**[NOTES] Virtual DOM vs 'real' DOM - rerender**

**The DOM vs Virtual DOM:**

* Real/Actual DOM:
  + The actual structure of HTML elements in your browser
  + Updating it is "expensive" (slow) because:
    - Browser needs to recalculate layouts
    - Repaint the screen
    - Process CSS changes
* Virtual DOM:
  + A lightweight JavaScript copy of the real DOM
  + Lives in memory
  + Much faster to update because it's just JavaScript objects

1. How React's Rerender Process Actually Works:
2. function Example() {
3. const [count, setCount] = useState(0);
5. return (
6. <div>
7. <h1>Count: {count}</h1>
8. <button onClick={() => setCount(count + 1)}>Add</button>
9. </div>
10. );
11. }

When this component rerenders, here's what happens:

1. Virtual DOM Update:
   * React first rerenders to the Virtual DOM
   * Creates new Virtual DOM tree with updated values
   * This is fast because it's just updating JavaScript objects
2. Diffing ("**Reconciliation**"):
   * React compares new Virtual DOM with previous version
   * Identifies what actually changed (in this case, just the count number)
3. Real DOM Update:
   * Only the necessary changes are applied to the real DOM
   * In our example, only the text content of the h1 would update

Here's a visual example:

1. // Initial render
2. <div>
3. <h1>Count: 0</h1>
4. <button>Add</button>
5. </div>
7. // After setState - Virtual DOM updates first
8. virtualDOM.update({
9. type: 'div',
10. children: [
11. { type: 'h1', children: 'Count: 1' },
12. { type: 'button', children: 'Add' }
13. ]
14. })
16. // React only updates this in real DOM:
17. document.querySelector('h1').textContent = 'Count: 1'
18. Key Benefits:

* Performance: Instead of updating everything, React only updates what actually changed
* Batch Updates: React can bundle multiple changes together
* Cross-Platform: Virtual DOM can be rendered to different targets (web, mobile, etc.)

1. Common Misconception: When we say "React rerenders", we mean:

* Component function runs again ✅
* New Virtual DOM tree is created ✅
* Diffing occurs ✅
* Minimal necessary real DOM updates happen ✅

NOT:

* Entire real DOM rebuilds ❌
* Everything repaints ❌

1. Why This Matters:
2. function ParentComponent() {
3. const [count, setCount] = useState(0);
5. return (
6. <div>
7. <h1>Count: {count}</h1>
8. <ExpensiveComponent data={someData} />
9. <button onClick={() => setCount(count + 1)}>Add</button>
10. </div>
11. );
12. }

Even though the whole component "rerenders", React's Virtual DOM means:

* Only the count text actually updates in the real DOM
* If someData hasn't changed, ExpensiveComponent might not even need to update

When Does React Rerender?

Let's understand rerenders through a practical example:

1. function Counter({label}) {
2. const [count, setCount] = useState(0);
3. console.log('Counter component is rerendering!');
5. return (
6. <div>
7. <h1>{label}</h1>
8. <p>Count: {count}</p>
9. </div>
10. );
11. }
13. function App() {
14. const [label, setLabel] = useState("My Counter");
15. console.log('App component is rerendering!');
17. return (
18. <div>
19. <Counter label={label} />
20. <button onClick={() => setLabel("My Counter")}>
21. Update Label (same value)
22. </button>
23. <button onClick={() => setLabel("New Label")}>
24. Update Label (new value)
25. </button>
26. </div>
27. );
28. }

Three Main Rerender Triggers:

1. **State Changes**:
2. // Clicking this button WILL cause Counter to rerender
3. <button onClick={() => setLabel("New Label")}>

* Component function runs again
* New Virtual DOM is created
* Real DOM updates because content changed

1. **Props Changes**:
2. <Counter label={label} />

* When parent changes label, Counter rerenders
* If new label is different, DOM updates
* If new label is same value, DOM doesn't update (but component still rerenders!)

1. **Parent Rerenders**:

* When App rerenders, Counter always rerenders by default
* This happens even if props didn't change
* **But DOM only updates if Virtual DOM is different**

Common Beginner Misconceptions:

❌ "Rerender means the DOM updates"

✅ Rerender means the component function runs again, DOM may or may not update

❌ "Preventing rerenders should be your first optimization"

✅ React's Virtual DOM diffing is usually efficient enough

Common Mistakes:

1. // ❌ Bad: Infinite rerenders
2. function BadCounter() {
3. const [count, setCount] = useState(0);
4. setCount(count + 1); // Triggers rerender, which triggers another setCount, and so on
5. return <div>{count}</div>;
6. }
8. // ✅ Good: Controlled rerender
9. function GoodCounter() {
10. const [count, setCount] = useState(0);
12. // State update only happens on user action
13. const handleClick = () => setCount(count + 1);
15. return (
16. <div>
17. {count}
18. <button onClick={handleClick}>Add</button>
19. </div>
20. );
21. }

Key Points:

1. Rerender = Component function execution
2. Rerender ≠ DOM update
3. Virtual DOM comparison determines if DOM updates
4. Child components rerender when parents do
5. State/prop changes trigger rerenders
6. Only optimize if you have performance issues