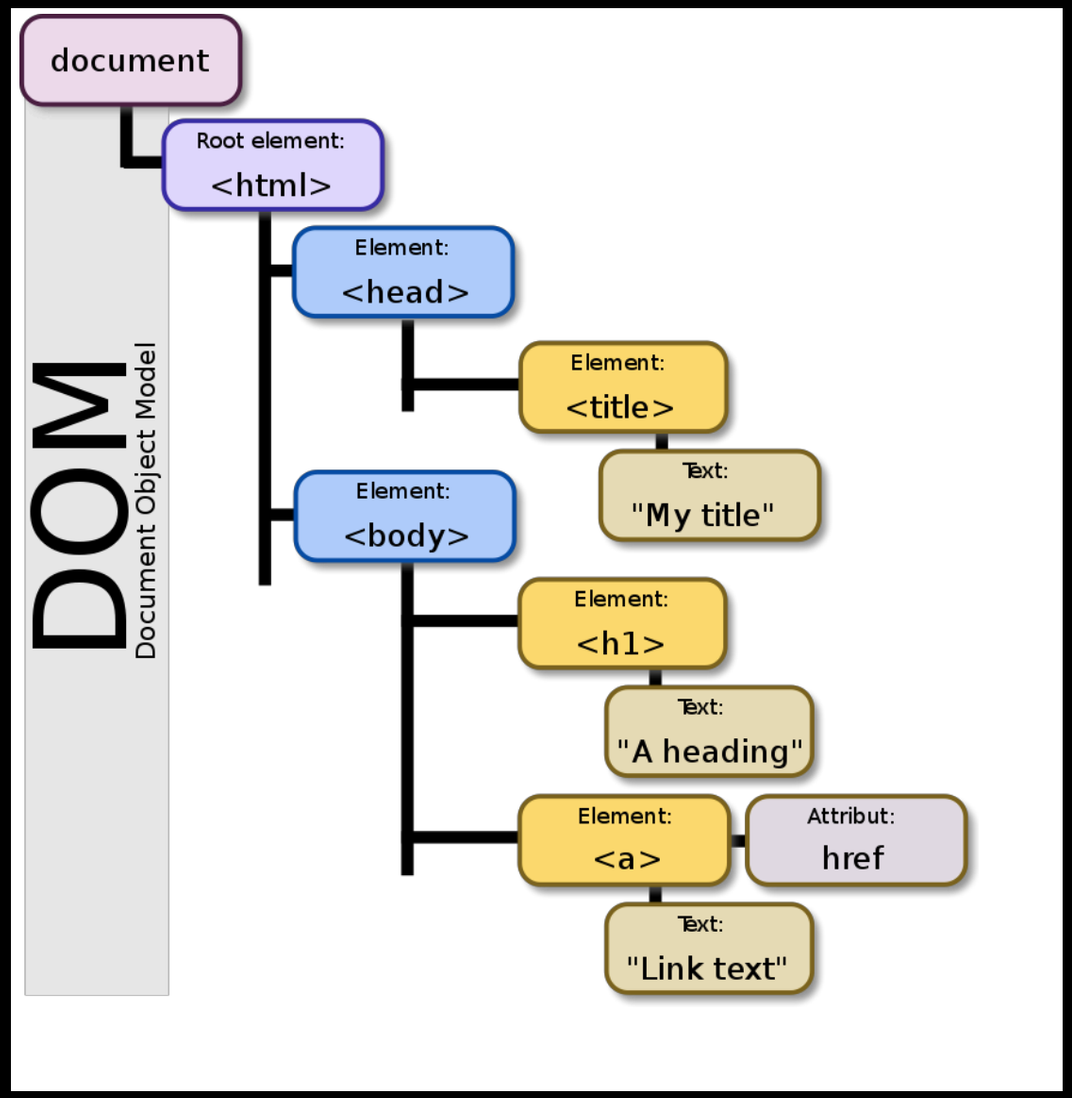
DANIEL GRAHAM PHD

REACT & JSX

DOCUMENT OBJECT MODEL

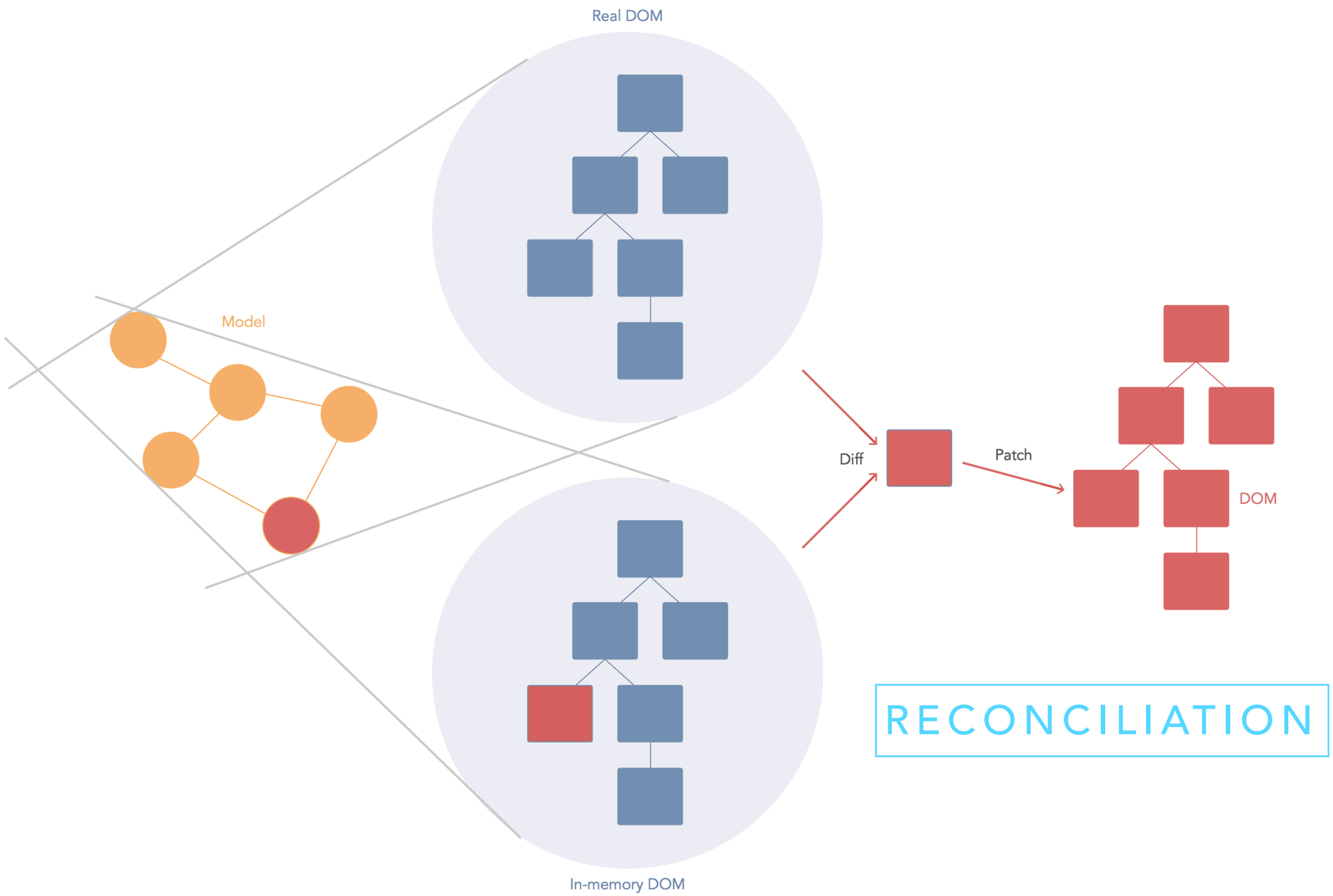



```
<html>
    <head>
        <title>
            My title
        </title>
    </head>
    <body>
        <h1>
            A heading
        </h1>
        <a href="http://www.google.com">
            Link text
        </a>
    </body>
</html>
```

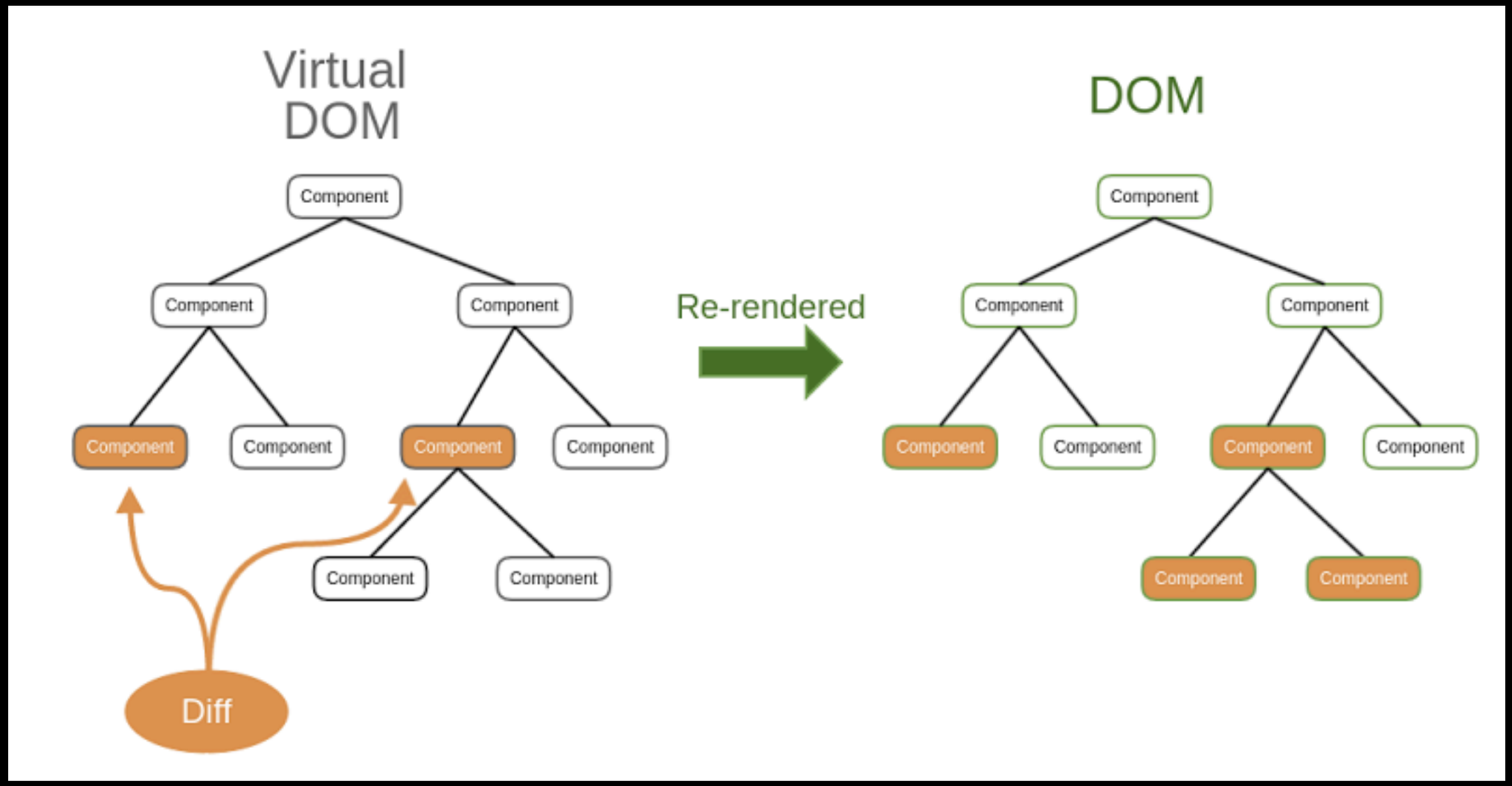
document.getElementById("TZA4S").removeChild(document.getElementById("lga"))

OPEN UP CHROME TO DEMO

REACT AND THE VIRTUAL DOM



REACT AND THE VIRTUAL DOM



https://medium.com/naukri-engineering/naukriengineering-virtual-dom-fa8019c626b

https://reactjs.org/docs/reconciliation.html

VIRTUAL DOM LIFE CYCLE METHODS

HOOKS INTO A COMPONENTS LIFE CYCLE

- shouldComponentUpdate allows the developer to prevent unnecessary re-rendering of a component by returning false if a render is not required.

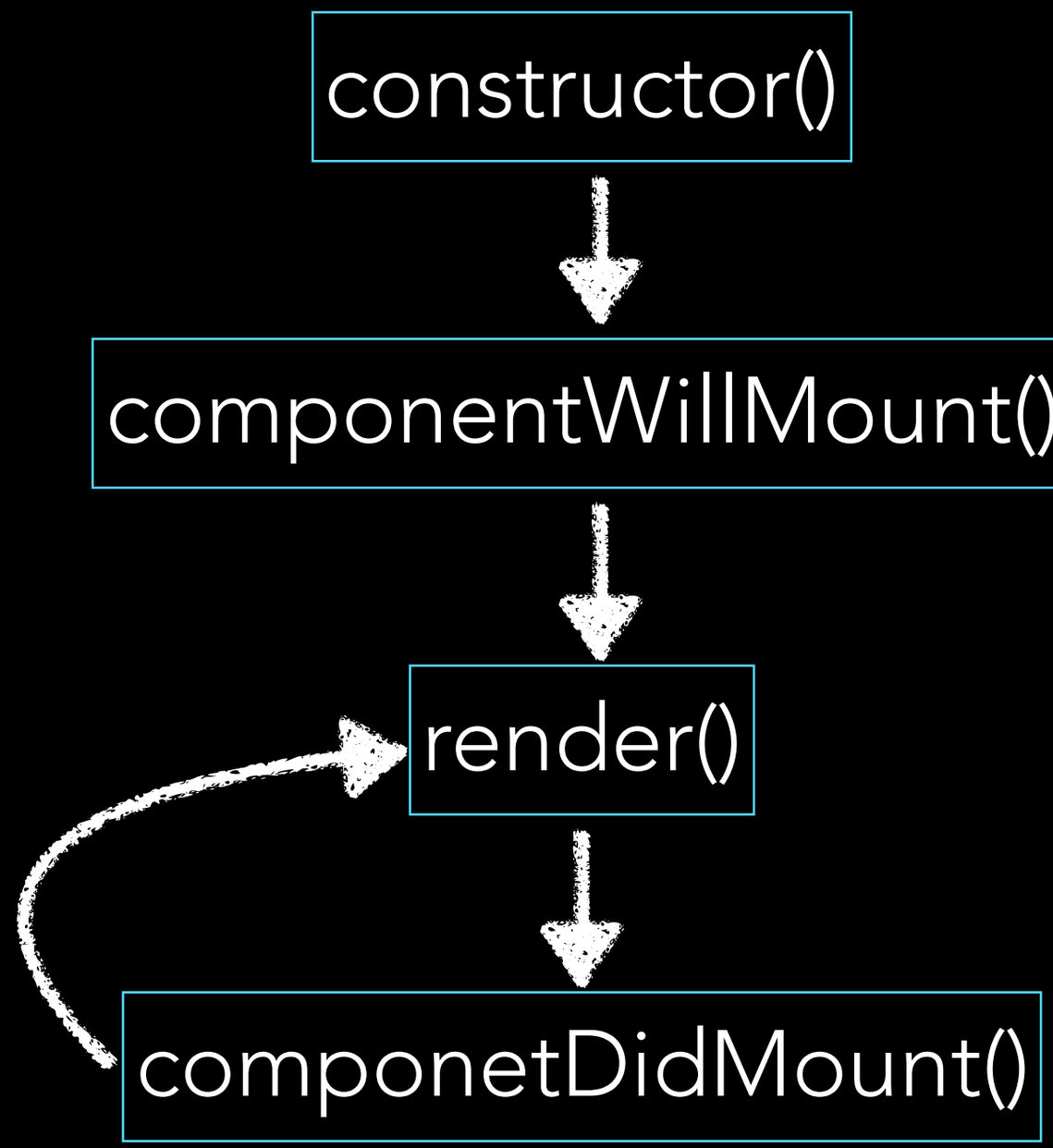
- componentDidMount is called once the component has 'mounted' (the component has been created in the user interface, often by associating it with a DOM node). This is commonly used to trigger data loading from a remote source via an API.

VIRTUAL DOM LIFE CYCLE METHODS

- componentWillUnmount is called immediately before the component is tore down or ‘unmounted'.

- render is the most important lifecycle method and the only required one in any component. It is usually called every time the component's state is updated, reflecting changes in the user interface.

COMPONENT LIFE CYCLE



GETTING SETUP

https://code.visualstudio.com/docs/nodejs/reactjs-tutorial

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

create-react-app my-app

 cd my-app

 npm start
```

MODIFYING INDEX.JS

- Don’t call the APP component lets start simple

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

```
ReactDOM.render(
    <h1>Hello, world!</h1>,
    document.getElementById('root')
  )
```

THIS IS STRANGE

Is it a string?

Is html?

IT IS CALLED JSX

Javascript XML

JSX

LET, VAR AND CONST ARE REQUIRED IN REACT NATIVE

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

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

ReactDOM.render(
    element,
    document.getElementById('root')
  )
```

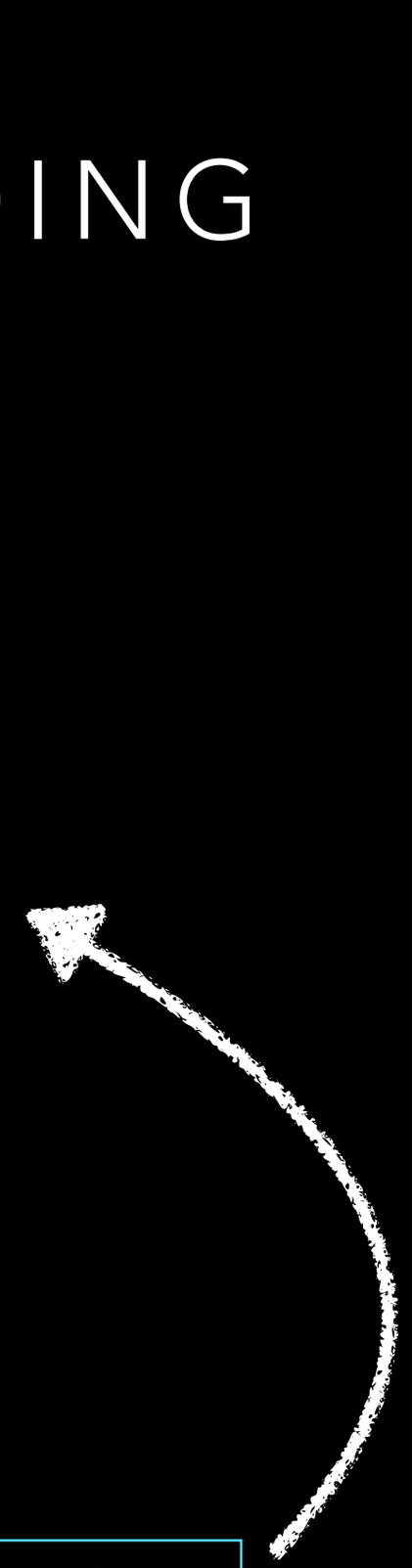
DEMO HOT SWAPS IN CHROME AND VISUAL STUDIO

JSX EXPRESSION EMBEDDING

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

const name = 'DeShea Perez';
const element = <h1>Hello, {name}</h1>;

ReactDOM.render(
  element,
  document.getElementById('root')
)
```



CAN BE ANY VALID JAVASCRIPT EXPRESSION

```
function buildWahoo(grit, grace, name) {
    wahoo.grace = grace
    wahoo.grit = grit
    wahoo.name = name
    return wahoo
  }

  let wahoo = {
    grit: 0,
    grace: 0,
    name: "Annoymous",
    readable: function(){
        return this.grit + " " + this.grace + " "+ this.name
    }
  }

  const element = (
    <h1>
      Hello, {buildWahoo(0.86, 0.9, "DeShea").readable()}!
    </h1>
  )

  ReactDOM.render(
    element,
    document.getElementById('root')
  )
```

JSX IS AN EXPRESSION TOO

```
function getGreeting(user) {
    if (user) {
      return <h1>Hello, {buildWahoo(user)}!</h1>
    }
    return <h1>Hello, Anonymous.</h1>
  }
```

After compilation, JSX expressions become regular JavaScript function calls and evaluate to JavaScript objects. This means that you can use JSX inside of if statements and for loops, assign it to variables, accept it as arguments, and return it from functions:

SPECIFYING ATTRIBUTES WITH JSX

```
let wahoo = {
    grit: 0,
    grace: 0,
    name: "Annoymous",
    avatarUrl: "http://www.w3schools.com/html/img_girl.jpg",
    readable: function(){
        return this.grit + " " + this.grace + " "+ this.name
    }
}

const tabElement = <div tabIndex="0"> </div>
const picture = <img src={wahoo.avatarUrl}></img>
```

DISTINCTION BETWEEN ELEMENTS AND COMPONENT

COMPONENTS ARE MADE UP OF ELEMENTS

COMPONENTS HAVE A RENDER LIFE CYCLE

JSX ELEMENTS CAN CONTAIN CHILDREN

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

JSX COMPILES TO REACT CREATE ELEMENT CALLS

```jsx
const element = (
    <h1 className="greeting">
      Hello, world!
    </h1>
)
```

COMPILES TO THIS

```jsx
const element = React.createElement(
    'h1',
    {className: 'greeting'},
    'Hello, world!'
  )
```

```jsx
const element = React.createElement(
  'h1',
  {className: 'greeting'},
  'Hello, world!'
)
```

CREATES AND OBJECT SIMILAR TO THIS

```jsx
// Note: this structure is simplified
const element = {
  type: 'h1',
  props: {
    className: 'greeting',
    children: 'Hello, world!'
  }
};
```

DEMONSTRATING THE VIRTUAL DOM

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


function tick() {
    const element = (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {new
          Date().toLocaleTimeString()}.</h2>
      </div>
    )
    ReactDOM.render(element,
     document.getElementById('root'))
  }

  setInterval(tick, 1000);
```

COMPONENTS AND PROPERTIES

```
function Clock() {
    let clockElement = <div>
                        <h1>This is a Clock</h1>
                        <h1>The Time is Now :
                       {new Date().toLocaleTimeString()}</h1>
                    </div>
    return  clockElement
  }
```

WE CAN ALSO MAKE THIS SELF CLOSING

```
<Clock/>
```

```
function tick() {
    const element = (
      <Clock></Clock>
    );
    ReactDOM.render(element, document.getElementById('root'))
  }

  setInterval(tick, 1000);
```

WHAT IF WE HAD DIFFERENT TYPE OF CLOCKS

HOW COULD WE CUSTOMIZE THEM

THIS IS WHERE

PROPERTIES COME IN

ASSIGNING PROPERTIES

WHEN DEFINING FUNCTION COMPONENT MUST US CAPITAL

```
function Clock(props) {
    let clockElement = <div>
                        <h1>This is a { props.type} Clock</h1>
                        <h1>The Time is Now : {props.time}</h1>
                    </div>
    return  clockElement
}
```

USING PROPERTIES

```
function tick() {
    const element = (
    <div>
        <Clock type={"English"} time ={new
                                Date().toLocaleTimeString()} />
        <Clock type={"Arabic"} time ={new
                            Date().toLocaleTimeString('ar-EG')} />
      </div>
    );
    ReactDOM.render(element, document.getElementById('root'))
  }

  setInterval(tick, 1000);
```

PROPERTIES ARE READ ONLY

COMPONENTS MUST BE PURE FUNCTION

```
//Pure function does not modify its parameters
function sum(a, b) {
    return a + b;
  }

//Not a pure function because it modifies its parameters
function withdraw(account, amount) {
    account.total -= amount;
  }
```

THE MEANS THAT COMPONENTS CAN NOT MODIFY THEIR PROPERTIES

CAN'T OVERRIDE PROPERTIES

```
function Clock(props) {
    props.type = "Override"
    let clockElement = <div>
                        <h1>This is a { props.type} Clock</h1>
                        <h1>The Time is Now : {props.time}</h1>
                    </div>
    return  clockElement
}
```

TypeError: Cannot assign to read only property 'type' of object '#<Object>'

Clock
src/index.js:15

```
  12 |   }
  13 |
  14 | function Clock(props) {
> 15 |     props.type = "Override"
  16 |     let clockElement = <div>
  17 |                         <h1>This is a { props.type} Clock</h1>
  18 |                         <h1>The Time is Now : {props.time}</h1>
```

View compiled

NEW SYNTAX FOR COMPONENTS

```jsx
function Welcome(props) {
    return <h1>Hello, {props.name}</h1>;
}
```

```jsx
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

```jsx
const element = <Welcome name="Sara" />;
  ReactDOM.render(
    element,
    document.getElementById('root')
  )
```

```
  class Clock extends React.Component{
   render(){
        let clockElement = <div>
                          <h1>This is a { this.props.type} Clock</h1>
                          <h1>The Time is Now : {this.props.time}</h1>
                      </div>
        return  clockElement
    }
  }
```

LET'S READ THAT DEFAULT APP JS FILE

PULL IT ALL TOGETHER

```jsx
import React, { Component } from 'react';
import logo from './logo.svg';
import './App.css';

class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
export default App;
```

```
class App extends React.Component {
    constructor(props) {
      super(props)
      this.state = {date: new Date()}
    }
     componentDidMount() {
      this.timerID = setInterval(
        () => this.tick(),
        1000
      )
    }
    componentWillUnmount() {
      clearInterval(this.timerID);
    }
    tick() {
      this.setState({
        date: new Date()
      })
    }
    render() {
      return (
        <div>
          <h1>Hello, world!</h1>
          <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
        </div>
      )
    }
  }
```

ASYNCHRONOUSLY CALLS RENDER

LET'S BUILD A STOCK REPORTING REACT APP

https://iextrading.com/developer/docs/#price

https://api.iextrading.com/1.0/stock/aapl/price

JUST SOME NOTES ON STATE

Since react components are treated as pure Functions. If want update property dynamically Use instance variables or special variable call state.

LET'S MODIFY THE TICK

```
tick() {
    this.setState({
       date: new Date()
     });
   }
```

LET'S DO THE FETCH HERE

FETCH IS A PROMISE

```
tick() {
     let fetchPromise  = fetch("https://api.iextrading.com/
1.0/stock/aapl/price")
     fetchPromise.then((resultPromise)=>{
       return resultPromise.text()
     }).then( (result)=>{
         this.setState({
           price: result
         })
       }
     )
   }
```

PROMISE CHAINING

USING THE AWAIT APPROACH

```
    async tick() {
      let response = await fetch("https://api.iextrading.com/1.0/
stock/aapl/price")
      let responseText = await response.text();
          this.setState({
            price: responseText
          })
        }
```

MORE DETAILS ON STATE STATE

```
// Wrong
this.setState({
  counter: this.state.counter + this.props.increment,
});
```

```
// Correct
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

```
  constructor(props) {
    super(props);
    this.state = {
      posts: [],
      comments: []
    }
  }

componentDidMount() {
    fetchPosts().then(response => {
      this.setState({
        posts: response.posts
      });
    });

    fetchComments().then(response => {
      this.setState({
        comments: response.comments
      });
    });
  }
```

LET'S SWITCH OVER TO OUR NEW ENVIRONMENT

HTTPS://SNACK.EXPO.IO/

I will post snacks on course website

https://expo.io/learn

Setting up local dev Environment

REACT NATIVE

```
export default class App extends React.Component {
  render() {
    let pic = {
      uri: 'http://www.learnglc.com/wp-content/uploads/2014/01/
UVA.jpg'
    }
    return (
      <Image source={pic} style={{flex:1}}/>
    )
  }
}
```

```
import * as React from 'react'
import { Image } from 'react-native'
import SplashScreen from './components/SplashScreen'

export default class App extends React.Component {
  render(){
    return(
      <SplashScreen/>
      )
  }
}
```

```
import { Image, Text } from 'react-native'
import * as React from 'react'
import Stock from './Stock'
export default class SplashScreen extends React.Component{
  constructor(props){
    super(props)
    this.state ={
      loading: true
    }
  }


  render() {
    if(this.state.loading){
        let pic = {
          uri: 'http://www.learnglc.com/wp-content/uploads/2014/01/UVA.jpg'
        }
        return (<Image source={pic} style={{flex:1}}/>)
      }
    }
    return(<Stock/>)

}
```