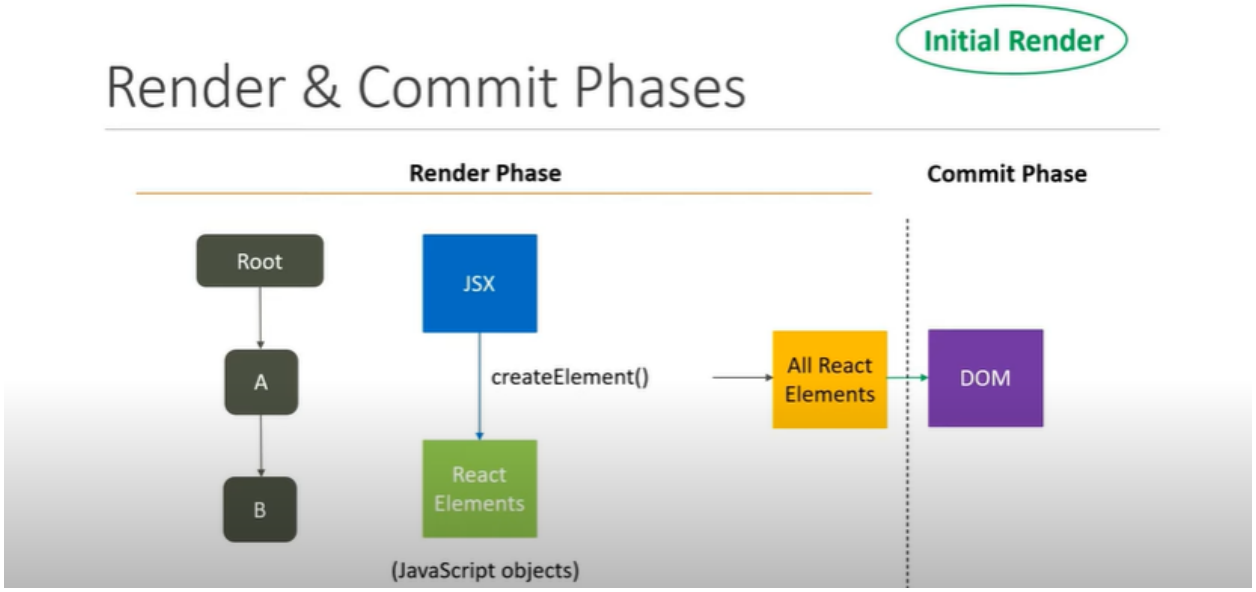
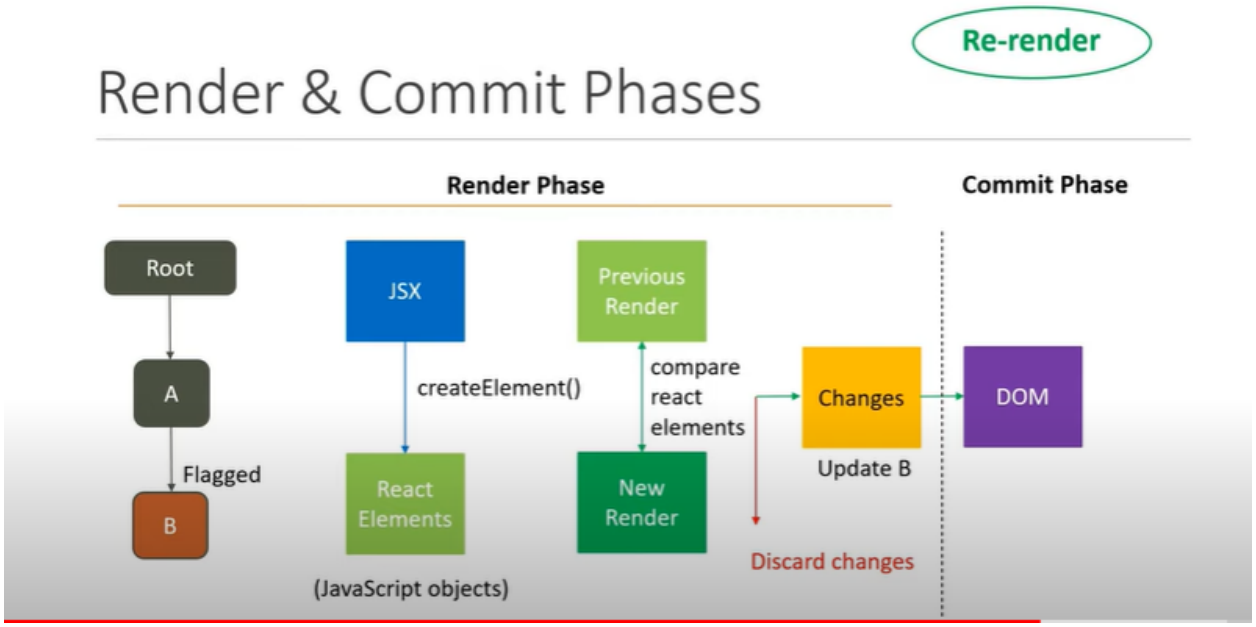




COMMIT PHASE IS USUALLY VERY FAST ,BUT RENDERING CAN BE SLOW

USE STATE vid-3

1. The setter function from a useState hook will cause the component to re-render
2. The exception is when you update a state hook to the same value as the current state.
3. Same value after the initial render? the component will not re-render
4. Same value after re-renders? react will render that specific component one more time and then bails out from any subsequent renders

USE REDUCER vid-4

1. The dispatch function from a useReducer hook will cause the component to re-render
2. The exception is when you update a state to the same value as the initial state.
3. Same value after the initial render? the component will not re-render
4. Same value after re-renders? react will render that specific component one more time and then bails out from any subsequent renders

STATE IMMUTABILITY vid-5

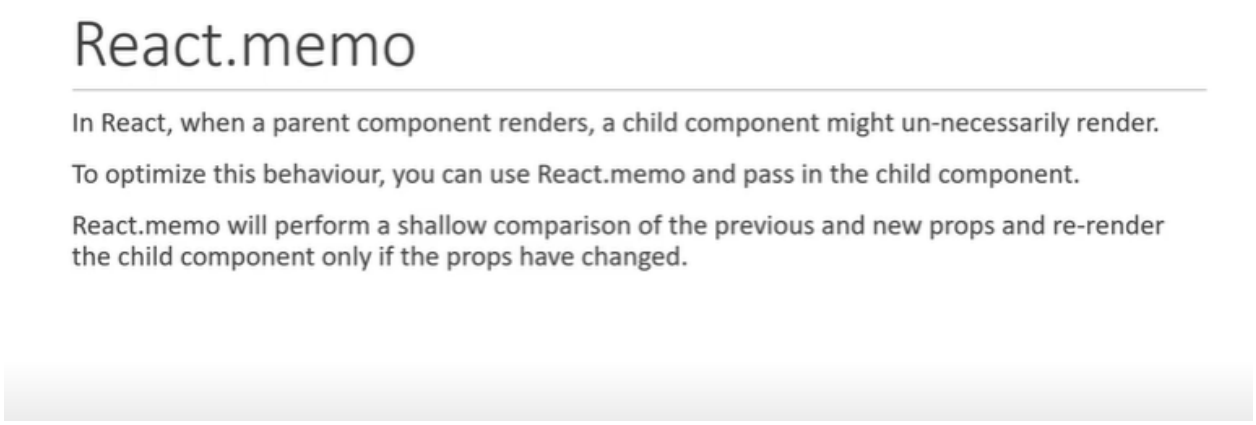
1. Mutating an object or an array as state will not cause a re-render when used with the useState or useReducer hook.
2. To re-render, make a copy of the existing state modify as necessary and then pass the new state to the setter function or while returning from a reducer function.
3. Directly mutating the state is an easy way to create bugs in your application. Make sure you don't do that!

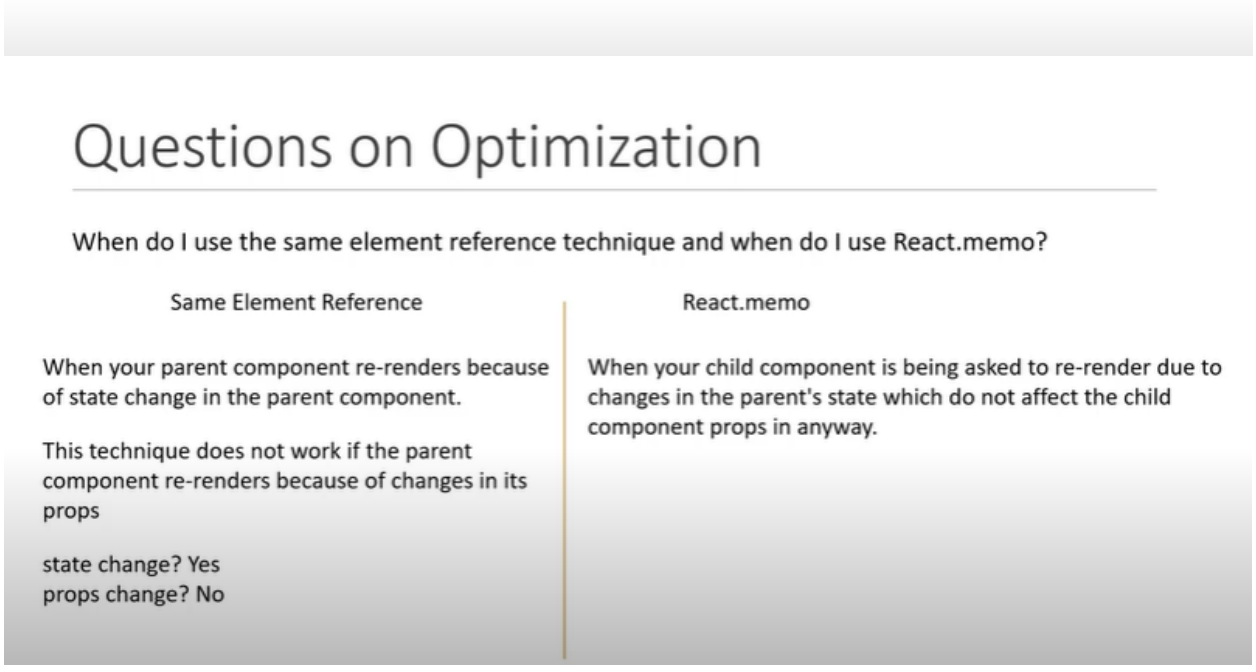
PARENT AND CHILD RENDERS vid-6

1. When a parent components renders react will automatically recursively render all of its child components.
2. New state same as old state after initial render? Parent & Child won't re-render.
3. New state same as the old state after re-render? Parent will re-render once, but child never re-renders.
4. Unnecessary renders
    a. Button click -> parent re-renders -> child component re-renders.
    b. DOM represented by child is never updated.
    c. Child went through render phase but not through the commit phase
    d. Unnecessary renders affect performance.

SAME COMPONENT RENDER vid-7

1. In react when a parent component renders it'll automatically render all of its child components.
2. "Unnecessary renders" when child goes through the render phase but not through the commit phase.
3. (*solution*)You can extract expensive child component and pass it as a prop to the parent component.
4. Whenever there is re-render caused by a state of the parent component (*grand parent*), react will optimize the re-render for you by knowing that the props has to referencing the same elements before and after the render.





Questions on Optimization

If React.memo provides the optimization by comparing the props, why not wrap every single component with React.memo?

Why doesnt React just internally memoize every component and not expose React.memo to the developers?

"Shallow comparisons aren't free. They're O(prop count). And they only buy something if it bails out.

All comparisons where we end up re-rendering are wasted. Why would you expect always comparing to be faster? Considering many components always get different props."

- Dan Abramov

React.memo and hypothetical time

Component render time – ***10ms***

React.memo time – ***2ms***

First re-render – No props change – ***2ms***

Second re-render – props change – ***2ms + 10ms = 12ms***

Component render history

Initial render – 10ms

Optimized render – 2ms

Re-render – 12ms

Re-render – 12ms

Re-render – 12ms

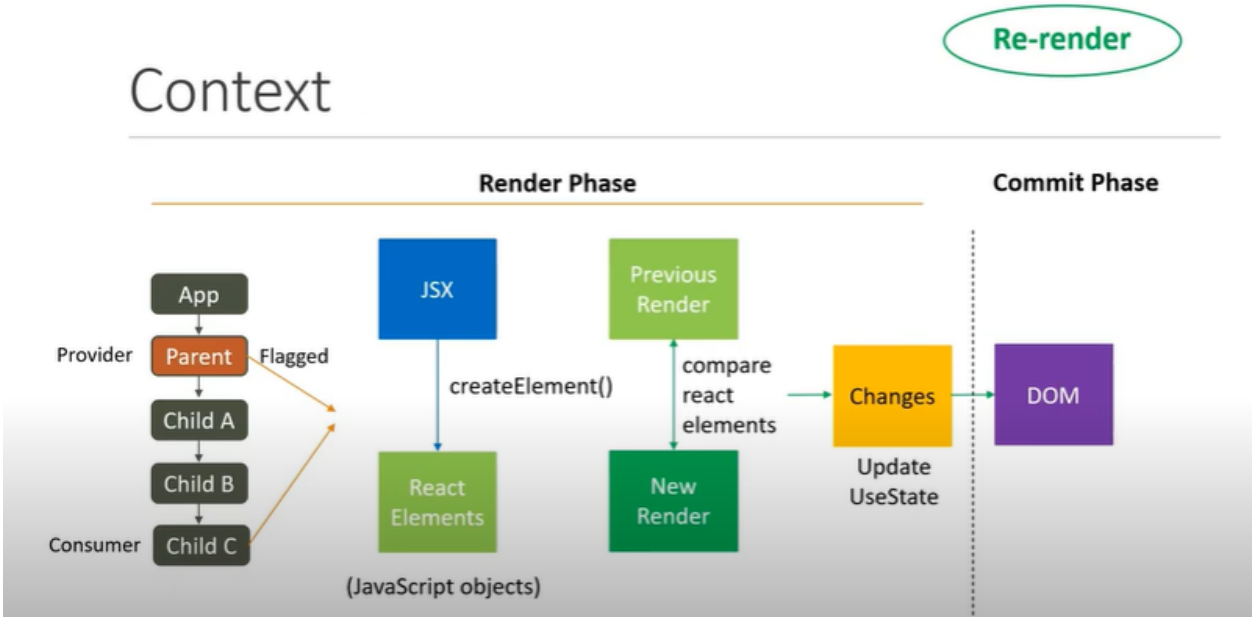
Re-render – 12ms

Render Optimization

When you optimize the rendering of one component, React will also skip rendering that component's entire subtree because it's effectively stopping the default ***"render children recursively"*** behavior of React.

NOTES to AVOID INCORRECT USING REACT MEMO

1. No need to use **Memo** in child component if child already has children.. As children are always considered new references.
2. Is child is displaying date/time it has to re-render .. don't use memo in such cases.
3. If parent is passing function/ object to the child component, new reference is created with each re-render to parent component, hence child will also re-render ,even though we have used useMemo
4. Fix for 3.
    a. Use **useMemo** to object and pass memoized object as props to child.
    b. Use **useCallback** to function and pass memoized function as props to child.



- Above is ideal that should happen i.e. only parent and child C must re-render, when corresponding context value changes.
- But when context value is changes parent stated changes and therefore all the children of parent also re-renders

Context and Render

When the context provider is in the parent component and the parent component's state updates, every child component re-renders and not just the component consuming the context value

Fix1. useMemo on ChildA (now only parent and child C will re-render)
Fix2. pass childA as a prop to parent (so now when state in parent changes it has no reason to change the props)