Redux

What is Redux?

Redux is a state container for JS apps.

To install redux for your app run:

```
yarn add redux react-redux
```

Why use Redux?

React state is stored locally within a component.

When it needs to be shared with other components it is passed down through props.

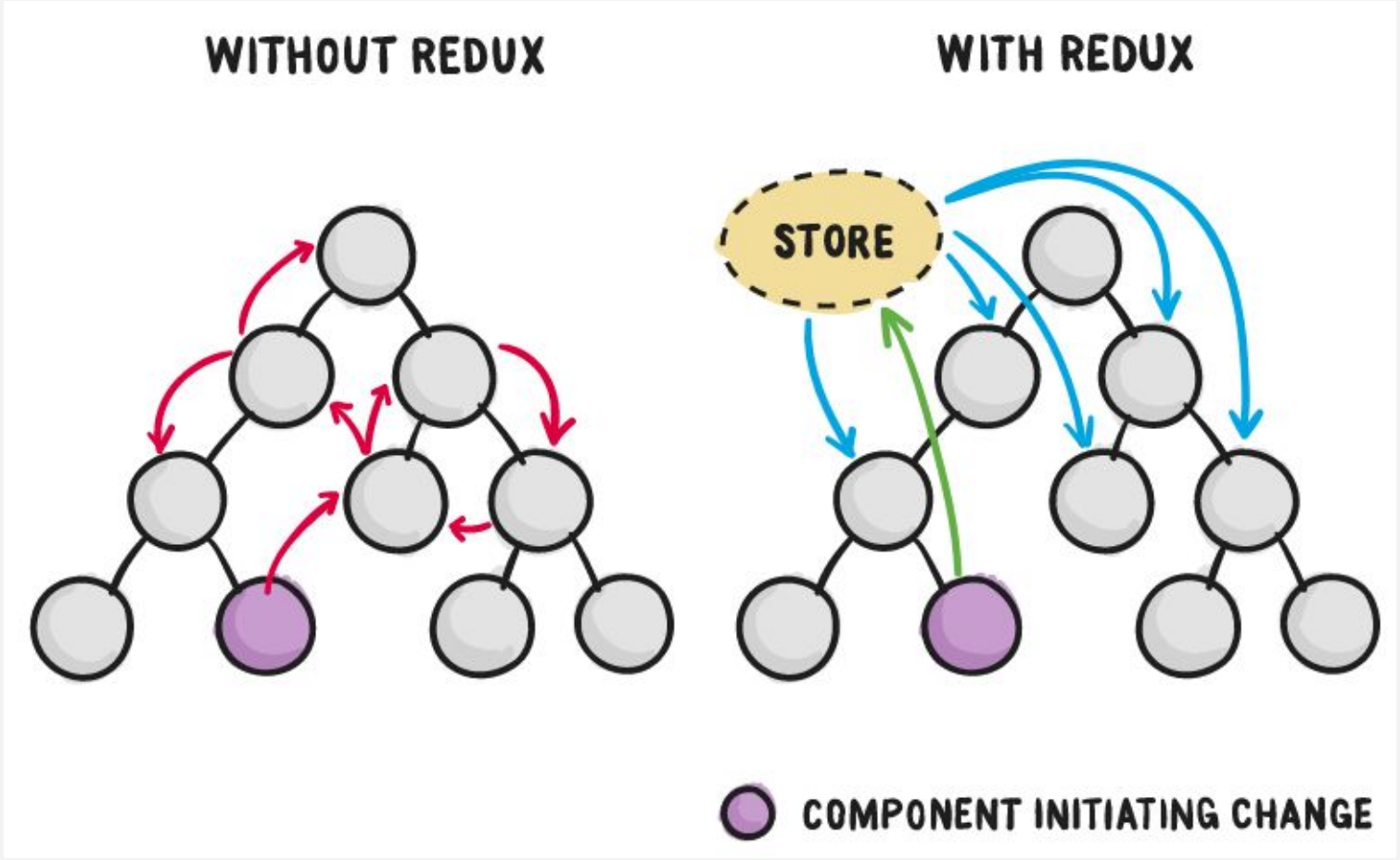
Non-redux way: Passing down props between components

```
class Parent extends React.Component {
    render() {
        return (
            <Child example="foo" />
        )
    }
}

class Child extends React.component {
    render() {
        return (
            <h1>{this.props.example}</h1>
        )
    }
}
```

Why use Redux?

When using Redux state is stored globally in the Redux store.


WITHOUT REDUX
WITH REDUX
STORE
COMPONENT INITIATING CHANGE

Actions

Actions are payloads of information which send data from your component to your store.

```
const ADD_TODO = 'ADD_TODO'
```

```
{
  type: ADD_TODO,
  text: 'Build my first Redux app'
}
```

Action Creators

Action creators return an action. You will need to use this to pass your data into an action.

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

Reducers

Reducers determine how an application's state will change in response to different action types.

```
function todos(state = [], action) {
  switch (action.type) {
    case ADD_TODO:
      return [
        ...state,
        {
          text: action.text,
          completed: false
        }
      ]
    case TOGGLE_TODO:
      return state.map((todo, index) => {
        if (index === action.index) {
          return Object.assign({}, todo, {
            completed: !todo.completed
          })
        }
        return todo
      })
    default:
      return state
  }
}
```

Store

The store holds the whole state tree of your application.

To create a store pass in your reducers to createStore.

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp)
```

Store

The store has 3 methods:

- getState() => allows access to state
- dispatch(action) => allows state to be updated
- subscribe(listener) => registers listener

```
// Dispatch some actions
store.dispatch(addTodo('Learn about actions'))
store.dispatch(addTodo('Learn about reducers'))
store.dispatch(addTodo('Learn about store'))
```

Containers

Alternatively, you can use a container to connect your rendered component to the store.

Containers

- A container is a React component that uses store.subscribe() to read the Redux state tree and pass in props to a presentational component it renders.
- You will need to use React Redux’s connect() method to assign the props to the component you’re planning to render.
- To assign dispatchers to props you will need to use mapDispatchToProps() and to assign state props you will need to use mapStateToProps()

```
const mapStateToProps = state => {
  return {
    todos: getVisibleTodos(state.todos, state.visibilityFilter)
  }
}

const mapDispatchToProps = dispatch => {
  return {
    onTodoClick: id => {
      dispatch(toggleTodo(id))
    }
  }
}

const VisibleTodoList = connect(
  mapStateToProps,
  mapDispatchToProps
)(TodoList)
```

```
import React from 'react'
import PropTypes from 'prop-types'
import Todo from './Todo'

const TodoList = ({ todos, onTodoClick }) => (
  <ul>
    {todos.map(todo => (
      <Todo key={todo.id} {...todo} onClick={() => onTodoClick(todo.id)} />
    ))}
  </ul>
)
```

Hooking Redux to your application

index.js

```
import React from 'react'
import { render } from 'react-dom'
import { Provider } from 'react-redux'
import { createStore } from 'redux'
import todoApp from './reducers'
import App from './components/App'

let store = createStore(todoApp)

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