Adding Interactivity With Hydration

6 min

Previously, we discussed server-side rendering and noted that while our web app is visible, it is not interactive. Let's get deeper into **hydration** and refine our mental model of how Next.js operates.

If the process of SSR is the first layer of paint, the second layer is hydration. After the server-sent HTML is loaded on the client's browser, the JavaScript bundle accompanying the HTML starts executing. This JavaScript includes the React code, which then "hydrates" the static HTML. Hydration involves attaching event handlers and linking the React components to their HTML counterparts. During this process, React also performs **reconciliation**, comparing the result from rendering components on the client side with the result from rendering on the server, ensuring they are in sync.

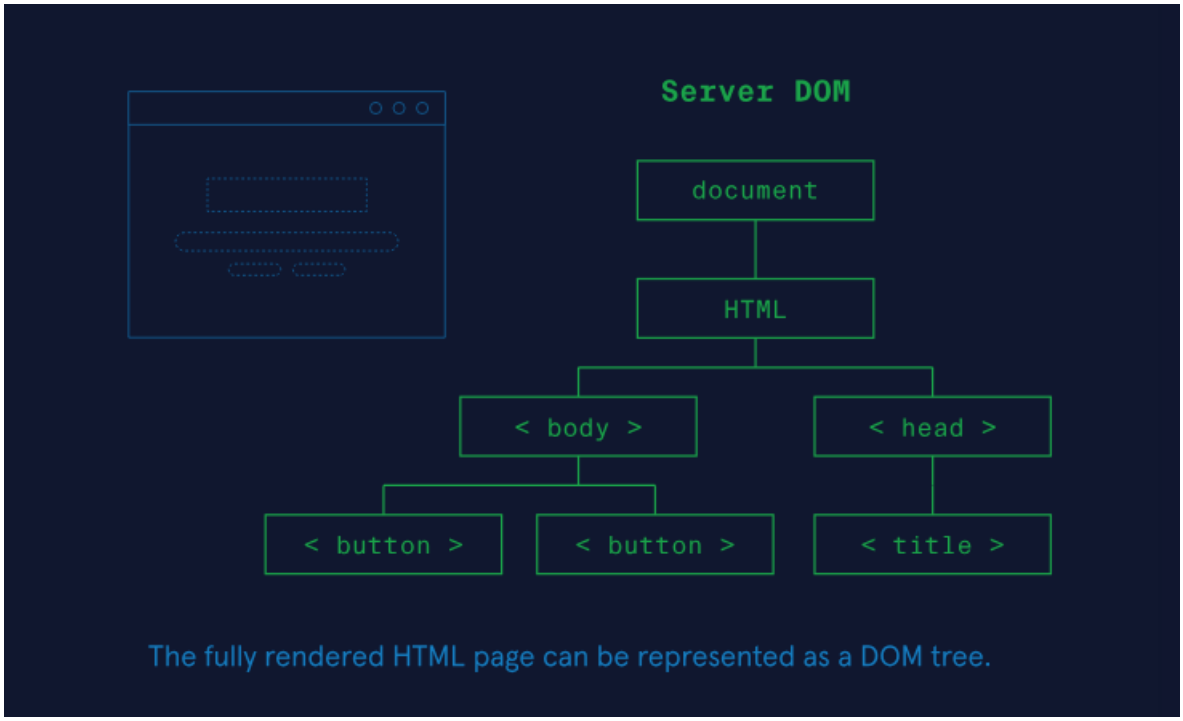
Once hydration is complete, the webpage becomes fully interactive. The interactive elements like buttons and forms can now respond to user inputs. From this point onwards, any updates to the page, such as user interactions or data fetching, lead to a re-render of the affected components. This re-rendering is handled entirely on the client side, and the components are updated to reflect new states or props. Consider it a touch-up of the paint layers!

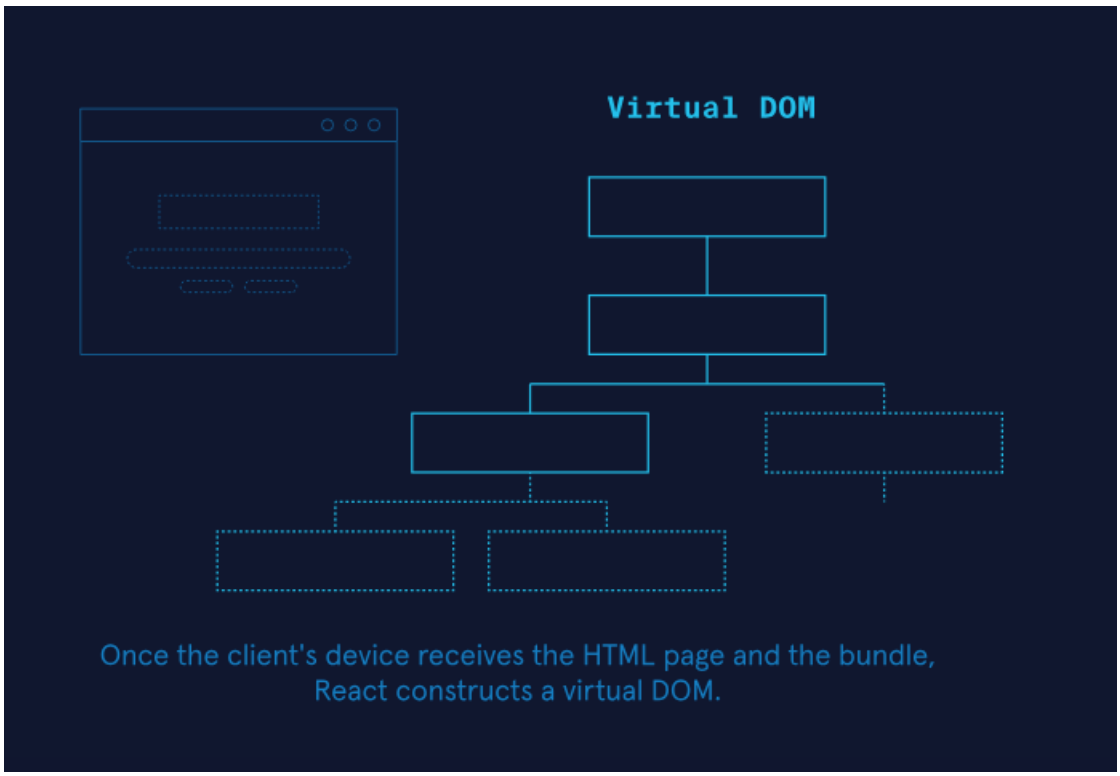
With hydration, we can see that rendering with Next.js is not confined to one side alone. A Next.js application typically represents a blend of different rendering techniques, leveraging server- and client-side strengths to deliver an optimal user experience.

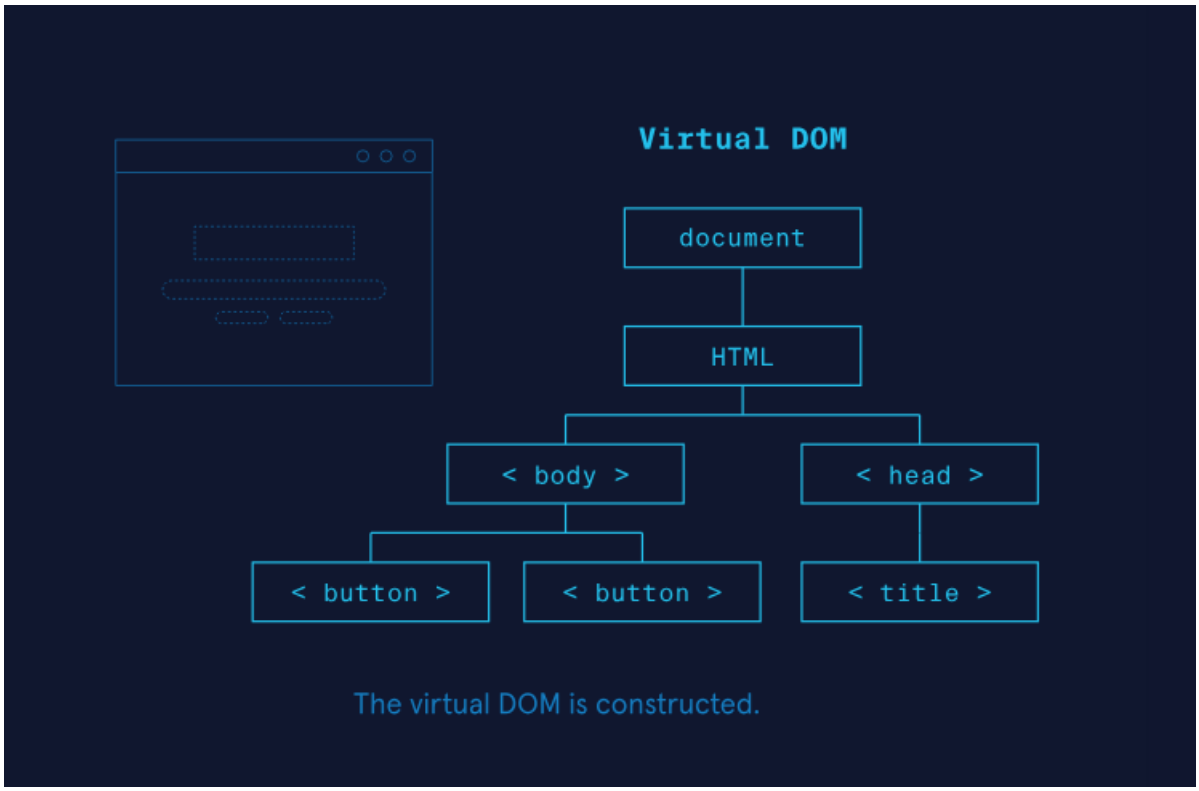
Instructions

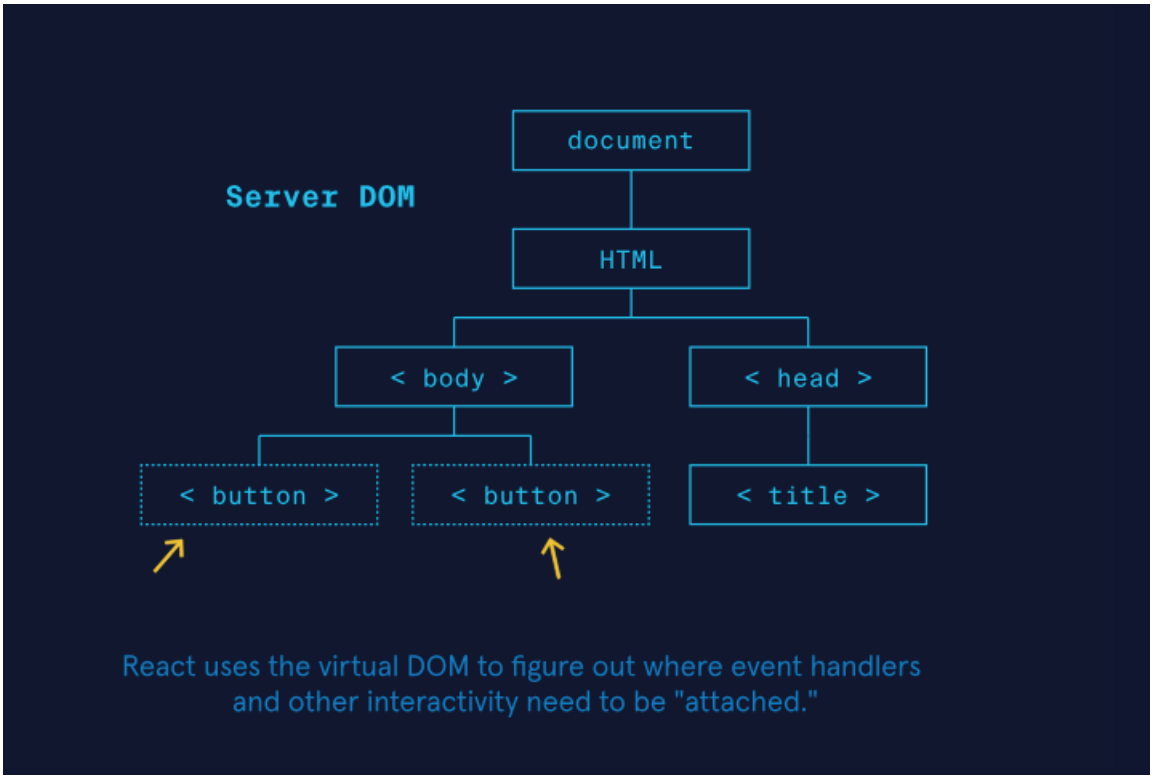
Take a look at the diagram provided in the web browser section. Observe how hydration injects interactivity into the web app.


From the Server
Website
JS
data
A client device receives a fully rendered HTML page. It also receives a bundle of JavaScript files, and any extra data needed to make the page is sent.

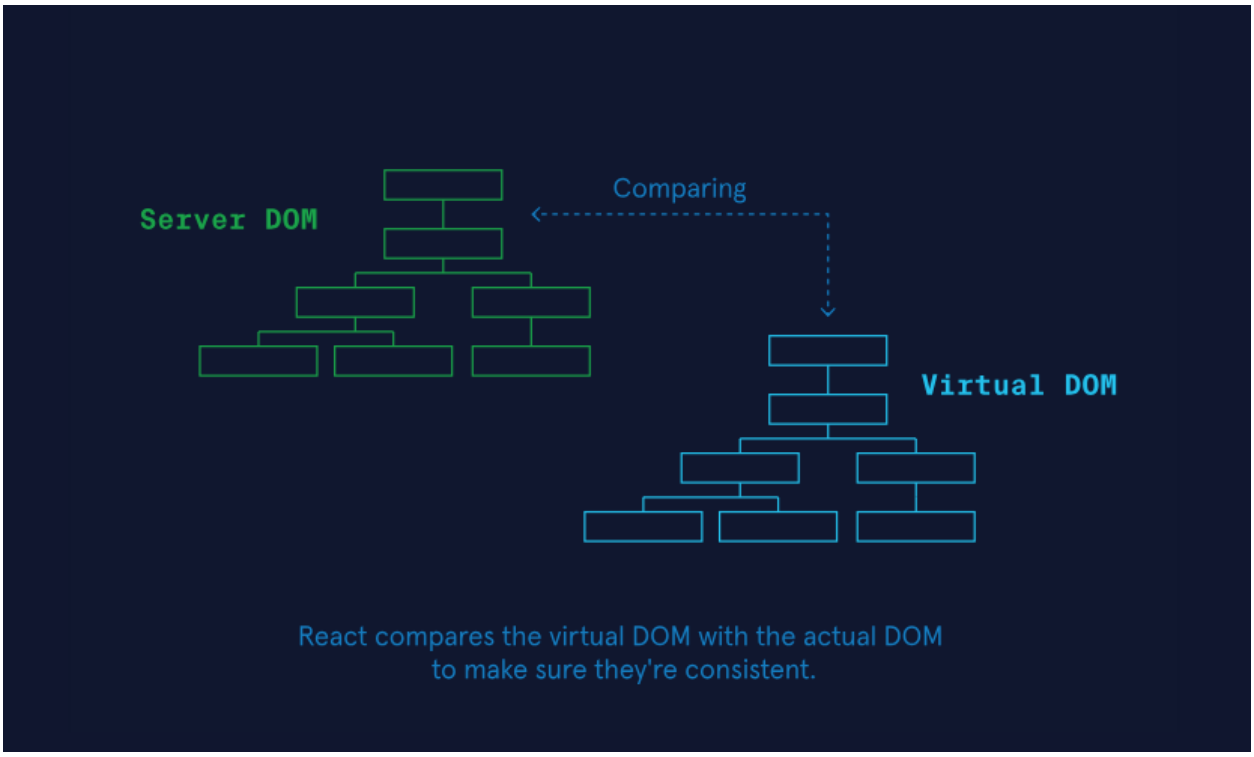

Server DOM
document
HTML
< body >
< head >
< button >
< button >
< title >
The fully rendered HTML page can be represented as a DOM tree.


Virtual DOM
Once the client's device receives the HTML page and the bundle,
React constructs a virtual DOM.

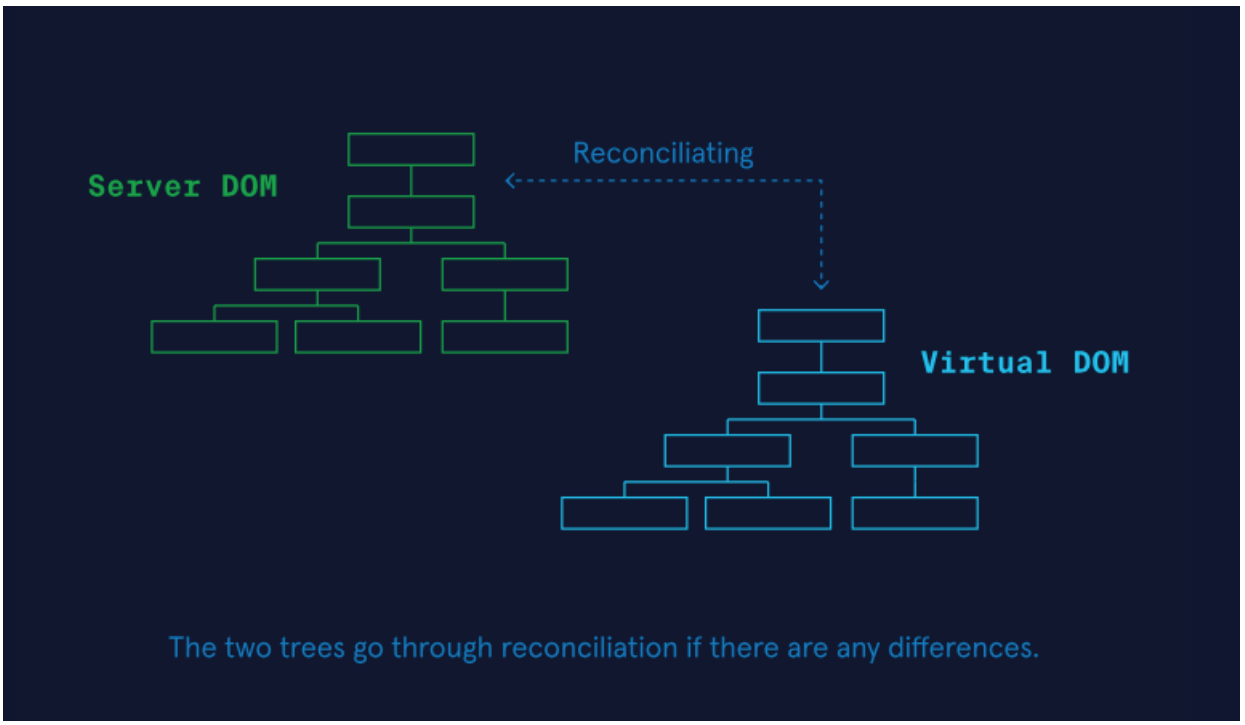

Virtual DOM
document
HTML
< body >
< head >
< button >
< button >
< title >
The virtual DOM is constructed.



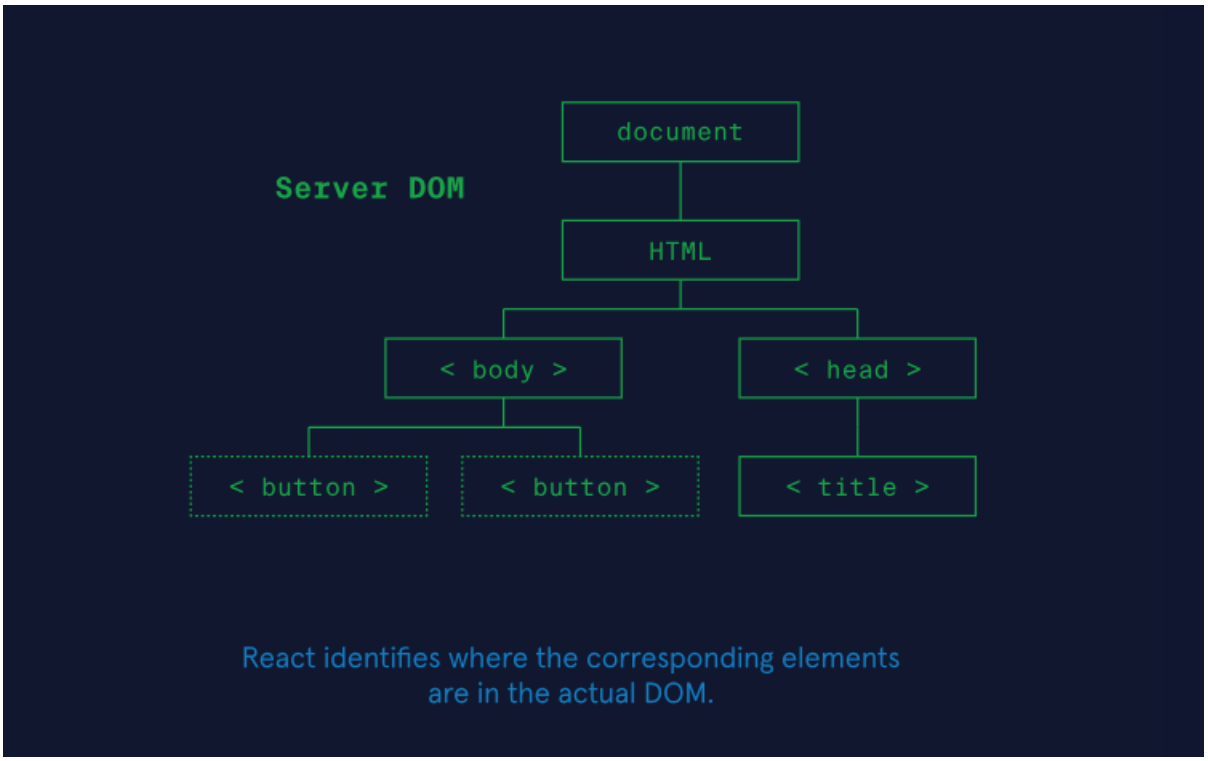
React uses the virtual DOM to figure out where event handlers and other interactivity need to be "attached."



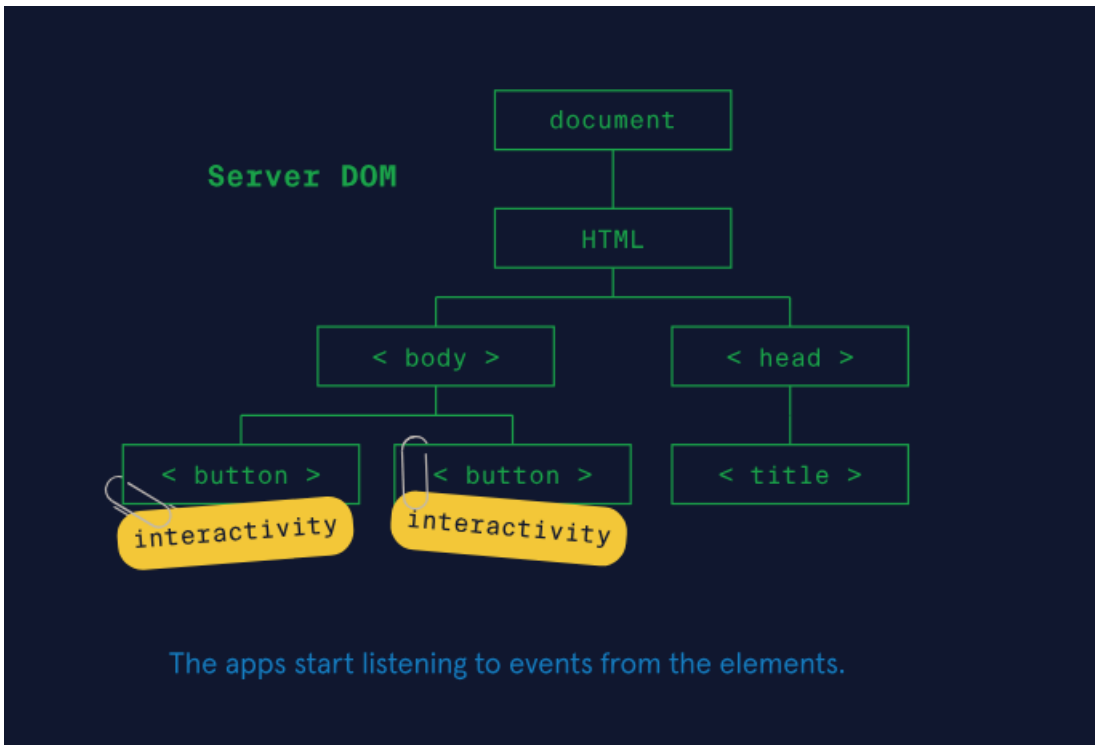
React compares the virtual DOM with the actual DOM to make sure they're consistent.

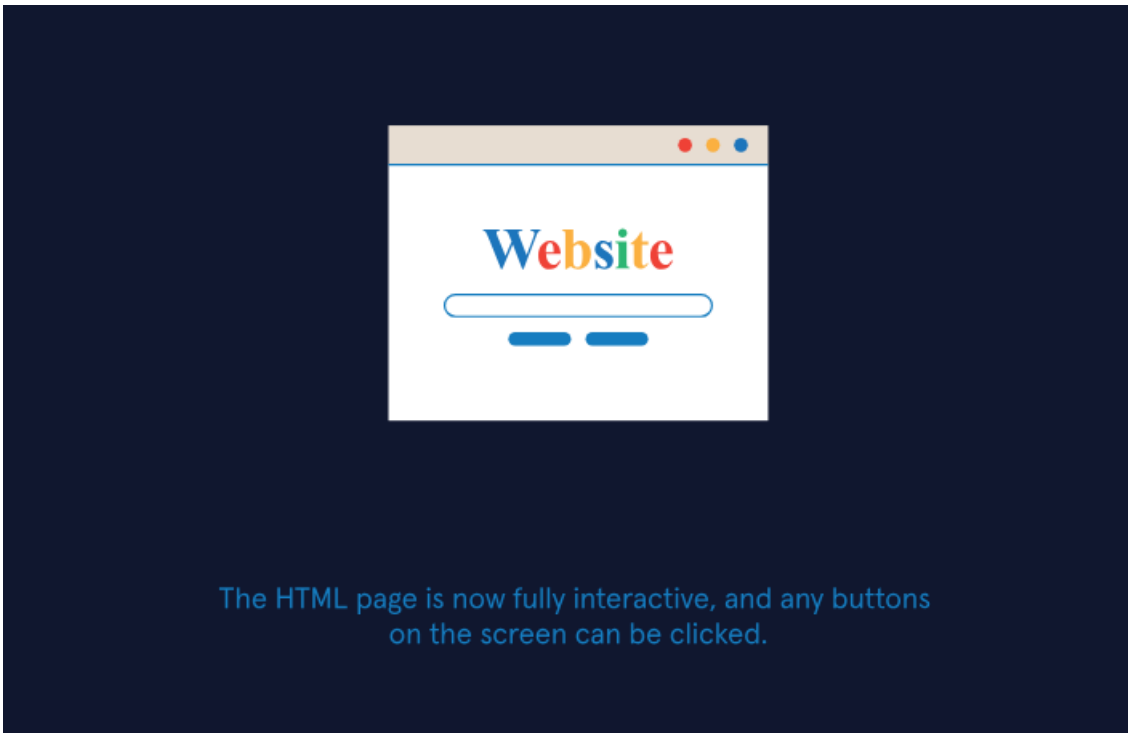


The two trees go through reconciliation if there are any differences.



React identifies where the corresponding elements
are in the actual DOM.


Server DOM
document
HTML
< body >
< head >
< button >
< button >
< title >
interactivity
interactivity
The apps start listening to events from the elements.


Website
The HTML page is now fully interactive, and any buttons
on the screen can be clicked.