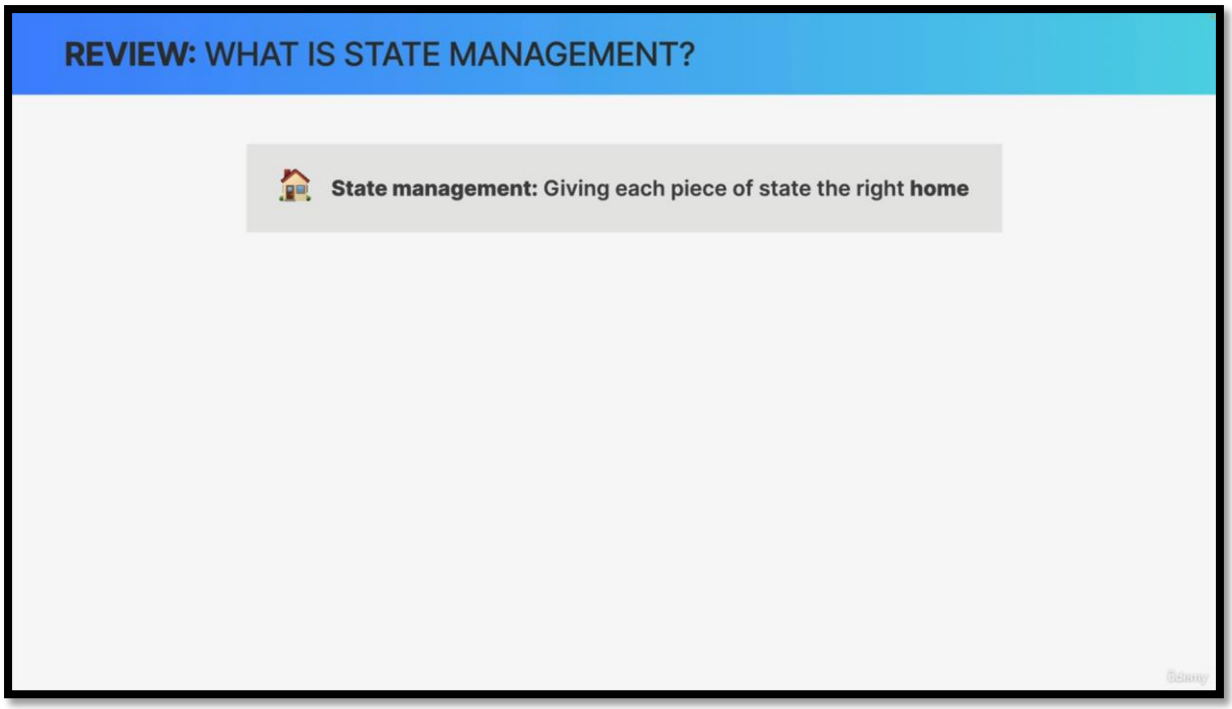
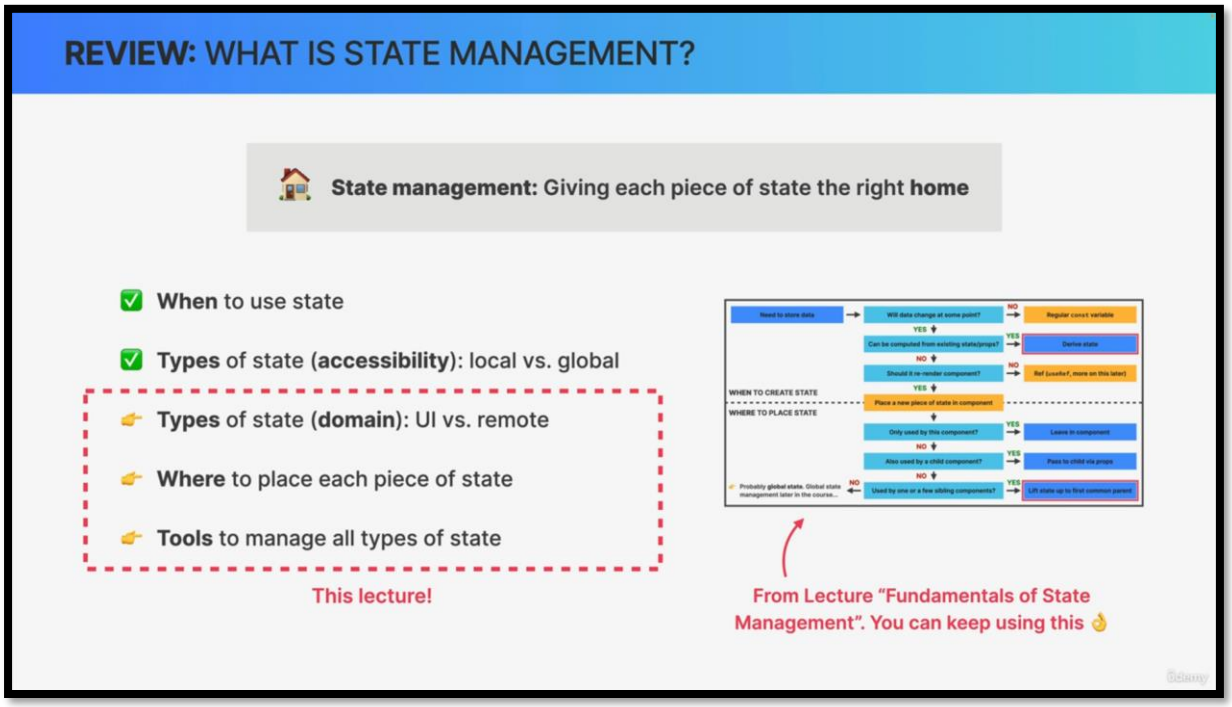
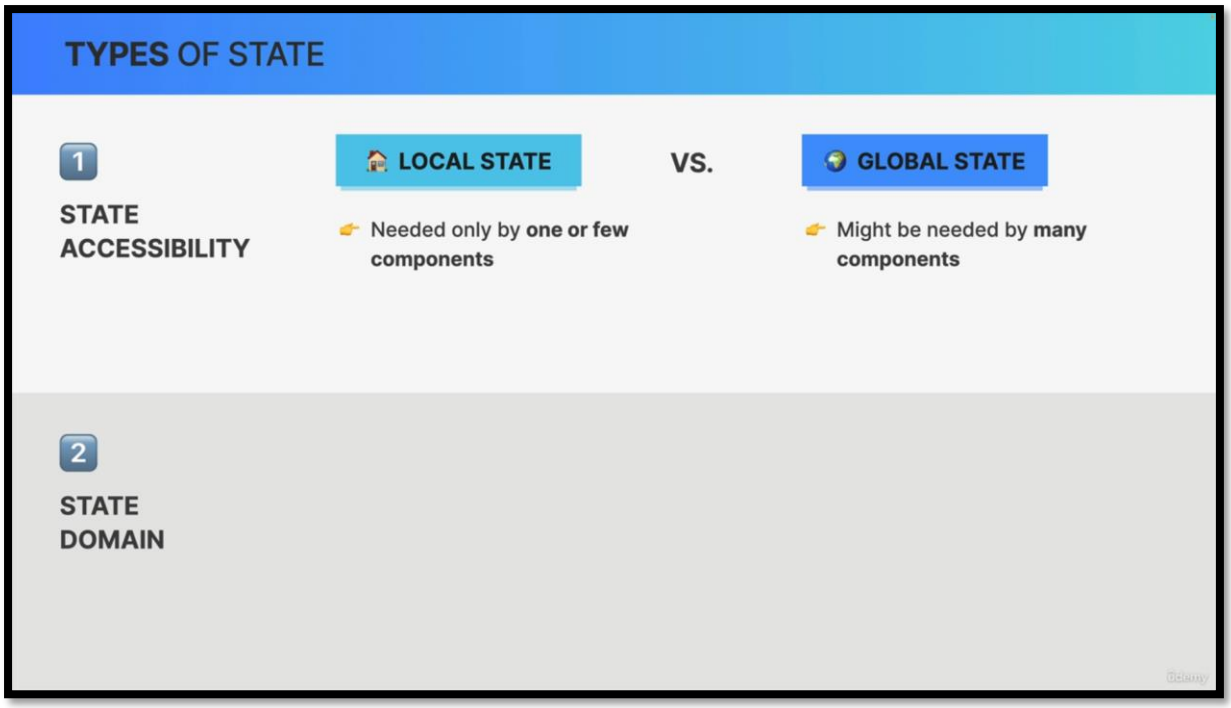


State management is basically just like giving each piece of state the right home.



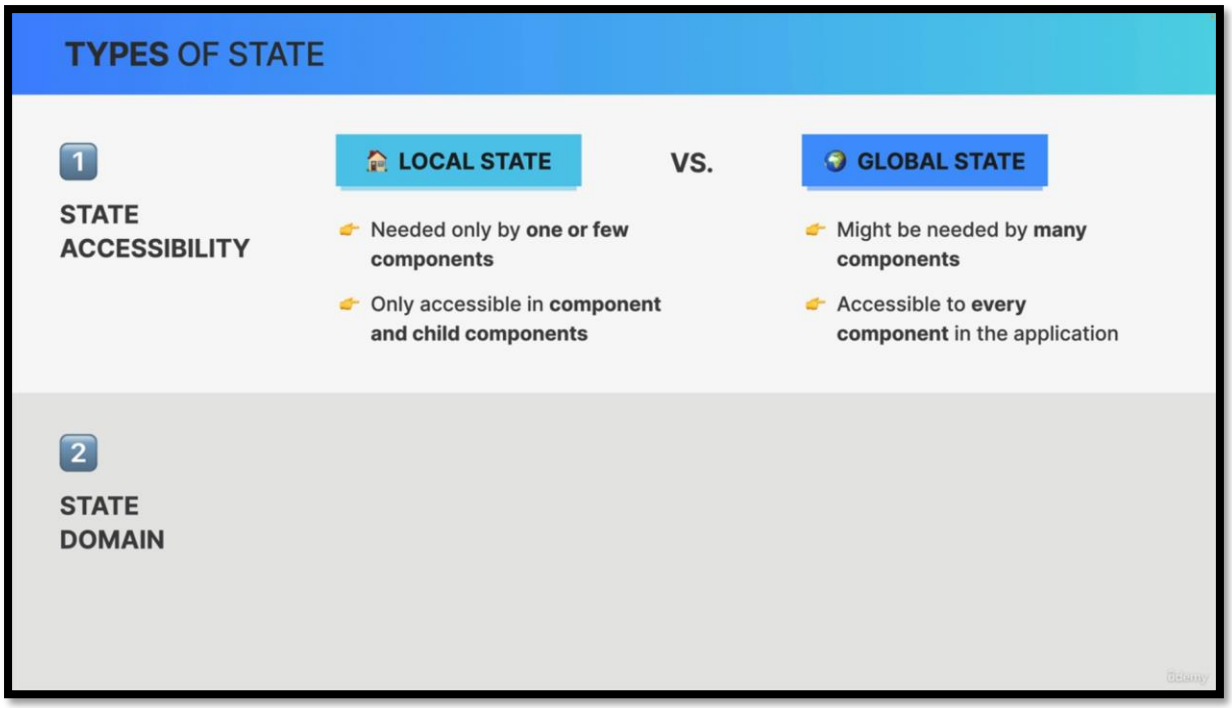


Let's actually start with these different types of state that can exist in all web applications.

We can classify state in terms of state accessibility and in terms of state domain.

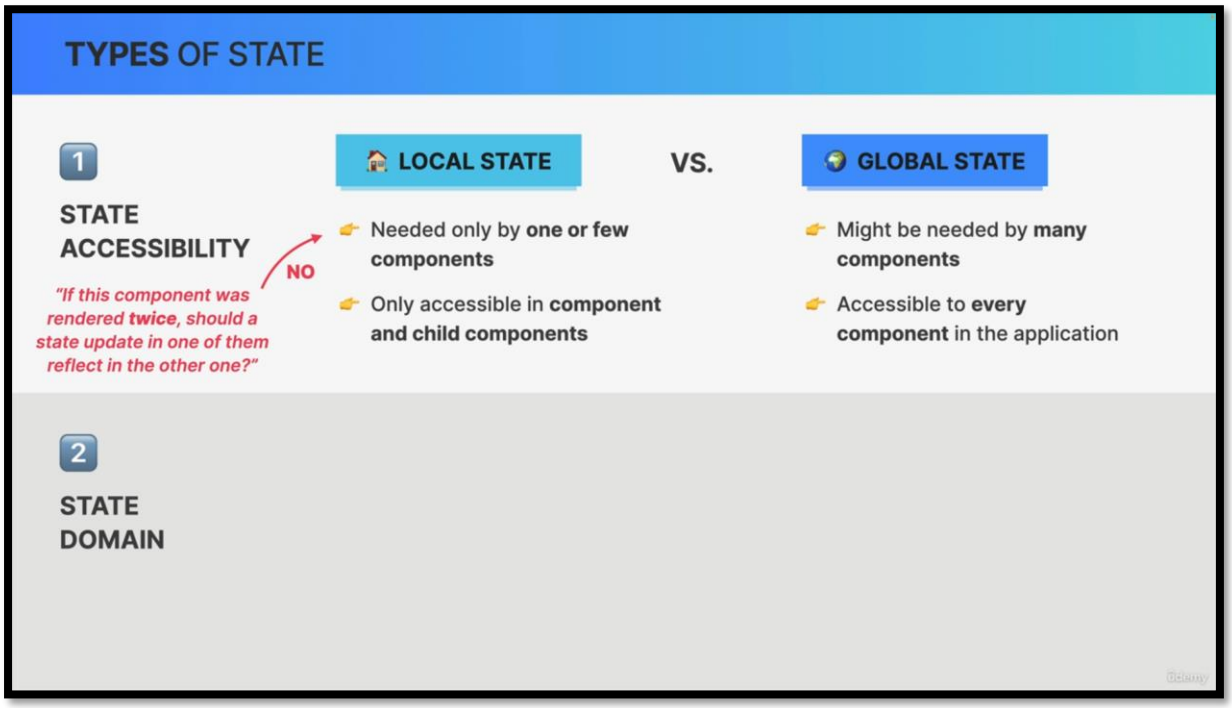
When it comes to state accessibility, state can be either a local state or global state.

So local state is only needed by one or a few components while global state might be needed by many components in different positions of the tree.



That's why local state is only accessible inside the component that it was defined in plus maybe its child components while global state can actually be made accessible to every single component in the application.

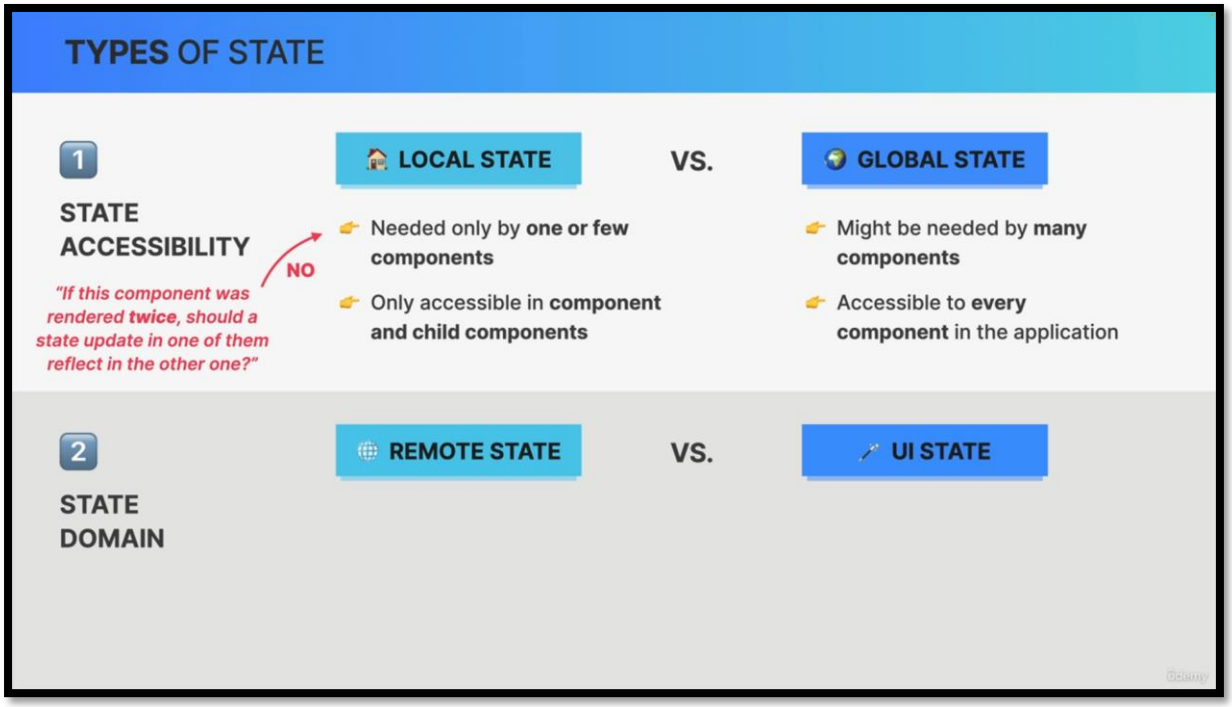
So, it's the accessibility of a state variable that changes between local and global state.



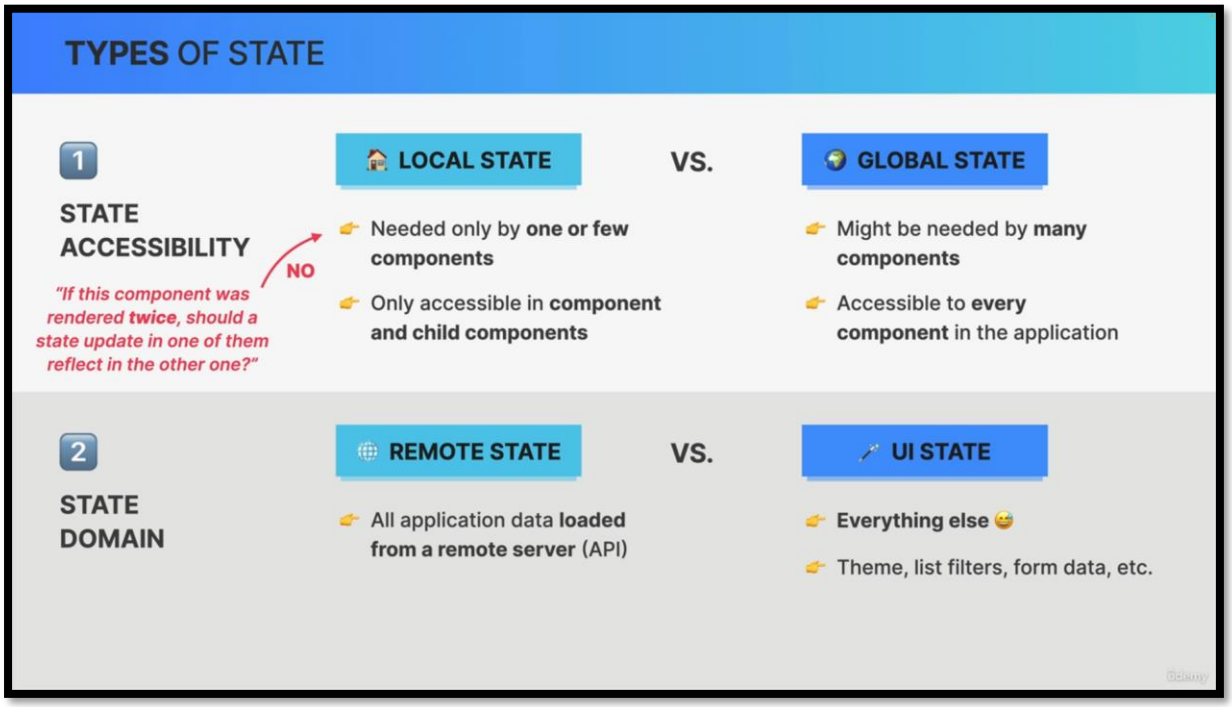
Now if you need to create a new state variable in a component but are not sure whether it should be local or global state, there is a very nice trick that can help.

So, all you need to do is to ask yourself if this component was rendered twice, should a state update in one of the components reflect in the other one?

If the answer is no, then it means that it's local state but if the answer is yes, you have found yourself a global state variable.



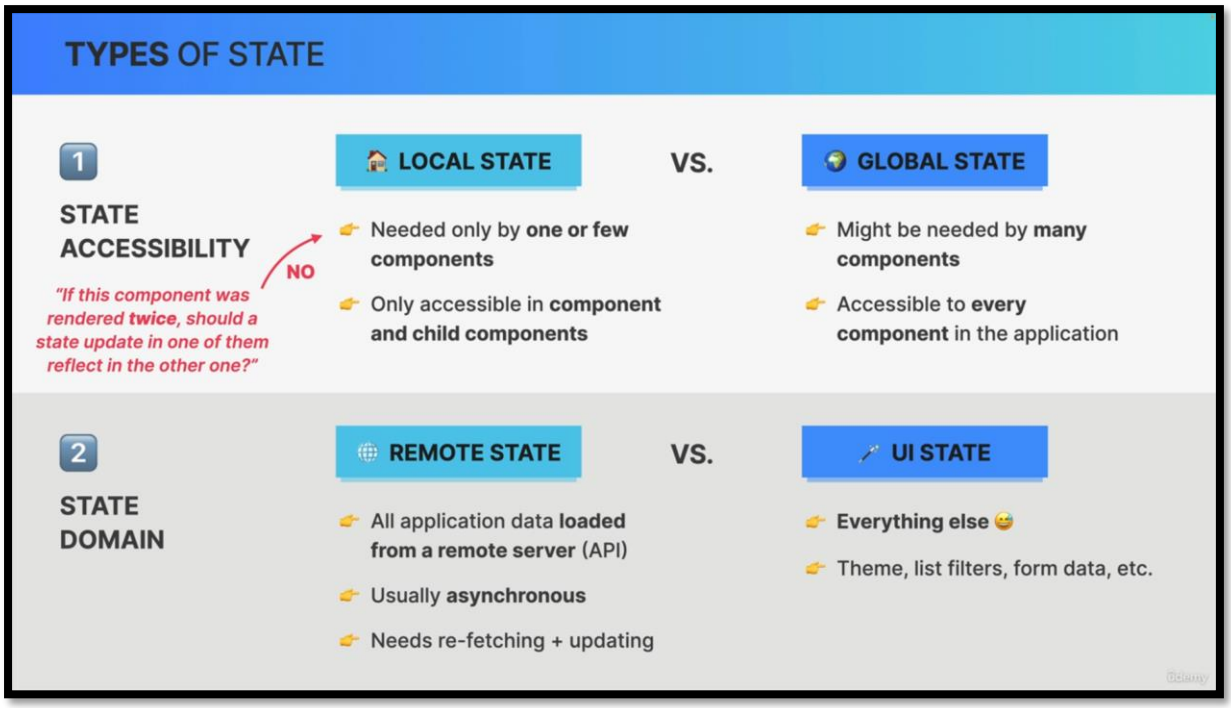
On the basis of state domain, we can classify each piece of state as either remote state or UI state.



So remote state is basically all application data that is loaded from a remote server. So usually using an API. So, it's basically state that lives on a server but that can be loaded into our application.

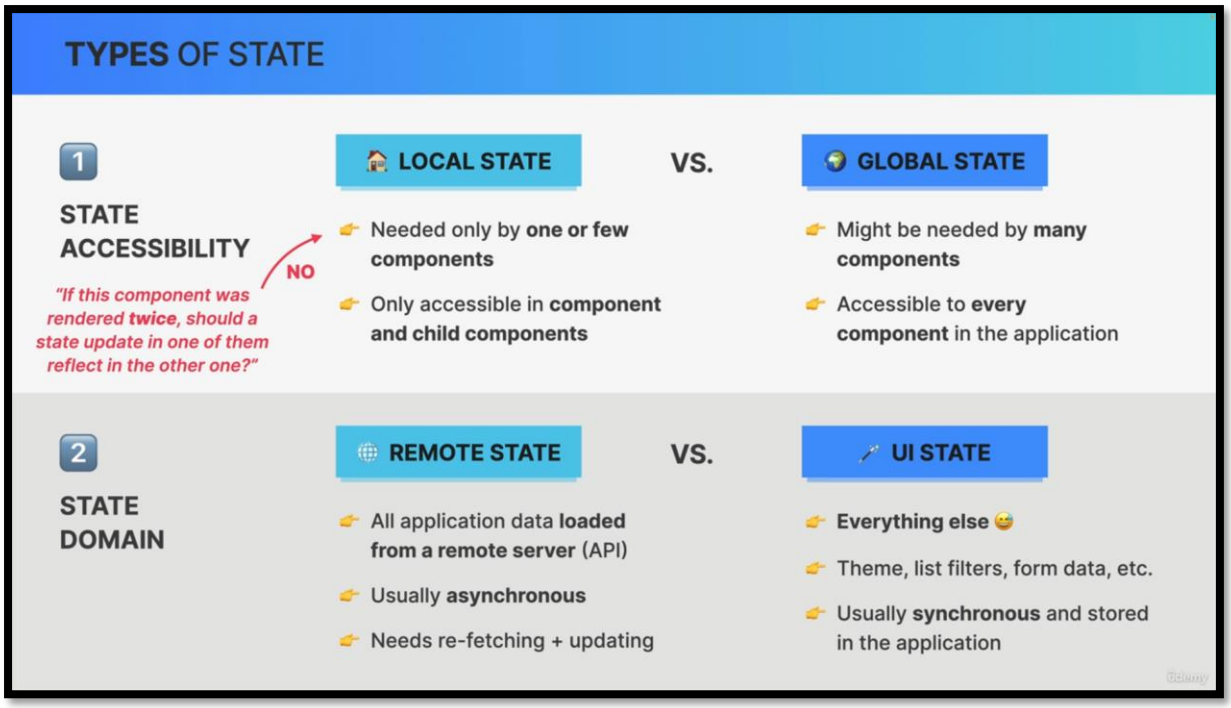
UI state on the other hand is basically everything else. So, things like the currently selected theme, applied list filters, form inputs, open panels and many, many more.

All state that is not the core application data that we usually fetch from an API is UI state. So, whenever you have some state, it's extremely important to always know whether you're dealing with remote state or with UI state because they need to be managed in a completely different way.



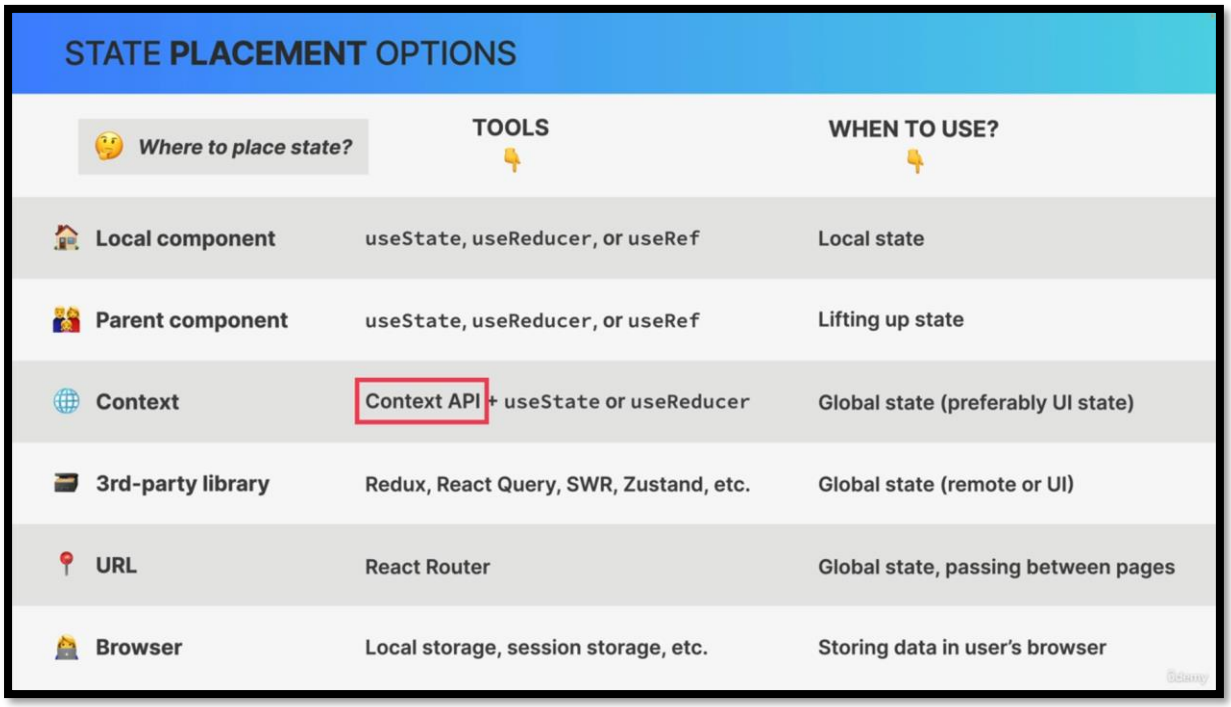
So remote state is fundamentally different from UI state because we usually get it in an asynchronous way and because the data might need to be re-fetched and updated frequently.

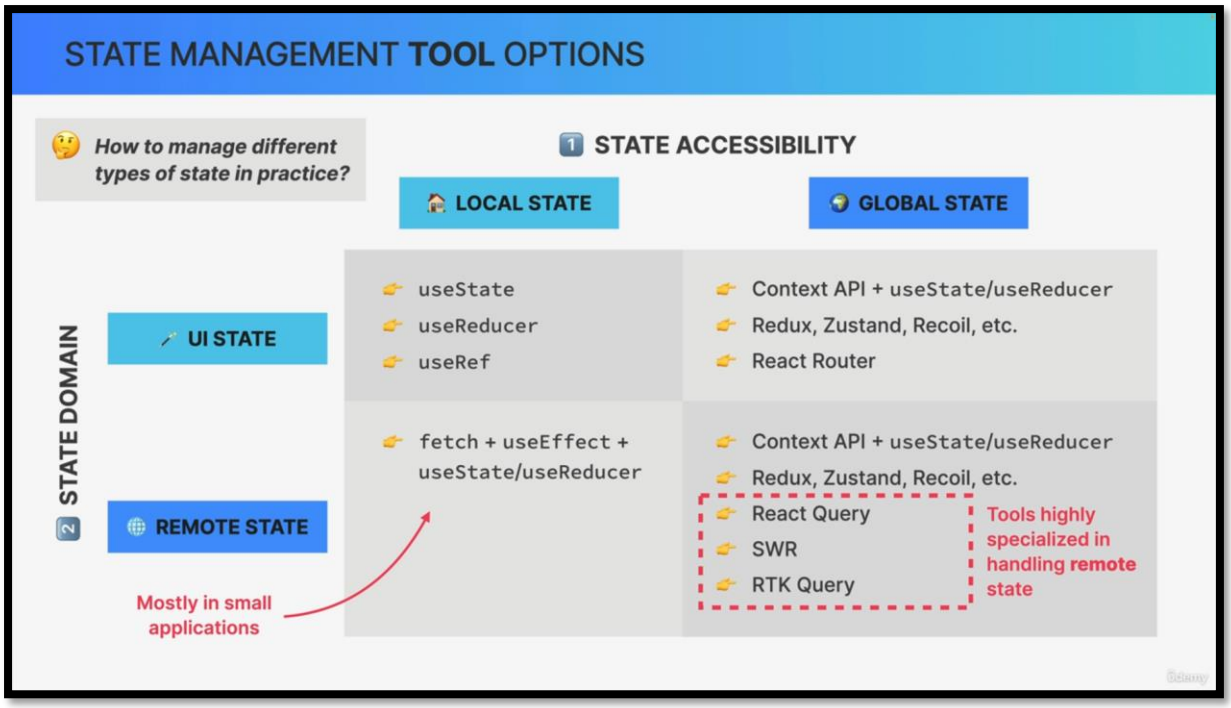
Therefore, in a large scale application, remote state should be cached, revalidated and so on and for that, we need some specialized tools.



UI state on the other hand is usually synchronous and stored right in the application and does not interact with any server at all.

This means that UI State is very easy to handle with the tools that we already know of. So, things like useState or useReducer.




STATE MANAGEMENT TOOL OPTIONS
How to manage different types of state in practice?
1 STATE ACCESSIBILITY
LOCAL STATE
GLOBAL STATE
2 STATE DOMAIN
UI STATE
useState
useReducer
useRef
Context API + useState/useReducer
Redux, Zustand, Recoil, etc.
React Router
REMOTE STATE
fetch + useEffect + useState/useReducer
Context API + useState/useReducer
Redux, Zustand, Recoil, etc.
React Query
SWR
RTK Query
Tools highly specialized in handling remote state
Mostly in small applications