud-services-global #cloud-services-china

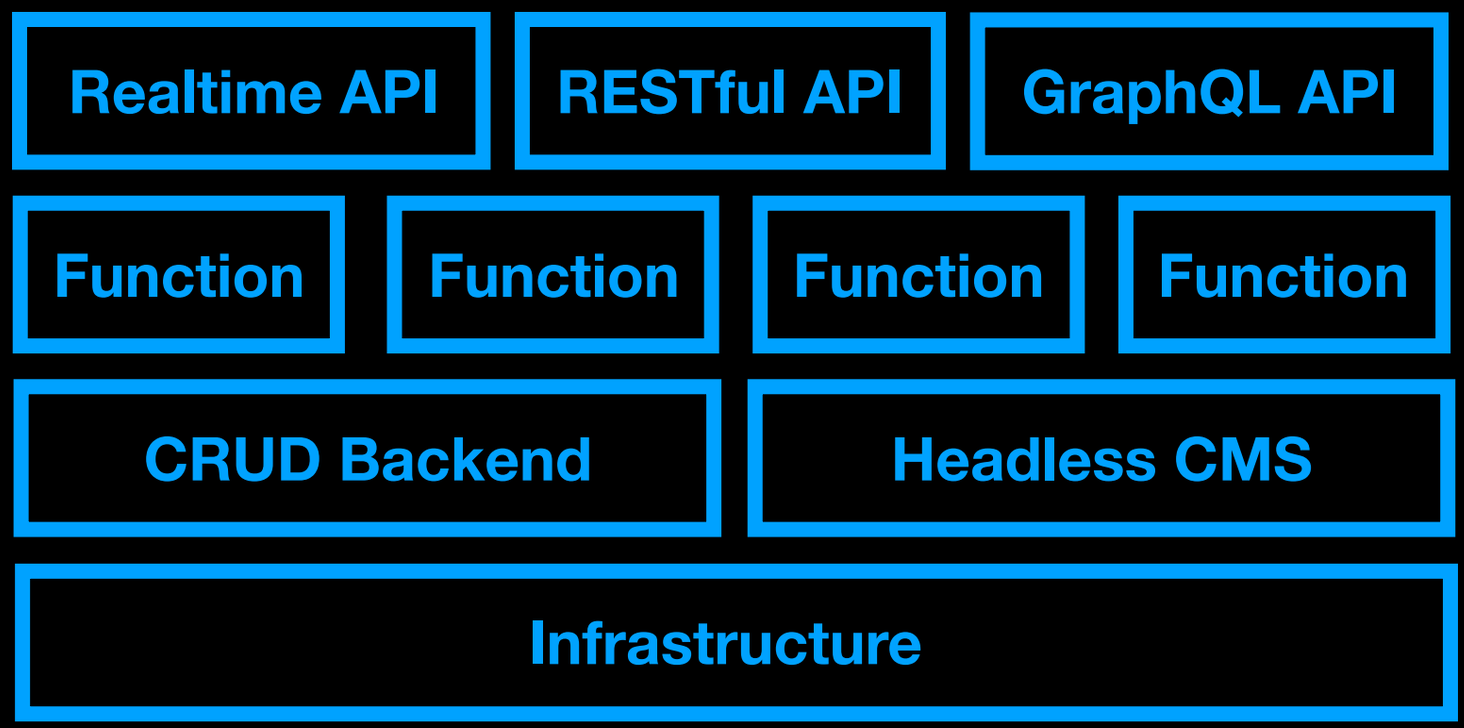
- CaaS
- PaaS
- FaaS / Serverless / WebHook
  - AWS Lambda 是 计算领域的 S3



业务逻辑的服务化：CRUD

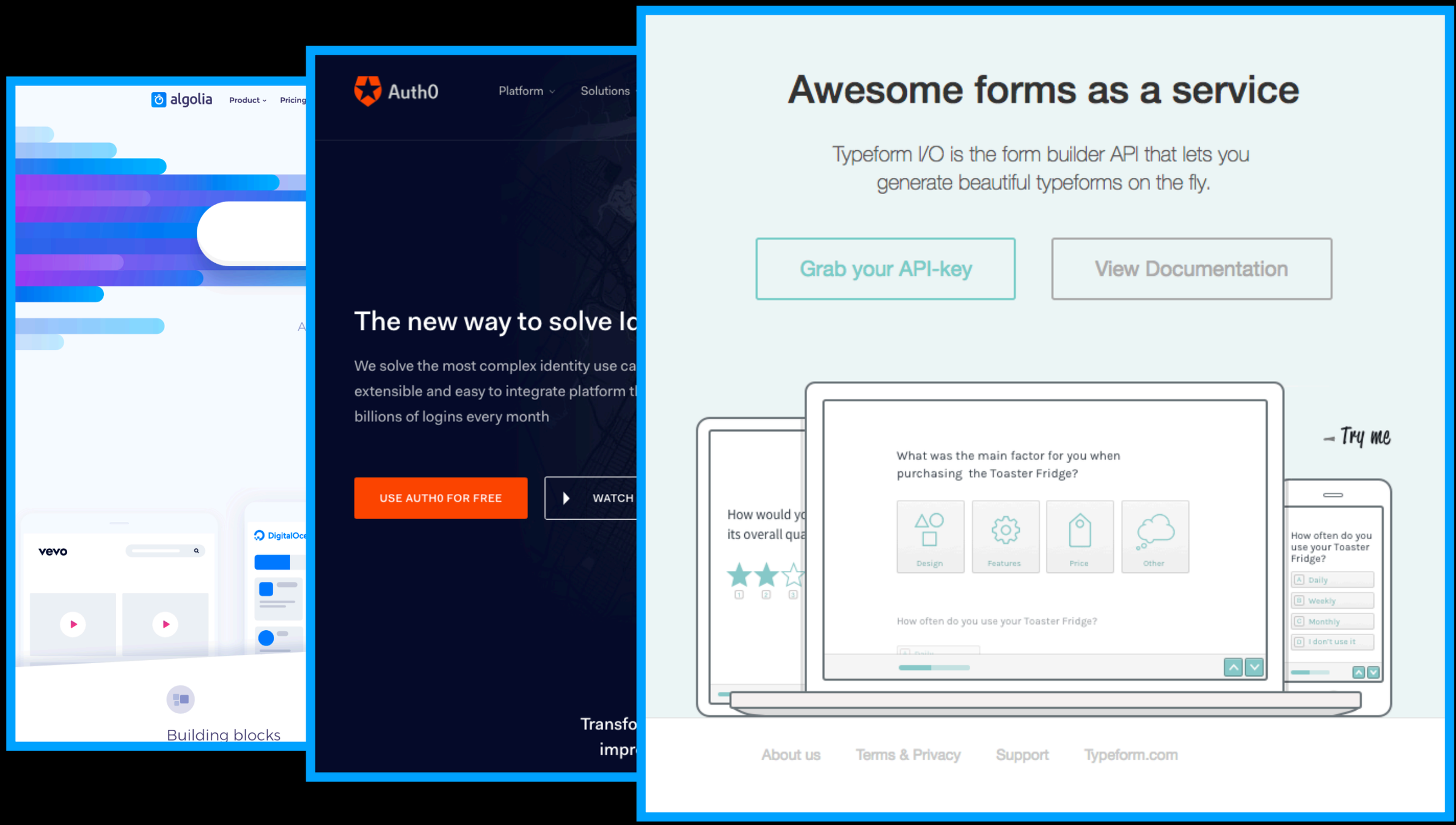
#cloud-services-global #cloud-services-china

- Realtime BaaS
- GraphQL + FaaS + BaaS
- Headless CMS



业务逻辑的服务化：不止通信和支付

#cloud-services-global #cloud-services-china



数据智能的服务化： AIaaS + BDaaS

#cloud-services-global #cloud-services-china

Speech synthesis

Speech recognition

Question answering

Automatic summarization

Topic segmentation

Predictive analytics

Information extraction

Sentiment analysis

Face recognition

Recommendation

Word segmentation

Statistical analysis

NLP

Computer Vision

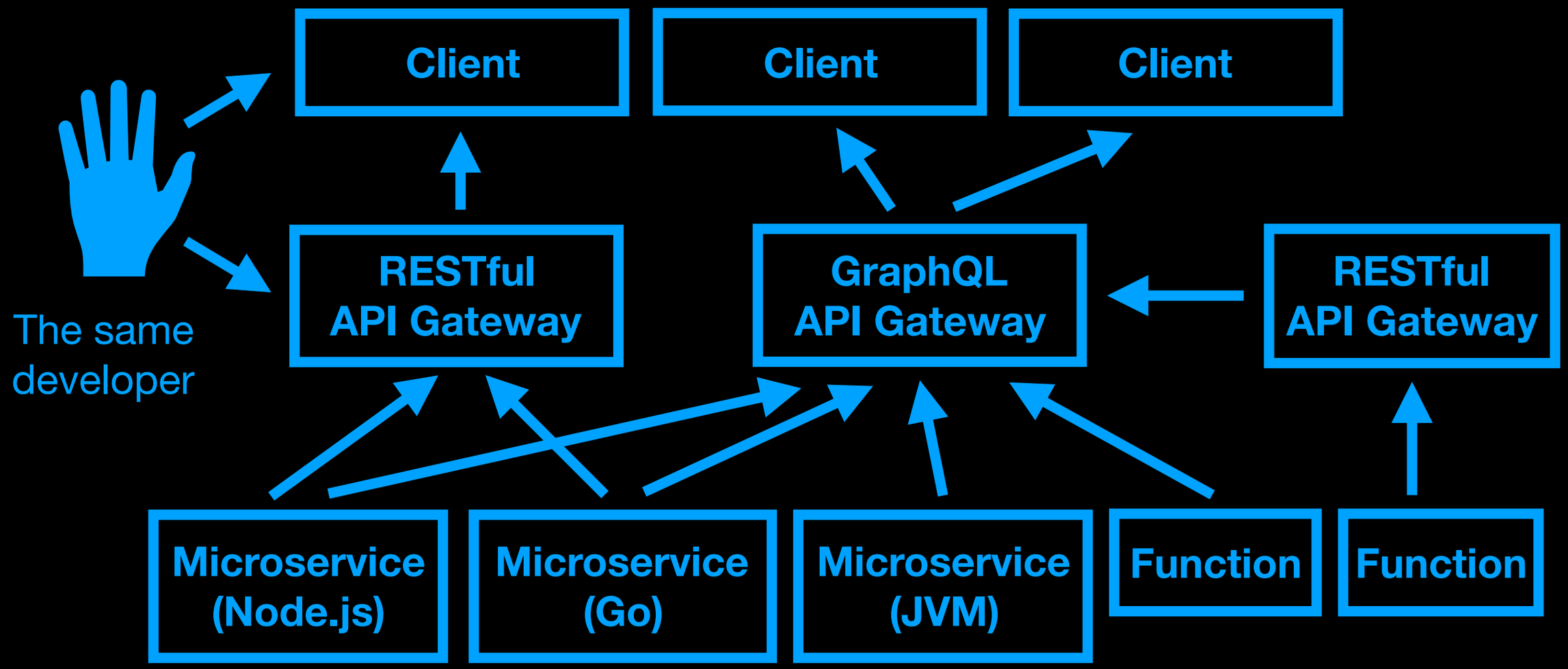
Big data analytics platform

Machine learning platform

Deep learning platform

Microservice / Serverless + API Gateway

- API service：无状态、自包含、易配置、易调度
  The Twelve-Factor App -> Microservices -> Serverless
  #server-side-best-practices #microservices--api-services-nodejs
- API Gateway：允许 API service 与客户端解耦
  #server-side-best-practices



Microservice / Serverless + API Gateway

- Authentication / Authorization
  - Cookies vs Tokens
  - session -> JWT
  - ID Token, Access Token, Refresh Token

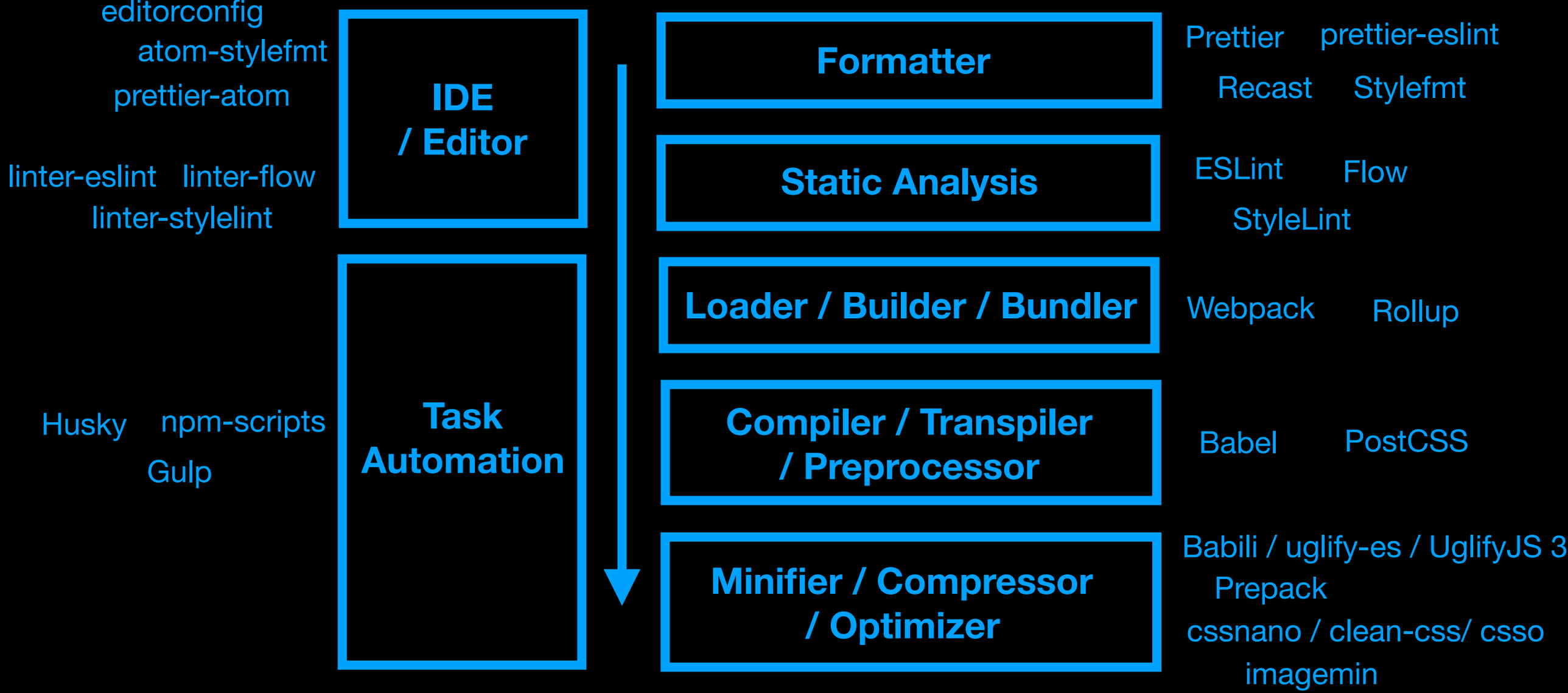
#server-side-best-practices

导读：工具

- 工具链
- 工作流

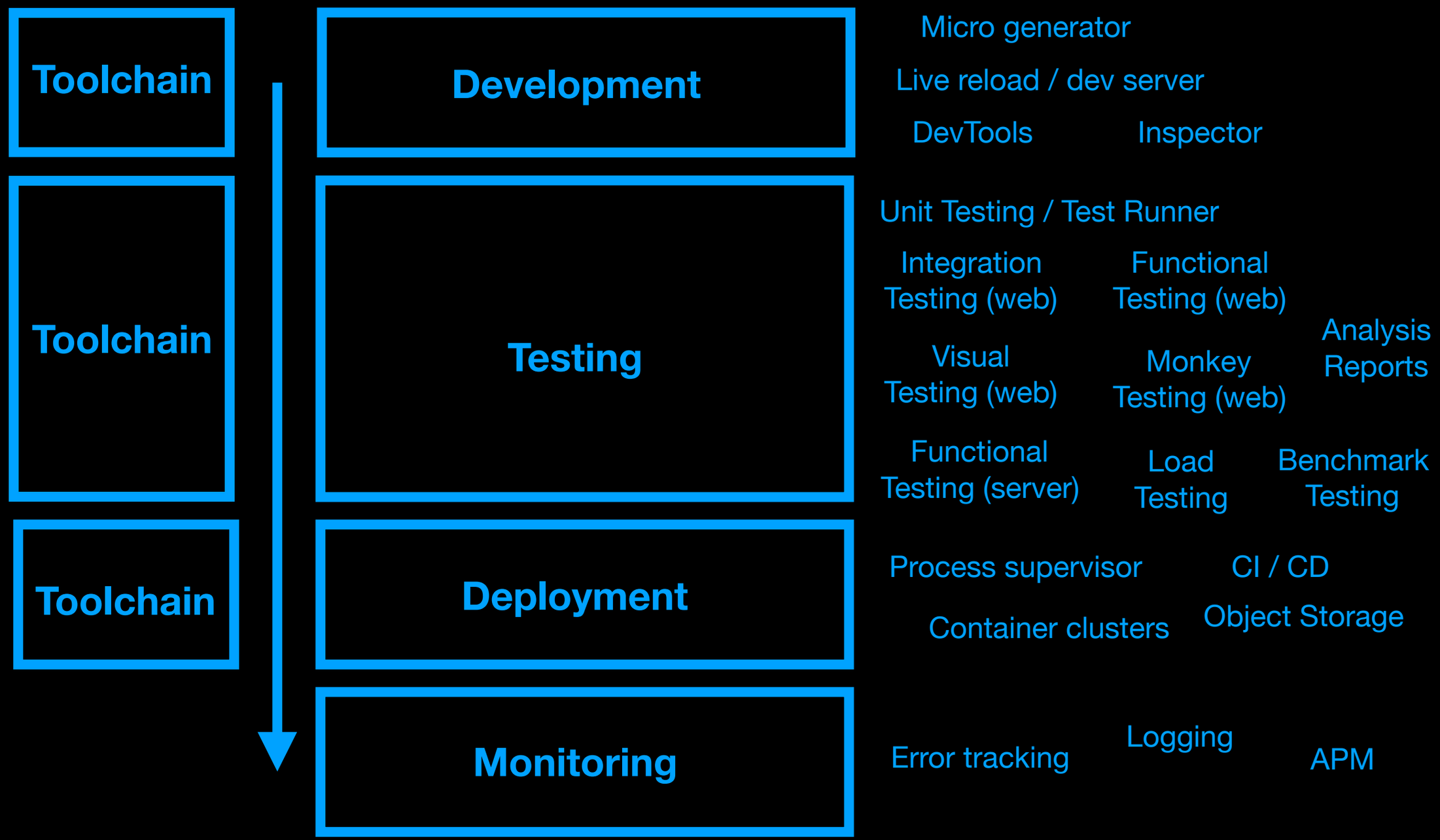
工具链

#toolchain



工作流

#workflow #testing



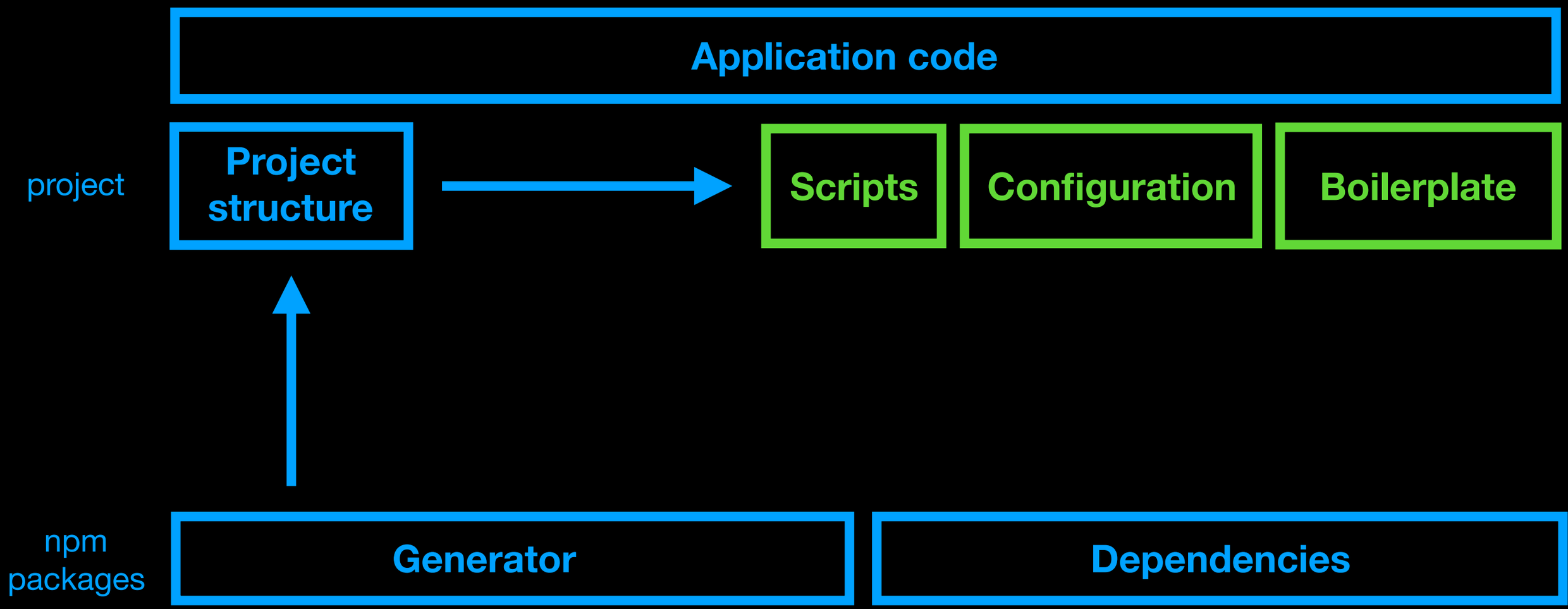
在 Flipboard 的实践

1. 把脚手架变成『库』
2. 自由流动的代码组织
3. 面向小规模，保持小规模

把脚手架变成『库』

- 尽可能集成所有主流工具和最佳实践，让每个项目都能自动获得技术红利，把选择成本降到接近零

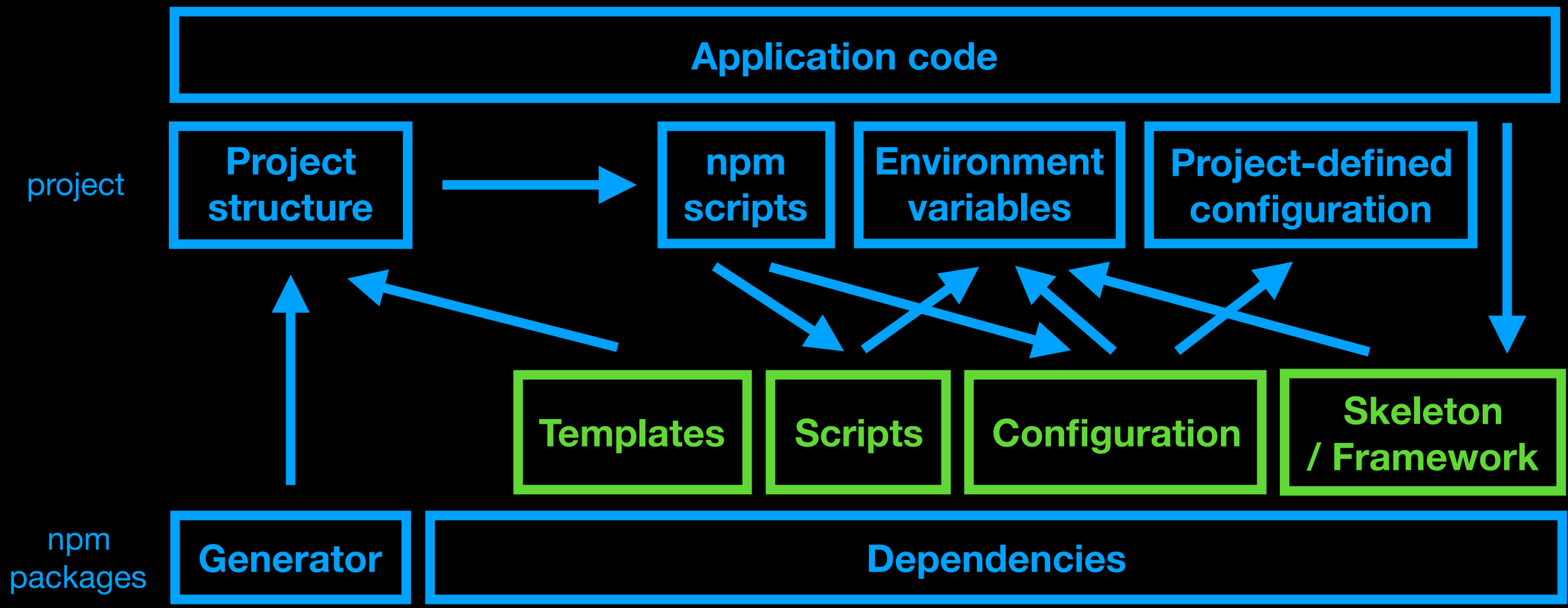
- 尽可能把不属于应用项目本身、在不同项目之间共享的脚手架代码抽象到 npm package 里



把脚手架变成『库』

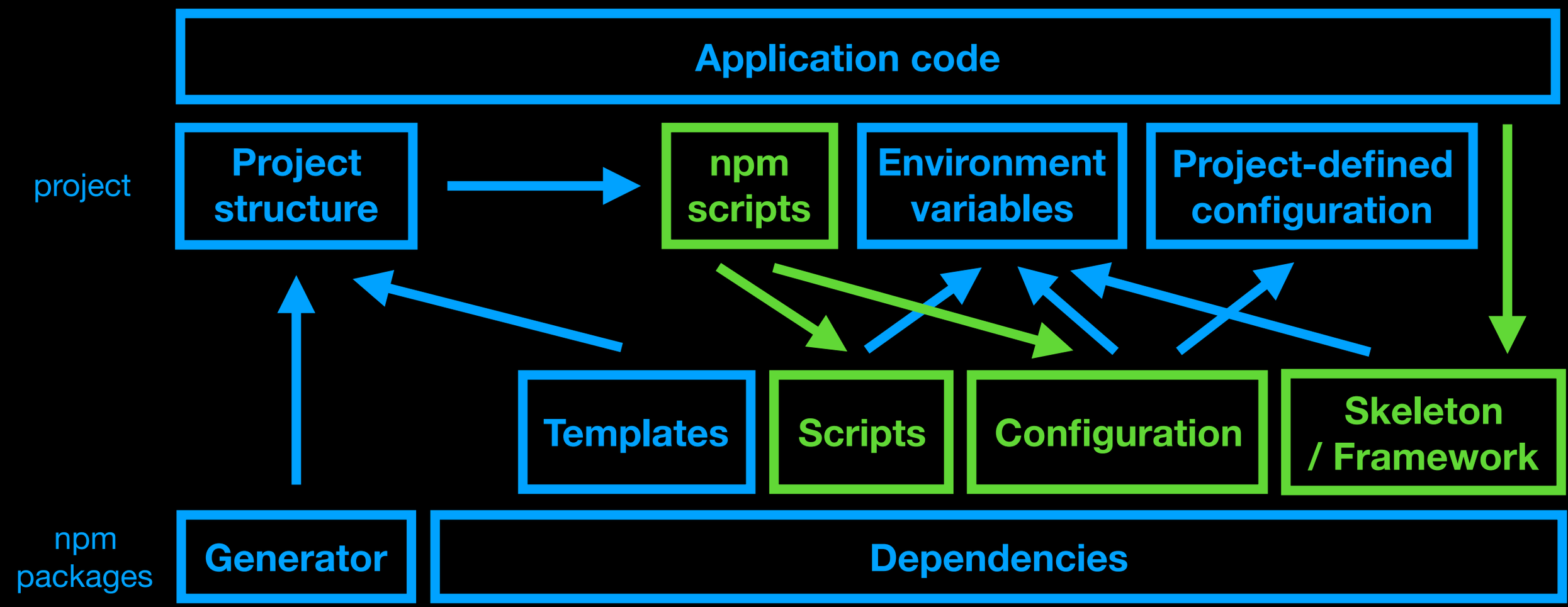
- 尽可能集成所有主流工具和最佳实践，让每个项目都能自动获得技术红利，把选择成本降到接近零

- 尽可能把不属于应用项目本身、在不同项目之间共享的脚手架代码抽象到 npm package 里



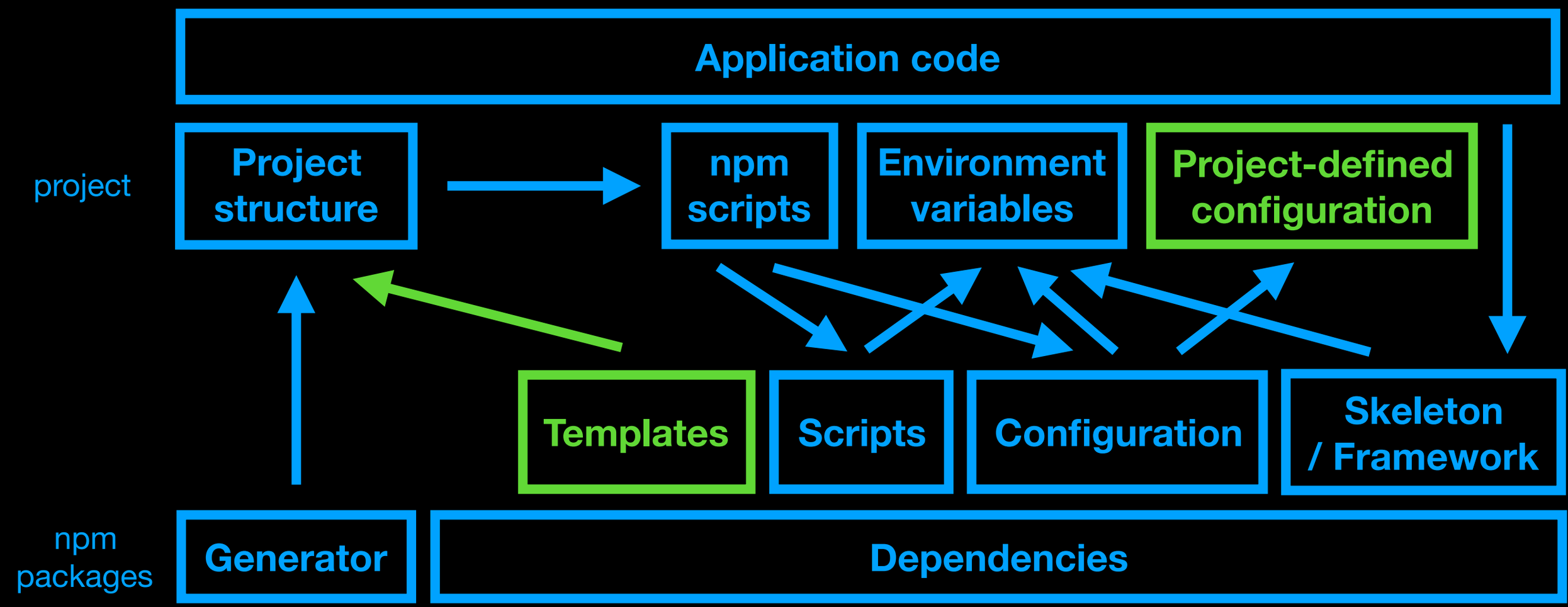
把脚手架变成『库』

Tradeoff：
为简单易用而提高抽象封装程度
vs
保持可组合可升级可替换、持续更新演进的能力



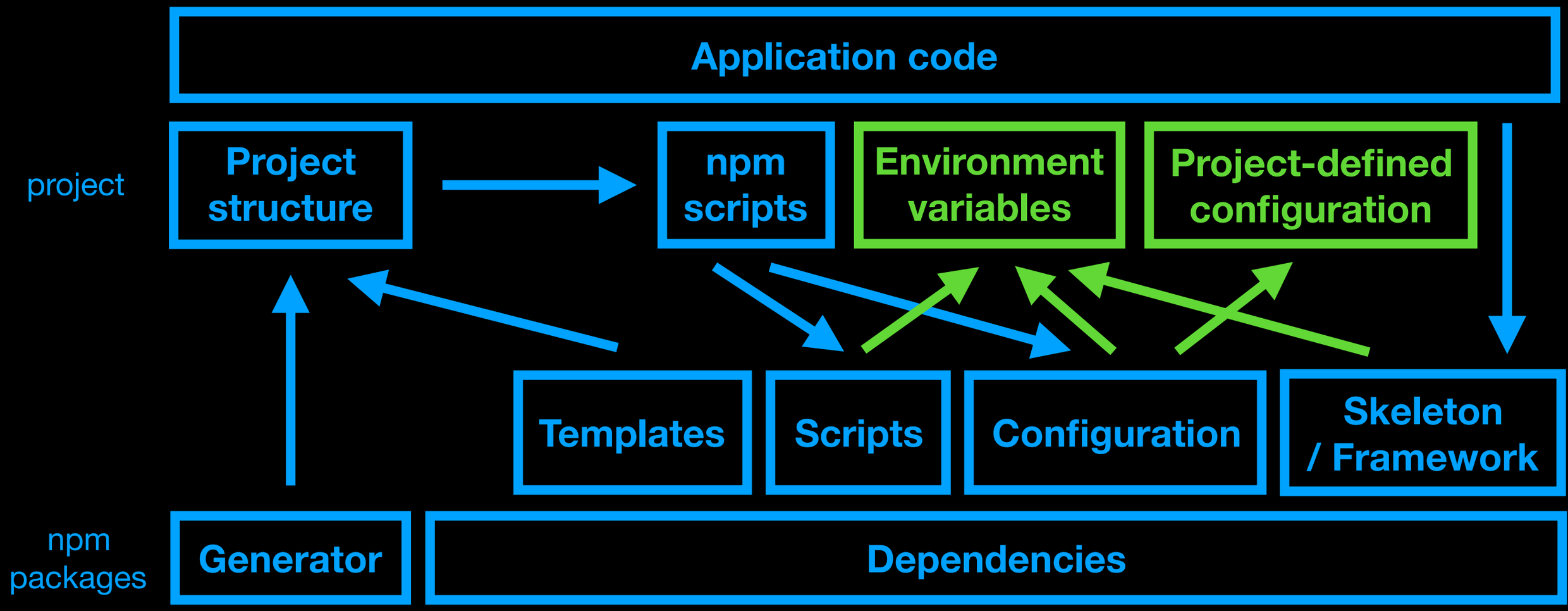
把脚手架变成『库』

弱化 generator 和 xxx-cli，鼓励 micro-generator



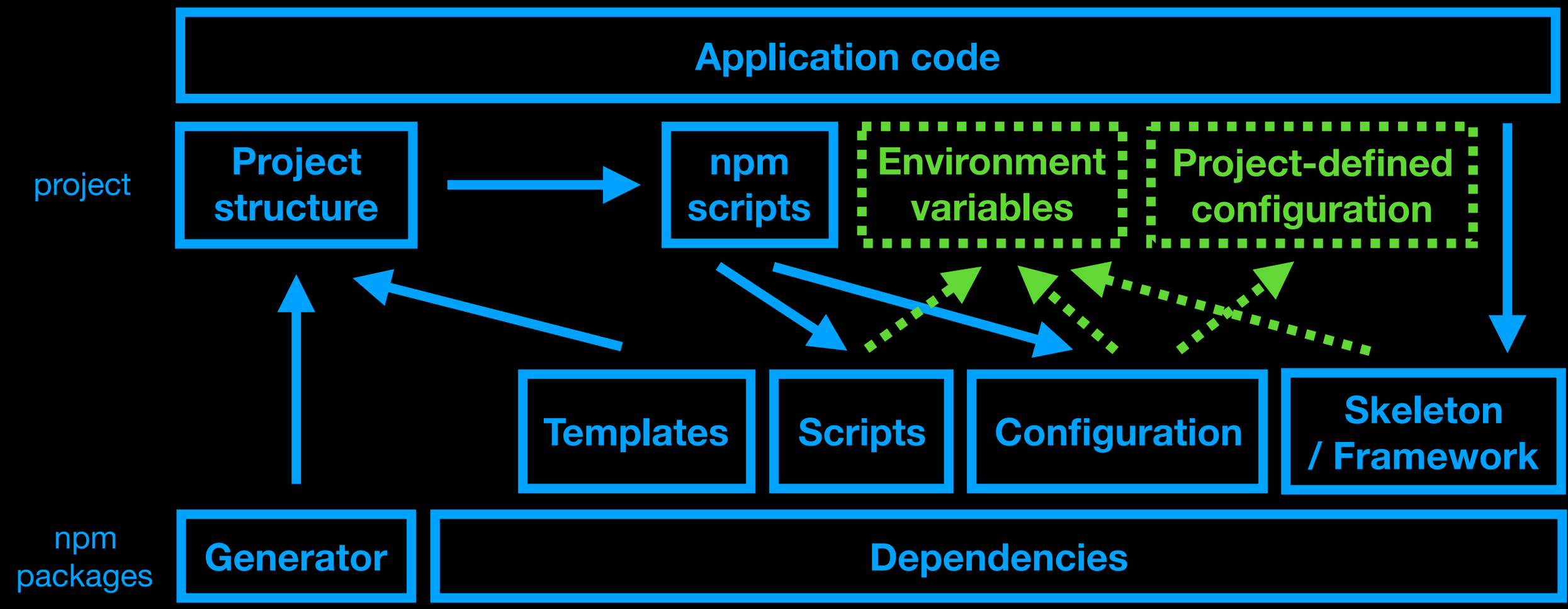
把脚手架变成『库』

定制能力以环境变量（多数）和插件/钩子（少数）的形式提供给应用项目



把脚手架变成『库』

约定大于配置，初始状态（无任何环境变量）即可运行，
默认状态即推荐状态



把脚手架变成『库』：客户端

- https://github.com/dexteryy/webcube
- 服务于 Universal Web（web app / web pages）
- 提供 AppSkeleton 抽象类和可插拔 Root，同时支持 React、React + Router、React + Redux + Router 等多种技术栈组合
- 贯彻以 JS 为中心的原则，SSR 和静态 HTML 容器都位于应用源代码之外
- 同时支持三种部署方式（Static Web 含两种子模式）
- 同时支持多页网站和单页应用（multiple entrypoints）
- 面向需求迭代和抽象演进、代码可『向上』或跨项目『流动』的文件结构

把脚手架变成『库』：服务器端

- https://github.com/dexteryy/nodecube
- 服务于 microservice 和 API gateway
- 整合 Docker 和 Docker Compose 用于本地开发测试和 CI/CD
- 基于 express，整合 API service 必备中间件和最佳实践，轻量封装
- 错误处理、外部服务（数据库、阿里云 SDK 等）以『库』的形式提供

自由流动的代码组织

```
[project]/
├── src/
│   ├── common/
│   │   └── [role]/
│   ├── [entrypoint]/
│   │   ├── common/
│   │   │   └── [role]/
│   │   ├── main/
│   │   │   └── [role]/
│   │   ├── [feature-set]/
│   │   │   ├── [role]/
│   │   │   ├── components/
│   │   │   ├── containers/
│   │   │   ├── actions/
│   │   │   │   └── utils/
│   │   │   ├── reducers/
│   │   │   │   └── utils/
│   │   │   ├── constants/
│   │   │   └── utils/
│   │   ├── [feature-set]/
│   │   └── [feature-set]/
│   ├── [entrypoint]/
│   └── [entrypoint]/
├── configs/
├── staticweb/
├── …
└── package.json
```

- project -> entrypoint -> feature set -> role -> file

自由流动的代码组织

```
[project]/
├── src/
│   ├── common/
│   │   └── [role]/
│   ├── [entrypoint]/
│   │   ├── common/
│   │   │   └── [role]/
│   │   ├── main/
│   │   │   └── [role]/
│   │   ├── [feature-set]/
│   │   │   ├── [role]/
│   │   │   ├── components/
│   │   │   ├── containers/
│   │   │   ├── actions/
│   │   │   │   └── utils/
│   │   │   ├── reducers/
│   │   │   │   └── utils/
│   │   │   ├── constants/
│   │   │   └── utils/
│   │   ├── [feature-set]/
│   │   └── [feature-set]/
│   ├── [entrypoint]/
│   └── [entrypoint]/
├── configs/
├── staticweb/
├── …
└── package.json
```

- project -> entrypoint -> feature set -> role -> file
- 在 entrypoint 之间复用代码、在 feature set 之间复用代码、在 role 之间共享代码、在 role 内部共享代码

自由流动的代码组织

```
[project]/
├── src/
│   ├── common/
│   │   └── [role]/
│   ├── [entrypoint]/
│   │   ├── common/
│   │   │   └── [role]/
│   │   ├── main/
│   │   │   └── [role]/
│   │   ├── [feature-set]/
│   │   │   ├── [role]/
│   │   │   ├── components/
│   │   │   ├── containers/
│   │   │   ├── actions/
│   │   │   │   └── utils/
│   │   │   ├── reducers/
│   │   │   │   └── utils/
│   │   │   ├── constants/
│   │   │   └── utils/
│   │   ├── [feature-set]/
│   │   └── [feature-set]/
│   ├── [entrypoint]/
│   └── [entrypoint]/
├── configs/
├── staticweb/
├── …
└── package.json
```

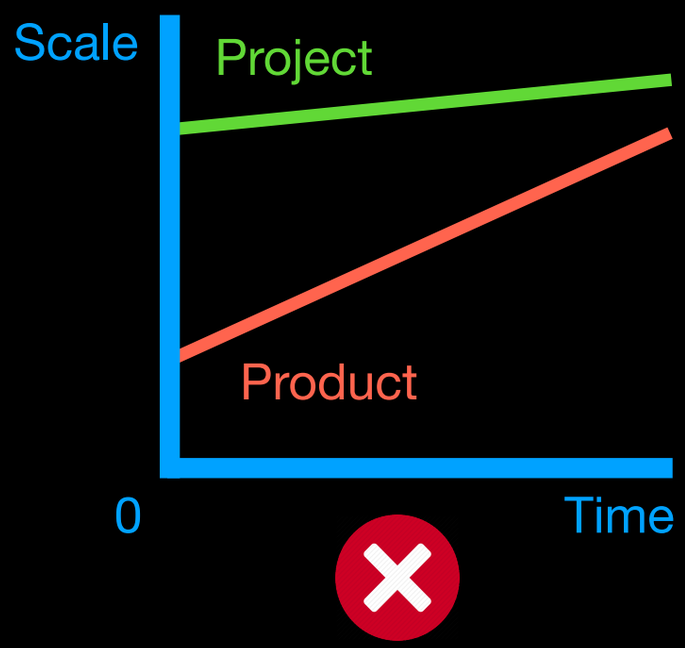
- project -> entrypoint -> feature set -> role -> file
- 在 entrypoint 之间复用代码、在 feature set 之间复用代码、在 role 之间共享代码、在 role 内部共享代码
- 向上流动、项目间流动、独立为新项目

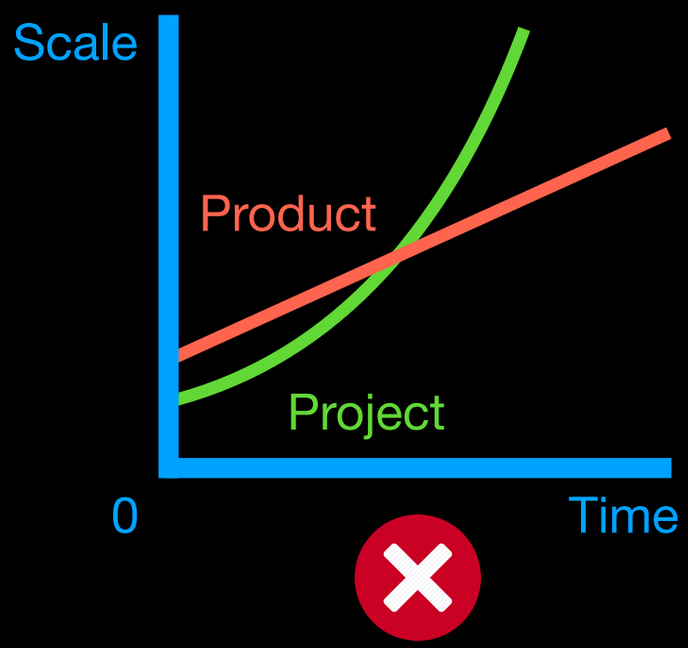
other entrypoint / other project

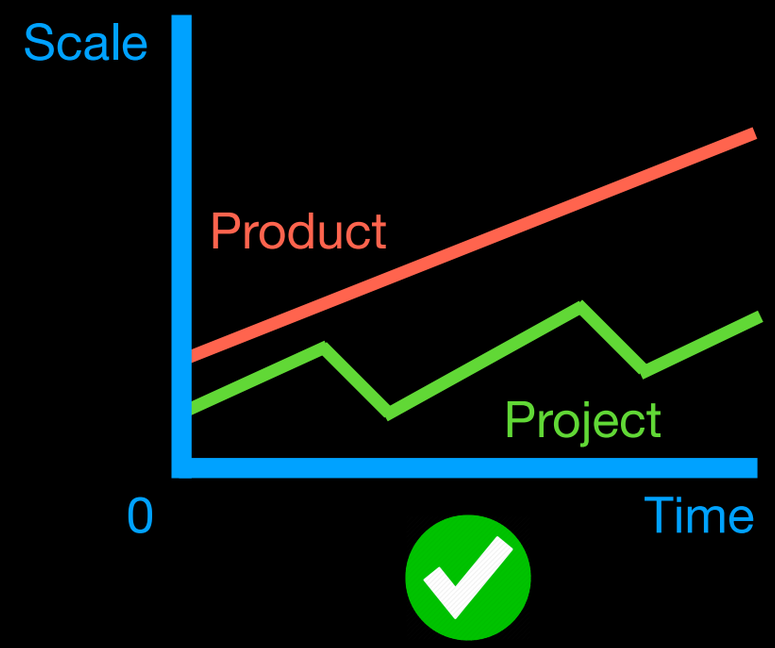
other project / new project

面向小规模，保持小规模

- 所有项目在产品成功之前，必须是小规模项目
- 所有项目在产品成功之后，有必要保持小规模或保持小规模化的能力
- 项目规模膨胀是一种技术债和代码腐坏
- 消除项目保持小规模的障碍：可调用可组装、约定大于配置、持续更新的脚手架、自由流动的代码组织、……







Thanks

dexter.yy@gmail.com

dexteryy @ Github

dexteryy @ Twitter

dexteryy @ 微博

幻灯片： https://speakerdeck.com/dexteryy/understanding-modern-web-development-at-jsconf-china-2017-zhong-wen

https://github.com/dexteryy/spellbook-of-modern-webdev

https://github.com/dexteryy/webcube

https://github.com/dexteryy/nodecube

Flipboard 中国招聘 web 应用开发工程师 / web 前端开发工程师