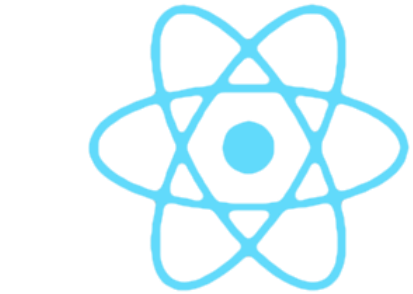
Introduction to React.js (and Redux!)

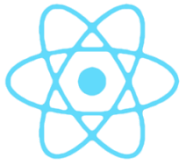


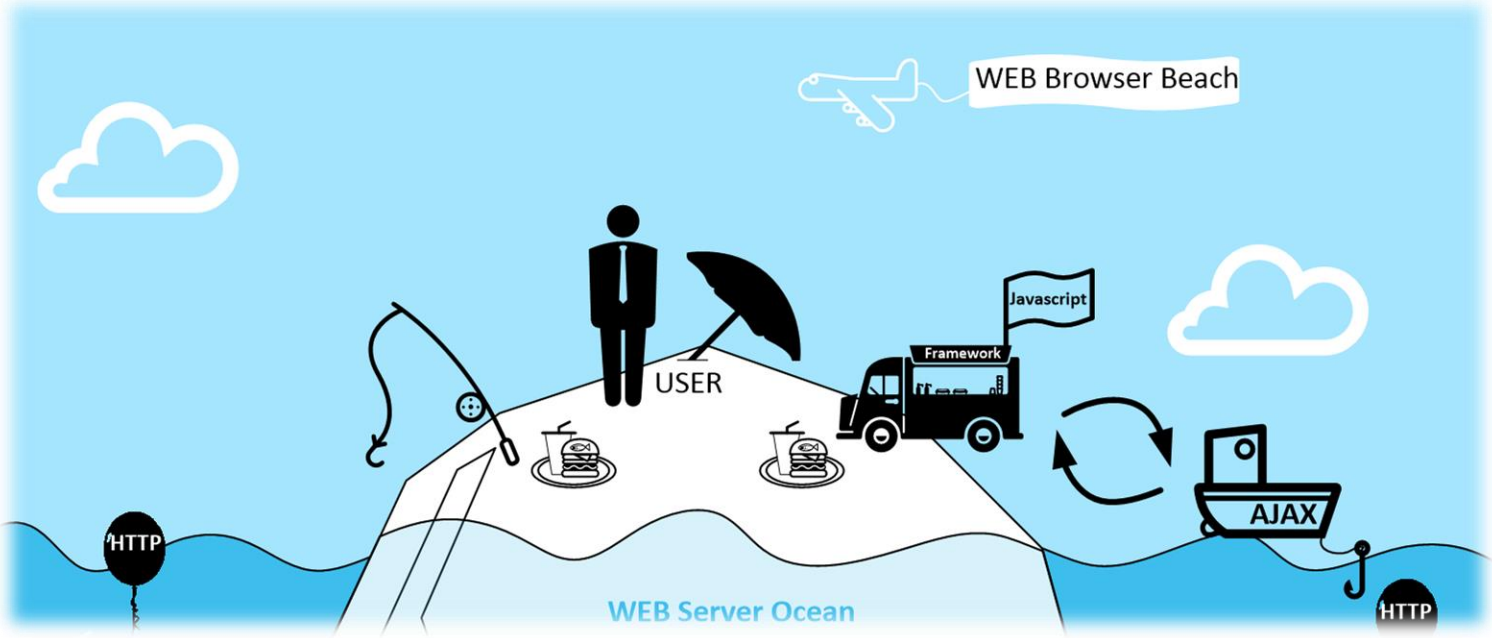
CPE LYON

What is a Front-End Framework ?

Server side pages generation

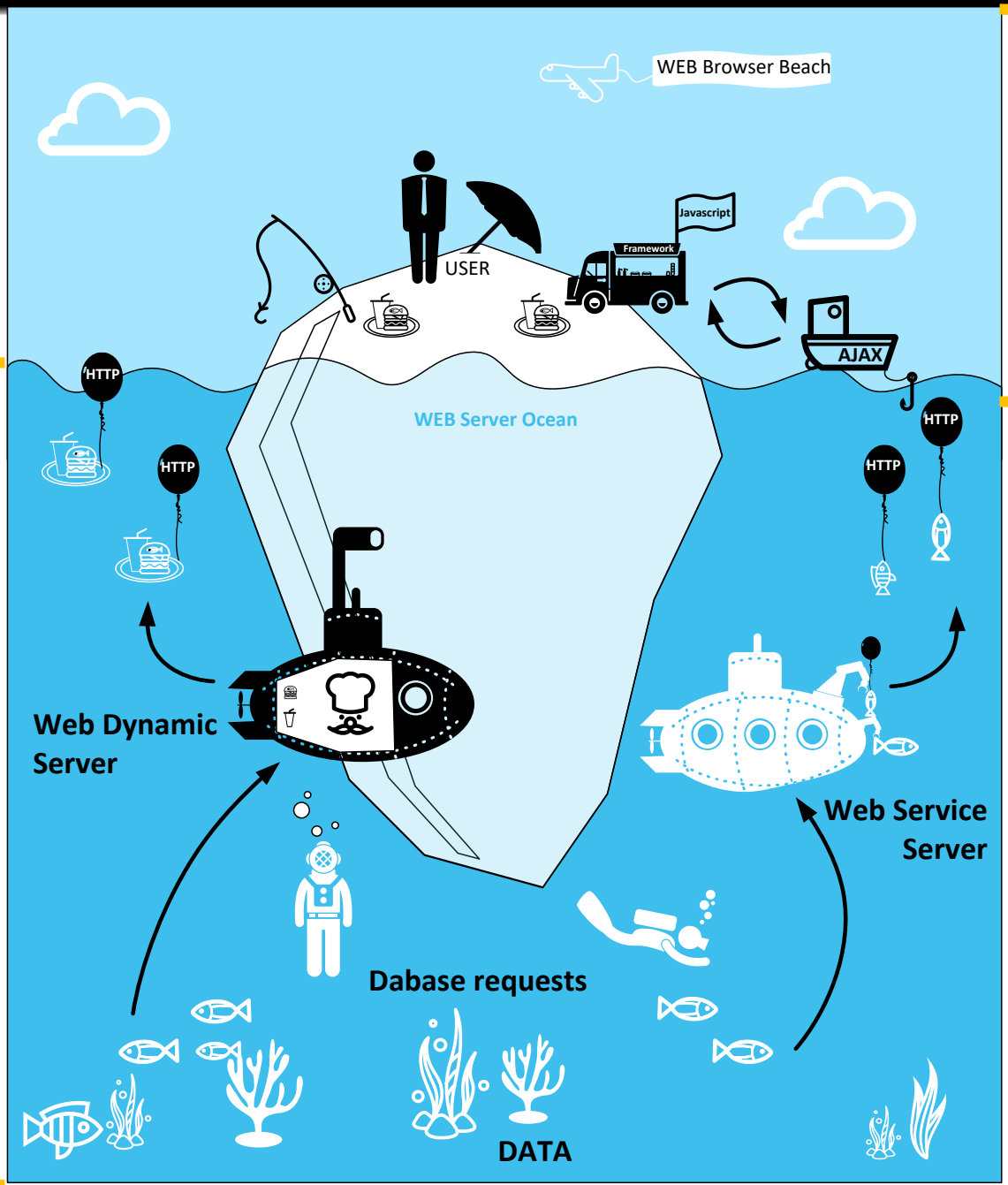
Web Browser side pages generation





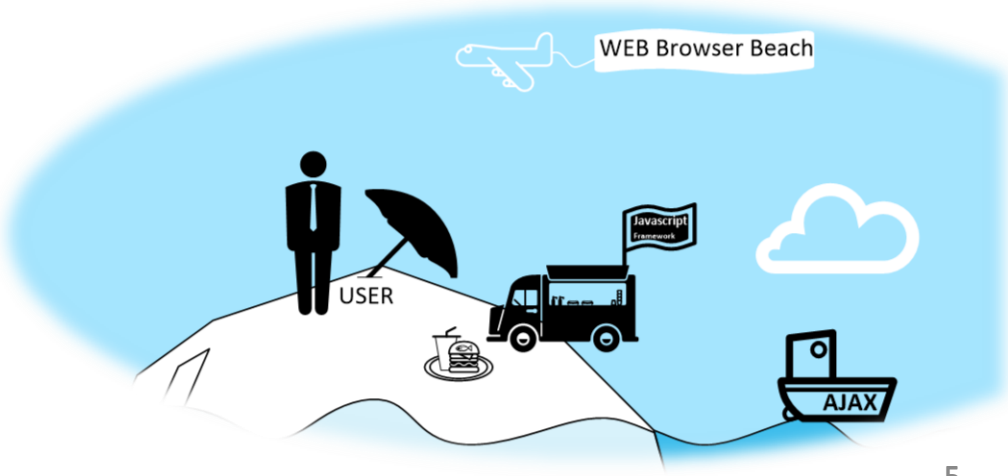
FRONT END

BACK END



Front End

- Everything running on the web browser !!
- Basic languages
  - HTML
  - CSS
  - JAVASCRIPT
- Lots of toolboxes !!
  - Jquery
  - AJAX
  - Canvas
  - WebGL
  - …



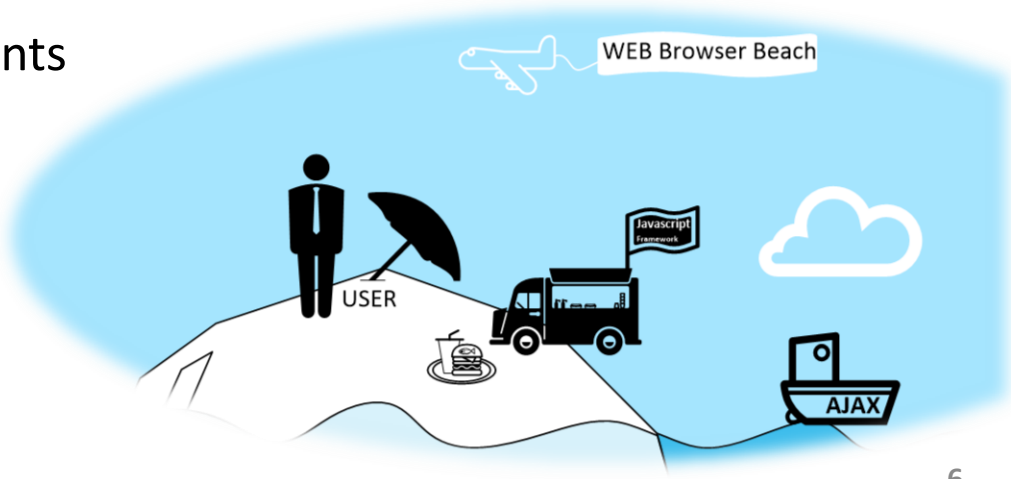
Why do we need additional tools ?

- Everything can be done through pure JAVASCRIPT !

  → right but hard!

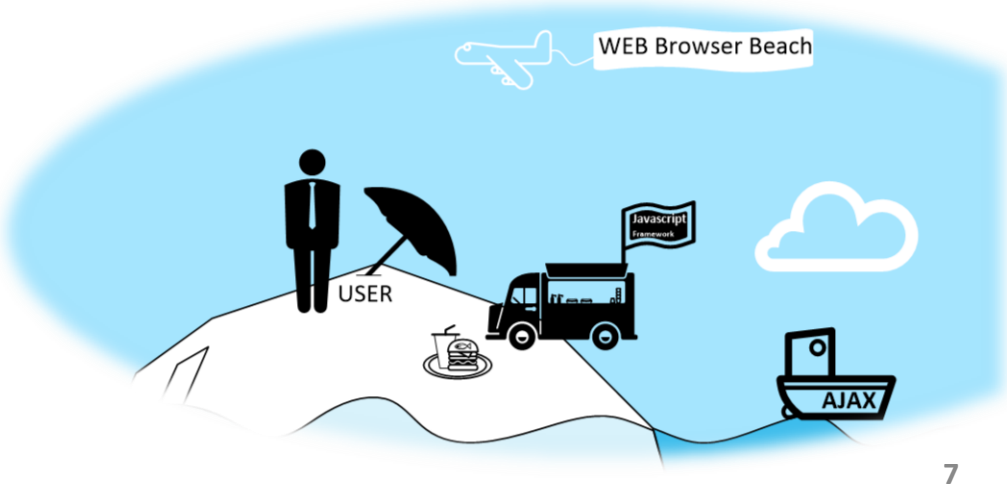
- Front End Framework
  - Help to organize front end development
  - Provide lots of predefined components
  - Allow the creation of components
  - Help to gain time !!

(depending on your front end framework knowledge !)

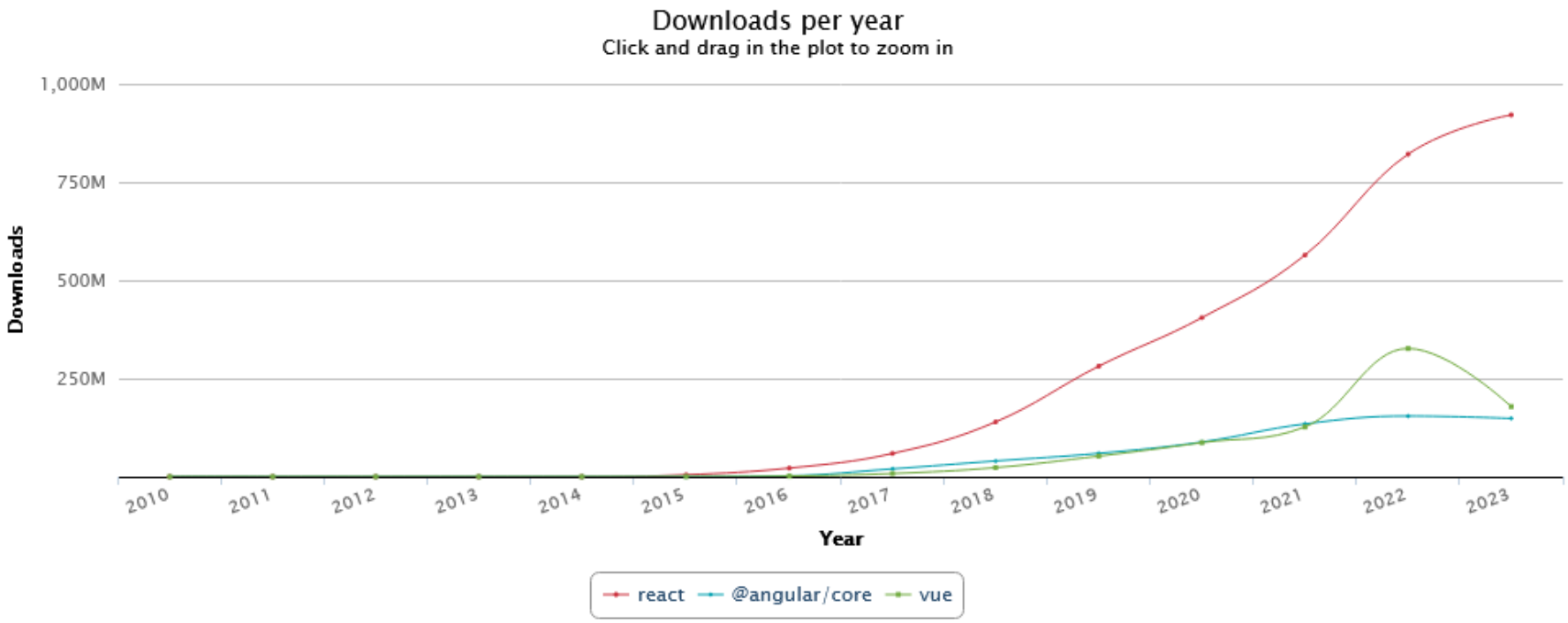


What is the best Frond End Framework ?

- It depends of what you want!
  - Time to learn
  - Front End efficiency
  - Modularity
  - Component creation complexity
  - Community Size
  - Maturity

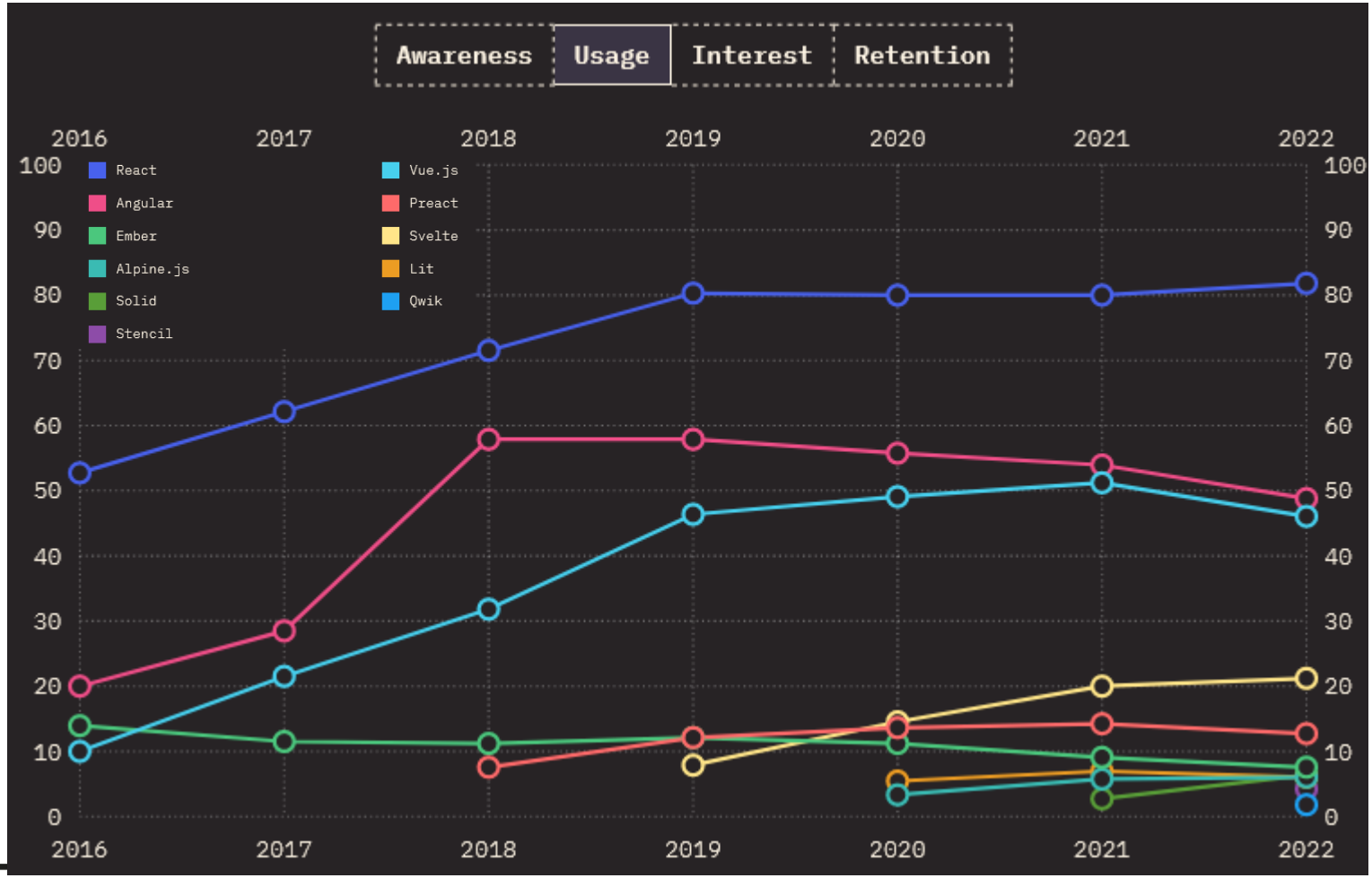


Frond-End Solutions Comparison



https://npm-stat.com/charts.html?package=react&package=vue&package=%40angular%2Fcore&from=2010-12-12&to=2023-11-20

Frond-End Solutions Comparison



https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/

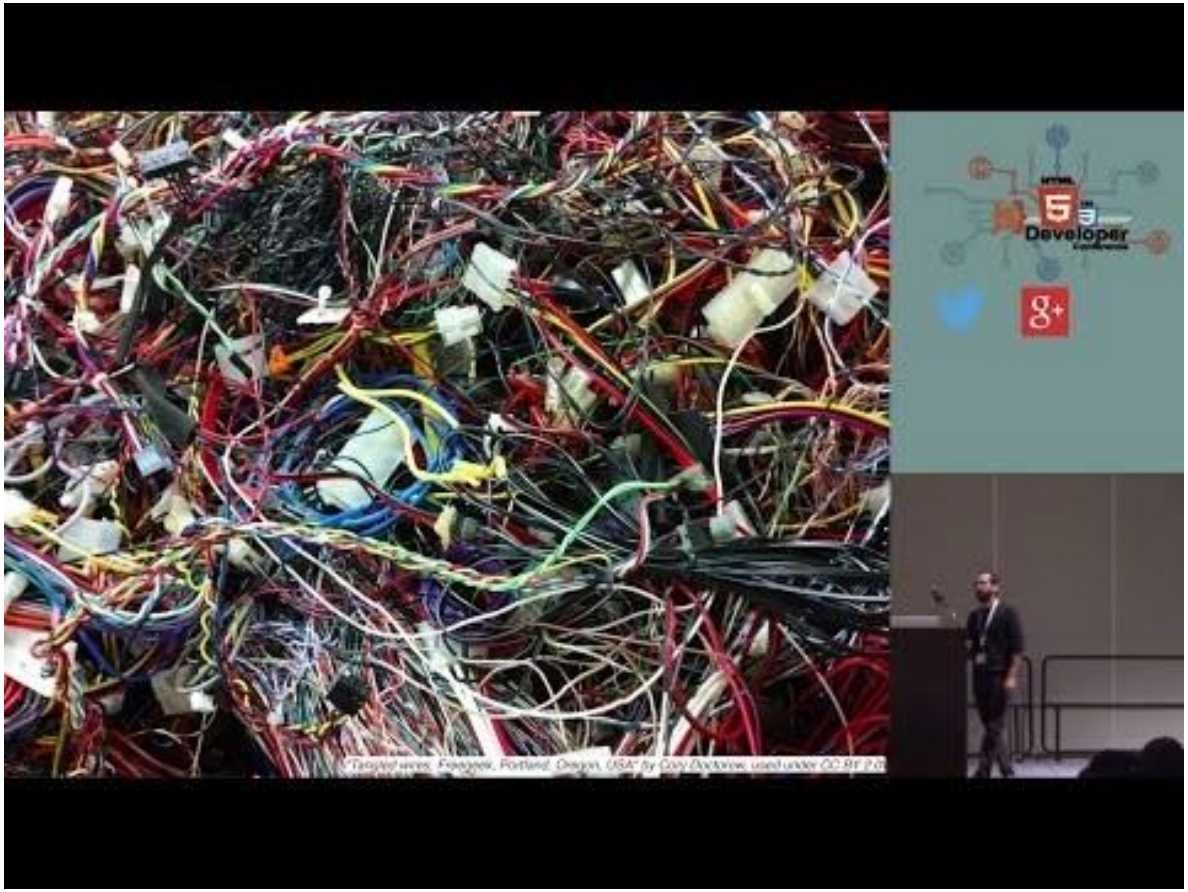
Frond-End Solutions Comparison

Differentiator	React	Angular	Vue
Founded By	Facebook	Google	Evan You
Type	JavaScript library	MVC framework	JavaScript framework
Language	JavaScript XML	TypeScript	JavaScript
First Released In	May 2013	September 2016	February 2014
Version	March 22, 2021	20, 2020	1, 2021
Min bundle size	80KB	500KB	80KB
Used For	update many views at a time	Best for SPAs that update a single view at a time	Best for UIs, and SPAs.
Popular Use-cases	Airbnb, Instagram, Netflix, Pinterest, Slack, Udemy, WhatsApp	Forbes, General Motors, Nike, Paypal, Telegram, Upwork	GitHub, Alibaba, 9gag, Xiomi
Learning Curve	Subtle	Steep	Subtle
Data Binding	One-way Data Binding	Two-way Data Binding	Two-way Data Binding
Benefits	Great UX, better testing capabilities, quicker development,	Cleaner code, easier error handling, high performance, seamless	User-friendly, easy to learn, highly scalable, versatile, well-
GitHub Stars	168k	59.6k	183k
Community/Support	StackOverflow, DEV's, Hashnode's, Reactiflux chat, Reddit, Spectrum	Dev.to, Gitter, freeCodeCamp, YouTube, Reddit, StackOverflow, Angular Docs, GitHub	StackOverflow, Vue.js Official community, GitHub

https://www.spaceotechnologies.com/front-end-frameworks/

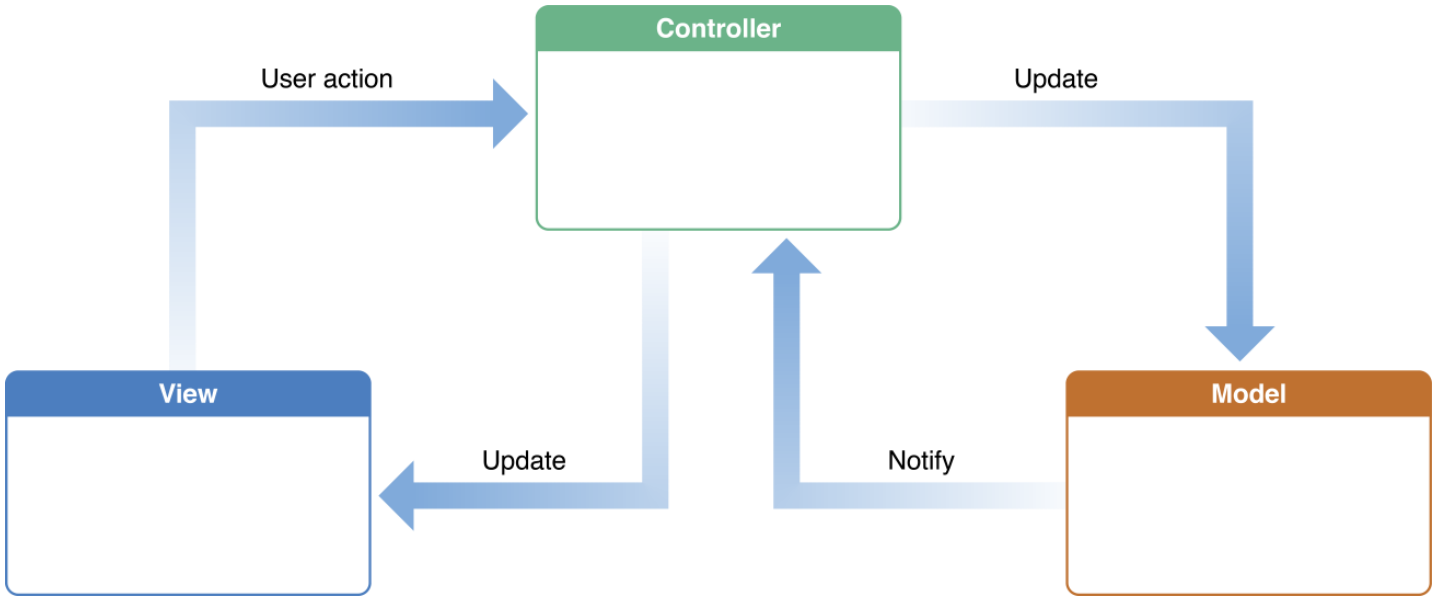
Flux and react.js Concepts

Why current approaches are not sufficient ?



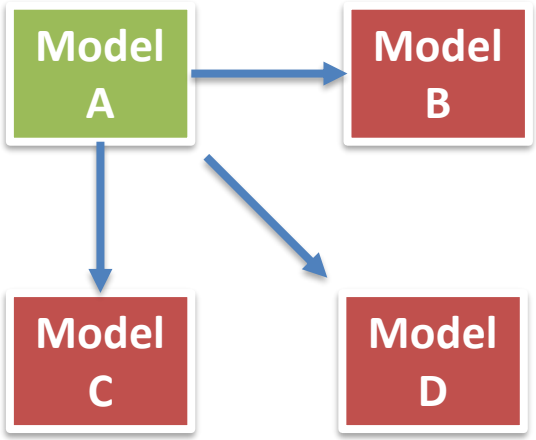
https://www.youtube.com/watch?v=Bic_sFiaNDI

Current State: MVC



https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html

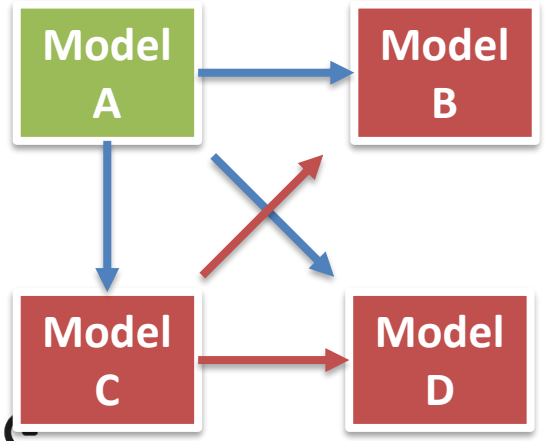
Current State: MVC



- Model B needs (depends on) model A

  → **Model A needs to be updated first and then model B**

- Idem if Model C and Model D needs model A



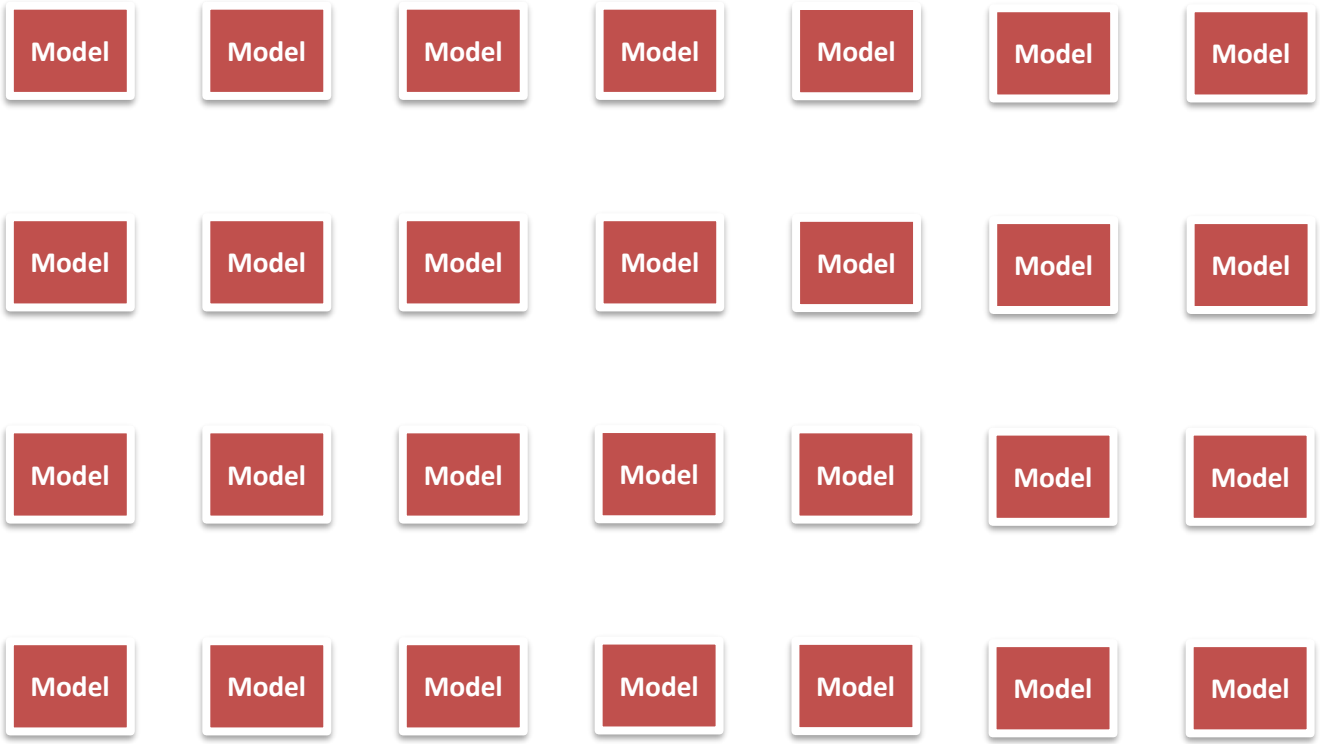
- Model B may needs also on model C

  → **Model A needs to be updated first, then model C and finally Model B**

- **Update needs to be propagated and we need to manage that**

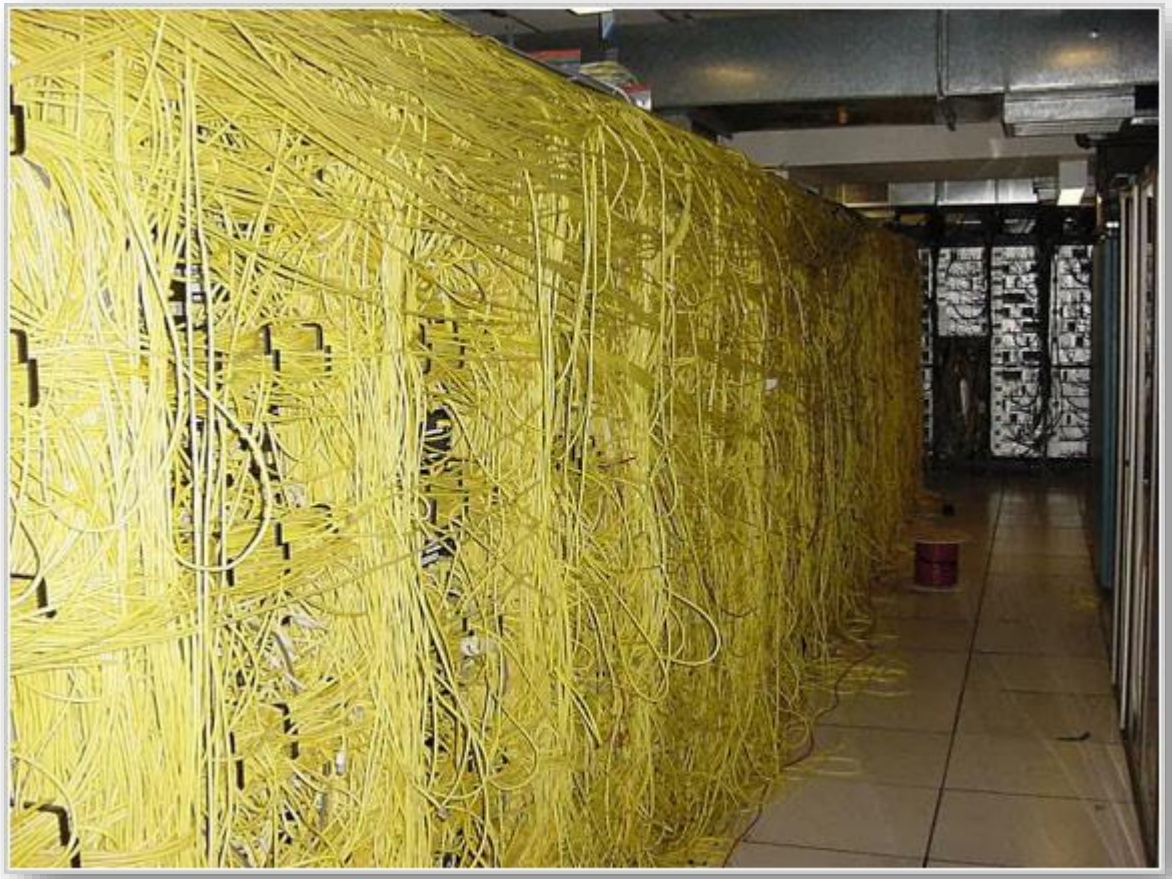
Current State: MVC

What happen when we scale up ?

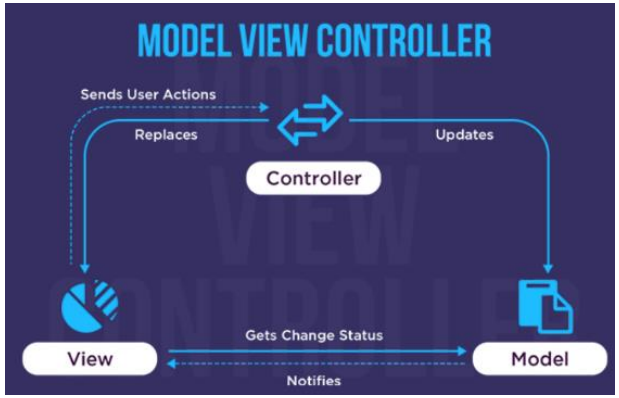


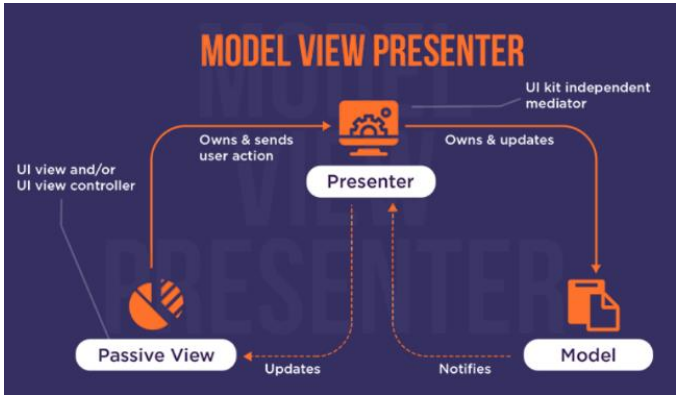
Current State: MVC

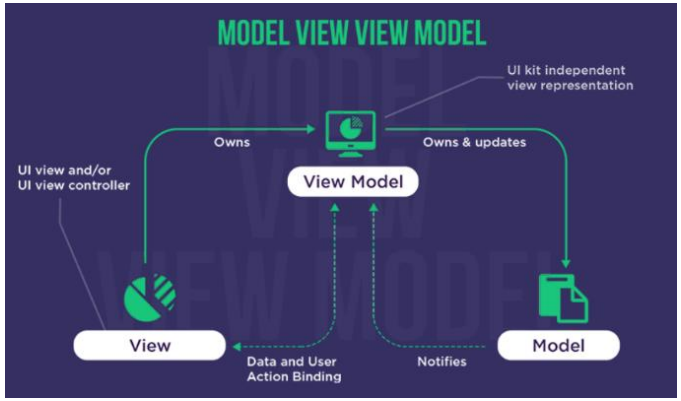
What happen when we scale up ?



Current State: Other Architectures



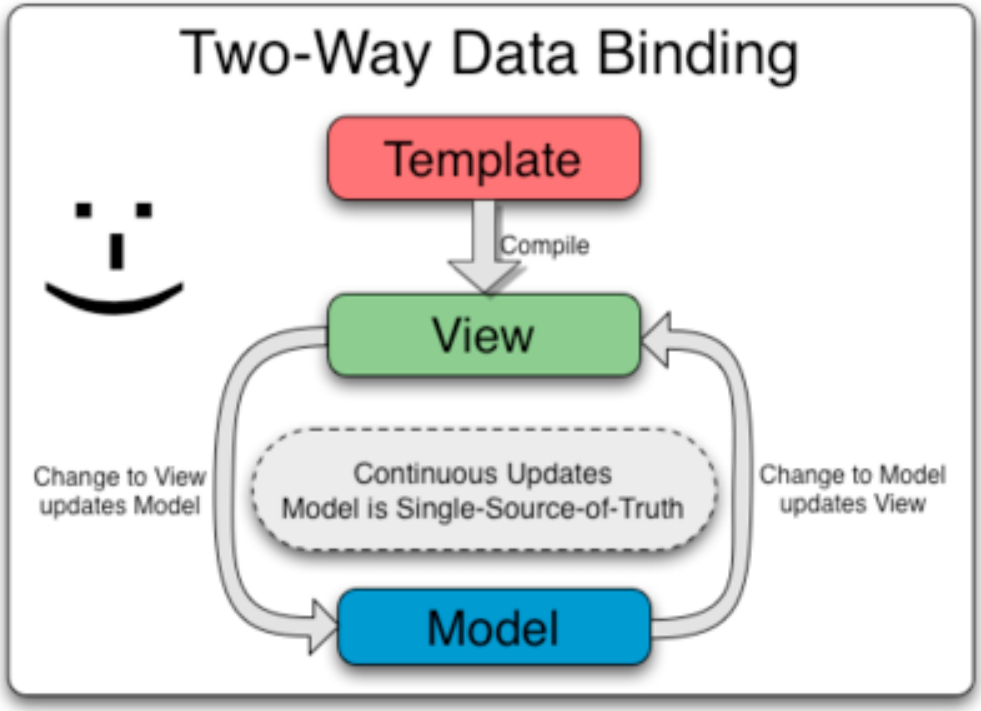




https://www.thirdrocktechkno.com/blog/architecture-presentation-patterns-mvc-vs-mvp-vs-mvvm/

Current State: MVVM (Model View ViewModel)

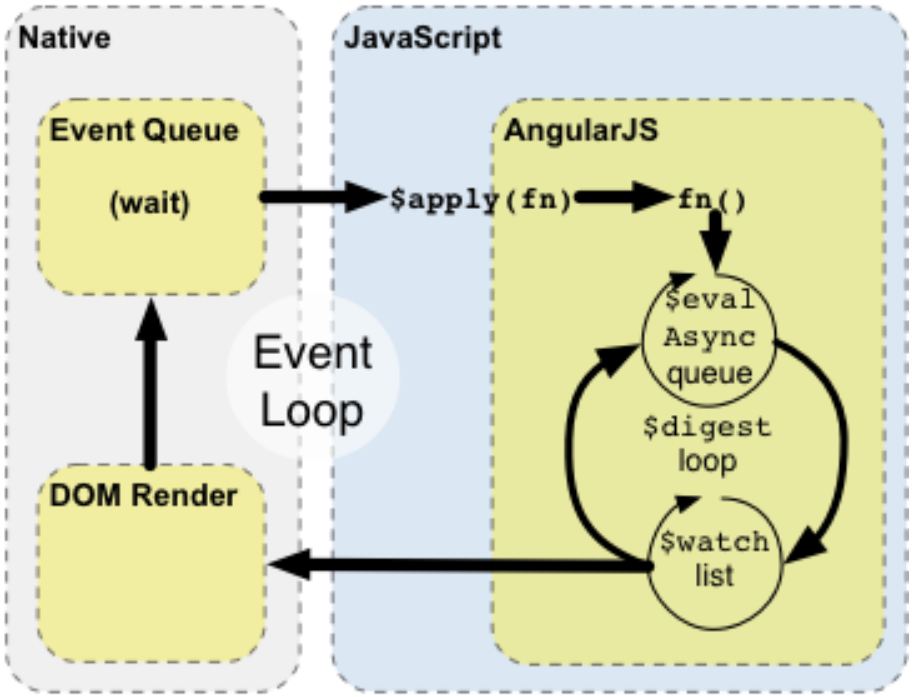
- Angular.js data binding



- No more controller for managing model update events instead

Current State: MVVM (Model View ViewModel)

- Angular.js data binding

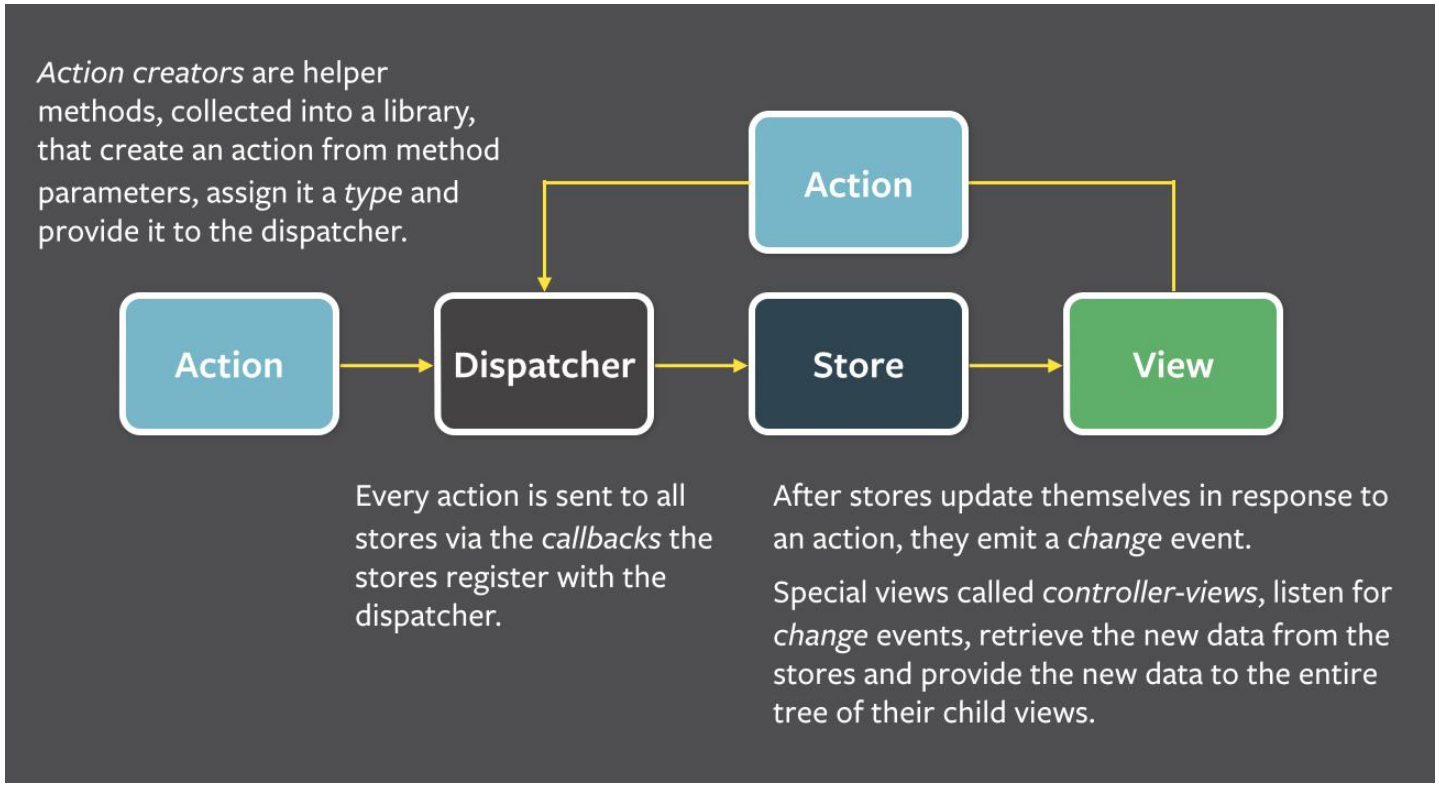


- More details information below

https://docs.angularjs.org/guide/scope#integration-with-the-browser-event-loop

Proposition: Flux

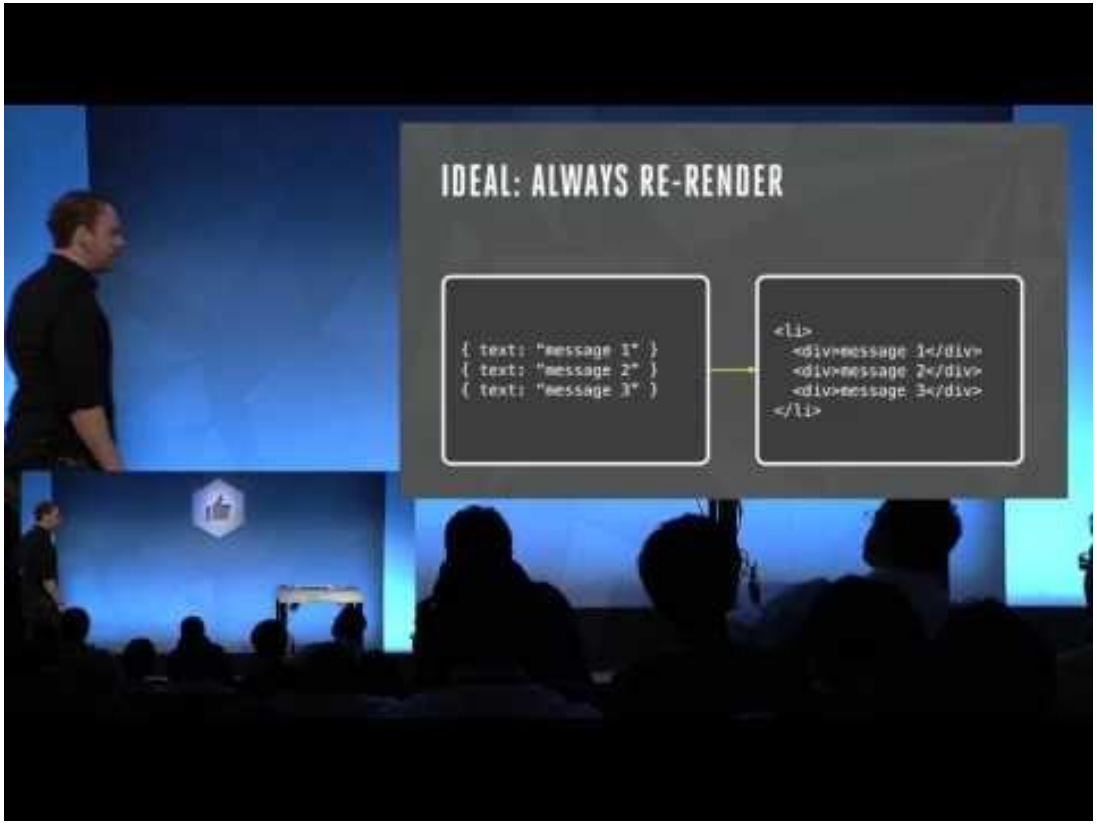
- Single direction data flow: FLUX



https://facebook.github.io/flux/docs/in-depth-overview.html#content

Proposition: Flux

- Details information and explanation here



https://facebook.github.io/flux/docs/in-depth-overview.html#content

UI update Issue

- How updating web page efficiency after data modification ?

```
{ text: 'message 1'}
{ text: 'message 2'}
```

+

```
{ text: 'message 3'}
```

→

```
<li>
    <div>message 1</div>
    <div>message 2</div>
<li>
```

```
Append
<div>message 2</div>
```

UI update Issue

- How updating web page efficiency after data modifications ?

```
{ text: 'message 1'}
{ text: 'message 2'}
{ text: 'message 3'}
```

```
<li>
    <div>message 1</div>
    <div>message 2</div>
    <div>message 3</div>
<li>
```

→ We want to always re-render!

Proposition: React.js

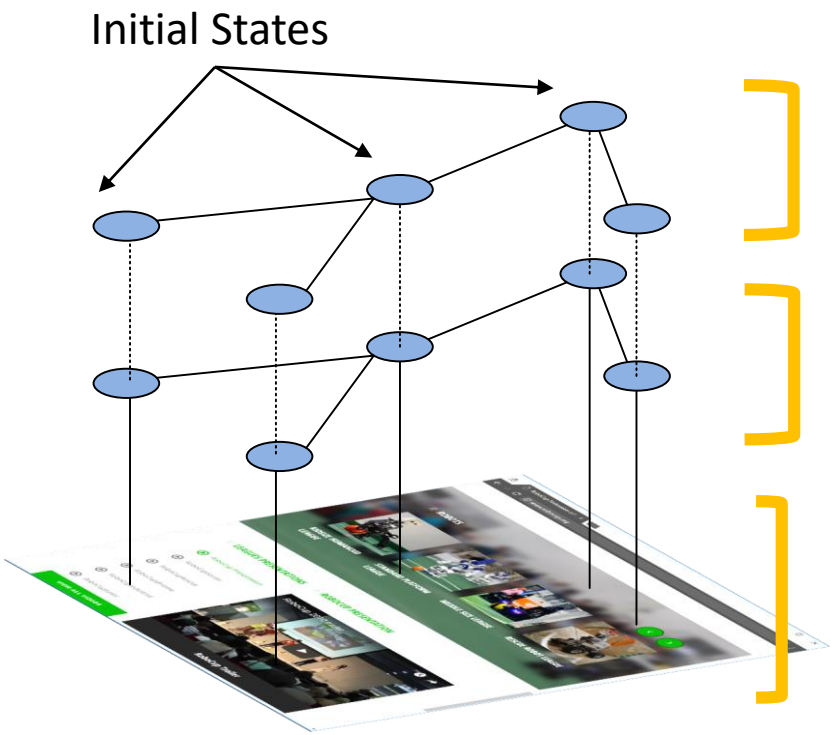
- What is React.js?
  - A library for building reusable UI components
  - Implements one-way reactive data flow
  - Mostly use as the V of the MVC.

- React.js Properties
  - Optimize the DOM Update through Virutal DOM
  - Use the Javascript syntax extension (JSX)

https://fr.reactjs.org/docs/introducing-jsx.html

CPE LYON

React.js : Virtual DOM

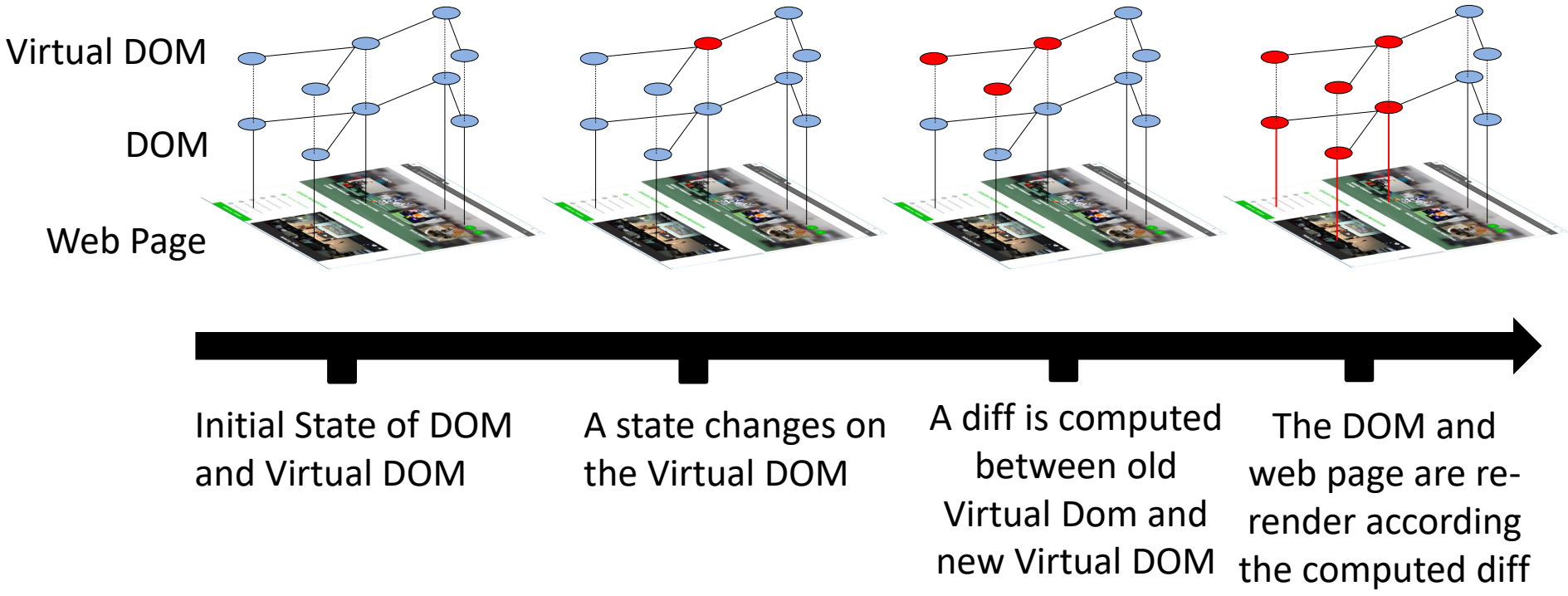


- **Light copy** of the DOM
- **Quick** navigation and update
- Detached from Browser specific

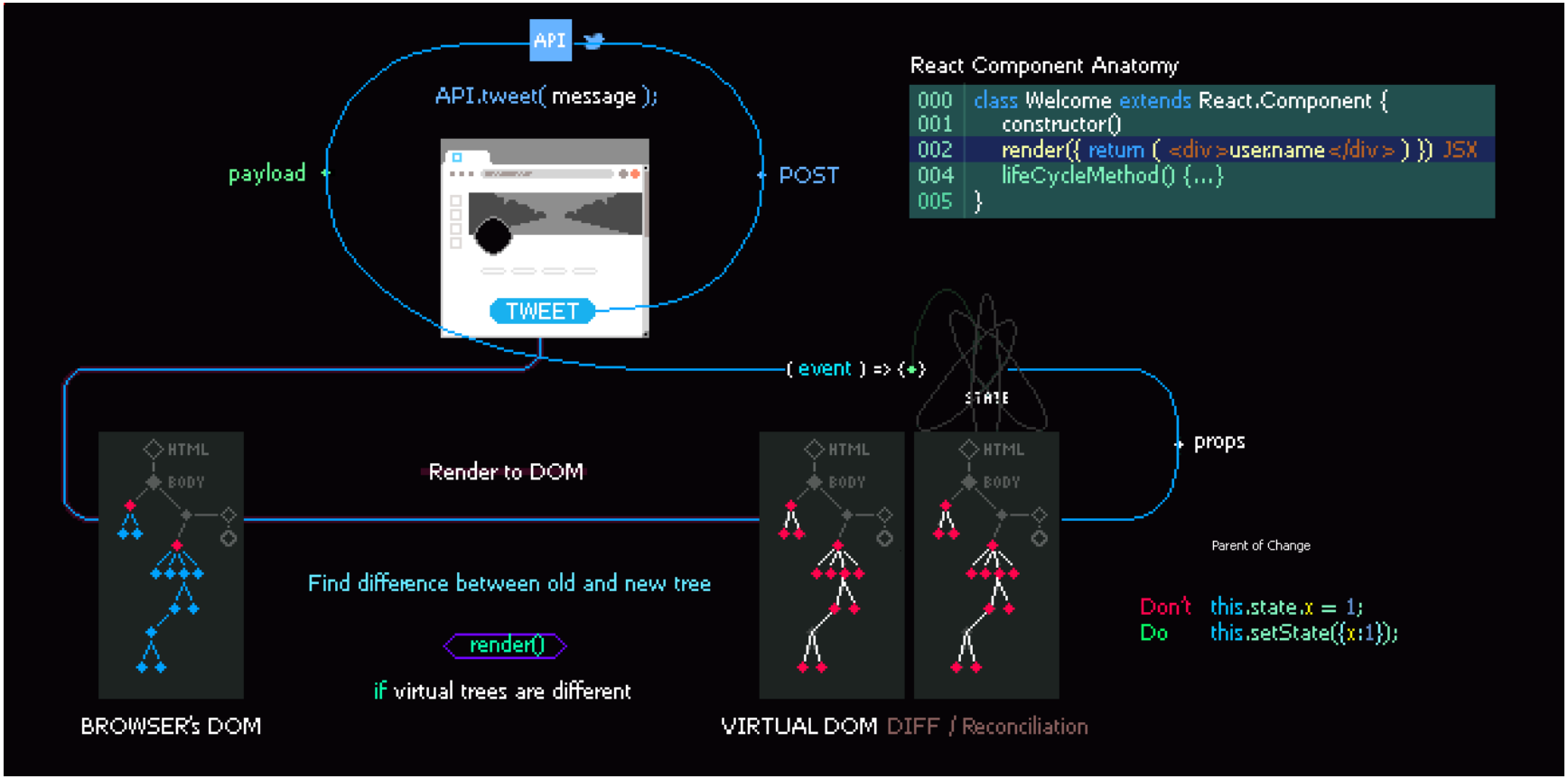
- Hierarchical component view
- Could update and navigate **slowly**
- Browser specific

Ref: http://reactkungfu.com/2015/10/the-difference-between-virtual-dom-and-dom/

React.js : Virtual DOM



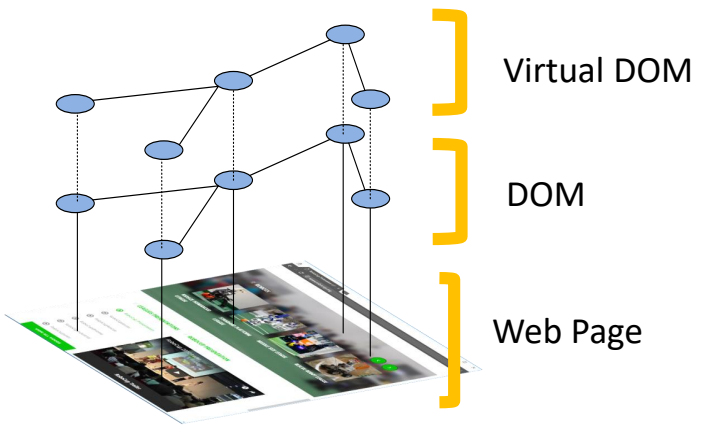
React.js : Virtual DOM



https://jstutorial.medium.com/react-animated-tutorial-7a46fa3c2b96

React.js objects

- ReactElement
  - Lowest type of virtual dom
- ReactNode
  - Hierachical Element of the virtual dom
  - ReactElement, string, number
  - Array of virtual nodes
- ReactComponent
  - Specification of how to build react elements



Proposition: JSX

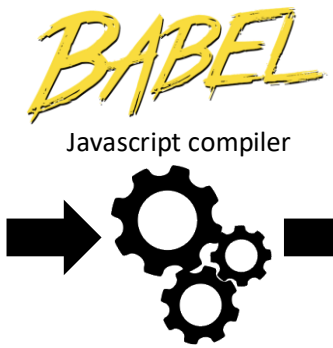
- What is JSX?
  - Syntax near the Html syntax helping describing visual result
  - Associate Javacript elts/results and UI elements
  - Helping building React components (base of Virutal Dom)

https://fr.reactjs.org/docs/introducing-jsx.html

- How JSX Works ?

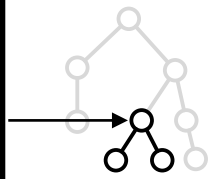
JSX

```
const txt = "Bonjour, monde !"
const elt = (
  <h1 className="greeting">
    {txt}
  </h1>
);
```



Virtual Dom Elt

```
// structure simplifiée
const elt = {
    type: 'h1',
    props: {
      className: 'greeting',
      children: 'Bonjour, monde !'
    }
  };
```



Proposition: JSX

JSX

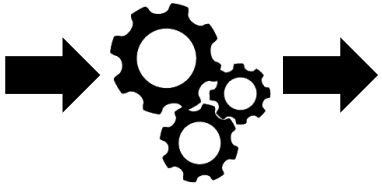
```jsx
const txt = "Bonjour, monde !"
const elt = (
  <h1 className="greeting">
    {txt}
  </h1>
);
```

JAVASCRIPT

```javascript
const txt = "Bonjour, monde !"
const elt = React.createElement(
  'h1',
  {className: 'greeting'},
  txt
);
```

BABEL

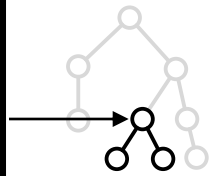
Javascript compiler



Virtual Dom Elt

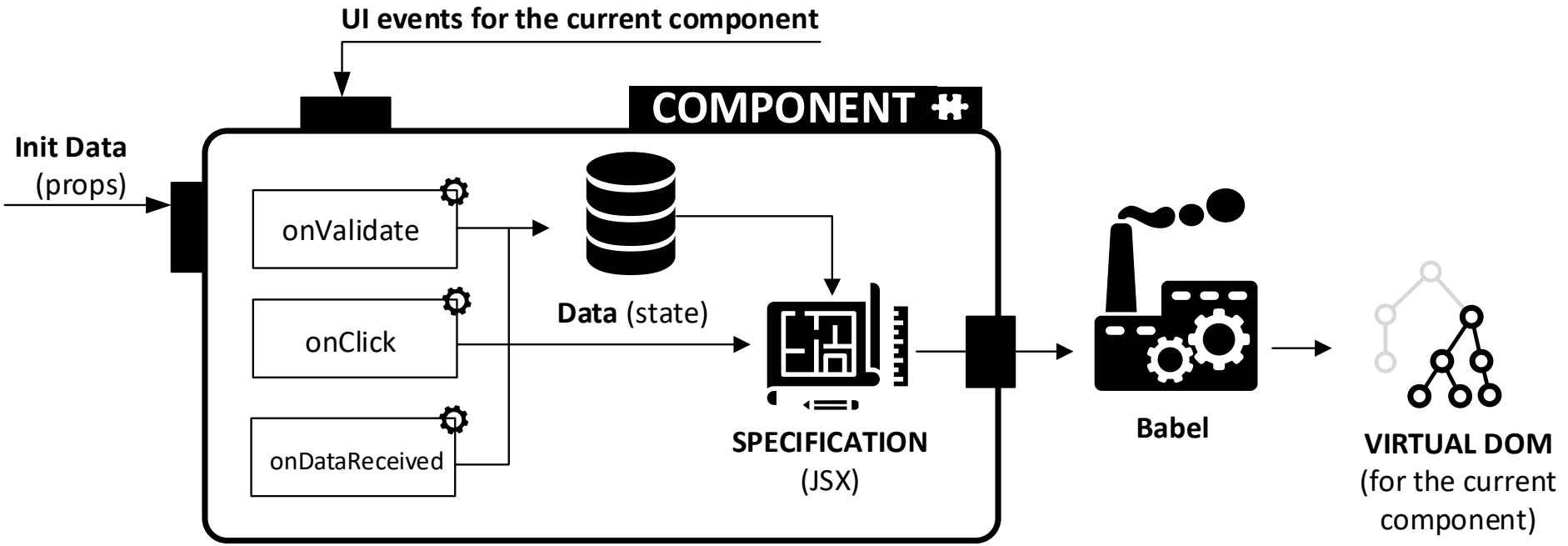
```javascript
// structure simplifiée
const elt = {
    type: 'h1',
    props: {
      className: 'greeting',
      children: 'Bonjour, monde !'
    }
  };
```

Virtual Dom Elt



React.js Component

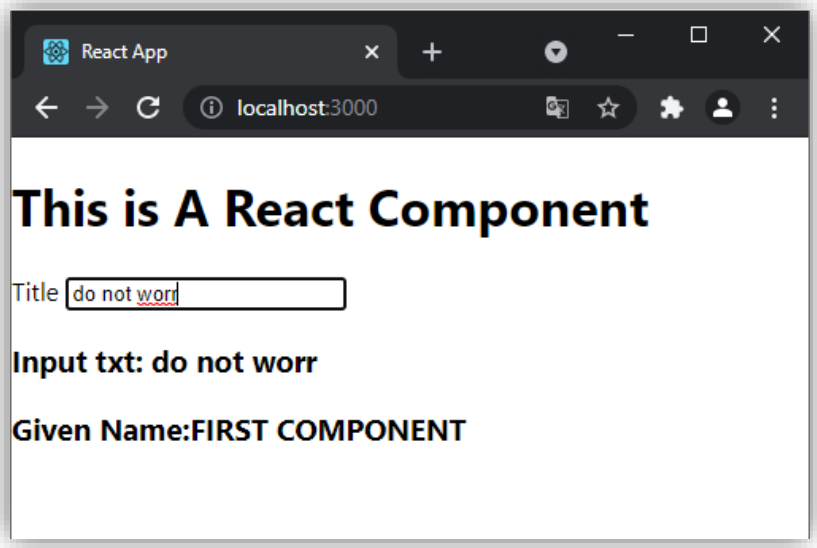
- In react.js everything (mostly) is a component



React.js Component

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import {App} from './App';

ReactDOM.render(
   <App name="FIRST COMPONENT"/>
  ,document.getElementById('root'));
```



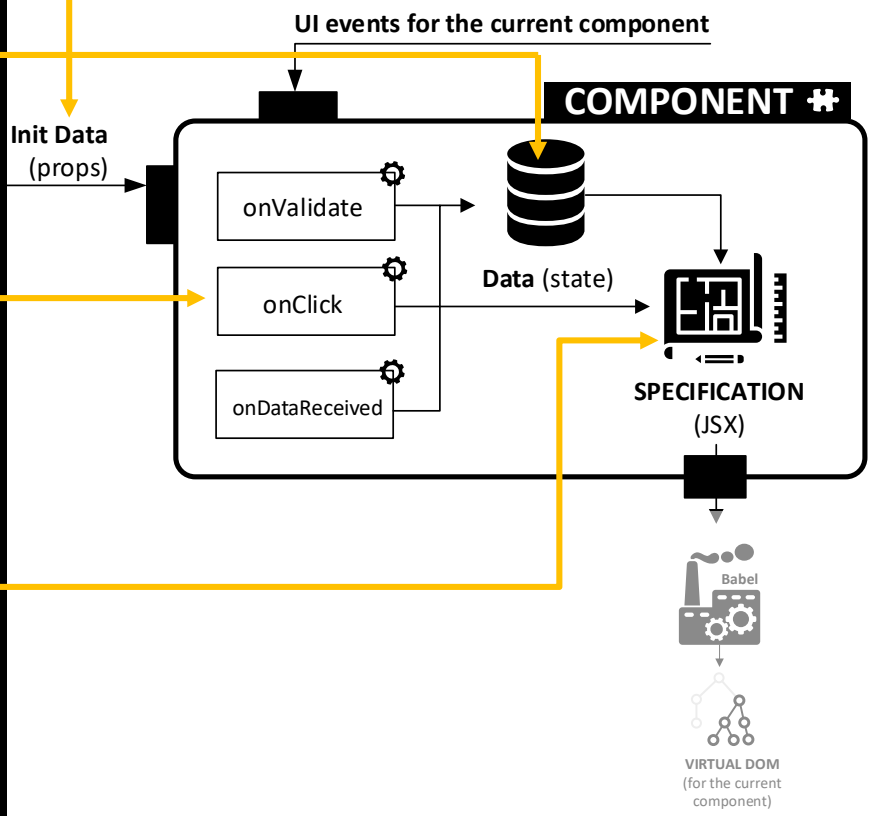
App.js

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

export const App =(props) => {
  //Get information from the props
  let given_name = props.name;

  //Create a local state
  const [title, setTitle] = useState(props.title);

   function handleChangeTitle(e){
      const txt= e.target.value
      //modify the locate state of txt
      setTitle(txt);
    }

return (
    <div className="App">
      <h1> This is A React Component</h1>
      <label htmlFor="titleInput">Title </label>
          <input type="text" id="titleInput"
              onChange={handleChangeTitle}
              value={title}
          />
          <h3>Input txt: {title}</h3>
          <h3>Given Name:{given_name}</h3>
    </div>
  );
}
```

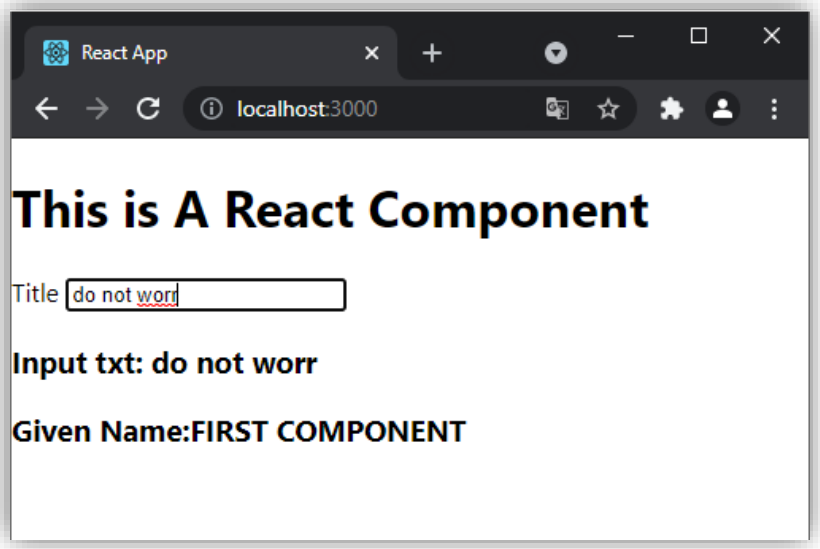
React.js Component

App.js

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

export const App =(props) => {
  //Get information from the props
  let given_name = props.name;

  //Create a local state
  const [title, setTitle] = useState(props.title);

   function handleChangeTitle(e){
      const txt= e.target.value
      //modify the locate state of txt
      setTitle(txt);
    }

return (
    <div className="App">
      <h1> This is A React Component</h1>
      <label htmlFor="titleInput">Title </label>
          <input type="text" id="titleInput"
              onChange={handleChangeTitle}
              value={title}
          />
          <h3>Input txt: {title}</h3>
          <h3>Given Name:{given_name}</h3>
    </div>
  );
}
```



React.js Component



Application DOM

```
<!DOCTYPE html>
<html lang="en"> event
▶ <head> ··· </head>
▼ <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
  ▼ <div id="root">
    ▼ <div class="App">
        <h1>This is A React Component</h1>
        <label for="titleInput">Title</label>
        <input id="titleInput" type="text" value="do not worr"> event
      ▼ <h3>
          Input txt:
          do not worr
        </h3>
      ▼ <h3>
          Given Name:
          FIRST COMPONENT
        </h3>
      </div>
    </div>
    <script src="/static/js/bundle.js"></script>
    <script src="/static/js/0.chunk.js"></script>
    <script src="/static/js/main.chunk.js"></script>
    <script src="/main.8b90e06a40b59e43e63e.hot-update.js"></script>
  </body>
</html>
```

Syntax

- Javascript based on the **ES6 standard**
- Object syntactic sugar
- New symbols usage (generator, arrow, functions, spread operators)
- Set of libraries (promises, new collections, types arrays)

Syntax ES6

From http://slidedeck.io/DonaldWhyte/isomorphic-react-workshop

CLASSES

```
class Point {
  constructor(x, y) {
       this.x = x; this.y = y;
   }
   toString() { return `(${this.x}, ${this.y})`; }
}
```

IMPORTING OTHER MODULES

Modules are JavaScript files.

```
// import foo.js in same dir as current file import foo
from './foo';
foo.foobar(42);

// import specific variables/functions from a module
import { foobar } from './foo'; foobar(42);
```

LET and CONST

```
const x_const
Let x_var

 x_var = x_var +1
x_const = x_const +50 // → error
```

INHERITENCE

```
class ColorPoint extends Point {
     constructor(x, y, color) {
         super(x, y);
         this.color = color;
    }
    toString() { return super.toString() + ' in ' + this.color; }
}
```

EXPORTING SYMBOLS

foo.js:

```
export function foobar(num) { console.log('FOOBAR:', num); }
```

EXPORTING ENTIRE OBJECTS

person.js:

```
export default { name: 'Donald', age: 24 };
```

another_file.js

```
import person as './person'; console.log(person.name);
// outputs 'Donald'
```

FUNCTION

```
()=>{return 'hello';}
```

SPREAD OPERATOR

```
let fruits = ['Apple','Orange','Banana'];
let newFruitArray = [...fruits];
```

CPE LYON

React.js Component

App.js

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

export const App =(props) => {
  //Get information from the prop
  let given_name = props.name;

  //Create a local state
  const [title, setTitle] = useState(props.title);

   function handleChangeTitle(e){
      const txt = e.target.value;
      //modify the local state of title
      setTitle(txt);
    }

return (
    <div className="App">
      <h1> This is A React Component</h1>
      <label htmlFor="titleInput">Title </label>
          <input type="text" id="titleInput"
              onChange={handleChangeTitle}
              value={title}
          />
          <h3>Input txt: {title}</h3>
          <h3>Given Name:{given_name}</h3>
    </div>
  );
}
```

Function Based

App.js

```
import React, { Component } from 'react';
class App extends Component {
  constructor(props) {
  super(props);
  this.state = { title:this.props.title, };
  this.handleChangeTitle=this.handleChangeTitle.bind(this);
   }

 handleChangeTitle(e){
   this.setState({title:e.target.value});
}

 render(){
  return(
    <div className="App">
      <h1> This is my first React Component</h1>
      <label htmlFor="titleInput">Title </label>
        <input type="text" id="titleInput"
          onChange={this.handleChangeTitle}
          value={this.state.title}
        />
        <h3>{this.state.title}</h3>
    </div>
  ); }}
export default App;
```

Class Based

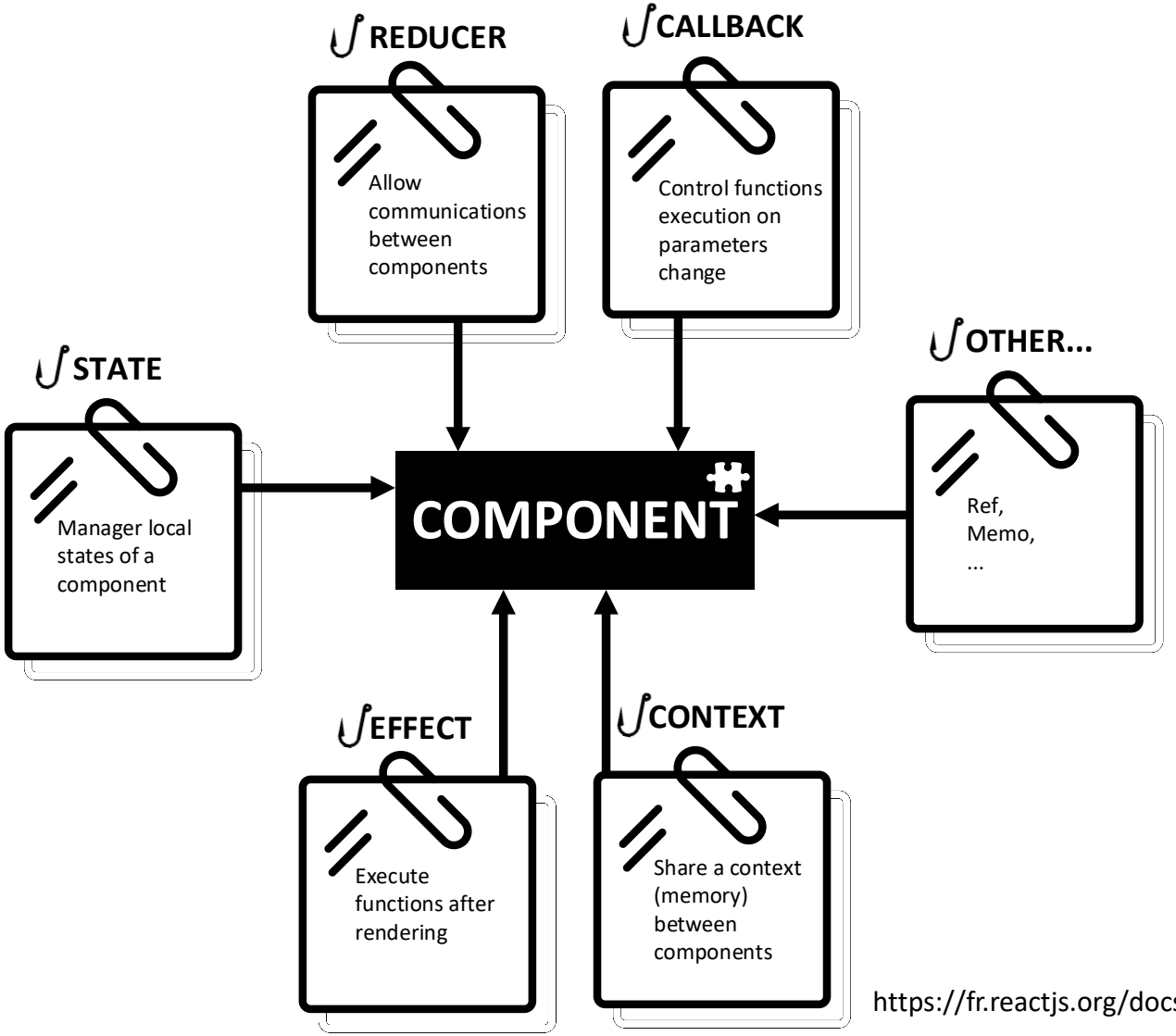
React.js Hooks

> But what is a Hook?
>
> Hooks are functions that let you "hook into" React state and lifecycle features from function components. Hooks don't work inside classes — they let you use React without classes. (We don't recommend rewriting your existing components overnight but you can start using Hooks in the new ones if you'd like.)

https://reactjs.org/docs/hooks-overview.html

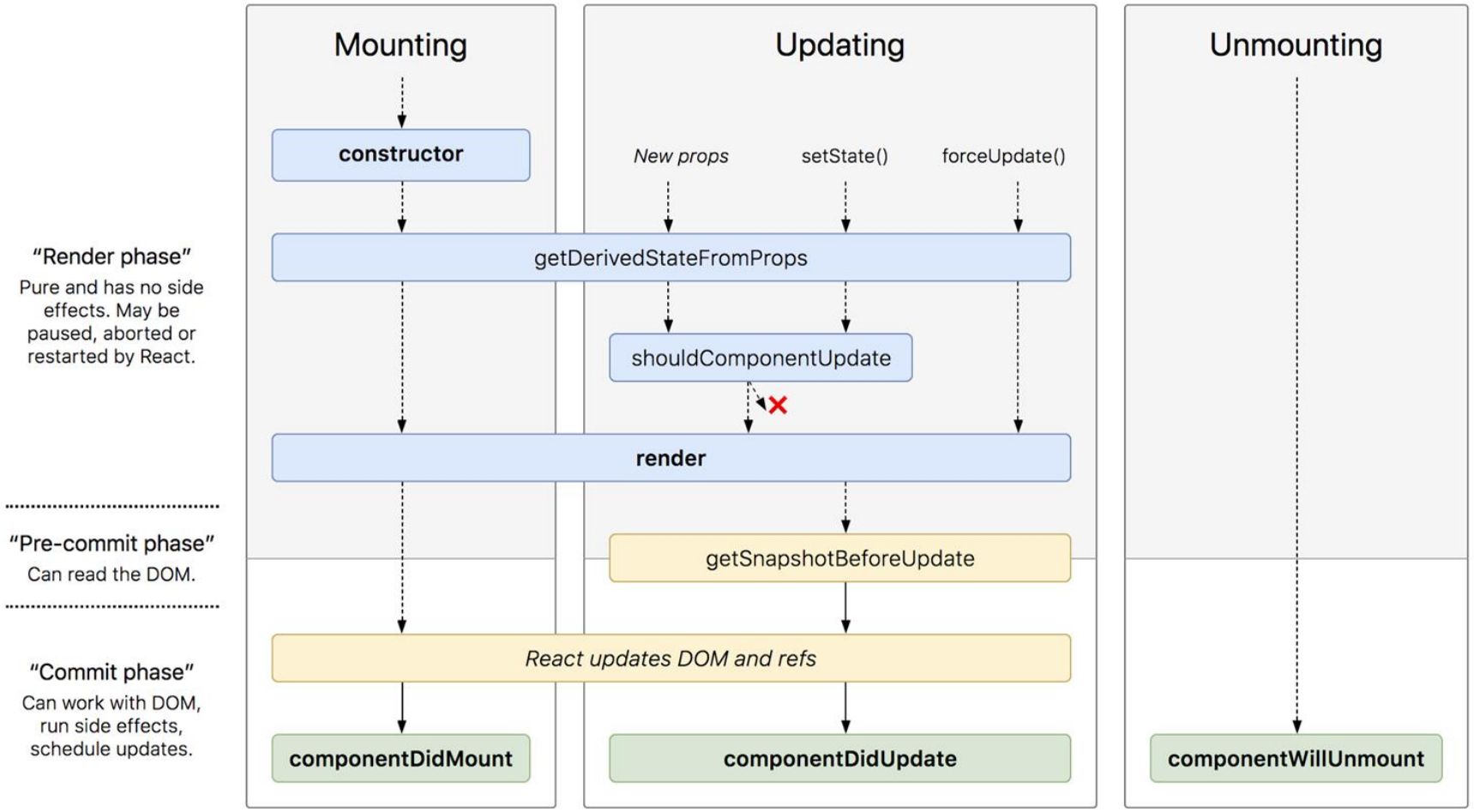
- Basics Hooks:
  - State Hooks
    - Enable to manage a local state. React will preserve this state between re-renders.
  - Effect Hooks
    - Enable to execution functions after flushing changes to the DOM

React.js Hooks



https://fr.reactjs.org/docs/hooks-reference.html

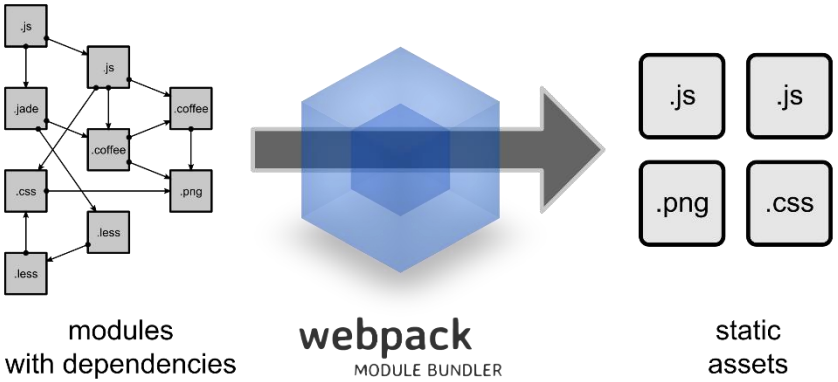
Component Lifecycle



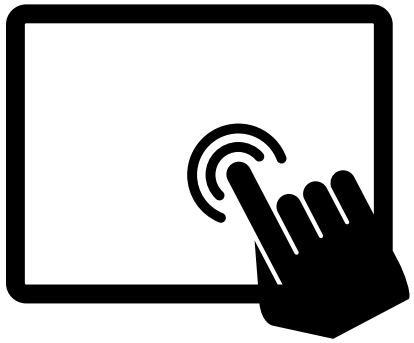
When does React re-render component?

- ❏ Update of props (changement of reference)
- ❏ Update of state (setState change the reference of current state)
- ❏ Parent is re-rendered

React ToolBoxe : Configure a read.js project



https://www.codementor.io/tamizhvendan/beginner-guide-setup-reactjs-environment-npm-babel-6-webpack-du107r9zr



React.js : Let's go!

Installation

https://react.dev/learn/start-a-new-react-project
https://vitejs.dev/guide/#scaffolding-your-first-vite-project

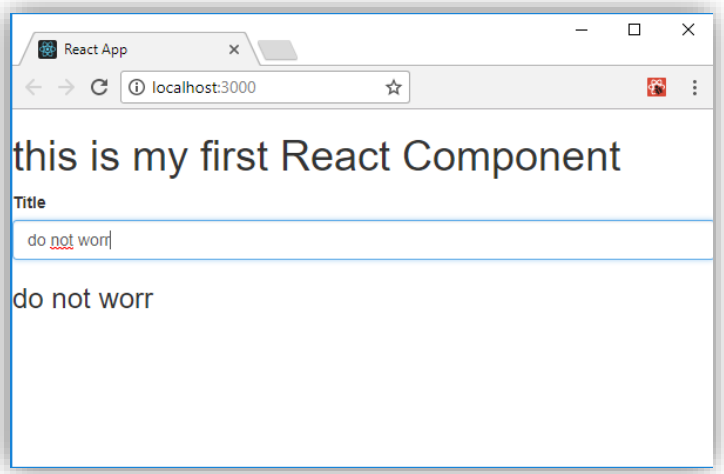
- Nodes.js
- Npm

```
npm create vite@latest my-app -- --template react
cd my-app
npm install
npm run dev
```

```
npm run build
```

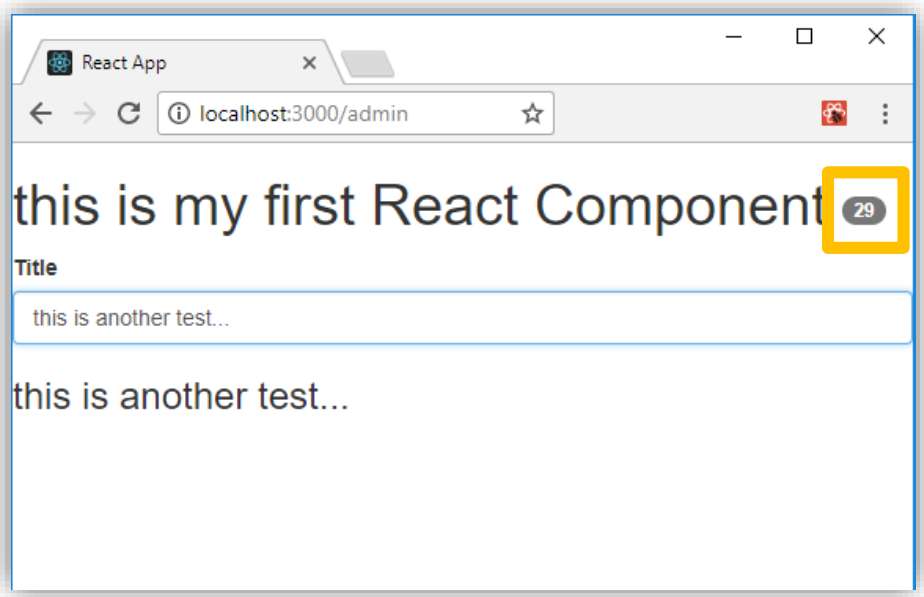
It is your turn !

- Create App allowing to get as input a title and print it below
- Your App component must be initialized with the title property = 'default_title '



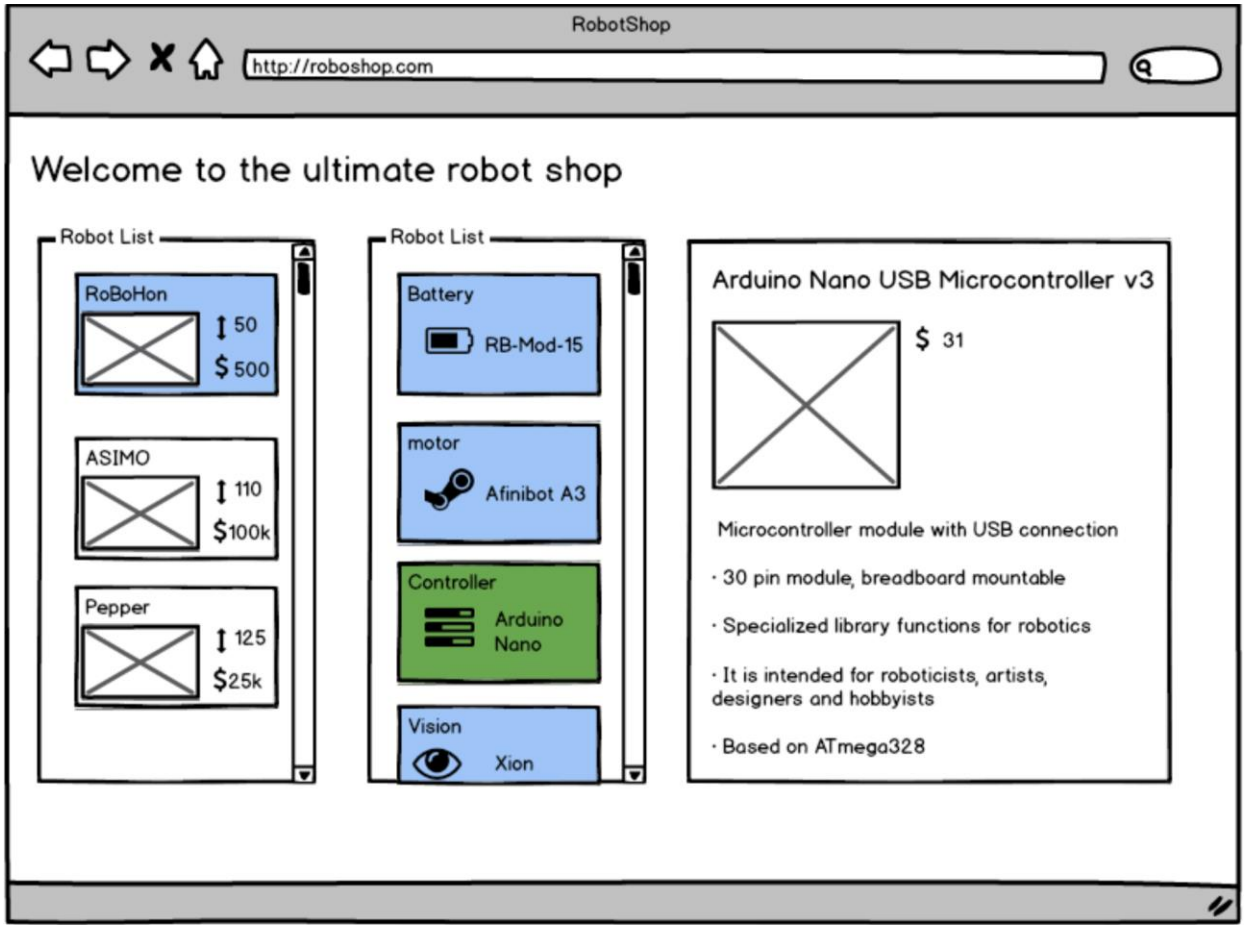
It is your turn !

- Update the number of mouse over the printed title



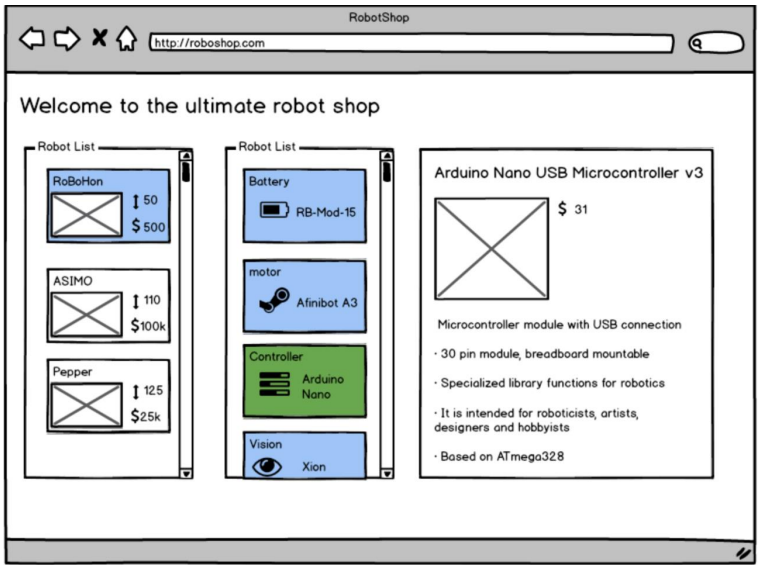
Best practices

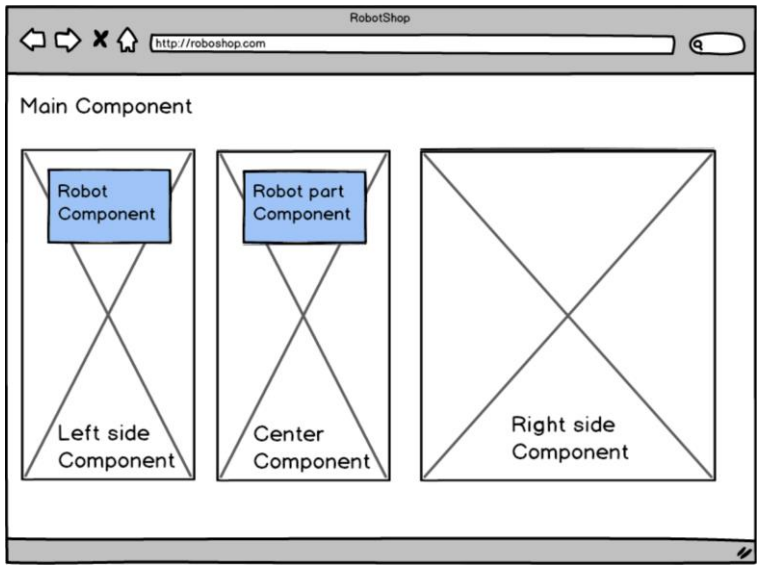
- Divide your application into components



Best practices

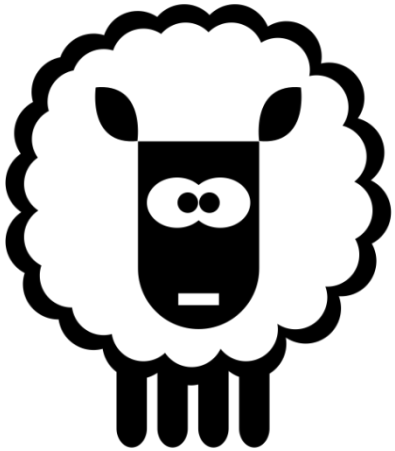
- Divide your application into components





Best practices

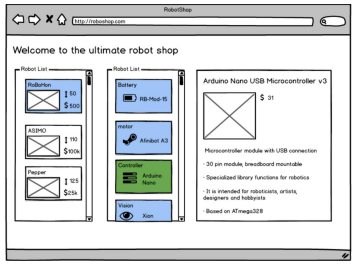
- **Displaying** (presentational Component) **Vs Processing** (container Component)

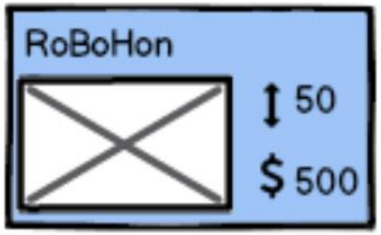


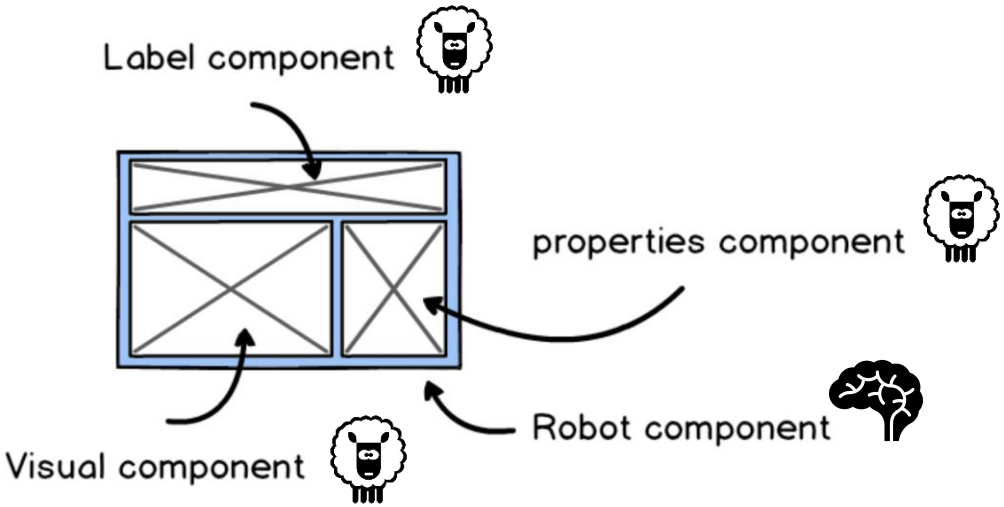
VS

Best practices

- Displaying Vs Processing

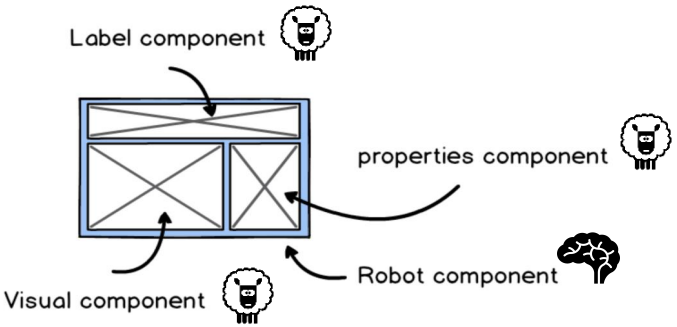




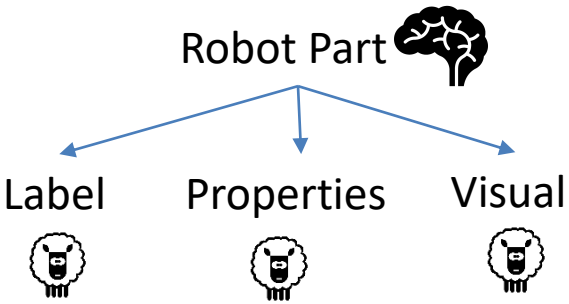


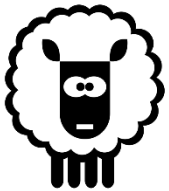
Best practices

- Displaying Vs Processing



Process Data

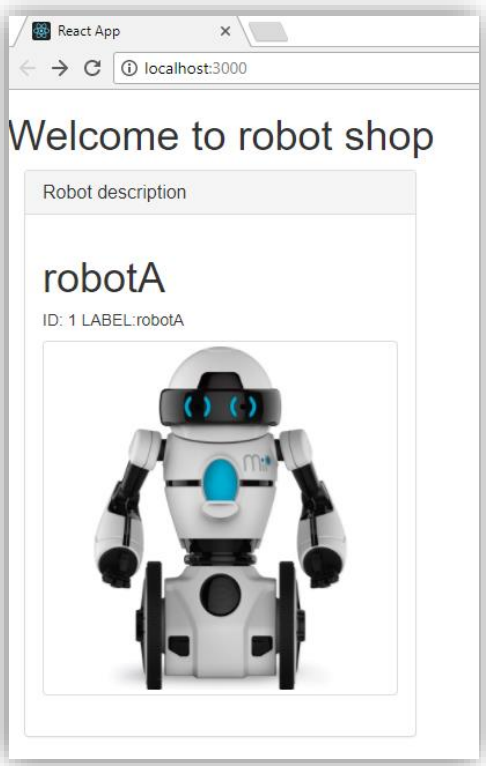




Display Data

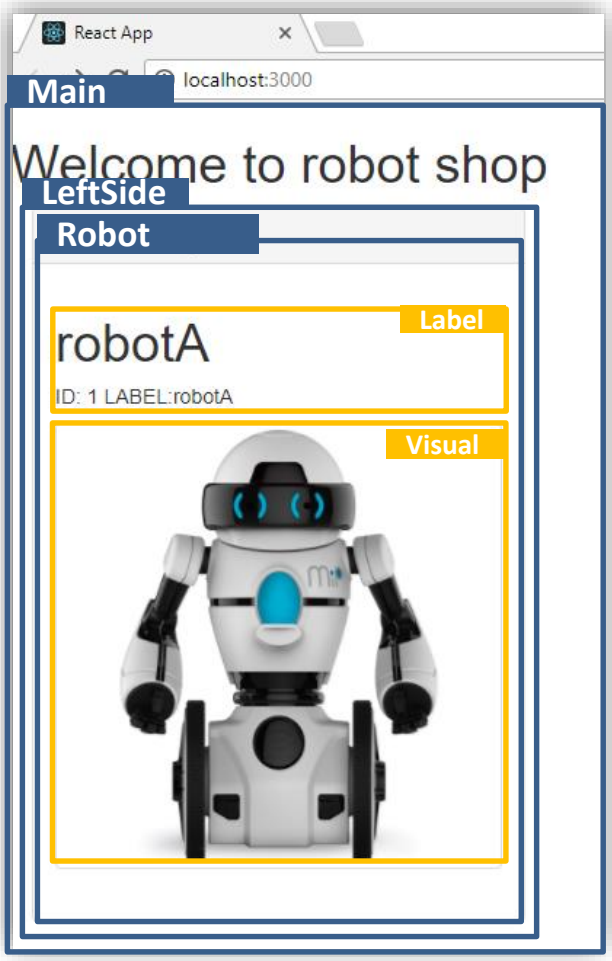
It is your turn !

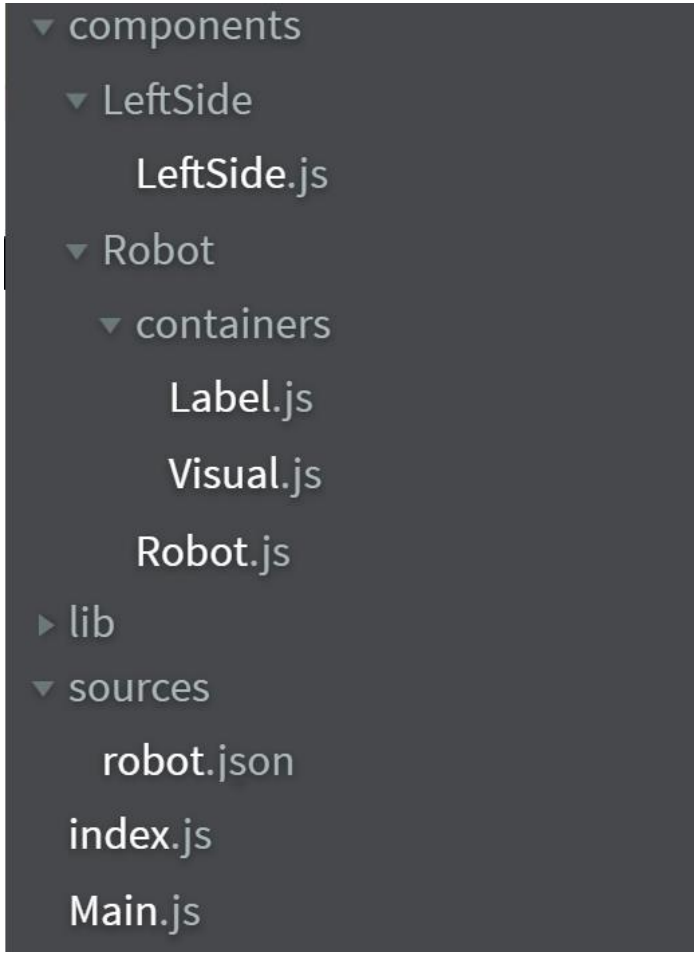
- Create the following application
- Create
  - A main component initialized with a json file
  - A Left side component
  - A Robot component
  - A Label component for the Robot Component



All steps available at:
https://github.com/jacques-saraydaryan/front-end-react.js.git

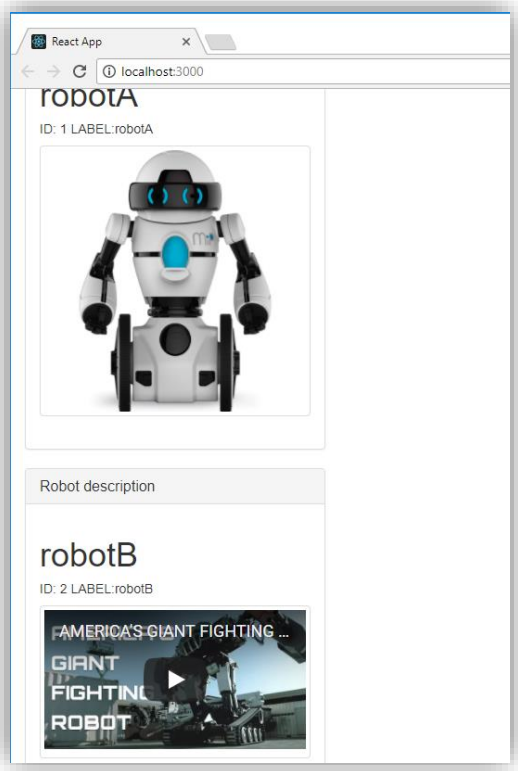
It is your turn !





It is your turn !

- Same as previously but with a list of robots
- Some robot can have a visual as a video



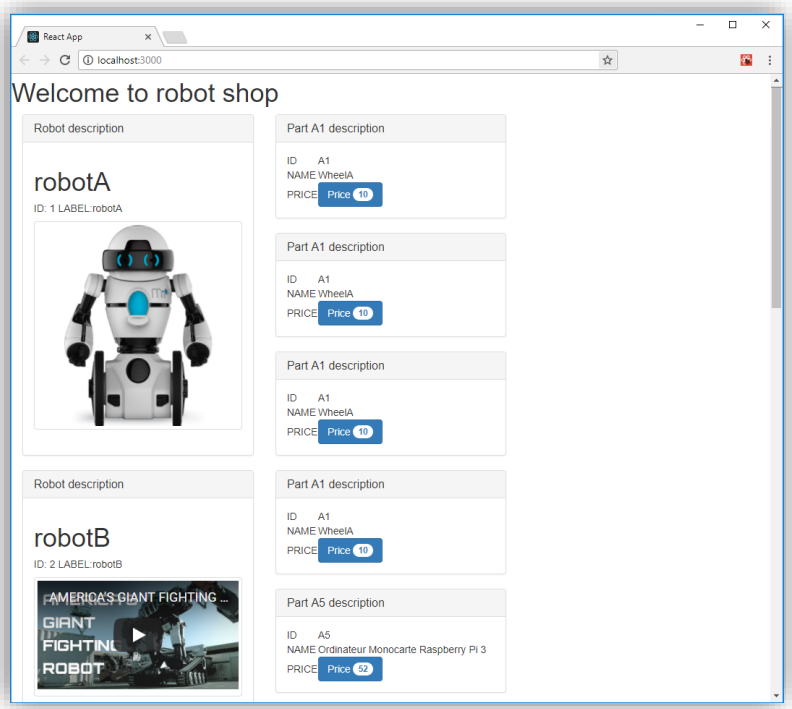
It is your turn !

- Tips
  - Use *<array>.map((item) => <your code>);* to display list of objects
  - Don't forget *key* that help react to identify which items have changed
  - More info. here https://reactjs.org/docs/lists-and-keys.html

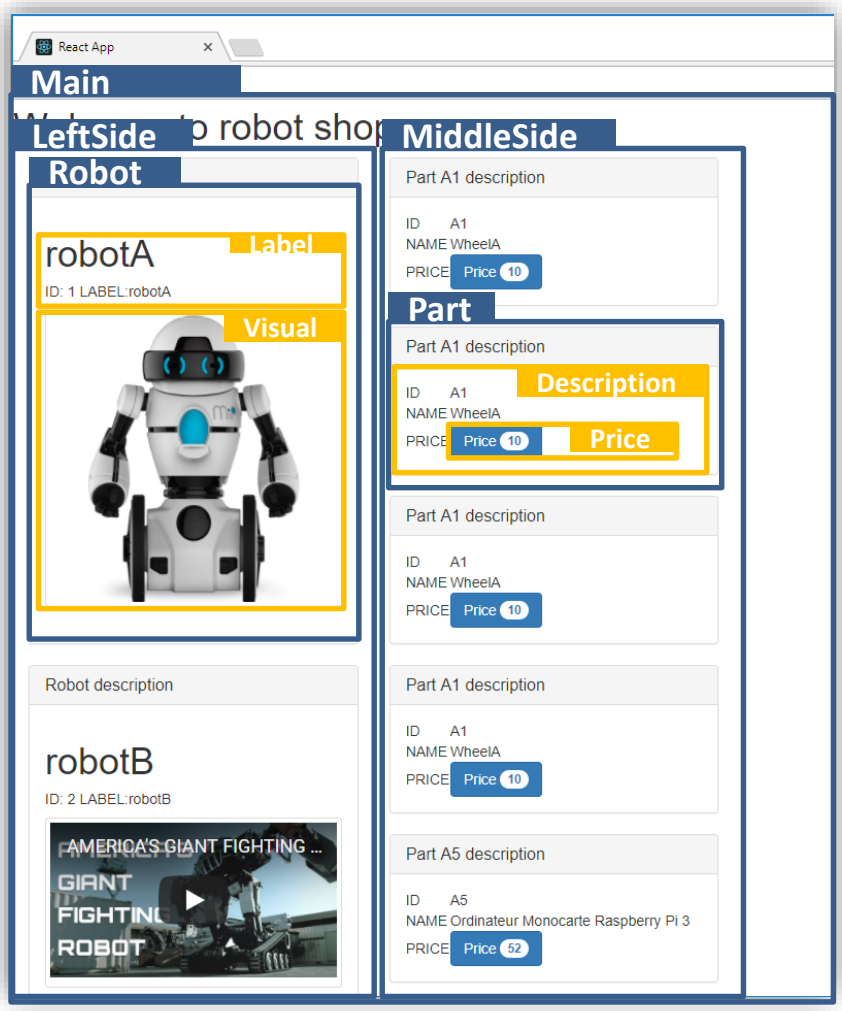
```
let objList=[
    {id:1,value:'A'},
    {id:1,value:'B'},
    {id:1,value:'C'}
];

let display= objList.map(
            (obj) =>
                <div key={obj.id}>
                    <h2>{obj.id}</h2>
                    <h3>{obj.value}</h3>
                </div>
            );
```

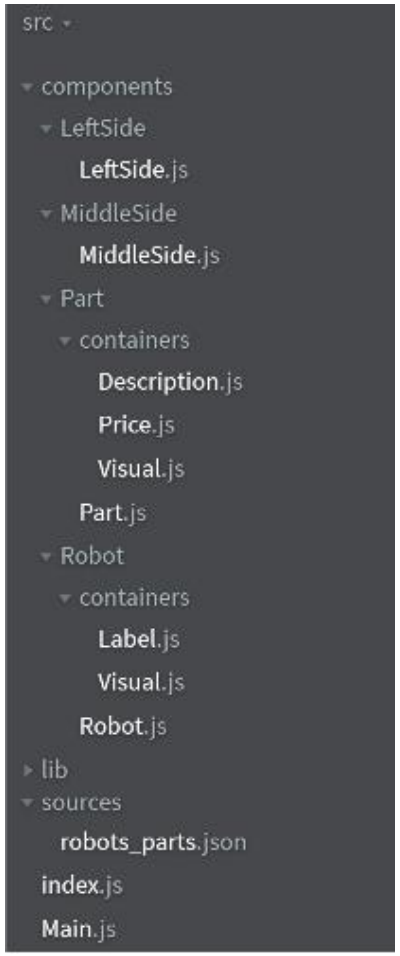
It is your turn !

- Add a Part Component using
  - Description Component
  - Price Component
  - Visual Component
- Add a MiddleSide Component displaying the list of parts of the selected robot



It is your turn !





It is your turn !

Robots_parts.json

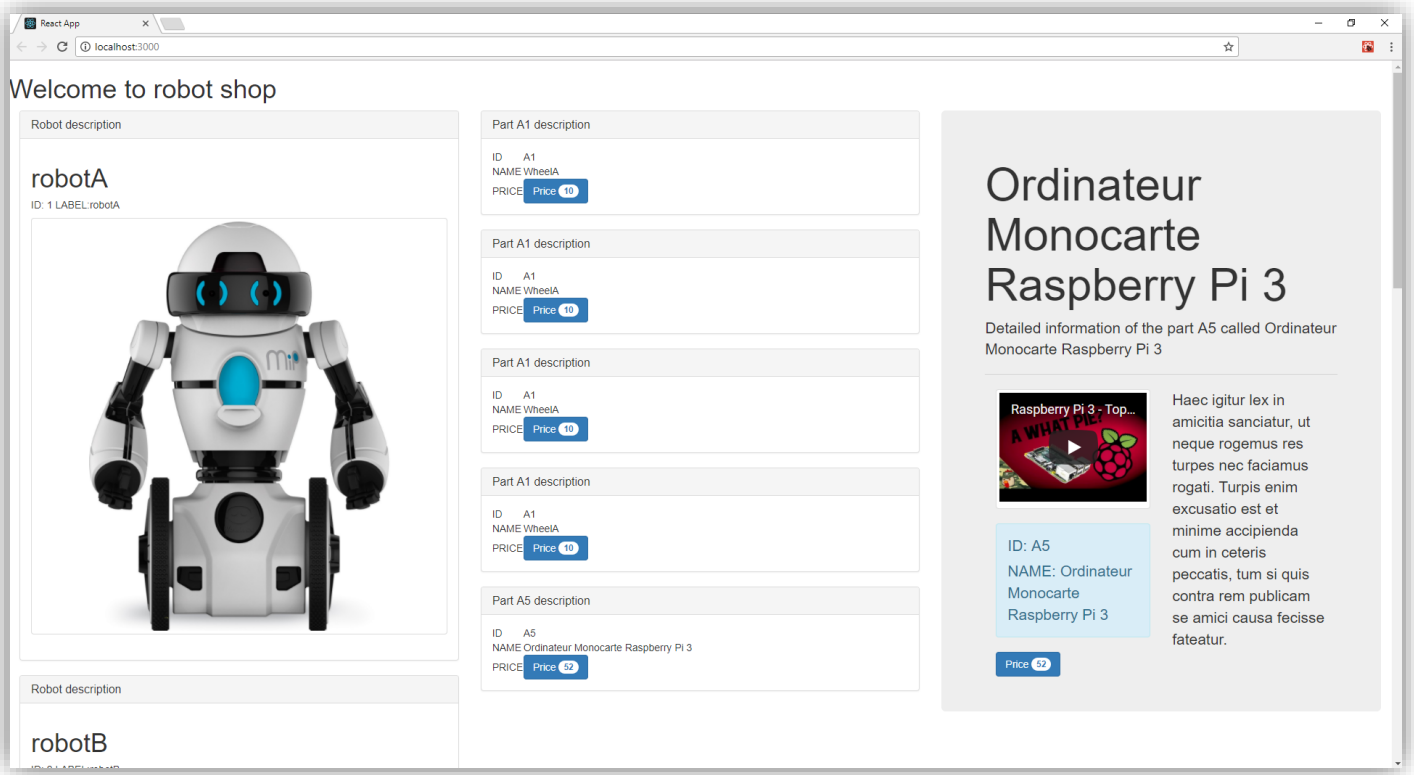

```
{"robots":[
  {
    "id":1,
    "title":"robotA",
    "visual_type":"img",
    "visual_src":"https://www.robot-advance.com/EN/ori-wowwee-
mip-white-robot-1281_1613.jpg",
    "parts":["A1","A1","A1","A1","A4"]
  },
  {
    "id":2,
    "title":"robotB",
    "visual_type":"video",
    "visual_src":"https://www.youtube.com/embed/ePINYZK4p5Y",
    "parts":["A2","A2","A4"]
  },
],
```

```
"parts":[
  {
    "id":"A1",
    "title":"WheelA",
    "price":10
  },
  {
    "id":"A2",
    "title":"WheelB",
    "price":15
  },
  {
    "id":"A3",
    "title":"WheelC",
    "price":150
  },
  {
    "id":"A4",
    "title":"Contrôleur de Servomoteurs USB SSC-32U Lynxmotion",
    "price":57
  }]
}
```

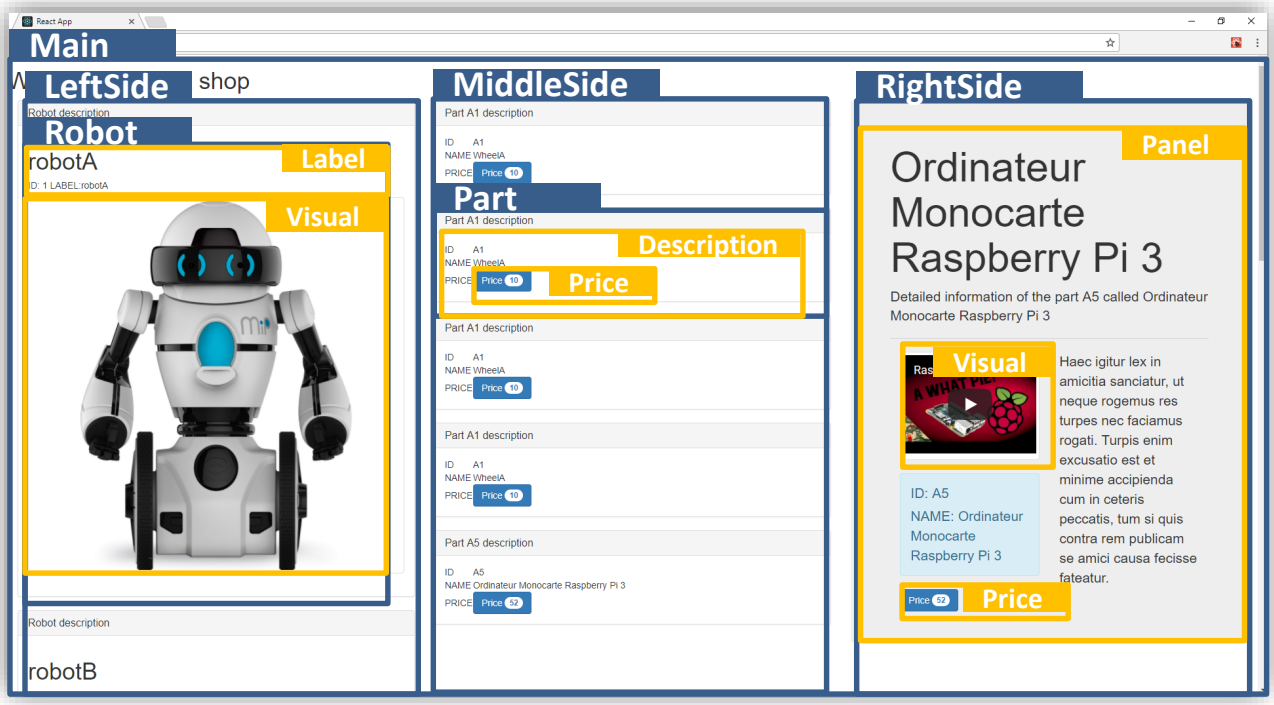

```
import data from './data/robots_parts.json'
```

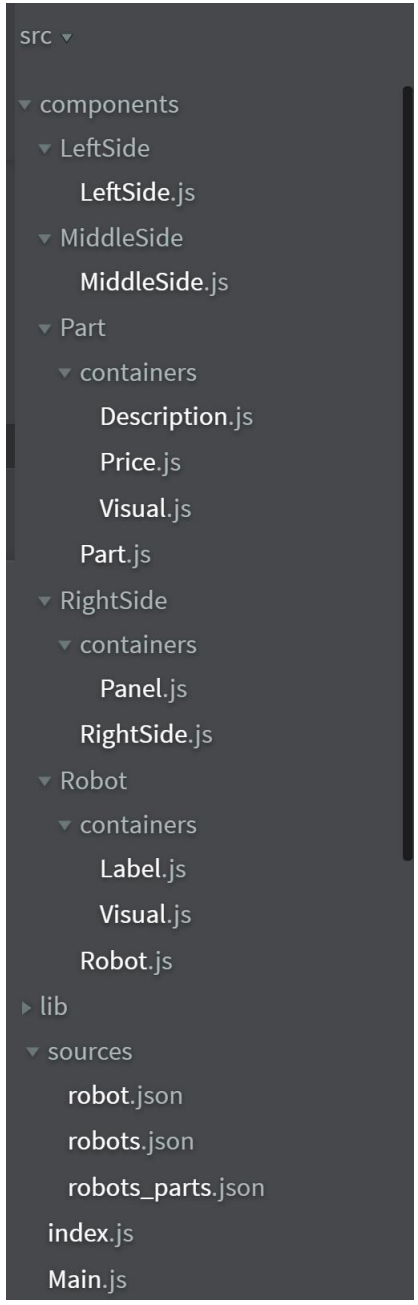
It is your turn !

- Add a RightSide Component using
  - Panel Component displaying selected part



It is your turn !

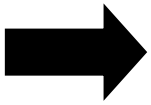


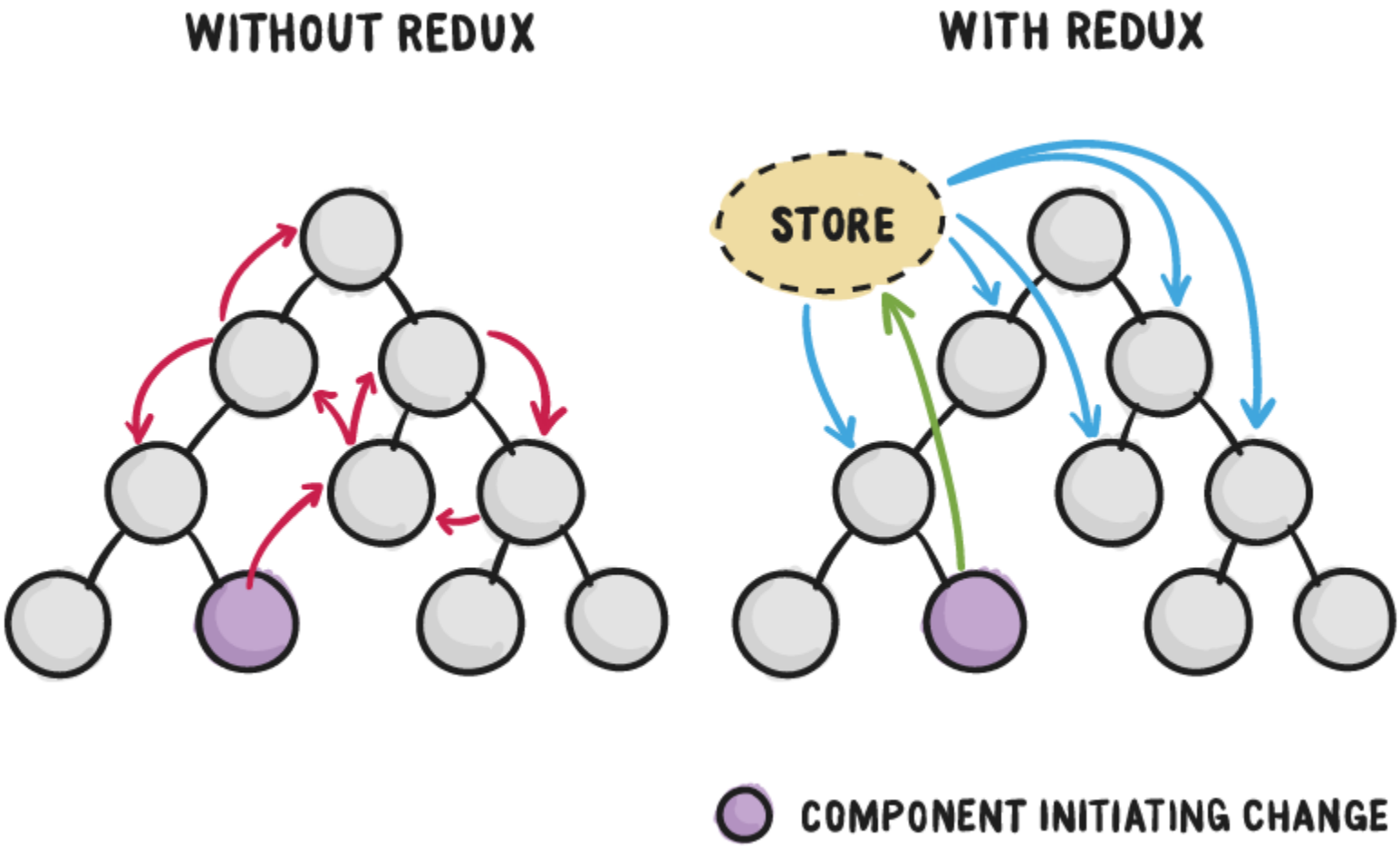


React.js :
Our current State
Redux as enhancement

React.js and so?

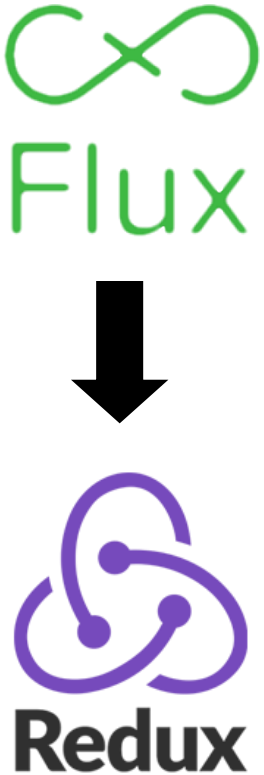
- **Pro**
  - Components Based
  - Extremely efficient (Virtual Dom)
  - Easy to write module base code
  - UI Test Cases easy to write
- **Con**
  - Only for the view layer
  - Write visual component into Javascript
  - Hard to learn
  - **Hierarchical dependencies !**





https://medium.com/@fknussel/redux-3cb5aac94a66

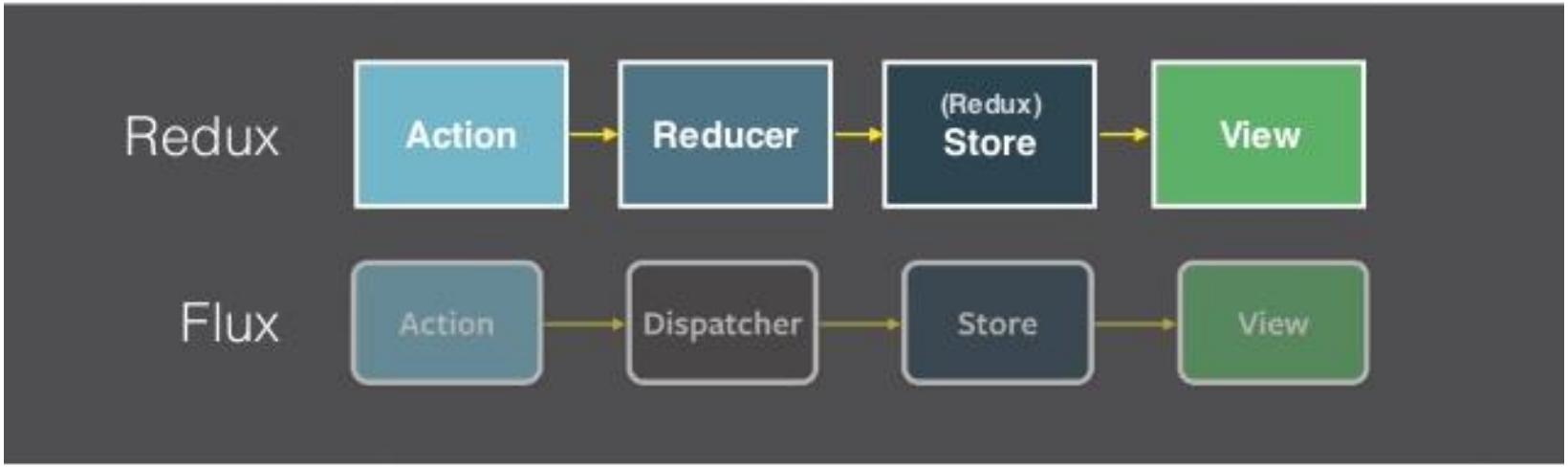
Redux



Can be view as an

implementation of Flux

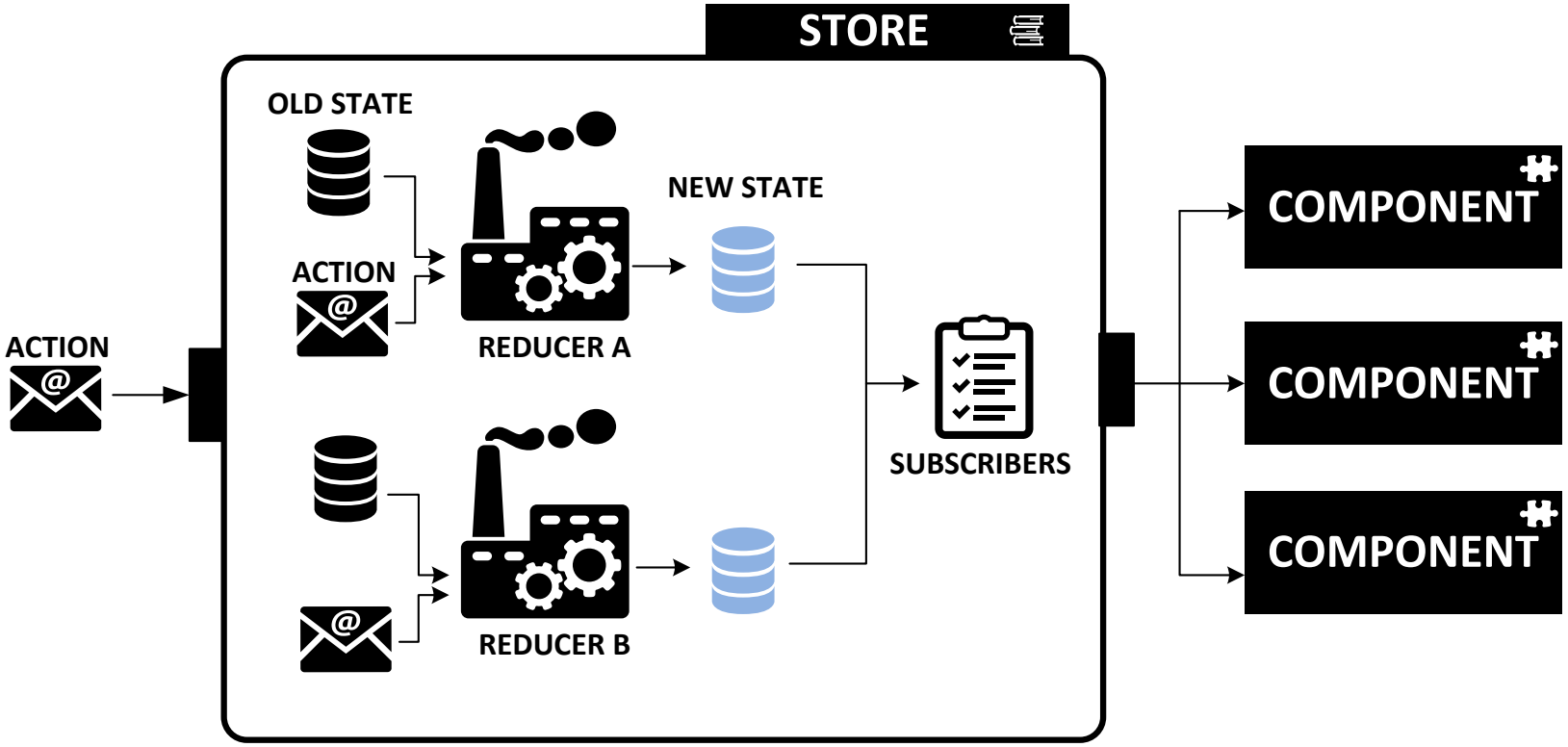
Redux



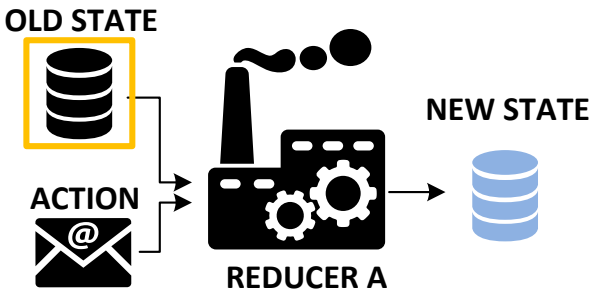
Redux vs traditional Flux

https://www.slideshare.net/JonasOhlsson/using-redux

Redux


STORE
OLD STATE
ACTION
REDUCER A
NEW STATE
ACTION
REDUCER B
SUBSCRIBERS
COMPONENT
COMPONENT
COMPONENT

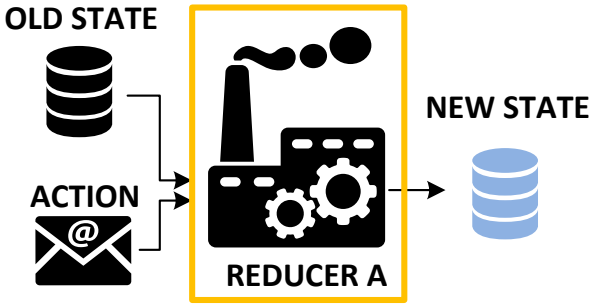
Redux - Slice



1 Define Init State

```
import { createSlice } from '@reduxjs/toolkit'

export const formatSlice = createSlice({
  name: 'text',
  initialState: {
    txt: '',
  },
  reducers: {
    format_txt: (state, action) => {
      let current_txt = action.payload.txt
      current_txt= current_txt.toUpperCase()
      current_txt= current_txt.replaceAll('A','@')
      current_txt= current_txt.replaceAll('FOR','4')
      current_txt= current_txt.replaceAll('E','3')
      state.txt = current_txt
    },
    reset_txt: (state) => {
      state.txt = ''
    },
  },
})

export const { format_txt, reset_txt } = formatSlice.actions

export default formatSlice.reducer
```

Redux - Slice



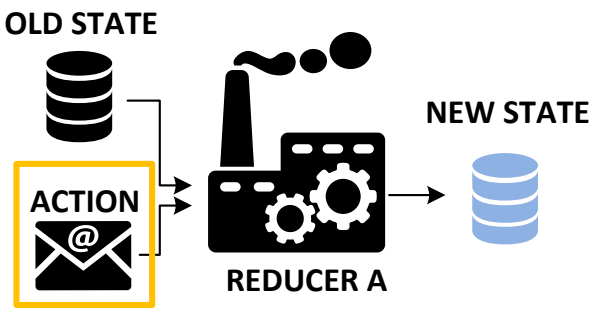
1. Define Init State

2. Define Behaviors (Reducer)

```
import { createSlice } from '@reduxjs/toolkit'

export const formatSlice = createSlice({
  name: 'text',
  initialState: {
    txt: '',
  },
  reducers: {
    format_txt: (state, action) => {
      let current_txt = action.payload.txt
      current_txt= current_txt.toUpperCase()
      current_txt= current_txt.replaceAll('A','@')
      current_txt= current_txt.replaceAll('FOR','4')
      current_txt= current_txt.replaceAll('E','3')
      state.txt = current_txt
    },
    reset_txt: (state) => {
      state.txt = ''
    },
  },
})

export const { format_txt, reset_txt } = formatSlice.actions

export default formatSlice.reducer
```

CPE LYON

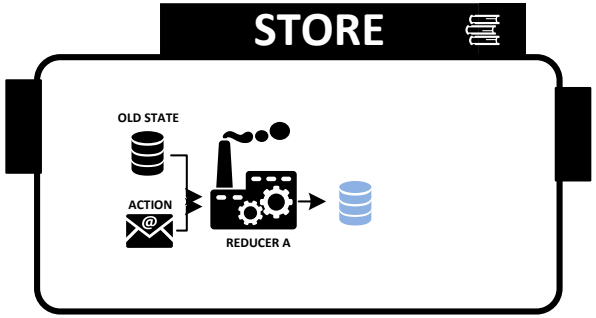
Redux - Slice



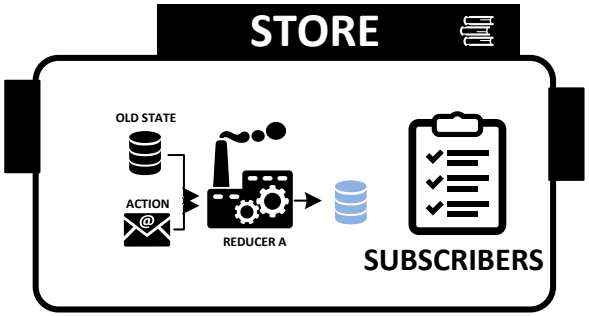
1. Define Init State
2. Define Behaviors (Reducer)
3. Define Entry Points (Action)

```
import { createSlice } from '@reduxjs/toolkit'

export const formatSlice = createSlice({
  name: 'text',
  initialState: {
    txt: '',
  },
  reducers: {
    format_txt: (state, action) => {
      let current_txt = action.payload.txt
      current_txt= current_txt.toUpperCase()
      current_txt= current_txt.replaceAll('A','@')
      current_txt= current_txt.replaceAll('FOR','4')
      current_txt= current_txt.replaceAll('E','3')
      state.txt = current_txt
    },
    reset_txt: (state) => {
      state.txt = ''
    },
  },
})

export const { format_txt, reset_txt } = formatSlice.actions

export default formatSlice.reducer
```

Redux



4 Create A Store with the Reducer



5 Component register to the Store

Store.js

```
import { configureStore } from '@reduxjs/toolkit';
import formatReducer from './slices/formatSlice';

export default configureStore({
  reducer: {
    formatReducer: formatReducer ,
    },
})
```

main.jsx

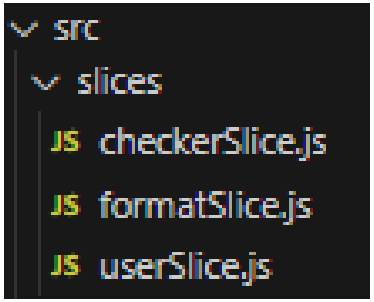
```
import React from 'react'
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import {App} from './App.jsx';
import store from './store.js';

ReactDOM.createRoot(document.getElementById(
'root')).render(
  <React.StrictMode>
    <Provider store={store} >
      <App title="Set your message"/>
    </Provider>
  </React.StrictMode>,
)
```

Redux in practice (1/4)

- Install reduce components

```
npm install redux

npm install react-redux

npm install @reduxjs/toolkit
```

- Create Slices into dedicated files and directories
- Use Store.js to merge several reducers



Store.js

```
import { configureStore } from '@reduxjs/toolkit';
import formatReducer from './slices/formatSlice';
import checkerReducer from './slices/checkerSlice';
import userReducer from './slices/userSlice';

export default configureStore({
  reducer: {
    formatReducer: formatReducer ,
    checkerR:      checkerReducer ,
    userR:         userReducer ,
    },
})
```

Redux in practice (1/4)

Troubleshooting

http://redux.js.org/docs/Troubleshooting.html

This is a place to share common problems and solutions to them.
The examples use React, but you should still find them useful if you use something else.

Nothing happens when I dispatch an action

Sometimes, you are trying to dispatch an action, but your view does not update. Why does this happen?
There may be several reasons for this.

Never mutate reducer arguments

It is tempting to modify the `state` or `action` passed to you by Redux. Don't do this!

lices
checkerSlice.js
formatSlice.js
userSlice.js

```
import userReducer from './slices/userSlice';

export default configureStore({
  reducer: {
    formatReducer: formatReducer ,
    checkerR:      checkerReducer ,
    userR:         userReducer ,
    },
})
```

Redux in practice (2/4)

- Use react-redux tools
  - ***Provider*** to deliver service to all components

```
import { Provider } from 'react-redux';
import store from './store';
…
```

index.js

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

Redux in practice (3/4)

```
import { useDispatch, useSelector } from "react-redux";
//Actions of the formatSlice are imported
import { format_txt, reset_txt } from './slices/formatSlice';

export const App =(props) => {
  let formatted_txt = useSelector(state =>
          state.formatReducer.txt);
  const [title, setTitle] = useState(props.title);
  const dispatch = useDispatch();
  function handleChangeTitle(value){
      const txt= value
      setTitle(txt);
      dispatch(format_txt(txt));
  }
return (
    <div className="App">
      <h1> this is my first React Component</h1>
      <label htmlFor="titleInput">Title </label>
          <input type="text" id="titleInput"
              onChange={e =>handleChangeTitle(e.target.value)}
              value={title} />
          <h3>Original txt: {title}</h3>
          <h3>Formatted txt:{formatted_txt}</h3>
    </div>
  );
}
```

- ***useDispatch*** to *connect* to the store (Provide needed) and to *dispatch* action

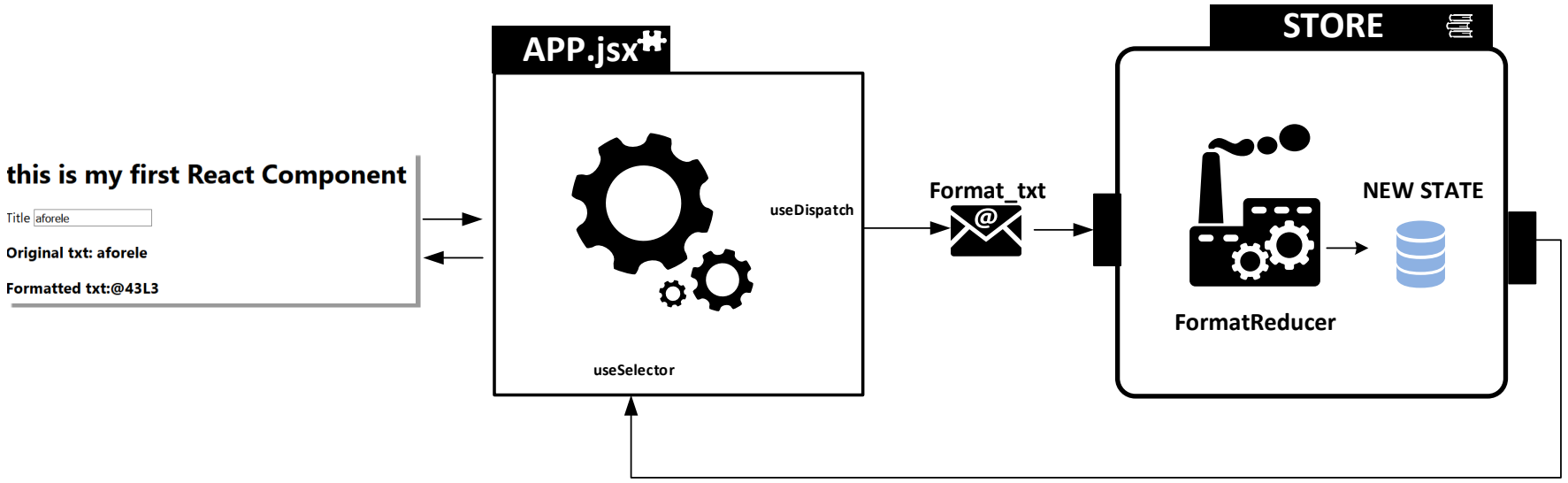
Redux in practice (3/4)

```
import { useDispatch, useSelector } from "react-redux";
//Actions of the formatSlice are imported
import { format_txt, reset_txt } from './slices/formatSlice';

let formatted_txt = useSelector(state =>
        state.formatReducer.txt);
  const [title, setTitle] = useState(props.title);
  const dispatch = useDispatch();

  function handleChangeTitle(value){
      const txt= value
      setTitle(txt);
      dispatch(format_txt(txt));
    }
 return (
    <div className="App">
      <h1> this is my first React Component</h1>
      <label htmlFor="titleInput">Title </label>
          <input type="text" id="titleInput"
              onChange={e =>handleChangeTitle(e.target.value)}
              value={title} />
          <h3>Original txt: {title}</h3>
<h3>Formatted txt:{formatted_txt}</h3 >
    </div>
  );
}
```

- ***useDispatch*** to *connect* to the store (Provide needed) and to *dispatch* action
- ***useSelector*** to *connect* to the store (Provide needed) and to *subscribe* to modification monitoring

Redux in practice

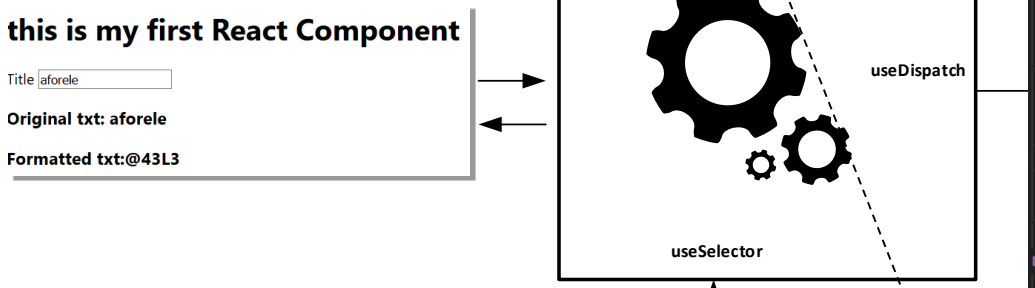
- Simple Example A



Example available at:
https://gitlab.com/js-asi2/simple-hooks

Redux in practice

- Simple Example A

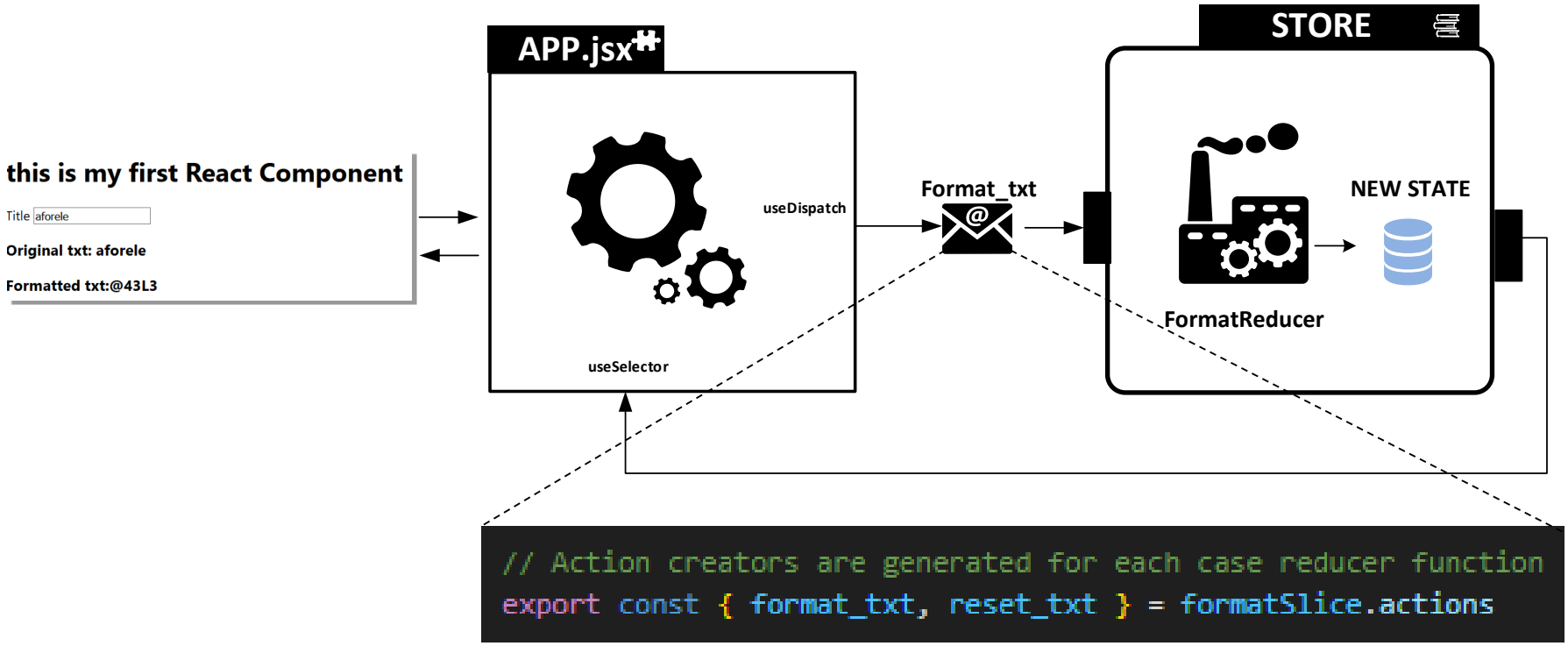



```jsx
import React, {useState} from 'react';
//Use to connect to the store
import { useDispatch, useSelector } from "react-redux";
// reducers of the formatSlice are imported
import { format_txt, reset_txt } from './slices/formatSlice';

export const App =(props) => {
  //Get information from the reducer
  let formatted_txt = useSelector(state => state.formatReducer.txt);
  //Create a local state
  const [title, setTitle] = useState(props.title);
  //Connect to the store
  const dispatch = useDispatch();

  function handleChangeTitle(e){
    const mytxt= e.target.value
    //modify the locate stat txt
    setTitle(mytxt);
    //call a reducer with associated action here parameter is {txt:mytxt}
    dispatch(format_txt({txt:mytxt}));
  }

return (
    <div className="App">
      <h1> this is my first React Component</h1>
      <label htmlFor="titleInput">Title </label>
          <input type="text" id="titleInput"
              onChange={handleChangeTitle}
              value={title}
          />
          <h3>Original txt: {title}</h3>
          <h3>Formatted txt:{formatted_txt}</h3>
    </div>
  );
}
```

Example available at:
https://gitlab.com/js-asi2/simple-hooks

Redux in practice

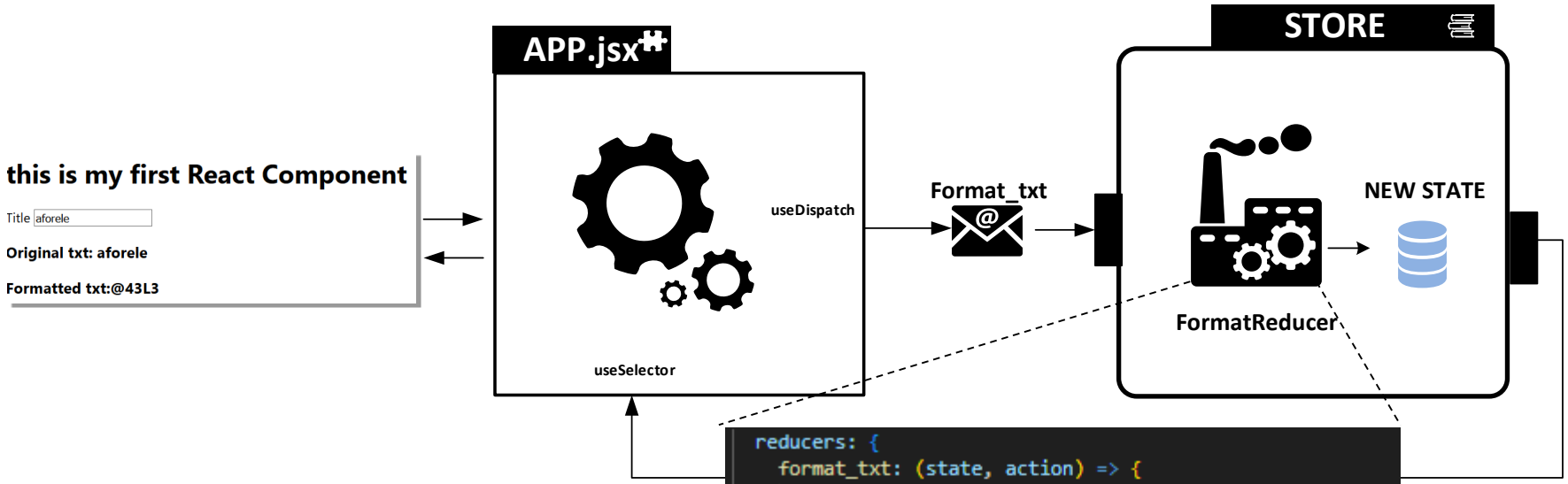
- Simple Example A



Example available at:
https://gitlab.com/js-asi2/simple-hooks

Redux in practice

- Simple Example A

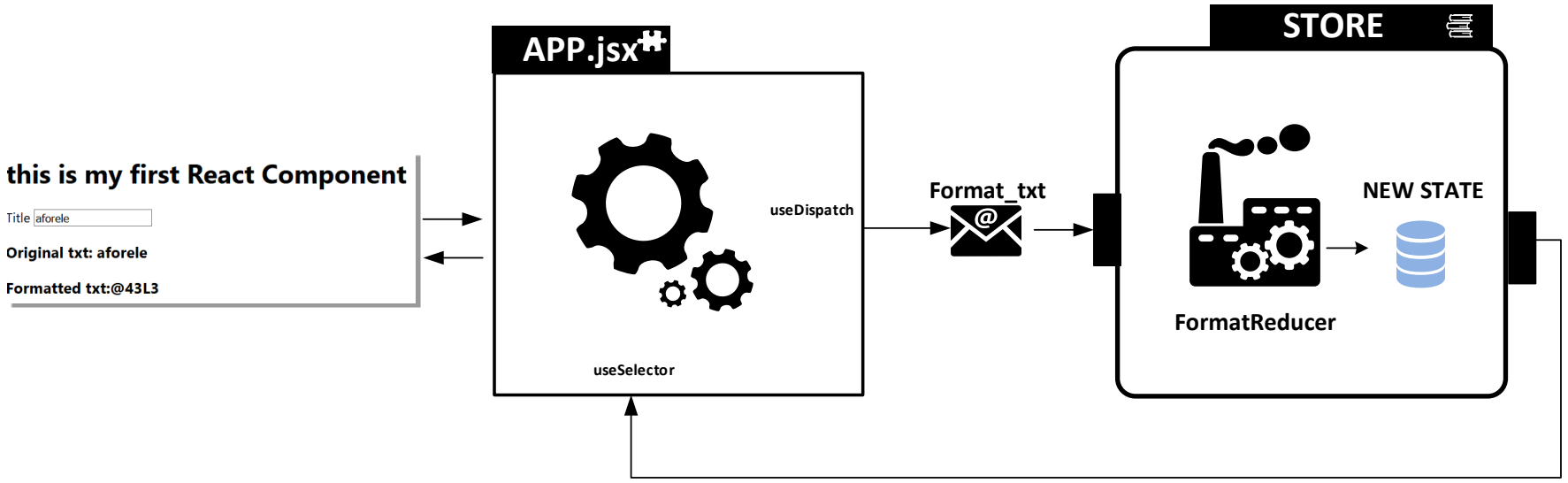



```
reducers: {
  format_txt: (state, action) => {
    // Here to access to the parameter content (action),
    // you need to target the .payload of the parameter
    let current_txt = action.payload.txt
    current_txt= current_txt.toUpperCase()
    current_txt= current_txt.replaceAll('A','@')
    current_txt= current_txt.replaceAll('FOR','4')
    current_txt= current_txt.replaceAll('E','3')
    state.txt = current_txt
  },
  reset_txt: (state) => {
  // additional parameter is not mandatory,
  // only the previsous state is needed
```

Example available at:
https://gitlab.com/js-asi2/simple-hooks

Redux in practice

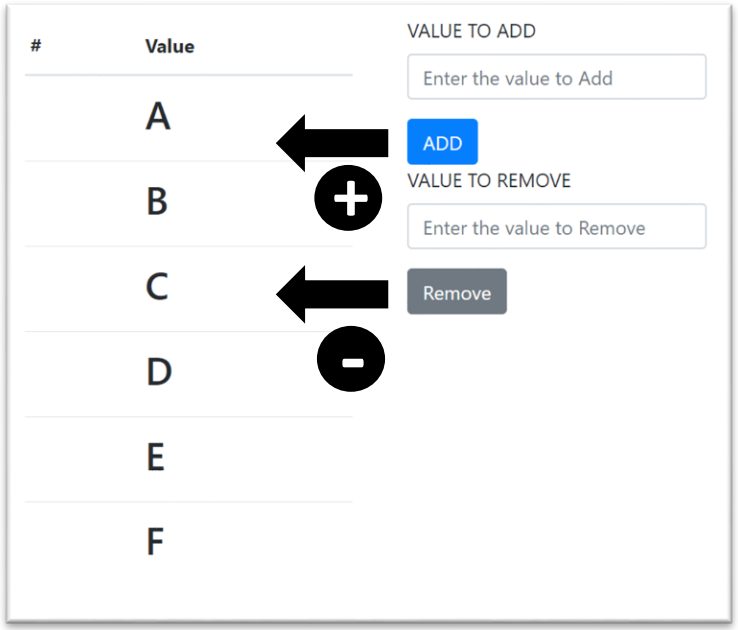
- Simple Example A



Example available at:
https://gitlab.com/js-asi2/simple-hooks

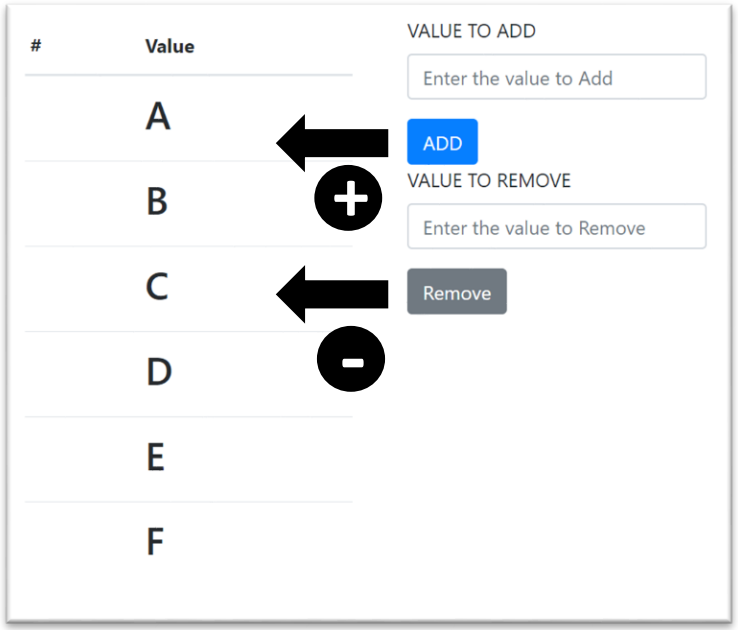
Redux in practice

- Simple Example B



Redux in practice

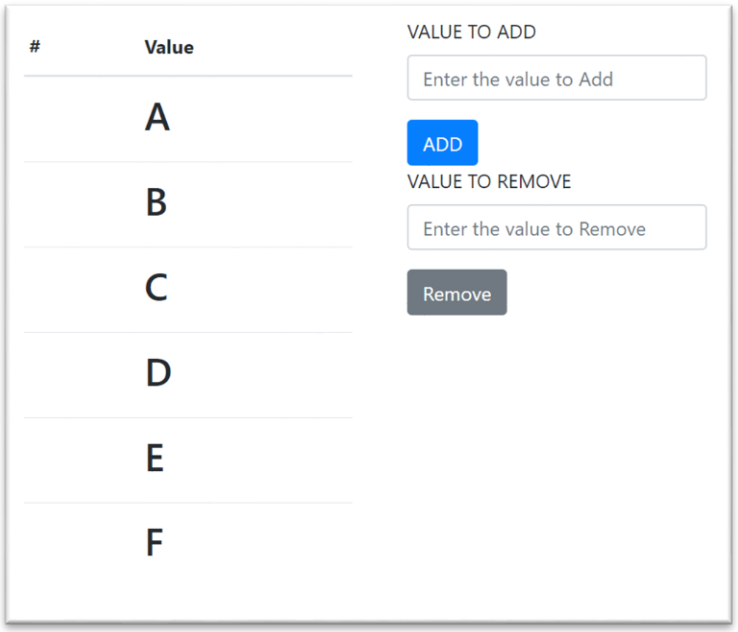
- Simple Example B

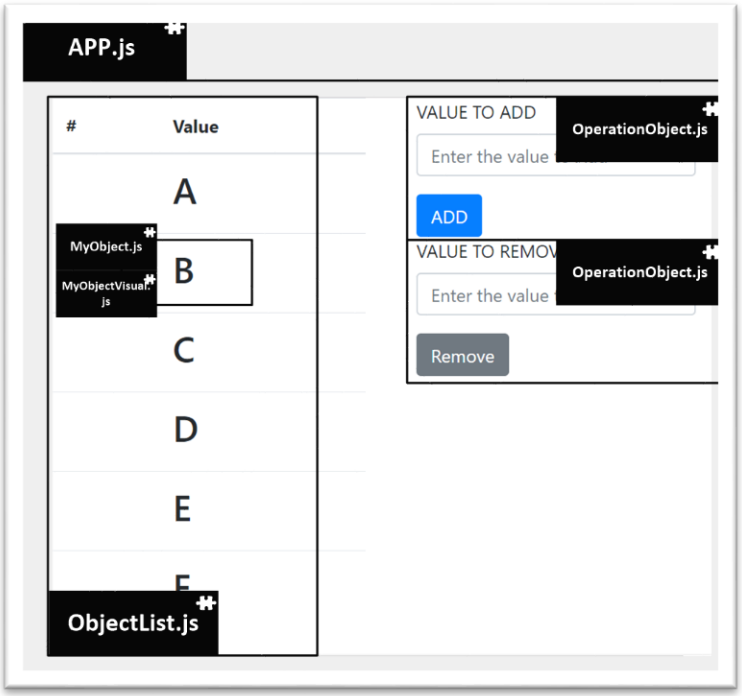


Example available at:
https://github.com/jacques-saraydaryan/front-end-react.js.git
redux-simple-example

Redux in practice

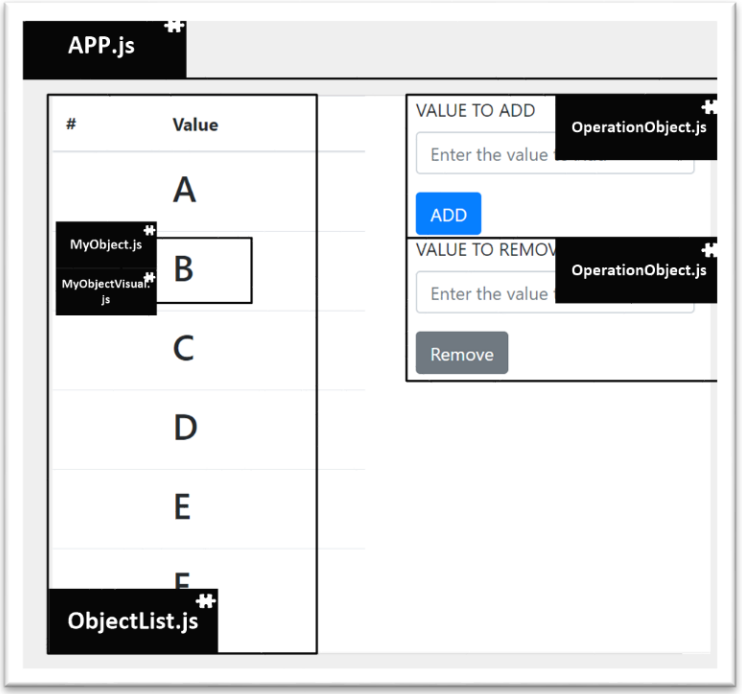
- Simple Example B

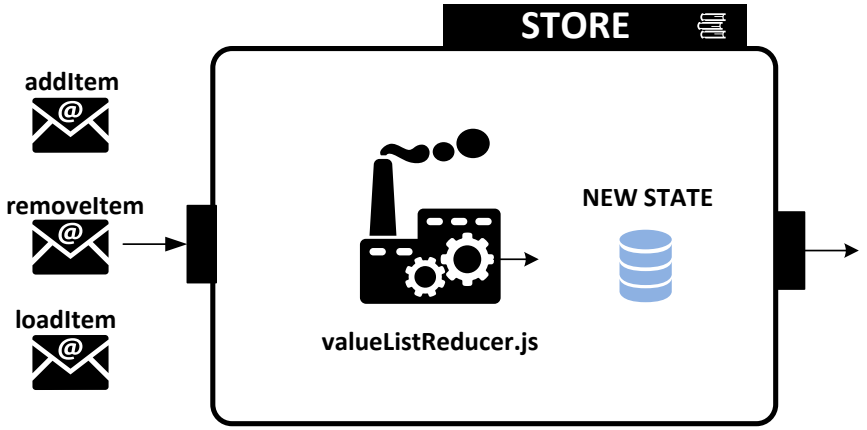




Redux in practice

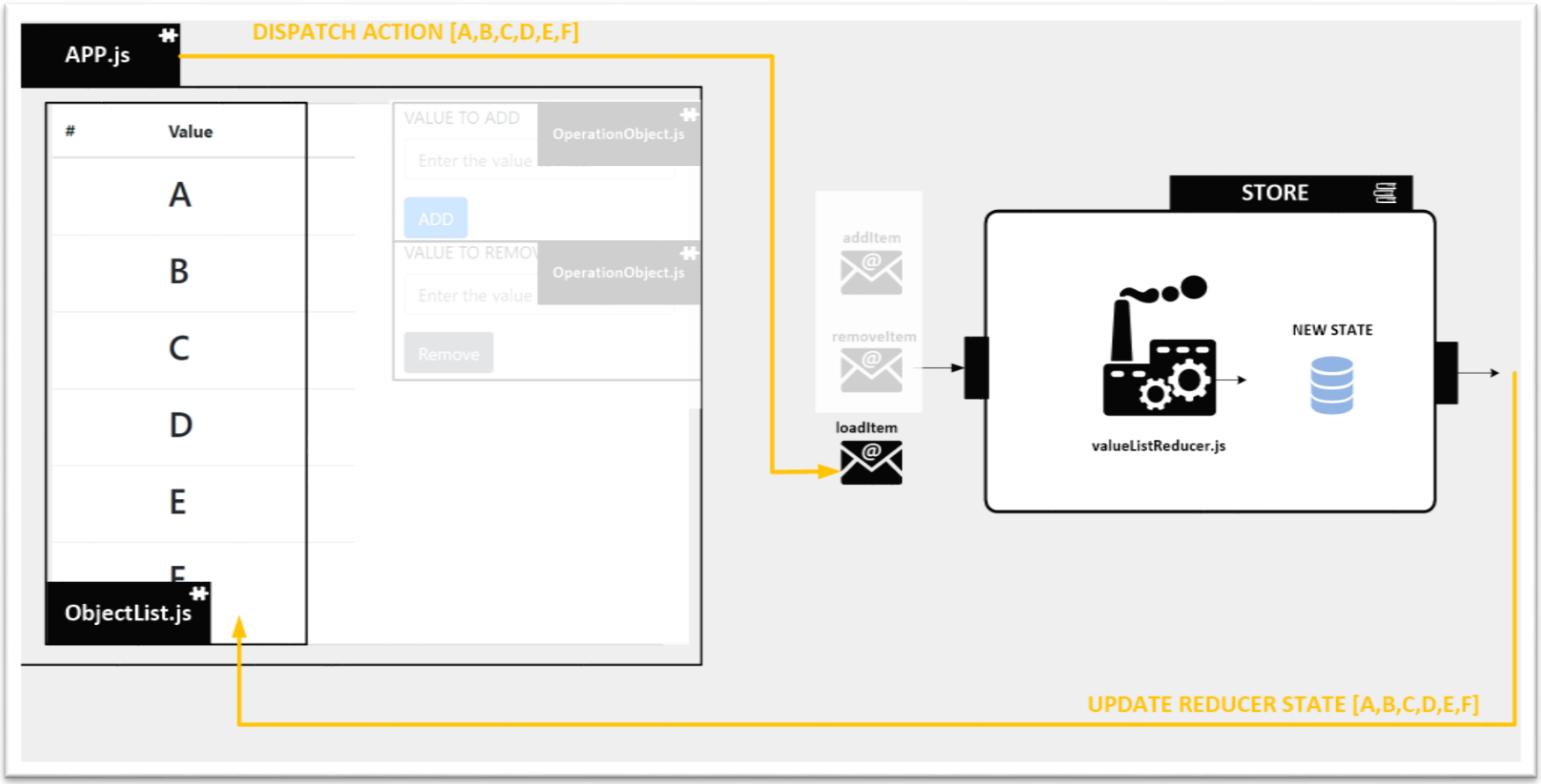
- Simple Example B





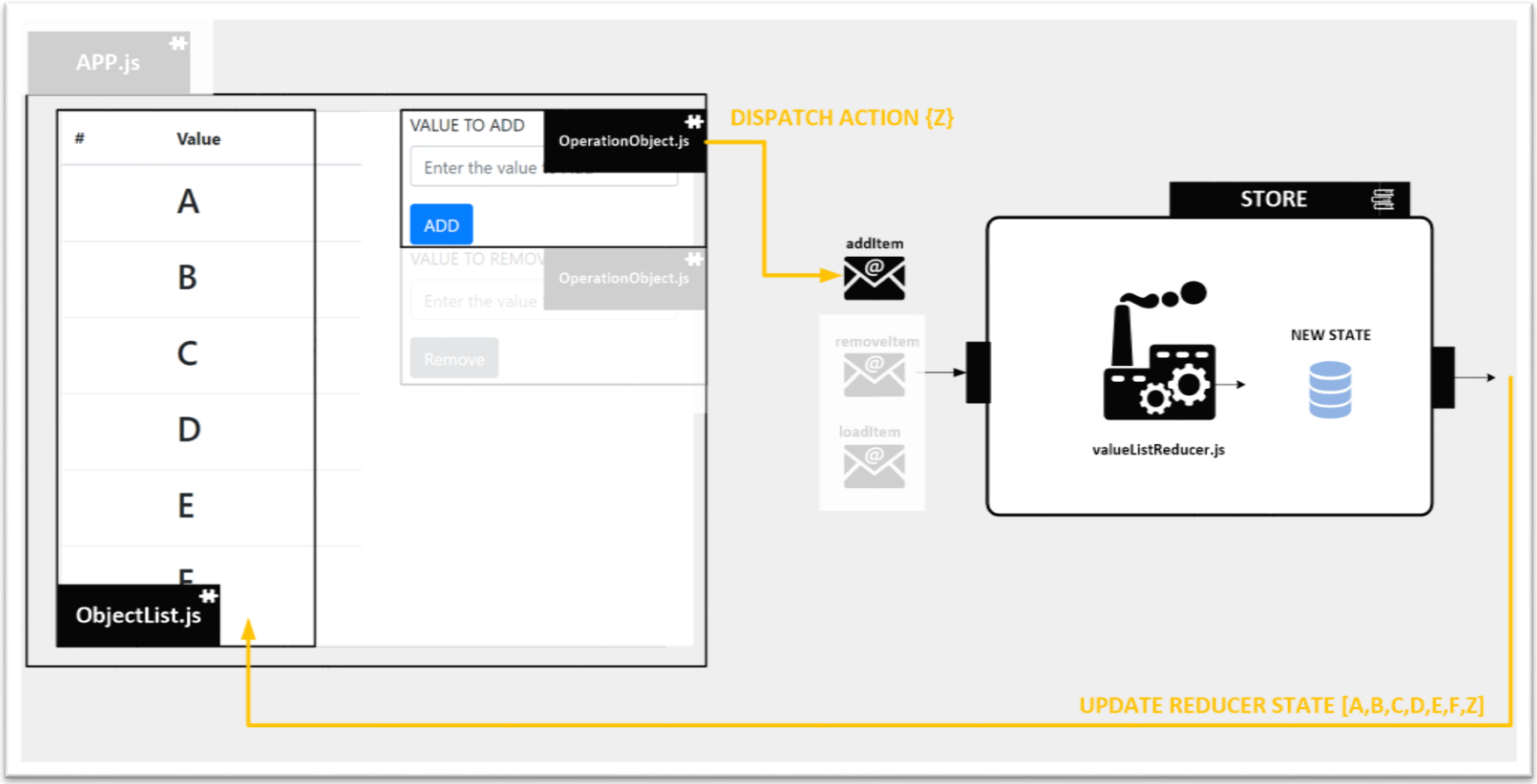
Redux in practice

- Simple Example B



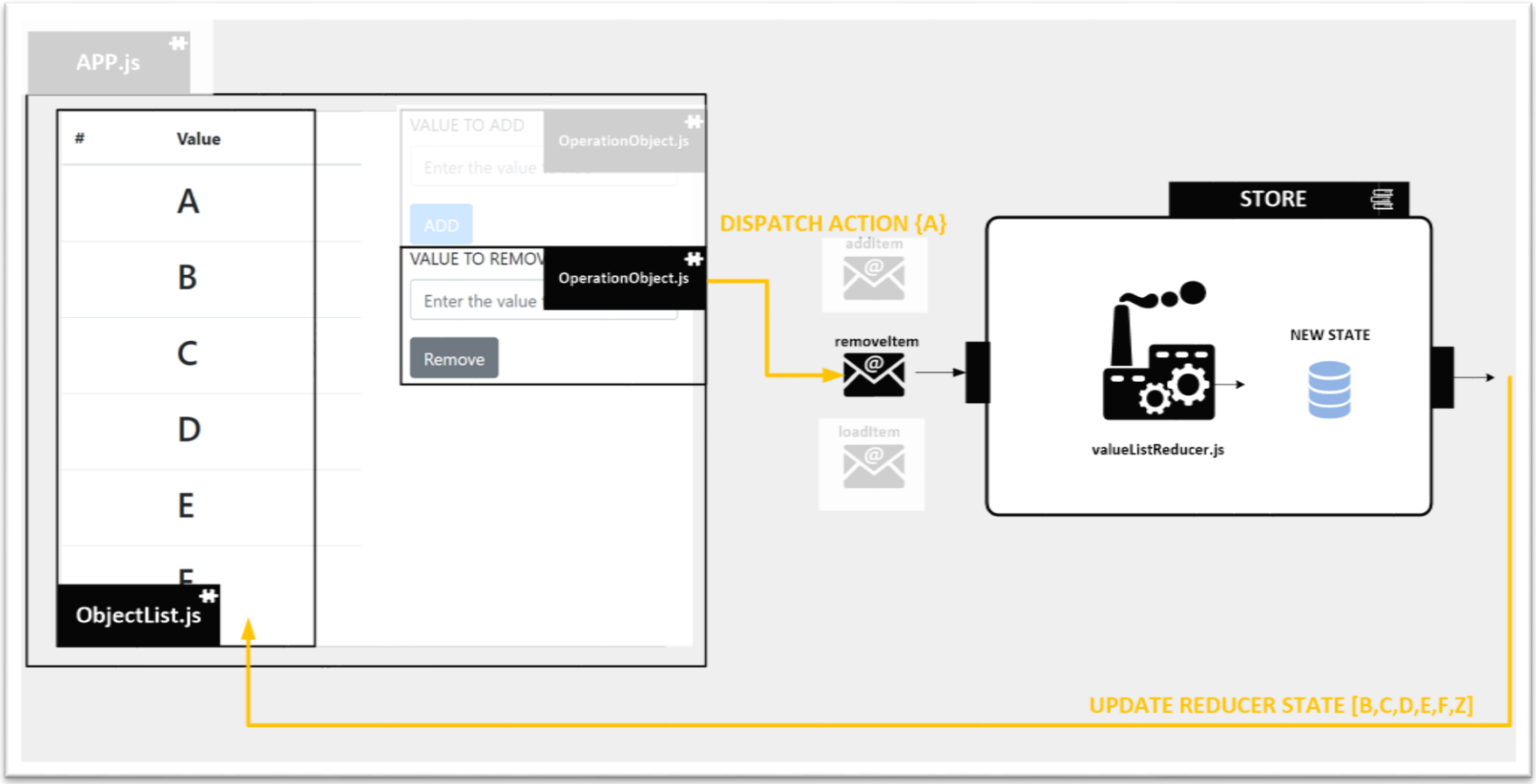
Redux in practice

- Simple Example B



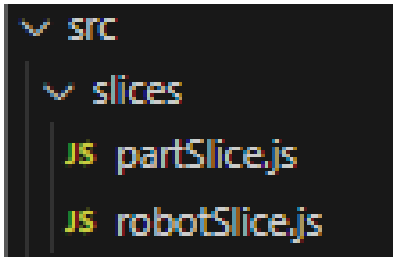
Redux in practice

- Simple Example B



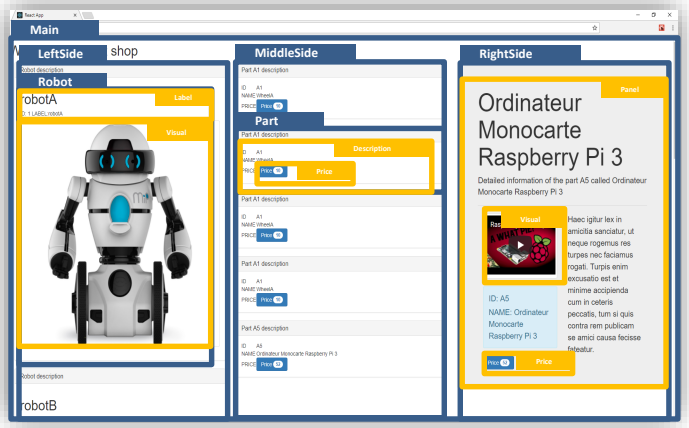
It is your turn !

- Create 2 different Slices
  - 1 modifying the selected Robot obj
  - 1 modifying the selected Part obj



It is your turn !

- Modifying the previous project so as to
  - Dispatch selected robot in the ***Robot*** component
  - Subscribe to store and update list of parts in the ***MiddleSide*** component
  - Dispatch selected part in the ***Part*** component
  - Subscribe to store and update Part in the ***RightSide*** component



References

References

- React.js and Flux
  http://soat.developpez.com/tutoriels/javascript/architecture-flux-avec-react/
  https://www.tutorialspoint.com/reactjs/reactjs_using_flux.htm
  https://medium.com/@jetupper/hello-react-js-b87c63526e3a
  DEVOX France: https://www.youtube.com/watch?v=IFM8krjbKmQ

- React.js and Redux
  http://www.troispointzero.fr/2016/03/reactjs-redux-pour-des-lendemains-qui-chantent-premiere-partie/
  https://www.codementor.io/mz026/getting-started-with-react-redux-an-intro-8r6kurcxf
  http://www.sohamkamani.com/blog/2017/03/31/react-redux-connect-explained/

- Tutorials
  https://github.com/HurricaneJames/dex/blob/master/doc/Building%20Components%20with%20React.js%20and%20Flux.md
  https://github.com/react-bootcamp/react-workshop

- Course tutorial
  https://github.com/jacques-saraydaryan/front-end-react.js

Jacques Saraydaryan

Jacques.saraydaryan@cpe.fr