

Modern React Development with Next.js and Server Components

ABOUT ME

Jonas Bandi

jonas.bandi@ivorycode.com

Twitter: [@jbandi](https://twitter.com/@jbandi)



- Freelancer, in den letzten 12 Jahren vor allem in Projekten im Spannungsfeld zwischen modernen Webentwicklung und traditionellen Geschäftsanwendungen.
- Dozent an der Berner Fachhochschule seit 2007
- In-House Kurse & Beratungen zu Web-Technologien im Enterprise: UBS, Postfinance, Mobiliar, AXA, BIT, SBB, Elca, Adnovum, BSI ...



Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

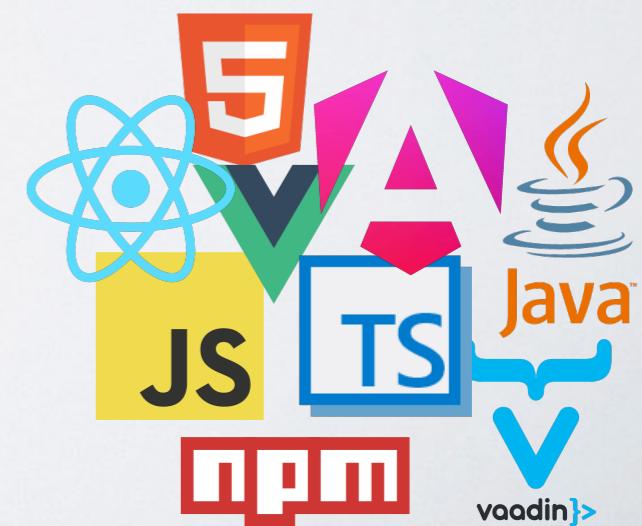


letsboot.ch
swiss dev training



JavaScript / Angular / React / Vue / Vaadin
Schulung / Beratung / Coaching / Reviews

jonas.bandi@ivorycode.com



About you ...

Agenda

Goal: Driven by Demo/Exercise:

[https://github.com/ivorycode/nextjs-chopen-2025/
blob/main/00-CourseMaterial/Exercises.md](https://github.com/ivorycode/nextjs-chopen-2025/blob/main/00-CourseMaterial/Exercises.md)

- Create your first Next.js App
- Routes, Layouts and Styling
- Introduce Components
- DB-Access with Drizzle ORM
- Data Fetching with Server Components
- Mutations with Server Functions
- Introducing a Client Component

Theory/Slides for Illustration

(Personal Goal: "Public Vibe Coding")

Material

Git Repository:

<https://github.com/ivorycode/nextjs-chopen-2025/>

Initial Clone:

git clone https://github.com/ivorycode/nextjs-chopen-2025/

Update: **git pull**

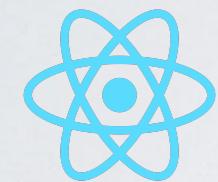
**git reset --hard
git clean -dfx
git pull**

(discard all local changes)

Slides & Exercises: <checkout>/00-CourseMaterial

NEXT.JS

Evolution of React



React
(Virtual Dom)

2013

React Native
components are based on ES2015 classes

2015

splitting React
Create 0.14
Create React App
NEXT.js
React 15

2016

React 16.8: Hooks
("Function Components for everything")

2019

Vite 2.0
React 18
Next.js App Router with RSCs

2022

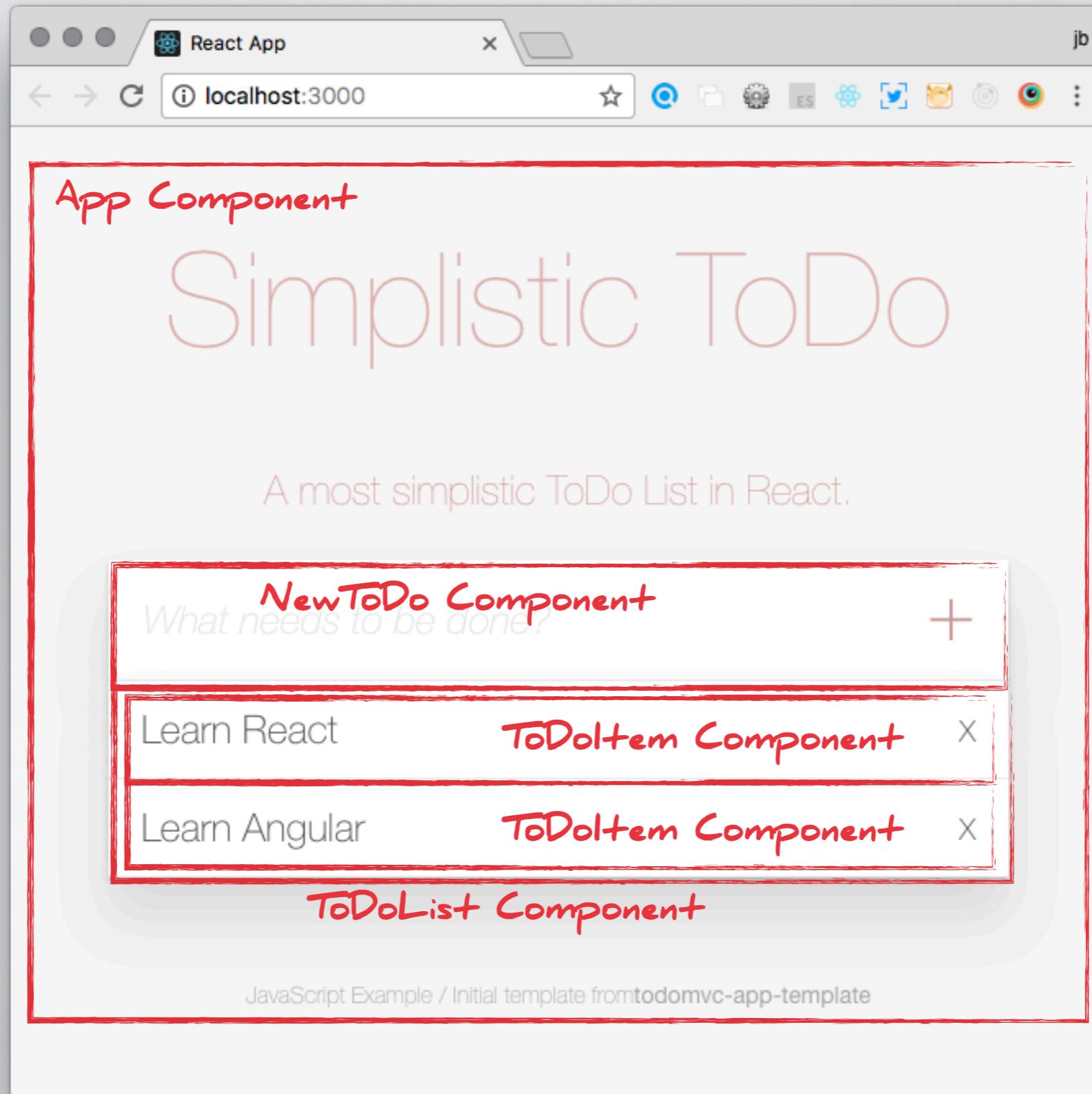
"official" release of RSCs
React Compiler

2024

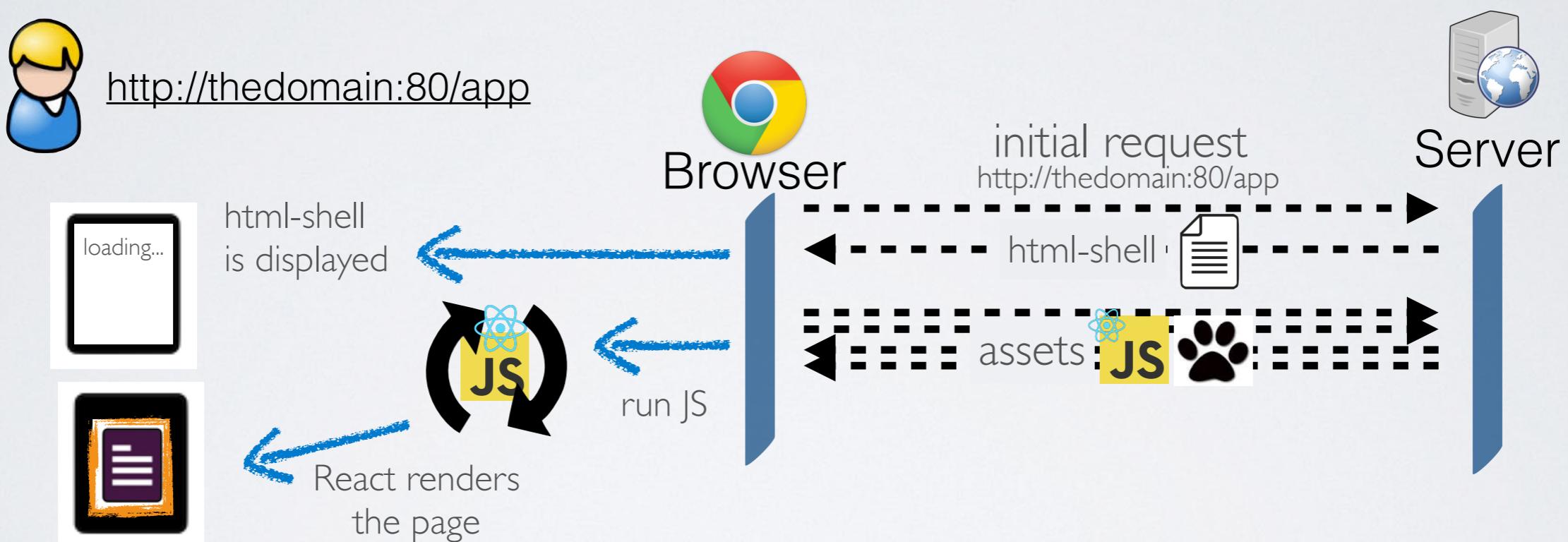
2025



Everything is a Component



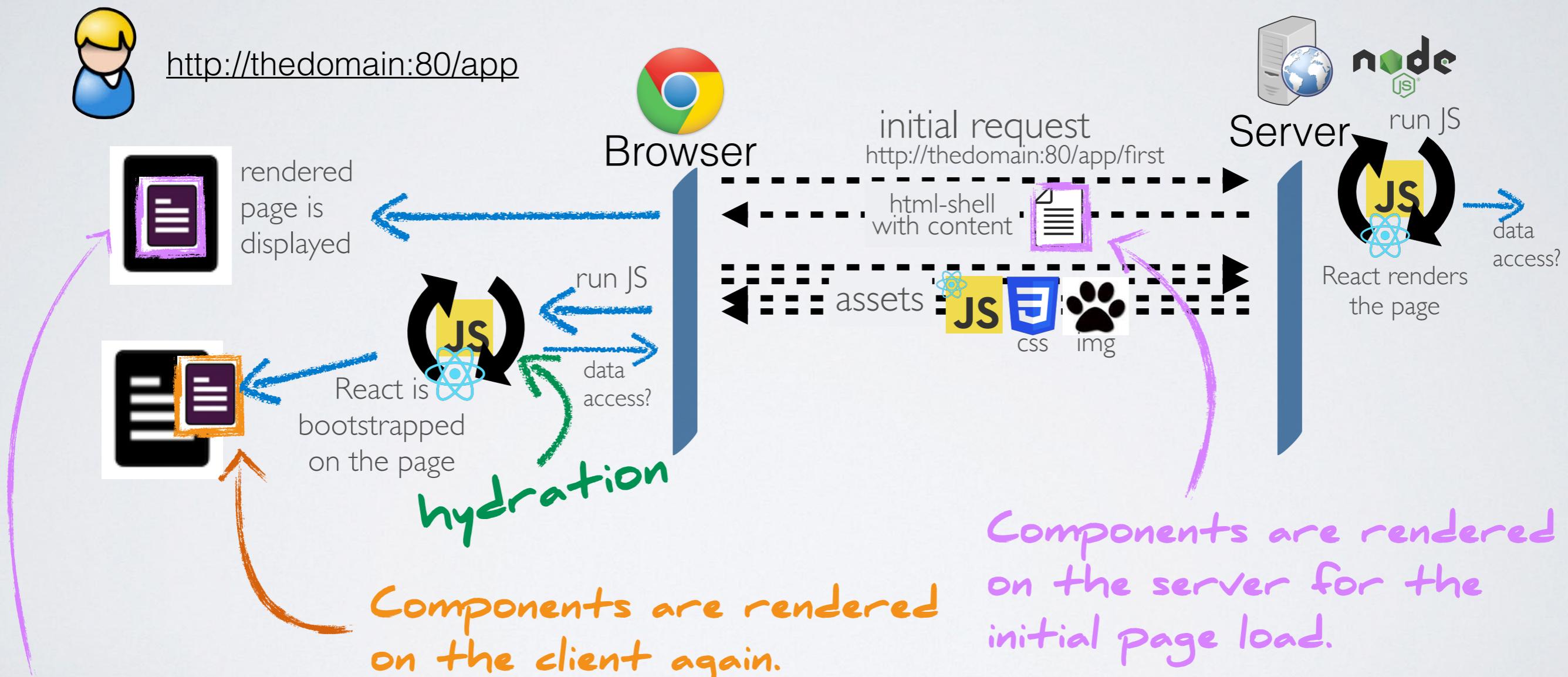
Traditional SPA: Client Side Rendering



**Components are rendered
on the client.**

SPA with Server Side Rendering (SSR)

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



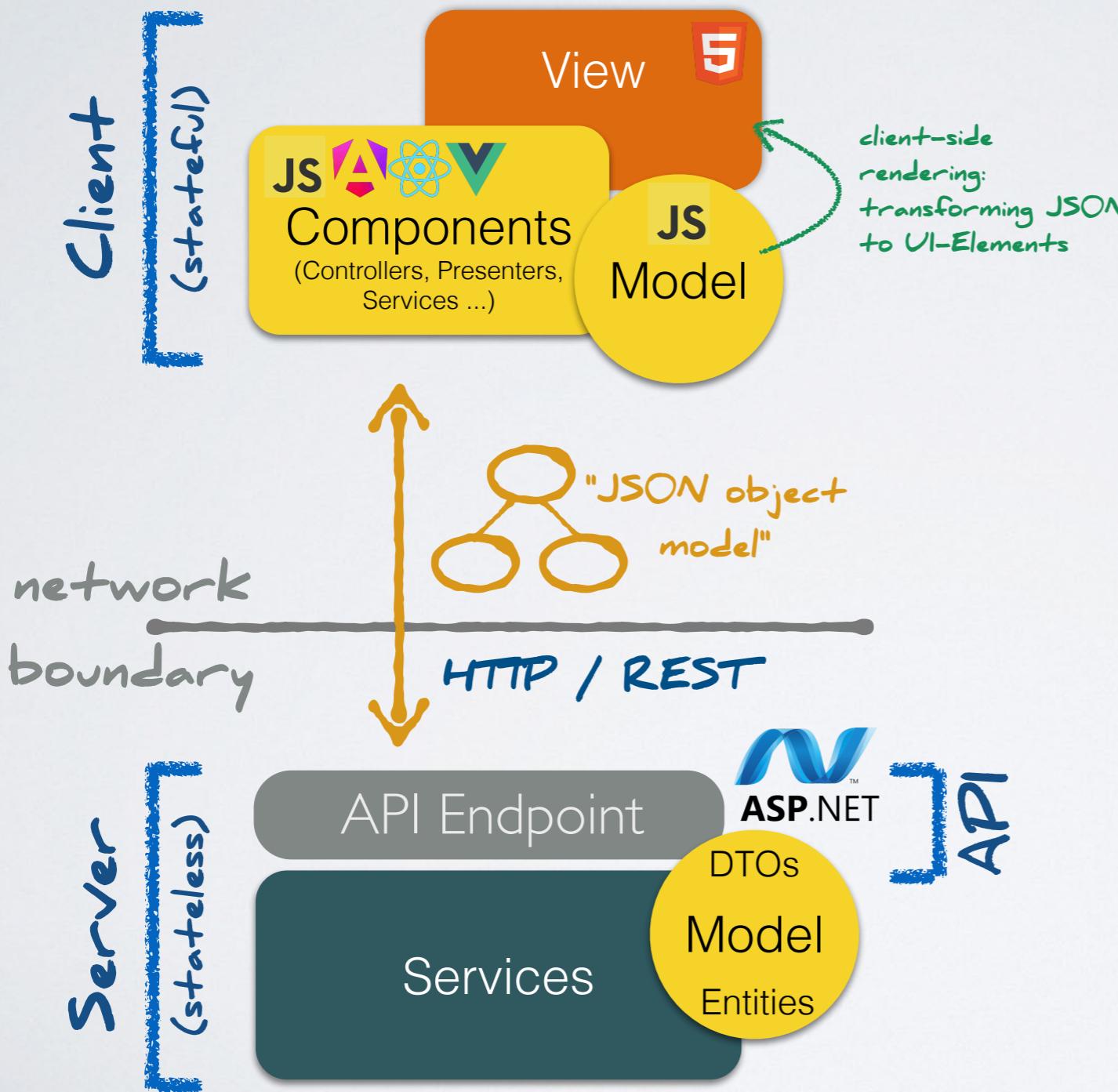
Advantages:

- 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)

SPA Architecture

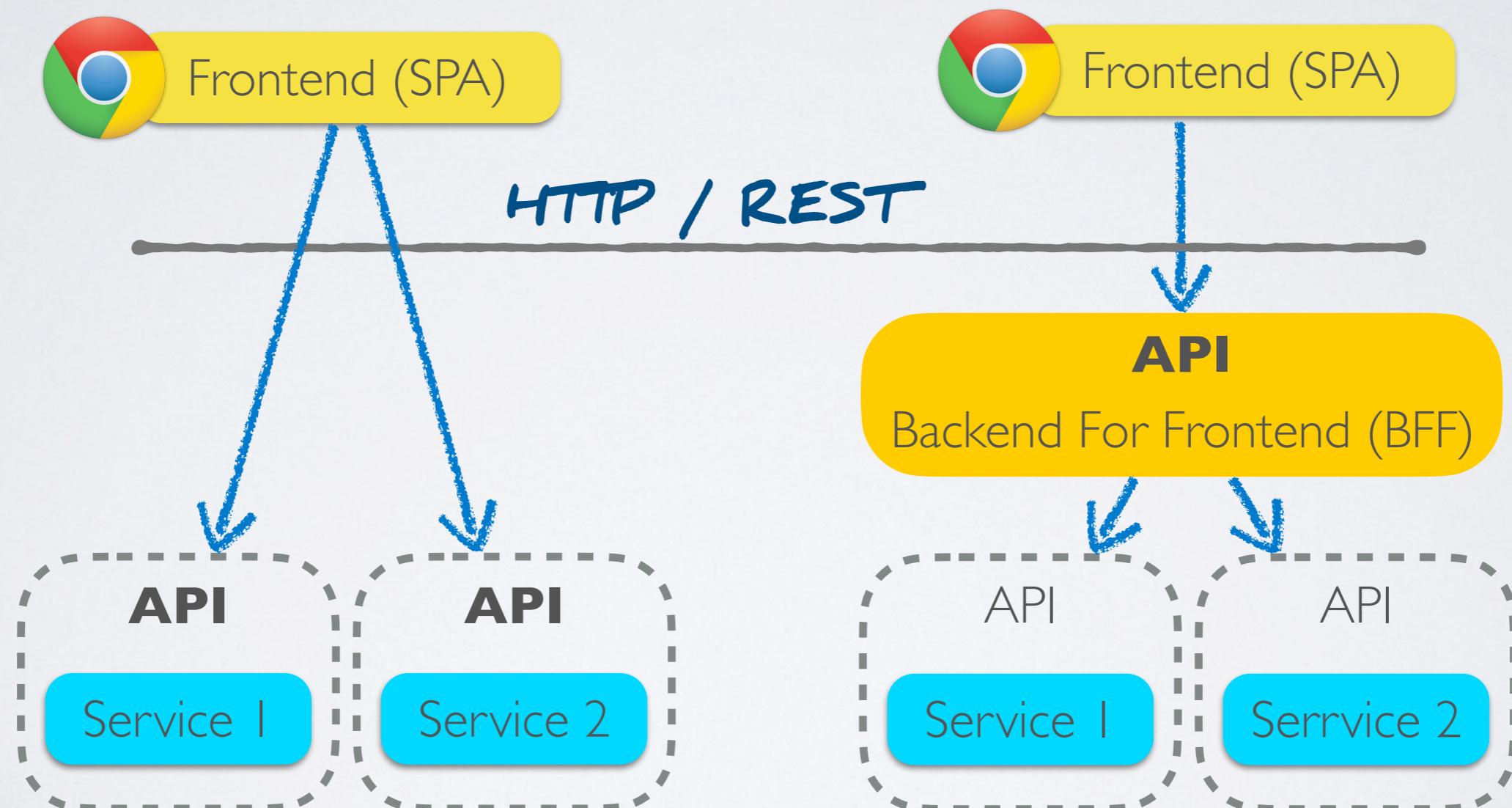


Rich client programming model in the browser.

Clear separation of concerns between client and server.

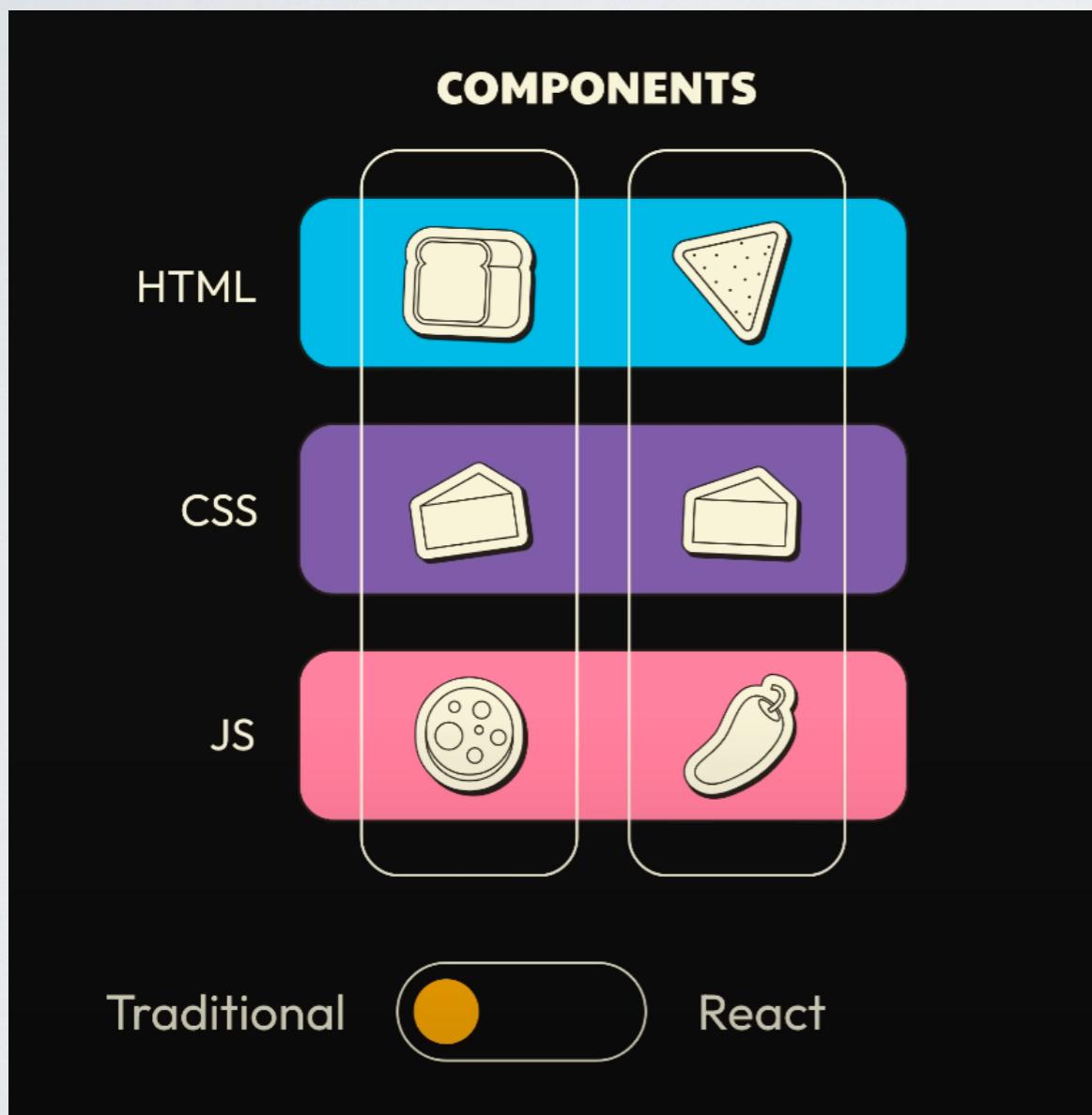
Traditional Architectures for SPAs

Client - API - Server

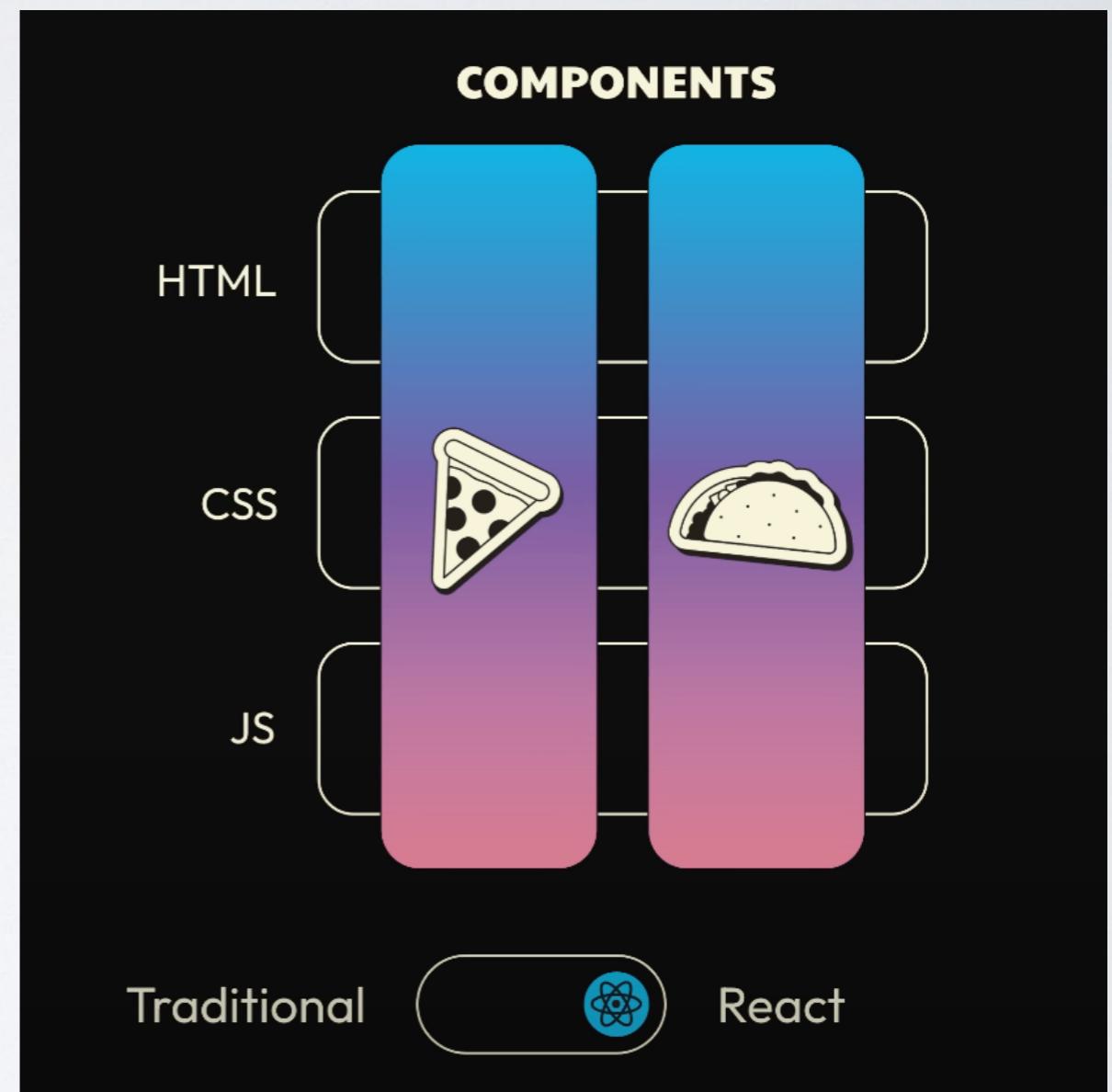


Separation of Concerns

Traditional Separation of Concerns in Web Dev:



Separation of Concerns in React:



Images from *ui.dev: The Story of react Query*
<https://www.youtube.com/watch?v=OrliU0e09io>



Dan Abramov

@dan_abramov

Following

Separating concerns by files is as effective as separating school friendships by desks. Concerns are “separated” when there is no coupling: changing A wouldn’t break B. Increasing the distance without addressing the coupling only makes it easier to add bugs.

9:37 PM - 16 Jun 2018

540 Retweets 1,817 Likes



40

540

1.8K



https://twitter.com/dan_abramov/status/1008131488481730561

React uses JSX to describe the UI in a declarative way:

```
function Counter() {  
  const [count, setCount] = useState(0);  
  
  function increase() {  
    setCount(count + 1);  
  }  
  
  return (  
    <div>  
      <h1>Count {count}</h1>  
      <button onClick={increase}>Increase</button>  
    </div>  
  );  
}
```



JSX

JSX is not an "embedded string". It is compiled at build time into code that produces an element-tree at runtime.

JSX is an XML-like syntax extension to ECMAScript without any defined semantics:
<https://facebook.github.io/jsx/>

JSX is used in many other frontend libraries: Preact, Solid.js, Qwik, Stencil, Vue.js, Inferno, Brisa ...

Back to Separation of Concerns

Theoretically you could split a component into a "controller" and a "view":

Controller.ts

```
import {View} from './View';

export function Controller {
    // state & behavior
    const data = ... /* fetched from API */

    function doWork() { ... }

    // delegate rendering
    return (
        <View data={data} onEvent={doWork} />
    );
}
```

View.js

```
export function View({data, onEvent}){
    return (
        <div>
            {data.message}
            <button onClick={()=>onEvent()}>
                Go!
            </button>
        </div>
    );
}
```

Note: This is an "academic" example that does not represent idiomatic usage of React. But the patterns "lifting state up" and "Container- and Presentation-Components" is based on this concepts:

<https://react.dev/learn/sharing-state-between-components>

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

The Virtual DOM



App

Title

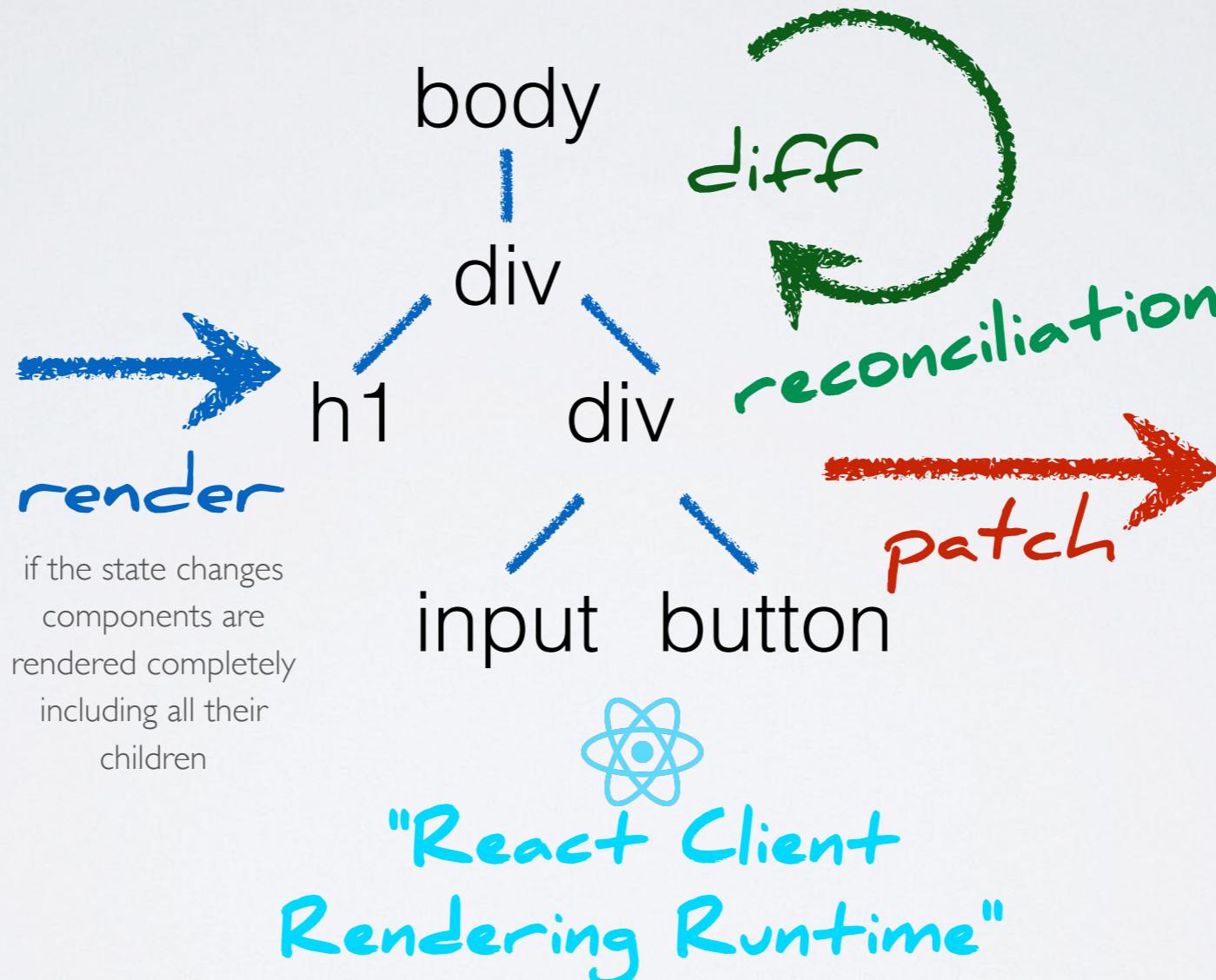
Content

input

button

Virtual DOM

In-Memory, implemented in JavaScript

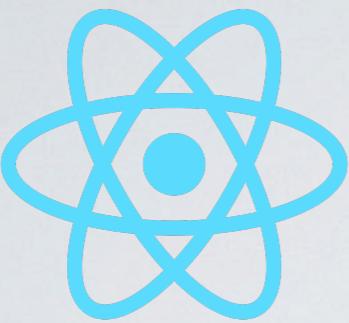


Browser DOM



```
<body>
<div>
<h1></h1>
<div>
<input/>
<button>
    </button>
</div>
</div>
</body>
```

The Virtual DOM also "enables" server-side rendering and rendering to iOS/Android UIs.



React Data-Access

effect functions
can't be async

ignoring
stale responses

cleanup function

effect
dependencies

```
export function useApiData() {
  let ignore = false;
  const [data, setData] = useState("");
  useEffect(() => {
    async function fetchData() {
      const messageText = await fetchDataFromApi();
      if (!ignore) {
        setData(messageText);
      }
    }
    fetchData();
  }, []);
  return () => {
    ignore = true;
  };
}, []);
return data;
}
```

no loading state, no
error state, no
caching



The sad face of React...

<https://react.dev/learn/synchronizing-with-effects#fetching-data>

The Story of React Query: <https://youtu.be/OrliU0e09io?si=ZVVHRfnUCgm-gCTw&t=127>

Full-Stack React

(aka. Meta Frameworks)

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

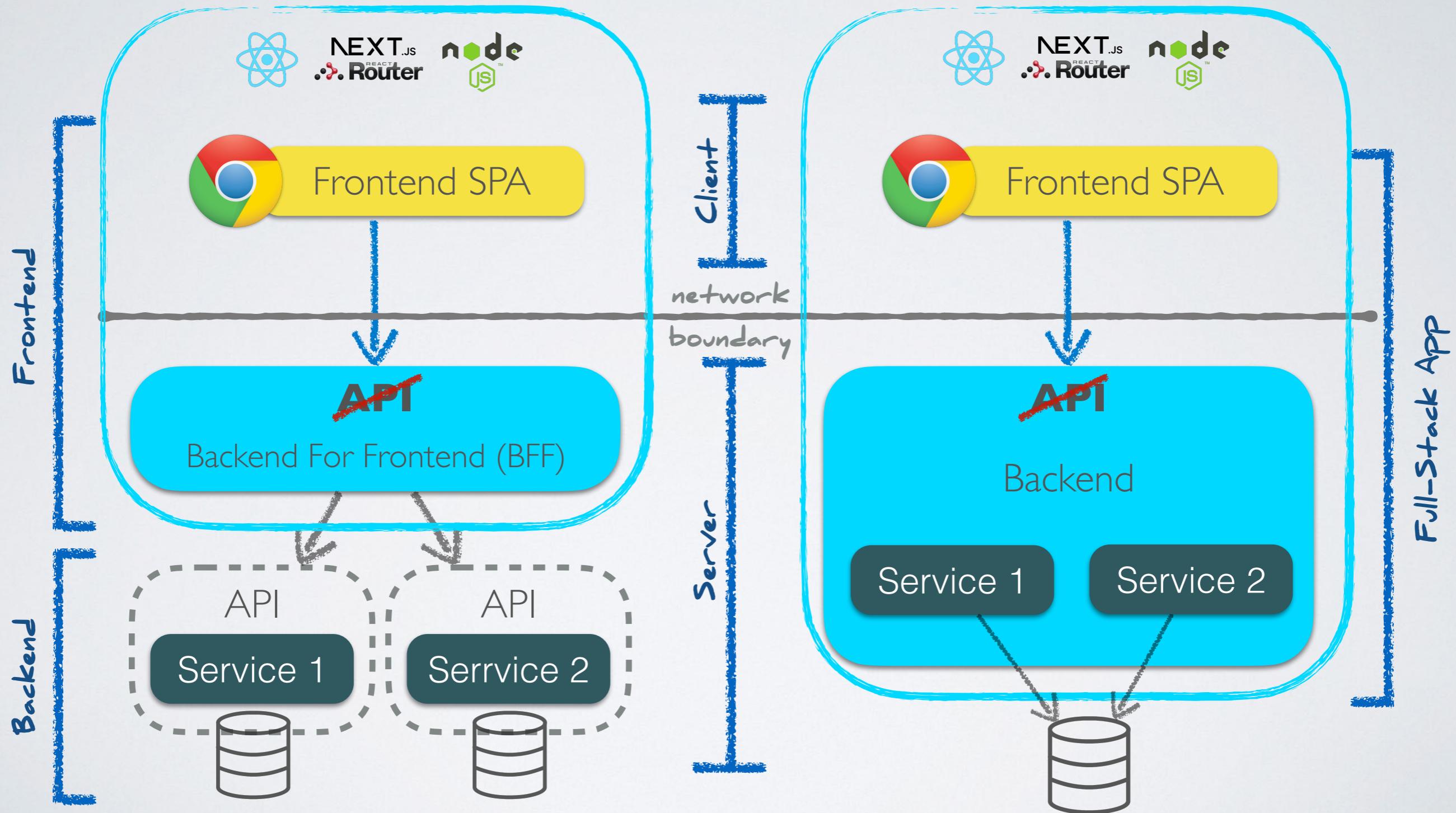
<https://react.dev/learn/creating-a-react-app#full-stack-frameworks>



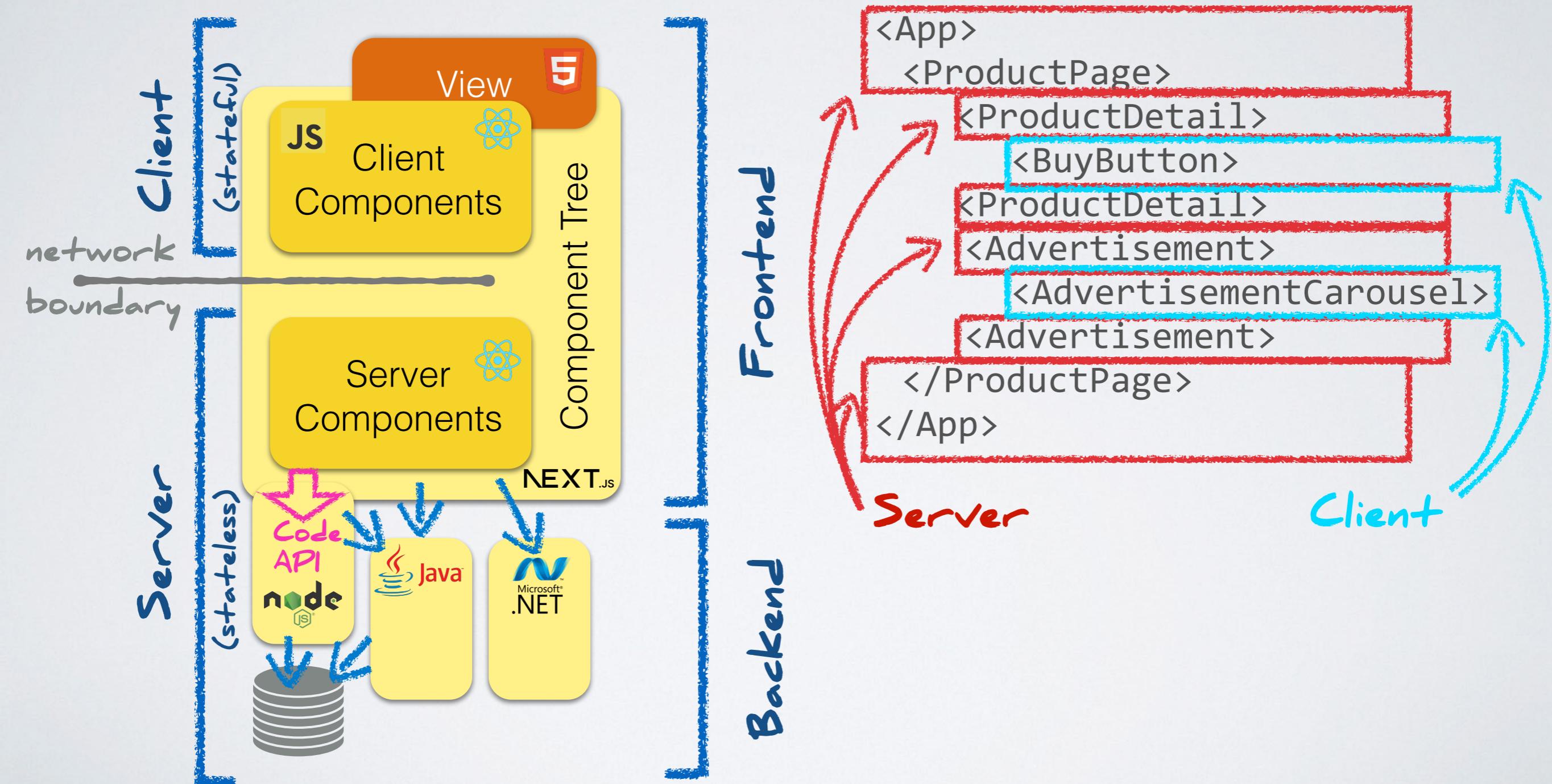
The common goal of those meta-frameworks is to simplify the project setup of frontend applications.
They include concepts for server-side-rendering, routing, data-fetching, mutations ...
But they come with their own conceptual overhead and learning curve!
These frameworks typically require running JavaScript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



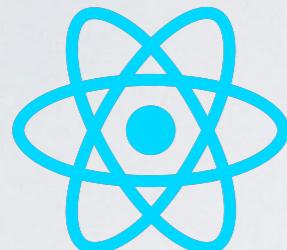
React Full-Stack Architectures



The Frontend Boundary



Vite is the default tooling for most modern frontend framework setups:



Remix



TanStack Start



Next.js uses TurboPack:



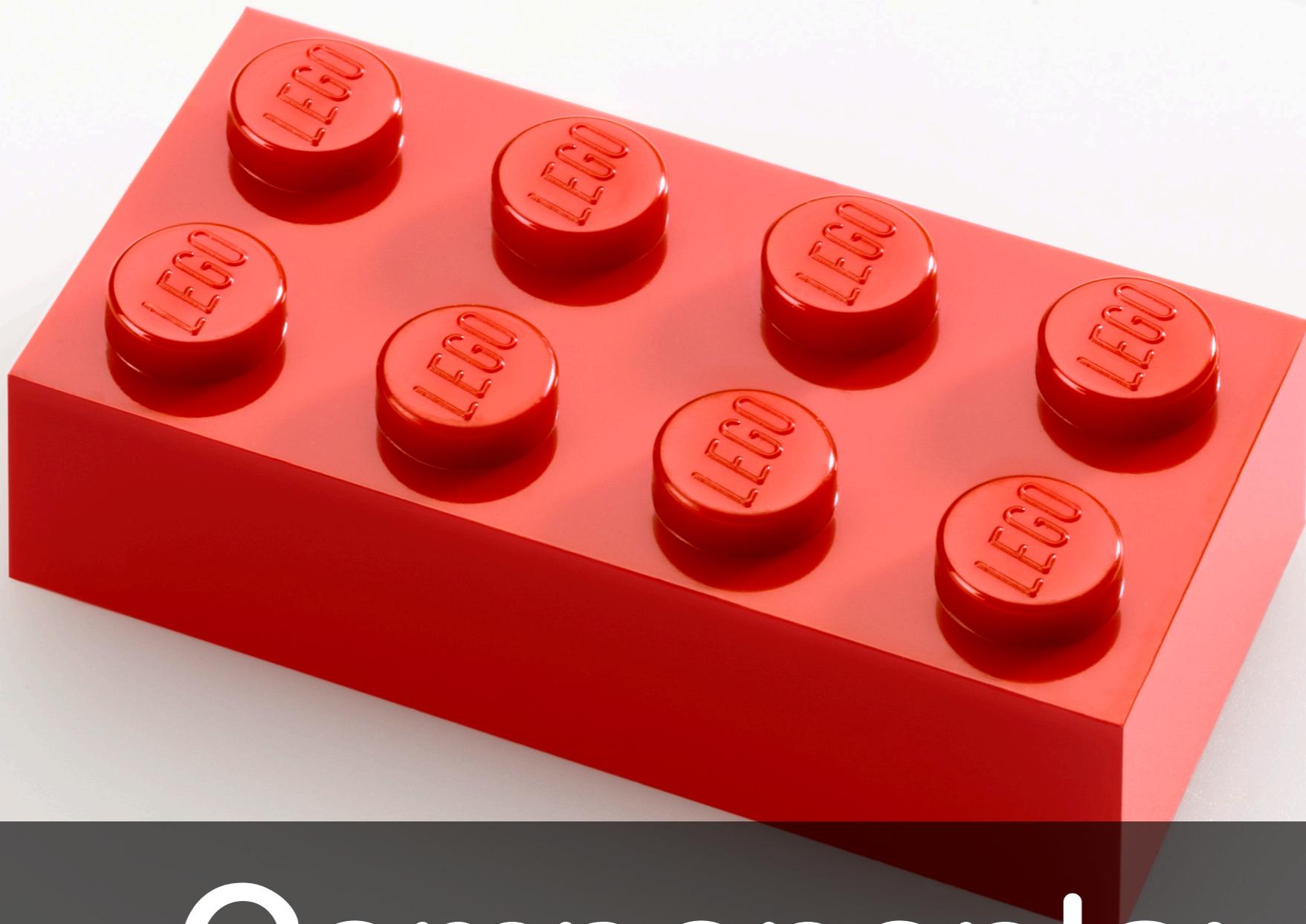
<https://turbo.build/>

<https://areweturboyet.com/>



Rspack

<https://rspack.dev/>

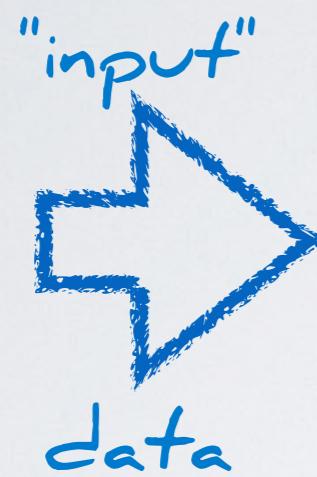


Components:

Functional Programming Model

Function Components

Components are written as plain JavaScript functions.



```
function AppComponent(props) {  
  return (  
    <div>  
      <h1>{props.title}</h1>  
      <p>{props.message}</p>  
    </div>  
  );  
}
```



The function is called each time the UI is rendered (i.e. with every "data-change")

React expects that the body of your component behaves like a *pure function*:
If the "inputs" (props, state, and context) are the same, it should return exactly the same JSX.
<https://react.dev/learn/keeping-components-pure>

Legacy: Class Components

```
import React from 'react';

export class Greeter extends React.Component {
  render() {
    return <h1>Hello World!</h1>;
  }
}
```

React 16.8 (February 2019) introduced "Hooks": a way to use state and lifecycle without writing a class.

This resulted in major changes in the React ecosystem.

React Hooks

introduced with React 16.8 in 2019.

<https://react.dev/reference/react/hooks>

Class Component:

```
class Counter1 extends React.Component {  
  
  state = {  
    count: 0  
  };  
  
  increase = () => {  
    this.setState({count: this.state.count + 1})  
};  
  
  render() {  
    return (  
      <div>  
        <h1>Count {this.state.count}</h1>  
        <button onClick={this.increase}>  
          Increase  
        </button>  
      </div>  
    );  
  }  
}
```

using ES2015 classes
using JSX
using proposed class fields

method derived from base class

Function Component with Hook:

```
function Counter2() {  
  
  const [count, setCount] = useState(0);  
  
  function increase() {  
    setCount(count + 1);  
  }  
  
  return (  
    <div>  
      <h1>Count {count}</h1>  
      <button onClick={increase}>Increase</button>  
    </div>  
  );  
}
```

using JavaScript functions
using JSX

Hook

the "new" way

the "old" way

Note : functional components and class components can be mixed in the same application.

Why Hooks (2019)?

<https://reactjs.org/docs/hooks-intro.html#motivation>

"Hooks solve a wide variety of seemingly unconnected problems of class components."

Hooks reduce complexity in components and enable better patterns for decoupling application logic and ui-logic.

Hooks enable better structure, better reuse and better composition of application logic.

Hooks are often an elegant replacement for previous reusability patterns like "higher order components" and "render props"

Hooks are not undisputed:

- <https://stevenkitterman.com/posts/the-catch-with-react-hooks/>
- <https://blog.logrocket.com/frustrations-with-react-hooks/>
- <https://dillonshook.com/a-critique-of-react-hooks/>
- <https://medium.com/swlh/the-ugly-side-of-hooks-584f0f8136b6>
- <https://medium.com/better-programming/why-react-hooks-are-the-wrong-abstraction-8a44437747c1>
- <https://danielrotterat/2022/01/16/some-reasons-for-disliking-react-hooks.html>

"Why React Hooks":

https://www.youtube.com/watch?v=eX_L39UvZes

Why Hooks (2025)?

Because the React ecosystem has embraced Hooks.

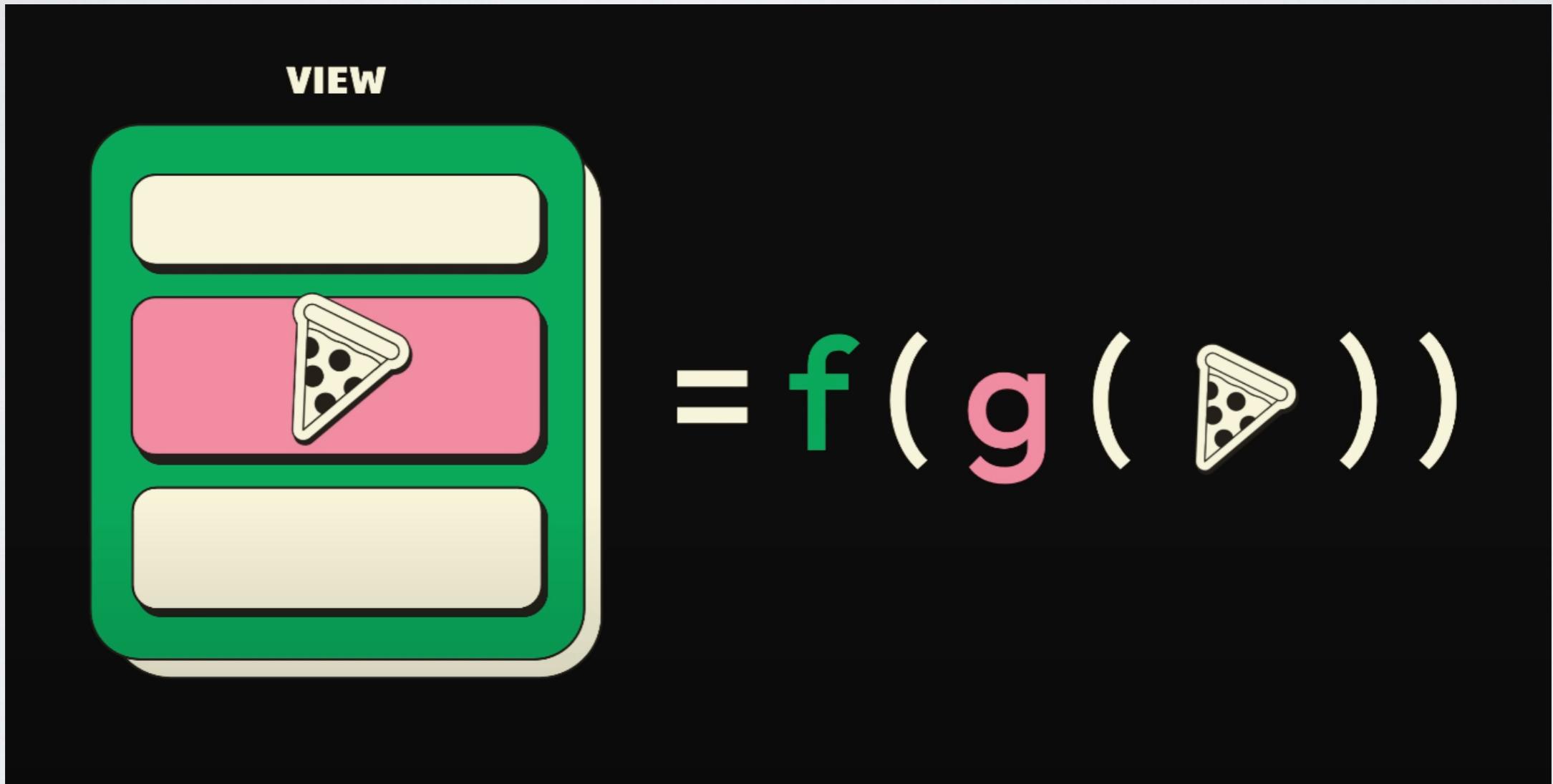
(and React without it's ecosystem is like a car without seats, wheels, windows, pedals ...)

All modern React libraries expose their APIs via Hooks:

- ReactRouter / TanStack Router
- Jotai, Zustand, LegendState, Valtio ...
- TanStackQuery, SWR
- react-i18next
- ...

UI Composition

UI Composition *is* Function Composition.



Once embraced, we can apply a whole set of patterns from functional programming.

Image from [ui.dev: The Story of react Query](https://ui.dev/The%20Story%20of%20react%20Query)
<https://www.youtube.com/watch?v=OrliU0e09io>

The power of React is a component model which enables simple & elegant composition ...



Cory House

@housecor

...

I love the simplicity of React's reuse model.

Repeating JSX? Create a component.

Repeating logic? Create a hook.

I can compose these simple building blocks in infinite ways.

1:08 PM · Nov 25, 2023 · 16.7K Views

<https://twitter.com/housecor/status/1728385239611789758>



<https://vercel.com/>

Vercel Sponsors the development of:
Next.js
React.js
Nuxt.js
Svelte.js

App Router vs. Page Router



App Router is based on React Server Components.
Other Frameworks are only slowly adopting server components:

- <https://reactrouter.com/how-to/react-server-components>
- <https://docs.rwsdk.com/core/react-server-components/>
- <https://waku.gg/>

React Server Components (RSC)

Seb ThisWeekInReact.com 
@sebastienlorber

React devs



8:52 AM · Dec 20, 2023 · 111.2K Views <https://twitter.com/sebastienlorber/status/1737380479383277636>

Adam Rackis
@AdamRackis

Server actions 🔥

```
function Bookmark({ slug }) {
  return (
    <button
      formAction={async () => {
        "use server";
        await sql`INSERT INTO Bookmarks (slug) VALUES (${slug})`;
      }}
    >
      <BookmarkIcon />
    </button>
  );
}
```

8:22 PM · Oct 26, 2023 · 4.8M Views

<https://twitter.com/AdamRackis/status/>

Original Talk: <https://www.youtube.com/watch?v=9CN9RCzznZc&t>



Are RSCs and NextJS Really That Bad?

Jack Herrington 

21K views · 2 days ago

<https://www.youtube.com/watch?v=u0OMdWlfdhg>



The growing divide among React developers... 

<https://www.youtube.com/watch?v=LJbHHeFSsE>



Evan You 
@youyuxi

I don't think RSC itself being a React feature is a problem - it unlocks some interesting patterns.

The issue is the tradeoffs involved in making it work. It leaks into, or even demands control over layers that are previously not in scope for client-side frameworks. This creates heavy complexity (and associated mental overhead) in order to get the main benefits it promises.

React team made a bet that they can work with Next team to polish the DX to the extent that the benefit would essentially be free - so that RSC can be a silver bullet for all kinds of apps, and that it would become the idiomatic way to use React. IMO, that bet has failed.

4:06 PM · Mar 27, 2025 · 175.4K Views 

<https://x.com/youyuxi/status/1905275294090662266>

React Server Components is an Architecture!

React as a *library* provides the "foundation" for React Server Components but you need a Framework to use them (deep integration into Router and Bundler).

Frameworks that support React Server Components implement the "React Architecture"
<https://react.dev/>

It's a React component ...

```
export function Greeter() {  
  
  console.log("Rendering Greeter");  
  
  return (  
    <div>  
      <h1>Display of Greeter.</h1>  
    </div>  
  );  
}
```



... but exclusively rendered on
the server!

It is still a SPA!

Your Code

generate a react tree on the client

Client Component



render instructions
running on the client

```
export function Greeter() {  
  return (  
    <h1>Hello World!</h1>  
  );  
}
```

1. render components into the
"React Server Component
Payload" on the server



2. send "RSC Payload"
to the client

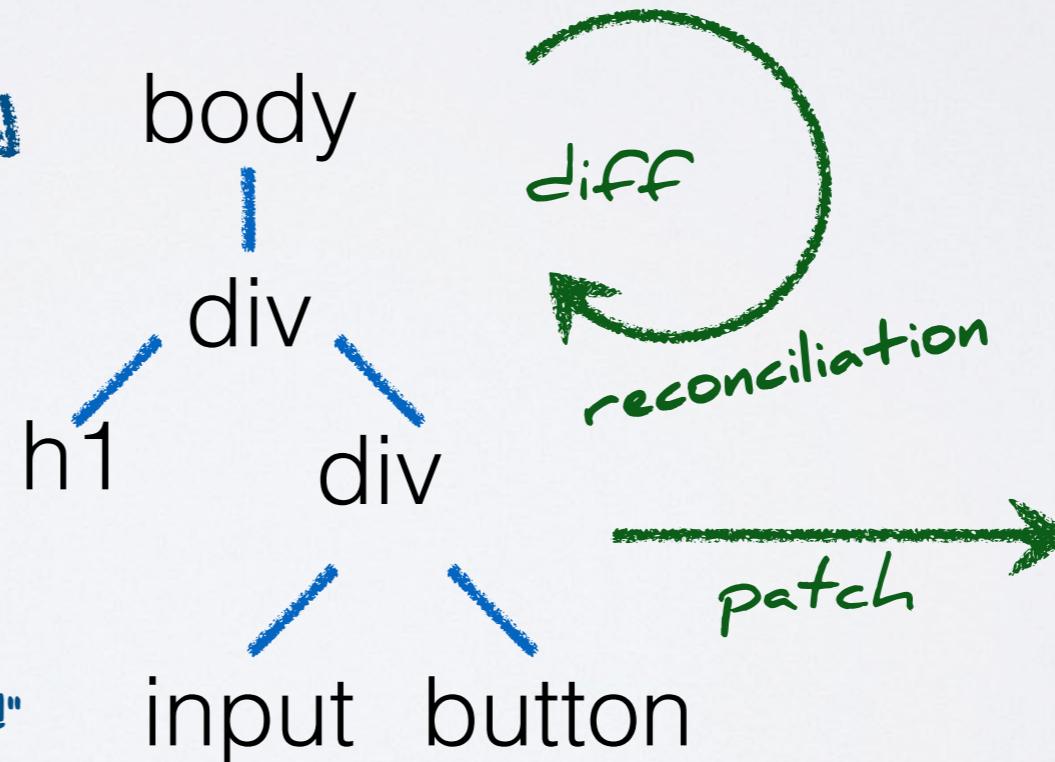
Server Component

generate a react tree on the server and send "render instructions" to the client

React Client Runtime

Virtual DOM

In-Memory, implemented in JavaScript



Browser DOM

```
<body>  
<div>  
  <h1>...</h1>  
<div>  
  <input/>  
  <button>  
    </button>  
</div>  
</div>
```



The RSC Payload

```
1:$react.fragment
2:I[6213,[],"OutletBoundary"]
4:I[6213,[],"MetadataBoundary"]
6:I[6213,[],"ViewportBoundary"]
0:{ "b": "lGRtPx18R9zG1IQbEB1F2", "f": [ [ "children", "01-simple", "children", "__PAGE__", [ "__PAGE__", {} ], [ "__PAGE__", [ "$", "$1", "c", { "children": [ [ "$", "div", null, { "className": "component_wrapper_A1SYt", "children": [ "$", "div", null, { "className": "component_server_jZH6G", "children": [ [ "$", "h1", null, { "children": "Simple Component" }, [ "$", "div", null, { "children": "11:00:40 PM" } ] ] ] } ], [ "$", "link", "0", { "rel": "stylesheet", "href": "/_next/static/css/3200fe5086d6c3a8.css", "precedence": "next", "crossOrigin": "$undefined", "nonce": "$undefined" } ], [ "$", "$L2", null, { "children": "$L3" } ] ] ], {}, null, false ], [ "$", "$1", "h", { "children": [ null, [ "$", "$1", "r2xzraK9UhNA7fwDVQYiH", { "children": [ [ "$", "$L4", null, { "children": "$L5" } ], [ "$", "$L6", null, { "children": "$L7" } ], [ "$", "meta", null, { "name": "next-size-adjust", "content": "" } ] ] ] ] ], false ] ], "S": false }
7:[ [ "$", "meta", "0", { "name": "viewport", "content": "width=device-width, initial-scale=1" } ] ]
5:[ [ "$", "meta", "0", { "charSet": "utf-8" } ], [ "$", "title", "1", { "children": "Create Next App" } ], [ "$", "meta", "2", { "name": "description", "content": "Generated by create next app" } ], [ "$", "link", "3", { "rel": "icon", "href": "/favicon.ico", "type": "image/x-icon", "sizes": "16x16" } ] ]
3:null
```

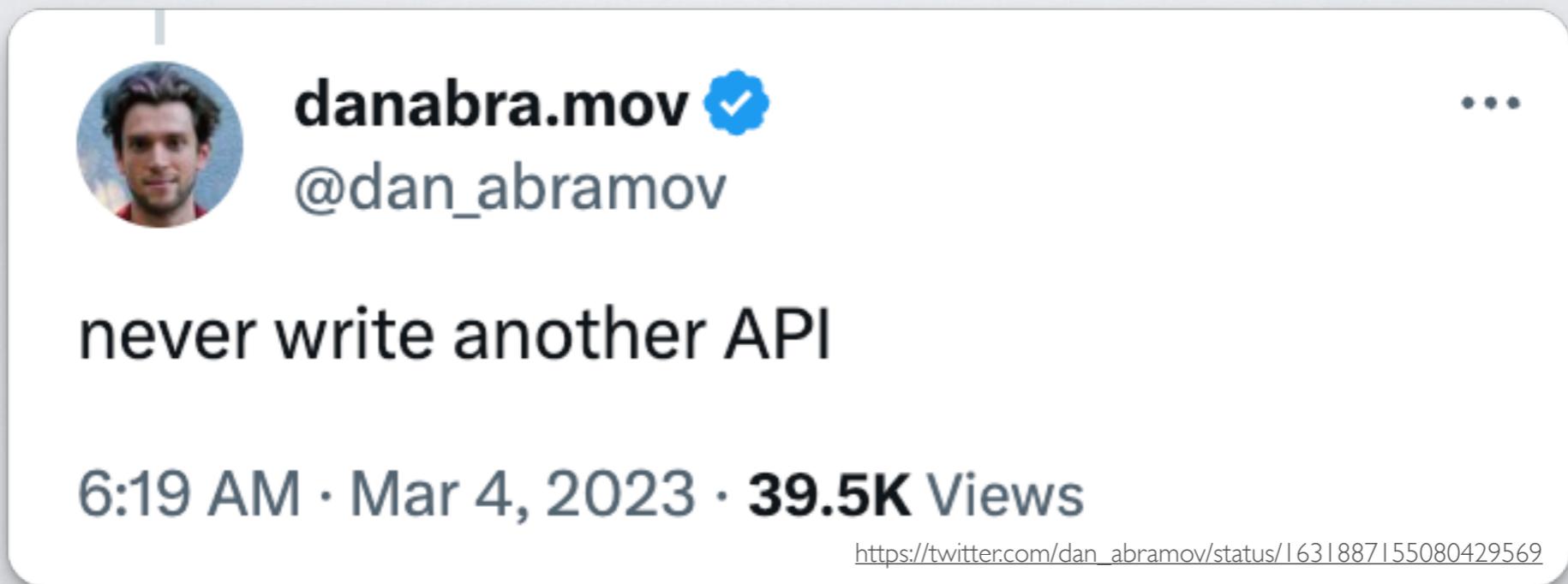
Load data on the server!

```
export async function Greeter() {  
  const dataFromDb = await queryDataFromDb();  
  
  return (  
    <div>  
      <h1>{dataFromDb}</h1>  
    </div>  
  );  
}
```



Asynchronous rendering!
Making data fetching easy!

SPA data fetching without HTTP-API!



A screenshot of a Twitter post from user **danabra.mov** (@dan_abramov). The post contains the text: "never write another API". It was posted at 6:19 AM · Mar 4, 2023, and has 39.5K views. The URL of the tweet is https://twitter.com/dan_abramov/status/1631887155080429569.

Wouldn't that be tempting?

Out of Order Streaming

```
<h3>Server Data:</h3>
<Suspense fallback={<Spinner />}>
  <Backend messageId={1} />
</Suspense>
<Suspense fallback={<Spinner />}>
  <Backend messageId={2} />
</Suspense>
<Suspense fallback={<Spinner />}>
  <Backend messageId={3} />
</Suspense>
```



All | Fetch/XHR | Doc | CSS | JS | Font | Img | Media | Manifest | WS | Wasm | Other | Blocked response cookies

Blocked requests 3rd-party requests

Name	Method	Status	Type	Initiator	Size	Time	Waterfall
03-streaming?v=31	GET	200	document	Other	4.1 kB	4.07 s	

```
<div hidden id="S:0">
  <div>
    <h1>Hello from DB!</h1>
    <p>10:14:32 PM</p>
  </div>
  <script>
    $RC("B:0", "S:0")
  </script>
```

```
<div hidden id="S:1">
  <div>
    <h1>Hello World!</h1>
    <p>10:14:32 PM</p>
  </div>
  <script>
    $RC("B:1", "S:1")
  </script>
```

<https://www.youtube.com/watch?v=23bHSDJD9y4>

React Client Components

... are rendered on the client and also initially on the server.

```
"use client"
export function Clock() {
  const [time, setTime] = useState(new Date())

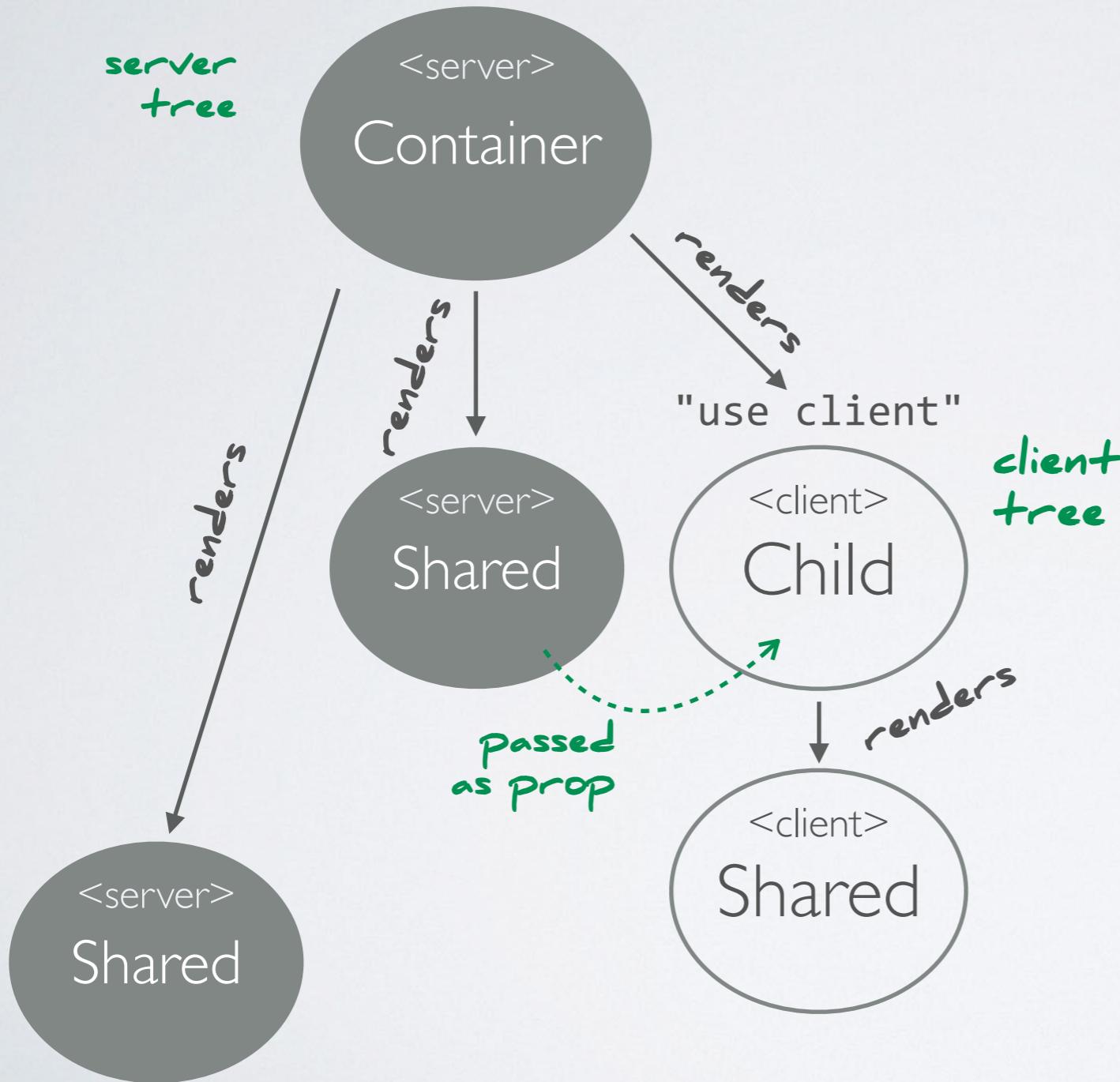
  useEffect(() => {
    setInterval(() => setTime(new Date()), 1000);
  }, []);

  return (
    <div>
      <h1>{time.toLocaleTimeString()}</h1>
    </div>
  );
}
```

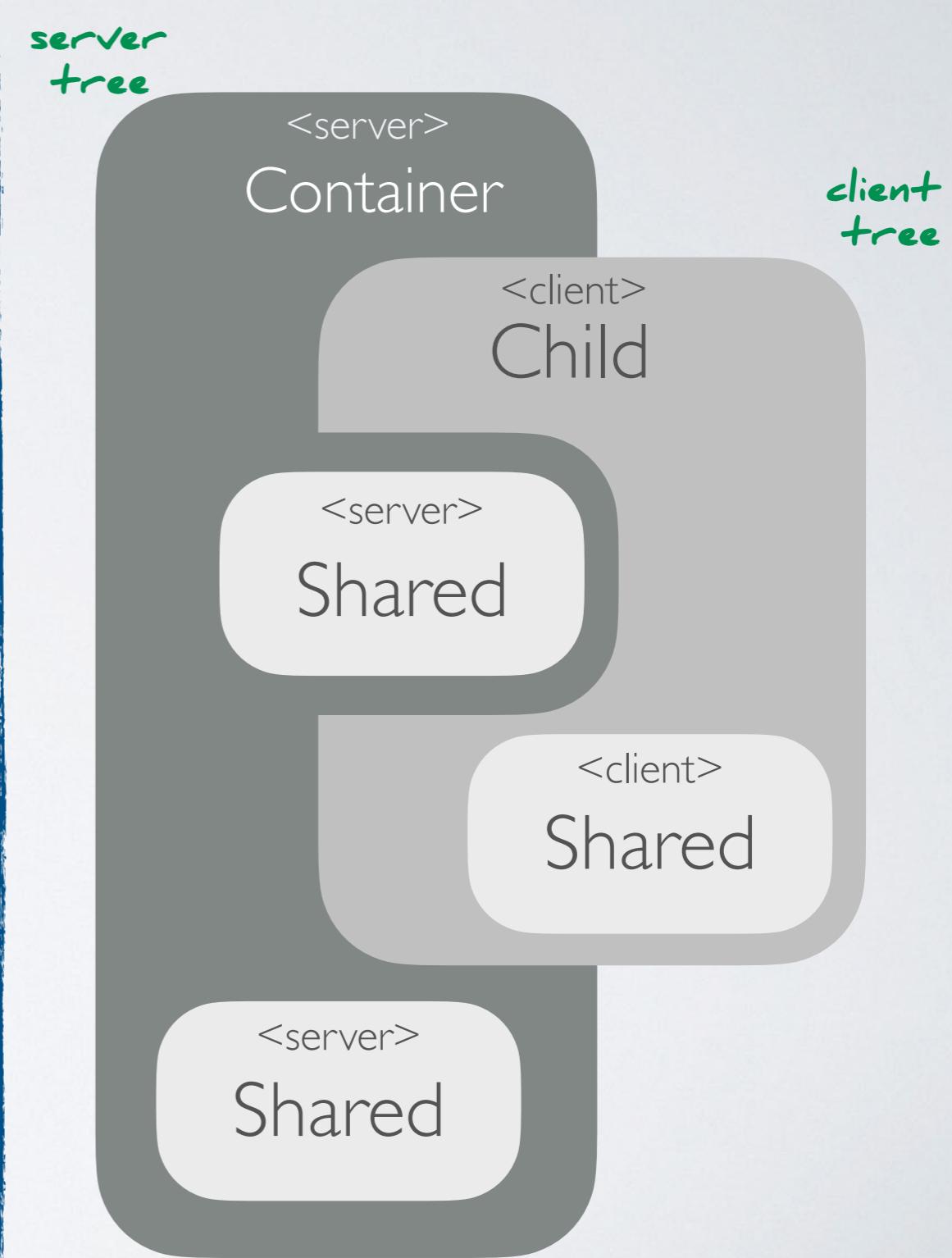
Client Components are "opt in".
Per default a component is a Server Component.

Composition

"logical perspective"

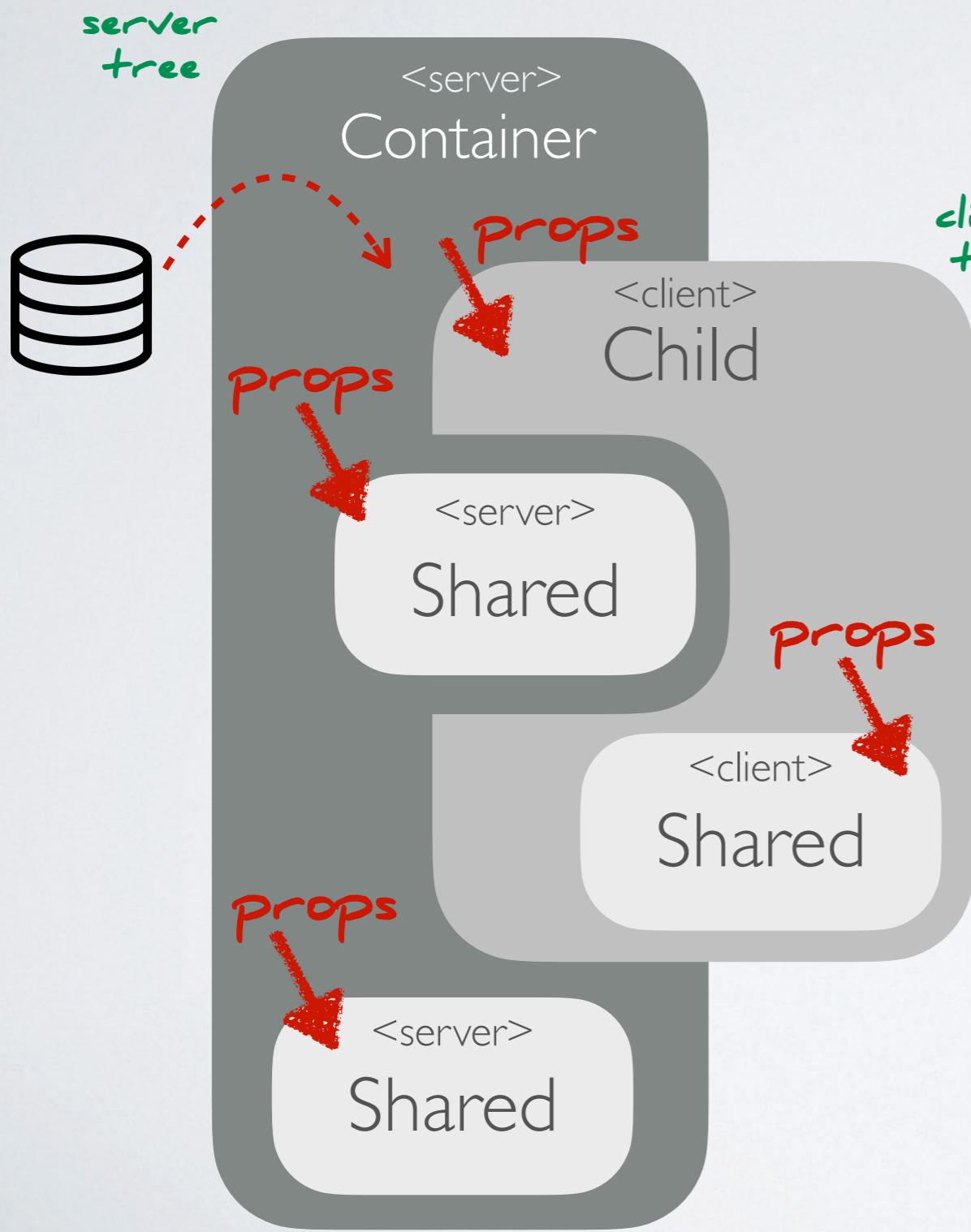


"composition perspective"

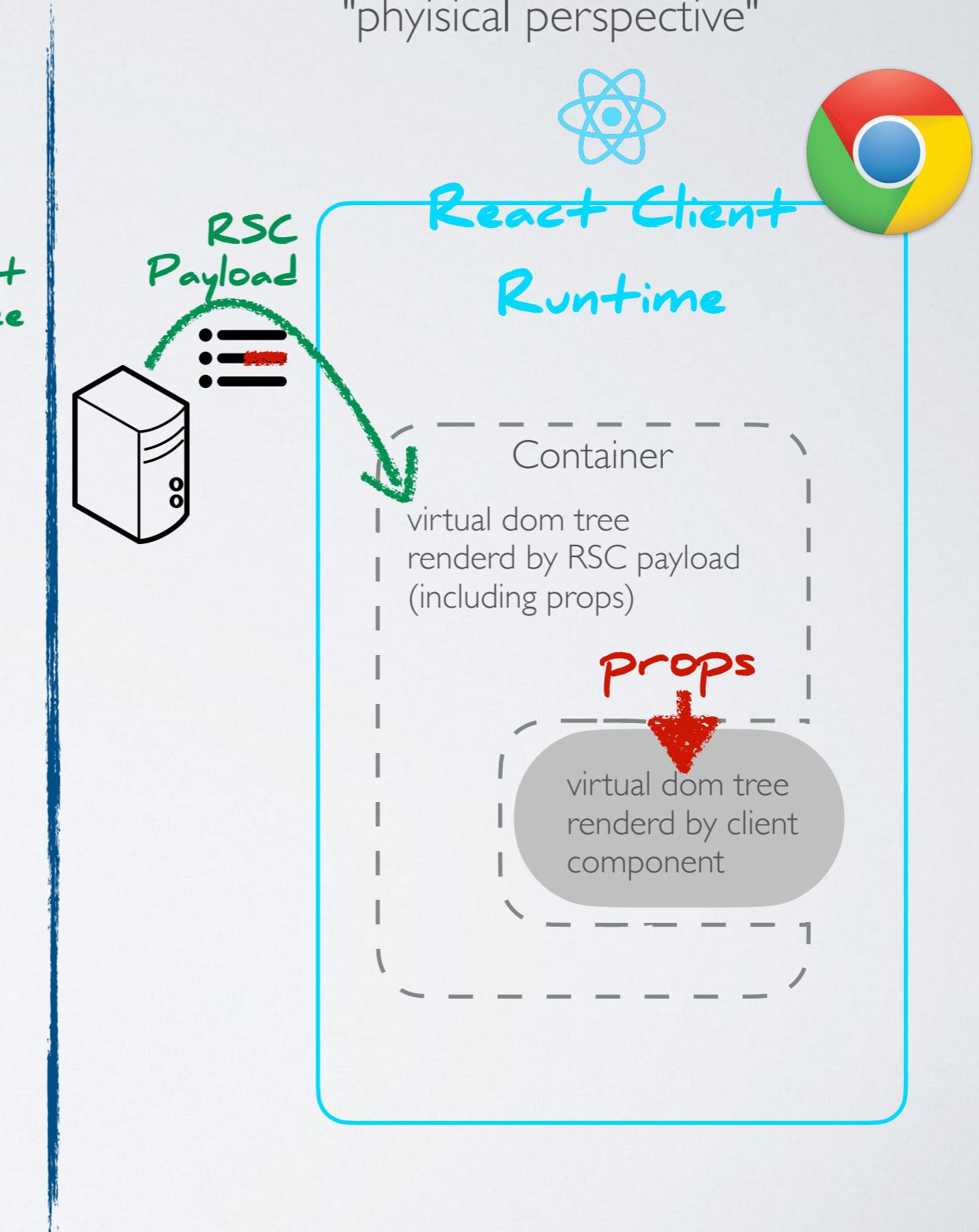


Full-Stack Data Flow

"composition perspective"



"physical perspective"



A component tree that spans
between client and server!

danabra.mov ✅
@dan_abramov

never write another API

6:19 AM · Mar 4, 2023 · 39.5K Views

https://twitter.com/dan_abramov/status/163188715500



In case you did not believe it
the first time ...



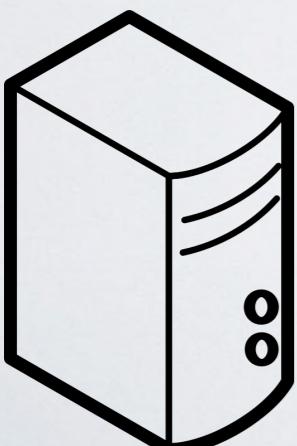
Server Functions



```
function ServerComponent(){  
  return (  
    <form action={serverActionRpc}>  
      <button>Submit</button>  
    </form>  
  )  
}
```

Server Function:

```
"use server";  
export async function serverActionRpc(arg) {  
  await updateDb(arg);  
  revalidatePath("/");  
}
```



Server

JavaScript
bundle loaded by
the browser

Network

JS



RSC Payload sent
to the browser

first scenario: server component
calling server function

 React Client
Runtime

(virtual dom tree)

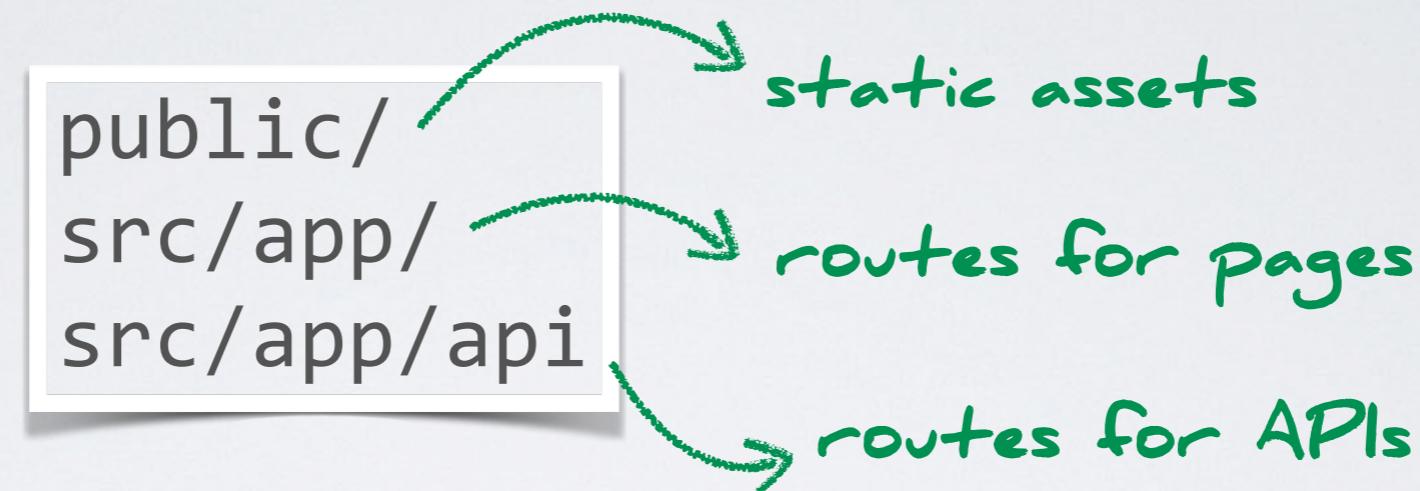
second scenario: client component
calling server function

```
"use client"  
function ClientComponent(){  
  return (  
    <button onClick={serverActionRpc}>  
      Update  
    </button>  
  )  
}
```

NEXT.JS

```
npx create-next-app@latest
```

Project Structure



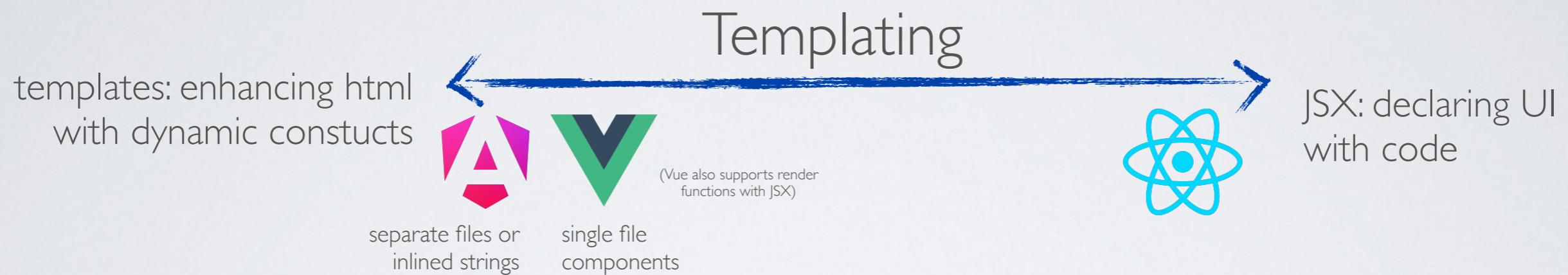
Next.js has a *file-based router*: directories and files in the `/app` define the routes based on convention.

Examples:

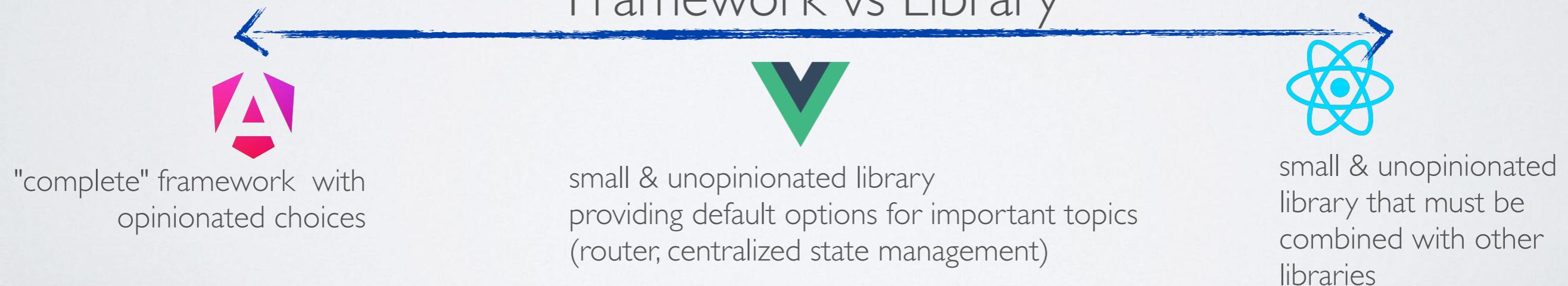
```
src/app/about/page.tsx  
src/app/invoice/[id].tsx  
src/app/api/status/route.ts
```

<https://nextjs.org/docs/app/getting-started/layouts-and-pages>
<https://nextjs.org/docs/app/api-reference/file-conventions>

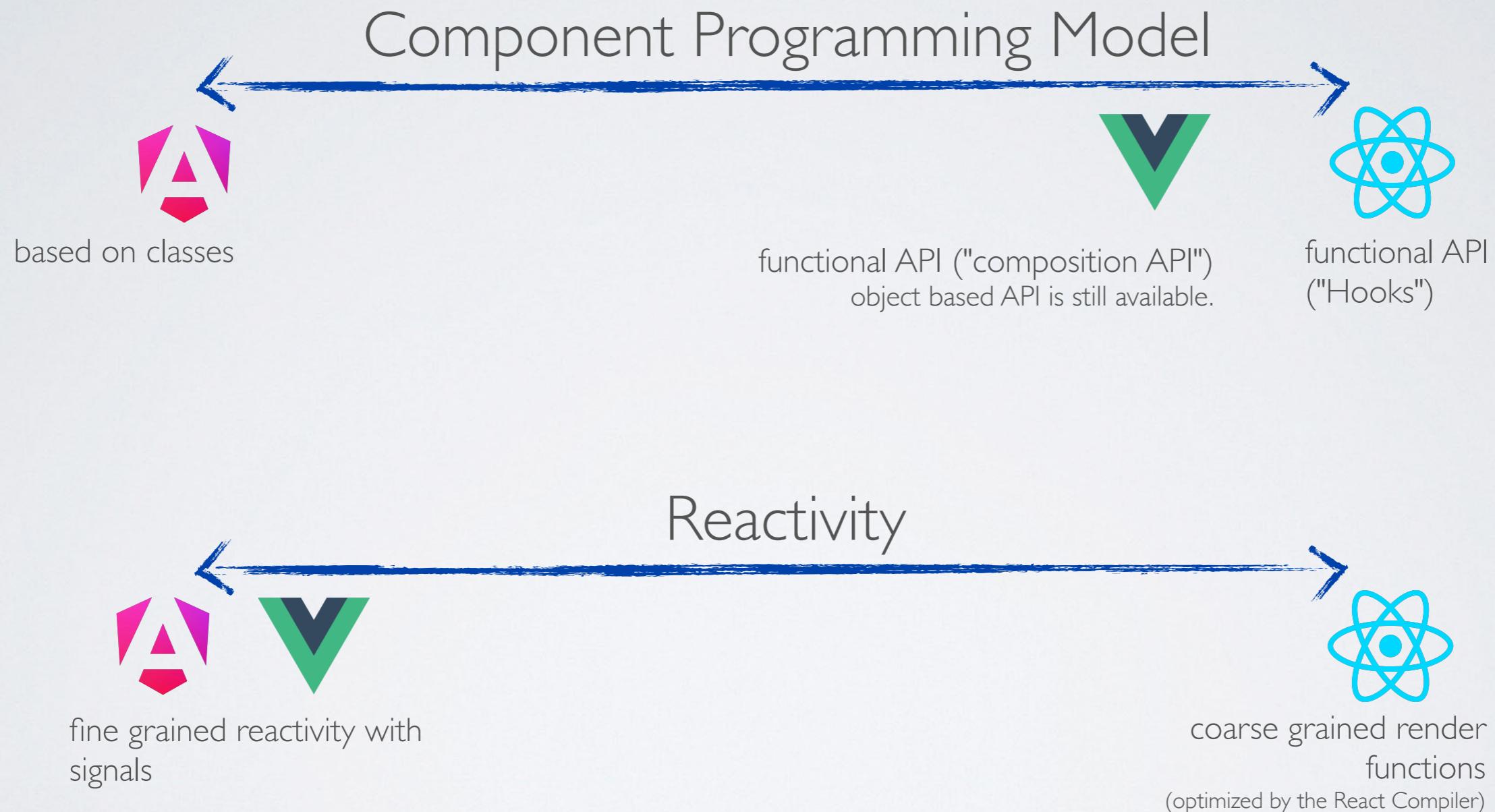
Key Differences



Framework vs Library

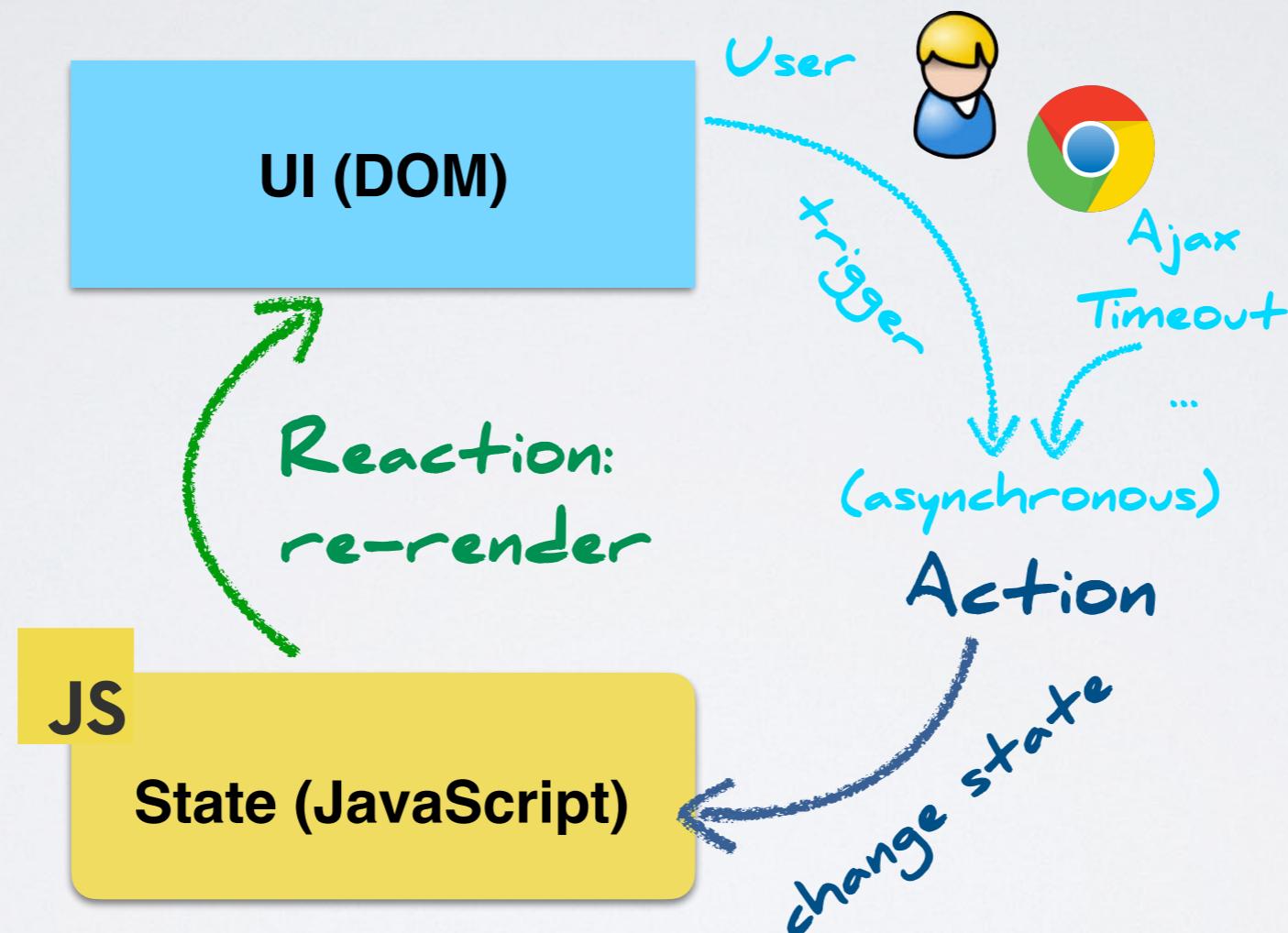


Key Differences



State is Managed in JavaScript

The UI renders the state and "emits" events.



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

Performance Comparison

Name Duration for...	vue-vapor-v3.6.0-alpha.2	 svelte-v5.13.0	 solid-v1.9.3	 vue-v3.6.0-alpha.2	 angular-cf-v20.0.1	 angular-cf-signals-v20.0.1	 react-hooks-v19.0.0	 angular-ngfor-v20.0.1	 react-compiler-hooks-v19.0.0	 react-hooks-use-transition-v19.0.0
Implementation notes										
Implementation link	code	code	code	code	code	code	code	code	code	code
create rows creating 1,000 rows. (5 warmup runs).	24.0 ± 0.1 (1.06)	23.6 ± 0.1 (1.04)	23.8 ± 0.1 (1.05)	26.9 ± 0.1 (1.19)	32.3 ± 0.2 (1.43)	32.1 ± 0.2 (1.42)	28.0 ± 0.4 (1.24)	32.1 ± 0.2 (1.42)	27.7 ± 0.4 (1.23)	35.2 ± 0.7 (1.56)
replace all rows updating all 1,000 rows. (5 warmup runs).	27.5 ± 0.2 (1.08)	27.8 ± 0.2 (1.09)	27.6 ± 0.2 (1.08)	30.9 ± 0.1 (1.21)	37.7 ± 0.3 (1.48)	37.3 ± 0.2 (1.46)	33.6 ± 0.3 (1.32)	37.0 ± 0.1 (1.45)	33.7 ± 0.3 (1.32)	36.5 ± 1.2 (1.43)
partial update updating every 10th row for 1,000 row. (3 warmup runs). 4 x CPU slowdown.	11.1 ± 0.3 (1.12)	10.8 ± 0.4 (1.09)	10.8 ± 0.4 (1.09)	12.9 ± 0.3 (1.30)	12.0 ± 0.4 (1.21)	12.3 ± 0.2 (1.24)	14.7 ± 0.2 (1.48)	11.5 ± 0.4 (1.16)	14.7 ± 0.5 (1.48)	34.5 ± 3.9 (3.48)
select row highlighting a selected row. (5 warmup runs). 4 x CPU slowdown.	2.4 ± 0.3 (1.09)	3.3 ± 0.2 (1.50)	2.6 ± 0.1 (1.18)	3.2 ± 0.2 (1.45)	3.6 ± 0.2 (1.64)	3.8 ± 0.2 (1.73)	4.3 ± 0.2 (1.95)	3.7 ± 0.1 (1.68)	4.5 ± 0.1 (2.05)	11.3 ± 0.9 (5.14)
swap rows swap 2 rows for table with 1,000 rows. (5 warmup runs). 4 x CPU slowdown.	13.7 ± 0.4 (1.10)	13.8 ± 0.3 (1.11)	14.1 ± 0.3 (1.14)	14.5 ± 0.2 (1.17)	15.1 ± 0.3 (1.22)	15.7 ± 0.3 (1.27)	104.5 ± 0.5 (8.43)	123.4 ± 1.0 (9.95)	104.4 ± 0.7 (8.42)	122.1 ± 3.6 (9.85)
remove row removing one row. (5 warmup runs). 2 x CPU slowdown.	10.4 ± 0.4 (1.07)	10.5 ± 0.1 (1.08)	10.8 ± 0.1 (1.11)	12.7 ± 0.1 (1.31)	11.5 ± 0.1 (1.19)	13.5 ± 0.2 (1.39)	11.8 ± 0.1 (1.22)	11.7 ± 0.1 (1.21)	11.8 ± 0.2 (1.22)	14.5 ± 0.4 (1.49)

<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.

Popularity: Angular vs. React vs. Vue

