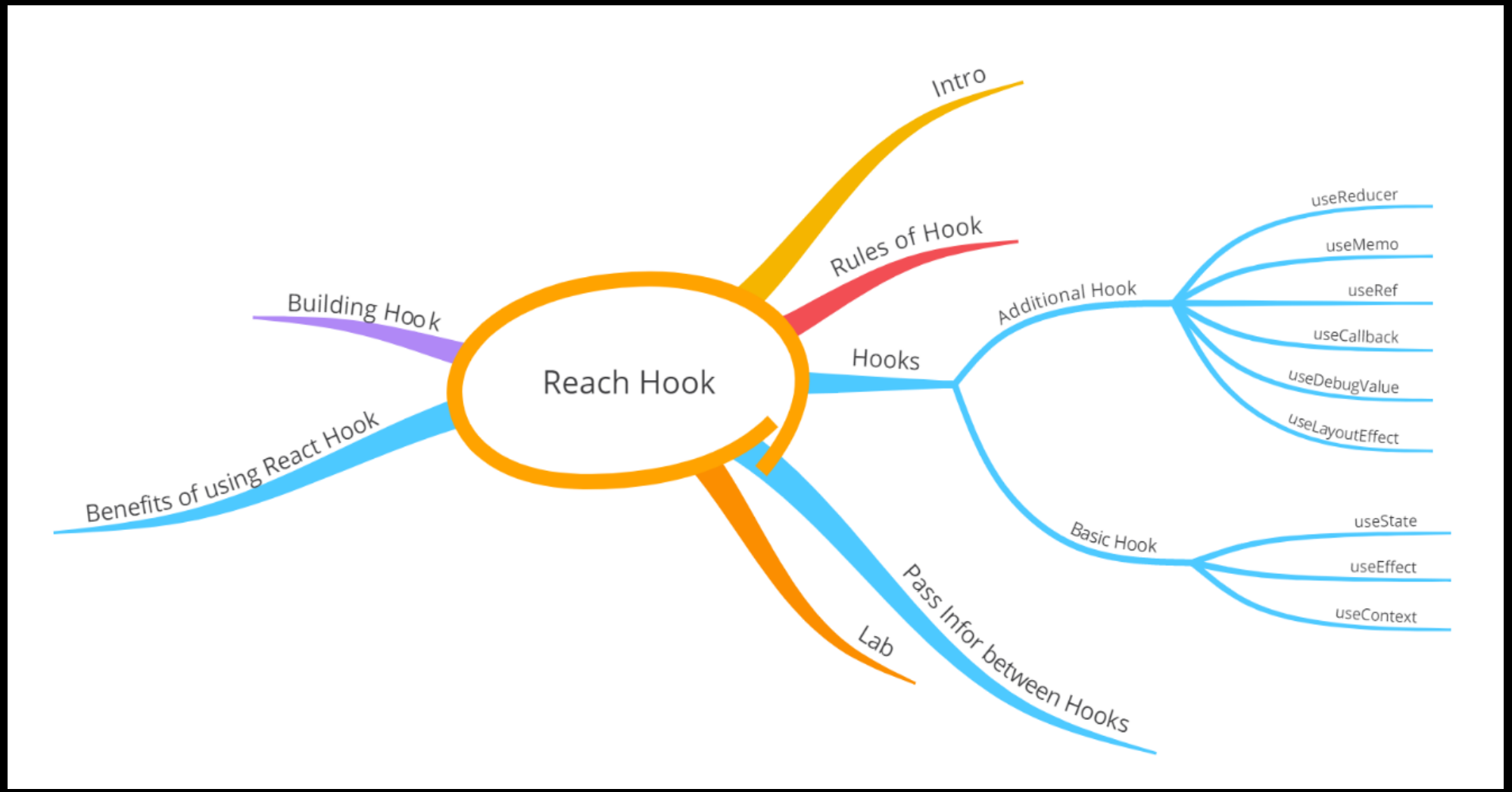
SESSION 6

REACT HOOK

OBJECTIVES


Reach Hook
Intro
Rules of Hook
Hooks
Additional Hook
useReducer
useMemo
useRef
useCallback
useDebugValue
useLayoutEffect
Basic Hook
useState
useEffect
useContext
Pass Infor between Hooks
Lab
Building Hook
Benefits of using React Hook

INTRODUCTION TO HOOKS

- ***Hooks*** are a new addition in **React 16.8**. They let you **use state** and other React features **without writing a class**.
- **Hooks are functions** that let you “hook into ” React state and lifecycle features from **function components**.

INTRODUCTION TO HOOKS

- Before the release of Hooks, React Components were divided into two broad categories depending on whether the component was class-based or function-based.
  - **React Hooks are a way to add React.Component features** to functional components. Features like:

    State

    Lifecycle

BENEFITS OF HOOK

- Improved code reuse
- Better code composition
- Better defaults
- Sharing non-visual logic with the use of custom hooks.
- Flexibility in moving up and down the components tree.

THE BENEFITS OF USING HOOKS

- Hooks have a lot of benefit to us as developers, and they are going to change the way we write components for the better.
- They already help us to write clearer and more concise code - it's like we went on a code diet and we lost a lot of weight and we look better and feel better.
- Not only is the code a lot smaller — the saved space certainly adds up for larger components — it’s also a lot more readable, which is a huge advantage of Hooks.

DEMO

```
import './App.css';
import React, { useState } from 'react';

const HooksExample = () => {
    const [counter, setCount] = useState(0);

    return (
        <div className="App">
            <header className="App-header">
                The button is pressed: { counter } times.
                <button
                    onClick={() => setCount(counter + 1)}
                    style={{ padding: '1em 2em', margin: 10 }} >
                    Click me!
                </button>
            </header>
        </div>
    )
}
export default HooksExample;
```

DEMO

index.js > ...

You, a few seconds ago | 1 author (You)

No Hook

```
import React, { Component } from 'react'

You, a few seconds ago | 1 author (You)
export default class Button extends Component {
  constructor () {
    super()
    this.state = { buttonText: 'Click me, please' }
    this.handleClick = this.handleClick.bind(this)
  }

  handleClick () {
    this.setState(() => {
      return { buttonText: 'Thanks, been clicked!' }
    })
  }

  render () {
    const { buttonText } = this.state
    return <button onClick={this.handleClick}>{buttonText}</button>
  }
}
```

index.js > ...

Hook

with Hook

```
import React, { useState } from 'react'

export default function Button() {

  const [buttonText, setButtonText] = useState('Click me, please')

  return (
    <button onClick={() => setButtonText('Thanks, been clicked!')}>
      {buttonText}
    </button>
  )
}
```

BASIC REACT HOOKS

- useState()
- useEffect()
- useContext()
- useReducer()

BASIC REACT HOOKS

- The useState() hook allows React developers to update, handle and manipulate state inside functional components without needing to convert it to a class component.
- Let's use the code snippet below is a simple Age counter component and we will use it to explain the power and syntax of the useState() hook.

```
function App() {
  const [age, setAge] = useState(19);
  const handleClick = () => setAge(age + 1)

  return
      <div>
          I am {age} Years Old
        <div>
        <button onClick={handleClick}>Increase my age! </button>
      </div>
   </div>
}
```

Hi, I am 19 Years old Increase my Age!

BASIC REACT HOOKS

- The useEffect() hook accepts a function that would contain effectual code.

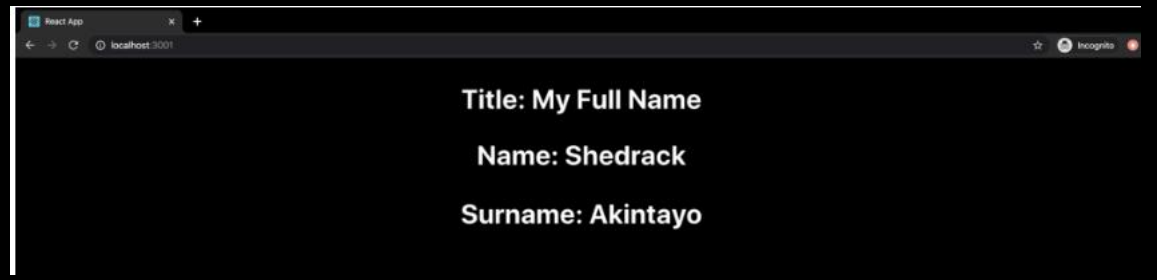
- If you're familiar with React class lifecycle methods, you can think of **useEffect Hook** as **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** combined.

STEP 1: DEFINE THE STATE OF YOUR APPLICATION


```
import React, {useState} from 'react';
function App() {
    //Define State
    const [name, setName] = useState({firstName: 'name', surname: 'surname'});
    const [title, setTitle] = useState('BIO');

    return(
        <div>
            <h1>Title: {title}</h1>
            <h3>Name: {name.firstName}</h3>
            <h3>Surname: {name.surname}</h3>
        </div>
    );
};
export default App
```

STEP 2: CALL THE USEEFFECT HOOK

```
import React, {useState, useEffect} from 'react';
function App() {
    //Define State
    const [name, setName] = useState({firstName: 'name', surname: 'surname'});
    const [title, setTitle] = useState('BIO');

    //Call the use effect hook
    useEffect(() => {
      setName({FirstName: 'Shedrack', surname: 'Akintayo'})
    }, [])//pass in an empty array as a second argument

    return(
        <div>
            <h1>Title: {title}</h1>
            <h3>Name: {name.firstName}</h3>
            <h3>Surname: {name.surname}</h3>
        </div>
    );
};
export default App
```



BASIC REACT HOOKS

useState, as the name describes, is a hook that allows you to use state in your function. We define it as follows:

const [**someState**, **updateState**] = useState(**initialState**)

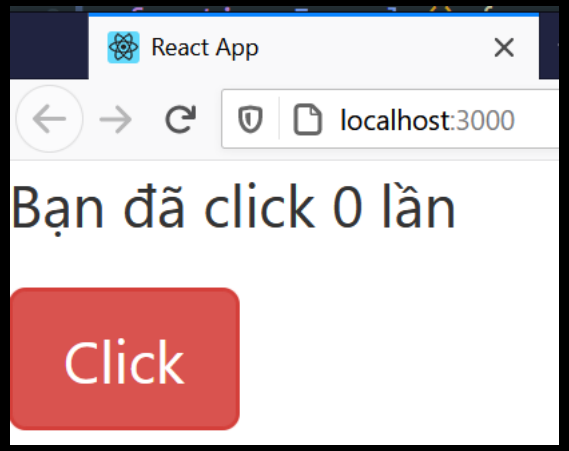
Let's break this down:

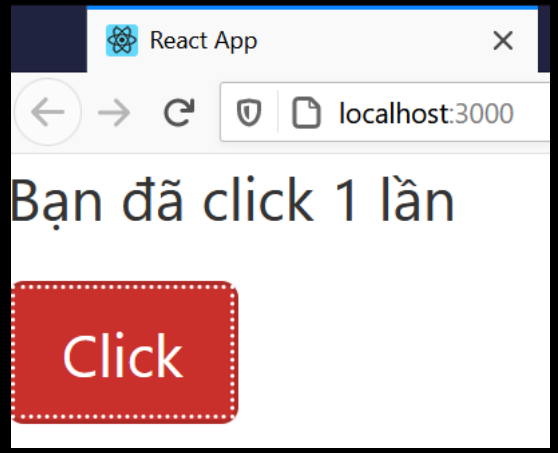
someState: lets you access the current state variable, someState
updateState: function that allows you to update the state — whatever you pass into it becomes the new someState
initialState: what you want someState to be upon initial render

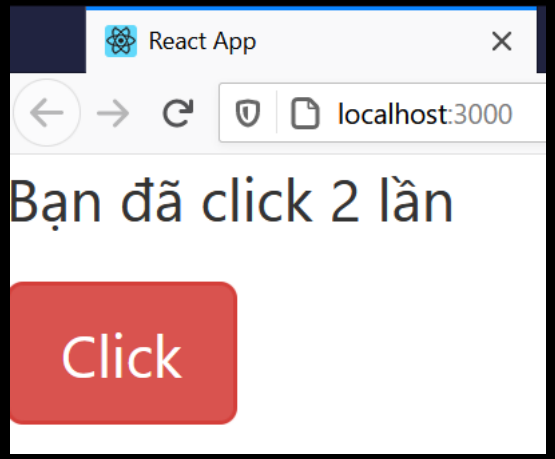
FIVE IMPORTANT RULES FOR HOOKS

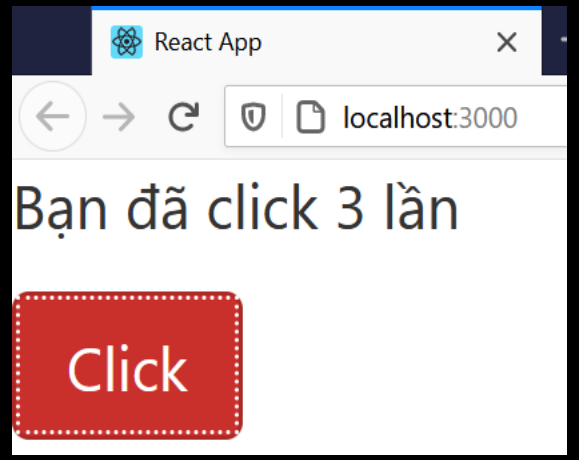
- Never call Hooks from inside a loop, condition or nested function
- Hooks should sit at the top-level of your component.
- Only call Hooks from React functional components.
- Never call a Hook from a regular function
- Hooks can call other Hooks

DEMO REACT HOOK









DEMO REACT HOOK

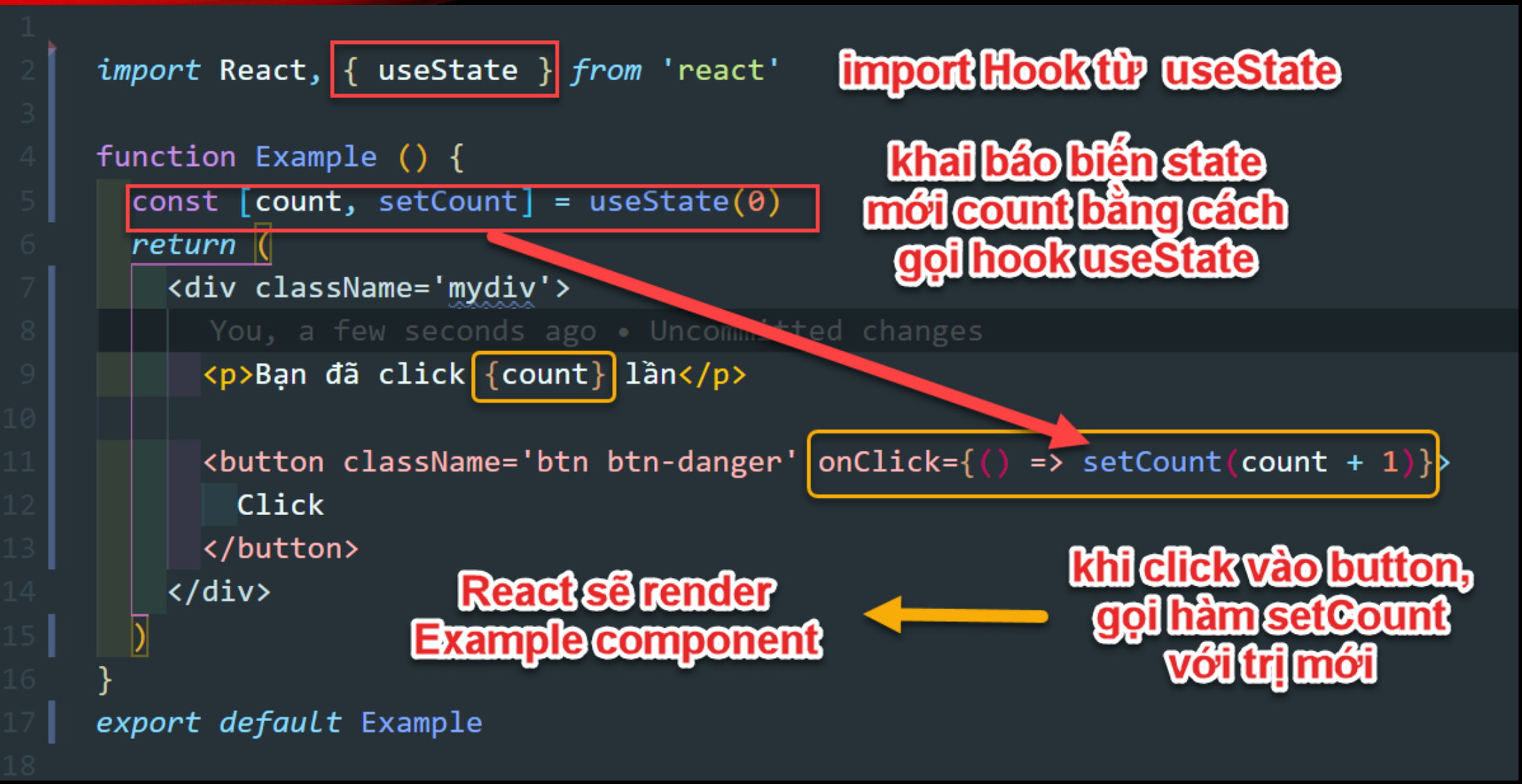
App.js

```
import React from 'react';
import './App.css';
class Example extends React.Component {
constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

  render() {
    return (
      <div>
        <p>Bạn đã bấm {this.state.count} lần</p>

        <button onClick={() => this.setState({ count: this.state.count + 1 })}>

          Click Me
        </button>
      </div>
    );
  }
}
export default Example
```

Sử dụng class component và state

We'll start learning about Hooks by comparing this code to an equivalent class example.

DEMO REACT HOOK



USEEFFECT

useEffect is another hook that handles **componentDidUpdate, componentDidMount,** and **componentWillUnmount** all in one call.
If you need to fetch data, for example, you could useEffect to do so, as seen below.

The component becomes a function and fetch gets called inside useEffect. Moreover, **instead of calling this.setState** I can **use setData** (an arbitrary function extracted from useState):

USEEFFECT()

```jsx
import React, { useState, useEffect } from 'react'

export default function DataLoader () {
  const [data, setData] = useState([])

  useEffect(() => {
    fetch('http://localhost:3001/links/')
      .then(response => response.json())
      .then(data => setData(data))
  })

  return (
    <div>
      <ul>
        {data.map(el => (
          <li key={el.id}>{el.title}</li>
        ))}
      </ul>
    </div>
  )
}
```

USEEFFECT()

`componentDidUpdate`! it's a lifecycle method which runs every time a component gets new props, or a state change happens. That's the trick. If you call `useEffect` as I did you would see an infinite loop.

For fixing this "bug" you would need to pass an empty array as a second argument to `useEffect`:

```
//
  useEffect(() => {
    fetch("http://localhost:3001/links/")
      .then(response => response.json())
      .then(data => setData(data));
  }, []); // << super important array

//
```

USEEFFECT()

Instead of HOCs and render props, we can encapsulate our logic in a React hook and then import that hook whenever we feel the need. In our example we can create a custom hooks for fetching data.

A custom hook is a JavaScript function whose name starts with "use", as a convention. Easier done than said. Let's make a `useFetch` hook then:

```
// useFetch.js
import { useState, useEffect } from "react";

export default function useFetch(url) {
  const [data, setData] = useState([]);

  useEffect(() => {
    fetch(url)
      .then(response => response.json())
      .then(data => setData(data));
  }, []);

  return data;
}
```

USEEFFECT()

- This is how you would use the custom hook:

```
import React from 'react'        You, a few seconds ago • Unc
import useFetch from './useFetch'

export default function DataLoader (props) {
  const data = useFetch('http://localhost:3001/links/')
  return (
    <div>
      <ul>
        {data.map(el => (
          <li key={el.id}>{el.title}</li>
        ))}
      </ul>
    </div>
  )
}
```

THE END