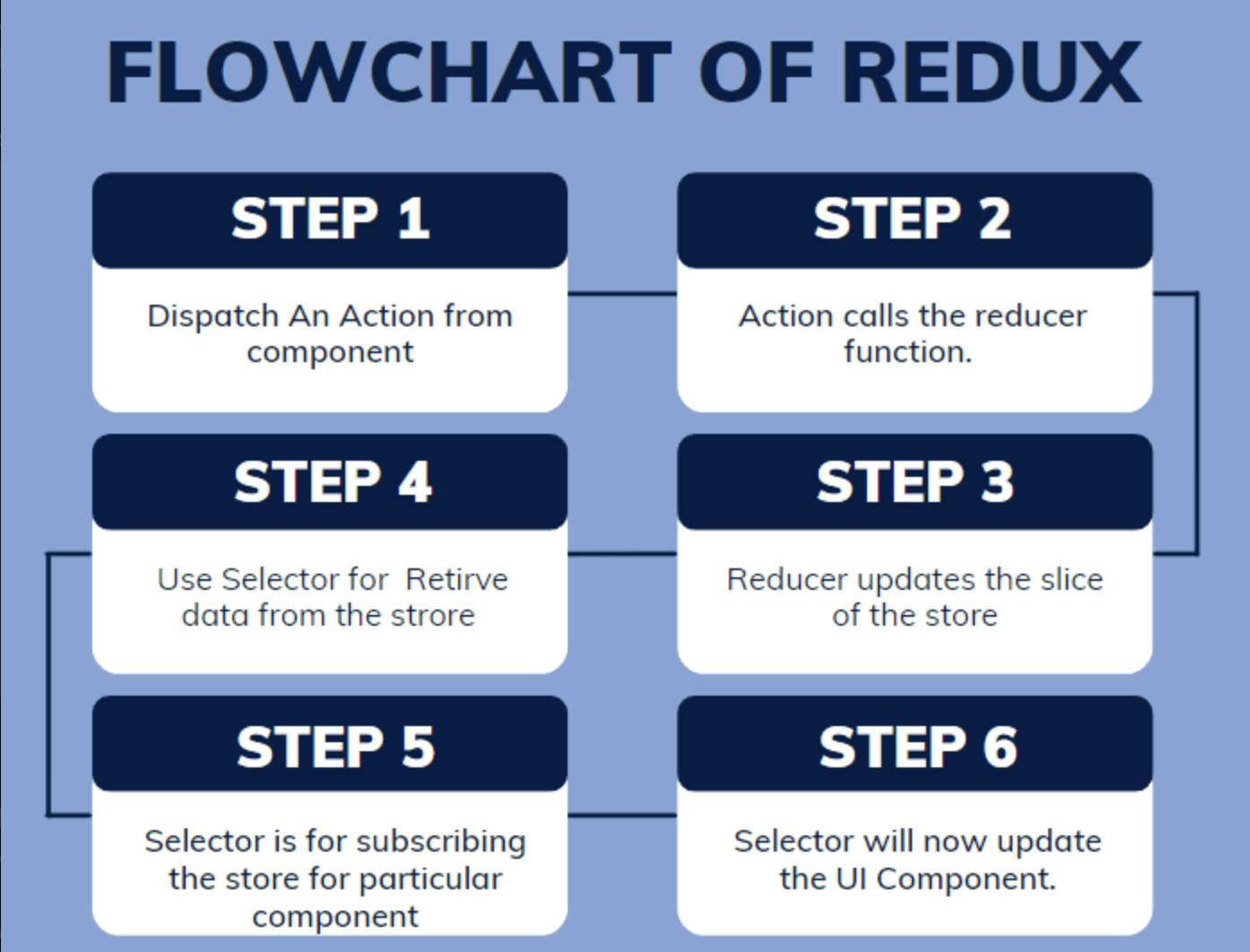
**Redux — State Management Library 🔥**

Redux **allows you to manage your app’s state in a single place and keep changes in your app more predictable and traceable.**



Redux store is like a big object. All components can access it.

State is restricted to only one component. Props helps to pass data from parent component to child component. Context is a centralized store to handle data. Similar to Context is Redux.

Redux has a single object to store all data.

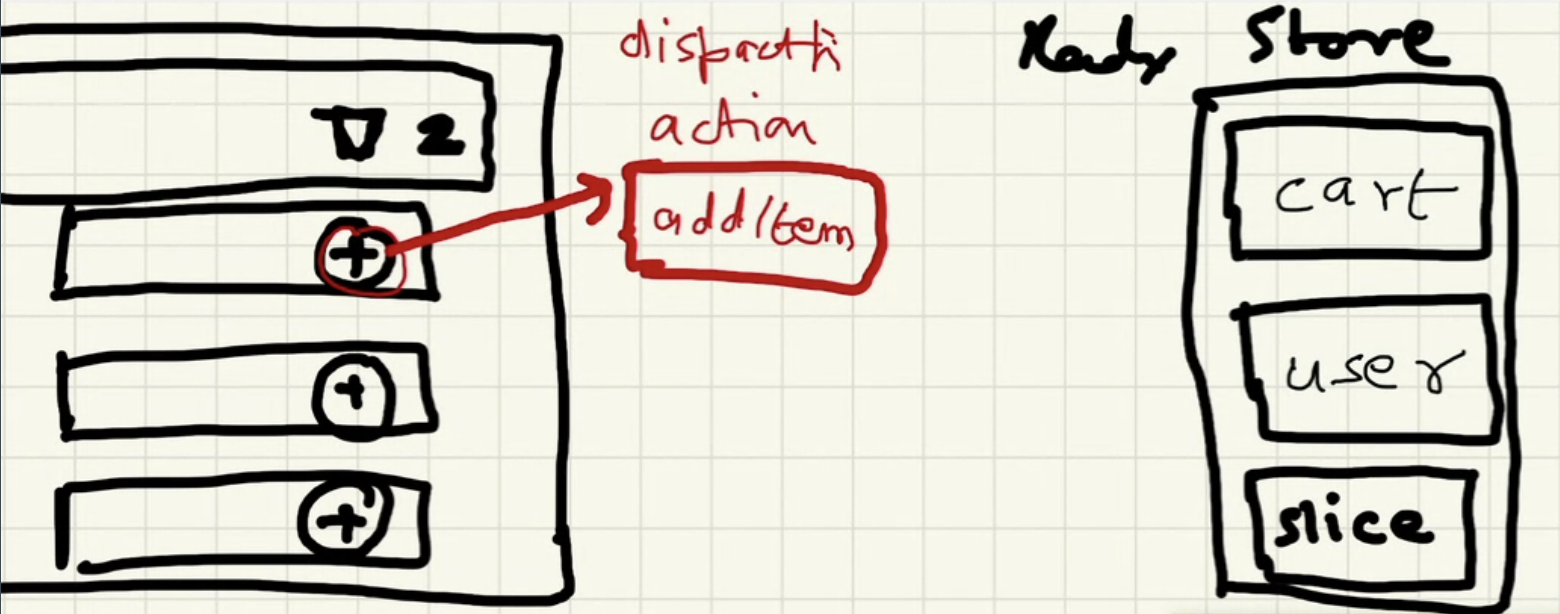
Is it a con to add all data?

We will create slices of of store. So big store has different slices of store. (Authentication slice, Login slice, Cart slice, User Slices)

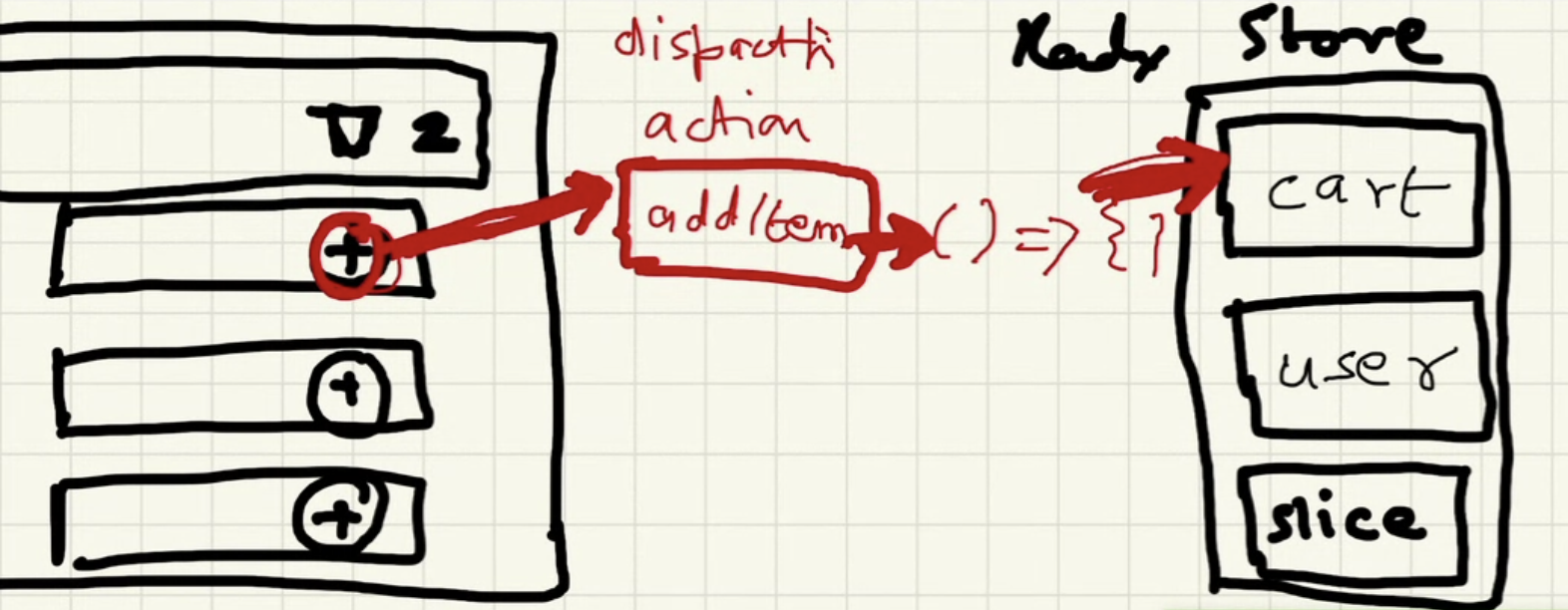
Redux Store

**Write to the Redux Store**

Our components cant directly modify the store. An action from our web app has to be dispatched.

Dispatch an Action

On Click the add button **(Dispatch an Action)** -> Calls a Function **(Reducer)** -> The function modifies the cart **(Slice of the Redux Store)**

On click of Add Button, it Dispatches an Action -> Calls a Reducer function -> Updates the Slice of the Redux Store

On click of Add Button, it Dispatches an Action which Calls a Reducer function which in turn Updates the Slice of the Redux Store

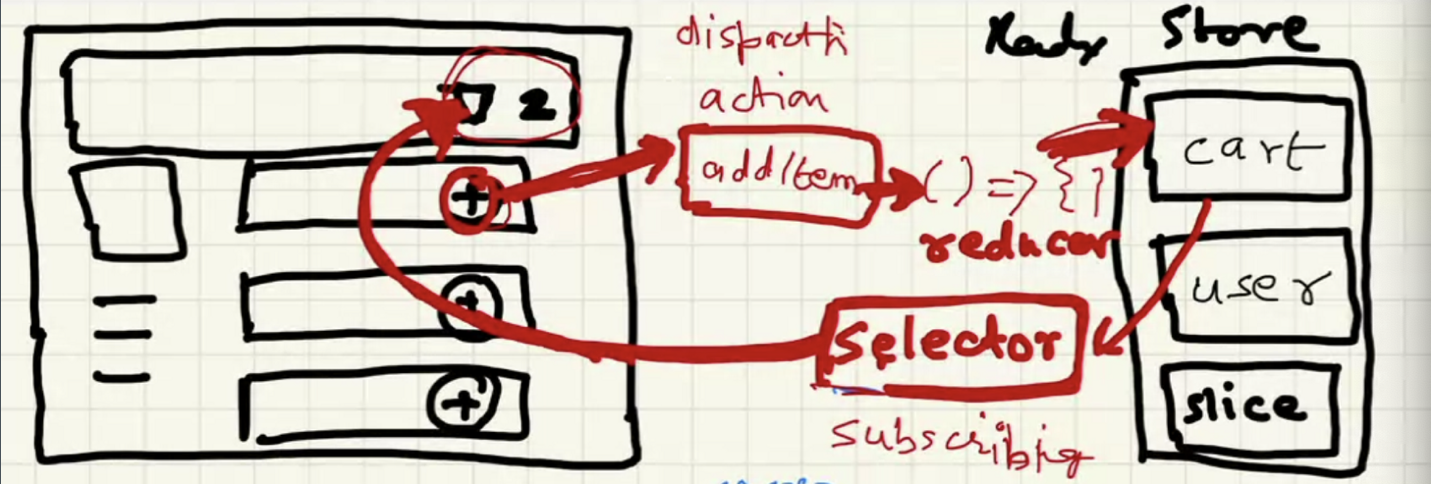
**Read from the Store**

To Read from the Store, we use selector. Selector is a Hook (Hook is a Function **(useSelector)**)

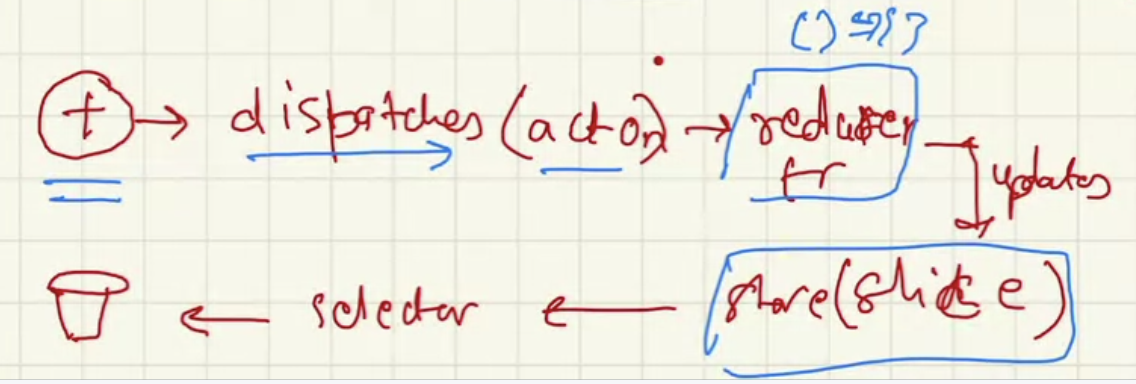
Why is it named selector?

* We are selecting a slice/ portion of the store.

Selector is otherwise called as subscribe

On click of Add Button, it Dispatches an Action -> Calls a Reducer function -> Updates the Slice of the Redux Store -> Selector (read data)

On click of Add Button, it Dispatches an Action which Calls a Reducer function which in turn Updates the Slice of the Redux Store and the Selector automagically updates the cart.



[**Redux - A predictable state container for JavaScript apps. | Redux**  
*A Predictable State Container for JS Apps Redux helps you write applications that behave consistently, run in different…*redux.js.org](https://redux.js.org/)

**Redux Toolkit**

The Redux Toolkit package is intended to be the standard way to write [Redux](https://redux.js.org/) logic. It was originally created to help address three common concerns about Redux:

* “Configuring a Redux store is too complicated”
* “I have to add a lot of packages to get Redux to do anything useful”
* “Redux requires too much boilerplate code”

[**Redux Toolkit | Redux Toolkit**  
*The official, opinionated, batteries-included toolset for efficient Redux development*redux-toolkit.js.org](https://redux-toolkit.js.org/)

**Installation**

npm i @reduxjs/toolkit  
npm i react-redux

**Why two lib?**

* npm i @reduxjs/toolkit -> core lib of redux
* npm i react-redux -> bridge between react and redux

**Code**

Configure Redux Store -> utils/store.js

//Redux store ---> utils/store.js  
import { configureStore } from '@reduxjs/toolkit' //core redux lib  
  
const store = configureStore({ //has slices  
  
})  
  
export default store;

Provider

Provide store to whole app -> App.js

Pass Store as a prop to the provider

import { Provider } from "react-redux";  
import store from "./utils/store";  
  
const AppLayout = () => {  
 const [user, setUser] = useState({  
 name: Priscilla Shamin",  
 email: "shamin@gmail.com",  
 });  
  
 return (  
 <>  
 <Provider store = {store}> // wrap it to the components  
 <UserContext.Provider  
 value={{  
 user: user,  
 setUser: setUser,  
 }}  
 >  
 <Header />  
 <Outlet />  
 <Footer />  
 </UserContext.Provider>  
 </Provider>  
 </>  
 );  
};

Fill up the store with slices -> utils/cartSlice.js

mport { createSlice } from "@reduxjs/toolkit";  
  
  
const cartSlice = createSlice({  
 name : 'cart', //name of slice  
 initialState : { // initial state is empty  
 items: [],  
 },  
 reducers : { // function to modify the state  
 addItem: (state, action) => {  
 state.items.push(action.payload); // no return -> it directly modifies  
 },  
 removeItem: (state, action) => {  
 state.items.pop();  
 },  
 clearCart: (state) => {  
 state.items = [];  
 }   
 }  
})  
  
const cartSlice = createSlice  
  
export default cartSlice.reducer; // combines all reducers and form one reducer  
  
  
// cartSlice = {  
// actions:{  
// addItem,  
// removeItem,  
// clearCart,  
// }  
// reducer: reducers  
// }

//Redux store  
import { configureStore } from '@reduxjs/toolkit' //core redux lib  
import cartSlice from './cartSlice';  
  
const store = configureStore({ //has slices  
 reducer: {  
 cart: cartSlice,  
 }  
})  
  
export default store;

Dispatch

import {addItem} from "../utils/cartSlice"  
import {useDispatch} from "react-redux"  
  
const dispatch = useDispatch();  
  
  
const handleAddItem = () => {  
dispatch(addItem("Grapes")  
}  
  
  
onClick{()=>handleAddItem}

**Flow**

Create store  
-- configureStore() - RTCTrackEvent  
  
Provide my store to app  
-- <Provider store ={store}> - import from react-redux  
  
Slice  
-- RTK -createSlice({  
 name: "",  
 initialState:  
 reducers:{  
 addItem: (state, action) => {state}  
 }  
})  
export const {addItem} = cartSlice.actions  
export default cartSlice.reducer  
  
  
Put the slice into store  
-- reducer: {  
 cart: cartSlice,  
 user: userSlice  
 }

What is useSelector used for?

To subscribe to the store.

Always subscribes to the particular slice of the store.

import {useSelector} from "react-redux"  
  
const cartItems = useSelector(store => store) // don't subscribe directly to the store  
// everytime the store changes, it redenderes the entire store and all slices  
// time consuming and illogical  
  
const cartItems = useSelector(store => store.cart.items) // subscribe to the store and get cart.items slice  
// everytime the store changes, it redenderes the particular slice in the store  
// speeds up the time

**Advantages**

* Used for large applications.
* Easy debugging and testing.

**Disadvantages**

* Complex.
* Difficult to set up.
* Redux requires too much boilerplate code.
* Any component can access the data which can cause security issues.

**Redux DevTools Extension**

[**Redux DevTools**  
*Redux DevTools for debugging application's state changes.*chrome.google.com](https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd?hl=en)