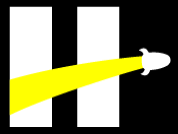
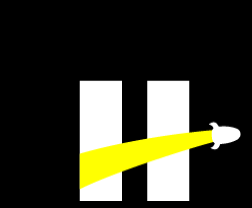
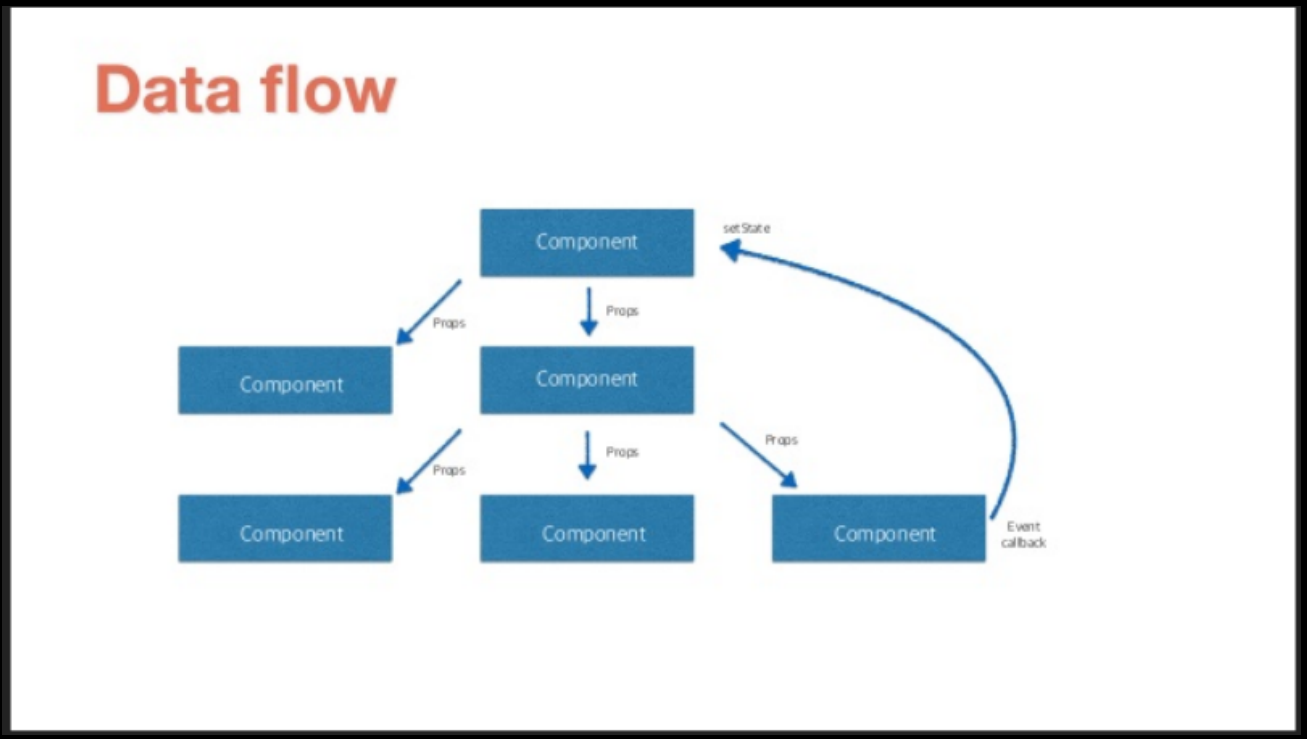

HENRY



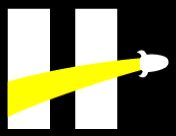
Redux



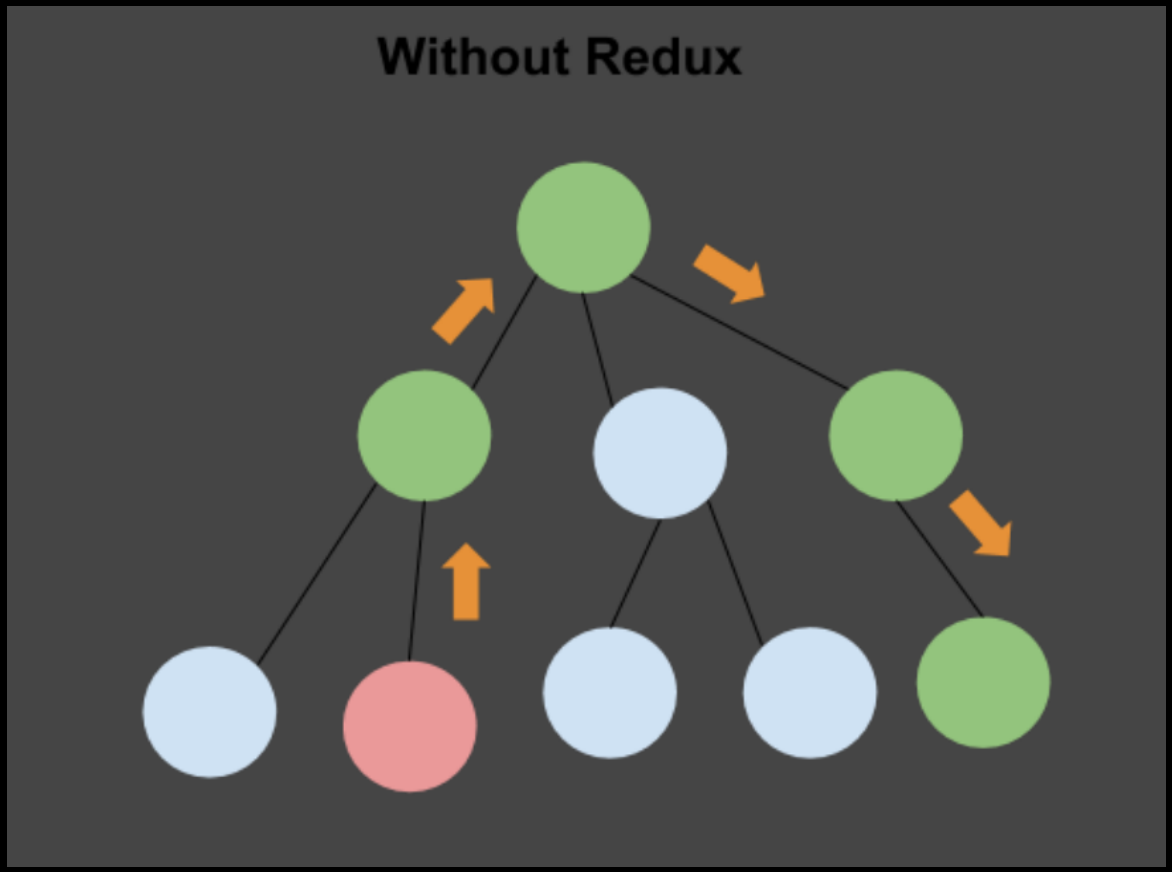
Redux



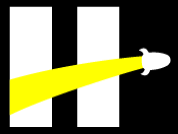
¿Se acuerdan del one-way Data Flow?



Redux

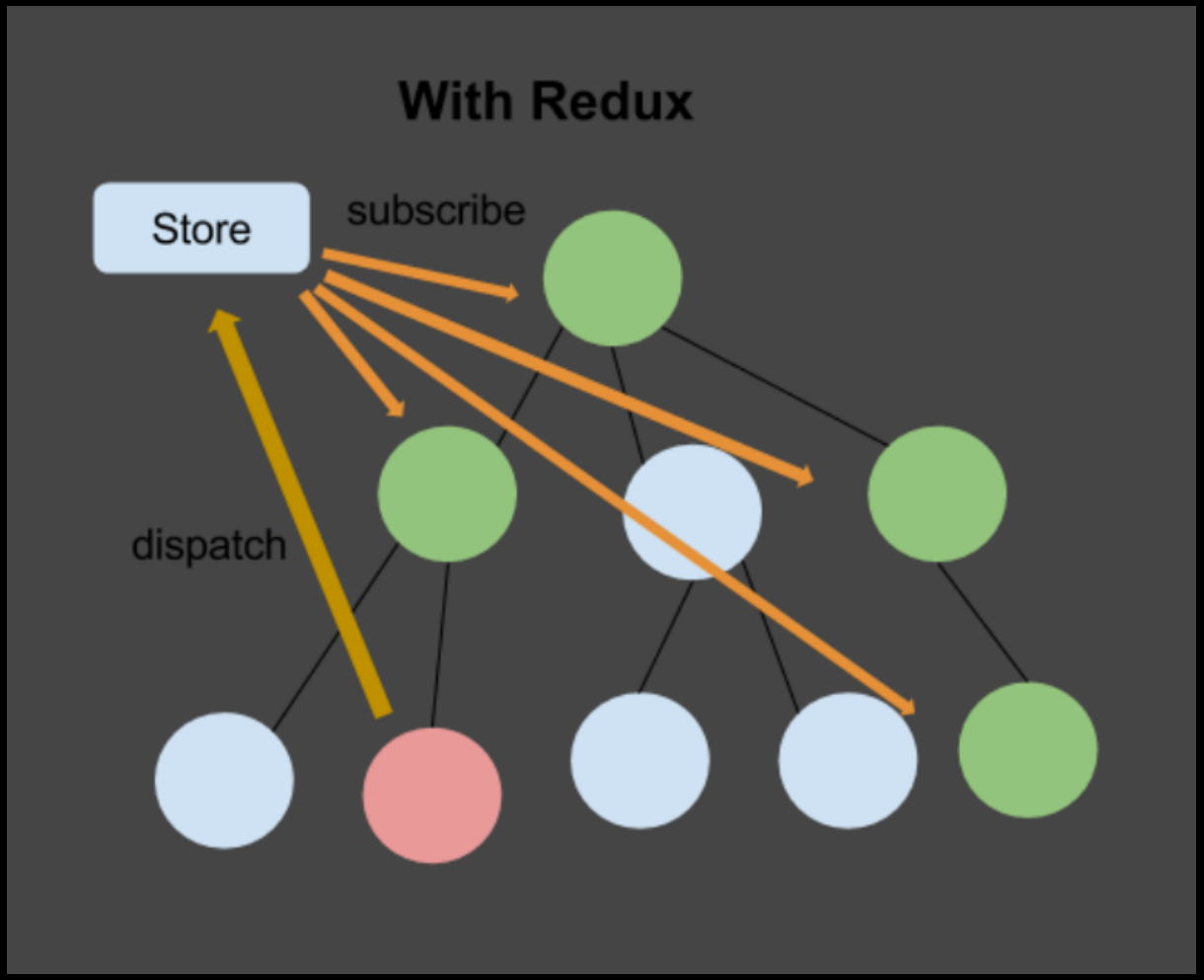


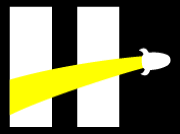
Podría generar ciertos problemas.



Redux

Redux



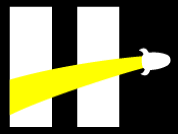


Los tres principios de Redux

Single source of truth

The state of your whole application is stored in an object tree within a single store.

```
console.log(store.getState())

/* Prints
{
  visibilityFilter: 'SHOW_ALL',
  todos: [
    {
      text: 'Consider using Redux',
      completed: true,
    },
    {
      text: 'Keep all state in a single tree',
      completed: false
    }
  ]
}
*/
```

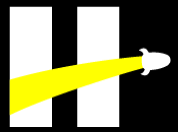


Los tres principios de Redux

State is read-only

The only way to change the state is to emit an action, an object describing what happened.

```
1
2  store.dispatch({
3    type: 'COMPLETE_TODO',
4    index: 1
5  })
6
7  store.dispatch({
8    type: 'SET_VISIBILITY_FILTER',
9    filter: 'SHOW_COMPLETED'
10 })
```



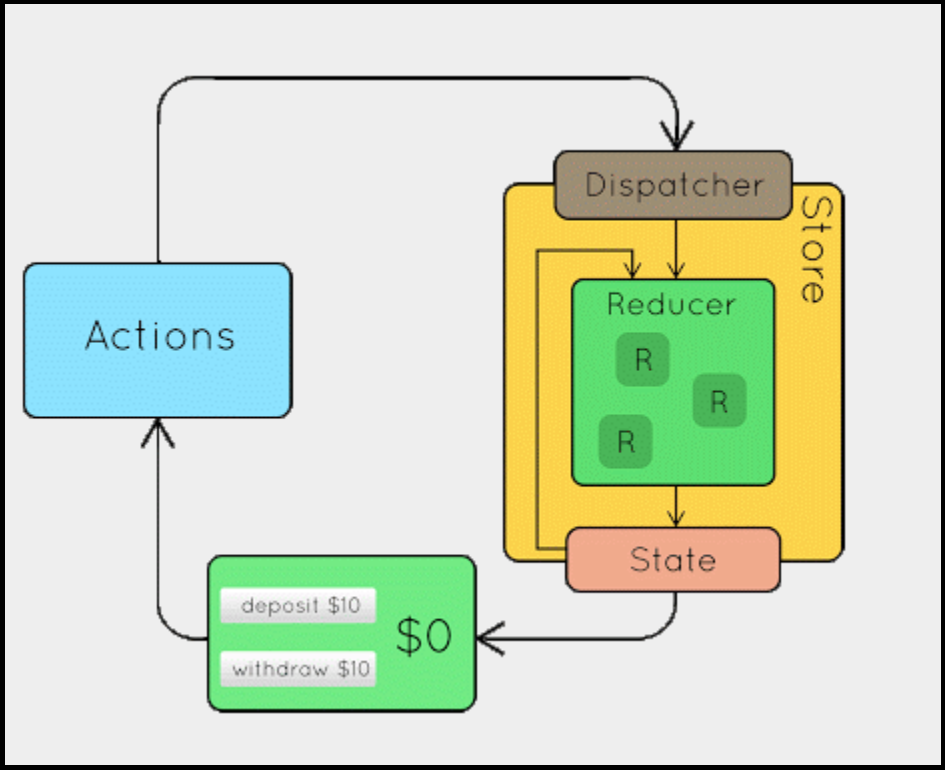
Los tres principios de Redux

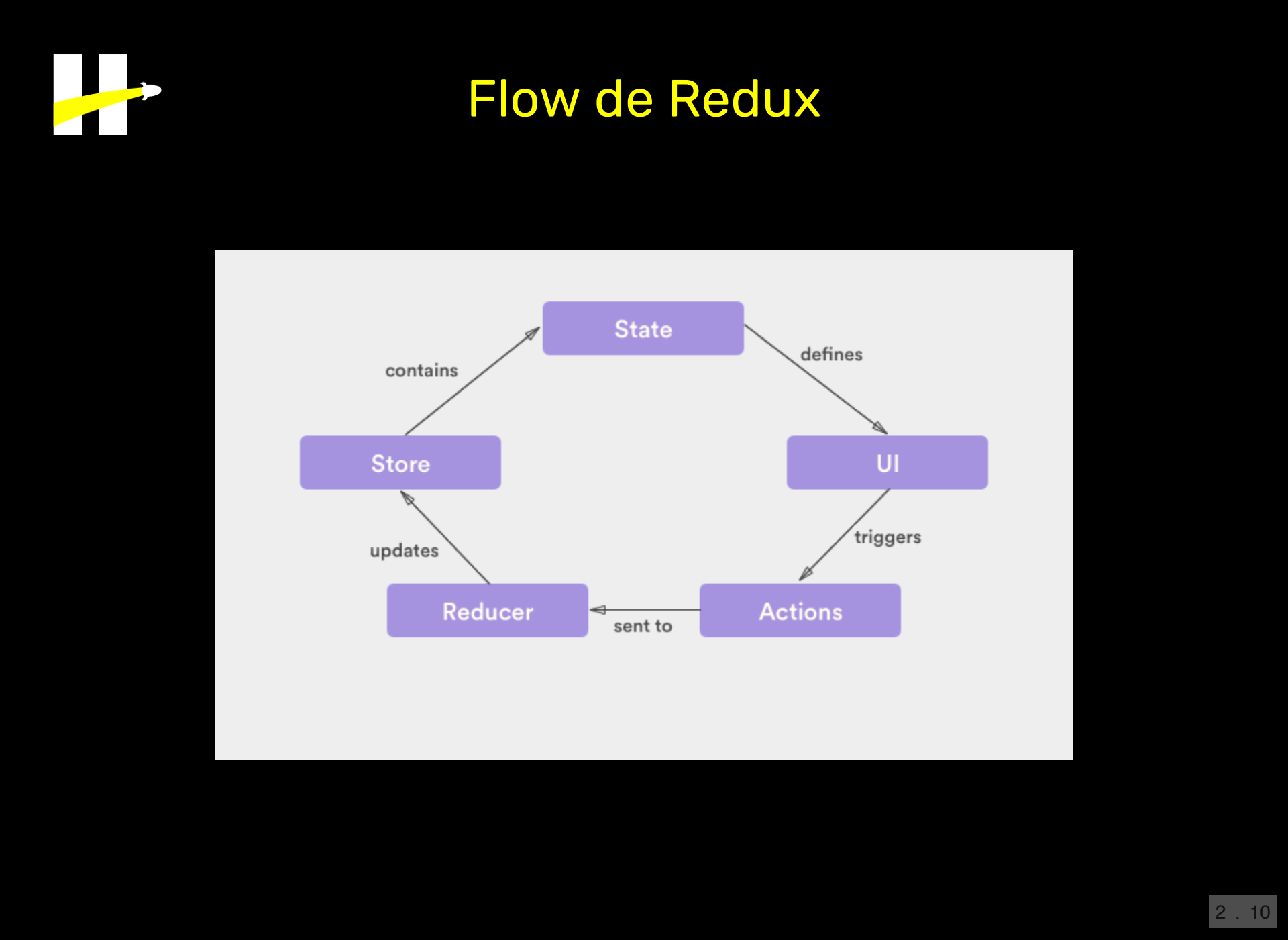
Changes are made with pure functions

To specify how the state tree is transformed by actions, you write pure reducers.

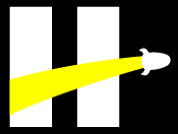
```
function visibilityFilter(state = 'SHOW_ALL', action) {
  switch (action.type) {
    case 'SET_VISIBILITY_FILTER':
      return action.filter
    default:
      return state
  }
}

function todos(state = [], action) {
  switch (action.type) {
    case 'ADD_TODO':
      return [
        ...state,
        {
          text: action.text,
          completed: false
        }
      ]
    case 'COMPLETE_TODO':
      return state.map((todo, index) => {
        if (index === action.index) {
          return Object.assign({}, todo, {
            completed: true
          })
        }
        return todo
      })
    default:
      return state
  }
}

import { combineReducers, createStore } from 'redux'
const reducer = combineReducers({ visibilityFilter, todos })
const store = createStore(reducer)
```

Flow de Redux





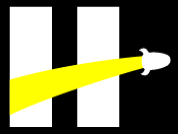
Flow de Redux



Actions

```
1 const ADD_TODO = 'ADD_TODO'
2
3 {
4   type: ADD_TODO,
5   text: 'Build my first Redux app'
6 }
```

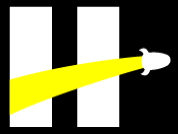
Las **acciones** son un bloque de información que envia datos desde tu aplicación a tu store. Son la *única* fuente de información para el store. Las envias al store usando store.dispatch()



Actions Creators

```
function addTodo(text) {
  return {
    type: ADD_TODO,
    text
  }
}
```

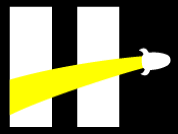
Los **creadores de acciones** son exactamente eso—funciones que crean acciones.



Dispatch()

```
1 import * as actions from'./actionsCreators';
2
3 store.dispatch(actions.increment());
4 store.dispatch(actions.addComment());
5 store.dispatch(actions.removeComment());
```

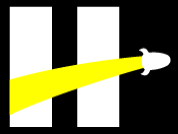
La función *dispatch* es la encargada de *enviar* las acciones al store.



Reducers

Las acciones describen que *algo pasó*, pero no especifican cómo cambió el estado de la aplicación en respuesta. Esto es trabajo de los reducers.

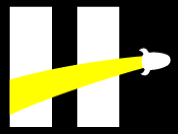
```
const addContact = (state, action) => {
  switch (action.type) {
    case 'NEW_CONTACT':
    return {
        ...state, contacts:
        [...state.contacts, action.payload]
    }
    case 'UPDATE_CONTACT':
      return {
        // Handle contact update
      }
    case 'DELETE_CONTACT':
      return {
        // Handle contact delete
      }
    case 'EMPTY_CONTACT_LIST':
      return {
        // Handle contact list
      }
    default:
      return state
  }
}
```



Reducers

Cuando una aplicación es muy grande, podemos dividir nuestros reducers en archivos separados y mantenerlos completamente independientes y controlando datos específicos.

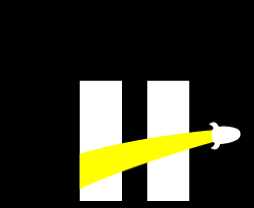
```
1 import { combineReducers } from 'redux'
2
3 const todoApp = combineReducers({
4   visibilityFilter,
5   todos
6 })
7
8 export default todoApp
```



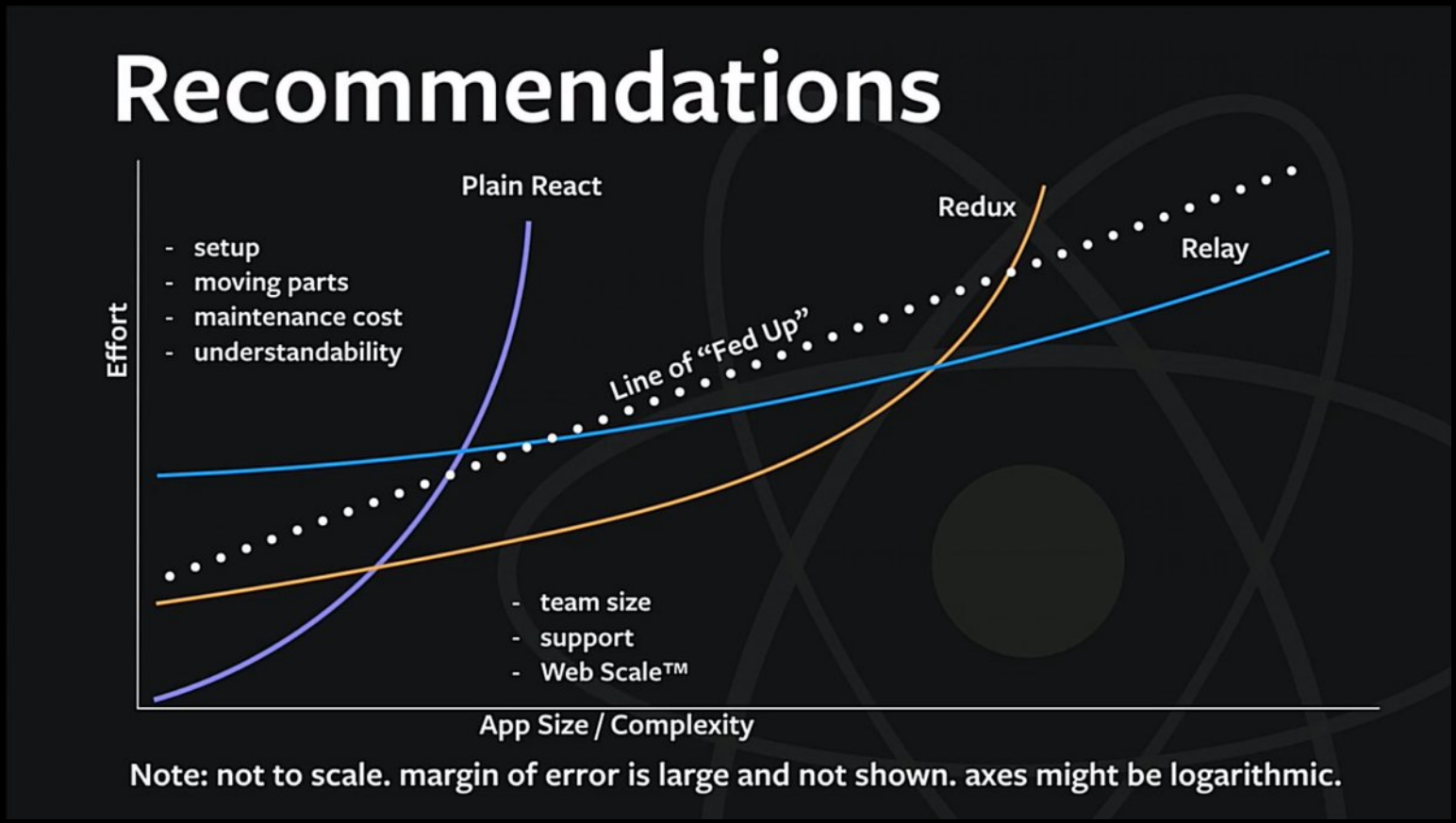
Store

- Contiene el estado de la aplicación;
- Permite el acceso al estado via getState();
- Permite que el estado sea actualizado via dispatch(action);
- Registra los *listeners* via subscribe(listener);
- Maneja la anuliación del registro de los *listeners* via el retorno de la función de subscribe(listener).

```
1 import { createStore } from 'redux'
2 import todoApp from './reducers'
3
4
5 let store = createStore(todoApp)
```



Redux


Recommendations
Plain React
Redux
Relay
- setup
- moving parts
- maintenance cost
- understandability
Effort
Line of "Fed Up"
- team size
- support
- Web Scale™
App Size / Complexity
Note: not to scale. margin of error is large and not shown. axes might be logarithmic.