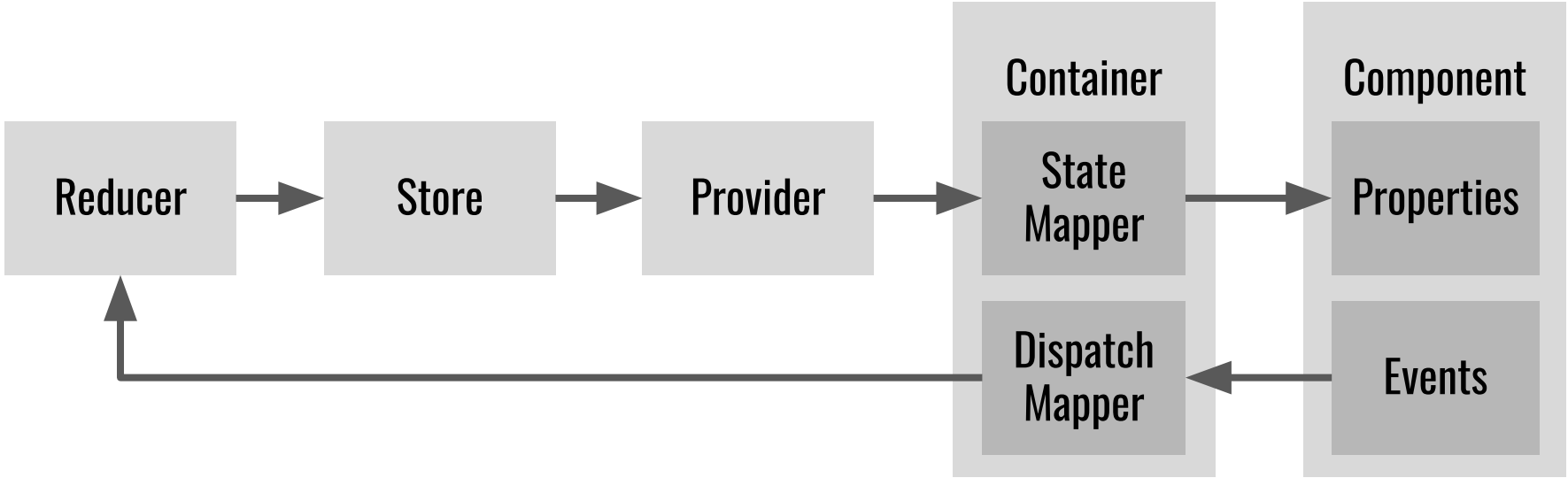
UNDERSTANDING REACT + REDUX

A Trivial Example

Dr. Jose Annunziato

Redux data flow



Install redux and create a new project

Create a trivial react application

$ create-react-app trivial-react-redux-example

Use npm to install redux for react, don’t forget to --save

$ npm install redux --save

$ npm install react-redux --save

To illustrate building an app from scratch, we can remove all content under **src/**

In src/index.js, import the necessary libraries

```
import React from 'react'
import ReactDOM from 'react-dom';

import { createStore } from 'redux'
import { Provider, connect } from 'react-redux'
```

Create some reducer to handle dispatched actions

```
const someReducer = (state =
    {someDefaultProperty: 'some state'}, action) => {
 switch (action.type) {
    case 'some action':
      alert('Some action was dispatch')
      return {someStateAttribute:
                 'some new state'}
    default: return state
}}
```

reducers create next state based on previous state + action

Render the app on the DOM

```
ReactDOM.render(
  <SomeApp />,
  document.getElementById('root')
);
```

Create the store and provide it to the application

```
const someStore =
    createStore(someReducer)

const SomeApp = () => (
  <Provider store={someStore}>
    <SomeContainer />
  </Provider>
)
```

The ***store*** is sole source of application state. ***Providers*** provide store to application. ***Reducers*** create state

Create some component dispatching some action

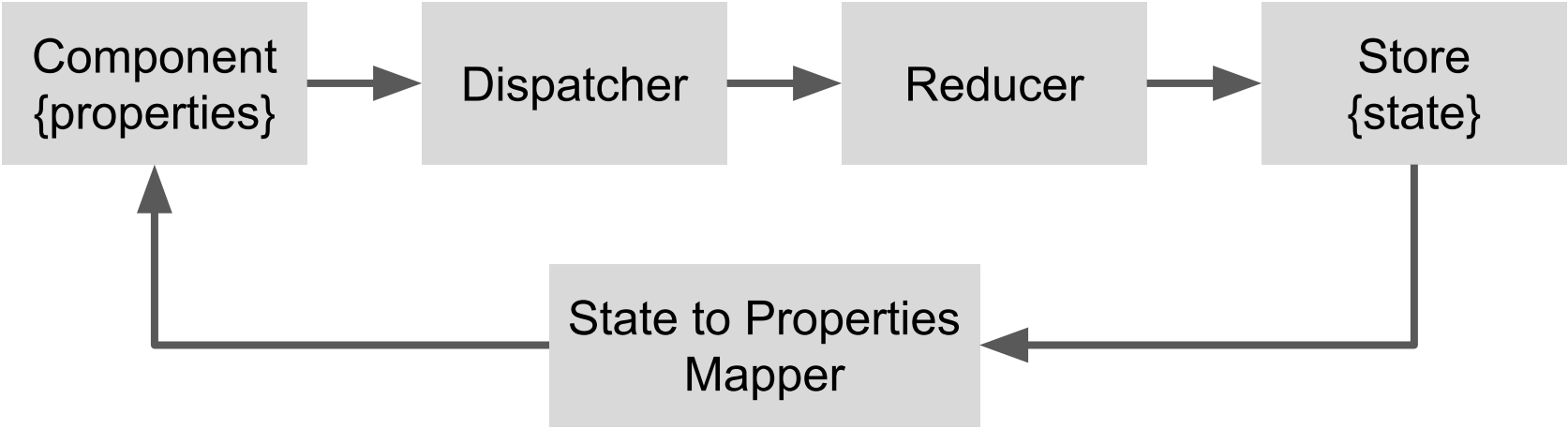
```
const SomeComponent = ({dispatch}) => (
  <button onClick={e => {
    dispatch({type: 'some action'})
  }}>Some Button</button>
)
const SomeContainer =
    connect()(SomeComponent)
```

type attribute is required. Uniquely identifies an ***action***

CONNECT REDUX STATE TO REACT COMPONENT PROPERTIES

Dr. Jose Annunziato

Redux data flow



Add some property to the component

```
const SomeComponent =
  ({someComponentProperty, dispatch}) => (
 <button onClick={e => {
  dispatch({type: 'some action'})
 }}>
  Some Button {someComponentProperty}
 </button>
)
```

Remember the reducer added attributes to state

```
const someReducer = (state =
    {someDefaultProperty: 'some state'}, action) => {
 switch (action.type) {
    case 'some action':
      alert('Some action was dispatch')
      return {someStateAttribute: 'some new state'}
    default: return state
 }
}
```

Map redux state to component properties

```
function someStateToPropsMapper(state) {
  return {
    someComponentProperty:
      state.someStateAttribute
}}

const SomeContainer = connect
(someStateToPropsMapper)(SomeComponent)
```

THAT'S IT!

That's all there is to React + Redux

WAIT! THERE'S MORE

Dr. Jose Annunziato

ACTION CREATORS

Dr. Jose Annunziato

It's best practice to wrap actions in functions

```
const someActionCreator =
    (someOptionalArgument) => {
 return {
    type: 'some action',
    someOptionalAttribute:
        'some value'
}}
```

type attribute is required. Add **other attributes** as parameters

Then actions are encapsulated

```
const SomeComponent =
   ({someComponentProperty, dispatch}) => (
 <button onClick={e => {
   dispatch(someActionCreator())
 }}>
   Some Button ...
 </button>
)
```

now component (view) doesn't need to know about action or data details

Best practice: declare all action types as const

```
const SOME_ACTION = 'some action'
const someActionCreator =
    (someArgumentIfAny) => {
 return {
    type: SOME_ACTION,
    someOptionalAttribute: 'some value'
}}
```

use meaningful action names

MAPPING REDUX DISPATCH TO REACT EVENT HANDLERS

Dr. Jose Annunziato

Recall the component dispatching some action...

```
const SomeComponent =
    ({someComponentProperty,
      dispatch}) => (
 <button
    onClick=
        {e => { dispatch(someActionCreator())}}>
            Some Button ...
 </button>)
```

Let's move the dispatch logic out from the component (view)

Declare event handlers as component properties...

```
const SomeComponent =
    ({someComponentProperty,
      someEventHandler}) => (
 <button
    onClick=
        {someEventHandler}>
            Some Button ...
 </button>)
```

...instead of dispatching from component (view).

Centralize, encapsulate dispatching

Map redux dispatchers to component properties

```
function someDispatchToPropsMapper(dispatch) {
  return {
    someEventHandler: () =>
      dispatch(someActionCreator())
}}
const SomeContainer = connect(
someStateToPropsMapper,
someDispatchToPropsMapper)(SomeComponent)
```

here's dispatch logic, but now in an object map. Redux can do mapping for us

BREAK UP LARGE REDUX APPS

Dr. Jose Annunziato

Break up applications by role

In large React + Redux applications you can break up code based on their role

Here's a typical directory structure

```
src/
    index.js        reducers/        actions/
    constants/      components/      containers/
```

Move all constants to their own folder/file

Declare all constants in **src/constants/index.js**

```
export const SOME_ACTION = 'some action'
```

In **src/index.js**, import constants you need

```
import {SOME_ACTION} from
    './constants'
```

when importing, **index.js** is default and optional

Replace literals with imported constants

```
import {SOME_ACTION} from './constants/index'

const someReducer = (...) => {
 switch (action.type) {
   case SOME_ACTION:
     return { ... }
   default:
     return state
}}
```

when importing, **index.js** is default, so it's optional

Move all actions to their own folders/files

Implement all actions in **src/actions/index.js**

```
import {SOME_ACTION} from '../constants'
export const someActionCreator =
 (someArgumentIfAny) => {
   return { type: SOME_ACTION,
     someOptionalAttribute: 'some value' }}
```

Import actions as needed in **src/index.js**

```
import {someActionCreator} from './actions'
```

index.js has been left out because it is implied

Move all reducers to their own folder/file

Implement all reducers in **src/reducers/index.js**

```
import {SOME_ACTION} from '../constants'
export const someReducer = ( … ) => {
    switch (action.type) {
        case SOME_ACTION:
            return {someStateAttribute: 'some other state'}
        default: return state}}
```

Import reducers from **src/index.js**

```
import {someReducer} from './reducers'
```

Move all components to their own folder/file

Implement each component in their own file in **src/components/*.js**

```
import React from 'react'
const SomeComponent =
    ({someComponentProperty, someEventHandler}) => (
    <button onClick={someEventHandler}>
        Some Button {someComponentProperty}
    </button>)
export default SomeComponent
```

Move all containers into their own folder/file

Implement all containers in **src/containers/*.js**
Move **SomeContainer** into **src/containers/SomeContainer.js**

The containers need the actions and components

```
import React from 'react'
import { connect } from 'react-redux'
import SomeComponent from '../components/SomeComponent'
import someActionCreator from '../actions'
```

Move all containers into their own folder/file

Declare redux to react mappers within the containers

```
function someStateToPropsMapper(state) {
    return { someComponentProperty:
        state.someStateAttribute }}

function someDispatchToPropsMapper(dispatch) {
    return { someEventHandler: () =>
        dispatch(someActionCreator()) }}
```

Move all containers into their own folder/file

Connect mappers component and export

```
const SomeContainer = connect
    (someStateToPropsMapper,
     someDispatchToPropsMapper)
       (SomeComponent)

export default SomeContainer
```

Bring it all together in index.js

```
import React from 'react'
import SomeContainer
    from './containers/SomeContainer'
import someReducer from './reducers'
import { createStore } from 'redux'
import { Provider } from 'react-redux'
```

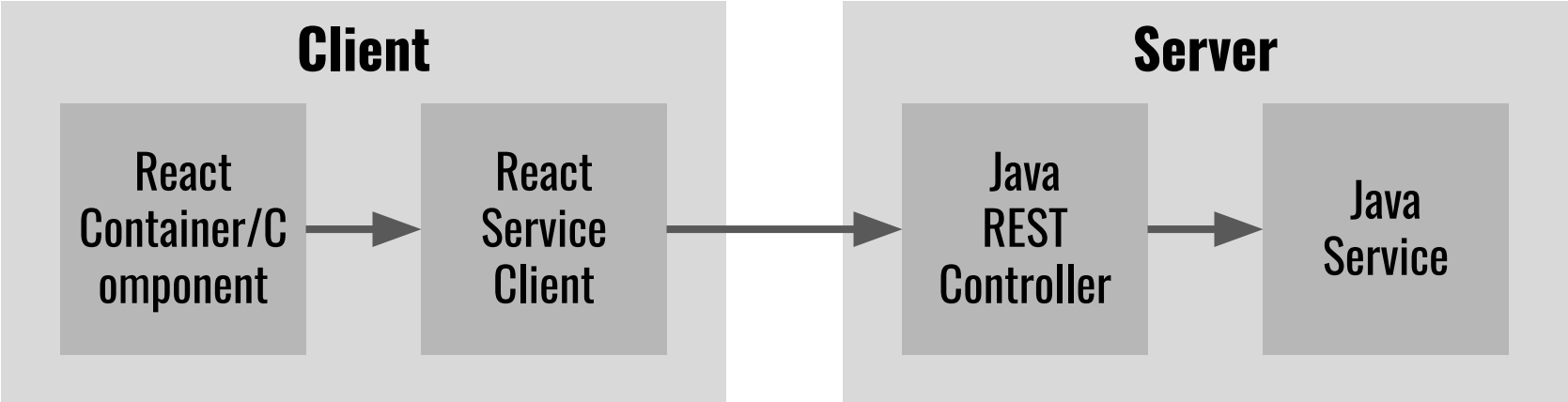
Bring it all together in index.js

```
const someStore = createStore(someReducer)

const SomeApp = () => (
 <Provider store={someStore}>
   <SomeContainer />
 </Provider>
)
```

Bring it all together in index.js

```
ReactDOM.render(
    <SomeApp/>
    document.getElementById('root')
);
```


Client
React Container/Component
React Service Client
Server
Java REST Controller
Java Service