

# 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

<code>alert("hello world!")</code>	dialog box with “ok” button
<code>confirm("are you sure?")</code>	dialog box with “ok” and “cancel” buttons
<code>prompt("Name?", "John")</code>	input box with prompt text and default value

# The Browser

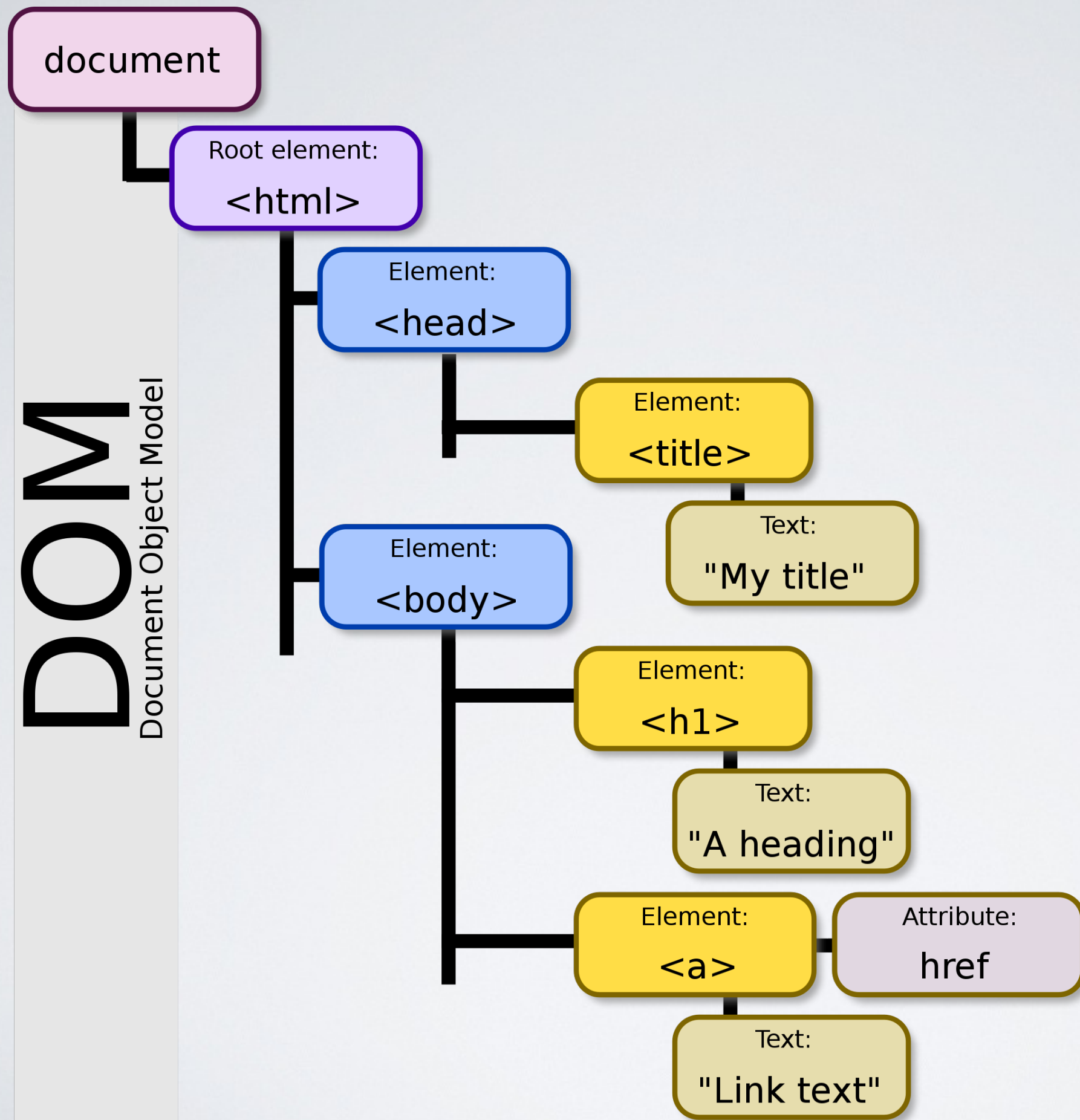
<b>screen</b>	the visitor's screen
<b>browser</b>	the browser itself
<b>window</b>	the current browser window
<b>url</b>	the current url
<b>history</b>	Back and forward URLs

# Document Object Model



# DOM

Document Object Model



# 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

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

# Events

# DOM events and handlers

<b>load</b>	when e is fully loaded
<b>click</b>	when e is clicked
<b>submit</b>	when e is submitted
<b>hover</b>	when the mouse is on top e
<b>keydown</b>	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

```
// 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**)
- Create a Frontend API (good practice)

New way



# The problem with Javascript interpreters

- ✓ **Good Javascript** is interpreted by browsers in a consistent way
- ⦿ **Bad javascript** code is loosely interpreted by browsers in an inconsistent way

## [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  
}
```

---

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



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

```
(function() {  
    "use strict";  
  
    let private = function() {  
        // private is not available from outside  
    }  
} ());
```

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

```
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;  
} ());
```

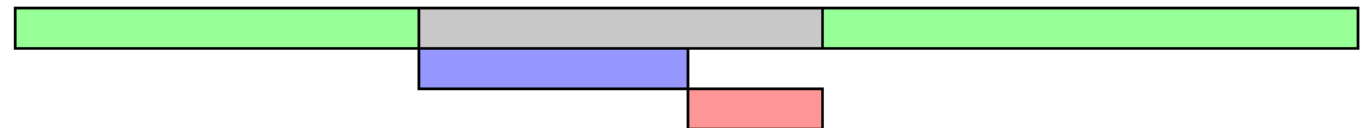
# [old way] Loading Javascript

## Legend

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

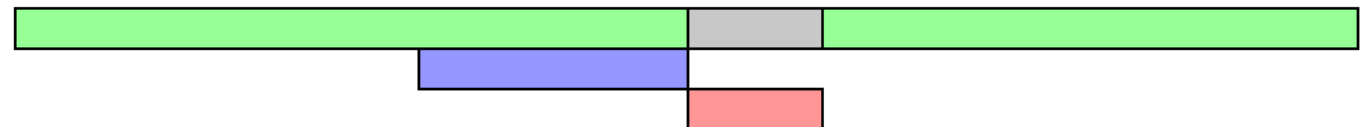
## <script>

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

