Event Handling

HTML vs React Events

```html
<!-- HTML -->
<button onclick="alert('Clicked!')">Click Me</button>
```

```jsx
<button onClick={() => alert('Clicked!')}>Click Me</button>
```

Key Differences:

- ❑ React uses camelCase (onClick, onChange, etc.)
- ❑ Pass a function reference, not a string
- ❑ React events work across browsers using SyntheticEvent (React’s wrapper)

Event Handler Basics

Handle basic events like click, change, and submit.

```
function App() {
  function handleClick() {
    alert('Button clicked!');
  }

  return <button onClick={handleClick}>Click Me</button>;
}
```

Common Events:

- ❑ onClick
- ❑ onChange
- ❑ onSubmit
- ❑ onMouseEnter / onMouseLeave
- ❑ onKeyDown / onKeyUp

Accessing Event Object

```
function InputBox() {
  function handleChange(e) {
    console.log(e.target.value); // Access input value
  }

  return <input type="text" onChange={handleChange} />;
}
```

SyntheticEvent:

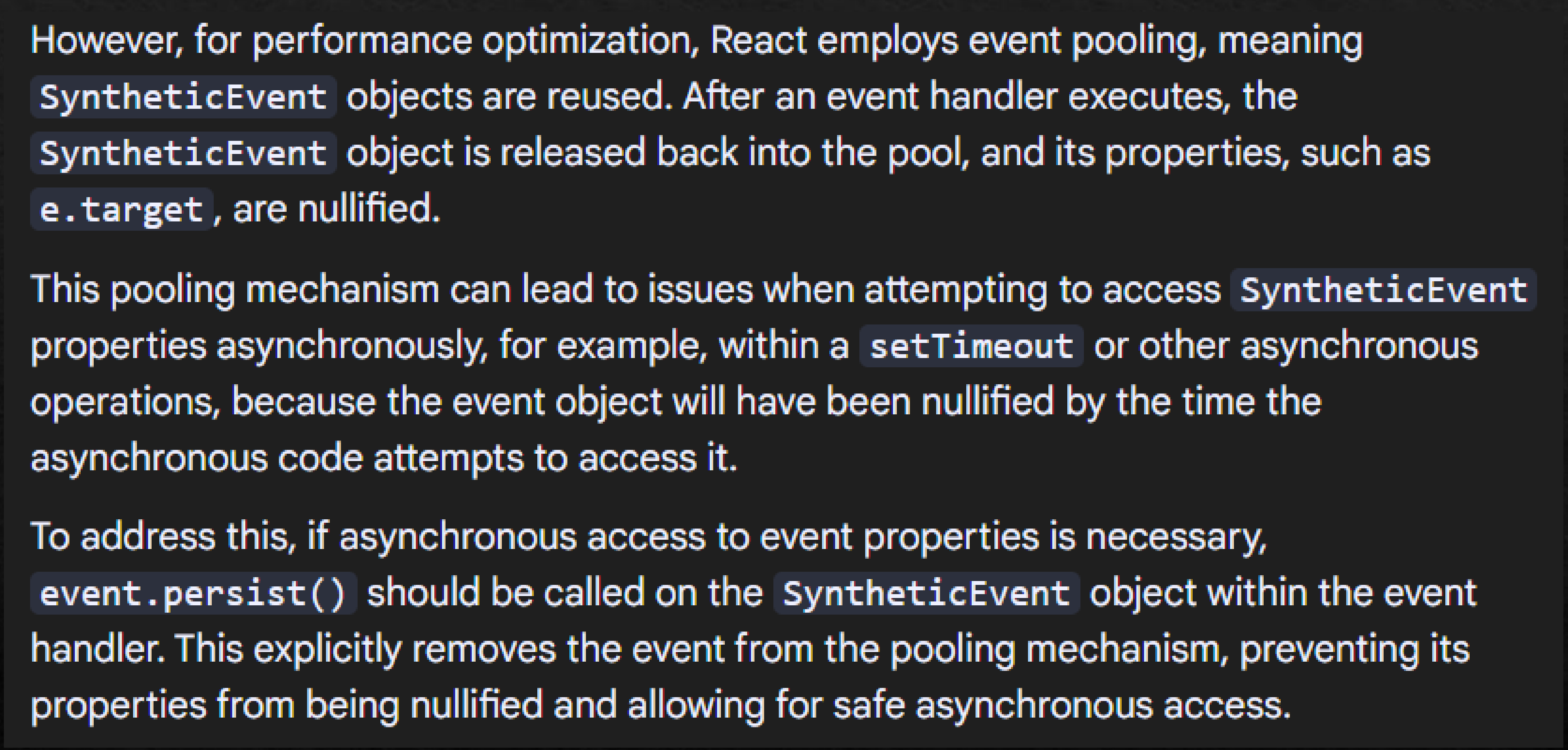
Event handling in React follows a similar pattern to standard javaScript event handling but with some key differences, such as using synthetic events for cross-browser compatibility and providing consistent event handling across different elements and browsers.

When you handle events in React, like clicking a button or typing in an input box, React wraps the native browser events in something called Synthetic Event. This Synthetic Event ensures consistent event handling across different browsers.

But after the event handler runs, React pools the event for performance (so e.target may become null in async code).

```
function handleChange(e) {
  setTimeout(() => console.log(e.target.value), 1000); // ❌ e.target is null
}
```

In React 16 and later versions, this problem is no more.



However, for performance optimization, React employs event pooling, meaning `SyntheticEvent` objects are reused. After an event handler executes, the `SyntheticEvent` object is released back into the pool, and its properties, such as `e.target`, are nullified.

This pooling mechanism can lead to issues when attempting to access `SyntheticEvent` properties asynchronously, for example, within a `setTimeout` or other asynchronous operations, because the event object will have been nullified by the time the asynchronous code attempts to access it.

To address this, if asynchronous access to event properties is necessary, `event.persist()` should be called on the `SyntheticEvent` object within the event handler. This explicitly removes the event from the pooling mechanism, preventing its properties from being nullified and allowing for safe asynchronous access.

Passing Arguments to Handlers

```
function App() {
  function handleClick(name) {
    alert(`Hello, ${name}`);
  }

  return <button onClick={() => handleClick('Manas')}>Greet</button>;
}
```

Beginner Mistake:

```
❌ onClick={handleClick('Manas')} → runs immediately
✅ onClick={() => handleClick('Manas')} → runs on click
```

Using State with Events

```
function Counter() {
  const [count, setCount] = useState(0);

  function handleIncrement() {
    setCount(prev => prev + 1);
  }

  return (
    <button onClick={handleIncrement}>
      Count: {count}
    </button>
  );
}
```

- React re-renders on setState
- Event handlers capture the current state snapshot

Pitfall: State updates are asynchronous, so never rely on immediate new values.

Prevent Default & Stop Propagation

Prevent Default

```
function Form() {
  function handleSubmit(e) {
    e.preventDefault(); // prevent form reload
    console.log("Form submitted");
  }

  return (
    <form onSubmit={handleSubmit}>
      <button>Submit</button>
    </form>
  );
}
```

Stop Propagation

```
function Parent() {
  function handleParent() {
    alert('Parent clicked');
  }

  function handleChild(e) {
    e.stopPropagation(); // stops bubbling
    alert('Child clicked');
  }

  return (
    <div onClick={handleParent}>
      <button onClick={handleChild}>Click Child</button>
    </div>
  );
}
```

Common Beginner Mistakes Recap

Mistake	Why it Happens	Fix
Calling handler immediately	Using `onClick={fn()}`	Use arrow function
Losing event object	React pools SyntheticEvent	Use React 16 or 16+
Forgetting `e.preventDefault()`	Form reloads on submit	Call it inside handler
Using state directly after `setState`	State update is async	Use callback or `useEffect`
Not stopping event bubbling	Both parent & child trigger	Use `e.stopPropagation()`

Thank You