TESTYANTRA

SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

REACT JS

EXPERIENTIAL
learning factory

Basics Introduction

- **Framework** : A framework, or software framework, is a platform for developing software applications. It provides a foundation on which software developers can build programs for a specific platform.
- **Library**: A Library refers to a collection of files, programs, routines, scripts, or functions that can be referenced in the programming code.
- **Applications:**

1. **Social Media** : Facebook, Instagram, Twitter
2. **Content Delivery** : Netflix, Spotify
3. **Developer tools**: React Native, Next JS

Introduction to React JS

- A JavaScript library for building user interfaces.
- React came into existence in 2011 and was only used by facebook later in 2013 facebook made it open source.
- Facebook developer Jordan Walke creator of React library.
- React is the library for creating views.
- ReactDOM is the library used to actually render the React Elements in the Virtual DOM using render() method.

What is React ?

- React is a front-end JavaScript library developed by Facebook in 2011.
- In Facebook, it was used for News Feed Feature.
- It follows the component based approach which helps in building reusable UI components.
- It is used for developing complex and interactive web and mobile UI.
- Even though it was open-sourced only in 2015, it has one of the largest communities supporting it.

- Current Version of React JS V17.0.1

Features of React & Why it is Used

Features of React :

- It uses the virtual DOM instead of the real DOM.
- It uses server-side rendering.
- It follows uni-directional data flow or data binding.

Why React is Used ?

- Easy to learn nature
- Reusable Components
- Can be used for Mobile Apps
- Increase performance
- Unidirectional Data flow
- Dedicated tools for easy debugging

Advantages & Limitations

Advantages:

- It increases the application's performance
- It can be conveniently used on the client as well as server side
- Because of JSX, code's readability increases
- React is easy to integrate with other frameworks like Meteor, Angular, etc
- Using React, writing UI test cases become extremely easy

Limitations :

- React is just a library, not a full-blown framework
- Its library is very large and takes time to understand
- It can be little difficult for the novice programmers to understand
- Coding gets complex as it uses inline templating and JSX

Concepts to be known

- JavaScript
- ES 6 Syntax :

Class based and Functional components, Arrow functions, Promises, Template Literals, Destructuring, Async / Await, Imports and Exports, Spread Operators, Methods like: Map, Reduce, Filter.

- Node JS and npm
- **Online Editor**: Code Sand box, Code pen etc..
- **React Developer Tools** : React Developer Tools is a Chrome DevTools extension for the open-source React JavaScript library. It allows you to inspect the React component hierarchies in the Chrome Developer Tools. It can be added for Google, Firefox etc..

- **SPA(Single Page Applications) : -**

  In an SPA, the browser initially loads one HTML document. As users navigate through the site, they actually stay on the same page. JavaScript destroys and creates a new user interface as the user interacts with the application.

- The DOM API is a collection of objects that JavaScript can use to interact with the browser to modify the DOM.

- Managing DOM changes with JavaScript efficiently can become very complicated and time-consuming. The solution is React.

How does React Work ?

- React creates a Virtual DOM in memory.
- Next step, Instead of Changing the Browser's DOM directly, React Creates a Virtual DOM in memory.
- All necessary changes are done in the Virtual DOM, Before making the changes in Browser DOM(Real DOM).
- Finally, React only changes, what needs to be changed.

So what is this Virtual DOM and Real DOM……?

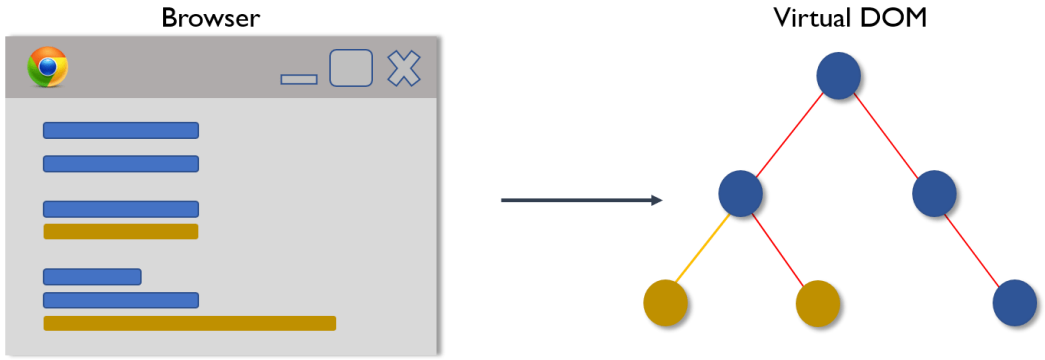
The Virtual DOM

- React is a library that is designed to update the browser DOM for us.
- With React, we do not interact with the DOM API directly. Instead, we interact with a *virtual DOM*, or set of instructions that React will use to construct the UI and interact with the browser.
- The virtual DOM is made up of React elements.
- We make changes to the Virtual DOM (JavaScript object), and React renders those changes for us using the DOM API as efficiently as possible.
- The browser DOM is made up of DOM elements. Similarly, the Virtual DOM is made up of React elements.

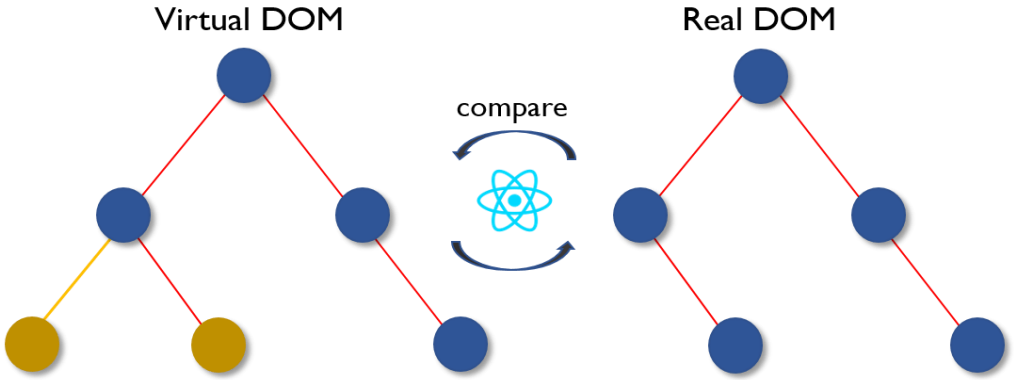
Virtual DOM Working continued..

This Virtual DOM works in three simple steps.

1.Whenever any underlying data changes, the entire UI is re-rendered in Virtual DOM representation.

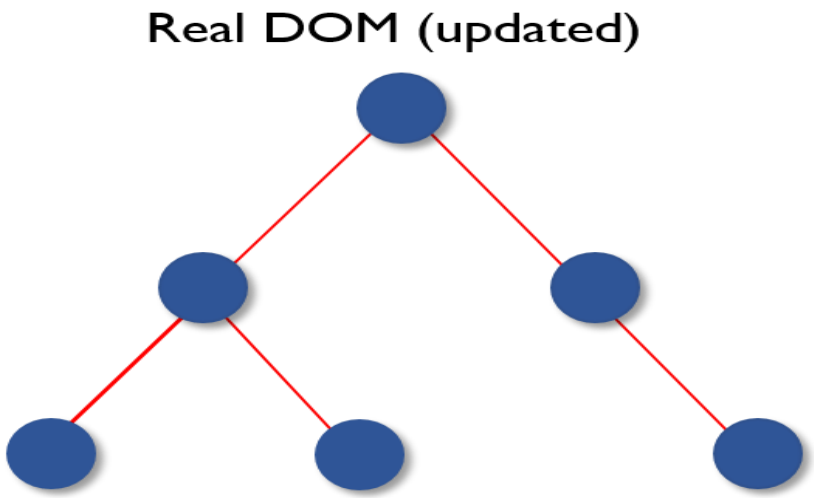


2. Then the difference between the previous DOM representation and the new one is calculated



Virtual DOM Working

3. Once the calculations are done, the real DOM will be updated with only the things that have actually changed.



Real DOM v/s Virtual DOM

Real DOM	Virtual DOM
1. It updates slow.	1. It updates faster.
2. Can directly update HTML.	2. Can't directly update HTML.
3. Creates a new DOM if element updates.	3. Updates the JSX if element updates.
4. DOM manipulation is very expensive.	4. DOM manipulation is very easy.
5. Too much of memory wastage.	5. No memory wastage.

Install Node JS and npm

1. npm i –g create-react-app
2. create-react-app < project name >
3. cd < project name >
4. npm start

(OR)

1. npx create-react-app my-app
2. cd my-app
3. npm start

Official Website: reactjs.org

Host name & Port No: localhost: 3000

Adding React in HTML

1. Create a React container in html page.

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

2. Adding the Scripts for html page.

```
<script src = "react@16/development"></script>
<script src = "react-dom@16/development"></script>
```

3. Adding the babel

```
<script src = "babel-standalone@6.15.0/babel.min.js"></script>
```

4. Create a React Element using the JSX

```
const ele = <h1> Hello World </h1>
ReactDOM.render(ele, document.getElementById('root'));
```

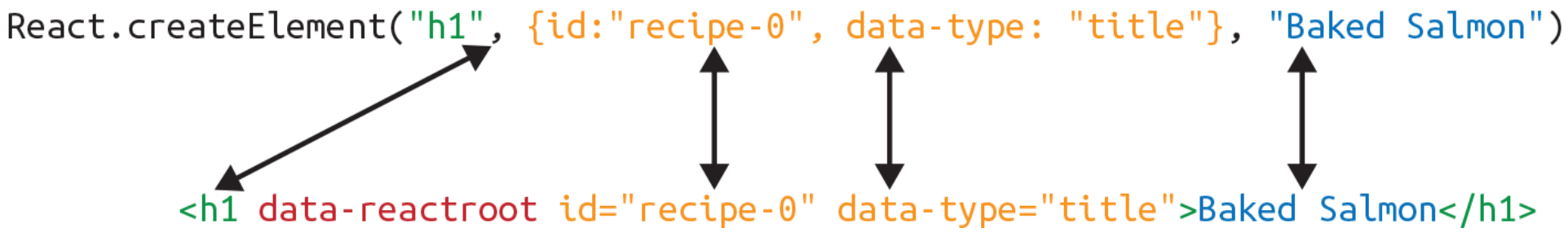
React Example


```
<!DOCTYPE html>
        <html>
            <head>
                <meta charset="utf-8">
                <title>Pure React Sample</title>
            </head>
            <body>
                <!-- Target container -->
                <div class="react-container"></div>
                <!-- React library & ReactDOM-->
                 <script src="https://unpkg.com/react@15.4.2/dist/react.js" > </script>
                 <script src="https://unpkg.com/react-dom@15.4.2/dist/react-dom.js"> </script>
                <script>
                        // Pure React and JavaScript code
                        const item = React.createElement("h1", null, "Baked Salmon")
                        ReactDOM.render( item, document.getElementById('react-container'))
                </script>
             </body>
        </html>
```

React Element to DOM Element

TESTYANTRA SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

EXPERIENTIAL learning factory



- data-reactroot will always appear as an attribute of the root element of your React component.

```
{
$$typeof: Symbol( React.element ),
"type": "h1",
"key": null,
"ref": null,
"props": {"children": "Baked Salmon"},
"_owner": null,
"_store": {}
}
```

ReactDOM

- React renders the HTML on the webpage using a function called as ReactDOM.render()
- The function takes two arguments HTML Code & HTML Element
- ReactDOM renders React elements in to the Virtual DOM.
- ReactDOM is where we will find the render method.
- const dish = React.createElement('h1',{id:'recipe-0',data-type:'title'},'Baked Salmon')
- ReactDOM.render (dish, document.getElementById('react-container'));
- *React added the h1 element to the target: react-container.*

```
<body>
<div id="react-container">
<h1>Baked Salmon</h1>
</div>
</body>
```

JSX(JavaScript XML)

JavaScript HTML


```
const element = <h1>Hello, world! </h1>;
```

- This tag syntax is neither a string nor HTML.
- It is called JSX, and it is a syntax extension to JavaScript.

```
const name = 'John';
const element = <h1>Hello, {name} </h1>
ReactDOM.render(element, document.getElementById('root'));
```

- Since JSX is closer to JavaScript than to HTML, React DOM uses camelCase property naming convention instead of HTML attribute names.

  Example : class becomes className in JSX and tabindex becomes tabIndex.

JSX Rules

1. Can put any valid JavaScript expression inside curly braces { } in JSX.

   For example:- 2 + 2. <h1>Hello, {2+2} </h1>

   const element =

2. Inserting a large block in HTML :

   For example:- const myList = (
   HTML
   CSS
)

3. Can an One top level Element.

   For example:-
   HTML
   CSS
)

4. Elements must be closed.

   For example:- const ele = < input type="text" />

Babel

- Babel is a JAVASCRIPT COMPILER. basically, it's a tool. which helps to compile the code.

- Babel compiles JSX down to React.createElement() calls.

- These two examples are identical :

```
const element = (
                <h1 className="greeting">
                        Hello, world!
                </h1>
                );

const element = React.createElement(
                'h1',
                { className : "greeting" },
                'Hello, world!'
                );
```

Web Pack

- webpack is a *static module bundler* for modern JavaScript applications
- webpack does not require a configuration file to bundle your project.

Note:

Our React JS Project Installation will add:

1. Third Party Libraries.
2. Development Server. (i.e. Lightweight) = webpak-dev-server
3. Webpack (for bundling the Files)
4. Babel (for Compiler)

React.createClass()

DEPRECATED in react16

```
const IngredientsList = React.createClass({
render() {
return React.createElement("ul", {"className": "ingredients"},
React.createElement("li", null, "1 lb Salmon"),
React.createElement("li", null, "1 cup Pine Nuts"),
React.createElement("li", null, "2 cups Butter Lettuce"),
React.createElement("li", null, "1 Yellow Squash"),
React.createElement("li", null, "1/2 cup Olive Oil"),
React.createElement("li", null, "3 cloves of Garlic")
);
}
});
const list = React.createElement(IngredientsList, null, null)
console.log(list)
ReactDOM.render(
list,
document.getElementById('react-container')
);
```

React.Component


```
class IngredientsList extends React.Component {

render() {

return React.createElement("ul", {className: "ingredients"},
            this.props.items.map( function(ingredient, i) {
            return React.createElement("li", { key: i }, ingredient)
 } )
)
}
}
const items = [
                    "1 lb Cool", "1 cup Pine Nuts", "2 cups Butter Lettuce",
                    "1 Yellow Squash", "1/2 cup Olive Oil", "3 cloves of Garlic"
                ]

const list = React.createElement(IngredientsList,{items});
ReactDOM.render(list ,document.getElementById('react-container'))
```

React.Component, an abstract class

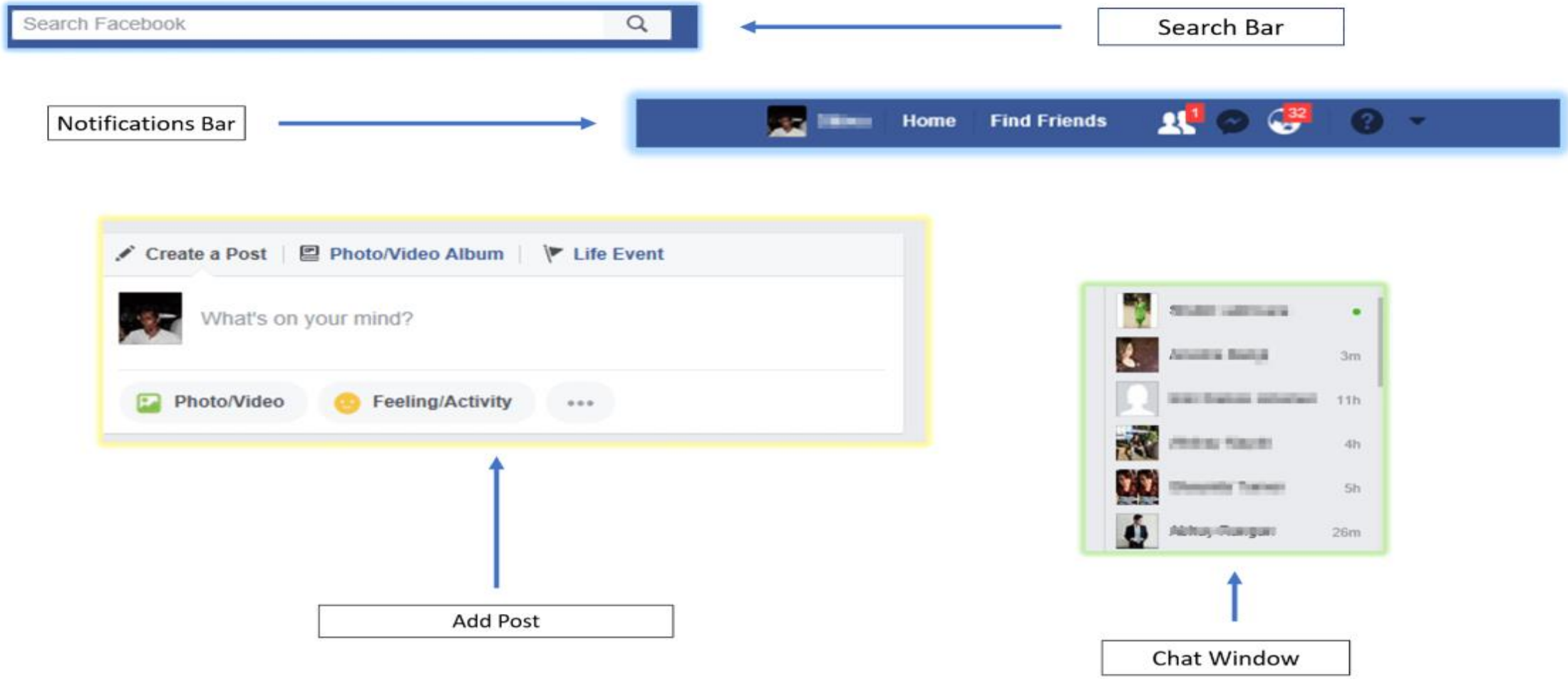
Elements

- Elements are the smallest building blocks of React apps.
- React elements are plain object, and are cheap to create.
- React DOM takes care of updating the DOM to match the React elements.
- React Only Updates What's Necessary.
- React DOM compares the element and its children to the previous one, and only applies the DOM updates necessary to bring the DOM to the desired state.

React Components

Let see a ex : Facebook Components

- Search Bar, Add Post , Notifications Bar, Feed Updates, Profile Info, Chat Window



Components

- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- The terms 'component', "React Component" & "component instance" all refer same.
- Components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.
- Collection of React Elements to build User Interfaces. It's a function which returns some UI.
- Components are Independent and Reusable bits of Code.
- Components enables us to keep the View and Logic Separate.
- Note: **Components should be Capitalized always**.

Types of Components

- Components are of two types:

1. **Function Components / State-less Components**
2. **Class Components / State-full Components**

- The simplest way to define a component is to write a JavaScript function:

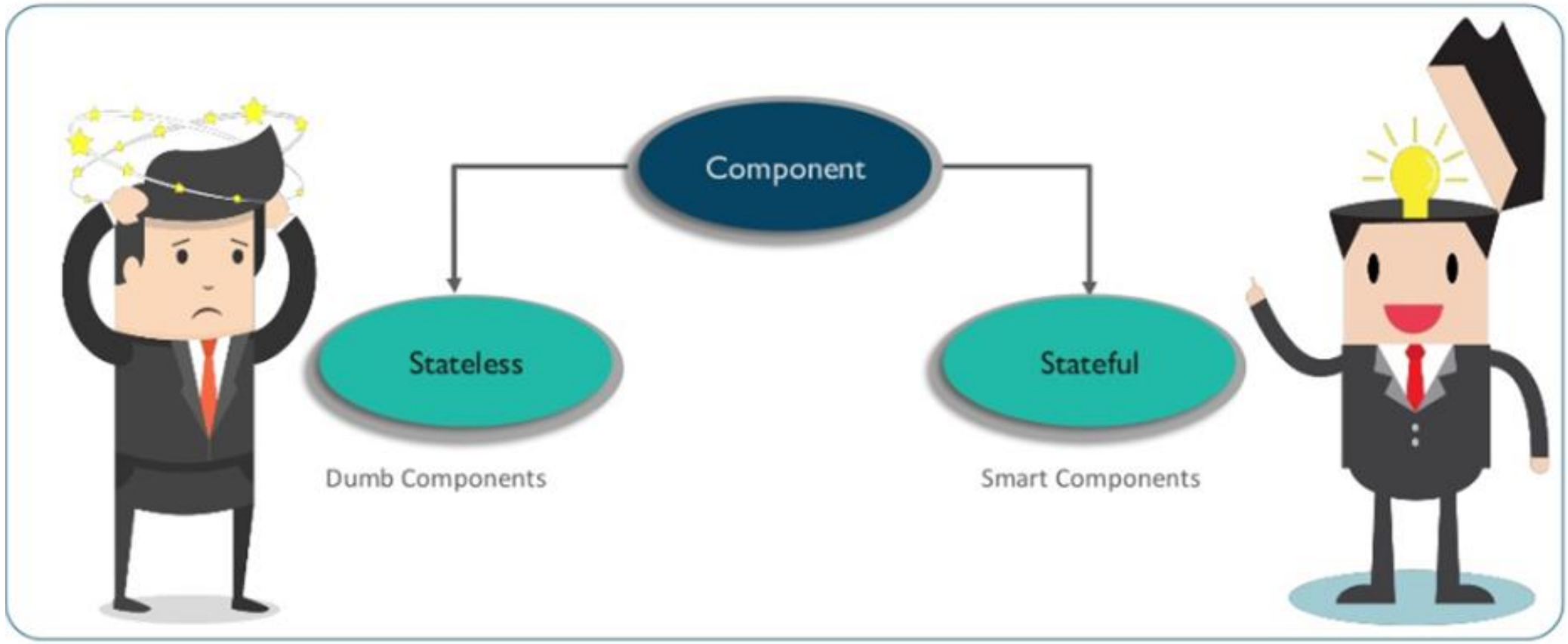
```
function Welcome(props){
        return <h1> Hello, {props.name} </h1>
}
                                        <Welcome name="Hita"/>
```

- Using class

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

State-less & State-full Components

- **State-less and State-full Components**



State-less and State-full Components

State-less

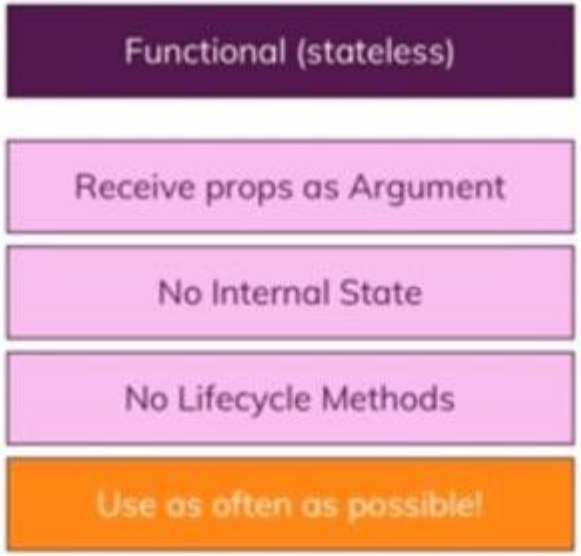
- ✓ Calculates the State Internal State of Components
- ✓ Contains No Knowledge of Past, Current and Possible future of State Changes.

State-full

- ✓ Core which stores information about components state in memory
- ✓ Contains Knowledge of Past, Current and Possible future of State Changes.

Stateful v/s Stateless

Stateful Component	Stateless Component
1. Stores info about component's state change in memory	1. Calculates the internal state of the components
2. Have authority to change state	2. Do not have the authority to change state
3. Contains the knowledge of past, current and possible future changes in state	3. Contains no knowledge of past, current and possible future state changes
4. Stateless components notify them about the requirement of the state change, then they send down the props to them.	4. They receive the props from the Stateful components and treat them as callback functions.



Rendering a Component

- const element = <Welcome name="John" />

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**".

```
ReactDOM.render(
        element,
        document.getElementById('root')
)
```

1. We call ReactDOM.render() with the <Welcome name="John" /> element.
2. React calls the Welcome component with { name : 'John' } as the props.
3. Our Welcome component return a <h1> Hello, John</h1> element as the result.
4. React DOM efficiently updates the DOM to match <h1> Hello, John</h1>

Note : React treats components starting with lowercase letters as DOM tags.
For example, represents an HTML div tag, but <Welcome /> represents a component and requires Welcome to be in scope.

Props

- Props are the arguments passed into React Component.
- They are the object in React that contains properties about the components.
- Props are passed to components via HTML Attributes.
- React Props:

  ⟹ Function Arguments in JS

  ⟹ Attributes in HTML

- **Props Usage's:**

1. To send the props inside a component.
2. Props can pass data from one component to another component.
3. Props through variable name
4. Props as an object
5. Props inside Constructor

State

- A component needs state when some data associated with it changes over time.
- React Components are built using the state objects.
- **In State Object:**

  we can store the property values that belongs to component.
  when state object changes the component re-renders.

- The most important difference between state and props is that props are passed from a parent component, but state is managed by the component itself.

- **A component cannot change its props, but it can change its state.**

Creating the State Object :

1. **State object is initialized in constructor**:
   Using this.state directly only in constructor() to modify the state of component.

State continued…

Note:
Use this.setState({ }) in all other functions or lifecycle methods because if we use this.state = 'something' the lifecycle methods will not be invoked. So never ever use this.state inside other functions except constructor().

2. State object can contain many properties.
   ex: this.state = {
   color: "Blue", brand: "Audi" }

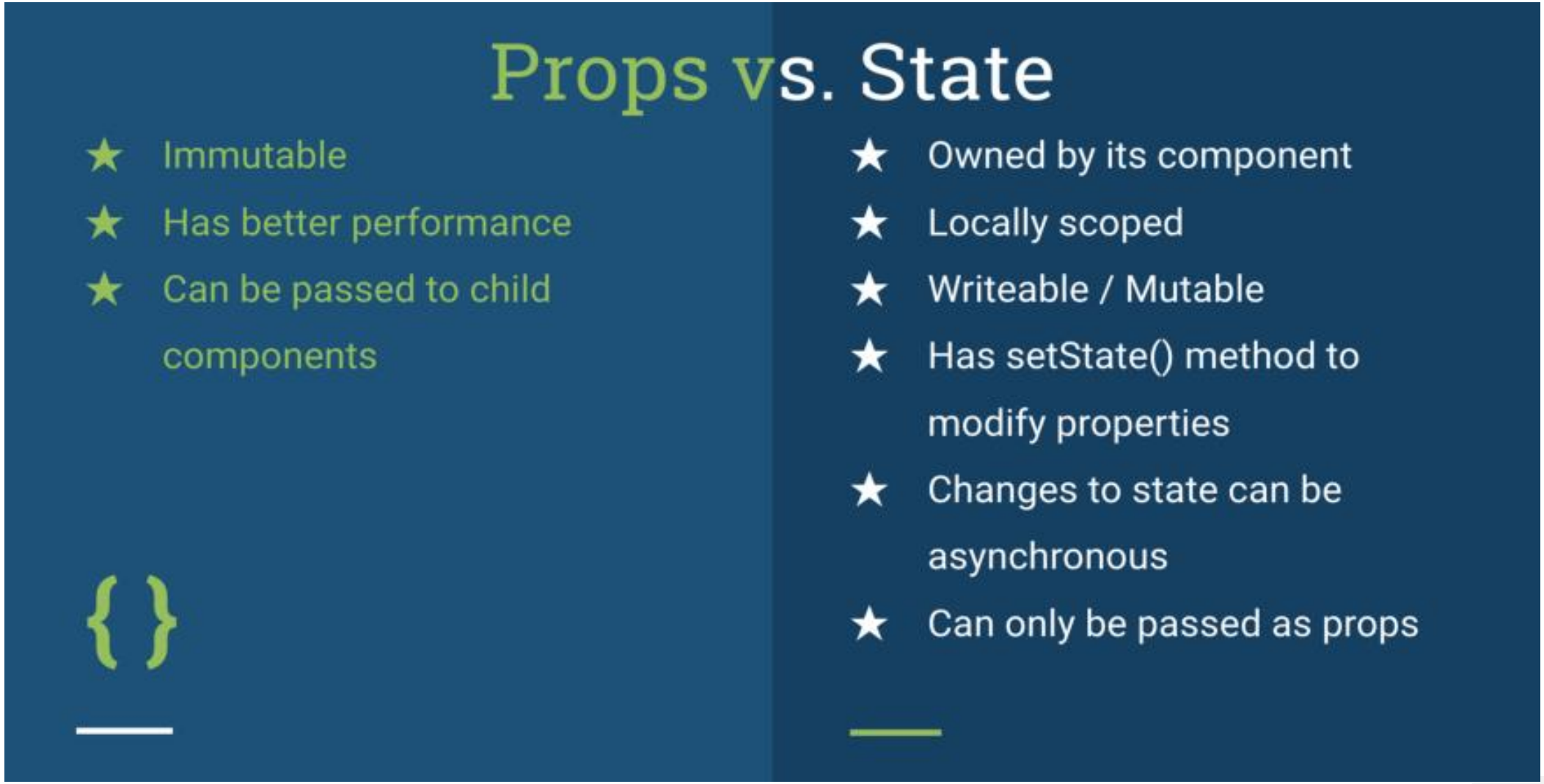
3. Using of the state object :
   this.state.propertyName ➔ { this.state.color }

4. Changing the state Object:
   this.setState() method

5. Make API calls in componentDidMount() lifecycle method.

6. Clean up process in componentWillUnmount() lifecycle method.

Props v/s State



Props validation

- Props validation is a tool that will help the developers to avoid future bugs and problems.
- React components used special property PropTypes that help you to catch bugs by validating data types of values passed through props.
- Validating Props:

App.propTypes is used for props validation in react component

Syntax:
```
class App extends React.Component {
render() {}
}
Component.propTypes = { /*Definition */};
```

Ex:

SN	PropsType	Description
1.	PropTypes.any	The props can be of any data type.
2.	PropTypes.array	The props should be an array.
3.	PropTypes.bool	The props should be a boolean.
4.	PropTypes.func	The props should be a function.
5.	PropTypes.number	The props should be a number.
6.	PropTypes.object	The props should be an object.
7.	PropTypes.string	The props should be a string.

Props Validation

Ex-1: import PropTypes from 'prop-types';

```
class Greeting extends React.Component {
render() {
return (
 <h1>Hello, {this.props.name}</h1>
);  }}
Greeting.propTypes = { name: PropTypes.string };
```

<Greeting name=“ ” />

Ex-2: With single Child

```
const children = this.props.children;
 return (
<div>   {children} </div>
); } }
MyComponent.propTypes = {  children: PropTypes.element.isRequired };
```

Note: Can have Default Props:

```
Greeting.defaultProps = { name: 'Stranger' };
```

Function to Class Components


```
function Clock(props){
        return (
        <div><h1>{props.date.toLocaleTimeString()}</h1></div>
        )}
```

Convert a function component to class in five steps.

1. Create an ES6 class, with the same name, that extends React.Component.
2. Add a single empty method to it called render().
3. Move the body of the function into the render() method.
4. Replace props with this.props in the render() body.
5. Delete the remaining empty function declaration.

```
class Clock extends React.Component {
        render(){
        return (
        <div><h1>{this.props.date.toLocaleTimeString()}</h1></div>
        )}
        }
```

React Router

- React router is a routing library built on top of the react which is used to create the routing in react apps.

- Install react-router-dom:

```
npm install react-router-dom --save
```

Router Components:

1. BrowserRouter: It helps to keep UI in sync with URL. & It's a Parent Component, which stores all other components.

2. Route: Its Conditionally shown component, It renders the UI only if the path matches the current URL.

3. Link: It helps to create the links to different routes & implements the navigation around Application. (Works like <a> tag)

4. Switch: It helps to render only the first route that matches the location. Basically to render a single components .

React Router continued…

Note:

1. Exact : It is used to match the exact value with URL.

2. Path: path specifies a path name we assign to our component.

3. Component: It refers to the component which will render a matching path.

Example:

```
import { Route, Link, BrowserRouter as Router, Switch } from 'react-router-dom';
const routing = (
<Router>
<div>
<Route exact path="/" component={App} />
<Route path="/users" component={Users} />
<Route path="/contact" component={Contact} />
</div>
</Router>
)
```

Router Link

Example:

```
const routing = (
<Router>
<div>
<ul>
        <li><Link to="/">Home</Link></li>
        <li><Link to="/users">Users</Link></li>
        <li><Link to="/contact">Contact</Link></li>
</ul>
        <Route exact path="/" component={App} />
        <Route path="/users" component={Users} />
        <Route path="/contact" component={Contact} />
</div>
</Router>
        )

ReactDOM.render(routing, document.getElementById('root'))
```

Events

- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.
- For example, the HTML :

```
<button onclick = " activateUser() ">  Activate Users </button>
```

In React with Stateless Component:

```
<button onClick = { activateUser() } >  Activate Users </button>
```

In React with Statefull Component:

```
constructor(props){
super(props);
// This binding is necessary to make 'this' work in callback
this.handleClick = this.handleClick.bind(this);
}
 <button onClick = { this.handleClick } >  Activate Users </button>
```

- Passing Arguments to Event Handlers.

```
<button onClick = { this.deleteRow.bind(this, id, name) } >  Delete Row </button>
```

Lifecycle Hooks

The React component goes through the following phases:

- Mounting Phase
  - constructor(props)
  - static getDerivedStateFromProps (props, state)
  - render()
  - componentDidMount ()

- Update Phase
  - static getDerivedStateFromProps(props, state)
  - shouldComponentUpdate (),
  - render()
  - getSnapShotBeforeUpdate (prevProps, prevState),
  - componentDidUpdate(prevProps, prevState, snapshot)

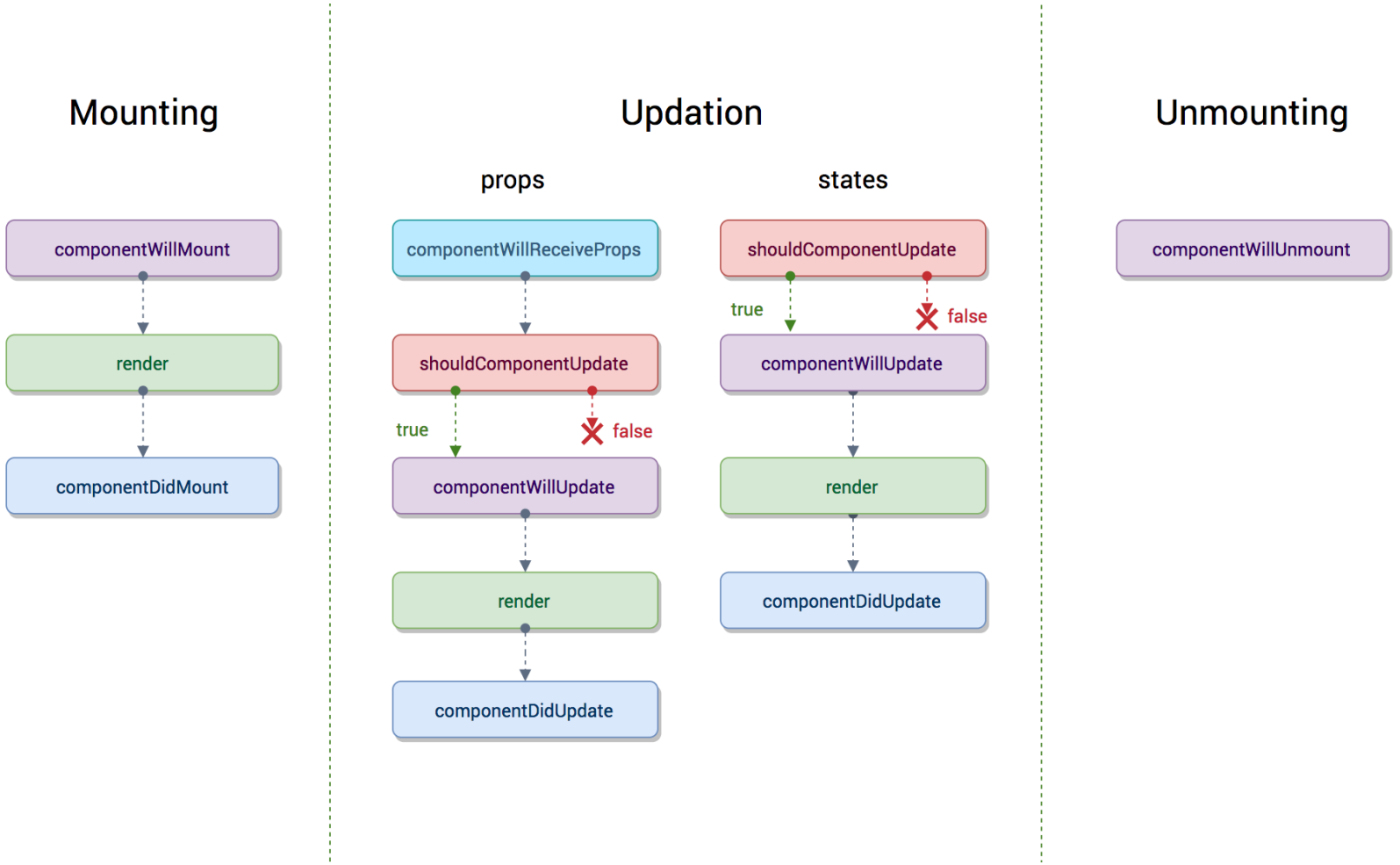
- Unmounting Phase
  - componentWillUnmount()

Note :

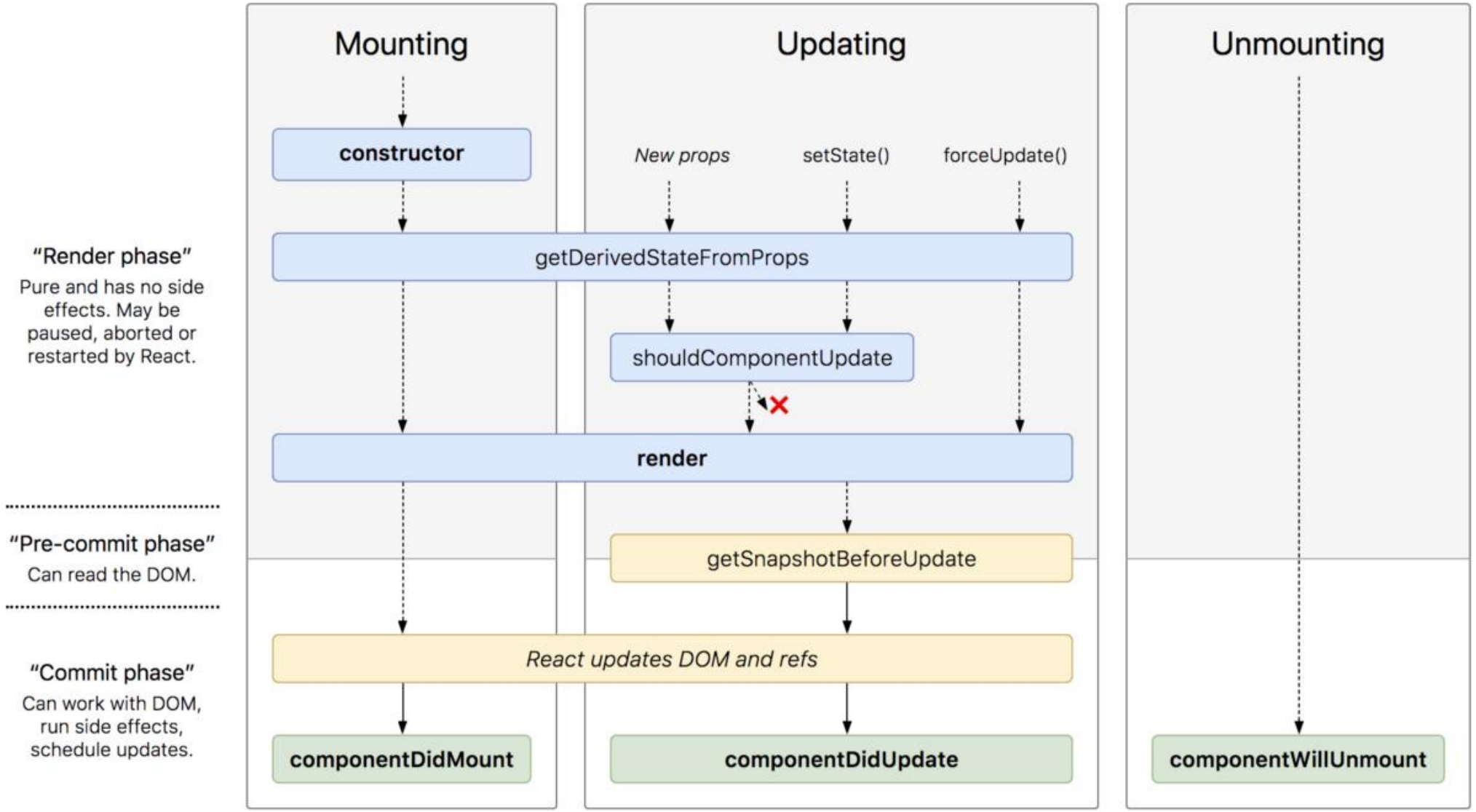
this.forceUpdate() to force a re-render

Lifecycle Hooks(Old)

- Deprecated = componentWillMount, componentWillReceiveProps, componentWillUpdate.



React16 Lifecycle Hooks



Conditional Rendering

- Conditional rendering in React works the same way conditions work in JavaScript.
- Use JavaScript operators like if or the conditional operator to create elements representing the current state.
- To render something based on some condition.
- Create multiple components and render them based on some conditions. This is also a kind of encapsulation supported by React.
- Ex:

```
 if (props.season === 'summer') {
return <Lake name="jenny Lake" />
} else {
return <Skiresort name =" JMR !!" />
}
```

React Fragments

- React introduced Fragments from the 16.2 and above version.
- Fragments allow you to group a list of children without adding extra nodes to the DOM.
- It helps in dealing with the issue of json components

Syntax:

```
<React.Fragment>
<Lake />
<Resort />
</React.Fragment>
```

or

```
<>
<Lake />
<Resort />
</>
```

Use of Fragments:

- It makes the execution of code faster as compared to the div tag.
- It takes less memory.

Note: Key is the only attributes that can be passed with the Fragments.

React Lists

- Lists are very useful when it comes to developing UI of any website.
- Lists are mainly used for displaying menus in a website, for example, the navbar menu.

Ex:

```
const numbers = [1,2,3,4,5];
 const updatedNums = numbers. map((number)=>{
 return <li>{number}</li>;
 });

ReactDOM.render( <ul> {updatedNums} </ul>,  document.getElementById('root') );
```

But,

If items are fetched from database and then displayed as lists in the browser. So from the component's point of view we can say that we will pass a list to a component using props and then use this component to render the list to the DOM. We can update the above code in which we have directly rendered the list to now a component which will accept an array as props and returns an unordered list.

React Lists


```
// Component that will return unordered list
      function Navmenu(props)  {
      const list = props.menuitems;
      const updatedList = list.map((listItems)=>{
       return <li>{listItems}</li>;
      });
      return(
      <ul>{updatedList}</ul>
       ); }

  const menuItems = [1,2,3,4,5];
 ReactDOM.render( <Navmenu menuitems = {menuItems} />,   document.getElementById('root') );
```

Note: **We get an Warning……?**

Warning: Each child in an array or iterator should have a unique "key" prop, It says that each of the list items in our unordered list should have an unique key.

React Keys

- A “key” is a special string attribute you need to include when creating lists of elements in React.
- Keys are used in React to identify which items in the list are changed, updated or deleted.
- Keys are used to give an identity to the elements in the lists.
- It is recommended to use a string as a key that uniquely identifies the items in the list.
- Key is an Identifier for dynamically created Element.

Note:

✓ Keys are not same as props, only the method of assigning “key” to a component is same as that of props.

✓ Keys are internal to React and can not be accessed from inside of the component like props.

✓ Therefore, we can use the same value we have assigned to the Key for any other prop we are passing to the Component.

React Keys

Ex: Adding keys for list of numbers:

1.First create the list of values.

```
const list = [1, 2, 3, 4, 5];
```

2. No id, so use toString() and map it.

```
function App({items}) {
return(
<ul> { items.map(item => (
<li key = { items.toString()> {items} </li>
 ))}
 </ul>
 );}
```

While Rendering,

```
<App items = {list} />
```

React Component API

- React JS component is a top-level API. It makes the code completely individual and reusable in the application.
- It includes various methods for: Creating elements, Transforming elements, Fragments.
- Important methods available in the React component API.

1. setState()
2. forceUpdate()

- setState() : This method is used to update the state of the component.

  Syntax: this.stateState(object newState[, function callback]);

- forceUpdate() : This method allows us to update the component manually.

  Syntax: Component.forceUpdate(callback);

React Forms

- React uses forms to allow users to interact with webpage.

1. Adding Forms to React.

```
<form>
<h1>My Form </h1>
<input  type="text" />
</form>
```

2. Handling Forms with React.

   It mainly depends how we handle the data when it changes or gets submitted.

In HTML,

form data ➔ DOM

In React,

form data ➔ Components

(When is handled by components all the data is stored in the form of state)

- The control change is handled by → onChange attribute
- The access to the field is given using the event.target.value.

React Forms

Ex:

```
this.state = {
            username: ' '  };
            }
formChangeHandler = (event) => {
                this.setState({ username: event.target.value });
                }
render(){
        return(
        <form>
                <h1> Hi {this.state.username}</h1>
                <input onChange={this.formCangeHandler} type="text" />
        </form> );
```

React Hooks

- Hooks are functions that let you “hook into” React state and lifecycle features from function components. Hooks don’t work inside classes — they let you use React without classes.

➢ useState(initialState) - useState returns a pair : the *current* state value and a function that lets you update it.

Ex1 - const [count, setCount] = useState(0)

0	-	initial state
count	-	variable to get the value of the current state.
setCount	-	function to update the state

Ex2 - const [todos, setTodos] = useState([{ text : ‘Send an email’ }])

[{ text : ‘Send an email’ }]	-	initial state
todos	-	variable to get the value of the current state.
setTodos	-	function to update the state

React Hooks

- useEffect( ) - useEffect adds the ability to perform side effects from a function component. It serves the same purpose as componentDidMount, componentDidUpdate and componentWillUnmount in React classes, but unified into a single API.

Ex1 -

```
const [ count, setCount ] = useState( 0 )
// Similar to componentDidMount and componentDidUpdate:
useEffect( () => {
        // Update the document title using the browser API
         document.title = `You clicked ${count} times`
})
```

Hooks are JavaScript functions, but they impose two additional rules:

1. Only call Hooks at the top level. Don't call Hooks inside loops, conditions, or nested functions.
2. Only call Hooks from React function components. Don't call Hooks from regular JavaScript functions.

Controlled Components

- A controlled component is bound to a value, and its changes will be handled in code by using event-based callbacks.
- Controlled components have functions that govern the data passing into them on every onChange event occurs. This data is then saved to state and updated with setState() method.
- It makes component have better control over the form elements and data.

Controlled	Uncontrolled
It does not maintain its internal state.	It maintains its internal states.
Here, data is controlled by the parent component.	Here, data is controlled by the DOM itself.
It accepts its current value as a prop.	It uses a ref for their current values.
It allows validation control.	It does not allow validation control.
It has better control over the form elements and data.	It has limited control over the form elements and data.

React Refs

- Refs is the shorthand used for references in React. (Similar to keys).
- Its an attribute which makes it possible to store a reference to particular DOM nodes or React elements.
- It provides a way to access React DOM nodes or React elements and how to interact with it.
- It is used when we want to change the value of a child component, without making the use of props.

When to Use Refs :

- Managing focus, text selection, or media playback.
- Triggering imperative animations.
- Integrating with third-party DOM libraries.
- Also used in callbacks

Note: The ref attribute cannot be used on **function components** because they don't have instances.

React Refs

Refs can be created by using React.createRef(). It can be assigned to React elements via the ref attribute.

Ex:

```
class MyRef extends React.Component {
constructor(props) {
super(props);
this.callRef = React.createRef();
this.addingRefInput = this.addingRefInput.bind(this);
}

addingRefInput() {
this.callRef.current.focus();
}

render() {
return (
<div>  <input  type="text"  ref={this.callRef}  />
<input  type="button"  value="Add text input"  onClick={this.addingRefInput}  />
</div>
```

React Refs

Callback Ref:

- It gives more control when the refs are set and unset.
- React allows a way to create refs by passing a callback function to the ref attribute of a component.

Syntax: `<input type="text" ref={element => this.callRefInput = element} />`

Forward Ref:

- Ref forwarding is a technique that is used for passing a ref through a component to one of its child components.
- It can be performed by making use of the React.forwardRef() method.
- This technique is particularly useful with higher-order components and specially used in reusable component libraries.

Note:

React with useRef() : react hook ref

Higher Order Components(HOC)

- HOC is an advanced technique for reusing component logic.
- HOC are common in third-party libraries.
- It is a function that takes a component and returns a new component.
- A component transforms props into UI, a higher-order component transforms a component into another component.

- A higher order component function accepts another function as an argument.

Ex: map function.

Syntax: `const NewComponent = higherOrderComponent(WrappedComponent);`

Note:

- Do not use HOCs inside the render method of a component.
- HOCs does not work for refs as 'Refs' does not pass through as a parameter or argument.
- The primary use of Higher-Order Component is to enhance the reusability of particular components in multiple modules or components.

React Context API

- Context provides a way to pass data through the component tree without having to pass props down manually at every level.

  - React.createContext
    const MyContext = React.createContext(defaultValue)

  - Context.Provider
    <MyContext.Provider value = { some value }>

  - Context.Consumer
    <MyContext.Consumer>
    { value => /* render something based on the context value */ }
    </MyContext.Consumer>

Axios

- The Axios is a Promise based HTTP client for the browser and node.js

The AXIOS HTTP Client Request Parameters

baseUrl : This is a base URL, it'll be pre-pended to any relative URL.
headers : This is an object of key/value pairs to be sent as headers.
params : This contains an object of key/value pairs that will be serialized and appended to the URL as a query string.
responseType : Defined response type in a format other than JSON.
auth : This is the HTTP Basic auth string which will send with each http request.
method : HTTP methods.

The Response Object of Axios

data : The payload returned from the server.
status : The HTTP code returned from the server.
statusText : The HTTP status message returned by the server.
headers : All the headers sent back by the server.
config : The original request configuration.
request : The actual XMLHttpRequest object.

Error Boundary

- The error boundary mechanism helps to show meaningful error message to user on production instead of showing any programming language error.
- Error boundaries are React components which catch JavaScript errors anywhere in our app, log those errors, and display a fallback UI.
- It does not break the whole app component tree and only renders the fallback UI whenever an error occurred in a component.
- Error boundaries catch errors during rendering in component lifecycle methods, and constructors of the whole tree complex application which have multiple components, we can declare multiple error boundaries

Syntax:

```
<ErrorBoundary>
    <Home/>
</ErrorBoundary>
```

Error Boundary continued…

There are 2 methods in Error Boundary:

1.static getDerivedStateFromError(error){ } : It takes error message as an input & return New State

Syntax:
```
static getDerivedStateFromError(error) {
    return (  hasError : true
        //Logic ) }
```

2. componentDidCatch (error, errorInfo){ }

Syntax:
```
componentDidCatch (error, errorInfo) {
    console.log (error);
    console.log (errorInfo);  }
```

Note: Whenever there is an error in JS/React object, it creates a Object of Error.

```
throw new Error("error");
```

* It is for the Customized Error.

TESTYANTRA

SOFTWARE SOLUTIONS (INDIA) PVT. LTD.

Thank You !!!

No.01, 3rd Cross Basappa Layout, Gavipuram Extension, Kempegowda Nagar, Bengaluru, Karnataka 560019

praveen.d@testyantra.com

www.testyantra.com

EXPERIENTIAL learning factory