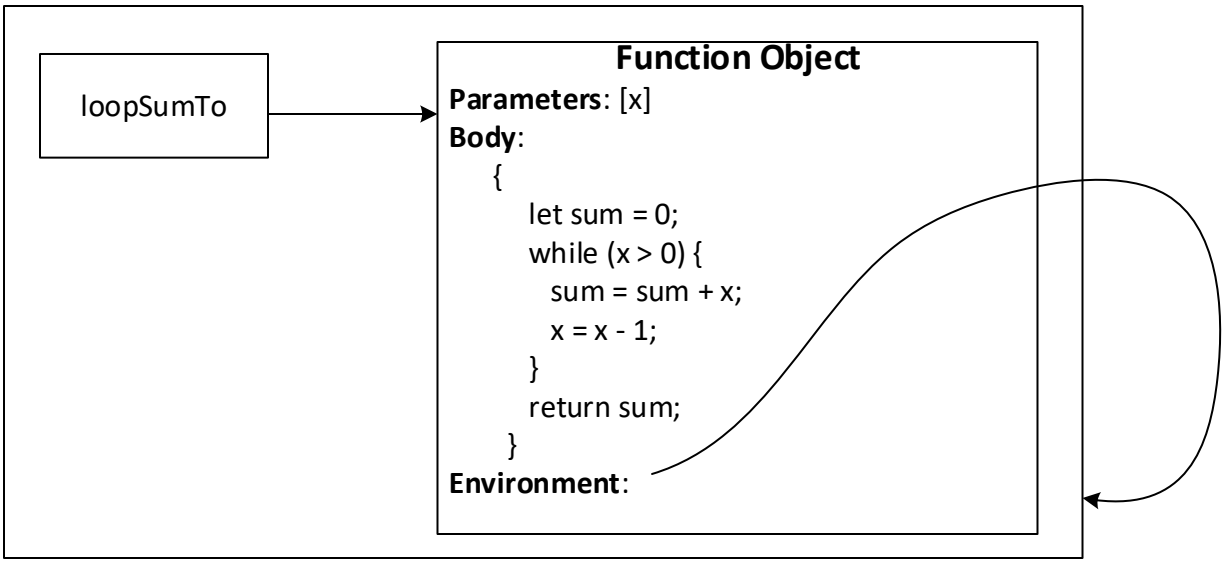
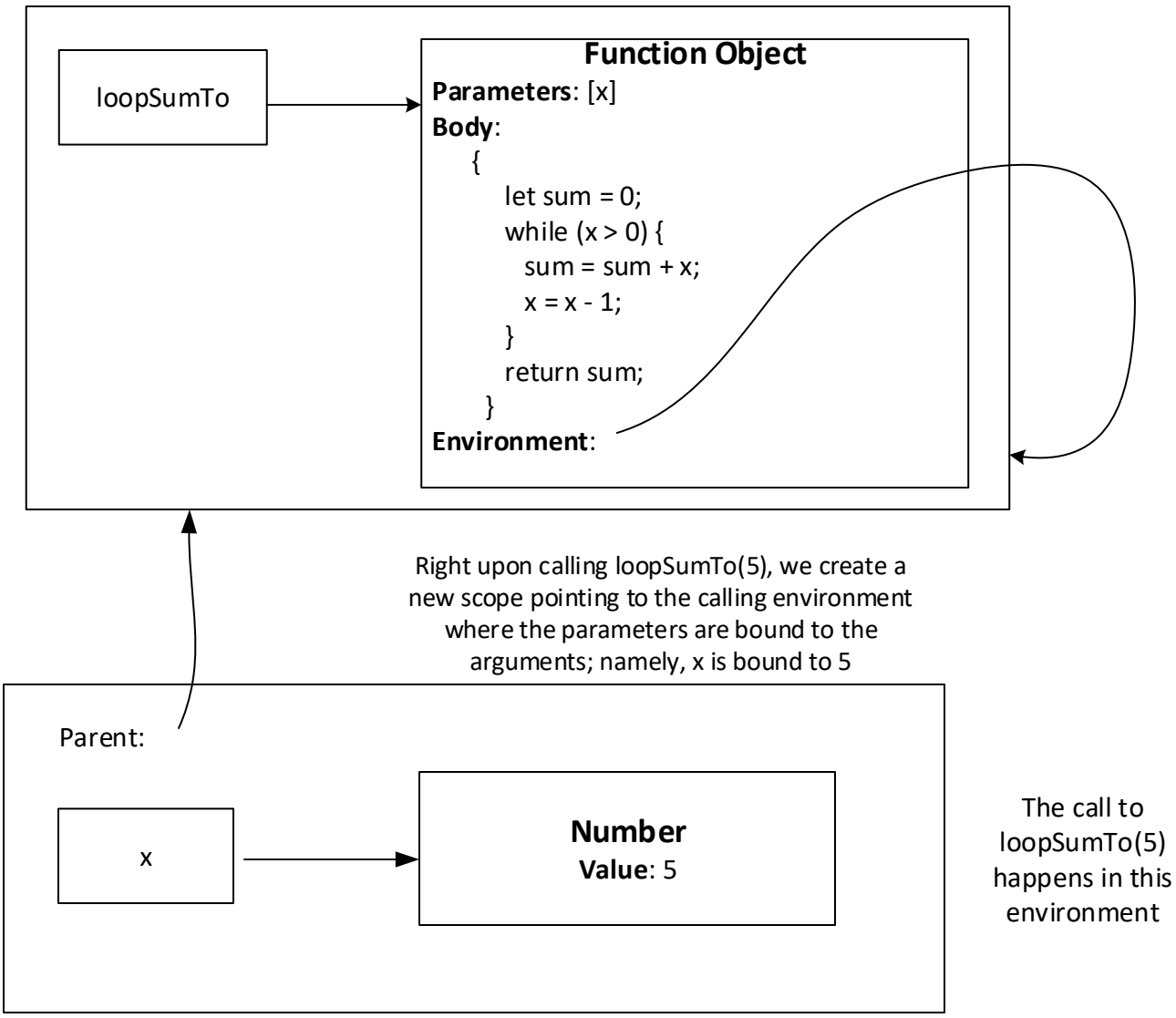
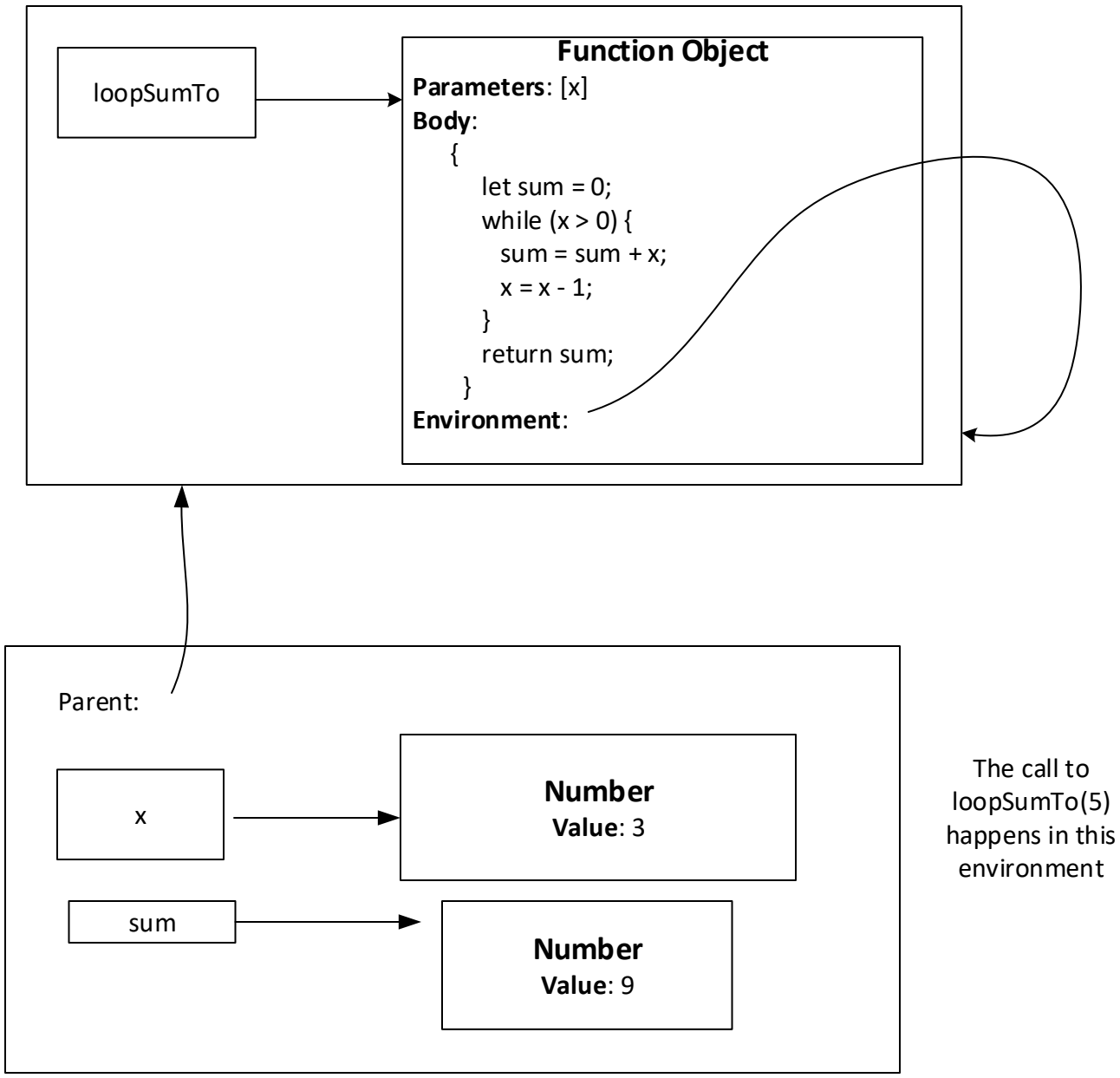

loopSumTo
Function Object
Parameters: [x]
Body:
{
let sum = 0;
while (x > 0) {
sum = sum + x;
x = x - 1;
}
return sum;
}
Environment:

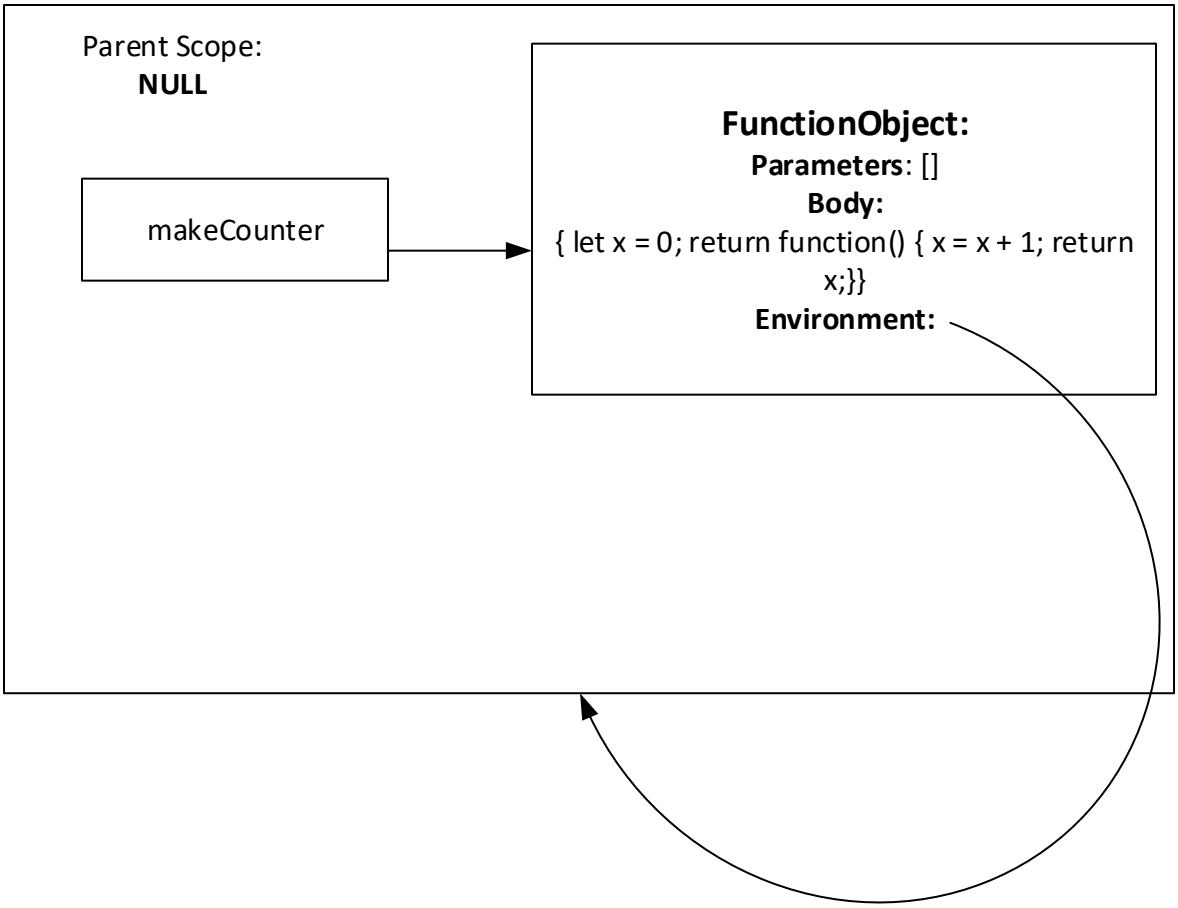

loopSumTo
Function Object
Parameters: [x]
Body:
{
let sum = 0;
while (x > 0) {
sum = sum + x;
x = x - 1;
}
return sum;
}
Environment:
Right upon calling loopSumTo(5), we create a new scope pointing to the calling environment where the parameters are bound to the arguments; namely, x is bound to 5
Parent:
x
Number
Value: 5
The call to loopSumTo(5) happens in this environment



We get recursion “for free” because if loopSumTo were to call loopSumTo, environment lookup would fail in loopSumTo’s scope, and would try all parent environments in order until it resolved the variable “loopSumTo”. It would find it in the initial environment.

After executing:

```
let makeCounter = function() { let x = 0; return
    function() { x = x + 1; return x; }; }
```

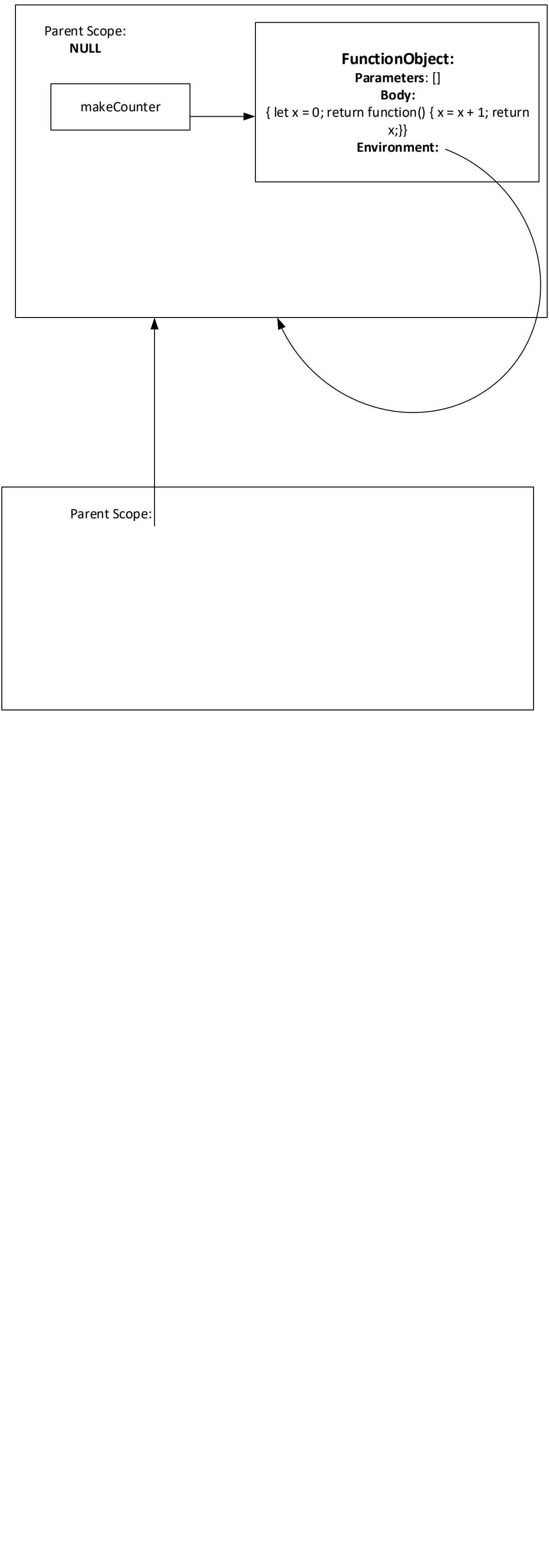


After executing:

let makeCounter = function() { let x = 0; return function() { x = x + 1; return x; };
}

In the process of evaluating the call to makeCounter() in

let counter = makeCounter();



After executing:

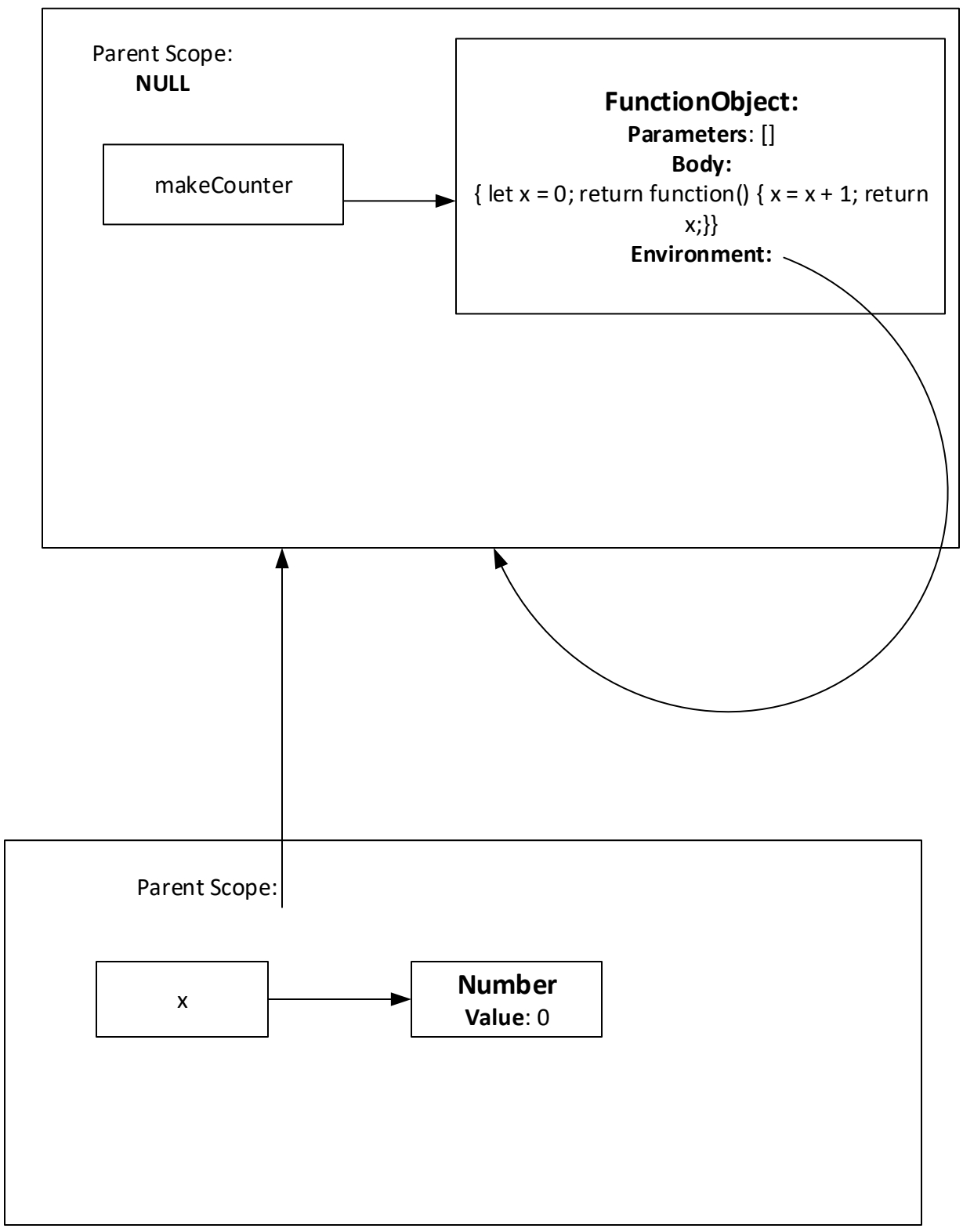
let makeCounter = function() { let x = 0; return function() { x = x + 1; return x; }; }

In the process of evaluating the call to makeCounter() in

let counter = makeCounter();

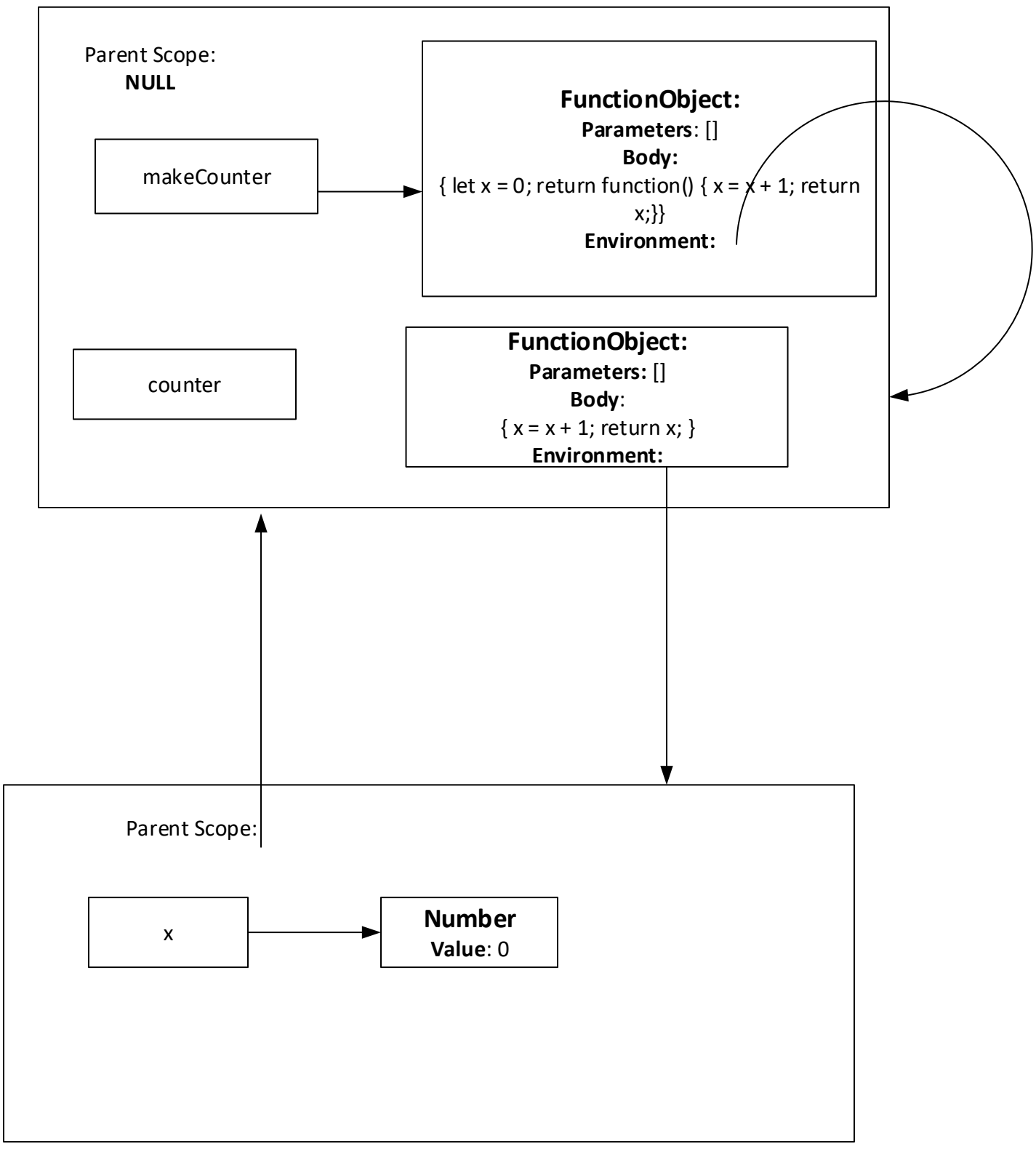
We've executed the first line of the makeCounter function,

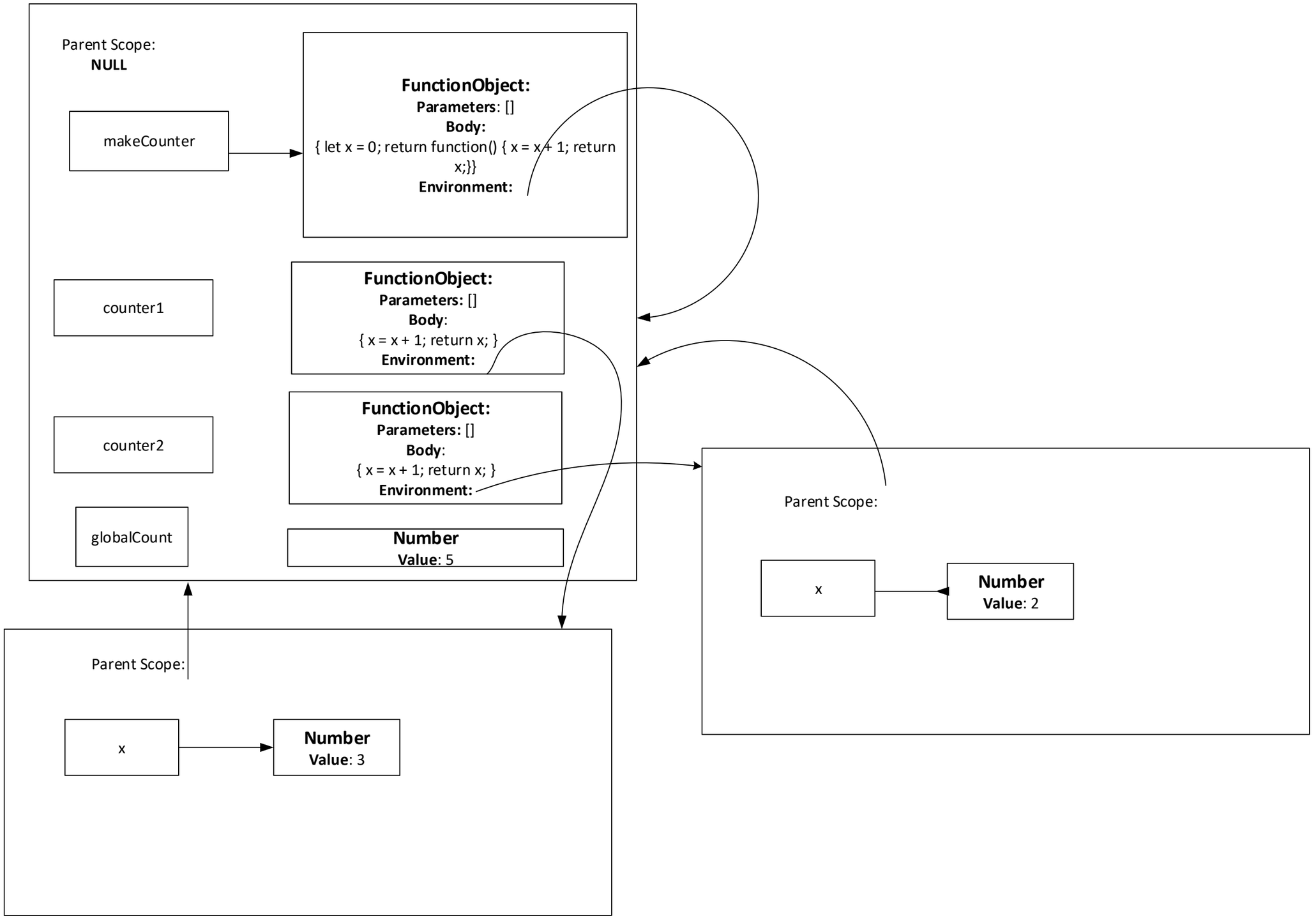
Let x = 0;



After executing:

```
let makeCounter = function() { let x = 0; return function() { x = x + 1; return x; }; }
    let counter = makeCounter();
```




Parent Scope:
NULL
makeCounter
FunctionObject:
Parameters: []
Body:
{ let x = 0; return function() { x = x + 1; return x;}}
Environment:
counter1
FunctionObject:
Parameters: []
Body:
{ x = x + 1; return x; }
Environment:
counter2
FunctionObject:
Parameters: []
Body:
{ x = x + 1; return x; }
Environment:
globalCount
Number
Value: 5
Parent Scope:
x
Number
Value: 3
Parent Scope:
x
Number
Value: 2