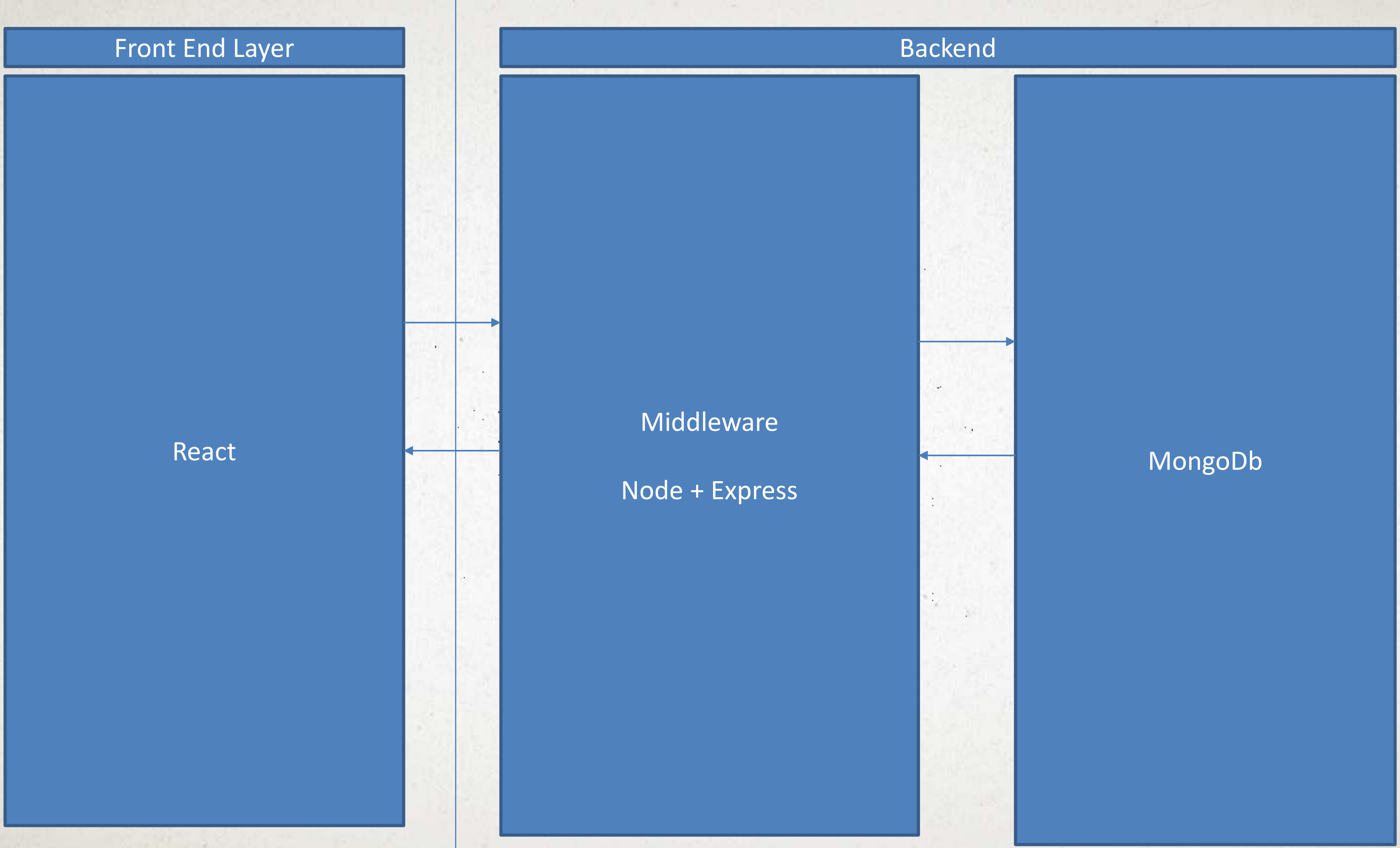
REACT JS

INTRODUCTION


Front End Layer
Backend
React
Middleware
Node + Express
MongoDb

CONTENT

WHAT IS REACT?

- An Open Source JavaScript Library For Building User Interfaces
- Not a Framework
- Renders your UI and responds to events.
- Rich Ecosystem
- It also uses the concept called Virtual DOM, creates an in-memory data structure cache, enumerates the resulting differences, and then updates the browser’s displayed DOM efficiently.
- One of the unique features of React.js is not only it can perform on the client side, but it can also be rendered on the server side, and they can work together interoperably.

WHY REACT?

- Created and maintained by Facebook
- Huge community support
- In demand Skillset
- Reusable Components
- React is declarative
- Seamlessly integrate react into any of your applications
- React Native for Mobile applications

PREREQUISITES

- HTML, CSS and Javascript Fundamentals
- ES6
- Javascript – ‘this’ keyword, filter, map and reduce
- ES6
    - Let & const
    - Rest and spread operator
    - Destructuring assignment
    - export default and import

Angular vs REACT

Angular has

- modules
- controllers
- directives
- scopes
- templating
- linking functions
- filters
- dependency injection

WHAT IS REACT?

~~Angular~~ React has **JUST COMPONENT**

- ~~modules~~
- ~~controllers~~
- ~~directives~~
- ~~scopes~~
- ~~templating~~
- ~~linking functions~~
- ~~filters~~
- ~~dependency injection~~

React has

- One way Data Flow
- React Element is a JS object

First React App

- Download Node
- Download VS Code

- npx create-react-app hello-world (npm package runner)
- cd hello-world
- **npm start**

- npm install create-react-app –g
- create-react-app hello-world

Folder Structure – React Application

Node_modules – Required to store the dependencies

Public – To hold the resources for web app

Src – Contains the App specific code

Readme.md

Package.json

Package-lock.json

.gitignore – ignores folders and files from pushing it to the repo

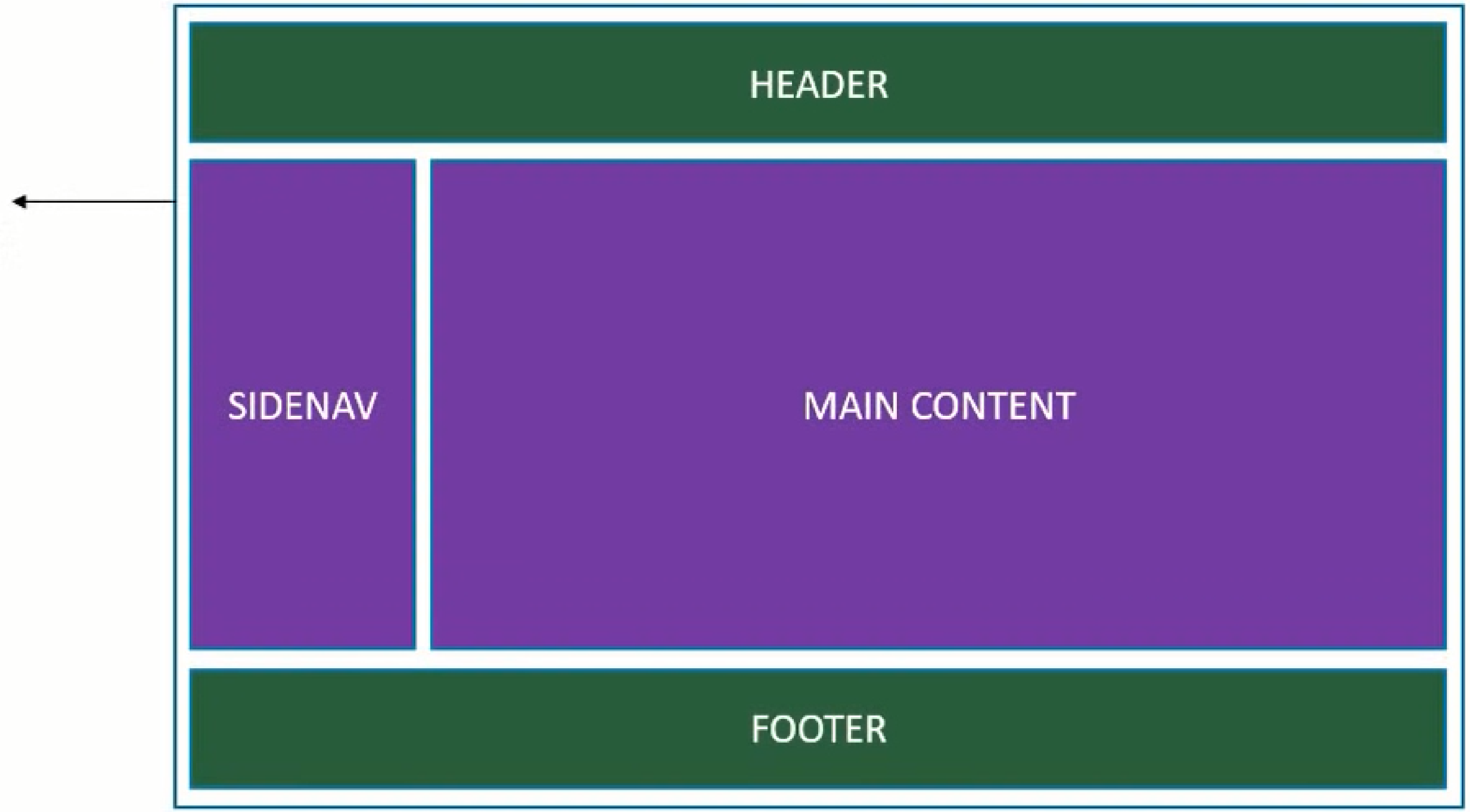
Index.js - `ReactDOM.render( <App/>, document.getElementById('root'));`

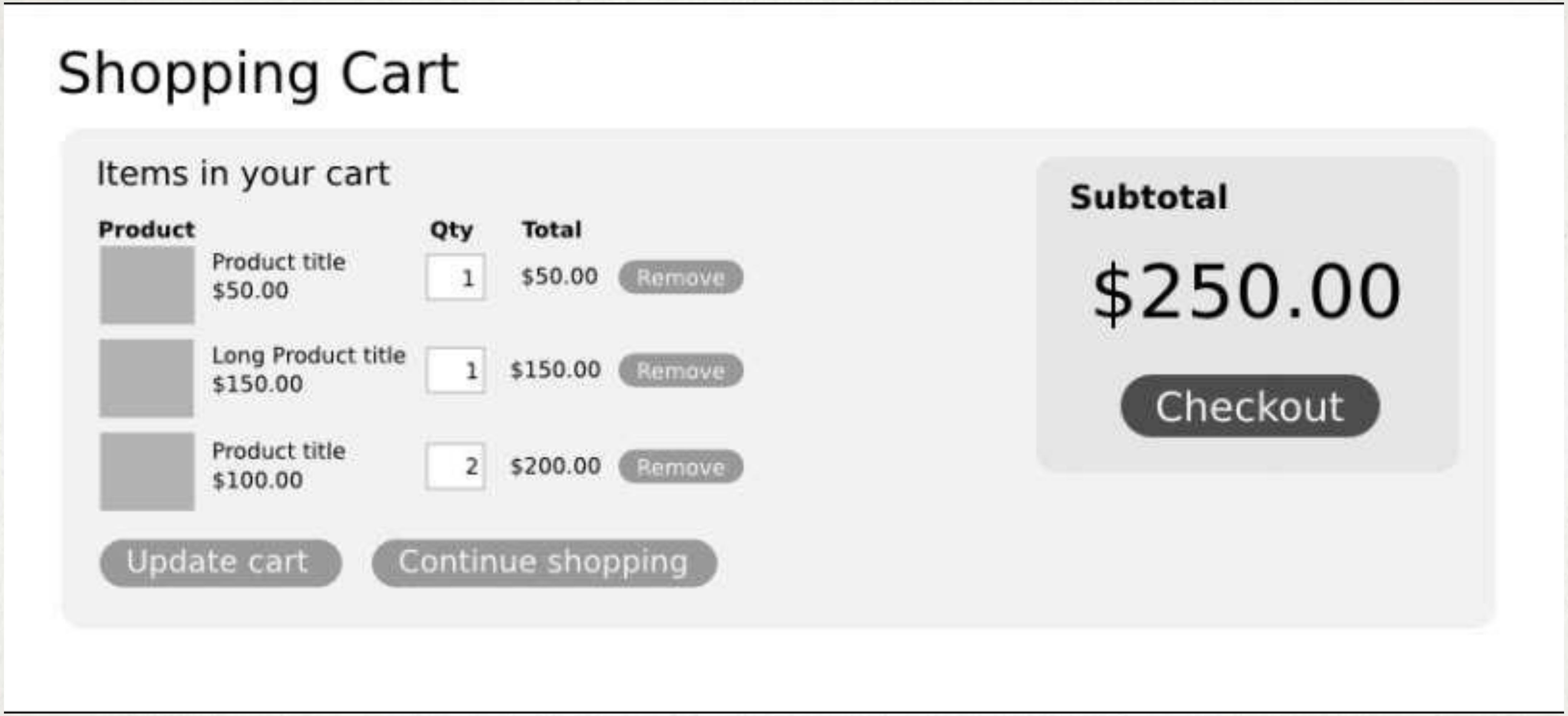
COMPONENT

- Components describe a part of the user interface
- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.
- Components can be nested inside other components
- 2 Types of Components in React
  - Functional components
  - Class components

Components

Root (App) Component





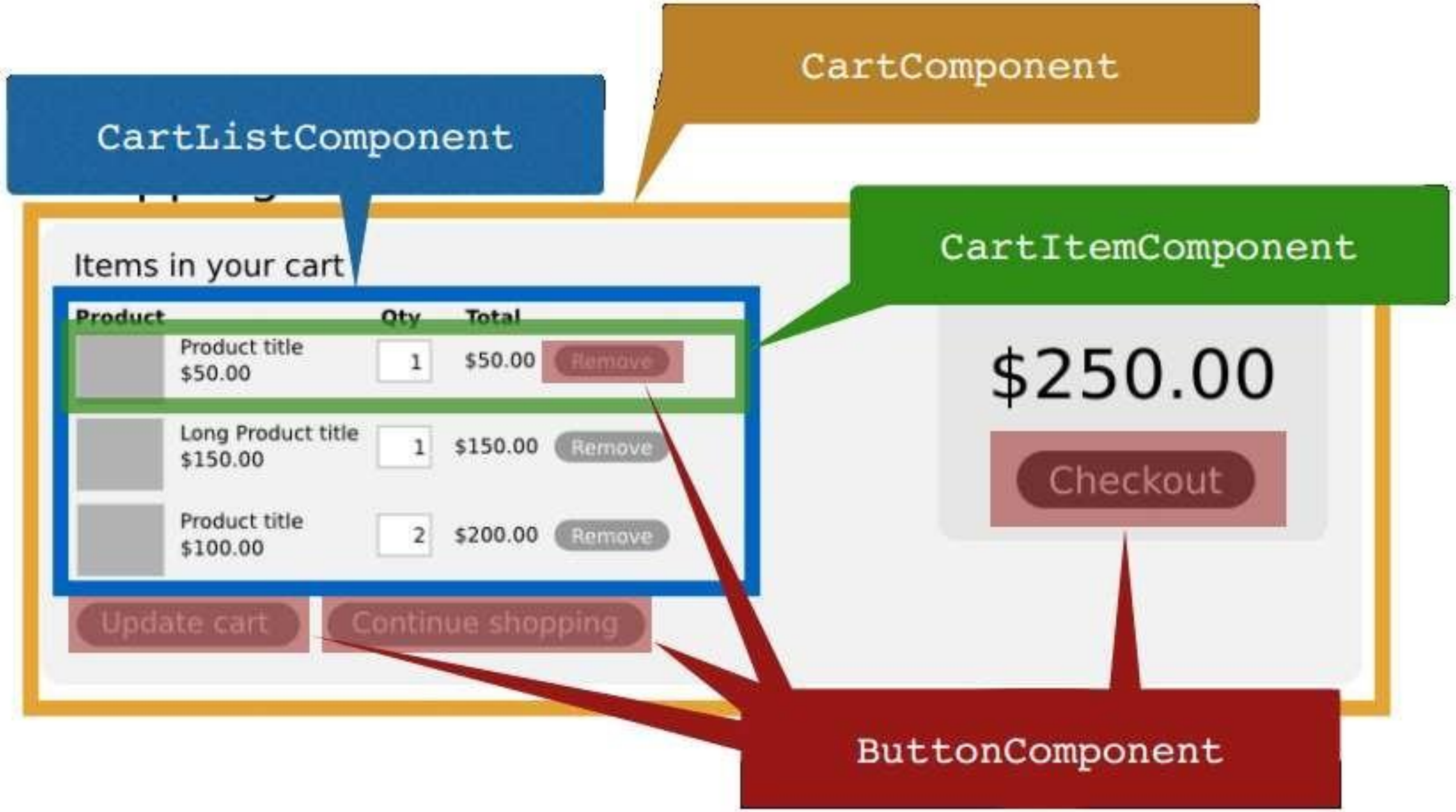
Shopping Cart

Items in your cart

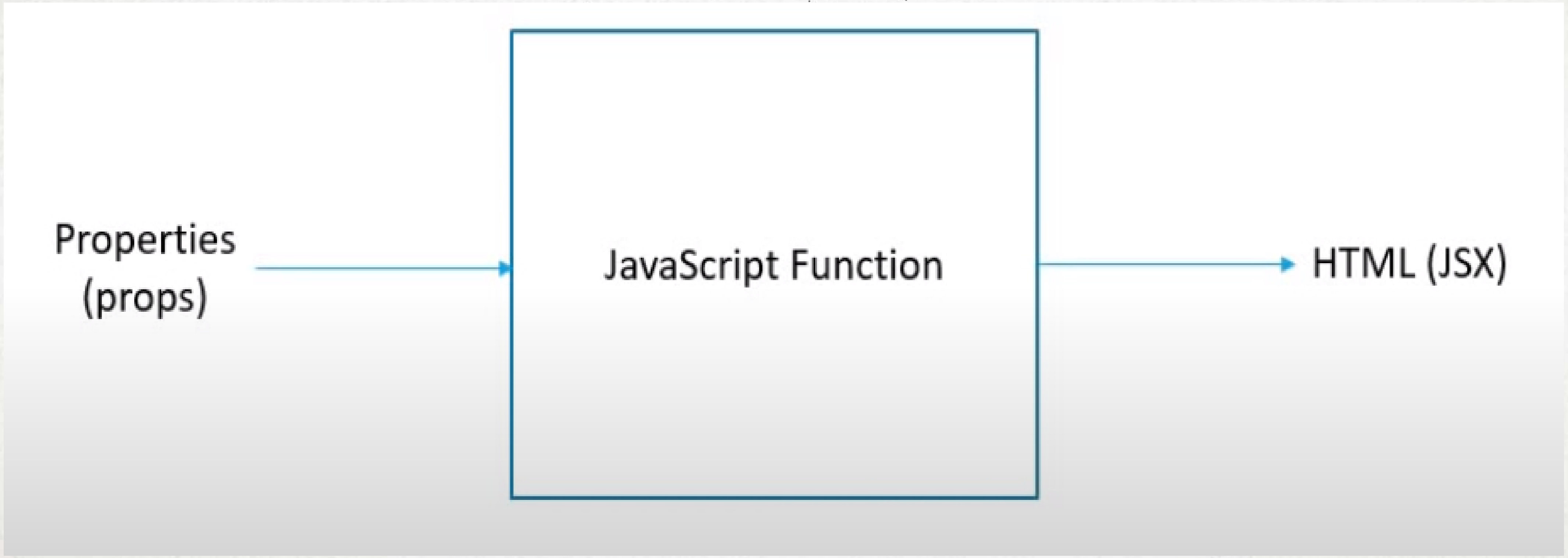
Product		Qty	Total	
	Product title $50.00	1	$50.00	Remove
	Long Product title $150.00	1	$150.00	Remove
	Product title $100.00	2	$200.00	Remove

Subtotal

$250.00


CartComponent
CartListComponent
CartItemComponent
Items in your cart
Product
Qty
Total
Product title
$50.00
1
$50.00
Remove
Long Product title
$150.00
1
$150.00
Remove
Product title
$100.00
2
$200.00
Remove
Update cart
Continue shopping
$250.00
Checkout
ButtonComponent

FUNCTIONAL COMPONENT



CLASS COMPONENT

```
class TodoInput extends React.Component {

  render() {

    return (
      <div className="form-inline">
        <input className="form-control" type="text" value={this.state.content}
               onChange={this.updateState}/>
      </div>
    );
  }
}
```

JSX

- Javascript XML – Extension to the Javascript language syntax
- JSX = Javascript + XML.
- JSX tags have tag name, attributes and children.
- JSX makes your react code simple and elegant

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

- class - > className
- for -> htmlFor
- onclick -> onClick

JSX

```
<script>
 var helloEl = React.createElement('div', { className: 'hello' }, 'Hello,
                world!');
 React.render(
  helloEl,
  document.body
 );
</script>
```

```
<script type="text/jsx">
 var helloEl = <div className: "hello">Hello, world!</div>;
 React.render(
  helloEl,
  document.body
 );
</script>
```

JSX

```
<script>
 var helloEl = React.createElement('div', { className: 'hello' }, 'Hello,
 world!');

 React.render(
  helloEl,
  document.body
 );
</script>
```

```
<script type="text/jsx">
 var helloEl = <div className: "hello">Hello, world!</div>;
 React.render(
  helloEl,
  document.body
 );
</script>
```

COMPONENT - PROPS

- Props is what you pass into the Component via attributes.
- Props is way to input data to the component.
- Props are immutable.
- Container component will define data that can be changed (attributes that are called props)
- Child Component will received data from parent component via props.

- {props.children} – will render the children inside the component tag

props vs state

props	state
props get passed to the component	state is managed within the component
Function parameters	Variables declared in the function body
props are immutable	state can be changed
props – Functional Components this.props – Class Components	useState Hook – Functional Components this.state – Class Components

COMPONENT – PROPS (in Class Component)

```
import React from 'react';

class App extends React.Component {
  render() {
    return (
      <div>
        <h1>{this.props.headerProp}</h1>
        <h2>{this.props.contentProp}</h2>
      </div>
    );
  }
}
export default App;
```

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App.jsx';

ReactDOM.render(
  <App headerProp="Header from props..."
contentProp="Content from props..."/>,
document.getElementById('app')
);

export default App;
```

COMPONENT - STATE

- Private data of component
- When state is changed -> Re-render Component
- Can’t read state from outside Component
- Do not update state directly always use setState
- Always use callback function called as second parameter in setState function

EVENT HANDLING

- Camel case for the event name
- Function name should be in the curly braces
- Also please note we have not used parenthesis for the calling function reference
- You need to bind the function to the this keyword, which is referring to the class scope

Event Binding

1. Bind the function in render function
2. Bind the event handler in the class constructor – Will use this mostly
3. Class property as arrow functions (Still an Experimental feature)

LIST RENDERING

- Map
- Key gives each element a identity
- Use index as key on when items do not have unique id and list is static list and the list never needs to be ordered or reordered

CONDITIONAL RENDERING

- If / Else
- Element Variable approach
- Ternary operator
- Short circuit operator

Lifecycle Methods

Mounting	When an instance of a component is being created and inserted into the DOM
Updating	When a component is being re-rendered as a result of changes to either its props or state
Unmounting	When a component is being removed from the DOM
Error Handling	When there is an error during rendering, in a lifecycle method, or in the constructor of any child component

Component Lifecycle Methods (only available in class components)

- **Mounting** – **constructor**, getDerivedStateFromProps, render, **componentDidMount** – called only once in lifecycle (perform AJAX Calls required for initialization)

- **Updating** – Called everytime a component is updated – getDerivedStateFromProps, shouldComponentUpdate, **render**, getSnapshotBeforeUpdate, componentDidUpdate - called only once

- **Unmounting** – componentWillUnmount – cleanup activities, cancelling subscriptions, etc.

- **Error Handling** – **getDerivedStateFromError**, **componentDidCatch**

Forms

- Controlled Components
- Uncontrolled Components

HTTP

- Axios
- Npm install axios

STYLING

- Using CSS
- Inline Styling
- Conditional Styling
- Using CSS module
- CSS in JS libraries

Fragment

- React Fragments are used to wrap multiple elements in React.Fragment without an additional enclosing tag

Pure Component

- Only renders **class** component when there is difference between props and state being passed to the component

Memo

- Works with **functional** component and rerenders the component only when there is difference in props and state

Error Boundary

A class component that implements either one or both of the lifecycle methods ***getDerivedStateFromError*** or ***componentDidCatch*** becomes an **error boundary**.

The static method *getDerivedStateFromError* method is used to render a fallback UI after an error is thrown and the *componentDidCatch* method is used to log the error information.

Error Boundary

Error Boundary catches error during rendering in lifecycle methods and in the constructors in the whole tree below it, it does not catches error in event handlers for that you need to use try, catch

Destructuring

1. Array Destructuring
2. Object Destructuring

UseState Hook

- useState hook allows you to add state to functional component
- useState does not automatically appends to the state object , you need to use **spread operator** to copy the state variables and then call setter function

Portals

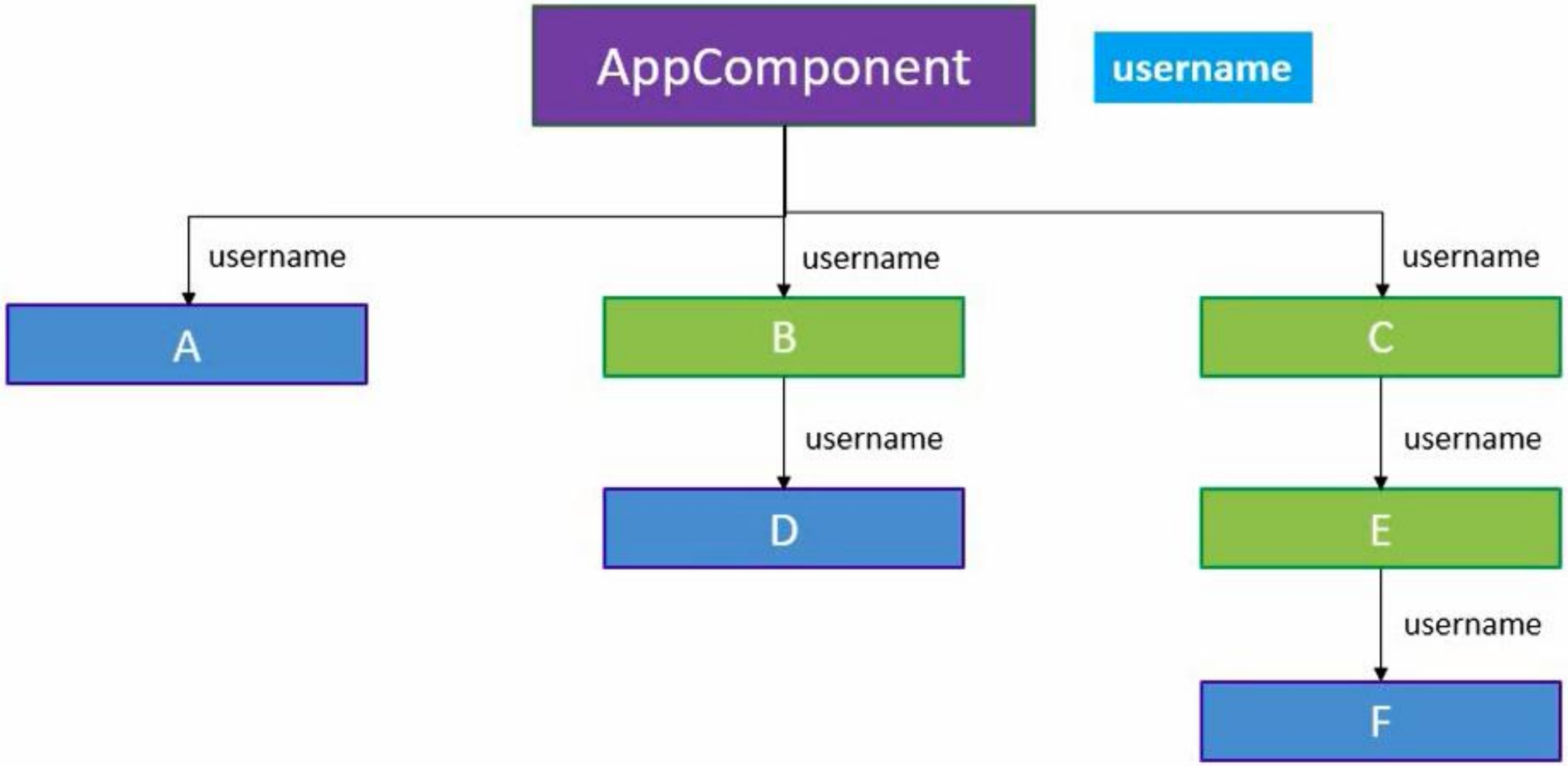
- Provide a way to render children components into a DOM node that is present outside the DOM hierarchy of the parent components

Context

Share the data to any hierarchical component without passing the value as prop to the component

1. Create the user context using createContext method from React
2. Create the Provider component and provide the value to be consumed across components
3. Use the Consumer component and pass in a function as its child, function receives the context value as a parameter

Context


AppComponent
username
username
username
username
A
B
C
username
username
D
E
username
F

Routing

- Npm install react-router-dom

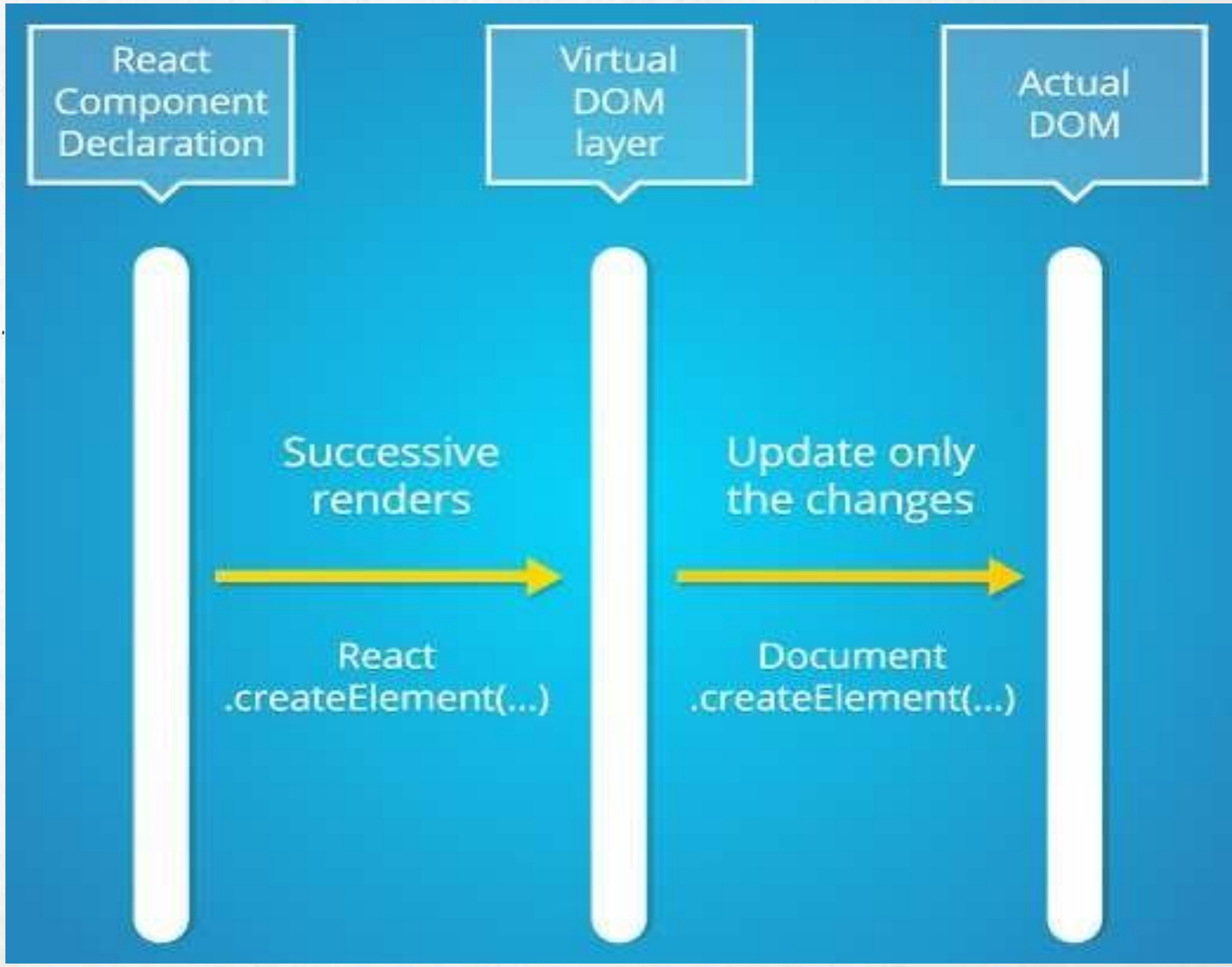
- Router
- Switch
- Route
- Link

UseEffect Hook

- The useEffect hook lets you perform side effects in functional component
- It is a close replacement for componentDidMount, componentDidUpdate, componentWillUnmount in functional component
- Call lifecycle methods after render
- Conditionally run use effects
- We can have multiple UseEffects in a functional component

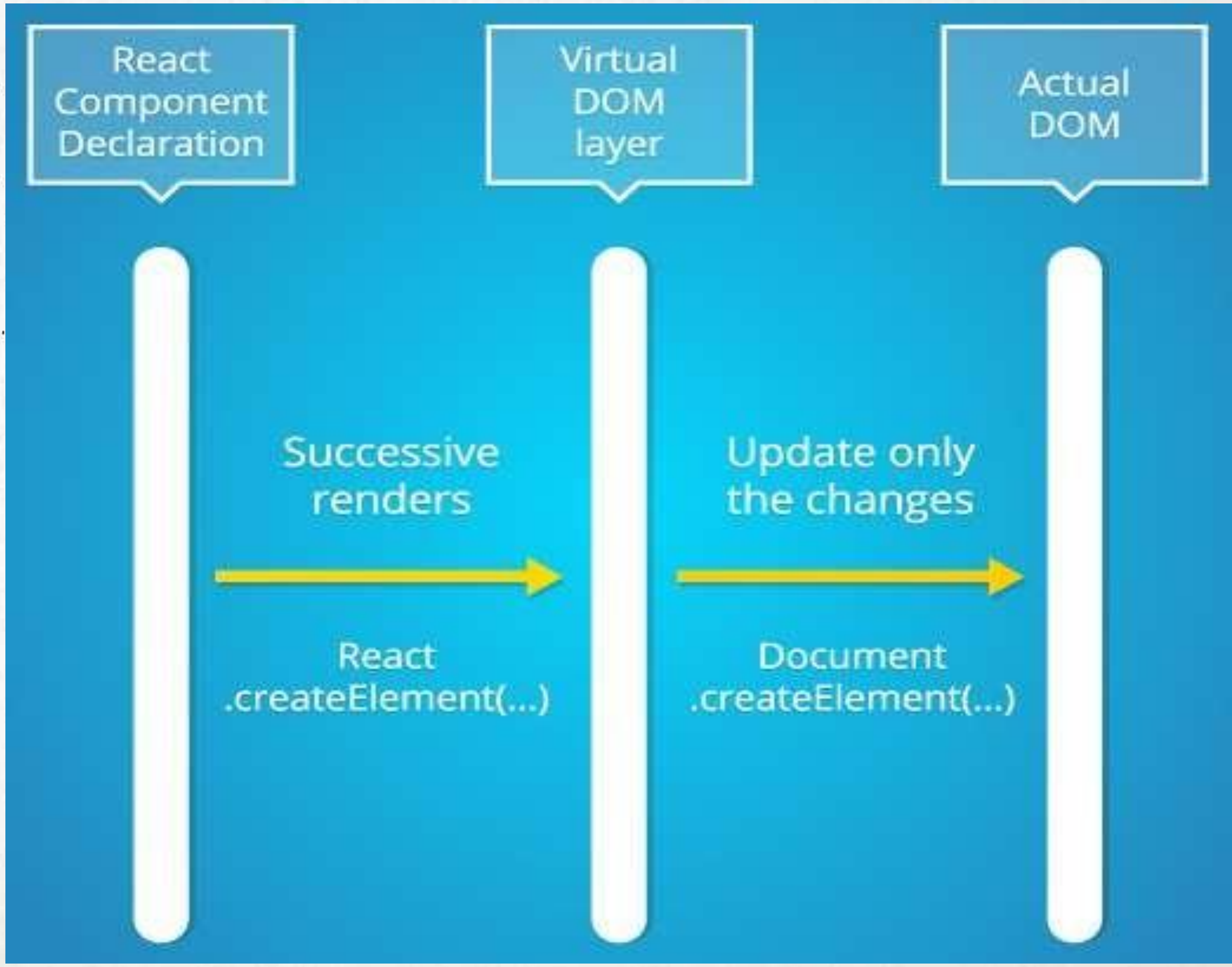
WHAT IS REACT? – VIRTUAL DOM

- Manipulate DOM is high cost.
- React first assembles the *entire* structure of your app **in-memory**, using those objects. Then, it **converts** that structure into **actual DOM nodes** and inserts them in your browser's DOM.


React Component Declaration
Virtual DOM layer
Actual DOM
Successive renders
React .createElement(...)
Update only the changes
Document .createElement(...)

WHAT IS REACT? – VIRTUAL DOM

```
<script>
 var helloEl = React.createElement(
  'div',
  { className: 'hello' },
  'Hello, world!`
 );
 React.render(helloEl,document.body);
</script>
```


React Component Declaration
Virtual DOM layer
Actual DOM
Successive renders
React .createElement(...)
Update only the changes
Document .createElement(...)

PROS & COS OF REACT.JS

THE GOOD POINTS:

- **React.js is extremely efficient**
  - Virtual DOM
- **It makes writing Javascript easier**
  - React.js uses a special syntax called JSX
- **It gives you out-of-the-box developer tools**
  - React.js chrome extension
- **It's awesome for SEO**
  - Server rendering
- **UI Test Cases**

THE BAD:

- React.js is only a view layer.
- There is a learning curve for beginners who are new to web development.
- Library size. (~ Angular)

Why you should use React.js:

- React.js works great for teams, strongly enforcing UI and workflow patterns.
- The user interface code is readable and maintainable.
- Componentized UI is the future of web development, and you need to start doing it now.
- And also, there is now a lot of demand for developers with ReactJS experience.

Why you should NOT use React.js:

- **Slow you down tremendously** at the start.
- You will reinvent a lot of wheels.

Appendix

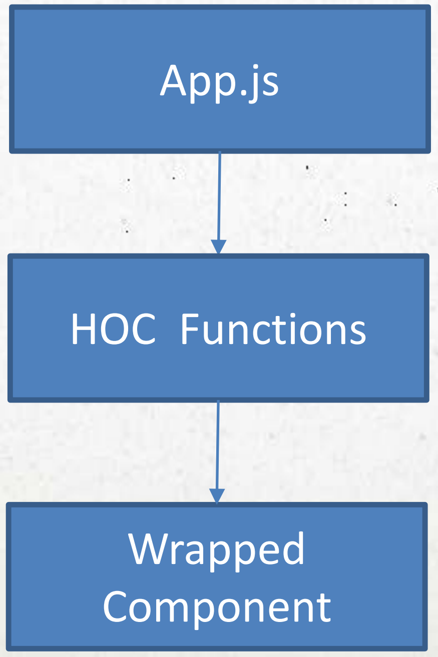
Refs – Need to access Dom Node

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

<input ref = {this.inputRef}></input>

CdM(){
   this.inputRef.current.focus()
}
```

- Refs helps to access DOM Node through React
- Forwarding of refs to child component is possible

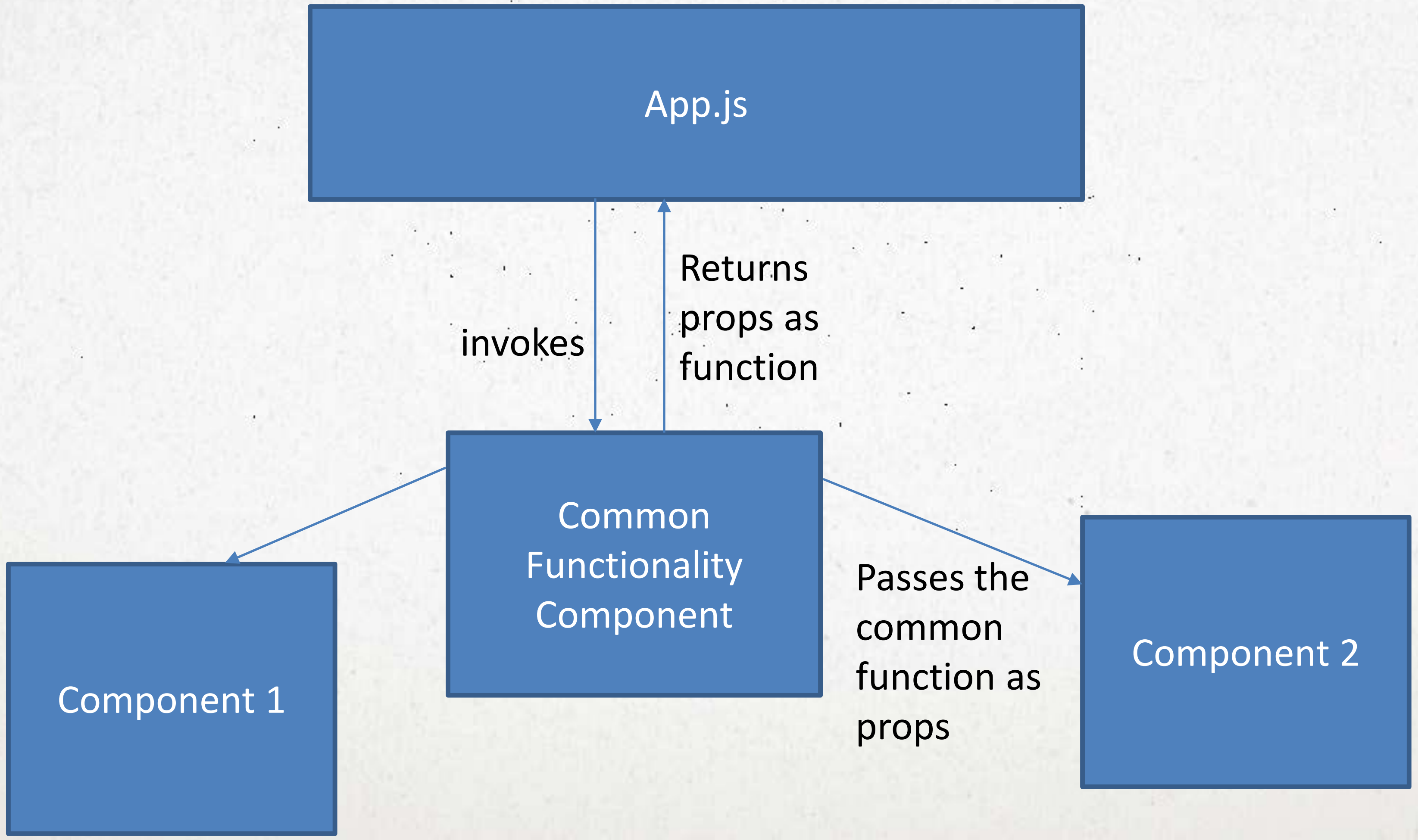
Higher order component - HOC

- HOC is required to share the common functionality between components
- A pattern where a function will take a component as an argument and returns a new component with common shared functionality
- Const NewComponent = higherOrderComponent(originalComponent)



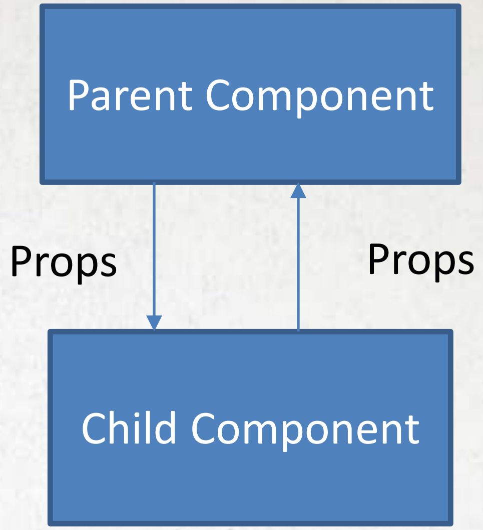
Render Props

- Render props is a technique for sharing code between React components using a prop whose value is a function


App.js
invokes
Returns props as function
Common Functionality Component
Component 1
Passes the common function as props
Component 2

Pass data from Child Component to Parent Component

- Call the function as a prop



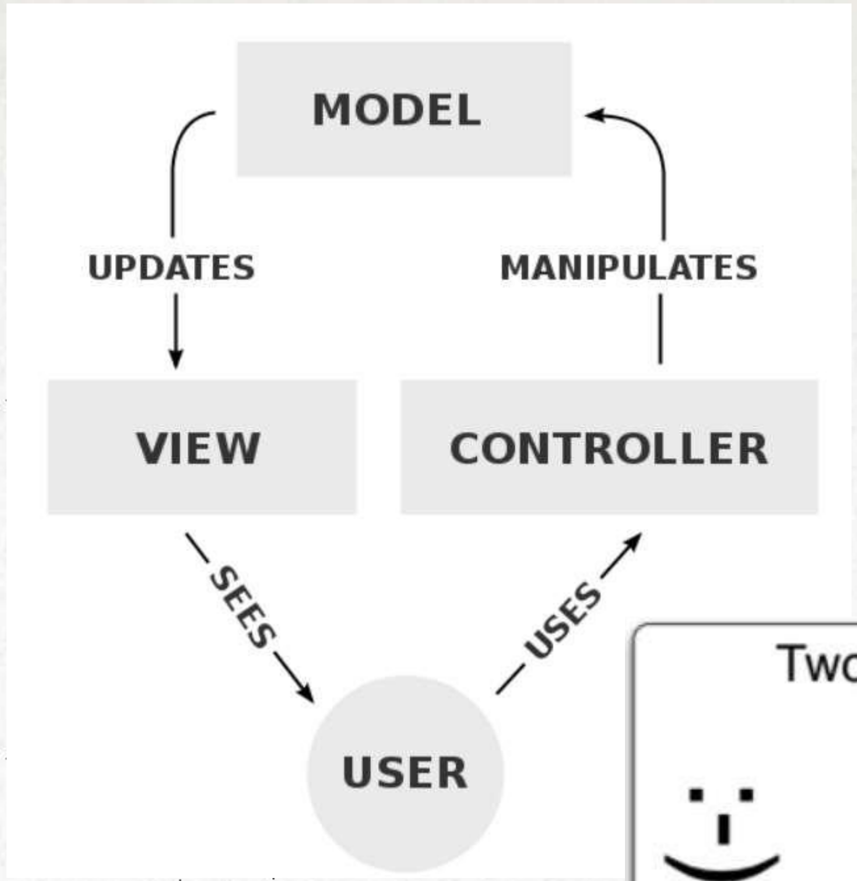
Event Binding

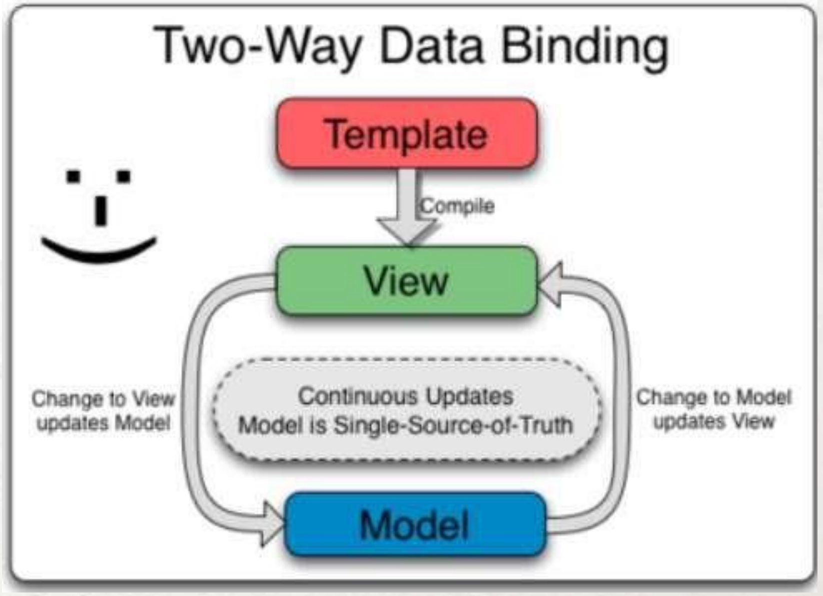
1. Bind the function in render function
2. Bind the event handler in the class constructor – Will use this mostly
3. Class property as arrow functions (Still an Experimental feature)

WHAT IS REACT?

#2 Single Source of Truth

MVC proposes that your **Model is the single source of truth**— all state lives there. Views are **derived** from the Model, and must be **kept in sync**. When the Model changes, so does the View.

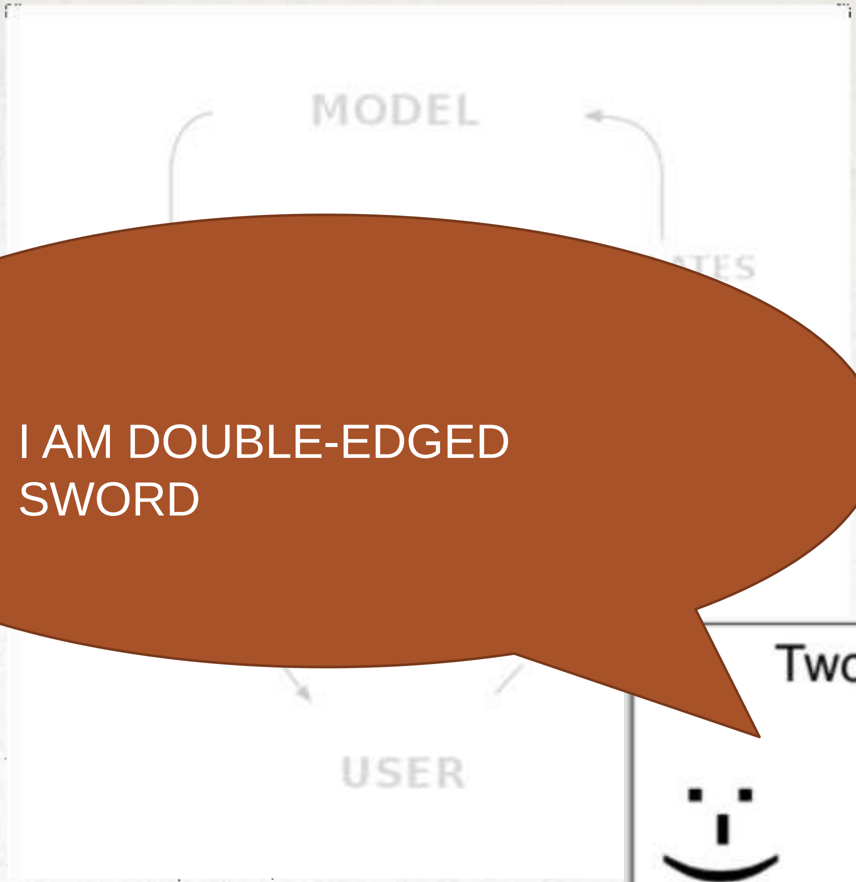



Two-Way Data Binding
Template
Compile
View
Continuous Updates
Model is Single-Source-of-Truth
Change to View updates Model
Change to Model updates View
Model

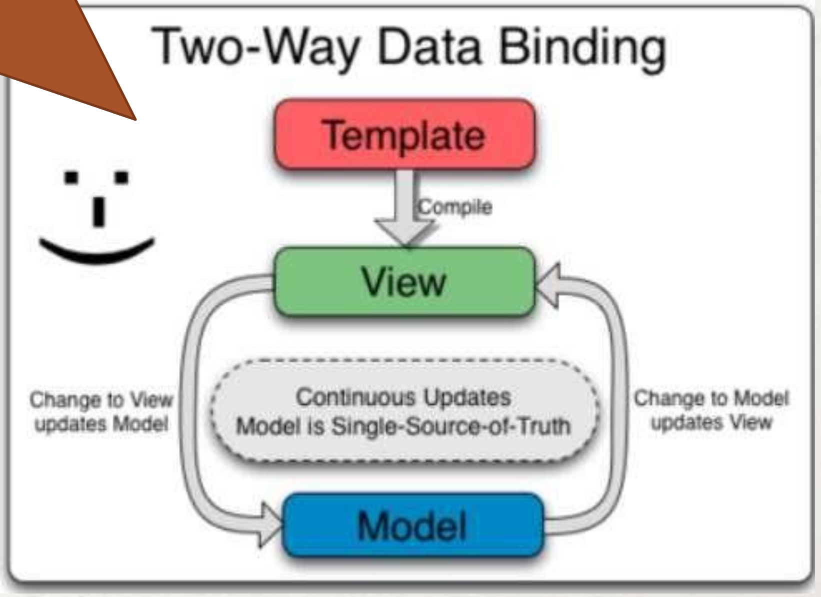
WHAT IS REACT?

#2 Single Source of Tru

MVC proposes that
Model is the single sou
of truth— all state lives there.
Views
are **derived** from the Model,
and must be **kept in sync**.
When the Model changes,
so does the View.



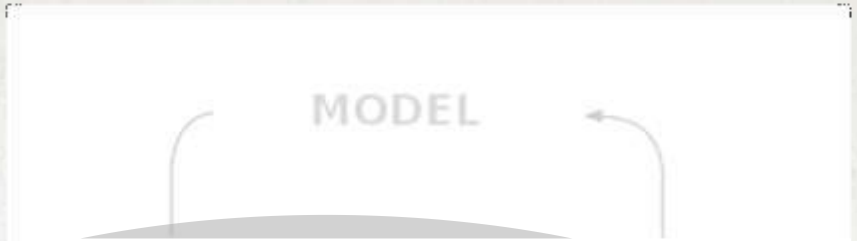
I AM DOUBLE-EDGED SWORD

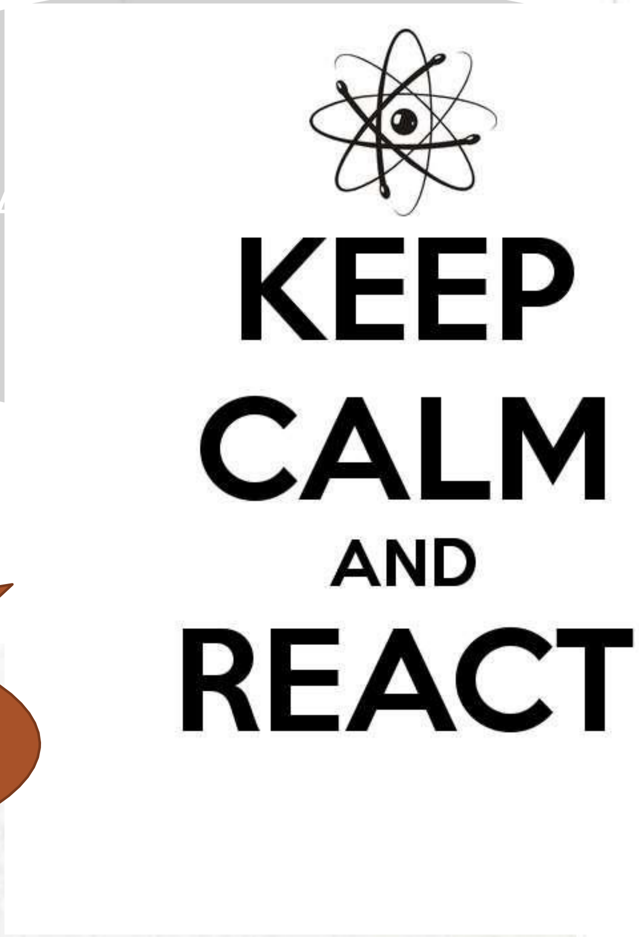

Two-Way Data Binding
Template
Compile
View
Continuous Updates
Model is Single-Source-of-Truth
Change to View updates Model
Change to Model updates View
Model

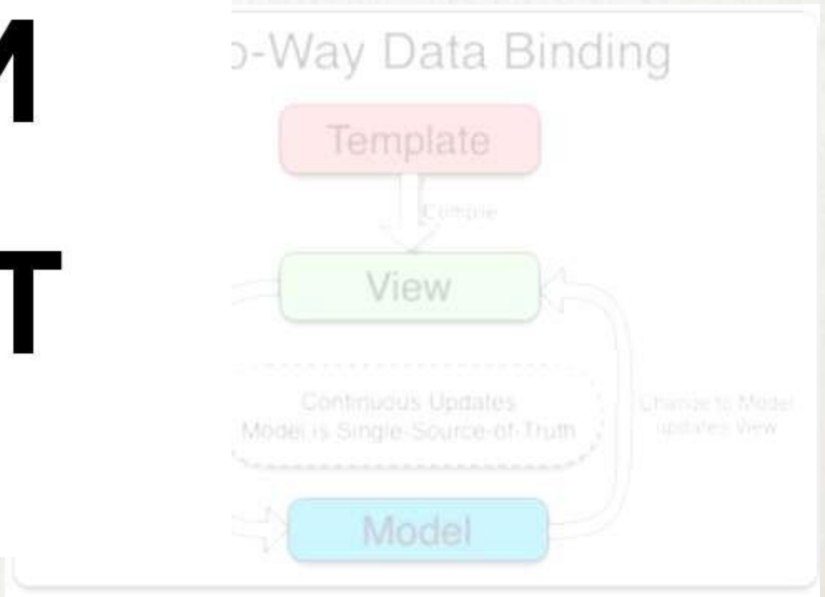
WHAT IS REACT?

#2 Single Source of Truth

MVC proposes that your **Model is the single source of truth**— all state lives there. Views are **derived** from the Model, and must be **kept in sync**. When the Model changes







Only render when state changed

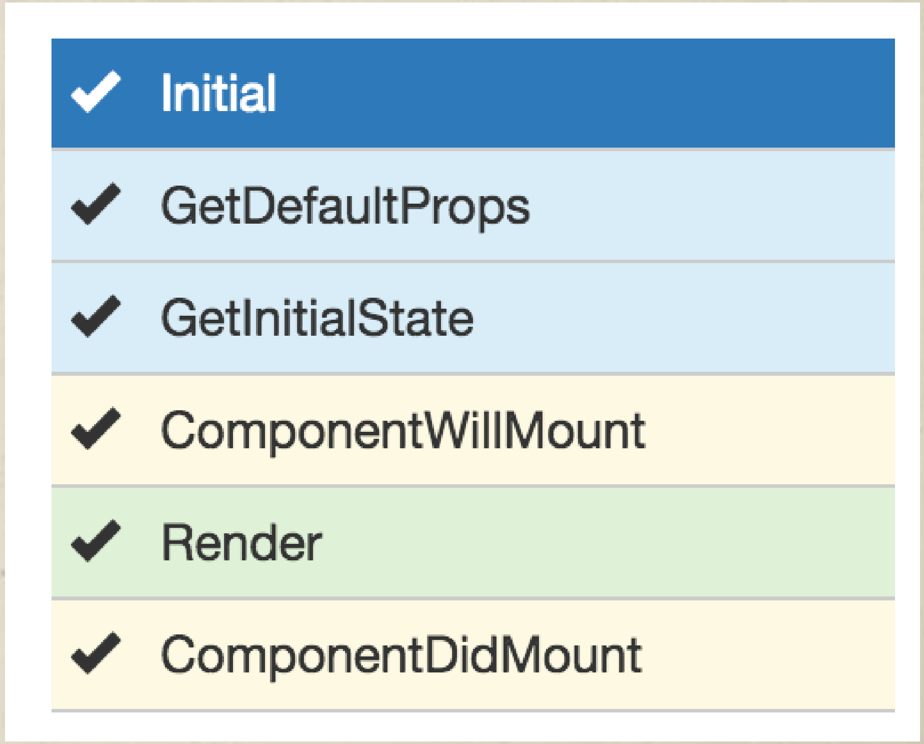
REACT COMPONENT LIFECYCLE

- **React** enables to create components by invoking the React.createClass() method which expects a *render* method and triggers a lifecycle that can be hooked into via a number of so called lifecycle methods.
- This short article should shed light into all the applicable functions.
- Understanding the component lifecycle will enable you to perform certain actions when a component is created or destroyed. Further more it gives you the opportunity to decide if a component should be updated in the first place and to react to props or state changes accordingly.

THE LIFECYCLE - INITIALIZATION

- 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'
    }
  }

  // ...
});
```



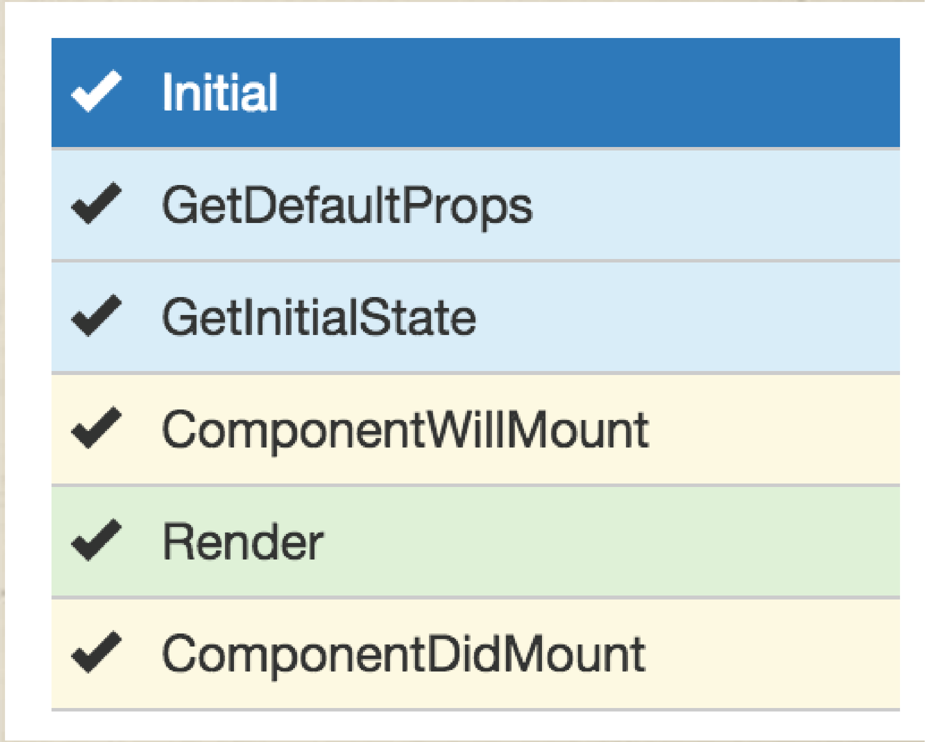
- ✓ Initial
- ✓ GetDefaultProps
- ✓ GetInitialState
- ✓ ComponentWillMount
- ✓ Render
- ✓ ComponentDidMount

THE LIFECYCLE - INITIALIZATION

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

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

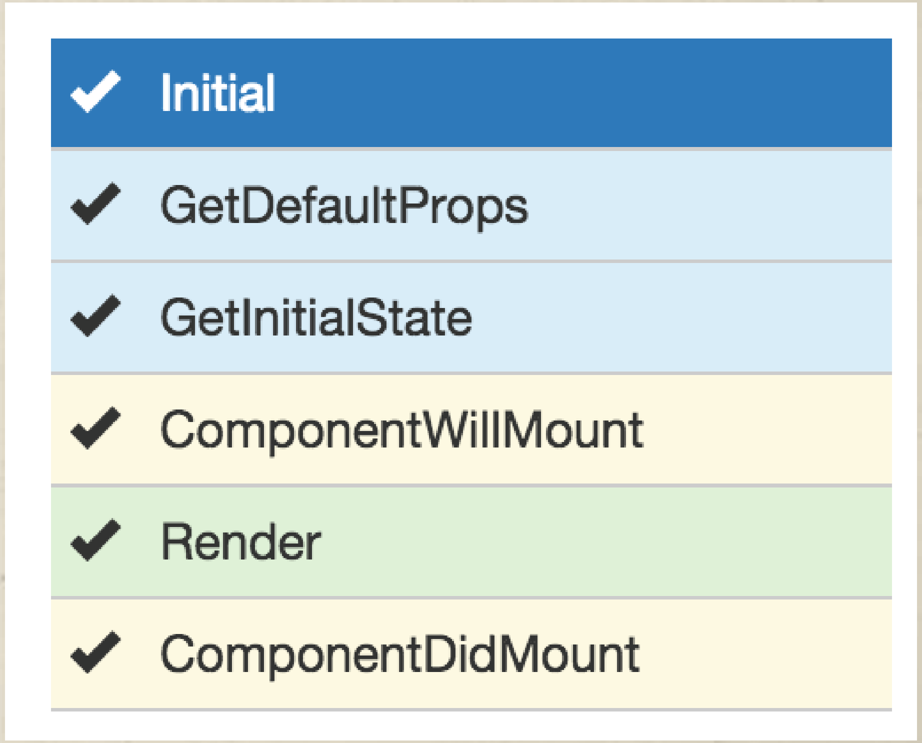
```
constructor(props){
  super(props);
  this.state = {
    name: 'Mary'
  }
}
```



✓ Initial
✓ GetDefaultProps
✓ GetInitialState
✓ ComponentWillMount
✓ Render
✓ ComponentDidMount

THE LIFECYCLE - INITIALIZATION

- Inside ComponentWillMount, setting state won’t trigger re-render whole component.
- We CAN NOT modify state in render method.
- DOM Manipulation is only permitted inside componentDidMount method.



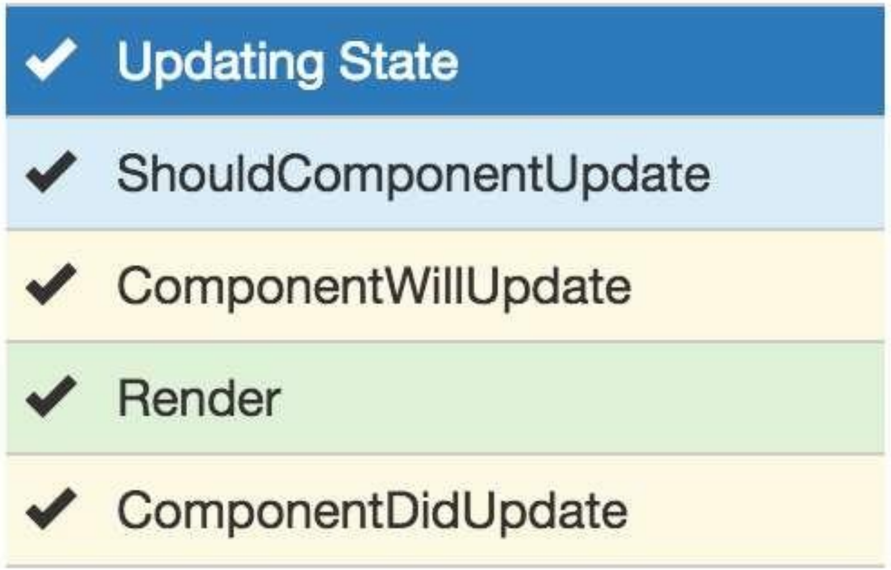
✓	Initial
✓	GetDefaultProps
✓	GetInitialState
✓	ComponentWillMount
✓	Render
✓	ComponentDidMount

THE LIFECYCLE - STATE

- 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,



✔ Updating State

✔ ShouldComponentUpdate

✔ ComponentWillUpdate

✔ Render

✔ ComponentDidUpdate

THE LIFECYCLE - PROPS CHANGES

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

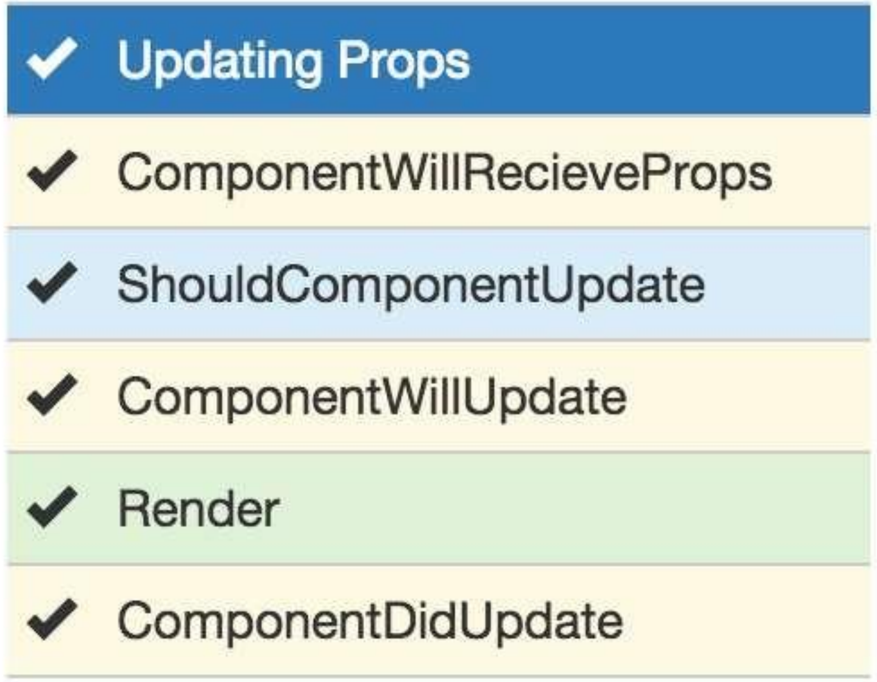
Props Change ➔ componentWillReceiveProps trigged

NOT

Props Change ⬄ componentWillReceiveProps

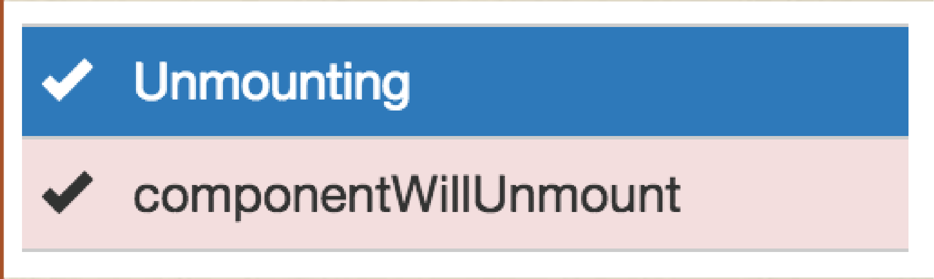
- Changing states in *ComponentWillReceiveProps* DID
NOT trigger re-render component.

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



✓ Updating Props
✓ ComponentWillRecieveProps
✓ ShouldComponentUpdate
✓ ComponentWillUpdate
✓ Render
✓ ComponentDidUpdate

THE LIFECYCLE - UNMOUNTING


✓ Unmounting

✓ componentWillUnmount

- Used to clean up data

Form Validation

REFERENCES

- https://firstdoit.com/how-i-learned-to-stop-worrying-and-love-react-4e22b0bb6c2a#.vt2mjxu6s
- https://offroadcode.com/journal/news/reactjs-whats-it-all-about/
- http://sixrevisions.com/javascript/why-i-ditched-angular-for-react/
- https://github.com/hpphat92/my-todo-reactjs
- https://hpphat.wordpress.com/category/web-dev/react/
- http://blog.andrewray.me/reactjs-for-stupid-people/
- \\192.168.1.240\share\Phat.Hong\Reactjs