HANDLING STATE WITH

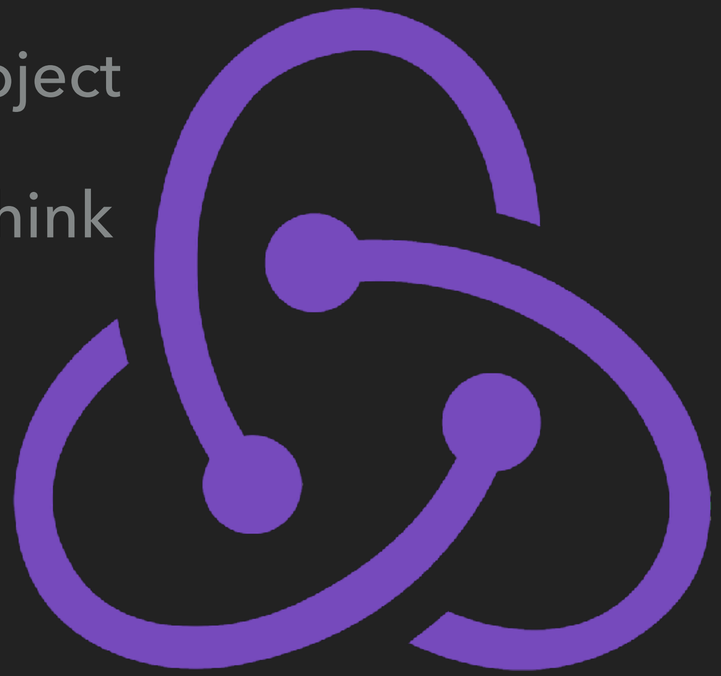
REDUX

LEARNING OBJECTIVES

- Understand data flow in Redux
  - app-wide state vs component-wide state
- Understand it's purpose and use cases
- Learn how to integrate Redux with React
- Practice using Redux in class

INTRO TO REDUX

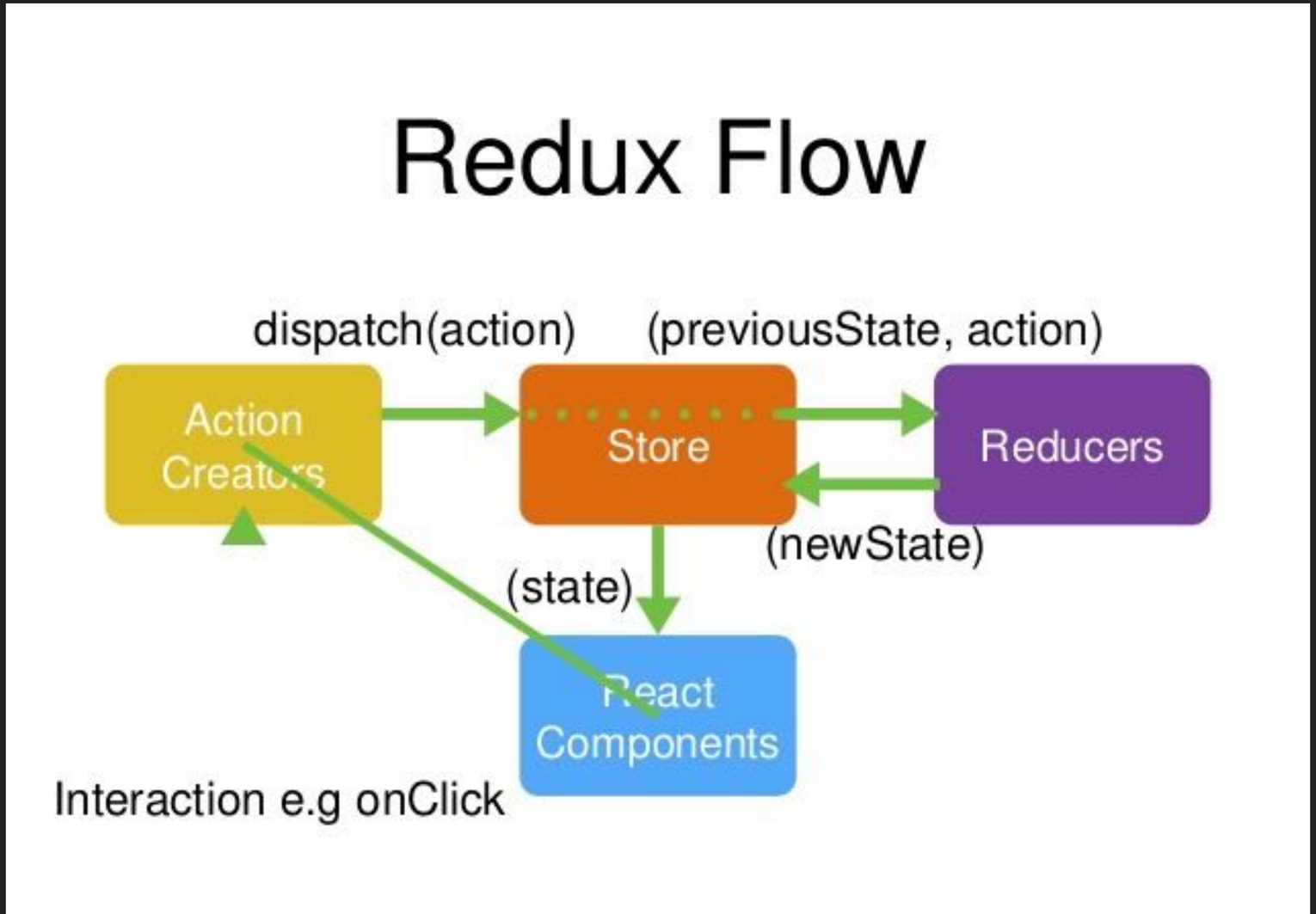
- Redux is a state container for JavaScript apps
- It allows us to store the state of an entire app in one object
- The way that Redux manages state makes it easier to think about complex applications
- It is a lightweight framework, which allows it to sit nicely with other technologies such as React



3 PRINCIPLES

- 1. Single Source of Truth
  - The state of your whole application is stored in an object tree within a single “store”.
- 2. State is Read Only
  - The only way to change the state is to emit an action, an object describing what happened.
- 3. Changes are made with pure functions
  - To specify how the state tree is transformed by actions, you write reducers.

REDUX DATA FLOW



ACTIONS & ACTION CREATORS

- Actions plain Javascript objects.
- They are payloads of information that send data from your application to your store. They are the only source of information for the store.
- Think of an action as a very brief snippet of news. “Mary liked article 42.” or “‘Read the Redux docs.' was added to the list of todos.”

ACTIONS & ACTION CREATORS

- Action Creators are functions that create Actions

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

REDUCERS

- Actions describe the fact that something happened, but don't specify how the application's state changes in response. This is the job of a reducer.
- Reducers take in an action, and the current state of the application as arguments. They modify the store and return the updated application state
- Because our app can have many pieces of state (users, posts, etc), we can can many Reducers, but for this app we will just use one.

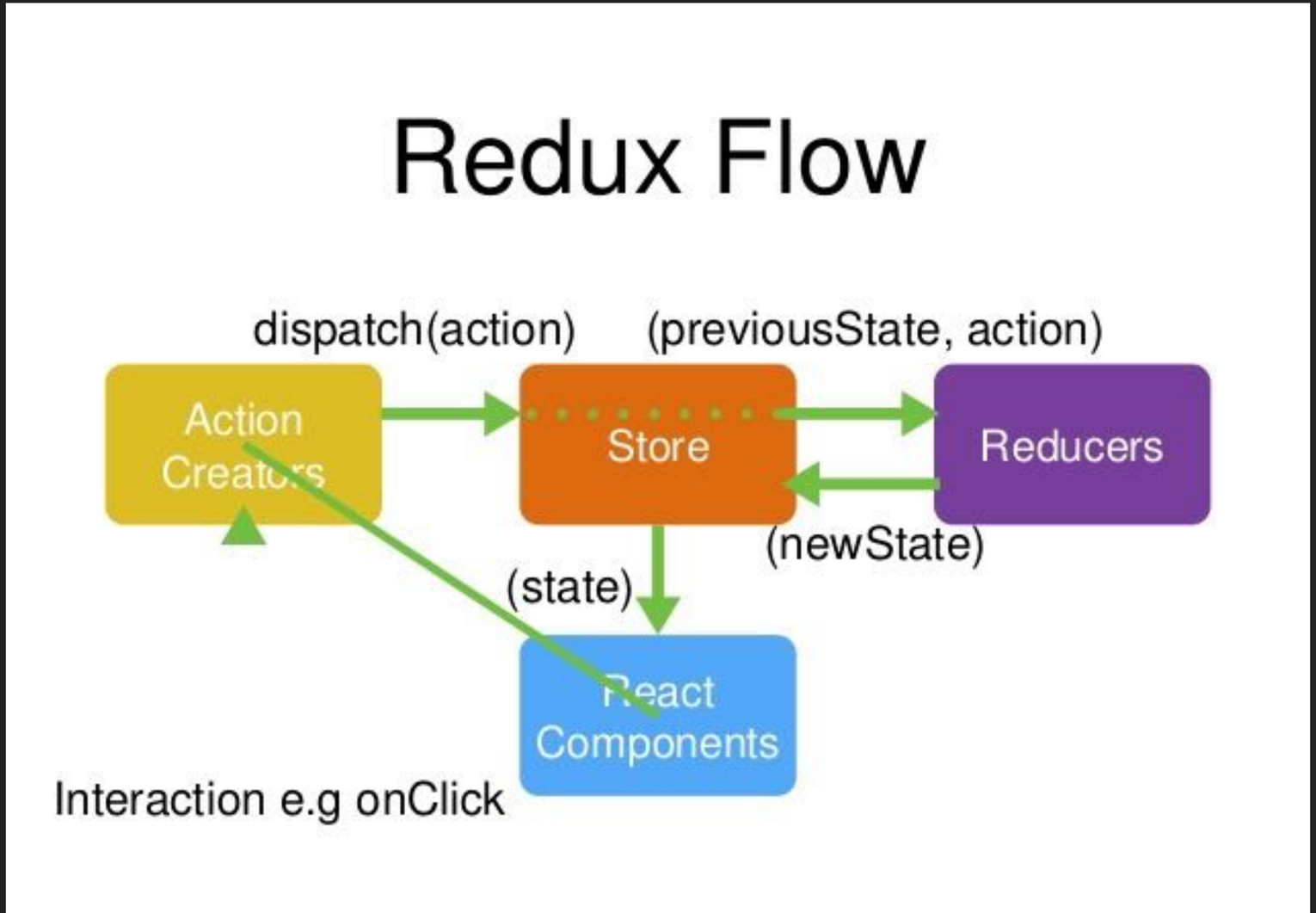
STORE

- In the previous sections, we defined the actions that represent the facts about “what happened” and the reducers that update the state according to those actions.
- The Store brings actions and reducers together and is responsible for:
  - Holding the application state
  - Allowing access to state
  - Allowing state to be updated

REDUX DATA FLOW

- 1. A view calls an action, which is just a function that describes what happened
- 2. The store calls the a reducer function, giving it the action that was called and the the current state of the app
- 3. The reducer manages the current state based on the action
- 4. When the store changes, the view updates with the changes

REDUX DATA FLOW



LET'S BUILD A SMALL REACT APP TO USE REDUX WITH

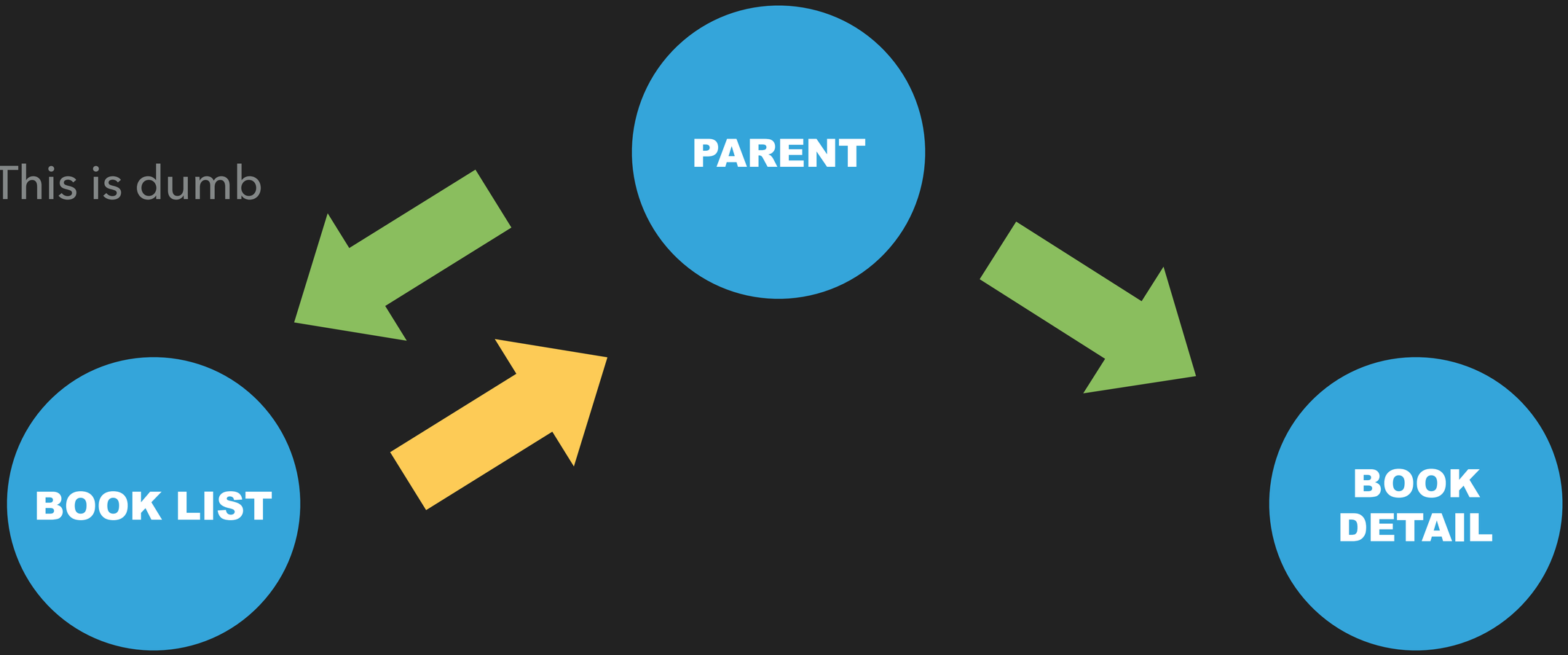
- Please pull from upstream
- You will find a React app that has already been started for you.
- Your goal is to add functionality to this app
  - When a user clicks on a book from the book list, the current book should change to that book.
  - Use the state of the book list component to update the current book

WHAT IS ANNOYING ABOUT THIS SET UP?

- We have to set state in a parent container
- A child component needs to invoke a state change in the parent, so that the other component can update its data
- We are passing data everywhere
- The child components are dependent on the parent and each other for functionality
- This is dumb

WHAT IS ANNOYING ABOUT THIS SET UP?

This is dumb



WHAT IF THERE WAS SOMETHING TO MAKE THIS BETTER....

- What could it be?
- What could be so amazing?
- Like Thanksgiving for every meal?
- Could it be?
- The cheese to your macaroni?
- Yes it really is!
- REDUX!

NPM!

- Run npm install — save redux react-redux react-dom
- BAM!

REDUX SETUP

- We need to tell our app to use Redux, and set up the store. In the /src/index.js file:

```
import React from 'react';
import ReactDOM from 'react-dom';
import { Provider } from 'react-redux';
import { createStore, applyMiddleware } from 'redux';

import App from './App';
import reducers from './reducers';

const createStoreWithMiddleware = applyMiddleware()(createStore);

ReactDOM.render(
  <Provider store={createStoreWithMiddleware(reducers)}>
    <App />
  </Provider>
  , document.querySelector('#root'));
```

REDUCING ALL BOOKS

- First, we should include all the books in our Redux state
- Create a folder in /src called /reducers
- Create 2 files in /reducers called index.js and books_reducer.js
- index.js will be our root reducer, books_reducer.js will be a reducer for maintaining all of the books

INDEX.JS REDUCER

```
import { combineReducers } from 'redux';
import BooksReducer from ‘./books_reducer';

const rootReducer = combineReducers({
  books: BooksReducer
});

export default rootReducer;
```

BOOKS_REDUCER.JS

Here, we will create our array of books:

```
export default function() {
  return[
       {
         title: "McElligot's Pool”,  pages: 64
       },
       {
         title: "I Wish That I Had Duck Feet”,  pages: 64
       },
       {
         title: "The Sneetches”, pages: 72
       },
       {
         title: "Yertle the Turtle”,  pages: 96
       }
     ]
}
```

BOOK-LIST.JS

- First, let's get rid of the componentWillMount function
- import { connect } from 'react-redux';

BOOK-LIST.JS

- function mapStateToProps(state) {
- return {
- books: state.books
- };
- } // anything returned form this function will end up as props on the booklist container
- export default connect(mapStateToProps)(BookList);
- You should now see the book list render data from the Redux store

BOOK DETAIL

- Now we need to set some things up. We want to have the Redux store determine which book is currently being viewed. To accomplish this, let's add a default active book.
- Make a new file in reducers called active_book_reducer.js

```
export default function() {
  return { title: "Yertle the Turtle",  pages: 96 }
}
```

- Let's require it, and add it to our reducers/index.js file

BOOK DETAIL

- In book-detail.js, follow a similar set up to the changes we made in book-list.js
- In App.js, remove the props sent to book-detail.js
- You should see a default book appear on the page
- BUT, we can’t change it now. Why? We are using Redux for the active book now!
- Let’s remove the selectBook function from App.js

SELECTING AN ACTIVE BOOK WITH REDUX

- The first thing we'll want to do is define an action and call it from the view
- Create a folder called actions and a file in it called index.js

```
export function selectBook(book) {
  // selectBook is an actioncreator and needs to reuturn
  // an object with a type property
  console.log(book)
  return {
    type: 'BOOK_SELECTED',
    payload: book
  };
}
```

SELECTING AN ACTIVE BOOK WITH REDUX

- Import the action into book-list.js and replace the click function
- We should be able to see each book console.log in the browser

Add to book-list: import { bindActionCreators } from ‘redux’;

```
function mapDispatchToProps(dispatch) {
  // whenever selectbook is called the result should be passed
  // to all of our reducers
  return bindActionCreators({ selectBook: selectBook }, dispatch)
}
```

add mapDispatch to connect method

SELECTING AN ACTIVE BOOK WITH REDUX

- Update our activeBook reducer:

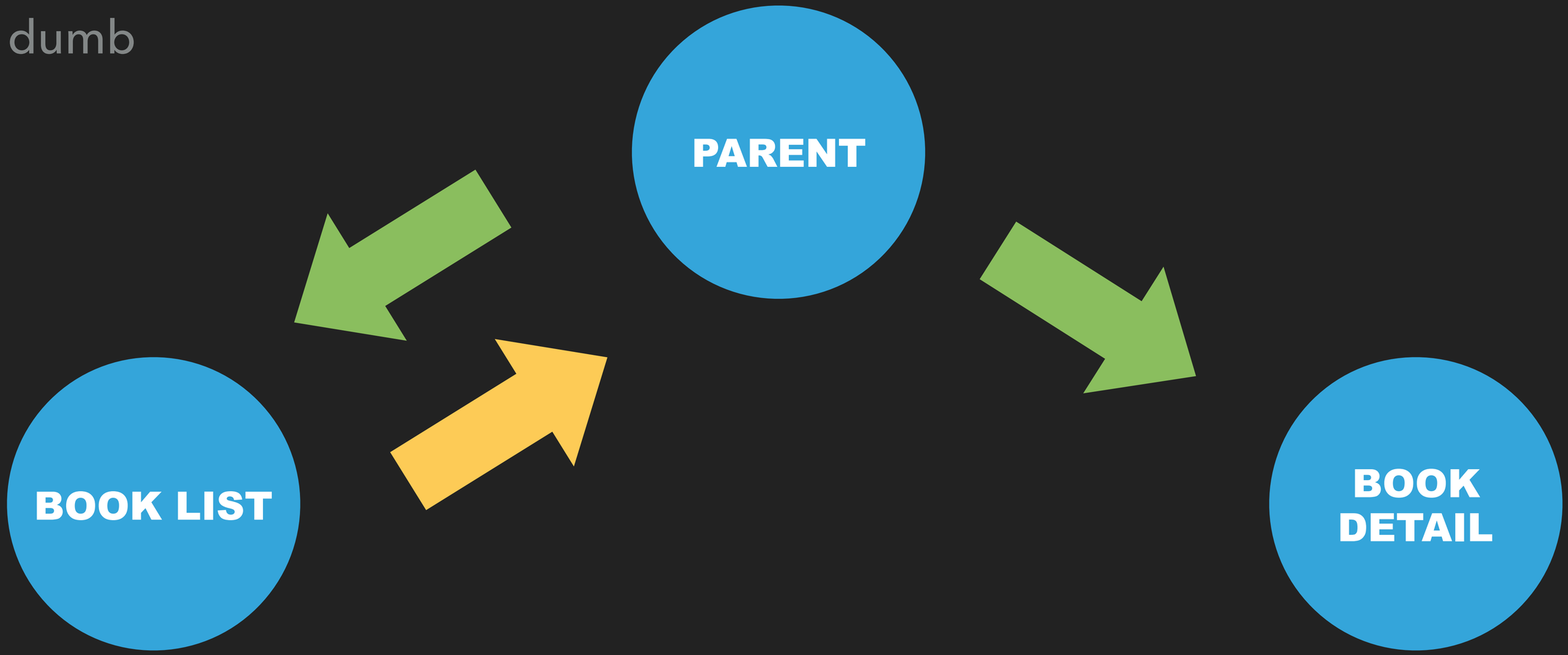
```
// state argument is not application state, only
// the state this reducer is resposible for
export default function(state = null, action) {
  switch(action.type) {
    case 'BOOK_SELECTED':
      return action.payload;
    default:
      console.log("not book selected")
      return state;
  }
}
```

IT SHOULD WORK!

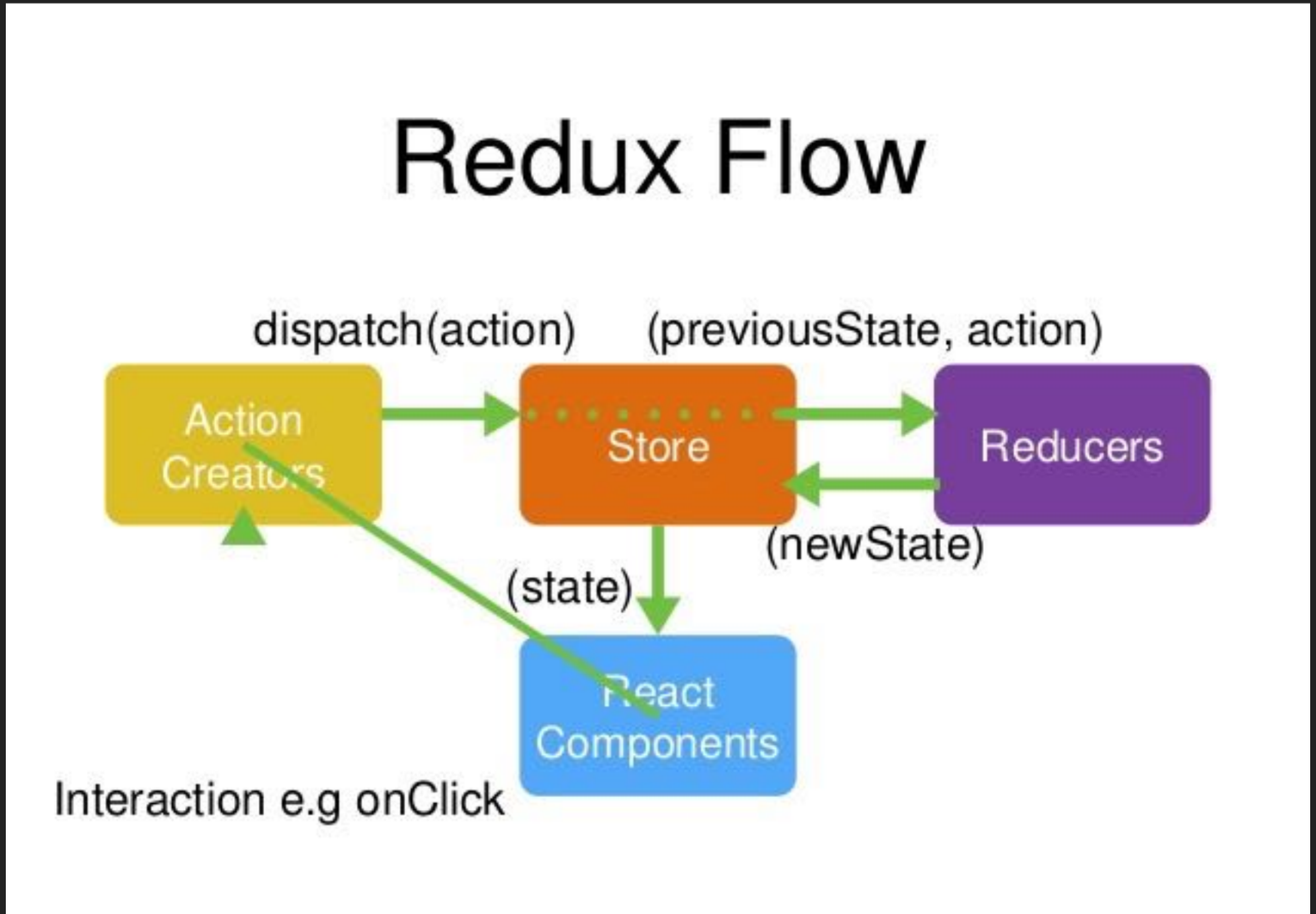
- Yay!
- Scully Scully Scully Scully
- RE-DUX RE-DUX.. okay you get the idea
- So, what just happend?

WHAT IS ANNOYING ABOUT THIS SET UP?

This is dumb



REDUX DATA FLOW



WHY REDUX IS COOL

- Redux makes React even cooler, so there's that
- Our components no longer have to deal with each other. They don't have to communicate, or be dependent on one another for changes in state.
- It's also WAY easier to think about the data in our app now
- AND makes our app very maintainable. You could keep building components and components and components on this app and you would be in good shape

WHY REDUX IS COOL

- And it has a neat logo