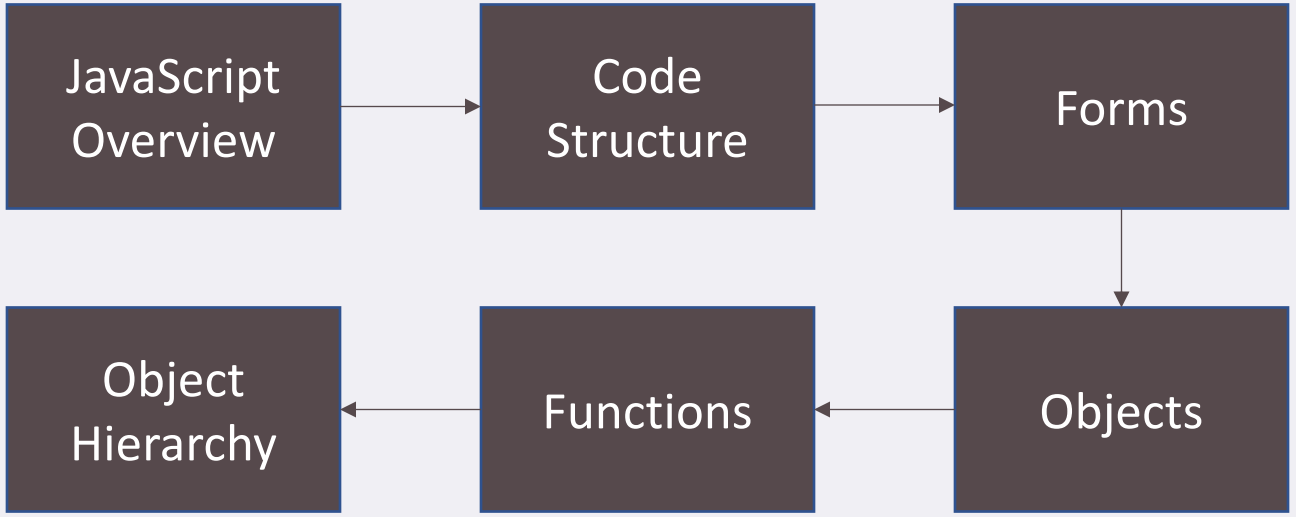
JAVASCRIPT

THE SCRIPTING LANGUAGE FOR WEB PROGRAMMING

TRAINING AGENDA



SOME QUICK CONSIDERATIONS BEFORE WE START

You need to code along with me!

Try all the coding challenges!

If you want the concepts to stick, take notes. Notes on code syntax, notes on theory concepts, notes on everything!

Before moving on from a topic, make sure that you understand exactly what was covered.

If you have an error or a question, start by trying to solve it yourself!

Most importantly, have fun!

JAVASCRIPT FUNDAMENTALS

A BRIEF INTRODUCTION TO JAVASCRIPT

WHAT IS JAVASCRIPT?

JAVASCRIPT IS A *HIGH-LEVEL*, *OBJECT-ORIENTED*, *MULTI-PARADIGM PROGRAMMING LANGUAGE.*

We don’t have to worry about complex stuff like memory management

We can use different styles of programming

Based on objects, for storing most kinds of data

Instruct computer to do things

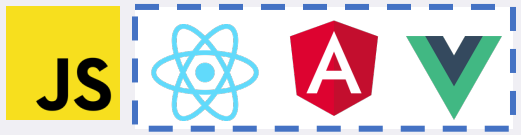
JAVASCRIPT FEATURES

HIGH-LEVEL	PROTOTYPE-BASED OBJECT-ORIENTED	MULTI-PARADIGM
INTERPRETED OR JUST-IN-TIME COMPILED	DYNAMIC	SINGLE-THREADED
NON-BLOCKING EVENT LOOP	FIRST-CLASS FUNCTIONS	GARBAGE-COLLECTED

THERE IS NOTHING YOU CAN'T DO WITH JAVASCRIPT

FRONT-END APPS

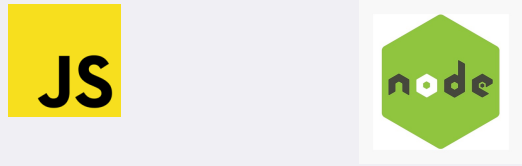
Dynamic effects and web applications in the browser



100% based on JavaScript.
They might go away,
but JavaScript won't!

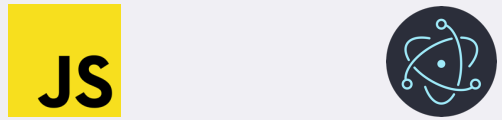
BACK-END APPS

Web applications on web servers

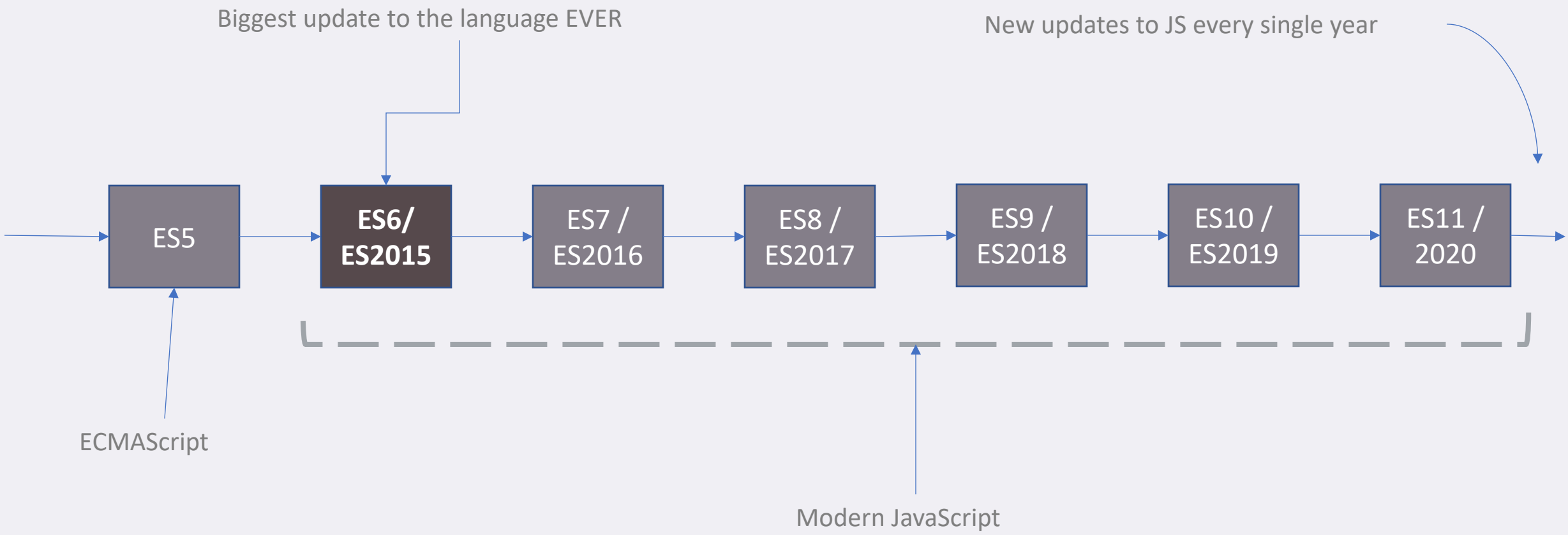


Native mobile applications

Native desktop applications



JAVASCRIPT RELEASES...



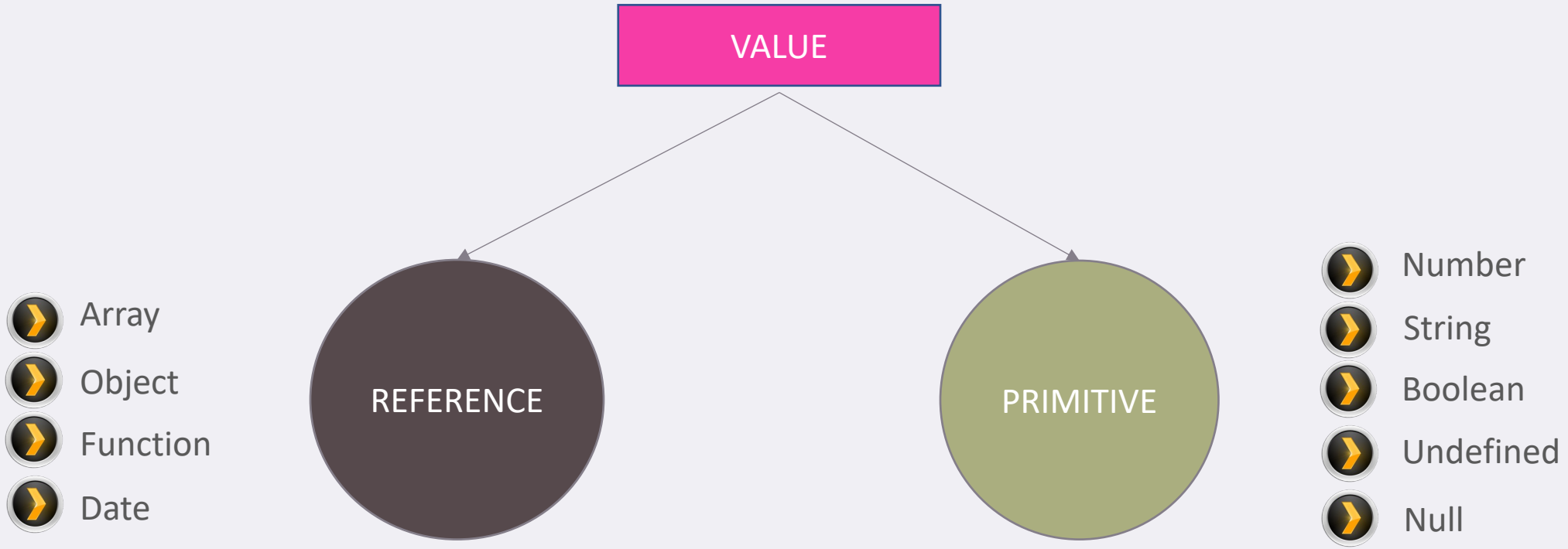
A BRIEF HISTORY OF JAVASCRIPT

1995	Brendan Eich creates the very first version of JavaScript in just 10 days. It was called Mocha.
1996	Mocha changes to LiveScript and then to JavaScript, in order to attract Java developers.
	Microsoft launches IE, copying JavaScript from Netscape and calling it JScript;
1997	With a need to standardize the language, ECMA releases ECMAScript 1 (ES1), the first official standard for JavaScript (ECMAScript is the standard, JavaScript the language in practice)
2009	ES5 (ECMAScript 5) is released with lots of great new features
2015	ES6/ES2015 (ECMAScript 2015) was released: the biggest update to the language ever!
	ECMAScript changes to an annual release cycle in order to ship less features per update
2016 onwards	Release of ES2016 / ES2017 / ES2018 / ES2019 / ES2020 / ES2021 / … / ES2089

DATA TYPES

LET’S GIVE IT A TYPE

OBJECTS AND PRIMITIVES

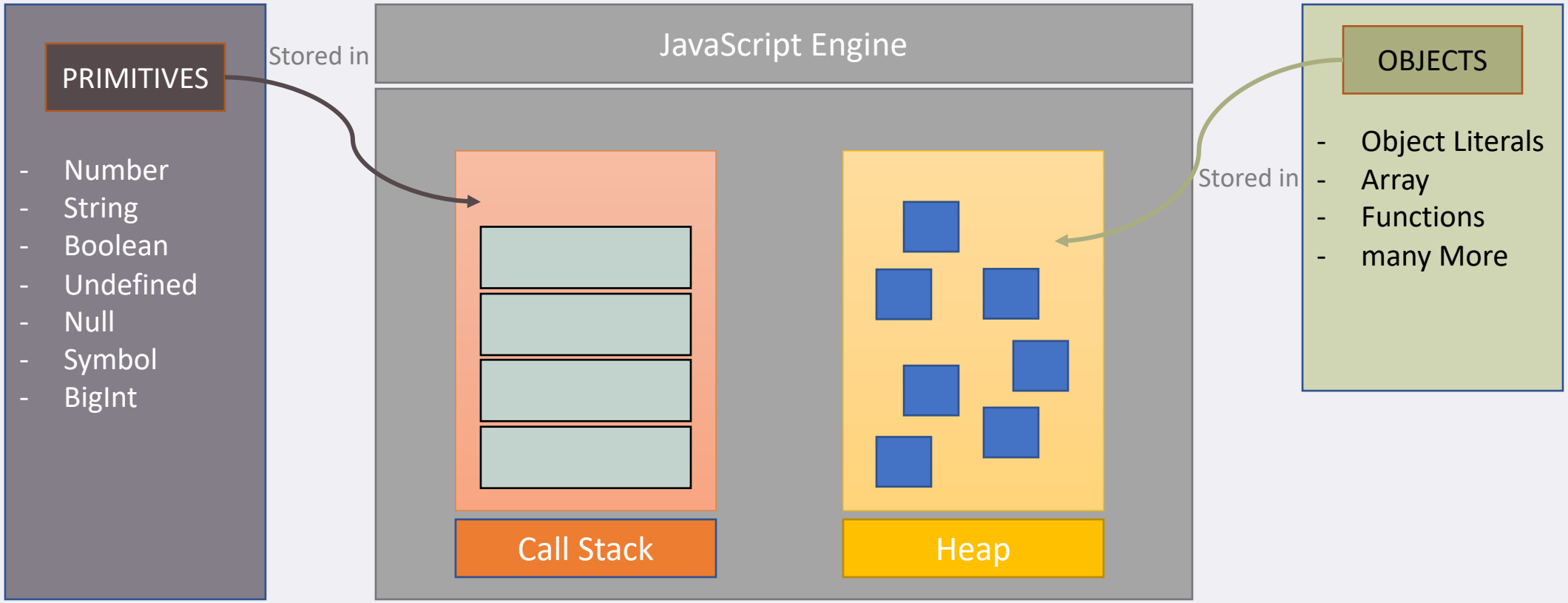


THE 7 PRIMITIVE DATA TYPES

Number	Floating point numbers. Used for decimals and integers
String	Sequence of characters. Used for text
Boolean	Logical type that can only be true or false. Used for taking decisions
Undefined	Value taken by a variable that is not yet defined (‘empty value’)
Null	Also means ‘empty value’
Symbol (ES2015)	Value that is unique and cannot be changed
BigInt (ES2020)	Larger integers than the Number type can hold

JavaScript has dynamic typing: We do not have to manually define the data type of the value stored in a variable. Instead, data types are determined automatically.

PRIMITIVES, OBJECTS AND THE JAVASCRIPT ENGINE



MISCELLANEOUS TOPICS

CONCEPTS FROM GROUND ZERO

MISCELLANEOUS TOPICS

let, const & var	Operators & Precedence	Strings & Template Literals
Type Conversion & Coercion	Conditional & Loops	Document Object Model

DATA STRUCTURE

STORING DATA IN OBJECT AND ARRAY

OBJECTS & ARRAYS

Objects represented by Flower Brackets – { }

- The Object type represents one of JavaScript's data types
- It is used to store various keyed collections and more complex entities
- Objects can be created using the Object() constructor or the object initializer / literal syntax

Arrays represented by Square Bracket - [ ]

- The Array object, as with arrays in other programming languages, enables storing a collection of multiple items under a single variable name, and has members for performing common array operations

WORKING WITH ARRAY

A COLLECTION OF MULTIPLE ITEMS UNDER A SINGLE VARIABLE NAME

ARRAY IN JAVASCRIPT

The Array object, as with arrays in other programming languages, enables storing a collection of multiple items under a single variable name, and has members for performing common array operations.

JavaScript arrays are resizable and can contain a mix of different data types

JavaScript arrays are not associative arrays and so, array elements cannot be accessed using strings as indexes

JavaScript arrays are zero-indexed

JavaScript array-copy operations create shallow copies

WHICH ARRAY METHOD TO USE?

I WANT …

TO MUTATE ORIGINAL ARRAY

- Add methods -
  - push()
  - unshift()
- Remove methods -
  - Pop()
  - shift()
  - Splice()
- Others -
  - Reverse()
  - Sort()
  - Fill()

A NEW ARRAY

- Computed from original-
  - Map()
- Filtered using condition-
  - Filter()
- Portion of original -
  - Slice()
- Adding original to other-
  - Concat()

AN ARRAY INDEX

- Based on value-
  - indexOf()
- Based on test condition
  - findIndex()
- An array element -
  - Find()

WHICH ARRAY METHOD TO USE?

I WANT …

KNOW IF ARRAY INCLUDES

- Based on value -
  - Includes()
- Based on test condition -
  - some()
  - every()
- Based on separator string-
  - Join()

TO TRANSFORM TO VALUE

- Based on accumulator
  - reduce()
- Based on callback-
  - forEach()

FUNCTIONS

FIRST CLASS CITIZENS IN JAVASCRIPT

FUNCTIONS

Function declaration
Function that can be used before it's declared

Function expression
Essentially a function value stored in a variable

Arrow function
Great for a quick one-line functions. (more later...)

```
function calcAge(birthYear) {
  return 2037 - birthYear;
}

const calcAge = function (birthYear) {
  return 2037 - birthYear;
};

const calcAge = birthYear ⇒ 2037 - birthYear;
```

Three different ways of writing functions, but they all work in a similar way -
Receive input data, transform data, and then output data.

FIRST-CLASS AND HIGHER-ORDER FUNCTIONS

FIRST-CLASS FUNCTIONS

JavaScript treats functions as first-class citizens.

This means that functions are simply values

Functions are just another “type” of object

Store functions in variables or properties

Pass functions as arguments to OTHER functions

Return functions FROM functions

HIGHER-ORDER FUNCTIONS

A function that receives another function as an argument, that returns a new function, or both

This is only possible because of first-class functions

Function that receives another function

Function that returns new function

CLOSURES

A closure is the closed-over variable environment of the execution context in which a function was created, even after that execution context is gone

Less Formal

A closure gives a function access to all the variables of its parent function, even after that parent function has returned. The function keeps a reference to its outer scope, which preserves the scope chain throughout time

Less Formal

A closure makes sure that a function doesn't loose connection to variables that existed at the function's birth place

SCOPE AND THE SCOPE CHAIN

WHERE DO OUR VARIABLES LIVE?

SCOPING AND SCOPE IN JAVASCRIPT: CONCEPTS

Scoping	How our program's variables are organized and accessed.
	"Where do variables live?" or "Where can we access a certain variable, and where not?";
Lexical scoping	Scoping is controlled by placement of functions and blocks in the code
Scope	Space or environment in which a certain variable is declared (variable environment in case of functions).
	There is global scope, function scope, and block scope;
Scope of a variable	Region of our code where a certain variable can be accessed

THE 3 TYPES OF SCOPE

GLOBAL SCOPE

- Outside of any function or block
- Variables declared in global scope are accessible everywhere

```
const me = 'Jonas';
const job = 'teacher';
const year = 1989;
```

FUNCTION SCOPE

- Variables are accessible only inside function, NOT outside
- Also called local scope

```
function calcAge(birthYear) {
  const now = 2037;
  const age = now - birthYear;
  return age;
}

console.log(now); // ReferenceError
```

BLOCK SCOPE (ES6)

- Variables are accessible only inside block (block scoped)
- HOWEVER, this only applies to let and const variables
- Functions are also block scoped (only in strict mode)

```
if (year >= 1981 && year <= 1996) {
  const millenial = true;
  const food = 'Avocado toast';
} ← Example: if block, for loop block, etc.

console.log(millenial); // ReferenceError
```

CONSIDERATION WHILE WORKING WITH SCOPE

Only *let* and *const* variables are block-scoped. Variables declared with *var* end up in the closest function scope.

In JavaScript, we have lexical scoping, so the rules of where we can access variables are based on exactly where in the code functions and blocks are written.

Every scope always has access to all the variables from all its outer scopes. This is the scope chain.

When a variable is not in the current scope, the engine looks up in the scope chain until it finds the variable it’s looking for. This is called variable lookup.

The scope chain is a one-way street: a scope will never, ever have access to the variables of an inner scope.

The scope chain has nothing to do with the order in which functions were called. It does not affect the scope chain at all.

THE *this* KEYWORD

this IS NOT THAT SAMPLE

HOW THE *this* KEYWORD WORKS

Special variable that is created for every execution context (every function). Takes the value of (points to) the “owner” of the function in which the this keyword is used.

‘this’ is NOT static. It depends on how the function is called, and its value is only assigned when the function is actually called.

Using ‘this’ with

- **Method** -> this = Object that is calling the method
- **Simple function call** -> this = undefined
- **Arrow functions** -> this = ‘this’ of surrounding function (lexical this)
- **Event listener** -> this = DOM element that the handler is attached to
- **call, apply, bind** -> change the function ‘this’ context

THE JAVASCRIPT ENGINE AND RUNTIME

HOW JAVASCRIPT WORKS BEHIND THE SCENES

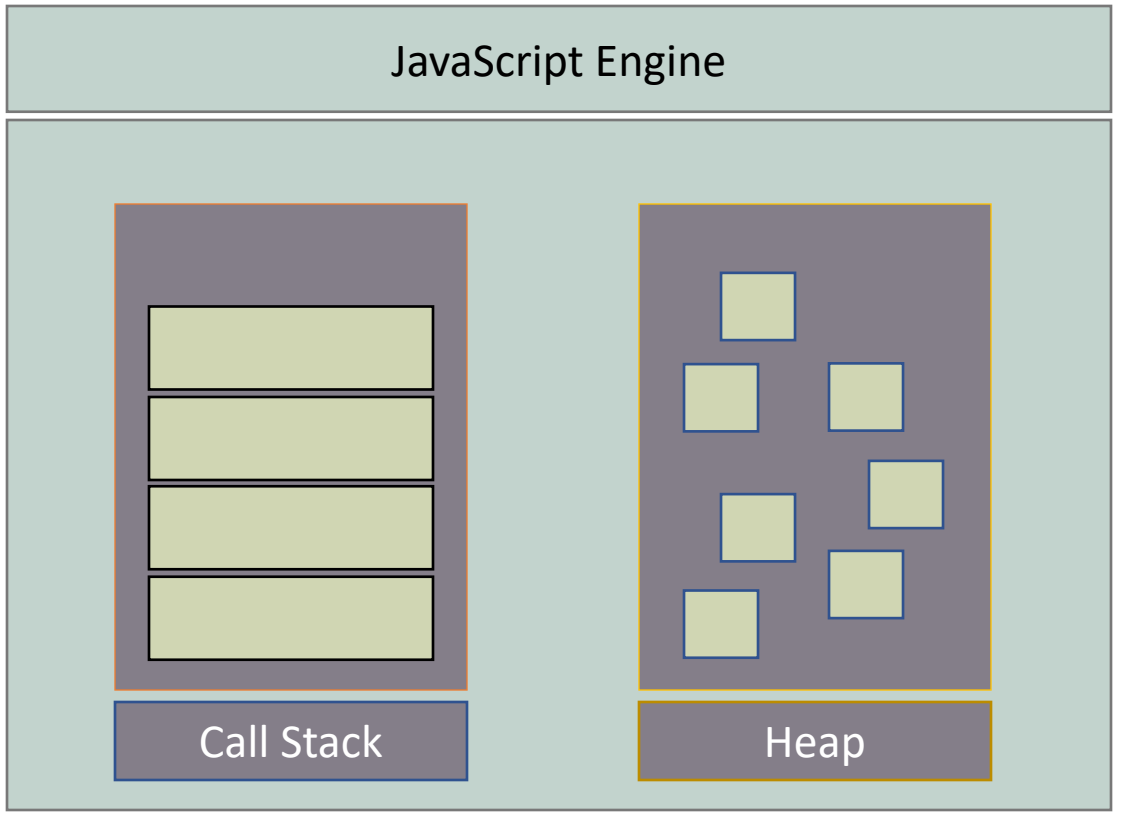
WHAT IS A JAVASCRIPT ENGINE?

JS ENGINE

PROGRAM THAT EXECUTES JAVASCRIPT CODE.

Example – V8 Engine

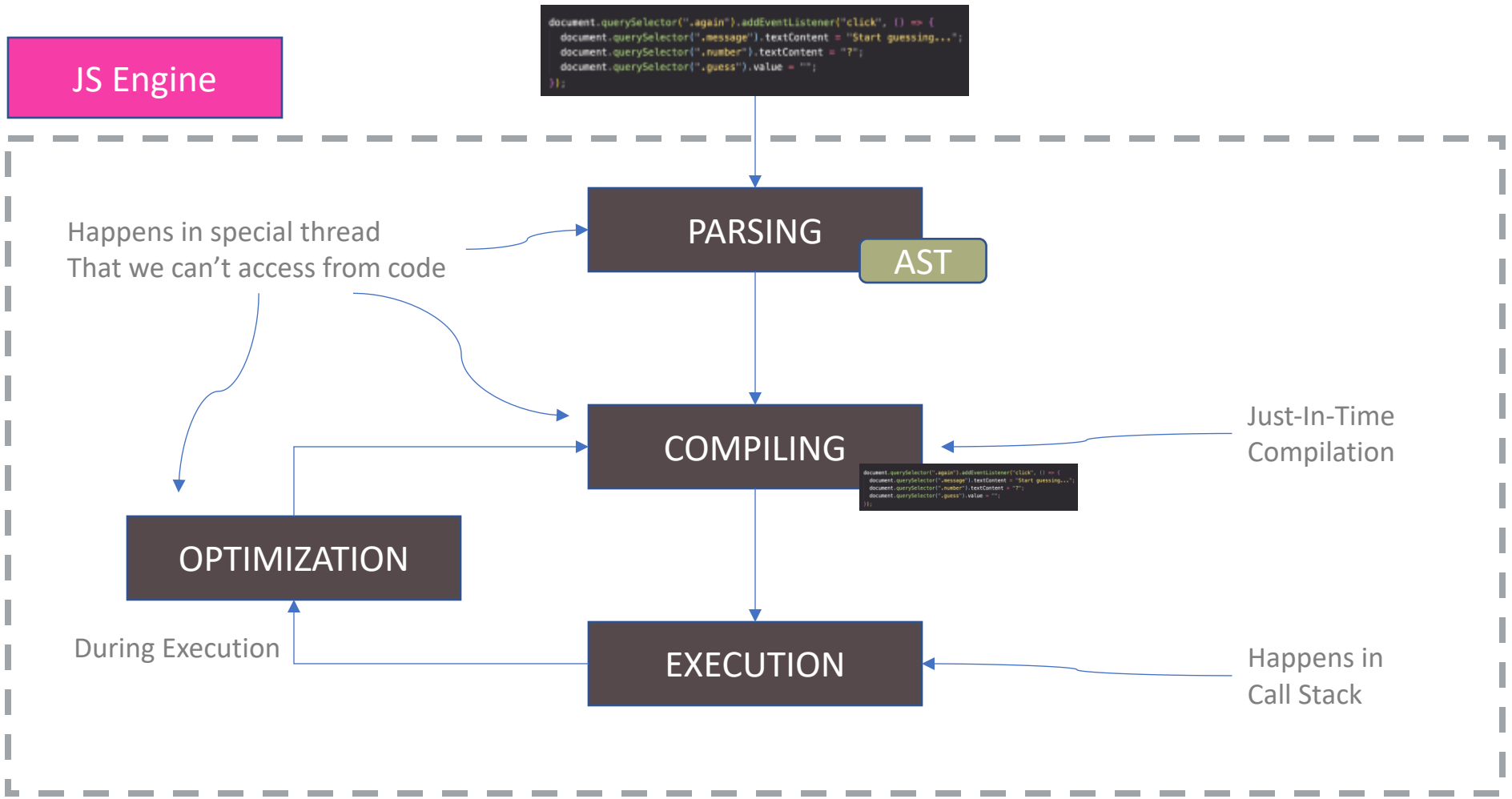
Execution Context



Where our code is executed

Where objects are stored

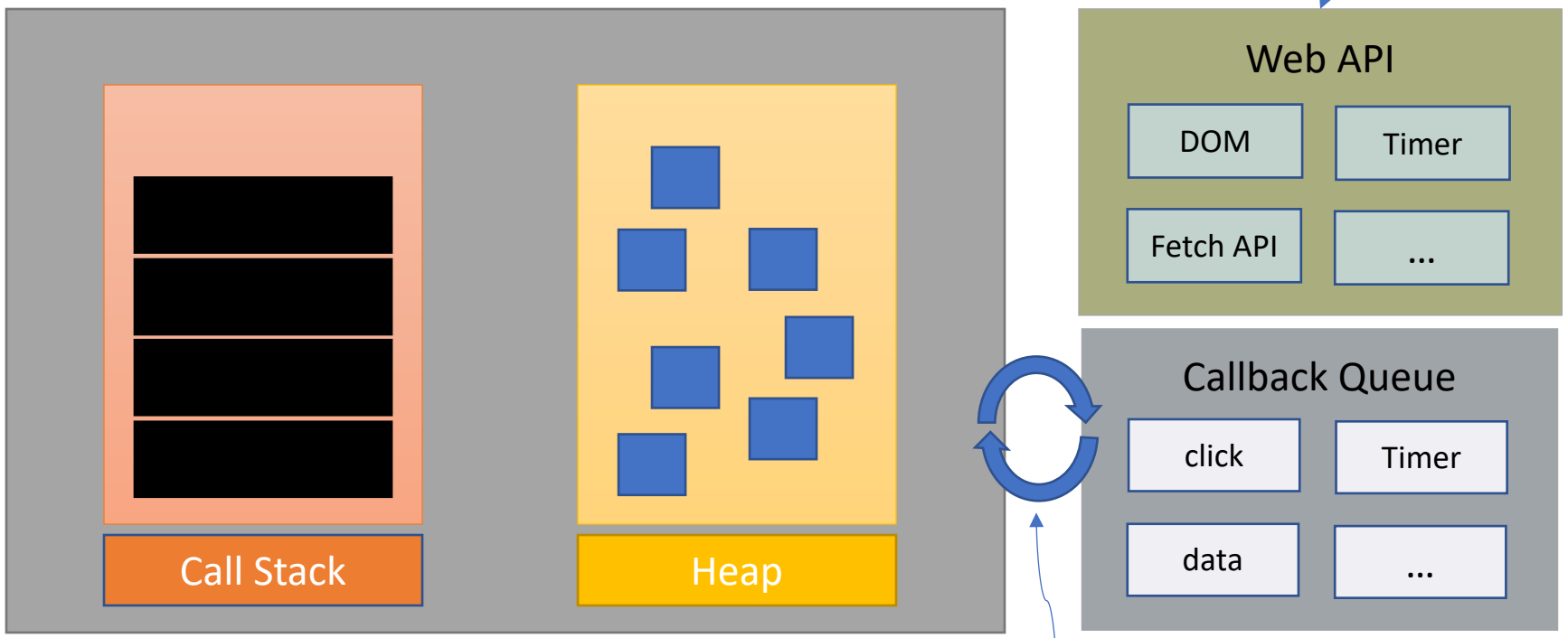
MODERN JUST-IN-TIME COMPILATION OF JAVASCRIPT



Container including all the things that we need to use JavaScript (in this case in the browser)

JS Runtime in the BROWSER

Functionalities provided to the engine, accessible on window object



Event Loop

Essential for non-blocking concurrency model

DOM AND EVENTS

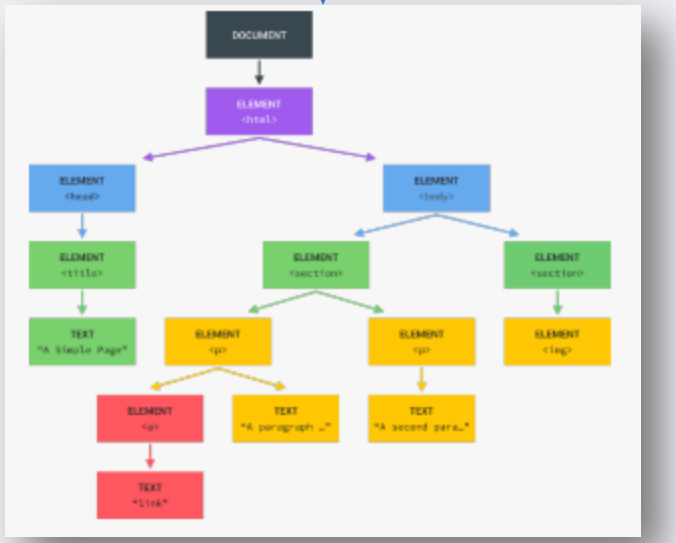
JAVASCRIPT IN THE BROWSER

WHAT IS THE DOM?

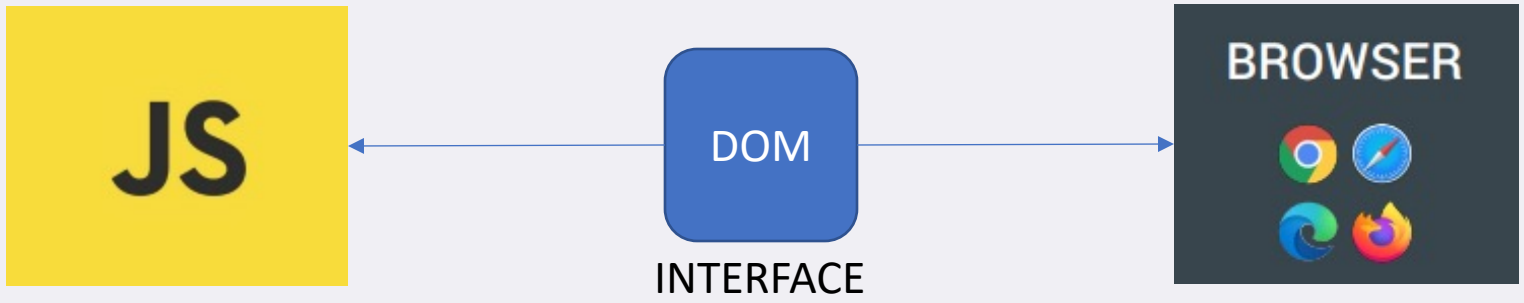
DOCUMENT OBJECT MODEL: STRUCTURED REPRESENTATION OF HTML DOCUMENTS. ALLOWS JAVASCRIPT TO ACCESS HTML ELEMENTS AND STYLES TO MANIPULATE THEM

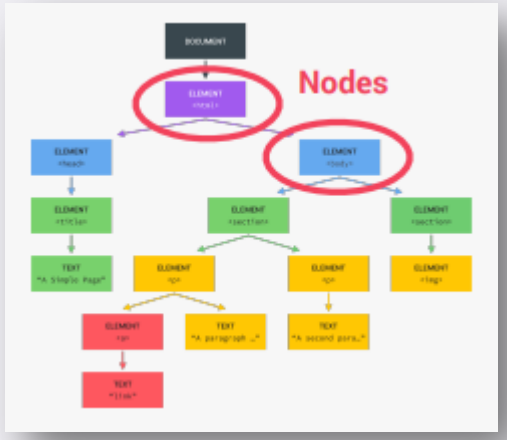
Change text, HTML attributes, and even CSS styles

Tree structure, generated by browser on HTML load



DOM IN DETAIL





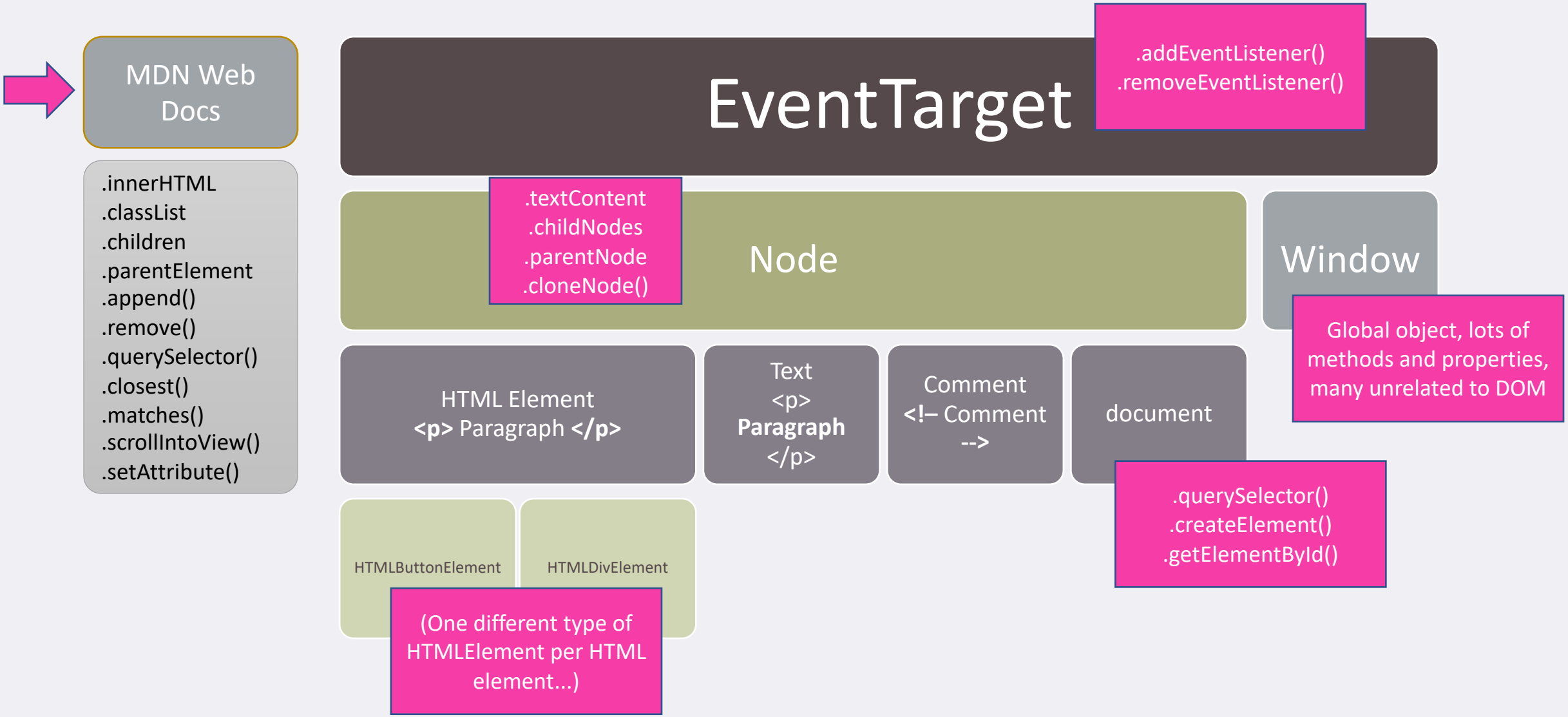
Allows us to make JavaScript interact with the browser

We can write JavaScript to create, modify and delete HTML elements set styles, classes and attributes; and listen and respond to events

DOM tree is generated from an HTML document, which we can then interact with

DOM is a very complex API that contains lots of methods and properties to interact with the DOM tree

HOW THE DOM API IS ORGANIZED BEHIND THE SCENES



REFERENCES

READING MATERIAL

- https://developer.mozilla.org/en-US/docs/Web/JavaScript
- https://javascript.info

VIDEO LINKS

- https://www.youtube.com/watch?v=POPLF-Qc0OU&list=PLsyeobzWxl7rrvgG7MLNIMSTzVCDZZcT4&index=2
- https://www.youtube.com/watch?v=chx9Rs41W6g