React what??

Functional Reactive Programming (FRP)

1. Functions at the heart of code
2. Data mutation is Bad (should be avoided)

PREDICTABILITY

1. Composability → The system is seen as a composition of functions
2. Purity → functions which are free of side-effects and don't mutate their own arguments
3. Higher Order functions → Have a function as an argument and/or return another function
4. Immutability → functions always return a new object / array or function derived from the arguments, but the arguments remain the same

FRP - Examples

```
function sumArray(values) {
    return values
        .map(Number)
        .reduce((total, value) => total + value);
}

sumArray([1, '2', '30']); // 33
```

```
function push(array, element) {
    // Add new element to array
    array.push(element);
    return element;
}

function concat(array, elements) {
    // Create new array with all values
    return array.concat(elements);
}
```

```
let arr = [1, 2, 3];
let addMore = true;

// Adding elements:
arr.push(4, 5);
if (addMore) {
    arr.push(6, 7);
}

// Or the functional way
arr = arr
    .concat([4, 5])
    .concat(addMore ? [6, 7] : []);
```

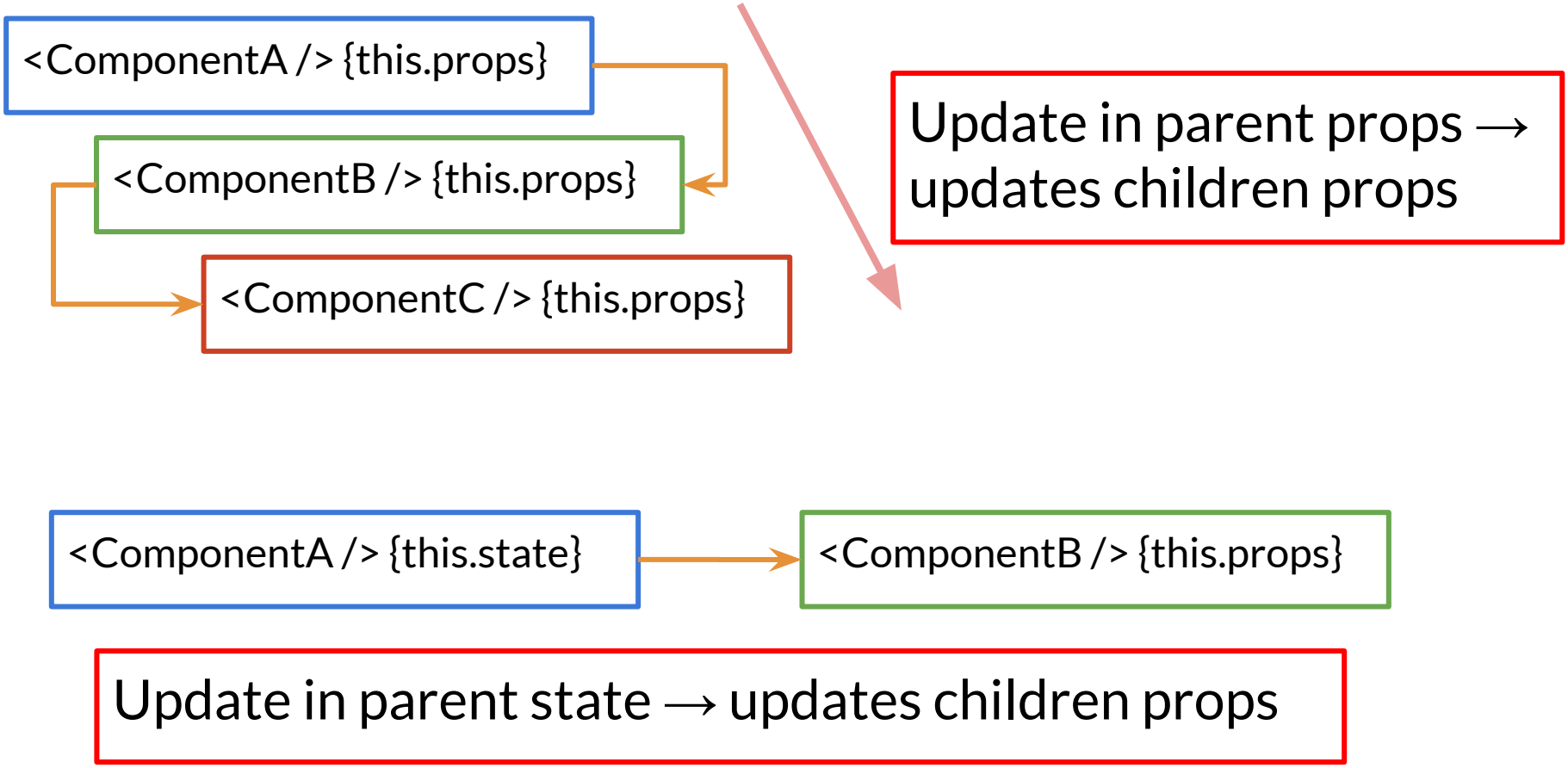
ReactJS and FRP

ReactJS fits the FRP Paradigm:

1. UI is **predictable** → **ReactDOM.render();**
2. UI is **composable** → **<ParentComonent><ChildComponent /></ParentComponent>**
3. Higher-order components give us more code **reusability** and increase *composability*.
    a. A component taking a component as an argument and wrapping aditional content around it to provide some context. This would be the case of a modal component.
    b. A component returning another component (i.e. a factory). This would be the case if you want to reuse the functionality of a base component into a new component.

And what about Purity and Immutability? … AMEN !

Reactivity: One Way Binding



Virtual DOM

THE PROBLEM:

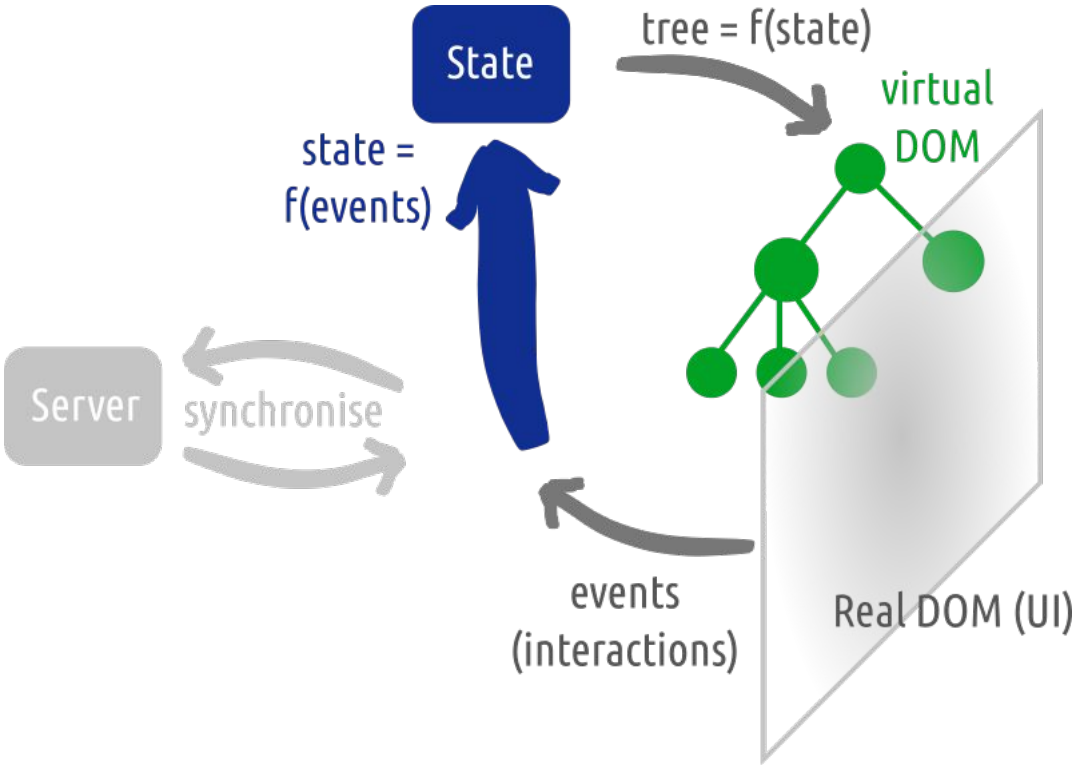
When we re-render a component, it will always return a new instance of a DOM element. The old element would then be removed from the tree and the new one would be added. With no existing mechanisms for the DOM to know if those two elements are related, we would have to re-render it completely.

THE SOLUTION:

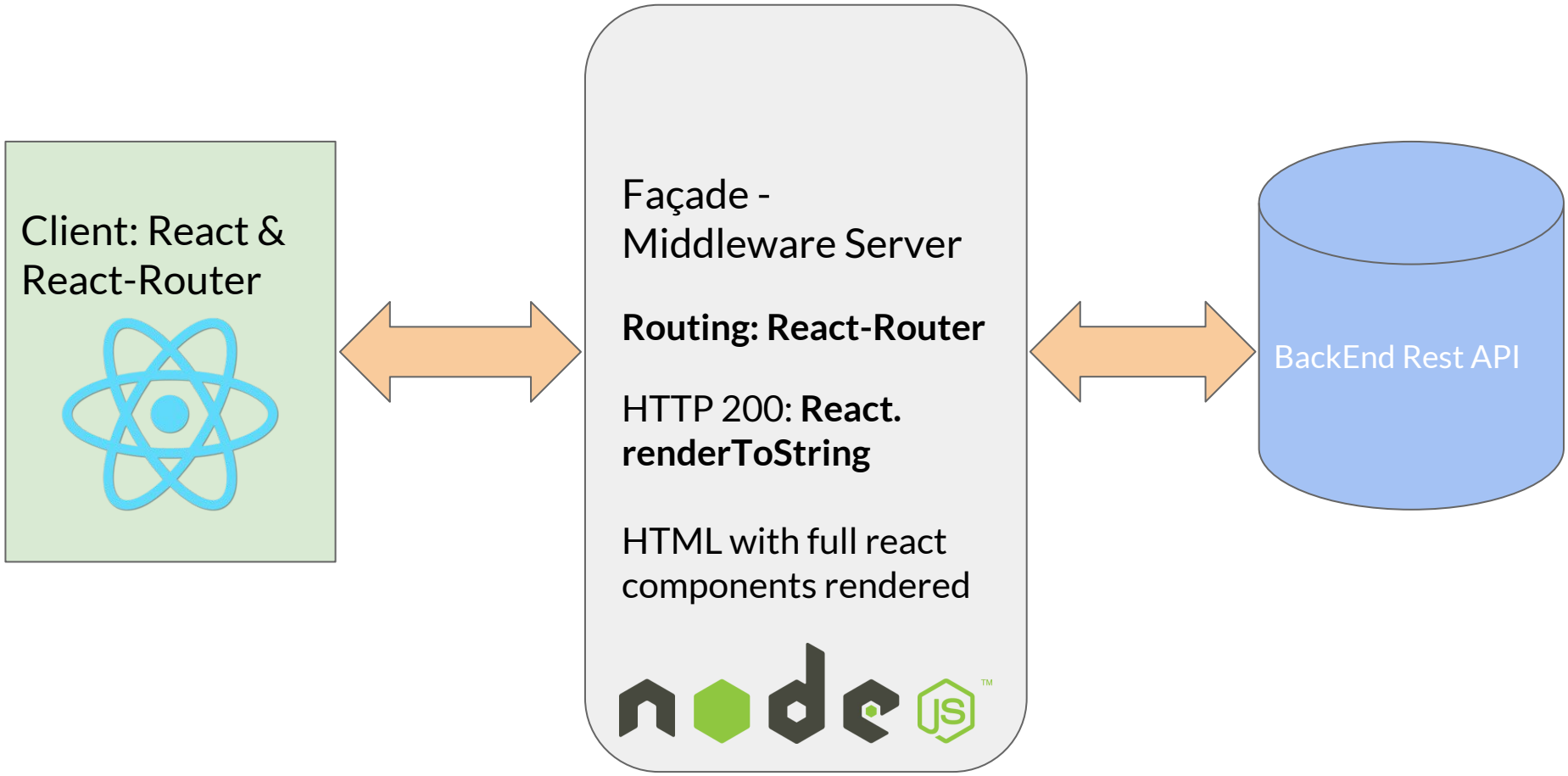
A component is a function creating virtual DOM elements instead of real DOM elements. When a re-rendering occurs, the virtual DOM runs a **diff algorithm** to create a set of changes to apply to the real DOM (a *patch*)

Application Flow: The Flux Pattern

1. An application has a state
2. This state is rendered in a tree of components, using a Virtual DOM. Data is flowing down the tree.
3. The rendered UI will generate events from user interactions which will update the application state.
4. Following the state update, the UI will be updated
5. The application state can also receive updates from the server



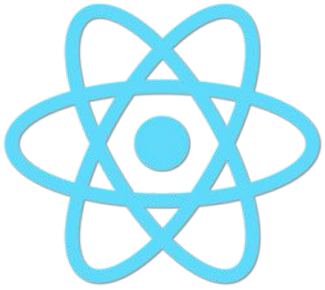
Server Side Rendering



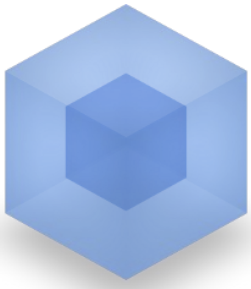
Who is using ReactJS?

9flats - Airbnb - Alipay - Atlassian - BBC - Box - Capital One - Coursera -Dailymotion - Deezer - Docker - Expedia - Facebook - Fotocasa - HappyFresh - IMDb - Instacart - Instagram - Khan Academy - Klarna - Lyft - NBC - Netflix - NFL - Paypal - Periscope - Ralph Lauren - Reddit - Salesforce - Stack Overflow - Tesla - Tmall - The New York Times - Twitter Fabric - Twitter Mobile - Uber - WhatsApp - Wired - Yahoo - Zendesk

Technology Stack



REDUX



webpack
MODULE BUNDLER

REACT/ROUTER

node JS™

JSX

JSX

JSX is a JavaScript syntax extension that looks similar to XML. You can use a simple JSX syntactic transform with React.
JSX is optional, however it is recommended because it is a concise and familiar syntax for defining tree structures with attributes.

```
<div> texto </div>

<div className=”rojo”> texto </div>

<Menu className=”navbar” />

<UsersTable>
    <User name=”Tyrion” />
</UsersTable>
```

React-DOM

The react-dom is a package provides DOM-specific methods that can be used at the top level of your app and as an escape hatch to get outside of the React model if you need to.
Method render: Render a ReactElement into the DOM in the supplied container and return a reference to the component.

```
import ReactDOM from 'react-dom';

ReactDOM.render(
    <App families={[“Stark”, “Lannister”, “Targaryen”,
“Baratheon” ]} />,
    document.querySelector('.container'),
    () => { console.log("I'm a callback, you know nothing");
);
```

Events & Callbacks

```
class Lannister extends React.Component{
    constructor(props) {
        super(props);
        this.state = { motto: "Hear me roar" };
        this.update = this.update.bind(this);
    }
    update() {  this.setState( { motto: "A lannister always pays his debts" } )  }
    render() {
        return (
            <div>
                <h3>Lannister</h3>
                name: {this.props.name} <br/>
                motto: {this.state.motto} <br/>
                <ButtonChange update={this.update} />
            </div>
         );
    }
}

const ButtonChange = ({update}) => {
    return ( <button onClick={update} > change </button> )
}

ReactDOM.render( <Lannister name="Tyrion" />, document.getElementById('container') );
```

Component State

Props vs State

Props:

are a Component's configuration, its options. They are received from above and immutable as far as the Component receiving them is concerned.
A Component cannot change its props, but it is responsible for putting together the props of its child Components.

Only Props ? functional components

State:

The state starts with a default value when a Component mounts and then suffers from mutations in time (mostly generated from user events).
A Component manages its own state internally, but—besides setting an initial state—has no business fiddling with the state of its children. You could say the state is private.

If you need to use state or function hooks, you need class based component

Props vs State

Props:

are a Com

They are r

as far as t

concerned

A Compor

responsibl

child Com

Only Prop

State:

e when a

rs from

d from user

te internally,

—has no

s children.

on hooks,

t

-	*props*	*state*
Can get initial value from parent Component?	Yes	Yes
Can be changed by parent Component?	Yes	No
Can set default values inside Component?*	Yes	Yes
Can change inside Component?	No	Yes
Can set initial value for child Components?	Yes	Yes
Can change in child Components?	Yes	No

Controlled components

Form components such as <*input*>, <*textarea*>, and <*option*> differ from other native components because they can be mutated via user interactions.
A controlled <input> has a value prop. Rendering a controlled <input> will reflect the value of the value prop.

```
handleChange: function(event) {
  this.setState({value: event.target.value});
},

render: function() {
  return (
    <input type="text" value={this.state.value}
      onChange={this.handleChange}
      defaultValue=”Fire and Blood”
    />
  );
```

A controlled <input> has a value prop. Rendering a controlled <input> will reflect the value of the value prop.
User input will have no effect on the rendered element because React has declared the value to be *Fire and Blood*.
To update the value in response to user input, you could use the onChange event

Components LifeCycle

Hook Name	Lifecycle Phase	setState	Client	Server
componentWillMount	Mounting	✔	✔	✔
componentDidMount	Mounting	☐	✔	☐
componentWillReceiveProps	Updating	✔	✔	✔
shouldComponentUpdate	Updating	☐	✔	✔
componentWillUpdate	Updating	☐	✔	✔
componentDidUpdate	Updating	☐	✔	☐
componentWillUnmount	Unmounting	☐	✔	☐