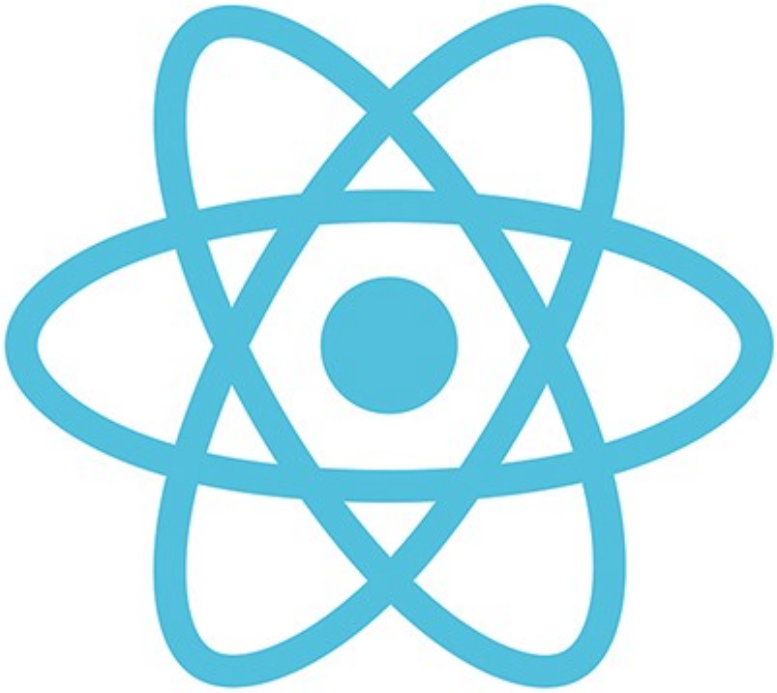
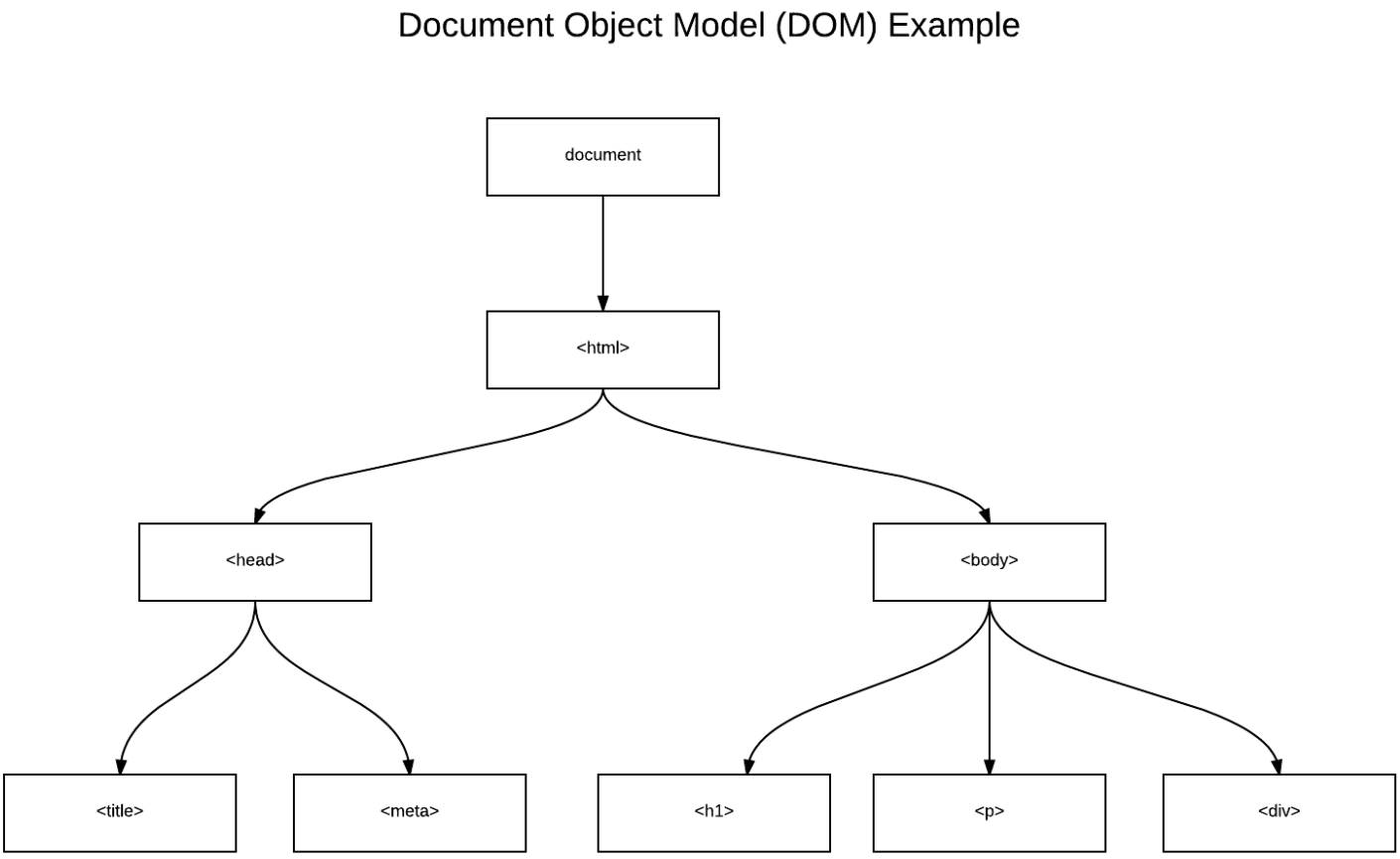
Dace
May 5, 2017 · 4 min read

# Understanding React's Virtual DOM vs. the real DOM



## What is the DOM?

Before we can best understand React and the Virtual DOM, we should first understand the DOM (Document Object Model). The DOM is an abstraction of a page's HTML structure. It takes HTML elements and wraps them in an object with a tree-structure—maintaining the parent/child relationships of those nested HTML elements. This provides an API that allows us to traverse nodes (HTML elements) and manipulate them in a number of ways—such as adding nodes, removing nodes, editing a node's content, etc.




While it was often the case to use jQuery for DOM manipulation, you can actually just use vanilla Javascript with methods provided by the DOM.

## DOM inefficiency

We use Javascript to manipulate the DOM. However, manipulation tends to be quite inefficient as the DOM was originally intended for static UIs—pages rendered by the server that don't require dynamic updates. When the DOM updates, it has to update the node as well as re-paint the page with it's corresponding CSS and layout.

With the growing popularity of Single Page Apps (SPAs), components on our page are increasingly more responsible for listening for updates and re-rendering those updates to the UI. It's not uncommon to come across pages that display thousands of dynamically generated nodes that also must continue to listen for future updates. This is where things can get quite expensive.

## Knowing when to update

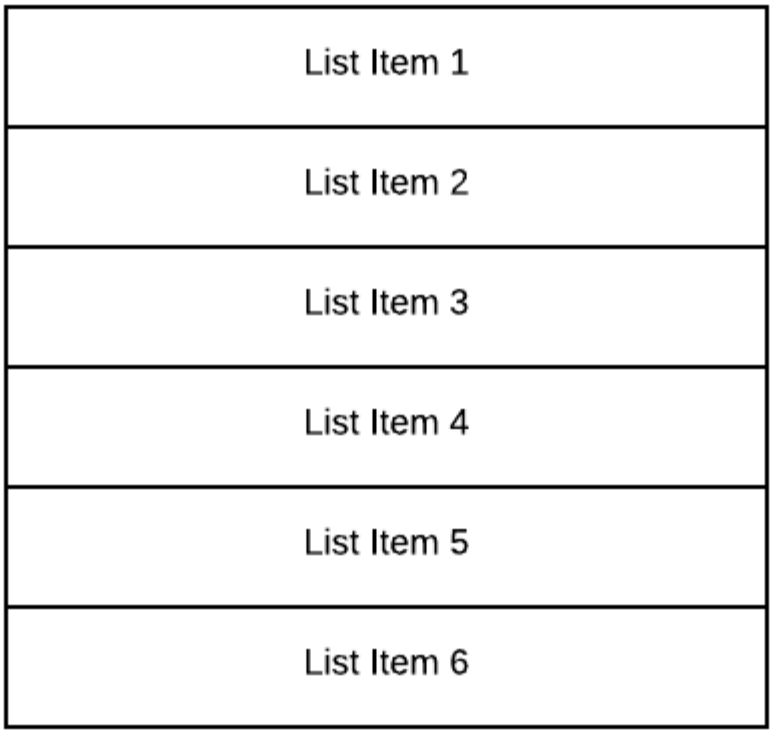
There are a couple of ways in which components can tell when a data update occurs and whether or not it needs to re-render to the UI:

- **Dirty Checking (slow)**—checks through all node's data at a regular interval to see if there have been any changes. This is inefficient because it requires traversing every single node recursively to make sure it's data isn't "dirty" (out of date).
- **Observable (fast)**— components are responsible for listening to when an update takes place. Since the data is saved on the state, components can simply listen to events on the state and if there is an update, it can re-render to the UI.

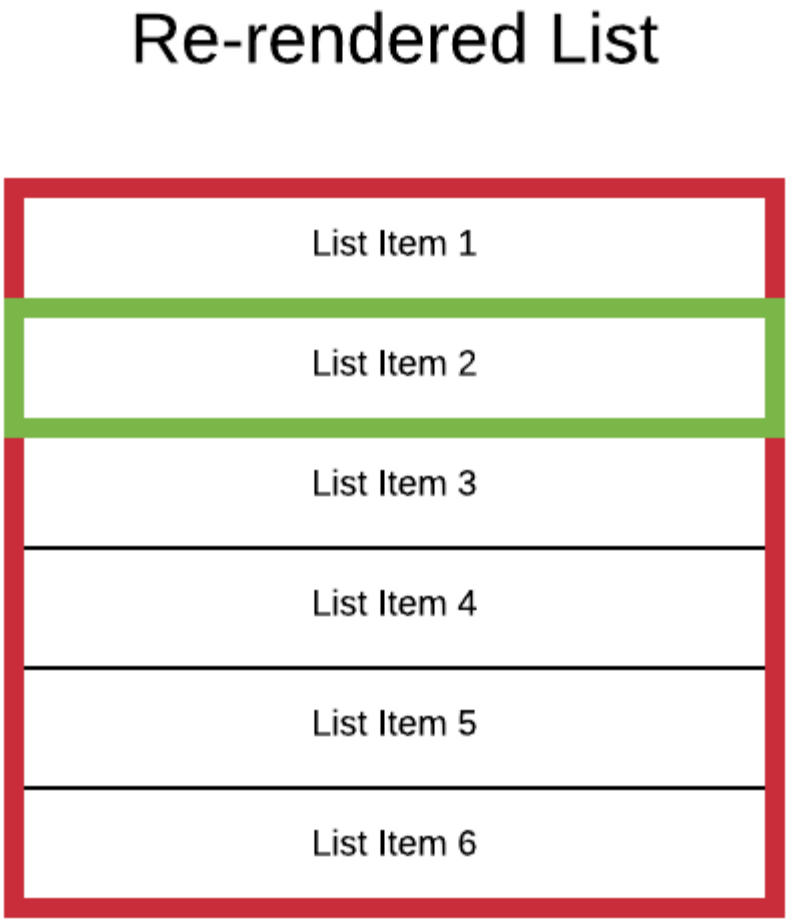
React uses the later method.

## How the DOM renders

Let's look at an example of a list of items. Within this list we have a number of items and any one of these items can receive updates.




If one of these list items updates, then the DOM re-renders the entire list. This is where the DOM's inefficiency stems from:




While this is a simplified example of one list containing a handful of items, this becomes incredibly inefficient if you have a SPA with hundreds or thousands of components that require re-rendering when updates get passed down. Ideally, we'd like to only re-render items that receive updates, leaving the rest of the items as-is.

React's use of the Virtual DOM helps to reduce this inefficiency.

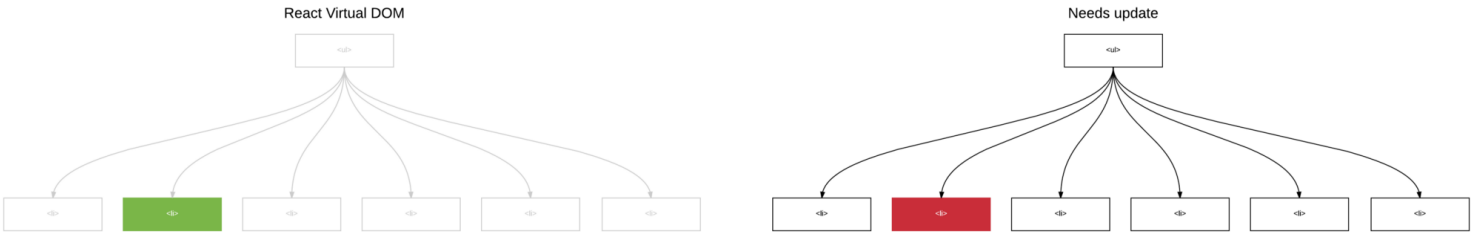
## Introducing: the Virtual DOM

The Virtual DOM is a light-weight abstraction of the DOM. You can think of it as a copy of the DOM, that can be updated without affecting the actual DOM. It has all the same properties as the real DOM object, but doesn't have the ability to write to the screen like the real DOM. The virtual DOM gains it's speed and efficiency from the fact that it's lightweight. In fact, a new virtual DOM is created after every re-render.

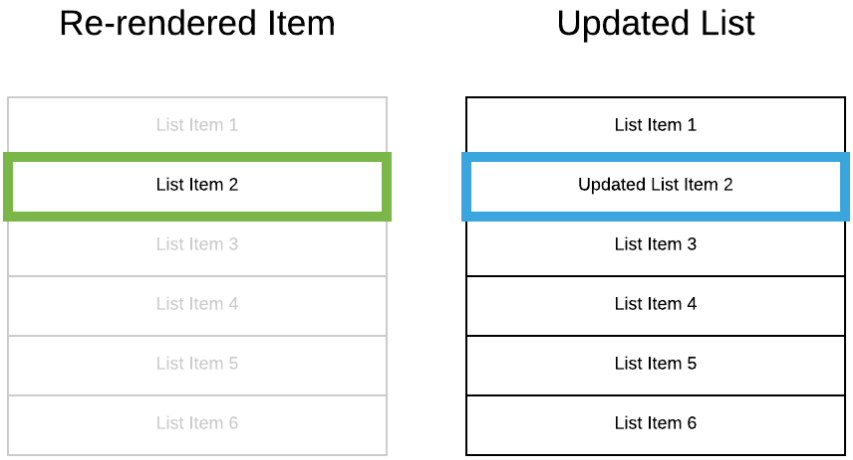
*It's important to note that React did not introduce the virtual DOM (there are a number of libraries who also make use of it). In fact, as React has been moving to a number of non-web platforms (React-Native, React-VR, etc.), the portion from previous React versions that dealt with interacting with the DOM has been extracted. It's now provided through a separate package—React-DOM that you must install/include if you wish to use React on the web.*

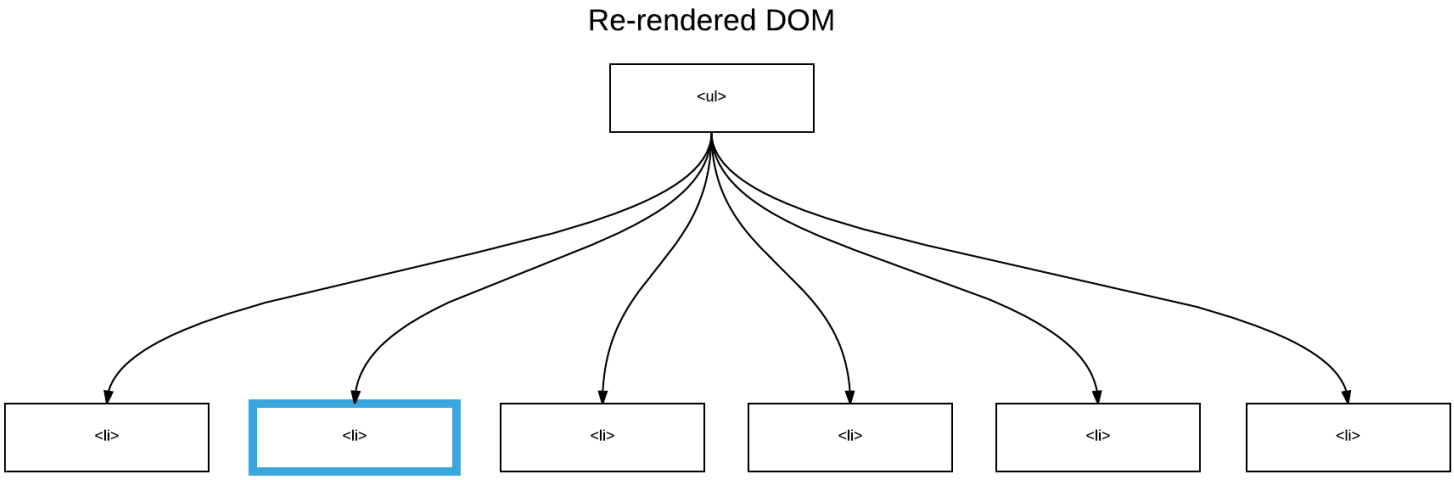
## Snapshots & Diffing

React takes a virtual DOM snapshot (record of the DOM state) right before applying any updates. It then uses this snapshot to compare against an updated virtual DOM before making changes.




When updates are supplied to the Virtual DOM, React uses a process called reconciliation—using a "diffing" algorithm that compares/contrasts changes in order to know what updates have taken place. React then only updates those elements that have changed, leaving alone those that have not.






I hope this helps to shed some light on the advantages of efficiency with React's use of a virtual DOM.