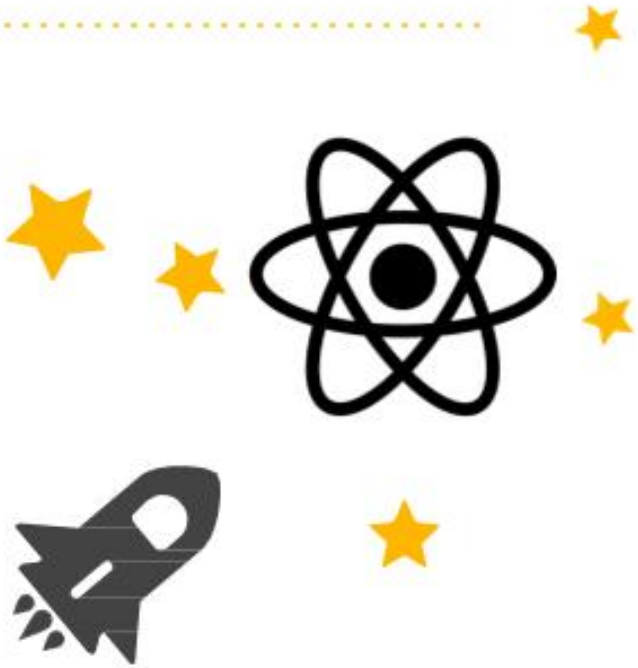
Introduction to React

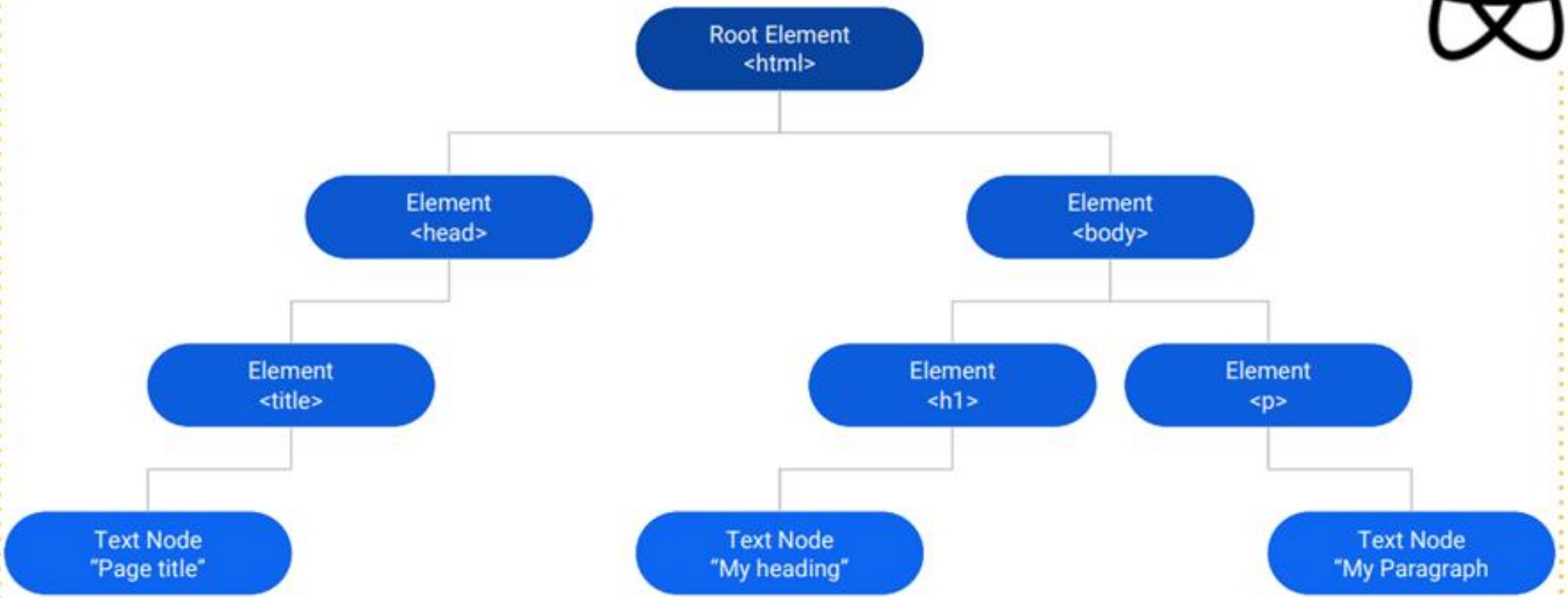
Vijesh M. Nair
Assistant Professor
Dept. of CSE (AI-ML)

React Js

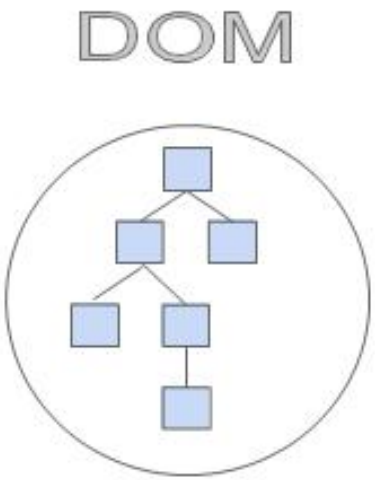


React is a javascript library for building interactive User Interface(UI).

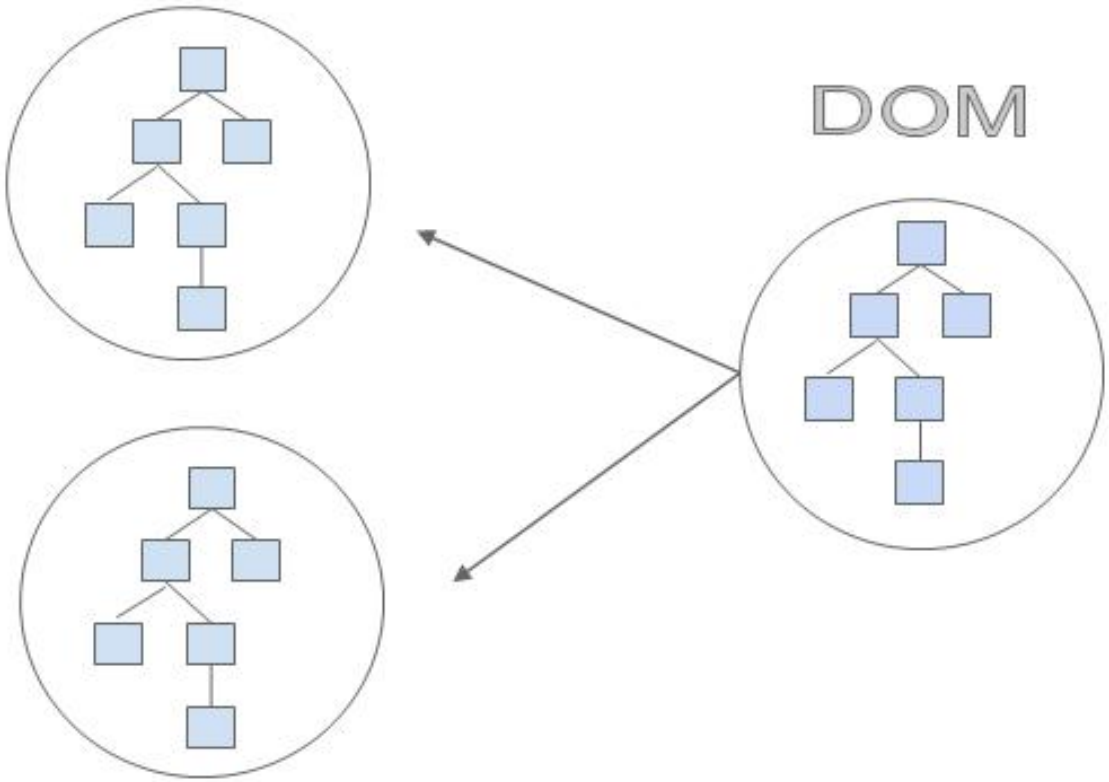
DOM



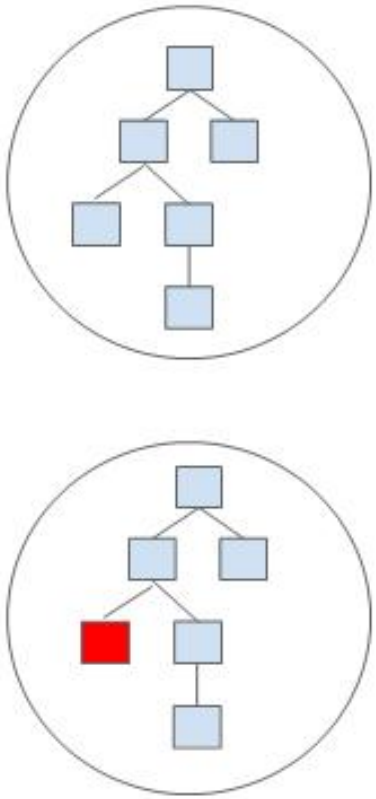
What makes React so fast?



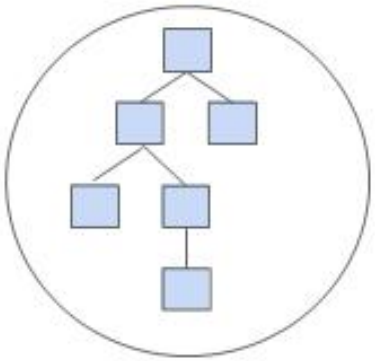
Virtual DOM



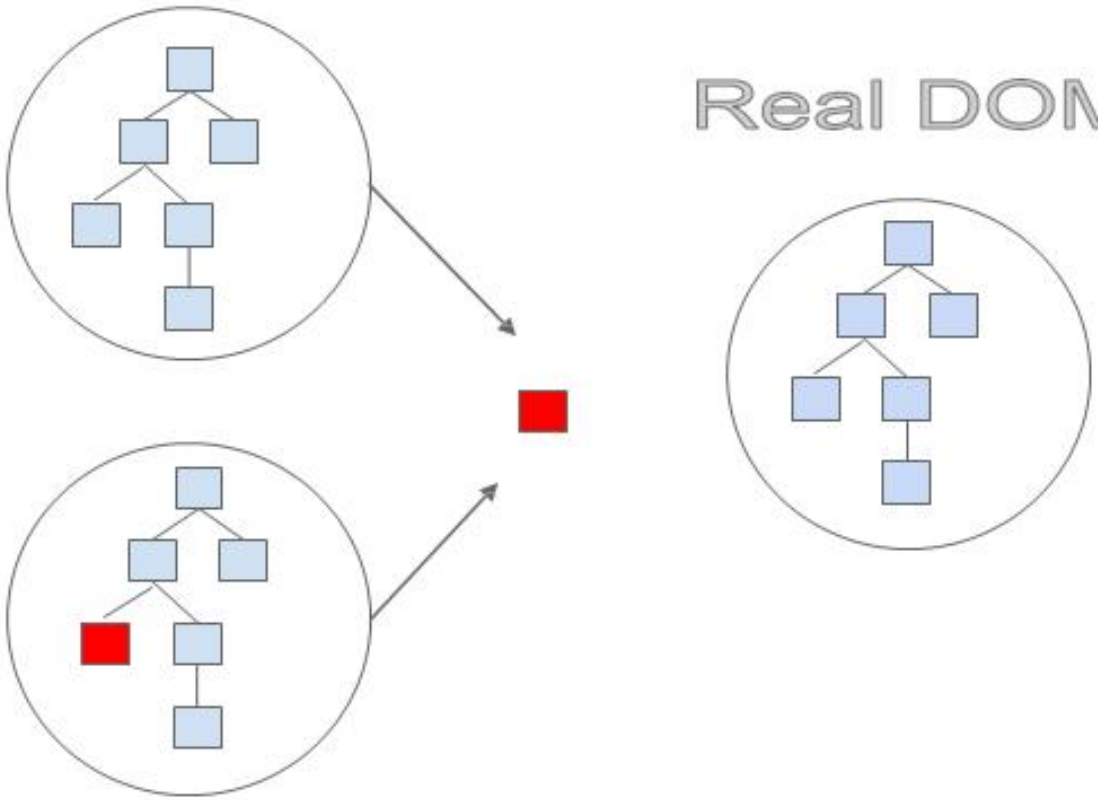
Virtual DOM



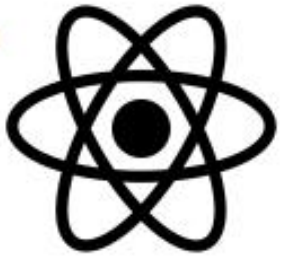
Real DOM



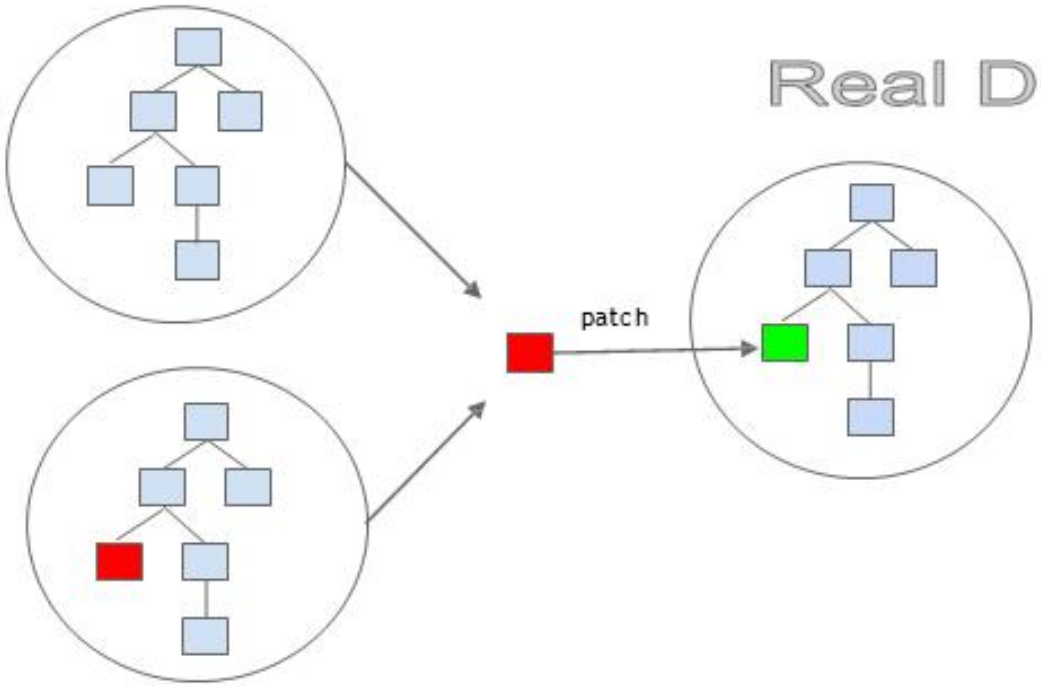
Virtual DOM



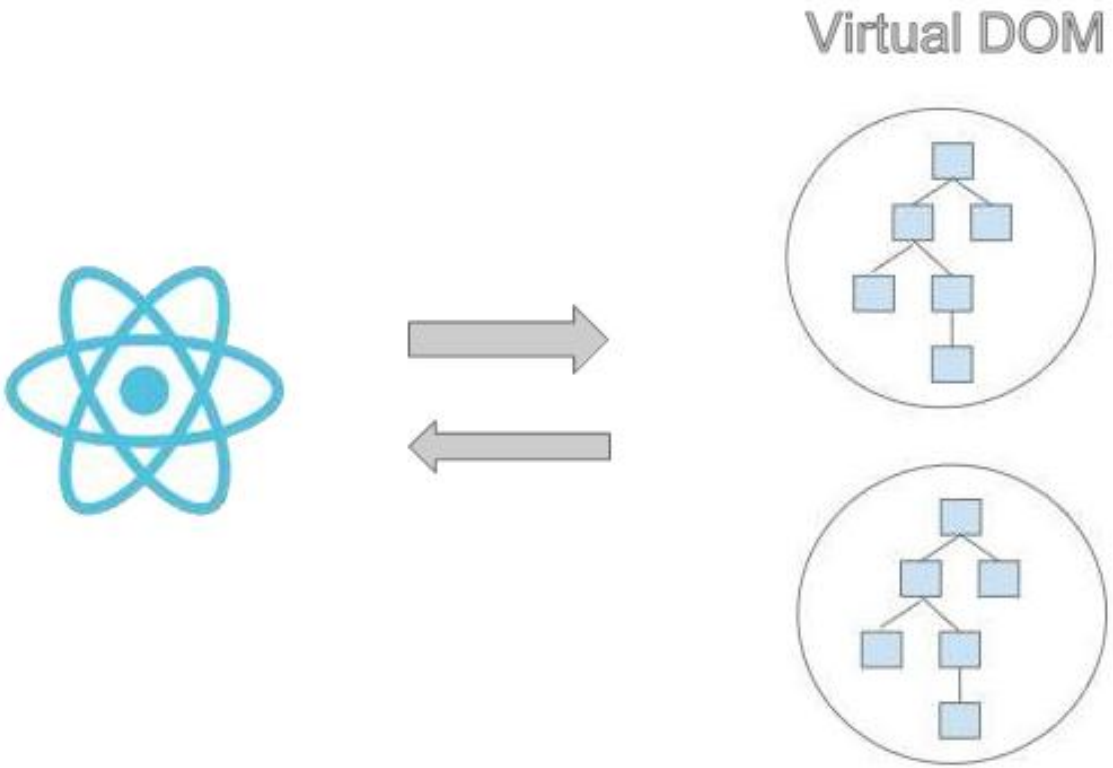
Real DOM



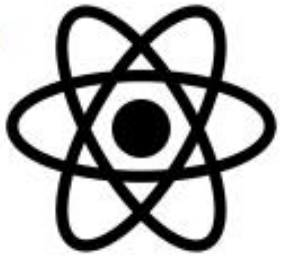
Virtual DOM



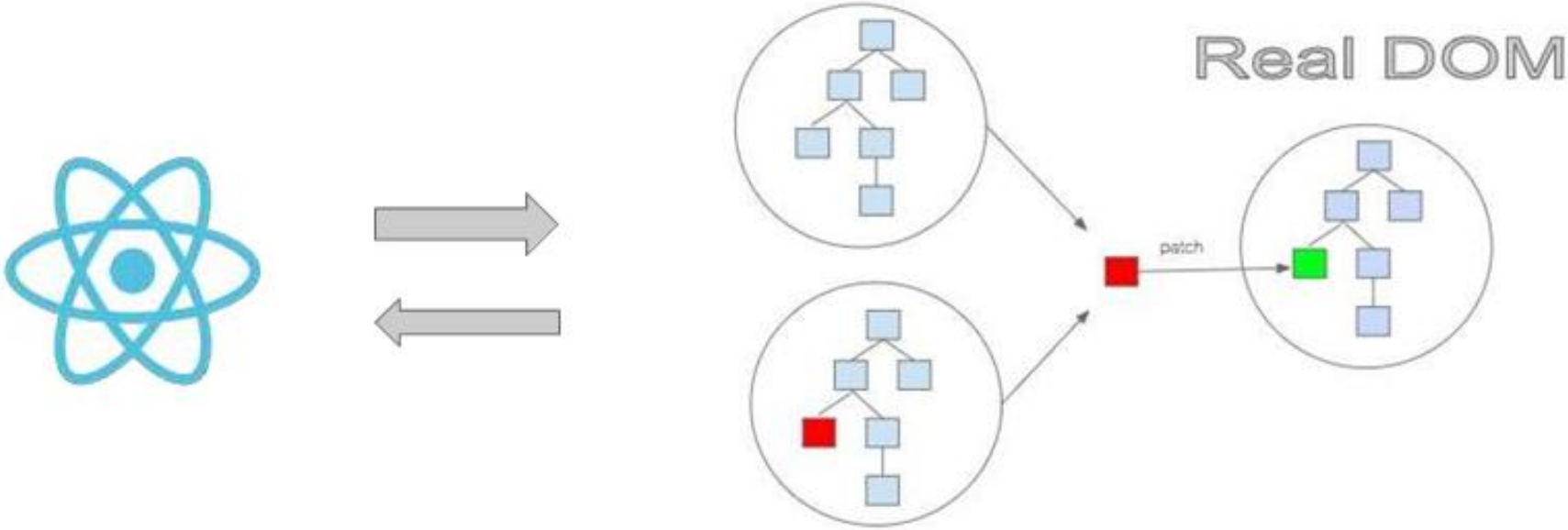
Why is React fast?



Why is React so fast?

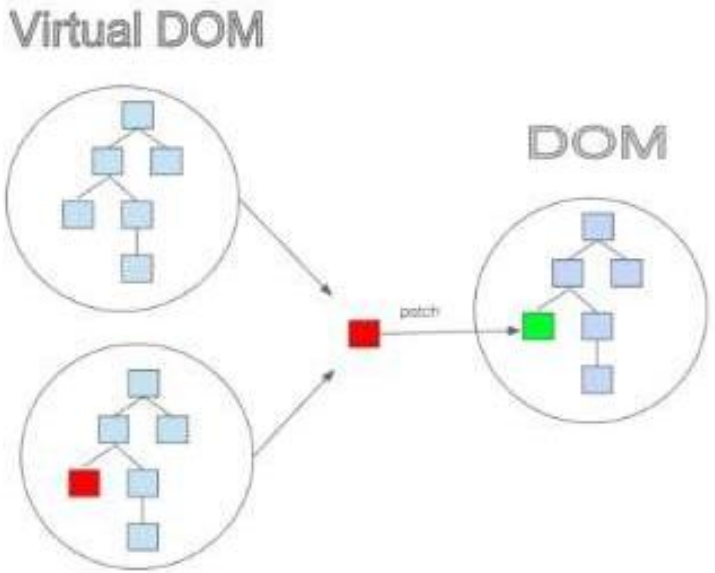


Virtual DOM

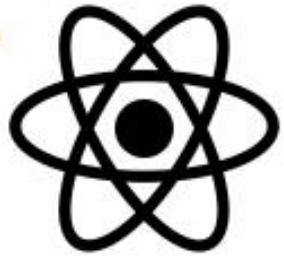


What is a Virtual DOM?

A Javascript Object that is a "virtual", representation of the 'real" DOM.

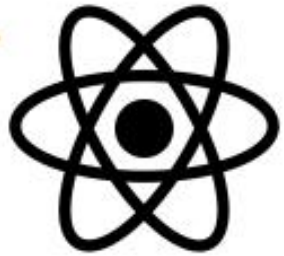


Why use react?

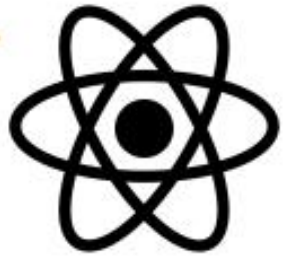


- Updates and renders only the elements that change/update in the DOM(hence quick rendering)
- Build encapsulated components that manage their own state.
- React can also render on the server using Node and powerful mobile apps using react native.

History of React



- Created by **Jordan Walke**, a software engineer at Facebook
- First deployed on ***Facebook's newsfeed in 2011*** and later on ***Instagram.com in 2012***
- Open-sourced at JSConf US in May 2013
- Facebook announced **React Fiber**, on April 18, 2017
- **React 360 V1.0.0** was released to the public on April 19, 2017



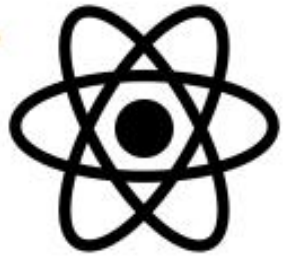
Set up React App

1- Add React in a Minute (Using React Scripts)

2- Using create-react-app

3 Using Parcel

4 Using Webpack and Babel



React Components

Components allow you to split the UI into independent reusable pieces of JS codes.

Class Based Components

Extend React Component Class and it requires us to use render()

Functional Based Components

Are pure JavaScript function that accepts props as its argument, and returns some JSX

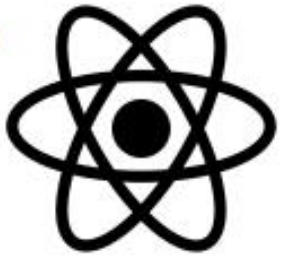
Class Based Components

```
class Heading extends React.Component {
  render() {
    return <h1>{this.props.children}</h1>
  }
}
```

Functional Based Components

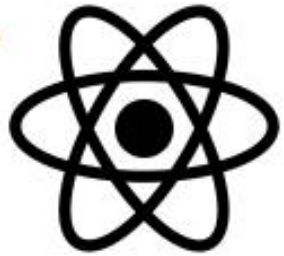
```
function Heading(props) {
  return <h1>{props.children}</h1>
}

const Button = (props) => <button className="button" {...props} />;
```

JSX (JavaScript-XML)



- JSX is a XML-like syntax extension to JavaScript that creates React elements.
- Its makes your code human readable, we had to use **React.createElement()**
- Gives us expressiveness of JavaScript along with HTML like template syntax.

Examples of using JSX



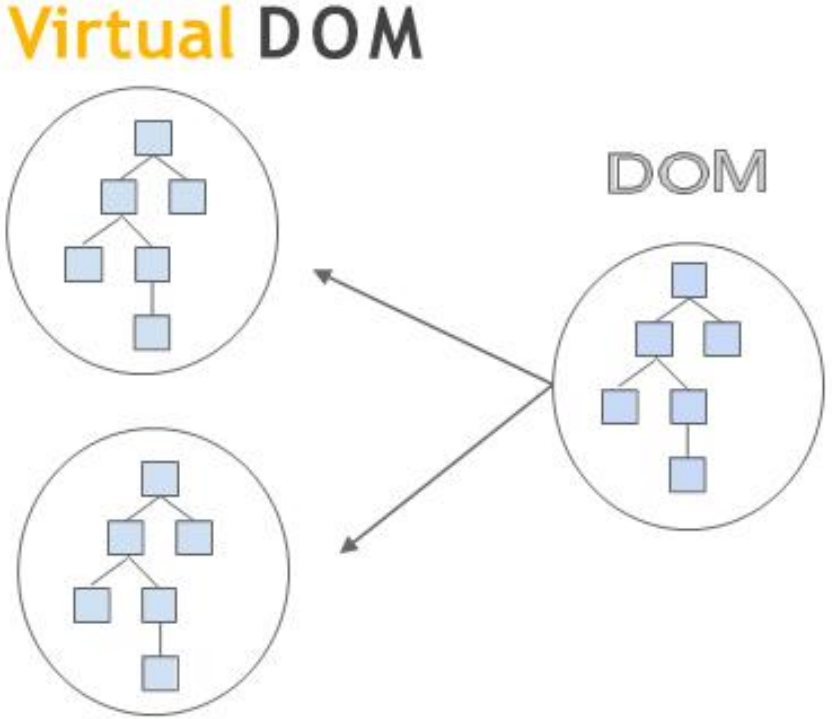
```jsx
const element = <h1>Hello, world!</h1>;
```

```jsx
const name = 'Josh Perez';
const element = <h1>Hello, {name}</h1>;
```

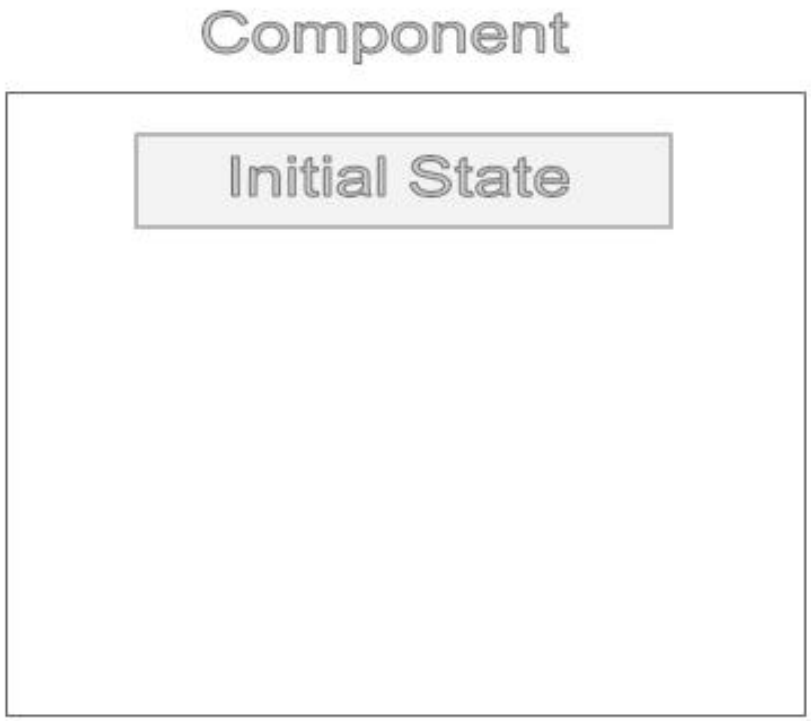
```jsx
const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

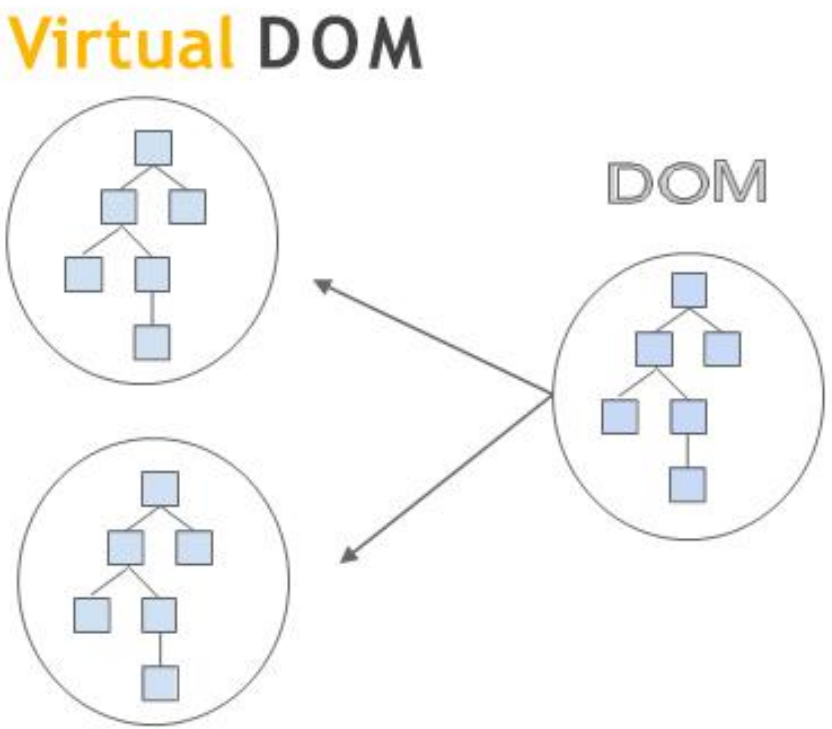
```jsx
const element = <img src={user.avatarUrl}></img>;
```

State of a Component

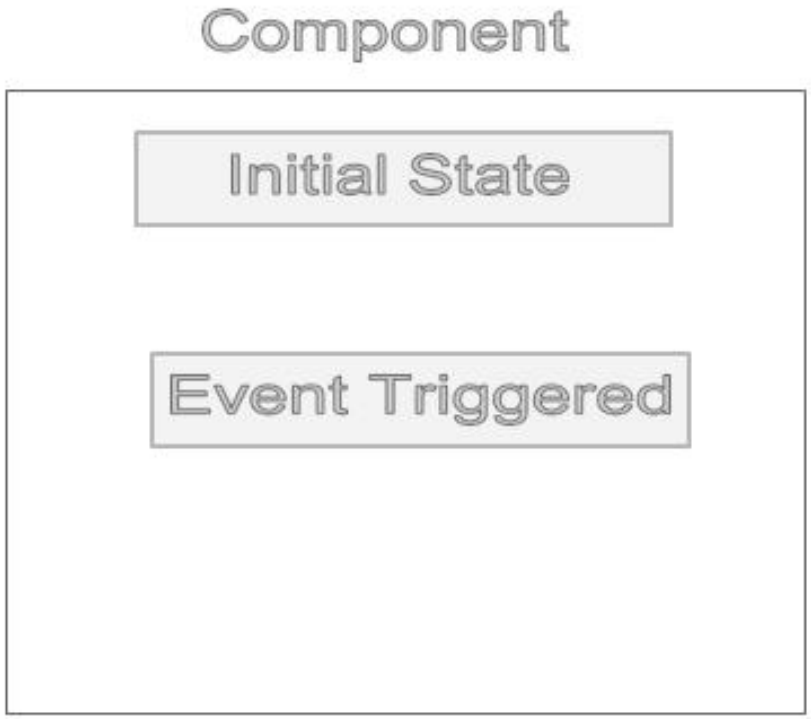


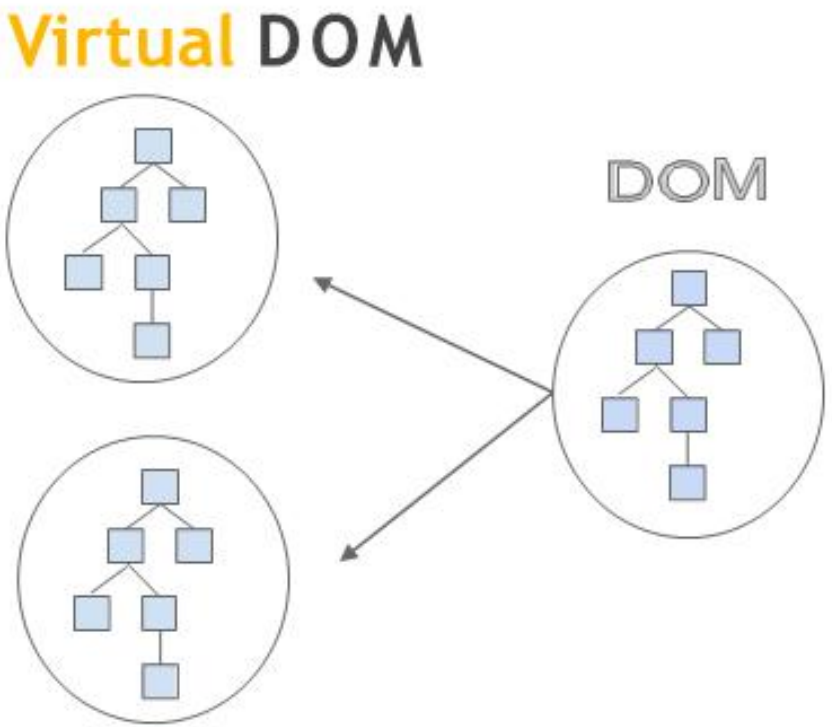
State of a Component



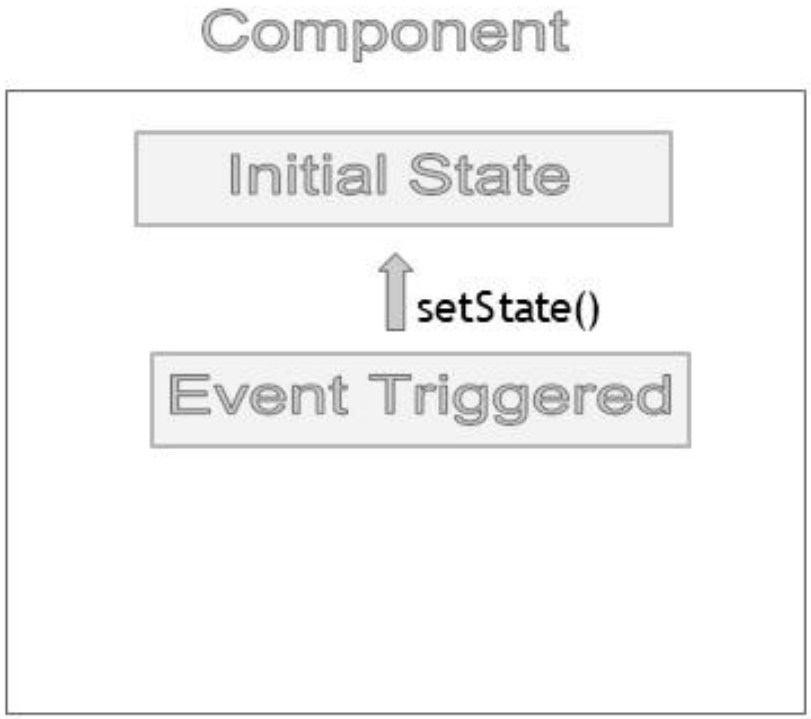


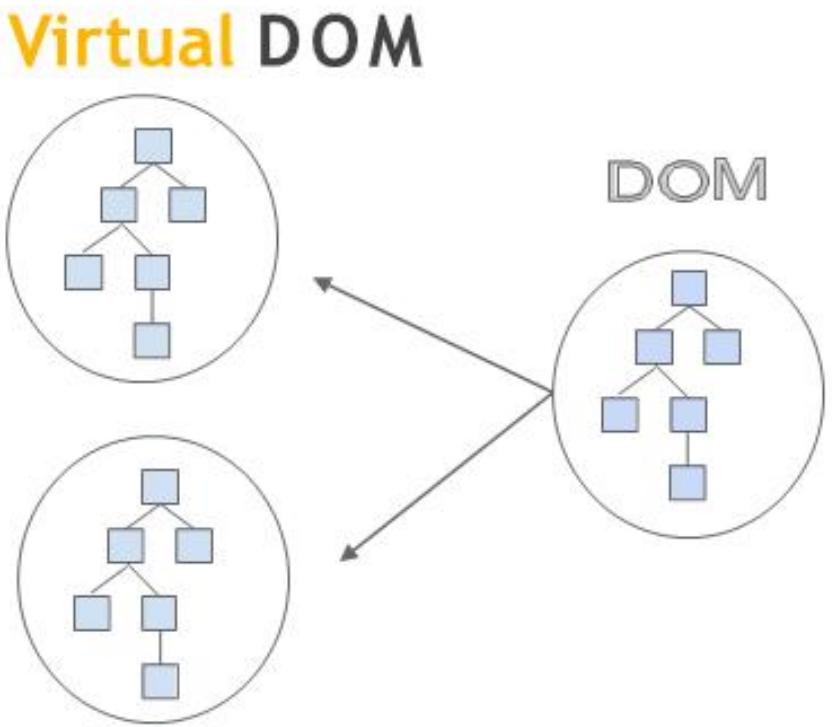
State of a Component



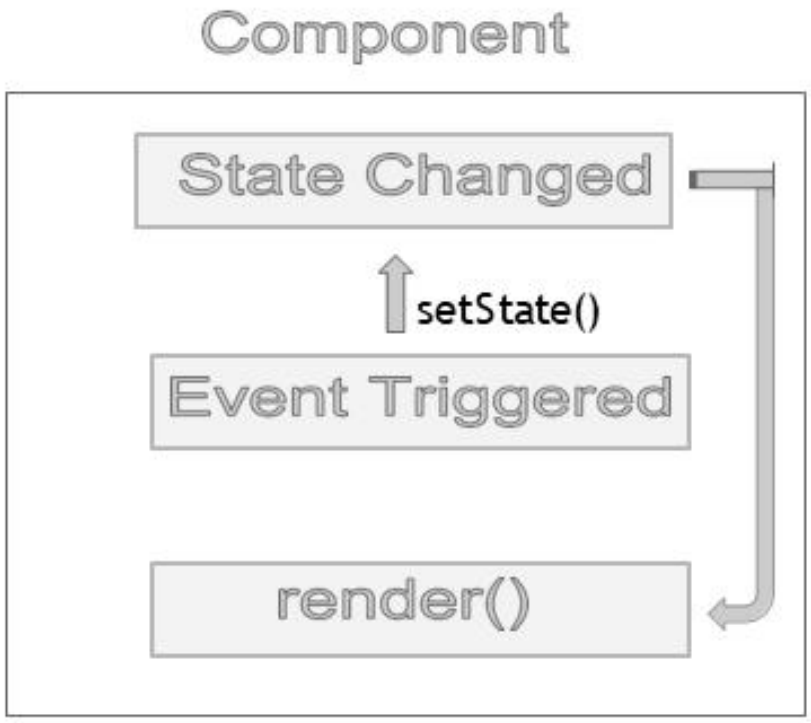


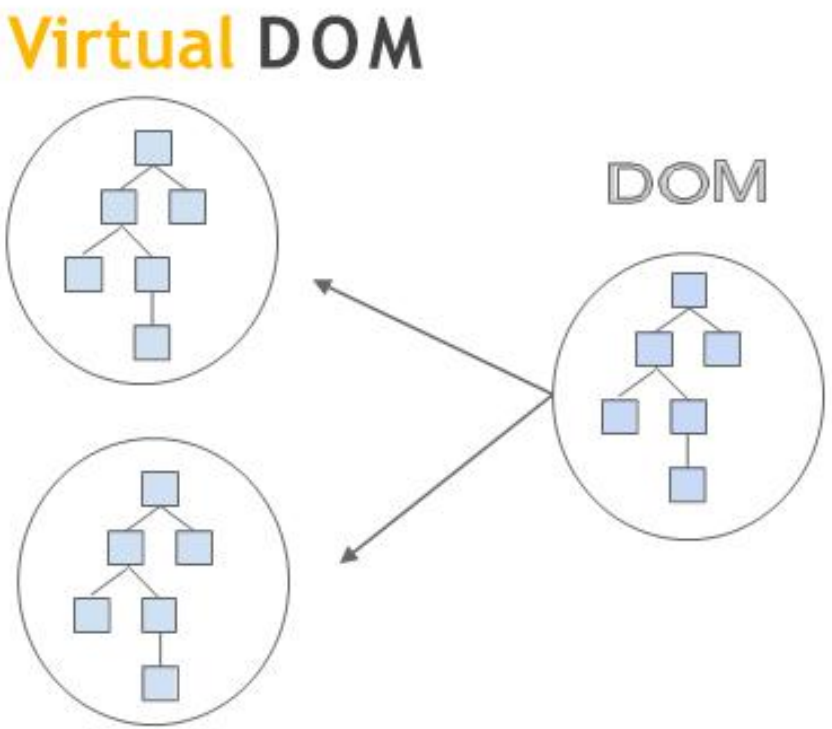
State of a Component



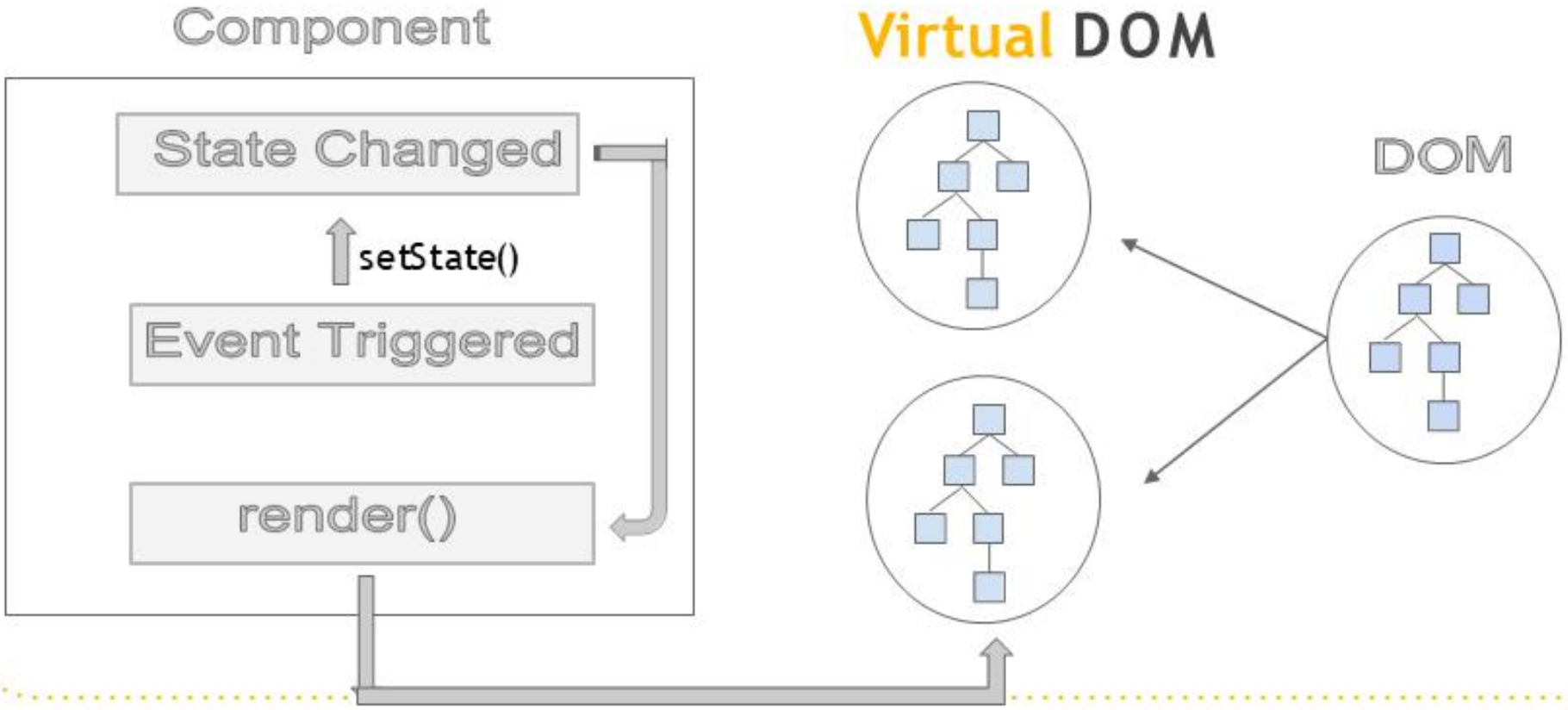


State of a Component

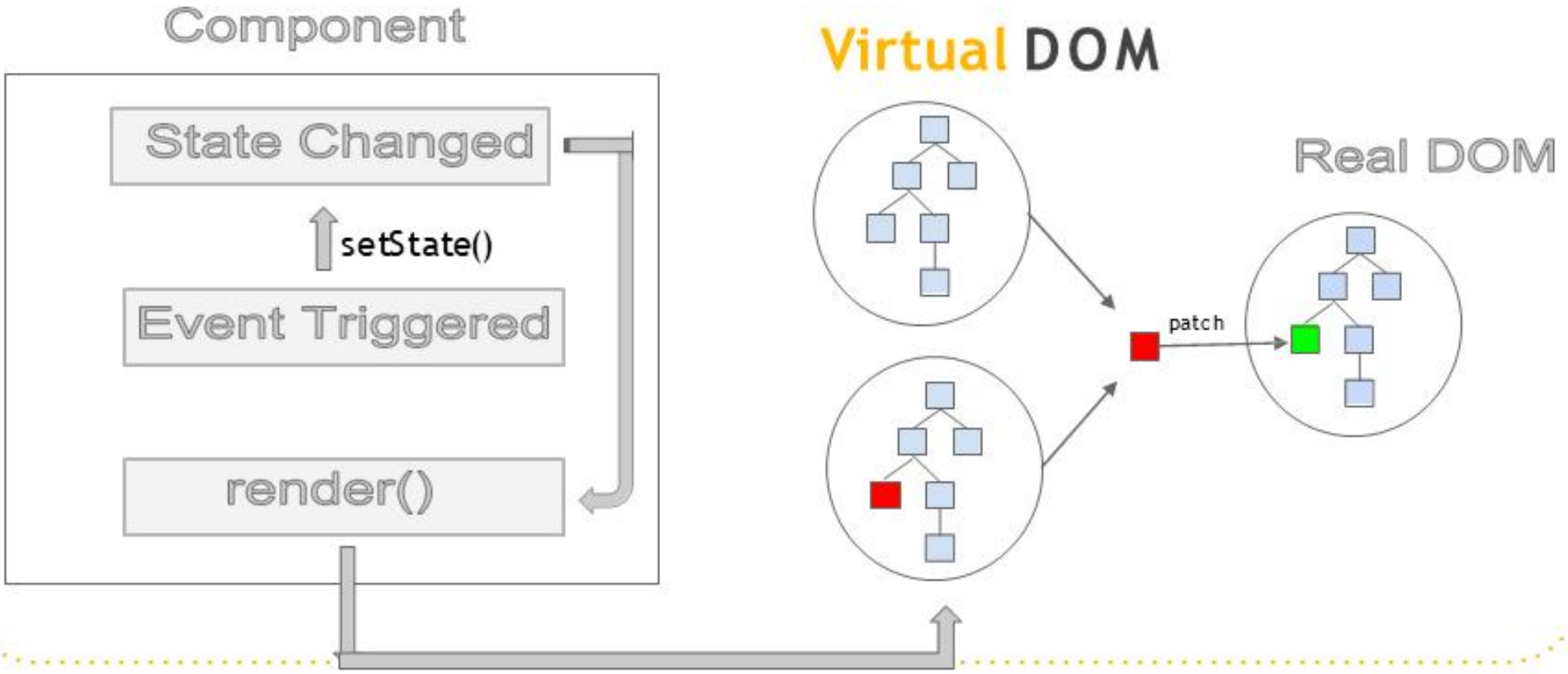




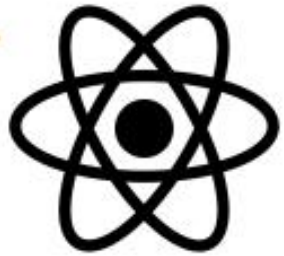
State of a Component



State of a Component

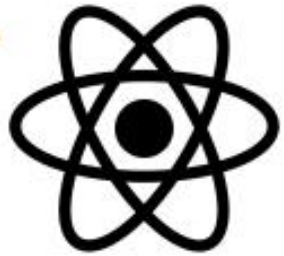


What is a component state?



- A State of a component is an object, that holds some information that may change throughout the component lifecycle.
- We define initial state and then we just have to notify that the state is changed and the react will automatically render those changes on the front end behind the scenes.
- Every time the state changes the changes get re-rendered so the UI(front end) changes automatically.

Props



- Props are inputs to components.

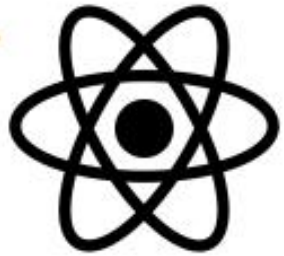
```
const element = <Welcome name="Sara" />;
```

- Single values or objects containing a set of values that are passed to components on creation, using a naming convention similar to HTML-tag attributes

- They are used to:
  1-Pass custom data to your component.
  2-Trigger state changes.

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

Important Points



- React may batch multiple setState() calls into a single update for performance.
- Because this.props and this.state may be updated asynchronously, you should not rely on their values for calculating the next state
- State is local or encapsulated. It is not accessible to any component other than the one that owns and sets it.
- A component may choose to pass its state down as props to its child components
- Imagine a component tree as a waterfall of props, each component's state is like an additional water source that joins it at an arbitrary point but also flows down

Difference between State & Props

The props(properties) and state are both JavaScript objects. Props are like an args of a function and states are like local variables inside the function

Props

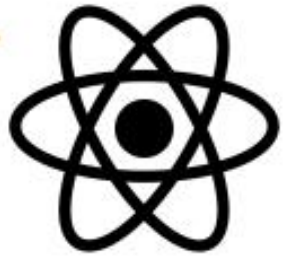
=Props is passed by the parent to the Child components

-A component should never modify its own props

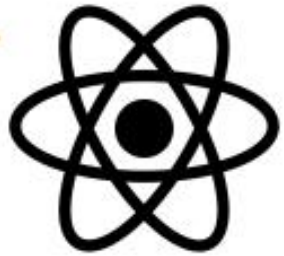
Component State

=State is managed within a component for internal communication

- State can be modified using setState() and when value of state changes render() is called.



Parent, Child & Nested Components



Create & Handle Routes

Component LifeCycle

Every React Component has a lifecycle of its own, which has different stages.

Mounting

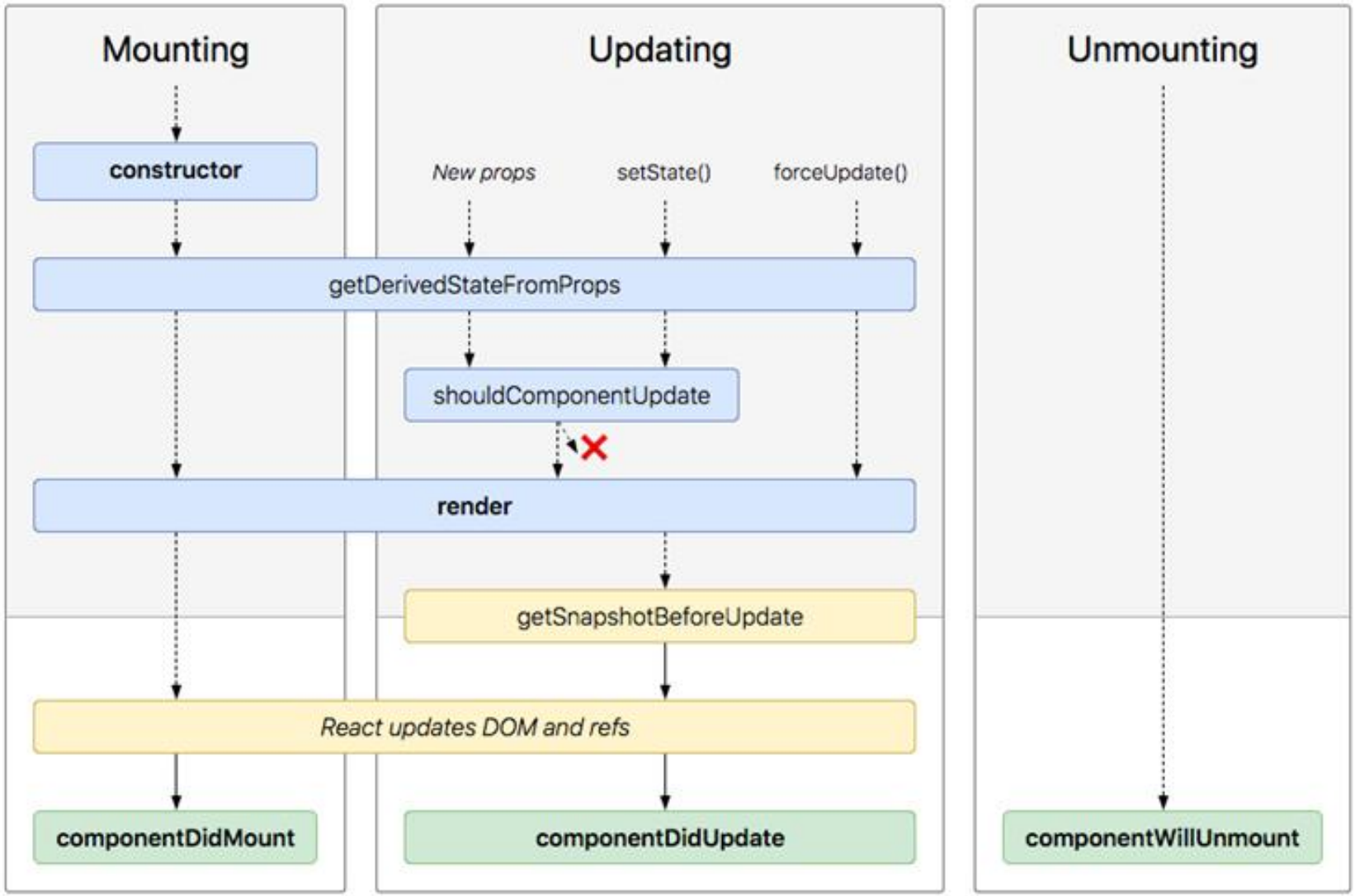
- Component is constructed with the given Props and default state.This is done inside **constructor()**
-Component came into the DOM

Updating

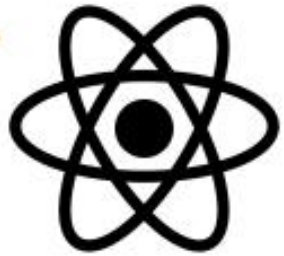
- When the state of a component is updated

Unmounting

- Component is removed from the page

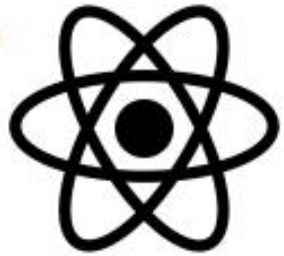

Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

Constructor



- Overrides the constructor of React.Component
- Required only when we initialize state or bind methods.
  E.g. *this.handleClick = this.handleClick.bind(this);*
- Call super(props) before any other statement.
- Do not call setState() inside it

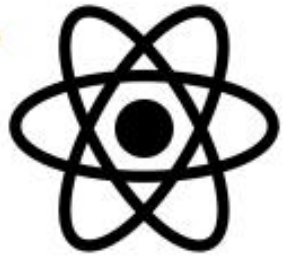
getDerivedStateFromProps



```
static getDerivedStateFromProps(props, state)
```

- Invoked right before calling the render method
- Called both on the initial mount and on subsequent updates
- Should return an object to update the state, or null to update nothing.
- Rare use cases where the state depends on changes in props

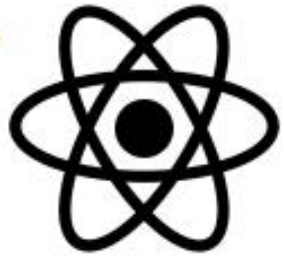
ComponentDidMount



```
componentDidMount()
```

- Load data from a remote endpoint
- Calling setState() here will trigger an extra rendering, but it will happen before the browser updates the screen.

shouldComponentUpdate



```
shouldComponentUpdate(nextProps, nextState)
```

- Exists as a performance optimization

- Not called for the initial render or when forceUpdate() is used.
- Use built-in PureComponent instead of writing shouldComponentUpdate()
- PureComponent performs a shallow comparison of props and state, and reduces the chance that you'll skip a necessary update.

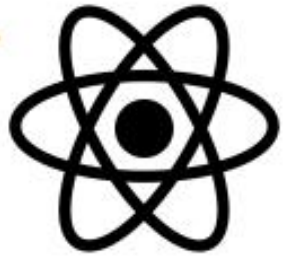
render

```
render()
```

- Required method in a class component

- Should return React Elements (JSX) or Arrays and fragments or Portals or String and numbers or Booleans or null
- Should not modify component state
- It does not directly interact with the browser.

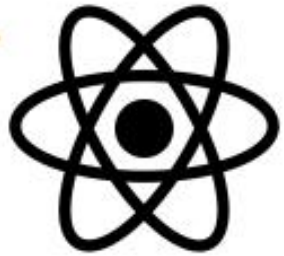
getSnapshotBeforeUpdate



```
getSnapshotBeforeUpdate(prevProps, prevState)
```

- Invoked right before the most recently rendered output is committed to e.g. the DOM
- Capture some information from the DOM (e.g. scroll position) before it is potentially changed
- Returned value will be passed as a parameter to componentDidUpdate()
- Example to use it: A chat thread that need to handle scroll position in a special way.

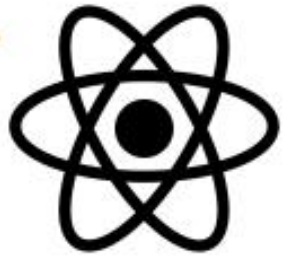
ComponentDidUpdate



```
componentDidUpdate(prevProps, prevState, snapshot)
```

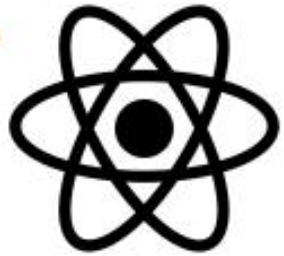
- Invoked immediately after updating occurs.
- Calling setState() here will trigger an extra rendering, but it
  will happen before the browser updates the screen.
- setState() can be called but it must be wrapped
  in a condition.
- If getSnapshotBeforeUpdate() is implemented, snapshot
  param will be available, else undefined.

ComponentWillUnmount

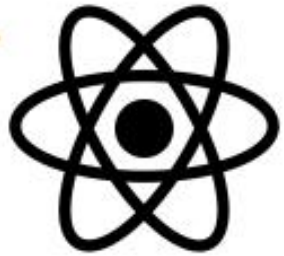


```
componentWillUnmount()
```

- invoked immediately before a component is unmounted.
- For e.g. Invalidating timers, canceling network requests, or
  cleaning up any subscriptions.
- should not call setState() as component will never be rerendered.

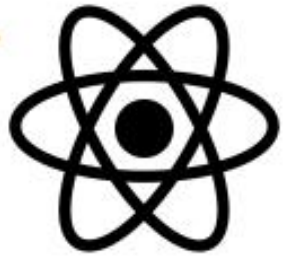


Handling forms and input values



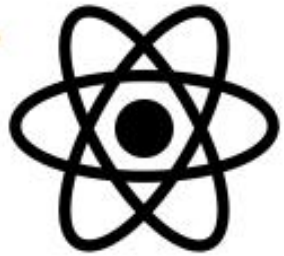
Arrow functions and Spread Operators

Reconciliation



- When a component's props or state change,
  React decides whether an actual DOM update is necessary by comparing,
  newly returned element =previously rendered one.?
  When they are not equal, React will update the DOM

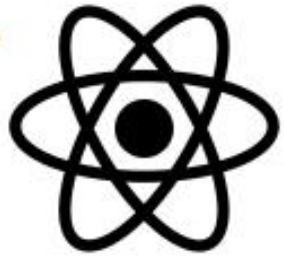
HOC



- A function that takes a component and returns a new component.
- Advanced technique in React for reusing component logic.
- Not part of the React API
- A pattern that emerges from React's compositional nature.

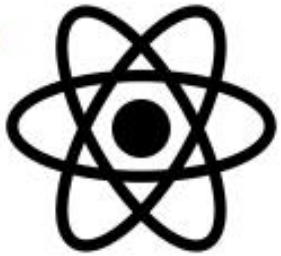
```
const EnhancedComponent = higherOrderComponent(WrappedComponent);
```

Pure Components



- **React.PureComponent** implements it with a shallow prop and state comparison and does not re-render if the they don't change.
- So it handles shouldComponentUpdate() for you
- Does not re-render if nextState = prevState

Memo

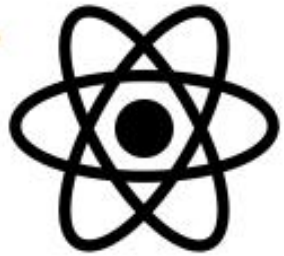


- **React.memo** is a higher order component.

- Similar to React.PureComponent but for functional based components instead of classes.
- If your function component renders the same result given the same props, you can wrap it in a call to React.memo

```
const MyComponent = React.memo(function MyComponent(props) {
  /* render using props */
});
```

Use of Refs



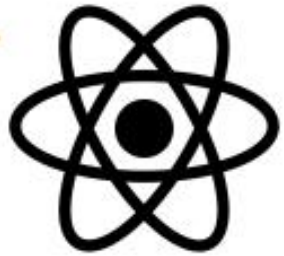
- **React.createRef** creates a **reference** that can be attached to React elements via the **ref attribute.**

```
this.inputRef = React.createRef();
```

- Avoid using jQuery as you have to import the jQuery library. as we're going to manipulate the dom
- Compared to using jQuery, elements can be faster referenced in a React way using Refs
- Refs are not present during the initial render. You can initialize it in constructor. However you must use it inside one of the React's lifecycle events such as componentDidMount or componentDidUpdate.

```
class MyComponent extends React.Component {
  constructor(props) {
    super(props);

    this.inputRef = React.createRef();
  }

  render() {
    return <input type="text" ref={this.inputRef} />;
  }

  componentDidMount() {
    this.inputRef.current.focus();
  }
}
```

Context

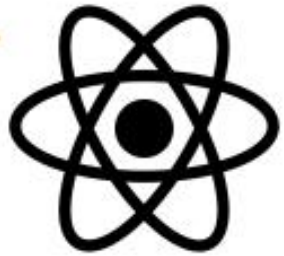


- Universal centralized data for your application.
- Used to share data, such as the current authenticated user, theme, or preferred language, between components.

- We can avoid passing props through intermediate elements

```
const MyContext = React.createContext(defaultValue);
```

1-Create Context



```
const DataContext = React.createContext( defaultValue: {
    hobby: 'Blogging',
    employer: 'rtCamp',
    handleHobbyChange() {},
    handleEmploymentChange() {}
});

export const Provider = DataContext.Provider;
export const Consumer = DataContext.Consumer;
```

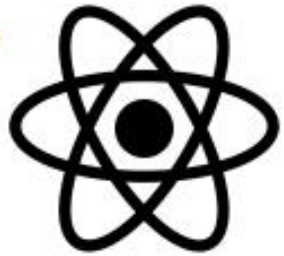
2-Wrap in Provider and pass the state to child component

```
import { Provider } from "./component/DataContext";
import Home from "./component/Home";

class App extends Component {

    constructor( props ) {
        super( props );
        this.state = {
            hobby: '',
            employer: '',
            handleHobbyChange: this.handleHobbyChange,
            handleEmploymentChange: this.handleEmploymentChange,
        }
    }

    handleHobbyChange = ( event ) => {...};

    handleEmploymentChange = ( event ) => {...};

    render() {
        return (
            <div className="App">
                <Provider value={this.state}>
                    <Home/>
                </Provider>
            </div>
        );
    }
}
```

3-Wrap in Consumer and receive context in Child Components

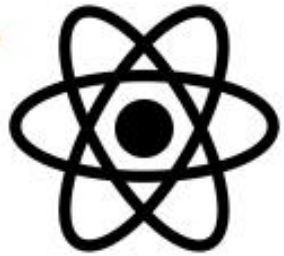
```
class Personal extends React.Component {

    render() {
        return(
            <Consumer>
                { context => (
                    <div>
                        <h2>Personal Component</h2>
                        <label htmlFor="hobby">
                            Hobby:
                            <input type="text" name="hobby" id="hobby" onChange={context.handleHobbyChange}/>
                            <h5>{ context.hobby }</h5>
                        </label>
                    </div>
                ) }
            </Consumer>

        );
    }
}
```

Pros of Context



- Set context on top and all the components in the middle don't have to know anything about it
- The one down at the bottom can access it
- Fills the same need as redux.

Cons of Context



- Makes debugging difficult
- Difficult to figure out what caused the error when you cannot see the data in the child components that is not importing Context
- You have to look at all the components which are Consuming it to figure out which one of them caused the problem
- Use context only when you have to.

Difference between Component state & Store state

Component State

=State is managed within a component

- State value can change, and then render function is called.

Store state

=Store State is updated, by the reducer, when an action is triggered.

Thank You!