Redux

Redux is a predictable state container for JavaScript apps.

The State

Imagine your app’s state is described as a plain object.

```
{
  todos: [{
    text: 'Eat food',
    completed: true
  }, {
    text: 'Exercise',
    completed: false
  }]
}
```

Actions

To change something in the state, you need to dispatch an action. An action is a plain JavaScript object.

```
{ type: 'ADD_TODO', text: 'Go to swimming pool' }
{ type: 'TOGGLE_TODO', id: 1 }
```

Action Creators

Pure functions that return an action.

```
const addTodo(description: string) => ({
  type: 'ADD_TODO',
  text: todo.text
})
```

Usage:

```
addTodo('Take the hobbits to Isengard')
```

Pure functions

A pure function is a function where the return value is only determined by its input values.

```
f(x) => Math.rand() 😠
```

```
f(x) => x * 2 😊
```

The Reducer

A function that takes state and action as arguments, and returns the next state of the app

```
f(state, action) => state
```

IRL Example

The Reducer of Life:

```
f(h: Hunger, action: Action) => Hunger
```

Let's use it.

```
({hungry: true}, dinner) => return {hungry: false}

({hungry: false}, sleep) => return {hungry: true}
```

The Reducer

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

3 Principles

Single Source of Truth:

The state of your whole application is stored in an object tree within a single store.

State is read-only

The only way to change the state is to emit an action, an object describing what happened.

Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure reducers.

Usage with React

React Redux is the official React binding for Redux. It lets your React components read data from a Redux store, and dispatch actions to the store to update data.

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

The Store

The Store is the object that brings them together. The store has the following responsibilities:

- Holds application state;
- Allows access to state via getState();
- Allows state to be updated via dispatch(action);
- A little bit more.

The Store at a Glance

```
import { createStore } from 'redux'
import reducer from './reducers'
const store = createStore(reducer)
```

Provider

React Redux provides `<Provider />`, which makes the Redux store available to the rest of your app:

```
ReactDOM.render(
  <Provider store={store}>
    <App />
  </Provider>,
  rootElement
)
```

Connecting the Dots

React Redux provides a connect function for you to connect your component to the store.

```
connect(
  mapStateToProps,
  mapDispatchToProps)(MyComponent)
```

mapState To Props

This will populate properties of your component with values from the state.

```
state => ({
  prop: state.prop,
  otherProp: state.otherProp
})
```

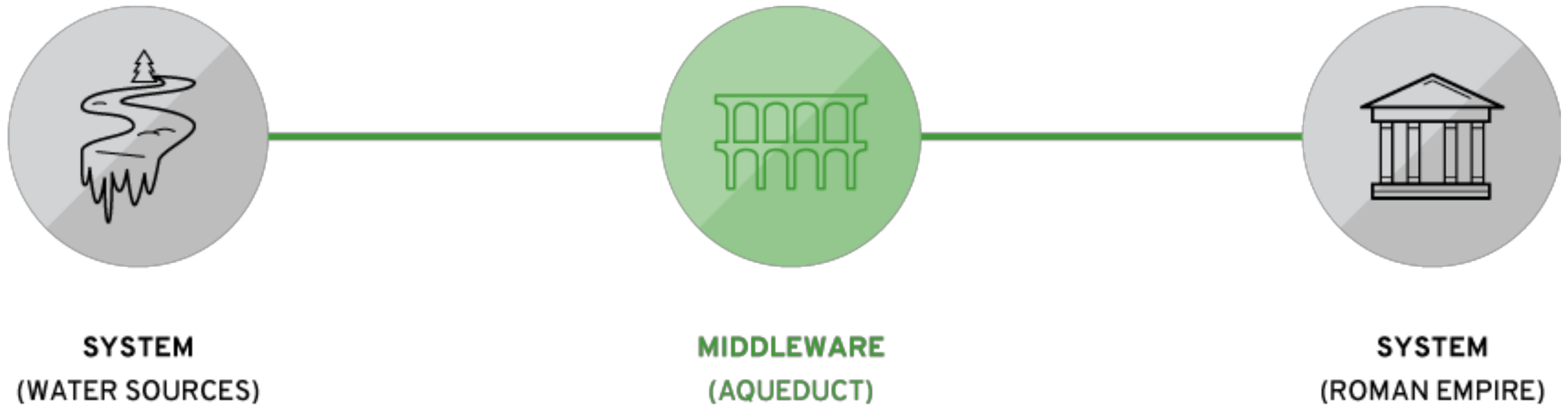
mapDispatchToProps

This function will populate your properties with1 action creators that will dispatch those actions to the store to modify the state.

```
dispatch => ({
  anAction: params => dispatch(anAction(params))
})
```

Middlewares

Is something you can put between the framework receiving a request, and the framework generating a response.



Example of a middleware

```
// logger.js
const logger = store => next => action => {
  console.log('dispatching', action)
  let result = next(action)
  console.log('next state', store.getState())
  return result
}

// index.js
import { createStore, combineReducers, applyMiddleware } from 'redux'

const store = createStore(
  reducer,
  applyMiddleware(logger)
)
```

Let's build something

Thunk Actions

Redux Thunk middleware allows you to write action creators that return a function to perform asynchronous dispatch.

A *thunk* is a function that wraps an expression to delay its evaluation.

The term originated as a humorous past-tense version of "think".

Installation

```
npm install redux-thunk
```

Then, to enable Redux Thunk, use applyMiddleware():

```
import thunk from 'redux-thunk';
// ...
const store = createStore(
  rootReducer,
  applyMiddleware(thunk)
);
```

Now what?

```
const getMovies() => {
  return dispatch => apiCall().then(result => {
    return dispatch(addMovie(result));
  })
}
```

Let's Thunk this Through

We'll pre-populate our list of movies.

`https://ghibliapi.herokuapp.com/films`