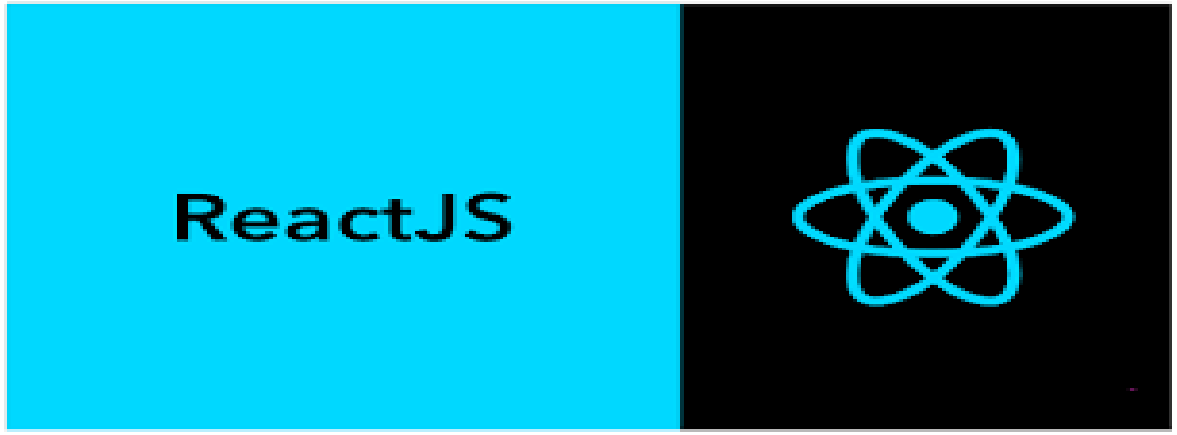


Karthik Raman

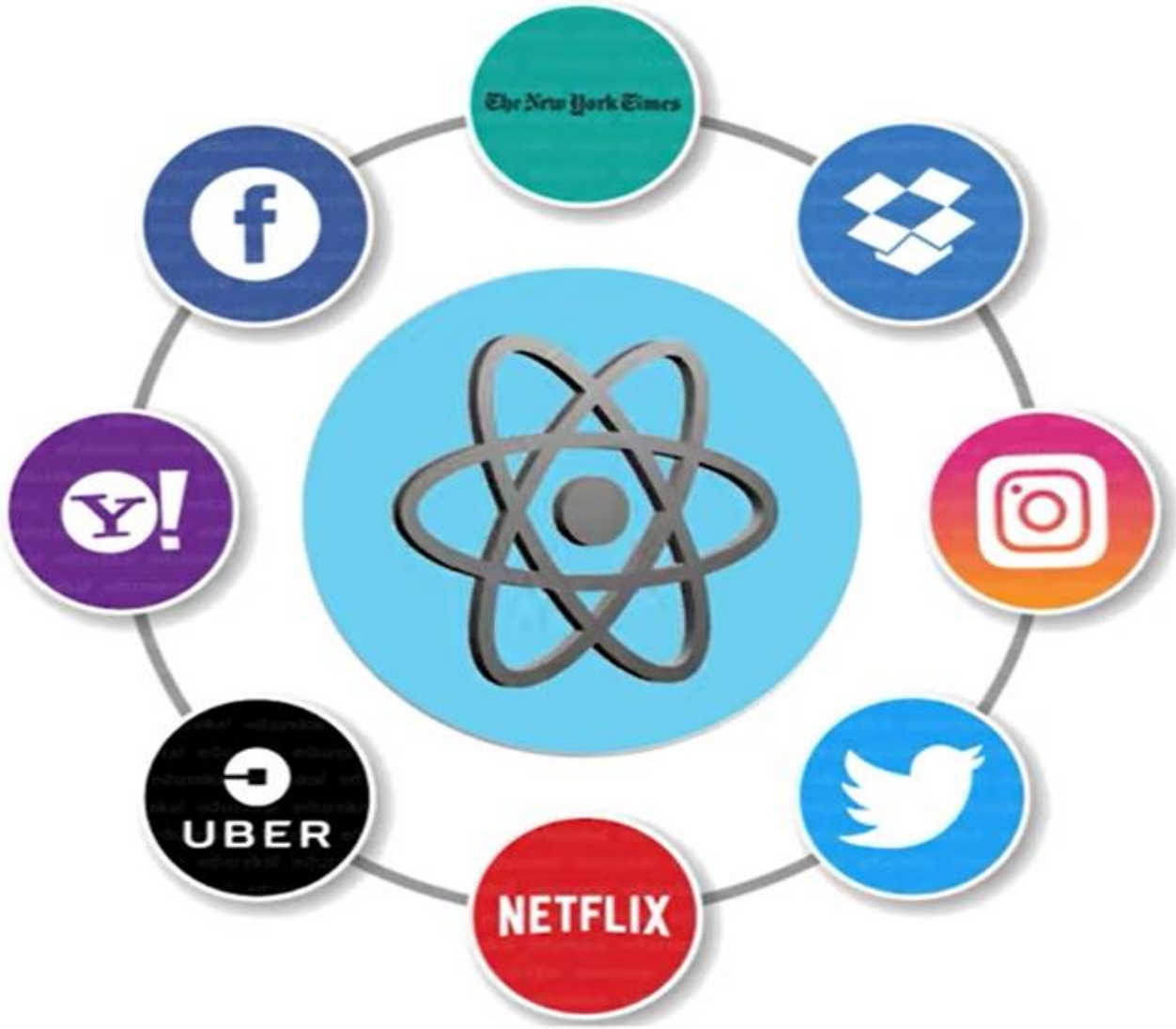
JavaScript Library

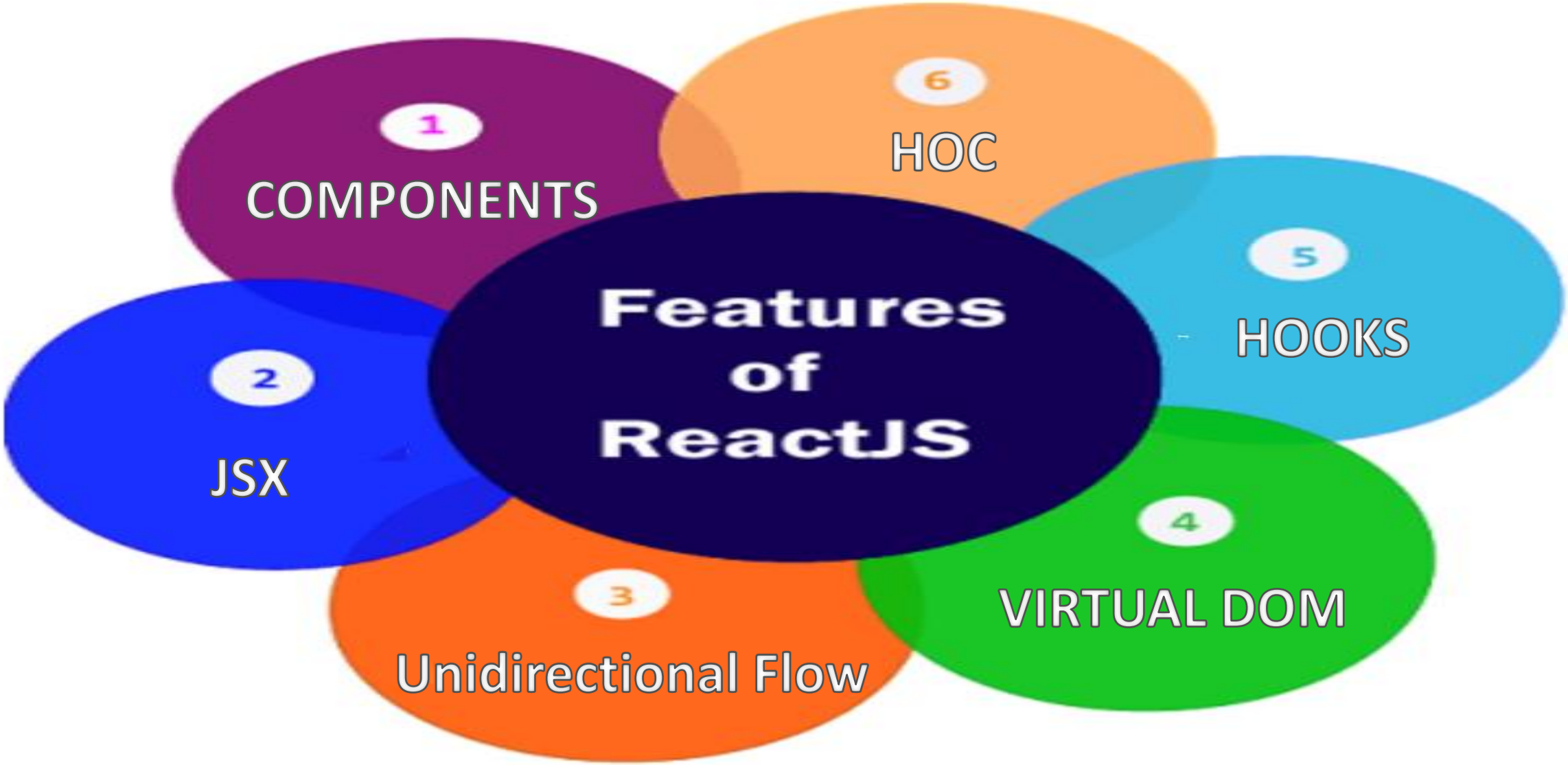
Developed by **Facebook**

Open Source

Used to develop Interactive **User Interfaces**

APPLICATIONS OF REACT.JS





Routers **Redux** **Axios**

- Components are the building blocks of React.
- Think of a component as a collection of HTML, CSS, JS
- Components can be written in Pure Javascript or JSX

Pure Javascript

Creating the UI Description

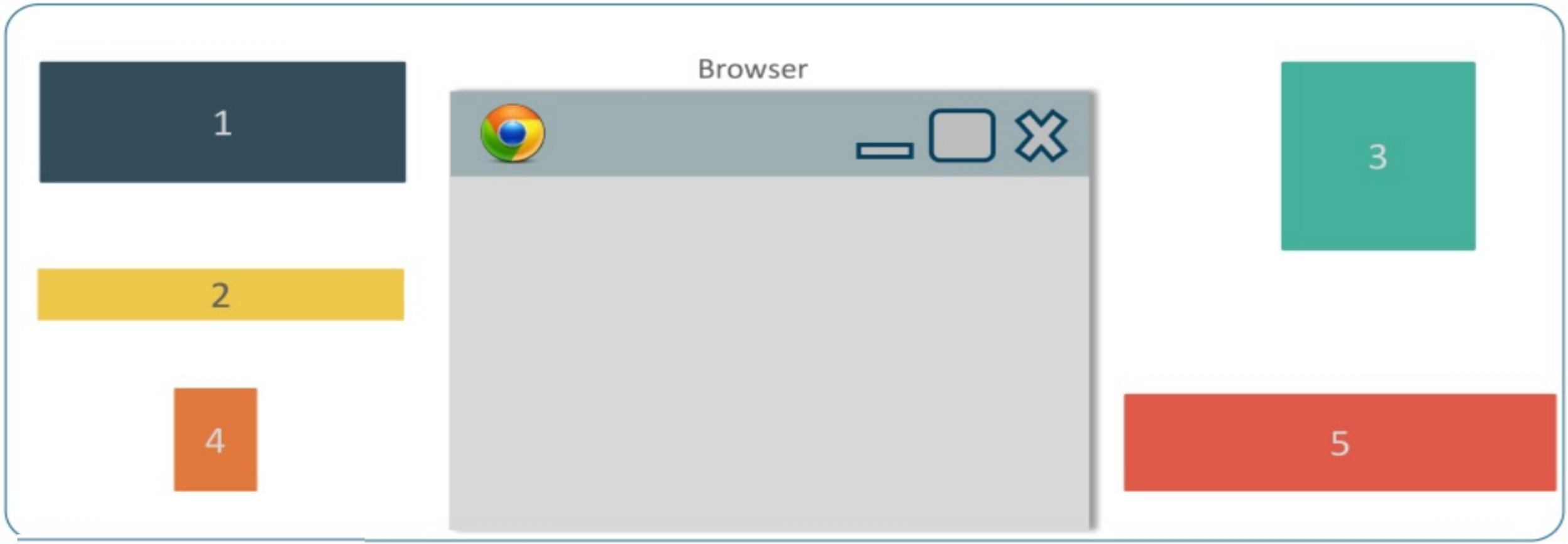
```
var child1 = React.createElement('li', null, 'First Text Content');
var child2 = React.createElement('li', null, 'Second Text Content');
var root = React.createElement('ul', { className: 'my-list' }, child1,
child2);
React.render(root, document.getElementById('example'));
```

First-R.html

First-RC.html

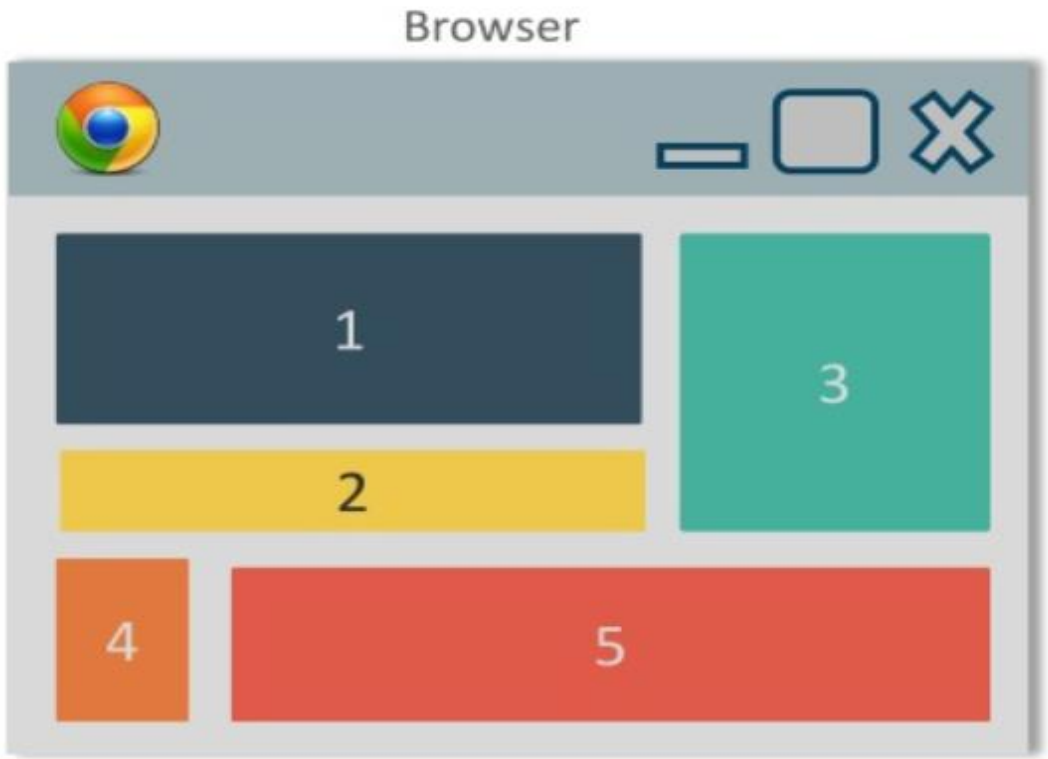
React Components

In React everything is a component



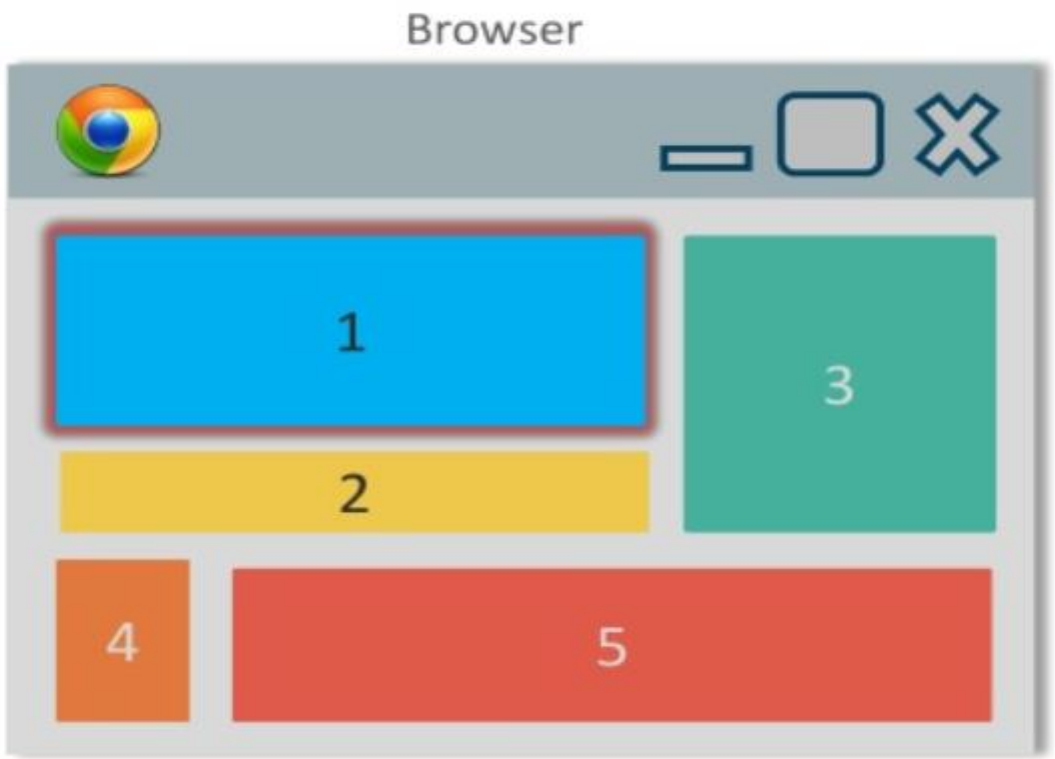
React Components

All these components are integrated together to build one application



React Components

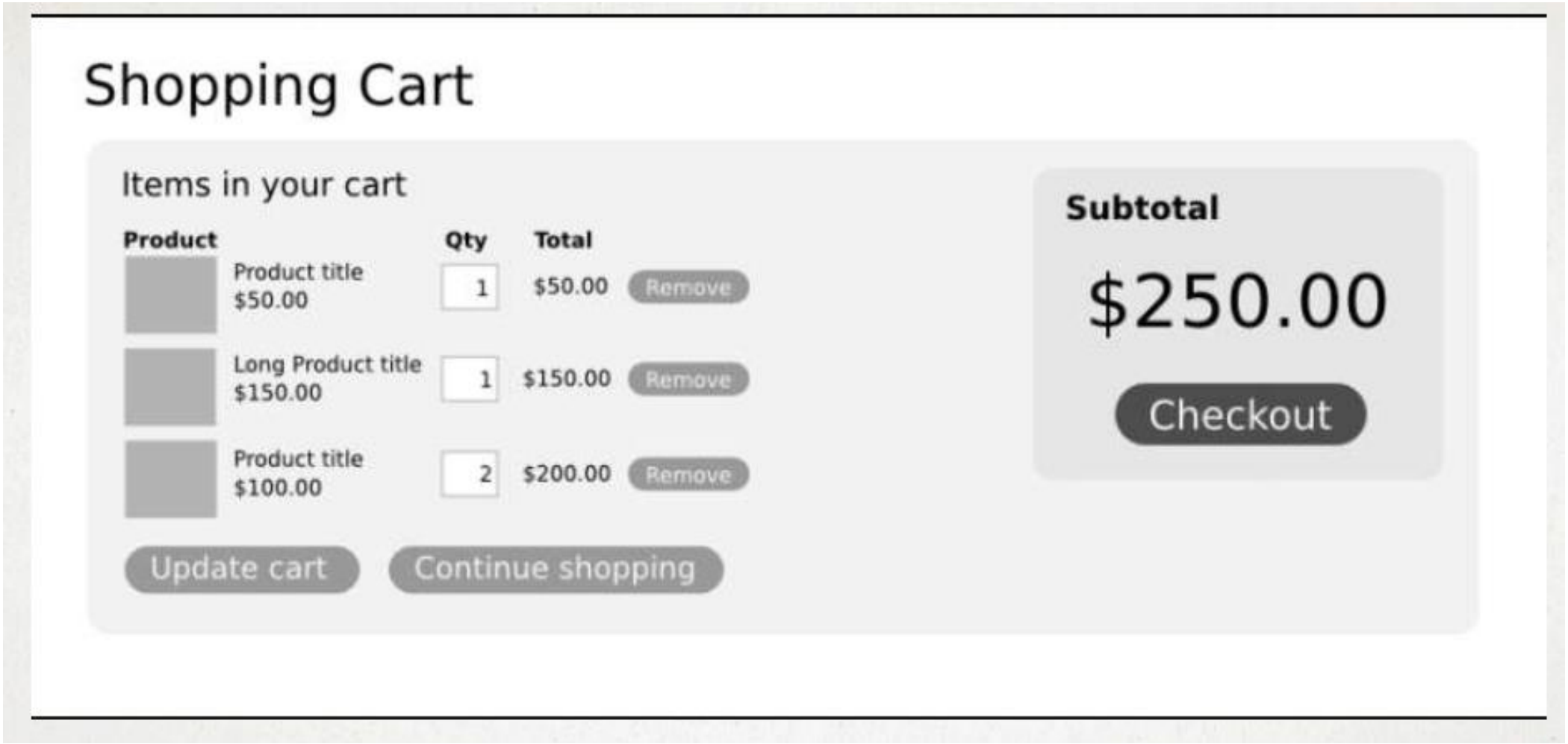
We can easily update or change any of these components without disturbing the rest of the application

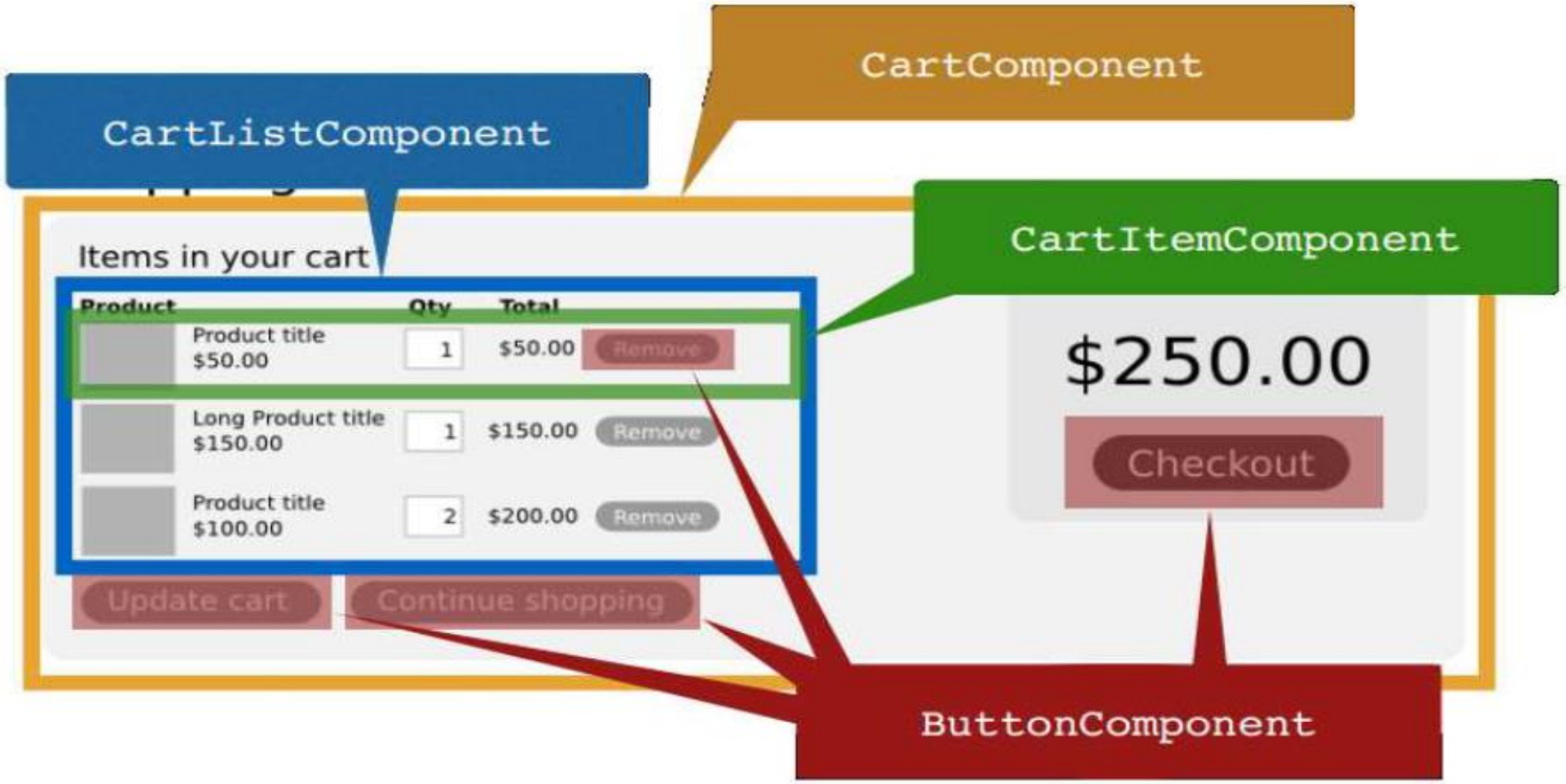


React Components – UI Tree

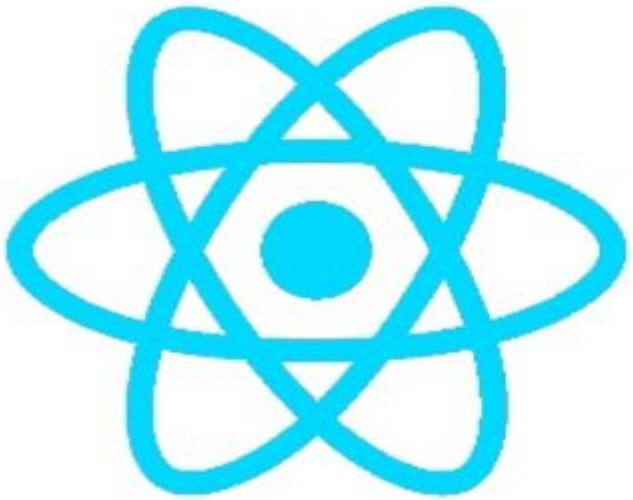
Single view of UI is divided into logical pieces. The starting component becomes the root and rest components become branches and sub-branches.

Component Structure

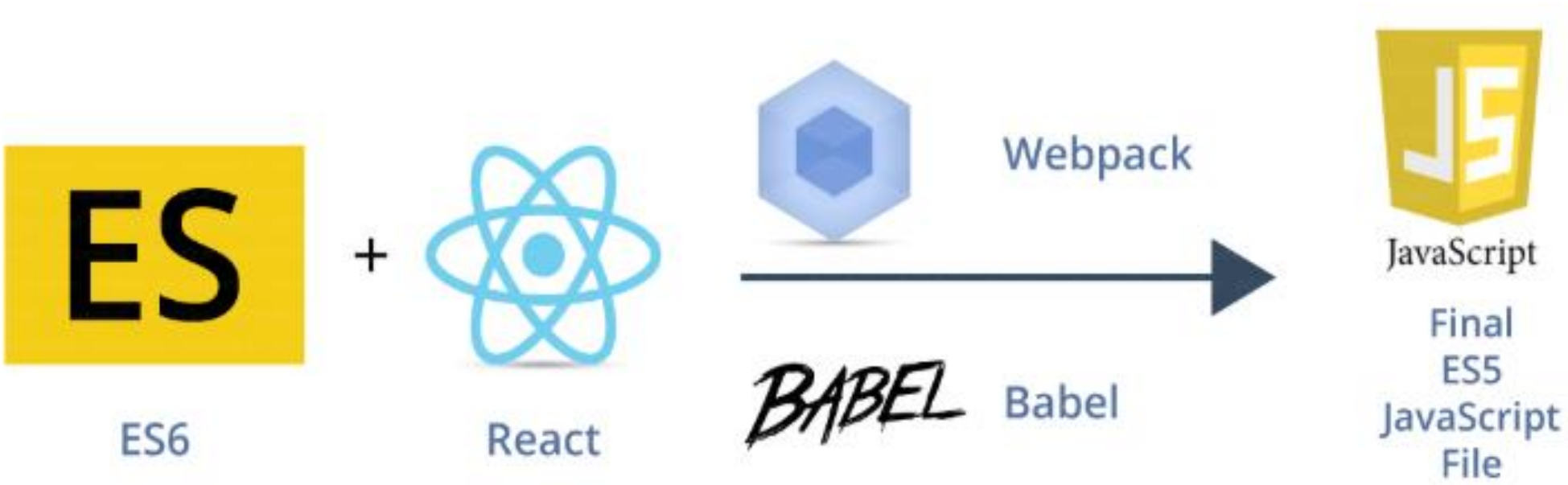




Components



1. Everything in ReactJS is component
2. Each Component Return a DOM Object
3. It splits the UI into independent reusable pieces
4. Each independent piece is processed separately
5. It can refer other components in output
6. It can be further split into smaller components



- ES6 and JSX transpilation using Babel
- Dev server with hot module reloading
- Code linting
- CSS auto-prefixing
- Build script with JS, CSS and image bundling, and sourcemaps
- Jest testing framework

- **Install NodeJS (https://nodejs.org/)**

- **install create-react-app globally with node package manager (npm)**

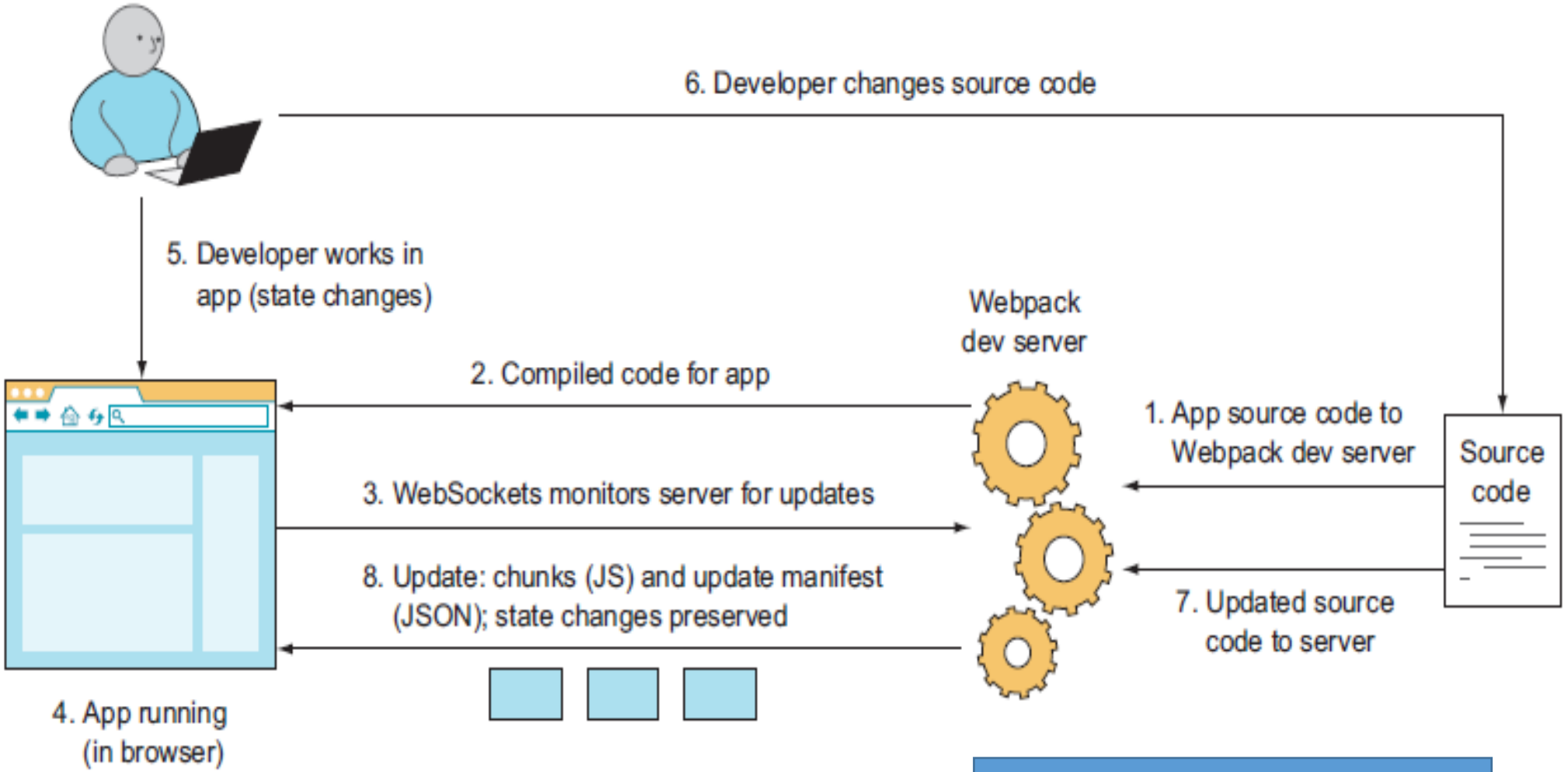
```
npm install -g create-react-app
```

- **Run the generator in your chosen directory**

```
create-react-app my-app
```

- **Navigate to the newly created directory and run the start script**

```
cd my-app/
npm start
```



HelloWorld.js

- ✓ JSX stands for JavaScript XML
- ✓ Makes HTML easy to understand
- ✓ It is Robust
- ✓ Boosts up the JS performance

Variable declaration

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

Expressions

```
const name = 'Karthik Raman';
const element = <h1> Hello, {name} </h1>;
```

```
function formatName(user) {
  return user.firstName + ' ' + user.lastName;
}
const user = {
  firstName: ‘Karthik',
  lastName: ‘Raman'
};
const element = (
  <h1>
    Hello, {formatName(user)}!
  </h1>
);
```

Conditional Rendering

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

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

```
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
   return <UserGreeting name="Karthik" />;
  }
  return <GuestGreeting />;
}
```

```
render(
  // Try changing to isLoggedIn={true}:
  <Greeting isLoggedIn={false} />
  );
```

Ternary Operator

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

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
```

```
render(

  {isLoggedIn ? ( <UserGreeting name="Karthik" /> )
              : ( <GuestGreeting /> )
 }
);
```

Only One Root Element

```
const element = (
  <div>
     <h1>Hello!</h1>
     <h2>Good to see you here.</h2>
  </div>
);
```

React.Fragment

```
const element = (
   <React.Fragment>
      <h1>Hello!</h1>
      <h2>Good to see you here.</h2>
   </React.Fragment>
);
```

<> and </>

```
const element = (
   <>
      <h1>Hello!</h1>
      <h2>Good to see you here.</h2>
   </>
);
```

If an attribute/prop is duplicated the last one defined wins

Omitting the value of an attribute/prop causes JSX to treat it as true

```
<input checked id type="checkbox" />
```

The class attribute has to be written className

The style attribute takes a reference to an object of camel-cased style properties

```
marginWidth  marginHeight  maxLength
readOnly   rowSpan   tabIndex
```

Spread attributes

```
<div>
< BlogPost id = {data.id} title={data.title}
date={data.date} desc={data.desc}  />
</div>
```

```
<div>
<BlogPost {...data} />
</div>
```

Looping

```
const elements = ['one', 'two', 'three'];
return (
  <ul>
     { elements.map ((value, index) =>  {
                return <li key={index}>{value}</li>
     })}
  </ul>
)
```

React Components

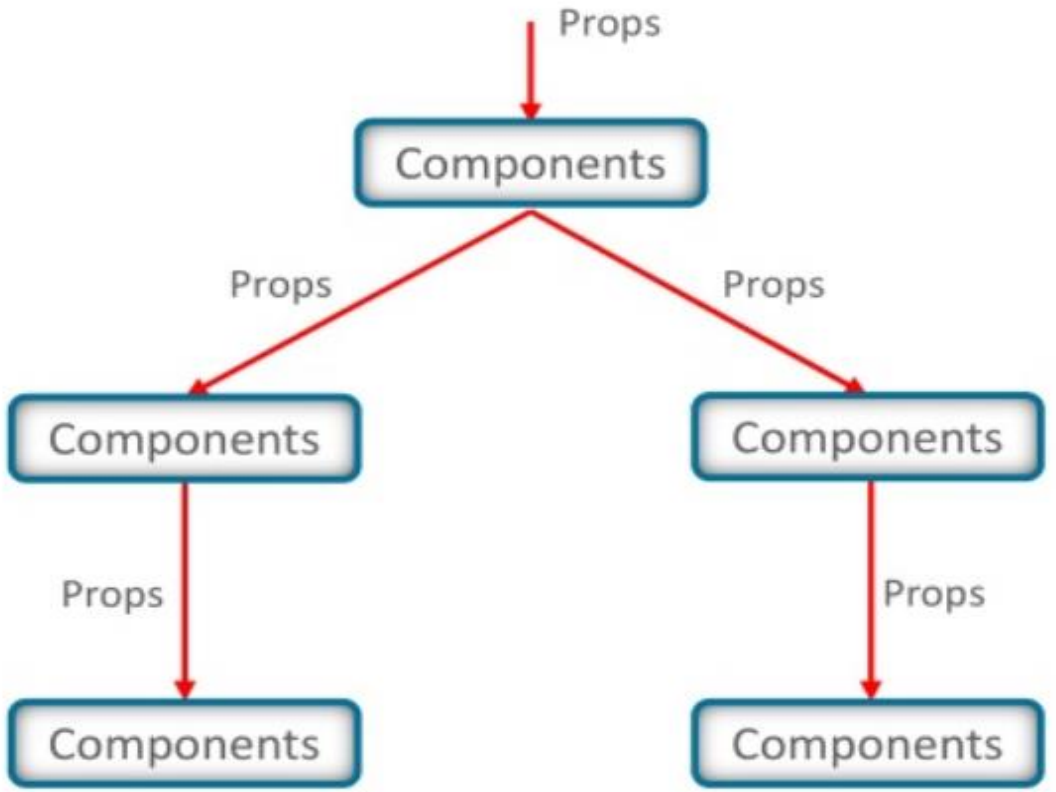
React components are controlled either by Props or States

Props

State

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



Data can be passed as props down to Child Components

ParentChild.js

props.children

is used to pass whatever included between the opening and closing tags when invoking a component.

PropChildExample.js

```
class PrimaryButton extends React.Component {
  render() {
    return (

      <button onClick={this.props.onClick}>
        {this.props.children}
      </button>
    )
  }
}
```

for validating any data we are receiving from props

npm install --save prop-types

- `PropTypes.string`
- `PropTypes.string`
- `PropTypes.number`
- `PropTypes.bool`
- `PropTypes.object`
- `PropTypes.array`
- `PropTypes.func`
- `PropTypes.shape`
- `PropTypes.any.isRequired`
- `PropTypes.objectOf(PropTypes.number)`
- `PropTypes.arrayOf(PropTypes.number)`
- `PropTypes.node`
- `PropTypes.instanceOf(Message)`
- `PropTypes.element`
- `PropTypes.oneOfType([PropTypes.number, ...])`

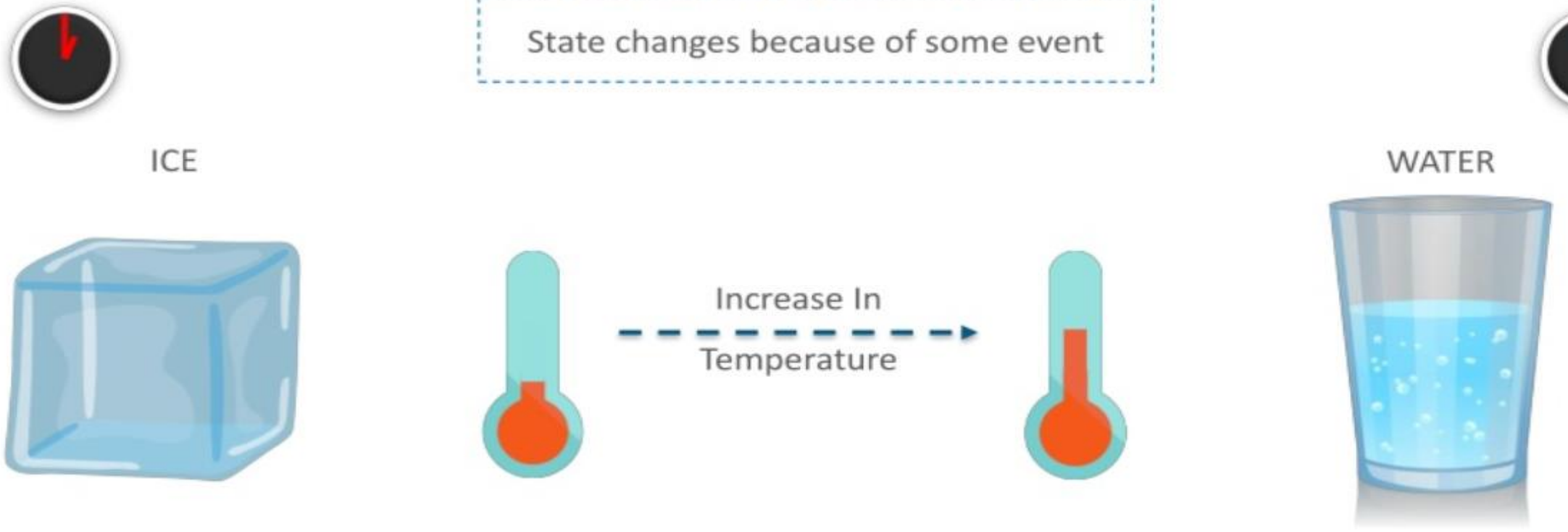
PropTypesExample.js

Check the Console for Warnings

State

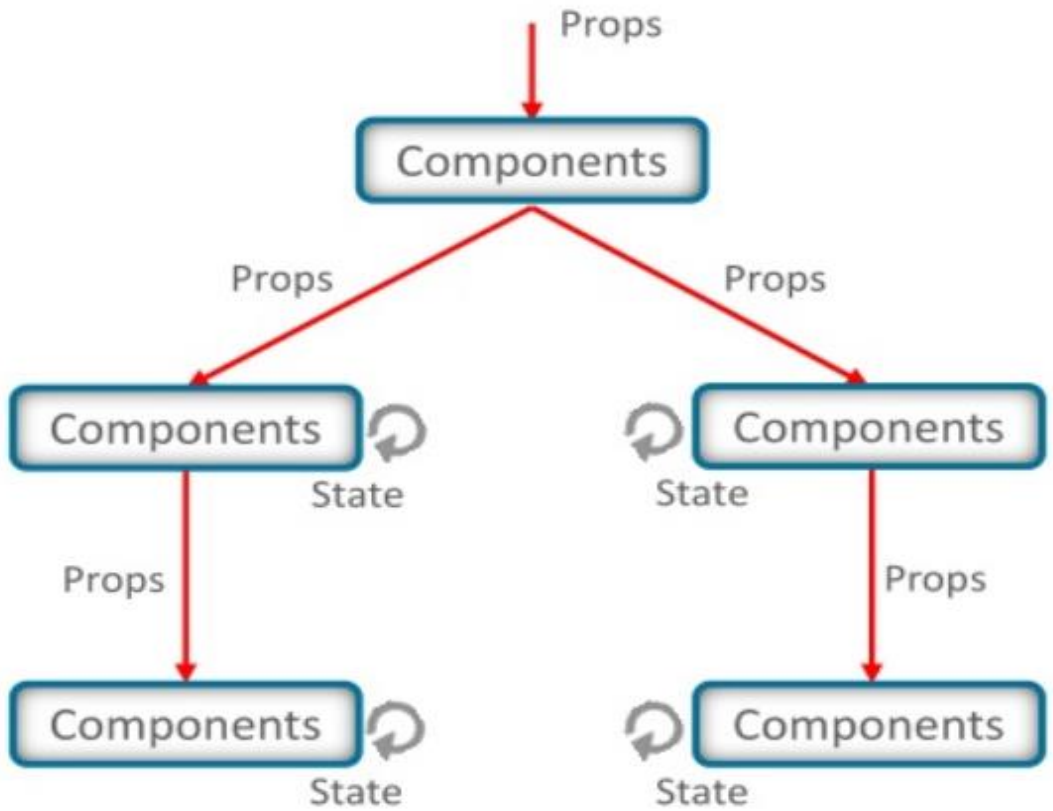
Components can change, so to keep track of updates over the time we use state

State changes because of some event

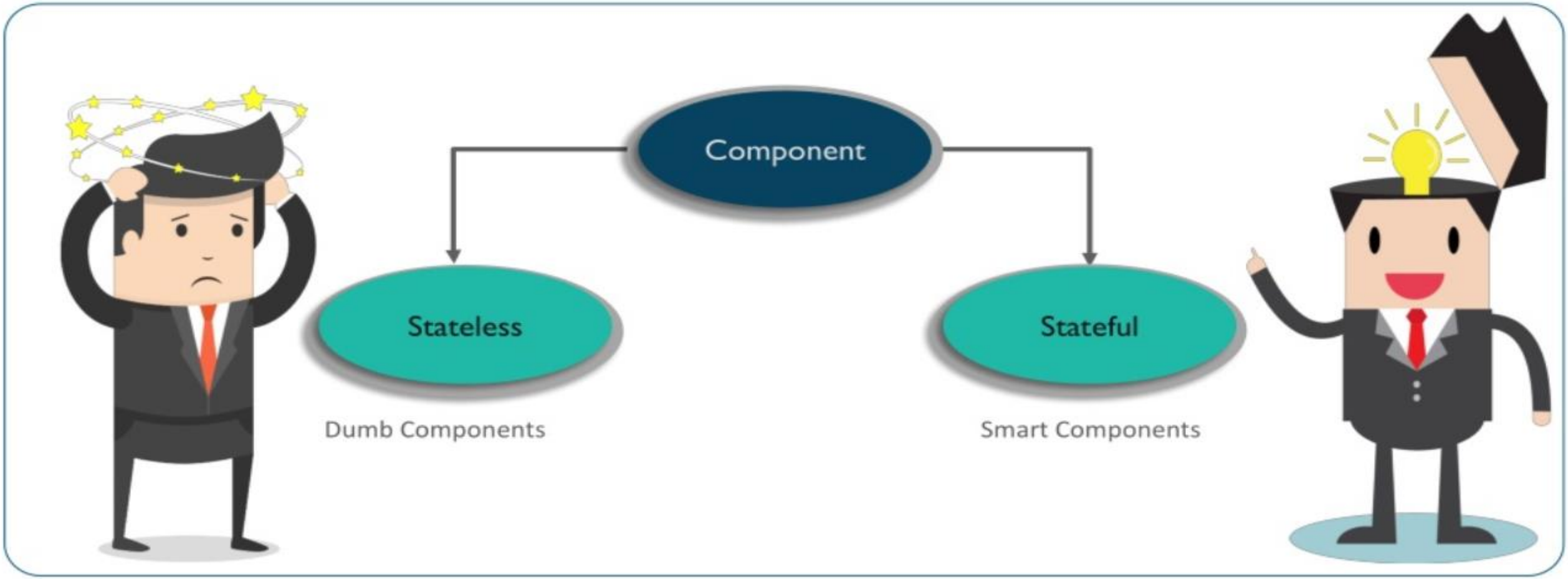


State

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



React Components



Initializing the State

```
constructor(props) {
      super(props);
      this.state = { count: 0 };
   }
```

Accessing the State

```
return (
        <div>
                Count: { this.state.count }
        </div>
      );
```

Changing the State

```
this.state. count = this.state. count++
```

Use this.setState

```
this.setState({ count: this.state.count + 1 });
```

setState is Asynchronous...

```
this.setState((prevState) => ({
          count: prevState.count + 1
        }));
```

Pass a function as a prop from the parent component to the child component, and the child component calls that function

ChildParent.js

Do Not Modify State Directly

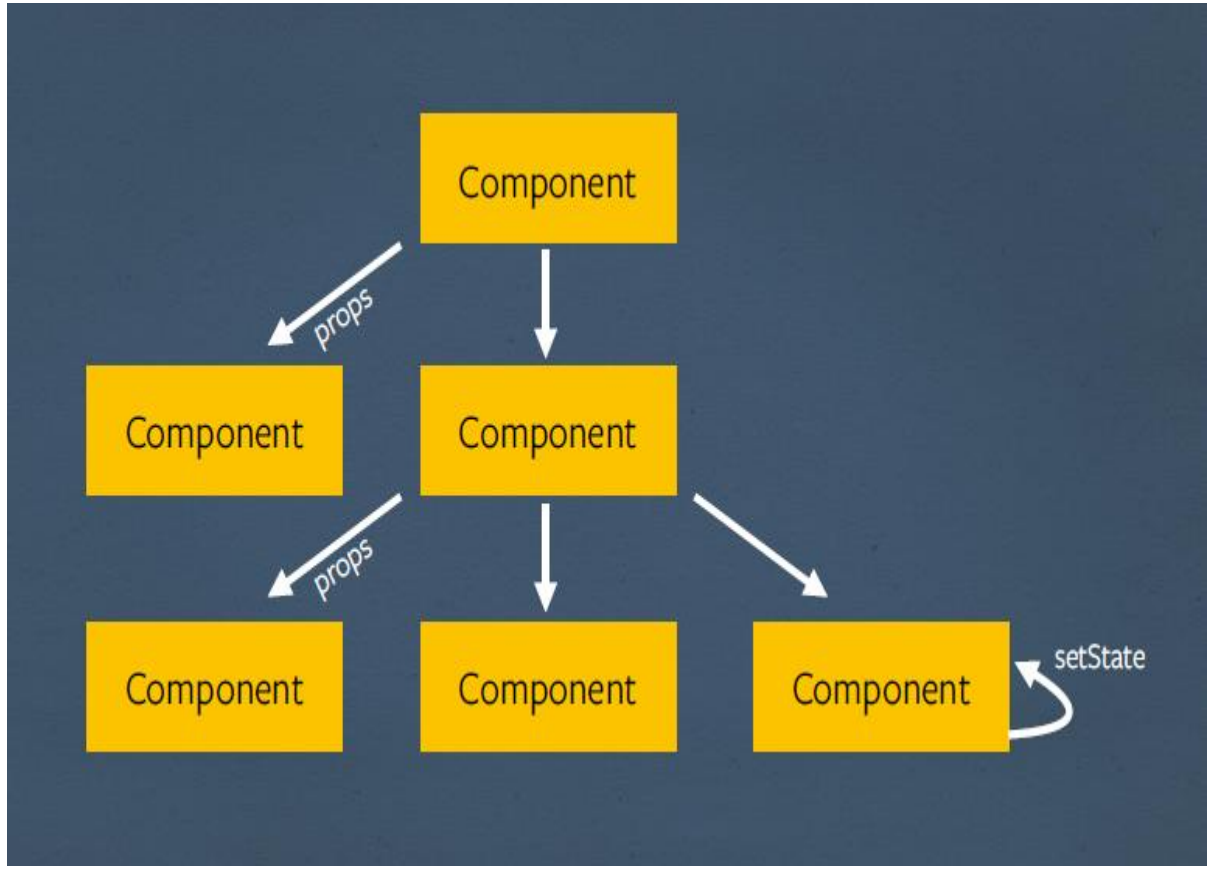
```
// Wrong
this.state.count = 10;          //Other than setting Initial State in Constructor

// Correct  ....  use setState():
this.setState( {count: 10} );
```

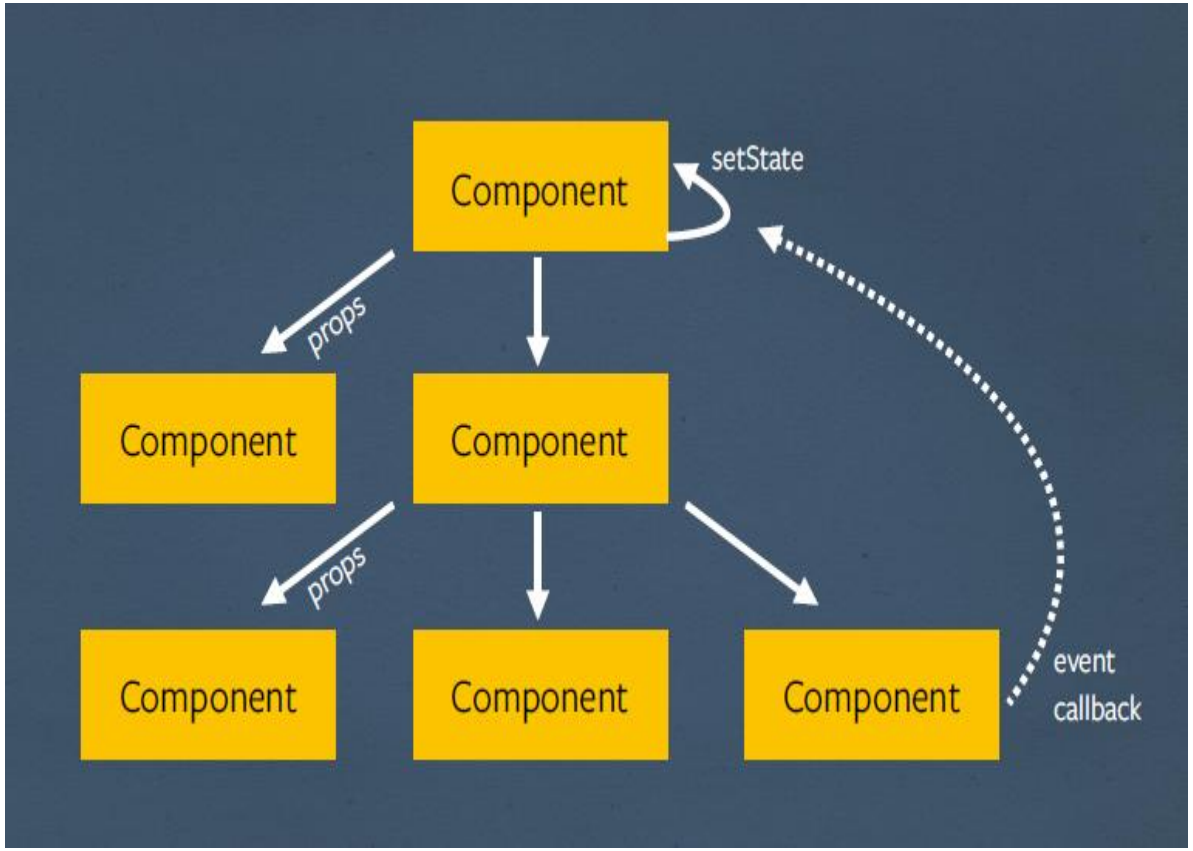
setState is Asynchronous

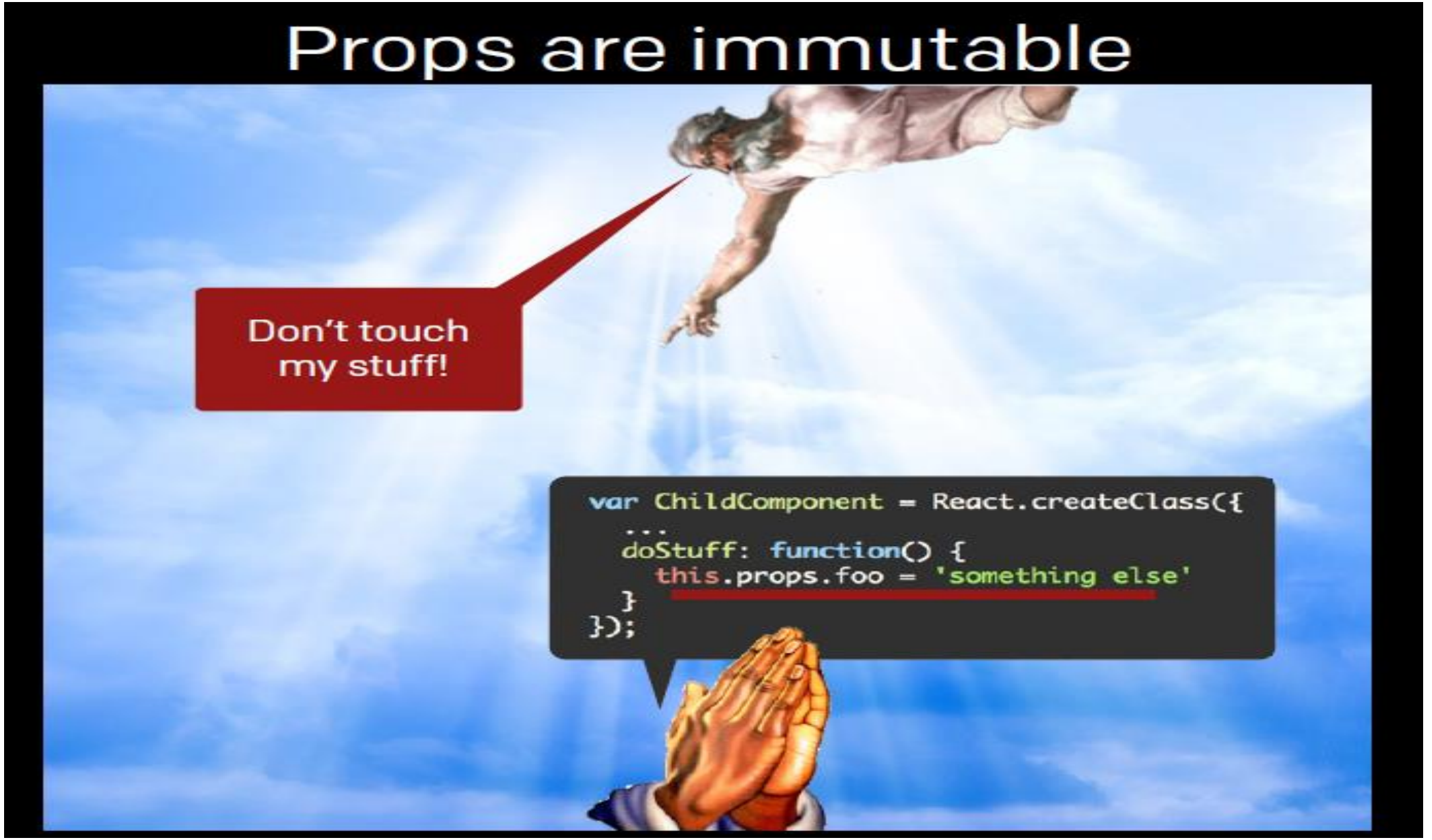
```
//Wrong
this.setState({ count: this.state.count++ });
//Correct
this.setState((prevState) => ({
            count: prevState.count + 1
        }));
```

In Class based Components, React merges the objects in the current state with the setState object

Props and State







Download the React DevTools for a better development experience
https://fb.me/react-devtools and Add as Extensions to the Chrome Web Store

To run React DevTools in a standalone mode

```
npm install -g react-devtools
```

Run react-devtools from the terminal to launch the standalone DevTools App

```
react-devtools
```

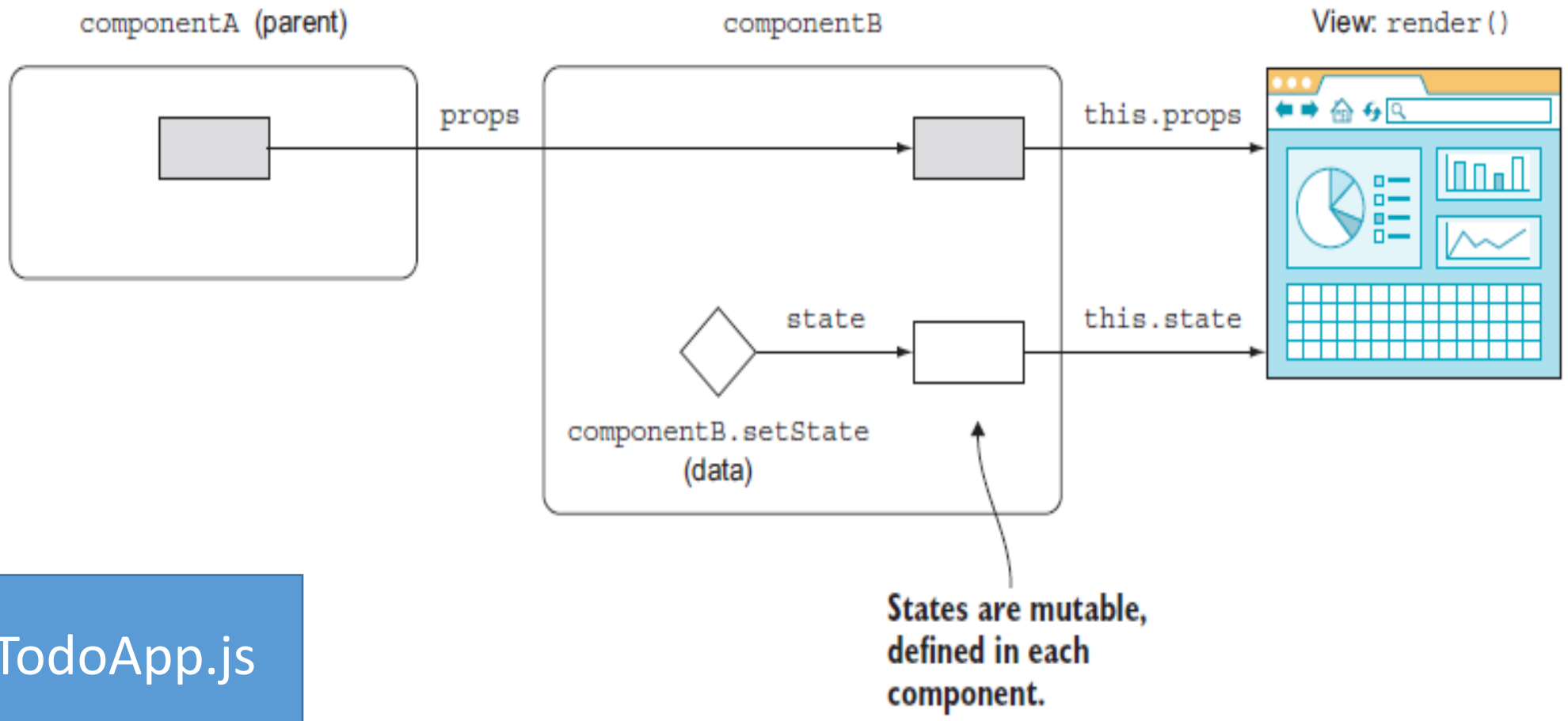
```
Add <script src="http://localhost:8097"></script> as the very first <script>
tag in the <head> in index.html
```

Don’t forget to remove it before deploying to production!

Lifting the shared state up to their closest common ancestor

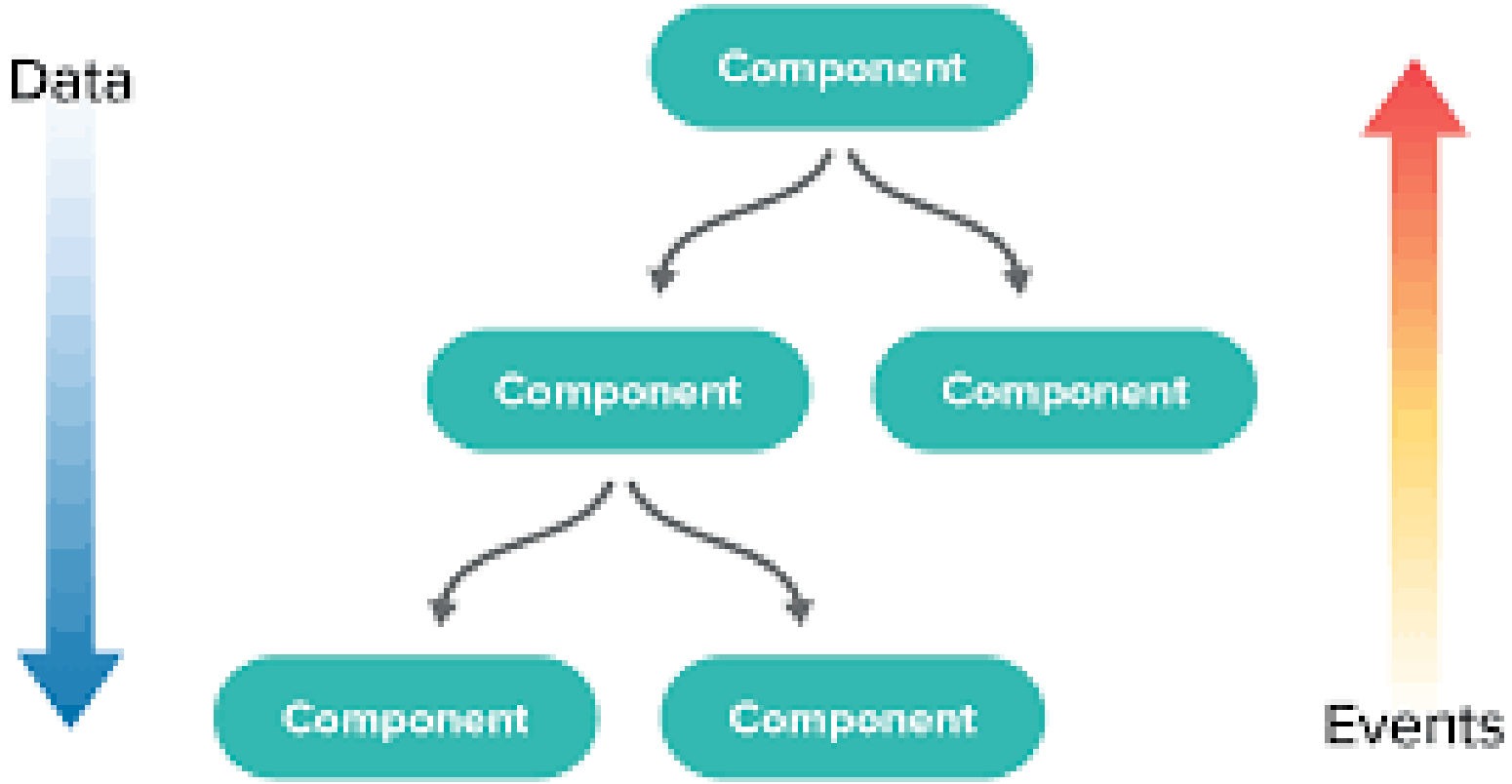
TempCalc.js

Props and State



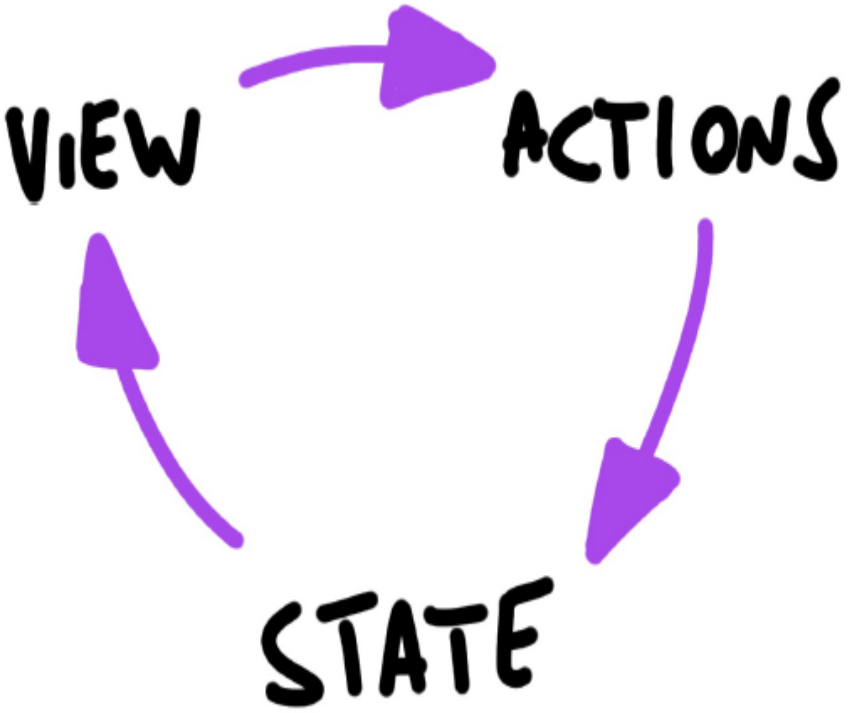
TodoApp.js

Flow of Data and Events



- state is passed to the view and to child components as props
- actions are triggered by the view
- actions can update the state
- the state change is passed to the view and to child components as props

- view is a result of the application state
- State can only change when actions happen
- When actions happen, the state is updated



Refs provide a way to access React elements created in the render method

Usecases such as :
Managing focus, text selection, or media playback
Triggering imperative animations

RefsExample.js

- **Including css files in index.html for Global Styling using className**
  GlobalStyleExample.js **Included Bootstrap.css in index.html**

- **Inline Styling (add css inline, as attributes and passed to elements)**
  InlineStyleExample1.js InlineStyleExample2.js

- **External css file** ExternalCSSExample.js

- **CSS Modules (CSS inside a module is available only for the component that imported it, and NO Name conflicts)**
  - **File Naming Convention name.module.css (name is your file name)**
  - **Classes are dynamically generated, unique, and mapped to the styles**
  - **Inspect the Element in the Browser to verify** CssModuleExample.js

- **Styled Components**
- **Stylable**
- **Radium**
- **Aphrodite**
- **Emotion**

Managing Form data using DOM itself (Traditional Way) using Refs

Called as “Uncontrolled Components”

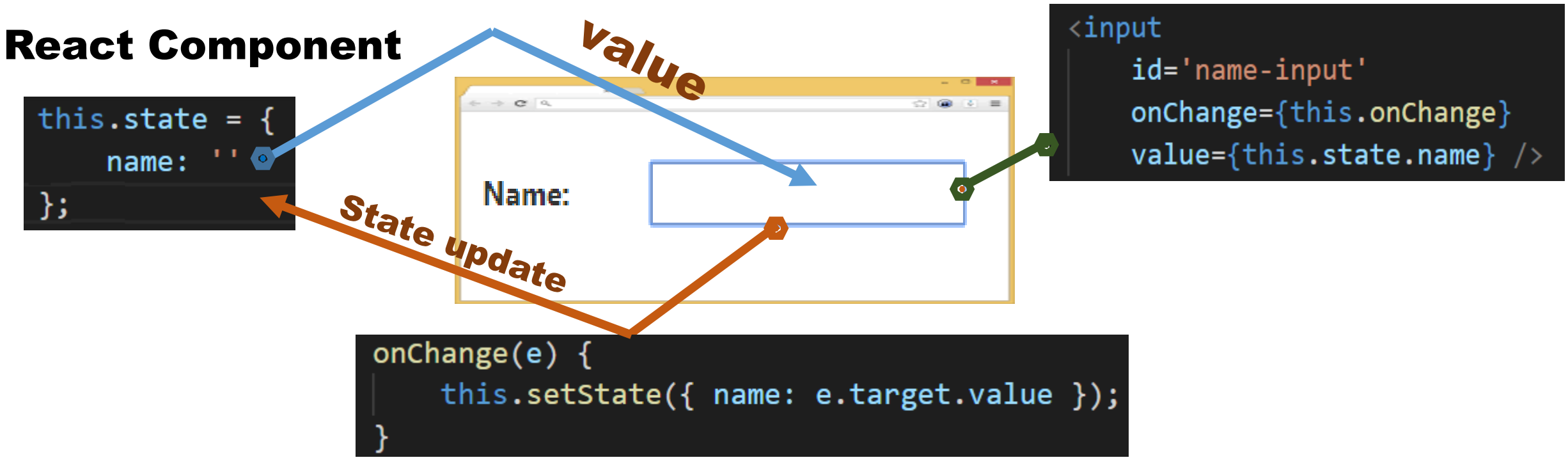
Uncontrolled input are like traditional HTML form inputs
which remember what was typed in the DOM element
use Ref to get the form values.

RefsExample.js

Clayfin

Managing Form Data using State
Called as “Controlled Components”

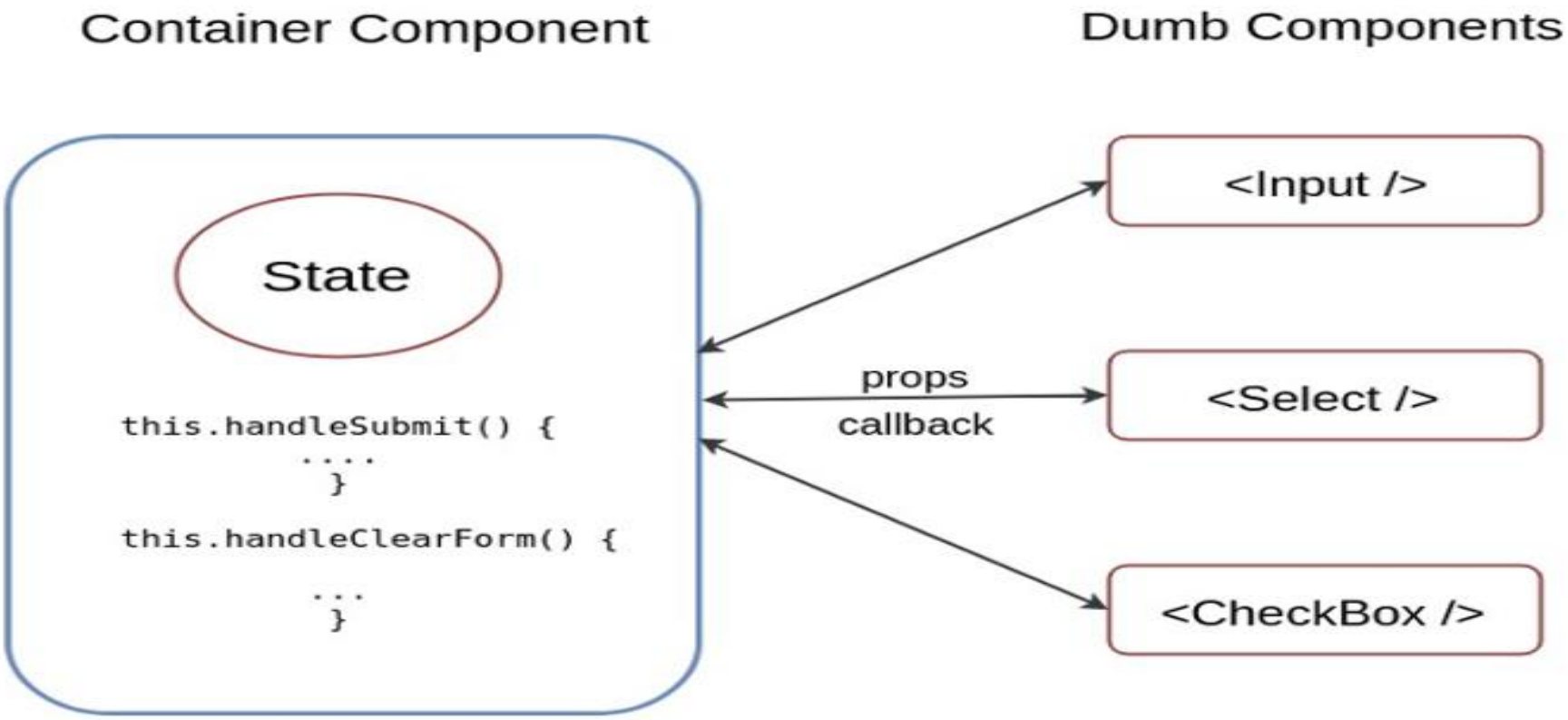
SingleElementForm.js



TwoElementForm.js

MultiSelect.js

Form Data Using State



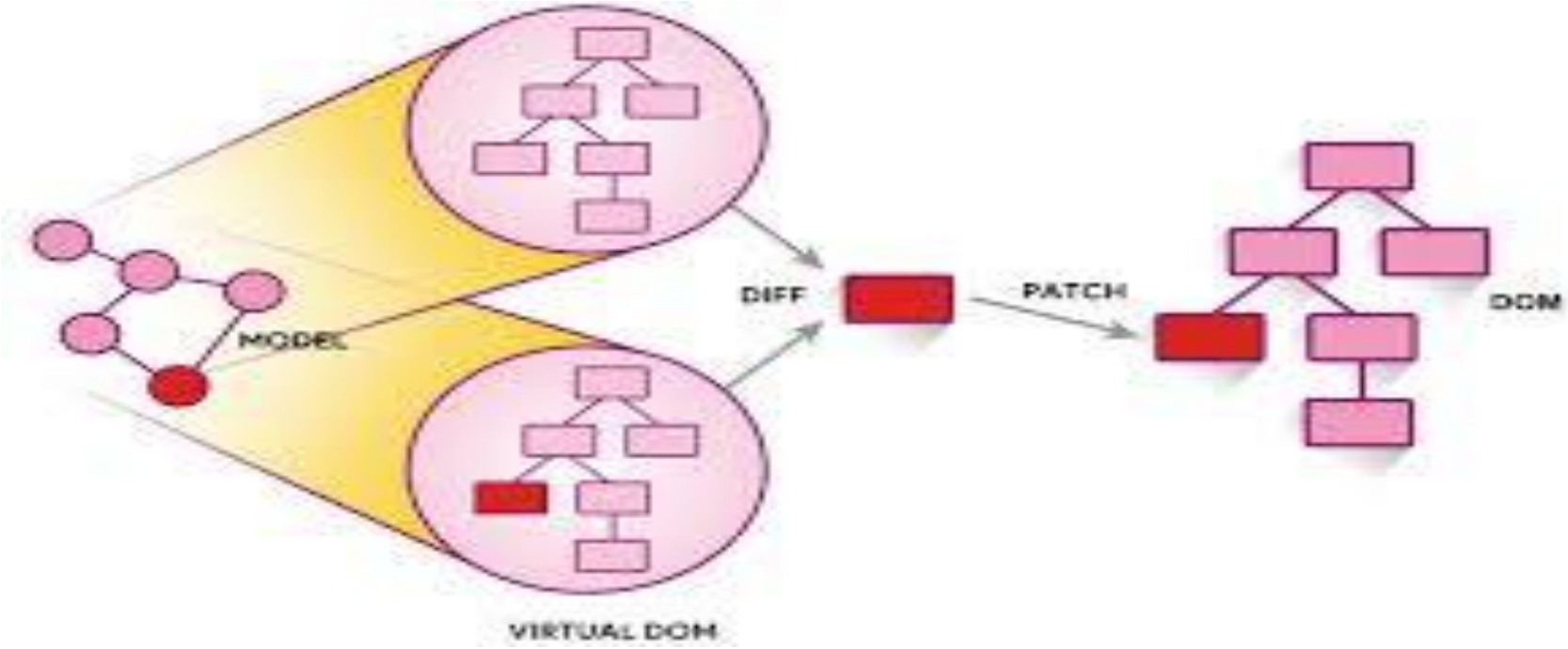
FormContainer1.js

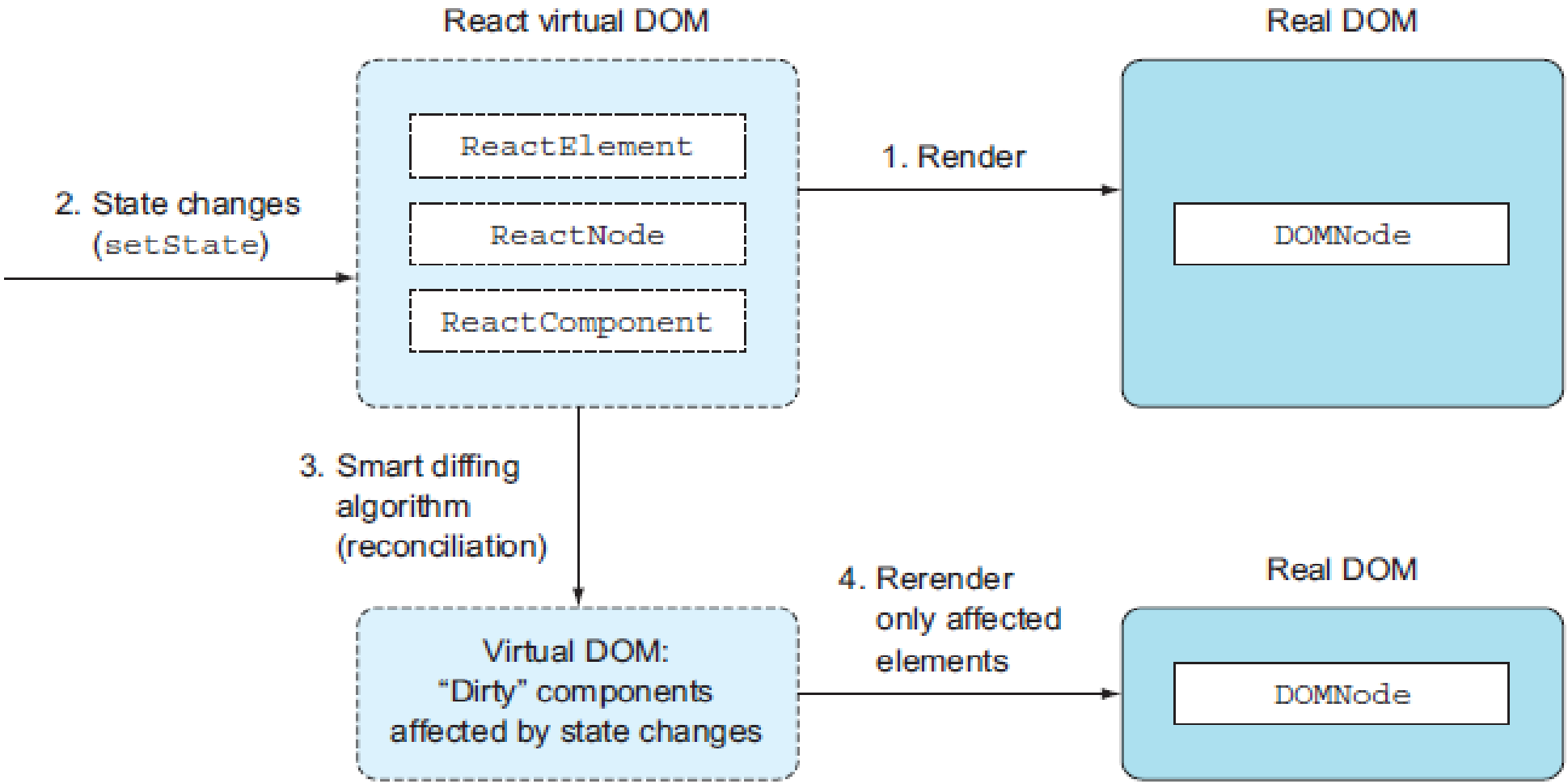
React creates an in-memory data structure cache, computes the resulting differences, and then updates the browser's displayed DOM efficiently

1. Create lightweight description of component UI

2. Diff it with the old one

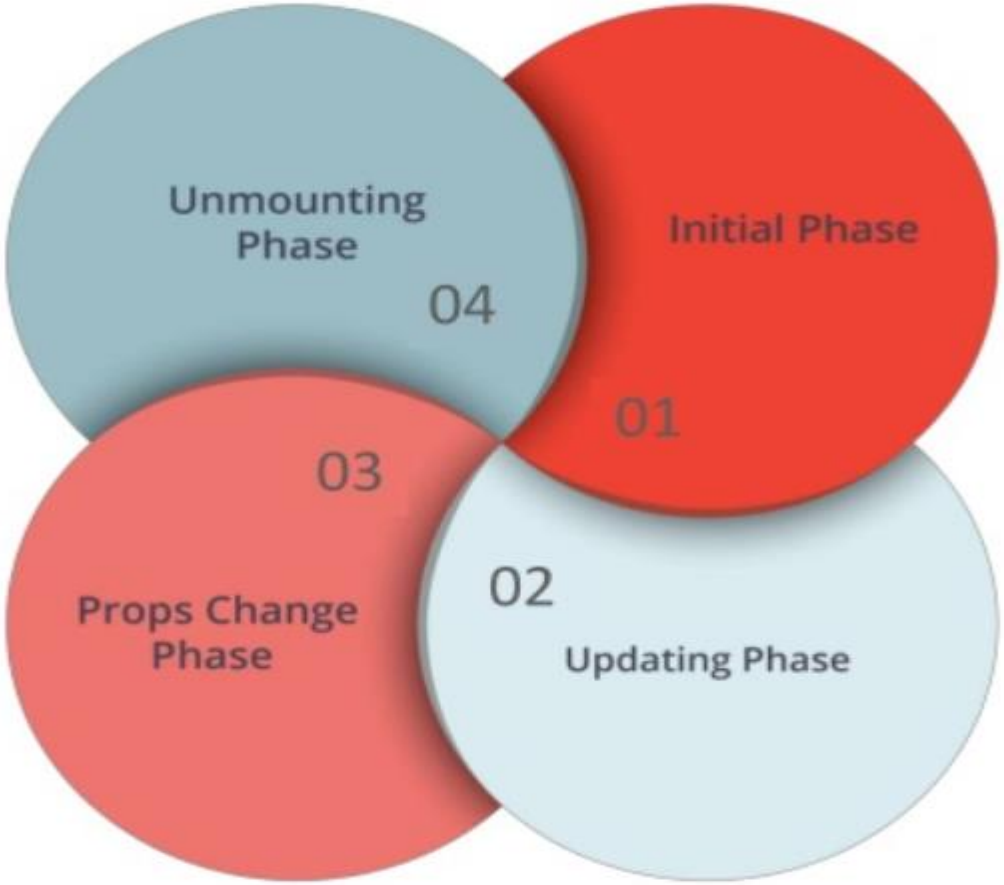
3. Compute minimal set of changes to apply to the DOM





- “Side Effect” is anything that affects something outside the scope of the function being executed.
- Making Network request, accessing Native DOM, logs to be recorded, caches to be updated, setting up a subscription are examples of tasks which can cause Side Effects.
- Changing the value of a closure-scoped variable, Pushing a new item onto an array that was passed in as an argument are also side effects.
- Functions that execute without side effects are called “Pure" functions they take in arguments, and they return values.
- Pure functions are deterministic (given an input, they always return the same output), but that doesn't mean that all impure functions have side effects. Generating a random value within a function makes it impure, but isn't a side effect

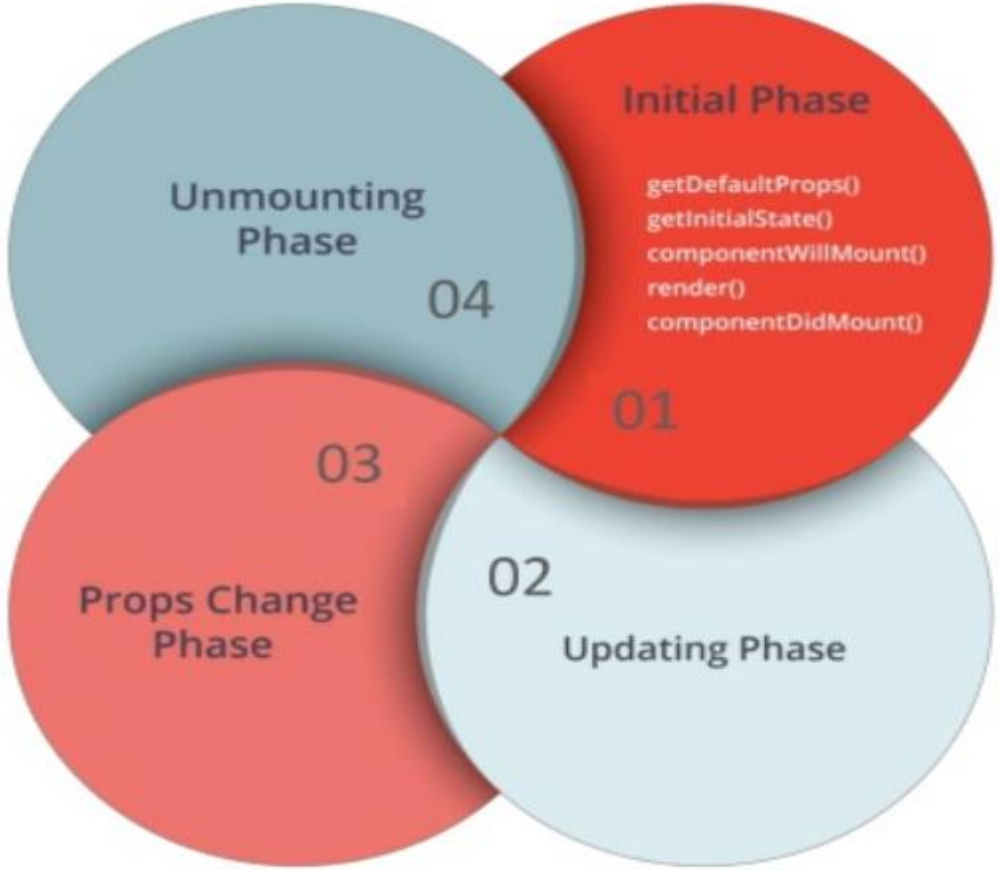
Lifecycle Phases



- 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

Initial Phase

Lifecycle Phases



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

1

- Occurs when the component is created.

```
var Greeting = React.createClass({
  propTypes: {
    name: React.PropTypes.string
  },

  getDefaultProps: function () {
    return {
      name: 'Mary'
    };
  },

  getInitialState: function () {
    return {
      helloSentence: 'Hello'
    }
  }

  // ...
});
```

THE LIFECYCLE - INITIALIZATION

- ✔ Initial
- ✔ GetDefaultProps
- ✔ GetInitialState
- ✔ ComponentWillMount
- ✔ Render
- ✔ ComponentDidMount

- getDefaultProps and getInitialState not exists when define Component as Class ES6.

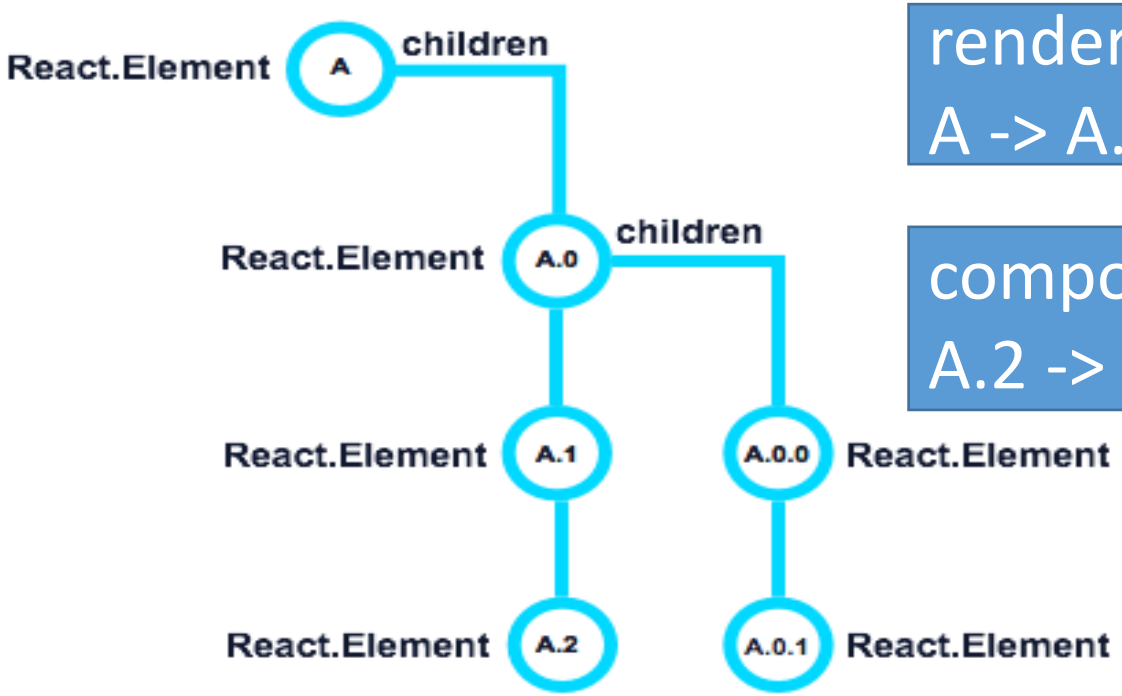
```
Greeting.defaultProps = {
  name: 'Mary'
};
```

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

THE LIFECYCLE - INITIALIZATION

Initial
✗ GetDefaultProps
✗ GetInitialState
✓ Constructor
✓ ComponentWillMount
✓ Render
✓ ComponentDidMount

ComponentDidMount Can Cause Side-Effects
AJAX Calls



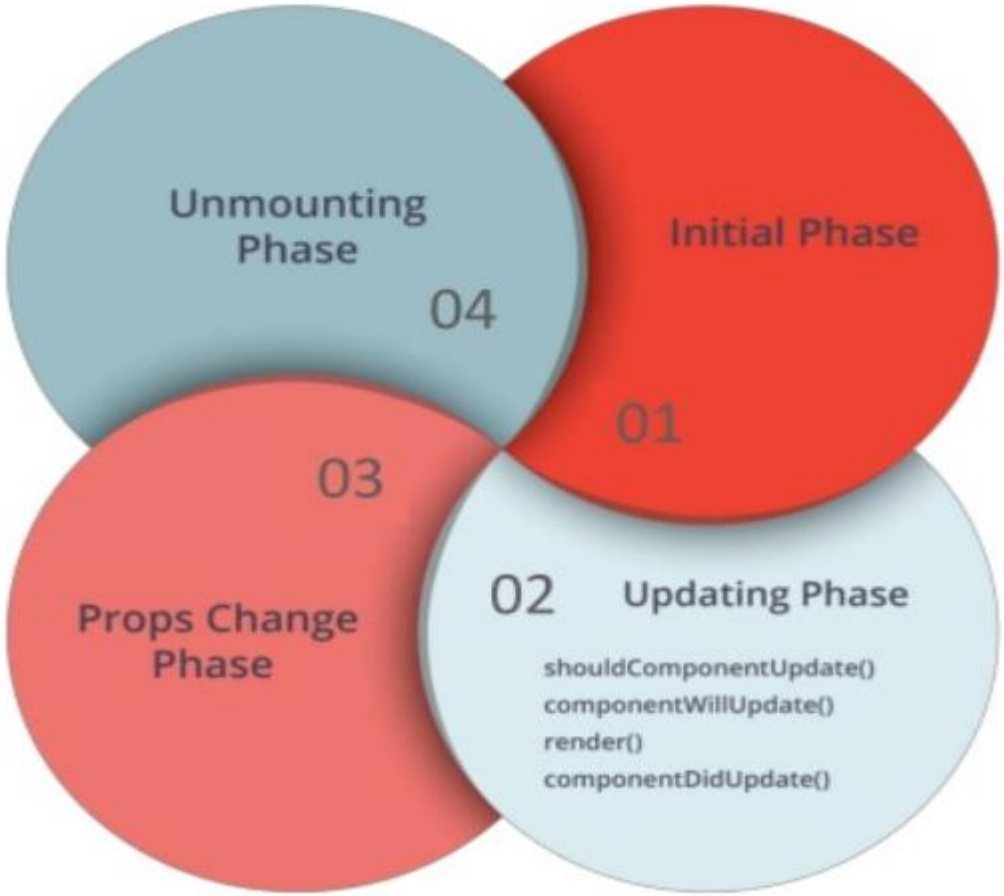
render() will be called in this order :
A -> A.0 -> A.0.0 -> A.0.1 -> A.1 -> A.2

componentDidMount() will be called in this order
A.2 -> A.1 -> A.0.1 -> A.0.0 -> A.0 -> A

render() should be a pure method
Should not mutate the state of the application
No Asynchronous functions

Updating State

Lifecycle Phases



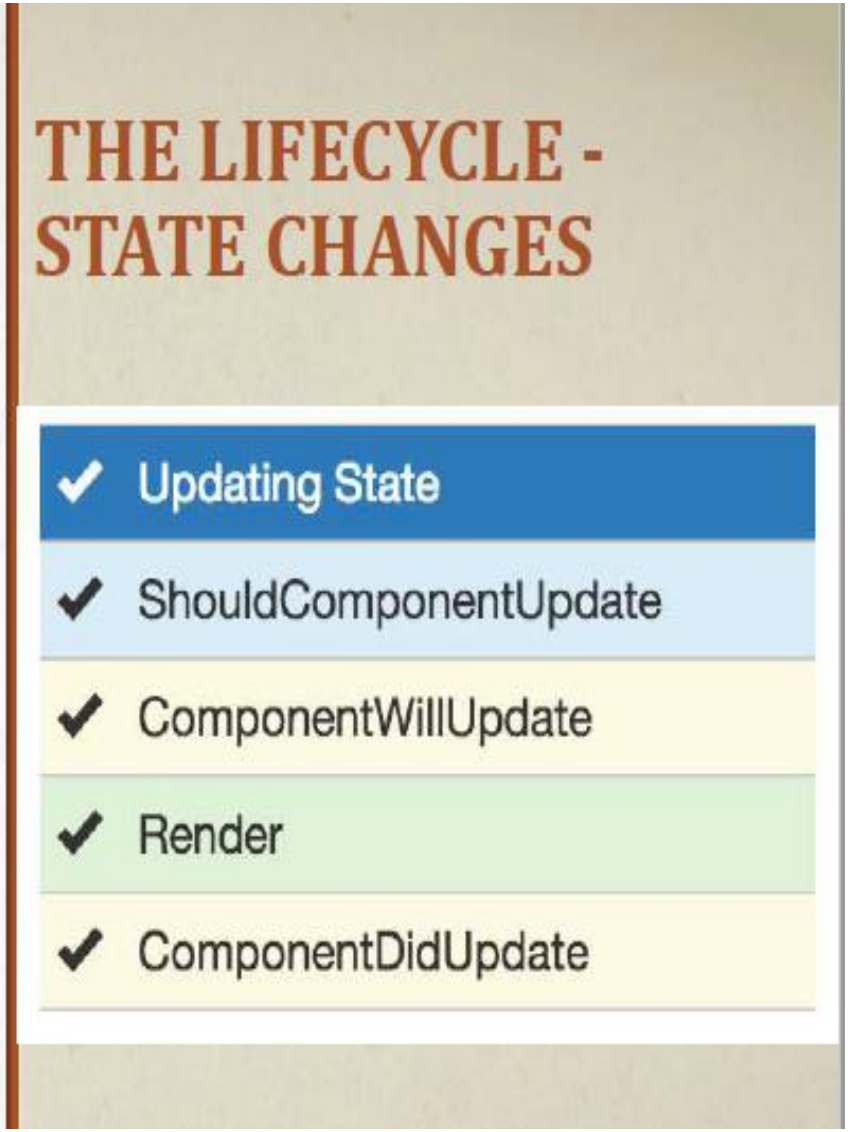
In this phase, component can update & re-render when a **state** change occurs

2

- Occur when state is changed (via this.setState(..)) except inside componentWillMount methods

```
shouldComponentUpdate: function(nextProps, nextState) {
  // return a boolean value
  return true;
}
```

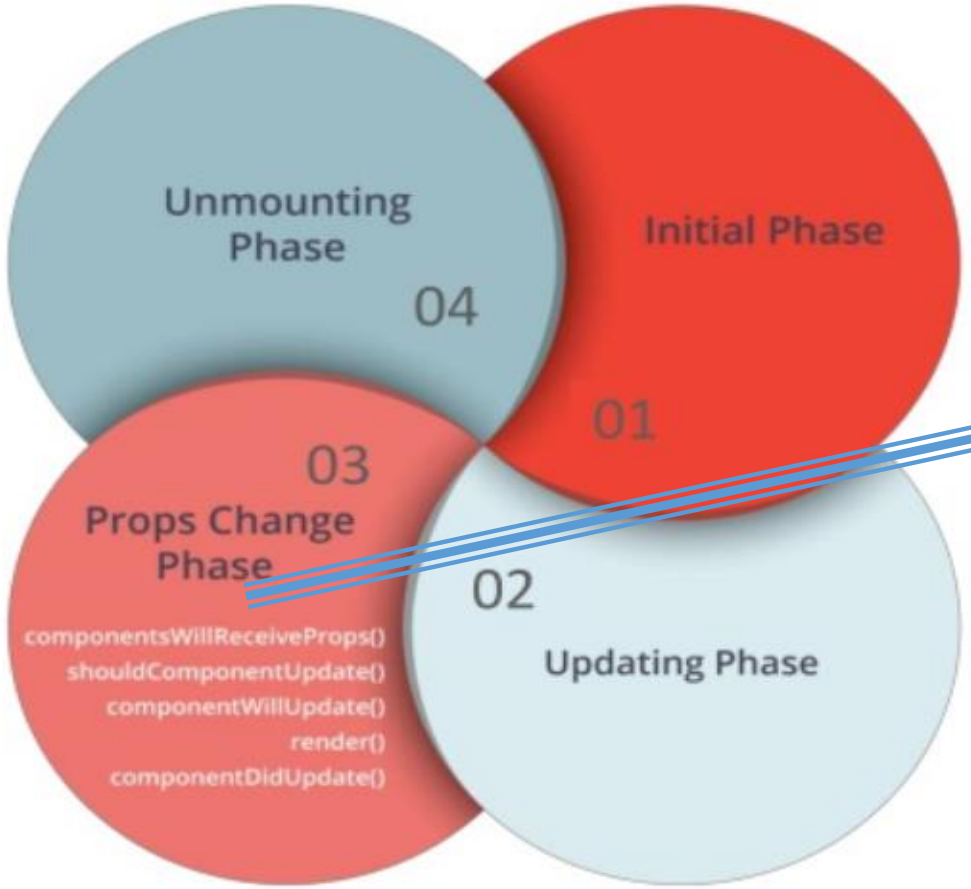
- shouldComponentUpdate returning false results in followed methods won't be triggerd also.
- shouldComponentUpdate won't triggered in the initial phase or when call forceUpdate().
- Current State of Component DID NOT have new value,



ComponentDidUpdate Can Cause Side-Effects with Caution
Condition to check for state or prop changes from previous value

Updating Props

Lifecycle Phases



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

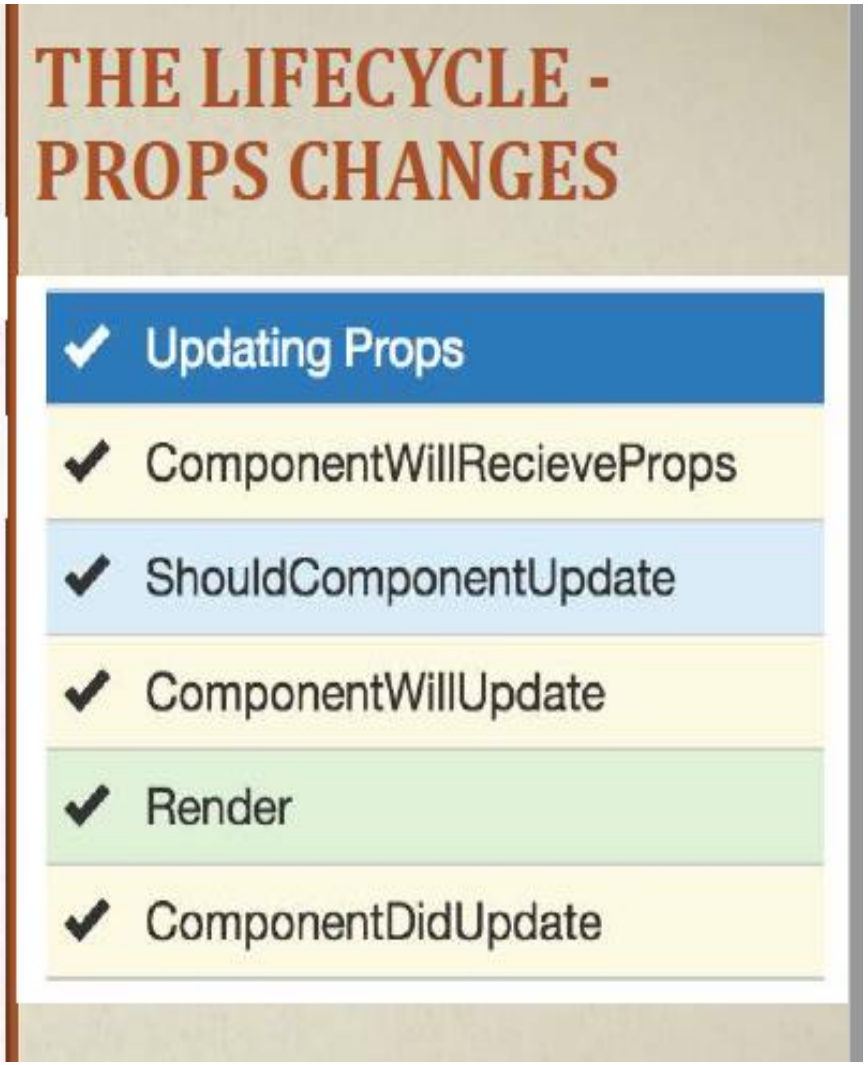
3

- Occurs when data passed from parent component to child component changed (via props).

Props Change → componentWillReceiveProps trigged

- Changing states in *ComponentWillReceiveProps* DID NOT trigger re-render component.

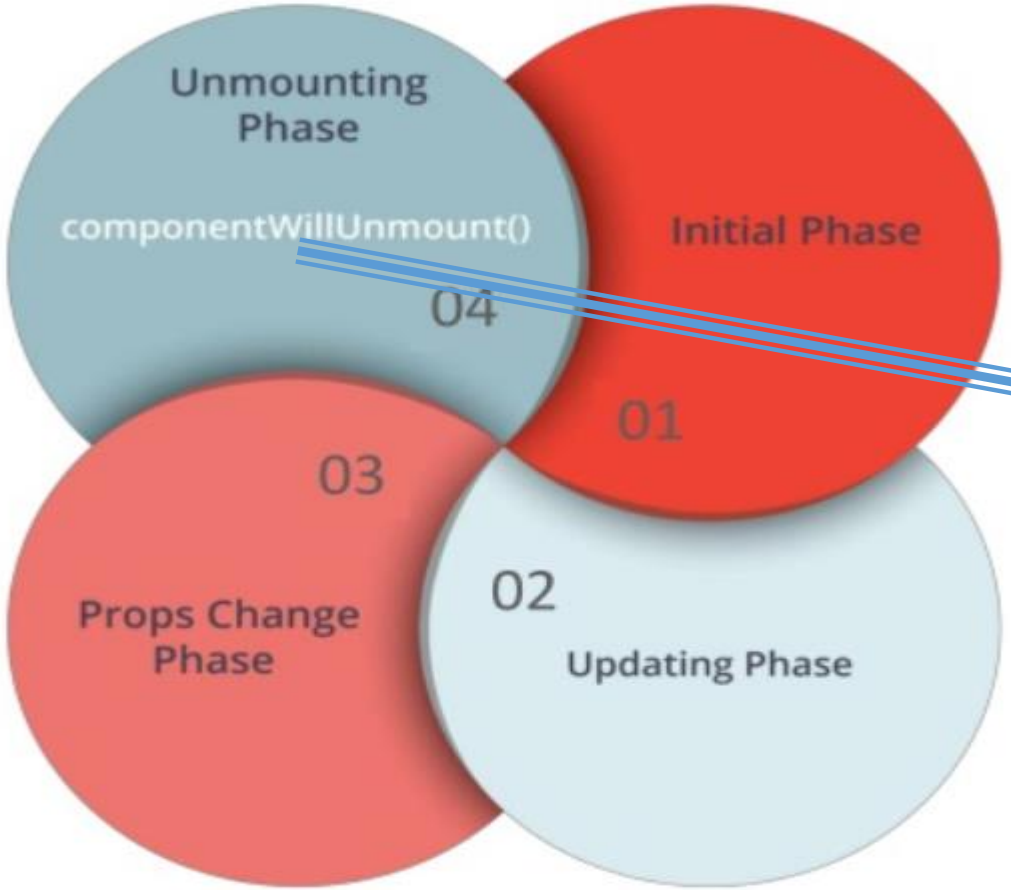
```
componentWillReceiveProps: function(nextProps)
{
  this.setState({
    // set something
  });
}
```



ComponentDidUpdate Can Cause Side-Effects with caution
Condition to check for state or prop changes from previous value

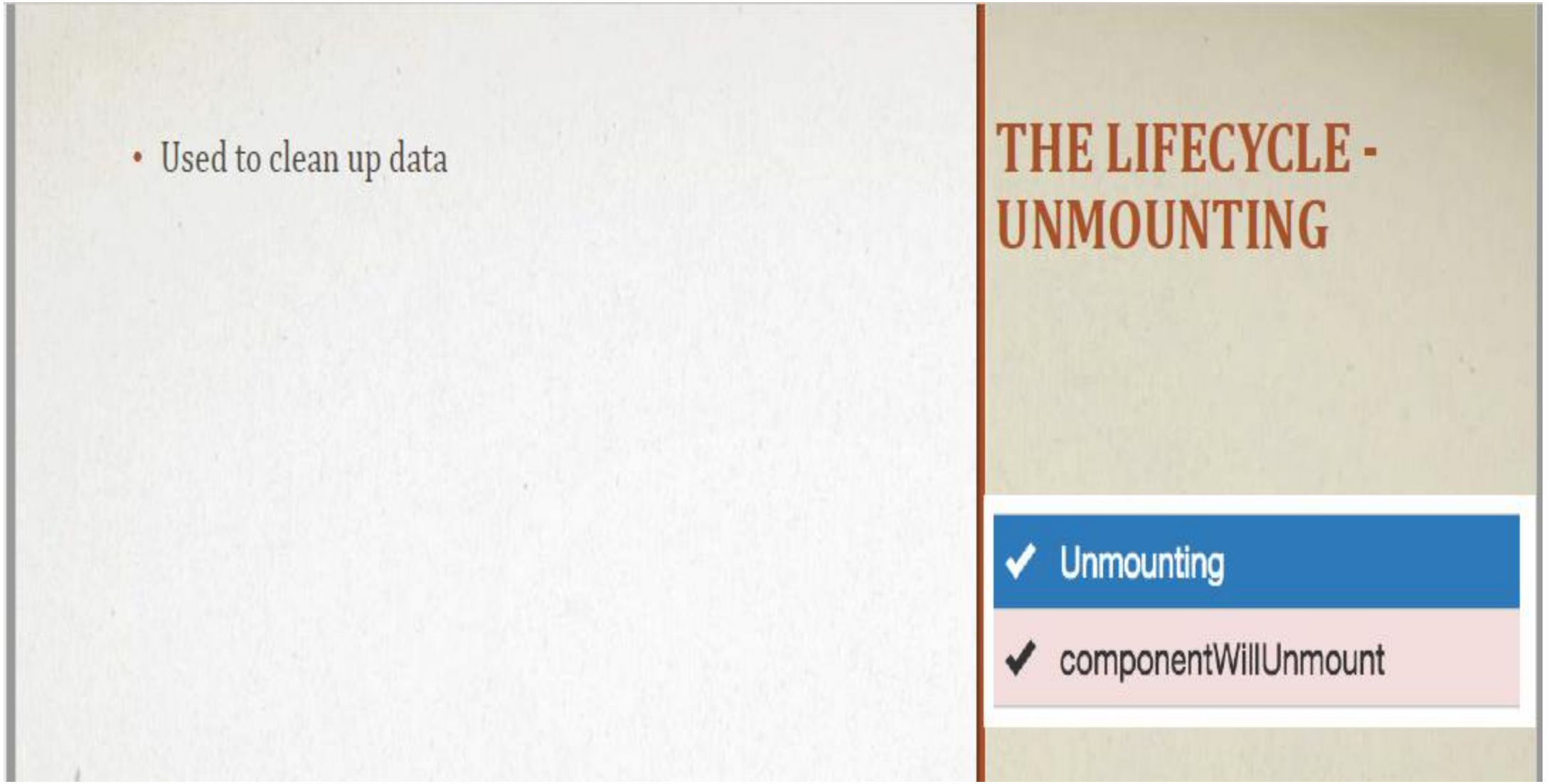
Unmounting Phase

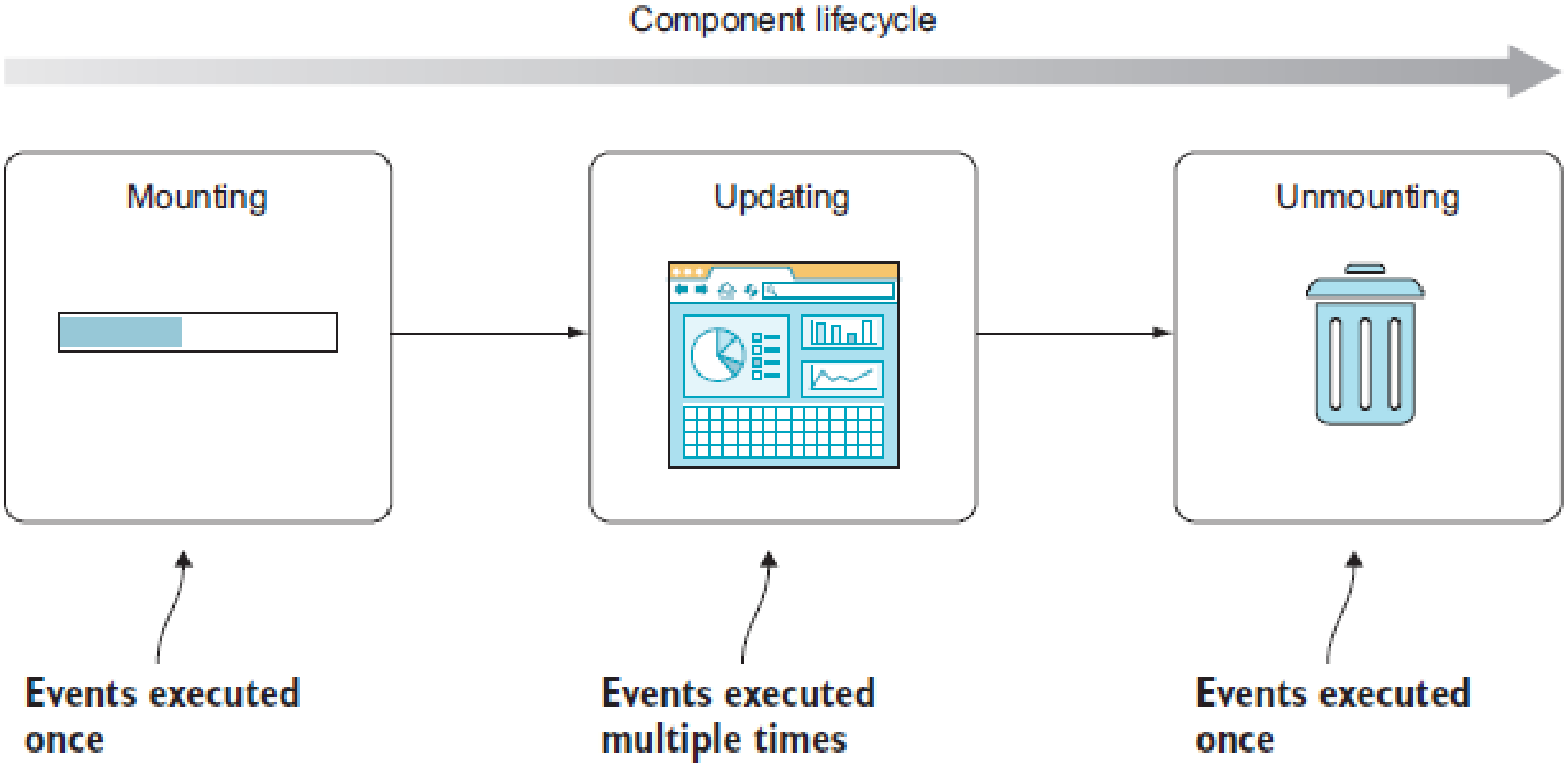
Lifecycle Phases



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

4



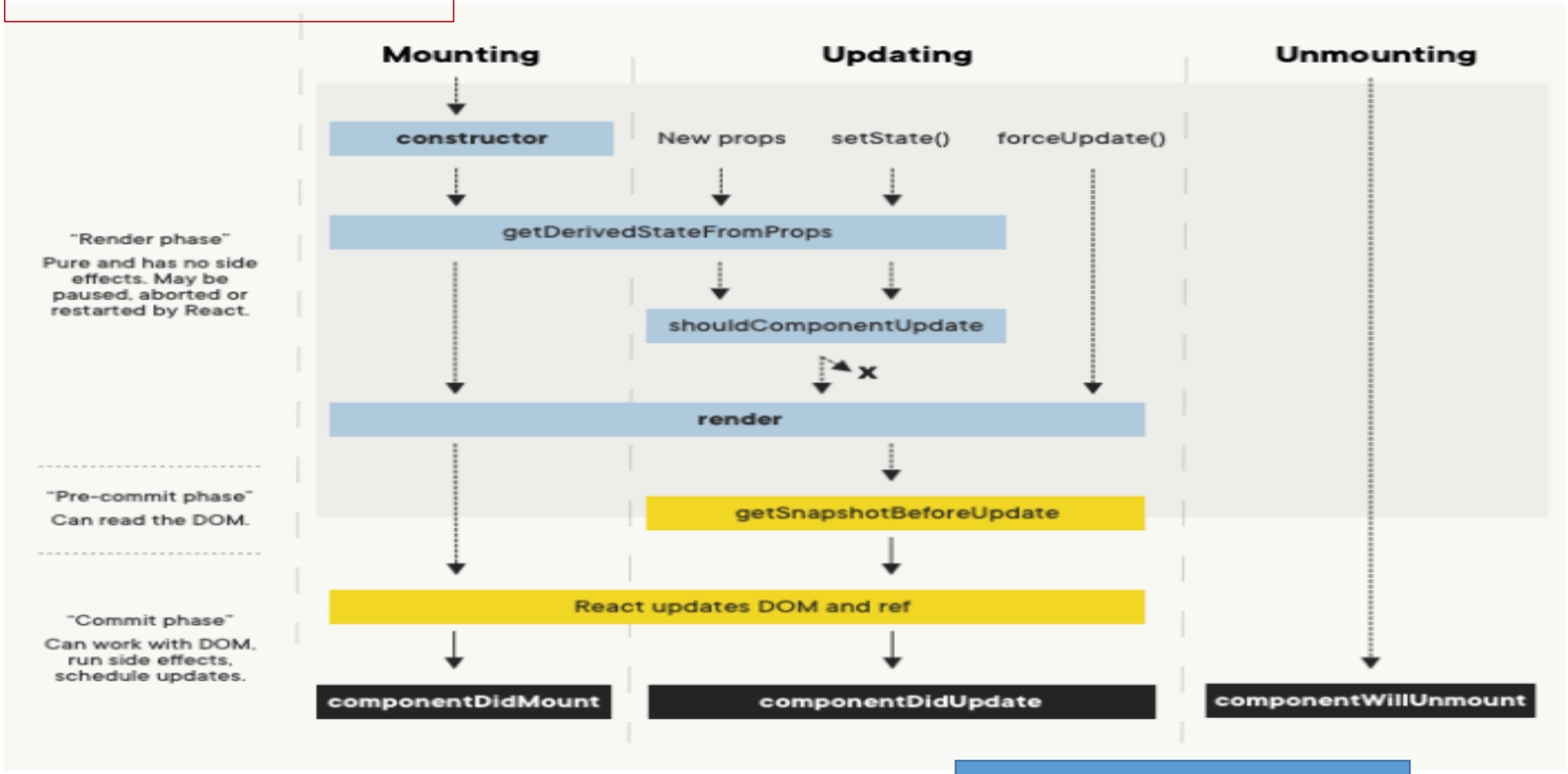


Mounting	Updating component state	Updating component properties	Unmounting
constructor()			
componentWillMount()			
		componentWillReceiveProps()	
	shouldComponentUpdate()	shouldComponentUpdate()	
	componentWillUpdate()	componentWillUpdate()	
render()	render()	render()	
	componentDidUpdate()	componentDidUpdate()	
componentDidMount()			
			componentWillUnmount()

LifeCycleExample1.js

Timer.js

Latest Methods



LifeCycleExample2.js

- **React.Component re-renders itself every time the props passed to it changes, parent component re-renders or if the shouldComponentUpdate() method is called**

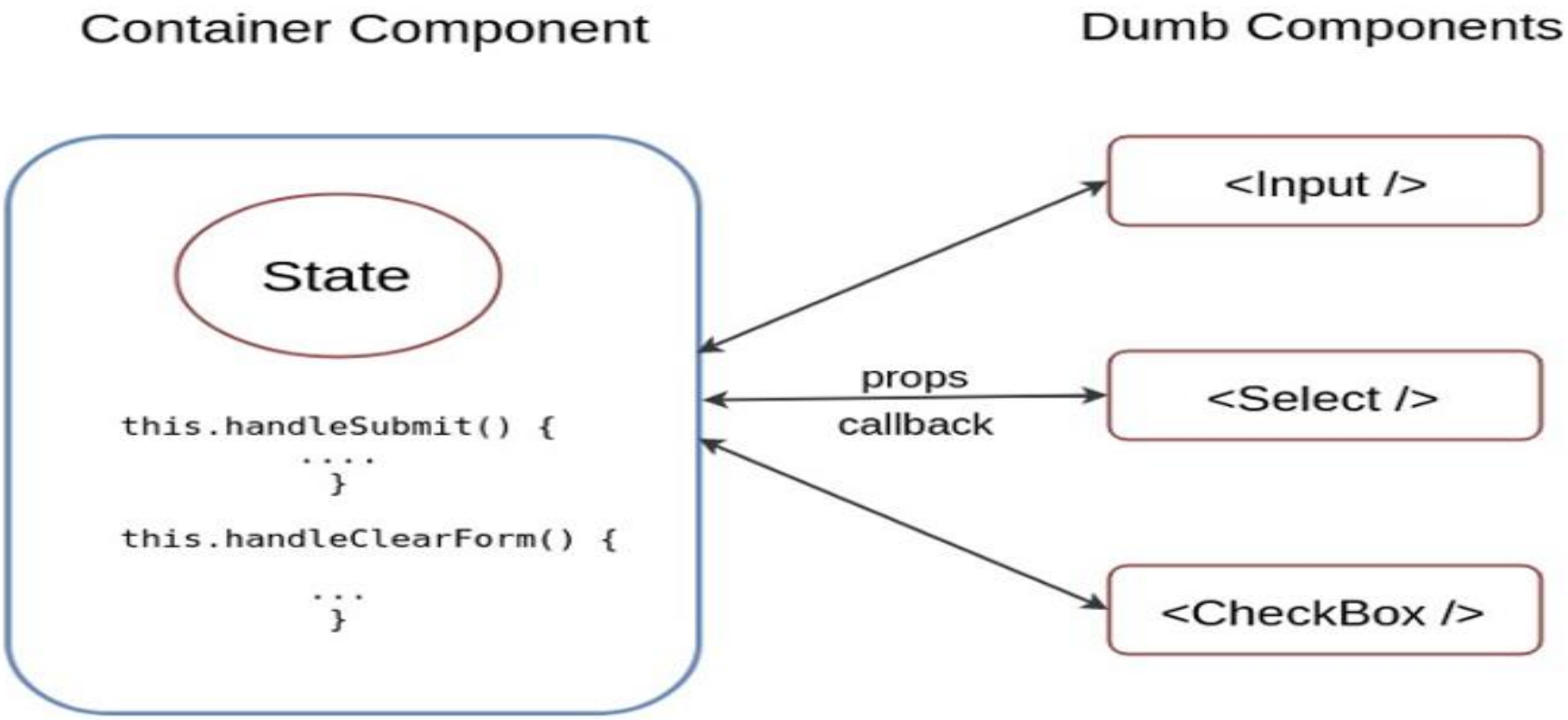
- **Pure Components in React are the components which do not re-renders when the value of state and props has been updated with the same values**

- **Takes care of "shouldComponentUpdate" implicitly**

- **Pure Components restricts the re-rendering ensuring the higher performance of the Component**

PureComp.js

Form Data Using State



FormContainer2.js

- **React Hooks was introduced from React v16.8**
- **Stateful Components can be developed ONLY using Class based Components and by using setState till version < 16.8**
- **React Hooks are special functions that let us hook into the React state and lifecycle features from function components. By this, we mean that hooks allow us to easily manipulate the state of our functional component without needing to convert them into class components**
- **Stateful Components can be developed using Functional Components and by using useState hook from version > 16.8**

- **useState is a Hook that lets you add React state to function components**

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

```
const [count, setCount] = useState(initialcount);
```

- **Only argument to useState is the initial state.**
- **Unlike this.state in class based components, the state in useState need not be compulsarily an object. state can be any type**
- **useState returns two variables which can be destructured:**
  - **The first variable is the value. (Similar to this.state)**
  - **The second variable is a function to update that value. (Similar to this.setState)**

StateHookExample.js

- **Unlike this.setState in a class based components, useState hook does NOT merge the old and new state together**
- **can use the State Hook more than once to store different values in the state within the same component**
- **Generally Variables “disappear” when the function exits but state variables are preserved by React**

MultipleStateHookExample.js

MultipleStateHookExample1.js

- **Effect Hook allows perform side effects in function components**

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

- **Data fetching (AJAX), or setting up a subscription or manually changing the DOM in React components are all examples of side effects**

- **useEffect Hook is componentDidMount, componentDidUpdate, and componentWillUnmount combined**

```
useEffect(didUpdate , [deps]);
```

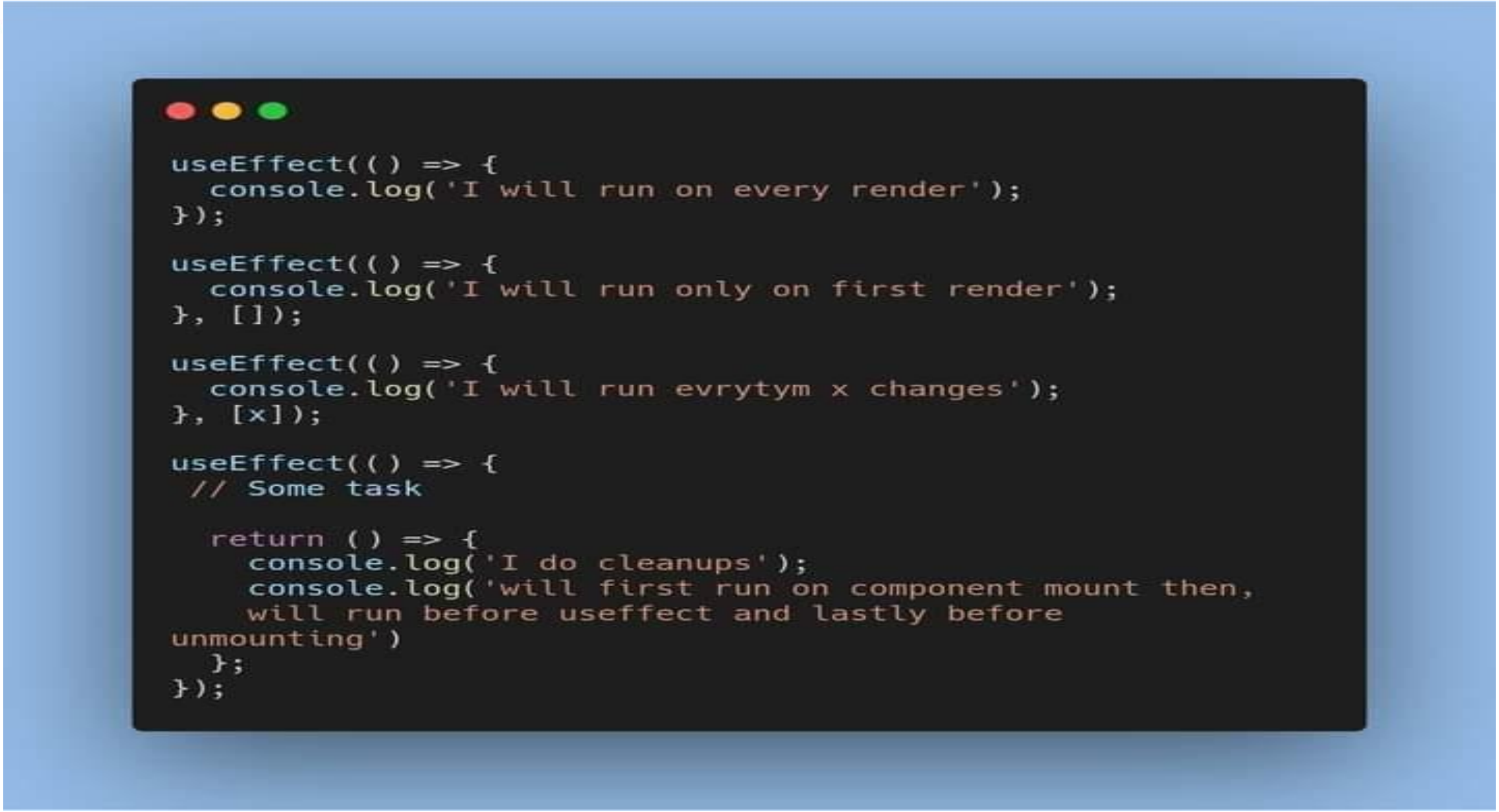
- **didUpdate() is a function which will be called after every render if no dependencies are mentioned**

```
useEffect(didUpdate , [deps]);
```

- **deps is an optional Array of dependencies and didUpdate() will run only if there are changes in any of the dependencies.[to avoid unnecessary renders]**
- **useEffect() can be called multiple times**
- **If empty Array is passed as second argument, didUpdate() once (mount time)**
- **If useEffect returns a callback function, this function will be called during unmount similar to that of componentWillUnmount**

EffectHookExample.js

EffectHookCleanupExample.js



```
useEffect(() => {
  console.log('I will run on every render');
});

useEffect(() => {
  console.log('I will run only on first render');
}, []);

useEffect(() => {
  console.log('I will run evrytym x changes');
}, [x]);

useEffect(() => {
 // Some task

  return () => {
    console.log('I do cleanups');
    console.log('will first run on component mount then,
    will run before useffect and lastly before
unmounting')
  };
});
```

- Context provides a way to pass data through the component tree without having to pass props down manually at every level
- alternative to “prop-drilling” up and down your component tree
- Instead of passing local data around and through several layers of components, it creates a global state
- Examples : data that needs to be shared across components (data such as themes, authentication, preferred language etc)
- Create a context and use Provider to provide the context to the component tree and then consume the context using Consumer

- **Create a context using React. createContext**

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

```
const colors = {
  blue: "#03619c",
  yellow: "#8c8f03",
  red: "#9c0312"
};

export const ColorContext = React.createContext(colors.blue);
```

- **Provide the context to whatever branch needs it using Context.Provider**

```
import { ColorContext } from "./ColorContext";

function App() {
  return (
    <ColorContext.Provider value={colors}>
      <Home />
    </ColorContext.Provider>
  );
}
```

- **This provides the context to the rest of the component (represented by the Home component). No matter how far a component is away from the Home component, as long as it is somewhere in the component tree, it will receive the ColorContext**

- **Consumer allows to use the value of the context**
- **<Context.Consumer> is available in both class and functional components**

```
<ColorContext.Consumer>
   {colors => <div style={colors.blue}>...</div>}
</ColorContext.Consumer>
```

- **Using useContext Hook**

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

const MyComponent = () => {
const colors = useContext(ColorContext);

  return <div style={{ backgroundColor: colors.blue }}>...</div>;
};
```

UseContextHookExample.js

- **Accessing the DOM nodes or React elements**

```
const CustomTextInput = () => {
const textInput = useRef();

 focusTextInput = () => textInput.current.focus();

 return (
  <React.Fragment >
   <input type="text" ref={textInput} />
   <button onClick={focusTextInput}>Focus the text input</button>
  </React.Fragment >
 );
}
```

UseRefHookExample.js

- **Keeping a mutable variable**
- **Keep the data between re-renders**

Class components:

Component state: Every time the state changes, the component will be re-rendered.

Instance variable: Variable will persist for the full lifetime of the component. Changes in an instance variable won't generate a re-render

Functional components:

State variable: useState or useReducer. Updates in state variables will cause a re-render of the component.

ref: equivalent to instance variables in class components. Mutating the .current property won't cause a re-render.

```
const counter = useRef(0);
const [name, setName] = useState("Karthik")

 useEffect(() => {
  // Every time the component has been re-rendered, the counter is incremented
  counter.current = counter.current + 1;
 });

 return (
  <React.Fragment >
     <h1>{`The component has been re-rendered ${counter.current} times`}</h1>
     <button onClick={() => setName(name === 'Karthik' ? 'Raman' : 'Karthik')}>
       Toggle
     </button>
  </React.Fragment >
 );
```

UseRefHookExample1.js

```
const intervalRef = useRef();
    useEffect(() => {
          const id = setInterval(() => {
          console.log("A second has passed" );
       }, 1000);
       intervalRef.current = id;
     },[]);

   const handleCancel = () => {
       console.log("clearInterval");
       clearInterval(intervalRef.current);
   }

   return (
       <React.Fragment>
          <button onClick={handleCancel}>Clear</button>
       </React.Fragment>
    );
```

UseRefHookExample2.js

Memoization

- **Memoization is basically an optimization technique which passes a complex function to be memoized or remembered (kind of memorization)**
- **Result is remembered, when the same exact parameters are passed-in and function is not executed when same parameters are passed again.**
- **Memoization is caching the result of expensive function calls and returning the cached version when the arguments are the same.**
- **Memoization optimizes our components, avoiding complex re-rendering when it isn't intended**

ExamplewithoutuseCallback.js

- **In the example shown, functions are re-instantiated after every re-renders This is proved using Set [to store unique values] but we observe that every time the render happens the functions get re-instantiated.**

- **useCallback gives you referential equality between renders for functions**

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

```
useCallback(fn, deps)
```

- **useCallback returns a memoized callback function and this callback function can be called later**

UseCallbackHookExample.js

- **useMemo very similar to useCallback, the difference is that useCallback returns a memoized callback which can be executed later but useMemo returns a memoized value, the result of that function call**

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

```
memoizedValue = useMemo(() => fb(a, b), [deps]);
```

When useMemo is called the first time, the passed in function runs, and returns a result. the result is cached for later runs. If the dependencies do not change on a second run, the result from the first run is passed back, and the passed in function is not run. When the dependencies change, the function is run and the cached result is updated.

UseMemoHookExample.js

- **useCallback returns a memoized callback which can be executed later**
  useMemo returns a memoized value, the result of that function call

- **One should NOT use 'useCallback' and 'useMemo' for everything. 'useMemo' should be used for big data processing while 'useCallback' is a way to add more dependency to your code to avoid useless rendering**

```
const f = () => { ... }

// The following are equivalent
const callbackF = useCallback(f, [])
const callbackF = useMemo(() => f, [])
```

UseCallbackUseMemoExample.js

- **The signature of useLayoutEffect is identical to useEffect, but it fires synchronously after all DOM mutations**

useEffect runs asynchronously and after a render is painted to the screen.

React renders the component
The screen is visually updated
THEN useEffect runs

useLayoutEffect runs synchronously after a render but before the screen is updated

React renders the component
useLayoutEffect runs, and React waits for it to finish.
The screen is visually updated

useEffect is the right choice most of the time. If the code is causing flickering, switch to useLayoutEffect

UseLayoutEffectExample.js

- **useReducer React Hook works on the concept of Javascript reduce method**
- **a reducer is that it will apply a bit of logic to a group of values and end up returning a single result**

Javascript reduce example

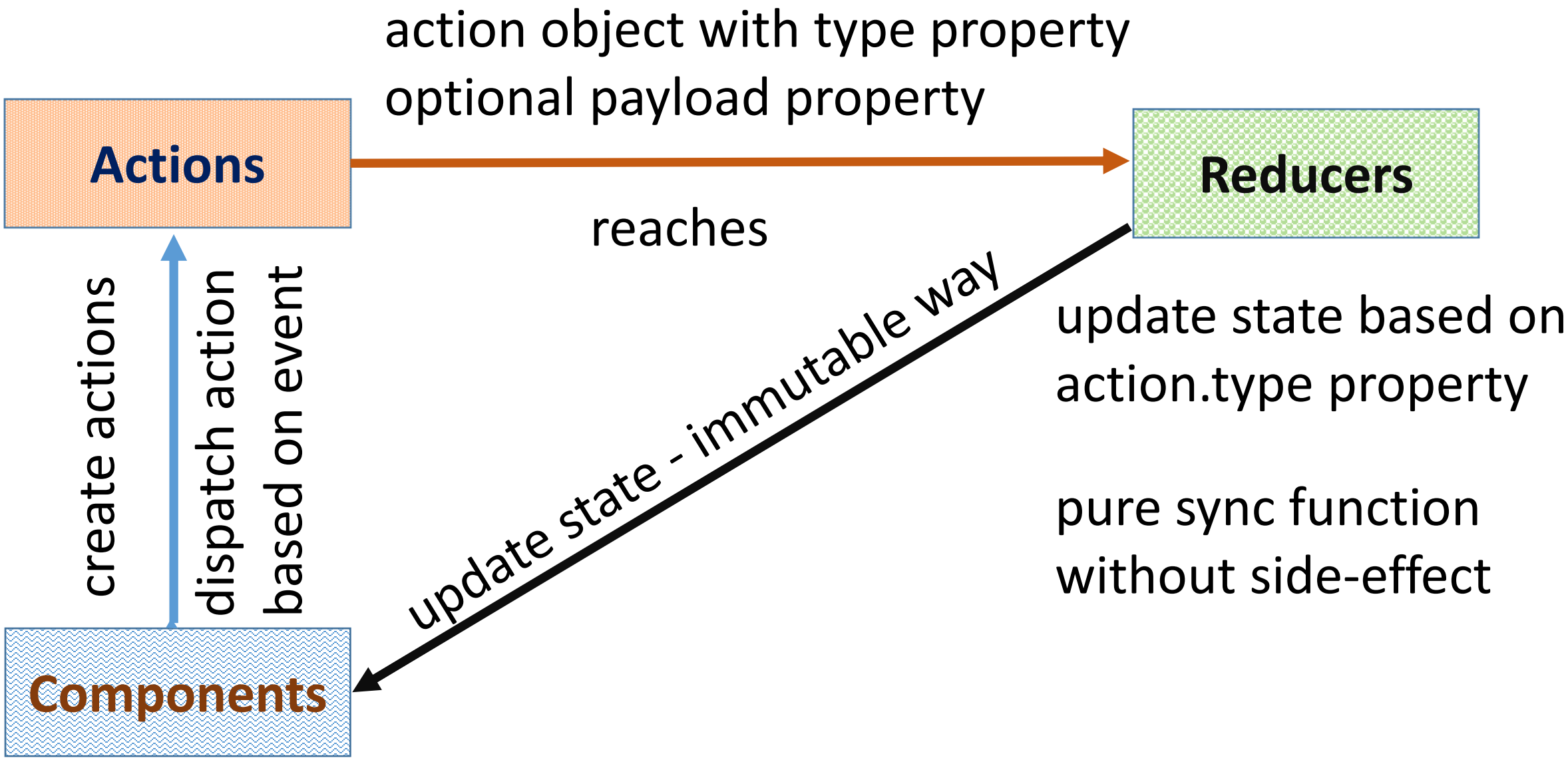
```
const numbers = [1, 2, 3];

const reducer = (tally, item) => { return tally + item }

InitValue = 0;

const total = numbers.reduce(reducer, InitValue);       //Result is 6
```

- **useReducer works with states and actions**
- **Alternative to useState**

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

```
const [state, dispatch] = useReducer(reducer, initialState);
```

- **Define an initial state.**
- **Provide a function (reducer) that contains actions to update the state.**
- **Returns the state and dispatch function**
- **dispatch function is used to trigger the reducer to update the state**
- **reducer function (prevstate , action) as input and returns the next state**
- **Action object comes with a mandatory type property and an optional payload:**
  - **type property chooses the conditional state transition**
  - **payload provides information for the state transition**



- **The state processed by a reducer function is immutable. That means the incoming state -- coming in as argument -- is never directly changed. reducer function always must return a new state object**
- **Reducer must be pure function without side effects**

UseReducerHookExample1.js

UseReducerHookExample2.js

UseReducerHookExample3.js

- **when the State to be managed is simple use useState**
- **when the State to be managed is complex and relies on another value of another element in your state, then use useReducer**

Hooks are JavaScript function, but they impose additional rules:

- Hooks can be called only from React function components
- Call Hooks only at the top level.
- Don’t call Hooks inside loops, conditions, or nested functions. This rule ensures Hooks are called in the same order each time a component renders
- Never call a Hook from a regular function
- Hooks can be called in Custom Hooks [building your own Hooks] Custom Hooks are JavaScript function whose name starts with ”use”

CustomHookExample1.js

- **Higher-order functions are regular functions that do one or both of the following:**
  - **Takes one or many functions as arguments**
  - **Returns a function**

```
function twice(f, v) {  // Function as Argument
  return f(f(v));
}
function add3(v) {
  return v + 3;
}
const result = twice(add3, 1);
console.log("Result is ", result);  //  Result is 7
```

```
// Returns a function
function addX(x) {
return function (y) {
        return x + y;
      }
    }
console.log(addX(1)(5));
// will return 6
```

- **Higher-order components (HOC) work on a similar concept of Javascript Higher Order Functions**
- **A higher-order component is a function that takes a component and returns a new component**

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

- **higher-order component pattern**

```
const higherOrderComponent = (WrappedComponent) => {
  class HOC extends React.Component {
    render() {
      return <WrappedComponent />;
    }
  }

  return HOC;
};
```

HOCExample1.js

HOCExample2.js

HOCExample3.js

- To communicate to the server through AJAX, HTTP Library is required
- Most popular libraries are
  - Axios - Promises way
  - Superagent - Callback way
  - Fetch API

- Axios

```
npm install --save axios
```

- Get Requests

```
axios.get ("API_URL")
.then(response => {
    // manipulate the response here
})
.catch(function(error) {
    // manipulate the error response here
});
```

AxiosGetExample.js

- **POST Requests** **Allow CORS in Chrome Browser**

```
axios.post ("API_URL“, { data } )
.then(response => {
   // manipulate the response here
})
.catch(function(error) {
   // manipulate the error response here
});
```

AxiosPostExample.js

- **DELETE Requests**

```
axios.delete ("API_URL“/data)
.then(response => {
   // manipulate the response here
})
.catch(function(error) {
   // manipulate the error response here
});
```

AxiosDeleteExample.js

- **Using async and await**

```
handleSubmit = async event => {
  event.preventDefault();

try {
  // Promise is resolved and value is inside of the response const.
  const response = await axios.delete(`users/${this.state.id}`);
  console.log(response);
  console.log(response.data);
} catch (error) {
    // manipulate the error response here
});
```

AxiosDeleteExample1.js

- **Multiple concurrent requests**

```
function getUserAccount() {
  return axios.get('/user/12345');
}

function getUserPermissions() {
  return axios.get('/user/12345/permissions');
}

axios.all ([getUserAccount(), getUserPermissions()])
  .then( axios.spread (function (acct, perms) {
    // Both requests are now complete
  }));
```

- **Axios APIs with request.config**

- **GET request config**

```
axios ({
  method: 'get',
  url: 'http://URL',
  responseType: 'json'
})
```

- **POST request config**

```
axios ({
  method: 'post',
  url: '/user/12345',
  data: {
    firstName: 'Fred',
    lastName: 'Flintstone'
  }
});
```

- **Global Axios Default Values**

```
axios.defaults.baseURL = 'https://api.example.com';
axios.defaults.headers.common['Authorization'] = AUTH_TOKEN;
axios.defaults.headers.post['Content-Type'] = 'application/x-www-form-urlencoded';
```

- **Axios Instance**

```
import axios from 'axios';

export default const instance = axios.create ({
  baseURL: ‘http://api.example.com /’,
  timeout: 1000,
  headers: {'X-Custom-Header': 'clayfin'}
});

instance.defaults.headers.common['Authorization'] = AUTH_TOKEN;
```

- **Request Interceptors**

```
axios.interceptors.request.use(function (config) {
    // Do something before request is sent
    return config;
  }, function (error) {
    // Do something with request error
    return Promise.reject(error);
  });
```

- **Response Interceptors**

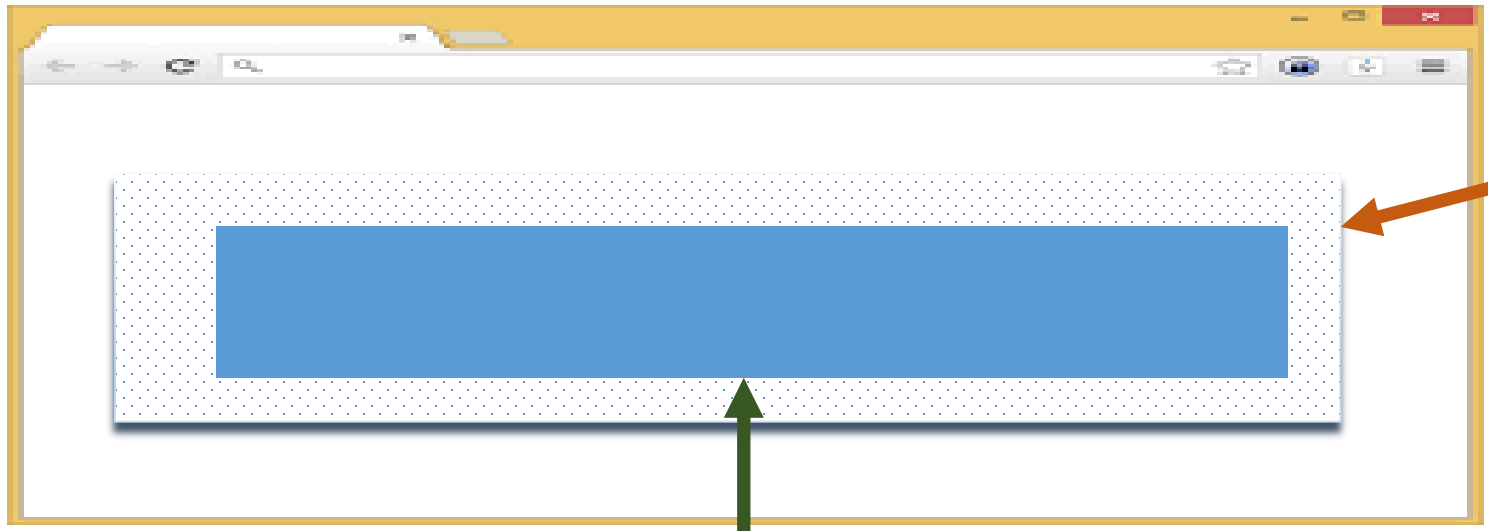
```
axios.interceptors.response.use(function (response) {
    // Do something with response data
    return response;
  }, function (error) {
    // Do something with response error
    return Promise.reject(error);
  });
```

AxiosInteceptorsExample.js

- **React Router is the standard routing library for React which helps to navigate through a React application with multiple Views**

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

- **Different Types of Routers**
  - **BrowserRouter - client side routing with URL segments (uses History API)**
  - **HashRouter - uses the hash portion of the URL (window.location.hash)**
  - **MemoryRouter - keeps the history of your "URL" in memory (does not read or write to the address bar**
  - **StaticRouter - never changes location and useful in server-side rendering**
- **History**
  - **router creates history object to keep track of the current location of the page**
- **Route - to render some UI when its path matches the current URL**
- **Link - generate Link to the route**
- **NavLink – styles the active element in the Link**



<BrowserRouter>

?

/

/products

/admin

Homepage Component
<Route>

Products Component
<Route>

Admin Component
<Route>

<BrowserRouter>

<MemoryRouter>

RoutingExample1.js

RoutingExample2.js

- **match prop inside the component**
  - **The match object contains information about how a <Route path> matched the URL**

length : (number), the number of entries in the history stack
params : (object), key/value pairs parsed from the URL corresponding to the dynamic segments of the path
isExact : (boolean), true if the entire URL was matched (no trailing characters)
path : (string), the path pattern used to match
url : (string), the matched portion of the URL.

- **history object allows you to manage and handle the browser history inside the components**

length : (number), the number of entries in the history stack
action : (string), the current action (PUSH, REPLACE or POP)
location : (object), the current location
push(path, [state]) : (function), pushes a new entry onto the history stack
replace(path, [state]) : (function), replaces the current entry on the history stack
go(n) : (function), moves the pointer in the history stack by n entries
goBack() : (function), equivalent to go(-1)
goForward() : (function,) equivalent to go(1)
block(prompt) : (function), prevents navigation

- **Location prop represent where the app is now, where you want it to go, or even where it was**

It's also found on history.location but you shouldn't use that because it's mutable.

A location object is never mutated so you can use it in the lifecycle hooks to determine when navigation happens

pathname : (string) The path of the URL
search : (string) The URL query string
hash : (string) The URL hash fragment
state : (object) location-specific state that was provided to e.g. push(path, state) when this location was pushed onto the stack

- **Routes Array and use Javascript function to dynamically create the Route**

RoutingExample3.js

- **withRouter HOC**
  - **to access react-router related information programmatically**

  React Router props are not passed on to the Nested Children

  Router props can be passed on to the Nested Children through withRouter HOC

- **history object contains a method called history.listen which will trigger your callback function whenever a route changes, allowing you to respond to that routing event**

RoutingExample4.js

- **useHistory hook gives access to the history object which can be used to navigate routes**

```
import { useHistory } from "react-router-dom"

const Homepage = () => {
  let history = useHistory()

  function goBack()  {   history.goBack()     }

  function goHome() {  history.push('/')    }

  return(
    <div>
      <button onClick={goBack}>Previous</button>
      <button onClick={goHome}>Home</button>
    </div>
  )
}
```

- useParams hook gives access to the params of the given route.
- Params are just parameters on a given URL that is dynamically set
- returns an object of key/value pairs of URL parameters

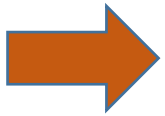
```
function Post(props) {
  let { id } = useParams()
  return <div>Now showing post {id}</div>
}

function App(){
  return(
   <div className='app'>
    <Router>
     <Switch>
      <Route exact path="/" />
       <Homepage />
      </Route>
      <Route path="/blog/:id">
       <Post />
      </Route>
     </Switch>
    </Router>
   </div>
 );  }
}
```

- **useLocation hook gives access to the location object that represents the current URL**

```
const LearnMore = () => {
 let location = useLocation()
 return(
  <div>
   You are currently at the following path {location.pathname}
  </div>
 )
}

function App(){
 return(
  <Router>
   <ul>
    <li>
     <Link to='/learn-more'>Learn More</Link>
    </li>
   </ul>
   <Switch>
    <Route path='/learn-more'> <LearnMore/> </Route>
   </Switch>
  </Router>
 )}
```

- **useRouteMatch hook attempts to match the current URL in the same way that a <Route> would but without actually rendering a <Route>**

```
function BlogPost() {
  return (
    <Route
      path="/blog/:slug"
      render={({ match }) => {
        // Do whatever you want with
        // the match...
        return <div />;
      }}
    />
  );
}
```



```
import { useRouteMatch } from "react-router-dom";

function BlogPost() {
  let match = useRouteMatch("/blog/:slug");

  // Do whatever you want with the match...
  return <div />;
}
```

- Redux is a library for managing application state in a central store

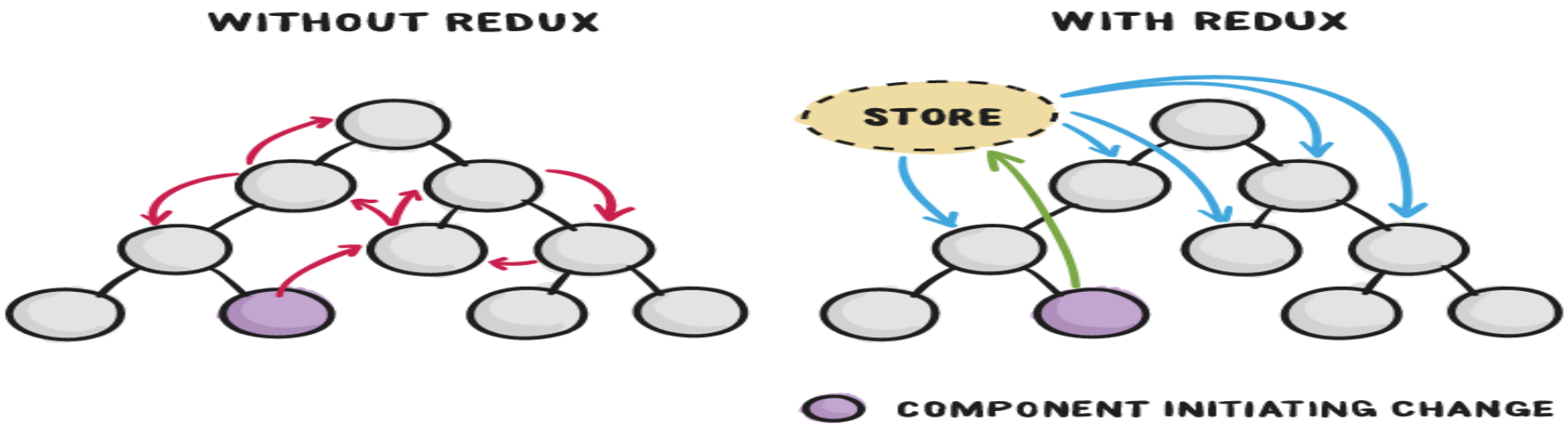
```
npm install --save redux
```

- React Redux is a binding of React Application with Redux library

```
npm install --save react-redux
```

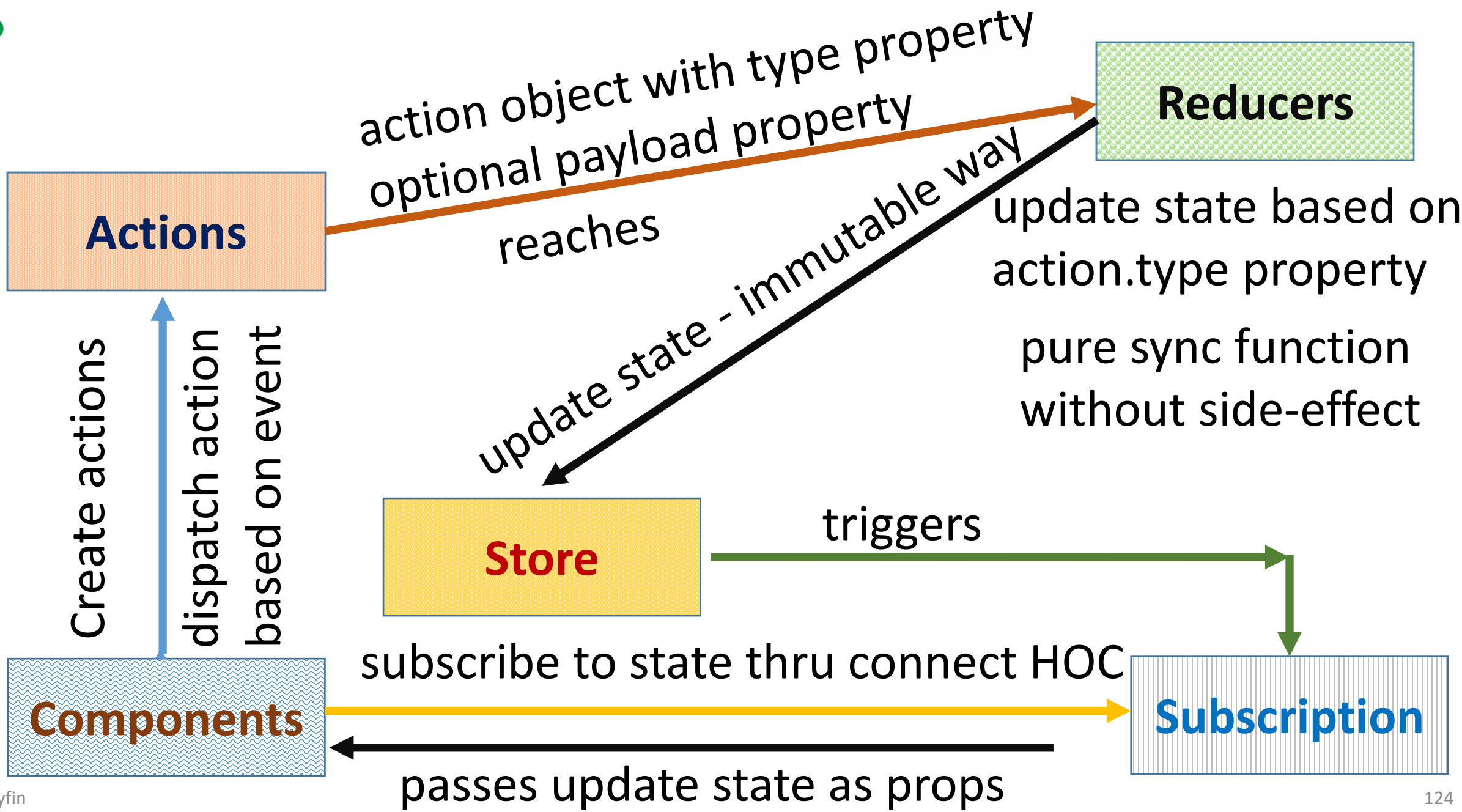
- 3 Guiding Principles of Redux :

1. Central store is the single source of truth for application state
2. dispatching an action is the only way for the application to express a need for a state change (type property drives how the state should change)
3. Reducers are the only way to manipulate the state



- keep logic and behaviours abstracted away from the UI
- Component Local Simple States - useState
- Component Local Complex States - useReducer
- Client States which needs to be shared across components - Redux

- Create a store with a reducer or combineReducers (combination of multiple reducers
- Provide the store information to the React Application through Provider
- Components subscribe to the store through Connect HOC specifying the types of action it can dispatch and the slice of state information it needs as props
- Components dispatch actions when an event occurs with the action type
- Reducers gets executed and based on the action type updates the state in the store
- Store automatically triggers this subscription for any state changes and passes the slice of the state information to the Component as props



- **Creating a central store**

```
import { createStore } from 'redux';
import { Provider } from 'react-redux';

function reducer()
{ //  reducer function to manipulate the state }

const store = createStore( reducer );

const App = () => (
  <Provider store={store}>
      <CounterApp/>
  </Provider>
);
```

Typically the central store is created in App.js before mounting the React Application Component

Reducer function to be written in a different file

Multiple Reducers

```
import { createStore, combineReducers } from
'redux';
// Combine Reducers
const reducers = combineReducers({
  userState: userReducer,
  accountsState: accountsReducer
});
const store = createStore(reducers);
```

- **Reducers**

```
const initialState = {
  count: 0
};

function reducer(state = initialState, action) {
  if(action.type === INCREMENT) {
    return {
      count: state.count + action.value
    };
  }

  return state;
}
```

update state based on
action.type property

pure sync function
without side-effect
(**no Async operations**)

update state - immutable way

Payload parameter

- **Component subscription and dispatch**

```
import { connect } from 'react-redux';

class App extends React.Component {
 render() {    return (
   <div>
     <h2>Counter : {this.props.counter}</h2>
     <button
onClick={()=>{this.props.OnIncrementCount(1)}}>+</button>
    </div>
  )  }
}
const mapStateToProps = (state, ownProps) => {
 return {
  counter: state.count
 };
}
const mapDispatchToProps =(dispatch, ownProps) => {
 return {
  OnIncrementCount: (val) => dispatch({ type: 'INCREMENT' , value : val})
 };
}
export default connect ( mapStateToProps, mapDispatchToProps ) (App);
```

Components subscribe to the central store with connect HOC

mapStateToProps – specifies the slice of state information the components requires from the store if that state is changed

mapDispatchToProps – specifies the type of action which will be dispatched by the component

actionCreators

({ type: 'INCREMENT' , value : val })

React Redux DevTools for a better development experience

Add Redux DevTools as Extensions to the Chrome Web Store

Details about this can be found at : https://github.com/zalmoxisus/redux-devtools-extension

Redux option in the Developer Tools will be enabled

Add this in the createStore method as a parameter to access the store thru Redux DevTools

```
const store = createStore(
   reducer,
   window.__REDUX_DEVTOOLS_EXTENSION__ && window.__REDUX_DEVTOOLS_EXTENSION__()
 );
```

Don’t forget to remove it before deploying to production!

- **Redux Examples**

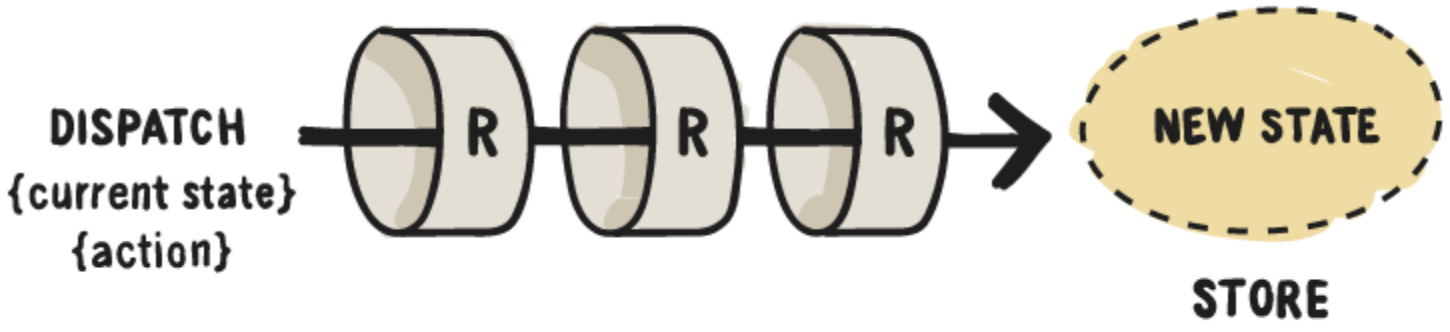
StatewithoutRedux.js

ReduxExample1.js

ReduxExample2.js

ReduxExample3.js

- **combinedReducers Examples**



ReduxExample4.js

ReduxExample5.js

Redux Hooks

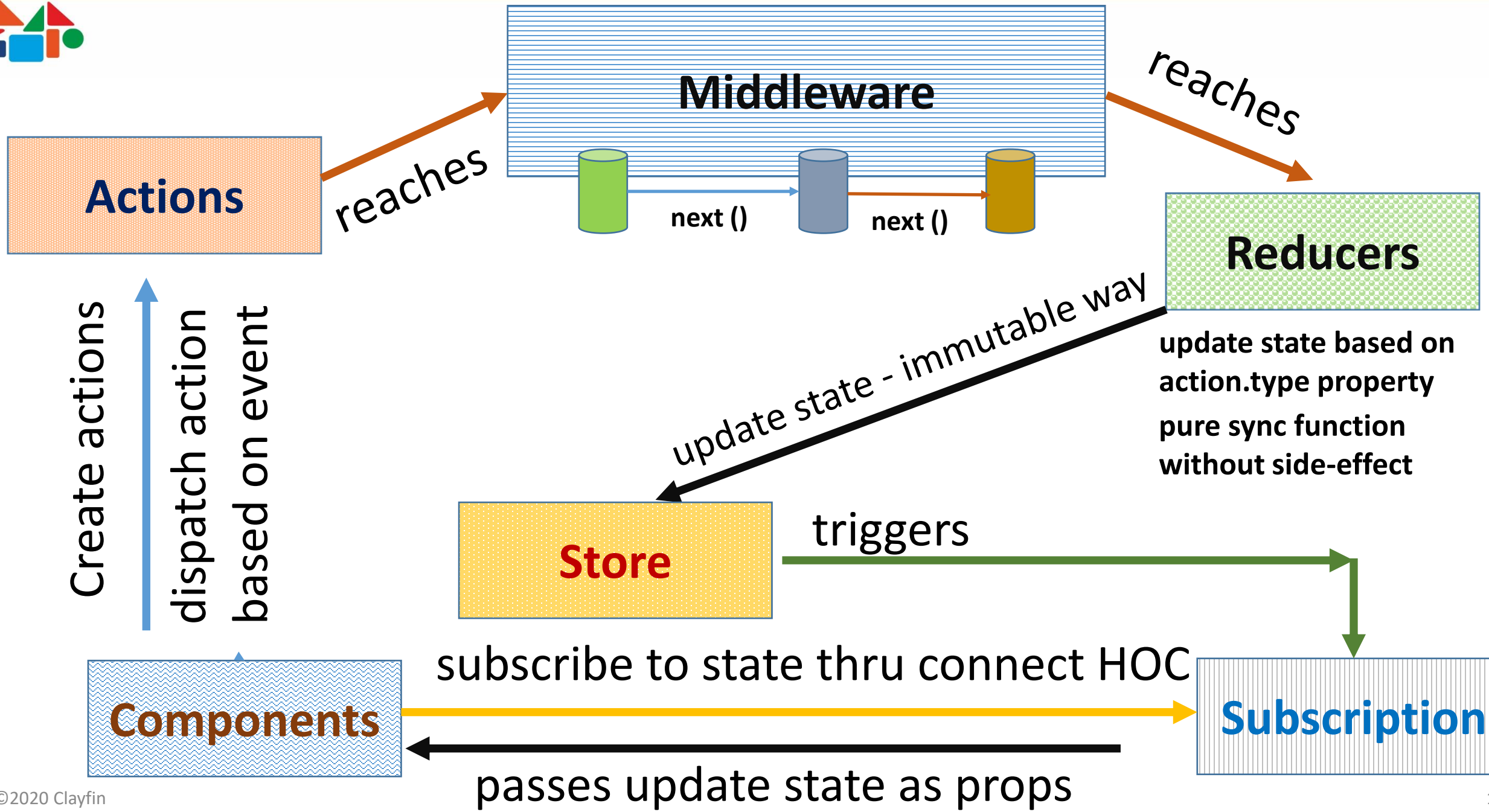
- **useSelector -** equivalent of **mapStateToProps**

- **useDispatch -** equivalent of **mapDispatchToProps**

ReduxHooks.js

- **Redux Middleware function provides a medium to interact with dispatched action before they reach the reducer**
- **Middleware intercepts the action object before Reducer receives it and gives the functionality to perform additional actions or any enhancements with respect to the action dispatched**
- **Middleware can have side-effects. Async operations can be performed**
- **Middleware functions can be composed in a chain**

- **Few examples for Middleware usage:**
  - ❑ **action logging**
  - ❑ **async requests**
  - ❑ **handling cookies**
  - ❑ **persisting data**
  - ❑ **analytics**



- **Few Third-party React Redux middleware**

  - **redux-thunk - The easiest way to write async action creators**
  - **redux-axios-middleware - Redux middleware for fetching data with axios HTTP client**
  - **redux-logger - Log every Redux action and the next state**
  - **redux-analytics - Analytics middleware for Redux**
  - **redux-saga - An alternative side effect model for Redux apps**
  - **apollo-client - A simple caching client for any GraphQL server and UI framework built on top of Redux**

- **Syntax of a middleware**

```
({ getState, dispatch }) => next => action
```

- **Middleware is a function returning a function, which takes next as a parameter. Then the inner function returns another function which takes action as a parameter and finally returns next(action)**
- **Middleware receives store's dispatch so that they can dispatch new action**
- **getState functions as arguments so that they can access the current state**

```
function logger( { getState } ) {
  return next => action => {
    console.log('action', action);
    const returnVal = next(action);
    console.log('state when action is dispatched', getState());
    return returnVal;
  }
}
```

- **Redux Thunk middleware allows to write action creators with Async operations and then return a function instead of an action**

```
npm install --save redux-thunk
```

- **to enable Redux Thunk, use applyMiddleware()**

```
import { createStore, applyMiddleware } from 'redux';
import thunk from 'redux-thunk';

const store = createStore(reducer , applyMiddleware(thunk));
```

- **Middleware Composition**

```
const middleWares = [ thunk,  logger]; // Put the list of Middlewares in an array

// Create the store with the applied middleWares
export const store = createStore(rootReducer, applyMiddleware( ...middleWares ));
```

- Thunk Middleware allows to write an action dispatcher which returns a function instead of an action object
- When using Redux without thunk, an action is always a plain Javascript action object
- When using Redux with Thunk middleware, an action can either be a plain Javascript object, OR an action can be a function

Sync ActionCreators without Redux Thunk

```
export const increment = (value) => {
    return {
        type : ADD,
        val : value
    };
};
```

Async ActionCreators with Redux Thunk

```
function incrementAsync( value ) {
  return dispatch => {
    setTimeout(() => {
            dispatch( increment( value ) ) ;
    }, 1000);
  };
}
```

Redux Thunk will execute this code

Refer Advanced store setup at

https://github.com/zalmoxisus/redux-devtools-extension

```
import { createStore, applyMiddleware, compose } from 'redux';

const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose;
const store = createStore(reducer, /* preloadedState, */ composeEnhancers(
  applyMiddleware(...middleware))
));
```

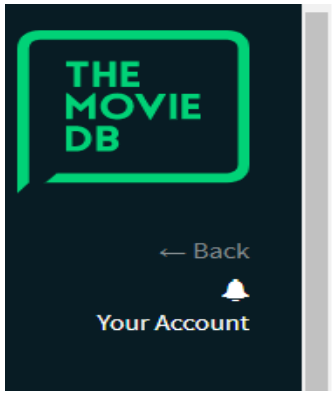
Don’t forget to remove it before deploying to production!

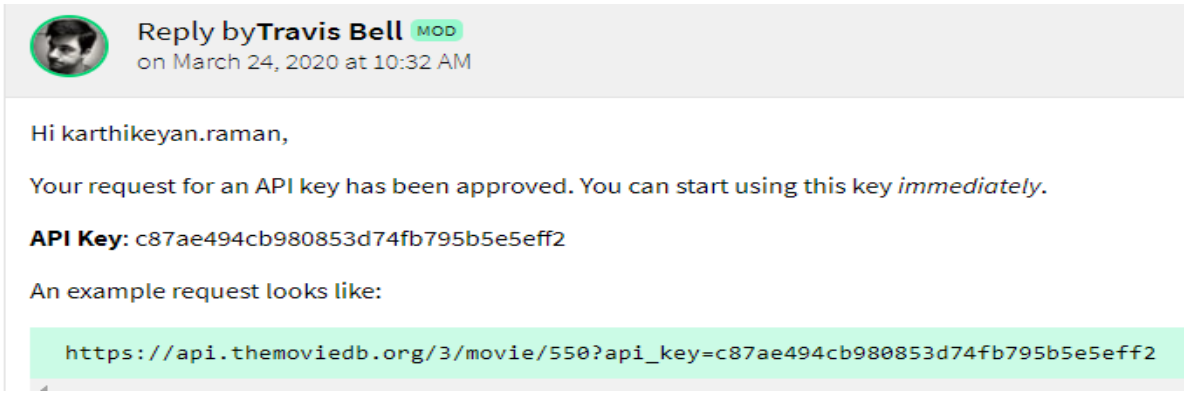
- **Redux Thunk Examples**

ReduxThunkExample1.js

ReduxThunkExample2.js

ReduxThunkExample3.js





ReduxThunkExample4.js

Webpack bundles all the javascripts in the project into one while building
File size of the bundle can be big and bundle will contain code that might never run because the user could stop on one page and never use the rest of the pages in the application

React.lazy and React.Suspense

perfect way to lazily load a dependency and only load it when needed

```
const TodoList = React.lazy(() => import('./routes/TodoList'))
const NewTodo = React.lazy(() => import('./routes/NewTodo'))
const App = () => (
<BrowserRouter >
<React.Suspense fallback={<p>Please wait</p>}>
<Switch>
<Route exact path="/" component={TodoList} />
<Route path="/new" component={NewTodo} />
</Switch>
</React.Suspense>
</BrowserRouter >
)
```

Most popular libraries : react-intl and react-i18next

react-i18next

```
npm install i18next react-i18next --save
```

```
npm install i18next-http-backend i18next-browser-languagedetector --save
```

I18next - core of the i18n functionality , react-i18next extends and glue it to react.

Create a new file `i18n.js` in src/config

Create translation files per language under src/locales

Example : src/locales/en.json - English

src/locales/fr.json - French

changeLanguage and Translation : withTranslation HoC useTranslation Hook

I18Example

Thank You

www.clayfin.com