

Full Stack Web Development

by Dr. Piyush Bagla



React

- **React** is a JavaScript library (developed by Facebook) that lets developers **build user interfaces (UIs)** by breaking them down into **small, reusable components**.
- React is a popular choice for building **single page applications (SPAs)**. In SPAs, when the user interacts with the application, instead of loading a new page, the content on the page changes dynamically.
- React is known for its performance and flexibility, and it is widely used in web development for both small and large applications.

History

- React was first designed by Jordan Walke, a software engineer at Facebook.
- It was first deployed for the Facebook News feed around 2011.
- In 2013, React was open-sourced at the JS conference.

Key features of React

- **Component-Based Architecture:** Each **component** is an independent, self-contained piece of the UI — for example, a button, a form input, or an entire page section — and manages its own **state** and **rendering**.
- Components can be **nested** inside other components, allowing developers to **compose** complex UIs out of simple building blocks. For example- <https://www.react.org>
- **Uses a Declarative Approach:** In declarative programming, you describe what you want the end result to look like, and the underlying system takes care of the details.
- **Virtual DOM:** React uses a Virtual DOM to manage updates to the actual DOM efficiently. When the state or props of a component change, React updates the Virtual DOM first and then compares it with the previous version to determine what needs to change in the actual DOM.
- **JSX:** React uses JSX (JavaScript XML), a syntax extension that allows developers to write UI components using a syntax similar to HTML, with the ability to embed JavaScript expressions within the markup.
- **Single Page Application.**

Important points

In React:

✅ **State** = Data that changes over time inside a component (example: user's input, counter values, fetched API data).

✅ **Rendering** = How a component draws (or re-draws) itself on the screen based on its state or props.

⚡ **In React, a component renders based on its state or props.**

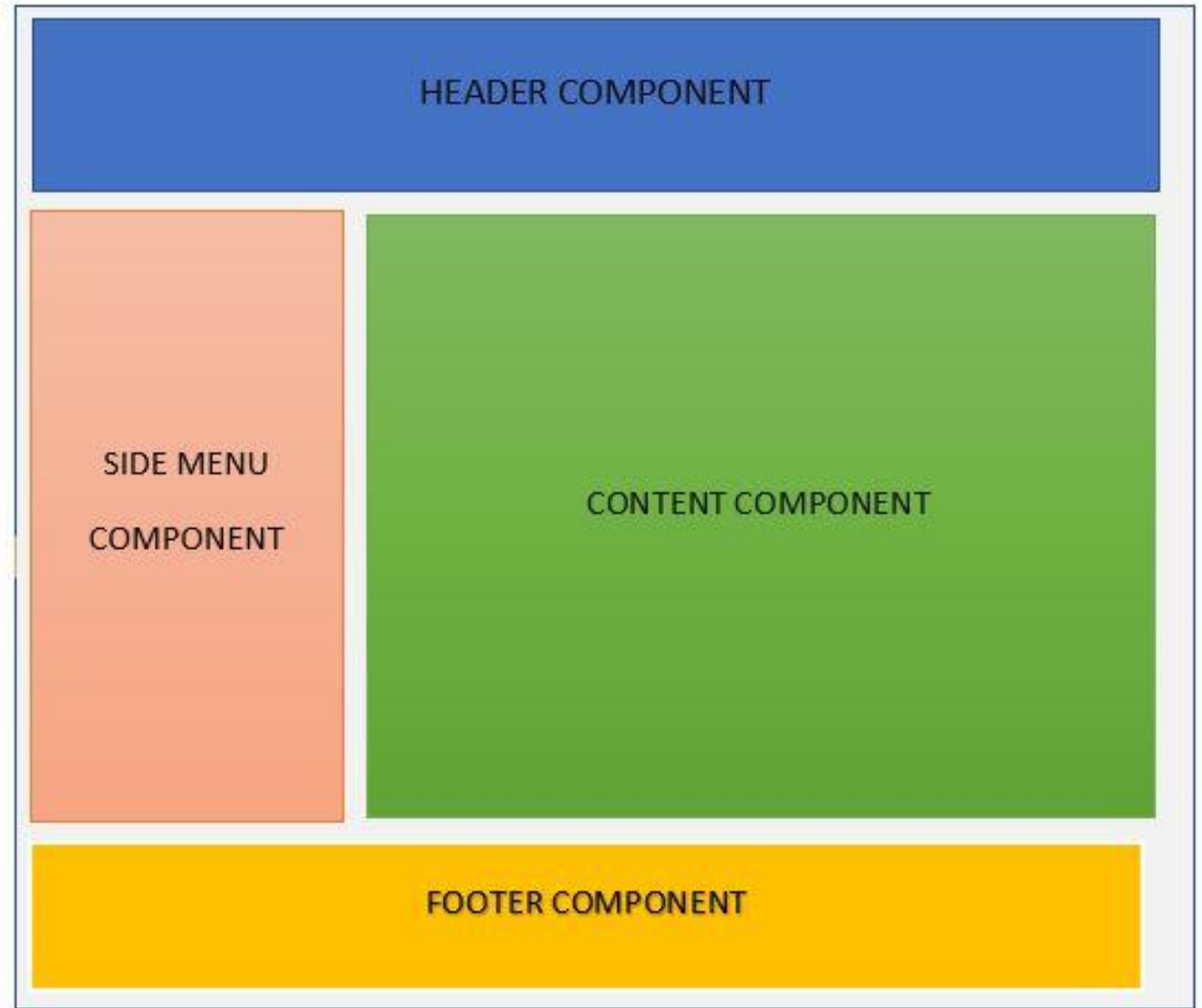
- **State** = Data **internal** to the component (can change over time).
- **Props** = Data **passed** into a component from its parent (read-only).

When either state or props change, React triggers a re-render of the component to reflect the new data in the UI.

In real world:

- **State** = what's in your cart (can change).
- **Props** = sending one product's info to show it nicely.

Component-Based Architecture




Declarative Approach

- An analogy for this concept in React is ordering a pizza at a restaurant:
- **Defining the UI:** In React, you define the desired user interface using **components** and **JSX**. You specify what the components should look like and how they should behave based on the data (props and state) you provide.
- **React's Job:** React takes care of rendering the components and updating the **actual DOM** based on the defined **state** and **props**. It efficiently handles changes to the UI by using the **Virtual DOM** and a **diffing algorithm**.
- **Final Result:** Just like the pizza, you get a user interface that matches your description, and React manages the rendering process for you.

Declarative Approach

React's Declarative Approach — Pizza Analogy

1. Defining the UI (Your Pizza Order)

- You walk into a restaurant and tell the waiter:
 "I want a **large pepperoni pizza** with **extra cheese**."
- You **describe what you want**, not **how** they should bake it, mix the dough, add the toppings, or manage the oven.
- In **React**, you similarly **define**:

Jsx

```
<Pizza size="Large" toppings={["Pepperoni", "Extra Cheese"]} />
```

You specify *what* the component should render — not *how* it gets rendered.

Declarative Approach

2. React's Job (Kitchen's Work)

- After your order, the **restaurant (kitchen)** handles:
 - Mixing ingredients
 - Baking the pizza
 - Managing the timing
- **You don't worry** about how they rearrange their kitchen for efficiency!


In React:

- React uses its **Virtual DOM** and **diffing algorithm** to:
 - Calculate what needs to change.
 - Efficiently update only the necessary parts of the real DOM.
- You **don't manually update** the DOM (no `document.createElement`, `appendChild`, etc.).

Declarative Approach



3. Final Result (Enjoying Your Pizza!)

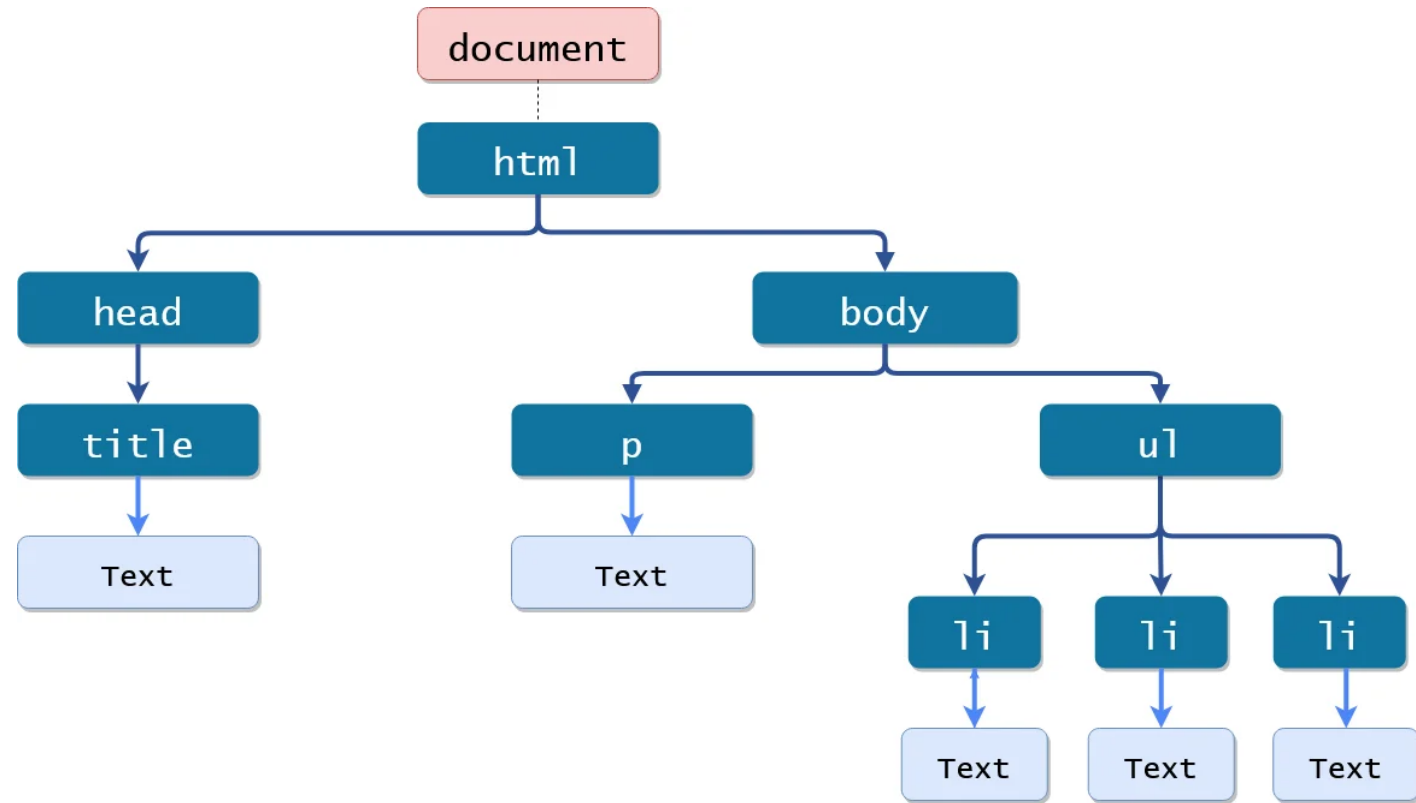
- The waiter brings you **exactly** the pizza you asked for. 
- You enjoy it without knowing all the behind-the-scenes complexity.

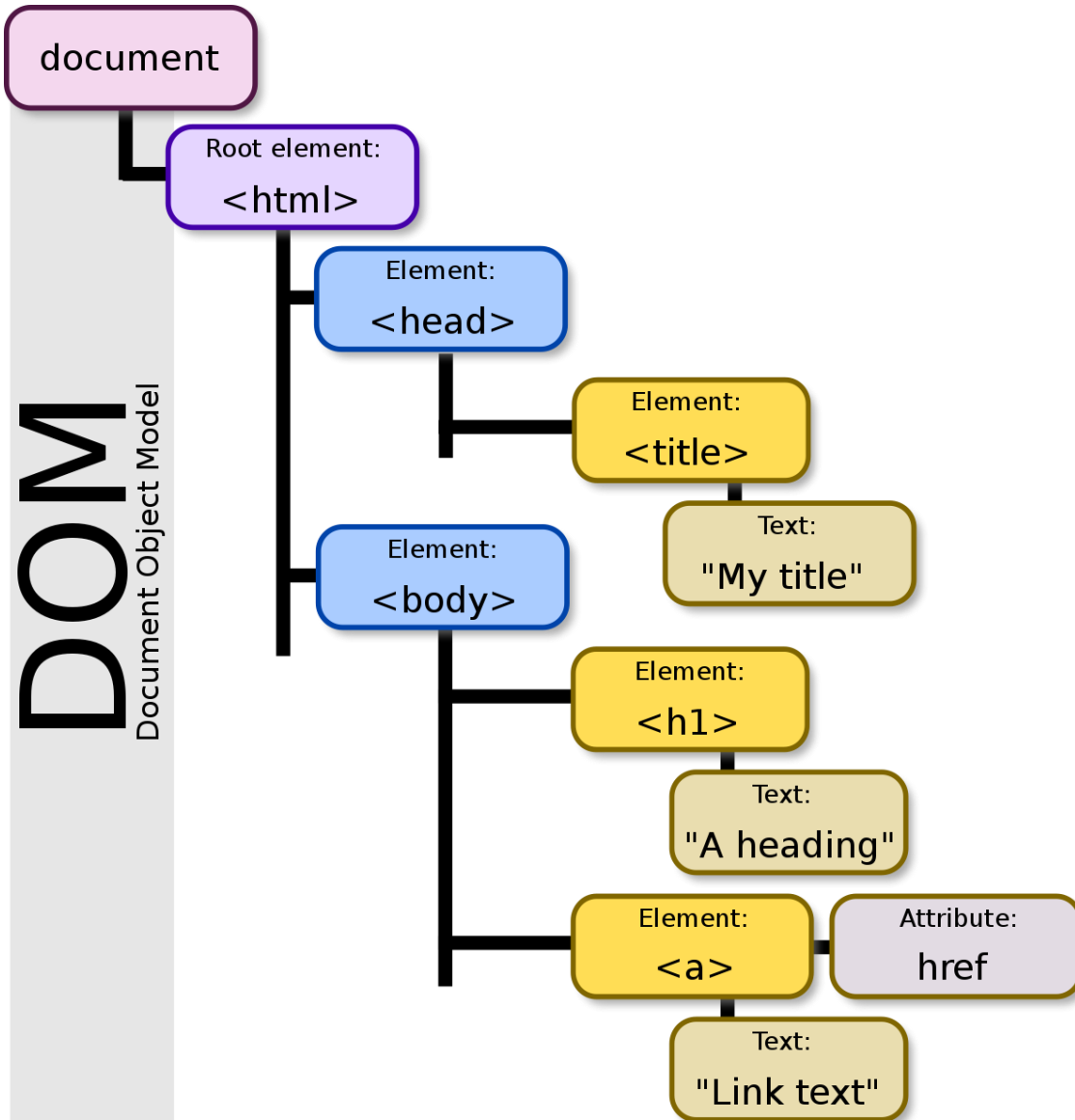
Similarly, in React:

- You see the **final user interface**, which exactly matches your **components** and **state/props**.
- React handles all the heavy lifting invisibly.

DOM

- The **Document Object Model** is a model that defines [HTML](#) and XML documents as a tree structure, where each node of the tree is an object which represents a part of the document.
- The **DOM** stands for Document Object Model, it is an interface that allows developers to interact and control the Elements of the web.

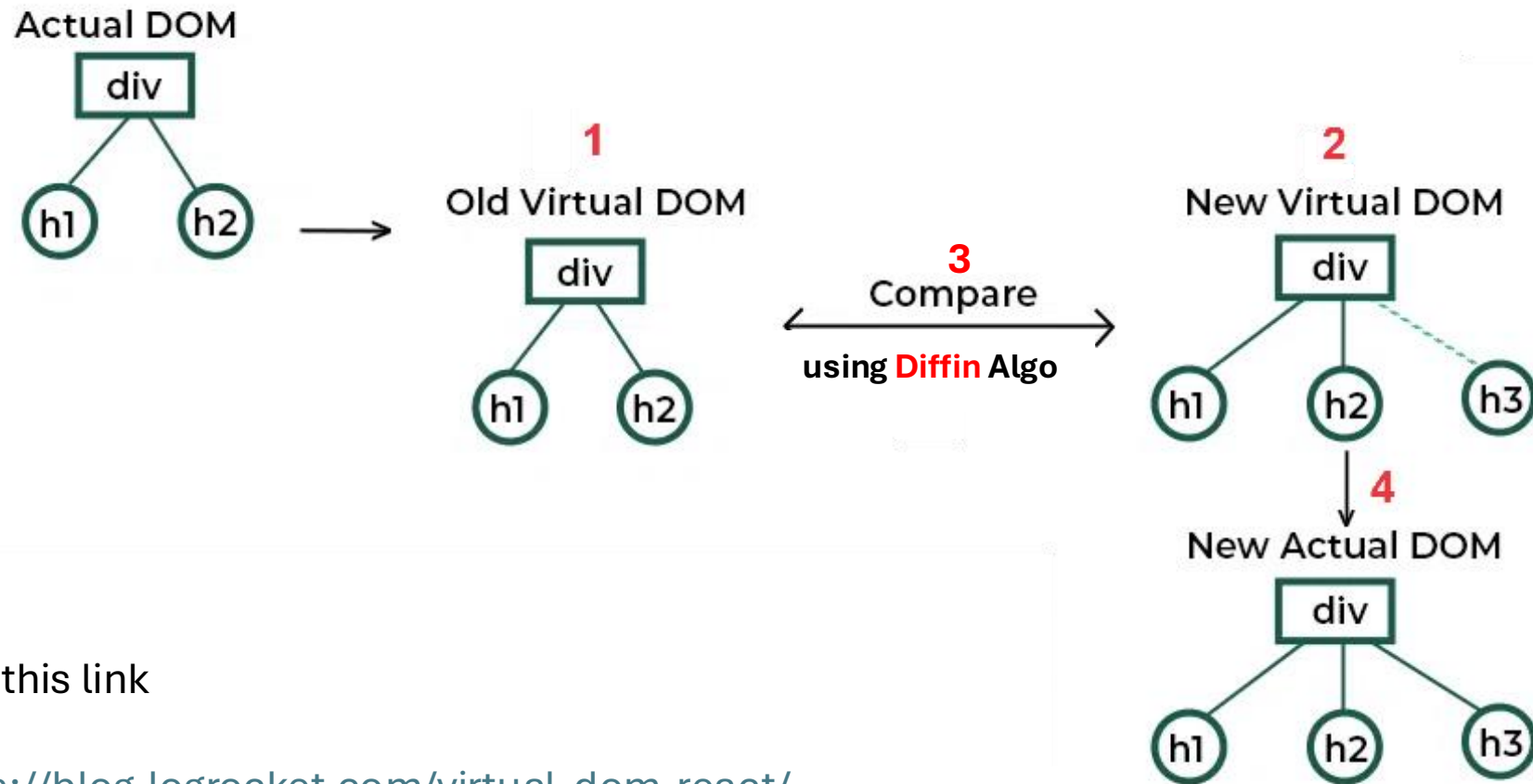




- **Disadvantages of real DOM :**

Every time the DOM gets updated, the updated element and its children have to be rendered again to update the UI of our page. For this, each time there is a component update, the DOM needs to be updated, and the UI components have to be re-rendered.

Virtual DOM



Visit this link

<https://blog.logrocket.com/virtual-dom-react/>

Virtual DOM

What is the Virtual DOM?

The Virtual DOM, often abbreviated as VDOM, is a simplified, lightweight, in-memory representation of the actual Document Object Model (DOM) of a web page. In a typical web application, the DOM represents the structure and content of the webpage. Any changes to the data or state of a web application are reflected in the DOM, causing the browser to update the user interface accordingly. However, manipulating the actual DOM can be slow and resource-intensive, especially when dealing with complex user interfaces or frequently changing data. This is where the Virtual DOM comes into play.

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.

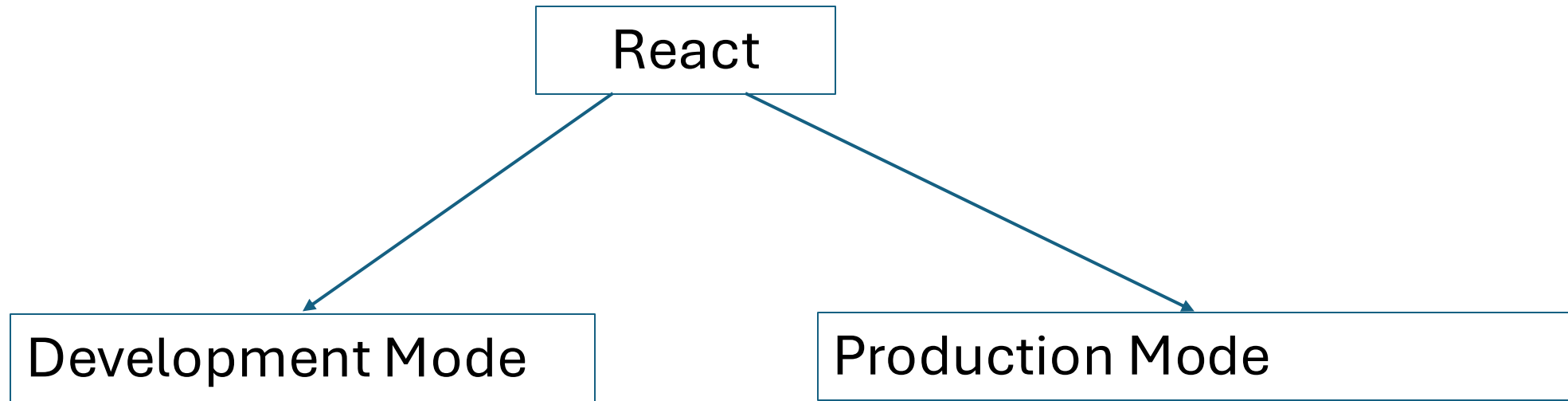
This entire process of transforming changes to the real DOM is called **Reconciliation**.

Diffin Algo

The "**Differential Algorithm**" in the context of the Virtual DOM refers to the process by which React identifies and calculates the differences (or "**diffs**") between two Virtual DOM trees: the previous Virtual DOM tree and the newly generated Virtual DOM tree after a component update. This algorithm is at the core of React's efficient updating mechanism.

Rendering | Render

In the context of React, "render" refers to the process of creating and updating the user interface (UI) of an application by translating React components into DOM elements. This involves taking the JSX code or React components and transforming them into HTML elements that can be displayed in the browser.



For local development, debugging and testing.

For live deployment and end-user access.

- These are major modes of React; however, there are other modes as well.
- As of May 2025, the latest stable version of React is **React 19.0.0**, released on **December 5, 2024**

Development Mode

```
<!DOCTYPE html>
<html>
<head>
<script src="https://unpkg.com/react@18/umd/react.development.js" crossorigin></script>
<script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js" crossorigin></script>
<script src="https://unpkg.com/@babel/standalone/babel.min.js"></script>
</head>
<body>
  <div id="mydiv"></div>
  <script type="text/babel">
    const container = document.getElementById('mydiv');
    const root = ReactDOM.createRoot(container);
    root.render(<h1>Hello World!</h1>);
  </script>
</body>
</html>
```


Here's a detailed explanation of how the provided code works and the purpose of each part:

Script Inclusions:

1. React Library (`react.development.js`):

- This script includes the React development library.
- It provides the necessary functions and classes for writing React code in your application.
- Since the file is the development version, it includes helpful debugging and error messages to aid in development.

2. ReactDOM Library (`react-dom.development.js`):

- This script includes the ReactDOM development library.
- ReactDOM is an extension of React that provides functions for working with the DOM (Document Object Model) in a React application.
- It includes methods such as `ReactDOM.createRoot()` and `root.render()` to control the rendering of React components into the DOM.

3. Babel Standalone (`babel.min.js`):

- This script includes the Babel standalone library.
- Babel is a JavaScript compiler that converts modern JavaScript code, including JSX syntax and ES6 features, into code that older browsers can understand.
- By using Babel, you can write React code in JSX syntax directly in your HTML file.

Babel is a tool that converts JSX syntax into plain JavaScript code.

HTML Structure:

- Root Container:

- The HTML file includes a `<div>` element with the ID `mydiv`.
- This element serves as the root container for the React application.
- When you render a React application, it needs a root container to attach the React components to.

React Code in HTML:

- Script Tag (`<script type="text/babel">`):

- The React code is placed inside a `<script>` tag with the type attribute set to `"text/babel"`.
- This tells the browser to use the Babel library to compile the code.

- React Code `<script type="text/babel"> </script>`:

`type="text/babel"` tells the browser to use Babel to **transpile JSX** inside this script block.

- Root Container Selection (`document.getElementById('mydiv')`):

- The script selects the root container (`<div>` element with ID `mydiv`) using `document.getElementById()`.
- This allows the script to reference the container where the React component will be rendered.

- **Creating a React Root (`ReactDOM.createRoot(container)`):**

- The `ReactDOM.createRoot()` method creates a React root instance in the specified container (`mydiv`).
- This method is part of ReactDOM and sets up the container for rendering React components.
- This process is about setting up the "entry point" of your application, where your root component will be inserted into the HTML.

- **Rendering the Component (`root.render(<Hello />)`):**

- The `root.render()` method renders the `Hello` component inside the root container.
- The component is written using JSX syntax (`<Hello />`) and is passed as an argument to `root.render()`.
- The rendering process updates the DOM by inserting the component's output (an `<h1>` element) into the root container.

JSX

- **What is JSX?**

- JSX stands for **JavaScript XML**.
- It allows writing **HTML-like code** inside JavaScript.

- **Why use JSX?**

- Makes React code **cleaner and more readable**.
- Avoids the need for createElement() and appendChild() methods.
- Converts HTML tags to **React elements** using React.createElement() behind the scenes.

- **Not Mandatory, but Helpful:**

- JSX is **not required**, but it greatly **simplifies UI code** in React apps.

JSX Key Syntax Rules

- Use **className** instead of class (because class is a JS keyword).
`const myElement = <h1 className="myclass">Hello World</h1>;`
- Embed JS **expressions** in { }:
`const name = "Swastik"; const greeting = <h1>Hello, {name}</h1>;`
- **If-statements** are not allowed *inside* JSX, but ternary operators are:
`const isLoggedIn = true; const message = <p>{isLoggedIn ? "Welcome back!" : "Please sign in."}</p>;`
- It follows XML Rules.

Without JSX

```
<body>  
<div id="root"></div>  
</body>
```

```
// Select the root element from the DOM  
const root = document.getElementById('root');  
  
// Create an <h1> element  
const myElement = document.createElement('h1');  
  
// Set the text content of the <h1> element  
myElement.textContent = 'I Love JSX!';  
  
// Append the <h1> element to the root element  
root.appendChild(myElement);
```

With JSX

```
<body>  
<div id="root"></div>  
</body>
```

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
ReactDOM.render(<h1>I Love JSX!</h1>, document.getElementById('root'));
```

Old Method

```
import React from 'react';  
import ReactDOM from 'react-dom/client';  
const myElement = <h1>I Love JSX!</h1>;  
const root = ReactDOM.createRoot(document.getElementById('root'));  
root.render(myElement);
```

New Method

Important Commands

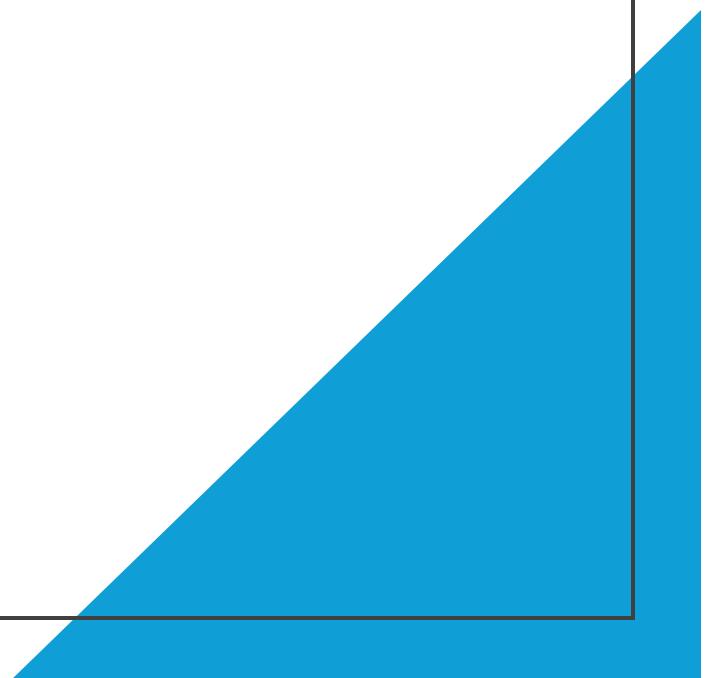
- **Node Version Check :** `npm -v`
- **npm & npx Version Check :** `npx/npm -v`

Create react app using npx :

`npx create-react-app reactproject`

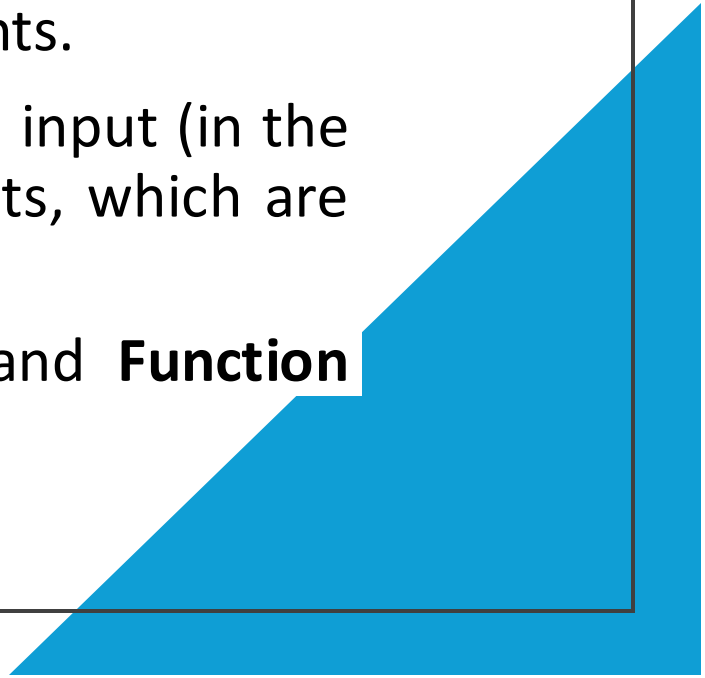
Create react app using npm :

`npm install -g create-react-app`
`create-react-app reactproject`



Components

- In React, components are the building blocks of user interfaces. They are reusable, self-contained pieces of code that encapsulate a specific functionality or piece of the user interface.
- Components are like functions that return HTML elements.
- Just like functions in JavaScript, React components take input (in the form of properties or props) and return React elements, which are essentially HTML elements.
- Components are of two types: **Class components** and **Function components**



Props

1. **Definition:** Props (short for properties) are inputs to a React component. They are passed from parent components to child components.
2. **Immutable:** Props are immutable, meaning they cannot be modified by the component receiving them. They are read-only.
3. **Usage:**
 1. Used for passing data from parent components to child components.
 2. Components can receive props as function parameters or access them via *this.props* in class components or directly in functional components.

Props Example

```
// ParentComponent.js
```

```
<ChildComponent name="John" age={30} />
```

```
// ChildComponent.js
```

```
function ChildComponent(props) {
```

```
  return <h1>Hello, {props.name}! You are {props.age} years old.</h1>;
```

```
}
```



State

1. **Definition:** State represents the internal data/state of a component. It is managed within the component and can be modified over time.
2. **Mutable:** Unlike props, state is mutable and can be updated using the *setState()* method provided by React.
3. **Usage:**
 1. Used for managing component-specific data that may change over time.
 2. State is typically initialized in the constructor (for class components) or using the *useState()* hook (for functional components).

Most of your components should simply take some data from props and render it. However, sometimes you need to respond to user input, a server request, or the passage of time. For this, you use state.

```
import React from "react";
import ReactDOM from 'react-dom';

class Student extends React.Component
{
    constructor()
    {
        super();
        this.state = {roll:"38", name: "Ram"};
    }
    render() {
        return (
            <>
                <h1 style={{textAlign:'center'}}> Roll number is {this.state.roll} </h1>
                <h2> Name is {this.state.name} </h2>
            </>
        );
    }
}

const a = document.getElementById("cse");
const b = ReactDOM.createRoot(a);
b.render(<Student/>);
```

Hook

- A hook in React is a special function that allows you to use state and other React features , such as lifecycle methods in functional components. Hooks were introduced in React 16.8 to provide a more expressive and composable way to write React components.
- Before hooks, functional components were stateless and couldn't use lifecycle methods or manage state.
- With hooks, functional components can now manage state, perform side effects, and have lifecycle behaviors similar to class components.

A large orange circle is positioned on the left side of the slide, partially cut off by the edge.

When would I use a Hook?

If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now, you can use a Hook inside the existing function component.



Rules for hooks:

- Hooks can only be called inside React function components.
- Hooks can only be called at the top level of a component.
- Hooks cannot be conditional
- Hooks will not work in React class components.



useState()

```
import React, { useState } from "react";

function MyComponent()
{
  var [mytext, setmytext] = useState('Full Stack Web Development');
  function changetext()
  {
    setmytext("Piyush Bagla");
  }
  return (
    <>
      <h1>{mytext}</h1>
      <button onClick={changetext}>Change</button>
    </>
  );
}

export default MyComponent;
```

React Forms

- In HTML, form data is usually handled by the DOM.
- In React, form data is usually handled by the components.
- When the data is handled by the components, all the data is stored in the component state.
- You can control changes by adding event handlers in the **onChange** attribute.
- We can use the **useState** Hook to keep track of each input value and provide a "single source of truth" for the entire application.
- To access the fields in the event handler use the **event.target.name** and **event.target.value** syntax.

React Forms

```
function MyForm()
{
    return (
        <form>
            <label>Enter your name: <input type="text" /> </label>
        </form> )
}

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<MyForm />);
```

Handling Forms

```
import React, { useState } from "react";

function ReactForm() {
  const [username, setUsername] = useState("");

  function f1(event) {
    setUsername(event.target.value);
  }

  return (
    <form action="">
      <label>Username
        <input type="text" value={username} onChange={f1} />
      </label>

    </form>
  );
}

export default ReactForm;
```

Submitting Forms

```
import React, { useState } from "react";
function ReactForm() {
  const [username, setUsername] = useState("");
  function f1(event) {
    setUsername(event.target.value);
  }
  function mysubmit(event) {
    event.preventDefault();
    console.log(username);
  }
  return (
    <form onSubmit={mysubmit}>
      <label>Username
        <input type="text" value={username} onChange={f1} />
      </label>
      <input type="submit" />
    </form>
  );
}
export default ReactForm;
```

React Router

- **React Router enables "client side routing".**
- In traditional websites, the browser requests a document from a web server, downloads and evaluates CSS and JavaScript assets, and renders the HTML sent from the server. When the user clicks a link, it starts the process all over again for a new page.
- Client side routing allows your app to update the URL from a link click without making another request for another document from the server. Instead, your app can immediately render some new UI and make data requests with fetch to update the page with new information.
- This enables faster user experiences because the browser doesn't need to request an entirely new document or re-evaluate CSS and JavaScript assets for the next page. It also enables more dynamic user experiences with things like animation.

React Router

- **Components:** All components are included in the initial JavaScript bundle loaded when the app first starts. React Router handles displaying the correct component based on the URL.
- **Data:** Specific data for each component is fetched from the server as needed. This data fetching happens asynchronously, allowing the app to update the content dynamically without a full page reload.
- So, while the components themselves come from the initially loaded JavaScript bundle, the data that these components display can come from the server via API calls.



Router Code – App.js

```
import React from "react";
import Home from "./pages/Home";
import AboutUs from "./pages/AboutUs";
import ContactUs from "./pages/ContactUs";
import { BrowserRouter, Route, Routes } from "react-router-dom";
import Layout from "./pages/Layout";

function App()
{
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home/>}/>
        <Route path="about" element={<AboutUs/>}/>
        <Route path="contact" element={<ContactUs/>}/>
      </Routes>
    </BrowserRouter>
  )
}
export default App;
```

Install router

install react-router-dom@6.

Demonstration for router

<https://reactrouter.com/en/main/start/overview>

Router Code – Home.js

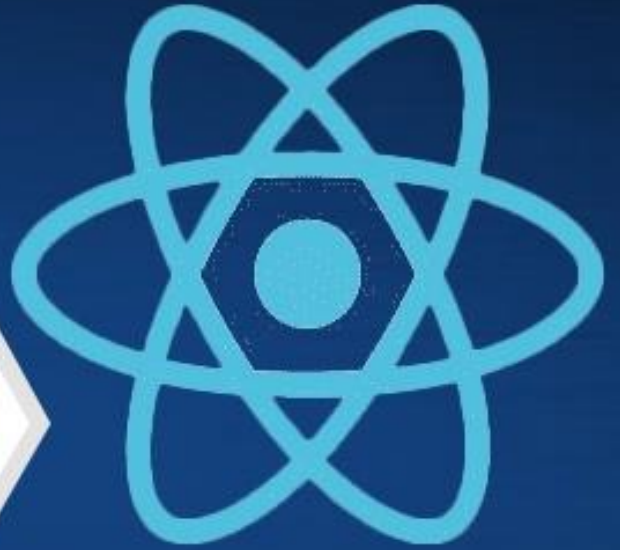
```
import React from "react";
import { Link } from "react-router-dom";
function Home()
{
  return(
    <>
    <header>
      <nav>
        <ul>
          <li><Link to= "/"> Home </ Link></li>
          <li><Link to= "/about"> About </ Link></li>
          <li><Link to= "/contact"> Contact Us</ Link></li>
        </ul>
      </nav>
    </header>
    < section>
      <h1>Home Page</ h1>
    </ section>
  </>
)
}
export default Home;
```

Router Code – AboutUs.js

```
import React from "react";
import { Link } from "react-router-dom";
function AboutUs()
{
  return(
    <>
      <header>
        <nav>
          <ul>
            <li><Link to= "/"> Home </ Link></li>
            <li><Link to= "/about"> About </ Link></li>
            <li><Link to="/contact"> Contact Us</ Link></li>
          </ul>
        </nav>
      </header>
      < section>
        <h1> AboutUs Page</ h1>
      </ section>
    </>
  )
}
export default AboutUs;
```

Router Code – ContactUs.js

```
import React from "react";
import { Link } from "react-router-dom";
function ContactUs()
{
  return(
    <>
      <header>
        <nav>
          <ul>
            <li><Link to= "/"> Home </ Link></li>
            <li><Link to= "/about"> About </ Link></li>
            <li><Link to="/contact"> Contact Us</ Link></li>
          </ul>
        </nav>
      </header>
      < section>
        <h1> ContactUs Page</ h1>
      </ section>
    </>
  )
}
export default ContactUs;
```



ANGULAR

VUE

REACT

Feature	React	Angular	Vue
Developed by	Facebook	Google	Evan You
Initial Release	2013	2010	2014
Language	JavaScript, JSX	TypeScript	JavaScript
Architecture	Component-based	Component-based, MVC	Component-based, MVVM
Data Binding	One-way data binding	Two-way data binding	Two-way data binding
DOM	Virtual DOM	Real DOM	Virtual DOM
Learning Curve	Moderate	Steep	Gentle
State Management	React Context, Redux, MobX	NgRx	Vuex
Template Syntax	JSX	HTML + Angular directives	HTML-based template syntax
Routing	React Router	Angular Router	Vue Router
Performance	Fast with Virtual DOM	Good with real DOM but heavier	Fast with Virtual DOM
Community Support	Large and active	Very large and strong	Growing and active
Ecosystem	Rich ecosystem, many third-party libraries	Integrated suite of tools and libraries	Growing ecosystem, many plugins
CLI	Create React App	Angular CLI	Vue CLI
Size	Smaller library size (~100 KB)	Larger framework size (~500 KB)	Smaller framework size (~80 KB)
Flexibility	Highly flexible, library approach	Less flexible, more opinionated framework	Flexible, progressive framework
Component Reusability	High	High	High
Scalability	Good for large applications, requires setup	Excellent for large applications, robust tooling	Good for medium to large applications, but can scale
Mobile Development	React Native	NativeScript, Ionic	Vue Native, Weex
Testing	Jest, Enzyme	Jasmine, Karma	Jest, Mocha, Chai
Documentation	Good, detailed	Excellent, comprehensive	Good, improving

Axios library for fetching data

Axios is a popular JavaScript library used to make HTTP requests from the browser or Node.js. It is often used in React and other JavaScript applications to fetch data from APIs. Axios provides a simple and clean API for performing various types of HTTP requests such as GET, POST, PUT, DELETE, and more.

Key Features of Axios

- **Promise-based:** Axios uses Promises, which makes it easy to work with asynchronous operations.
- **Interceptors:** Allows you to intercept requests or responses before they are handled by then or catch.
- **Automatic JSON Data Transformation:** Automatically transforms JSON data if the response content-type is JSON.
- **Error Handling:** Provides robust error handling.
- **CSRF Protection:** Supports Cross-Site Request Forgery (CSRF) protection.
- **Request and Response Transformation:** You can transform the request and response data.
- **Cancellation:** Supports request cancellation using cancel tokens.
- **Node.js Support:** Can be used in both browser and Node.js environments.