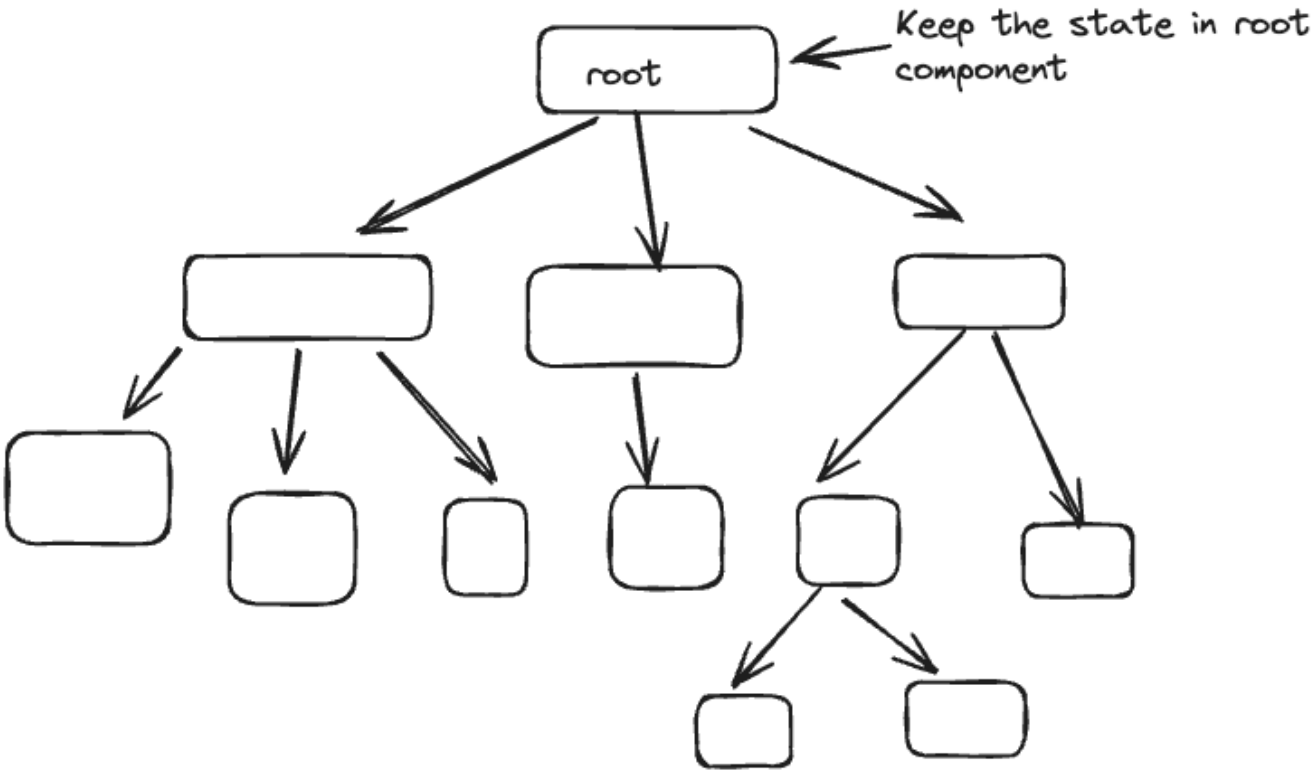
Lifting the state up

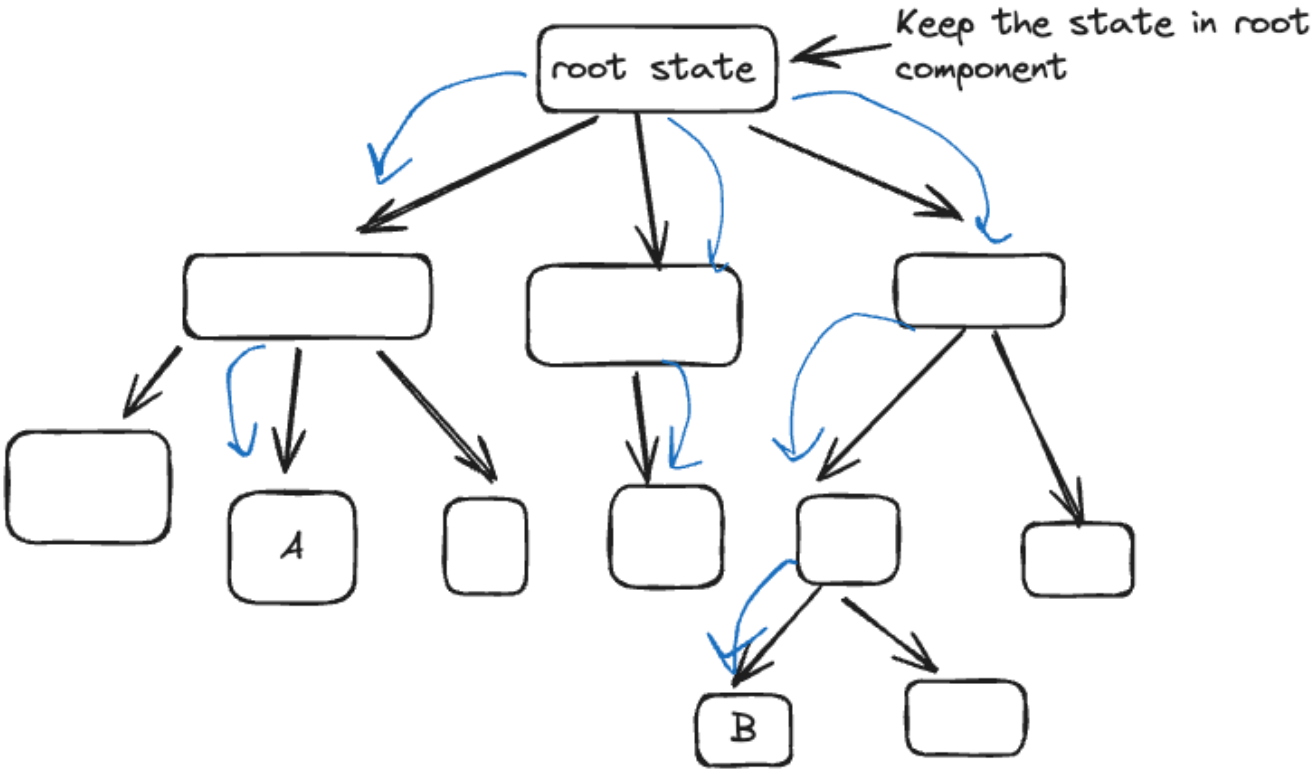
If we have a very complex component hierarchy, then if we want to update a state in a component and see it's effects in another component, then it will be a pretty challenging task. We might have to use a lot of callbacks and then only we will be pass things around.
But having too many callbacks passing the states here there, will make the components very tightly coupled.

That means we need to have a better mechanism.

That's where lifting the state up in the component hierarchy comes into the picture. Meaning of lifting the state up is that, we try to bring up the state at the very top level of the component hierarchy.

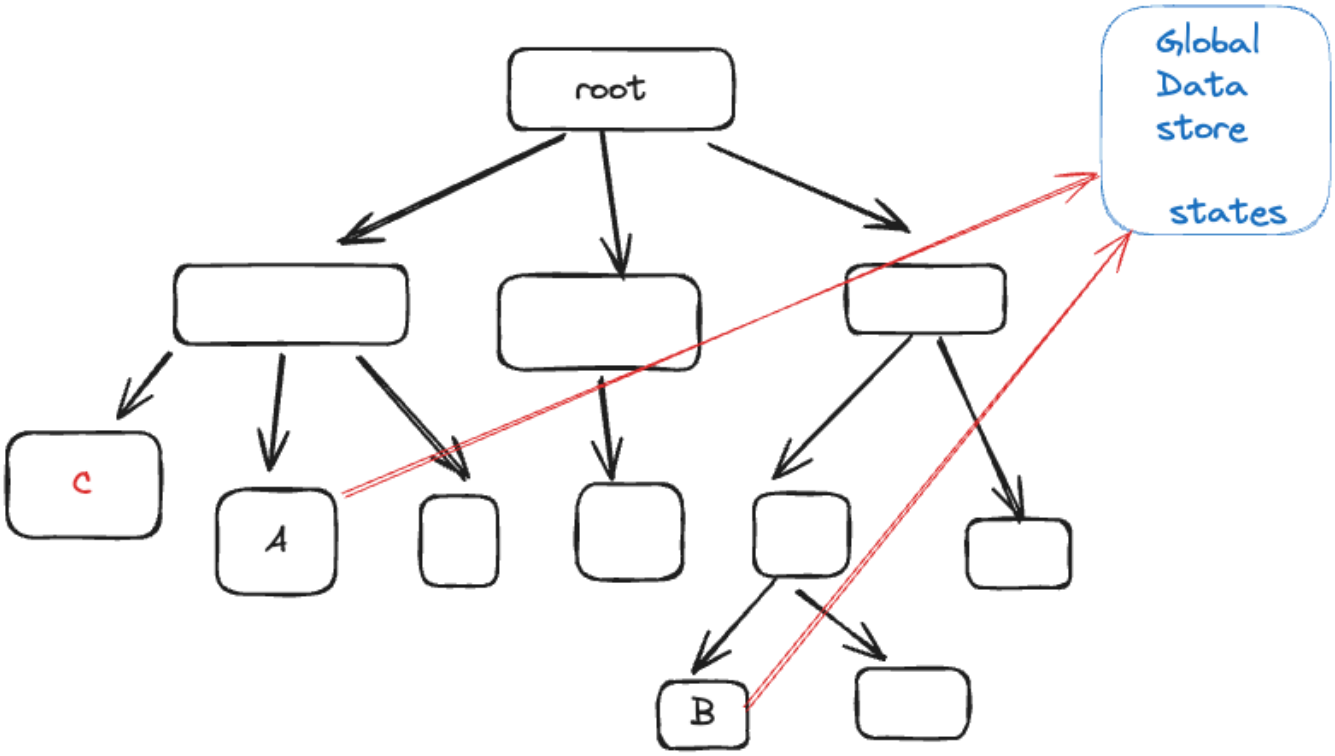


With this approach we can actually have the states mentioned in the root component and then root component will pass down these states and their updater function in the component hierarchy.
But there is one more problem with this approach.



From the very top of the hierarchy we need to pass our states as a prop very low level in the hierarchy, now if the application is super complex, then this props passing can be extremely hard to maintain. And this problem is also referred as Prop drilling.

Maintain the state as a global store



If we want a couple of components to share a state, then we can maintain a global data store, which will be having some states, which can be access by any component and can be updated by any other component. For example, may be A component updates the state and B component consumes the state, so when A triggers an update, B will re-render itself.

How to achieve this:

- React context API
- Redux
- Zustand
- MobX
- and more....

React context API

React from some of the latest versions, started supporting an in house solution for better state management. This is called as context API. Context api helps us to describe a state storage and then help us to access it from the component hierarchy without prop drilling.

1. Create a context object

```
// CurrencyContext.js
import { createContext } from "react";
```

```
export const CurrencyContext = createContext(); // Create a context object
```

2. Once we have created the context object, then we can use it to store some states and acsess them.
3. To make the context accessible across multiple components, we can wrap those components inside `Context.Provider` component. It is a component, that will be automatically given to use by the context object.

```
function App() {
        const [currency, setCurrency] = useState('usd');
        return (
                <>
                        <CurrencyContext.Provider value={ { currency,
setCurrency } }>
                                <Home />
                        </CurrencyContext.Provider>
                </>
        )
}
```

5. Here CurrencyContext.Provider is a component that is automatically created by the context object. We will wrap out `Home` component inside it, due to which in the complete hierarchy of the home component we can access the CurrencyContext.
6. Now In the provider, we can add a value prop, which takes an object and inside this object we can store any state or functions, that we want to make accessible inside our `Home` component. The line `value={ { currency, setCurrency } }` passed currency and setCurrency in an object which is assigned to the value prop.
7. To access this context object anywhere in the `Home` component we can use the `useContext` Hook. This hook takes in the context object as a paramerter and return the value object to us, that we can de-structure, however we want.

```
function Navbar() {
        const { setCurrency } = useContext(CurrencyContext);
        return (
                <>
                        ....
                </>
        );
}
```

```
function CoinTable() {

        const {currency} = useContext(CurrencyContext);
        return (
                <>
                        {currency}
                        ....
                </>
        )

}
```