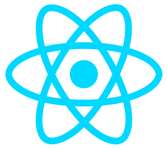
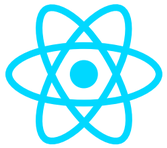


What's New in

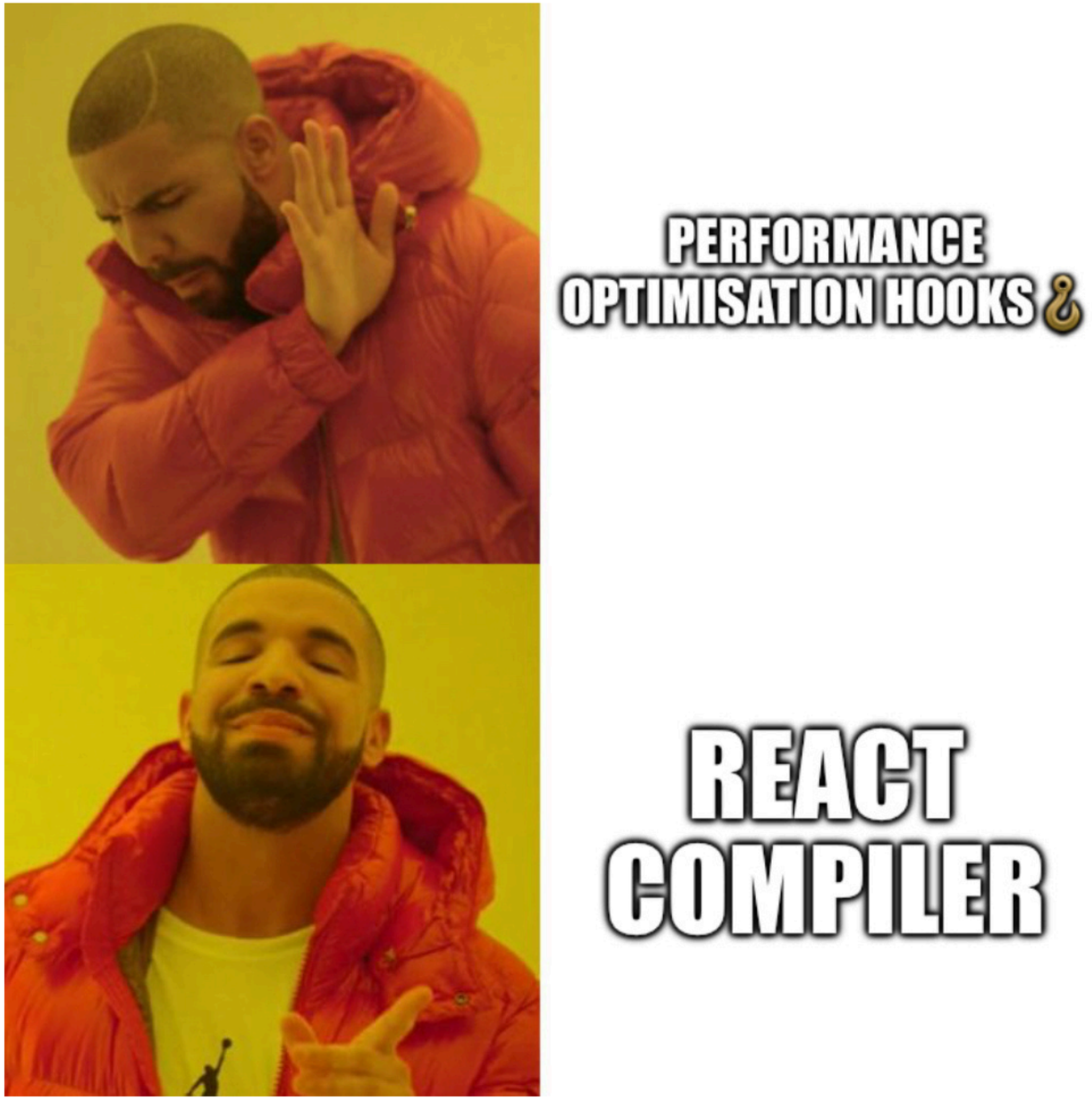
React v19

Let's Explore

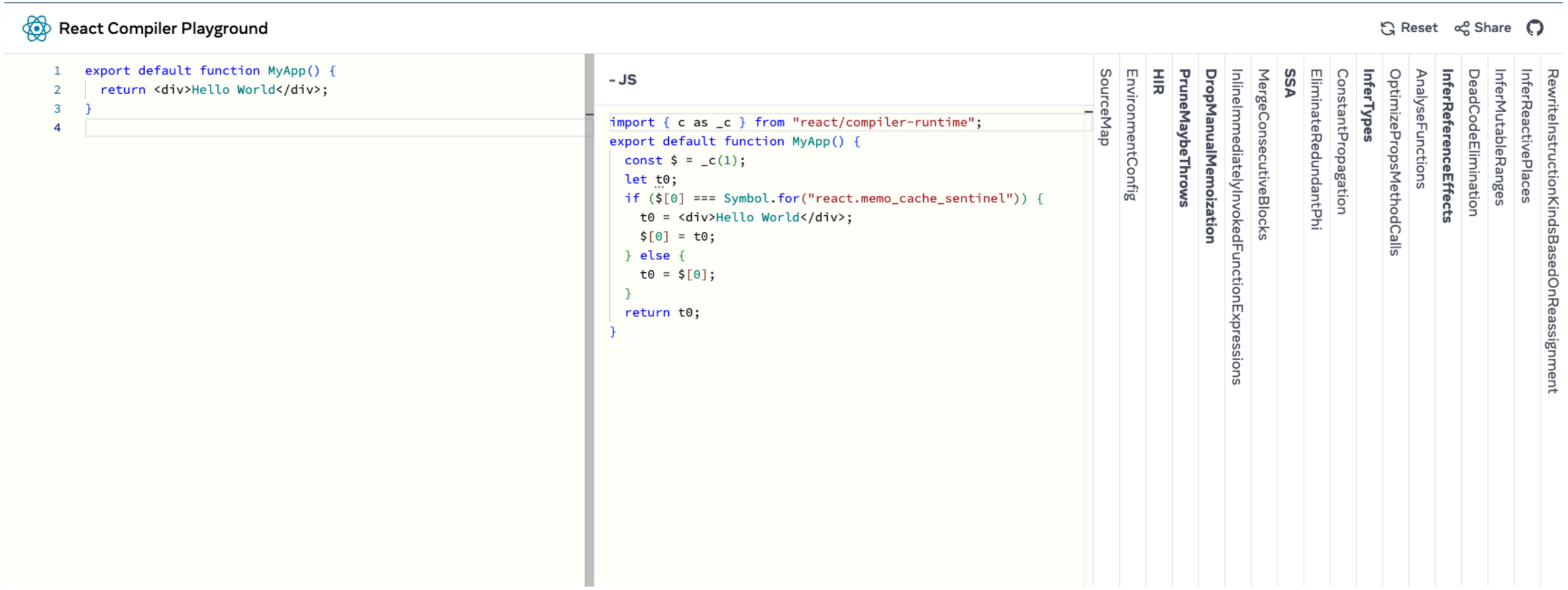

useMemo()
useCallBack()
COMPILE
JAKE-CLARK.TUMBLR
imgflip.com

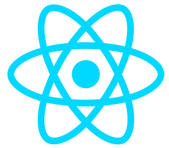


Compiler


PERFORMANCE
OPTIMISATION HOOKS
REACT
COMPILER

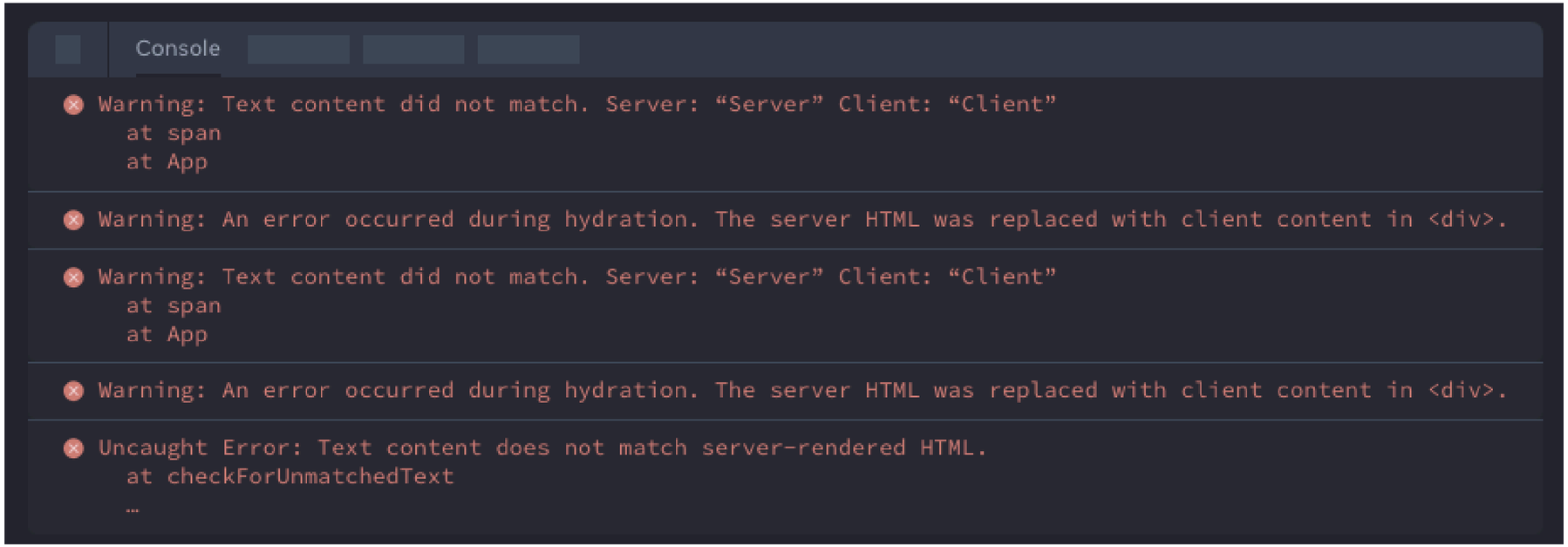
React Complier Playground



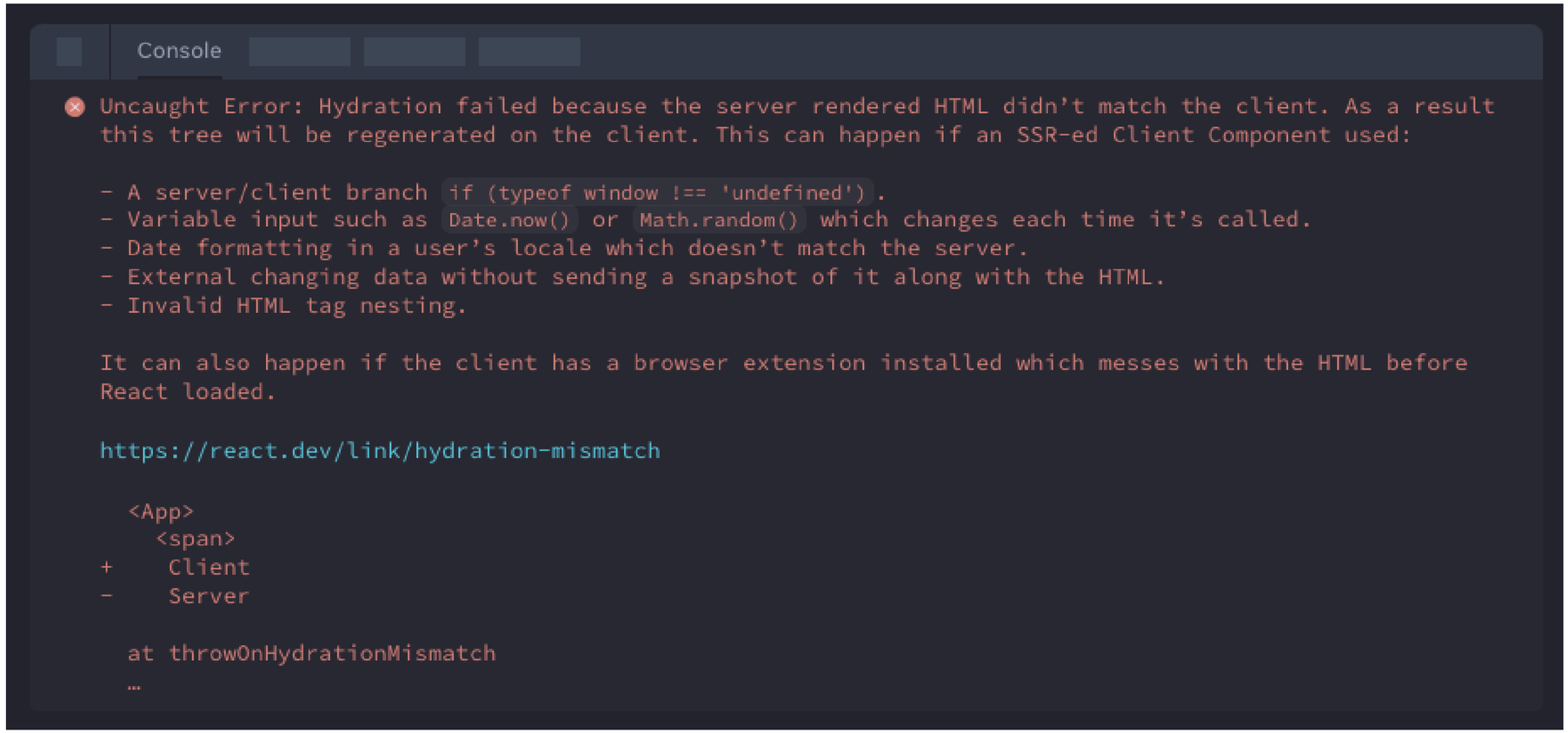


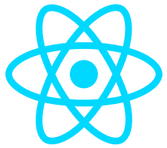
Errors

Existing Logs



Improvised Logs

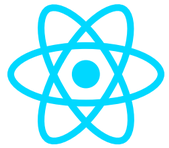




SEO

Support for MetaData

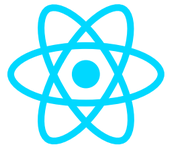
```
function BlogPost({post}) {
  return (
    <article>
      <h1>{post.title}</h1>
      <title>{post.title}</title>
      <meta name="author" content="Josh" />
      <link rel="author" href="https://twitter.com/joshcstory/" />
      <meta name="keywords" content={post.keywords} />
      <p>
        Eee equals em-see-squared...
      </p>
    </article>
  );
}
```



Stylesheets

Support for Stylesheets

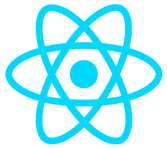
```
function ComponentOne() {
  return (
    <Suspense fallback="loading...">
      <link rel="stylesheet" href="foo" precedence="default" />
      <link rel="stylesheet" href="bar" precedence="high" />
      <article class="foo-class bar-class">
        {...}
      </article>
    </Suspense>
  )
}

function ComponentTwo() {
  return (
    <div>
      <p>{...}</p>
      <link rel="stylesheet" href="baz" precedence="default" />  <-- will be inserted between foo & bar
    </div>
  )
}
```



‹script/›

Support for async scripts

```
function MyComponent() {
  return (
    <div>
      <script async={true} src="..." />
      Hello World
    </div>
  )
}

function App() {
  <html>
    <body>
      <MyComponent>
      ...
      <MyComponent> // won't lead to duplicate script in the DOM
    </body>
  </html>
}
```



Shhh!

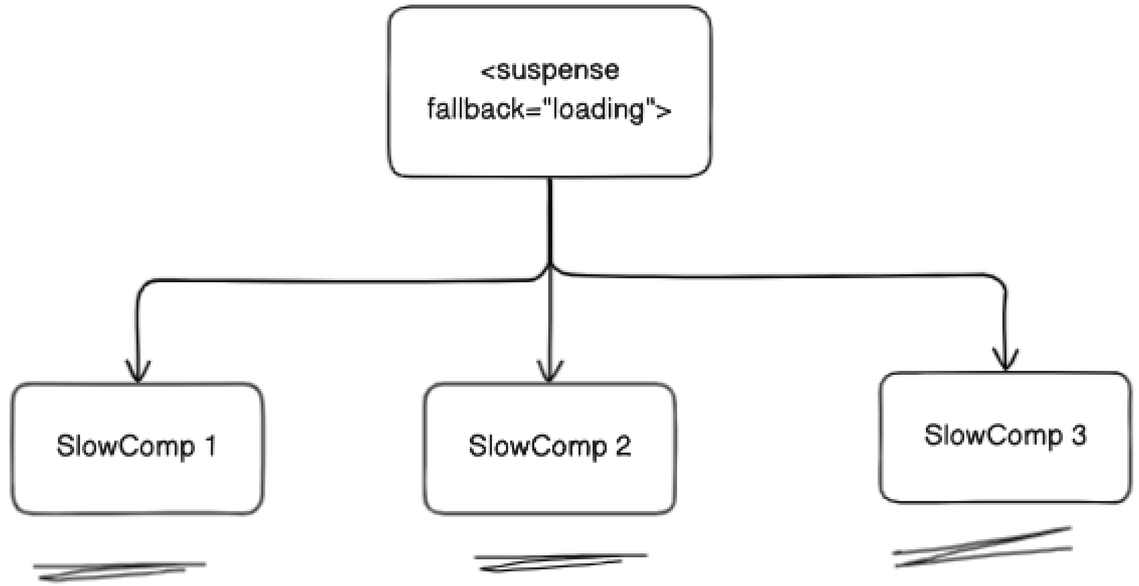
It’s a Suspense

Snippet

```jsx
import React, { Suspense } from "react";
import { use } from "react";

const delayedPromise = (sec) => {
  return new Promise((resolve) => setTimeout(resolve, sec));
};

const SlowComponent = (props) => {
  use(delayedPromise(props.time));
  return <div>SlowComponent {props.time}</div>;
};

const SuspenseDemo = () => {
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <SlowComponent time={1000} />
      <SlowComponent time={2000} />
      <SlowComponent time={3000} />
    </Suspense>
  );
};

export default SuspenseDemo;
```

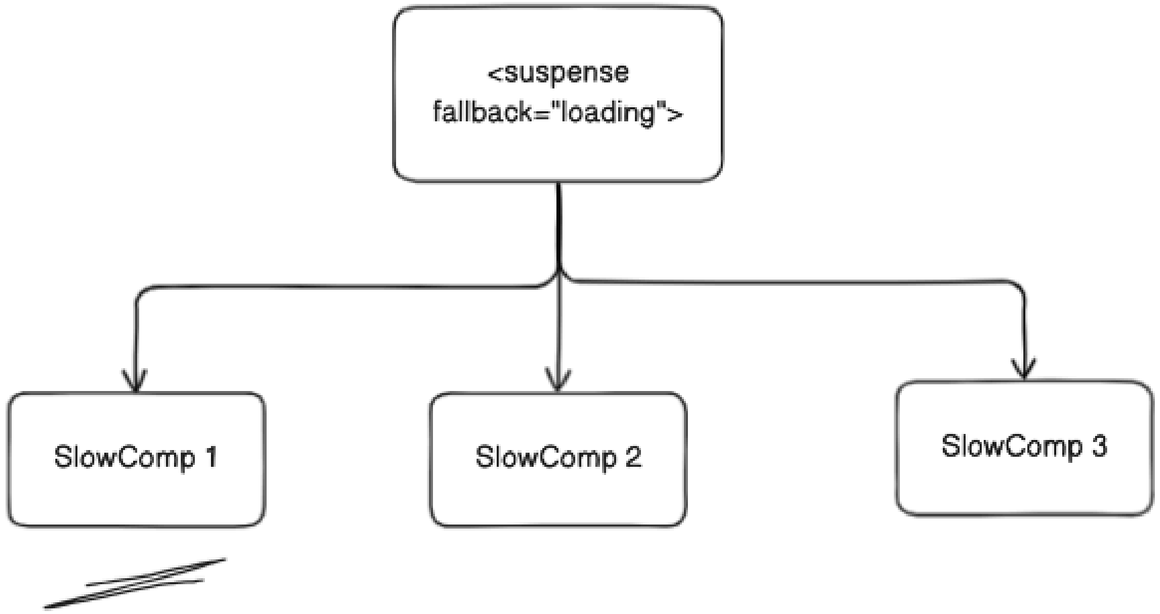
Editor

Existing Suspense

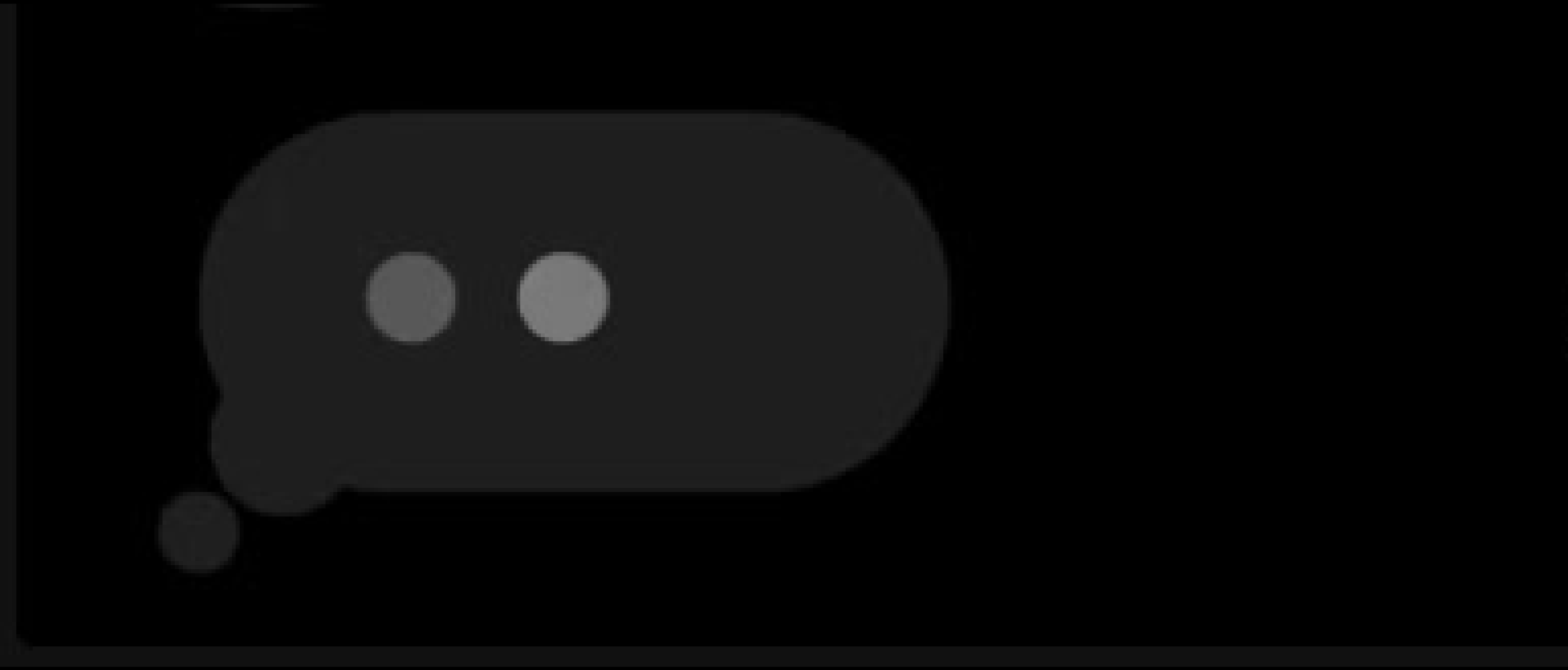


when a component is suspended, the suspended siblings were rendered and then the fallback was committed.

v19
Suspense

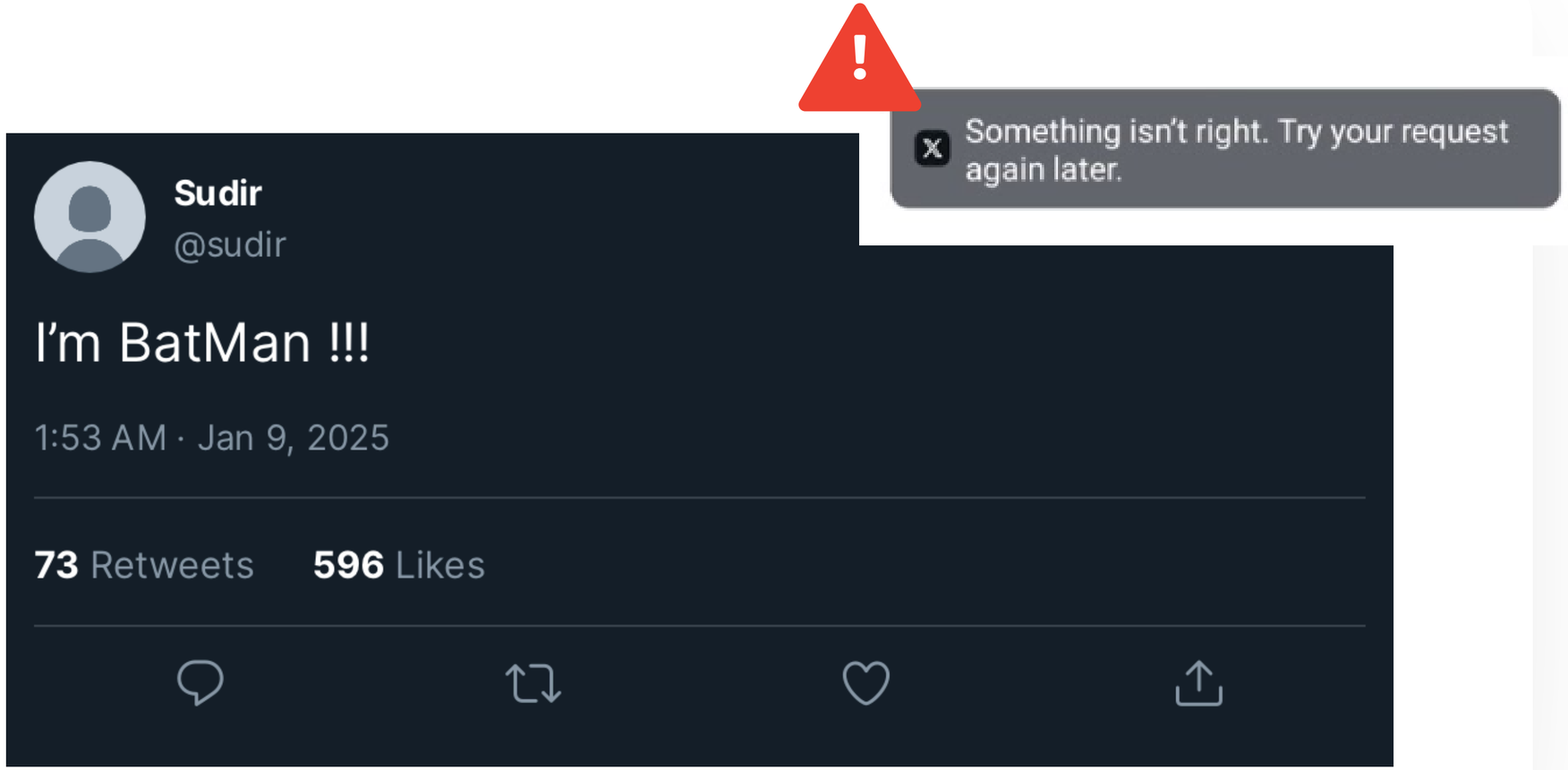


In React 19, when a component suspends,
the fallback is committed and then the suspended siblings are rendered.




Sudir
@sudir
I’m BatMan !!!
1:53 AM · Jan 9, 2025
73 Retweets
596 Likes

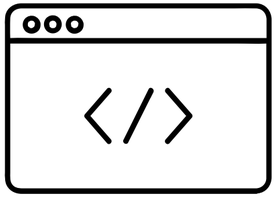
Editing a Tweet


Something isn't right. Try your request again later.
Sudir
@sudir
I'm BatMan !!!
1:53 AM · Jan 9, 2025
73 Retweets
596 Likes

Hooks

useOptmistic()

Code



New API

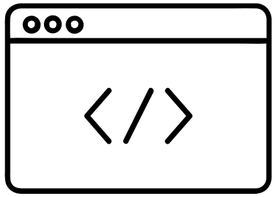
use()

Usage

use API

1. Promise Handling
2. Context Consumption

Code



Benefits

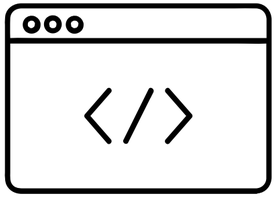
use API

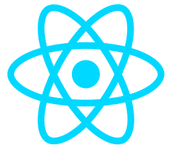
1. Less Declarative Code
2. Automatic Suspense Integration
3. Cleaner Components

Hooks

Context API

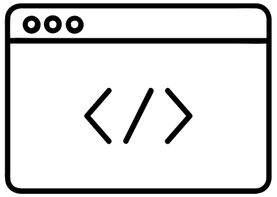
Code





Refs

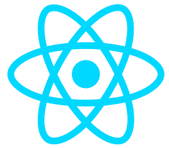
Code



Cleanup functions for refs

```
<input
  ref={(ref) => {
    // ref created

    // cleanup function
    return () => {
      // ref cleanup
    };
  }}
/>
```

Well There's more



Forms Actions

Form actions introduce a streamlined way to handle form submissions, making it easier to manage operations directly within the form components. This feature allows to specify functions that can be executed when a form is submitted, **significantly simplifying the process of connecting forms to server-side logic.**

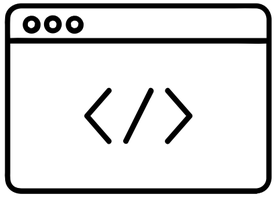
Key Features of Form Actions:

- **Direct Function Specification:** Developers can pass a function directly as the **action prop** of a <form>, <input>, or <button>. This function can handle the submission logic without needing separate API endpoints or manual state management.
- **Automatic Form Reset:** When a form action succeeds, React automatically resets the form for uncontrolled components, enhancing user experience by clearing inputs after submission.
- **Integration with New Hooks:** The useActionState hook can be used alongside form actions to manage form state and submission status seamlessly. This hook provides the current state, a function to trigger the action, and a boolean indicating if the submission is pending.
- **Server-Side Processing:** Actions can include server-side processing directly within the specified function using **"use server"** syntax, allowing for immediate handling of form data without additional setup.

Hooks

useActionState()

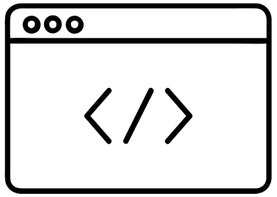
Code



Hooks

useFormSatus()

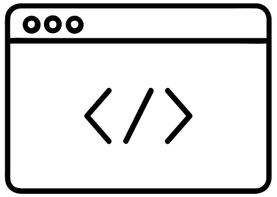
Code

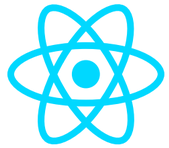


Hooks

useTransition()

Code





Server Actions

Server Actions in React 19 are a powerful feature that allows developers to define asynchronous functions that **run on the server side and can be invoked from client components**. This capability simplifies the interaction between client and server, **enabling direct access to server-side resources like databases without the need for separate API endpoints**.

Key Features of Server Actions:

Asynchronous Execution: Server Actions must be defined as async functions, allowing them to perform operations such as database queries or file system access.

Use of "use server" Directive: To designate a function as a Server Action, you need to include the "use server" directive at the top of the function. This indicates that the function will run on the server.

Integration with Client Components: Server Actions can be invoked from client components through forms or event handlers, making it easy to submit data back to the server.

Progressive Enhancement: Server Actions support progressive enhancement, meaning forms will still work even if JavaScript is not fully loaded or enabled.

Automatic Handling of Form Submissions: When a form is submitted, React can handle it without refreshing the page, providing a smoother user experience.

Snippet

```
// actions.js (Server Action file)
export async function createUser(data) {
    "use server"; // Designate this function as a Server Action
    const { name, email } = data;

    // Simulate database operation
    await db.users.create({ name, email });

    return { success: true };
}

// UserForm.js (Client Component)
import { useActionState } from "react";
import { createUser } from './actions';

export function UserForm() {
    const [state, submitAction, isPending] = useActionState(createUser, { success: false });

    return (
        <form action={submitAction}>
            <input type="text" name="name" placeholder="Name" required />
            <input type="email" name="email" placeholder="Email" required />
            <button type="submit" disabled={isPending}>Create User</button>
            {state.success && <p>User created successfully!</p>}
        </form>
    );
}

```

Thank You