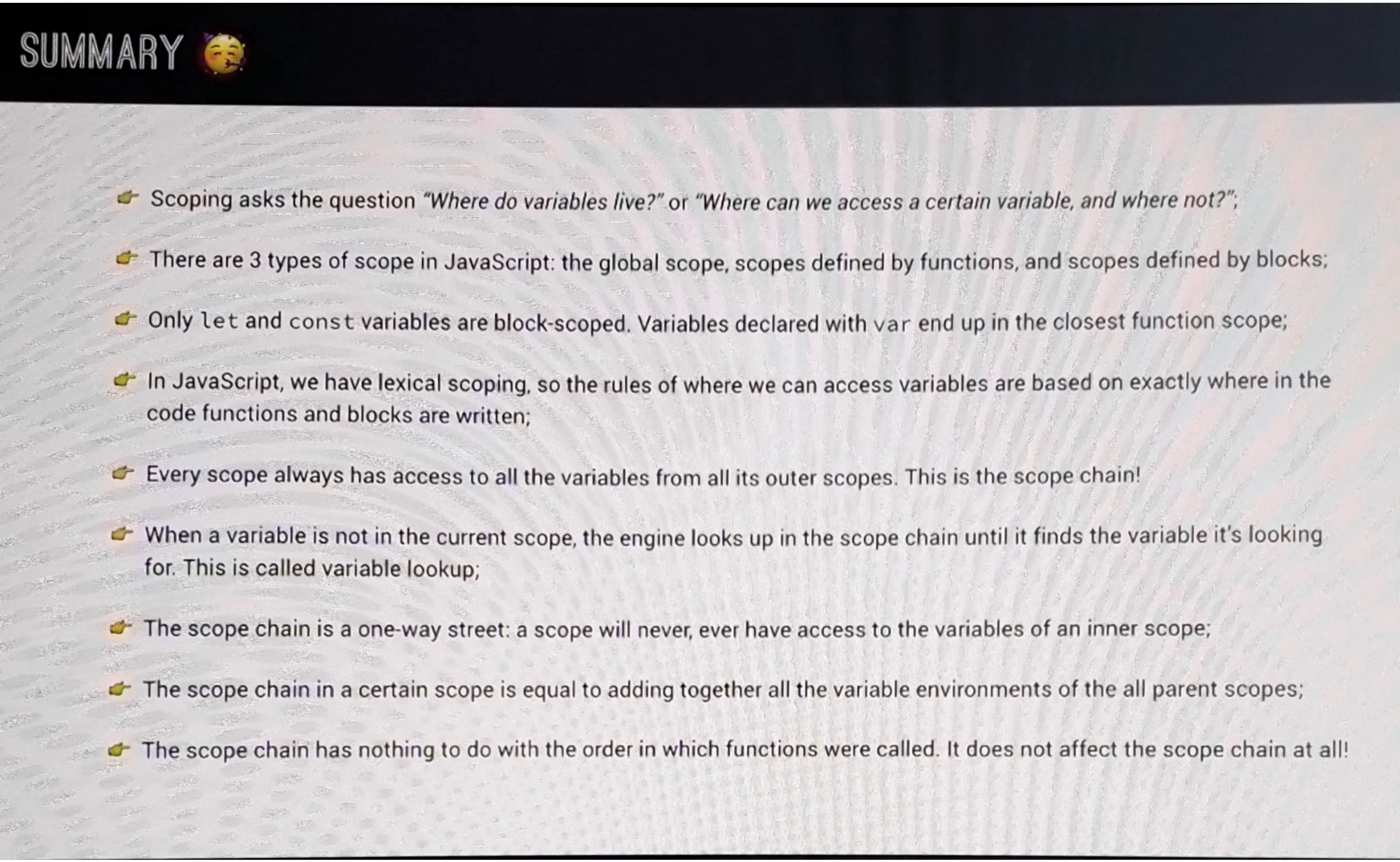
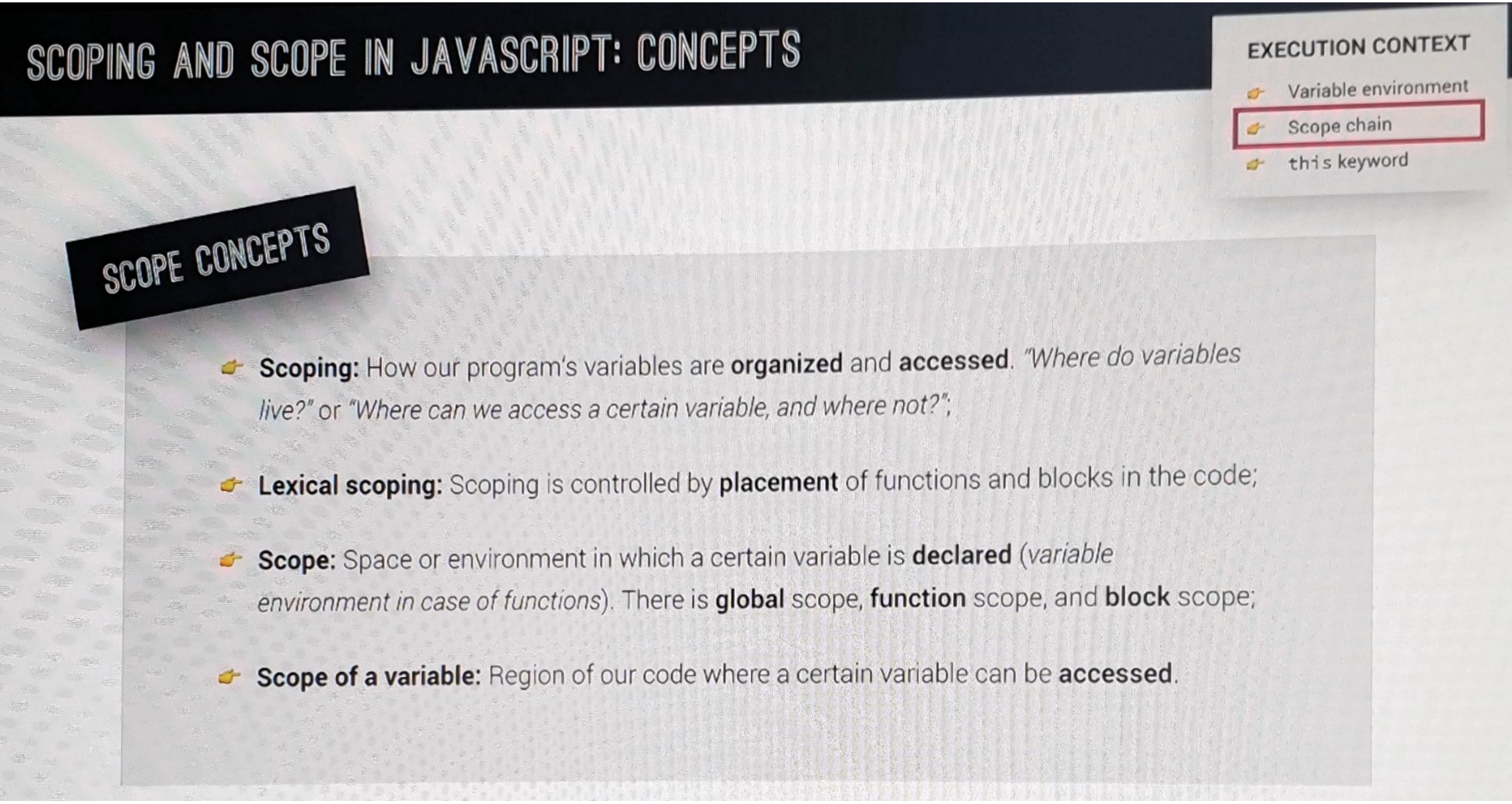


SUMMARY

- Scoping asks the question *“Where do variables live?”* or *“Where can we access a certain variable, and where not?”*;
- There are 3 types of scope in JavaScript: the global scope, scopes defined by functions, and scopes defined by blocks;
- 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 in a certain scope is equal to adding together all the variable environments of the all parent scopes;
- The scope chain has nothing to do with the order in which functions were called. It does not affect the scope chain at all!



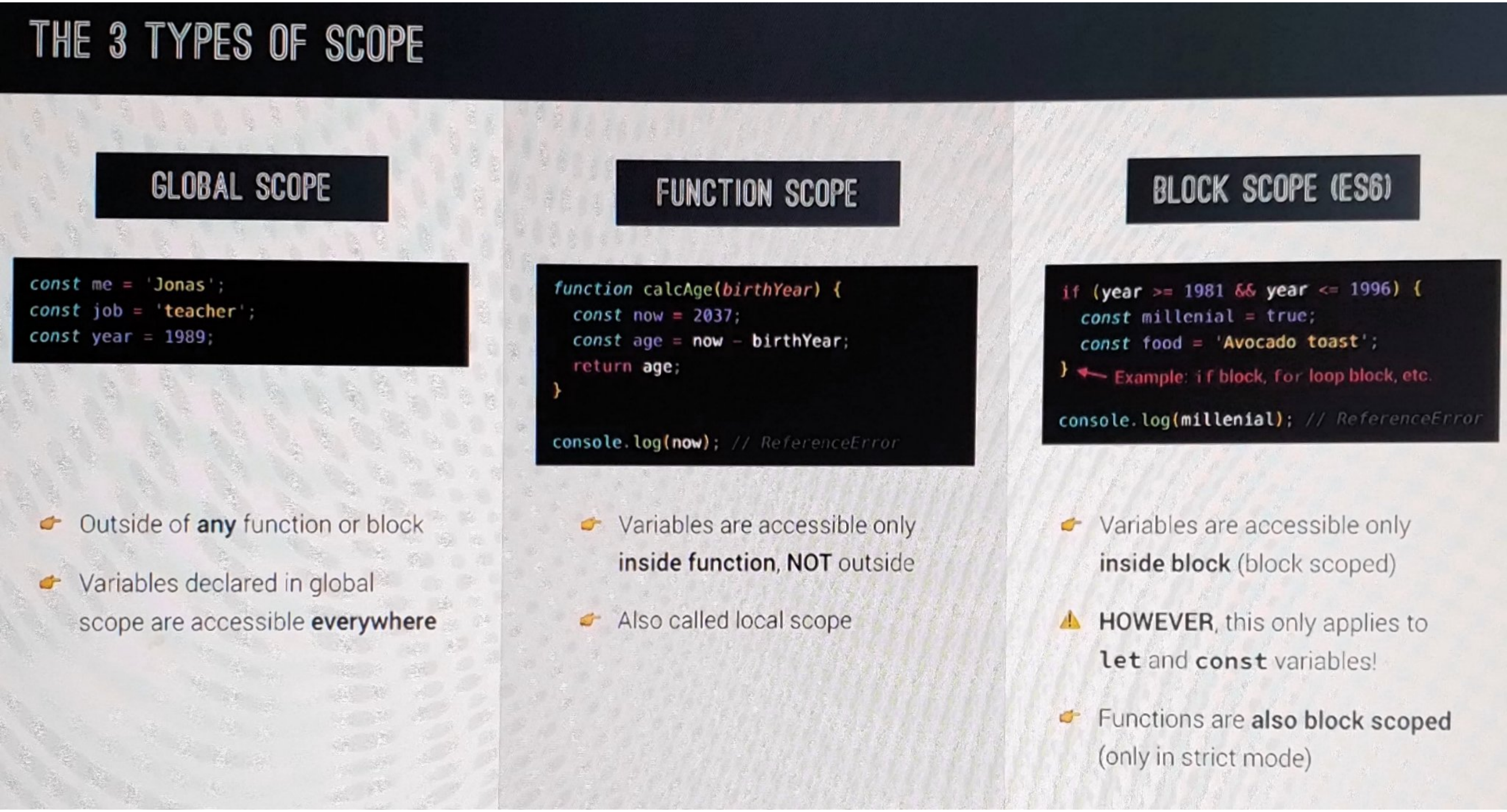
SCOPING AND SCOPE IN JAVASCRIPT: CONCEPTS

EXECUTION CONTEXT

- Variable environment
- Scope chain
- `this` keyword

SCOPE 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

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

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

FUNCTION SCOPE

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

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

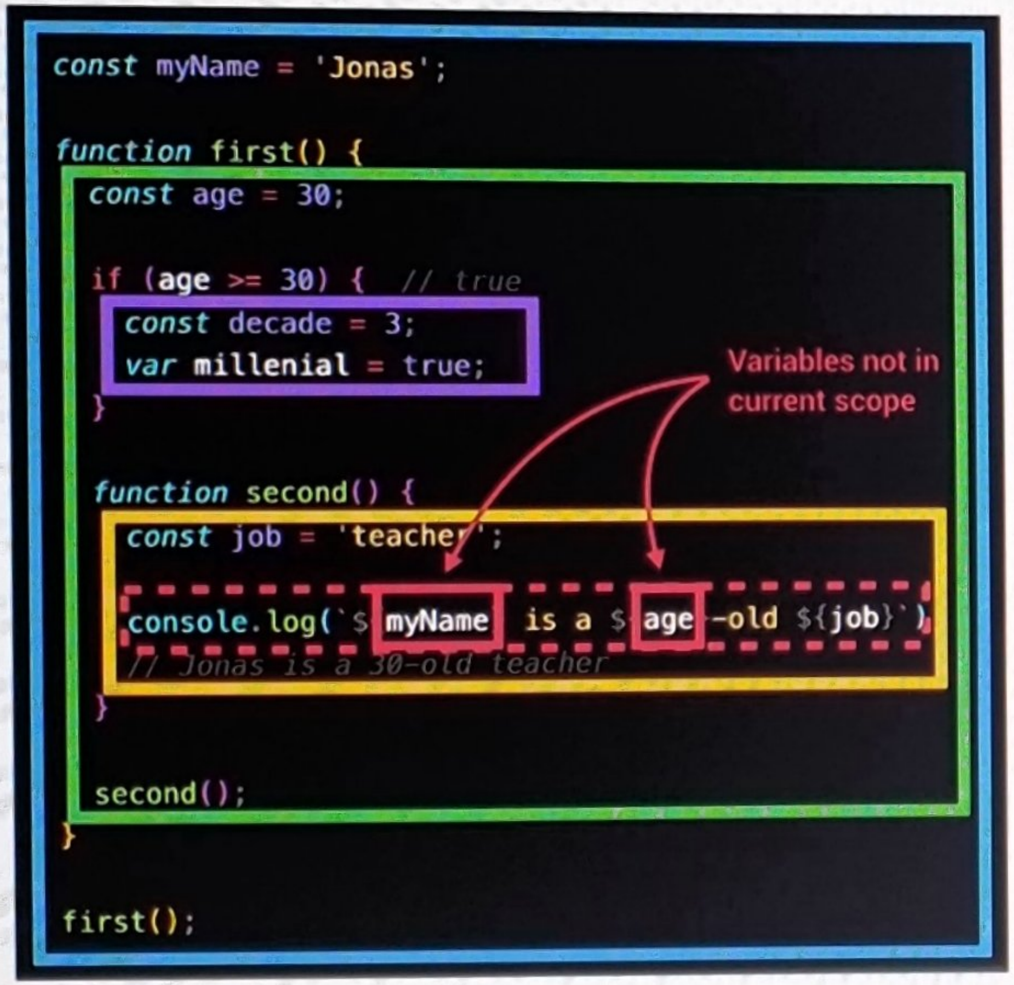
- Variables are accessible only **inside function**, **NOT** outside
- Also called local scope

BLOCK SCOPE (ES6)

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

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

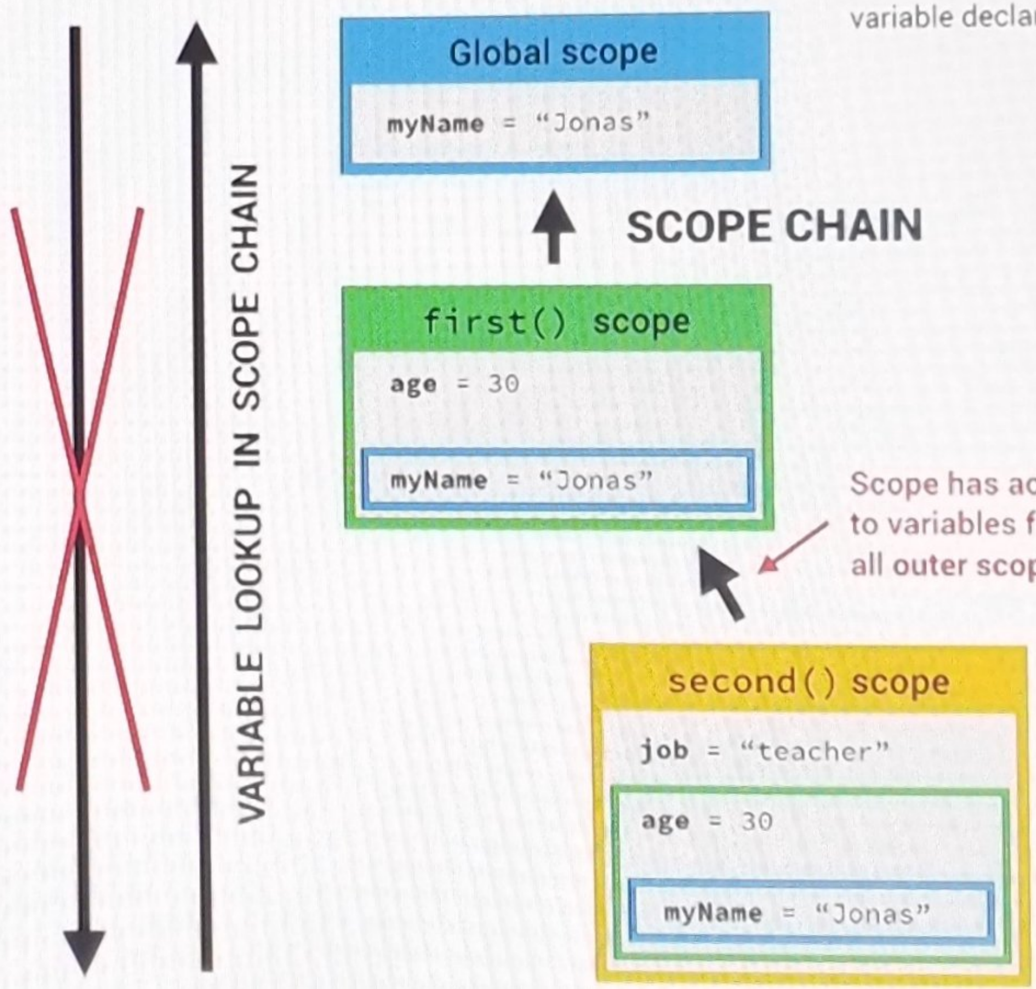
- Variables are accessible only **inside block** (block scoped)
- ⚠ **HOWEVER**, this only applies to **`let`** and **`const`** variables!
- Functions are **also block scoped** (only in strict mode)

THE SCOPE CHAIN

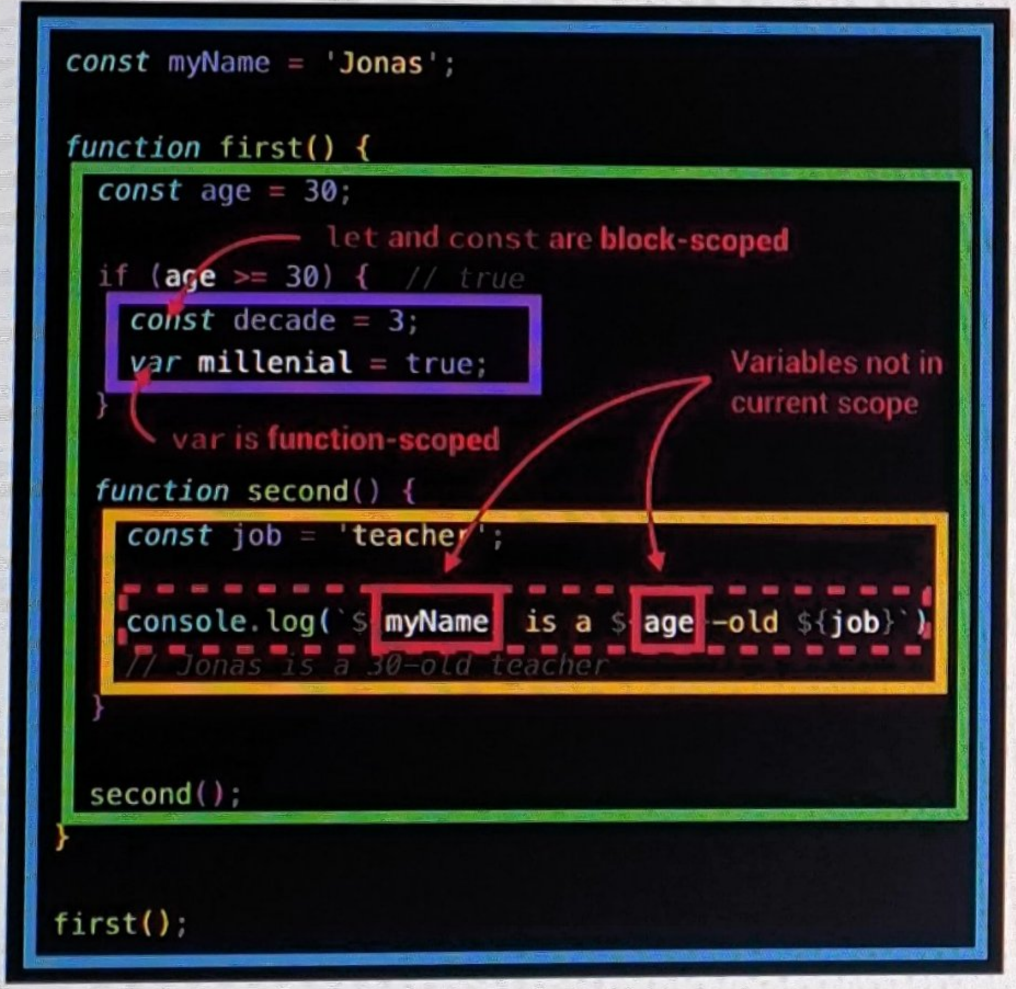


```
const myName = 'Jonas';

function first() {
  const age = 30;

  if (age >= 30) { // true
    const decade = 3;
    var millenial = true;
  }

  function second() {
    const job = 'teacher';

    console.log(`${myName} is a ${age}-old ${job}`);
    // Jonas is a 30-old teacher
  }

  second();
}

first();
```

(Considering only variable declarations)



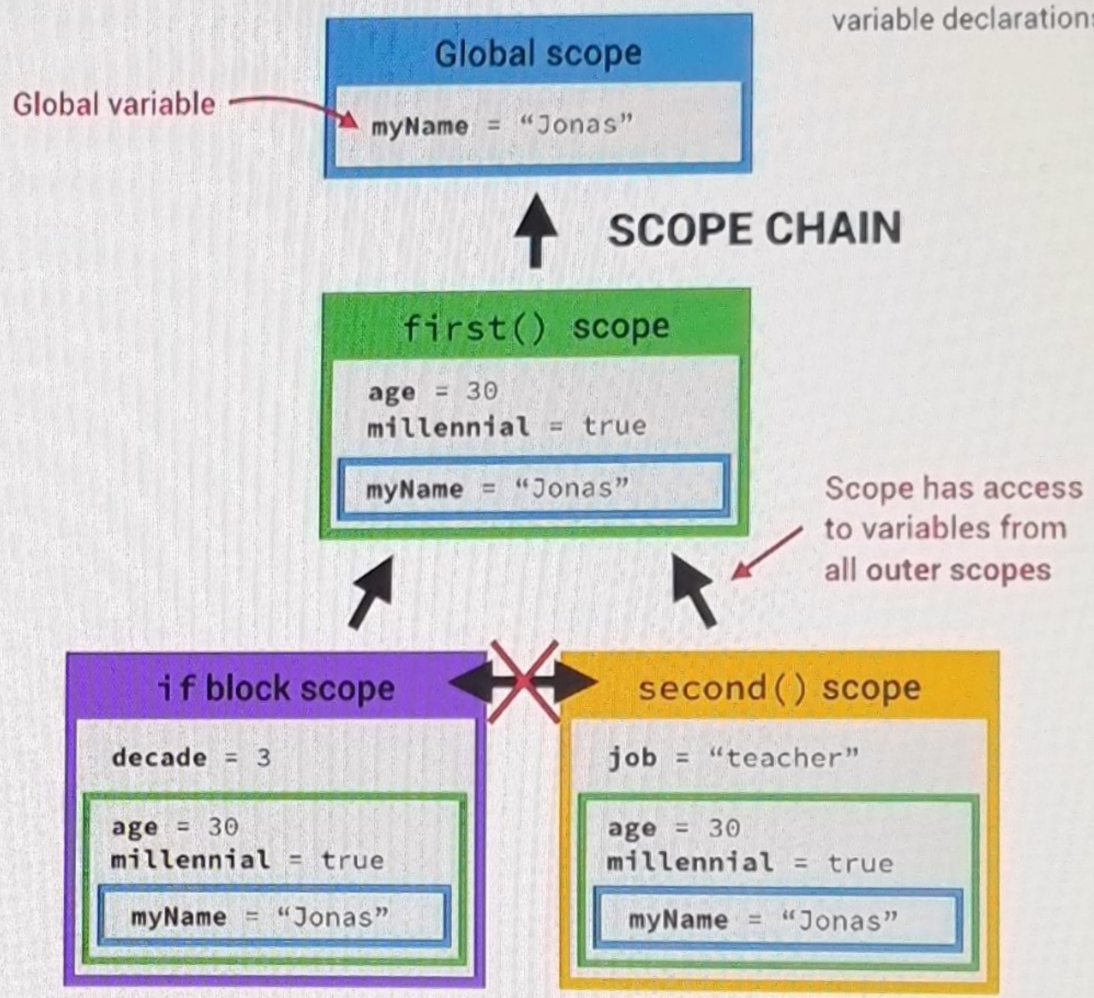
THE SCOPE CHAIN



```
const myName = 'Jonas';

function first() {
  const age = 30;

  if (age >= 30) { // true
    const decade = 3;
    var millenial = true;
  }

  function second() {
    const job = 'teacher';

    console.log(`${myName} is a ${age}-old ${job}`);
    // Jonas is a 30-old teacher
  }

  second();
}

first();
```

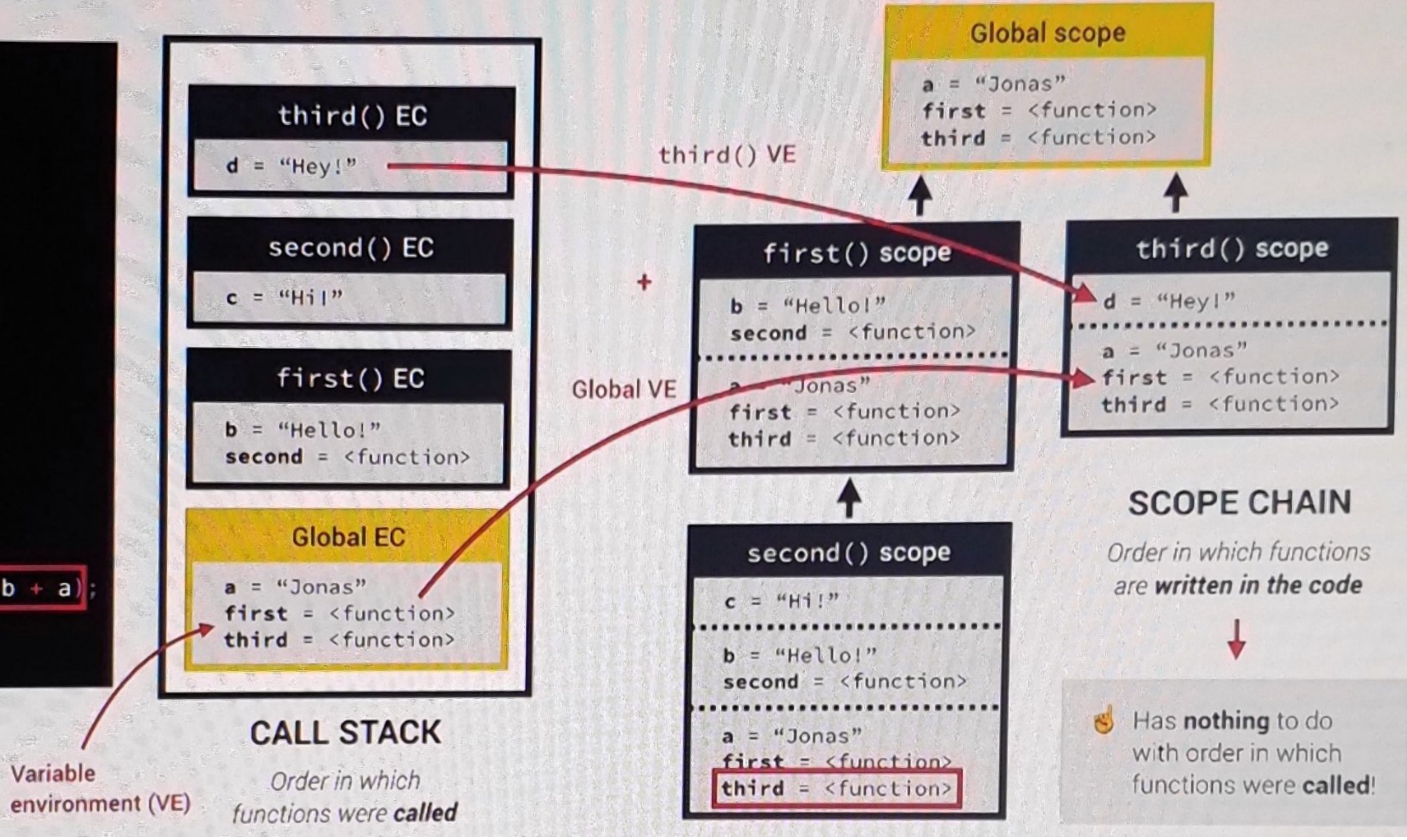
let and const are block-scoped
var is function-scoped
Variables not in current scope

(Considering only variable declarations)



SCOPE CHAIN VS. CALL STACK

```
const a = 'Jonas';
first();

function first() {
  const b = 'Hello!';
  second();

  function second() {
    const c = 'Hi!';
    third();
  }
}

function third() {
  const d = 'Hey!';
  console.log(d + c + b + a);
  // ReferenceError
}
```



third() EC
d = "Hey!"
second() EC
c = "Hi!"
first() EC
b = "Hello!"
second = <function>
Global EC
a = "Jonas"
first = <function>
third = <function>
Variable environment (VE)
CALL STACK
Order in which functions were **called**
third() VE
+
Global VE
Global scope
a = "Jonas"
first = <function>
third = <function>
first() scope
b = "Hello!"
second = <function>
a = "Jonas"
first = <function>
third = <function>
third() scope
d = "Hey!"
a = "Jonas"
first = <function>
third = <function>
second() scope
c = "Hi!"
b = "Hello!"
second = <function>
a = "Jonas"
first = <function>
third = <function>
SCOPE CHAIN
Order in which functions are **written in the code**
Has **nothing** to do with order in which functions were **called**!