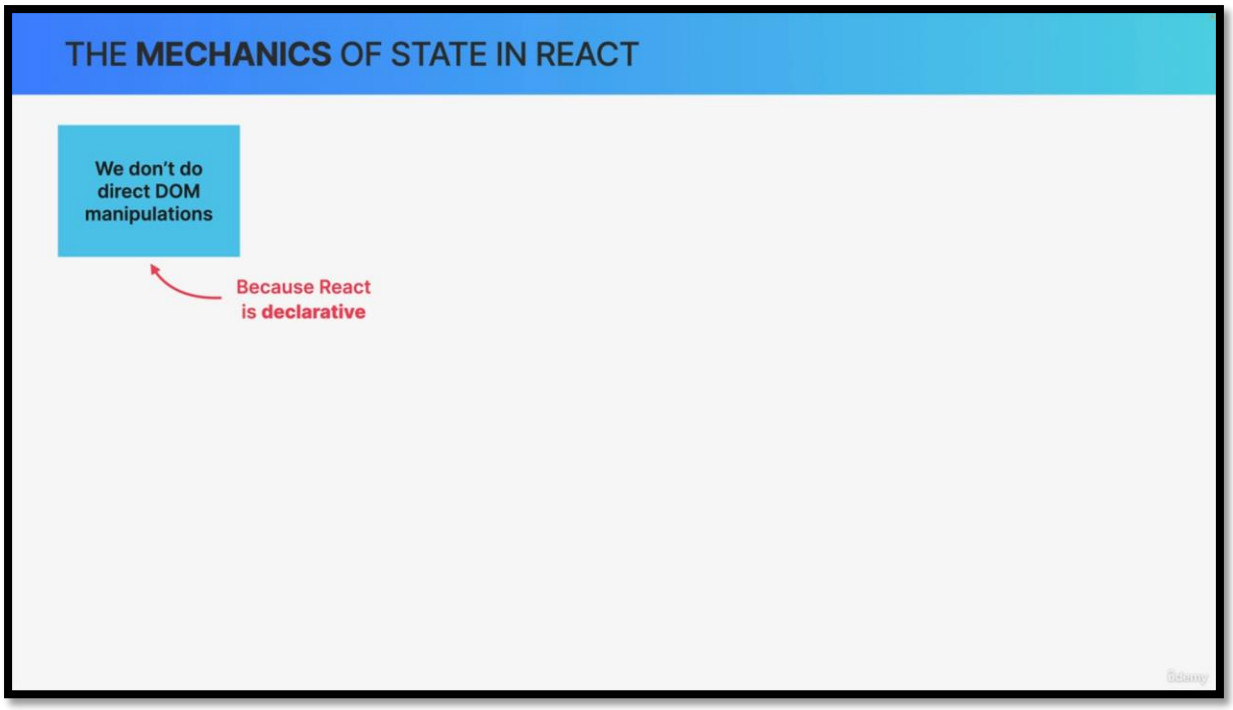
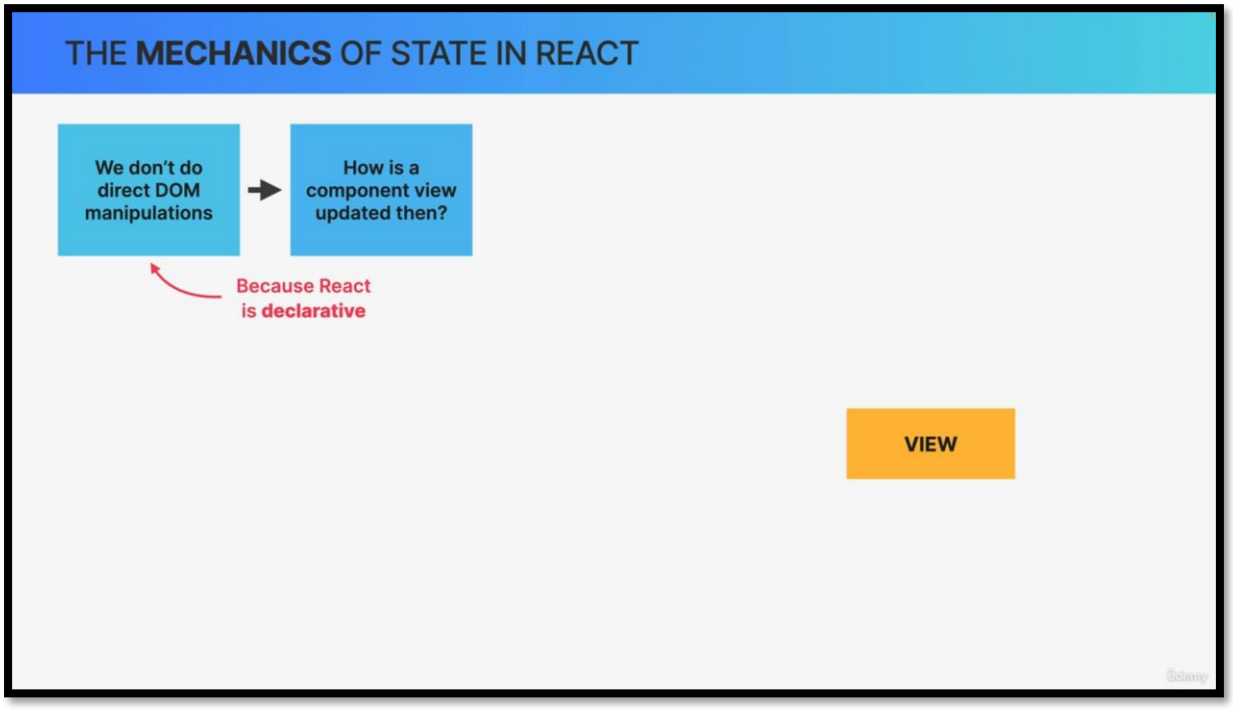
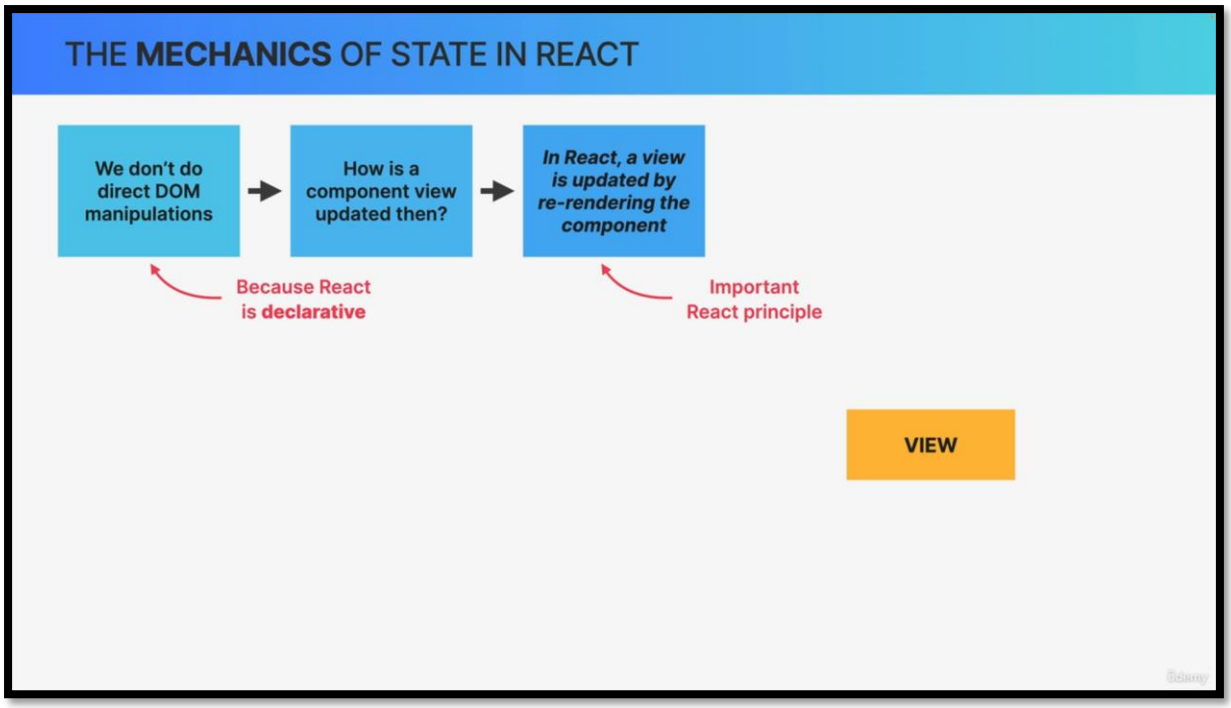


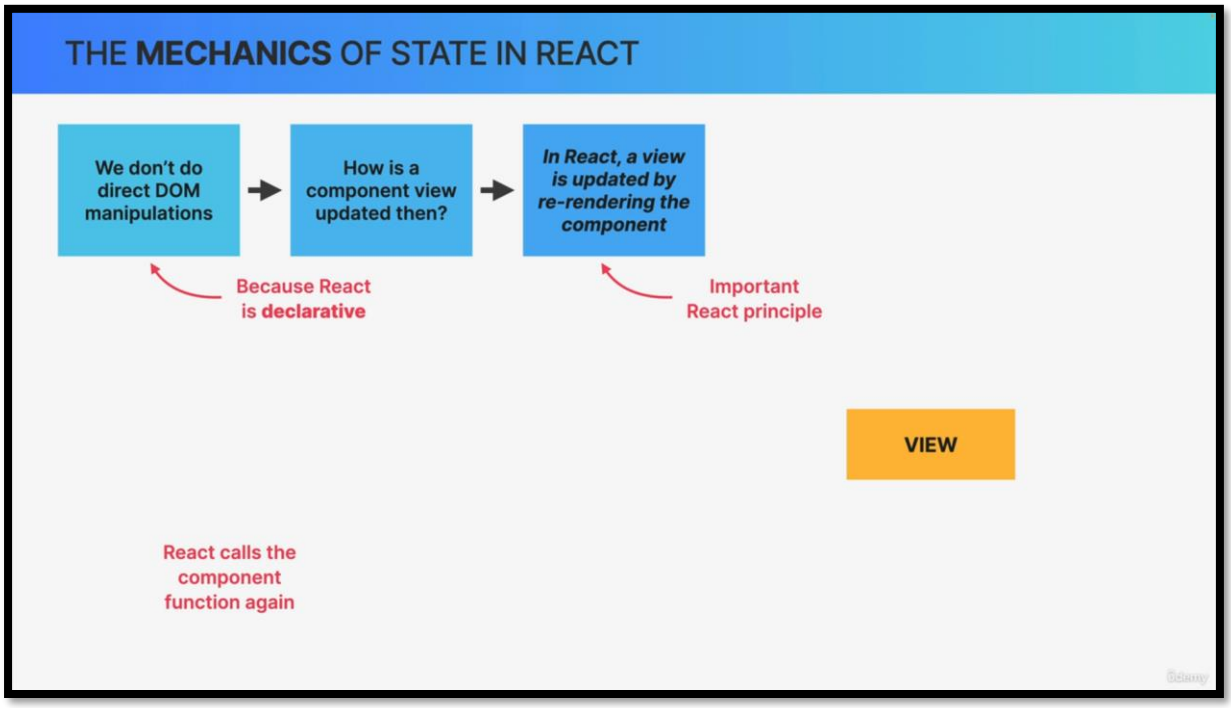
We do not manipulate the DOM directly when we want to update a component's view. Because React is declarative, not imperative.



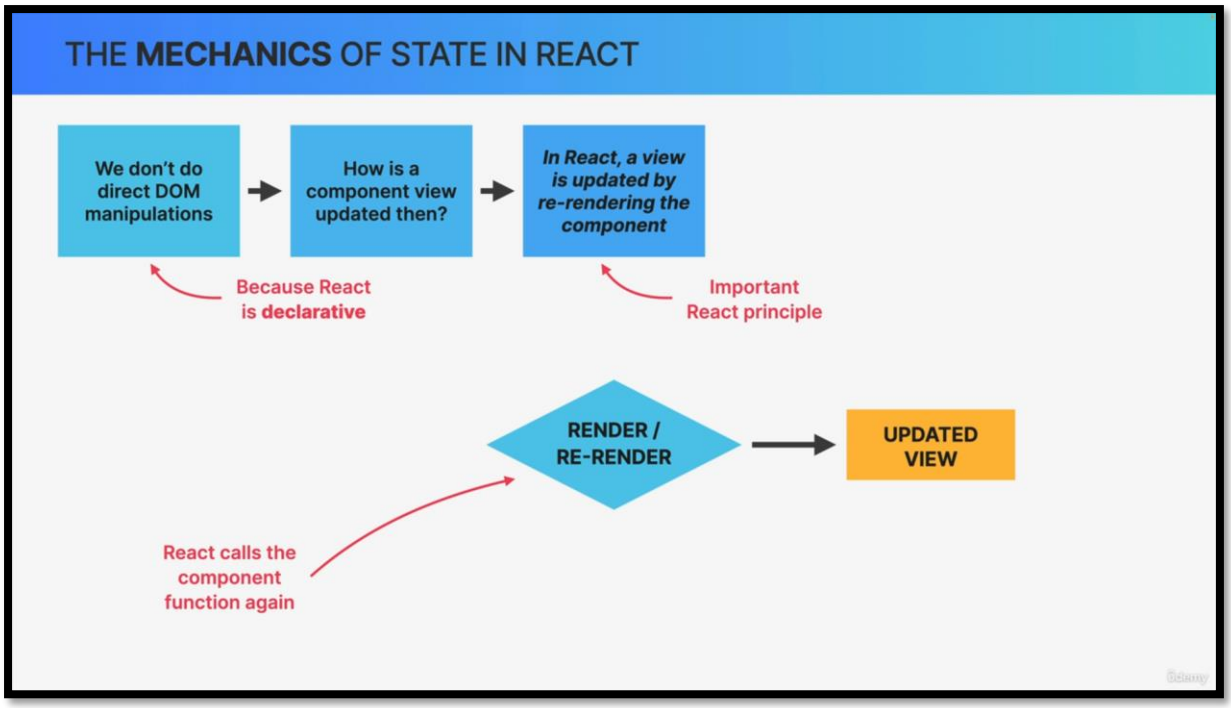
But if that's the case then this leads us to the question of how do we update the component on the screen whenever some data changes or whenever we need to respond to some event like a click?



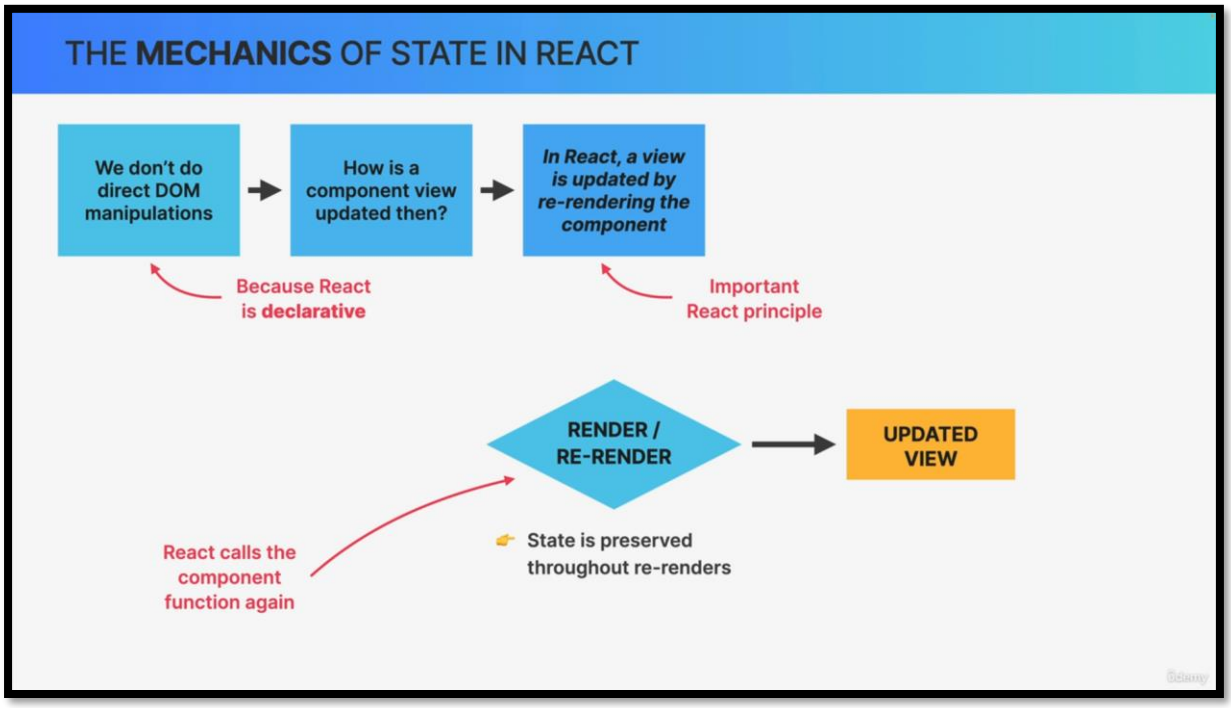
React updates a component view by re-rendering that entire component whenever the underlying data changes.



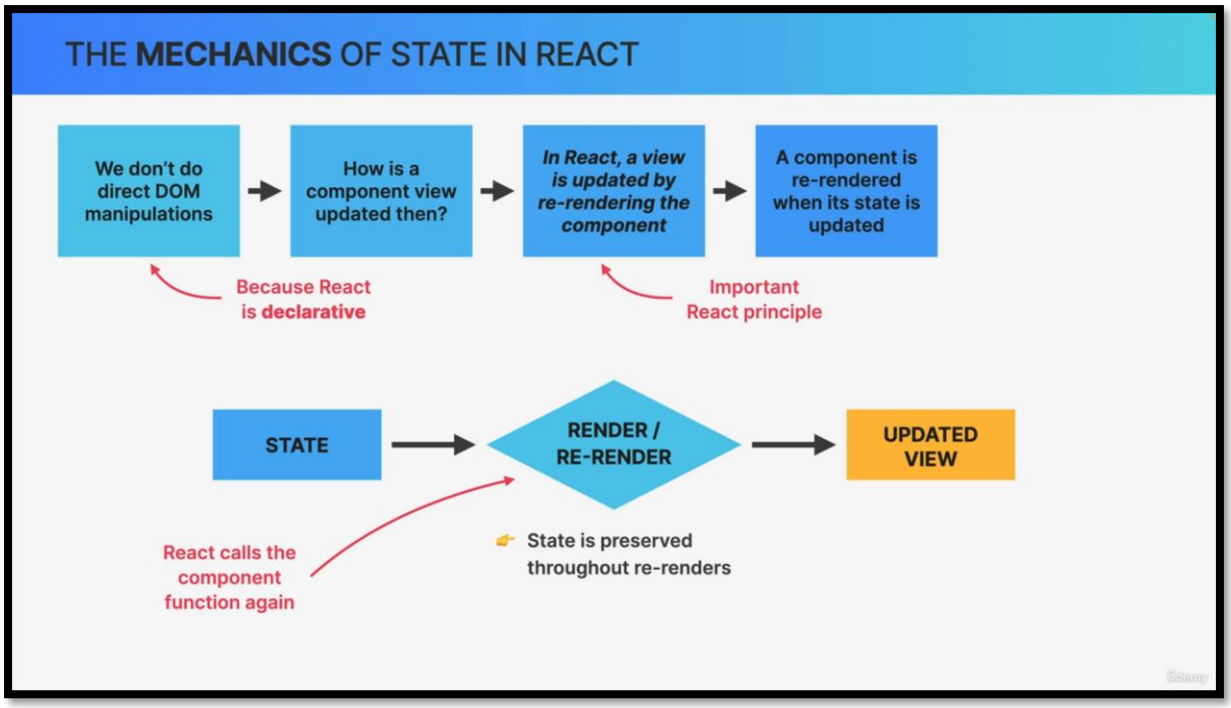
Re-rendering basically means that React calls the component function again each time the component is rendered.

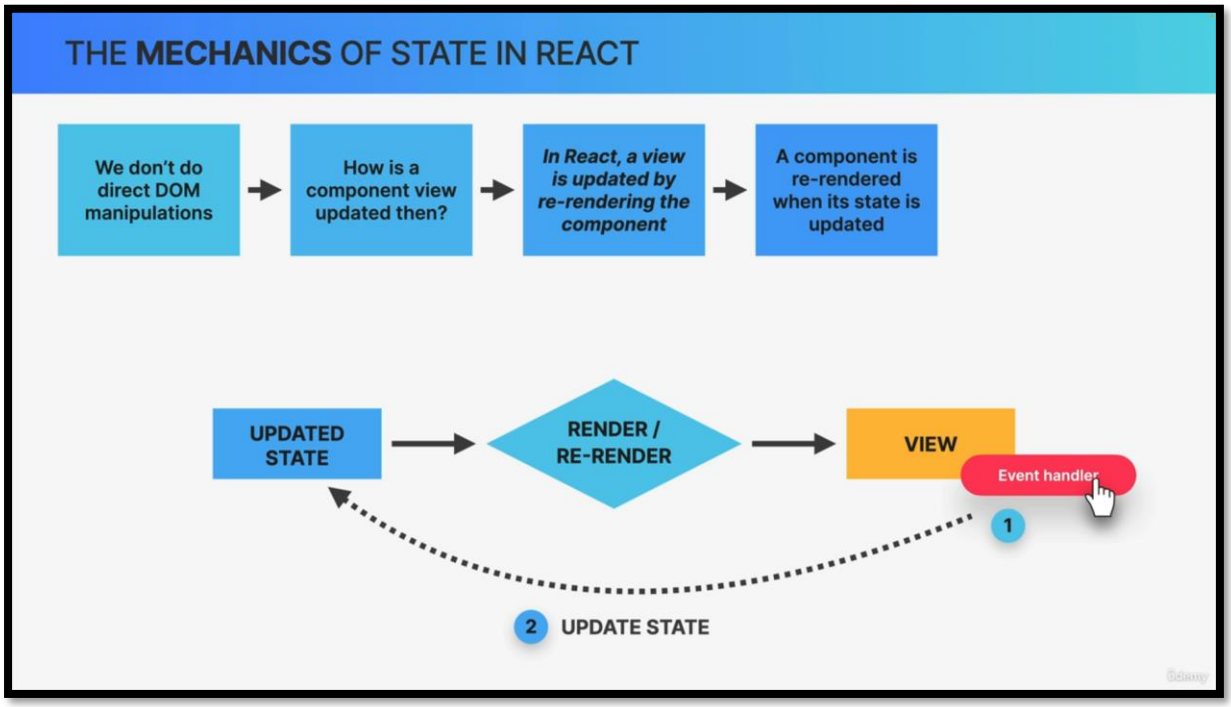


Conceptually we can imagine this as React removing the entire view and replacing it with a new one each time a re-render needs to happen.

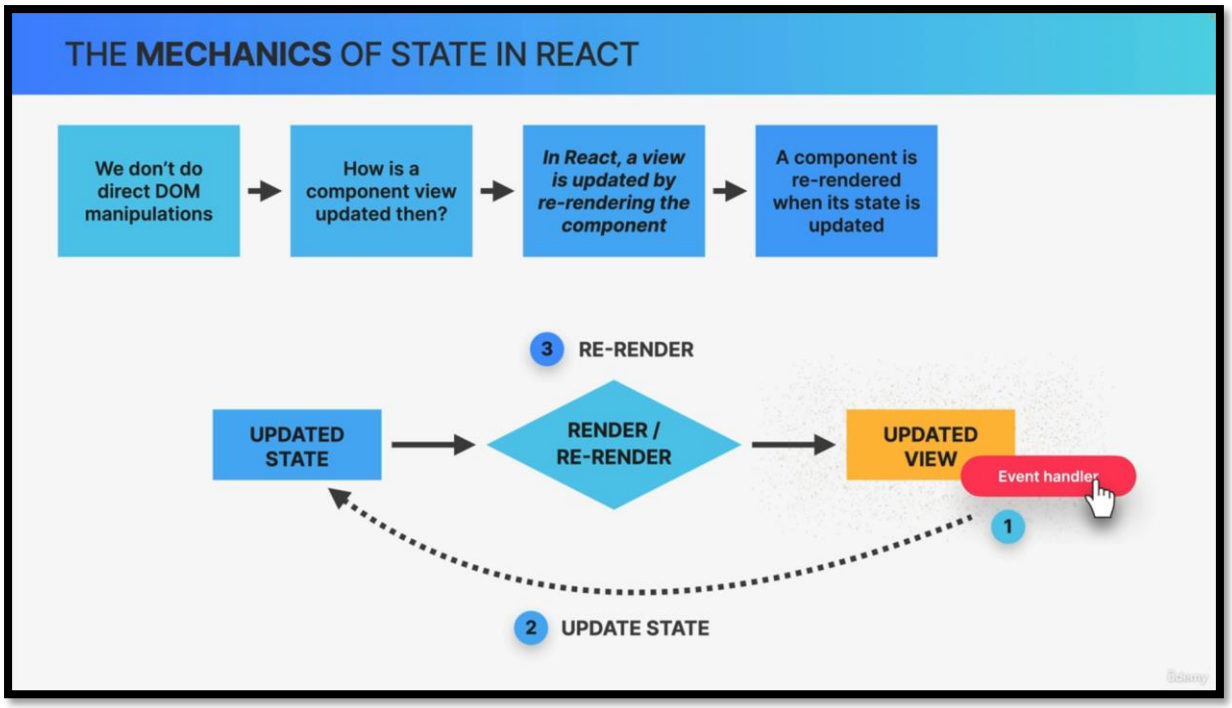


React preserves the component state throughout re-renders and even though a component can be rendered and re-rendered time and time again, the state will not be reset unless the component disappears from the UI entirely, which is what we call unmounting.



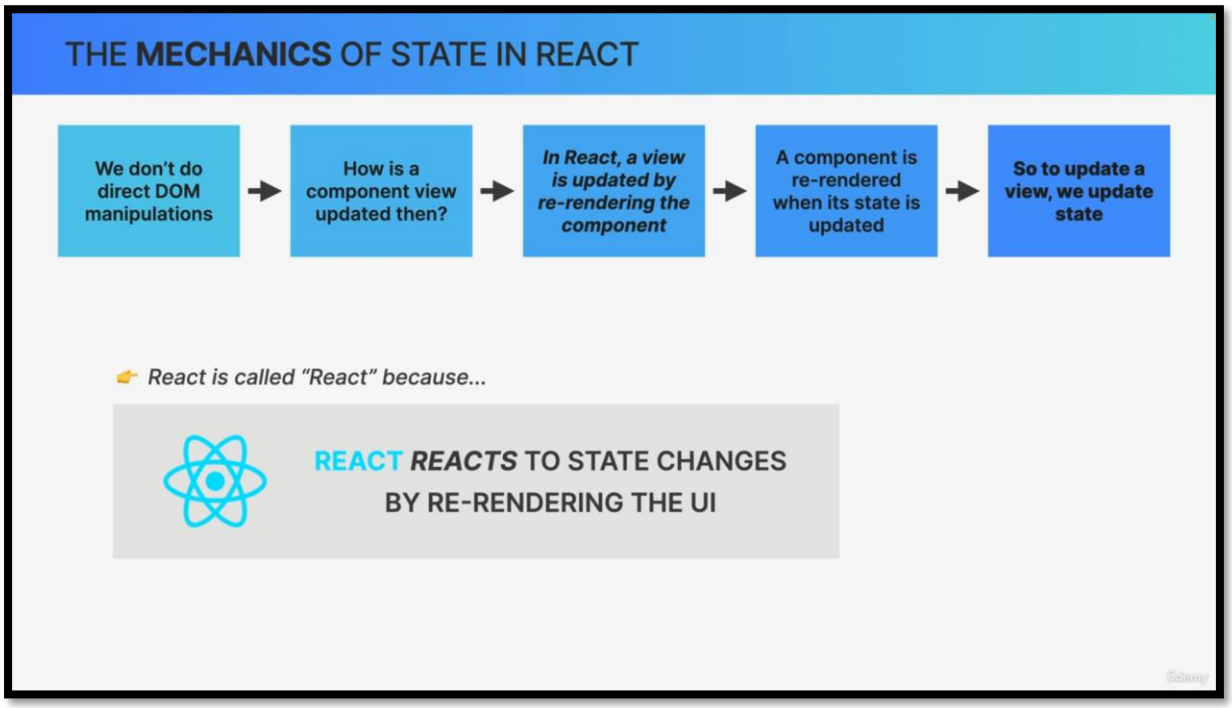


Let's imagine that there is an event handler in the view, for example, on a button that the user can click.

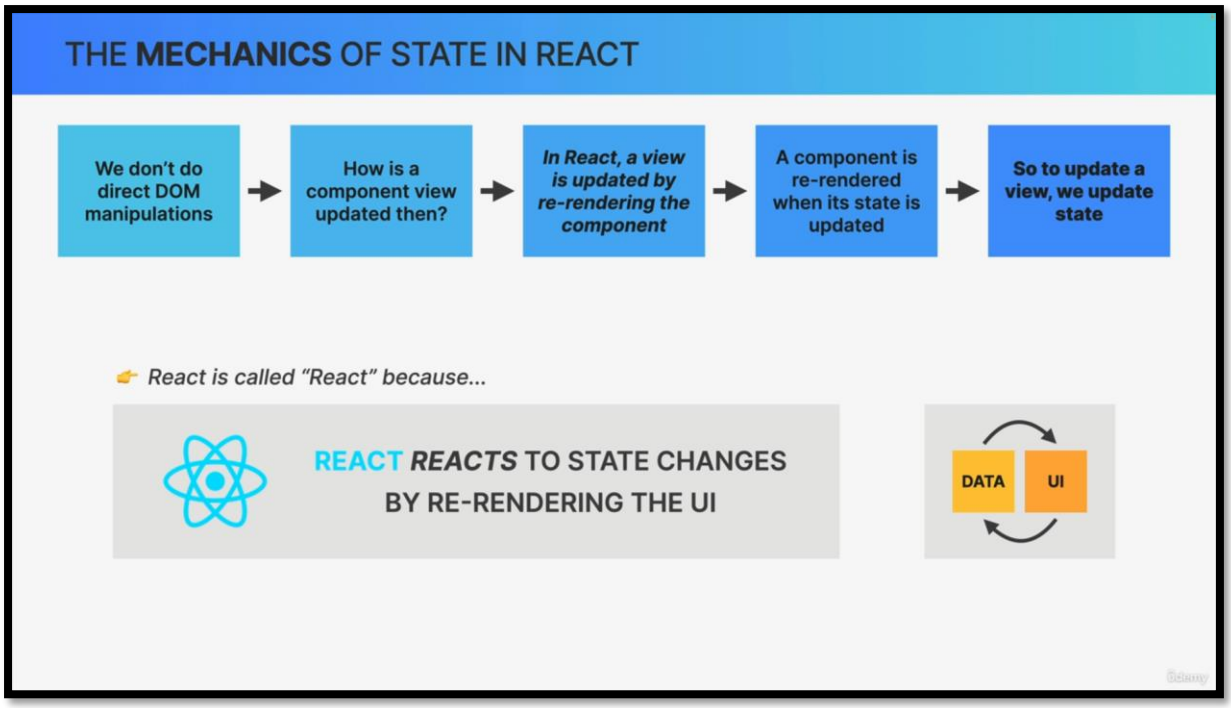


The moment that button is clicked, we can update a piece of state in our component using the setter function coming from the useState hook.

When React sees that the state has been changed, it'll automatically re-render the component which will result in an updated view for this component.



As React developers, whenever we want to update a component view, we update its state.



React will then react to that update and do its things. This whole mechanism is so fundamental to React that it's actually the reason why React is called React in the first place.

This is how REACT keeps UI in sync with data.