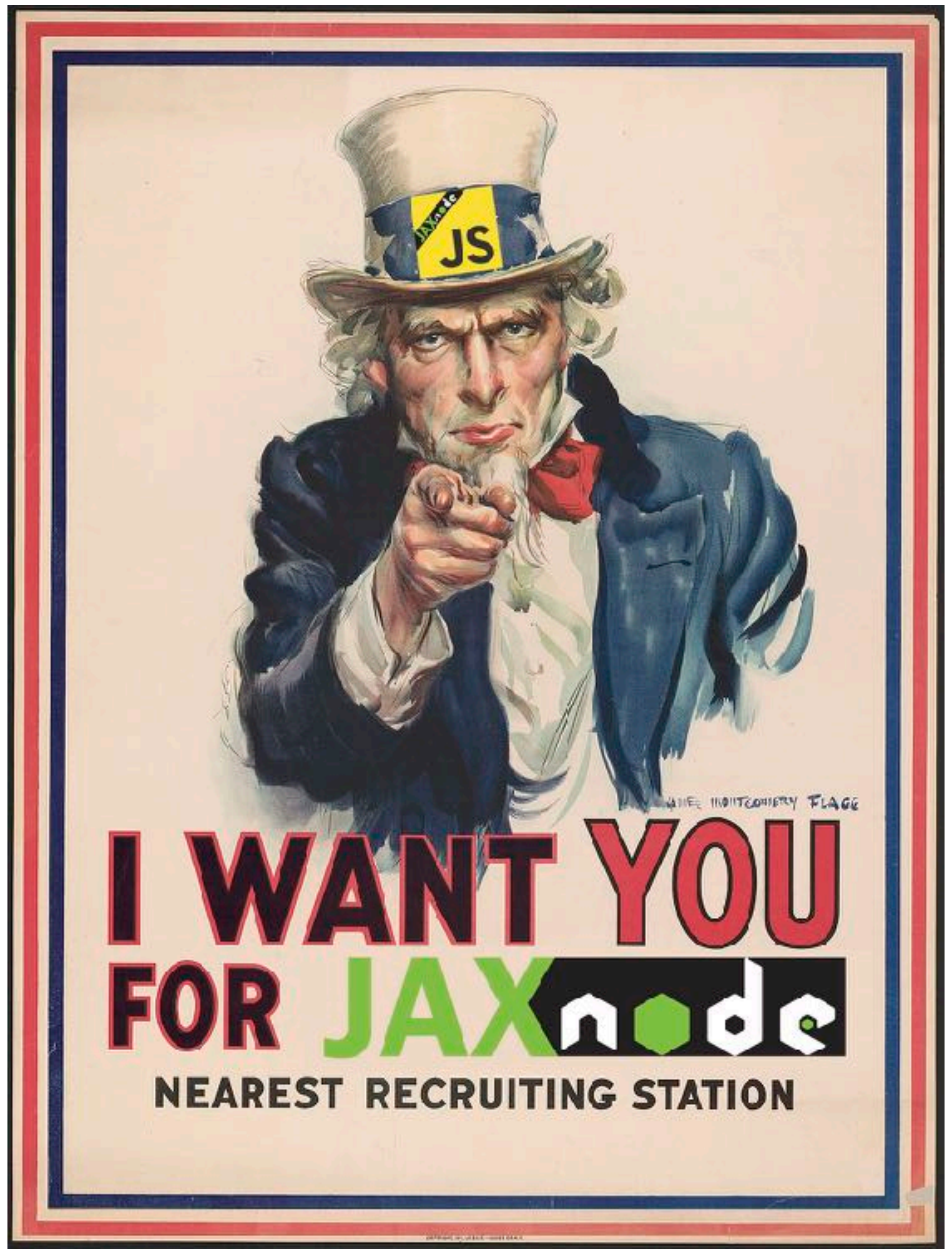
React 19

Whats new in React?

SCAN ME

JaxNode 2025

Looking for Speakers



About me

David Fekke

- JaxNode user group
- Web and Mobile Developer
- JS, TS, React, C#, Swift, Obj-C, Kotlin, Java and SQL
- fek.io/blog/1
- youtube.com/c/polyglotengineer
- github.com/davidfekke
- @jaxnode @polyglotengine1

What is React

- React is a framework that can be used for building many types of applications using reusable components
- It is primarily thought of as a client side web based view framework, but has grown into a much larger set of applications
- Using JSX or TSX files, you can combine HTML in your JavaScript creating components that can be combined into views

```
function MyButton() {
  return (
    <button>I'm a button</button>
  );
}

export default function MyApp() {
  return (
    <div>
      <h1>Welcome to my app</h1>
      <MyButton />
    </div>
  );
}
```

Quick History of React

React over the years

- Facebook starts work on React 2013
- MVC does not work well at Facebook
- Facebook open sources React 2014
- React Native introduced in 2015
- Component class based framework
- Moved to functional components and hooks in 2019
- Recently saw the introduction of React Server Components for server side rendering

React Frameworks

React ecosystem

- Redux, Mobx, Zustand and Jotai for state management
- React Native for native applications, including support for mobile, Windows and Mac
- Vite has become popular for scaffolding, create-react-app is deprecated
- Next.js is a popular Node.js framework for server side React development
- React Router and Remix have merged into a single framework
- Next.js 15 has full support for React 19

React as a front-end

- React can be used with any backend
- Many popular backends including Laravel, Flask, .NET and Go lang can all be used with React
- React can also be served statically using Next.js or Gatsby
- You can use services like Firebase for backend
- You can use services like Clerk for auth

React 19

New features in React 19

- Actions allow async functions to be used within transitions for handling pending states, error and forms
- The useActionState hook simplifies form states and submissions
- The useFormStatus hook manages the status of the last form submission
- The useOptimistic hook allows for optimistic UI updates during async data mutations
- Server Components now part of React framework

React 19

New features continued

- New `use` API allows for you to read resources like Promises or context within render functions, but it is considered experimental
- You can now use `Ref` as a prop, eliminating the need for a forwardRef
- Improved error reporting
- Support for document metadata for the <title>, <link> and <meta> elements directly in components
- Enhanced Asset loading for pre-loading scripts, fonts and stylesheets

React compiler

Never use useMemo or useCallback again

- Meta began using the React compiler before React 19 was released
- The React compiler is not on by default, you have to opt-in
- The React compiler optimizes your code so you do not have to use useMemo or useCallback to prevent re-renders
- Can install using ESLint plugin:
- `$ npm install -D eslint-plugin-react-compiler@beta`

```
import reactCompiler from 'eslint-plugin-react-compiler'

export default [
  {
    plugins: {
      'react-compiler': reactCompiler,
    },
    rules: {
      'react-compiler/react-compiler': 'error',
    },
  },
]
```

Upgrading

How to upgrade React 19

- Install React 19 and React-dom 19
- Use a code mod
- `$ npx codemod@latest react/19/migration-recipe`
- Errors in render are not rethrown
- propTypes and defaultProps have been deprecated
- There is a code mod for propTypes if you are using TypeScript

Module Pattern Factories

```
// Before
function FactoryComponent() {
  return {
   render() {
          return <div />;
      }
   }
}
```

```
// After
function FactoryComponent() {
  return <div />;
}
```

Forms and Actions

Coming from Next.js

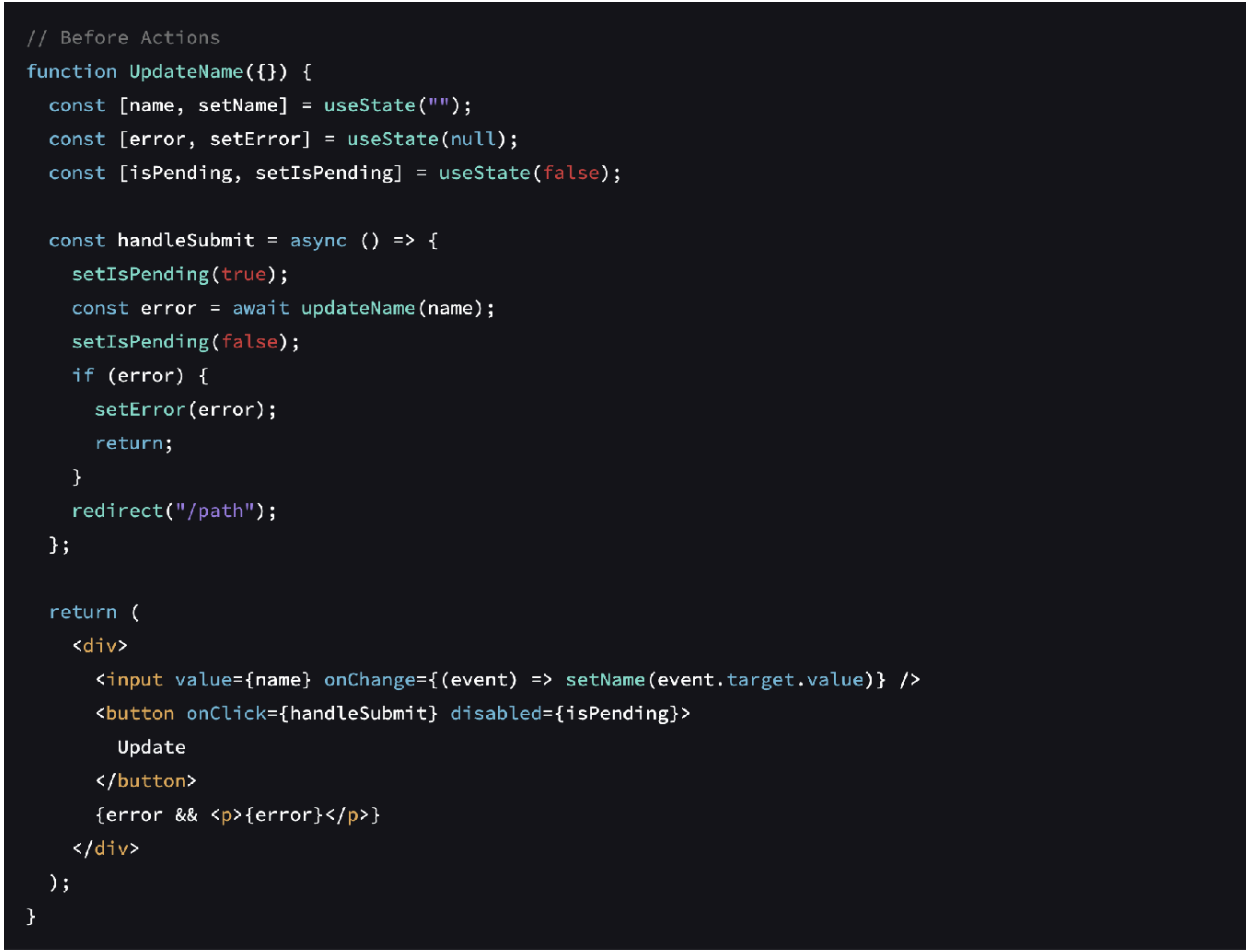
- Several changes coming from Next.js
- Next.js added support for server actions last year
- Several of the new hooks work well with these Next changes
- Use actions instead of onSubmit for form submission
- useOptimistic, useActionState, useFormStatus all enhance forms

Actions

- This is part of the new form features in `react-dom` API
- Use the `action` prop on the form
- <form action={actionFunction}>
- Use the `formAction` prop on the `input` and `button` elements
- When the form succeeds, it will reset the form
- You can call the `requestFormReset` on the React DOM API

useTransistion hook

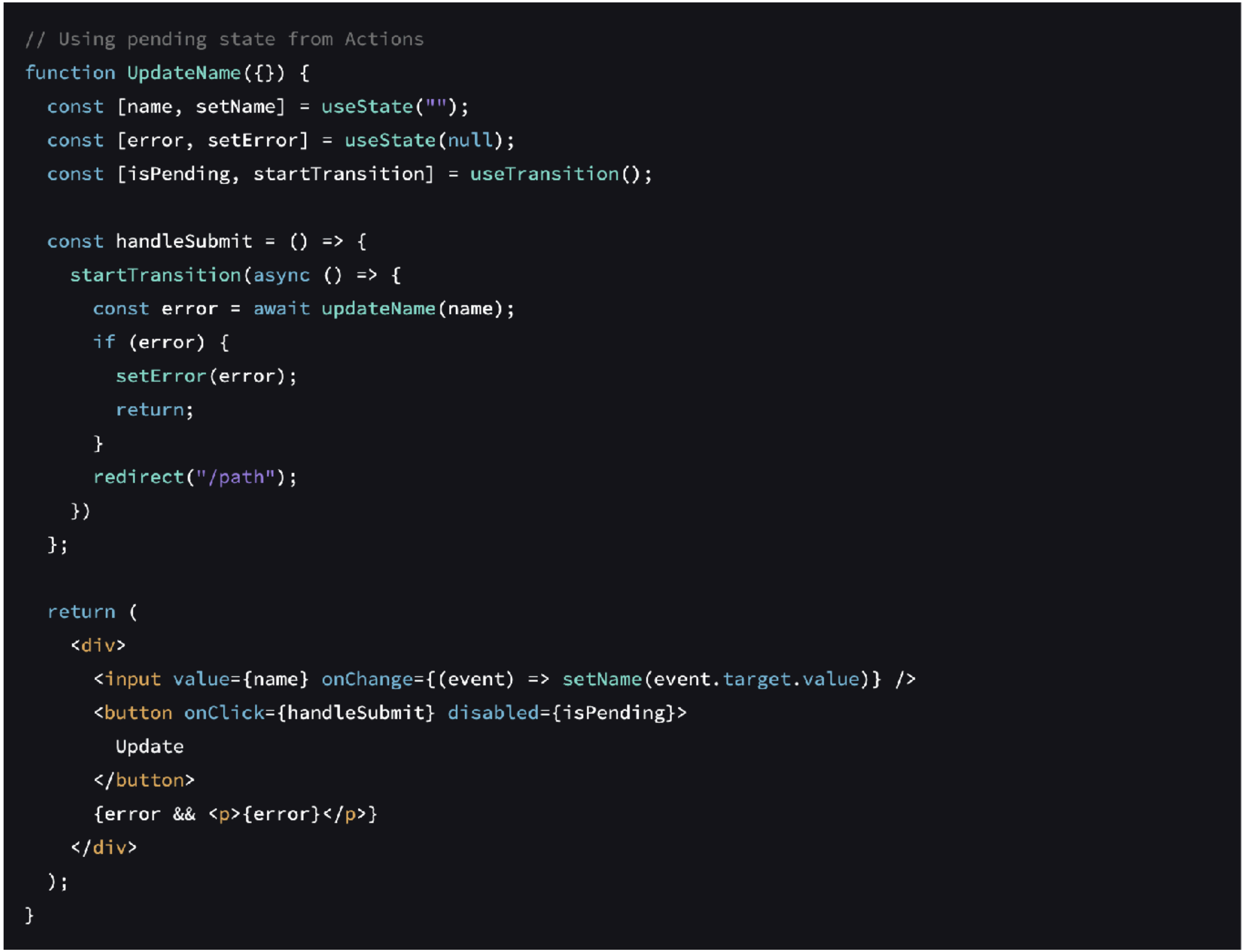
- `useTransition` will handle pending state
- const [isPending, startTransition] = useTransition();
- Use the `isPending` state to track when a transition is occurring and when it has been completed
- Async transitions are called “Actions”



```
// Before Actions
function UpdateName({}) {
  const [name, setName] = useState("");
  const [error, setError] = useState(null);
  const [isPending, setIsPending] = useState(false);

  const handleSubmit = async () => {
    setIsPending(true);
    const error = await updateName(name);
    setIsPending(false);
    if (error) {
      setError(error);
      return;
    }
    redirect("/path");
  };

  return (
    <div>
      <input value={name} onChange={(event) => setName(event.target.value)} />
      <button onClick={handleSubmit} disabled={isPending}>
        Update
      </button>
      {error && <p>{error}</p>}
    </div>
  );
}
```



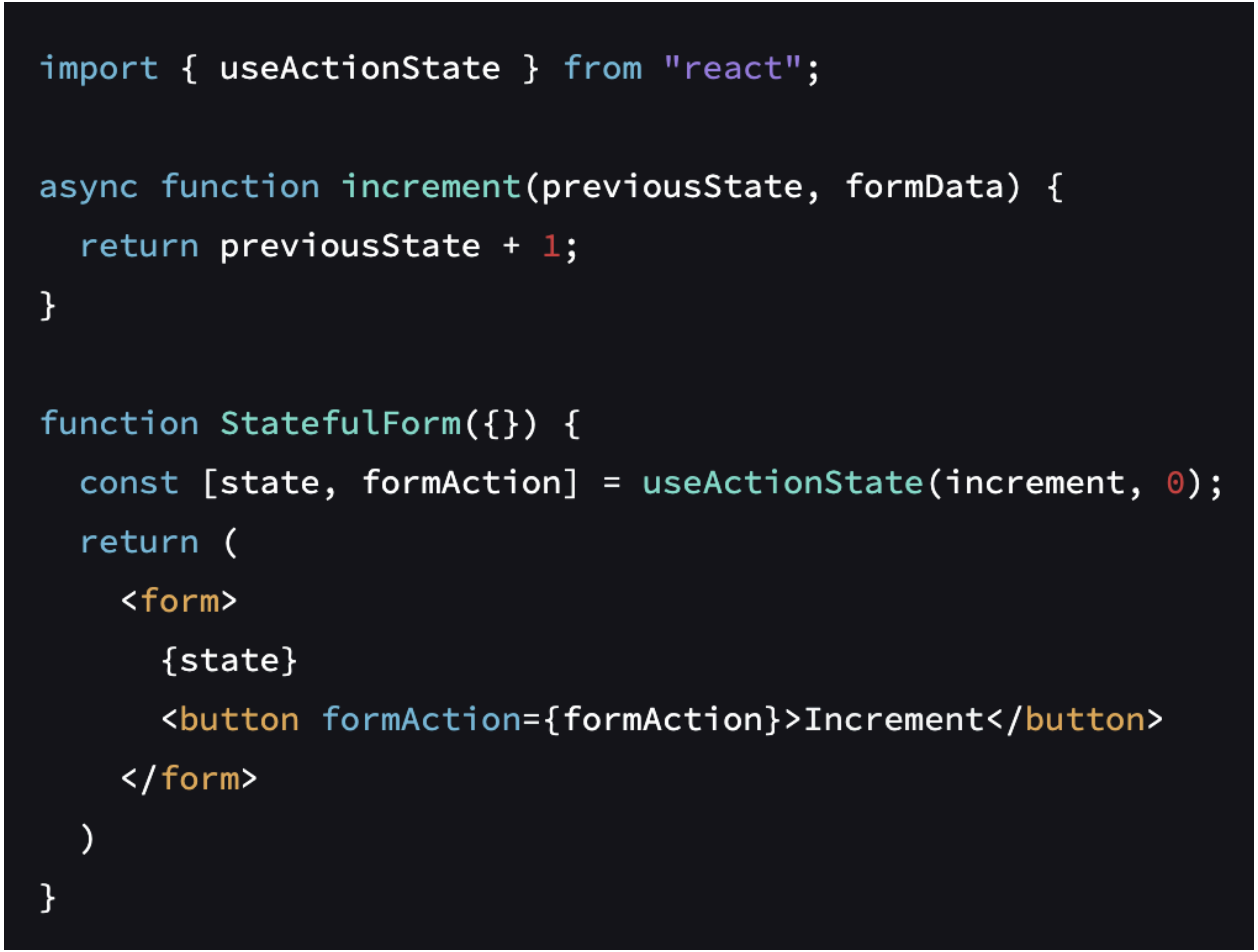
```
// Using pending state from Actions
function UpdateName({}) {
  const [name, setName] = useState("");
  const [error, setError] = useState(null);
  const [isPending, startTransition] = useTransition();

  const handleSubmit = () => {
    startTransition(async () => {
      const error = await updateName(name);
      if (error) {
        setError(error);
        return;
      }
      redirect("/path");
    })
  };

  return (
    <div>
      <input value={name} onChange={(event) => setName(event.target.value)} />
      <button onClick={handleSubmit} disabled={isPending}>
        Update
      </button>
      {error && <p>{error}</p>}
    </div>
  );
}
```

useActionState hook

- The `useActionState` hook can be used for handling common cases for Actions
- const [state, formAction, isPending] = useActionState(fn, initialState, permalink?);
- The `fn` takes two parameters, previousState and FormData



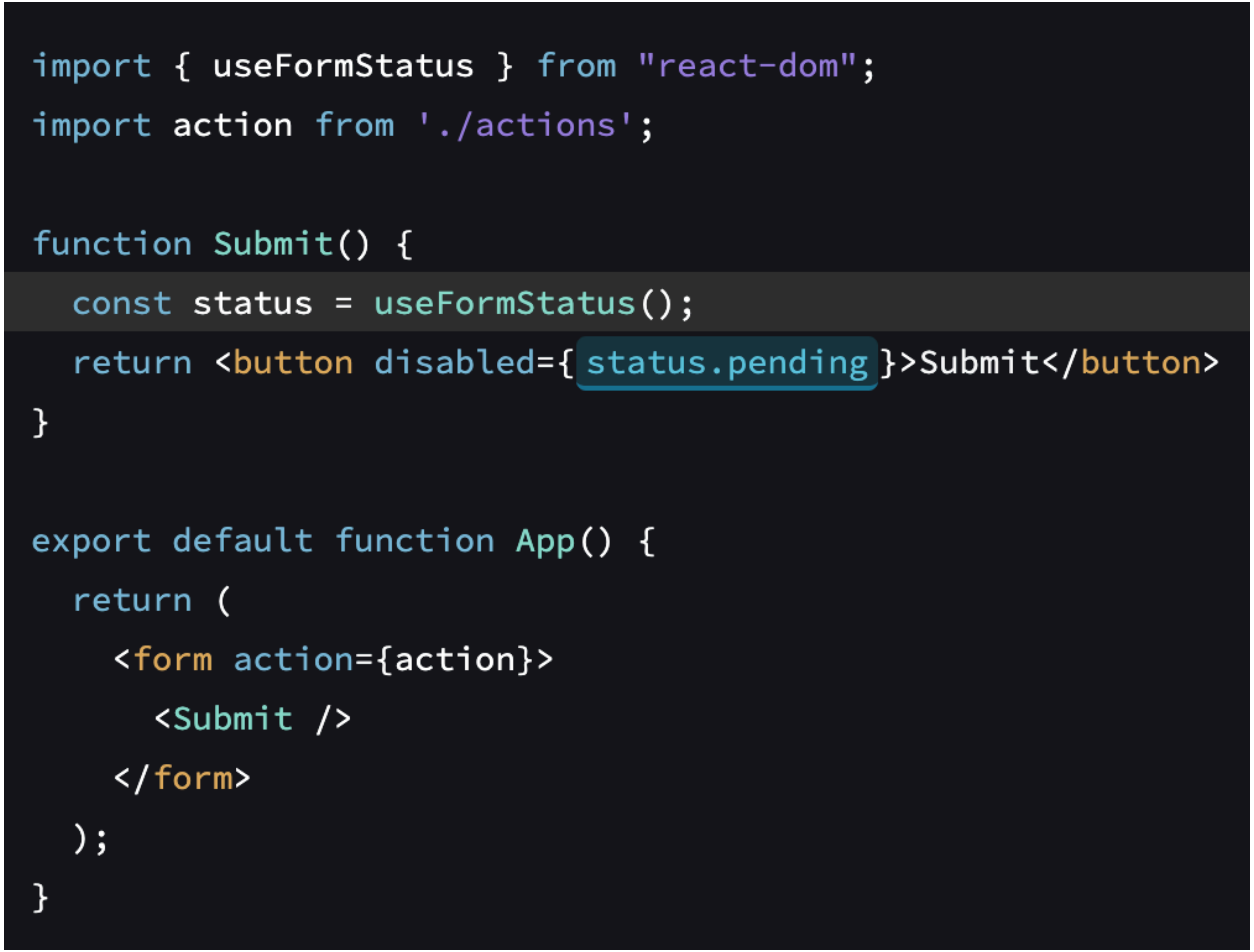
```
import { useActionState } from "react";

async function increment(previousState, formData) {
  return previousState + 1;
}

function StatefulForm({}) {
  const [state, formAction] = useActionState(increment, 0);
  return (
    <form>
      {state}
      <button formAction={formAction}>Increment</button>
    </form>
  )
}
```

useFormStatus hook

- The `useFormStatus` hook gives you the status of the last form submission
- This is useful for a child component inside of a form
- const { pending, data, method, action } = useFormStatus();
- Handy if you need to disable a button or other element while a form is being submitted



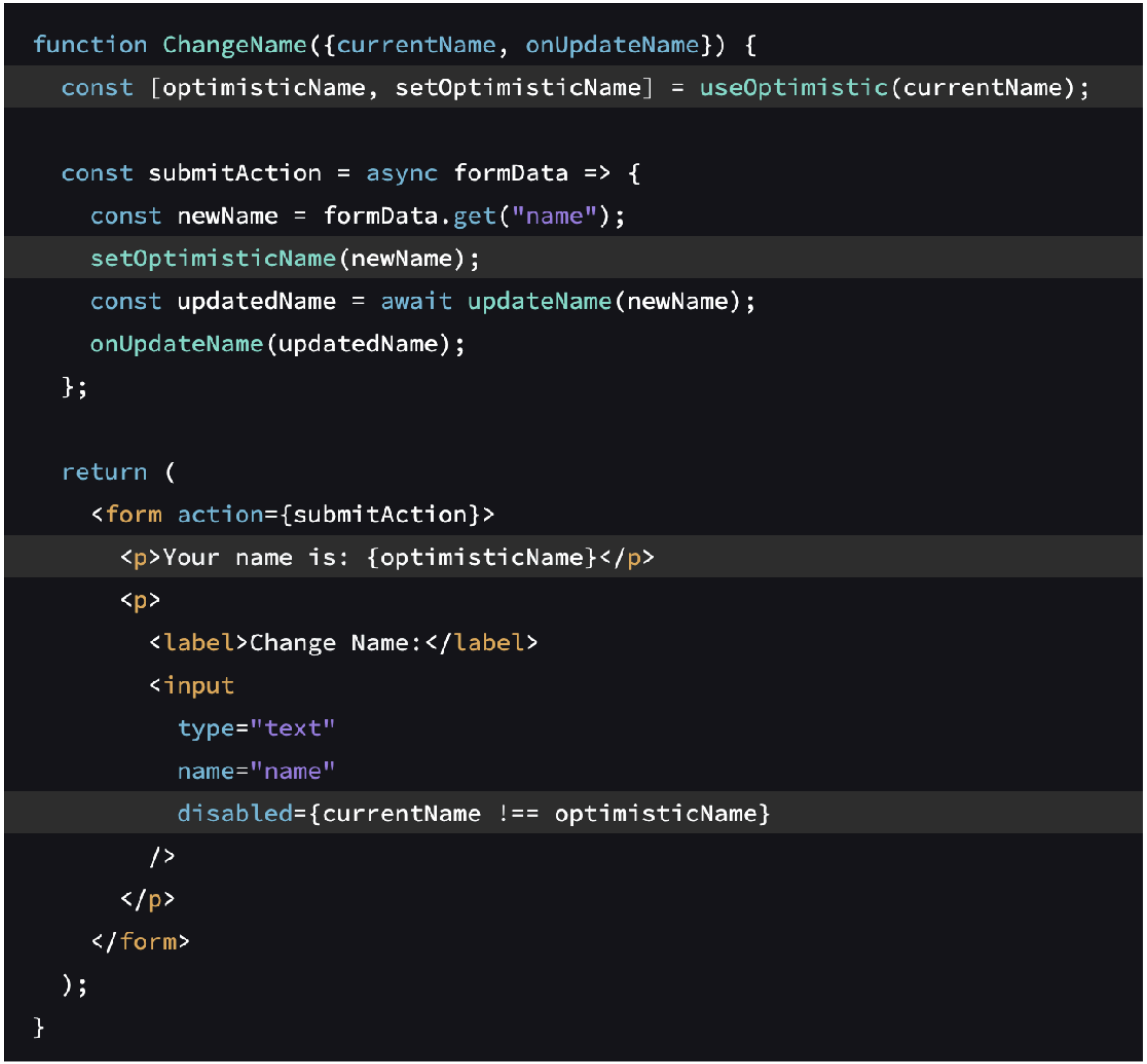
```
import { useFormStatus } from "react-dom";
import action from './actions';

function Submit() {
  const status = useFormStatus();
  return <button disabled={status.pending}>Submit</button>
}

export default function App() {
  return (
    <form action={action}>
      <Submit />
    </form>
  );
}
```

useOptimistic hook

- This hook allows you to show a final state optimistically while an async request is occurring
- const [optimisticState, addOptimistic] = useOptimistic(state, updateFn);
- The useOptimistic hook will immediately render the optimisticName while the updateName request is in progress



```
function ChangeName({currentName, onUpdateName}) {
  const [optimisticName, setOptimisticName] = useOptimistic(currentName);

  const submitAction = async formData => {
    const newName = formData.get("name");
    setOptimisticName(newName);
    const updatedName = await updateName(newName);
    onUpdateName(updatedName);
  };

  return (
    <form action={submitAction}>
      <p>Your name is: {optimisticName}</p>
      <p>
        <label>Change Name:</label>
        <input
          type="text"
          name="name"
          disabled={currentName !== optimisticName}
        />
      </p>
    </form>
  );
}
```

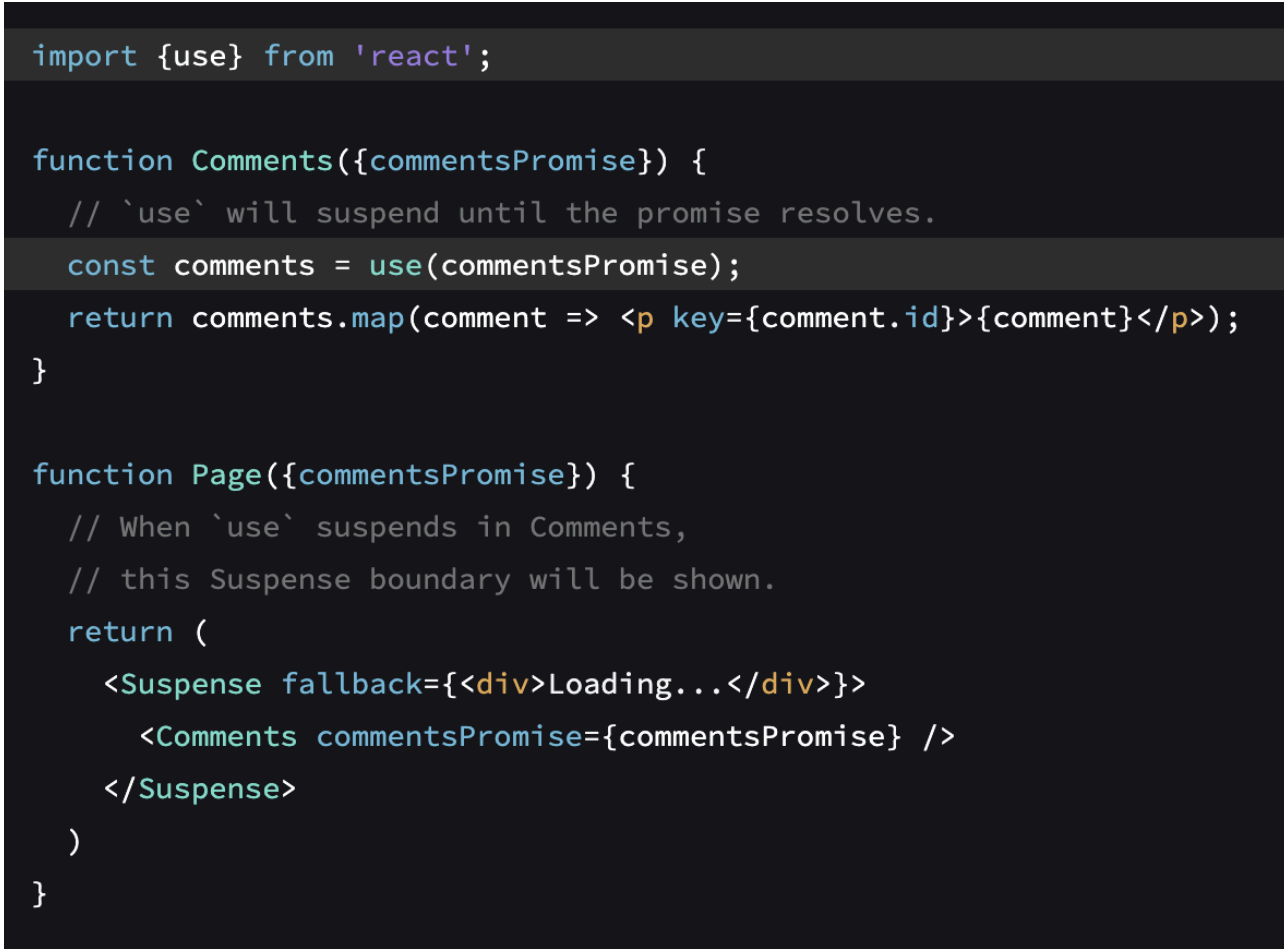
React Server Components

Generic React support

- This has been supported in frameworks like Next.js for over a year
- Now officially part of React
- Allows you to render components on the server and push them to the browser
- Also knows as SSR
- These can be used without a ‘Server’ if they are pre-rendered

`use` API

- Used to read resources in render
- You can read a Promise with the `use` API, and it will suspend until the promise resolves
- You cannot create a Promise in the render and try to pass that promise
- You will need to use a suspense powered library that supports caching of promises



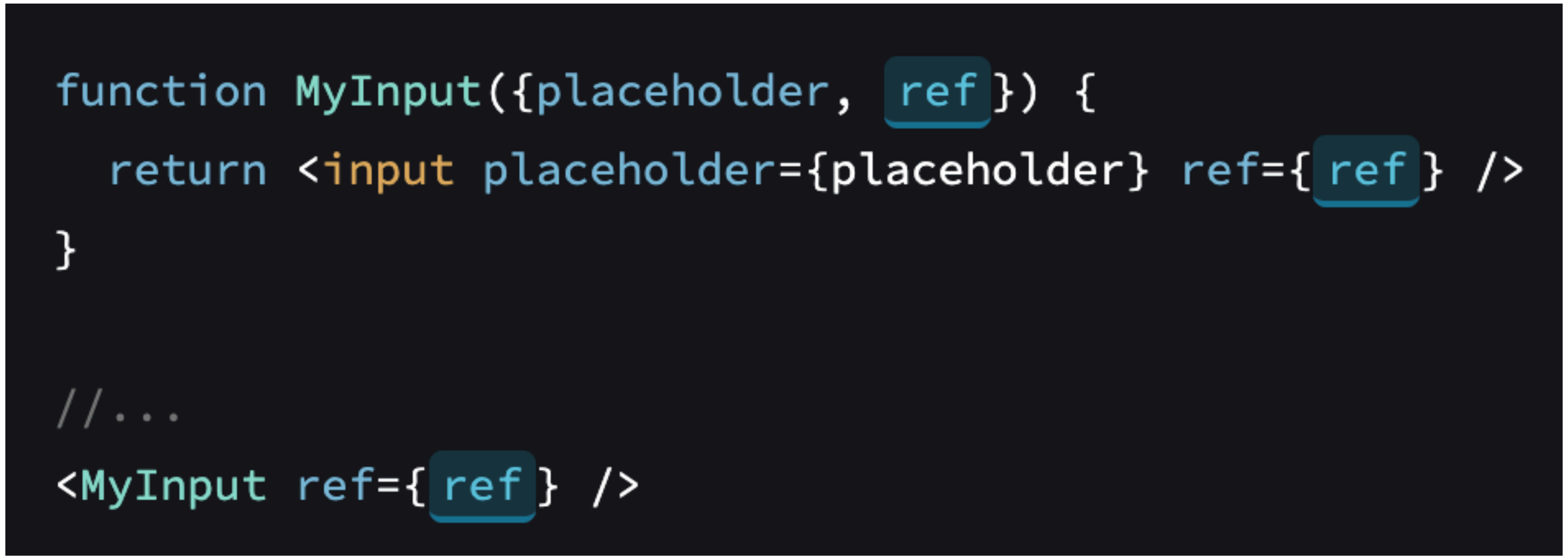
```
import {use} from 'react';

function Comments({commentsPromise}) {
  // `use` will suspend until the promise resolves.
  const comments = use(commentsPromise);
  return comments.map(comment => <p key={comment.id}>{comment}</p>);
}

function Page({commentsPromise}) {
  // When `use` suspends in Comments,
  // this Suspense boundary will be shown.
  return (
    <Suspense fallback={<div>Loading...</div>}>
      <Comments commentsPromise={commentsPromise} />
    </Suspense>
  )
}
```

`ref` Prop

- `ref` can be passed as a prop into functional components
- New function components will no longer need `forwardRef`
- There is a Codemod that will remove `forwardRef` from your existing code



```
function MyInput({placeholder, ref}) {
  return <input placeholder={placeholder} ref={ref} />
}

//...
<MyInput ref={ref} />
```

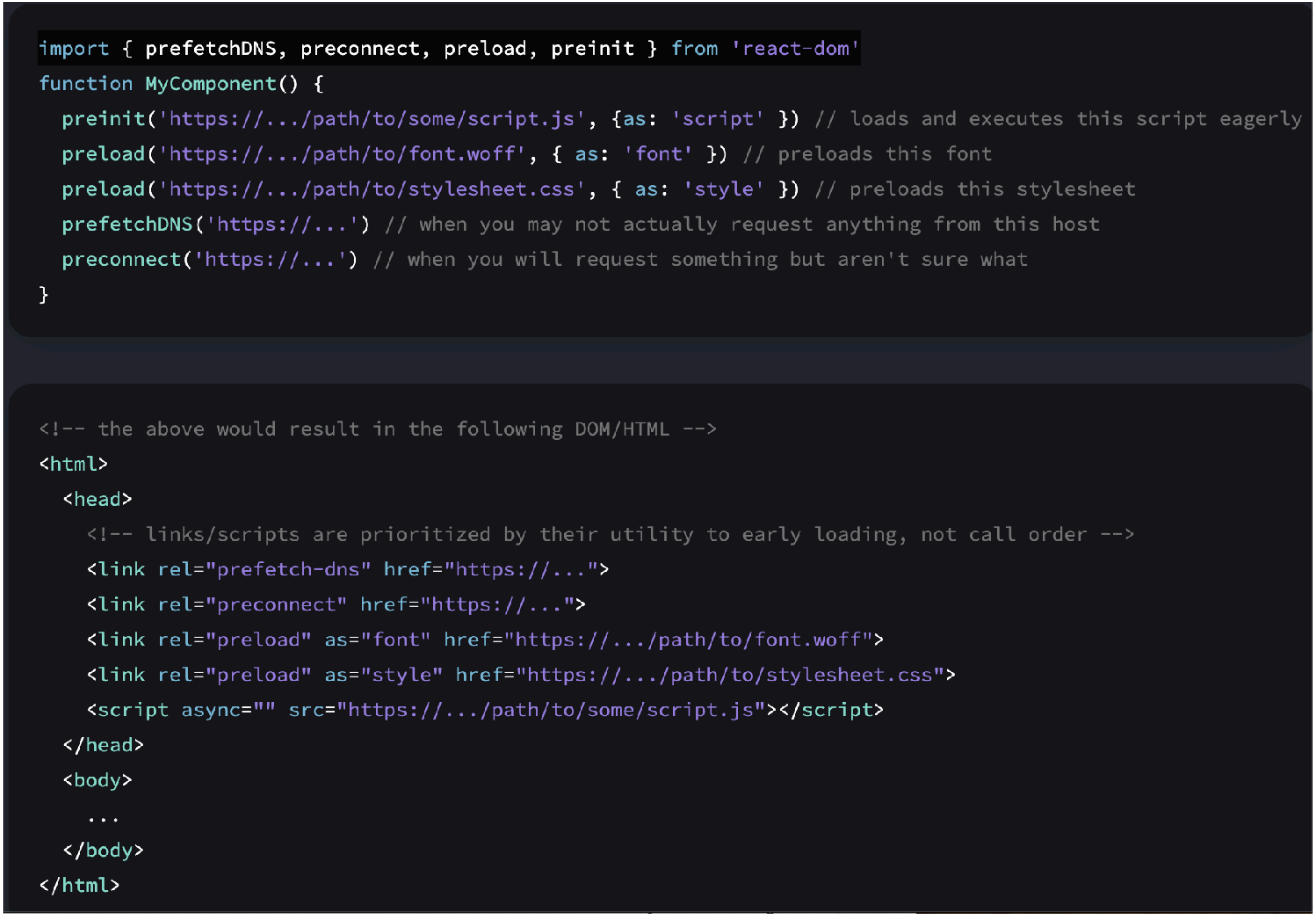
Document Metadata

Hoists your metadata

- New metadata feature removes the need to use tools like Helmet to set your head elements
- Useful for setting <title>, <link> and <meta> tags in your components
- Support for stylesheets
- This is great for feature for SEO
- Support for Async script tags

Preloading Resources

- Give the browser specific instructions about resources it will need
- Improve the performance of resources on page load
- import { prefetchDNS, preconnect, preload, preinit } from ‘react-dom'



```javascript
import { prefetchDNS, preconnect, preload, preinit } from 'react-dom'
function MyComponent() {
  preinit('https://.../path/to/some/script.js', {as: 'script' }) // loads and executes this script eagerly
  preload('https://.../path/to/font.woff', { as: 'font' }) // preloads this font
  preload('https://.../path/to/stylesheet.css', { as: 'style' }) // preloads this stylesheet
  prefetchDNS('https://...') // when you may not actually request anything from this host
  preconnect('https://...') // when you will request something but aren't sure what
}
```

```html
<!-- the above would result in the following DOM/HTML -->
<html>
  <head>
    <!-- links/scripts are prioritized by their utility to early loading, not call order -->
    <link rel="prefetch-dns" href="https://...">
    <link rel="preconnect" href="https://...">
    <link rel="preload" as="font" href="https://.../path/to/font.woff">
    <link rel="preload" as="style" href="https://.../path/to/stylesheet.css">
    <script async="" src="https://.../path/to/some/script.js"></script>
  </head>
  <body>
    ...
  </body>
</html>
```

Demo

Questions?

SCAN ME