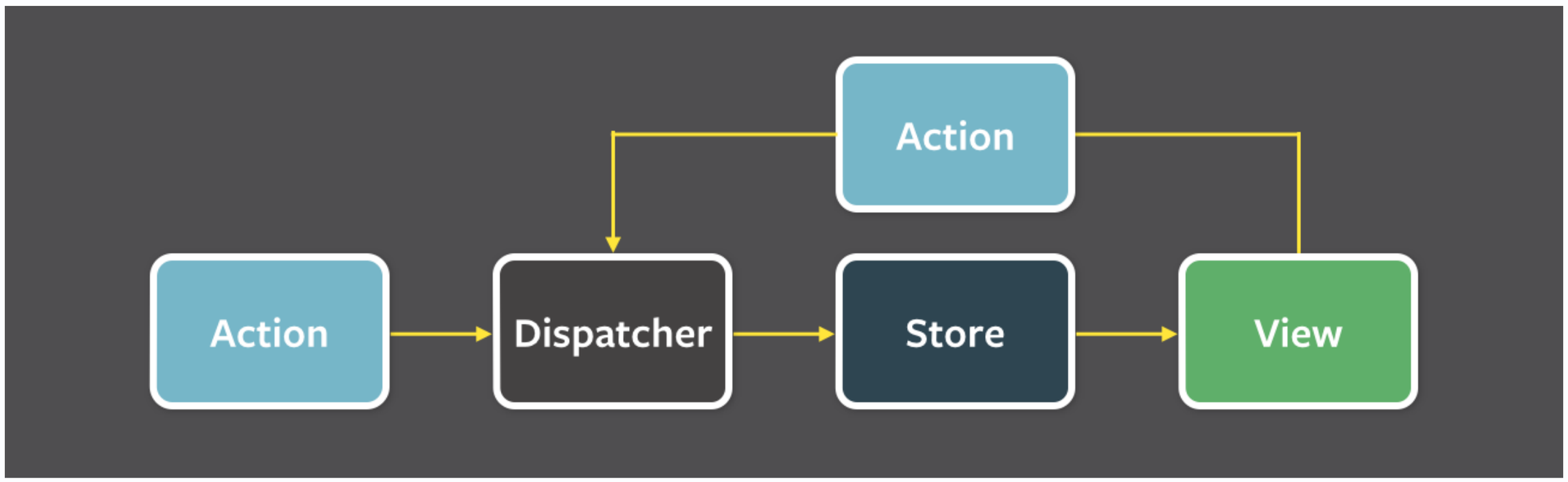
Redux

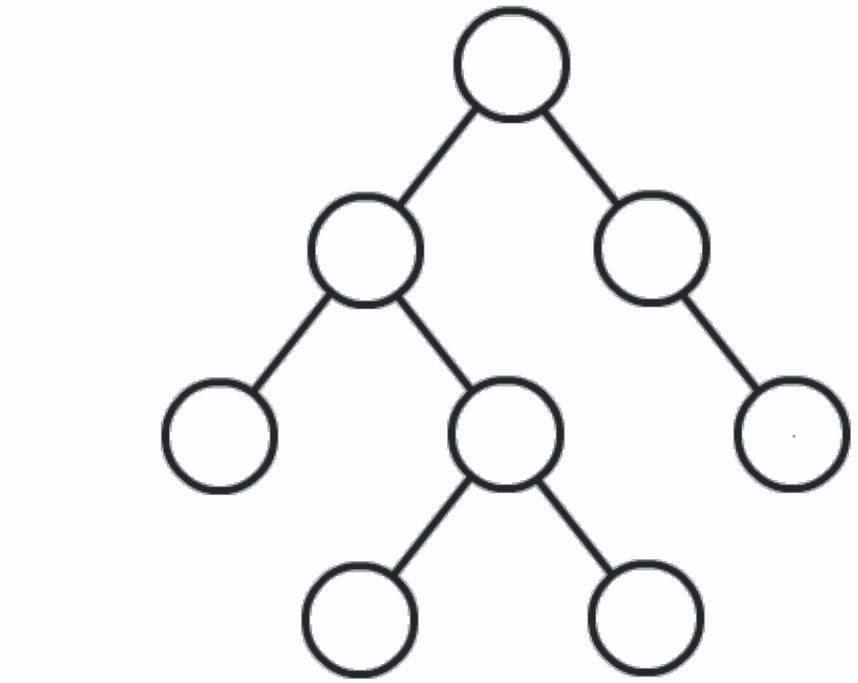
Redux - Introduction

- Redux est un conteneur d'état (state container)
  Une bibliothèque dont le rôle et de stocker l'état de l'application et ainsi de la réaffirmer dans l'état précédent lorsque l'historique est manipulé.

- Inspiré par Flux (Facebook)
  Redux est inspiré de Flux, une architecture proposée par Facebook pour les applications front-end. Différente bibliothèque propose de simplifier leur mise en place, dont Redux, même si quelques concepts sont différents.

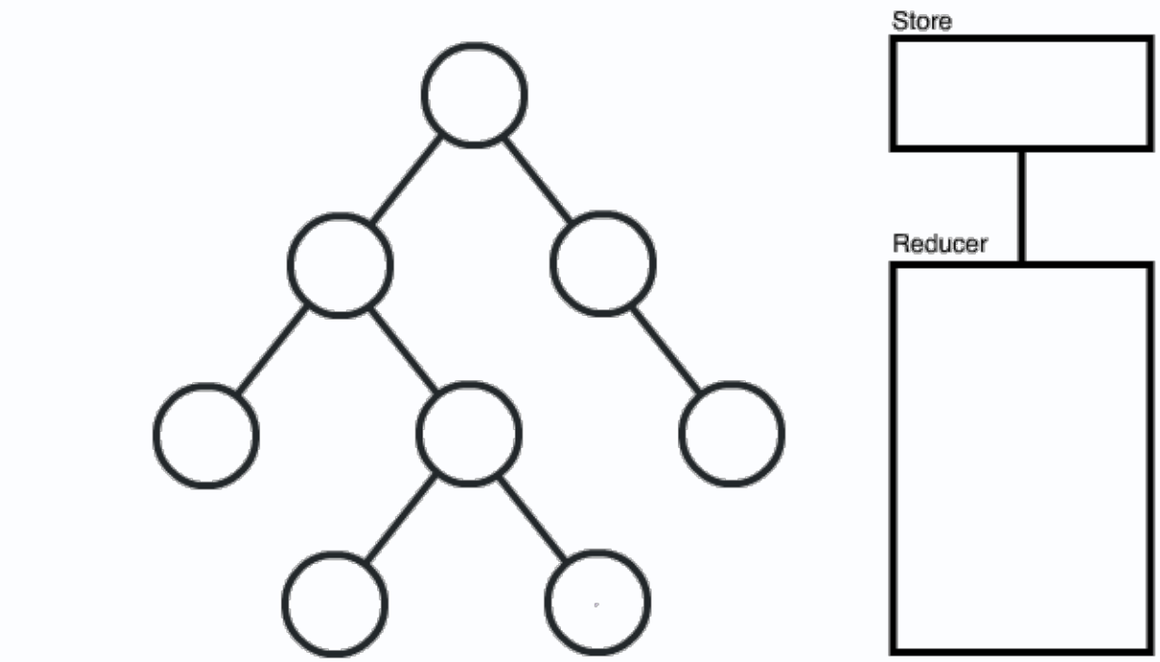


Redux - Introduction



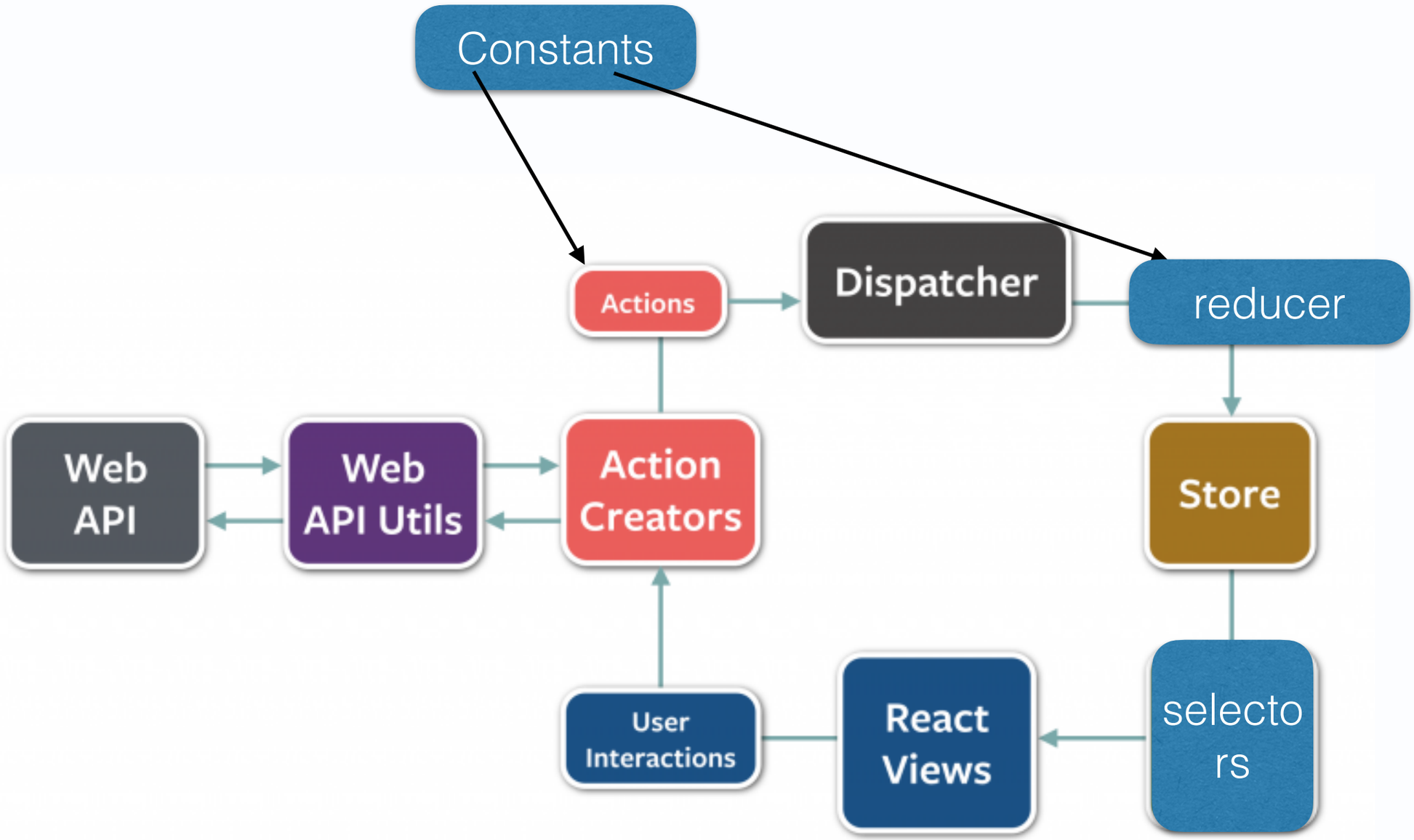
State change initited

State change



State change initited

State change


Constants
Actions
Dispatcher
reducer
Web API
Web API Utils
Action Creators
Store
User Interactions
React Views
selecto rs

Redux - Quand utiliser Redux ?

- Extrait de la doc du Redux : https://redux.js.org/docs/faq/OrganizingState.html#organizing-state-only-redux-state
  - La même donnée est-elle utilisée pour piloter différents composants ?
  - Avez vous besoin de créer de nouvelles données dérivées de ces données ?
  - Y-a-t'il une valeur que vous souhaiteriez restaurer dans un état précédent (time travel debugging, undo/redo) ?
  - Voulez-vous mettre cette donnée en cache ?
- Si oui à l'une de ces question : Redux
- Voir aussi MobX / Recoil
- State React vs State Redux ? https://github.com/reduxjs/redux/issues/1287#issuecomment-175351978

Redux - Installation

- Redux
  - `npm install redux`
  - `yarn add redux`
- Intégration avec react
  - `npm install react-redux`
  - `yarn add react-redux`
- Intégration avec l’extension Redux DevTools
  - `npm install redux-devtools-extension --save-dev`
  - `yarn add redux-devtools-extension --dev`

Redux - Quelques principes

- Immutable State Tree
  - Contrairement à Flux, Redux maintient l'ensemble de l'état de l'application dans un arbre unique.
  - Cet arbre stocké sous la forme d'un plain object JavaScript ou bien tout autre structure (voir Immutable.js).
  - Il doit être immuable, une modification doit entrainement la création d'un nouvel objet en mémoire et non la modification de l'objet existant, ceci pour permettre des fonctionnalités plus avancées comme un undo/redo.
- Pas de modification directe du State Tree
  - Il ne faut pas modifier directement le State Tree, au lieu de ça on va « dispatcher » des actions pour indiquer les modifications à apporter à l'arbre.
  - Une action est un objet JS qui décrit le changement à apporter au State Tree.

Redux - Quelques principes

- Actions
  - Une action doit avoir à minima une propriété nommée type.
    Exemple pour un compteur :

```
const incrementAction = {
  type: 'INCREMENT',
};

const decrementAction = {
  type: 'DECREMENT',
};
```

  - Chaque action doit décrire le minimum du changement à effectuer dans l'application Redux.

```
const addTodoAction = {
  id: 123,
  value: 'Apprendre à utiliser Redux',
  type: 'ADD_TODO',
};
```

  - Chaque changement intervenant dans l'application, clic utilisateur, nouvelle données reçues du serveur, texte saisi… devrait être décrit par une action la plus simple possible.

Redux - Quelques principes

- Actions creators
  - Pour faciliter la création d'actions et les pouvoir les réutiliser plus facilement à différents endroit de l'application, on utilise des fonctions appelées actions creators

```
export const setVisibilityFilter = (filter) => ({
  type: 'SET_VISIBILITY_FILTER',
  filter
});
```

Redux - Quelques principes

- Fonction pure
  - Retourne toujours la même valeur lorsque appelée avec les des paramètres identiques.
  - Aucun effet parallèle comme l'écriture dans un fichier
  - Ne modifie par ses paramètres d'origines (objets, tableaux…)

```
// fonction pure
const addition = function(a, b) {
  return Number(a) + Number(b);
};

// fonctions impures
const getRandomIntInclusive = function(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
};

const validateUser = function(user) {
  localStorage.setItem('user', user);
  return user === 'Romain';
};

const userToUpperCase = function(user) {
  user.prenom = user.prenom.toUpperCase();
  return user;
};
```

Redux - Quelques principes

- Reducers
  - Fonction pure
  - Reçoit l'état précédent et l'action dispatchée
  - Retourne l'état suivant
  - Peut conserver les références vers les objets non-concernés par l'action

```
var counterReducer = function(state, action) {
  if (state === undefined) {
    return 0;
  }

  switch (action.type) {
    case 'INCREMENT':
      return state + 1;
    case 'DECREMENT':
      return state - 1;
  }

  return state;
};
```

Redux - Mise en place

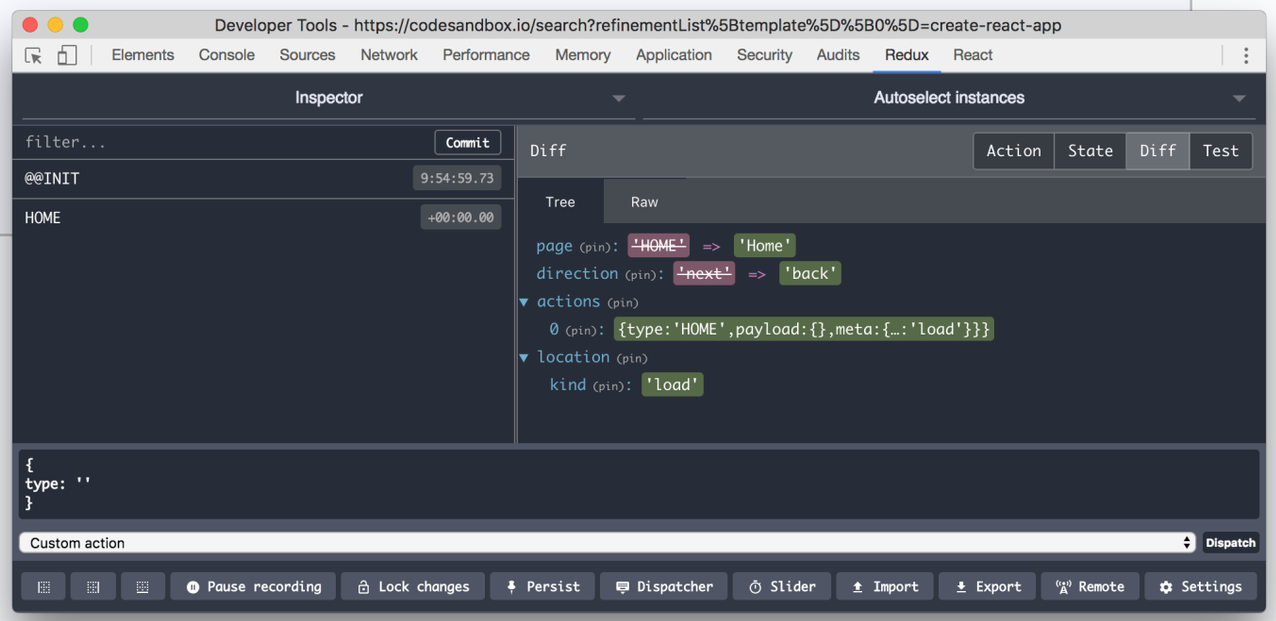
- Store
  - Objet dans lequel est stocké le state (store.getState())
  - Doit être créé à partir d'un reducer ou d'une combinaison de reducers
  - Peut recevoir un state initial, qui pourrait être persisté dans le localStorage par exemple
  - Peut également recevoir des plugins appelés middlewares

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

const counterReducter = (state, action) => {
  // ...
};

const store = createStore(counterReducter);
```

Redux - Mise en place

- Redux DevTools
  - Installer *redux-devtools-extension*
  - Passer *composeWithDevTools* en 2e paramètres de *createStore*

```
import { createStore, combineReducers } from 'redux';
import { composeWithDevTools } from 'redux-devtools-extension';
import { counter } from './reducers/counter';

export const store = createStore(
  counter,
  composeWithDevTools()
);
```



Redux - Mise en place

- React Redux
  - Pour que le store soit disponible dans l'application React, on utilise le composant Provider de react-redux (il doit être à la racine)

```
import React from 'react';
import ReactDOM from 'react-dom';
import App from './components/App';
import { store } from './store';
import { Provider } from 'react-redux';

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

Redux - Mise en place

- La fonction connect de react-redux permet d'associer de rendre disponible la méthode dispatch de redux et d'accéder à certaines propriétés du state

```
import React, { Component } from 'react';
import { connect } from 'react-redux'
import { counterIncrement } from '../actions/counter';

export class ButtonCounter extends Component {
  constructor() {
    super();
    this.handleClick = this.handleClick.bind(this);
  }
  handleClick() {
    this.props.dispatch(counterIncrement());
  }
  render() {
    return <button onClick={this.handleClick}>{this.props.count}</button>;
  }
}

const mapStateToProps = (state) => ({
  count: state.count,
});

export const ButtonCounterContainer = connect(mapStateToProps)(ButtonCounter);
```

Redux - Mise en place

- On peut également associer directement des méthodes avec *mapDispatchToProps*

```
import React from 'react';
import { connect } from 'react-redux'
import { counterIncrement } from '../actions/counter';

export const ButtonCounter = (props) => {
  return <button onClick={props.handleClick}>{props.count}</button>;
};

const mapStateToProps = (state) => ({
  count: state.count,
});

const mapDispatchToProps = (dispatch) => ({
  handleClick: () => dispatch(counterIncrement()),
});

export const ButtonCounterContainer = connect(
  mapStateToProps,
  mapDispatchToProps,
)(ButtonCounter);
```

Redux - Code asynchrone

- Redux Thunk
  https://stackoverflow.com/questions/35411423/how-to-dispatch-a-redux-action-with-a-timeout/35415559#35415559
- Redux Saga
  https://stackoverflow.com/questions/35411423/how-to-dispatch-a-redux-action-with-a-timeout/35415559#38574266