

MANAGE STATE

OF

A LARGE REACT APP

www.thatJSDude.com

WE WILL COVER

1. What is State
2. 5 scenarios to Manage state
3. Redux
4. Redux-saga

WHAT IS A STATE?

THINK FOR A SECOND...

- It's summer time
- You are sitting in the beach
- Under a large beach umbrella
- And your friend is grilling some steak

Your state is: **pretty RELAXED**

THINK FOR ANOTHER SECOND...

- You are in a big Conference
- Talking in front of 30+ people
- Trying to explain what is state
- Everyone is staring at you

Your state is: **pretty NERVOUS**

This means state of your mind changes based on the situation

SO WHAT IS A STATE?

State is something

- That might change
- Like: your mood, your confidence level
- Or London Weather: winter, rain, 2 weeks of summer!

IN A REACT APP

You can think of two level of states

1. Application State: Global State

- Like: Session storage/ Database/others
- Simply think it like a database
- Any component, anywhere in the app can access

2. Component State

- Within that specific component
- Can only be updated within that component
- Can be passed down to its children via props

React States

THINK ABOUT AN APPLICATION

EMA-JOHN

Your manager tells you to

ADD MORE FEATURES...

MORE FEATURES MEAN...

- More components
- More data, more props has to pass data
- More event handler has to be managed
- State grows and becomes complicated

- And you create more mess...

But Don't worry: the bigger is the mess, this higher is the job security

REDUX

Will start knocking at your door

REDUX WILL ASK YOU

- Put app state out of your component
- Create state as plain objects and arrays
- Describe changes as plain objects
- Handling changes as pure functions

None of these is a must to build an app

ADDING REDUX

- Will add an indirection to your problem
- Will shift the complexity in a different direction
- Could make your situation **WORSE**

Is redux a must?

ASSES YOUR APPLICATION

- How many sibling Components have to communicate?
- Does child component contain state from parent component?
- How many different views need to share state?

ASSESS YOUR TEAM

- Do a proof of concept in a feature branch
- Explore how it solves the problem before diving into it
- Does team understand: Redux core concepts.
- Can handle extra level of complexity
- Immutability, pure function, etc.

USE CASE-1

You are building from scratch

START WITH A MOCKUP

Only show products in stock
Football \$49.99
Baseball \$9.99
Basketball \$29.99
iPod Touch \$99.99
iPhone 5 \$399.99
Nexus 7 \$199.99

```
[  
  {price: "$49.99", stocked: true, name: "Football"},  
  {price: "$9.99", stocked: true, name: "Baseball"},  
  {price: "$29.99", stocked: false, name: "Basketball"},  
  {price: "$99.99", stocked: true, name: "iPod Touch"},  
  {price: "$399.99", stocked: false, name: "iPhone 5"},  
  {price: "$199.99", stocked: true, name: "Nexus 7"},  
];
```

Think in React

YOU STARTED LOADING DATA FROM THE API

```
componentWillMount() {
  Promise.all([
    api.shop.getProducts(),
    api.shop.getCurrentOrder()
  ]).then(([products, order]) => {
    // Extract ids of the products in my order now
    const orderProductIds = order.map(p => p.id);
    // Set "selected" flag on all products
    const uiProducts = products.map(p => {
      const selected = orderProductIds.indexOf(p.id) !== -1;
      return { ...p, selected };
    });
    this.setState({ products: uiProducts, order });
  });
}
```

Adding two lists in the state

```
render() {  
  return (<ul>  
    {this.state.products.map(product => (  
      <li key={product.id}> {product.name} {product.price}  
        <button  
          onClick={e => this.toggleProductInOrder(product)}>  
            {!product.selected ? 'Add' : 'Remove'}  
        </button>  
      </li>  
    )}  
  </ul>);  
}
```

HANDLE A PRODUCT TOGGLE

```
toggleProductInOrder(product) {
  // Take all products, flip selection of the toggled one
  const newProducts = this.state.products.map(p => {
    if (p.id !== product.id) { return p; }
    return { ...p, selected: !p.selected };
  });
  // Filter out the selected products
  const newOrderProducts = newProducts.filter(p => p.selected)
  // Update both the order and the products
  return api.shop.updateOrder(newOrderProducts)
    .then(newOrder => {
      this.setState({ order: newOrder, products: newProducts });
    });
}
```

CAN WE MAKE STATE SIMPLER?

- Try to Avoid changing data you get from the API
- Handle Derived Property in Render
- Try to avoid 2 or more state change with one event

```
componentWillMount() {
  Promise.all([
    api.shop.getProducts(),
    api.shop.getCurrentOrder()
  ]).then(([products, order]) => {
    // No pre-processing of our data from the server
    this.setState({ products, order });
  });
}

getSelected(product) {
  return this.state.order.find(p => p.id === product.id);
}
```

```
render() {  
  const { order, products } = this.state;  
  return (<ul>  
    {products.map(product => (  
      <li key={product.id}> {product.name} {product.price}  
        <button type="button"  
          onClick={() => this.toggleProductInOrder(product)}>  
            {!this.getSelected(product) ? 'Add' : 'Remove'}  
          </button>  
        </li>  
      )}  
    </ul>);  
}
```

If possible calculate derived properties of your data during render()

```
toggleProductInOrder(product) {
  const order = this.state.order;
  const selectedProduct = this.getSelected(product);
  //Updating an order is just adding or removing on item
  const updatedOrder = selectedProduct
    ? without(order, product) : concat(order, product) ;
  return api.shop.updateOrder(updatedOrder)
    .then(newOrder => {
      this.setState({ order: newOrder }); //update one state
    });
}
```

Manage State in React

DIGESTIBLE EXAMPLE

Compute From states

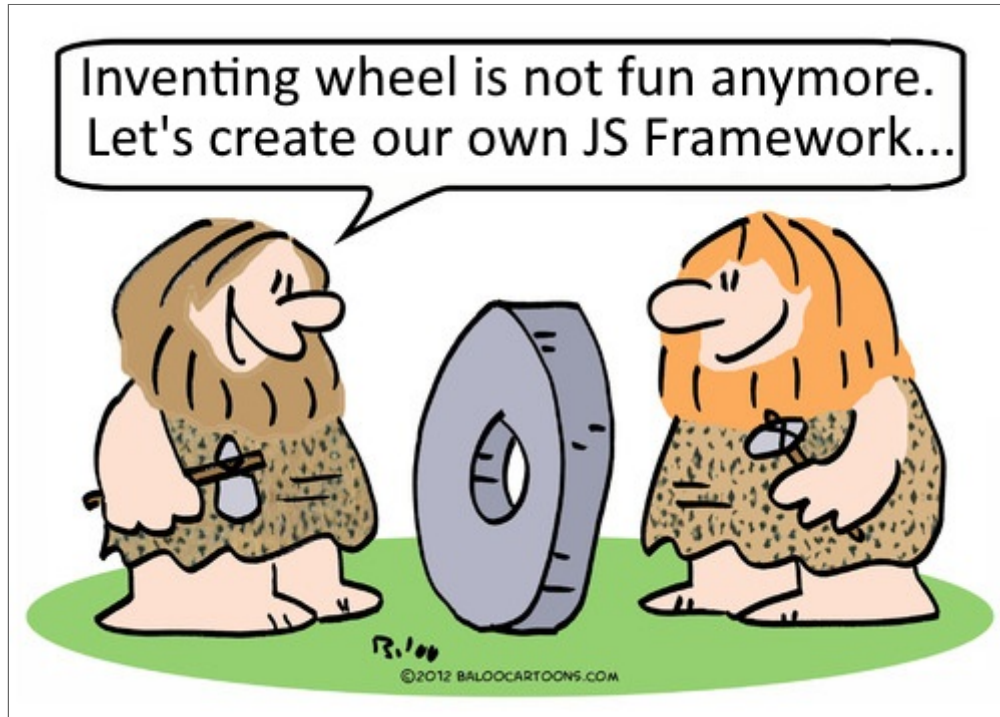
```
class MyComponent extends Component {
  constructor(props) {
    super(props);
    this.state = { message: props.message + "!" };
  }
  componentWillReceiveProps(props) {
    this.setState({ message: props.message + "!" });
  }
  render() {
    return (
      <div>{this.state.message}</div>
    );
  }
}
```


Just read from the props/state

```
render() {  
  const excitedMessage = this.props.message + "!";  
  return (<div>{excitedMessage}</div>);  
}
```

Avoid using state for data which can be calculated from props or other state

WHAT IF YOU CAN RECREATE THE WHEEL



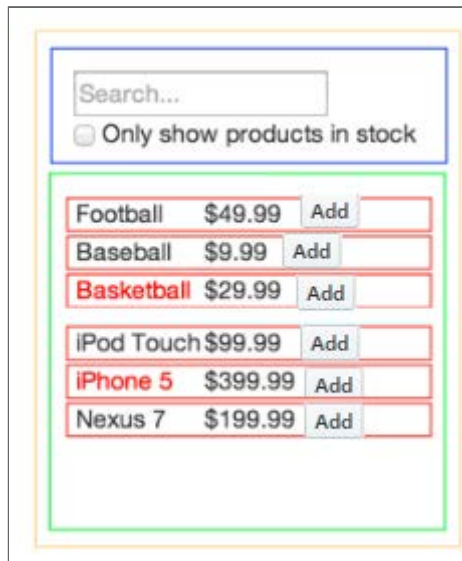
IDENTIFY COMPONENT HIERARCHY

- Draw boxes around component
- Use Single responsibility principle
- One component should ideally do one thing



NAME YOUR COMPONENTS

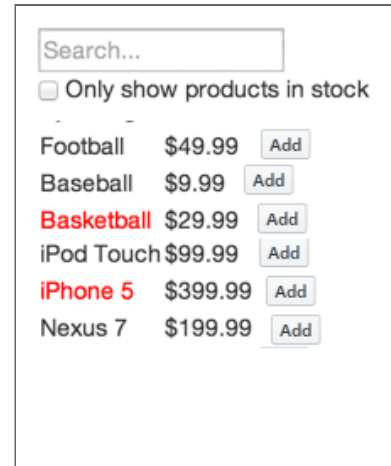
1. ProductRow (red)
2. ProductContainer (green)
3. SearchBar (blue)
4. FilterableProductContainer (orange)



DATA YOU MIGHT HAVE...

During Run time

- List of products
- Order list
- Search text the user has entered
- Value of the checkbox
- Filtered list of products



Search...

Only show products in stock

Football	\$49.99	Add
Baseball	\$9.99	Add
Basketball	\$29.99	Add
iPod Touch	\$99.99	Add
iPhone 5	\$399.99	Add
Nexus 7	\$199.99	Add

IDENTIFY UI STATE

It's not a React state if...

1. Just passed from parent via props
2. Remain unchanged over time
3. Can be computed by using other state/props

WHERE TO PUT THE STATE

1. Identify component that just has to render
2. Put state in the common owner in the higher up
3. If you don't find a common owner, create one

(Apply these 3 rules)

- ProductContainer doesn't have filters information
- SearchBar doesn't have product list
- FilterableProductList is the common owner
- FilterableProductList will contain the state

working code

STATE ME IF YOU CAN...

The screenshot shows a web interface for a product catalog. At the top, there is a search input field labeled 'Search...' and a checkbox labeled 'Only show products in stock'. Below this is a list of products, each with a name, price, and an 'Add' button. The products are: Football (\$49.99), Baseball (\$9.99), Basketball (\$29.99), iPod Touch (\$99.99), iPhone 5 (\$399.99), and Nexus 7 (\$199.99). The 'Basketball' and 'iPhone 5' items are highlighted in red. The interface is annotated with colored boxes: a blue box around the search and checkbox, a green box around the product list, and a red box around the 'Basketball' and 'iPhone 5' items.

Product	Price	Add
Football	\$49.99	Add
Baseball	\$9.99	Add
Basketball	\$29.99	Add
iPod Touch	\$99.99	Add
iPhone 5	\$399.99	Add
Nexus 7	\$199.99	Add

- Products: state (could be a property)
- Filtered product can be computed via search text and checkbox: not state
- Is item selected: not state
- Order list changes on toggle: state
- Search text user can entered: state
- Value in the checkbox: state

SUMMARY

- Try not to change data you get from server
- Avoid storing data in `this.state` which can be calculated from props/state
- Try to put derived property in `render()`
- Try to avoid single event changing multiple state

Handle state in React

USE CASE - 2

YOU HAVE A LITTLE MESS

When you need to Share state

HIGHER ORDER COMPONENT

1. Divide components into containers and presenters
2. Pass callbacks to hand as state change in children
3. State sharing components in container

Lifting up state

LIGHTER STATE MANAGEMENT

- Higher Order Component
- Local Storage
- Route Parameter
- Database (if you are already using it)

ema-john

SUMMARY

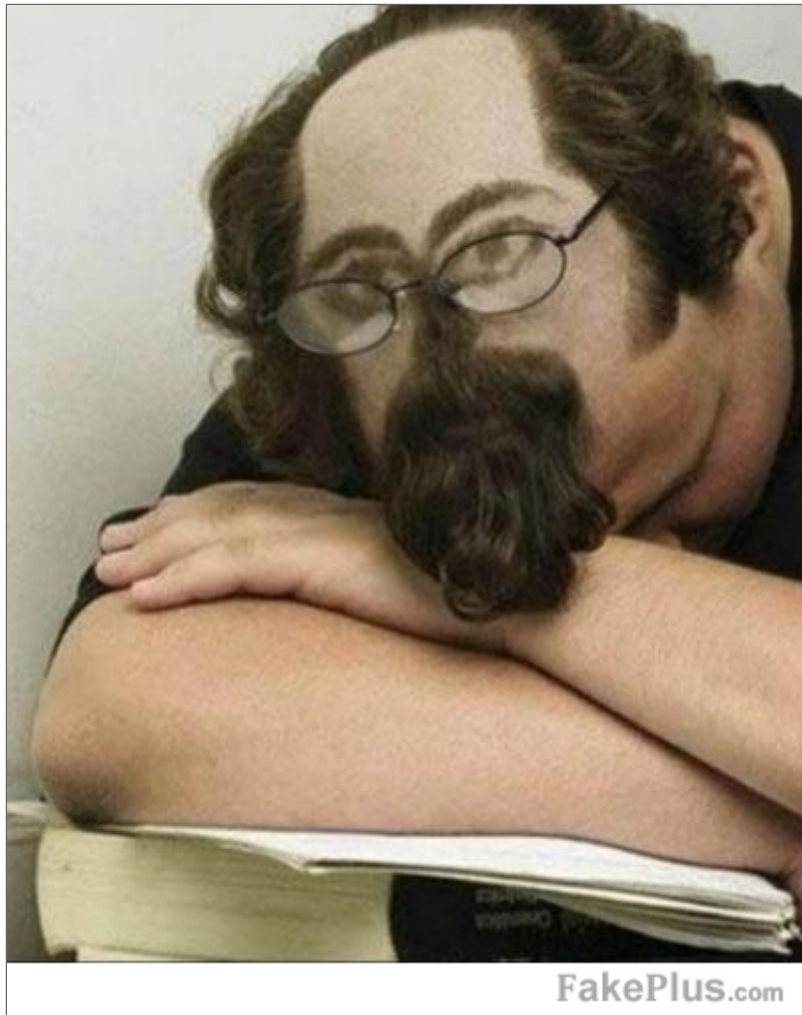
1. **Think in React**
2. **Read about composition**
3. Find where to put your state
4. **You Might Not Need Redux**
5. You can apply ideas from Redux without using it
6. Come back to Redux if you find a real need for it

React state without Redux



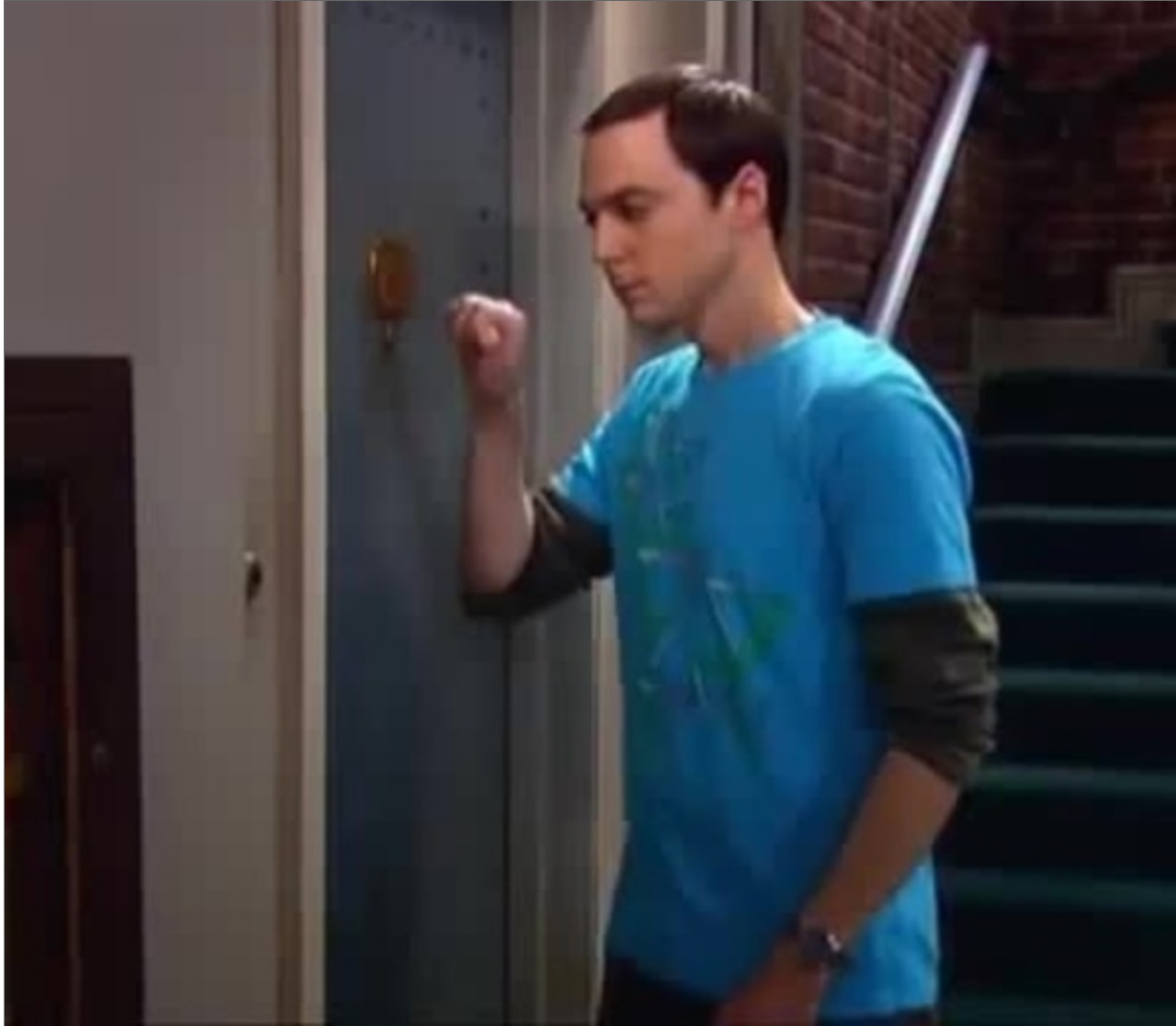






QUIZ

Who was the guy knocking at the door?

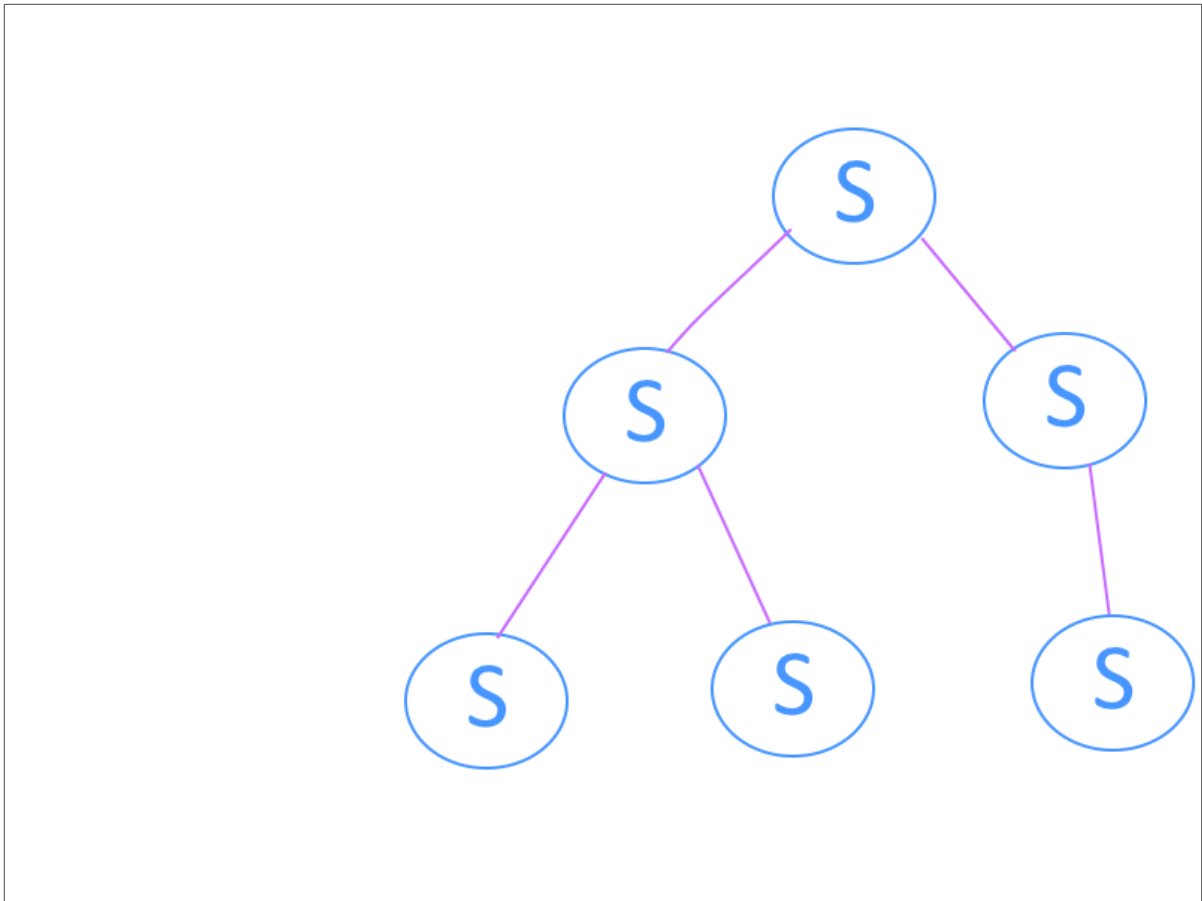


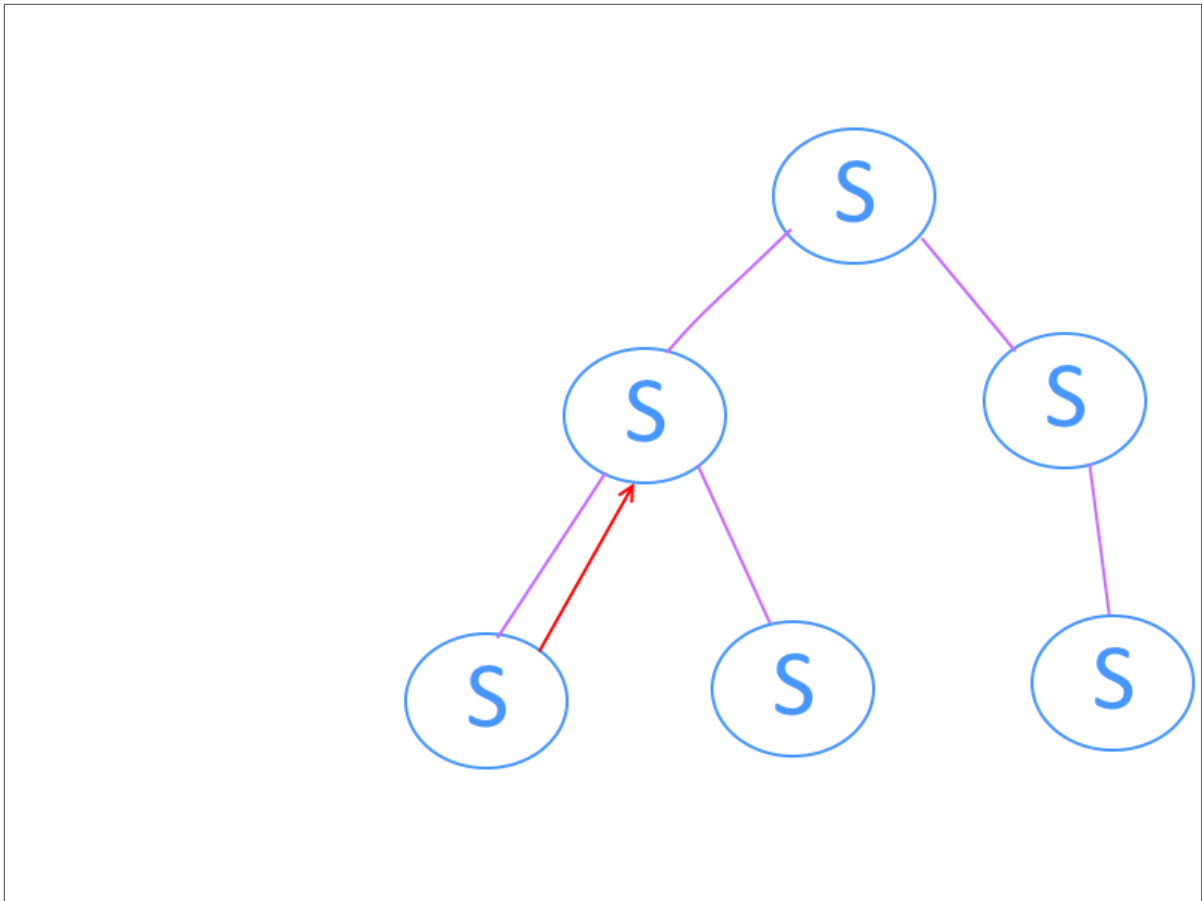
USE CASE-3

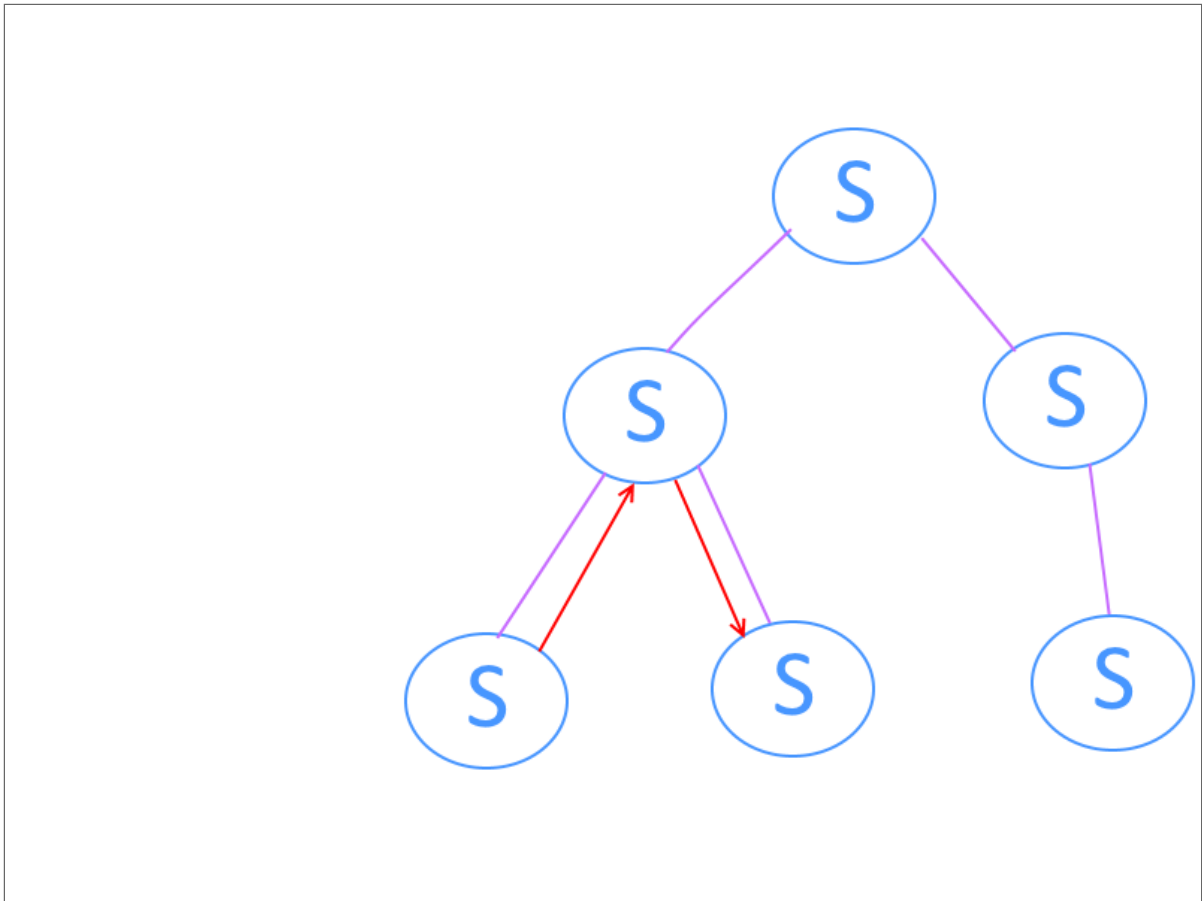
STATE MANAGEMENT LIBRARY: REDUX

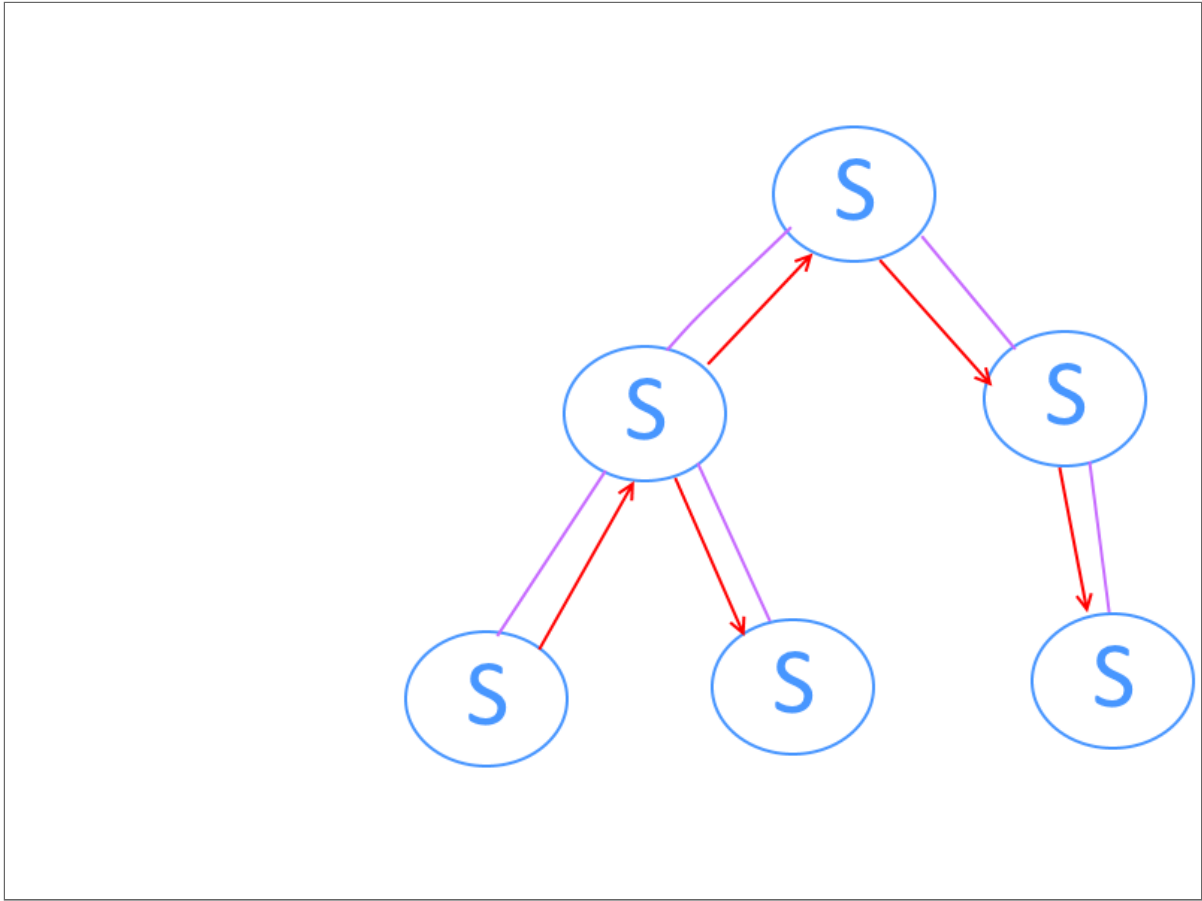
When you need to share state a lot



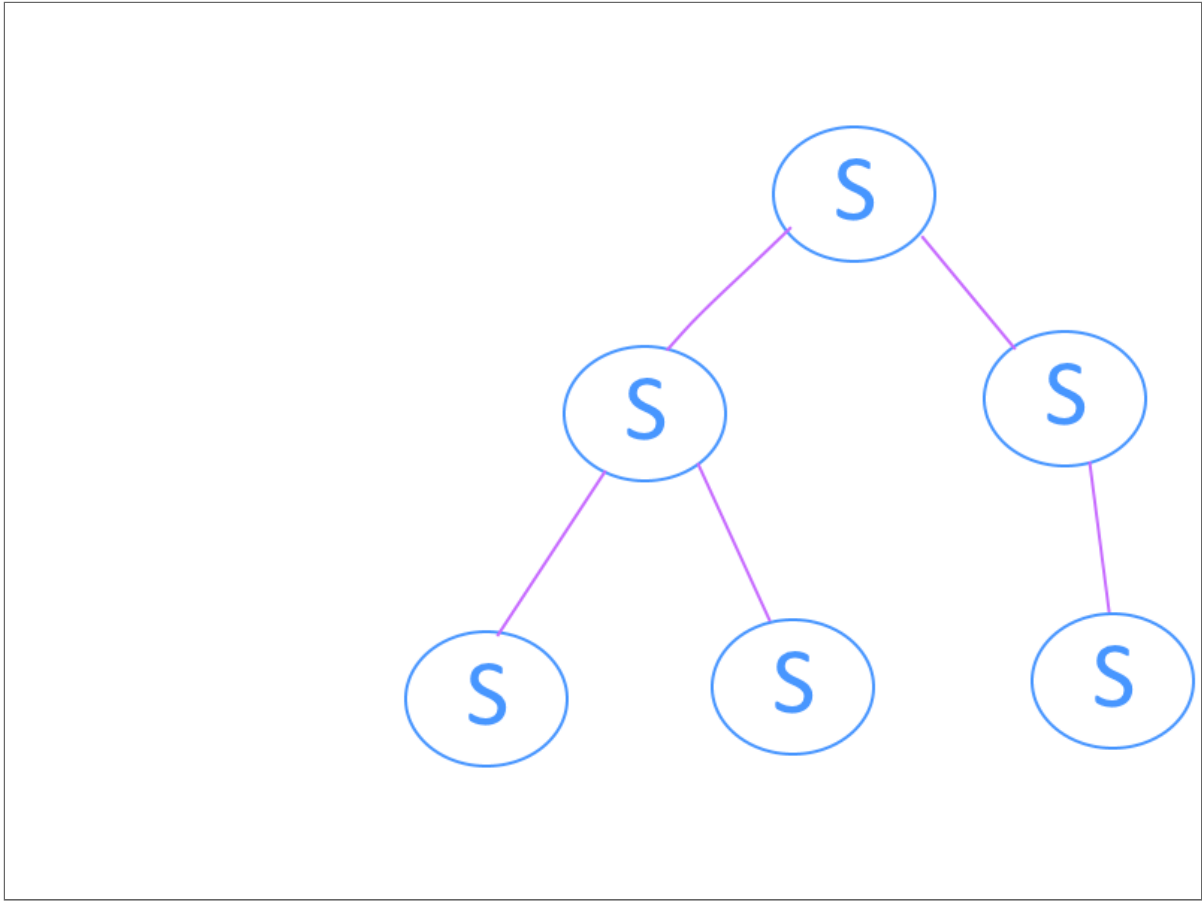


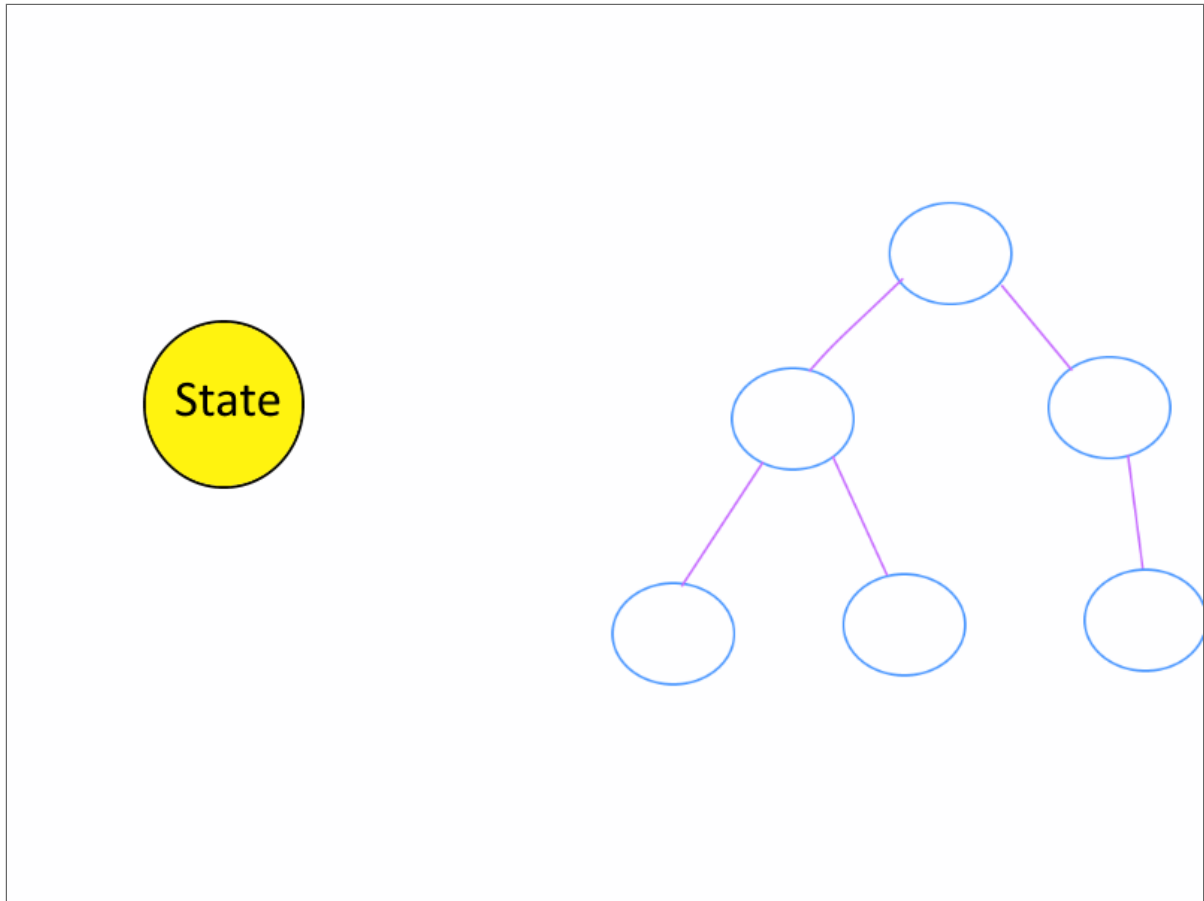


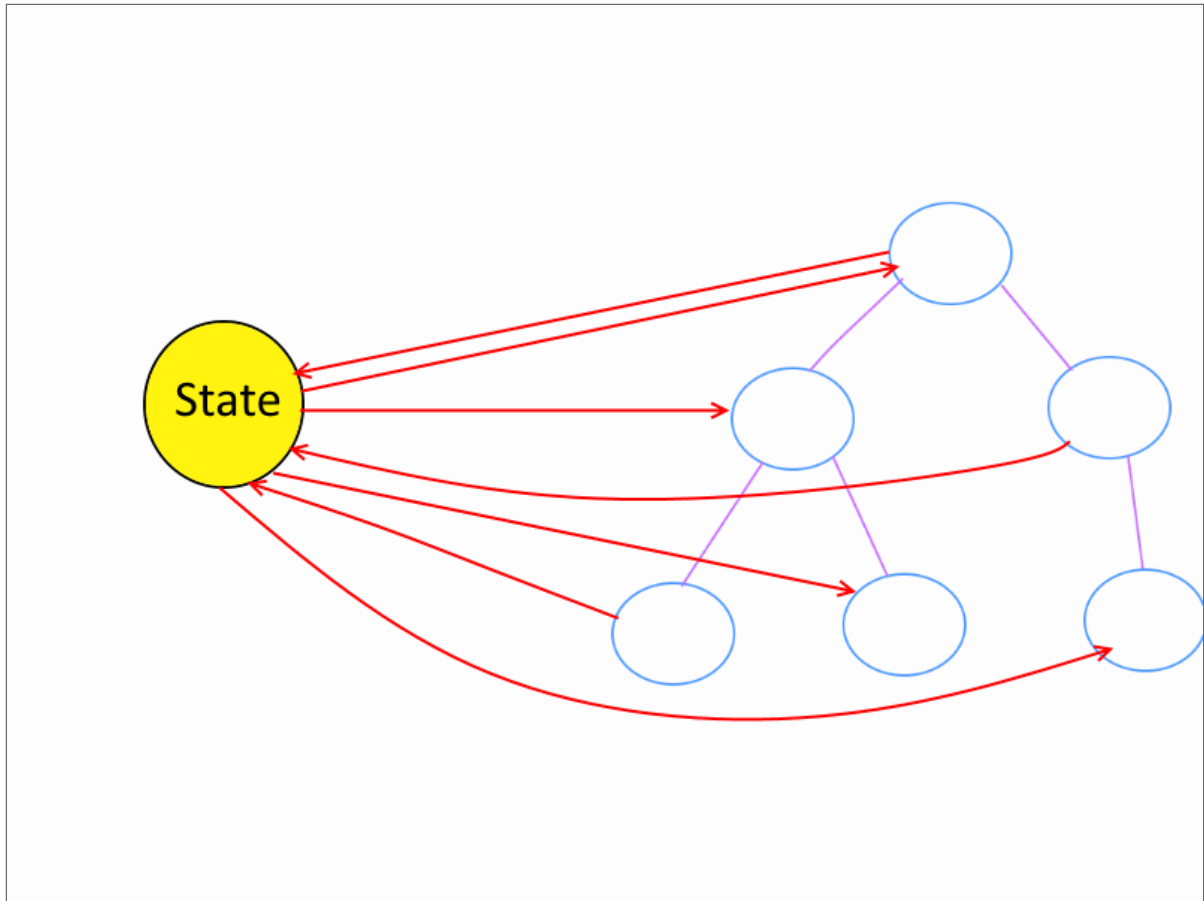


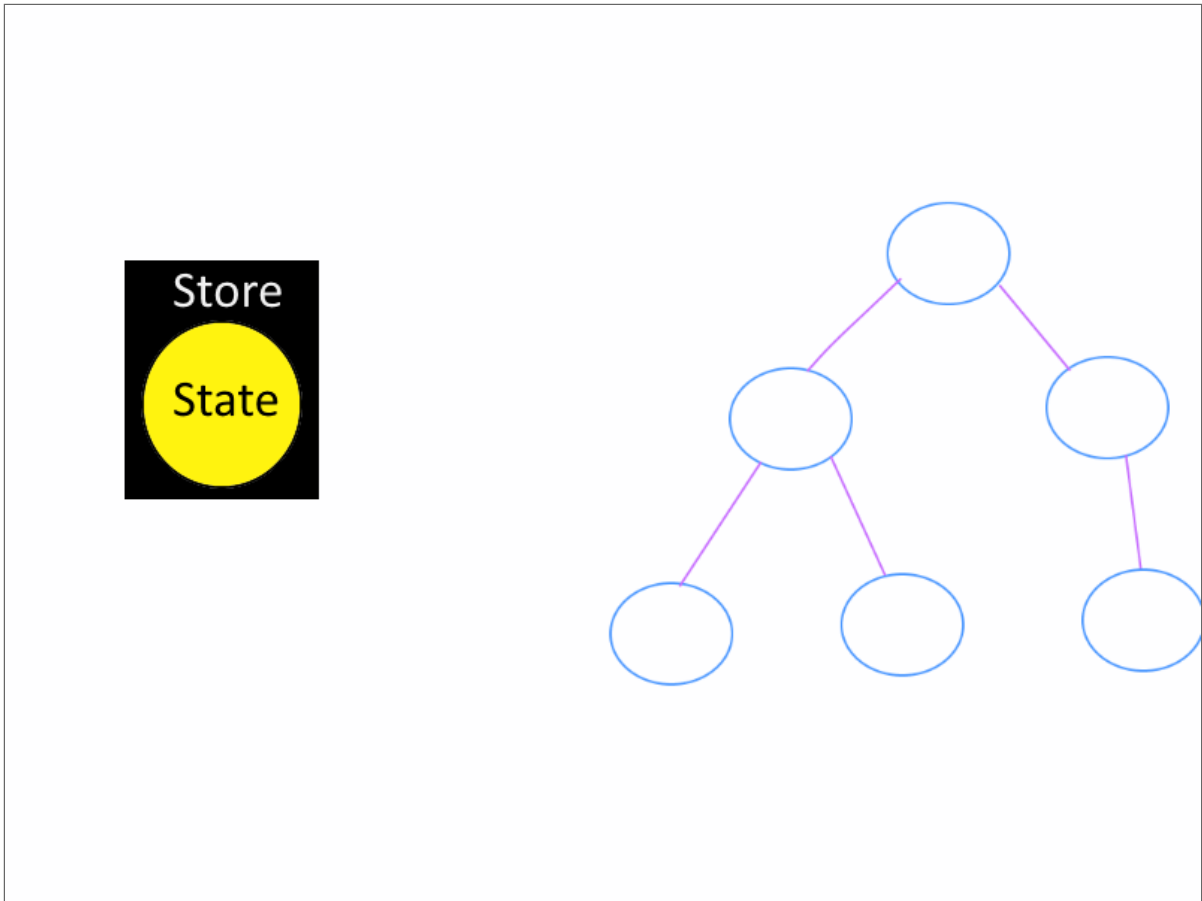


WHAT IF WE PUT THEM IN ONE PLACE



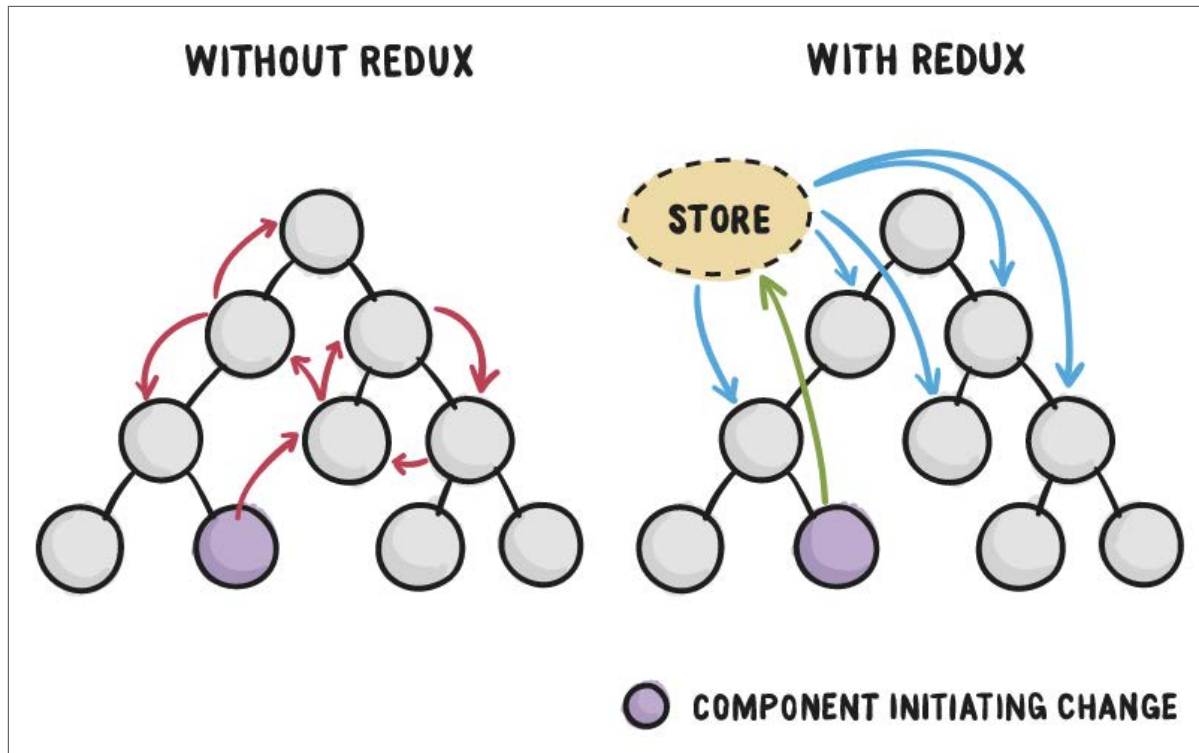




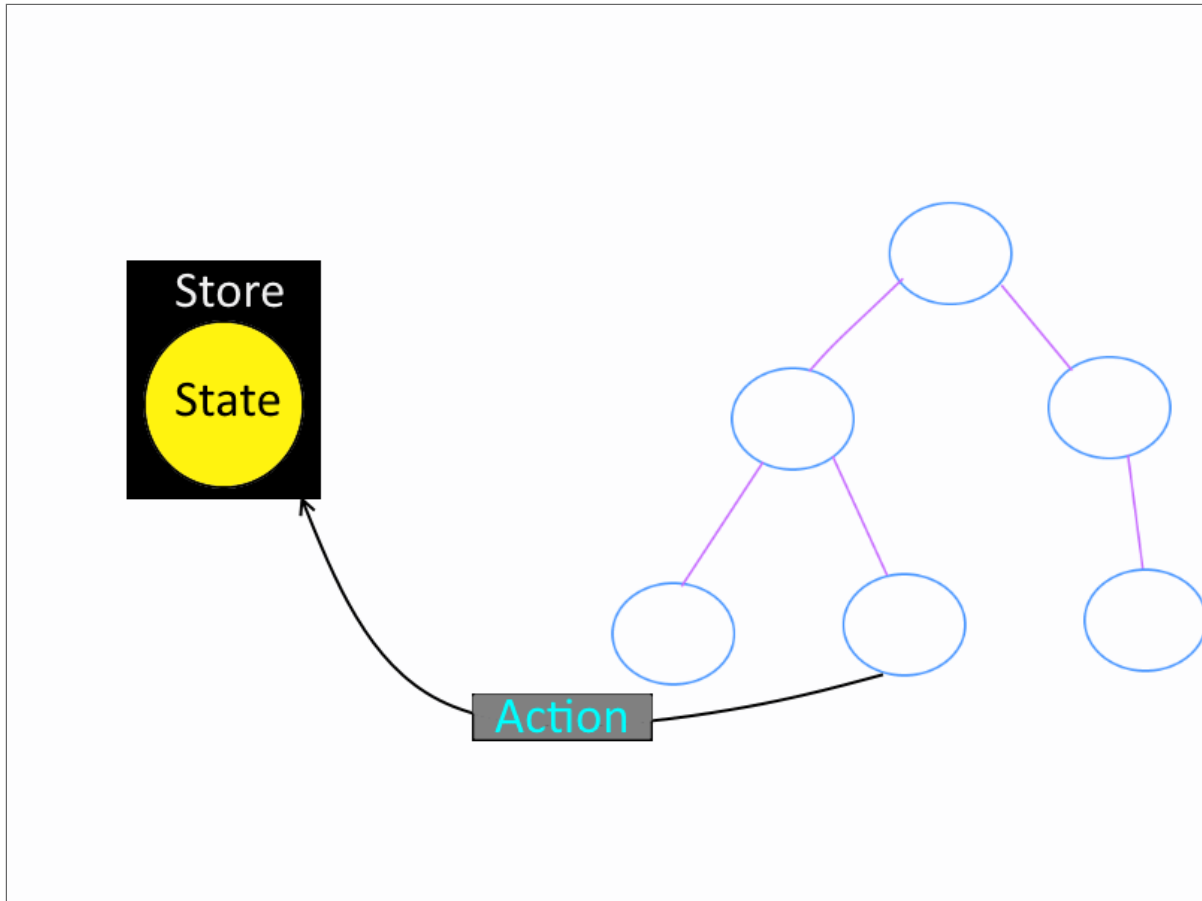


STORE

- Holds application state
- Only one store for all its application state
- Single source of truth



HOW WILL WE CHANGE THE STATE?

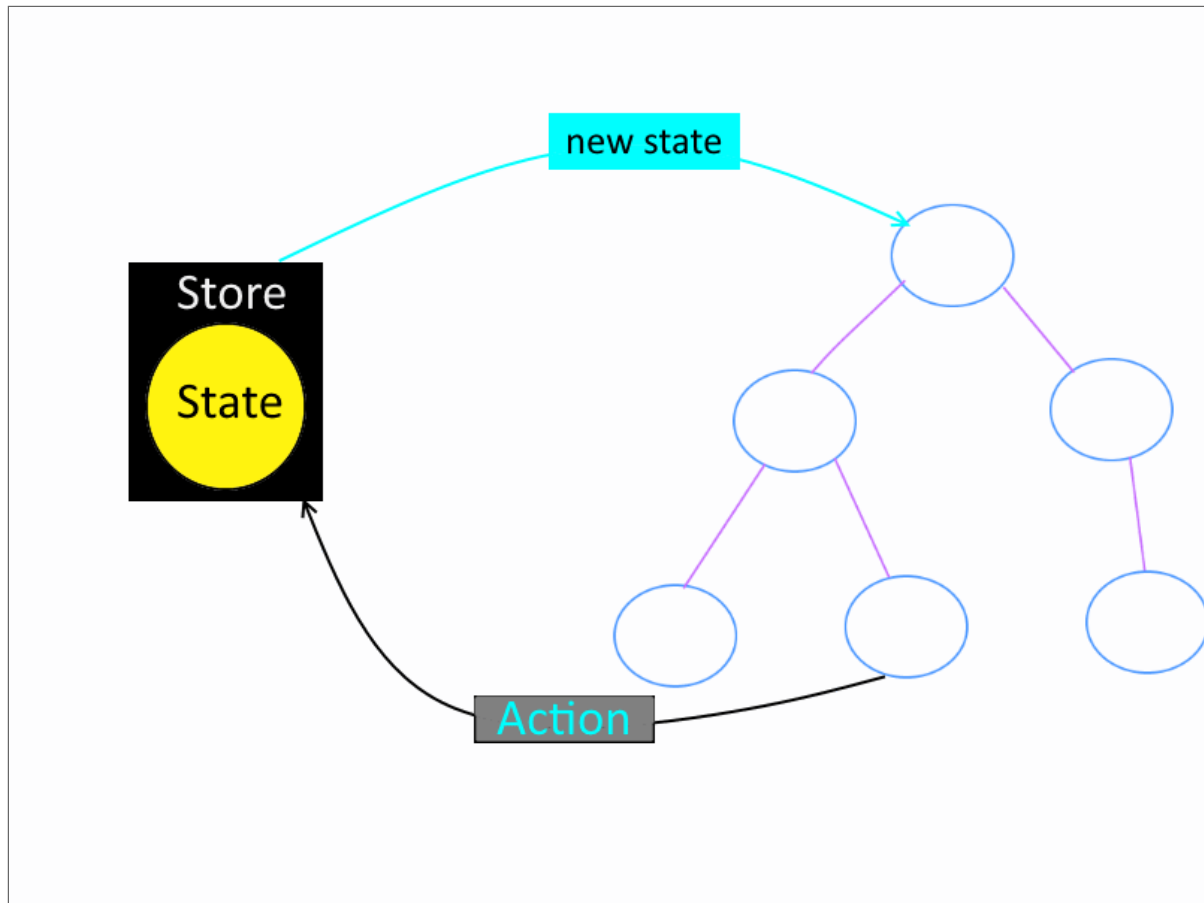


ACTION

- Actions
- Action container

```
{  
  type: 'ADD_TODO',  
  text: 'Take 8 bathroom breaks'  
}
```

- dispatch action



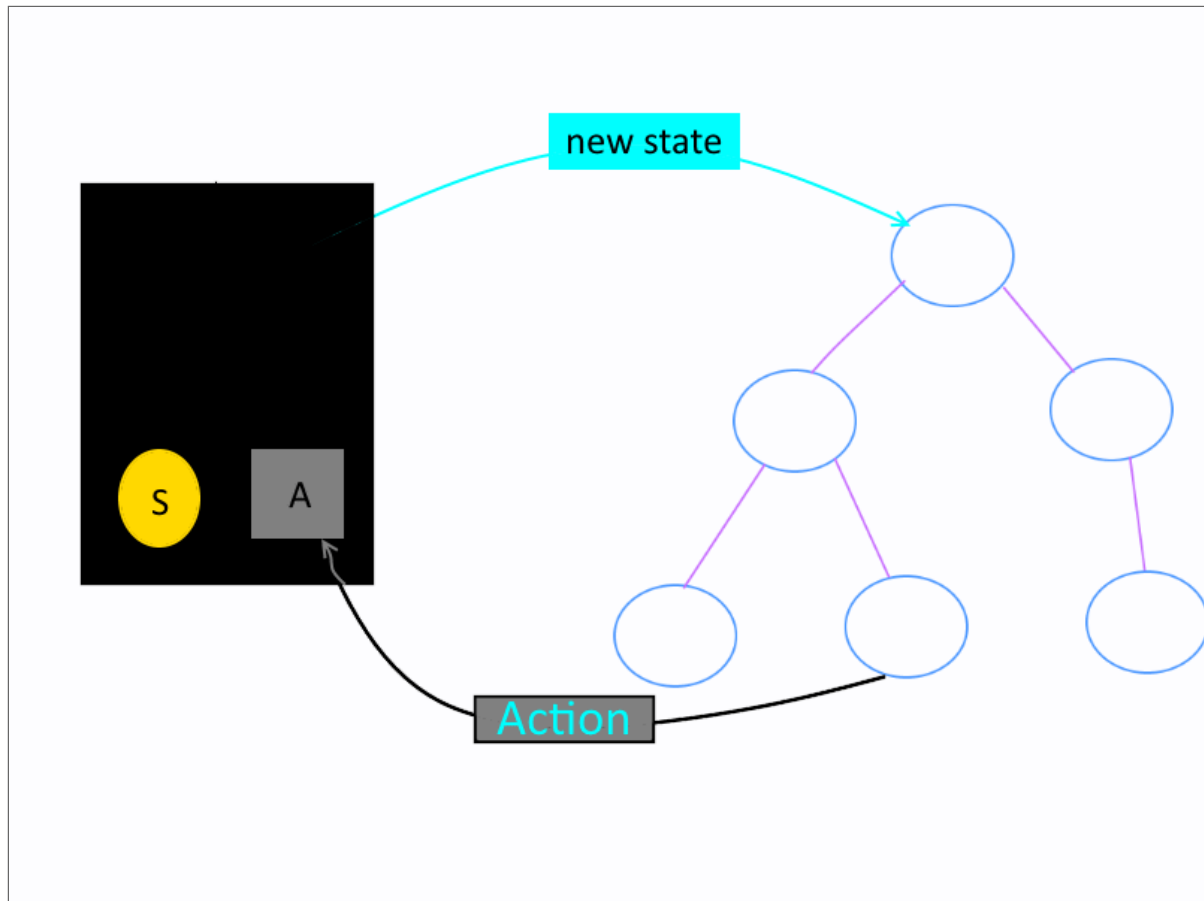
BUT YOU DIDN'T TELL HOW STATE WILL CHANGE?

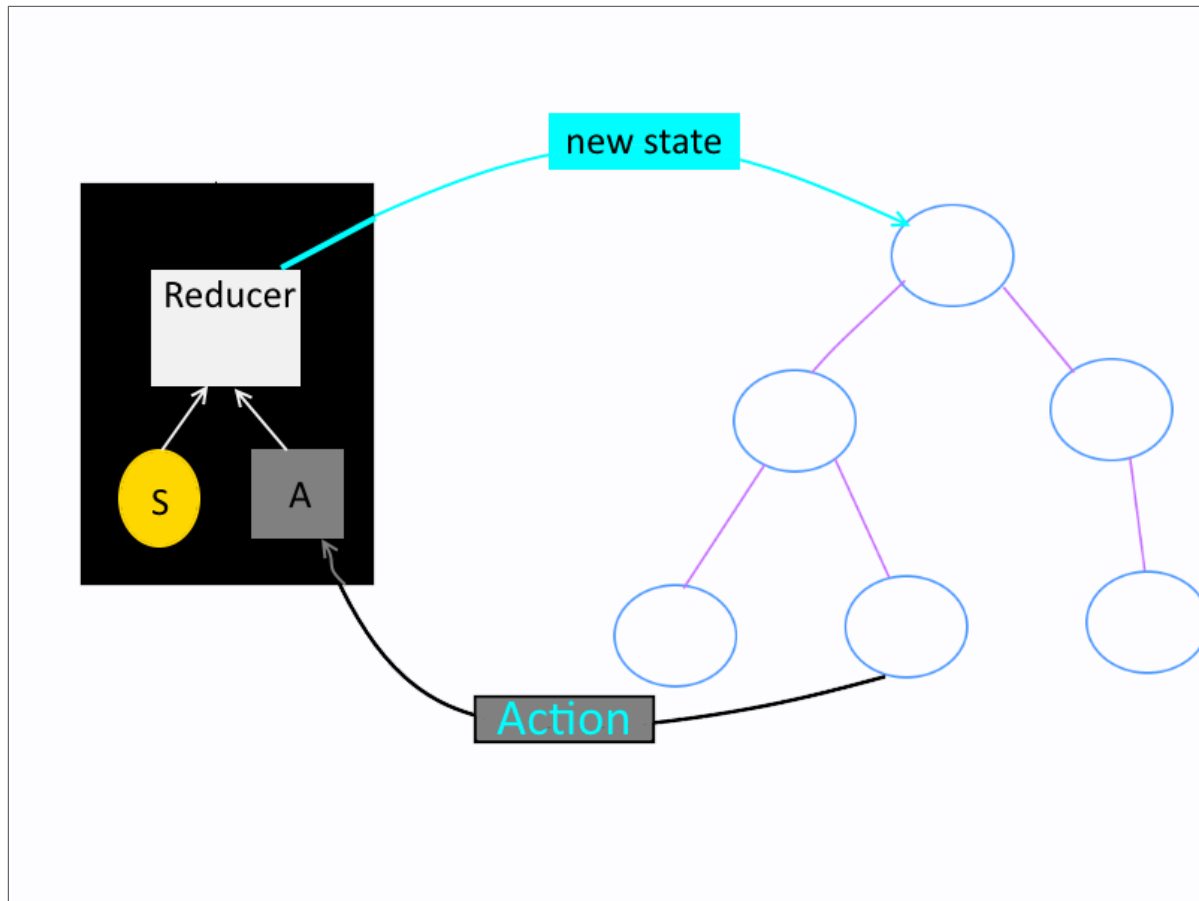
ACTUALLY STATE WILL NOT CHANGE

WE WILL CREATE A NEW STATE WITH THE CHANGE

```
function add(x, y){  
  return x + y;  
}
```

```
add(1, 2); // output is 3  
add(1, 2); // output still is 3  
add(1, 2); // WILL ALWAYS output 3
```



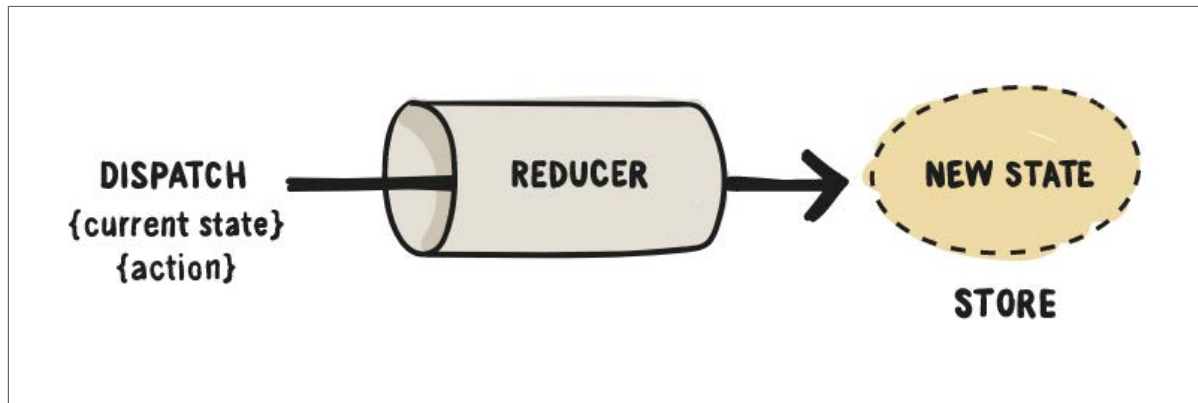


REDUCER

- Actions describe the fact that something happened
- Action don't specify how the application's state should change
- How state will change will be defined by **reducer**

```
case ADD_TODO:
  return [
    ...todos,
    {
      text: action.text,
      completed: false
    }
  ]
```


REDUCER



DATA FLOW

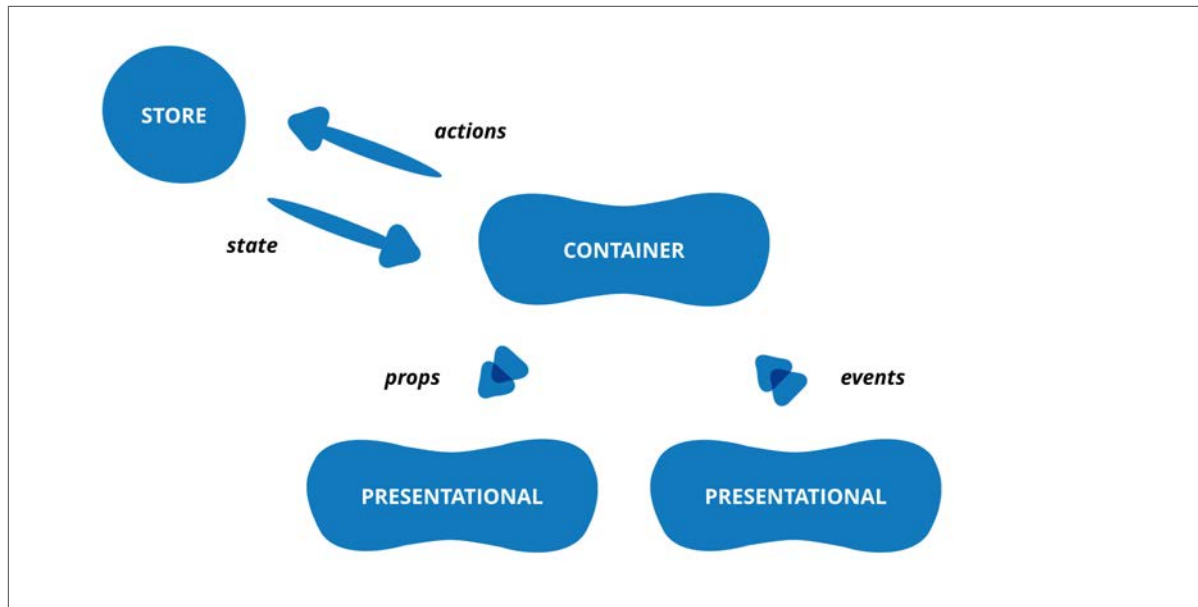
- Action is a plain object describing what happened
- Call `store.dispatch(action)` from anywhere in your app
- Store calls the reducer function you gave for the given action
- In root reducer you may combine multiple reducers into a single state tree
- Store saves the complete state tree returned by the root reducer

BASIC BUILDING BLOCK

- Store
- Action
- Dispatch
- Reducer

IMPLEMENT REDUX

```
npm install --save redux  
npm install --save react-redux
```



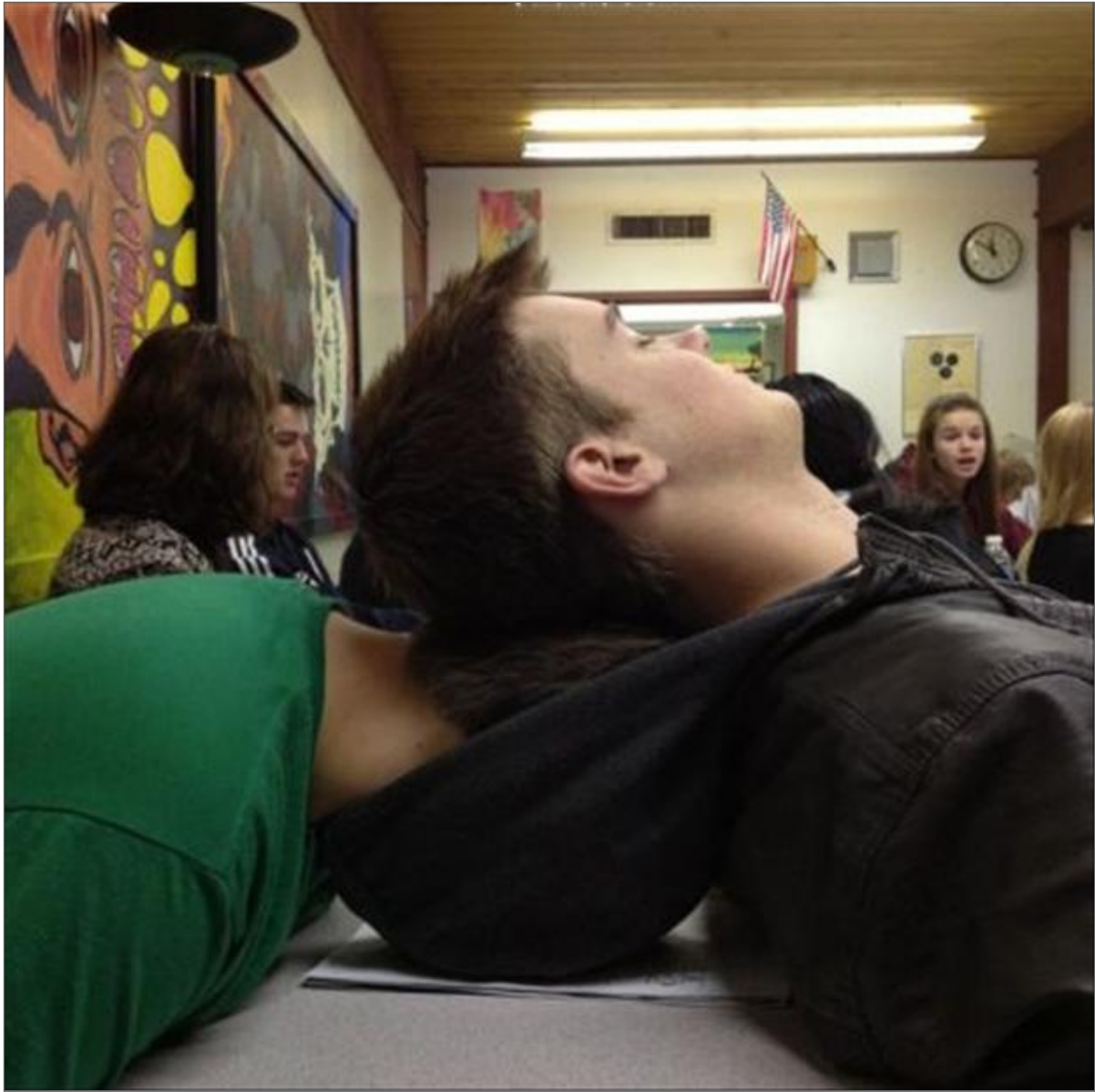
ref: [unit test react redux app](#)

	Presentational Components	Container Components
Purpose	How things look (markup, styles)	How things work (data fetching, state updates)
Aware of Redux	No	Yes
To read data	Read data from props	Subscribe to Redux state
To change data	Invoke callbacks from props	Dispatch Redux actions
Are written	By hand	Usually generated by React Redux

SEE SOME CODE

github.com/khan4019/redux-todo

state management



QUIZ

What's the name of the other speaker?

USE CASE -4

SCALE WITH REDUX

When you are building really large App

REDUX IS GOOD..

1. Keep an history of your app
2. Time-travel over the actions
3. Even store the app state in the local storage

But it's not a silver bullet

Redux Dev tool



MESSED UP WITH REDUX

- Do you have too many global action creators?
- Do you have a huge reducer that manages everything?
- Do you have conflicts between your actions or reducers that modify the same piece of state?

redux scaleable architecure



CONSIDER

- Redux actions creators and reducers with feature
- Create Data module for data that is not tightly bound to a container component
- Put distinct domains into sub-store and split reducers
- Isolate network communication in one place
- Use Redux-saga or thunk (middleware)
- Use **selector** to pass bits of state to components
- Consider **reselect** to optimize performance

slice reducer, Destructuring react store

IS REDUX THE ONLY OPTION?

MOBX IS ONE OPTION

Redux	MobX
single store	multiple stores
flat state	nested state
functional programming	OOP & reactive programming
immutable	mutable
plain JavaScript	“magic” JavaScript
more boilerplate	less boilerplate

MobX , Redux or MobX , MobX vs Redux

OTHER OPTIONS

- **Relay/Apollo & GraphQL**
- **Unstated, Explore unstated**
- **Rematch: Redux best practices without the boilerplate**
- **Local Storage**
- **Just React**

USE CASE-5

REDUX SAGA

When you need to handle async call with Redux

REDUX-SAGA

- Handles application side effects
- Asynchronous things like data fetching
- Access the browser cache

REDUX-SAGA

- A separate thread in your application
- A redux middleware
- Start, pause or cancel by redux actions
- Can access redux state
- Can dispatch redux actions

ES6 GENERATOR

```
function *foo() {  
  yield 1;  
  yield 2;  
  yield 3;  
  yield 4;  
  yield 5;  
}
```

```
var fooed = foo();  
fooed.next(); //{value: 1, done: false}  
fooed.next(); //{value: 2, done: false}  
fooed.next(); //{value: 3, done: false}  
fooed.next(); //{value: 4, done: false}  
fooed.next(); //{value: 5, done: false}  
fooed.next(); //{value: undefined, done: true}
```

SAGA HELPERS

- **put:** dispatches an action.
- **select:** get data from the state

- **fork:** performs a non-blocking operation
- **call:** runs a function. If it returns a promise, pauses the saga until the promise is resolved.

- **take:** pauses until action received.
- **takeEvery:** return results for all the calls triggered.
- **takeLatest:** If we trigger several cases, it's going to ignore all of them except the last one

CREATE SAGA

```
function* loadUser() {
  try {
    //1st step: call getUser, then assign value to user
    const user = yield call(fetch, '/getUser', {method: 'GET'});

    //2nd step: dispatch action
    yield put({type: 'FETCH_USER_SUCCESS', payload: user});
  }
  catch(error) {
    //optional step: if something fails dispatch fail
    yield put({type: 'FETCH_FAILED', error});
  }
}
```


REGISTER SAGA GENERATOR

```
function* sagas() {  
  yield[  
    fork(loadUser),  
    takeLatest('LOAD_DASHBOARD', loadDashboardSequenced)  
  ];  
}
```

INJECT THE SAGA MIDDLEWARE

```
import { applyMiddleware, createStore } from 'redux';
import createSagaMiddleware from 'redux-saga';
import reducers from './reducers';
import sagas from './sagas';

const sagaMiddleware = createSagaMiddleware();
let middleware = applyMiddleware(sagaMiddleware);

const store = createStore(reducers, middleware);
sagaMiddleware.run(sagas);

// inject sagas into the middleware
sagaMiddleware.run(sagas);
```

ADVANCED USAGE

- Nesting Sagas
- Sequential Sagas
- Race saga
- Multiple Sagas
- Task Cancellation
- Testing Saga

REFERENCES

- **Async operations using redux-saga**
- **Handling async in Redux with Sagas**
- **From actions creators to sagas**
- **Redux Saga With Example**
- **Use set state as a function**
- **Testable code with Saga**
- **list of important resources**
- **Saga vs thunk**

FREE TIP

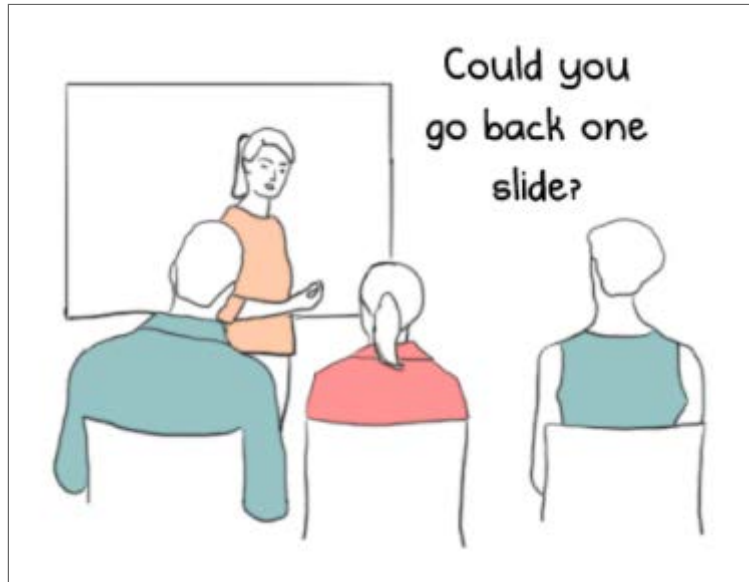
4 STEPS TO APPEAR SMART IN A MEETING

STEP - 1

Sleep first 30min

Or use your cell phone

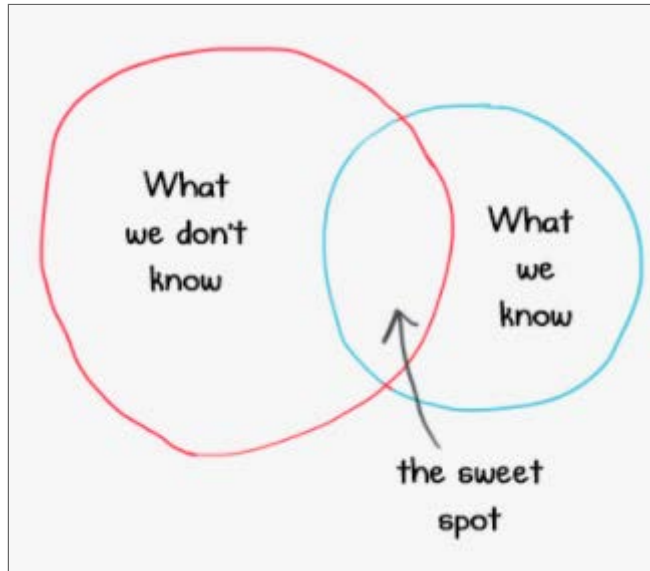
STEP-2



STEP-3



STEP-4



- Spend 50min to talk about the Sweet spot
- Ask the junior developer to reschedule the meeting

7 TIPS TO APPEAR BUSY AT WORK

APPEAR BUSY AT WORK

1. Ctrl + 1 when big brother is around
2. Always carry extra bag and print out
3. Don't leave before ur manager
4. Dont use the same rest room in a row
5. Do lunch at your desk not break room
6. Always leave a extra jacket on your chair
7. Compose email during office hours, send it
midnight or weekends

Look busy

THANK YOU

-Slides: khan4019.github.io/manage-state

-Website: thatjsude.com

-Youtube: youtube.com/c/ThatJSDude

-Email: khan4019@gmail.com