DOM Manipulations

Web Dev, Spring 2021

Last time

JavaScript — a dynamically typed imperative language running in the browser

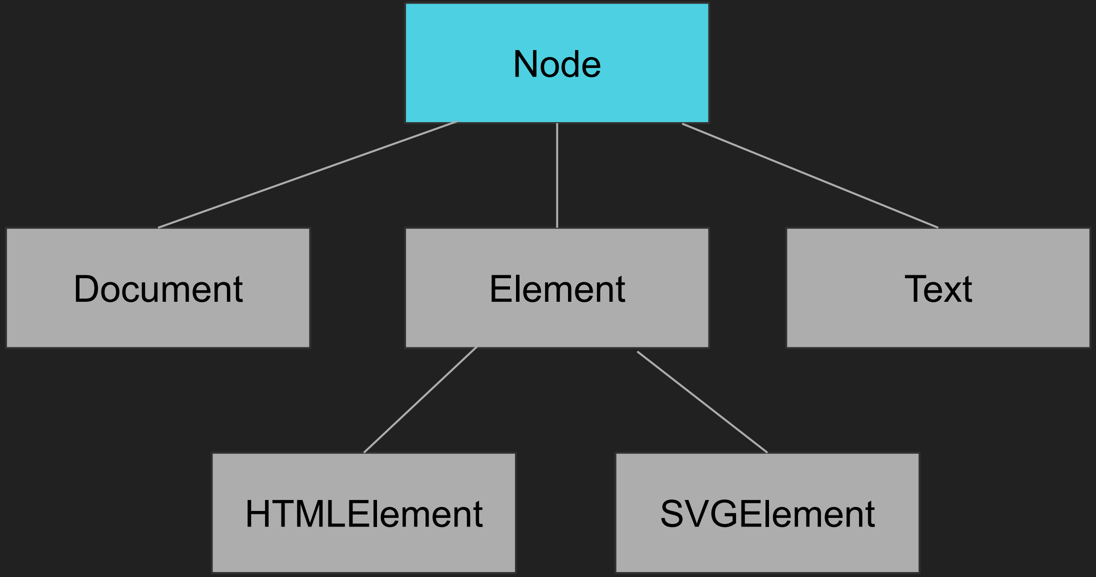
Why?

- To manipulate an HTML document displayed by the browser

Recall: the HTML document is parsed into a tree of objects

- DOM = Document Object Model
- DOM defines the objects making up the tree
- These objects have methods that can be used to manipulate the tree

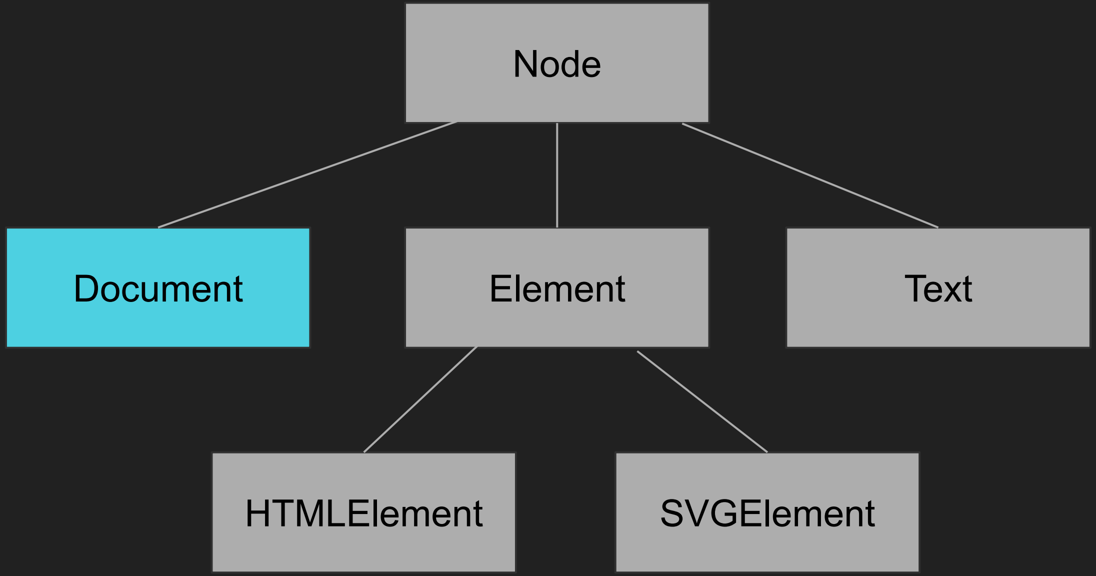
DOM class hierarchy (simplified)



Node —

Abstract base class for all DOM content

DOM class hierarchy (simplified)



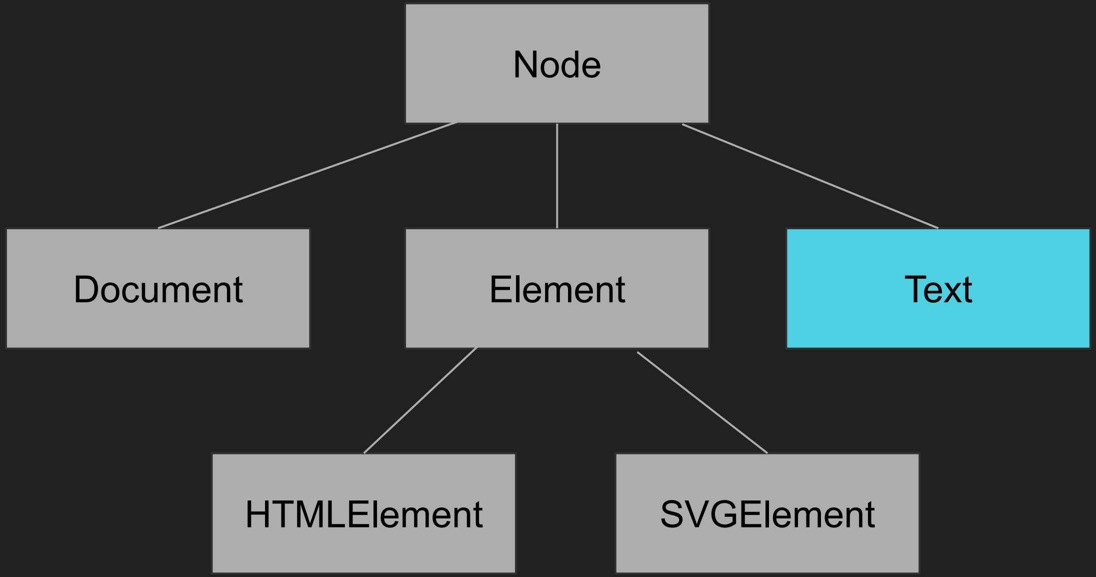
Document —

Instance at the root of DOM tree

Global var `document` holds the tree

FIelds `head` and `body`

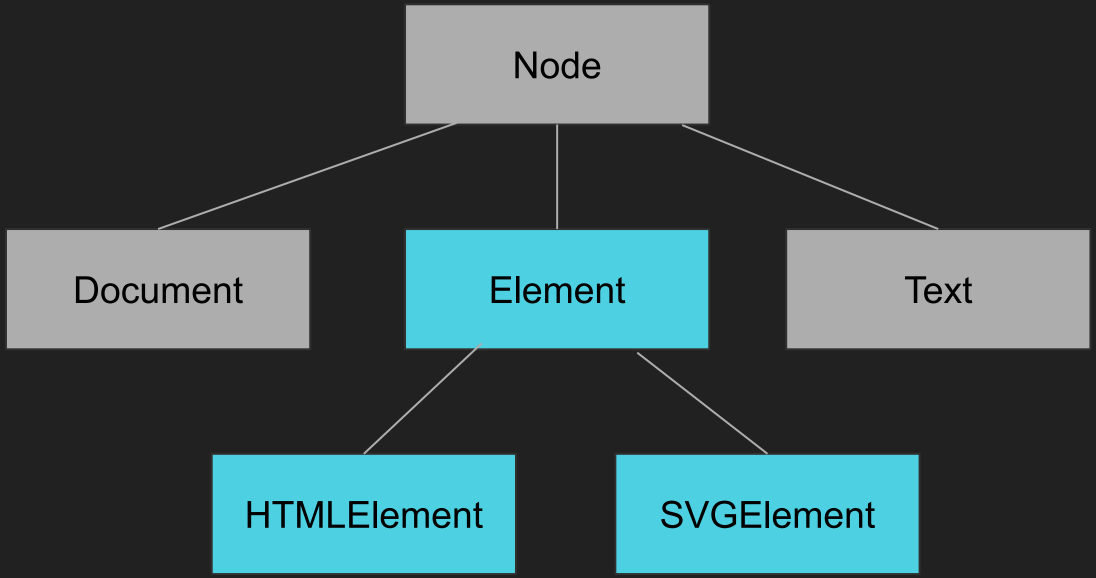
DOM class hierarchy (simplified)



Text —

DOM tree node holding text (no markup)

DOM class hierarchy (simplified)



Element —

Abstract base class for DOM tree nodes corresponding to tags

Three main classes of operations

- Find a specific element or set of elements in the DOM tree
- Modify the style of an element in the DOM tree
- Create (or delete) elements in the DOM tree

1. Find an element or set of elements

Two ways to navigate the DOM tree:

- start at the root, and follow a path down to the node(s) you're interested in
- find the node(s) you're interested in directly

Given an element `elt` — say, `document.body`

`elt.parentNode`	parent of `elt`
`elt.childNodes`	children of `elt` (array-like NodeList)
`elt.getAttribute(attr)`	attribute `attr` of `elt`

1. Find an element or set of elements

Methods in the Document class:

`document.getElementById(id)`	returns an Element
`document.querySelector(selector)`	returns an Element
`document.querySelectorAll(selector)`	returns a NodeList

Element class also implements these methods

- to search only through descendants of an element in the DOM tree

1. F

Meth

Elem

-

Basic selectors:

`*`	all elements
`name`	all elements with tag *`name`*
`#`*`name`*	the element with ID *`name`*
`.`*`name`*	all elements with class *`name`*

Complex selectors:

selector$_1$ > *selector*$_2$	all descendants of elements matching *selector*$_1$ that match *selector*$_2$

...

2. Modify style of an element

Given an HTMLElement `elt`

- `elt.style` is an object whose keys are CSS properties
- Represents the *inline style* of the element (style attribute)
- Update properties of the style object:

```
elt.style.fontSize = '16px'
```

CSS properties can have hyphens, but to make them compatible with JavaScript, properties are camelCased

`font-size` → `fontSize` `background-color` → `backgroundColor`

Demo — style

File *dom.html*

3. Create elements

Document class has creation methods

```
    const newElt = document.createElement(tag)
```

Set attribute of new element:

```
    newElt.setAttribute(attr, value)
```

Created element is not attached to the DOM tree

Add it as a child of another node `elt`

```
    elt.appendChild(newElt)
```

Demo — elements

File *dom.html*

Events

An event is the browser's way of telling your code something interesting has happened

Inversion of control

- your code gets called by browser

You hook up your code to events via event listeners

- events associated with controls: buttons, checkboxes, selectors, ...
- events associated with mouse / keyboard: click, hover, keypress, ...
- events associated with external actions: page loading, browser resizing, ...

Event listeners

An event listener is attached to a DOM element

```
    elt.addEventListener(event, fn)
```

You can add multiple event listeners to the same element

The function `fn` (event handler) is called when the event is triggered

- gets passed an event value giving details about the event
  `evt.target` → the DOM element that triggered the event
- useful if the same function handles events from different sources

Controls

Classic user interface elements:

```
<button>Click me!</button>

<select>
  <option>First option</option>
  <option>Second option</option>
  <option>Third option</option>
</select>

<input type="text">

<input type="checkbox">
```

…

Controls

Classic user interface elements:

```
<button>Click me!</button>

<select>
  <option>First option</option>
  <option>Second option</option>
  <option>Third option</option>
</select>

<input type="text">

<input type="checkbox">

…
```

Interesting events:

click — triggered when button is clicked

Controls

Classic user interface elements:

```
<button>Click me!</button>

<select>
  <option>First option</option>
  <option>Second option</option>
  <option>Third option</option>
</select>

<input type="text">

<input type="checkbox">
…
```

Interesting events:

change — triggered when selection changes

The select element *value* property holds the selected option's value (its text by default)

Can override option value with a value attribute

Controls

Classic user interface elements:

```
<button>Click me!</button>

<select>
  <option>First option</option>
  <option>Second option</option>
  <option>Third option</option>
</select>

<input type="text">

<input type="checkbox">

…
```

Interesting events:

input — triggered when text changes
change — triggered when text change is committed

The input element *value* property holds the input text

Controls

Classic user interface elements:

```
<button>Click me!</button>

<select>
  <option>First option</option>
  <option>Second option</option>
  <option>Third option</option>
</select>

<input type="text">

<input type="checkbox">

…
```

Interesting events:

change — triggered when checkbox is checked or unchecked

The input element *checked* property is true exactly when the checkbox is checked

Forms

Historical artifact — but still used nowadays

- collection of input fields meant to get information and "submit" it to a website
- first real feature outside of "show hypertext" initial web functionality
- often used with PHP, which makes it particularly easy to deal with forms data
- we'll come back to forms — they don't make sense without a backend

Controls and input fields are more general than forms though, and we'll mostly use them independently of forms

Demo — controls

File *events.html*

- Picture selector
- Add new picture

Mouse / keyboard events

Every element in the DOM tree can listen to:

click	— triggered when user clicks on the element's box
mouseover	— triggered when mouse pointer enters the element's box
mouseout	— triggered when mouse pointer leaves the element's box
…	

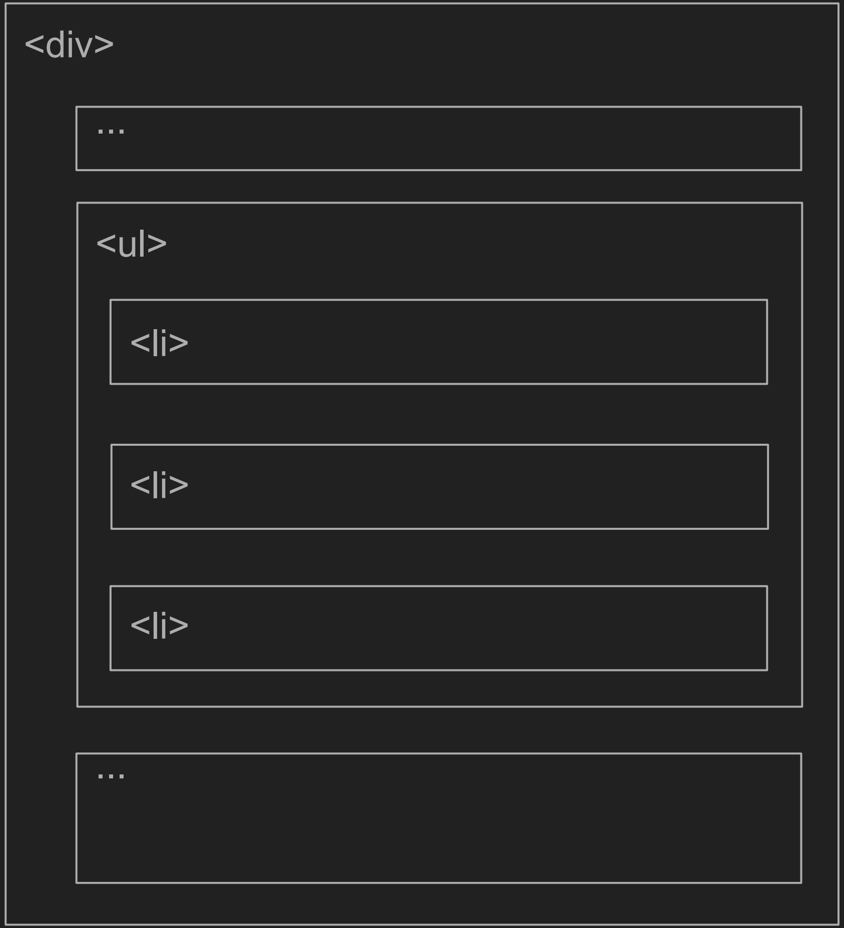
A point of the browser screen may be in multiple elements' boxes

- which elements see the event?   all of them!

Event bubbling (simplified)

Mostly applies to mouse events

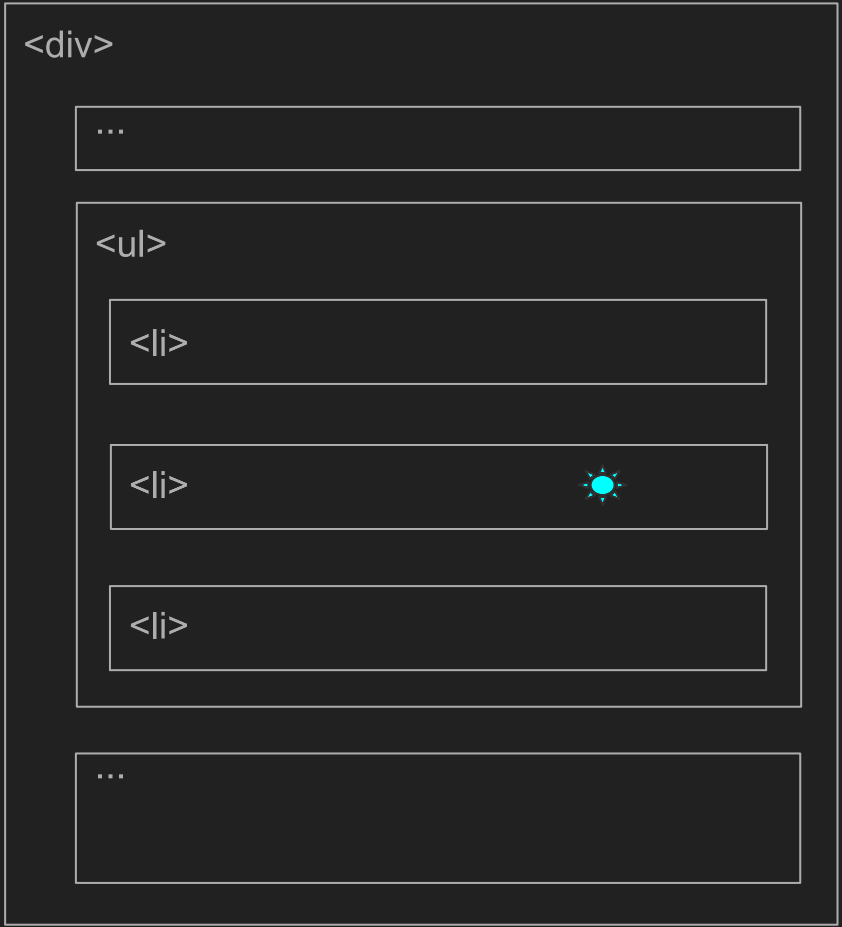
- browser finds the deepest element to which the event applies
- triggers any listener for that event on the element
- browser goes to the parent of the element, and triggers any listener for that event on that element ("bubbling up")
- keep going until at the root



Event bubbling (simplified)

Mostly applies to mouse events

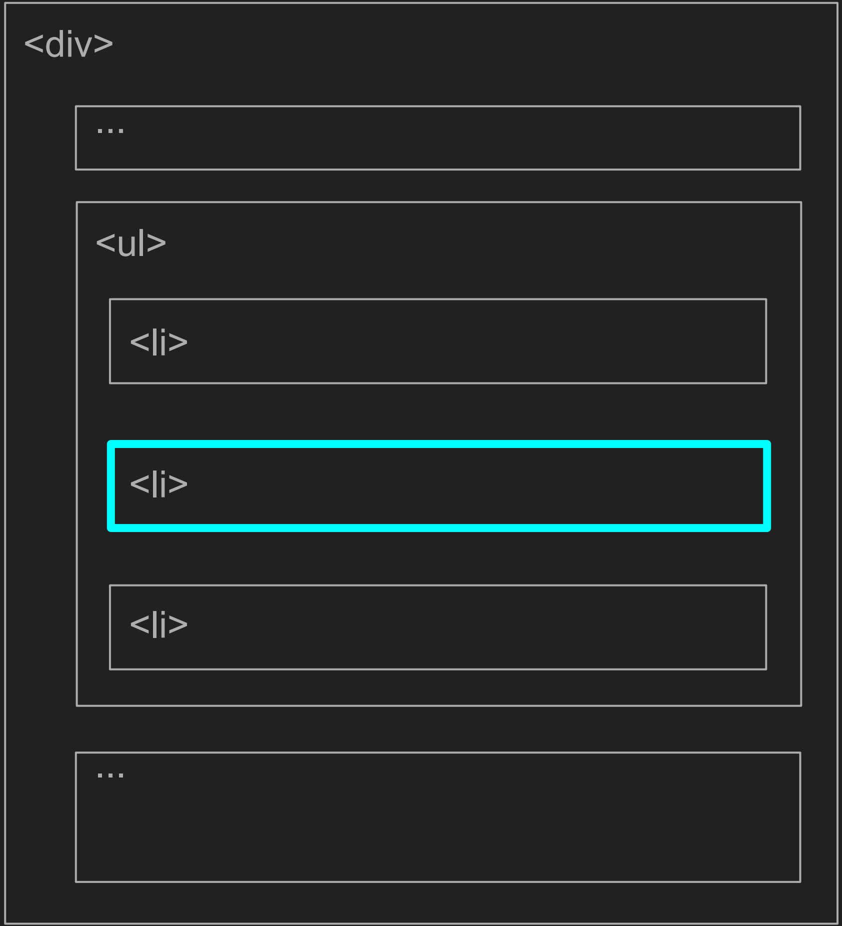
- browser finds the deepest element to which the event applies
- triggers any listener for that event on the element
- browser goes to the parent of the element, and triggers any listener for that event on that element ("bubbling up")
- keep going until at the root



Event bubbling (simplified)

Mostly applies to mouse events

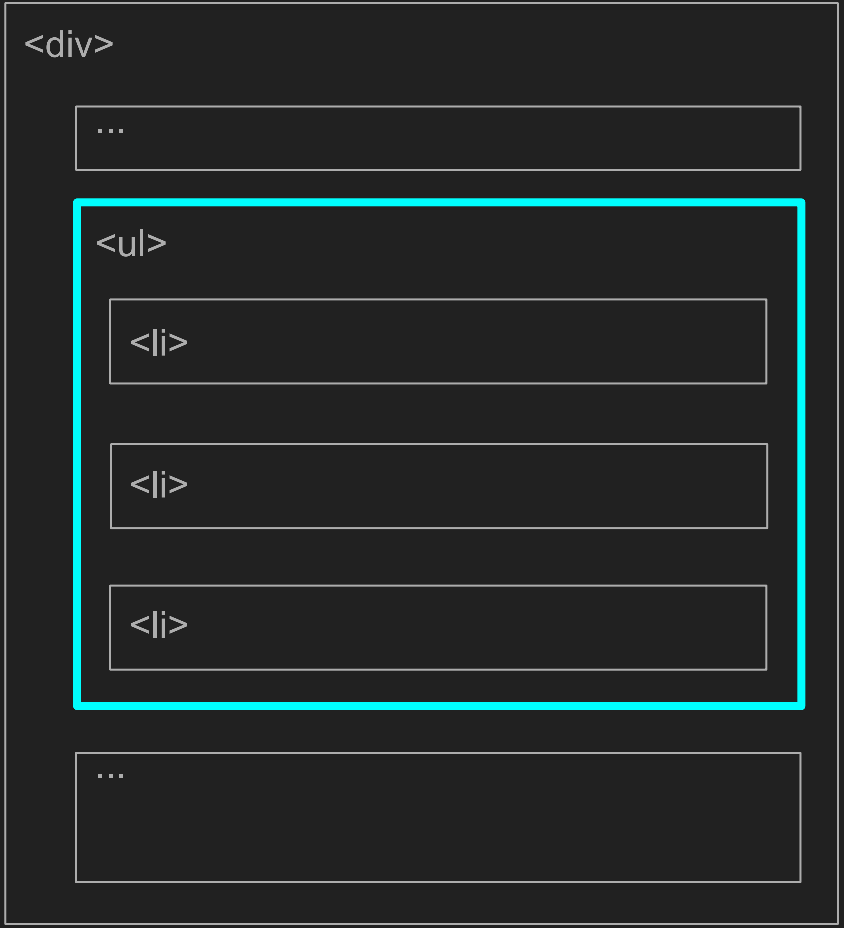
- browser finds the deepest element to which the event applies
- triggers any listener for that event on the element
- browser goes to the parent of the element, and triggers any listener for that event on that element ("bubbling up")
- keep going until at the root



Event bubbling (simplified)

Mostly applies to mouse events

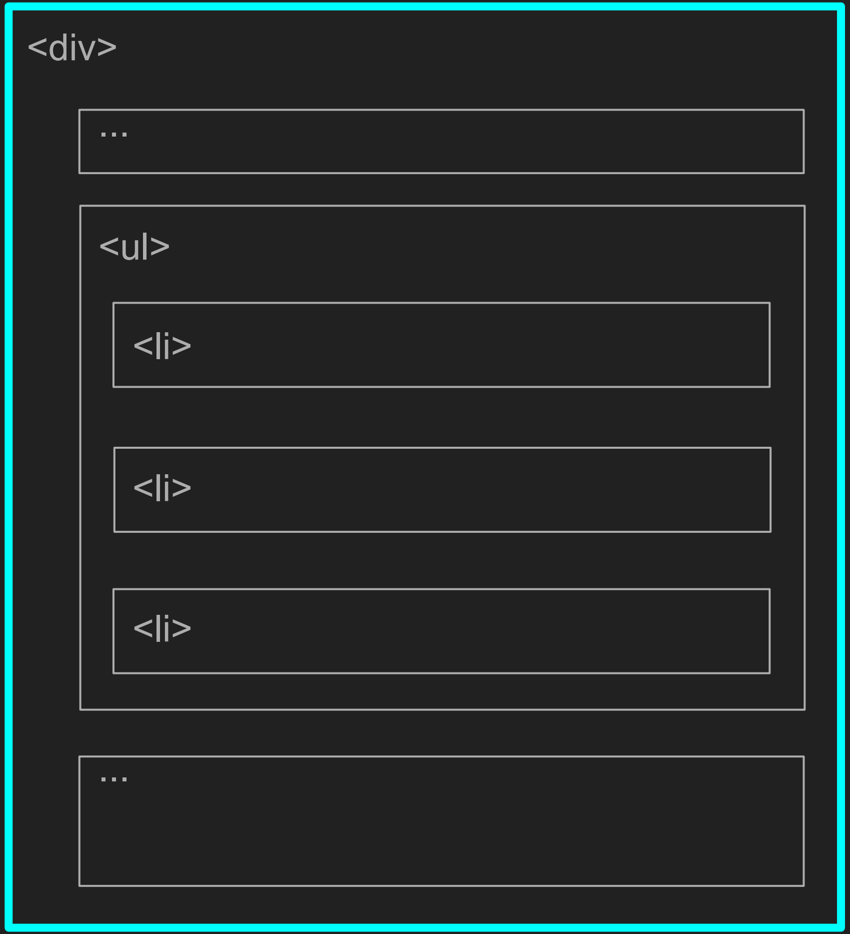
- browser finds the deepest element to which the event applies
- triggers any listener for that event on the element
- browser goes to the parent of the element, and triggers any listener for that event on that element ("bubbling up")
- keep going until at the root



Event bubbling (simplified)

Mostly applies to mouse events

- browser finds the deepest element to which the event applies
- triggers any listener for that event on the element
- browser goes to the parent of the element, and triggers any listener for that event on that element ("bubbling up")
- keep going until at the root



Demo - mouse events

File *events.html*

- show date added when hovering over picture

Next time

How can we structure code to tame inversion of control?

- MVC — Model-View-Controller architecture