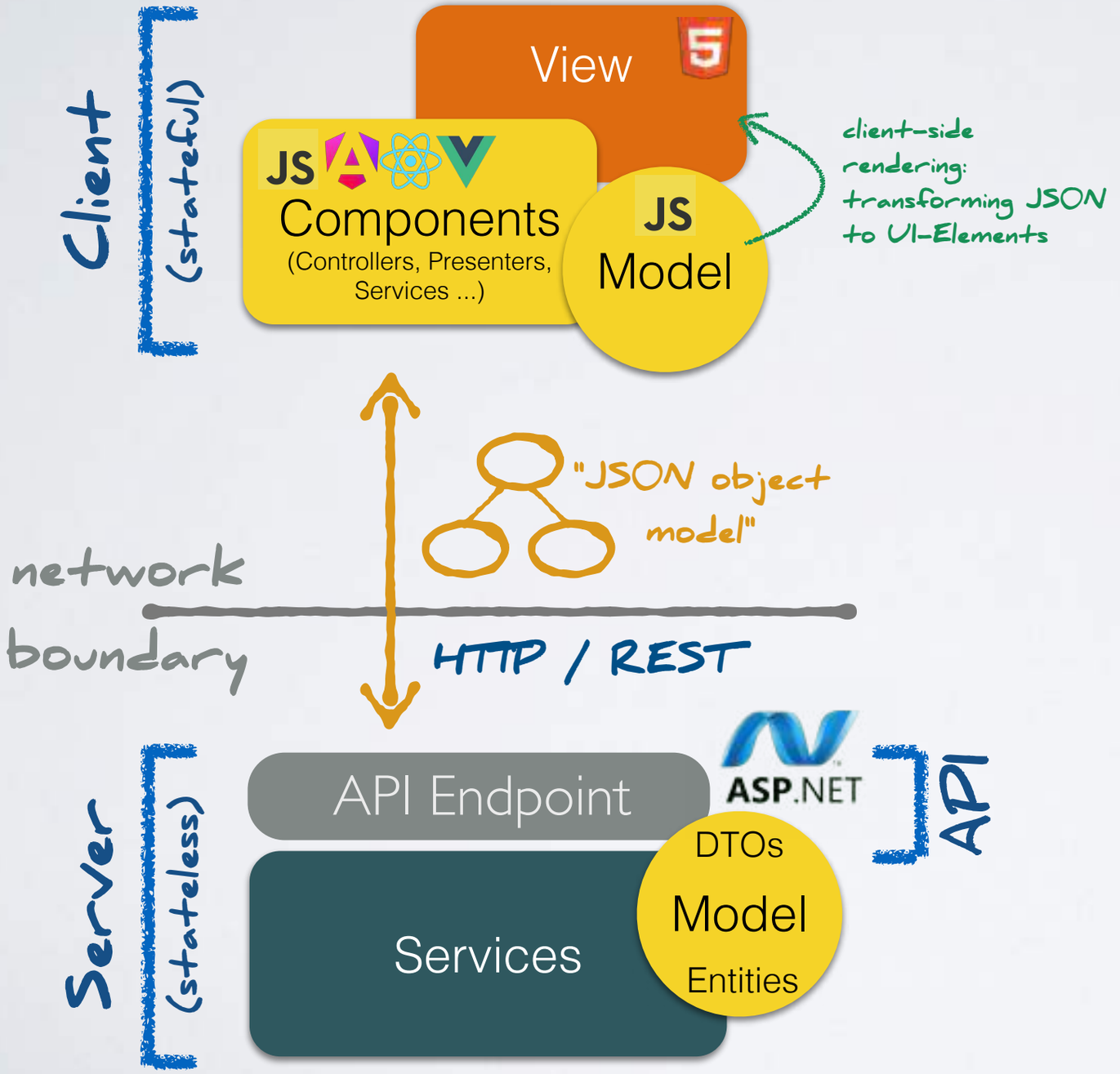

NOT SURE IF APP IS CHANGING PAGES
WITHOUT ME NOTICING
OR APP IS A SPA
memegenerator.net

SPA Basics

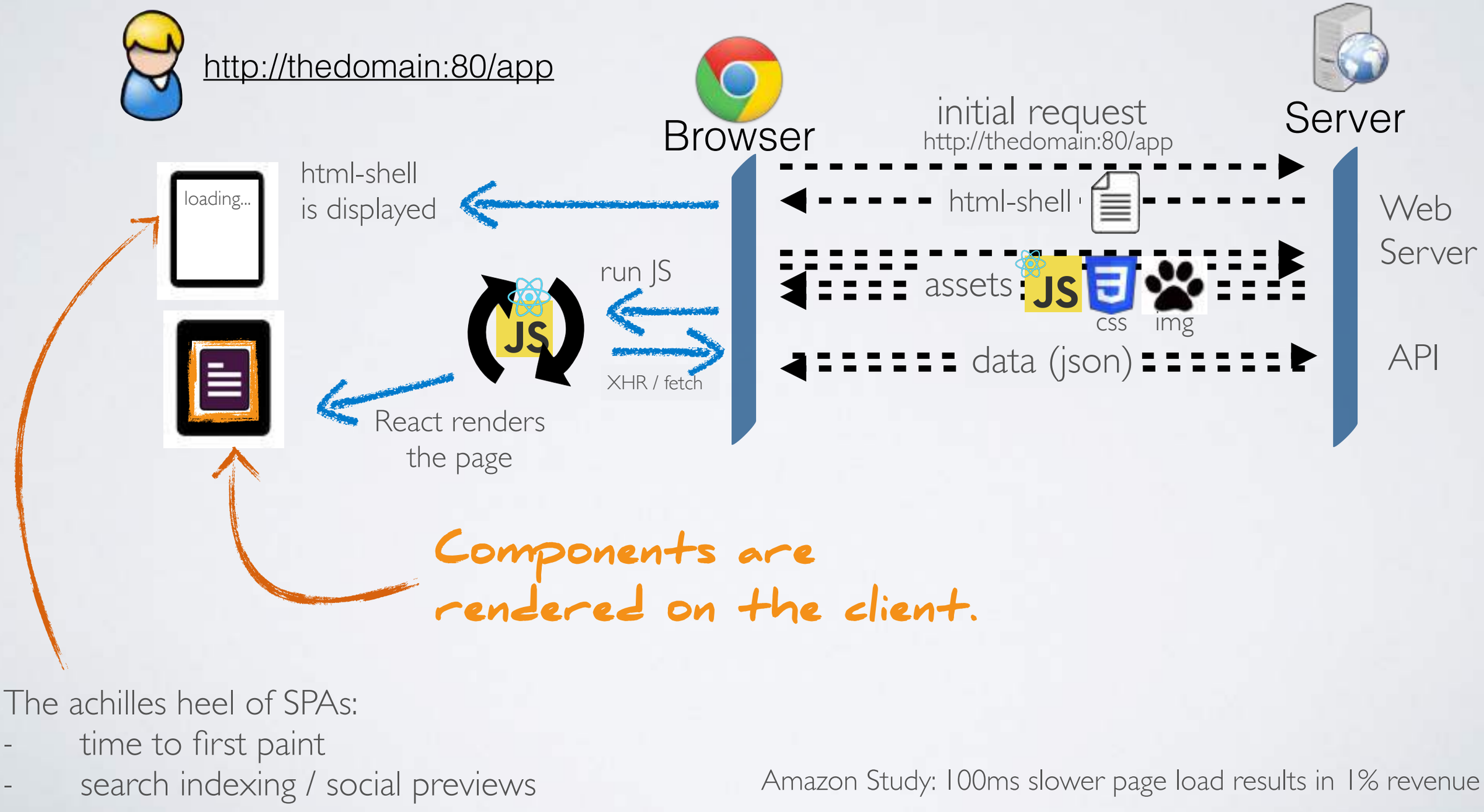
SPA Architecture



Rich client programming model in the browser.

Clear separation of concerns between client and server.

Traditional SPA:
Client Side Rendering



The achilles heel of SPAs:

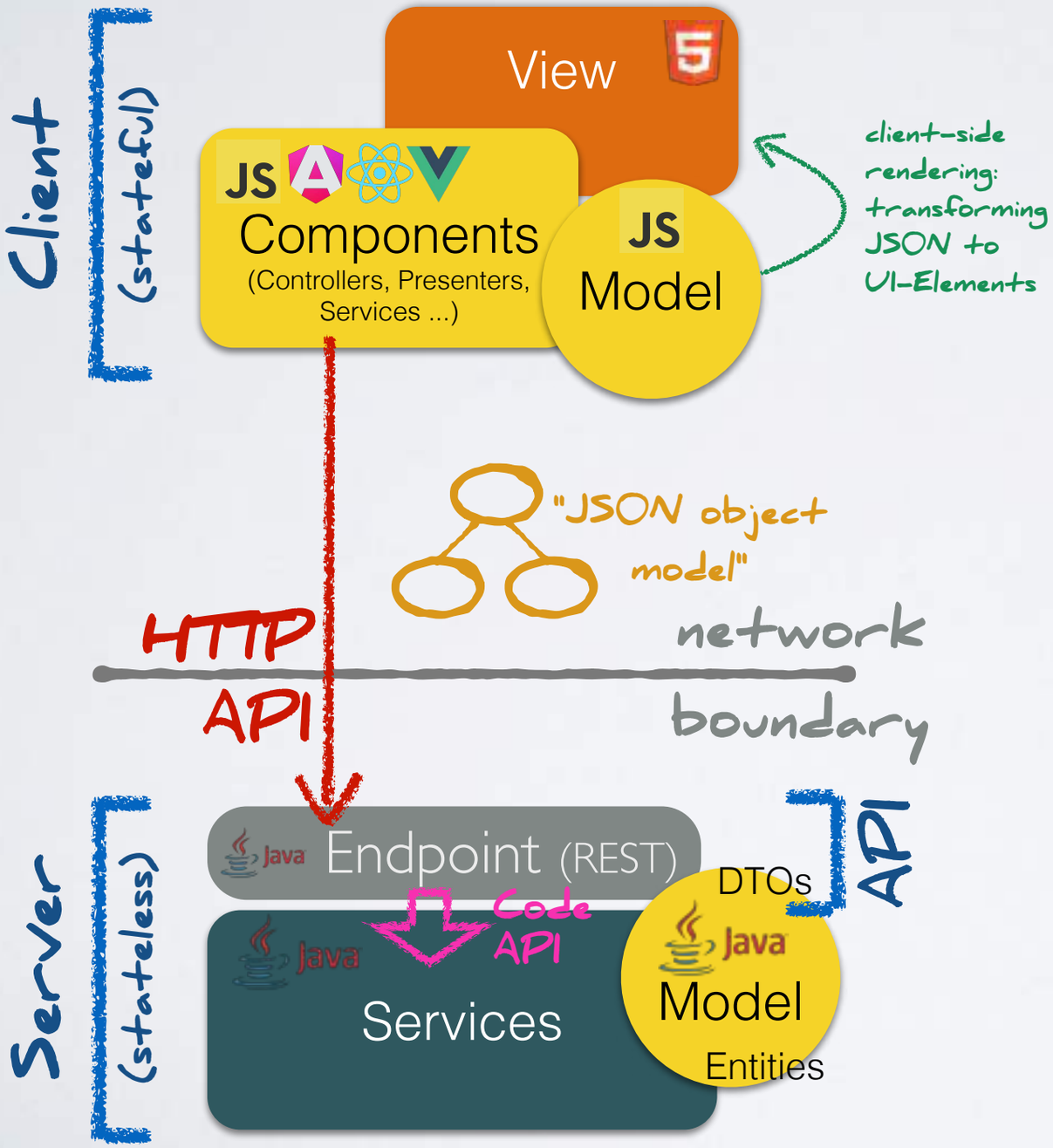
- time to first paint
- search indexing / social previews

Amazon Study: 100ms slower page load results in 1% revenue loss!

SPA Data-Fetching

The architecture for Single Page Applications implied a HTTP API.

The Role of the API



The rise of SPA development caused a "de-facto" architecture of formalized HTTP/REST-APIs.

Symptoms:

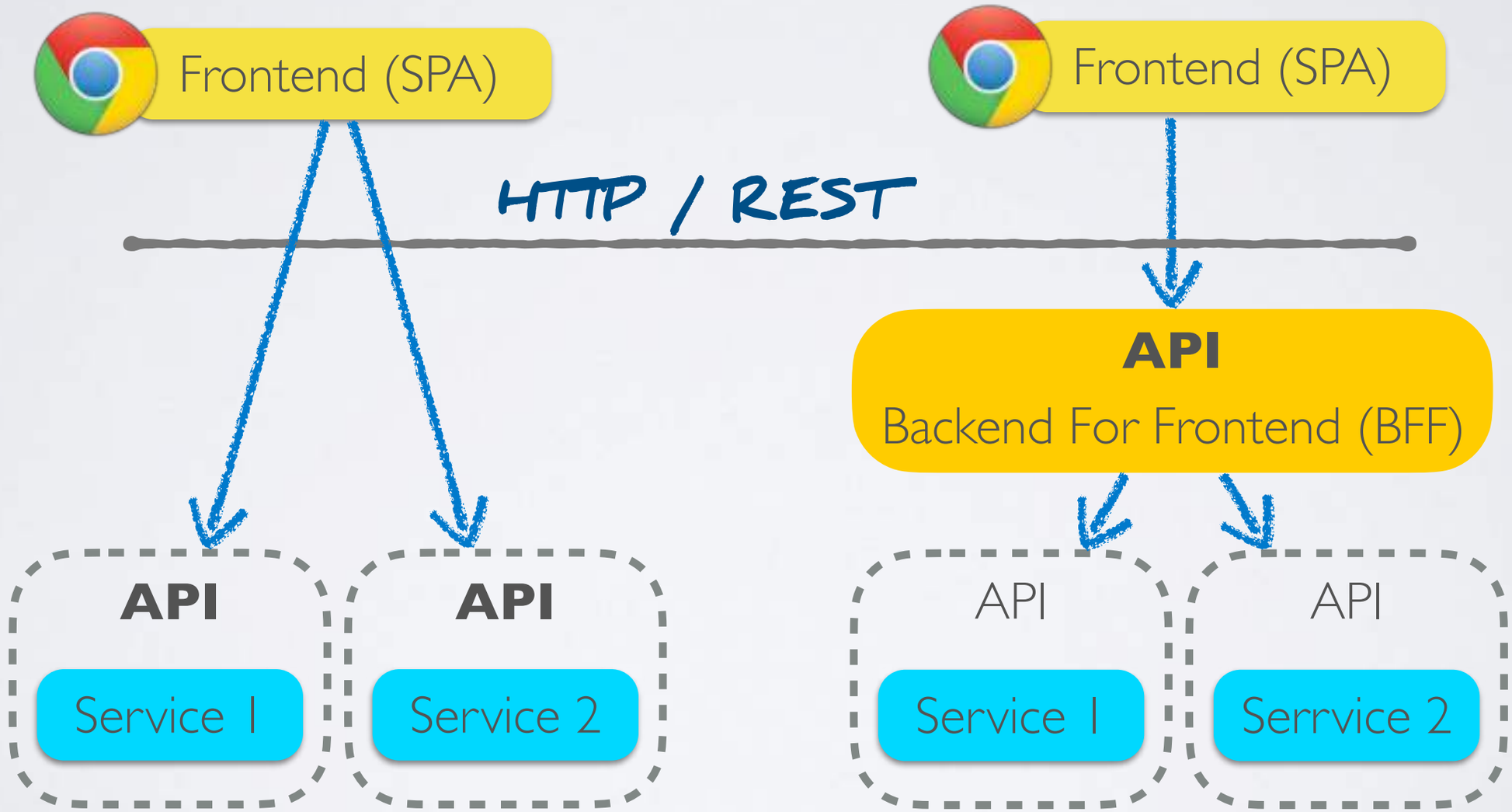
- "API-First" Design
- "The central role of API-Gateways" (the return of ESBs)
- ...

Creating a formalized API is a non-trivial effort:
Proper design of URLs, Mapping, Serialization, Security ...

There are advantages in a formalized HTTP API: separation of concern, clearly specified and testable boundaries, reuse, team separation ...

Traditional Architectures for SPAs

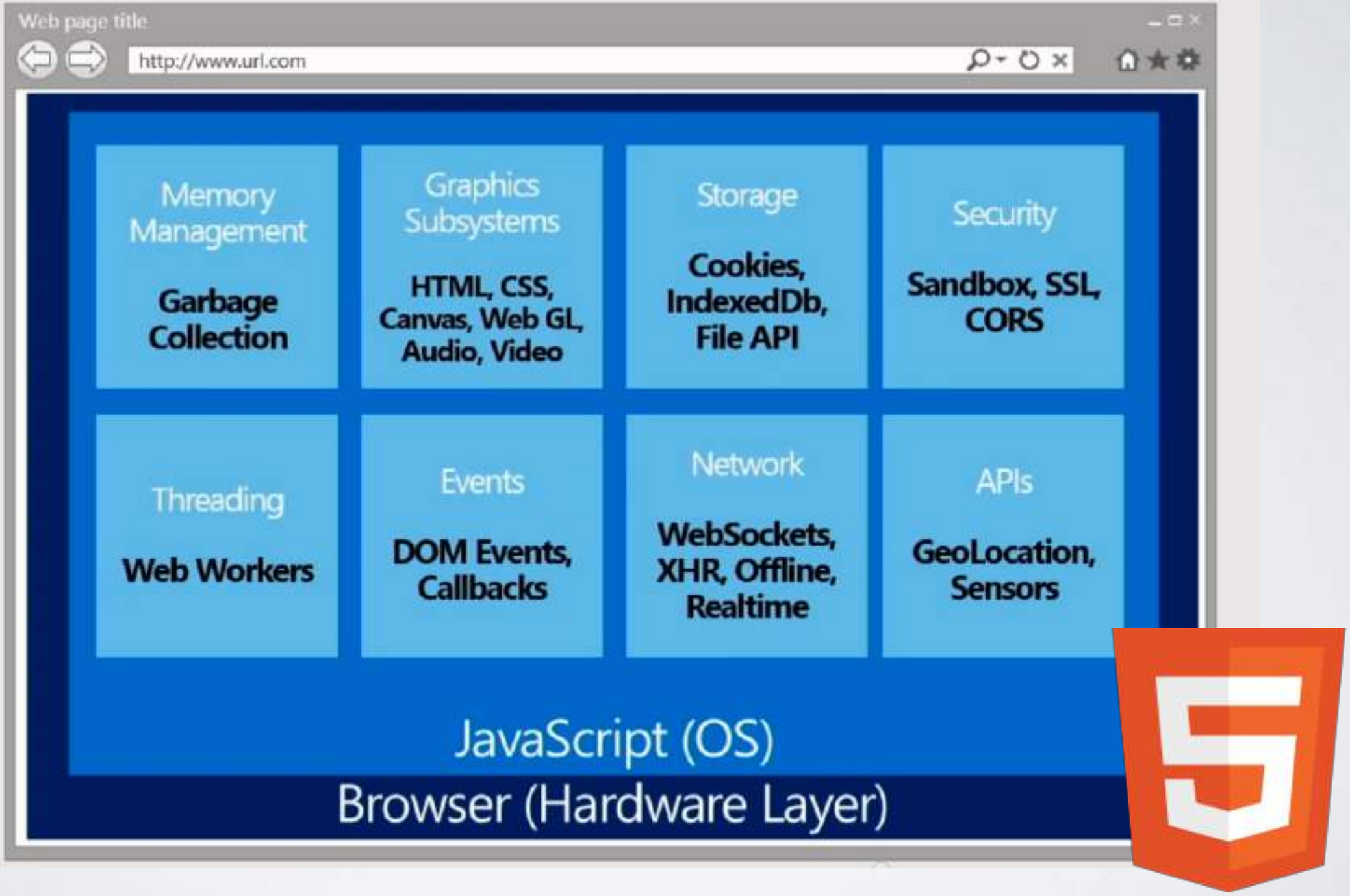
Client - API - Server



https://samnewman.io/patterns/architectural/bff/

Why should we create a SPA?

The browser: An application platform?



Scott Hanselman Fluent 2014 keynote "Virtual Machines, JavaScript and Assembler"
https://www.youtube.com/watch?v=UzyoT4DziQ4

Single Page Applications

+

- User Experience
- Architecture: Clear Separations:
  - Client vs Backend
  - API usable by different Clients
- Design & Implementation:
  - embrace the web concepts and technologies
  - stateful client / stateless server
  - "rich client" programming in the browser

−

- different development environments for client & server programming
- The browser platform is not the most productive platform
- distributed state (synchronization, validation ...)

Components

Component Architecture

Current Frameworks (Angular, React, Vue.js) moved away from the "traditional" MVC pattern to a "Component Architecture".

Components:

- divide & structure
- encapsulation (view & logic)
- reuse
- defined boundaries
- data flow

Typically the whole application itself is a component ("root component"). The html document is just a "shell" in which the application is loaded.

Components

Components in a Web-UI typically consist of two parts:

- a "template" which represents the DOM structure (potentially including styling)
- JavaScript code that implements the behavior

Modern SPA Frameworks like Angular or React deliver components in a JavaScript bundle. The template is compiled to DOM manipulation statements.

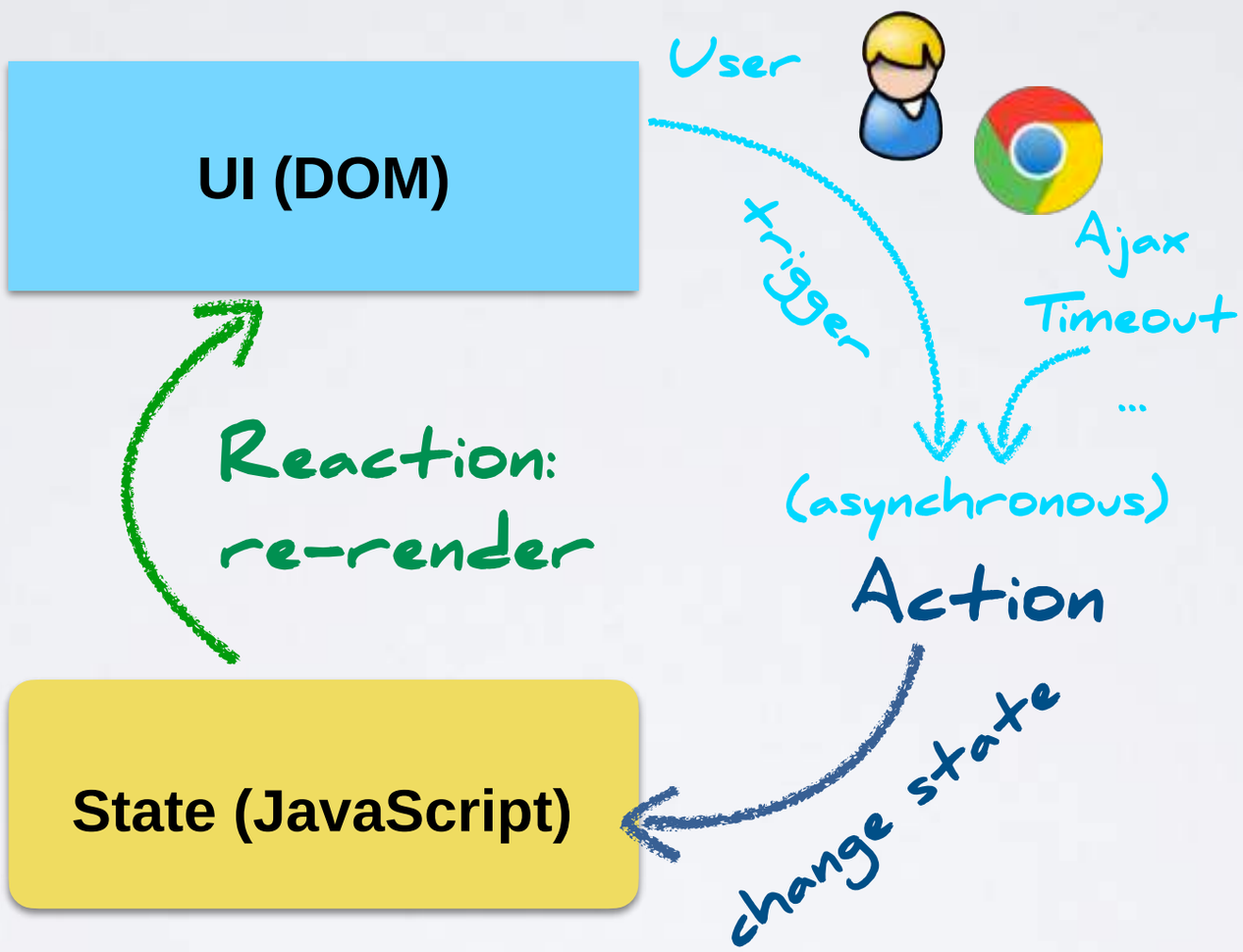
Examples:

- jQuery template & Handlebars load HTML snippets as text and put them in the DOM ("element.innerHTML")
- In React JSX is compiled to JavaScript
- In Angular and Vue templates are compiled to JavaScript


"Reactivity"

State is Managed in JavaScript

The UI renders the state and "signals" events.



Reactivity in a SPA: The application reacts on state changes and updates the UI.

Evan You - Reactivity in Frontend JavaScript Frameworks: https://www.youtube.com/watch?v=r4pNEdIt_I4
http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

Reactivity ... What and Why?

Traditional "DOM-centric" applications

UI = state

Browsers have "built-in reactivity": If the DOM is changed, the UI is re-rendered.

With modern SPA architectures (MVC, MVP, Components ...) the client state is represented as JavaScript objects.

UI = f(state)

When to call?

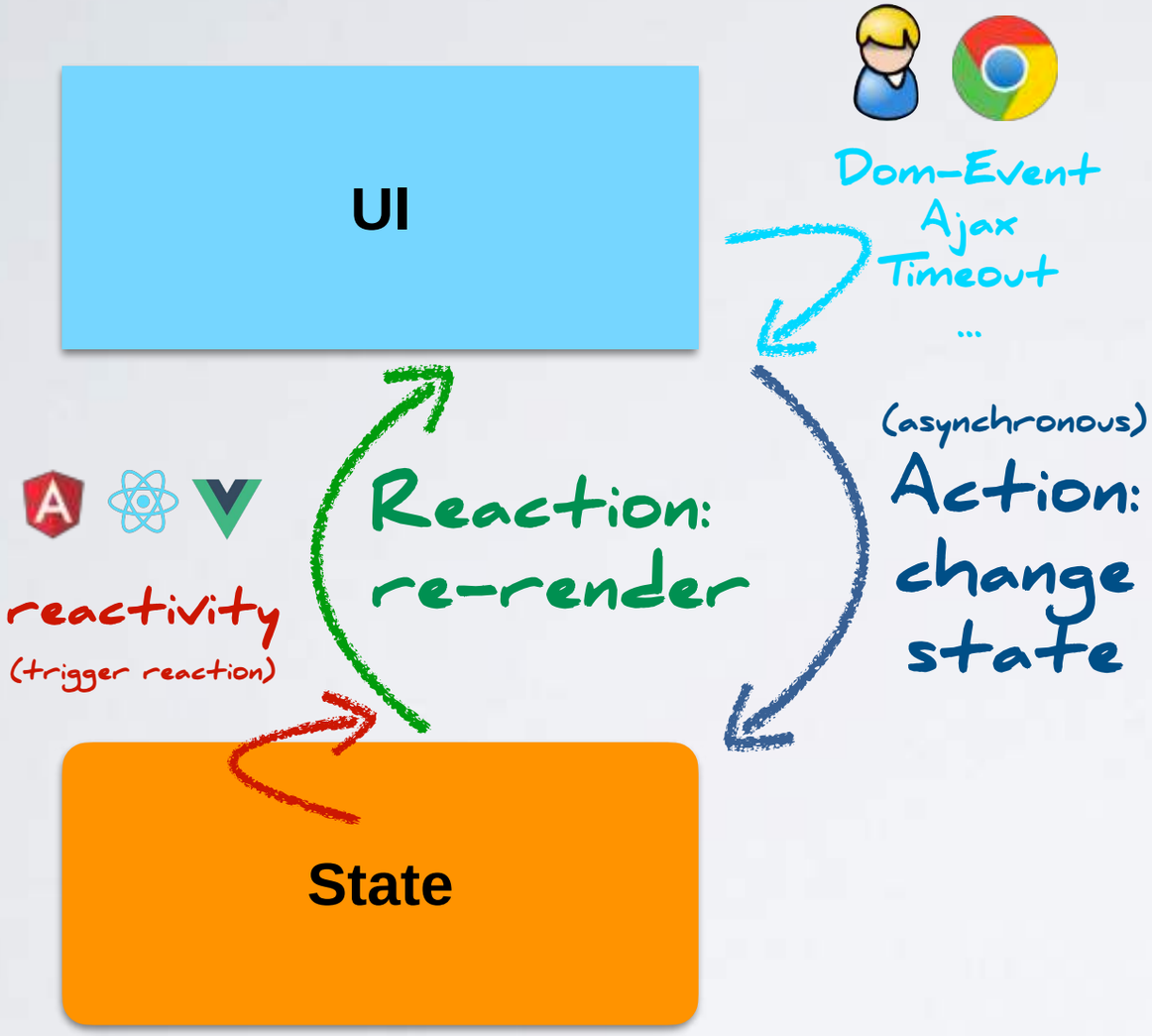
The UI that you can see and manipulate on screen is the result of painting a visual representation of data.

Reactive programming in general addresses the question: How to deal with change over time?

The UI should (automatically) re-render when the state changes.

https://blog.danlew.net/2017/07/27/an-introduction-to-functional-reactive-programming/

Frameworks Reactivity



Front end development and change detection (Angular vs. React):
https://www.youtube.com/watch?v=1i8klHov3vA
https://github.com/in-depth-education/change-detection-in-web-frameworks

https://blog.thoughtram.io/angular/2016/02/22/angular-2-change-detection-explained.html
https://vuejs.org/v2/guide/reactivity.html

Angular: different approachs to Reactivity

Angular comes with *Zone.js*. This is a library that patches the native Browser APIs in order to notify the app when a potential change happens. It triggers the Angular change-detection algorithm which implements dirty checking.
In Angular changes are often explicitly modeled as streams of events/data with RxJS Observables.
With signals Angular offers another API to explicitly model reactive state.
Future versions of Angular will be zoneless and offer fine grained reactivity based on reactive state.

React: Reactivity is based on `useState` / `setState`

`useState` triggers a complete rendering of the component and its child components into the virtual DOM (this can be optimized).

Vue: Reactive state base on `ref()` and `reactive()` (basically signals)

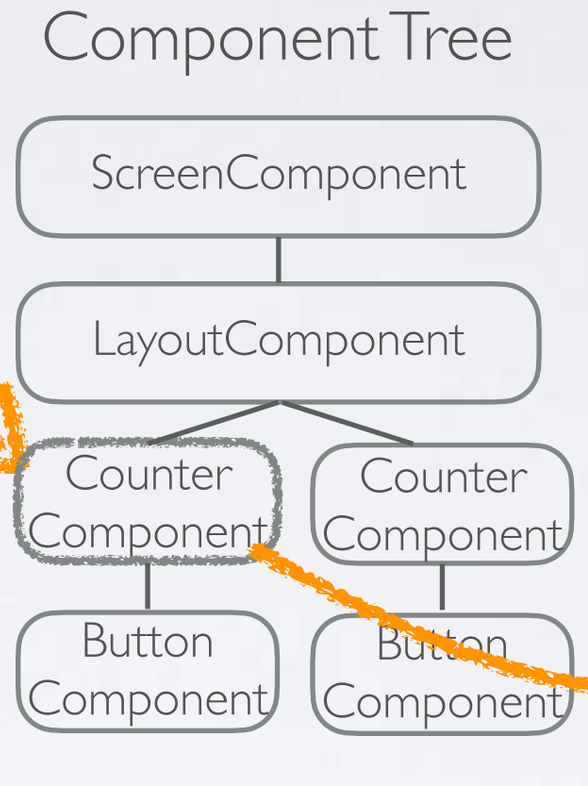
Vue transparently converts properties to *getter/setters*. This allows dependency-tracking and change-notification.

Fine Grained Reactivity

"optimized re-rendering on state changes"

```
JS etInterval(
    () => this.count.set(
      this.count() + 1;
    )
1000);
```

trigger re-rendering



DOM

```
...
<div>
 Count 1:
 <span>42</span>
 <button>Increase</button>
<div>
...
```

Fine grained reactive state only triggers re-rendering on components (or blocks) that depend on changed state.

Many frameworks already implement (more or less) fine grained reactivity: Solid, Qwik, Svelte, Vue ...

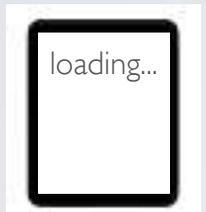
Client-Side Routing in a SPA

http://thedomain:80/app/first

Browser

Server

initial request
http://thedomain:80/app



html-shell is loaded

"navigate" to "first" screen

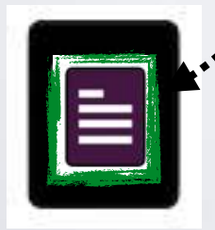
load data &
render screen with JS

http://thedomain:80/app/**first**

navigation

"navigate" to "second" screen

load data &
render screen with JS



http://thedomain:80/app/**second**

html-shell

assets

optional ajax request
http://thedomain:80/api/data/first

data

optional ajax request
http://thedomain:80/api/data/second

data

app bootstrapping

Client Side Routing in a SPA

The traditional web is built on the concept of linked documents (URL, bookmarking, back-button …)

In a SPA the document is just the shell for an application. When you navigate away from the document, the application is "stopped".

As a consequence a SPA should "emulate" the traditional user-experience on the web:

- navigate via urls, links & back-button
- bookmarks and deep links

Client-Side Routing

Single Page Applications can contain several "pages"/views.

Routing maps URLs to views. aka: "Deep-Linking"

Parts of the URL are evaluated on the client.

Traditional approach: "Hash-Routing" (#)

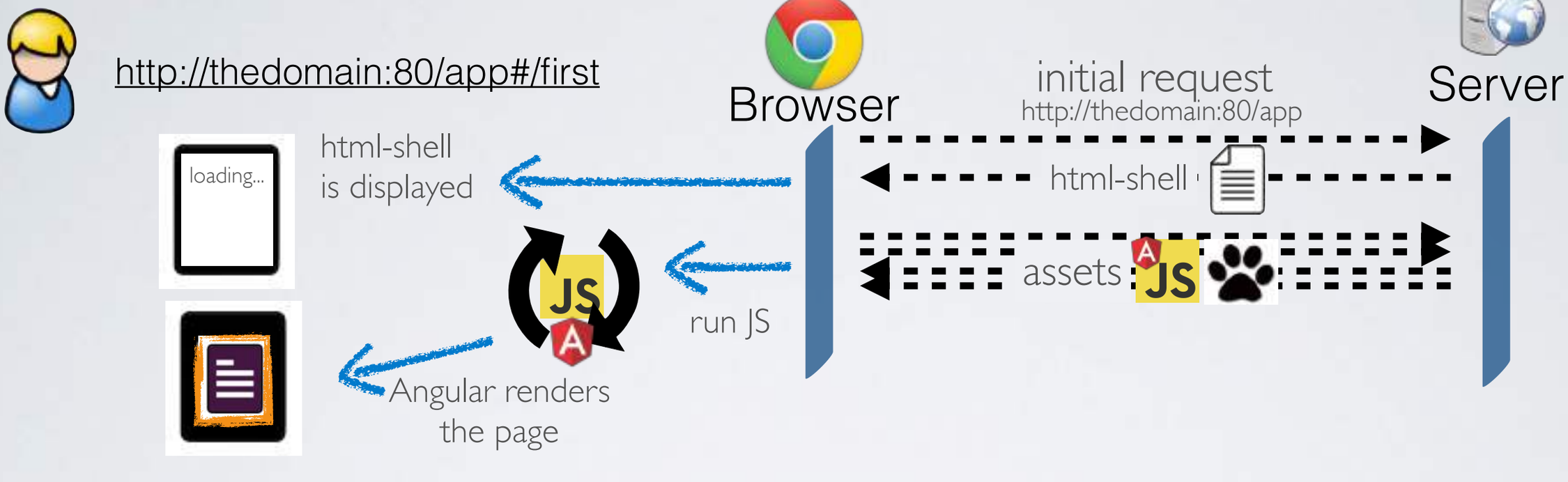
- client-route (part after the **#**) is not sent to the server with the initial http-request
- navigation: browser does not send a request to the server since only the part after the # is changing.

Modern approach: "HTML5 Routing" / "Push-State Routing"

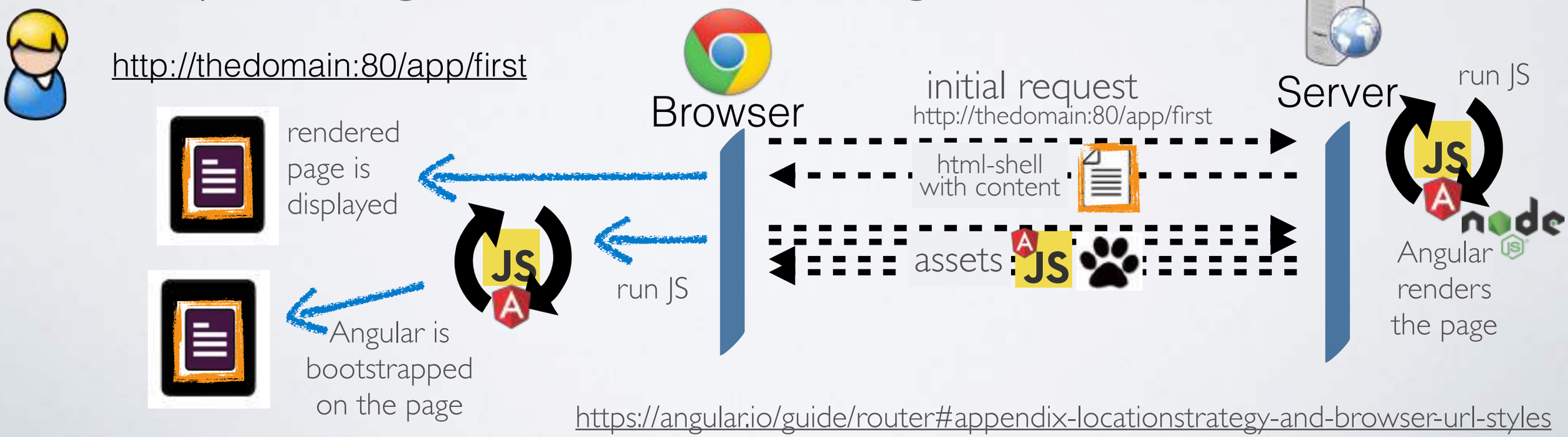
- client-route is sent to the server with the initial http-request
- navigation: url is changed on the client, but the browser does not send a request to the server
- Pitfalls:
  - Server must respond with the "default" shell for unknnown urls (url-rewrite, fallback-url, https://angular.io/guide/deployment#server-configuration)
  - Only supportet in browsers > IE 9 (https://caniuse.com/#feat=history)

Hash-Routing vs. HTML5 Routing

Traditional SPA:

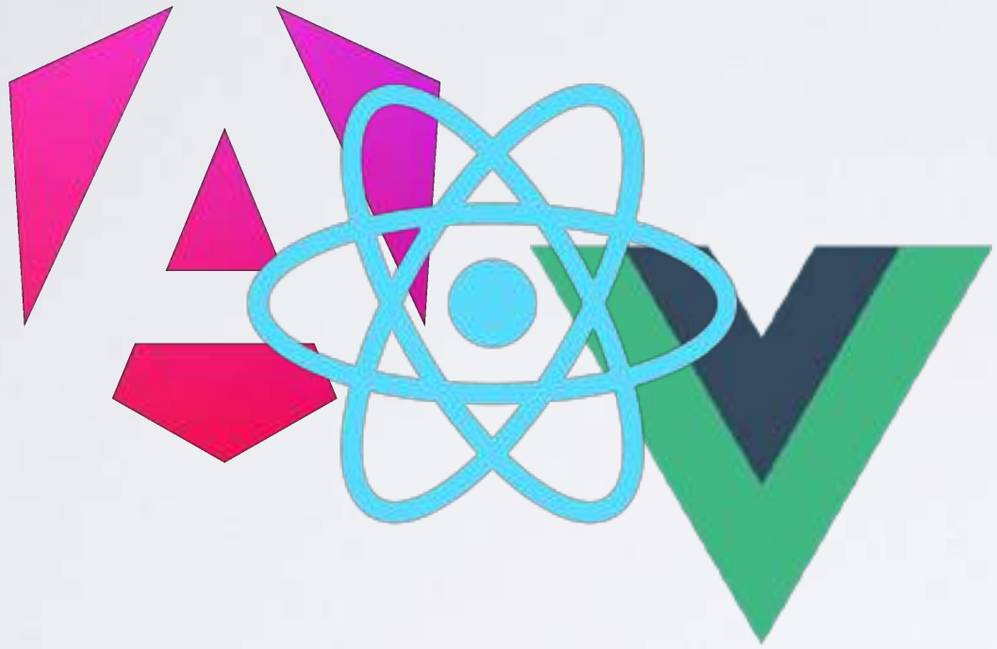


Potentially enabling Server Side Rendering:



https://angular.io/guide/router#appendix-locationstrategy-and-browser-url-styles

Server Side Rendering (SSR)



Today every modern frontend framework is capable of server side rendering.

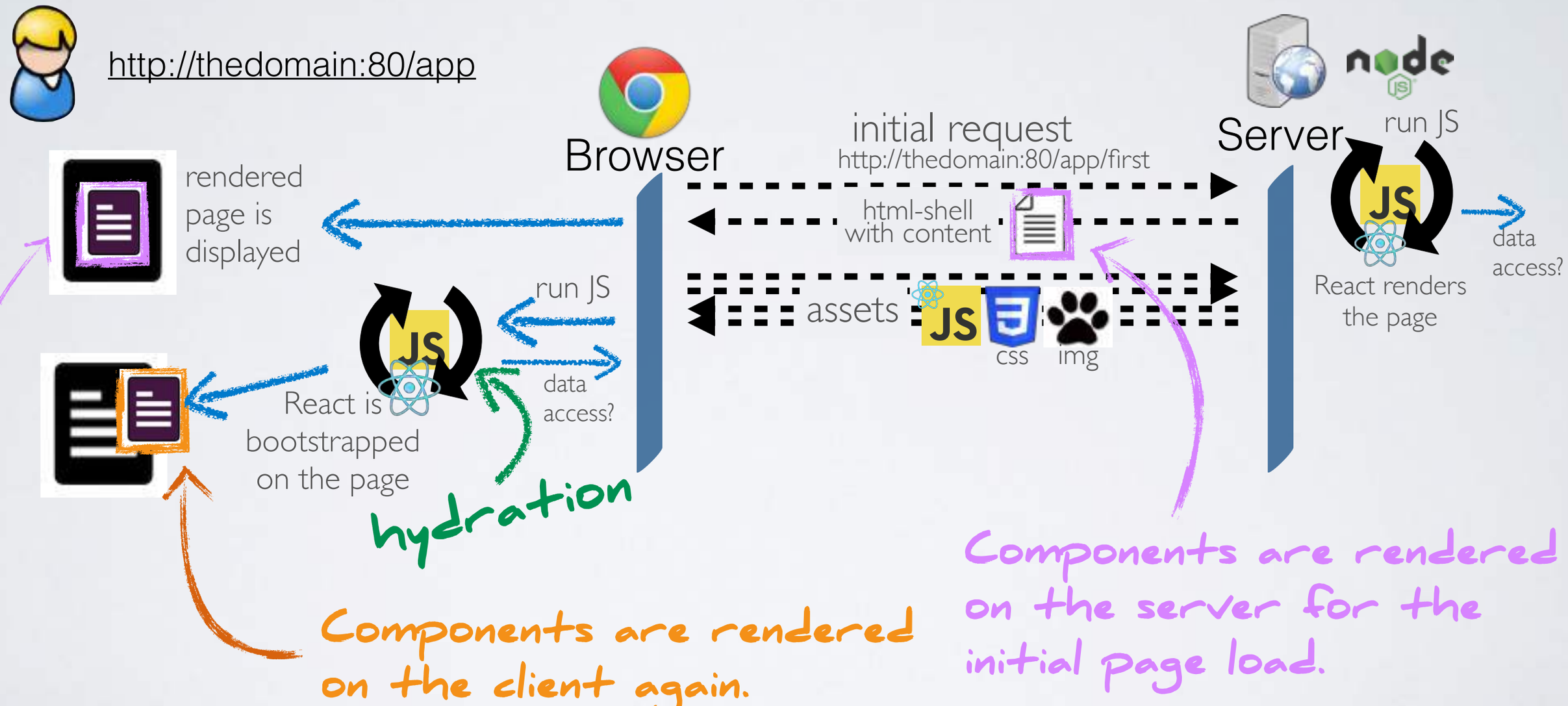
"Production-grade SSR" goes beyond rendering components to a html-string. This requires deeper integrations into bundling as well as data-fetching and routing.

- In React and Vue this one of the reason for "Meta-Framework": Next.js, Remix/ReactRouter v7, Nuxt, Quasar ...
- Angular is capable of SSR "out-of-the-box" since v17 and they are improving SSR in every version.

SPA with Server Side Rendering (SSR)

(*initial* rendering on the server - *hydration* on the client)

Analogy to a game engine: the first frame is rendered on the server,
every subequent frame is renderd on the client.



Advantages:

- search indexing / social previews
- improving time to first contentful paint

SSR has its own challenges:

- UX (page is not interactive on first render)
- Data Access (different mechanisms on the client and server)
- Browser APIs (not available on the server)
- rendering differences between client and server & hydration errors

"Full-Stack" Frameworks

NEXT.JS

React Router
Remix
v7

RedwoodJS

TanStack Start

Nuxt

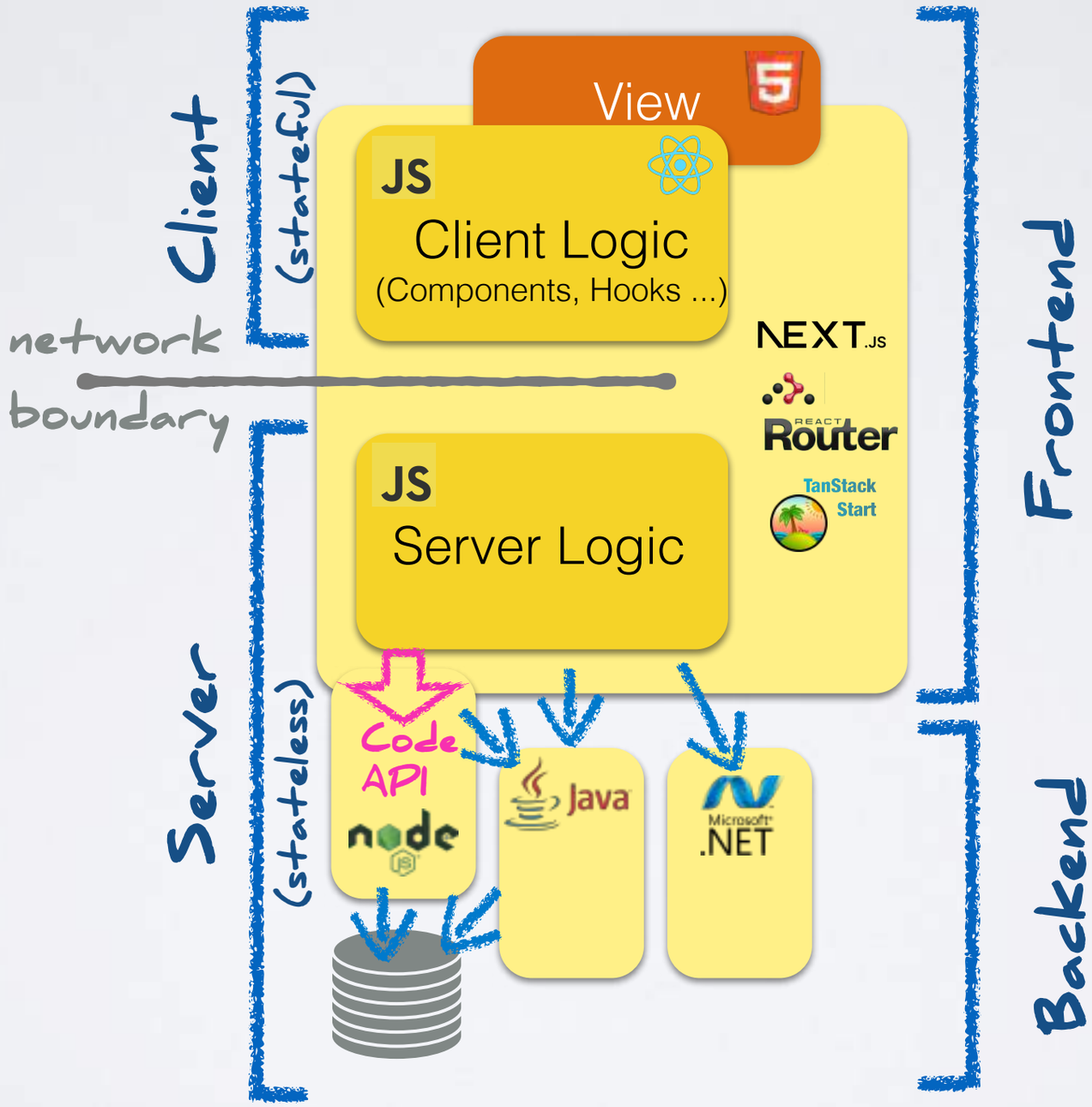
Analog

SOLIDSTART

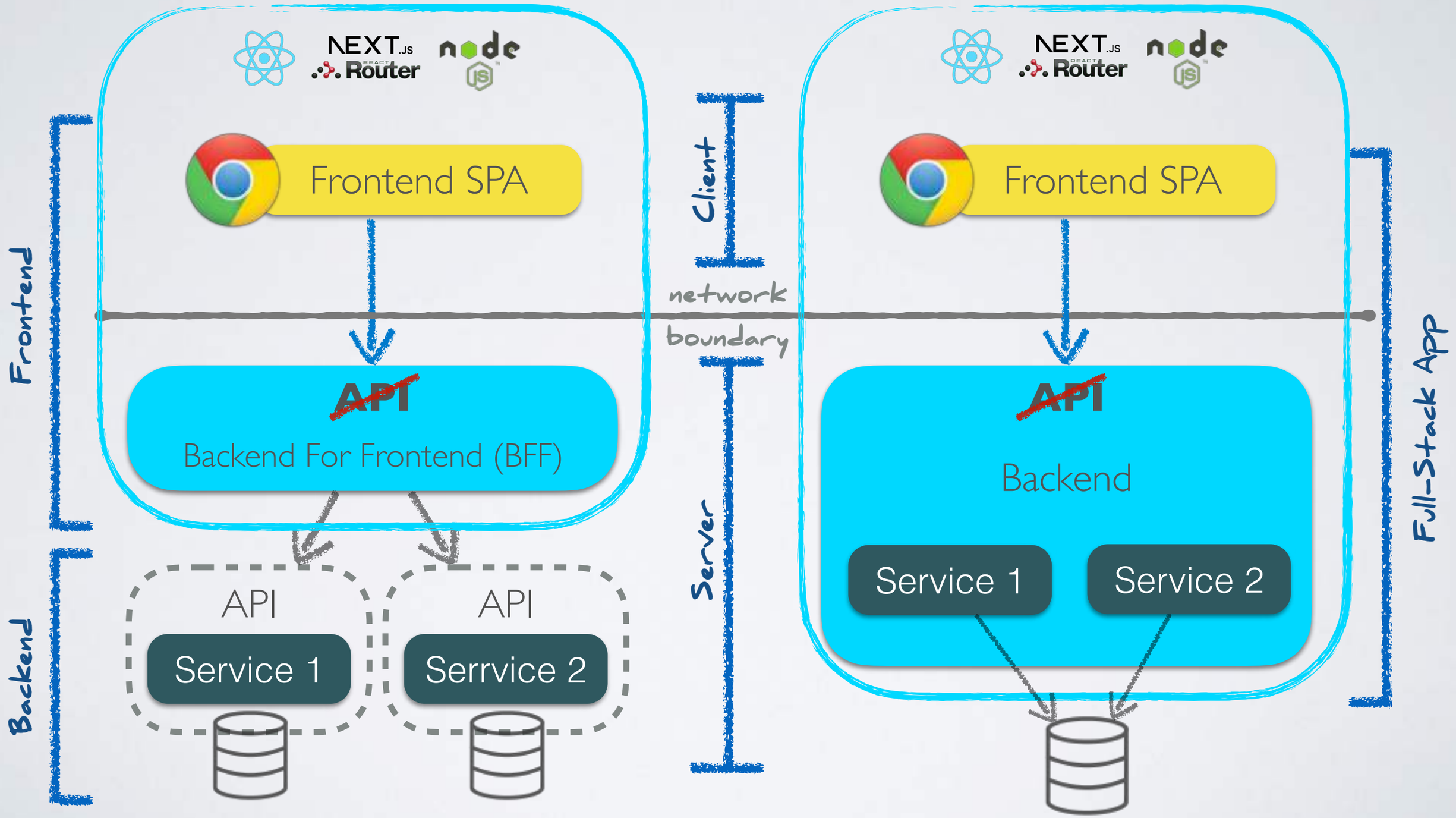
SVELTEKIT

qwik Qwik City

Full-Stack Frontend Frameworks

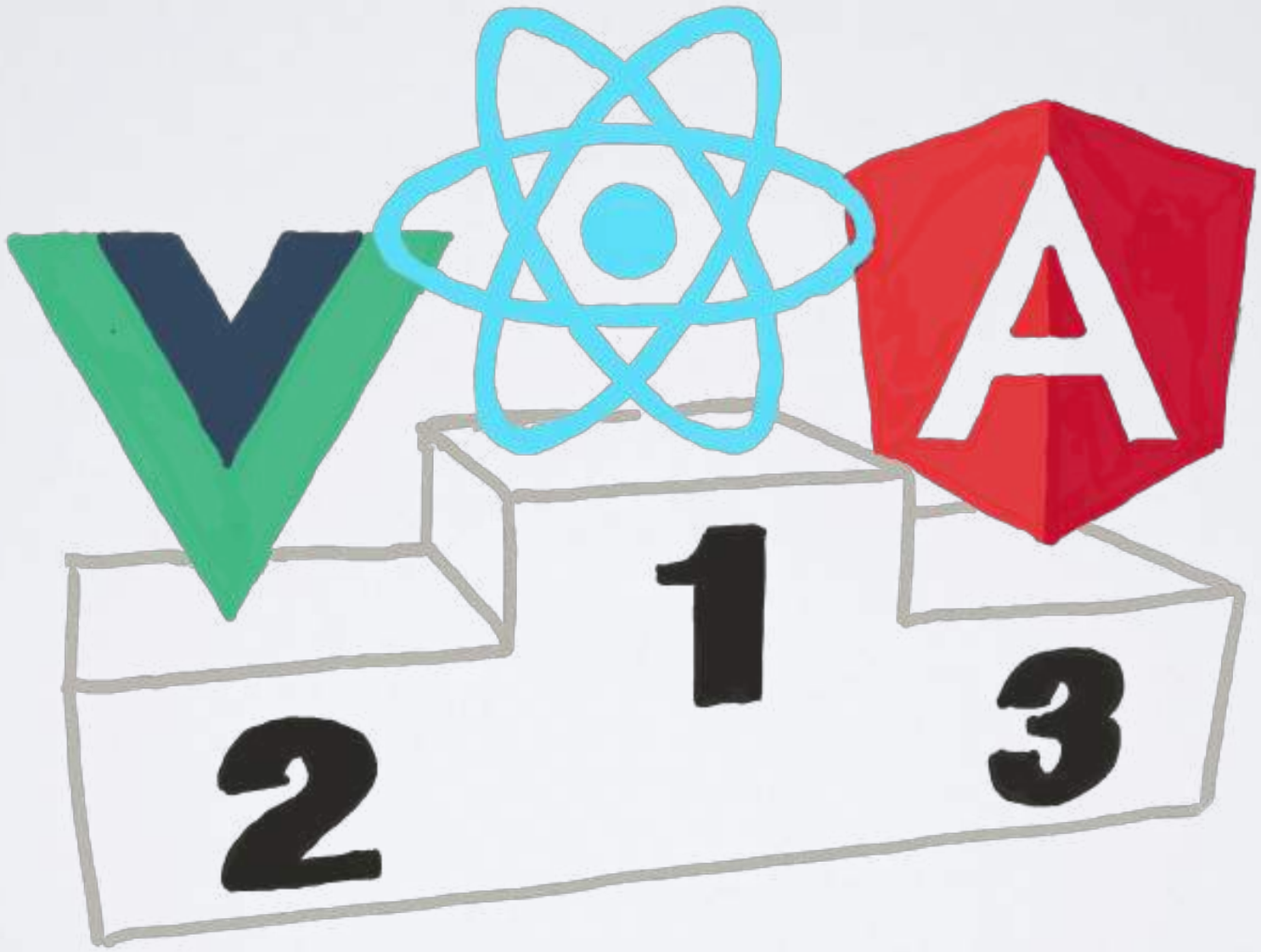


React Full-Stack Architectures



https://samnewman.io/patterns/architectural/bff/

Which one is The Best™ ?



The "Big 3" of SPA Frameworks

React

- A framework for single page applications (DOM is rendered by React)
- Components as the primary building blocks
- Views are based JavaScript (JSX)
- Leverages the power of JavaScript
- Works perfectly with TypeScript
- Just the "UI-library". Must be combined with other libraries for building applications.
- Very popular, many public applications (outlook.com, jira, zalando, netflix ...)

```
npm create vite@latest
```

https://react.dev/

Angular

- A framework for single page applications (DOM is rendered by Angular)
- Components as the primary building blocks
- Views are based on templates
- Aspires to be a "complete framework" for application development.
- Fully embraces TypeScript
- Popular in many "enterprise"-companies (government, banks, insurances ...) for in-house applications.
- Tendencies to move away from the "mainstream" JavaScript ecosystem
- Some unfortunate (legacy) design choices.
- "The Angular Renaissance" started in 2023 with v17

```
npx @angular/cli@latest new
```

https://angular.dev/

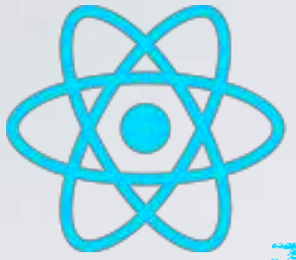
Vue.js

- A "progressive" framework
  - Can be used to enhance an existing DOM
  - Can be used to create complete client-side rendered single page applications
  - Components as the primary building blocks
  - Views are based on templates
  - Works perfectly with TypeScript
  - Many "similarities to AngularJS"
  - Very popular in Asia (Alibaba, Baidu ...).

```
npm create vue@latest
```

https://vuejs.org/

Framework Usage



React

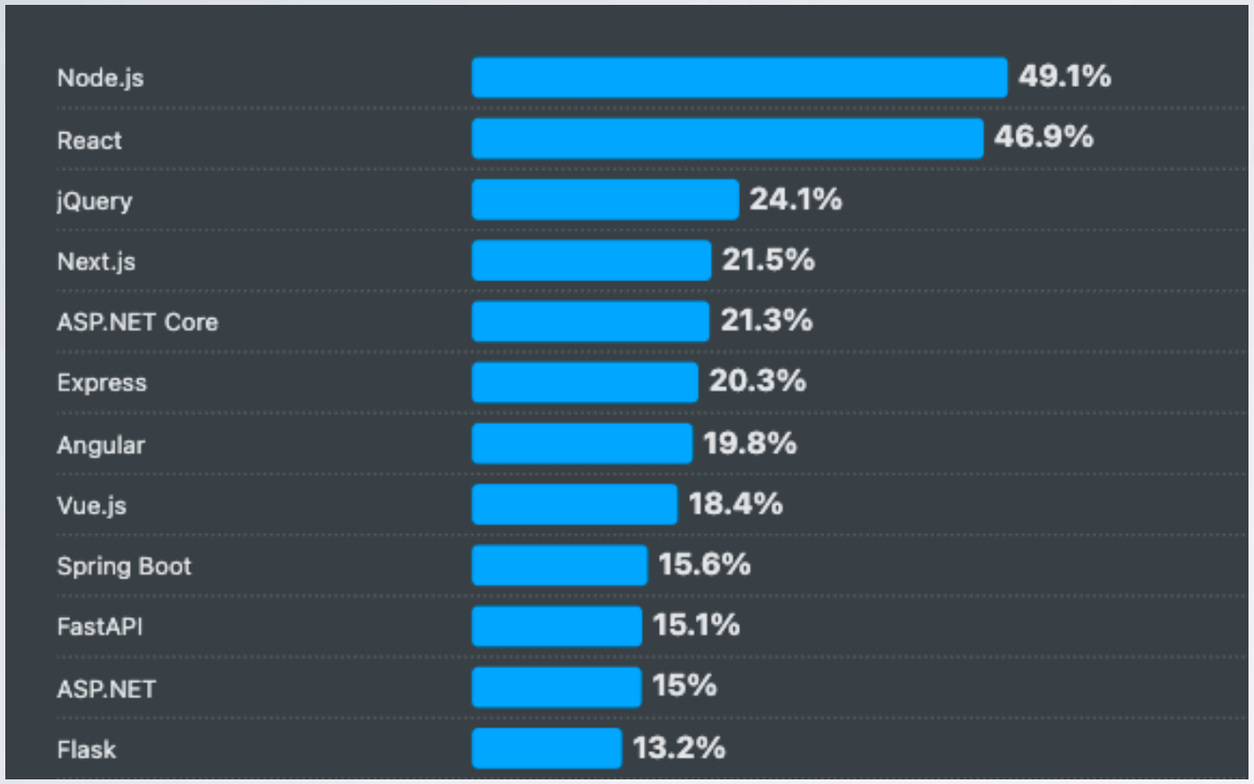
Angular

"2000+ projects at google"

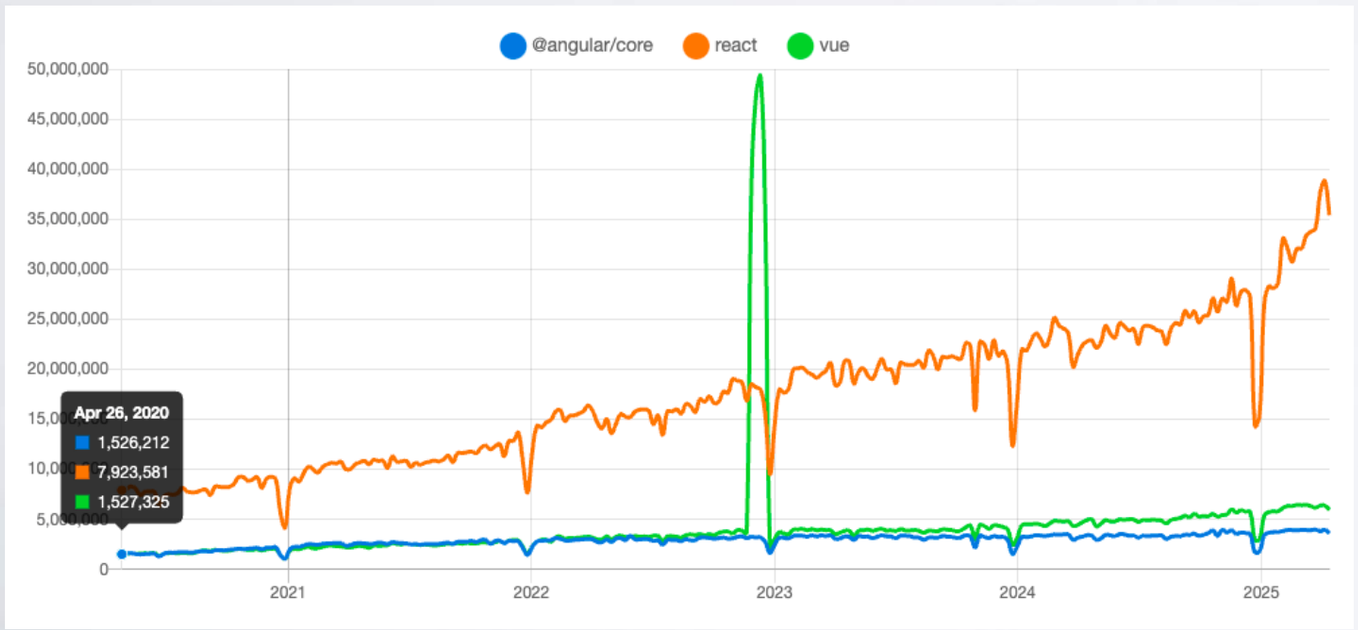
Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Vue.js

Popularity: Angular vs. React vs. Vue

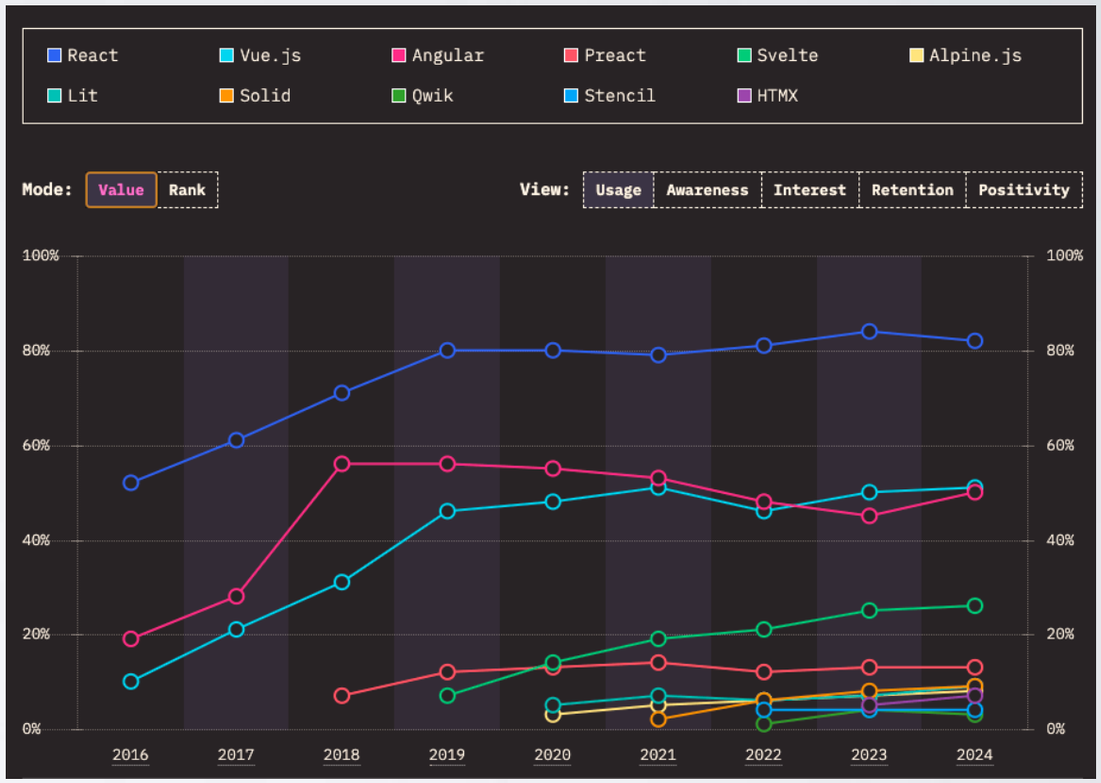


https://survey.stackoverflow.co/2025/technology#1-web-frameworks-and-technologies



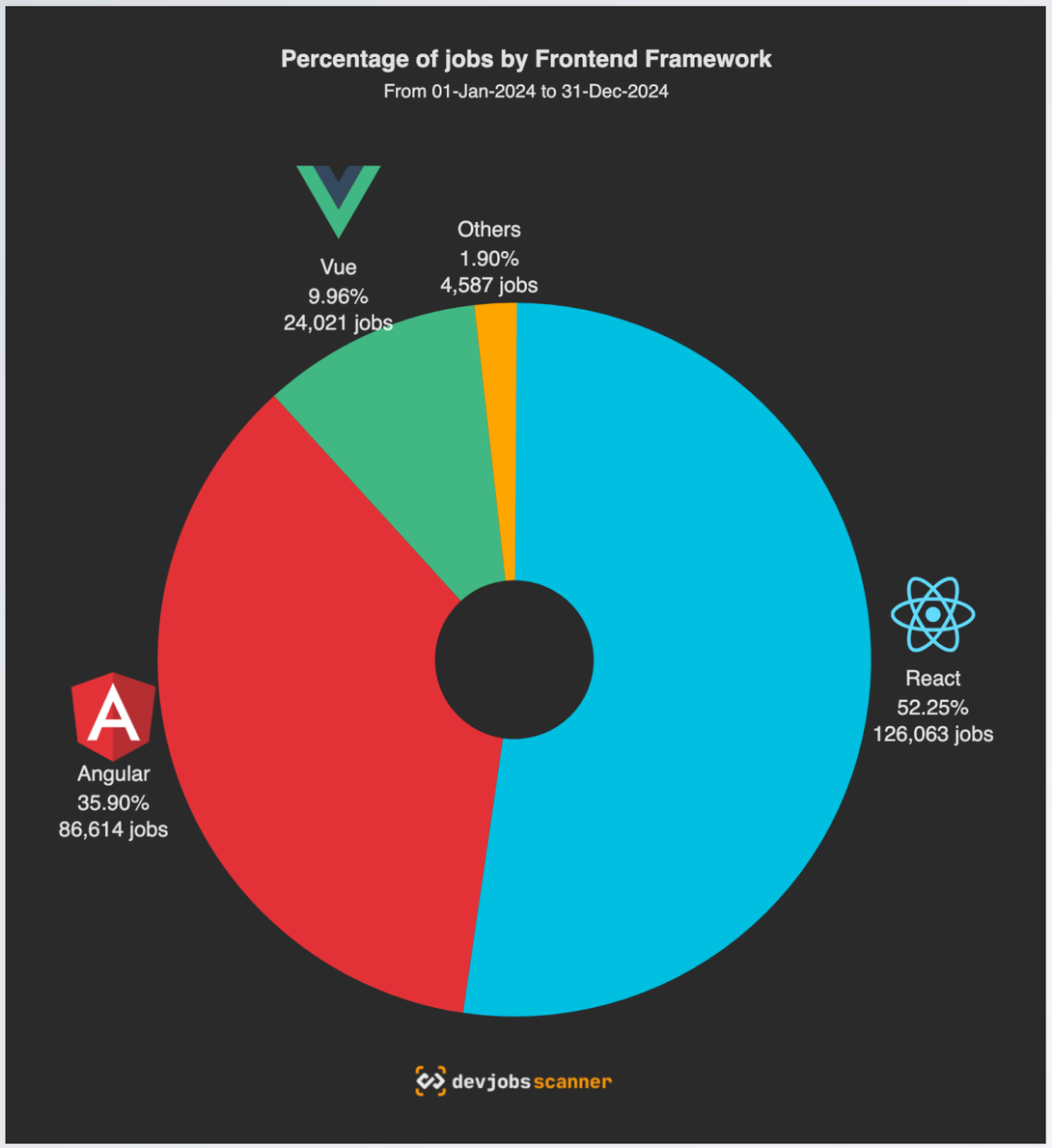
https://npmtrends.com/@angular/core-vs-react-vs-vue

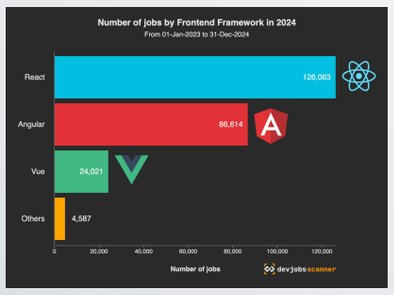
State of JavaScript Survey 2024

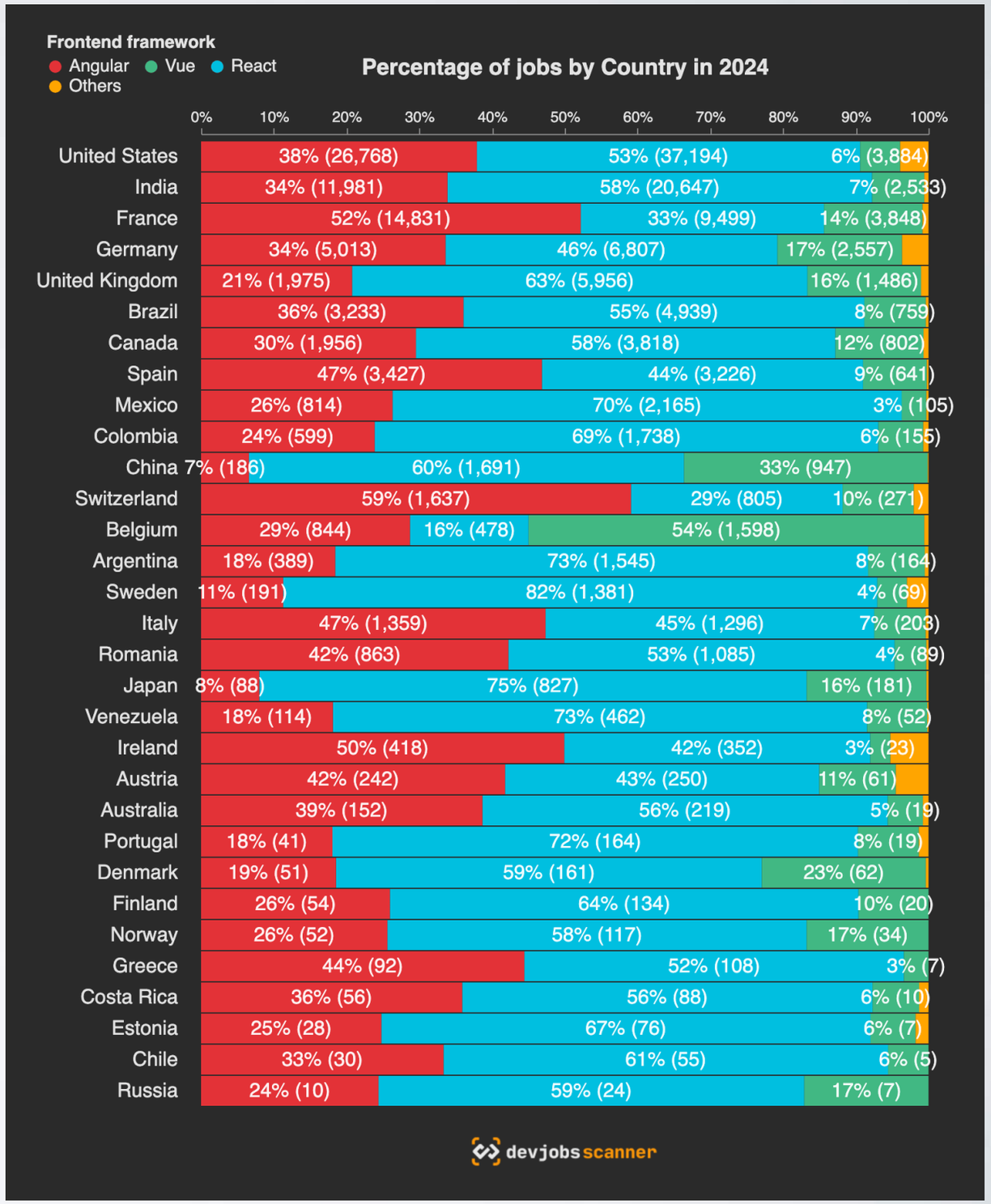


https://2024.stateofjs.com/en-US/libraries/front-end-frameworks/

Popularity: Angular vs. React vs. Vue



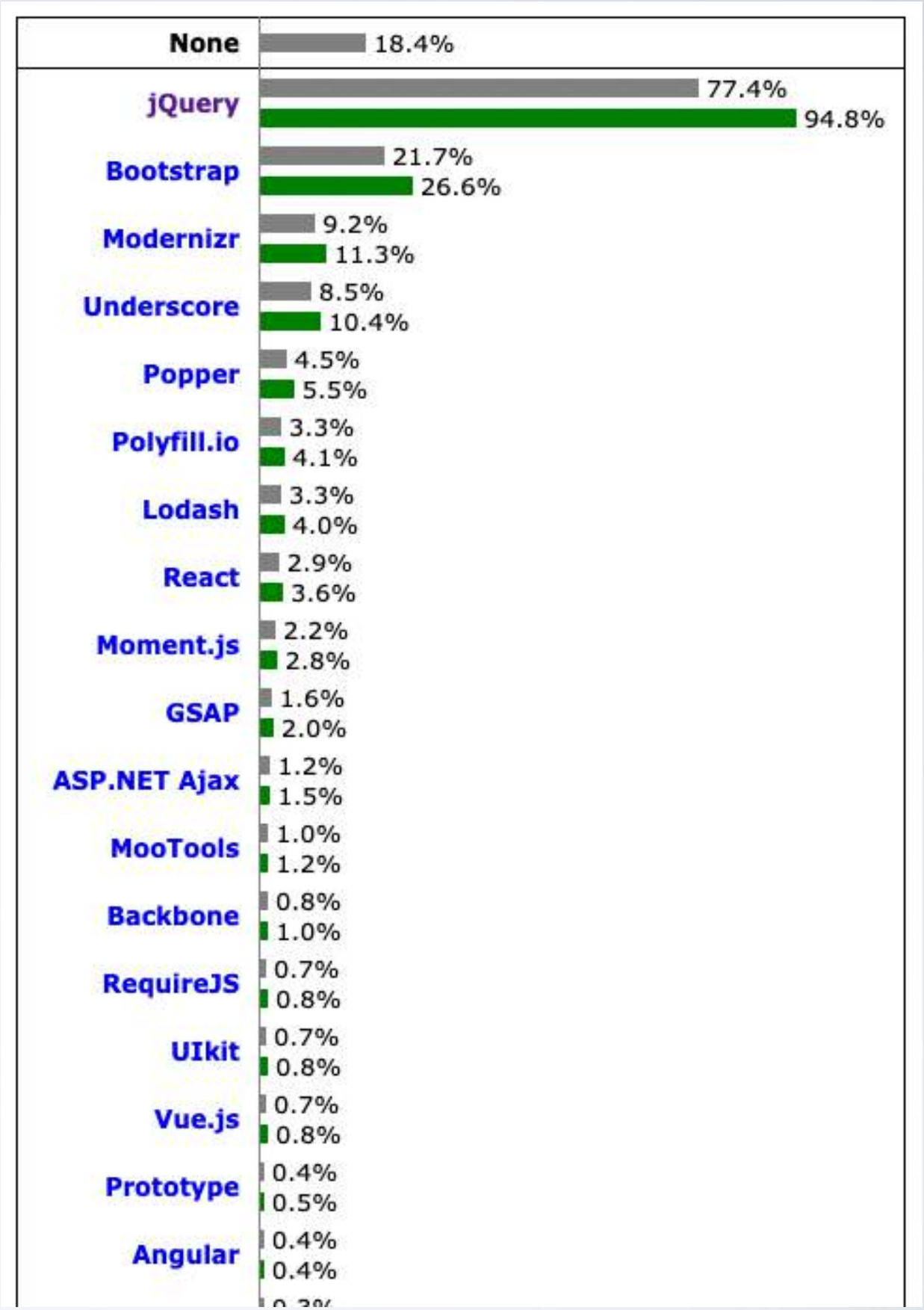




https://www.devjobsscanner.com/blog/the-most-demanded-frontend-frameworks/

🤫 by the way ...

... the web still runs on jQuery



https://w3techs.com/technologies/overview/javascript_library

Performance Comparison

Duration in milliseconds ± 95% confidence interval (Slowdown = Duration / Fastest)

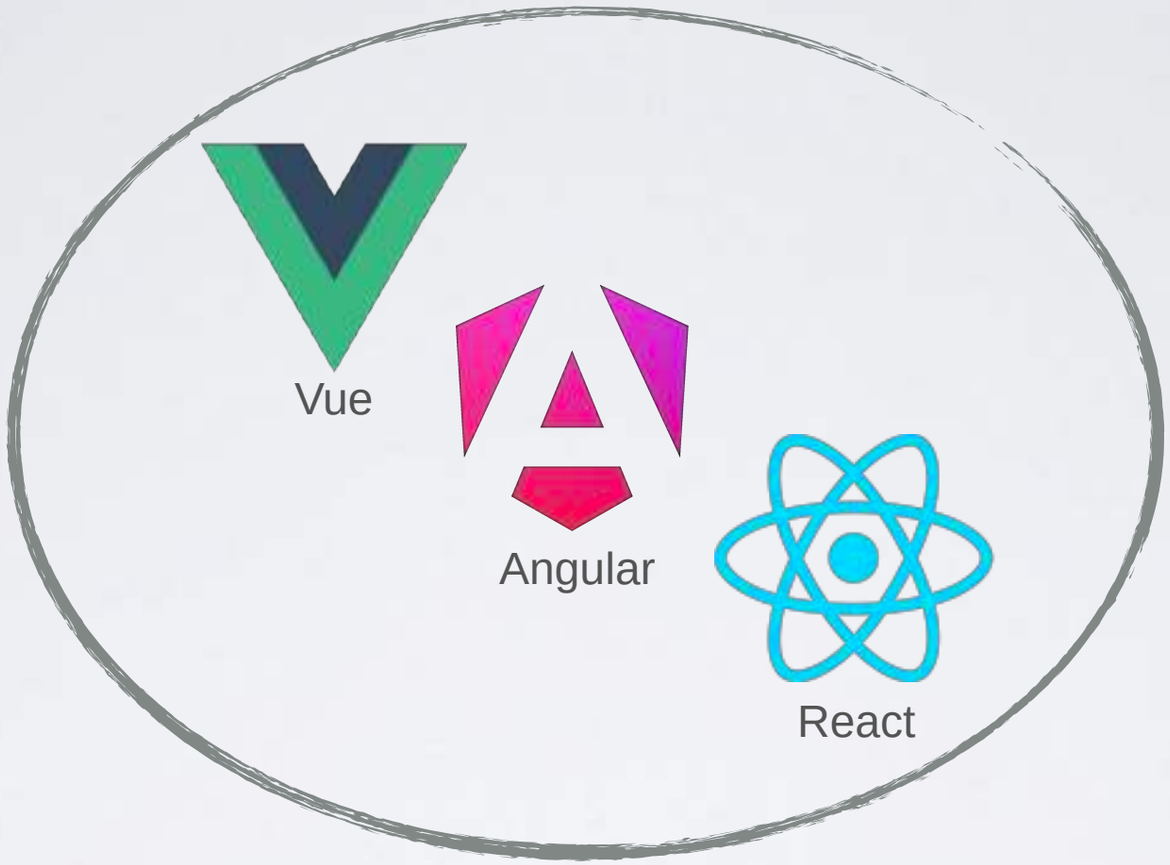
Name Duration for...	solid-v1.8.15	svelte-v5.0.5	vue-vapor-v3.5.6	vue-v3.5.3	angular-cf-nozone-v18.0.1	angular-cf-v18.0.1	angular-cf-signals-v18.0.1	react-jotai-v17.0.1 + 1.7.2	angular-ngfor-v18.0.1	react-compiler-hooks-v19.0.0-rc-4c58fce7-20240904	react-hooks-v18.2.0	react-zustand-v18.2.0 + 4.3.6
Implementation notes												
Implementation link	code	code	code	code	code	code	code	code	code	code	code	code
create rows creating 1,000 rows. (5 warmup runs).	38.2 ±0.3 (1.08)	38.8 ±0.2 (1.10)	43.6 ±0.3 (1.24)	45.3 ±0.3 (1.28)	47.4 ±0.3 (1.34)	48.1 ±0.2 (1.36)	48.0 ±0.3 (1.36)	51.1 ±0.3 (1.45)	48.3 ±0.2 (1.37)	47.3 ±0.4 (1.34)	47.6 ±0.3 (1.35)	50.5 ±0.4 (1.43)
replace all rows updating all 1,000 rows. (5 warmup runs).	43.8 ±0.2 (1.12)	43.7 ±0.3 (1.11)	48.6 ±0.4 (1.24)	50.5 ±0.4 (1.29)	55.2 ±0.2 (1.41)	58.7 ±0.5 (1.50)	59.3 ±0.3 (1.51)	56.3 ±0.4 (1.44)	57.7 ±0.3 (1.47)	54.3 ±0.2 (1.39)	56.9 ±0.3 (1.45)	61.9 ±0.2 (1.58)
partial update updating every 10th row for 1,000 row. (3 warmup runs). 4 x CPU slowdown.	18.0 ±0.4 (1.10)	18.3 ±0.3 (1.12)	21.3 ±0.3 (1.30)	21.0 ±0.4 (1.28)	19.3 ±0.3 (1.18)	20.6 ±0.3 (1.26)	19.9 ±0.5 (1.21)	23.7 ±0.4 (1.45)	20.2 ±0.5 (1.23)	23.0 ±0.4 (1.40)	23.0 ±0.3 (1.40)	26.1 ±0.4 (1.59)
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	3.5 ±0.2 (1.17)	4.4 ±0.2 (1.47)	5.0 ±0.2 (1.67)	5.0 ±0.2 (1.67)	5.6 ±0.3 (1.87)	6.3 ±0.2 (2.10)	6.5 ±0.2 (2.17)	4.7 ±0.2 (1.57)	6.2 ±0.2 (2.07)	7.7 ±0.2 (2.57)	7.7 ±0.3 (2.57)	8.2 ±0.3 (2.73)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	21.4 ±0.3 (1.09)	21.5 ±0.4 (1.09)	22.4 ±0.5 (1.14)	22.7 ±0.5 (1.15)	22.5 ±0.3 (1.14)	23.4 ±0.4 (1.19)	23.7 ±0.3 (1.20)	176.7 ±1.6 (8.97)	197.3 ±1.1 (10.02)	176.9 ±1.2 (8.98)	176.4 ±1.6 (8.95)	178.2 ±1.4 (9.05)
remove row removing one row. (5 warmup runs). 2 x CPU slowdown.	16.5 ±0.3 (1.09)	16.7 ±0.5 (1.11)	17.8 ±0.3 (1.18)	19.7 ±0.2 (1.30)	17.6 ±0.1 (1.17)	17.8 ±0.2 (1.18)	19.6 ±0.3 (1.30)	19.4 ±0.2 (1.28)	17.9 ±0.1 (1.19)	18.8 ±0.2 (1.25)	19.3 ±0.3 (1.28)	19.6 ±0.4 (1.30)

https://krausest.github.io/js-framework-benchmark/current.html

Conclusion:

- Angular, React and Vue are very close when it comes to framework performance.
- The differences are not significant for typical "line of business" applications.
- There are many modern frameworks that are faster than Angular, React and Vue.

Typical Single Page Applications

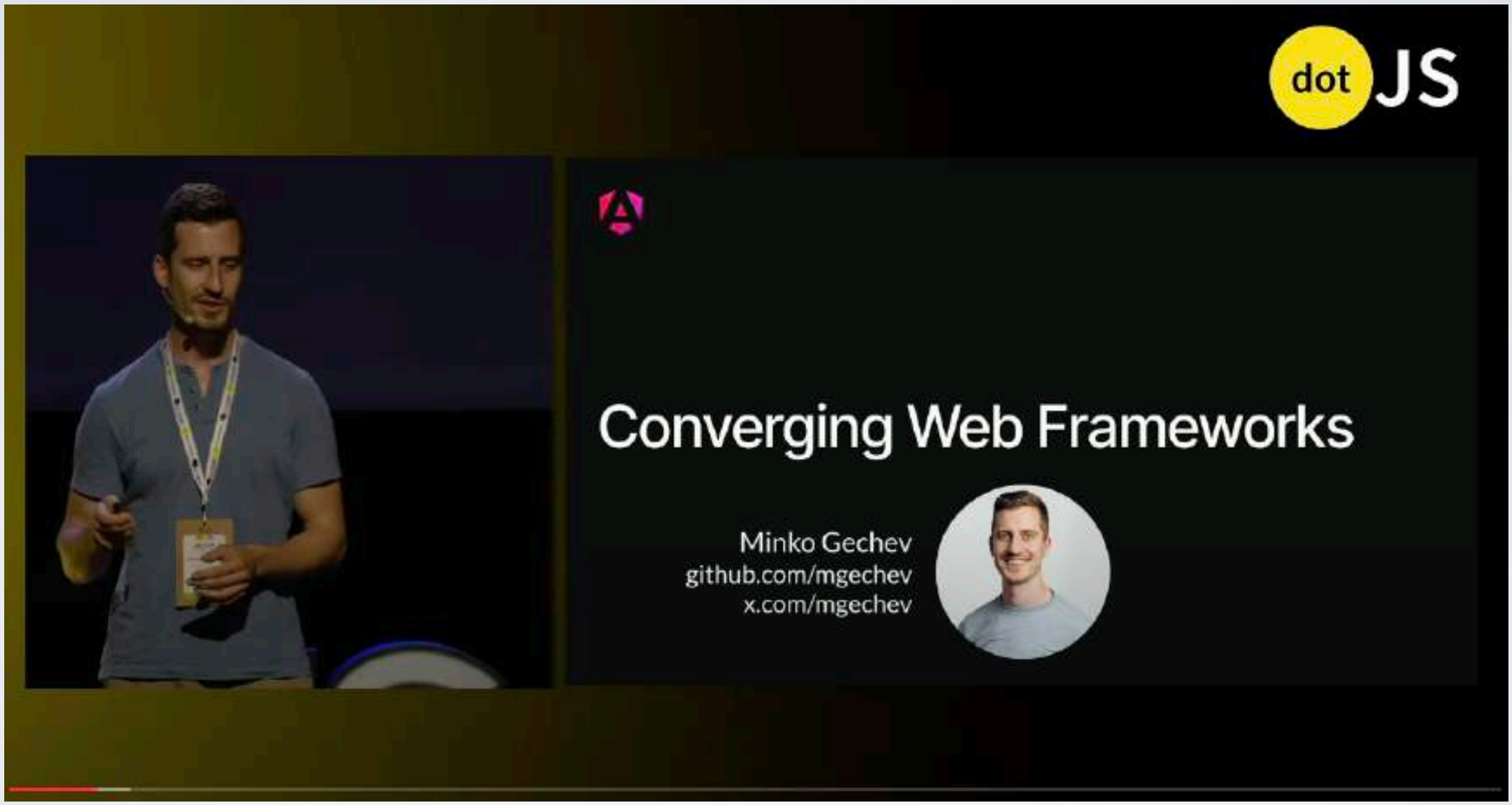


Angular, React and Vue Projects are very similar:

- based on modern JavaScript (incl. TypeScript support)
- built on top of the npm ecosystem
- require Node.js & npm tooling at build-time
- heavy conceptual separation of frontend and backend
- "unopinionated" about backend technologies
- embracing the Browser as "application platform"

Thesis:

Current Frameworks have become very similar



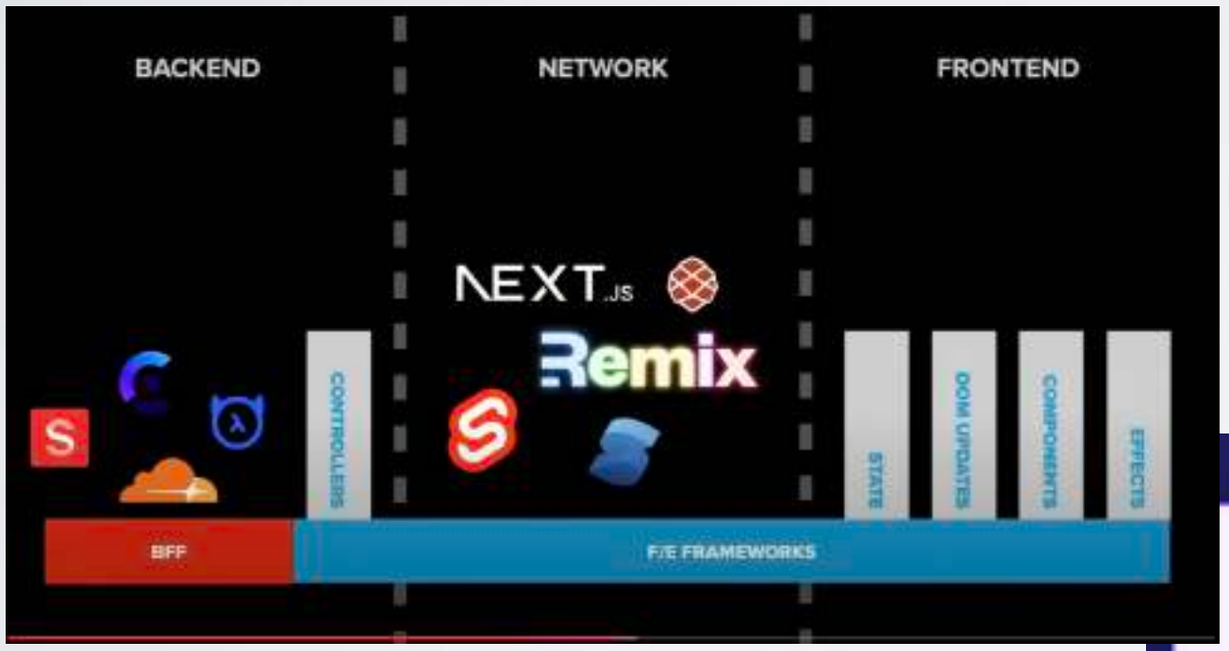
dotJS 2024 - Minko Gechev - Product Lead for Angular at Google
Converging Web Frameworks: https://www.youtube.com/watch?v=grRH8e46Pso

dotJS 2024 - Minko Gechev - Product Lead for Angular at Google
Converging Web Frameworks: https://www.youtube.com/watch?v=grRH8e46Pso

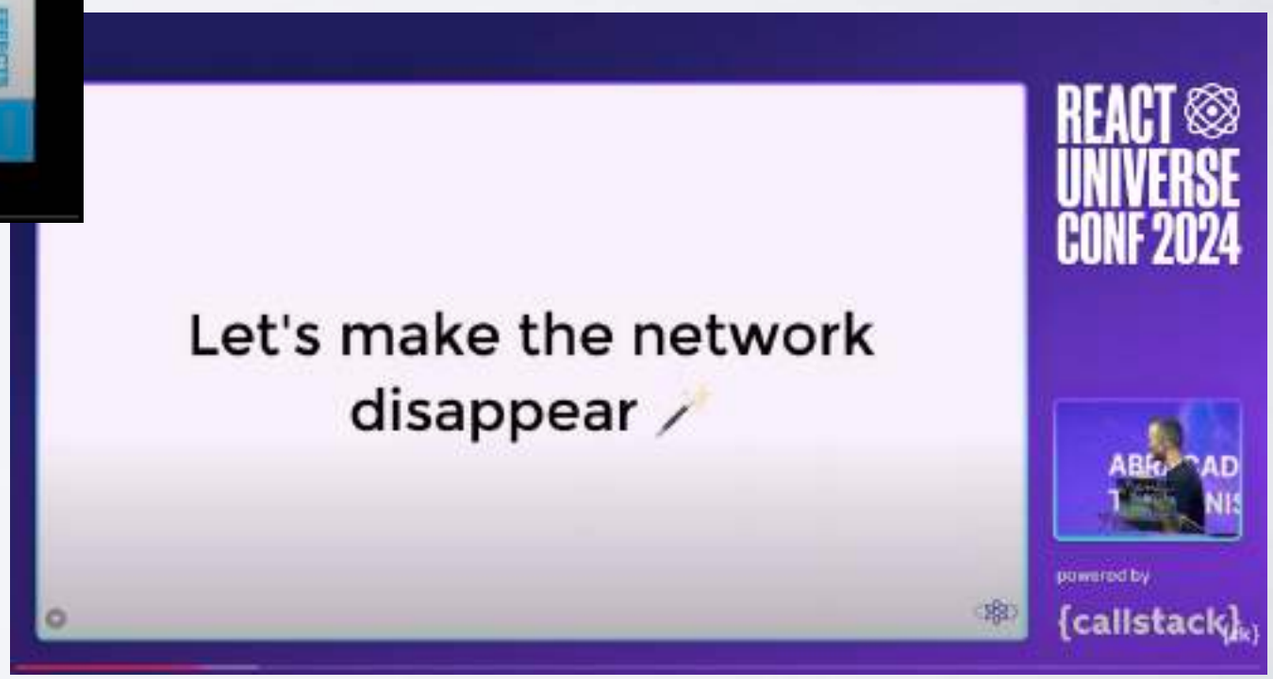
Consider non-technical factors:
Stability & Reliability, Community, Enjoyment

Thesis:

Innovation in frontend frameworks is moving towards the server-side and the full-stack perspective of web development.



"Mind The Gap" by Ryan Florence
at Big Sky Dev Con 2024
https://www.youtube.com/watch?v=zqhE-CepH2g



Abracadabra: The vanishing network — Kent C. Dodds | React Universe Conf 2024
https://www.youtube.com/watch?v=E8LLty9rTWw

Does it even matter?

Angular, React and Vue are very similar.

Choosing the flavor of "state-of-the-art" SPA frontend framework is *not a critical success factor* for your project.
Architecture, tooling, development process and organizational factors have *an order of magnitude more influence* on the success of your project.

Key Differences

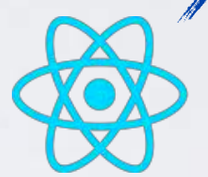
Templating

templates: enhancing html with dynamic constucts

separate files or inlined strings

single file components

(Vue also supports render functions with JSX)

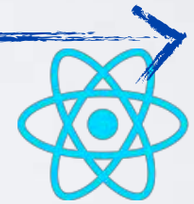


JSX: declaring UI with code

Framework vs Library

"complete" framework with opinionated choices

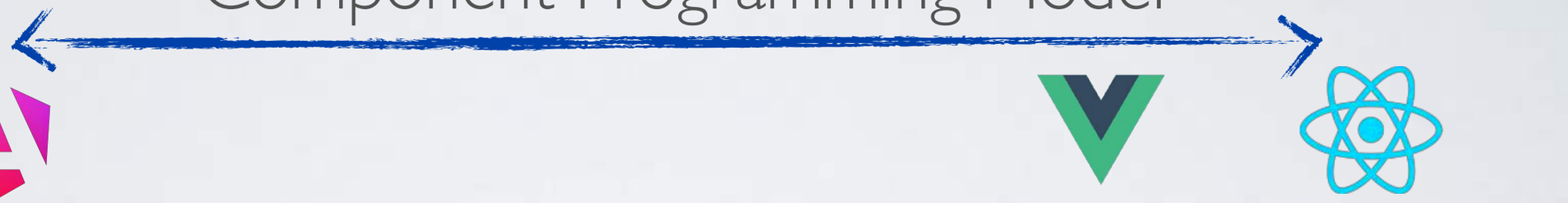
small & unopinionated library providing default options for important topics (router, centralized state management)



small & unopinionated library that must be combined with other libraries

Key Differences

Component Programming Model

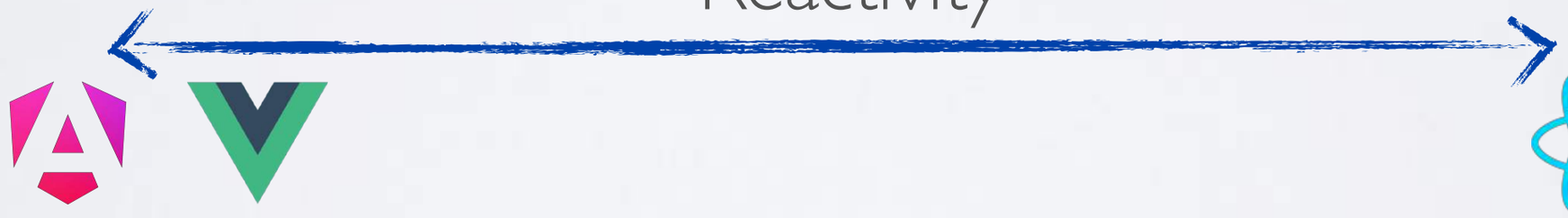


based on classes

functional API (composition API)
object based API is still available.

new functional API
("Hooks")

Reactivity



fine grained reactivity with signals

coarse grained render functions
(optimized by the React Compiler)

Angular 17 - The Angular Renaissance

https://www.youtube.com/live/Wq6GpTZ7AX0?si=tD6RO6rFQvZMNW5O

New Homepage:
https://angular.dev/

New Logo:

Angular is changing ...

"The Angular Renaissance"

https://twitter.com/Enea_Jahollari/status/1669008447042473985

Full-Stack React

(aka. Meta Frameworks)

The official React documentation is recommending to use a "production-grade framework".

https://react.dev/learn/creating-a-react-app#full-stack-frameworks

NEXT.JS

https://nextjs.org/

React Router v7

Remix "Framework Mode"

https://reactrouter.com/

The common goal of those meta-frameworks is to simplify the project setup of frontend applications.
They include concepts for server-side-rendering, routing, data-fetching, mutations ...
But they come with their own conceptual overhead and learning curve!
These frameworks typically require running JavaScript on the server (like Node.js).

RedwoodJS

RedwoodSDK

https://redwoodjs.com/
https://docs.rwsdk.com/

TanStack Start

https://tanstack.com/start/

MODERN.JS

https://modernjs.dev/

Waku

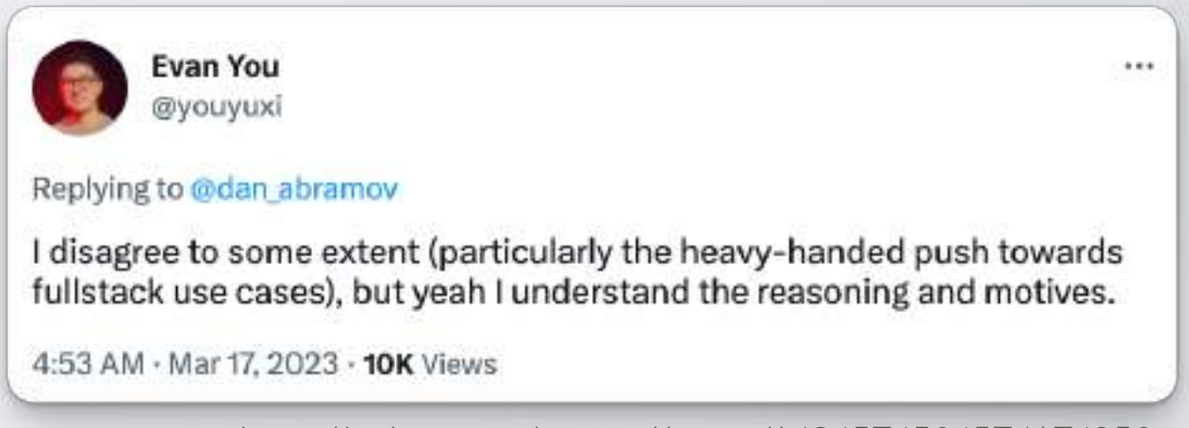
https://waku.gg/

Remix was merged into React Router v7: https://remix.run/blog/react-router-v7

2023 - the year when React became Full-Stack ...

In 2023 the "new" React doumentation was officially published,
recommending to use a "production grade" framework ...

https://twitter.com/RyanCarniato/status/1636501181039276032



https://twitter.com/youyuxi/status/1636576506574176258

https://twitter.com/1Marc/status/1636481900381388802

https://twitter.com/_cloudmu/status/1636900018643775488

https://react.dev/learn/creating-a-react-app#full-stack-frameworks
https://react.dev/learn/build-a-react-app-from-scratch

The React/Vercel Controversy

Vercel is a company that provides cloud infrastructure:
https://vercel.com/

Vercel is funding Next.js, the most popular React metaframework.

Vercel hired several React core developers.

The React documentation started to recommend "production grade frameworks".

Next.js started to use unreleased features of React.
(bundling "canary version" of React)

Next.js started promoting React Server Components, which run on the backend / in the cloud 🤔.

Next.js is not easy to self-host, some features only run on Vercel Infrastructure.

Vue: less Hype, less Drama

It is often overlooked that:

- Vite was create in the Vue ecosystem
  - also **`unjs`** and **`Nitro`**, which are widely used outside of the Vue ecosystem
- Vue was the first mainstream framework which provided signal-style reactivity (fine grained reactivity)
- Pionieered single-file components processed by build tools
- Vue is a progressive framework enabling a wide range of usage scenarios
- Vue remains independent (i.e. not dominantly owned by a single company)

Conclusion

- Angular tries to be the "stable and reliable" framework. But right now it is in a phase of heavy changes.

- React is the wild west:
  You have to choose a framework on top of React and/or choose & maintain a stack of libraries.

- Vue is currently the most stable framework and many innovations came out of the Vue ecosystem.

Many Similarities!

There are *many similarities* between Angular, React & Vue:

Single Page Application
Component Architecture
State managed in JavaScript
Binding the DOM to JavaScript
Dataflow Architecture
Typically "complex" frontend build setup based on Node & npm
Support for TypeScript

The *main differences* are:

- The mechanism to declare the UI
  (template vs. render function)
- Scope of the framework/library (which functionality is "built in"
  and how flexible is it to combine with other libraries)
- "Reactivity" concept: change detection which triggers UI updates

The Wild West of React

React is just a "library for creating user interfaces".

The ecosystem around React is the wild wild west.

Metaframeworks:

- Next.js
- Remix
- Redwood
- TanStack Start

Routers:

- React Router
- TanStack Router
- Chicane
- Wouter

Data Fetching Libraries:

- TanStack Query
- SWR
- Apollo
- axios
- ky
- ...

State Management Libraries:

- Zustand
- Jotai
- Legend State
- Recoil
- Redux
- MobX
- ...

Component Libraries:

- MUI
- Chackra UI
- Ant Design
- Radix UI
- shadcn
- Next UI
- ...

Styling Libraries:

- CSS Modules
- Emotion.js
- StyleX
- TSS React
- Pigment CSS
- Vanilla Extract
- Tailwind
- ...

Form Libraries:

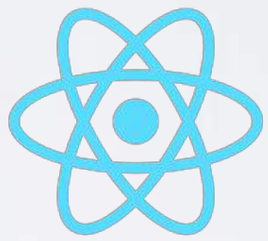
- Formik
- Rect Hook Forms
- HouseForm
- Felte
- Conform
- ...

i18n Libraries:

- i18next
- React-intl
- Lingui
- FBT

Who do you (want to) trust?

Google 

facebook 

Open Source Community 

(with many financial sponsors)