We have 3 main event concepts which are very useful to understand internal working of event:

- Event Bubbling
- Event Capture
- Event Propagation

Event Bubbling:

Event Bubbling is a concept in which, if we trigger an event on any child, then that event is also tracked by ancestors(like parents) of the corresponding child.

```
<main>
 <section>
   <div>
       <form id="form">
          <input id="inp" types="text" />
       </form>
       <button id = "btn">
            Click Me
        </button>
       </div>
    </section>
  </main>
```

Here if we attach click event listener to the `button` then also start tracking click event on the parent div, grandparent section and great grand parent main then the click event will also be tracked by them.

```
document.getElementById("btn").addEventListener("click",()=>{
  console.log("button clicked");
});

document.querySelector("main").addEventListener("click",()=>{
  console.log("event reached main");
});
document.querySelector("section").addEventListener("click",()=>{
  console.log("event reached section");
});

document.querySelector("div").addEventListener("click",()=>{
```

```
  console.log("event reached div");
});
```

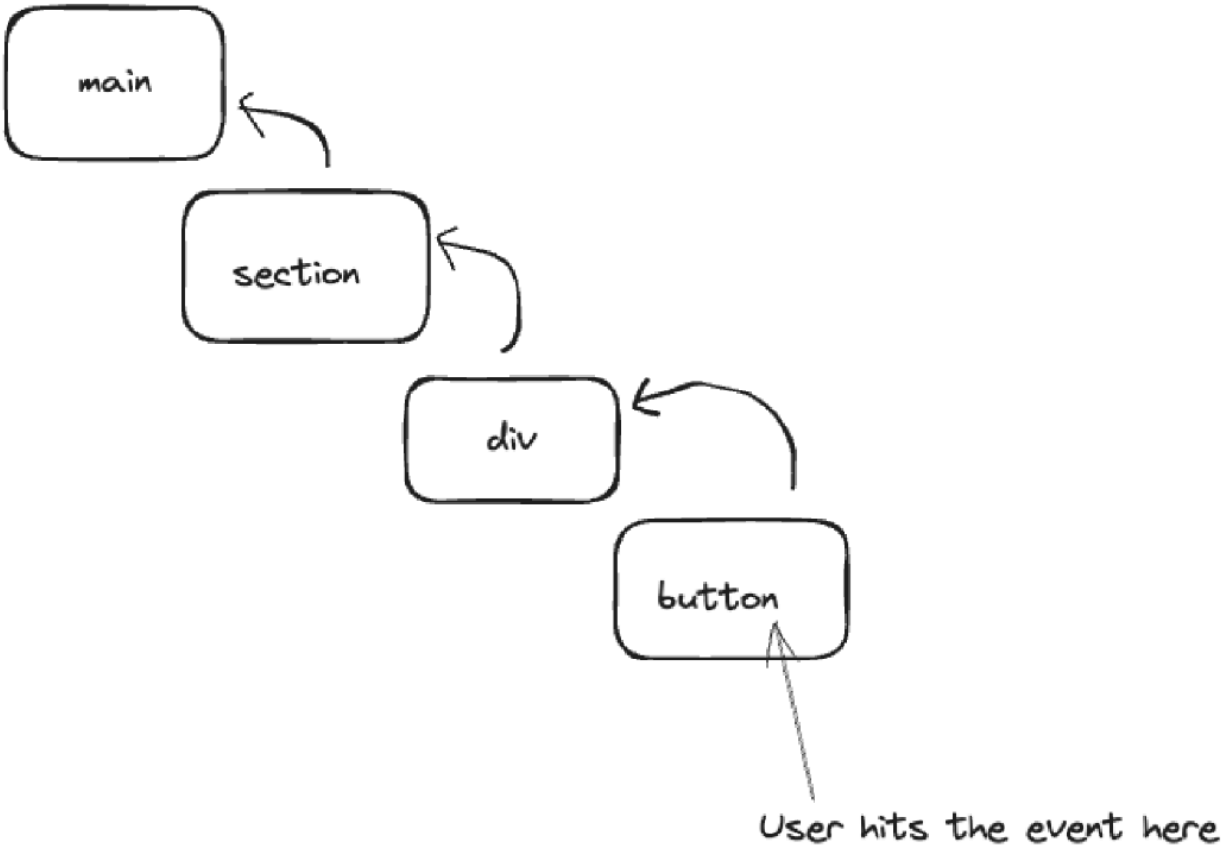
If we click on the button the output we will get is:

`"button clicked"`

`"event reached div"`

`"event reached section"`

`"event reached main"`



can we stop event bubbling ?

yes, we can do it by using `event.stopPropagation`. So on whatever element we call event.stopPropagation, right on that element event bubbling stops.

```
document.getElementById("btn").addEventListener("click",()=>{
  console.log("button clicked");
});

document.querySelector("main").addEventListener("click",(event)=>{
  console.log("event reached main");
  event.stopPropagation();
```

```
});
document.querySelector("section").addEventListener("click",()=>{
  console.log("event reached section");
});

document.querySelector("div").addEventListener("click",()=>{
  console.log("event reached div");
});
```

output will be:

`button clicked`
`event reached main`
because we use event.stopPropagation method to stop execution of other events.

Event Delegation

This is a technique in which instead of starting from child to the parent everybody listens for the event, we just delegate this task to the parent only to listen to the corresponding fired events.

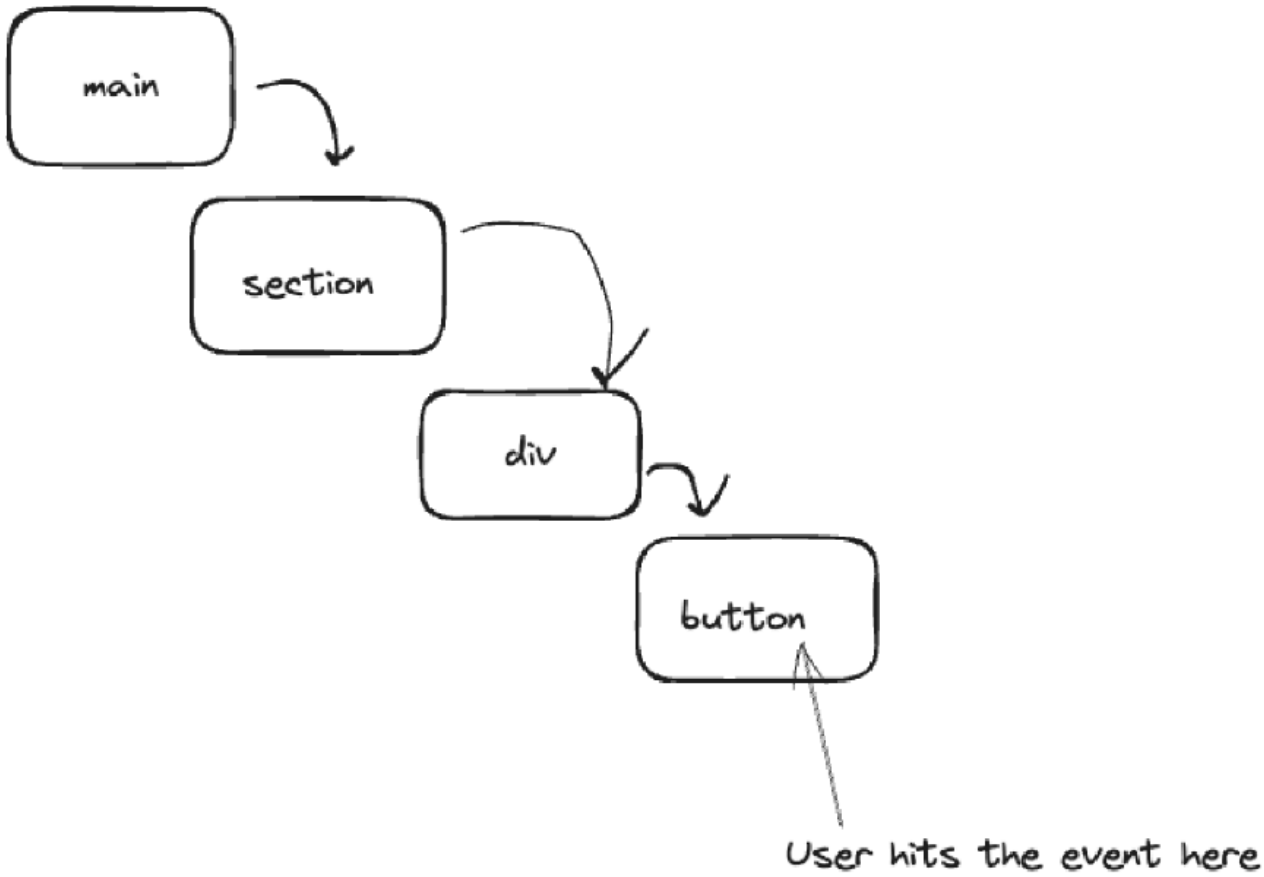
```
document.querySelector("main").addEventListener("click",()=>{
  console.log("event reached main");
});
```

Here instead of the whole , we selected one of the ancestors and then, added the event listener to them.

Event Capturing:

The `addEventListener` method takes a third argument called as `useCapture` which is a boolean argument (by default it is false). This argument enables event capturing for us. what is event capturing?

Event capturing is a mechanism in which event on the parent is tracked first and then event on any nested child is tracked.



So here if the user clicks on the button then, first the click on event of main is trigger then section then div and then button. And we can stop this capturing anywhere in between using `event.stopPropagation`.

```
document.getElementById("btn").addEventListener("click",()=>{
  console.log("button clicked");
}, true);

document.querySelector("main").addEventListener("click",()=>{
  console.log("event reached main");
},true);
document.querySelector("section").addEventListener("click",()=>{
  console.log("event reached section");
},true);

document.querySelector("div").addEventListener("click",()=>{
  console.log("event reached div");
},true);
```

So here the third parameter is passed as true enabling event capture. If we enable useCapture on only few elements then only those will be tracked with their capturing phase, else everyone

will work with bubbling.