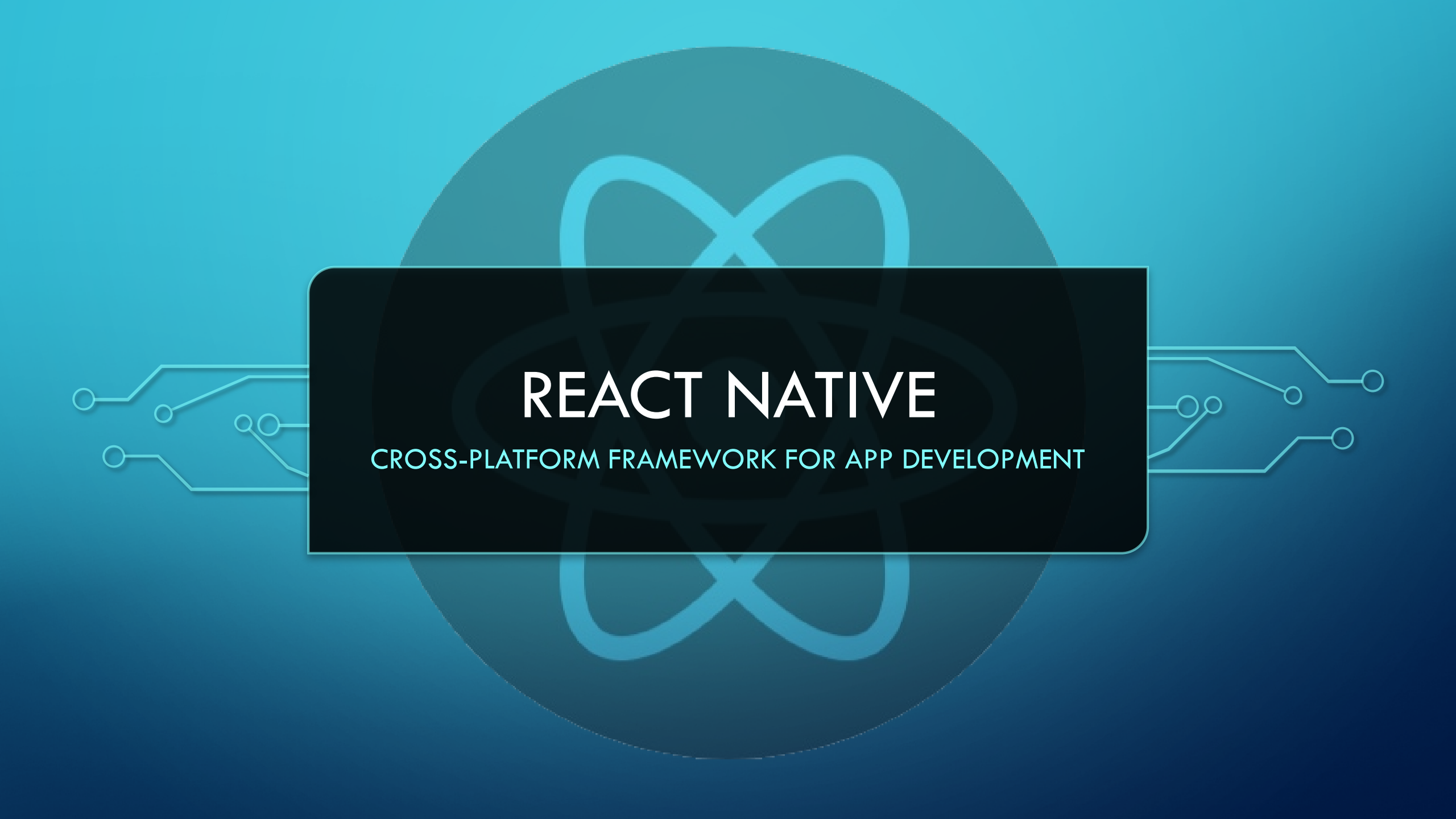

REACT NATIVE
CROSS-PLATFORM FRAMEWORK FOR APP DEVELOPMENT

INTRODUCTION

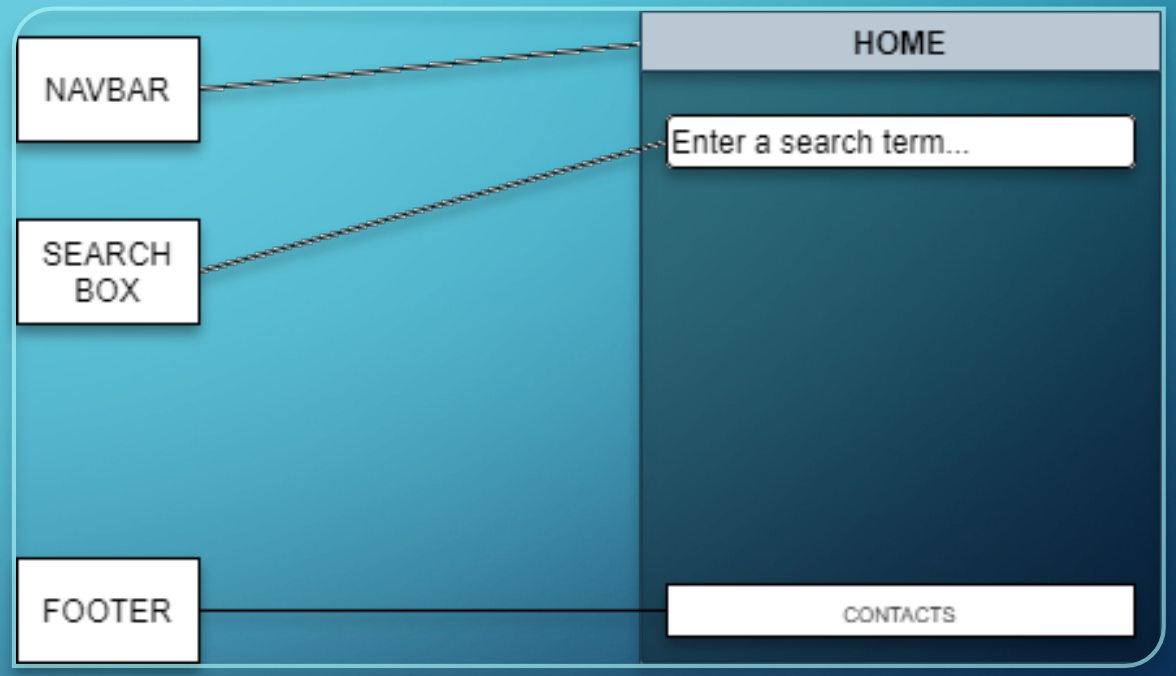
- React Native is a cross-platform open-source framework that implements Facebook's ReactJS (React.js) library
- Language: JavaScript (JSX – JavaScript and XML)
- It can be used for developing Android and iOS mobile applications

REACT OVERVIEW

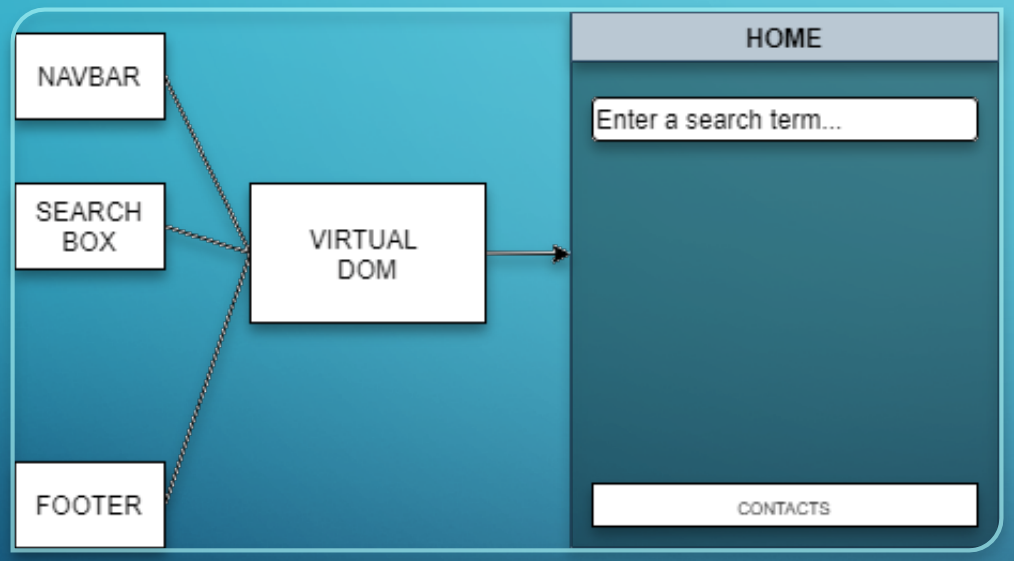
- Framework for creating user interfaces developed by Facebook
- Component-based
- Uses a virtual DOM that makes it very fast

COMPONENT BASED

- Every different part of the application is a component
- Every component is indipendent and reusable
- They are JavaScript functions that accept input data (props) and return react elements



VIRTUAL DOM (VDOM)



- React uses a JavaScript representation of the DOM, the VIRTUAL DOM
- The actual DOM is based on it
- Everytime that something changes, React creates a new VDOM and compares it with the old one. Than renders only what has been modified

PROPS AND STATE

- Props (properties) are used to configure a component when it renders, to customize a component
- Props are ‘read only’
- They’re used to influence components in a top-down approach (Parent to Child)

```
class Board extends React.Component {
  renderSquare(i) {
    return <Square value={i} />;
  }
}
```

PROPS AND STATE

- State is used to keep track of any component data that is expected to change over time due to a user action, network response, etc
- It can be used to influence components in a bottom-up approach (Child to Parent)

```
class Square extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        value: null,
      };
    }

    render() {
      return (
        <button
          className="square"
          onClick={() => this.setState({value: 'X'})}
        >
          {this.state.value}
        </button>
      );
    }
  }
```

STATE - DETAILS

```
// Wrong
this.state.greetings = 'Hello';

//Right
this.setState({greetings: 'Hello'});
```

- To modify the state, it should be used setState()

```
const [value, setValue] = useState(null);

...

<button
    className="square"
    onClick={() => setValue('X')};
> ...
```

- React may merge multiple calls to setState() into one but these updates may be asynchronous!
- Another way to manage the state is to use the 'Hooks' (React 16.8)

REDUX

- Redux library can be used to make the state update synchronous
- To connect Redux to any application, we need to create a *reducer* and an *action:*
  - *An action is an object (with a type and an optional payload) that represents the will to change the state*
  - *A reducer is a function that takes the previous state and an action as arguments and returns a new state*

- Redux also introduces two functions: mapStateToProps (used to make state accessible to the screens that implements it) and mapDispatchToProps (to access to defined actions)

REDUCER

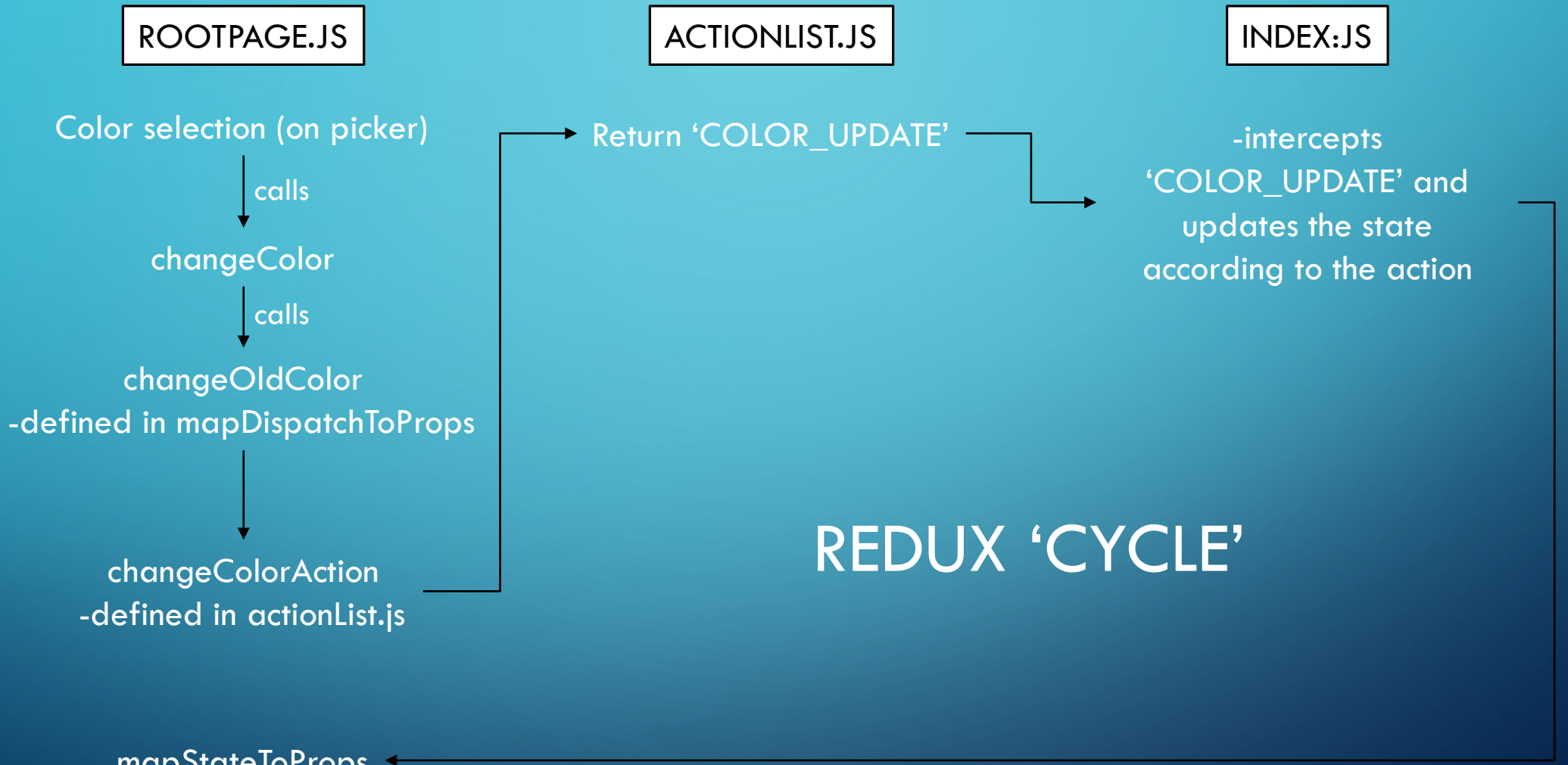
```
  const initialState = {
    oldColor: "#FF7700",
    redValue: 255,
    greenValue: 119,
    blueValue: 0,
};

const redValue = (state = initialState.redValue, action) => {
    switch(action.type) {
        case 'RED_UPDATE':
            return action.value;
        case 'COLOR_UPDATE':
            return tinycolor(action.oldColor).toRgb().r;
        default:
            return state;
    }
};
```

ACTION

```
export const changeColorAction = oldColor => ({
  type: 'COLOR_UPDATE',
  oldColor
});
```

REDUX 'CYCLE'


ROOTPAGE.JS
ACTIONLIST.JS
INDEX:JS
Color selection (on picker)
calls
changeColor
calls
changeOldColor
-defined in mapDispatchToProps
changeColorAction
-defined in actionList.js
Return 'COLOR_UPDATE'
-intercepts 'COLOR_UPDATE' and updates the state according to the action
mapStateToProps

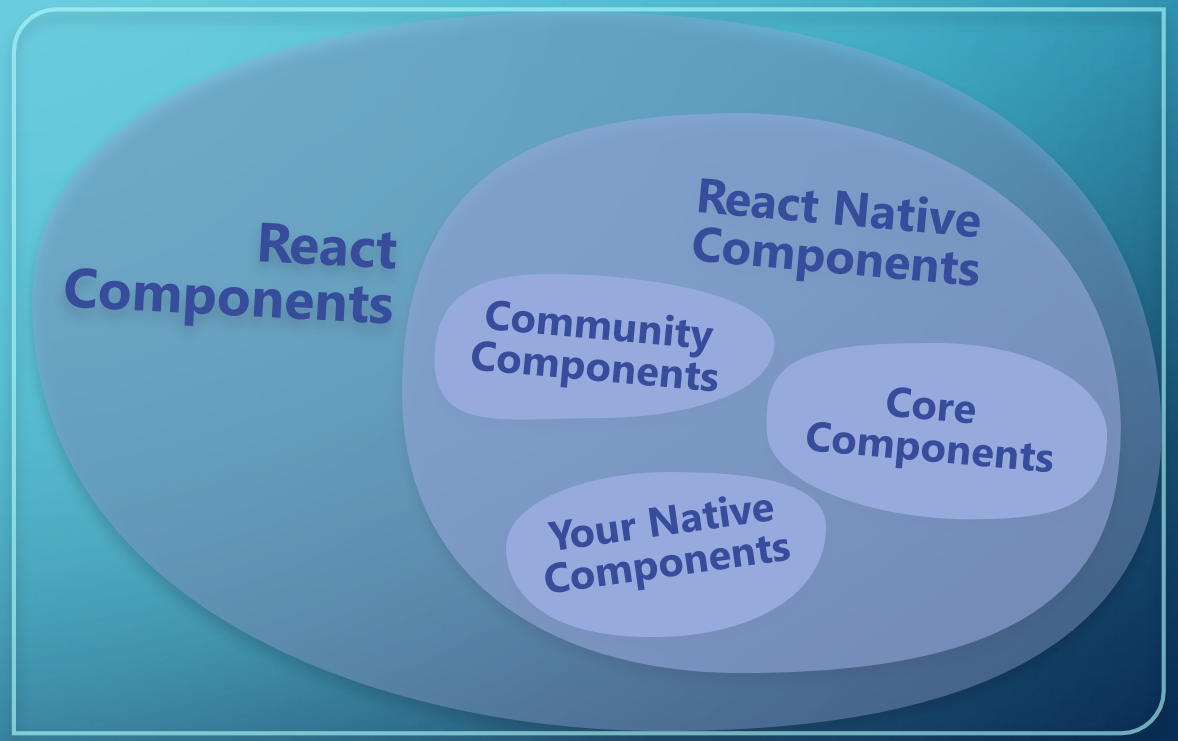
BACK TO REACT NATIVE

FEATURES

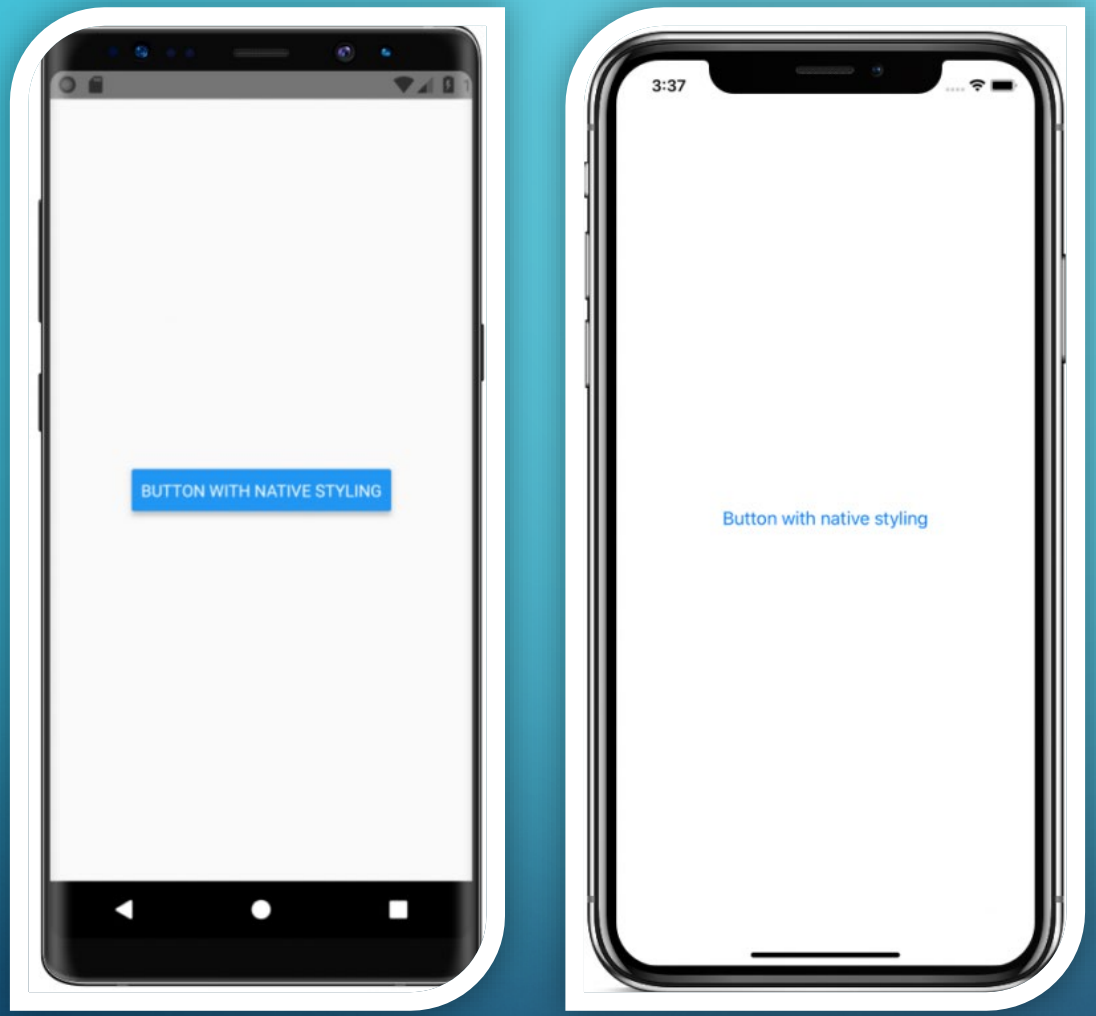
- Native look and feel
- Allows to create and use personalized components
- Active community
- Hot Reloading
- Possibility of integrating React Native with an already existing mobile application

COMPONENTS

- Thank to a bridge, all these components are render as native component according to the device's operating system
- Components could be native or personalized, created by the React Native community or by yourself



EXAMPLE: BUTTON




```
<Button
  title="Button with native styling"
/>
```

The same code is render in two different ways according to the OS of the device

CLASS COMPONENT EXAMPLE

```
import React, { Component } from 'react';
import { Text } from 'react-native';

class Cat extends Component {
    render() {
        return (
            <Text>Hello I am your cat! </Text>
        );
    }
}

export default Cat;
```

- Requires to extend from React.Component
- Creates a render function that returns a React element
- Permits to use setState() and lifecycle hooks
- Known as ‘stateful’ component

FUNCTIONAL COMPONENT EXAMPLE

- Plain JavaScript function that accepts props as argument and returns a React element
- Doesn't allow to use setState() and lifecycle hooks
- Is a best practice
- Known as 'stateless' component

```
import React from 'react';
import { Text } from 'react-native';

const Cat = () => {
    return (
        <Text>Hello I am your cat! </Text>
    );
}

export default Cat;
```

A REACT NATIVE PROJECT

SETTING UP THE ENVIRONMENT

- We need NodeJS
- It can be used either Expo CLI (simpler) or React Native CLI

```
1) npm install -g expo-cli

2) expo init AwesomeProject

3) cd AwesomeProject
   npm start (or expo start)
```

- With Expo CLI we need the Expo Client (Expo Go) in our device

EXPO TEMPLATE

```
import { StatusBar } from 'expo-status-bar';
import React from 'react';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

- This is the default template created with Expo init (App.js file)
- It's a functional component (App) that contains a JSX template
- View and Text are (child) components as well
- To run the app, type 'expo start' or 'npm start', it will open a new developer tools panel

- *View* component is similar to div component in HTML
- *Text* component is essential to insert some text
- *StyleSheet* is used style components, we use style property
- Styles could be defined at the bottom of the file or (better) in a separated file, using rules similar to css ones
- Different rules could be divided using a keyword

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

BIBLIOGRAPHY

- https://reactnative.dev/
- https://it.reactjs.org/
- https://www.digitalocean.com/community/tutorials/react-react-native-redux
- https://djoech.medium.com/functional-vs-class-components-in-react-231e3fbd7108