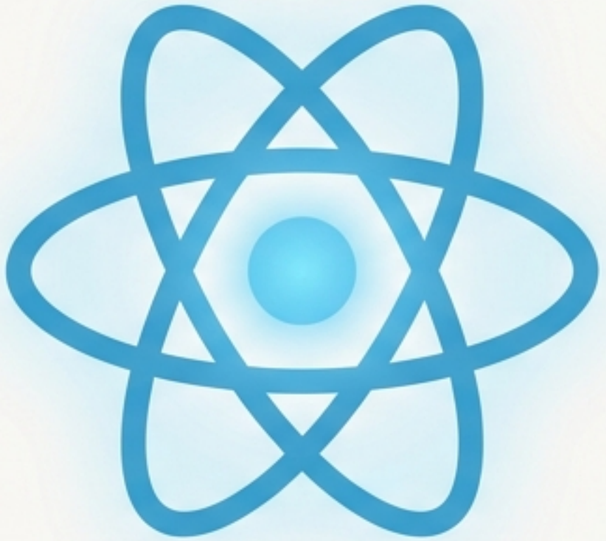
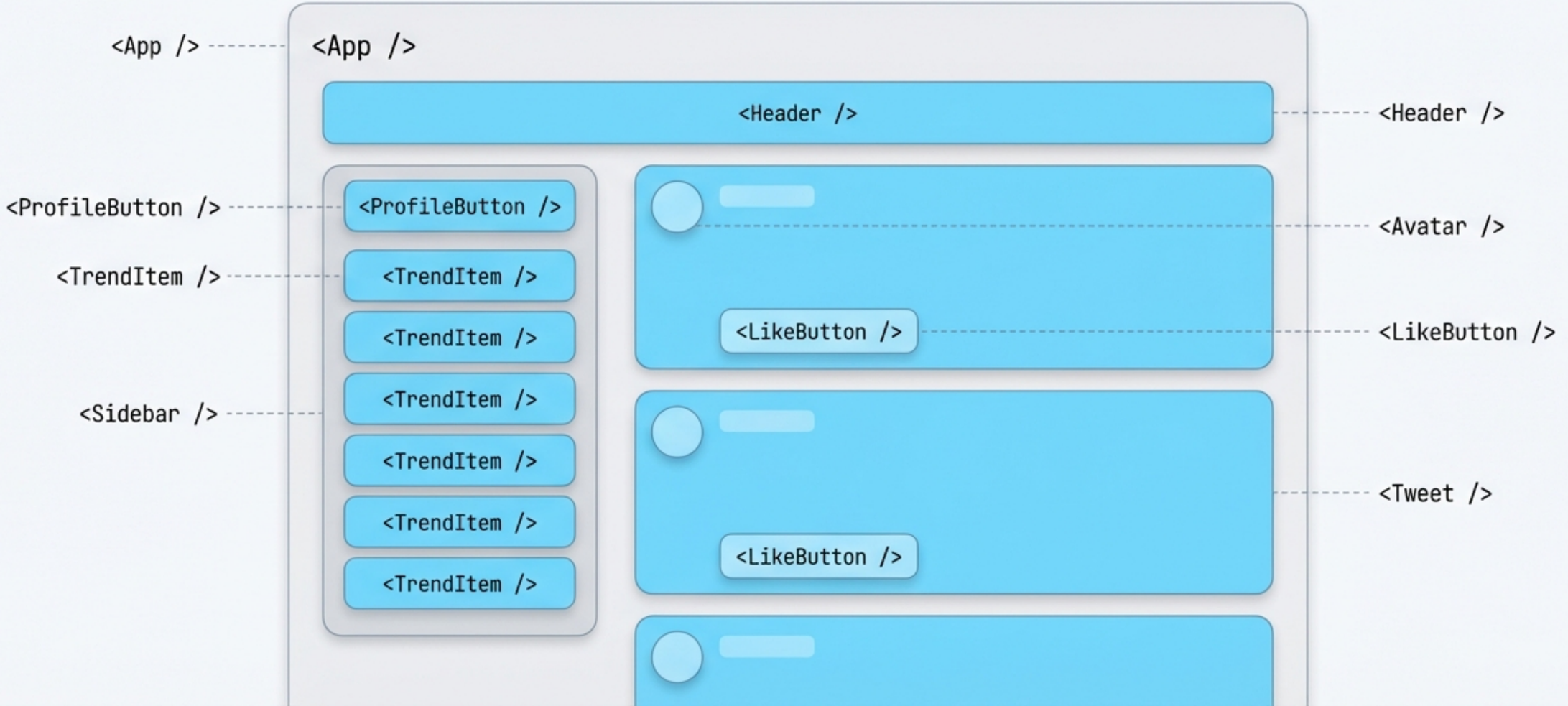


React: A Quick Start

A visual journey through the 80% of concepts you'll use every day.

React Apps are Built from Components.

A component is a reusable piece of the UI with its own logic and appearance. Think of them as custom HTML tags you create. A component can be as small as a button or as large as an entire page.



Components are JavaScript Functions that Return UI.

Core Idea

React components are simply JavaScript functions.

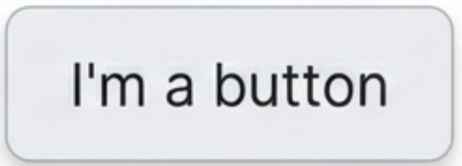
Rule 1

Their names **must** start with a capital letter (e.g., `MyButton`), which is how React distinguishes them from standard HTML tags (e.g., `button`).

Rule 2

They return the UI markup that should be rendered on the screen.

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



Writing UI with JSX: JavaScript and HTML, Together

The Concept

What is it?

JSX is a syntax extension for JavaScript that looks like HTML. It's optional, but nearly all React projects use it for its convenience.

Key Rule #1

Tags must be closed. For example, `<br>` becomes `<br />`.

Key Rule #2

You can only return a single root element from a component. To return multiple elements, wrap them in a shared parent like a `<div>...</div>` or an empty `<>...</>` fragment.

The Example

```
function AboutPage() {
  return (
    <>
      <h1>About</h1>
      <p>Hello there.<br />
       How do you do?</p>
    </>
  );
}
```

A fragment (<>) groups elements without adding an extra node to the DOM.

Self-closing tag.

Displaying Dynamic Data by ‘Escaping’ into JavaScript with {}

The Concept

Use curly braces {} inside your JSX to embed JavaScript variables, expressions, or function calls directly into your markup.

This works for both content (like text inside an <h1>) and attributes (like the src of an <img>). When used for attributes, curly braces replace the quotes.

The Example

```
const user = {
  name: 'Hedy Lamarr',
  imageUrl: 'https://i.imgur.com/yXOvdOSs.jpg'
};

// In the component's return:
<h1>{user.name}</h1>
<img
  className="avatar"
  src={user.imageUrl}
/>
```

Hedy Lamarr

Adding Styles to Components with CSS

The Concept

In React, you specify a CSS class using the `className` attribute.

This is the equivalent of the standard HTML `class` attribute (the name is different to avoid conflicts with the `class` keyword in JavaScript).

You then write your CSS rules in a separate `.css` file just as you normally would.

The Example

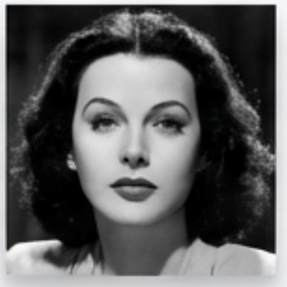
JSX Code

```jsx
<img className="avatar" />
```

CSS Code

```css
/* In your CSS file */
.avatar {
  border-radius: 50%;
}
```

Before



After

Showing Content Conditionally.

The Concept

There is no special React syntax for conditions. You use standard JavaScript logic.

- You can use an `if` statement outside of your JSX.
- For more compact, inline logic, use the ternary operator (`condition ? <A /> : <B />`).
- When you don’t need an `else` branch, you can use the logical AND operator (`condition && <A />`).

The Example

```
<div>
  {isLoggedIn ? (
    <AdminPanel />
  ) : (
    <LoginForm />
  )}
</div>
```

The ternary operator is a clean way to choose between two components to render.

Transforming Data into UI with the `.map()` Method.

The Concept

To render a list of components, use the standard JavaScript `.map()` array method. This function transforms each item in your data array into a JSX element.

Critical Insight

You must provide a unique `key` prop for each item in the returned list. The key should be a stable string or number (like a database ID) that uniquely identifies an item among its siblings. React uses keys to track items during re-renders, insertions, and deletions for performance optimization.

The Example

```
const products = [{ title: 'Cabbage',
id: 1 }, ...];
```

```
const listItems = products.map(product =>
  <li key={product.id}>
    {product.title}
  </li>
);

return (
  <ul>{listItems}</ul>
);
```

Always add a unique key!

Responding to User Actions with Event Handlers.

The Concept

Declare event handler functions directly inside your component.

Pass the function itself as a prop to the element. The prop name matches the event, like `onClick`, `onMouseEnter`, etc.

Important

You pass the function reference (e.g., `onClick={handleClick}`), you do not *call* the function (`onClick={handleClick()}`). React will call your handler when the event occurs.

The Example

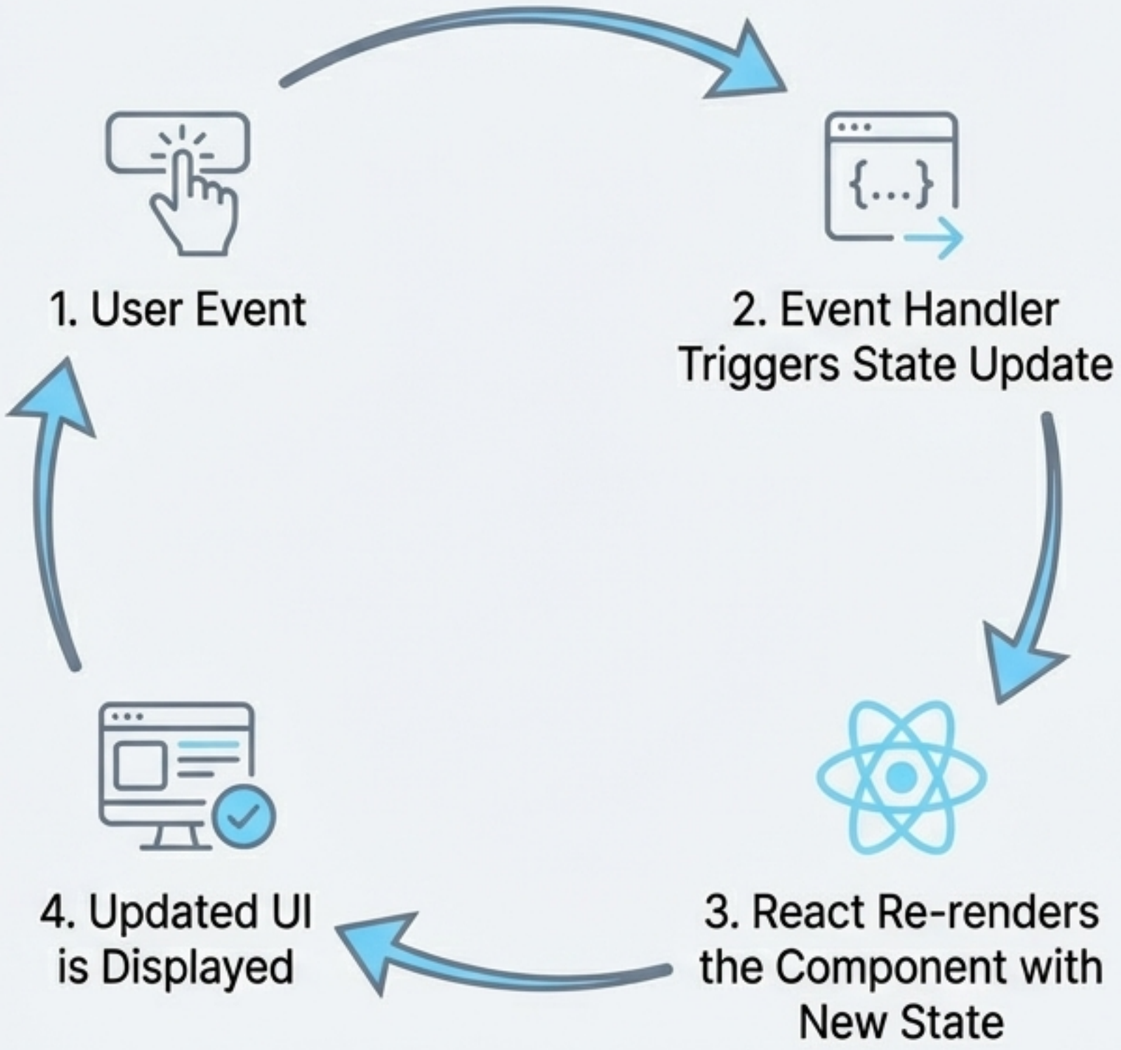
```
function MyButton() {
  function handleClick() {
    alert('You clicked me!');
  }

  return (
    <button onClick={handleClick}>
      Click me
    </button>
  );
}
```

Notice: no parentheses on handleClick! We are passing the function, not its result.

Giving Components a Memory with State.

Often, a component needs to ‘remember’ information that can change over time (like a counter or form input). This remembered information is called **state**.

When a component's state is updated, When a component's state is updated, React automatically re-renders the and its children, ensuring the UI always reflects the current state.



Using the `useState` Hook to Manage State.

The Syntax

Step 1: Import

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

Step 2: Declare

Inside your component, call `useState` to declare a state variable.

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

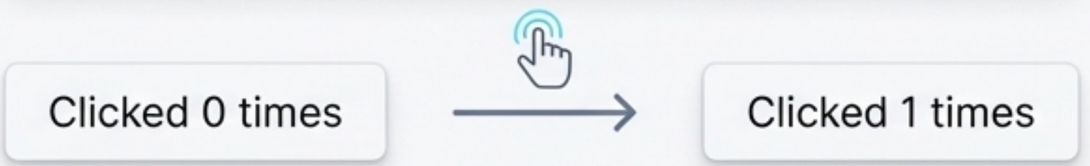
What you get

`useState` returns a pair:

- `count`: The current value of the state variable (with its initial value of `0`).
- `setCount`: The “setter” function you use to update the state.

The Example

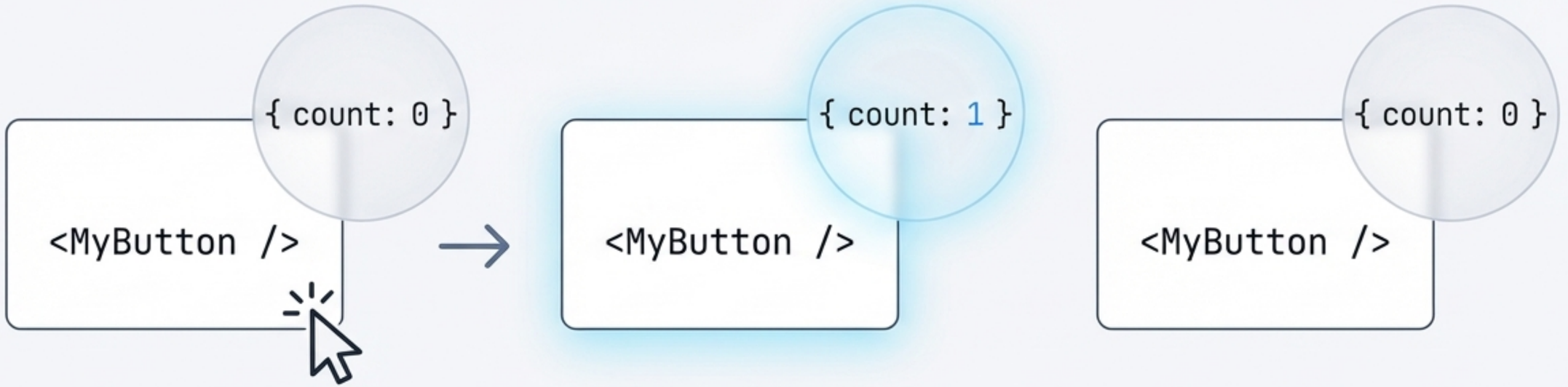
```
import { useState } from 'react';

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

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <button onClick={handleClick}>
      Clicked {count} times
    </button>
  );
}
```



The Challenge: How Do Components Share Data?

If you render the same component multiple times, each one gets its own independent state. Each button “remembers” its own count and doesn’t affect other buttons.

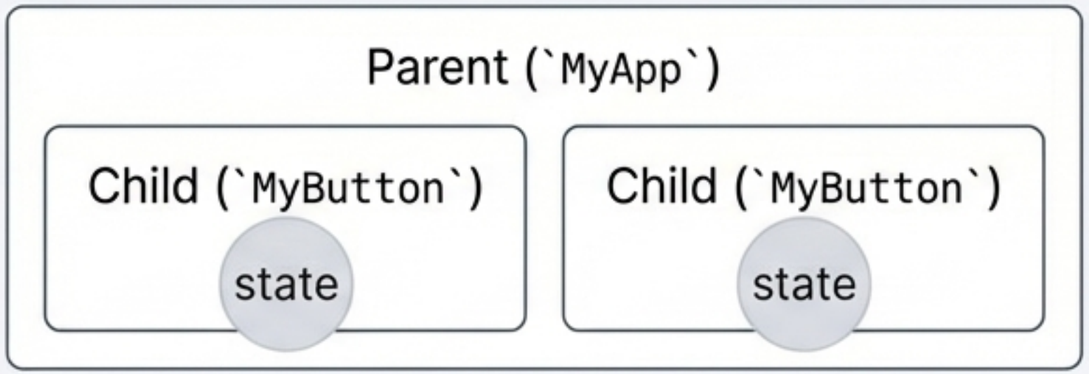
But what if we need components to share data and update together? For example, a single counter that is updated by multiple buttons.



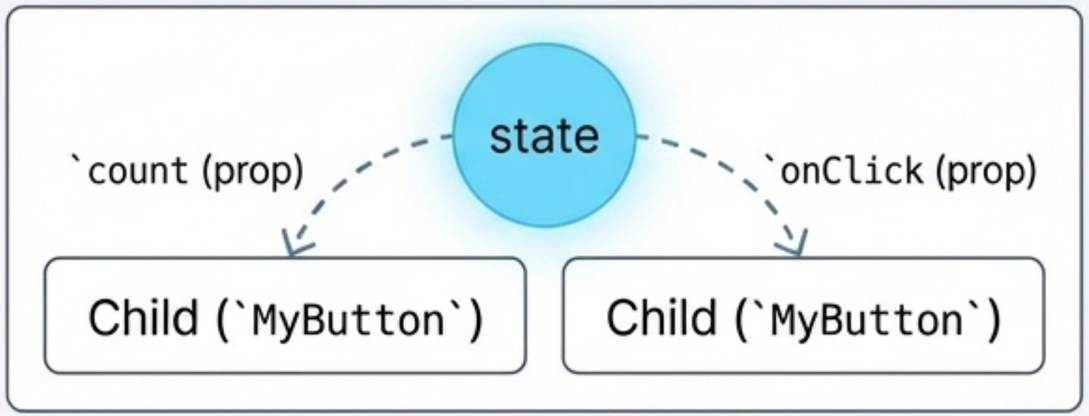
The Solution: Lift State Up to a Common Parent.

1. Identify the closest common parent component that contains all the components that need the shared data.
2. Move the state (the `useState` call) from the child components up into that common parent.
3. The parent then passes the state value down to the children as **props**.
4. The parent also passes the event handler function down as a prop, so the children can tell the parent when to update the state.

Before



After



Sending and Receiving Data with Props.

Parent Component (`MyApp`)

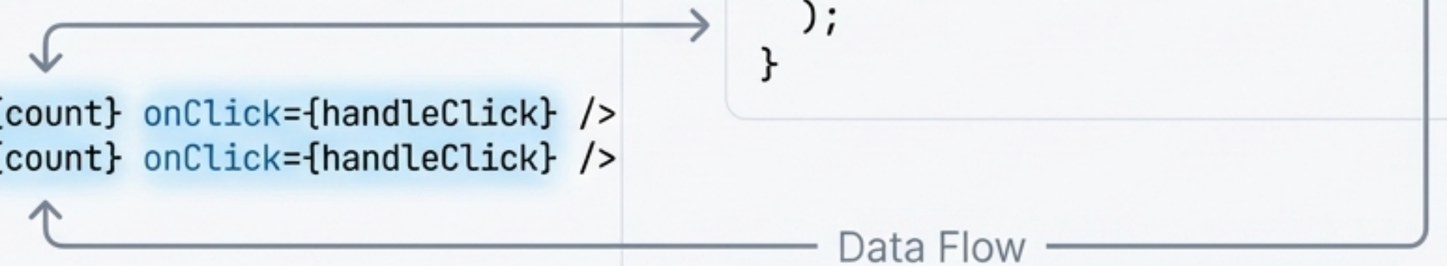
The parent now owns the state and the click handler. It passes them down to the child component using attributes in JSX. These are called “props”.

```
export default function MyApp() {
  const [count, setCount] = useState(0);

  function handleClick() {
    setCount(count + 1);
  }

  return (
    <div>
      <MyButton count={count} onClick={handleClick} />
      <MyButton count={count} onClick={handleClick} />
    </div>
  );
}
```

Child Component (`MyButton`)

The child component is now “controlled” by its parent. It receives its data and behavior via props. Props are received as the arguments to the component function, often destructured for convenience.

```
function MyButton({ count, onClick }) {
  return (
    <button onClick={onClick}>
      Clicked {count} times
    </button>
  );
}
```



You've Mastered the Core of React!

Recap

- **Components & JSX:** Building blocks of UI.
- **Displaying Data & Styles:** Using `{}` and `className`.
- **Conditional & List Rendering:** With JavaScript logic and `.map()`.
- **Events & State:** Making UIs interactive with `onClick` and `useState`.
- **Props & Sharing Data:** "Lifting state up" to communicate between components.

Next Steps

By now, you know the basics of how to write React code. The next step is to put it all into practice.

Check out the official React Tutorial to build your first mini-app (a Tic-Tac-Toe game).

react.dev/learn/tutorial-tic-tac-toe