Modern Frontend - Back To The Server

ABOUT ME

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

- Freelancer, in den letzten 9 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 ...

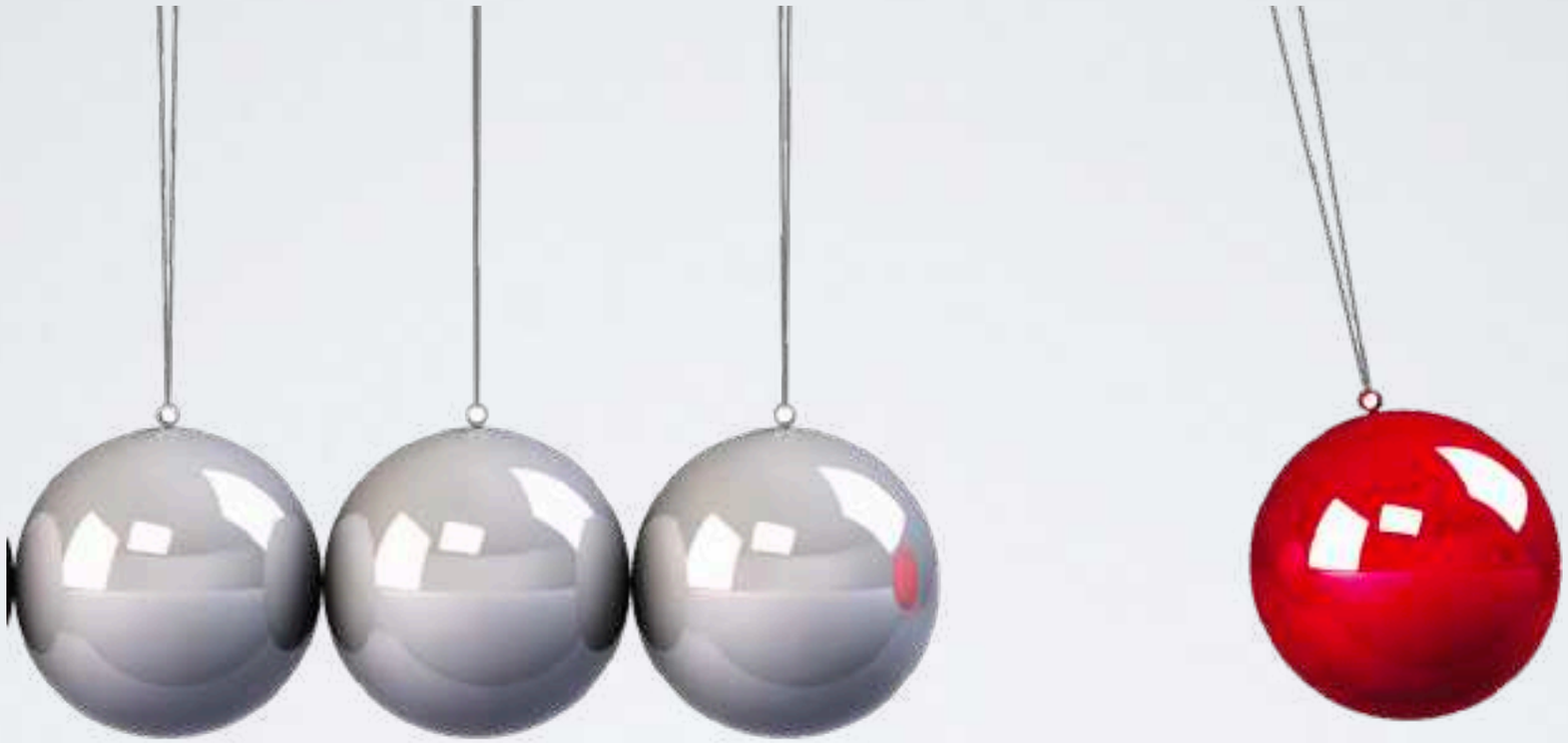
letsboot.ch
swiss dev training

digicomp

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

JavaScript / Angular / React / Vue / Vaadin
Schulung / Beratung / Coaching / Reviews
jonas.bandi@ivorycode.com

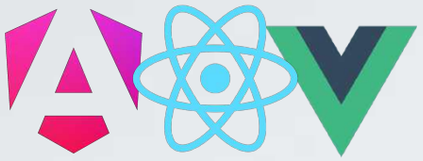
The Pendulum is swinging ...



Server ⟵ ⟶ Client

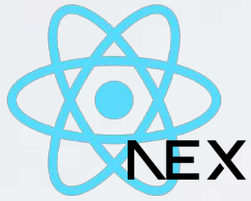
... will it ever stop?

AGENDA



Where are we? - The Era of SPAs ...

(React Refresher)



React Server Components

Prior Art:

astro Island Architecture

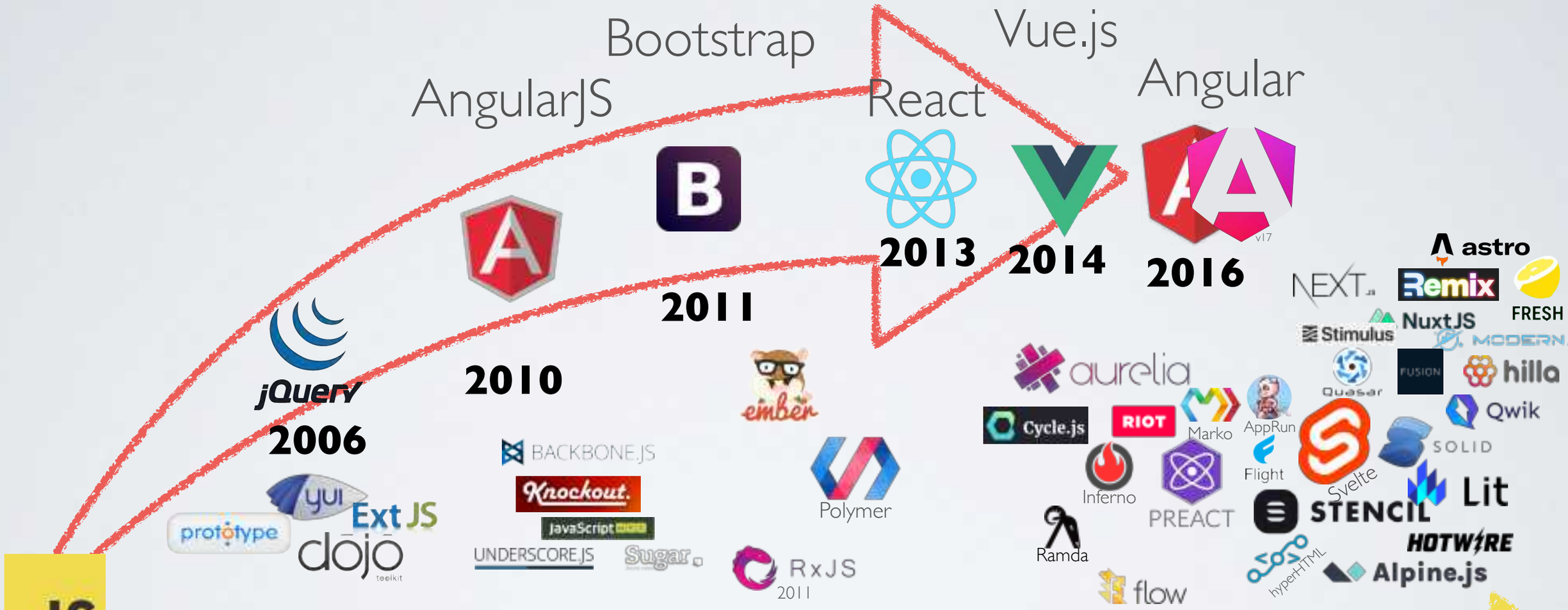
Remix RPC-style fetching and actions

vaadin Server-Driven SPA

My goal is to make you feel like this:

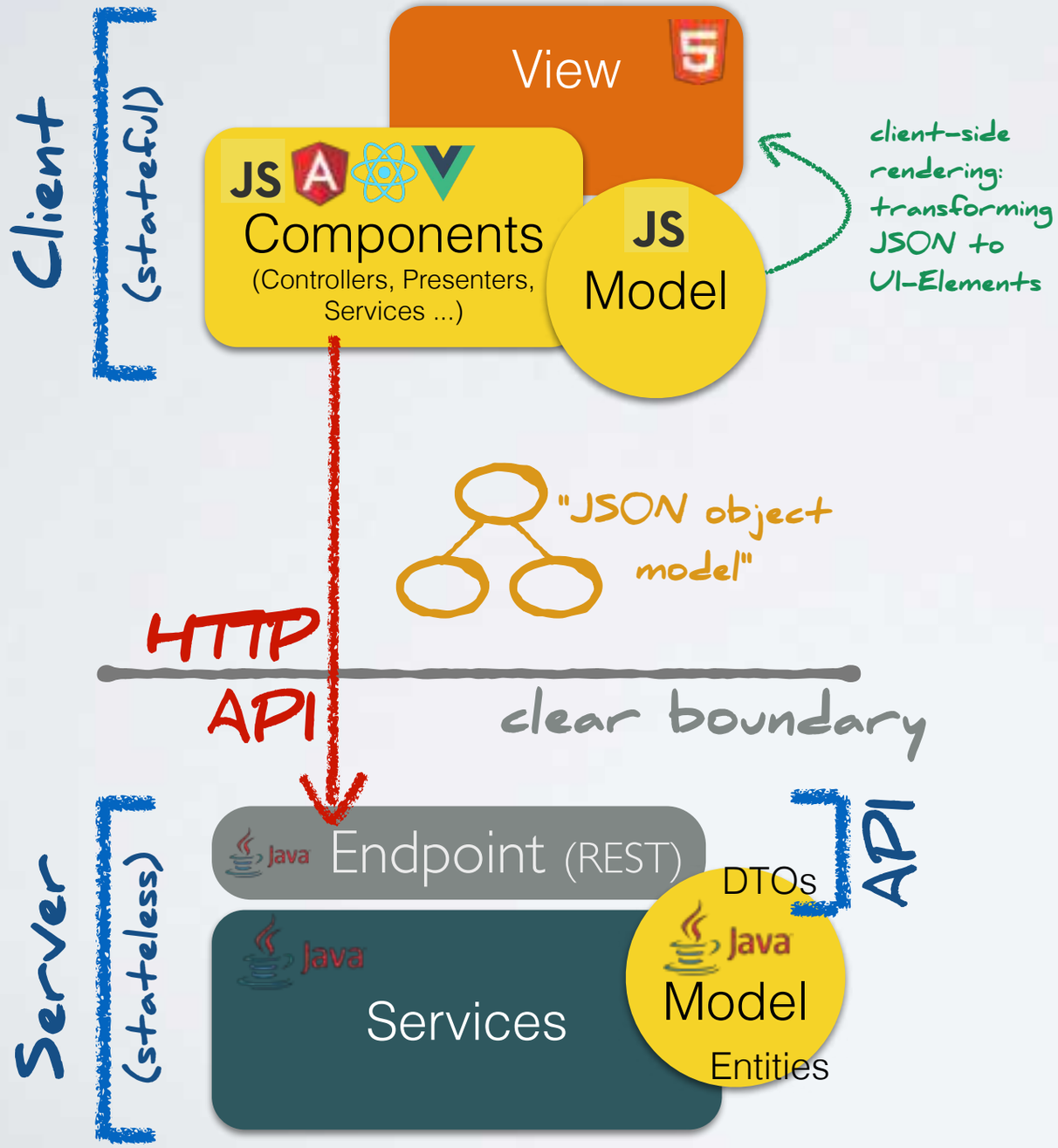
The Era of Single Page Applications

The Frontend JavaScript Ecosystem



Architecture for Single Page Applications:

The Role of the API



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

Symptoms:

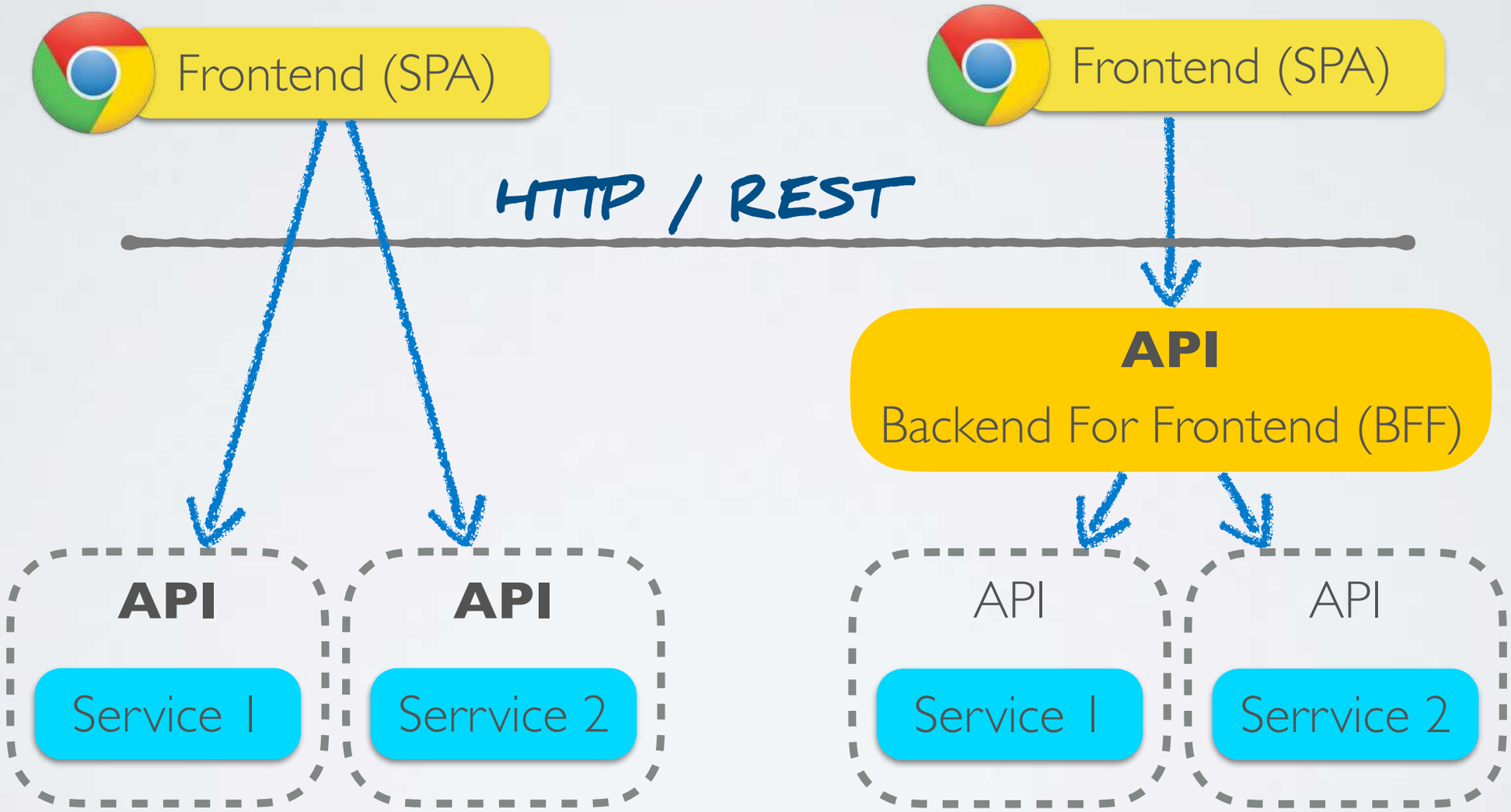
- "API-First" Design
- "The central role of API-Gateways" (the return of ESBs)

...

Creating a formalized API is a non-trivial effort: 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 ...

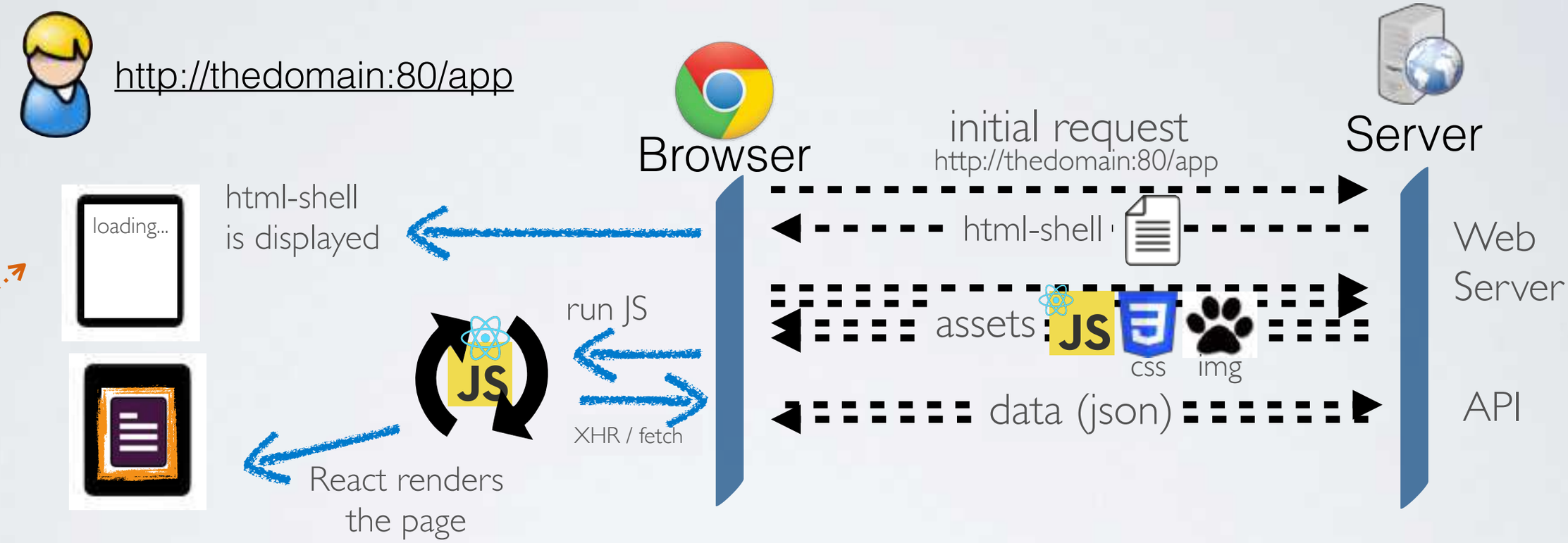
Architecture: APIs for SPAs



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

Pure SPA Demo

Traditional SPA:
Client Side Rendering



Components are rendered on the client.

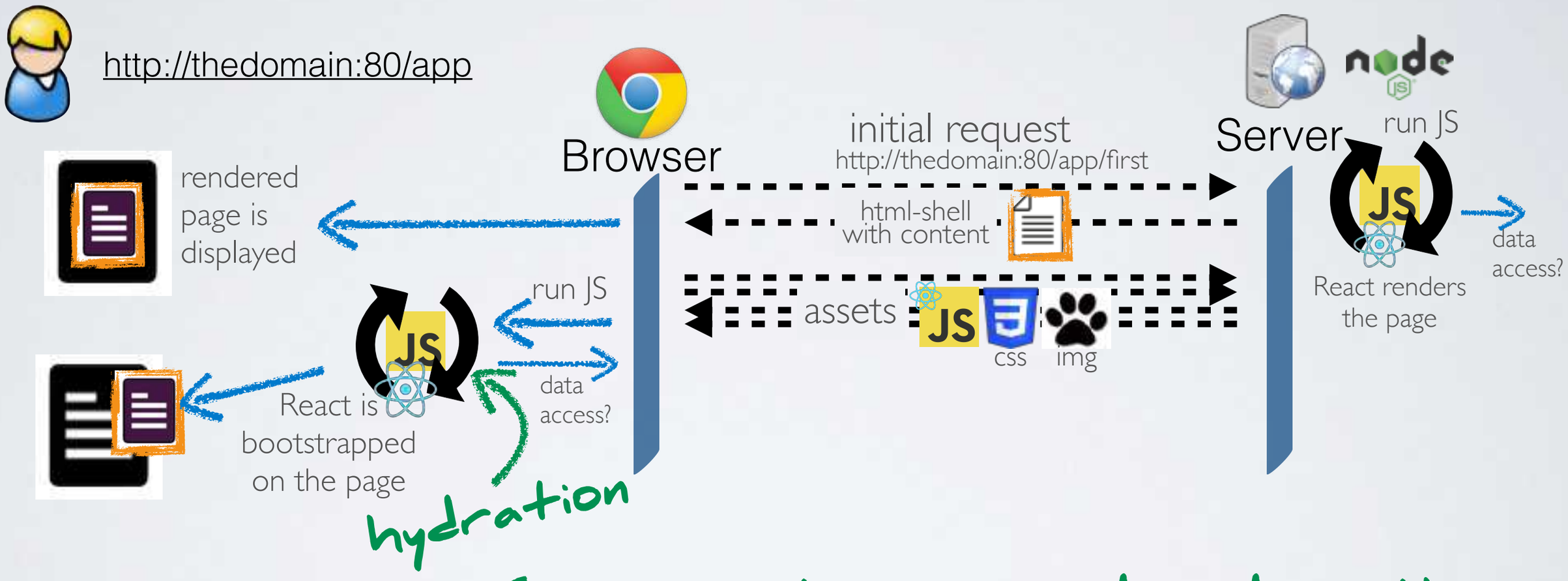
The achilles heel of SPAs:

- time to first paint
- search indexing / social previews

SPA SSR Demo

SPA with Server Side Rendering (SSR)

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



Components are rendered on the server and the client.

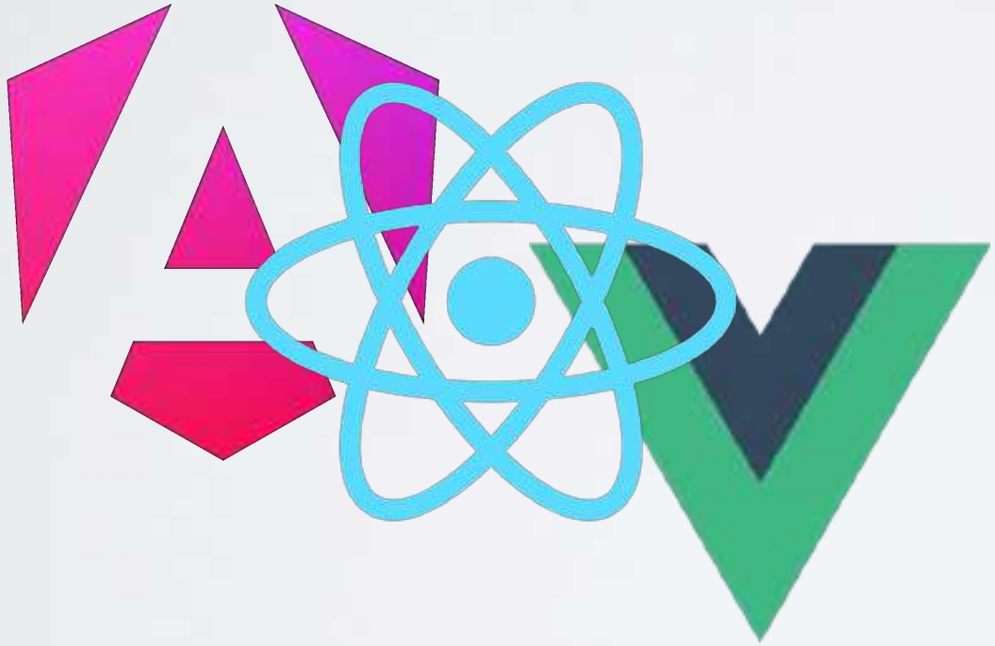
Advantages:

- 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)

This talk is not about Server Side Rendering!



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

Even Angular supports non-destructive hydration since v16..
The Angular CLI introduced SSR scaffolding in v17.

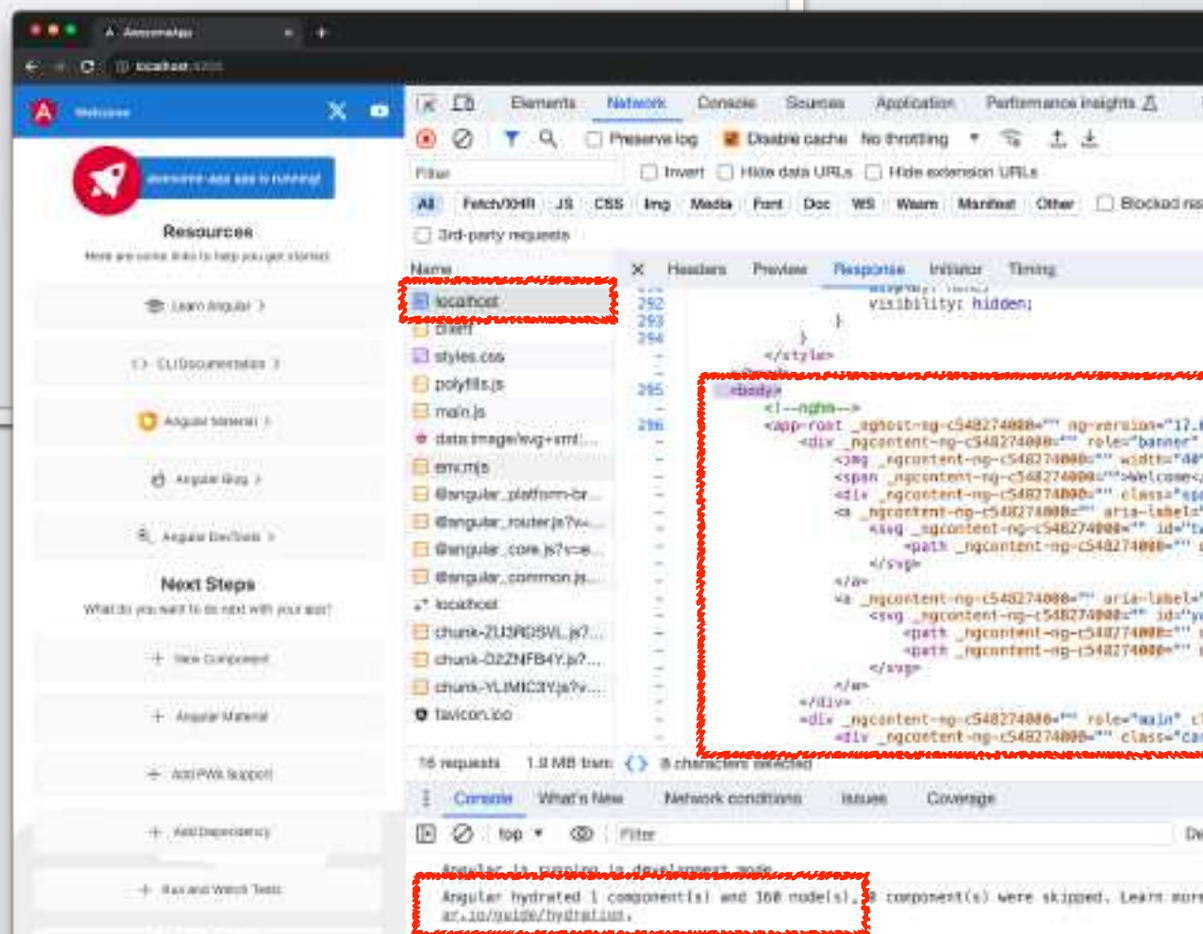
https://angular.io/guide/hydration
https://blog.angular.io/angular-v16-is-here-4d7a28ec680
https://angular.dev/cli/new

Server Side Rendering (SSR)

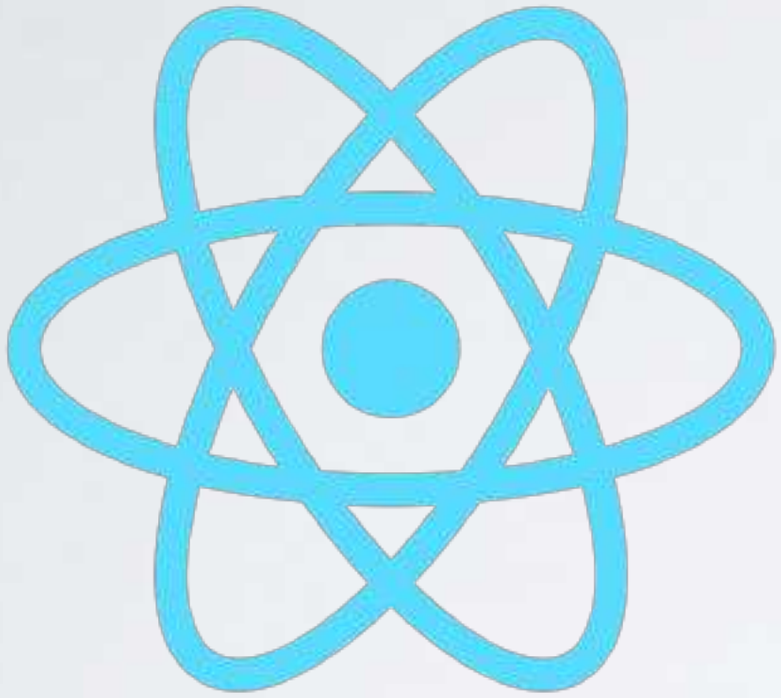
🤫 ... even Angular can do that ...

Using Angular CLI v17

```
> npx @angular/cli@next new ng-ssr
? Which stylesheet format would you like to use? CSS
? Do you want to enable Server-Side Rendering (SSR)
    and Static Site Generation (SSG/Prerendering)? Yes

...
✔ Packages installed successfully.
cd ng-ssr
npm start
```

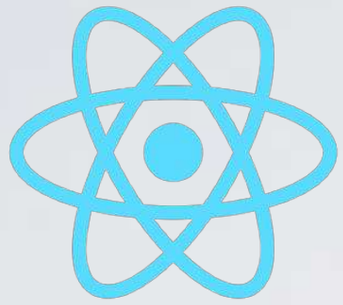


React Refresher:



Good old React as we know it ...

React is now11 years old! (released in May 2013).
jQuery was 7 years old, when React was released ...



React is Easy!

component

```
export function App() {
   return (
      <div><Greeter/></div>
   );
}
```

component

3rd party component

```
export function Greeter() {
   const {id} = useIdParamFromUrl();
   return <Alert severity="info">This id a demo: {id}!</Alert>;
}
```

Passing Props

custom hook

```
export function useIdParamFromUrl() {
   const { id } = useParams();
   return userId;
}
```

3rd party hook

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

The Virtual DOM

Components

App

Title

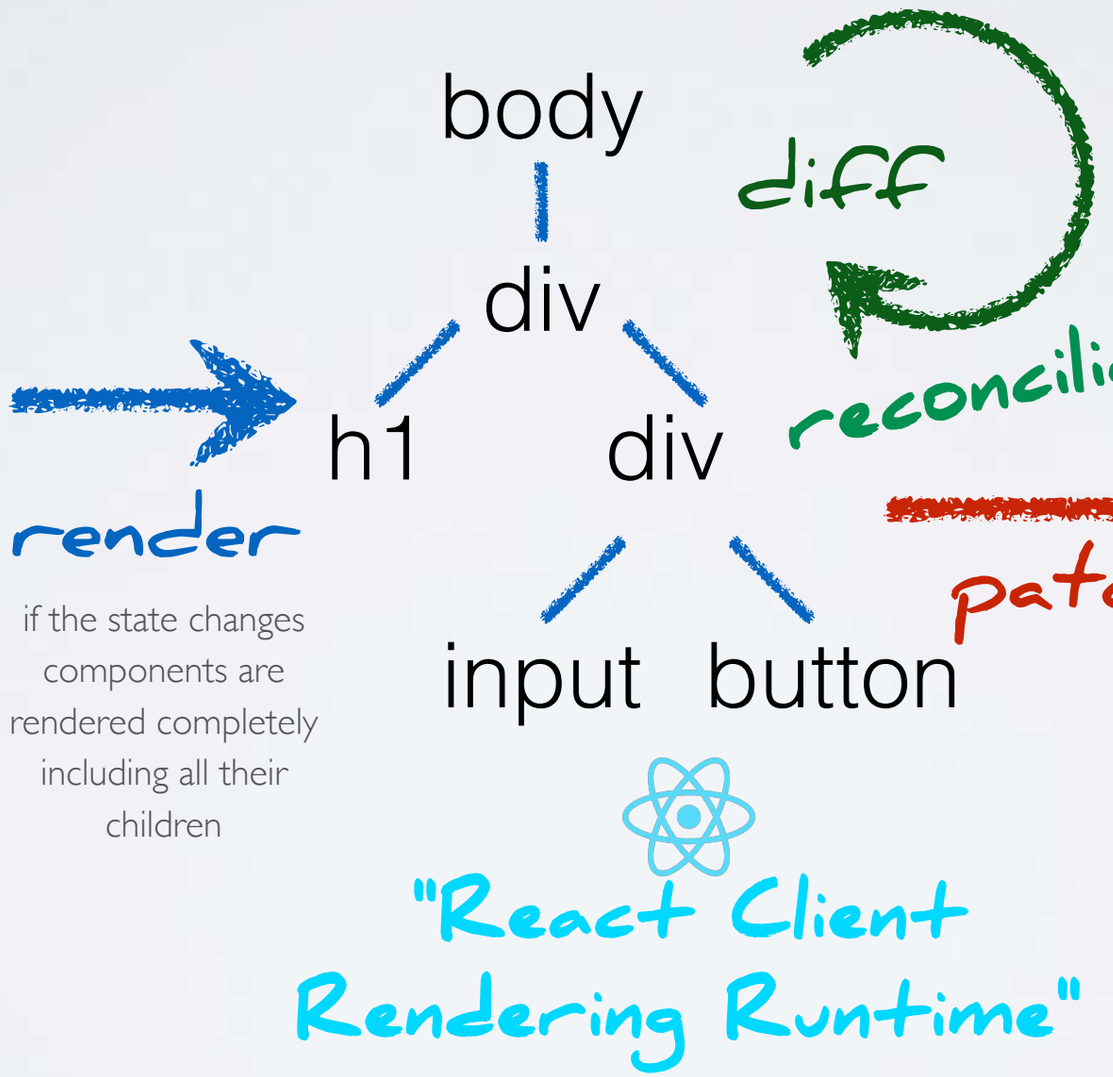
Content

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

```
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

ignoring stale responses

cleanup function

effect dependencies

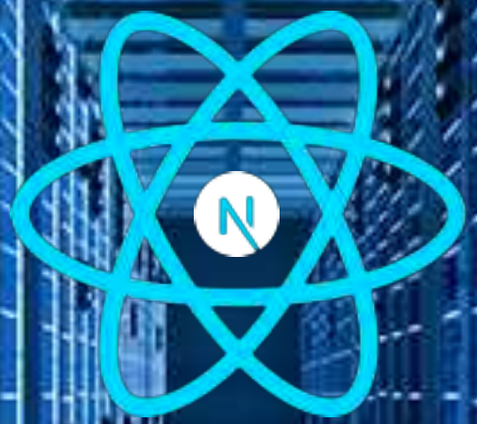
The sad face of React...

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

React Server Components

React Server Components is an Architecture!


I want you to forget everything
you know about React!



Introducing:

"React Server"

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

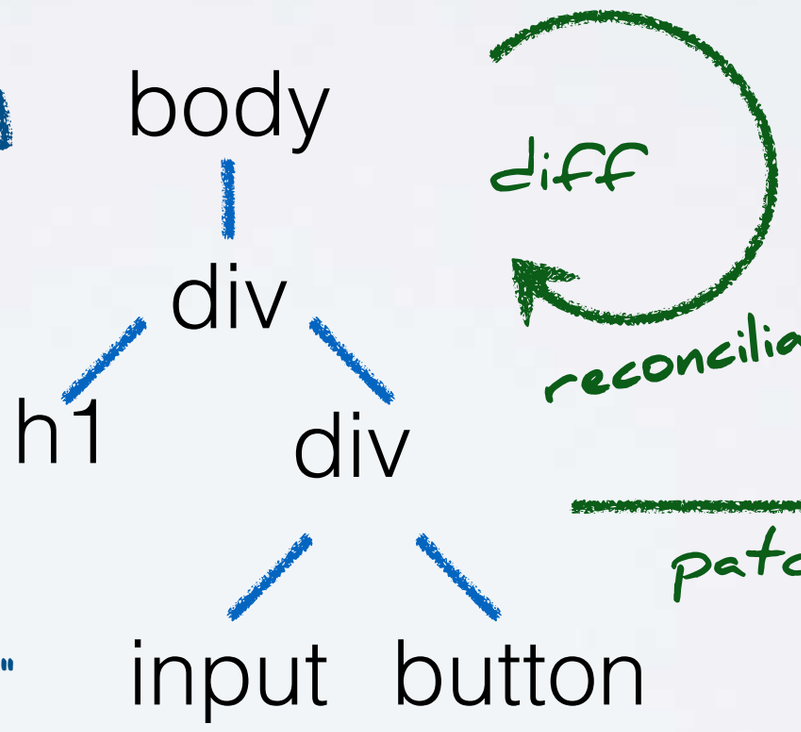
render instructions running on the client

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

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

2. send "RSC Payload" to the client

3. execute on the client



diff

reconciliation

patch

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

Server Component

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

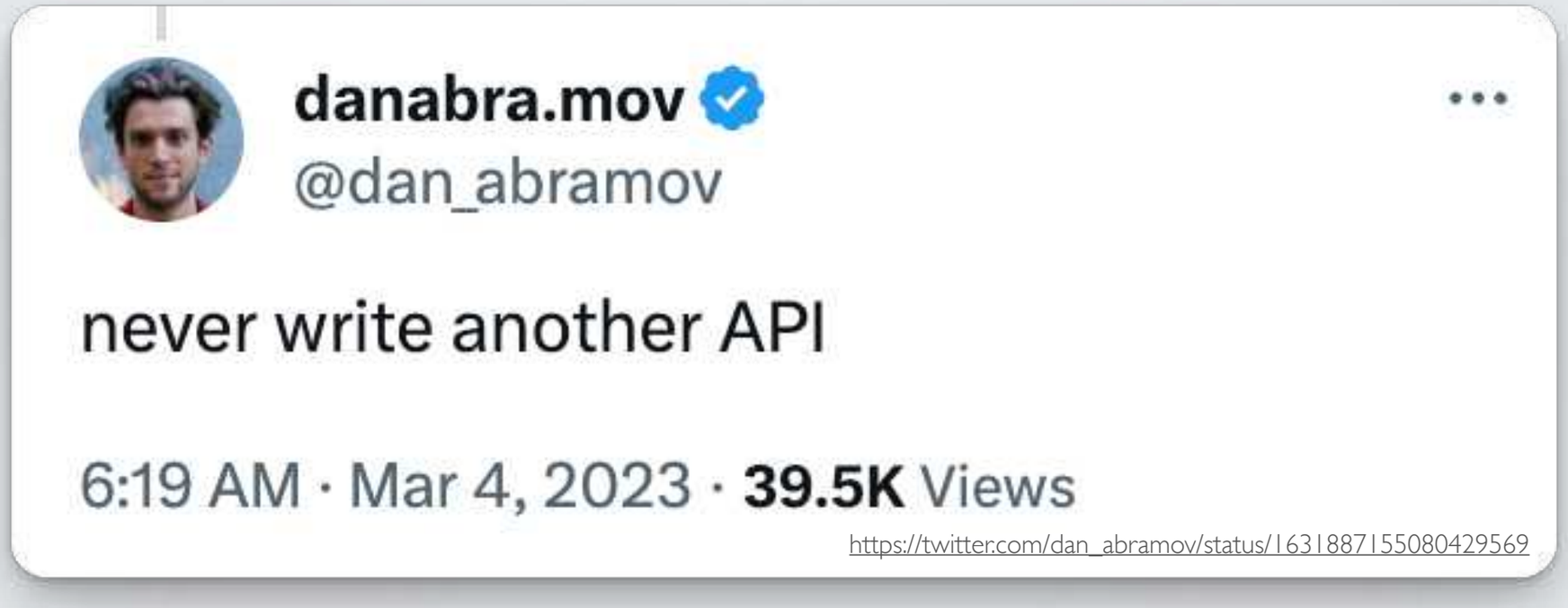
Load data on the server!

```
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

Wait!

It looks like PHP

from 25 years ago!

Prepare for more ...

Limitations

React Server Components

... are rendered on the server only

- Can't use hooks:
  no state: `useState`, `useReducer`, `useContext`
  no lifecylce: `useEffect`
- Can't handle DOM events:
  `onClick`, `onBlur` ...
- Can't use browser APIs:
  `localstorage`, `geolocation` ...

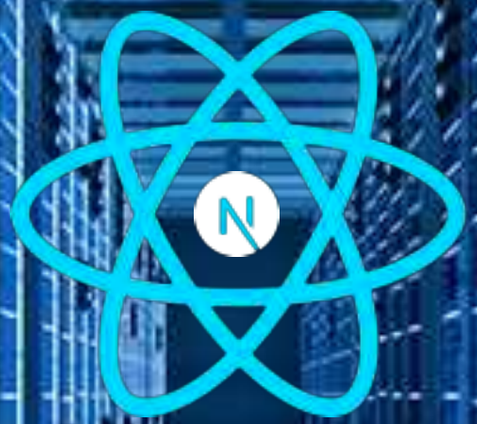

Restore Memory ...

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.



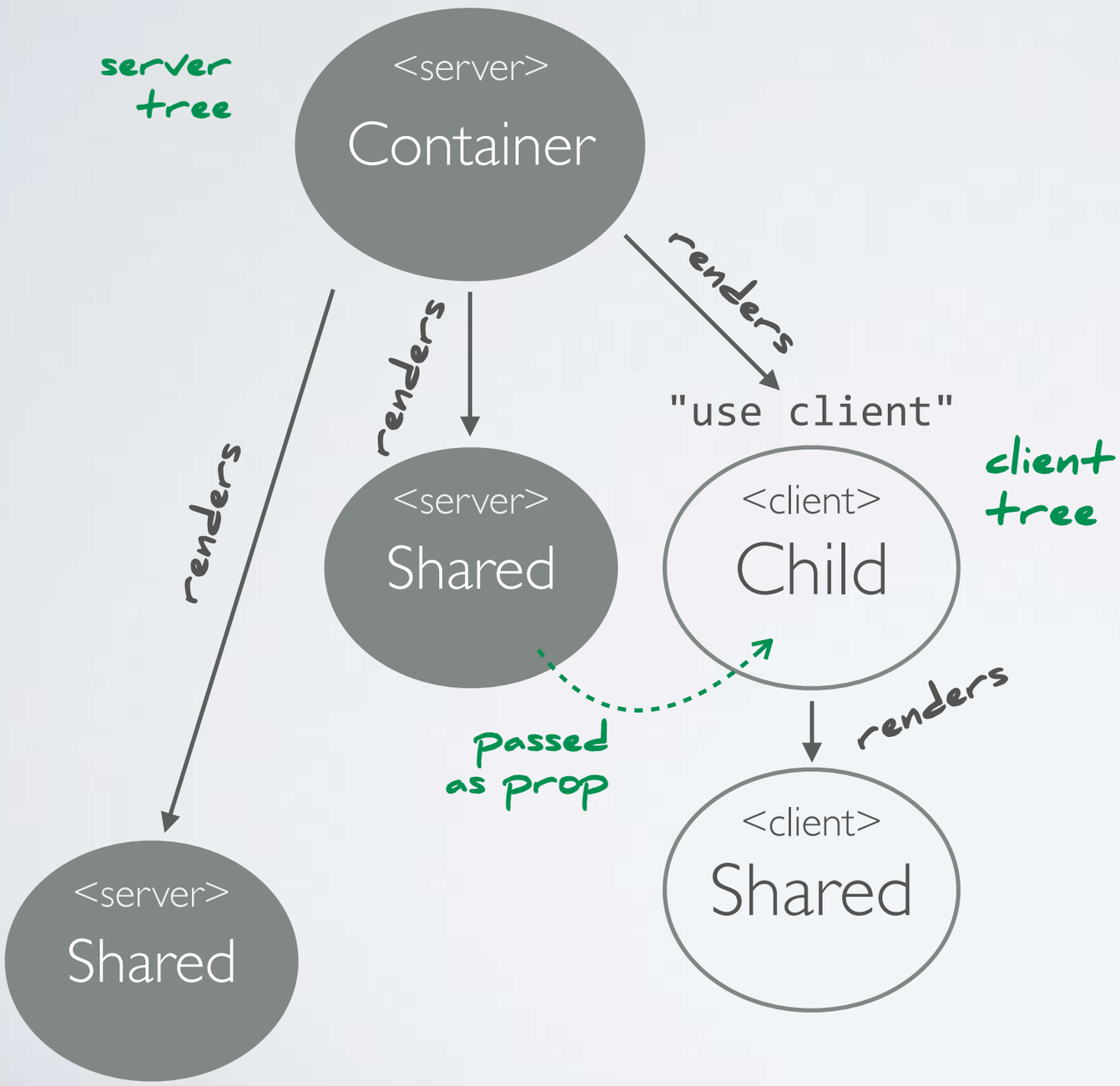
Introducing:

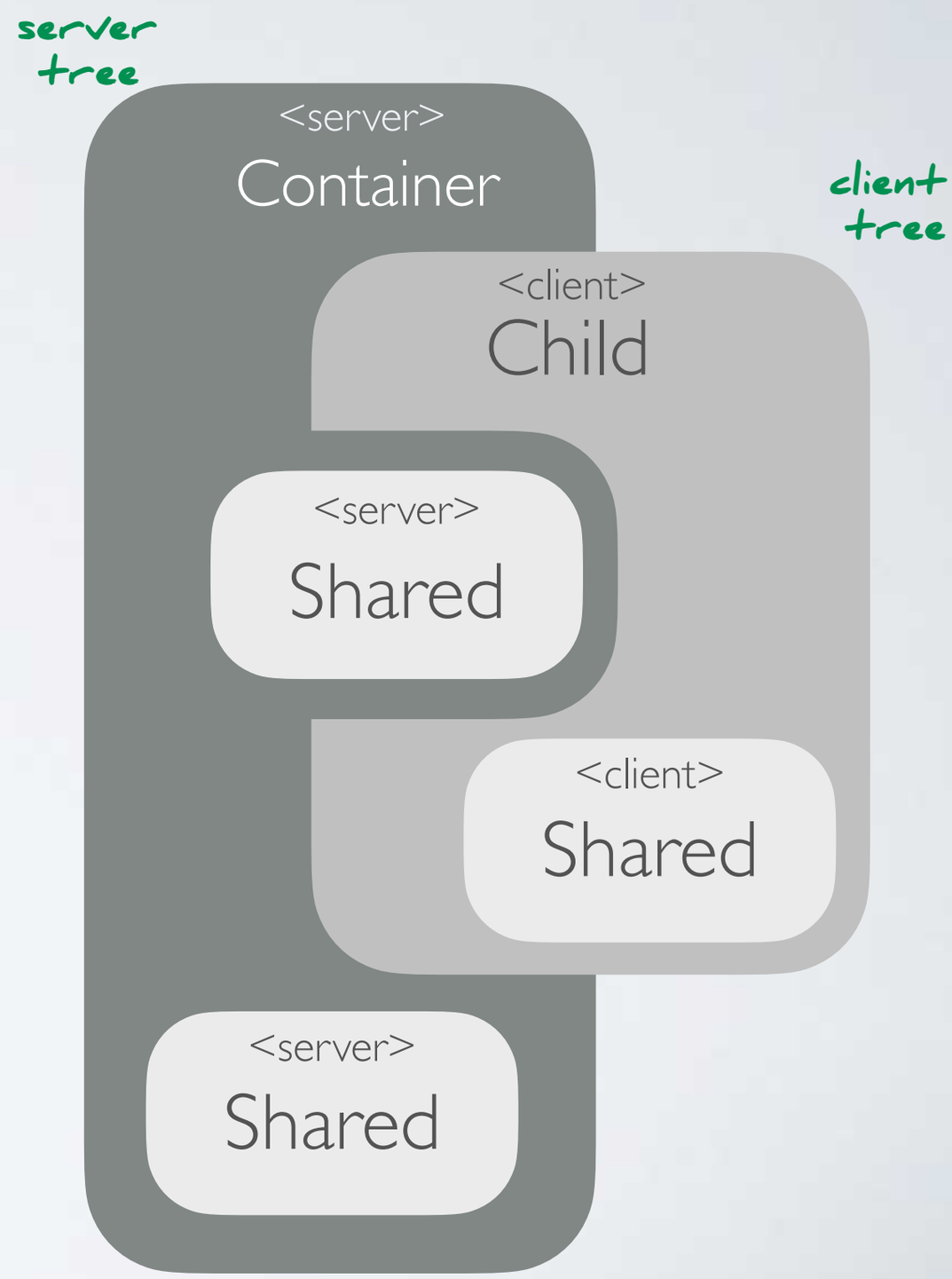
"React Server" + (traditional) Components

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

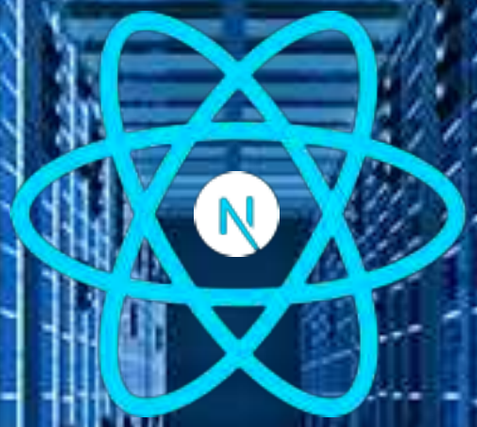
virtual dom tree
renderd by RSC payload
(including props)

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 ...



Introducing:

"React Server"

+ Client Components

+ Server Actions

Browser

Network

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

RSC Payload sent to the browser

rendering

first scenario: server component calling server function

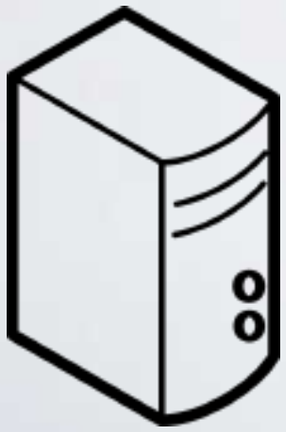
React Client Runtime

(virtual dom tree)

RPC endpoint

API call

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



Server

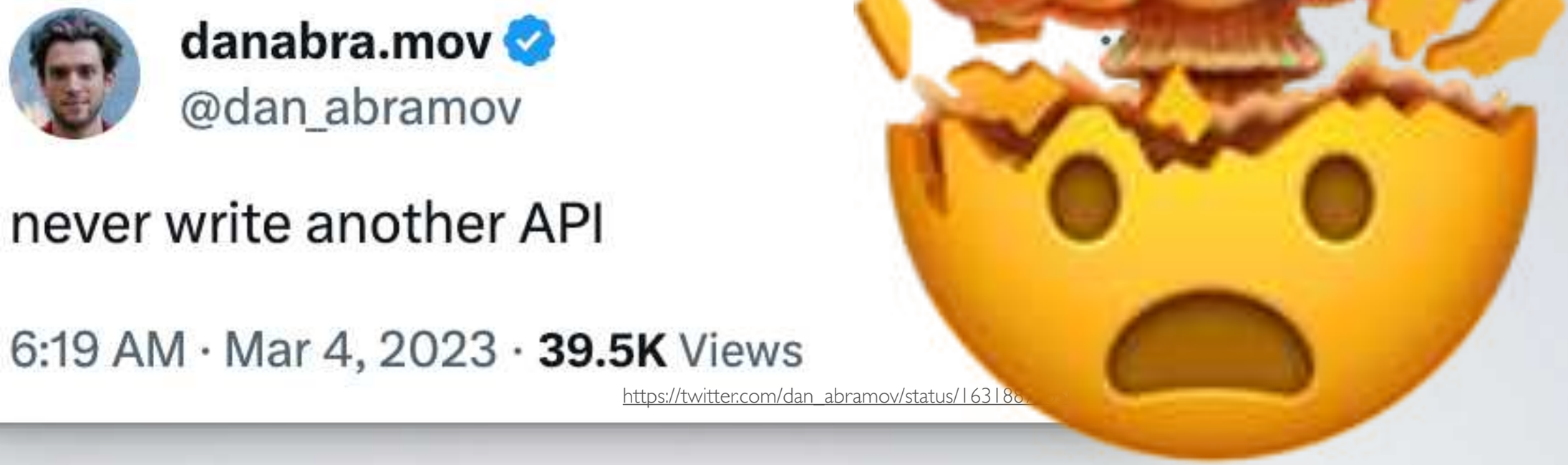
JavaScript bundle loaded by the browser

JS

rendering

second scenario: client component calling server function

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



In case you need the repetition:
Also no HTTP-API for mutating ....

Server ... Client ... it's confusing ...

```
'use client'
'use server'
```

network boudary,
js bundle shipped to client

network boudary,
RPC endpoint called by client

danabra.mov @dan_abramov · Oct 28

"use server" makes a server function callable from the client. it doesn't change where the function runs (which as you correctly noted is on the server regardless).

1 2 18 1.5K

danabra.mov @dan_abramov · Oct 28

you can think of "use server" as "here's an entry point into server code". the opt-in is important because you don't want arbitrary code to be callable.

1 15 1.7K

danabra.mov
@dan_abramov

"use server" = server code that can be referenced by the client (becomes API endpoints)

"use client" = client code that can be referenced by the server (becomes <script> tags)

10:23 PM · Oct 28, 2023 · **20K** Views

https://twitter.com/dan_abramov/status/1718362813733785970

New Hooks

There are new hooks for managing async operations with Server Actions in Client Components:

- `useFormState`

  https://react.dev/reference/react-dom/hooks/useFormStatus

- `useFromStatus`

  https://react.dev/reference/react-dom/hooks/useFormStatus

- `useOptimistic`

  https://react.dev/reference/react/useOptimistic

Summary

"React Server Components"

- is a full-stack architecture
- is based on the proven component model of React
- extends the composability patterns of React to the server-side
- solves client-server communication with a consistent programming model based on components transparent RPC
- is the answer for data-fetching and mutations in React
- has huge potential for an ecosystem of 3rd party full-stack components

- also improves performance by enabling smaller JS bundles and streaming server responses.

Disclaimer

NEXT.JS

The demos in this talk were based on Next.js.
Next.js is currently the only mature framework that implements React Server Components.
https://nextjs.org/

In reality it is difficult (and frustrating) to draw the boundary between features of React Server Components Next.js.

Waku

Waku is an experimental framework that implements RSCs.
https://waku.gg/

Remix

Remix announced RSC integration in a future version.
https://remix.run/

Prior Art

Island Architecture

Prior Art:

Island Architecture

The islands architecture encourages small, focused chunks of interactivity within server-rendered web pages.

astro

https://astro.build

FRESH

https://fresh.deno.dev

https://docs.astro.build/en/concepts/islands/

https://www.patterns.dev/posts/islands-architecture

STAR WARS

RETURN OF THE RPC

RPC-Style data fetching & mutations

Remix

NEXT.JS
React Server Components

https://remix.run/docs/en/main/route/loader
https://remix.run/docs/en/main/guides/data-writes#remix-mutation-start-to-finish

SOLIDSTART

https://start.solidjs.com/core-concepts/data-loading
https://start.solidjs.com/core-concepts/actions

SVELTEKIT

https://kit.svelte.dev/docs/load
https://kit.svelte.dev/docs/form-actions

Hilla
https://hilla.dev/

qwik Qwik City

https://qwik.builder.io/docs/route-loader/
https://qwik.builder.io/docs/action/
https://qwik.builder.io/docs/server$/

Also:
https://trpc.io/

Server Driven SPA

Prior Art:

Server Driven SPAs

aka "Live View"

Vaadin

Blazor Server

Phoenix Framework

Enabling SPAs with a server-side programmig model and no need for a REST API.

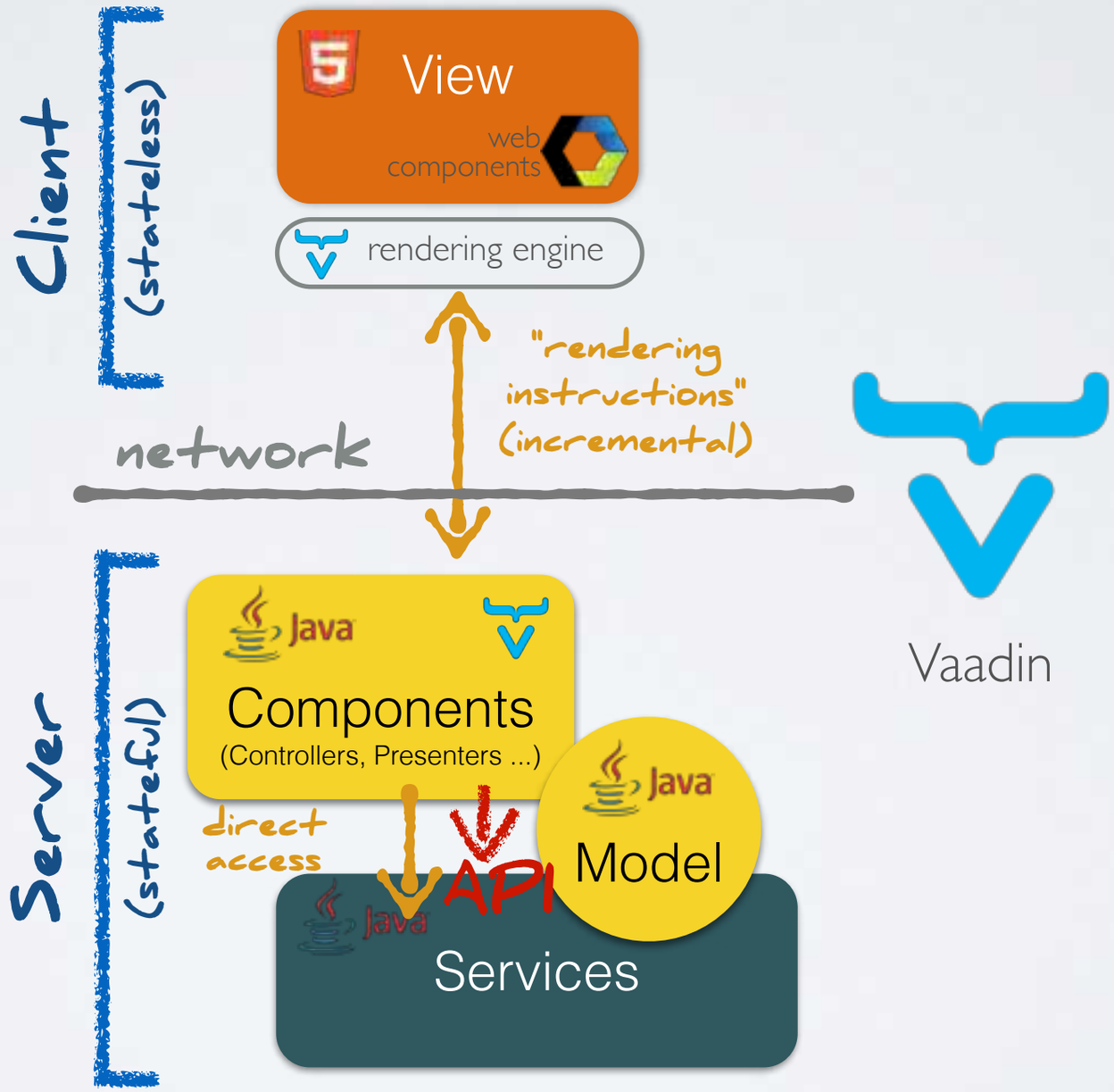
Demos:

https://labs.vaadin.com/business/
https://blazor.syncfusion.com/demos/datagrid/overview?theme=bootstrap4
https://liveview.zorbash.com/

Term definition: https://github.com/dbohdan/liveviews

Vaadin Architecture

using the browser just as a "rendering engine"



Same concepts are used in .NET with Blazor Server

https://vaadin.com/

Thank you!

Slides & Code: https://github.com/jbandi/guild42-rsc

Questions? Discussions ...

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

jonas.bandi@ivorycode.com