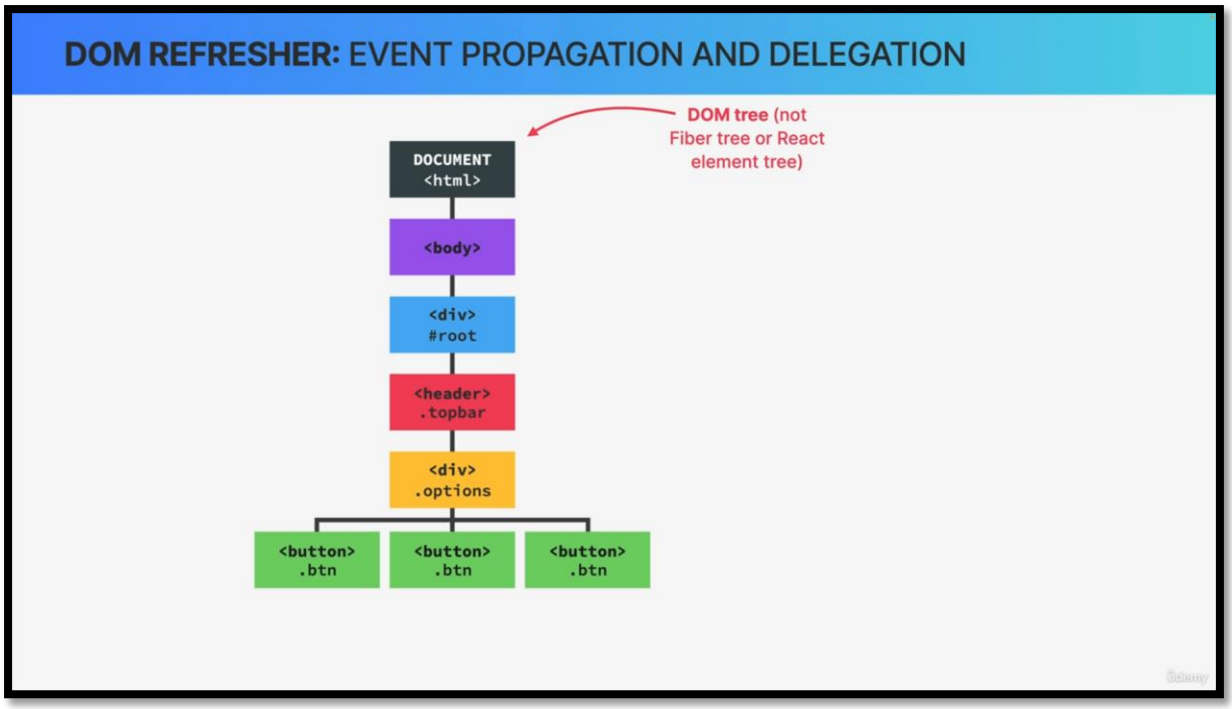
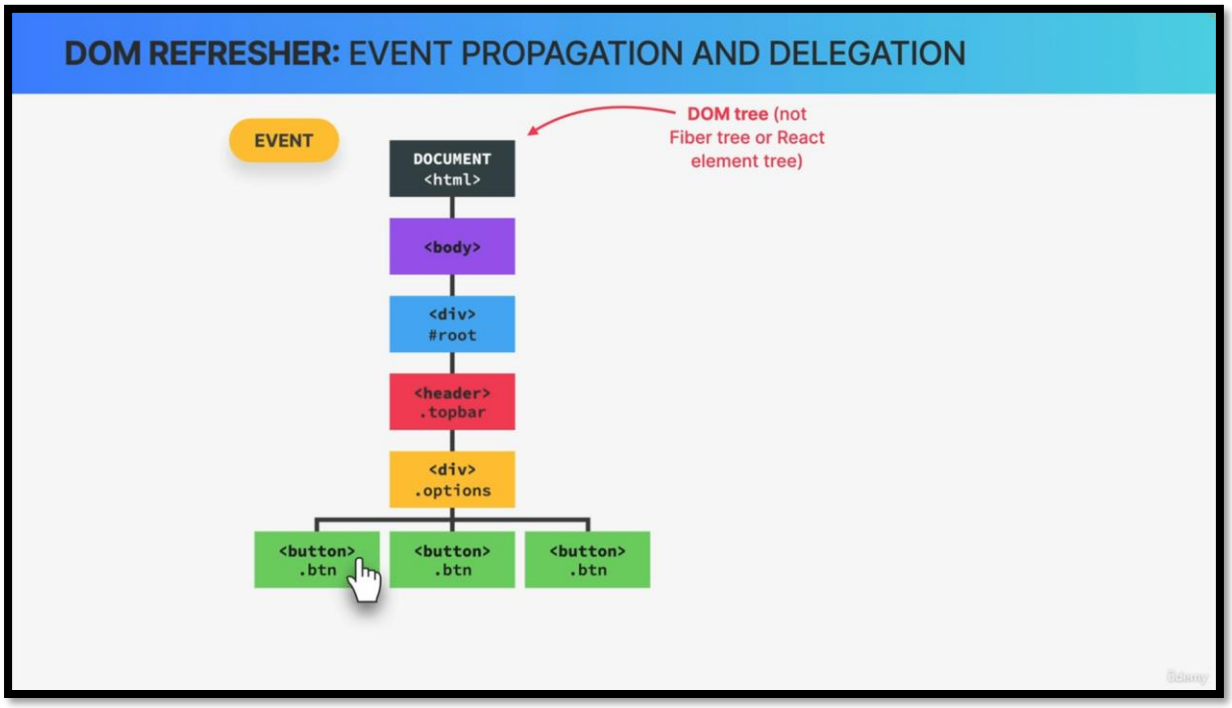
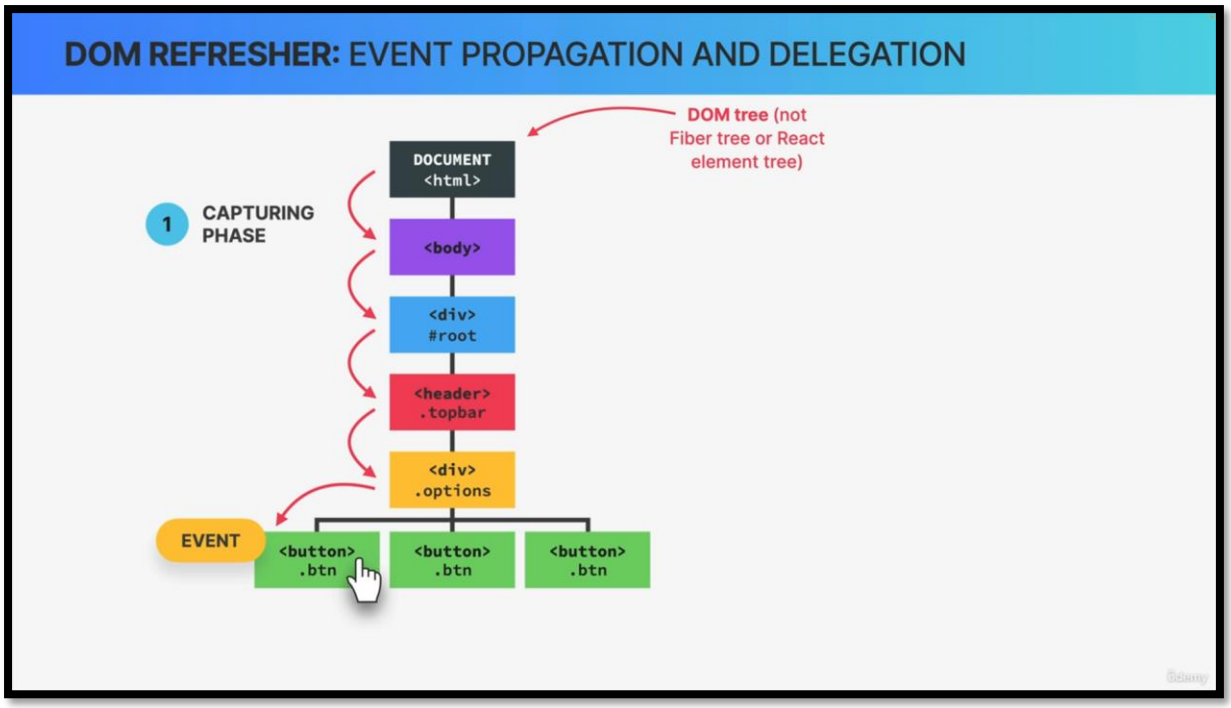


So, let's consider this tree of DOM elements, and note that this really is a DOM tree, not a Fiber tree or a React element tree.

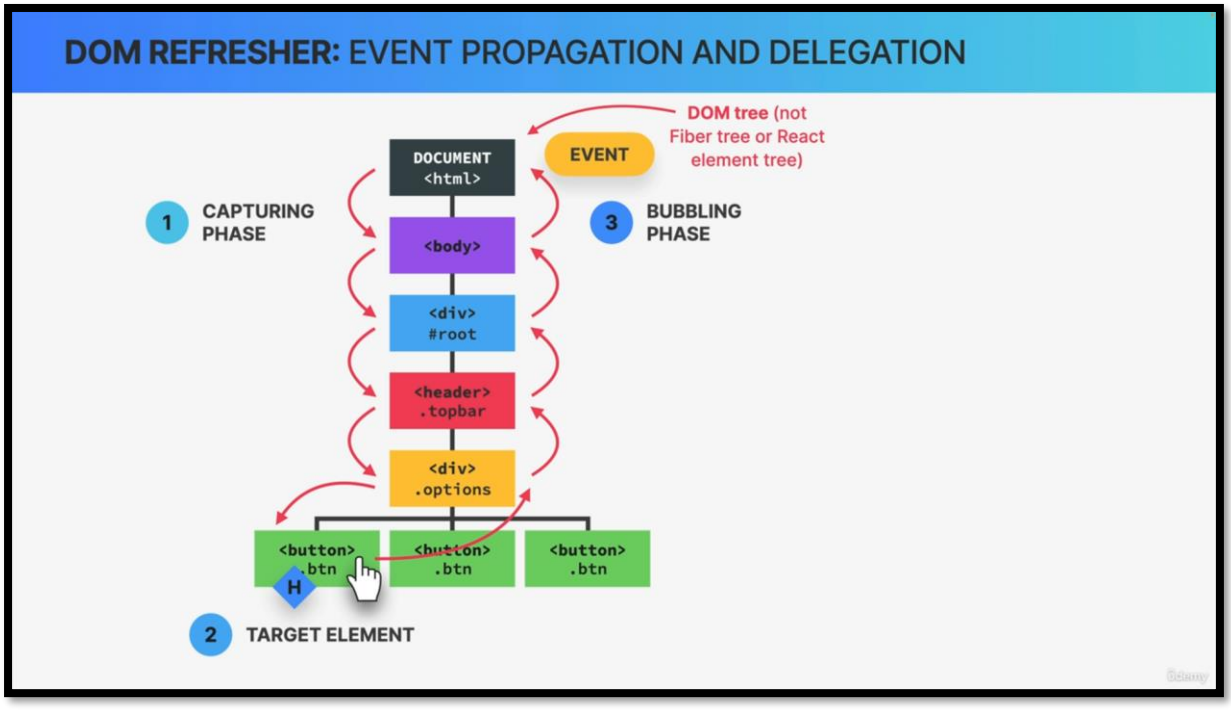


Now, let's say that some event happens, like a click on one of the three buttons and so here is what's gonna happen in the browser. As soon as the event fires, a new event object will be created, but it will not be created where the click actually happened.



Instead, the object will be created at the root of the document, So at the very top of the tree. From there, the event will then travel down the entire tree during the so-called capturing phase, all the way, until it reaches the target element and the target element is simply the element on which the event was actually first triggered.

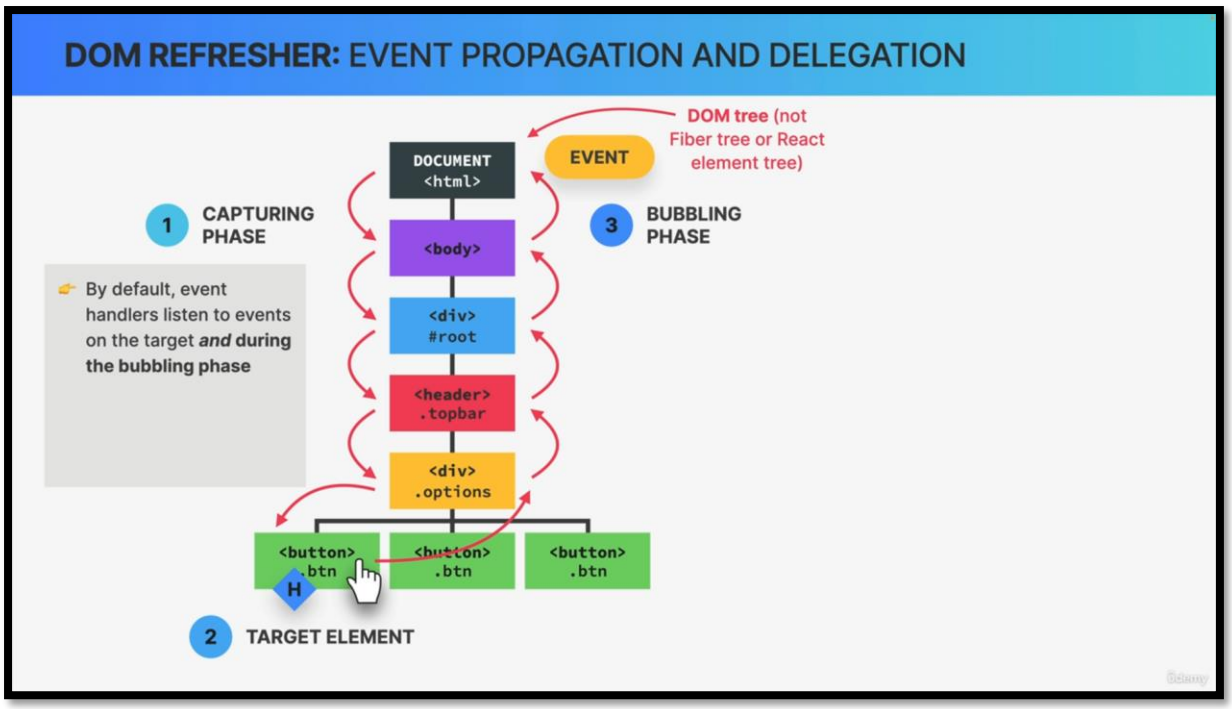
At the target, we can choose to handle the event by placing an event handler function on that element which usually is exactly what we do.



Then immediately after the target element has been reached, the event object travels all the way back up the entire tree during the so-called bubbling phase.

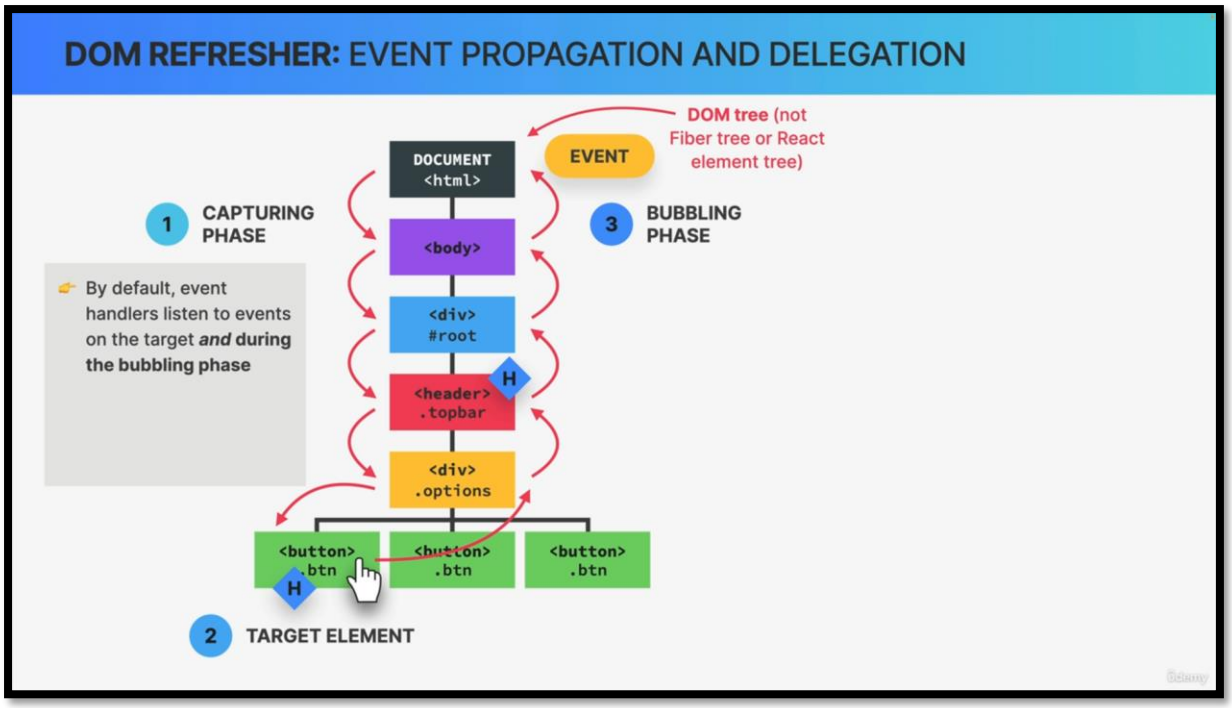
Now, there are two very important things to understand about this process.

1. The first is that during the capturing and bubbling phase, the event really goes through every single child and parent element one by one. In fact, it's if the event originated or happened in each of these DOM elements.

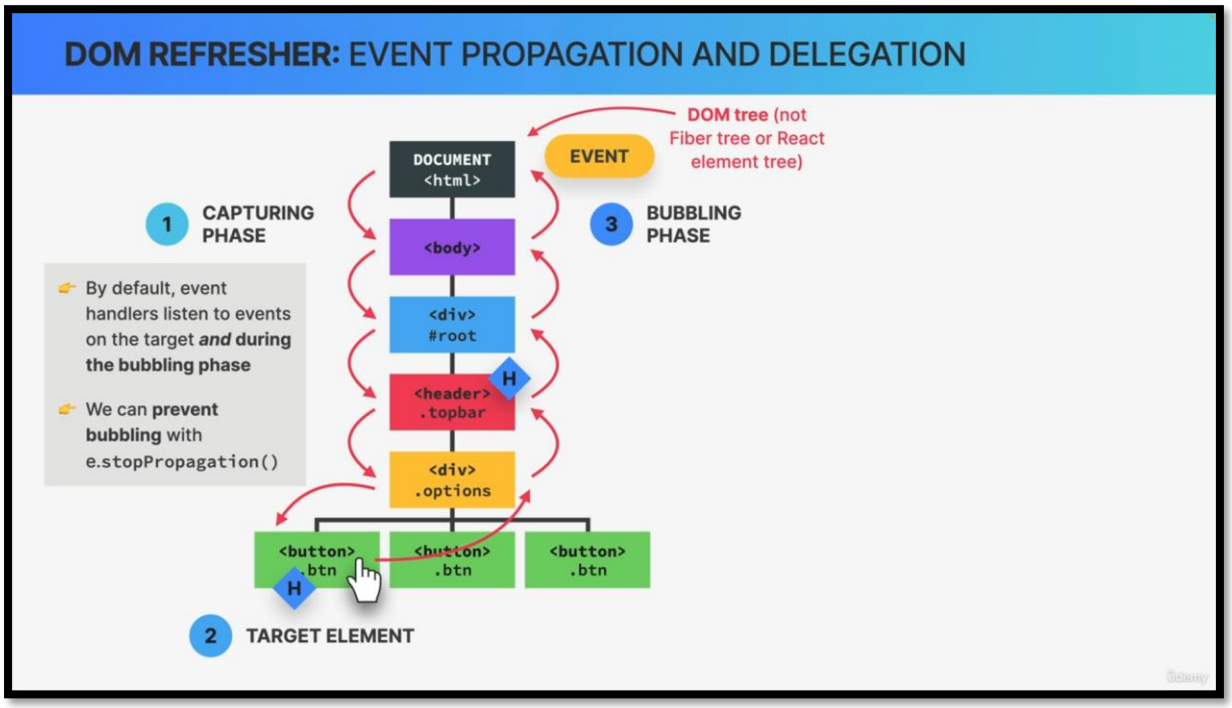


2. The second important thing is that by default, event handlers listen to events not only on the target element but also during the bubbling phase.

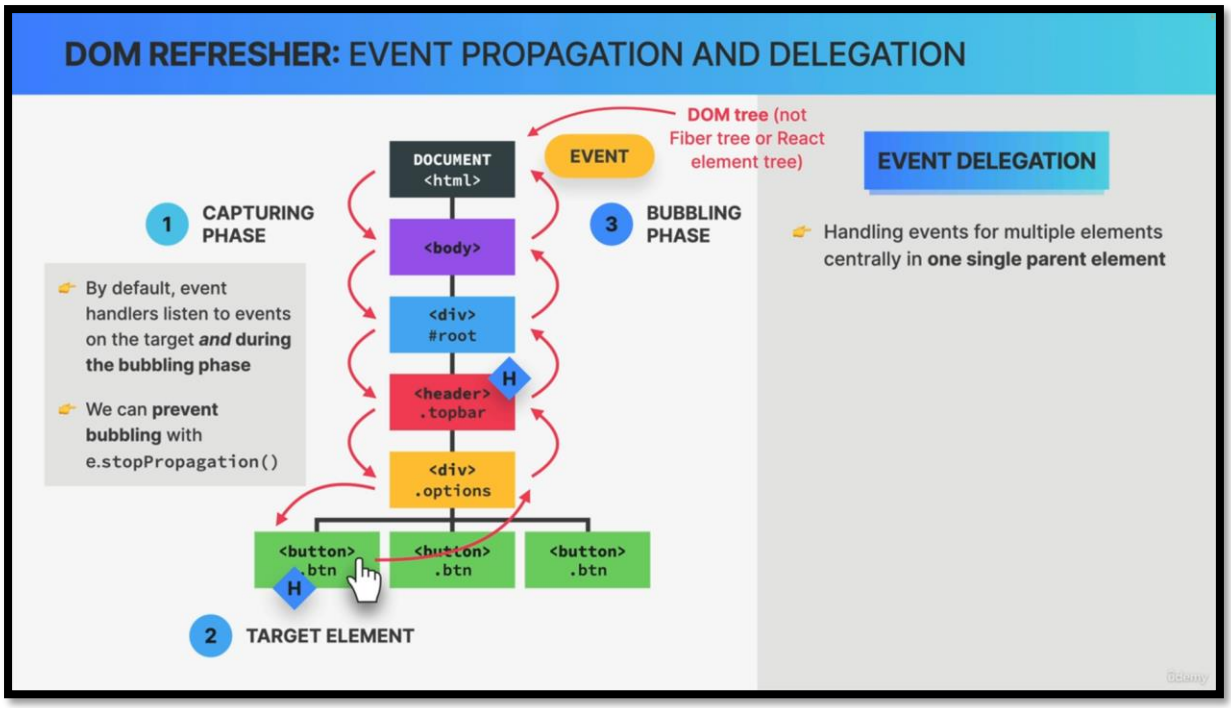
So, if we put these two things together, it means that every single event handler in a parent element will also be executed during the bubbling phase as long as it's also listening for the same type of event.



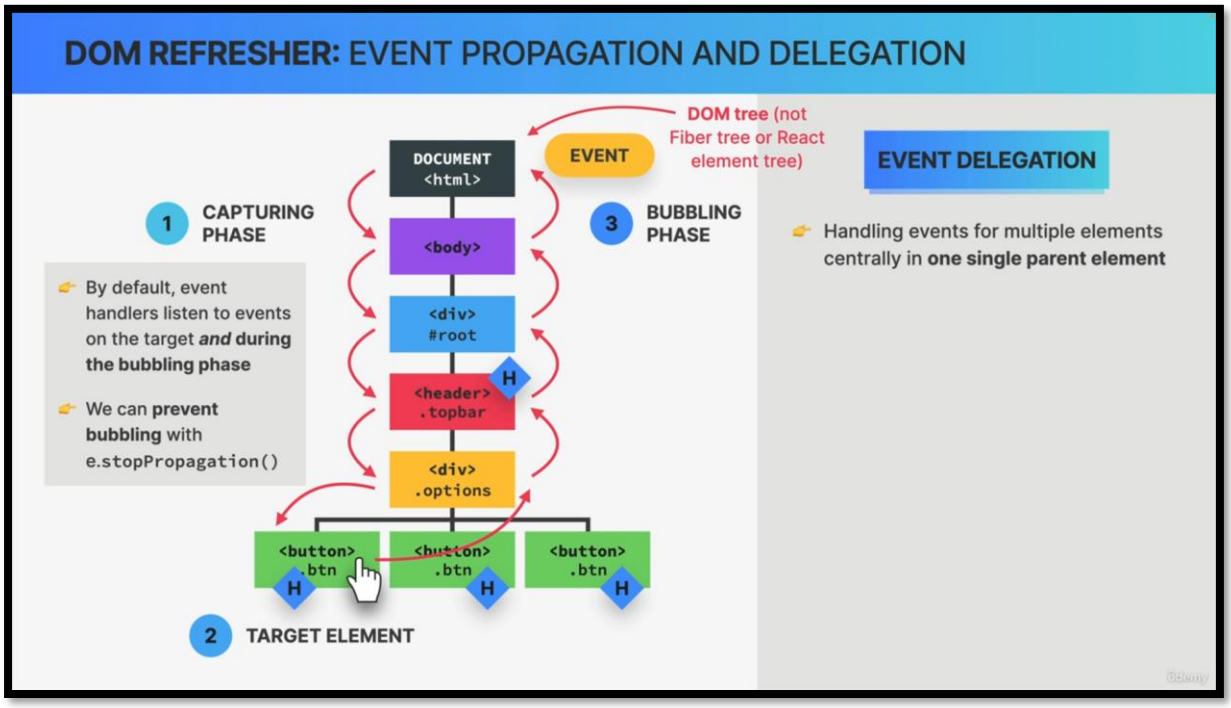
For example, if we add another click event handler to the header element, then during this whole process, both the handlers at the target and the header element would be executed when the click happens.



Now, sometimes we actually don't want this behaviour, and so in that case, we can prevent the event from bubbling up any further simply by calling the stopPropagation method on the event object, and this works in vanilla JavaScript, and also in React, but it's actually very rarely necessary, so only use this if there really is no other solution.



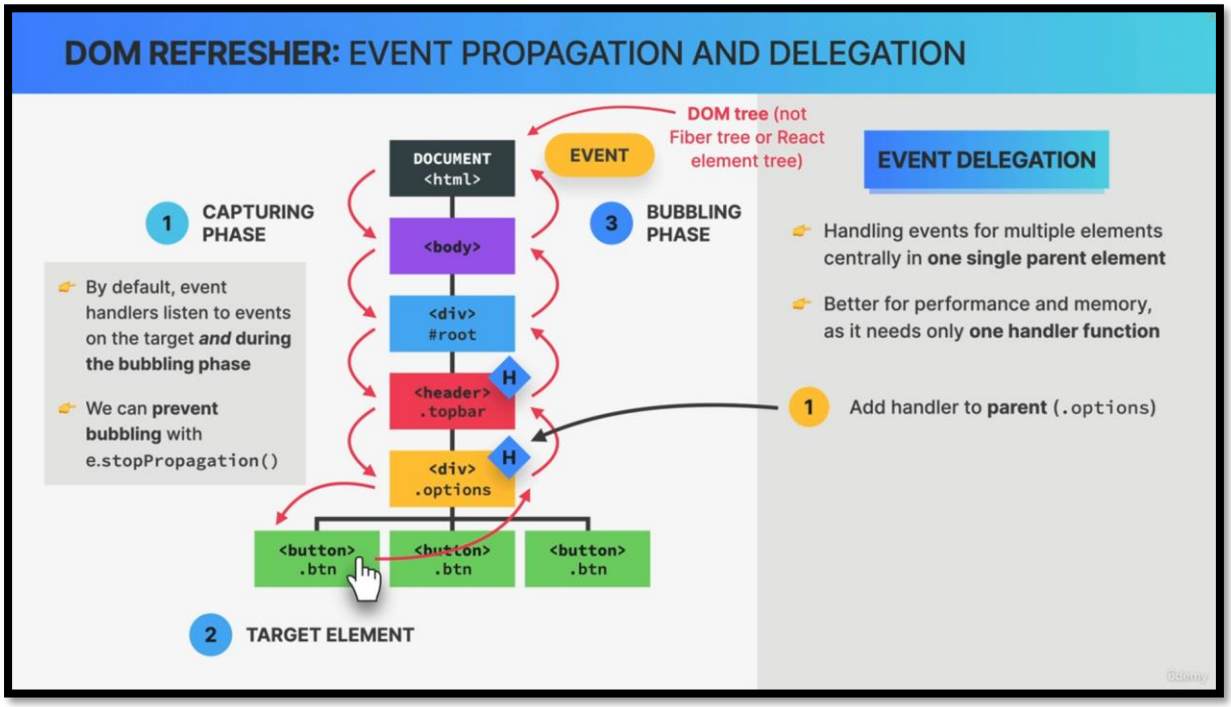
The fact that events bubble like this allows developers to implement a very common and very useful technique called event delegation.

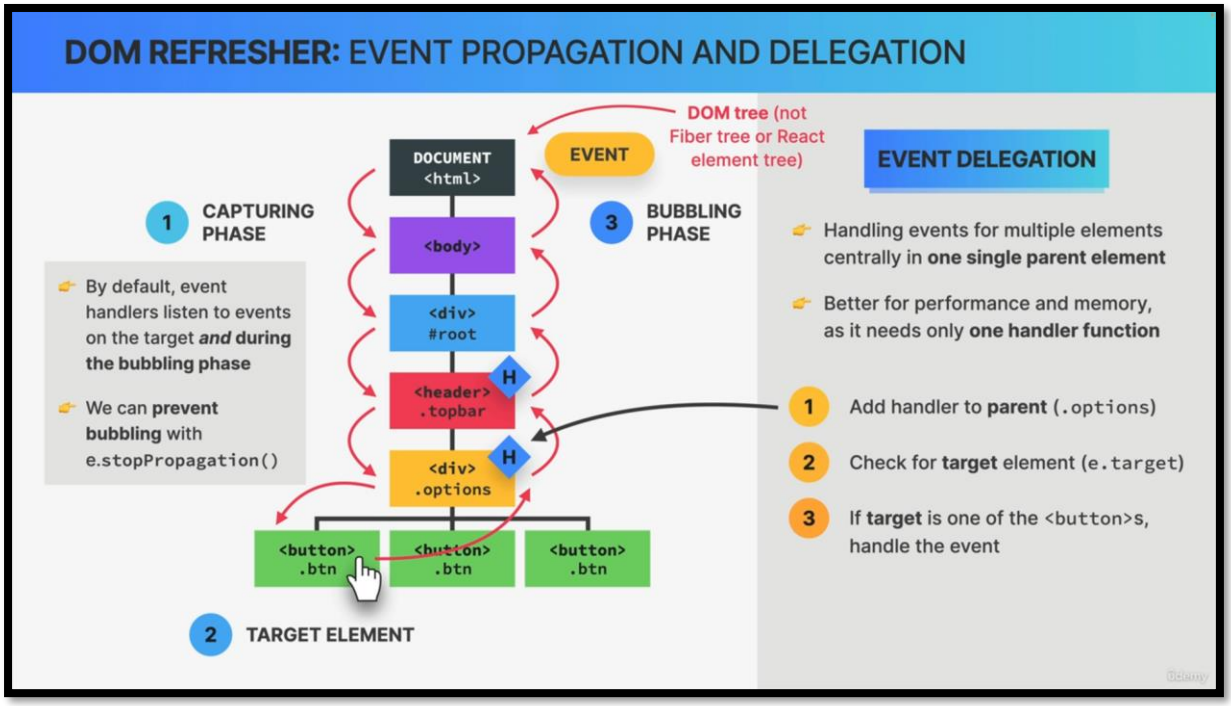


So, with event delegation, we can handle events for multiple elements in one central place which is one of the parent elements.

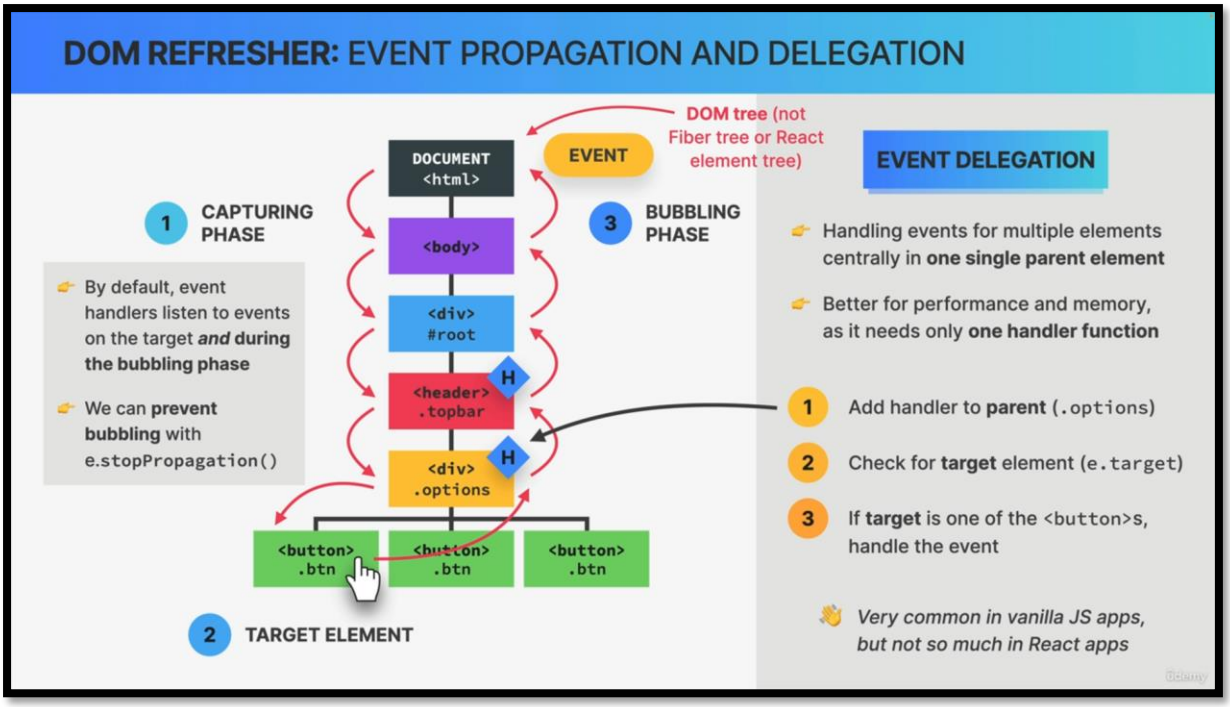
So, imagine that instead of three buttons, there would be like, 1,000 buttons. Now, if we wanted to handle events on all of them, each button would have to have its own copy of the event handler function, which could become problematic for the app's performance and memory usage.

So instead, by using event delegation, we can simply add just one handler function to the first parent element of these buttons. Then when a click happens on one of the buttons, the event will bubble up to the options div in this example where we can then use the events target property in order to check whether the event originated from one of the buttons or not, and if it did, we can then handle the event in this central event handler function.





DOM REFRESHER: EVENT PROPAGATION AND DELEGATION



1 CAPTURING PHASE

- By default, event handlers listen to events on the target ***and*** **during the bubbling phase**
- We can **prevent bubbling** with `e.stopPropagation()`

3 BUBBLING PHASE

2 TARGET ELEMENT

EVENT DELEGATION

- Handling events for multiple elements centrally in **one single parent element**
- Better for performance and memory, as it needs only **one handler function**

1. Add handler to **parent** (`.options`)
2. Check for **target** element (`e.target`)
3. If **target** is one of the `<button>`s, handle the event

👋 *Very common in vanilla JS apps, but not so much in React apps*