DANIEL GRAHAM PHD

REDUX PART II

REDUX OVERVIEW

- Action
- Reducers
- Store

DESIGN EXERCISE

WITH BUG: UPDATES STATE BUT DOES NOT AUTOMATICALLY RERENDER

- Let's build an application that increments the value in the store.

https://snack.expo.io/@professorxii/simple-redux-example

The props and state aren't changing

FIXING THE STATE

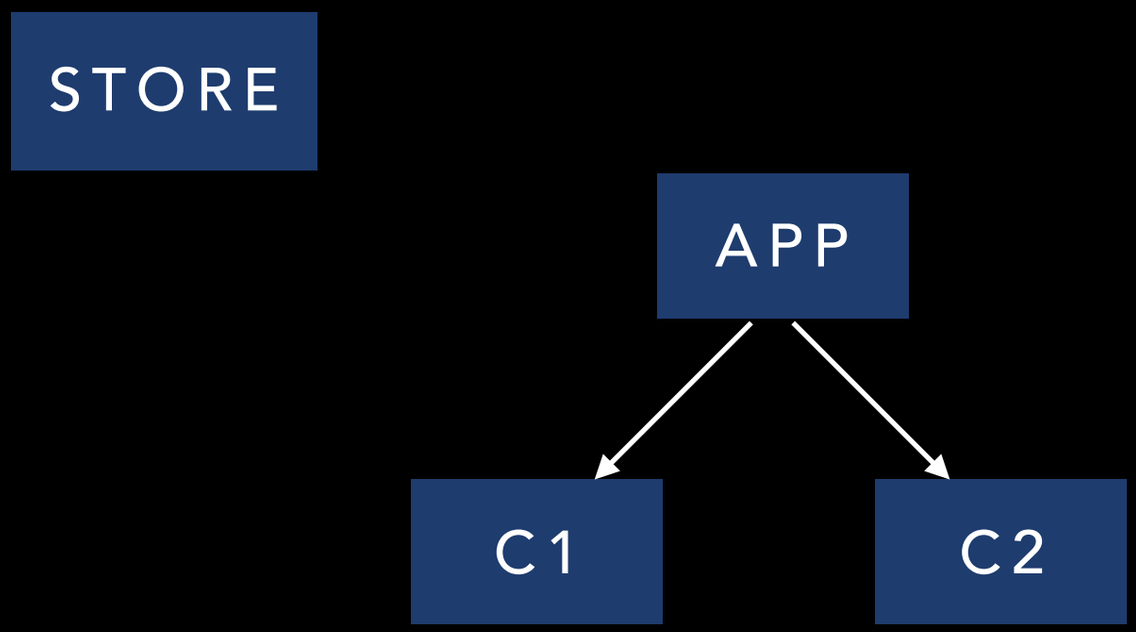
- Update the state when every we dispatch to store.
- Not the best solution Ideally we would want to have listeners that attach to the state.

HTTPS://SNACK.EXPO.IO/@PROFESSORXII/SIMPLE-REDUX-EXAMPLE-STATEFIX

FIXING THE UPDATE BUG

Provider Architecture

The <Provider /> makes the Redux store available to any nested components that have been wrapped in the connect() function.



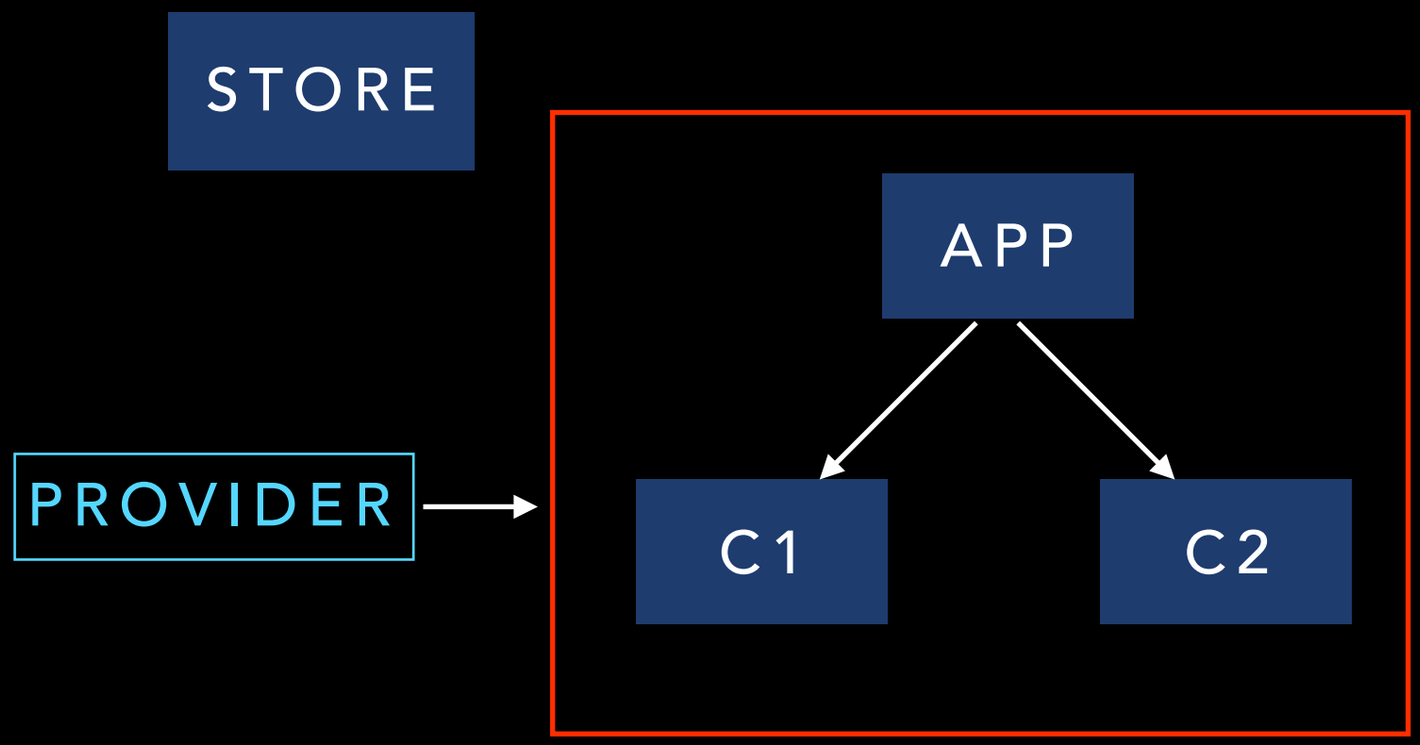

```
<Provider store={store}>
  <App />
</Provider>,
```

HTTPS://SNACK.EXPO.IO/@PROFESSORXII/SIMPLE-REDUX-EXAMPLE-FIXED

FIXING THE UPDATE BUG

Provider Architecture

The <Provider /> makes the Redux store available to any nested components that have been wrapped in the connect() function.



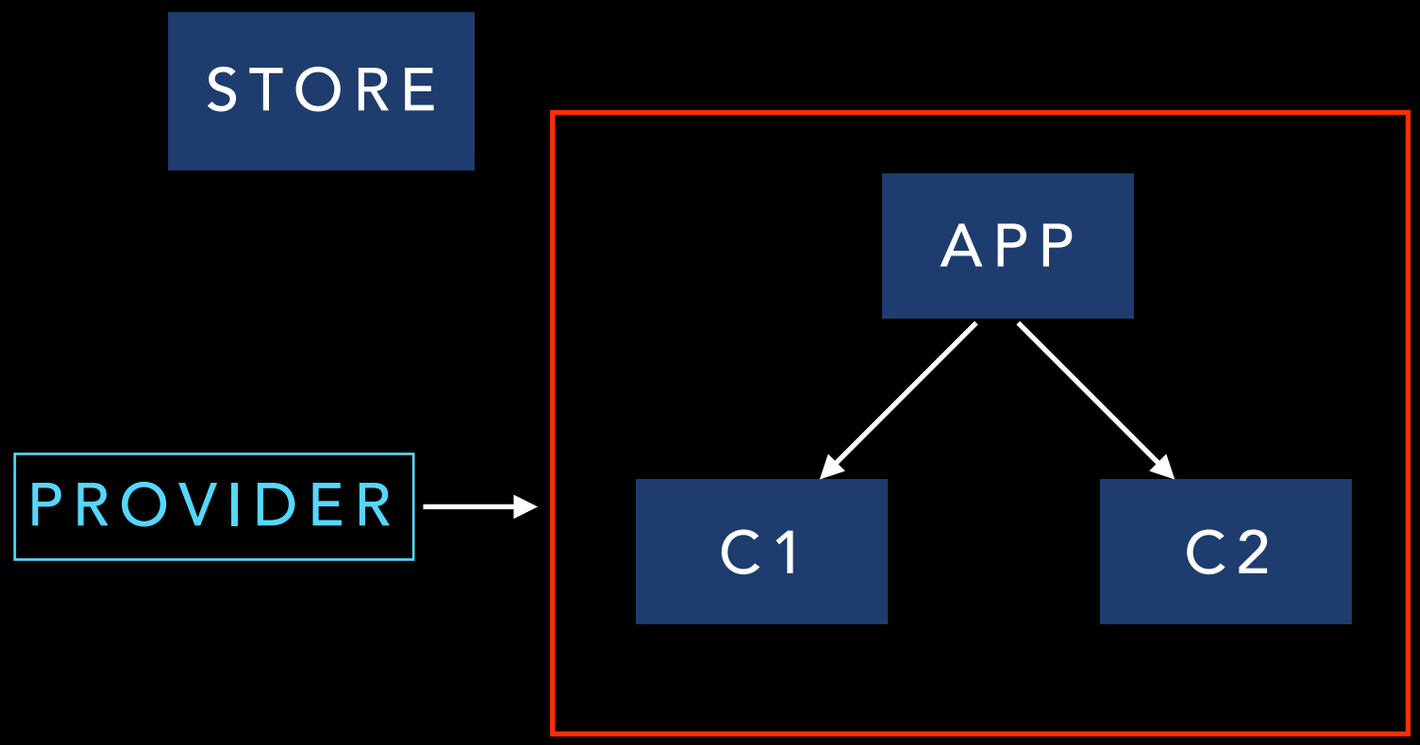

```
<Provider store={store}>
  <App />
</Provider>,
```

HTTPS://SNACK.EXPO.IO/@PROFESSORXII/SIMPLE-REDUX-EXAMPLE-FIXED

FIXING THE UPDATE BUG

Provider Architecture

The <Provider /> makes the Redux store available to any nested components that have been wrapped in the connect() function.




```
<Provider store={store}>
  <App />
</Provider>,
```

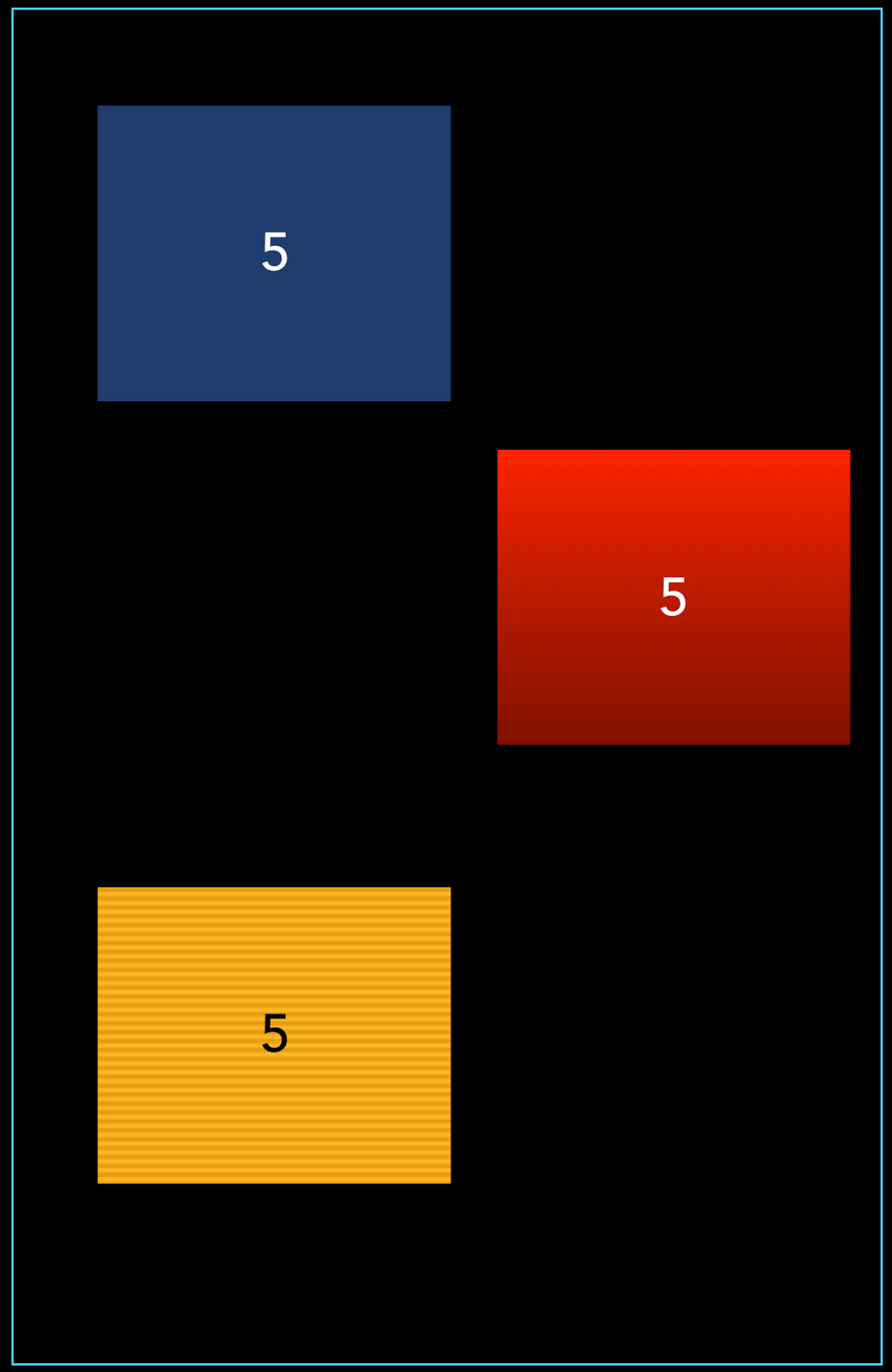
HTTPS://SNACK.EXPO.IO/@PROFESSORXII/SIMPLE-REDUX-EXAMPLE-FIXED

LET'S TAKE THIS FURTHER

- Let's build the example from the though experiment
  - Let's have the store be available to all the subcomponent
  - Let's have all the sub components subscribe to updates from the store.

THOUGHT EXPERIMENT

- Imaging an APP three views
- Click updates the number in all of the views



```
import Reducer from './reducers/reducers'
import { Provider, connect } from 'react-redux';
import { createStore } from 'redux'

export default class App extends React.Component {
  constructor(props){
    super(props)
    this.store = createStore(Reducer)
  }

  render() {
    return (
      <Provider  store={this.store}>
            <View style={styles.container}>
              <Box1/>
              <Box2/>
              <Box3/>
            </View>
        </Provider>
    );
  }
}
```

MOVE ALL THE REDUCERS SEPARATE FILE

MAKES THE STORE AVAILABLE TO ALL SUB COMPONENTS

The <Provider /> makes the Redux store available to any nested components that have been wrapped in the connect() function

WHY IS THIS A BETTER ARCHITECTURE

Let’s consider the Todo app. Let’s rewrite it with in Flux architecture

```
const store = createStore(rootReducer)

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

ACTIONS

Best practice to store actions in separate file and export

```
let nextTodoId = 0
export const addTodo = text => ({
  type: 'ADD_TODO',
  id: nextTodoId++,
  text
})


export const toggleTodo = id => ({
  type: 'TOGGLE_TODO',
  id
})
```

REDUCERS

```
const todos = (state = [], action) => {
    switch (action.type) {
      case 'ADD_TODO':
        return [
          ...state,
          {
            id: action.id,
            text: action.text,
            completed: false
          }
        ]
      case 'TOGGLE_TODO':
        return state.map(todo =>
          todo.id === action.id ? { ...todo, completed: !todo.completed } : todo
        )
      default:
        return state
    }
  }

  export default todos
```

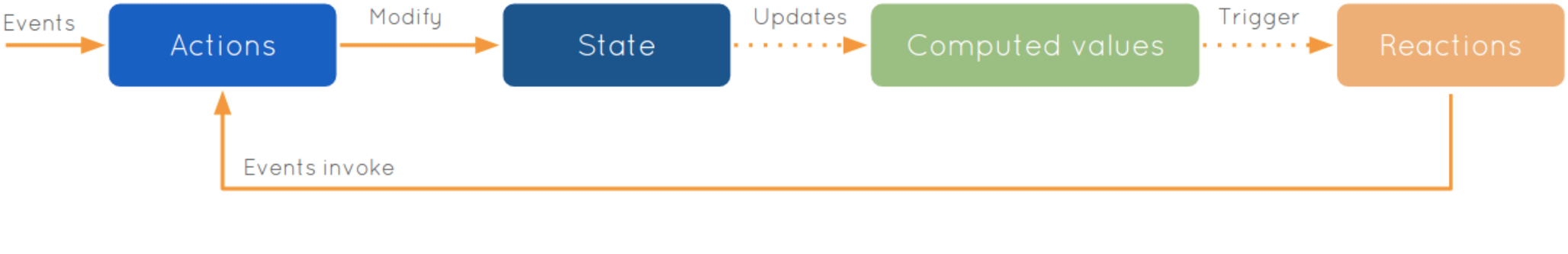
PRESENTATION LAYER

```
    <View>
      {todos.map(todo => (
        <Todo key={todo.id} {...todo} onClick={() => toggleTodo(todo.id)} />
      ))}
    </View>
```

Todos added by calling

```
dispatch(addTodo(input.value))
```

MOBX



Events invoke actions. Actions are the only thing that modify state and may have other side effects.

```
@action onClick = () => {
  this.props.todo.done = true;
}
```

State is observable and minimally defined. Should not contain redundant or derivable data. Can be a graph, contain classes, arrays, refs, etc.

```
@observable todos = [{
  title: "learn MobX",
  done: false
}]
```

Computed values are values that can be derived from the state using a pure function. Will be updated automatically by MobX and optimized away if not in use.

```
@computed get completedTodos() {
  return this.todos.filter(
    todo => todo.done
  )
}
```

Reactions are like computed values and react to state changes. But they produce a side effect instead of a value, like updating the UI.

```
const Todos = observer({ todos } =>
  <ul>
    todos.map(todo => <TodoView ... />
  </ul>
)
```

```
import { observable } from "mobx"

class Todo {
    id = Math.random()
    @observable title = ""
    @observable finished = false
}
```

```
import { decorate, observable } from "mobx"

class Todo {
    id = Math.random()
    title = ""
    finished = false
}
decorate(Todo, {
    title: observable,
    finished: observable
})
```

```
class TodoList {
    @observable todos = []
    @computed get unfinishedTodoCount() {
        return this.todos.filter(todo => !todo.finished).length
    }
}
```

```
import React, { Component } from "react"
import ReactDOM from "react-dom"
import { observer } from "mobx-react"

@observer
class TodoListView extends Component {
    render() {
        return (
            <div>
                <ul>
                    {this.props.todoList.todos.map(todo => (
                        <TodoView todo={todo} key={todo.id} />
                    ))}
                </ul>
                Tasks left: {this.props.todoList.unfinishedTodoCount}
            </div>
        )
    }
```

The component will now behave as if it has state

HTTPS://JSFIDDLE.NET/MWESTSTRATE/WV3YOPO0/

```
const TodoView = observer(({ todo }) => (
    <li>
        <input
            type="checkbox"
            checked={todo.finished}
            onClick={() => (todo.finished = !todo.finished)}
        />
        {todo.title}
    </li>
))

const store = new TodoList()
ReactDOM.render(<TodoListView todoList={store} />, document.getElementById("mount"))
```

REFERENCES

- https://redux.js.org/basics/example
- https://medium.com/@pavsidhu/using-redux-with-react-native-9d07381507fe
- https://www.valentinog.com/blog/redux/