Introduction to ReactJS

Overview

React is a JavaScript library by Facebook

- It describes itself as a javascript library for building user interfaces.
- Developers often call it the V in MVC

Overview

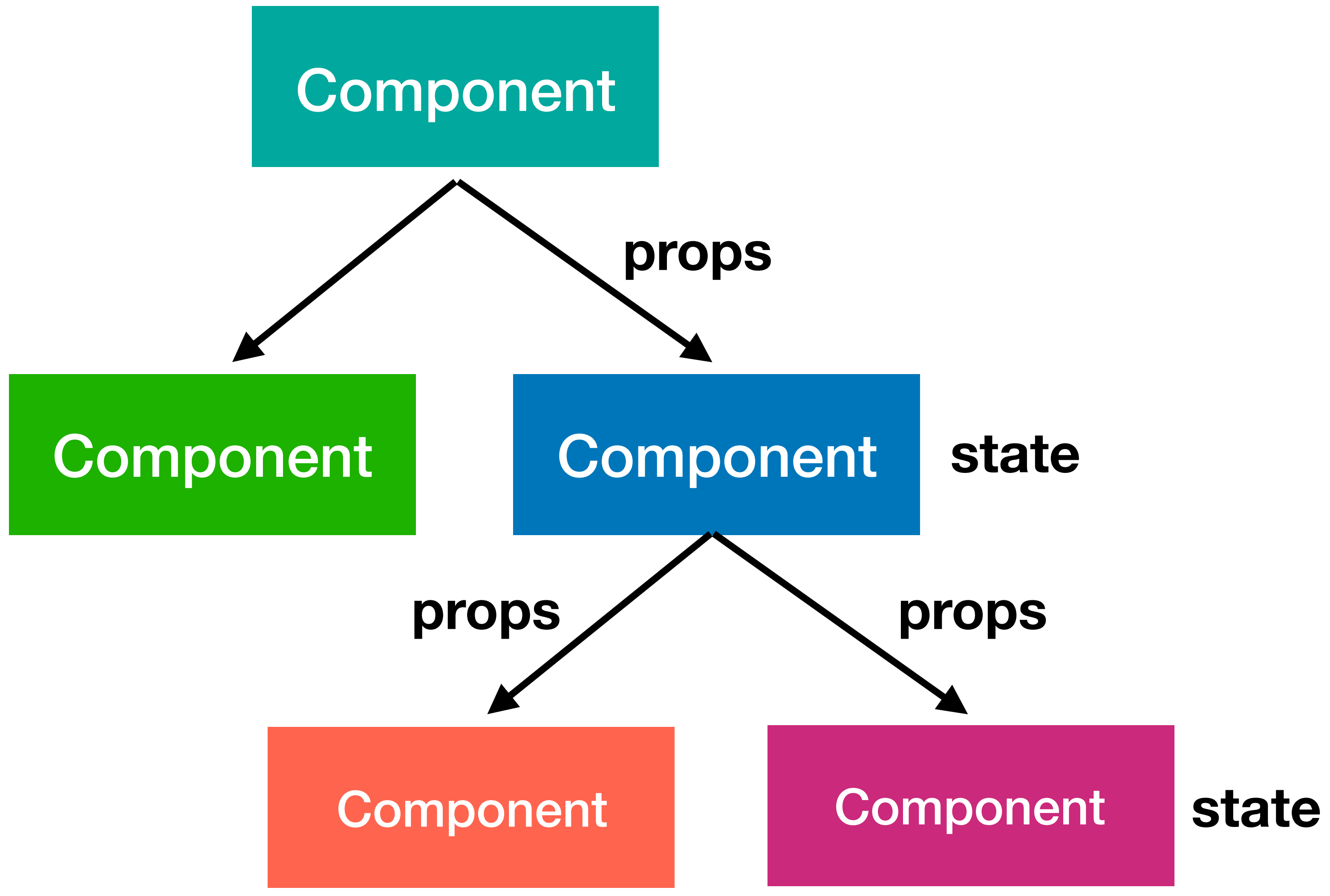
Key elements of React

- Component
- JSX
- Virtual DOM
- Component lifecycle

Overview

Components

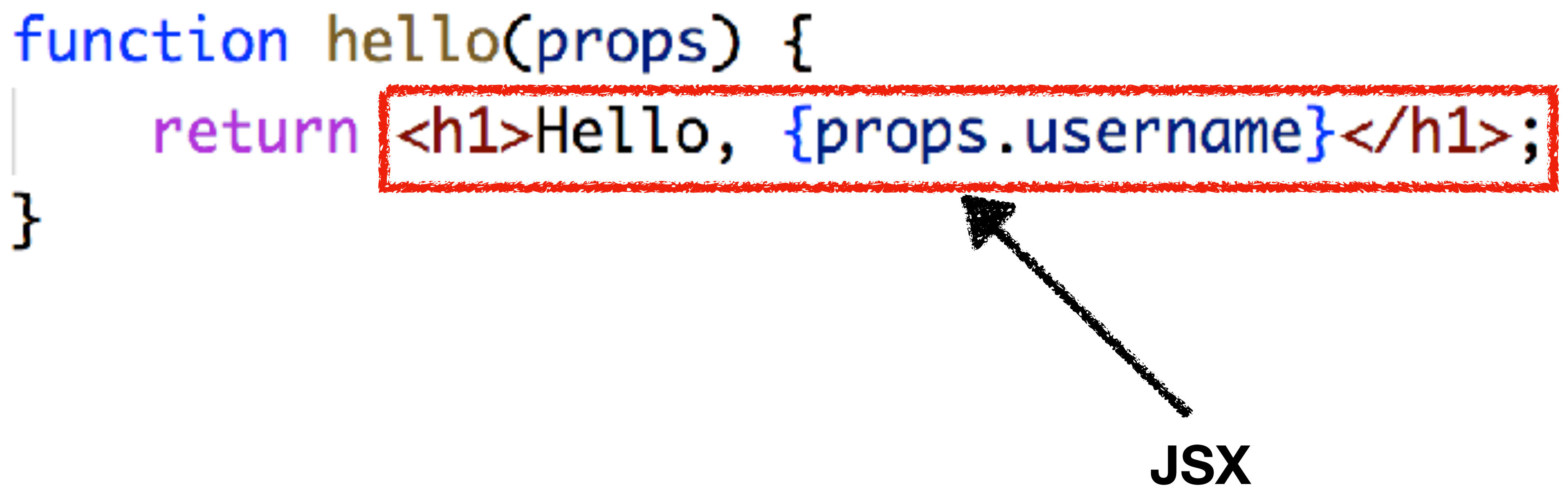
- Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.



Components

Conceptually, components are like JavaScript functions.

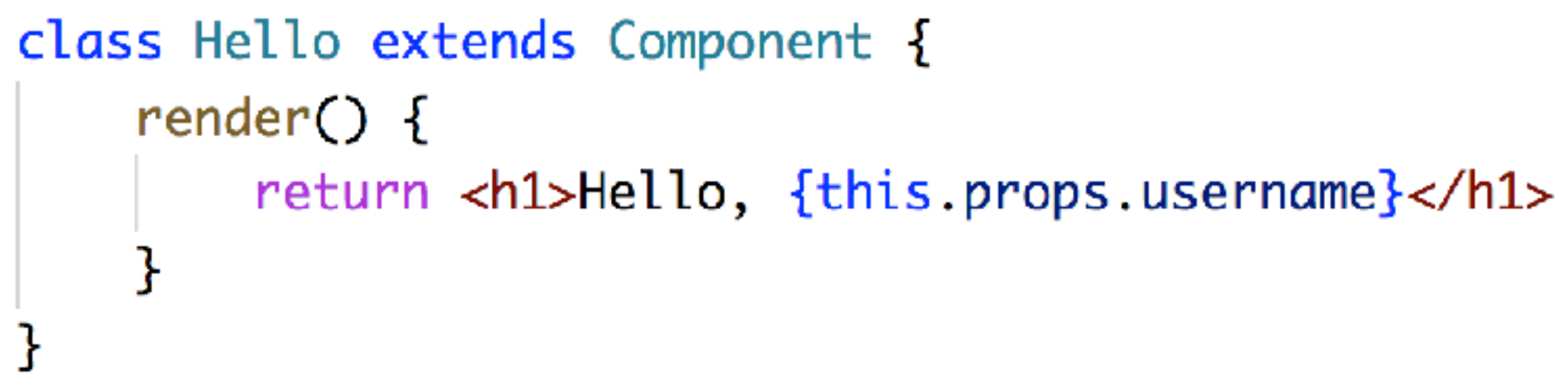
- They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.
- The simplest way to define a component is to write a JavaScript function:



```
function hello(props) {
    return <h1>Hello, {props.username}</h1>;
}
```

Components

You can also use an ES6 class to define a component:

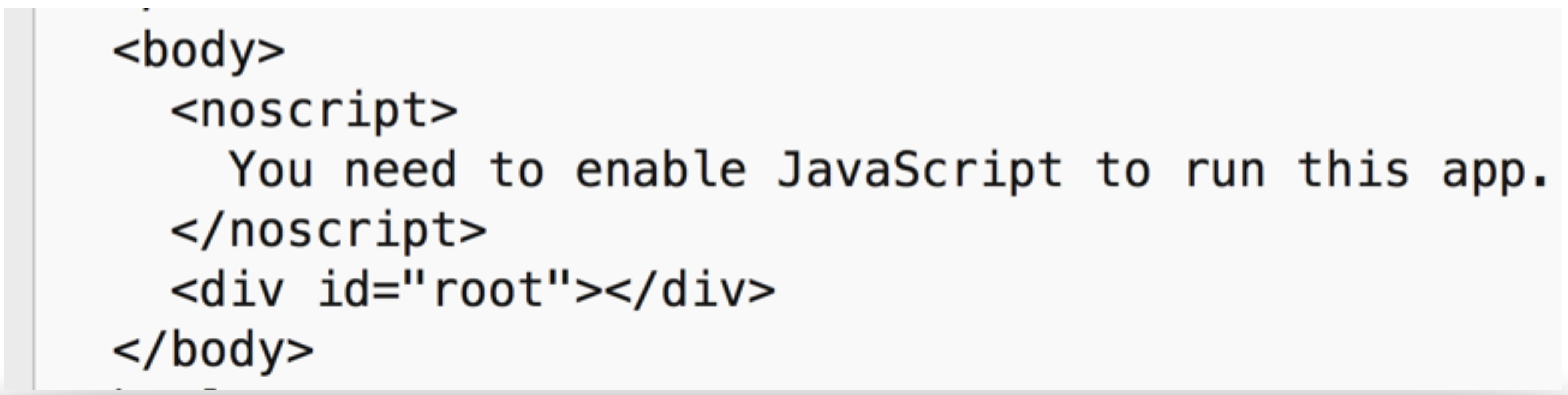


```
class Hello extends Component {
    render() {
        return <h1>Hello, {this.props.username}</h1>
    }
}
```

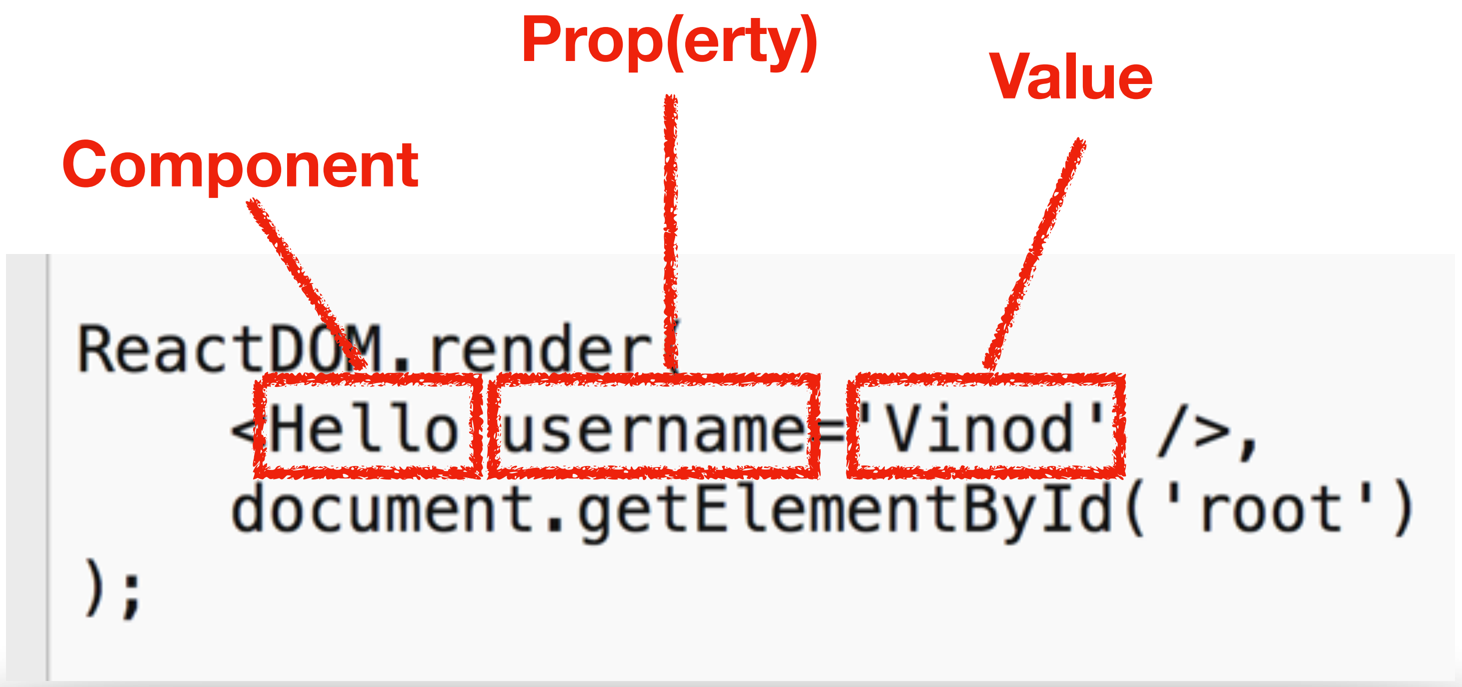
- render() is one many lifecycle methods of a React component
- User defined components must start with a capital letter
- **this.props** gets the value from **<Hello username="Vinod" />**

Components

Rendering component



```
<body>
  <noscript>
    You need to enable JavaScript to run this app.
  </noscript>
  <div id="root"></div>
</body>
```

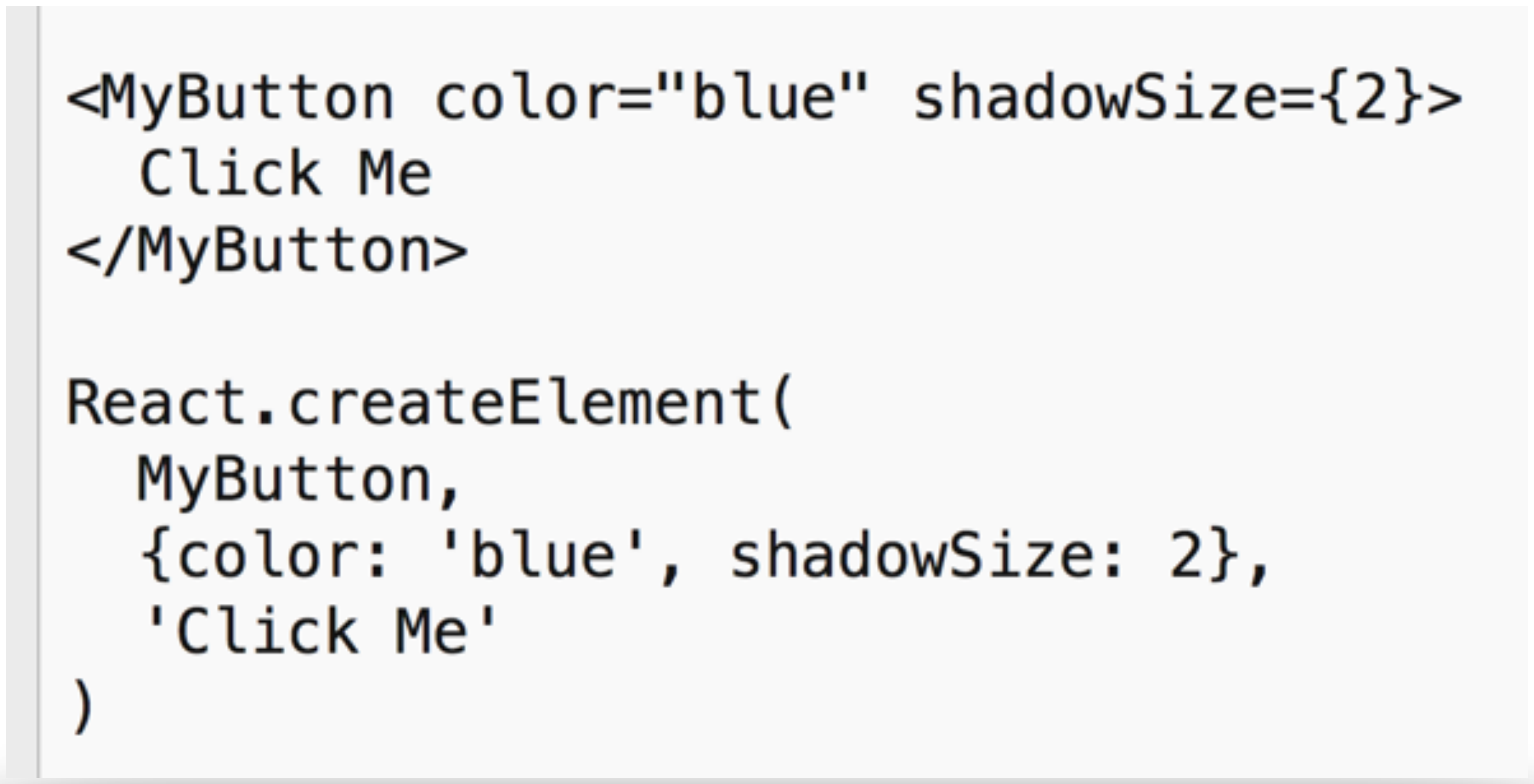


```
ReactDOM.render(
    <Hello username='Vinod' />,
    document.getElementById('root')
);
```

JSX

JavaScript XML

- JSX is a XML-like syntax extension to ECMAScript without any defined semantics
- JSX just provides syntactic sugar for the React.createElement(...) function.



```
<MyButton color="blue" shadowSize={2}>
  Click Me
</MyButton>

React.createElement(
  MyButton,
  {color: 'blue', shadowSize: 2},
  'Click Me'
)
```

JSX

JavaScript XML

- Some of the commonly used HTML attributes can not be used directly, as they collide with JavaScript reserved words

- <div **class**="…" > should be <div **className**="…">
- <label **for**="..."> should be <label **htmlFor**="…">

JSX

JavaScript XML

- You can use variables and expressions inside the JSX inside { }

- <Hello username={my_name} />
- <h1>Hello {props.username}</h1>
- <button onClick={ ()=>alert('Hi') }>Click me</button>

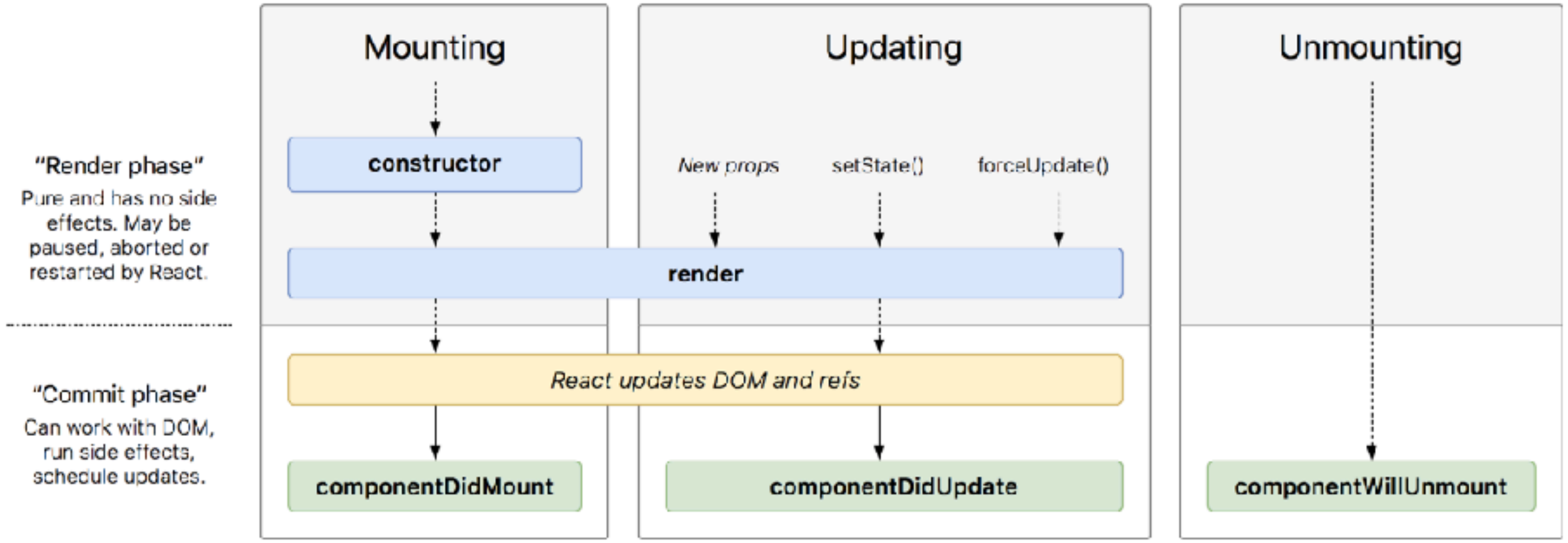
Virtual DOM

Virtual Document-Object-Model (DOM)

- React creates an in-memory data structure cache, computes the resulting differences, and then updates the browser's displayed DOM efficiently.
- This allows the programmer to write code as if the entire page is rendered on each change, while the React libraries only render sub components that actually change.

Component Lifecycle

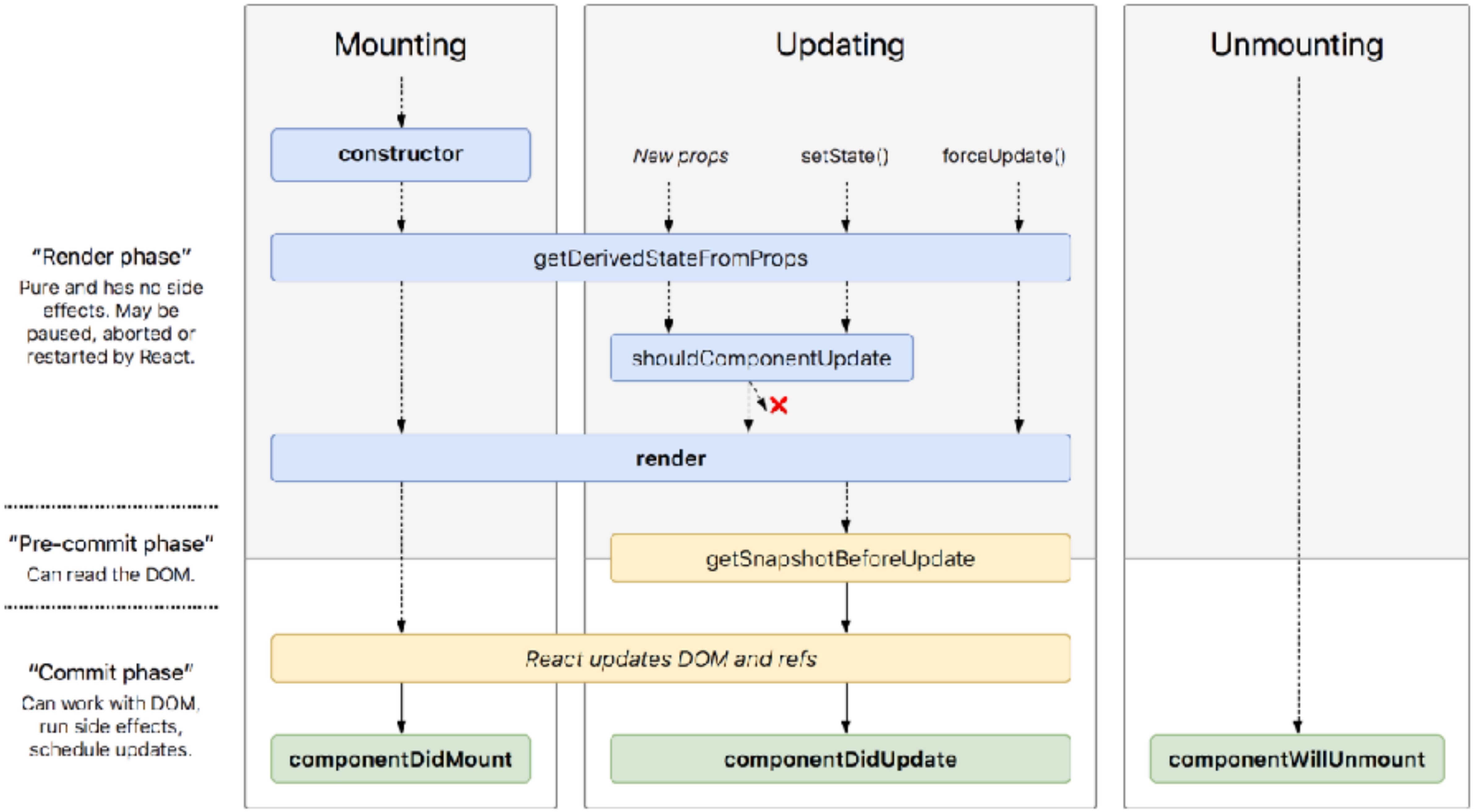
Common lifecycle hooks



http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/

Component Lifecycle

All lifecycle hooks



© https://vinod.co

Mounting

These methods are called in the following order when an instance of a component is being created and inserted into the DOM:

- constructor()
- static getDerivedStateFromProps()
- render()
- componentDidMount()

Updating

An update can be caused by changes to props or state.

These methods are called in the following order when a component is being re-rendered:

- static getDerivedStateFromProps()
- shouldComponentUpdate()
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

Unmounting

This method is called when a component is being removed from the DOM:

- componentWillUnmount()

Error Handling

This method is called when there is an error during rendering, in a lifecycle method, or in the constructor of any child component.

- componentDidCatch()

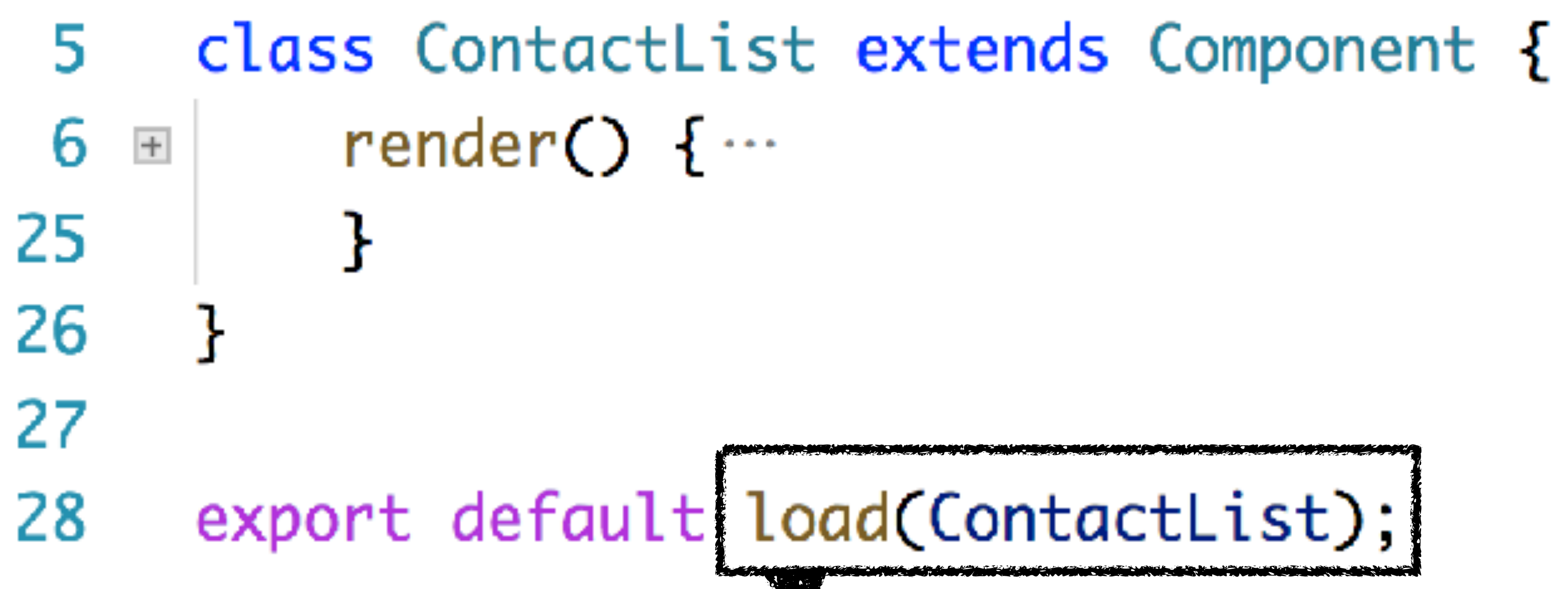
Higher-Order Component

HOC is an advanced technique in React for reusing component logic

- They are a pattern that emerges from React's compositional nature.
- Use HOCs For Cross-Cutting Concerns
- It is a function that takes a component and returns a new component.

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

https://reactjs.org/docs/higher-order-components.html



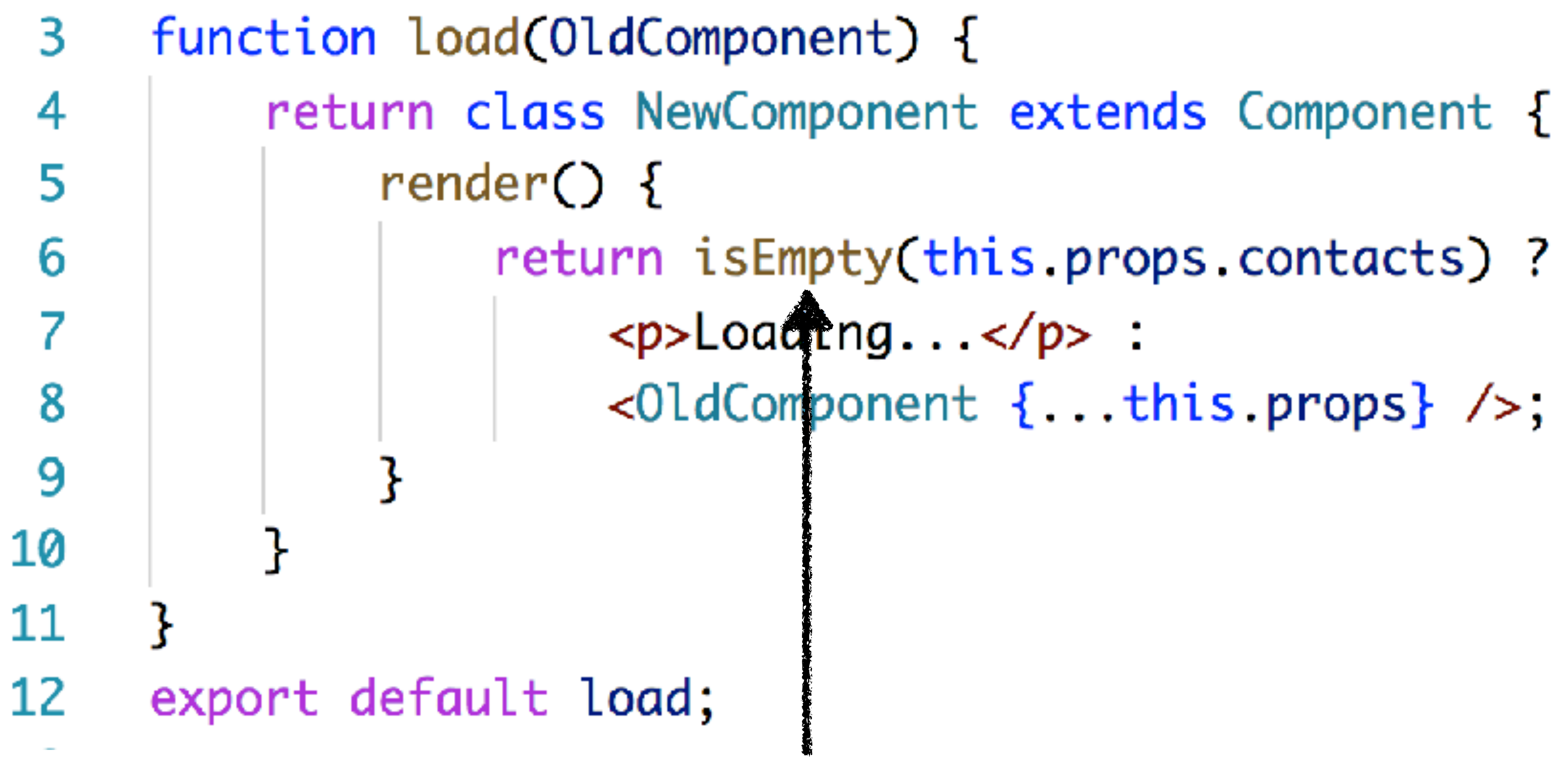
```
class ContactList extends Component {
    render() {···
    }
}

export default load(ContactList);
```

*NewComponent

Higher Order Component

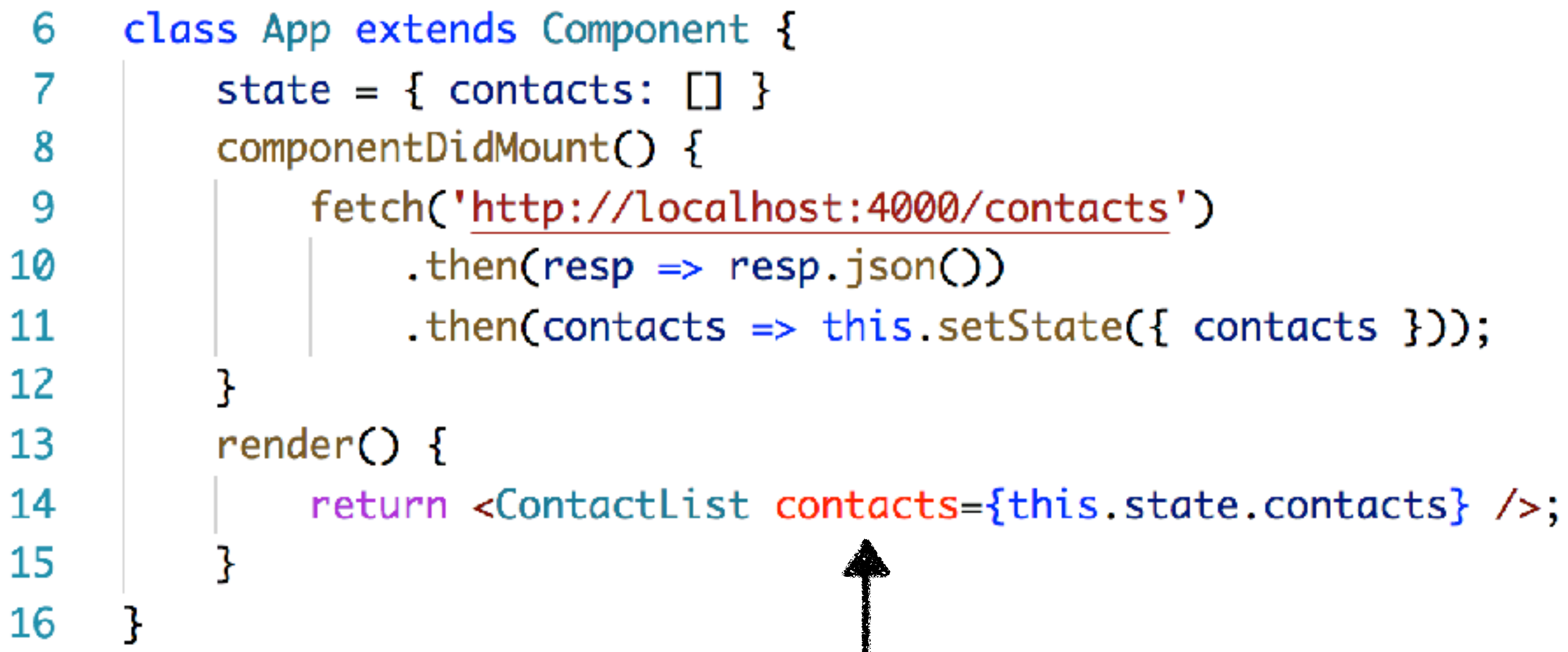
(accepts one component and returns another)

**Checks if the “contacts” prop is empty.
If yes, returns a ‘loading…’ component,
else returns the same (ContactList)**



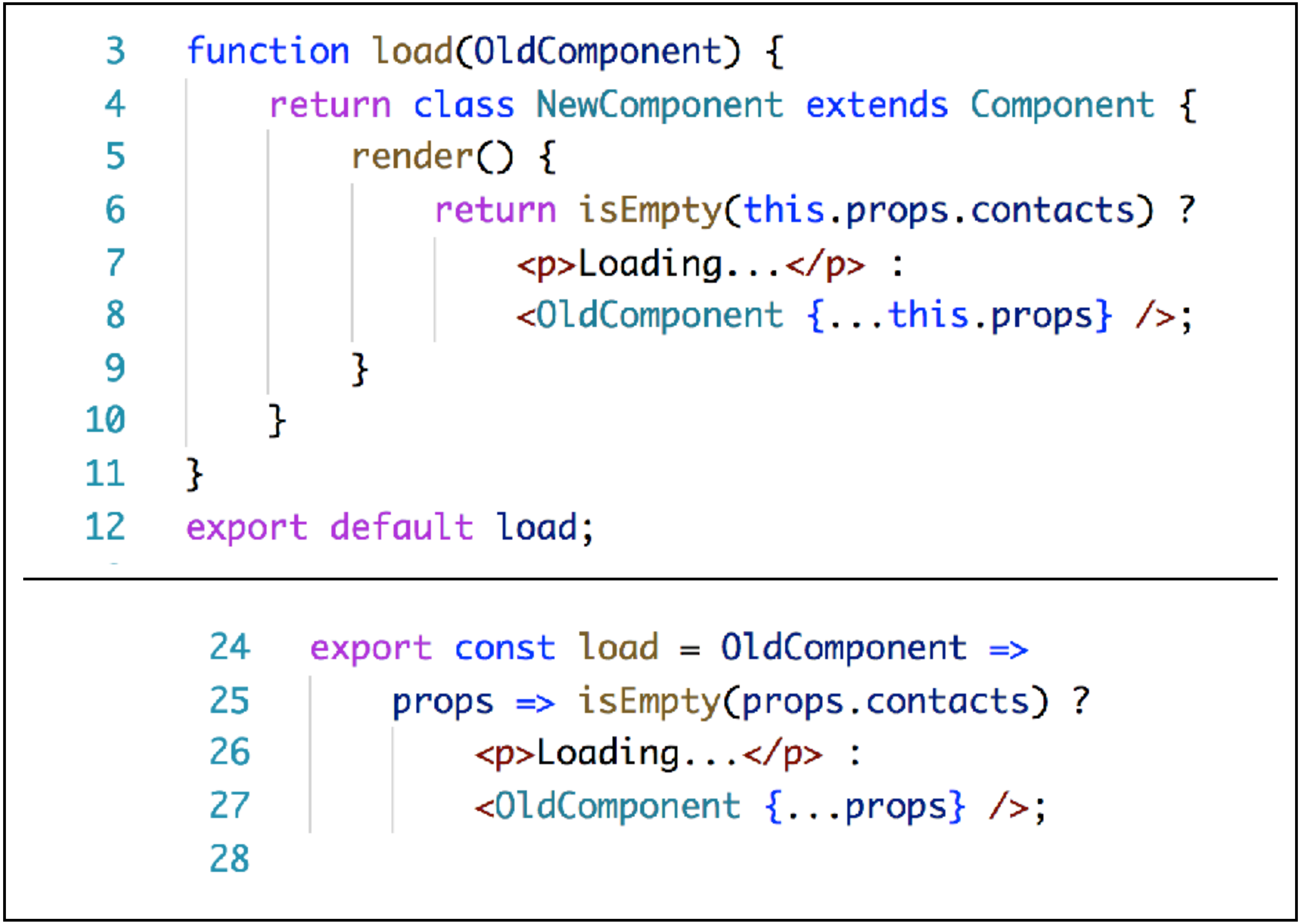
```
function load(OldComponent) {
    return class NewComponent extends Component {
        render() {
            return isEmpty(this.props.contacts) ?
                <p>Loading...</p> :
                <OldComponent {...this.props} />;
        }
    }
}
export default load;
```

User defined function



```
class App extends Component {
    state = { contacts: [] }
    componentDidMount() {
        fetch('http://localhost:4000/contacts')
            .then(resp => resp.json())
            .then(contacts => this.setState({ contacts }));
    }
    render() {
        return <ContactList contacts={this.state.contacts} />;
    }
}
```

Passed to “NewComponent”



```
function load(OldComponent) {
    return class NewComponent extends Component {
        render() {
            return isEmpty(this.props.contacts) ?
                <p>Loading...</p> :
                <OldComponent {...this.props} />;
        }
    }
}
export default load;
```

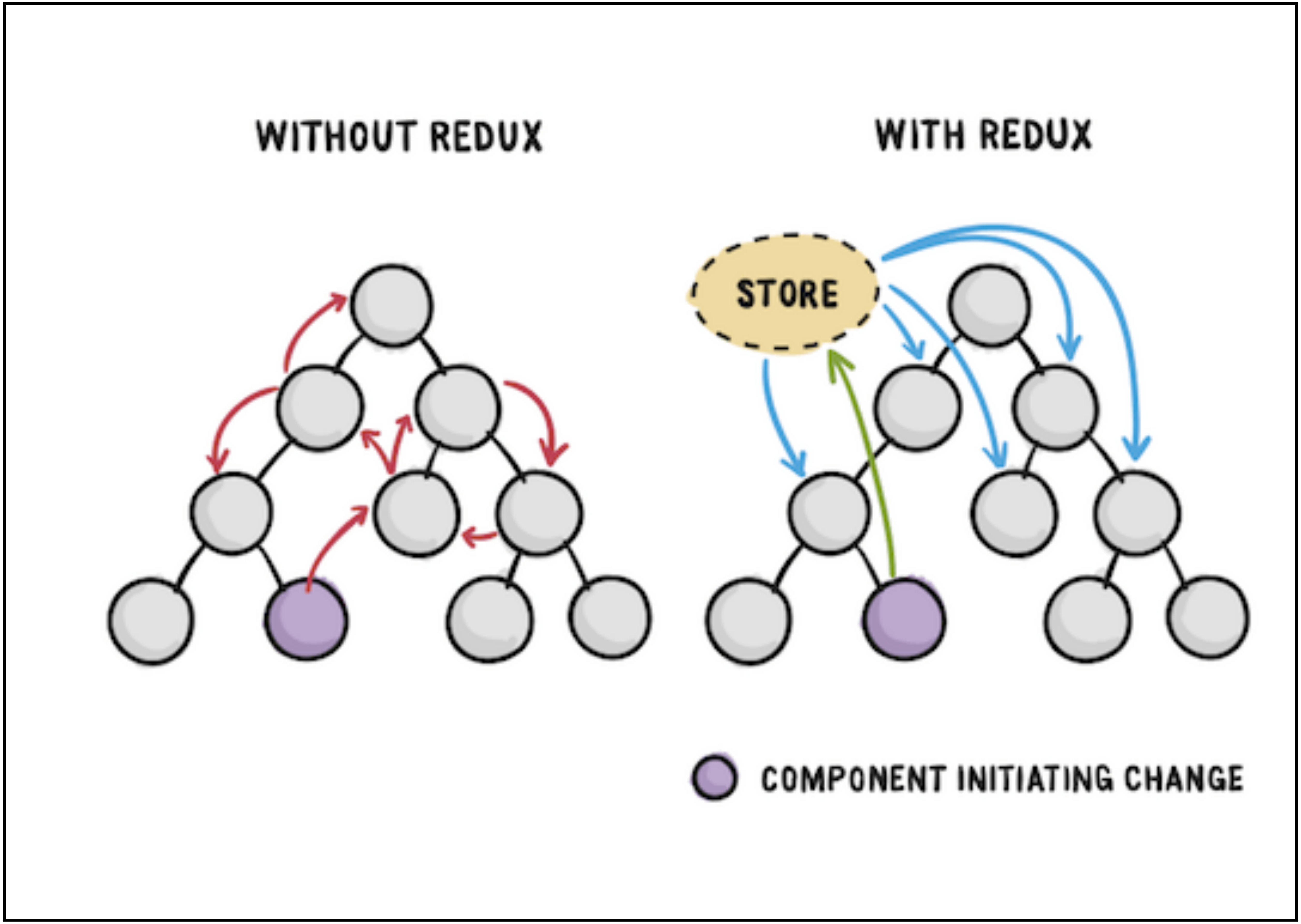
```
export const load = OldComponent =>
    props => isEmpty(props.contacts) ?
        <p>Loading...</p> :
        <OldComponent {...props} />;

```

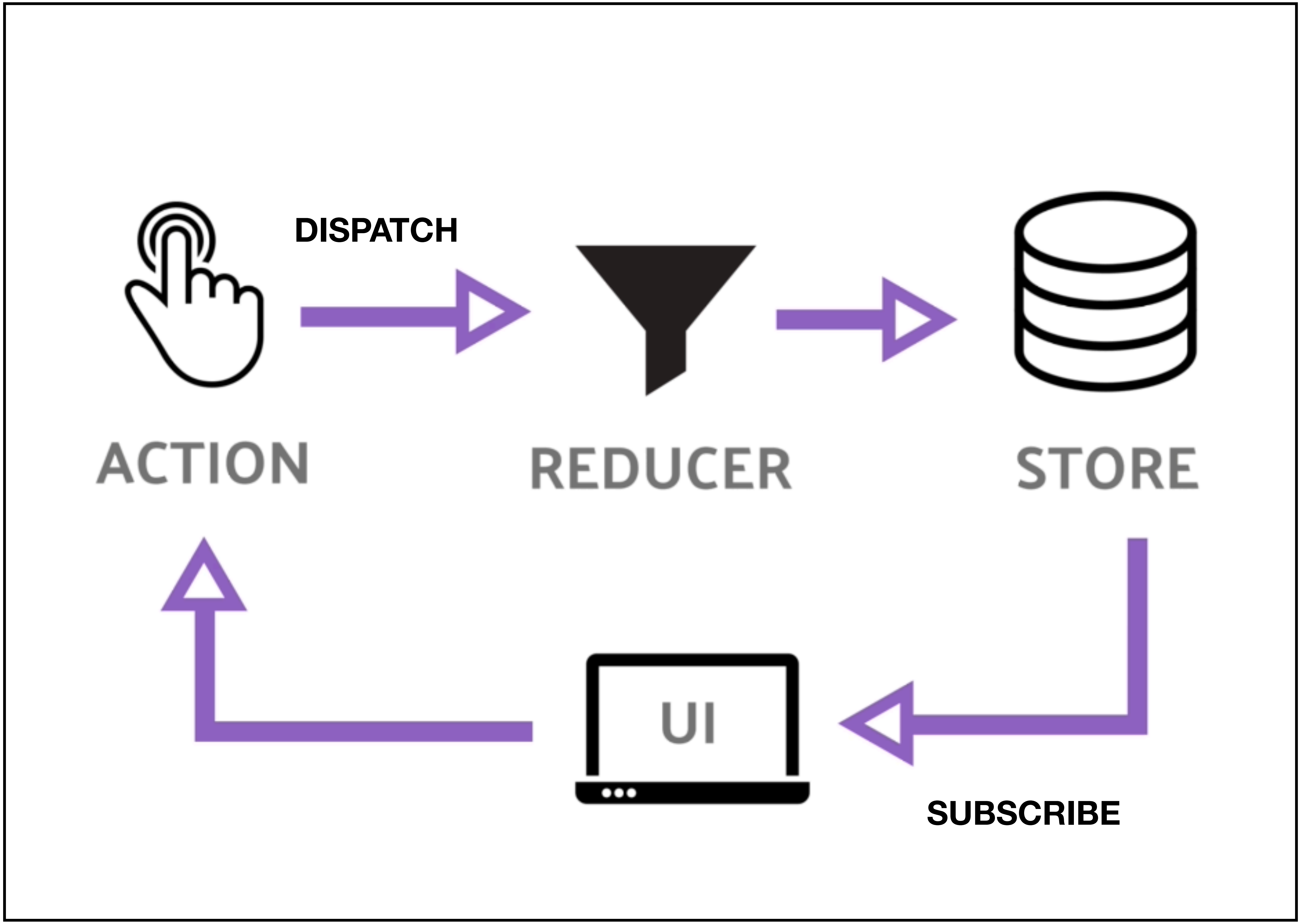
Redux

Introduction

What is Redux?

- Redux is an open-source JavaScript library for managing application state.
- It is most commonly used with libraries such as React or Angular.
- It is similar to (and inspired by) Flux architecture
- Was created by Dan Abramov and Andrew Clark.

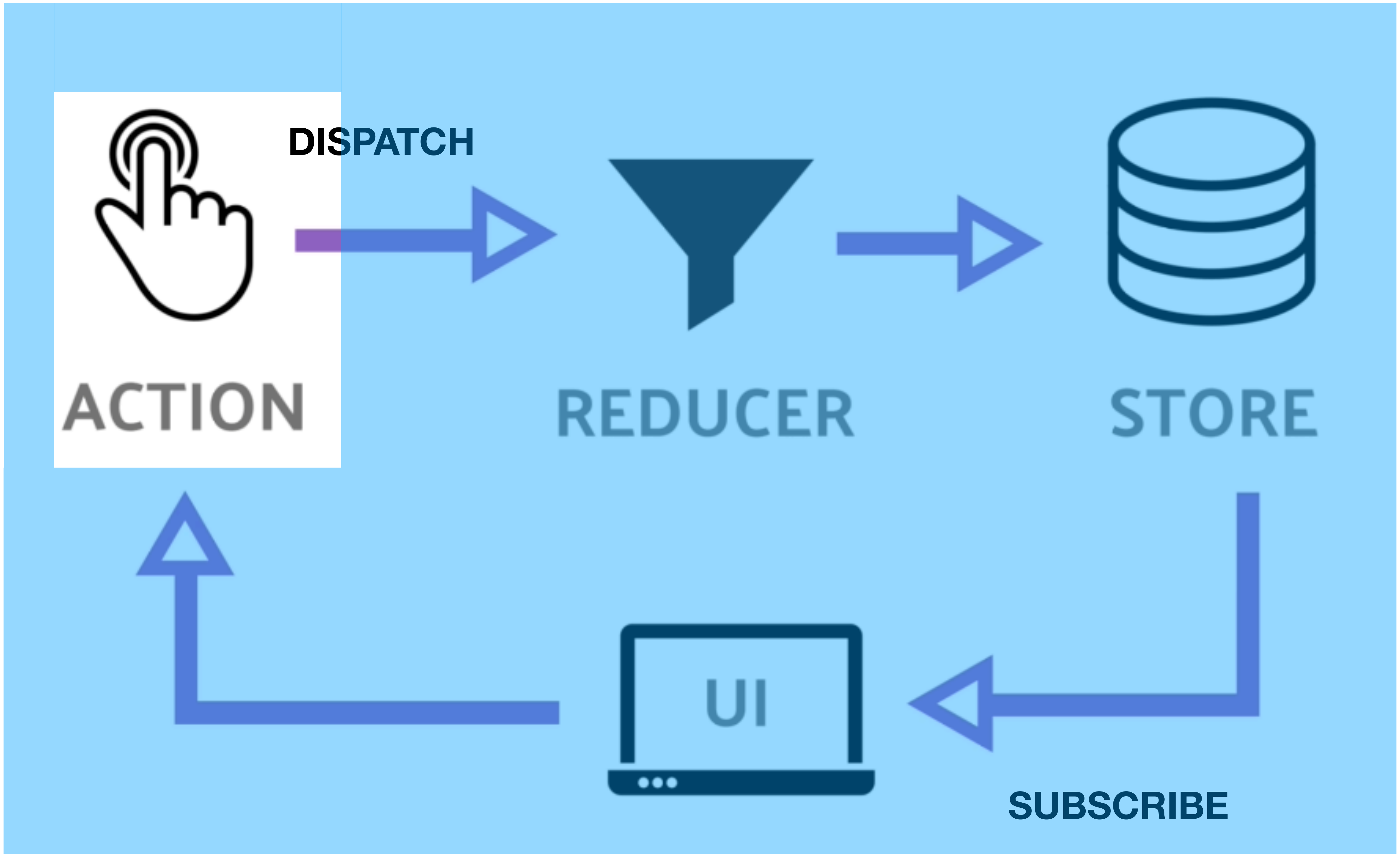

WITHOUT REDUX
WITH REDUX
STORE
COMPONENT INITIATING CHANGE


DISPATCH
ACTION
REDUCER
STORE
UI
SUBSCRIBE

Redux Architecture

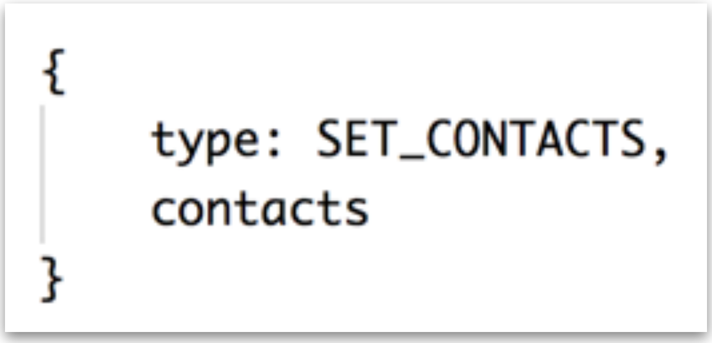
Key pointers

- Actions / Action creators
- Reducers
- Store
- Store (reducers + action) — Component relationship

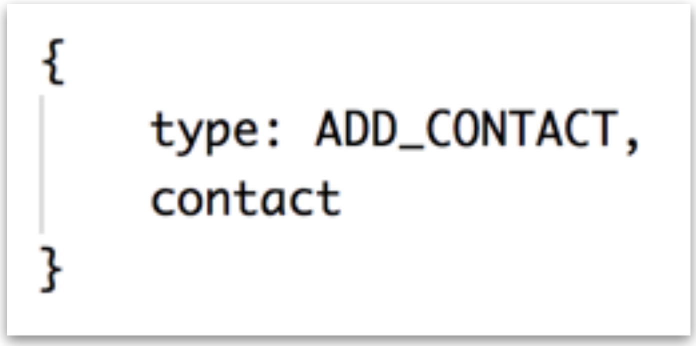

DISPATCH
ACTION
REDUCER
STORE
UI
SUBSCRIBE

Actions:

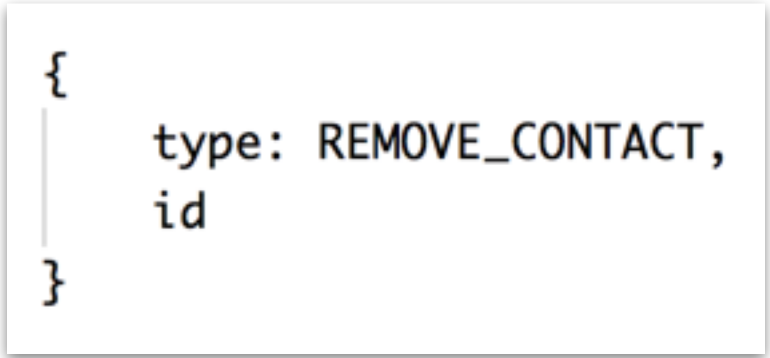
- Actions are payloads of information that send data from your application to your store.
- They are the only source of information for the store.
- You send them to the store using **store.dispatch()**.
    - This is taken care by react-redux bindings in a React application



```
{
    type: SET_CONTACTS,
    contacts
}
```



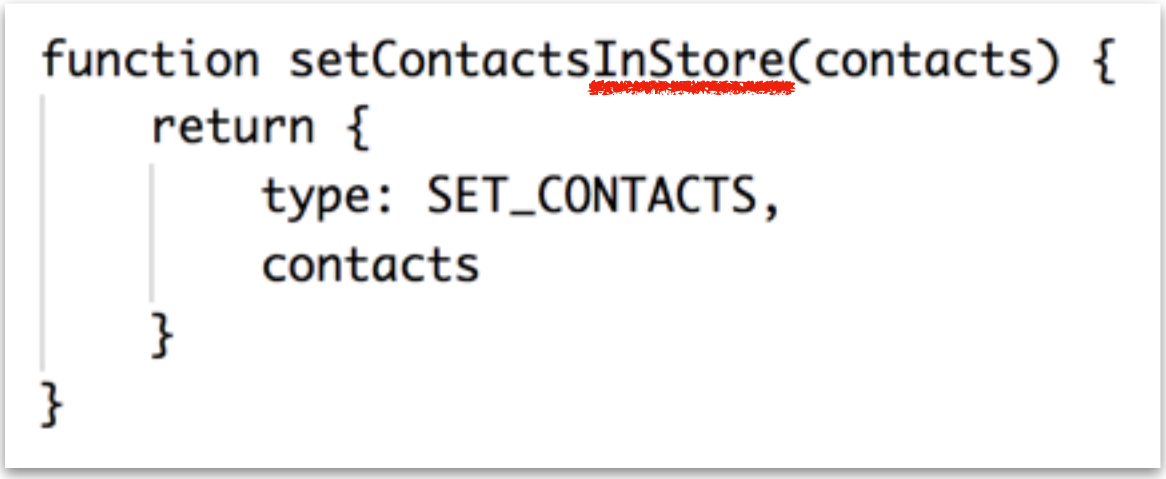
```
{
    type: ADD_CONTACT,
    contact
}
```



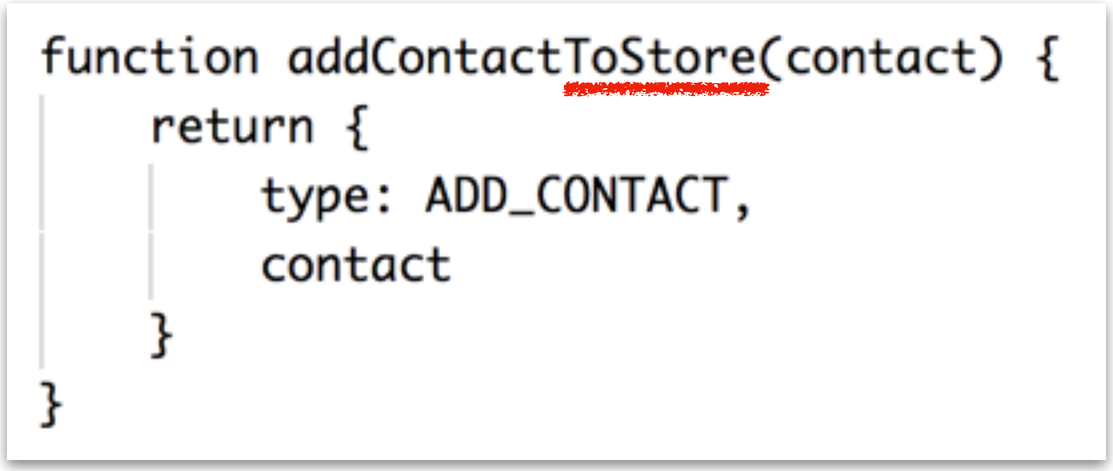
```
{
    type: REMOVE_CONTACT,
    id
}
```

Action Creators:

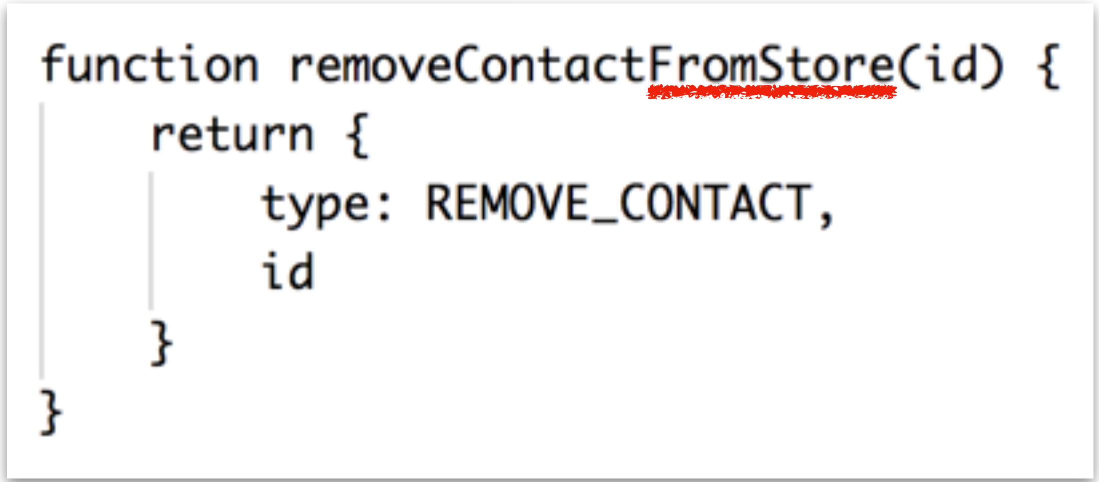
- Action creators are exactly that - functions that create actions.
- It's easy to conflate the terms “action” and “action creator”, so do your best to use the proper term.



```
function setContactsInStore(contacts) {
    return {
        type: SET_CONTACTS,
        contacts
    }
}
```



```
function addContactToStore(contact) {
    return {
        type: ADD_CONTACT,
        contact
    }
}
```

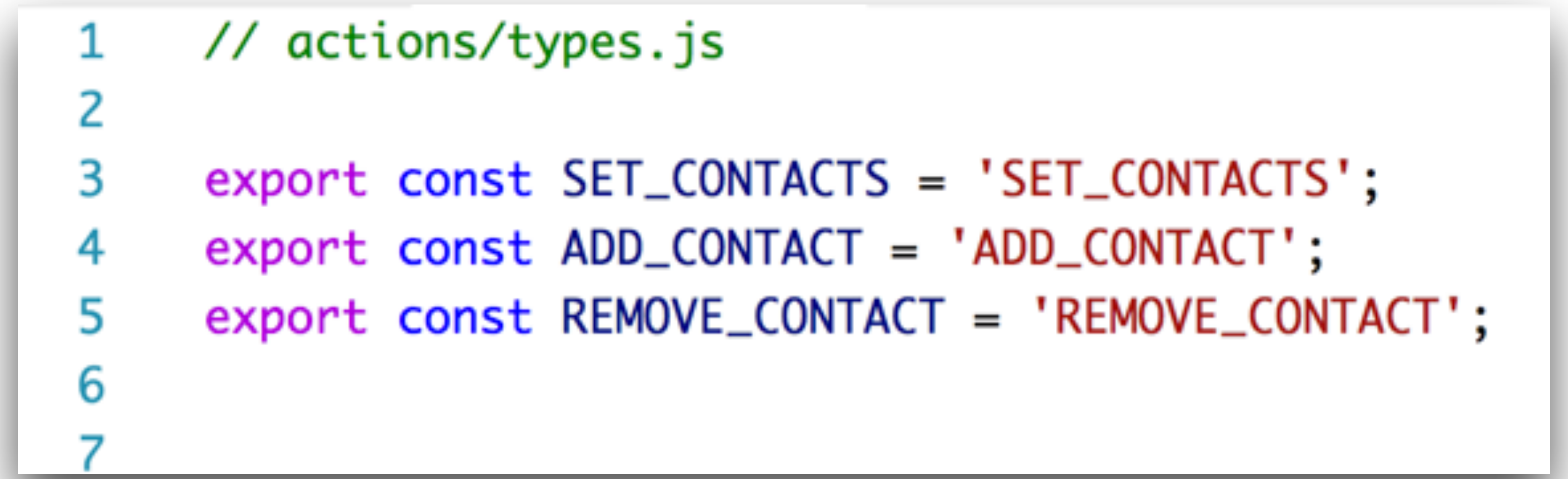


```
function removeContactFromStore(id) {
    return {
        type: REMOVE_CONTACT,
        id
    }
}
```

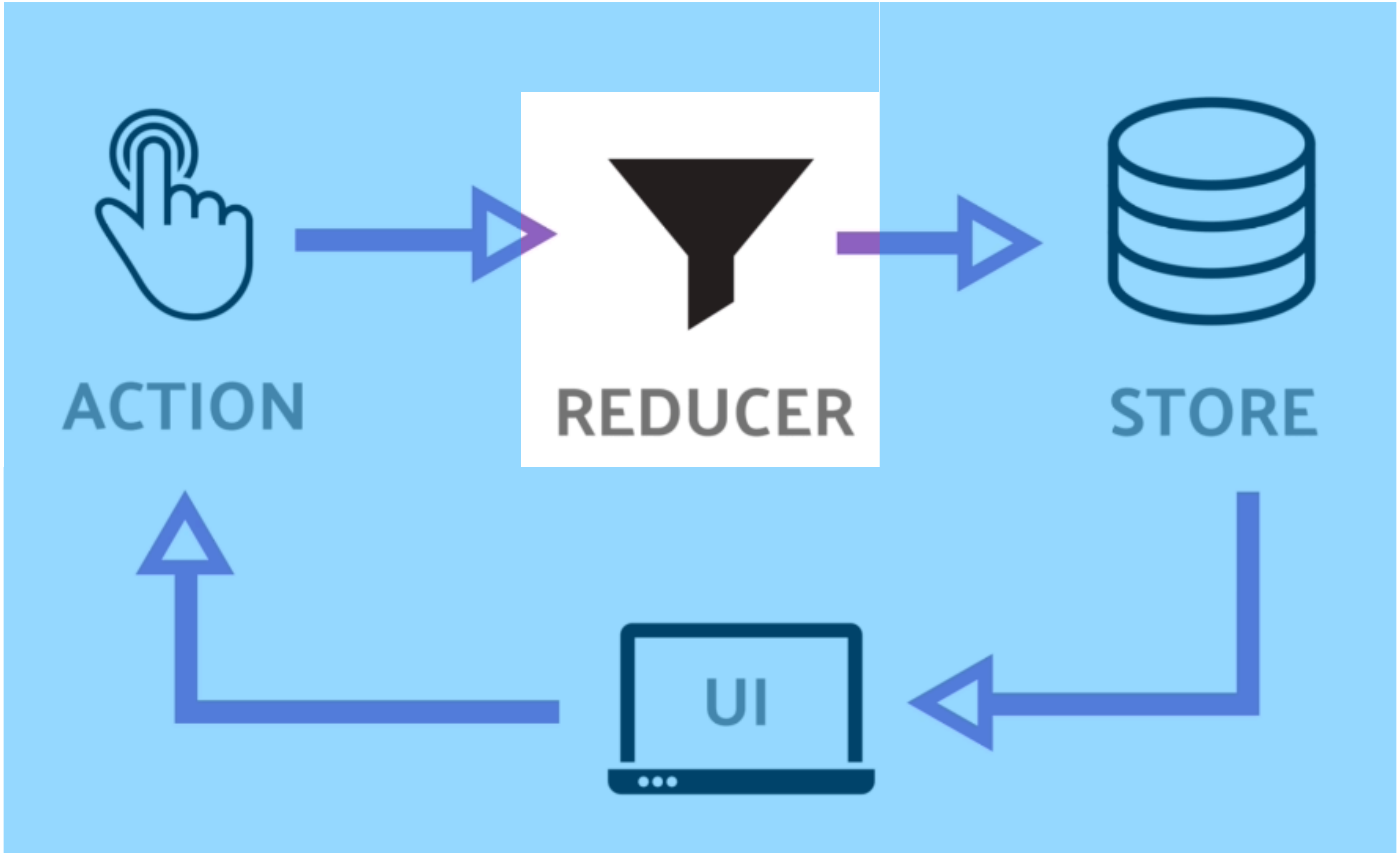
* not necessary

Action Types:

- Constants (usually string)
- Indication of the type of action
- Exported from a module (/actions/types.js)
- Typically UPPER_CASE



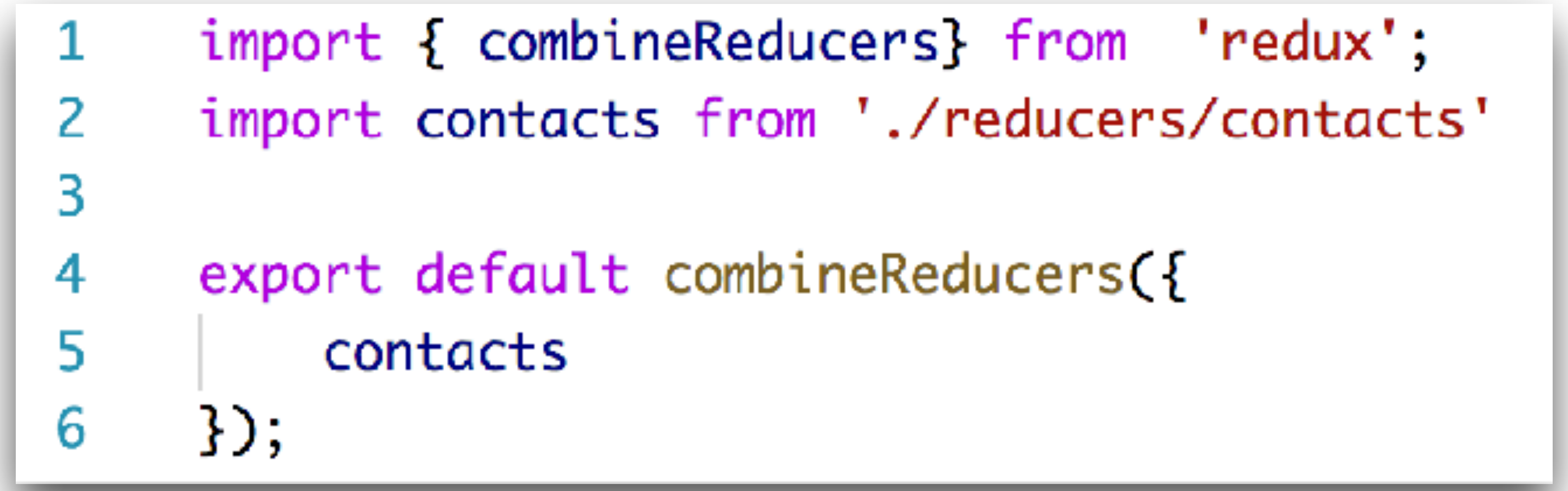
```
// actions/types.js

export const SET_CONTACTS = 'SET_CONTACTS';
export const ADD_CONTACT = 'ADD_CONTACT';
export const REMOVE_CONTACT = 'REMOVE_CONTACT';

```


ACTION
REDUCER
STORE
UI

Reducers:

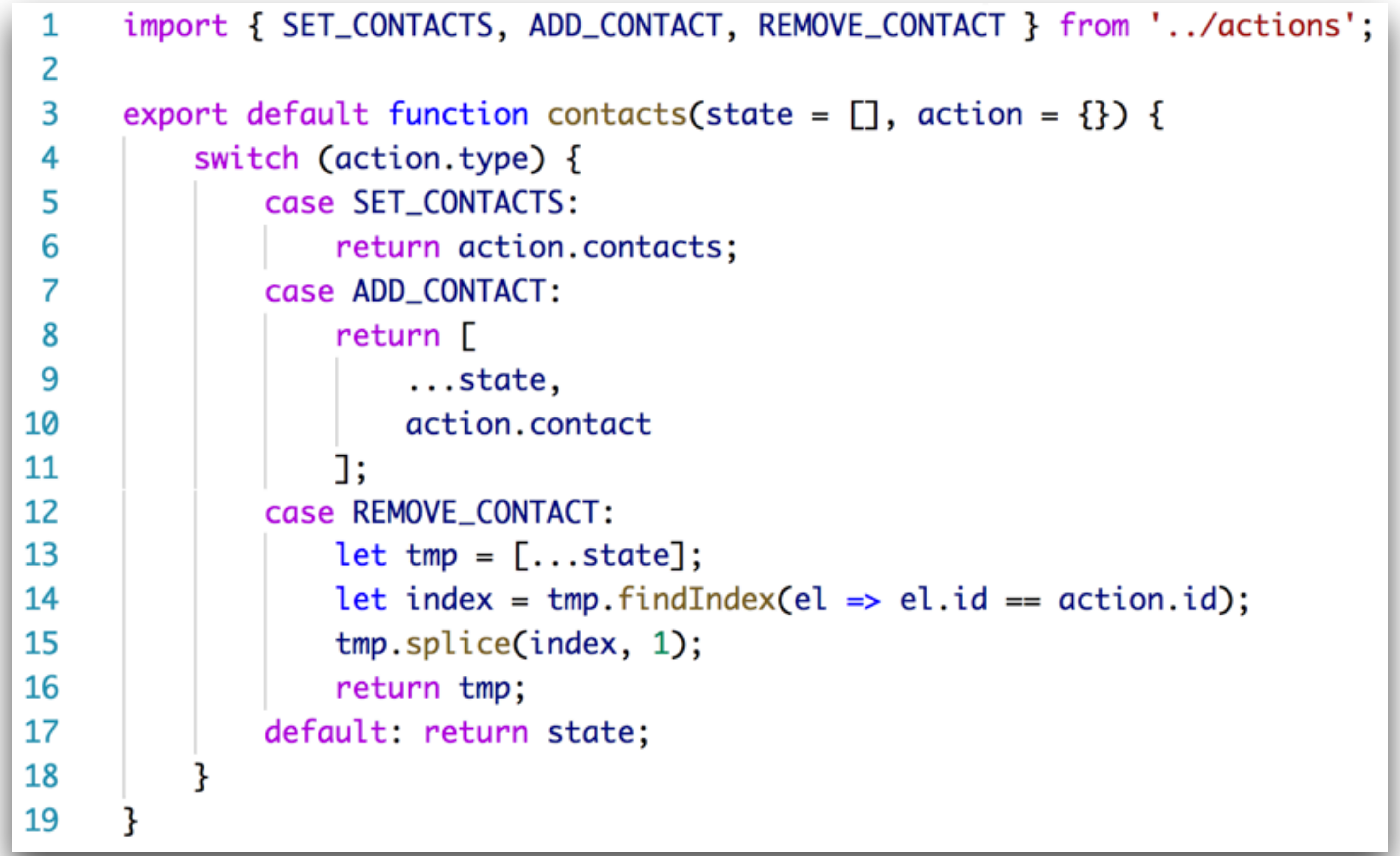
- Reducers specify how the application's state changes in response to actions sent to the store.
- Remember that actions only describe what happened, but don't describe how the application's state changes.
- One or more reducers are combined together before giving it to the store

rootReducer.js

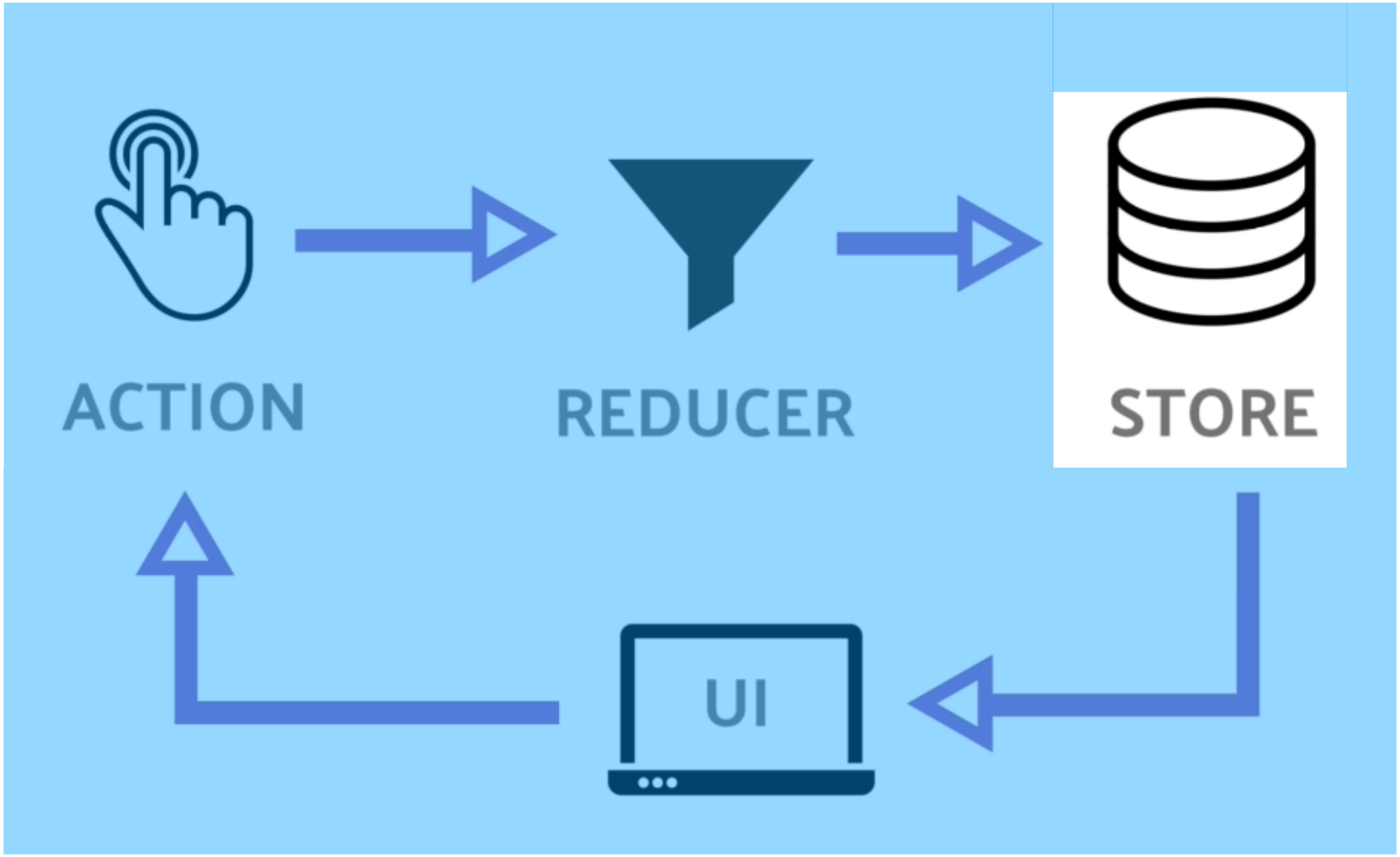


```
import { combineReducers} from  'redux';
import contacts from './reducers/contacts'

export default combineReducers({
    contacts
});
```

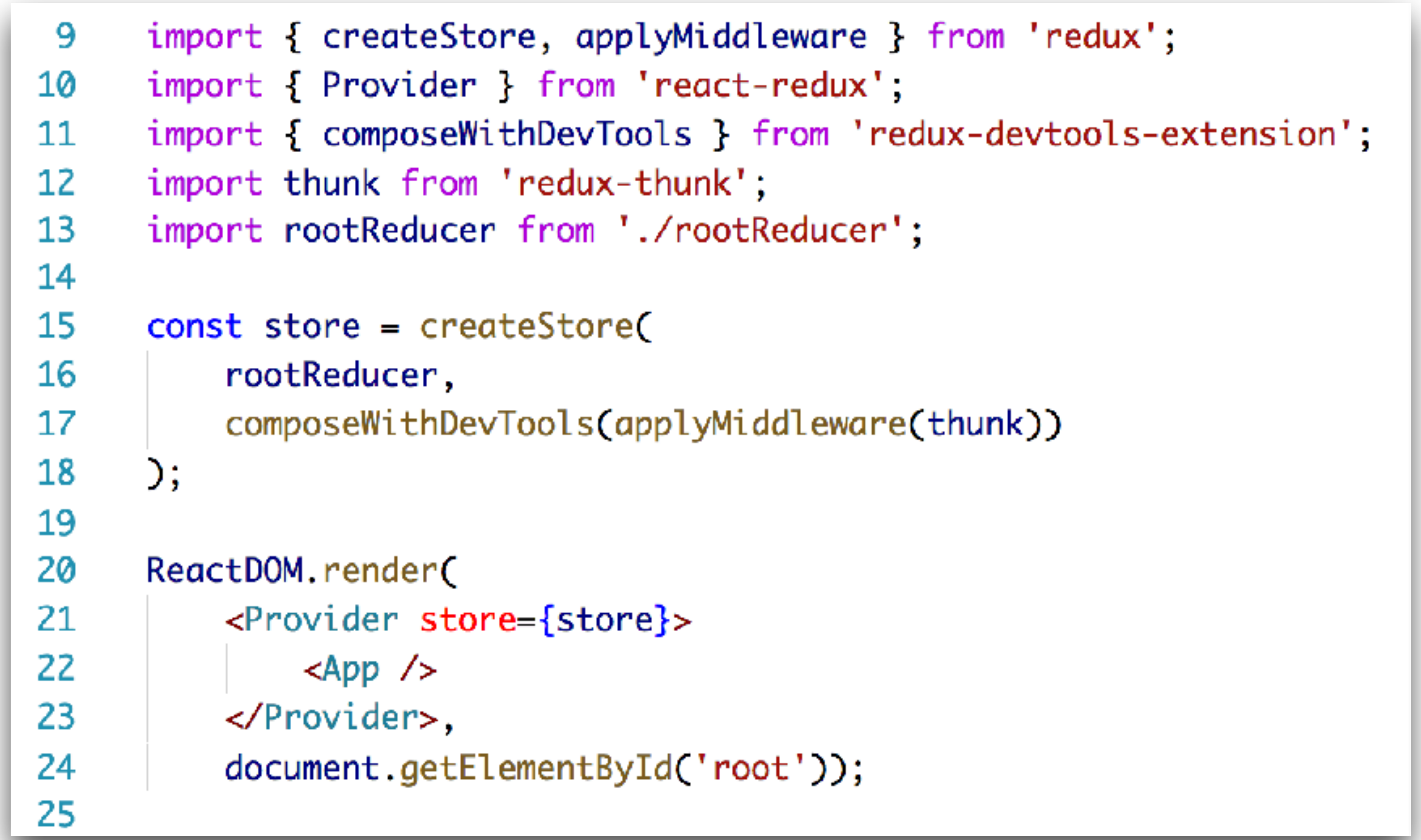
reducers/contact.js



```
import { SET_CONTACTS, ADD_CONTACT, REMOVE_CONTACT } from '../actions';

export default function contacts(state = [], action = {}) {
    switch (action.type) {
        case SET_CONTACTS:
            return action.contacts;
        case ADD_CONTACT:
            return [
                ...state,
                action.contact
            ];
        case REMOVE_CONTACT:
            let tmp = [...state];
            let index = tmp.findIndex(el => el.id == action.id);
            tmp.splice(index, 1);
            return tmp;
        default: return state;
    }
}
```

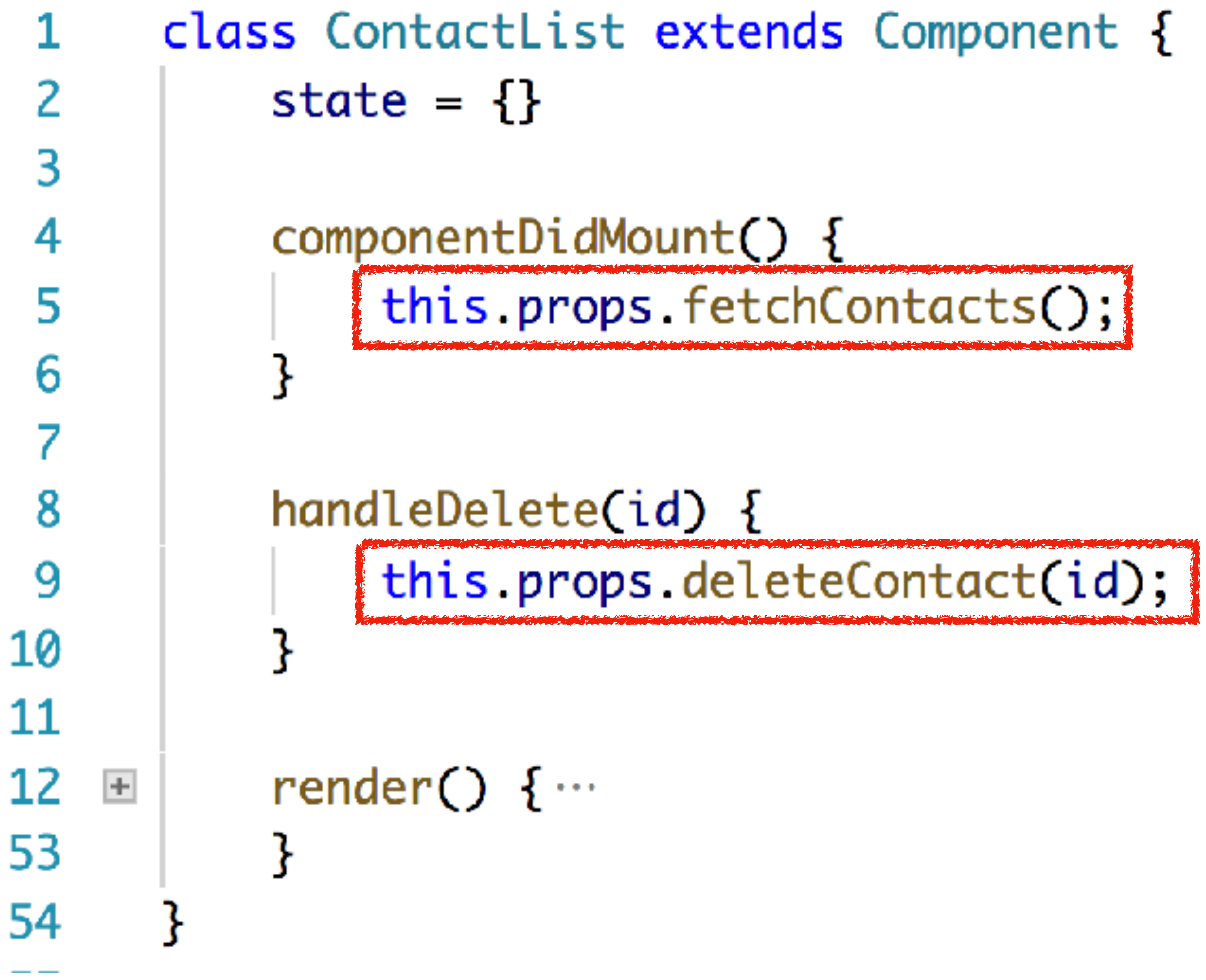

ACTION
REDUCER
STORE
UI

Store



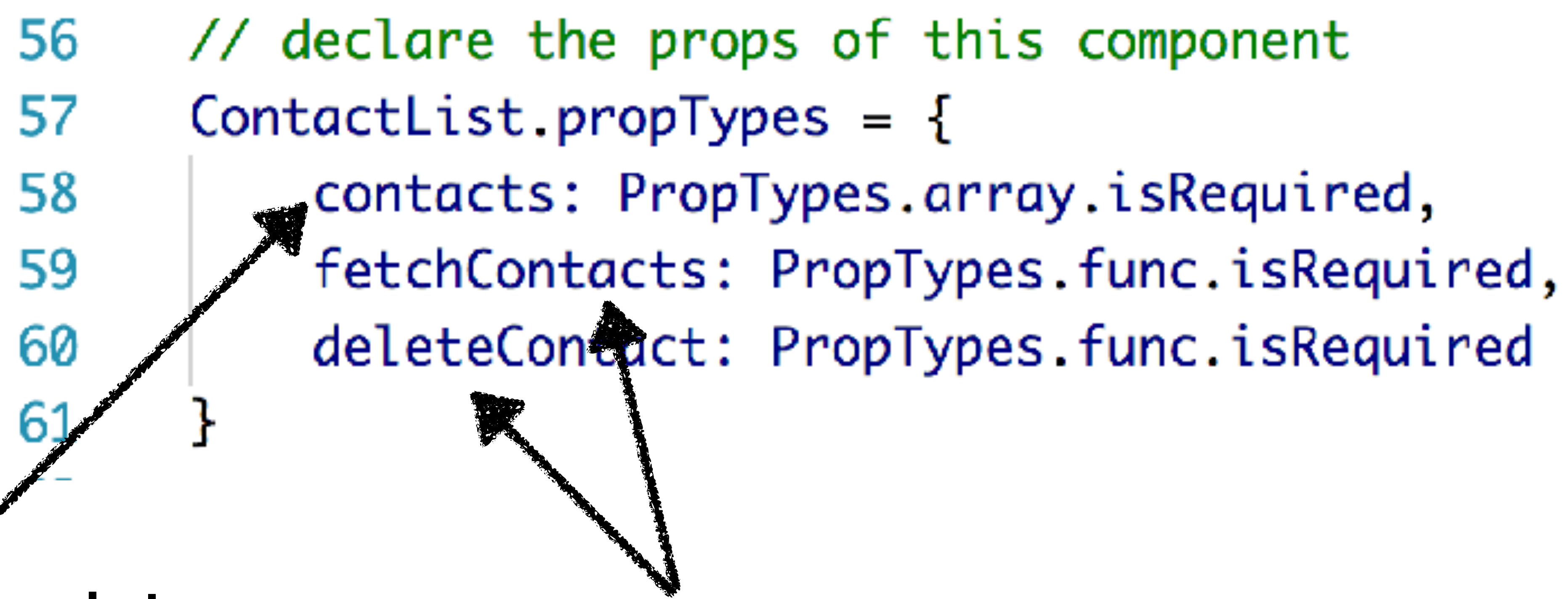
```
import { createStore, applyMiddleware } from 'redux';
import { Provider } from 'react-redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import thunk from 'redux-thunk';
import rootReducer from './rootReducer';

const store = createStore(
    rootReducer,
    composeWithDevTools(applyMiddleware(thunk))
);

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

```

Component / store (reducer+action) relationship



```
class ContactList extends Component {
    state = {}

    componentDidMount() {
        this.props.fetchContacts();
    }

    handleDelete(id) {
        this.props.deleteContact(id);
    }

    render() {…
    }
}
```

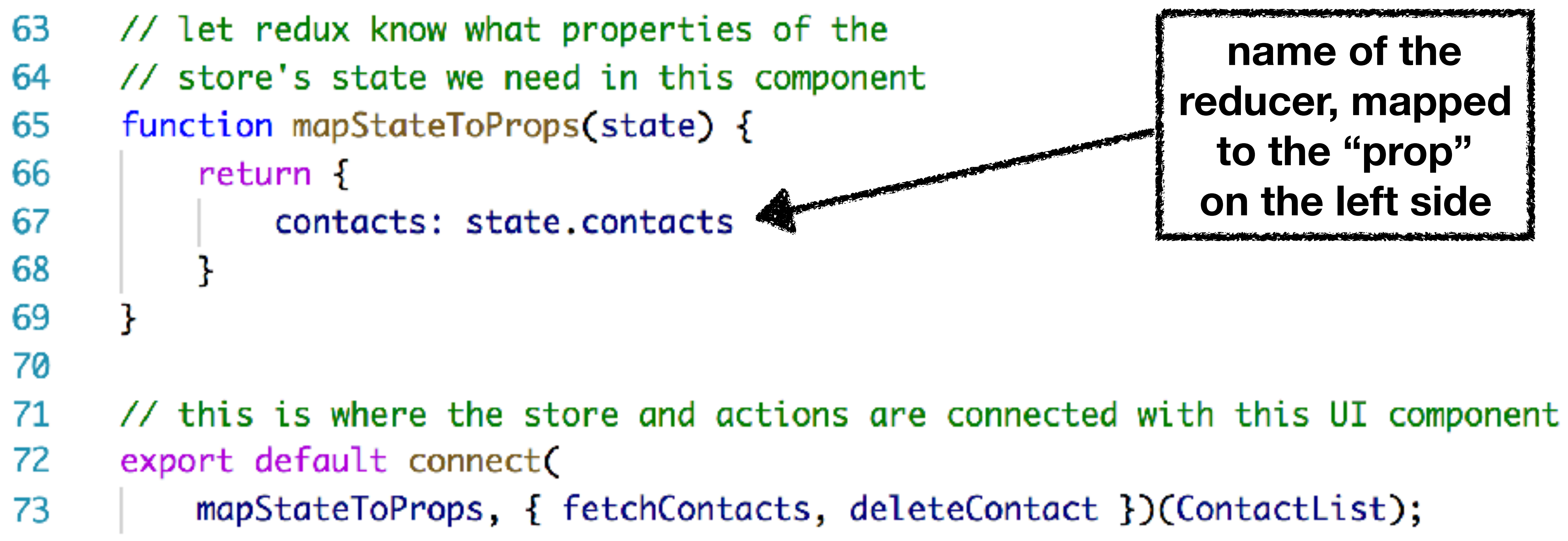
Optionally declare component's props



```
// declare the props of this component
ContactList.propTypes = {
    contacts: PropTypes.array.isRequired,
    fetchContacts: PropTypes.func.isRequired,
    deleteContact: PropTypes.func.isRequired
}
```

corresponds to
the reducer's
return value

correspond to
the action
creators



```
// let redux know what properties of the
// store's state we need in this component
function mapStateToProps(state) {
    return {
        contacts: state.contacts
    }
}

// this is where the store and actions are connected with this UI component
export default connect(
    mapStateToProps, { fetchContacts, deleteContact })(ContactList);
```

The "connect" function takes two parameters:
- **A function that maps state to props**
- **The list of action creators needed in the component**

The return value of "connect" is a Higher-Order-Component that wraps the actual component, binding REDUX with the COMPONENT