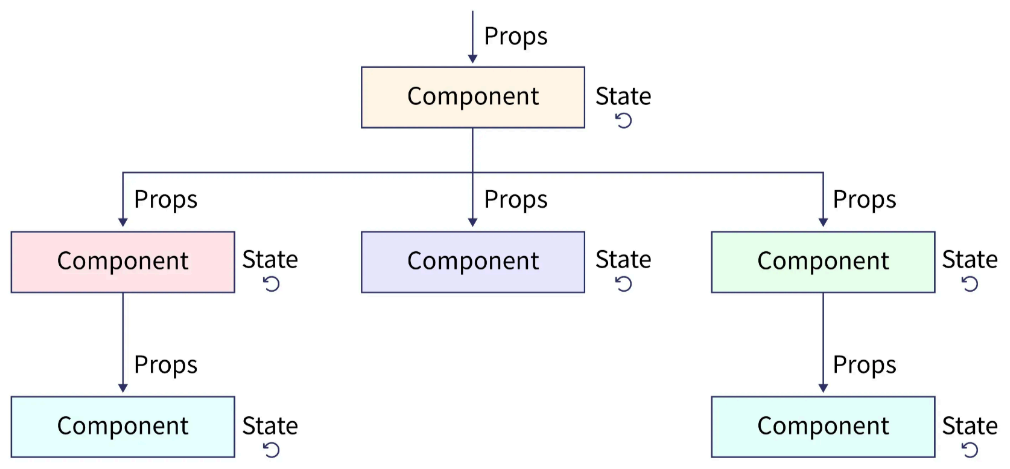
Sharing state between components

1. From parent to children
2. From children to parent
3. From children to sibling
4. From cousin to cousin

In React, **data flow** is a fundamental concept that determines how components communicate with each other.

React follows a **unidirectional data flow**, meaning data is passed from parent components to child components via **props**. However, there are several ways to pass data between components in different relationships, such as from children to parents, siblings, or even cousins.



1. From Parent to Children

This is the most common and straightforward way to pass data in React. Data flows from a **parent component** to a **child component** via **props**.

How It Works:

- The parent component defines the data and passes it to the child component as a **prop**.
- The child component receives the data as a **prop** and uses it to render or perform logic.

```
// Parent Component
function Parent() {
  const message = "Hello from Parent!";
  return <Child message={message} />;
}

// Child Component
function Child({ message }) {
  return <p>{message}</p>;
}
```

Key Points:

- Data flows **downward** from parent to child.
- Props are **read-only** in the child component. The child cannot modify the props directly.

2. From Children to Parent

To pass data from a **child component** to a **parent component**, you can use **callback functions**. The parent passes a function as a prop to the child, and the child calls this function to send data back to the parent.

How It Works:

- The parent component defines a function and passes it to the child as a prop.
- The child component calls this function and passes data as an argument.

```
// Parent Component
function Parent() {
  const handleDataFromChild = (data) => {
    console.log("Data from child:", data);
  };

  return <Child sendDataToParent={handleDataFromChild} />;
}

// Child Component
function Child({ sendDataToParent }) {
  const data = "Hello from Child!";
  return <button onClick={() => sendDataToParent(data)}>Send Data to Parent</button>;
}
```

Key Points:

- Data flows **upward** from child to parent.
- The parent controls the logic for handling the data.

3. From Children to Sibling

To pass data between **sibling components**, you need to **lift the state up** to their closest common parent. The parent component manages the shared state and passes it down to both siblings as props.

How It Works:

- The parent component holds the state and passes it to both siblings as props.
- One sibling updates the state (via a callback function), and the other sibling receives the updated state.

```
// Parent Component
function Parent() {
  const [sharedData, setSharedData] = useState("");
  const handleDataFromSibling = (data) ⇒ {
    setSharedData(data);
  };

  return (
    <div>
      <SiblingA sendDataToParent={handleDataFromSibling} />
      <SiblingB sharedData={sharedData} />
    </div>
  );
}

// Sibling A (Child Component)
function SiblingA({ sendDataToParent }) {
  const data = "Hello from Sibling A!";
  return <button onClick={() ⇒ sendDataToParent(data)}>Send Data to Sib
ling B</button>;
}

// Sibling B (Child Component)
function SiblingB({ sharedData }) {
  return <p>Data from Sibling A: {sharedData}</p>;
}
```

Key Points:

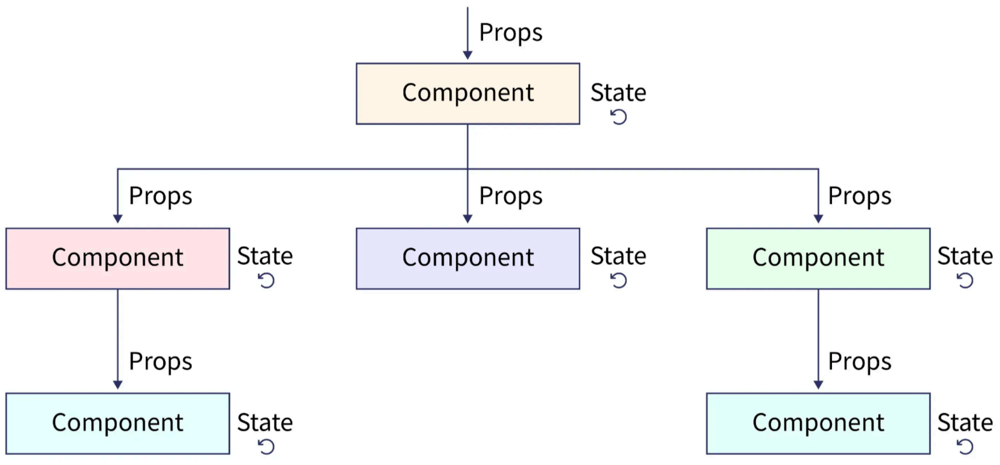
- Data flows through the **common parent**.
- The parent acts as a **mediator** between the siblings.

4. From Children to Cousin

To pass data between **cousin components** (components that are not directly related), you can use one of the following approaches:

1. **Lift the state up** to a common ancestor.
2. Use **React Context** for global state management.

3. Use a **state management library** like Redux or Zustand.



Summary

Relationship	How Data is Passed
Parent to Children	Pass data as **props** from parent to child.
Children to Parent	Pass a **callback function** as a prop from parent to child. Child calls it with data.
Children to Sibling	**Lift state up** to the common parent and pass it down as props.
Children to Cousin	**Lift state up** to a common ancestor or use **React Context** for global state.

Assignments

- **1. Parent-Child Counter**
  Create a parent component with a counter and two child components. One child should display the counter value, and the other should have buttons to increment/decrement it. Use the concepts of parent-to-child and child-to-parent data flow.
- **2. Sibling Communication**
  Build a form in one sibling component that collects user information (name, email) and displays it in real-time in another sibling component. Implement this using state lifting as discussed.

- **3. Shopping Cart Components**
  Create a shopping cart system with three components: ProductList, Cart, and CartTotal. Implement data flow so that adding products in ProductList updates both Cart and CartTotal components using the parent-as-mediator pattern.
- **4. Multi-Level Form**
  Design a multi-step form where data entered in child components needs to be accumulated in a parent component. Use callback functions to pass data upward from children to parent.
- **5. Theme Switcher**
  Implement a theme switcher that affects multiple unrelated components (cousins). Practice by lifting state to a common ancestor.