React state Management libraries and How to choose

state management = How you store the state + How you change it

Storage: 1) keep variables in local component state

2) keep the data in a store, using a library like Redux, Recoil, MobX, or Zustand

3) keep them on the `window` object globally

Some frameworks, like svelte and Vue, "watch" for things, and update accordingly. React doesn't "watch for changes" and magically re-render. You (or something) needs to tell it to do that.

1) w/ useState, useReducer, or this.setState, React will re-render when you call one of the setter functions.

2) if you keep the data in Redux, MobX, Recoil, or some other store, then that store will tell React when something has changed, and trigger the re-render for you.

3) if on `window`, you need to tell React to update after you change that data.

once you get past 3-5 useState calls in a single component, things are probably going to get hard to keep track of. Especially if those bits of state depend on each other.

↳ Alternative: state machine

A useReducer can only hold one value, but it's more common for that single value to be an object containing multiple values.

Avoiding prop drilling w/ React Context.

↙

Downside: performance (unless you're very careful)

Every component that calls useContext will re-render when the Provider's value prop changes. If the value is an object containing several different bits of state that change frequently and independently, every time one of those values changes, every component that uses any of them would re-render.

↳ Alternative: store small chunks of related data in each Context and split up data across multiple Contexts

Redux

- Functional style, immutability
- Small bundle size
- Drawback: Heavy reliance on immutability can make it cumbersome to write reducers. This is mitigated by adding the Immer library or using Redux Toolkit (which includes Immer).

MobX (Managing state in a truly "reactive" way)

- where Redux is all about being explicit and functional, MobX takes the opposite approach
- It's based on the observer/observable pattern.
- You'll create an observable data model, make your components as "observers" of that data, and MobX will automatically track which data they access and re-render them when it changes.

MobX state tree (MST)

- Layer on top of MobX for a reactive state tree.
- The model can have views (computed properties) and actions (setter functions)
- If you're changing data very rigidly, MST might not be the best fit.

Recoil

- Its API looks like a combination of React's useState and Context APIs.
- By keeping track of calls to useRecoilState, Recoil keeps track of which components use which atoms. It can re-render only the components that "subscribe" to a piece of data when that data changes.

react-query

- Data fetching library

XState

- State machines and statecharts

More libraries: Zustand and easy-peasy (and Jotai)