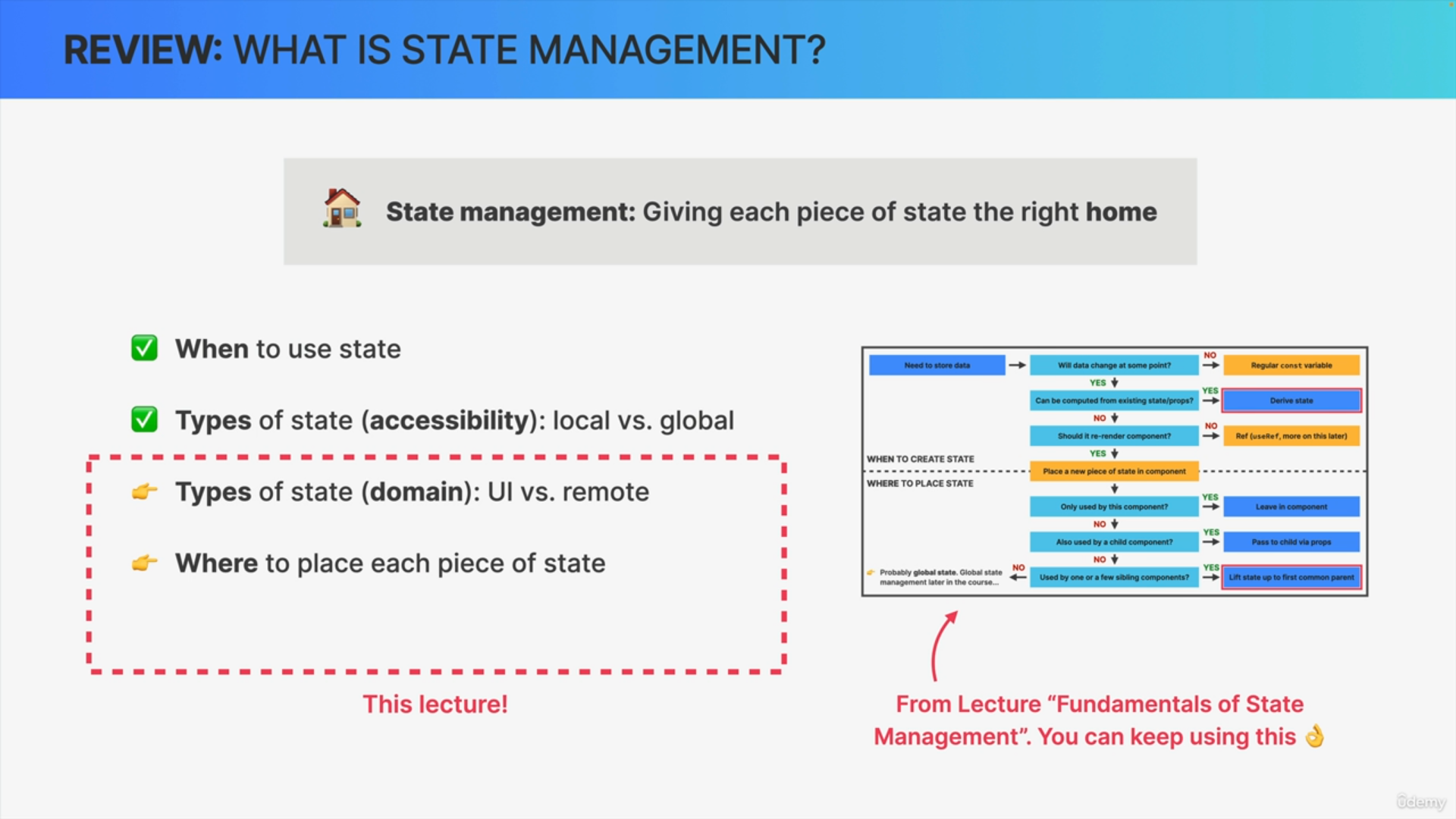
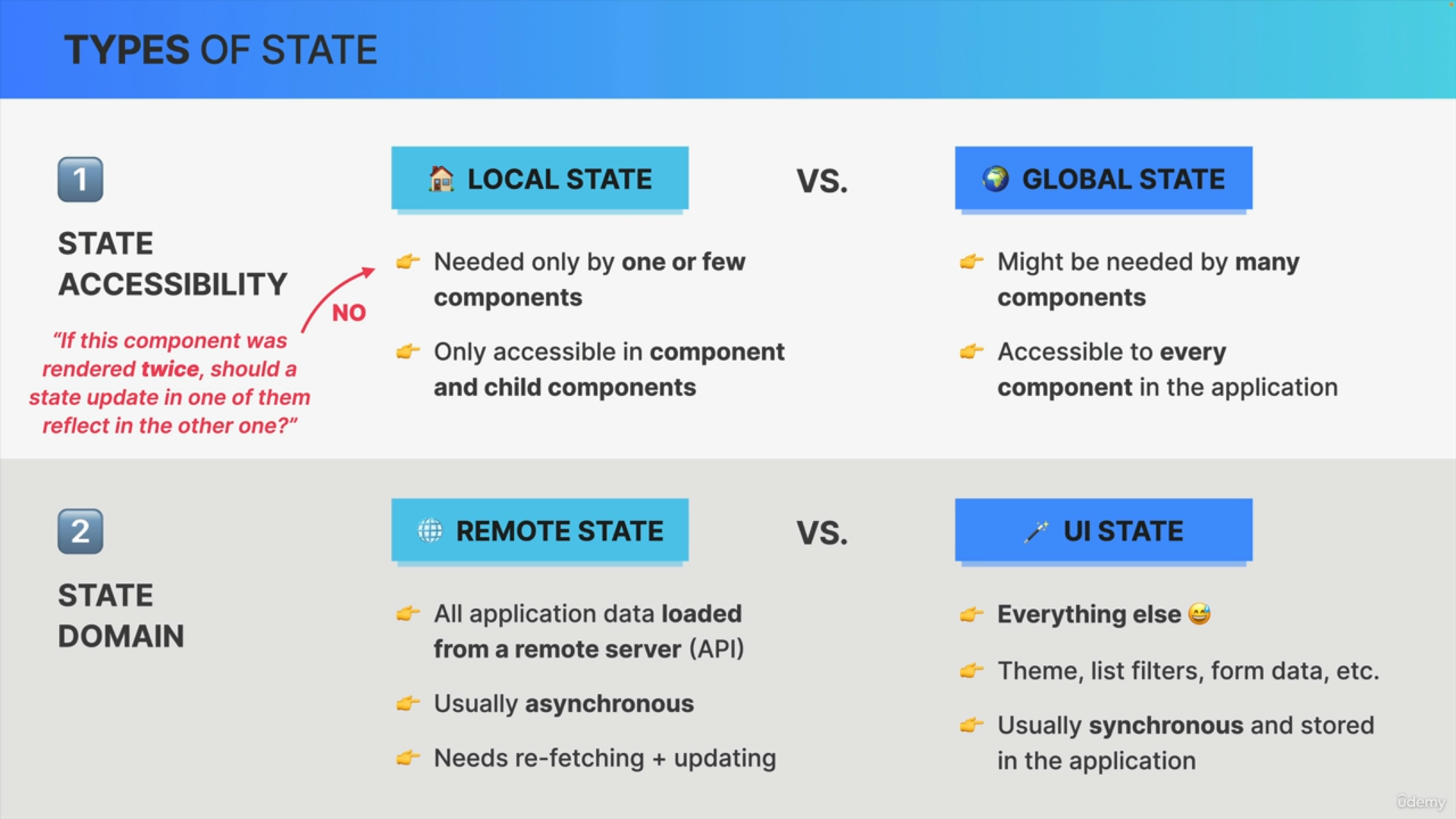
**Types of State**

* Accessibility: Local State (scoped to a component and its children) vs. Global State (accessible to many components across the application).
* Domain: UI State (local state for UI interactions like form inputs, filters) vs. Remote State (data fetched asynchronously from a server).



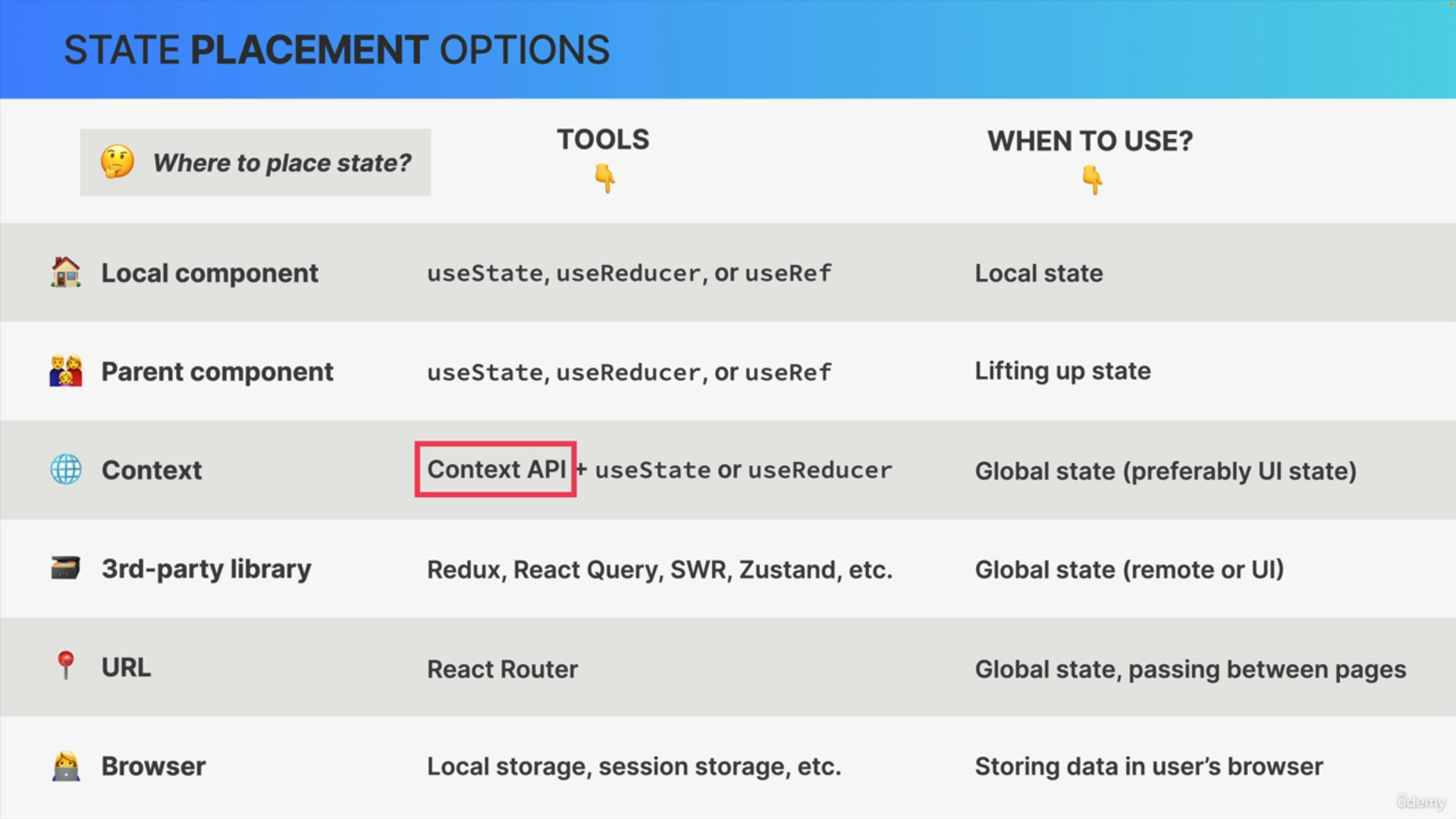


**Decision Criteria**

Use the "render twice" trick to determine if state should be local (changes should not affect another instance) or global (changes should reflect across instances).

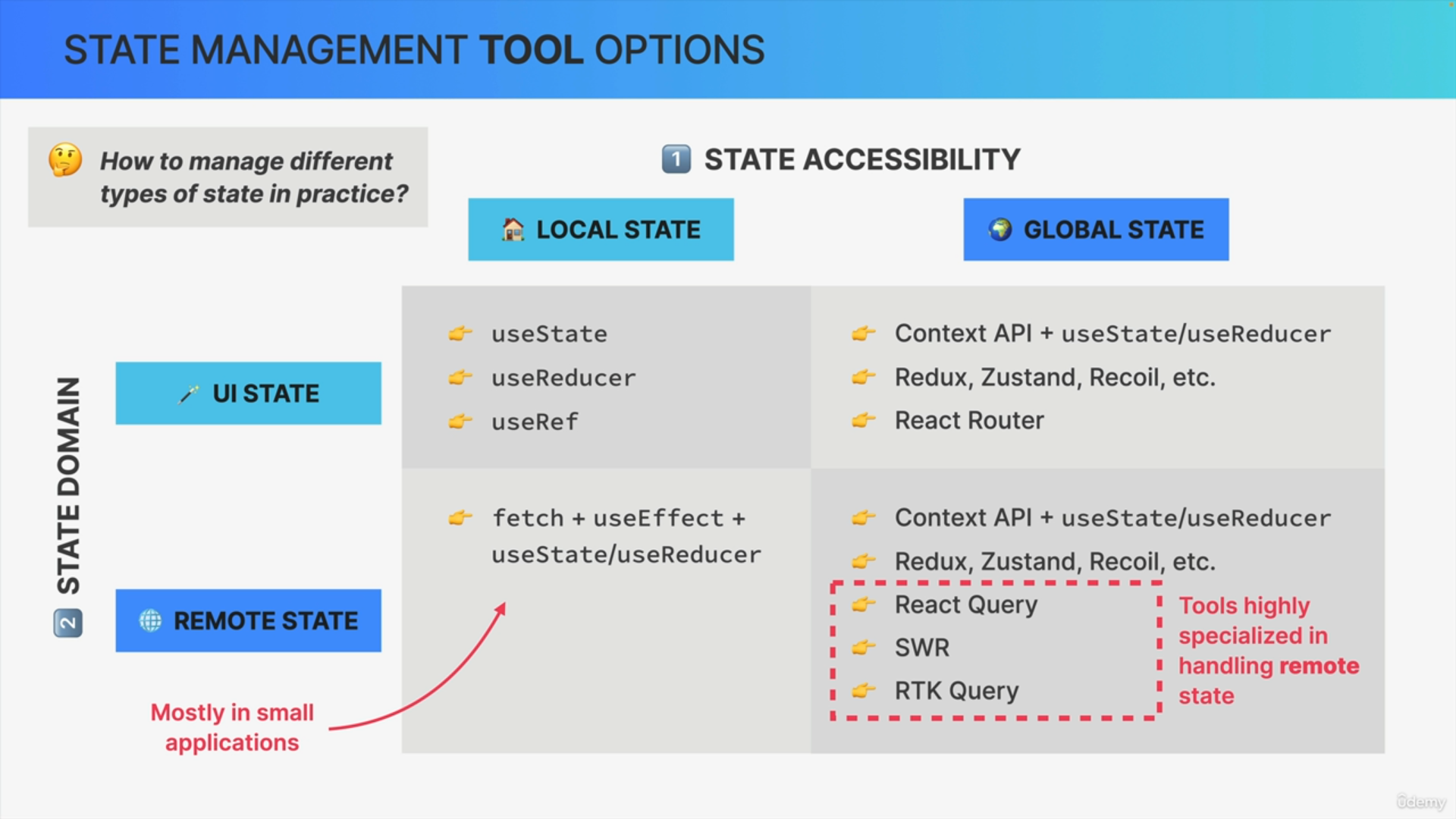
**Placement Options for State**

* Local State: Managed within a component using useState, useReducer, or useRef.
* Parent Component State: Lift state up to a common ancestor of components that need access.
* Global State: Managed with the Context API for UI state or third-party libraries like Redux, React Query, SWR, or Zustand for more specialized global state management.
* URL: Use for shareable global state (via query parameters).
* Local Storage / Session Storage: Store state data within the browser for persistence across sessions.



**Tools for State Management**

* Local UI State: Managed with useState, useReducer, or useRef.
* Local Remote State: Use useEffect with fetch to load data from APIs and manage with useState or useReducer.
* Global Remote State: Managed using Context API coupled with useState or useReducer, or specialized tools like Redux, React Query, SWR, or RTK Query for efficient data fetching, caching, and revalidation.
* Global UI State: Context API combined with useState or useReducer, or third-party libraries like Redux for performance optimization and centralized state management.



**Recommendations**

* Choose tools based on the scale and complexity of the application.
* Redux and React Query are particularly useful for managing large-scale global state and remote data respectively.

**Conclusion**

Understanding where and how to manage different types of state in React applications is crucial for maintaining scalability, performance, and code clarity. By choosing the right tools and placement strategies, developers can effectively manage state across various components and application scales.