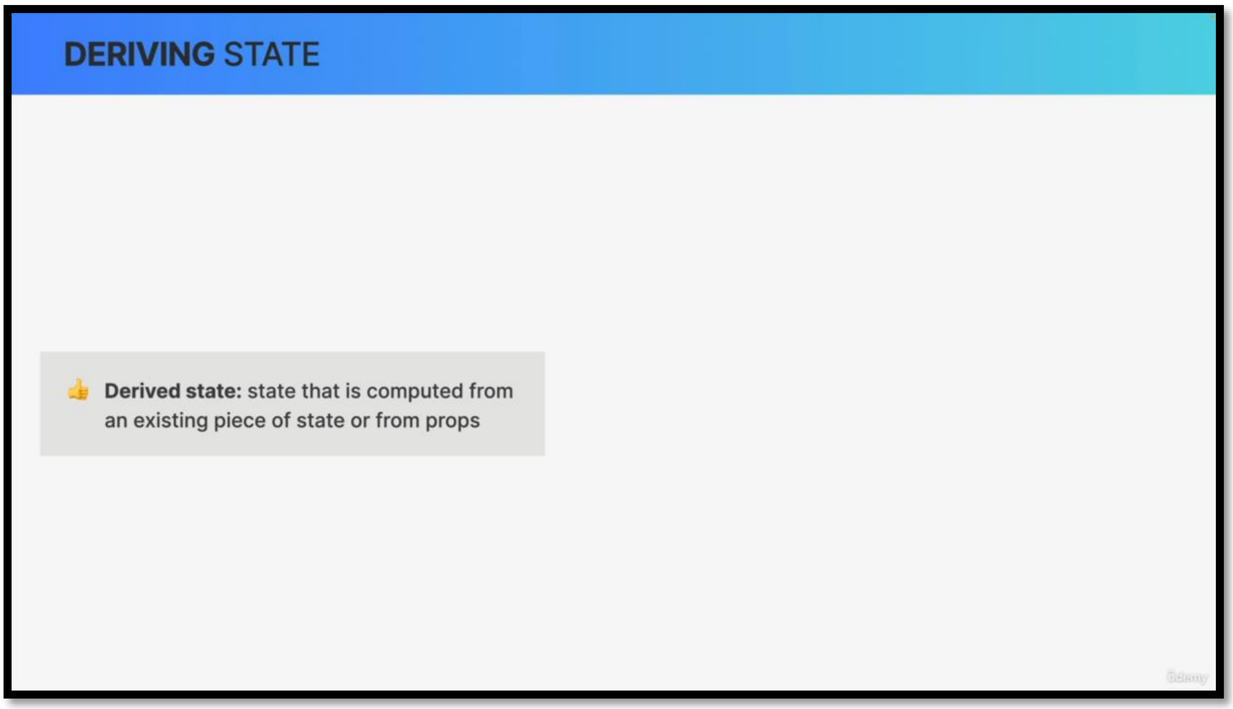
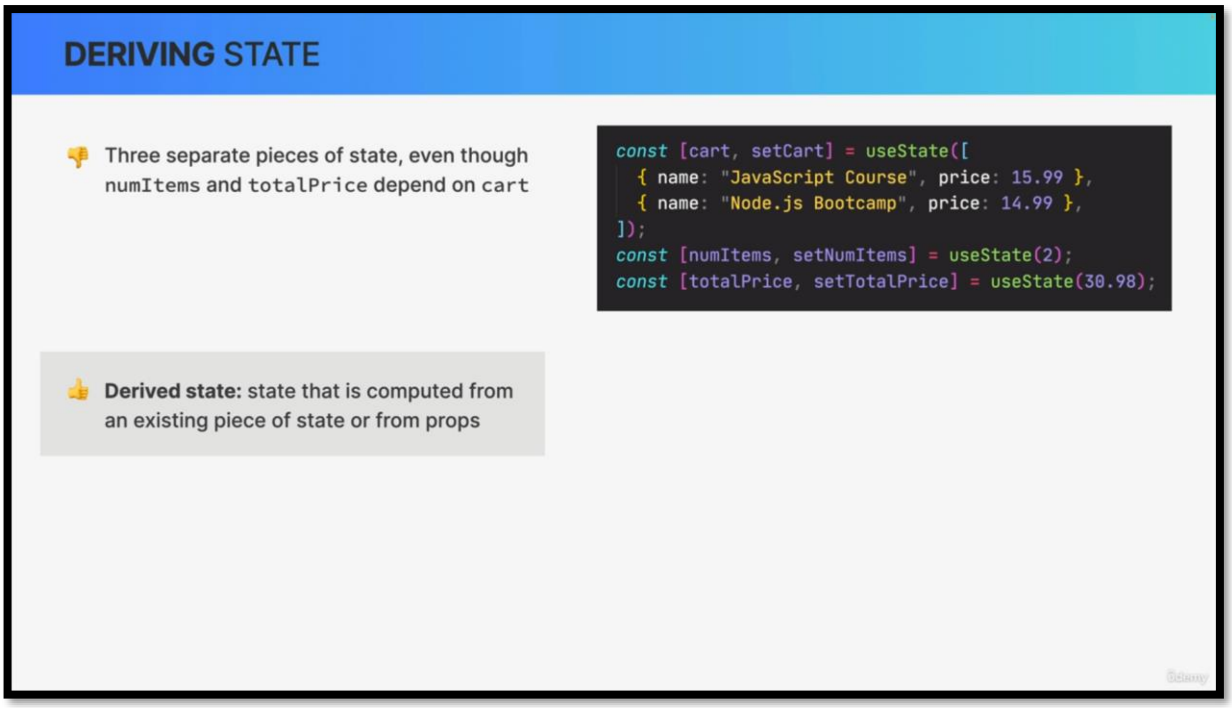

## DERIVING STATE

👍 **Derived state:** state that is computed from an existing piece of state or from props

Derived state is simply state that is computed from another existing piece of state or also from props.

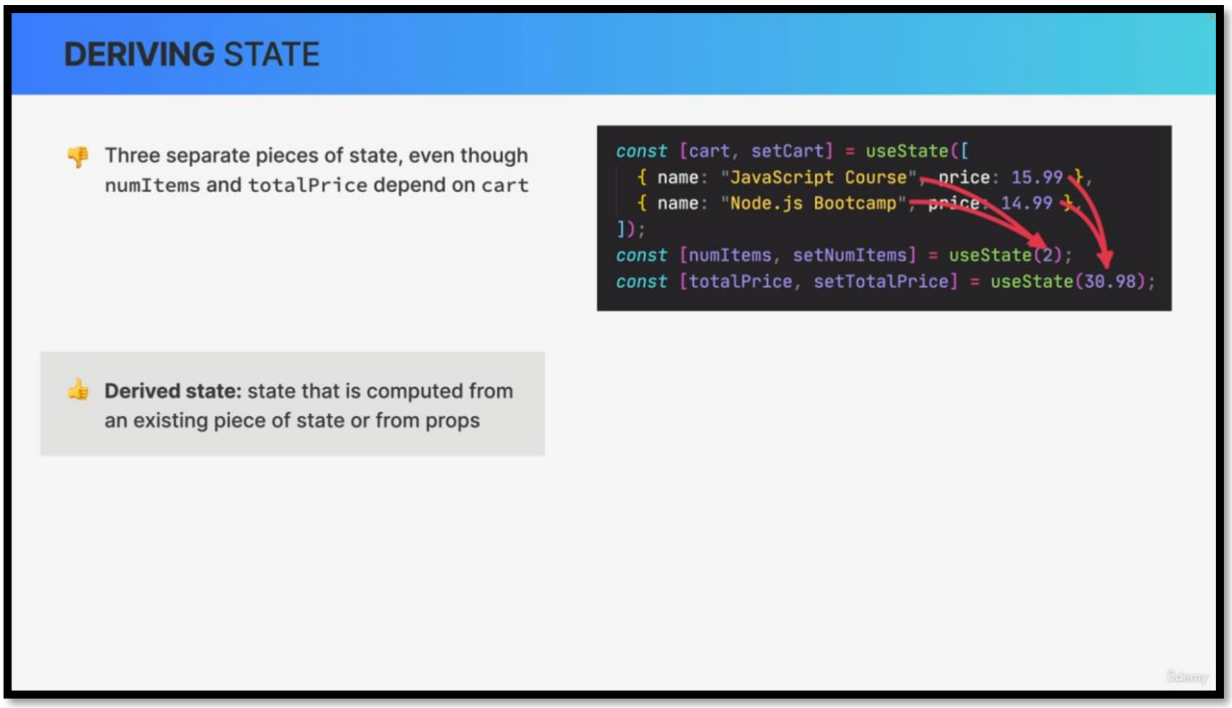

## DERIVING STATE

👎 Three separate pieces of state, even though `numItems` and `totalPrice` depend on `cart`

```
const [cart, setCart] = useState([
  { name: "JavaScript Course", price: 15.99 },
  { name: "Node.js Bootcamp", price: 14.99 },
]);
const [numItems, setNumItems] = useState(2);
const [totalPrice, setTotalPrice] = useState(30.98);
```

👍 **Derived state:** state that is computed from an existing piece of state or from props

Here, we have three pieces of state as we can see by the three use state function calls.



## DERIVING STATE

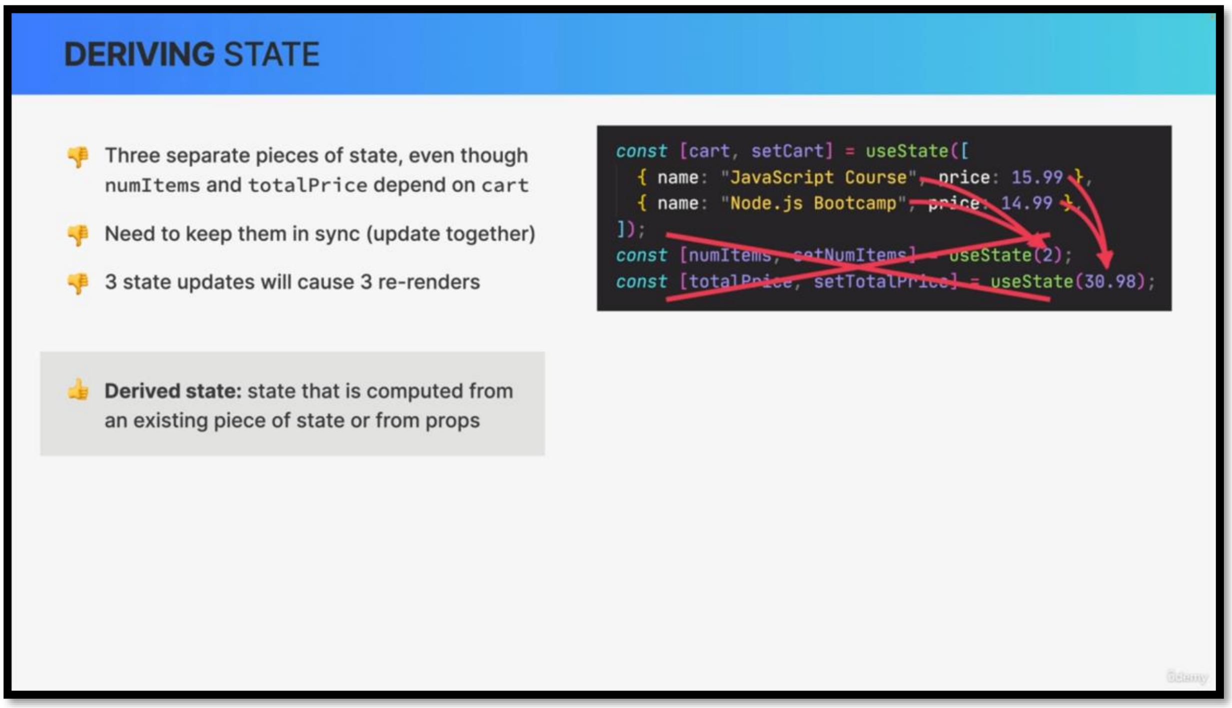
👎 Three separate pieces of state, even though `numItems` and `totalPrice` depend on `cart`

```
const [cart, setCart] = useState([
  { name: "JavaScript Course", price: 15.99 },
  { name: "Node.js Bootcamp", price: 14.99 },
]);
const [numItems, setNumItems] = useState(2);
const [totalPrice, setTotalPrice] = useState(30.98);
```

👍 **Derived state:** state that is computed from an existing piece of state or from props

However, if we analyze these states, it actually doesn't make much sense that all of them exist because numItems and totalPrice depend entirely on the cart. So, numItems is simply the number of items in the cart and totalPrice is the sum of all the prices in the cart.

All the data for these two pieces of state is actually already in the cart. So, there's no need to create these additional state variables.



## DERIVING STATE

👎 Three separate pieces of state, even though `numItems` and `totalPrice` depend on `cart`

👎 Need to keep them in sync (update together)

👎 3 state updates will cause 3 re-renders

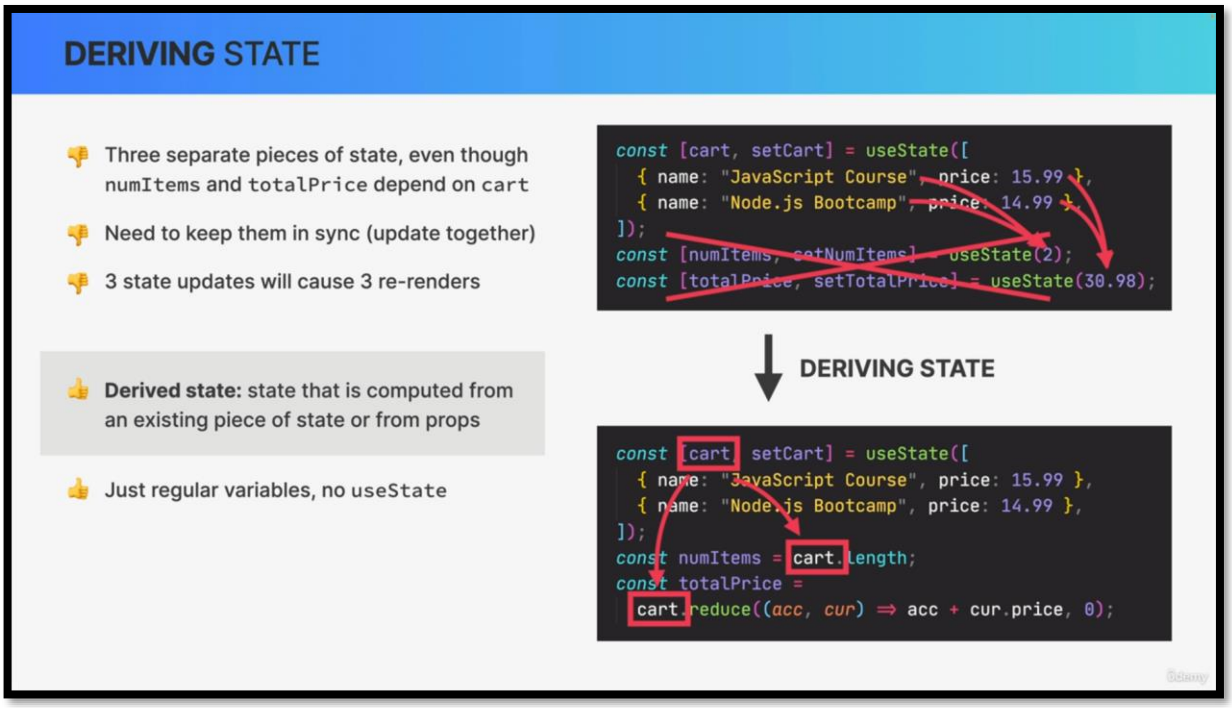
```
const [cart, setCart] = useState([
  { name: "JavaScript Course", price: 15.99 },
  { name: "Node.js Bootcamp", price: 14.99 },
]);
const [numItems, setNumItems] = useState(2);
const [totalPrice, setTotalPrice] = useState(30.98);
```

👍 **Derived state:** state that is computed from an existing piece of state or from props

Doing so is actually quite problematic.

1. First, because now we have to keep all these states in sync. So, we need to be careful to always update them together. So, in this situation, whenever we update the cart, we will also need to manually update the number of items and the total price. Otherwise, our states would get out of sync.

2. Updating these three states separately creates a second problem because that will then re-render the component three times which is absolutely unnecessary in this example.



## DERIVING STATE

- Three separate pieces of state, even though `numItems` and `totalPrice` depend on `cart`
- Need to keep them in sync (update together)
- 3 state updates will cause 3 re-renders

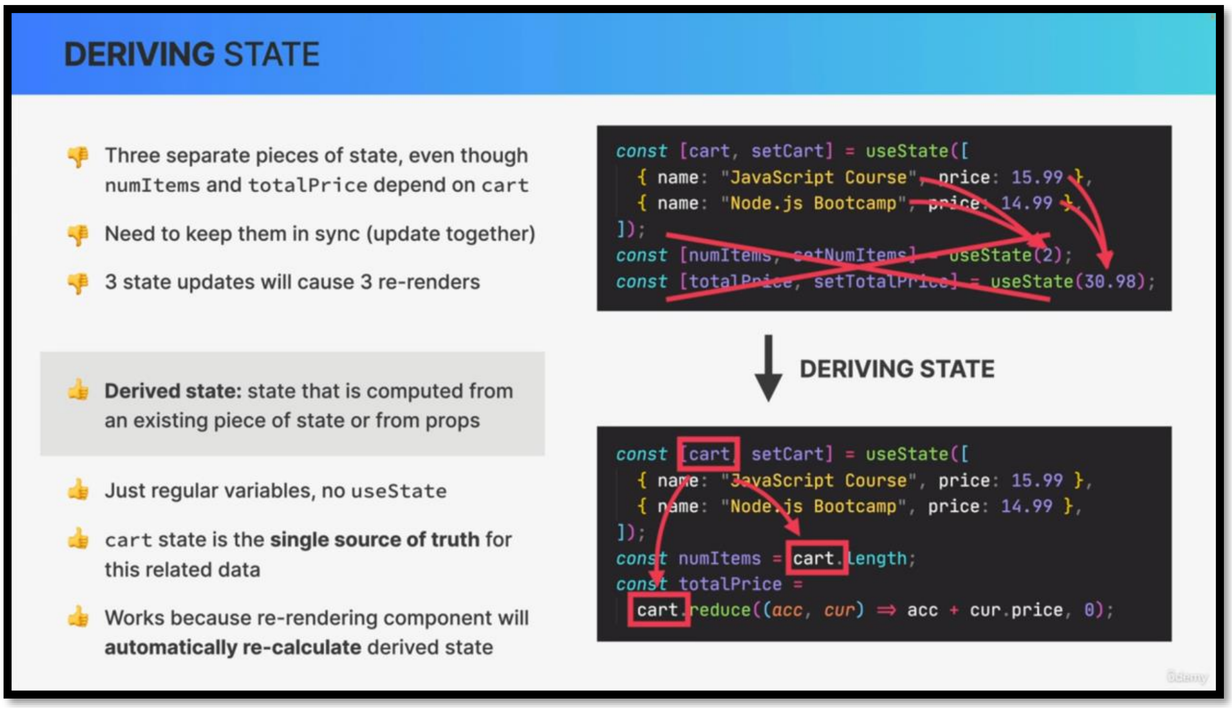
```
const [cart, setCart] = useState([
  { name: "JavaScript Course", price: 15.99 },
  { name: "Node.js Bootcamp", price: 14.99 },
]);
~~const [numItems, setNumItems] = useState(2);~~
~~const [totalPrice, setTotalPrice] = useState(30.98);~~
```

DERIVING STATE

- **Derived state:** state that is computed from an existing piece of state or from props
- Just regular variables, no `useState`

```
const [cart, setCart] = useState([
  { name: "JavaScript Course", price: 15.99 },
  { name: "Node.js Bootcamp", price: 14.99 },
]);
const numItems = cart.length;
const totalPrice =
  cart.reduce((acc, cur) => acc + cur.price, 0);
```

Instead, we can simply derive the numItems and totalPrice state from the cart and therefore solve all these problems because the cart already contains all the data that we need.



## DERIVING STATE

- 👎 Three separate pieces of state, even though `numItems` and `totalPrice` depend on `cart`
- 👎 Need to keep them in sync (update together)
- 👎 3 state updates will cause 3 re-renders

👍 **Derived state:** state that is computed from an existing piece of state or from props

- 👍 Just regular variables, no `useState`
- 👍 `cart` state is the **single source of truth** for this related data
- 👍 Works because re-rendering component will **automatically re-calculate** derived state

```
const [cart, setCart] = useState([
  { name: "JavaScript Course", price: 15.99 },
  { name: "Node.js Bootcamp", price: 14.99 },
]);
~~const [numItems, setNumItems] = useState(2);~~
~~const [totalPrice, setTotalPrice] = useState(30.98);~~
```

DERIVING STATE

```
const [cart, setCart] = useState([
  { name: "JavaScript Course", price: 15.99 },
  { name: "Node.js Bootcamp", price: 14.99 },
]);
const numItems = cart.length;
const totalPrice =
  cart.reduce((acc, cur) => acc + cur.price, 0);
```

So here, we simply calculate numItems as the cart length and totalPrice as the sum of all prices and store them in regular variables. There is no useState required here which will cause no unnecessary re-renders.

The cart state acts as a single source of truth for these related pieces of state making sure that everything will always stay in sync.

This works because updating the cart will re-render the component which means that the function is called again.

So then, as all the code is executed, again, numItems and totalPrice will also, automatically, get recalculated.