

Re-Rendering

- Before we Learn about re-rendering we must know what rendering is.
- Rendering in React means turning your JSX code into actual HTML that appears on the screen.
- For example :

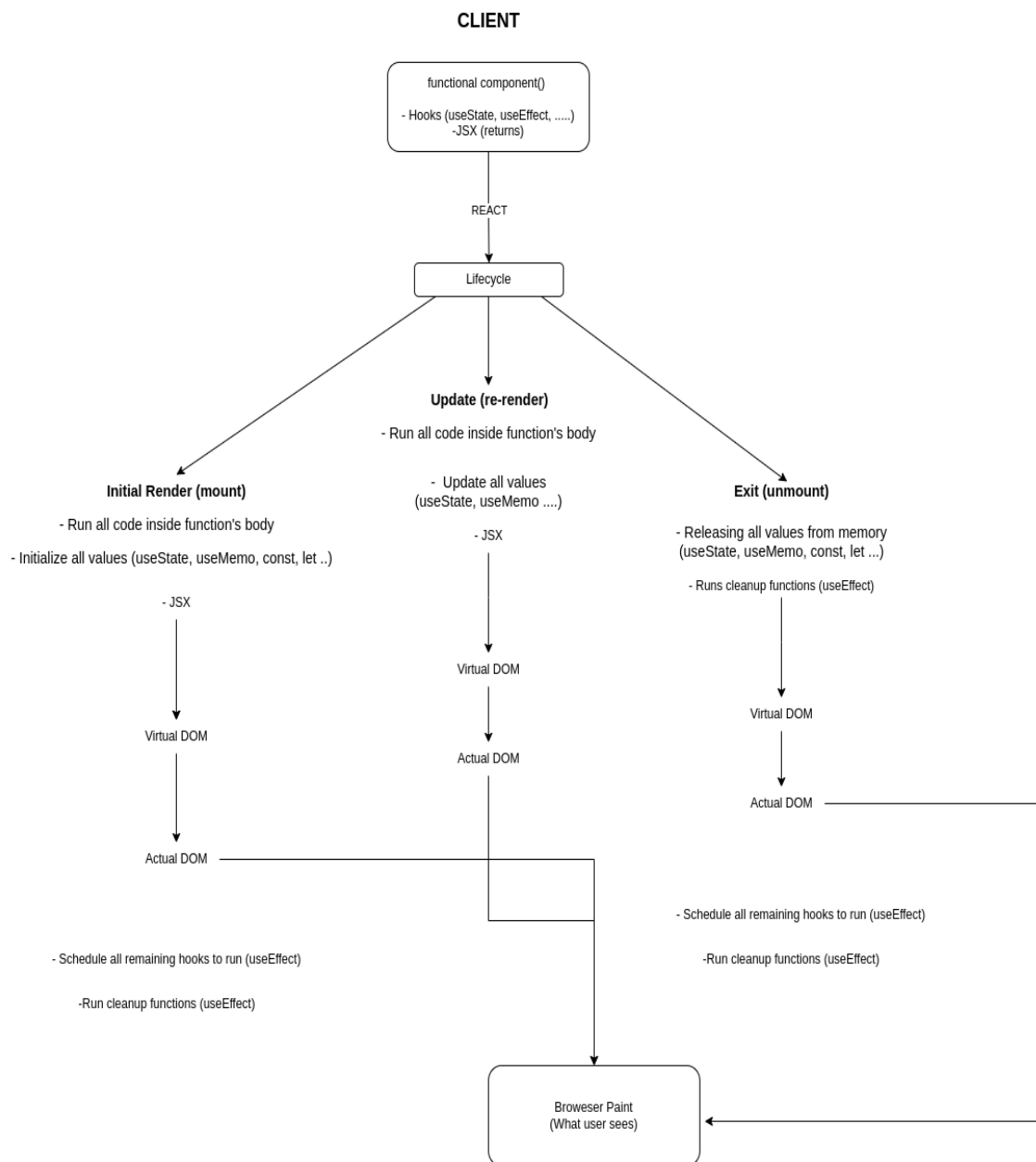
```
function Greeting() {  
  return (  
    <div>  
      <h1>Hello, Anshul!</h1>  
      <p>Welcome to React!</p>  
    </div>  
  );  
}
```

- Above is an example of JSX code and we know that JSX is not regular javascript. The browser doesn't understand it directly. React uses a tool like Babel to convert it into something like:

```
React.createElement("div", null,  
  React.createElement("h1", null, "Hello, Anshul!"),  
  React.createElement("p", null, "Welcome to React!")  
);
```

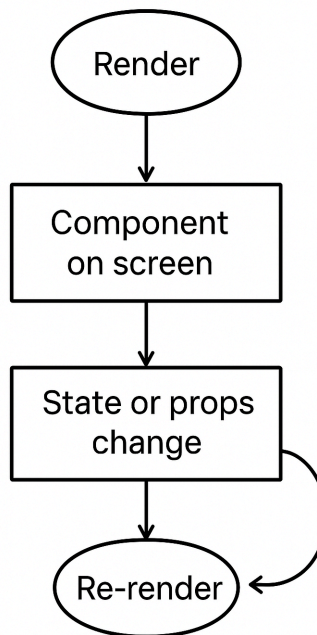
- After React renders it, the browser sees this:

```
<div>  
  <h1>Hello, Adam!</h1>  
  <p>Welcome to React!</p>  
</div>
```

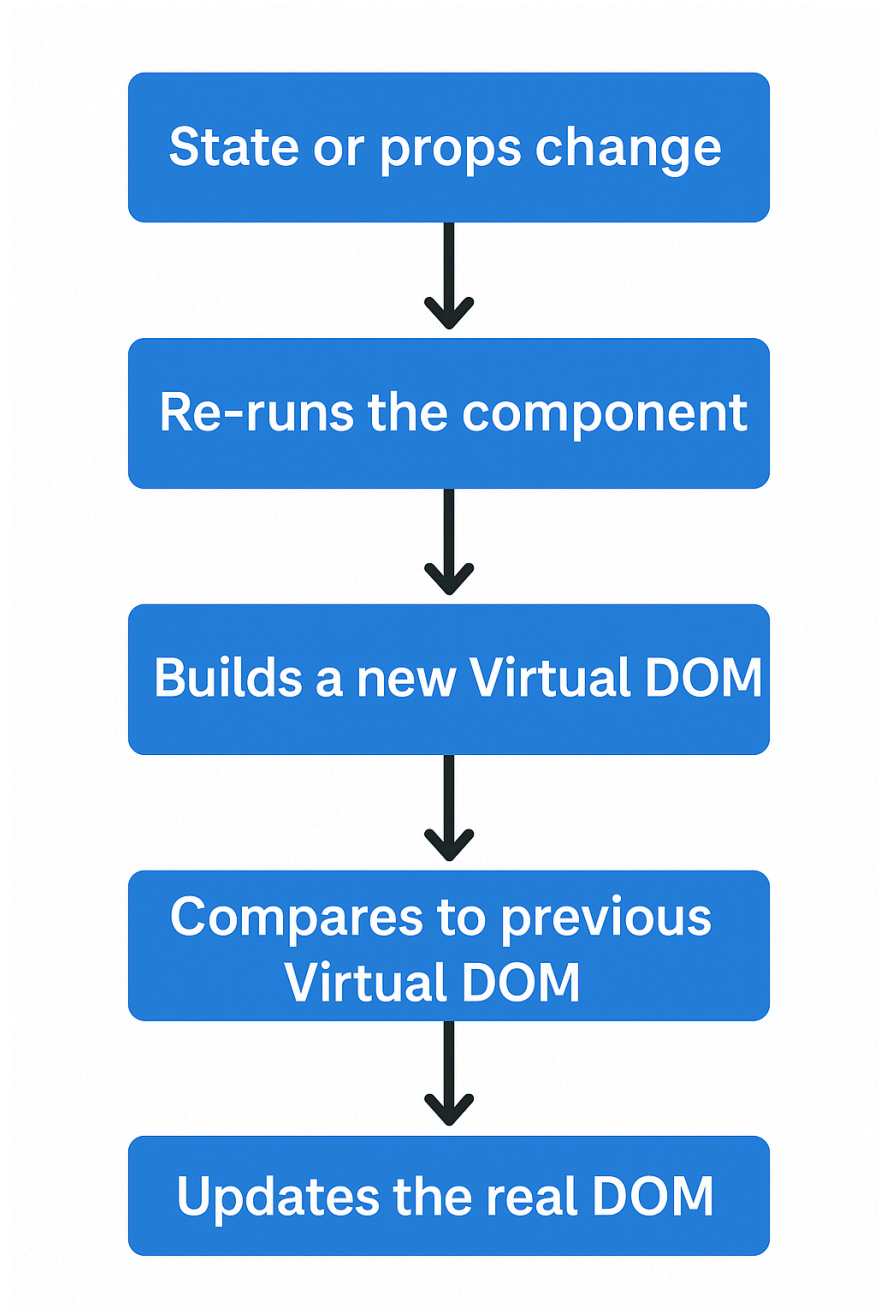


What is rerendering?

- Now onto the topic,
 - Re-rendering in React means:
 - React updates the UI by running the component function again to reflect the latest changes in state or props.
 - Re-rendering means React is updating the screen
 - It happens when something in your component changes — like state or props.
 - React redraws the part that changed
 - It doesn't reload the whole page. Just the part that needs to change.
 - Example:
 - If you click a button that updates a counter, React re-renders the number on the screen to show the new count.



During Rerender Process flow will be:



Why You Should Not Manually Update the DOM in React

- **React controls the DOM through the Virtual DOM**

React maintains a virtual version of the DOM and decides what changes are needed. If you manually update the real DOM, React has no idea — and it might undo or ignore your changes.

- **It breaks React's declarative approach**

React is declarative: you describe what the UI should look like, and React takes care of how to make it happen. Manual DOM updates are imperative — they confuse React's flow and make your code harder to manage.

- **You create multiple sources of truth**

React's single source of truth is its state/props. Manually changing the DOM adds a second, unmanaged source — which causes UI mismatches, glitches, or errors.

- **React may overwrite your changes during re-render**

If state or props change and React re-renders, your manual DOM change could disappear instantly because React rebuilds the DOM based on state.

- **You lose performance optimizations**

React is optimized to update only what's needed. Manual DOM changes bypass these optimizations, leading to extra reflows, repaints, and slower performance.

- **You bypass lifecycle methods and hooks**

React uses lifecycle phases (`useEffect`, `cleanup`, etc.) to control what happens when. Manual DOM updates skip these, which means React can't run `cleanup` or prepare effects properly.

- **Harder to debug and maintain**

Mixing manual DOM updates with React logic makes the codebase harder to read, test, and fix. Developers may not know what's causing the bug: React, or the DOM updates.

- **Breaks consistency in large applications**

In team projects, everyone expects the UI to follow React's patterns. If you manually change the DOM, others may be confused or cause conflicting behaviors.

- **Leads to unexpected side effects**

You might change an element's content or style manually — and React might change it back later. This causes flickering, layout shifts, or unpredictable behavior.

- **Manual updates aren't reusable or scalable**

React components are reusable because their output depends only on inputs (props/state). But manual DOM edits aren't reusable — they're hard-coded for specific situations.

Why Does Re-rendering Happen in React?

1. React Works Declaratively

We must know that React doesn't manually update the screen. Instead, we declare what the UI should look like based on data (state/props). So if the data changes, React must re-run the component to rebuild the UI.

2. React Needs to Reflect the Latest State or Props

When state or props change, the existing rendered content is now out of sync with the actual data. React re-renders to fix this mismatch and show the correct info.

3. Functional Components Are Just Functions

In React, components are functions. Re-rendering means React is simply calling the function again to get fresh JSX output. That's how it knows what to update.

4. React Ensures a Predictable UI

Re-rendering allows React to control the UI and ensure it's always predictable, up-to-date, and reliable — no manual DOM changes required.

Why Is Re-rendering Necessary?

1. To Keep the UI in Sync with Data

Without re-rendering, your UI would show stale or incorrect information. If a user updates something, the screen needs to reflect that instantly.

2. To Make User Interfaces Interactive

Every click, input, scroll, or change should update the screen. Re-rendering makes React apps feel alive and interactive by responding to changes.

3. To Maintain the “Single Source of Truth”

React treats state as the only “truth” of your app. The UI is just a reflection of that state. If the truth changes, the UI must update to match it — and that’s done via re-rendering.

4. To Avoid Manual DOM Work

Before React, we had to manually change the DOM (slow and error-prone). React handles this with automatic re-renders + Virtual DOM, so you don’t have to touch the DOM directly anymore.

What Causes Re-rendering in React?

1. Calling `setState` / `useState`

Whenever you update state (`setCount(1)`, `setText("hello")`), React re-renders that component and its children to reflect the new state.

2. Receiving New Props

If a parent component re-renders and sends new or changed props to a child, the child also re-renders to show the new data.

3. Changing Context Values

If you use `useContext()` and the value in the Context Provider changes, any component using that context will automatically re-render.

4. Force Updates or Ref Changes

Sometimes refs or external libraries cause changes (like dimensions, timers, etc.). In rare cases, you might force a re-render manually (e.g., by updating a state-like dummy value).

How does it happen?




1. Trigger: Data Changes (State, Props, or Context)

- The re-render process starts when data changes inside a React component.
 - You call `setState` or `useState` to update local state.
 - A parent component passes new props.
 - A shared context value is updated.
- React detects that something has changed, and flags the component for re-rendering.
- 🔍 Example: Clicking a button increases a counter → `setCount(count + 1)` triggers re-render.

2. React Re-runs the Component Function

- In functional components, React just calls your component function again.
- This re-run generates a fresh JSX output — the *new version* of what your UI should look like.
- It doesn't touch the screen yet. It's just preparing the "virtual design".
- Think of it like redrawing your app's blueprint based on the updated data.

3. React Builds a New Virtual DOM Tree

- The returned JSX from the re-run is used to build a new Virtual DOM — a lightweight JavaScript version of the UI.
- React does not update the real DOM yet.
- The Virtual DOM is faster, memory-efficient, and easy to compare with previous renders.
-  React now has two versions:
 -  Old Virtual DOM (from last render)
 -  New Virtual DOM (from current render)

4. React Compares Old vs New Virtual DOM (Diffing Phase)

- React uses a smart algorithm called the “diffing algorithm” to compare the old and new Virtual DOM trees.

- It looks for exactly what has changed — a new text? A new button? A deleted element?
- React creates a “patch list” — instructions for what needs to be updated in the real DOM.
- This step ensures performance optimization — only what’s different is touched.

5. React Updates the Real DOM Based on Differences

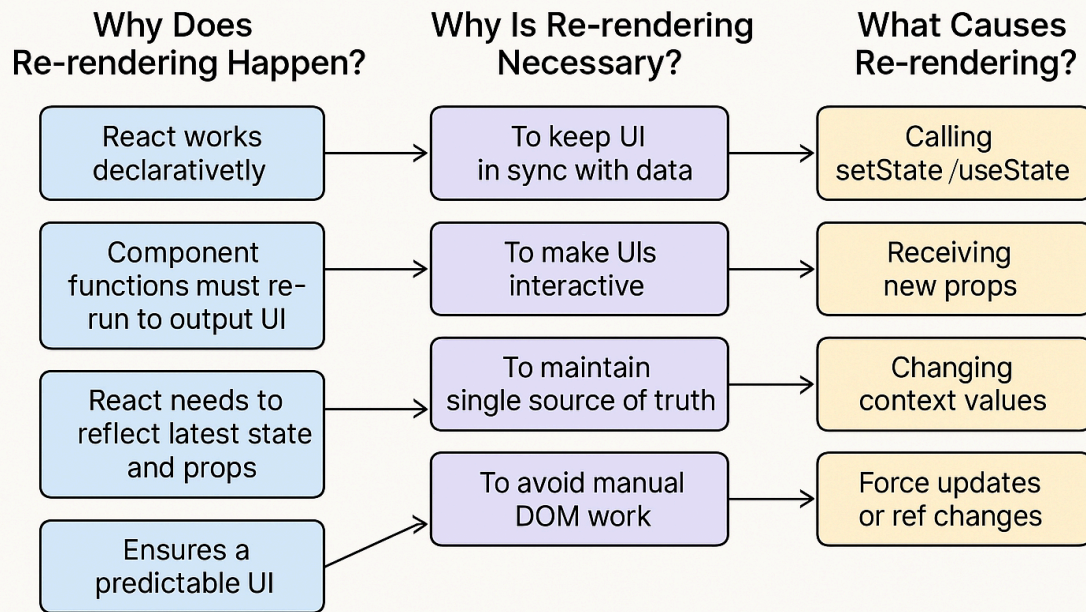
- Using the patch list, React updates only the necessary parts of the actual DOM:
 - Adds, removes, or updates elements.
 - Changes attributes or event listeners.
 - Modifies text content.
- The update is fast and efficient because React avoids touching the whole DOM.
- This is what makes React apps feel smooth, fast, and responsive.

6. React Lifecycle Continues (Effects and Cleanup)

- After the DOM is updated, React continues the lifecycle:
 - It runs `useEffect`, `useLayoutEffect`, and other hook effects if needed.
 - It also calls any cleanup functions from previous effects (`return () => { ... }`).
- This phase is where things like:

- API calls,
 - event subscriptions,
 - DOM measurements,
 - **and animation triggers happen.**
- And now, the component is “fresh” again — ready for the next interaction or update.

Understanding Re-rendering in React



Sources

- <https://react.dev/learn/render-and-commit>
- <https://www.joshwcomeau.com/react/why-react-re-renders/>
- <https://medium.com/simform-engineering/react-re-rendering-exploring-what-why-and-how-d180d5305892>
- https://www.reddit.com/r/reactjs/comments/13t78sp/need_help_in_understanding_react_renders/?rdt=51620
- <https://medium.com/%40islamghany3/demystifying-re-rendering-diffing-and-reconciliation-in-react-3a821ebc36c9>
- <https://medium.com/@itsanuragjoshi/mastering-react-understanding-real-dom-vs-virtual-dom-and-the-dom-update-process-78a233454ff8>