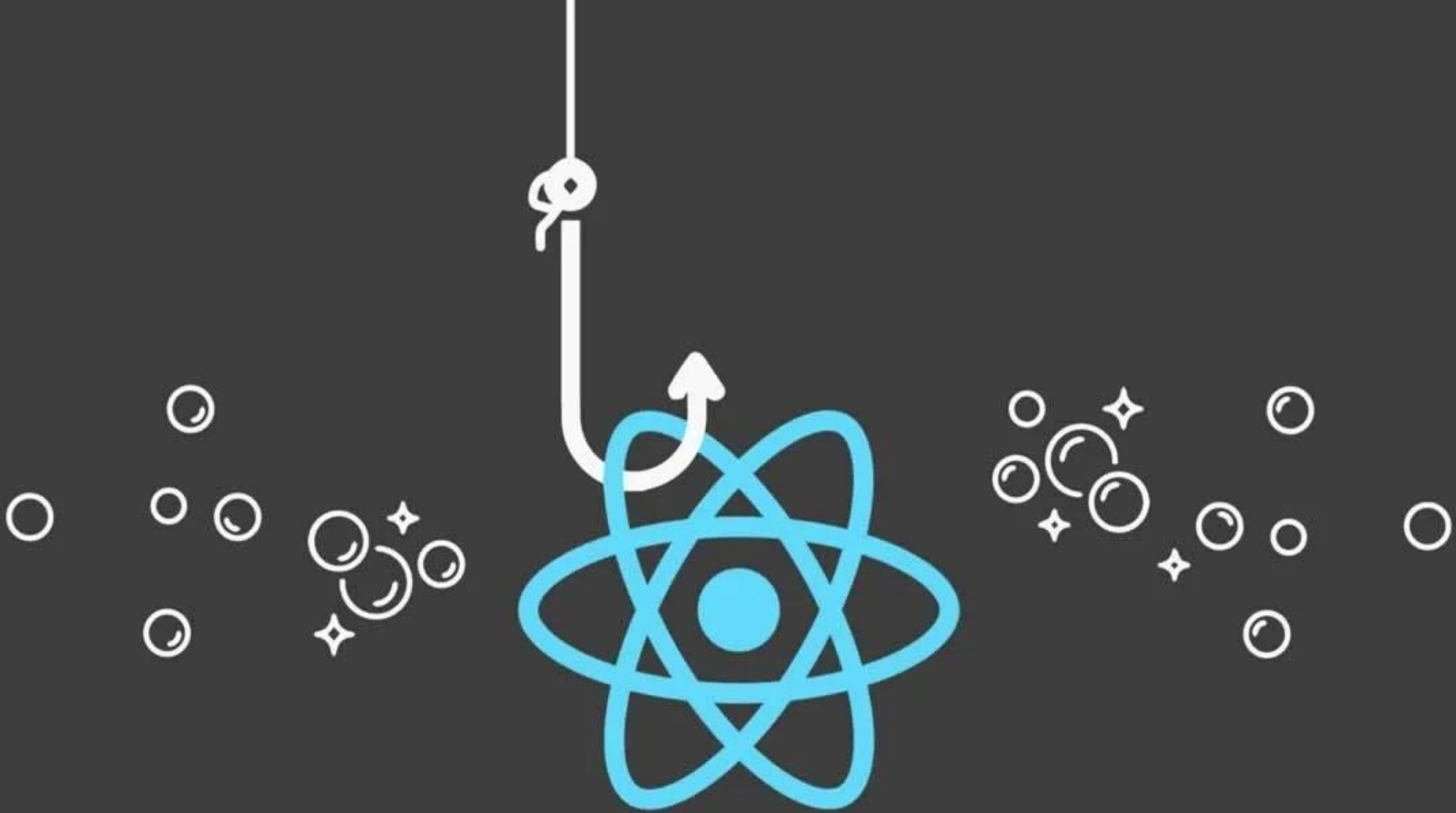


Welcome to Lecture 26

Agenda

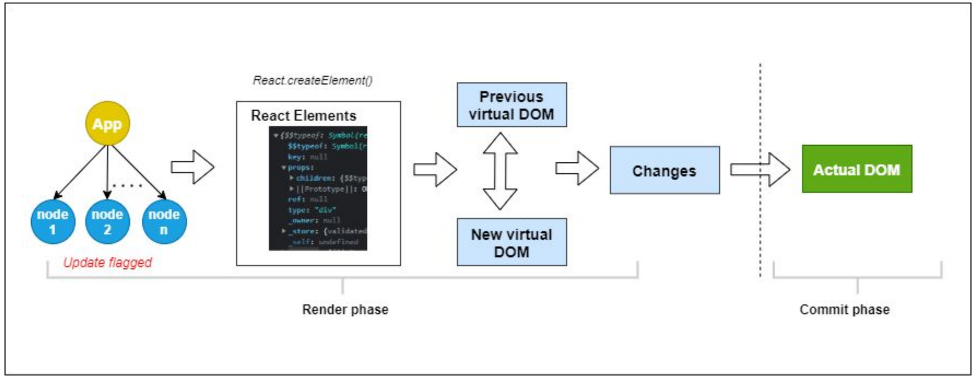
Session Objectives

- Introduction to Hooks
- React Hooks
  - useState
  - useEffect
- API calling in React within useEffect
- React Forms
- Quiz

The "Why" - From Classes to Functions

What Does It Really Mean to "Render"?

- We know a component renders when its state changes. But what is React actually doing?
- A "render" is when React calls your component function
- Your function returns a blueprint of the UI (JSX)
- React compares this new blueprint to the old one using its Virtual DOM
- It then calculates the absolute minimum DOM changes needed and applies only those to the real browser DOM
- **Key Idea**: Rendering is a highly optimized and intelligent update process, not a brute-force rewrite

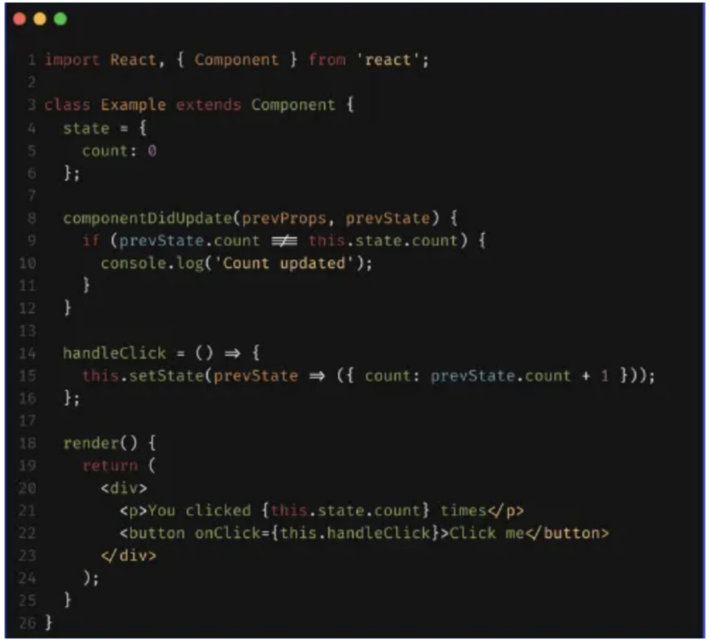


```
▾{$$typeof: Symbol(re
  $$typeof: Symbol(r
  key: null
 ▾props:
  ▸children: {$$typ
  ▸[[Prototype]]: O
  ref: null
  type: "div"
  _owner: null
 ▸_store: {validated
  _self: undefined
```

Source

Why the Shift Away from Class Components?

- **Complexity and Boilerplate:** Class components require more code (constructor, super, render method) for simple tasks
- **The** this **Keyword**: As we saw, this in JavaScript can be confusing. In classes, you constantly need to remember to .bind(this) in the constructor to ensure methods have the correct context when used as event handlers. This is a huge source of bugs for beginners and experts alike.
- **Logic Sharing**: It was difficult to share stateful logic between different class components. You often had to resort to complex patterns like "render props" or "higher-order components."

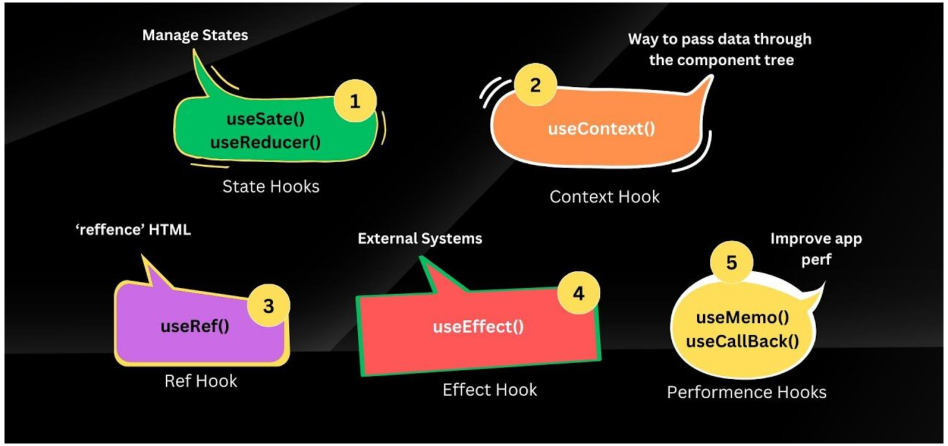


```
import React, { Component } from 'react';

class Example extends Component {
  state = {
    count: 0
  };

  componentDidUpdate(prevProps, prevState) {
    if (prevState.count !== this.state.count) {
      console.log('Count updated');
    }
  }

  handleClick = () => {
    this.setState(prevState => ({ count: prevState.count + 1 }));
  };

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button onClick={this.handleClick}>Click me</button>
      </div>
    );
  }
}
```

Source

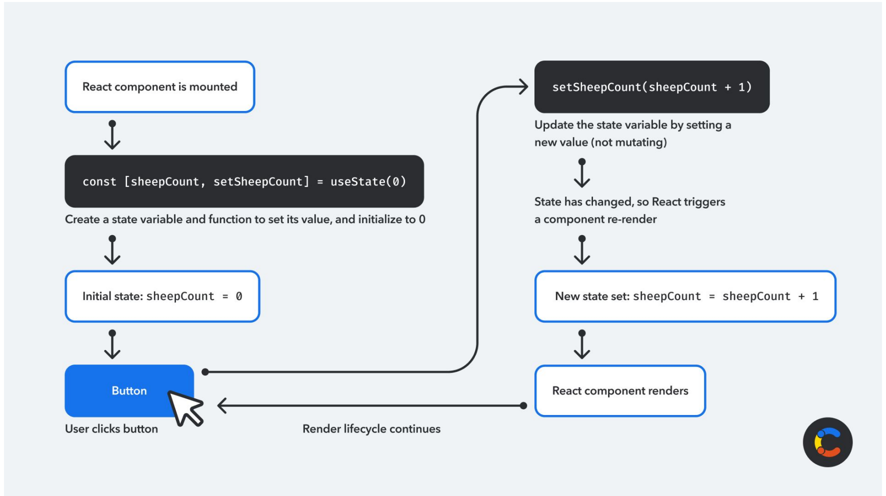
What Are Hooks?

- Hooks are special functions, always starting with use, that let you "hook into" React features (like state) from within functional components
- They allow us to have stateful logic in simple JavaScript functions, solving all the problems of class components
- **The Benefit**: We can now write our entire application with simpler, cleaner functional components without losing any power



The useState Hook

- The most important hook. It lets you add a single piece of state to a component.
- How is it used?
  - const [stateVariable, setStateFunction] = useState(initialValue);
- It takes one argument: the initial value of the state.
- It returns an array containing exactly two things:
  - The current value of the state.
  - The setter function, which you use to update the state.



React component is mounted
const [sheepCount, setSheepCount] = useState(0)
Create a state variable and function to set its value, and initialize to 0
Initial state: sheepCount = 0
Button
User clicks button
setSheepCount(sheepCount + 1)
Update the state variable by setting a new value (not mutating)
State has changed, so React triggers a component re-render
New state set: sheepCount = sheepCount + 1
React component renders
Render lifecycle continues

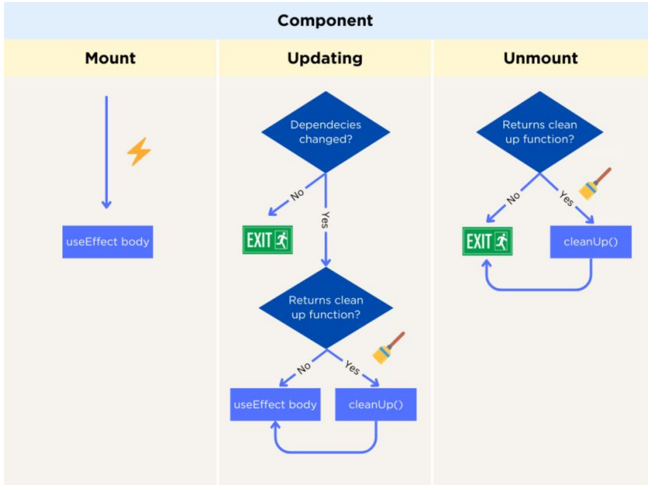
Source

Demo!

useEffect & Rendering Data

useEffect: Interacting with the Outside World

- What is a "Side Effect"?
  - Any work in your component that affects something outside of itself
  - Examples: fetching data from an API, using timers or performing manual DOM manipulation
- The useEffect Hook is the correct place any code with side effects. It synchronizes your component with an external system
- Syntax:
  - useEffect(() => { /* effect code */ }, [dependencyArray]);



Source

The Dependency Array: Controlling Your Effects

- The second argument to useEffect is the dependency array
- It tells React when to re-run your effect
  - [] (Empty Array): The effect runs only once, right after the component first mounts. Perfect for our initial API call.
  - [someValue]: The effect runs after the first render, AND any time someValue changes.
  - No Array (Omitted): The effect runs after every single render. This can be the cause of infinite loops

Demo!

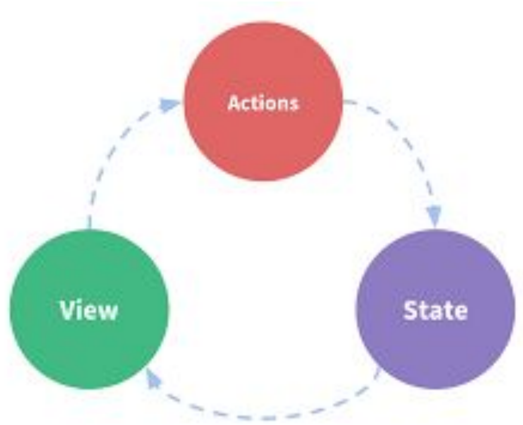
Understanding the key property

- When you render a list of items with .map(), React needs a way to uniquely identify each item in the list
- The key prop gives each element a stable identity
- Why?
  - When the list changes (items are added, removed, or re-ordered), React uses the key to quickly identify what changed
  - Without it, React has to re-evaluate the entire list, which is very inefficient and can lead to bugs
- **Rule**: Keys must be unique among siblings and should be stable
  - Don't use Math.random() or the array index if the list can be re-ordered
  - An item's unique id from the database is the perfect key

React Forms

React Forms: The "Controlled Component"

- In React, the DOM doesn’t handle form state. The React component "controls" the form input
- The "Single Source of Truth": The component's state is the master source of data
- The Two-Way Binding
  - The value of the input is set from a state variable. Eg: value={myState}
  - When the user types, the onChange event fires
  - The onChange handler calls the setState function to update the state variable (onChange={e => setMyState(e.target.value)})



Source

Demo!

That’s it for today.
Questions?

Quiz!

Question 1

- What is the primary reason to use React Hooks?
    - A) To allow styling of components without using CSS.
    - B) To replace the need for JavaScript in React applications.
    - C) To let developers use state and lifecycle features in functional components, avoiding the complexity of classes
    - D) To automatically handle API data fetching and form submissions

Question 2

- If a useEffect hook to fetch initial data for a component is being called repeatedly in an infinite loop, what is the common cause of this bug?
    - A) The API server is sending back an error.
    - B) The useState setter function is named incorrectly.
    - C) The component is not wrapped in a <React.StrictMode> tag
    - D) The useEffect hook is missing its dependency array, causing it to run after every single render

Question 3

- When rendering a list of items using the .map() method, why is it critical to provide a key prop to each element?
  - A) The key prop is used to apply CSS styles to each list item.
  - B) The key provides a stable identity for each item, allowing React to efficiently update, add, or remove elements without re-rendering the entire list.
  - C) The key prop must be set to the array index for the list to render correctly.
  - D) The key prop is a security feature to prevent data injection.

Question 4

- What does the useState hook return?
  - A) A single value representing the current state.
  - B) An object containing the state and several lifecycle methods.
  - C) A function that, when called, returns the current state value.
  - D) An array containing two elements: the current state value and a function to update it.

Question 5

- In a "Controlled Component" form pattern, what is considered the "single source of truth" for the input field's value?
  - A) The DOM element's internal state.
  - B) A variable stored in localStorage.
  - C) The React component's state, managed by useState
  - D) A prop passed down from a parent component

Have a
good one!