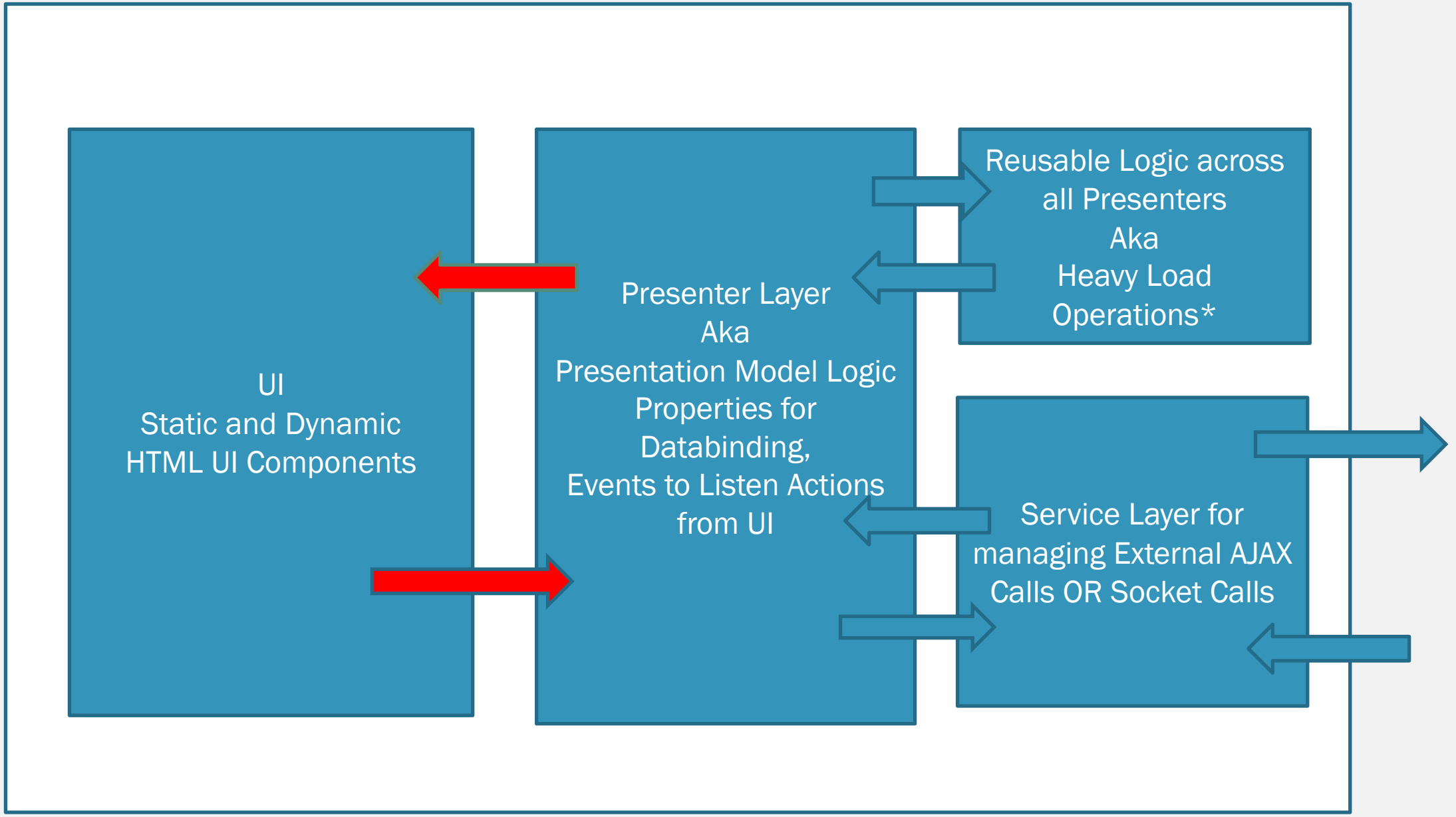
CONSUMER ORIENTED APPLICATIONS

- Apps designed for
  - All devices
  - All Browsers
  - Partner App Integration
  - Consumer App Integration
  - Client-Side Processing Capabilities
  - Security

Front-End Application



BROWSER

HTML Static DOM Parser

The Static HTML Rendering

Dynamic JavaScript Parser JSOM

1. Objects
2. Events
3. Functions
4. Types

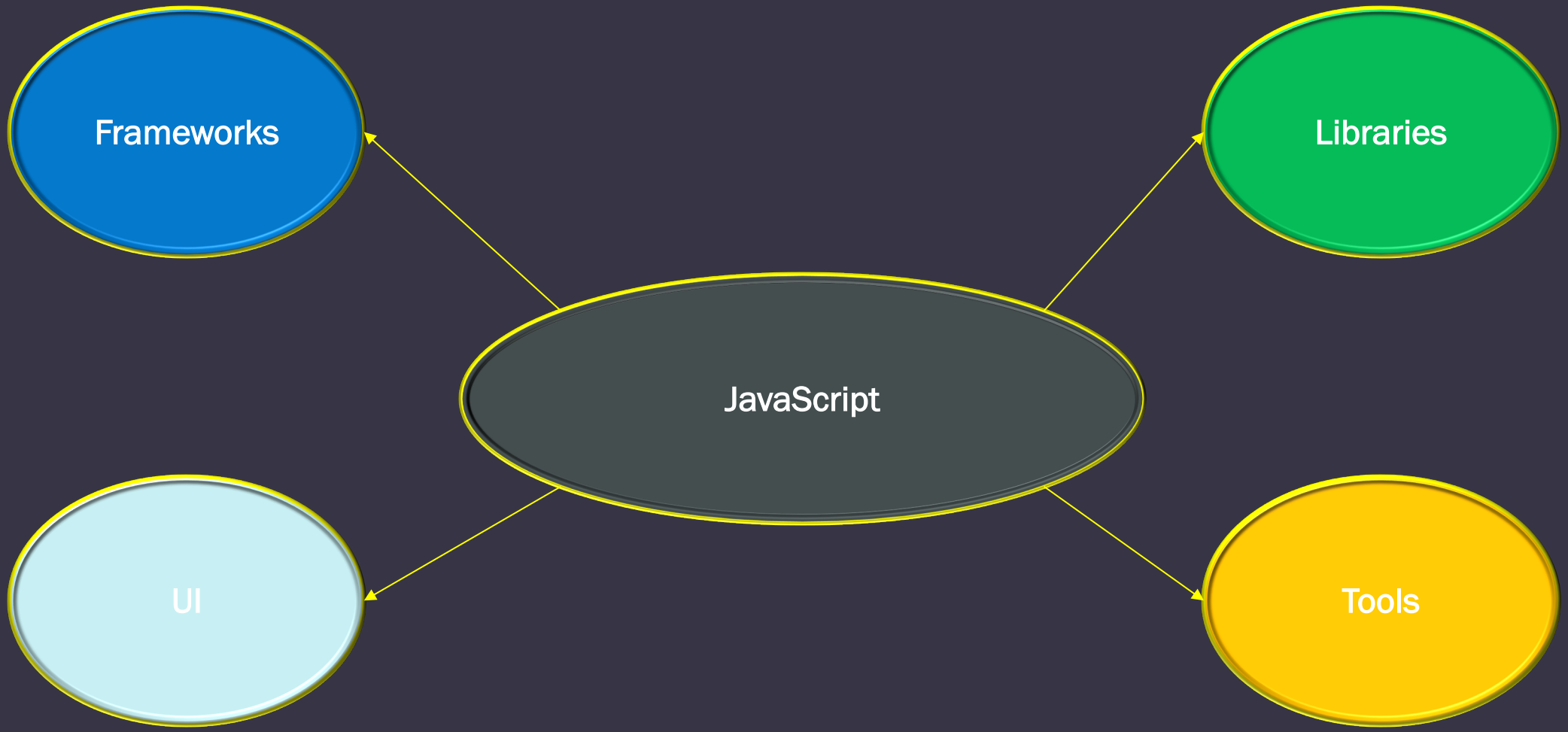
HTML 5 API Systems

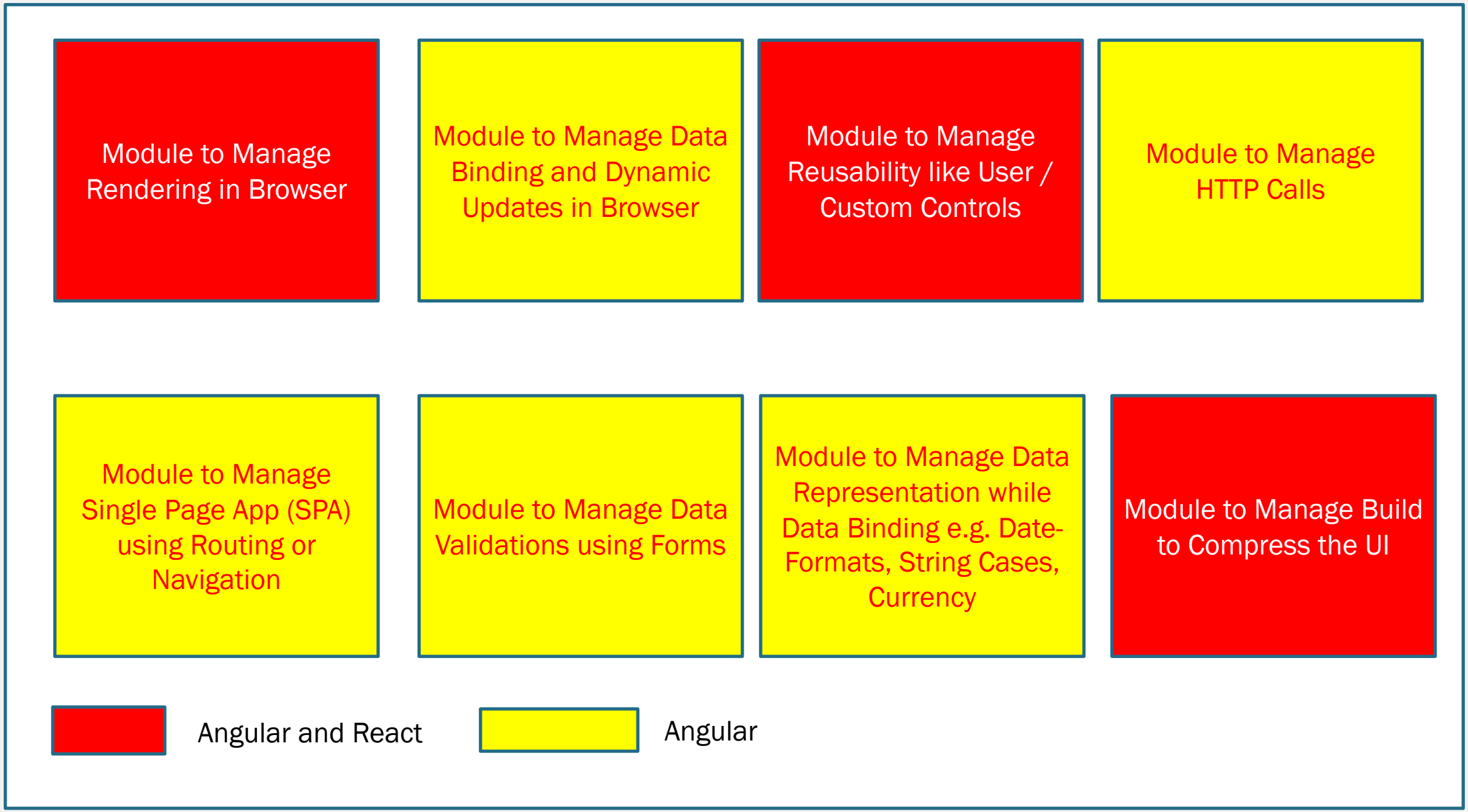
1. DragDrop
2. Worker
3. Files
4. Geolocation
5. Sensors
6. Canvas and SVG
7. Web Worker
8. Media

Networking

1. Sockets
2. Http

WORLD OF JAVASCRIPT




Module to Manage Rendering in Browser
Module to Manage Data Binding and Dynamic Updates in Browser
Module to Manage Reusability like User / Custom Controls
Module to Manage HTTP Calls
Module to Manage Single Page App (SPA) using Routing or Navigation
Module to Manage Data Validations using Forms
Module to Manage Data Representation while Data Binding e.g. Date-Formats, String Cases, Currency
Module to Manage Build to Compress the UI
Angular and React
Angular

POWER OF JAVASCRIPT

- Improved Client-Side Capabilities
- Availability in the form of
  - Libraries
  - Frameworks
  - Components
- More Expectations
  - Isomorphic Applications.
  - Modern Web applications.
- A World of Isomorphic Applications
  - Using JavaScript on a web server as well as the browser reduces the impedance mismatch between the two programming environments which can communicate data structures via JSON that work the same on both sides of the equation. Duplicate form validation code can be shared between server and client, etc.

REACT

- Some FAQs for Front-End Application Development
  - How **mature are the frameworks / libraries**?
  - Are the frameworks likely to **be around for a while**?
  - How **extensive and helpful are their corresponding communities**?
  - How **easy is it to find developers** for each of the frameworks?
  - What are the **basic programming concepts** of the frameworks?
  - How easy is it to use the frameworks **for small or large applications**?
  - What does the **learning curve** look like for each framework?
  - What kind of **performance** can you expect from the frameworks?
  - Where can you have a **closer look under the hood**?
  - How **can you start developing** with the chosen framework?

REACT

- **What is React.js?**
  - A JavaScript library for building user interfaces.
  - It is created by the Facebook for the V of MVC by reusable and interactive UI components.
  - It abstracts away the DOM from developer by giving a simpler programming model and better performance.
  - It can also render on the server using Node.
  - It can power native apps using React Native.
  - React implements one-way reactive data flow which reduces boilerplate and is easier to reason about than traditional data binding.

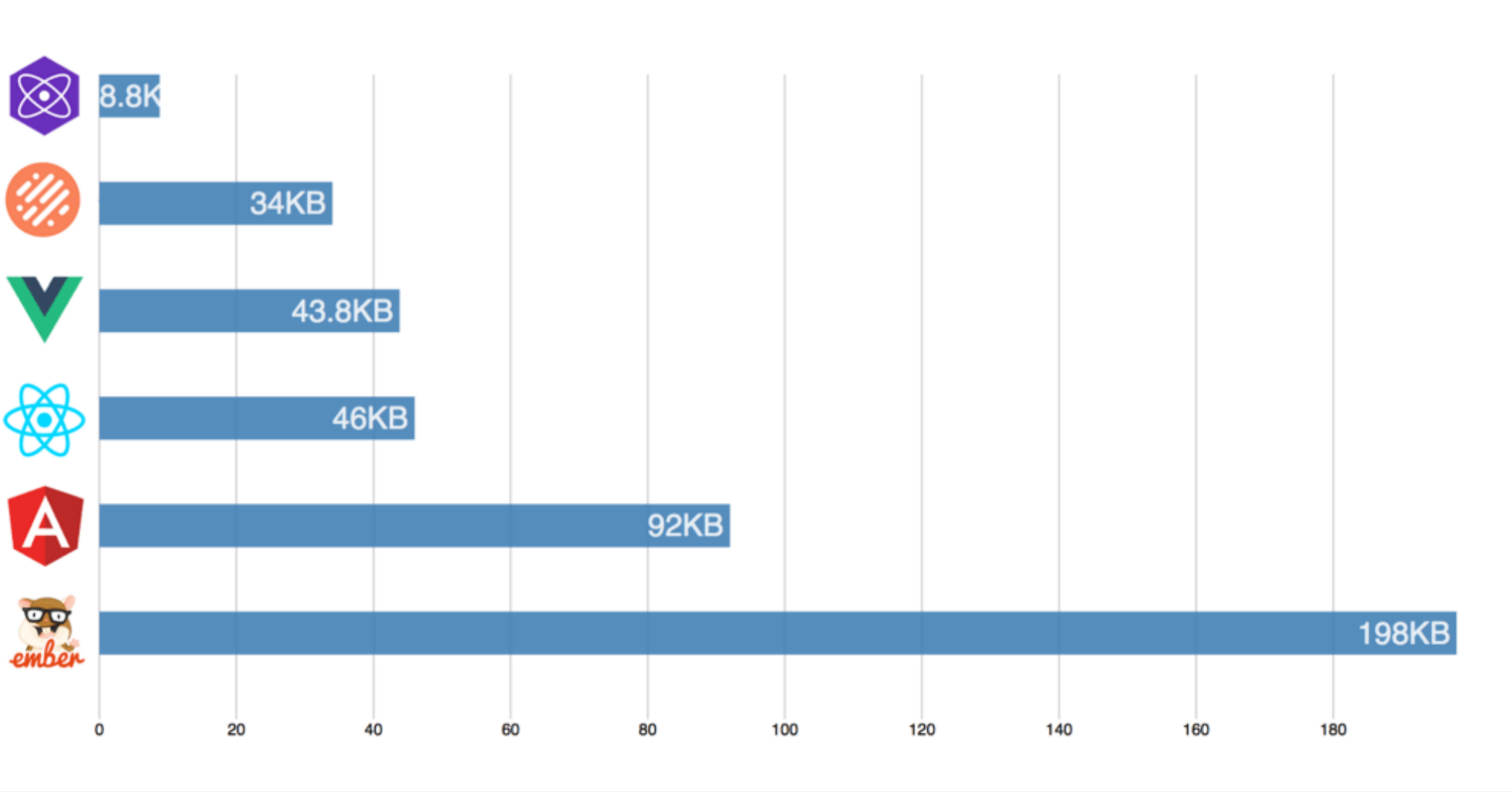
WHY REACT

- V(view) of MVC Solution of View in MVC
- Virtual DOM
- Unidirectional Data Flow
- UI Components
- Coding is simpler because of JSX
- React own debugger

REACT CORE CONCEPTS

JSX	JSX is JavaScript syntax extension. It isn't necessary to use JSX in React development, but it is recommended.
Components	React is all about components. You need to think of everything as a component. This will help you to maintain the code when working on larger scale projects.
Virtual DOM	React uses virtual DOM which is JavaScript object. This will improve apps performance since JavaScript virtual DOM is faster than the regular DOM.
Unidirectional Data Flow	React implements one way data flow which makes it easy to reason about your app.

COMPARISON FRAMEWORK SIZE

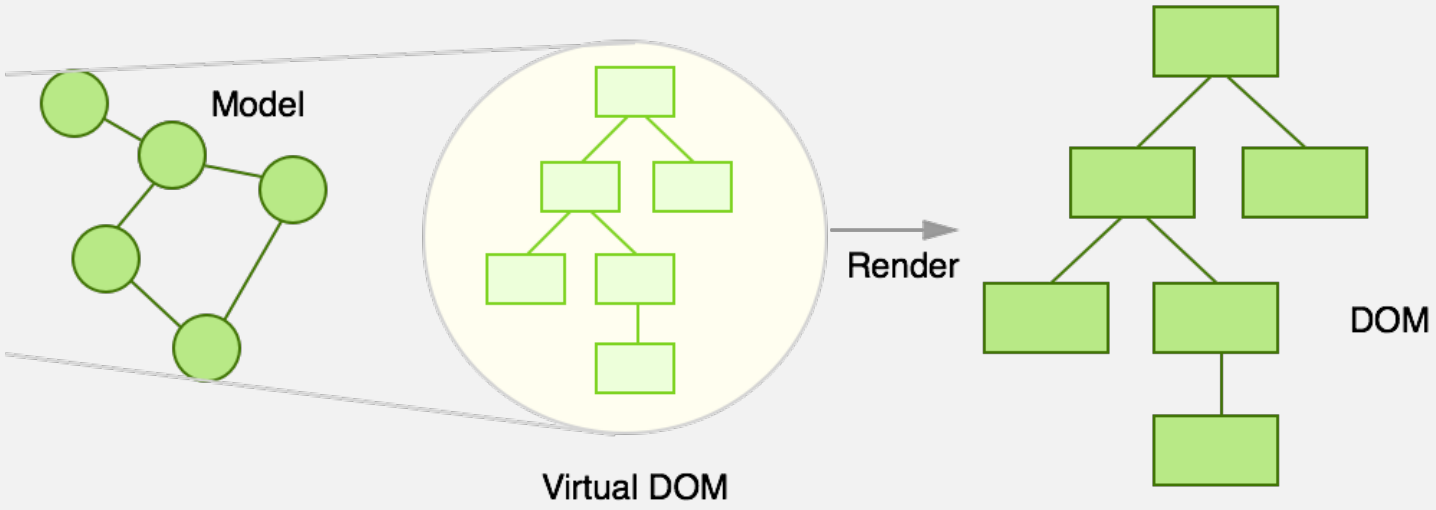


https://programmingwithmosh.com/javascript/react-vs-vue-a-wholesome-comparison/

	React	ANGULAR	Vue.js
	A declarative, efficient, and flexible JavaScript library for building user interfaces.	One framework. Mobile & desktop.	A progressive, incrementally adoptable JavaScript framework for building UI on the web.
	116 993 ⭐	59 302 ⭐	121 050 ⭐
Original author	Jordan Walke	Miško Hevery	Evan You
Developers	Facebook	Google	
Initial release	May 29, 2013	October 20, 2010	February 2014
Npm weekly downloads	3 940 035	433 361	709 943
Size	109.7 KiB production 774.7 KiB development	167 kB production 1.2 MB development	30.67 KB production 279 KB development
Easy to learn	Medium	Learn TypeScript	Yes
Coding speed	Normal	Slow	Fast
Documentation	✓	✓	✓
Performance	✓	✓	✓
Startup time	Quick	Longer due to its large codebase	Quick
Complete web apps	Needs to be integrated with many other tools	Can be used on standalone basis	Requires third party tools
Data binding	Uni-directional	Bi-directionall	Bi-directional
Rendering	Server side	Client side	Server side
Model	Virtual DOM	MVC	Virtual DOM
Code reusability	No, only CSS	Yes	Yes, CSS and HTML
When to use	Production, custom UI apps	Production, esp. enterprise apps with Material UI	Startups, production
Big companies	Facebook, Yahoo, Netflix, Atlassian, KhanAcademy	Netflix, Upwork, PayPal, TheGuardian	Facebook, Alibaba, Adobe, Grammarly, GitLab

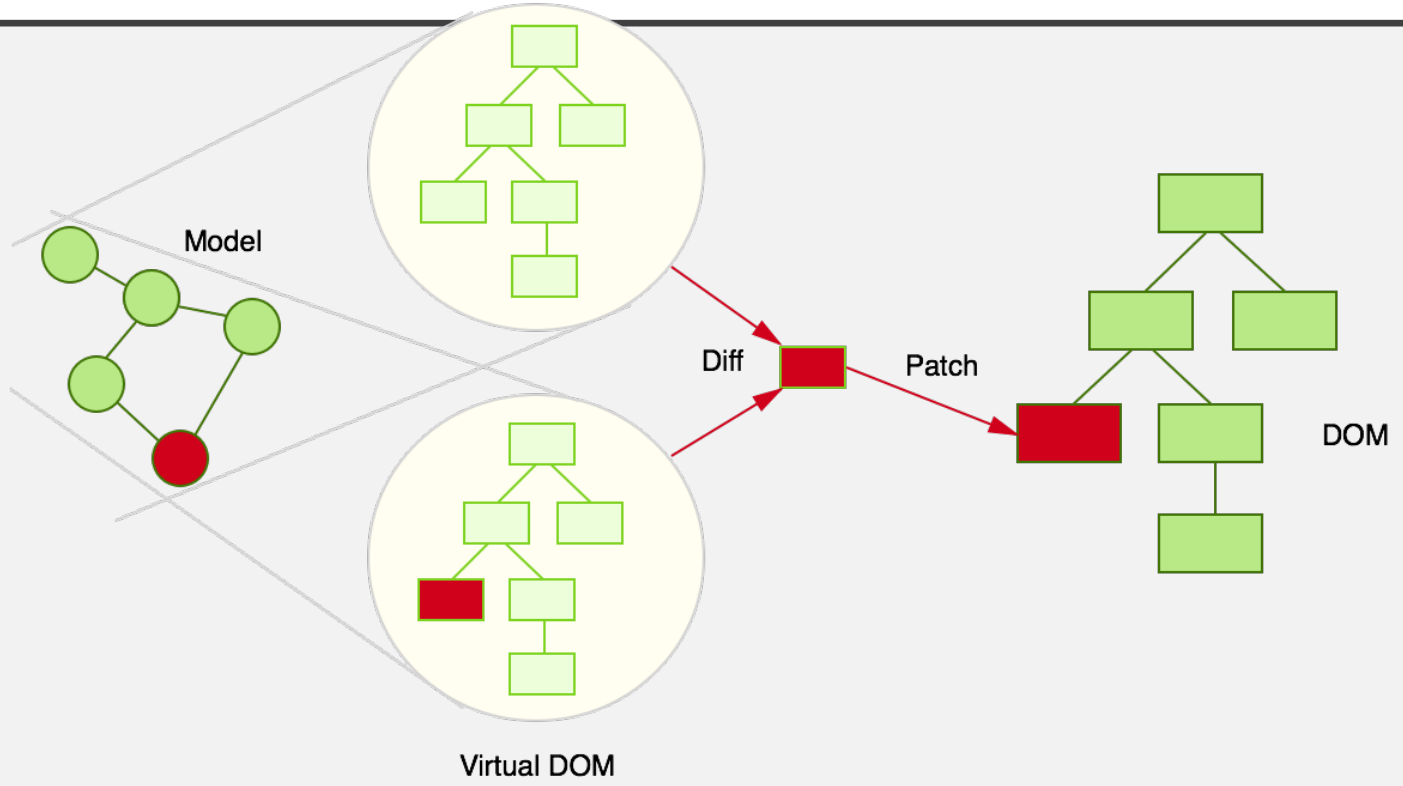
https://dzone.com/articles/infographic-react-vs-angular-vs-vuejs

VIRTUAL DOM



Source: http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

VIRTUAL DOM

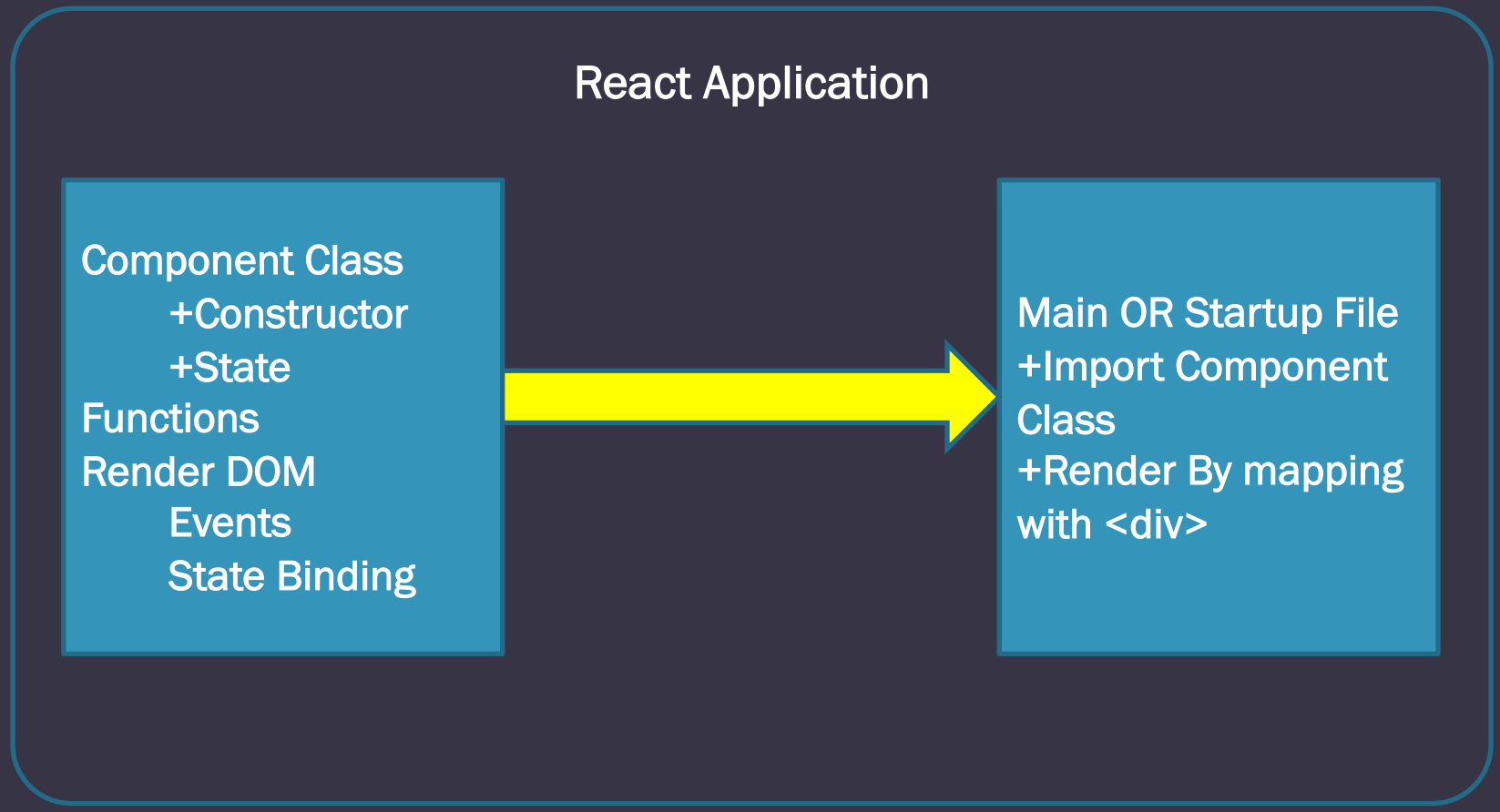


Source: http://teropa.info/blog/2015/03/02/change-and-its-detection-in-javascript-frameworks.html

REACT

- React Limitations
  - React only covers view layer of the app so you still need to choose other technologies to get a complete tooling set for development.
  - React is using inline templating and JSX. This can seem awkward to some developers.

REACT


React Application
Component Class
+Constructor
+State
Functions
Render DOM
Events
State Binding
Main OR Startup File
+Import Component Class
+Render By mapping with

REACT

- Important Modules
  - react
    - Provides React and Component classes.
    - State and Props Objects for data flow.
    - Used to define Component with render function.
    - Binds functions with events of DOM elements.
    - Binds state and props with the DOM elements
  - react-dom
    - Provides ReactDOM class
      - Provides render method to render Component in the container.

SETTING UP REACT APPLICATION

SETTING UP PROJECT

- Package.json → Dependencies
  - "react": "16.9.0",
  - "react-dom": "16.9.0",
  - "redux": "4.0.4",
  - "react-redux": "7.1.0",
  - "redux-saga": "1.0.5“
- devDependencies
- "@babel/cli": "^7.2.0",
  - "@babel/core": "^7.2.0",
  - "@babel/polyfill": "^7.0.0",
  - "@babel/preset-env": "^7.2.0",
  - "@types/react": "^16.7.13",
  - "@types/react-dom": "^16.0.11",
  - "@types/react-redux": "^6.0.10",
  - "awesome-typescript-loader": "^5.2.1",

```
"webpack": "4.40.2",
"webpack-cli": "3.3.9",
"webpack-dev-server": "3.8.1"
```

SETTING UP PROJECT

- Webpack.config.js

```
context: path.join(basePath, "src"),
resolve: {
    extensions: ['.js', '.ts', '.tsx']
},
entry: ['@babel/polyfill',
    './main.tsx'
],
output: {
    path: path.join(basePath, 'dist'),
    filename: 'bundle.js'
},
devtool: 'source-map',
devServer: {
    contentBase: './dist', // Content base
    inline: true, // Enable watch and live reload
    host: 'localhost',
    port: 8088,
    stats: 'errors-only'
},
```

```
module: {
    rules: [{
            test: /\.(ts|tsx)$/,
            exclude: /node_modules/,
            loader: 'awesome-typescript-loader',
            options: {
                useBabel: true,
                "babelCore": "@babel/core", // needed for Babel v7
            },
        },
        {
            test: /\.css$/,
            use: [MiniCssExtractPlugin.loader, "css-loader"]
        },
        {
            test: /\.(png|jpg|gif|svg)$/,
            loader: 'file-loader',
            options: {
                name: 'assets/img/[name].[ext]?[hash]'
            }
        },
    ],
},
```

```
plugins: [
    //Generate index.html in /dist => https://github.com/ampedandwired/html-webpack-plugin
    new HtmlWebpackPlugin({
        filename: 'index.html', //Name of file in ./dist/
        template: 'index.html', //Name of template in ./src
        hash: true,
    }),
    new MiniCssExtractPlugin({
        filename: "[name].css",
        chunkFilename: "[id].css"
    }),
],
```

SETTING UP PROJECT

- Tsconfig.json

```
{
  "compilerOptions": {
    "target": "es6",
    "module": "es6",
    "moduleResolution": "node",
    "declaration": false,
    "noImplicitAny": false,
    "jsx": "react",
    "sourceMap": true,
    "noLib": false,
    "suppressImplicitAnyIndexErrors": true
  },
  "compileOnSave": false,
  "exclude": [
    "node_modules"
  ]
}
```

DEVELOPING REACT APPLICATION

REACT

- Component
  - The heart of the React application.
  - Whole application is divided into components.
  - They are interactive, reusable and stateful.
  - Can be implemented with and without JSX

REACT

- **Components without JSX:**
  - Using React without JSX is especially convenient when you don't want to set up compilation in your build environment.
  - DOM Can be created using React functions explicitly
    - React.createElement(‘<DOM Element>’,{object},{PROP-Types});
  - E.g.
    - React.createElement('div',null,`Hello ${this.props.message}`);

REACT

```
class NoJSXComponent extends React.Component{
render(){
        return React.createElement('div',null,`Hello ${this.props.message}`);
    }
}
```

In App.js, the following code renders the Component.

```
ReactDOM.render (React.createElement(NoJSXComponent,{message:'World!!'},null),
document.getElementById('app'));
```

DEMO
REACT WITHOUT JSX

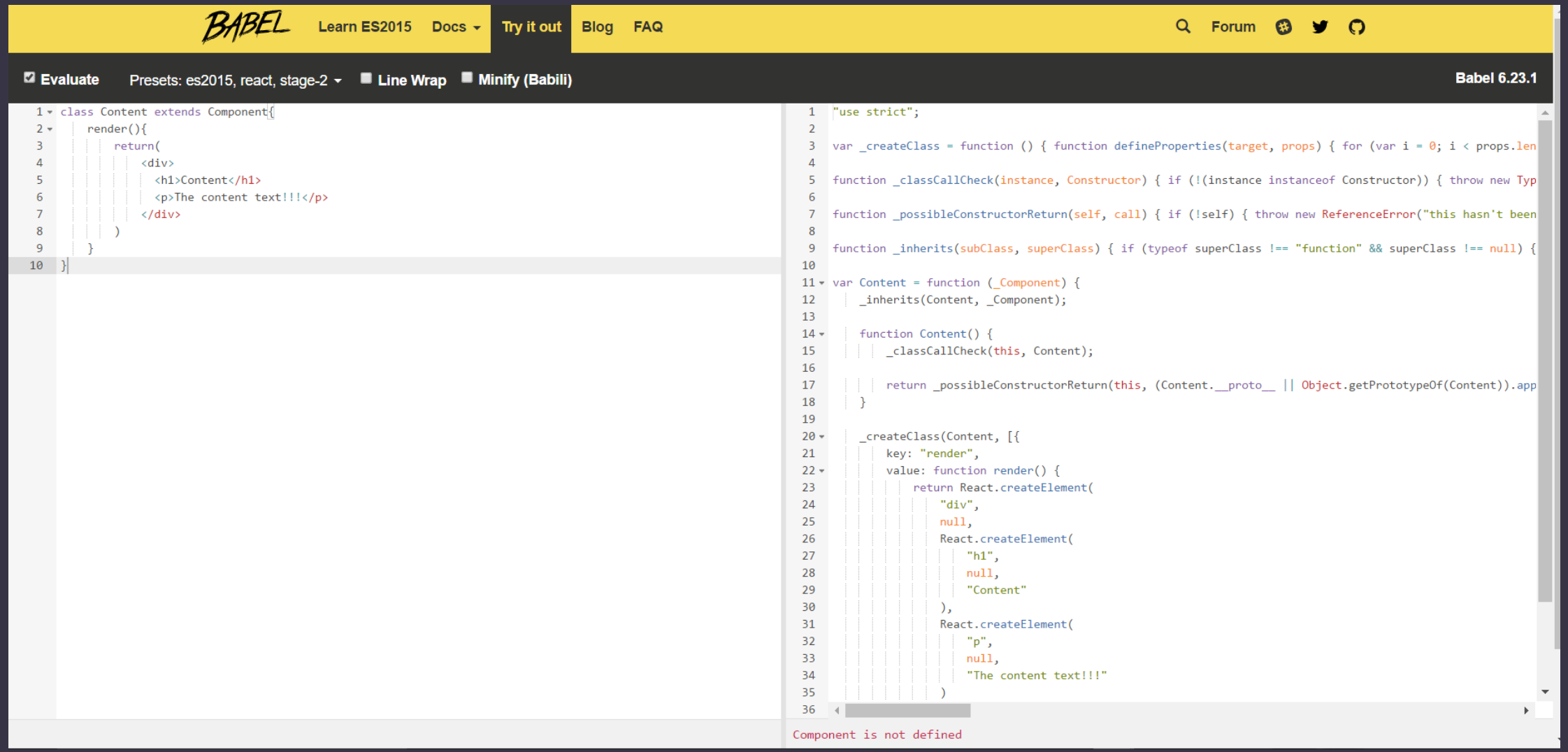
JSX

REACT

- JSX
  - It is a preprocessor step that adds XML syntax to JavaScript.
  - This is more useful to make React a lot more elegant.
  - Like XML tags, JSX tag have name, attribute and children.
- Advantages
  - JSX is faster because it performs optimization while compiling code to JavaScript.
  - It is also type-safe and most of the errors can be caught during compilation.
  - JSX makes it easier and faster to write templates if you are familiar with HTML.
  - JSX gets compiled into simple JavaScript.

REACT

https://facebook.github.io/react/jsx-compiler.html



REACT

- **Stateless Component**
  - The component act as a container for other components without any data-flow.
- **Stateful Component**
  - This mechanism is used when the component needs to render HTML based on the data.
  - This provides a huge performance boost for larger number of dynamically created elements.

REACT

- Creating Component

```
class Content extends Component{
    render(){
        return(
            <div>
              <h1>Content</h1>
              <p>The content text!!!</p>
            </div>
        )
    }
}
```

```
import React, {Component} from 'react'
```

```
class Header extends Component{
    render(){
        return(
            <div>
              <h1>Header</h1>
            </div>
        )
    }
}
```

```
class App extends Component {
    render () {
        return (
            <div>
                <Header/>
                <Content/>
            </div>
        )
    }
}
```

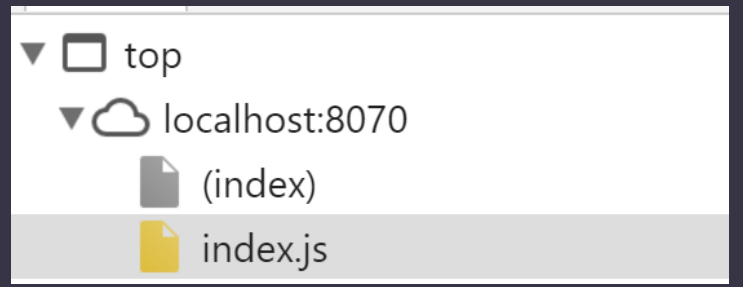
```
export default App
```

REACT

- Loading Component in DOM

```
import App from './App.jsx';
ReactDOM.render( < App / > , document.getElementById('app'));
```


```
<div id="app">
  <div data-reactroot>
    <div>
      <h1>Header</h1>
    </div>
    <div>
      <h1>Content</h1>
      <p>The content text!!!</p>
    </div>
  </div>
</div>
```



REACT

- **State**
  - This is a place where data is set for the elements in the DOM component.
- The Component class
  - setState()
    - Function to set the state property.
  - render()
    - Function to render DOM.
  - state
    - Property to define JSON schema for the data values for current DOM
  - props
    - Property to define property system for passing data from parent component to child.

REACT FORMS AND REACT COMPONENTS

CONTROLLED VS UNCONTROLLED COMPONENTS

- Controlled and UnControlled Components
  - In most cases, react recommends using controlled components to implement forms.
  - In a controlled component, form data is handled by a React component.
  - The alternative is uncontrolled components, where form data is handled by the DOM itself.

REACT LIFECYCLE

REACT

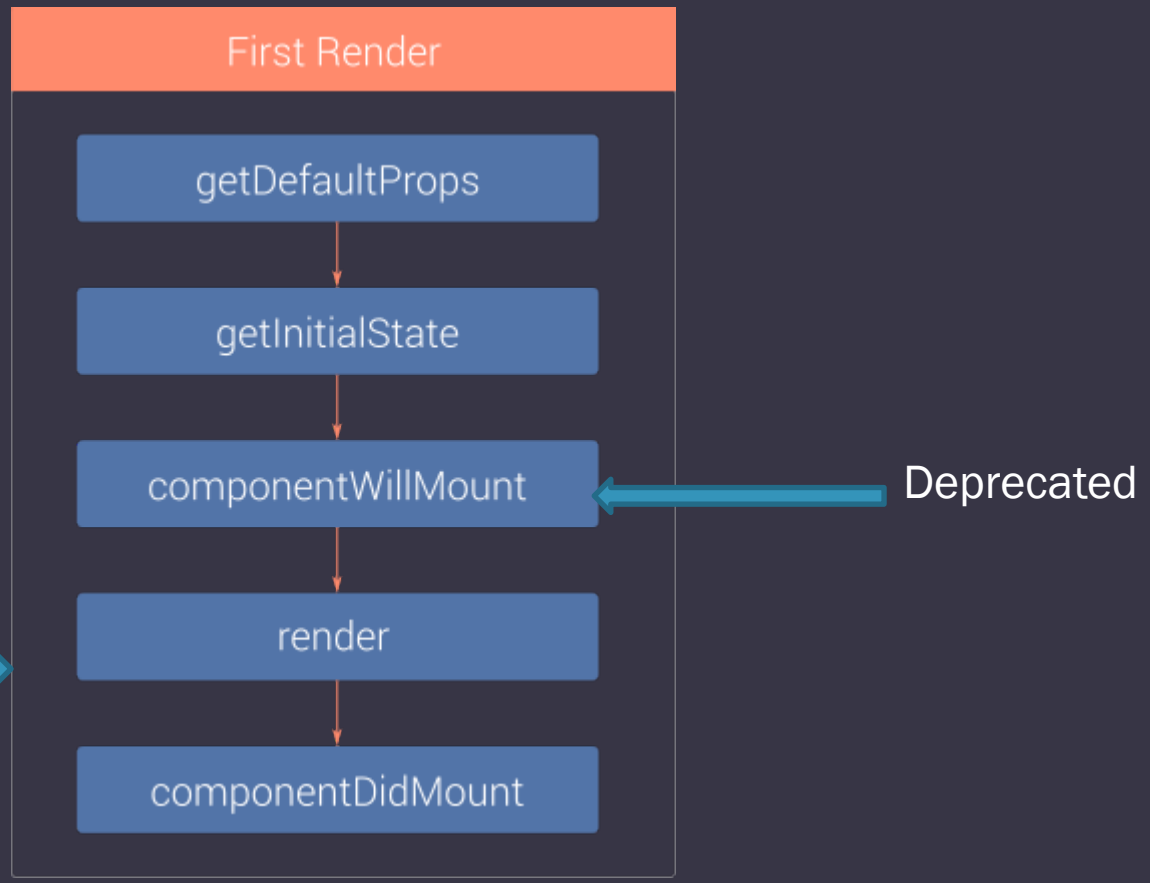
- Lifecycle Stages
  - Initial Render
  - Props Change
  - State Change
  - Component Unmount

INITIAL RENDER

```
componentDidCatch(errorString, errorInfo) {
  this.setState({
    error: errorString
  });
  ErrorLoggingTool.log(errorInfo);
}

render() {
  if(this.state.error) return <ShowErrorMessage error=
{this.state.error} />
  return (
    // render normal component output
  );
}
```

componentDidCatch(errorString, errorInfo)

New is 16.x



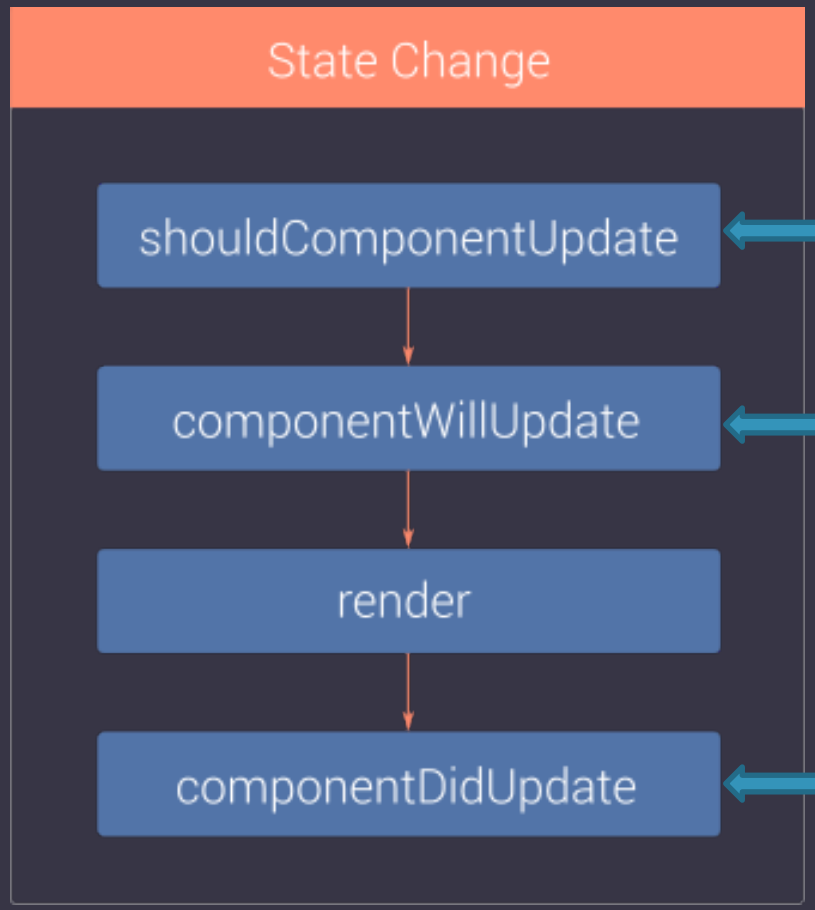
http://javascript.tutorialhorizon.com/2014/09/13/execution-sequence-of-a-react-components-lifecycle-methods/

PROPS CHANGE

Deprecated

http://javascript.tutorialhorizon.com/2014/09/13/execution-sequence-of-a-react-components-lifecycle-methods/

STATE CHANGE



nextProps, nextState, nextContext

Depricated

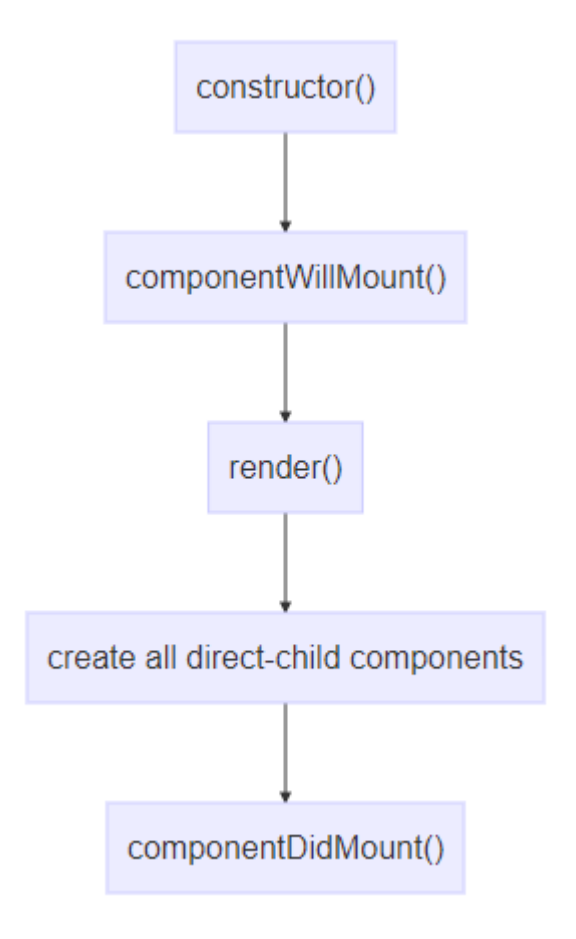
prevProps, prevState, prevContext

http://javascript.tutorialhorizon.com/2014/09/13/execution-sequence-of-a-react-components-lifecycle-methods/

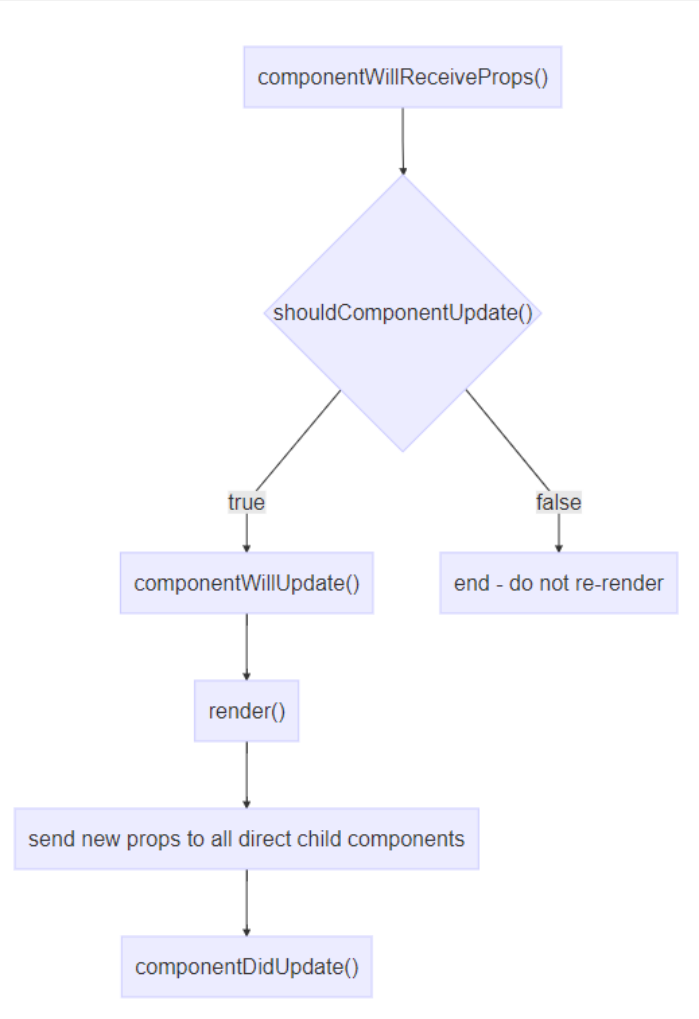
COMPONENT UNMOUNT

http://javascript.tutorialhorizon.com/2014/09/13/execution-sequence-of-a-react-components-lifecycle-methods/

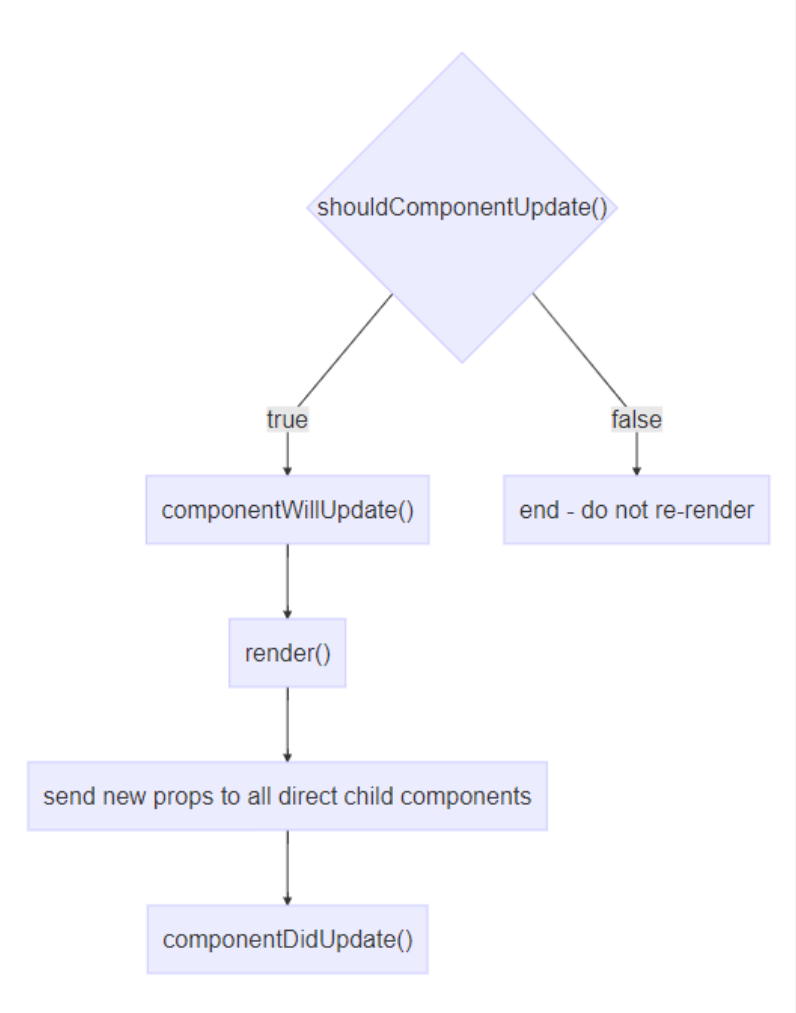
Component creation



Component re-rendering due to re-rendering of the parent component

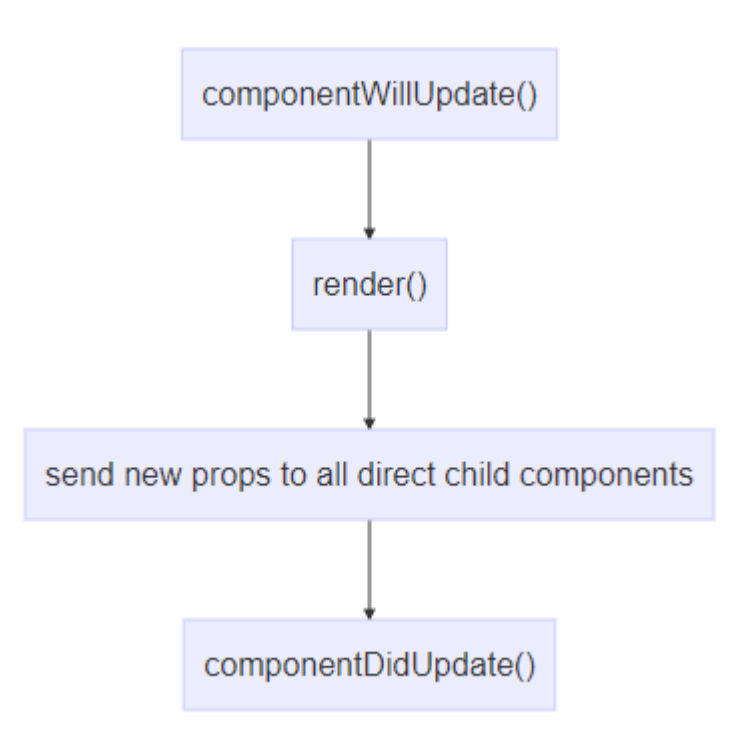


Component re-rendering due to internal change (e.g. a call to this.setState())

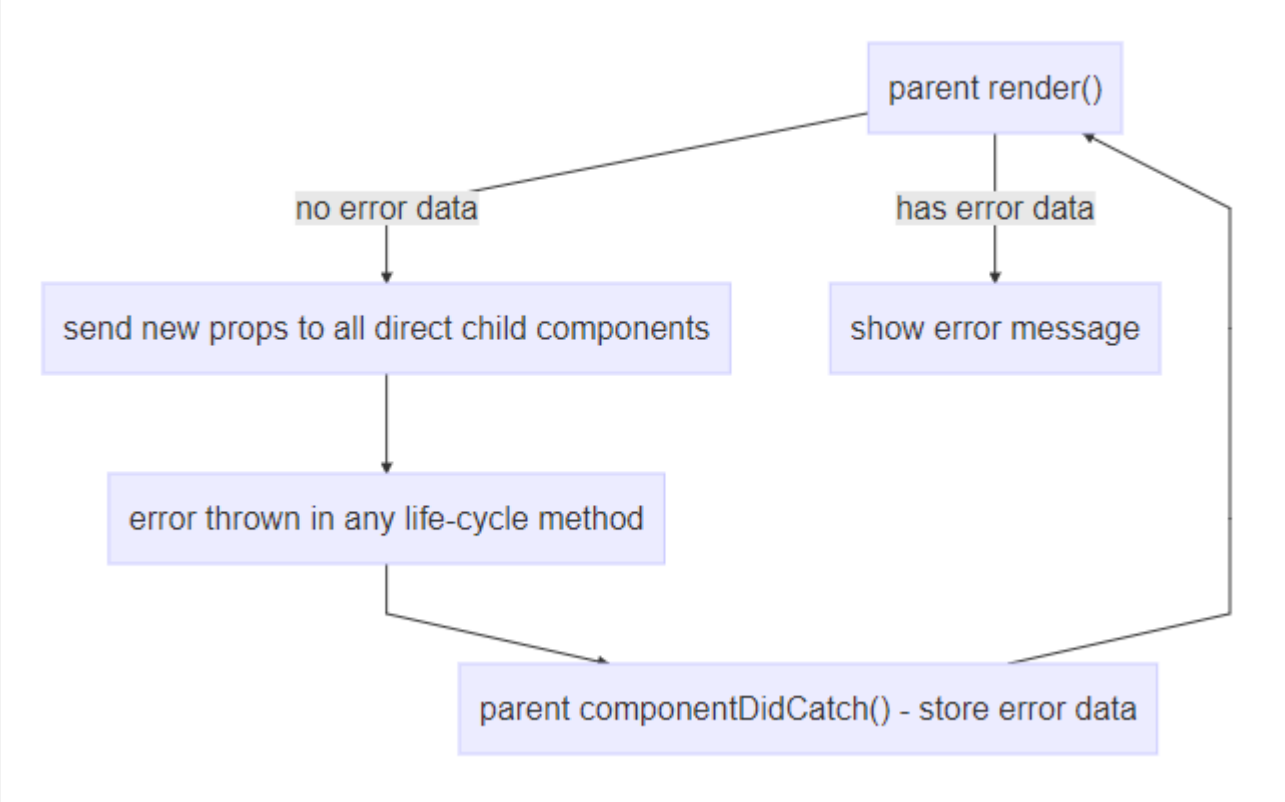


https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d

Component re-rendering due to call to this.forceUpdate

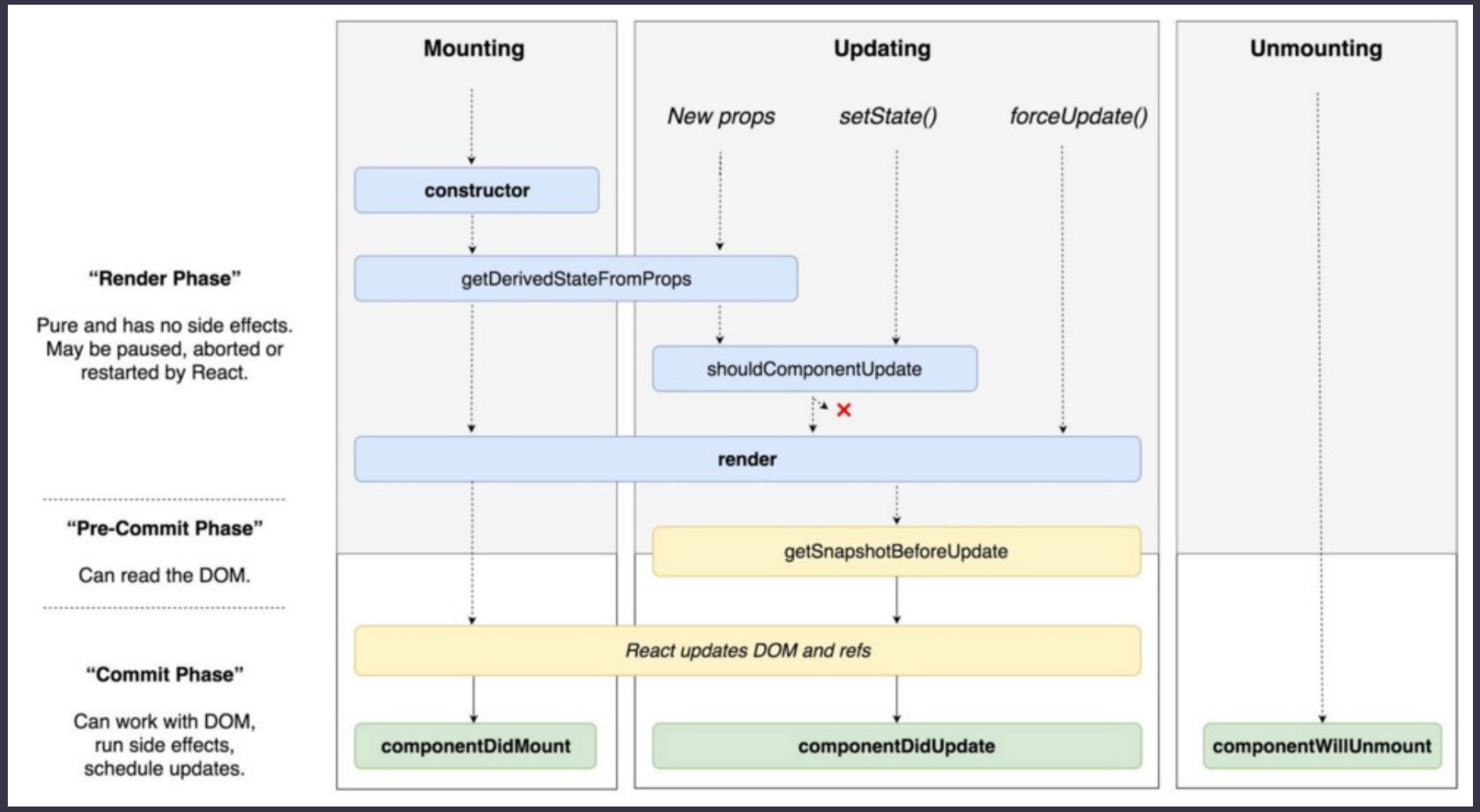


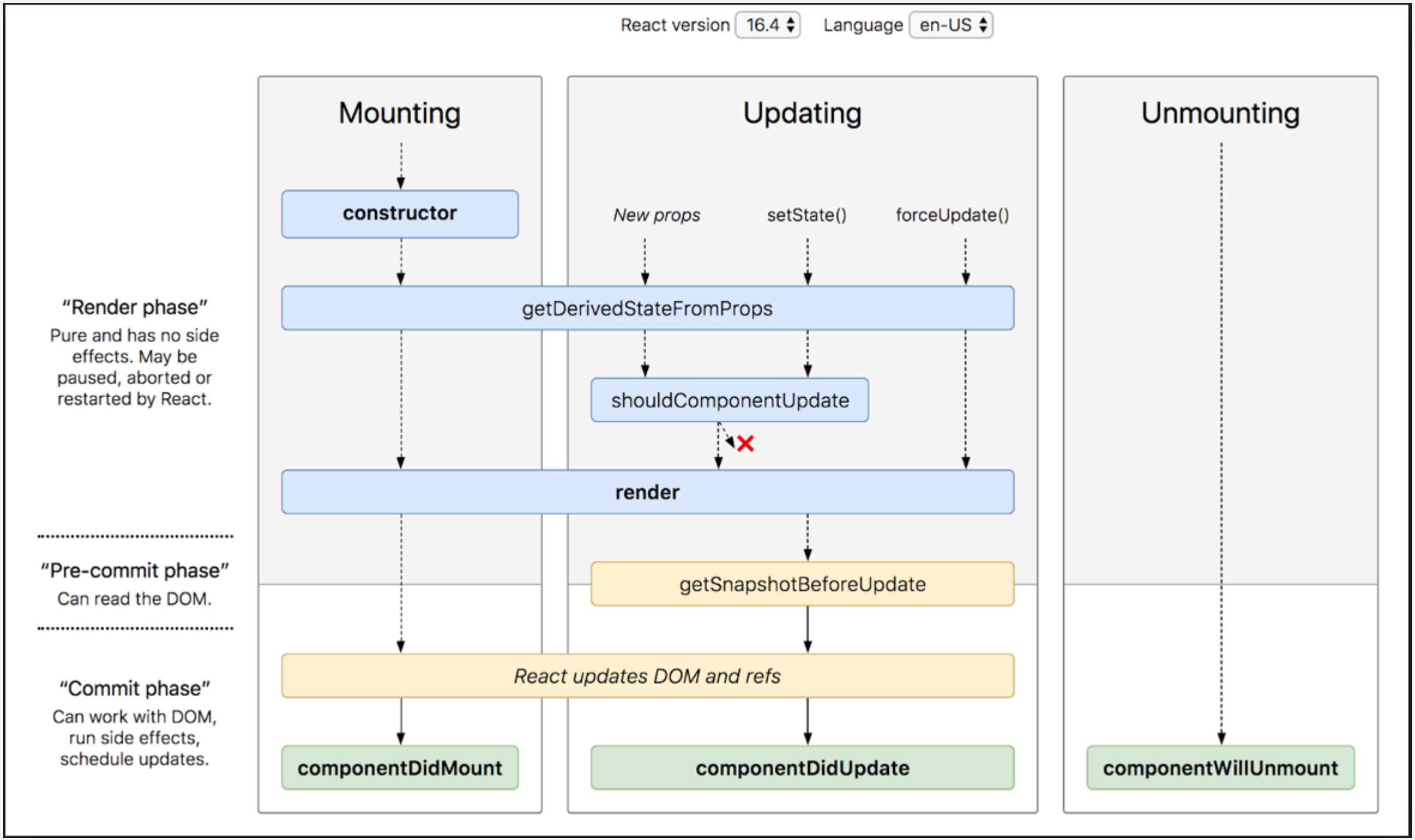
Component re-rendering due to catching an error



https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d

REACT 16.X


Mounting
Updating
Unmounting
New props
setState()
forceUpdate()
constructor
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render Phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-Commit Phase"
Can read the DOM.
"Commit Phase"
Can work with DOM, run side effects, schedule updates.


React version 16.4
Language en-US
Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
getDerivedStateFromProps
shouldComponentUpdate
render
getSnapshotBeforeUpdate
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount
"Render phase"
Pure and has no side effects. May be paused, aborted or restarted by React.
"Pre-commit phase"
Can read the DOM.
"Commit phase"
Can work with DOM, run side effects, schedule updates.

REACT HOOKS

REACT HOOKS

- New features introduced in React 16.8
- Supporting Functional Components
- Uses all features of React like State, Props, etc.
- Provides mechanism to performs Lifecycle operations
- Property Based Operations for
  - State
  - Effects
  - Context
- Examples
  - useState()
  - useContext()
  - useEffect()

REACT HOOKS

- useState()
  - const [employee, updateState] = useState({ EmpNo: 0, EmpName: "" });
  - const [employees, display] = useState([]);
  - onChange={evt =>
  - updateState({ ...employee, EmpNo: parseInt(evt.target.value) })
  - }

- useContext()
  - Create Context
    - import { createContext } from "react";
    - export const DataContext = createContext(null);
  - Using Context

    <DataContext.Provider value={{employees,updateState}}>

    <TableComponentContext/>

    </DataContext.Provider>
  - Reading Values
    - const data = useContext(DataContext);

REACT HOOKS

- useEffect()
  - Performs Operations for
    - External Calls
    - Managing Conditional Rendering
    - Clean-Up operations
    - Performs ComponentDidUpdate()/render()/ComponentDidMount()
- useEffect() Syntax
  - useEffect(operation to be executed, return the cleanup operation, watch state)

REACT FORMS

- Forms consist of the following parts
  - DOM for rendering
  - Field Definition, a.k.a. ***Template***
  - Client-Side Logic, e.g. UI Logic, a.k.a. ***Form Model.***
  - Fields to be exposed to UI, a.k.a. ***Domain Model.***

REACT FORMS

- A Unit of the Data Management
  - Must be enclosed in <form> tag.
  - Must define ‘name’ attribute for the for each input and select element.
  - Must define ‘OnChange’ event for each input and select element to read value from it.
- Created using Controlled Component (recommended) and UnControlled Component.
- Used for developing LOB Apps View

REACT FORMS

```
onChange(e) {
this.setState({
[e.target.name]: e.target.value
});
}
```

REACT VALIDATIONS

REACT VALIDATIONS

- Defining properties for the component using
  - propTypes object
    - React.PropTypes
      - Array
      - Number
      - String
      - Bool
      - Object
      - Func

```
{this.props.bool?'True':'False'}
```

```
{this.props.pFunc(100)}
```

REACT

. Setting Validation Rules

```
App.propTypes = {
    pArray:React.PropTypes.array.isRequired,
    pNumber:React.PropTypes.number,
    pString:React.PropTypes.string,
    pBoolean:React.PropTypes.bool.isRequired,
    pObject:React.PropTypes.object,
    pFunc:React.PropTypes.func
}
```

```
App.defaultProps = {
    pArray:[10,20,30,40,50],
    pNumber:100,
    pString:'M.R.Sabnis',
    pBoolean:true,
    pObject:{
        val1:'A',
        val2:'B',
        val3:'C'
    },
    pFunc:function(x){return x}
}
```

REACT.JS
ADVANCE GUIDE

CODE SPLITTING

CODE SPLITTING

- Code Splitting
- To avoid winding up with a large bundle, it’s good to get ahead of the problem and start “splitting” your bundle.
- Code-splitting your app can help you “lazy-load” just the things that are currently needed by the user, which can dramatically improve the performance of your app.
- Using import() object

Before:

```
import { add } from './math';

console.log(add(16, 26));
```

After:

```
import("./math").then(math => {
  console.log(math.add(16, 26));
});
```

CODE SPLITTING

- Code Splitting
- React.lazy()

The `React.lazy` function lets you render a dynamic import as a regular component.

Before:

```
import OtherComponent from './OtherComponent';
```

After:

```
const OtherComponent = React.lazy(() => import('./OtherComponent'));
```

This will automatically load the bundle containing the `OtherComponent` when this component is first rendered.

`React.lazy` takes a function that must call a dynamic `import()`. This must return a `Promise` which resolves to a module with a `default` export containing a React component.

CODE SPLITTING

- Code Splitting
- The lazy component should then be rendered inside a Suspense component, which allows us to show some fallback content (such as a loading indicator) while we're waiting for the lazy component to load.

```
import React, { Suspense } from 'react';

const OtherComponent = React.lazy(() => import('./OtherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <OtherComponent />
      </Suspense>
    </div>
  );
}
```

CODE SPLITTING

- Code Splitting
- The fallback prop accepts any React elements that you want to render while waiting for the component to load. You can place the Suspense component anywhere above the lazy component. You can even wrap multiple lazy components with a single Suspense component.

```
import React, { Suspense } from 'react';

const OtherComponent = React.lazy(() => import('./OtherComponent'));
const AnotherComponent = React.lazy(() => import('./AnotherComponent'));

function MyComponent() {
  return (
    <div>
      <Suspense fallback={<div>Loading...</div>}>
        <section>
          <OtherComponent />
          <AnotherComponent />
        </section>
      </Suspense>
    </div>
  );
}
```

ERROR BOUNDARIES

ERROR BOUNDARIES

- Error boundaries are
  - React components that **catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI** instead of the component tree that crashed.
  - Error boundaries catch errors during rendering, in lifecycle methods, and in constructors of the whole tree below them.

ERROR BOUNDARIES

- A class component becomes an error boundary if it defines either (or both) of the lifecycle methods static getDerivedStateFromError() or componentDidCatch().
- Use static getDerivedStateFromError() to render a fallback UI after an error has been thrown.
- Use componentDidCatch() to log error information.

```
<ErrorBoundary>
  <MyWidget />
</ErrorBoundary>
```

```
class ErrorBoundary extends React.Component {
  constructor(props) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error) {
    // Update state so the next render will show the fallback UI.
    return { hasError: true };
  }

  componentDidCatch(error, errorInfo) {
    // You can also log the error to an error reporting service
    logErrorToMyService(error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      // You can render any custom fallback UI
      return <h1>Something went wrong.</h1>;
    }

    return this.props.children;
  }
}
```

REACT HIGHER-ORDER-COMPONENT

HIGHER-ORDER-COMPONENTS

- A **higher-order component** is a function that takes a component and returns a new component.
- A **higher-order component** (HOC) is the advanced technique in React.js for reusing a component logic.
- **Higher-Order Components** are not part of the React API. They are the pattern that emerges from React's compositional nature.
- The component transforms props into UI, and a **higher-order component** converts a component into another component.
- **Technically its is a Pure Function.**

REACT HOC

- React **Higher-Order Components**
- A higher-order component (HOC) is an advanced technique in React for reusing component logic.
- HOCs are not part of the React API, per se.
- They are a pattern that emerges from React's compositional nature.
- Concretely, **a higher-order component is a function that takes a component and returns a new component.**
- A higher-order component transforms a component into another component.

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

REACT HOC

```
import React, {
    Component
} from 'react';

export default function Hoc(HocComponent, data) {
    return class extends Component {
        constructor(props) {
            super(props);
            this.state = {
                data: data
            };
        }

        render() {
            return ( <HocComponent
                        data = {this.state.data} { ...this.props}
                />
            );
        }
    }
}
```

A Function that accepts Component and Data as input parameter.

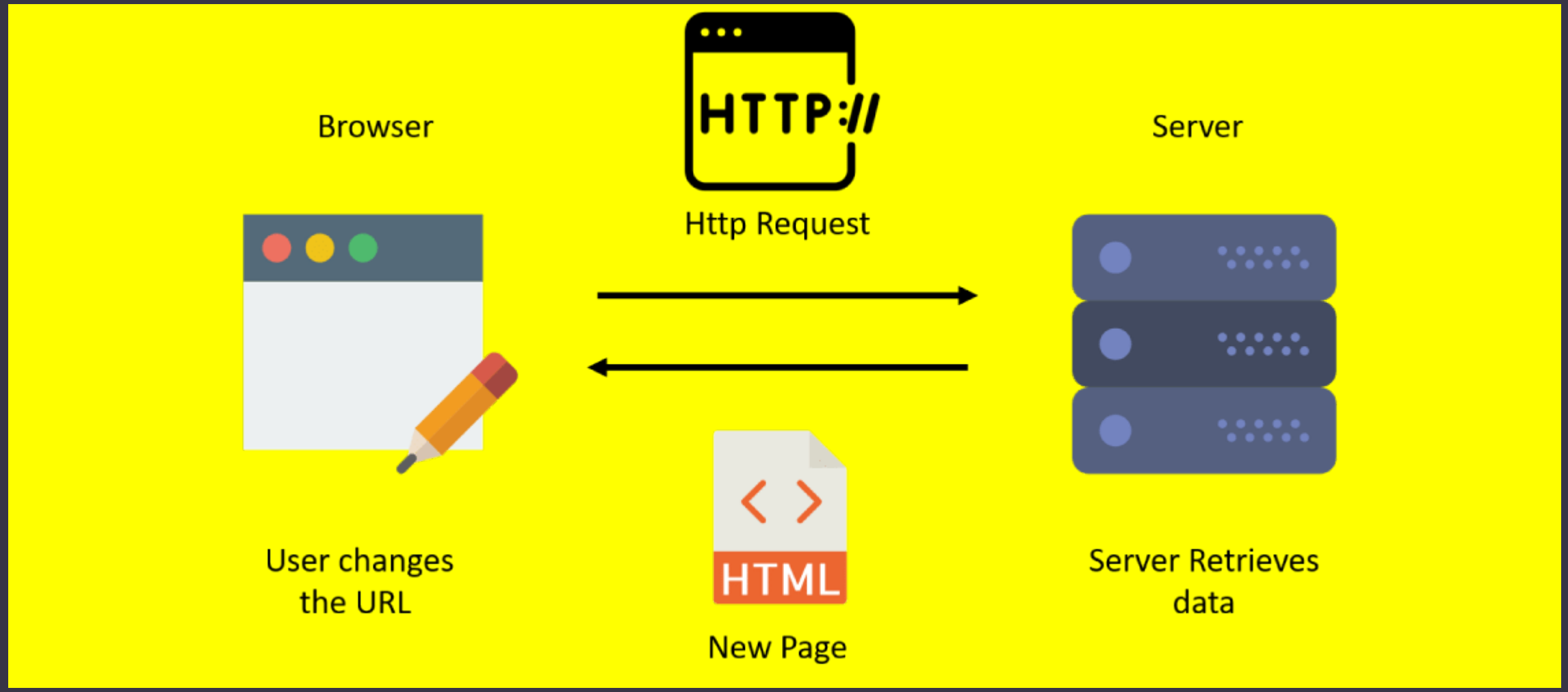
ROUTING

ROUTING

- Routing
  - Thinking to build Single Page Application
  - Creating a navigation mechanism across components
  - Used for showing data across components by creating requirement specific components
  - Standard used by various technologies to avoid complex UI and UX
  - Can control the amount of DOM loaded in browser

ROUTING

- Conventional Routing



ROUTING

- Implementation
- Using ‘react-router-dom’ package
- Classes offered
  - Route
  - Link
  - Switch
  - Redirect

Page/Component	Address
Home	/
About	/about
Contact	/contact
Info	/info/:id

ROUTING

- Implementation
  - import {Route, Link, Switch,Redirect} from 'react-router-dom';

```
<tr>
    <td>
        <Link to="/">Home</Link>
    </td>
    <td>
        <Link to={`/about/${this.state.id}`}>About</Link>
    </td>
    <td>
        <Link to="/contact">Contact</Link>
    </td>
</tr>
```

ROUTING

- Defining Route Table

```
<div>
    <Switch>
        <Route exact path="/" component={HomeComponent}></Route>
        <Route exact path="/about/:id" component={AboutComponent}></Route>
        <Route exact path="/contact" component={ContactComponent}></Route>
        <Redirect to="/"/>
    </Switch>
</div>
```

- Navigation using Events

```
navigateToContact(){
    this.props.history.push('/contact');
}
```

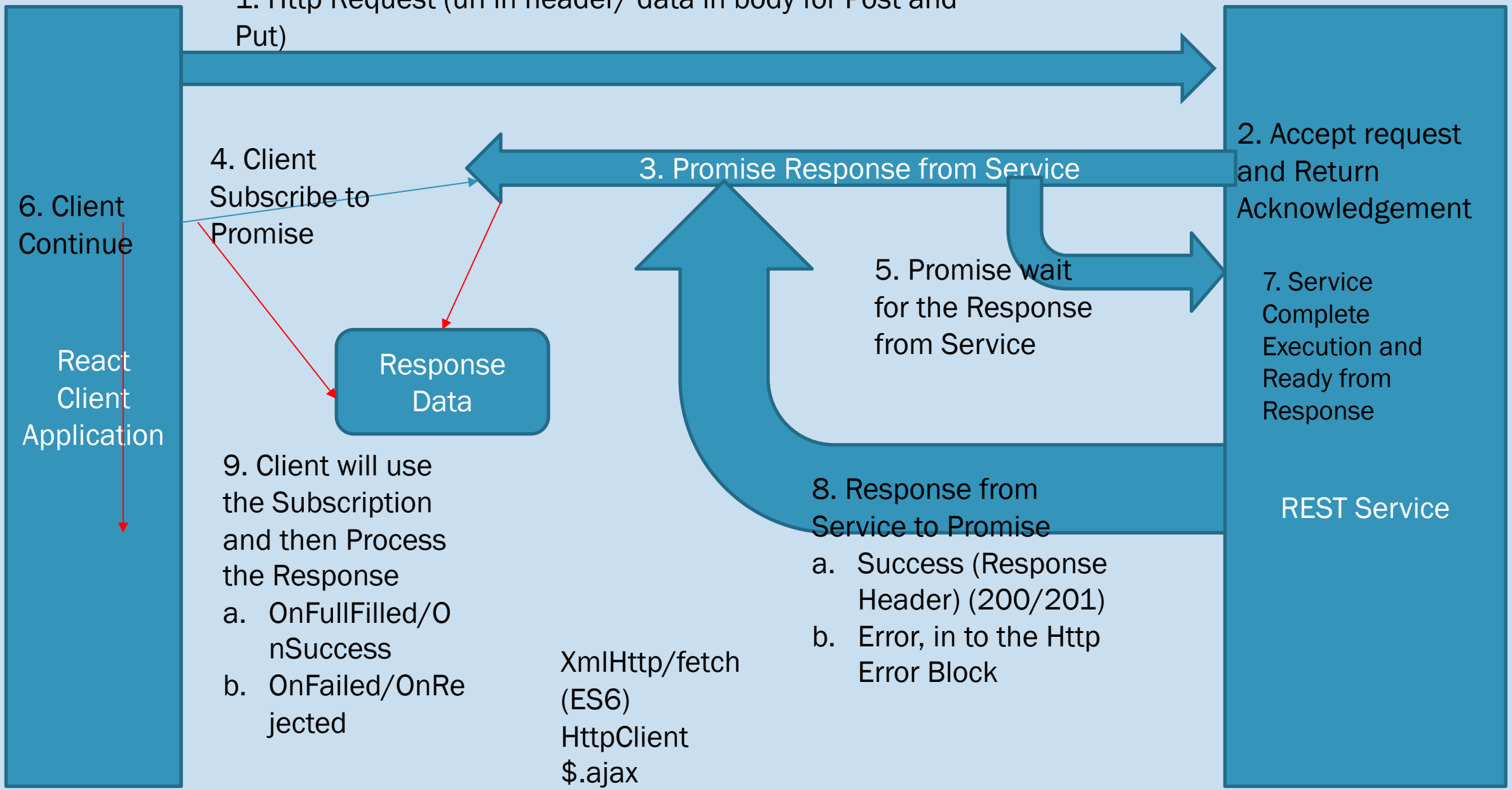
```
<input type="button" value="Navigate to Contact"
onClick={this.navigateToContact.bind(this)} className="btn btn-warning"/>
```

ROUTING

- Reading Route Parameter

```
<div className="container">{this.state.message} {this.props.match.params.id}</div>
```

REACT
WORKING WITH NETWORKING
HTTP


1. Http Request (url in header/ data in body for Post and Put)
2. Accept request and Return Acknowledgement
3. Promise Response from Service
4. Client Subscribe to Promise
5. Promise wait for the Response from Service
6. Client Continue
7. Service Complete Execution and Ready from Response
8. Response from Service to Promise
a. Success (Response Header) (200/201)
b. Error, in to the Http Error Block
9. Client will use the Subscription and then Process the Response
a. OnFullFilled/OnSuccess
b. OnFailed/OnRejected
Response Data
React Client Application
REST Service
XmlHttp/fetch (ES6) HttpClient $.ajax

REACT

- AJAX Calls
  - Http GET

```
fetch('http://localhost:21163/api/EmployeeInfoAPI')
            .then(res => res.json())
            .then(data => {
                this.setState({employees: data});
                alert(JSON.stringify(this.state.employees));
            });
```

REACT

- AJAX Calls
  - Http POST

```
fetch('http://localhost:21163/api/EmployeeInfoAPI', {
            method: "POST",
            headers: {
                'Accept': 'application/json',
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(data)
                        })
            .then(function (res) {
                return res.json();
            })
            .then(function (data) {
                alert('Error ' + JSON.stringify(data));
            })
```

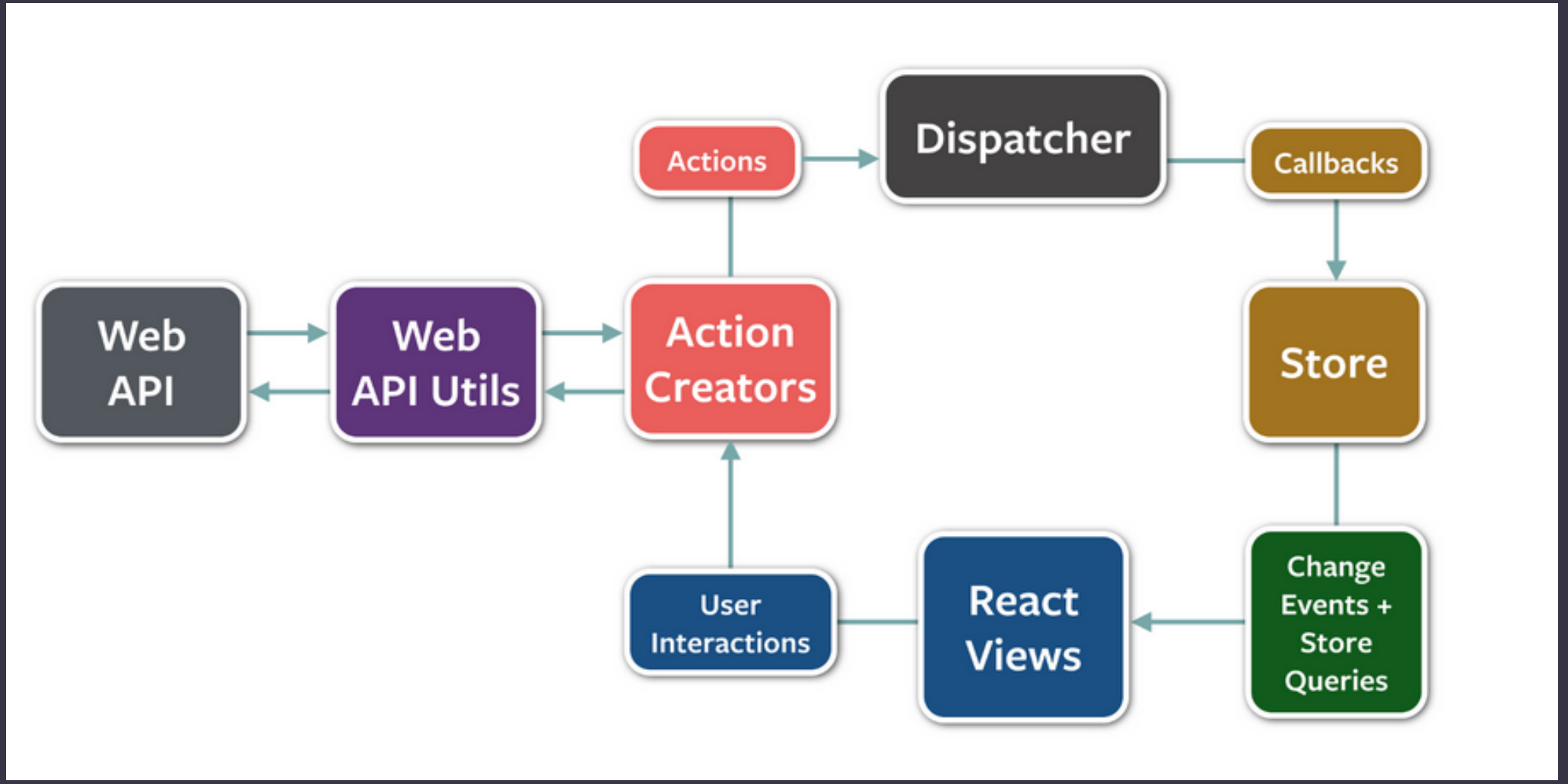
FLUX

Concept for using React.js for more complex applications

WHAT IS FLUX

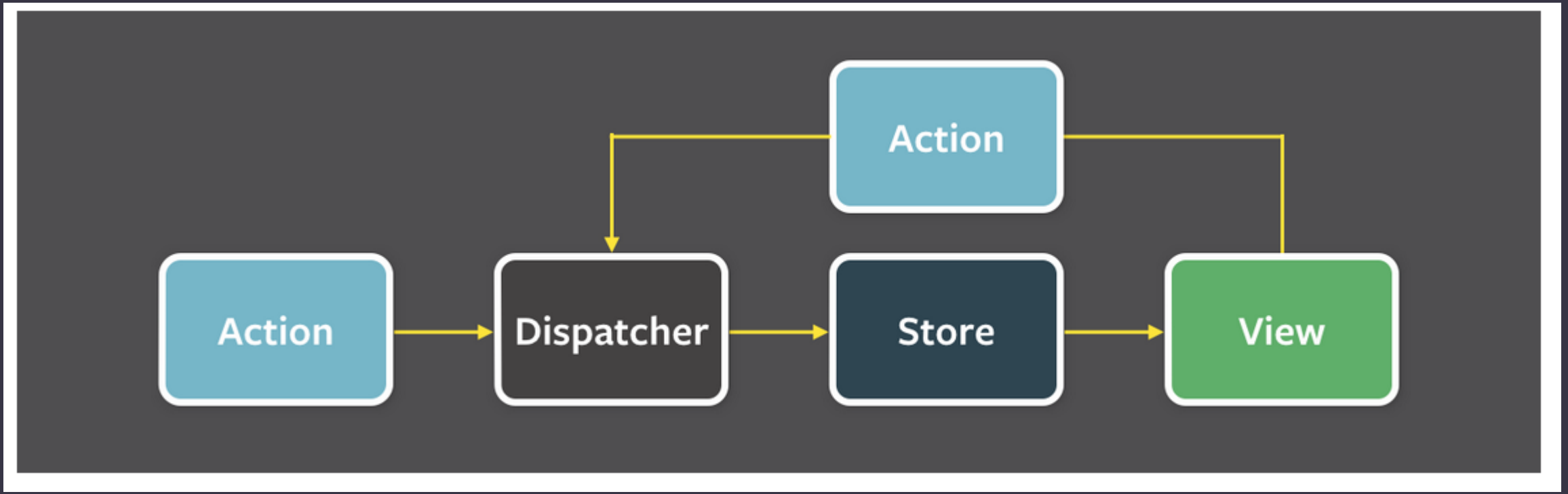
- Flux is the application architecture that Facebook uses for building client-side web applications.
- It complements React's composable view components by utilizing a unidirectional data flow.
- It's more of a pattern rather than a formal framework, and you can start using Flux immediately without a lot of new code.
- This is not MVC

FLUX BY DIAGRAM



https://github.com/facebook/flux

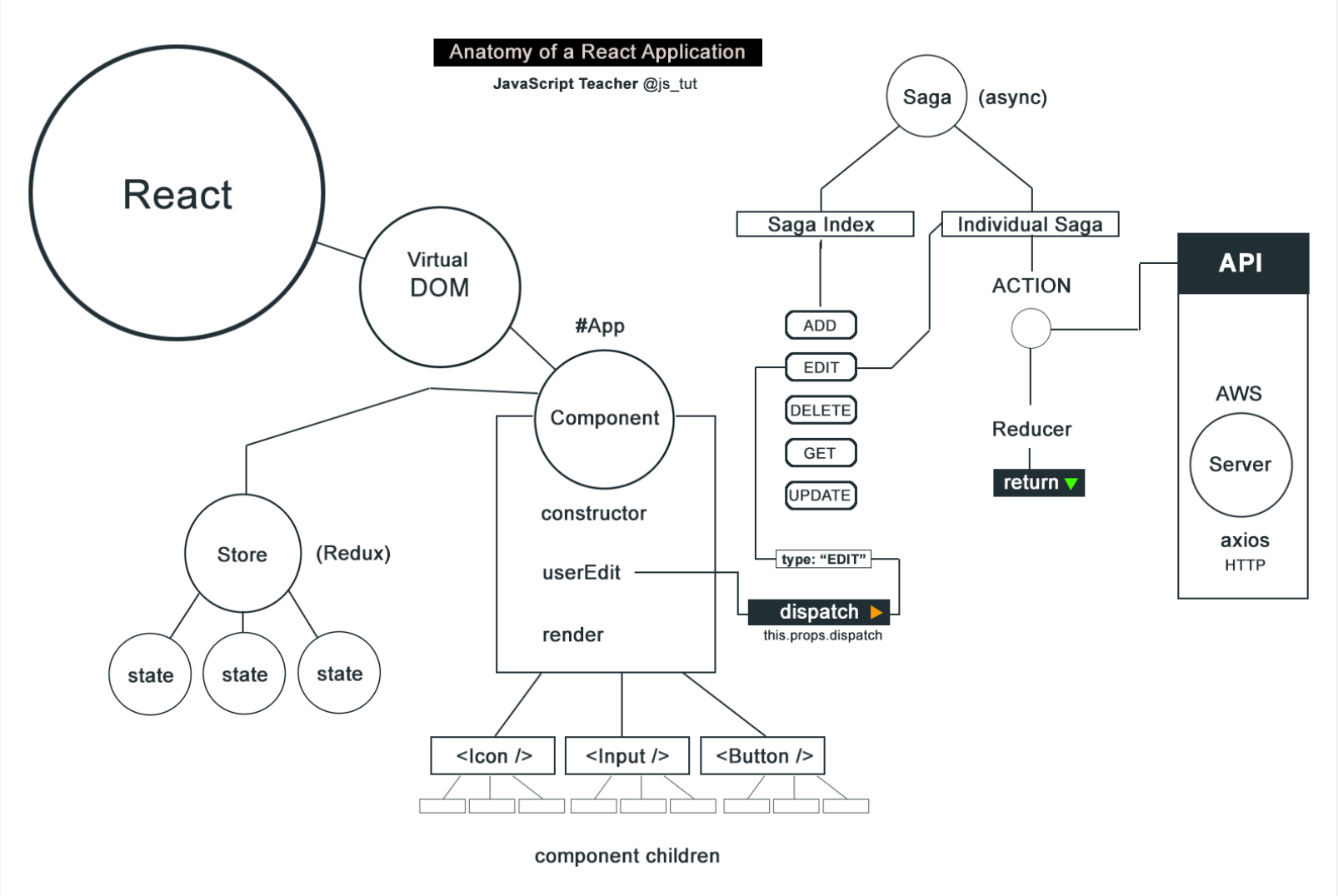
HOW FLUX WORKS



1. Views send actions to the dispatcher.
2. The dispatcher sends actions to every store.
3. Stores send data to the views.

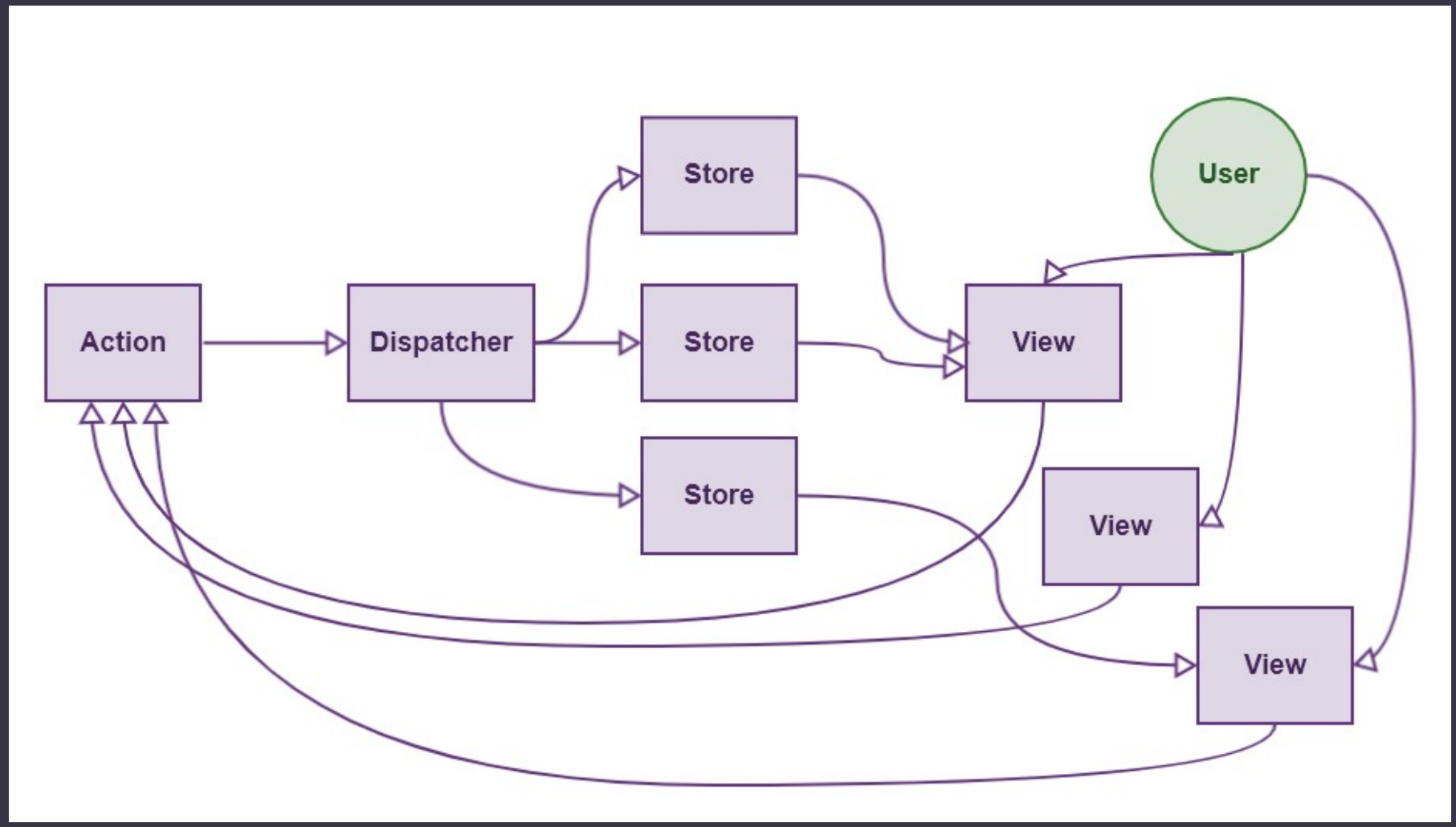
https://github.com/facebook/flux/tree/master/examples/flux-concepts

REACT APPLICATION — ARCHITECTURE



https://jstutorial.medium.com/anatomy-of-a-react-application-d465739ec6be

FLUX

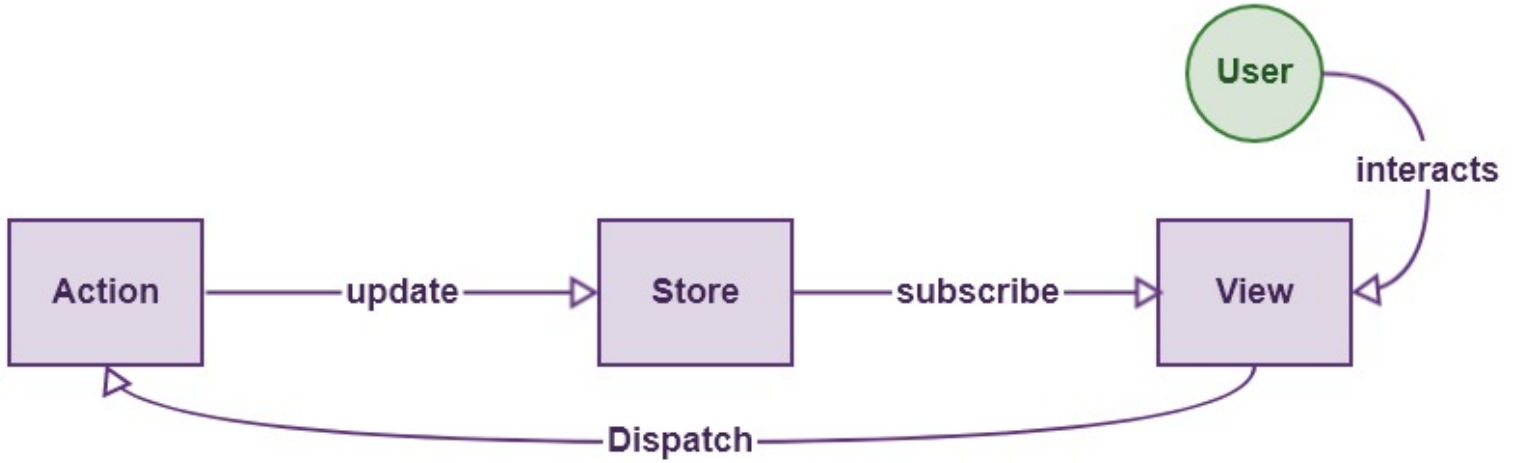


It is a unidirectional data flow. When the application gets bigger and bigger, then multiple stores manage the data.

https://appdividend.com/2017/08/23/redux-tutorial-example-scratch/

REDUX

- **Redux** is the predictable state container for JavaScript applications. Redux is also following the **Unidirectional flow,** but it is entirely different from **Flux. Flux has multiple stores.**



https://appdividend.com/2017/08/23/redux-tutorial-example-scratch/

Three Principles Of Redux

1. Single source of truth.
2. The state is read-only.
3. Changes are made with pure functions.

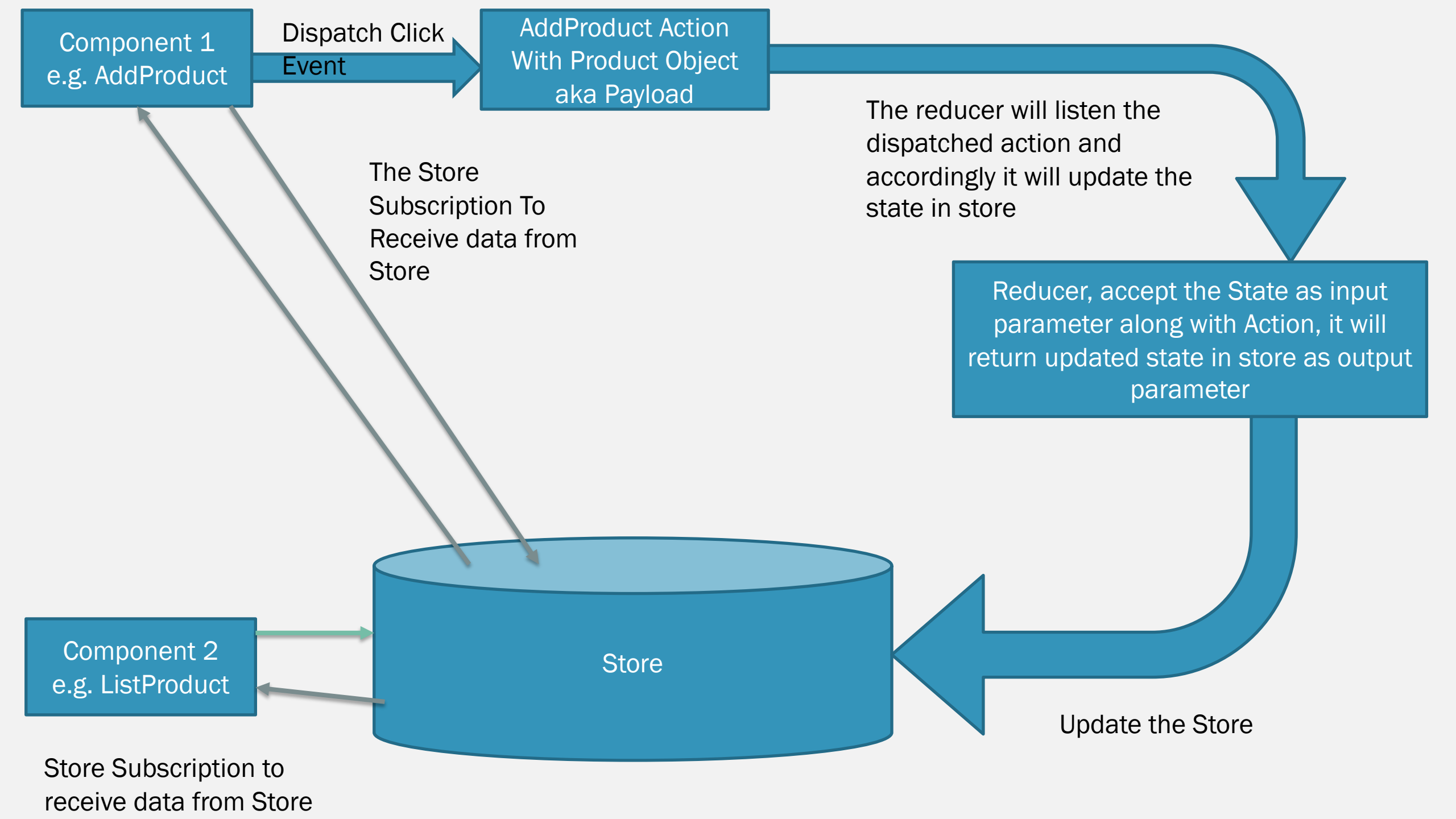
REDUX

Redux is a predictable state container for JavaScript applications.
It is the state of our whole application is stored in an object within a single store.

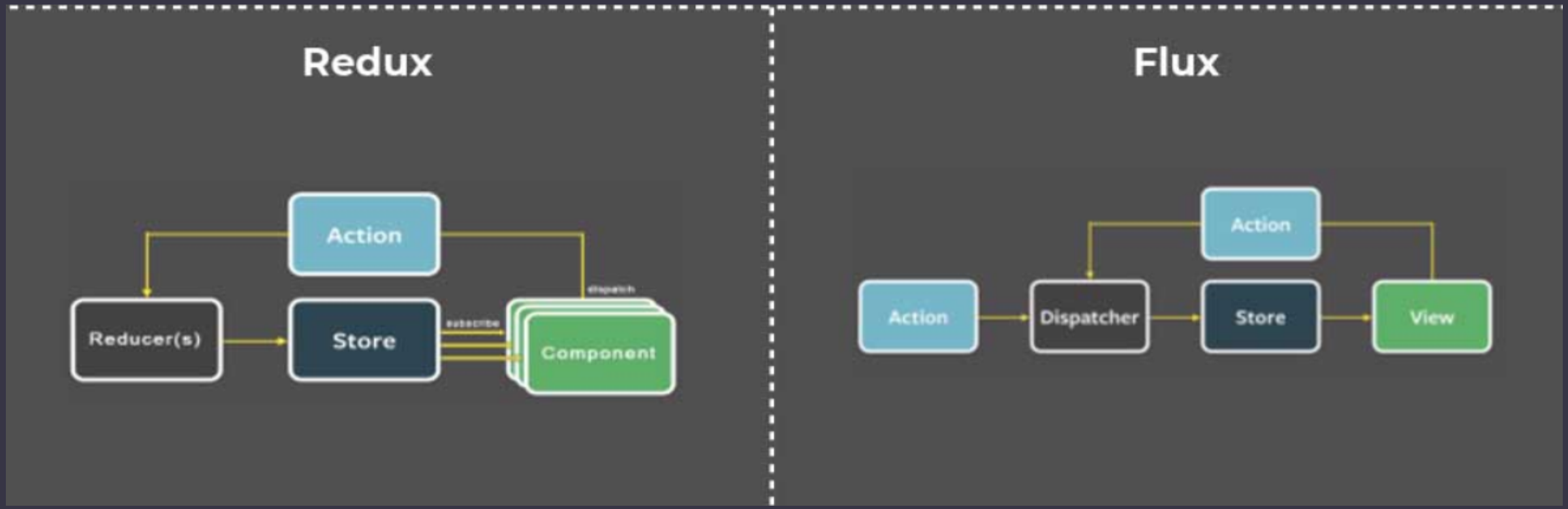
There is an only way to change the state is to emit an action, an object describing what happened.

To specify how actions transform the state, you write pure reducers.

https://appdividend.com/2017/08/23/redux-tutorial-example-scratch/


Component 1
e.g. AddProduct
Dispatch Click Event
AddProduct Action With Product Object aka Payload
The reducer will listen the dispatched action and accordingly it will update the state in store
The Store Subscription To Receive data from Store
Reducer, accept the State as input parameter along with Action, it will return updated state in store as output parameter
Component 2
e.g. ListProduct
Store
Update the Store
Store Subscription to receive data from Store

FLUX VS REDUX



https://www.educba.com/redux-vs-flux/

FLUX VS REDUX

The Basis Of Comparison Between Redux vs Flux	REDUX	FLUX
Developed	Dan Abramov and Andrew Clark	By facebook
Stable release	4.0.0(April 2017)	3.1.3(Nov 2016)
Initial release	June 2, 2015	the Year 2011
Store	Single store	Multiple stores
Dispatcher	No	Singleton dispatcher
State	Immutable	Mutable
GitHub Stats	43.2K stars	15.5K stars
Integration	With React, jumpsuit, Meatier and react.js boilerplate	React, TuxedoJS and Fluxxor
Pro's	• Predictable state • Work well with React • Easy debugging • Log everything • Test without browser • Hot reloading • The state stored in single object	• Unidirectional data flow • Architecture • No MVC • Open source

REDUX

Actions

Actions are payloads of information that send data from your application to your store. You send them to the store using. `store.dispatch()`

Actions are plain JavaScript objects. Actions must have a `type` property that indicates the type of action being performed. Types should typically be defined as string constants.

```
import { ADD_TODO, REMOVE_TODO } from '../actionTypes'
```

Action Creators

Action creators are exactly the functions that create actions.

```
function addTodo(text) {
  return {
    type: ADD_TODO,
    text
  }
}
```

https://appdividend.com/2017/08/23/redux-tutorial-example-scratch/

REDUX

Reducers

Actions describe the fact that something happened but don't specify how the application's state changes in response. That is the job of reducers.

Handling Actions

```
(previousState, action) => newState
```

This is called a reducer because it is the type of function you would pass to `Array.prototype.reduce(reducer, ?initialValue)`. It is essential that the reducer stay pure.

Following are the things you should **never** do inside a reducer:

- Mutate reducer's arguments;
- Perform side effects like database calls, API calls, and routing transitions;
- Call non-pure functions, e.g., `Date.now()` or `Math.random()`

https://appdividend.com/2017/08/23/redux-tutorial-example-scratch/

DEFINING REDUCERS

```
export function addProductReducer(state, action) {
  console.log(`In Add Reducer ${JSON.stringify(state)}`);
  switch (action.type) {
    case ADD_PRODUCT:
      // some other logic that reducer wants to execute
      return {
        product: action.product
      };
    default: {
      return state;
    }
  }
}
```

DEFINING REDUCER

```
// reducer method
export function listProductReducer(state = [], action) {
  switch (action.type) {
    case ADD_PRODUCT:
      return [...state, addProduct(undefined, action)];
    default:
      return state;
  }
}

// combine reducer
const productAppReducer = combineReducers({ listProductReducer });
```

MAPPING STATE/PROPS/COMPONENTS

```
render() {
  // props constants
  const { dispatch, visibleproducts } = this.props;
  // the AddProductClick will dispatch the trigger
  return (
    <div>
      <AddProductComponent
        AddProductClick={product => dispatch(addProduct(product))}
      />
      <hr />
      <ProductsListComponent listProductReducer={visibleproducts} />
    </div>
  );
}
```

CONNECT STATE/PROPS/COMPONENTS

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

```
function mapStateToProps(state) {
  return {
    visibleproducts: state.listProductReducer
  };
}

// connect react component to redux
export default connect(mapStateToProps)(MainReactReduxComponent);
```

REDUX

Store

A store is an object that brings them together. A store has the following responsibilities:

- Holds application state;
- Allows access to state via `getState()`;
- Allows state to be updated via `dispatch(action)`;
- Registers listeners via `subscribe(listener)`;
- It handles unregistering of listeners via the function returned by `subscribe(listener)`

> It is important to note that you will only have a single store in a Redux application. If you want to split your data handling logic, you will use reducer composition instead of many stores.

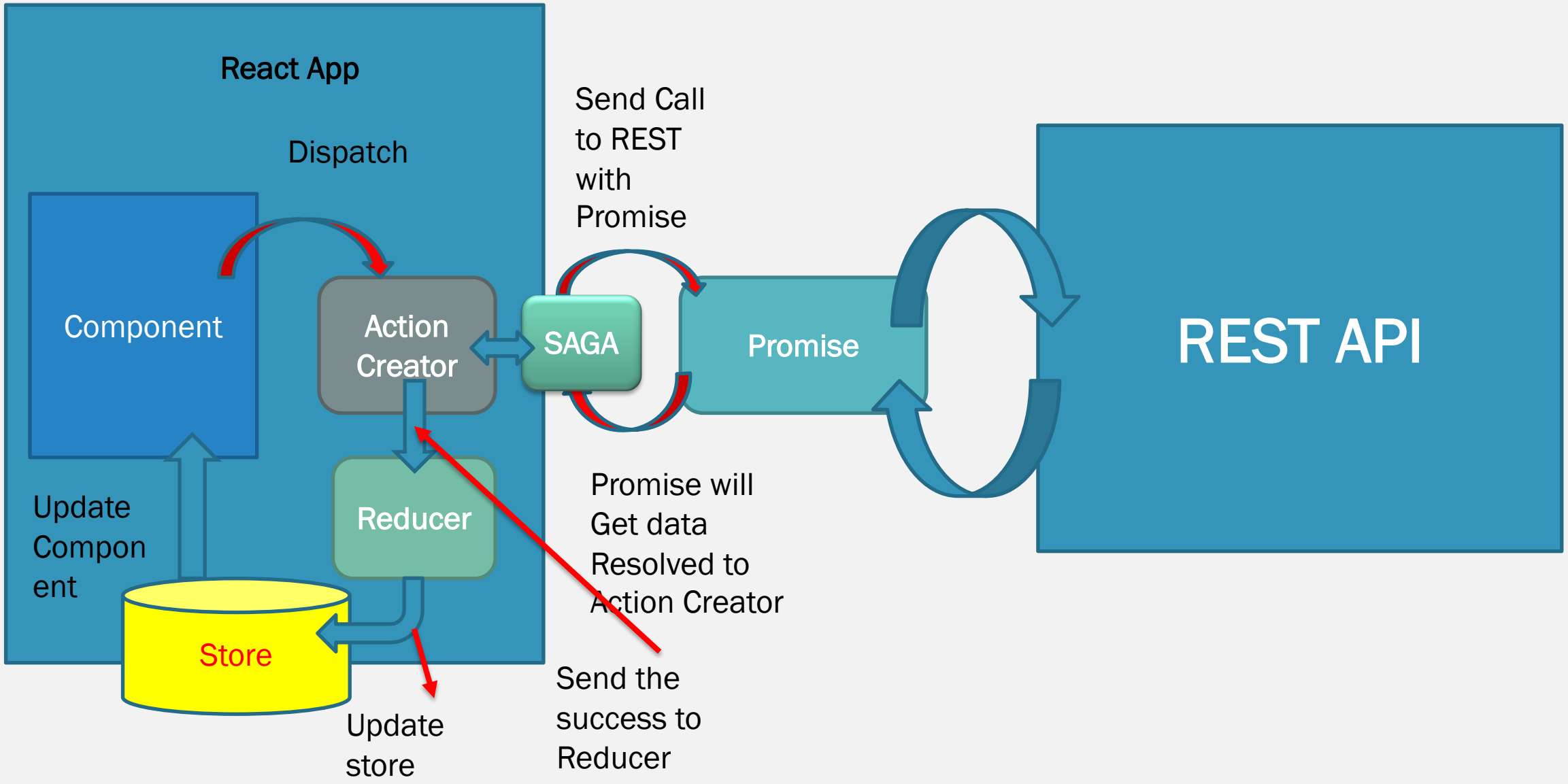
https://appdividend.com/2017/08/23/redux-tutorial-example-scratch/

CREATING AND USING STORE

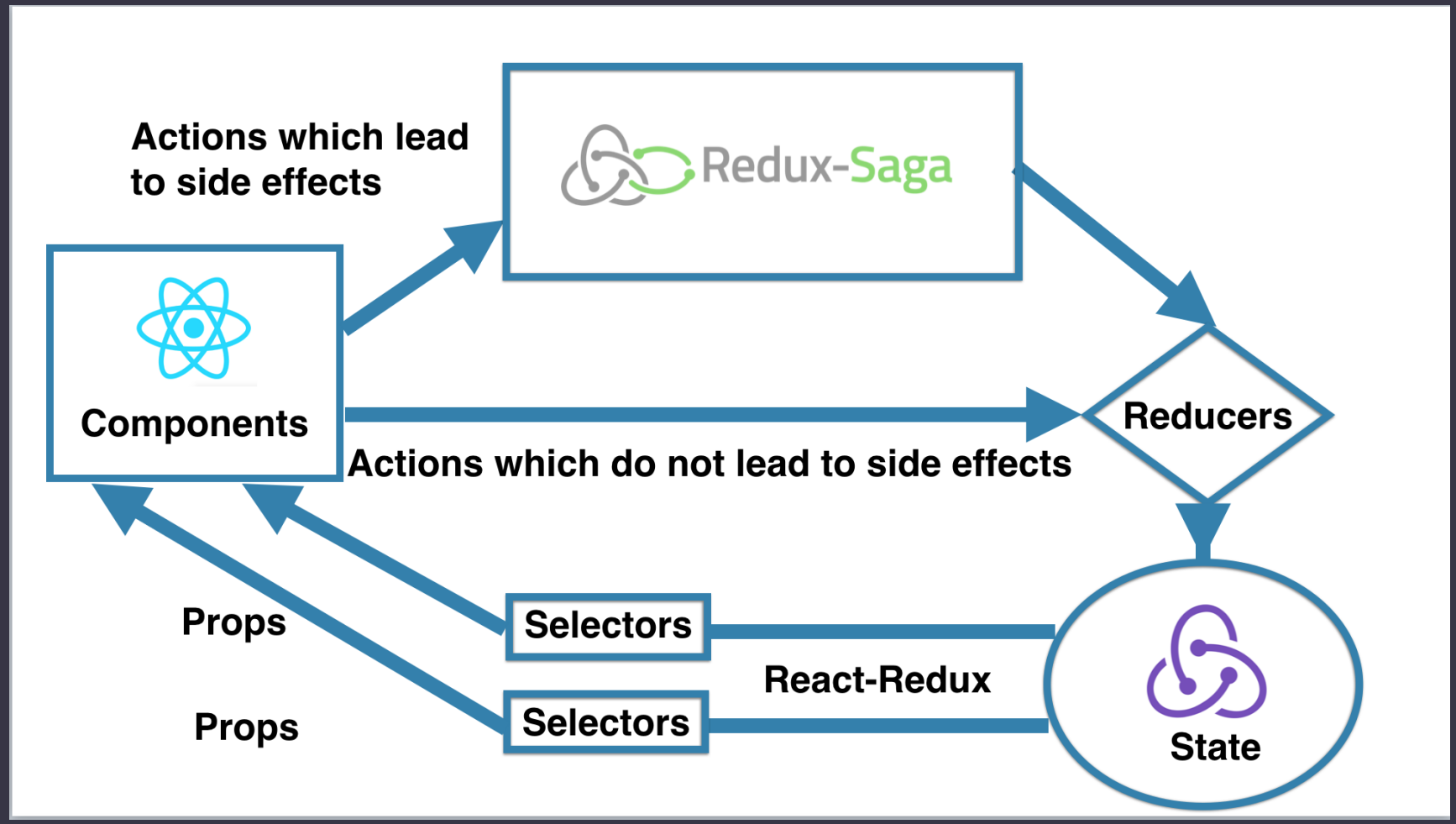
```
import { createStore } from "redux";
import { Provider } from "react-redux";
import reducer from "./reduxapp/reducers/reducers";

let store = createStore(reducer);

ReactDom.render(
  <Provider store={store}>
    <MainReactReduxComponent />
  </Provider>,
  document.getElementById("app")
);
```

SAGA ASYNC CALLS IN REDUX APPS

Redux Using AJAX Calls / Promise Calls



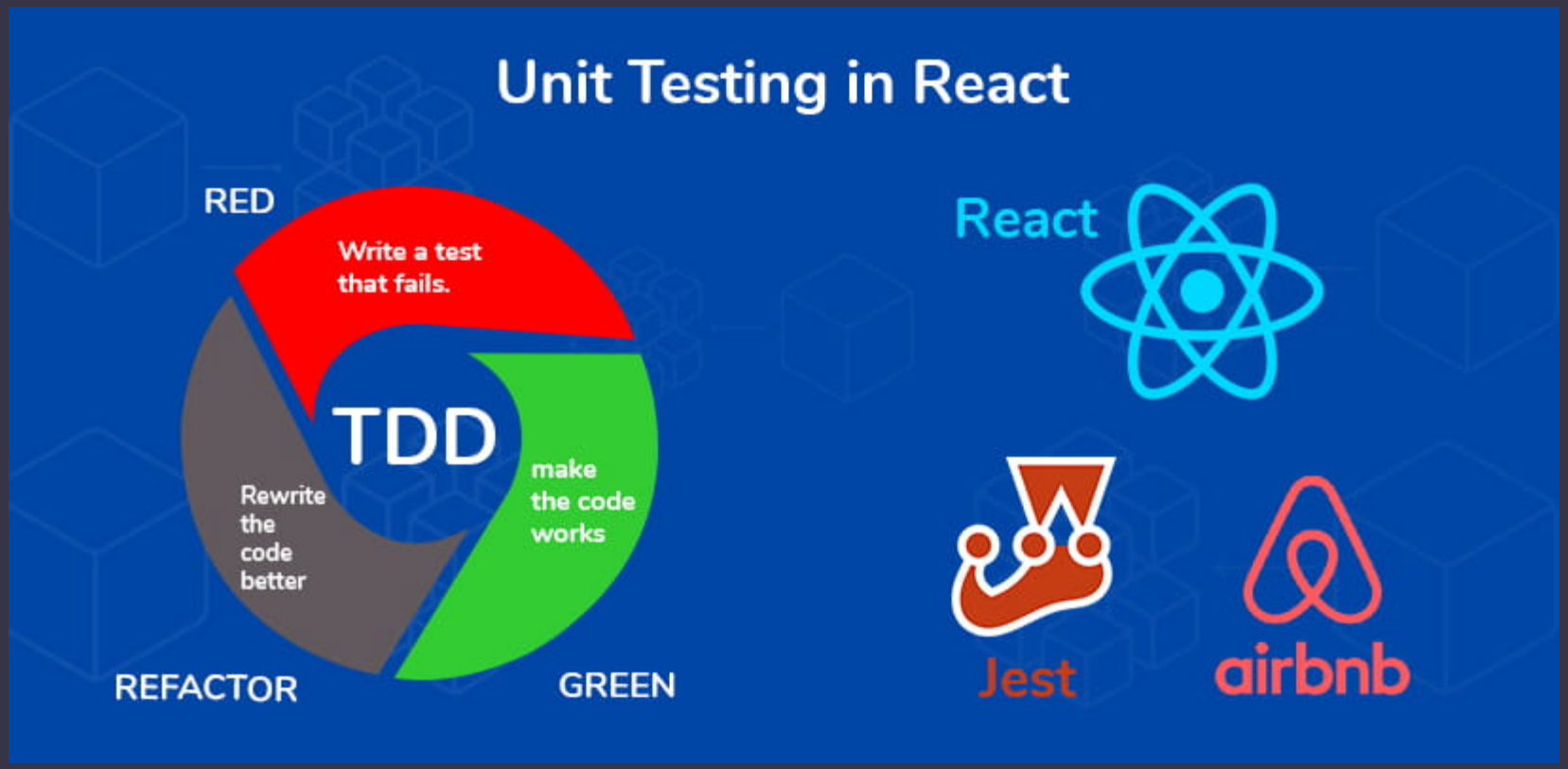
SAGA



https://medium.com/practo-engineering/testing-redux-sagas-with-a-plan-e59124c5d139

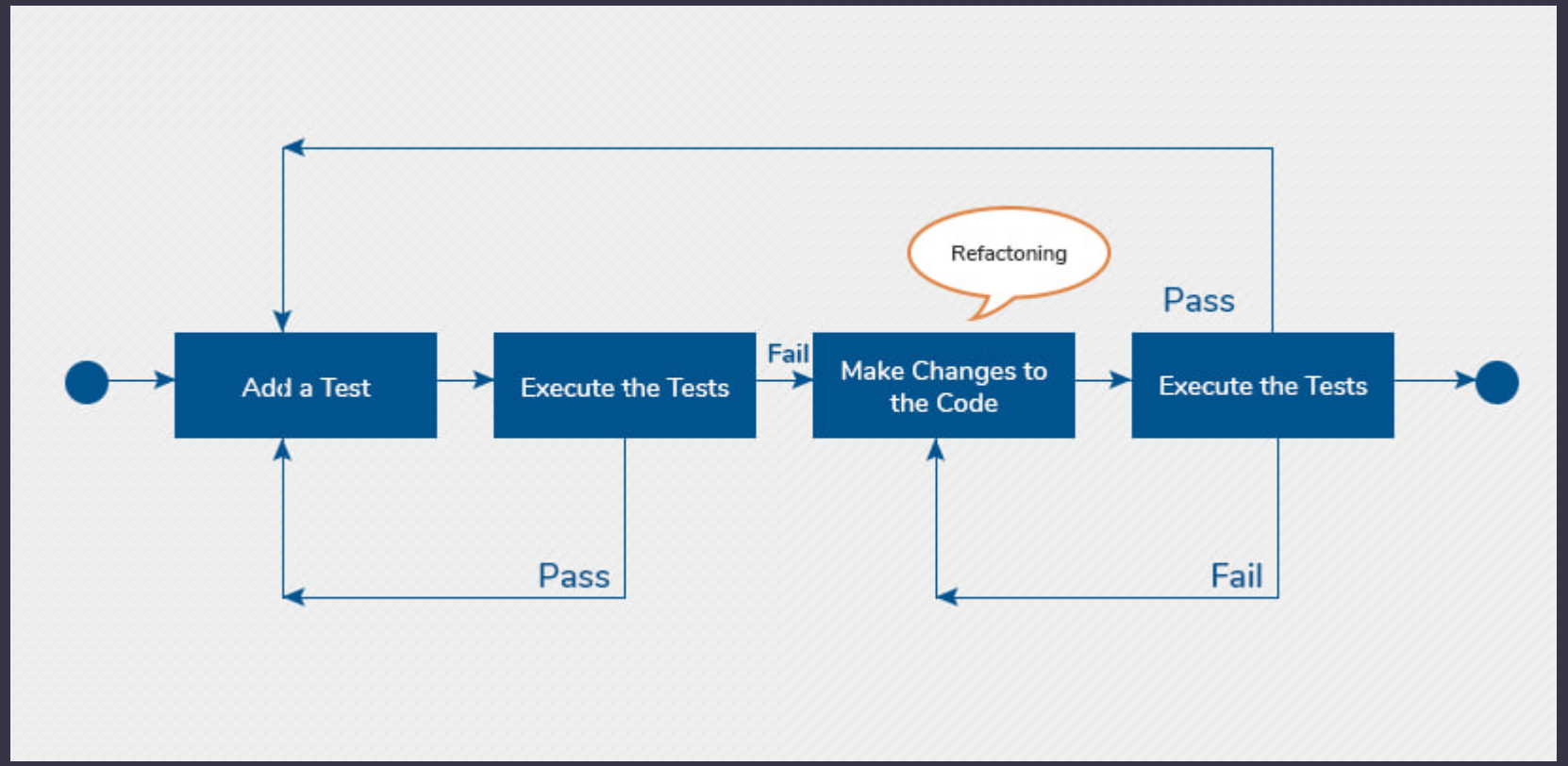
TESTING

TESTING



https://www.vtnetzwelt.com/web/unit-testing-in-reactjs-using-jest-enzyme/

TDD



https://www.vtnetzwelt.com/web/unit-testing-in-reactjs-using-jest-enzyme/

TDD

- Enzyme
- Enzyme is a JavaScript Testing utility for React that makes it easier to test your React Components' output. You can also manipulate, traverse, and in some ways simulate runtime given the output.
- Enzyme's API is meant to be intuitive and flexible by mimicking jQuery's API for DOM manipulation and traversal.
- npm i --save-dev enzyme enzyme-adapter-react-16

USING ENZYME

```
"devDependencies": {
  "check-prop-types": "^1.1.2",
  "enzyme": "^3.8.0",
  "enzyme-adapter-react-16": "^1.7.1",
  "husky": "^2.3.0",
  "jest": "^23.6.0",
  "jest-enzyme": "^7.0.1",
  "moxios": "^0.4.0"
}
```

```
import Enzyme from 'enzyme';
import EnzymeAdapter from 'enzyme-adapter-react-16';

Enzyme.configure({
    adapter: new EnzymeAdapter(),
    disableLifecycleMethods: true
});
```

USING ENZYME

- Shallow Rendering
  - Shallow rendering is useful to constrain yourself to testing a component as a unit, and to ensure that your tests aren't indirectly asserting on behavior of child components.
  - This basically renders a single component each time. In other words, Enzyme won’t consider the child elements for the test. Consider situations where you’d like to test the component itself isolated from the others around or inside of it. This render type is useful when you prefer unit testing rather than a full integrated test.

USING ENZYME

- mount
- This is the opposite of shallow, working with the full DOM rendering, which includes all the child elements. It’s ideal for situations where each component interacts intensively with the others—the DOM API.

USING ENZYME

- render
- It renders to static HTML. This includes all the child elements. At the same time, it prevents access to React lifecycle methods, which, in turn, provides less flexibility and functionalities for testing—besides, it’s much faster. It is built on top of Cheerio, a DOM manipulation and traversal API based on jQuery Core for the server. So, you’ll have all the power of jQuery in your hands.

USING ENZYME

```
import checkPropTypes from 'check-prop-types';
import { applyMiddleware, createStore } from 'redux';
import rootReducer from './../src/reducers';
import { middlewares } from './../src/createStore';

export const findByTestAtrr = (component, attr) => {
    const wrapper = component.find(`[data-test='${attr}']`);
    return wrapper;
};

export const checkProps = (component, expectedProps) => {
    const propsErr = checkPropTypes(component.propTypes, expectedProps,
        'props', component.name);
    return propsErr;
};

export const testStore = (initialState) => {
    const createStoreWithMiddleware = applyMiddleware(...middlewares)(createStore);
    return createStoreWithMiddleware(rootReducer, initialState);
};
```

NEXT.JS

SSR IS JUST ANOTHER WORD FOR "ISOMORPHIC EXECUTION"

i.e. code that runs both on server and client.

Opinionated React

- 🤔 React is ever changing
- 🤔 There's no one way to do things in React
- 🤔 People are trying to settle on a standard: React Router, Redux
- 🤔 Next.js offers an opinionated React – a trade off with big benefits

Server-side rendering is hard

- Next handles isomorphic / universal / bipartisan / polyphonic rendering for you
- **Benefits of server rendering:**
- Better SEO
- Faster user experience
- Progressive: works in browsers without JavaScript

Basic structure of Next.js

- Components live in **/components**
- Pages live in **/pages**

Next.JS comes with

- Server-side rendering
- Automatic code splitting
- Prefetch support
- Inline critical CSS
- Built in routing
- *Production ready*

```
"scripts": {
"dev": "next",
"build": "next
build", "export":
"next export",
"start": "next
start", },
```

REACT ROUTING

REACT ROUTER

- Single Page Application using React.js application
- Supported thorugh
  - React-router package
  - react-router-dom package

```
import { Route, Link, Switch, Redirect } from 'react-router-dom';
```

REACT ROUTER

```
<td><Link to="/">Home</Link></td>
<td><Link to="/messages">Messages</Link> </td>
<td><Link to="/about">About</Link></td>
```

```
<Switch>
  <Route exact path="/" component={Home} />
  <Route path="/messages" component={Messages} />
  <Route path="/about" component={About} />
  <Redirect to="/" />
</Switch>
```

REACT ROUTE

- Parameterized Routing

```
const Message = ({ match }) => (
  <h3> Hello Link with ID { match.params.id } </h3>
);

  <ul>
  {
      [...Array(10).keys()].map(n => {
          return <li key={n}>
                    <Link to={`${match.url}/${n+1}`}>
                      Hello Link {n+1}
                    </Link>
                  </li>;
      })
  }
  </ul>
```

REACT ROUTER

- Route Parameter based Switch

```
<Switch>
  <Route path={`${match.url}/:id(\\d+)`} component={Message} />
  <Route
    path={match.url}
    render={() => <h3>Please select a message</h3>}
  />
</Switch>
```

REACT.JS ADD-ONS

REACT.JS ADD-ONS

- React.js Unit Testing Tools
  - React-unit
  - Jest
  - Enzyme
- GraphQL

GRAPHQL

HTTPS://WWW.GRAPHQL.COM/

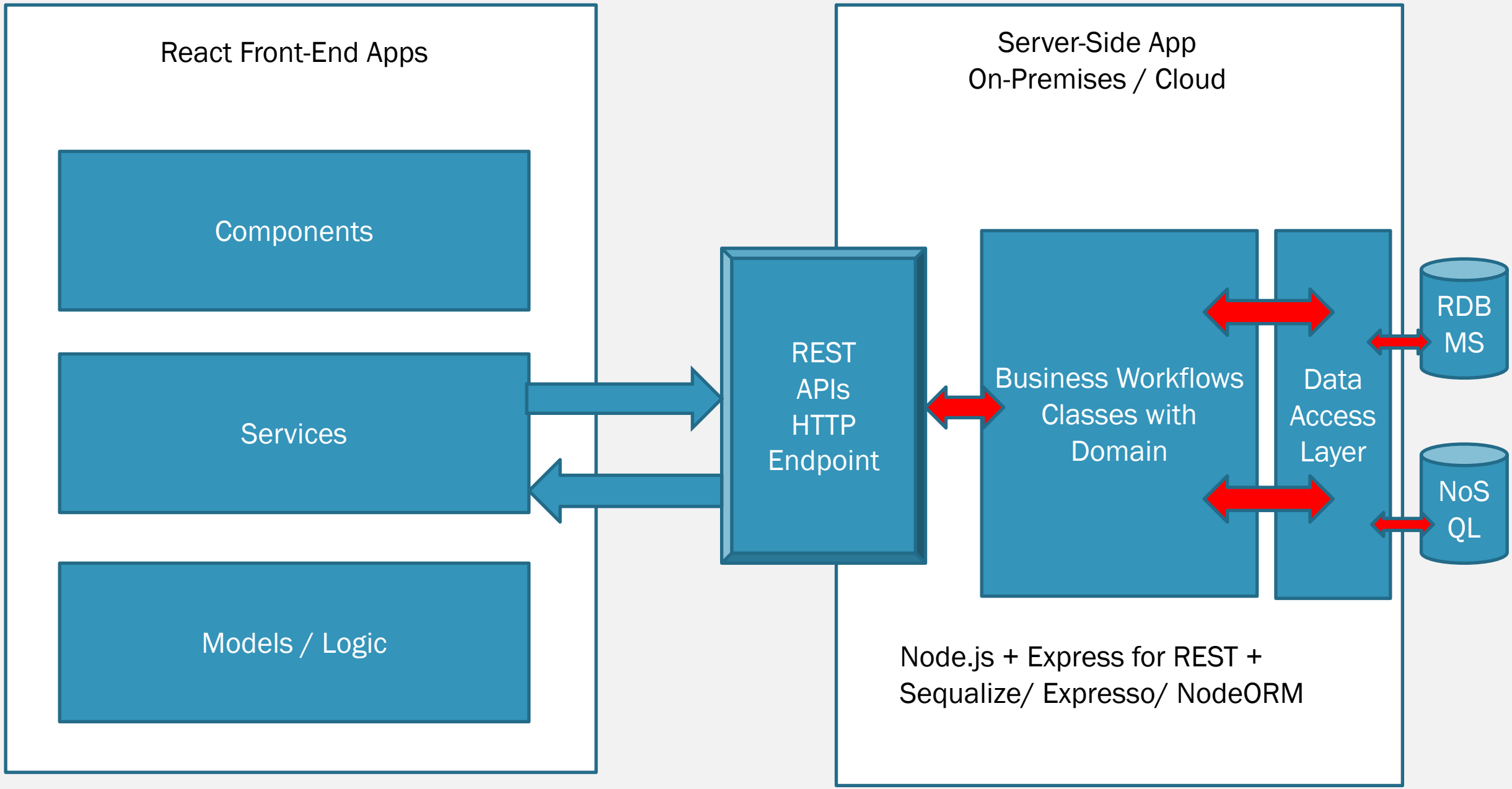
- GraphQL is an open spec for a flexible API layer. Put GraphQL over your existing backends to build products faster than ever before.
- **Faster frontend development**
  - Iterate quickly on apps without waiting for new backend endpoints. Simplify data fetching and management code by getting the data in the shape you need.
- **Use your existing data**
  - You can use GraphQL on top of your existing infrastructure: REST, SOAP, existing databases, or anything else. Organize your data into a clean, unified API and query it all at once.
- **Fewer bytes and roundtrips**
  - Make your apps more responsive than ever before by only loading the data you're actually using, and reduce the number of roundtrips to fetch all of the resources for a particular view.

REACT.JS
THERE IS SOMETHING ABOUT TESTS

- Enzyme is an open source JavaScript testing utility by Airbnb that makes it fun and easy to write tests for React. In this article, we will be going through writing tests for React using Enzyme and Jest.
- Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.
- Enzyme's API is meant to be intuitive and flexible by mimicking jQuery's API for DOM manipulation and traversal.

- Jest is a fast JavaScript testing utility by Facebook that enables you to get started with testing your JavaScript code with zero configuration.

- This means that you can easily perform tasks like code-coverage by simply passing --coverage option when running your tests.
- "enzyme": "^3.8.0",
- "enzyme-adapter-react-16": "^1.9.1"


React Front-End Apps
Components
Services
Models / Logic
REST
APIs
HTTP
Endpoint
Server-Side App
On-Premises / Cloud
Business Workflows
Classes with
Domain
Data
Access
Layer
RDB
MS
NoS
QL
Node.js + Express for REST +
Sequalize/ Expresso/ NodeORM