5. Redux (in-app state management)

Monday, 22 December 2025 12:24 PM

Great question 👍 — this confusion is **very common**, so let's clear it step-by-step and relate it to things you already know.

Short answer first

Redux is NOT like Redis cache.
Redux is **in-app state management**, not storage or cache.
You can think of Redux as **controlled internal memory for your React app**, but it **lives only while the app is running**.

What Redux actually is

Redux = a predictable state container for JavaScript apps (mostly React)
It helps you:

- Keep **shared/global data** in one place
- Update that data in a **strict, predictable way**
- Avoid passing props deeply (prop drilling)

Examples of data stored in Redux:

- Logged-in user info
- Cart items
- Theme (dark/light)
- App-wide settings
- API response data (sometimes)

Is Redux "internal memory"?

✔ Yes — but only inside the browser/app

Redux is:

- In **JavaScript memory (RAM)**
- Reset on **page refresh**
- Lost when the tab/browser closes

So it's **runtime memory**, not persistence.

Redux vs React state (useState / useReducer)

Feature	useState / useReducer	Redux
Scope	Component / feature	Whole app (global)
Lifetime	While component exists	While app runs
Sharing state	Hard across many components	Easy
Rules	Flexible	Very strict & predictable

Redux internally **uses the same reducer idea** you already learned 👍

Redux vs Redis (VERY important difference)

Redux	Redis
Frontend (browser / JS)	Backend (server)
In-memory JS object	In-memory database
Per user / per tab	Shared across users
Lost on refresh	Persistent (optional)
For UI state	For caching, sessions, queues

Analogy:

- **Redux** → Whiteboard inside your room

- **Redis** → Central library storage

Redux flow (mental model)

Think of Redux like a **bank ledger**:

1. **Action** – "I want to change something"
2. **Reducer** – "Given current state + action → return new state"
3. **Store** – "Single source of truth"

dispatch({ type: "ADD_TO_CART", payload: item });
➡️ Reducer decides
➡️ Store updates
➡️ All subscribed components re-render

Simple diagram (conceptual)

Component
↓ dispatch(action)
Redux Store
↓ reducer(state, action)
New State
↓
All Components update

Is Redux global memory?

✔️ **Yes — global for the app**
❌ **No — not global across refreshes or users**

If you want persistence:

- Redux + **localStorage**
- Redux + **backend API**
- Redux + **Redux Persist**

When should you use Redux?

Use Redux when:

- Many components need same data
- State updates are complex
- App is large
- You need predictable debugging (Redux DevTools)

❌ Avoid Redux when:

- Small app
- Simple parent-child state
- useState/useContext is enough

One-line takeaway

Redux is like structured internal memory for your React app — not a cache like Redis, and not permanent storage.

If you want, next I can:

- Compare **Redux vs Context**
- Explain **Redux Toolkit**
- Show **Redux example vs useReducer**
- Explain **how Redux is used in real apps**