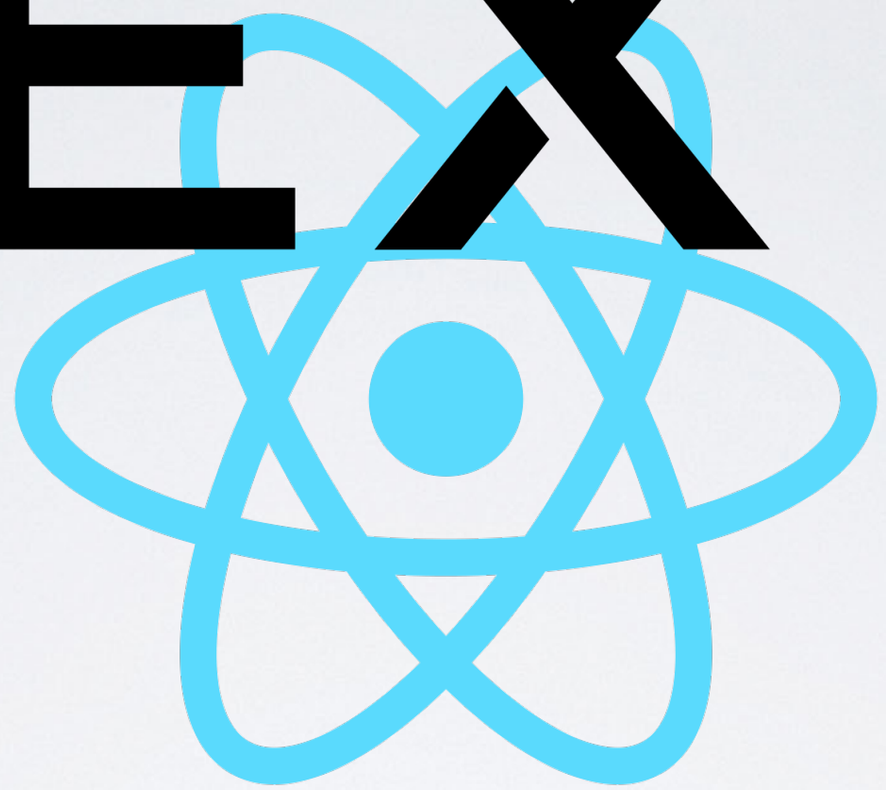
NEXT.JS



Modern React Development with Next.js and Server Components

ABOUT ME

Jonas Bandi
jonas.bandi@ivorycode.com
Twitter: @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 ...

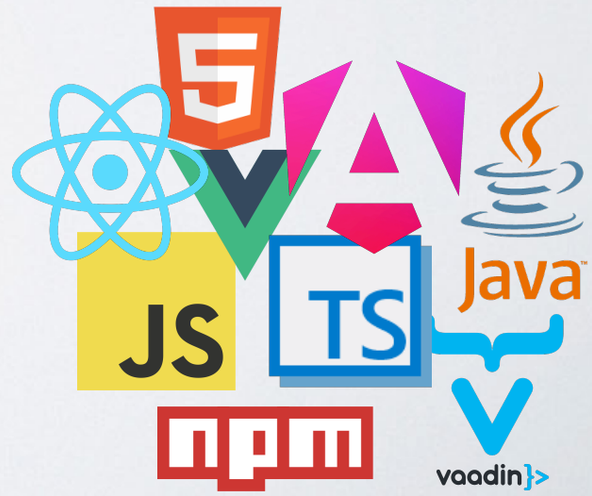
Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

letsboot.ch
swiss dev training

digicomp

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

- 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

2013 — React (Virtual Dom)

2015 — React Native components are based on ES2015 classes

2016 — splitting React and ReactDOM
React 0.14
Create React App
NEXT.JS
React 15

2019 — React 16.8: Hooks ("Function Components for everything")

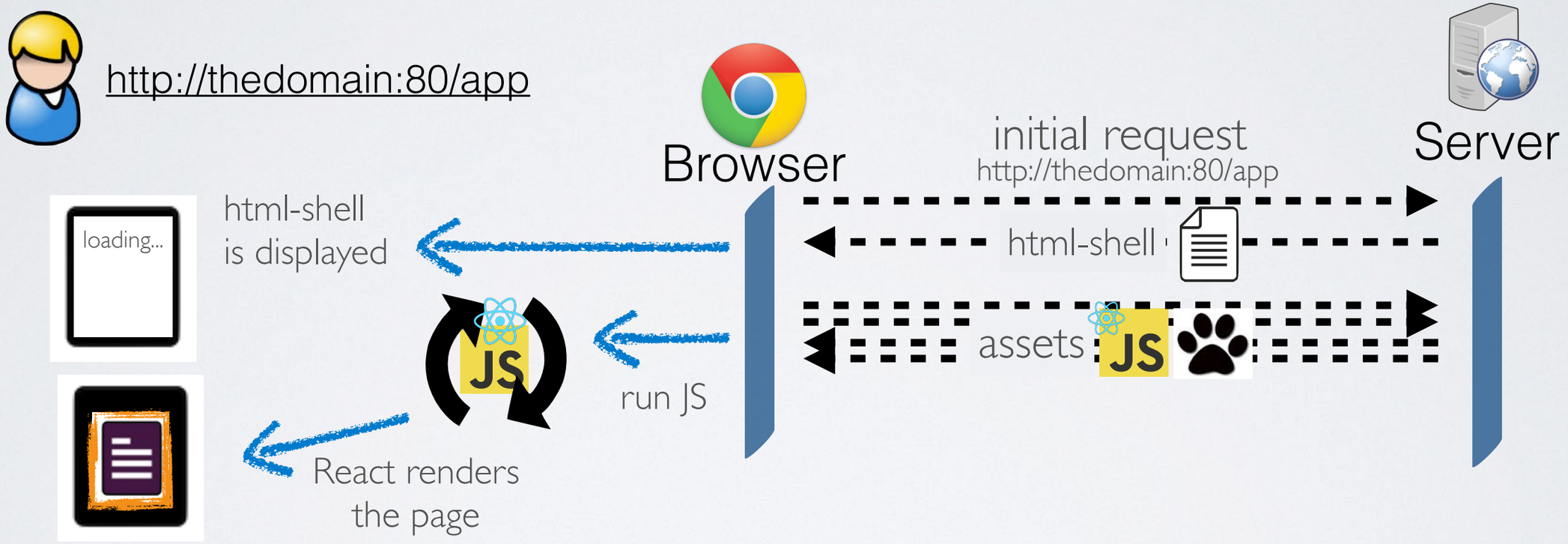
2022 — Vite 2.0
React 18
Next.js App Router with RSCs

2024 — React 19
"official" release of RSCs

2025 — React Compiler

Everything is a Component

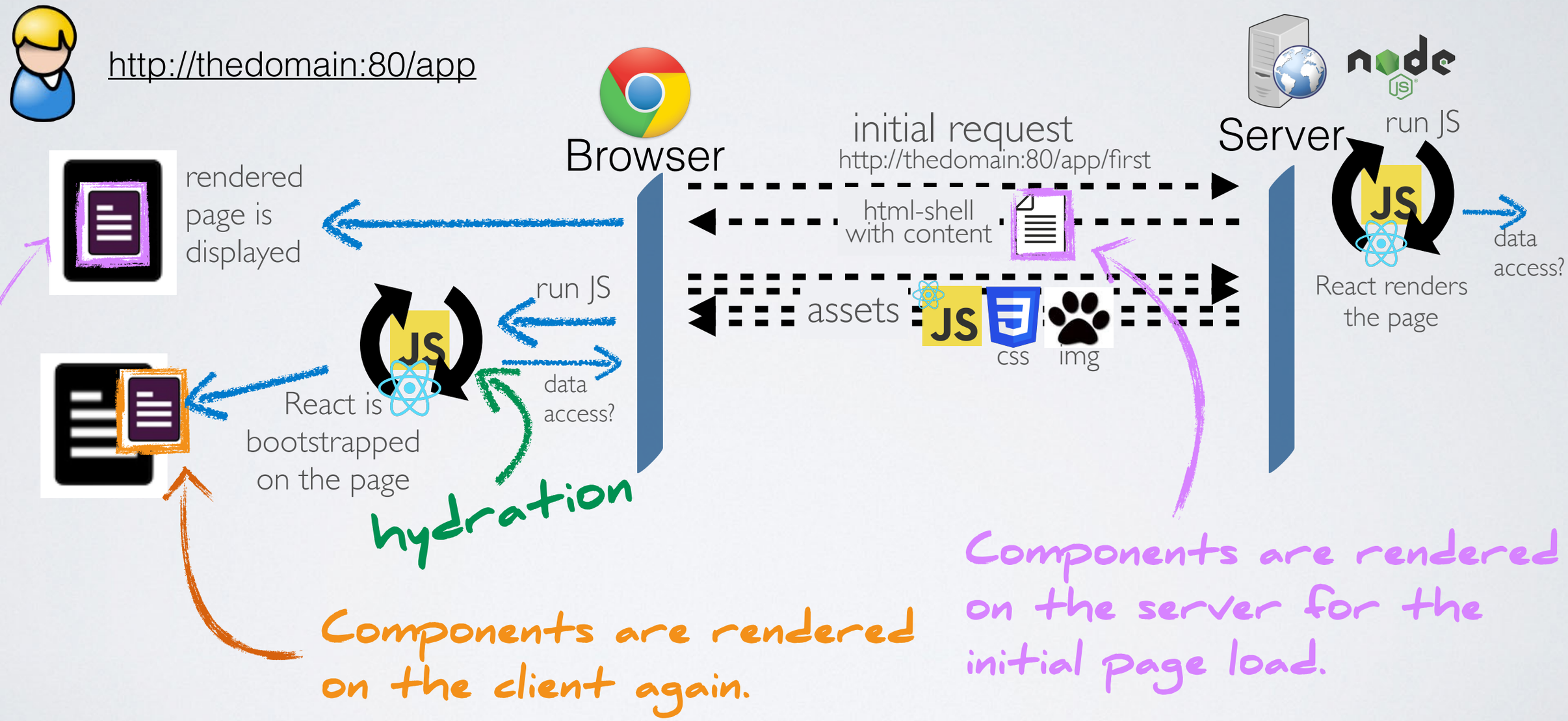
Traditional SPA:
Client Side Rendering



Components are rendered on the client.

SPA with Server Side Rendering (SSR)

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



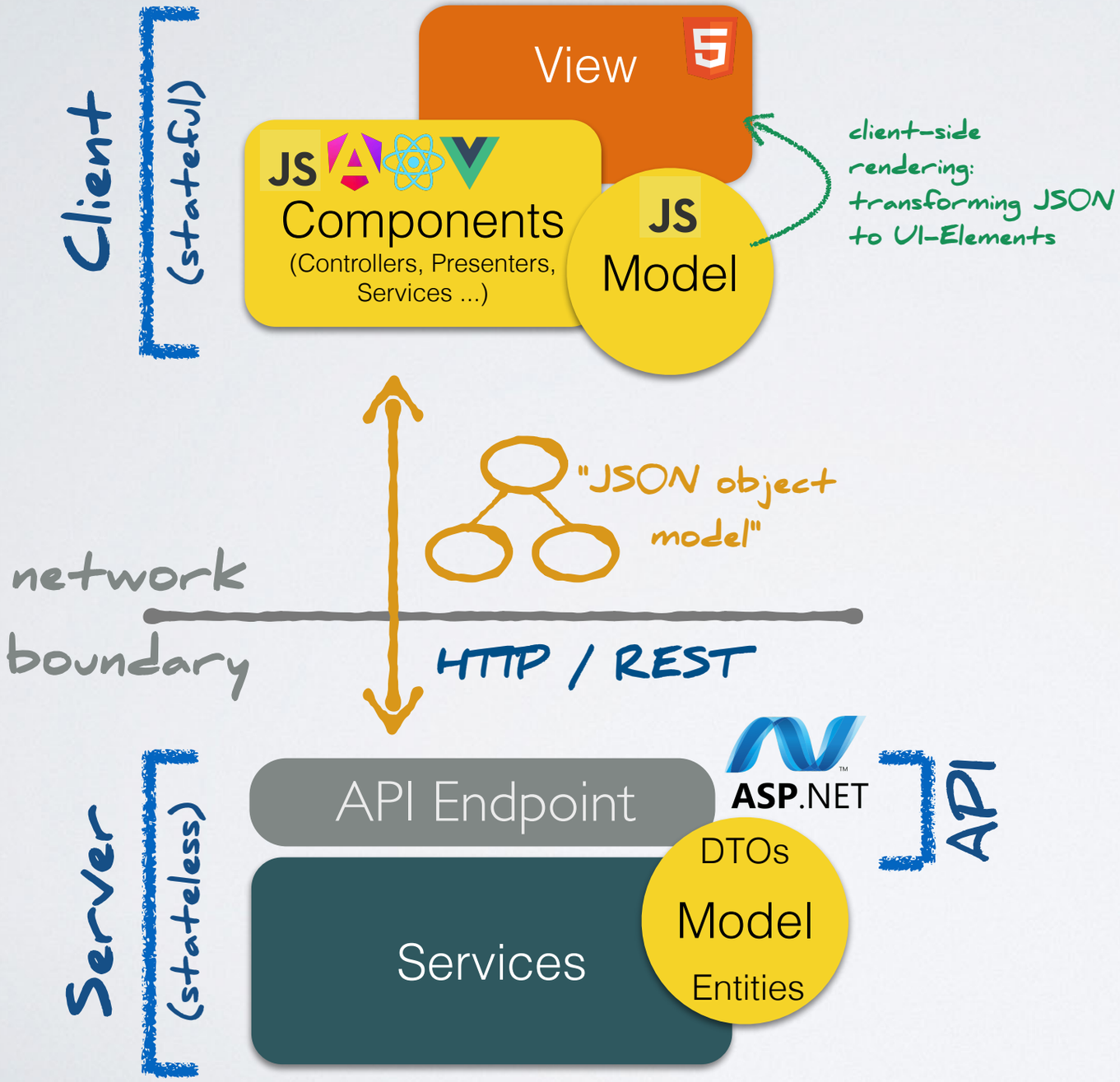
Advantages:

- 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

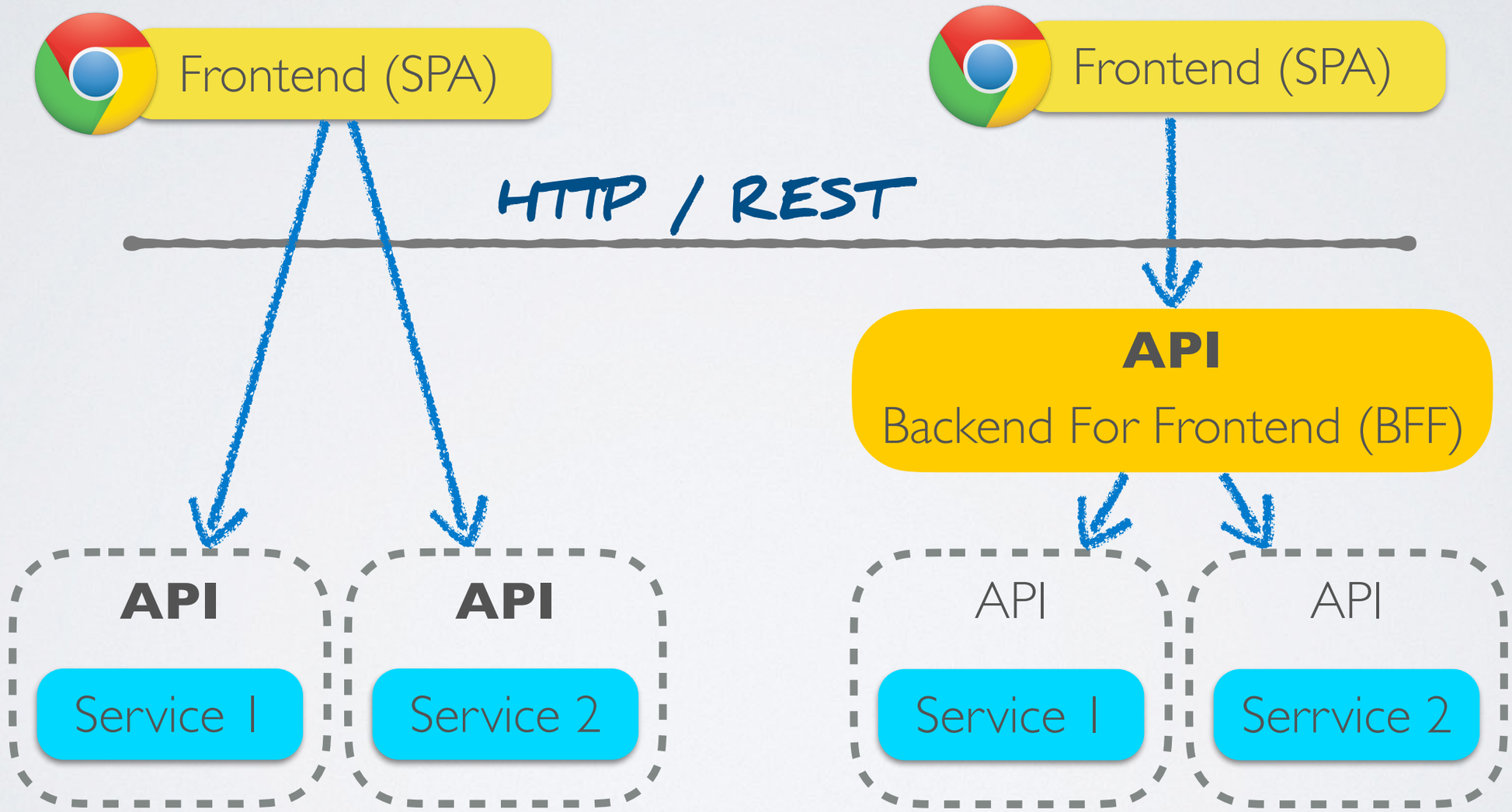


Rich client programming model in the browser.

Clear separation of concerns between client and server.

Traditional Architectures for SPAs

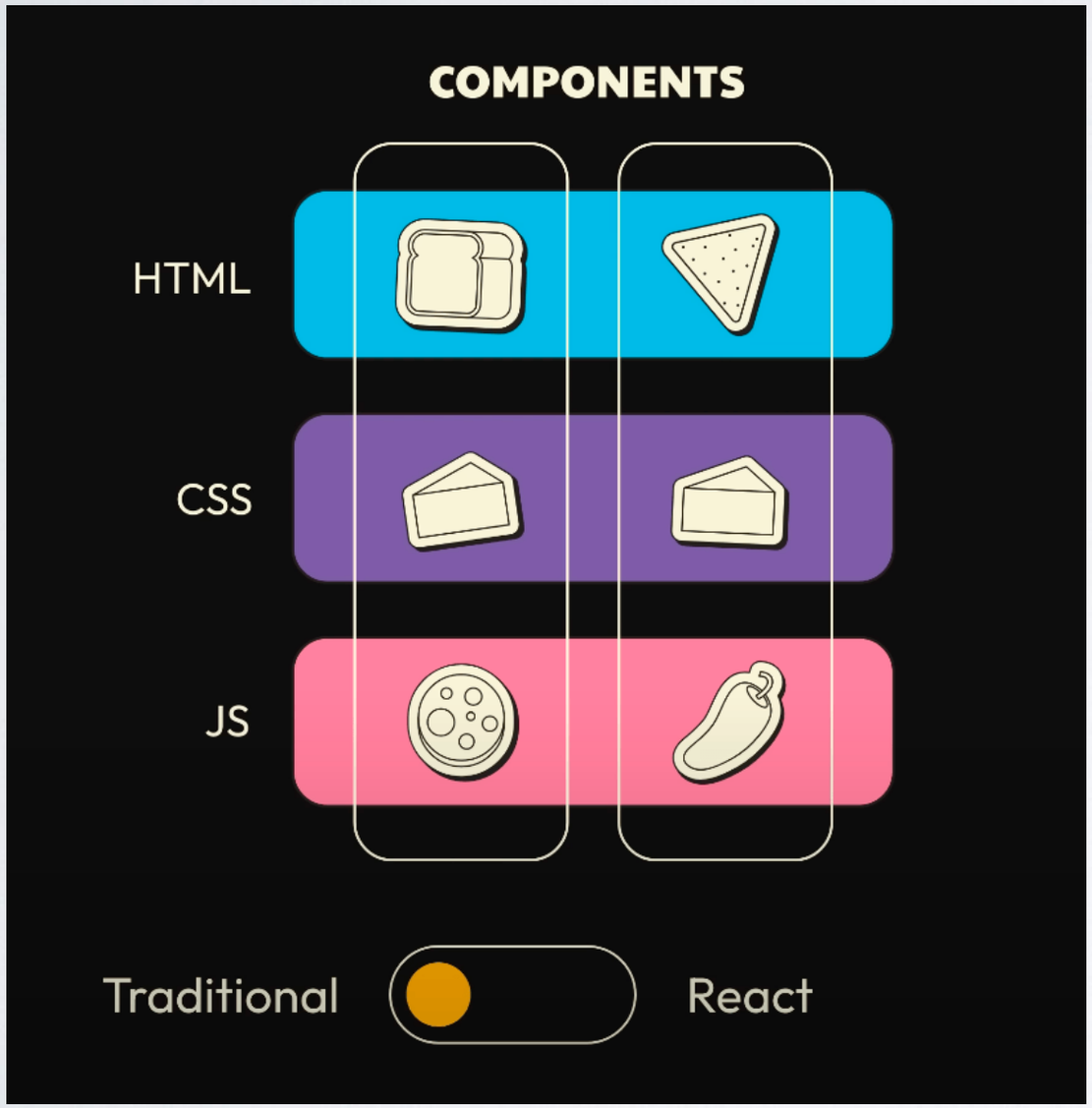
Client - API - Server



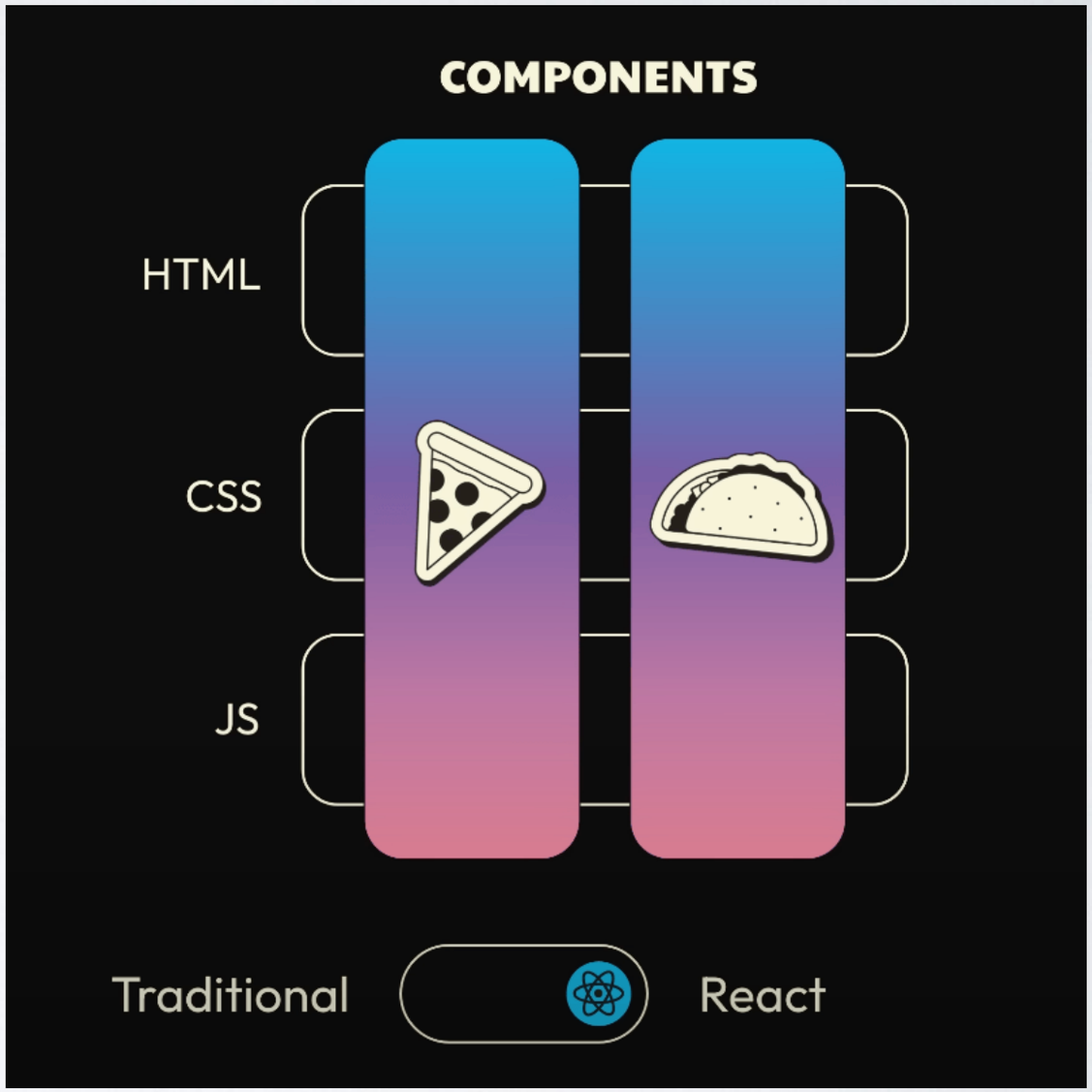
https://samnewman.io/patterns/architectural/bff/

Separation of Concerns

Traditional Seperation of Concerns in Web Dev:



Seperation 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

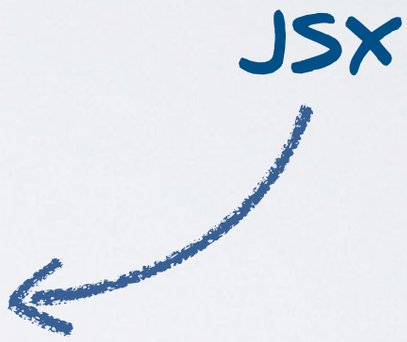
540 Retweets **1,817** Likes

40 540 1.8K

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

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

```jsx
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 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 ideomatic 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

Components

Virtual DOM

In-Memory, implemented in JavaScript

Browser DOM

App

Title

Content

input

button

render

if the state changes components are rendered completely including all their children

body

div

h1 div

input button

diff

reconciliation

patch

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

"React Client Rendering Runtime"

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

In 2013 the virtual DOM was the reason why React was faster than other Frameworks.

Today the Virtual DOM is the reason why React is slower than other modern frameworks.
https://svelte.dev/blog/virtual-dom-is-pure-overhead

React Data-Access

```
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;
}
```

effect functions can't be async

no loading state, no error state, no caching

ignoring stale responses

cleanup function

effect dependencies

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

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

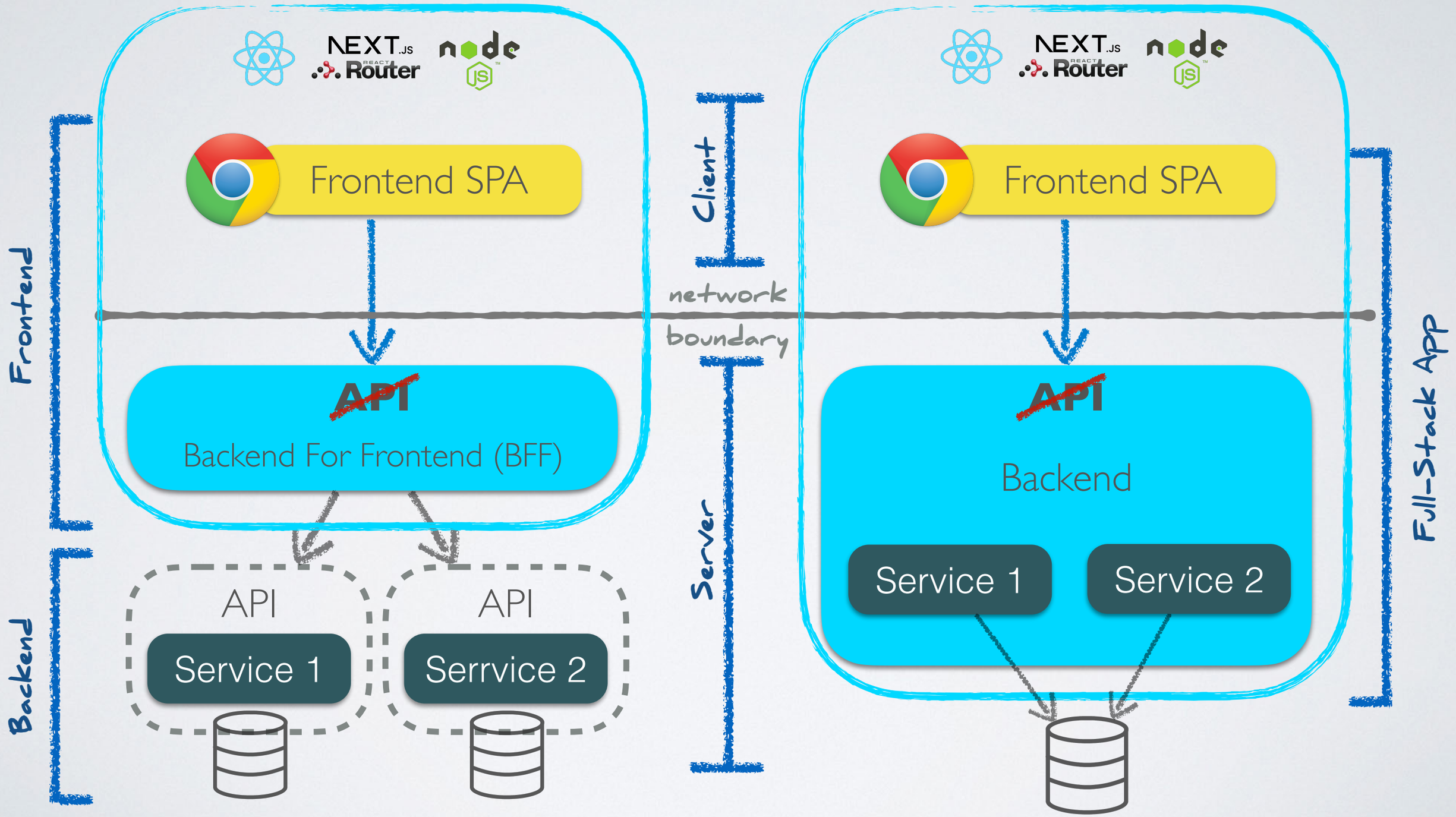
https://modernjs.dev/

Waku

https://waku.gg/

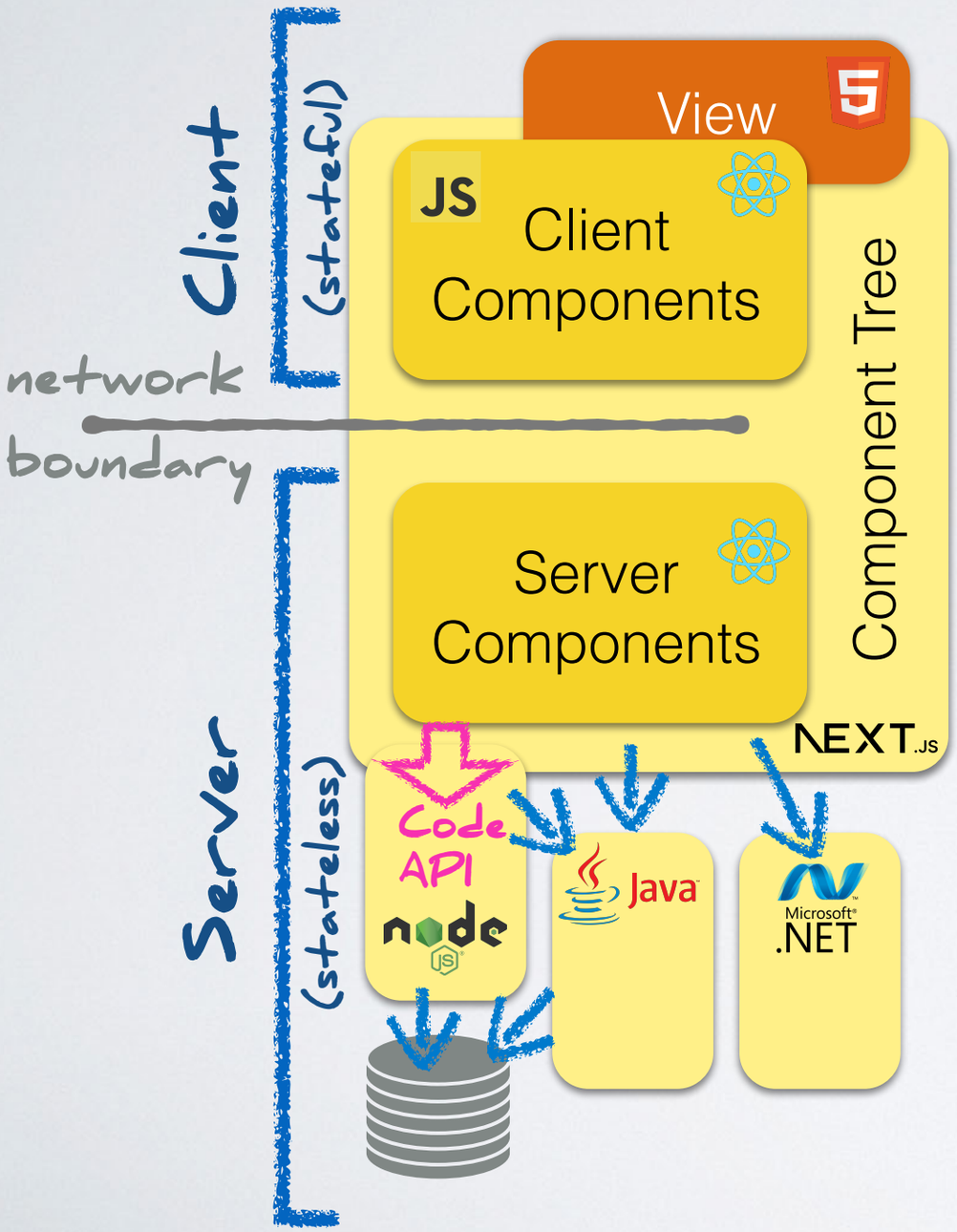
Remix was merged into React Router v7: https://remix.run/blog/react-router-v7

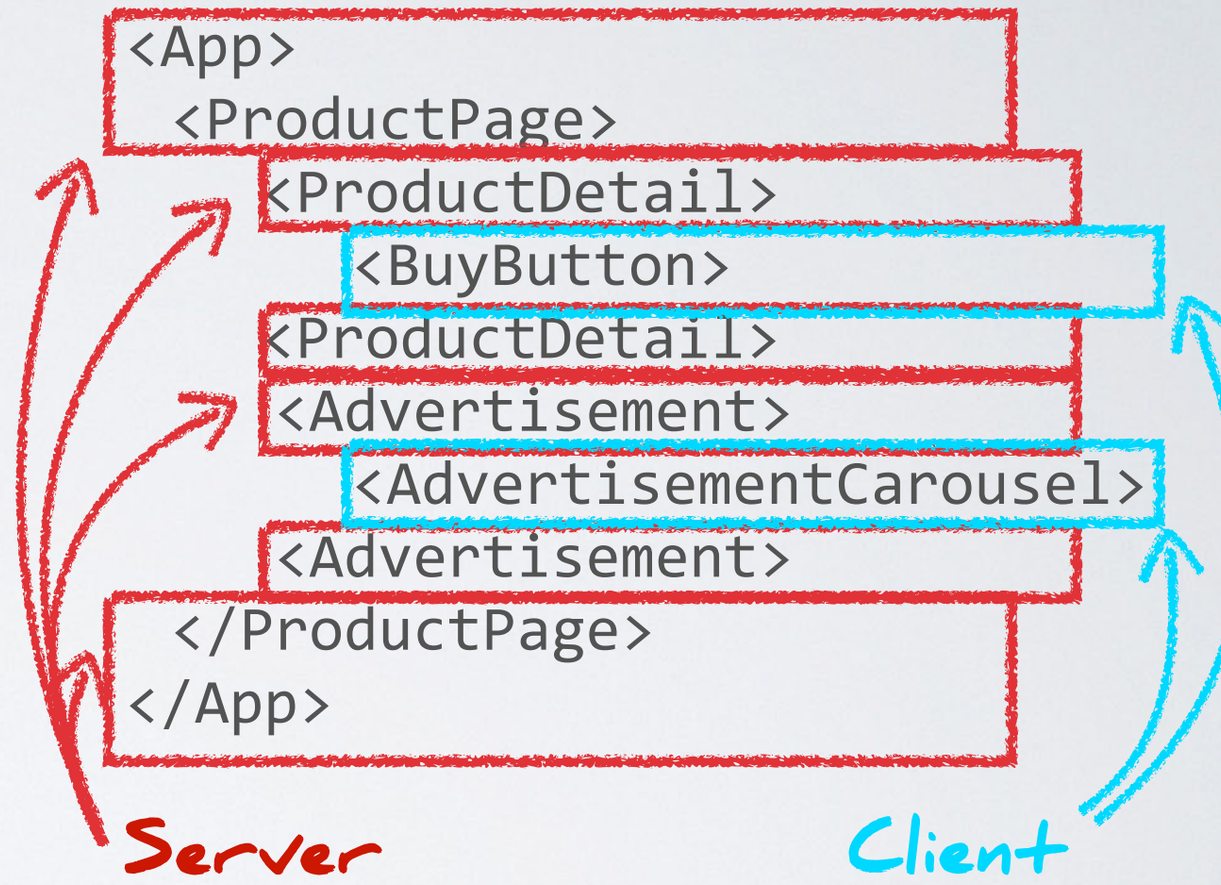
React Full-Stack Architectures



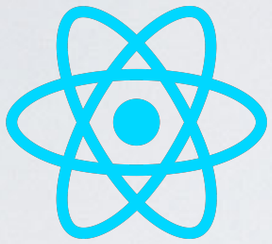
https://samnewman.io/patterns/architectural/bff/

The Frontend Boundary





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



qwik

Remix

RedwoodJS

TanStack Start

SVELTE KIT

SVELTE

SOLIDSTART

SOLIDJS

Nuxt

https://viteconf.org/

Next.js uses TurboPack:

TURBOPACK

https://turbo.build/
https://areweturboyet.com/

Rspack

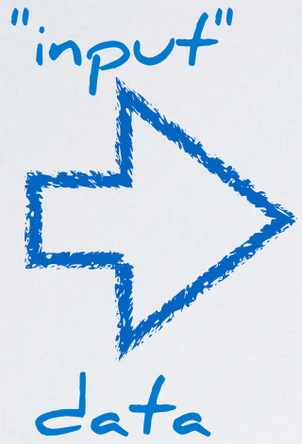
https://rspack.dev/

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

A Visual Guide To React Mental Models:
https://obedparla.com/code/a-visual-guide-to-react-mental-models/

Legacy: Class Components

```
import React from 'react';

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

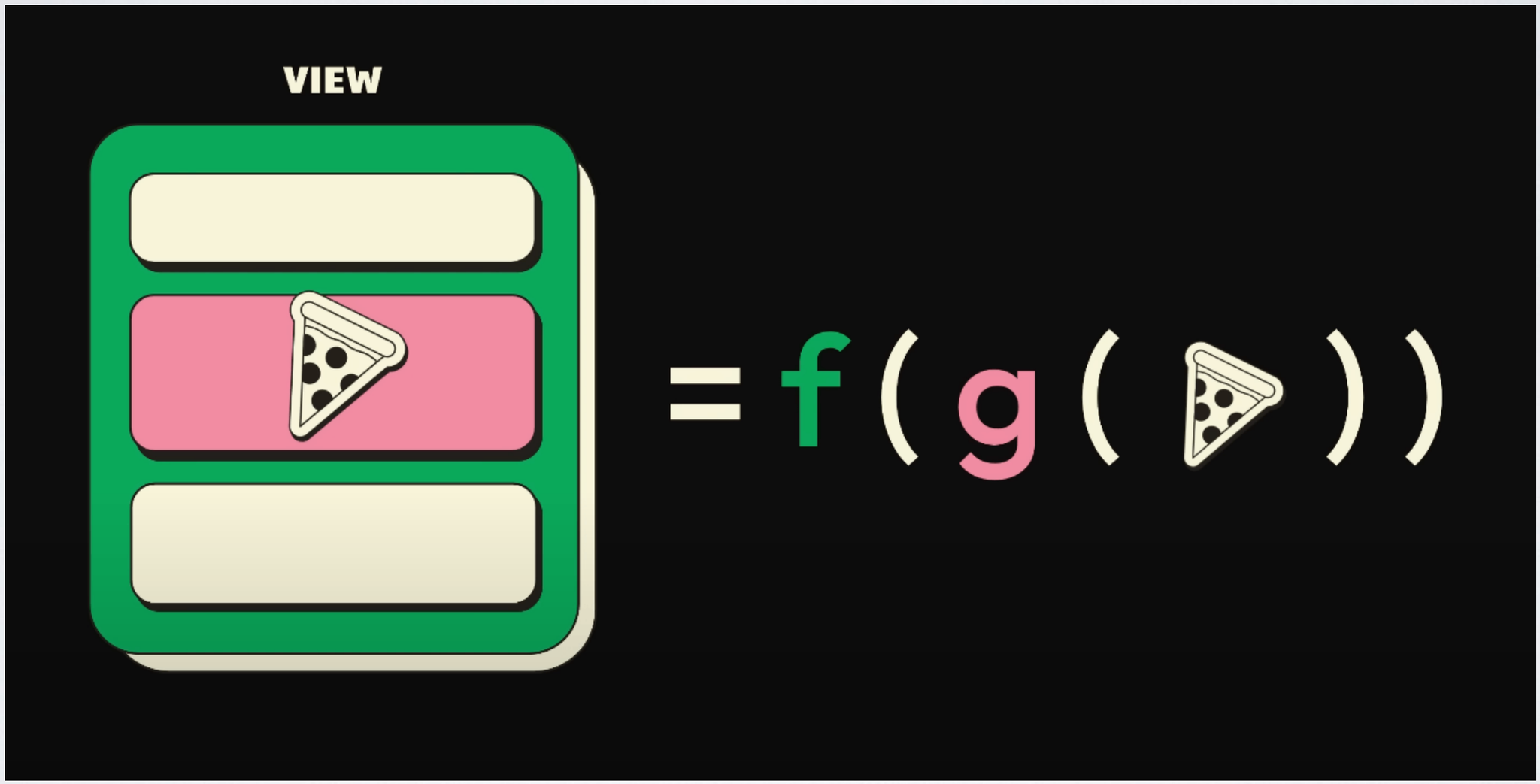
React 16.8 (February 2019) introduced "Hooks": a way to use state and lifecycle without writing a class.

This resulted in major changes in the React ecosystem.

https://playfulprogramming.com/posts/react-history-through-code

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://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 Spronsors the development of:

Next.js

React.js

Nuxt.js

Svelte.js

App Router vs. Page Router

new

old

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)

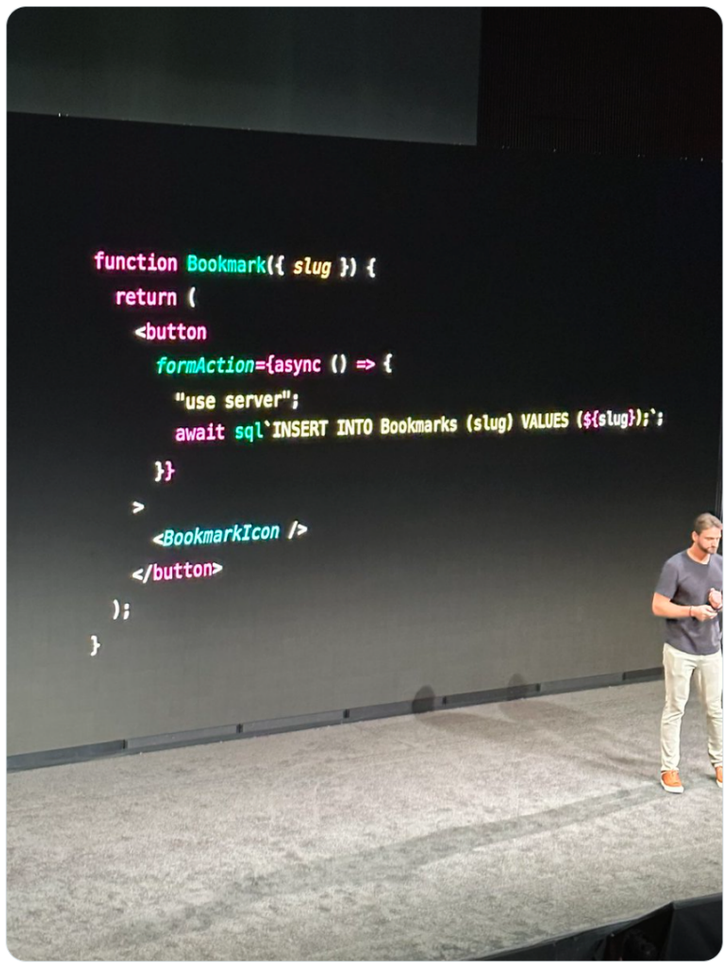
Seb ThisWeekInReact.com
@sebastienlorber

React devs

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

Adam Rackis
@AdamRackis

Server actions 🔥



8:22 PM · Oct 26, 2023 · **4.8M** Views

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

Original Talk: https://www.youtube.com/watch?v=9CN9RCznZc&t

The growing divide among React developers... https://www.youtube.com/watch?v=ILJbHHeFSsE

Are RSCs and NextJS Really That Bad?

Jack Herrington ✔

21K views • 2 days ago https://www.youtube.com/watch?v=u0OMdWlfdhg

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/

https://www.joshwcomeau.com/react/server-components/

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

React Client Runtime

generate a react tree on the client

Client Component

Virtual DOM

In-Memory, implemented in JavaScript

Browser DOM

render instructions running on the client

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

body
div
h1
div
input
button

diff

reconciliation

patch

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

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

2. send "RSC Payload" to the client

3. execute on the client

Server Component

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

The RSC Payload


```
1:"$Sreact.fragment"
2:I[6213,[],"OutletBoundary"]
4:I[6213,[],"MetadataBoundary"]
6:I[6213,[],"ViewportBoundary"]
0:{"b":"lGRtPxl8R9zGlIQbEBlF2","f":[["children","01-simple","children","__PAGE__",
["__PAGE__",{}],["__PAGE__",["$","$1","c",{"children":[["$","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":"$undefined","nonce":"$undefi
ned"}]],["$","$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
```

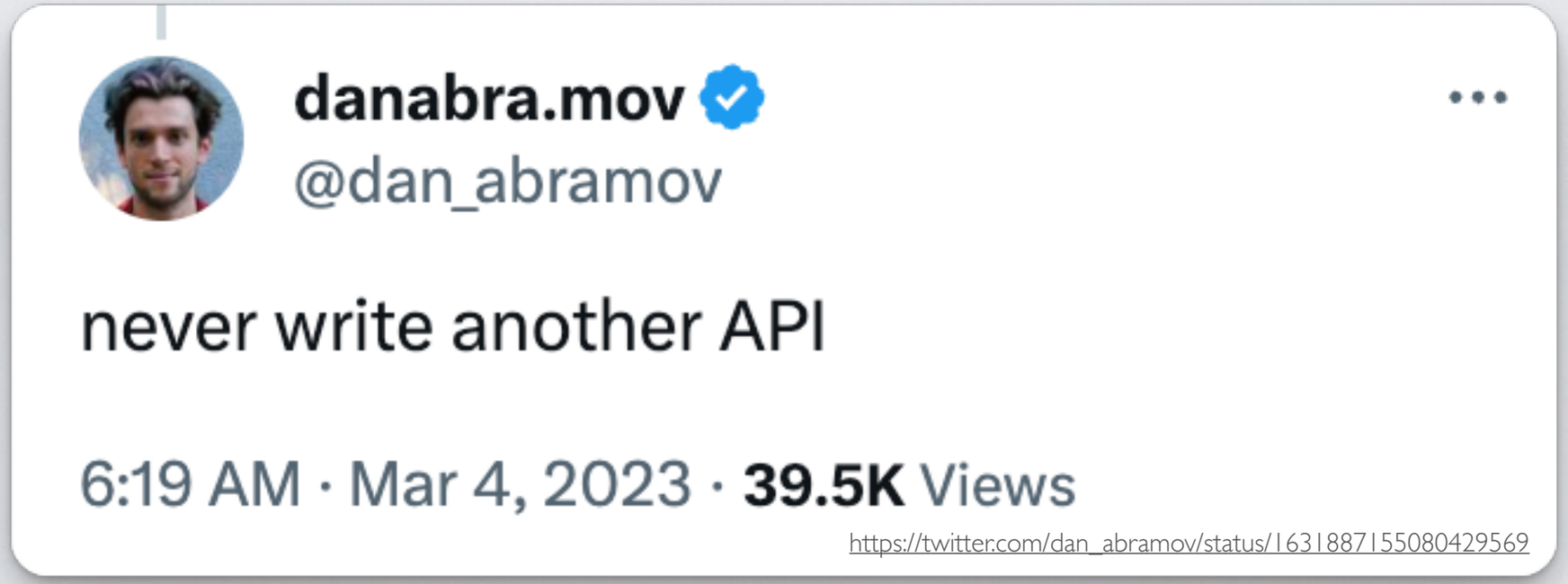
Load data on the server!

```
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!



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>
</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>
</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.

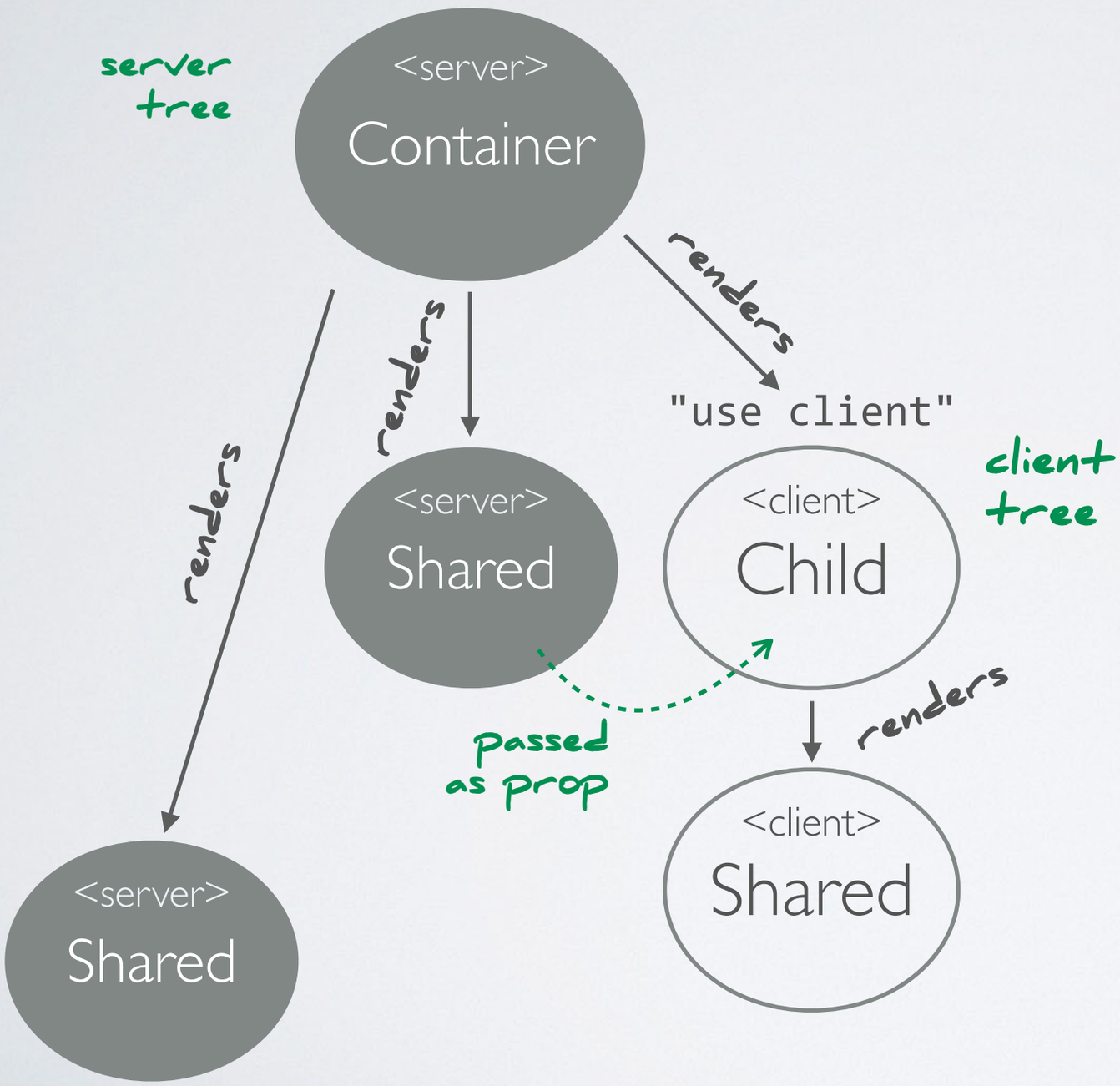
```
"use client"
export function Clock() {
  const [time, setTime] = useState(new Date())

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

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

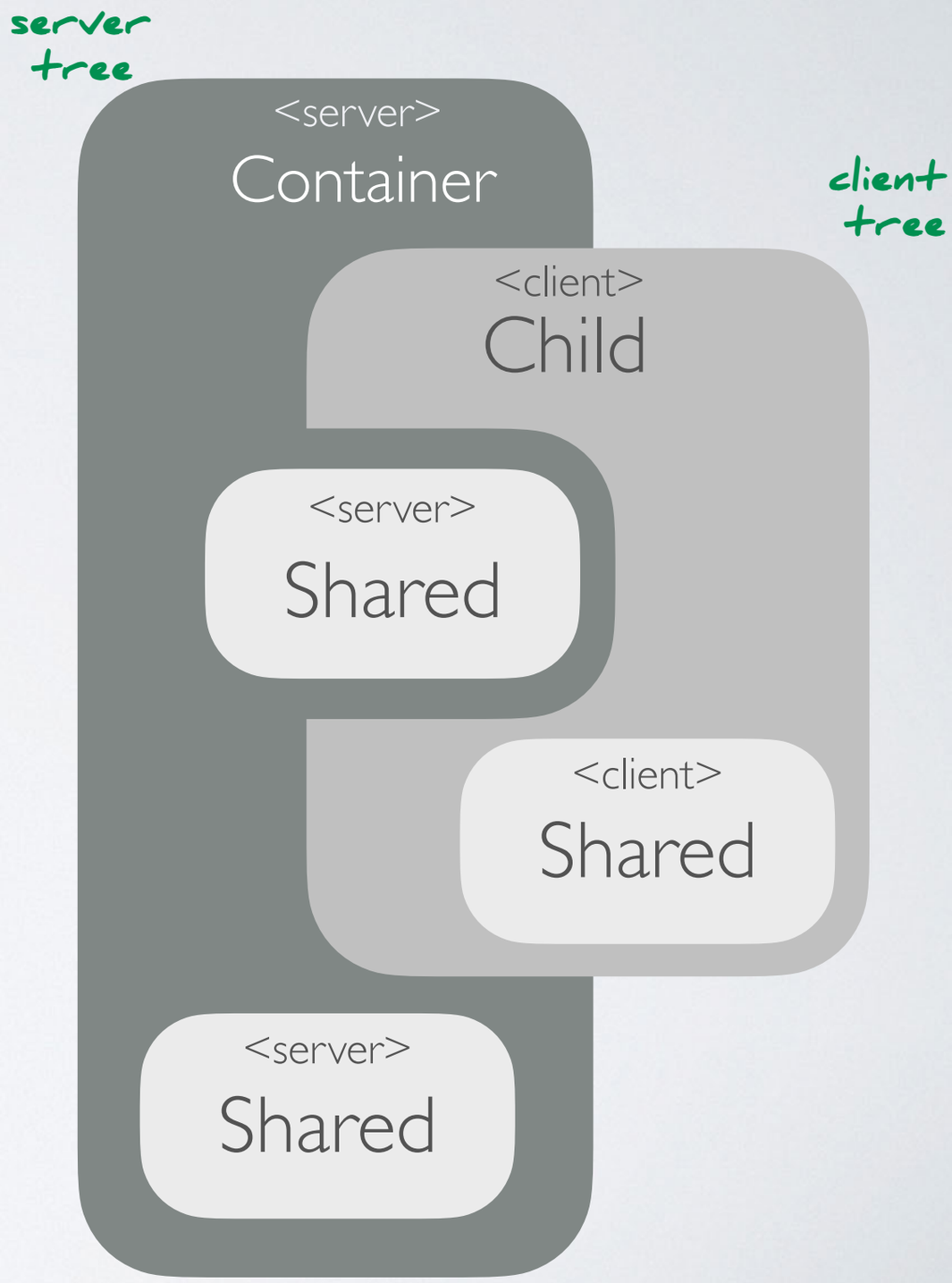
Client Components are "opt in".
Per default a component is a Server Component.

Composition

"logical perspective"

"composition perspective"





Full-Stack Data Flow

"composition perspective"

server tree

<server>
Container

props

<client>
Child

props

<server>
Shared

client tree

props

<client>
Shared

props

<server>
Shared

"phyisical perspective"

RSC Payload

React Client Runtime

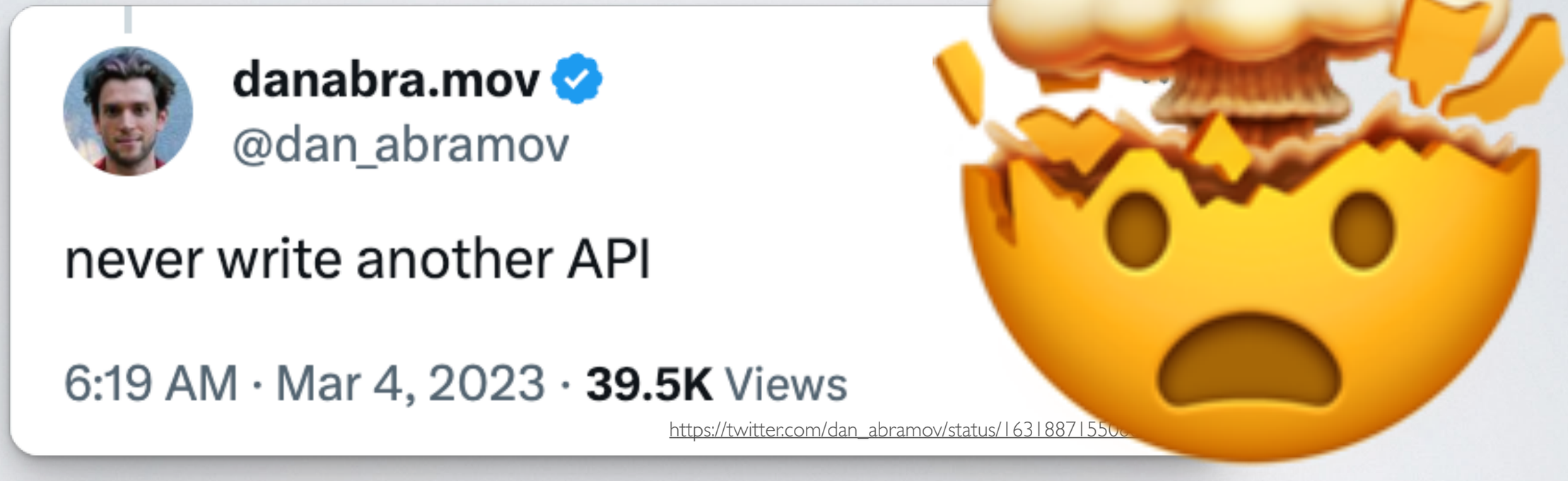
Container

virtual dom tree
renderd by RSC payload
(including props)

props

virtual dom tree
renderd by client
component

A component tree that spans between client and server!



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

Server Functions

Browser

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

RSC Payload sent to the browser

first scenario: server component calling server function

rendering

RPC endpoint

Server Function:

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

API call

React Client Runtime

(virtual dom tree)

rendering

second scenario: client component calling server function

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

JavaScript bundle loaded by the browser

JS

Server

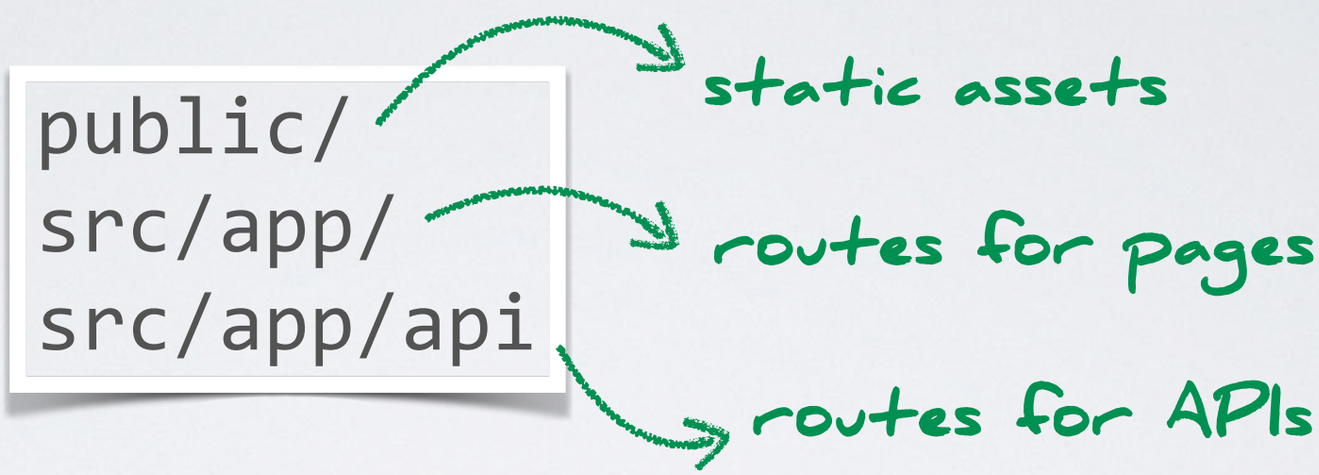
Network

https://react.dev/reference/rsc/server-functions

NEXT.JS

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

Project Structure



Next.js has a *file-based router*: directries 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