

---

# SERVER COMPONENTS

REACT & NEXT.JS OFFICIAL DOCS STUDY

DongMin Kim

---

# CONTENTS

- **History of Web Development**
  - Static HTML
  - SSR
  - AJAX
  - CSR & SPA
  - Hybrid Rendering
    - Page Router
    - App Router
- **Official Docs**
  - Server Components
  - Server Actions
  - Directives
- **Recap**
- **QnA & Discussion**
- **Retrospect**
- **Interim Summary**

# GOALS

History of web development

CSR, SSR, SSG, ISR, Hybrid Rendering

What is Server Components?

Why Server Components?

# History of Web Development

The evolution of web technology

RFC 1945 - Hypertext Transfer Protocol -- HTTP/1.0

Network Working Group  
Request for Comments: 1945  
Category: Informational

T. Berners-Lee  
MIT/LCS  
R. Fielding  
UC Irvine  
H. Frystyk  
MIT/LCS  
May 1996

**Hypertext Transfer Protocol -- HTTP/1.0**

**Status of This Memo**

This memo provides information for the Internet community. This memo does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

**IESG Note:**

The IESG has concerns about this protocol, and expects this document to be replaced relatively soon by a standards track document.

**Abstract**

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. It is a generic, stateless, object-oriented protocol which can be used for many tasks, such as name servers and distributed object management systems, through extension of its request methods (commands). A feature of HTTP is the typing of data representation, allowing systems to be built independently of the data being transferred.

HTTP has been in use by the World-Wide Web global information initiative since 1990. This specification reflects common usage of the protocol referred to as "HTTP/1.0".

**Table of Contents**

1. Introduction .....	4
1.1 Purpose .....	4
1.2 Terminology .....	4
1.3 Overall Operation .....	6

**DataTracker**

**RFC 1945**  
**RFC - Informational**

**Document type**  
**RFC - Informational**  
May 1996

**Select version**  
04 RFC 1945

**Compare versions**  
draft-ietf-http-v10-spec-04  
RFC 1945

**Authors**  
Henrik Nielsen, Roy T. Fielding, Tim Berners-Lee

**RFC stream**  
IETF

**Other formats**  
txt html pdf w/errata bibtex

**Additional resources**  
Mailing list discussion

## World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#) , [Policy](#) , November's [W3 news](#) , [Frequently Asked Questions](#) .

### [What's out there?](#)

Pointers to the world's online information, [subjects](#) , [W3 servers](#), etc.

### [Help](#)

on the browser you are using

### [Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#) ,[X11 Viola](#) , [NeXTStep](#) , [Servers](#) , [Tools](#) , [Mail robot](#) , [Library](#) )

### [Technical](#)

Details of protocols, formats, program internals etc

### [Bibliography](#)

Paper documentation on W3 and references.

### [People](#)

A list of some people involved in the project.

### [History](#)

A summary of the history of the project.

### [How can I help ?](#)

If you would like to support the web..

### [Getting code](#)

Getting the code by [anonymous FTP](#) , etc.

<https://info.cern.ch/hypertext/WWW/TheProject.html>

# Static HTML

The Revolution of Information Delivery

# Static HTML

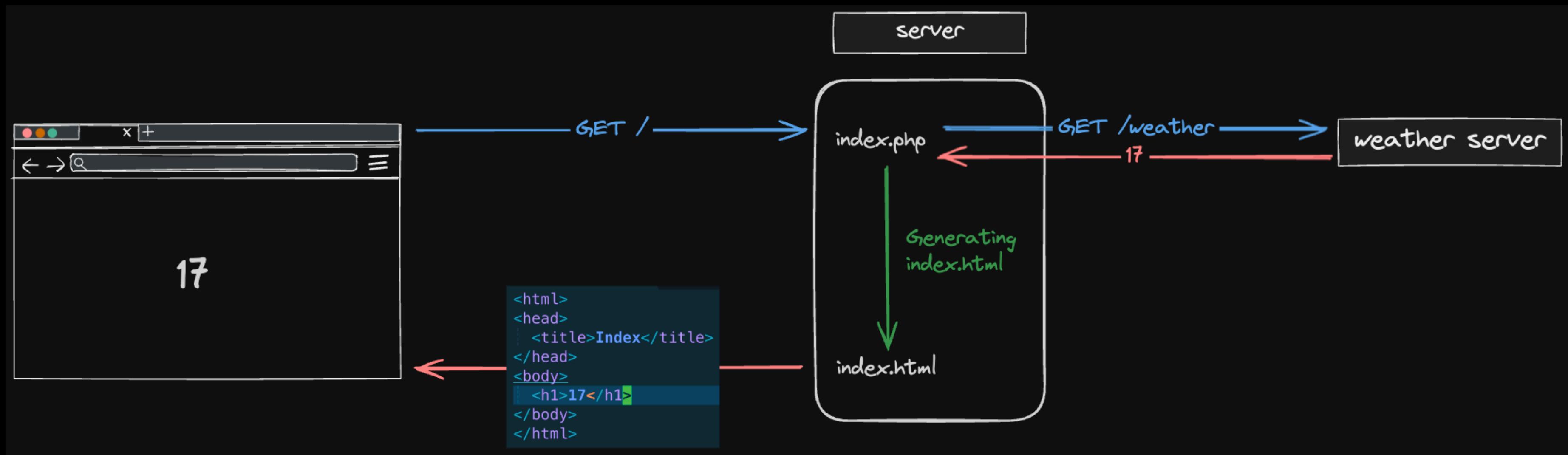


```
<%@ page contentType="text/html; charset=UTF-8" language="java" %>
<!DOCTYPE html>
<html>
<head>
    <title>JSP 예제</title>
</head>
<body>
    <h2>이름을 입력하세요:</h2>
    <form action="hello.jsp" method="post">
        <label for="name">이름: </label>
        <input type="text" id="name" name="name" required>
        <button type="submit">전송</button>
    </form>
    <%
        String name = request.getParameter("name");
        if (name != null && !name.trim().isEmpty()) {
    %>
        <h3>안녕하세요, <%= name %>님!</h3>
    <%
        }
    %>
</body>
</html>
```

# SSR (Server Side Rendering)

Dynamic HTML Generation

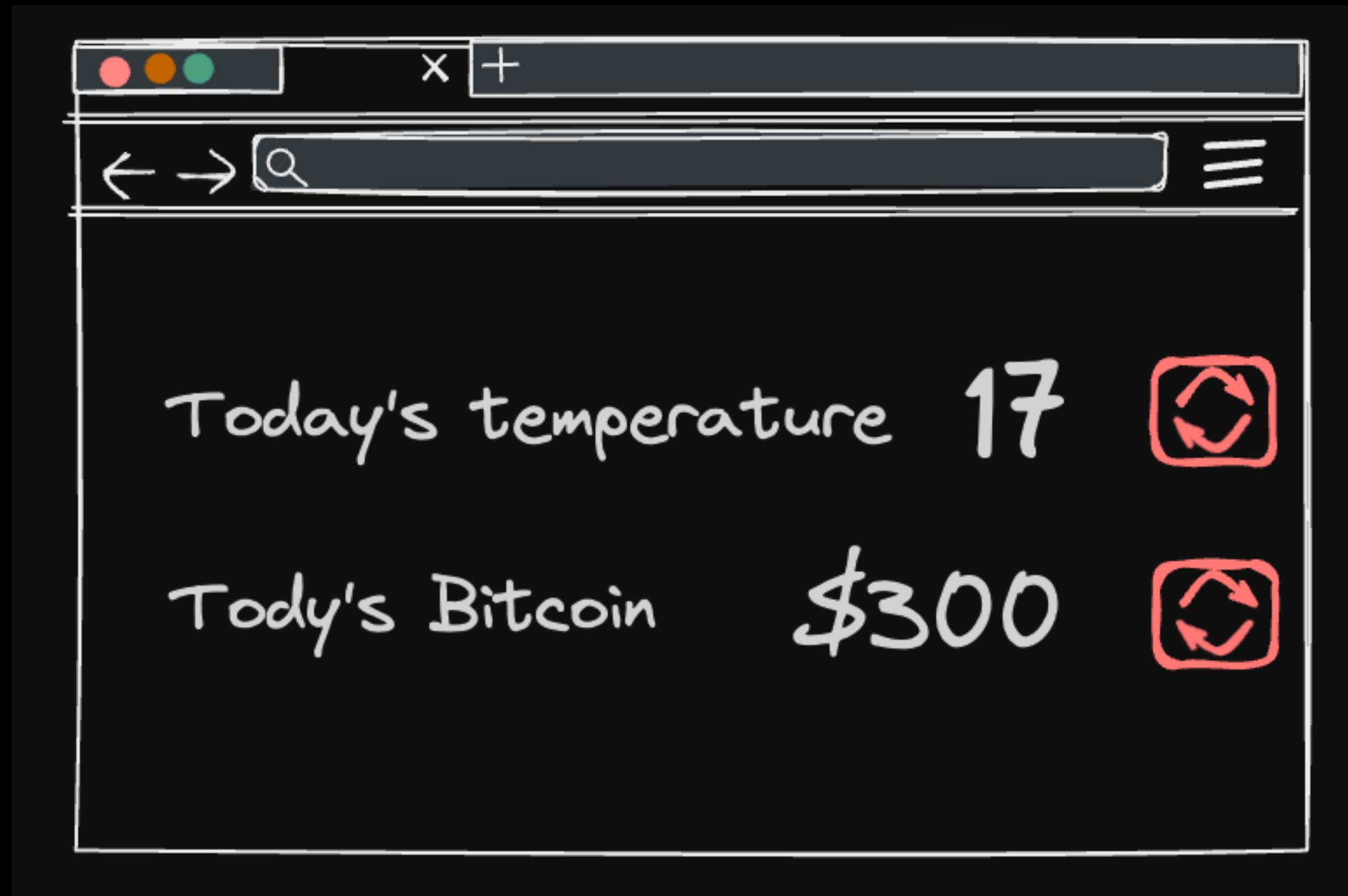
# SSR(Server Side Rendering)

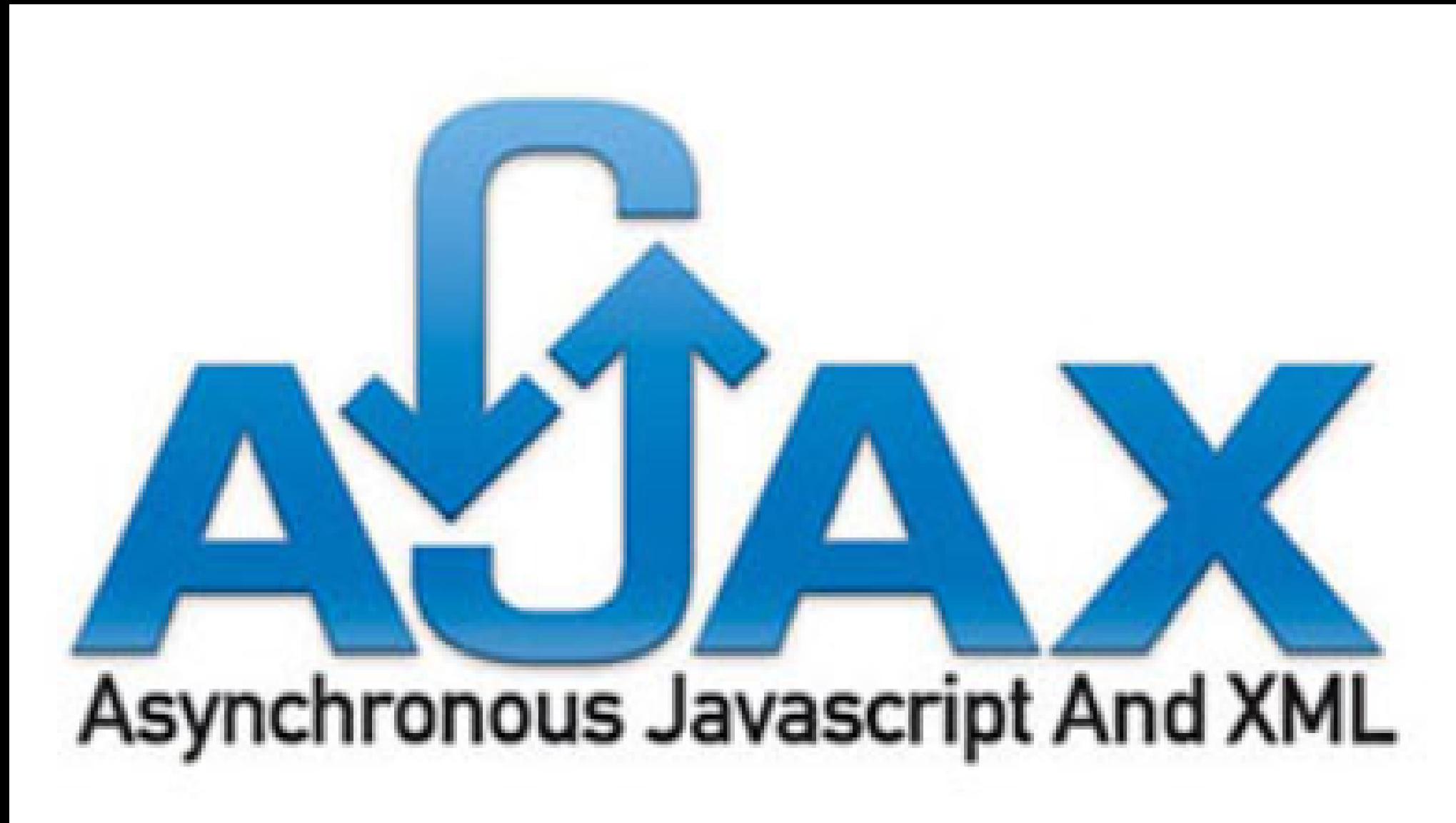


# SSR(Server Side Rendering)



# SSR(Server Side Rendering)





**AJAX**

Asynchronous JavaScript and XML

---

---

# AJAX - Practice

[https://github.com/gdsc-konkuk/24-25-study-react-nextjs-docs/tree/main/11\\_01\\_ServerComponents/AJAX\\_Practice](https://github.com/gdsc-konkuk/24-25-study-react-nextjs-docs/tree/main/11_01_ServerComponents/AJAX_Practice)

---

# AJAX

The screenshot shows the classic Gmail interface from 2004. At the top, there's a search bar, a 'Search Mail' button, and a 'Search the Web' button. Below the search bar are links for 'Show search options' and 'Create a filter'. The main navigation bar includes 'Compose Mail', 'Back to Inbox' (which is currently selected), 'Archive', and 'More actions...'. A status message '1 of 25 Old' is visible on the right.

**Inbox**

Starred ★ Sent Mail All Mail Spam Trash

**Labels**

- quir
- Orkut
- Prius (18)

[Edit labels](#)

**Screenshot** [Inbox](#) [Apply label...](#)

**Jason Shellen** Kevin, Please send me screenshots... 3:31pm (7 minutes ago)

**Kevin Fox** to Jason More options 3:37pm (1 minute ago)

Don't worry, I'm sure nobody'll find out that you're using work resources to serve these images from [shellen.com](#). Just in case they cut you off though, I'll post the screenshots on my blog at [fury.com](#) as well.

Seeya later,  
Kevin  
[fury.com](#)

- Show quoted text -

[Reply](#) [Forward](#)

**Back to Inbox** Archive More actions... [1 of 25 Old](#)

You are currently using 0 MB (0%) of your 1000 MB.

Shortcuts: o-open y-archive c-compose j-older k-newer [more»](#)

[Terms of Use](#) - [Privacy Policy](#) - [Program Policies](#) - [Google Home](#)

©2004 Google

[Open in new window](#)

[Print conversation](#)

[Expand all](#)

**Related Pages**

[Mac OS X Hosting - Apache Hosting - Macintosh Web Site Hosting...](#)

Offers affordable Mac OS Apache web server unix hosting - BSD unix ...  
[www.inno-tech.com](#)

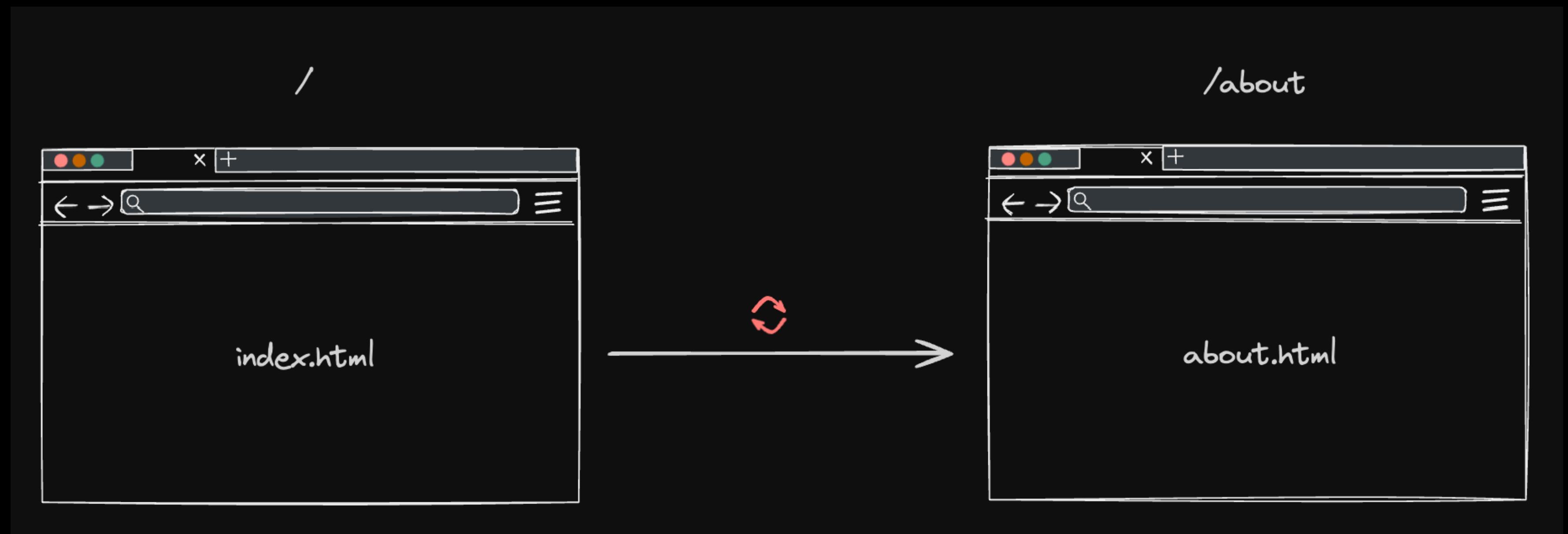
[Rivercom - Web Hosting - Web Development - Multimedia Development...](#)

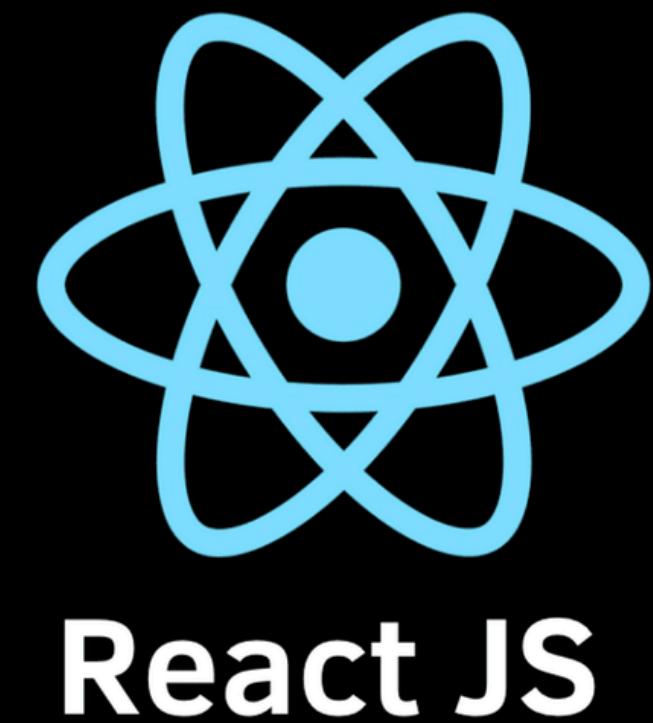
We combine original graphic design, intuitive navigation controls, ...  
[www.rivercom.com](#)

[About these links](#)

A screenshot of the Google Maps homepage. The top navigation bar includes links for Web, Images, Maps, News, Shopping, Gmail, and more. The user's email address, brownb6483@gmail.com, is displayed along with links for My Profile, Saved Locations, Help, Web History, My Account, and Sign out. The main header features the "Google Maps" logo and a search bar with placeholder text "e.g. '10 market st, san francisco' or 'hotels near lax'". Below the search bar are buttons for "Search the map", "Find businesses", and "Get directions". A secondary navigation bar below the main one shows "Search Results" and "My Maps". To the right of the map are links for Print, Send, and Link to this page. The central feature is a map of the United States and surrounding regions, including Canada, Mexico, and parts of Central America. The map is overlaid with state and province boundaries and names. Various controls are visible on the left side of the map area, including a vertical zoom slider, a compass rose, and a scale bar indicating 1000 m. The bottom of the page contains copyright information: "© 1996-2008 Google | 1996-2008 Google - Map data ©2008 NAVTEQ™, Europe Technologies - Terms of Use".

# AJAX





# CSR & SPA

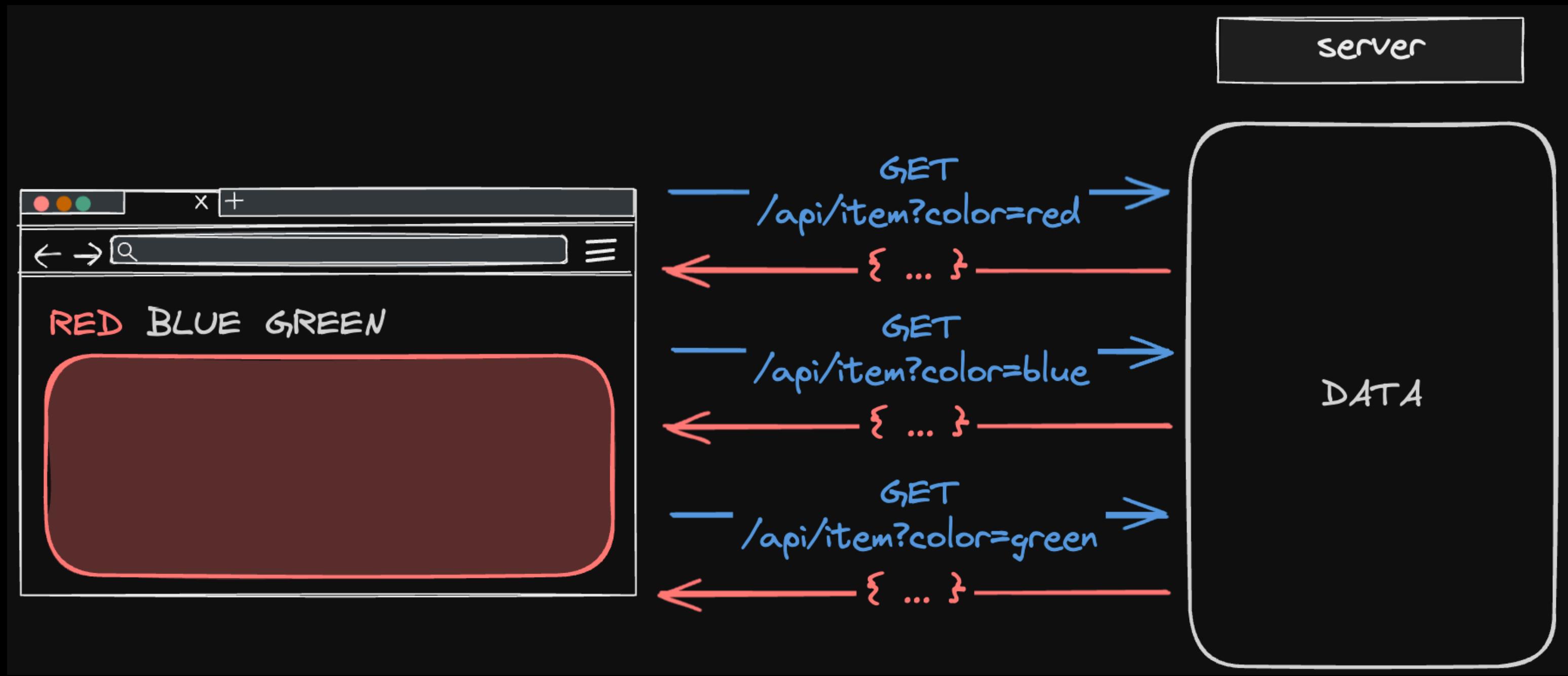
Client Side Rendering  
Single Page Application

---

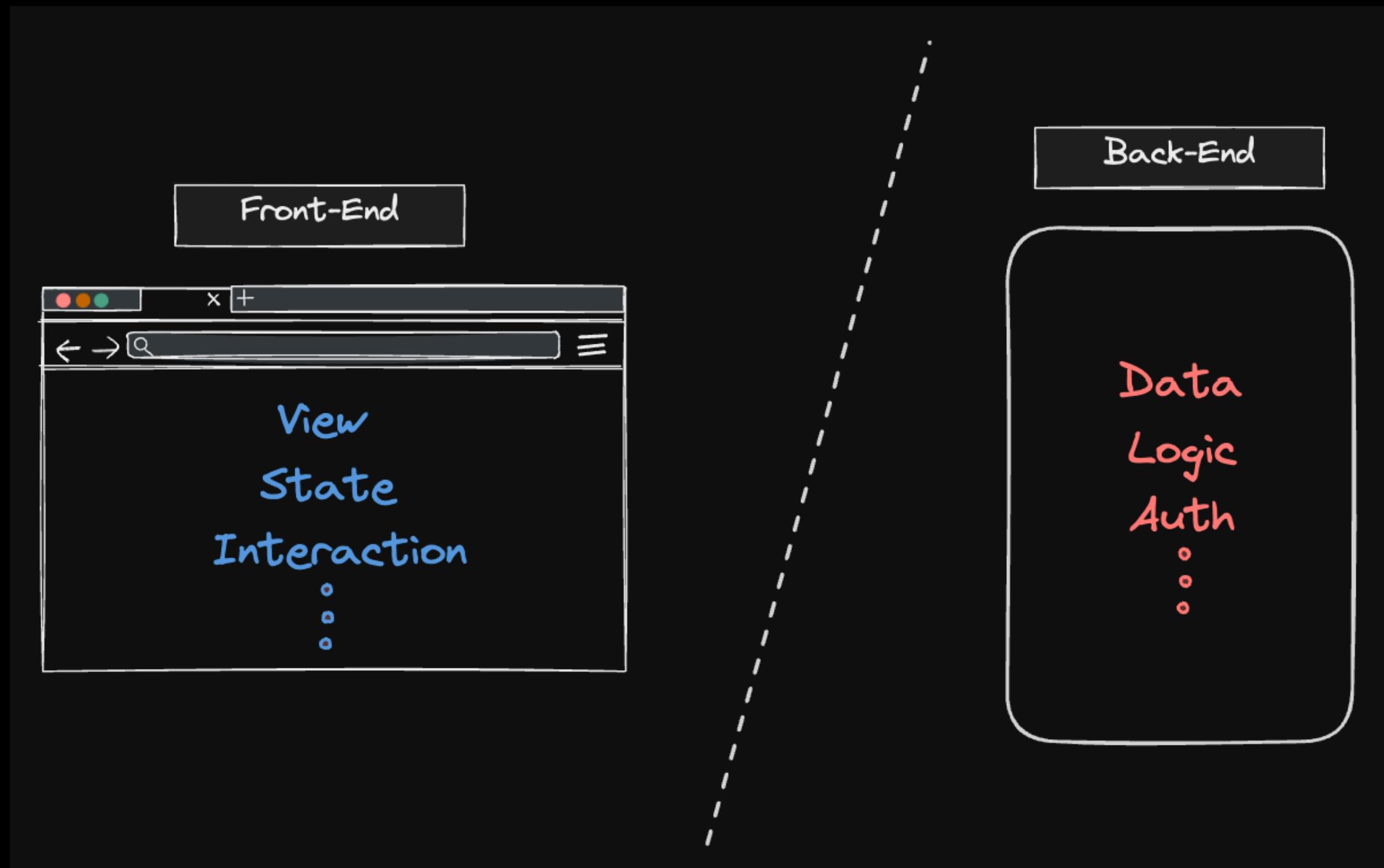
# CSR & SPA



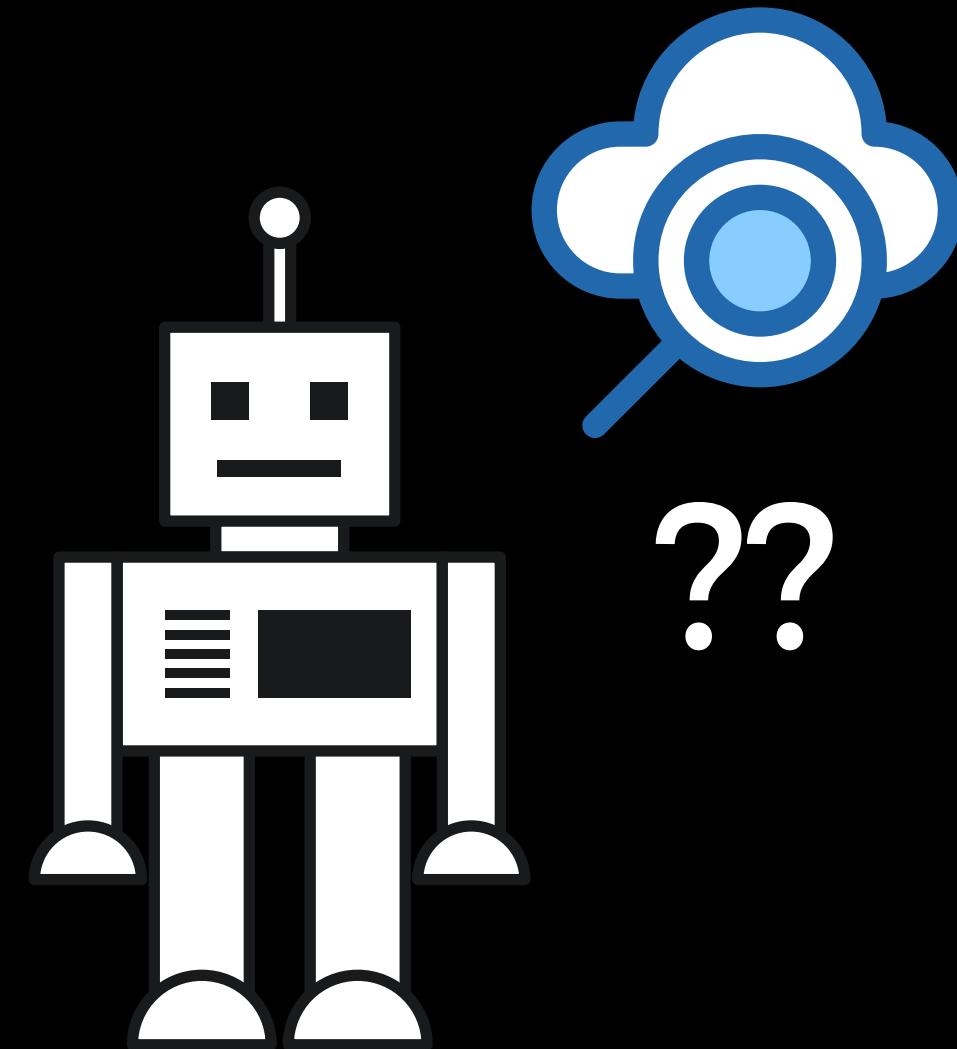
# CSR & SPA



# CSR & SPA



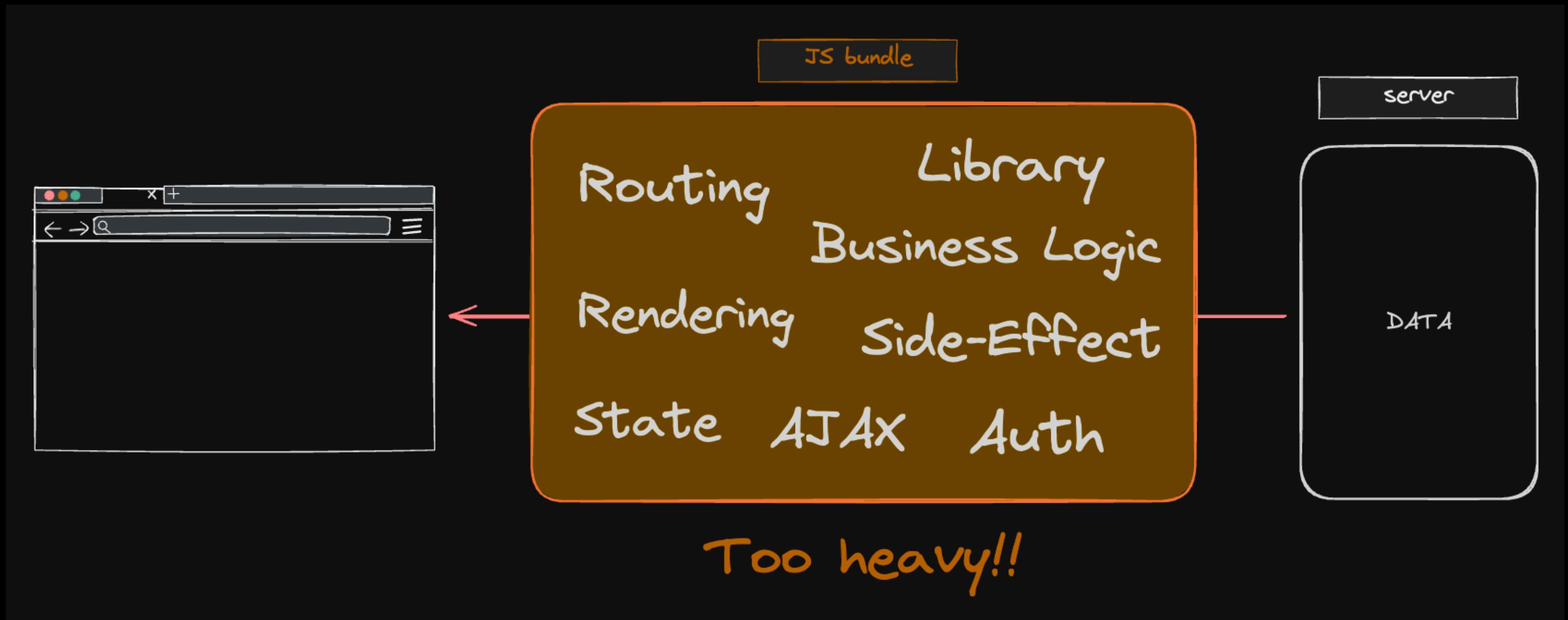
# CSR & SPA



??

```
<html>
  <head>
    <title>Index</title>
  </head>
  <body>
    <div id="root"></div>
  </body>
</html>
```

# CSR & SPA



# CSR & SPA

```
export default function Page() {
  const [state, setState] = useState(0);

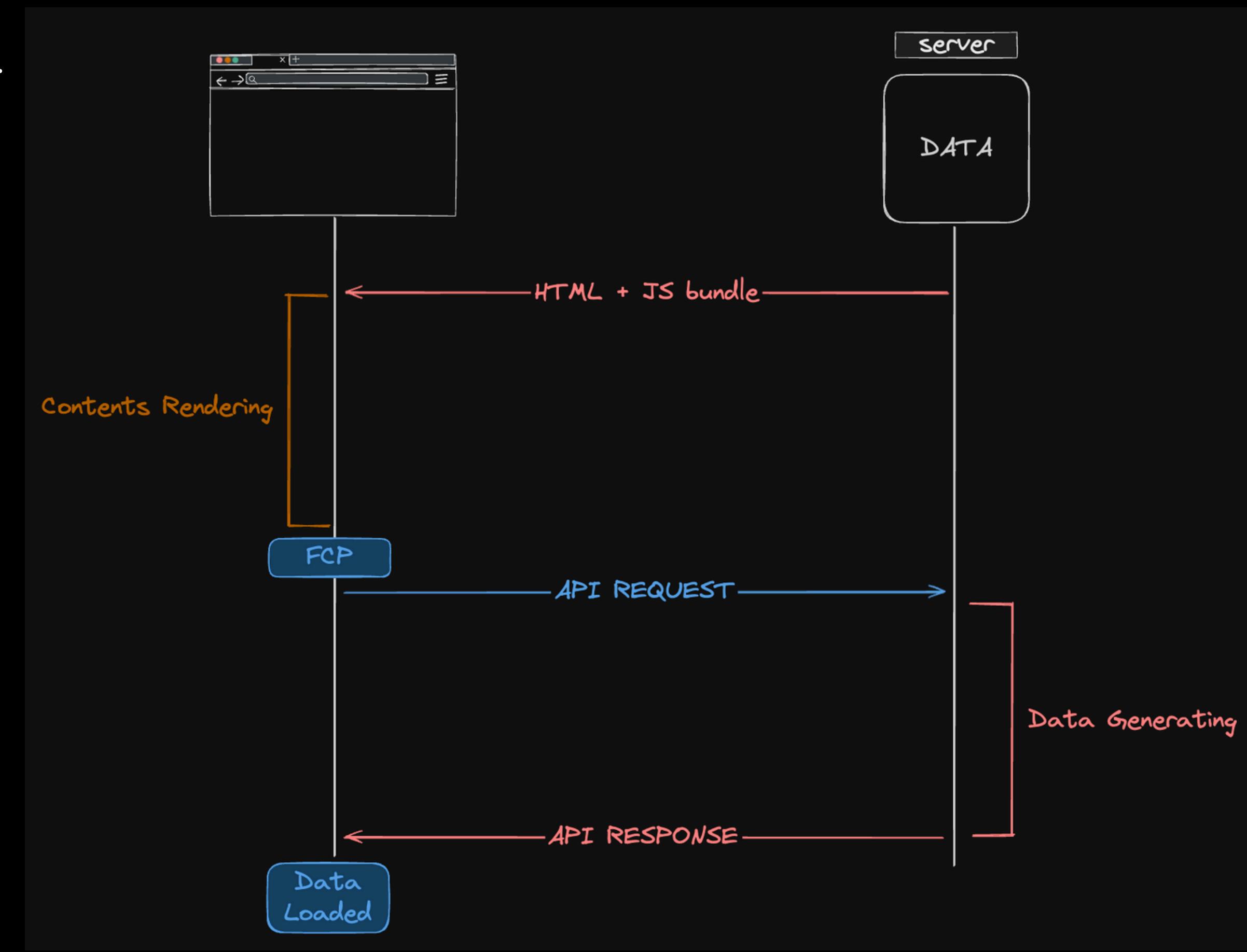
  const fetchData = async () => {
    const response = await fetch("...");
    const data = await response.json();

    setState(data);
  };

  useEffect(() => {
    fetchData();
  }, []);

  if (!state) return "Loading...";

  return <div>...</div>;
}
```





The Next.js logo is displayed in white against a black background. The text "NEXT" is in a bold, sans-serif font, with a large "X" where the two "E"s would be. To its right, ".JS" is written in a smaller, regular sans-serif font.

# Hybrid Rendering

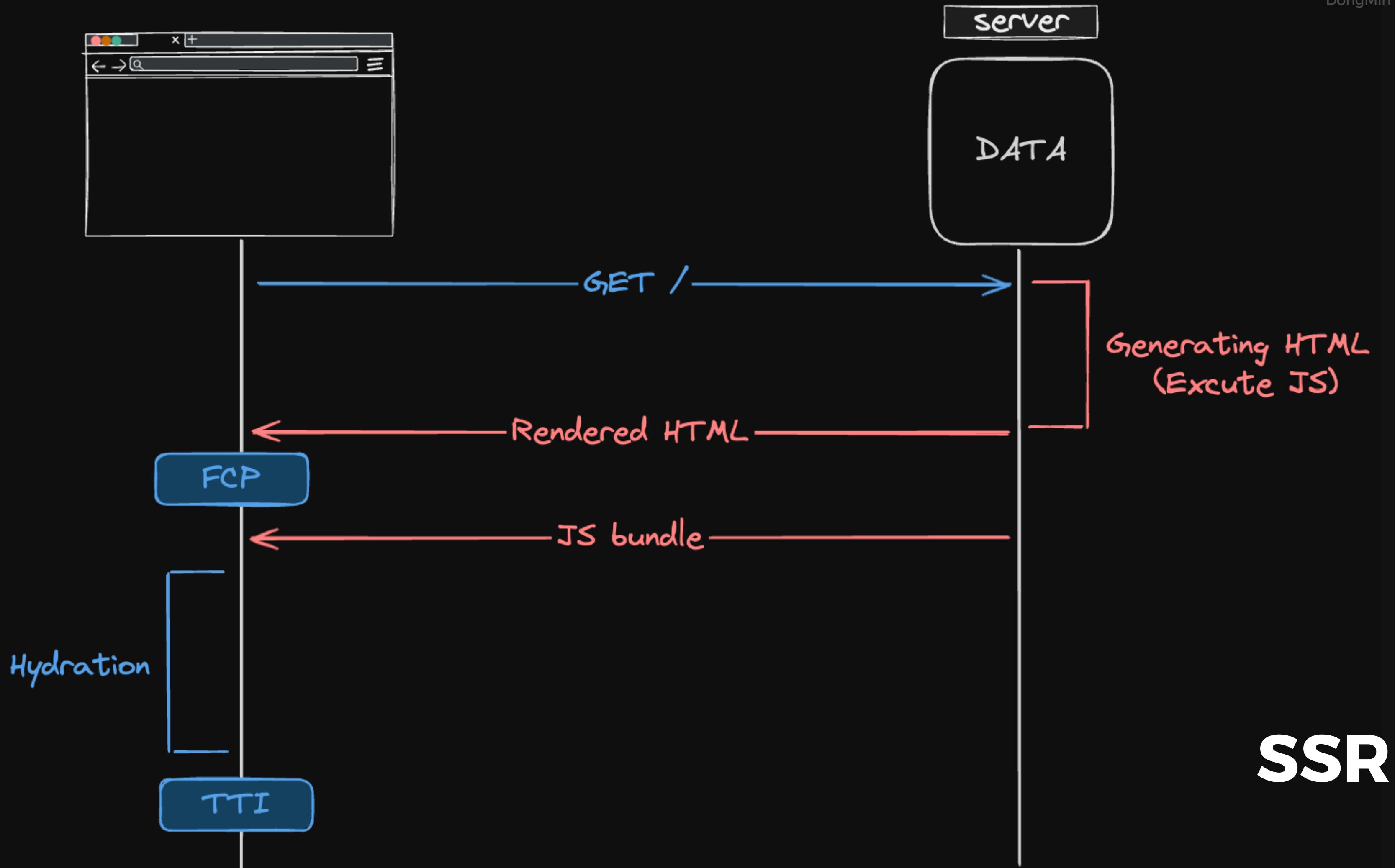
Revival of SSR

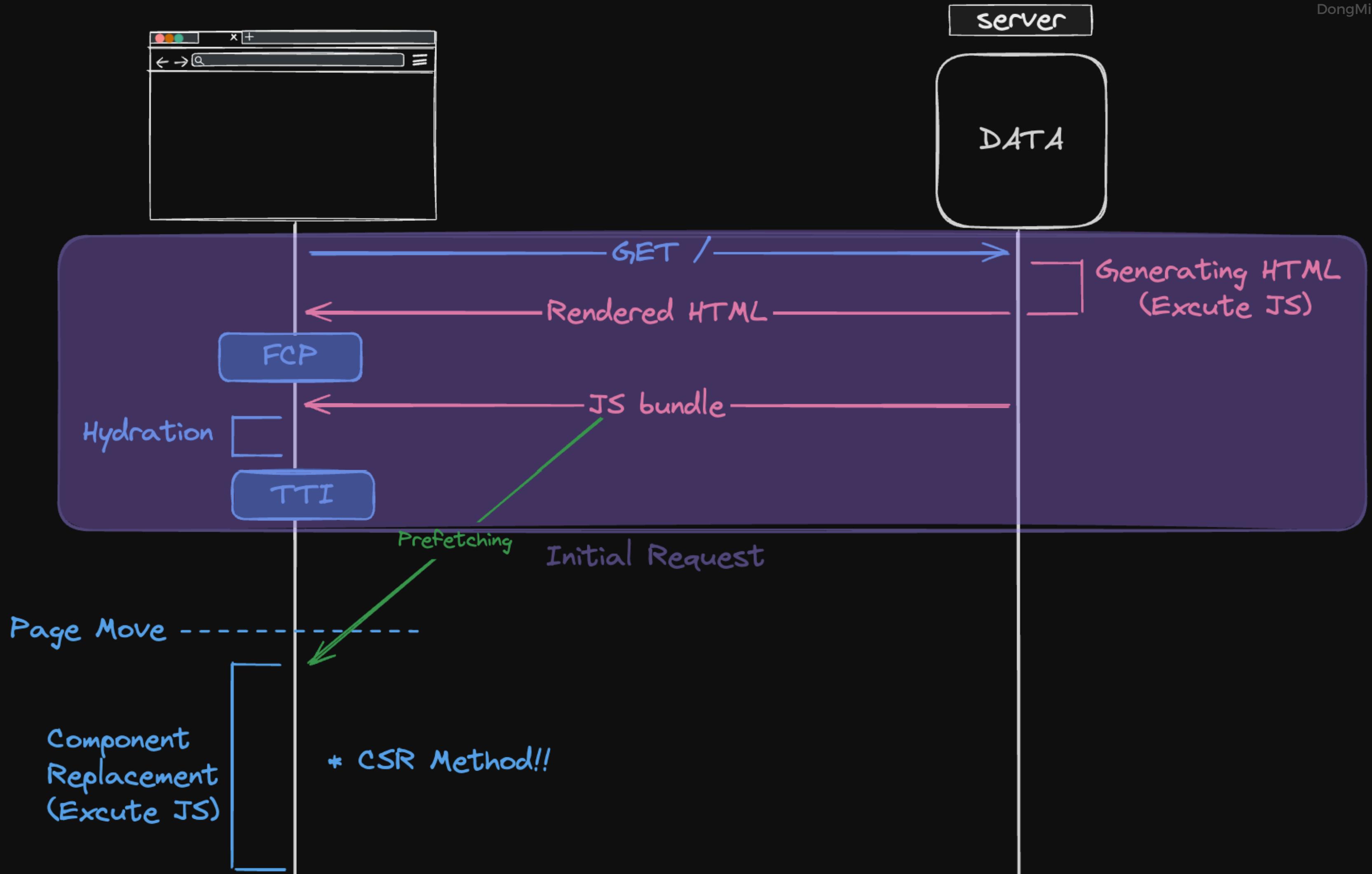


# Page Router

Fast FCP + CSR

---





# Page Router

Pre-rendering

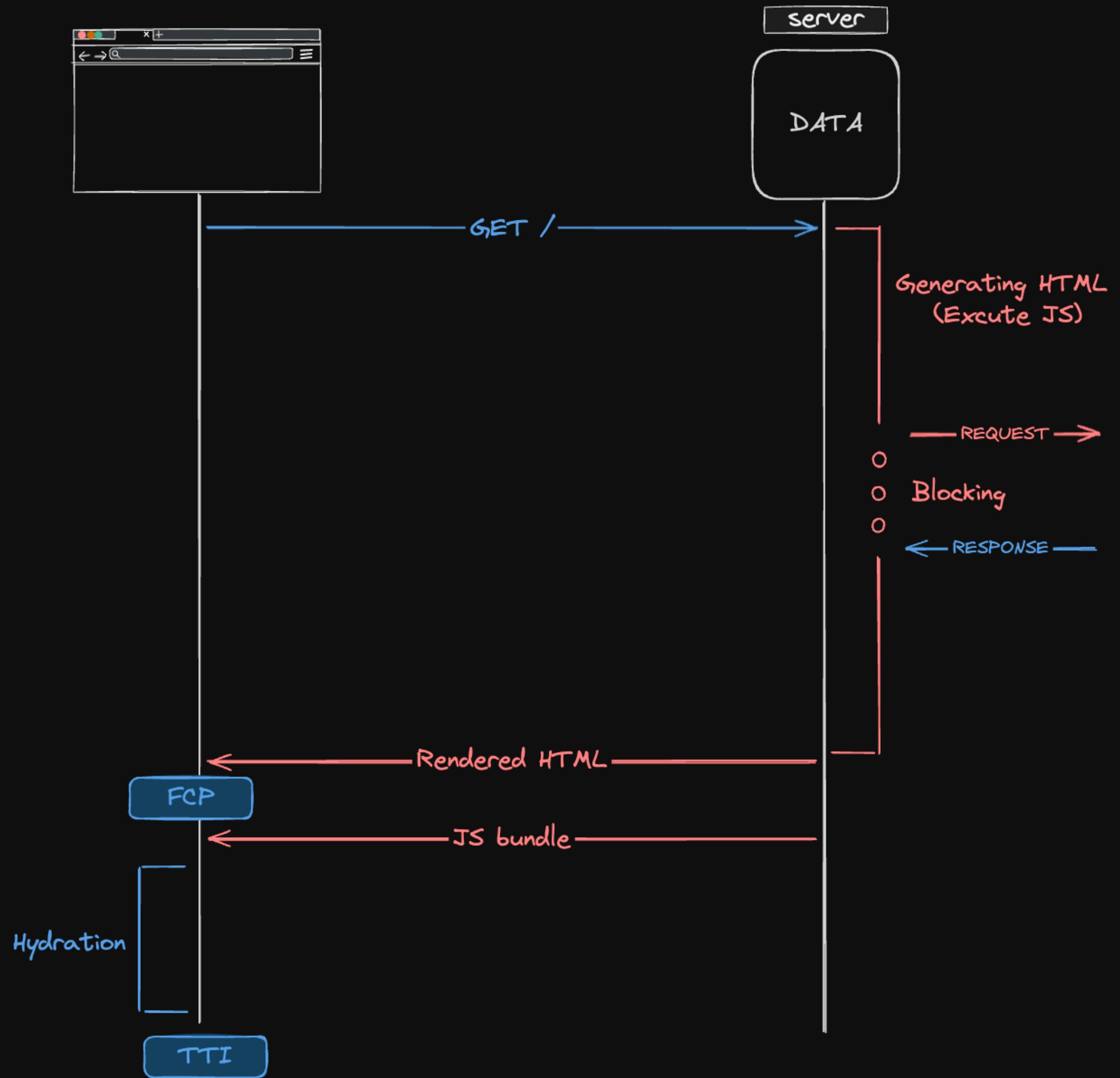
Overcome Disadvantage

Fast FCP

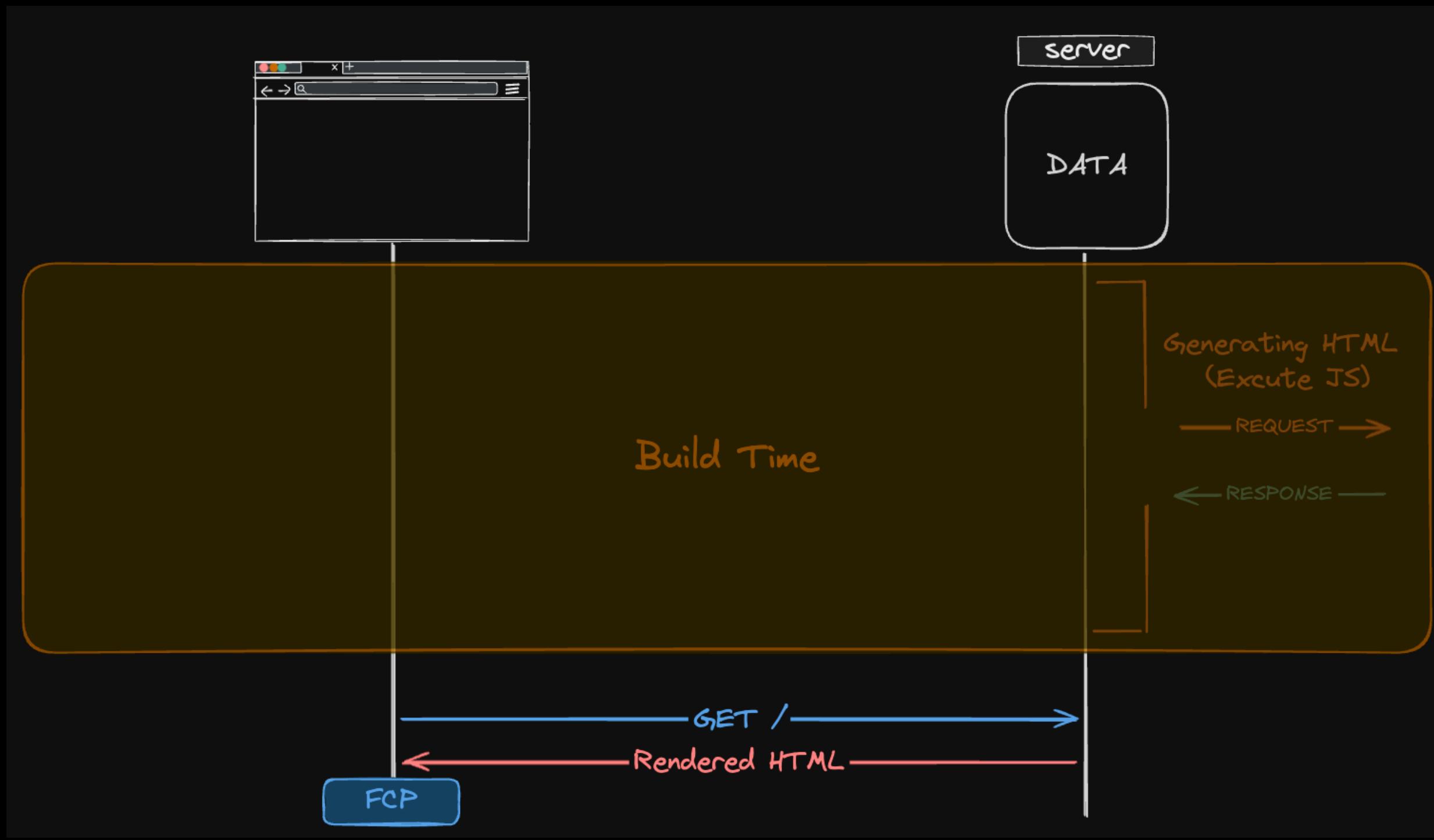


Inherit Advantage

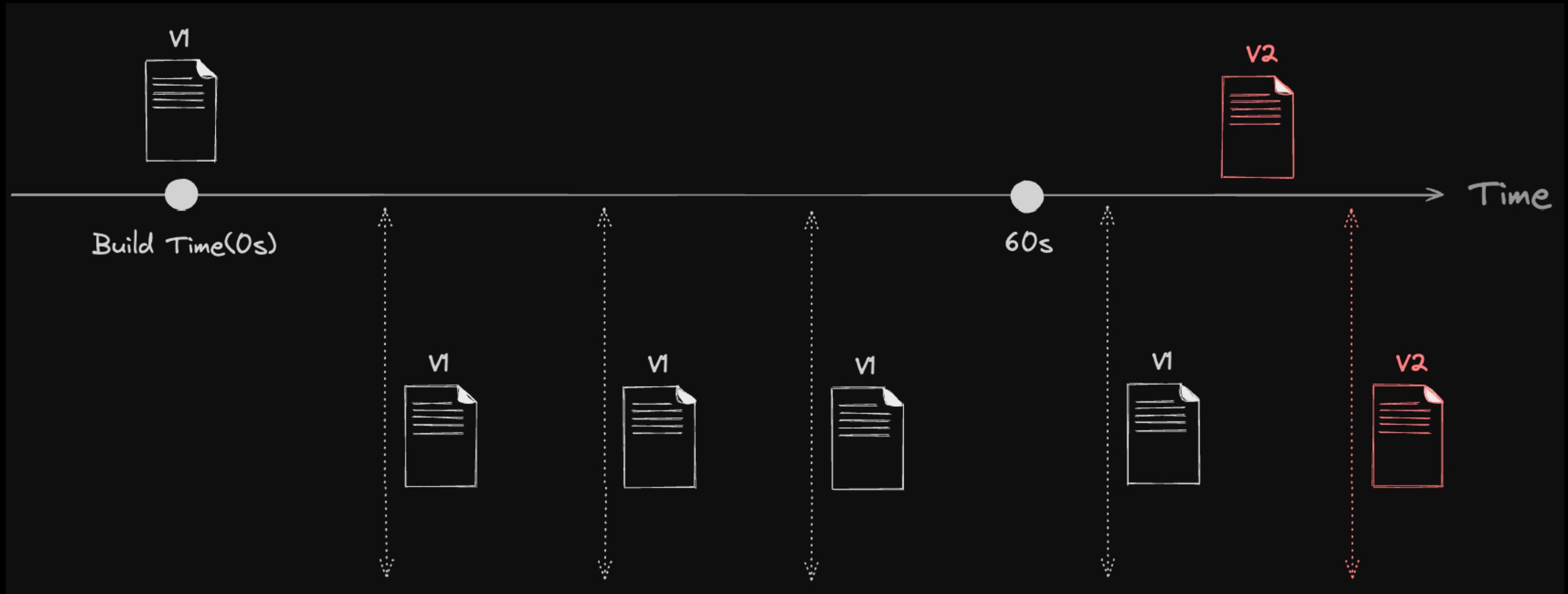
Fast Page Move



# SSG(Static Site Generation)



# ISR(Incremental Static Regeneration)



# Page Router

SSR

CSR

SSG

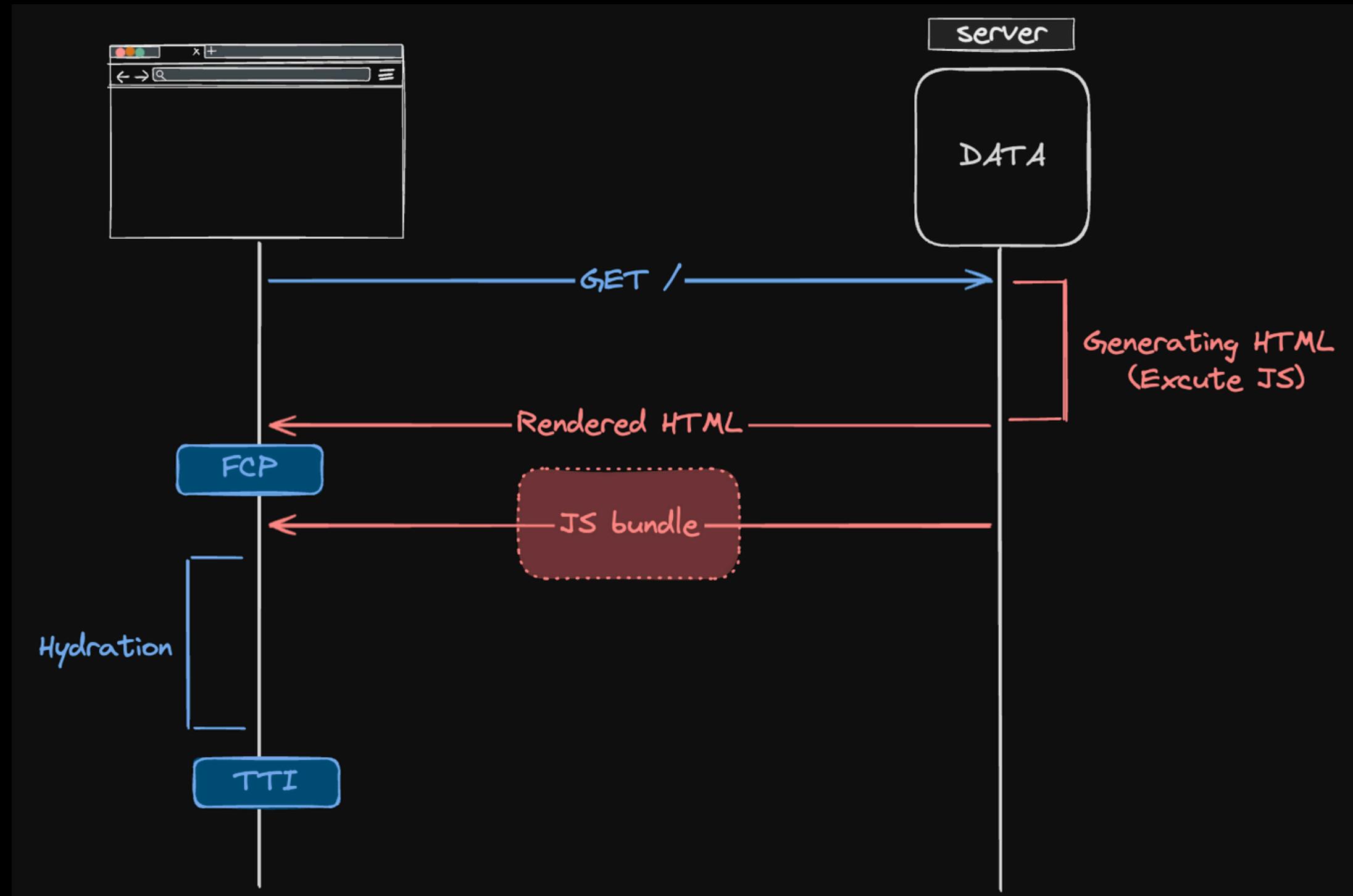
ISR



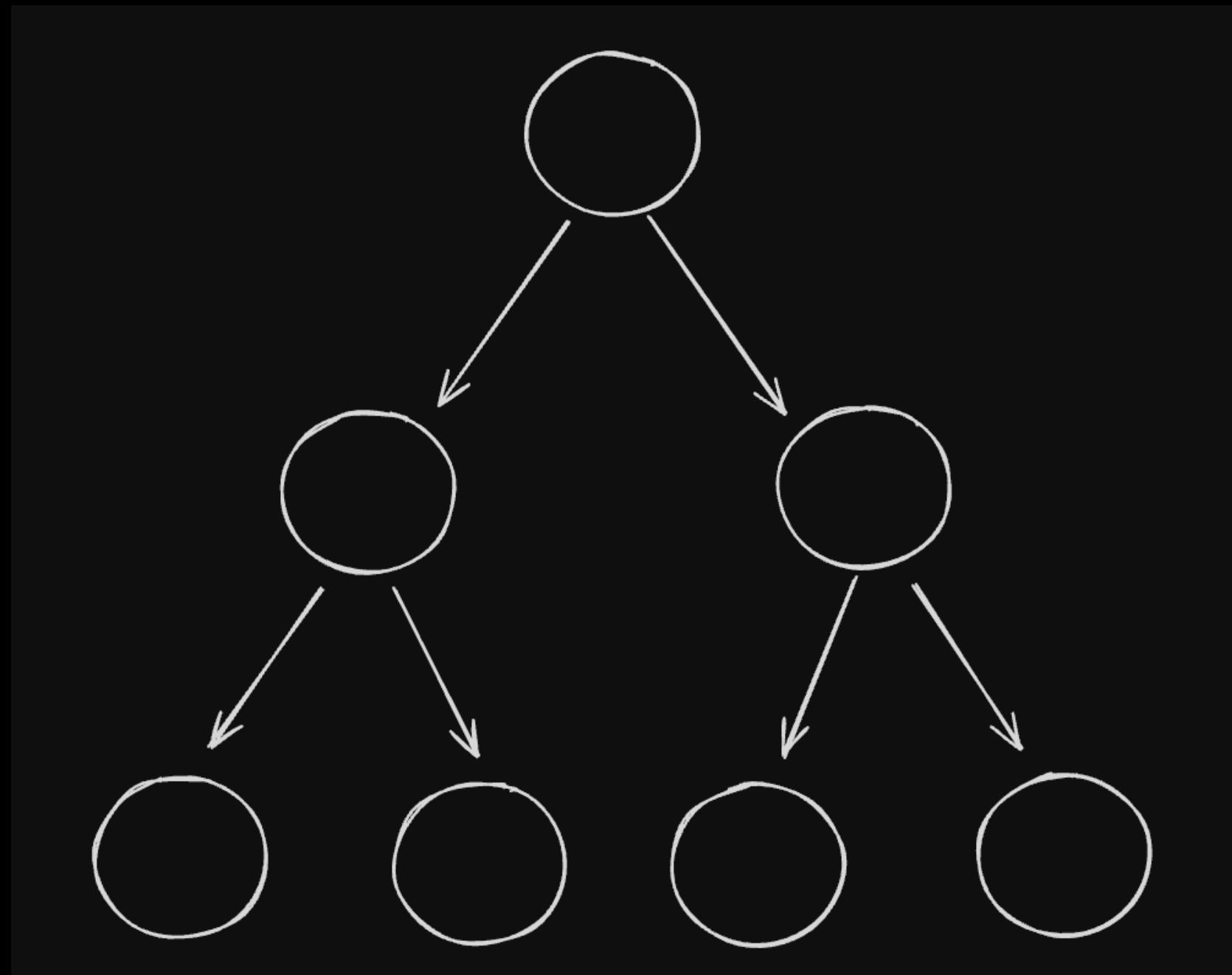
# App Router

Server Components

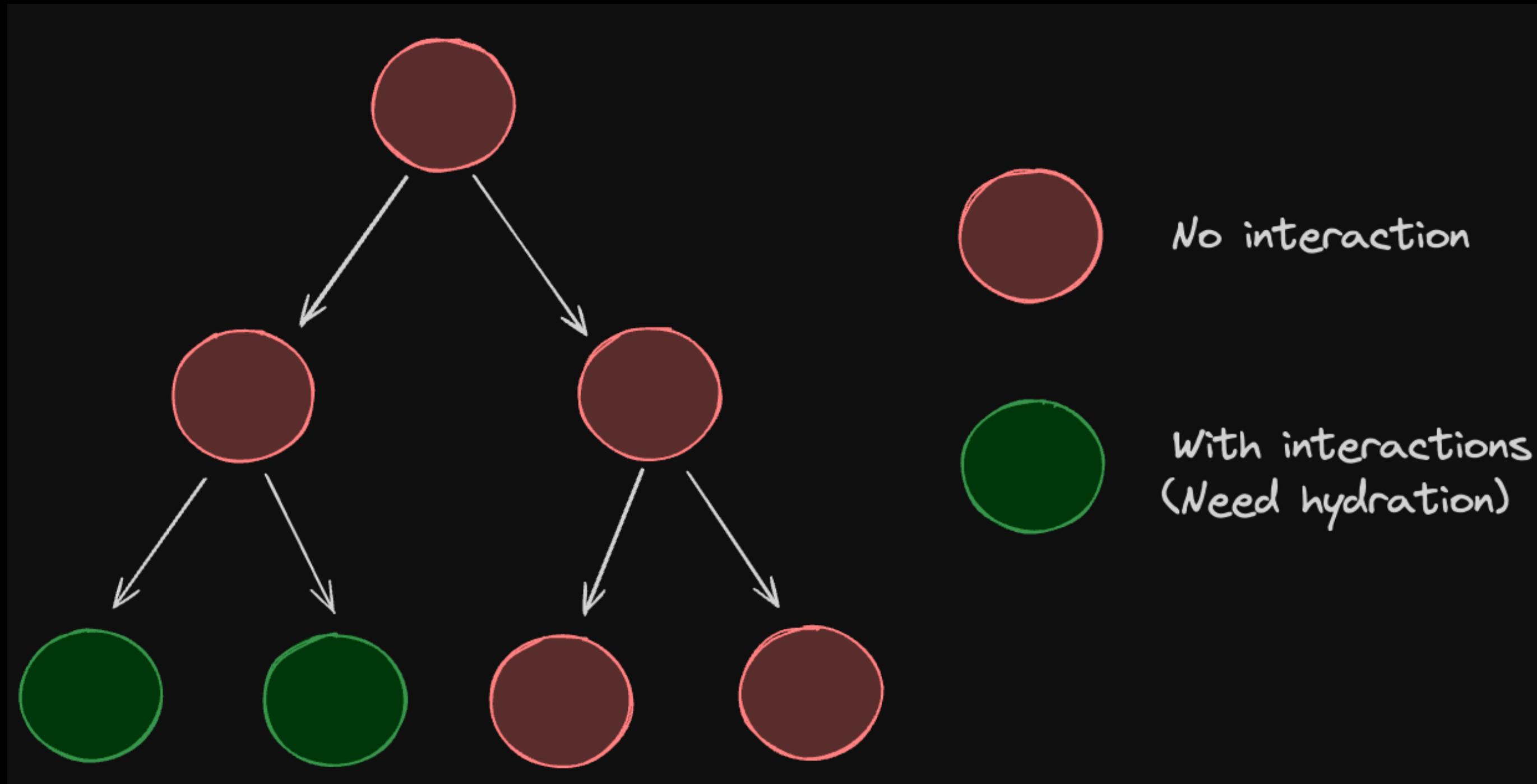
# Page Router



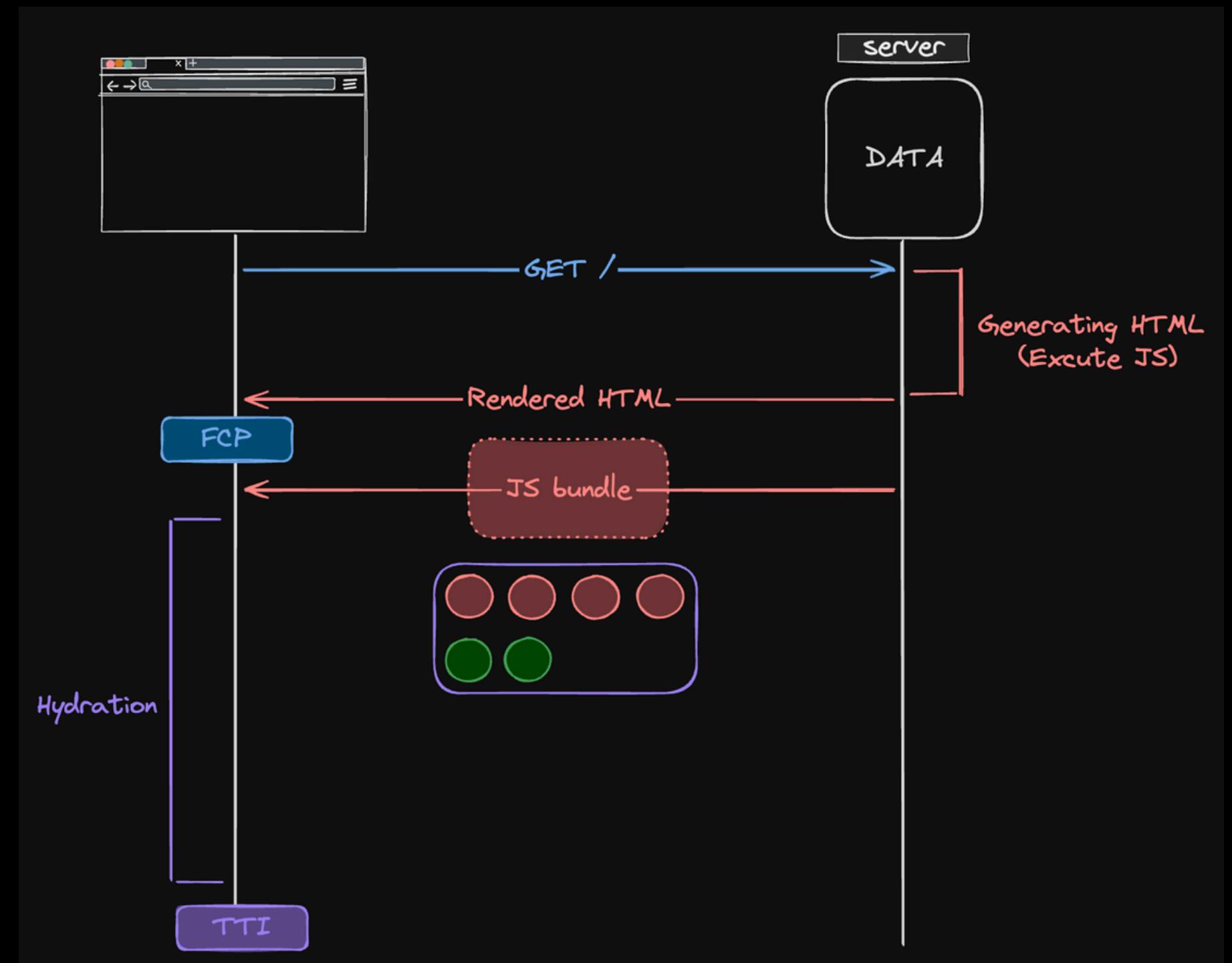
# Page Router



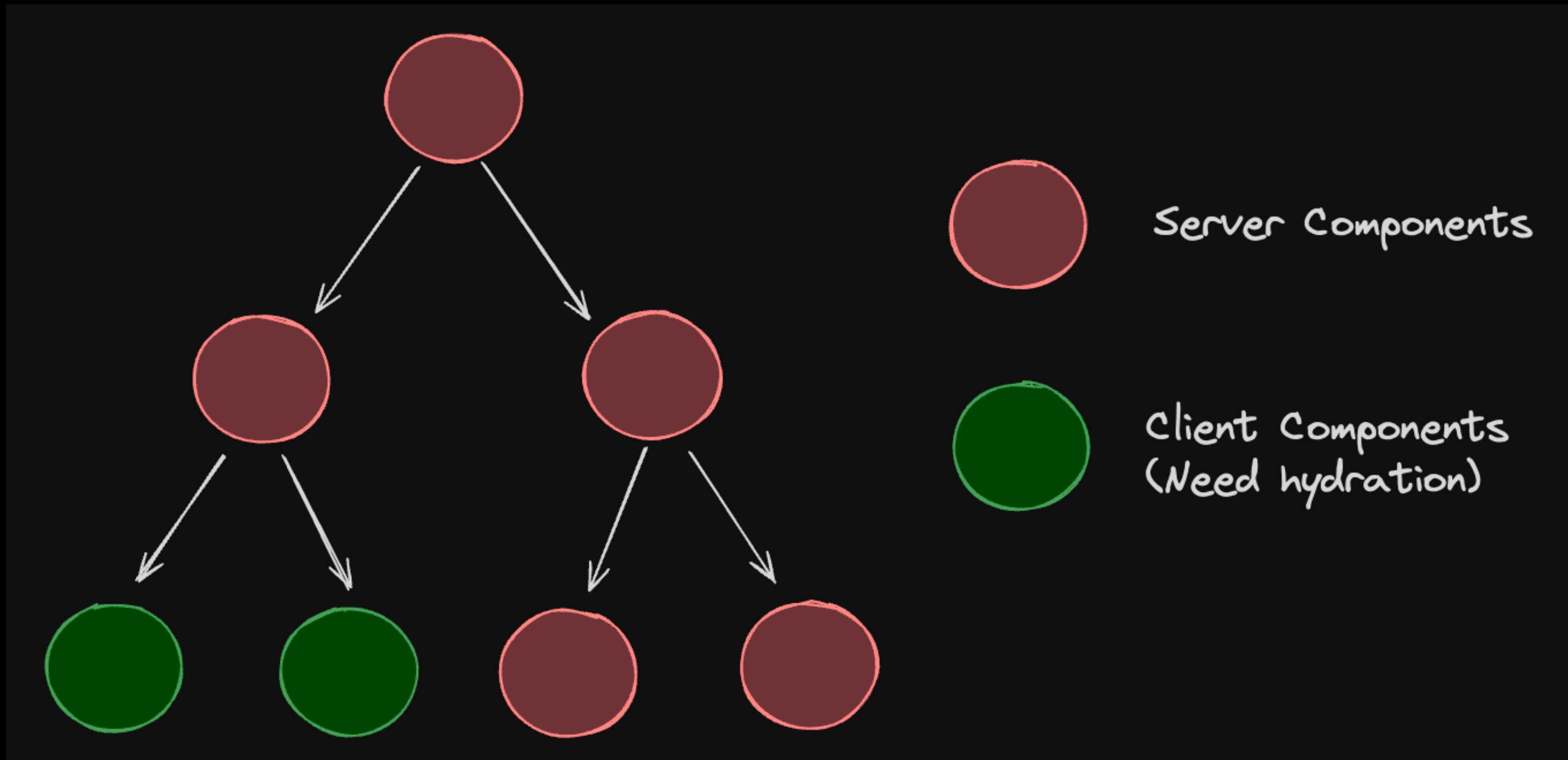
# Page Router

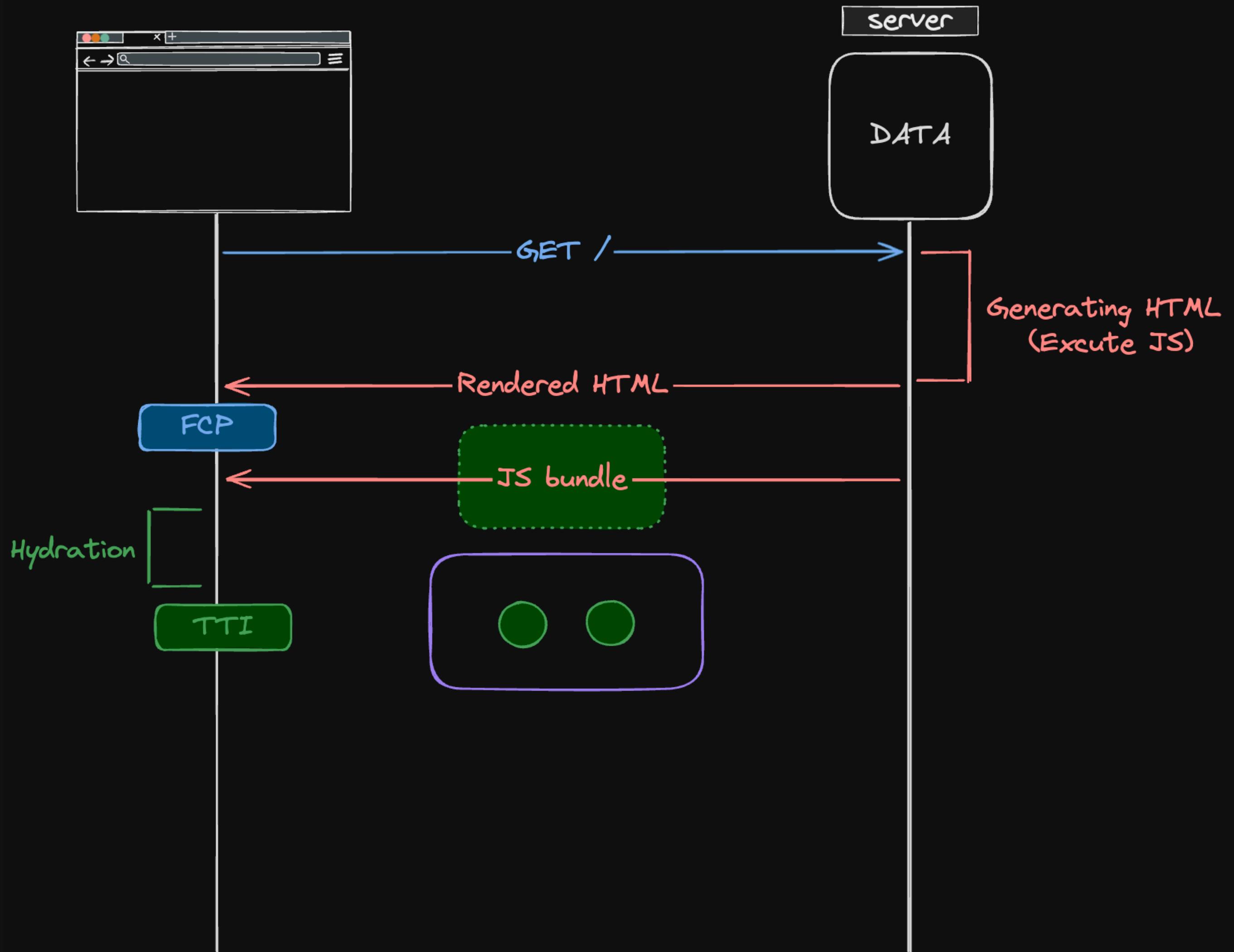


# Page Router

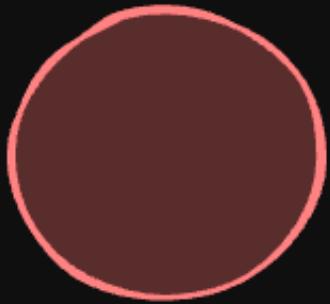


# App Router



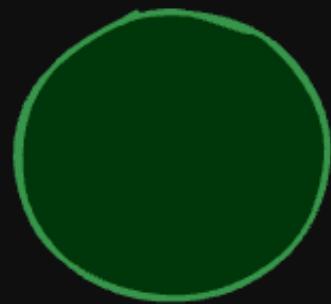


# App Router



Server Components

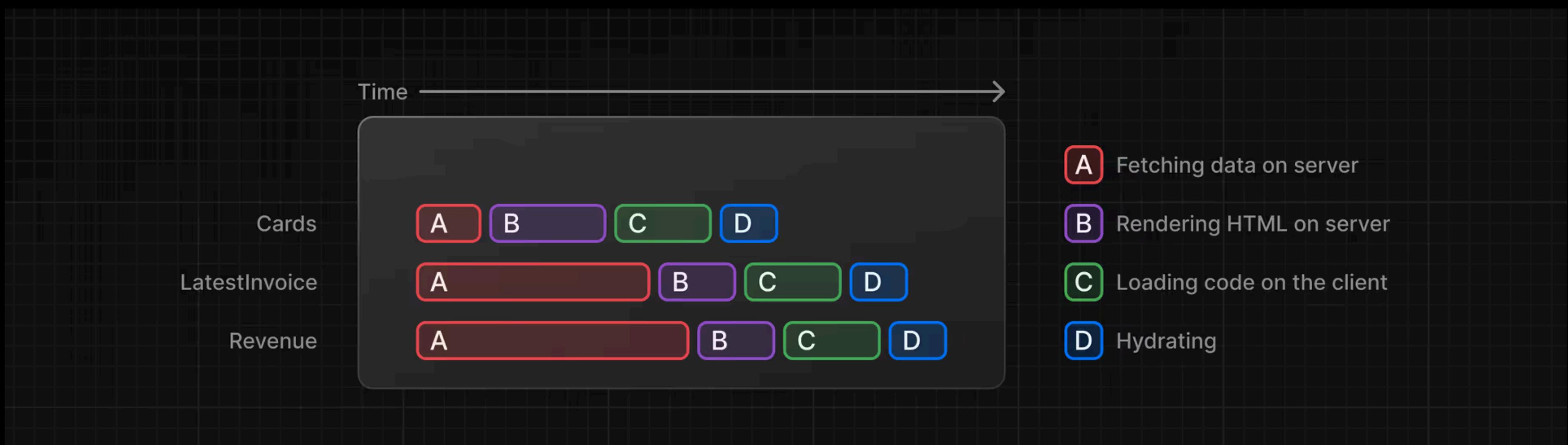
1. No interaction
2. Executed on the Server Only



Client Components

1. With interaction
2. Executed on Both Sides (Pre-render, hydration)

# App Router - Streaming

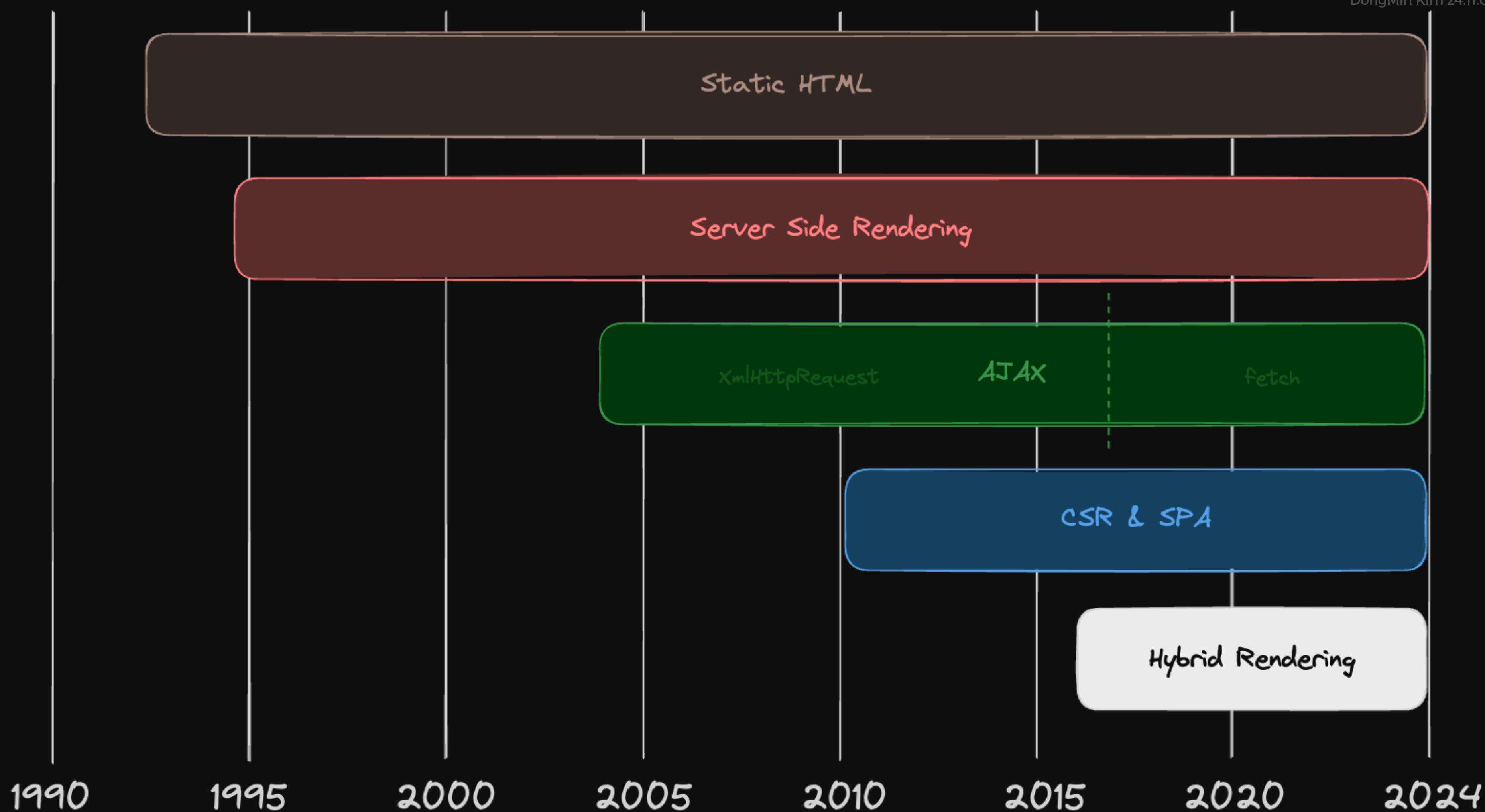


---

# Interim Summary

☒Keep going





# Official Docs

Came a long way

The screenshot shows the React v18.3.1 documentation site. The left sidebar has sections for react@18.3.1 (Overview, Hooks, Components, APIs) and react-dom@18.3.1 (Hooks, Components, APIs, Client APIs, Server APIs). The main content area is titled "API REFERENCE > React Server Components". It describes Server Components as a new type of Component that renders ahead of time, before bundling, in an environment separate from your client app or SSR server. It explains that this separate environment is the "server" in React Server Components. Server Components can run once at build time on your CI server, or they can be run for each request using a web server. A list of related topics includes: Server Components without a Server, Server Components with a Server, Adding interactivity to Server Components, and Async components with Server Components. A "Note" section discusses building support for Server Components, noting that while the components themselves are stable, the underlying APIs used to implement them do not follow semver and may break between minors in React 19.x. It recommends pinning to a specific React version or using the Canary release. A "Legacy APIs" link is also present.

---

# Server Components

Server Components are a new type of Component that renders ahead of time, before bundling, in an environment separate from your client app or SSR server.

---

# Server Components

- Server Components without a Server
- Server Components with a Server
- Adding interactivity to Server Components
- Async components with Server Components

SSG

SSR

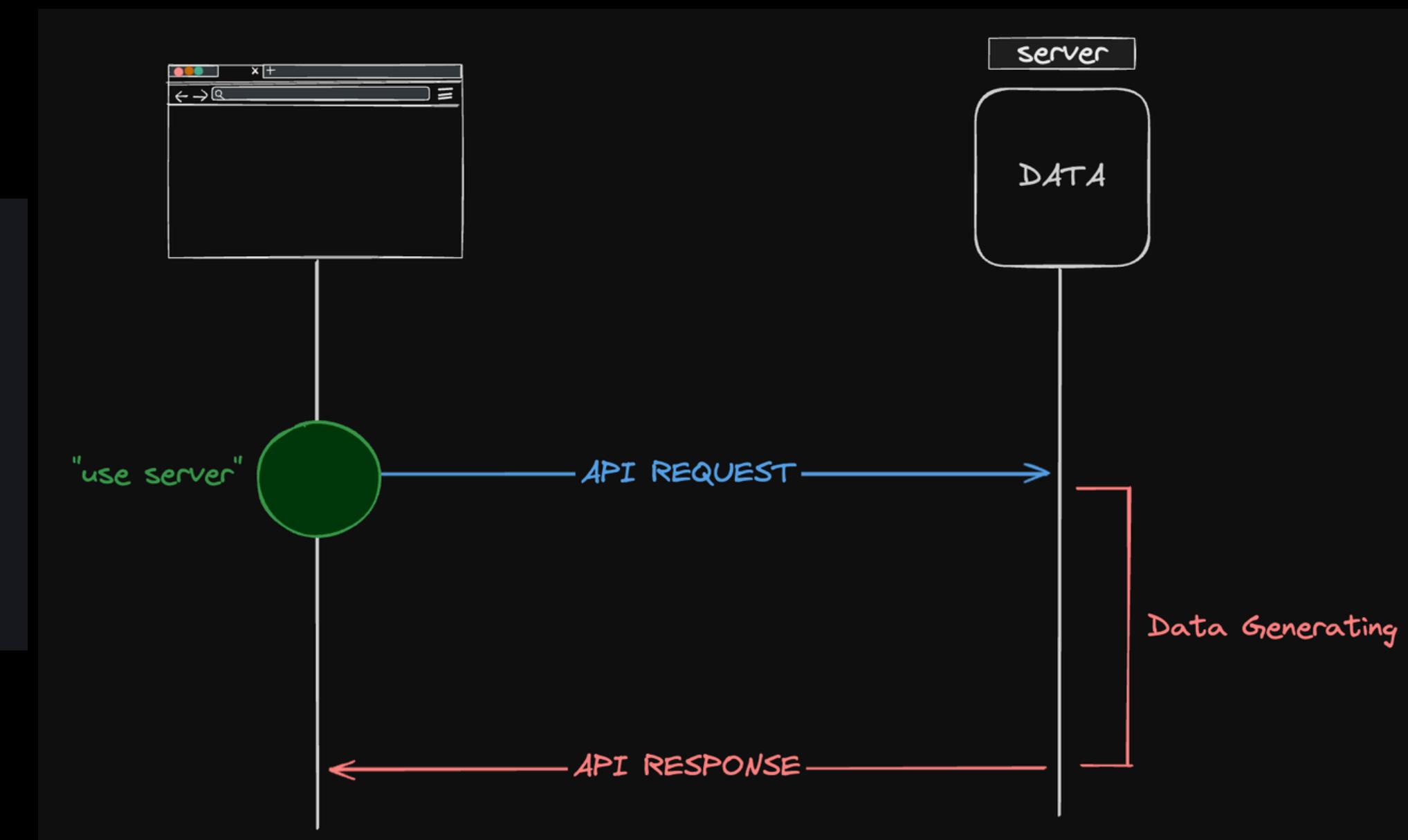
Hydration

Streaming

# Server Actions

Server Actions allow Client Components to call async functions executed on the server.

```
async function createNoteAction() {  
  // Server Action  
  'use server';  
  
  await db.notes.create();  
}
```



# Directives - “use server”

Notably, these are not supported:

- React elements, or JSX
- Functions, including component functions or any other function that is not a Server Action
- Classes
- Objects that are instances of any class (other than the built-ins mentioned) or objects with a null prototype
- Symbols not registered globally, ex. `Symbol('my new symbol')`

Supported serializable return values are the same as [serializable props](#) for a boundary Client Component.

# Directives - “use client”

## Advantages of Server Components

- Server Components can reduce the amount of code sent and run by the client. Only Client modules are bundled and evaluated by the client.
- Server Components benefit from running on the server. They can access the local filesystem and may experience low latency for data fetches and network requests.

## Limitations of Server Components

- Server Components cannot support interaction as event handlers must be registered and triggered by a client.
  - For example, event handlers like `onClick` can only be defined in Client Components.
- Server Components cannot use most Hooks.
  - When Server Components are rendered, their output is essentially a list of components for the client to render. Server Components do not persist in memory after render and cannot have their own state.

# Directives - “use client”

Notably, these are not supported:

- Functions that are not exported from client-marked modules or marked with `'use server'`
- Classes
- Objects that are instances of any class (other than the built-ins mentioned) or objects with a null prototype
- Symbols not registered globally, ex. `Symbol('my new symbol')`

# Directives - “use server”

Notably, these are not supported:

- React elements, or JSX
- Functions, including component functions or any other function that is not a Server Action
- Classes
- Objects that are instances of any class (other than the built-ins mentioned) or objects with a null prototype
- Symbols not registered globally, ex. `Symbol('my new symbol')`

Supported serializable return values are the same as [serializable props](#) for a boundary Client Component.

---

# THANK YOU

---

# QnA & Discussion

---

# Retrospect

Please write in free format and upload.

**NICKNAME.MD**

in 11\_01\_ServerComponents/Retrospect

[https://github.com/gdsc-konkuk/24-25-study-react-nextjs-docs/tree/main/11\\_01\\_ServerComponents](https://github.com/gdsc-konkuk/24-25-study-react-nextjs-docs/tree/main/11_01_ServerComponents)

---