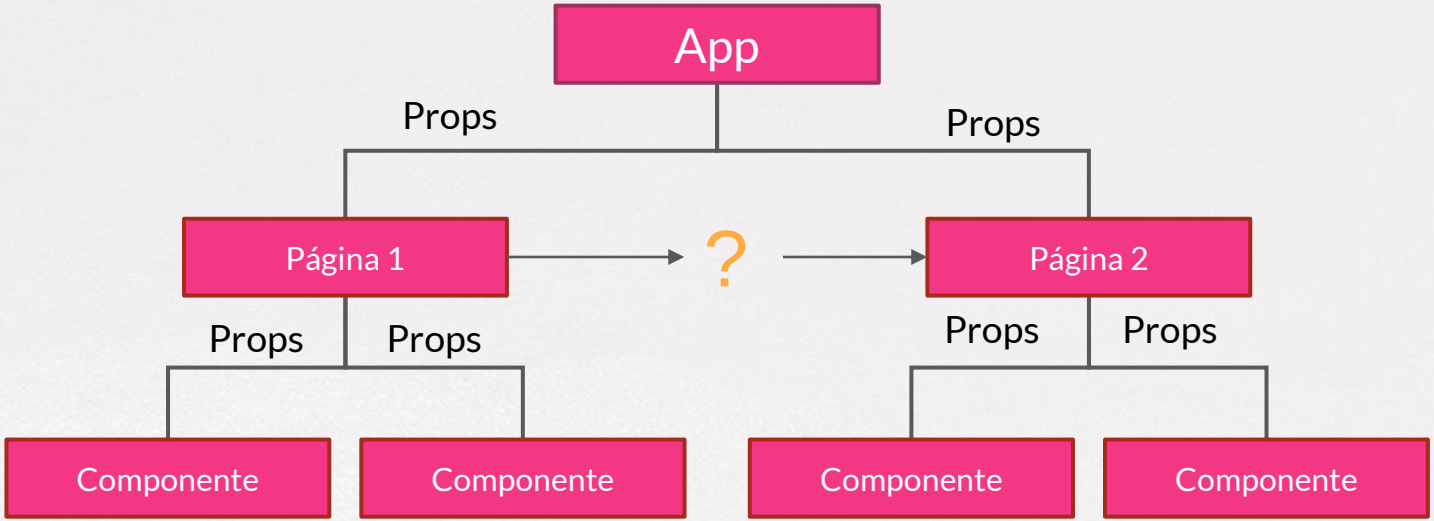
React intermedio

Gestión del estado

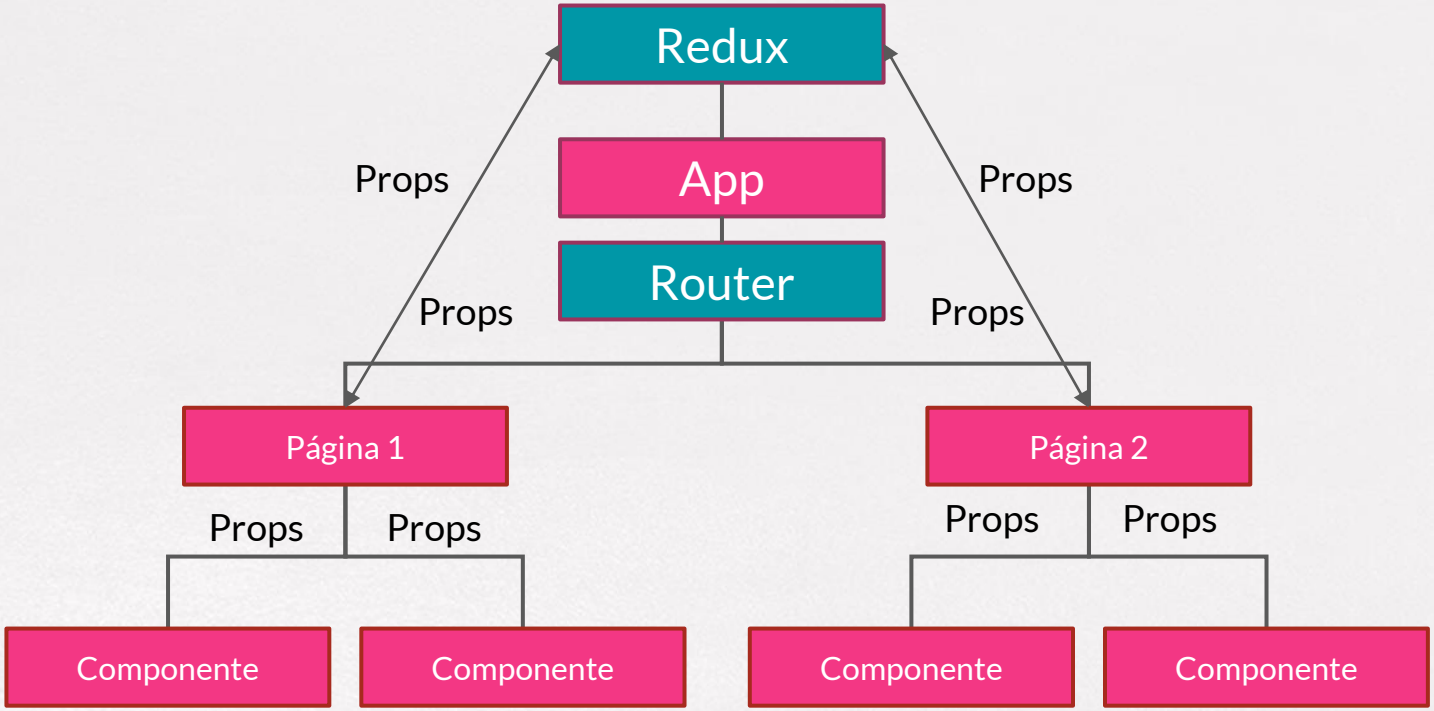
Estado centralizado y hooks

ÍNDICE

El estado centralizado



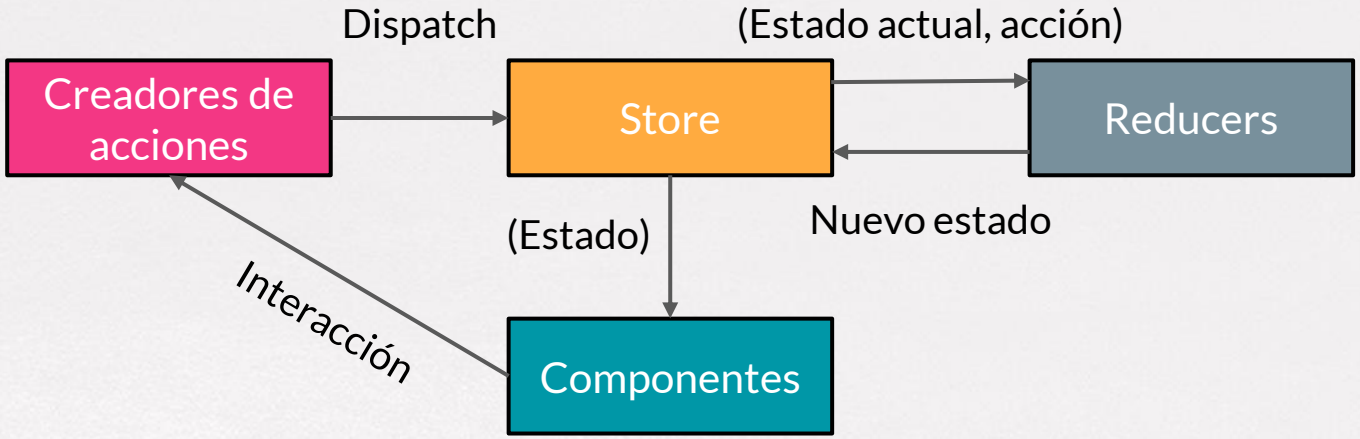
El estado centralizado



Patrón Redux

- En el patrón Redux existe una única fuente de verdad llamada **store** o **almacén** y todo el estado de la aplicación se encuentra en ella
- El estado de las aplicaciones Redux es de sólo lectura y cambiarlo **depende de seguir el flujo interno**
- Los cambios en el estado **se realizan con funciones puras** que devuelven un nuevo estado de la aplicación

Patrón Redux



Redux con hooks: useReducer

```
const [state, dispatch] = useReducer(reducer, initialState);
```

- El hook **useReducer** es una alternativa al hook **useState** que permite utilizar un reducer para gestionar el estado
- Al invocarlo recibe como entrada un estado inicial y una función reductora y **devuelve el estado y una función para ejecutar las acciones**

Redux con hooks: useReducer

```
const [state, dispatch] = useReducer(reducer, initialState);
```

- El hook **useReducer** es una alternativa al hook **useState** que permite utilizar un reducer para gestionar el estado
- Al invocarlo recibe como entrada un estado inicial y una función reductora y **devuelve el estado y una función para ejecutar las acciones**

Redux con hooks: useReducer

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

const initialState = {
  count: 0,
  dispatched: 0
};

const reducer = (state, action) => {
  switch (action.type) {
    case 'incrementa':
      return { ...state, count: state.count + 1, dispatched: state.dispatched + 1 };
    case 'decrementa':
      return { ...state, count: state.count - 1, dispatched: state.dispatched + 1 };
    case 'set':
      return { ...state, count: action.count, dispatched: state.dispatched + 1 };
    default: throw new Error('Acción no encontrada');
  }
};
```

Redux con hooks: useReducer

```
const Ejemplo = () => {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      <div>
        {state.count} - {state.dispatched}
        <button onClick={() => dispatch({ type: 'incrementa' })}>+1</button>
        <button onClick={() => dispatch({ type: 'decrementa' })}>-1</button>
        <button onClick={() => dispatch({ type: 'set', count: 0 })}>reset</button>
      </div>
    </>
  );
};

export default Ejemplo;
```

PARA RESUMIR

- Existen **varios patrones de centralización del estado**, pero generalmente sólo varían en el flujo de acción y se basan en principios similares
- El patrón Redux se basa en crear una **única fuente de verdad** en la aplicación, únicamente de **sólo lectura** y **modificable a través de acciones controladas**
- El hook **useReducer** nos permite implementar **nativamente en la librería** patrones de gestión del estado como Redux