Javascript

in the Browser

Thierry Sans

Example

```
<html>
<body>
<script type="text/javascript">
        document.body.innerHTML = "<h1>This is a heading</h1>";
</script>
</body>
</html>
```

Javascript: Inline, embedded or separate file?

Inline

```
<button onclick="console.log("Hello World!);">Click me</button>
```

Embedded

```
<script type="text/javascript">
    console.log("Hello World!);
</script>
```

Separate file

```
<script src="js/script.js"></script>
```

Warning: this is not the proper way to load a Javascript module (more later)

Javascript in the browser is restrictive

✓ You can access elements of the webpage and the browser

✓ You can track user actions on the webpage (events)

✓ You can create threads (web workers)

✓ You can open sockets (web sockets)

✓ …

◉ You cannot access the file system (only via the upload form)

◉ You cannot access to other programs

◉ You cannot access to other tabs in the browser

◉ …

The Browser

Pop-up Boxes

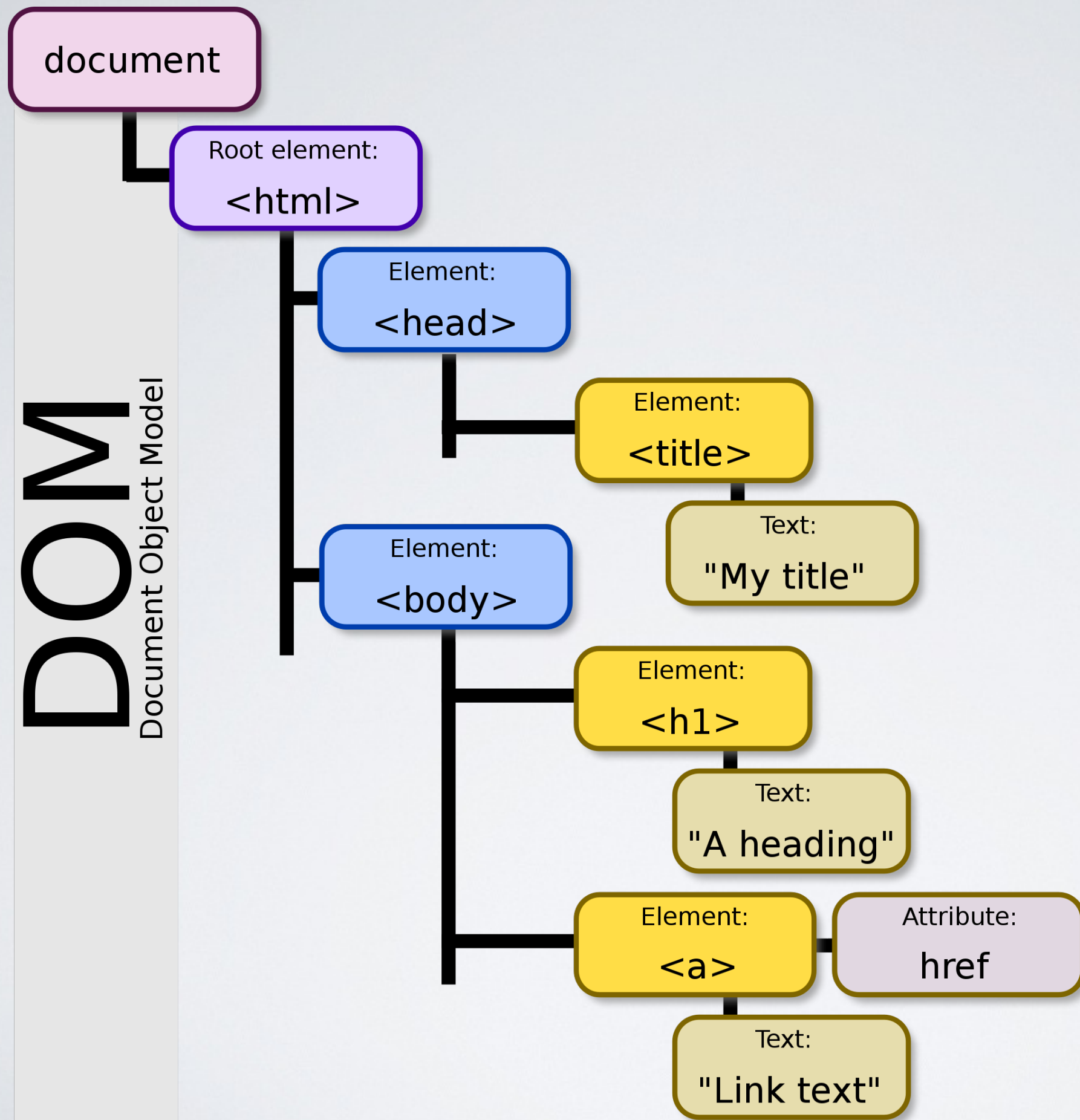|	
`alert("hello world!")`	dialog box with "ok" button
`confirm("are you sure?")`	dialog box with "ok" and "cancel" buttons
`prompt("Name?","John")`	input box with prompt text and default value

The Browser

screen	the visitor's screen
browser	the browser itself
window	the current browser window
url	the current url
history	Back and forward URLs

Document Object Model

DOM
Document Object Model

document

Root element:
<html>

Element:
<head>

Element:
<body>

Element:
<title>

Text:
"My title"

Element:
<h1>

Text:
"A heading"

Element:
<a>

Attribute:
href

Text:
"Link text"

image source: *wikipedia*

Node accessors

The root node

```
document
```

Accessors

```
document.getElementById("id")
document.getElementsByTagName("p");
document.getElementsByClassName("class");
document.querySelector("#id .class p");
document.querySelectorAll("#id .class p");
```

DOM methods

`x.innerHTML`	the content of x
`x.attributes`	the attributes nodes of x
`x.style`	css of x
`x.parentNode`	the parent node of x
`x.children`	the child nodes of x
`x.appendChild`	insert a child node to x
`x.removeChild`	remove a child node from x
...	...

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Events

DOM events and handlers

load	when e is fully loaded
click	when e is clicked
submit	when e is submitted
hover	when the mouse is on top e
keydown	when a key is pressed while e is in focus
. . .	. . .

https://developer.mozilla.org/en-US/docs/Web/Events

Different ways to handle events

```
// Overwrite the existing handler
e.onclick = function(e){

    ...

};


// Add another event handler
e.addEventListener('click', function(e){

    ...

});
```

User-defined events and listeners

```
// Listen for the event
document.addEventListener('myEvent', function(e){
    ...
});


// Dispatch the event
document.dispatchEvent(new Event('myEvent'));
```

Custom events

```javascript
// Listen for the custom event
document.addEventListener('onSomething', function(e){
    console.log(e.detail);
});


// Dispatch the custom event
document.dispatchEvent(new CustomEvent('onSomething',
{ e.detail: 'Hello World!'}));
```

Building Good Frontends

Recipes to become a good frontend developer

- ~~Write good Javascript code (strict mode)~~
- ~~Load Javascript code correctly and efficiently~~
- ~~Encapsulate Javascript in closures~~

> Old way

- Use Javascript Modules (since `es6`)

> New way

- Create a Frontend API (good practice)

The problem with Javascript interpreters

✓ **Good Javascript** is interpreted by browsers
   in a <u>consistent way</u>

◎ **Bad javascript** code is loosely interpreted by browsers
   in an <u>inconsistent way</u>

[old way] Using **strict mode**

➡ Force the browser to validate Javascript against the standard

✓ Dynamically raises errors (or warnings) in the console when the code is not compliant with the standard

```
"use strict";
let doSomething = function() {
    // this runs in strict mode
}
```

[old way] Problem with scoping

➡ In the browser, all Javascript files share the same execution environment i.e they share the same scope

 ◉ variable (and function) naming conflicts

 ◉ strict mode applied to all

[old way] Scoping problem with variable names

```
------------------------------------------------------
                                            file1.js
let doSomething = function() {
    // first declaration of doSomething
}


------------------------------------------------------
                                            file2.js
let doSomething = function() {
    // conflicting doSomething from file 1
}
```

[old way] Scoping problem with strict mode

```
---------------------------------------
                                         file1.js
"use strict";
let doSomething = function() {
    // strict mode applies

}

---------------------------------------
let doSomethingElse = function() {      file2.js
    // strict mode applies too

}
```

[old way] Encapsulate Javascript in **a closure**

```javascript
(function() {
    "use strict";

    let private = function() {
        // private is not available from outside

    }
}());
```

[old way] encapsulate and export the **namespace**

```javascript
let $ = (function() {
    "use strict";

    let module = {};

     let private = function(){
        // private is not available from outside
     }


    module.public = function(){
        // public is available from outside
    }

    return module;
}());
```

[old way]
Loading Javascript

Legend

- HTML parsing
- HTML parsing paused
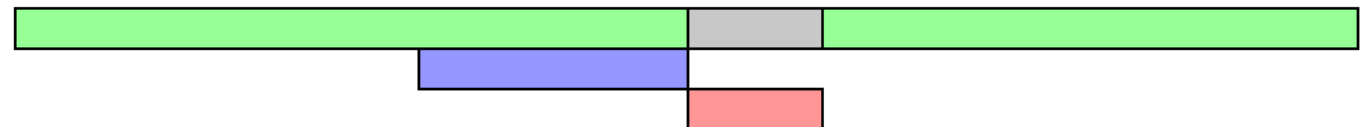- Script download
- Script execution

<script>

Let's start by defining what `<script>` without any attributes does. The HTML file will be parsed until the script file is hit, at that point parsing will stop and a request will be made to fetch the file (if it's external). The script will then be executed before parsing is resumed.

<script async>
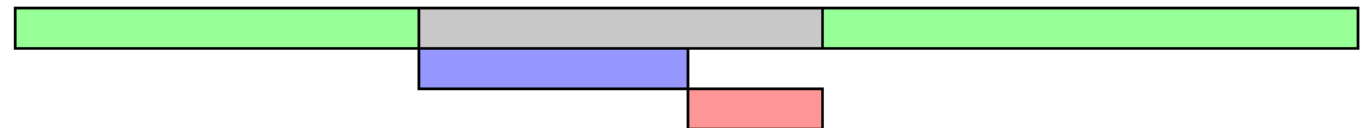
`async` downloads the file during HTML parsing and will pause the HTML parser to execute it when it has finished downloading.

<script defer>

`defer` downloads the file during HTML parsing and will only execute it after the parser has completed. `defer` scripts are also guaranteed to execute in the order that they appear in the document.

[new way] Javascript Modules (new in es6)

➡ provides encapsulation and namespace by default

➡ load the code asynchronously and defer execution by default

----
                                                index.html

```
<script type="module" src="file1.mjs">
```

----
                                                  file1.mjs

```
export doSomething = ...
```

----
                                                  file2.mjs

```
import { doSomething } from "/file1.mjs"
```

Structuring the Frontend

Frontend API

Backend Web API

Push & Pull

UI events

UI update

Controller