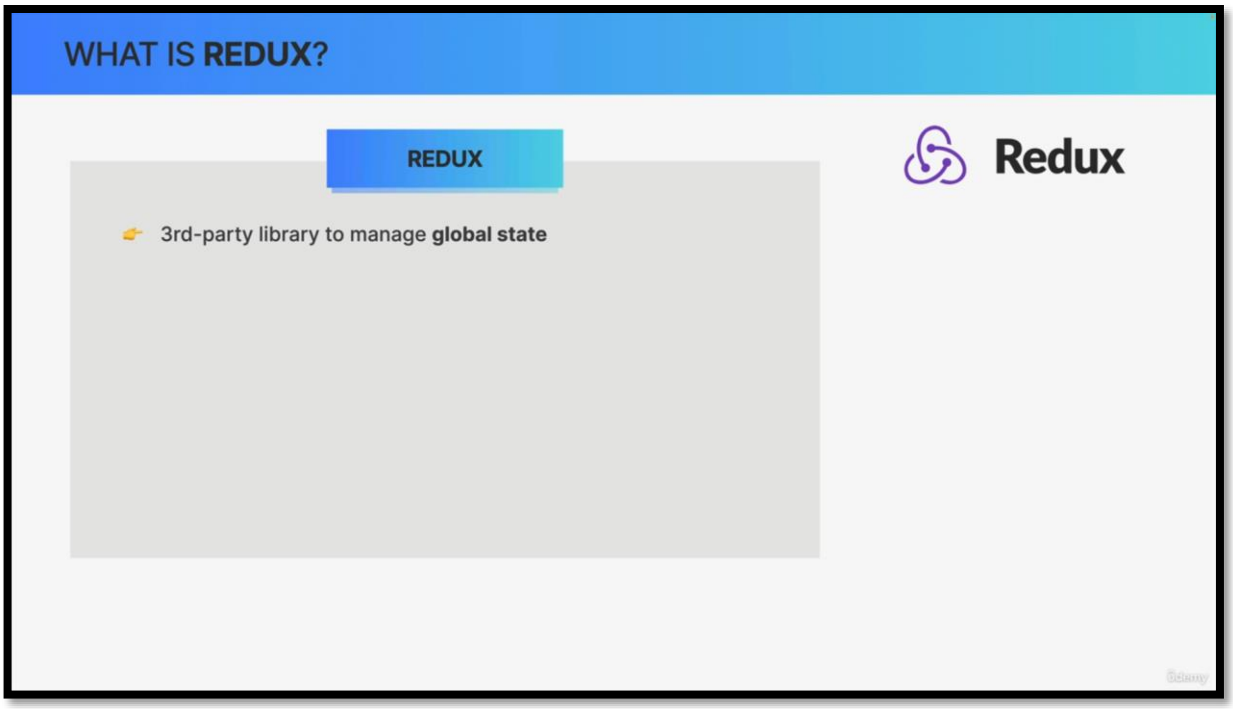


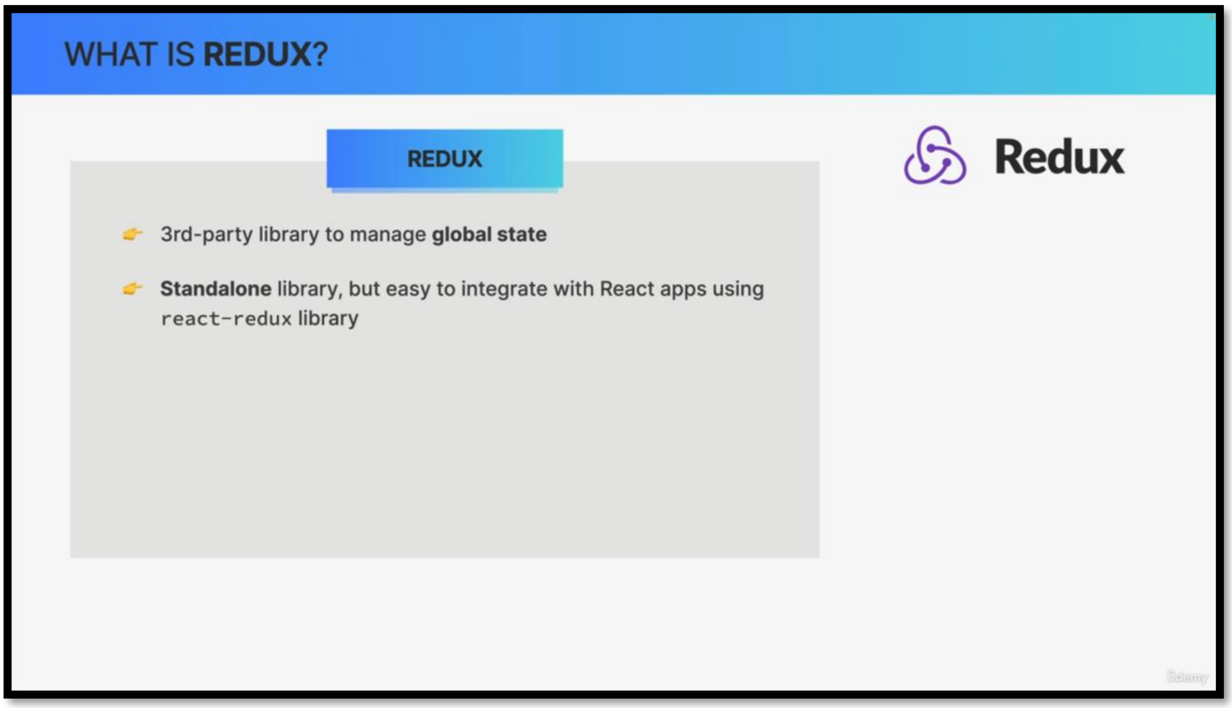
WHAT IS **REDUX?**

REDUX

- 3rd-party library to manage **global state**

Redux

Redux is a third-party library that we can use to manage global state in a web application.



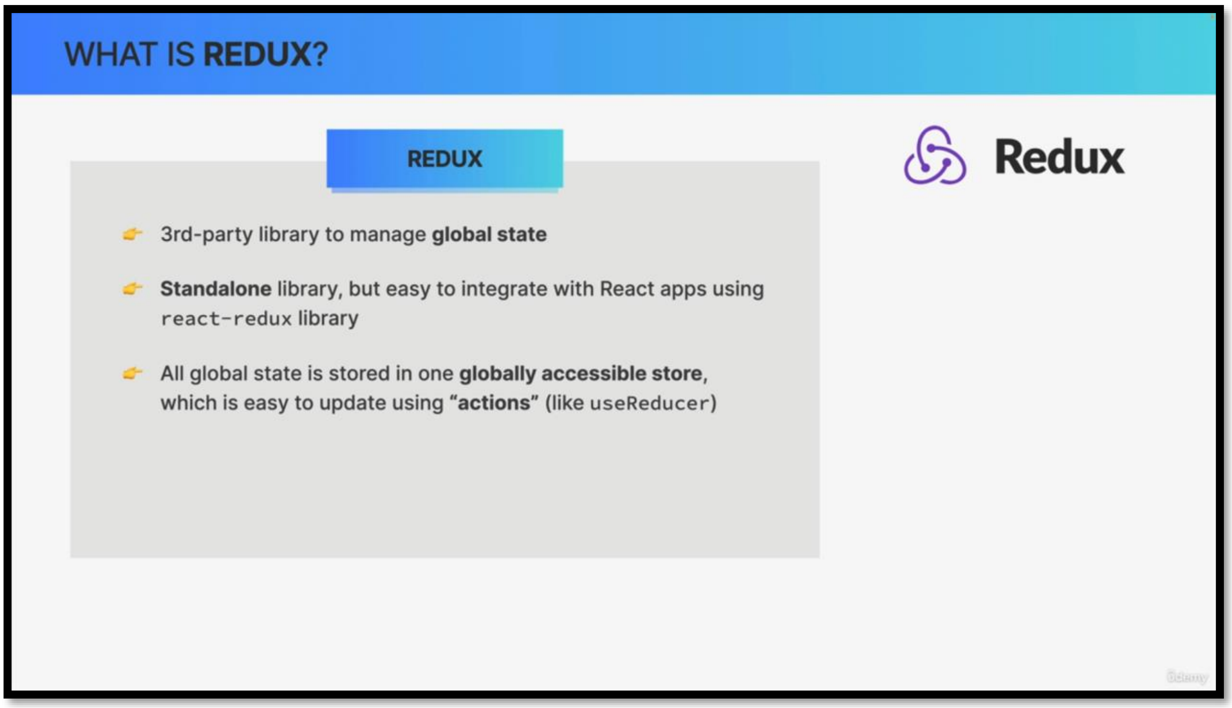
WHAT IS **REDUX?**

REDUX

- 3rd-party library to manage **global state**
- **Standalone** library, but easy to integrate with React apps using `react-redux` library

Redux

Because Redux is actually a complete standalone library that we could use with any framework or even Vanilla JavaScript but Redux has always been tightly linked to React and they are actually quite easy to connect using the react-redux library.



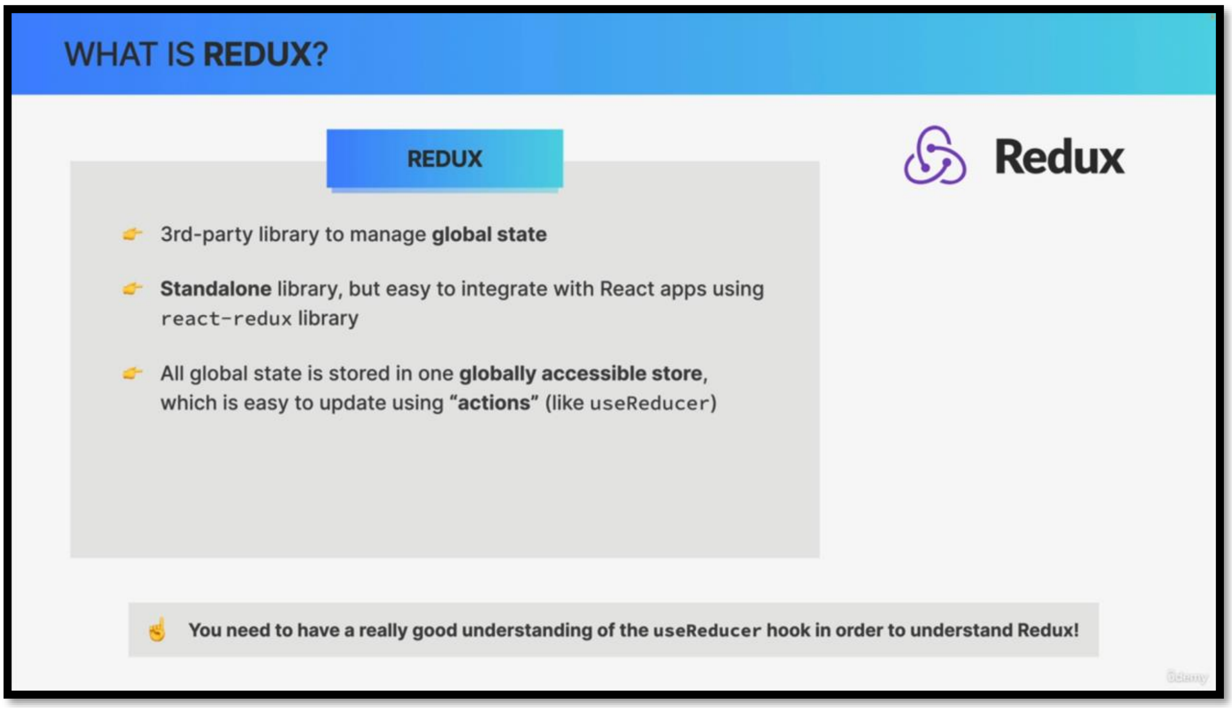
## WHAT IS **REDUX?**

**REDUX**

- 3rd-party library to manage **global state**
- **Standalone** library, but easy to integrate with React apps using `react-redux` library
- All global state is stored in one **globally accessible store**, which is easy to update using **"actions"** (like `useReducer`)

The big idea behind Redux is that all the global state in your application can be stored inside this one globally accessible store which we can then update using actions.

This sounds familiar, it's because the useReducer hook implements a very similar pattern. So, updating state by dispatching actions.



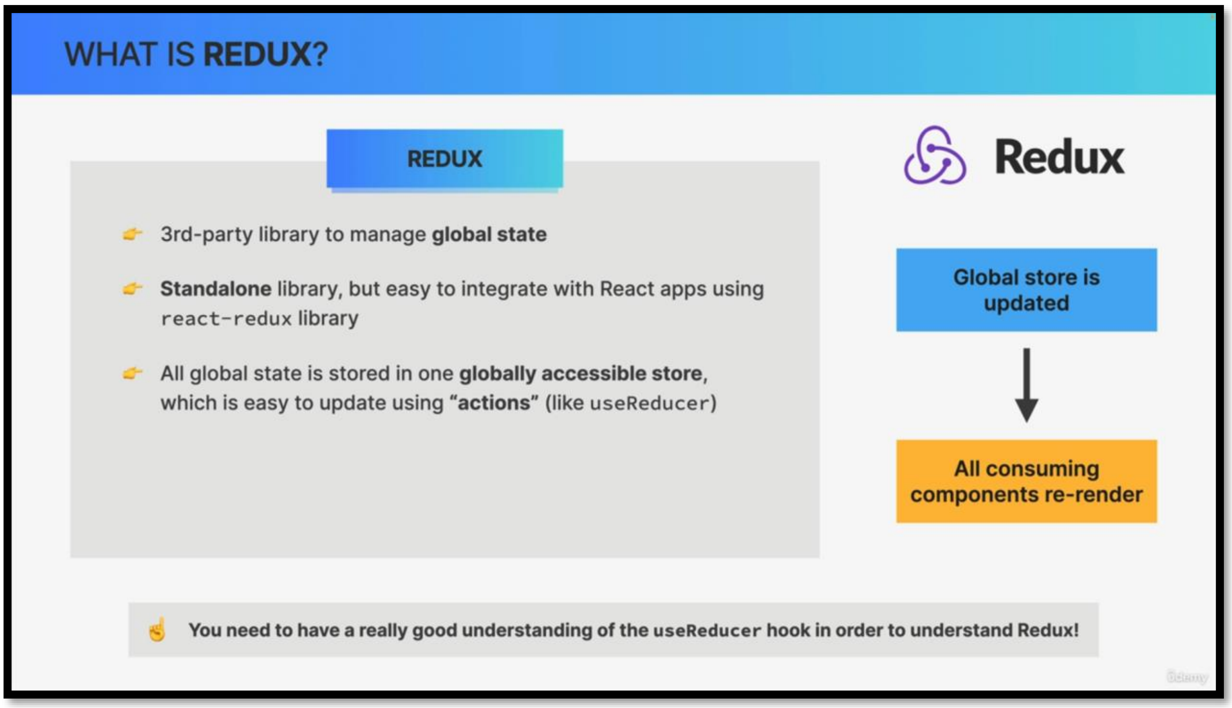
## WHAT IS **REDUX?**

**REDUX**

- 3rd-party library to manage **global state**
- **Standalone** library, but easy to integrate with React apps using `react-redux` library
- All global state is stored in one **globally accessible store**, which is easy to update using **"actions"** (like `useReducer`)

☝ **You need to have a really good understanding of the `useReducer` hook in order to understand Redux!**

The idea of updating state in React and in Redux is always the same.



## WHAT IS REDUX?

REDUX

- 3rd-party library to manage **global state**
- **Standalone** library, but easy to integrate with React apps using `react-redux` library
- All global state is stored in one **globally accessible store**, which is easy to update using **"actions"** (like `useReducer`)

Redux

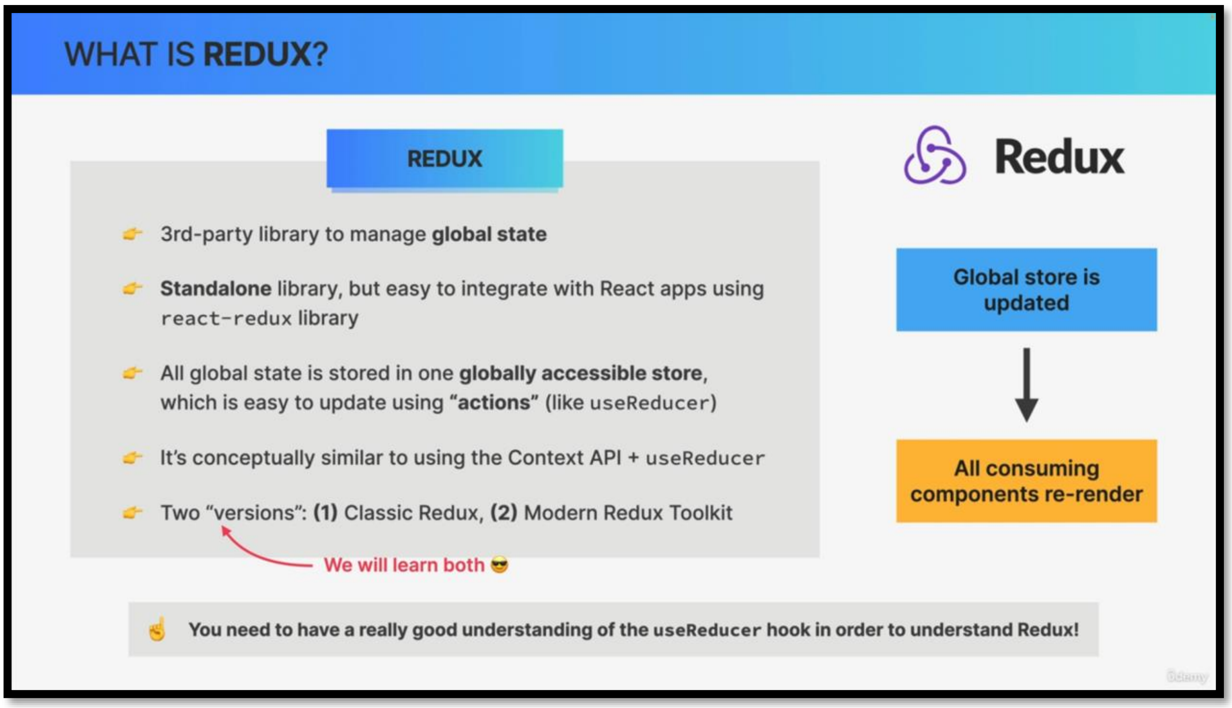
Global store is updated → All consuming components re-render

☝️ You need to have a really good understanding of the `useReducer` hook in order to understand Redux!

```
So, in the case of Redux, as soon as we update the store all the React
components that consume some data from the store will be re-rendered.

So, conceptually, if we think about this Redux is actually quite similar
to combining the Context API with the useReducer hook as we have done in
the World Wise app.

In fact, many developers even say that the Context API is a replacement
for Redux.
```



## WHAT IS REDUX?

REDUX

- 3rd-party library to manage **global state**
- **Standalone** library, but easy to integrate with React apps using `react-redux` library
- All global state is stored in one **globally accessible store**, which is easy to update using **"actions"** (like `useReducer`)
- It's conceptually similar to using the Context API + `useReducer`
- Two "versions": **(1)** Classic Redux, **(2)** Modern Redux Toolkit

We will learn both 😎

Redux

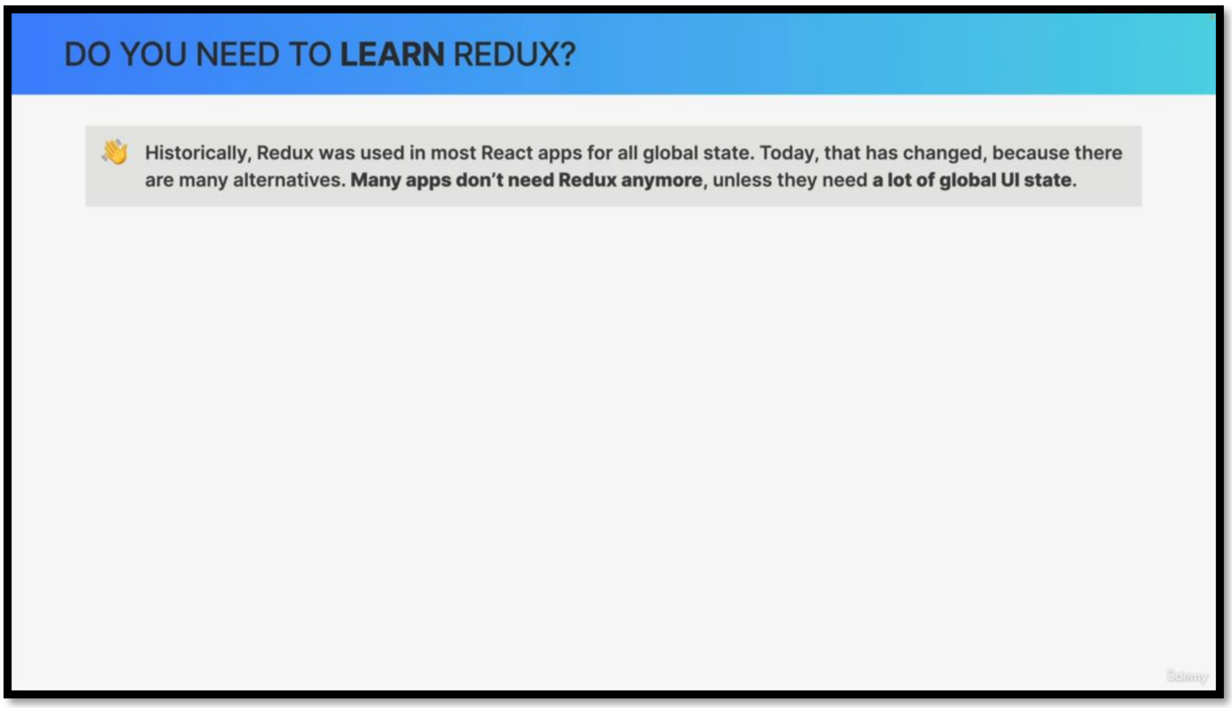
Global store is updated → All consuming components re-render

☝️ You need to have a really good understanding of the `useReducer` hook in order to understand Redux!

Redux has a long history and there are basically two versions of Redux.

So, two different ways of writing Redux applications, but which are totally compatible with each other.

So, we have the classic Redux and we have the more modern way of writing Redux with Redux Toolkit.



## DO YOU NEED TO **LEARN** REDUX?

👋 Historically, Redux was used in most React apps for all global state. Today, that has changed, because there are many alternatives. **Many apps don't need Redux anymore**, unless they need **a lot of global UI state**.

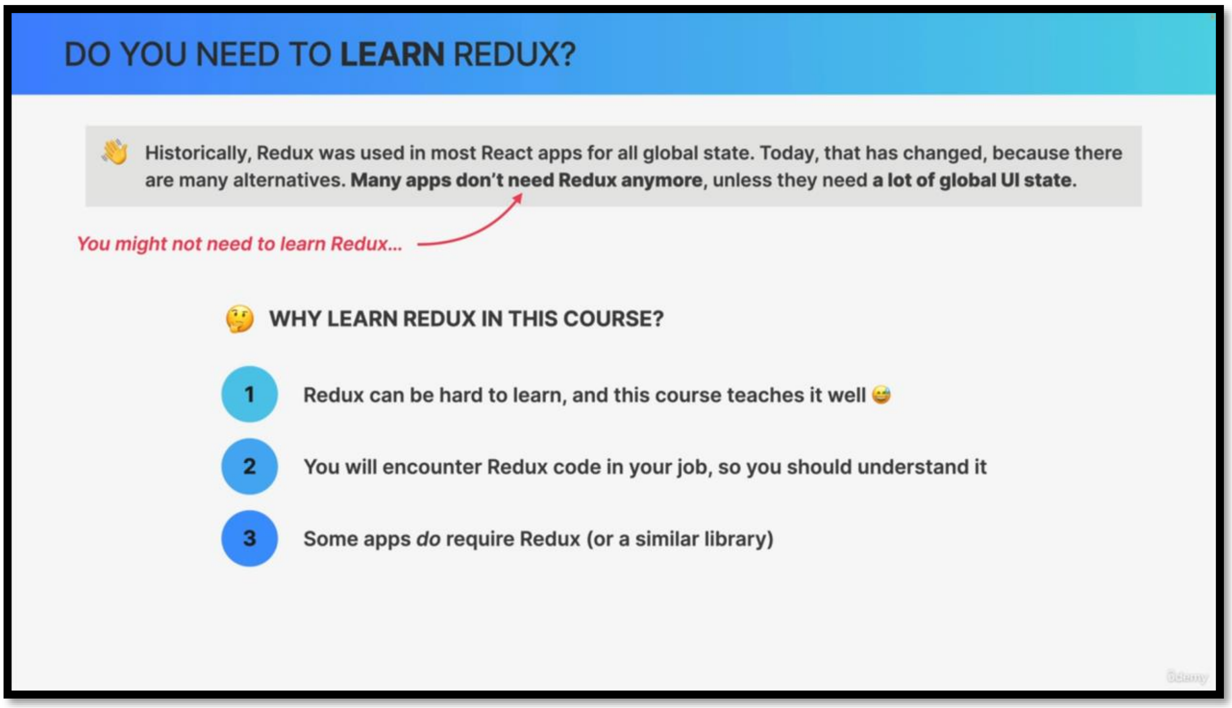
So historically, a few years back, Redux was used in almost every React app out there for all global state management needs.

It was as if Redux was even part of React.

Today, however the landscape has changed tremendously, because there are now many alternatives of state management.

So today, many apps don't actually need Redux anymore and also don't use Redux anymore and this means that you might not even need to learn Redux right now.

So why is it included?

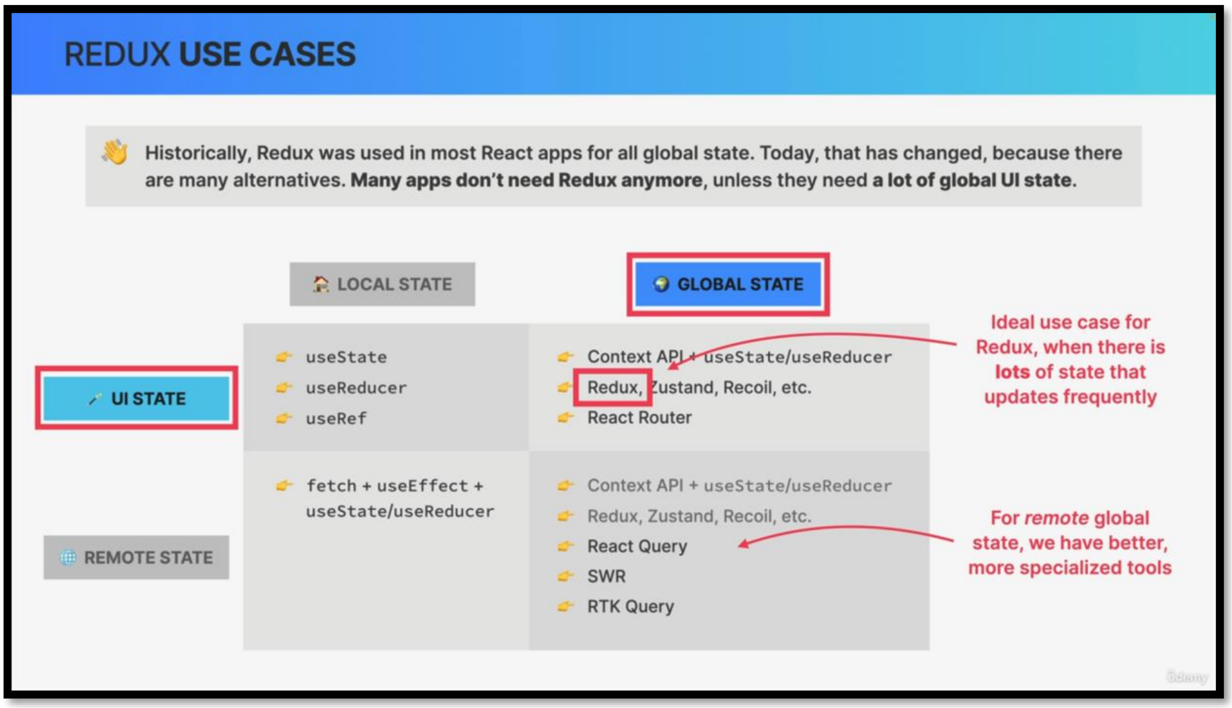

## DO YOU NEED TO LEARN REDUX?

👋 Historically, Redux was used in most React apps for all global state. Today, that has changed, because there are many alternatives. **Many apps don't need Redux anymore**, unless they need **a lot of global UI state**.

*You might not need to learn Redux...*

🤔 **WHY LEARN REDUX IN THIS COURSE?**

1. Redux can be hard to learn, and this course teaches it well 😅
2. You will encounter Redux code in your job, so you should understand it
3. Some apps *do* require Redux (or a similar library)


## REDUX USE CASES

👋 Historically, Redux was used in most React apps for all global state. Today, that has changed, because there are many alternatives. **Many apps don't need Redux anymore**, unless they need **a lot of global UI state**.

| | LOCAL STATE | GLOBAL STATE |
|---|---|---|
| UI STATE | useState<br>useReducer<br>useRef | Context API + useState/useReducer<br>Redux, Zustand, Recoil, etc.<br>React Router |
| REMOTE STATE | fetch + useEffect + useState/useReducer | Context API + useState/useReducer<br>Redux, Zustand, Recoil, etc.<br>React Query<br>SWR<br>RTK Query |

Ideal use case for Redux, when there is **lots** of state that updates frequently

For *remote* global state, we have better, more specialized tools

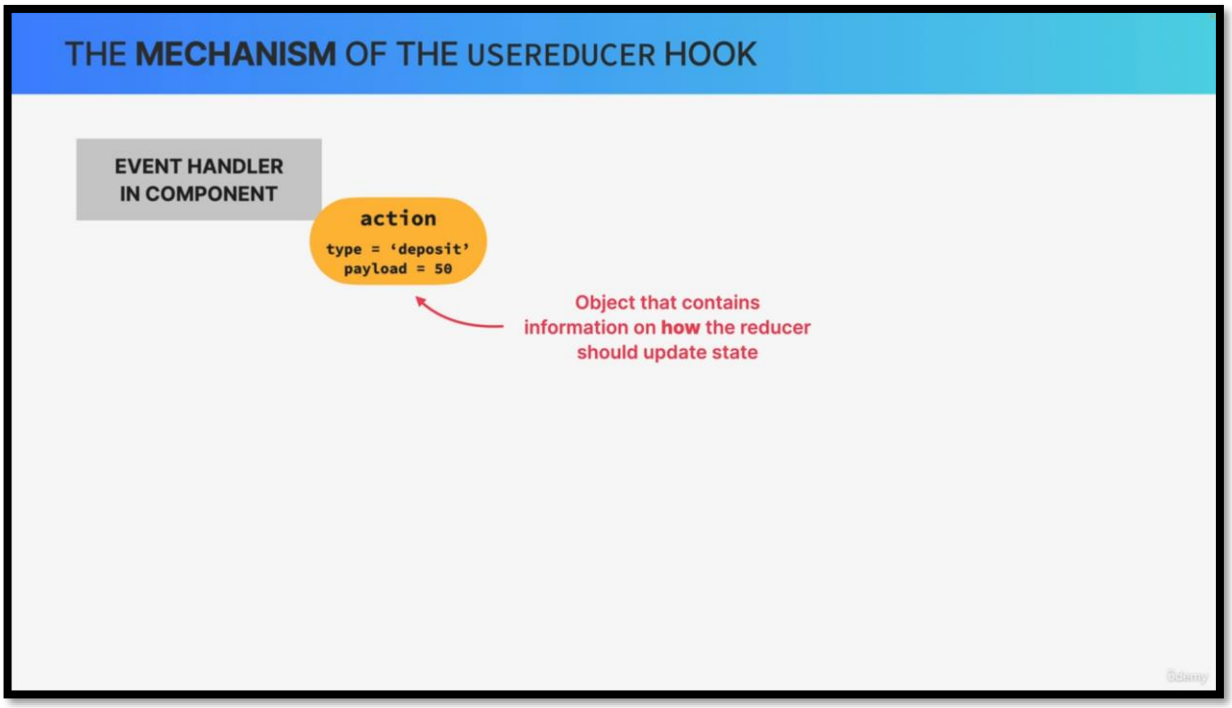
The recommendation of many experts is to use a global state management library like Redux, only when you have lots of global UI state that you need to update frequently.

UI state basically means state that is not fetched from a server that would be remote state. So, UI state is simply data about the UI itself or data that doesn't need to be communicated to an API or so.

This distinction is really important, because remember for global remote state we have many better options nowadays like React Query, SWR, or even a tool that is kind of included in modern Redux Toolkit, which is Redux Toolkit Query.

So, when you need to manage a lot of non-remote state in your app, then Redux might be perfect for that.

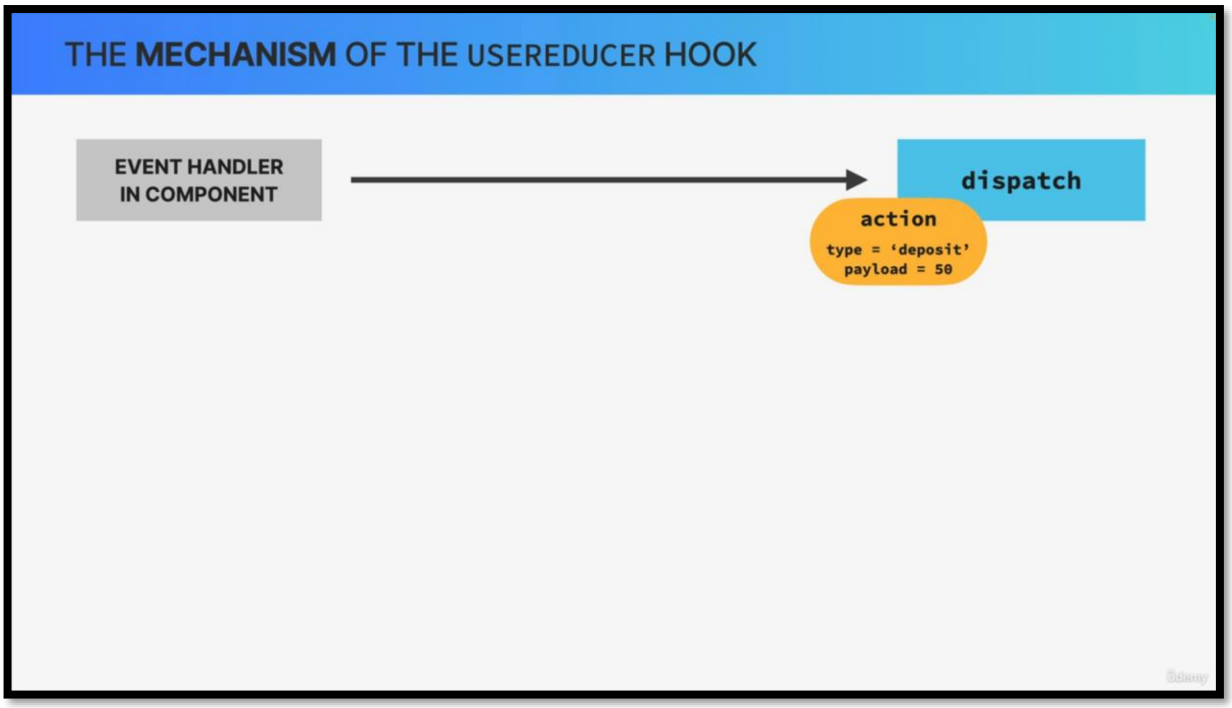
But the truth is that most global state is actually remote which is the reason why so many apps don't use Redux anymore.

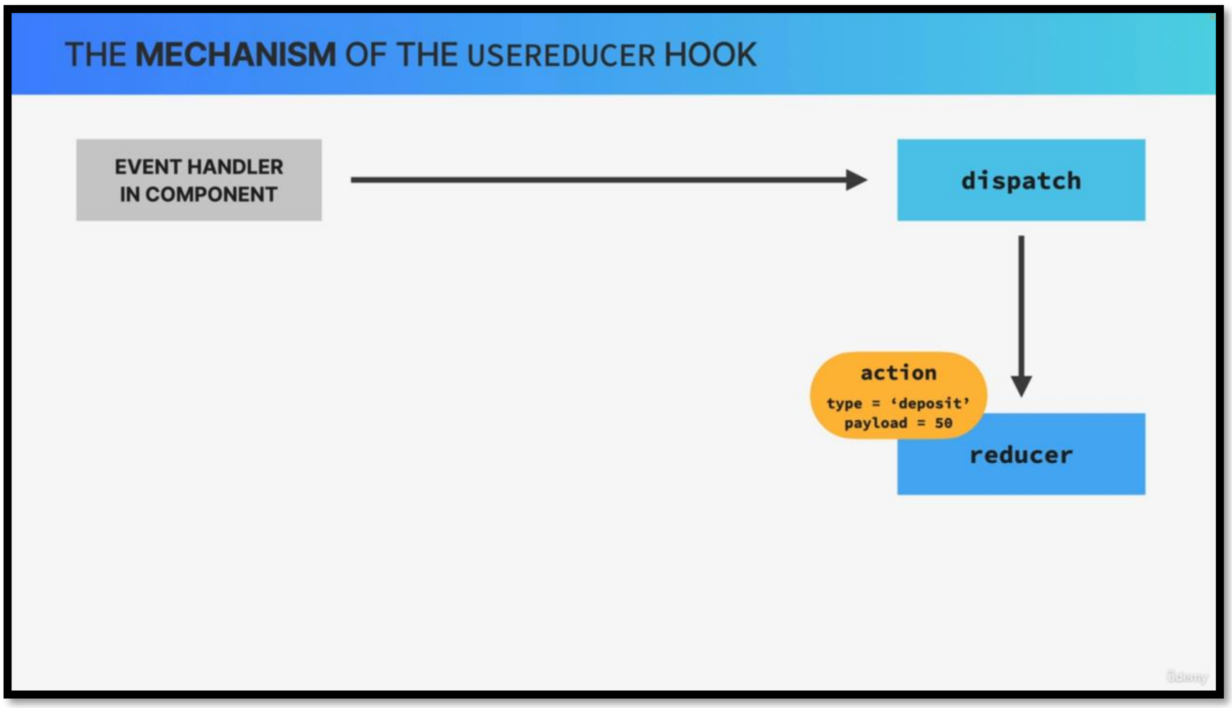


## THE MECHANISM OF THE USEREDUCER HOOK

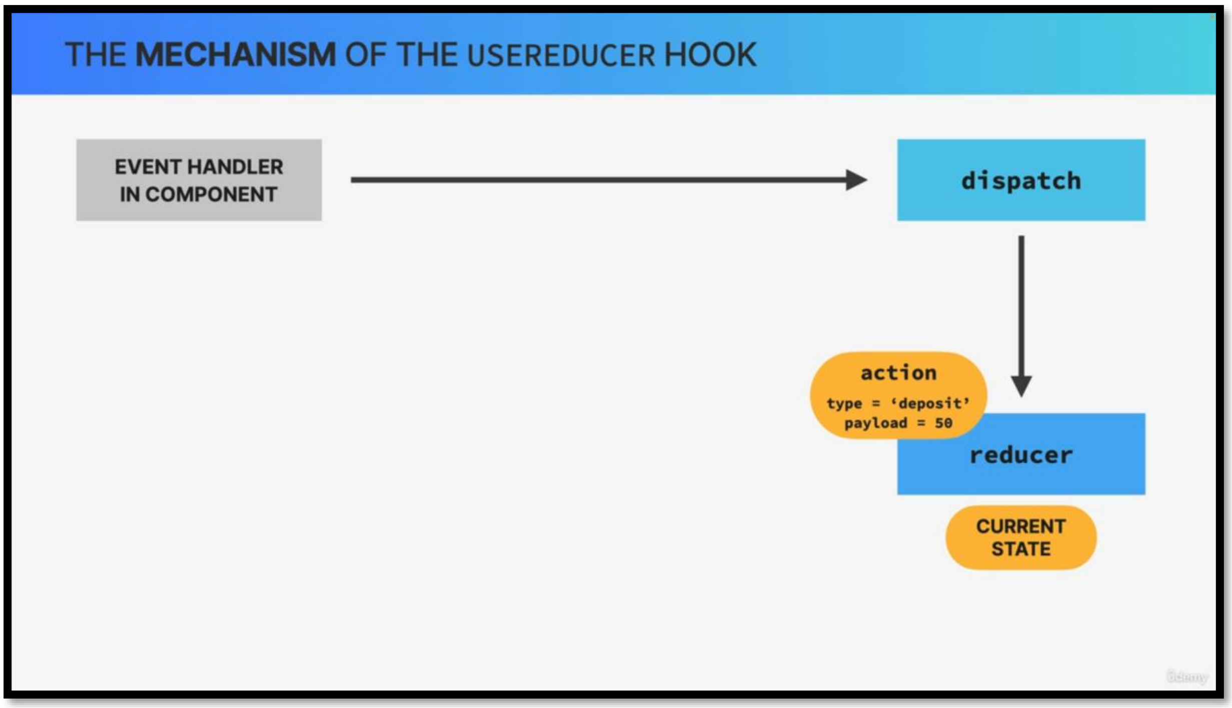
EVENT HANDLER IN COMPONENT
action
type = 'deposit'
payload = 50
Object that contains information on how the reducer should update state

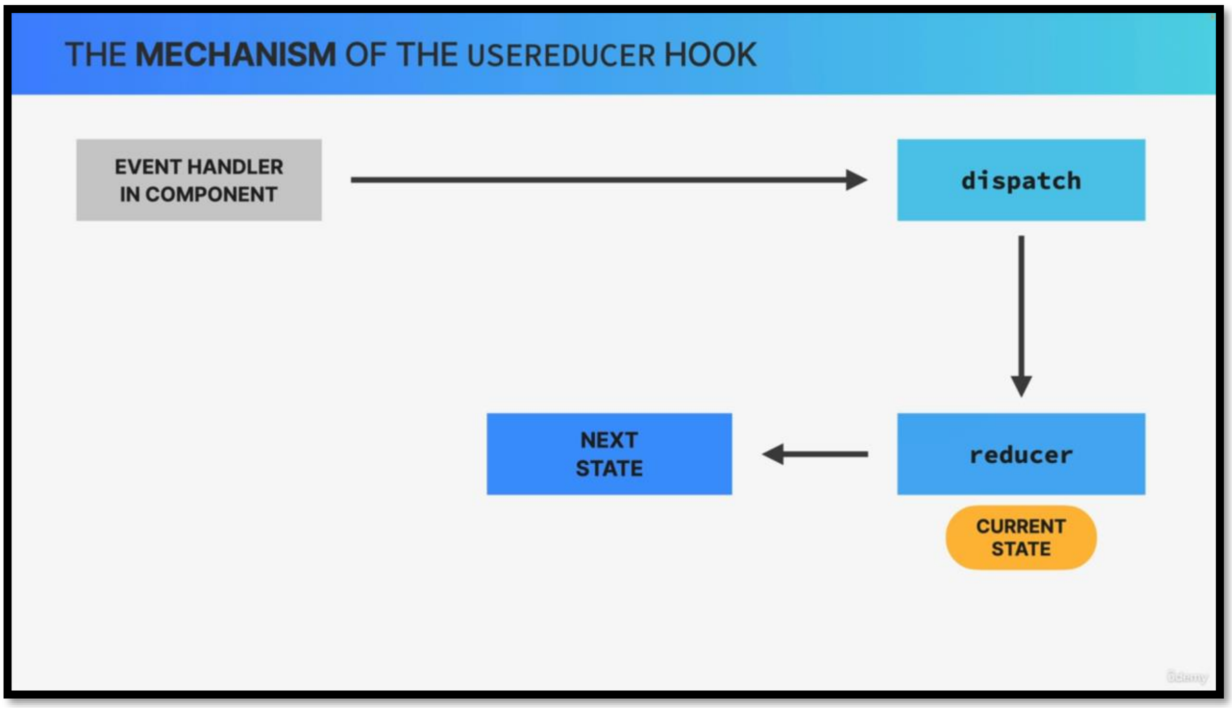
With useReducer when we want to update state from an event handler in a component, we start by creating an action. This action is simply an object that usually contains a type and a payload, which is information about how the state should be updated.


THE MECHANISM OF THE USEREDUCER HOOK
EVENT HANDLER IN COMPONENT
dispatch
action
type = 'deposit'
payload = 50

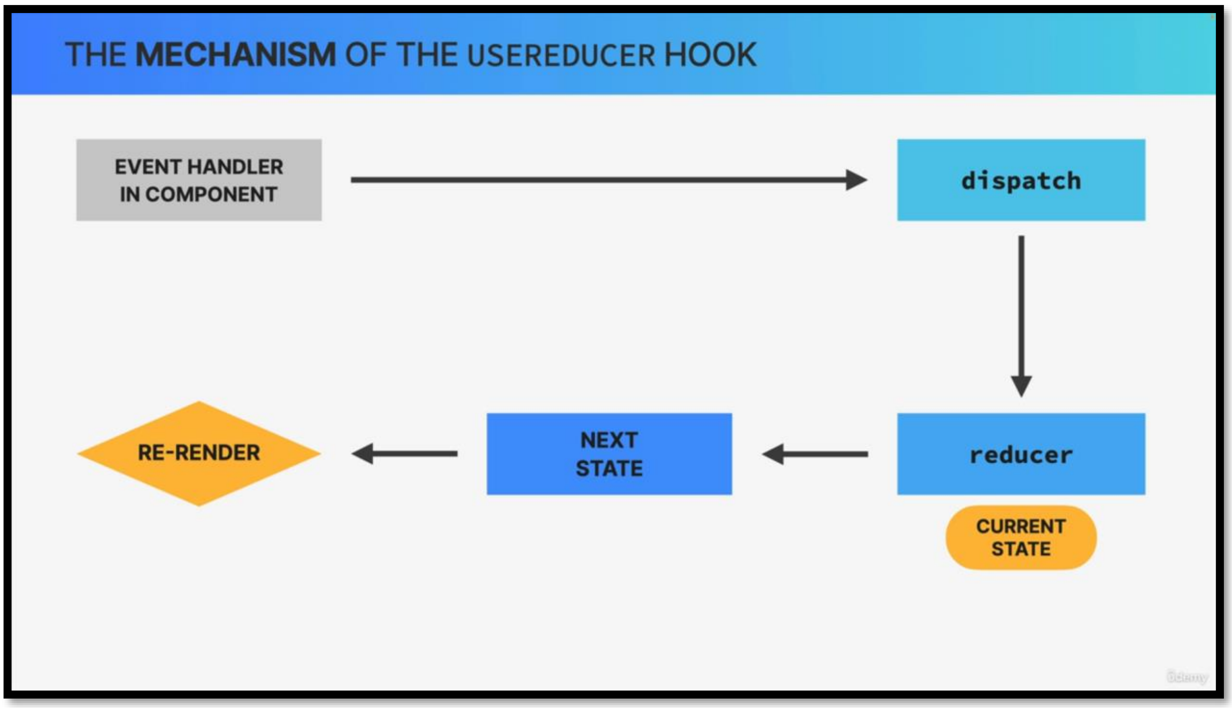

THE MECHANISM OF THE USEREDUCER HOOK
EVENT HANDLER IN COMPONENT
dispatch
action
type = 'deposit'
payload = 50
reducer

We then dispatch that action to a so-called reducer function.


## THE MECHANISM OF THE USEREDUCER HOOK

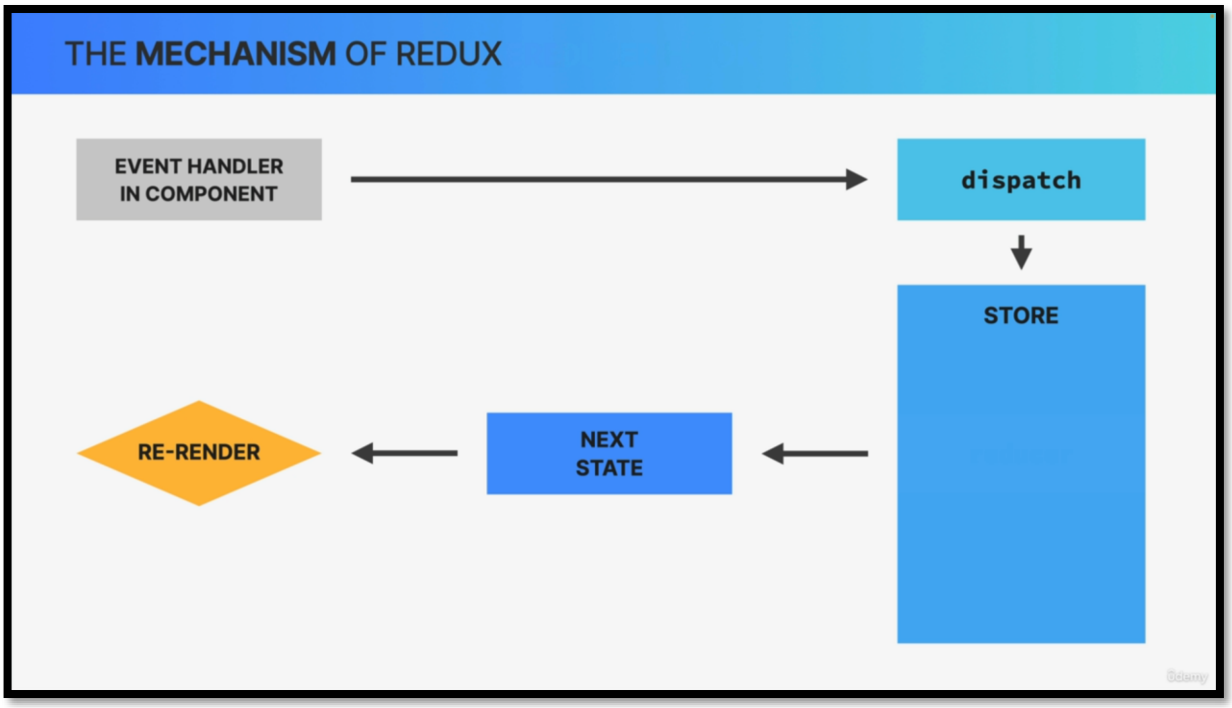

## THE MECHANISM OF THE USEREDUCER HOOK

The reducer takes the action type and the payload, and together with the current state calculates the next state, so the new state value.

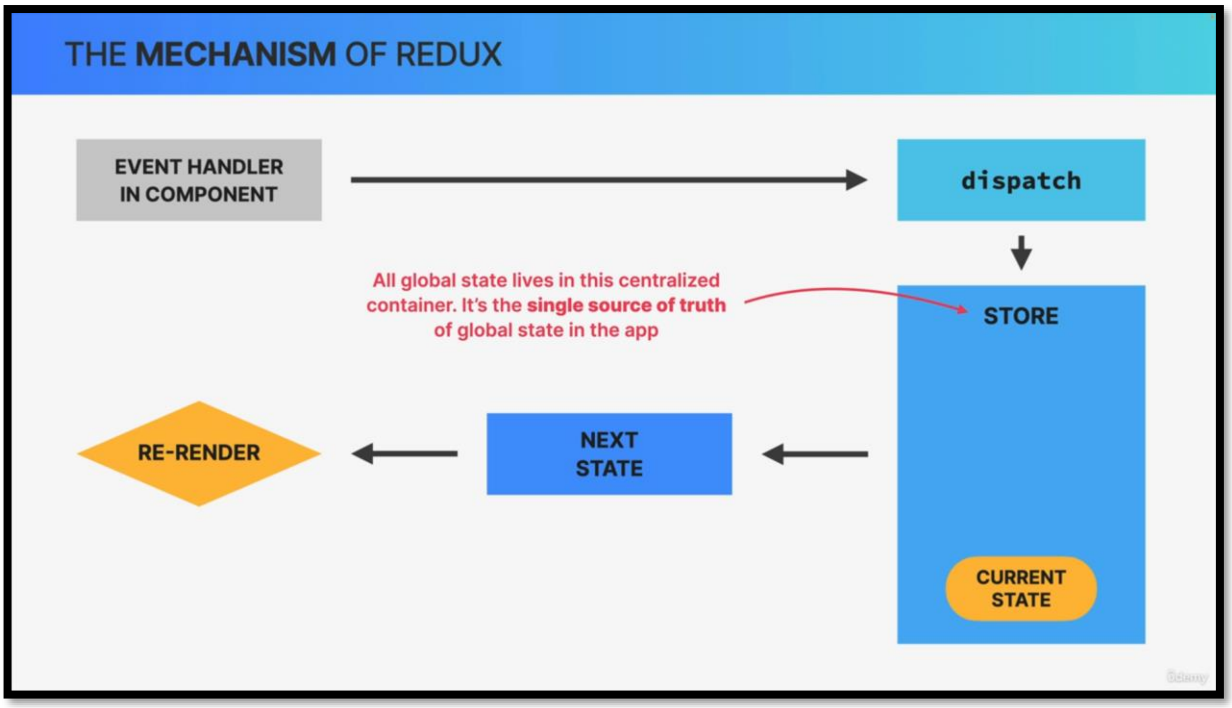

THE MECHANISM OF THE USEREDUCER HOOK
EVENT HANDLER IN COMPONENT
dispatch
reducer
CURRENT STATE
NEXT STATE
RE-RENDER

To finish as the state updates, the component that originated the state transition will re-render.

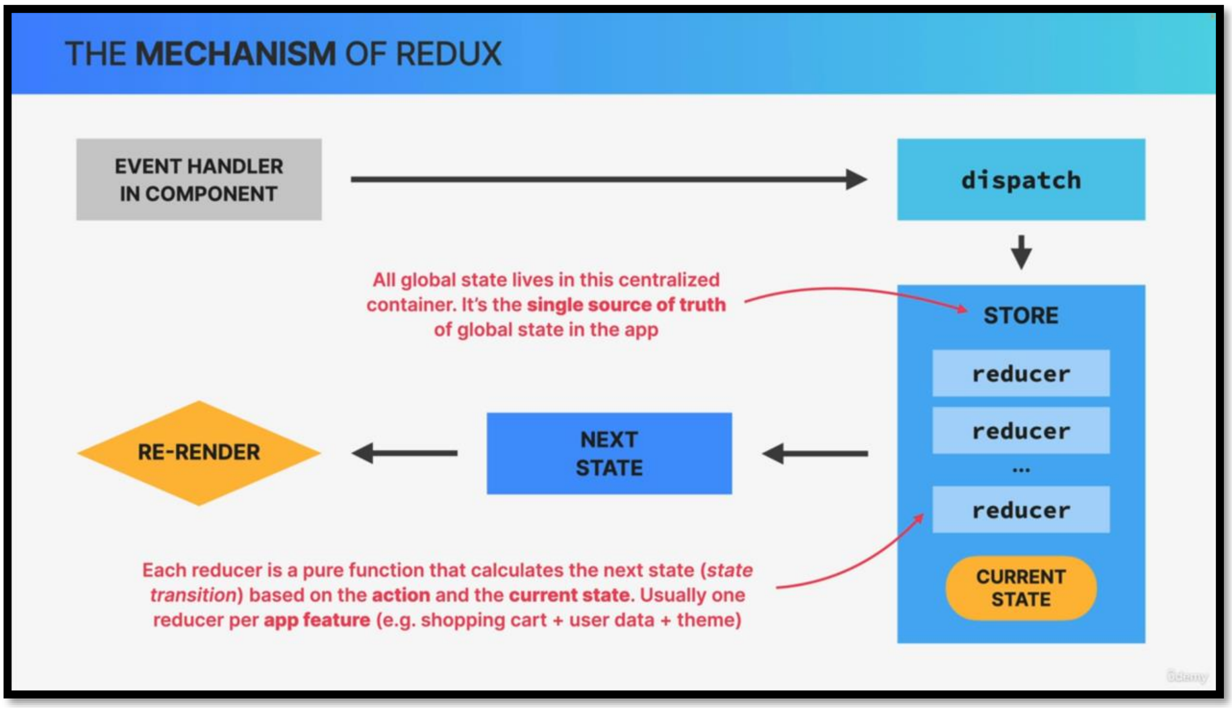
So this mechanism should be familiar to you at this point, because now we're gonna add two more things onto this in order to learn how Redux works.



So, the first difference between useReducer and Redux is that in Redux we actually dispatch actions not simply to the reducer.


THE MECHANISM OF REDUX
EVENT HANDLER IN COMPONENT
dispatch
All global state lives in this centralized container. It's the single source of truth of global state in the app
STORE
RE-RENDER
NEXT STATE
CURRENT STATE

So, this store is a centralized container where all global state lives. It's like the single source of truth of all global state across the entire application.

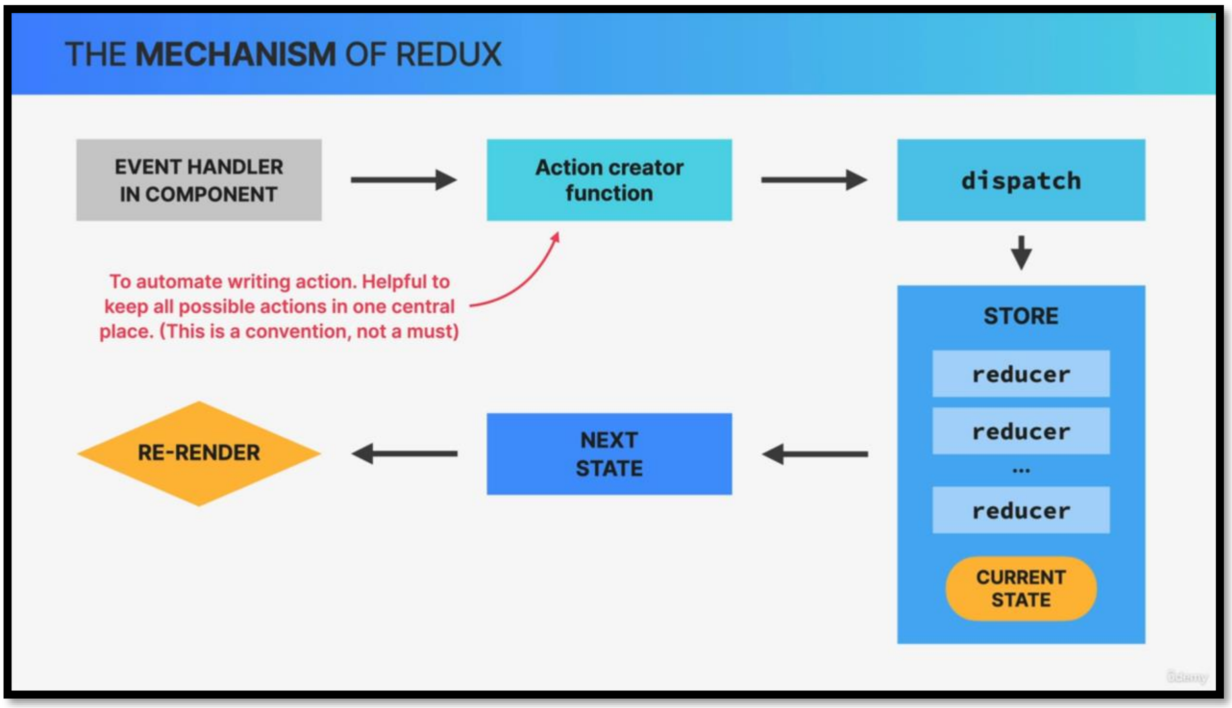

THE MECHANISM OF REDUX
EVENT HANDLER IN COMPONENT
dispatch
All global state lives in this centralized container. It's the single source of truth of global state in the app
STORE
reducer
reducer
...
reducer
RE-RENDER
NEXT STATE
CURRENT STATE
Each reducer is a pure function that calculates the next state (state transition) based on the action and the current state. Usually one reducer per app feature (e.g. shopping cart + user data + theme)

The store is also where one or multiple reducers live and each reducer must be a pure function that has the single task of calculating the next state based on the action that was dispatched to the store and the current state that's already in the store as well.

Now, you might be wondering why there are multiple reducers in this store.

Well, it's because we should create one reducer per application feature or per data domain in order to keep things separated. For example, in a shopping app, you could have one reducer for the shopping cart, one for some user data, and one for the application color theme.

Finally, any component that consumes the state that has been updated in the store will as always get re-rendered by React.


THE MECHANISM OF REDUX
EVENT HANDLER IN COMPONENT
Action creator function
dispatch
To automate writing action. Helpful to keep all possible actions in one central place. (This is a convention, not a must)
STORE
reducer
reducer
...
reducer
CURRENT STATE
NEXT STATE
RE-RENDER

So, in the real world when we use Redux, we usually use functions called action creators in order to automate the process of writing actions.

So, basically, instead of always writing these action objects by hand, we create functions that do this automatically.

This has the advantage to keep all actions in one central place which reduces bugs, because developers don't have to remember the exact action type strings.

Just note that this is optional and not a feature of Redux. It's just how we build real world Redux apps.

The big goal of all this is to make the state update logic separate from the rest of the application.