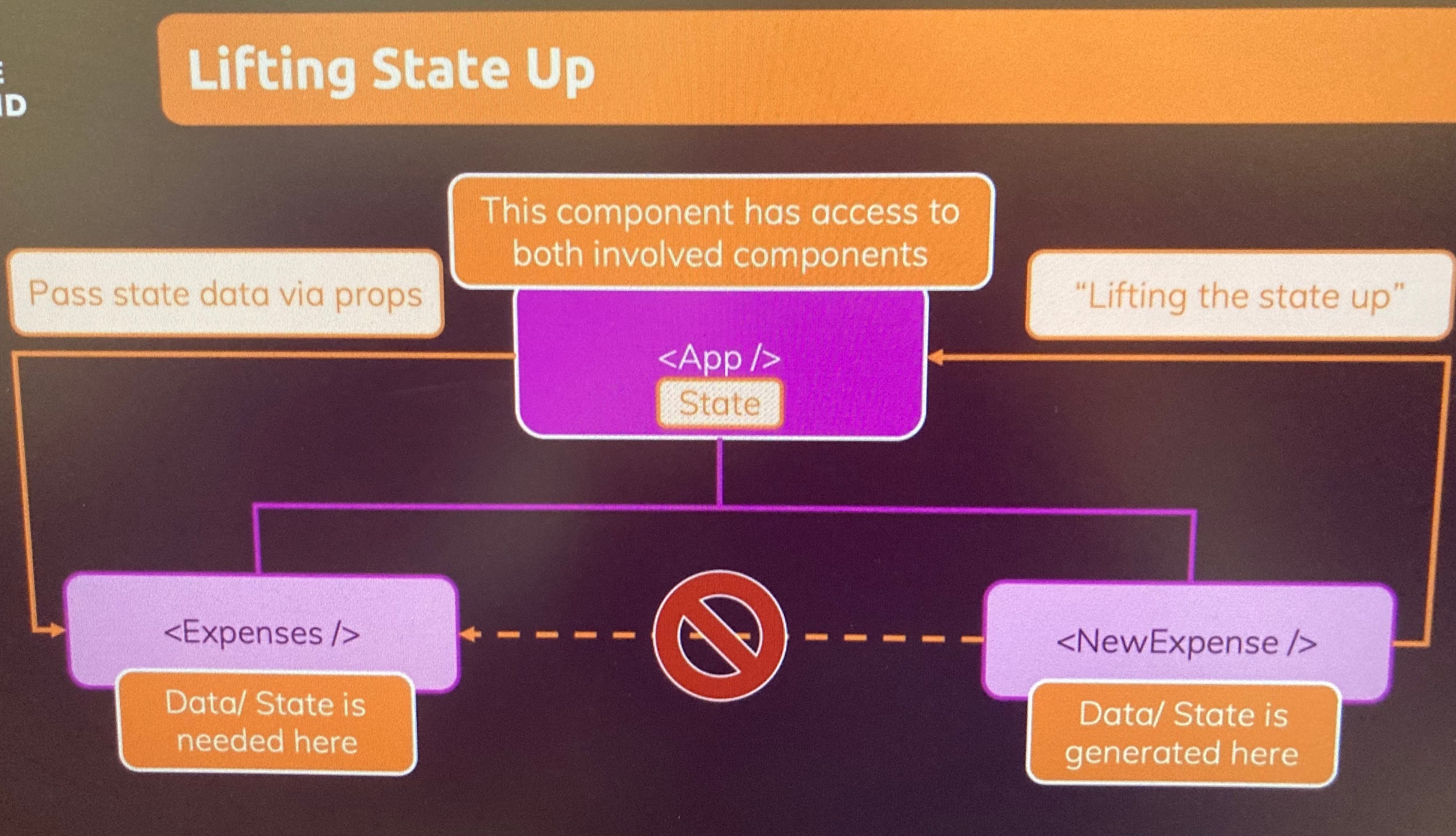
Section 4 react state and working with events

* **Overview of section/what to expect.**
* More core features of react
* Section dedicated more to handling user interaction and handling user events clicks,inputs etc..
* “State” in react apps
  + Declarative approach– we define desired target “states” and react takes care of the JS DOM instructions
    - So far- the “state” never changes in what we’ve made… it’s static and the info is always exactly the same
    - Big part of react is making apps reactive that react to clicks/user inputs/ and changes based on these occurrences
    - Reacting to events/input changes the “state” making sure that whats visible on the screen changes and we reach a diff. Target “state”
* **Listening to events and working with event handlers**
  + User events
  + Clicks
    - Native dom events are accessible for buttons and there is a prop version in react
    - onClick event gets set with a value which is code that has to run when the click occurs
      * Typically in JS if you want an event to occur if something is clicked you create a function
    - All these event handler props want a function passed as a value.. Very simple function to enact a response.
      * These typically written before return
    - Any element that supports an event you can add a listener in react in the same way
    - Conventional to name an event with the type and handler ex. clickHandler
    - const ExpenseItem = (props) => {
    - Const title = props.title;
    - const clickHandler () => {
    - title= ‘updated!’

console.log(‘updated’)

* + - };
    - return (
    - <Card className="expense-item">
    - <ExpenseDate date={props.date} />
    - <div className="expense-item\_\_description">
    - <h2>{title}</h2>
    - <div className="expense-item\_\_price">${props.amount}</div>
    - </div >
    - <button onClick={clickHandler}>Change Title</button>
    - </Card >
* Changing what shows up on the screen
  + This is simple with one catch simply writing a function that changes something on the page when its clicked doesn’t automatically work by default…it will execute but won’t necessarily bring about the visual change you specify by default.. ^ this will execute when clicked, but won’t change the title as-is.. Issue is this component is a function but we never run it.. It runs on its own so we have to communicate with react… the function is essentially getting called on another page..
  + <ExpenseItem
  + title={props.expenses[0].title}
  + amount={props.expenses[0].amount}
  + date={props.expenses[0].date}
  + />
  + React calls any function it encounters then the functions those functions called etc.. until no more…
  + It goes down the line..starting at index where it points at the App component Executes components down the line until it reaches no more, then it evaluates into DOM instructions which renders the results on the screen…
    - The reason there was no change above ^ is because react only goes through once before rendering- but it would have to revisit the component ^ to be aware of the necessary change.
    - This is where state comes into the mix–
* **State**
  + Not react specific but important
  + The variable that is supposed to change gets ignored by default. ^ it gets evaluated, but the function is not run again to enact the desired change. React doesn’t care about the variable changing unless tell it to
  + Way around is to use a named import with a function provided by the React library called useState
    - Allows us to define values as State where changes to these values will reflect in the function being called again
      * Then you call useState();
        + This is a react hook
        + Hooks start with use and have to be called directly within component functions (1 exception that is covered later)
        + useState returns 2 values- first value 2nd is updated
  + const [title, setTitle] = useState(props.title)
  + Use state creates a special kind of variable where changes will lead to this component being called again and it returns access to this special variable
  + First value is the variable itself, 2nd is an updating function.. Can use array destructuring to store both values in constants
  + Calling this function doesn’t automatically assign a new variable, but the initial variable is managed by react and when the state updating function si called the component function will run again…. Then it runs with the changed state and will projet the changes onto the screen
  + Keep in mind that when yo urun this the value doesn’t change immediately, but the update is scheduled… so if you console.log the initial value will still get logged
  + State is necessary when you have anything you want updated. .changing variables won’t do this in react so you need state
  + Notes on state
    - When you have a function with an updated state function if that function is called 4times you have 4 separate states created, all int eh same way but managed independently by react, so if you interact with one of the objects you are interacting with that independent state despit the fact that they all spawn from the same component/function
  + Note on const usage with useState()
    - This is a value that gets updated, but you can use const… it isn’t updated with = but with a function… concrete value is managed somewhere else by react so it doesn’t create a problem
    - Capturing that updated state if it happens behind the sidelines
      * useState creates two values one is a brand new snapshot of that state when that component function reexicutes
      * React keeps track of state changes and when we call it
    - React determines and keeps track of state change and if an event occurs a second time it looks at the fact that a state change already occurred then goes with the recent updated state change.. State change only occurred the first time
  + **State in a nutshell**
    - Register state with useState
    - Get back the value itself and updating function
    - Call updating function when state should change
    - Use first element when you want state value to output in jsx code
      * State updates are vcommon for user events, but can be used for anything. Ex. http request or setTimeout()
    - You can and often will have multiple states per component
  + **Multiple state slices vs alternative**
    - If you have a form with 3 inputs- there are 3 values to catch but you could consider capturing the value of the form as single state slice
    - Individual is more popular but both are common
      * With multiple- updating states in objects can be problematic as you are resorting to the previous value which can be problematic
    - const [userInput, setUserInput] = useState({
    - enteredTitle: '',
    - enteredAmount: '',
    - enteredDate: ''
    - })
    - const amountChangeHandler = (event) => {
    - // setEnteredAmount(event.taget.value)
    - setUserInput({
    - ...userInput,
    - enteredAmount: event.target.value,
    - })
    - }
    - This will work but can cause problems and lead to issues….
    - Note
      * Whenever you update state depending on previous state… you use alternative form of state updating function… by passing function to it that is auto run by react that accepts prevState as an argument
    - setUserInput((prevState) => {
    - return { ...prevState, enteredTitle: event.target.value }
    - });
    - };
    - Insures you always operate on the latest state snapshot ^
* **2 way binding**
  + Setting up the inputs to take on new states allows flexibility and using 2 way binding
    - For inputs we don’t just listen but can pass a new value back in
      * Reset or change the input programatically
        + Add the default “value” attribute to the input element
        + Sets the internal value that every input has to a new value
        + Now we not only listen to changes in the input and update the state, but also feed the state back into the input so that when we change it we also change the input
        + Allows us to take the functions returned from state() and reset to an empty string in the case of this form when it is submitted
    - Allows you to gather info and change it… for ex. During form submission
* **Sending info from child to parent**
  + Before we’ve only used props to send info from the top down.
  + Now we want to take info from form and send it back up to app.js
  + The form inputs are essentially their own little components, but different from the custom components we create
  + Using props to receive function from parent then call in child
  + We can create our own event props that expect functions as values
    - Allows us to pass function from parent to child then call inside child
      * Then we can pass data to that function as parameter
* **Lifting state up**
  + Concept similar to creating a function to get something from child to parent element
  + Can’t pass anything between sibling elements, but can find shared parent element
    - Ex. you have info in one component that you want in another component that is its sibling… you can’t do that
      * But can grab from the closest parent that has direct/inderect access to both involved components… it renders both diff
      * components in its returned JSX code
      * Can store our state in that closest involved component by lifting our state up
      * Good to use as we often need that info in the parent, but also makes it accessible to sibling elements either and allows easy flow of info
* ****
* Practice exercise
  + Add filter component
    - Component has graph that filters spending by year.
    - There is a drop down that allows you to pick a year
  + Markup and styling is included
  + Test is to listen to changes on the dropown
    - listen to changes on the dropown when the value changes
    - Forward picked value to expenses component
    - Run ExpenseFilter in expenses component
    - Forward picked year from expense filter, cue the expenses component, then in the expenses component store in a state
  + Summed up- listen to change event, forward data to higher level(expense) store in a state
* **Controlled component**
  + Whenever you use 2 way binding you are creating a controlled component
  + Also can create controlled custom component
    - Meaning
    - A value is passed to a parent by props and received by the parent componenet through props as well
    - When… the logic is in the parent component
      * The value and the changes to the value are not handled in the component itself, but the parent component
  + In basically all react apps there are presentational, dumb components, and stateful and smart components.
    - More likely to have more presentational components than stateful
    - Typical to manage state in a few componentes, then pass it around through props