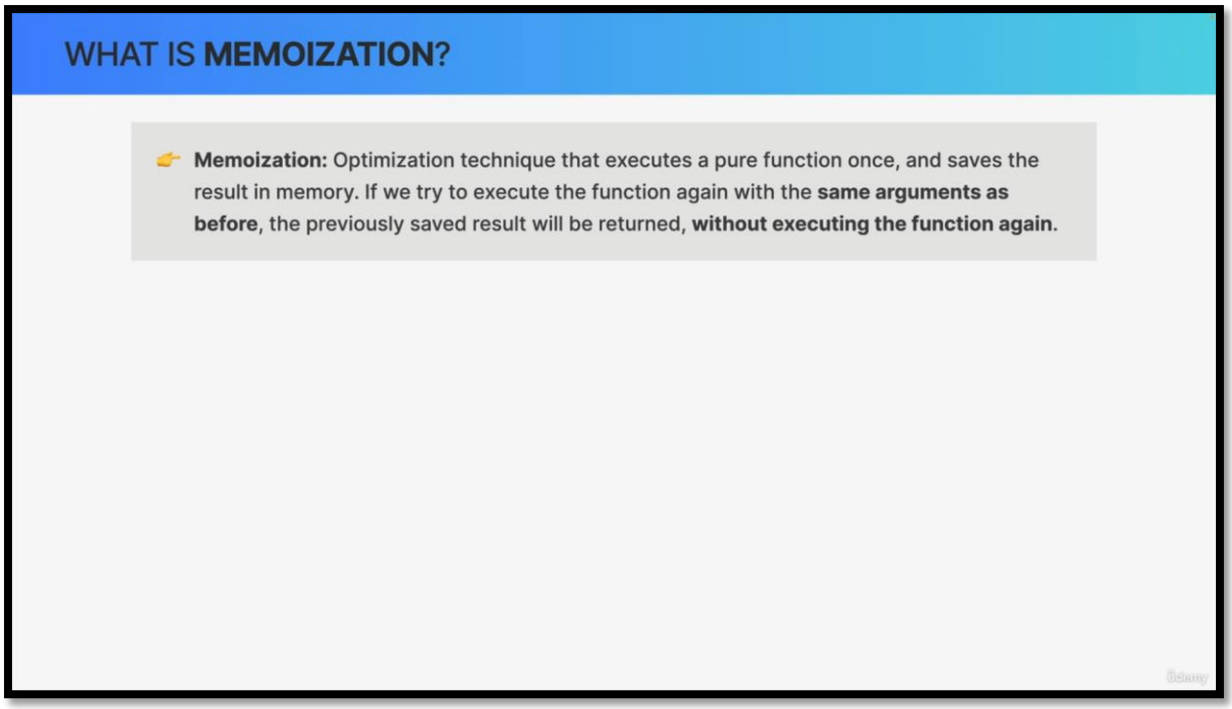
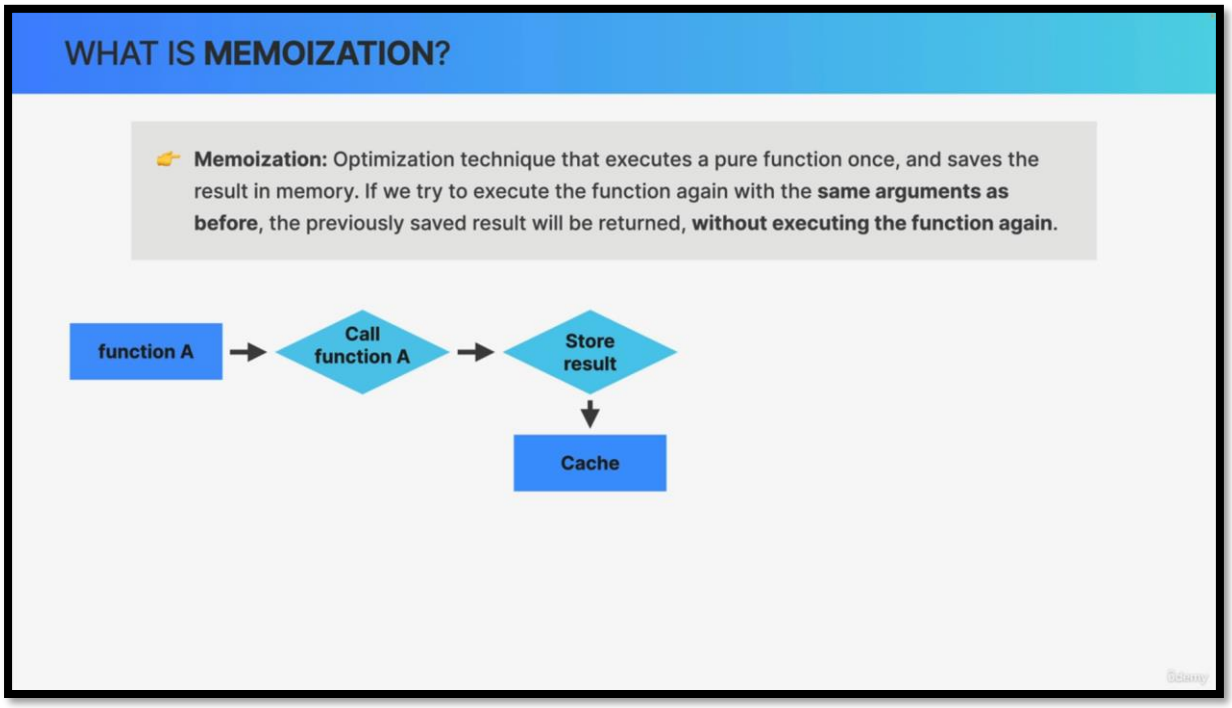
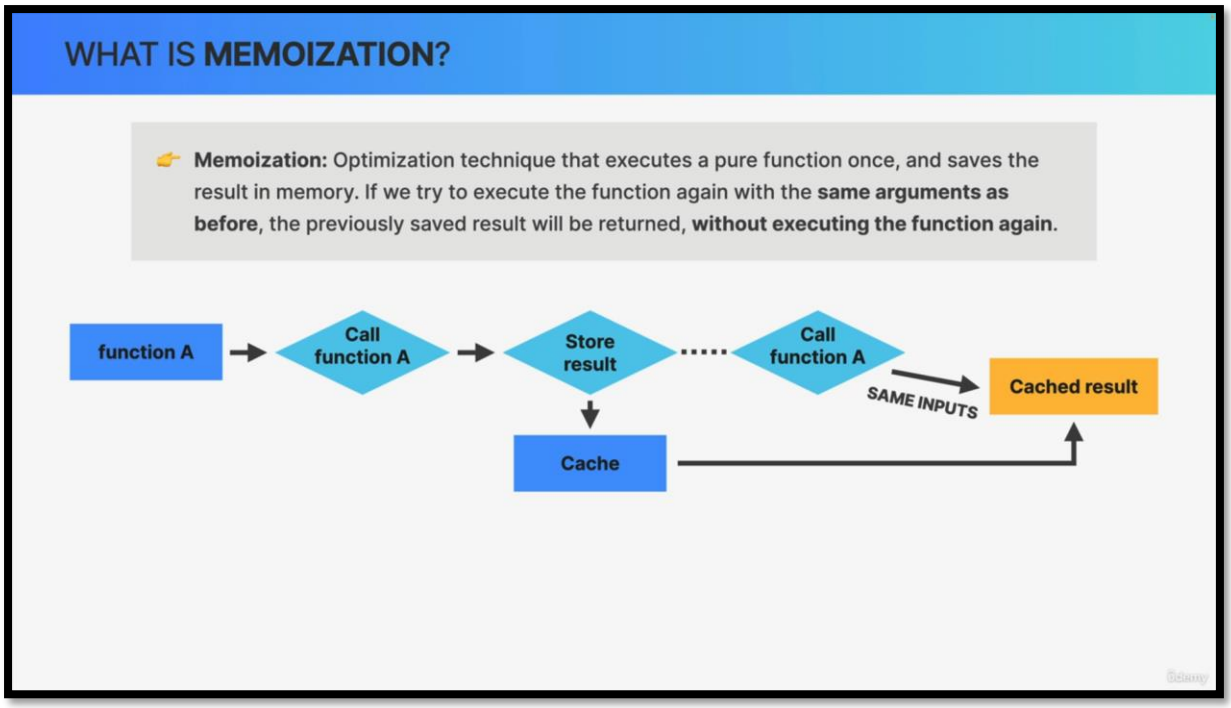


So memoization is an optimization technique that executes a pure function one time.



So, let's say function A and then it stores the results in memory that is in a cache.



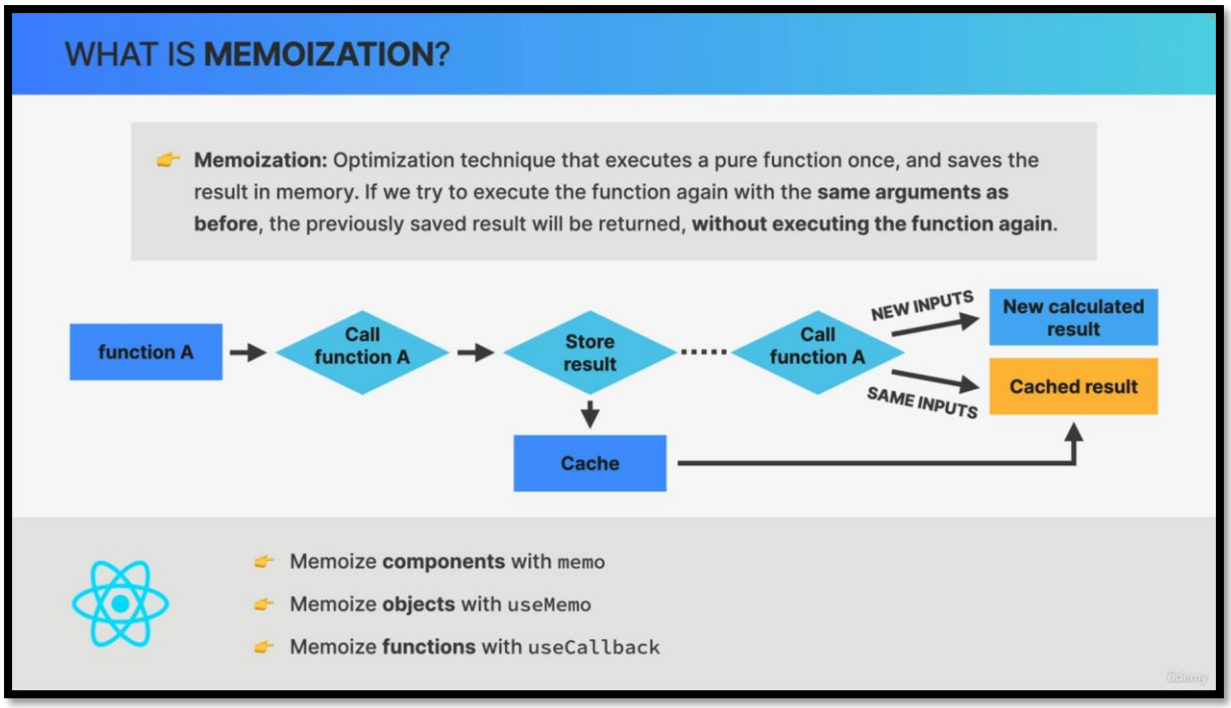
Then if we later try to execute the same function again with the same inputs it will simply return the results that was previously stored in the cache. So, the function will not be executed again.

So, if the arguments are exactly the same as before, it means that in a pure function, the output will be the same.

Therefore, it makes no sense to execute the entire code again if we can just read the cached results.

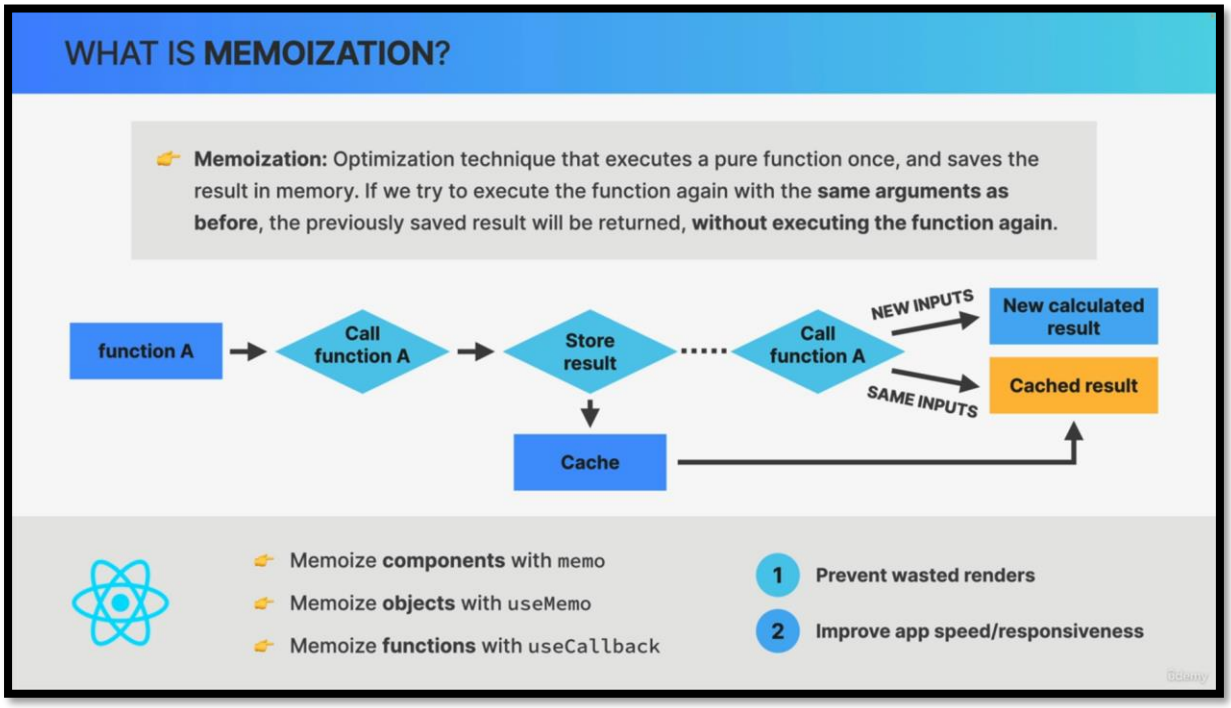
On the other hand, if the inputs are different, then of course the function will simply be executed again.

So that's the concept of memoization.

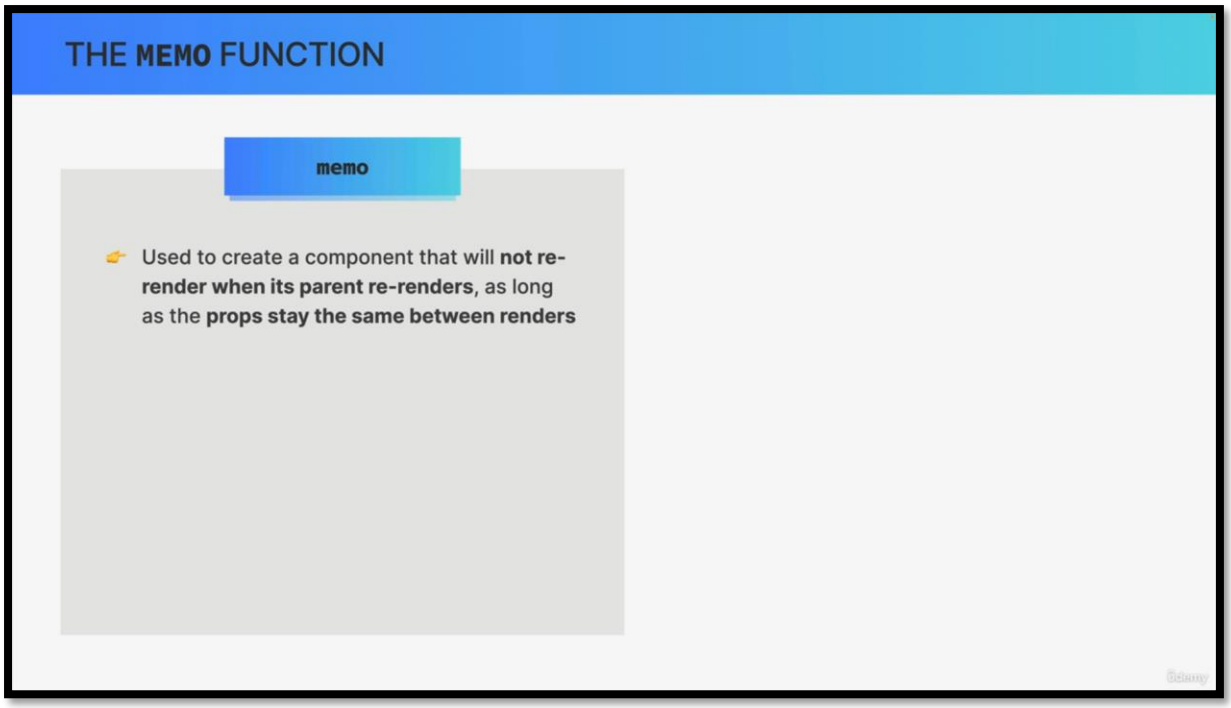


In React we can use this technique to optimize our applications.

We can use the memo function to memoize components with exactly this principle.



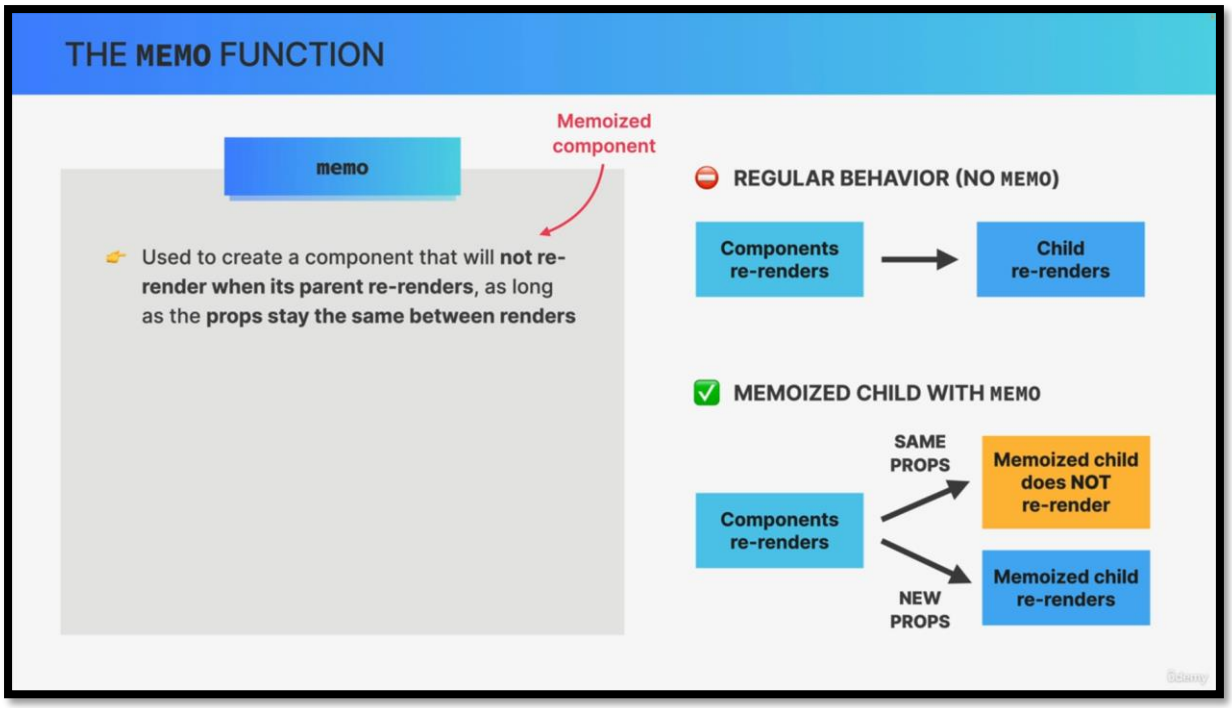
We can use the use memo and use callback hooks to memoize objects and functions and doing so will help us prevent wasted renders and improve the overall application speed.



React contains a memo function and we can use this function to create a component that will not re-render when its parent re-renders, as long as the past props stay the same between renders

In other words, we use memo to create a memoized component.

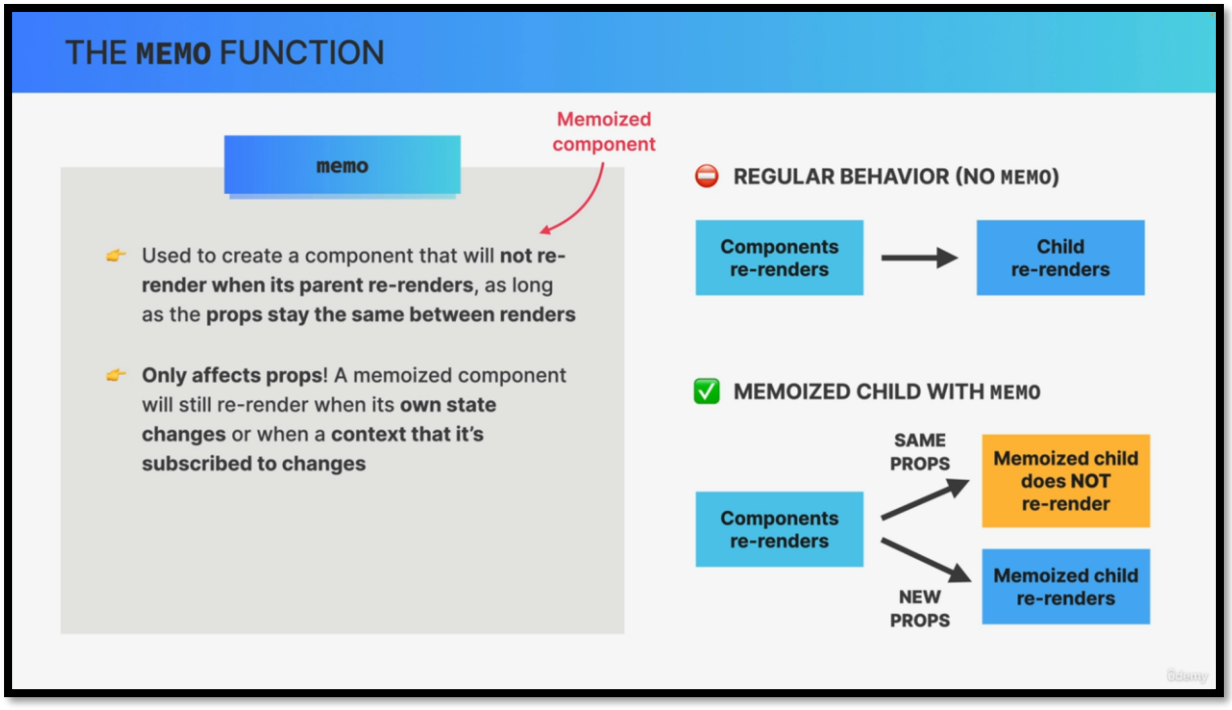
The function inputs are props and calling the function multiple times is equivalent to re-rendering in React and therefore, memoizing a React component means to not re-render it if props stay the same across renderers.



The regular behavior in React without using Memo is of course that whenever a component re-renders, the child component re-renders as well.

On the other hand, if we memoize the child component, it will not re-render as long as the props are the same as in the previous render.

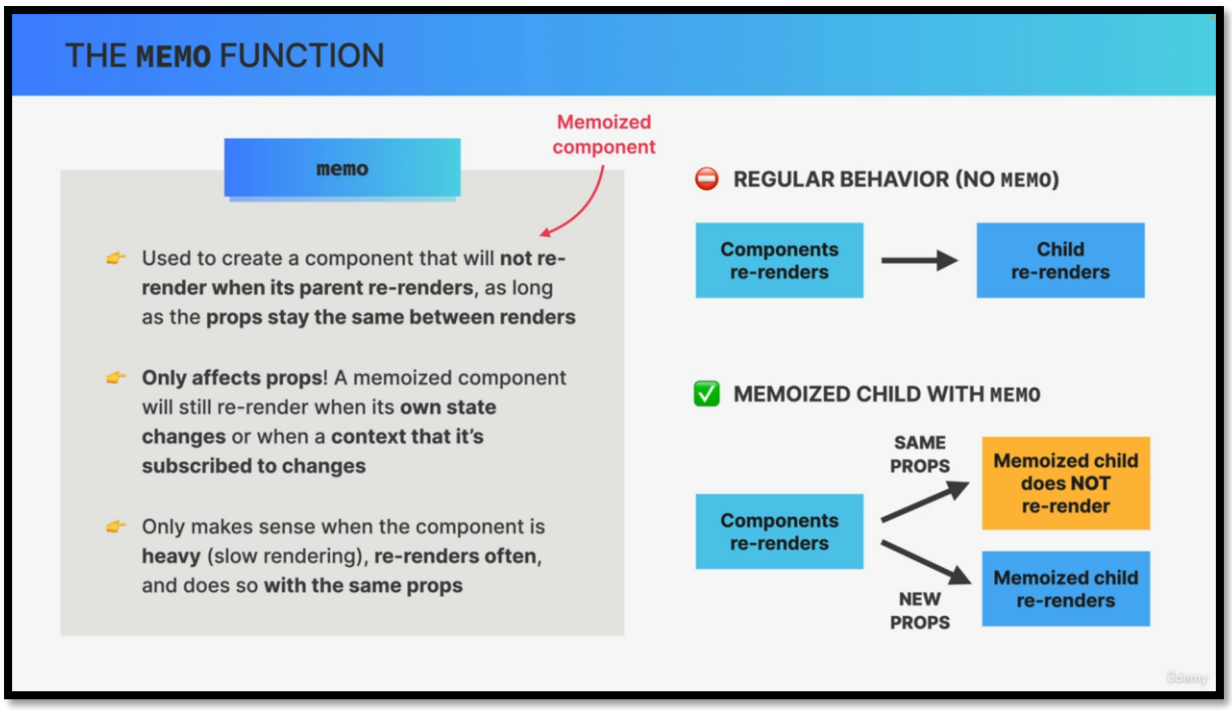
Now, of course, if the props do change, then the child component will need to re-render as well in order to reflect this new data that it received.



Now, it's super important to keep in mind that memoizing a component really only affects props.

This means that a memoized component will still re-render whenever its own state changes or whenever there is a change in a context that a component is subscribed to.

Because in these two situations, the component basically has new data that it needs to show in the UI, and so then it'll always re-render no matter if it's been memoized or not.



memo is only useful when we're dealing with a heavy component, so a component that creates a visible lag or a delay when it's rendered.

Also, in order for memo to actually make sense, the component should render quite frequently and it should do so with the same props.

So first, if the props are usually different between re-renders, memo has no effect anyway and then there is absolutely no need to use it.

Second, we learned in the first lecture of the section that wasted renders are only a problem when the re-rendering happens too frequently or when the component is slow in rendering.

Therefore, if the component only re-renders from time to time or if the component is lightweight and fast anyway, then memoizing brings no benefit at all.