

edureka!



React Components

Agenda



React
Components



Props



State



Flow Of Stateless
& Stateful
Components



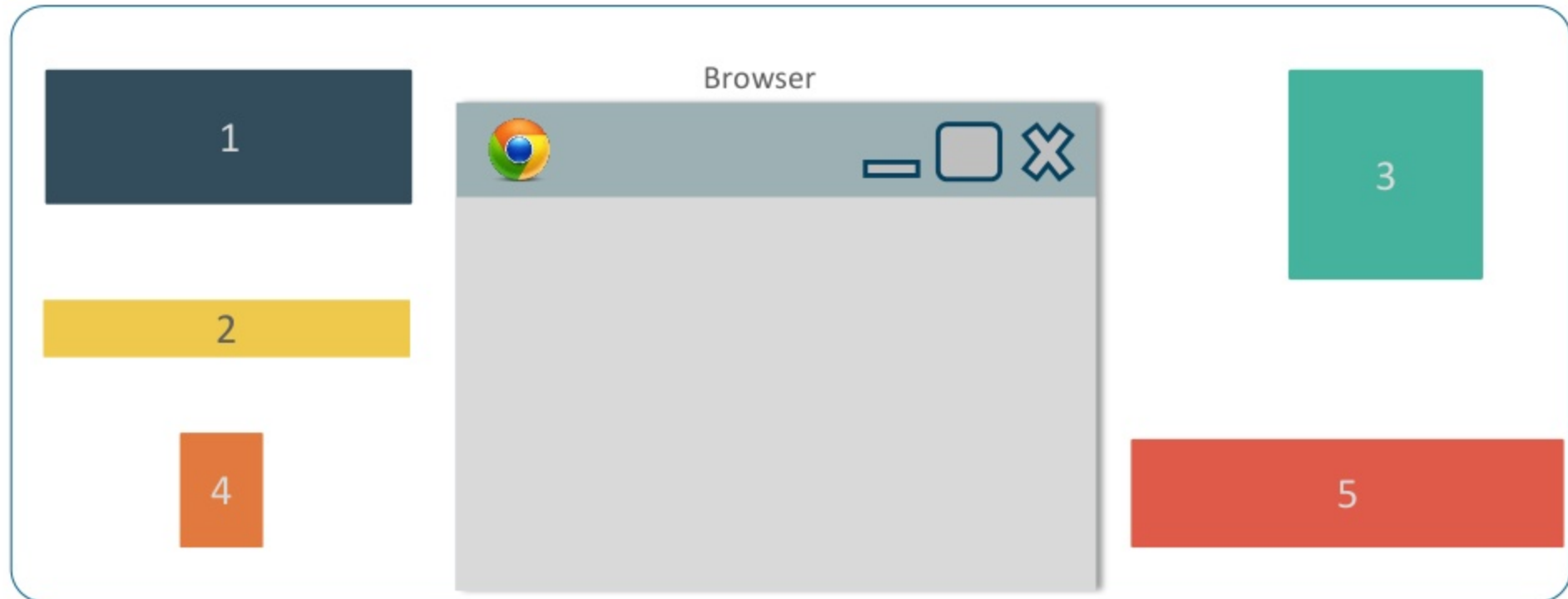
Life Cycle Of
Components



React Components

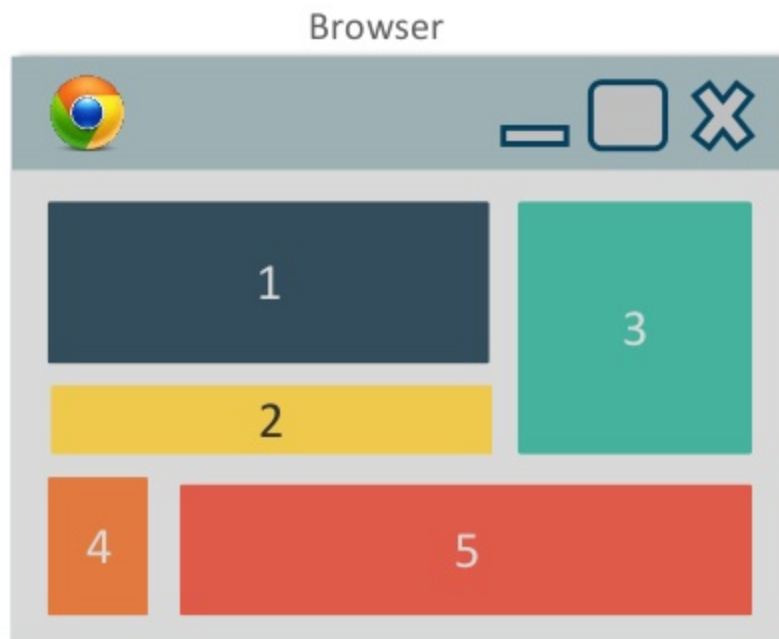
React Components

In React everything is a component



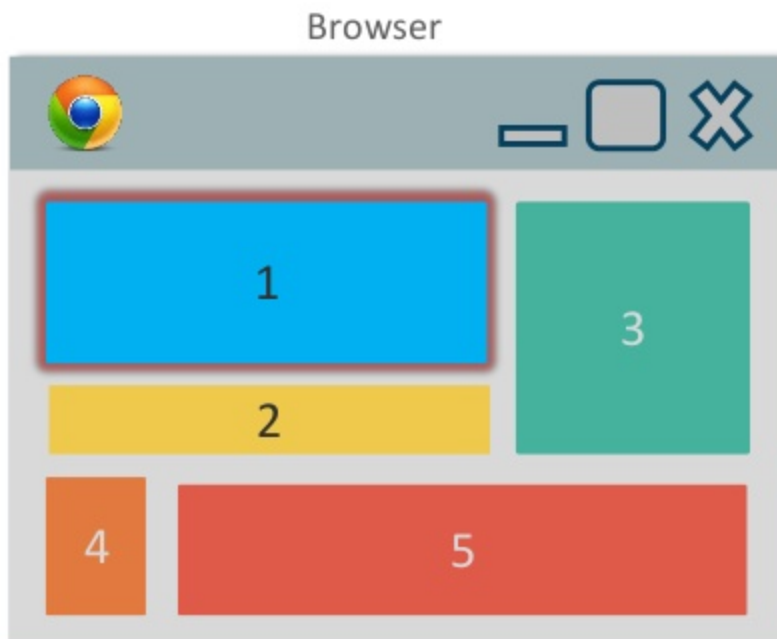
React Components

All these components are integrated together to build one application



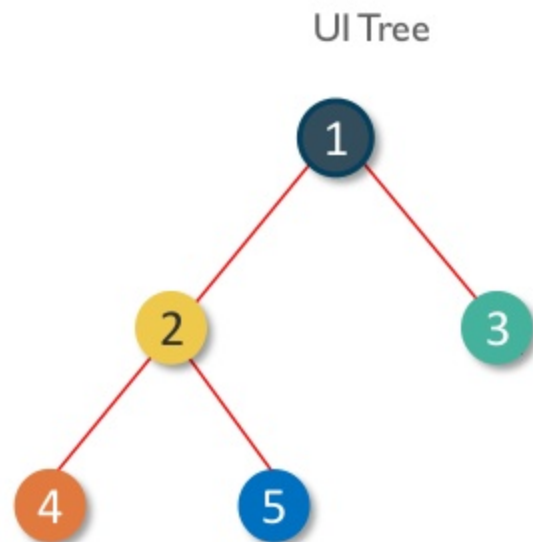
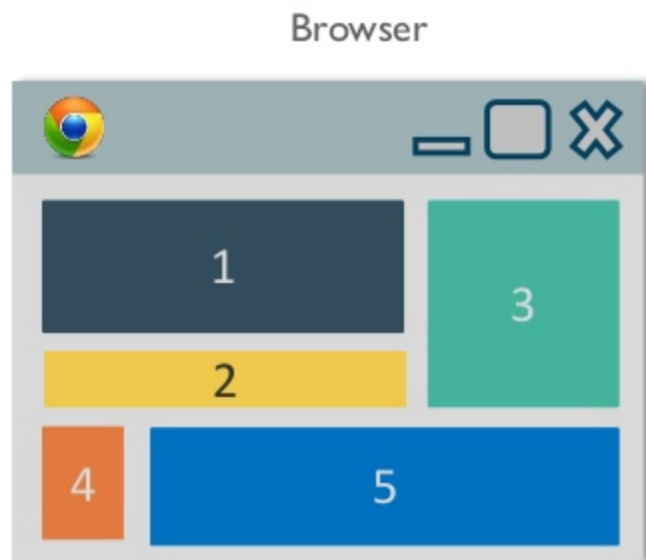
React Components

We can easily update or change any of these components without disturbing the rest of the application



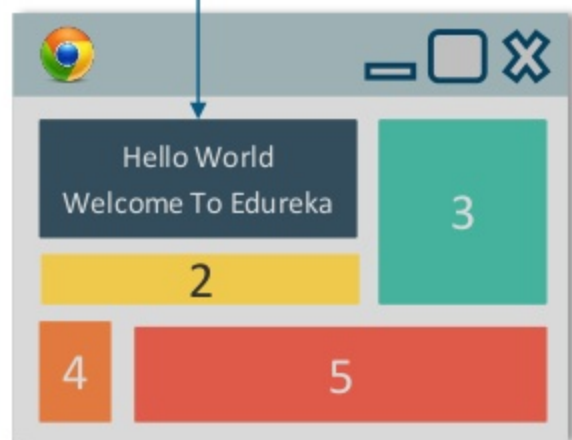
React Components – UI Tree

Single view of UI is divided into logical pieces. The starting component becomes the root and rest components become branches and sub-branches.



ReactJS Components – Sample Code

Each component returns ONE DOM element, thus JSX elements must be wrapped in an enclosing tag



```
class Component1 extends React.Component {  
  render() {  
    return (  
      <div> ← Enclosing Tag  
        <h2>Hello World</h2>  
        <h1>Welcome To Edureka</h1>  
      </div> ← Enclosing Tag  
    );  
  }  
}  
  
ReactDOM.render(  
  <Component1 />, document.getElementById('content')  
);
```


React Components

React components are controlled either by Props or States

PROP



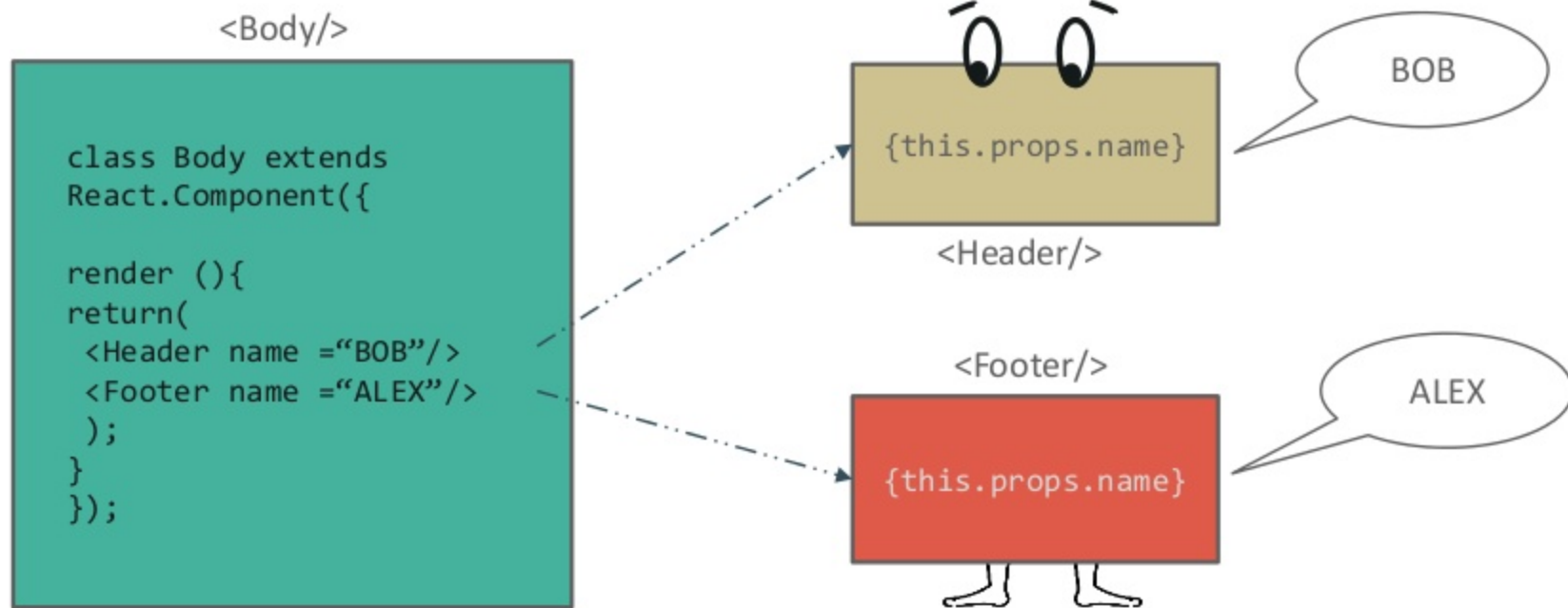
STATE



Props

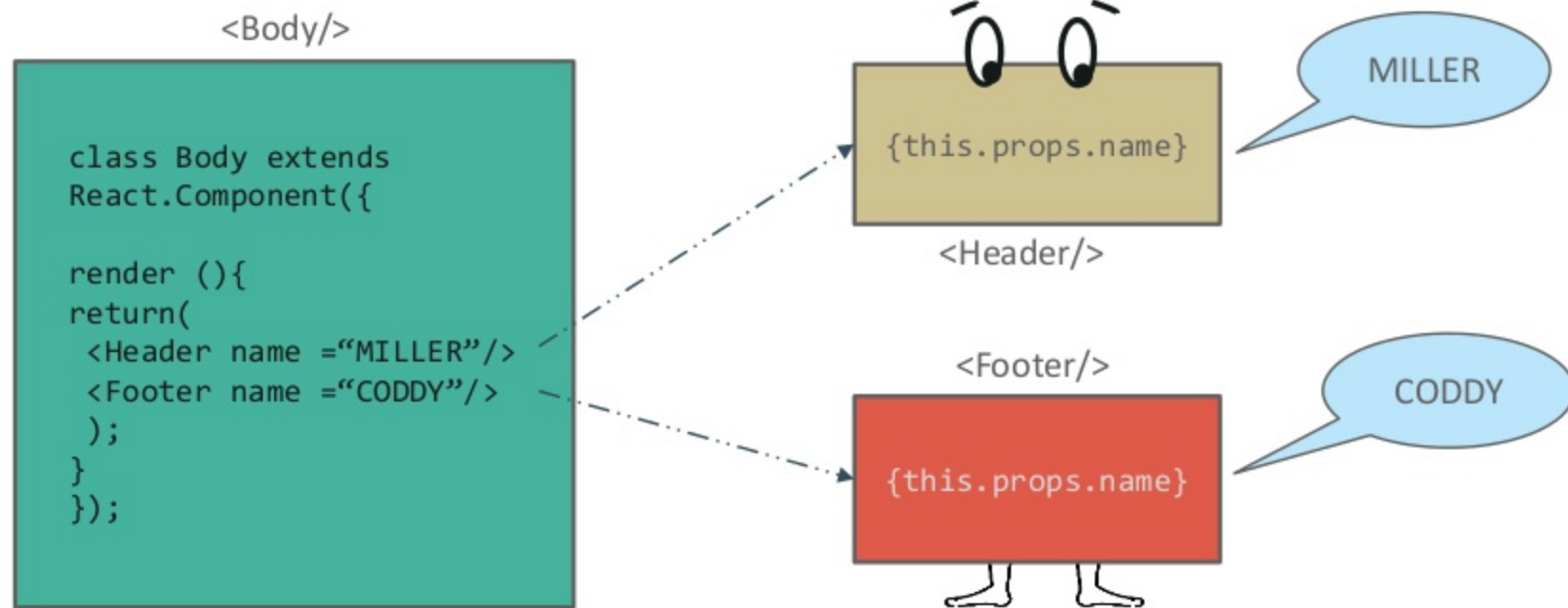
Props

Props help components converse with one another.



Props

Using Props we can configure the components as well



Props

1

Work similar to HTML attributes

2

Data flows downwards from the parent component

3

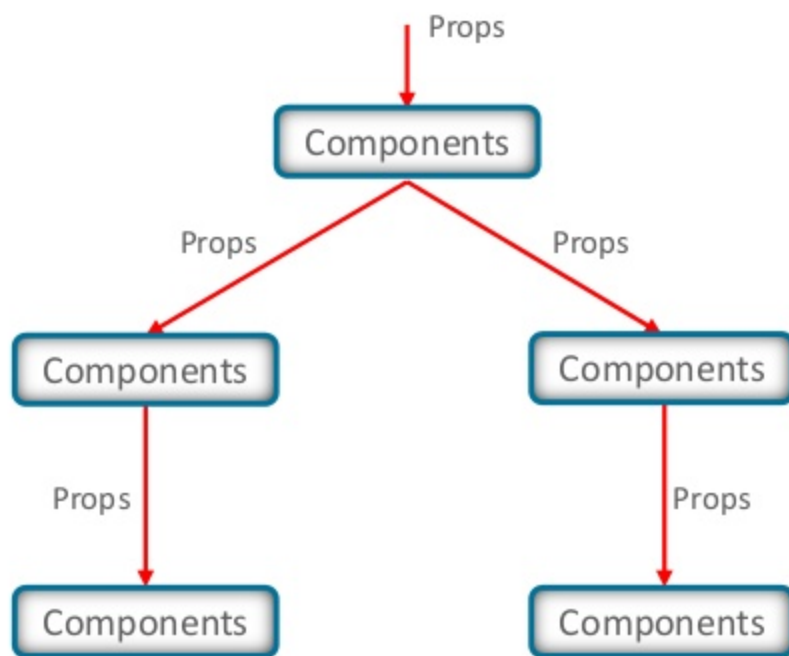
Uni-directional data flow

4

Props are immutable i.e pure

5

Can set default props



Props

ES5

Parent Component

```
var Body = React.createClass(  
  {  
    render: function() {  
      return (  
        <div>  
          <h1> Hello World from Edureka!!</h1>  
  
          <Header name = "Bob"/>  
          <Header name = "Max"/>  
          <Footer name = "Allen"/>  
        </div>  
      );  
    }  
  }  
);
```

} Sending Props

Props

ES5

The diagram illustrates the flow of props in ES5. It shows two component definitions, `Header` and `Footer`, both using `React.createClass`. Each component's `render` function returns an `<h2>` element with a `name` prop, represented as `{this.props.name}`. Dashed boxes highlight these prop expressions. Arrows point from these expressions to a label 'Receiving Props'. Another set of arrows points from the `Header` and `Footer` definitions to a label 'Child Components'. The code concludes with `ReactDOM.render` rendering the `<Body>` into a container with the ID 'container'.

```
var Header = React.createClass({
  render: function () {
    return(
      <h2>Head Name: {this.props.name} </h2>
    );
  }
});
var Footer = React.createClass({
  render: function () {
    return(
      <h2>Footer Name: {this.props.name}</h2>
    );
  }
});
ReactDOM.render(
  <Body/>, document.getElementById('container')
);
```



State

State

Components can change, so to keep track of updates over the time we use state



ICE



State changes because of some event



— — — — —
Increase In
Temperature →



WATER



State

1

Unlike Props, component States are mutable

2

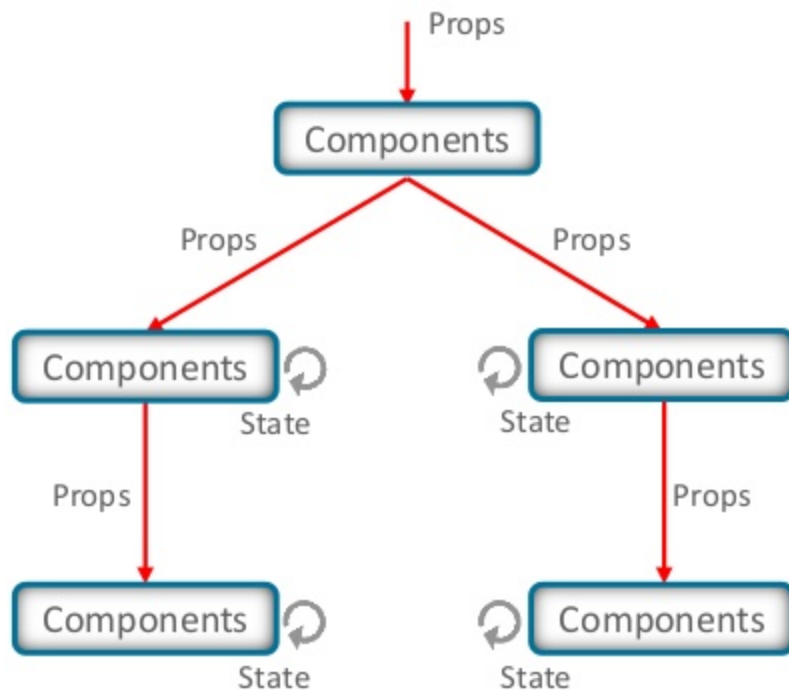
Objects which control components rendering and behavior

3

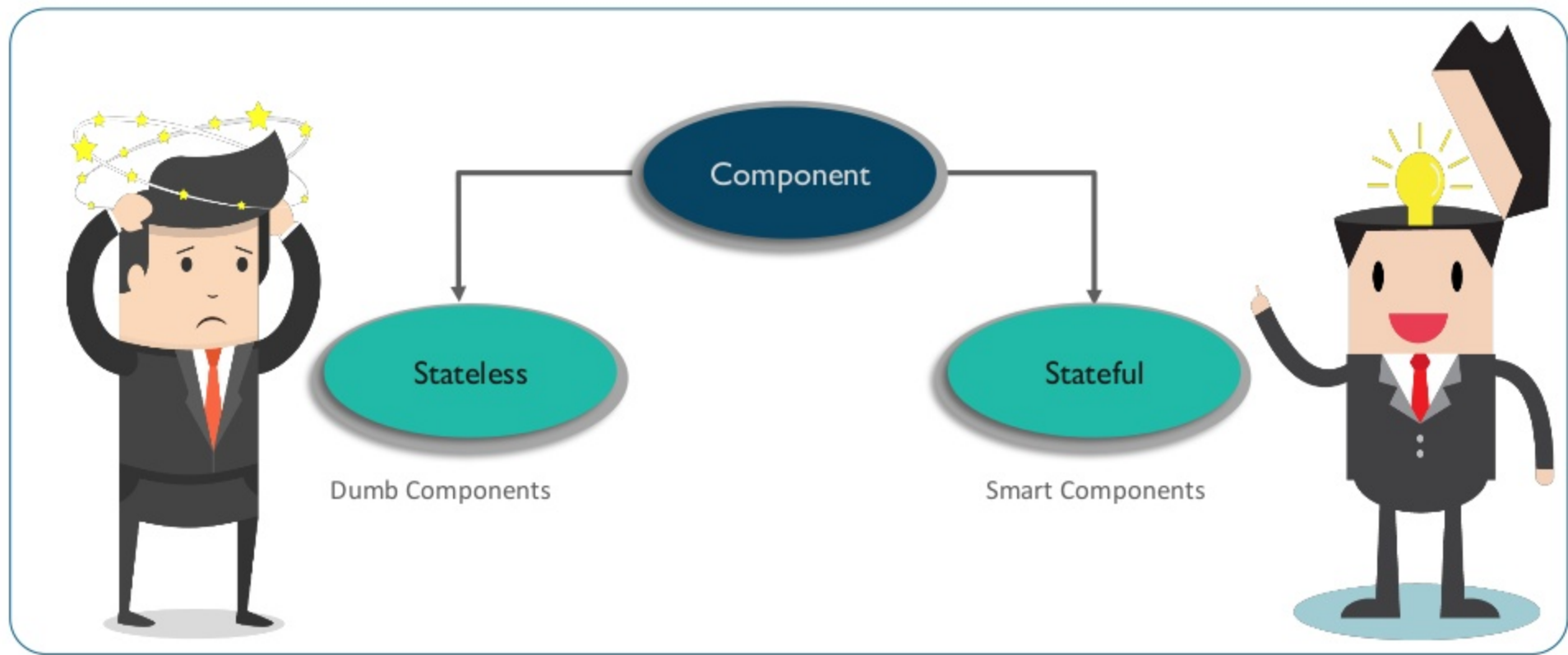
Core of React Components

4

Must be kept simple



State



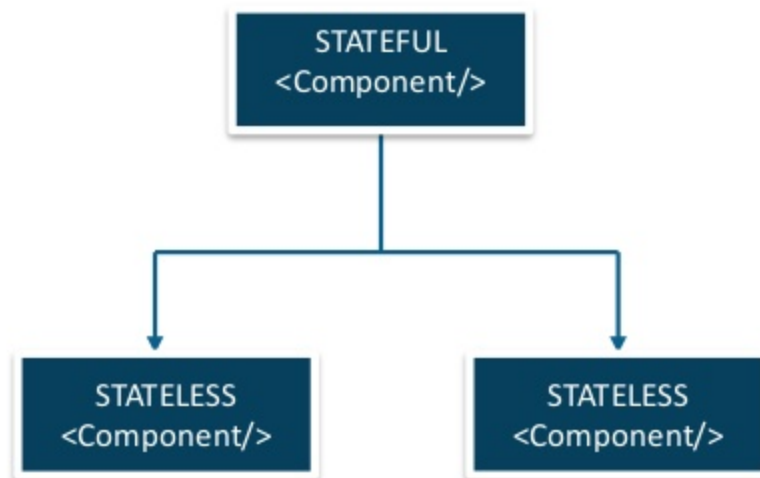
State

Stateless

- Calculates states internal state of components
- Contains no knowledge of past, current and possible future state changes

Stateful

- Core which stores information about components state in memory
- Contains knowledge of past, current and possible future state changes





Flow Of Stateless & Stateful Components

Flow Of Stateless & Stateful Components

ES6

```
class MyApp extends React.Component {  
  constructor(props) {  
    super(props);  
  
    this.state = { isLoggedIn: false };  
  }  
  
  receiveClick() {  
    this.setState({ isLoggedIn: !this.state.isLoggedIn });  
  }  
}
```

Parent Component

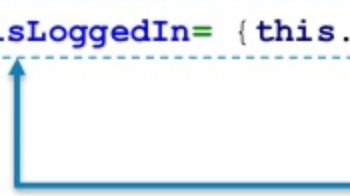
Setting Initial State

Changing State

Flow Of Stateless & Stateful Components

ES6

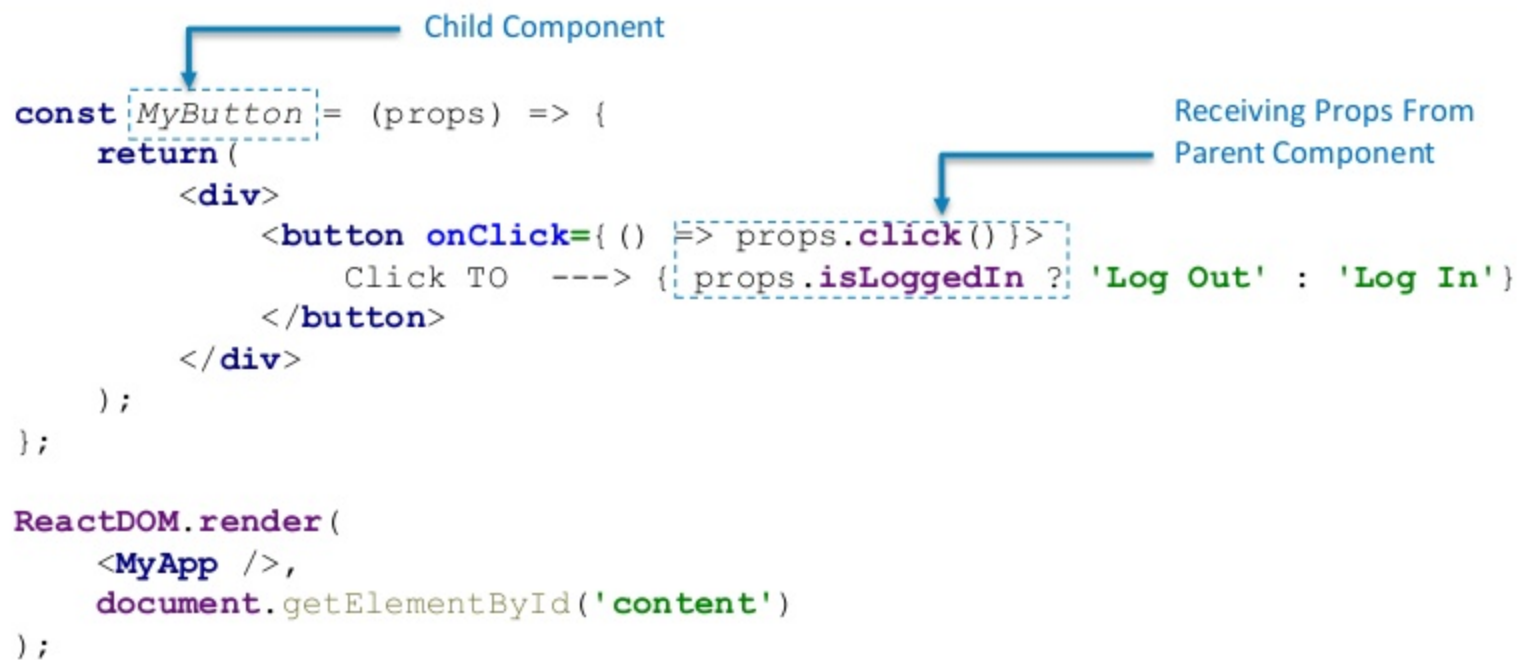
```
render() {  
  return(  
    <div>  
      <h1>Hello.. I am Stateful Parent Component</h1><br/>  
      <p>I monitor if my user is logged in or not!!</p> <br/>  
      <p>Let's see what is your status : <h2><i>{this.state.isLoggedIn ?  
        'You Are Logged In' : 'Not Logged In'}</i></h2></p><br/>  
      <h2>Hi, I am Stateless Child Component</h2>  
      <p>I am a Logging Button</p><br/>  
      <p><b>I don't maintain state but I tell parent component if user clicks me  
<MyButton click={this.receiveClick.bind(this)} isLoggedIn= {this.state.isLoggedIn}/>  
      </b></p><br/>  
    </div>  
  );  
}
```

A blue arrow originates from the `isLoggedIn` prop in the `<MyButton>` component call and points upwards to the `isLoggedIn` prop in the `<h2>` component call, illustrating the flow of data from the child back to the parent.

Passing Props To Child Component

Flow Of Stateless & Stateful Components

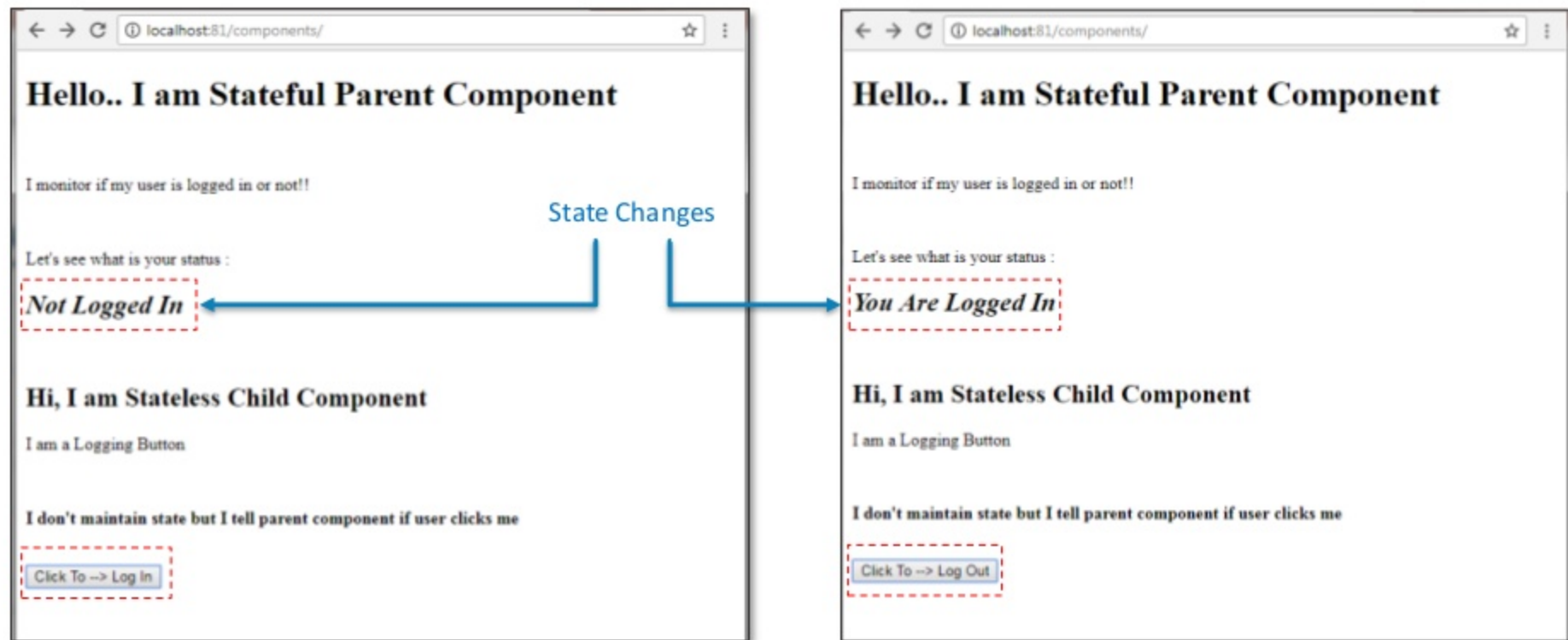
ES6



The diagram illustrates the flow of props from a parent component to a child component. A blue arrow labeled "Child Component" points to the `MyButton` variable in the code. Another blue arrow labeled "Receiving Props From Parent Component" points to the `props` parameter in the function definition and the `props.isLoggedIn` property access in the button's `onClick` handler.

```
const MyButton = (props) => {  
  return (  
    <div>  
      <button onClick={() => props.click()}>  
        Click TO ---> {props.isLoggedIn ? 'Log Out' : 'Log In'}  
      </button>  
    </div>  
  );  
};  
  
ReactDOM.render(  
  <MyApp />,  
  document.getElementById('content')  
);
```

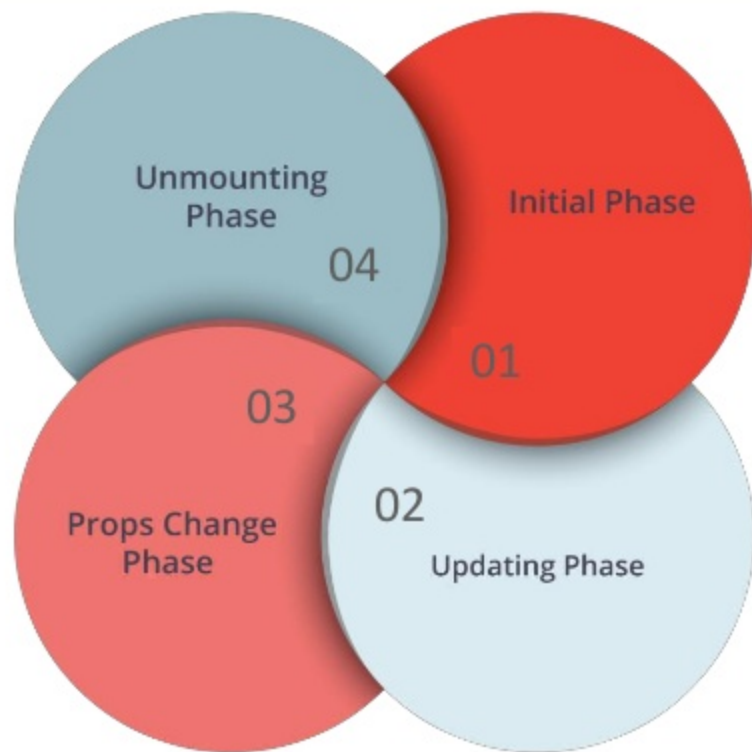

Flow Of Stateless & Stateful Components





Component Lifecycle

Lifecycle Phases

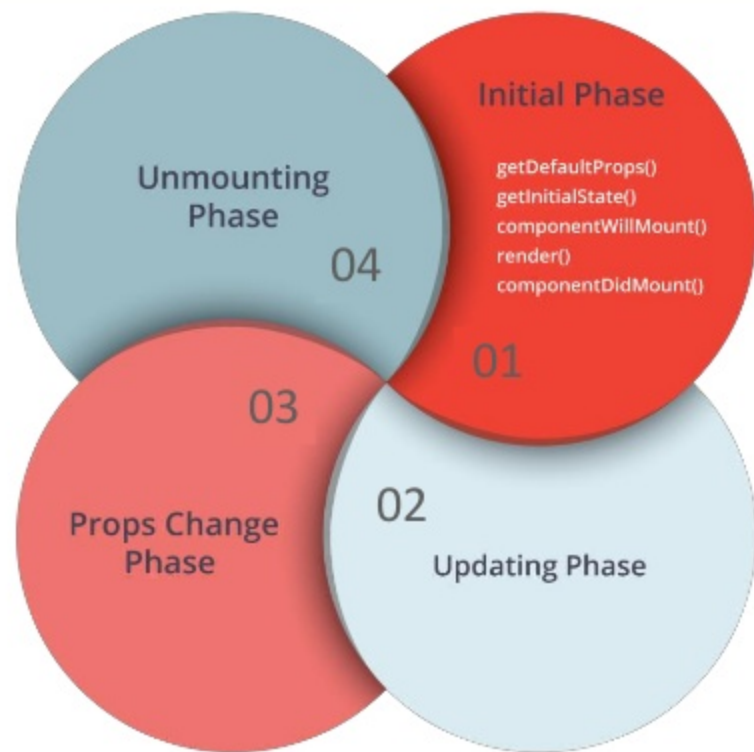


Lifecycle methods notifies when certain stage of the lifecycle occurs

Code can be added to them to perform various tasks

Provides a better control over each phase

Lifecycle Phases

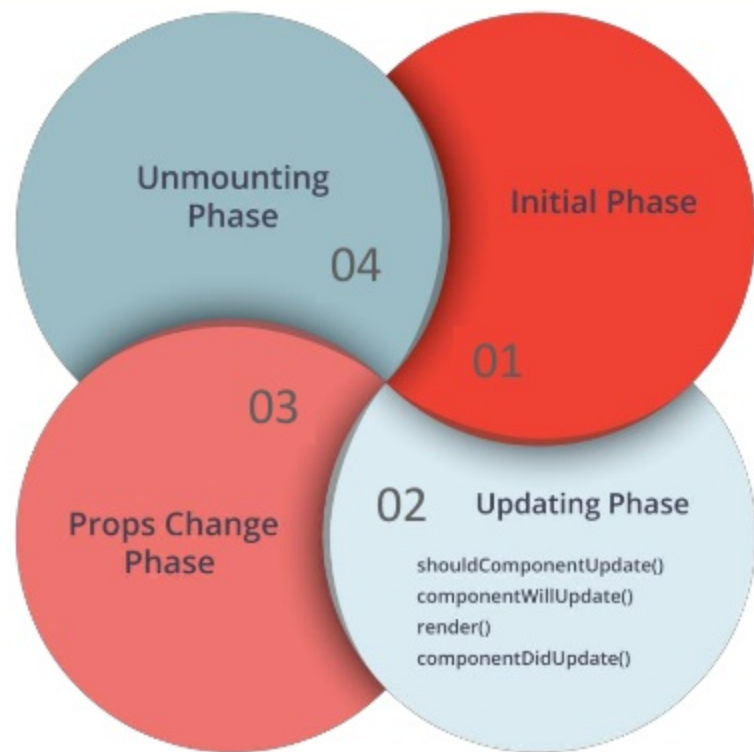


In this phase,
component is
about to make its
way to the DOM

1



Lifecycle Phases

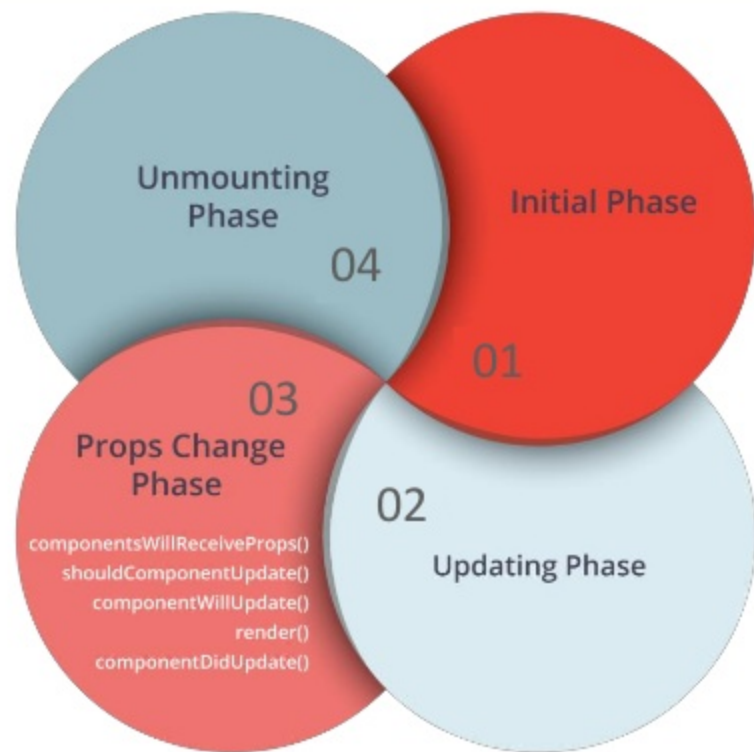


In this phase, component can update & re-render when a **state** change occurs

2



Lifecycle Phases

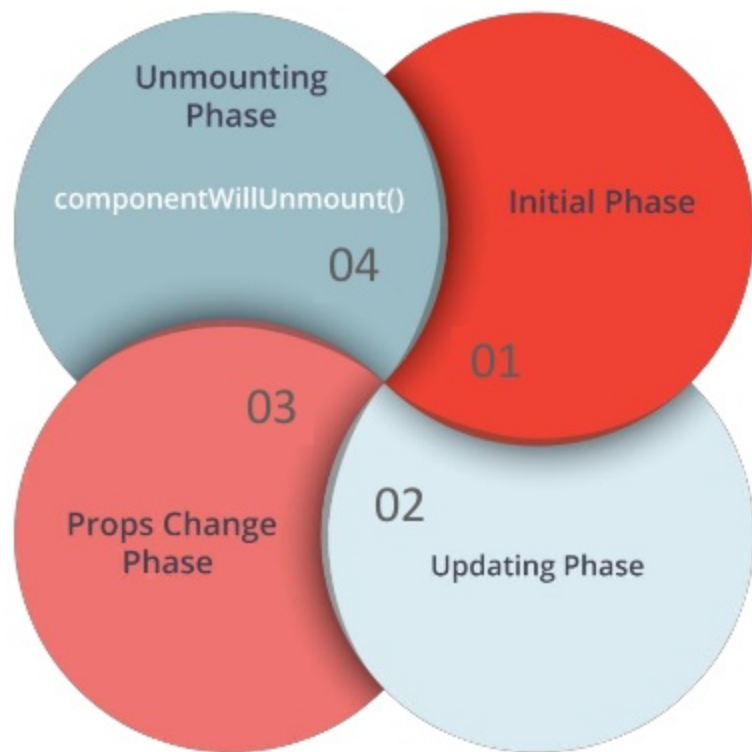


In this phase, component can update & re-render when a **prop** change occurs

3



Lifecycle Phases

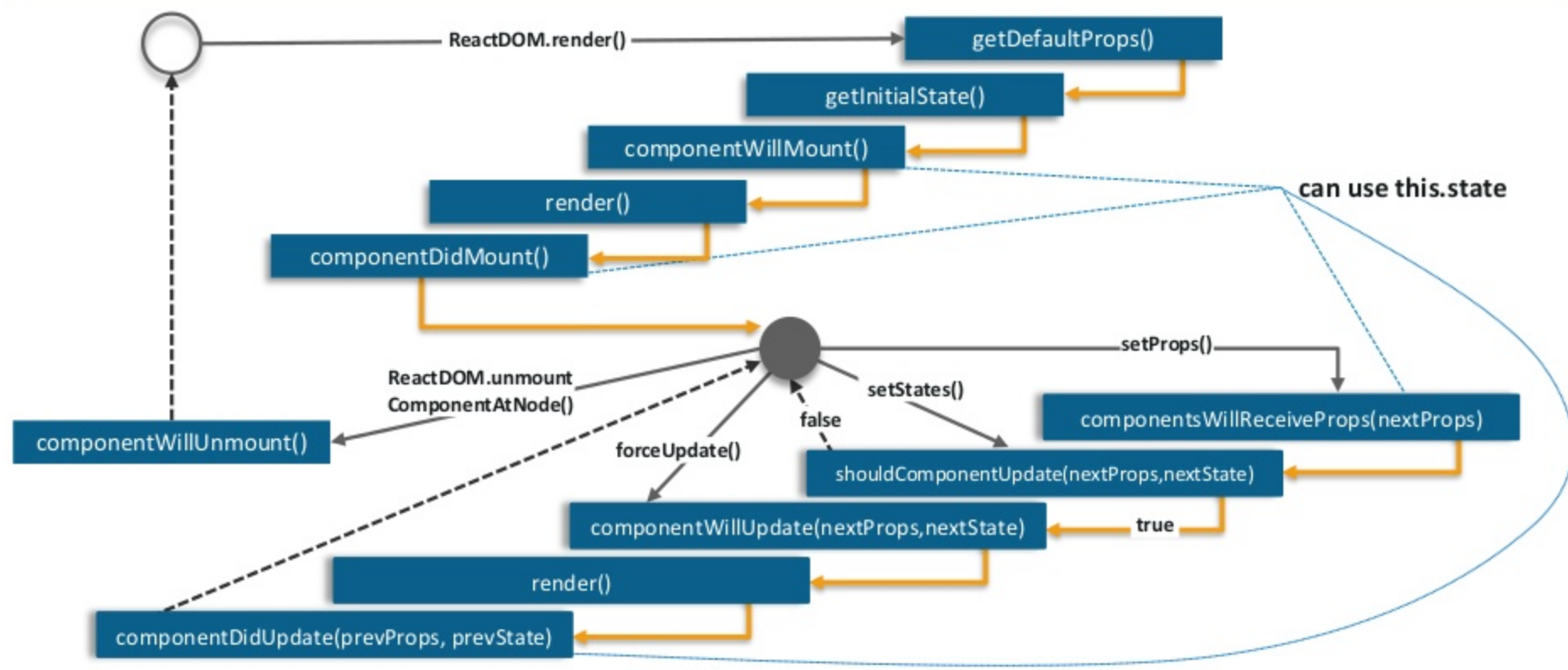


In this phase, the component is destroyed and removed from DOM

4



Component Lifecycle In A Glance





Thank You

👍 Like

💬 Comment

🔗 Share

For more information please visit our website
www.edureka.co