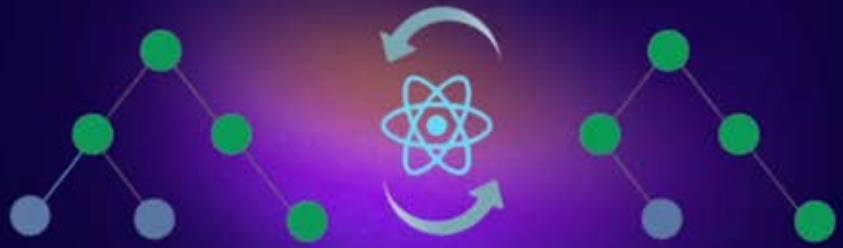
# Virtual DOM VS Real DOM



PROGRAMMING HERO

If Document Object Model Is So Cool, Then Why Go With Virtual DOM?

# Consider a situation -

- Decorate the room with 100 balloons & you have done.
- Now I have a condition that the color of **each balloon** should be red. So you remove all the balloons and now decorate the room with red balloons.
- Now I have set a new condition that **change balloon no 90** with a blue-colored balloon. Now you remove all balloons and bring 99 red balloons and 1 blue balloon and paint accordingly.
- I am again asking you to **change balloon no 80** to green. You remove all the balloons. Go market bring 98 red, 1 blue, 1 green. And decorate the room accordingly.

WOW, will you work this way? No. right? but your Document Object Model does. So, how to make DOM intelligent? Now, VIRTUAL DOM comes to rescue us.

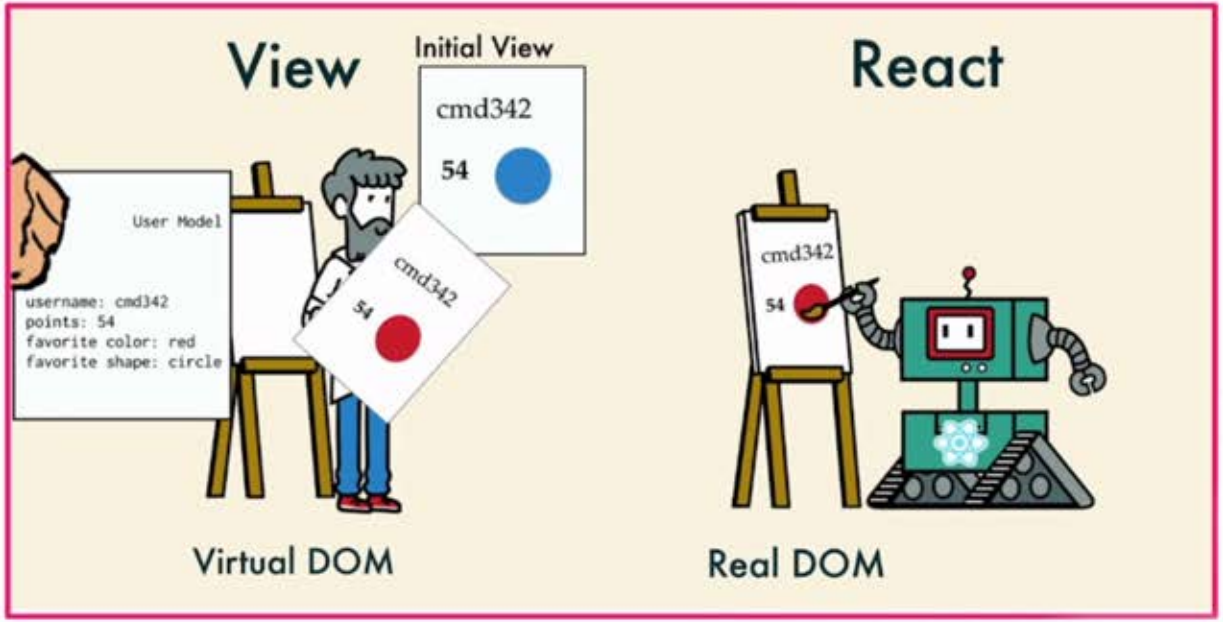
# What Is The Virtual DOM In React JS?

# What is **Virtual DOM?**

Virtual DOM –

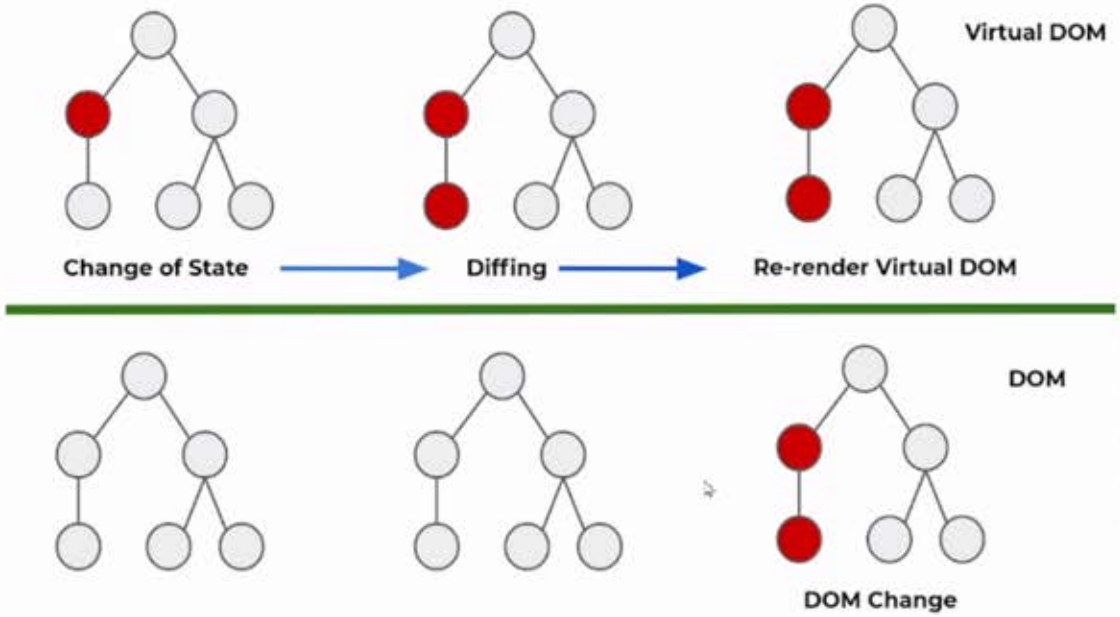
- Lightweight copy of a DOM object.
- Has properties the same as real DOM object.
- Make changes in the DOM with the help of the Diff Algorithm.
- It's like: instead of moving actual rooms in a house, you edit the blueprint.

# Virtual DOM vs Real DOM

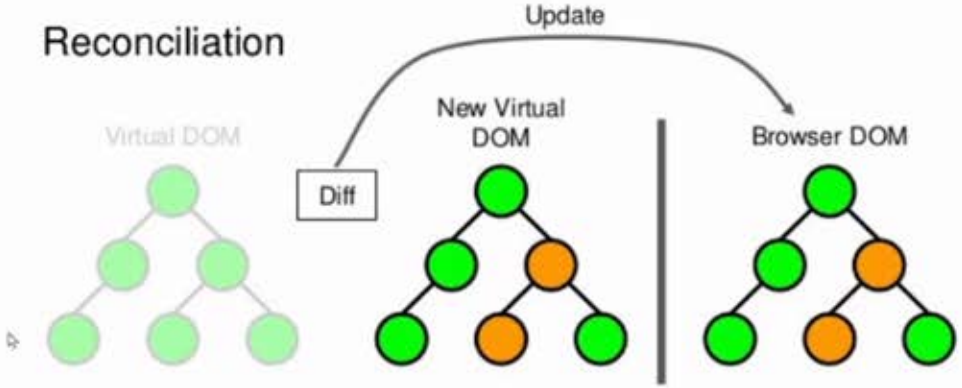


## Now, How Does Virtual DOM Work?

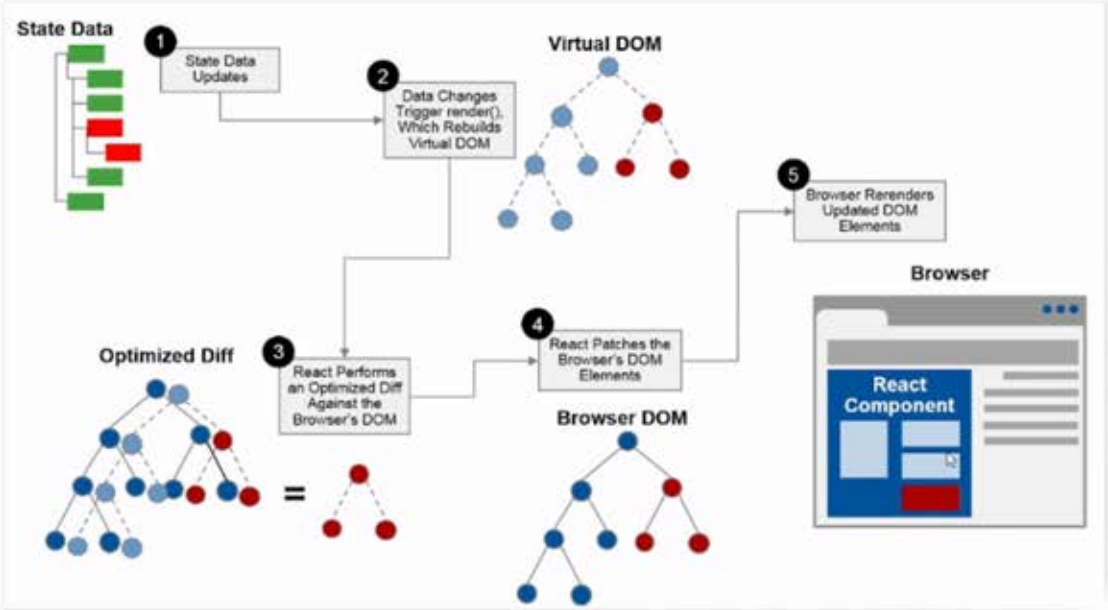
# How does virtual dom work?



# How does **virtual dom work?**



# How does virtual dom work?



# Virtual DOM vs Real DOM

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

README.md Update README.md 7 years ago

README.md

# React Fiber Architecture

## Introduction

React Fiber is an ongoing reimplementation of React's core algorithm. It is the culmination of over two years of research by the React team.

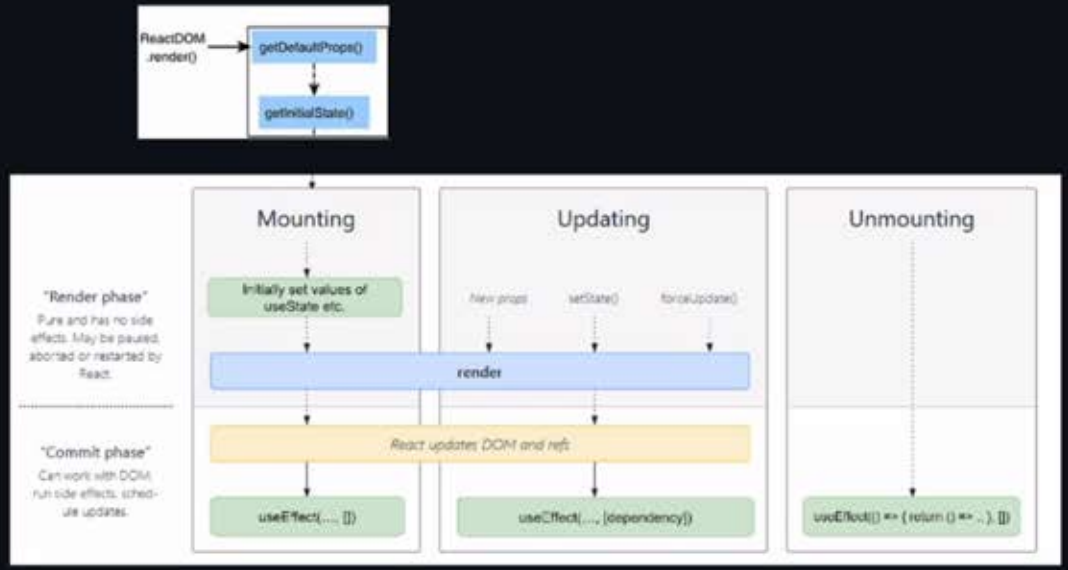
The goal of React Fiber is to increase its suitability for areas like animation, layout, and gestures. Its headline feature is **incremental rendering**: the ability to split rendering work into chunks and spread it out over multiple frames.

Other key features include the ability to pause, abort, or reuse work as new updates come in; the ability to assign priority to different types of updates; and new concurrency primitives.

## About this document

Fiber introduces several novel concepts that are difficult to grok solely by looking at code.

README.md




## React functional component lifecycle methods

```
1. Mounting
    - useState and useEffect are called during the mounting phase.
```

# Lifecycle of Reactive Effects

Effects have a different lifecycle from components. Components may mount, update, or unmount. An Effect can only do two things: to start synchronizing something, and later to stop synchronizing it. This cycle can happen multiple times if your Effect depends on props and state that change over time. React provides a linter rule to check that you've specified your Effect's dependencies correctly. This keeps your Effect synchronized to the latest props and state.

## You will learn

- How an Effect's lifecycle is different from a component's lifecycle
- How to think about each individual Effect in isolation
- When your Effect needs to re-synchronize, and why
- How your Effect's dependencies are determined
- What it means for a value to be reactive
- What an empty dependency array means
- How React verifies your dependencies are correct with a linter
- What to do when you disagree with the linter