Reconciliation in React

[(https://github.com/learn-co-curriculum/react-hooks-reconciliation)](https://github.com/learn-co-curriculum/react-hooks-reconciliation) [(https://github.com/learn-co-curriculum/react-hooks-reconciliation/issues/new)](https://github.com/learn-co-curriculum/react-hooks-reconciliation/issues/new)

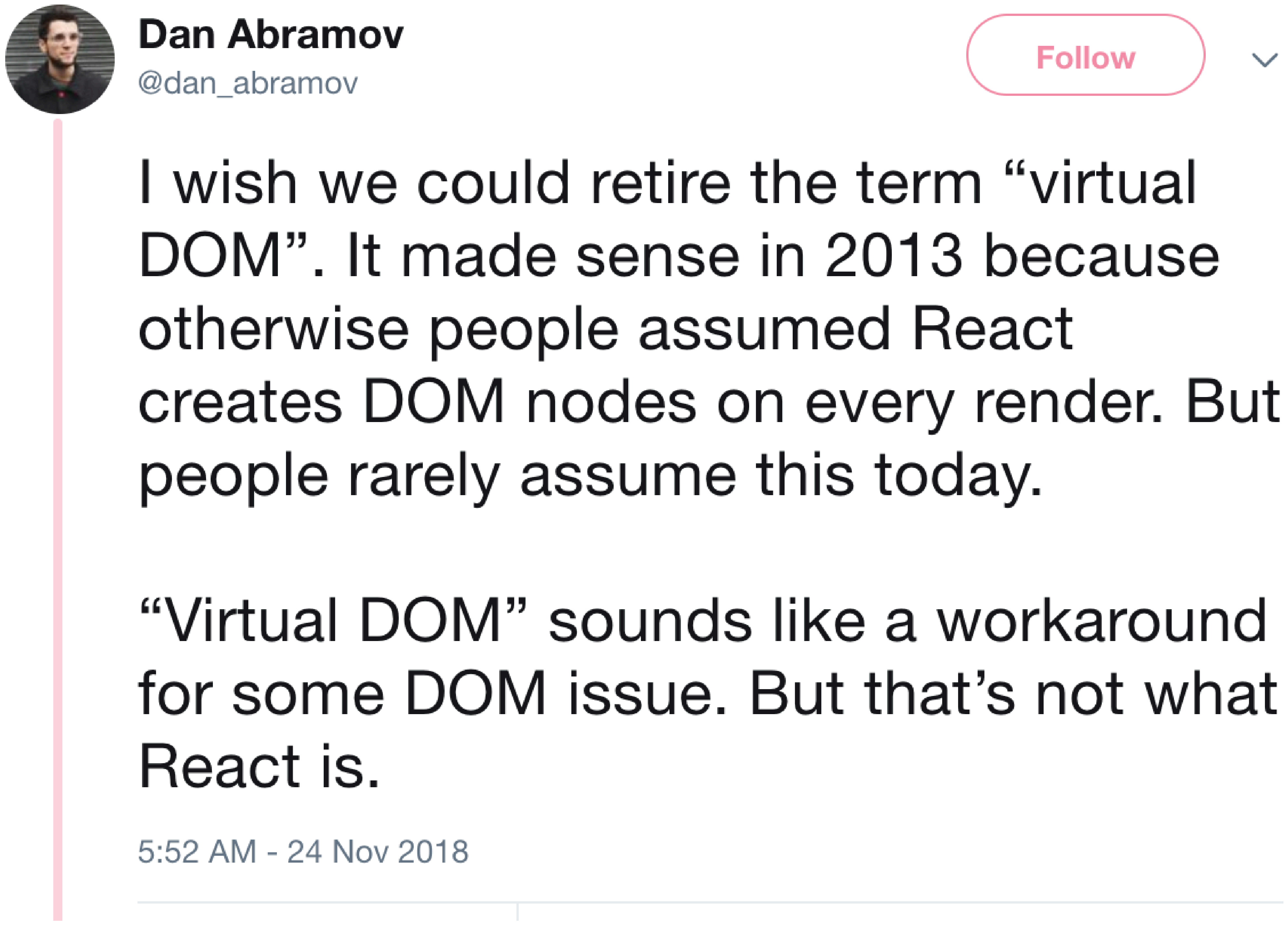
Learning Goals

- Understand how React handles DOM updates in a performant manner

Introduction

Earlier in the history of React, the term "Virtual DOM" was used to explain how React was able to perform better than the traditional DOM.

The term 'Virtual DOM' fails to really explain what is happening and may lead to a misunderstanding of what is going on behind the scenes when React renders.

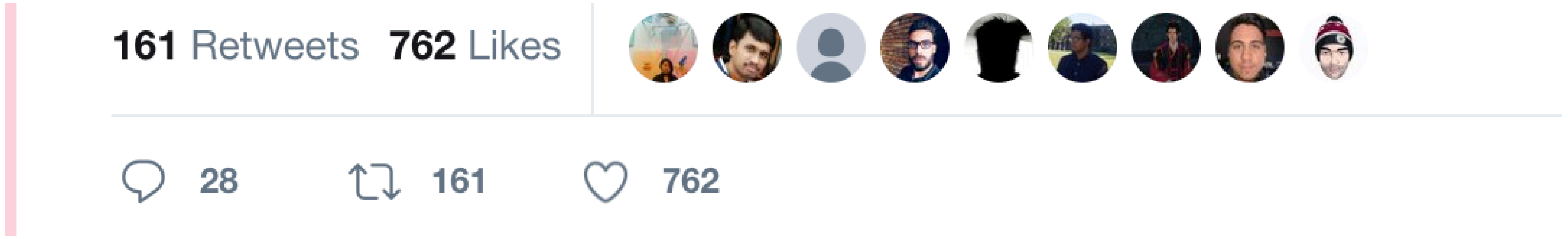


Dan Abramov
@dan_abramov

I wish we could retire the term “virtual DOM”. It made sense in 2013 because otherwise people assumed React creates DOM nodes on every render. But people rarely assume this today.

“Virtual DOM” sounds like a workaround for some DOM issue. But that’s not what React is.

5:52 AM - 24 Nov 2018



In this lesson, we're going to briefly review how React handles updates to the screen. This process is known as [Reconciliation (https://reactjs.org/docs/reconciliation.html)](https://reactjs.org/docs/reconciliation.html).

Updating the DOM

By now, you should already know what the DOM is: a programmatic representation of the document we see in the browser. In JavaScript applications, DOM elements can be added and changed with code. It's possible to build highly complex websites with hundreds or thousands of DOM elements using plain JavaScript. Maybe more importantly, through the DOM, JavaScript allows us to build highly interactive webpages that update dynamically without refreshing. This can come with some challenges, though.

When the DOM updates, the browser recalculates CSS, lays out the DOM tree and 'repaints' the display. This typically happens so fast you barely notice. However, on a highly interactive website, or on a website where the JavaScript is updating the DOM excessively, the process of recalculating and repainting the display can result in noticeably poor performance.

Any time you want your website or app to update without refreshing, you'll need to update the DOM; there is no avoiding it. However, React has some neat tricks for being smart about these updates.

Reconciliation, Briefly

In React, we know that we write components that return JSX elements. These JSX elements represent DOM elements, and when rendered, become those elements on a webpage.

During the initial render, React *also* uses these elements to build a 'tree' that *represents* what the DOM currently looks like, referred to as the **current** tree.

When updates are made that would cause a re-render in React, a *second* tree, the **workInProgress** tree is created, representing what the DOM *will* look like. When all updates are processed, the **workInProgress** tree is used to update the DOM and the **current** tree is updated to reflect the new updates.

This is a key part of React's performance optimization — React uses these trees as an intermediate step between updates within components, such as a change of state, and updates to the DOM. This approach enables React to use two techniques that improve performance: grouped updates and diffing changes.

Grouped Updates

Because React creates a **workInProgress** tree, updates can be grouped together. By waiting until all updates are processed before committing the **workInProgress** tree to the DOM, excessive repaints are avoided.

Say, for instance, you have an app with many components, each colored a shade of blue, and a button that, when pressed, turns all those components to red. When that button is pressed, React will put together a tree containing all the components along with their updated properties, *THEN* commit all the changes to the DOM at once. This only requires one repaint. Without this design, we could end up with code that updates the DOM for each individual part of the app, one repaint for each part.

Diffing Changes

In addition to grouping updates to the DOM, React can apply a diffing algorithm to identify changes between what the current DOM looks like (the **current** tree) and what it will look like (the **workInProgress** tree), and quickly see what specific pieces of DOM *need* to be updated and how. This reduces the number of DOM changes that need to be made and lets React be particular in its updates, improving performance.

In plain JavaScript — which you'll remember is what React is creating for us under the hood — some DOM changes are better than others in terms of performance. For example, say you want to add something inside a `ul` in your DOM. Using `innerHTML` will work:

```
ul.innerHTML += "<li>A final list item</li>";
```

But this *rebuilds* the entire DOM inside `div` . On the other hand, using `append` would *not* cause a rebuild:

```
let li = document.createElement("li");
li.textContent = "A final list item";
ul.append(li);
```

React compares the **current** and **workInProgress** trees, identifies which pieces of the DOM need updating, and then uses the most efficient means for making each update.

A more detailed explanation of the steps of this diffing process can be found in **React's Reconciliation documentation (https://reactjs.org/docs/reconciliation.html)** .

Conclusion

There are some **misconceptions (https://www.quora.com/Why-is-Reacts-virtual-DOM-so-much-faster-than-the-real-DOM) floating (https://news.ycombinator.com/item?id=9155564) around (https://www.reddit.com/r/javascript/comments/6115ay/why_do_developers_think_the_dom_is_slow/)** regarding the DOM being slow, often related to how frameworks like React can improve performance. While DOM manipulation *itself* isn't 'slow,' repainting what is displayed in the browser can be.

React can be very smart about handling DOM updates, which improves performance. Primarily, it does this in two ways: grouping DOM updates to prevent excessive repaints and being selective about what specifically needs to update and how.

Read a more in-depth dive on these concepts **here (https://medium.com/react-in-depth/inside-fiber-in-depth-overview-of-the-new-reconciliation-algorithm-in-react-e1c04700ef6e)** .

Resources

- **Reconciliation (https://reactjs.org/docs/reconciliation.html)**
- **Inside Fiber: in-depth overview of the new reconciliation algorithm in React (https://medium.com/react-in-depth/inside-fiber-in-depth-overview-of-the-new-reconciliation-algorithm-in-react-e1c04700ef6e)**