
REDUX
& FRIENDS

Head of Engineering @ eMoteev

former MTI && ACU @ EPITA

Intro

QUENTIN PRÉ

This is a just a text holder in which you assume that I wrote a lot of interesting things, which obviously I have not.
If you can’t read it easily, you are too far away, get closer to the scene.
If you still have difficulties reading this, raise your hand and ask your question.

RULES

You are encouraged to make mistakes,
You are *forbidden* to make faults.

EFFORT

Do not expect knowledge to fall into your hands.

FOCUS

If you do not intend to follow the lecture, assume consequences for your actions and stay home.

You don’t need your laptop outside of tutorials.

ASK

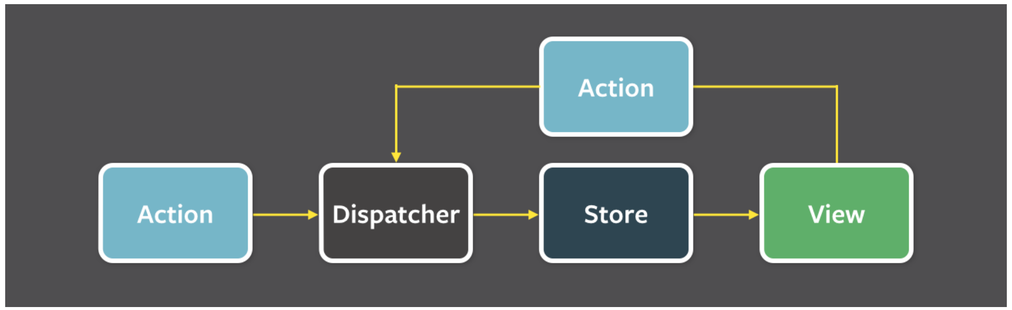
No question makes you stupid,
Asking no questions though…

note: your question might get answered by another question.

State Management

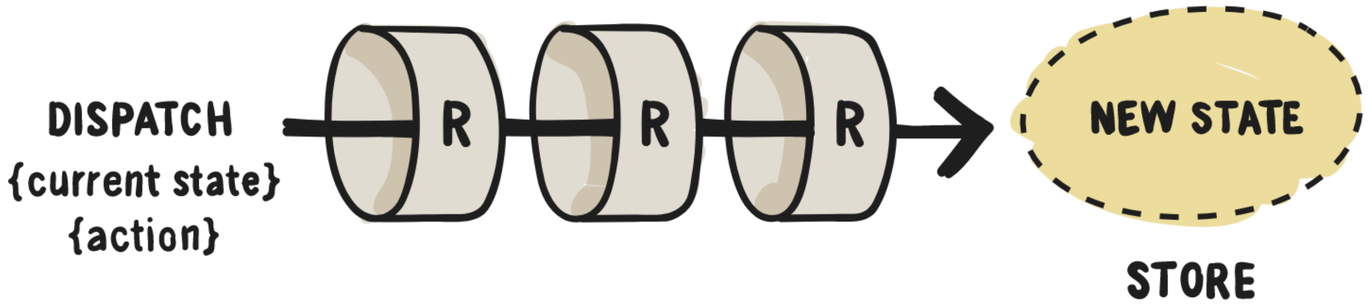
What the Flux ?

- Applications are gaining in complexity
- It’s getting hard to keep track of what, why and how something happens.
- Facebook’s Flux relies on unidirectional data flow



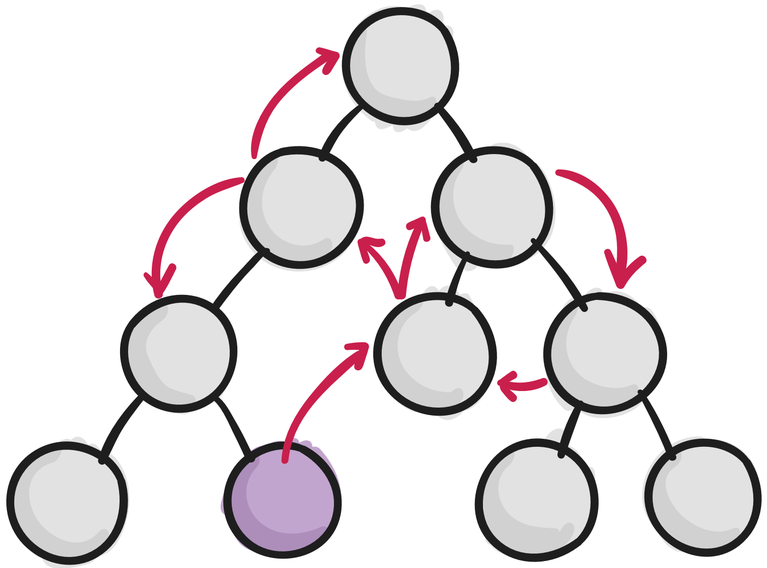
- Redux takes inspiration (even though it’s not a strict implementation of it)

Redux: Three Principles

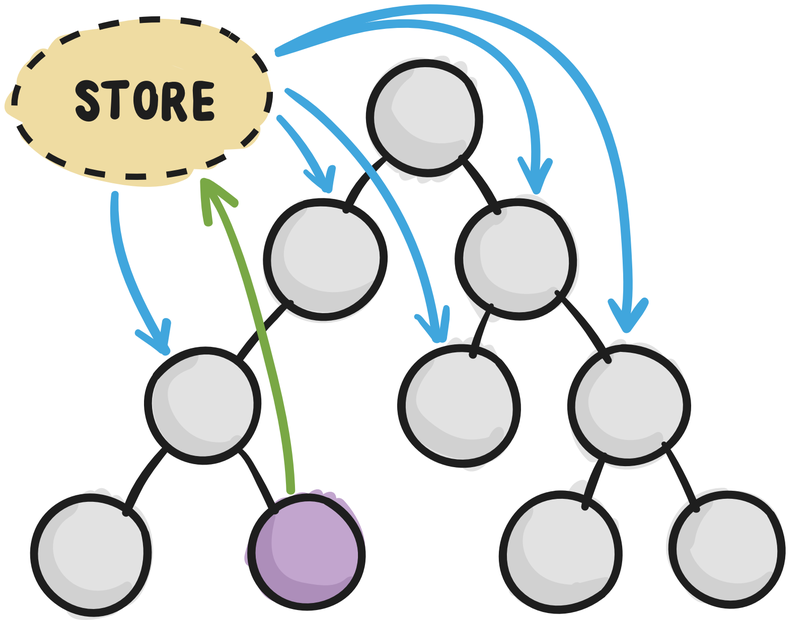


- **Single Source of Truth**: There is a single state, in a single store.
- **State is read-only**: the only way to change state is to dispatch an action with will spawn a new state.
- **Changes are made with pure functions**: aka ‘reducers’

WITHOUT REDUX



WITH REDUX



COMPONENT INITIATING CHANGE

A basic Redux app

```
import { createStore } from 'redux'

function counter(state = 0, action) {
  switch (action.type) {
  case 'INCREMENT':
    return state + 1
  case 'DECREMENT':
    return state - 1
  default:
    return state
  }
}

let store = createStore(counter)

store.subscribe(() => console.log(store.getState()))

store.dispatch({ type: 'INCREMENT' }) // => 1
store.dispatch({ type: 'INCREMENT' }) // => 2
store.dispatch({ type: 'DECREMENT' }) // => 1
```

Reducers

- Reducers are pure functions
- They return a new state.
- combineReducers
- A reducer's name will be the key to access its state from the store.

```
import { combineReducers, createStore } from 'redux';

const dumb = (state = [], action) => state;
const dumber = (state = [], action) => state;

let rootReducer = combineReducers([
  dummy,
  counter,
]);

let store = createStore(rootReducer);

const { dumb, dumber } = store.getState();
```

Action Creators

- **Actions** are objects with a type field
- **Action Creators** are functions returning actions
- use **dispatch** to propagate your actions.

```
const PICTURE_ADD = "PICTURE_ADD";

function addPicture(data) {
  return {
    type: PICTURE_ADD,
    data,
  };
}

const data = { src: 'https://serv.er/picture/123' };
store.dispatch(addPicture(data));
```

Action Creators

- **Actions** are objects with a type field
- **Action Creators** are functions returning actions
- use **dispatch** to propagate your actions.

```
const PICTURE_ADD = "PICTURE_ADD";

function addPicture(data) {
  return {
    type: PICTURE_ADD,
    data,
  };
}

const data = { src: 'https://serv.er/picture/123' };
store.dispatch(addPicture(data));
```

Redux-thunk

- **Redux-thunk** is a middleware for Redux, it allows you to return functions instead of actions in your Action Creators.
- The returned function will be given (dispatch, getState) as arguments.
- Use it to tame asynchrony in your app.

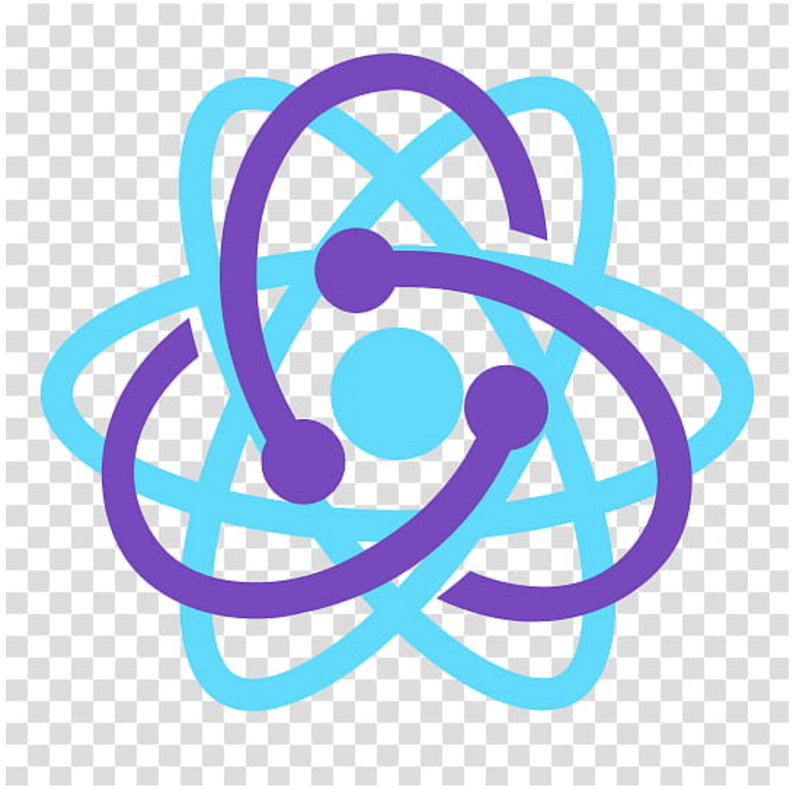
```
const REQUEST_PICTURE = 'REQUEST_PICTURE';
const RECEIVE_PICTURE = 'RECEIVE_PICTURE';

const receivePicture = (data) => ({ type: RECEIVE_PICTURE, data });
const requestPicture = () => ({ type: REQUEST_PICTURE });

funtion fetchPicture(picId) {
  return (dispatch, getState) => {
    dispatch(requestPicture());

    fetch(`https://serv.er/pictures/${picId}`)
      .then(() => dispatch(receivePicture));
  }
}

store.dispatch(fetchPicture(123));
```

Debug

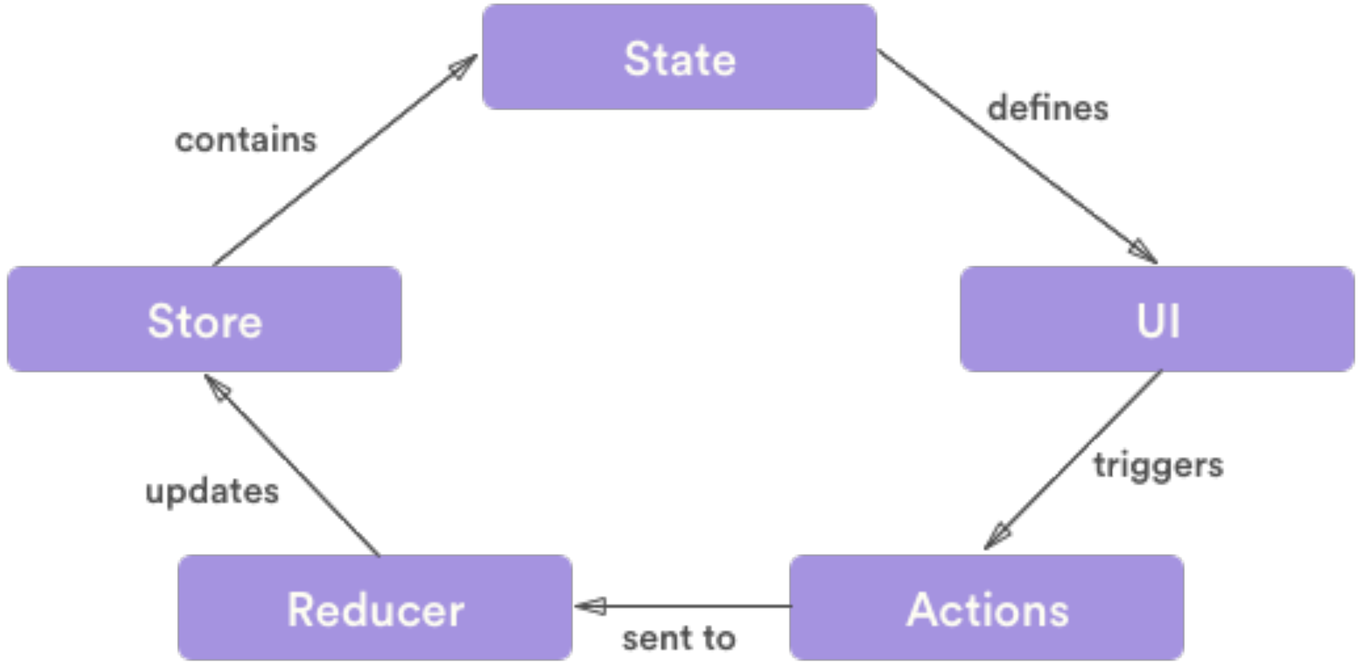
- https://github.com/zalmoxisus/redux-devtools-extension

Redux

React



react-redux



- NPM package react-redux
- Allows you to connect your React app to your Redux store

<Provider />

- **Provider** is a **React Component**
- It provides an access to the **store** to all its children
- Therefore it must be at the top of your tree

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

connect

- **Connect** gives you a **Higher Order Component,** give it your component and get a connected component
- You can use it to cherry pick relevant informations from the App state to pass to your connected component as props
- A common pattern is to have “Controller Components” or (layouts, container…) connected to the state, that pass information to dummy components via props

```
// inject 'dispatch' as a prop
export default connect()(Component)

// inject dispatch and get props from store
function mapStateToProps(state, ownProps) {
  const pic = state.pictures.find(p => p.id === ownProps.id);

  // expose all 'pic' fields as component props
  return { ...pic };
}

export default connect(mapStateToProps)(Picture);

// NEVER do this
export default connect(state => state)(Component);
```

r e a c t - r e d u x

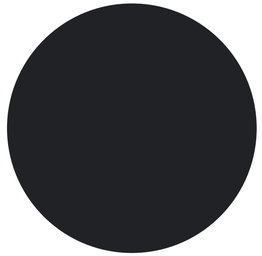
Hooks

- ***useStore()*** -> get a reference on the root state for nearest <Provider />

- ***useDispatch()*** -> get a reference to a dispatch function for the nearest <Provider />

- ***useSelector(fn)*** -> works as useStore, but allows you to get a specific part of the state via fn

Persons of Interest



Jordan Walke

@jordwalke

Creator of React, ReasonML.

Christopher Chedeau

@vjeux

Co-creator of React Native and Prettier.
Creator of “CSS-in-JS".
Former EPITA.

Cheng Lou

@_chenglou

Member of React’s core team at Facebook

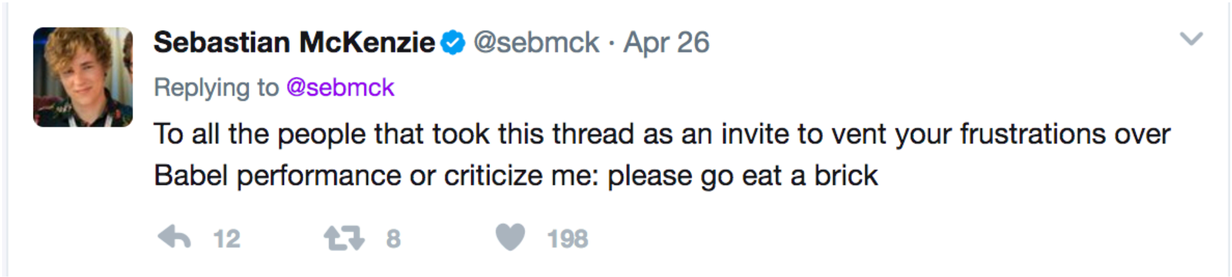
Dan Abramov

@dan_abramov

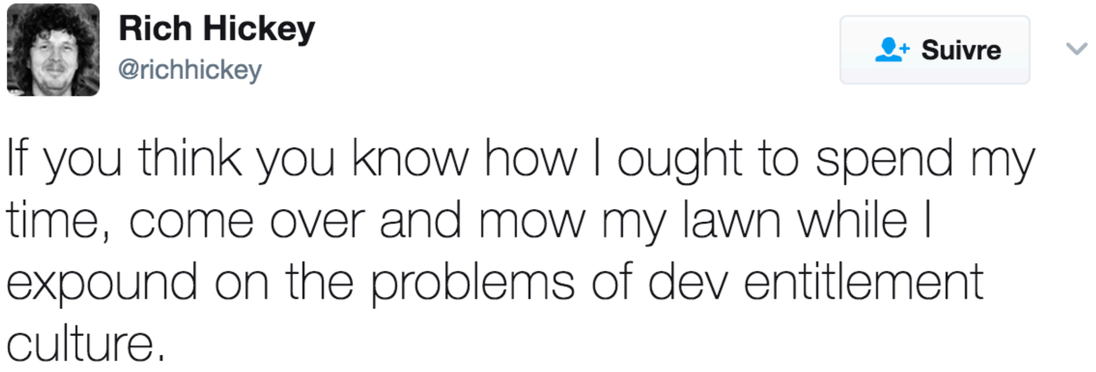
Co-creator of Redux, core-maintainer of React at Facebook

If you have a problem with the tools you use:

From the creator of 6to5 (now BabelJS):



From the creator of Clojure



Stop complaining and get to work.