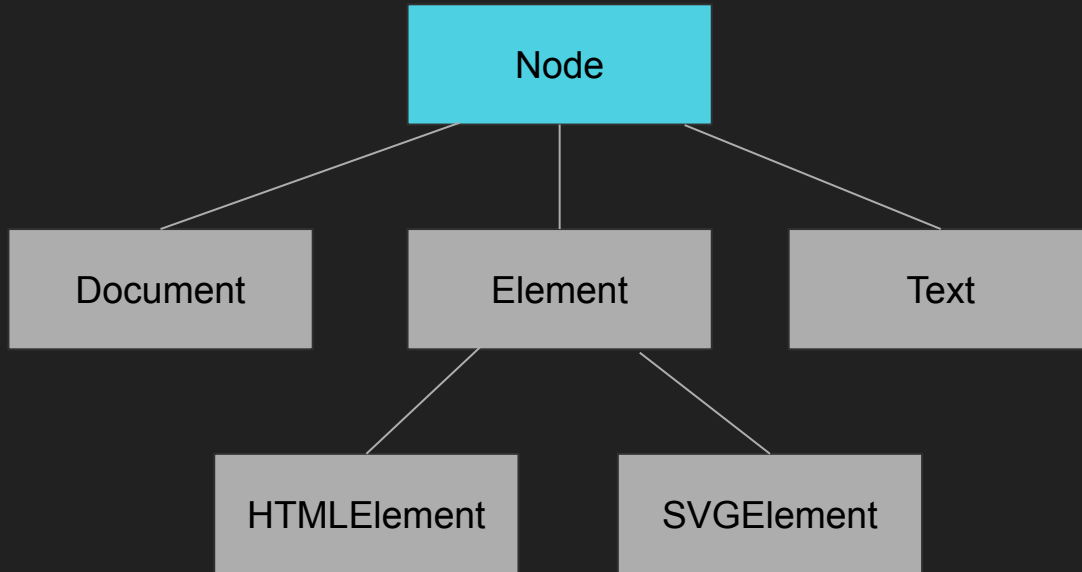# The Document Object Model

Spring 2024
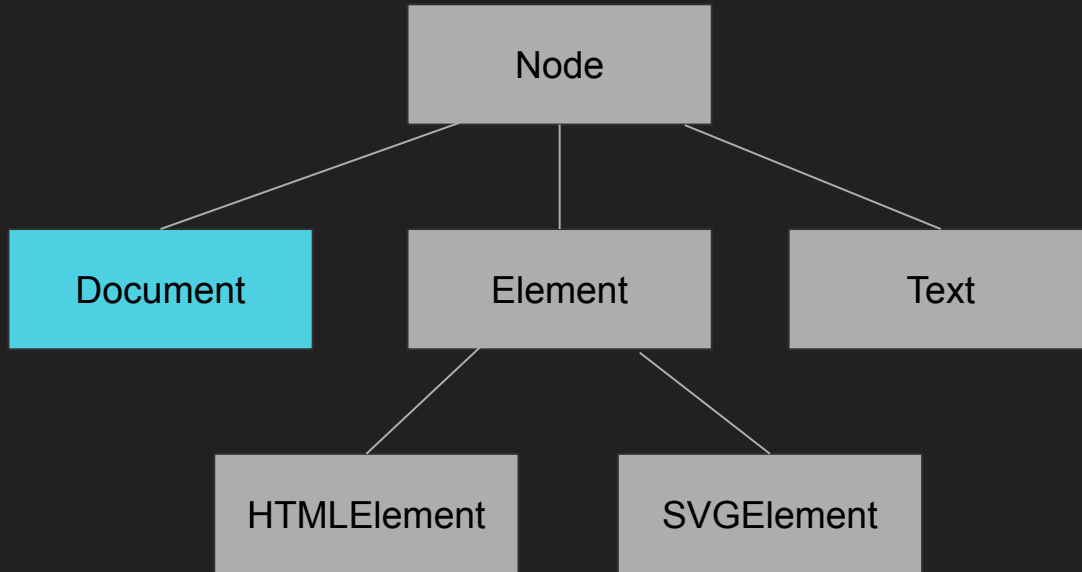
DOM class hierarchy (simplified)

Node —

Abstract base class for all DOM content

DOM class hierarchy (simplified)
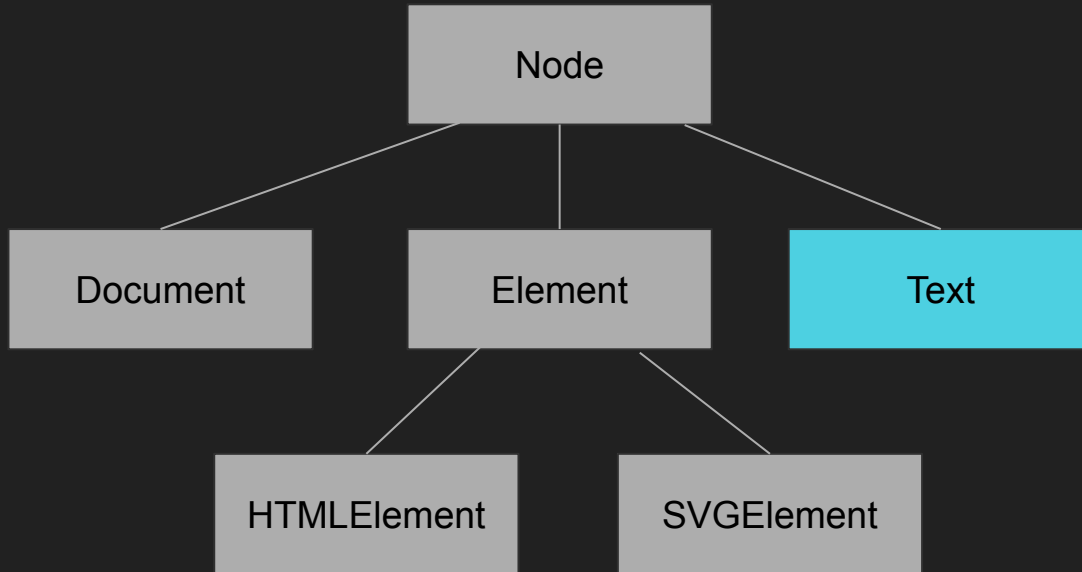
```
                    ┌──────────────┐
                    │     Node     │
                    └──────────────┘
            ┌──────────────┼──────────────┐
    ┌──────────────┐ ┌──────────────┐ ┌──────────────┐
    │   Document   │ │   Element    │ │     Text     │
    └──────────────┘ └──────────────┘ └──────────────┘
                   ┌──────────┴──────────┐
           ┌──────────────┐      ┌──────────────┐
           │  HTMLElement │      │  SVGElement  │
           └──────────────┘      └──────────────┘
```

Document —

Instance at the root of DOM tree

Global var `document` holds the tree

Fields `head` and `body`

DOM class hierarchy (simplified)

Node

Document

Element

Text

HTMLElement

SVGElement

Text —

DOM tree node holding text (no markup)

DOM class hierarchy (simplified)

Node

Document

Element

Text

HTMLElement

SVGElement
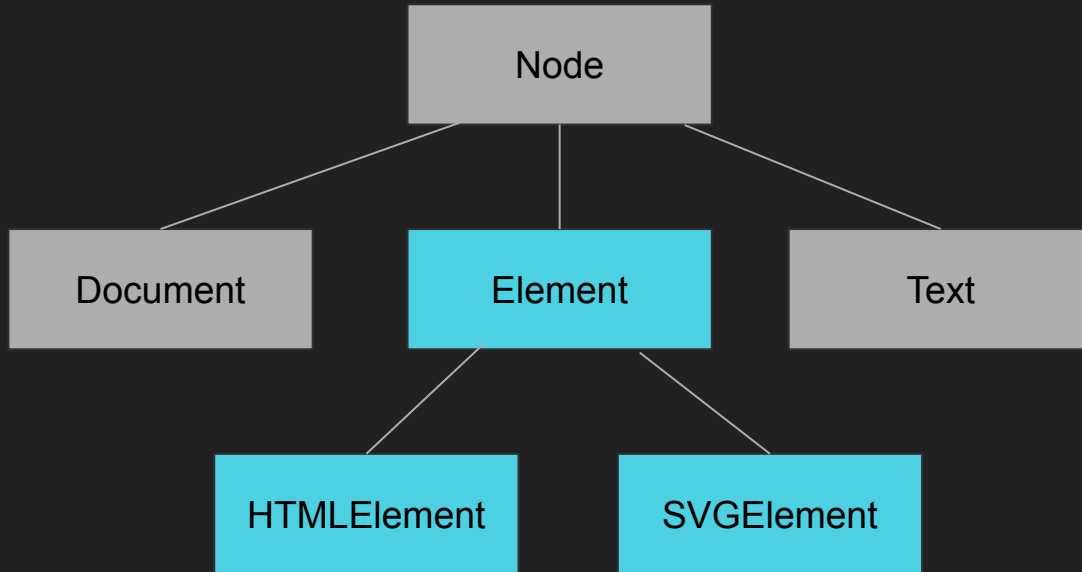
Element —

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)
document.querySelector(selector)
document.querySelectorAll(selector)
```

returns an Element

returns an Element

returns a NodeList

Element class also implements these methods

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

1. FI

Meth

Elen

-

Basic selectors:

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

Complex selectors:

selector₁ > *selector₂*   all descendants of elements matching
                selector₁ that match *selector₂*

...

2. Modify an element

Given an HTMLElement *elt*

Can set/update attribute of element:

    elt.setAttribute(*attr*, *value*)

Can set/update style of element:

- *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'

2. Modify an el...

Given an HTMLElen...

Can set/update attri...

    elt.setAttri...

CSS properties in the `style` object use camelCase:

      `font-size` → `fontSize`

      `background-color` → `backgroundColor`

Can set/update style of element:

- *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'`

2. Modify an element

Can set/update the classes of element

- *elt*.`classList` is an object that behaves like an array of class names
- but also has methods such as `contains(), add(), remove()`

Adding and removing classes from an element is preferred to directly changing the style

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

- createCell
- createRandomCell
- clearCells
- *elt*.classList

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 handlers

An event handler is attached to a DOM element

  `elt.onevent = fn`

  `elt.addEventListener(event, fn)`

`addEventListener()` lets you add multiple event handlers to an element

The function `fn` 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

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!

DEMO

- onmouseover
- onmouseout
- onclick

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

Event bubbling (simplified)
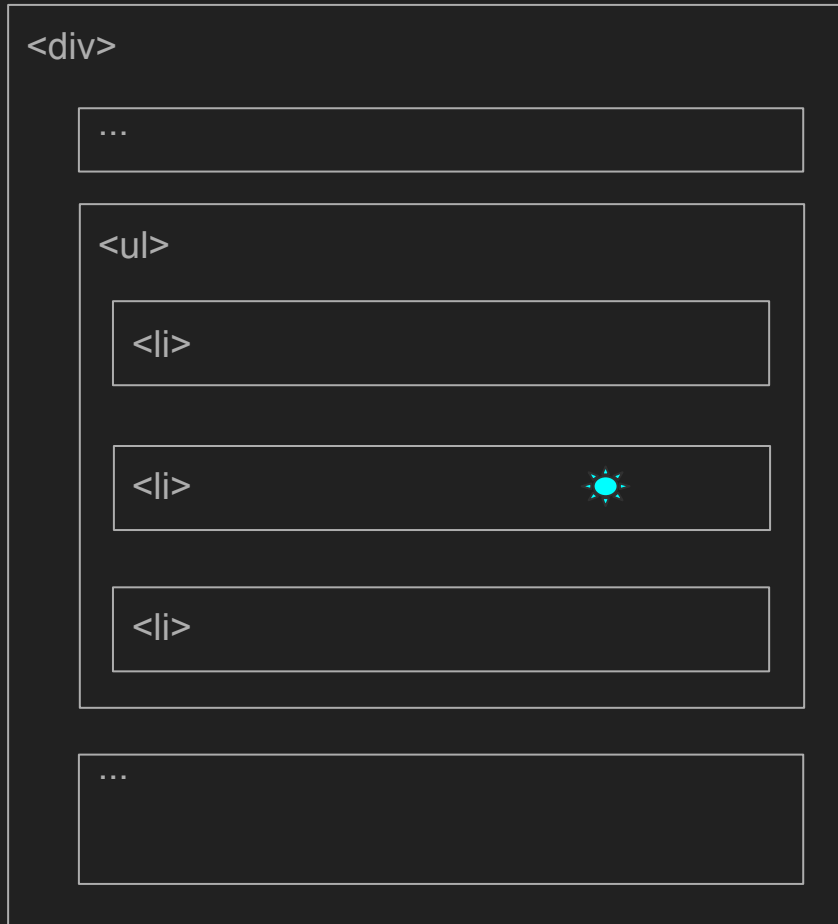
Mostly applies to mouse events

- 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

```
<div>

  ...

  <ul>

    <li>

    <li>

    <li>

  ...

```

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

```
<div>
    ...

    <ul>
        <li>

        <li>                    ☀

        <li>

    ...
```

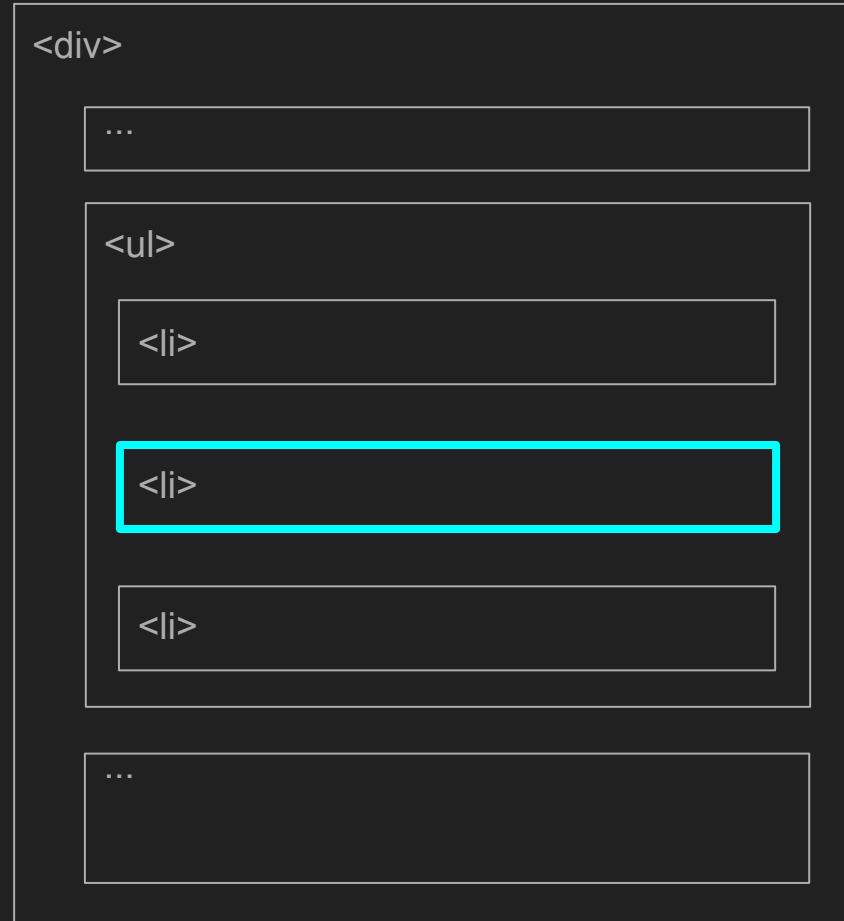# Event bubbling (simplified)

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

```
<div>
    ...

    <ul>
        <li>

        <li>

        <li>
    ...
```

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

```
<div>

    ...

    <ul>

        <li>

        <li>

        <li>

    ...

</div>
```

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

```
<div>
    ...

    <ul>
        <li>

        <li>

        <li>

    ...
</div>
```

DEMO

- window.onload for initialization
- create random button
- clear all button
- color selector
- create