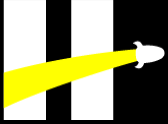


**HENRY**

A bright yellow beam of light originates from the left edge of the frame and points towards the letter 'R' in the word 'HENRY'. The beam is wider on the left and tapers as it moves to the right. The word 'HENRY' is written in a bold, black, sans-serif font. The beam appears to be shining on the 'R', which has a small white highlight on its upper curve.

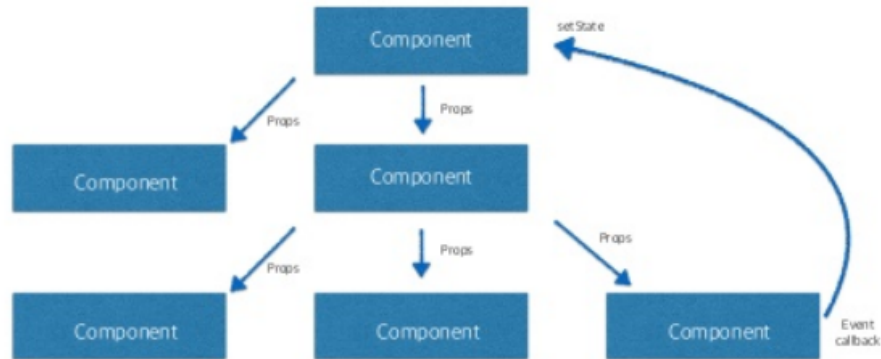


# Redux



# Redux

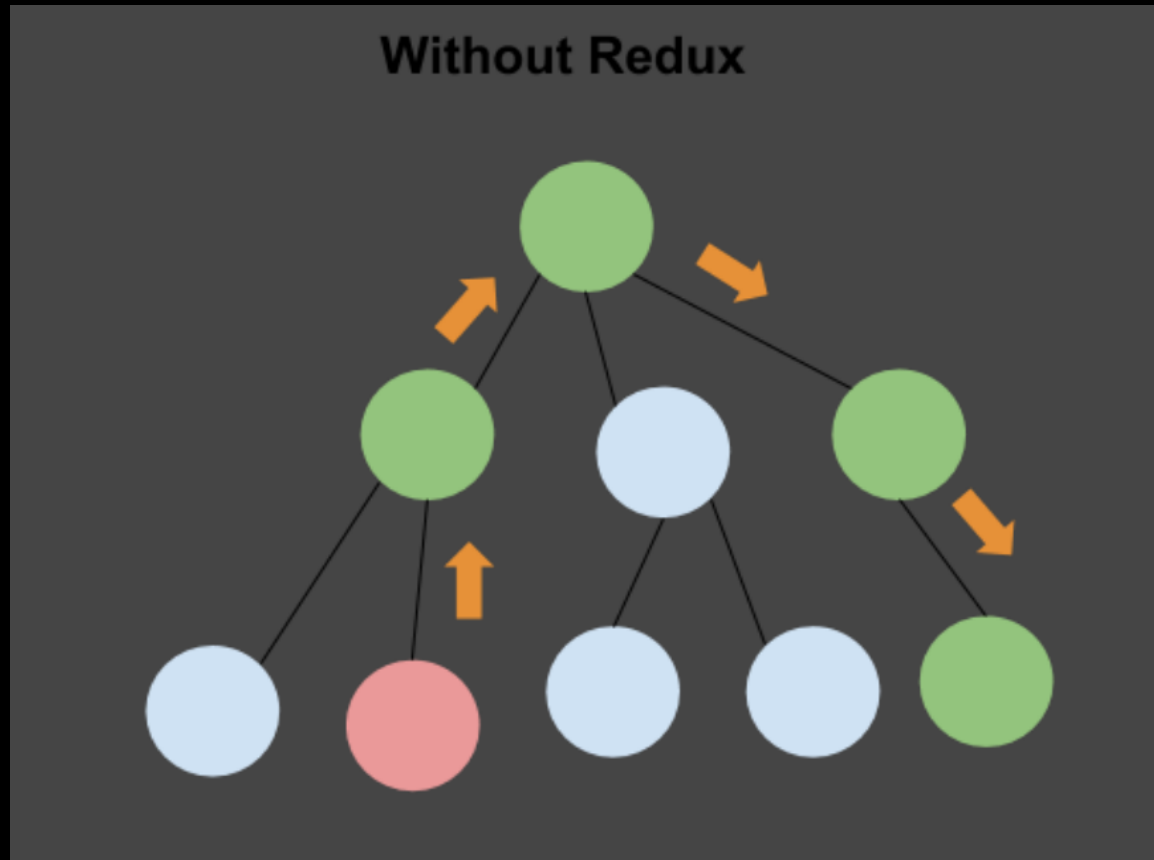
## Data flow



¿Se acuerdan del one-way Data Flow?



# Redux



Podría generar ciertos problemas.



# Redux

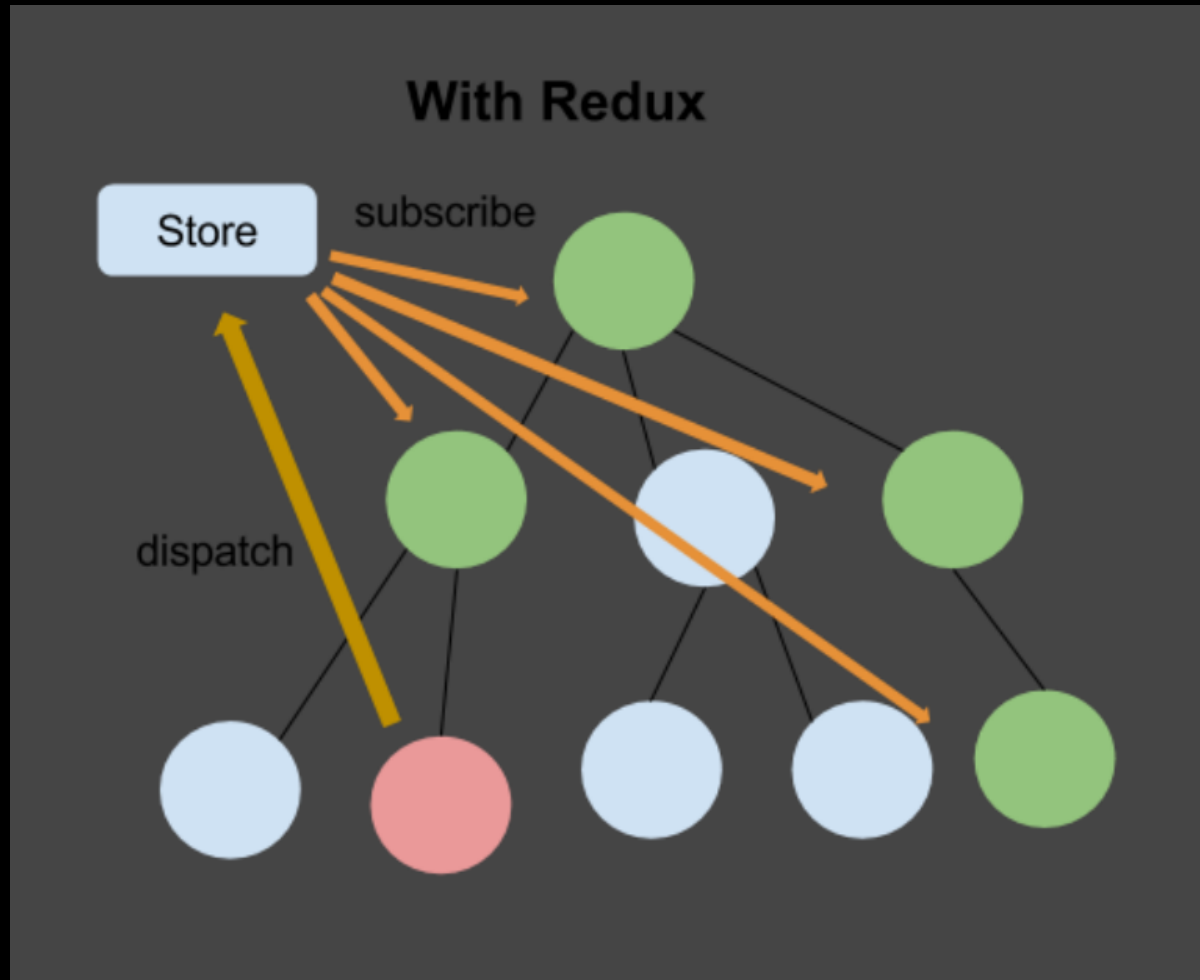


# Redux

A Predictable State Container for JS Apps



# 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**.

```
1 console.log(store.getState())
2
3 /* Prints
4 {
5   visibilityFilter: 'SHOW_ALL',
6   todos: [
7     {
8       text: 'Consider using Redux',
9       completed: true,
10    },
11    {
12      text: 'Keep all state in a single tree',
13      completed: false
14    }
15  ]
16 }
17 */
```



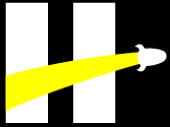
# Los tres principios de Redux

## 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

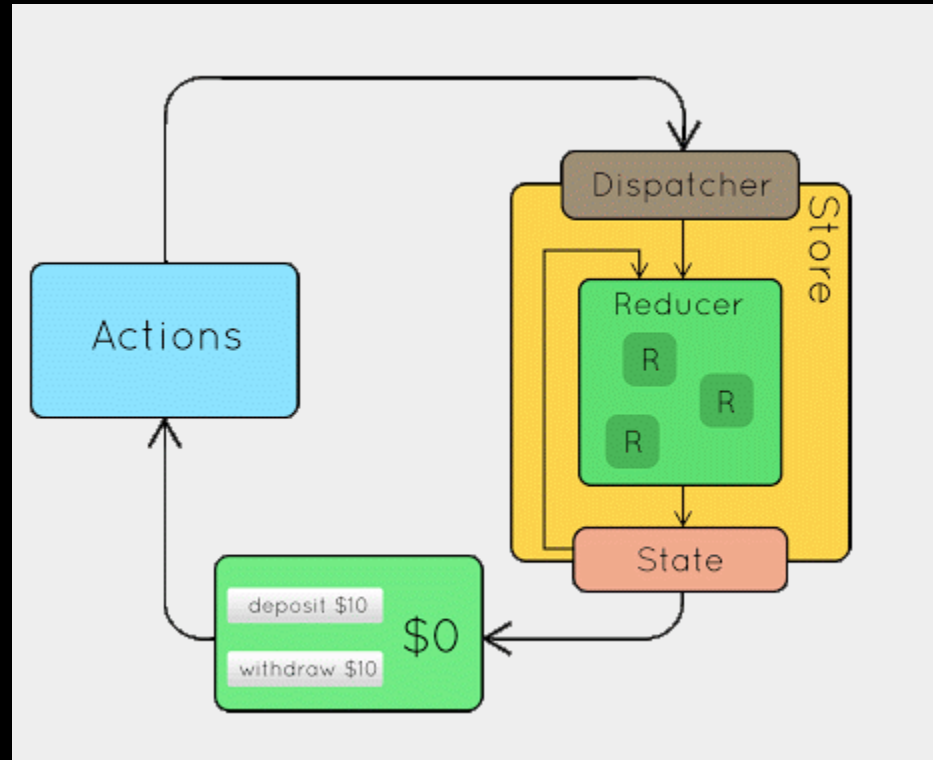
## Changes are made with pure functions

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

```
1 function visibilityFilter(state = 'SHOW_ALL', action) {
2   switch (action.type) {
3     case 'SET_VISIBILITY_FILTER':
4       return action.filter
5     default:
6       return state
7   }
8 }
9
10 function todos(state = [], action) {
11   switch (action.type) {
12     case 'ADD_TODO':
13       return [
14         ...state,
15         {
16           text: action.text,
17           completed: false
18         }
19       ]
20     case 'COMPLETE_TODO':
21       return state.map((todo, index) => {
22         if (index === action.index) {
23           return Object.assign({}, todo, {
24             completed: true
25           })
26         }
27         return todo
28       })
29     default:
30       return state
31   }
32 }
33
34 import { combineReducers, createStore } from 'redux'
35 const reducer = combineReducers({ visibilityFilter, todos })
36 const store = createStore(reducer)
```

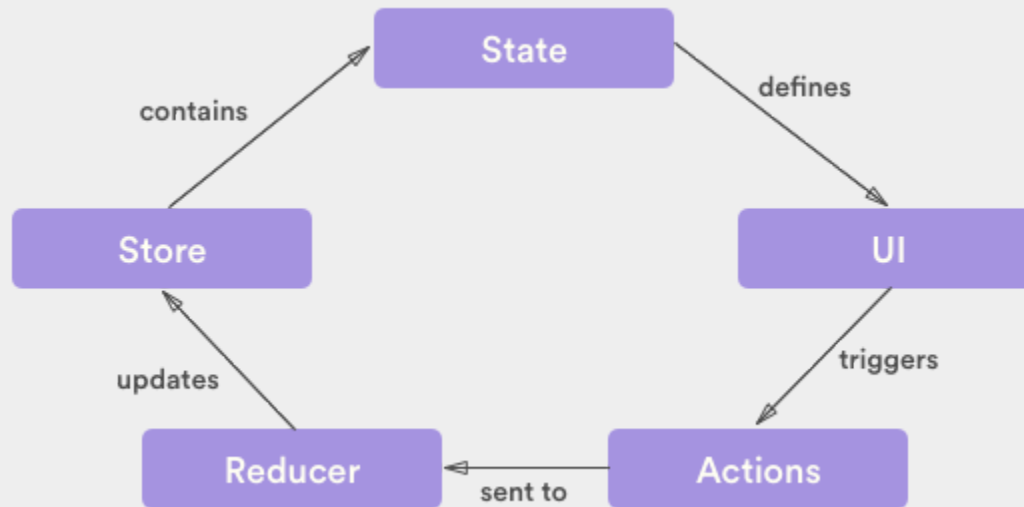


# Flow de Redux





# 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 envía datos desde tu aplicación a tu store. Son la *única* fuente de información para el store. Las envías al store usando `store.dispatch()`



# Actions Creators

```
1 function addTodo(text) {  
2   return {  
3     type: ADD_TODO,  
4     text  
5   }  
6 }
```

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.

```
1  const addContact = (state, action) => {
2    switch (action.type) {
3      case 'NEW_CONTACT':
4        return {
5          ...state, contacts:
6            [...state.contacts, action.payload]
7        }
8      case 'UPDATE_CONTACT':
9        return {
10          // Handle contact update
11        }
12      case 'DELETE_CONTACT':
13        return {
14          // Handle contact delete
15        }
16      case 'EMPTY_CONTACT_LIST':
17        return {
18          // Handle contact list
19        }
20      default:
21        return state
22    }
23 }
```



# 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

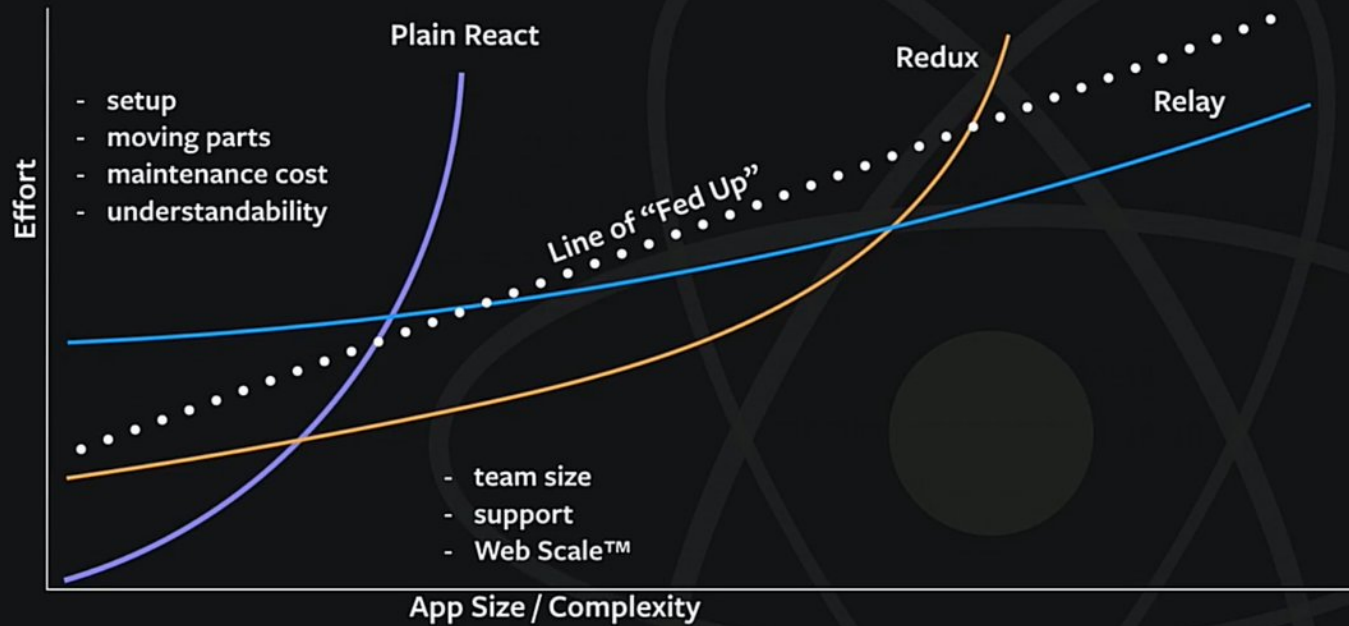
- Contiene el estado de la aplicación;
- Permite el acceso al estado via `getState()`;
- Permite que el estado sea actualizado via `dispatch(action)`;
- Registra los *listeners* via `subscribe(listener)`;
- Maneja la anulación del registro de los *listeners* via el retorno de la función de `subscribe(listener)`.

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



# Redux

## Recommendations



Note: not to scale. margin of error is large and not shown. axes might be logarithmic.