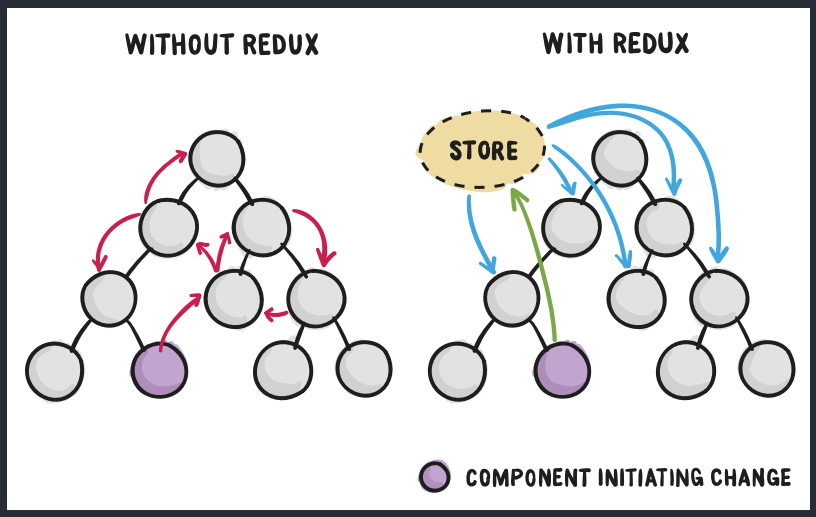
# REACT-REDUX

Predictable state container for JavaScript apps.

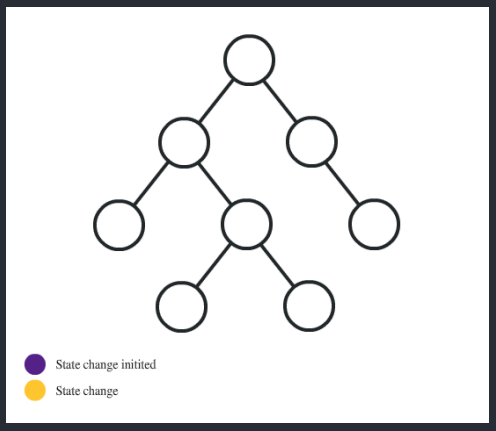
# Why Redux?

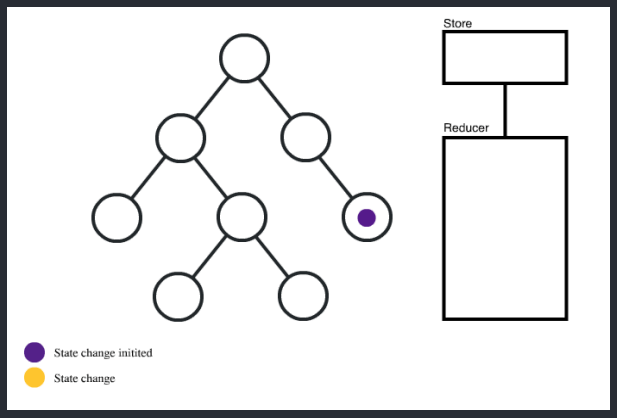
As your App keeps changing and growing, it's state will be harder to mantain, and you could lost control over the when, why, and how of its state.




This is where Redux comes in handy. Redux offers a solution of storing all your application state in one place.

# React vs Redux



# Three Principles

- **Single source of truth**

The state is stored in an object tree within a single store.

- **State is read-only**

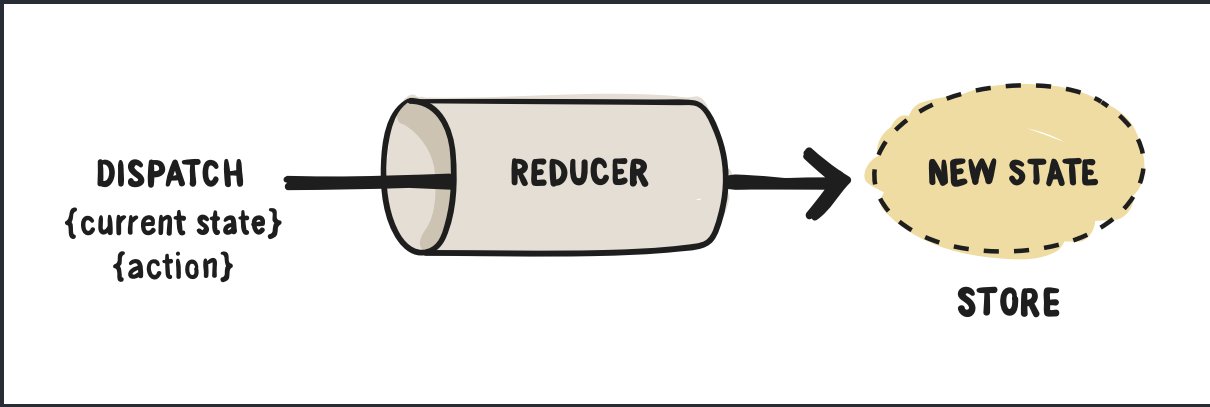
The only way to change the state is to emit an action.

- **Changes are made with pure functions**

The state tree is transformed by actions, you write pure reducers.

# Redux Data Flow

- Call store.dispatch(action)
- The store calls the reducer function
- The root reducer combine multiple reducers into a single state tree.
- The store saves the complete state tree returned by the root reducer.



# Actions

Actions are payloads of information that send data from your application to your store.

Action creators are functions that create actions

```
import { ADD_TODO } from '../actionTypes'
 ....
 function addTodo(text) {
   return {
     type: ADD_TODO,
     text
   }
 }
```

# Reducers

The application's state changes is the job of reducers.

- Don't mutate the state
- Return previous state in the default case
- Reducers must be pure functions

```
function todoApp(state = initialState, action) {
 switch (action.type) {
   ...
   case ADD_TODO:
     return Object.assign({}, state, {
       todos: [
         ...state.todos,
         {
           text: action.text,
           completed: false
         }]
       })
   default:
     return state
....
```

# Store

The Store is the object that brings actions and reducers together, note that you'll only have a single store in a Redux application.

- Holds application state;
- Allows access to state via getState();
- Allows state to be updated via dispatch(action);
- Registers listeners via subscribe(listener);

```
import { createStore } from 'redux'
import todoApp from './reducers'
let store = createStore(todoApp)
```

# Components

React bindings for Redux embrace the idea of separating presentational and container components.

| | 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 |

# Redux Practice

Lets implement Redux in our Photo Album App!

# JEST UNIT TESTING

Delightful JavaScript Testing

# Why Jest?

Jest is used by Facebook to test all JavaScript code including React applications.

Jest always run in a Node environment and not in a real browser. This lets us enable fast iteration speed and prevent flakiness.

- Zero configuration
- Built-in code coverage reports
- Snapshot Testing

# Running Tests

Jest will look for test files with any of the following popular naming conventions:

- Files with .js suffix in __tests__ folders
- Files with .test.js suffix
- Files with .spec.js suffix

```
npm test
```

# Writing Tests

To create tests, add it() block with the name of the test and its code.

You may optionally wrap them in describe() blocks for logical grouping.

Jest provides a built-in expect()

```
import sum from './sum';

it('sums numbers', () => {
  expect(sum(1, 2)).toEqual(3);
  expect(sum(2, 2)).toEqual(4);
});
```

# Enzyme

Enzyme is a JavaScript Testing utility for React that makes it easier to assert, manipulate, and traverse your React Components' output.

- **Shallow**

Testing a component as a unit, and to ensure that your tests aren't indirectly asserting on behavior of child components

- **Mount**

Full DOM rendering is ideal for use cases where you have components that may interact with DOM APIs

- **Render**

used to render react components to static HTML and analyze the resulting HTML structure

# Testing Components

```
describe('Photo', () => {
  const { component, props } = setup();

  it('renders all properties', () => {
    expect(component.find({
      title: props.photo.title,
      description: props.photo.description,
      url: props.photo.url,
    }));
  });

  it('should render a div with photo class', () => {
    expect(component.find('.photo'));
  });
});
```

# Testing Redux

```
 // Reducer
it('should handle ADD_ALBUM', () => {
  const action = {
    type: actionTypes.ADD_ALBUM,
    album,
    key
  }
  const result = albumReducer(initialState = {}, action);
  expect(result).toEqual({
    [key]: album,
    ...initialState
  })
});

// Action
it('should create an action to add an Album', () => {
  const expectedAction = {
    type: actionTypes.ADD_ALBUM,
    album,
    key
  }
  expect(albumActions.addAlbum(album)).toEqual(expectedAction);
});
```

# Coverage Reporting

Jest has an integrated coverage reporter that works well with ES6 and requires no configuration.

```
npm test -- --coverage
```

```
PASS src/App.test.js
  App
    ✓ renders a welcome view (6ms)

----------|----------|----------|----------|----------|----------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------|----------|----------|----------|----------|----------------|
All files |      100 |      100 |      100 |      100 |                |
 App.js   |      100 |      100 |      100 |      100 |                |
----------|----------|----------|----------|----------|----------------|
```

# Jest Practice

Lets implement Jest unit testing in our Photo Album App!

# References

- Redux Docs
- Why Redux?
- Presentational and Container Components
- When do I know I'm ready for Redux?
- Leveling Up with React: Redux
- Jest Docs
- Running Tests
- Enzyme Docs
- Testing React components with Jest and Enzyme

# THANKS FOR COMMING!!

Arturo De la Garza

https://github.com/agzertuche