JavaScript
Beginner Course
Part2

jan.schulz@devugees.org

aemal.sayer@devugees.org

Agenda

1. What happens to our code?
2. Execution contexts and the Execution Stack
3. Hoisting
4. Scoping
5. Scoping VS Execution Stacks
6. this - Variable

BEHIND THE SCENES

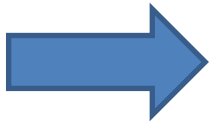
1. What happens to our code?

JAVASCRIPT CODE

```
function quickSort(items, left, right) {
    var index = 0;
    if (items.length > 1){
        left = typeof left != 'number' ? 0 : left;
        right = typeof right != 'number' ? items.length-1 : right;
        index = partition(items, left, right);
        if (left < index-1) {
            quickSort(items, left, index - 1);
        }
        if (index < right) {
            quickSort(items, index, right);
        }
    }
    return items;
}
// first call
var result = quickSort(items);
```

1. What happens to our code?

JAVASCRIPT CODE

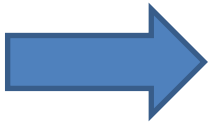


PARSER

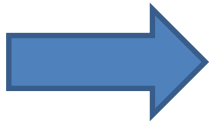
```
function quickSort(items, left, right) {
    var index = 0;
    if (items.length > 1){
        left = typeof left != 'number' ? 0 : left;
        right = typeof right != 'number' ? items.length-1 : right;
        index = partition(items, left, right);
        if (left < index-1) {
            quickSort(items, left, index - 1);
        }
        if (index < right) {
            quickSort(items, index, right);
        }
    }
    return items;
}
// first call
var result = quickSort(items);
```

1.What happens to our code?

JAVASCRIPT CODE

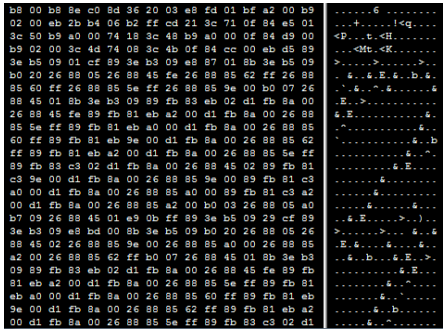


PARSER



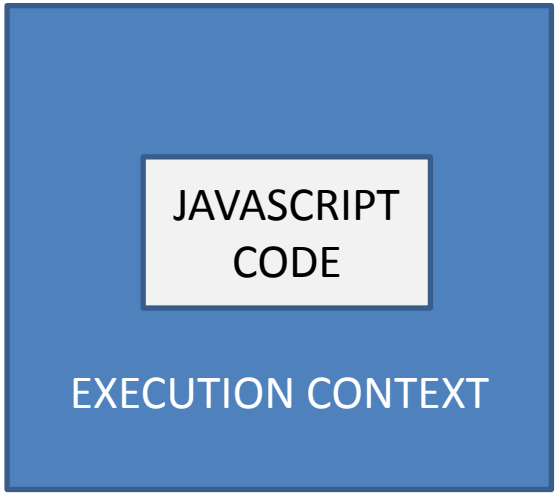
RUNTIME COMPILER

```
function quickSort(items, left, right) {
    var index = 0;
    if (items.length > 1){
        left = typeof left != 'number' ? 0 : left;
        right = typeof right != 'number' ? items.length-1 : right;
        index = partition(items, left, right);
        if (left < index-1) {
            quickSort(items, left, index - 1);
        }
        if (index < right) {
            quickSort(items, index, right);
        }
    }
    return items;
}
// first call
var result = quickSort(items);
```



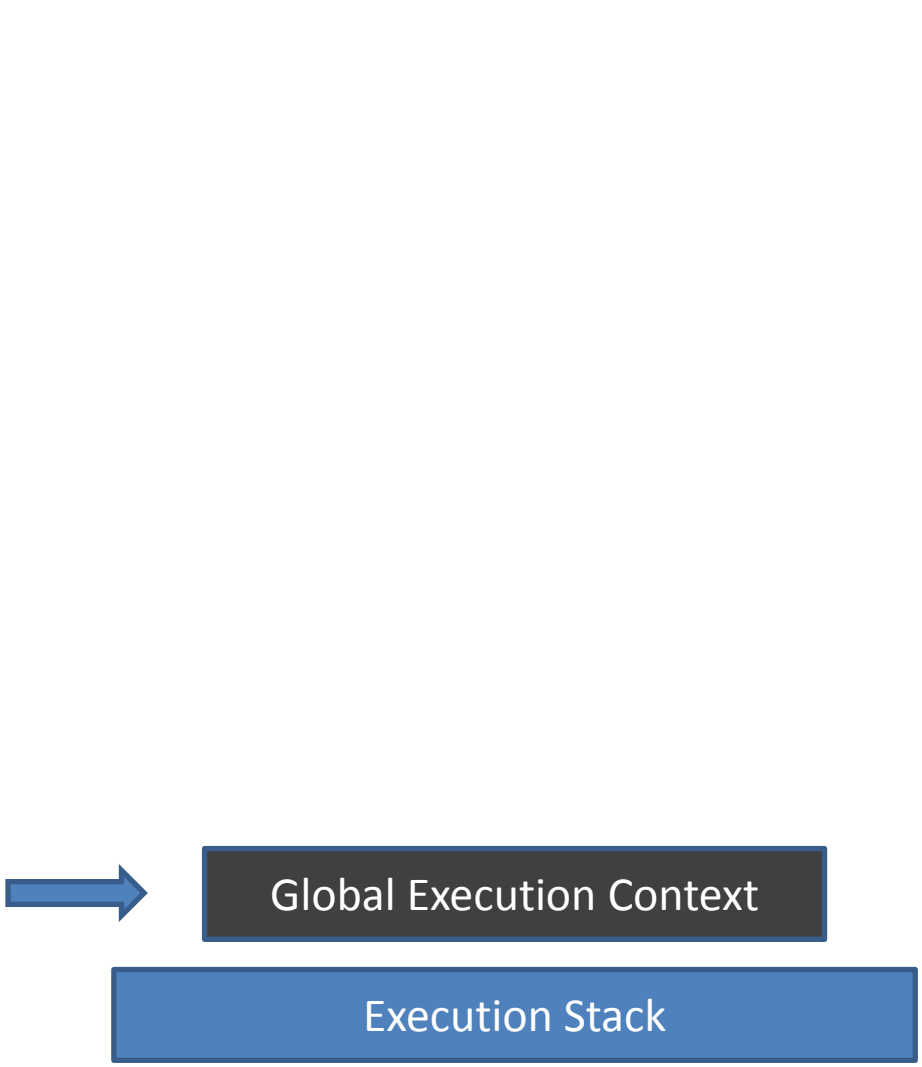
2. Execution contexts and Execution Stack

Execution Context is a Box/ Container/ Wrapper



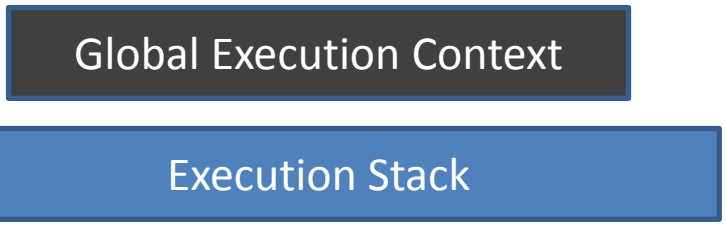
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



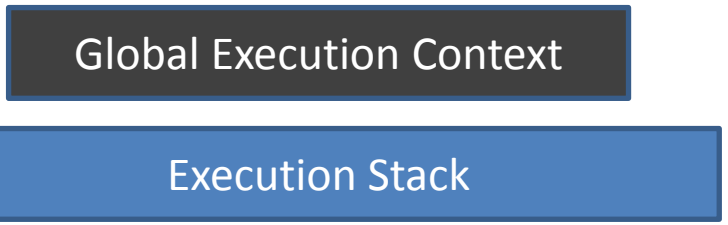
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



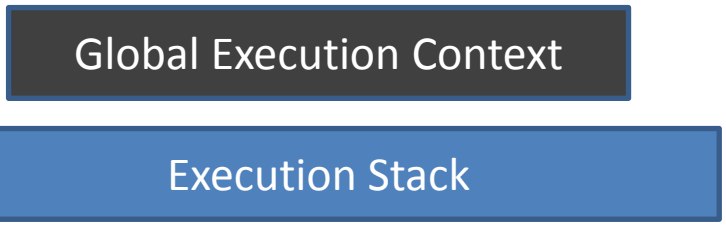
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



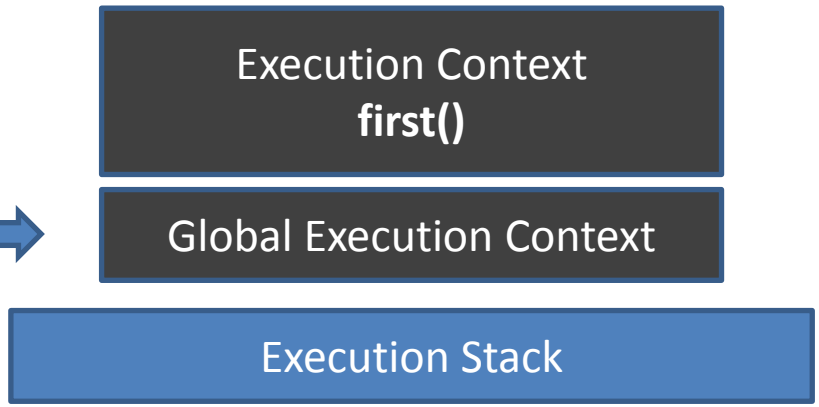
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



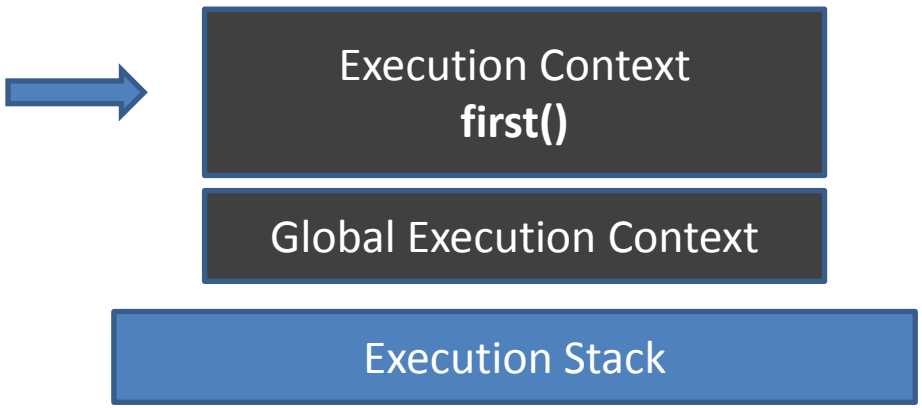
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```

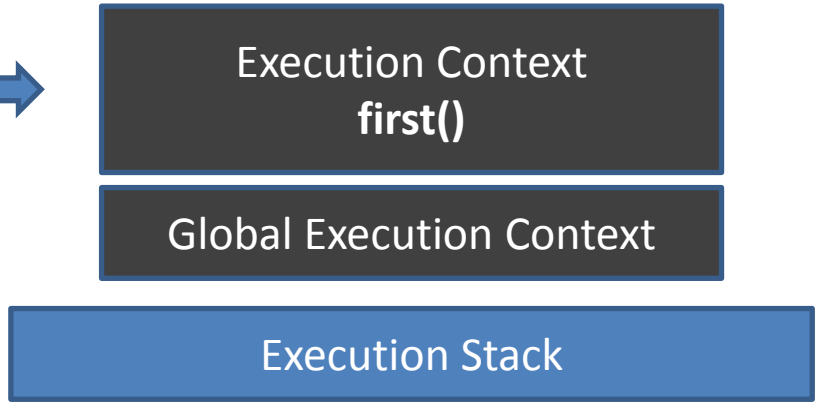


2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```

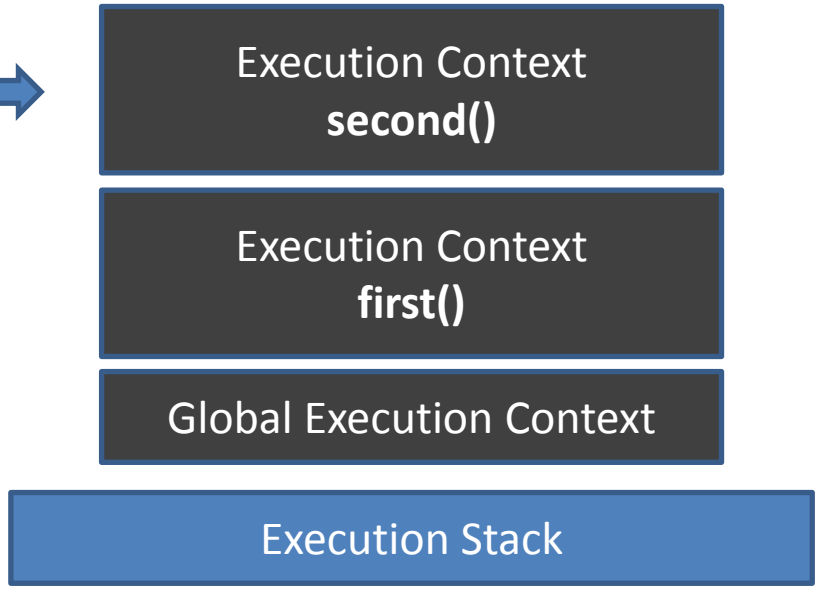
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



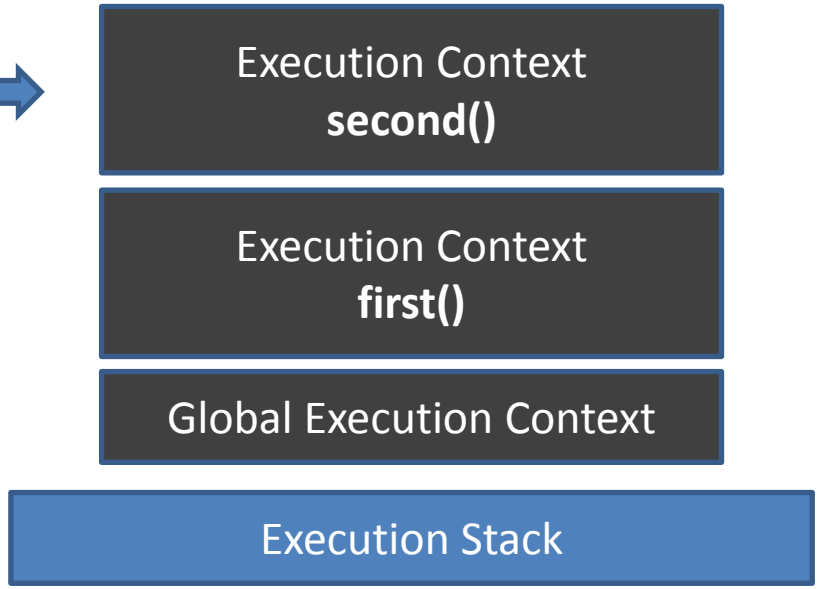
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



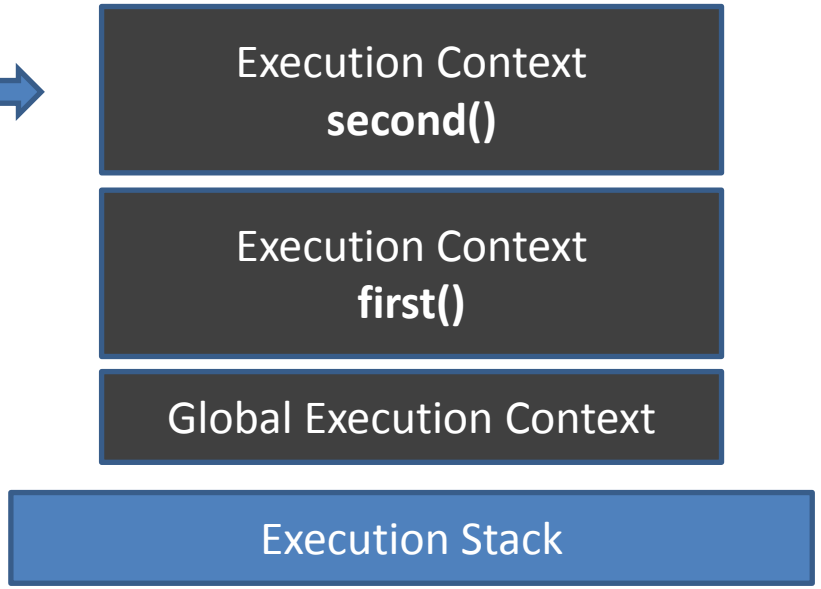
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



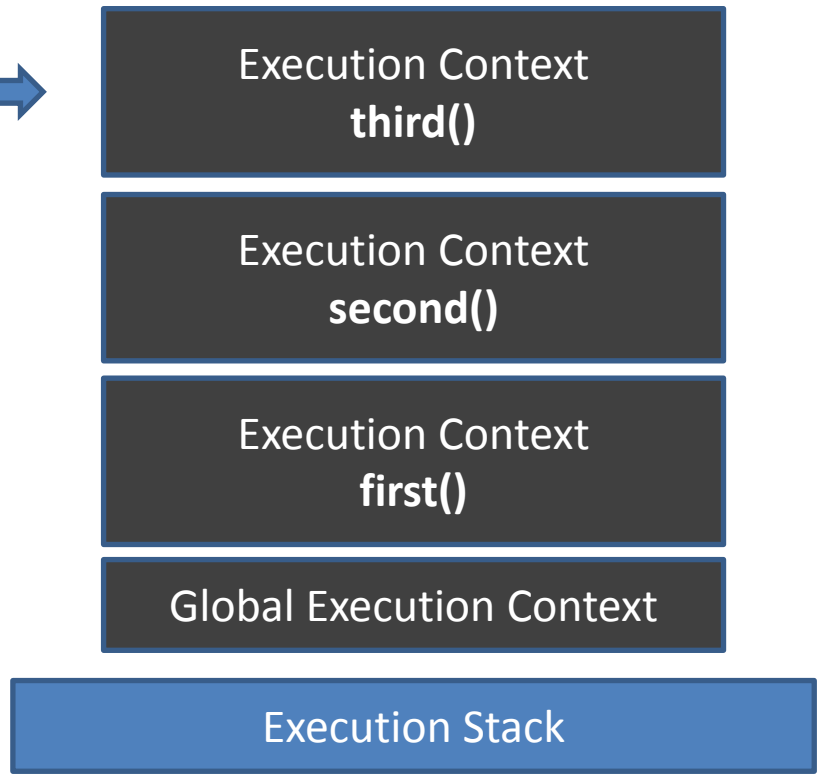
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



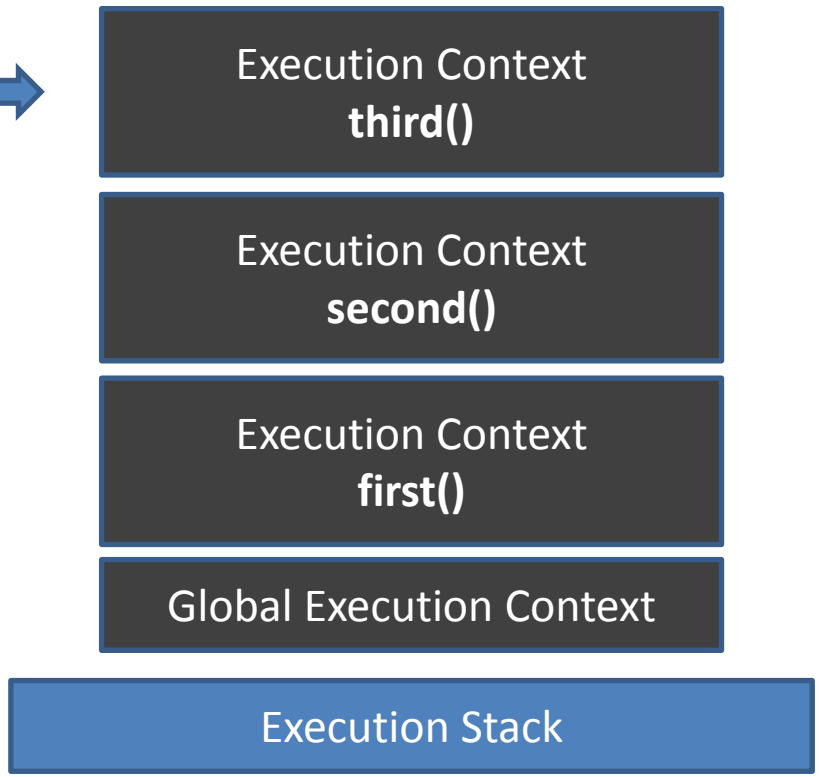
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



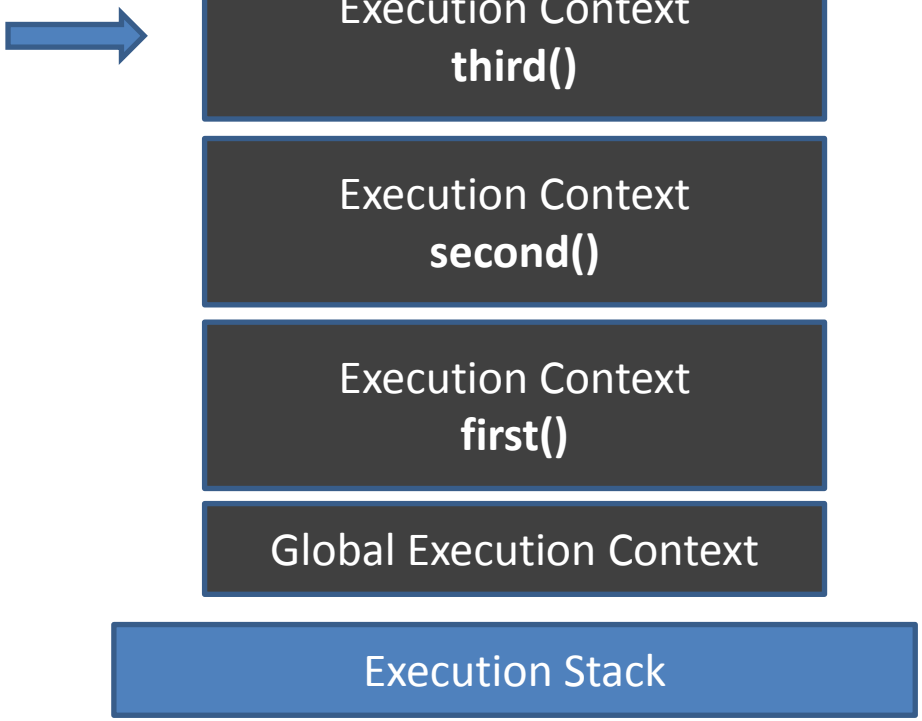
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



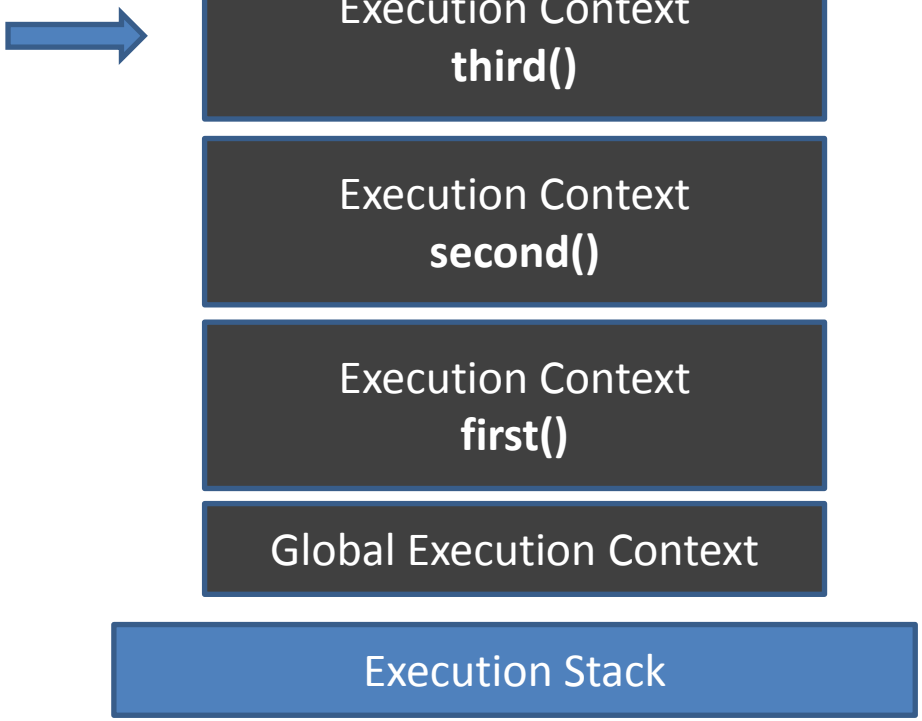
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



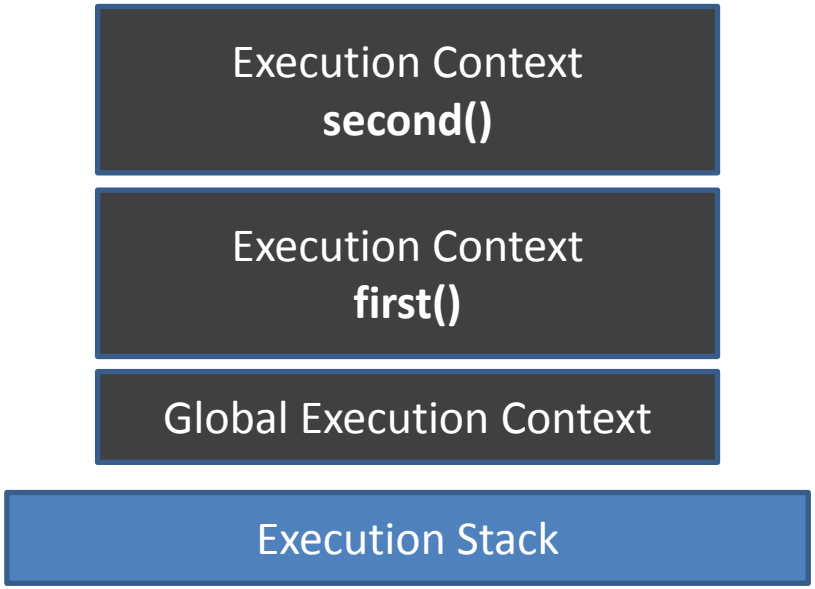
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



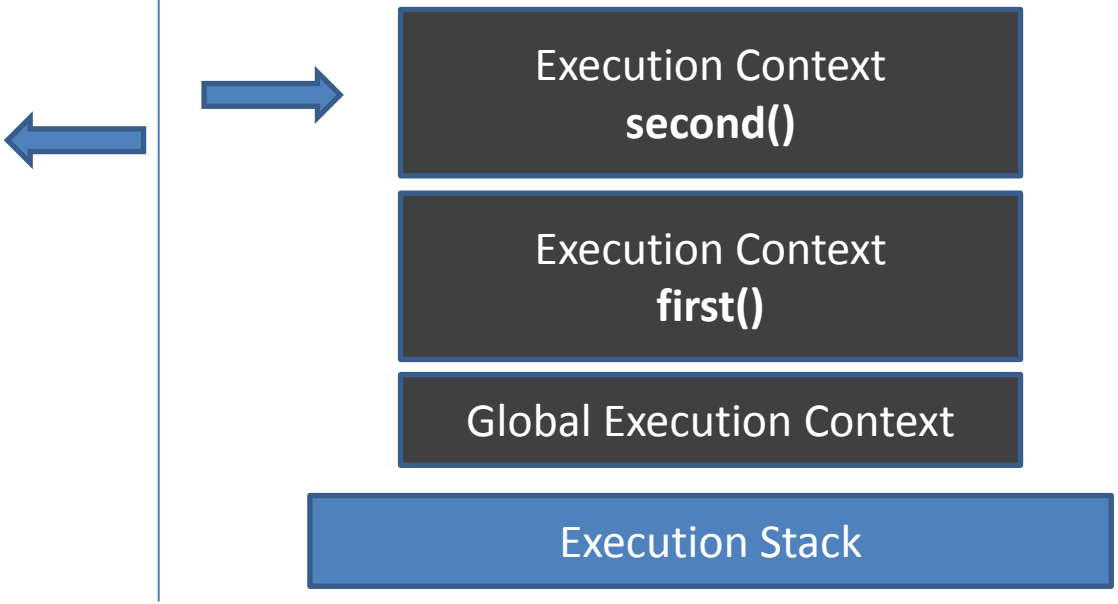
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



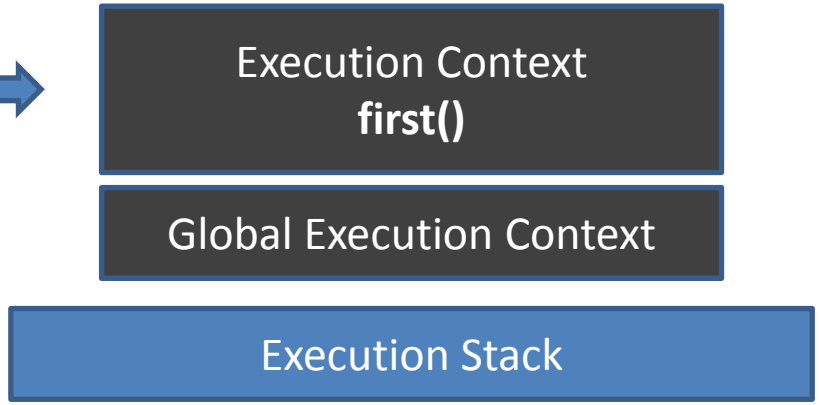
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



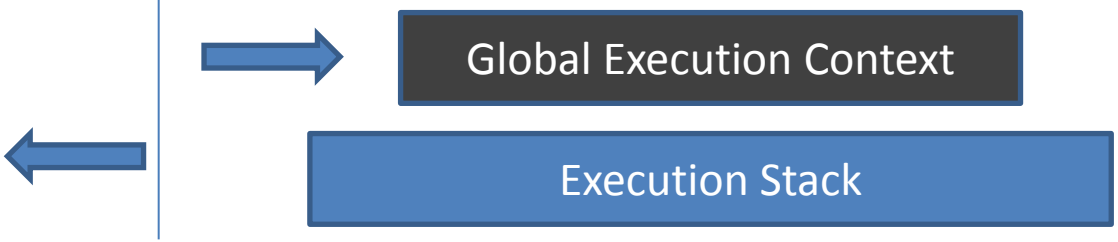
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



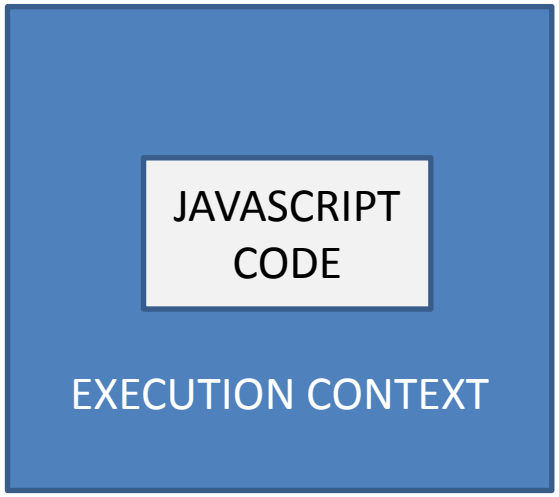
2. Execution contexts and execution stack

```
var name = 'John';

function first() {
    var greeting = 'Hello! ';
    var x = greeting + name;

    second();

    console.log(x);
}

function second() {
    var greeting = 'Hi! ';
    var x = greeting + name;

    third();

    console.log(x);
}

function third() {
    var greeting = 'Hey! ';
    var x = greeting + name;

    console.log(x);
}

first();
```



2. Execution contexts and execution stack

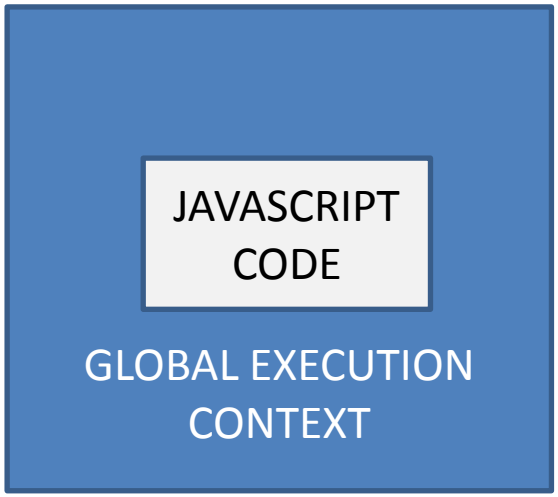
Execution Context is a Box /Container /Wrapper



= OBJECT

2. Execution contexts and execution stack

Global Execution Context

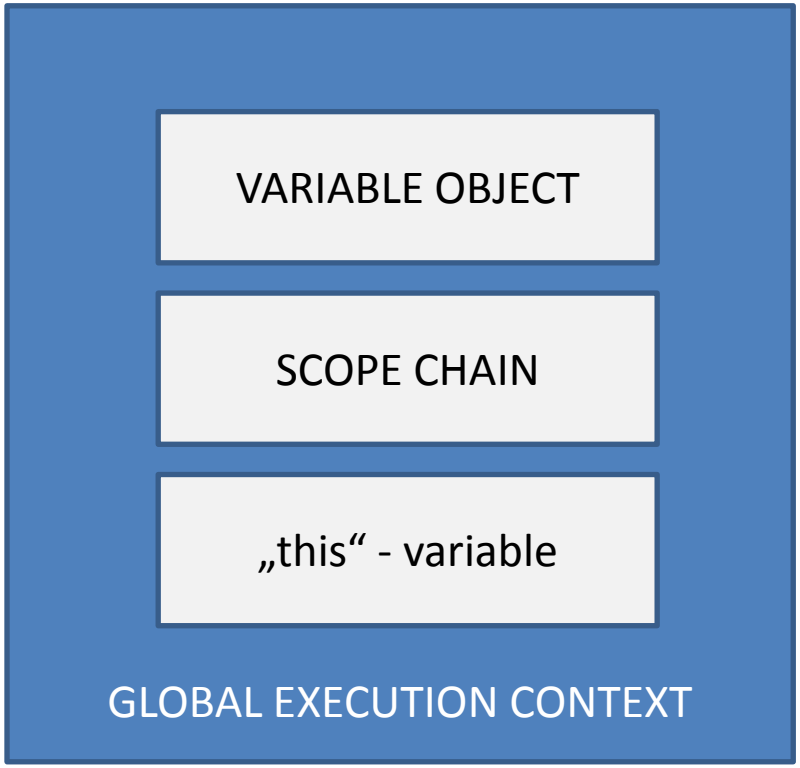


= WINDOW OBJECT

- Code that is not inside any of our functions
- Associated with the global object

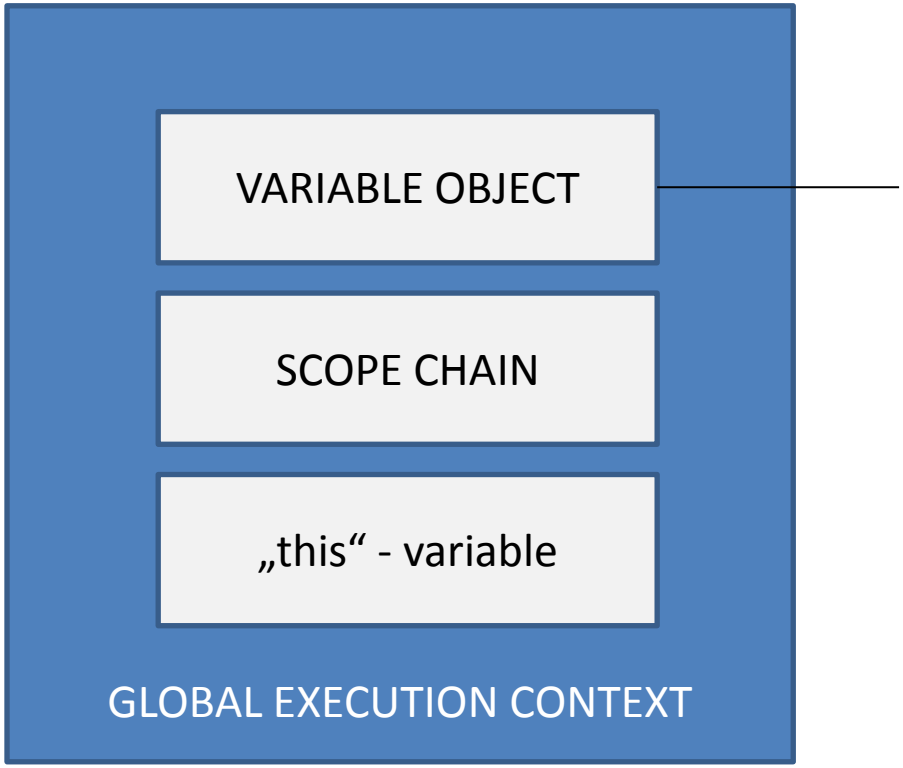
2. Execution contexts and execution stack

Each Execution Context consists of **three Parts** ...



2. Execution contexts and execution stack

Parts of an Execution Context



When e.g. Google Chrome compiles your JS – Code:

HOISTING

Before the code is **executed** line by line, the code is **analyzed** by e.g. Google Chrome which makes

- function declarations
- variable declarations

available!

3. Hoisting

- *„A variable can be declared after it has been used.“*
- ***Before** Execution, the code is analyzed by the JavaScript Parser of declarations of **functions** and **variables***

```
 x = “world“;
var x;
hello( x );

function hello(p) {
   console.log( “hallo “ + p );
}
```

3. Hoisting

- *„A variable can be declared after it has been used.“*
- ***Before** Execution, the code is analyzed by the JavaScript Parser of declarations of **functions** and **variables***

```
 x = “world“;
var x;
hello( x );

function hello(p) {
   console.log( “hallo “ + p );
}
```

***Variable Object** has*

One new property x which is undefined

And

One new function which is hello(p)

3. Hoisting

- *„A v*
  used
- ***Befo***
  Java
  vari

When Chrome executes the code

1. x is set to “world“
2. Hello(x) is called

```
 x = “worl
var x;
hello( x );

function h
    console.log( “hallo “ + p );
}
```

is undefined

And

One new function which is hello(p)

4. Scoping

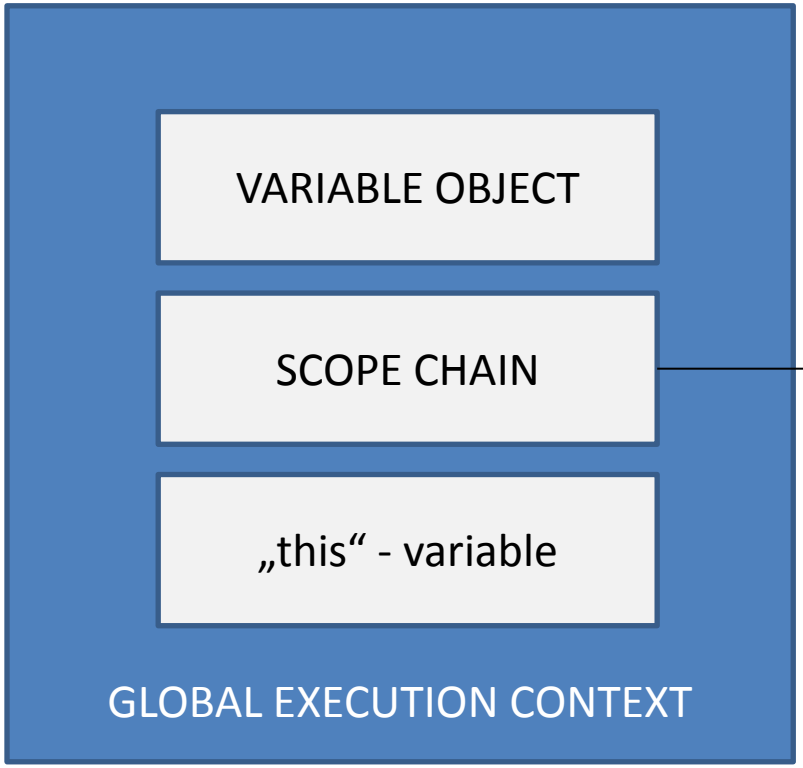
- Scoping answers the question

  „Where can we access a certain variable?“

- Each new function creates a scope
  - The space/environment in which the variables it defines are accessible

4. Scoping

Parts of an Execution Context



Contains Variable Objects of all **parents**

4. Scoping

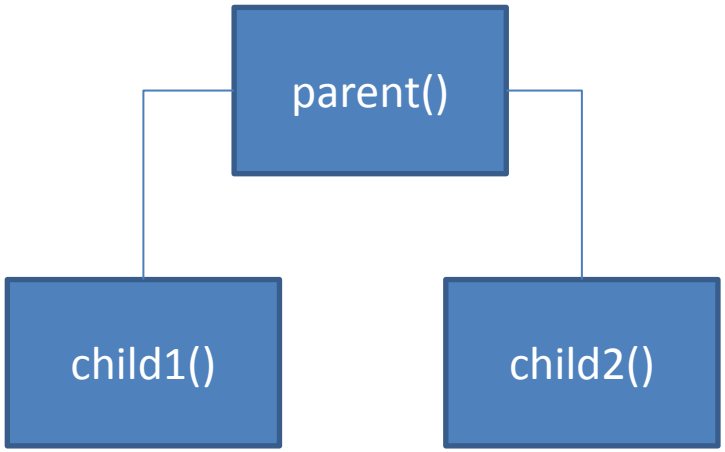
- What is a „parent“ and a „child“

```
var g = 5;

function parent() {
   var x = 1;

   function child1() {
      var z = 1 + x + g;
      console.log(“z is “ + z);
   }

   function child2()  {
      var z = 2 + x + g;
      console.log(“z is “ + z);
   }
}
```

4. Scoping

- What is a „parent“ and a „child“

```
var g = 5;

function parent() {
   var x = 1;

   function child1() {
      var z = 1 + x + g;
      console.log(“z is “ + z);
   }

   function child2()  {
      var z = 2 + x + g;
      console.log(“z is “ + z);
   }
}
```



4. Scoping

- What is a „parent“ and a „child“

```
var g = 5;

function parent() {
   var x = 1;

   function child1() {
      var z = 1 + x + g;
      console.log(“z is “ + z);
   }

   function child2()  {
      var z = 2 + x + g;
      console.log(“z is “ + z);
   }
}
```

GLOBAL SCOPE

4. Scoping

- What is a „parent“ and a „child“

```
var g = 5;

function parent() {
  var x = 1;

  function child1() {
    var z = 1 + x + g;
    console.log(“z is “ + z);
  }

  function child2()  {
    var z = 2 + x + g;
    console.log(“z is “ + z);
  }
}
```

GLOBAL SCOPE

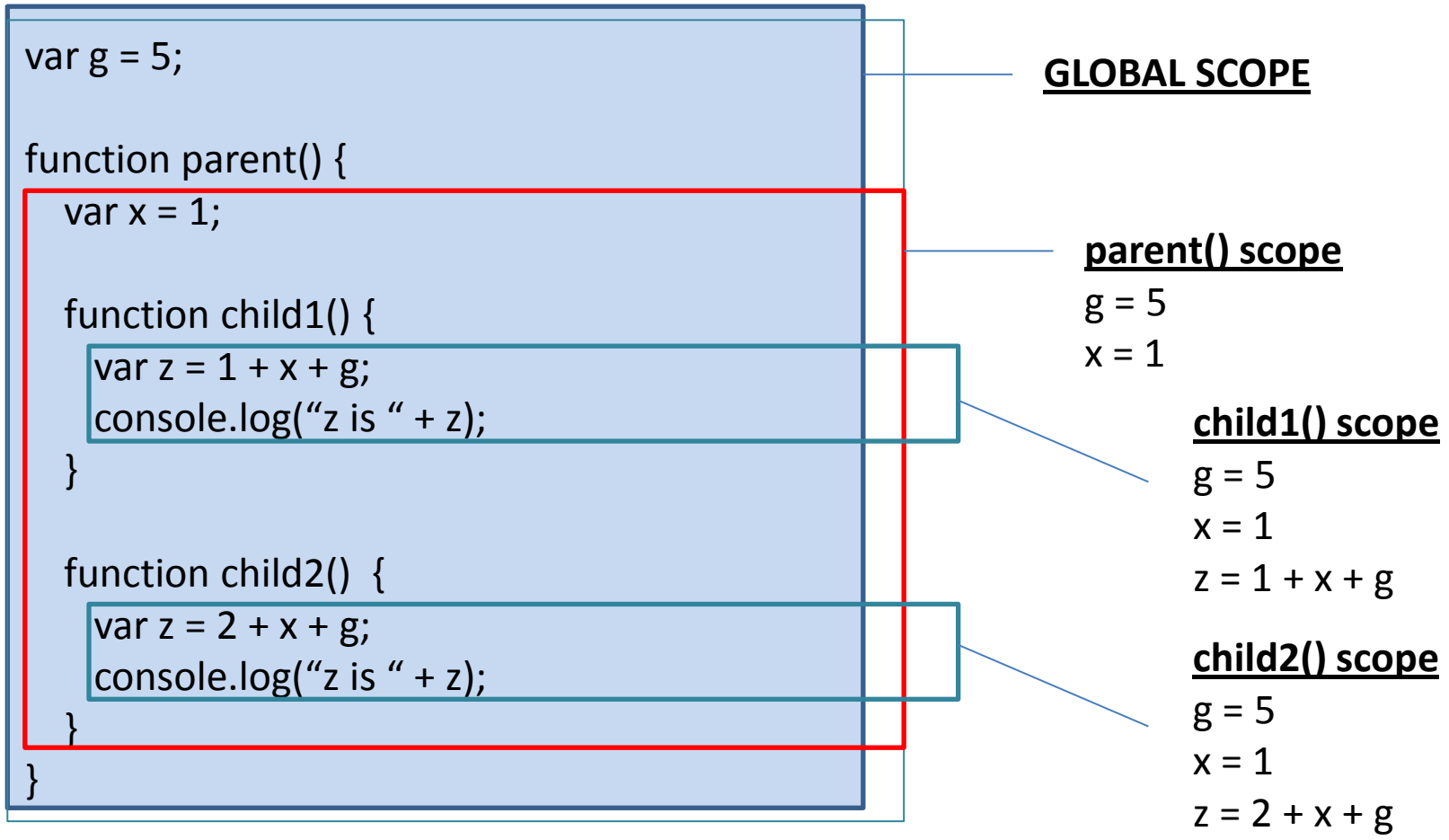
parent() scope

g = 5

x = 1

4. Scoping

- What is a „parent“ and a „child“



5. Scoping VS Execution Stacks

What is the difference between

execution stack

and

scope chain

?

5. Scoping VS Execution Stacks

Execution Stack:

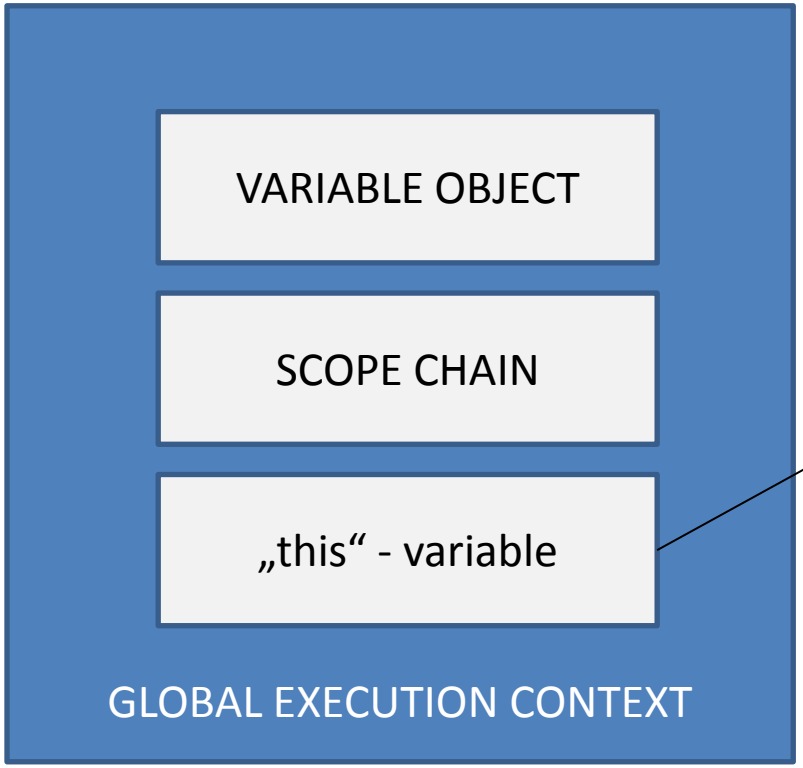
Order in which functions are called.

Scope Chain:

Order in which functions are written lexically.

6. this - Variable

- Each Execution Context has



Function Call
this is the global object (the window object in the browser)

Method Call
this is the points to the object that is calling the method

this *depends on the function/method it is in !*