



# Curso **Profesional de React.js y Redux**

Mariangélica Useche  
@musarte.dev

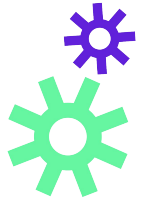






# Conceptos claves para empezar

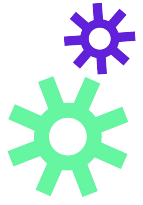




# Redux

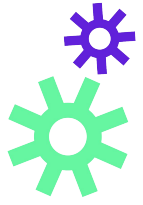
- ¿Qué es Redux?
- ¿Qué necesitamos?
  - Dónde almacenar
  - Cómo acceder
  - Cómo actualizar





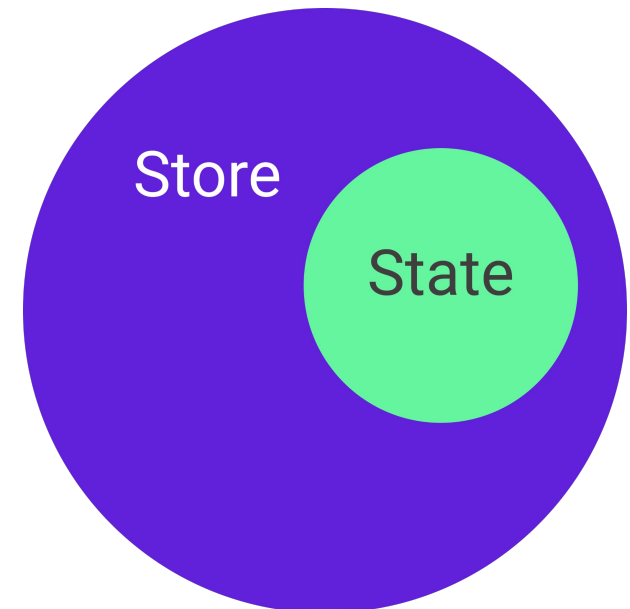
# Tres principios

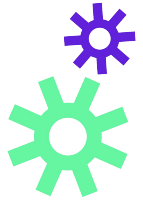
- Única fuente de verdad (**store**).



# Tres principios

- Única fuente de verdad (**store**).



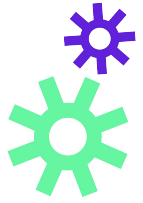


# Tres principios

- Única fuente de verdad (**store**).
- El estado es de solo lectura (**actions**).

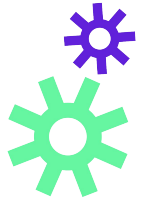
```
const addTodoAction = {  
  type: 'todos/todoAdded',  
  payload: 'Buy milk'  
}
```





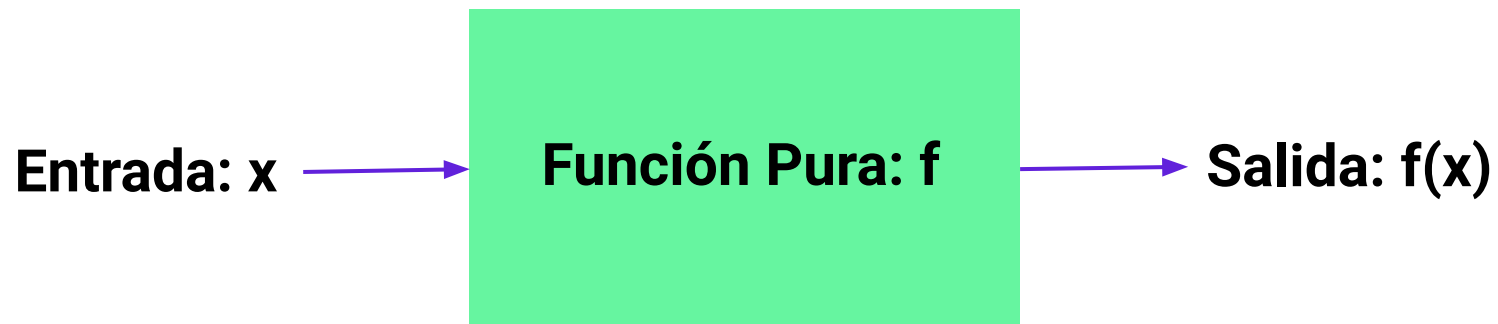
# Tres principios

- Única fuente de verdad (**store**).
- El estado es de solo lectura (**actions**).
- Los cambios deben realizarse a través de funciones puras (**reducers**).

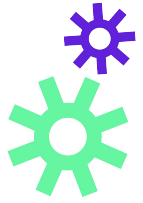


# Funciones Puras

- El valor retornado cambia si la entrada cambia.
- Misma entrada, misma salida.
- Sin efectos colaterales.







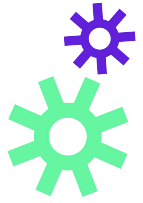
# Reducers

- Calcular el nuevo estado basado solo en los parámetros (state, action).
- No modificar el estado directamente.
- No tener lógica asíncrona.

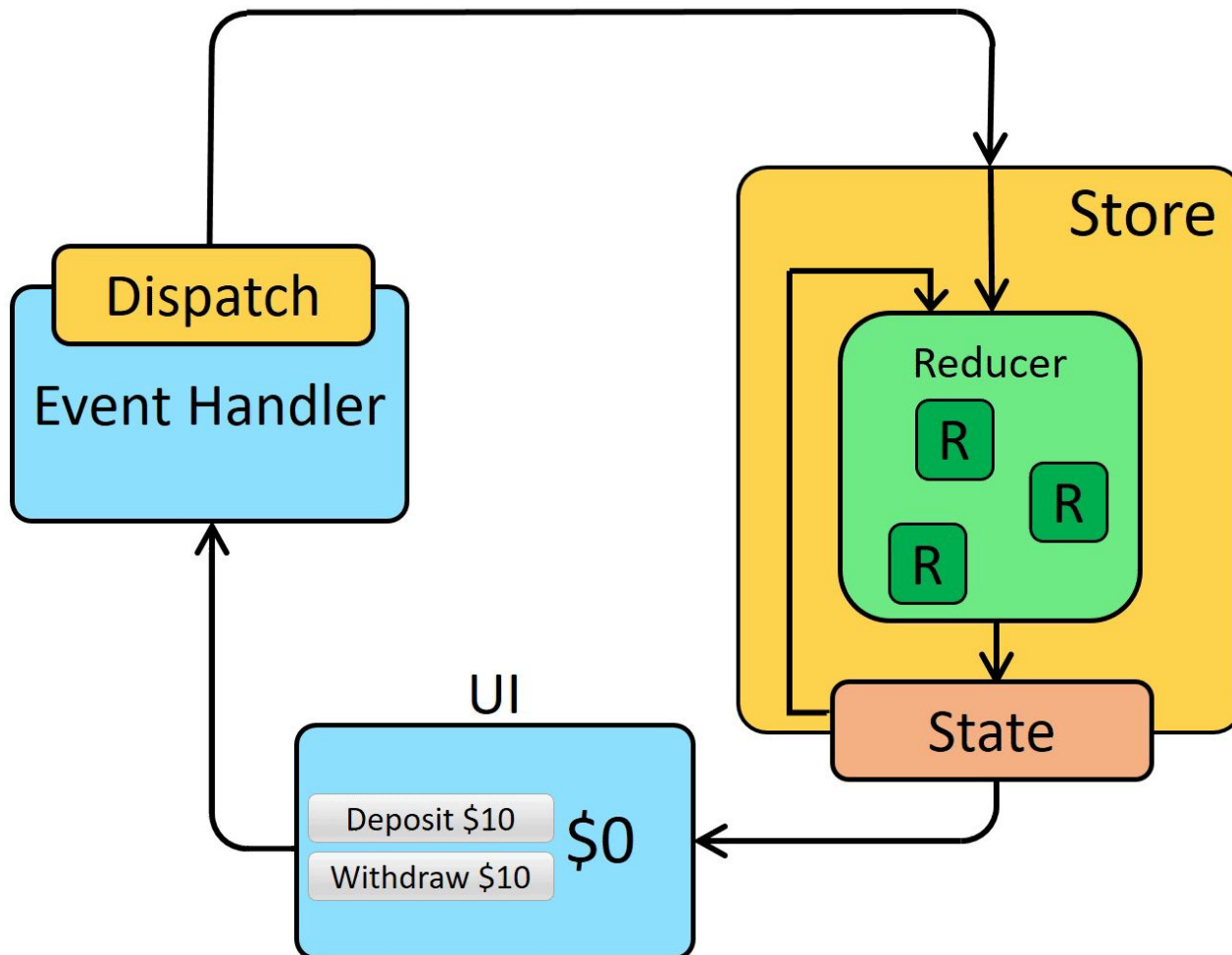


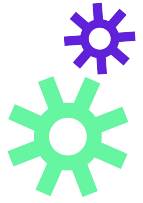
# Ciclo de vida de Redux



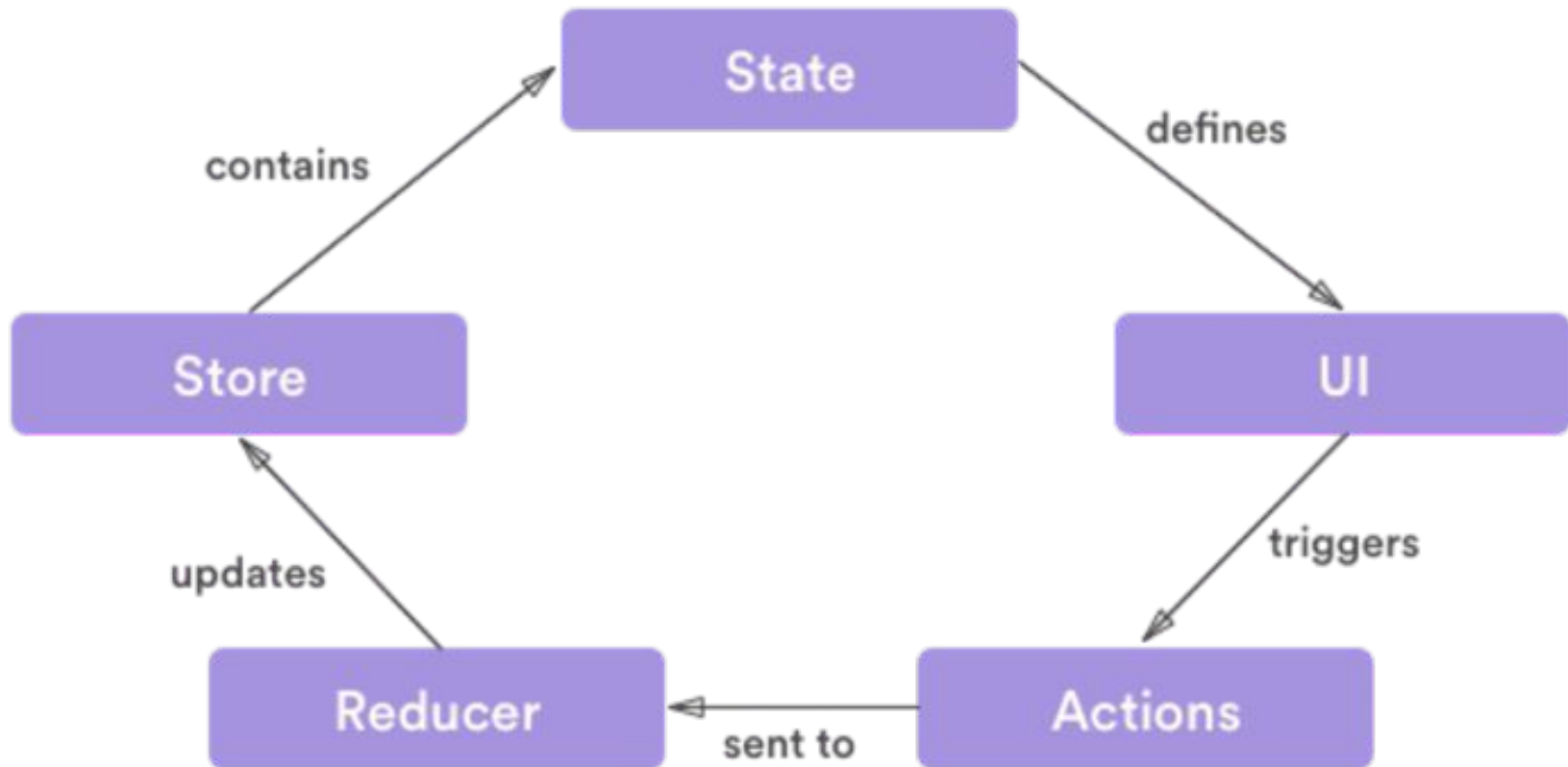


# Ciclo de Vida

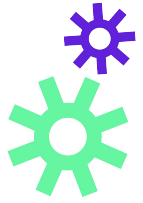




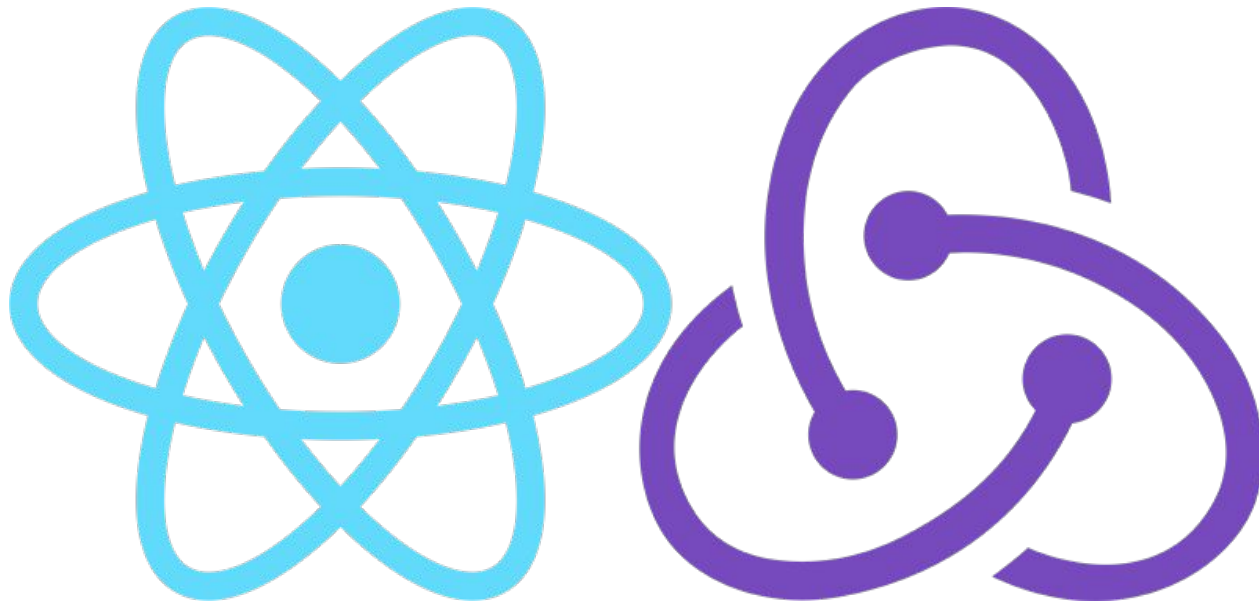
# Ciclo de Vida







# Redux + React





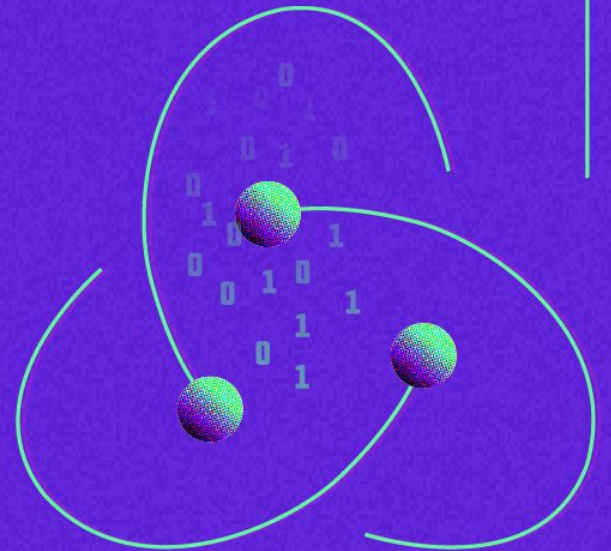
# Redux vs. Context API

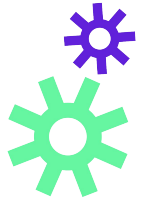




***“Cuando un sistema es opaco y no determinista, es difícil reproducir errores o agregar nuevas características”.***

*Redux Docs*

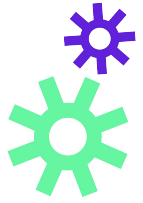




# Context API

- ¿Qué hace?
- ¿Cuándo usarlo?
- La podemos utilizar desde la versión **16.3** de **React**.





# Redux vs. Context API

- Depuración
- Bundle size
- Middlewares
- Curva de aprendizaje
- Rendering



# Introducción a **nuestro proyecto**







# Pokedex



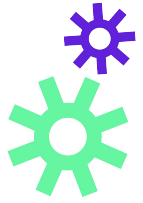




# Hooks vs. Connect





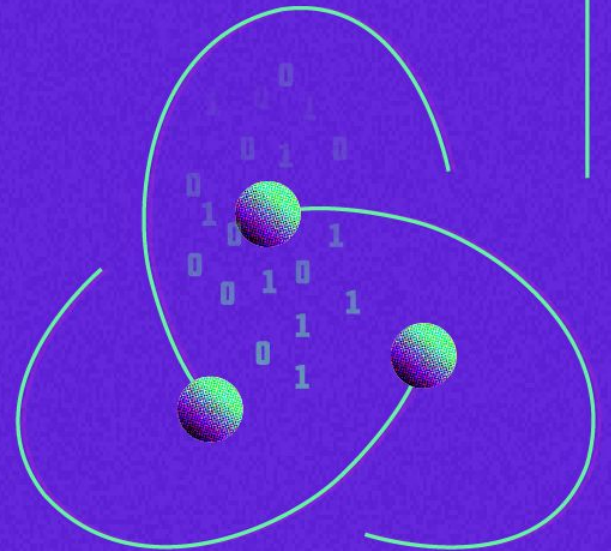


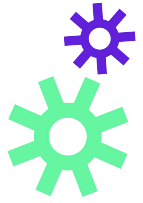
# useSelector vs. Connect

- Boilerplate.
- Separación de responsabilidades.
- Testing.

***“Redux ahora recomienda  
usar su Hooks API”.***

*Redux Docs*





# Hooks API

- useSelector

```
const list = useSelector(state => state.list);
```

- useDispatch

```
const dispatch = useDispatch();  
dispatch(myAction());
```





# Redux DevTools

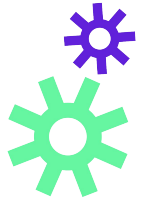






# Middleware





# Glosario

- store creator (createStore)
- enhancer (potenciador)
- compose





# Asincronismo en Redux







# Redux Thunk

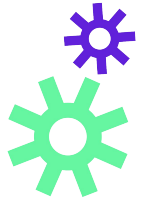






# Middleware alternativos





# Thunk vs. Saga

Redux-Thunk	Redux-Saga
Less boilerplate code	More boilerplate code
Easy to understand as compared to redux-saga	Difficult to understand as there are multiple concepts to learn like generator functions and redux-saga effects
May be difficult to scale up	Easy to scale as compared to redux-thunk
Action creators may hold too much async logic	Action creators stay pure
May get difficult to test	Comparatively easy to test as all your async logic remains together





# Agreguemos un loader







# Agreguemos favoritos







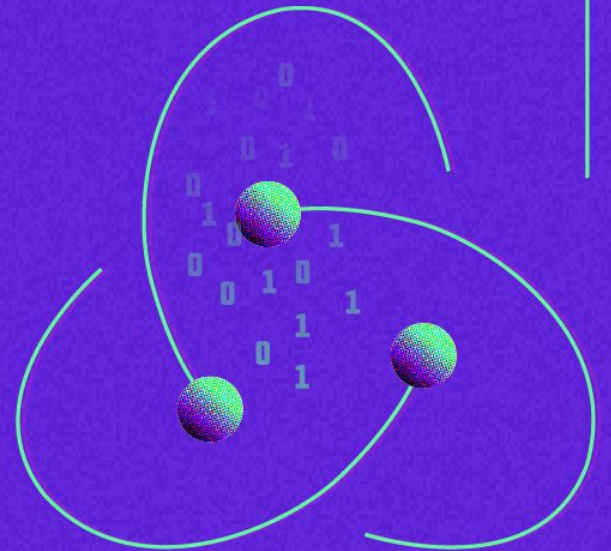
# ¿Qué es inmutabilidad?



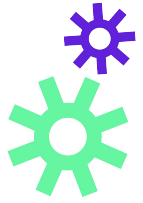


***“Algo que no puede ser  
cambiado después de su  
creación”.***

RAE









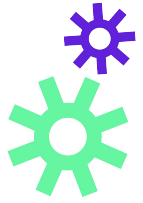
# ¿Por qué es importante?

Redux no indicará a la UI que debe renderizar nuevamente, si su **estado inicial y el estado retornado son exactamente iguales.**



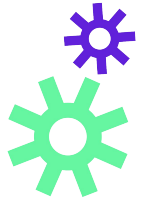
# ¿Cómo trabajar la inmutabilidad en JavaScript?





# Inmutabilidad con JS

- **Object.assign**
- **Spread operator**



# Desventajas

- Generación constante de objetos.
- Propenso a errores humanos.
- Menos trazabilidad.

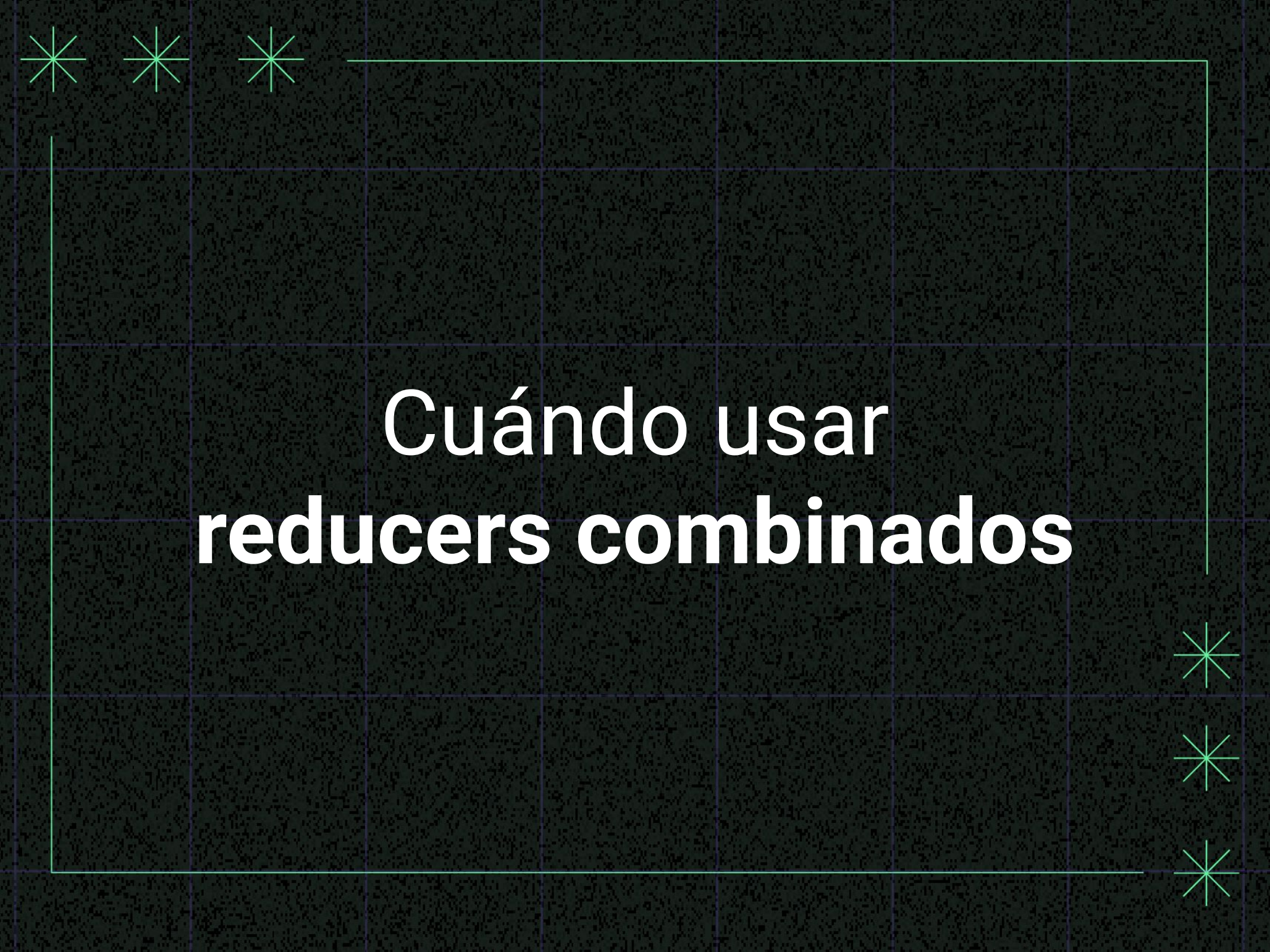




# Agregando **inmutabilidad** a nuestra **Pokedex**







# Cuándo usar **reducers combinados**





# Redux Toolkit







# Redux Toolkit: `createAsyncThunk`







# Conclusiones

@musarte.dev

