

Client-side Technologies

Eng. Niveen Nasr El-Den
iTi

Day 6

Browser Object Model

cont.

Browser Engine & JavaScript

- **Browser engine** is a core software component of every major web browser. The primary job of a browser engine is to transform HTML documents and other resources of a web page into an interactive visual representation on a user's device.

➤ e.g. **Blink**, Gecko, webkit etc.



BLINK
ENGINE



WEBKIT
ENGINE



TRIDENT
ENGINE



GECKO
ENGINE

- All Chromium-based browsers use Blink browser engine.
- **JavaScript engine** is a computer program that executes JavaScript (JS) code
 - e.g. **V8**, spiderMonkey etc.
- In 2019, **Microsoft** announced plans to rebuild the browser as **Chromium-based** with **Blink** and **V8** engines.

Example

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
    console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
    setTimeout(function () {  
        console.log("timeout immediately");  
    }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

setTimeout()

fun()

main()

...

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

console.log("start")

main()

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```



Event Loop

Callback Queue

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

wait5Sec()

main()



Event Loop

Callback Queue

```
console.log("start");
```

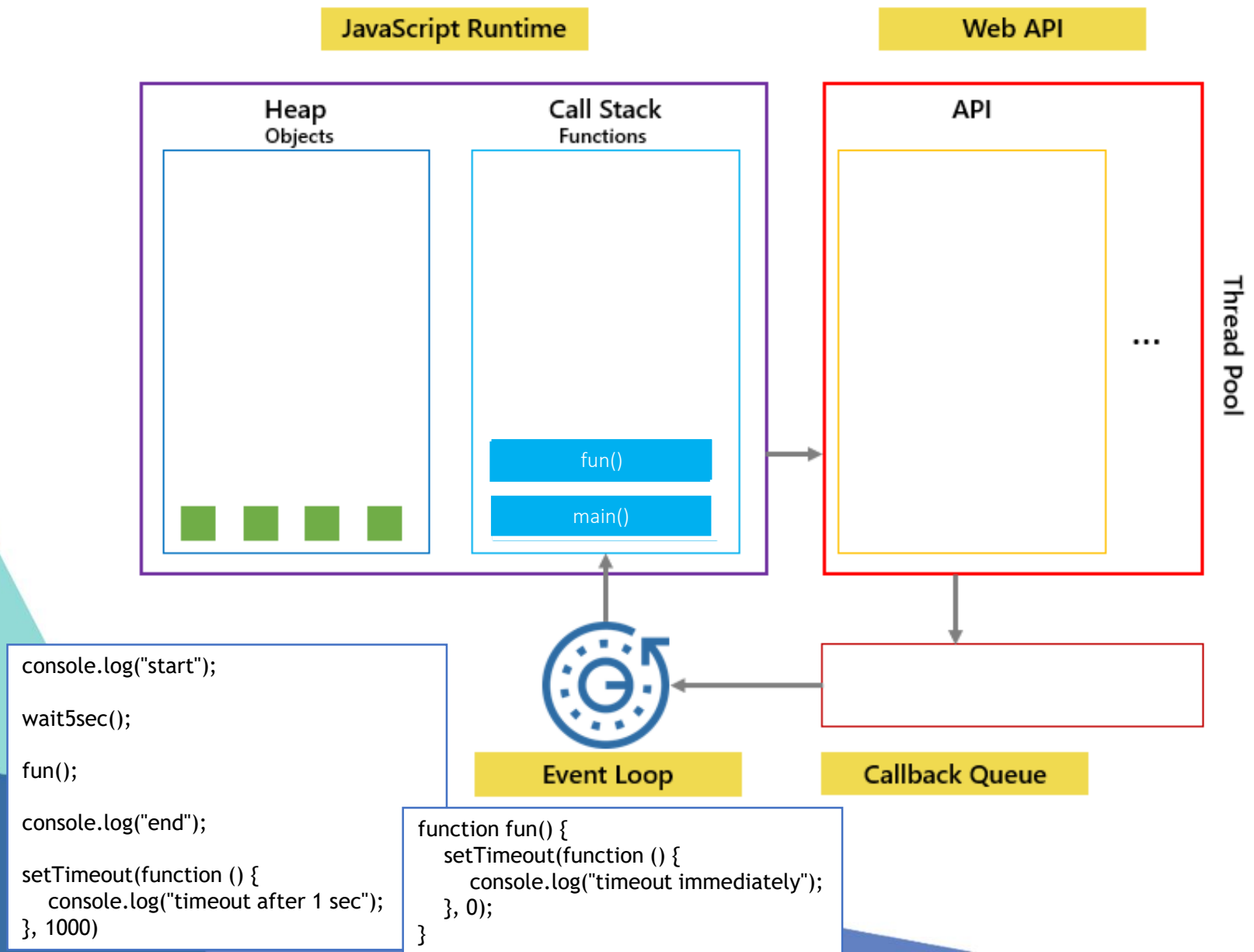
```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```



JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

setTimeout()

fun()

main()

...

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

setTimeout()

fun()

main()



fn{}



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

setTimeout()

fun()

main()

fn{}



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

setTimeout()

fun()

main()

fn{}

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

fun()

main()

fn{}



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

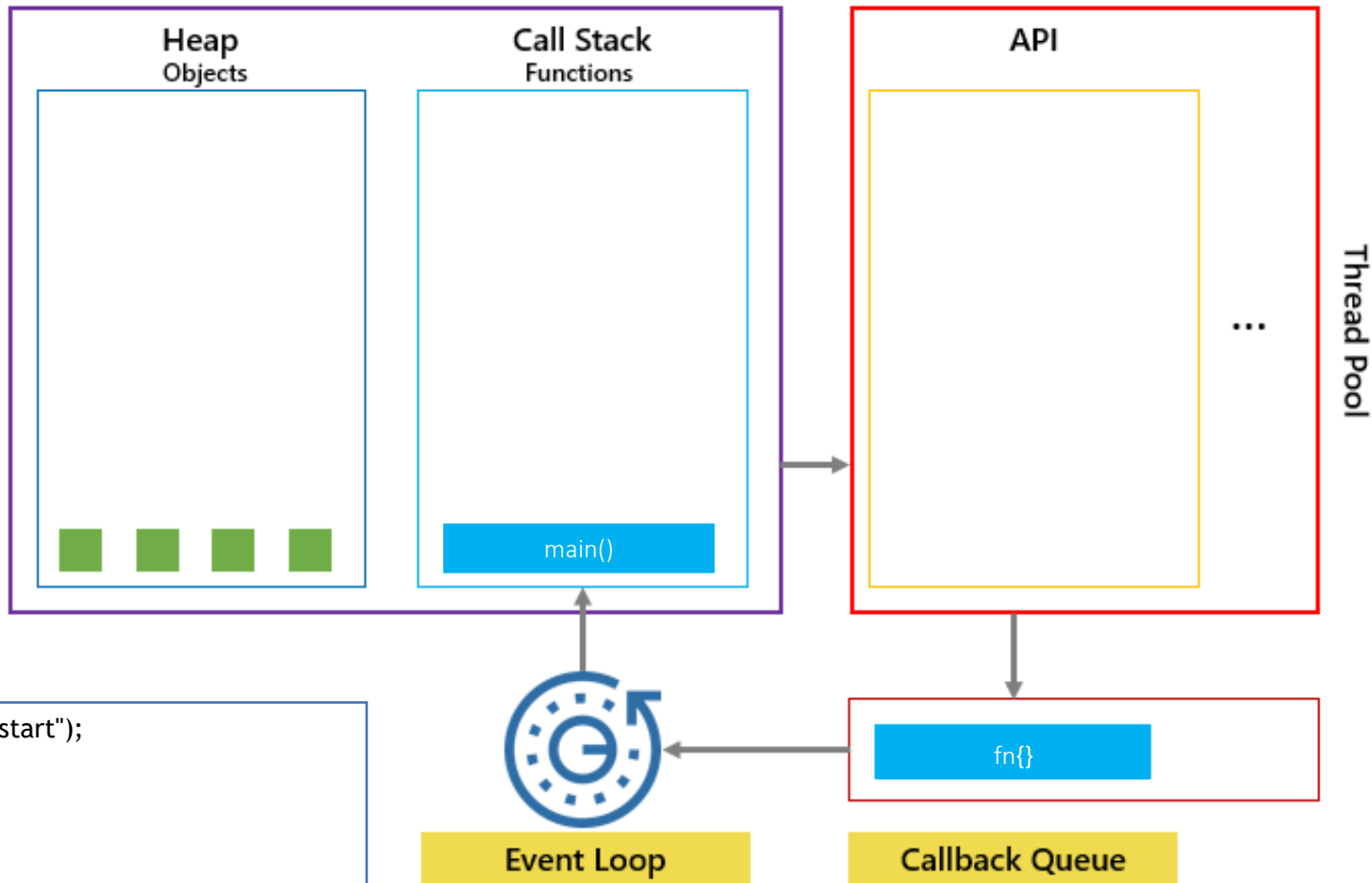
```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API



```
console.log("start");
```

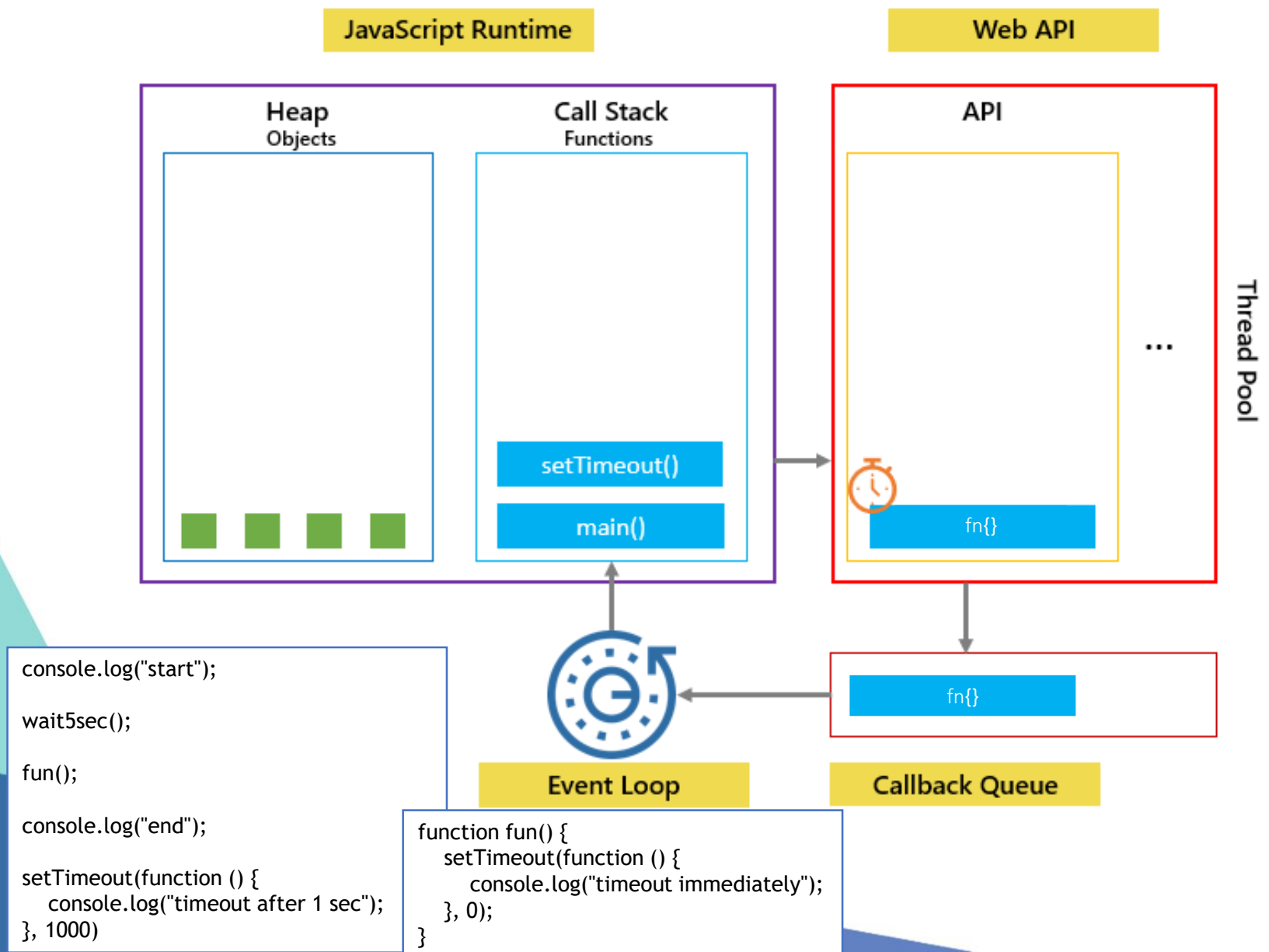
```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

main()

fn{

fn{



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool



Event Loop

fn{

fn{

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

fn{}

fn{}



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

fn{}

fn{}



Event Loop

Callback Queue

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

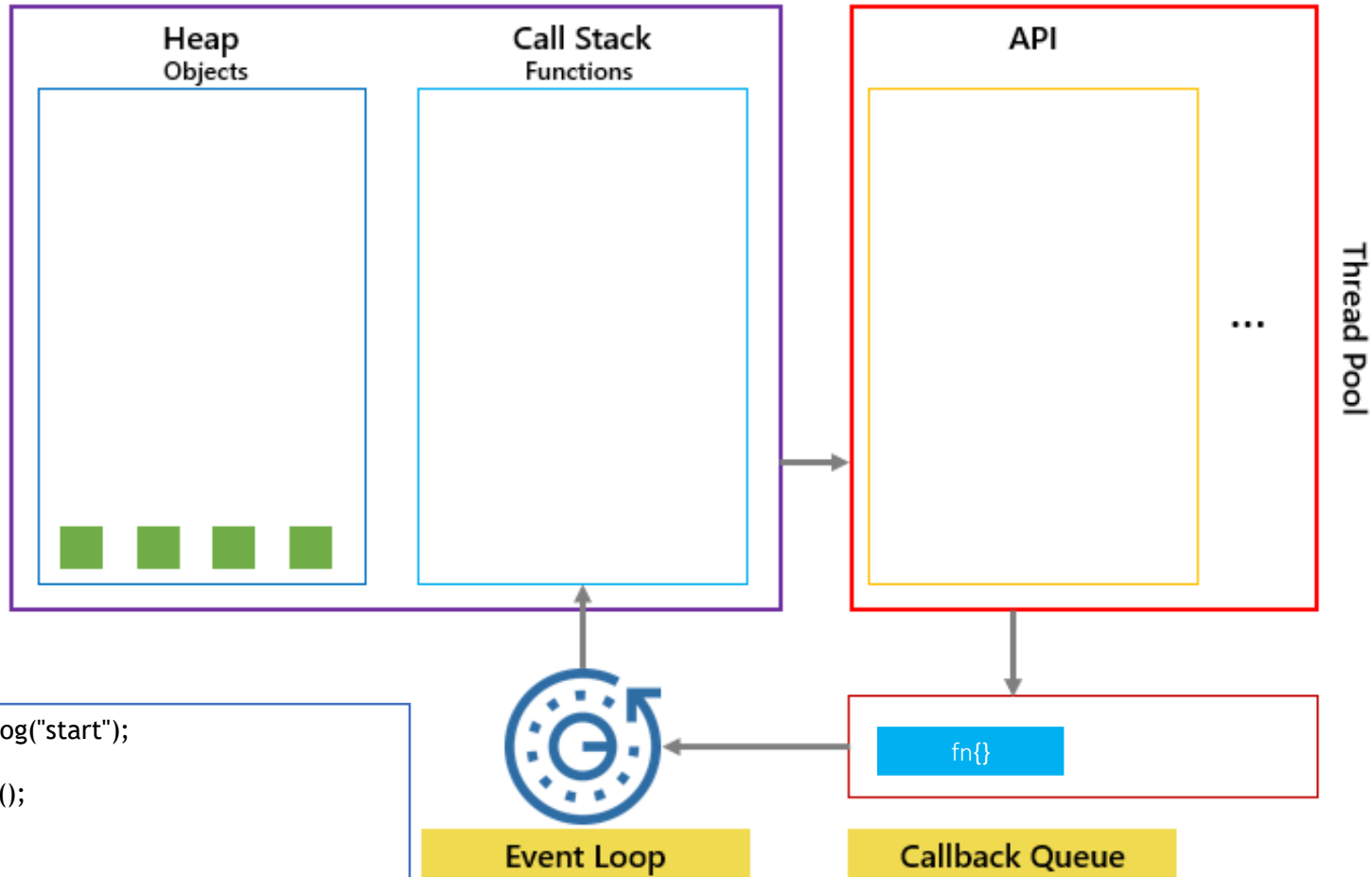
```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API



```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```

JavaScript Runtime

Web API

Heap
Objects

Call Stack
Functions

API

Thread Pool

fn{}

```
console.log("start");
```

```
wait5sec();
```

```
fun();
```

```
console.log("end");
```

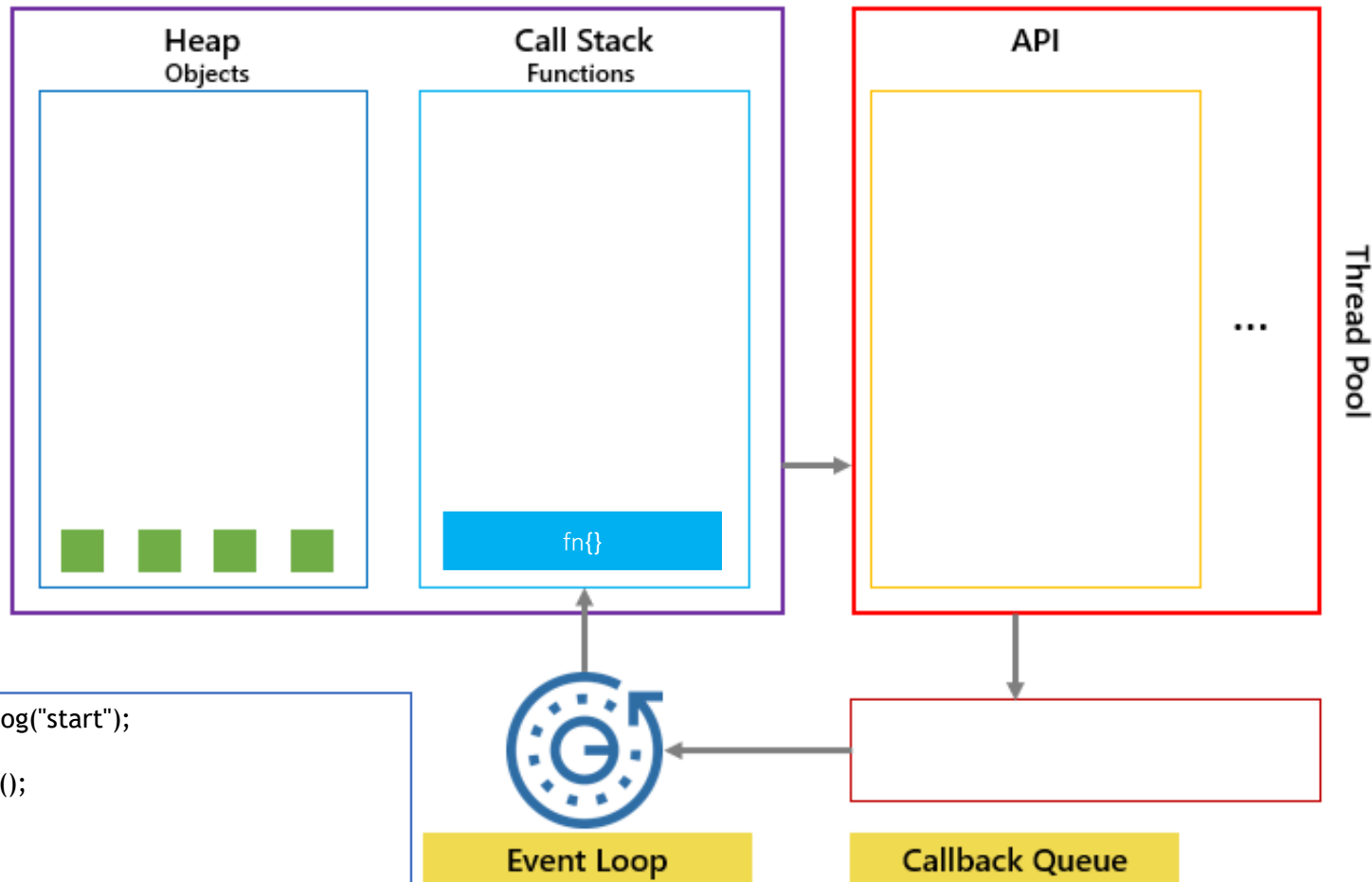
```
setTimeout(function () {  
  console.log("timeout after 1 sec");  
}, 1000)
```

```
function fun() {  
  setTimeout(function () {  
    console.log("timeout immediately");  
  }, 0);  
}
```



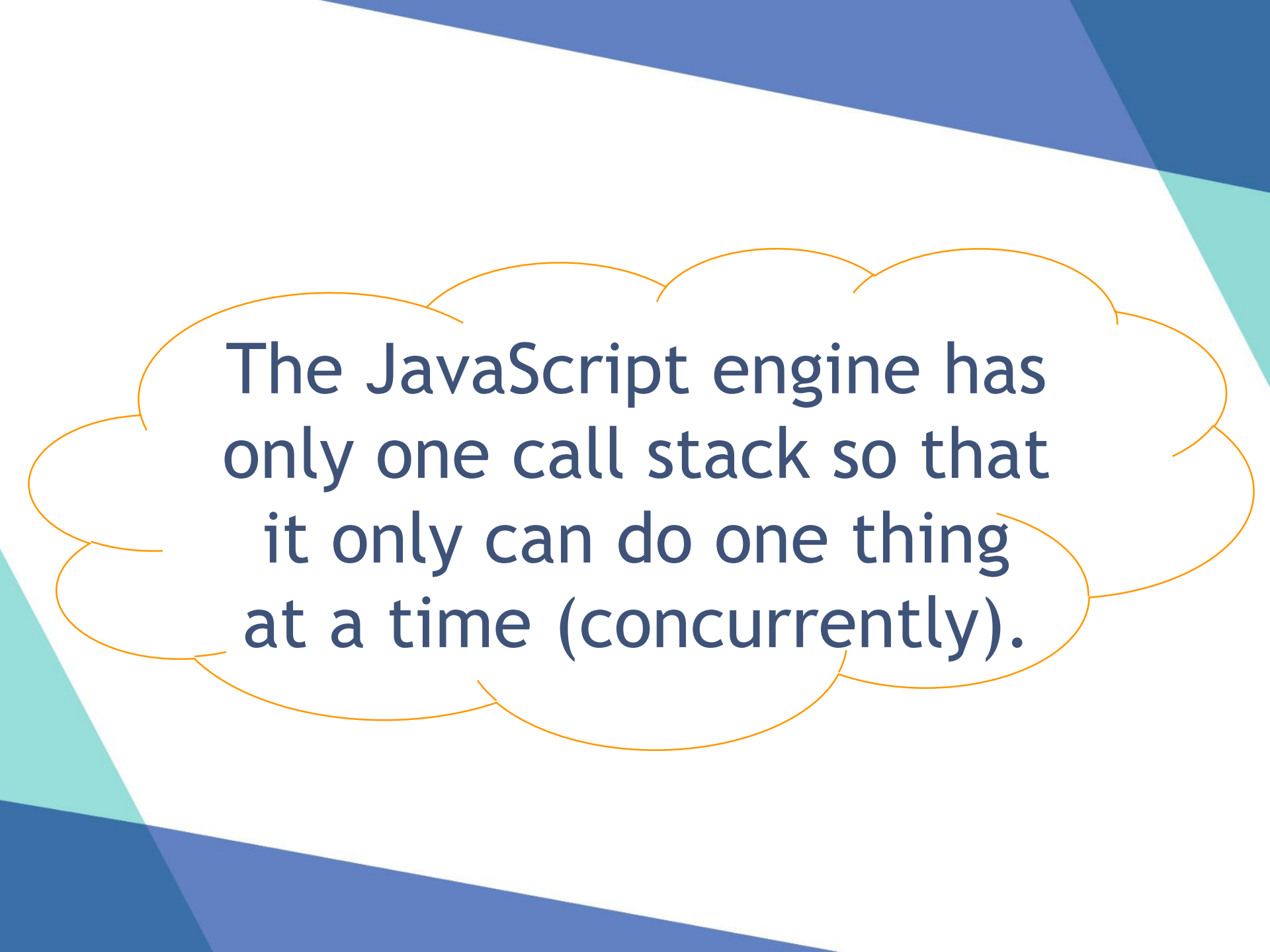
Event Loop

Callback Queue





JavaScript is the single-threaded
programming language.



The JavaScript engine has only one call stack so that it only can do one thing at a time (concurrently).



JavaScript execution is
synchronous.

When executing a script,
the JavaScript engine executes code
from top to bottom,
line by line.

Navigator

- The navigator object represents the browser application.
- This object allows scripts to get information about the browser like its type, version, language etc..
- Object Model reference:
 [window.]navigator
- All of its properties are read-only.

Navigator Properties & Methods

- Methods:

- ▷ **javaEnabled()**

<https://developer.mozilla.org/en-US/docs/Web/API/Navigator>

- Properties

Name	Description	Syntax
appName (deprecated)	get the name of the browser	navigator.appName
appVersion (deprecated)	get the version of the browser	navigator.appVersion
language	get the default language of the browser	navigator.language
cookieEnabled	returns whether the browser allows cookies or not	navigator.cookieEnabled
platform (deprecated)	return the name of the OS	navigator.platform
vendor (deprecated)	Returns the vendor name of the current browser	navigator.vendor

HTML5 New API →

geolocation

returns a Geolocation object allowing accessing the location of the device

Example!

Location

- The Location object is part of a Window object.
- The location Object refers to the current URL.
- Location contains information about the current URL of the browser. The most common usage of Location is simply to use it to automatically navigate (redirect) the user to another web page.
- It has a set of properties to hold the different components of the URL

- Object Model Reference:

[window.]location

```
<script type="text/javascript">  
    window.location=http://www.google.com  
</script>
```

Location Properties

Name	Description	Syntax
href	is the default property of the location object, returns the entire URL	location.href
protocol	represents the protocol of the URL.	location.protocol
hostname	specifies the host name	location.hostname
port	specifies the communication port.	location.port
host	is a combination of the host name and port	location.host
pathname	is the directory to find the document on the host, and the name of the file	location.pathname
search	specifies the queryString	location.search

Location Methods

- *replace* method loads the specified URL over the current history entry.

`location.replace(URL)`

- *reload* method Reloads the current document over the current history entry.

`location.reload()`

- *assign* method is almost the same as *replace* method. The difference is that it creates an entry in the browser's history list, while `replace()` doesn't.

`location.assign(URL)`

- *toString* method returns a string representation containing the whole URL

`location.toString ()`

Example!

History

- The history Object lets you send the user to somewhere in the history list from within a JavaScript program.

- Object Model reference:

[window.]history

- Properties:

length

- Methods:

back()	forward()	go()
--------	-----------	------

Example!

Document Object Model

DOM

DOM

- DOM Stands for Document Object Model.
- W3C standard.
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model
- Its an API that interact with documents like HTML, XML.. etc.
- Defines a standard way to access and manipulate HTML documents.
- Platform independent.
- Language independent

DOM

- The **document** object in the **BOM** is the top level of the **DOM** hierarchy.
- DOM is a representation of the whole document as nodes and attributes.
- You can access each of these nodes and attributes and change or remove them.
- You can also create new ones or add attributes to existing ones.

DOM is a **subset** of **BOM**.

In other word: **the document is yours!**

DOM

methods and properties, allows
accessing any element on the page,
modify, **delete** or **remove** elements,
or **add** new ones.

Document Object

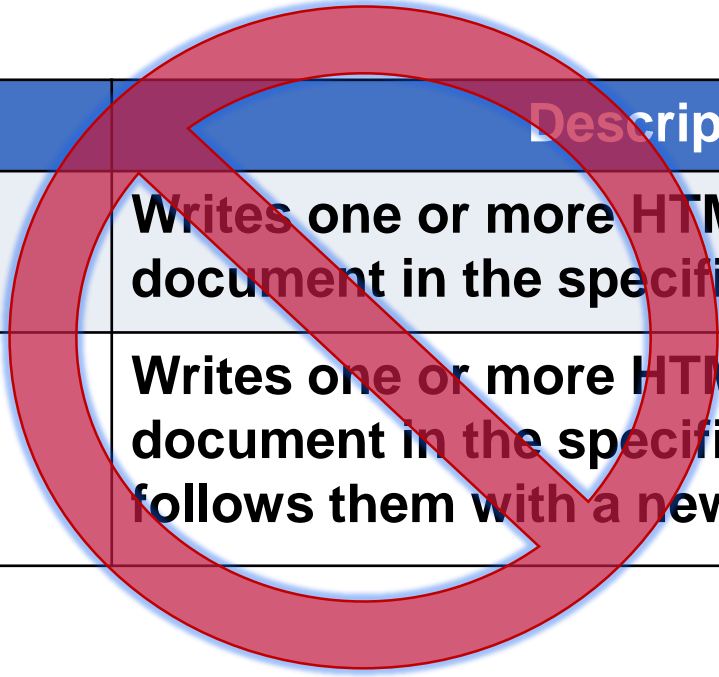
- The *document* object represents the entire HTML document and can be used to access all elements in a page.
- A *page* is what appears within the browser window.
- So, every window is associated with a document object.
- Document Object has its own set of Properties, Collections, Methods & Event handlers.



The **document** object in
the **BOM**
is the top level of
the **DOM** hierarchy.

Document Methods

Method	Description
write()	Writes one or more HTML expressions to a document in the specified window.
writeln()	Writes one or more HTML expressions to a document in the specified window and follows them with a new line character.



Example!

Document Methods

for accessing document elements

Method	Description
<code>getElementById("id")</code>	For accessing any element on the page via its ID attribute
<code>getElementsByName("name")</code>	Returns a collection of objects with the specified name
<code>getElementsByTagName("tagName")</code>	Returns a collection of objects with the specified tagname
<code>getElementsByClassName("className")</code>	Returns a collection of objects with the specified classname

Example!

New HTML5 Selectors

- In HTML5 we can select elements by ClassName

```
var elements = document.getElementsByClassName('entry');
```

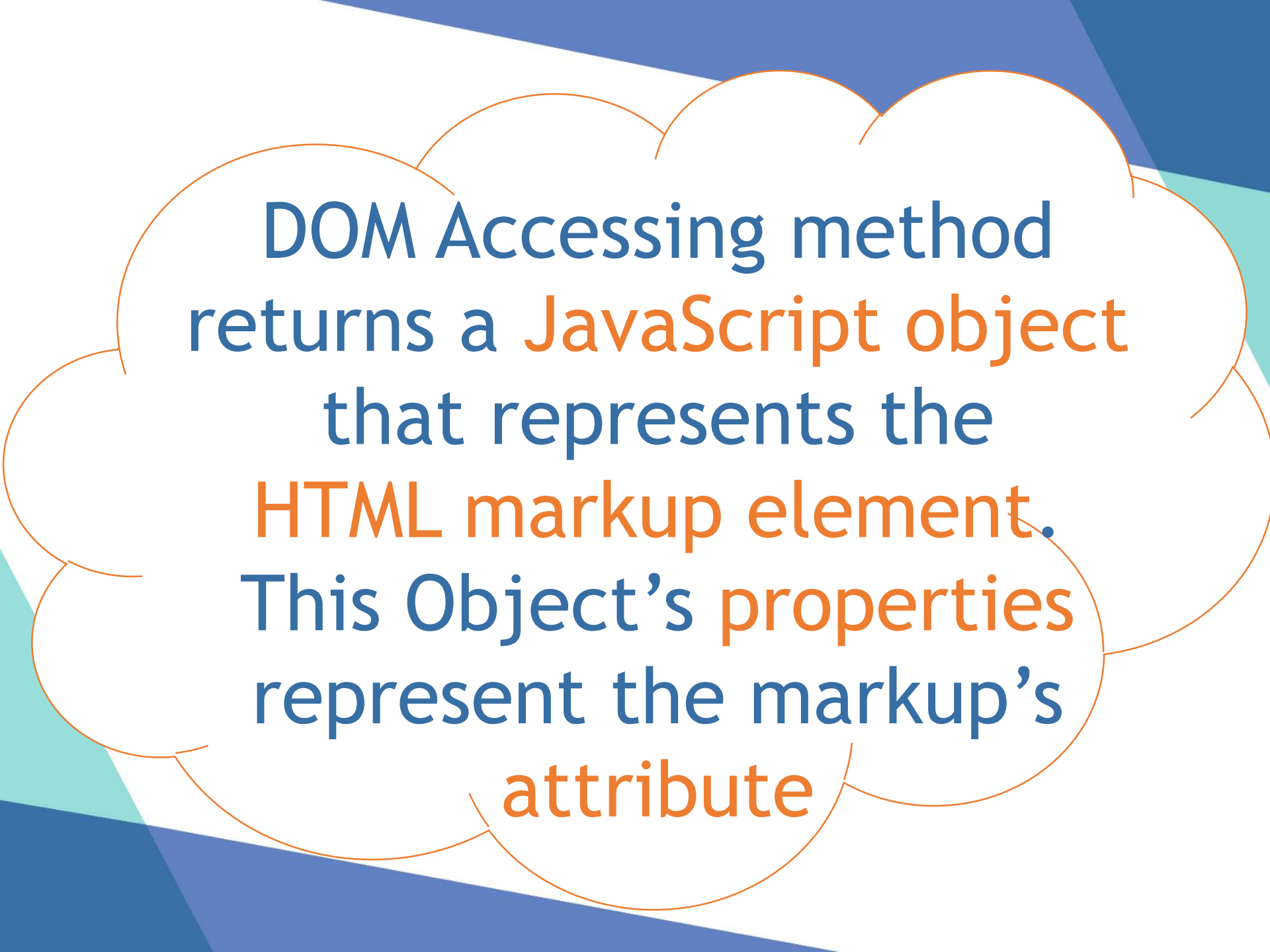
- Moreover there's now possibility to fetch elements that match provided CSS syntax

```
var elements = document.querySelectorAll(".someClasses");
```

```
var elements = document.querySelectorAll("div,p");
```

```
var elements = document.querySelector("#someID");
```

```
var first_td = document.querySelector("span");
```



DOM Accessing method
returns a JavaScript object
that represents the
HTML markup element.
This Object's properties
represent the markup's
attribute



We can access any
markup content via
`innerHTML`, `innerText`, or
`textContent`
properties

Document Properties

Property	Description
bgColor	A string that specifies the background color.
fgColor	A string that specifies the text color.
linkColor	The color of text for a link that the user has not yet visited.
vlinkColor	The color of text for a link that the user has already visited.
alinkColor	The color of text for a link that the user clicks.
title	A string that specifies the contents of the TITLE tag.
cookie	A string containing the name/value pair of cookies in the document.

Document Collection

- links, anchors, images, forms are collection/array containing all occurrences of those objects within the document.
- Since they are treated as arrays, they have the *length* property which specifies the number of entries in the collection/array.
- To access a specific entry in any of these collections, we can use either their index or name.

Document Collection

Collection	Description
forms[]	An array containing an entry for each form in the document
images[]	An array containing an entry for each image in the document
anchors[]	An array containing an entry for each anchor in the document.
links[]	An array containing an entry for each link in the document.

Document Collection

- An item from an object collection can be referenced in one of the following ways:
 1. `collection[i]`
 2. `collection.item(i)`
 3. `collection.namedItem(id)`
 4. `collection["name"]`
 5. `collection["id"]`
 6. `collection.name`
- Example:
 - `document.forms[0]`
 - `document.forms["myForm"]`
 - `document.forms["frmId"]`
 - `document.myForm`

Document Event Handler

onclick
ondblclick
onkeydown
onkeypress
onkeyup
onmousedown
onmouseup

Image

- The Image object is an image on an HTML form, created by using the HTML '**IMG**' tag.
- Any images created in a document are then stored in an array in the **document.images** property.
- Image Object has its own set of Properties, Methods & Event handlers.

Image

- Object Model Reference:

[window.]document.imageName

[window.]document.imageID

[window.]document.images[i]

[window.]document.images[imageName]

document.img1.src="img1.jpg"
document.images[0].src="img1.jpg"

document.img2.src="img2.jpg"
document.images[1].src="img2.jpg"

```
<html>
<body>
  ....
  
  
  ...
</body>
</html>
```

Image Properties & Event handlers

Properties

name	id	src	height	width
------	----	-----	--------	-------

Event handlers

onabort	onload	onerror	onclick	ondblclick	onmouseover
---------	--------	---------	---------	------------	-------------

Example!

Form

- By using the form you have at your disposal; information about the elements in a form and their values.
- A separate instance of the form object is created for each form in a document.
- Objects within a form can be referred to by a numeric index or be referred to by name.
- Object Model Reference:
 - [window.]document.formname
 - [window.]document.forms[i]
 - [window.]document.forms["formNAME"]
 - [window.]document.forms["formID"]

Form

Properties

```
<form  
  [name="formName"]  
  [target="frameName or windowName"]  
  [onsubmit="handlerText Or Function"]  
  [onreset="handlerText Or Function"]  
>  
</form>
```

Events

Form Properties

Property	Description
elements[]	An array containing all of the elements of the form. Use it to loop through form easily.
length	The number of elements in the form.
name	The name of the form.
id	The id of the form.
target	The name of the target frame or window form is to be submitted to.

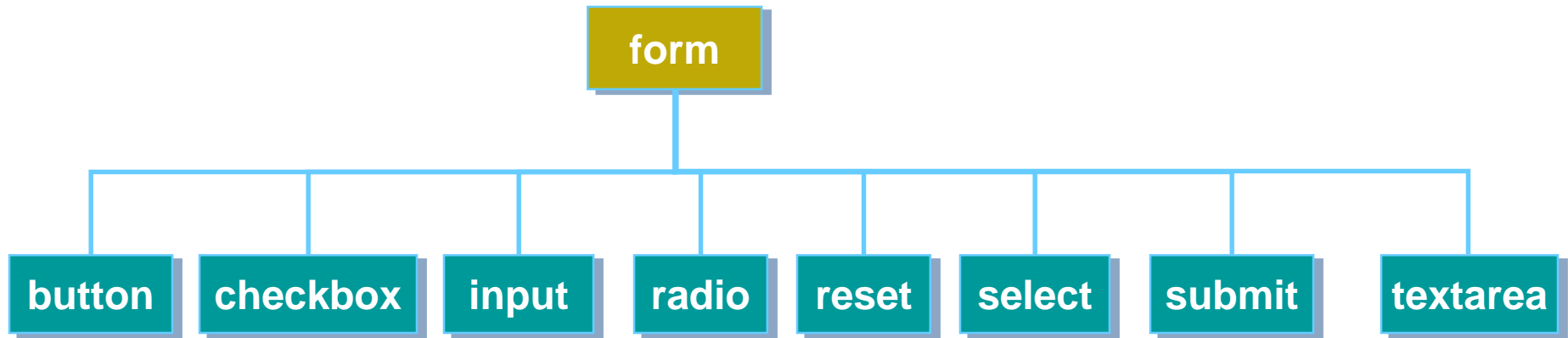
Form Methods

Method	Description
reset()	Resets the form. Clicking the reset button clears all contents that the user has made..
submit()	Submits a form. Clicking the submit button submits the content of the form to the server

Form Event Handler

Event	Description
onreset	Code is executed when the form is reset (by clicking on "reset" button)
onsubmit	Code is executed when form is submitted

Form



Text

```
<input type="text"  
      id="id"  
      value="string"  
      maxlength="n"  
      size="x"  
>
```

Text Event Handler

Event	Event Handler	Description
focus	onfocus	Fires when the field gains focus when the user tabs into or clicks inside the control.
blur	onblur	Fires when the field loses focus when the user tabs from or clicks outside the control.
change	onchange	Fires after changing the field value and losing being focused.
input	oninput	Fires every time the field value changes.
select	onselect	Fires when the content of the field being selected.

Text Methods

Method	Description
blur()	Removes focus from the field.
focus()	Assigns focus to the field; places the cursor in the control.
select()	Selects, or highlights, the content of the control.

Example!

Drop-down lists

```
<select  
  id="id"  
  multiple  
  size="n"  
>  
  <option value="string" selected > label </option>  
  <option value="string2" > label2 </option>  
  ...  
</select>
```

Drop-down lists Properties

Property	Description
length	The number of options in the list.
selectedIndex	The index number, beginning with 0, of the selected option.
options[]	An array of the options in the list. Used to reference properties associated with the options; e.g., options[1].value or options[2].text.
selected	A true or false value indicating whether an option is chosen.
value	The value associated with an option
text label	The text label associated with an option.

Drop-down lists Event Handler

Event Handler	Description
onfocus	The control gains focus.
onblur	The control loses focus.
onchange	A different option from the one currently displayed is chosen.

Example!

Radio Button

```
<input type="radio"  
  id="id"  
  name="name"  
  value="string"  
  checked  
>
```


Radio Button Properties

Property	Description
length	The number of radio buttons with the same name.
checked	A true or false value indicating whether a button is checked.
value	The value attribute coded for a button. A checked button with no assigned value is given a value of "on".

Radio Button Event Handler

Event Handler	Description
onfocus	The control gains focus.
onblur	The control loses focus.
onclick	The button is clicked.

Example!

Checkbox

```
<input type="checkbox"
  id="id"
  name="name"
  value="string"
  checked
/>
```

Button

```
<input type="button"  
  id="id"  
  value="string"  
>
```

Button Event Handler

Event Handler	Description
onclick	The mouse is clicked and released with the cursor positioned over the button.
ondblclick	The mouse is double-clicked with the cursor positioned over the button.
onmousedown	The mouse is pressed down with the cursor positioned over the button.
onmouseout	The mouse cursor is moved off the button.
onmouseover	The mouse cursor is moved on top of the button.
onmouseup	The mouse button is released with the cursor positioned over the button.

Reminder DOM References

- Use **this** keyword is used to refer to the current object.
 - ▷ e.g. the calling object in a method.

- Self reference to the object is used :

```
<input type="text"  
      onfocus = "this.value='You are in!'" />
```

- Passing current Object as a function parameter:

```
function myFunction(myObject) {  
    myObject.value = "In the function!!"  
}  
  
<input type="text" onclick="myFunction(this)" />
```

Assignment