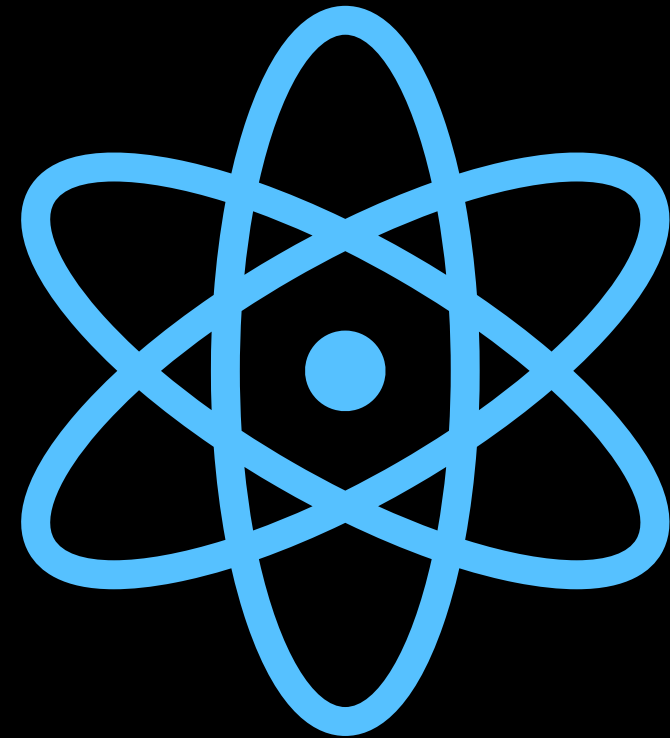




Gerson Montenegro
Consultor

amaris



Selectores & Hooks



React Foundations

¿De qué va esto?

1. **Cómputo de datos derivados**
2. **Selectores**
3. **Patrón memoize**
4. **Utilidades para el cálculo de datos**
5. **¿Cómo ocupar reselect?**
6. **Hooks**
7. **useState**
8. **useMemo**
9. **useCallback**
10. **useEffect**
11. **Hooks & Redux**

1. C3mputo de datos derivados

- 1. state
- 2. props
- 3. store
-

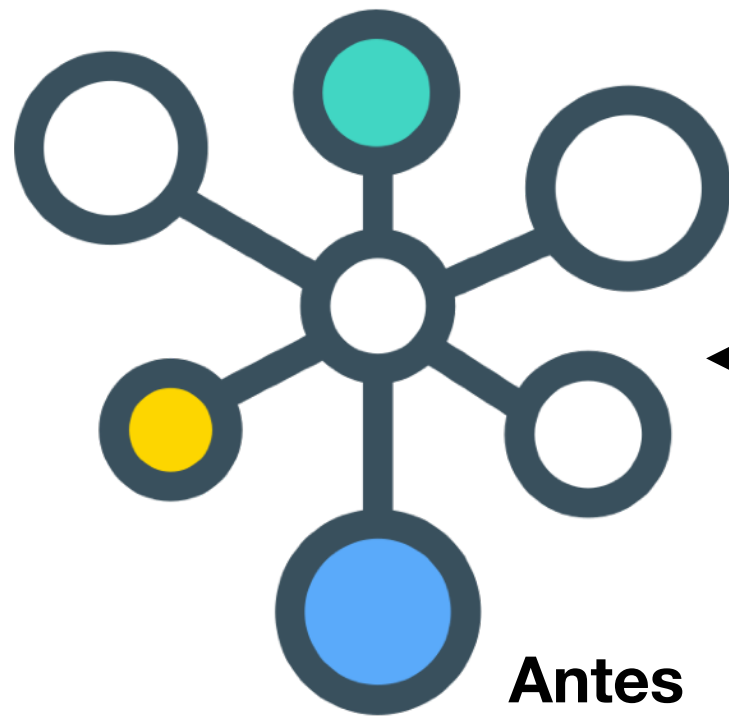


1. C mputo de datos derivados

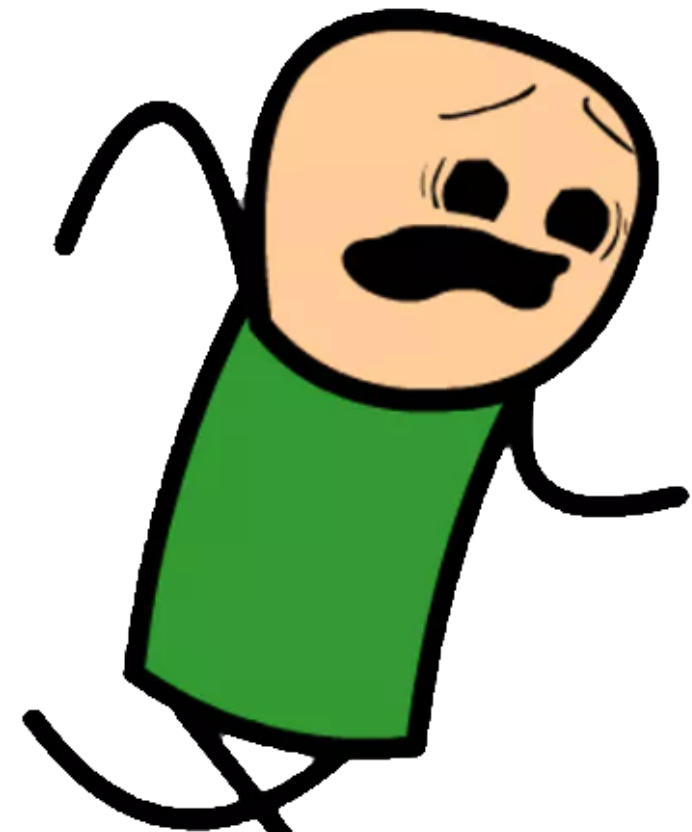
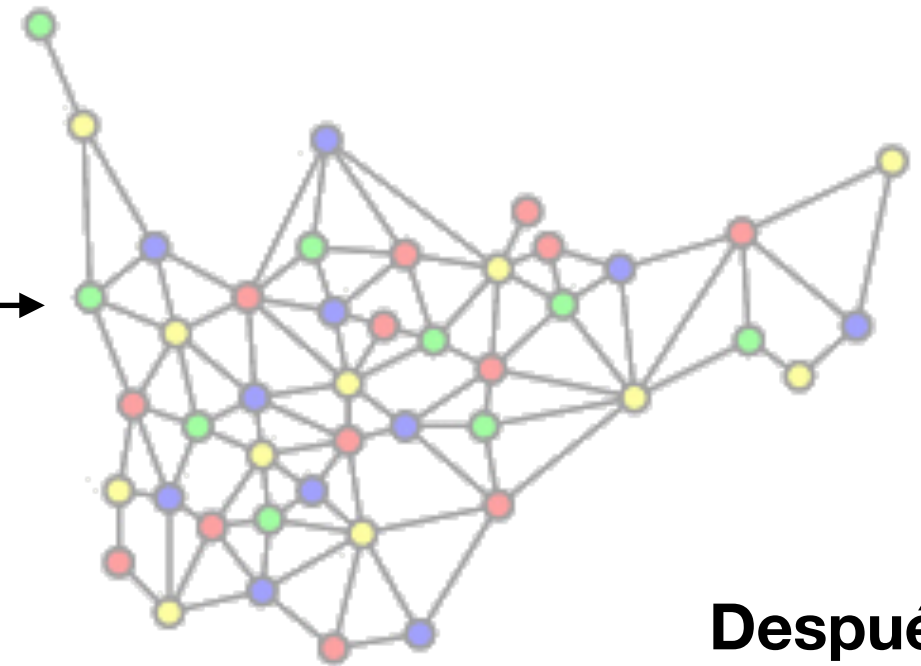


```
const filter = state => state.data.filter(item => item.likes > 100);
```

2. Selectores



state



2. Selectores



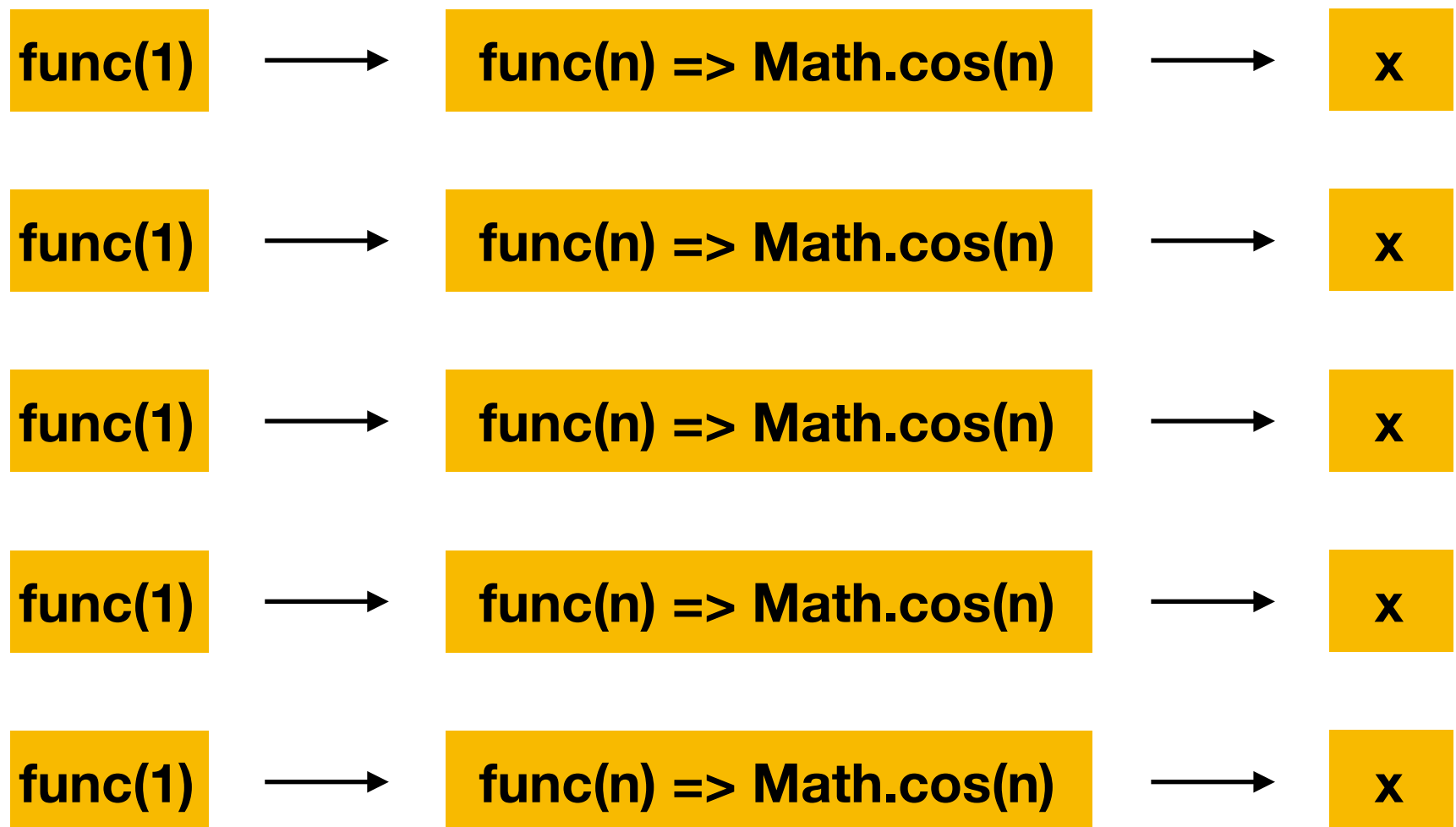
```
//sin selector
const mapStateToProps = state => {
  return { results: state.data.results }
}

//sin selector con más niveles
const mapStateToProps = state => {
  return {
    results: state.data.branch.country.city.store.results
  }
}

//con selector
const mapStateToProps = state => {
  return { results: getResults(state) }
}
```

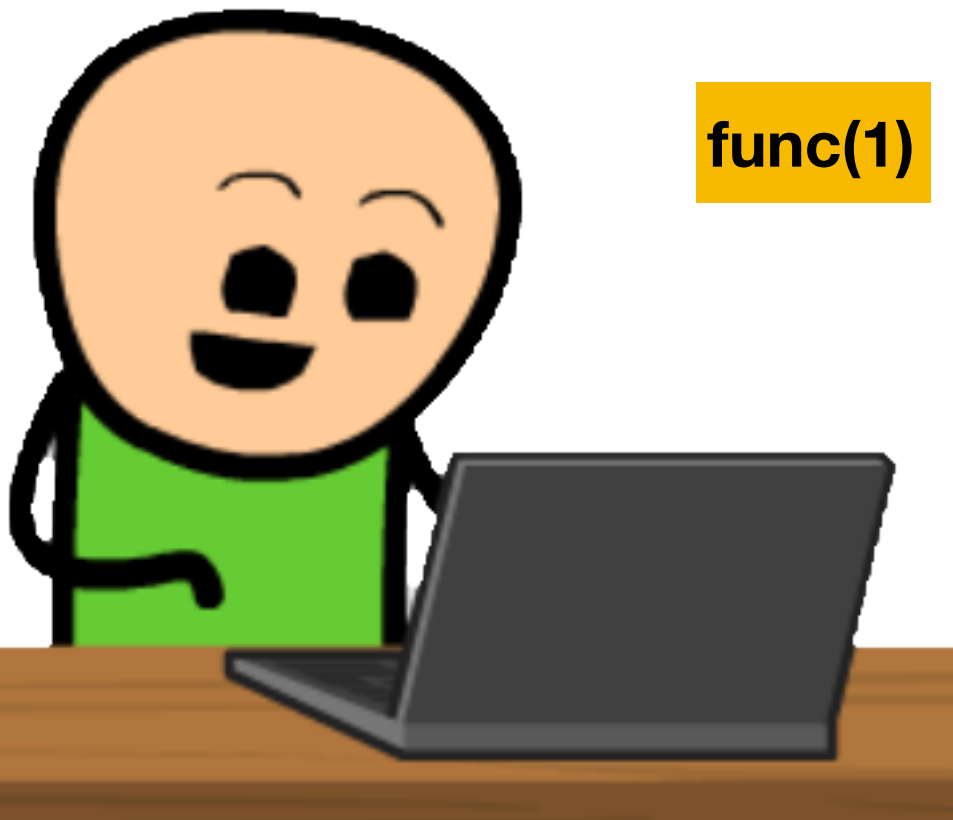
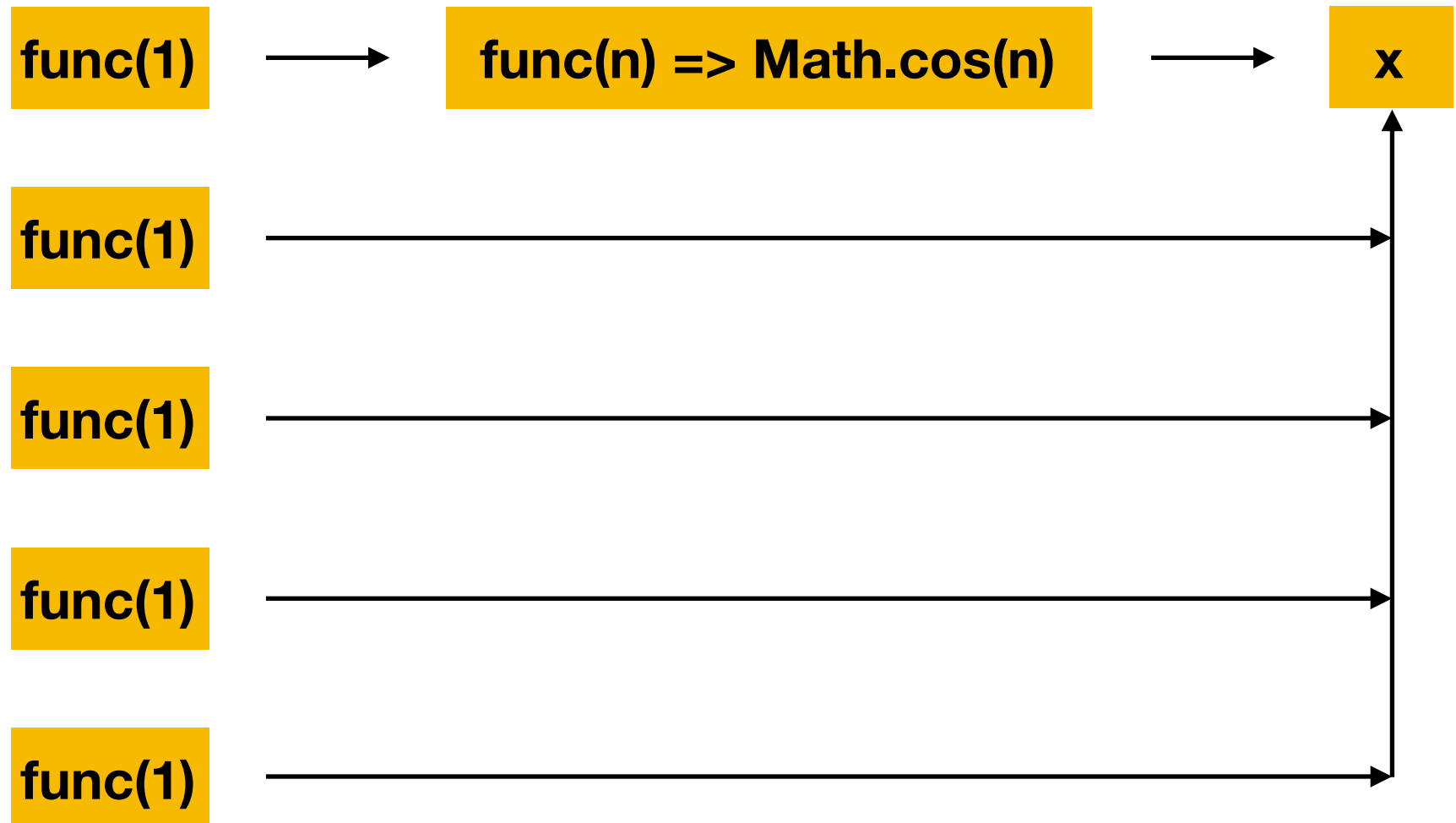
3. Patrón memoize

Comportamiento regular



3. Patrón memoize

Comportamiento memoized



3. Patrón memoize

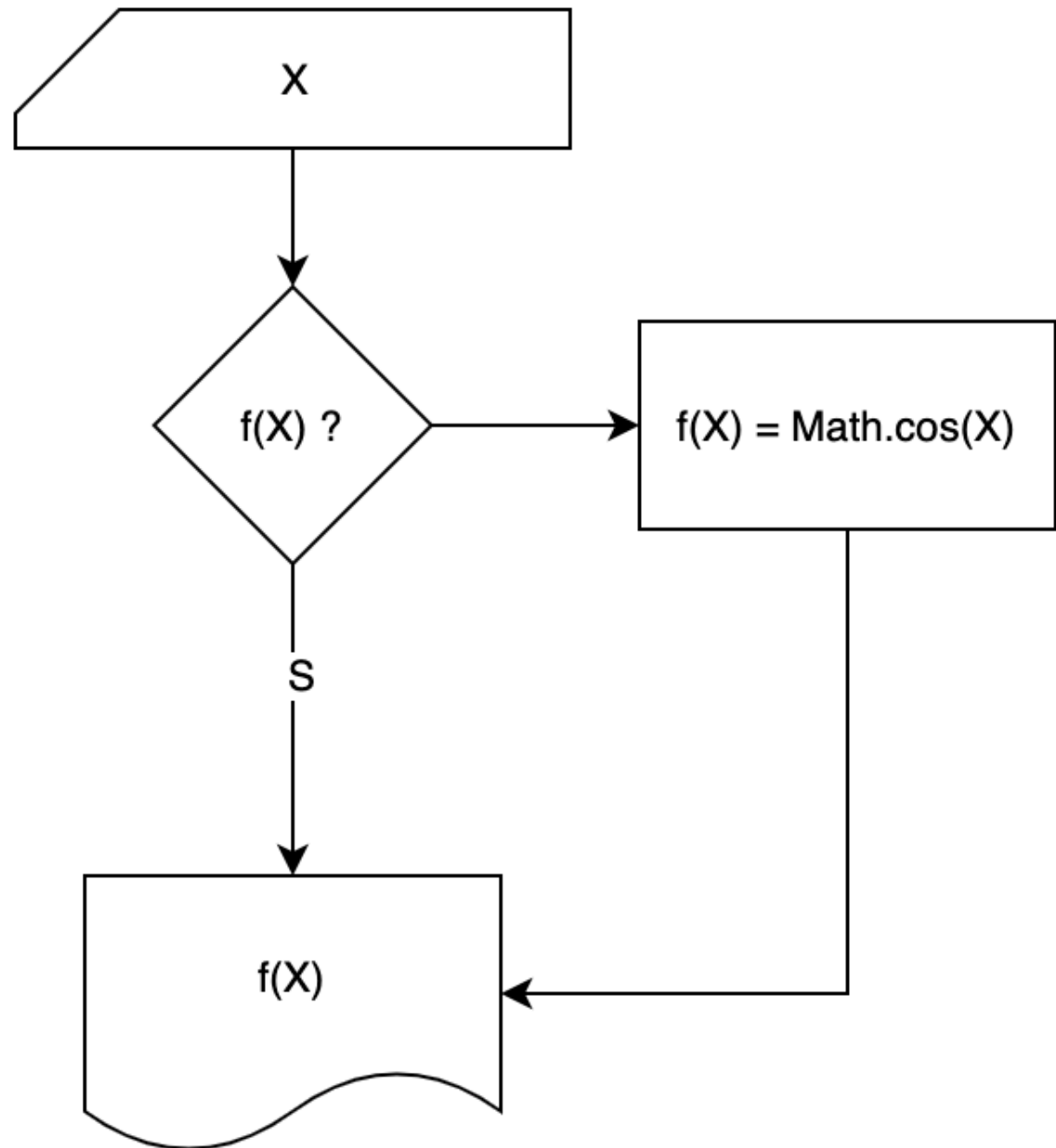
Comportamiento memoized

Angle	Sin	Cos	Tan
1	0.8414709848	0.5403023059	1.557407725
2	0.9092974268	-0.4161468365	-2.185039863
3	0.1411200081	-0.9899924966	-0.1425465431
4	-0.7568024953	-0.6536436209	1.157821282
5	-0.9589242747	0.2836621855	-3.380515006
6	-0.2794154982	0.9601702867	-0.2910061914
7	0.6569865987	0.7539022543	0.8714479827
8	0.9893582466	-0.1455000338	-6.799711455
9	0.4121184852	-0.9111302619	-0.4523156594
10	-0.5440211109	-0.8390715291	0.6483608275
11	-0.9999902066	0.004425697988	-225.9508465
12	-0.536572918	0.8438539587	-0.6358599287
13	0.4201670368	0.9074467815	0.4630211329
14	0.9906073557	0.1367372182	7.244606616
15	0.6502878402	-0.7596879129	-0.8559934009
16	-0.2879033167	-0.9576594803	0.300632242



3. Patrón memoize

Comportamiento memoized




3. Patrón memoize - Personalizado

```

// a simple function to add something
const add = (n) => (n + 10);
add(9);
// a simple memoized function to add something
const memoizedAdd = () => {
  let cache = {};
  return (n) => {
    if (n in cache) {
      console.log('Fetching from cache');
      return cache[n];
    }
    else {
      console.log('Calculating result');
      let result = n + 10;
      cache[n] = result;
      return result;
    }
  }
}
// returned function from memoizedAdd
const newAdd = memoizedAdd();
console.log(newAdd(9)); // calculated
console.log(newAdd(9)); // cached

```


3. Patrón memoize - Lodash



```
import { memoize } from 'lodash';
// a simple function to add something
export const add = (n) => {
  console.log('Calculating result');
  return (n + 10);
};

// a simple memoized function to add something
const memoizedAdd = () => {
  let cache = {};
  return (n) => {
    if (n in cache) {
      console.log('Fetching from cache');
      return cache[n];
    }
    else {
      let result = add(n);
      cache[n] = result;
      return result;
    }
  }
}
// returned function from memoizedAdd
export const newAdd = memoize( memoizedAdd );
```

3. Patrón memoize - Componente funcional

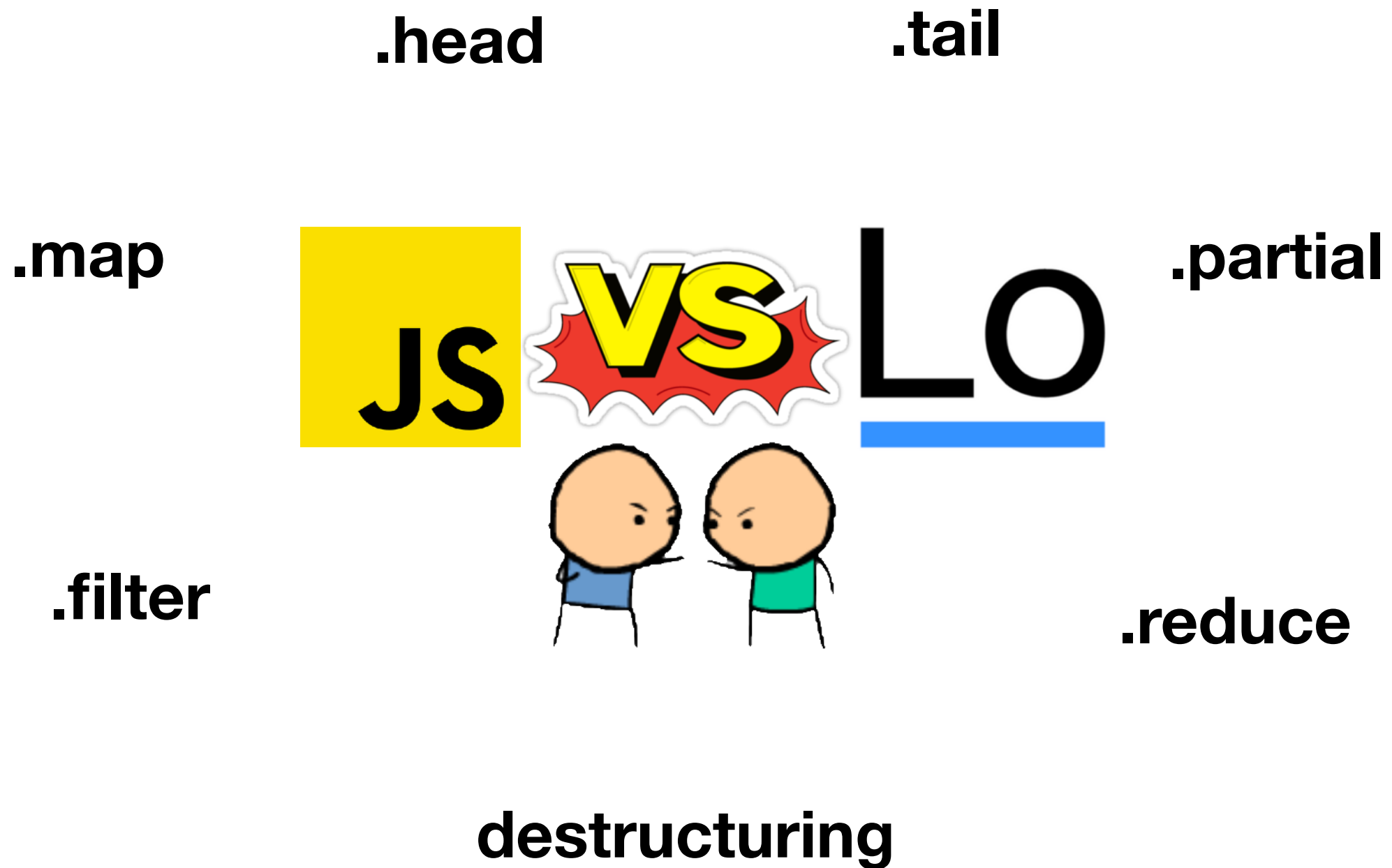


```
import React, { memo } from 'react';

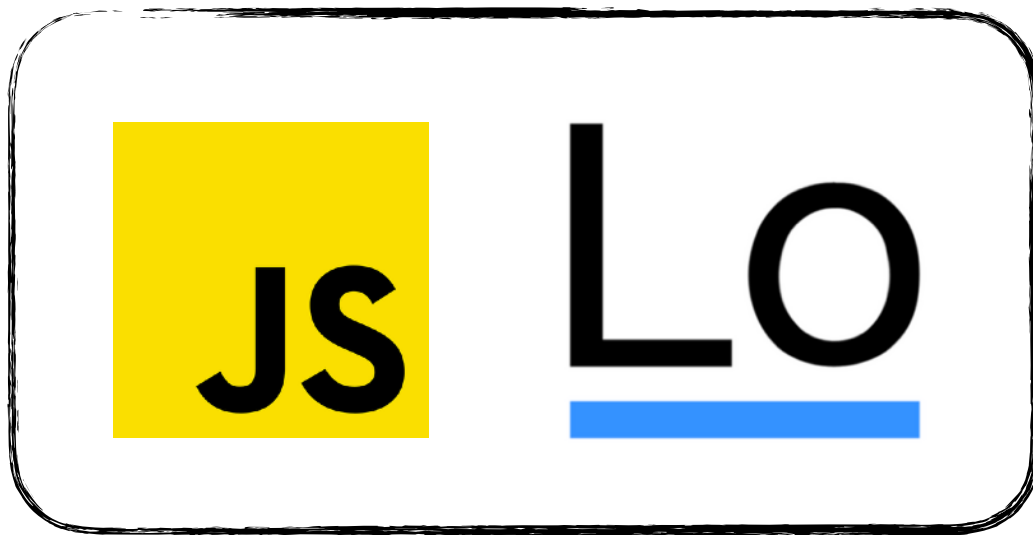
const FpMemo = ({miProp}) => {
  console.log("Renderizando FpMemo");
  return (
    <div>
      FpMemo : {miProp}
    </div>
  );
}

export default memo(FpMemo);
```

4. Utilidades para el cálculo de datos

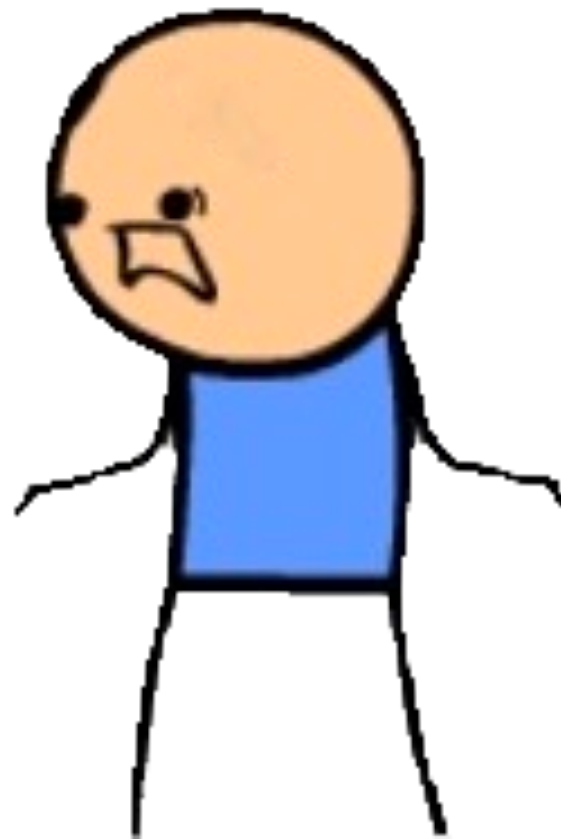


4. Utilidades para el calculo de datos



5. ¿Cómo ocupar reselect?

¿reselect... qué?

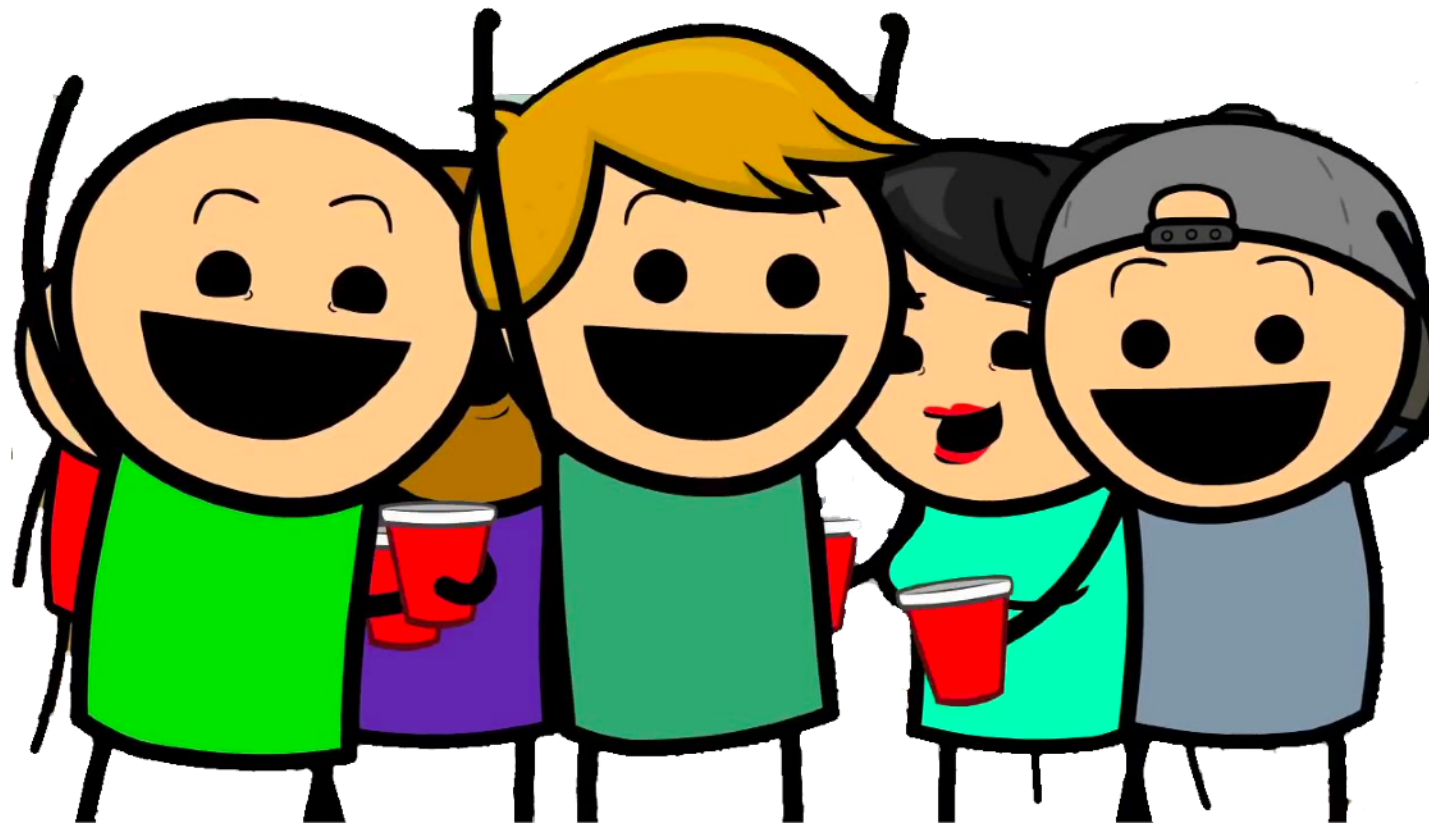


<https://github.com/reduxjs/reselect>

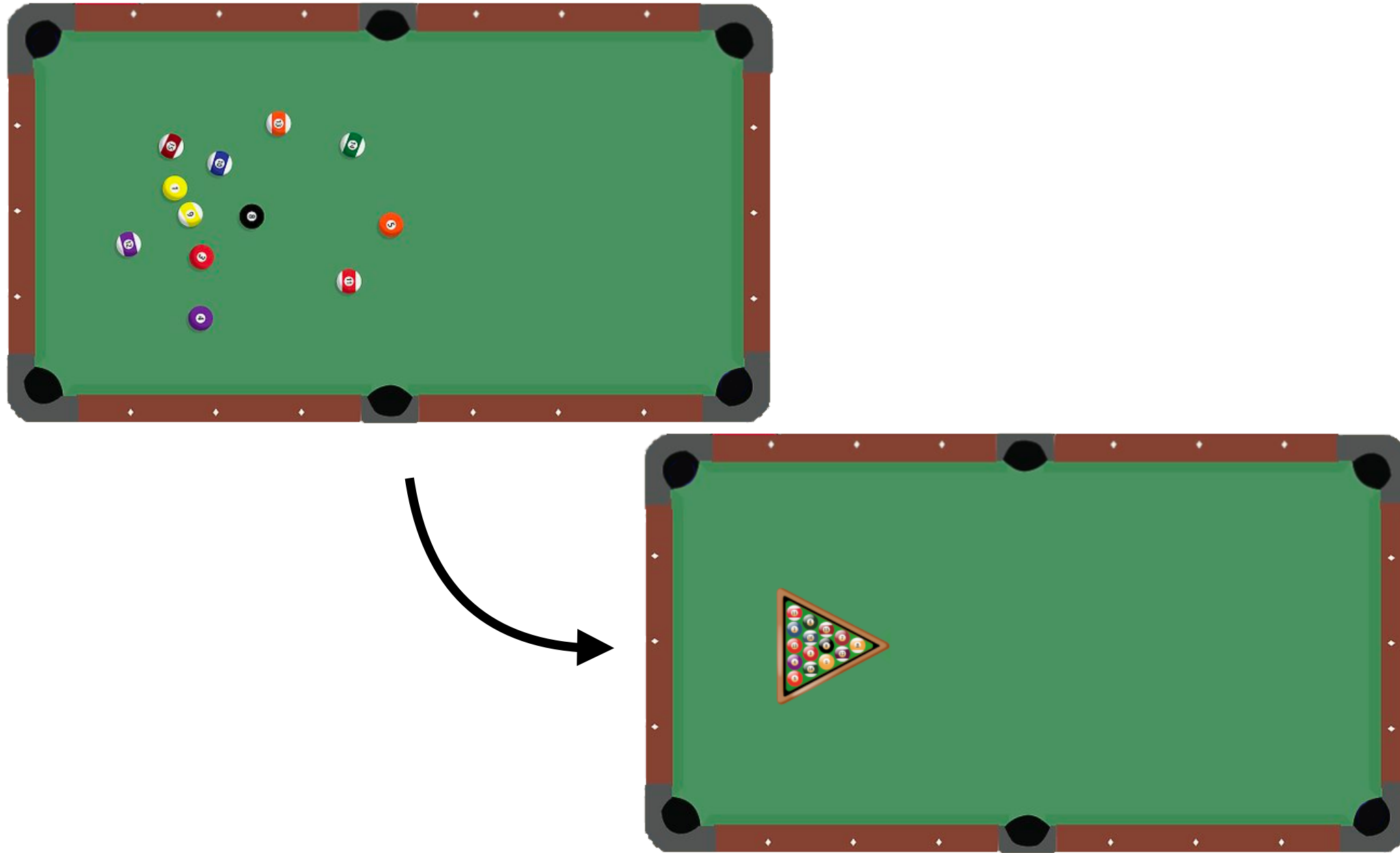
5. ¿Cómo ocupar reselect?



Memoize



5. ¿Cómo ocupar reselect?



5. ¿Cómo ocupar reselect? - mapStateToProps



```
//mapStateToProps sin reselect
const mapStateToProps = (state) => {
  return {
    results: state.data.results,
    counter: state.data.counter
  };
}

//mapStateToProps con reselect
const mapStateToProps = (state) => {
  return {
    results: getFilteredData(state),
    counter: getCounter(state)
  };
}
```

5. ¿Cómo ocupar reselect? - Reducer normal


```
import {combineReducers} from 'redux';
import data from '../data/';

const DATA_STATE = {
  results: data,
  counter: 0
}

const dataReducer = (state = DATA_STATE, action) => {
  switch(action.type){
    case 'FILTER_ITEMS':{
      const filterData = state.data.filter(item => item.title.indexOf(action.payload) !== -1);
      return {...state, results: filterData, filter: action.payload};
    }
    case 'ADD_COUNTER':{
      return {...state, counter: state.counter + parseInt(action.payload, 10)};
    }
    default: {
      return state;
    }
  }
}

export default combineReducers(
  {
    data: dataReducer
  }
);
```

5. ¿Cómo ocupar reselect? - Selector



```
import {createSelector} from 'reselect';

const getAll = state => state.data.data;
const getFilter = state => state.data.filter;

export const getCounter = state => state.data.counter;

export const getFilteredData = createSelector(
  [getAll, getFilter],
  (items, filter) => {
    return items.filter(item => item.title.toLowerCase().indexOf(filter) !== -1)
  }
);
```

6. Hooks. React v16.8.0.

¿Qué son los hooks en React?

Elementos que nos permiten manipular fácilmente el estado de nuestro componente funcional sin necesidad de convertirlos en componentes de clase.

Los hooks no funcionan dentro de las clases.

Al usarlos, podemos evitar totalmente el uso de elementos de ciclo de vida, como

- **componentDidMount**
- **componentDidUpdate**
- **componentWillUnmount**

De igual forma, podemos evitar el uso de ***this***, puesto que el uso de clases queda en el pasado.

6. Hooks. React v16.8.0.

¿Cómo adaptarse al uso de hooks?

- Puede probar Hooks en algunos componentes sin volver a escribir ningún código existente. Pero no tienes que aprender o usar Hooks ahora si no quieres.
- Disponible ahora. Los hooks ahora están disponibles con el lanzamiento de v16.8.0.
- No hay planes para eliminar clases de React...aún ;)
- Los hooks no reemplazan tu conocimiento de los conceptos React.

6. Hooks. React v16.8.0.

Hooks Out of the box

useState

useContext

useReducer

useCallback

useMemo

useRef

useEffect

useImperativeHandle

useDebounce

useLayoutEffect

usePrevious

useDebugValue

useOnScreen

useOnClickOutside

useAnimation

useWindowSize

useEventListener

useWhyDidYouUpdate

useHover

useLocalStorage

6. Hooks. React v16.8.0.

Reglas

- **Llamarlos en un nivel superior. No dentro de bucles, condiciones o funciones anidadas.** Esta regla asegura que hooks se llamen en el mismo orden cada vez que se procesa un componente. Eso es lo que permite a React preservar correctamente el estado de Hooks entre múltiples `useState` y llamadas `useEffect`.
- **Sólo llamar a los hooks en componentes funcionales. No llame a hooks desde las funciones regulares de JavaScript.** Al seguir esta regla, se asegura de que toda la lógica de estado en un componente sea claramente visible desde su código.

6. Hooks. React v16.8.0.

Plugin de Eslint

Existe un complemento creado por el **React Team** que impone estas dos reglas

```
// Your ESLint configuration
{
  "plugins": [
    // ...
    "react-hooks"
  ],
  "rules": {
    // ...
    "react-hooks/rules-of-hooks": "error", // Checks rules of Hooks
    "react-hooks/exhaustive-deps": "warn" // Checks effect dependencies
  }
}
```

7. useState

Declarando el State en una clase

```
1  import React, { Component } from 'react';
2
3  class Component extends Component {
4      constructor (props) {
5          super(props);
6          this.state = {
7              counter: 0
8          }
9      }
10 }
```

Declarando el State con useState



```
1  import React, { useState } from 'react';
2
3  ✓ const Component = () => {
4      |     const [counter, setCounter] = useState(0);
5      | }
6  }
```

7. useState

¿Qué hace el llamado a useState?

Devuelve una variable de estado, y una función para modificarla.

La variable de estado es **counter**, pero es personalizado.

La función es **setCounter**, que también puede ser personalizada.

```
setCounter(100);
```

El hook **useState** entrega las mismas capacidades que **this.state** y **this.setState** proporcionan a una clase.

7. useState

Lectura y escritura de estados con hooks

Visualizando el estado en una clase

```
6      return (  
7          <Text>  
8              Counter times: {this.state.counter}  
9          </Text>  
10     );
```



Visualizando el estado con useState

```
6      return (  
7          <Text>  
8              Counter times: {counter}  
9          </Text>  
10     );
```

7. useState

Lectura y escritura de estados con hooks

Visualizando el estado en una clase

```
6      onClickButton = () => {  
7      |      this.setState({ counter: this.state.counter + 1 });  
8      }
```



Visualizando el estado con useState

```
6      const onClickButton = () => {  
7      |      setCounter(counter + 1);  
8      }
```

8. useMemo

Devuelve un valor memoizado.

Recibe como argumento una función y un arreglo de argumentos que recibe la función, useMemo solo calculara cuando los valores sean distintos a los ya ingresados, esta optimizaron ayuda evitando cálculos costosos en cada render.

Si no se proporciona una matriz, se calculará un nuevo valor en cada render.

```
33     const createList = () => usersList();  
34     const createList = useMemo(() => usersList(), [users]);
```

9. useCallback

useCallback

Devuelve una **retrollamada** (función, clausura o argumento de función) **memoizada** (o memorizada según el **patrón memoized**).

```
26     const onClickPush = useCallback(() => {  
27       |     actionOnClick();  
28     }, []);
```


10. Hooks & Redux

11. useEffect

`componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`

Recomendaciones finales

- **Usar reselect**
- **No optimizar prematuramente**
- **Medir el rendimiento**
- **Usar ESLint**
- **Leer la documentación oficial de React**

¡Gracias!



<https://github.com/jacm80/reactfoundations>

