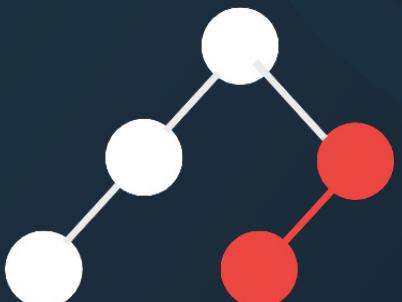


SUNIL VISHWAKARMA  
@linkinsunil



# React



## Virtual DOM

vs

**WTF is a  
Virtual DOM?**



## Real DOM



save it



follow me

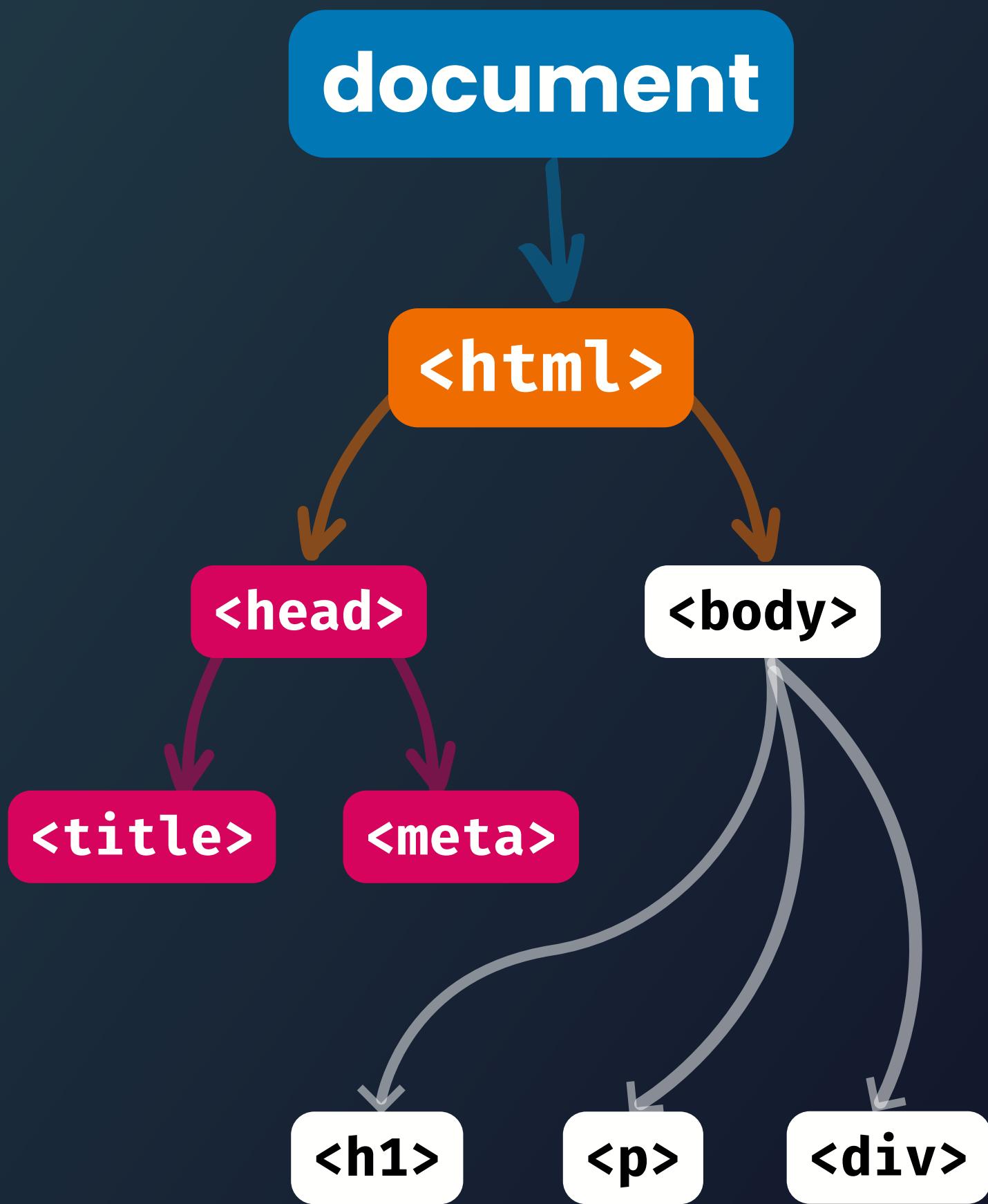
# What is DOM?

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.

Ahh !!  
Like a Family  
Tree?



# DOM: An HTML Structure



# The Problem with DOM

```
let fruits = ['Apple', 'Orange', 'Banana']
```

Lets say we want to update here from Orange to Lemon. Then we need to create a new array.

```
let fruits = ['Apple', 'Lemon', 'Banana']
```

In an efficient way we can just traverse to the fruits[1] and update only this element.

Now it's common to have a thousands node in a single SPA. So repainting the whole page for each change is **very-very expensive**.

Ideally, we'd like to only re-render items that receive updates, leaving the rest of the items as-is.

# The Methods of update

## Dirty Checking (slow)

In AngularJS 1.x, data changes were checked by recursively **traversing every node** at fixed intervals, which was inefficient as it required examining all nodes even if their data was up-to-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 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.

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 its speed and efficiency from the fact that it's lightweight.

A new virtual DOM is created after every re-render.

# Under the hood

**Reconciliation** is a process to compare and keep in sync the two files (Real and Virtual DOM). **Diffing algorithm** is a technique of reconciliation which is used by React.

## Is **Shadow DOM** same as the **Virtual DOM**?

No, they are different. The **Shadow DOM** is a browser technology designed primarily for **scoping variables and CSS** in web components. The virtual DOM is a concept implemented by libraries in JavaScript on top of browser APIs.

# Update Process in React

On the first load, `ReactDOM.render()` will create the Virtual DOM tree and real DOM tree.

As React works on Observable patterns, when any `event`(like key press, left click, api response, etc.) `occurred`, Virtual DOM tree nodes are notified for props change, If the properties used in that node are updated, the `node is updated` else left as it is.

React compares Virtual DOM with real DOM and updates real DOM. This process is called Reconciliation. React uses Diffing algorithm technique of Reconciliation.

Updated real DOM is repainted on browser.

# Additional Info

Virtual DOM is pure JS file and light weight, So capturing any update in Virtual DOM is much faster than directly updating on Real DOM.

React takes a few milliseconds before reconciliation. This allows react to bundle few processes. This increases efficiency and avoids unnecessary reprocessing. Because of this delay we should not rely on `state` just after `setState()`.

React does shallow comparison of props value. We need to handle deep comparison separately, immutable is the most common way to handle it.

Want explanation on

- Reconciliation and Diffing Algorithm?
- Use of keys in lists in React?

Let me know in the **COMMENT** section



SUNIL VISHWAKARMA  
@linkinsunil

# Thats a Wrap!

If you liked it, visit my profile and checkout for other  
**short and easy explanations**

**Context API vs Redux-Toolkit**

Feature ▾	Context API ▾	Redux-Toolkit ▾
State Management	Not a full-fledged state management tool. Passes down values and update functions, but does not have built-in ability to store, get, update, and notify changes in values.	A full-fledged state management tool with built-in ability to store, get, update, and notify changes in values.
Usage	Best for passing static or infrequently updated values and moderately complex state that does not cause performance issues when passed using props.	Best for managing large-scale, complex state that requires asynchronous actions and side-effects.
Code Complexity	Minimal setup and low learning curve. However, can become complex when used with a large number of components and nested Contexts.	
Performance	Can cause unnecessary re-renders if the state passed down is not simple and can require the use of additional memoization techniques to optimize performance.	
Developer Tools	Does not come with pre-built developer tools but can be used with third-party tools like React DevTools.	
Community	Has a large and active community	

**JavaScript Evolution**

- ES6 ES2015**
  - 1. let and const
  - 2. Arrow functions
  - 3. Default parameters
  - 4. Rest and spread operators
  - 5. Template literals
  - 6. Destructuring assignment
  - 7. Classes and inheritance
  - 8. Promises for asynchronous programming
  - 9. Symbols for creating unique object keys
  - 10. Iterators and generators
- ES9 ES2018**
  - 1. Object.getOwnPropertyDescriptors()
  - 2. Spread syntax for objects
  - 3. Promise.prototype.finally()
- ES10 ES2019**
  - 1. Array.prototype.flat()
  - 2. Array.prototype.flatMap()
  - 3. String.prototype.trimStart()
  - 4. String.prototype.trimEnd()
  - 5. Array.prototype.sort() (stable)
- ES11 ES2020**
  - 1. BigInt
  - 2. Nullish coalescing operator (??)
  - 3. Optional chaining operator (?)
  - 4. Promise.allSettled()
- ES12 ES2021**
  - 1. String.prototype.replaceAll()
  - 2. Logical assignment operators (|=, &=&, ??=)
- ES13 ES2022**
  - 1. Array.prototype.lastIndexOf()
  - 2. Object.hasOwn()
  - 3. at() for strings and arrays
  - 4. Top level await()

**React**

**Virtual DOM**

**useRef()**  
referencing values in React

When you want a component to remember some information, but you don't want that information to trigger new renders, you can use a ref.

**Lets See into**

1. How to add a ref to component?
2. How to update a ref's value?
3. How refs are different from state?
4. When to use refs?
5. Best practices for using refs?

{ Current }

⚠ Please Like & Share for no reason

**Redux Toolkit**  
Easiest Explanation Ever

React Redux Toolkit

like and share

swipe →

swipe →

share

# share

