

we have a list with two question items which clearly have no key prop but let's see what happens when we add a new item to the top of the list.

The two list items that we already had are clearly still the same, but they will now appear at different positions in the React element tree. They're no longer the first and second children but now they are the second and the third children.

We basically have the same elements but at different positions in the tree.

According to the diffing rules these two DOM elements will be removed from the DOM and then immediately recreated at their new positions.

This is obviously bad for performance because removing and rebuilding the same dumb element is just wasted work. The thing is that React doesn't know that this is wasted work.

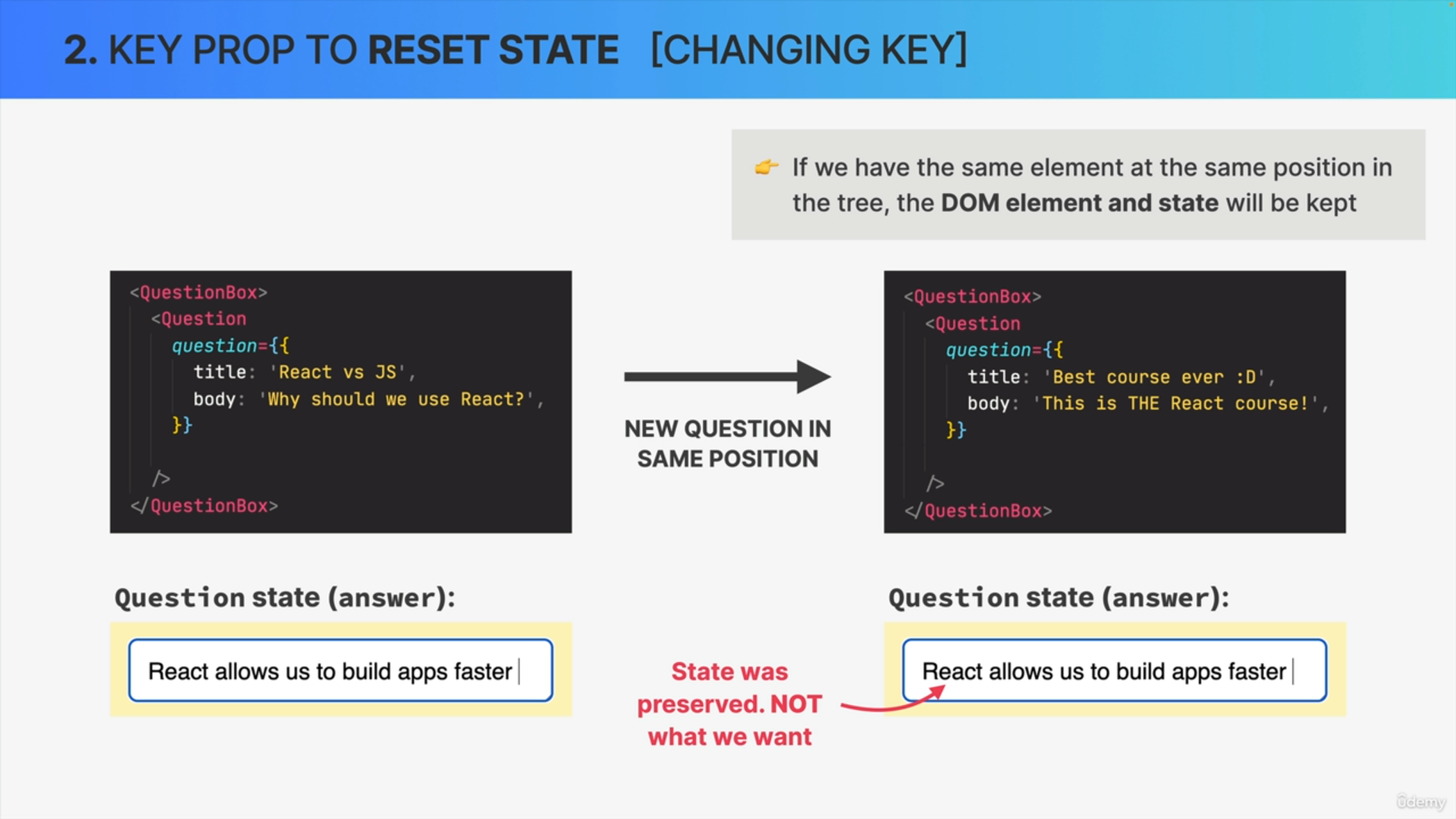
A key allows us developers to uniquely identify an element so we can give React that information that it doesn't have on its own.

When we add a new item to the top of the list, the two original elements are of course, still in different positions of the tree but they do have a stable key.

So, a key that stays the same across renders. So that's q1 and q2 in this case and according to the diffing rules, these two elements will now be kept in the DOM even though their position in the tree is different. So, they will not be destroyed.

Entry result will be a bit more of a performant UI. Now of course, you won't really notice this difference on small lists, but it will make a huge difference when you have a really big list with thousands of elements, which can actually happen in some applications.

So, in summary, always use the key prop when you have multiple child elements of the same type.

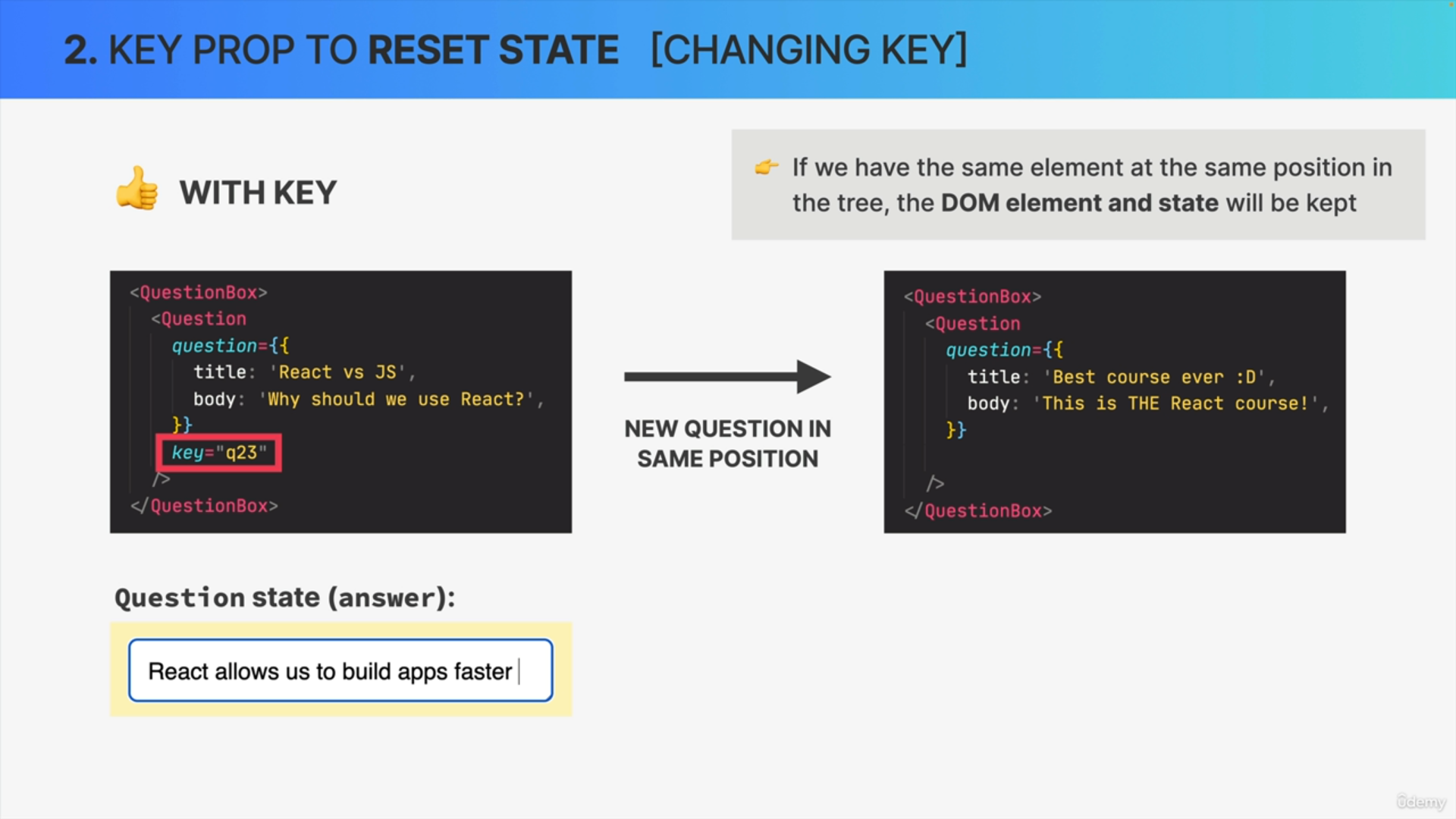


Let's say we have this question, inside question box and we pass in this object as a prop. Now the question component instance has an answer state, which right now is set to React allows us to build apps faster.

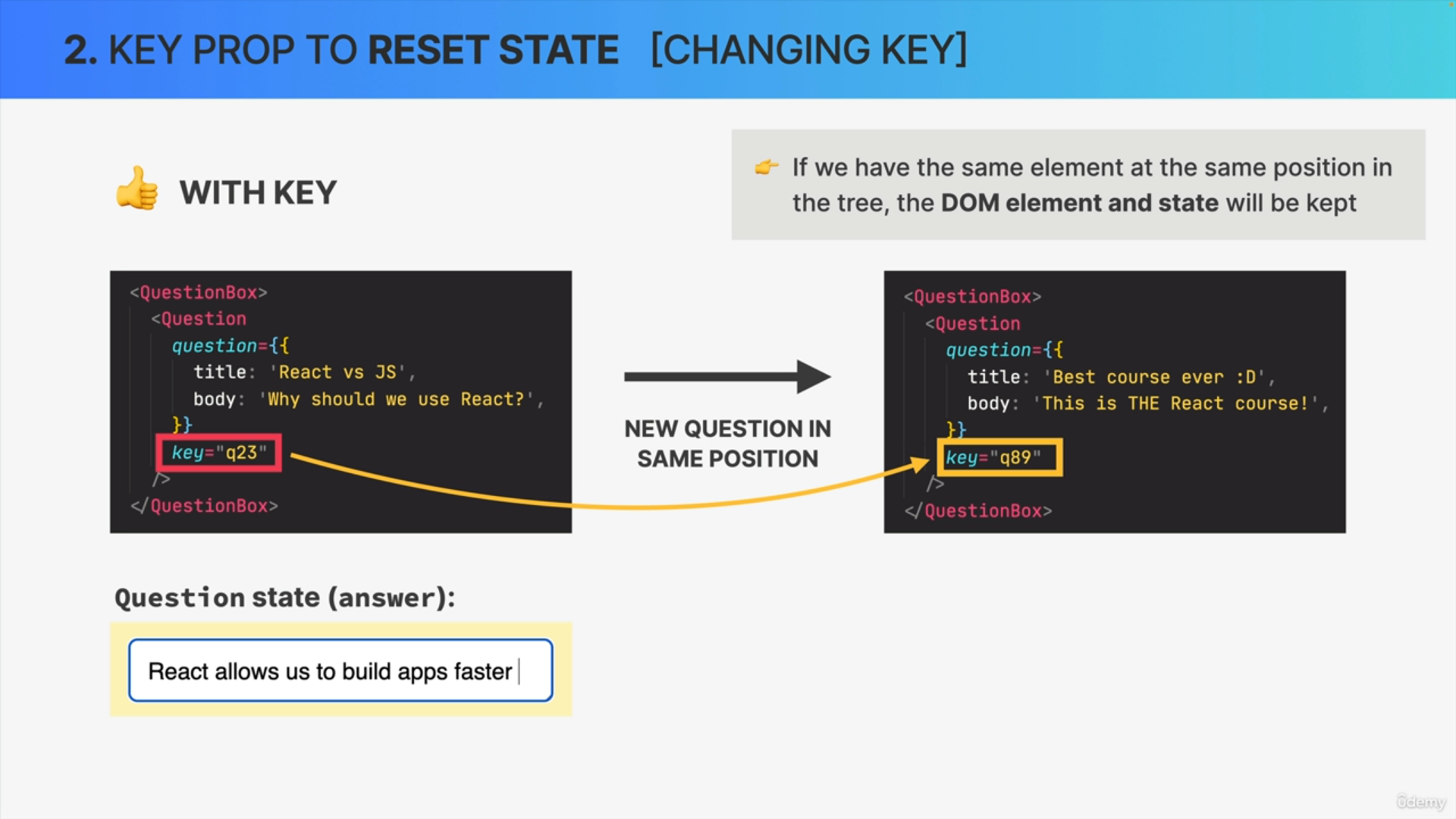
Let's imagine that the question changes to another one. We still have the same element at the same position in the tree. All that changed was the question prop.

According to diffing rules if we have the same element at the same position in the tree, the DOM element and its state will be kept. Therefore, what's gonna happen is that the state of question will be preserved.

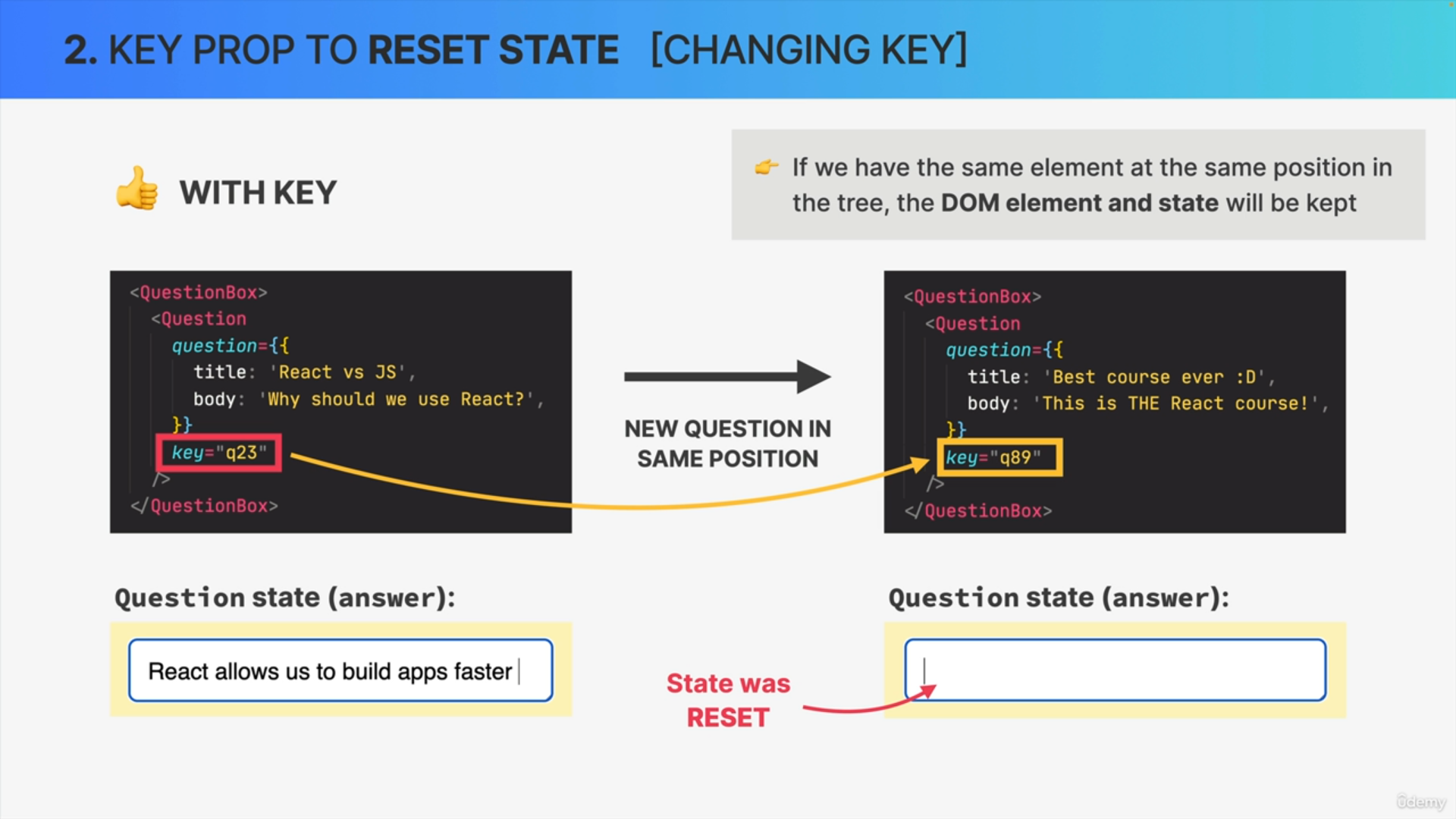
So, it will still show the answer that was in the component state before. But that answer is of course completely irrelevant to this new question it doesn't make any sense to keep this state around here. What we need is a way to reset this state.



So now, we have a key of q23 in this first question, which allows React to uniquely identify this component instance.



When a new question appears, we can give it a different key and so by doing this, we tell React that this should be a different component instance and therefore, it should create a brand-new DOM element.



The result of doing this is that the state will be reset which is exactly what we need in the situation in order to make this small app work in a logical way.