Execution Context

JavaScript Execution Context is the environment in which JavaScript code is executed, encompassing the internal workings of the code. It determines the accessibility of functions, variables, and objects used in the code. The execution context involves parsing the code line by line and storing variables and functions in memory. Think of it as a container that stores variables and evaluates and executes the code. Ultimately, the execution context provides the necessary environment for the code to be executed.

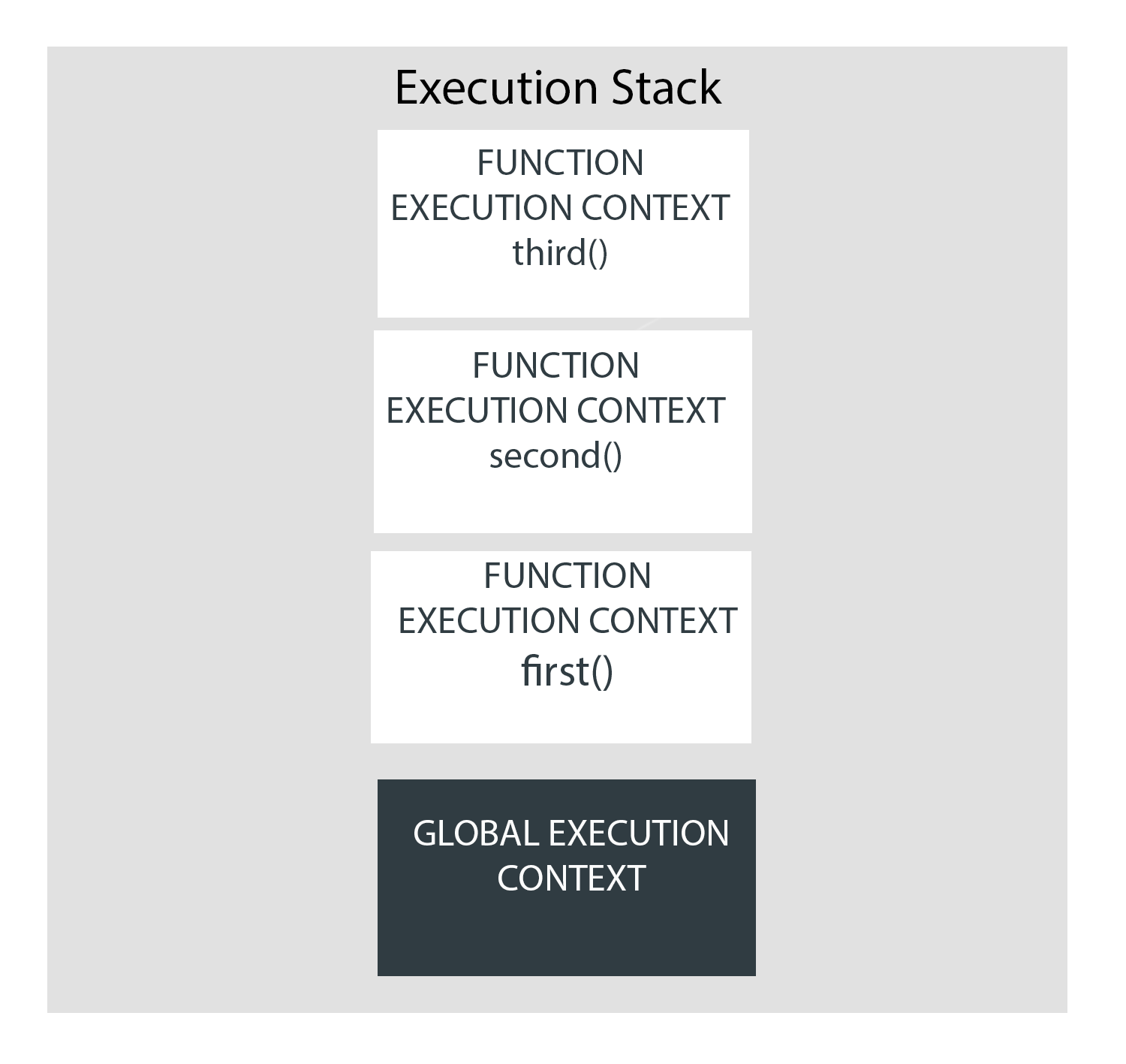
JavaScript has following types of execution contexts: Global Execution Context (GEC), Functional Execution Context (FEC)

When the JavaScript engine receives a script file, it first establishes a default Execution Context called the Global Execution Context (GEC). The GEC serves as the main context where any JavaScript code that is not within a function is executed.

The GEC is the default execution context for code that is not inside any function. It creates a global object (e.g., Window object for browsers) and stores functions and variables in memory. Only one GEC exists per code execution. The GEC is responsible for executing code at the global level in the script. It is unique to each JavaScript file, and only one GEC exists per file.

FEC is created when a function is called. Each function has its own FEC, and multiple FECs can exist. FECs can access code in the GEC but not the other way around, when a function is invoked, the JavaScript engine creates a separate type of Execution Context called a Function Execution Context (FEC) within the Global Execution Context. This FEC is specifically created to evaluate and execute the code within that function.

The execution stack, also known as the call stack, keeps track of all the execution contexts during the script's lifecycle. JavaScript is single-threaded, meaning only one task can be executed at a time A call stack is like a list that an interpreter uses to remember which functions it needs to run and in what order. When a script calls a function, it gets added to the top of the stack, and the interpreter starts running that function. If that function calls other functions, they also get added to the stack and run when their turn comes.



Variable 1

Variable2

Function 1

etc

Parsed code lines

Memory

Code