Handwritten Notes ➡

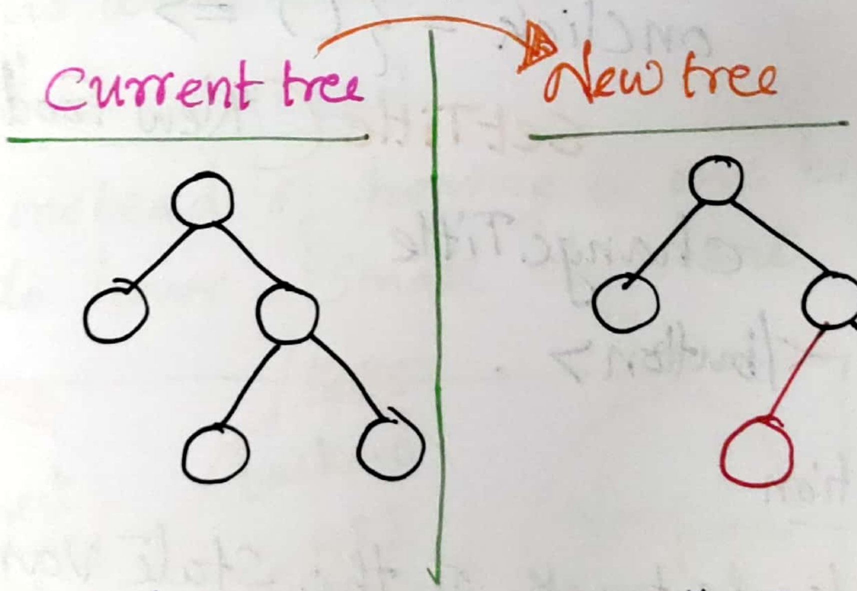
"The cornerstone of ReactJs : Reconciliation"

🔖 Save for Later

Ashraya K K
@ashrayaa

Why React is fast ?

- Because it has virtual DOM, Reconciliation, Diff algorithm.
- In Diff algorithm, current tree is compared with the new tree and the difference is reflected on the DOM.
- React Fibre is the updated reconciliation algm



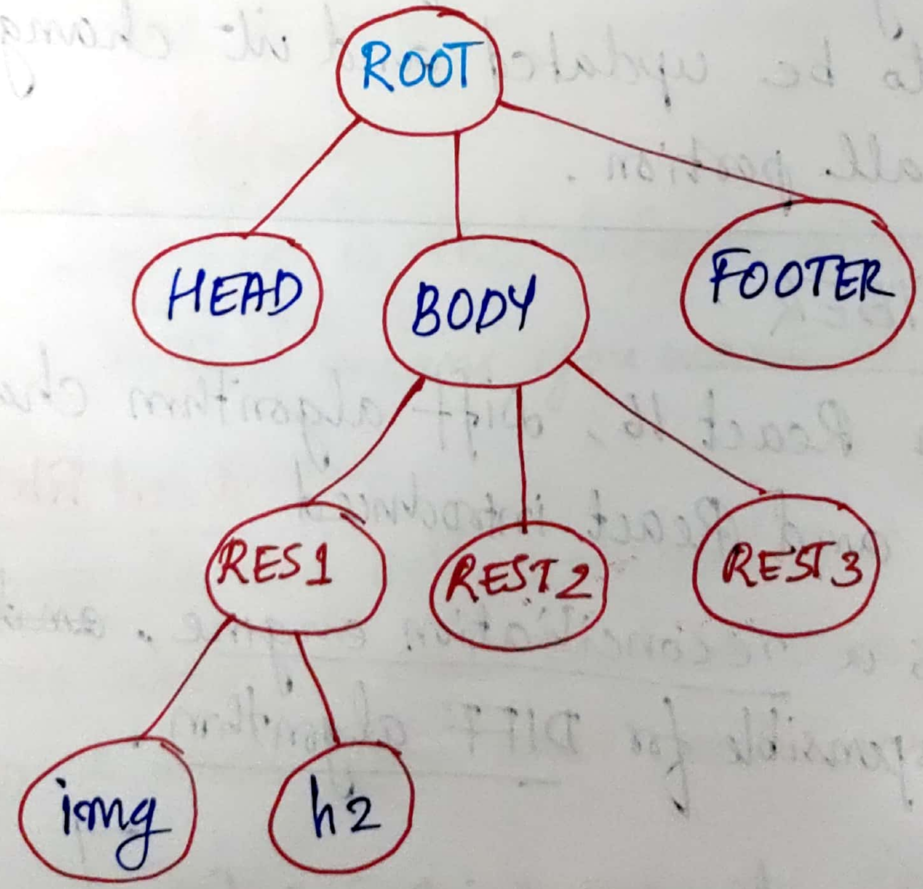
- React is fast because of it's fast DOM manipulation.
- Diff algm 'detects' what exactly got changed in the page & it'll just change that while re-rendering the whole tree.

VIRTUAL DOM

→ Let the structure of our DOM look like ↴

```
<>
    <head>
    <body>
        <Rest 1>
        <Rest 2> <img ../>
        <Rest 3>
    </body>
    ~~</head>~~ <footer/>
</>
```

→ We keep a representation DOM with us, which is known as virtual DOM.



Scanned with CamScanner

→ We need Virtual DOM for Reconciliation

* → Reconciliation is an algorithm that React uses to diff one tree from other.
It uses Diff Algorithm and in determining what needs to change and what does not in UI.

[To find out DIFFERENCE between one tree (Actual DOM) and other (VIRTUAL DOM)]

→ Diff Algorithm then finds out what needs to be updated and it change only that small portion.

REACT FIBER

→ In React 16, Diff algorithm changed a little and React introduced React Fiber

→ It's a reconciliation engine, ~~and~~ which is responsible for DIFF algorithm.

Render - means updating something in the DOM.

Reconciliation

→ helps to make React applications fast and efficient by minimizing the amount of work that needs to be done to update the changes.

→ So, you don't have to worry about what changes on every update.

Eg:-

```
<ul>
      <li> first </li>   } siblings.
      <li> second </li>
<ul>
```

when adding an element at the end of the children: The tree works well

```
<ul>
      <li> first </li>
      <li> second </li>
~~</ul>~~ <li> third </li>
</ul>
```

- render() function as creating a tree of React elements.
- On the next state or props update, render() fn will return a different tree of React elements.

Whenever react is updating the DOM, for eg :-

```
<ul>
    <li> Duke </li>
    <li> Villanova </li>
</ul>
```

Now, I introduced one child over the top, then react will have to do lot of efforts, react will have to re-render everything. That means, [react will have to change the whole DOM tree]

```
<ul>
    <li> Connecticut </li>
    <li> Duke </li>
    <li> Villanova </li>
```

As react has to re-render everything, it will not give you good performance.

In large-scale application, it is far too expensive.

SOLUTION — Introduction of Keys

→ React supports 'key' attribute.

→ When children have keys, React uses the key to match ~~#~~ children in the original tree with children in subsequent tree. Thus, making tree-conversion efficient

```
<ul>
    <li Key="2014"> Connecticut </li>
   <li Key="2015"> Duke </li>
   <li Key= "2016"> Villanova </li>
</ul>
```

Thus, react has to do very less work.

So, always use keys whenever you have multiple children.