

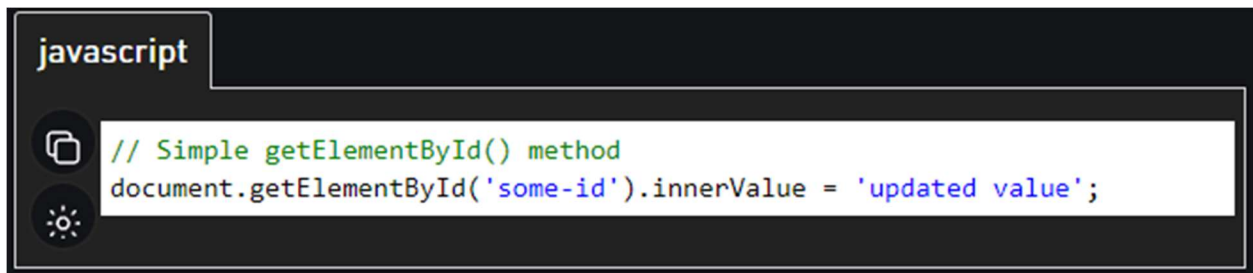
React Virtual DOM vs Real DOM

What is DOM?

Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document.

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 several ways - such as adding nodes, removing nodes, editing a node's content, etc.

Accessing DOM example



```
javascript
// Simple getElementById() method
document.getElementById('some-id').innerHTML = 'updated value';
```

When writing the above code in the console or in the JavaScript file, these things happen:

- The browser parses the HTML to find the node with this id.
- It removes the child element of this specific element.
- Updates the element(DOM) with the 'updated value'.
- Recalculates the CSS for the parent and child nodes.
- Update the layout.
- Finally, traverse the tree and paint it on the screen(browser) display.

So as we know now that updating the DOM not only involves changing the content, it has a lot more attached to it. Also recalculating the CSS and changing the layouts involves complex

algorithms, and they do affect the performance. So React has a different approach to dealing with this, as it makes use of something known as **Virtual DOM**.

DOM inefficiency

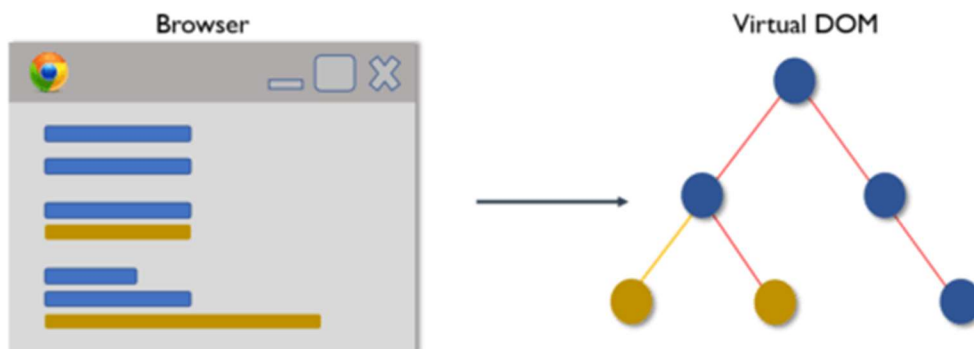
DOM was originally intended for static UIs — pages rendered by the server that don't require dynamic updates. When the DOM updates, it must update the node as well as re-paint the page with its corresponding CSS and layout.

- Dirty Checking (slow) — Checks through all node's data at a regular interval to see if there have been any changes. This was used in AngularJS 1x
- Observable (fast) — Components are responsible for listening to when an update takes place. React uses it.

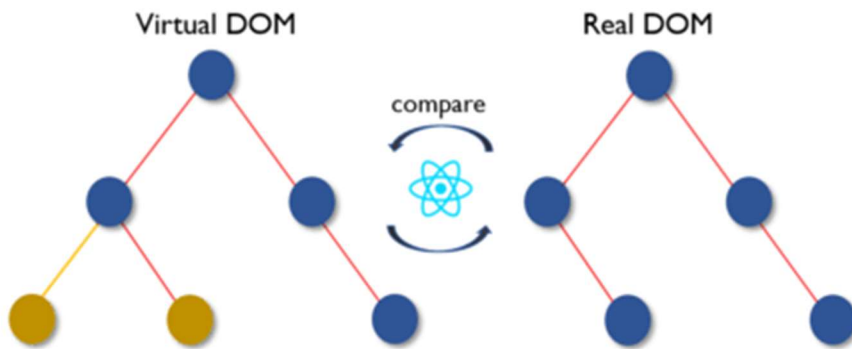
Virtual DOM

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.

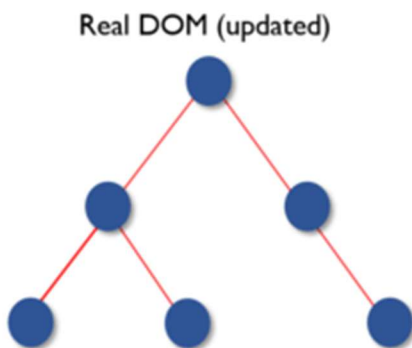
How Virtual DOM actually make things faster: When anything new is added to the application, a virtual DOM is created and it is represented as a tree.



Each element in the application is a node in this tree. So, whenever there is a change in the state of any element, a new Virtual DOM tree is created. This new Virtual DOM tree is then compared with the previous Virtual DOM tree and make a note of the changes.



After this, it finds the best possible ways to make these changes to the real DOM. Now only the updated elements will get rendered on the page again.



How Virtual DOM helps React: In react, everything is treated as a component be it a functional component or class component. A component can contain a state. Whenever the state of any component is changed react updates its Virtual DOM tree. React maintains two Virtual DOM at each time, one contains the updated Virtual DOM and one which is just the pre-update version of this updated Virtual DOM. Now it compares the pre-update version with the updated Virtual DOM and figures out what exactly has changed in the DOM like which components have been changed. This process of comparing the current Virtual DOM tree with the previous one is known as '**diffing**'. Once React finds out what exactly has changed then it updates those objects only, on real DOM. React uses something called batch updates to update the real DOM. It just means that the changes to the real DOM are sent in batches instead of sending any update for a single change in the state of a component. We have seen that the re-

rendering of the UI is the most expensive part and React manages to do this most efficiently by ensuring that the Real DOM receives batch updates to re-render the UI. This entire process of transforming changes to the real DOM is called Reconciliation. So the Virtual DOM significantly improves the performance

The object, as seen below, is the virtual DOM. It represents the HTML user interface.

```
▼ Object { "$$typeof": Symbol("react.element"), type: Symbol("react.fragment"), key: null, ref: null, props: {},  
  _owner: null, _store: {_, _ }  
    "$$typeof": Symbol("react.element")  
    _owner: null  
    _self: null  
    _source: null  
    ▶ _store: Object { _ }  
    key: null  
  ▶ props: Object { children: (3) [_, {_, {_} ]  
    ▼ children: Array(3) [ {_, {_, {_} ]  
      ▶ 0: Object { "$$typeof": Symbol("react.element"), type: "h3", key: null, _ }  
      ▶ 1: Object { "$$typeof": Symbol("react.element"), type: "form", key: null, _ }  
      ▶ 2: Object { "$$typeof": Symbol("react.element"), type: "span", key: null, _ }  
      length: 3  
      ▶ <prototype>: Array []  
    ▶ <prototype>: Object { _ }  
    ref: null  
    type: Symbol("react.fragment")
```

Methods of DOM:

- write("string"): Writes in document area
- writeln("string"): Writes in new line in document area
- getElementById("id"): Returns the element with passed ID
- getElementByName(" name"): Returns the element with passed name
- getElementByTagName("tagname"): Returns the element with the passed tag name
- getElementByClassName("classname"): Returns the element with passed class name

Example:

Let's take a simple example of an array

```
let languages = [cpp, java, python]
```

Now we want to replace python with javascript. For that, we need to create a new array.

```
let languages = [cpp, java, javascript]
```

Instead of this, we can just traverse to languages[2] and update only the element.

Instead of redoing the whole thing, we just changed the element which we needed to update.

The same thing is done by **Virtual DOM**

Difference between DOM and Virtual DOM:

| DOM | Virtual DOM |
|--|--|
| DOM manipulation is very expensive | DOM manipulation is very easy |
| There is too much memory wastage | No memory wastage |
| It updates Slow | It updates fast |
| It can directly update HTML | It can't update HTML directly |
| Creates a new DOM if the element updates | Update the JSX if the element update |
| It allows us to directly target any specific node (HTML element) | It can produce about 200,000 Virtual DOM Nodes / Second. |
| It represents the UI of your application | It is only a virtual representation of the DOM |