ANECO ACADEMY

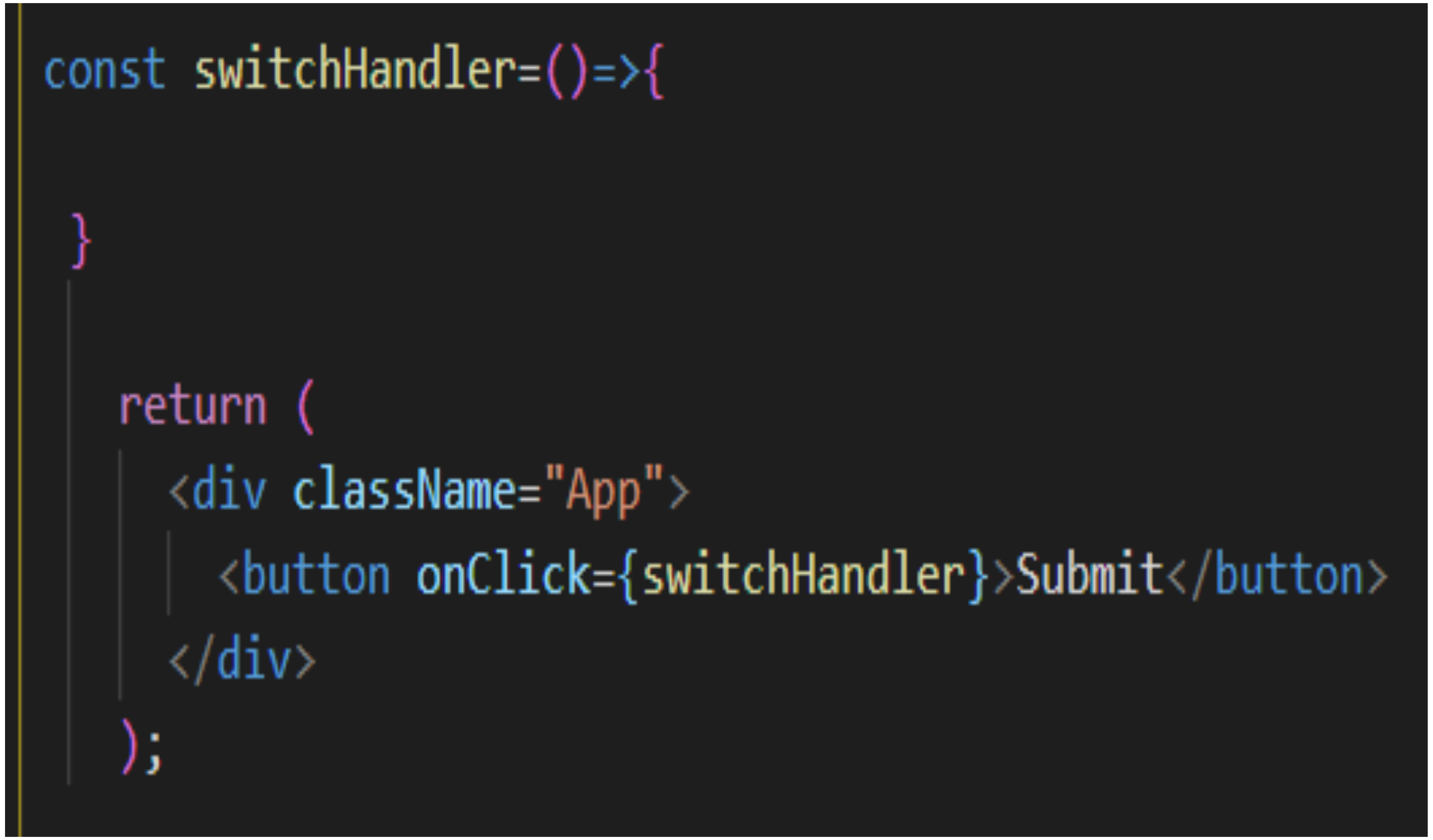
REACTJS - FUNCTIONAL COMPONENT - STATE

Events in functional component

In normal JavaScript you will have events like onclick,onchange,onblur,onfocus,onsubmit.
In React you will have the the same events but onClick,onChange,onBlur,onFocus. The second letter in capital.
Syntax:

```
const switchHandler=()=>{
alert("hello")
}
<button onClick={switchHandlers}></button>
```

Example events in functional component

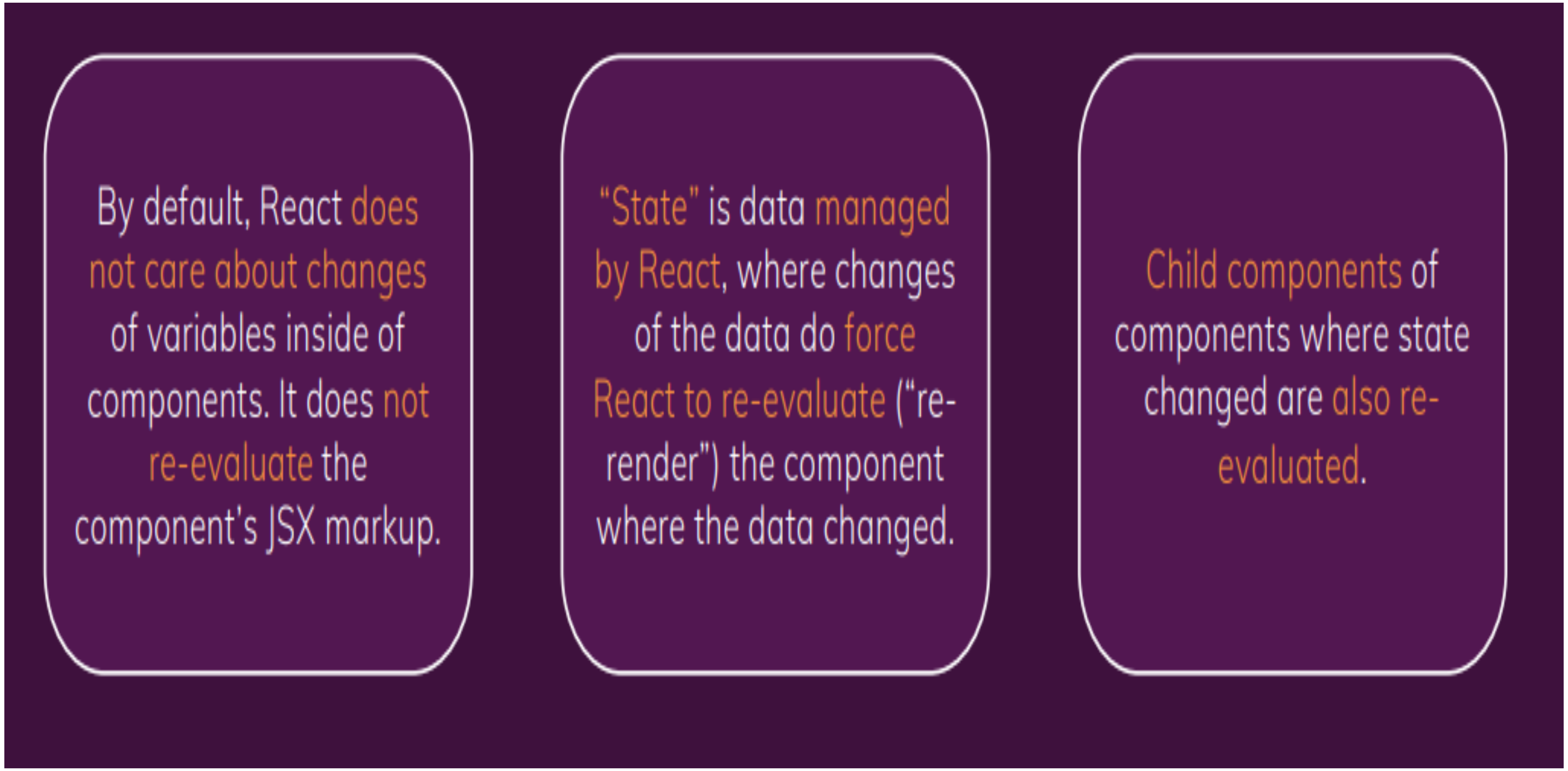


```
const switchHandler=()=>{


 }


  return (
    <div className="App">
      <button onClick={switchHandler}>Submit</button>
    </div>
  );
```

State in functional component

To use state in functional component we need to useState() property from React package.
In functional component you can have multiple state in one component
useState() property will return a array with two indexes.
First index will represent the get updated state details
Second index will represent updating the state details with existing one.

What is state

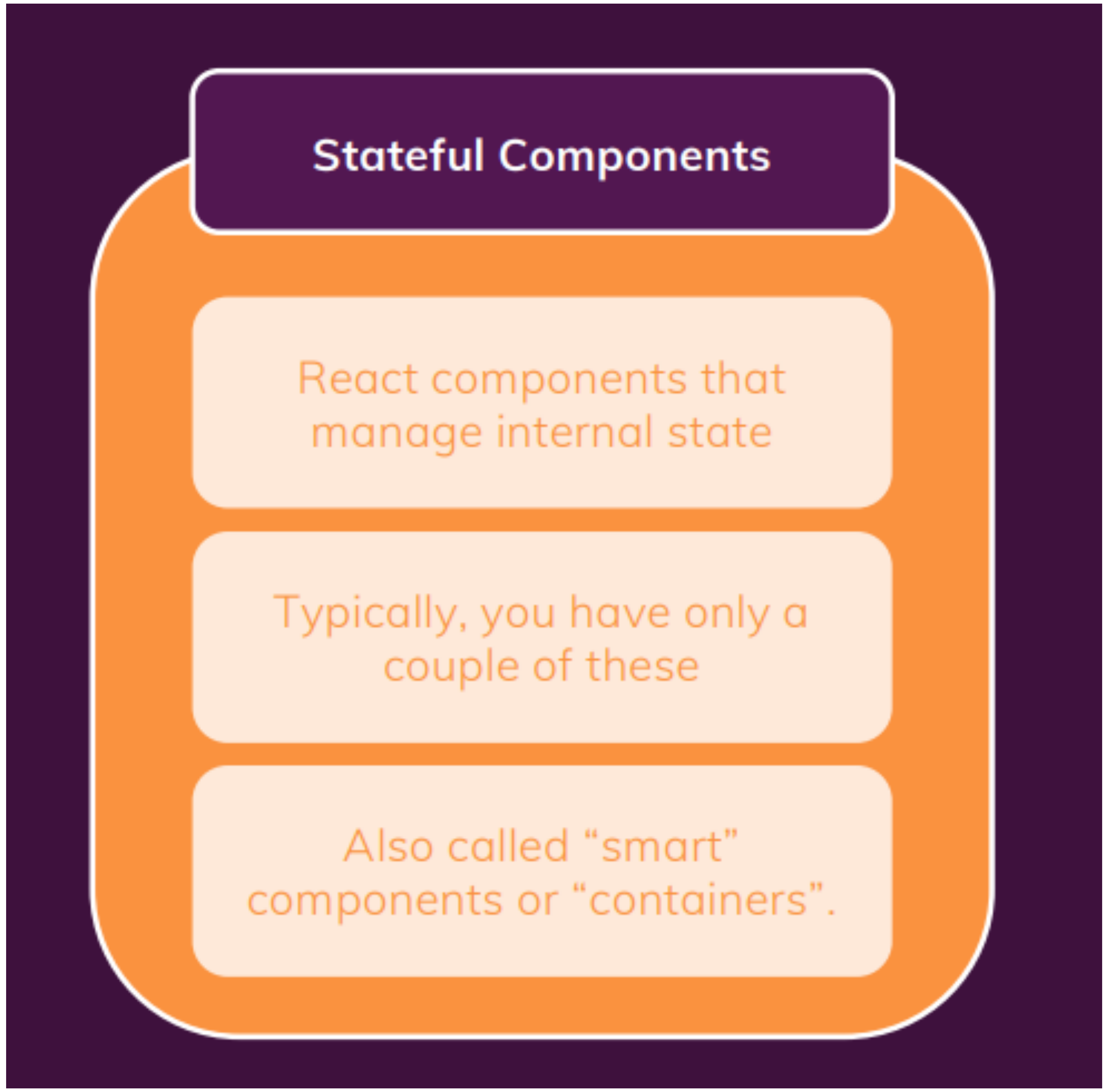


By default, React does not care about changes of variables inside of components. It does not re-evaluate the component's JSX markup.

"State" is data managed by React, where changes of the data do force React to re-evaluate ("re-render") the component where the data changed.

Child components of components where state changed are also re-evaluated.

Statefull components

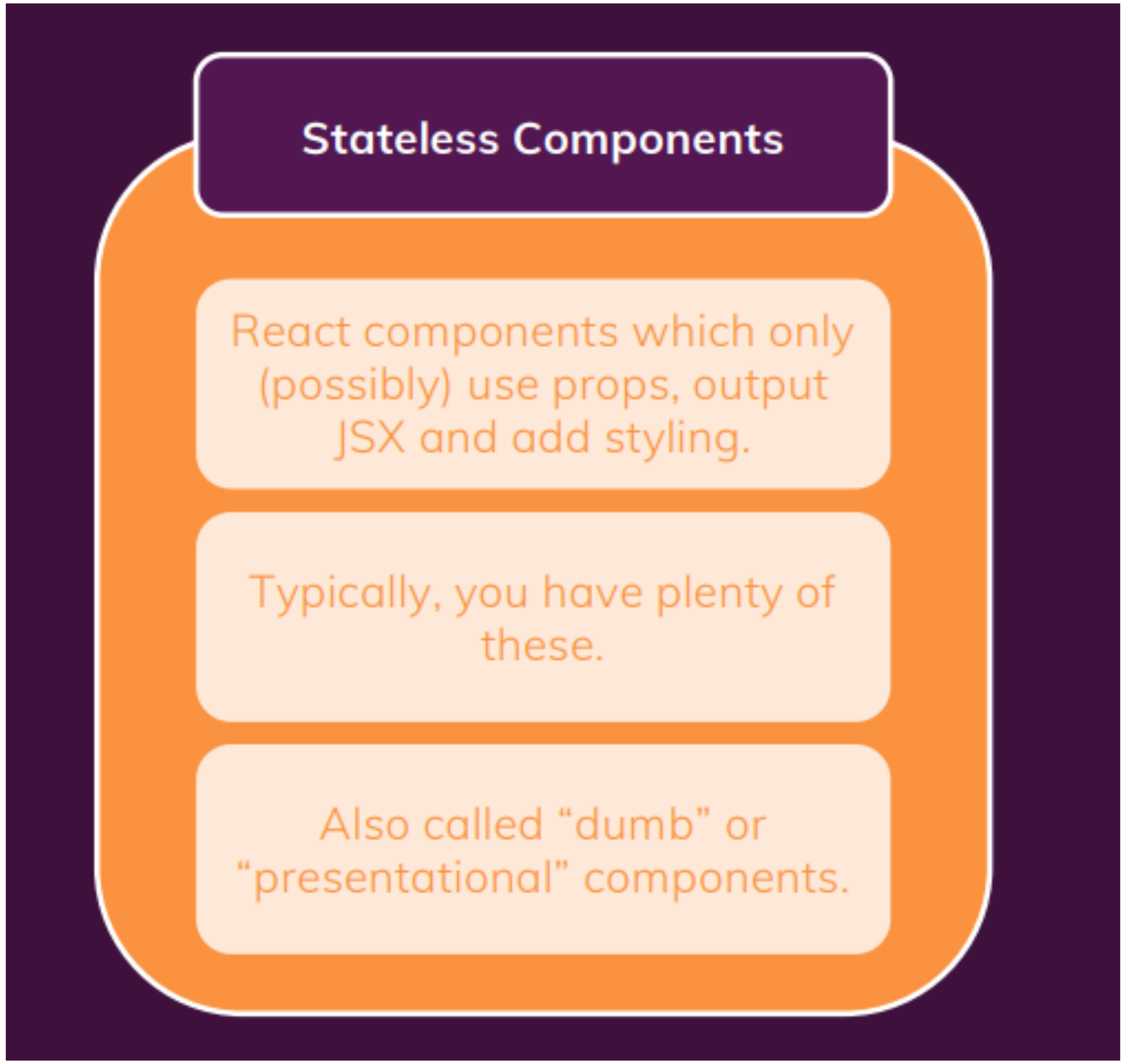

Stateful Components

React components that manage internal state

Typically, you have only a couple of these

Also called “smart” components or “containers”.

Stateless components



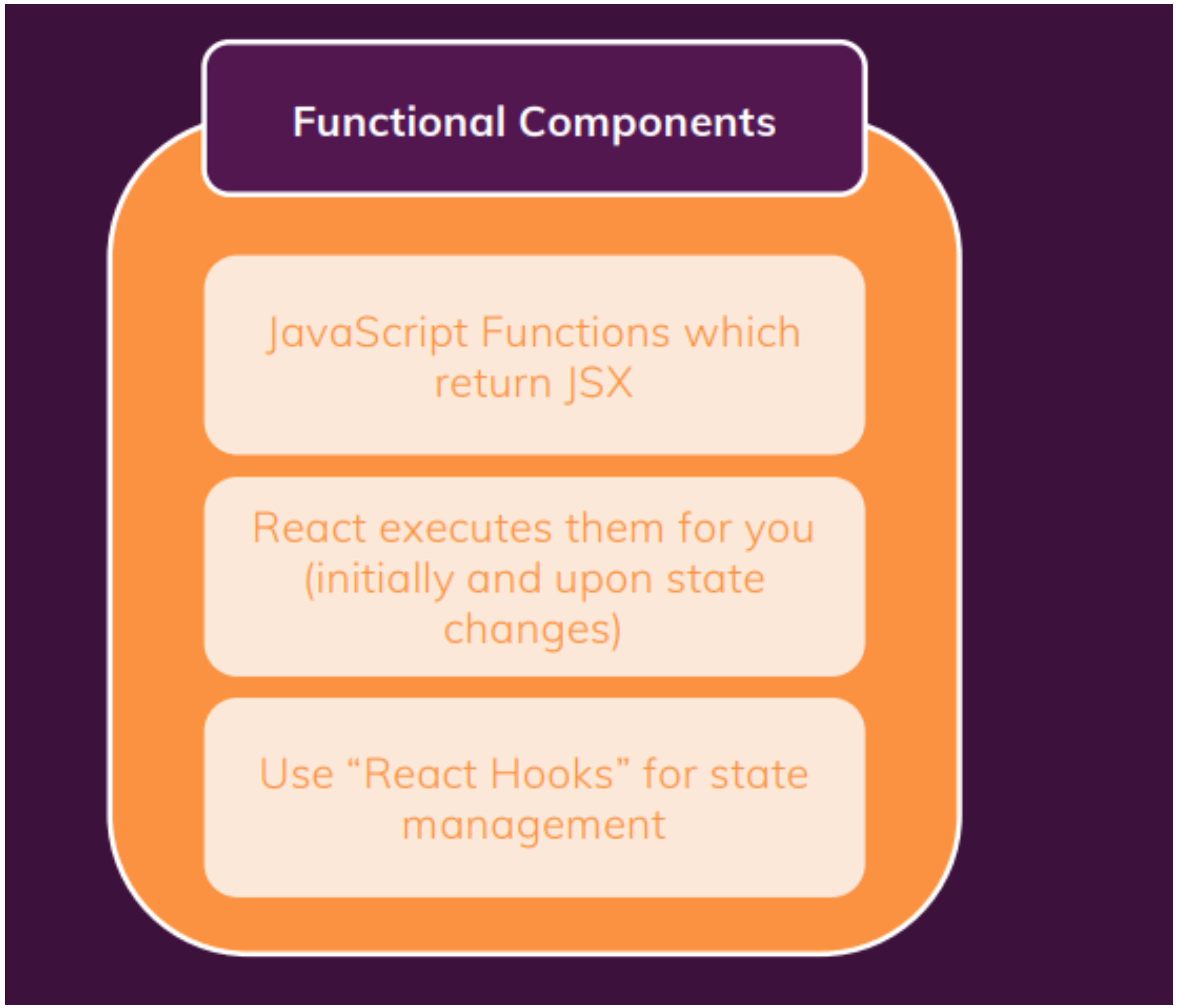
Stateless Components

React components which only (possibly) use props, output JSX and add styling.

Typically, you have plenty of these.

Also called “dumb” or “presentational” components.

Type 1:Building components

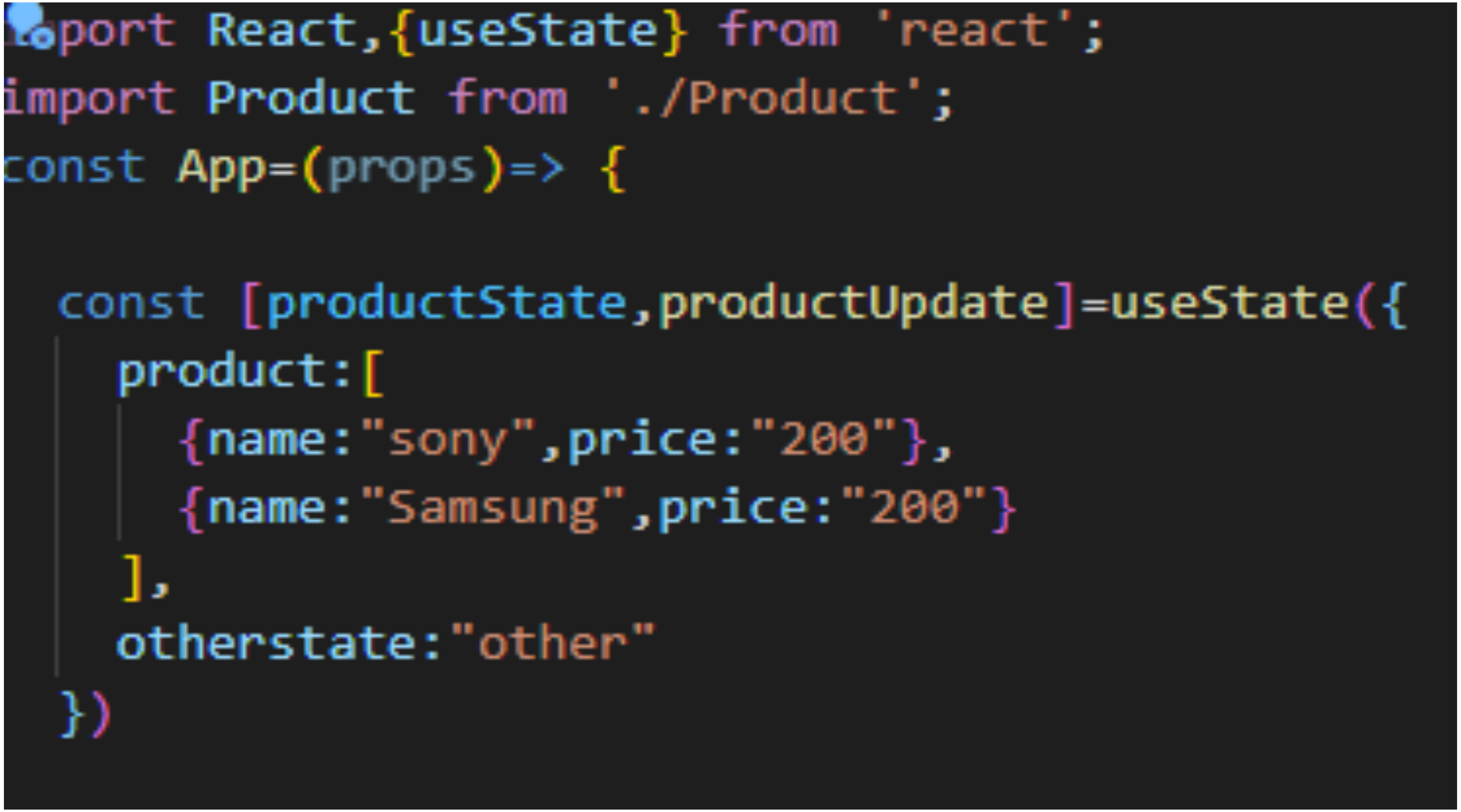


Functional Components

JavaScript Functions which return JSX

React executes them for you (initially and upon state changes)

Use “React Hooks” for state management



```
port React,{useState} from 'react';
import Product from './Product';
const App=(props)=> {

  const [productState,productUpdate]=useState({
    product:[
      {name:"sony",price:"200"},
      {name:"Samsung",price:"200"}
    ],
    otherstate:"other"
  })
```

Single State in functional component-2

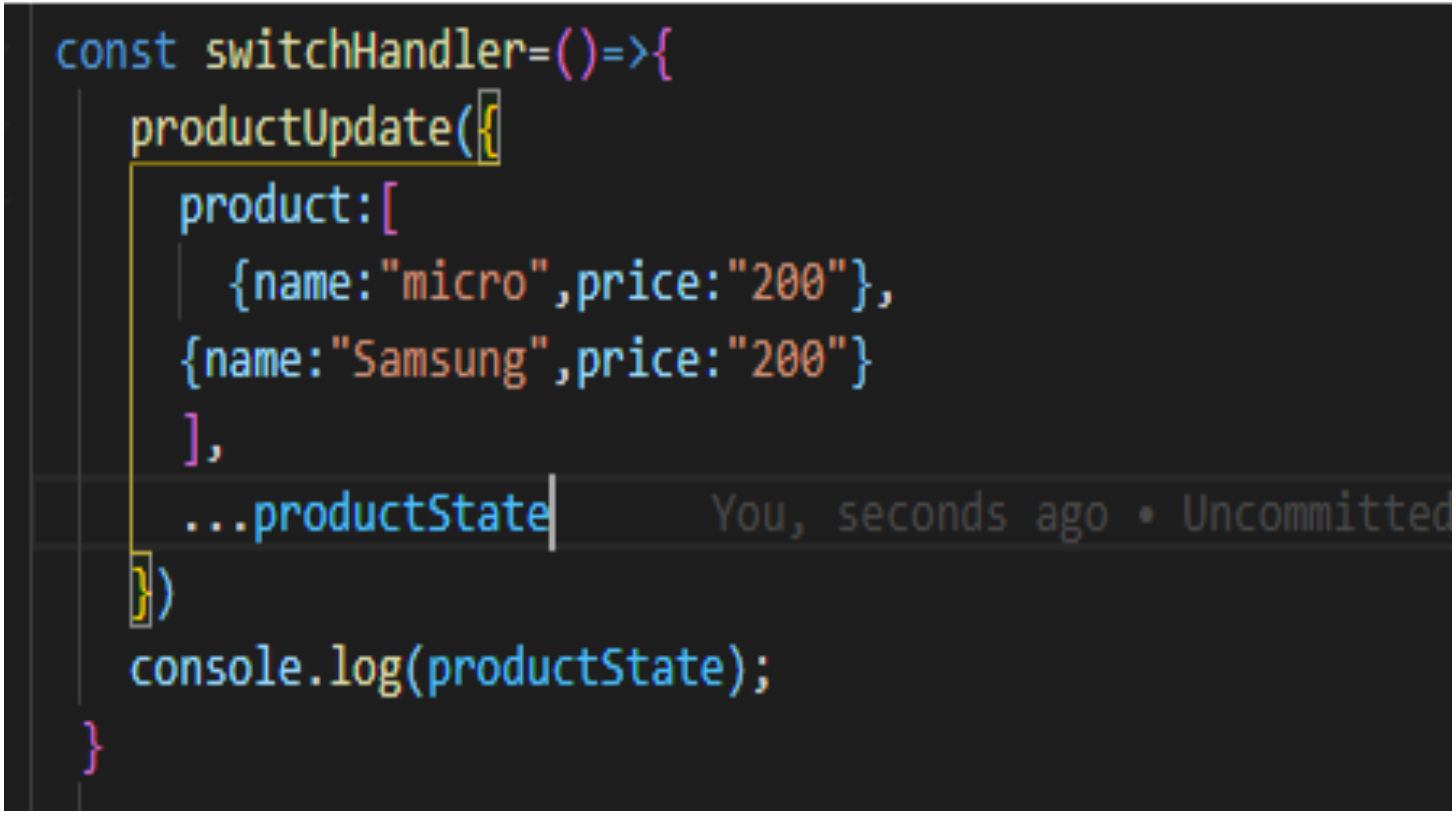
In Single state we have two properties

Product

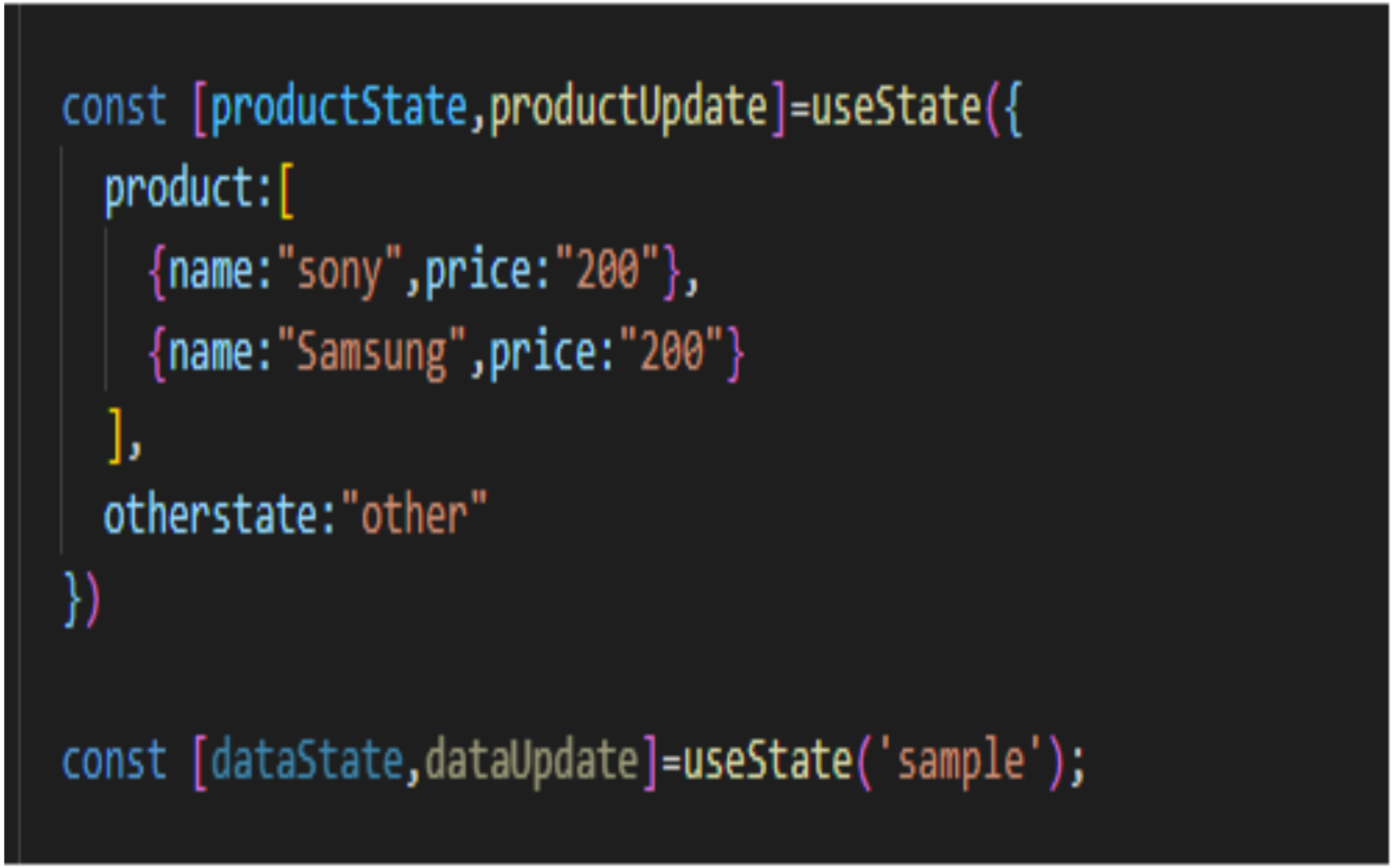
Otherstate

If there is change happen in product property we need to copy the existing state of Otherstate property as well.

STATE



```
const switchHandler=()=>{
    productUpdate({
      product:[
        {name:"micro",price:"200"},
      {name:"Samsung",price:"200"}
      ],
      ...productState          You, seconds ago • Uncommitted
    })
    console.log(productState);
 }
```

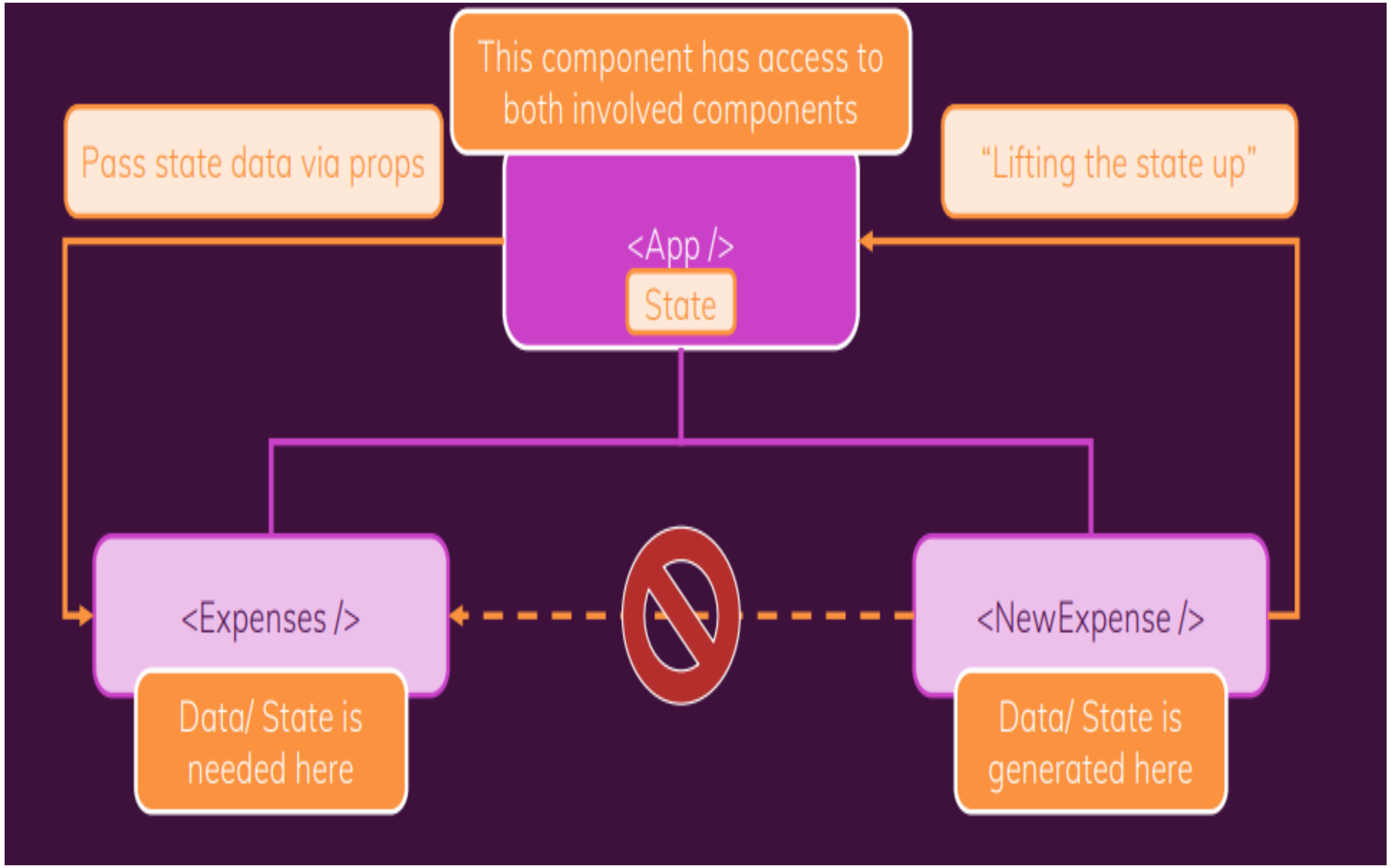


```
const [productState,productUpdate]=useState({
  product:[
    {name:"sony",price:"200"},
    {name:"Samsung",price:"200"}
  ],
  otherstate:"other"
})

const [dataState,dataUpdate]=useState('sample');
```

Multiple state in functional component-1

- If you want to maintain multiple state in one component is possible.
- We can use for complicated scenario where maintaining the multiple state is needed
- If you change one state then other state will not get impacted both will have separate instance

Lifting state up


This component has access to both involved components
Pass state data via props
"Lifting the state up"
<App />
State
<Expenses />
<NewExpense />
Data/ State is needed here
Data/ State is generated here

LIFTING STATE UP

Lifting state up is an important pattern for React developers because sometimes we have state that's located within a particular component that also needs to be shared with sibling components.

Instead of using an entire state management library like Redux or React Context, we can just lift the state up to the closest common ancestor and pass both the state variables the state values down as well as any callbacks to update that state.