

React Fundamentals

Module -3

What is React ?

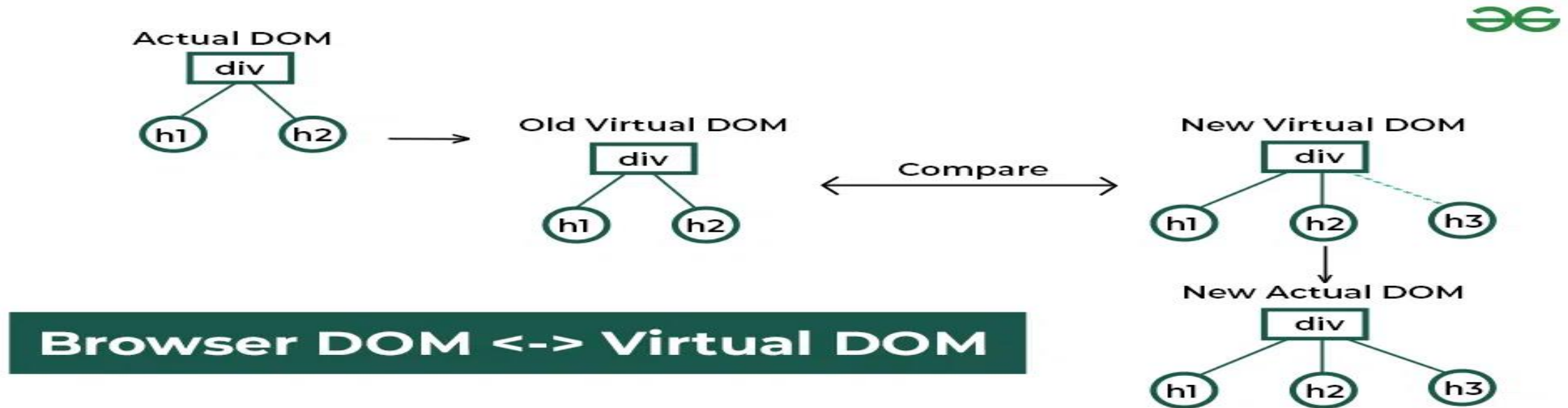
- React is a JavaScript library for building user interfaces (UIs) on the web. React is a declarative, component based library that allows developers to build reusable UI components and It follows the Virtual DOM (Document Object Model) approach, which optimizes rendering performance by minimizing DOM updates. React is fast and works well with other tools and libraries.

History of React

- React was invented by Facebook developers who found the traditional DOM slow. By implementing a virtual DOM, React addressed this issue and gained popularity rapidly.
- The current stable version of ReactJS is 18.2.0, released on June 14, 2022. The library continues to evolve, introducing new features with each update.

How does React work?

- React operates by creating an in-memory virtual DOM rather than directly manipulating the browser's DOM. It performs necessary manipulations within this virtual representation before applying changes to the actual browser DOM. React is efficient, altering only what requires modification.



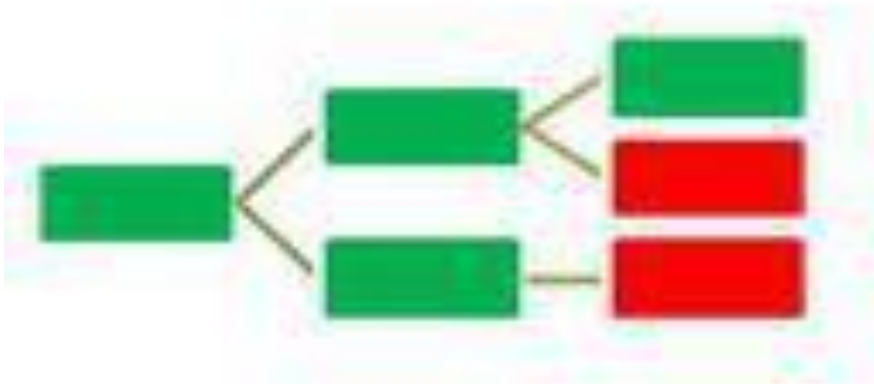
Features of React

- React is one of the most demanding JavaScript libraries because it is equipped with a ton of features which makes it faster and production-ready.
- **1. Component-Based Architecture**
- React provides the feature to break down the UI into smaller, self-contained components. Each component can have its **own state and props..**
- **2. JSX (JavaScript Syntax Extension)**
- JSX is a syntax extension for JavaScript that allows developers to write HTML-like code within their JavaScript files. It makes React components more readable and expressive.
- **const name="GeekforGeeks";**
- **const xyz = <h1>Welcome to {name}</h1>;**

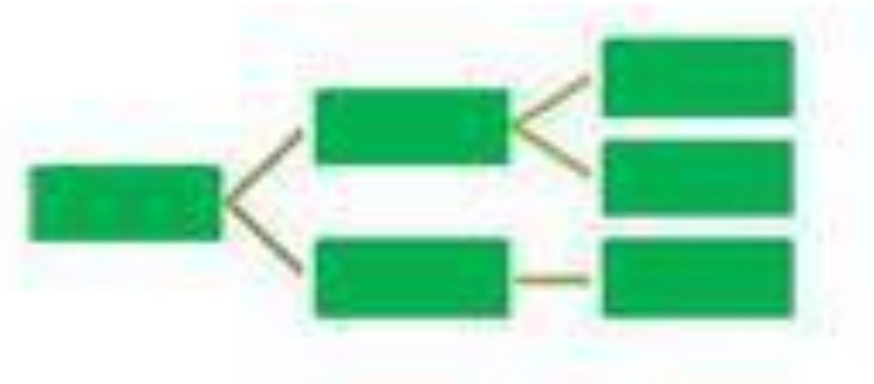
- **3. Virtual DOM**

- React maintains a lightweight representation of the actual DOM in memory. When changes occur, React efficiently updates only the necessary parts of the DOM.

Virtual DOM

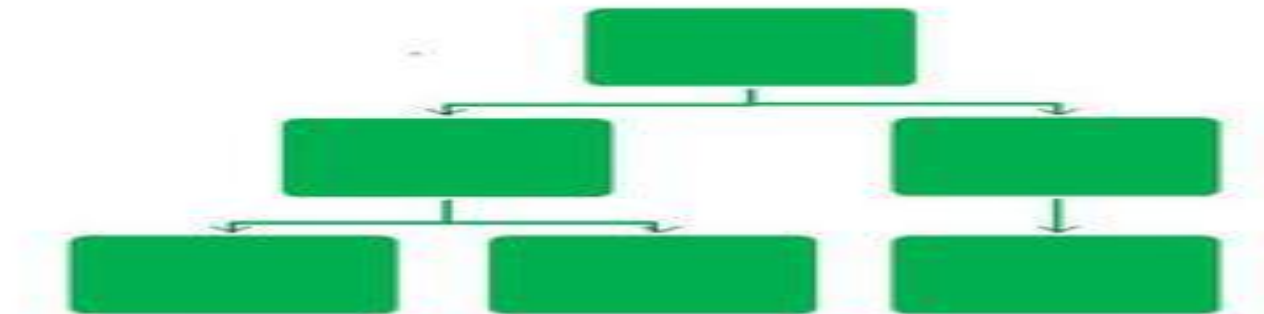


DOM



- **4. One-way Data Binding**

- One-way data binding, the name itself says that it is a one-direction flow. The data in react flows only in one direction i.e. the data is transferred from top to bottom i.e. from parent components to child components. The properties(props) in the child component cannot return the data to its parent component but it can have communication with the parent components to modify the states according to the provided inputs.

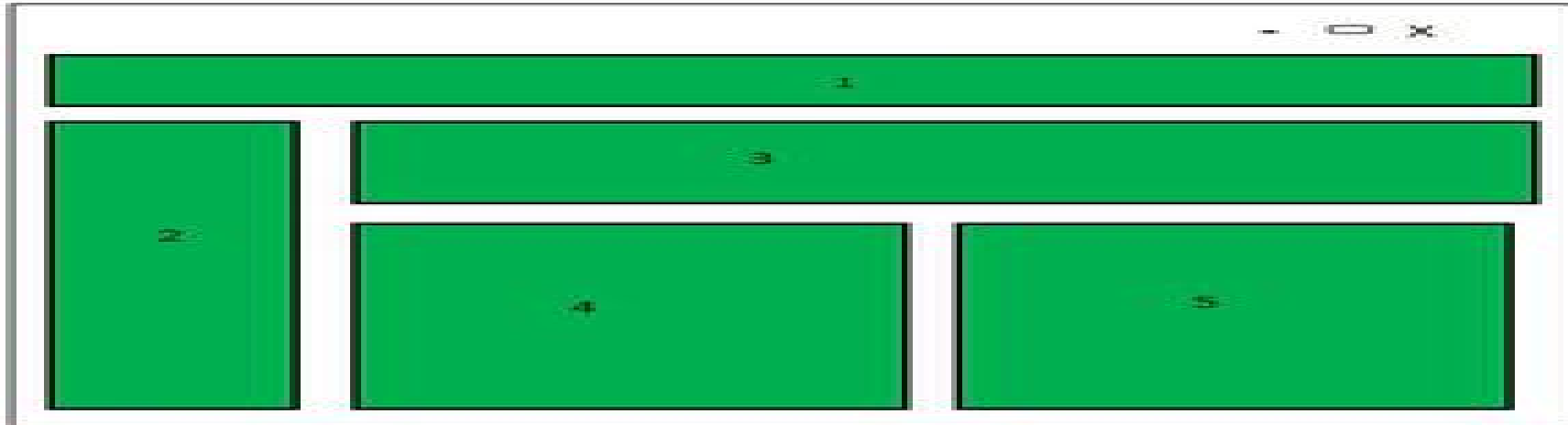


- **5. Performance**

- As , react uses virtual DOM and updates only the modified parts. So , this makes the DOM to run faster. DOM executes in memory so we can create separate components which makes the DOM run faster.

- **6. Components**

- React divides the web page into multiple components as it is component-based. Each component is a part of the UI design which has its own logic and design as shown in the below image. So the component logic which is written in JavaScript makes it easy and run faster and can be reusable.



- **7. Single-Page Applications (SPAs)**

- React is recommended in creating SPAs, allowing smooth content updates without page reloads. Its focus on reusable components makes it ideal for real-time applications.

Advantages of ReactJS

- Reusable Components
- Open Source
- Efficient and Fast
- Works in Browser
- Large Community

Components

- Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.
- Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.

Component Based



Types Of Components

- Components come in two types,
 1. Class components
 2. Function components



Class Component

- A class component must include the `extends React.Component` statement. This statement creates an inheritance to **React.Component**, and gives your component access to **React.Component's** functions.
- The component also requires a **render() method**, this method returns HTML.

Class Components

A class component requires you to extend from `React.Component`. The class must implement a `render()` member function which returns a React component to be rendered, similar to a return value of a functional component. In a class-based component, props are accessible via `this.props`.

Syntax:-

```
class class_name extends Component {  
  render( ) {  
    return React Element  
  }  
}
```

```
class Student extends Component {  
  render( ) {  
    return <h1>Hello {this.props.name}</h1>  
  }  
}
```

Ex:-

```
class Student extends Component {  
  render( ) {  
    return <h1>Hello Rahul</h1>  
  }  
}
```

Function Component

It is a JavaScript function which accepts a single “props” object argument with data and returns a React Element.

Syntax:-

```
function func_name ( ) {  
    return React Element;  
}
```

```
const Student = ( ) => {  
    return <h1>Hello Rahul</h1>  
}
```

Ex:-

```
function Student( ){  
    return <h1>Hello Rahul</h1>  
}
```

Syntax:-

```
function func_name (props) {  
  return React Element;  
}
```

Ex:-

```
function Student(props){  
  return <h1>Hello Rahul</h1>  
}
```

```
function Student(props){  
  return <h1>Hello {props.name}</h1>  
}
```

```
const Student = (props) =>{  
  return <h1>Hello {props.name}</h1>  
}
```

Rendering a Component

```
ReactDOM.render(<Student />, document.getElementById("root"));
```


```
ReactDOM.render(<Student name="Rahul"/>, document.getElementById("root"));
```

Ex:-

```
function Student(props){  
  return <h1>Hello {props.name}</h1>  
}
```

```
ReactDOM.render(<Student name="Rahul"/>, document.getElementById("root"));
```

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".

<u>Functional Components</u>	<u>Class Components</u>
<p>A functional component is just a plain JavaScript pure function that accepts props as an argument and returns a React element(JSX).</p>	<p>A class component requires you to extend from React. Component and create a render function that returns a React element.</p>
<p>There is no render method used in functional components.</p>	<p>It must have the render() method returning JSX (which is syntactically similar to HTML)</p>
<p>Functional components run from top to bottom and once the function is returned it can't be kept alive.</p>	<p>The class component is instantiated and different life cycle method is kept alive and is run and invoked depending on the phase of the class component.</p>
<p>Also known as Stateless components as they simply accept data and display them in some form, they are mainly responsible for rendering UI</p>	<p>Also known as Stateful components because they implement logic and state.</p> 

React lifecycle methods (for example, `componentDidMount`) cannot be used in functional components.

React lifecycle methods can be used inside class components (for example, `componentDidMount`).

Hooks can be easily used in functional components to make them Stateful.

It requires different syntax inside a class component to implement hooks.

Example:

Example:

```
const [name,SetName]=  
  React.useState(' ')
```

```
constructor(props) {  
  super(props);  
  this.state = {name: ' '}  
}
```

Constructors are not used.

Constructor is used as it needs to store state.

Functional vs Class Component

Use functional components if you are writing a presentational component which doesn't have its own state or needs to access a lifecycle hook. You cannot use `setState()` in your component because Functional Components are plain JavaScript functions,

Use class Components if you need state or need to access lifecycle hook because all lifecycle hooks are coming from the `React.Component` which you extend from in class components.

Phases / Lifecycle of Components

- Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.
- The three phases are:
 - Mounting
 - Updating
 - Unmounting



Phases / Lifecycle of Components



REACT

Mounting means putting elements into the DOM.



Mounting



Updating



UnMounting

The Last phase in the lifecycle is when a component is removed from the DOM



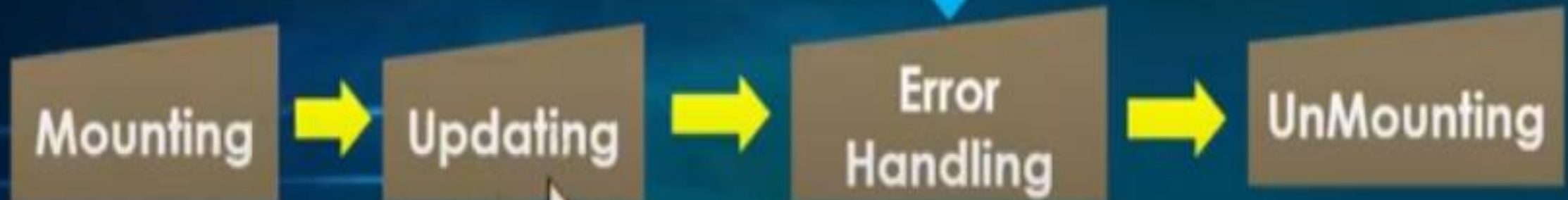
A component is updated whenever there is a change in the component's state or props.

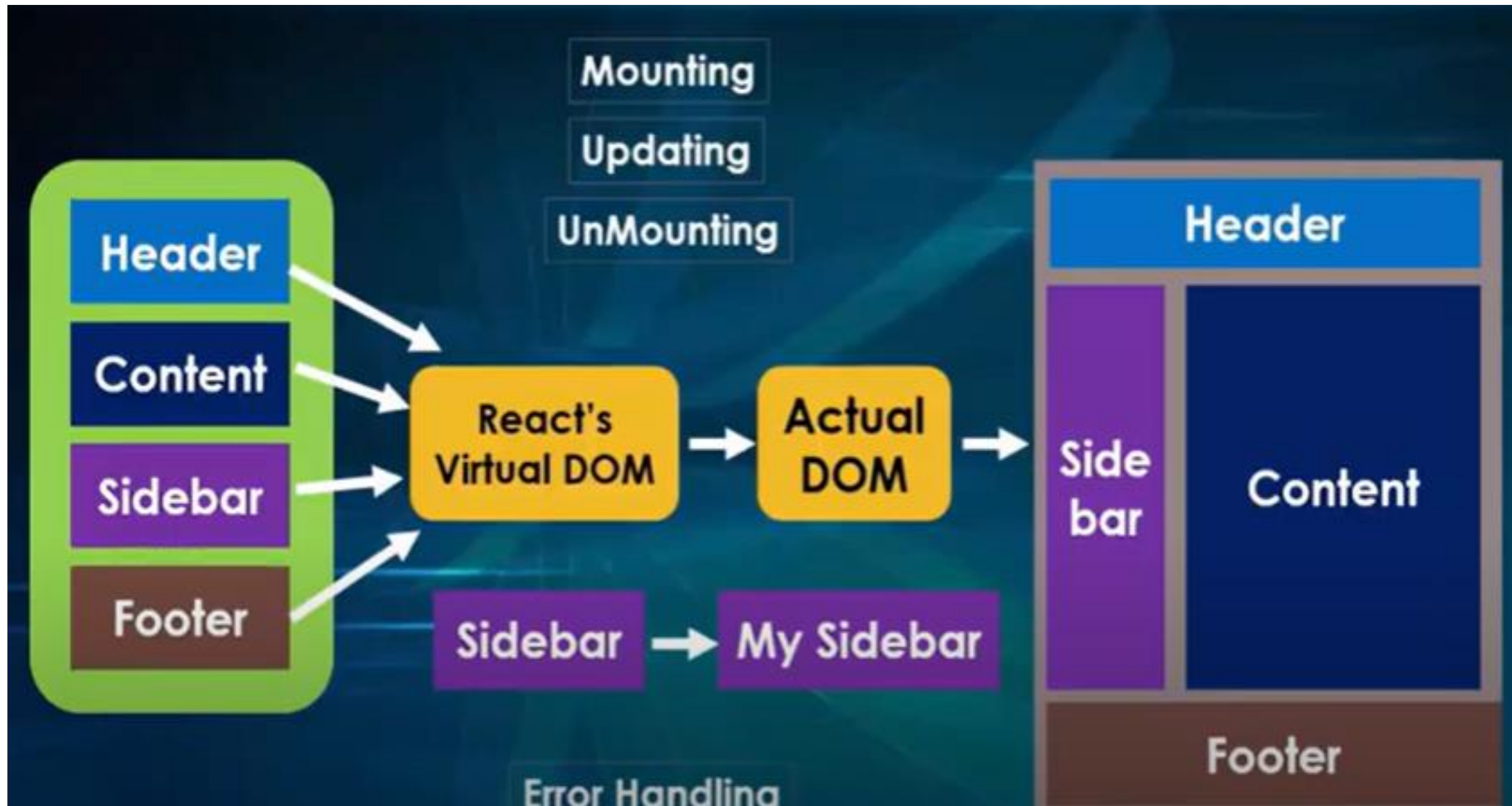


Phases / Lifecycle of Components



Whenever some error occurs during Mounting or updating then this phase Comes into place





React's Virtual DOM

- **Virtual DOM:** React uses Virtual DOM exists which is like a lightweight copy of the actual DOM(a virtual representation of the DOM).
- So for every object that exists in the original DOM, there is an object for that in React Virtual DOM.
- It is exactly the same, but it does not have the power to directly change the layout of the document.



React's Virtual DOM

- Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen.
- So each time there is a change in the state of our application, the virtual DOM gets updated first instead of the real DOM.

- These 3 phases or life cycle only applied to **class components** not on functional component.
- There is another way of applying life cycle on functional components are called **hooks**.

Mounting

- Mounting means putting elements into the DOM.
- In this phase, an instance of a component is being created and inserted into the DOM.
- **React has four built-in methods that gets called, in this order, when mounting a component:**
 1. `constructor()`
 2. `getDerivedStateFromProps()`
 3. `render()`
 4. ~~`componentDidMount()`~~

Mounting

- The **render()** method is required and will always be called, the others are optional and will be called if you define them.

Constructor(props)

- It is a special function that executes when a new component is created.
- *We can use constructor for 2 purposes.*
 1. For initializing state.
 2. For binding the event handlers.
- You cannot use **HTTP requests (APIs)** in the constructor.
- We have to call parent class constructor in our child class constructor by using **super(props)**.

Constructor(props)

- When implementing the constructor for a **React.Component** subclass, we have to call **super(props)** on the first place in the constructor otherwise **this.props** will be undefined in the constructor, which can lead to bugs.

Constructor Syntax

```
• constructor(props) {  
•   super(props)  
•  
•   this.state = {  
•     name: "Adil"  
•   }  
• }
```


getDerivedStateFromProps()

- It is used when the state of the component depends on changes in props over time.
- Lets say, you have a component but the initial state of the component depends on the props being passed to the component, in this scenario you can use this method to set the state.
- This is very **rarely** used method.
- It is invoked right before calling the **render function**, both on the initial mount and on subsequent updates.

getDerivedStateFromProps()

- We cannot use this keyword inside this function, it means we cannot update the state by using this.setState method.
- It should return an object to update the state, or null to update nothing.

Syntax

getDerivedStateFromProps()

```
static getDerivedStateFromProps(props, state){  
  •  
  •  
  }  
  •
```


Render()

- It is the only **required (necessary) method** in the class component, so all other methods are optional.
- In this method, we can read props & state and return JSX.
- In this method, we cannot change the state or interact with the DOM or make ajax calls.
- Children components life cycle methods are also executed.

Render() Syntax

```
render() {  
  return (  
    <div>  
      <h1>Learning Never Ends</h1>  
    </div>  
  )  
}
```


componentDidMount()

- It is invoked immediately after a component is mounted (inserted into the DOM), after the **render()** method.
- Invoked immediately after a component have been rendered to the Dom.
- Invoked immediately after a component and all its children components have been rendered to the Dom.
- We can use **HTTP requests, interaction with the DOM or make ajax calls** in this method.
- This method is executed only **once** in a life cycle of component.



componentDidMount()

- This method is also used for integration with other JavaScript frameworks and other functions with delayed execution like **setTimeout** or **setInterval**.

- ***Syntax:***

- ```
componentDidMount(){

}
```



# Updating

- The next phase in the lifecycle is when a component is updated. A component is updated whenever there is a change in the component's state or props.
- React has **five** built-in methods that gets called, in this order, when a component is **updated**:


1. `getDerivedStateFromProps()`
2. `shouldComponentUpdate()`
3. `render()`
4. `getSnapshotBeforeUpdate()`
5. `componentDidUpdate()`

# Updating

- The **render()** method is required and will always be called, the others are optional and will be called if you define them.



# getDerivedStateFromProps()

- It is used when the state of the component depends on changes in props over time.
- Lets say, you have a component but the initial state of the component depends on the props being passed to the component, in this scenario you can use this method to set the state.
- This is very **rarely** used method. 
- It is invoked right before calling the **render function**, both on the initial mount and on subsequent updates.

# getDerivedStateFromProps()

- We cannot use this keyword inside this function, it means we cannot update the state by using this.setState method
- It should return an object to update the state, or null to update nothing.

## Syntax

### getDerivedStateFromProps()

```
static getDerivedStateFromProps(props, state){
 // ...
}
```



# shouldComponentUpdate()

- Use **shouldComponentUpdate()** to let React Know if a component's output is not affected by the current change in state or props.
- It means should React **re-render or not ?**
- **shouldComponentUpdate()** is called before render method.
- This method return **true** by default.
- Render() method will not be called if **shouldComponentUpdate()** returns false.

# shouldComponentUpdate()

- We cannot use HTTP requests (API calls) in this method and also we cannot call the **setState()** Method.
- This is Rarely used method according to React documentation.

## Syntax

### shouldComponentUpdate()

```
shouldComponentUpdate (nextProps, nextState) {

}
```



# getSnapshotBeforeUpdate()

- This method is called right before the Virtual DOM is about to make changes to the Actual DOM (**before DOM is updated**).
- It allows our components to capture the current values or some information from the DOM (**eg: Scroll Position**) before it is potentially changed.
- This method will return NULL or return a value.

# getSnapshotBeforeUpdate()

- Any **returned value by this function** will be passed as **third parameter** to last function **componentDidUpdate()**.
- This is also **rarely used method** according to React documentation.

## Syntax

### getSnapshotBeforeUpdate()

```
getSnapshotBeforeUpdate (prevProps, prevState) {

}
```



# componentDidUpdate()

- This method is invoked right after updating occurs.
- This method is not called for the initial render.
- This method is called after the render is completed in the re-render cycles.
- This method is guaranteed to call only once in a life cycle.
- You can make Ajax Calls in this method.
- This method will not be called if **shouldComponentUpdate()** returns false.



# Syntax componentWillUpdate()

```
• componentWillUpdate (prevProps, prevState, snapshot) {

 }
• }
```



# Unmounting

- The next phase in the lifecycle is when a component is removed from the DOM, or unmounting as React likes to call it.
- React has only **one** built-in method that gets called when a component is unmounted:
- **`componentWillUnmount()`**

# UnMounting

- **ComponentWillUnmount** is the only method that executes in **unmount phase**.
- Just before the component gets removed from actual DOM, this method gets called.
- Along with removal of this component from DOM tree, **all children of this component** also gets removed automatically.
- The **componentWillUnmount()** method allows us to execute the React code when the component gets destroyed or unmounted from the DOM (Document Object Model)



- This method is called during the Unmounting phase of the React Life-cycle **i.e before the component gets unmounted.**
- All the cleanups such as **invalidating timers, canceling network requests, or cleaning up any subscriptions** that were created in **componentDidMount()** should be coded in the **componentWillUnmount()** method block.
- Cleanup activities helps in **improving performances, memory leakages and maintain security.**
- You should not call **setState()** in **componentWillUnmount** because the component will never be re-rendered.

## Initialization

## Mounting

## Updation

## Unmounting

Set Props and  
Initial State of  
the component  
in the constructor

**render()**

componentDidMount()

For Props  
getDerivedStateFromProps()

For State  
setState()

shouldComponentUpdate()

getSnapshotBeforeUpdate()

**render()**

ComponentDidUpdate()

ComponentWillUnmount()

React Methods



ReactDOM Methods



# Props

- Props stand for "**Properties**." They are **read-only** components.
- It is an object which stores the value of attributes of a tag and work similar to the HTML attributes.
- It gives a way to pass data from one component to other components.
- It is similar to function arguments. Props are passed to the component in the same way as arguments passed in a function.
- Props are **immutable** so we cannot modify the props from inside the component. Inside the components, we can add attributes called props.
- When we need immutable data in the component, we have to add props to **ReactDOM.render()** method in the **main.js** file of your ReactJS project and used it inside the component in which you need.

# PROPS In React

- Props stands for **properties**.
- Props are **arguments** passed into **React components**.
- Props are passed to components via **HTML attributes**.
- React Props are like **function arguments in JavaScript** and **attributes in HTML**.
- To send props into a component, use the same syntax as HTML attributes.
- Props also act like an **object**.



# PROPS In React

- If you have a variable to send, and not a string. you just put the **variable name inside curly brackets**.
- **Note:** React Props are **read-only (immutable)** You will get an error if you try to change their value.
- Props can be used as **Children Props**.

```
function ShowMessage(msg)
{
 document.write(msg);
}
```

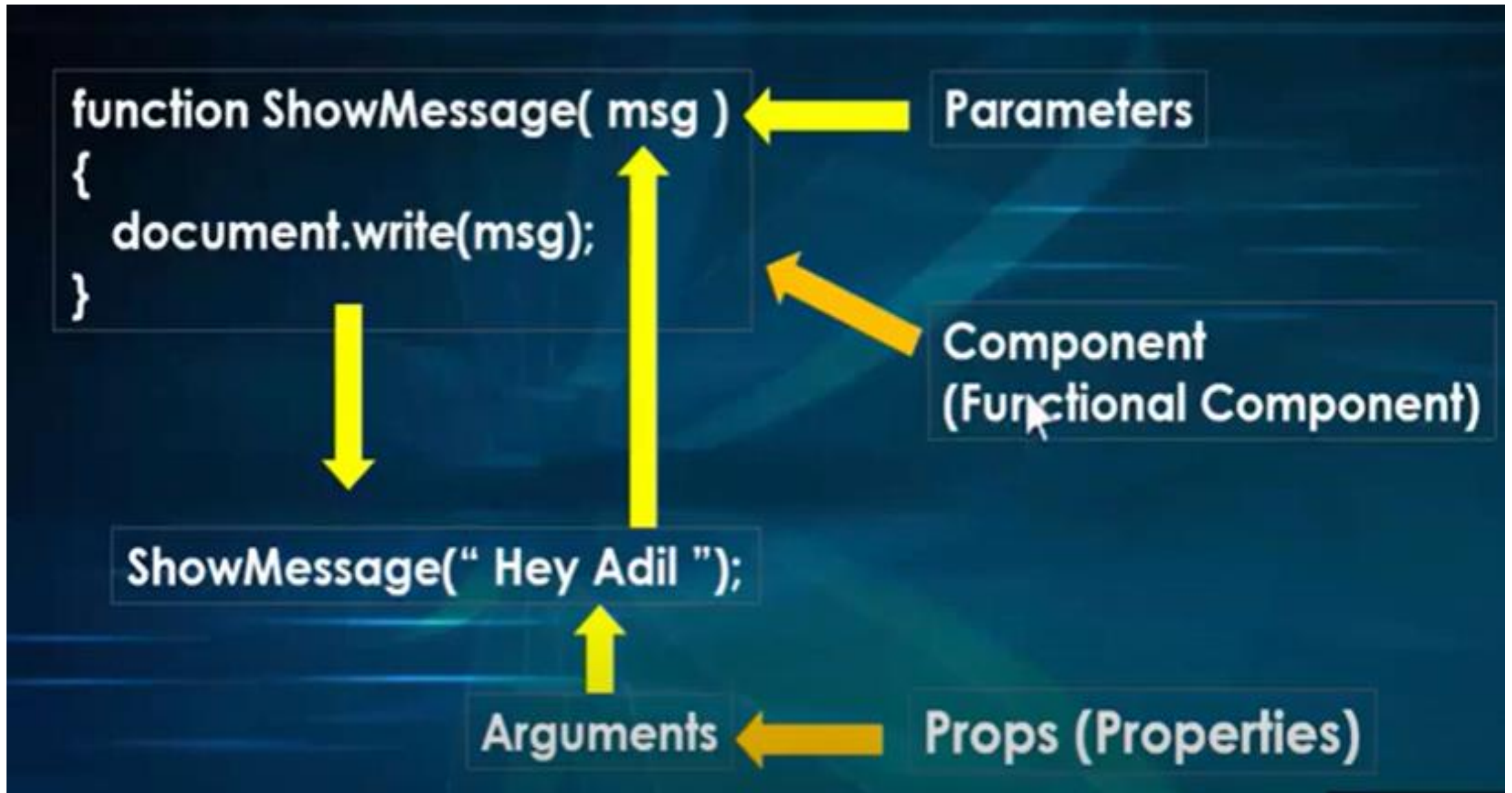
Parameters

Component  
(Functional Component)

```
ShowMessage(" Hey Adil ");
```

Arguments

Props (Properties)



properties

PROPS



Inside src folder

React  
Component

properties

PROPS

arguments



Inside src folder

HelloMessage.js

Functional  
Components

Hello Adil  
Hello Kumar  
Hello Zain





## What is “State” ?

- State is a built-in object of the React Components.
- The state object is used to store all the data that belongs to that particular component only.
- The state is only accessible inside the component it belongs to.
- State is mutable, it can be changed as per the need.
- Whenever the state object changes the component is re-rendered.
- To use state inside a class component “**this.state**” is used.
- To use state inside a functional component “**useState**” hook is used.

# What Is State in ReactJS?

- The state is a built-in React object that is used to contain data or information about the component.

properties

PROPS



Inside src folder

React  
Component

immutable

Un-changeable

props



properties

PROPS

Function parameters

Inside src folder

React  
Component

Index.js / App.js ← `<Student name="Adil"/>`

State Created &  
Managed within  
the component

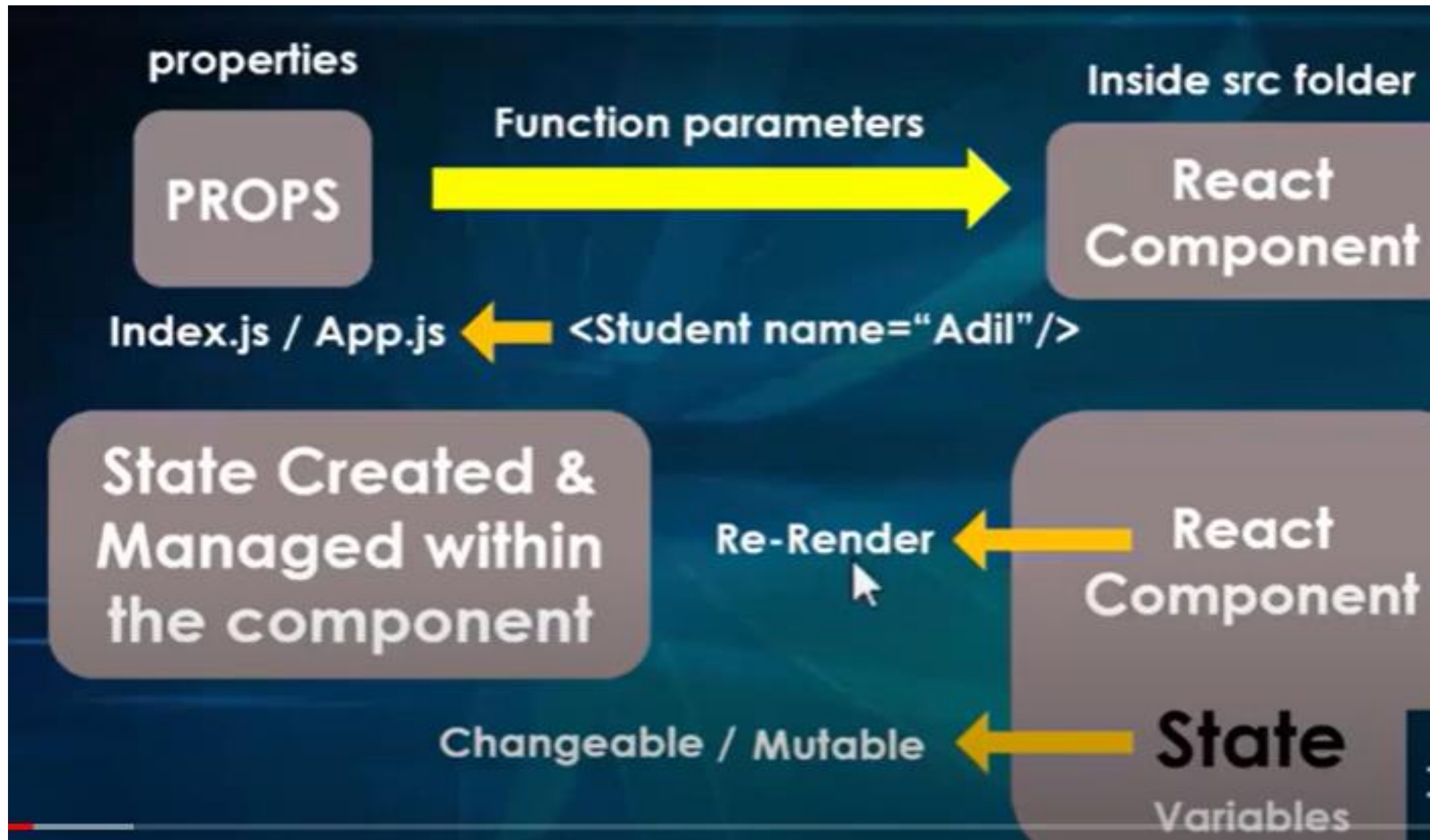
Re-Render

React  
Component

Changeable / Mutable

State

Variables



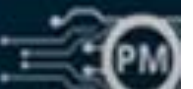


# What Is State in ReactJS?

- A component's state **can change** over time, whenever it changes, the component **re-renders**.
- The change in state can happen as a response to user action or system-generated events and these changes determine the behavior of the component and how it will render.
- **Constructor** is used to initialize the **object's state**.

# Important Points Regarding State

- State is similar to props, but it is private and fully controlled by the component.
- We can create state only in class component.
- It is possible to update the state.
- A state can be modified based on user action or network changes
- Every time the state of an object changes, React re-renders the component to the browser





# Important Points Regarding State

- The state object is initialized in the constructor
- The state object can store multiple properties
- We can update state on a button click
- `this.setState()` is used to change the value of the state object.
- We can set props data to state.

# There Are 2 Ways Of Initialize State

- 1. With Constructor
- 2. Without Constructor



# Difference B/W Props And State

## PROPS

- Props get passed to the component.
- Function Parameters.
- Props are immutable / unchangeable

## STATE

- State is created and managed within the component.
- Variables.
- State is mutable / changeable.



# Difference between State and Props

| SN | Props                                                                               | State                                                                |
|----|-------------------------------------------------------------------------------------|----------------------------------------------------------------------|
| 1. | Props are read-only.                                                                | State changes can be asynchronous.                                   |
| 2. | Props are immutable.                                                                | State is mutable.                                                    |
| 3. | Props allow you to pass data from one component to other components as an argument. | State holds information about the components.                        |
| 4. | Props can be accessed by the child component.                                       | State cannot be accessed by child components.                        |
| 5. | Props are used to communicate between components.                                   | States can be used for rendering dynamic changes with the component. |
| 6. | Stateless component can have Props.                                                 | Stateless components cannot have State.                              |
| 7. | Props make components reusable.                                                     | State cannot make components reusable.                               |
| 8. | Props are external and controlled by whatever renders the component.                | The State is internal and controlled by the React Component itself.  |

# React Router

- React Router is a standard **library** for **routing** in React.

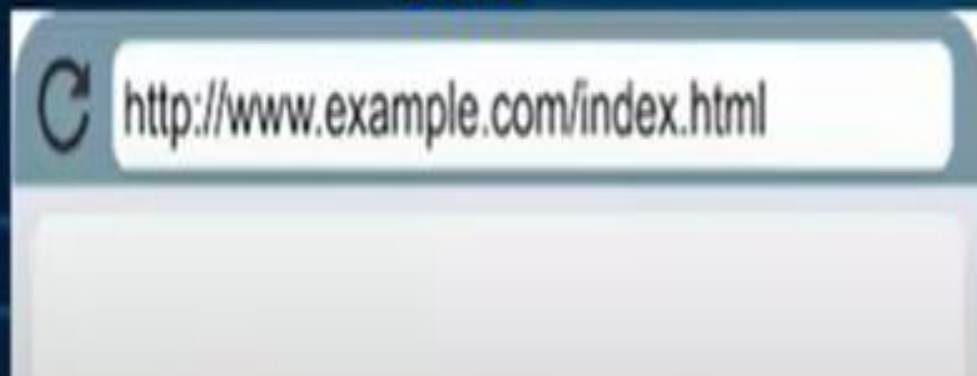


REACT

# What is Routing ?

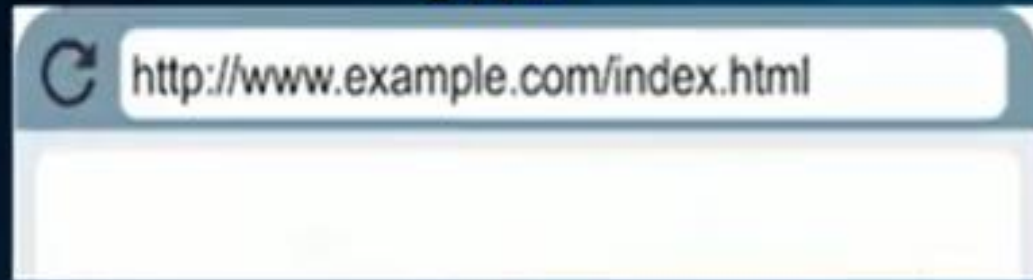
- **Routing** is the **mechanism** by which requests (as specified by a **URL** and HTTP method) are routed to the **code** that **handles** them.

**URL**



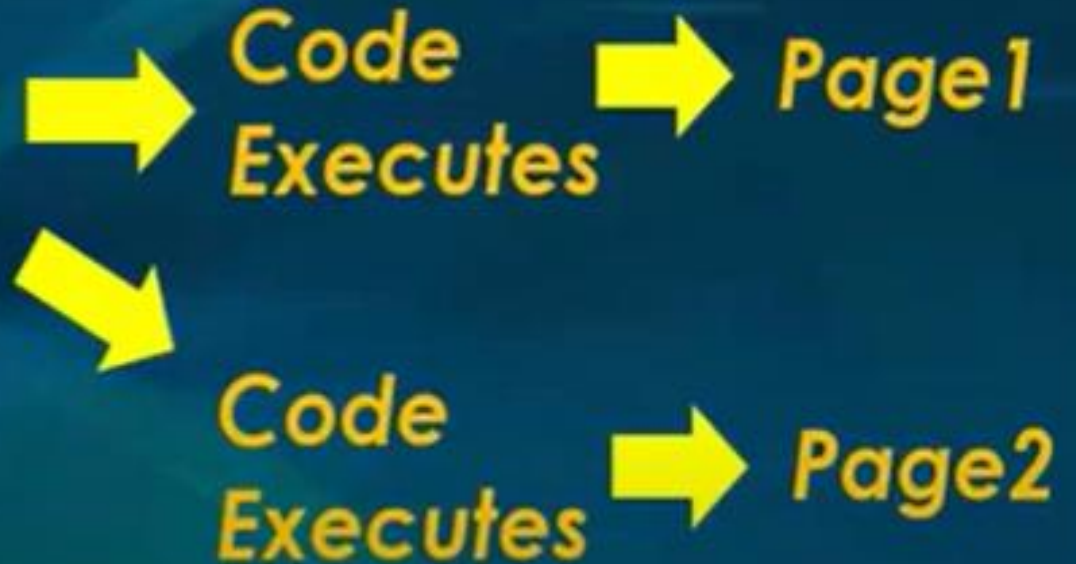
# What is Routing ?

**URL**



**Without  
Page Reload**

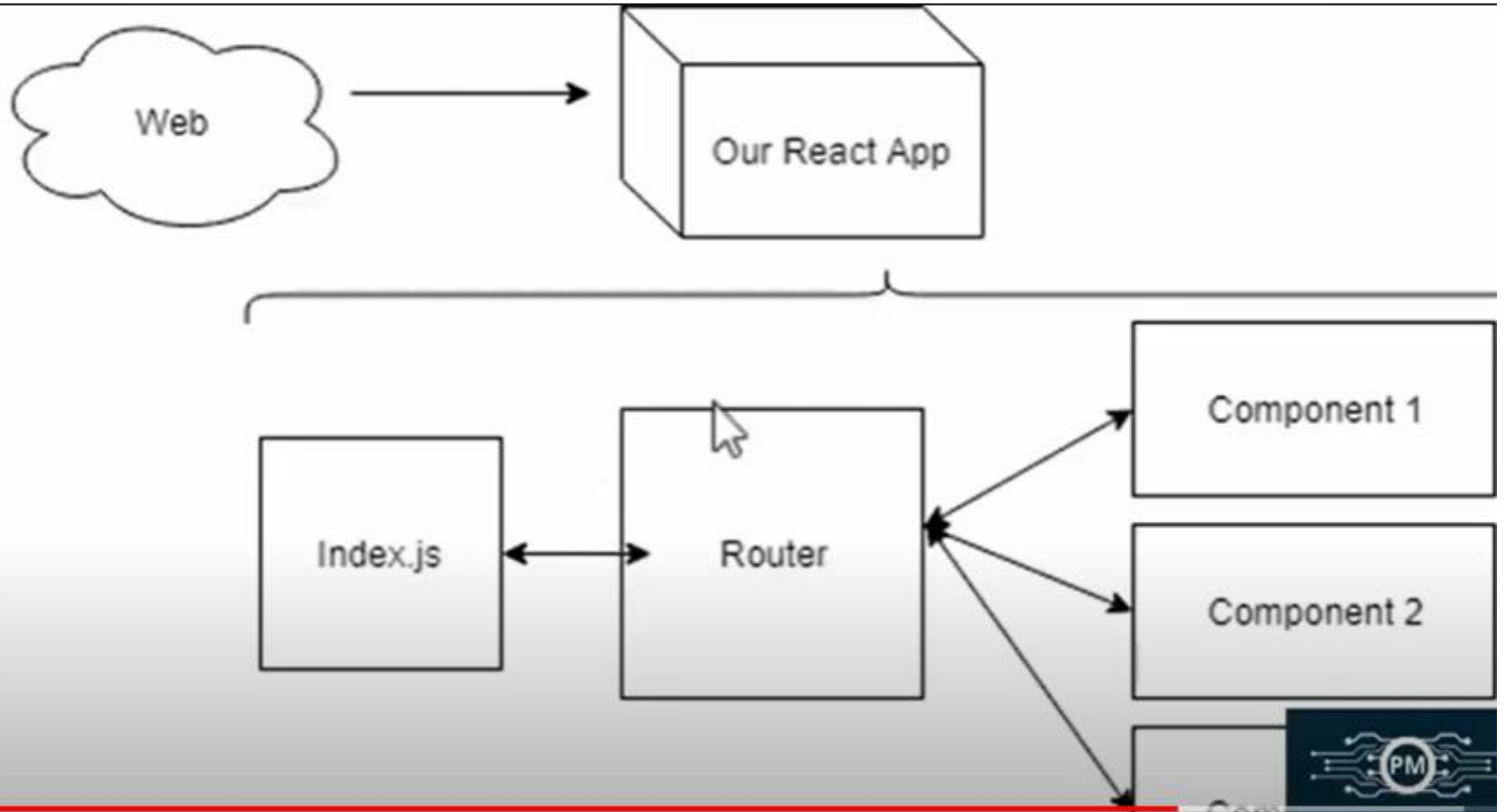
**react-router-dom**



REACT







# React Routing

- It enables the **navigation** among views of **various components** in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.
- React Router is **a fully-featured client and server-side routing library** for React, a JavaScript library for building user interfaces.
- **React Router runs anywhere React runs;** on the web, on the server with **node.js**, and **on React Native**



# React Routing

- **Create React App** doesn't include **page routing** by default.
- **React Router** is the most popular **solution**.
- It provides **unique URLs for different components** in the app and **makes UI easily shareable with other users**.

# Types of Web Applications

The current web application development scene can be divided into two approaches. They are:

- Multi-Page Application
- Single Page Application

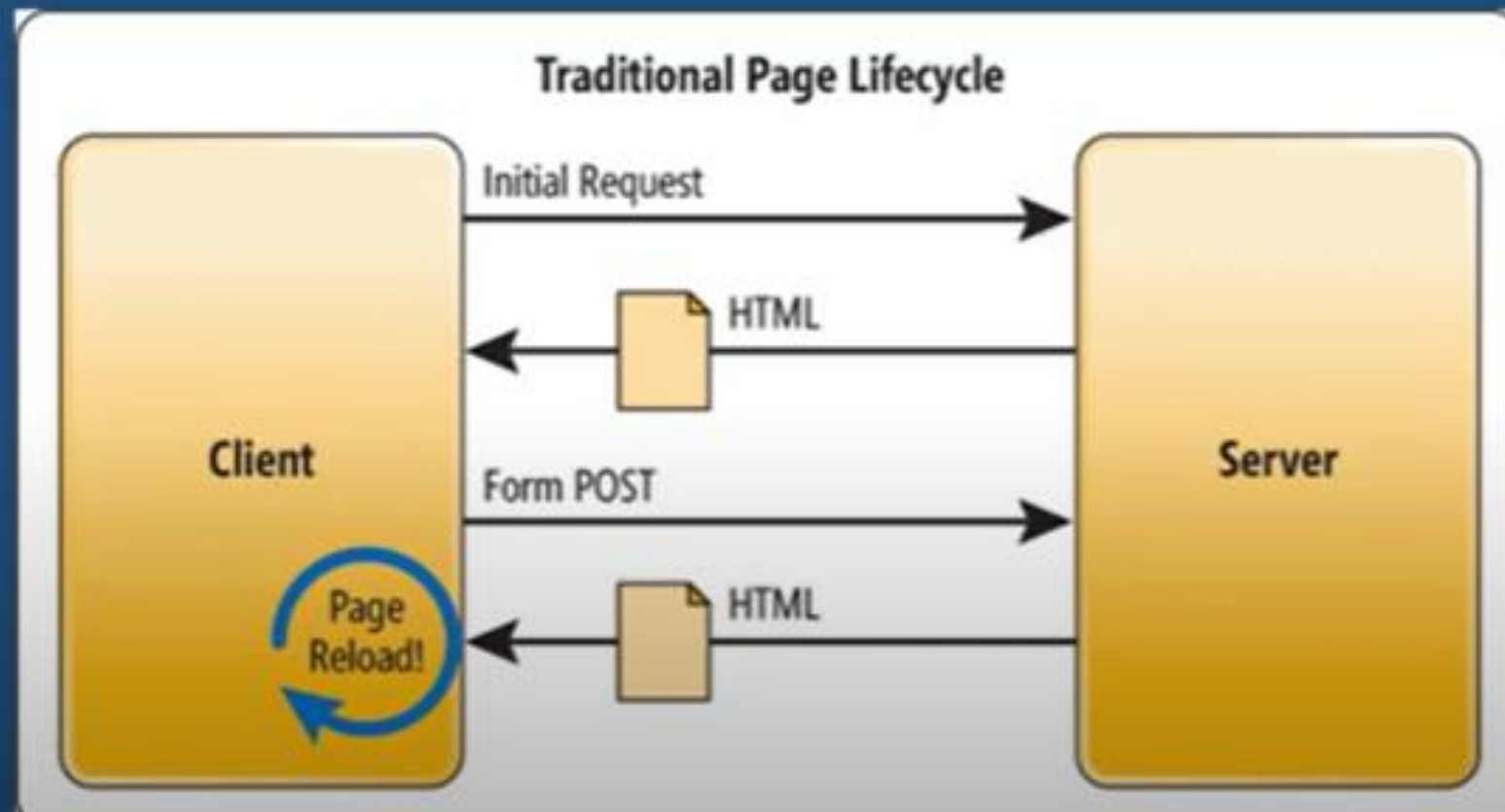


# What is Multi Page Application?

- ✓ Multi-page applications are the traditional web applications that reload the entire page and displays the new one when a user interacts with the web app.
- ✓ Each time when a data is exchanged back and forth, a new page is requested from the server to display in the web browser. This process takes time to generate the pages on the server, sending it to a client and displaying in the browser which may affect the user experience.
- ✓ However, AJAX had made it possible to render just a particular component of the application, but this makes the development process even more difficult and complex.

# Multi Page Application

- Every time the app calls the server, the server renders a new HTML page.



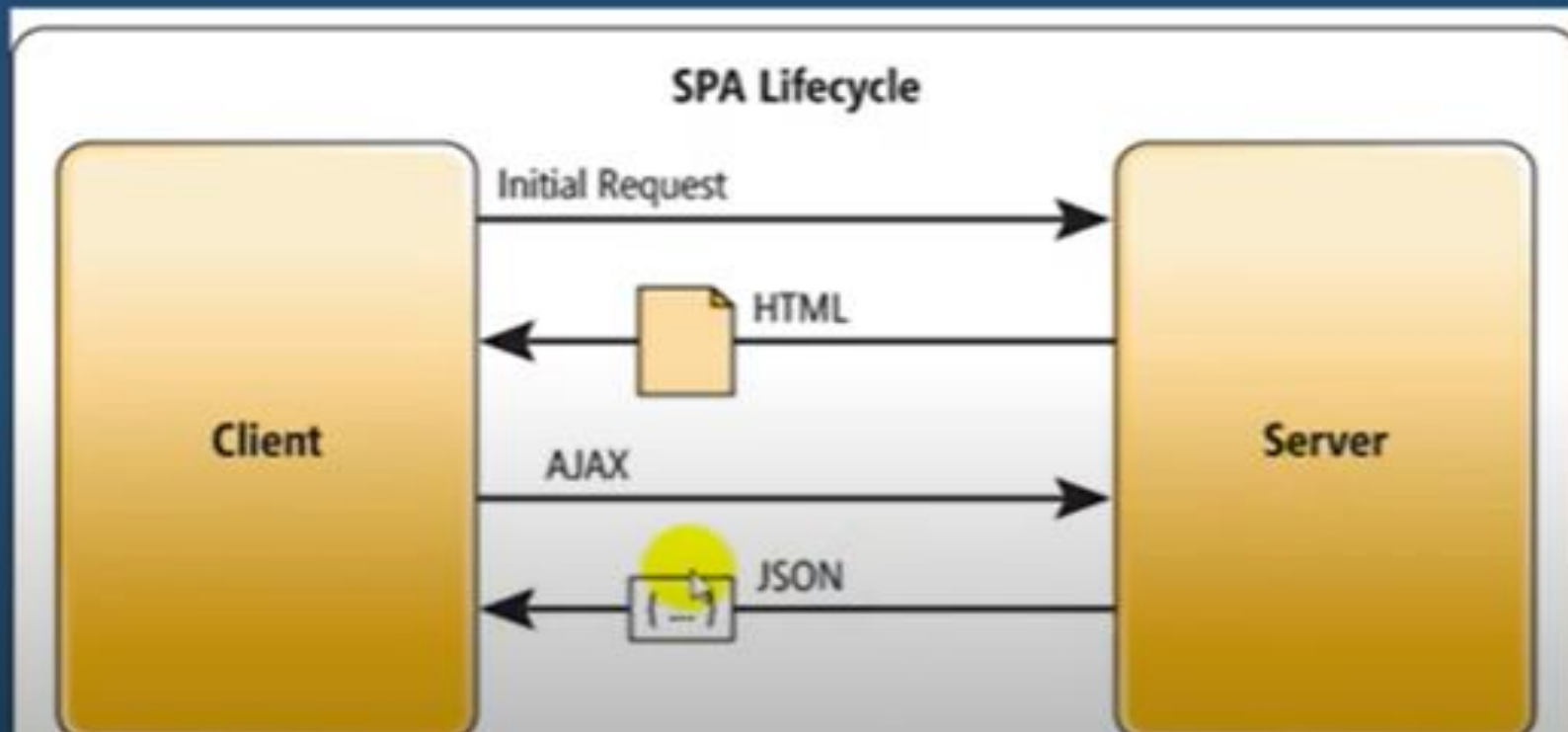
# What is a Single Page Application?

- ✓ In traditional web applications, when the client sends a request, the page server sends a response. Every time the page reload happens there are many pages and each one requires a Form Post.
- ✓ SPAs are faster than traditional web applications because they execute the logic in the web browser itself rather than on the server. And after the initial page load, only data is sent back and forth instead of the entire HTML that reduces the bandwidth.



# SPAs Life Cycle

- After the first page loads, all interaction with the server happens through AJAX calls.





# Advantage of SPA

- **1. Quick Loading Time**
- With the SPA approach, your full page loads quicker than traditional web applications, as it only has to load a page at the first request. On the other hand, traditional web apps have to load pages at every request, taking more time.
- **2. Seamless User Experience**
- SPAs deliver an experience like a desktop or mobile app. Users do not have to watch a new page load, as only the content changes, not the page, making the experience enjoyable.

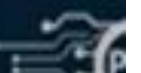
- **3. Ease in Building Feature-rich Apps**
- SPA application makes adding advanced features to a web application easy. For example, it is easier to build a content-editing web app with real-time analysis using SPA development. Doing this with a traditional web app requires a total page reload to perform content analysis.
- **4. Uses Less Bandwidth**
- It is no surprise that SPAs consume less bandwidth since they only load web pages once. Besides that, they can also do well in areas with a slow internet connection. Hence, it is convenient for everyone, regardless of internet speed.

# Disadvantage of SPA

- **1. Doesn't Perform Well With SEO**
- **2. Uses a Lot of Browser Resources**
- **3. Security Issues**

# React Forms

- Forms are an **integral part** of any modern **web application**.
- It allows the **users to interact with the application** as well as **gather information from the users**.
- Forms can perform many tasks that depend on the nature of your business requirements and logic such as **authentication of the user, adding user, searching, filtering, booking, ordering, etc.**
- A form can contain **text fields, buttons, checkbox, radio button, etc**





# React Forms

- The form has the default HTML form behavior of browsing to a new page when the user submits the form.
- If you want this behavior in React, it just works. But in most cases, it's convenient to have a JavaScript function that handles the submission of the form and has access to the data that the user entered into the form.
- The standard way to achieve this is with a technique called “**controlled components**”.
- HTML form elements work a bit differently from other DOM elements in React, because form elements naturally keep some internal state.

# React Forms

- In React, state is typically kept in the state property of components, and only updated with **setState()**.
- We can use **useState()** Hook in functional components with react forms.



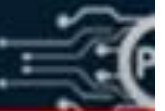
# Handling Forms

- Handling forms is about how you handle the data when it **changes value** or **gets submitted**.
- 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.

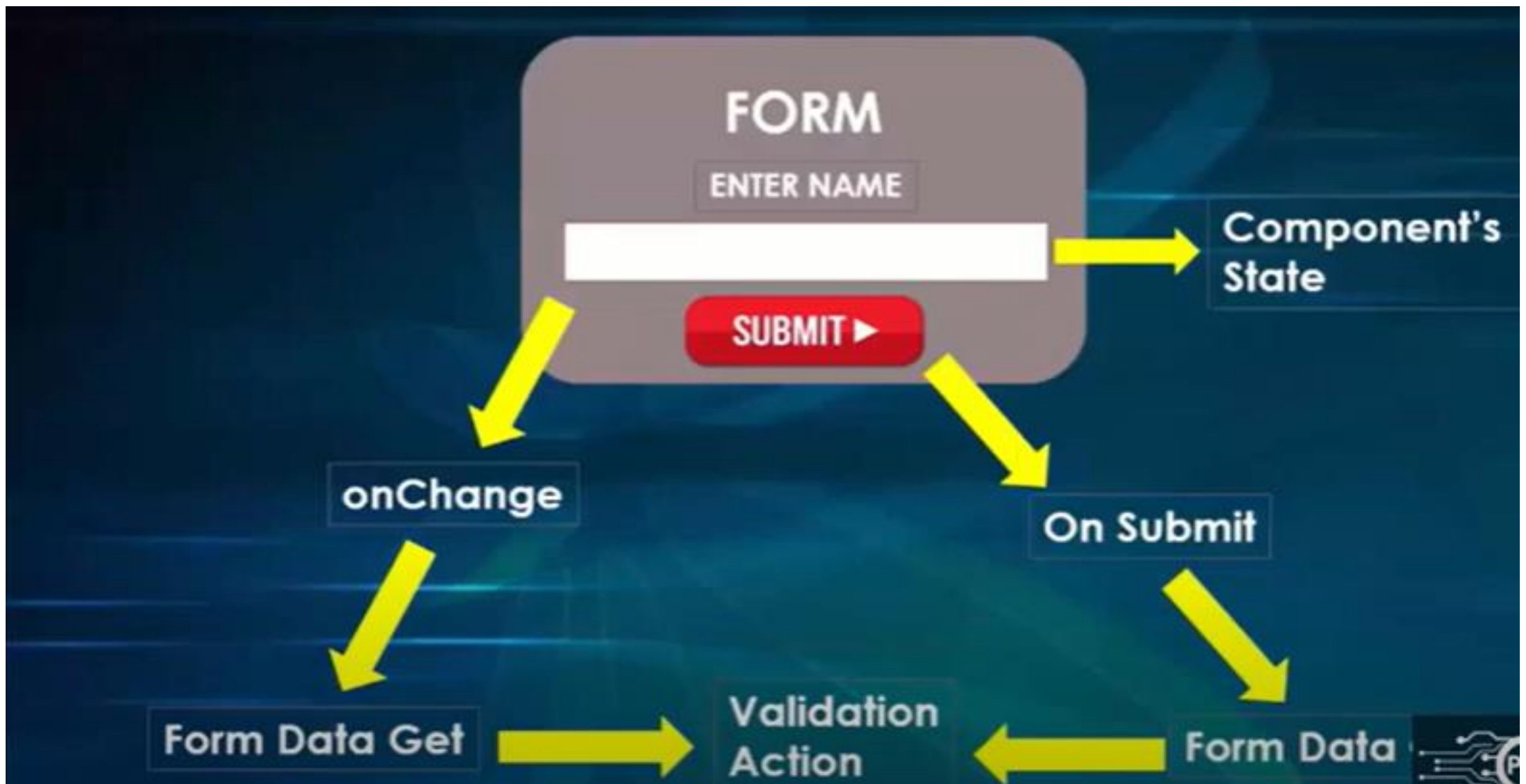


# Creating Form

- React offers a stateful, reactive approach to build a form.
- The component rather than the DOM usually handles the React form.
- In React, the form is usually implemented by using controlled components.
- **There are mainly two types of form input in React.**
  - **Controlled component**
  - **Uncontrolled component**







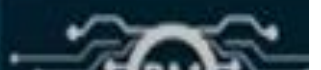
# Controlled Component

- In HTML, form elements typically **maintain their own state** and **update it according to the user input**.
- In the **controlled component**, the input form element is handled by the **component** rather than the **DOM**.
- Here, the **mutable state** is kept in the **state** property and will be updated only with **setState()** method.



# Controlled Component

- **Controlled components** have functions that govern the data passing into them on every **onChange** event, rather than grabbing the data only once, e.g., when you click a submit button.
- This data is then saved to state and updated with **setState() method or useState() Hook**. This makes component have better control over the form elements and data.



# Uncontrolled Component

In a controlled component, form data is handled by a React component. The alternative is uncontrolled components, where form data is handled by the DOM itself.

To write an uncontrolled component, instead of writing an event handler for every state update, you can use a *ref* to get form values from the DOM.

## When Use Uncontrolled Component-

You do not need to write an event handler for every way your data can change and pipe all of the input state through a React component.

Converting a preexisting codebase to React, or integrating a React application with a non-React library.