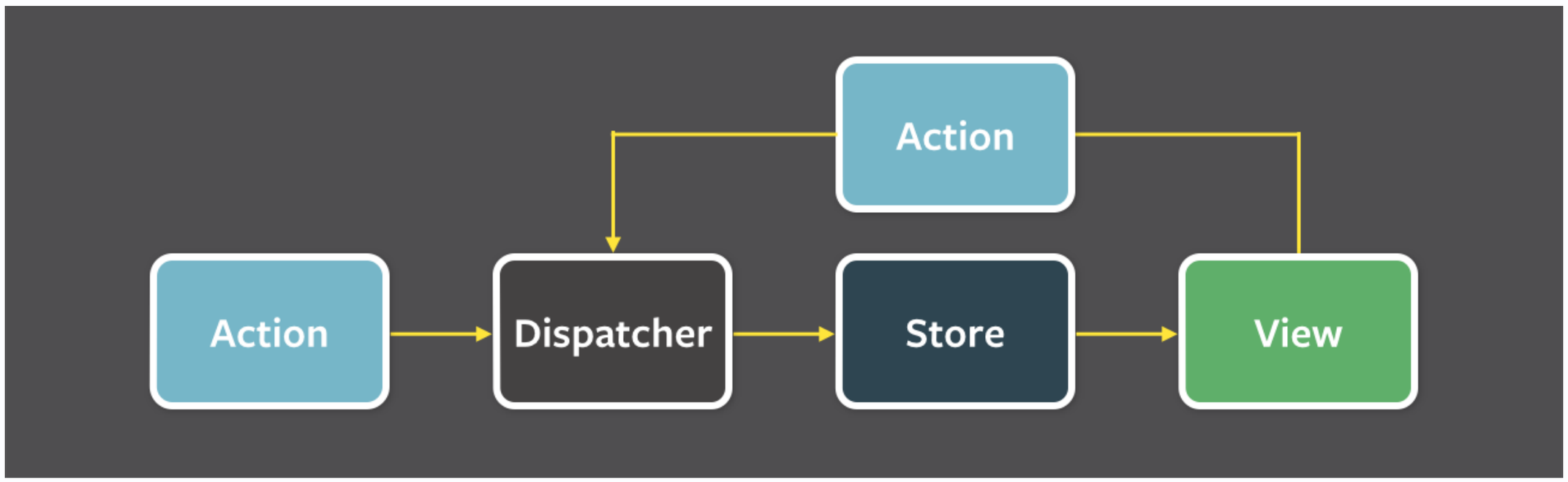
Redux

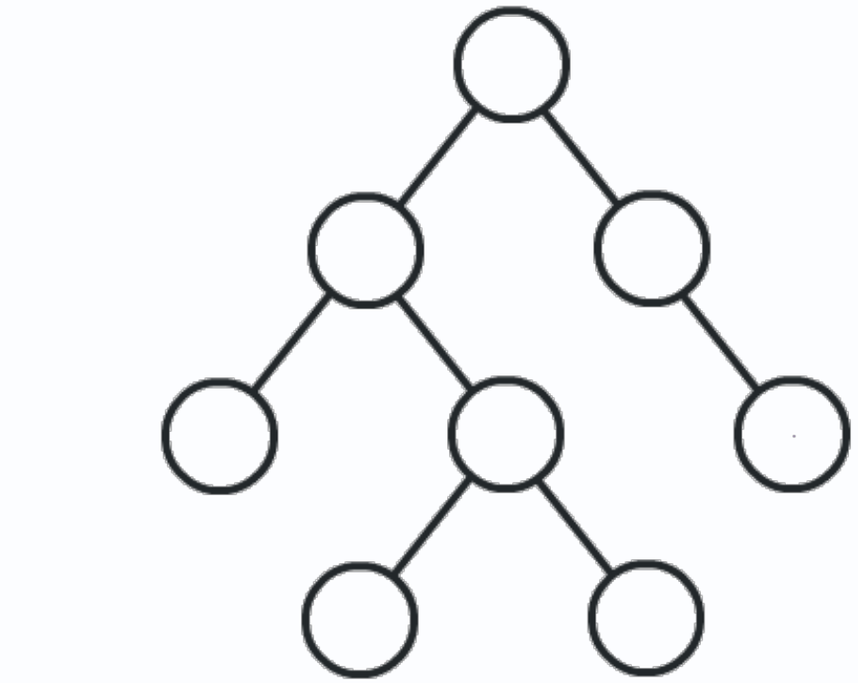
Redux - Introduction

- 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.

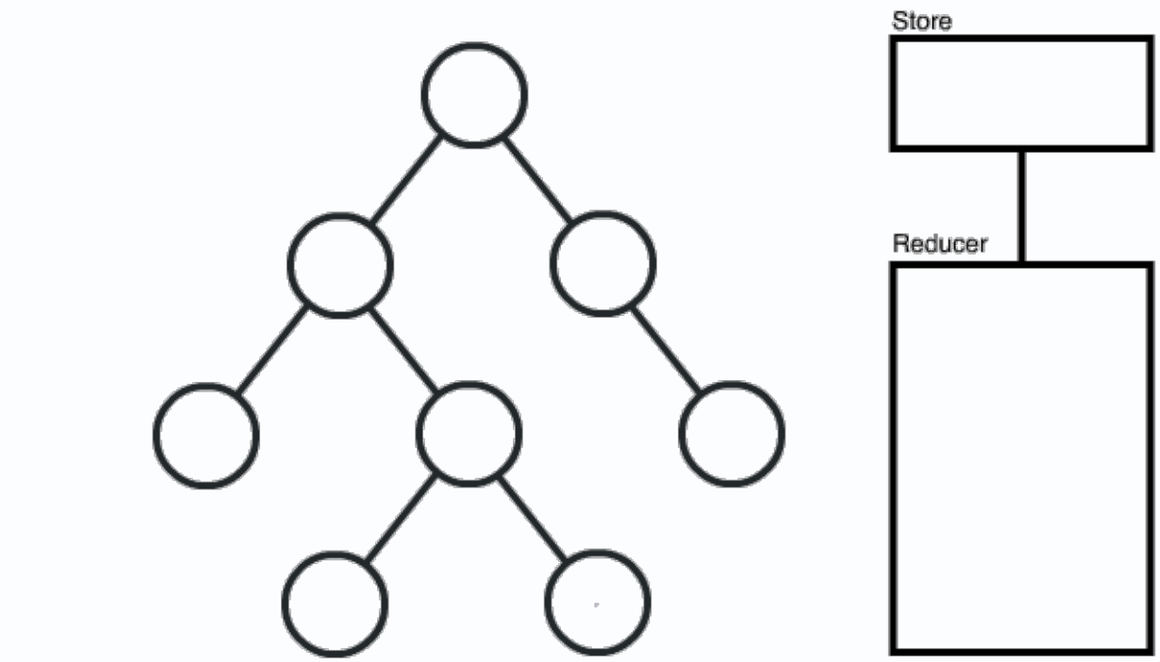


Redux - Introduction



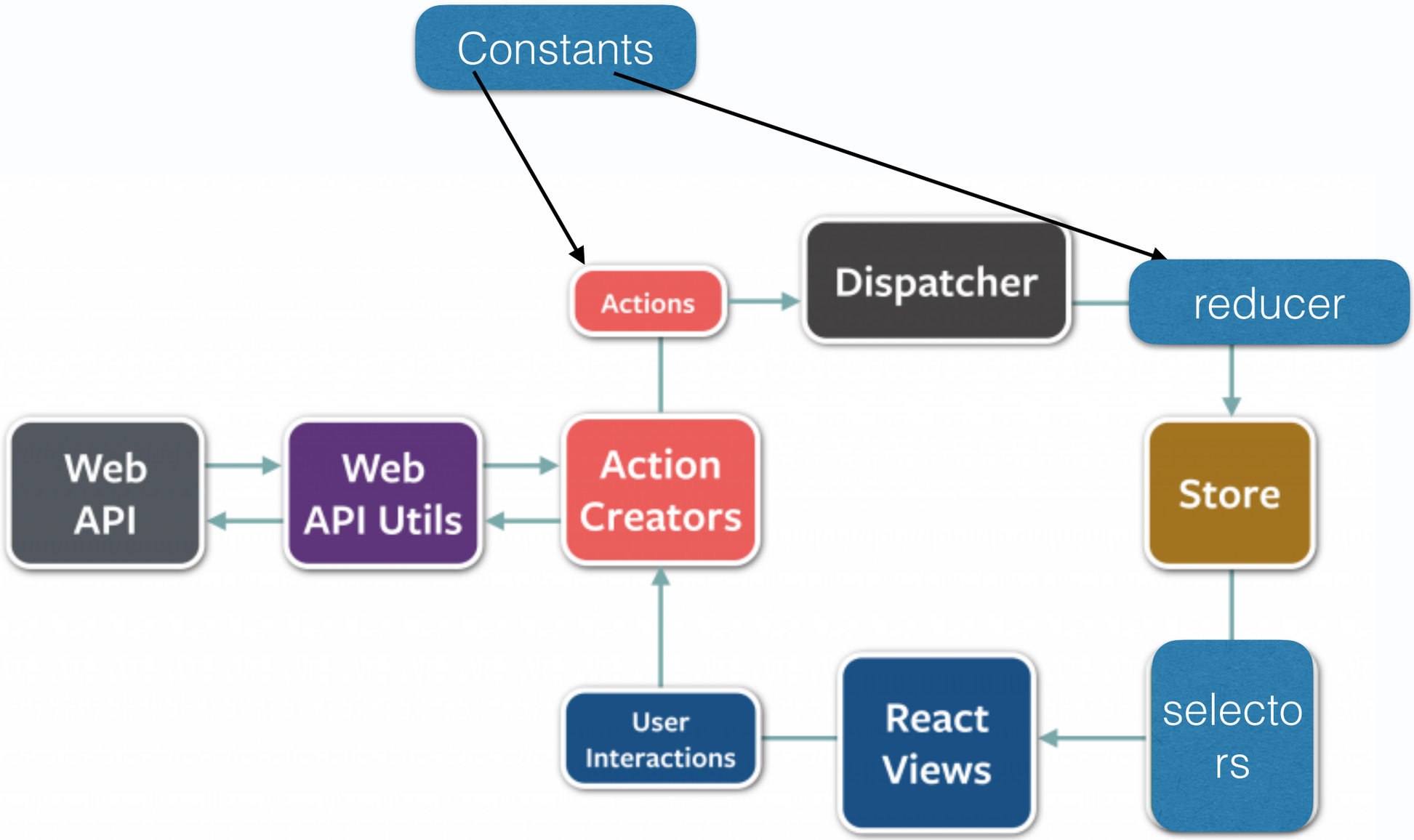
State change initited

State change



State change initited

State change


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

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 :

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

  - 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.

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

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

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

```
// 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

```
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

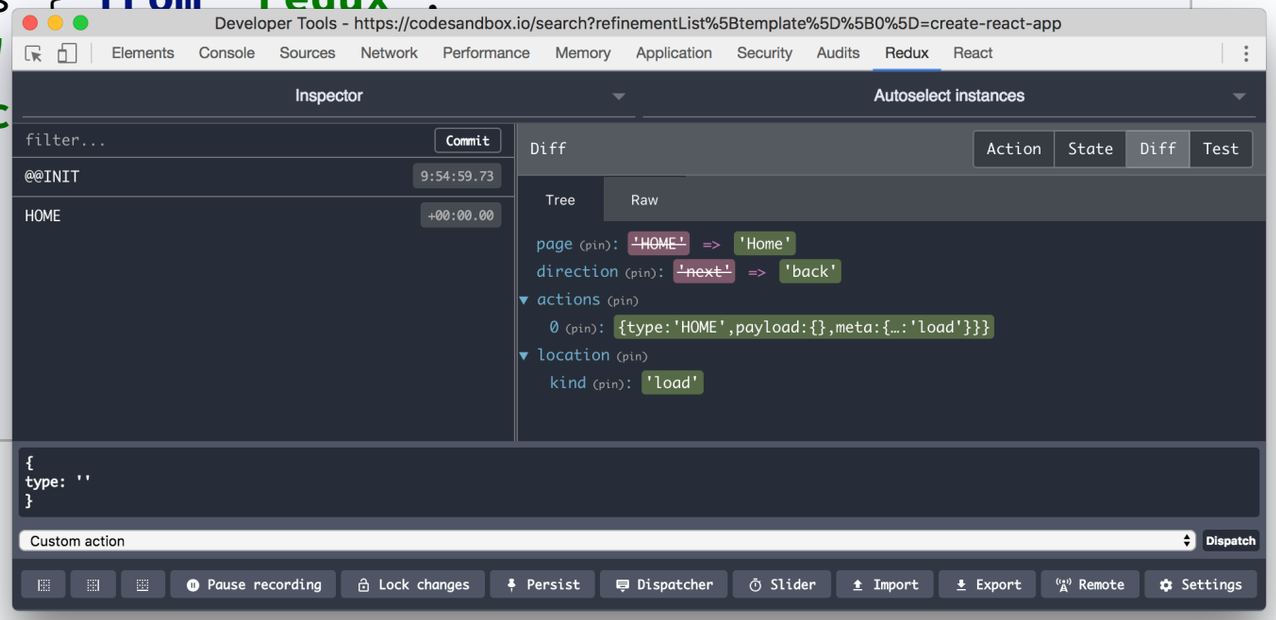
- Store
  - Objet dans lequel est stocké le state (store.getState())
  - Doit être créé à partir d'un reducer ou d'une combinaison de reducers
  - Peut recevoir un state initial, qui pourrait être persisté dans le localStorage par exemple
  - Peut également recevoir des plugins appelées middlewares

```
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 '
import { counter } from './reducers/c

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*

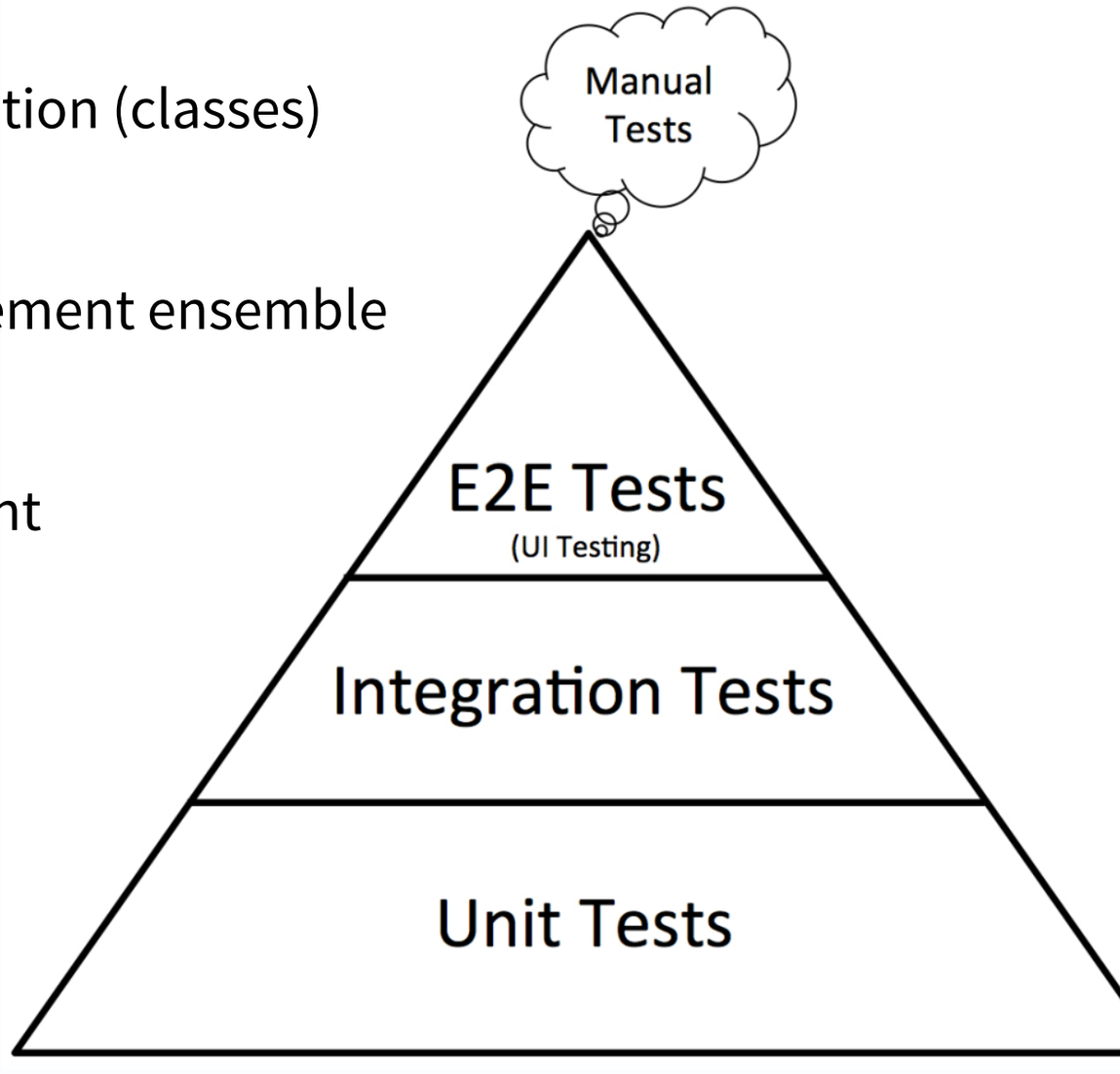
```
import React from 'react';
import { connect } from 'react-redux'
import { counterIncrement } from '../actions/counter';

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

Tests Automatisés - Introduction

- Avec les tests automatisés, les scénarios de test sont codés et peuvent être rejoués régulièrement.
- 4 types de test :
  - Test unitaire
    Permets de tester les briques d'une application (classes)
  - Test d'intégration
    Teste que les briques fonctionnent correctement ensemble
  - Test fonctionnel
    Vérifie l'application du point de vue du client
  - Test End-to-End (E2E)
    Vérifie l'application dans le client



Jest - Introduction

- Framework de test créé en 2014 par Facebook
- Sous Licence MIT depuis septembre 2017
- Permet de lancer des tests :
  - unitaires / d'intégration (dans Node.js)
  - fonctionnels / E2E (via Puppeteer)
- Peut s'utiliser avec ou sans configuration
- Les tests se lancent en parallèle dans les Workers Node.js
- Intègre par défaut :
  - Calcul de coverage (via Istanbul)
  - Mocks (natifs ou en installant Sinon.JS)
  - Snapshots

Jest - Installation

- Installation

```
npm install --save-dev jest
yarn add --dev jest
```

Jest - Hello, world !

- Sans configuration, les tests doivent se trouver dans un répertoire __tests__, ou bien se nommer *.test.js ou *.spec.js

```
// __tests__/hello.js
const hello = require('../src/hello');

test('Hello, world !', () => {
  expect(hello()).toBe('Hello World !');
  expect(hello('Romain')).toBe('Hello Romain !');
});
```

```
module.exports = hello;
```

Jest - Lancements des tests

- Si Jest localement
  `node_modules/.bin/jest`
- Si Jest globalement
  `jest`
- Avec un script test dans package.json
  `npm run test`
  `npm test`
  `npm t`

```
// package.json
{
  "devDependencies": {
    "jest": "^22.0.6"
  },
  "scripts": {
    "test": "jest"
  }
}
```

```
MacBook-Pro:hello-jest romain$ node_modules/.bin/jest
 PASS  __tests__/hello.js
  ✓ Hello, world ! (3ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.701s, estimated 1s
Ran all test suites.
```

Jest - Watchers

- En mode Watch

```
node_modules/.bin/jest --watchAll
jest --watchAll
npm t -- --watchAll
```

```
MacBook-Pro:hello-jest romain$ npm t -- --watchAll
 PASS  __tests__/hello.js
 PASS  __tests__/calc.js

Test Suites: 2 passed, 2 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.65s, estimated 1s
Ran all test suites.

Watch Usage
 › Press f to run only failed tests.
 › Press o to only run tests related to changed files.
 › Press p to filter by a filename regex pattern.
 › Press t to filter by a test name regex pattern.
 › Press q to quit watch mode.
 › Press Enter to trigger a test run.
```

Jest - Coverage

- Avec calcul du coverage

```
node_modules/.bin/jest --coverage
jest --coverage
npm t -- --coverage
```

- Par défaut le coverage s'affiche dans la console et génère des fichiers Clover, JSON et HTML dans le dossier coverage

```
MacBook-Pro:hello-jest romain$ npm t -- --coverage
 PASS  __tests__/calc.js
 PASS  __tests__/hello.js

Test Suites: 2 passed, 2 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        0.722s, estimated 1s
Ran all test suites.
----------|----------|----------|----------|----------|----------------|
File      |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
----------|----------|----------|----------|----------|----------------|
All files |    86.67 |      100 |       60 |      100 |                |
 calc.js  |    83.33 |      100 |       50 |      100 |                |
 hello.js |      100 |      100 |      100 |      100 |                |
----------|----------|----------|----------|----------|----------------|
```

Jest - Mocks

- Jest intègre par défaut une bibliothèque de Mocks

```
// __tests__/Array.prototype.forEach.js
const names = ['Romain', 'Edouard'];

test('Array forEach method', () => {
  const mockCallback = jest.fn();
  names.forEach(mockCallback);
  expect(mockCallback.mock.calls.length).toBe(2);
  expect(mockCallback).toHaveBeenCalledTimes(2);
});
```

Jest - Tester les timers

- La fonction jest.useFakeTimers() transforme les timers (setTimeout, setInterval...) en mock

```
// src/timeout.js
const timeout = (delay, arg) => {
  return new Promise((resolve) => {
    setTimeout(resolve, delay, arg);
  });
};

module.exports = timeout;
```

```
// __tests__/timeout.js
jest.useFakeTimers();

const timeout = require('../src/timeout');

test('waits 1 second', () => {
  const arg = timeout(10000, 'Hello');

  expect(setTimeout).toHaveBeenCalledTimes(1);
  expect(setTimeout).toHaveBeenLastCalledWith(expect.any(Function), 10000,
'Hello');
});
```

Jest - React

- Une application créé avec create-react-app est déjà configurée pour fonctionner avec React
- Sinon il faudrait installer des dépendances comme babel, babel-jest... https://facebook.github.io/jest/docs/en/tutorial-react.html

```
// src/App.js
import React, { Component } from 'react';
import { Hello } from './Hello';
import { CounterButton } from './CounterButton';


class App extends Component {
  render() {
    return (
      <div>
        <Hello firstName="Romain" />
        <hr />
        <CounterButton/>
      </div>
    );
  }
}

export default App;
```

Jest - React

- Pour tester un composant React il faut en faire le rendu

- 2 inconvénients ici :
  - Nécessite que document existe (exécuter les tests dans un navigateur ou utiliser des implémentations de document côté Node.js comme JSDOM)
  - Les composants enfants du composant testé seront également rendu, le test n'est pas un test unitaire mais un test d'intégration

```
// src/App.test.js
import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';

it('renders without crashing', () => {
  const div = document.createElement('div');
  ReactDOM.render(<App />, div);
});
```

Jest - Snapshot Testing

- Facebook fourni un paquet npm pour simplifier les tests : react-test-renderer
- Ici on fait un simple Snapshot, c'est à dire une capture du rendu du composant, si lors d'un test futur le rendu est modifié le test échoue

```
import React from 'react';
import renderer from 'react-test-renderer';
import { Hello } from './Hello';

test('it renders like last time', () => {
  const tree = renderer
    .create(<Hello />)
    .toJSON();
  expect(tree).toMatchSnapshot();
});
```

Jest - Shallow Rendering

- On peut également faire appel à ShallowRenderer qui ne va faire qu'un seul niveau de rendu, et donc rendre le test unitaire

```
// src/App.test.js
import React from 'react';
import App from './App';
import ShallowRenderer from 'react-test-renderer/shallow';
import { Hello } from './Hello';
import { CounterButton } from './CounterButton';

it('renders without crashing', () => {
  const renderer = new ShallowRenderer();
  renderer.render(<App />);
  const result = renderer.getRenderOutput();

  expect(result.type).toBe('div');
  expect(result.props.children).toEqual([
    <Hello firstName="Romain" />,
    <hr />,
    <CounterButton/>,
  ]);
});
```

Jest - Enzyme

- Facebook recommande également l'utilisation de la bibliothèque Enzyme, créée par AirBnB.
- Elle fourni un API haut niveau (proche de jQuery) pour manipuler les tests des composant

```
import React from 'react';
import { Hello } from './Hello';
import { shallow } from 'enzyme';

test('it renders without crashing with enzyme', () => {
  shallow(<Hello />);
});

test('it renders without crashing with enzyme', () => {
  const wrapper = shallow(<Hello />);
  expect(wrapper.contains(<div>Hello  !</div>)).toEqual(true);
});

test('it renders without crashing with enzyme', () => {
  const wrapper = shallow(<Hello firstName="Romain"/>);
  expect(wrapper.contains(<div>Hello Romain !</div>)).toEqual(true);
});
```

Jest - Tester des événements

```
import React, { Component } from 'react';

export class CounterButton extends Component {
  constructor() {
    super();
    this.state = {
      count: 0,
    };
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState({
      count: this.state.count + 1,
    });
  }

  render() {
    return (
      <button onClick={this.handleClick}>{this.state.count}</button>
    );
  }
}
```

Jest - Tester des événements

```
import React from 'react';
import { CounterButton } from './CounterButton';
import { shallow } from 'enzyme';

test('it renders without crashing', () => {
  shallow(<CounterButton />);
});

test('it contains 0 at first rendering', () => {
  const wrapper = shallow(<CounterButton />);
  expect(wrapper.text()).toBe('0');
});

test('it contains 1 after click', () => {
  const wrapper = shallow(<CounterButton />);
  wrapper.simulate('click');
  expect(wrapper.text()).toBe('1');
});
```