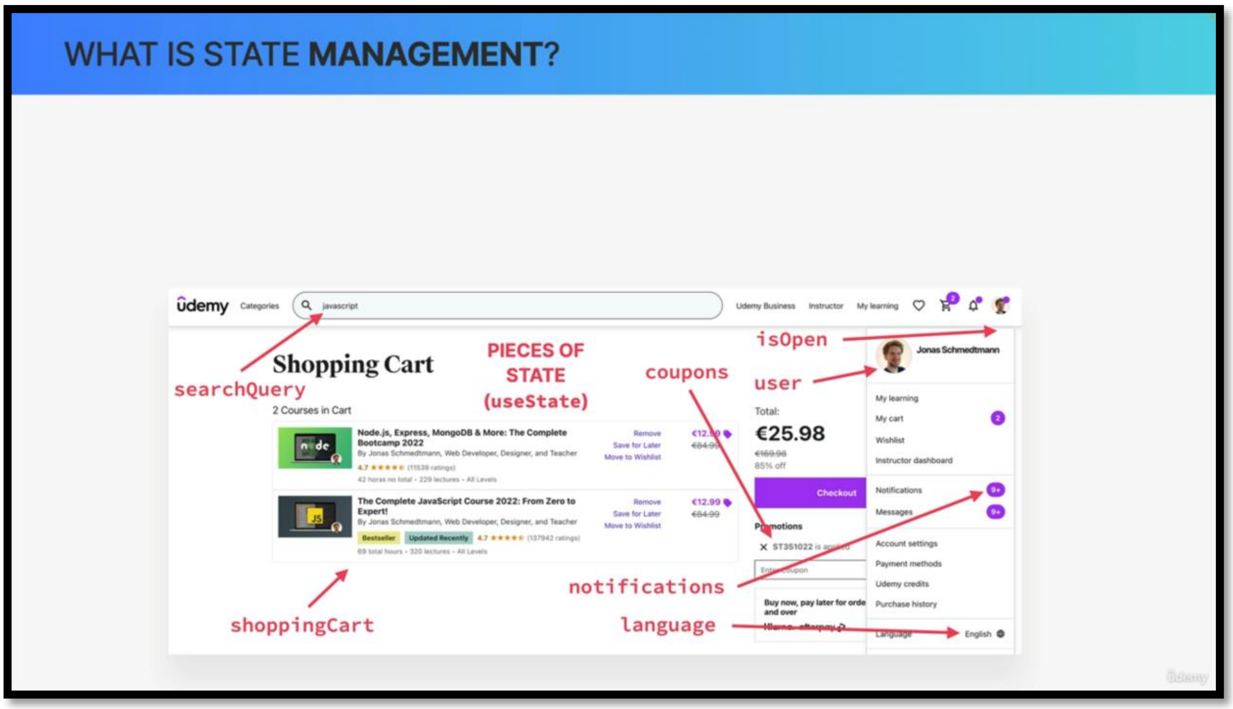


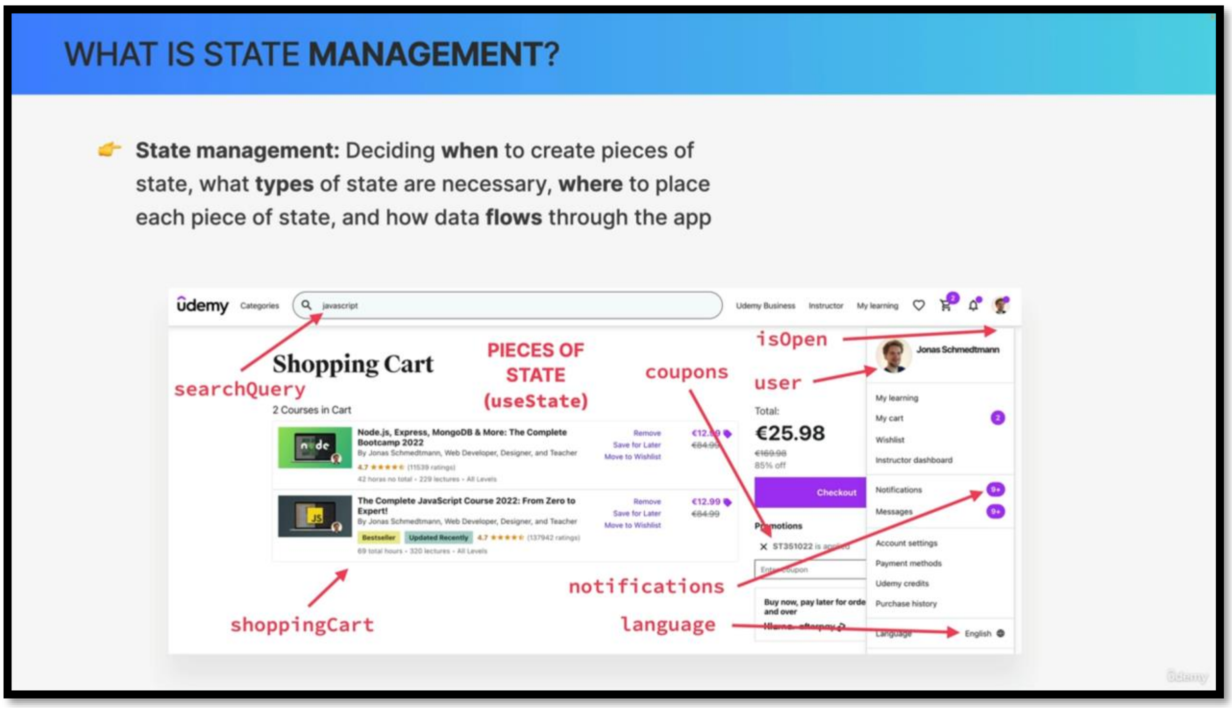
WHAT IS STATE MANAGEMENT?

State is the most important concept in React. Therefore, managing state is the most important aspect when it comes to thinking in React.

We can use the useState function to create multiple pieces of state in order to track data that changes over the life cycle of an application.

How do we know that we even need all of these pieces of state? How do we know where exactly to place them inside the code?

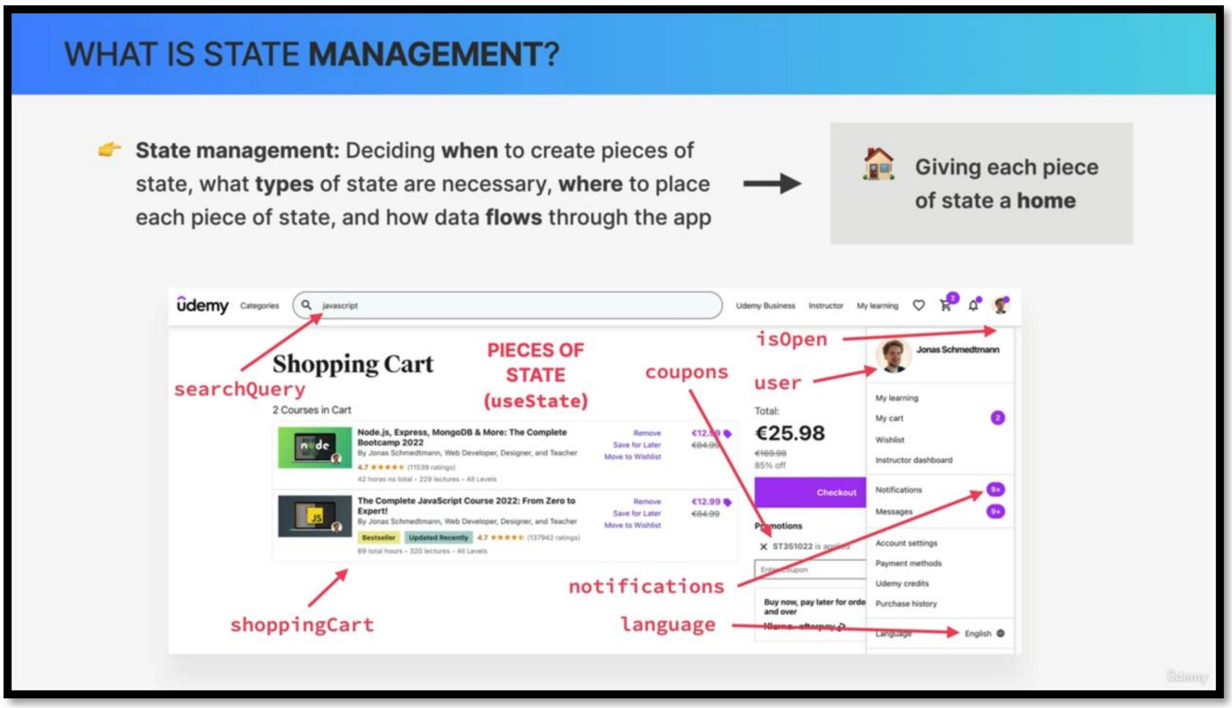
That's where state management comes into play.



WHAT IS STATE MANAGEMENT?

👉 **State management:** Deciding **when** to create pieces of state, what **types** of state are necessary, **where** to place each piece of state, and how data **flows** through the app

We can think of state management as deciding when we need to create new pieces of state, what types of state we need, where to place each piece of state inside our code base, and also how all the data should flow through the app.

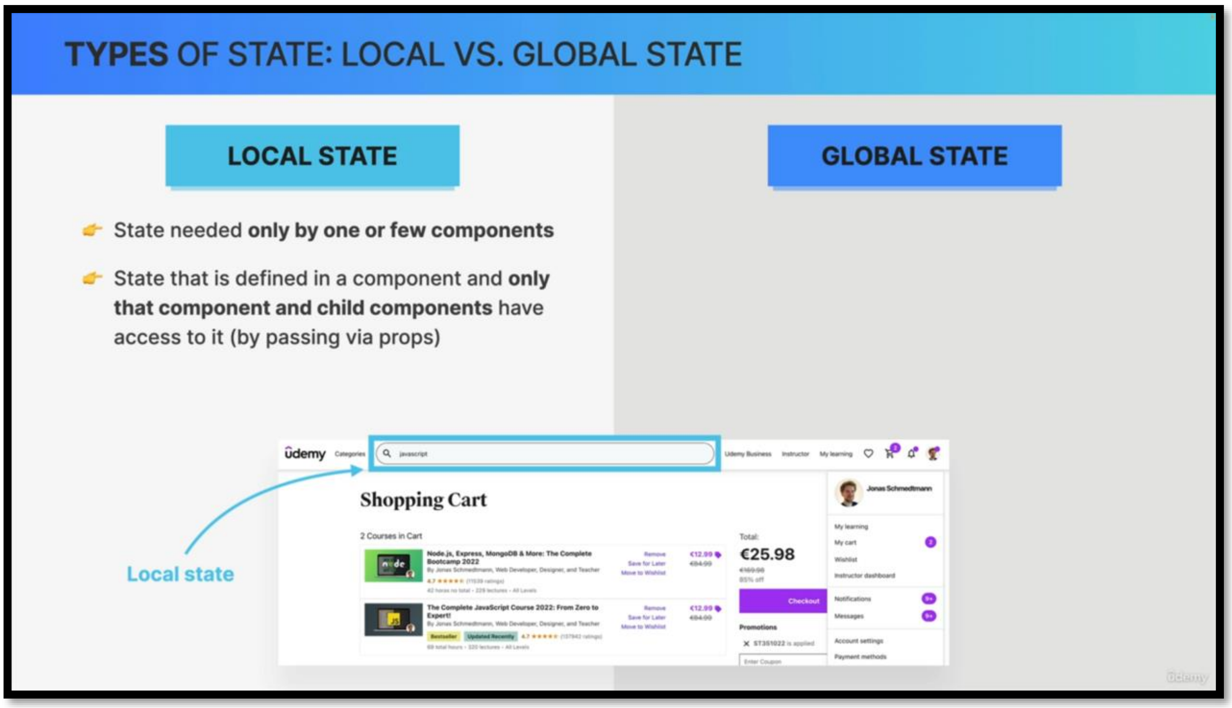


WHAT IS STATE MANAGEMENT?

👉 **State management:** Deciding **when** to create pieces of state, what **types** of state are necessary, **where** to place each piece of state, and how data **flows** through the app → 🏠 Giving each piece of state a **home**

State management is basically giving each piece of state a home within our code base.

As an application grows, the need to find the right home for each piece of state start to become really important, no matter if that home is the component where we first need that state, some parent component or even global state.

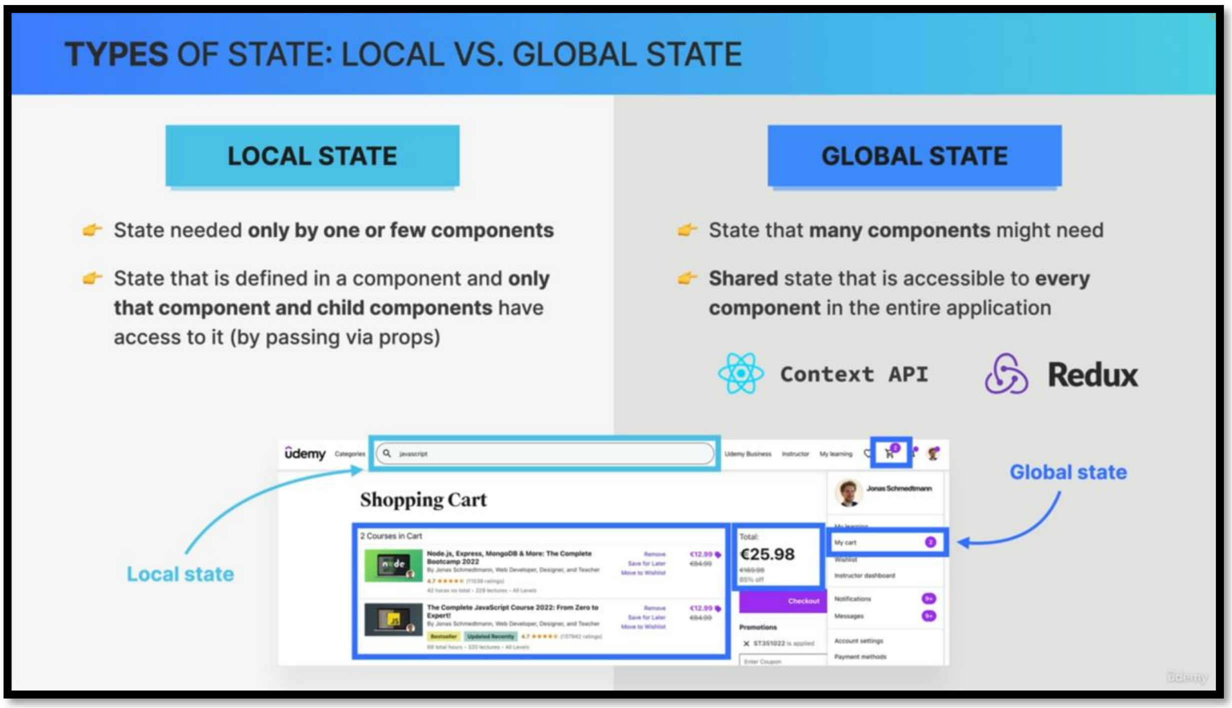


TYPES OF STATE: LOCAL VS. GLOBAL STATE

LOCAL STATE

GLOBAL STATE

- State needed **only by one or few components**
- State that is defined in a component and **only that component and child components** have access to it (by passing via props)



TYPES OF STATE: LOCAL VS. GLOBAL STATE

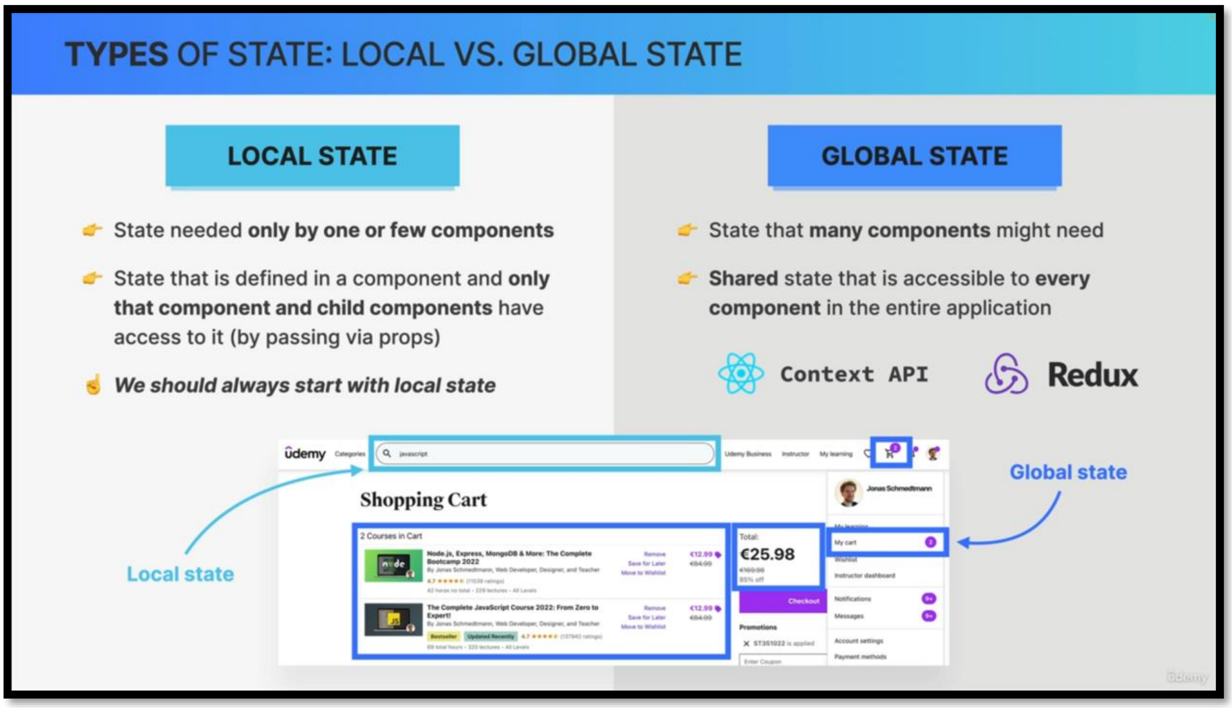
LOCAL STATE

- State needed **only by one or few components**
- State that is defined in a component and **only that component and child components** have access to it (by passing via props)

GLOBAL STATE

- State that **many components** might need
- **Shared** state that is accessible to **every component** in the entire application

Context API   Redux



TYPES OF STATE: LOCAL VS. GLOBAL STATE

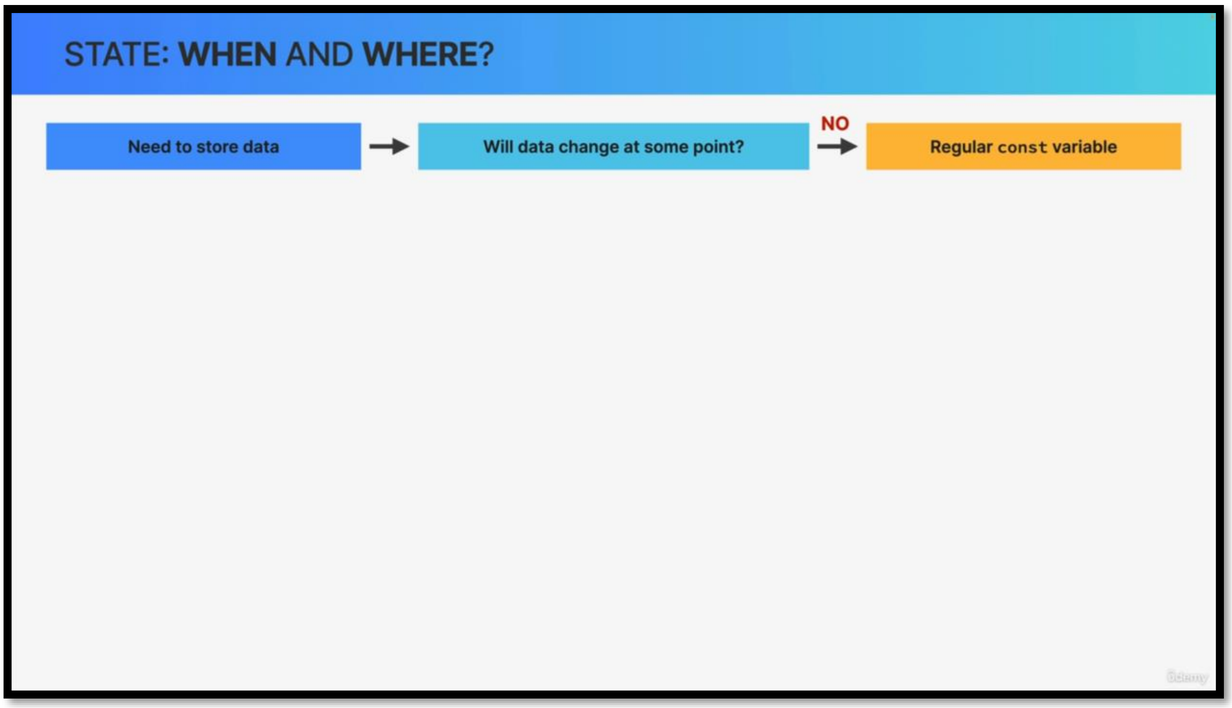
LOCAL STATE

- State needed **only by one or few components**
- State that is defined in a component and **only that component and child components** have access to it (by passing via props)
- ***We should always start with local state***

GLOBAL STATE

- State that **many components** might need
- **Shared** state that is accessible to **every component** in the entire application

Context API   Redux

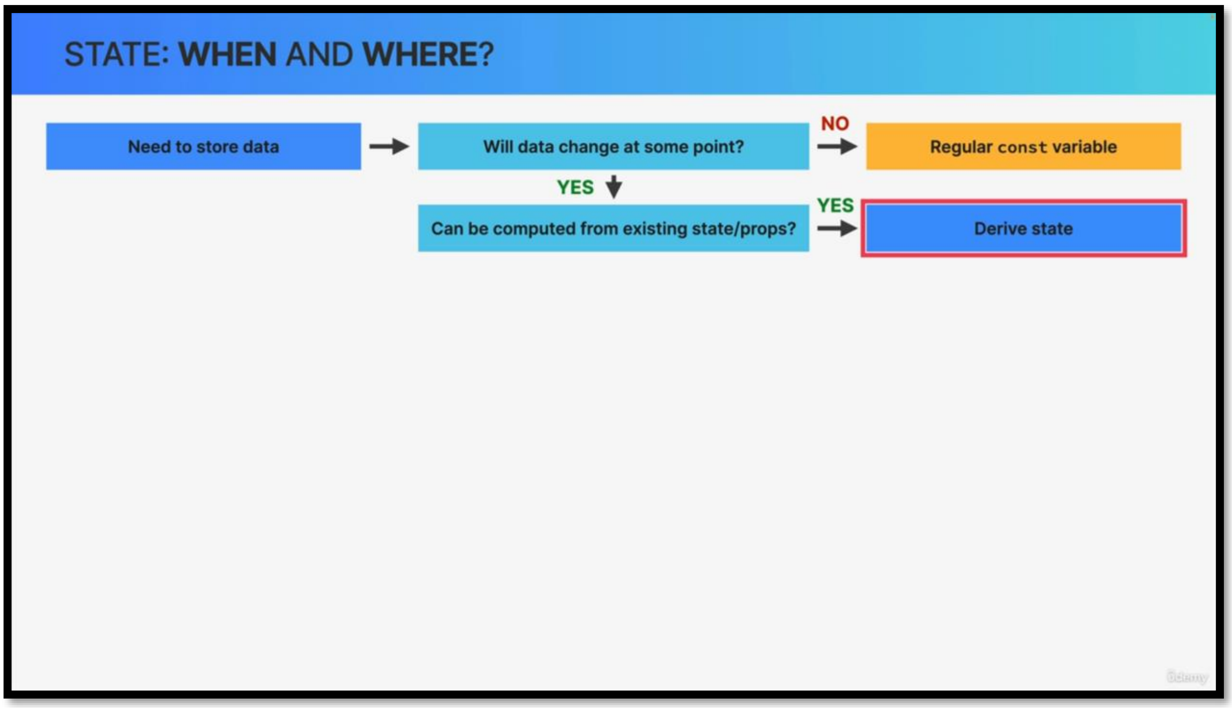


STATE: WHEN AND WHERE?

Need to store data → Will data change at some point? → NO → Regular `const` variable

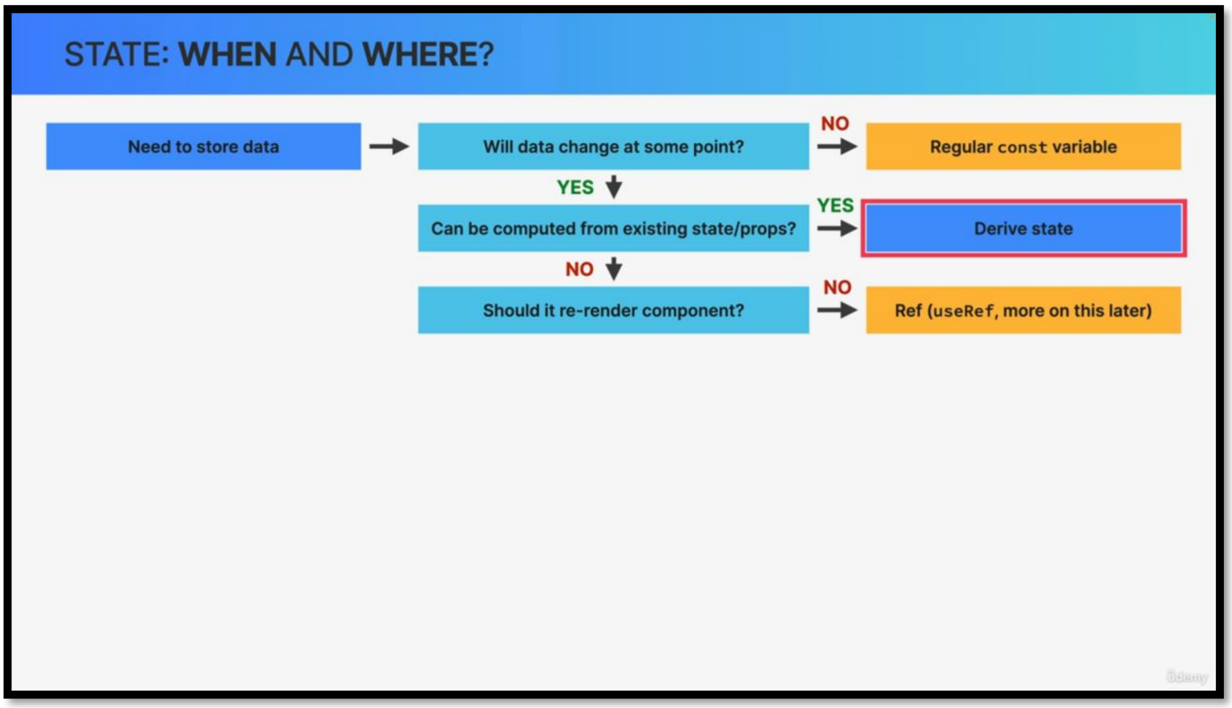
It all starts with you realizing that you need to store some data. Now when this happens, the first question to ask is, will the data change at some point in time?

If the answer is no then all you need is a regular variable i.e. a const variable.



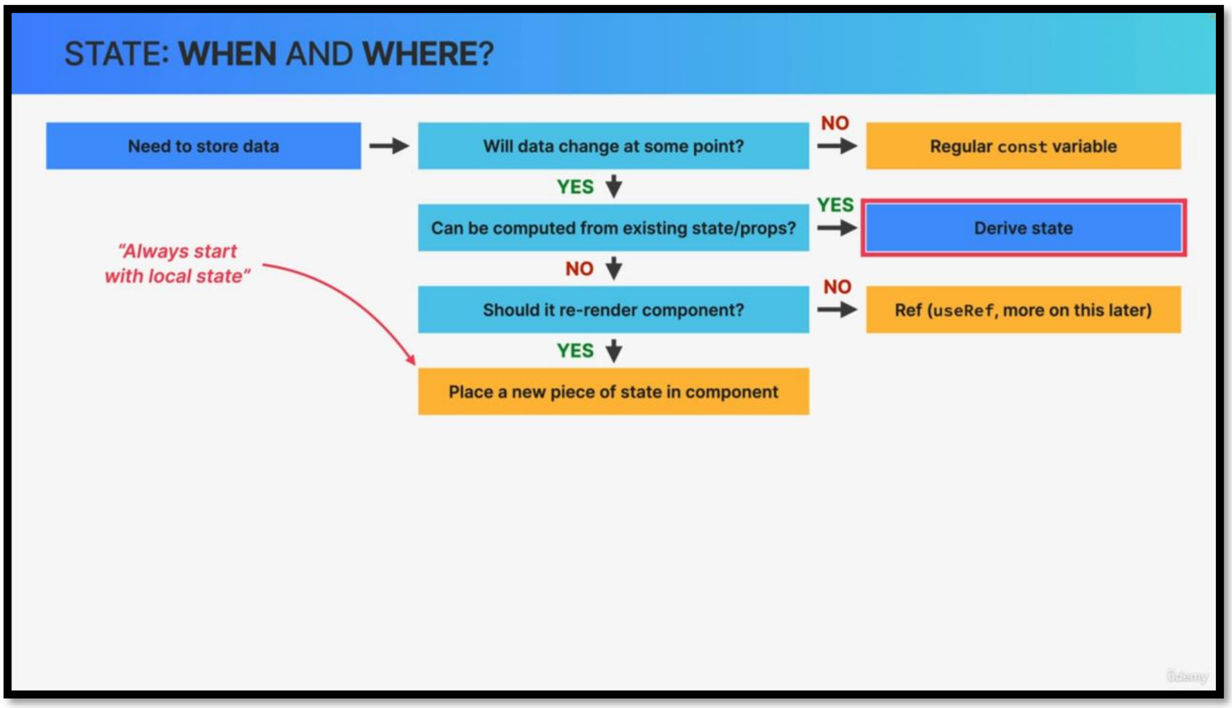
```
If the data does need to change in the future, the next question is, it is
possible to compute or to calculate this new data from an existing piece
of state or props?

If that's the case, then you should derive the state.
```


STATE: WHEN AND WHERE?
Need to store data
Will data change at some point?
NO
Regular const variable
YES
Can be computed from existing state/props?
YES
Derive state
NO
Should it re-render component?
NO
Ref (useRef, more on this later)

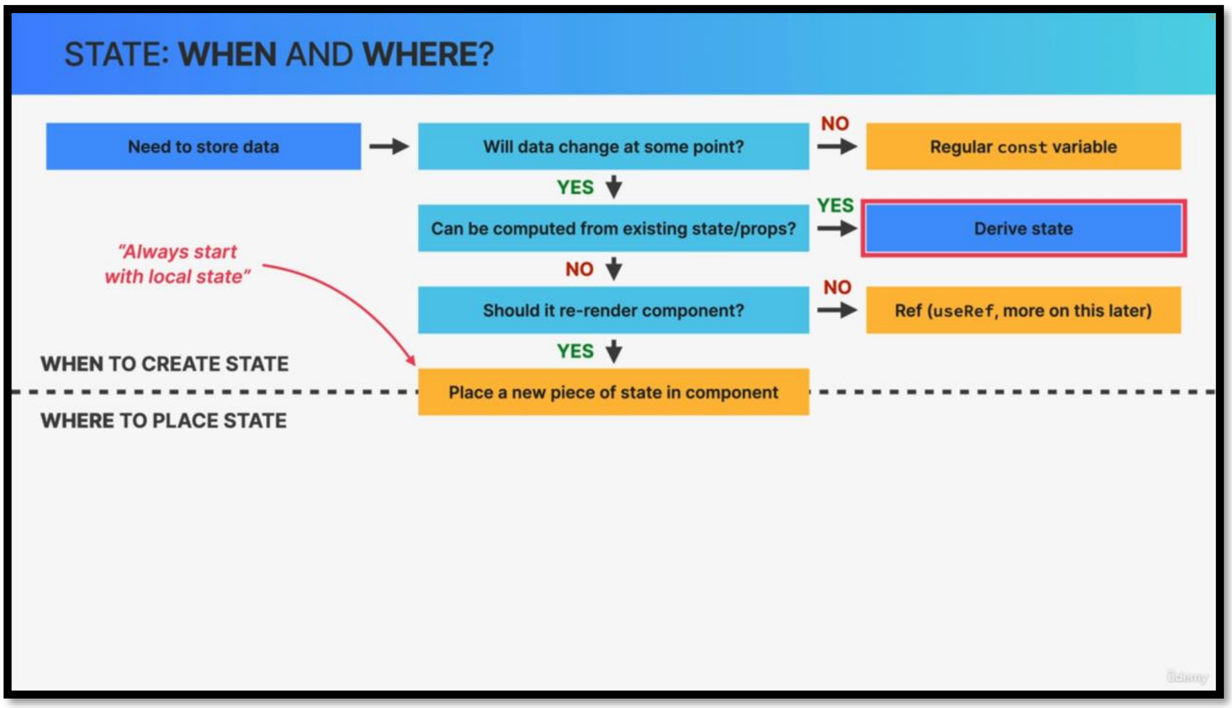
```
However, most of the time you cannot derive state. So in that case, you
need to ask yourself whether updating the state should re-render the
component.
```

Updating state always re-renders a component but there is actually something called a Ref which persists data over time like regular state but does not re-render a component.

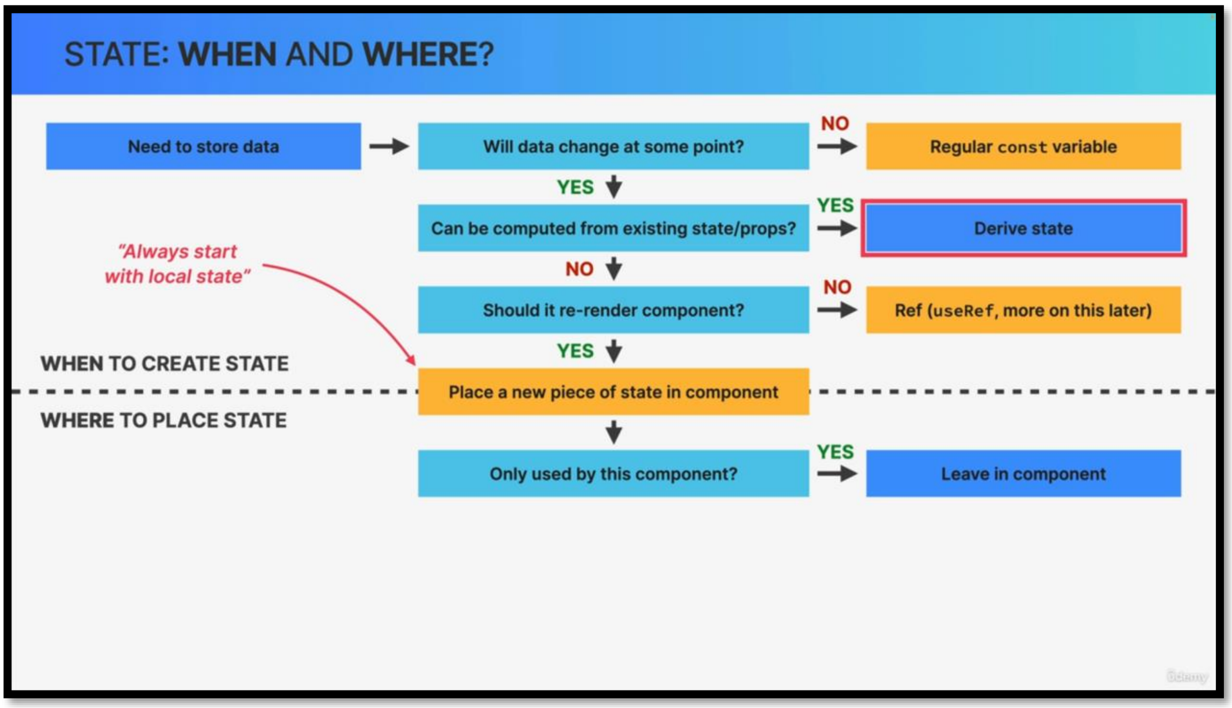

STATE: WHEN AND WHERE?
Need to store data
Will data change at some point?
NO
Regular const variable
YES
Can be computed from existing state/props?
YES
Derive state
NO
Should it re-render component?
NO
Ref (useRef, more on this later)
YES
Place a new piece of state in component
"Always start with local state"

However, most of the time you actually do want state to re-render the component.

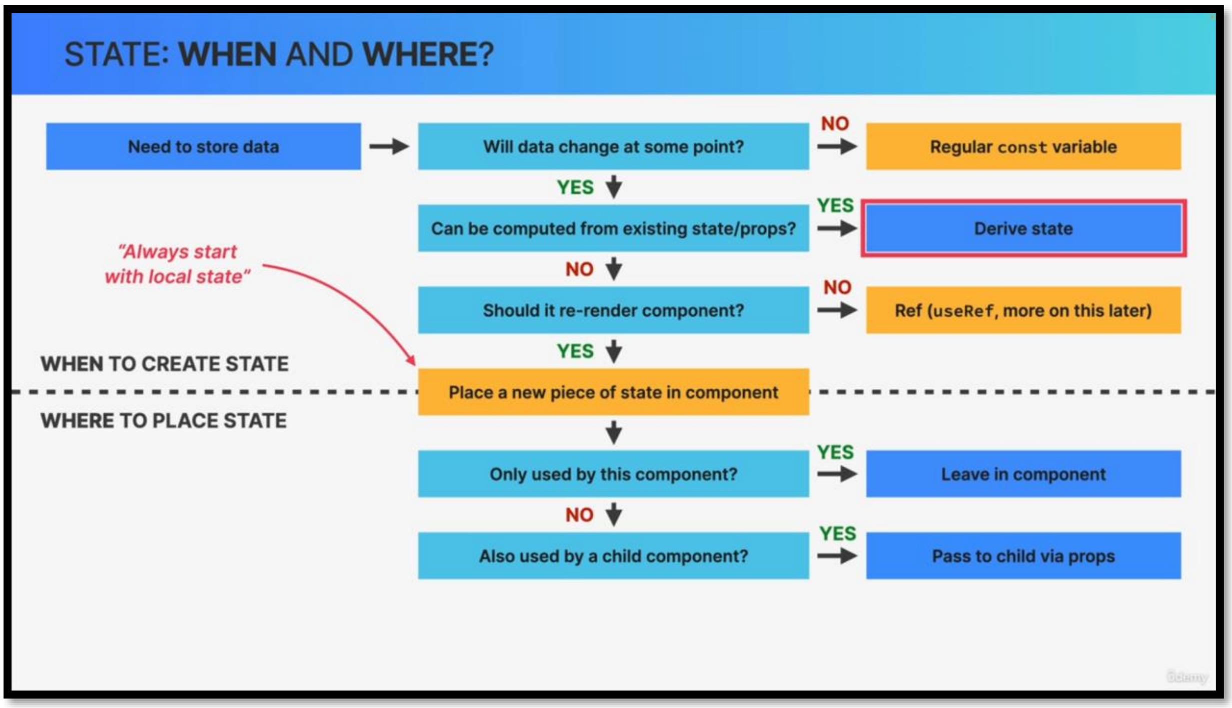
So, what you do is to create a new piece of state using the useState function and you then place that new piece of state into the component that you are currently building. So that's the always start with local state guideline.


STATE: WHEN AND WHERE?
Need to store data
Will data change at some point?
NO
Regular const variable
YES
Can be computed from existing state/props?
YES
Derive state
NO
Should it re-render component?
NO
Ref (useRef, more on this later)
YES
"Always start with local state"
WHEN TO CREATE STATE
Place a new piece of state in component
WHERE TO PLACE STATE

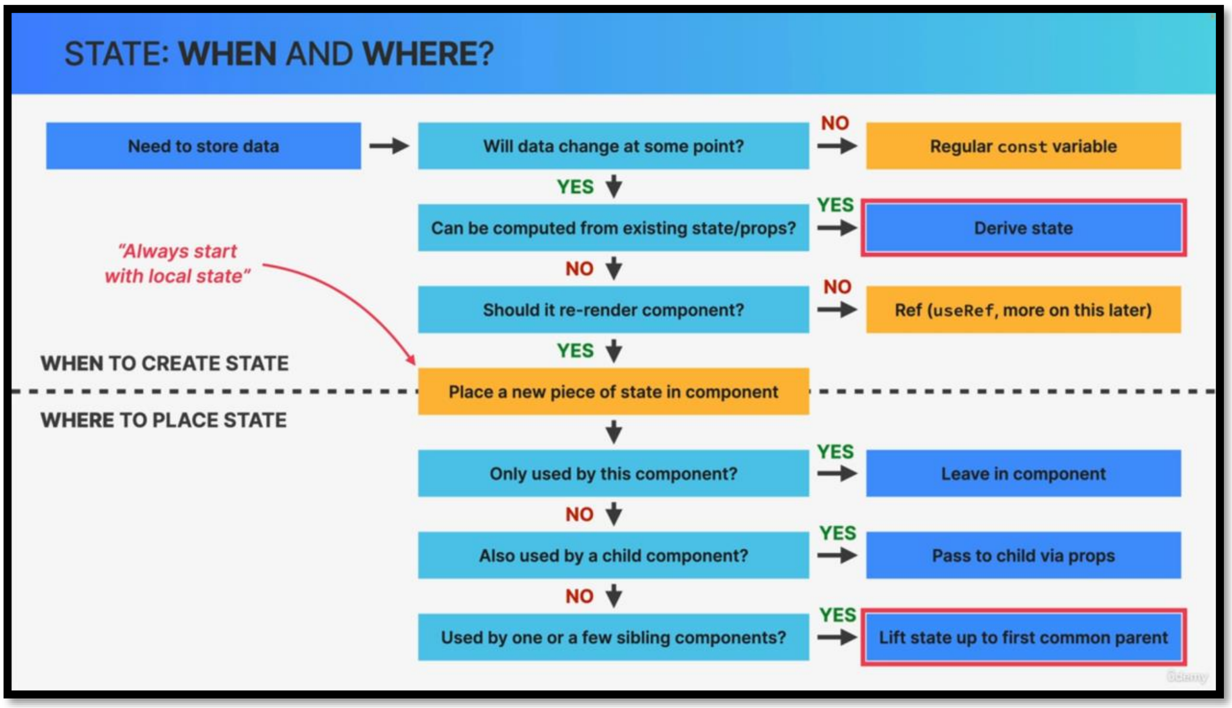
With this, we have completed the decision process of when to create state.


STATE: WHEN AND WHERE?
Need to store data
Will data change at some point?
NO
Regular const variable
YES
Can be computed from existing state/props?
YES
Derive state
NO
Should it re-render component?
NO
Ref (useRef, more on this later)
YES
"Always start with local state"
WHEN TO CREATE STATE
Place a new piece of state in component
WHERE TO PLACE STATE
Only used by this component?
YES
Leave in component

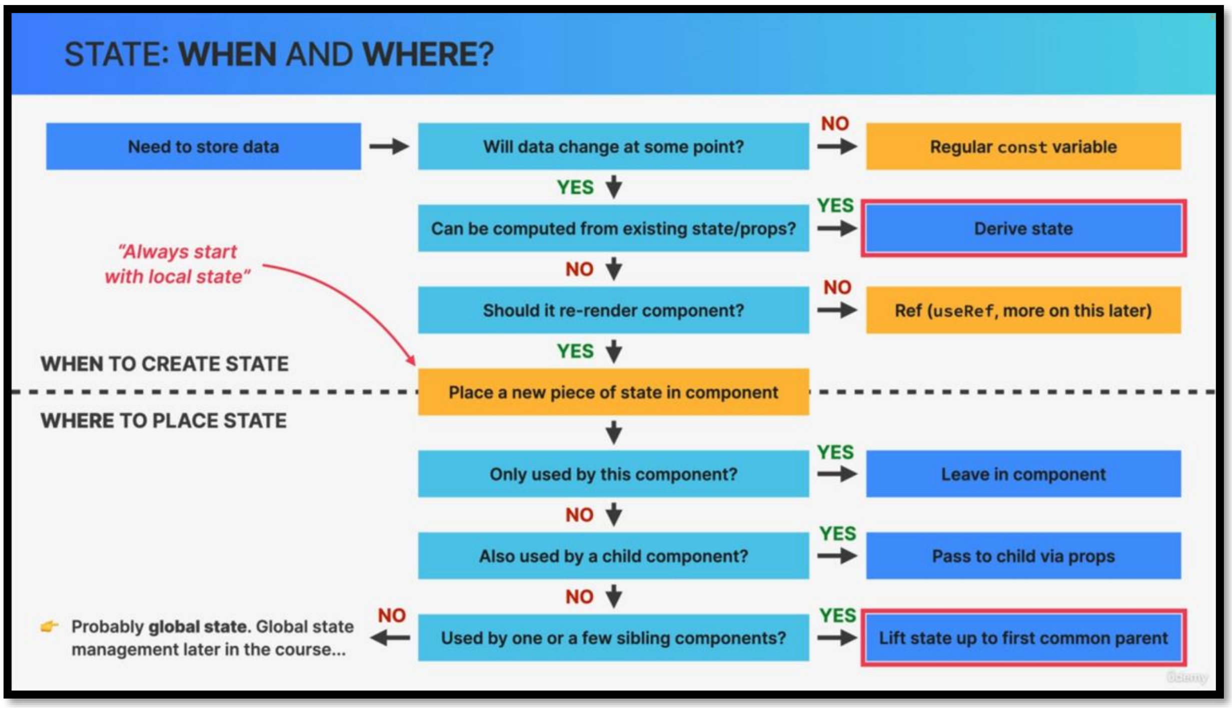
If the state variable that we just created is only used by the current component then simply leave it in that component and you're done.


STATE: WHEN AND WHERE?
Need to store data
Will data change at some point?
NO
Regular const variable
YES
Can be computed from existing state/props?
YES
Derive state
NO
Should it re-render component?
NO
Ref (useRef, more on this later)
YES
"Always start with local state"
WHEN TO CREATE STATE
Place a new piece of state in component
WHERE TO PLACE STATE
Only used by this component?
YES
Leave in component
NO
Also used by a child component?
YES
Pass to child via props

However, the state variable might also be necessary for a child component. In that case, simply pass the state down into the child component by using props.


STATE: WHEN AND WHERE?
Need to store data
Will data change at some point?
NO
Regular const variable
YES
Can be computed from existing state/props?
YES
Derive state
NO
Should it re-render component?
NO
Ref (useRef, more on this later)
YES
"Always start with local state"
WHEN TO CREATE STATE
Place a new piece of state in component
WHERE TO PLACE STATE
Only used by this component?
YES
Leave in component
NO
Also used by a child component?
YES
Pass to child via props
NO
Used by one or a few sibling components?
YES
Lift state up to first common parent

If the state variable is also necessary for one or a few sibling components or even for a parent component of your current component, it's time to move that state to the first common parent component. In React, this is what we call lifting state up.


STATE: WHEN AND WHERE?
Need to store data
Will data change at some point?
NO
Regular const variable
YES
Can be computed from existing state/props?
YES
Derive state
NO
Should it re-render component?
NO
Ref (useRef, more on this later)
YES
"Always start with local state"
WHEN TO CREATE STATE
Place a new piece of state in component
WHERE TO PLACE STATE
Only used by this component?
YES
Leave in component
NO
Also used by a child component?
YES
Pass to child via props
NO
Used by one or a few sibling components?
YES
Lift state up to first common parent
NO
Probably global state. Global state management later in the course...

Finally, the state variable might be needed in even more than just a few siblings. So, it might be necessary all over the place in the component tree.

In this case we use global state.