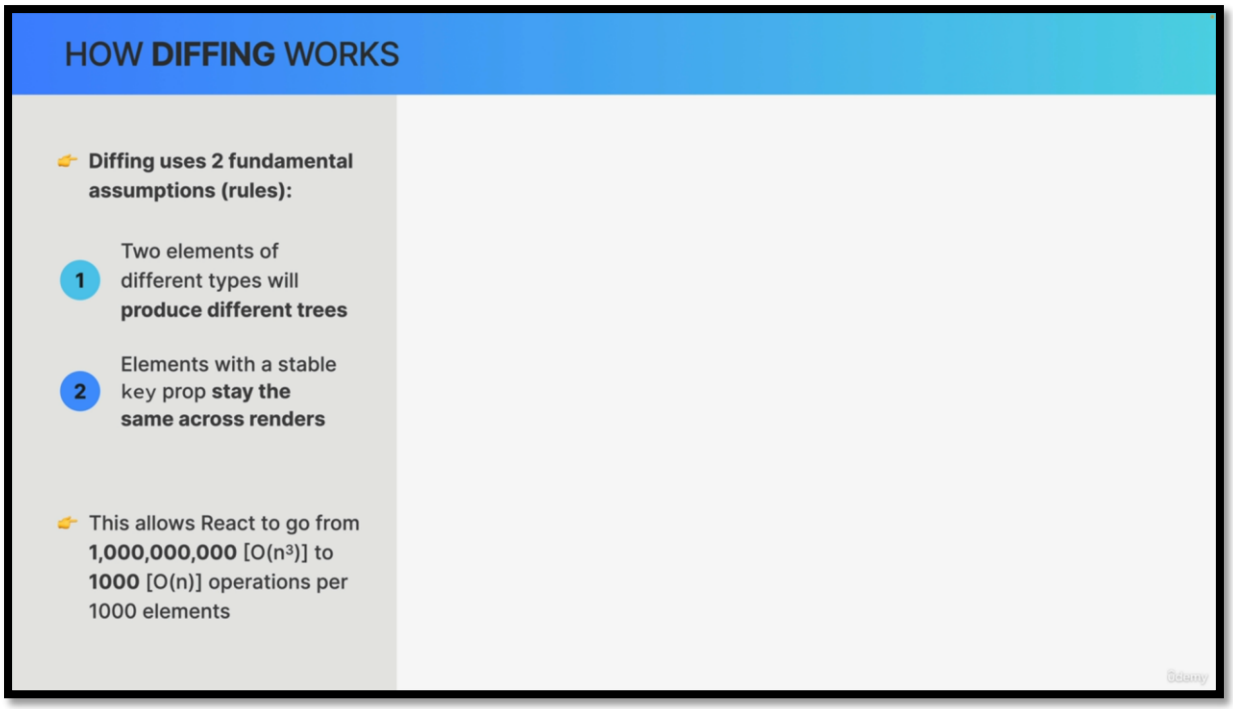
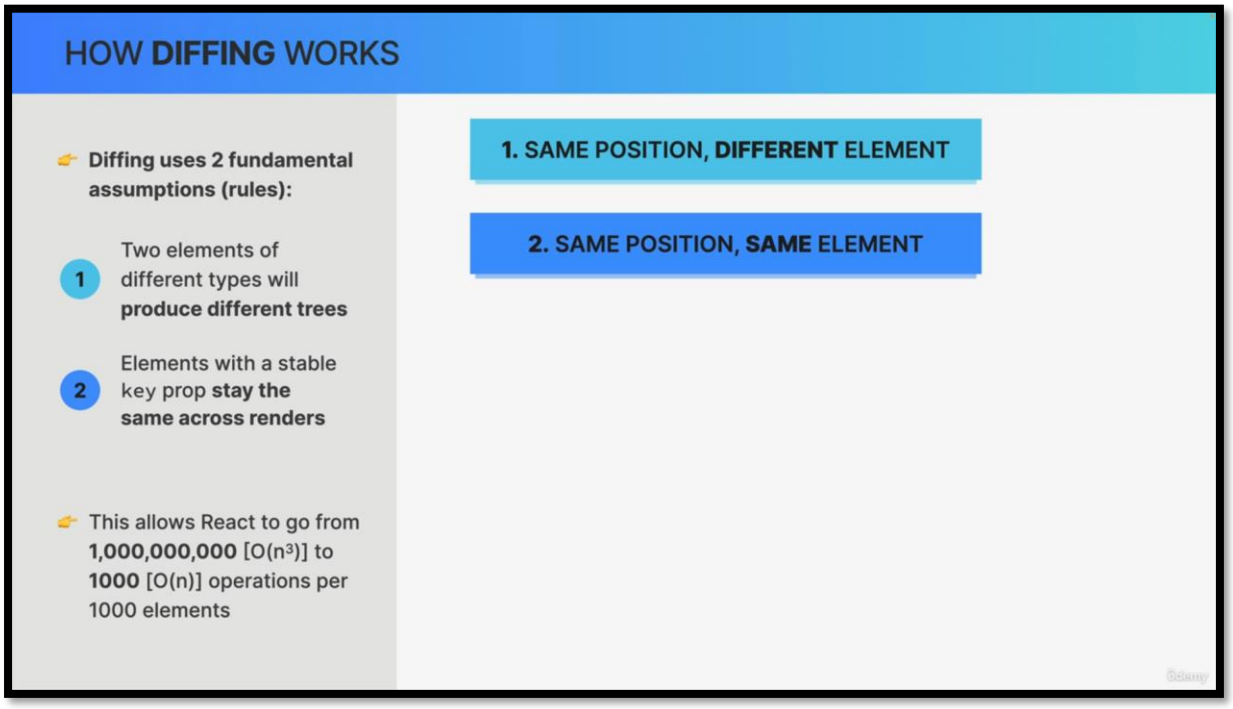


Diffing is first of all based on two fundamental assumptions.

1. The first one is that two elements of different types will produce different trees.

2. The second assumption is that elements with a stable key, a key that is consistent over time, will stay the same across renders.

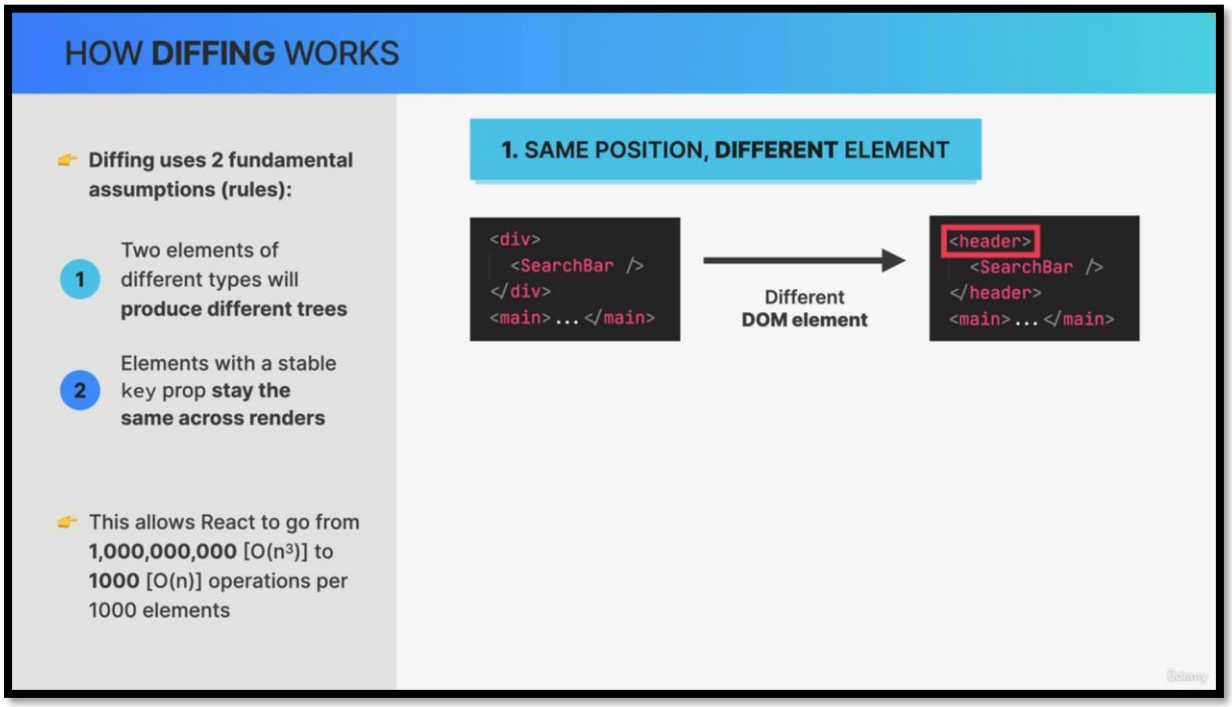
These assumptions may seem pretty obvious especially the first assumption but they allow the algorithm to be orders of magnitude faster going.



Diffing is comparing elements step-by-step between two renders based on their position in the tree. There are basically two different situations that need to be considered.

First, having two different elements at the same position in the tree between two renders.

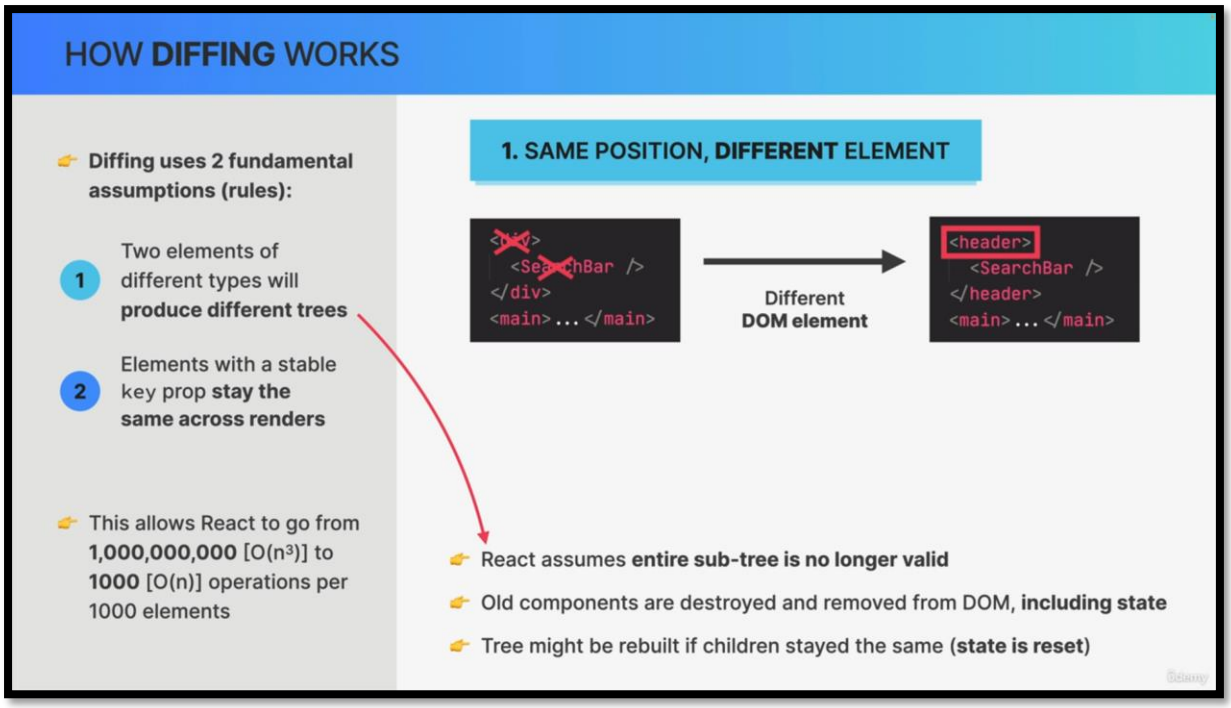
Second having the same element at the same position in the tree.



So, let's say that at some point an application is re-rendered, and in the diffing process we find that an element has changed in a certain position of the tree.

Now here we're actually not looking at a tree but at the JSX code, which leads to that tree.

In this case of a DOM element changing like this, changing simply means that the type of the element has changed like in this example from a div to a header.



So, in a situation like this, React will assume that the element itself plus all its children are no longer valid.

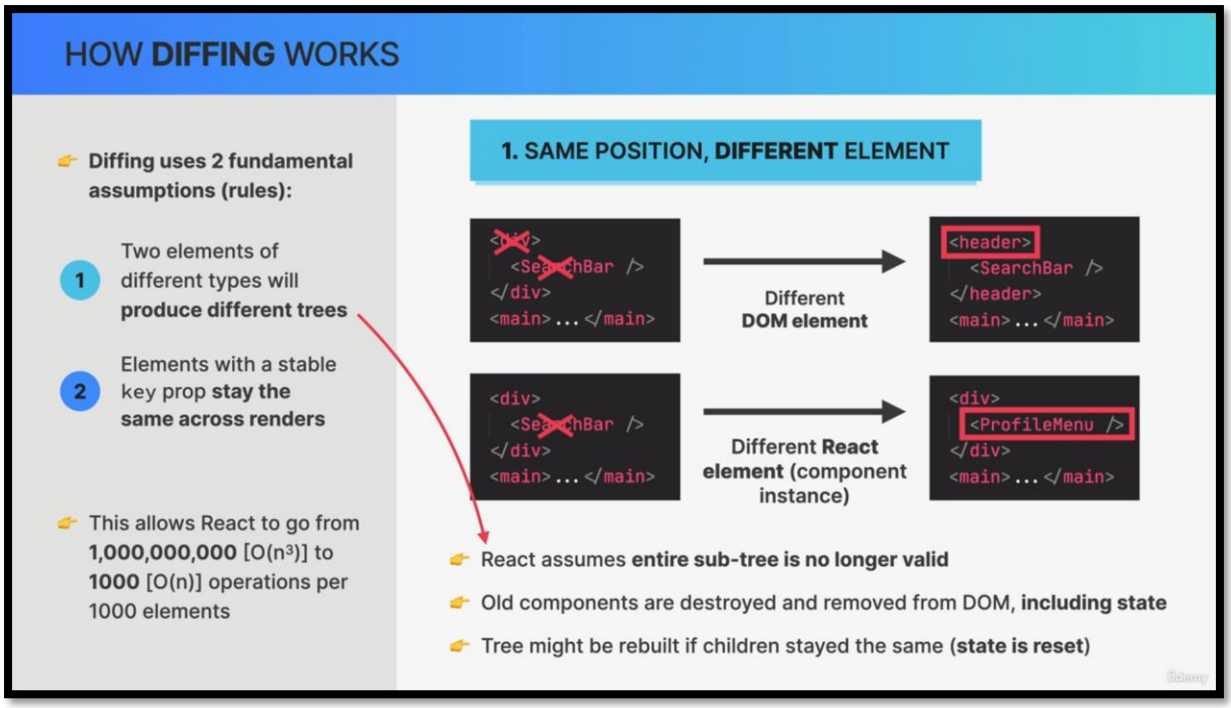
Therefore, all these elements will actually be destroyed and removed from the DOM. That also includes their state.

Both the div element and the search bar component will be removed from the DOM and will then be rebuilt as a header with a brand-new search bar component instance as the child.

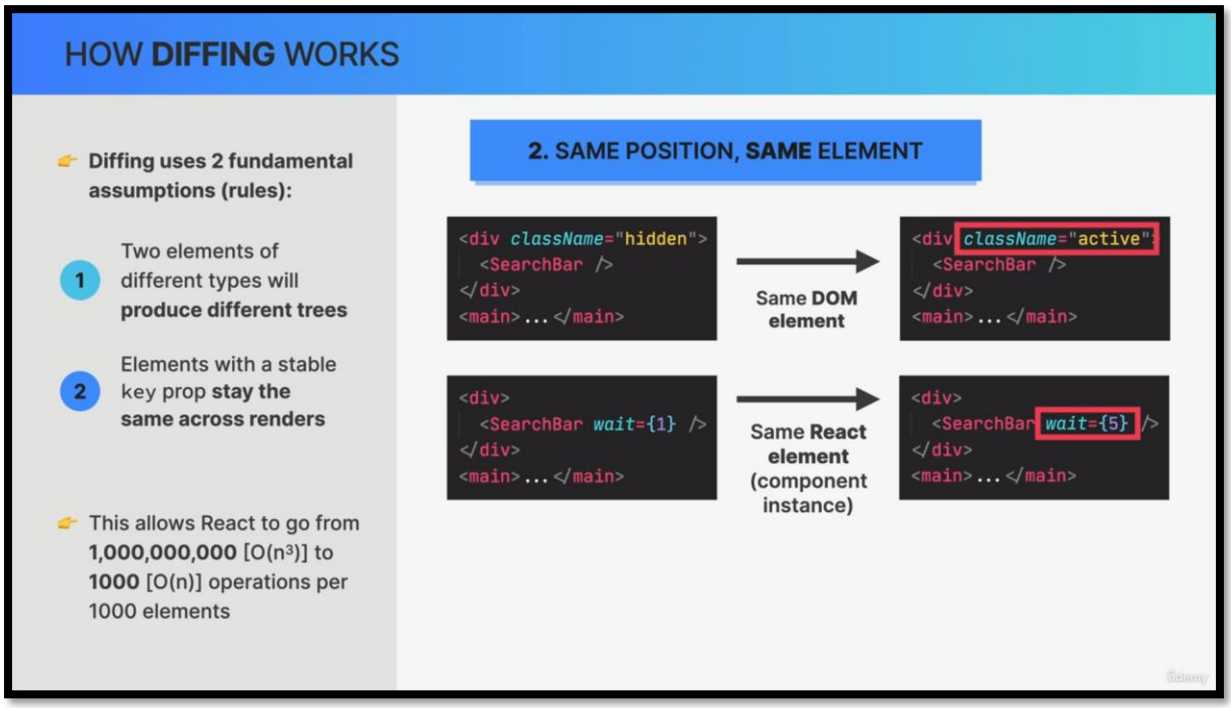
So, if the child element stays the same across renders, the tree will actually get rebuilt but it gets rebuilt with brand new elements.

So, if there were any components with state that state will not be recovered.

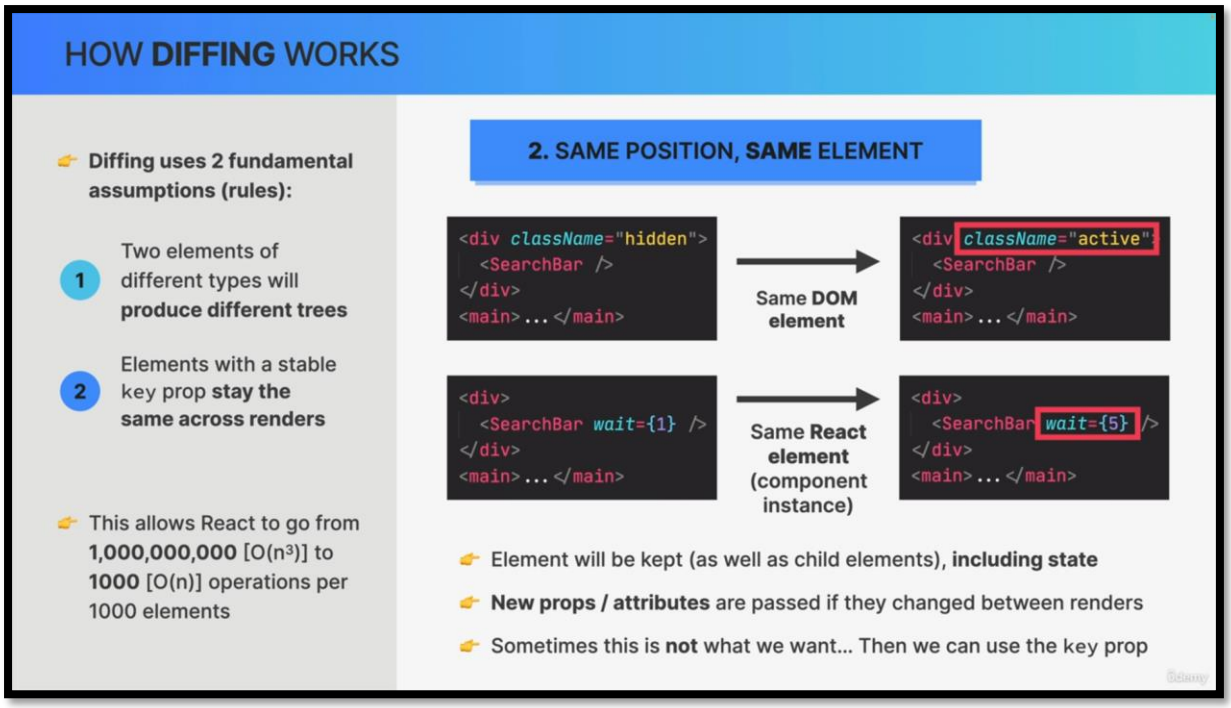
So, this effectively resets state and this has huge implications for the way that React applications work in practice.



In the second example, the search bar component changed to a profile menu component and therefore the search bar is again completely destroyed including its date and removed from the DOM.



The second situation is when between two renders we have the exact same element at the same position in the tree.



If after a render, an element at a certain position in the tree is the same as before, like in these examples right here, the element will simply be kept in the DOM. That includes all child elements and more importantly, the components state.

Looking at these examples we actually see that something has changed. However, it was not the element type that has changed but simply the class name attribute in the div and the wait prop in the search power component.

So, what React is gonna do is to simply mutate the DOM element attributes. In the case of React elements it'll pass in the new props, but that's it.

React tries to be as efficient as possible and so the DOM elements themselves will stay the same. They're not removed from the DOM, and even more importantly the state will not be destroyed.