CHAPTER 5

DOCUMENT OBJECT MODEL

The DOM specifies how:

The DOM specifies how:

1

Browsers
create a model of
an HTML page

The DOM specifies how:

1

Browsers
create a model of
an HTML page

2

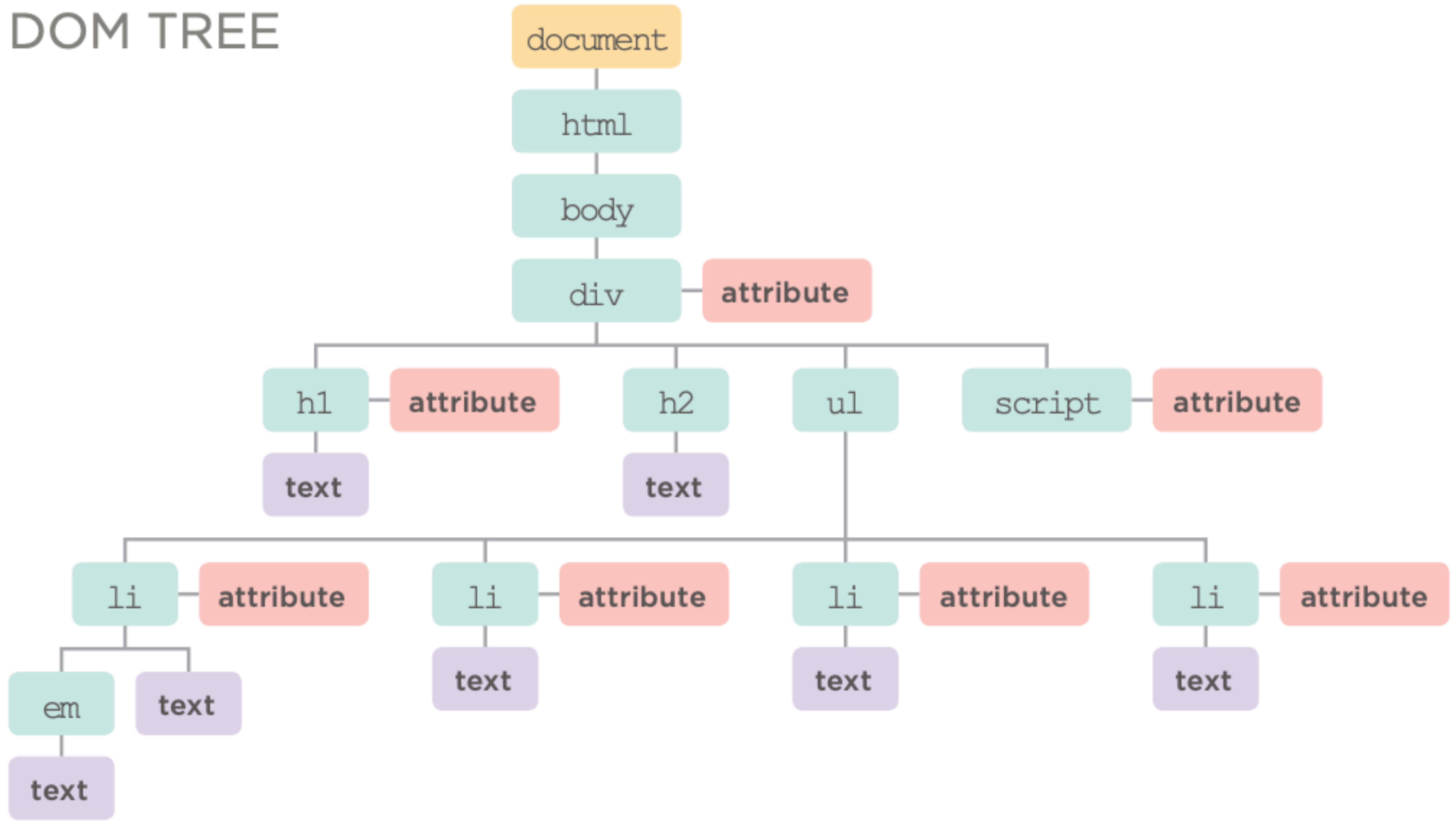
JavaScript
accesses / updates
an HTML page

THE DOM TREE

BODY OF HTML PAGE

```
<html>
  <body>
    <div id="page">
      <h1 id="header">List</h1>
      <h2>Buy groceries</h2>
      <ul>
        <li id="one" class="hot"><em>fresh</em> figs</li>
        <li id="two" class="hot">pine nuts</li>
        <li id="three" class="hot">honey</li>
        <li id="four">balsamic vinegar</li>
      </ul>
      <script src="js/list.js"></script>
    </div>
  </body>
</html>
```

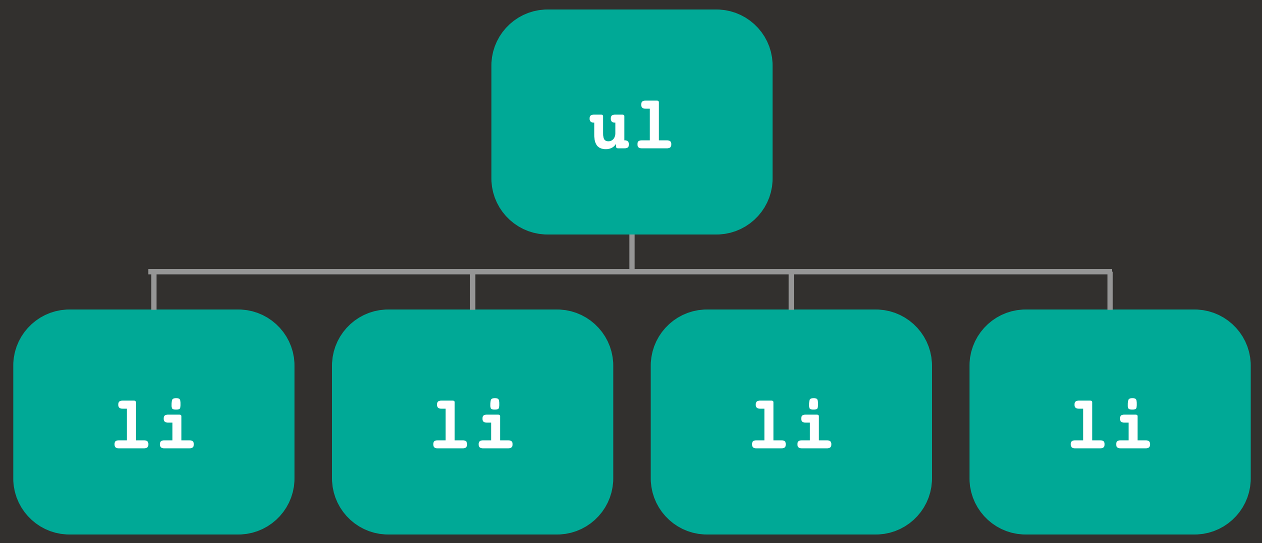
DOM TREE



```
<ul>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
</ul>
```

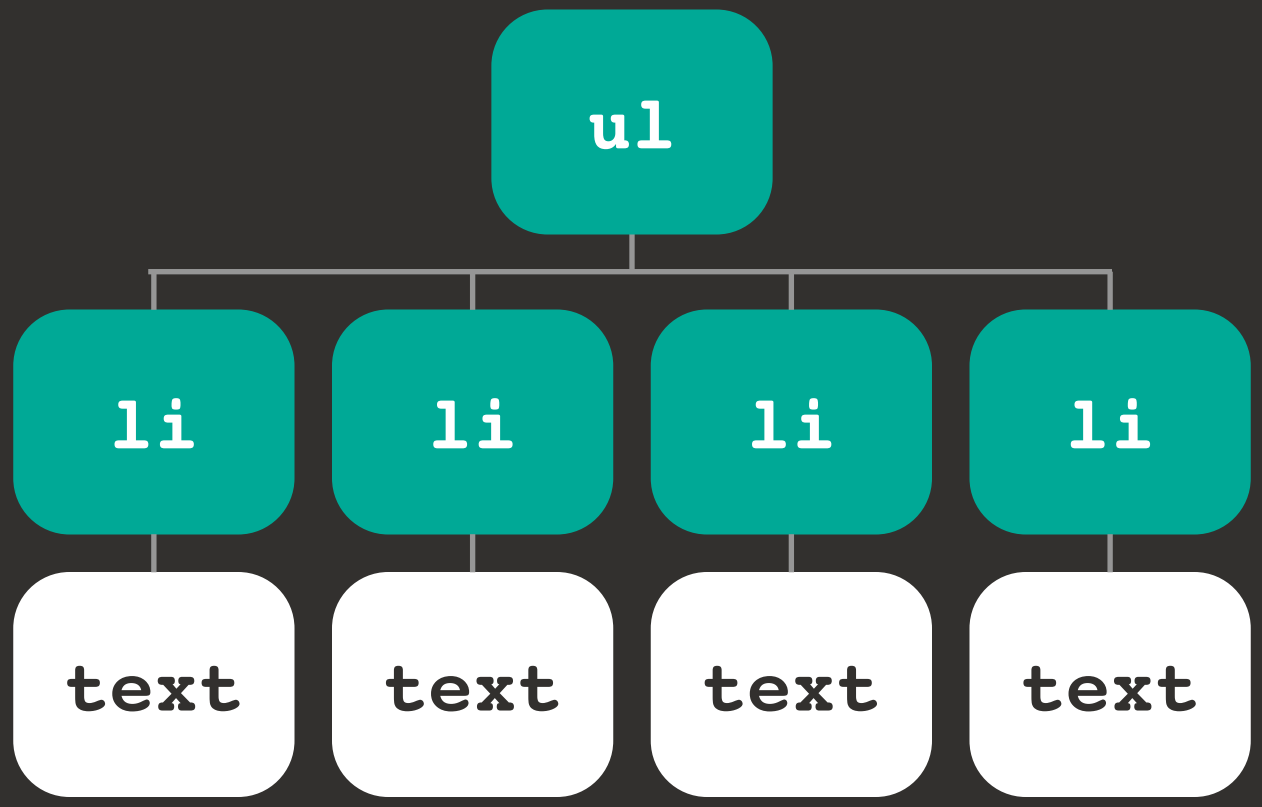
ELEMENT NODES

```
<ul>
  <li></li>
  <li></li>
  <li></li>
  <li></li>
</ul>
```



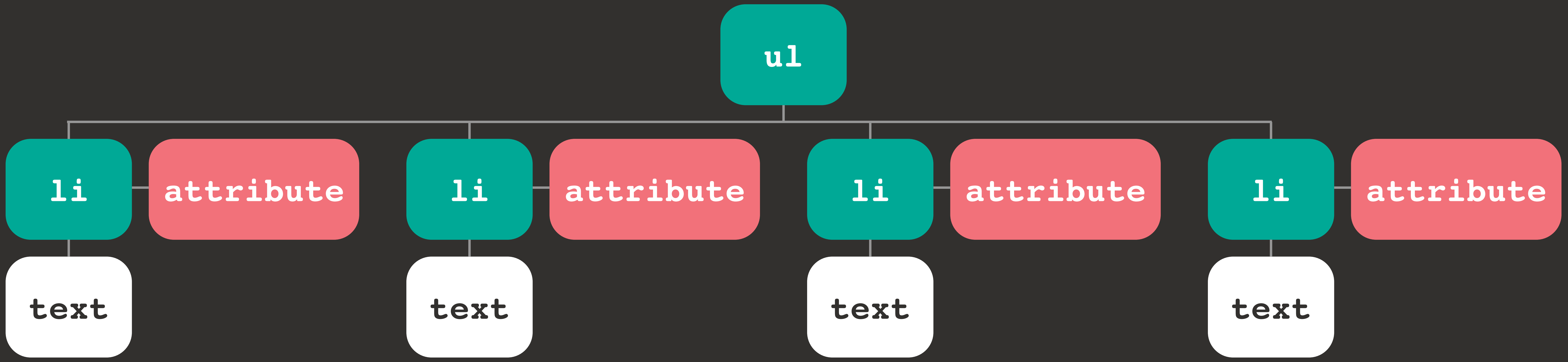
TEXT NODES

```
<ul>
  <li>fresh figs</li>
  <li>pine nuts</li>
  <li>honey</li>
  <li>balsamic vinegar</li>
</ul>
```



ATTRIBUTE NODES

```
<ul>
  <li id=“one” class=“hot”>fresh figs</li>
  <li id=“two” class=“hot”>pine nuts</li>
  <li id=“three” class=“hot”>honey</li>
  <li id=“four”>balsamic vinegar</li>
</ul>
```



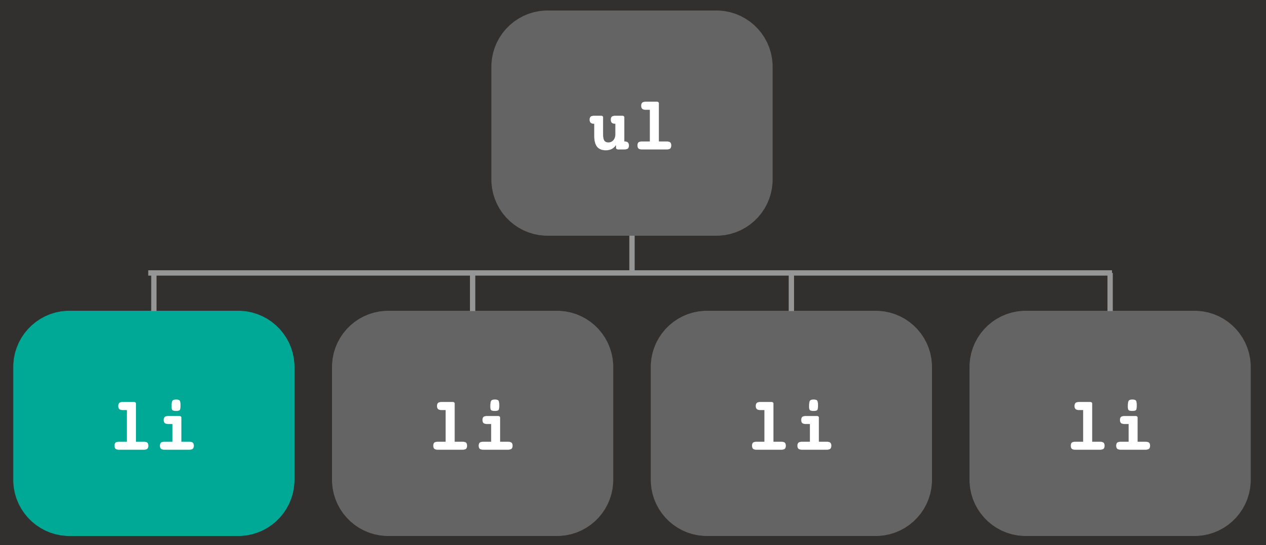
To access and update the HTML, first you select the element(s) you want to work with.

Here are some of the ways ways to select element nodes.

They are known as **DOM queries**.

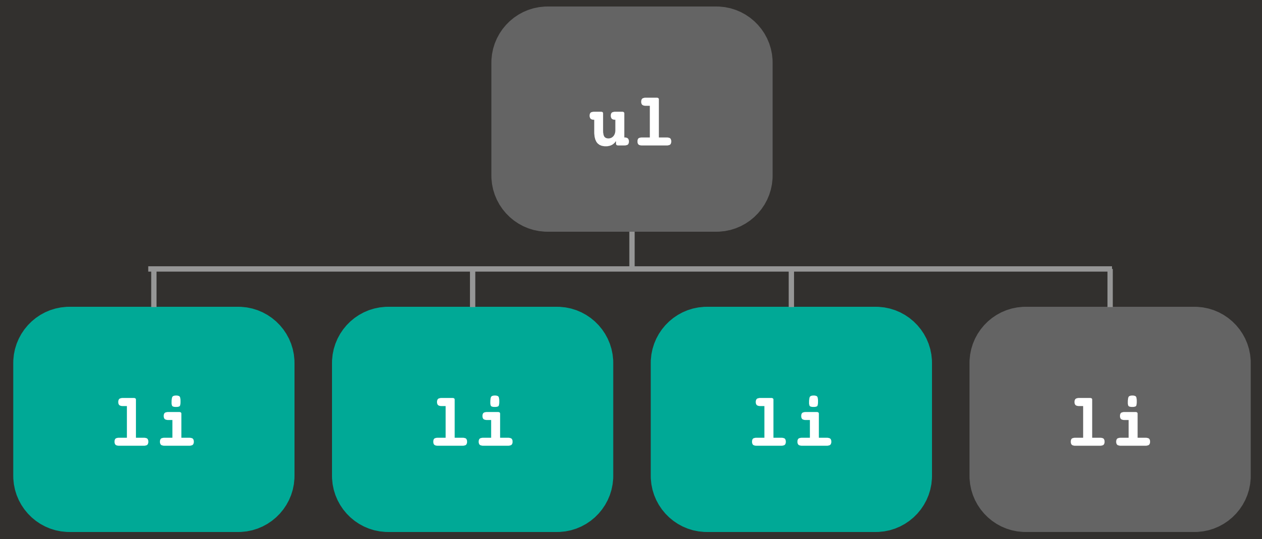
DOM QUERIES

```
<ul>
  <li id=“one” class=“hot”>fresh figs</li>
  <li id=“two” class=“hot”>pine nuts</li>
  <li id=“three” class=“hot”>honey</li>
  <li id=“four”>balsamic vinegar</li>
</ul>
```



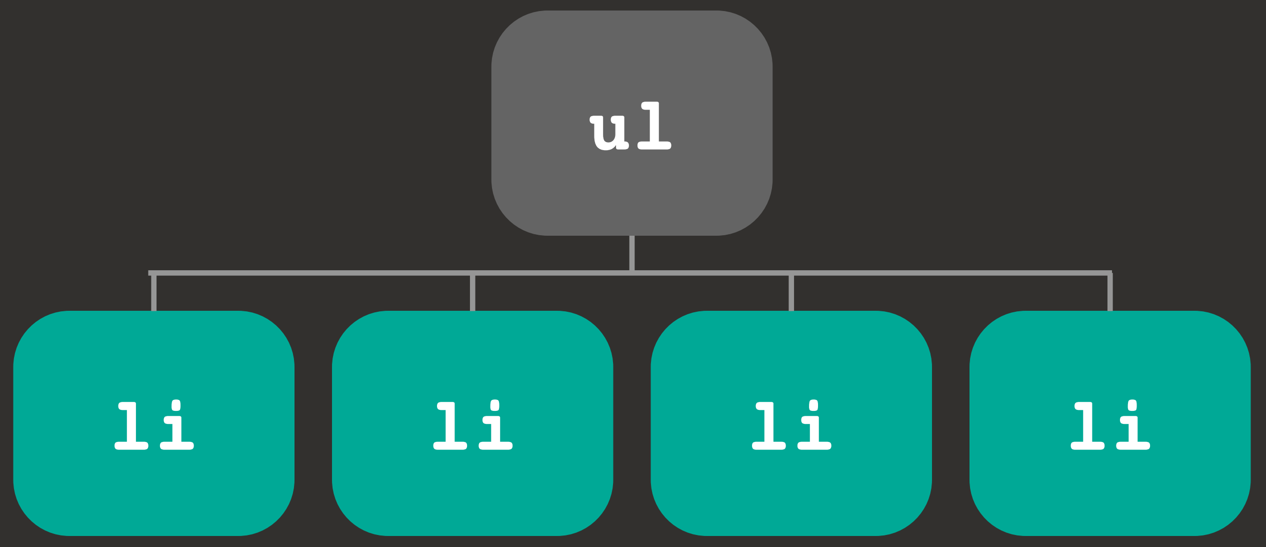
```
getElementById(‘one’);
```

```
<ul>
  <li id="one" class="hot">fresh figs</li>
  <li id="two" class="hot">pine nuts</li>
  <li id="three" class="hot">honey</li>
  <li id="four">balsamic vinegar</li>
</ul>
```



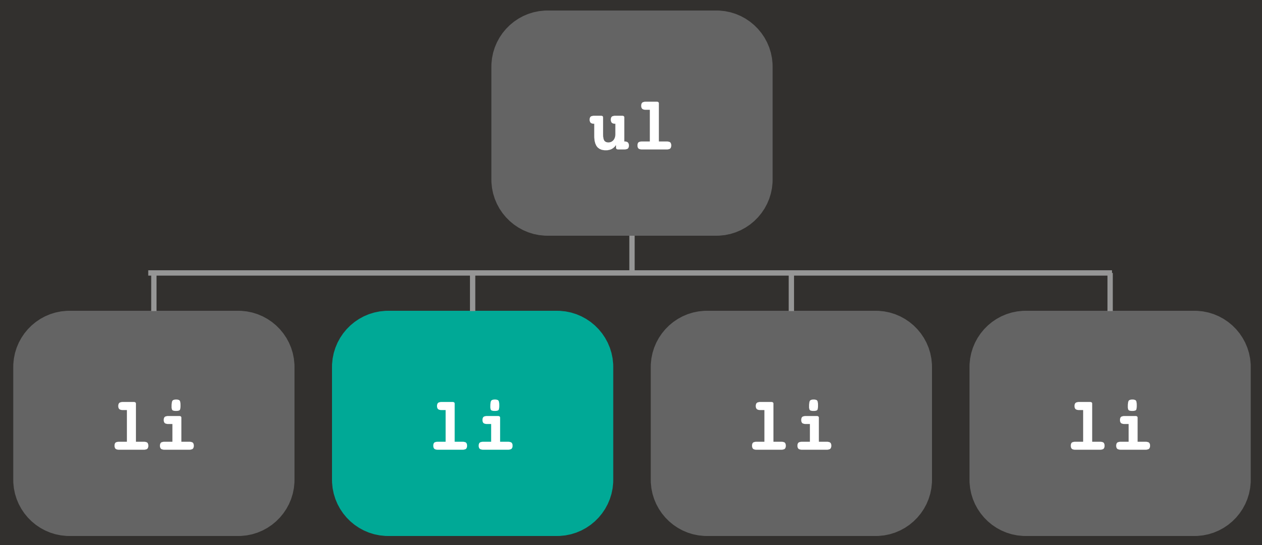
```
getElementsByClassName('hot');
```

```
<ul>
  <li id="one" class="hot">fresh figs</li>
  <li id="two" class="hot">pine nuts</li>
  <li id="three" class="hot">honey</li>
  <li id="four">balsamic vinegar</li>
</ul>
```



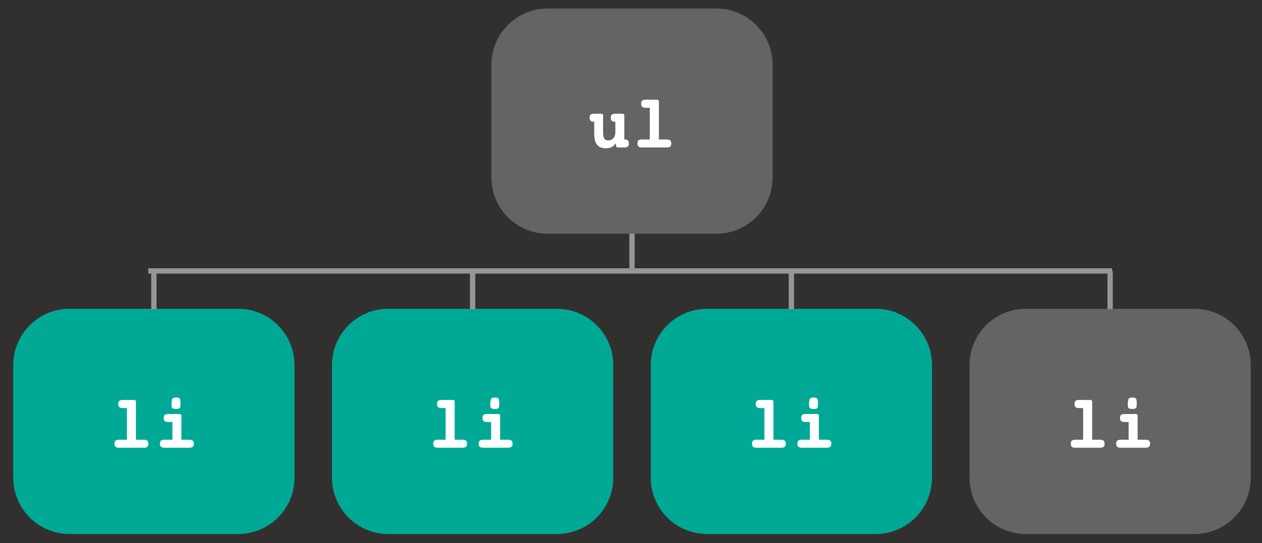
```
getElementsByTagName('li');
```

```
<ul>
  <li id=“one” class=“hot”>fresh figs</li>
  <li id=“two” class=“hot”>pine nuts</li>
  <li id=“three” class=“hot”>honey</li>
  <li id=“four”>balsamic vinegar</li>
</ul>
```



```
querySelector(‘#two’);
```

```
<ul>
  <li id="one" class="hot">fresh figs</li>
  <li id="two" class="hot">pine nuts</li>
  <li id="three" class="hot">honey</li>
  <li id="four">balsamic vinegar</li>
</ul>
```



```
querySelectorAll('li.hot');
```

NODELISTS

If a DOM query returns more than one element, it is known as a **NodeList**.

Items in a NodeList are numbered and selected like an array:

```
var elements;

elements = getElementsByClassName(‘hot’);

var firstItem = elements[0];
```

You can check if there are elements before using a NodeList:

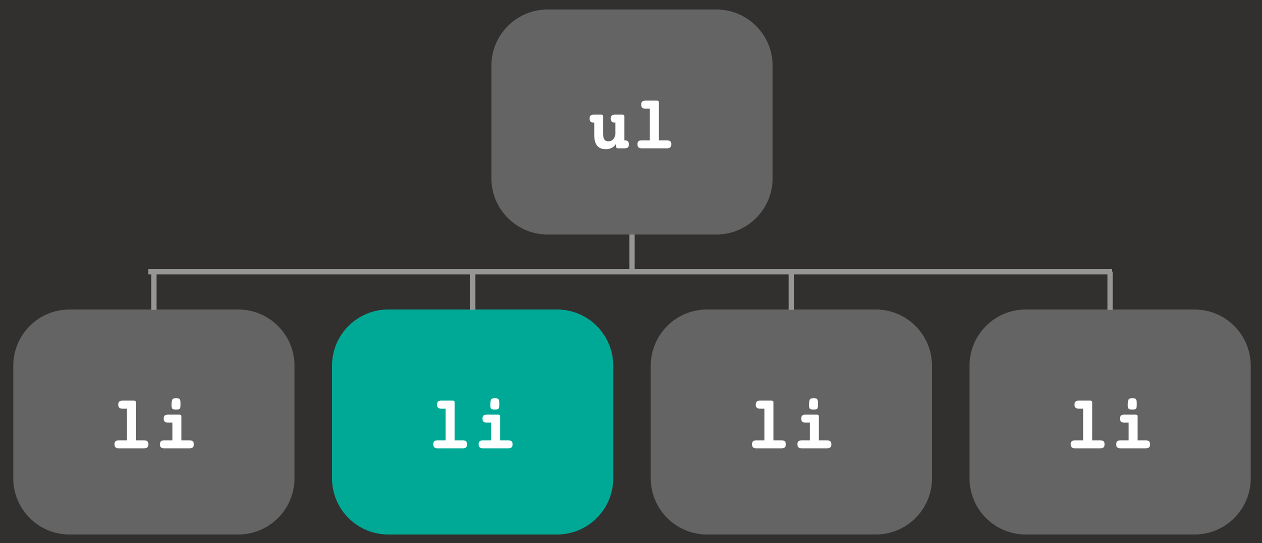
```
if (elements.length >= 1) {
  var firstItem = elements[0];
}
```

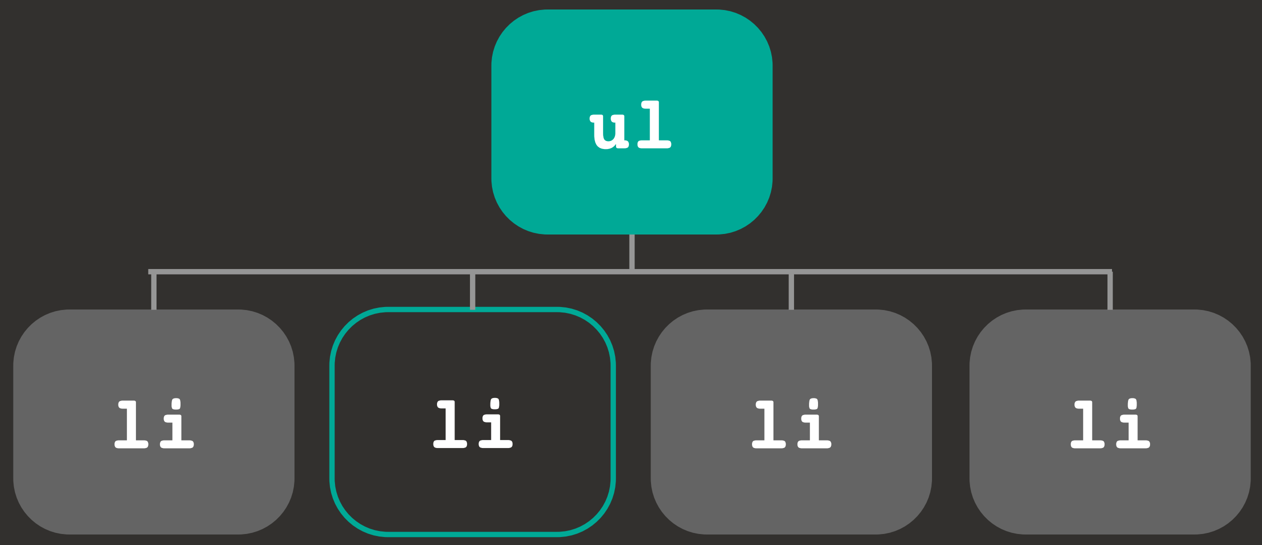
TRAVERSING THE DOM

You can move from one node to another if it is a relation of it.

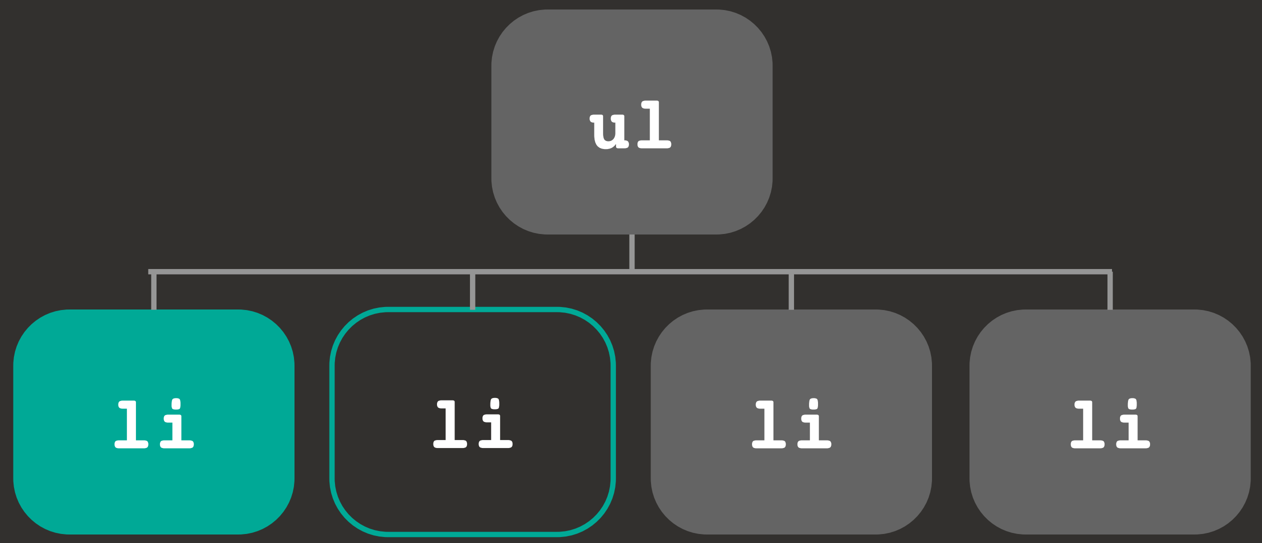
This is known as **traversing the DOM**.

STARTING ELEMENT

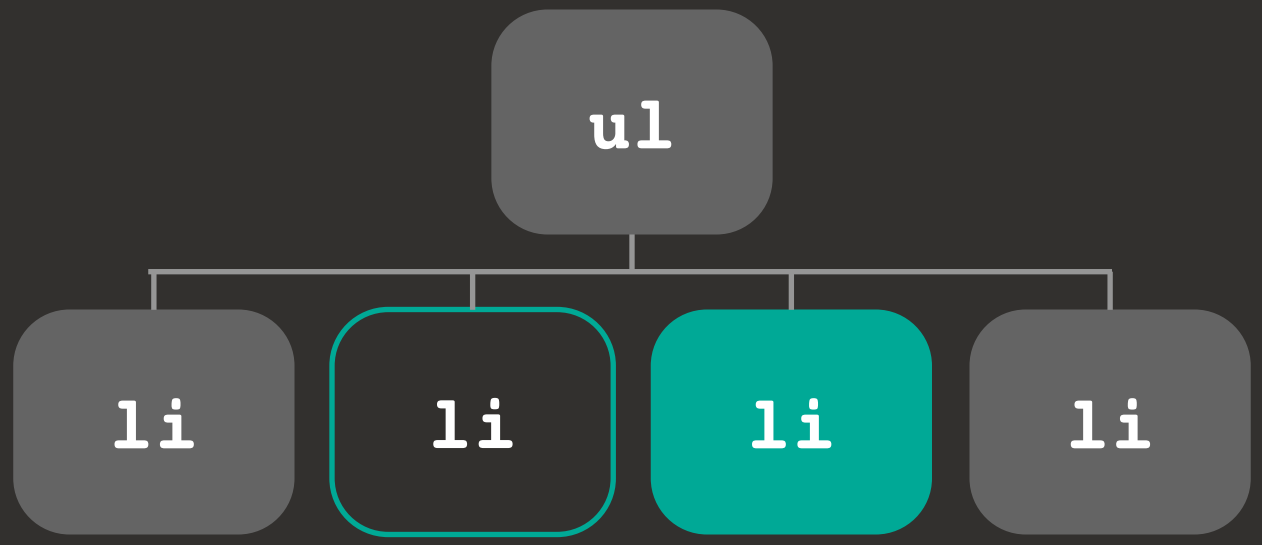




parentNode



previousSibling
or
prevsiousElementSibling

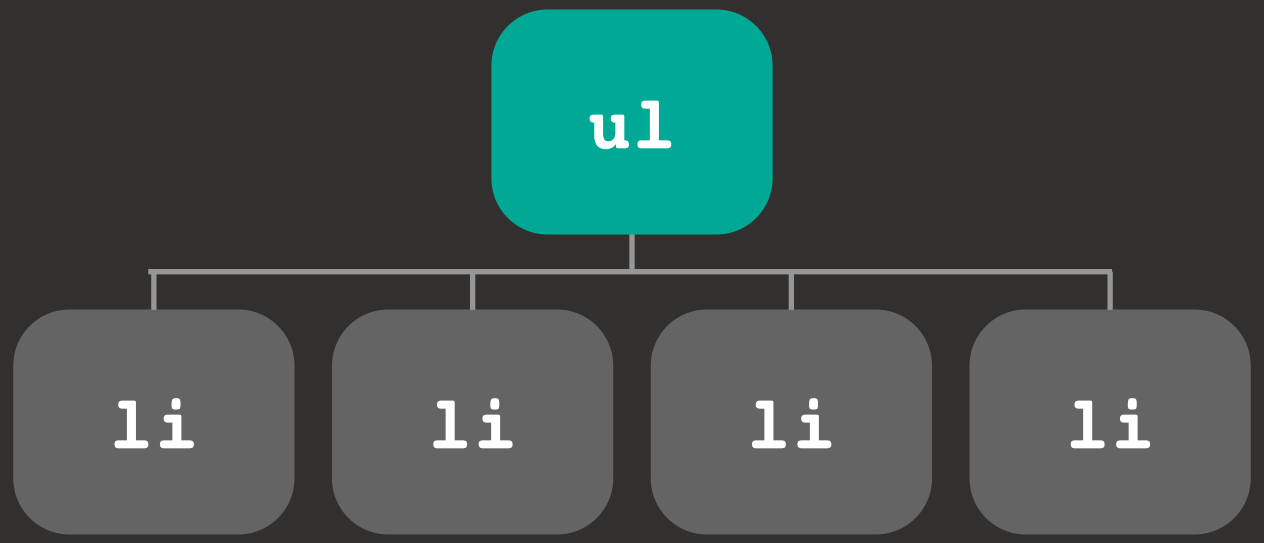


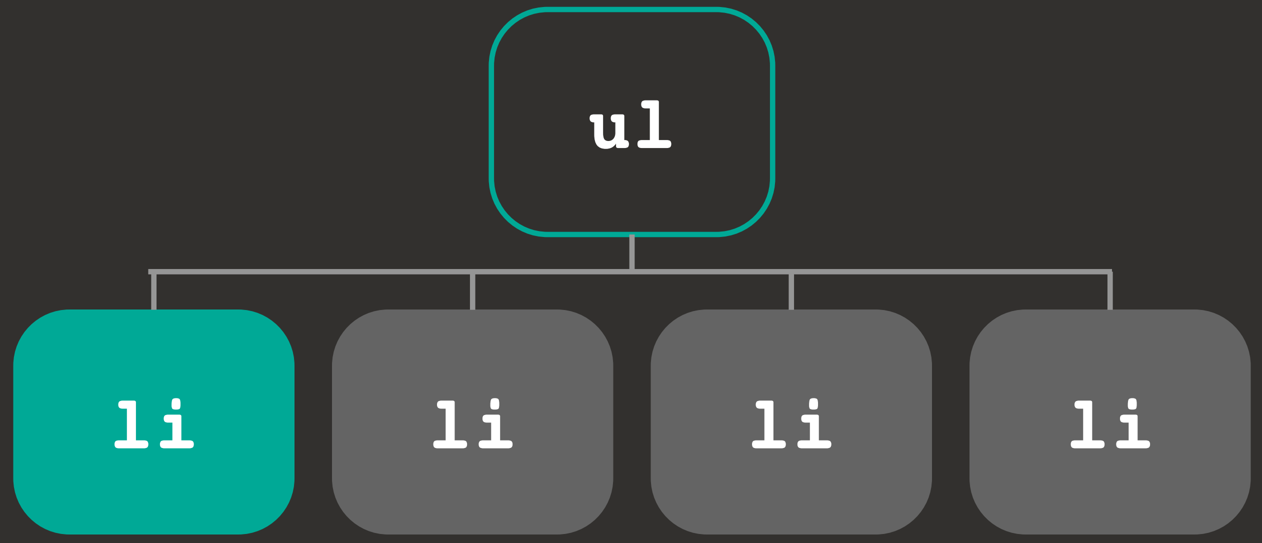
nextSibling

or

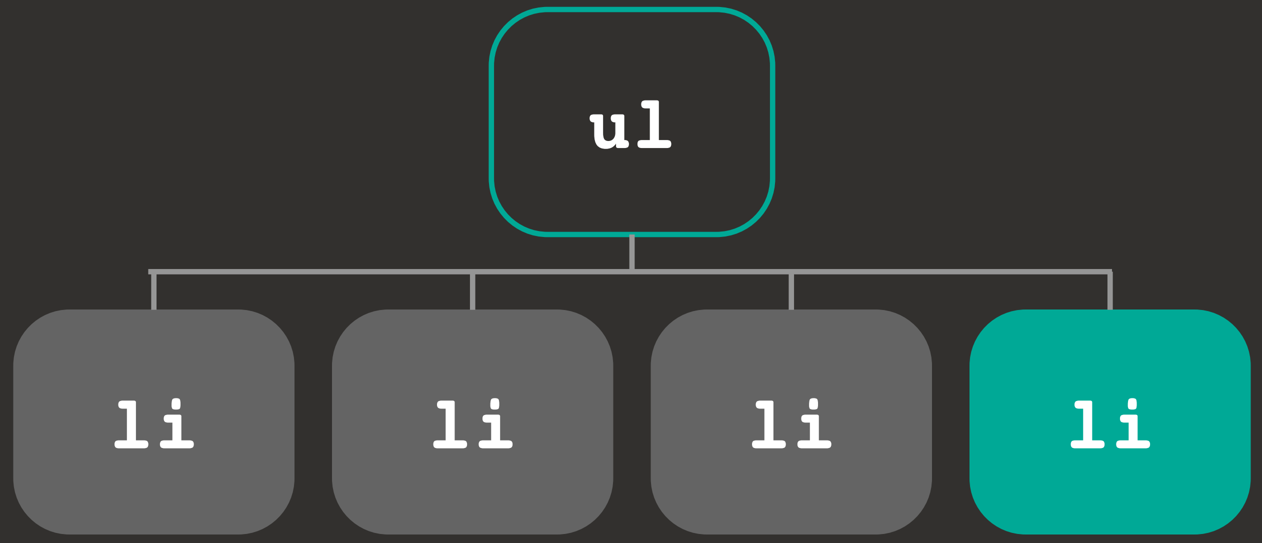
nextElementSibling

STARTING ELEMENT





firstChild
or
firstElementChild

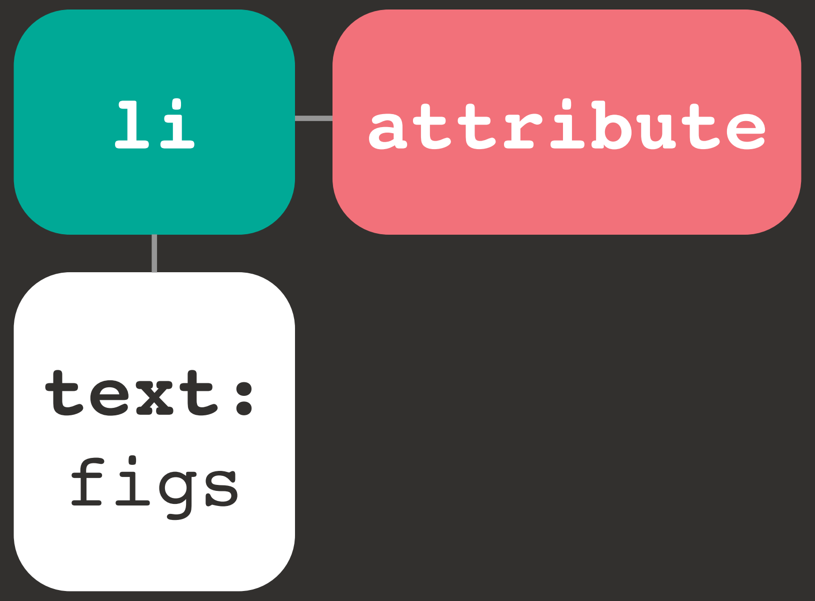


lastChild
or
lastElementChild

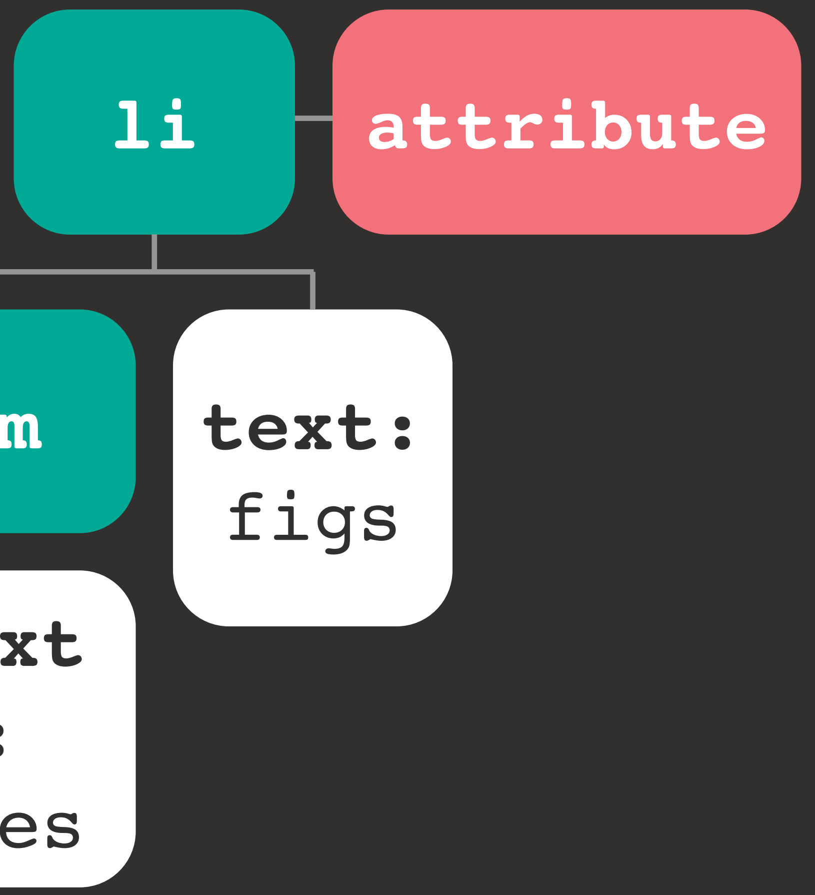
WORKING WITH ELEMENTS

Elements can contain:

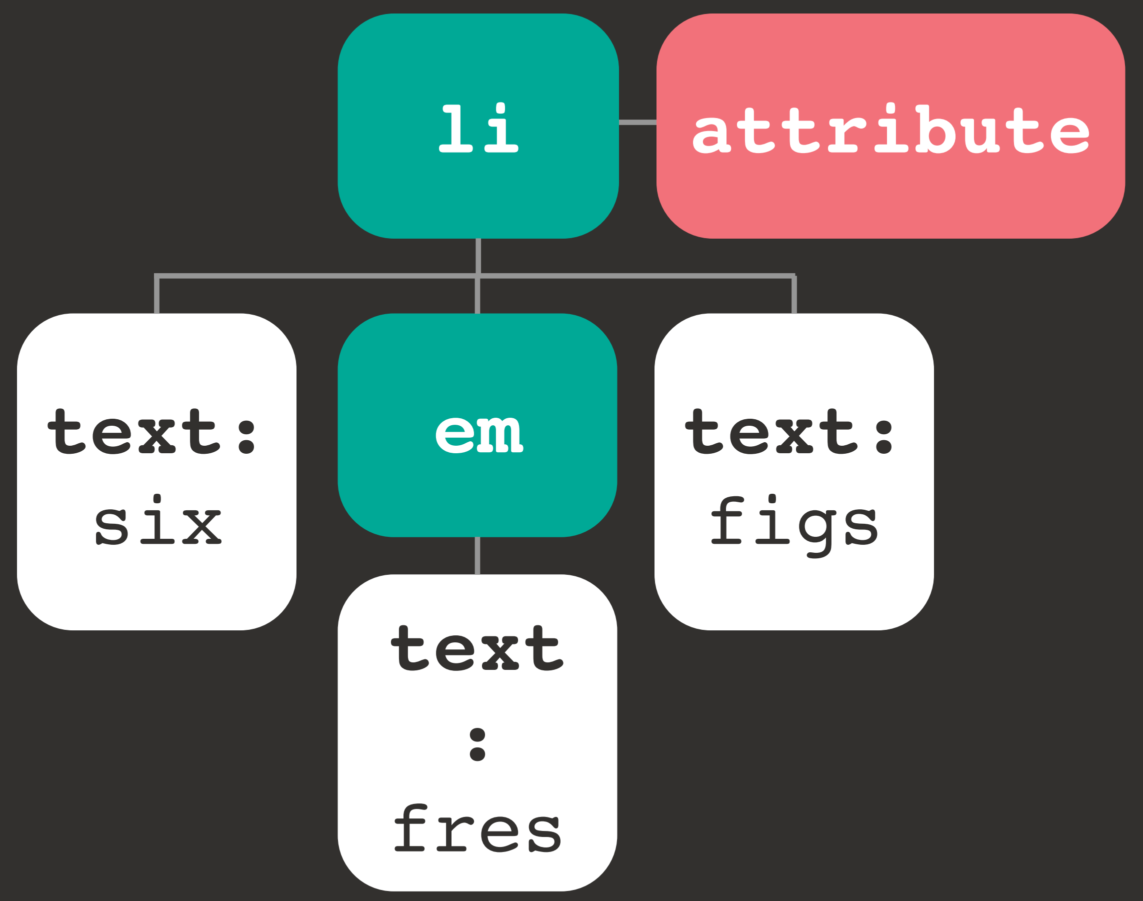
- Text nodes
- Element content
- Attributes



```
<li id="one">figs</li>
```



```
<li id="one"><em>fresh</em> figs</li>
```

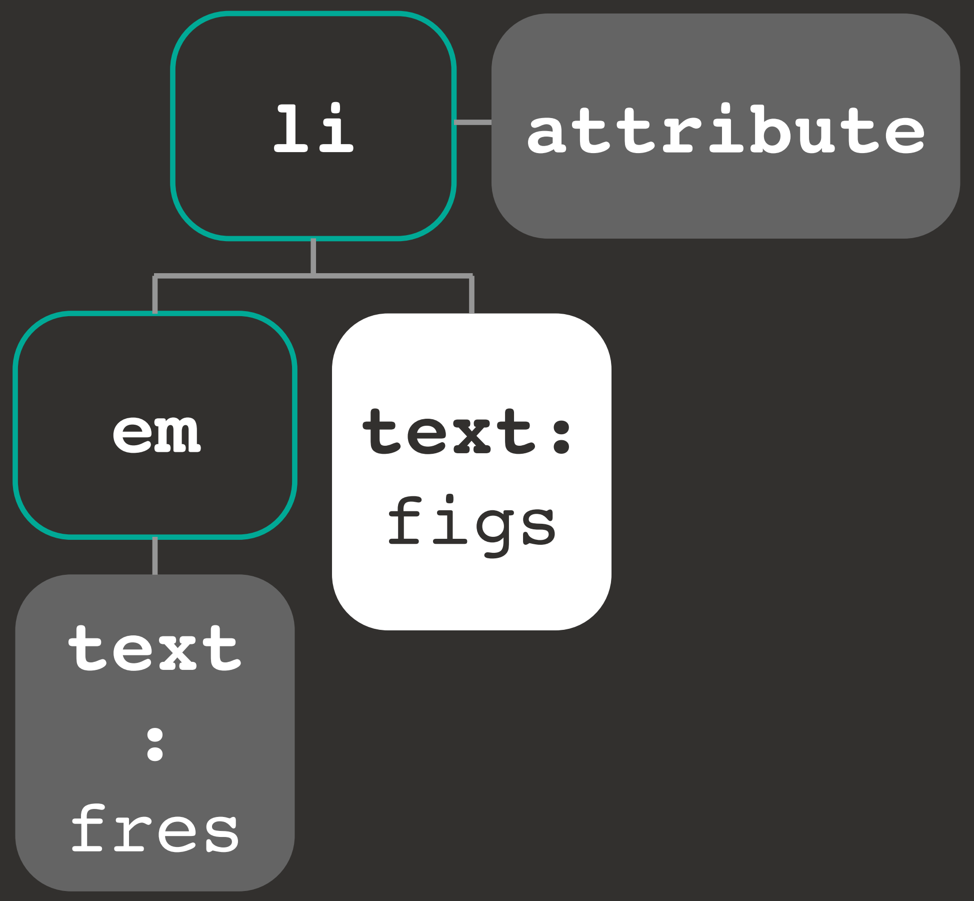


```
<li id="one">six <em>fresh</em> figs</li>
```

To access their content you can use:

- `nodeValue` on text nodes
- `textContent` for text content of elements
- `innerHTML` for text and markup

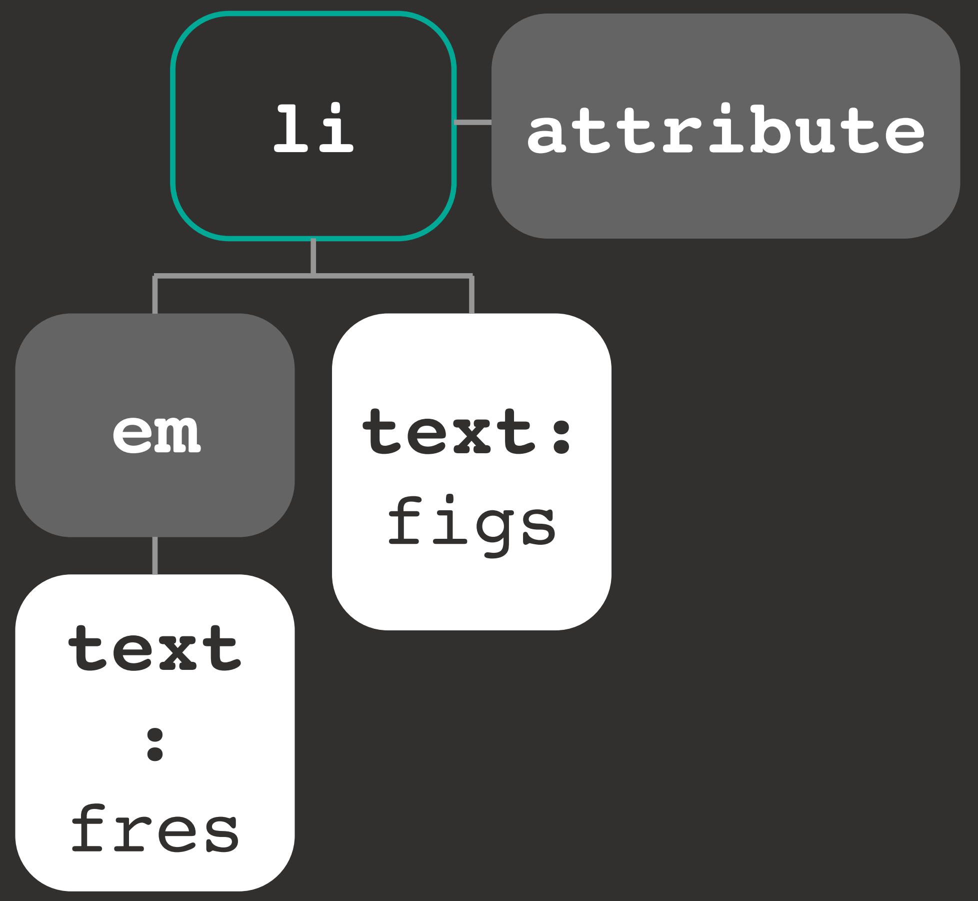
`nodeValue` works on text nodes



```
var el = document.getElementById('one');
el.firstChild.nextSibling.nodeValue;
```

returns: figs

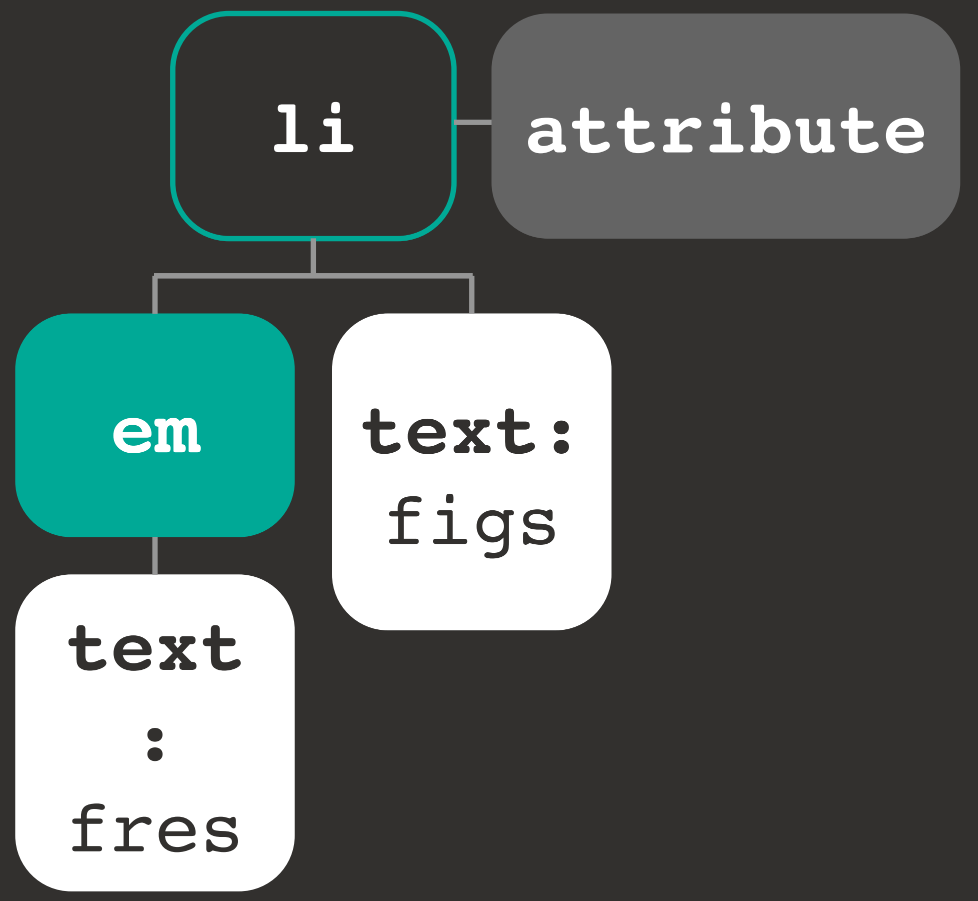
`textContent` just collects text content



```
document.getElementById(‘one’).textContent;
```

returns: fresh figs

`innerHTML` gets text and markup



```
document.getElementById(‘one’).innerHTML;
```

returns: <em>fresh</em> figs

DOM MANIPULATION VS `innerHTML`

```
createElement()
createTextNode()
appendChild()
```

- Builds up a string
- Contains markup
- Updates elements

CROSS-SITE SCRIPTING (XSS) ATTACKS

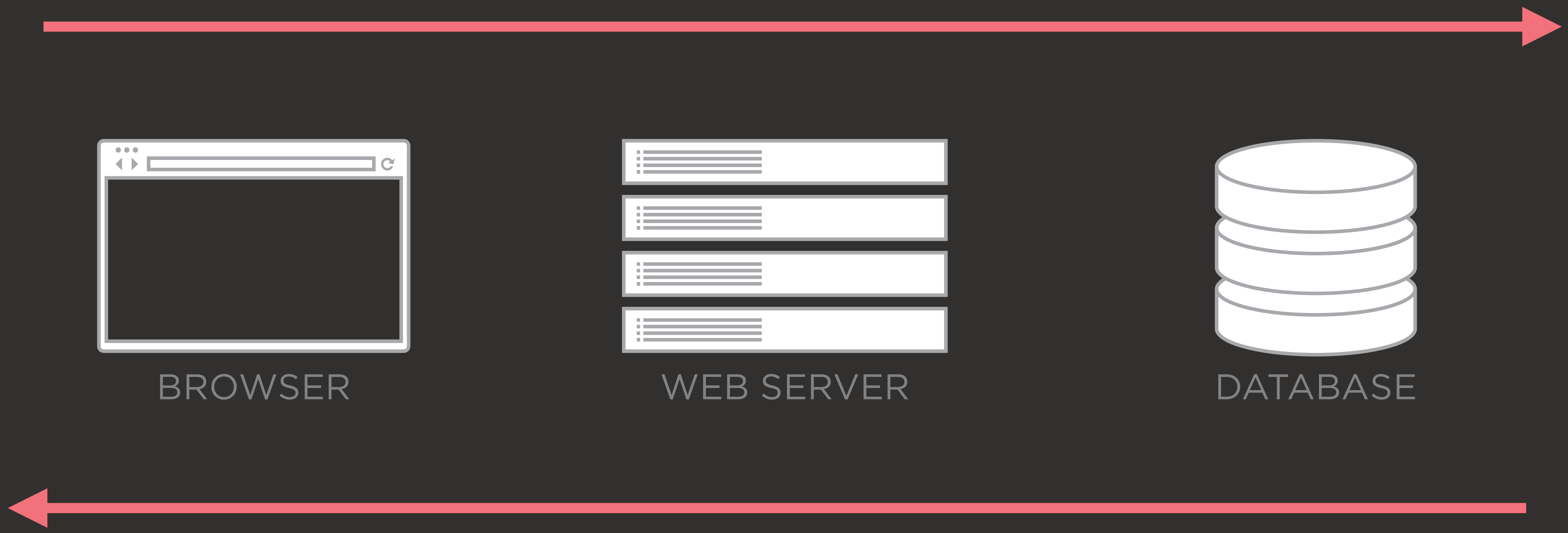
Untrusted data is content you do not have complete control over. It can contain malicious content.

Sources of untrusted data:

- User creates a profile
- Multiple contributors
- Data from third-party sites
- Files such as images / videos are uploaded

DEFENDING AGAINST XSS

Validate all input that is sent to the server



Escape data coming from the server

WORKING WITH ATTRIBUTES

ACCESSING AN ATTRIBUTE

1. Use a DOM query to select an element:

```
var el = document.getElementById(‘one’);
```

2. Method gets attribute from element:

```
el.getAttribute(‘class’);
```

UPDATING AN ATTRIBUTE

Check for attribute and update it:

```
var el = document.getElementById(‘one’);

if (el.hasAttribute(‘class’) {
  el.setAttribute(‘class’, ‘cool’);
}
```

CHAPTER 6

EVENTS

WHAT IS AN EVENT?

Events are the browser’s way of saying, “Hey, this just happened.”

When an event **fires**, your script can then react by running code (e.g. a function).

By running code when an event fires, your website responds to the user’s actions.

It becomes **interactive**.

DIFFERENT EVENT TYPES

USER INTERFACE EVENTS

```
load
unload
error
resize
scroll
```

KEYBOARD EVENTS

```
keydown
keyup
keypress
```

MOUSE EVENTS

```
click
dblclick
mousedown
mouseup
mouseover
mouseout
```

FOCUS EVENTS

```
focus / focusin
blur  / focusout
```

FORM EVENTS

```
input
change
submit
reset
cut
copy
paste
select
```

HOW EVENTS TRIGGER JAVASCRIPT CODE

1

1

Select the **element** node(s) the script should respond to

1

Select the **element** node(s) the script should respond to

2

1

Select the **element** node(s) the script should respond to

2

Indicate the **event** on the selected node(s) that will trigger a response

1

Select the **element** node(s) the script should respond to

2

Indicate the **event** on the selected node(s) that will trigger a response

3

1

Select the **element** node(s) the script should respond to

2

Indicate the **event** on the selected node(s) that will trigger a response

3

State the code you want to run when the event occurs

BINDING AN EVENT TO AN ELEMENT

There are three ways to bind an event to an element:

- HTML event handler attributes
- Traditional DOM event handlers
- DOM Level 2 event listeners

The following examples show a **blur** event on an element stored in a variable called **`el`** that triggers a function called **`checkUsername()`**.

HTML EVENT HANDLER ATTRIBUTES
(DO NOT USE)

```
<input type=“text” id=“username”
       onblur=“checkUsername()”>
```

HTML EVENT HANDLER ATTRIBUTES
(DO NOT USE)

ELEMENT

```
<input type=“text” id=“username”
       onblur=“checkUsername()”>
```

HTML EVENT HANDLER ATTRIBUTES (DO NOT USE)

```
<input type="text" id="username"
       onblur="checkUsername()">
```

$\underbrace{\texttt{onblur}}_{\text{EVENT}}$

HTML EVENT HANDLER ATTRIBUTES
(DO NOT USE)

```
<input type=“text” id=“username”
       onblur=“checkUsername()”>
```

FUNCTION

TRADITIONAL DOM EVENT HANDLERS

```
el.onblur = checkUsername();
```

TRADITIONAL DOM EVENT HANDLERS

$$\underbrace{\texttt{el}}_{\text{ELEMENT}}\texttt{.onblur = checkUsername();}$$

TRADITIONAL DOM EVENT HANDLERS

$$\texttt{el.}\underbrace{\texttt{onblur}}_{\text{EVENT}}\texttt{ = checkUsername();}$$

TRADITIONAL DOM EVENT HANDLERS

$$\texttt{el.onblur = }\underbrace{\texttt{checkUsername();}}_{\text{FUNCTION}}$$

EVENT LISTENERS

```
el.addEventListener(‘blur’, checkUsername, false);
```

EVENT LISTENERS

```
el.addEventListener(‘blur’, checkUsername, false);
```

ELEMENT

EVENT LISTENERS

$$\texttt{el.addEventListener(‘}\underbrace{\texttt{blur}}_{\textbf{EVENT}}\texttt{’, checkUsername, false);}$$

EVENT LISTENERS

$$\texttt{el.addEventListener('blur', }\underbrace{\texttt{checkUsername}}_{\text{FUNCTION}}\texttt{, false);}$$

EVENT LISTENERS

```
el.addEventListener('blur', checkUsername, false);
```

$\underbrace{\texttt{false}}_{\text{BOOLEAN (OPTIONAL)}}$

BOOLEAN (OPTIONAL)

Because you cannot have parentheses after the function names in event handlers or listeners, passing arguments requires a workaround.

PARAMETERS WITH EVENT LISTENERS

```
el.addEventListener(‘blur’, function() {
   checkUsername(5);
}, false);
```

PARAMETERS WITH EVENT LISTENERS

```
el.addEventListener(‘blur’, function() {
   checkUsername(5);
}, false);
```

An anonymous function is used as the second argument.

PARAMETERS WITH EVENT LISTENERS

```
el.addEventListener(‘blur’, function() {
    checkUsername(5);
}, false);
```

Inside the anonymous function, a named function is called.

IE5 - 8 had a different event model and did not support **`addEventListener()`** but you can provide fallback code to make event listeners work with older versions of IE.

SUPPORTING OLDER VERSIONS OF IE

```
if (el.addEventListener) {

  el.addEventListener(‘blur’, function() {
    checkUsername(5);
  }, false);

} else {

  el.attachEvent(‘onblur’, function() {
    checkUsername(5);
  });

}
```

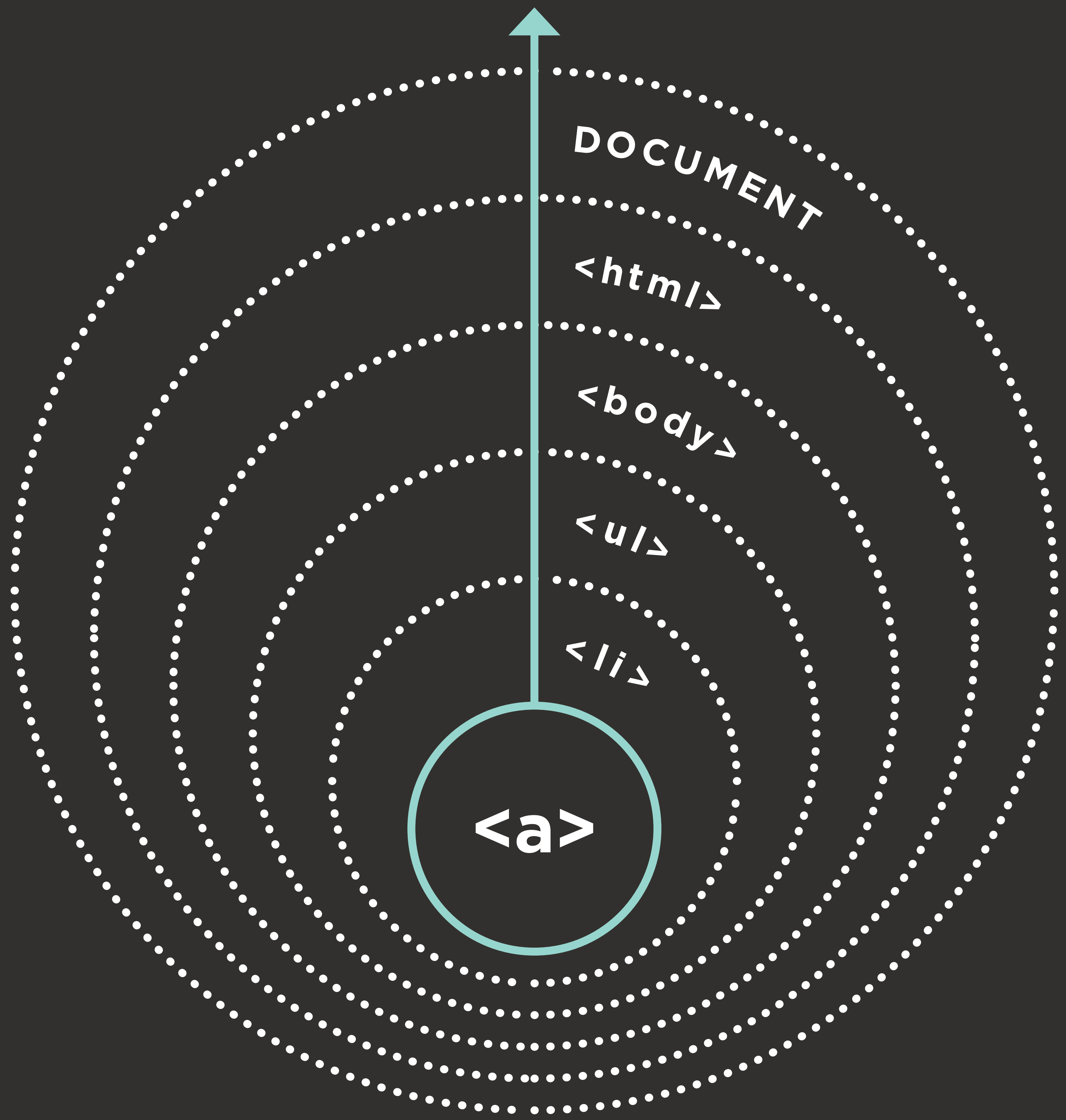
SUPPORTING OLDER VERSIONS OF IE

```
if (el.addEventListener) {

  el.addEventListener(‘blur’, function() {
    checkUsername(5);
  }, false);

} else {

  el.attachEvent(‘onblur’, function() {
    checkUsername(5);
  });

}
```

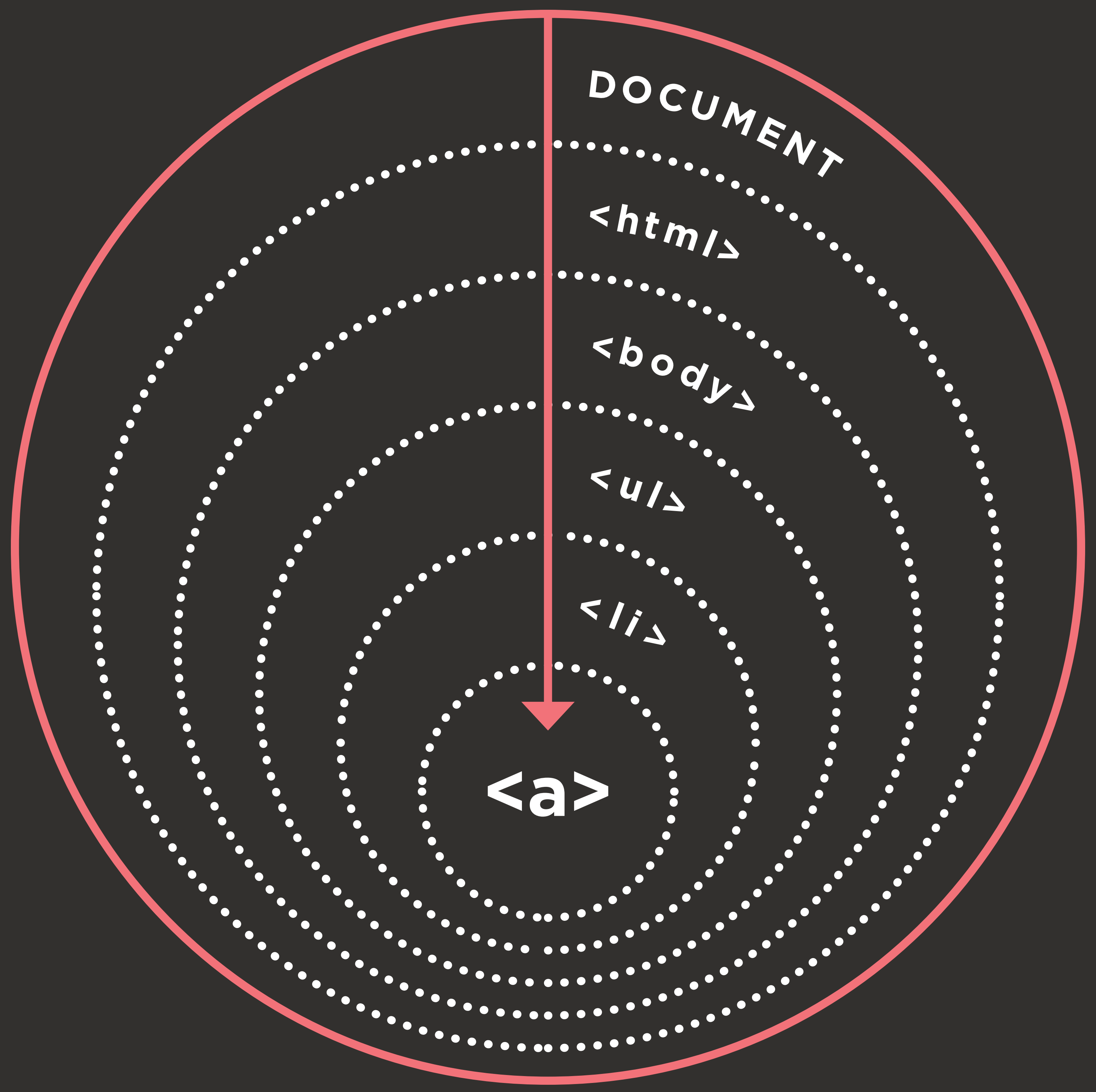
EVENT FLOW

HTML elements nest inside other elements. If you hover or click on a link, you will also be hovering or clicking on its parent elements.

EVENT BUBBLING



EVENT CAPTURING



THE EVENT OBJECT

When an event occurs,the **`event`** object can tell you information about it and which element it happened upon.

PROPERTIES

```
target
type
cancelable
```

METHODS

```
preventDefault()
stopPropagation()
```

ELEMENT AN EVENT OCCURRED ON

1: EVENT LISTENER CALLS FUNCTION

```
function checkUsername(e) {
  var target = e.target;
}

var el = document.getElementById(‘username’);
el.addEventListener(‘blur’, checkUsername, false);
```

ELEMENT AN EVENT OCCURRED ON

2: EVENT OBJECT PASSED TO FUNCTION

```
function checkUsername(e) {
  var target = e.target;
}

var el = document.getElementById(‘username’);
el.addEventListener(‘blur’, checkUsername, false);
```

ELEMENT AN EVENT OCCURRED ON

3: ELEMENT THAT EVENT HAPPENED ON

```
function checkUsername(e) {
  var target = e.target;
}

var el = document.getElementById(‘username’);
el.addEventListener(‘blur’, checkUsername, false);
```

EVENT DELEGATION

Creating event listeners for a lot of elements can slow down a page, but event flow allows you to listen for an event on a parent element.

Placing an event listener on a container element:

- Works with new elements
- Solves limitations with the `this` keyword
- Simplifies code