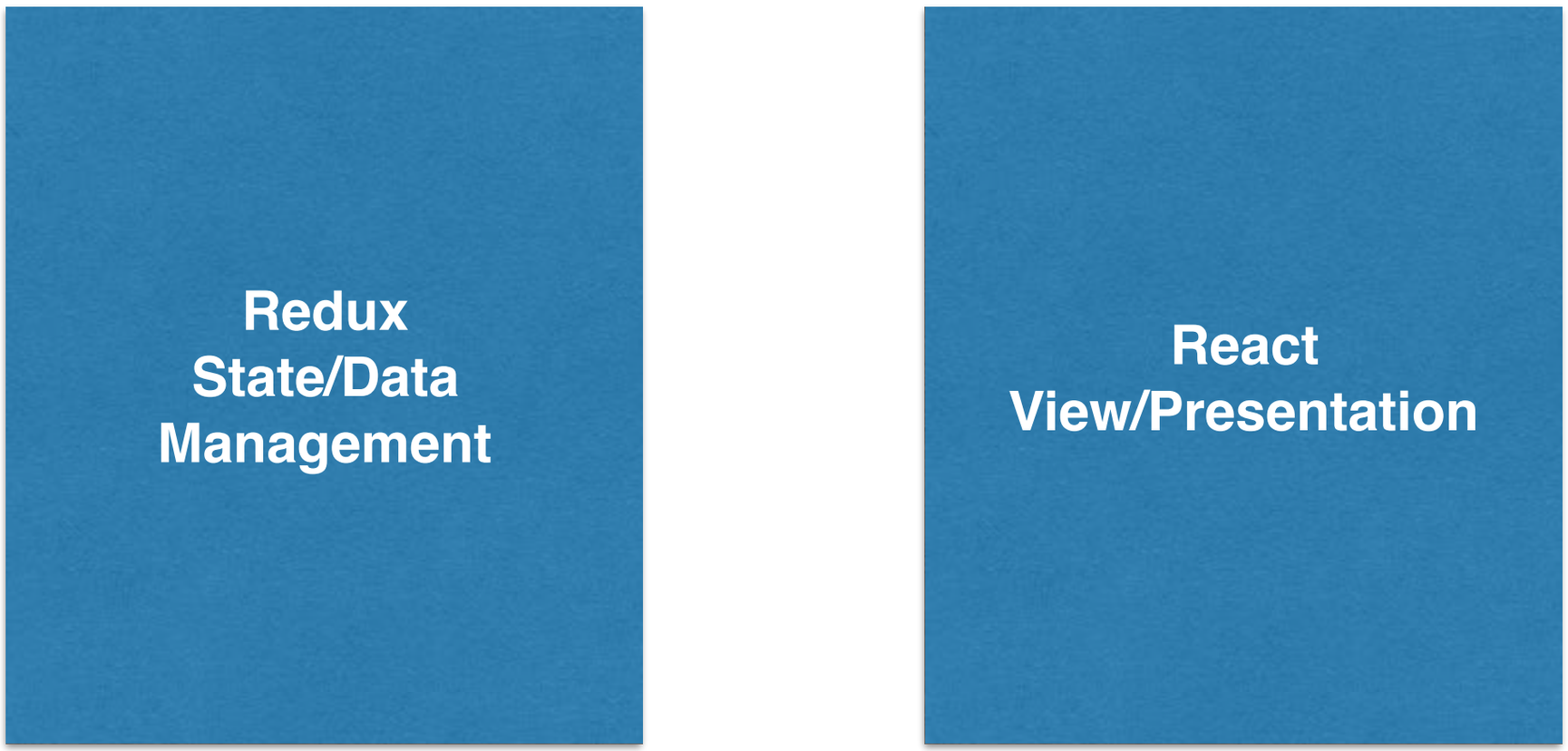
Redux

Gopalakrishnan Subramani
www.nodesense.ai
gs@nodesense.ai
+91 9886991146

NODESENSE.AI

Introduction



Redux is a **Predictable State Container**

What is in React

- Component
- Render
- State
- Props
- JSX

NODESENSE.AI

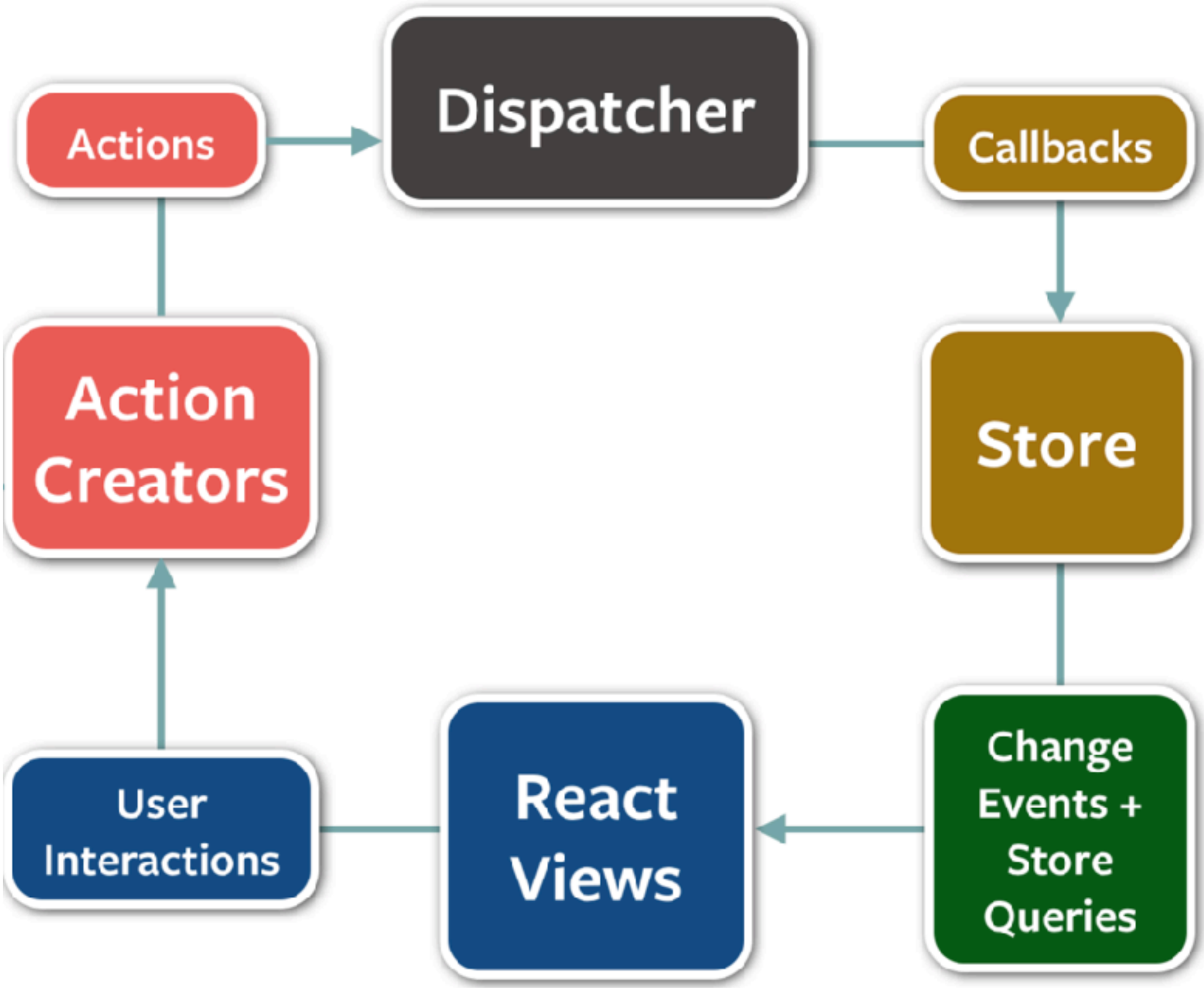
What is in Flux?

- Flux is an Architecture for larger React Application
- Redux is an implementation of Flux

- Action
- Action Creator
- Dispatch
- Callback/Reducer
- Store

NODESENSE.AI

FLUX



NOBLESENSE.AI

Flux Implementation

Redux

1. Functional Style
2. Pure Function
3. Immutable
4. Publish/Subscribe

5.2 Millions Downloads per month

Mobx

1. OOP
2. Classes
3. Mutable collection
4. Publish/Subscribe

0.75 Millions Downloads per month

Redux

- Redux is a framework for managing the state for a web application,
- React components render that state
- A single data store contains the state for your app
- Your application emits an action, that defines something that just happened that will affect the state
- Reducers specify how to change the state when the action is received
- Hot reloading of code changes
- State changes can be tracked, and replayed

NODESENSE.AI

Redux

- Redux is a Flux implementation
- Minimal APIs
- Holds application state
- Allows access to state via getState();
- Allows state to be updated via dispatch(action);
- Registers listeners via subscribe(listener) (to update view)
- Handles unregistering of listeners via the function returned by subscribe(listener)

NODESENSE.AI

Functional Programming

- Keeping React Components as View only
- De-couple states from React Component
- Calling React view with same input must produce same output (mean, no states maintained at React)
- Predictable outcome, easy to test

Without Redux

With Redux

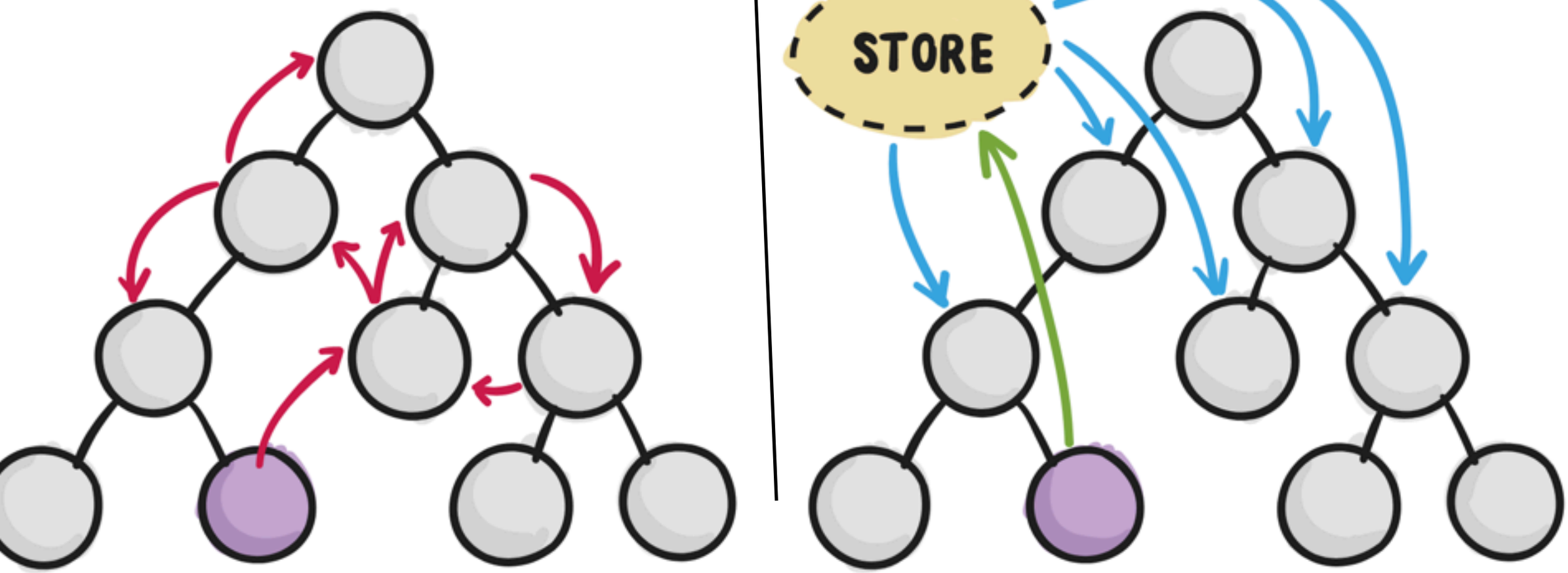


Image credit: Internet

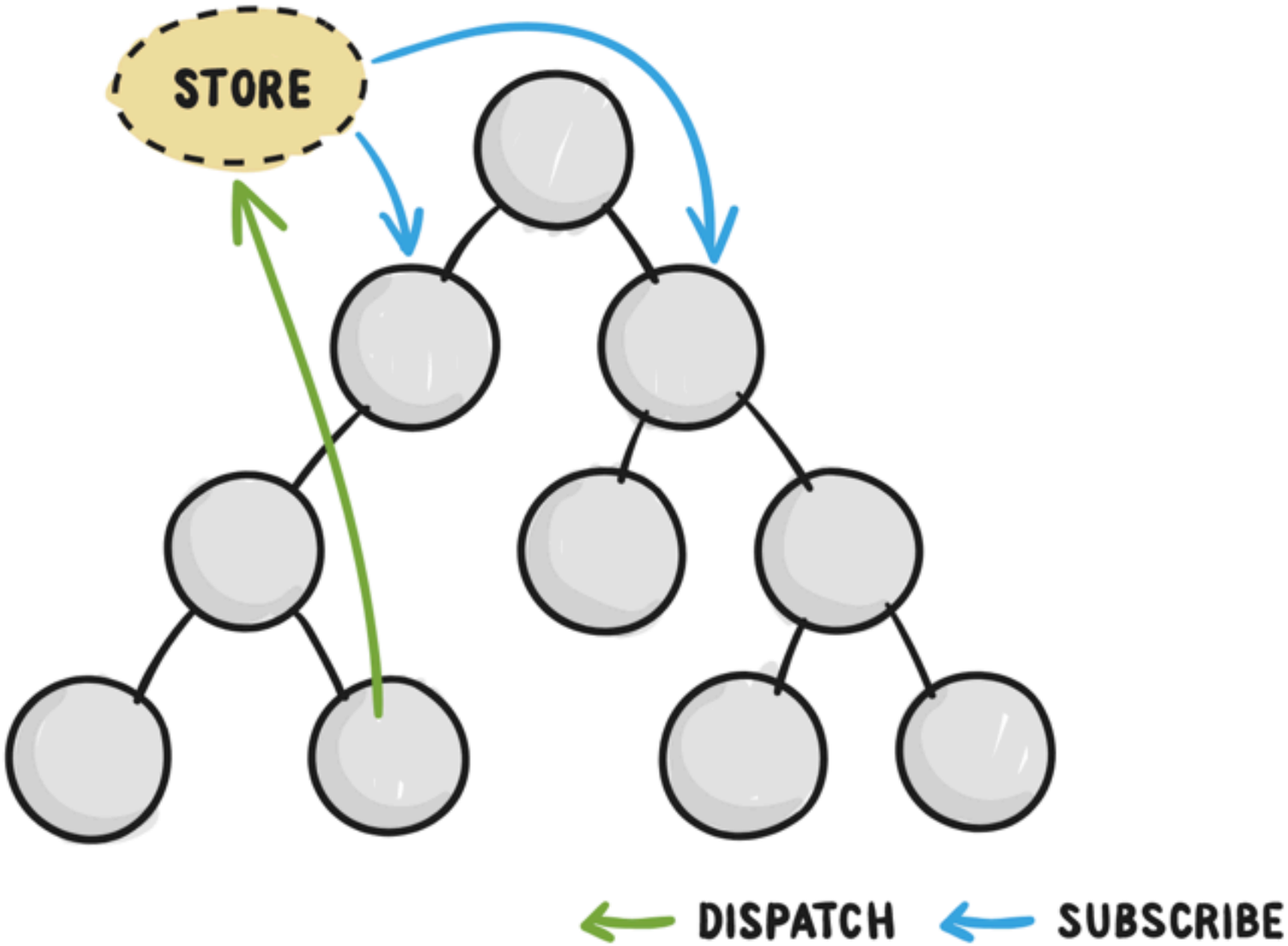


Image credit: Internet

React + Redux

- The props for React components come from the Redux store that tracks the state.

- React components react to user input and emit actions, either directly or indirectly.

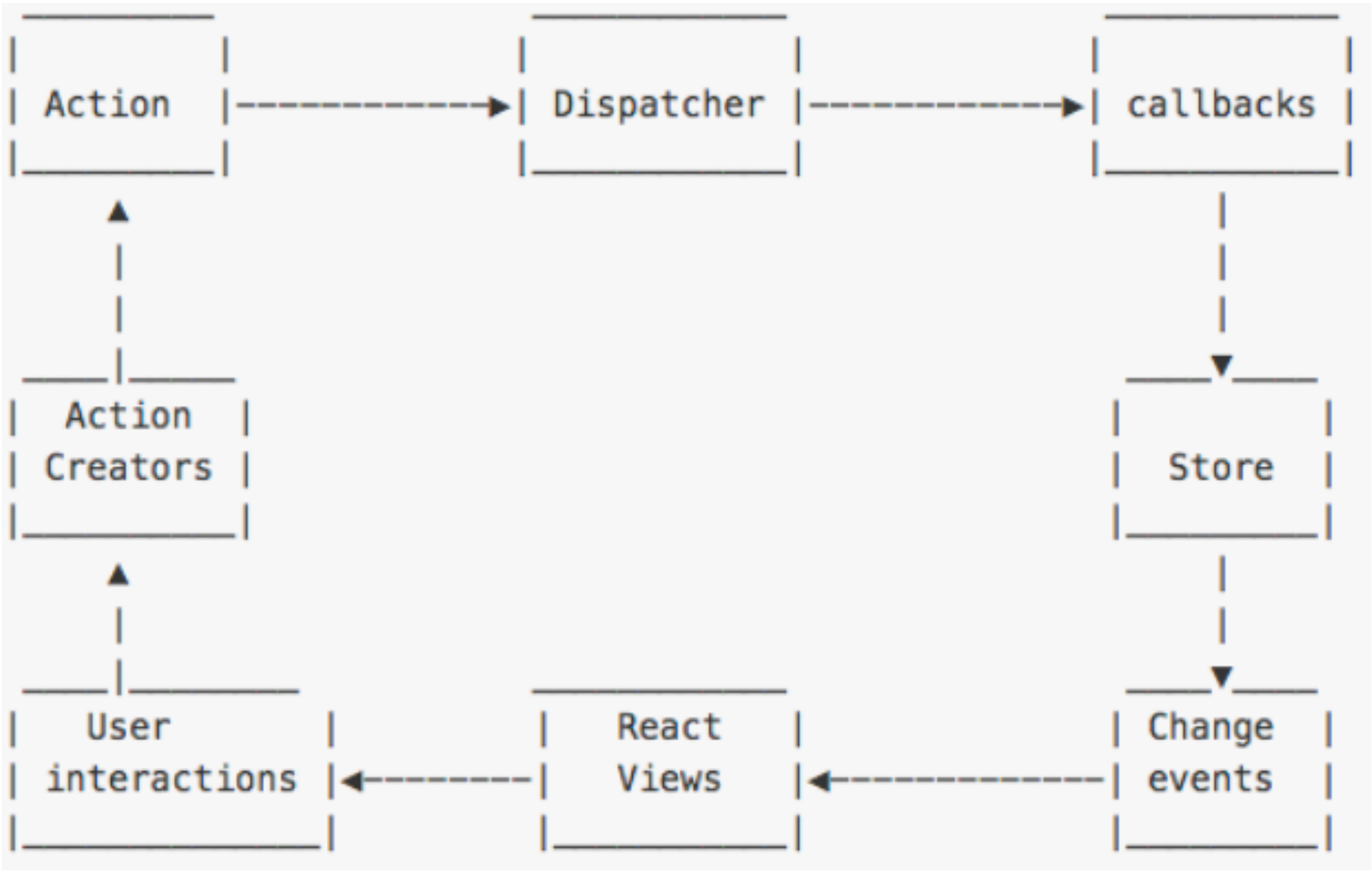
- Redux handles the action by running the appropriate reducers which transform the current state into a new state.

- React components react to the new state and update the DOM.

- React components themselves are stateless (most of the time), all of the state is kept in the Redux store, one common place, for simplicity

NODESENSE.AI

Redux + React Workflow



NODESENSE.AI

Actions

- Actions are object, contains type and payloads of information
- You send action to the store using store.dispatch().

```
let action = {
    type:”INCREMENT”,
    value: 10
}
```

OR

```
let action = {
    type:”INCREMENT”,
    payload: {
        value: 10
    }
}
```

NODESENSE.AI

Reducers

- Manages application state changes
- Reducers called with current state with action
- Redux stores all the data in one single object
- (previousState, action) => newState
- Every Action produces new state, i.e. you should not mutate the state

NODESENSE.AI

Reducers

- Given the same arguments, it should calculate the next state and return it.
- No surprises.
- No side effects.
- No API calls.
- No mutations. Just a calculation.

Reducer

```
const INITIAL_STATE = 0
function counterReducer(state=INITIAL_STATE, action) {

    switch(action.type) {
        case "INCREMENT": {
            return state + action.payload.value
        }
        case “DECREMENT": {
            return state - action.payload.value
        }
        case “RESET": {
            return INITIAL_STATE
        }
        default:
            return state;
    }
}
```

NODESENSE.AI

Reducer with List

```
const INITIAL_STATE = []
function cartReducer(state = INITIAL_STATE,
                                    action) {
 switch(action.type) {
  case ”ADD_TO_CART”:
      return [...state, action.payload.item]

  case “REMOVE_ITEM_FROM_CART”:
      return state.filter (item =>
                  item.id != action.payload.id)

  case “EMPTY_CART”:
      return []

   default:
      return state;
    }
}
```

NODESENSE.AI

Store

- Model/Data Management
- Manages the data for the application
- Updates data on events
- Notify the views after changes in data

Store

```
import {createStore} from "redux";

let store = createStore(counterReducer);
```

```
//to get last known state 0
Let state = store.getState();
console.log(store.getState())
```

NODESENSE.AI

Seed Data

```
import {createStore} from "redux";

// Initialise state with 100
let store = createStore(counterReducer, 100);
```

```
//to get last known state 100
Let state = store.getState();
console.log(store.getState())
```

NODESENSE.AI

combineReducer

- One store can have only one reducer
- What if we have more than one reducer?
- Redux has combineReducers

combineReducer

```
import {createStore, combineReducers} from "redux";

let rootReducer = combineReducers ({
    counter: counterReducer,
    productState: productReducer,
    cartState: cartReducers
})

let store = createStore(rootReducer)
var state = store.getState();
//state => {counter: 100, productState: {products: []},
                cartState: [{id: 1, price: 100…]
//state.productState
//state.counter
//state.cartState
```

NODESENSE.AI

Dispatch

Dispatch is the only way to call reducers

Dispatch dispatches actions to store. Store deliver to all reducers.

Upon receiving actions, reducers should respond with new state, if no state change, return existing state as it is.

Dispatch

Syntax:

```
store.dispatch({action})

store.getState() => 0

store.dispatch({
    type:”INCREMENT”,
    payload: { value: 10}
})

store.getState() => 10

store.dispatch({
    type:”DECREMENT”,
    payload: {value: 5}
})
store.getState() => 5
```

NODESENSE.AI

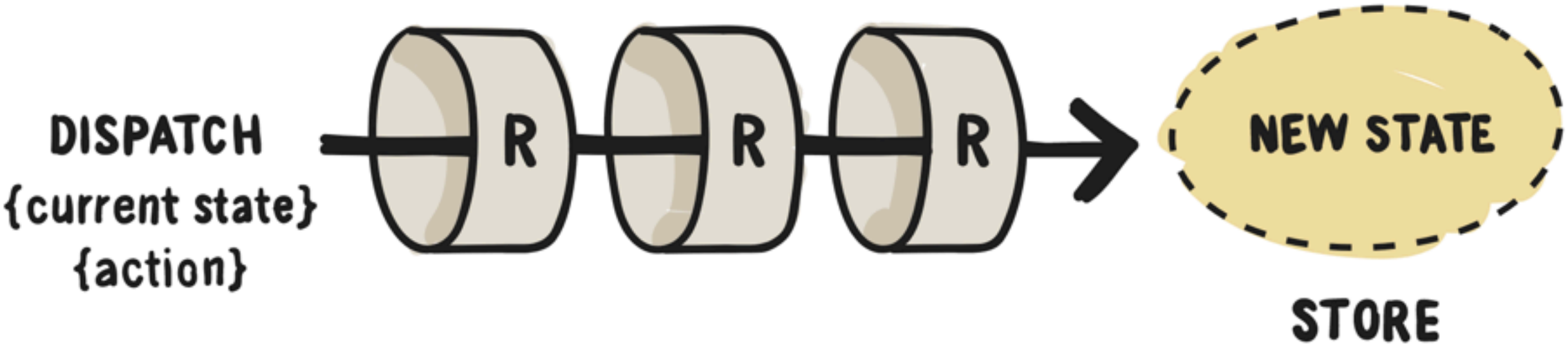


Image credit: Internet

NODESENSE.AI

Image credit: Internet

Action Creators

- Action creators are helper method, that create actions

Action Creators

```
function incrementActionCreator(value)
{
    return {
        type: “INCREMENT”,
        value: value
    }
}
```

```
function decrementAction(value) {
    return {
        type: ”DECREMENT”,
        value
    }

}
 store.dispatch(incrementAction(10))
store.dispatch(decrementAction(5))
```

NODESENSE.AI

Subscribe

- Any components interested in data from store can subscribe
- Subscribe is be called for every dispatch
- Subscription can be unsubscribed

Subscribe

```
let unsubscribeFn = store.subscribe ( () => {
    console.log(“updated values", store.getState());
})

// at the end, don’t fail to unsubscribe
unsubscribeFn()
```

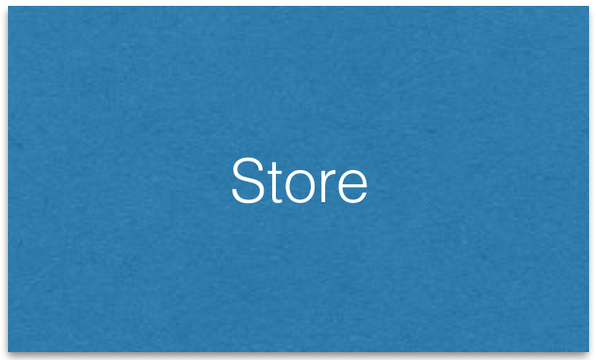
NODESENSE.AI

With React Component

```
componentDidMount() {
        this.unsubscribe = store.subscribe( () => {
            this.setState({
                result: store.getState()
            })
    }

componentWillUnmount() {
        if (this.unsubscribe)
            this.unsubscribe();
    }
```

NODESENSE.AI

```
const INITIAL_STATE = 0;
function reducer(state = INITIAL_STATE,
                                action) {
    switch(action.type) {
        case "INCREMENT":
            return state + 1;

        default:
            return state;
    }
}
```



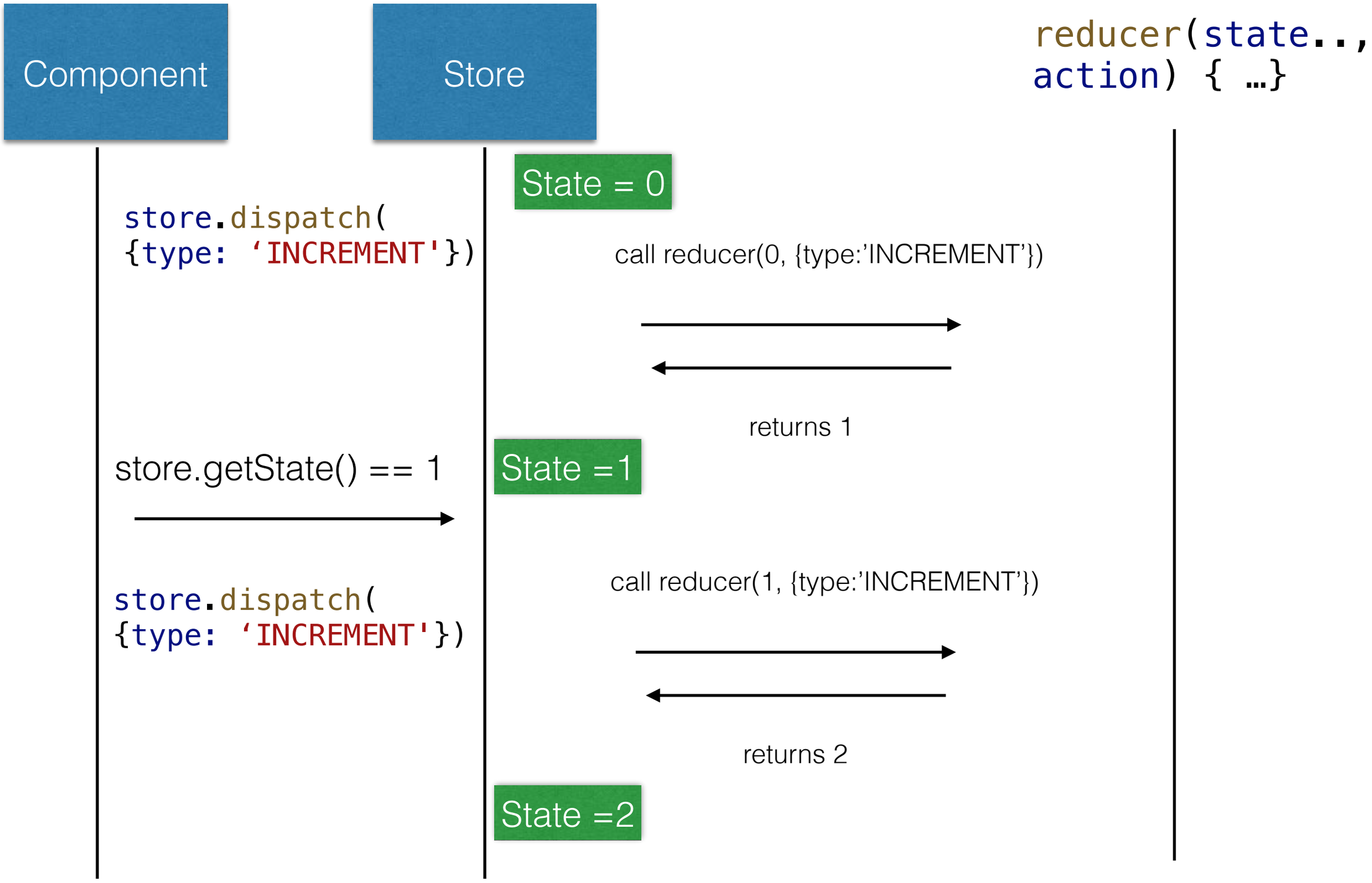
call reducer(0, {type:'INCREMENT'})

returns 1

```
Function
reducer(state..,
action) { …}
```

```
store.dispatch({type: 'INCREMENT'});
```

NODESENSE.AI



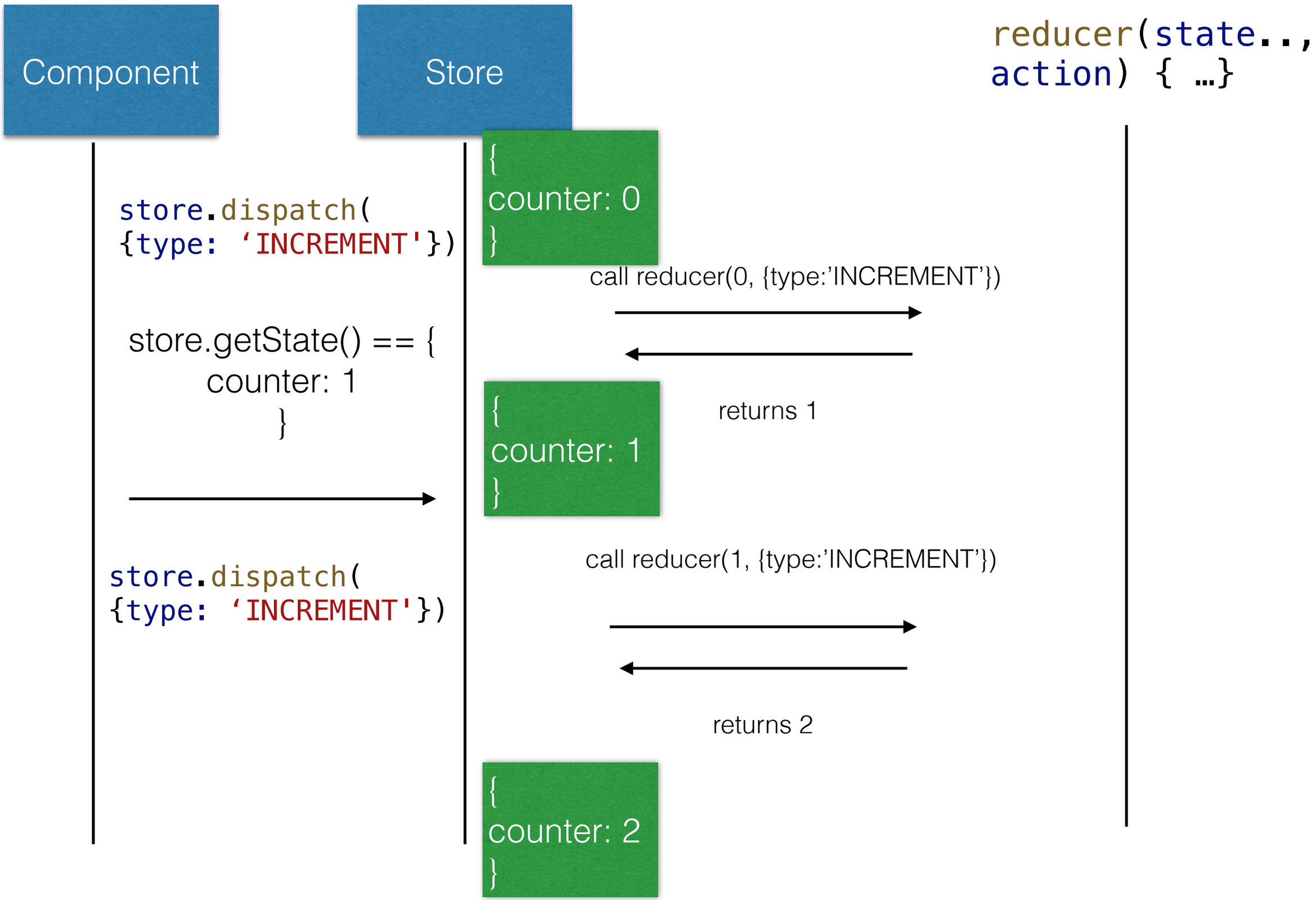
NODESENSE.AI

combineReducer

```
import {createStore, combineReducers} from "redux";

let rootReducer = combineReducers ({
    counter: counterReducer
})

let store = createStore(rootReducer)
var state = store.getState();

//state => {counter: 100}
//state.counter
```

NODESENSE.AI

combine Reducer



NODESENSE.AI

```
let rootReducer = combineReducers({
    //stateName: reducer fn
    counter: counterReducer,
    cartItems: cartReducer
    //
})
```

NODESENSE.AI

combine Reducer

Component

Store

```
reducer(state..,
action) { …}
```

```
store.dispatch(
{type: ‘INCREMENT'})
```

store.getState() == {
counter: 1,
cartItems: []
}

{
counter: 0,
cartItems: []
}

call reducer(0, {type:’INCREMENT’})

returns 1

call cartReducer([], {type:’INCREMENT’})

returns []

{
counter: 1,
cartItems: []
}

```
store.dispatch(
{type: ‘ADD_ITEM..’})
```

call counterReducer(1, {type:’ADD_ITEM’})

returns 1 (default case)

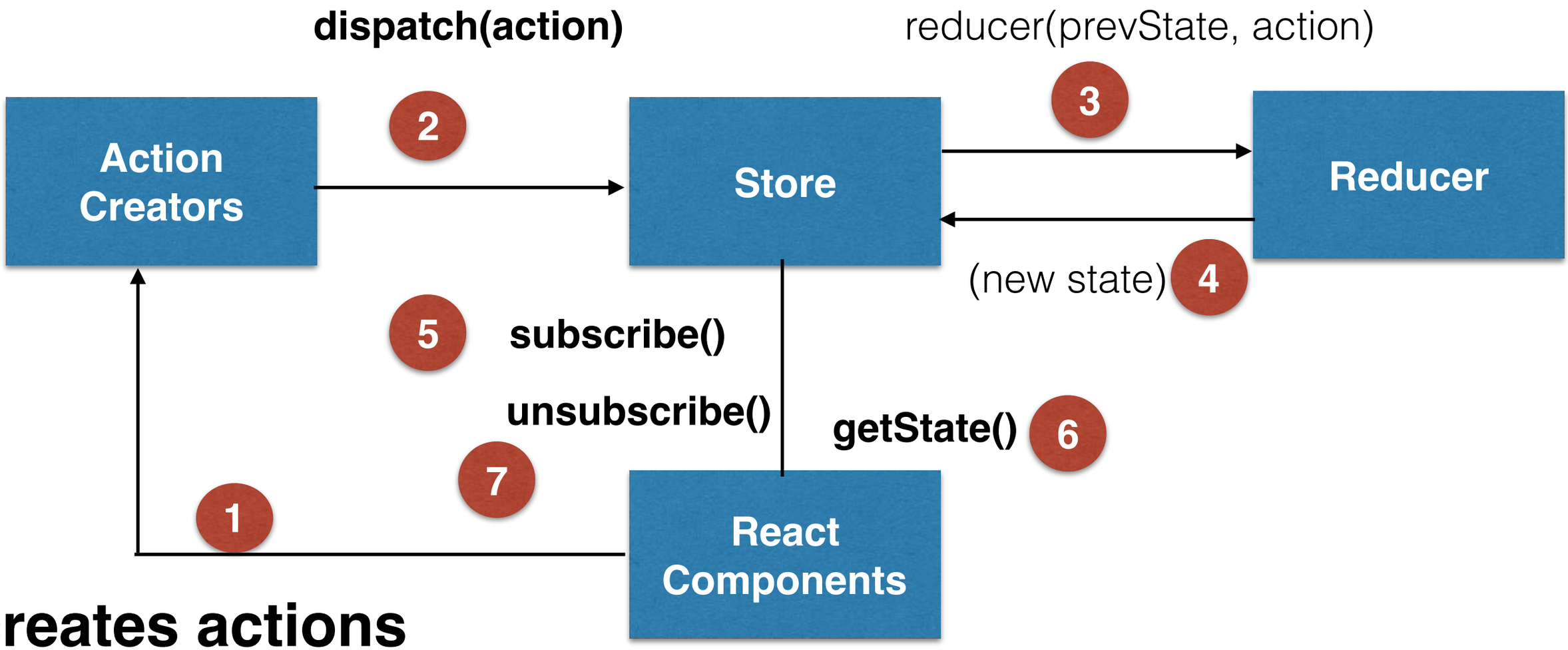
call cartReducer([], {type:’ADD_ITEM’})

returns. [{id: 1, name..]

{
counter: 1,
cartItems: [{id: 1]
}

NODESENSE.AI

Redux Flow



NODESENSE.AI

Middleware

- Middleware are added between Dispatch and Reducers

- Often useful to log the data, perform actions with promises, modify data on the fly

- Middleware can stop action going to reducer as well

Example

```
const logger = store => next => action => {
  console.log('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}
```

next(…) calls next middleware/reducer in the chain

Redux-thunk

```
npm install redux-thunk --save
```

For handling async actions and promises

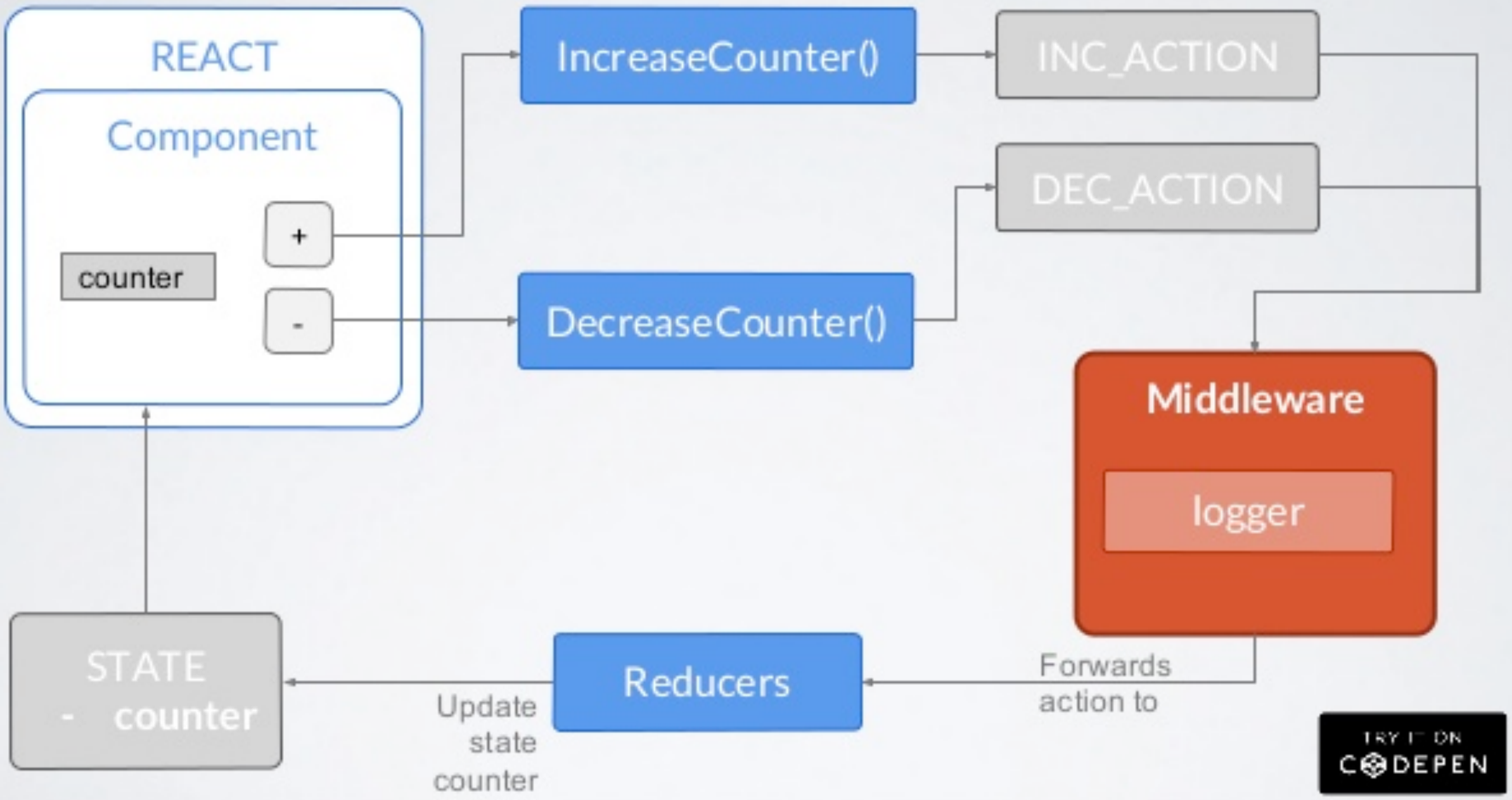
Store with Middleware

```
import {createStore, applyMiddleware} from "redux";
import thunk from “redux-thunk”

let store = createStore(math_reducer,
                        initialState,
                        applyMiddleware(thunk,
                                        logger));
```

NODESENSE.AI

Redux life-cycle with middlewares

Simplest example - logger



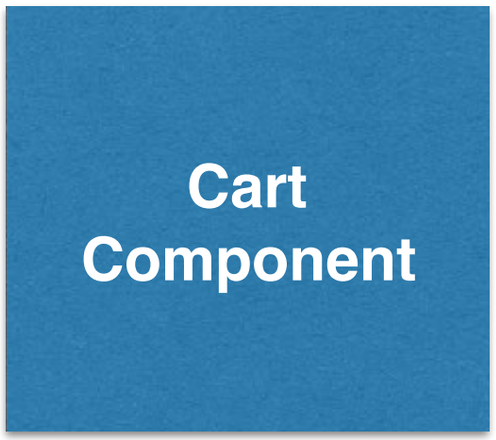
React + Redux

- react-redux library
- mapStateToProps
- mapDispatchToProps
- connect

React Redux

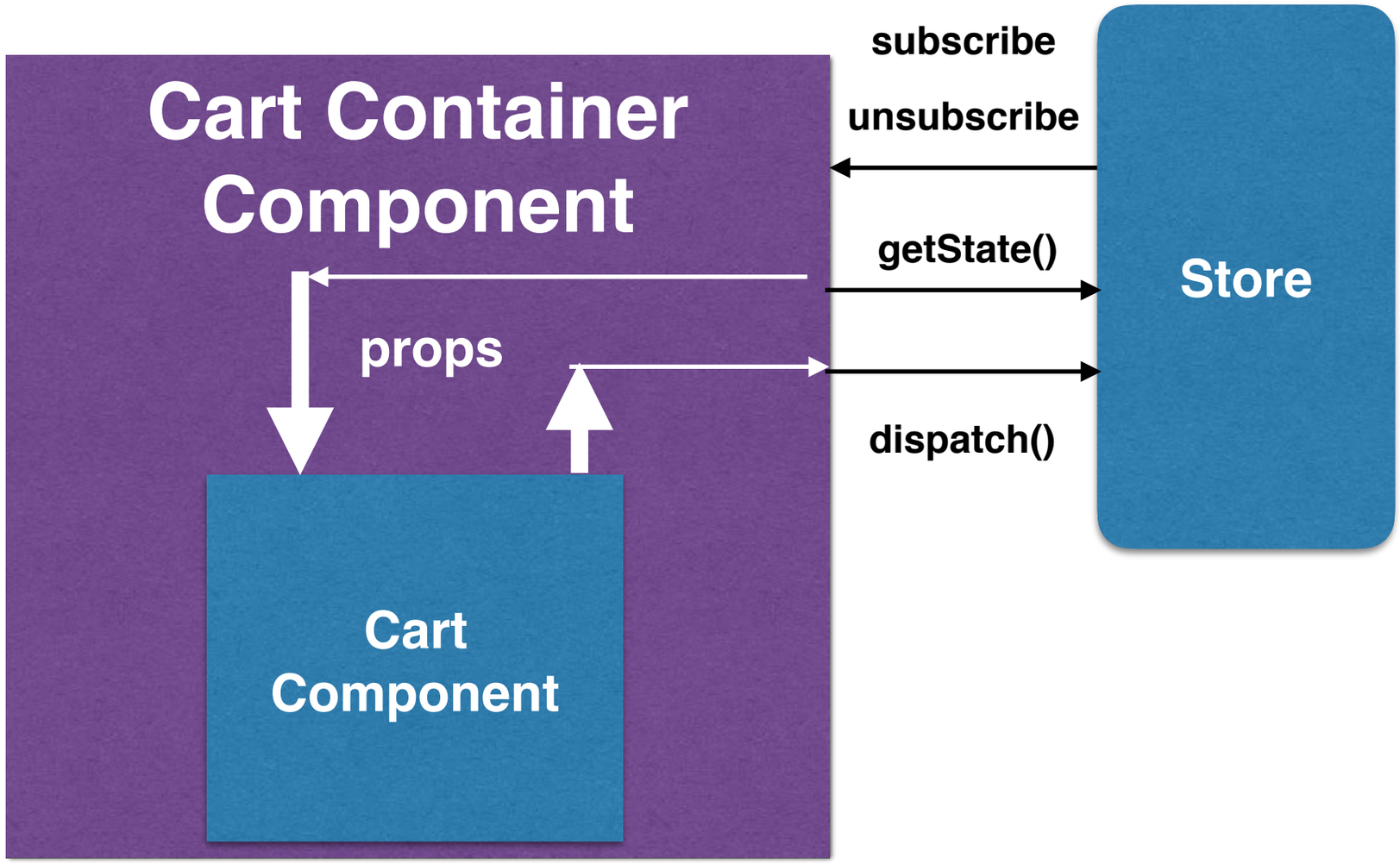
- Connects React and Redux
- Wrap the React components within its **container component**
- **Provides clear segregation between React and Redux, keep them independent**

React World



Your
Component

Redux World

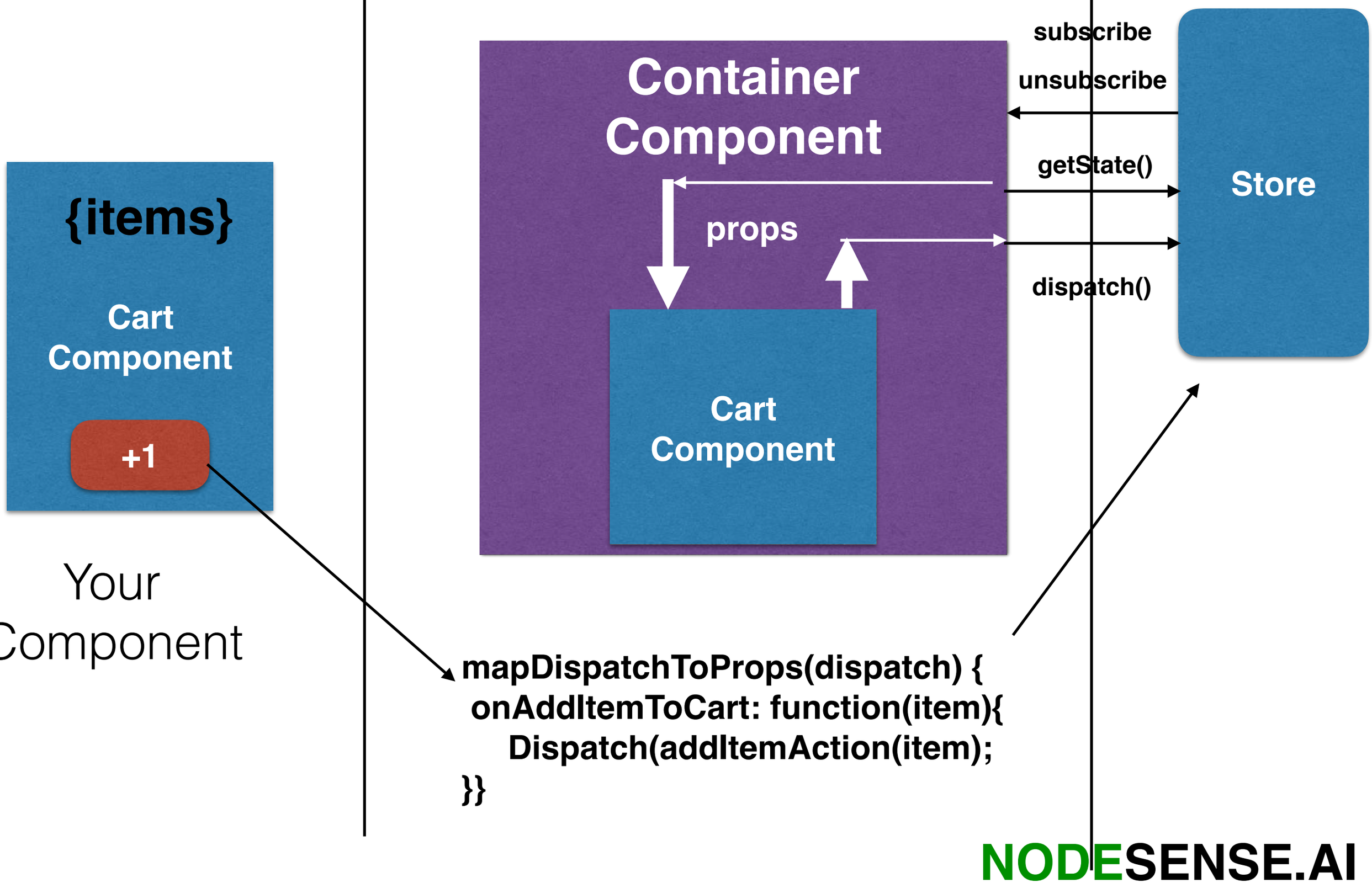


Containers

NODESENSE.AI

React World

```
mapStateToProps(state) {
   return { count: state.cartItems.length;
}
```



NODESENSE.AI

React-Redux

- npm install react-redux —save
- Connect React and Redux

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

render( (
            <Provider store={store}>
                <App />
            </Provider>
        )       , document.getElementById("root"));
```

NODESENSE.AI

React-Redux

- React-Redux maps Redux State to Component Params or Presentation Views functional arguments
- React-Redux maps Redux Store Dispatch to props
- Finer Abstraction over Redux while using React Components

React Redux

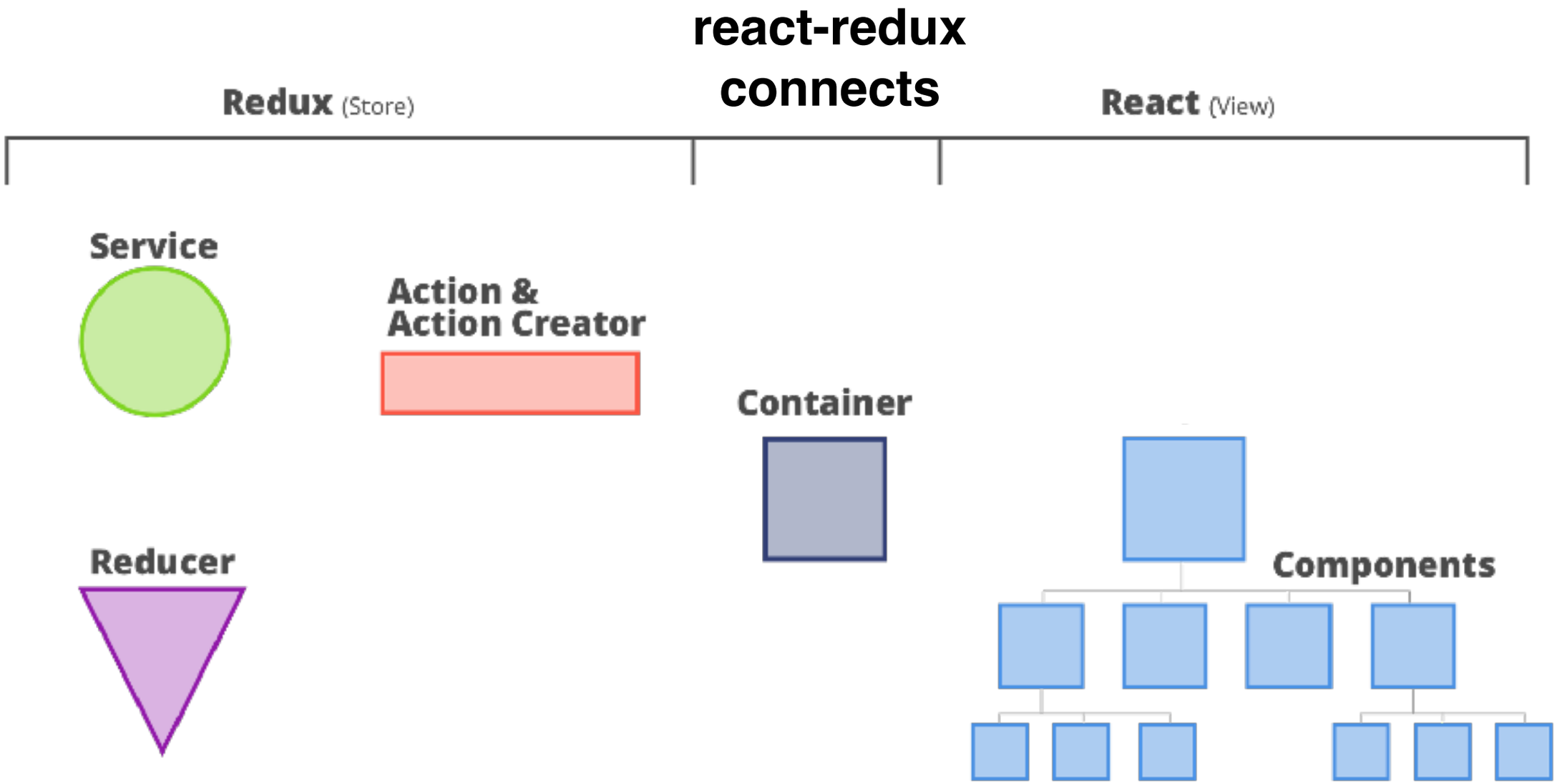


Image credit: Internet

NODESENSE.AI

React Redux

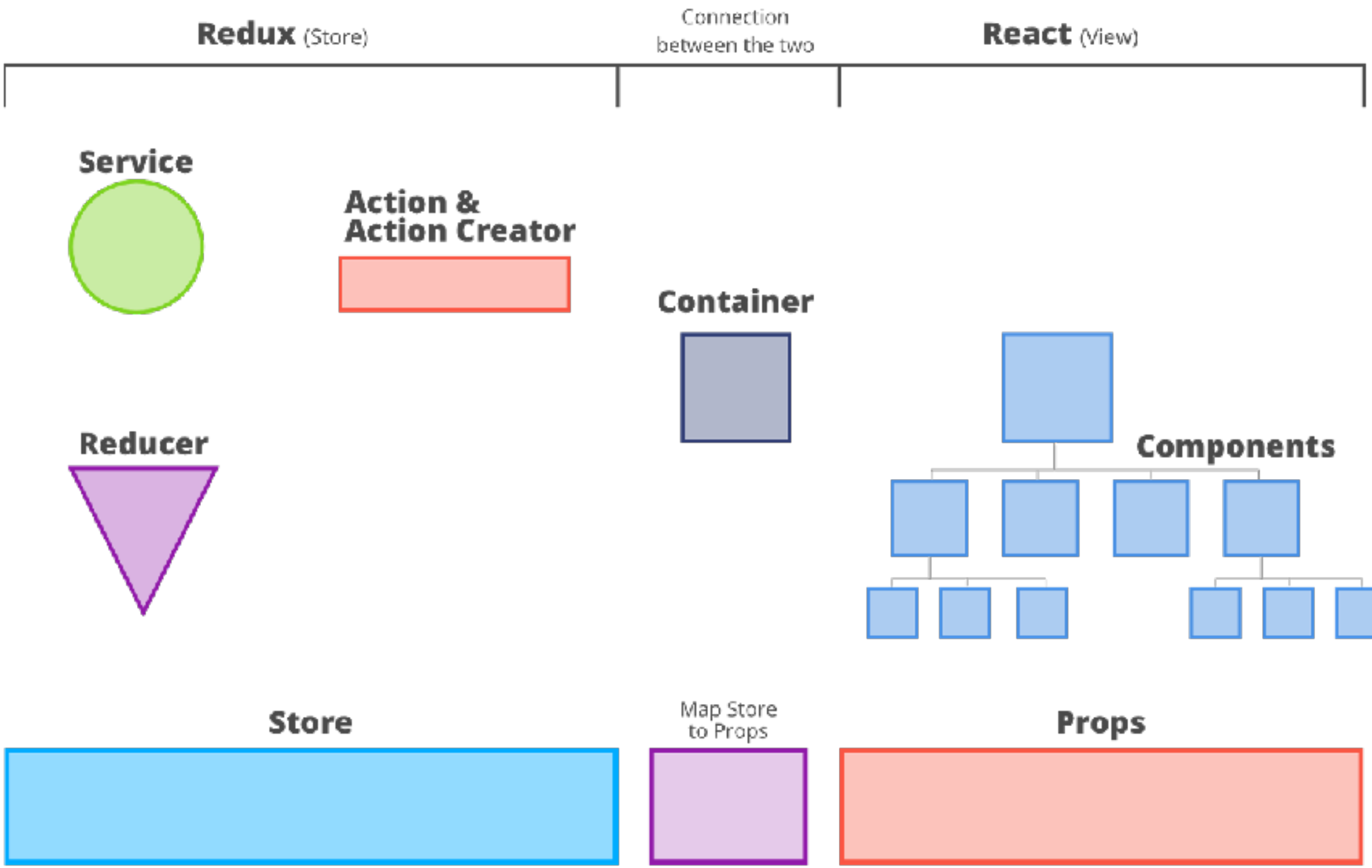


Image credit: Internet

NODESENSE.AI

React Redux

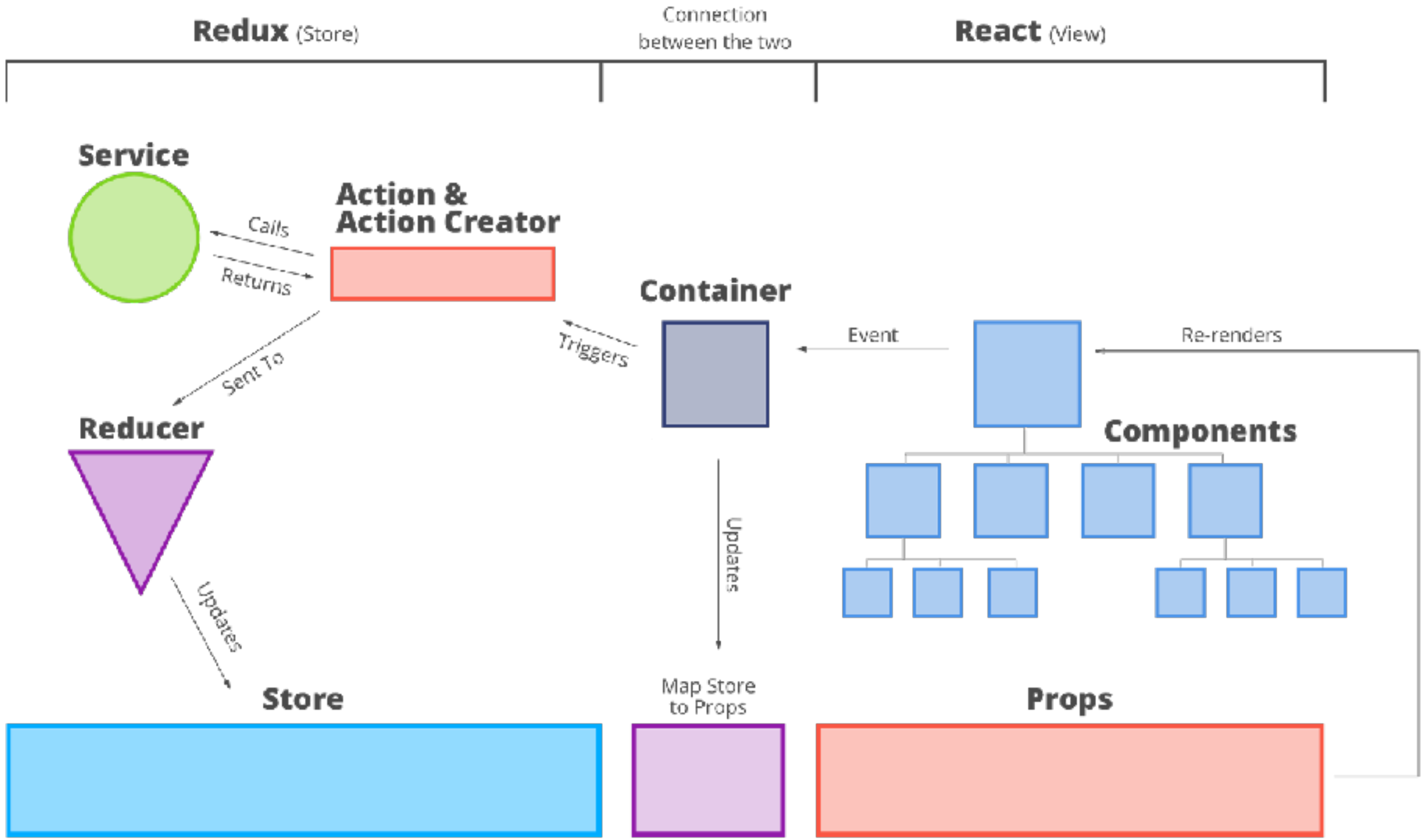


Image credit: Internet

NODESENSE.AI

```
import React, {PropTypes} from "react";

class Cart extends React.Component {
render() {
    return (
        <div>
        <h1>{this.props.items.length}</h1>
        <button onClick={this.props.emptyCart}>
            Empty Cart
        </button>
        </div>
    )
    }
}

Cart.propTypes = {
    items: PropTypes.array
}
```

NODESENSE.AI

```
import {connect} from "react-redux";
import Cart from “../Cart”
const mapStatesToProps = (state) => {
    return {
        items: state.cartItems
    }
}

const mapDispatchToProps = (dispatch) => {
    return {
        emptyCart: () => {
            dispatch({type: 'EMPTY'});
        }
    }
}

CartContainer = connect(mapStatesToProps,
                mapDispatchToProps)(Cart);

export default CartContainer;
```

NODESENSE.AI