Exploring Modern React Architectures

Let's explore CSR, SSR, SSG, ISR, RSC, and even more acronyms!

React Router vs TanStack Router vs Next.js vs Astro showdown!

Kevin Van Cott
Senior Software Engineer &
OSS Maintainer

What We Will Learn in this Workshop

- React – *obviously*
- React Meta-Frameworks - their pros and cons
  - Vite (a bundler that multiple frameworks are built on top of)
    - React Router (with and without Framework mode – formerly Remix)
    - TanStack Router (with and without Start)
    - Astro
  - Next.JS
- The differences between CSR, SSR, SSG, ISR, RSCs, and more
- Other important concepts such as Hydration, Streaming, and Suspense
- React Query (TanStack)

What We Won’t Learn, But We Can Talk About

- Other React Meta frameworks/tools like CRA, Redwood, Gatsby, etc.
- Competing JavaScript Frameworks like Svelte, Vue, Solid, Angular, etc.
- The numerous state management libraries to choose from in React
- How to write and organize your CSS
- Monorepos
- Authentication
- Unit and e2e Testing
- Whether or not you should use TypeScript (It’s 2025!!!)

Your Main Takeaways Should Be

- An understanding of what modern React usage looks like
- The Pros and Cons of different data loading strategies
- The major differences between all major React Meta Frameworks
- A few tidbits of wisdom (or at least knowledge) here and there
- **Is React a good fit for what you want to build?**

Rough Schedule (6 Hour Workshop)

- Hour 1 – Intro/Slides
- Hours 2 & 3 – Deep Dive on SPAs and React Query
- Hours 4 & 5 – Deep Dive on Meta Frameworks for SSR, SSG, RSCs
- Hour 6 – We'll vote on what we want to do
  - An hour of scaffolding a React app with your framework of choice in groups
  - Cover previous material in more depth (if we were running low on time)
  - Bonus Topics and Rants

My Background

- I’m a software developer with 7 years of experience
  - Worked at State of Nebraska, Talent Plus, ALLO, Fusion Medical, Manifest Cyber, RentVision
- Open-Source Contributor and Maintainer
  - Built Material React Table and Mantine React Table
  - TanStack Maintainer for 3+ years
    - I maintain or work on TanStack Table, Pacer, and Virtual mostly
  - TanStack Consultant
- Coding Bootcamp Instructor - Nucamp

The Acronyms – What Do They Mean?

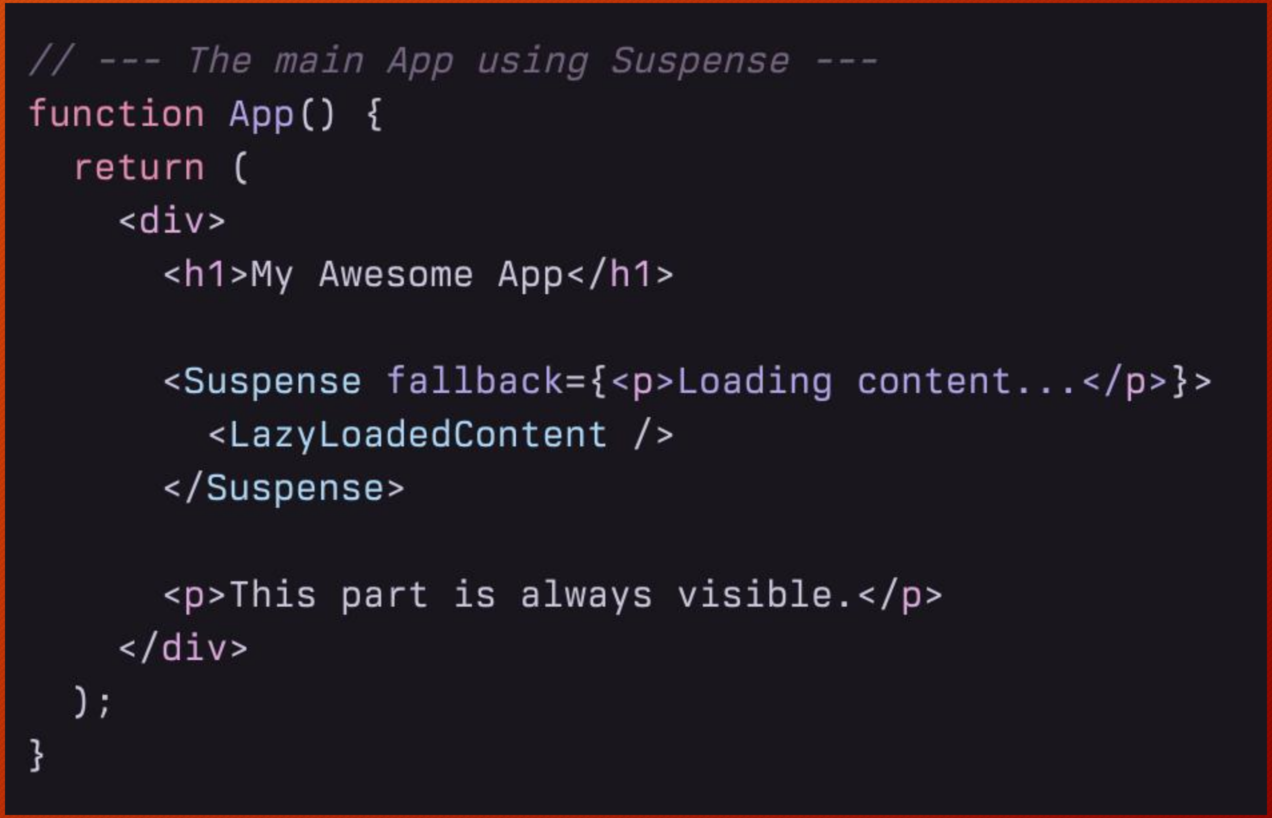
• **CSR**	- Client-Side Rendering
• **SSR**	- Server-Side Rendering
• **SSG**	- Static Site Generation
• **ISR**	- Incremental Static Regeneration
• **RSCs**	- React Server Components

Some Other Important Terms To Know

- **Hydration** - What does it mean?
  - It’s actually an easy concept with a confusing name
  - *In the simplest terms, "hydrating" JavaScript means* ***taking the static HTML that was generated on the server and bringing it to life with interactive JavaScript from the client****.*
  - So like… how most modern websites work…
  - But, as we’ll see, each React Meta-framework has different capabilities with hydration
- **Streaming** – What does it mean
  - *In essence, "streaming" means the server doesn't wait for the entire page to be done before sending it. It sends what it has ready, and then continues to send the rest in chunks as they become available, making the user experience much quicker and more responsive.*

React Suspense

- Used to be an experimental
- Now used everywhere in newer framework versions
- Renders fallback loading UI when a child component throws a promise
- Makes Streaming easier



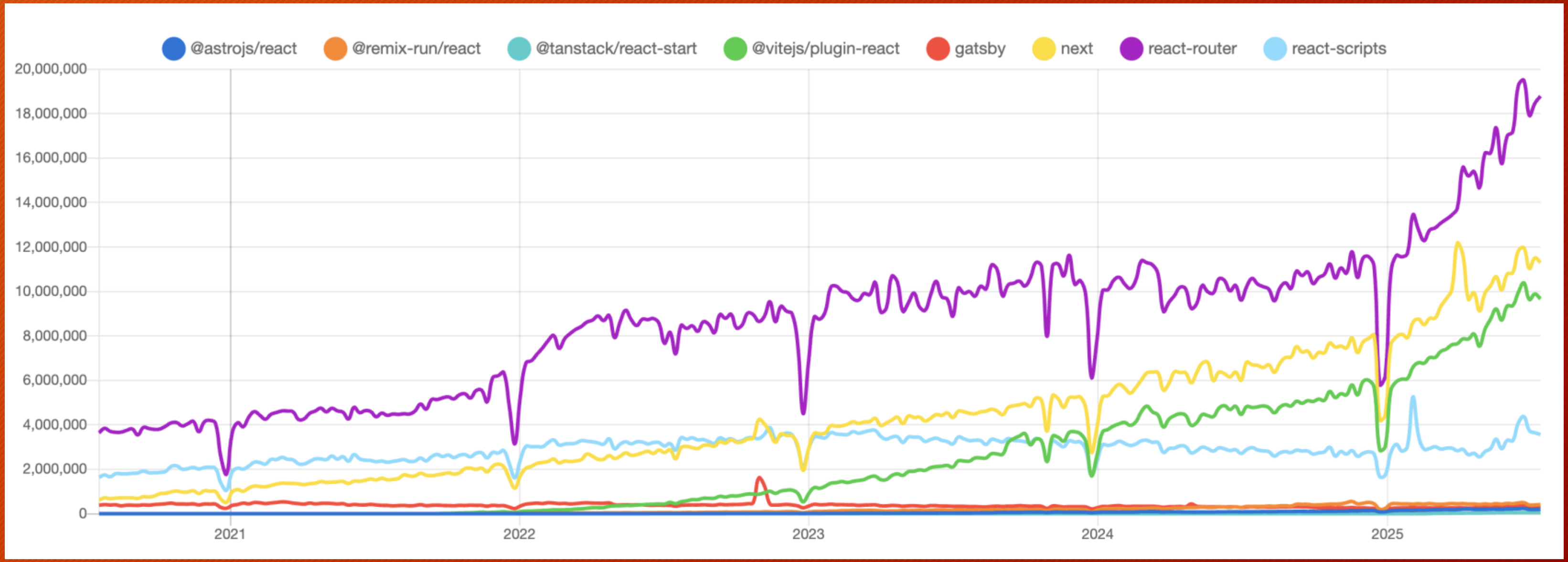
```
// --- The main App using Suspense ---
function App() {
  return (
    <div>
      <h1>My Awesome App</h1>

      <Suspense fallback={<p>Loading content...</p>}>
        <LazyLoadedContent />
      </Suspense>

      <p>This part is always visible.</p>
    </div>
  );
}
```

Top React Meta-Frameworks

- **React Router** (formerly also Remix and React Router)
- **TanStack Router/Start**
- **Next.js**
- **Astro**
- *Gatsby*
- *Redwood*
- *Blitz.js*
- *Waku*
- *Create React App*
- *Single-SPA*



Not To Be Confused With Bundlers

- Webpack – Everyone used to use Webpack, lots still do
- Turbopack – Next.js’s Webpack replacement
- RSPack – Webpack, but faster (Rust btw)
- Rollup
- Vite – Was Built on top of Rollup and ESBuild, soon Rolldown
- Parcel

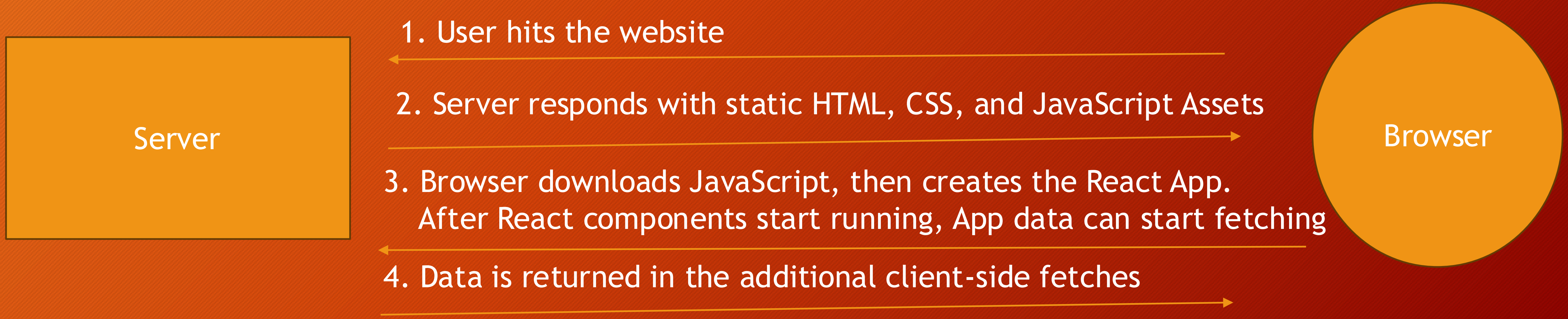
WHY ARE THERE SO MANY WAYS TO BUILD REACT APPS?

It’s All About Data

…for the most part

CSR – Client-Side Rendering (SPAs)

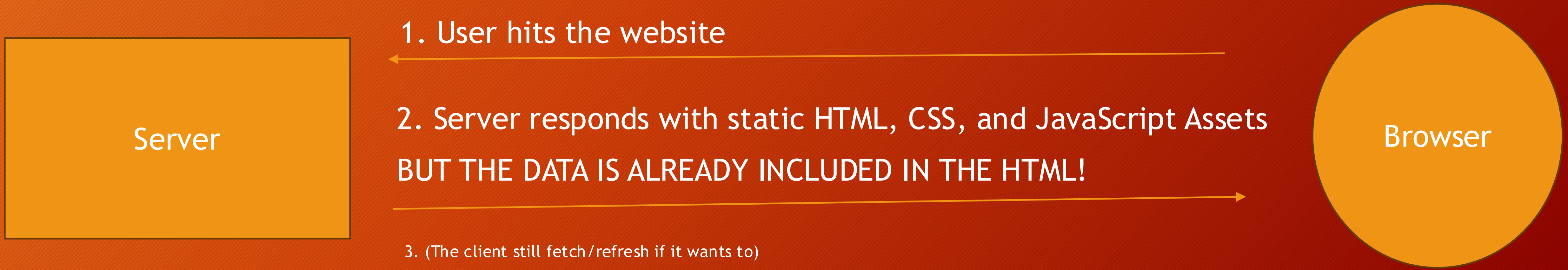
- Rendering and data fetching only happens on the Client
- A shell HTML file is delivered to the browser, and JavaScript renders the page
- Most React Apps (~80%?) are probably still built with only CSR
- Called Single Page Application because it is literally 1 page



1. User hits the website
2. Server responds with static HTML, CSS, and JavaScript Assets
3. Browser downloads JavaScript, then creates the React App.
   After React components start running, App data can start fetching
4. Data is returned in the additional client-side fetches

SSR – Server-Side Rendering

- Data is fetched at request time on the server
- Pages are rendered on the server into populated HTML
- Most realtime SEO friendly websites are built this way



1. User hits the website

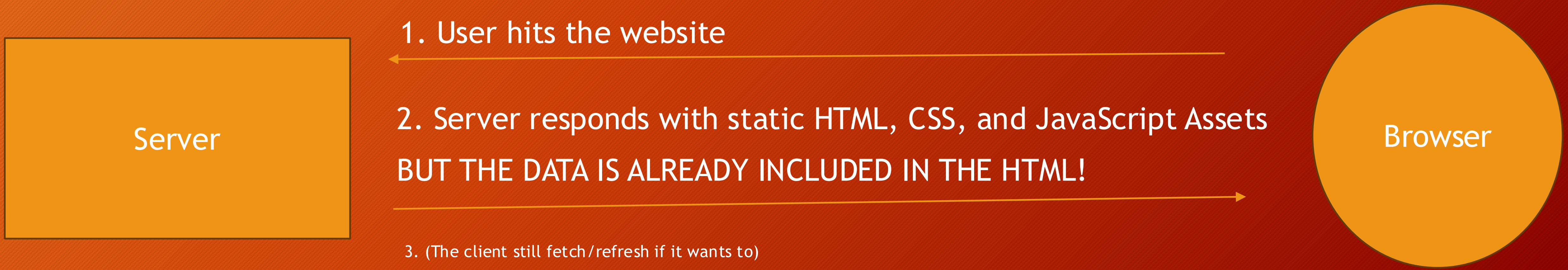
2. Server responds with static HTML, CSS, and JavaScript Assets

BUT THE DATA IS ALREADY INCLUDED IN THE HTML!

3. (The client still fetch/refresh if it wants to)

SSG – Static Site Generation

- Data is fetched at build time instead of request time
- Pages are rendered/built on the server (or CI/CD pipeline)
- Most static websites are built this way



1. User hits the website

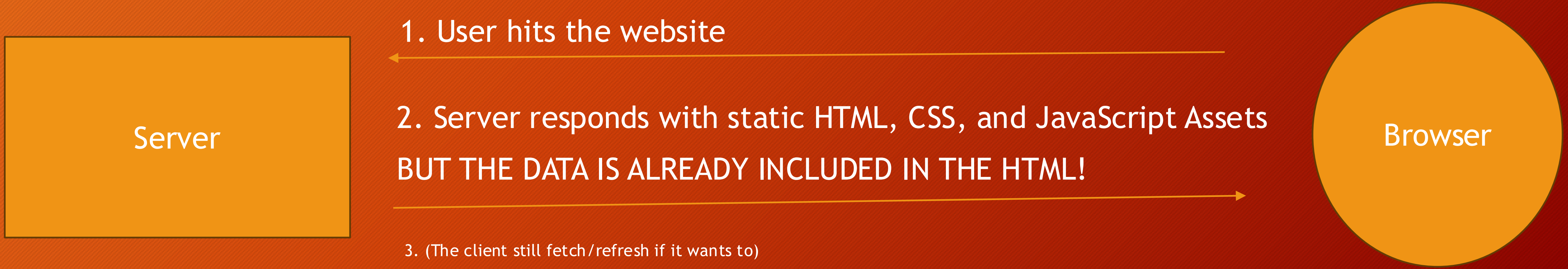
2. Server responds with static HTML, CSS, and JavaScript Assets

BUT THE DATA IS ALREADY INCLUDED IN THE HTML!

3. (The client still fetch/refresh if it wants to)

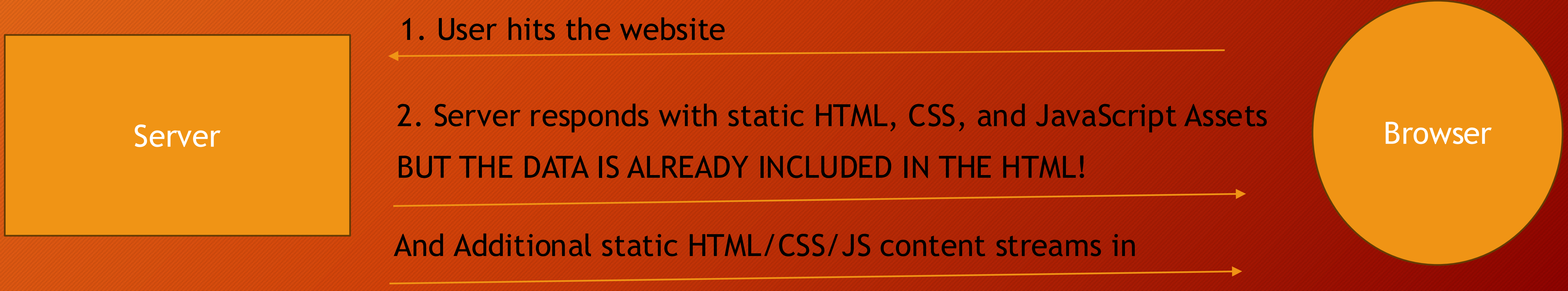
ISR – Incremental Static Regeneration

- Just like SSG, but with options to rebuild itself on the server when data is stale
- Frameworks like Next.js and Astro have gradually introduced these features


1. User hits the website
Server
2. Server responds with static HTML, CSS, and JavaScript Assets
BUT THE DATA IS ALREADY INCLUDED IN THE HTML!
Browser
3. (The client still fetch/refresh if it wants to)

RSCs – React Server Components (Streaming)

- What are RSCs? – React Server Components
- React Components that ONLY FETCH ON THE SERVER (And only run on the server in general)
- Fetch on a per server component basis instead of being limited to fetching per page/route
- Results in a reduced bundle size, because RSCs don't run any* client-side JavaScript
- New HTML can stream in (instead of JSON fetches)


Server
1. User hits the website
2. Server responds with static HTML, CSS, and JavaScript Assets
BUT THE DATA IS ALREADY INCLUDED IN THE HTML!
And Additional static HTML/CSS/JS content streams in
Browser

Why use React Query?

- Automatic Data State Management for all GET requests
- Automatic Loading and Reloading states and logic
- Automatic Error Handling and error handling states
- Dedicated Mutation logic (POST, UPDATE, DELETE, PATCH requests)
- CACHING! (and everything to do with caching – state time, gc/cache time, cache invalidation/busting)
- PREFETCHING!
- Polling and Re-fetching features
- Pagination and Infinite Scrolling features
- Better memoization and re-rendering optimizations via structural sharing
- Can work with SSR/SSG
- Can work with RSCs
- Offline features
- Persistence to session or local storage (or IndexedDB or SQLite)
- Awesome Devtools
- Comparison | React Query vs SWR vs Apollo vs RTK Query vs React Router | TanStack Query React Docs
- @reduxjs/toolkit vs @tanstack/query-core vs react-query vs react-relay vs swr | npm trends