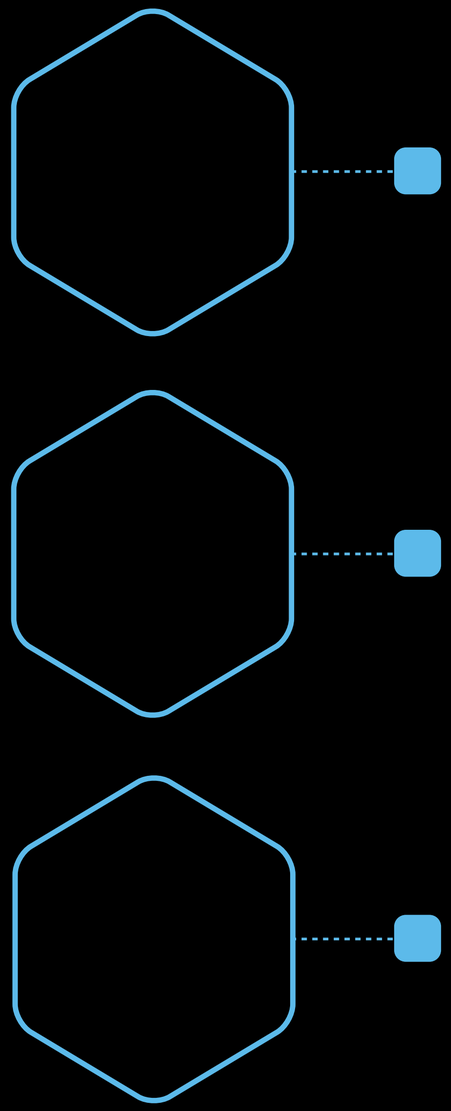
REDUX: FP FOR THE WEB

Pratik Patel @prpatel



ARCHITECTURAL LIB

ONE WAY DATA FLOW

USED WITH REACT (CAN ALSO BE USED WITH ANGULAR)

@prpatel

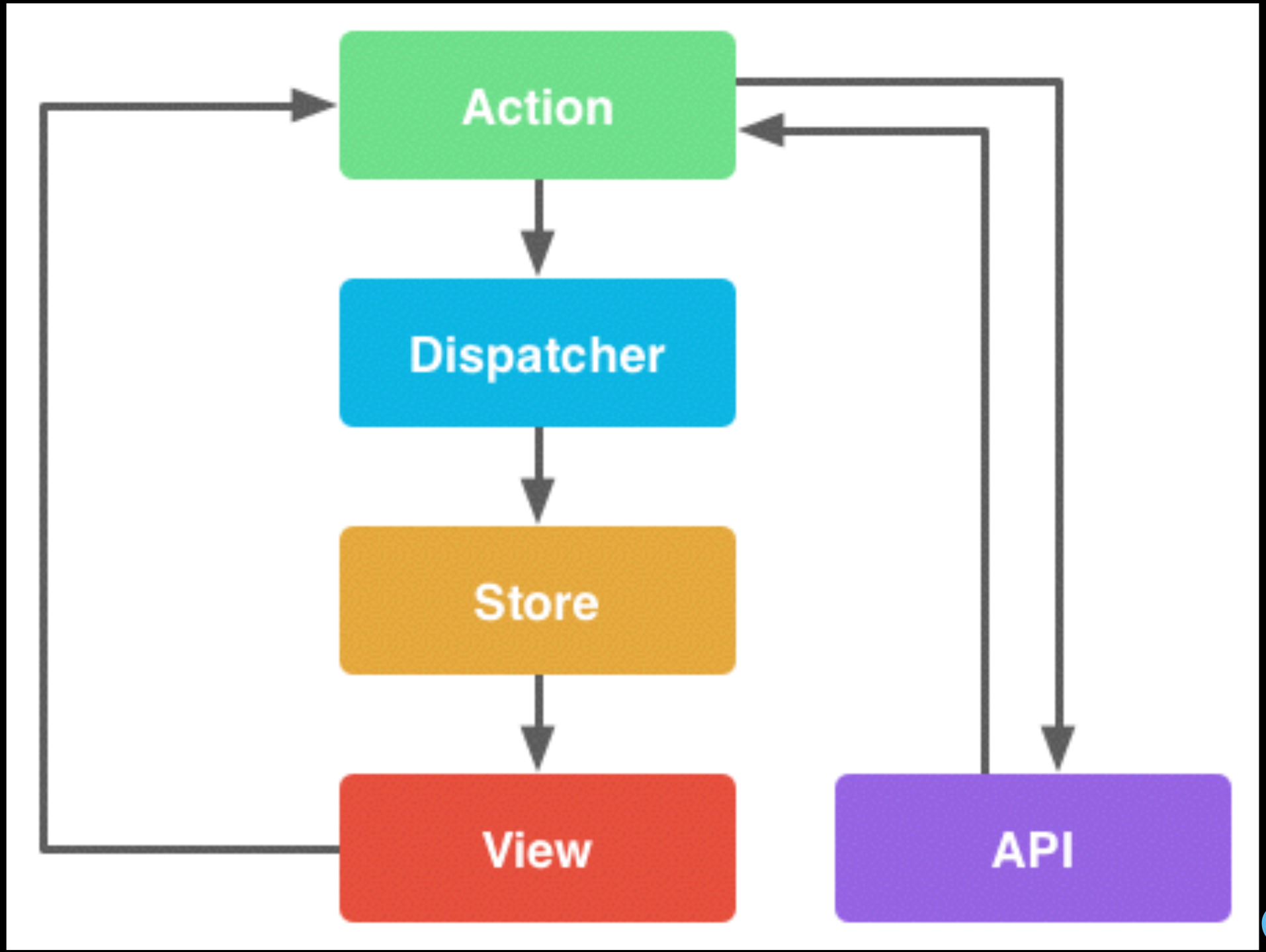
WE WILL DISCUSS REDUX + REACT

@prpatel

ONE WAY DATA FLOW

ARCHITECTURAL PATTERN

FLUX



NO “DATA BINDING”

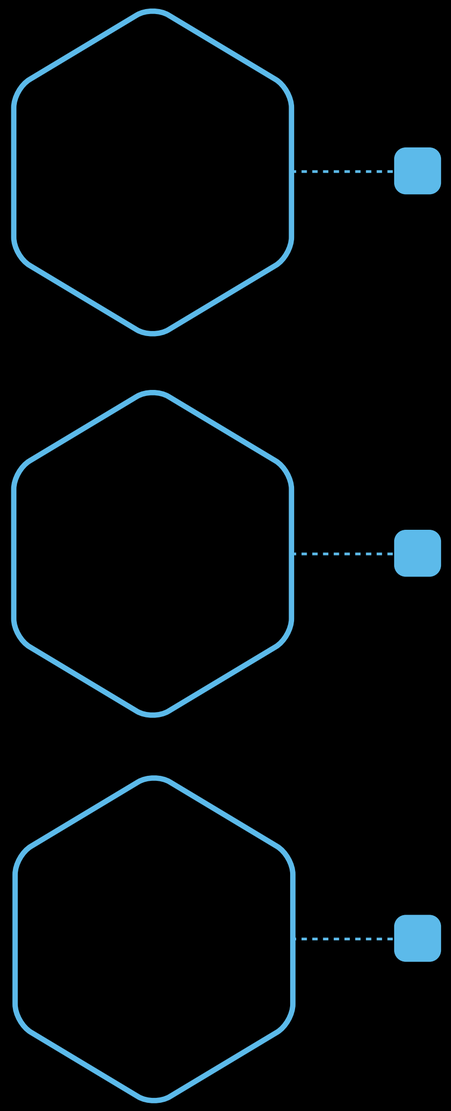
@prpatel

REFLUX, FLUMMOX, ALT, ETC

FLUX IMPLEMENTATIONS

REDUX IS A FLUX IMPLEMENTATION

@prpatel

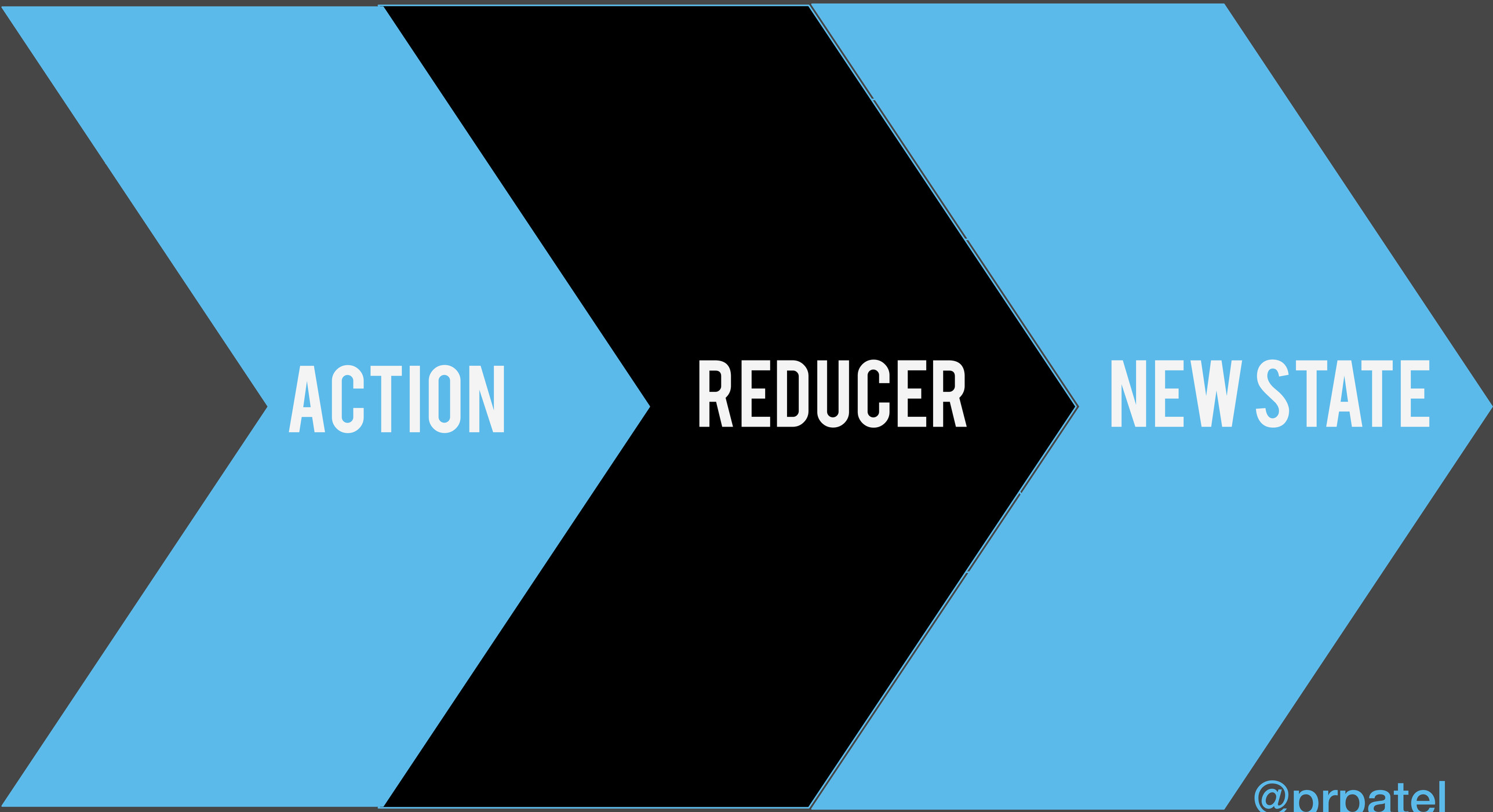


STORES

ACTIONS

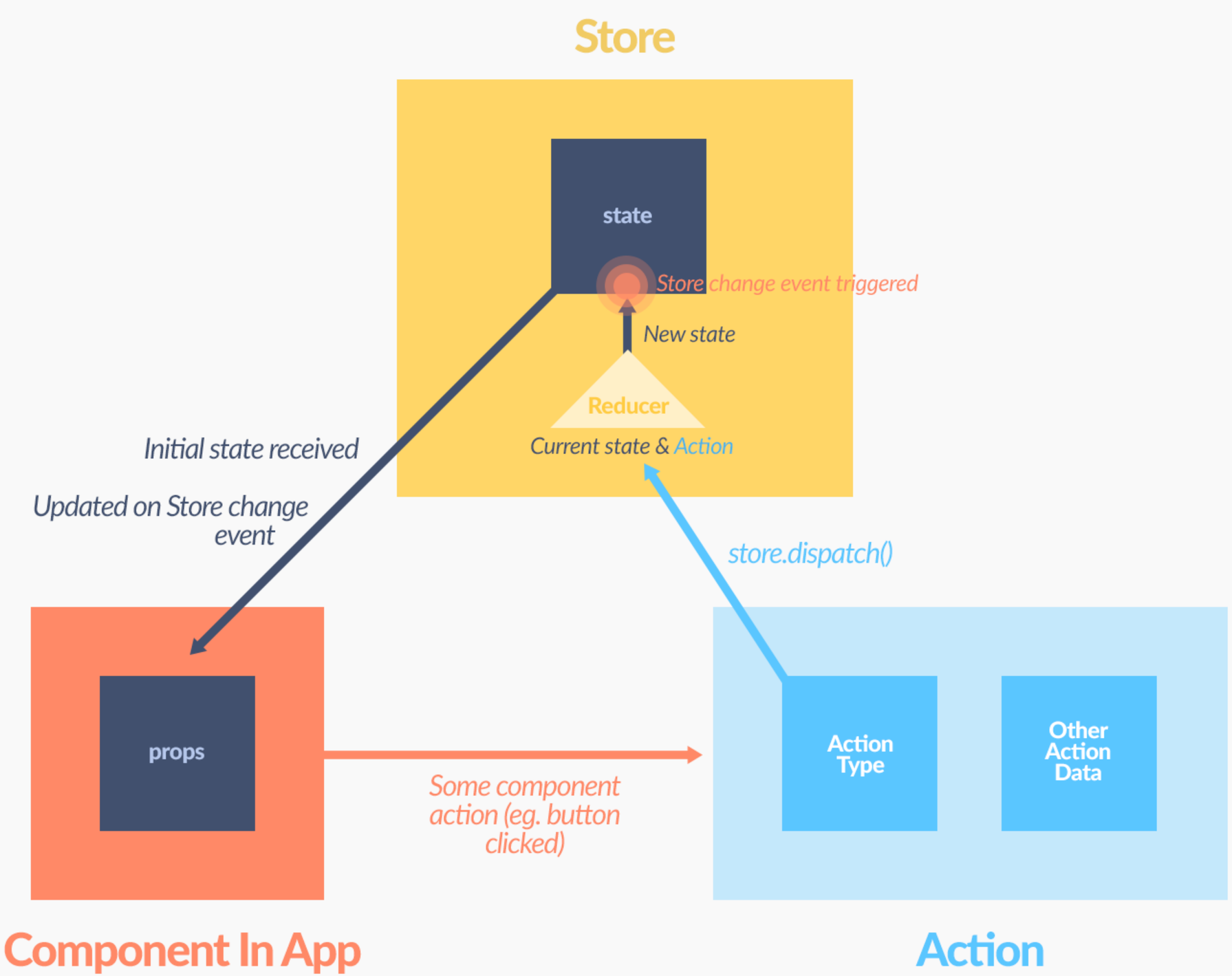
REDUCERS

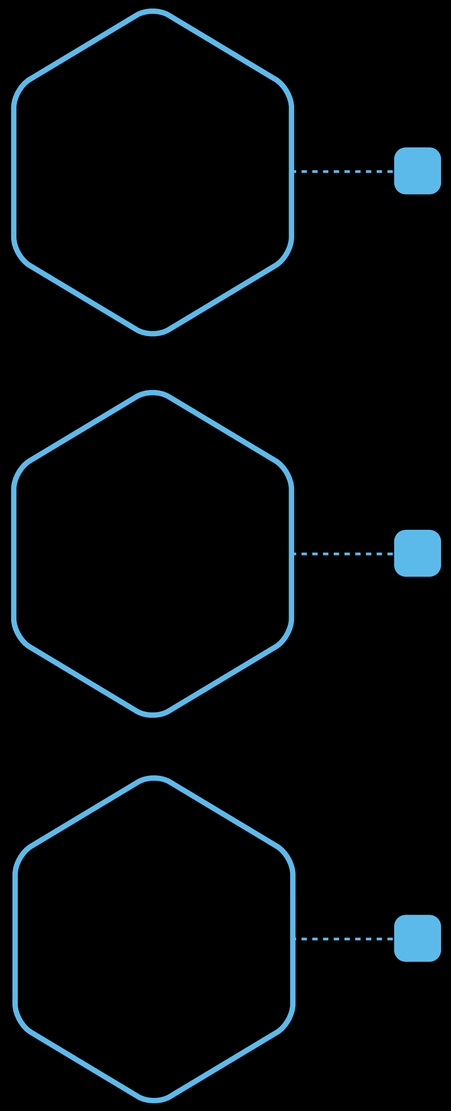
@prpatel


ACTION
REDUCER
NEW STATE

REDUX FLOW DIAGRAM

@prpatel


Store
state
Store change event triggered
New state
Reducer
Current state & Action
Initial state received
Updated on Store change event
store.dispatch()
props
Some component action (eg. button clicked)
Action Type
Other Action Data
Component In App
Action



- STATE TREE
- READ ONLY
- BASED ON PURE FUNCTIONS

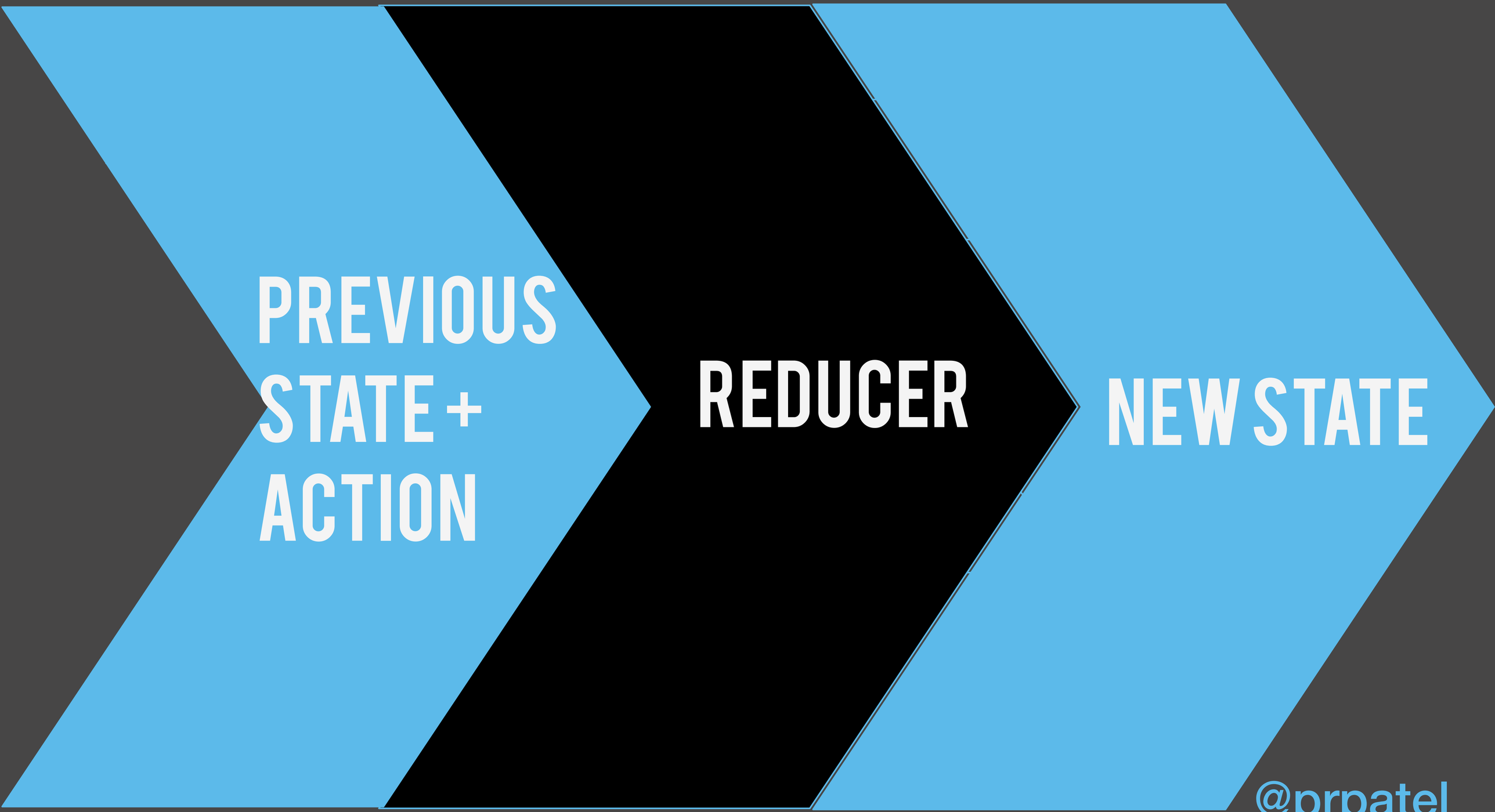
@prpatel

BEGINS BY THINKING ABOUT "APPLICATION STATE" DATA STRUCTURE

DESIGNING A REDUX APP

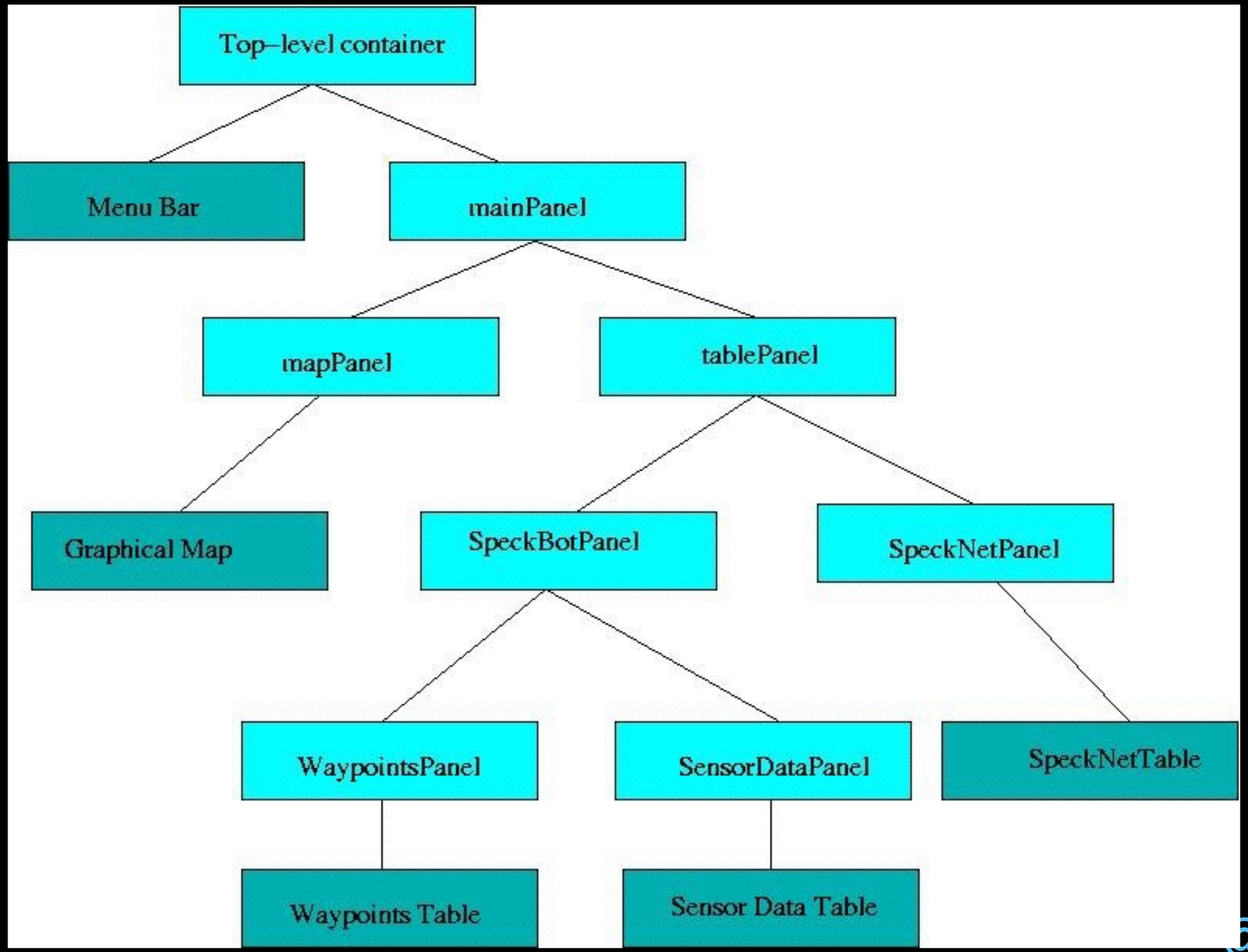
APPLICATION STATE IS ALL STORED IN ONE SINGLE TREE STRUCTURE

@prpatel


PREVIOUS STATE + ACTION
REDUCER
NEW STATE

REDUCER TRANSFORMS PREVIOUS STATE TO NEW STATE

@prpatel


Top–level container
Menu Bar
mainPanel
mapPanel
tablePanel
Graphical Map
SpeckBotPanel
SpeckNetPanel
WaypointsPanel
SensorDataPanel
SpeckNetTable
Waypoints Table
Sensor Data Table

STORES

STORES THE STATE OF YOUR APPLICATION OVER TIME

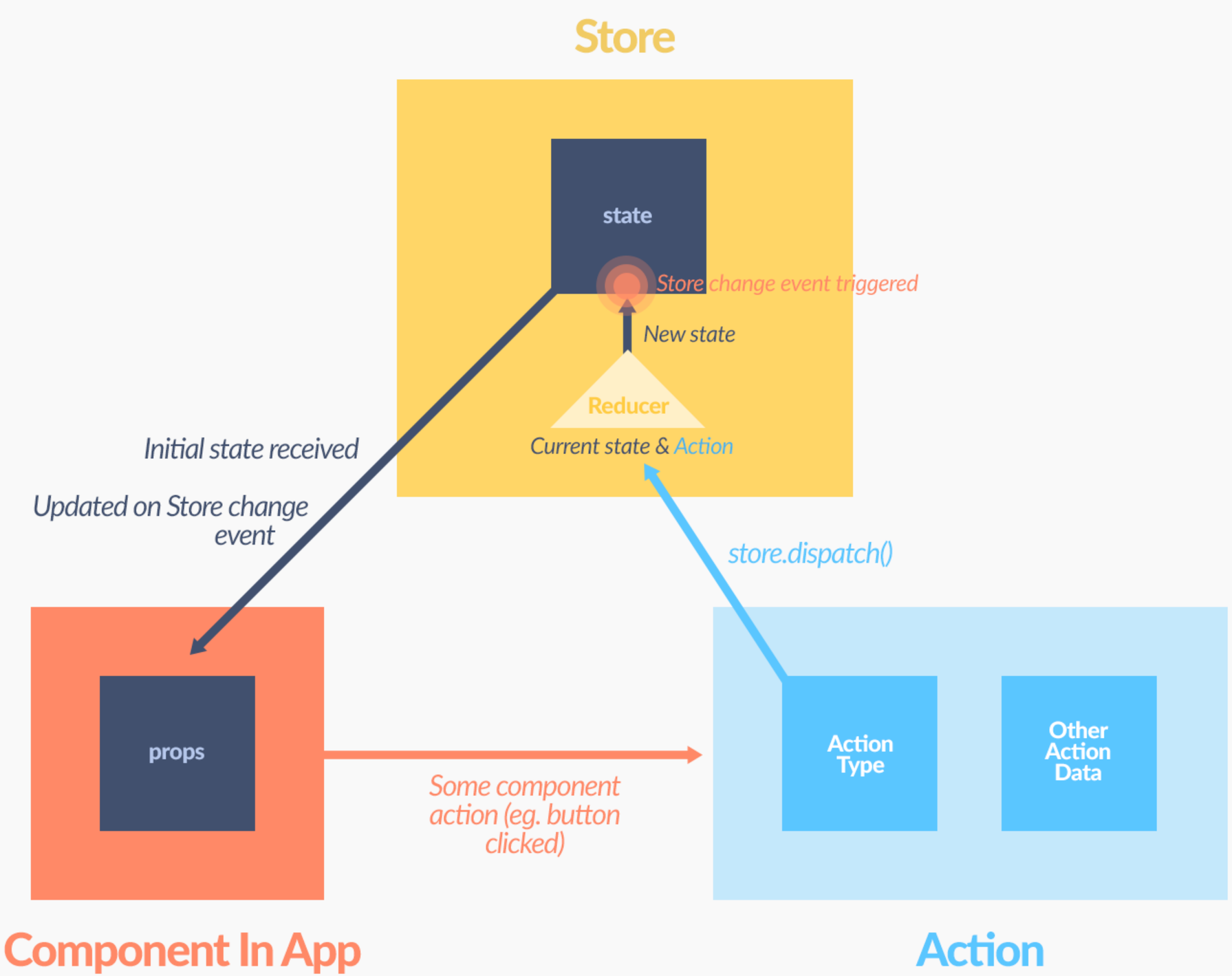
REDUX STORES

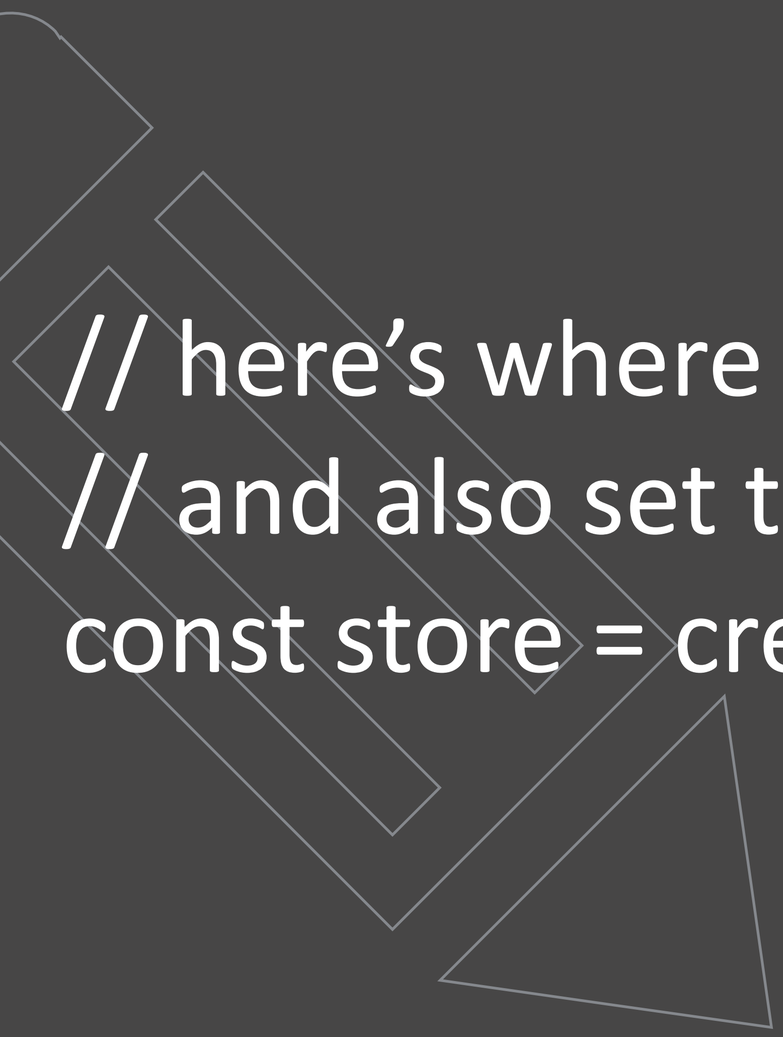
RECEIVE ACTIONS THAT EVOLVE THE STATE FROM ONE VERSION TO THE NEXT

REDUX STORES

CENTRAL POINT OF THE APP

REDUX STORES


Store
state
Store change event triggered
New state
Reducer
Current state & Action
Initial state received
Updated on Store change event
store.dispatch()
props
Some component action (eg. button clicked)
Action Type
Other Action Data
Component In App
Action



```
// here's where we tie the reducers to the store
// and also set the initial state
const store = createStore(rootReducer, initialState)
```

ACTIONS

LAYER OF INDIRECTION

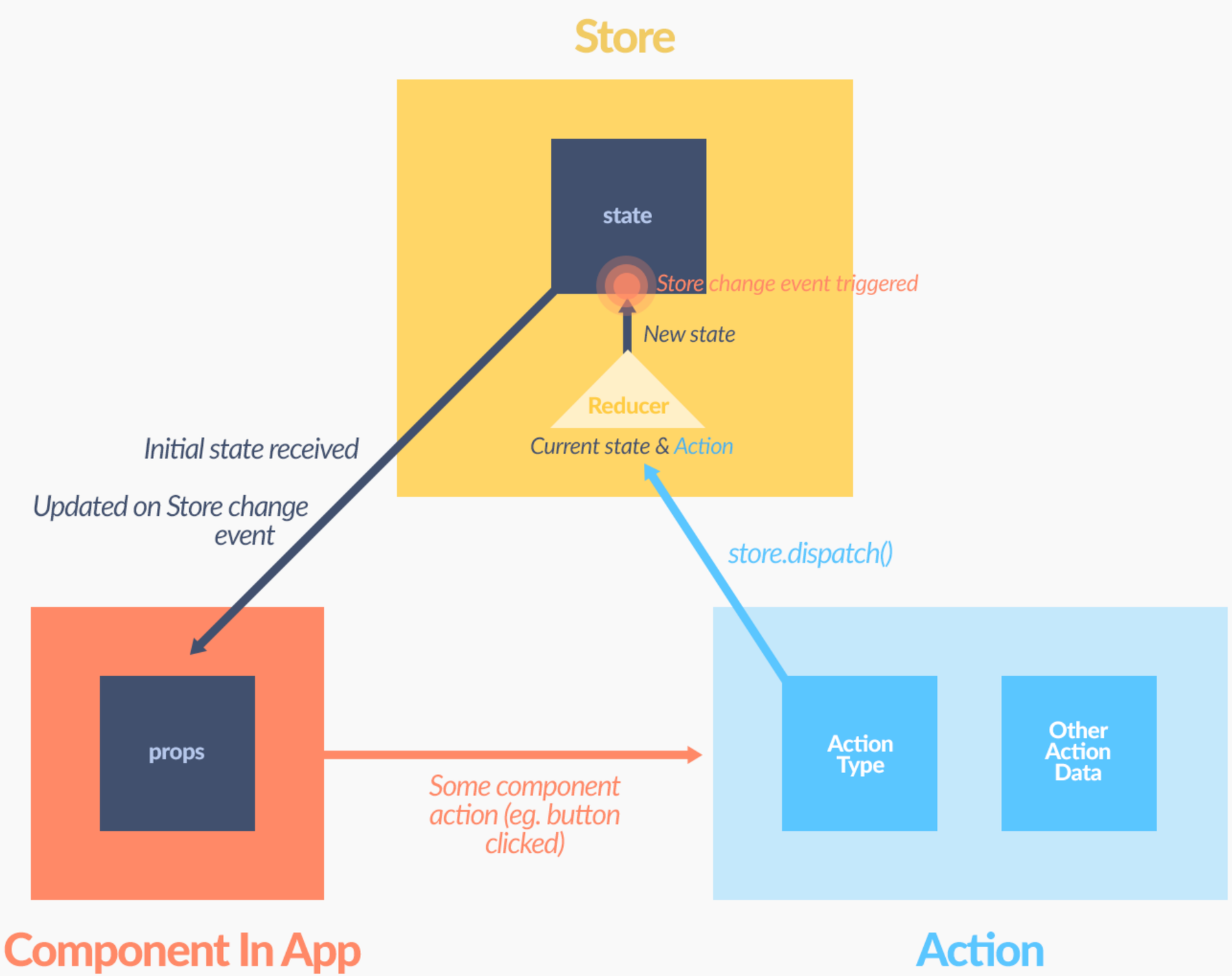
REDUX ACTIONS

SIMPLE DATA STRUCTURE
THAT DESCRIBES A CHANGE
THAT SHOULD OCCUR IN YOUR
APP STATE

REDUX ACTIONS

DESCRIPTION OF A FUNCTION CALL PACKAGED INTO A LITTLE OBJECT

REDUX ACTIONS


Store
state
Store change event triggered
New state
Reducer
Current state & Action
Initial state received
Updated on Store change event
store.dispatch()
props
Some component action (eg. button clicked)
Action Type
Other Action Data
Component In App
Action

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

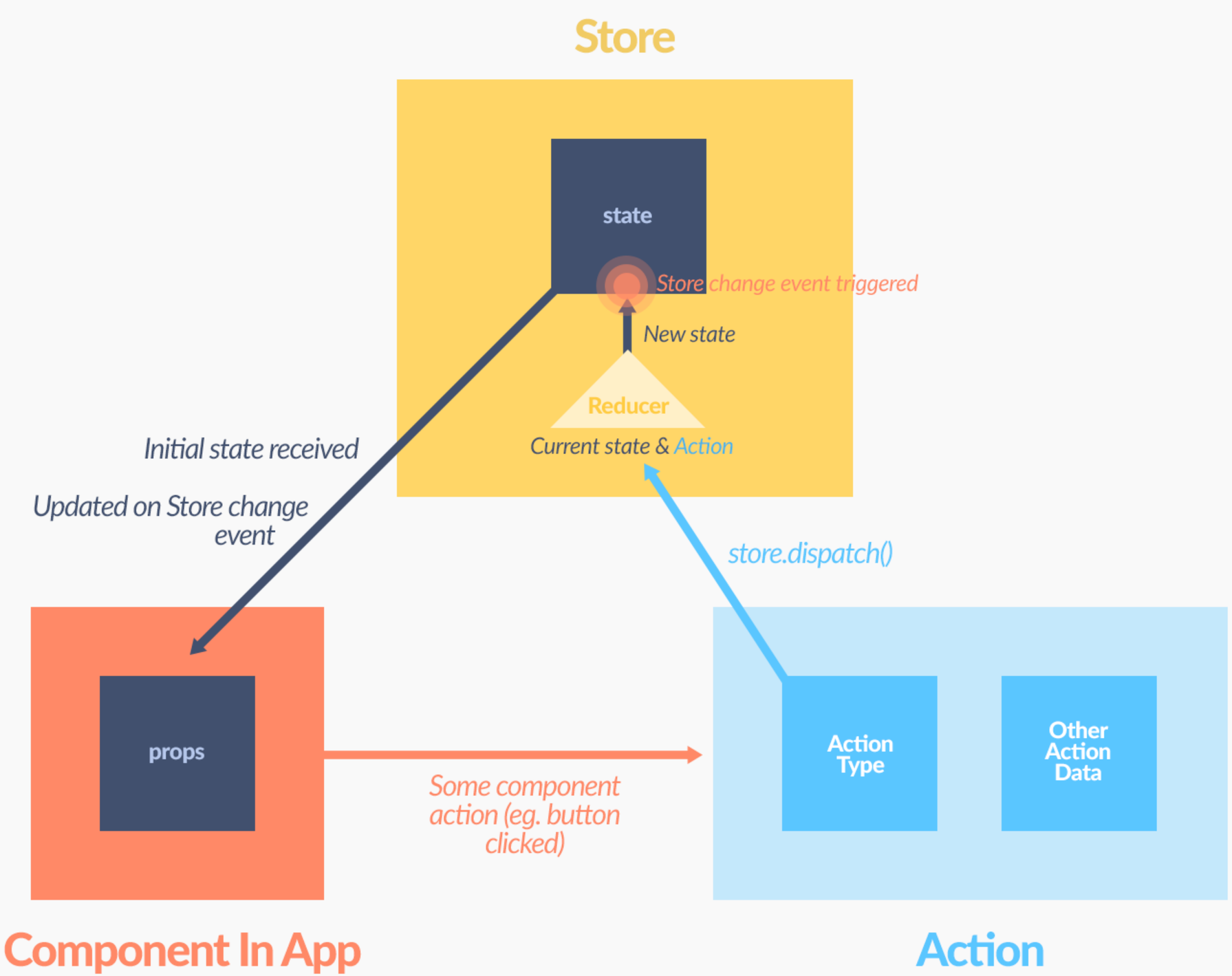
REDUCERS

REDUCER SHOULD DELEGATE TO ONE OF THE CORE FUNCTIONS BASED ON THE TYPE OF THE ACTION

REDUX REDUCER

MAIN REDUCER FUNCTION ONLY HANDS PARTS OF THE STATE TO LOWER-LEVEL REDUCER FUNCTIONS

REDUX ACTIONS


Store
state
Store change event triggered
New state
Reducer
Current state & Action
Initial state received
Updated on Store change event
store.dispatch()
props
Some component action (eg. button clicked)
Action Type
Other Action Data
Component In App
Action

```
case ADD_TODO:
    return [
      {
        id: state.reduce((maxId, todo) =>
Math.max(todo.id, maxId), -1) + 1,
        completed: false,
        text: action.text
      },
      ...state
    ]
```

WIRING

MUST CONNECT THE REACT VIEW, STORE, AND REDUCER

REDUX CONNECT

```
function mapStateToProps(state) {
  return {
    todos: state.todos
  }
}
function mapDispatchToProps(dispatch) {
  return {
    actions: bindActionCreators(TodoActions, dispatch)
  }
}
```

```
export default connect(
  mapStateToProps,
  mapDispatchToProps
)(App)
```

THE STATE TREE
MAPS TO REACT
COMPONENT PROPS

STATE -> PROPS

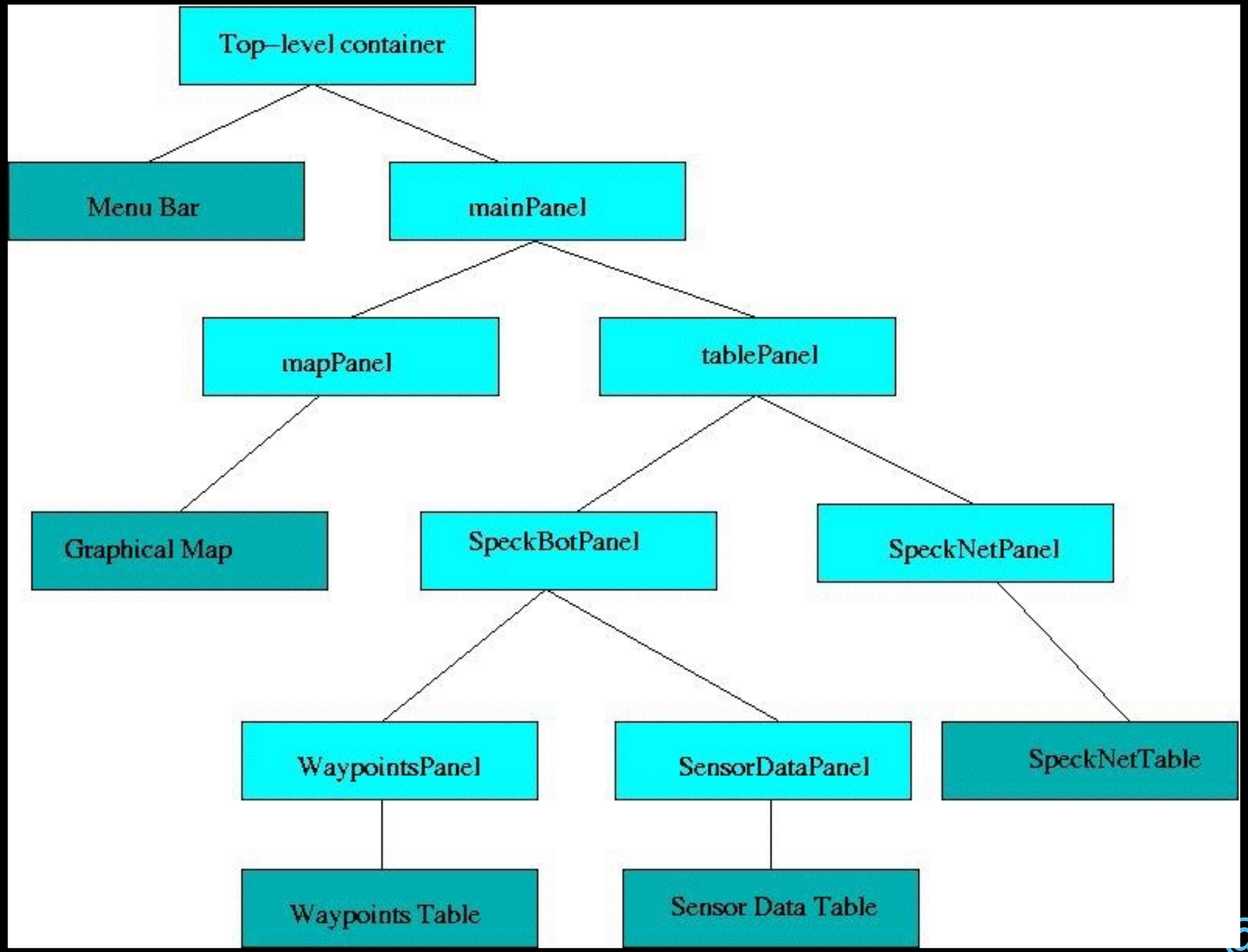
@prpatel

STATE TREE MIRRORS COMPONENT TREE

MIRRORED TREES

ACTIONS AND DATA PASSED AS PROPS

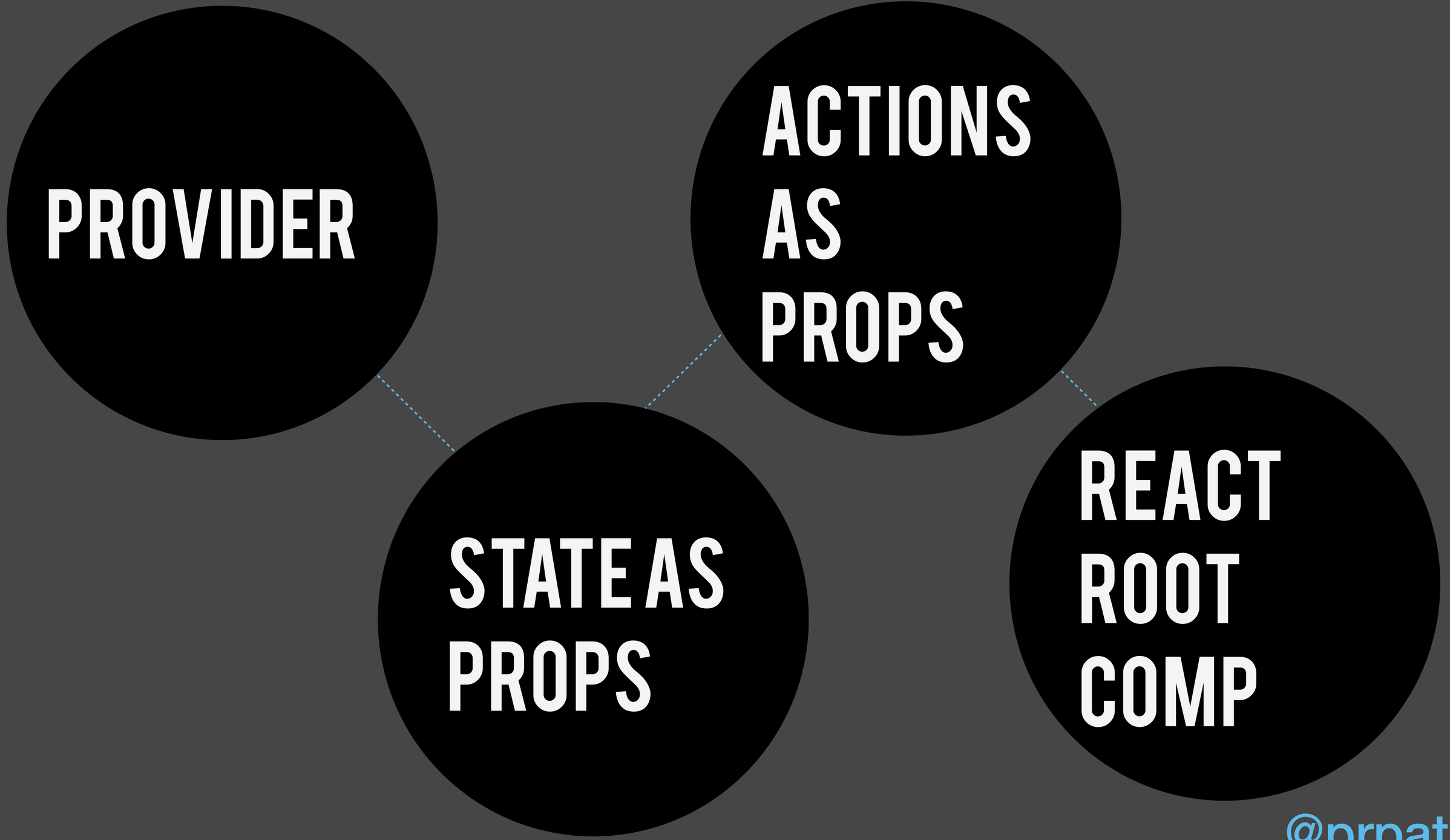
PARENT-CHILD


Top–level container
Menu Bar
mainPanel
mapPanel
tablePanel
Graphical Map
SpeckBotPanel
SpeckNetPanel
WaypointsPanel
SensorDataPanel
SpeckNetTable
Waypoints Table
Sensor Data Table

```
class App extends Component {
  render() {
    const { todos, actions } = this.props
    return (
      <div>
        <Header addTodo={actions.addTodo} />
        <MainSection todos={todos} actions={actions} />
      </div>
    )
  }
}
```

```
const store = configureStore()

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


PROVIDER
ACTIONS AS PROPS
STATE AS PROPS
REACT ROOT COMP
@prpatel

WALKTHROUGH

```
// VIEW REQUESTS ACTION
<button className="destroy"
        onClick={() =>
deleteTodo(todo.id)} />
```

ACTION IS DISPATCHED AUTOMATICALLY (DUE TO BINDACTIONCREATORS)

DISPATCH

STORE RECEIVES ACTION, SENDS CURRENT STATE AND ACTION TO ROOT REDUCER

STORE

```
// ACTION definition
export function deleteTodo(id) {
  return { type: types.DELETE_TODO, id }
}
```

```
// REDUCER does some work to manipulate state tree
case DELETE_TODO:
    return state.filter(todo =>
      todo.id !== action.id
    )
```

ROOT REDUCER CONSTRUCTS FINAL NEW STATE TREE, RETURNS IT TO THE STORE

REDUCER -> STORE

REPLACES THE OLD STATE TREE WITH THE NEW ONE FROM REDUCER

STORE

STORE TELLS VIEW LAYER
BINDING THERE IS NEW STATE

STORE -> VIEW

VIEW LAYER BINDING GETS THE NEW STATE FROM STORE

VIEW <- STORE

VIEW LAYER BINDING TRIGGERS A RE-RENDER WITH NEW STATE

VIEW -> RENDER

WHERE'S THE FUNCTIONAL PROGRAMMING PART?

@prpatel

STATE TREE IS NEVER MANIPULATED – NEW TREE IS ALWAYS CREATED FROM OLD

IMMUTABLE DATASTRUCTS

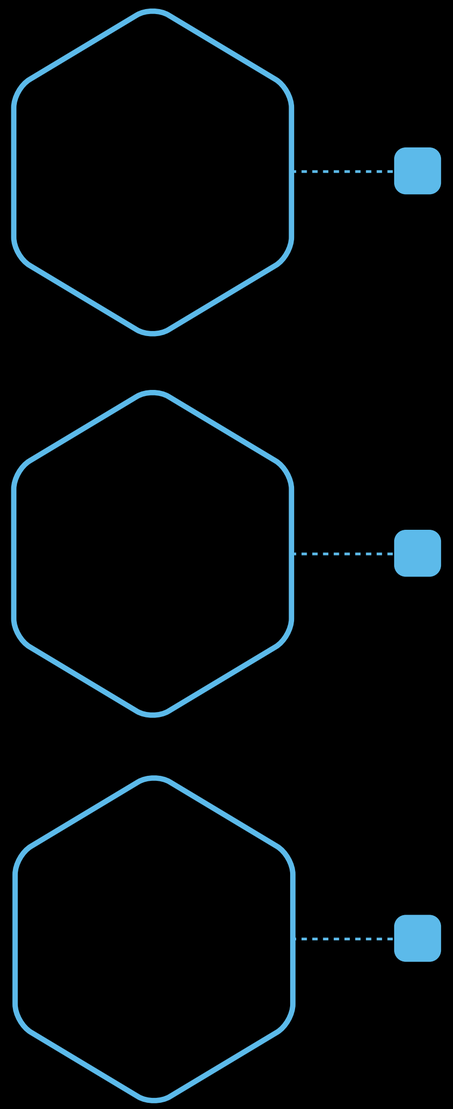
NO SIDE-EFFECTS

PURE FUNCTIONS

USE OPERATIONS LIKE MAP, REDUCE, FILTER

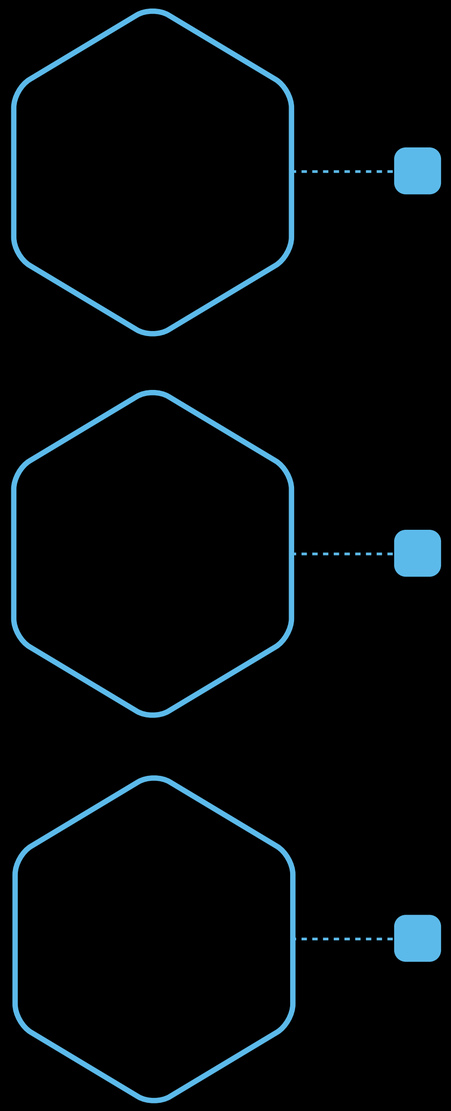
FP STYLE

WHY IS THIS BETTER?



- EASIER TO UNDERSTAND
- COMPLEXITY EASIER TO MANAGE
- EASIER TO TEST

WHAT DIDN’T WE COVER?

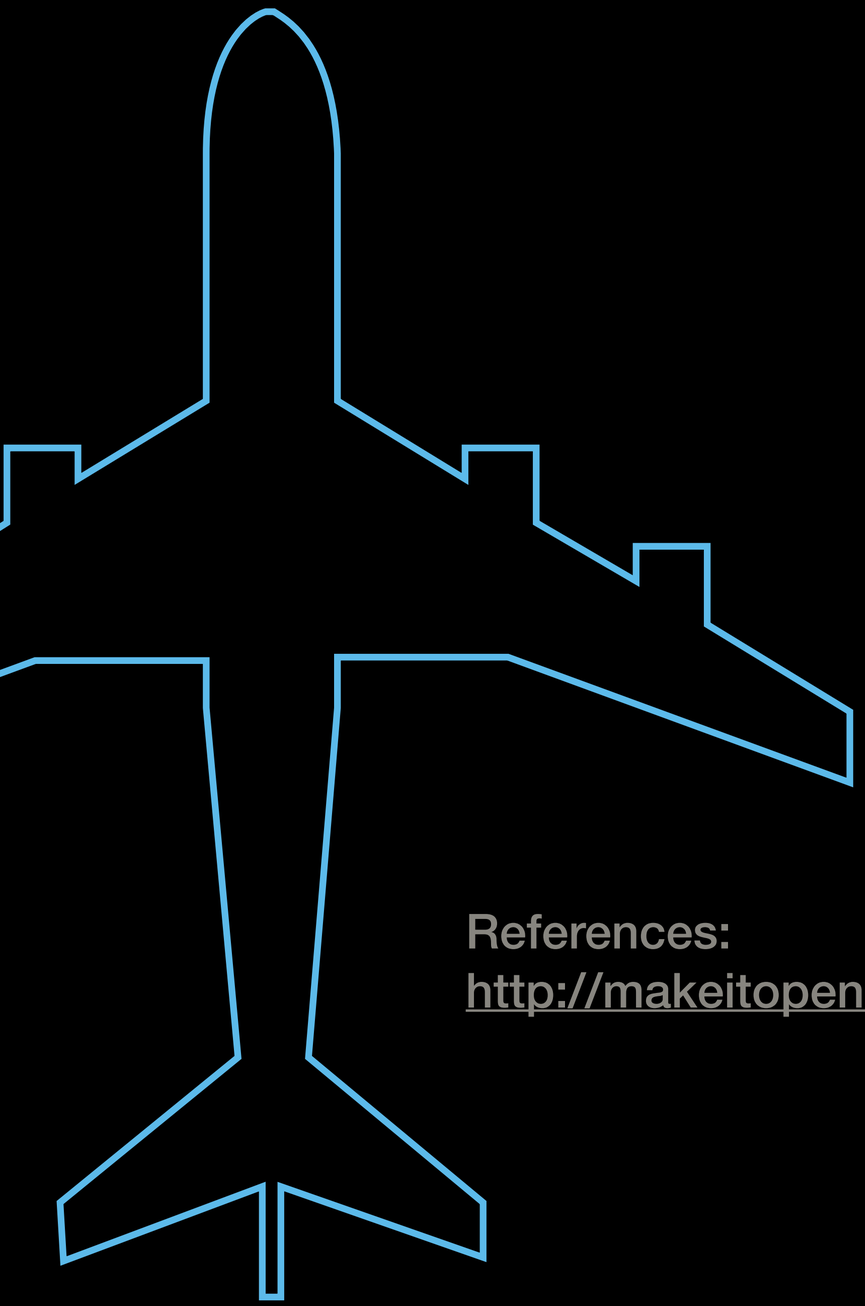


TESTING

SMART VS DUMB COMPONENTS

SUB REDUCERS

@prpatel



THANK YOU

References:
http://makeitopen.com/tutorials/building-the-f8-app/data/

@prpatel