**🧩 What is useReducer?**

* useReducer is a **React hook for managing state** — especially **complex state logic**.
* It’s like a **mini Redux inside a component**.
* Instead of directly updating state like useState, you **dispatch actions** and a **reducer function** calculates the new state.

**useReducer Example:**

const initialState = { count: 0, step: 1 };

function reducer(state, action) {

switch(action.type) {

case "increment":

return { ...state, count: state.count + state.step };

case "decrement":

return { ...state, count: state.count - state.step };

case "setStep":

return { ...state, step: action.payload };

default:

return state;

}

}

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

**🔹 Real-Life Analogy**

* Imagine you’re **managing a bank account**:
  + State = { balance: 1000 }
  + Actions = deposit, withdraw
  + Reducer = “a teller who decides how balance changes based on action”

dispatch({ type: "deposit", payload: 500 }); // new balance = 1500

dispatch({ type: "withdraw", payload: 200 }); // new balance = 1300

🔹 **When to Use useReducer**

| **Situation** | **Recommendation** |
| --- | --- |
| Simple state (single number or string) | useState ✅ |
| Complex state (object, multiple dependent variables) | useReducer ✅ |
| Multiple ways to update the same state | useReducer ✅ |
| Want predictable, testable state updates | useReducer ✅ |

🔹 **Interview Questions**

| **Question** | **Answer / Key Point** |
| --- | --- |
| What is useReducer? | Hook for managing complex state via reducer function |
| When to prefer useReducer over useState? | When state is complex, has multiple fields, or updates depend on previous state |
| What is the difference between dispatch and setState? | dispatch sends an action to reducer → reducer returns new state; setState directly updates state |
| Can useReducer work with context? | Yes! Combine useReducer + useContext for app-level state without Redux |
| Is reducer function pure? | ✅ Yes. Must return new state based on current state and action |

**✅ Key Takeaways**

1. useReducer = **centralized, predictable state management**
2. Perfect for **complex state logic**
3. Works well with useContext → **mini Redux** inside component tree
4. Each dispatch → reducer calculates **new state** → triggers re-render

**1️⃣ If you are using Redux**

* Redux already provides a **central store**, **dispatch**, and **reducers**.
* Every component can **dispatch actions to update global state** and **subscribe to slices**.
* Because of this, **using useReducer inside components becomes redundant** for global state.

You might still use useReducer for **very local, component-specific complex state**, but it’s usually rare because Redux handles most state centrally.

**2️⃣ If you are NOT using Redux**

* useReducer is perfect for **complex state inside a single component** or **a small part of the component tree**.
* Example: To-do list, form with multiple dependent fields, counters with multiple steps, or a mini cart in a component.
* It **centralizes state logic**, making it easier to manage, debug, and test.