EVENT LISTENERS AND HANDLERS

Listen up!

YOU SHOULD BE ABLE TO

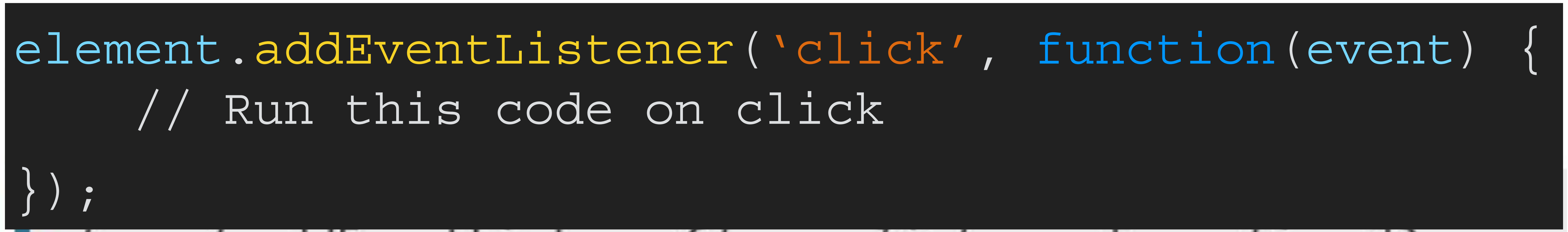
- Understand what **events** are
- Implement an **event listener**
- Understand event **propagation** and event **delegation**
- Understand **this** in the context of events

WHAT ARE EVENTS?

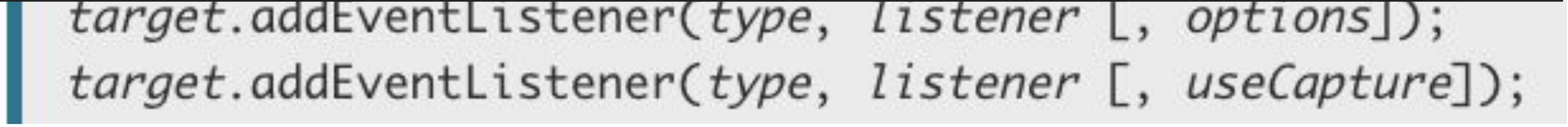
- **Actions** or **occurrences** that happen in the system you are programming, which the system tells you about so you can respond to them in some way ***if desired***
- Event examples:
  - click
  - submit (for forms)
  - mouseover
  - scroll

EVENT HANDLERS

- Blocks of JavaScript code that run ***when*** an event is fired in the DOM



```
element.addEventListener('click', function(event) {
    // Run this code on click
});
```



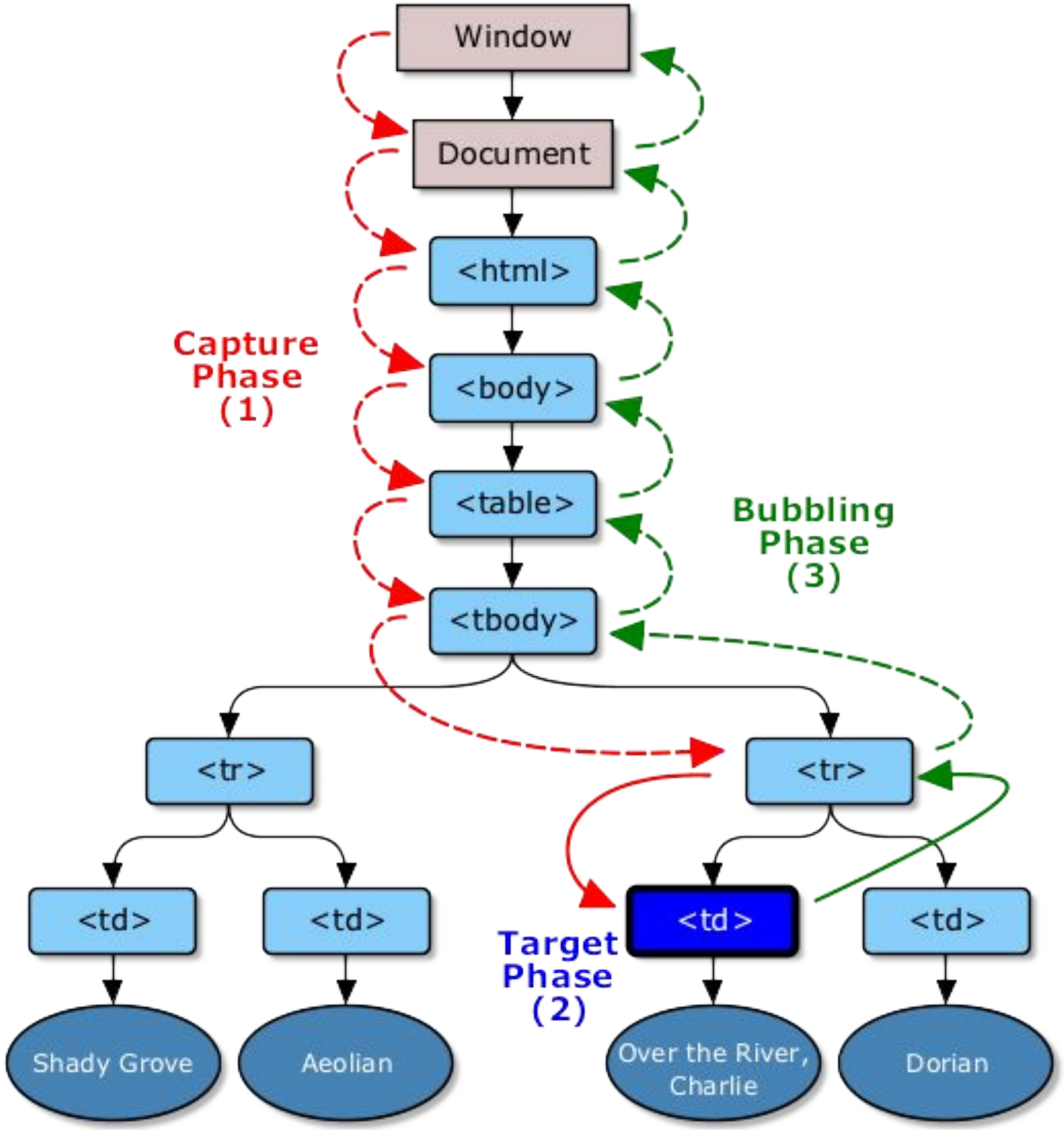
```
target.addEventListener(type, listener [, options]);
target.addEventListener(type, listener [, useCapture]);
```

MDN: EventTarget.addEventListener()

EVENT LIFE CYCLE

- Capturing Phase:
  - From the root, an event is ***directed*** to its intended target
  - If there is a **matching event listener** along the way, it is **triggered**
- Target Phase:
  - The event **reaches its intended target** and the event fires on the target node
  - If there is a **matching event listener**, it is **triggered**
- Bubbling Phase:
  - From the intended target, the **event *bubbles* up** back up to the root of the document
  - If there is a **matching event listener** along the way, it is **triggered**

EVENT LIFE CYCLE

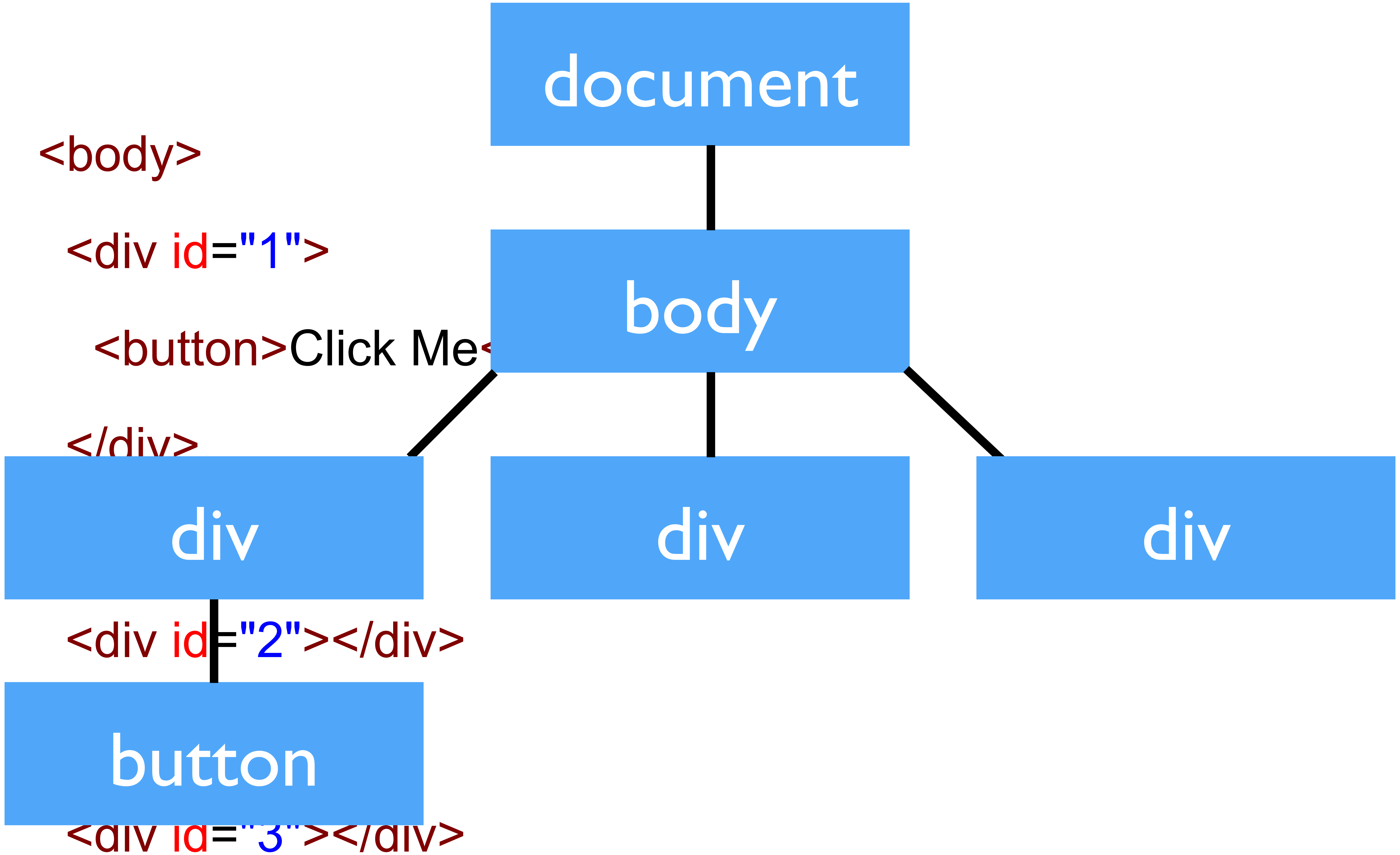


BUBBLING PHASE

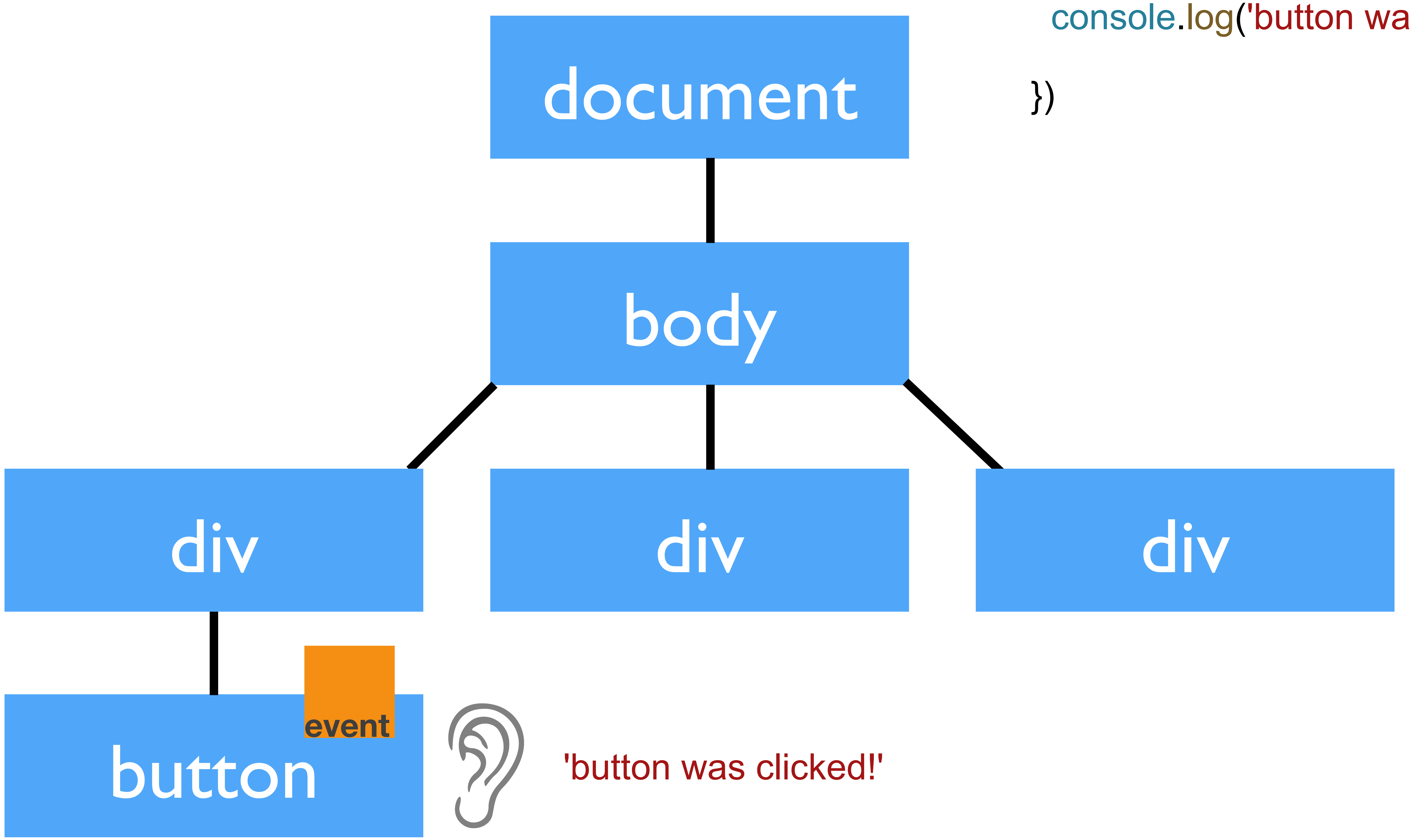
```
<body>
  <div id="1">
    <button>Click Me</button>
  </div>

  <div id="2"></div>

  <div id="3"></div>
</body>
```

BUBBLING PHASE

```
<body>
  <div id="1">
    <button>Click Me<
  </div>
  <div id="2"></div>
  <div id="3"></div>
</body>
```



```
const button = document.getElementsByTagName('button')[0]

button.addEventListener('click', function (evt) {

  console.log('button was clicked!')

})
```



document

body

div

div

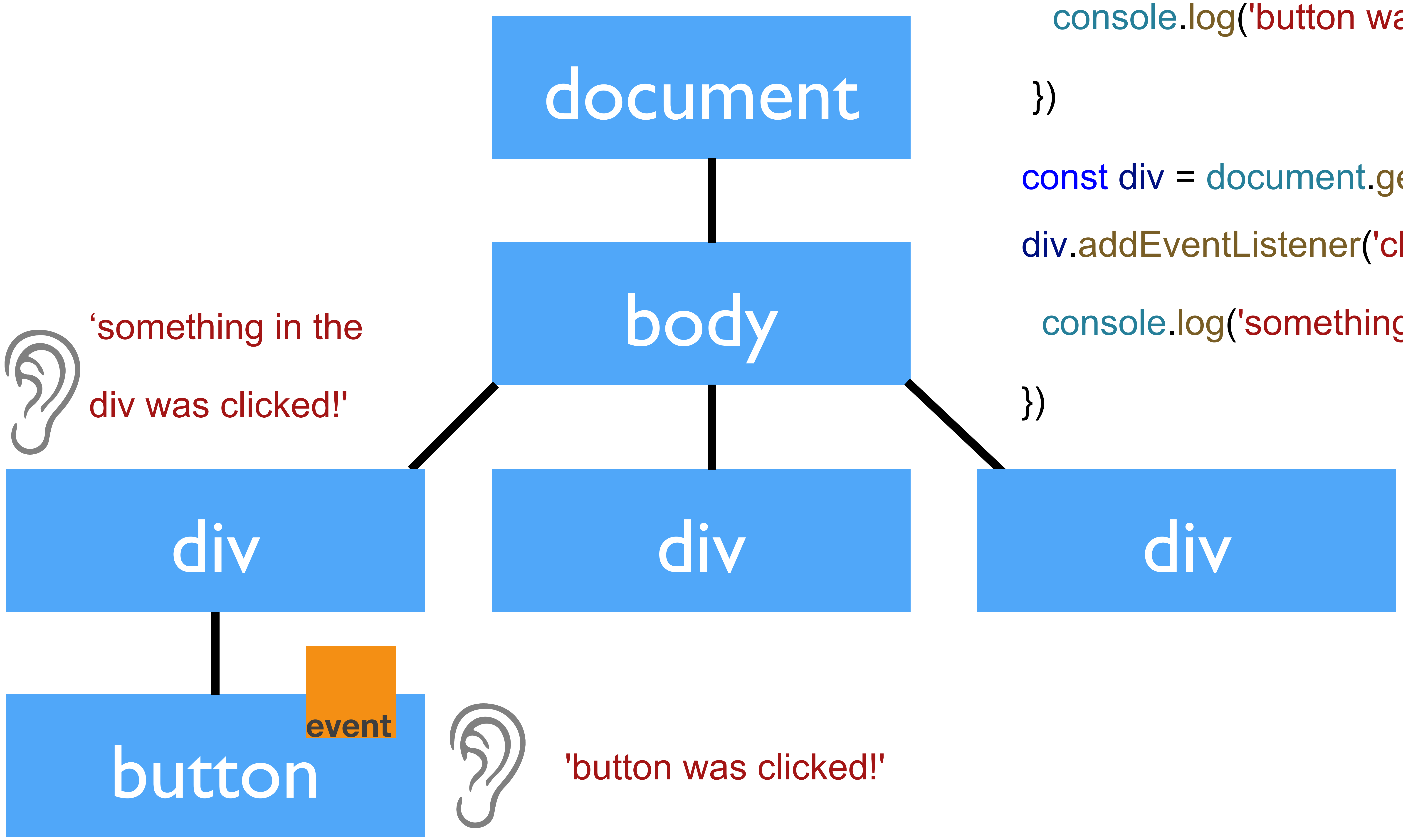
div

button

event

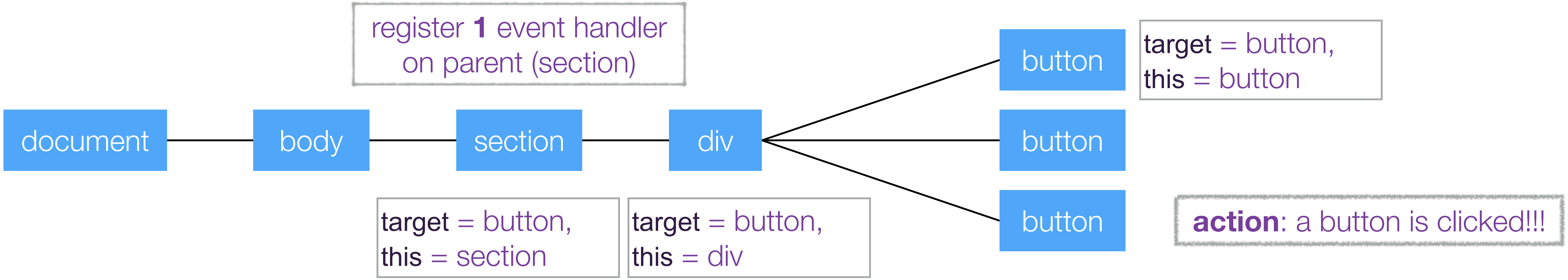
'button was clicked!'

```
const button = document.getElementsByTagName('button')[0]

button.addEventListener('click', function (evt) {

  console.log('button was clicked!')

})

const div = document.getElementById('one')

div.addEventListener('click', function (evt) {

  console.log('something in the div was clicked!')

})
```



EVENT DELEGATION

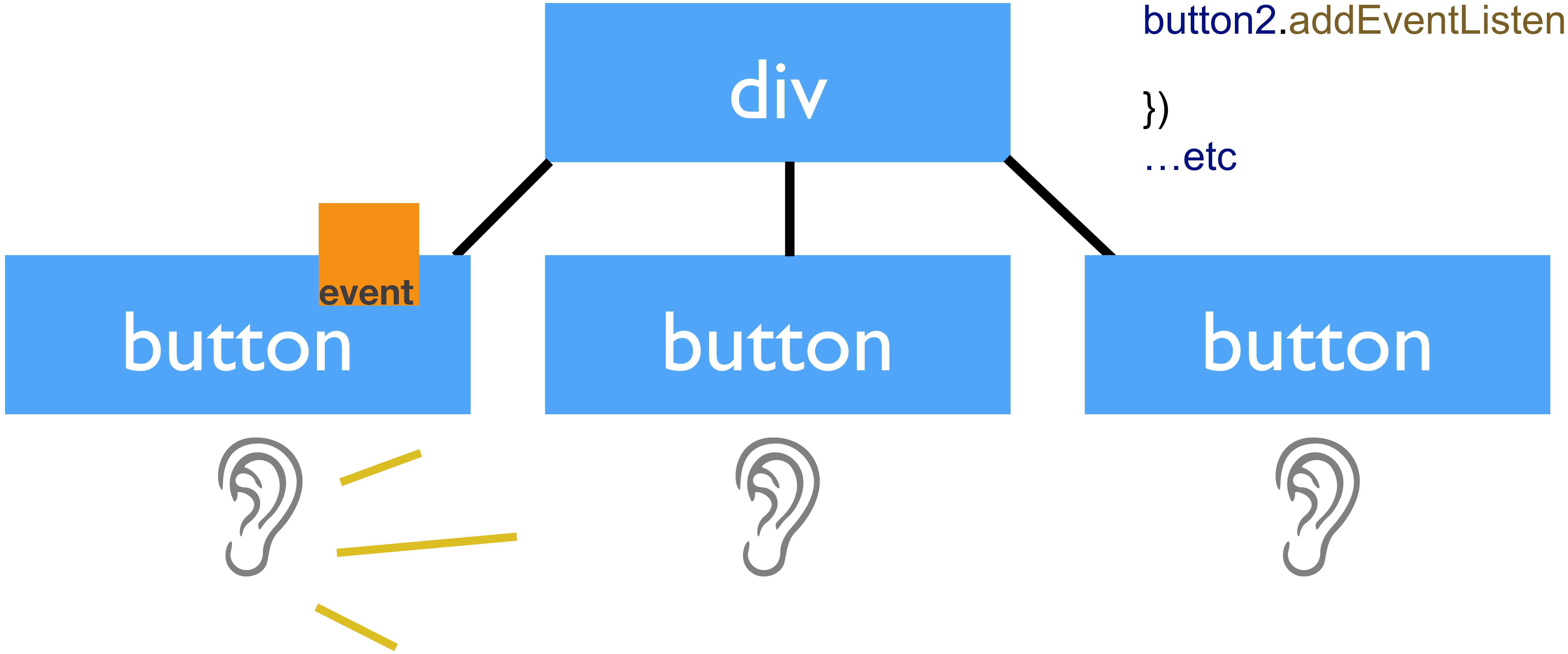
- The process of using event propagation to handle events at a higher level in the DOM
- Allows for a single event listener



register **1** event handler on parent (section)
document
body
section
div
button
button
button
target = button, this = button
target = button, this = section
target = button, this = div
action: a button is clicked!!!

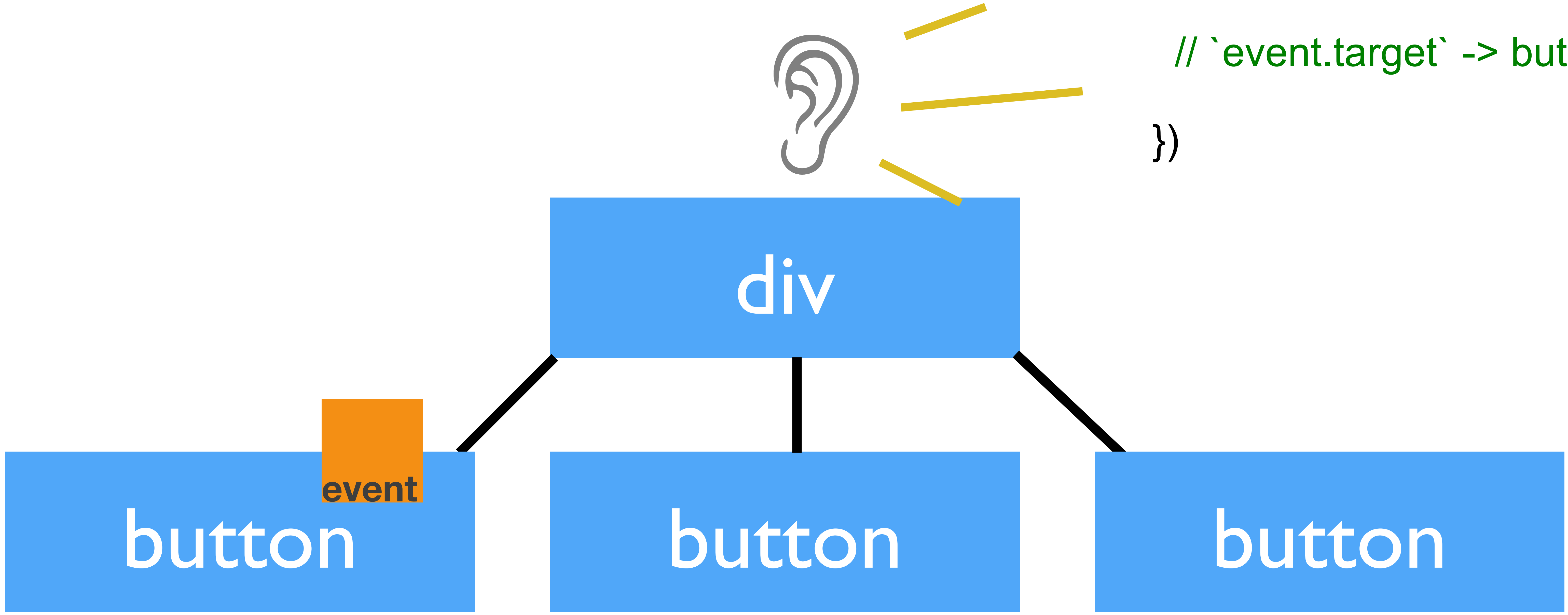
Without Event Delegation

```
const button1 = document.getElementsByTagName(‘button')[0]

const button2 = document.getElementsByTagName(‘button')[1]

const button3 = document.getElementsByTagName(‘button')[2]


button1.addEventListener('click', function (event) {

})
button2.addEventListener(‘click', function (event) {

})
…etc
```



With Event Delegation

```
const div = document.getElementById('button-container')

div.addEventListener('click', function (event) {

  // `this` -> div

  // `event.target` -> button

})
```



THIS

THIS

- …is the "context" for a function.
- …is determined when a function is ***invoked***, not when it is defined **(*exception*: arrow functions)**.

For event handlers, "*this*" is whatever the event handler was attached to.

"*event.target*" is whatever triggered the event. (i.e. the clicked button in the previous "Event Delegation" slide).

TYPES OF CONTEXT BINDING AND CALL-SITE

- Default binding: **func();**
- Implicit binding: **obj.func();**
- Explicit binding: **func.call(obj);**
- “new” binding: **new func();**

THE **.BIND** METHOD

- Requires one argument, a **thisArg**.
- Returns a new function whose **this** is always the **thisArg**.
- Does ***not*** invoke the function. It makes a ***copy*** of the function it's called on.

```
const boundFunc = oldFunc.bind(thisArg);

boundFunc(); //invoked with thisArg as this
```

Is it an arrow func?

Same as `this` in the scope where it was defined

Invoked using `new`?

The “new” instance object

Is it a bound func?

The object passed in as the first argument to bind

Invoked with call/apply?

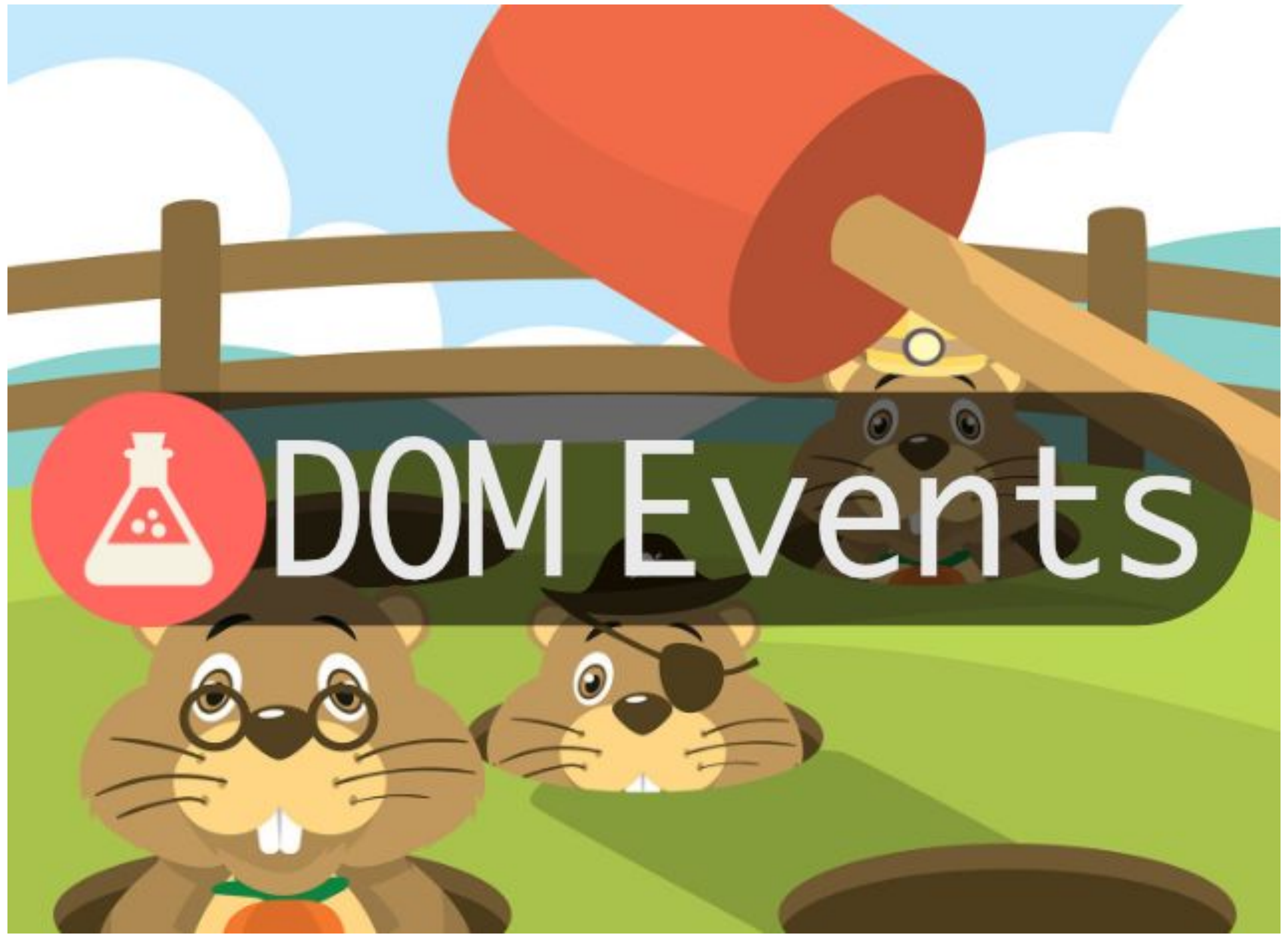
The object passed in as the first argument to call/apply

Invoked on an object?

The object

The global context (or undefined when in ‘strict mode’)

LAB: WHACK-A-MOLE



PAIR EXERCISE: PIXELATE