L9: Extended Environment Model

SWS3012: Structure and Interpretation of Computer Programs

Martin Henz

July 13, 2023

Outline

- Arrays
- Loops
- Arrays and Loops
- Environments of Arrays and Loops
- Extended Environment Model

Outline

- Arrays
- Loops
- Arrays and Loops
- Environments of Arrays and Loops
- Extended Environment Model

Arrays

- An ***array*** is a data structure that stores a sequence of data elements

```
const seq = [10, 5, 8];  // array of length 3
let my_array = [];       // empty array
```

- ***Array access*** — each data element can be accessed by using the array's name and a *non-negative integer* ***index***
  - The **first element** has index **0**

```
seq[0]; ➔ 10
seq[2]; ➔ 8
```

Arrays

- ***Array assignment*** — each data element can be assigned to with new value

```
seq[0] = 20;
seq[0]; ➔ 20
```

Array Length

- The primitive function **array_length** returns the length of an array

```
array_length(seq); → 3
array_length(my_array); → 0
```

- The length of an array can be increased by assigning to index position beyond the “last element”

```
seq[10] = 99;
seq[10]; → 99
array_length(seq); → 11
```

Array Example

```
const things = [123, "cat", "orange"];
things;        → [123, "cat", "orange"]
array_length(things); → 3
things[0]; → 123
things[2]; → "orange"
things[2] = "apple";
things[2]; → "apple"
things[4] = 456;
array_length(things); → 5
things;        → [123, "cat", "apple", undefined, 456]
things[4]; → 456
things[3]; → undefined
```

Another Array Example

```
let my_array = [];  // creates an empty array

array_length(my_array); ➔ 0

my_array[5] = 100;

my_array; ➔ [undefined, undefined, undefined,
             undefined, undefined, 100]

array_length(my_array); ➔ 6
```

Random Access

- Arrays support ***random access***
  - Any element in an array can be **accessed (read)** in **constant time**
  - Any element in an array can be **assigned (written) to** in **constant time**
    - **Exception**: Assigning to an array element **`A[i]`**, where index **`i`** ≥ **`array_length(A)`**, takes **Θ(`i` – `array_length(A)`)** time

"Two-Dimensional" Array Example

```
let table = [[1,  2,  3,  4],
             [5,  6,  7,  8],
             [9, 10, 11     ]];
```

```
array_length(table); ➔ 3
```

```
table[1][2]; ➔ 7
```

```
array_length(table[0]); ➔ 4
```

```
array_length(table[2]); ➔ 3
```

Processing Arrays — array_1_to_n

```
// array_1_to_n(n) returns an array that
//   contains elements 1 thru n.
function array_1_to_n(n) {
    const a = [];
    function iter(i) {
        if (i < n) {
            a[i] = i + 1;
            iter(i + 1);
        }
    }
    iter(0);
    return a;
}
array_1_to_n(3);  // [1, 2, 3]
```

Show in Playground

Processing Arrays — `map_array`

```
function map_array(f, arr) {
    const len = array_length(arr);
    function iter(i) {
        if (i < len) {
            arr[i] = f(arr[i]);
            iter(i + 1);
        }
    }
    iter(0);
}

const seq = [3, 1, 5];
map_array(x => 2 * x, seq);
seq; // [6, 2, 10];  destructive operation
```

Show in Playground

Outline

`while` Loop

- **Syntax:**

```
    while (expression) {
        statement
    }
```

- Evaluates **condition expression** *`expression`* and if the result is `true`, executes the body *`statement`* of the loop, after which the process **repeats**. The loop **terminates** when the condition expression evaluates to `false`.

Factorial Using `while` Loop

```
function factorial_r(n) {
    return (n === 1) ? 1 : n * factorial_r(n - 1);
}
```

```
function factorial_i(n) {
    function f(acc, k) {
        if (k <= n) {
            return f(acc * k,
                     k + 1);
        } else {
            return acc;
        }
    }
    return f(1, 1);
}
```

```
function factorial_w(n) {
    let acc = 1;
    let k = 1;
    while (k <= n) {
        acc = acc * k;
        k = k + 1;
    }
    return acc;
}
```

Show in Playground

for Loop

- **Syntax:**

```
for (stmt1; expression; assignment) {
    statement
}
```

- **Equivalent to**

```
{
    stmt1;
    while (expression) {
        statement
        assignment;
    }
}
```

Note:
This is only a simplified translation/view of the **for**-loop.

For accurate description, please refer to the Source §3 specifications.

Environment model for **for**-loop will not be in assessments.

`for` Loop

- **Syntax:**

```
for (stmt1; expression; assignment) {
    statement
}
```

- `stmt1;` can only be
  - an **assignment statement** or
  - a **variable declaration statement** (e.g. `let x = 1;`)
    - The variable is called a ***loop control variable***

Restrictions on Loops in Source §3

- The declared **loop control variable** for a `for` loop cannot be assigned to in the body
- All components in the header of a `for` loop are non-optional
  - For example, `for (;;) {...}` is not allowed

Factorial Using `for` Loop

```
function factorial_f(n) {
    let acc = 1;
    for (let k = 1; k <= n; k = k + 1) {
        acc = acc * k;
    }
    return acc;
}
```

```
function factorial_w(n) {
    let acc = 1;
    let k = 1;
    while (k <= n) {
        acc = acc * k;
        k = k + 1;
    }
    return acc;
}
```

Show in Playground

List Length

```
function list_length(xs) {
    return is_null(xs) ? 0 : 1 + list_length(tail(xs));
}
```

```
function list_length_loop(xs) {
    let count = 0;
    for (let p = xs; !is_null(p); p = tail(p)) {
        count = count + 1;
    }
    return count;
}
```

Show in Playground

The **break;** Statement

- **break;** terminates the current execution of the loop and also terminates the entire loop

```
for (let i = 1; i < 5; i = i + 1) {
    display(stringify(i) + " here");
    if (i === 2) {
        break;
    }
    display(stringify(i) + " there");
}
display("OK");
```

```
Output:
"1 here"
"1 there"
"2 here"
"OK"
```

Show in Playground

The **continue;** Statement

- `continue;` terminates the **current** execution of the loop and continues with the loop

```
for (let i = 1; i < 5; i = i + 1) {
    display(stringify(i) + " here");
    if (i === 2) {
        continue;
    }
    display(stringify(i) + " there");
}
display("OK");
```

Output:

```
"1 here"
"1 there"
"2 here"
"3 here"
"3 there"
"4 here"
"4 there"
"OK"
```

Show in Playground

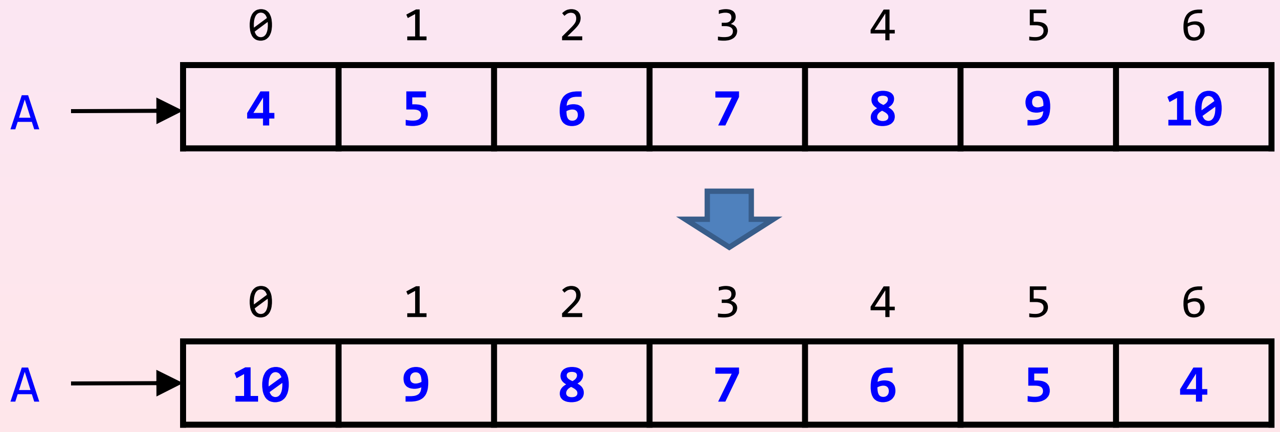
Outline

- Arrays
- Loops
- Arrays and Loops
- Environments of Arrays and Loops
- Extended Environment Model

Loops and Arrays — `reverse_array`

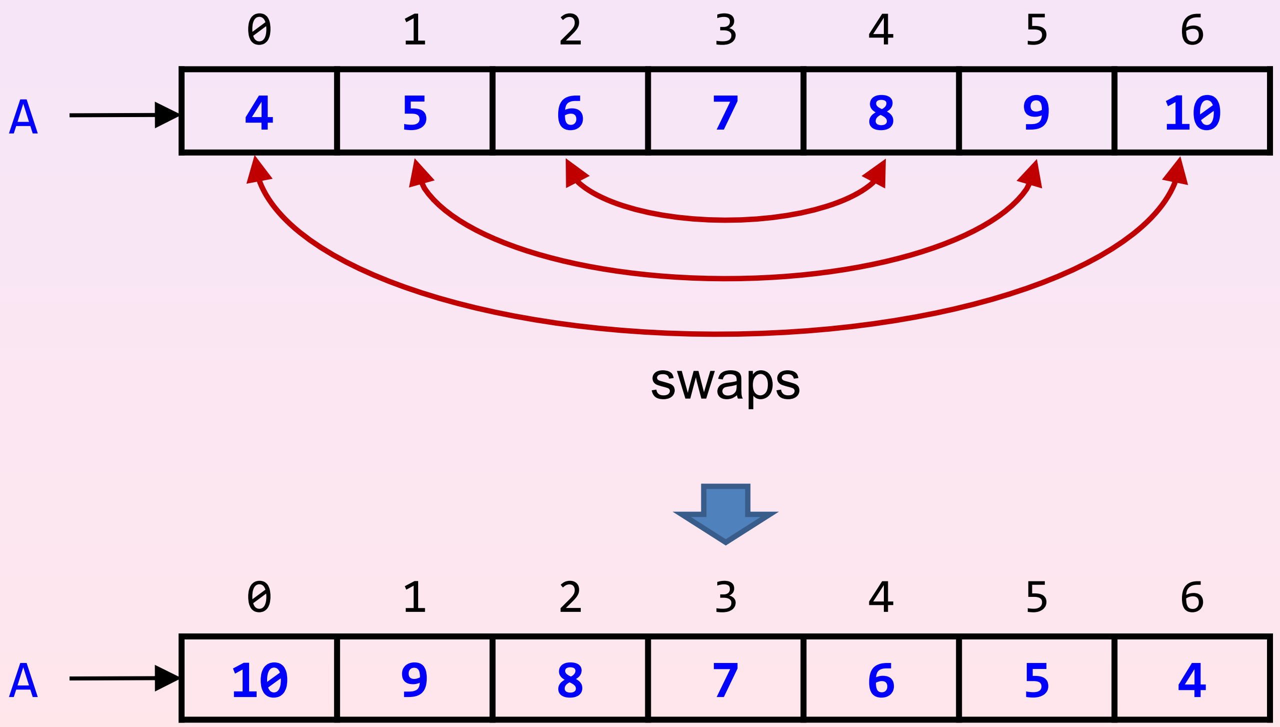
- **Wanted:** A `reverse_array` function to reverse the input array
- **Example:**

```
const A = [4, 5, 6, 7, 8, 9, 10];
reverse_array(A);
A; ➔ [10, 9, 8, 7, 6, 5, 4]
```



Loops and Arrays — `reverse_array`

- **How to reverse?**



Loops and Arrays — `reverse_array (Attempt #1)`

- **Attempt #1:**

```
function swap(x, y) {
    let temp = x;
    x = y;
    y = temp;
}

function reverse_array(A) {
    const len = array_length(A);
    const half_len = math_floor(len / 2);
    for (let i = 0; i < half_len; i = i + 1) {
        swap(A[i], A[len - 1 - i]);
    }
}
```

Show in Playground

Loops and Arrays — `reverse_array (Attempt #1)`

- **Testing:**

```
const A = [4, 5, 6, 7, 8, 9, 10];
reverse_array(A);
A; ➔ [4, 5, 6, 7, 8, 9, 10]
```

- **What is wrong?**

Loops and Arrays — `reverse_array (Attempt #2)`

- **Attempt #2:**

```
function swap(A, i, j) {
    let temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}

function reverse_array(A) {
    const len = array_length(A);
    const half_len = math_floor(len / 2);
    for (let i = 0; i < half_len; i = i + 1) {
        swap(A, i, len - 1 - i);
    }
}
```

Show in Playground

Loops and Arrays — `zero_matrix`

```
// Returns a 2D array that represents
//   a rows x cols zero matrix.
function zero_matrix(rows, cols) {
    const M = [];
    for (let r = 0; r < rows; r = r + 1) {
        M[r] = [];
        for (let c = 0; c < cols; c = c + 1) {
            M[r][c] = 0;
        }
    }
    return M;
}

const mat3x4 = zero_matrix(3, 4);
```

Show in Playground

Loops and Arrays — `matrix_multiply_3x3`

```
// Returns a 2D array represents the results
//   of multiplying two 3x3 matrices.
function matrix_multiply_3x3(A, B) {
    const M = [];
    for (let r = 0; r < 3; r = r + 1) {
        M[r] = [];
        for (let c = 0; c < 3; c = c + 1) {
            M[r][c] = 0;
            for (let k = 0; k < 3; k = k + 1) {
                M[r][c] = M[r][c] + A[r][k] * B[k][c];
            }
        }
    }
    return M;
}
```

Show in Playground

$$\begin{bmatrix} m_{0,0} & m_{0,1} & m_{0,2} \\ m_{1,0} & m_{1,1} & m_{1,2} \\ m_{2,0} & m_{2,1} & m_{2,2} \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} * \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix}$$

Outline

- Arrays
- Loops
- Arrays and Loops
- Environments of Arrays and Loops
- Extended Environment Model

`while` Loop

- **Syntax:**

```
while (expression) {
    statement
}
```

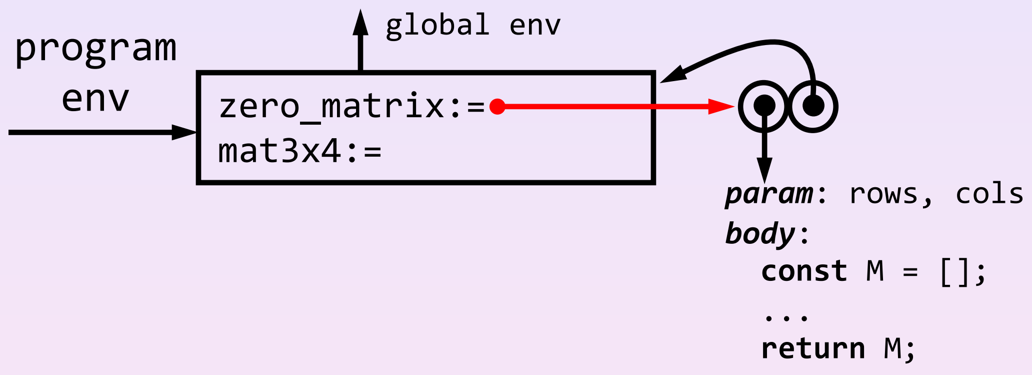
- The **loop body** is in a **new block** (`{ statement }`)

- ***Every time*** when the **body block** is evaluated, it extends the environment by adding a **new frame**
    - **No new frame** is created if the block has **no constant & variable declaration**

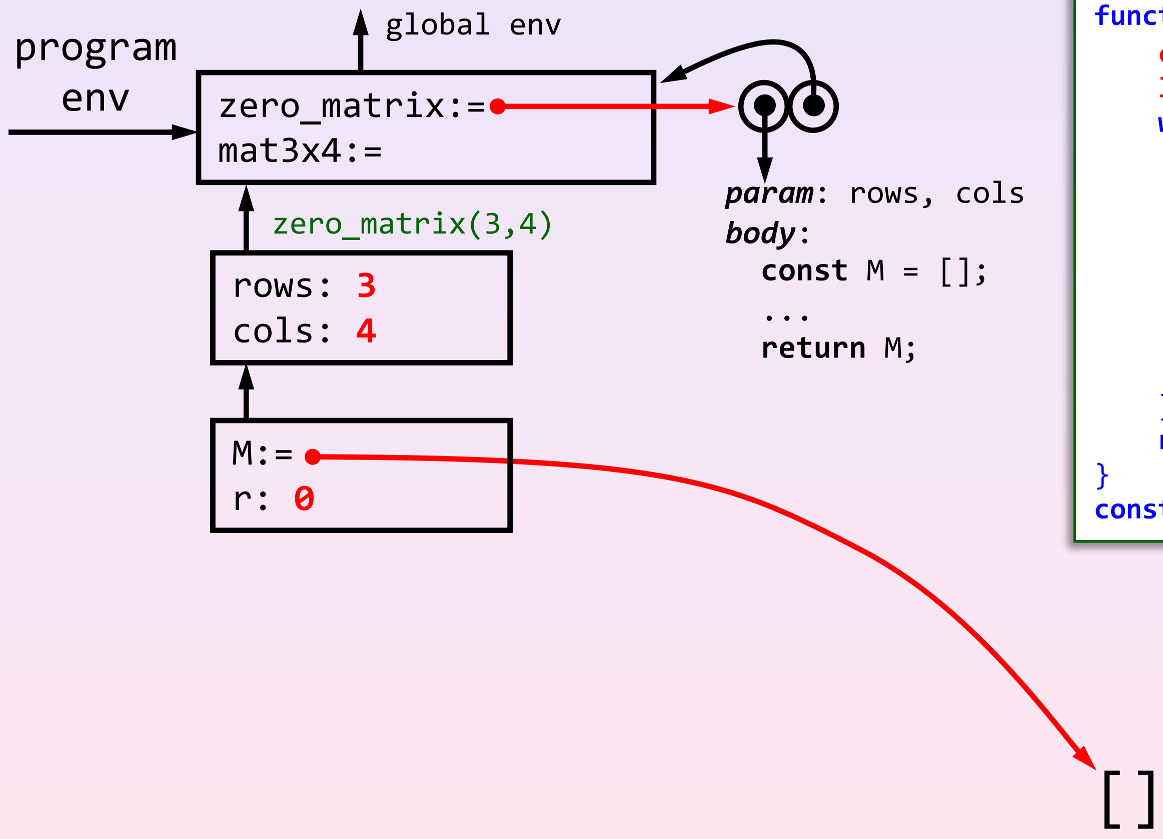
Environments of Loops and Arrays: Example

```
// Using while loops
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}

const mat3x4 = zero_matrix(3, 4);
```

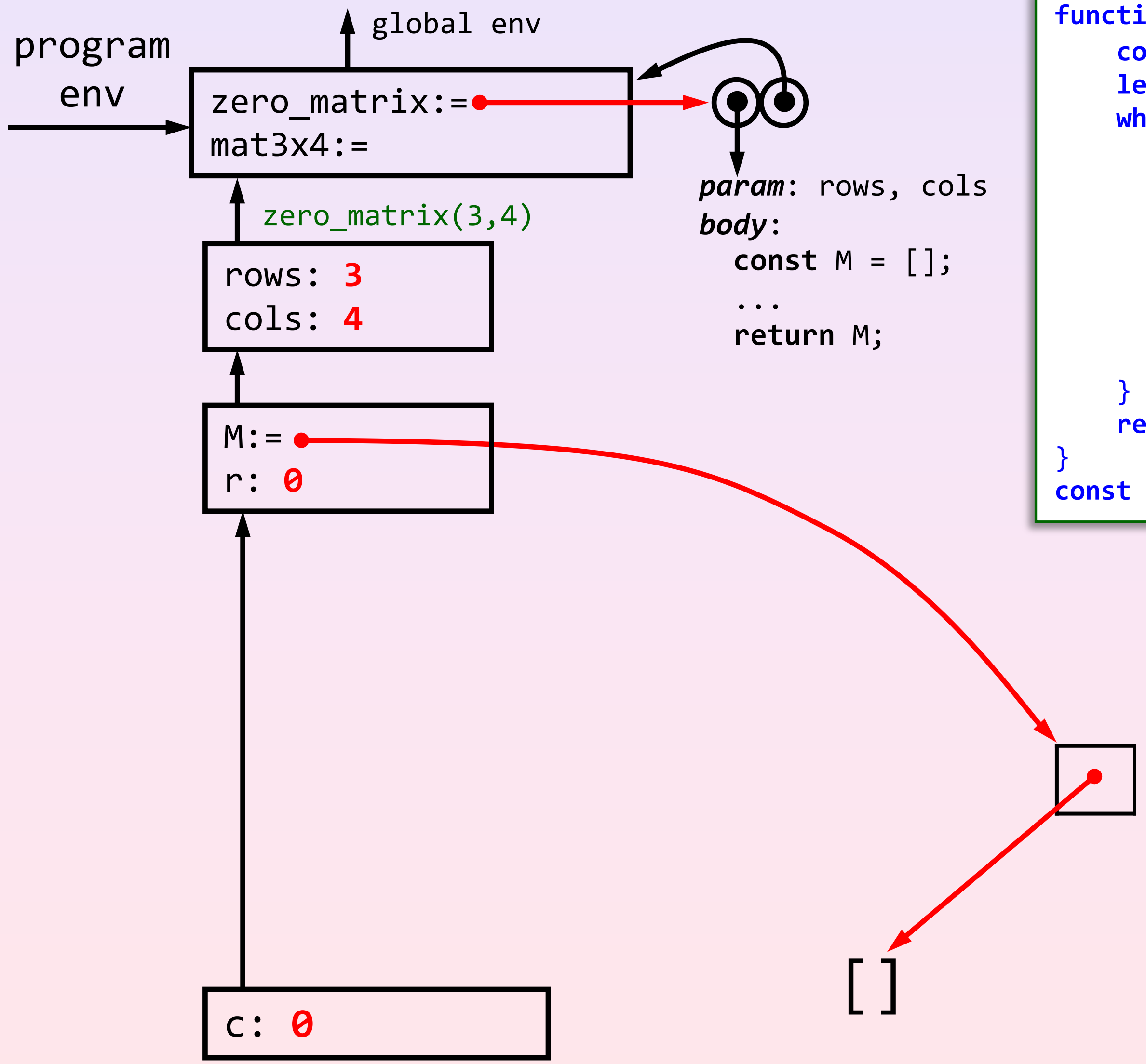
Show in Playground



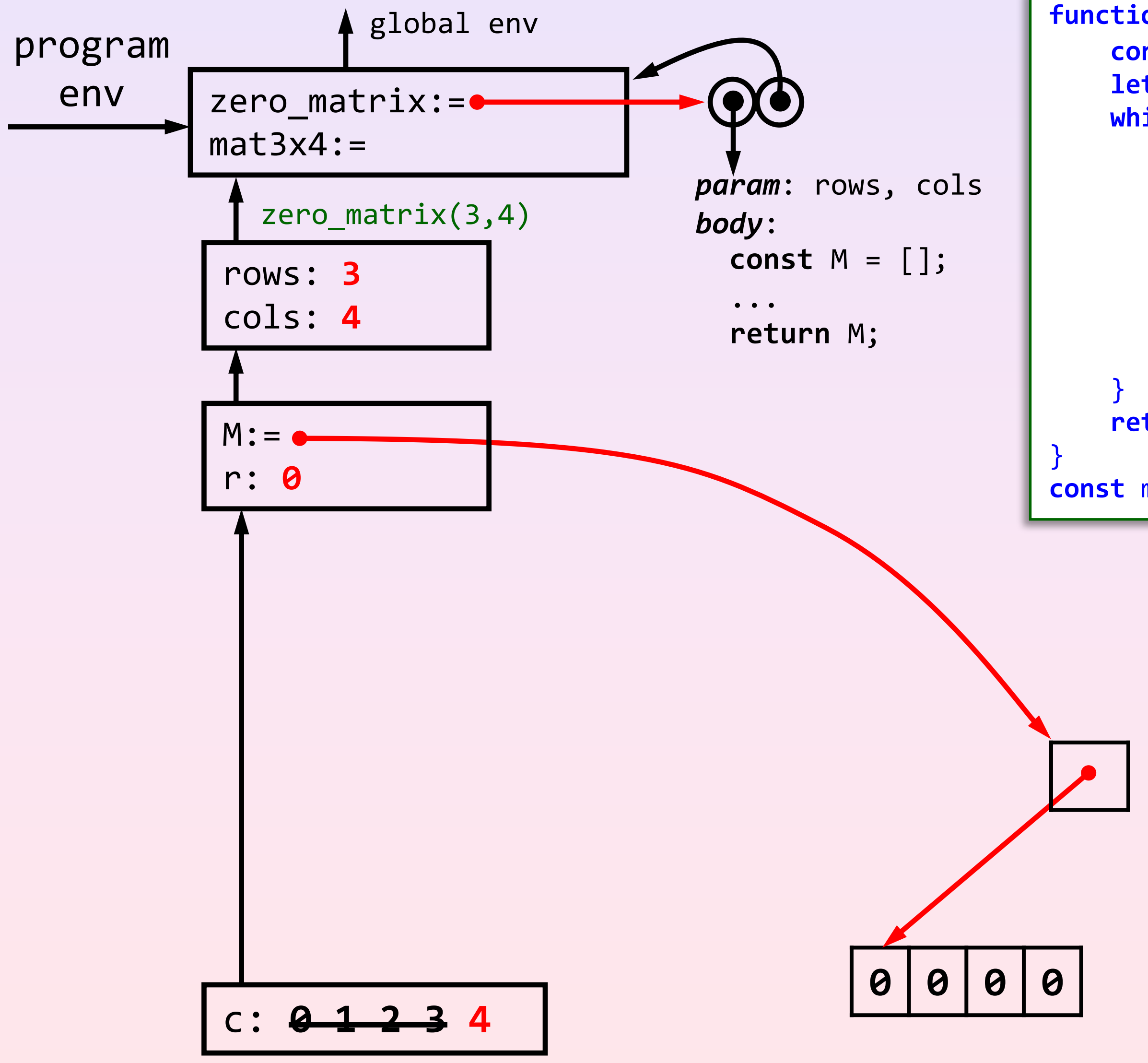
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



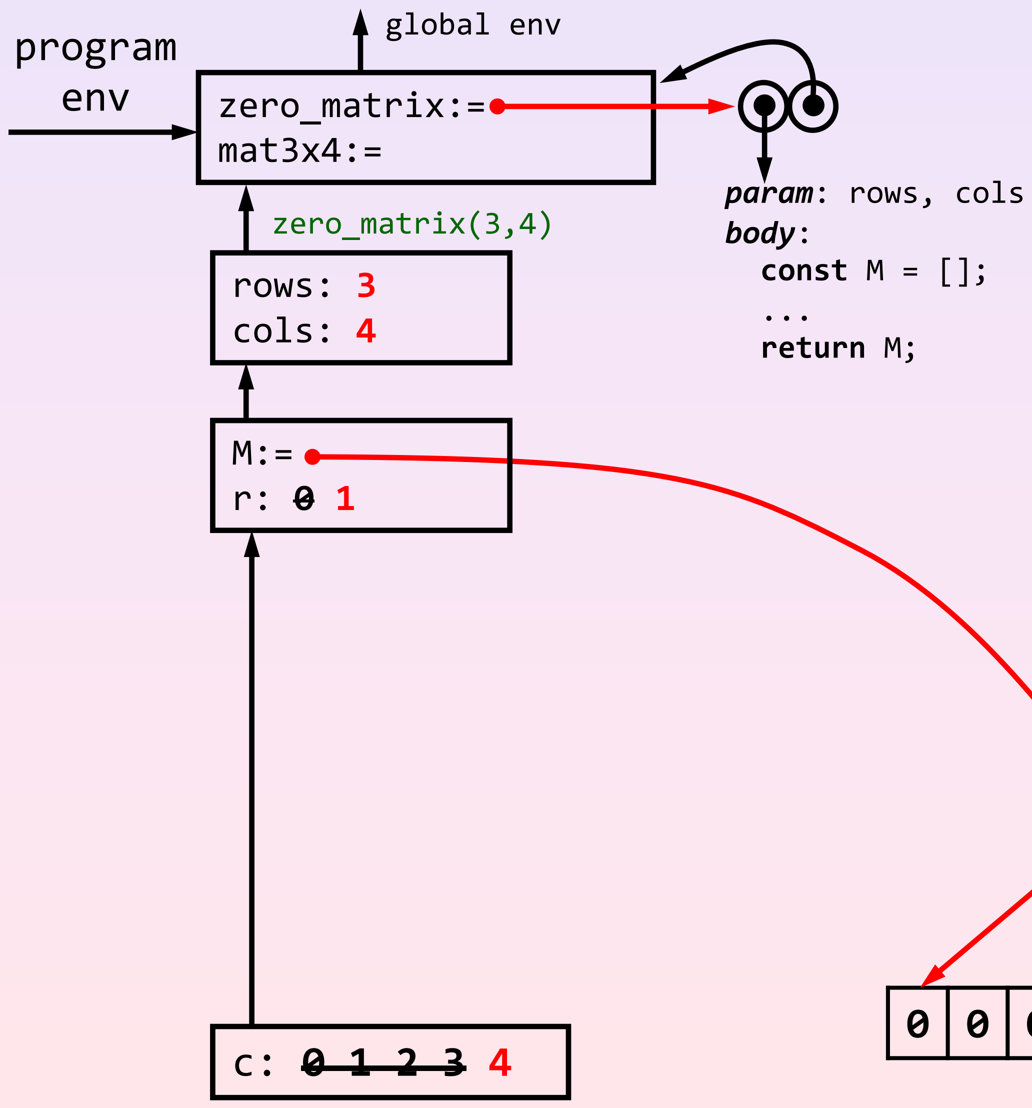
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



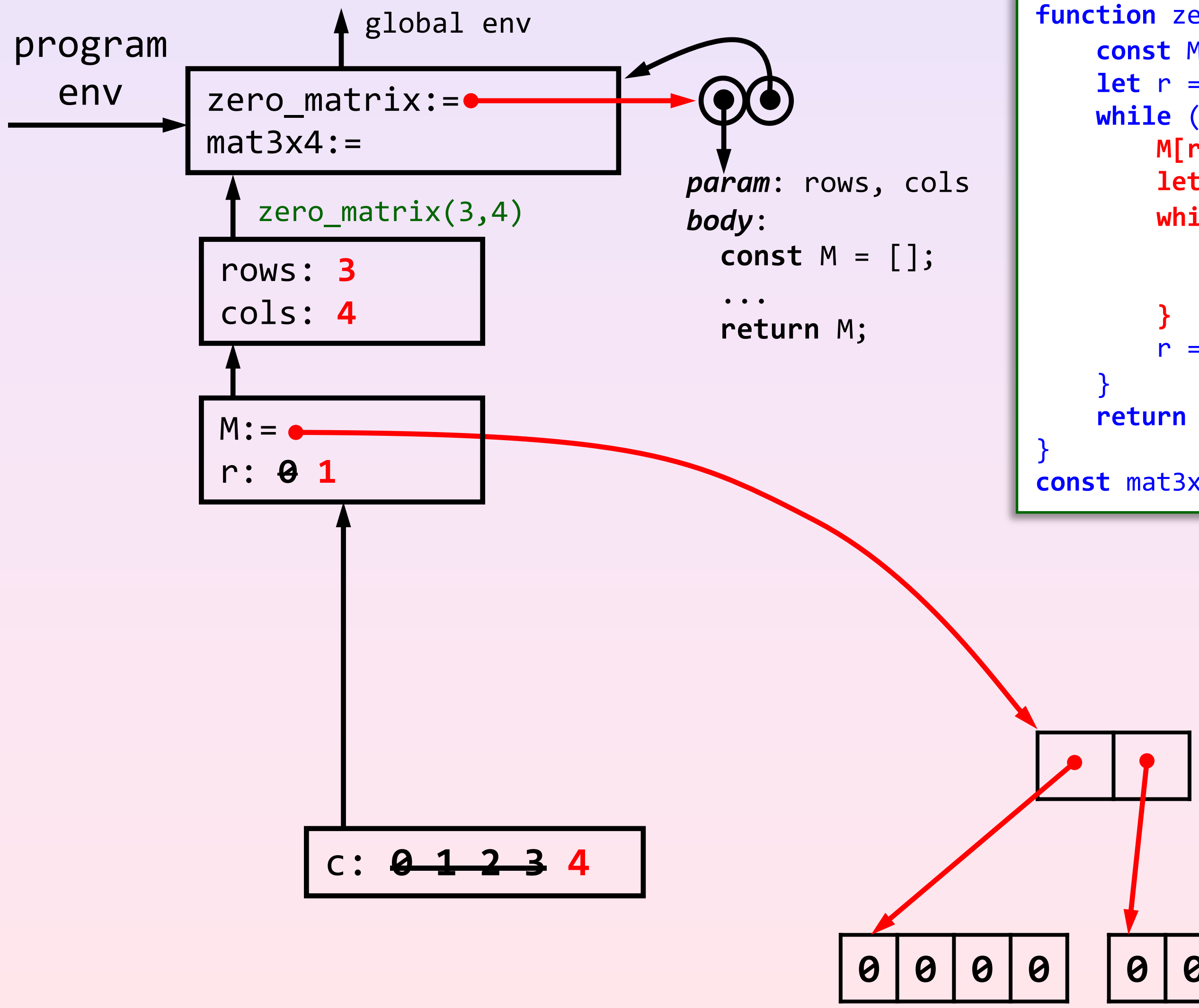
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



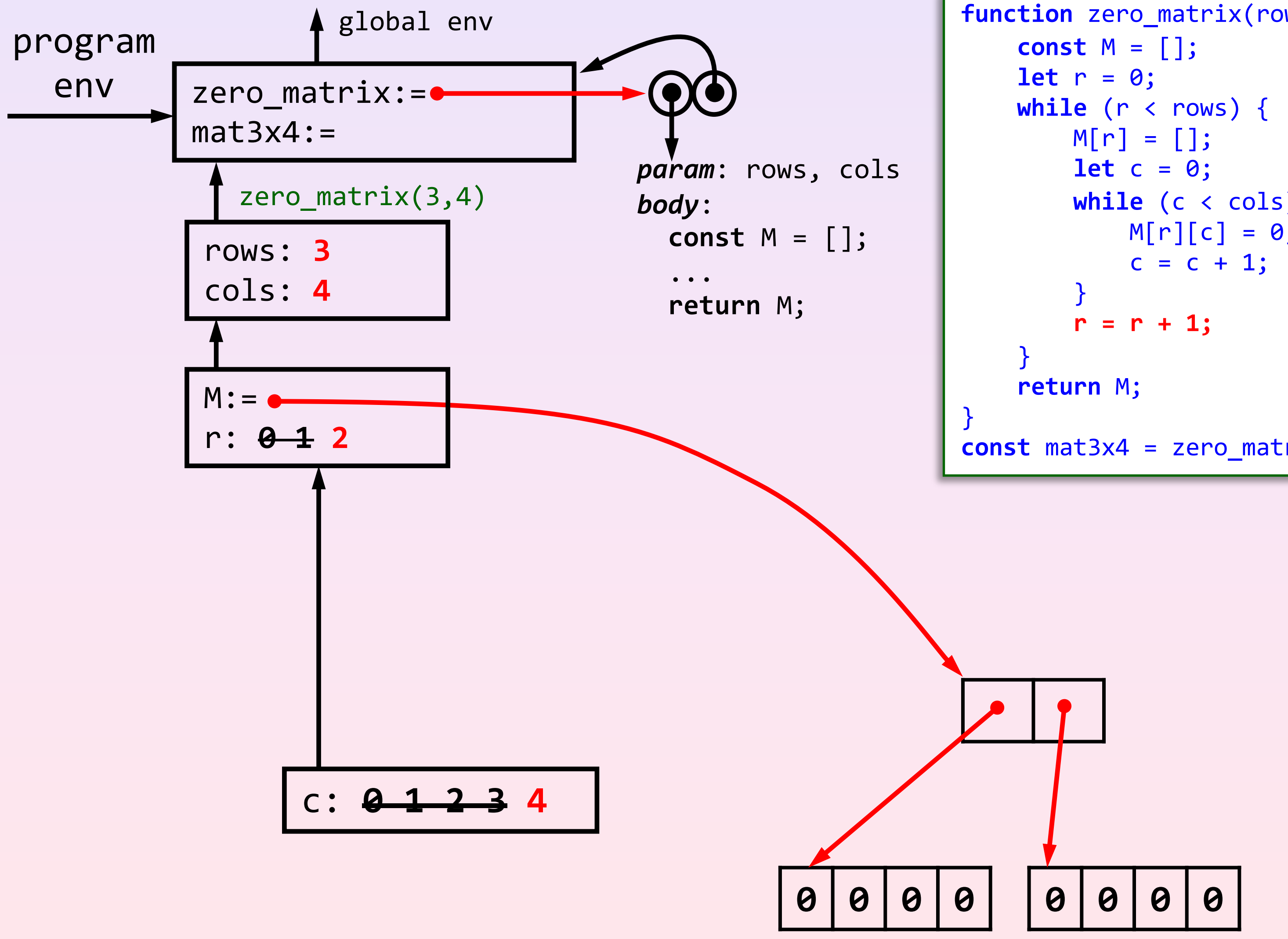
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



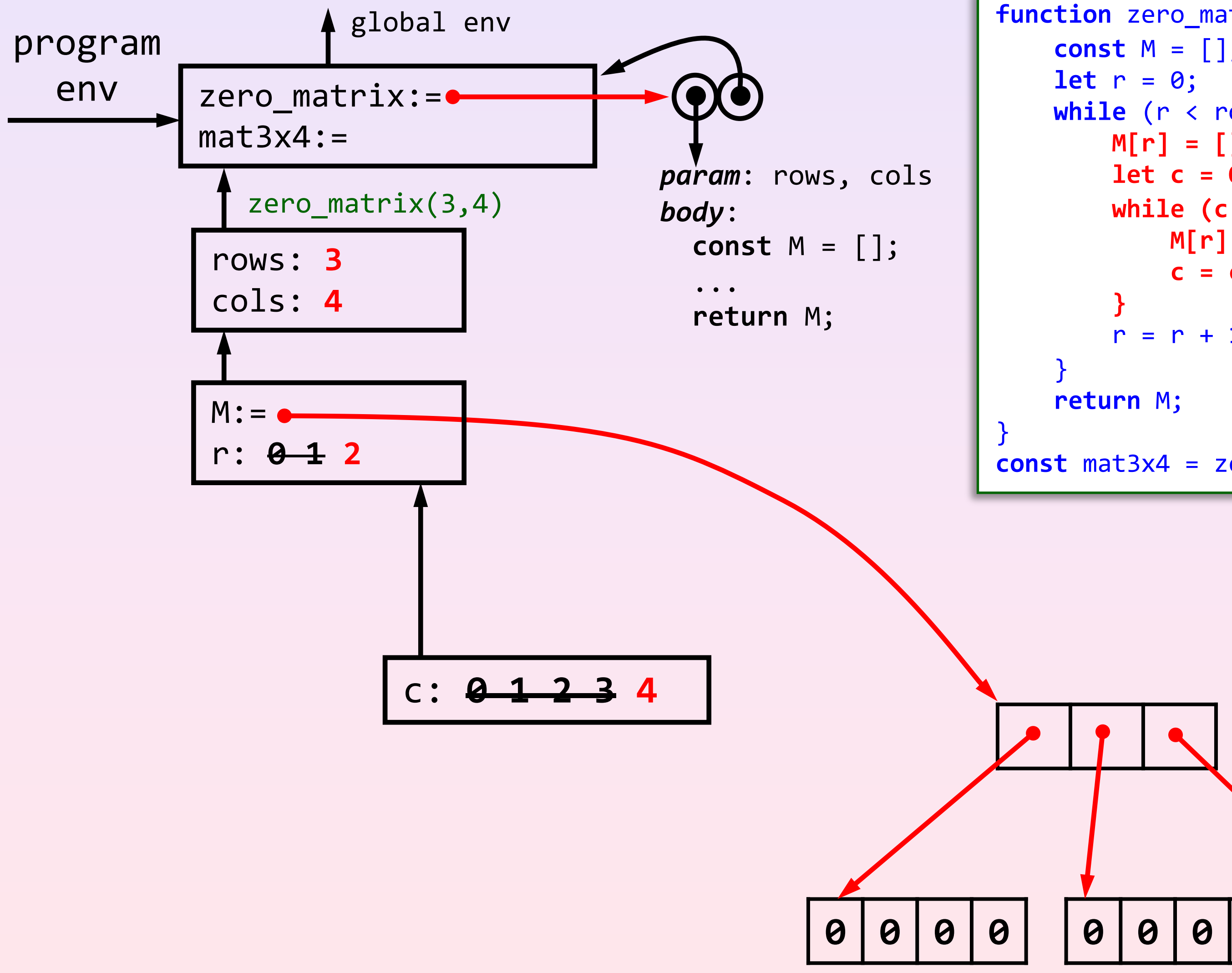
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



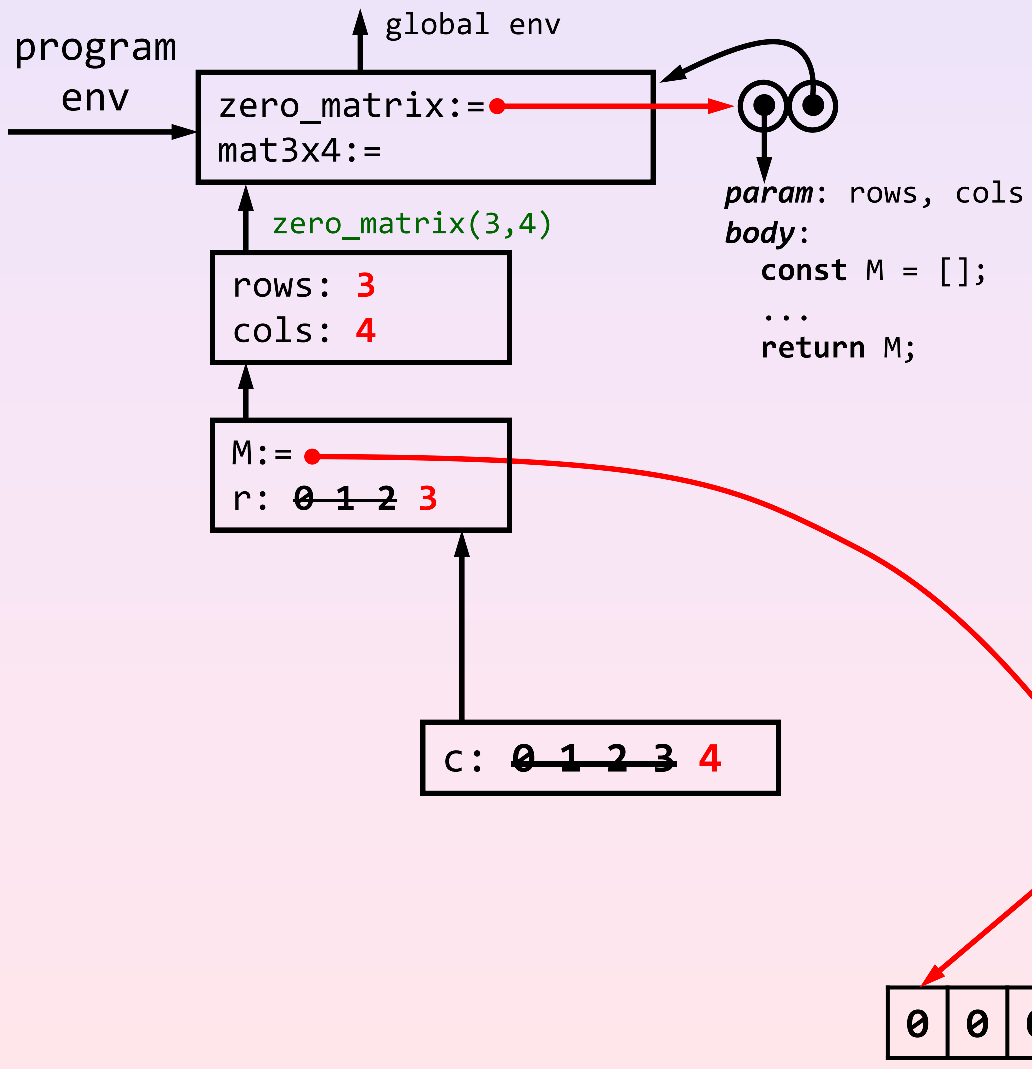
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



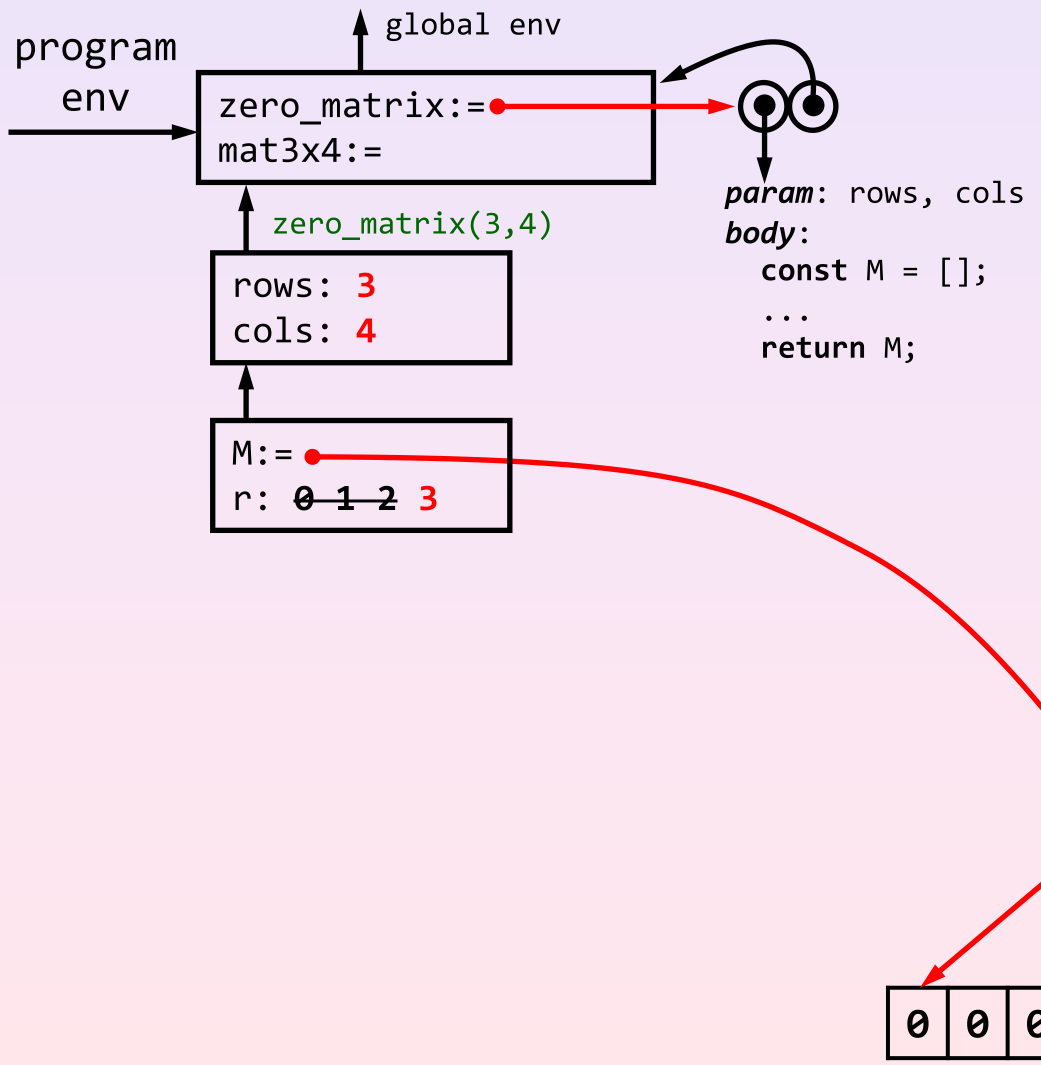
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



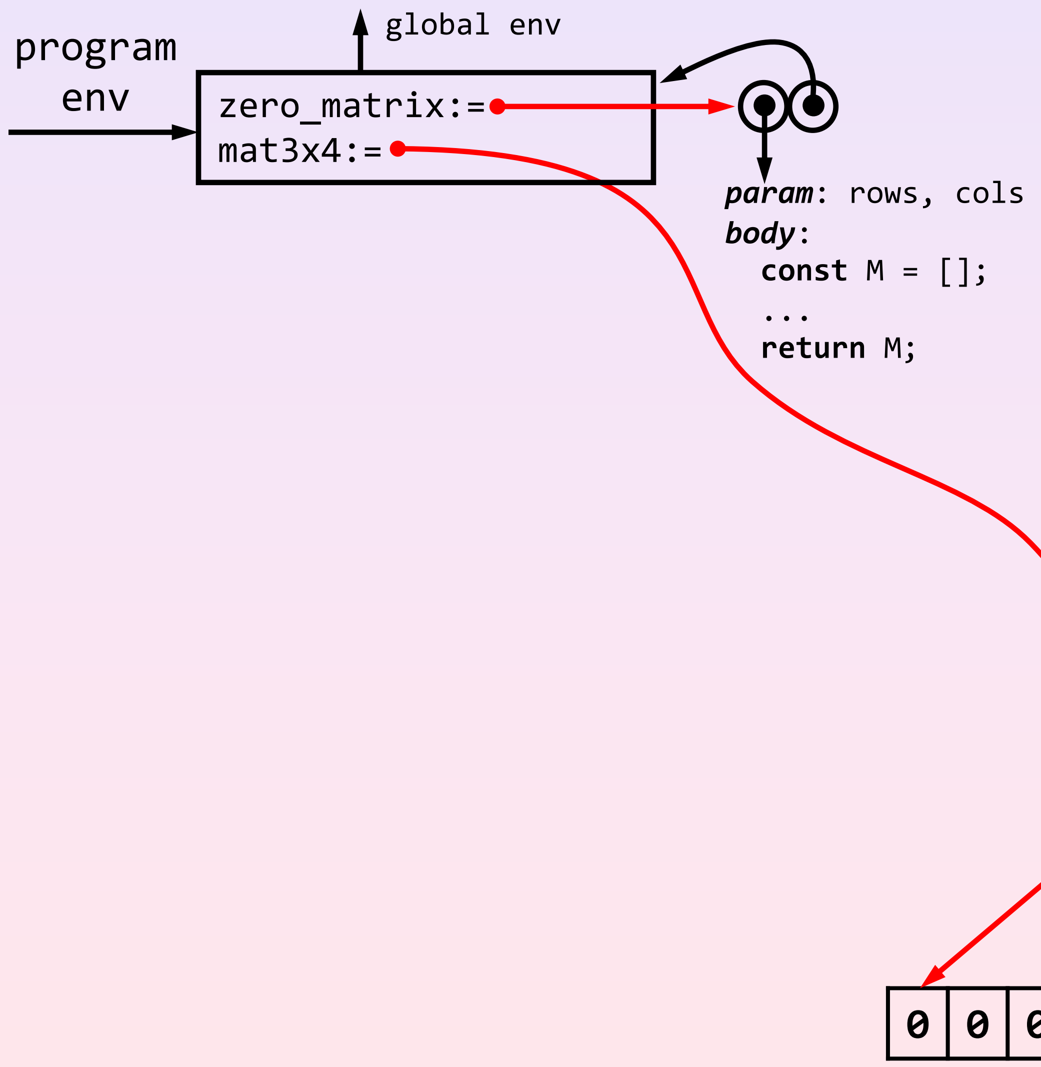
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



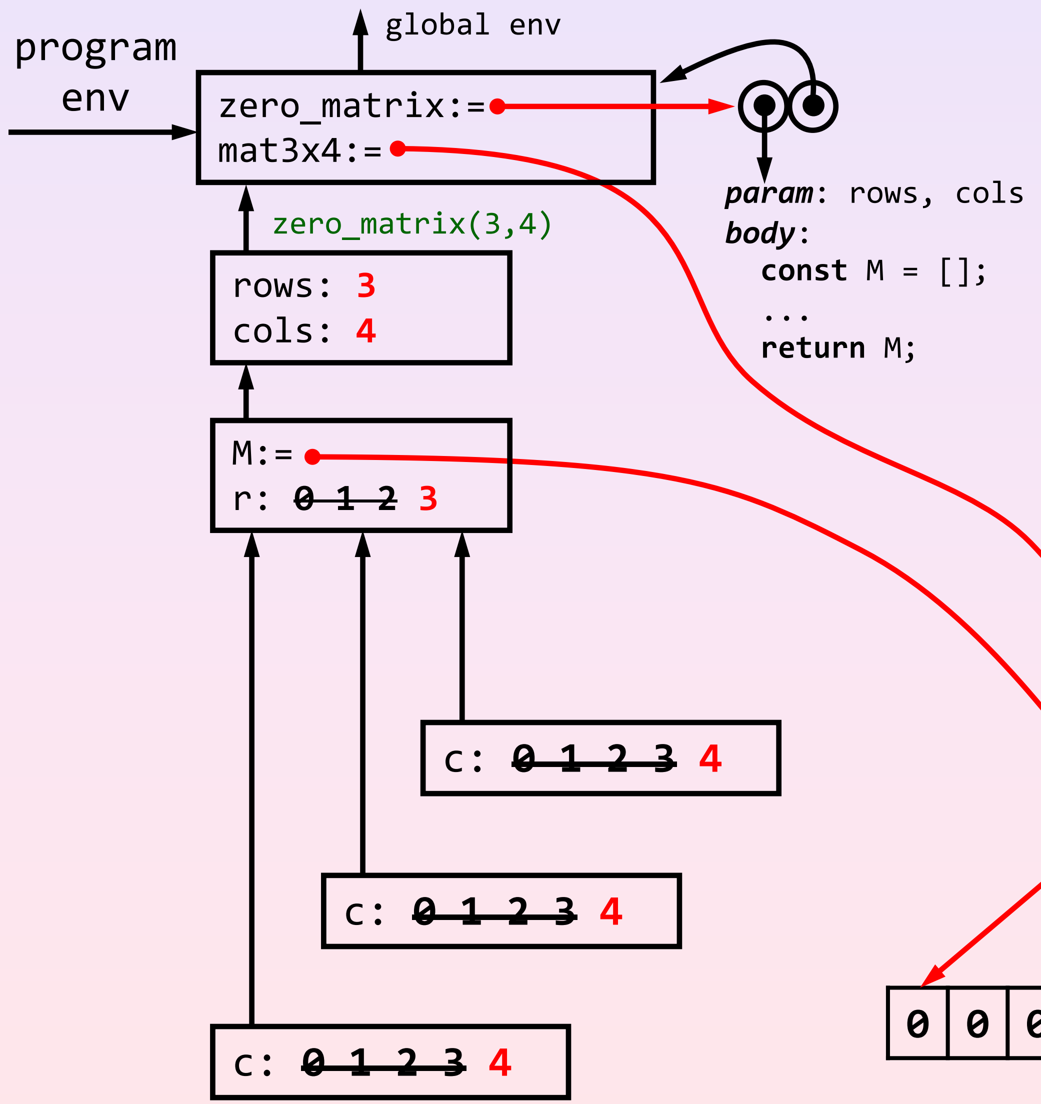
```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```

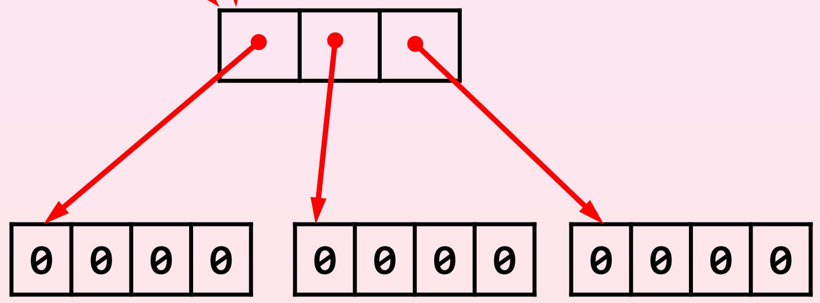


```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```



```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```





```
function zero_matrix(rows, cols) {
    const M = [];
    let r = 0;
    while (r < rows) {
        M[r] = [];
        let c = 0;
        while (c < cols) {
            M[r][c] = 0;
            c = c + 1;
        }
        r = r + 1;
    }
    return M;
}
const mat3x4 = zero_matrix(3, 4);
```

Showing all frames!

Order of Growth in Time of `zero_matrix`

```
function zero_matrix(rows, cols) {
    const M = [];
    for (let r = 0; r < rows; r = r + 1) {
        M[r] = [];
        for (let c = 0; c < cols; c = c + 1) {
            M[r][c] = 0;
        }
    }
    return M;
}
```

Show in Playground

- What is the order of growth in time?
  - Θ(rows * cols)

Summary on Arrays and Loops

- **Arrays** support **random access** to the elements
- **Loops** are convenient for **iterative** computations
- **`for`** loops add convenience and readability to **`while`** loops
- **`break`** and **`continue`** add flexibility
- **Loops** can be **nested** inside other loops

Outline

- Arrays
- Loops
- Arrays and Loops
- Environments of Arrays and Loops
- Extended Environment Model

The journey

- **Calculator language**
- Add conditionals, Booleans, sequences
- Add blocks, declarations, names
- Add function declaration and application (simple return)
- Restoring environments
- Further language features

Program consists of a single expression statement

```
1 + (2 * 3 - 4)
```

Evaluate expression statement:
    find operator

```
1 + (2 * 3 - 4)
```

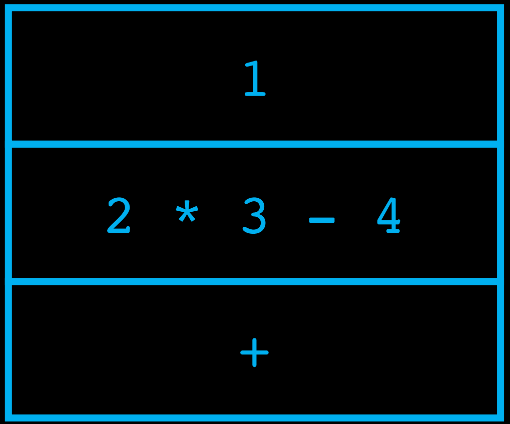
Operator combination:

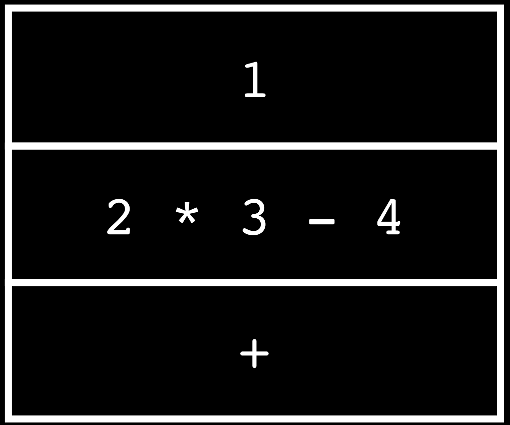
separate operands and operator

```
1 + (2 * 3 - 4)
```

Operator combination:

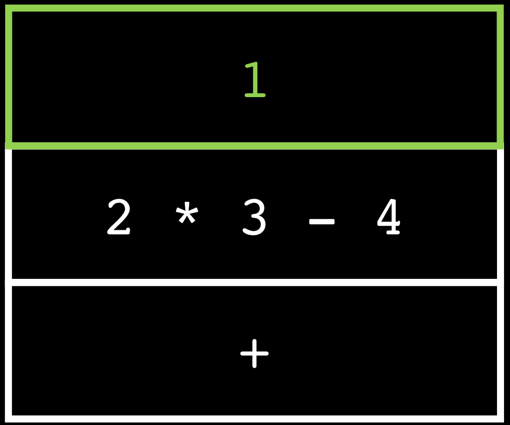
separate operands and operator




1
2 * 3 - 4
+

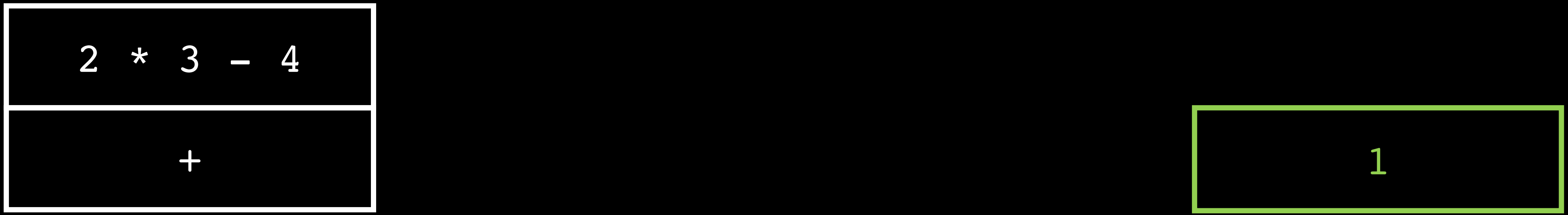
Literal value:

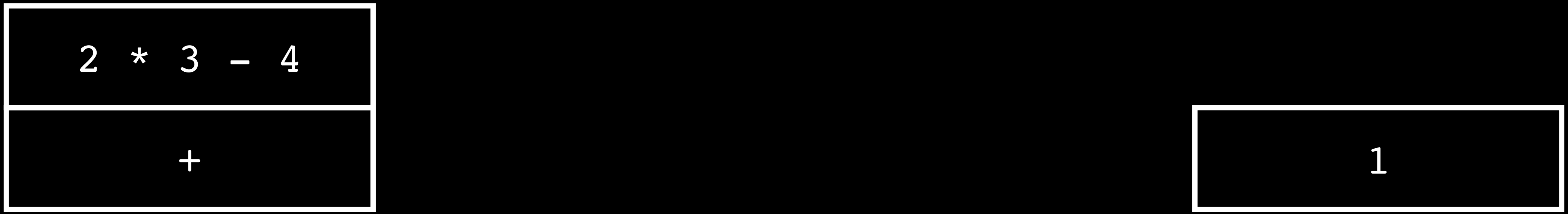
set aside for future use



Literal value:

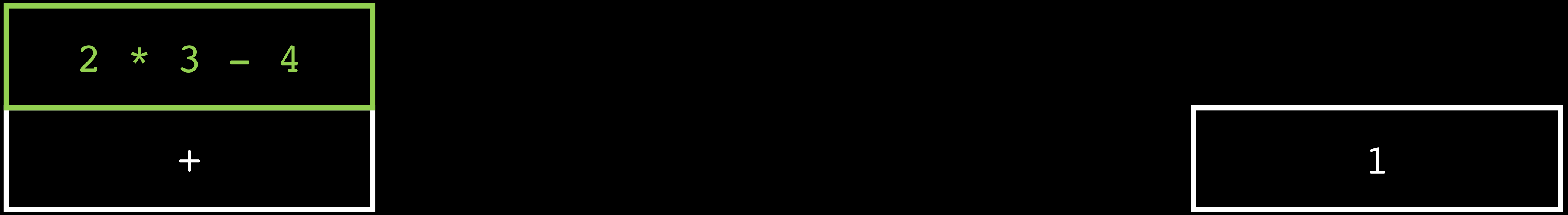
set aside for future use




2 * 3 - 4
+
1

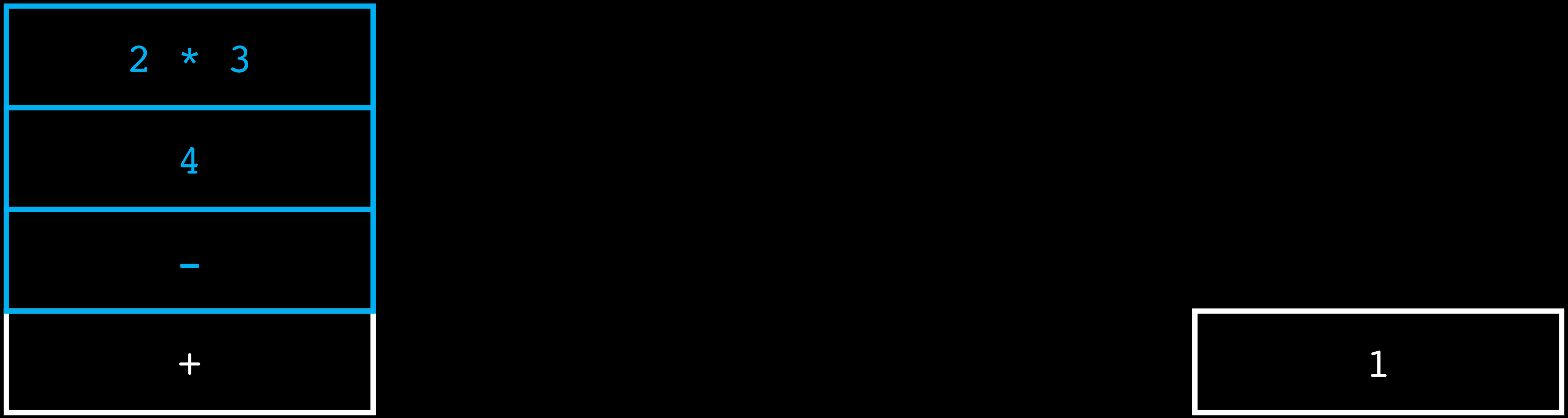
Operator combination:

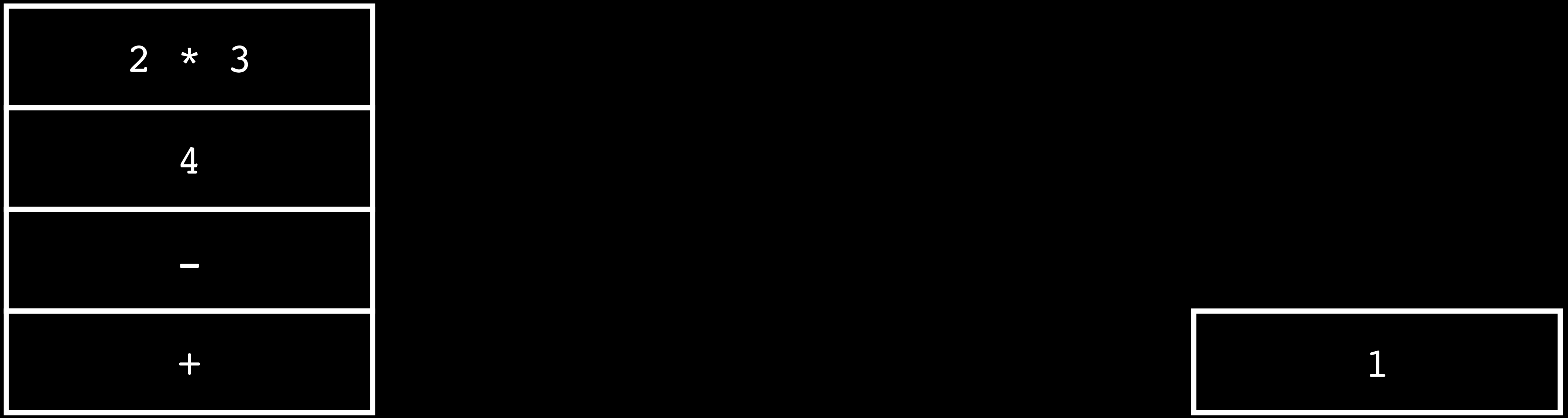
separate operands and operator



Operator combination:

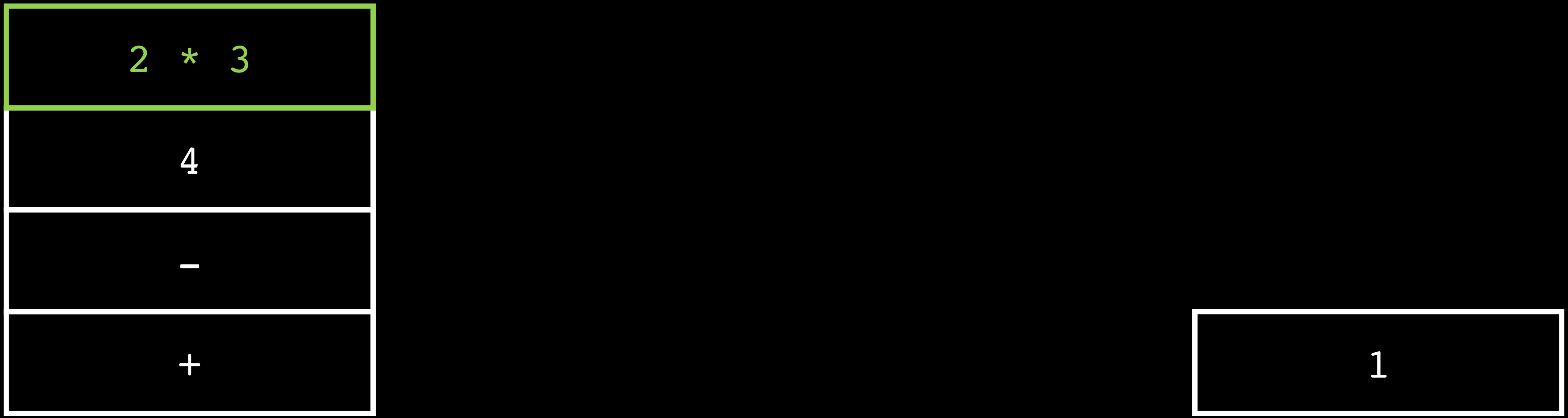
separate operands and operator



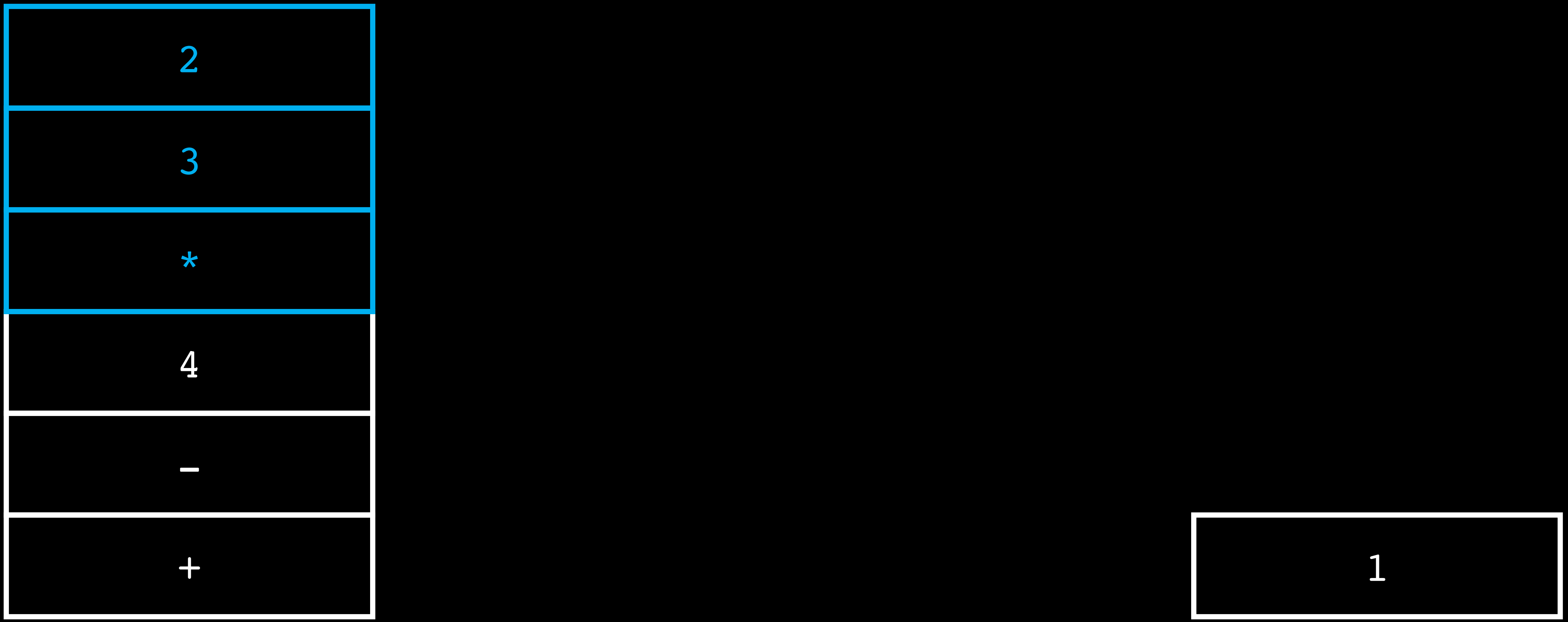

2 * 3
4
-
+
1

Operator combination:

separate operands and operator

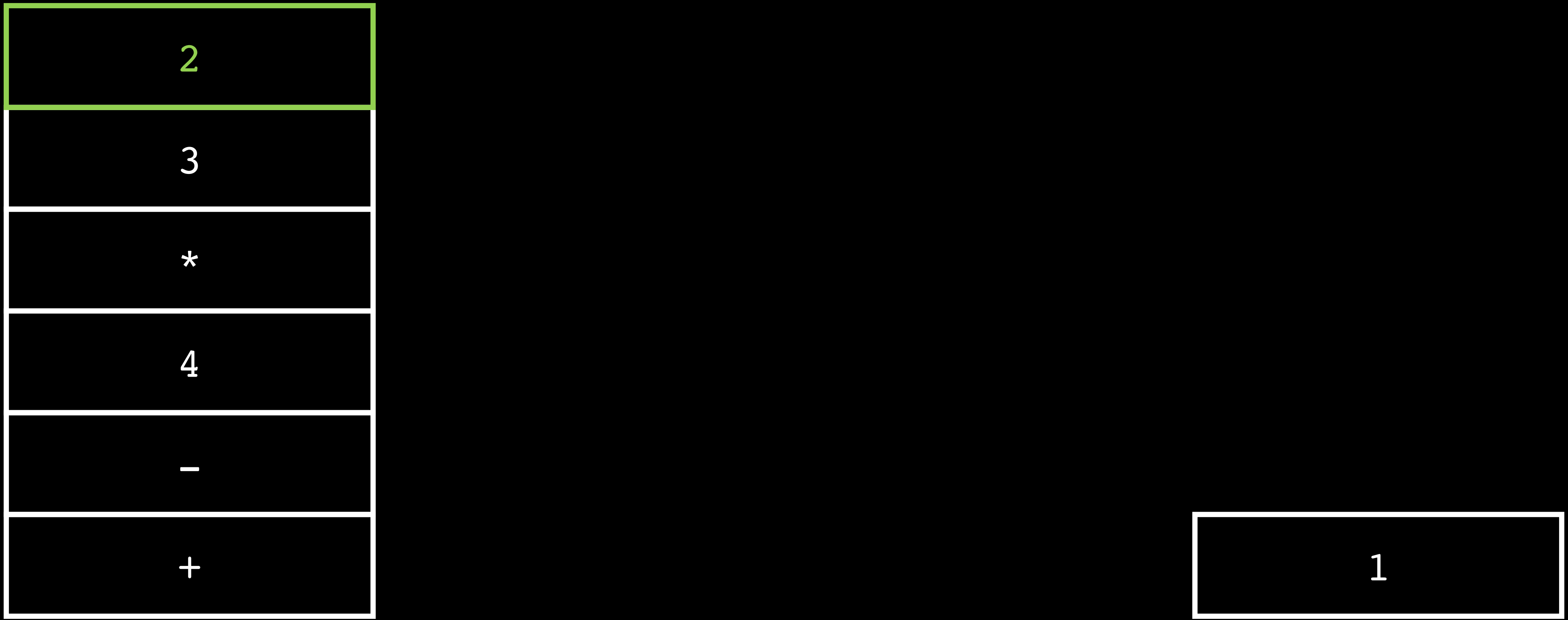


Operator combination:
separate operands and operator



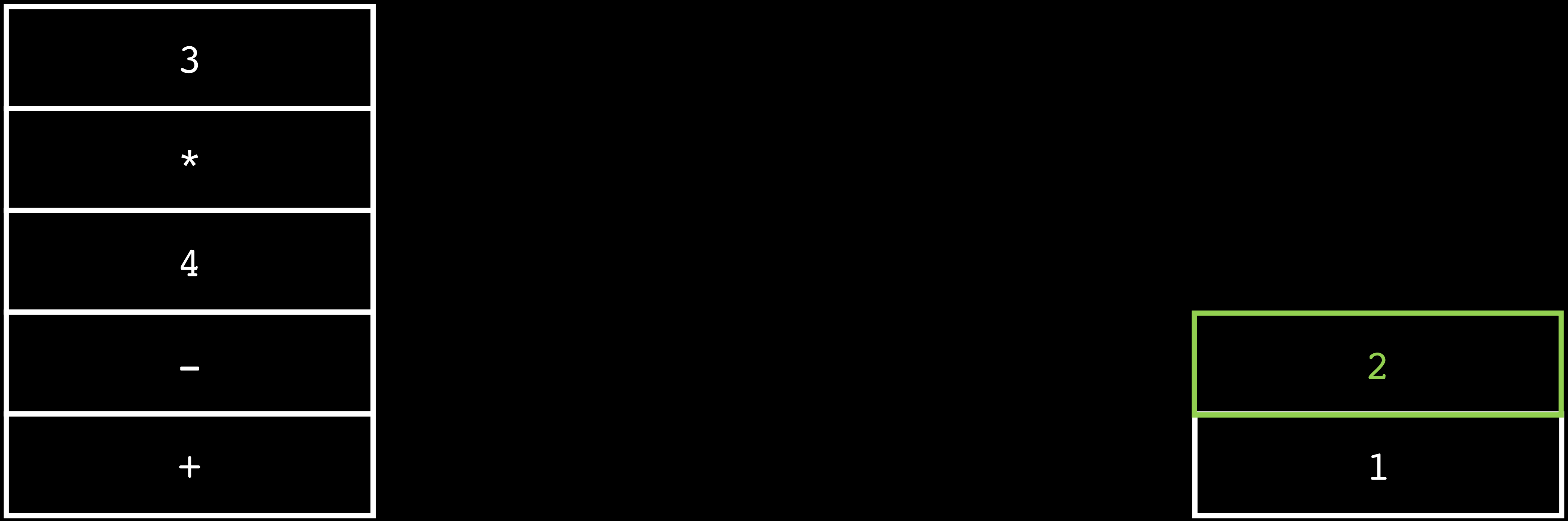
Literal value:

set aside for future use

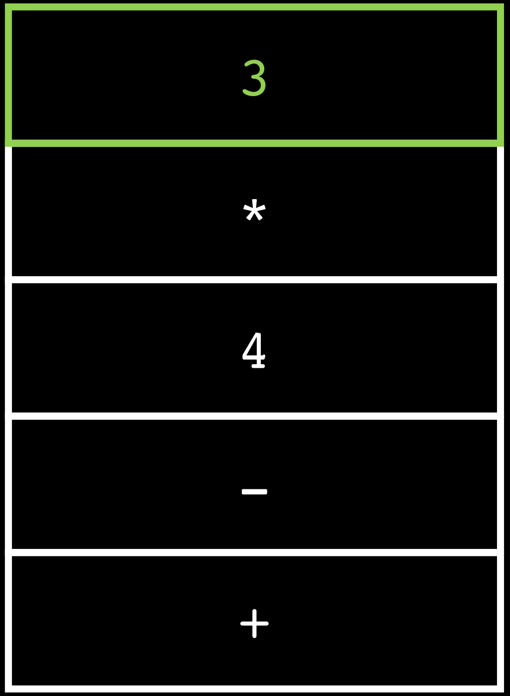


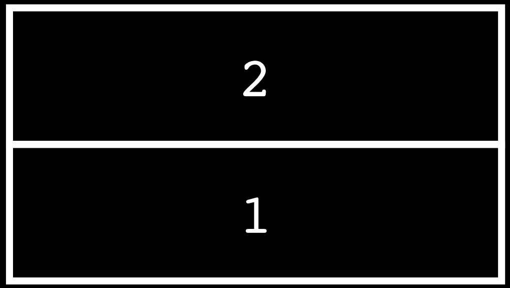
Literal value:

set aside for future use



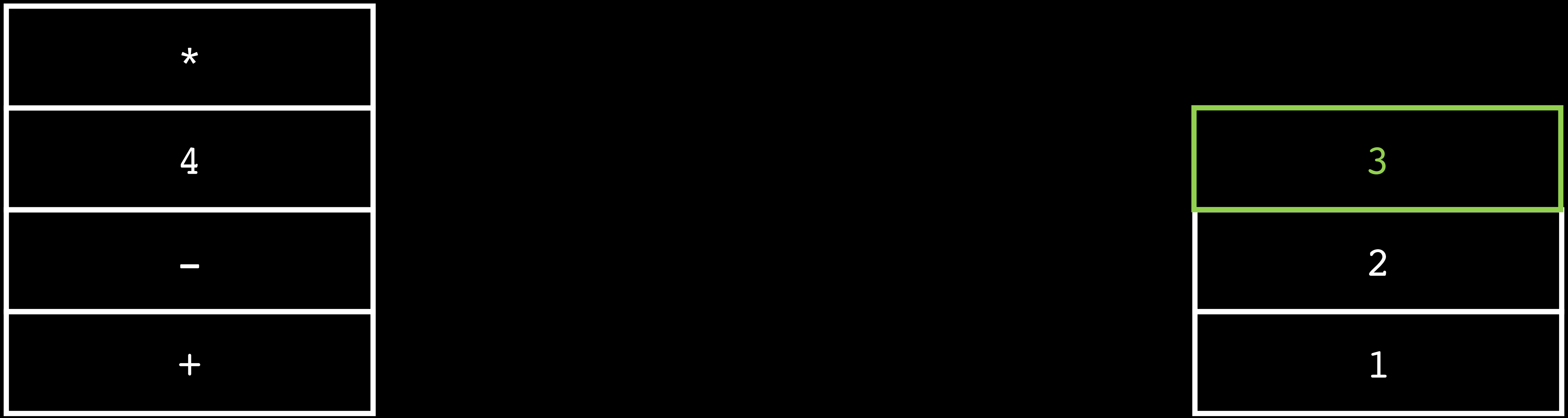
Literal value:
    set aside for future use





Literal value:

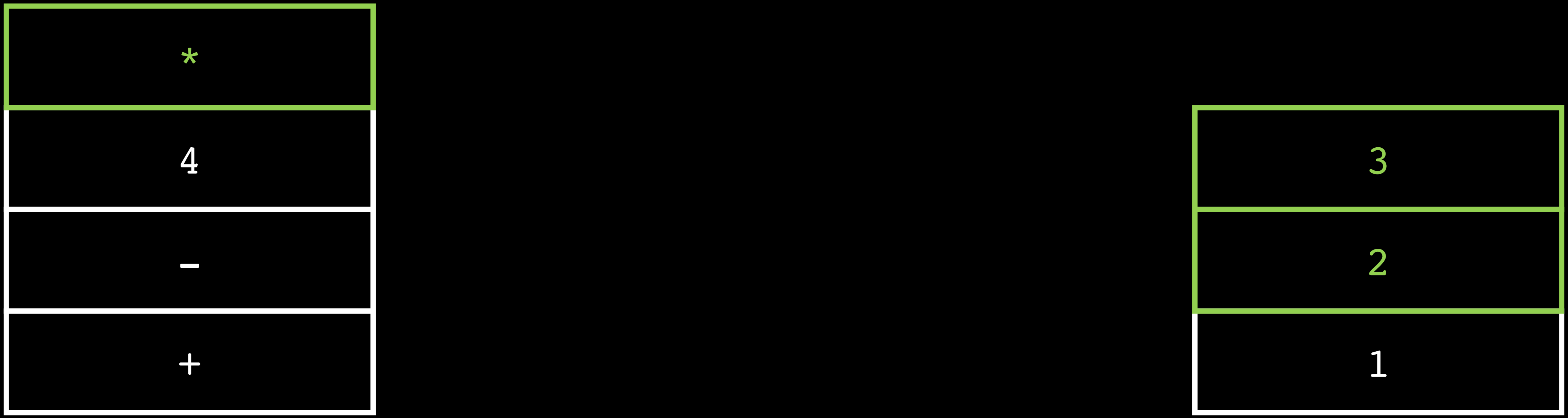
set aside for future use



Operator:

operate on top values set aside

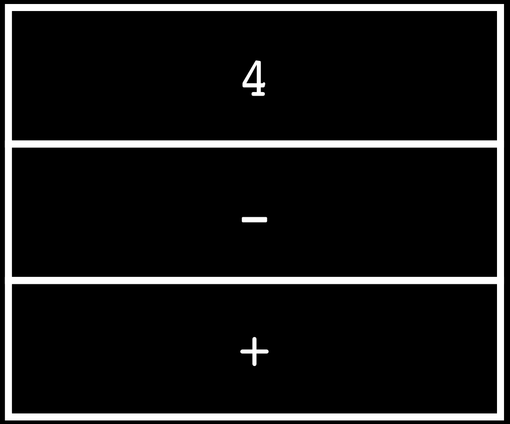
set result aside for future use

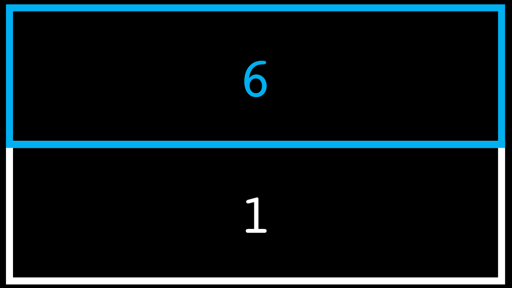


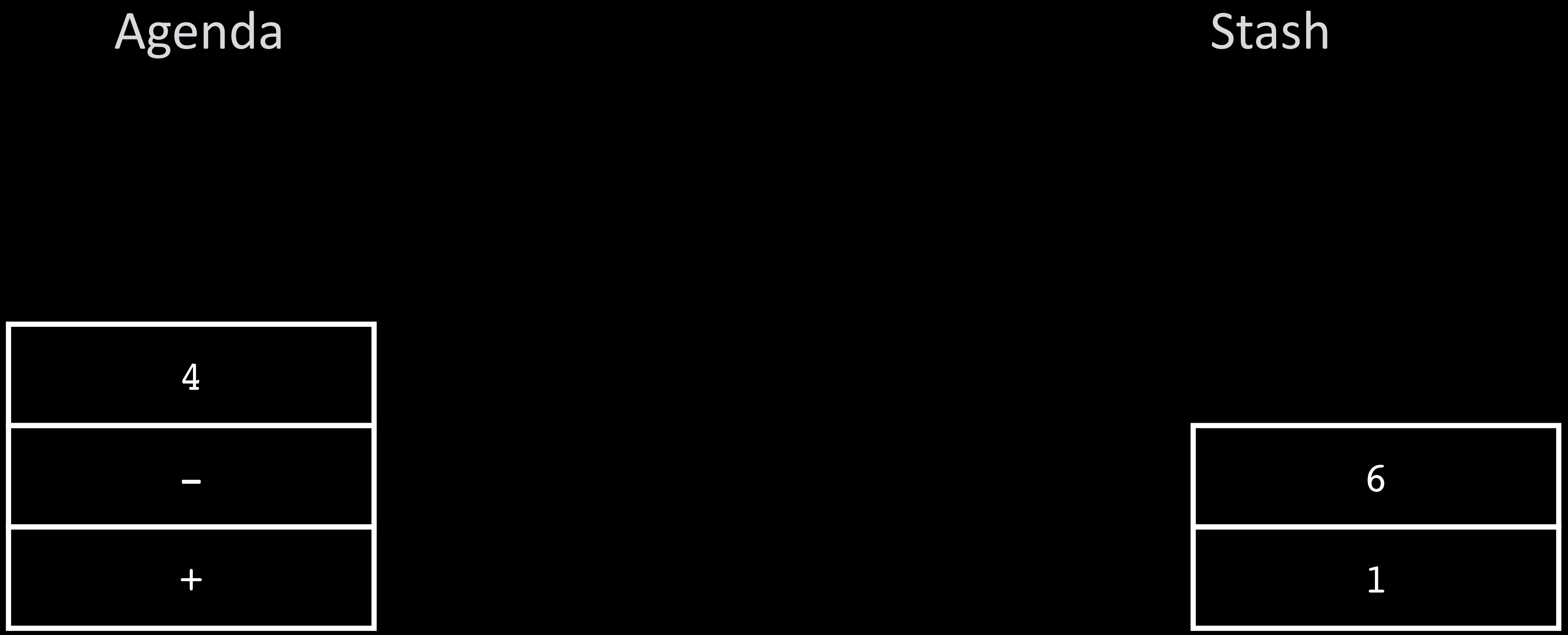
Operator:

operate on top values set aside

set result aside for future use

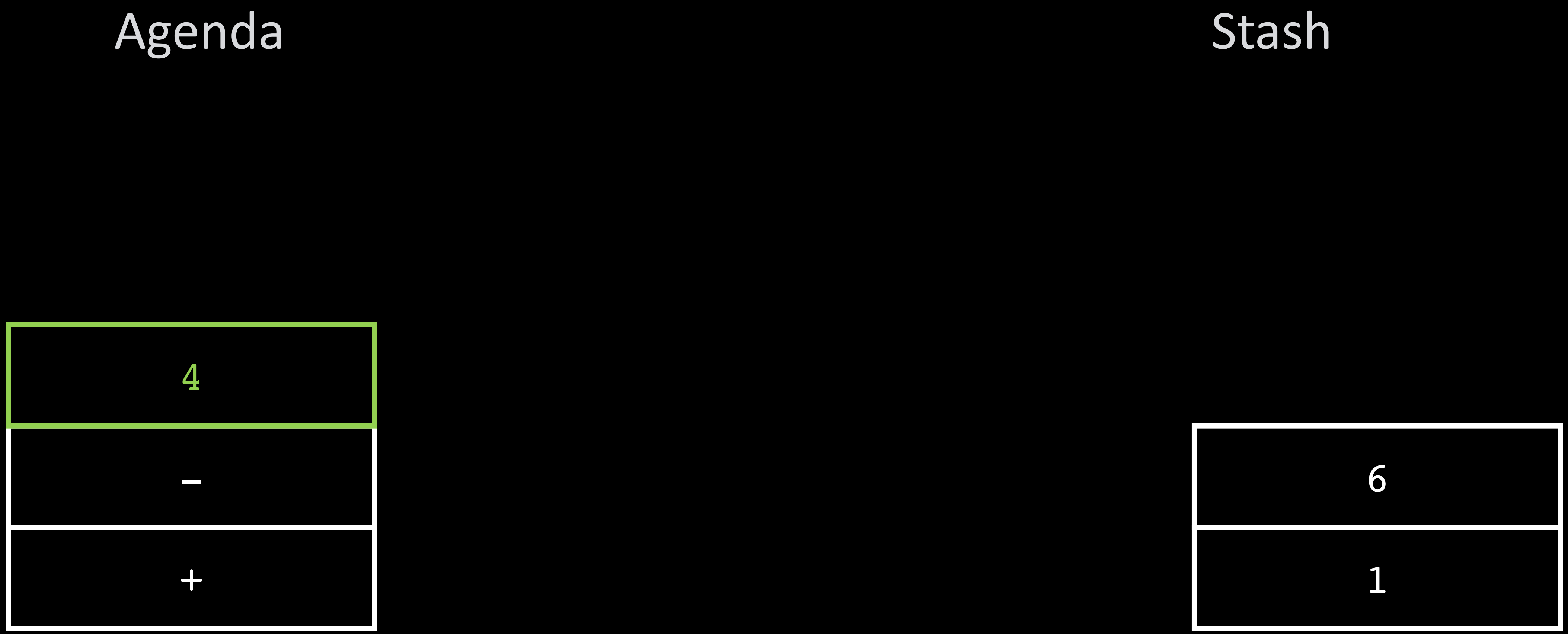





Agenda
Stash
4
−
+
6
1

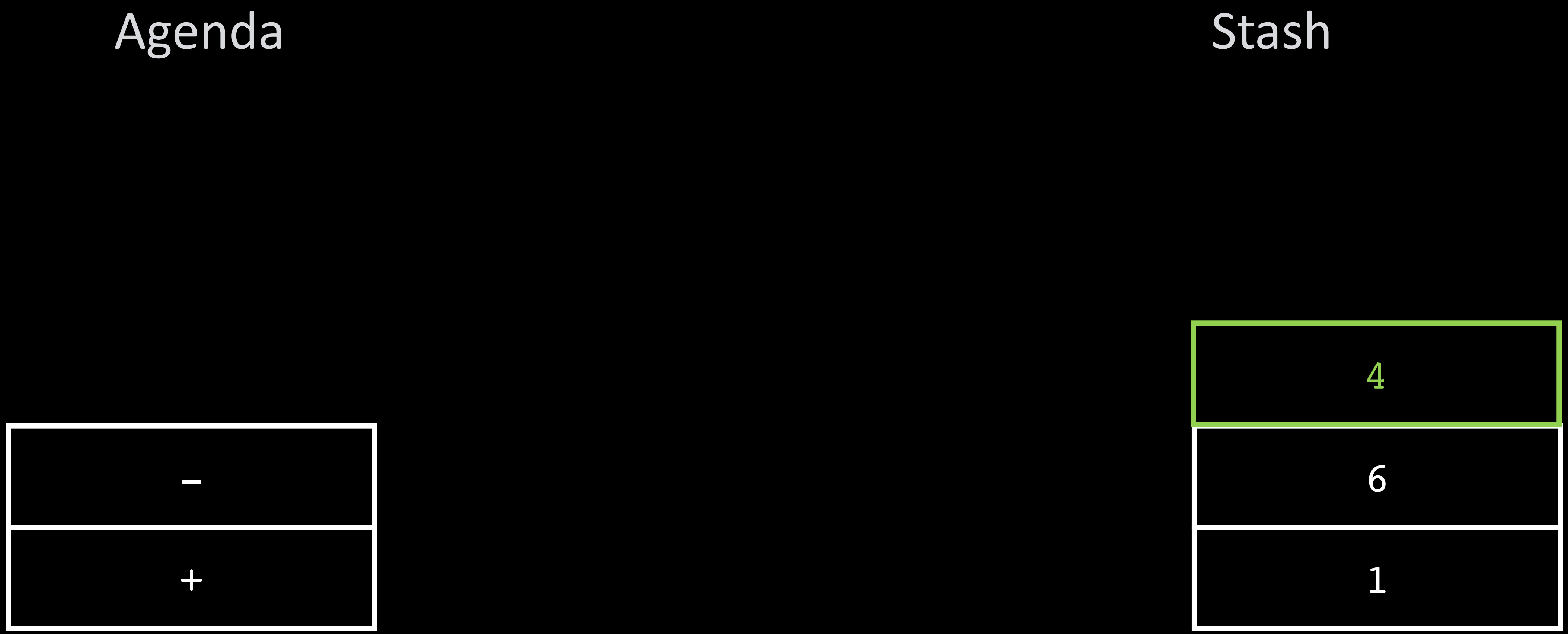
Literal value:

- **pop** from agenda
- **push** on stash



Literal value:

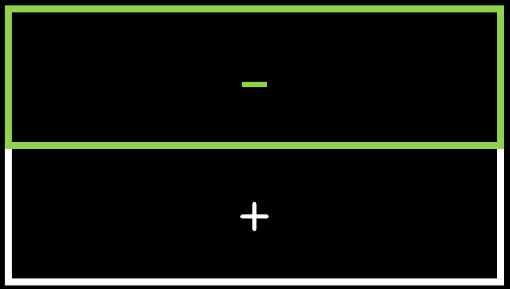
- **pop** from agenda
- **push** on stash



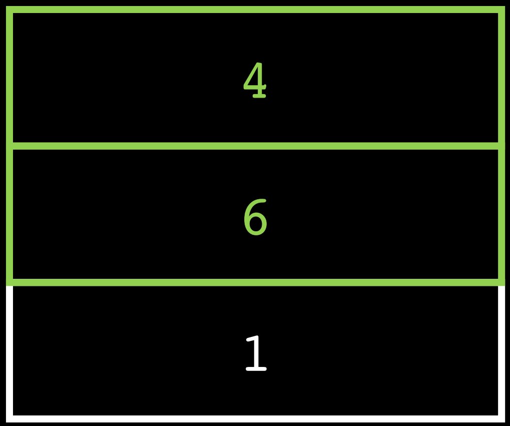
Operator:

- **pop** operands from stash
- **pop** operator from agenda
- compute result
- **push** result on stash

Agenda

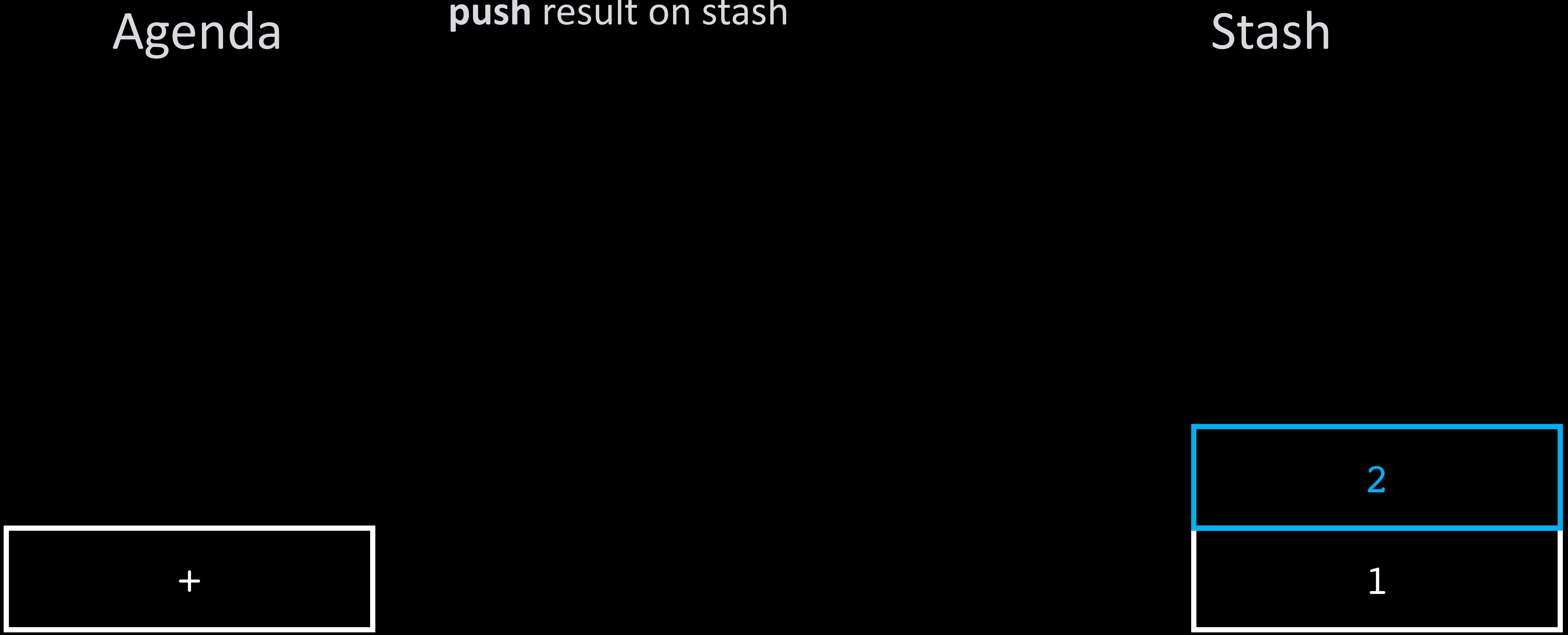


Stash



Operator:

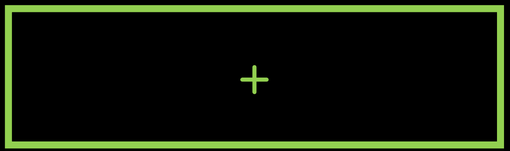
pop operands from stash
pop operator from agenda
compute result
push result on stash



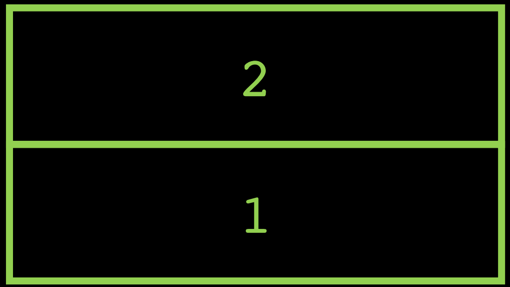
Operator:

    pop operands from stash
    pop operator from agenda
    compute result
    push result on stash

Agenda



Stash



Operator:

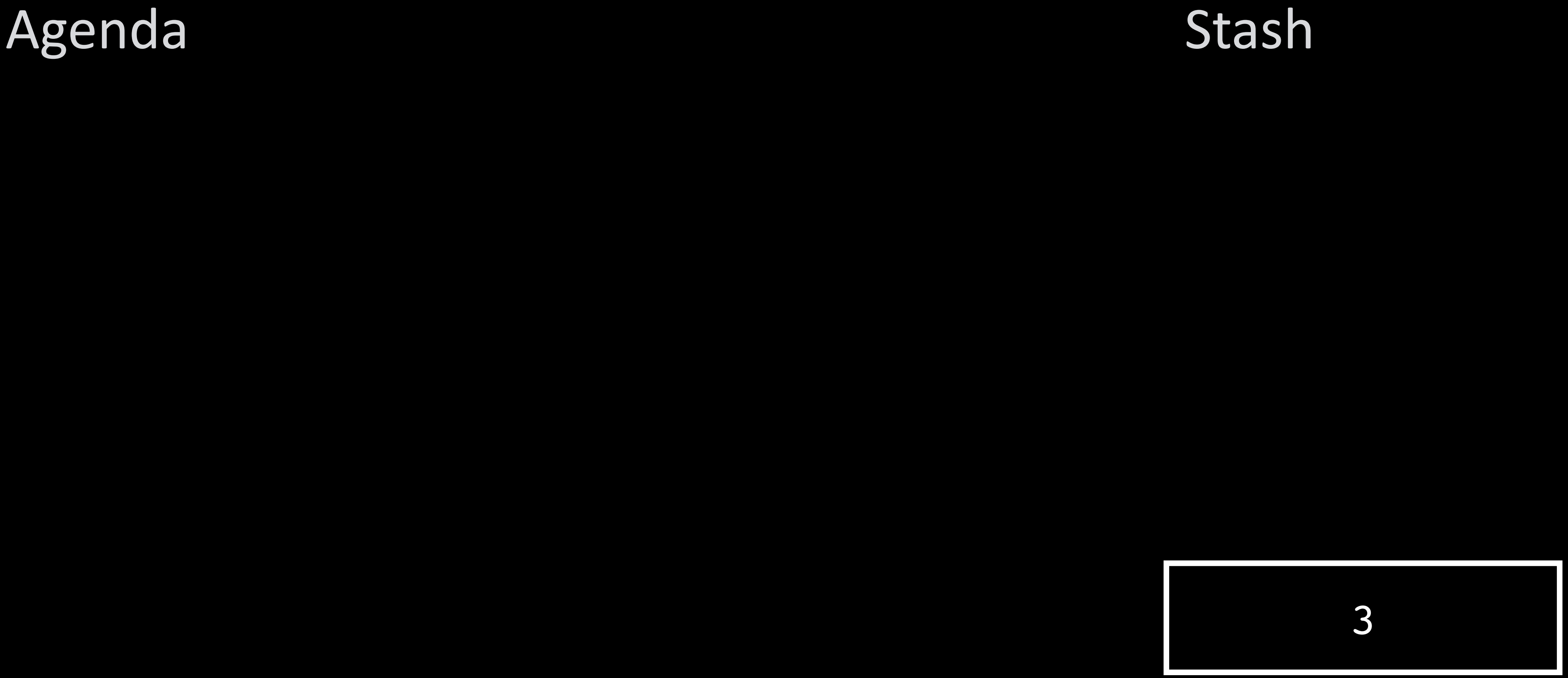
- pop operands from stash
- pop operator from agenda
- compute result
- push result on stash

Agenda

Stash

$\boxed{3}$

Agenda empty:
    done!
    result is on top of stash



The journey

- Calculator language
- **Add conditionals, Booleans, sequences**
- Add blocks, declarations, names
- Add function declaration and application (simple return)
- Restoring environments
- Further language features

Sequence

```
1 + 2;
3 * 4;
```

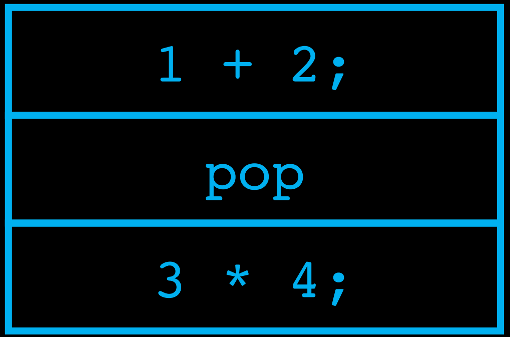
Sequence:

split sequence into components
separated by pop instructions

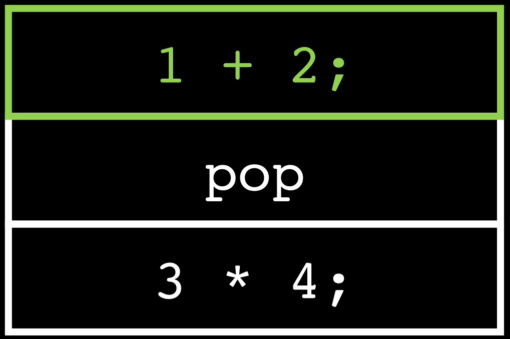
```
1 + 2;
3 * 4;
```

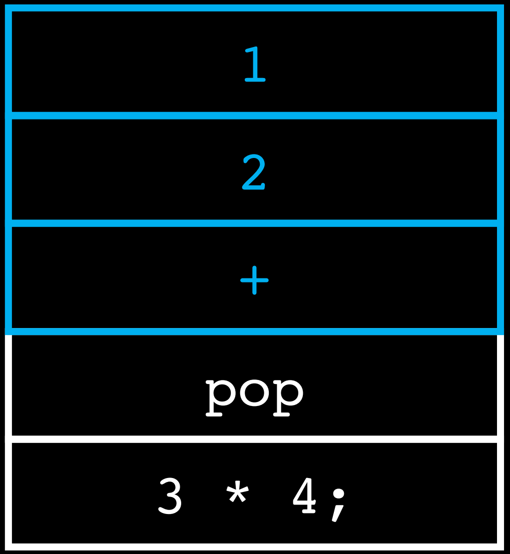
Sequence:

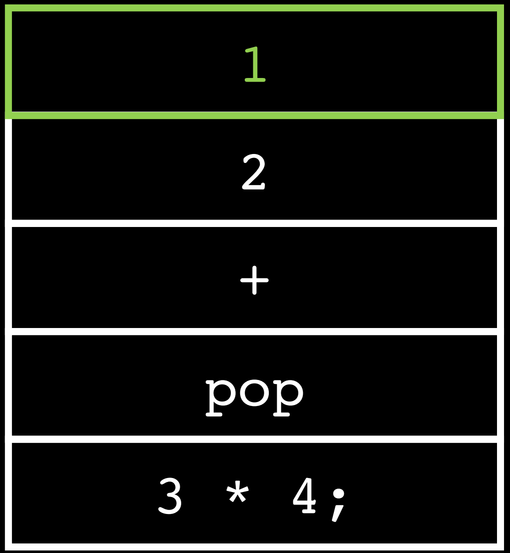
split sequence into components
separated by pop instructions

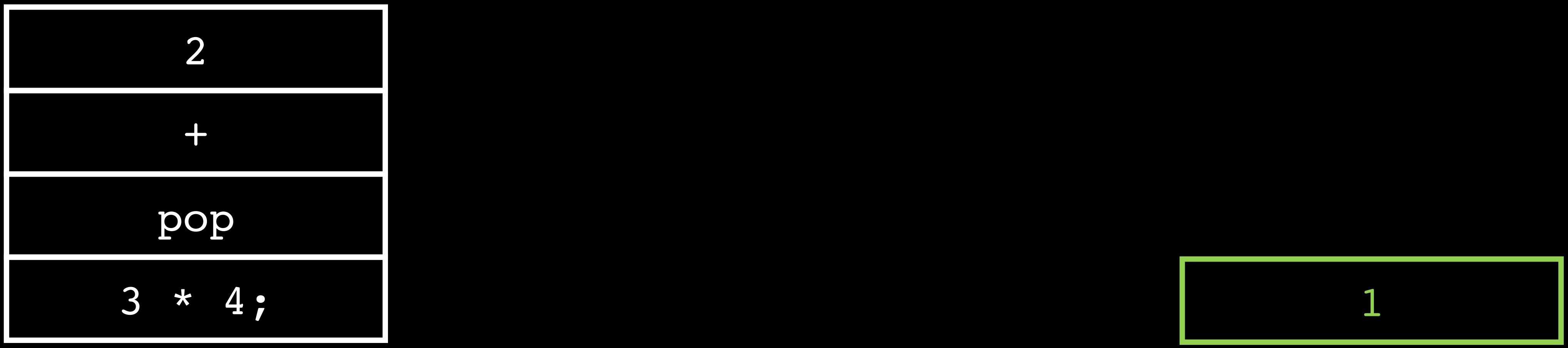


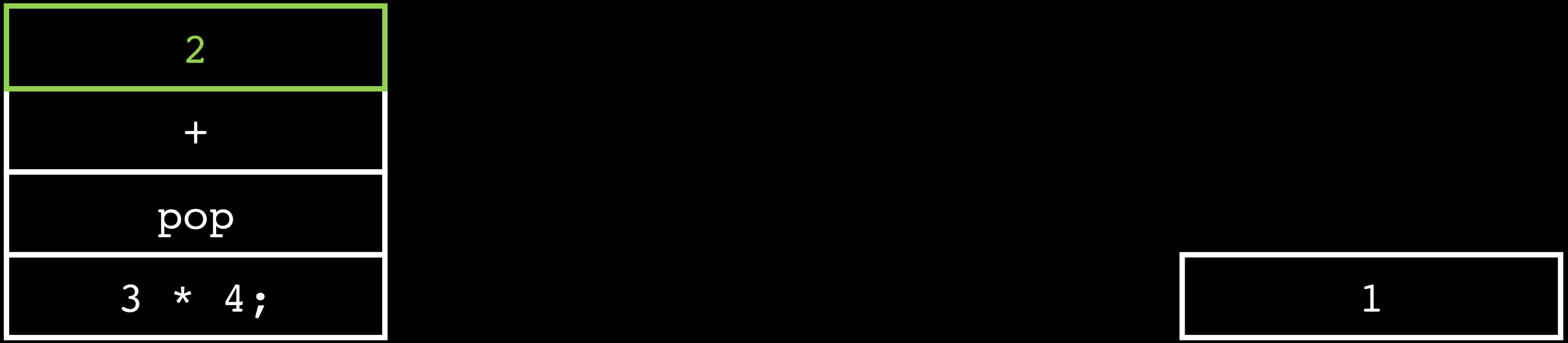
```
1 + 2;
pop
3 * 4;
```

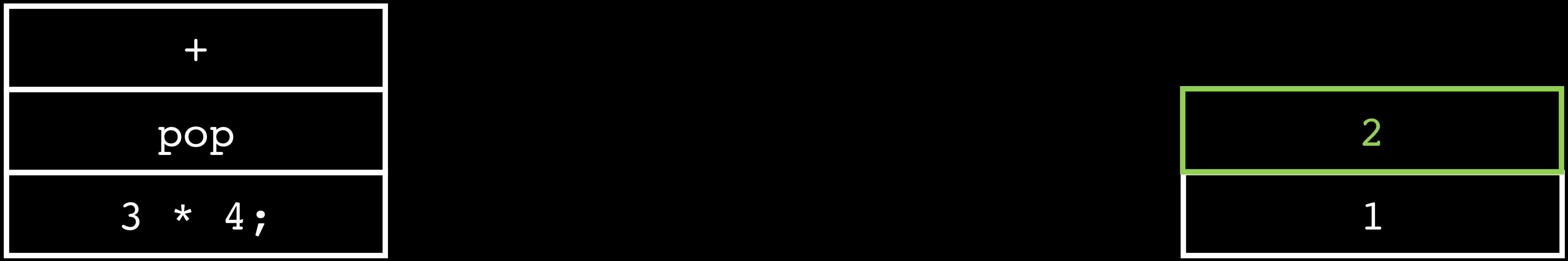

1 + 2;
pop
3 * 4;

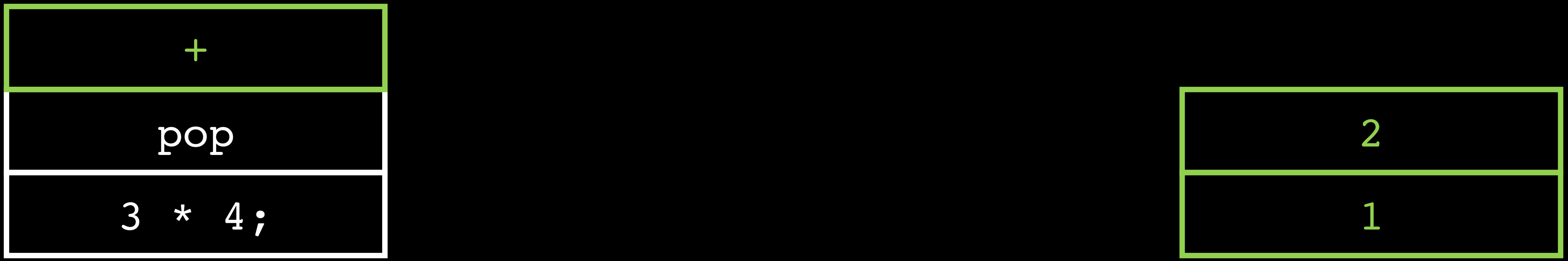

1
2
+
pop
3 * 4;

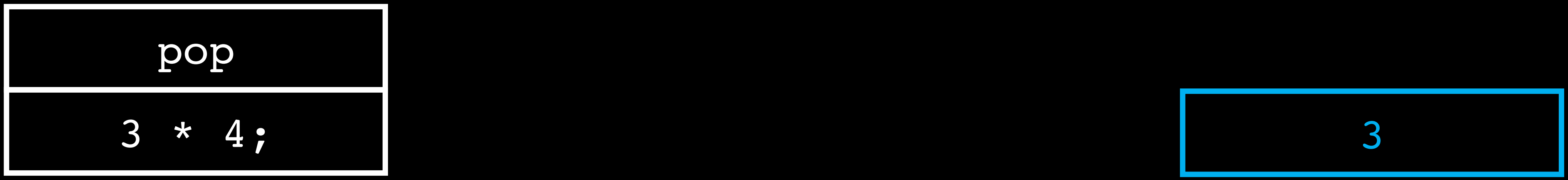

1
2
+
pop
3 * 4;


2
+
pop
3 * 4;
1


2
+
pop
3 * 4;
1


+
pop
3 * 4;
2
1


+
pop
3 * 4;
2
1



```
3 * 4;
```

Pop instruction:

- pop top value from stash
- pop the pop instruction from agenda

```
pop
3 * 4;
```

```
3
```

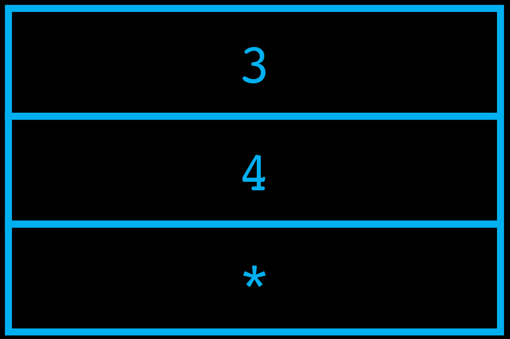
Pop instruction:

  pop top value from stash

  pop the pop instruction from agenda

```
3 * 4;
```

```
3 * 4;
```


3
4
*

…eventually

Agenda empty:

  done!

  result is on top of stash

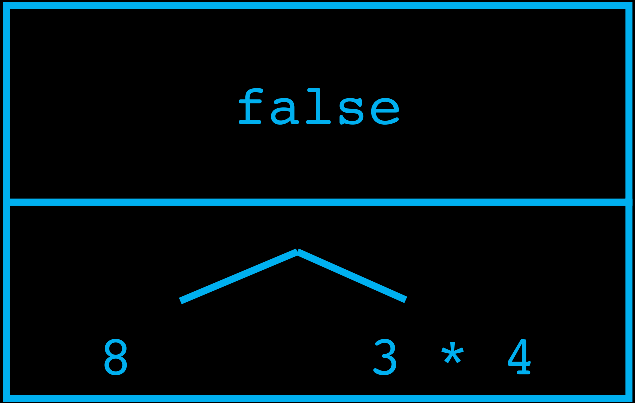
Conditional Expression

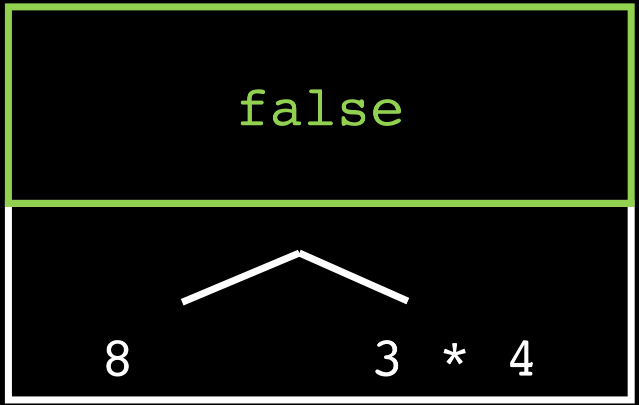
```
false ? 8 : 3 * 4;
```

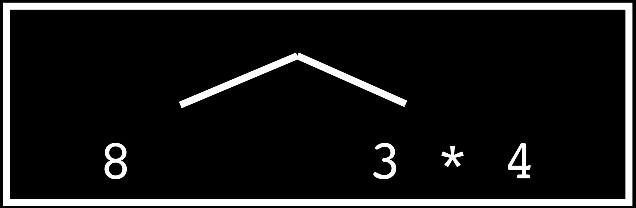
```
Conditional expression:
    pop conditional from agenda
    push branch on agenda
    push predicate on agenda
```

```
false ? 8 : 3 * 4;
```

```
Conditional expression:
    pop conditional from agenda
    push branch on agenda
    push predicate on agenda
```



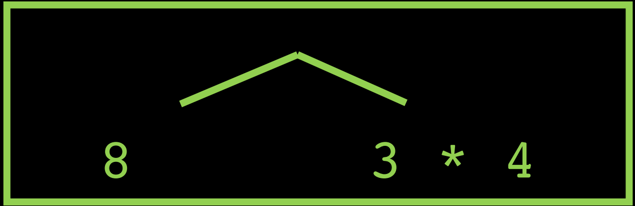

false
8
3 * 4



```
false
```

Branch instruction:

pop Boolean value from stash
pop branch instruction from agenda
push correct alternative on agenda



`false`

Branch instruction:

pop Boolean value from stash
pop branch instruction from agenda
push correct alternative on agenda

```
3 * 4
```

…eventually

Agenda empty:

    done!
    result is on top of stash

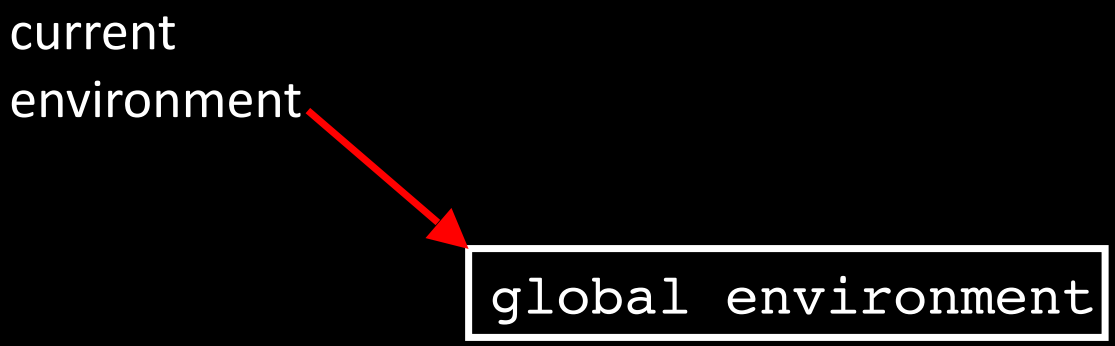
The journey

- Calculator language
- Add conditionals, Booleans, sequences
- **Add blocks, declarations, names**
- Add function declaration and application (simple return)
- Restoring environments
- Further language features

Block

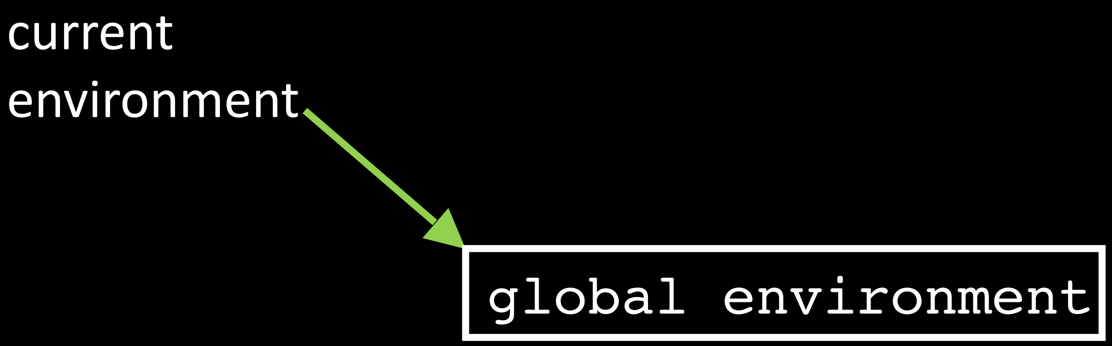
```
{
   const x = 3 * 4;
   const y = x + 2;
   x * y;
}
```

Block



```
{
    const x = 3 * 4;
    const y = x + 2;
    x * y;
}
```

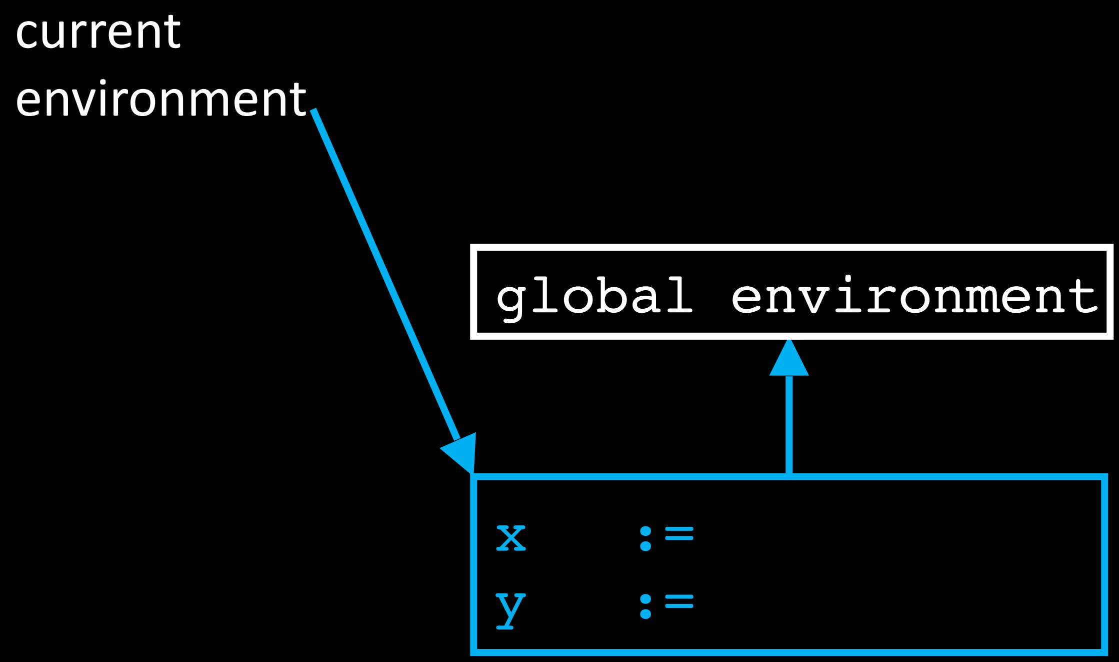
Block



```
Block:
    extend current environment by
        frame with declared names
    set current environment to
        start at new frame
    pop block from agenda
    push body of block on agenda
```

```
{
    const x = 3 * 4;
    const y = x + 2;
    x * y;
}
```

Block

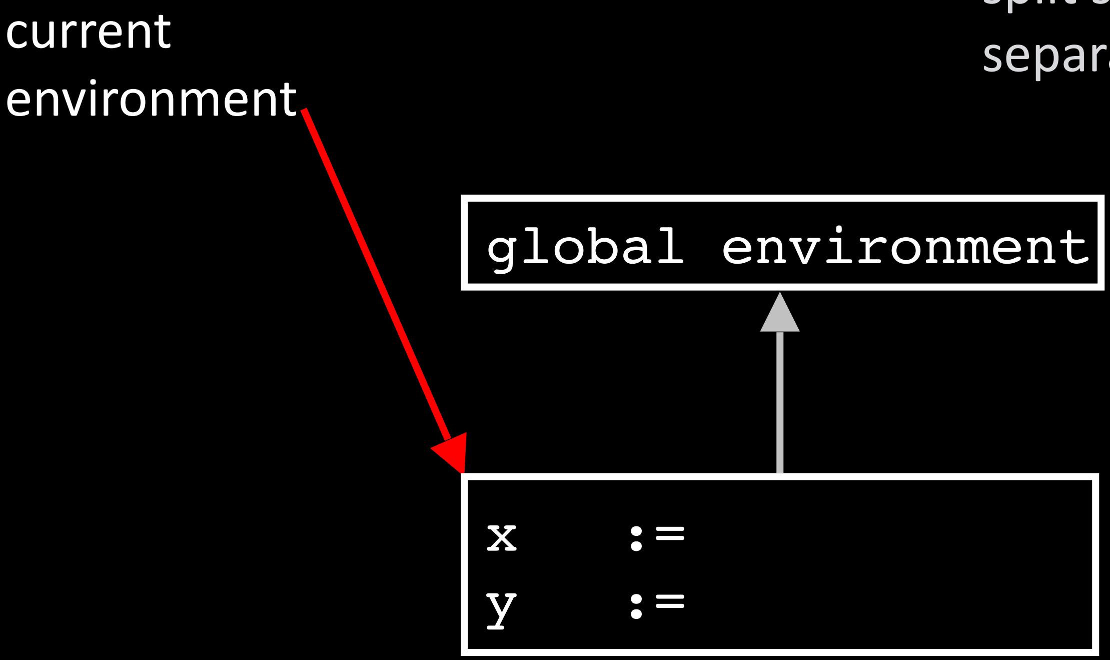


Block:
- extend current environment by frame with declared names
- set current environment to start at new frame
- pop block from agenda
- push body of block on agenda

```
const x = 3 * 4;
const y = x + 2;
x * y;
```

Sequence:

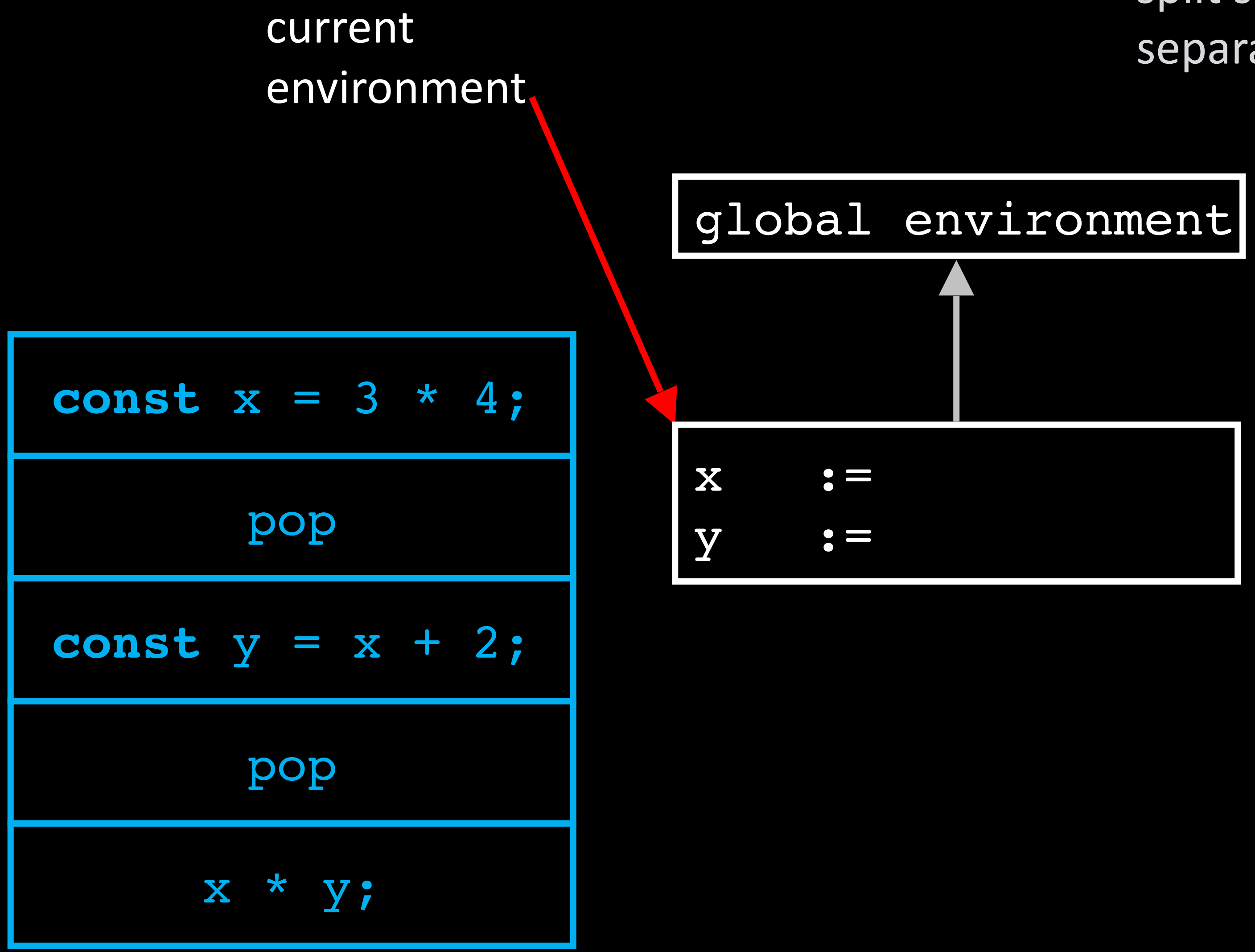
split sequence into components
separated by pop instructions



```
const x = 3 * 4;
const y = x + 2;
x * y;
```

Sequence:

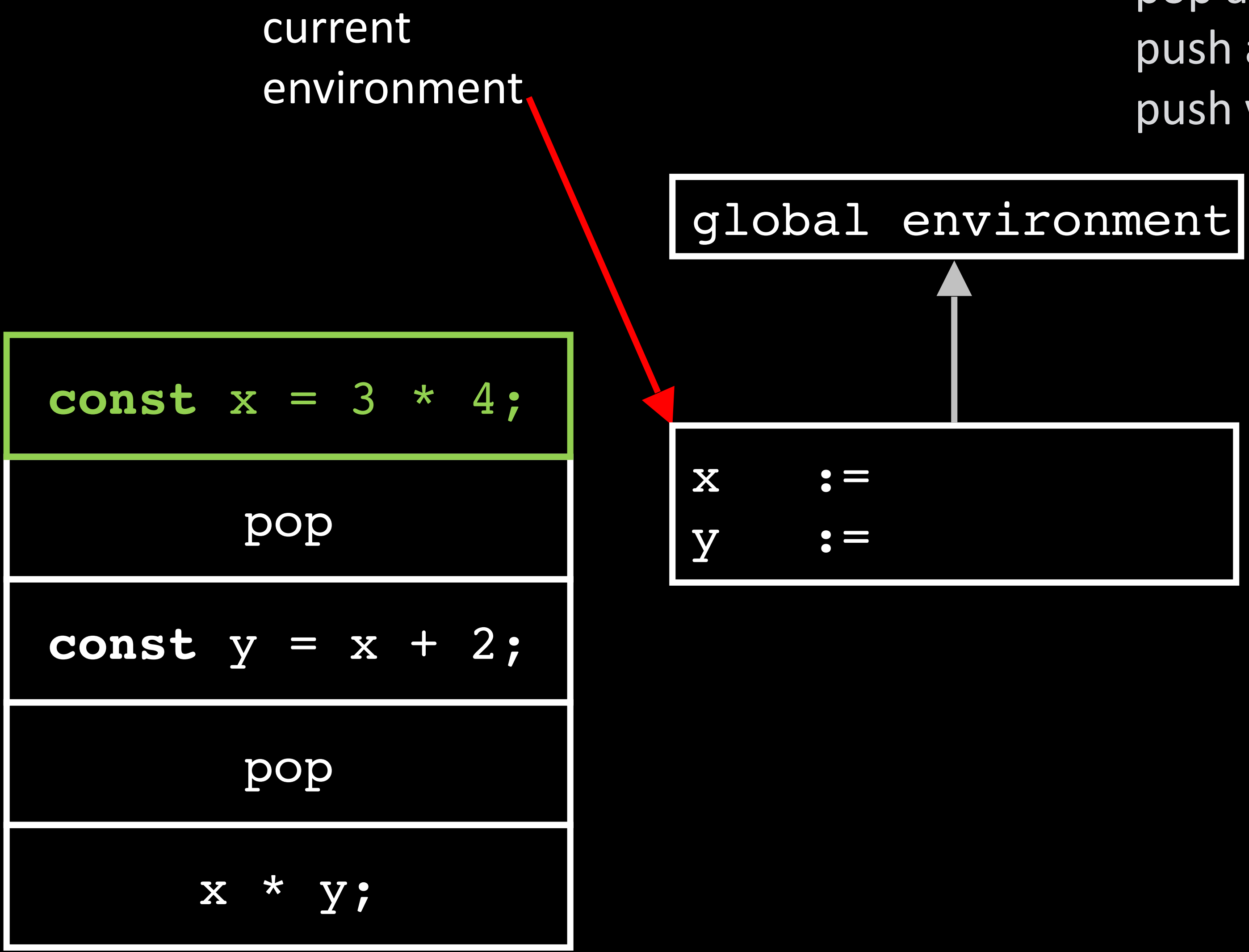
split sequence into components
separated by pop instructions



Assignment

Assignment:

pop assignment from agenda
push assign instruction on agenda
push value expression on agenda



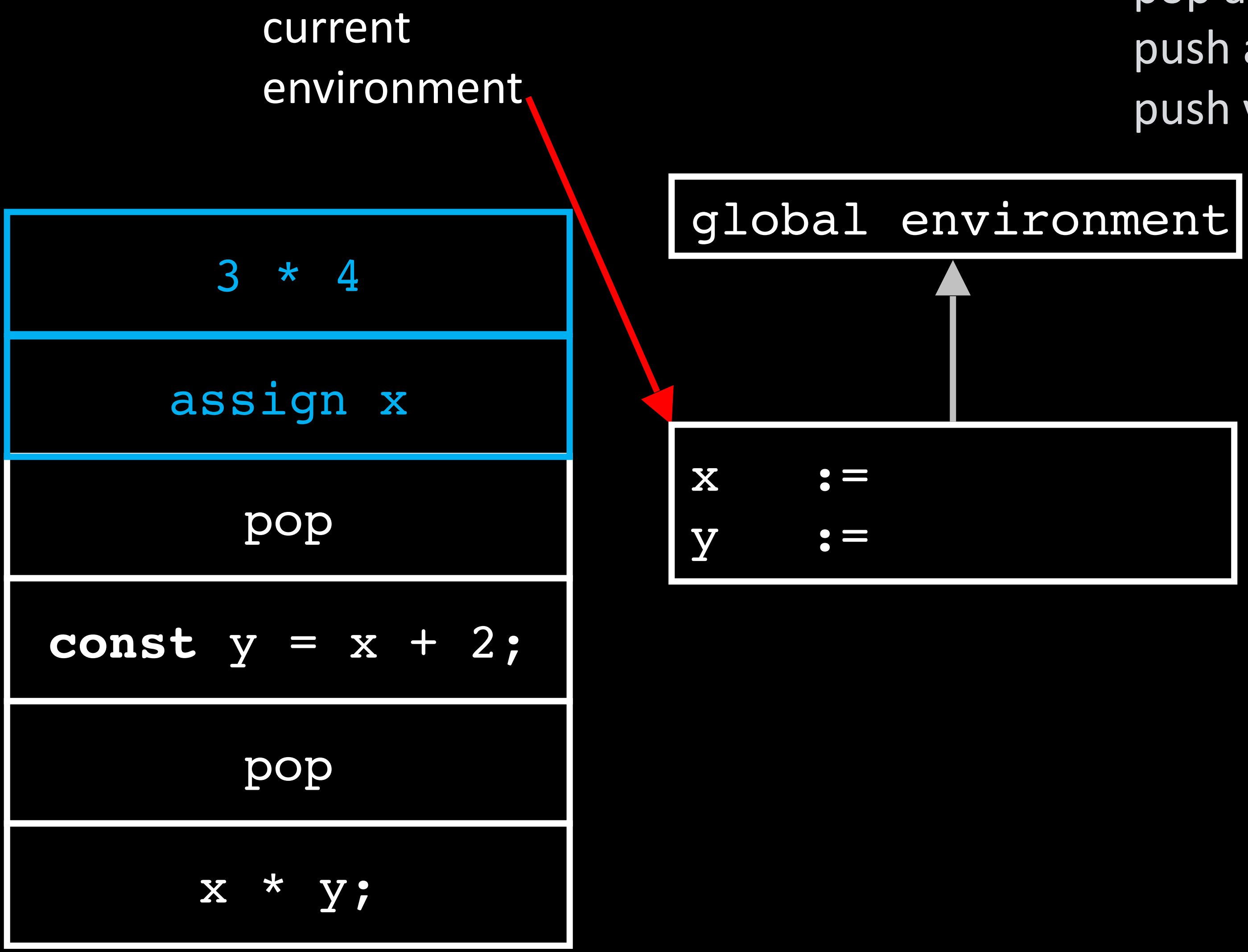
Assignment

Assignment:

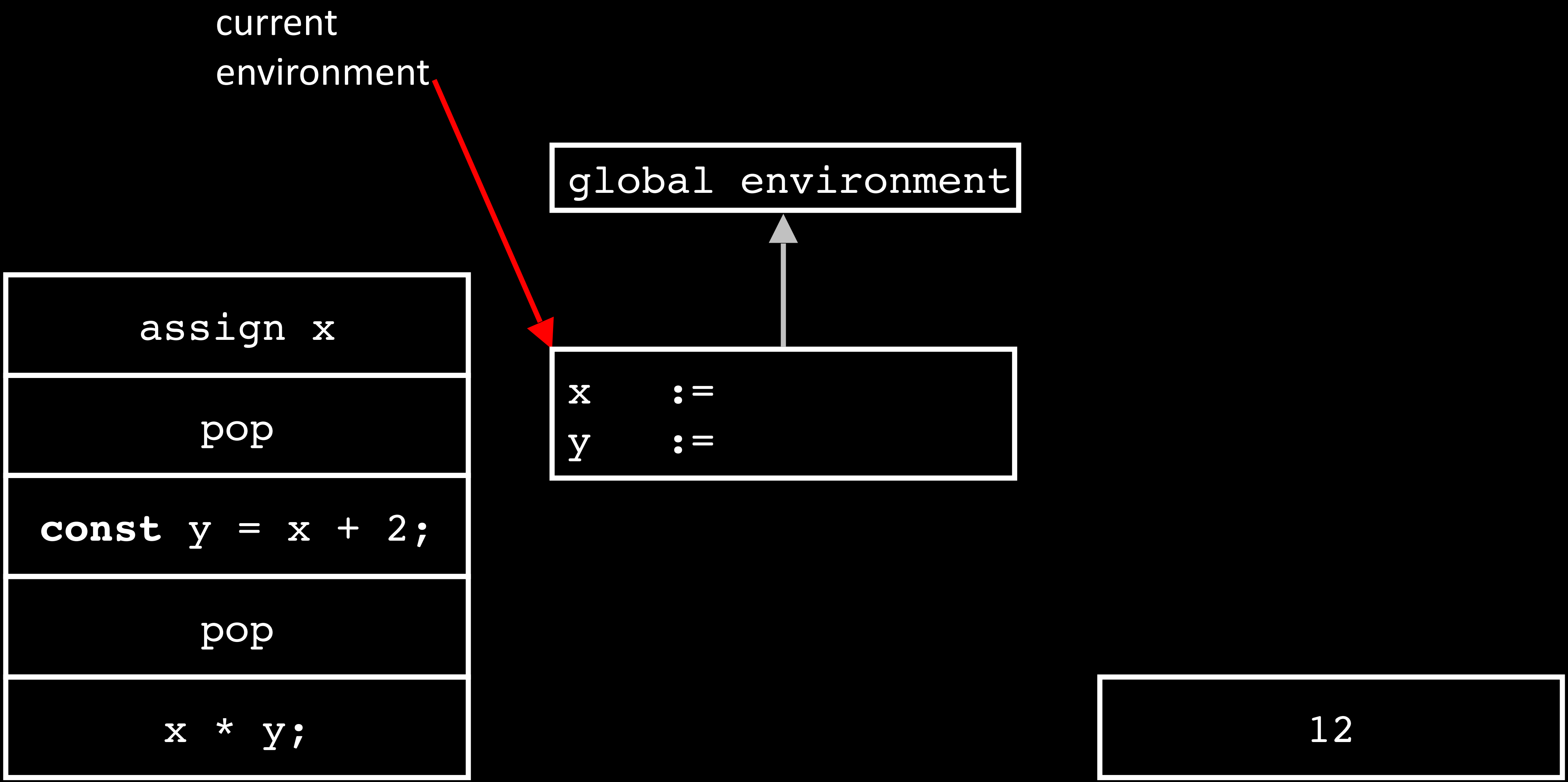
pop assignment from agenda

push assign instruction on agenda

push value expression on agenda

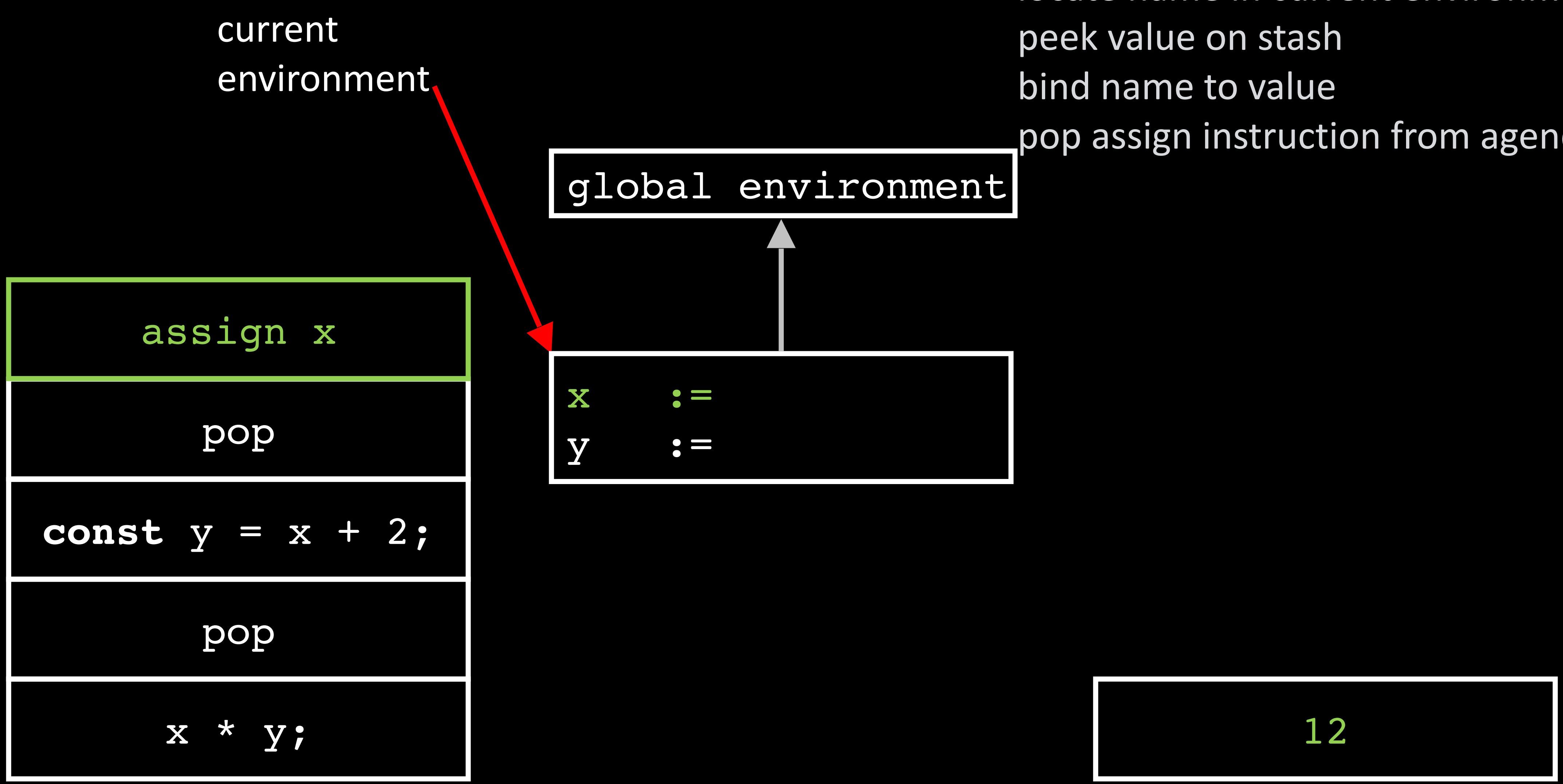


...eventually


current environment
global environment
x :=
y :=
assign x
pop
const y = x + 2;
pop
x * y;
12

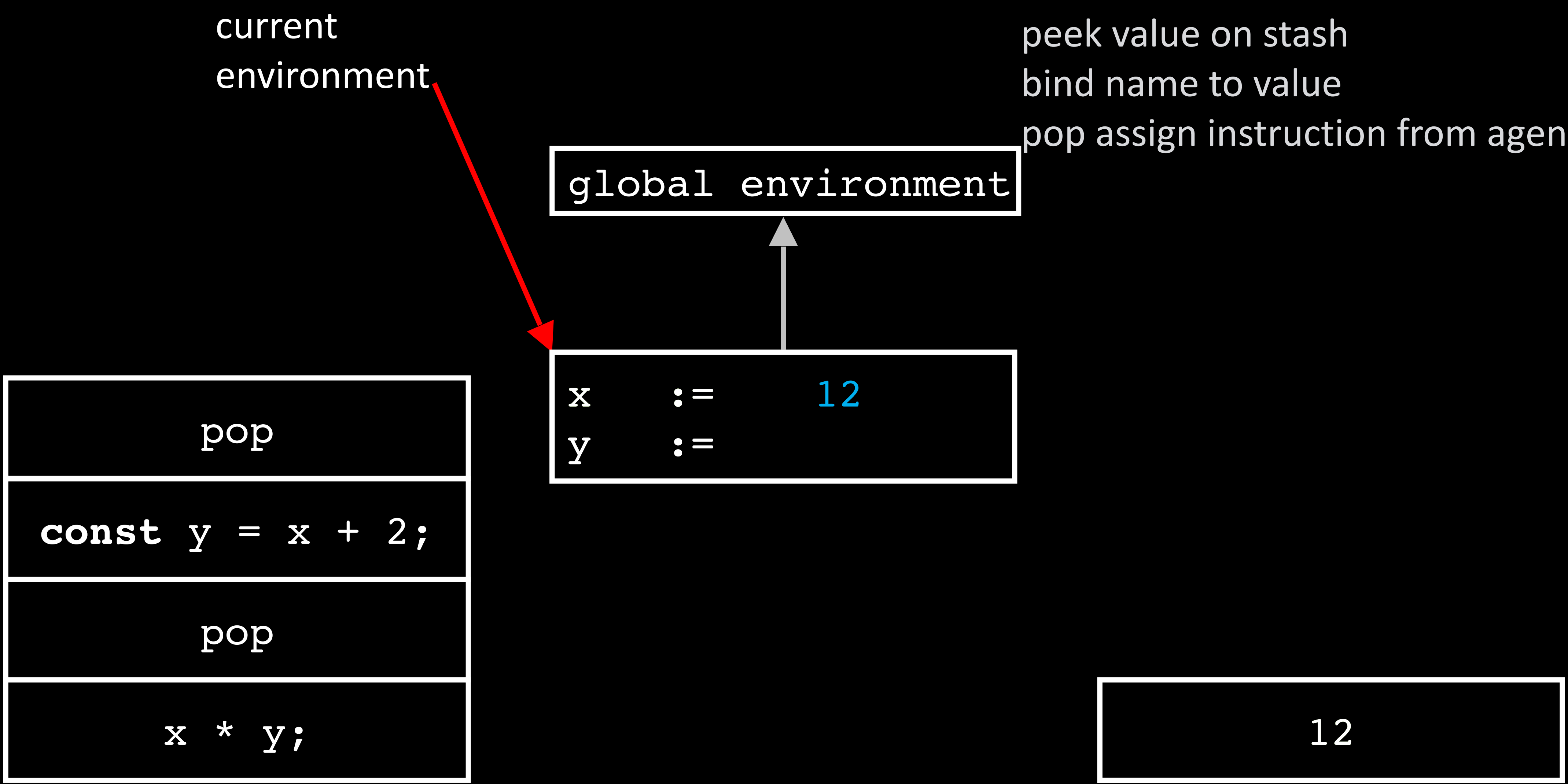
Assign instruction

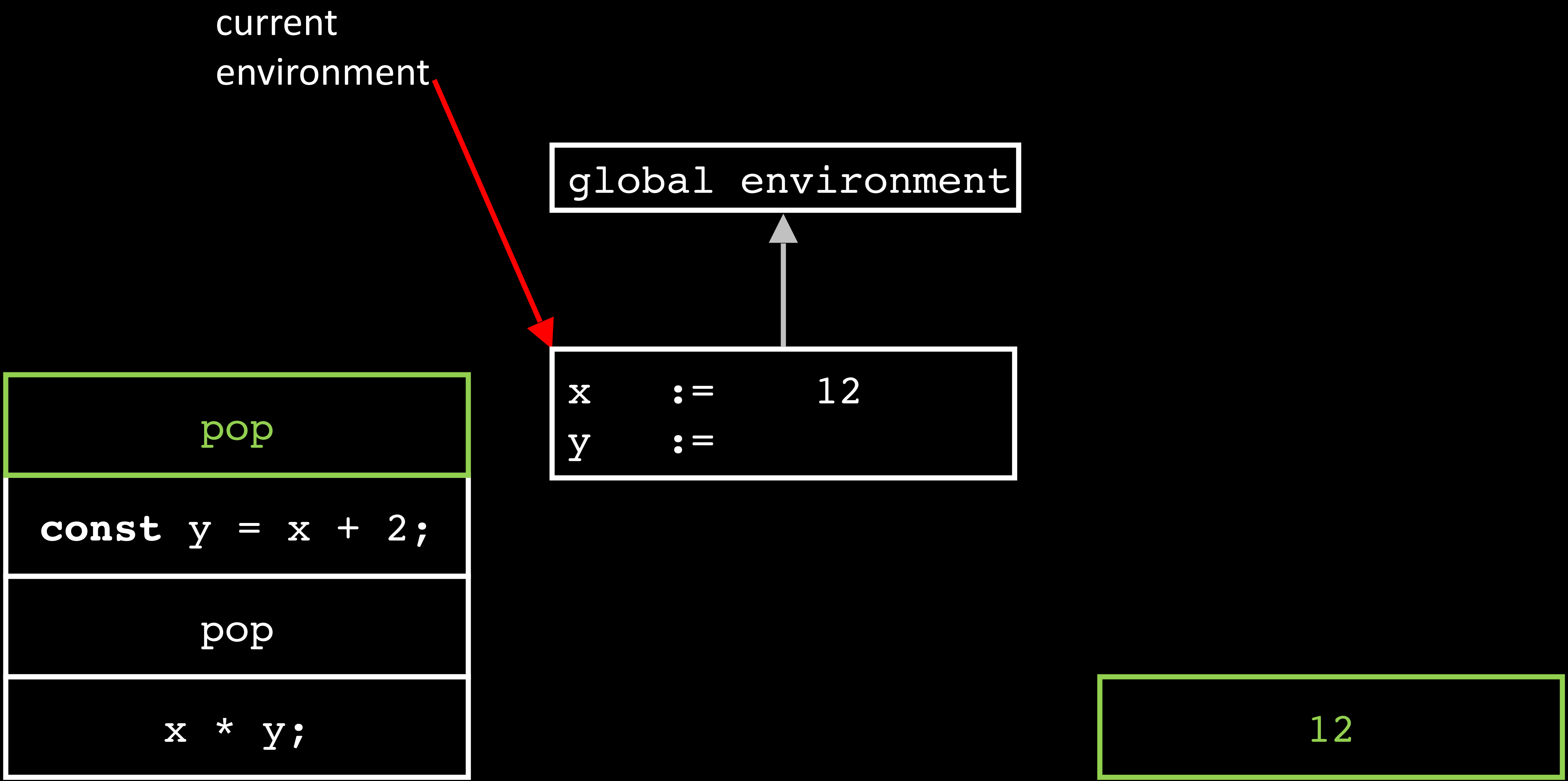
Assign instruction:
- locate name in current environment
- peek value on stash
- bind name to value
- pop assign instruction from agenda



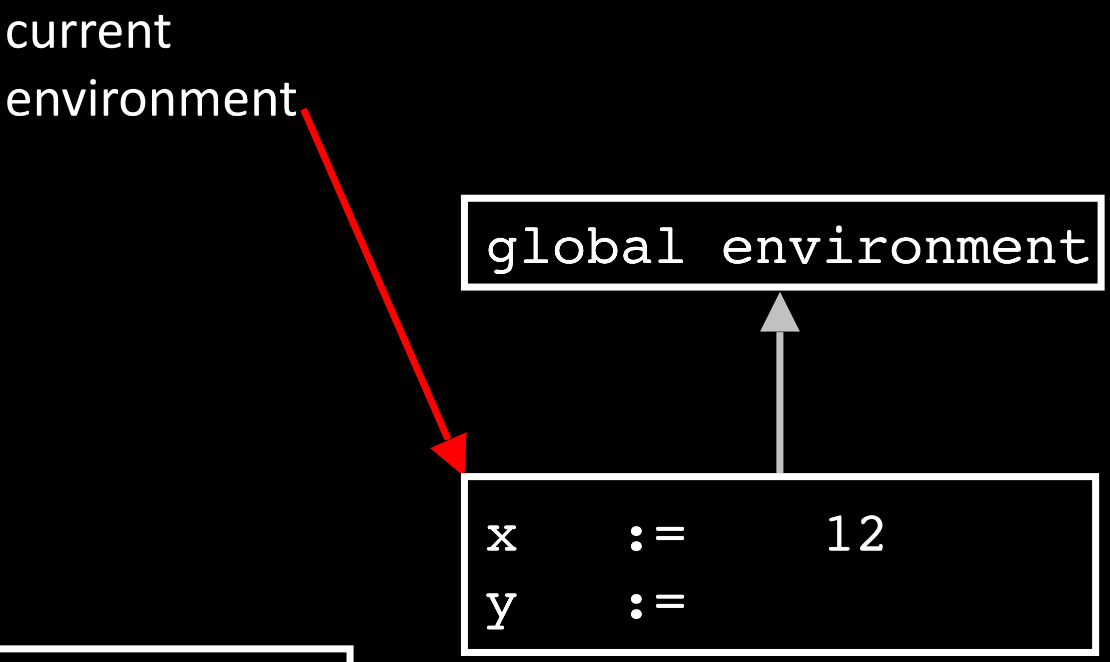
Assign instruction

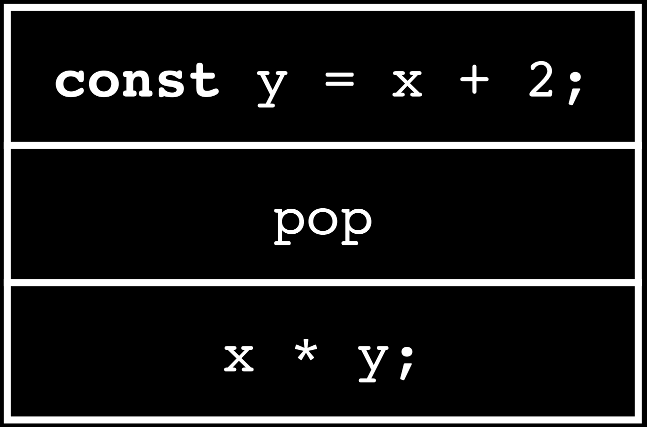
Assign instruction:
- locate name in current environment
- peek value on stash
- bind name to value
- pop assign instruction from agenda





current environment
global environment
x := 12
y :=
pop
const y = x + 2;
pop
x * y;
12


current environment
global environment
x := 12
y :=


const y = x + 2;
pop
x * y;

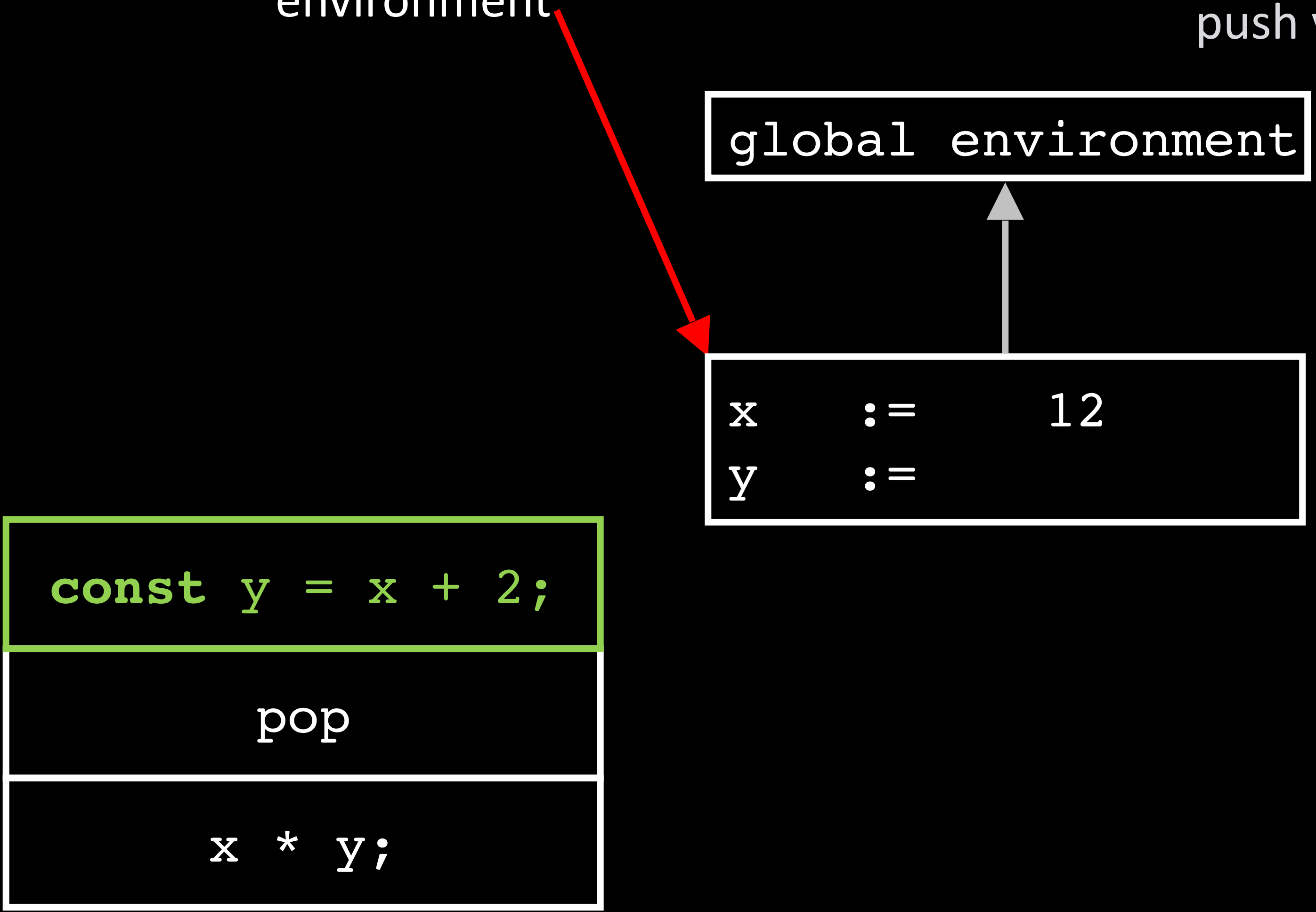
Assignment

Assignment:

pop assignment from agenda

push assign instruction on agenda

push value expression on agenda



```
const y = x + 2;
```

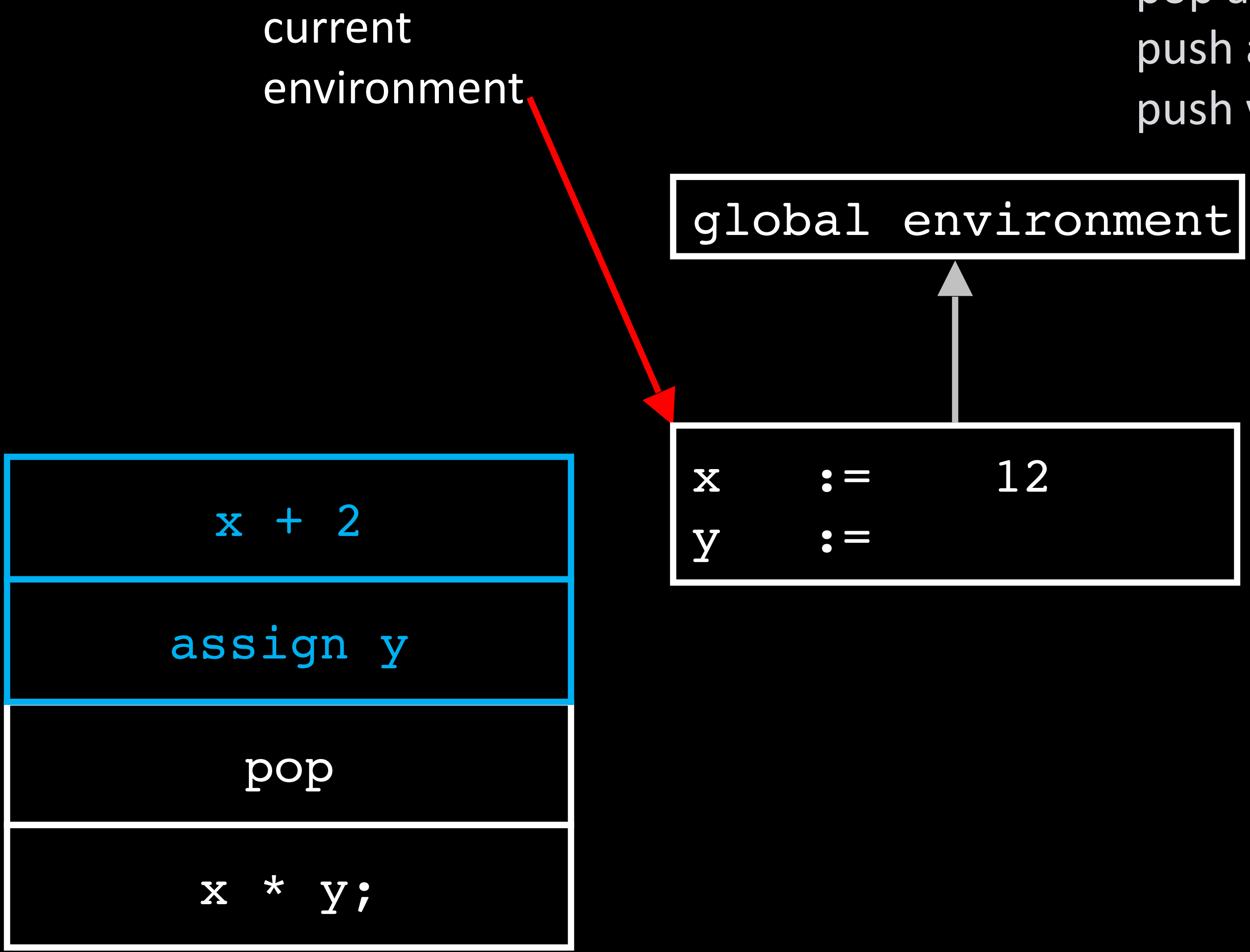
```
pop
```

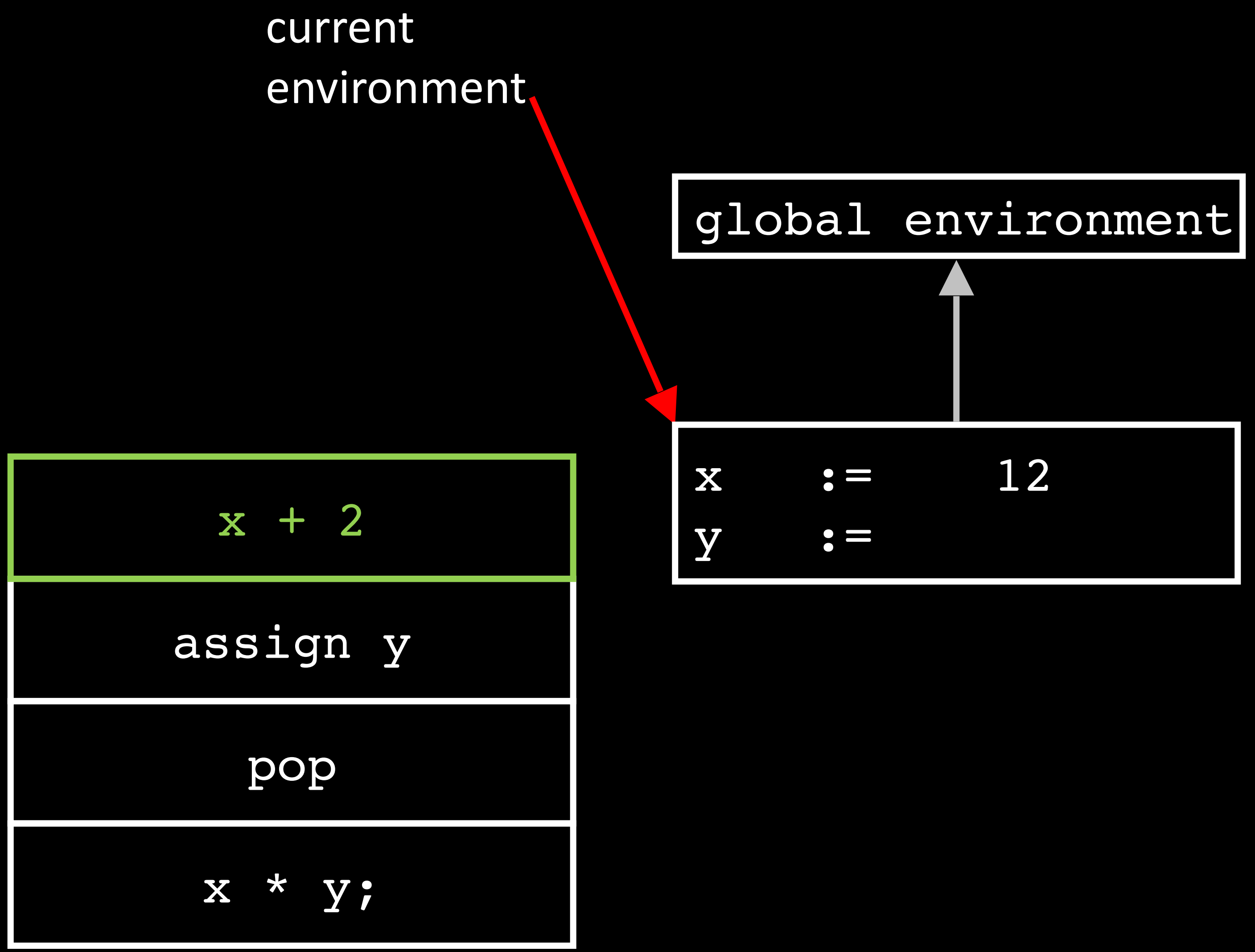
```
x * y;
```

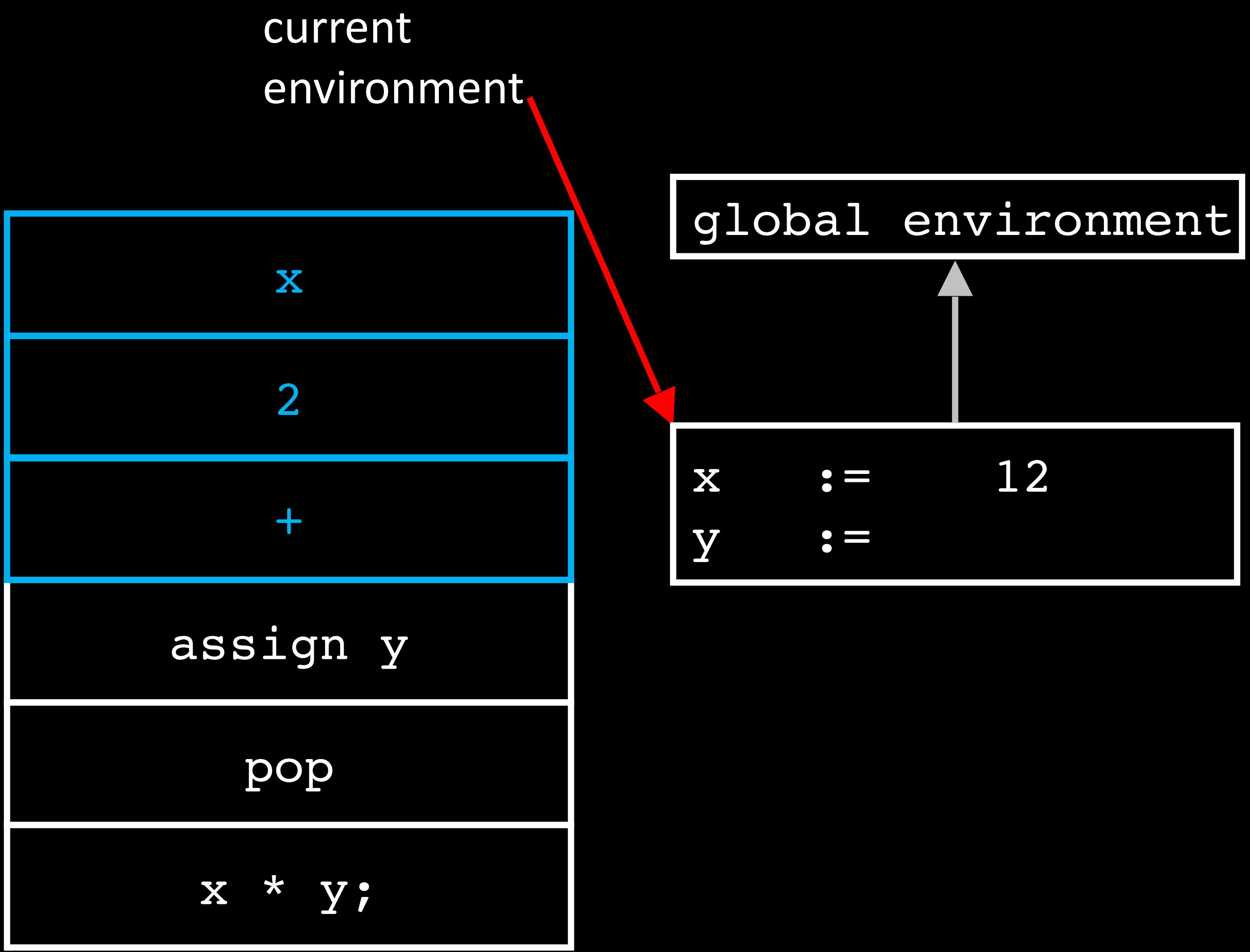
Assignment

Assignment:

- pop assignment from agenda
- push assign instruction on agenda
- push value expression on agenda

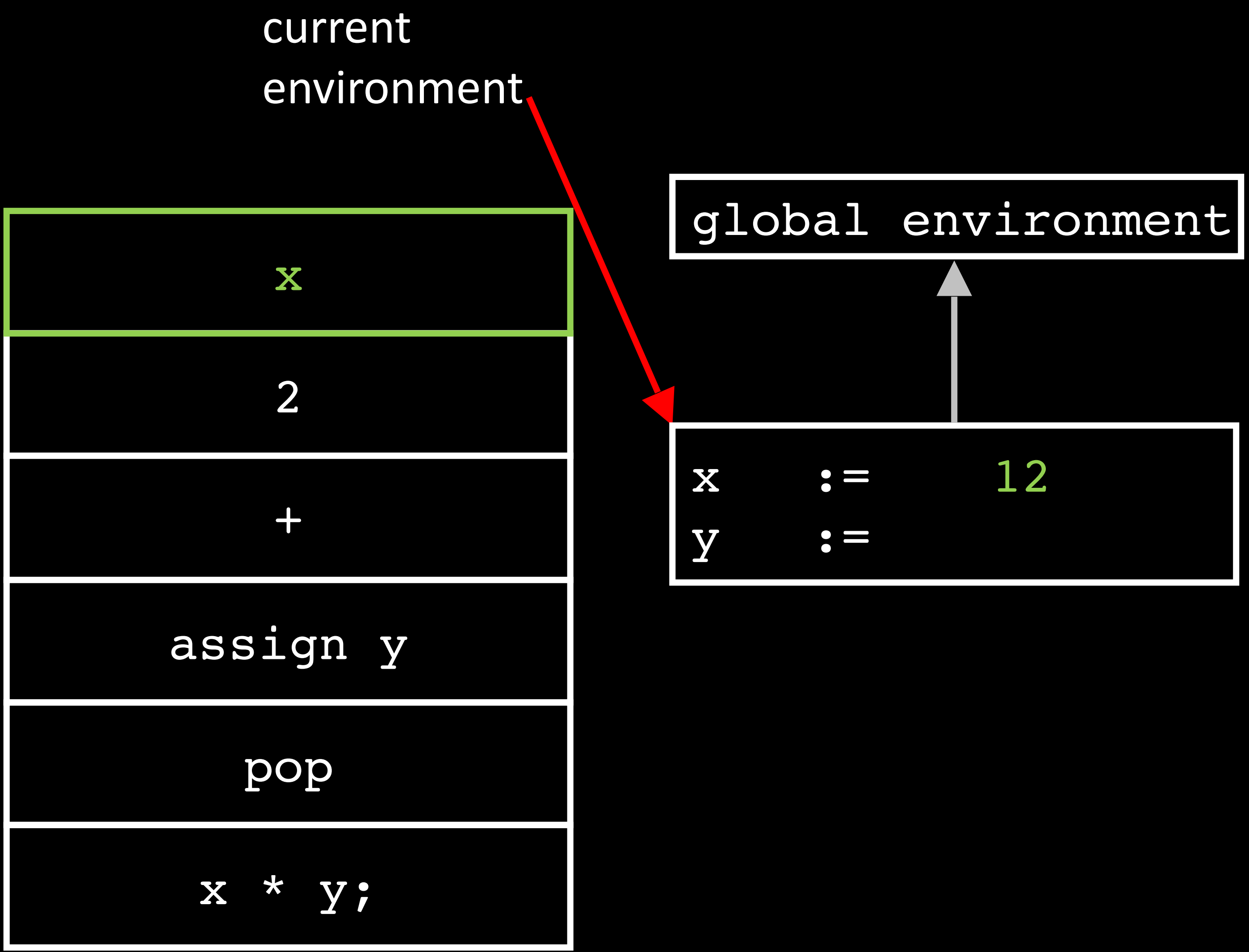



current environment
global environment
x := 12
y :=
x + 2
assign y
pop
x * y;


current environment
x
2
+
assign y
pop
x * y;
global environment
x := 12
y :=

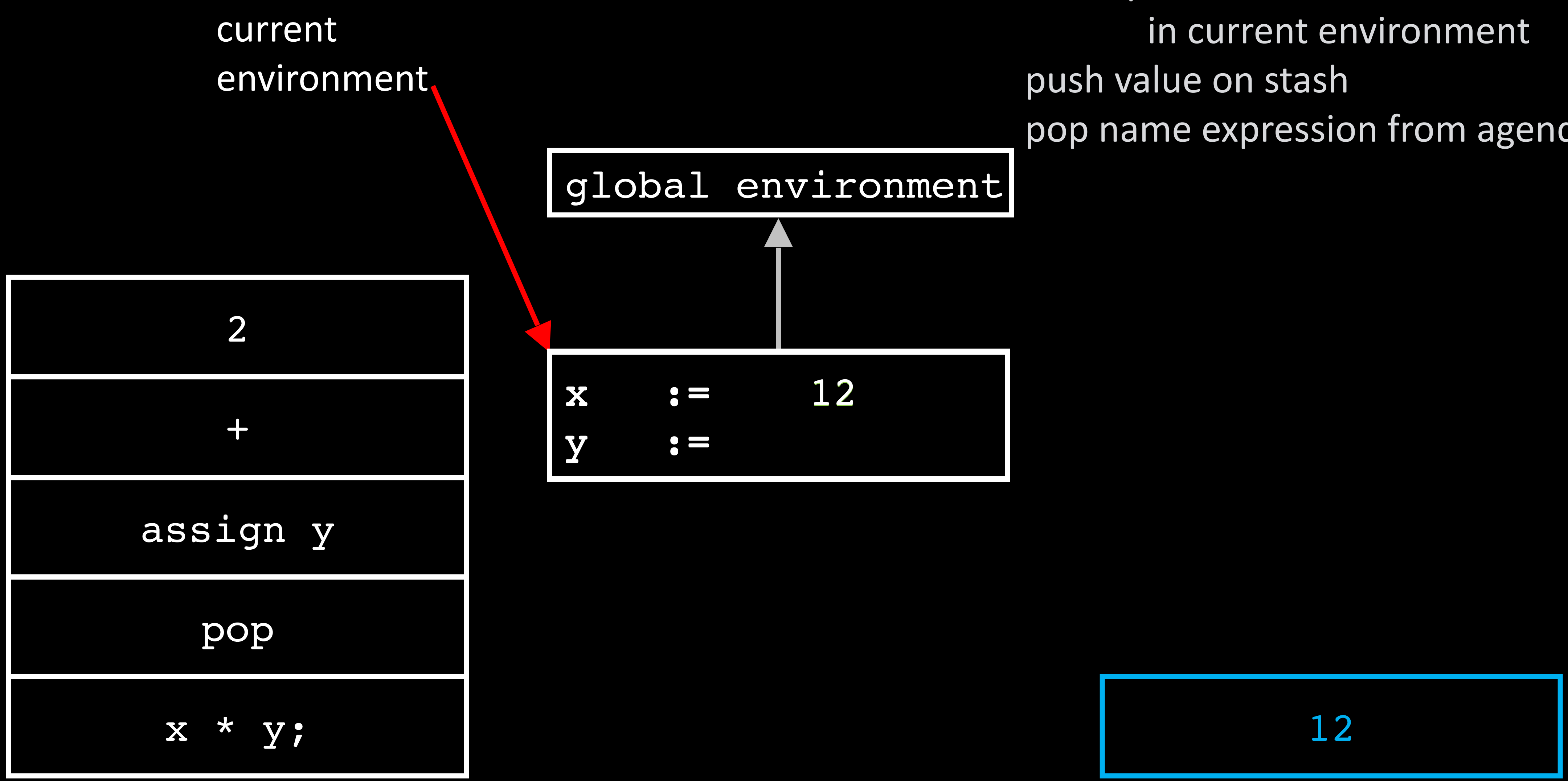
Name expression

Name expression:
- look up value of name in current environment
- push value on stash
- pop name expression from agenda

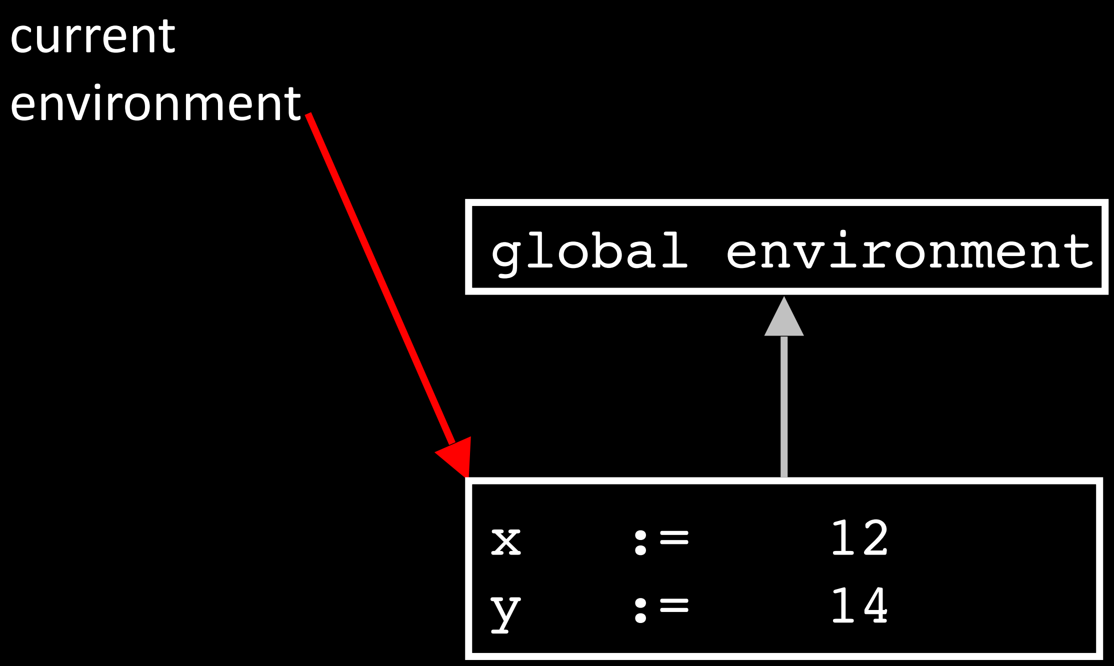


Name expression

```
Name expression:
    look up value of name
            in current environment
    push value on stash
    pop name expression from agenda
```



Name expressions ...eventually



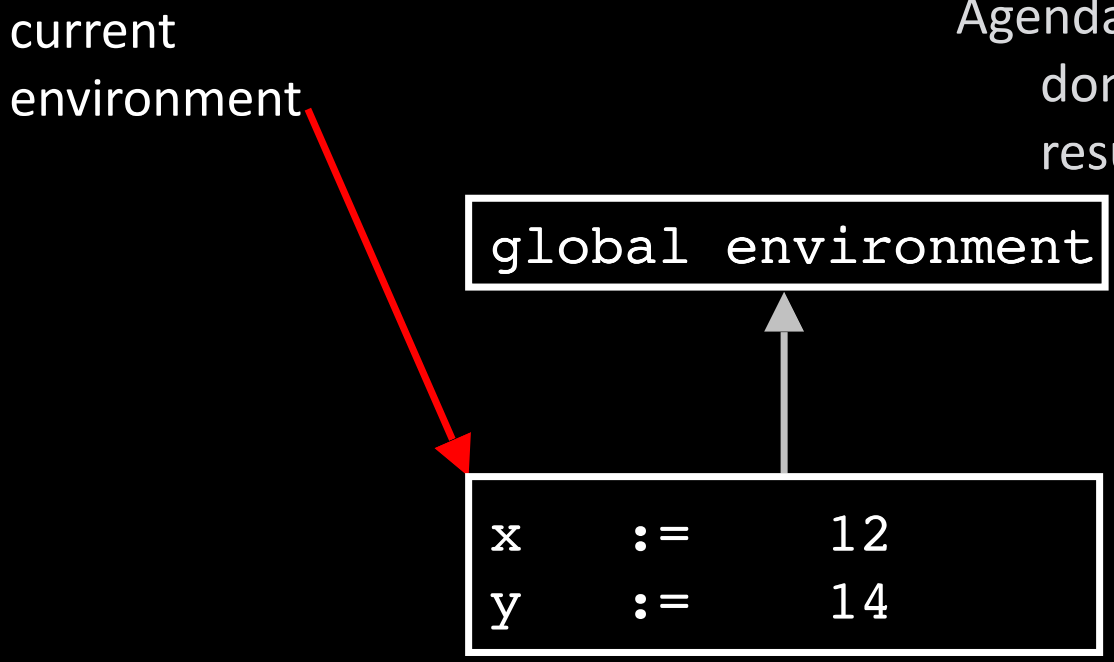
```
x * y;
```

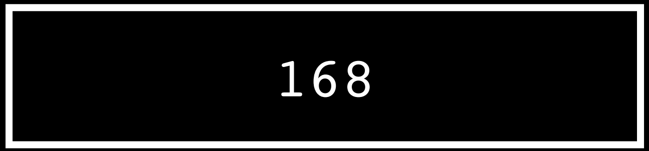
...eventually

Agenda empty:

done!

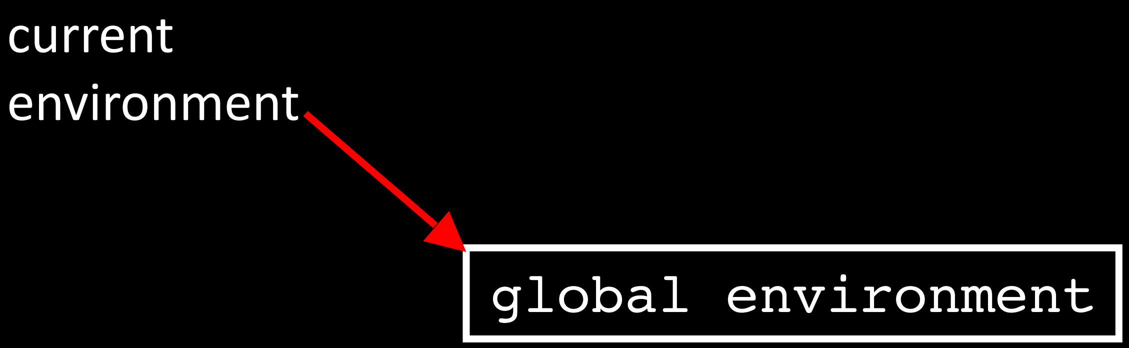
result is on top of stash





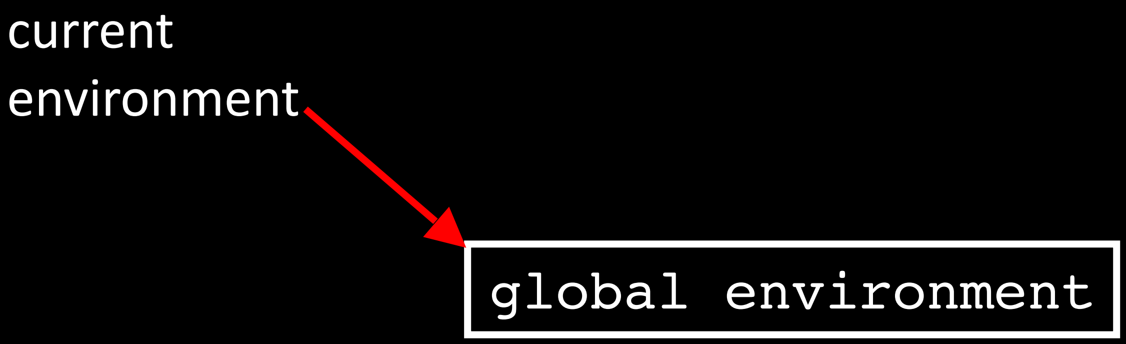
The journey

- Calculator language
- Add conditionals, Booleans, sequences
- Add blocks, declarations, names
- **Add function declaration and application (simple return)**
- Restoring environments
- Further language features



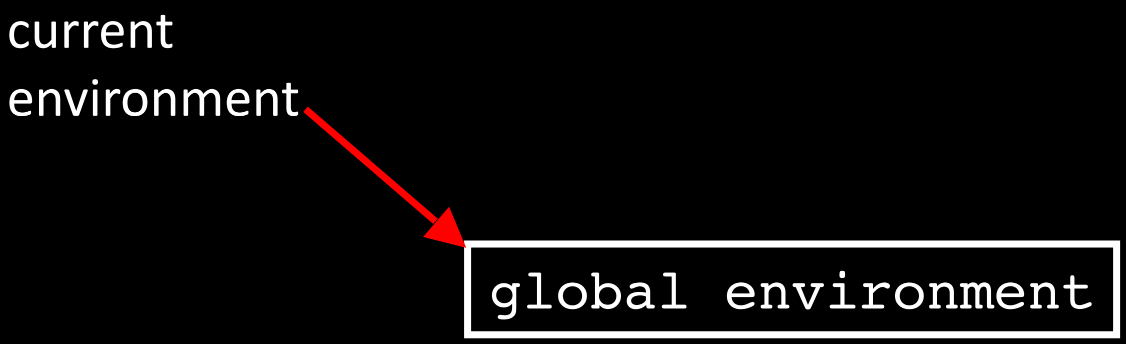
```
function fact(n) {
   return n === 1
   ? 1
   : n * fact(n - 1);
}
fact(4);
```

Every program is included in implicit top-level block



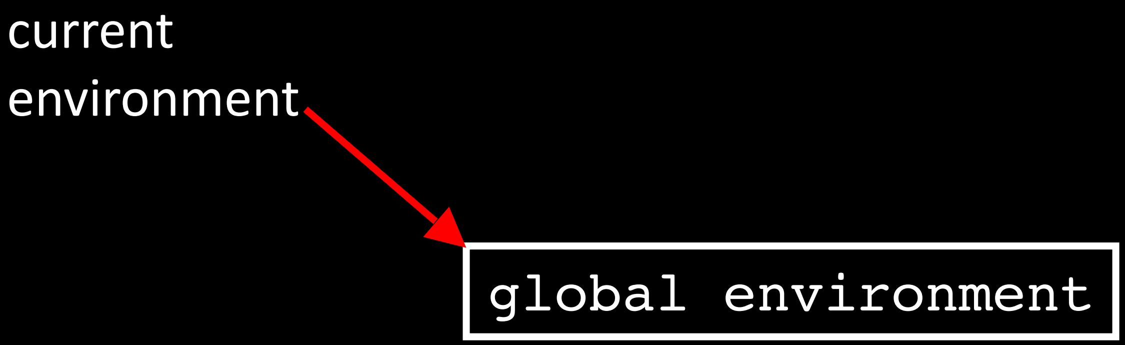
```
function fact(n) {
   return n === 1
   ? 1
   : n * fact(n - 1);
}
fact(4);
```

Every program is included in implicit top-level block



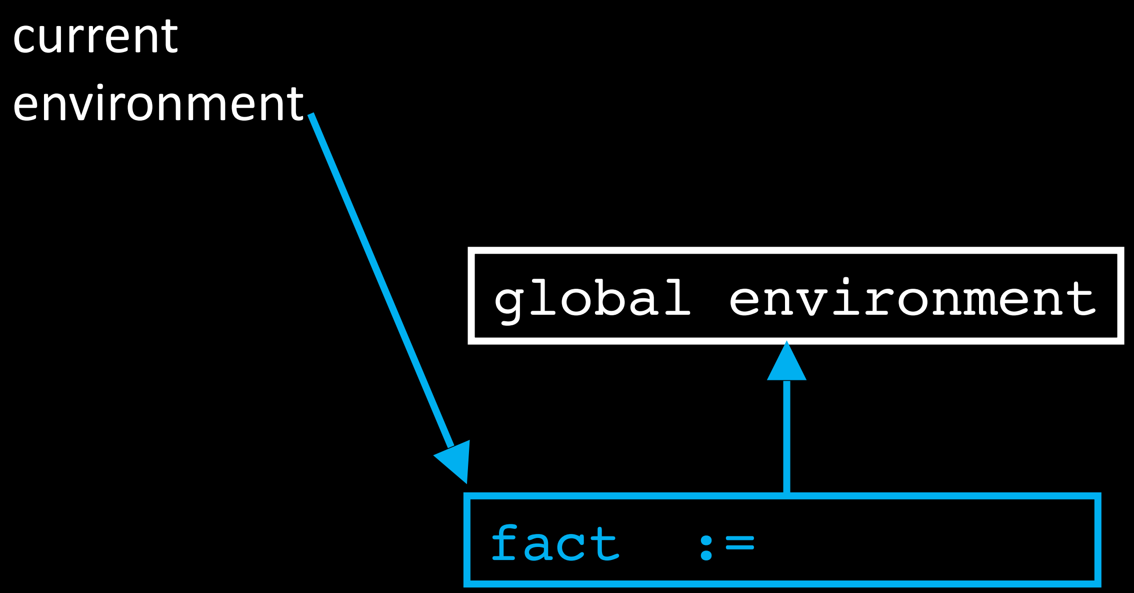
```
{
 function fact(n) {
   return n === 1
   ? 1
   : n * fact(n - 1);
 }
 fact(4);
}
```

Block

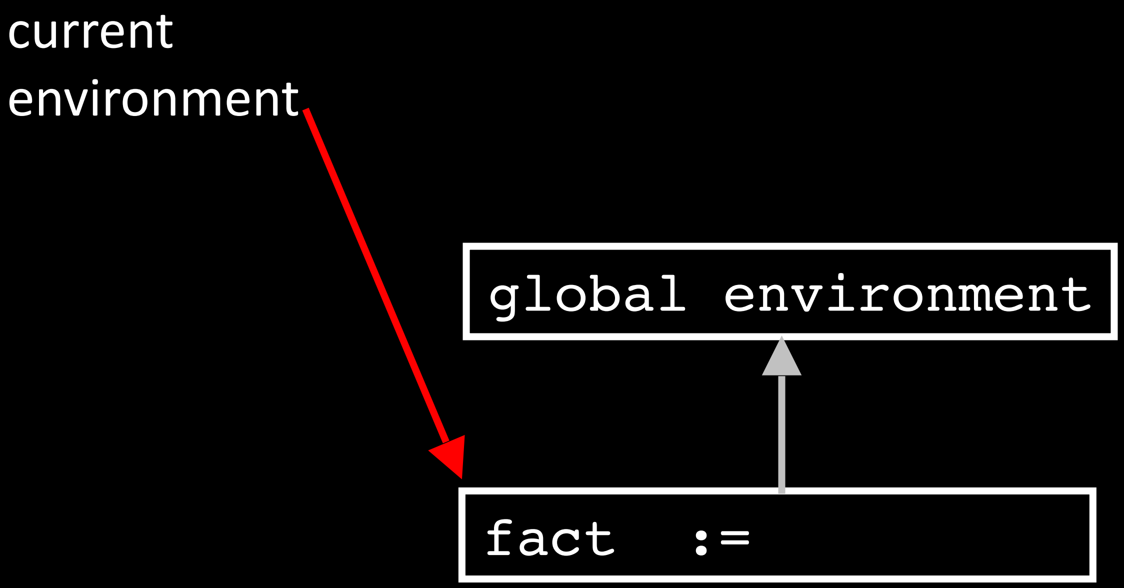


```
{
 function fact(n) {
   return n === 1
   ? 1
   : n * fact(n - 1);
 }
 fact(4);
}
```

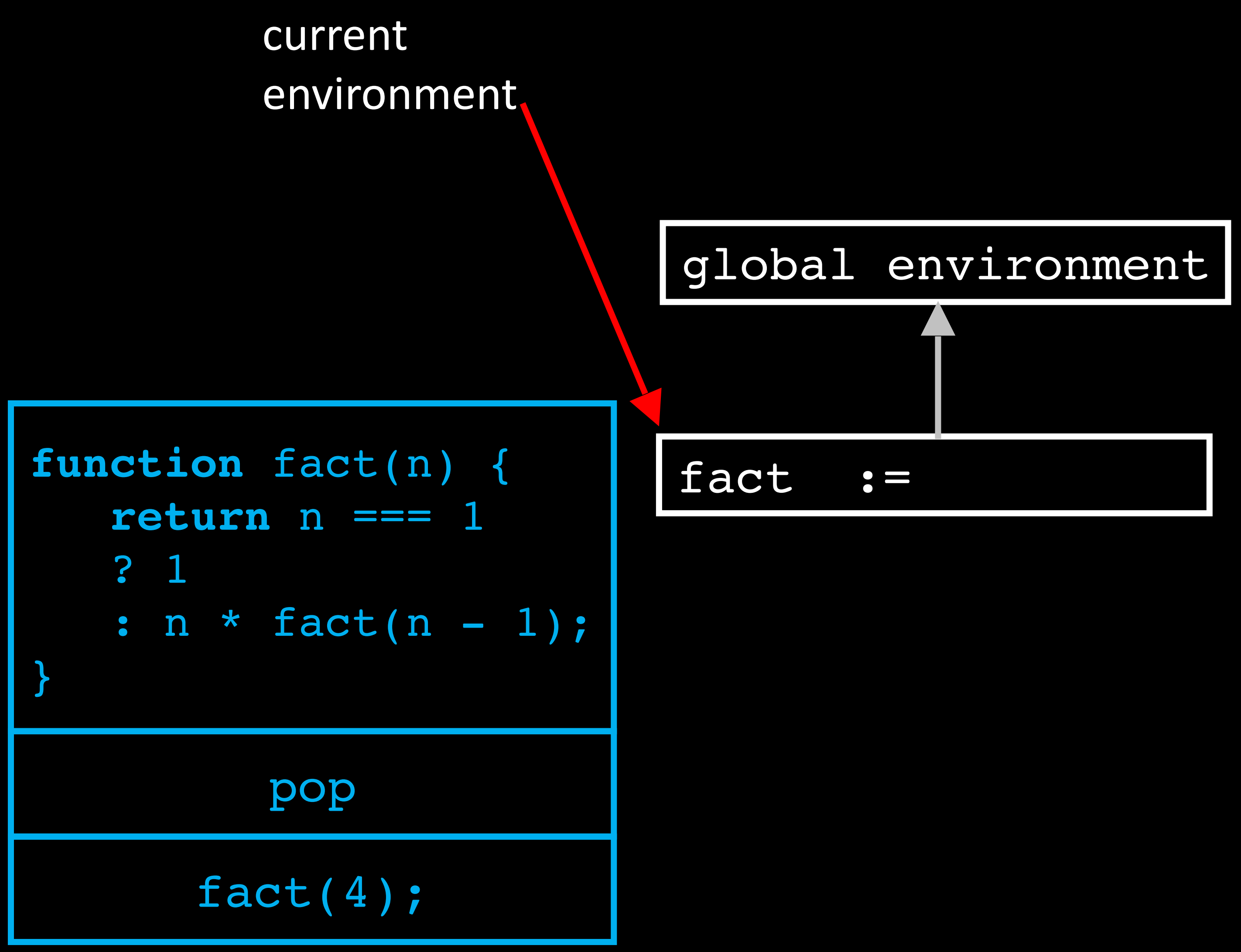
Block



```
function fact(n) {
   return n === 1
   ? 1
   : n * fact(n - 1);
}
fact(4);
```



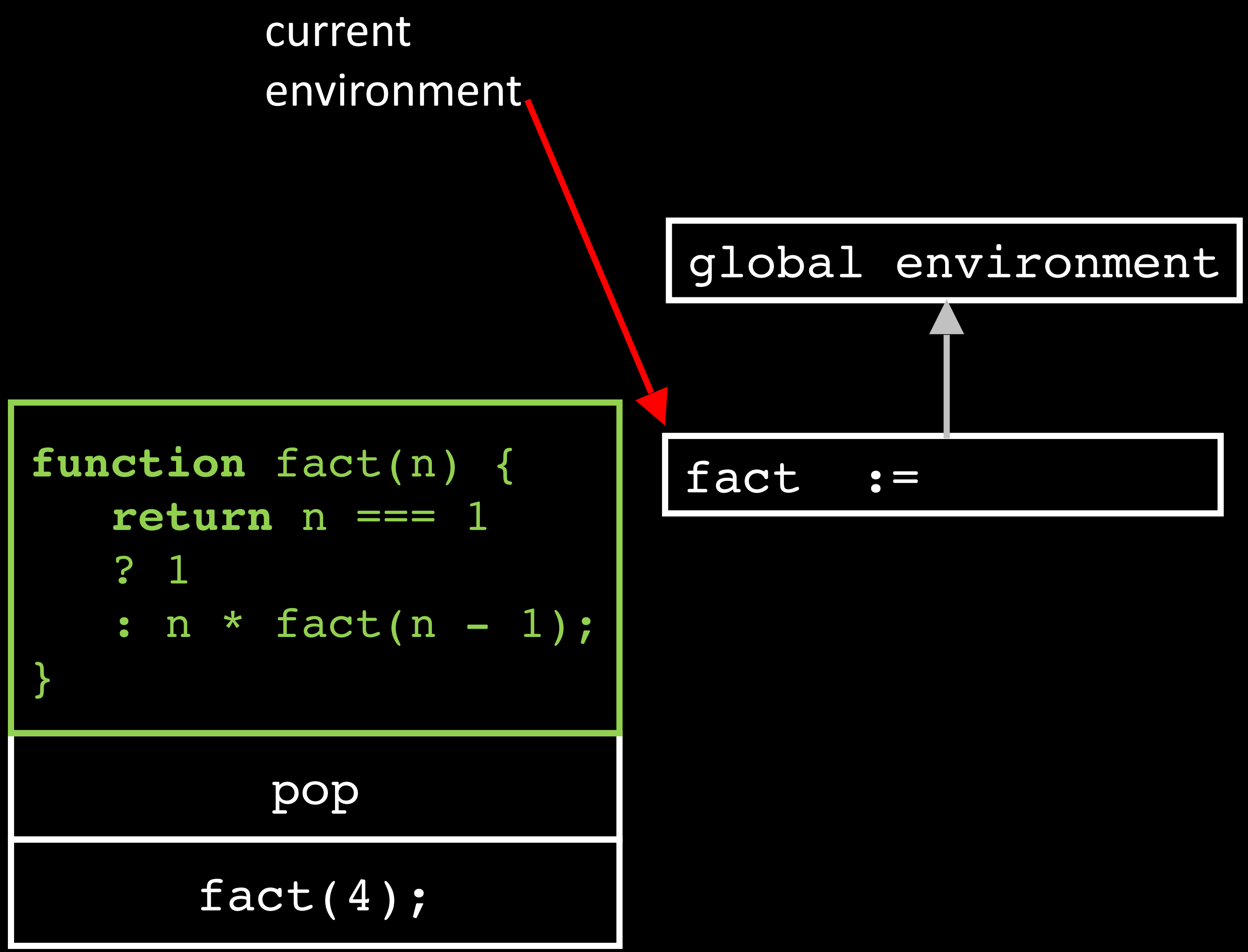
```
function fact(n) {
   return n === 1
   ? 1
   : n * fact(n - 1);
}
fact(4);
```



```
function fact(n) {
    return n === 1
    ? 1
    : n * fact(n - 1);
}
```

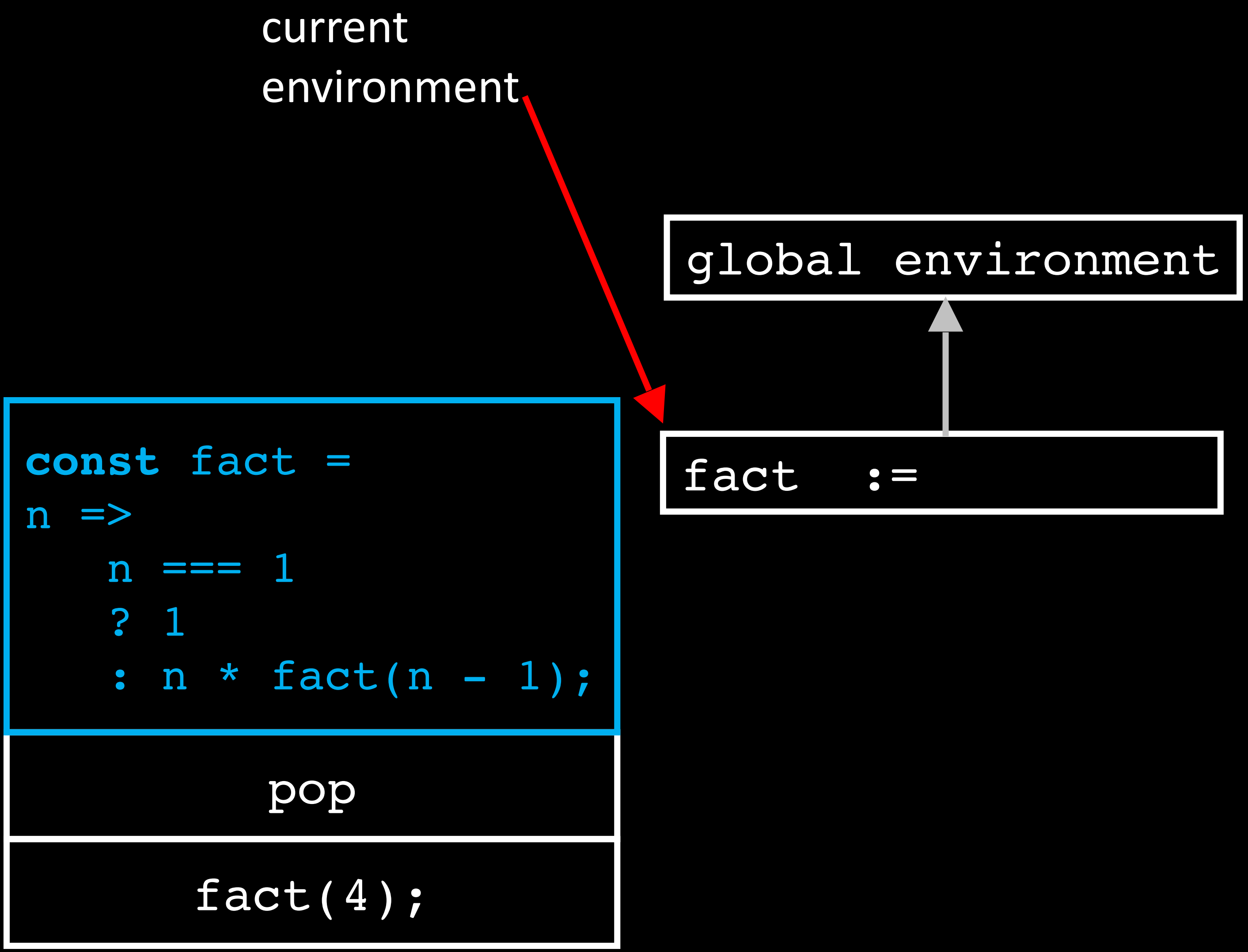
```
fact(4);
```

Desugaring function declaration



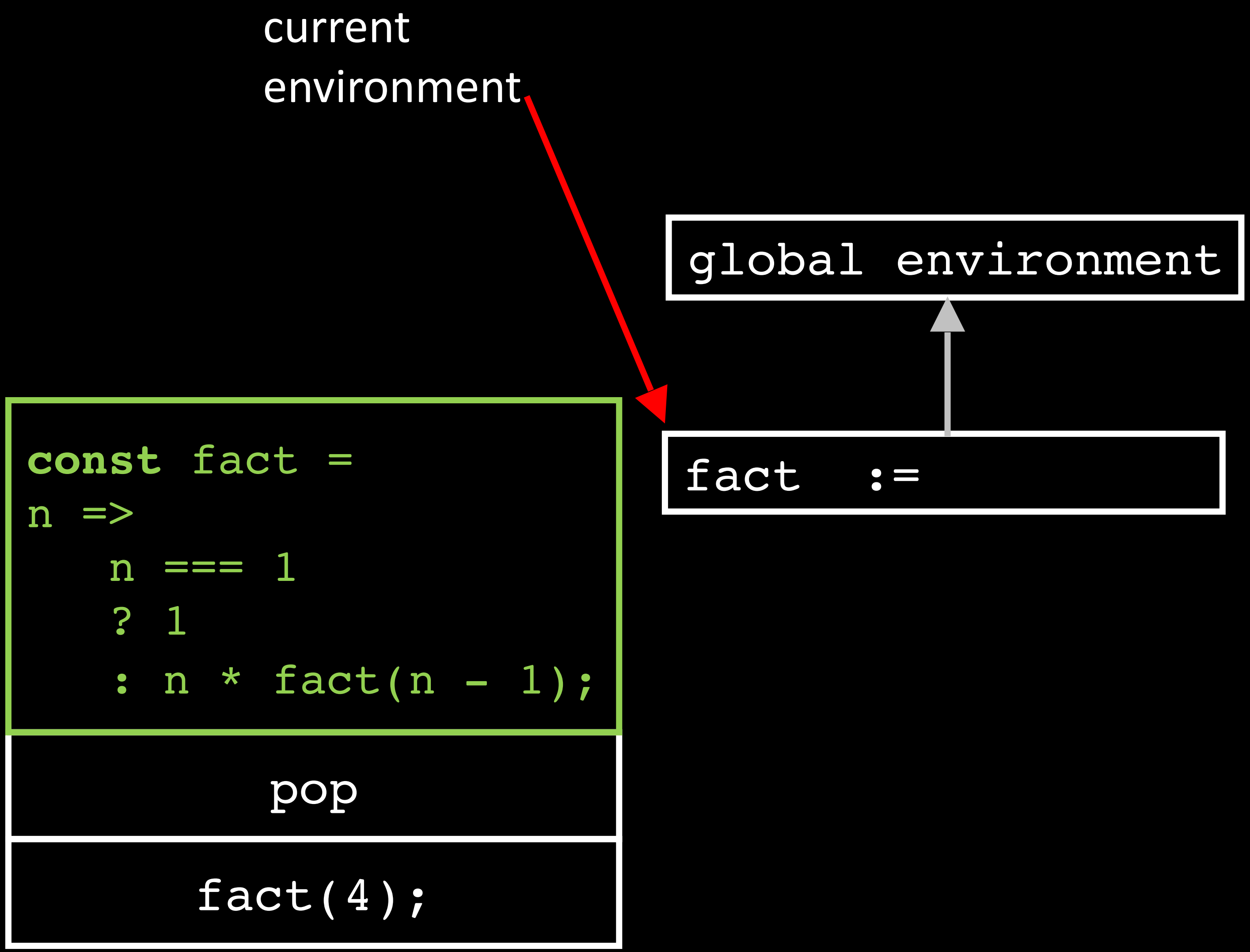
```
function fact(n) {
   return n === 1
   ? 1
   : n * fact(n - 1);
}
```

Desugaring function declaration



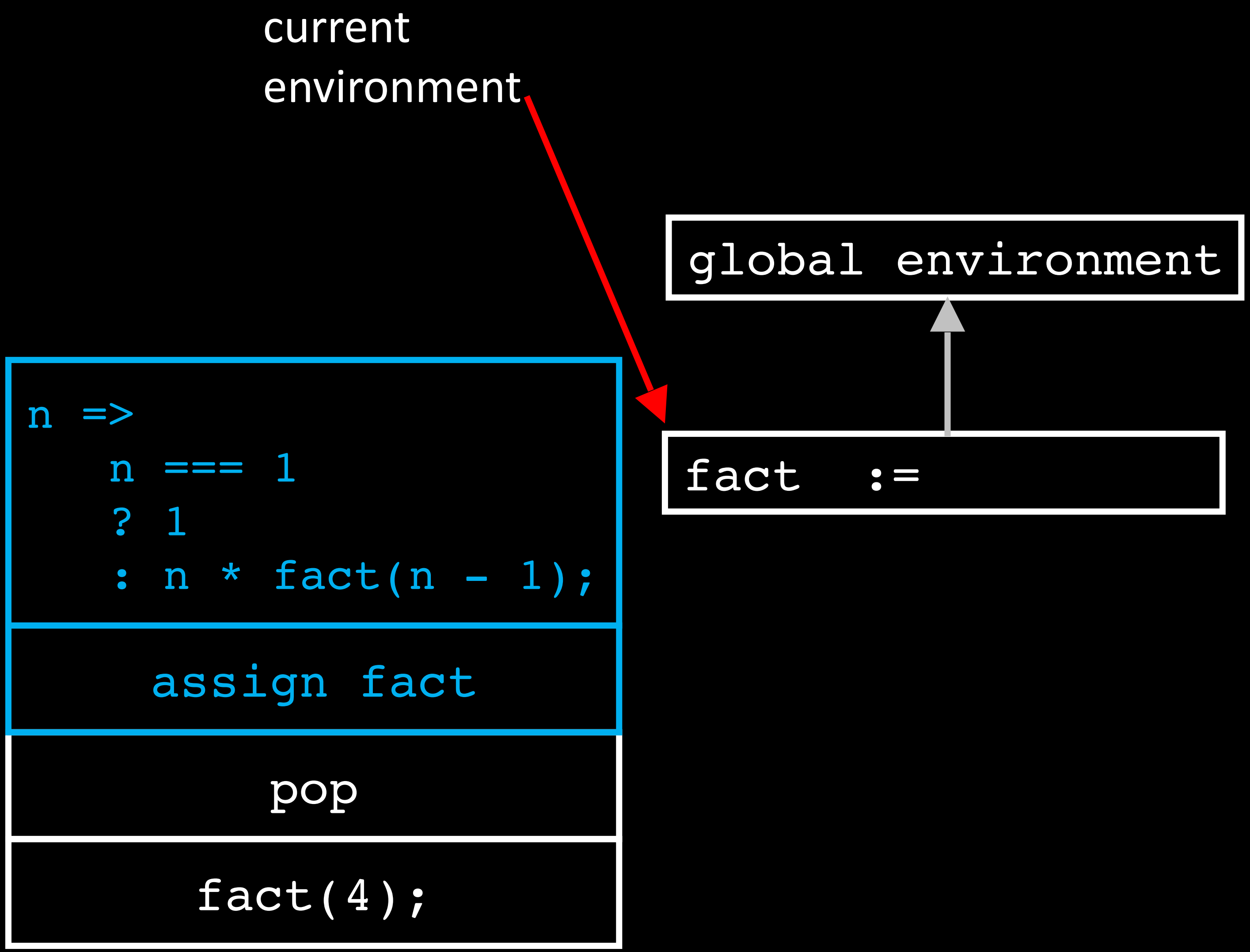
```
const fact =
n =>
    n === 1
    ? 1
    : n * fact(n - 1);
```

Constant declaration



```
const fact =
n =>
   n === 1
   ? 1
   : n * fact(n - 1);
```

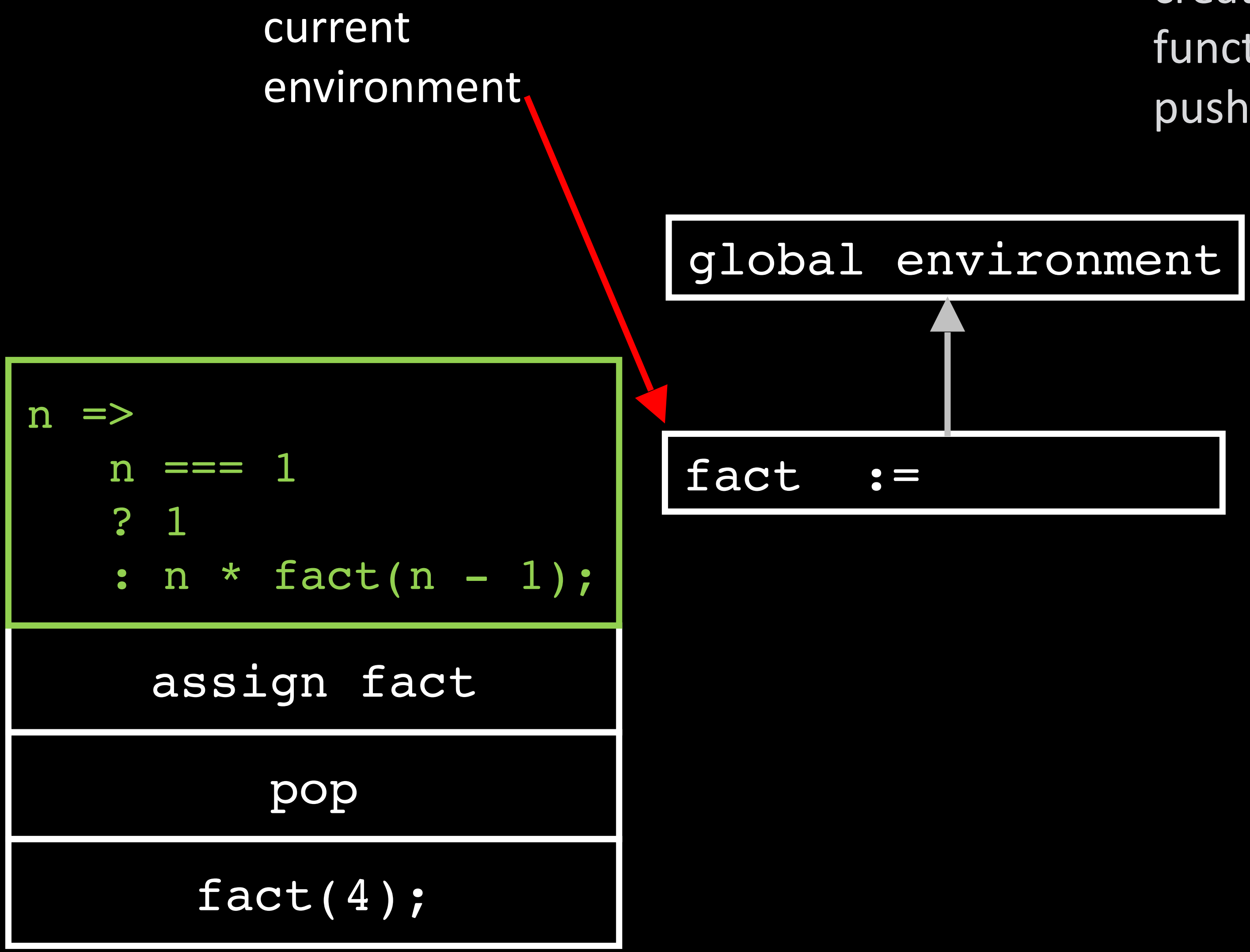
Constant declaration



Lambda expression

Lambda expression:

- create function value
- function value gets **current** environm.
- push reference to function value on stash



```
n =>
   n === 1
   ? 1
   : n * fact(n - 1);
```

```
assign fact
```

```
pop
```

```
fact(4);
```

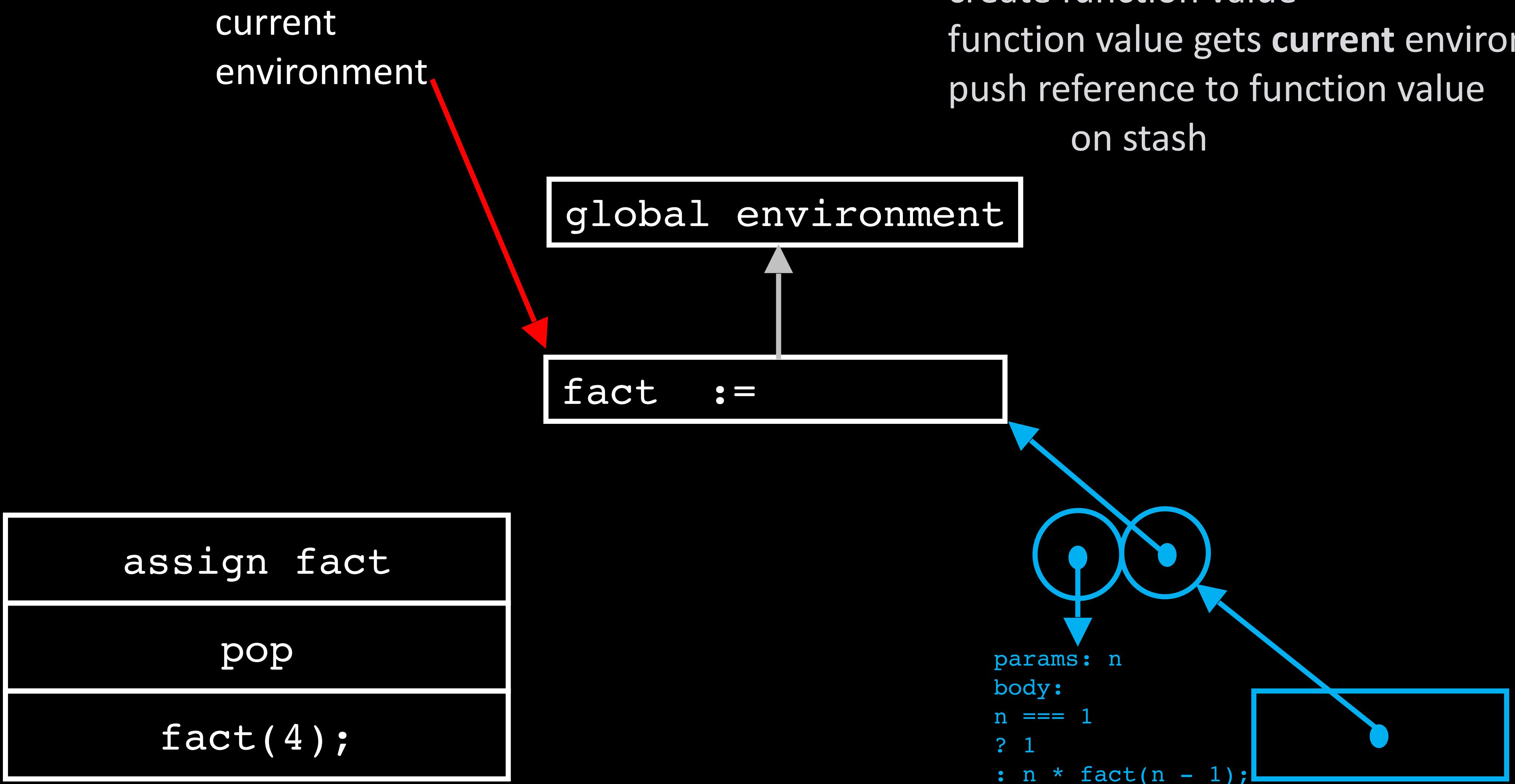
Lambda expression

Lambda expression:

create function value

function value gets **current** environm.

push reference to function value on stash



Assign instruction

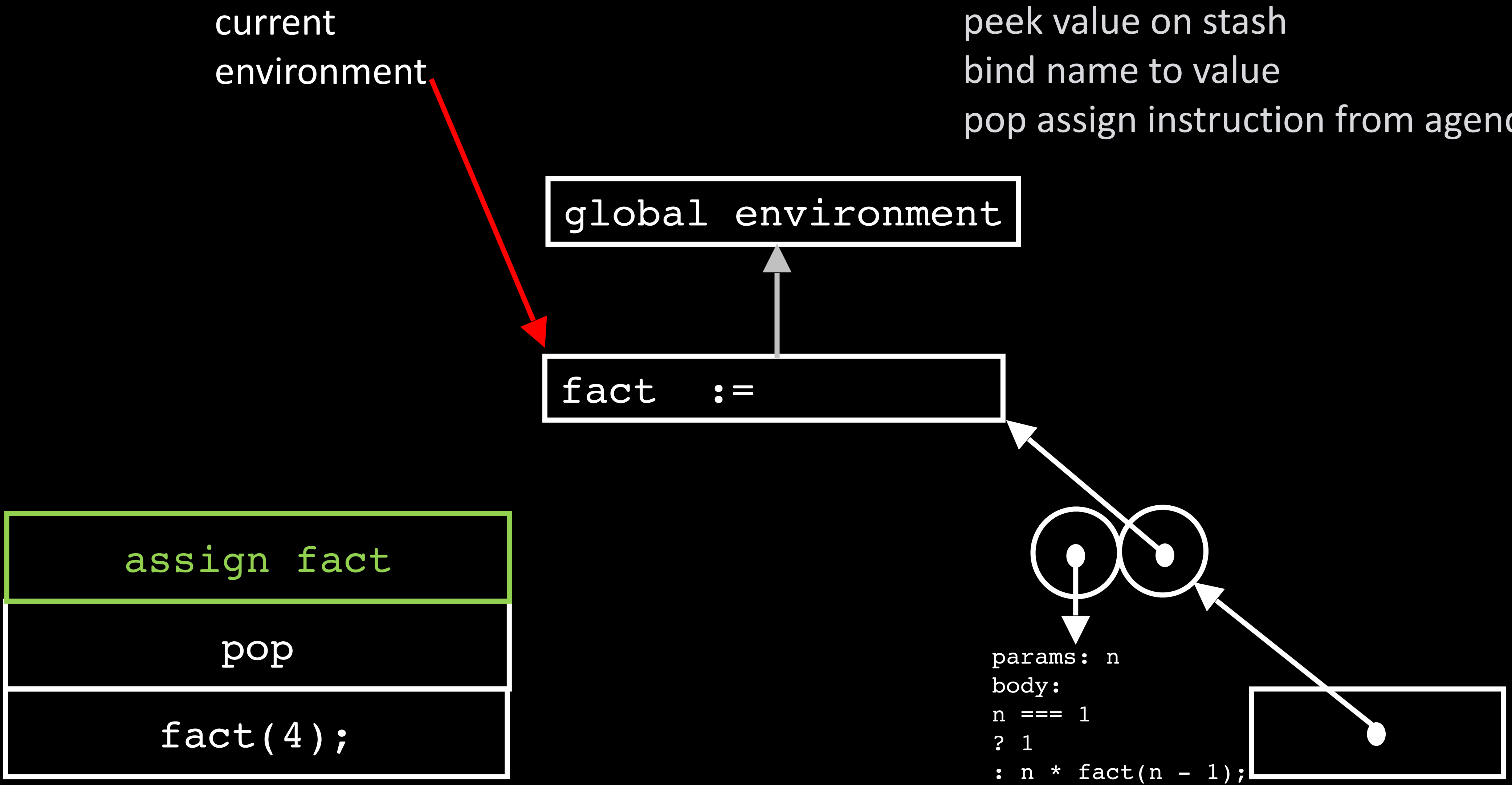
Assign instruction:

locate name in current environment

peek value on stash

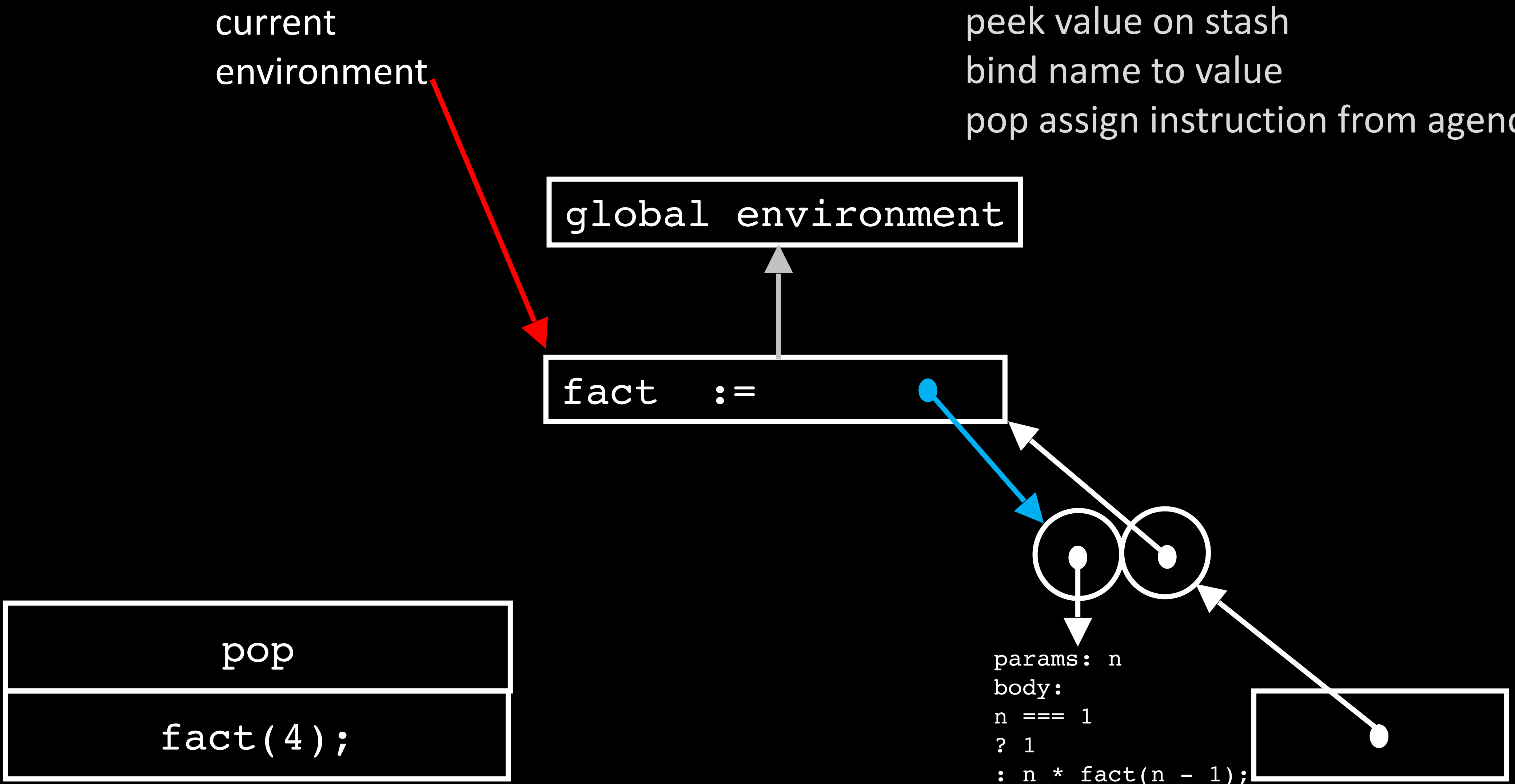
bind name to value

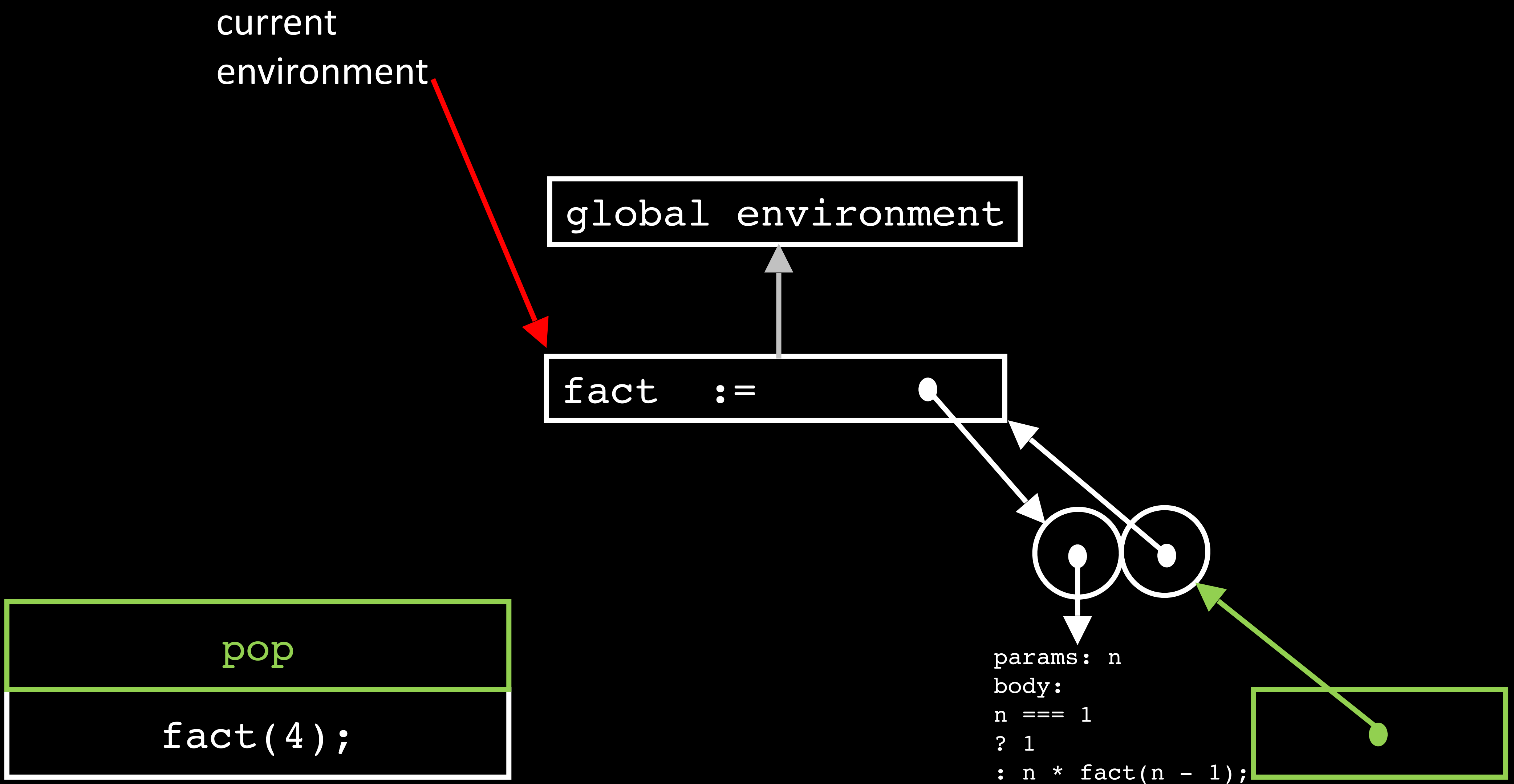
pop assign instruction from agenda



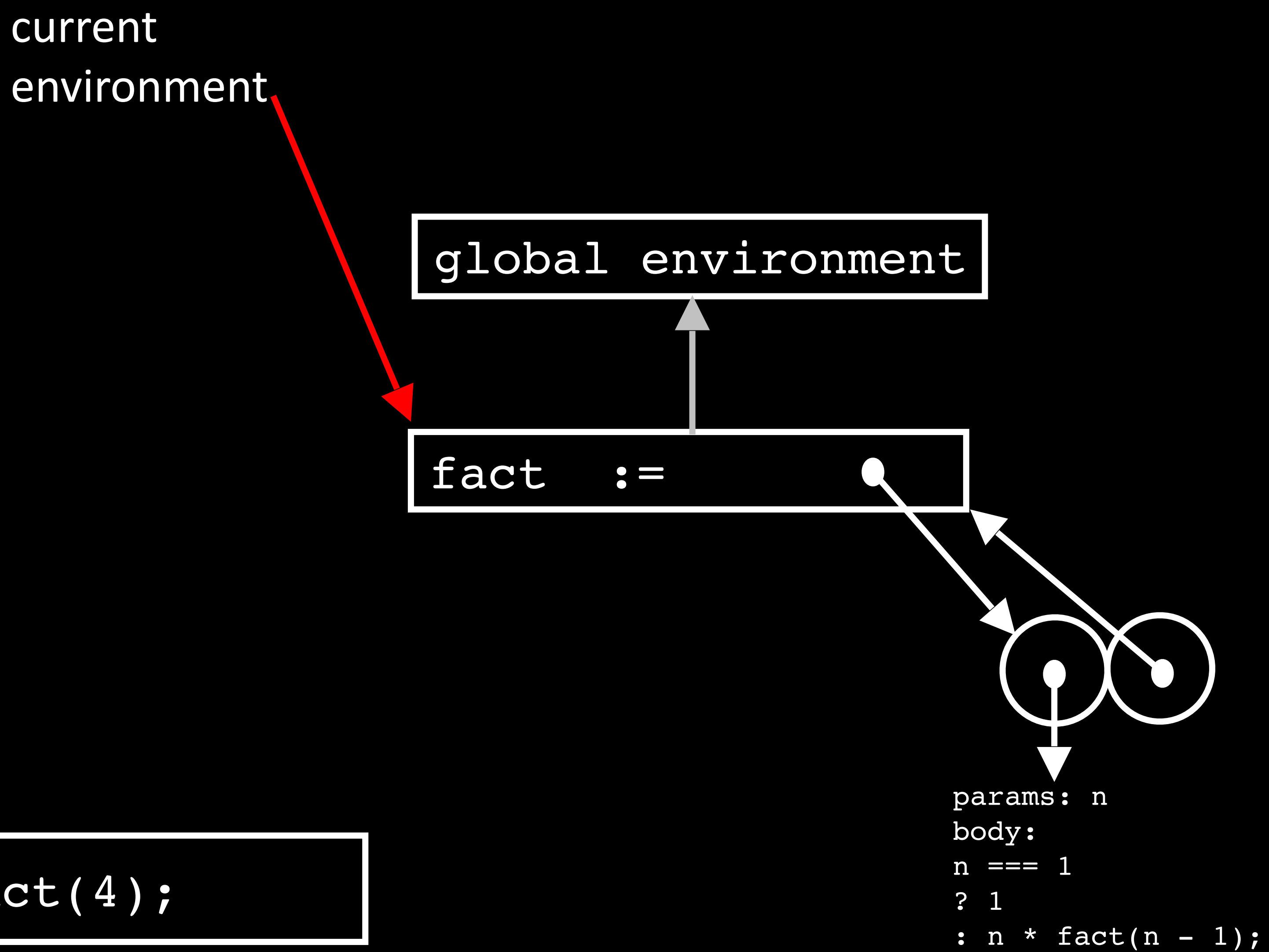
Assign instruction

Assign instruction:
locate name in current environment
peek value on stash
bind name to value
pop assign instruction from agenda





```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

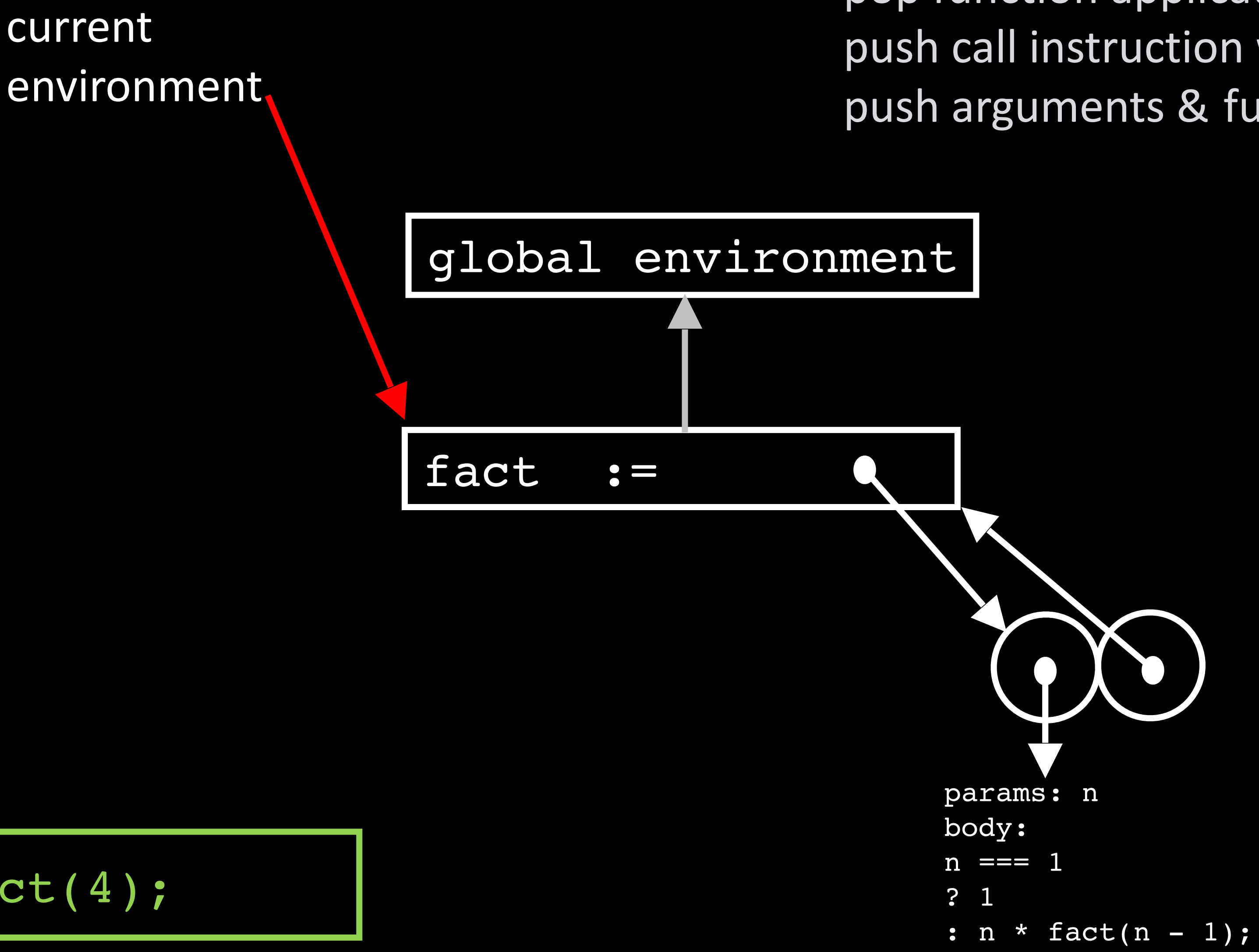


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

```
fact(4);
```

Function application

Function application:
- pop function application from agenda
- push call instruction with # arguments
- push arguments & function expression

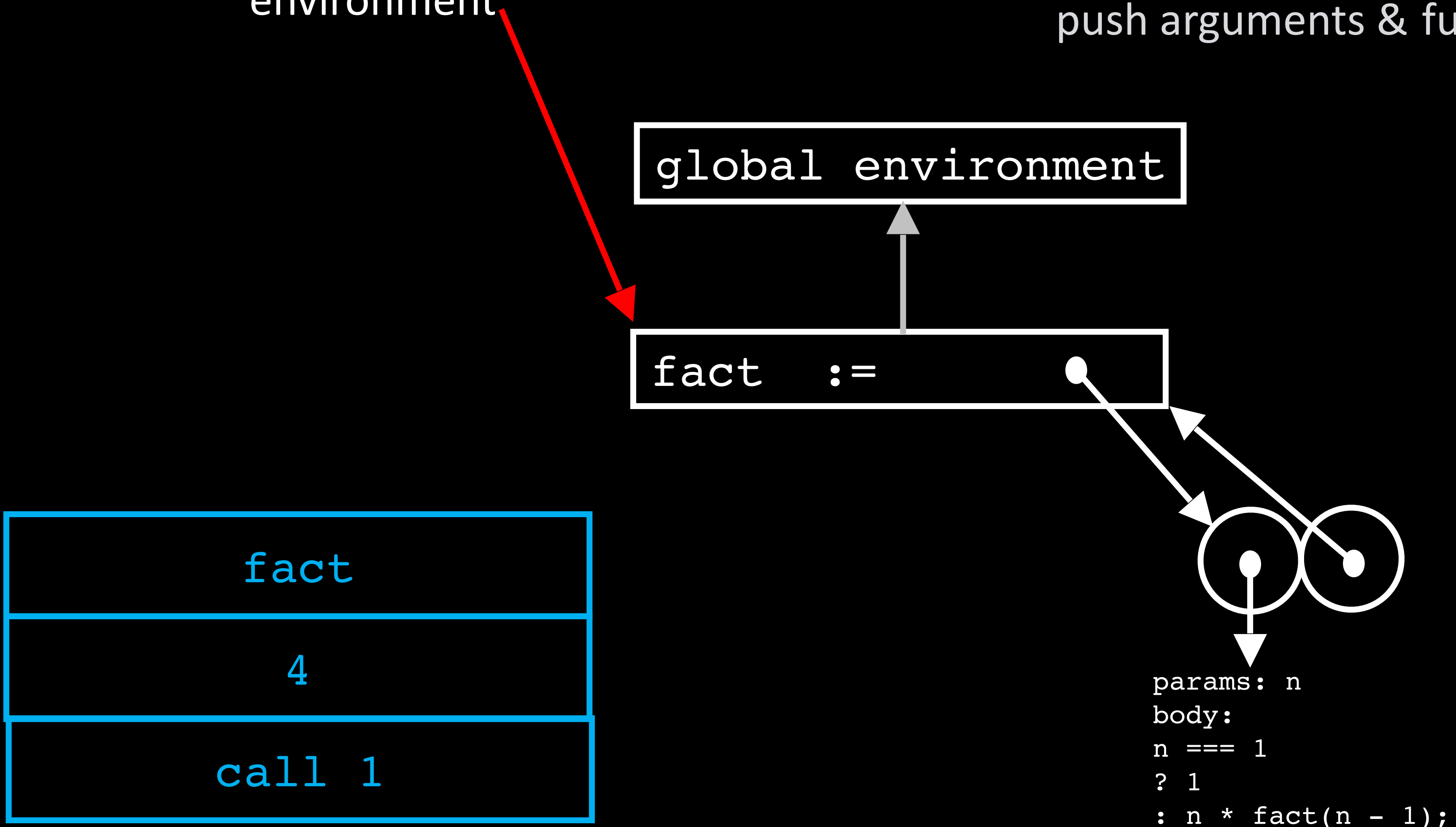


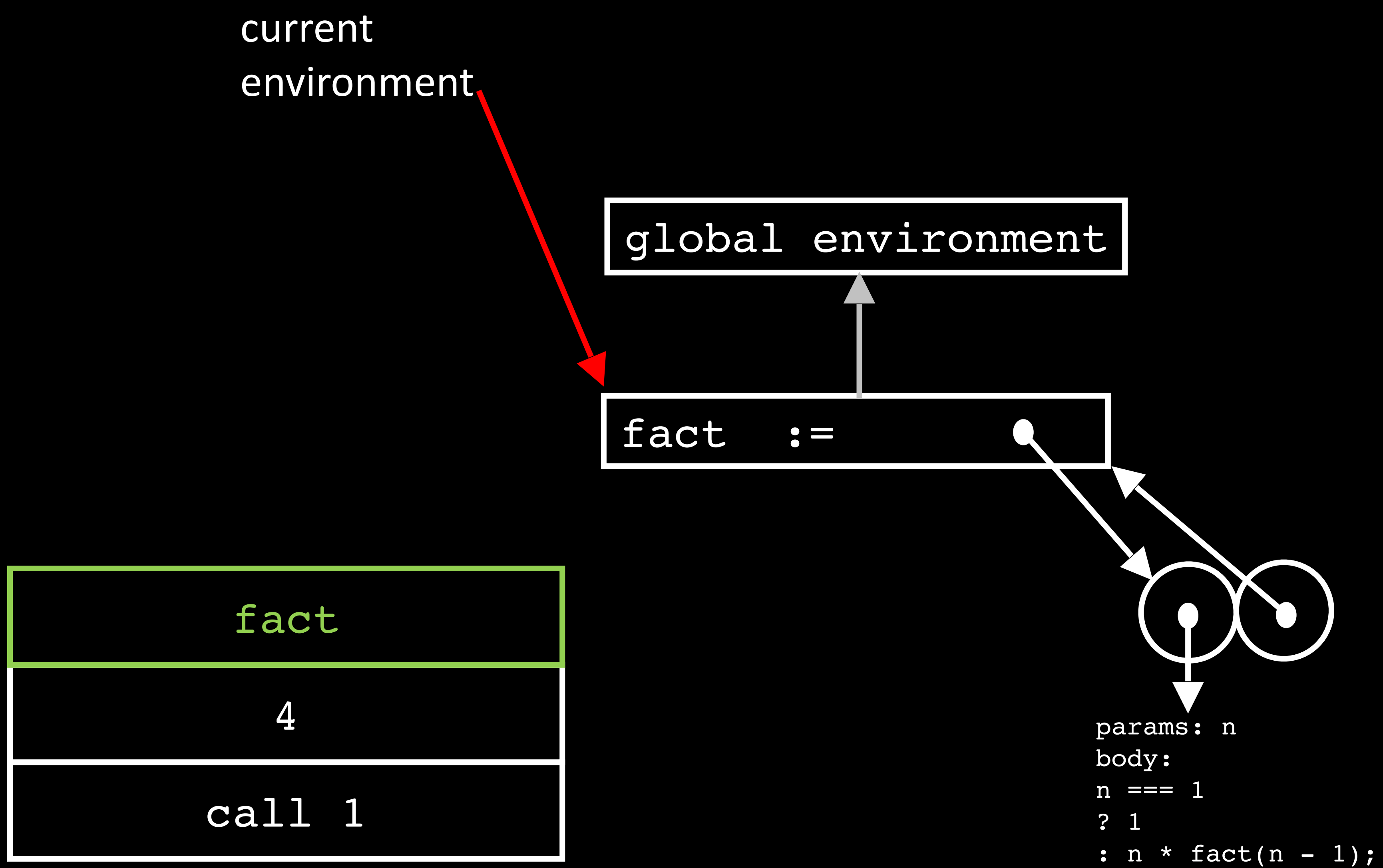
```
fact(4);
```

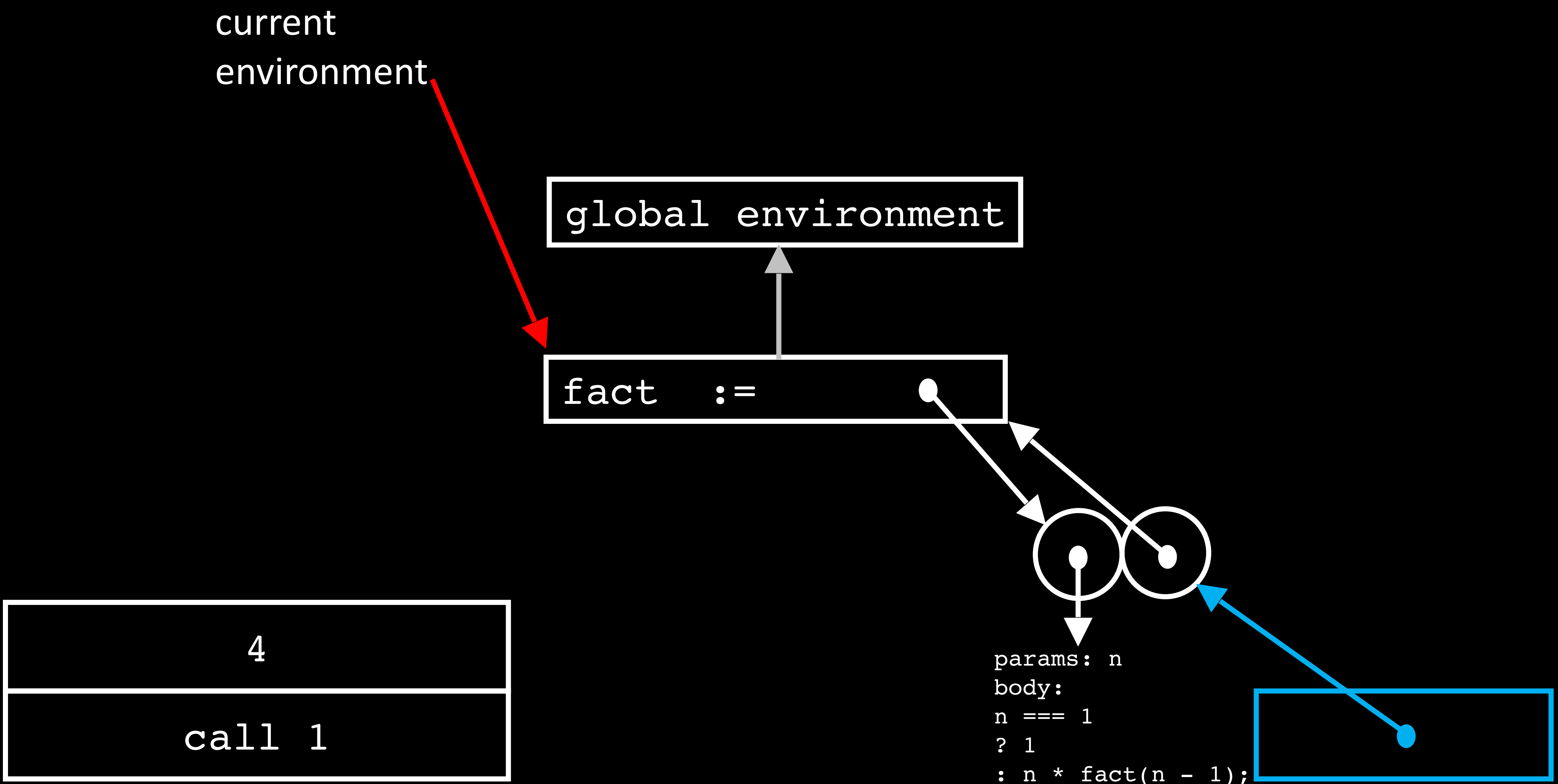
Function application

Function application:

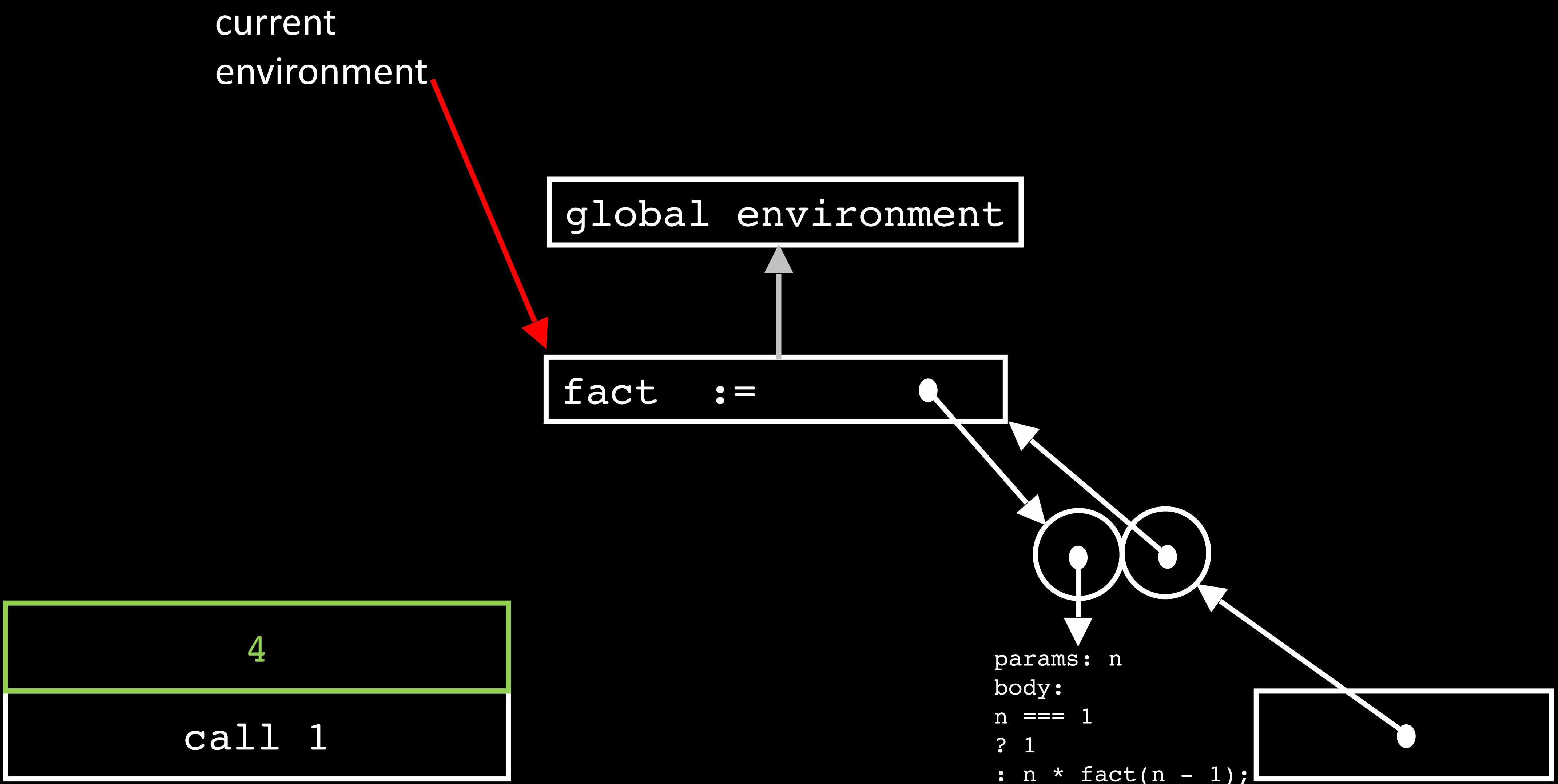
- pop function application from agenda
- push call instruction with # arguments
- push arguments & function expression

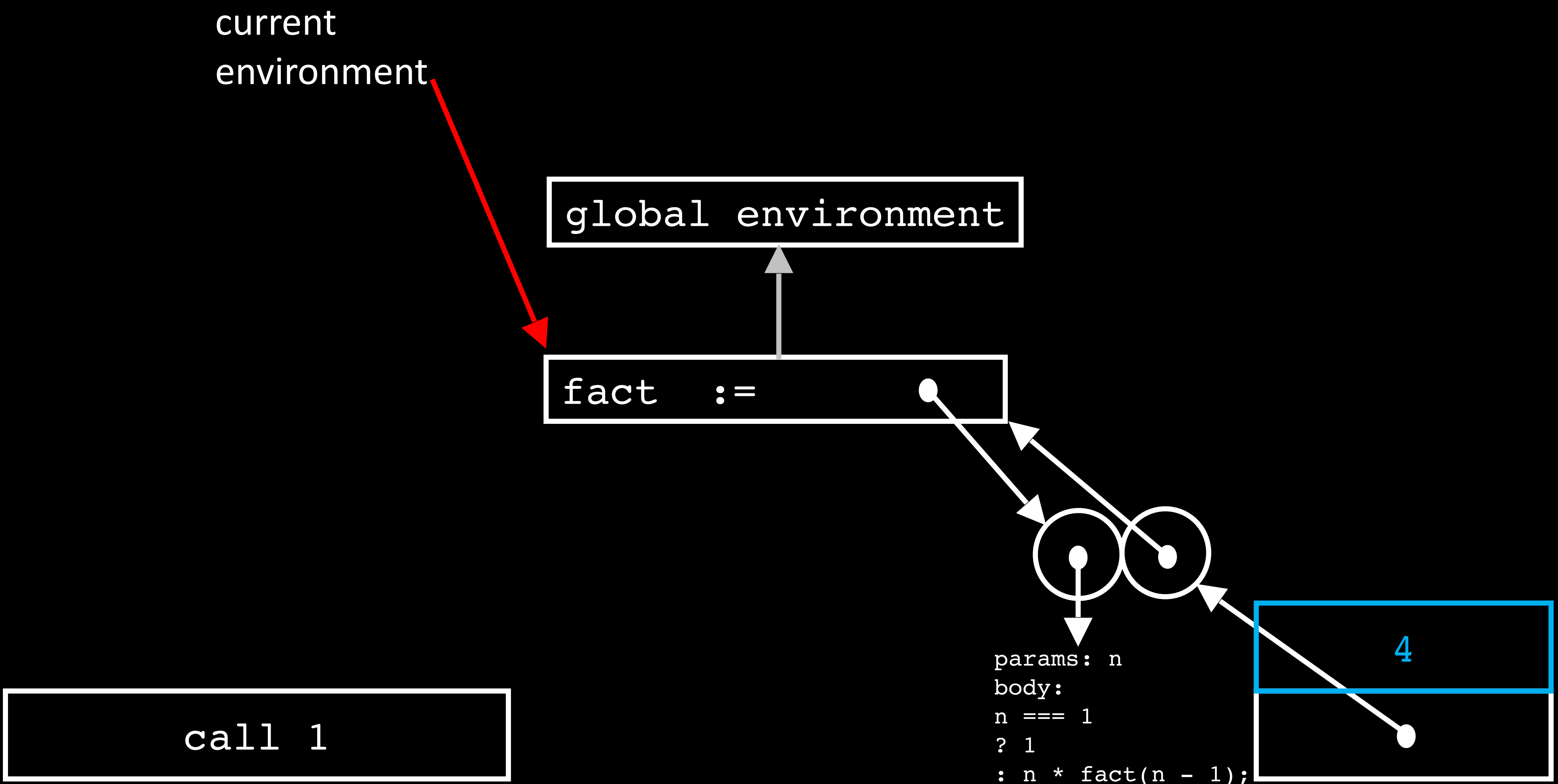



current environment
global environment
fact :=
fact
4
call 1
params: n
body:
n === 1
? 1
: n * fact(n - 1);



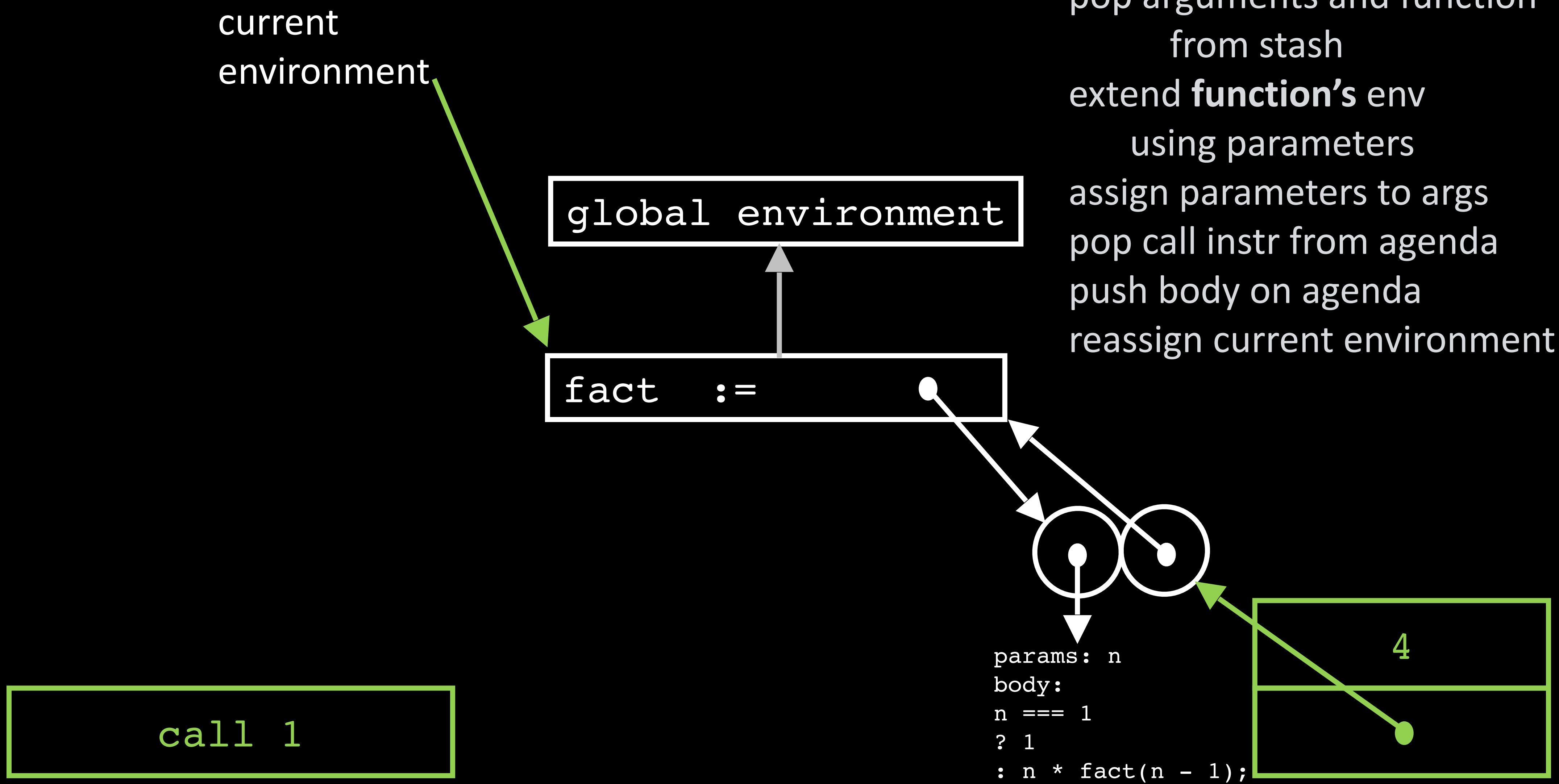
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```


current environment
global environment
fact :=
params: n
body:
n === 1
? 1
: n * fact(n - 1);
4
call 1



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

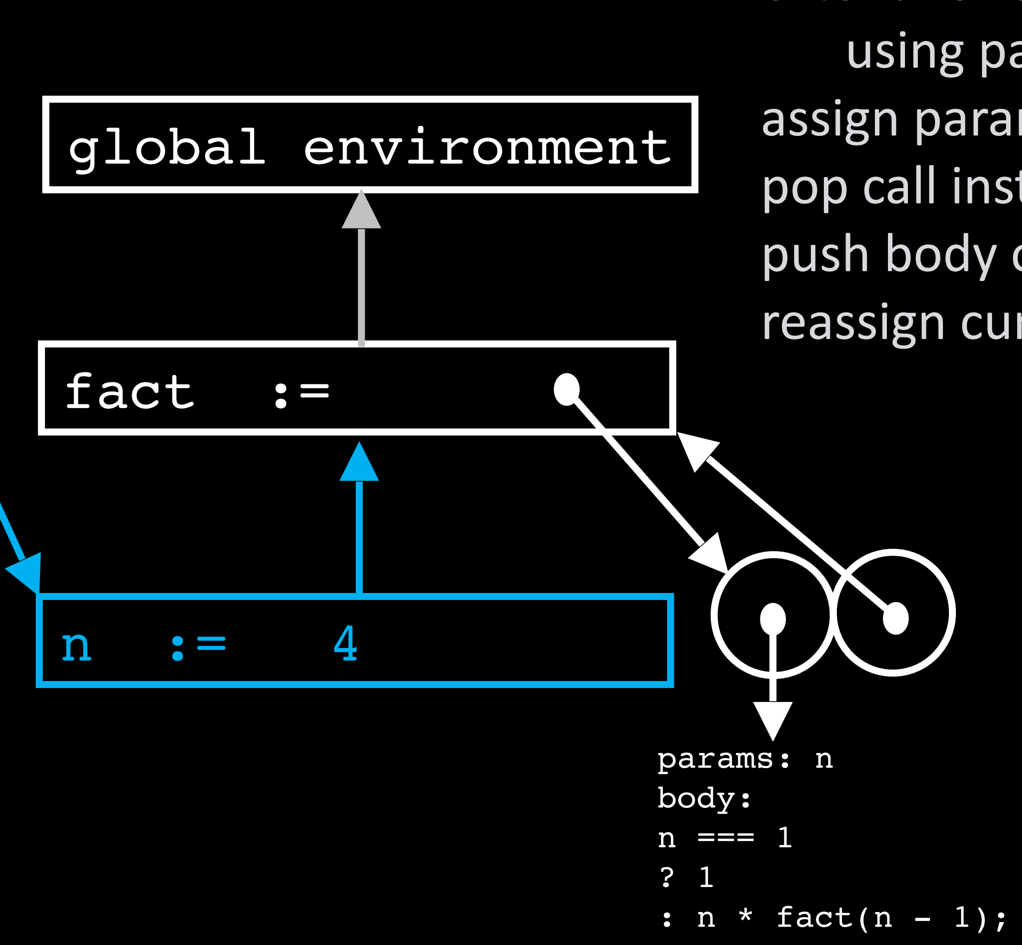
Call instruction



Call instruction:

pop arguments and function
from stash
extend **function's** env
using parameters
assign parameters to args
pop call instr from agenda
push body on agenda
reassign current environment

Call instruction



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Call instruction:

pop arguments and function
from stash

extend **function’s** env
using parameters

assign parameters to args

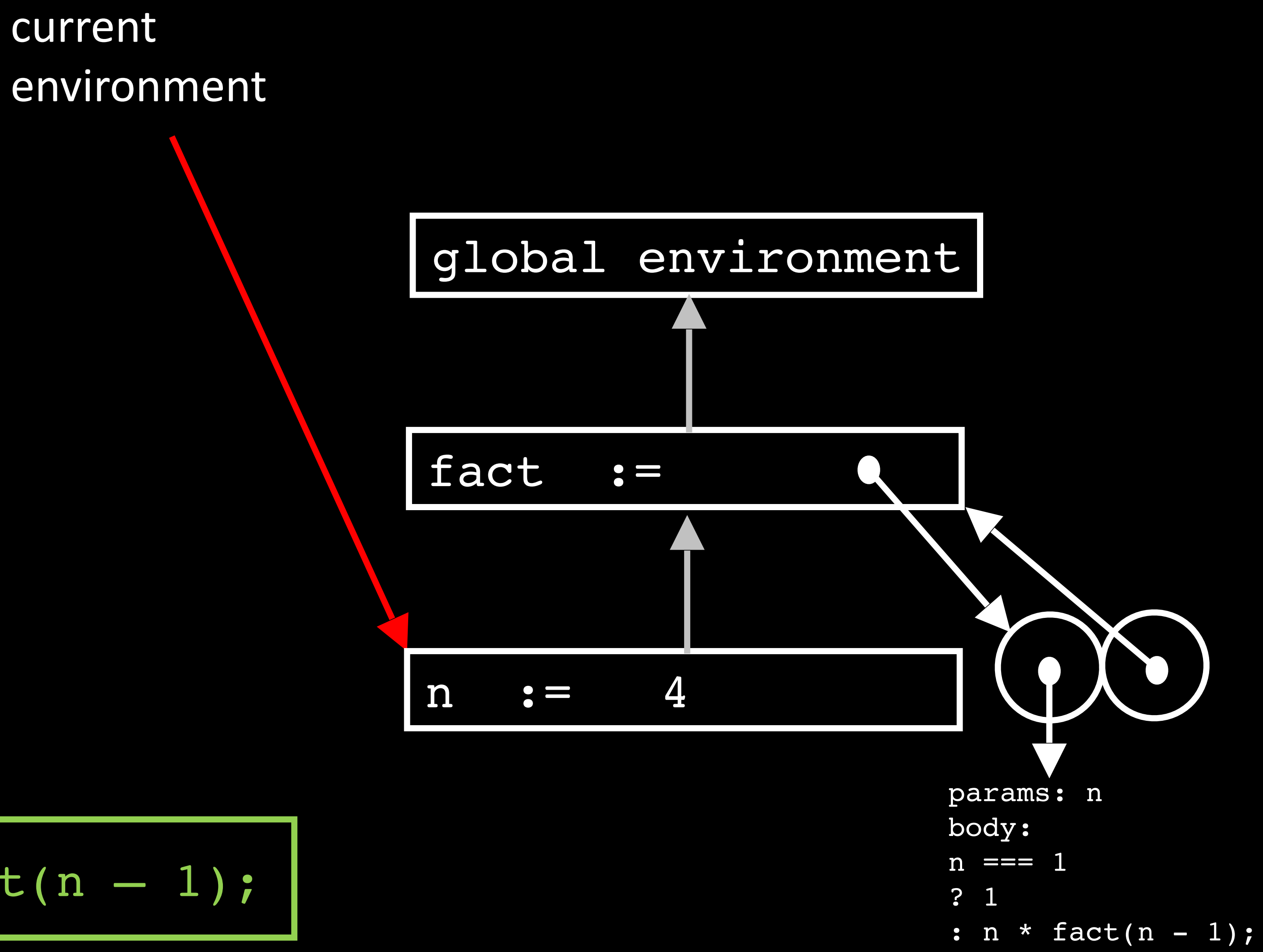
pop call instr from agenda

push body on agenda

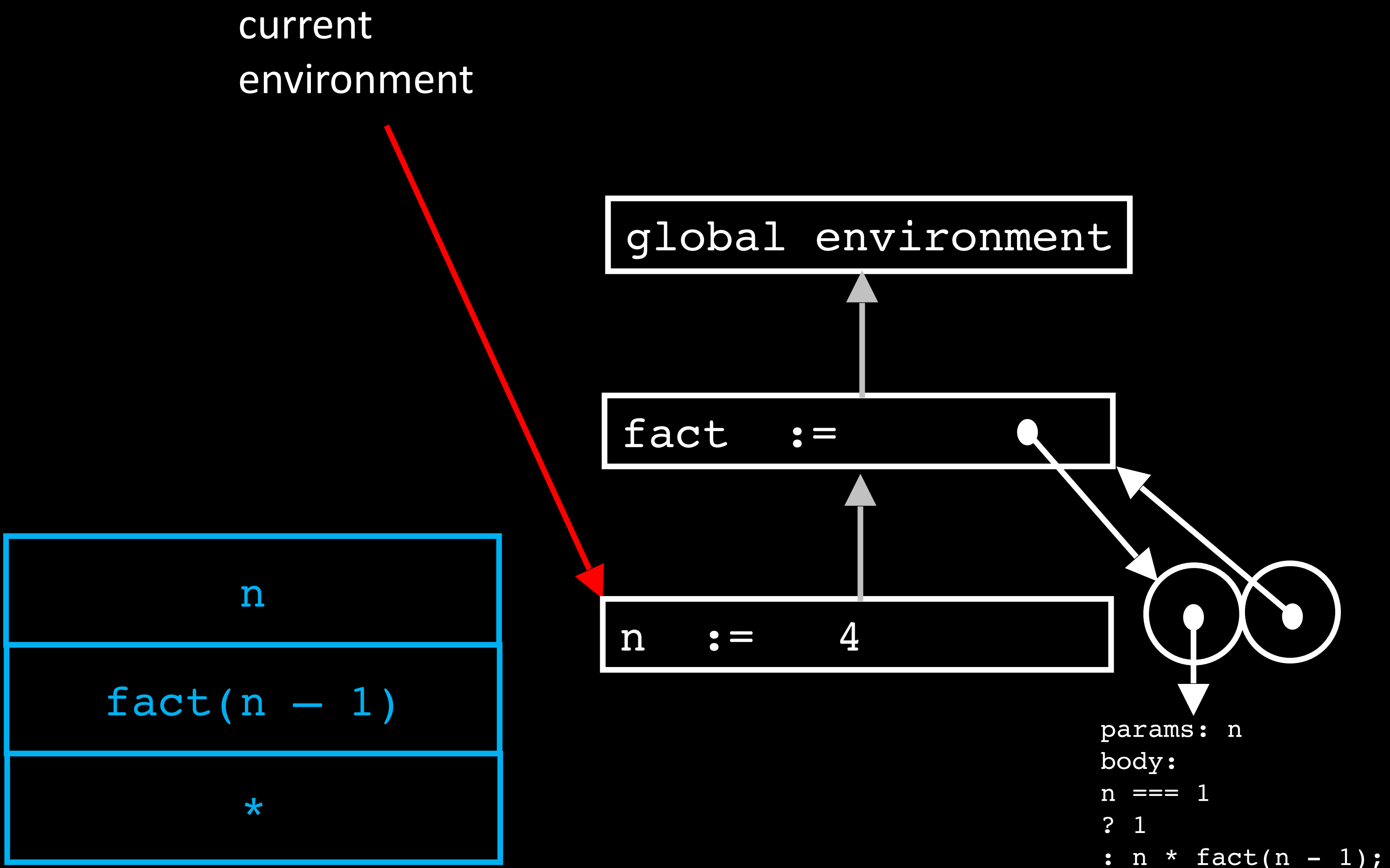
reassign current environment

```
n === 1
? 1
: n * fact(n – 1);
```

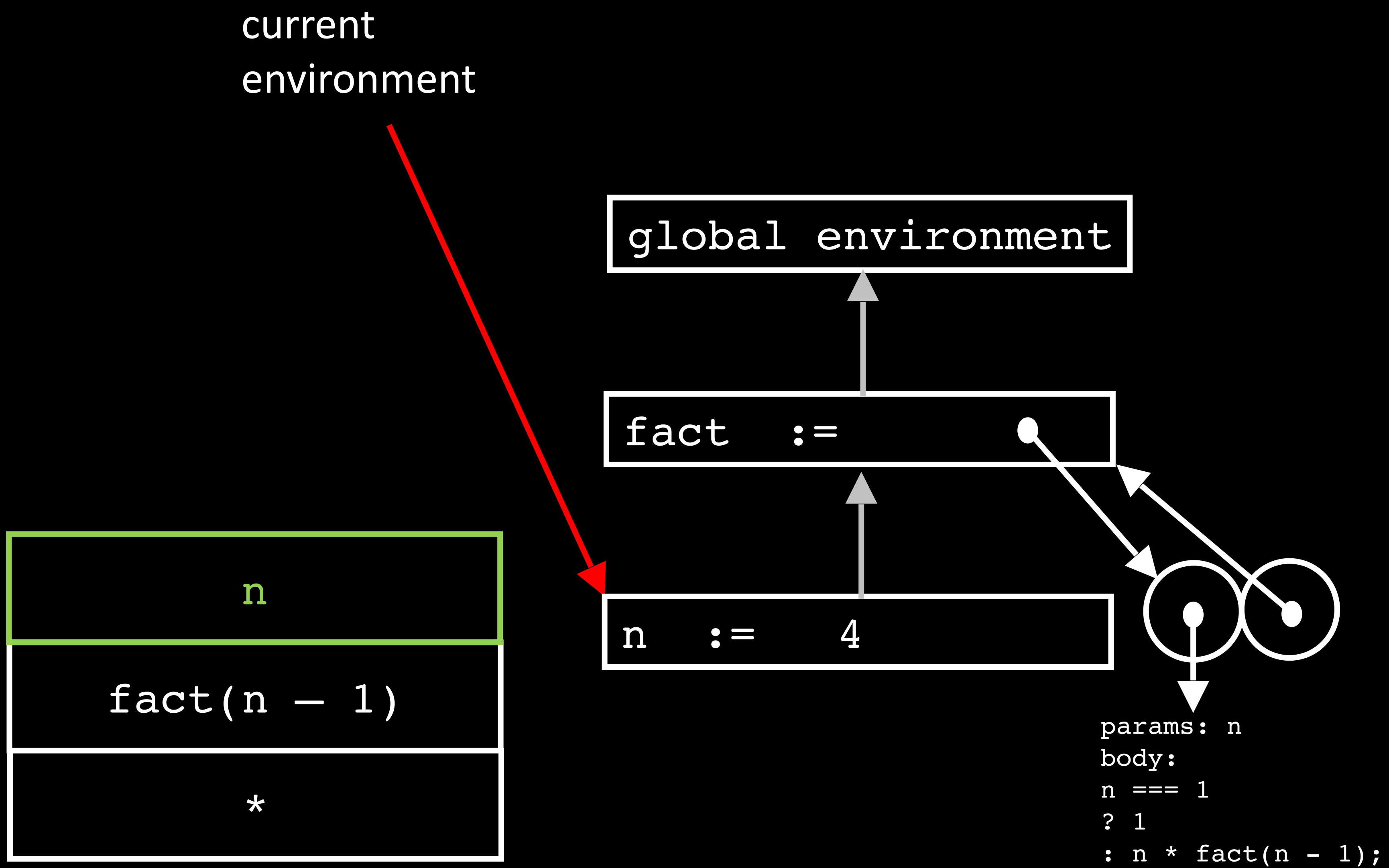
...eventually



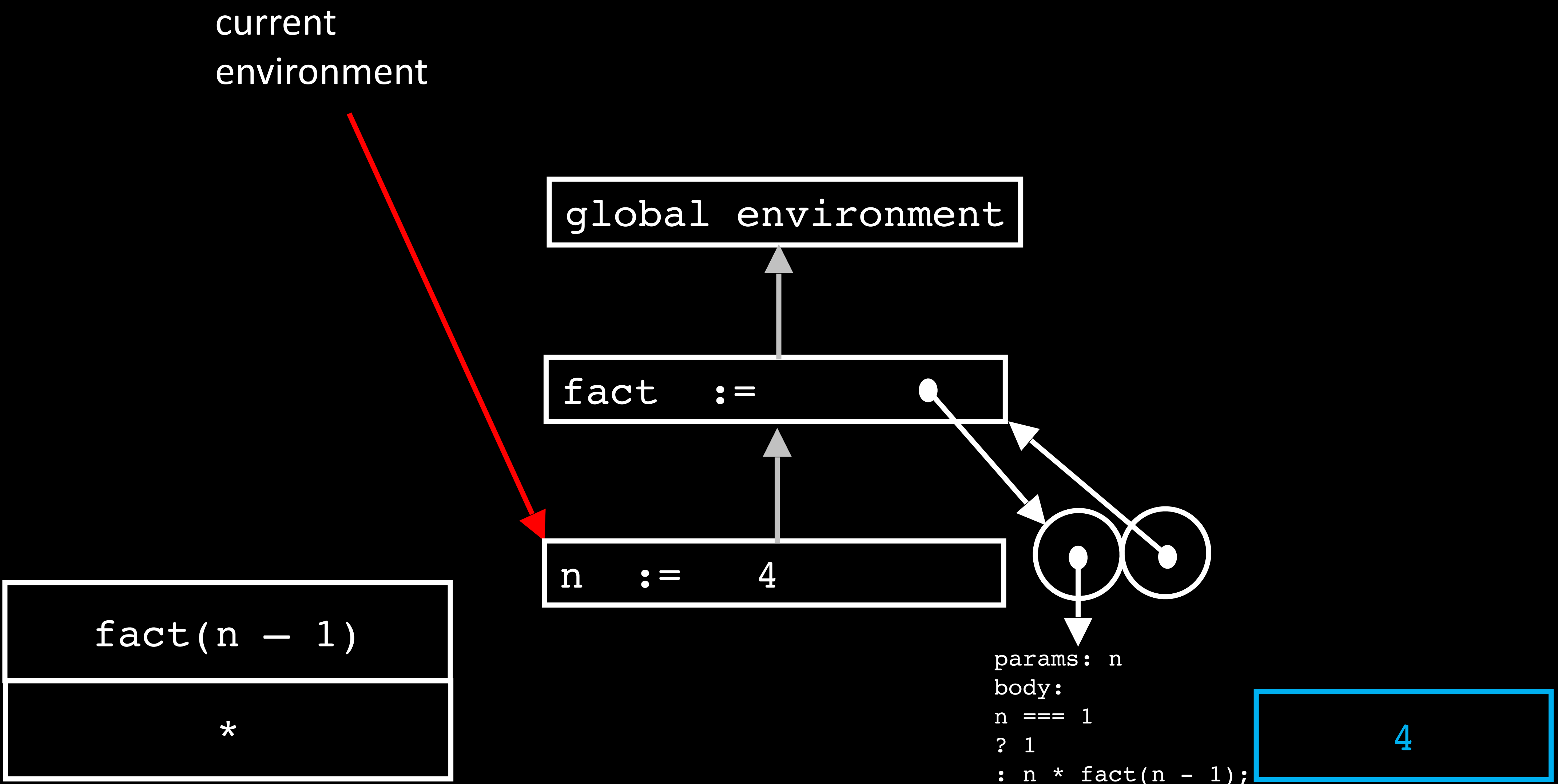
```
n * fact(n – 1);
```



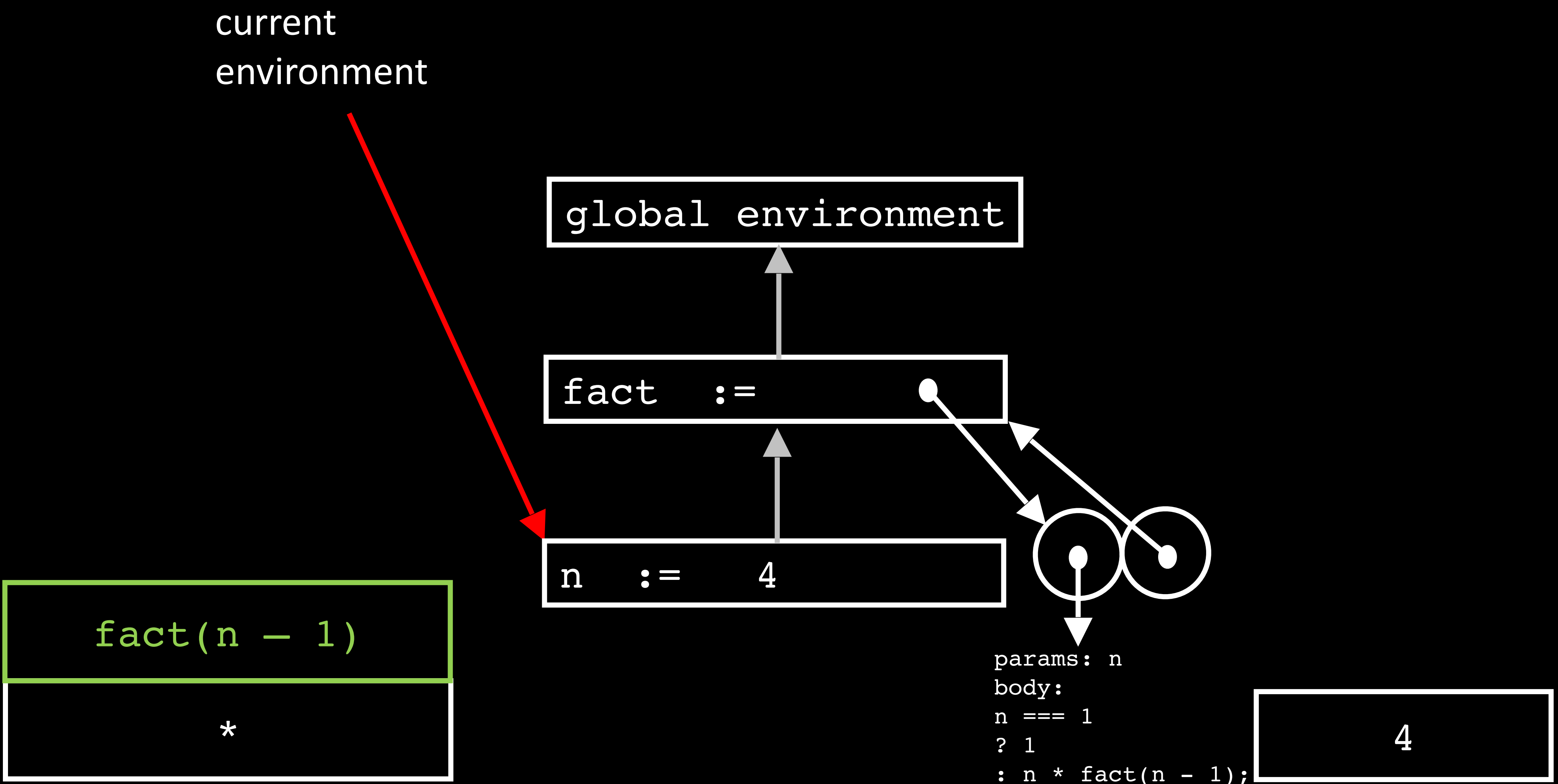
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



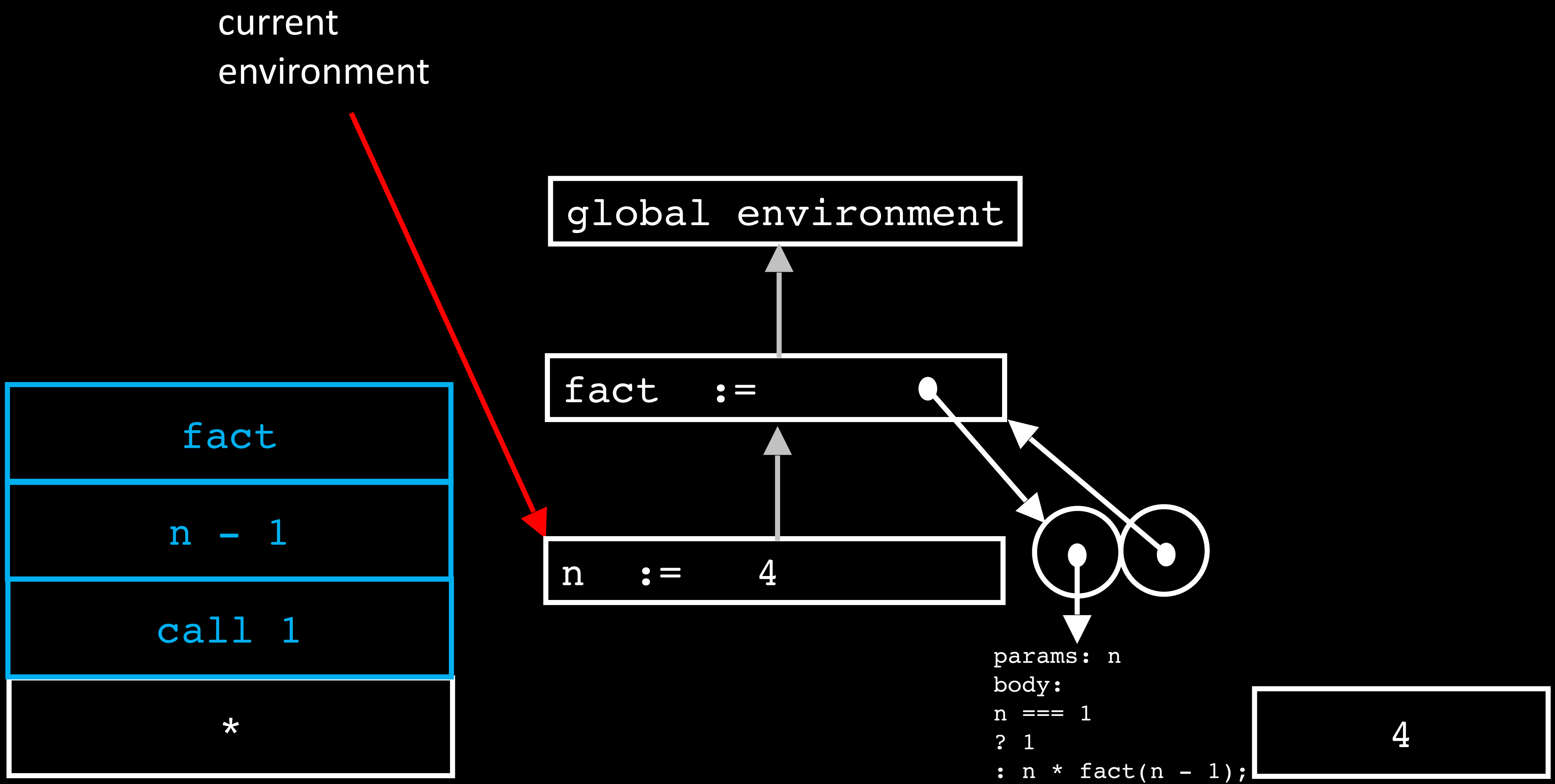
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



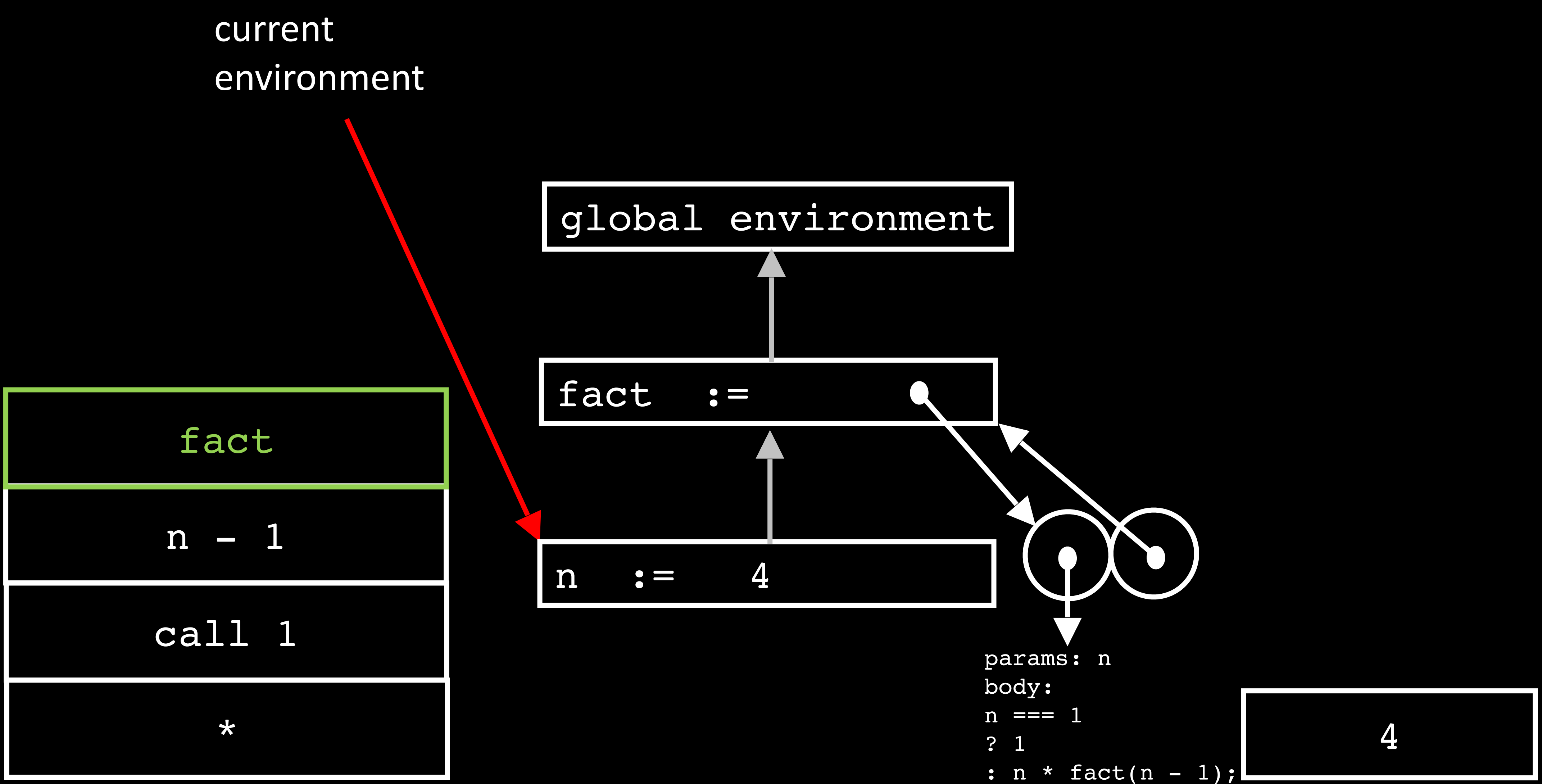
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



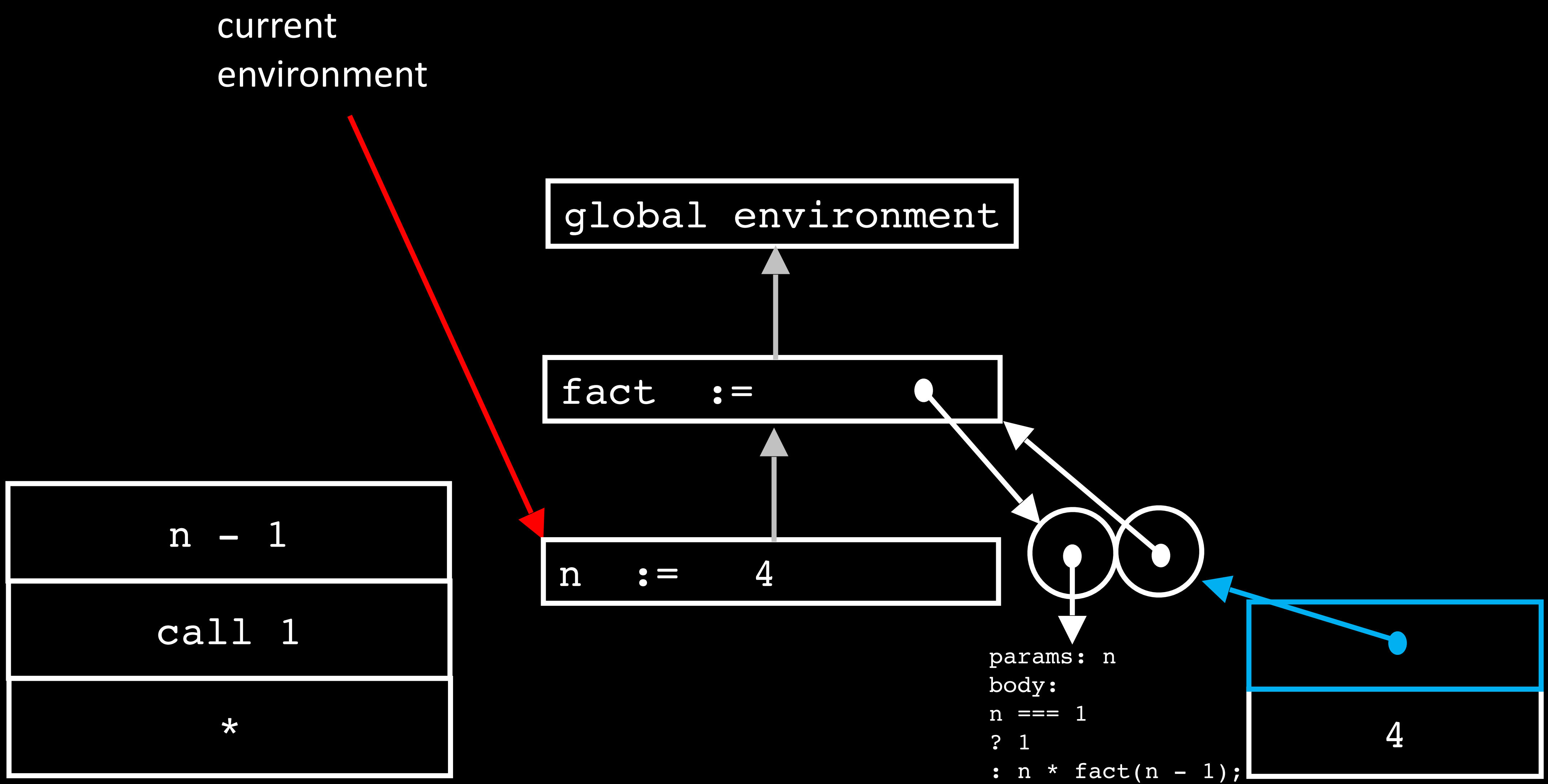
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

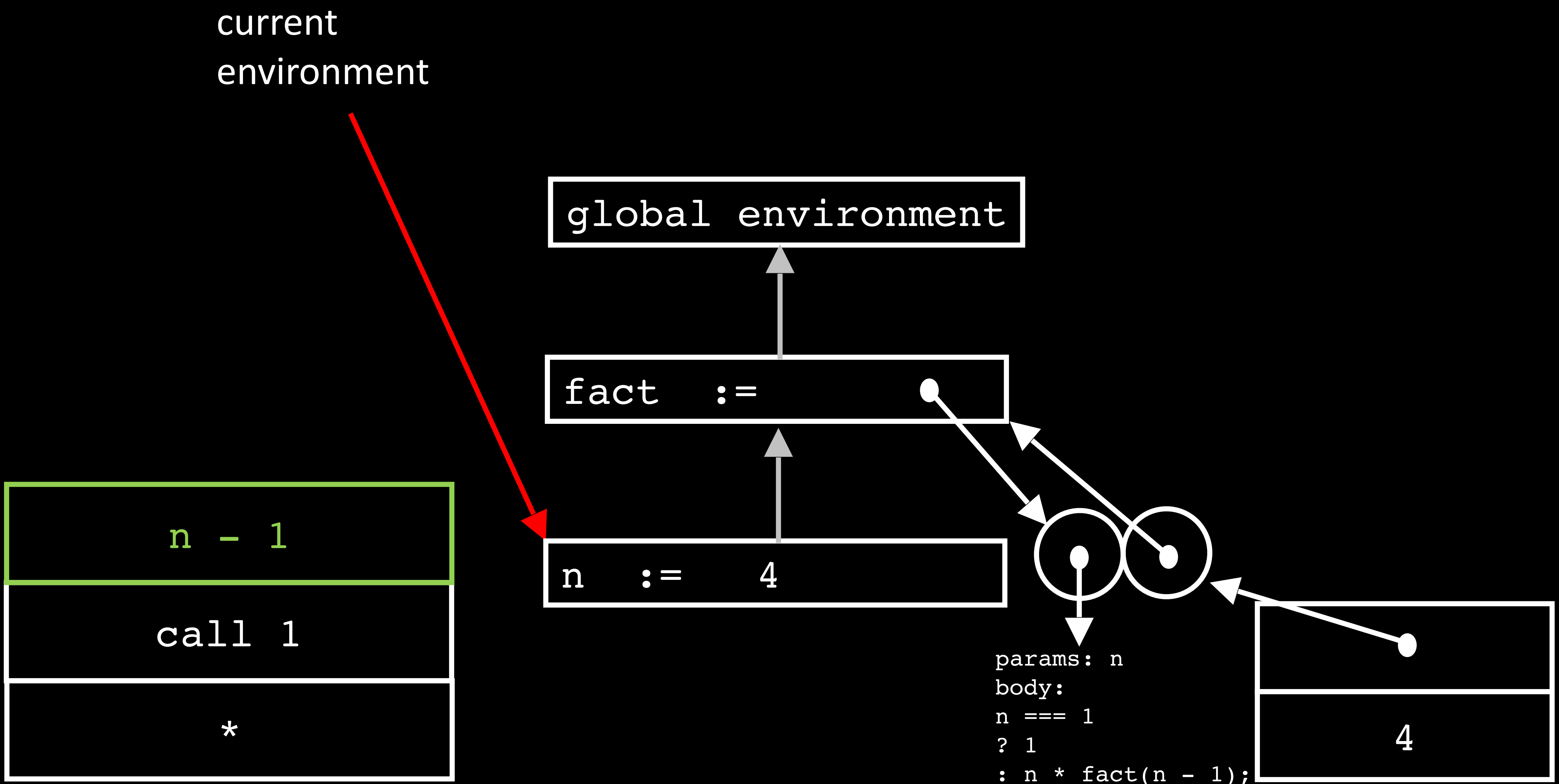


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

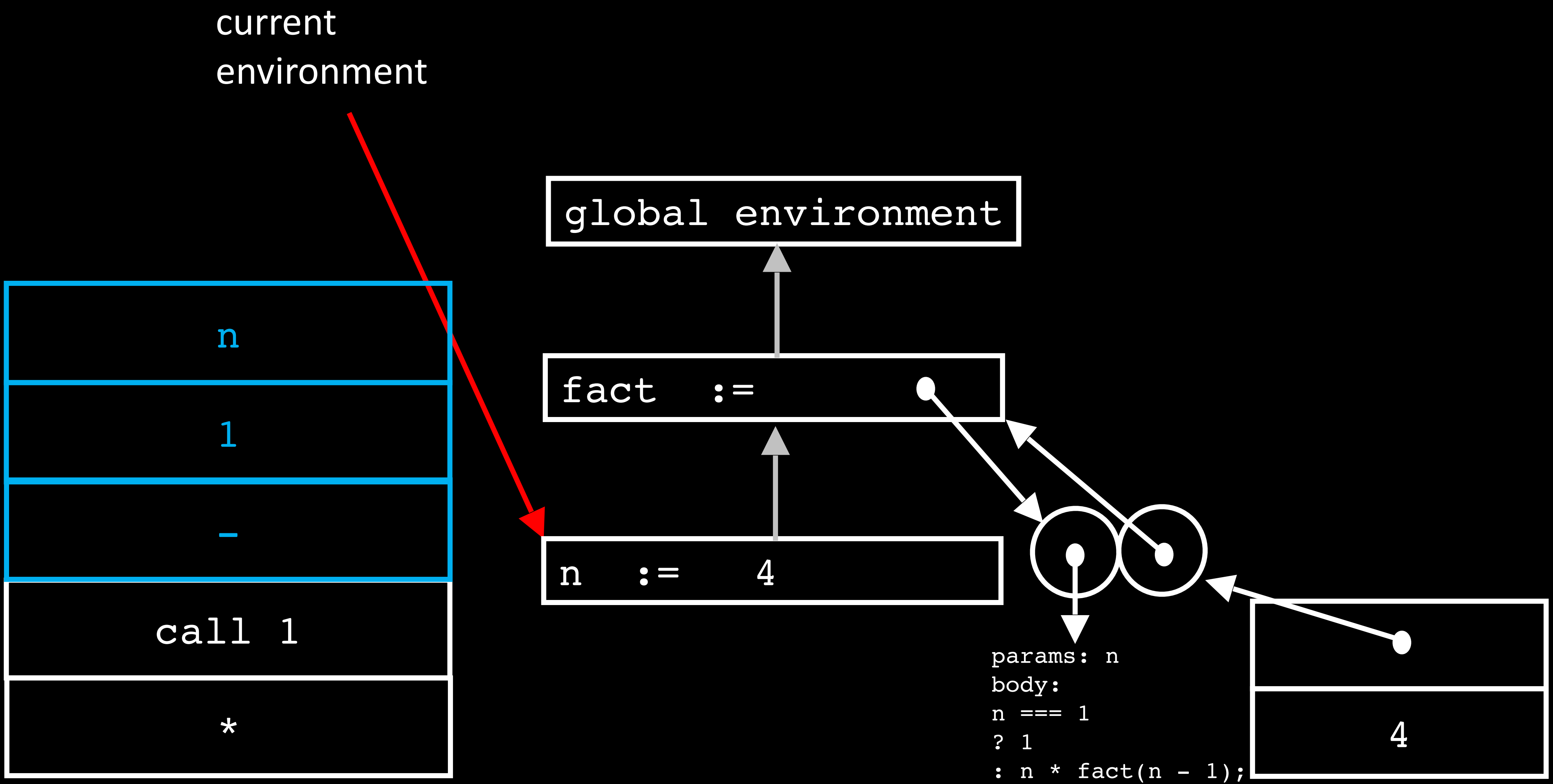


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

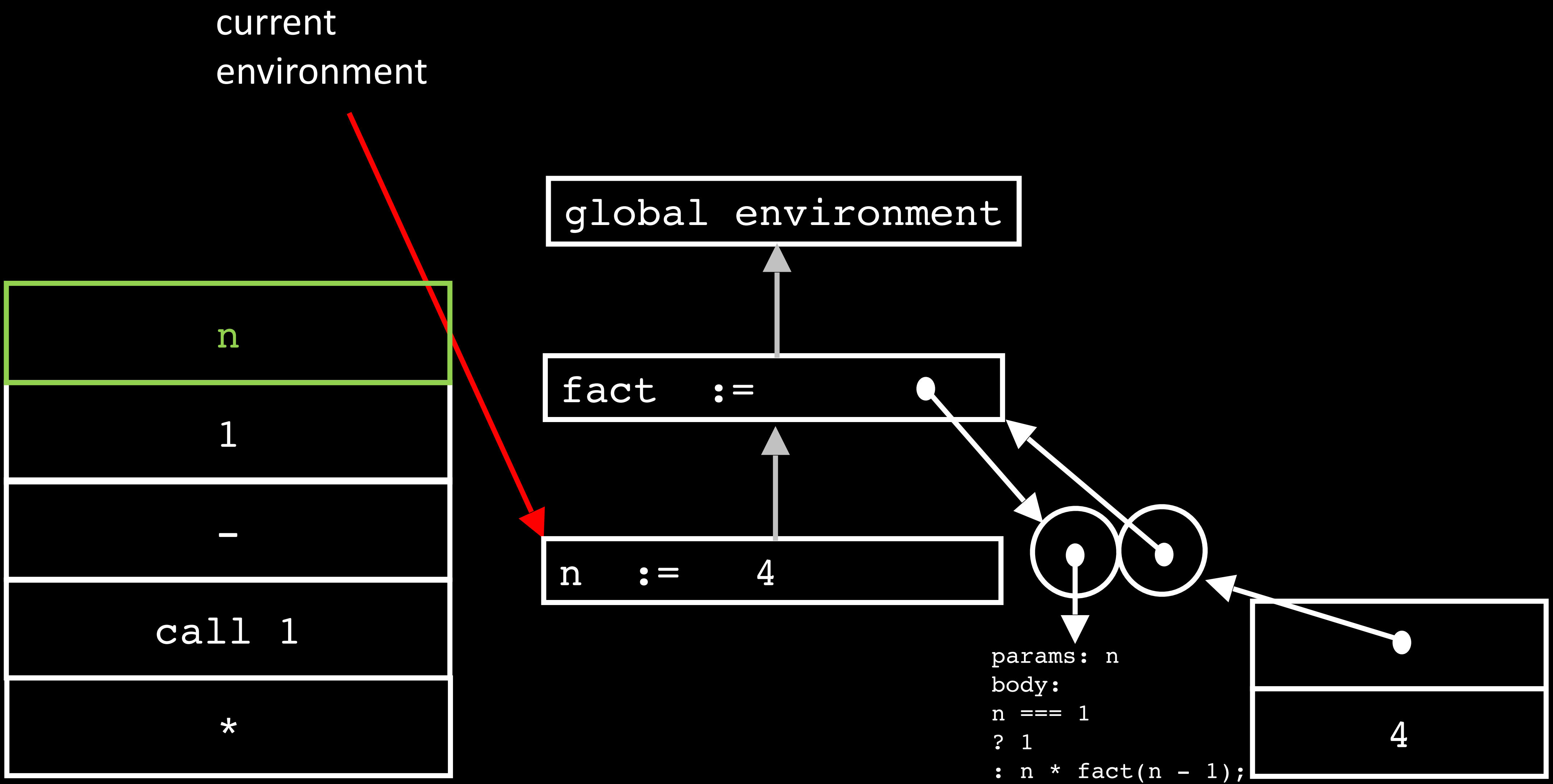

current
environment
global environment
fact :=
n := 4
n - 1
call 1
*
params: n
body:
n === 1
? 1
: n * fact(n - 1);
4



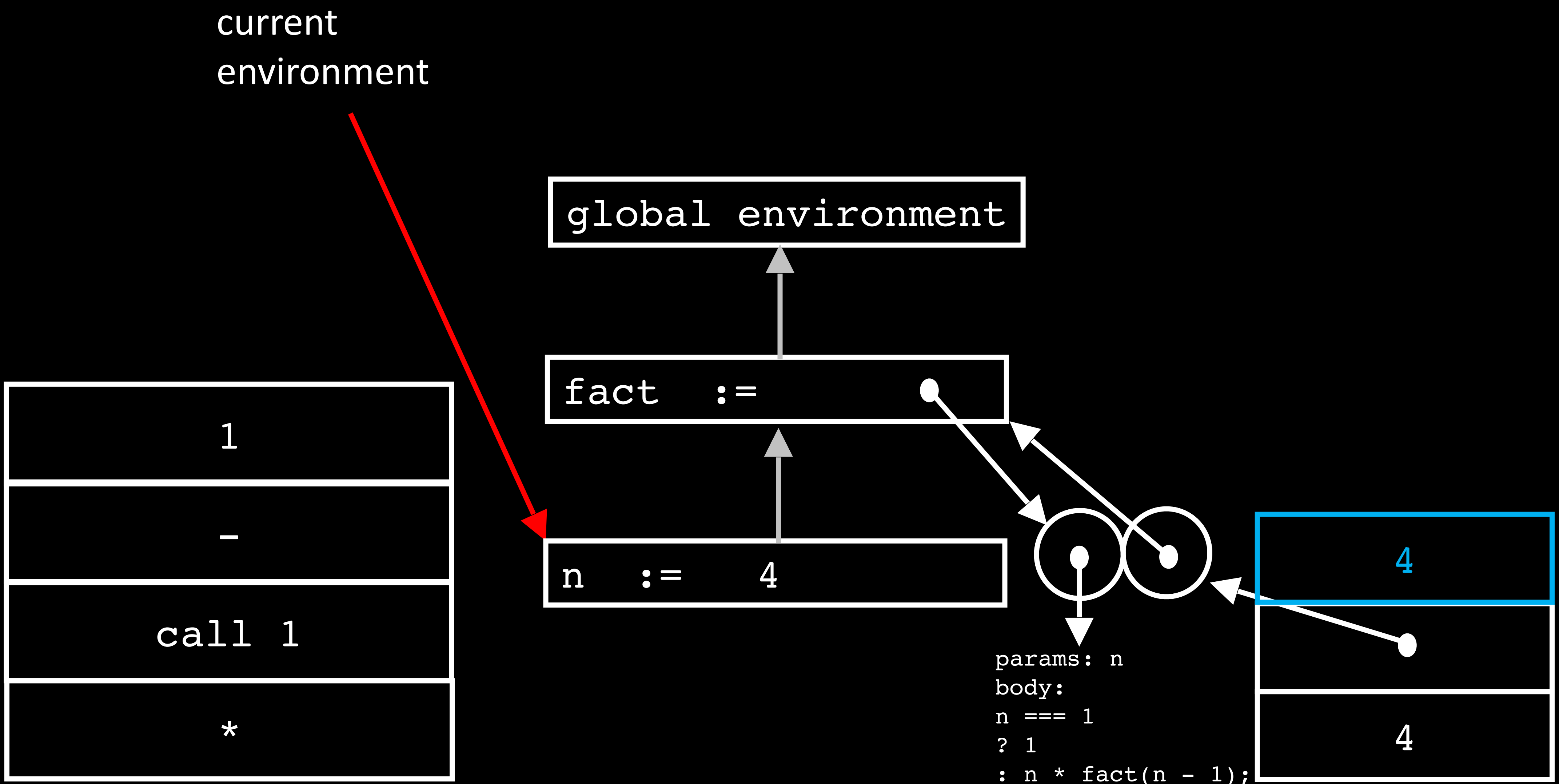
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



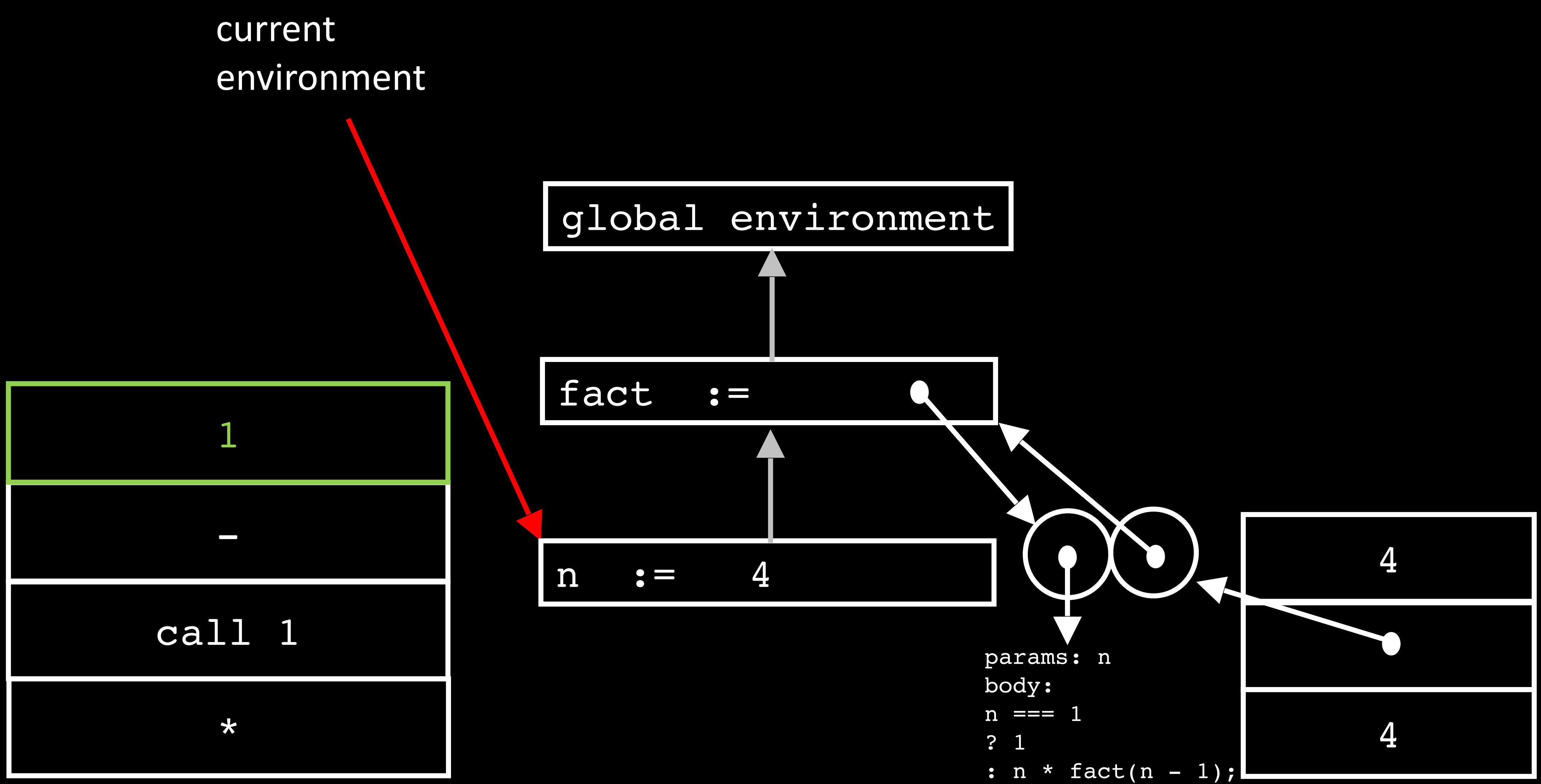
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

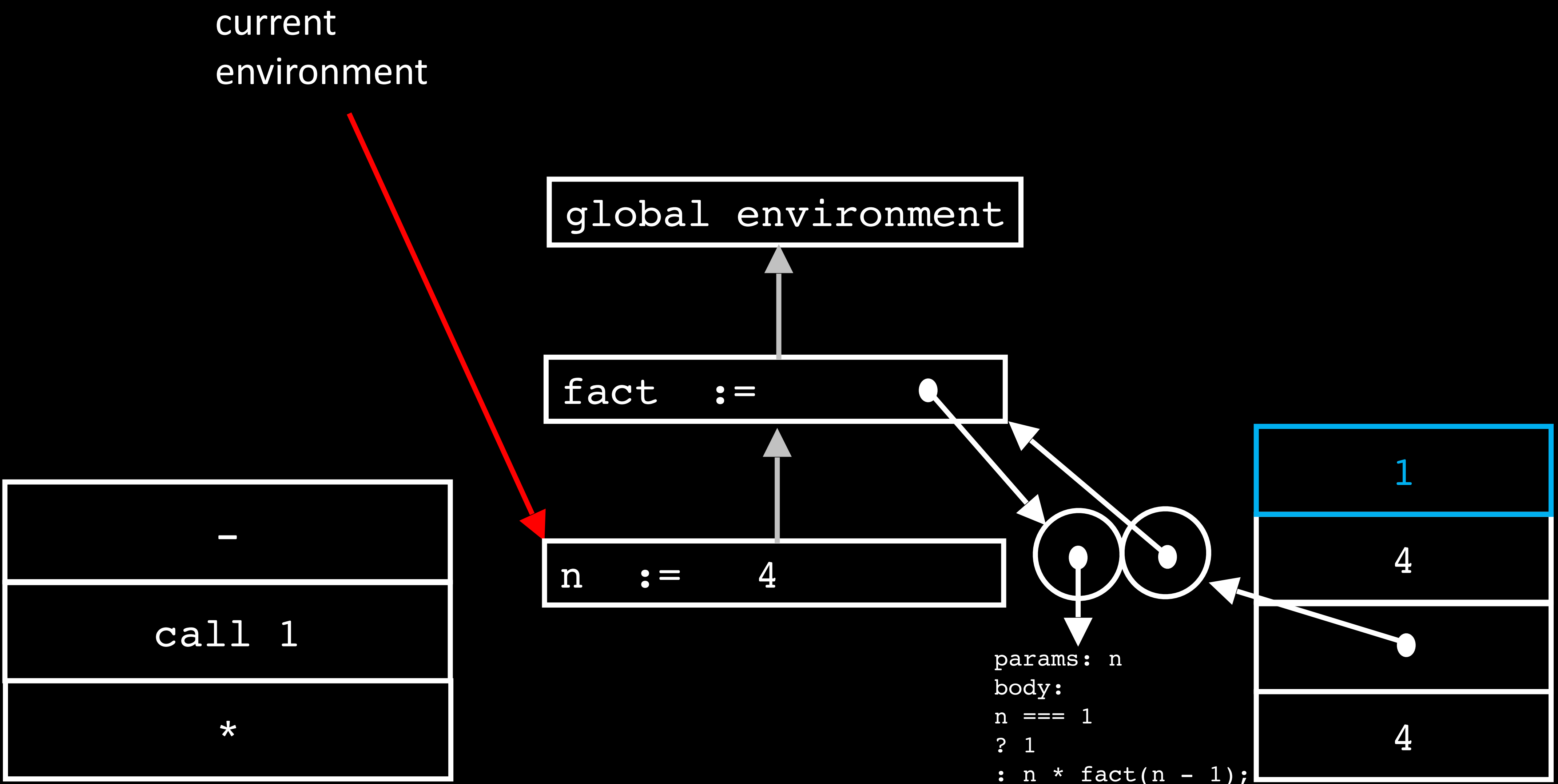


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

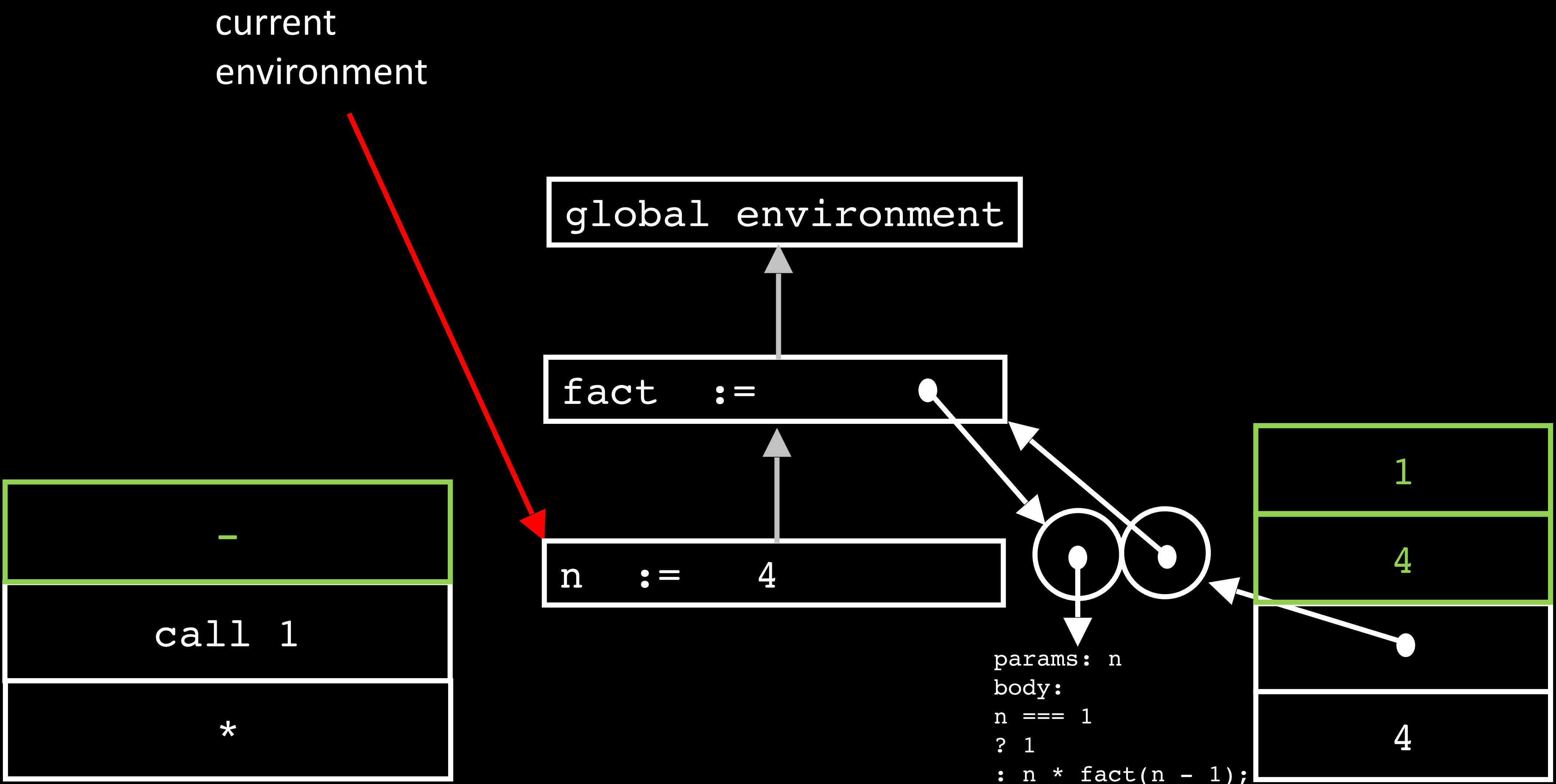


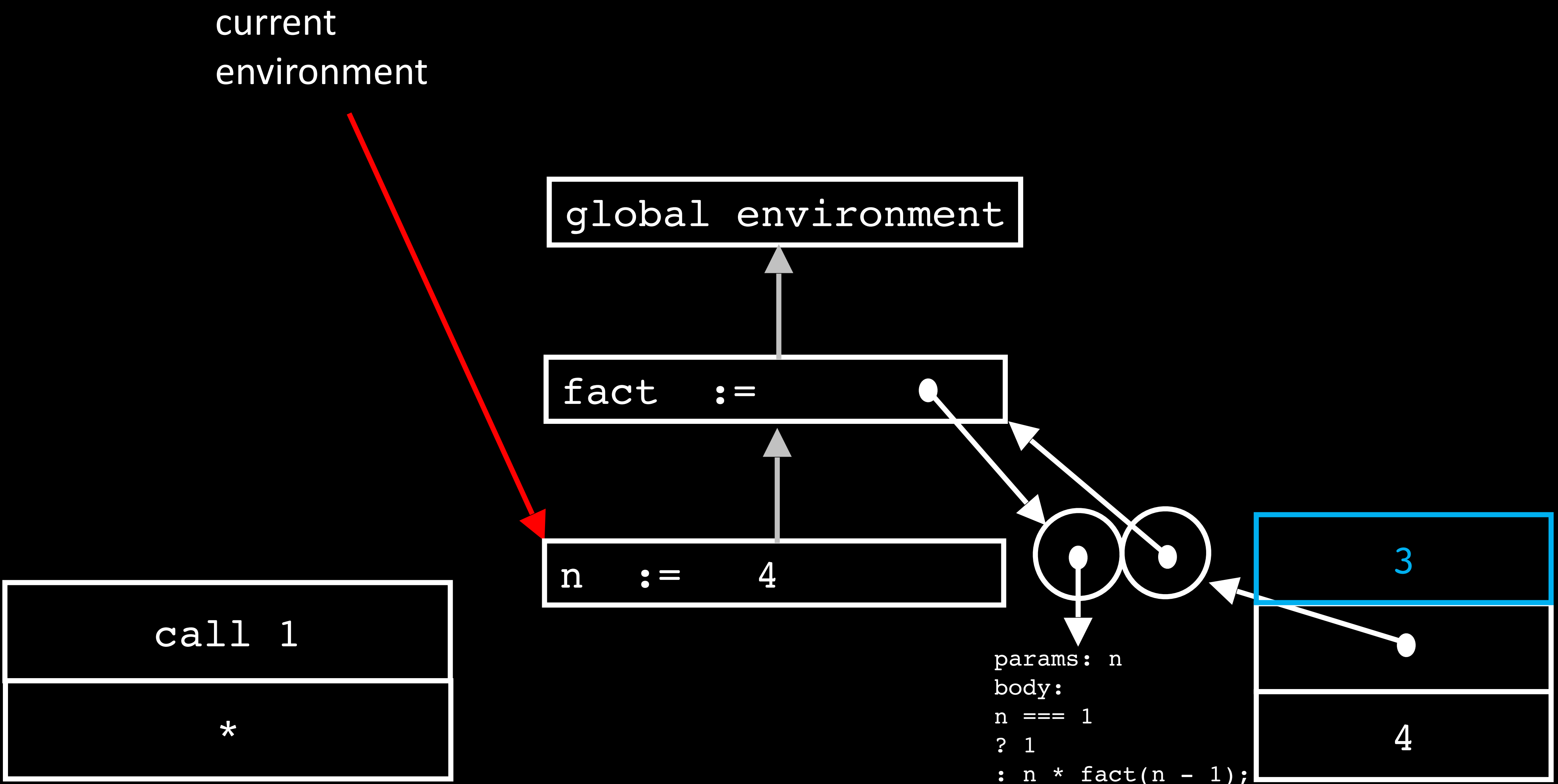
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```


current environment
global environment
fact :=
n := 4
1
-
call 1
*
4
4
params: n
body:
n === 1
? 1
: n * fact(n - 1);



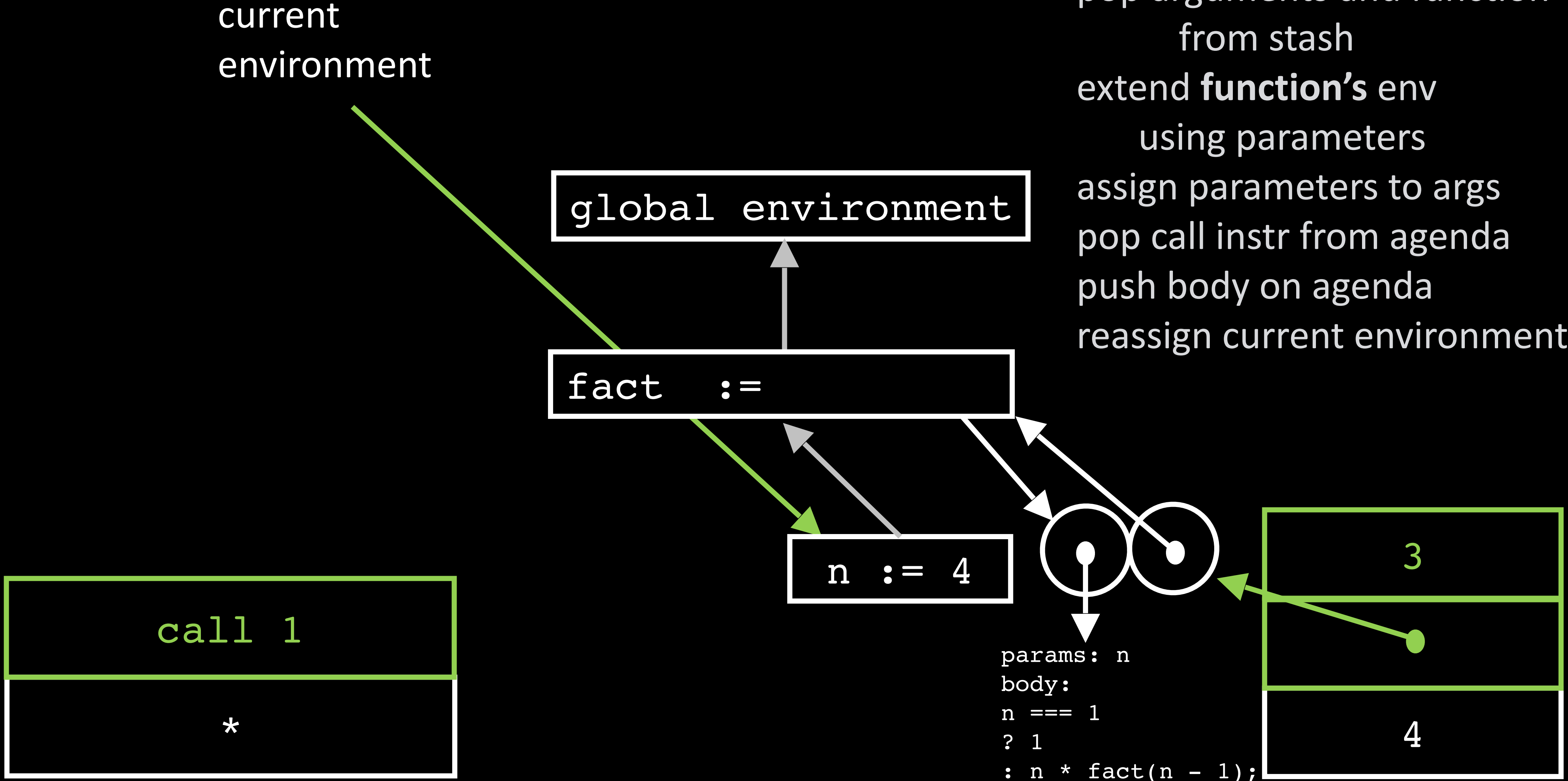
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```


current
environment
global environment
fact :=
n := 4
-
call 1
*
1
4
4
params: n
body:
n === 1
? 1
: n * fact(n - 1);



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Call instruction:
- pop arguments and function from stash
- extend **function's** env using parameters
- assign parameters to args
- pop call instr from agenda
- push body on agenda
- reassign current environment



Call instruction:

pop arguments and function from stash

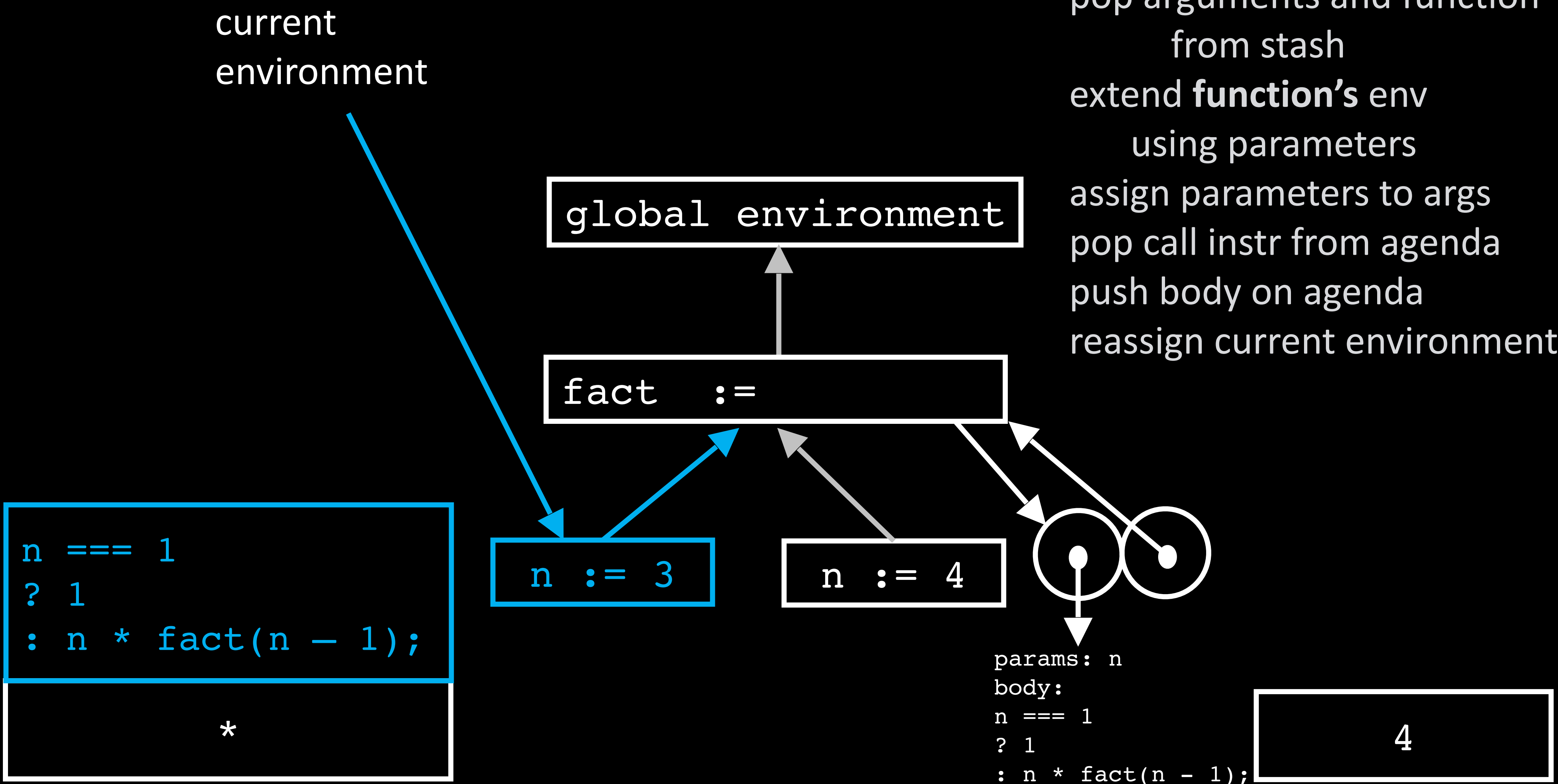
extend **function's** env using parameters

assign parameters to args

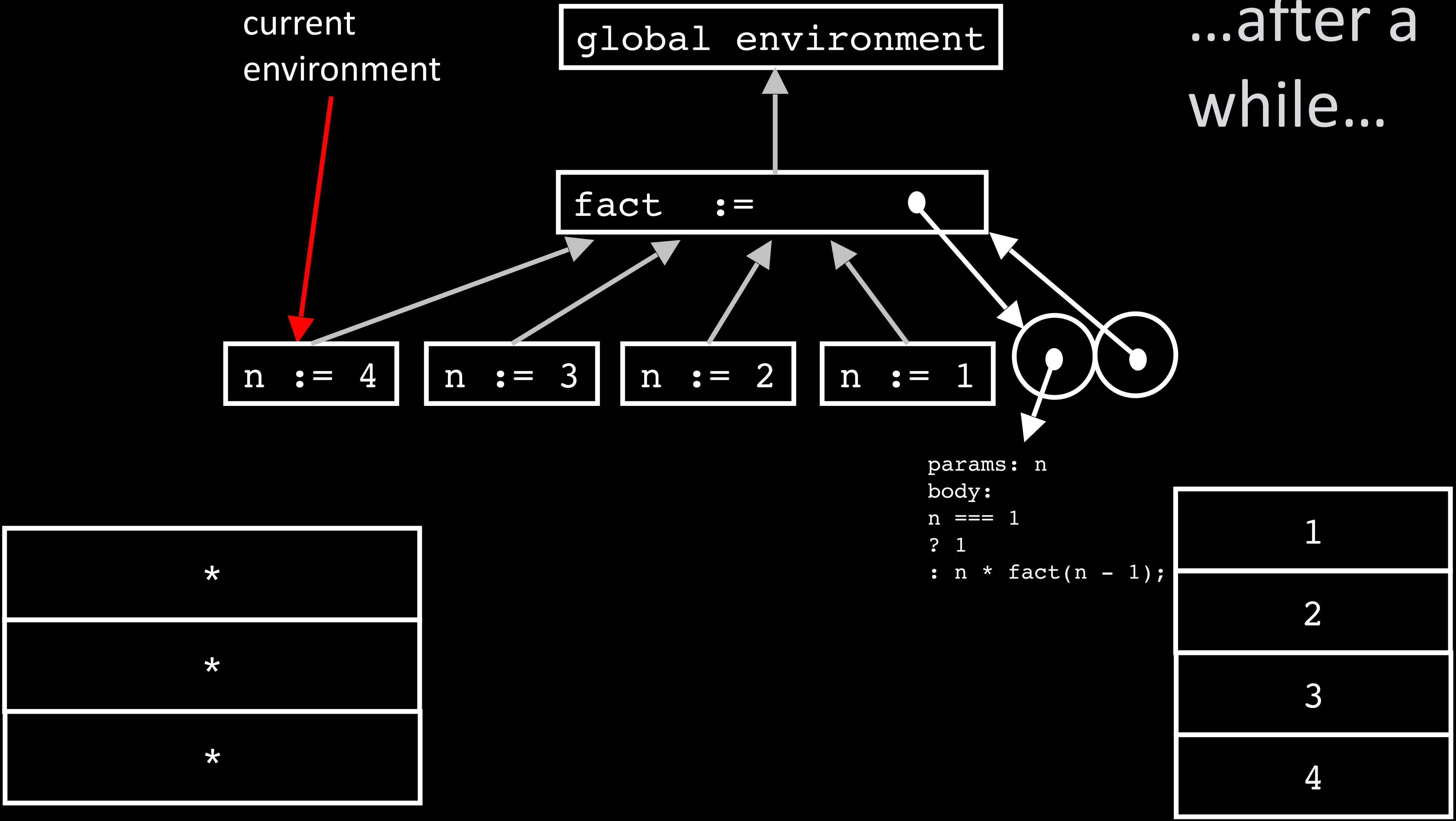
pop call instr from agenda

push body on agenda

reassign current environment



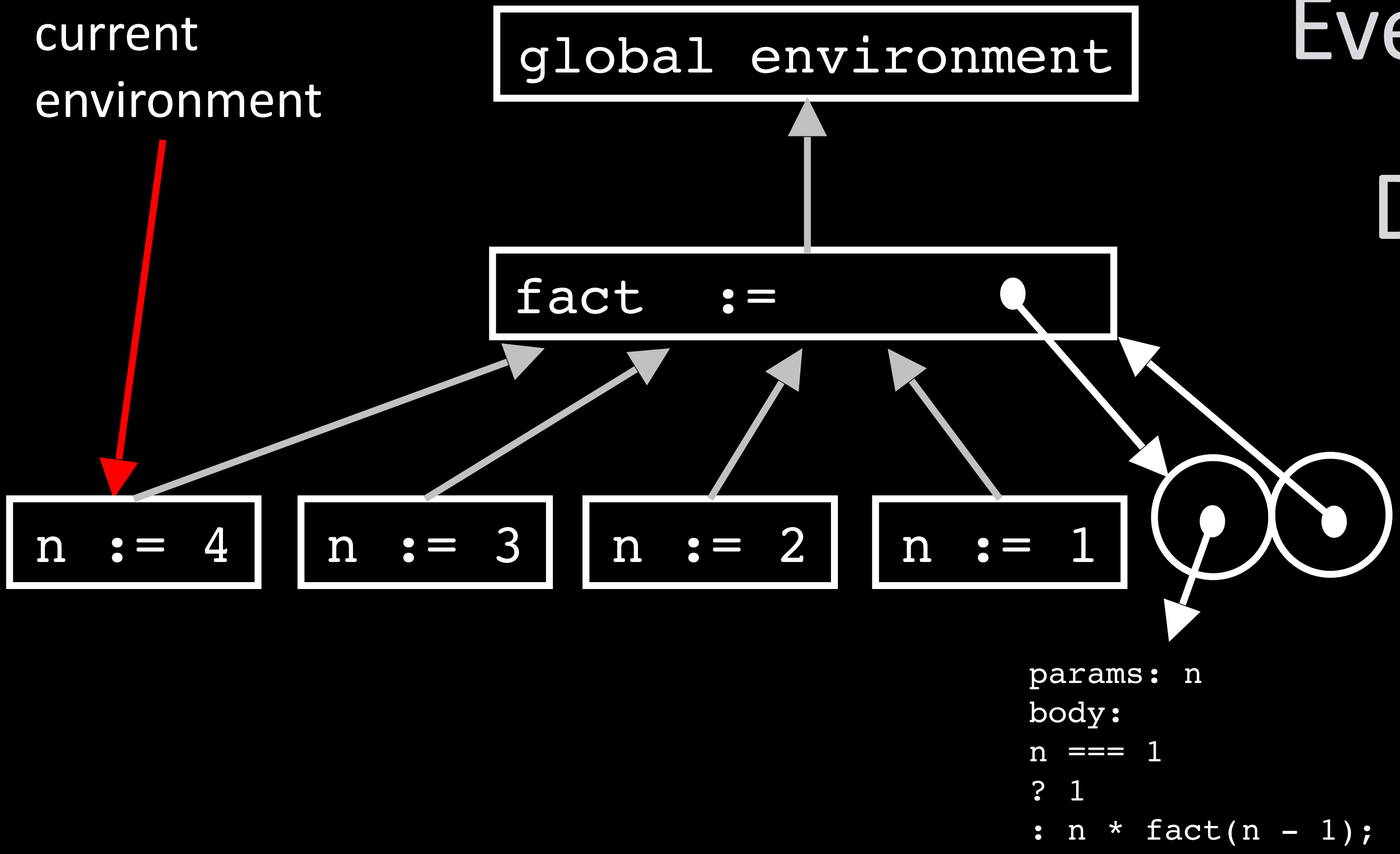
…after a while…



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Eventually...

Done!



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

24

The journey

- Calculator language
- Add conditionals, Booleans, sequences
- Add blocks, declarations, names
- Add function declaration and application (simple return)
- Restoring environments
- Further language features

The need for preserving environments (1)

```
const x = 1;
{
    const x = 42;
    display(x);
}
display(x);
```

The need for preserving environments (1)

```
const x = 1;
{
    const x = 42;
    display(x);
}
display(x);
```

After executing the block, we need the previous `x`.

The need for preserving environments (2)

```
const n = 42;

function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}

fact(4) + n;
```

The need for preserving environments (2)

```
const n = 42;

function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}

fact(4) + n;
```

- After returning from the recursive call, we need the previous **n**.

The need for preserving environments (2)

```
const n = 42;

function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}

fact(4) + n;
```

- After returning from the recursive call, we need the previous **n**.
- After fact(4), we need the n of the program environment.

Solution

Solution

Instructions that change current environment insert a “restore environment” instruction on the agenda:

Solution

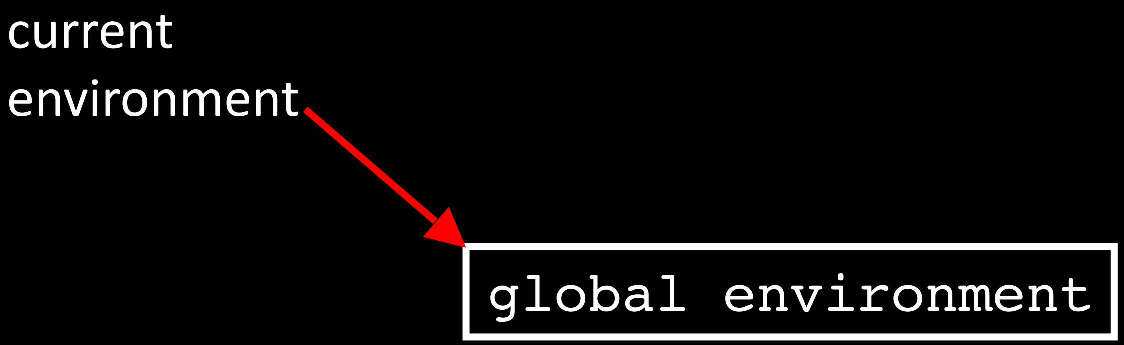
Instructions that change current environment insert a “restore environment” instruction on the agenda:

- Block

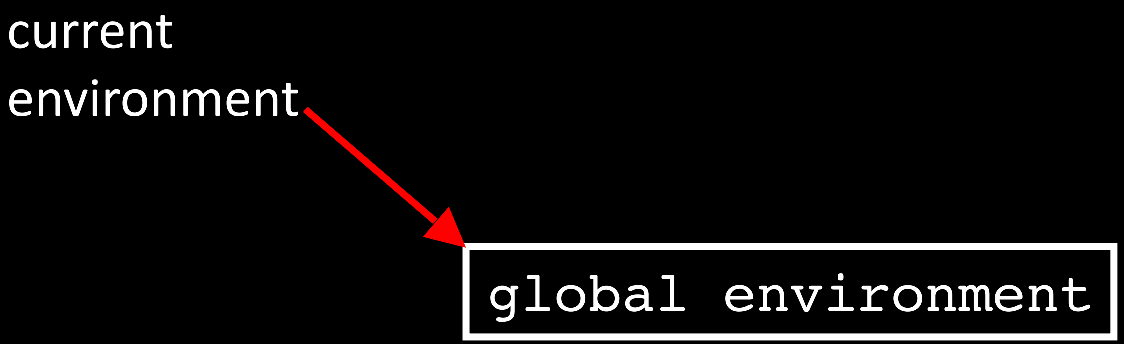
Solution

Instructions that change current environment insert a “restore environment” instruction on the agenda:

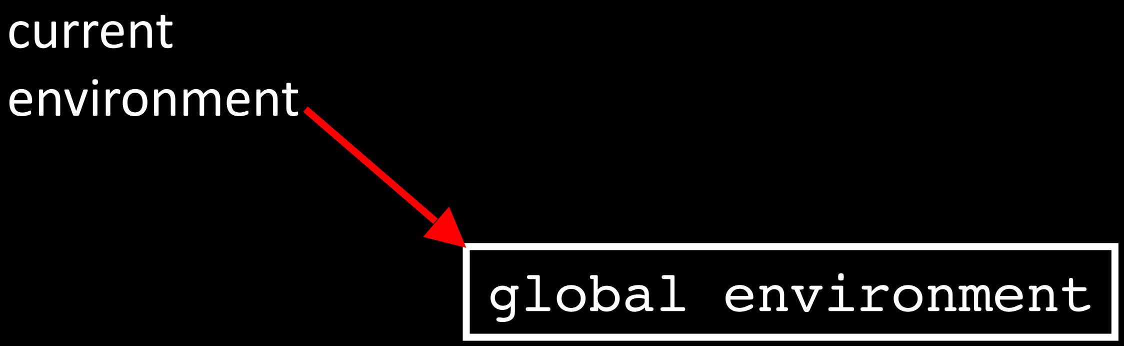
- Block
- Call instruction



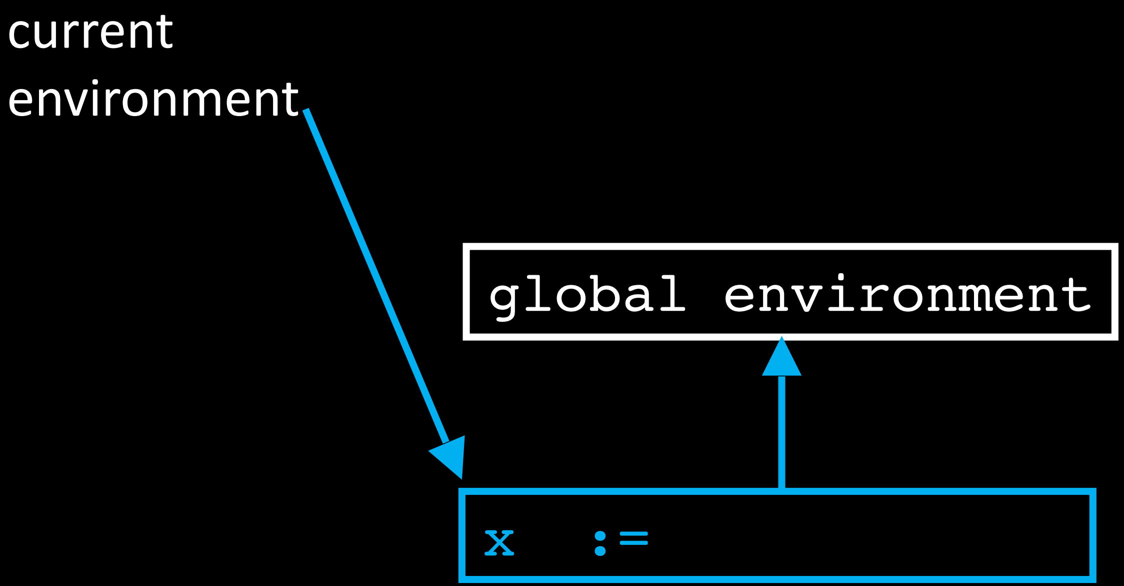
```
const x = 1;
{
    const x = 42;
    display(x);
}
display(x);
```



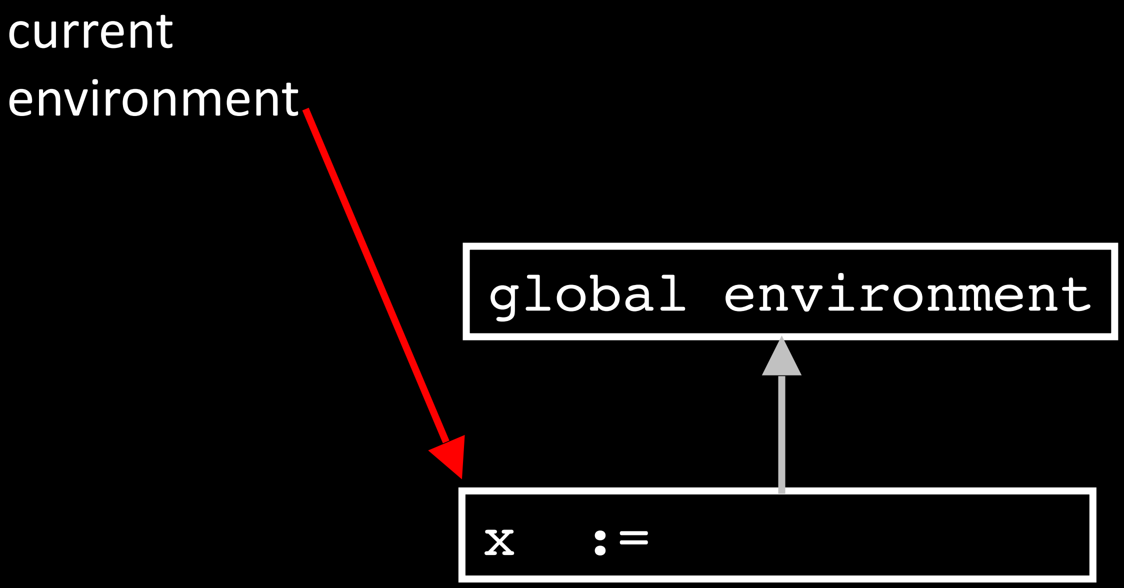
```
{
    const x = 1;
    {
        const x = 42;
        display(x);
    }
    display(x);
}
```



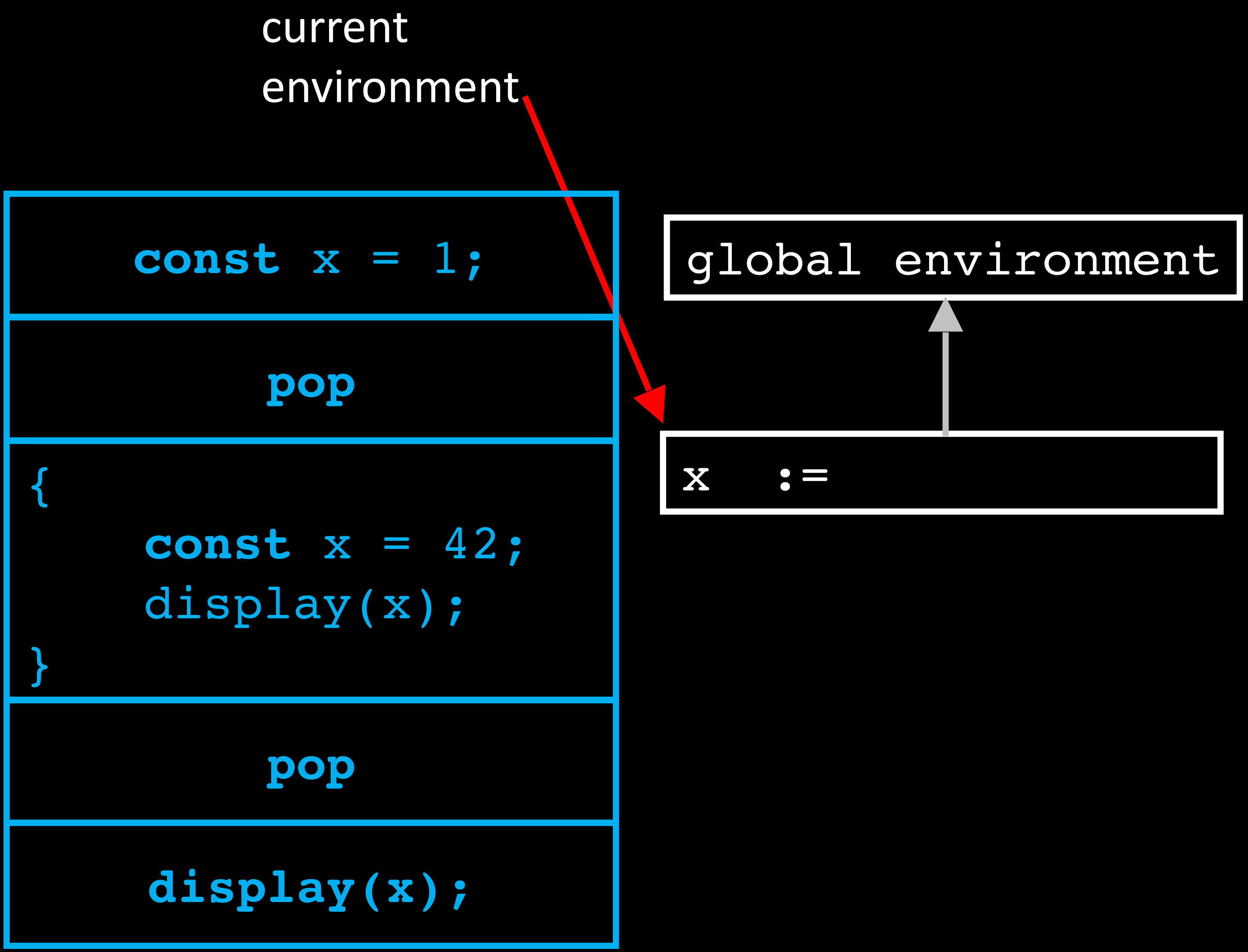
```
{
    const x = 1;
    {
        const x = 42;
        display(x);
    }
    display(x);
}
```



```
const x = 1;
{
    const x = 42;
    display(x);
}
display(x);
```



```
const x = 1;
{
    const x = 42;
    display(x);
}
display(x);
```



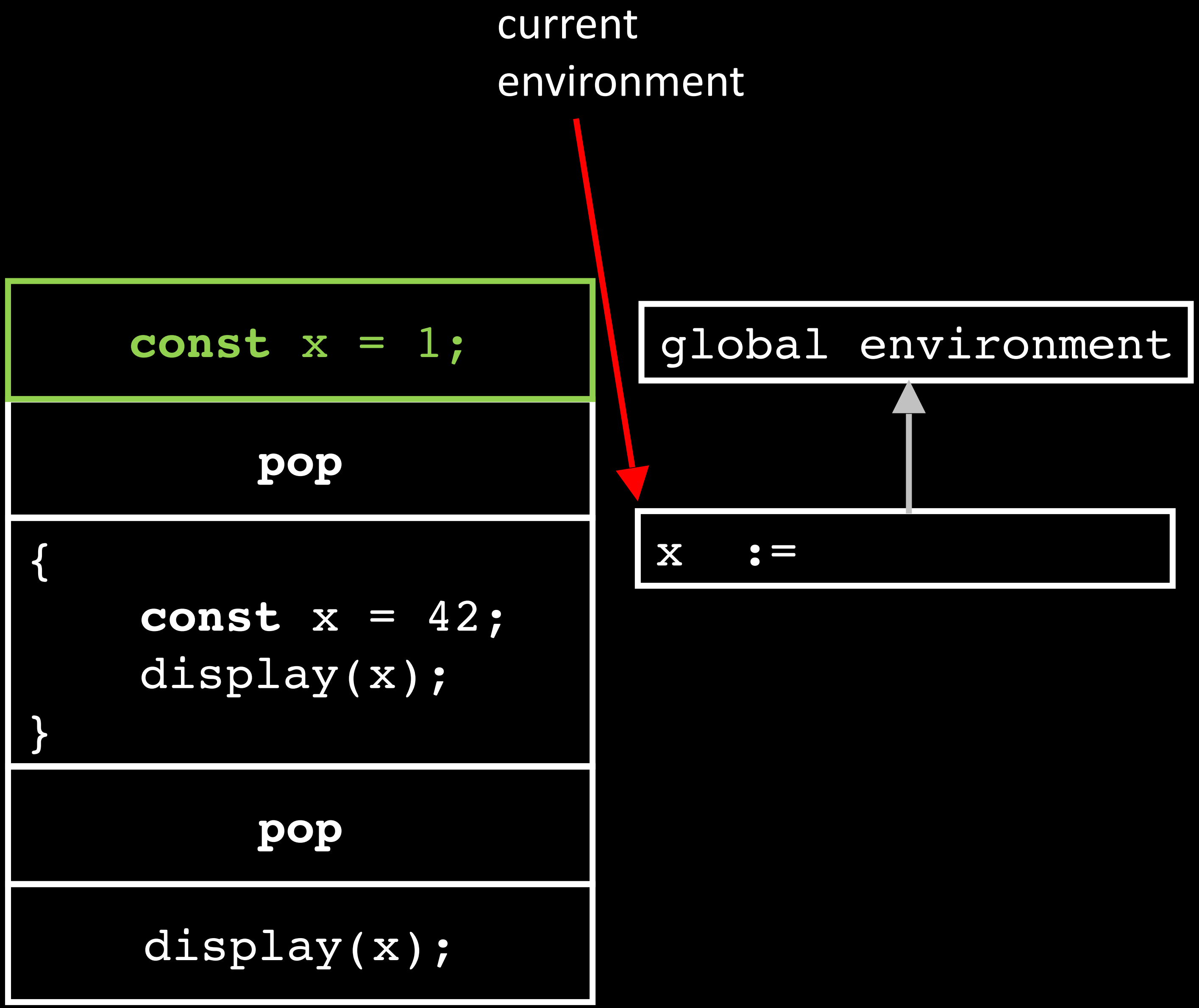
```
const x = 1;
```

```
pop
```

```
{
    const x = 42;
    display(x);
}
```

```
pop
```

```
display(x);
```



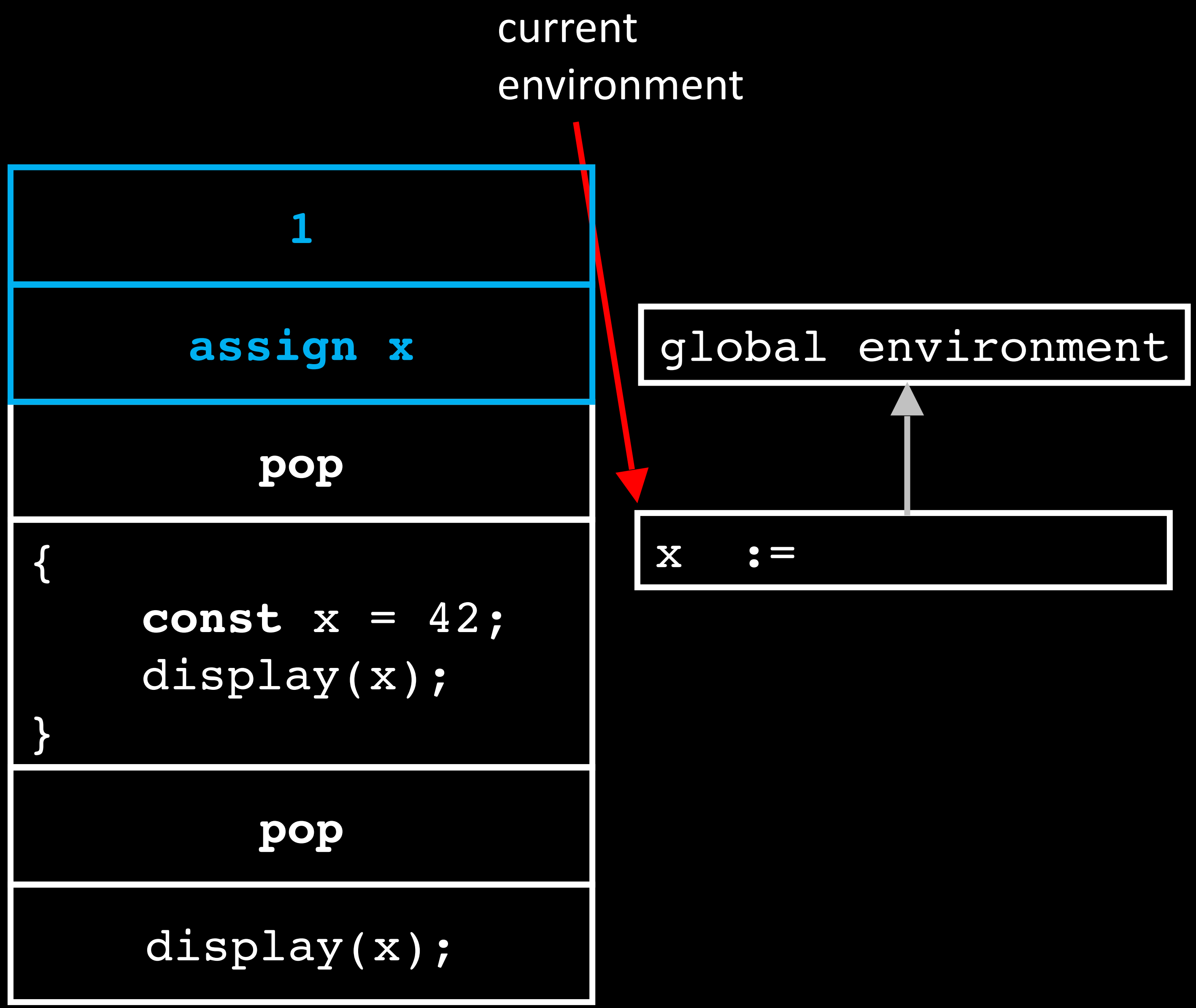
```
const x = 1;
```

```
pop
```

```
{
    const x = 42;
    display(x);
}
```

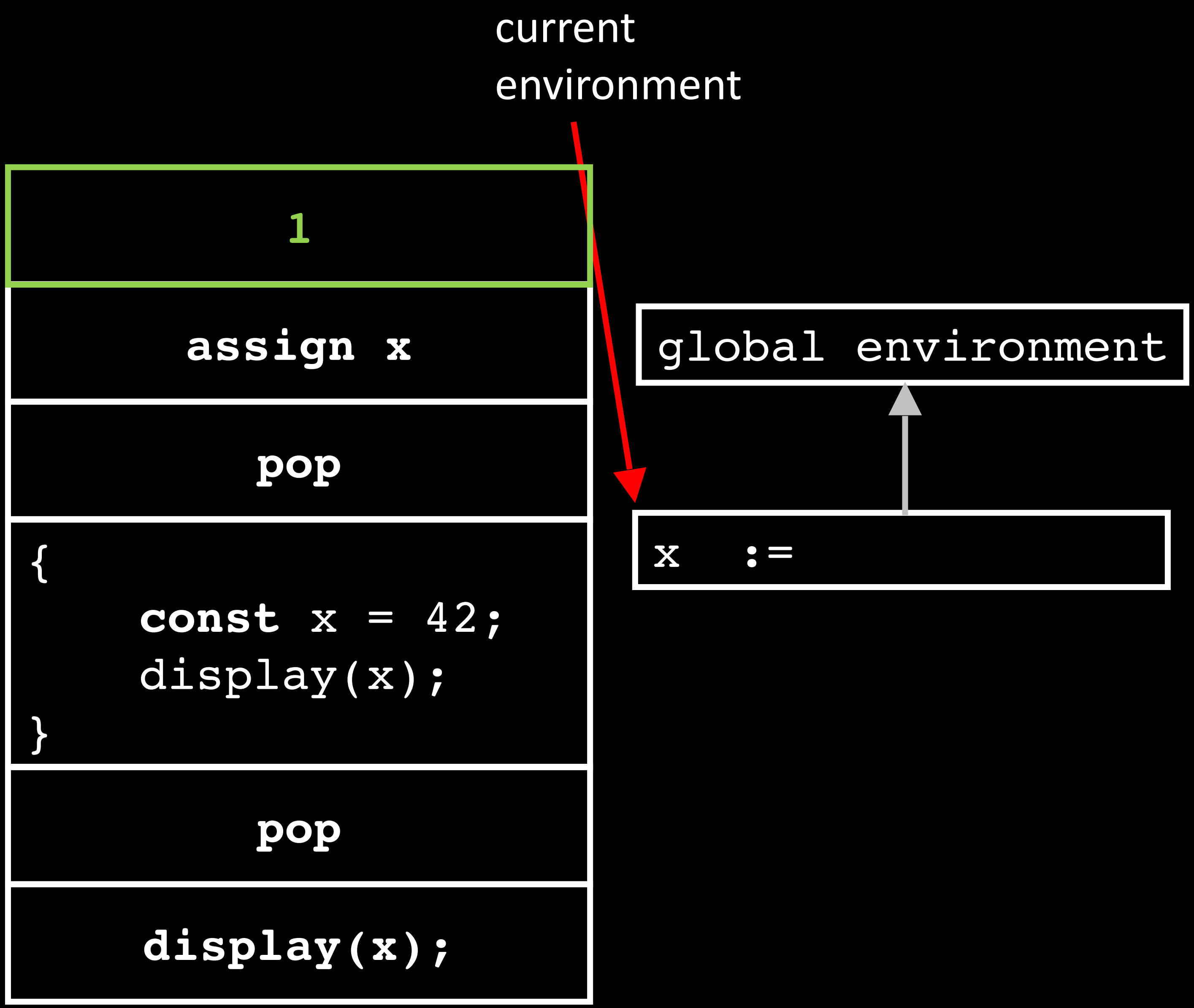
```
pop
```

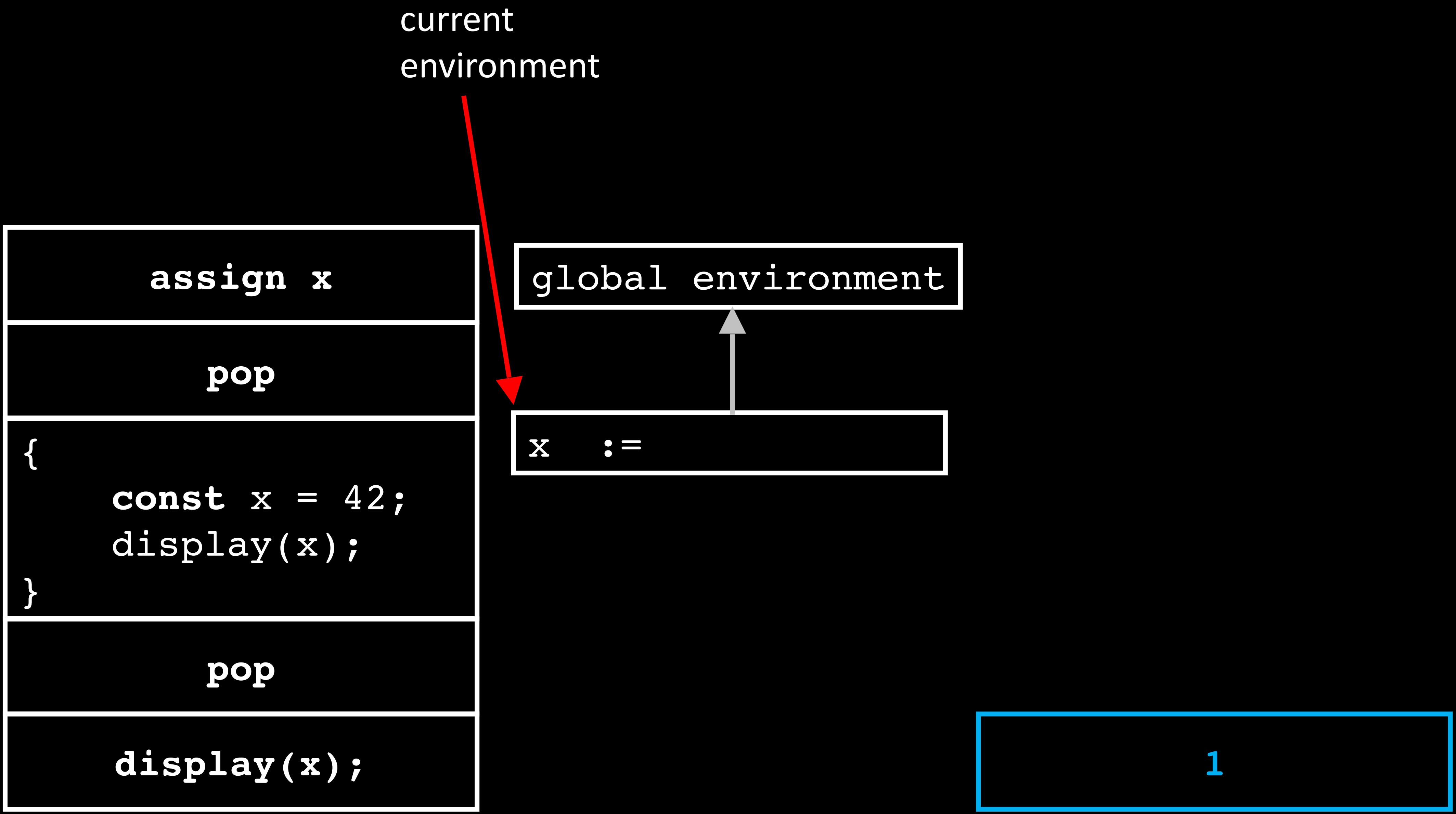
```
display(x);
```

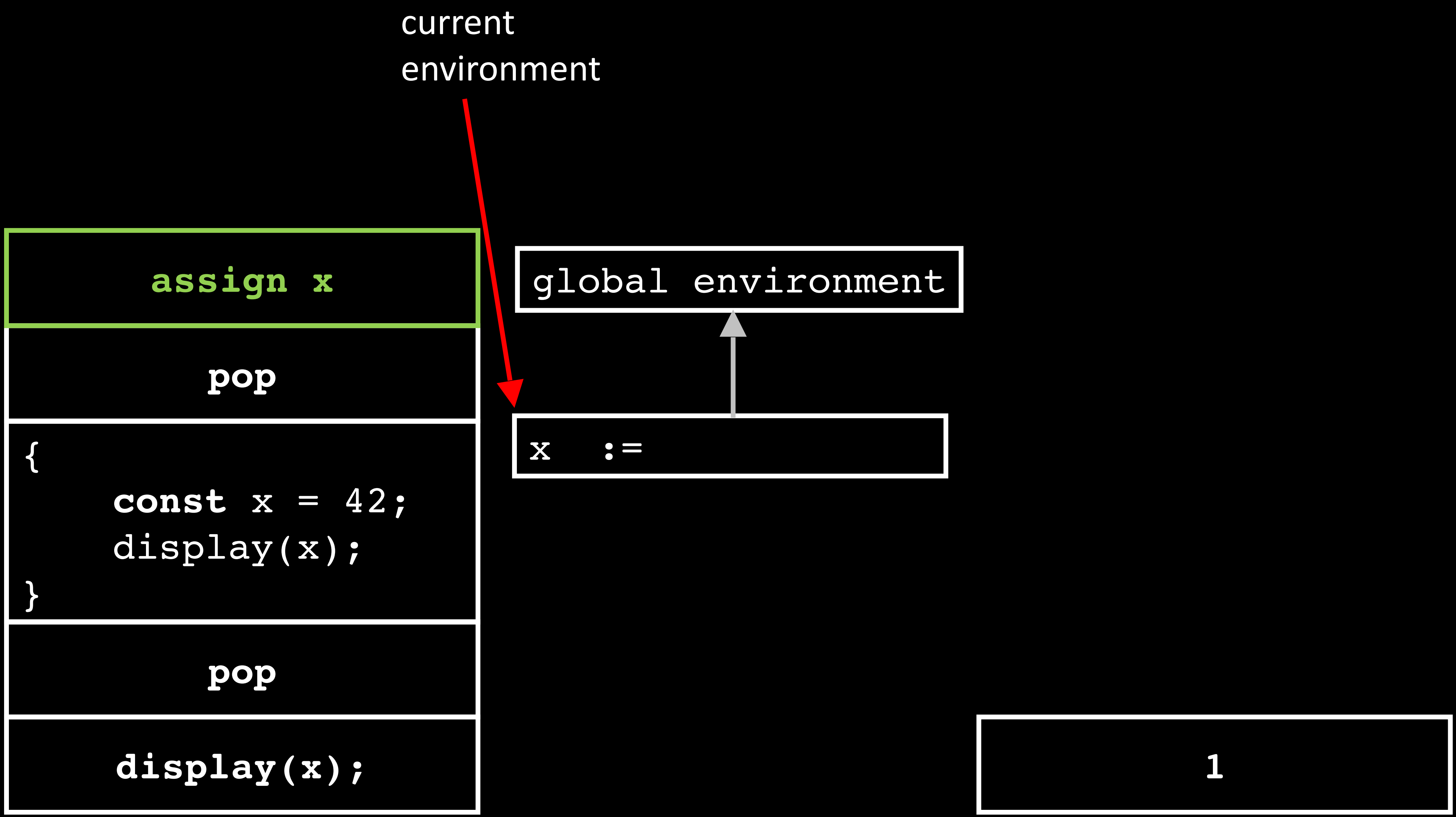


```
{
    const x = 42;
    display(x);
}
```

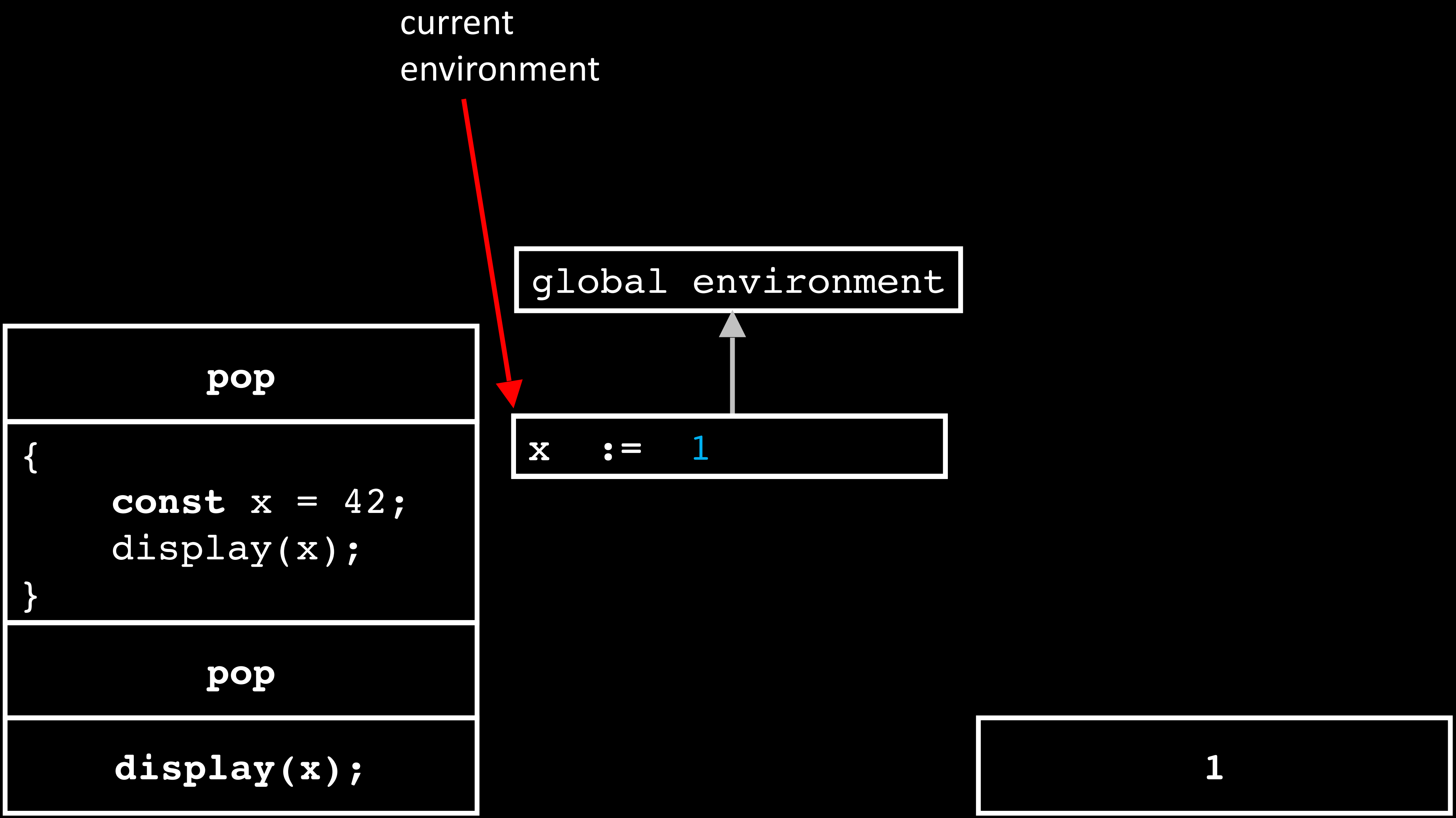
```
display(x);
```

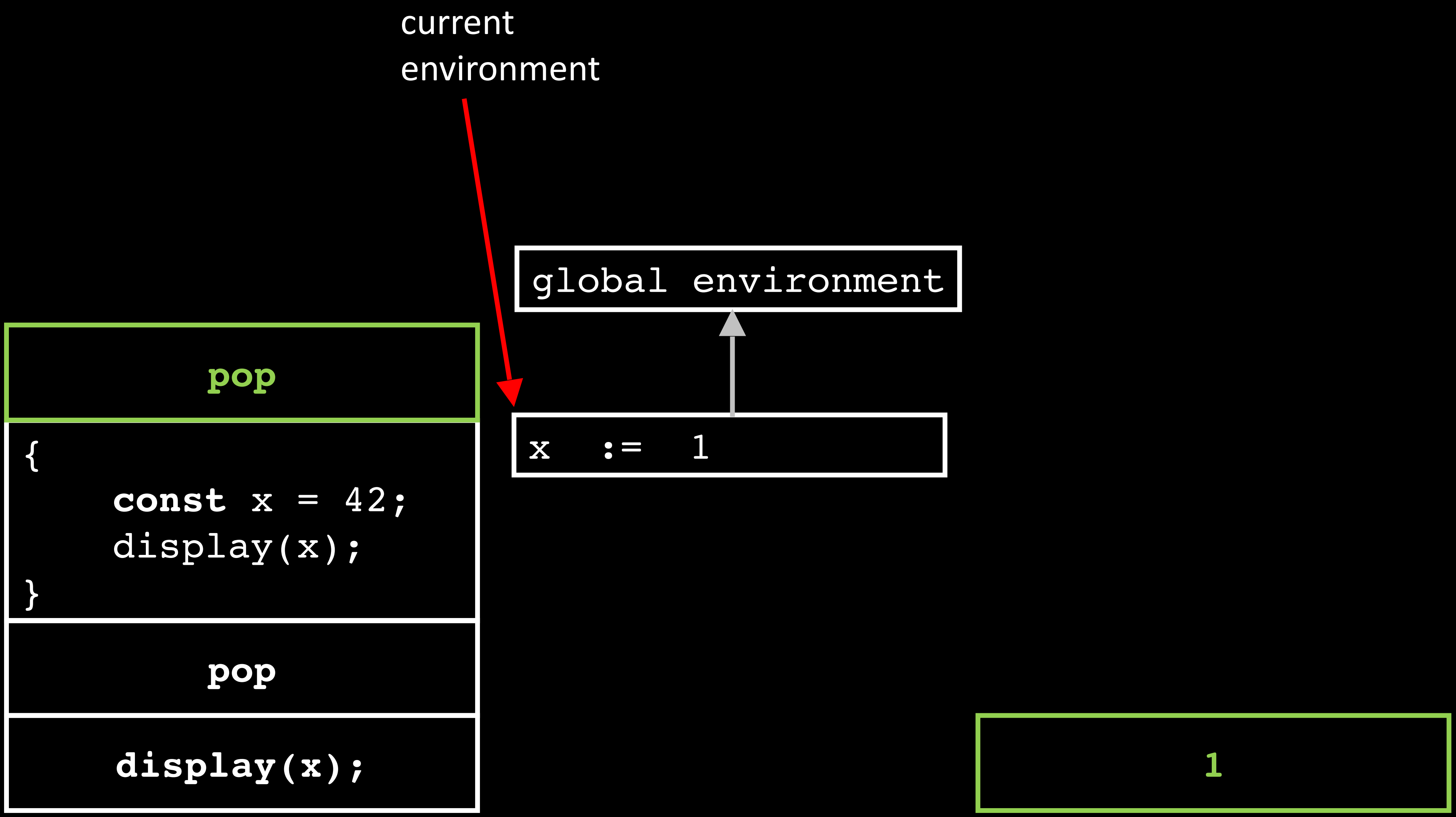

current
environment
1
assign x
pop
{
const x = 42;
display(x);
}
pop
display(x);
global environment
x :=

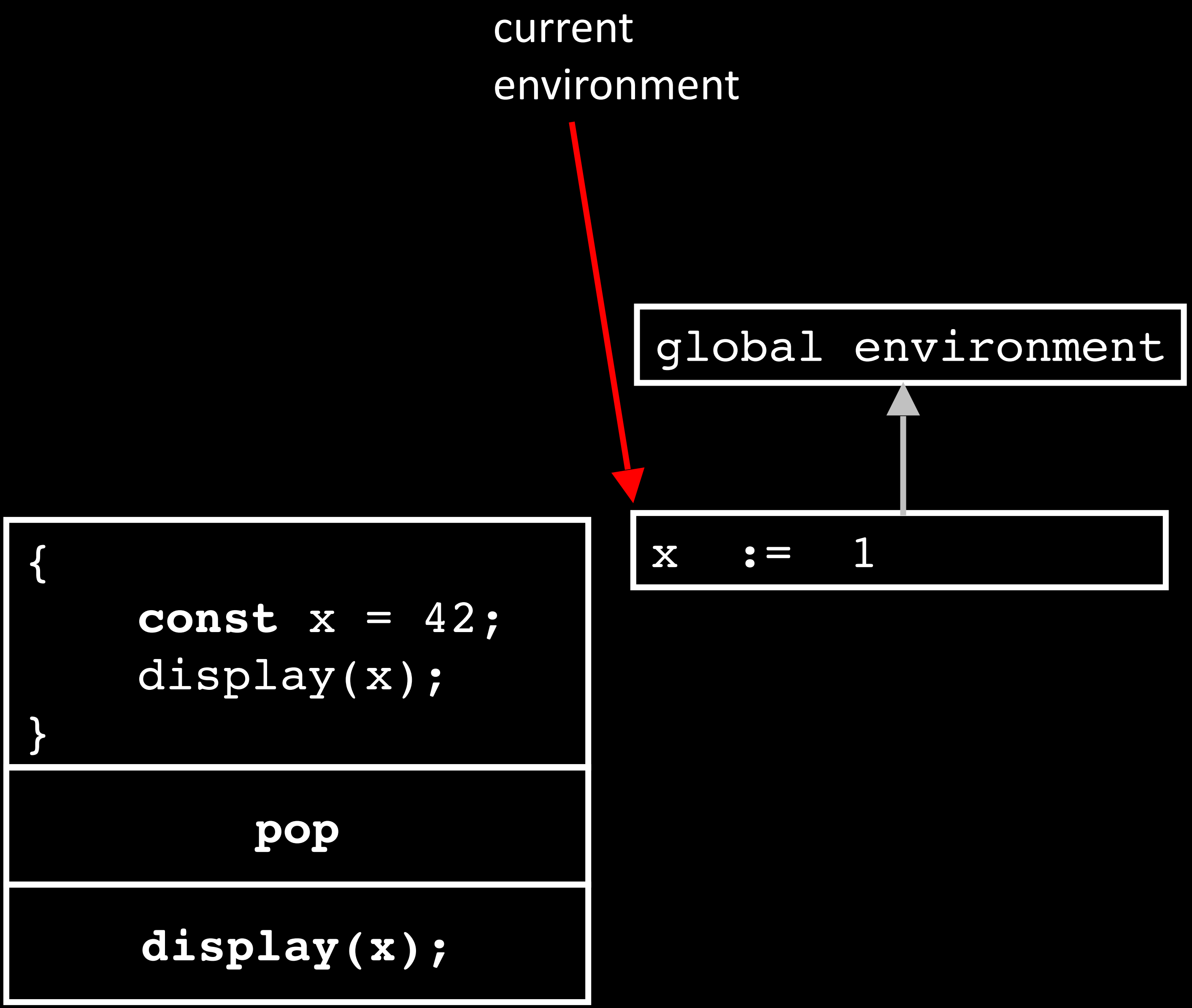

current environment
assign x
pop
{
    const x = 42;
    display(x);
}
pop
display(x);
global environment
x :=
1



```
{
    const x = 42;
    display(x);
}
```


current
environment
global environment
x := 1
pop
{
const x = 42;
display(x);
}
pop
display(x);
1

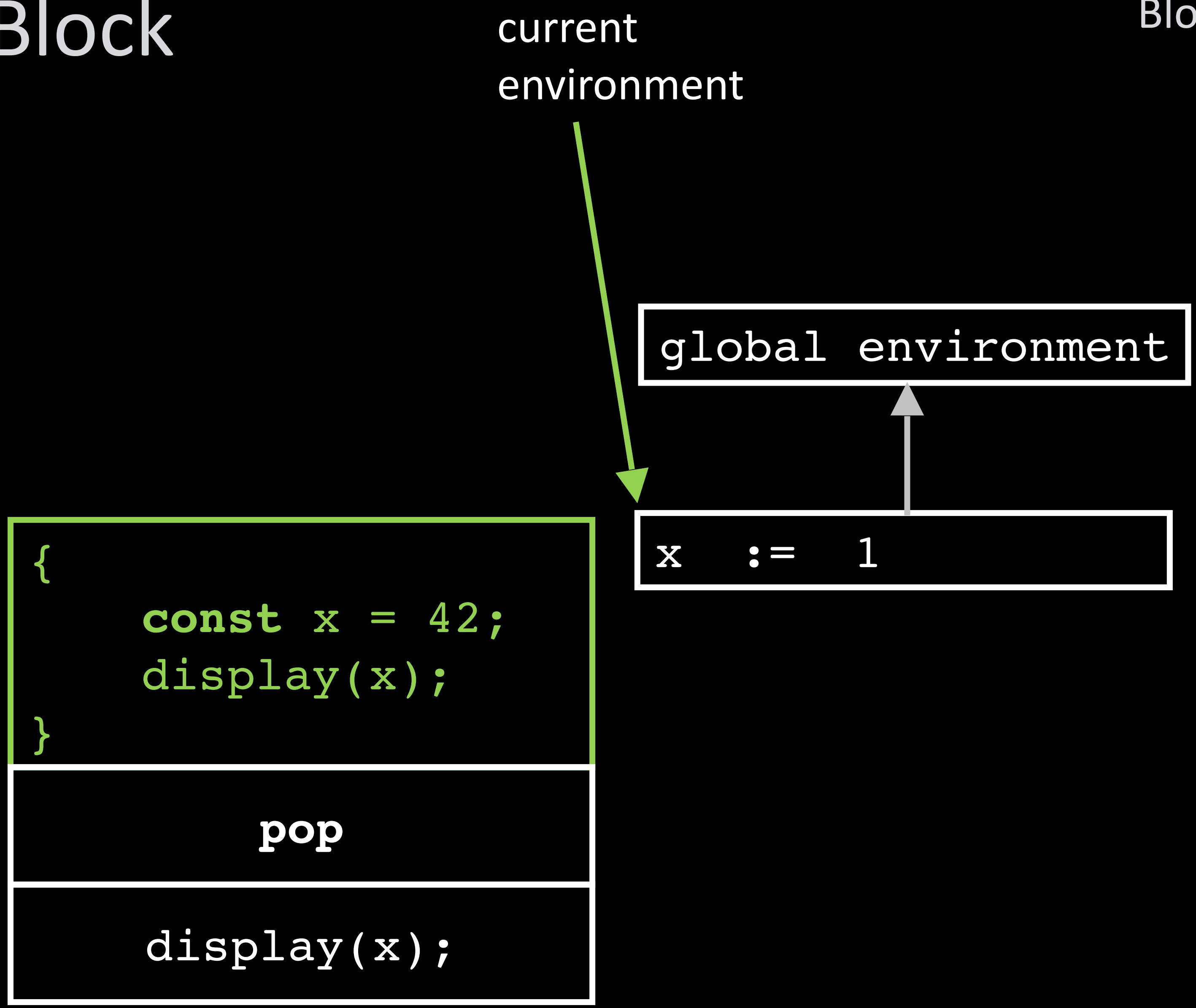

current environment
global environment
x := 1
pop
{
const x = 42;
display(x);
}
pop
display(x);
1



```
{
    const x = 42;
    display(x);
}
```

```
display(x);
```

Block



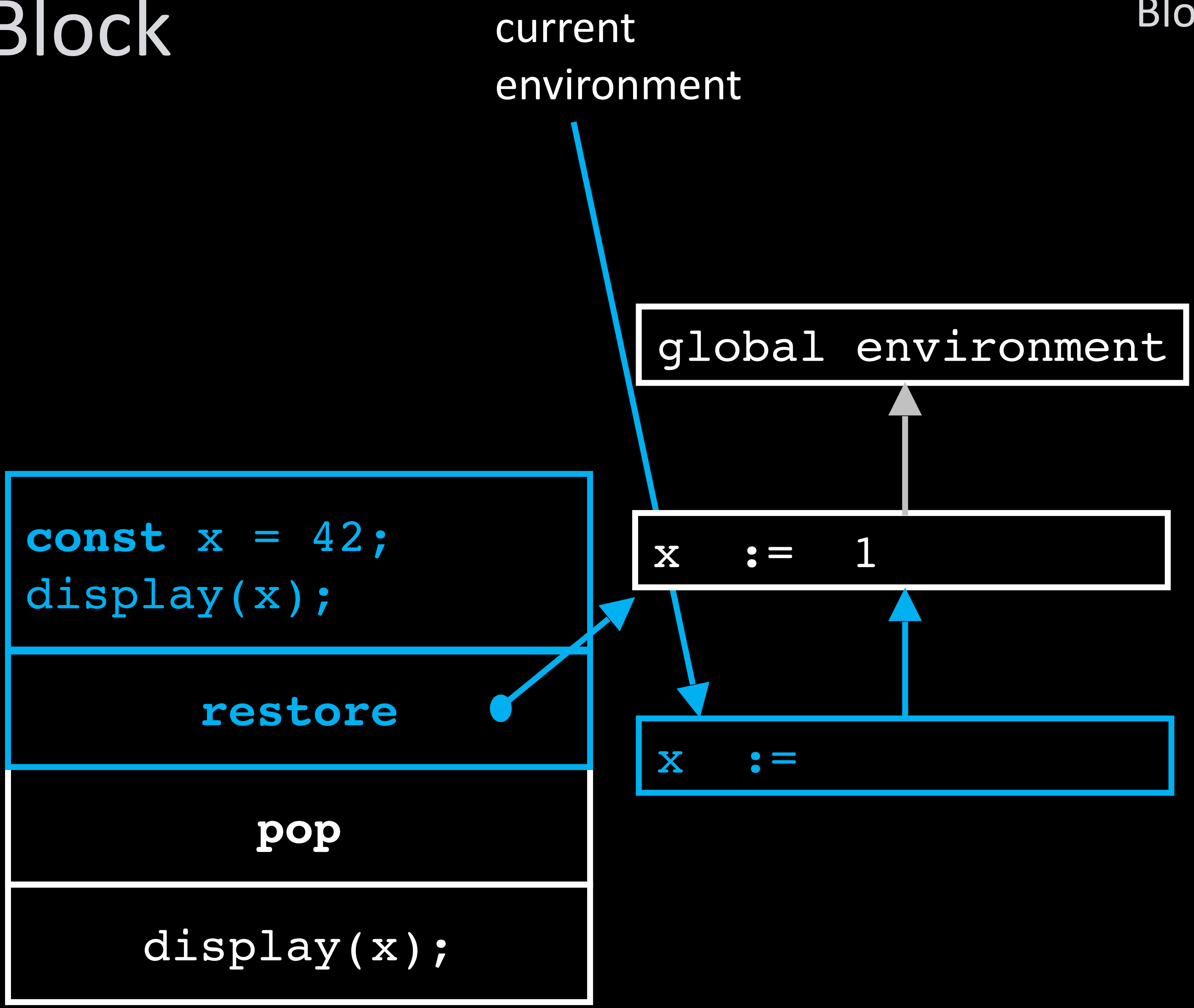
```
{
    const x = 42;
    display(x);
}
```

```
display(x);
```

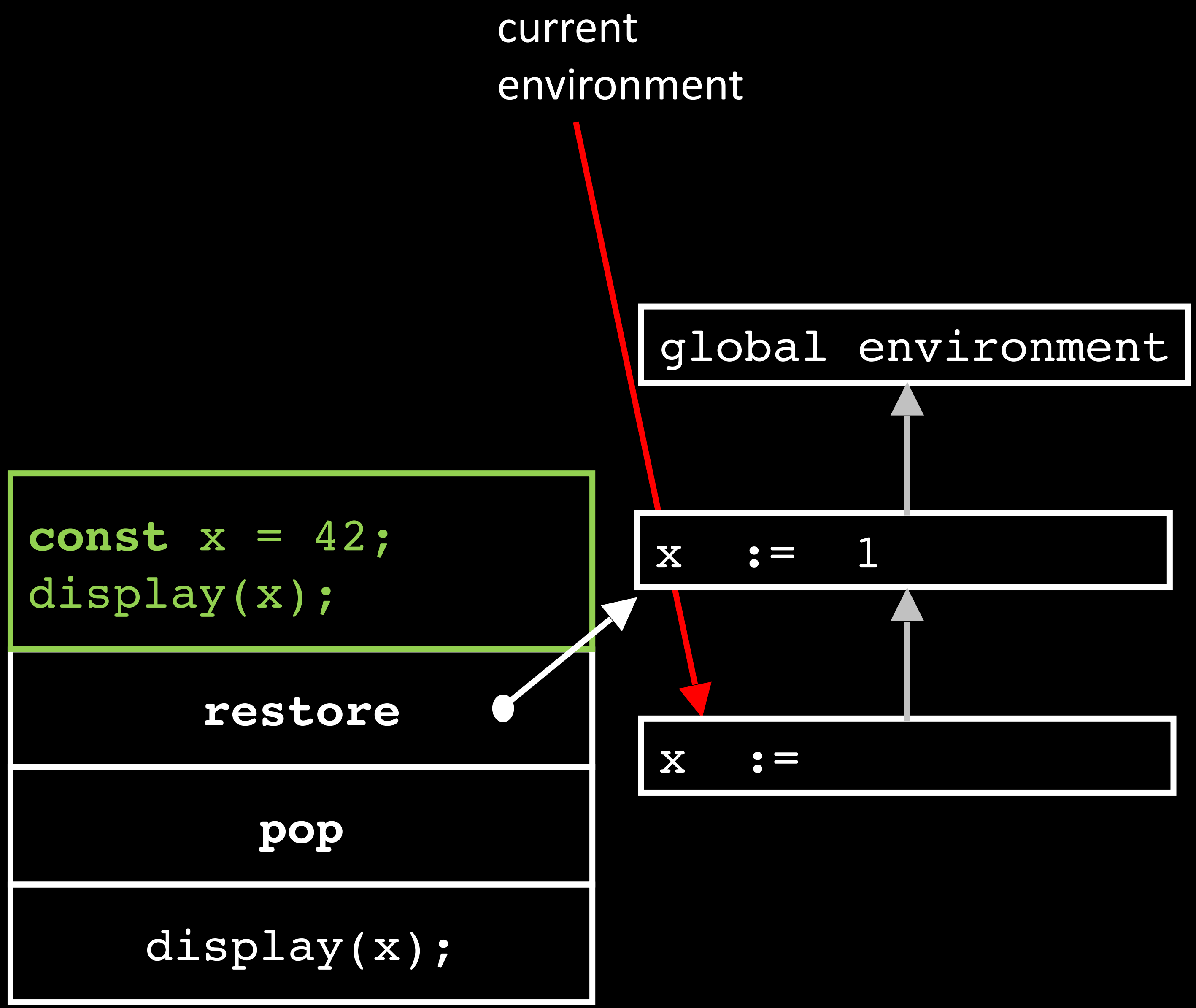
Block:

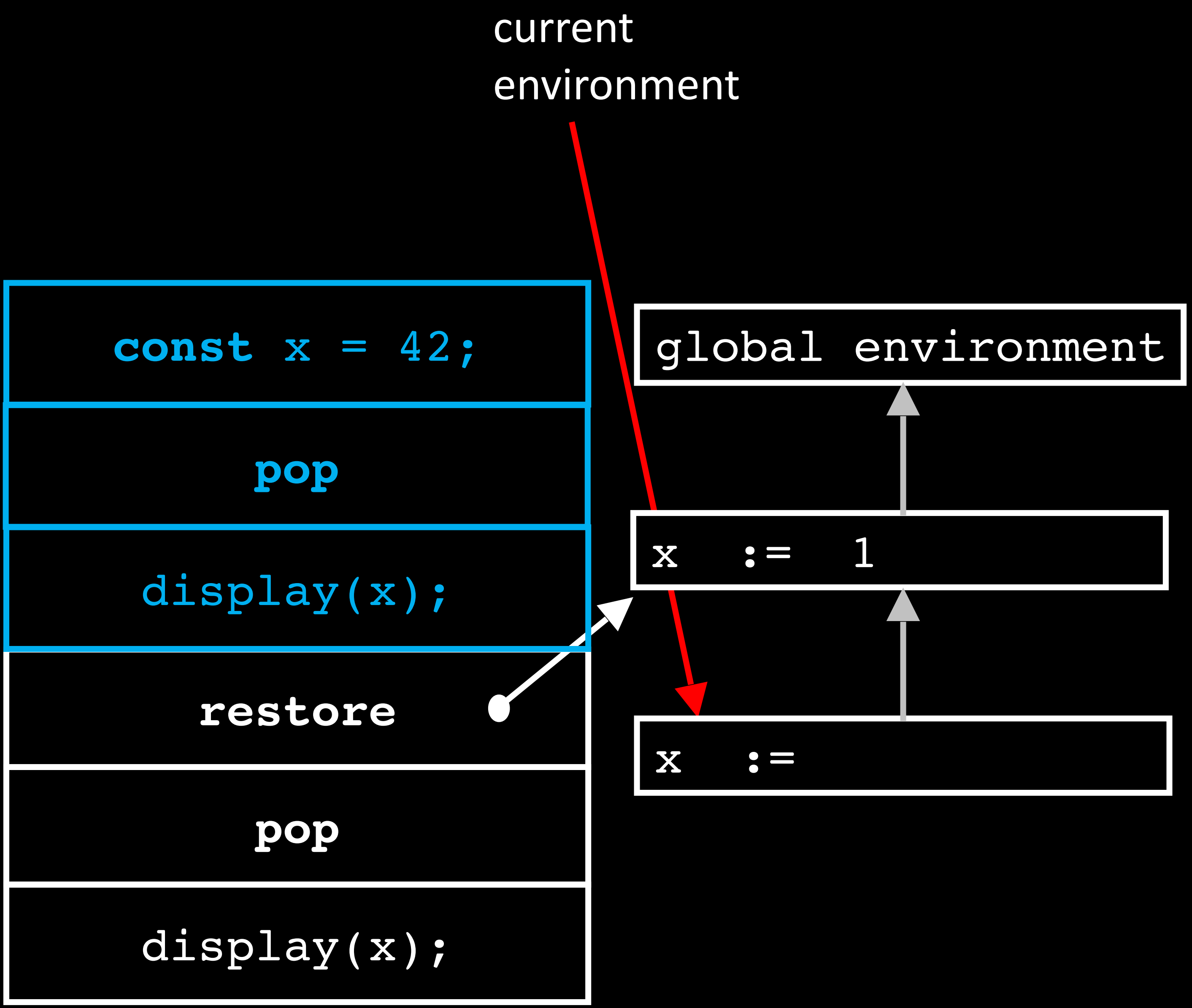
- **pop block from agenda**
- **push instruction to restore current env on agenda**
- **push body of block on agenda**
- extend current environment by frame with declared names
- set current environment to start at new frame

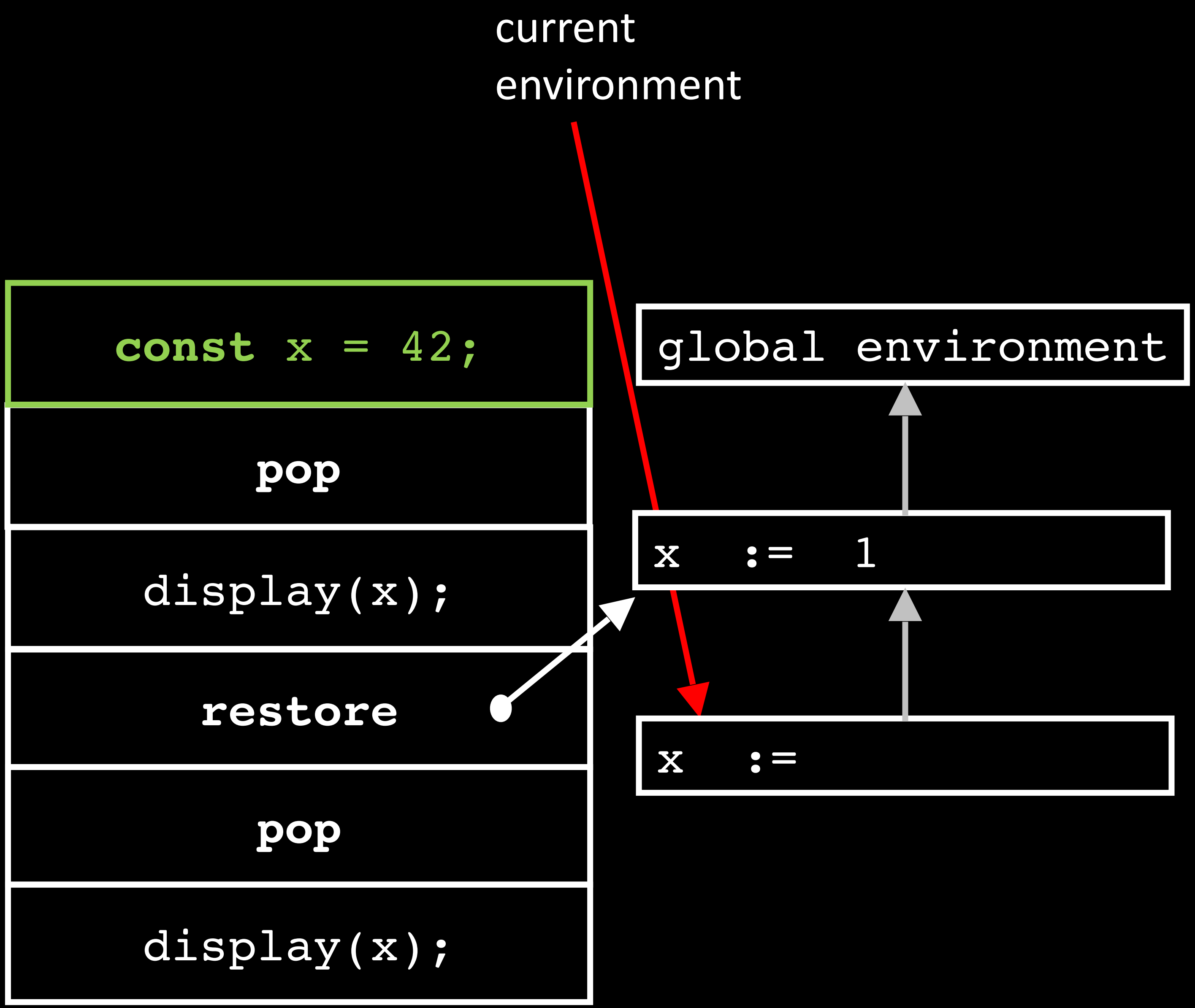
Block

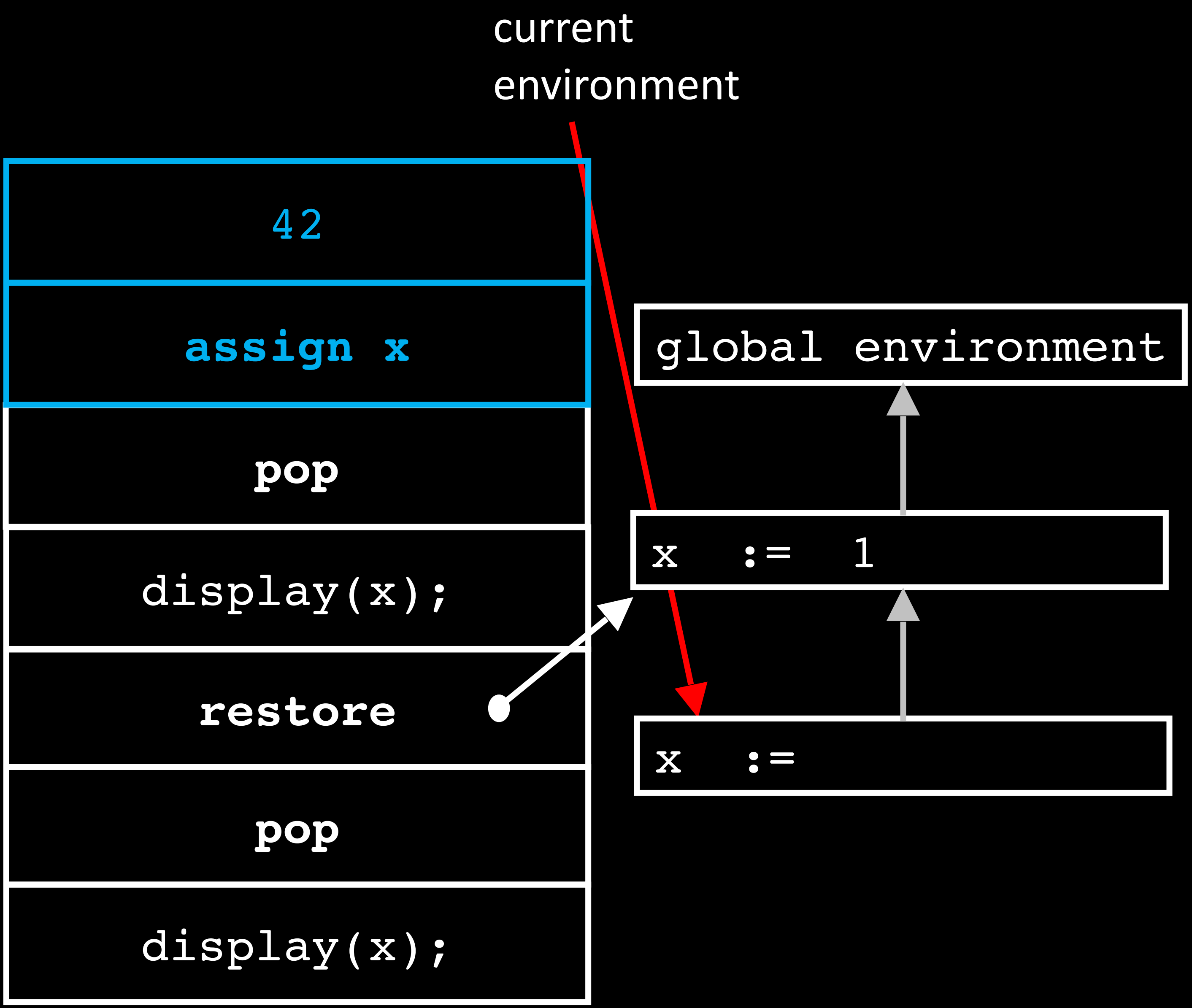


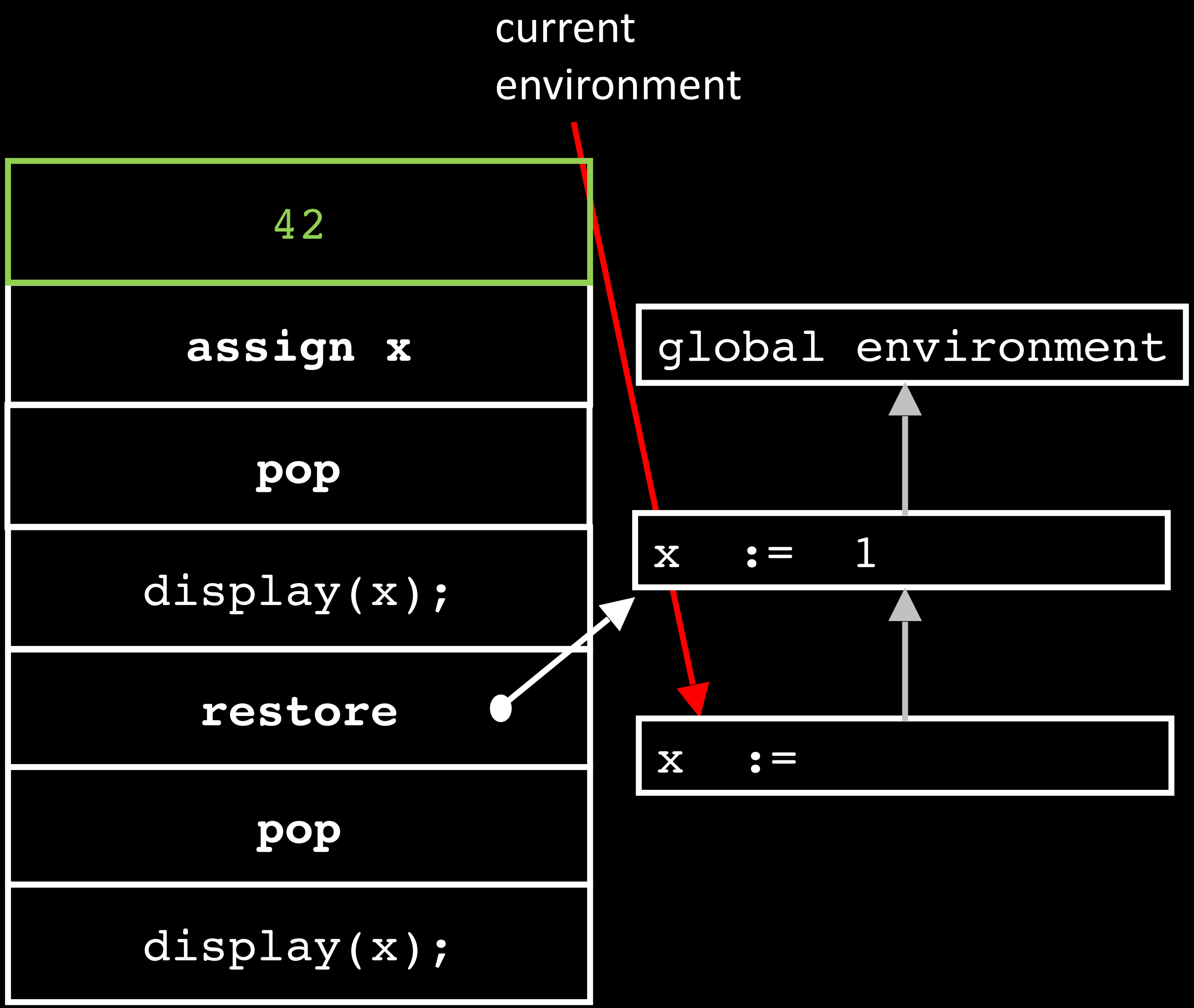
```
Block:
    pop block from agenda
    push instruction to restore
        current env on agenda
    push body of block on agenda
    **extend current environment by**
        **frame with declared names**
    **set current environment to**
        **start at new frame**
```

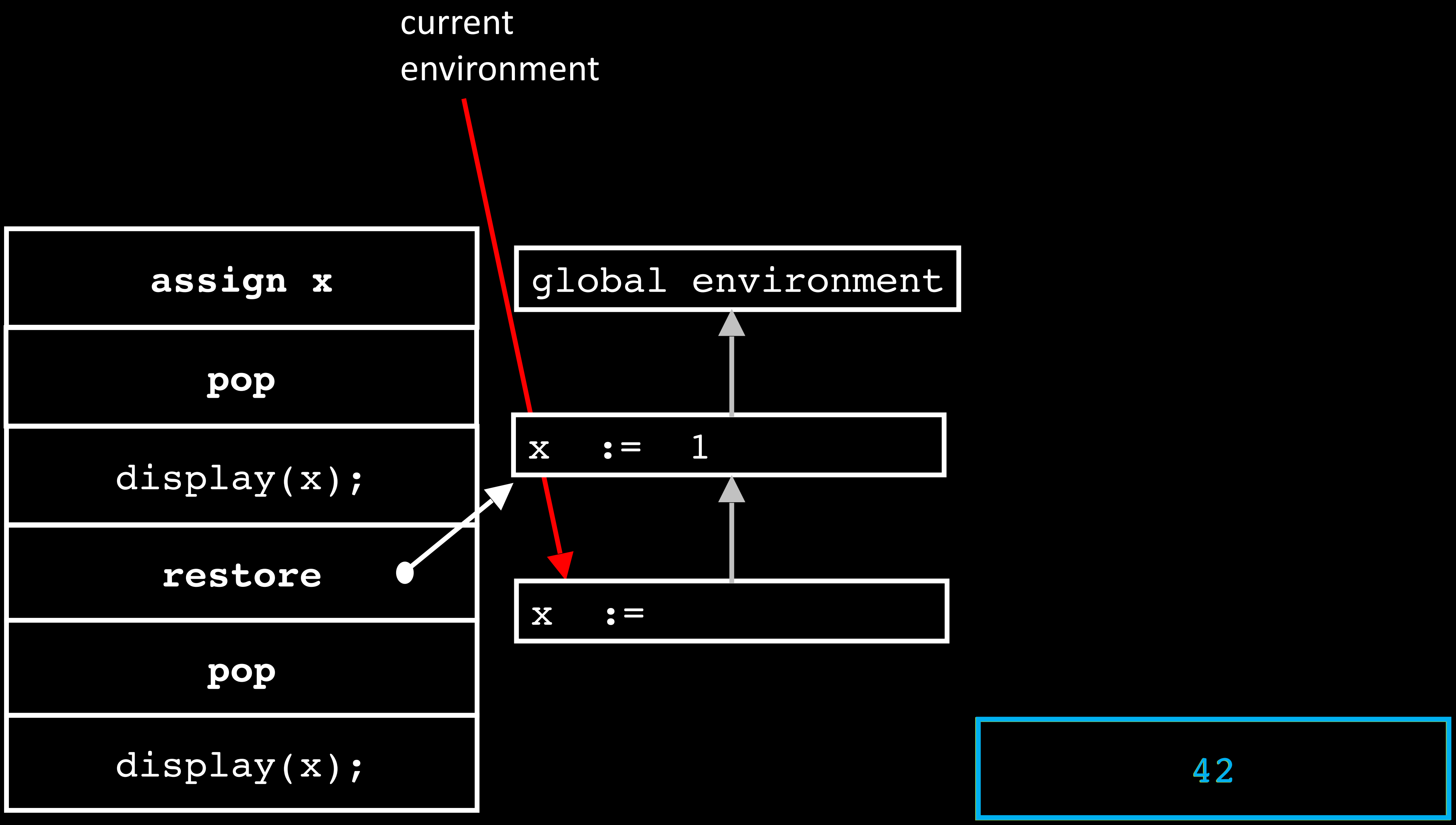

current environment
global environment
x := 1
x :=
const x = 42;
display(x);
restore
pop
display(x);

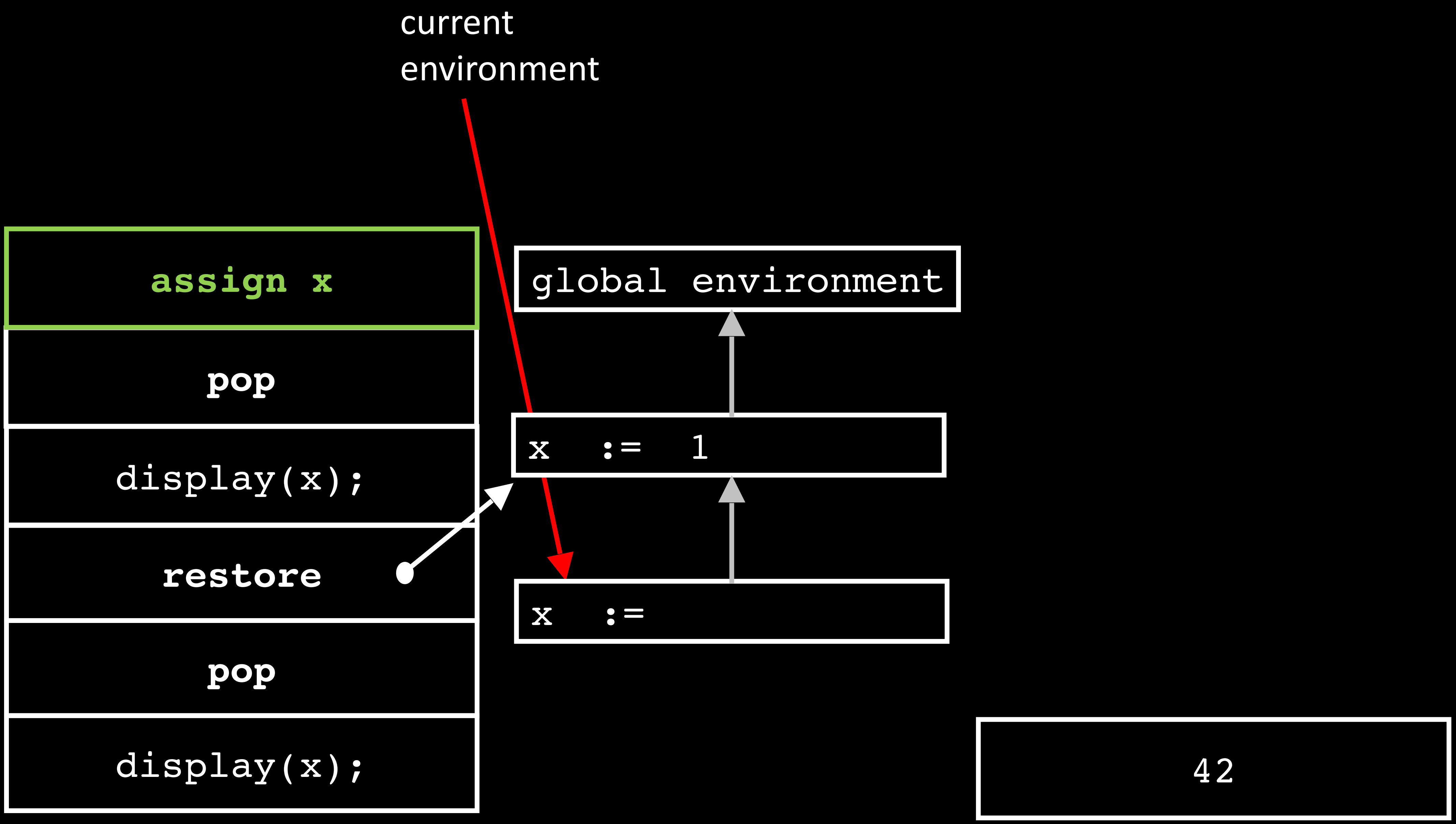

current environment
const x = 42;
pop
display(x);
restore
pop
display(x);
global environment
x := 1
x :=

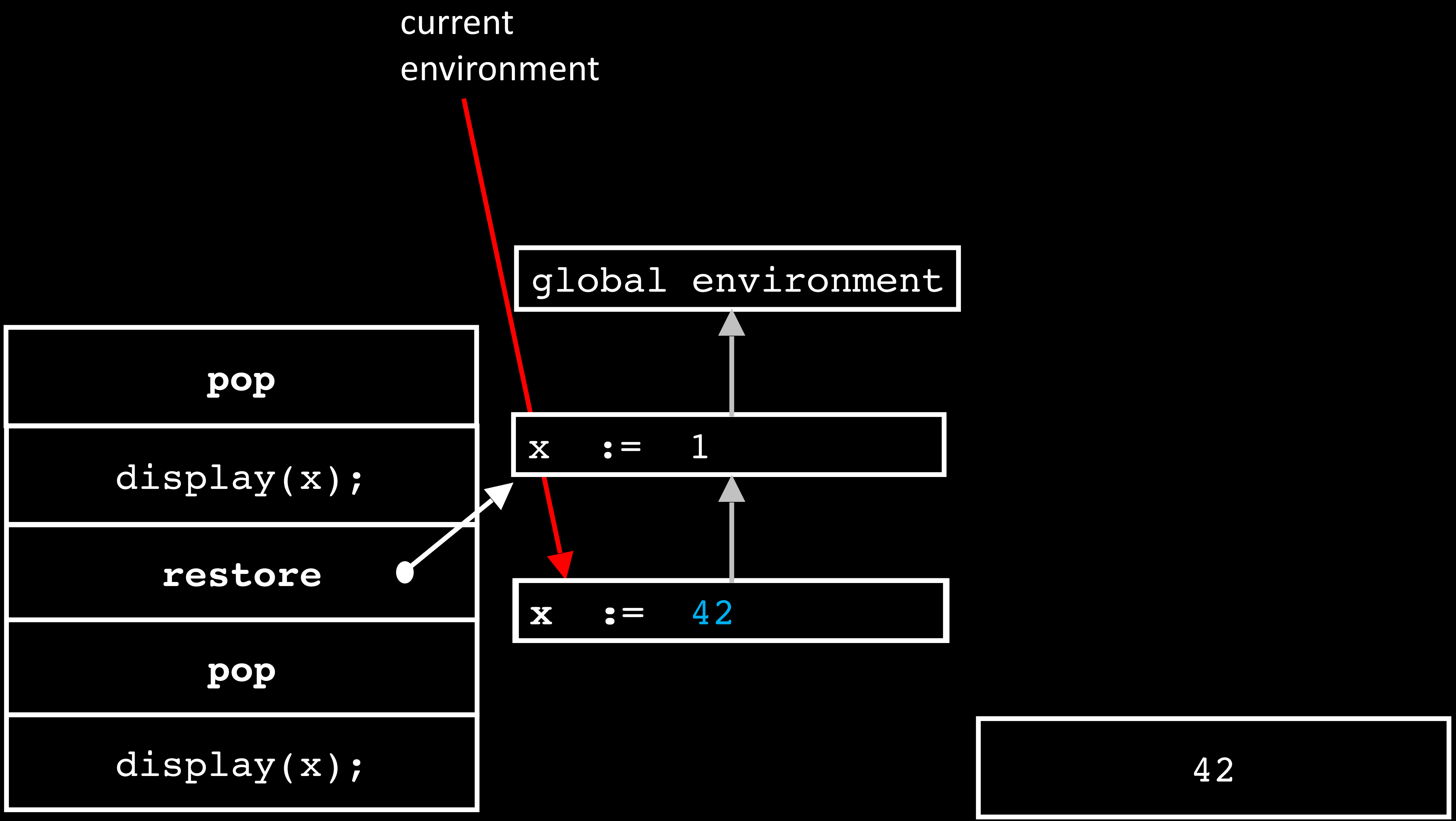

current environment
const x = 42;
pop
display(x);
restore
pop
display(x);
global environment
x := 1
x :=

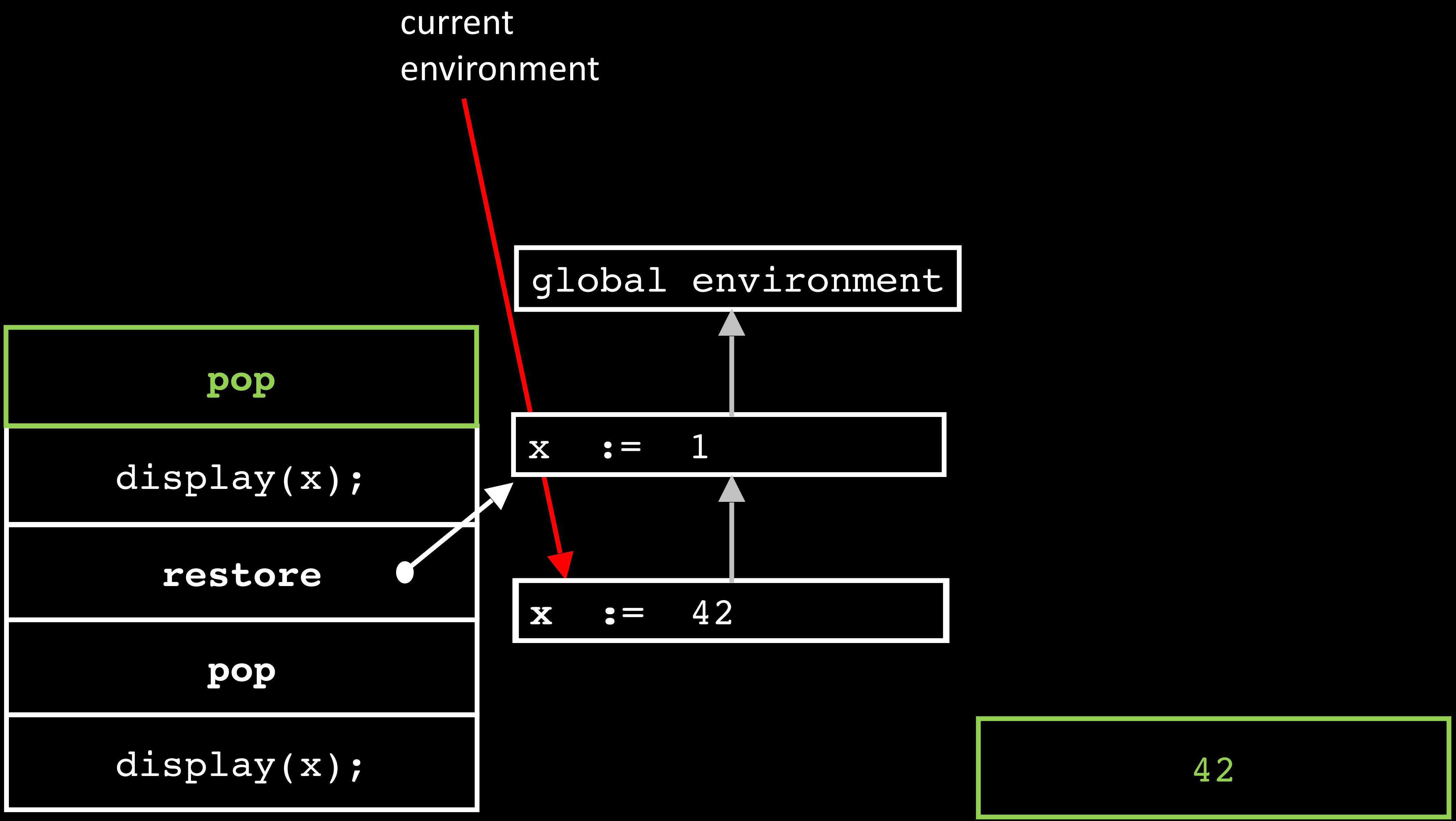

current environment
42
assign x
pop
display(x);
restore
pop
display(x);
global environment
x := 1
x :=

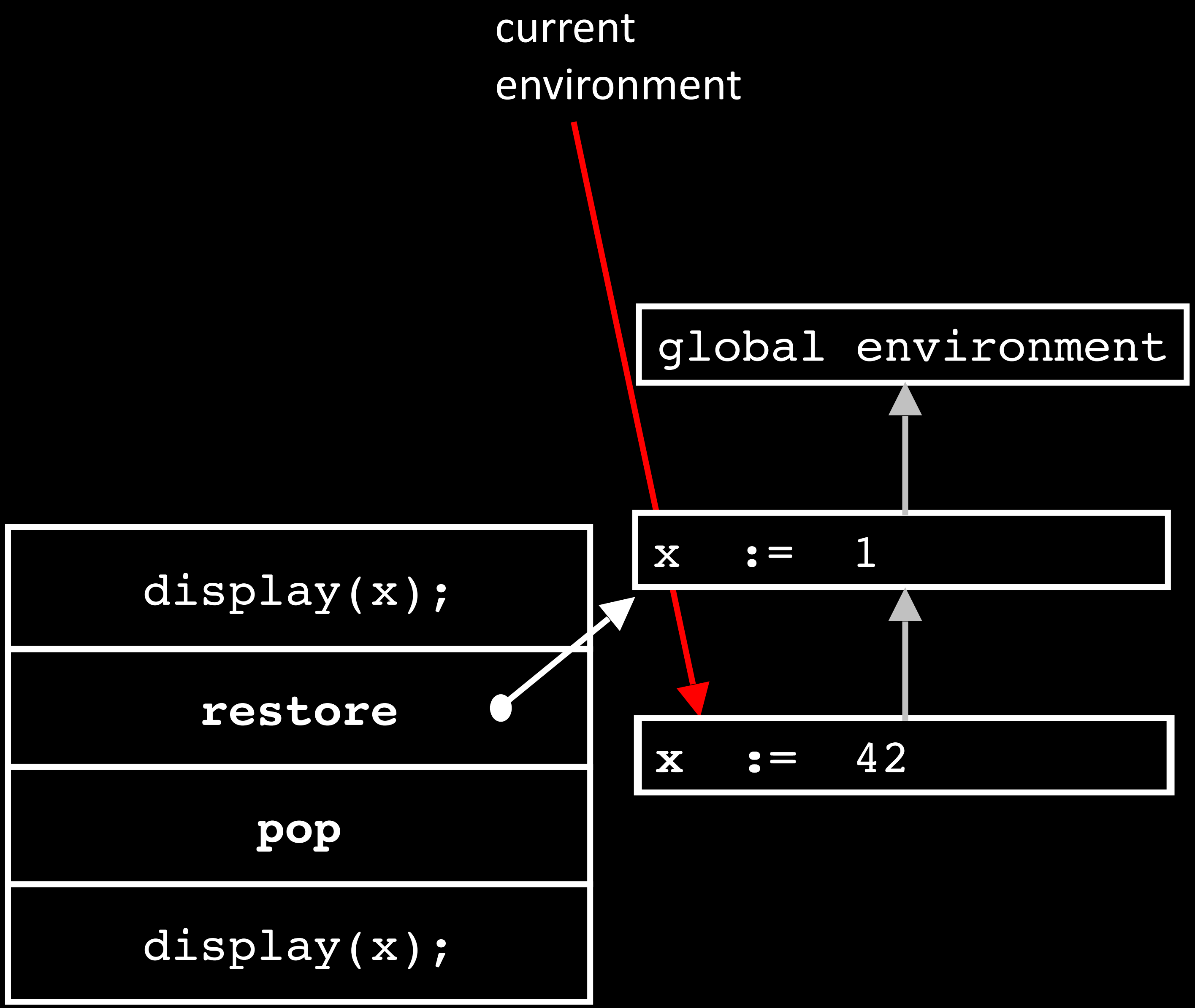

current
environment
42
assign x
pop
display(x);
restore
pop
display(x);
global environment
x := 1
x :=

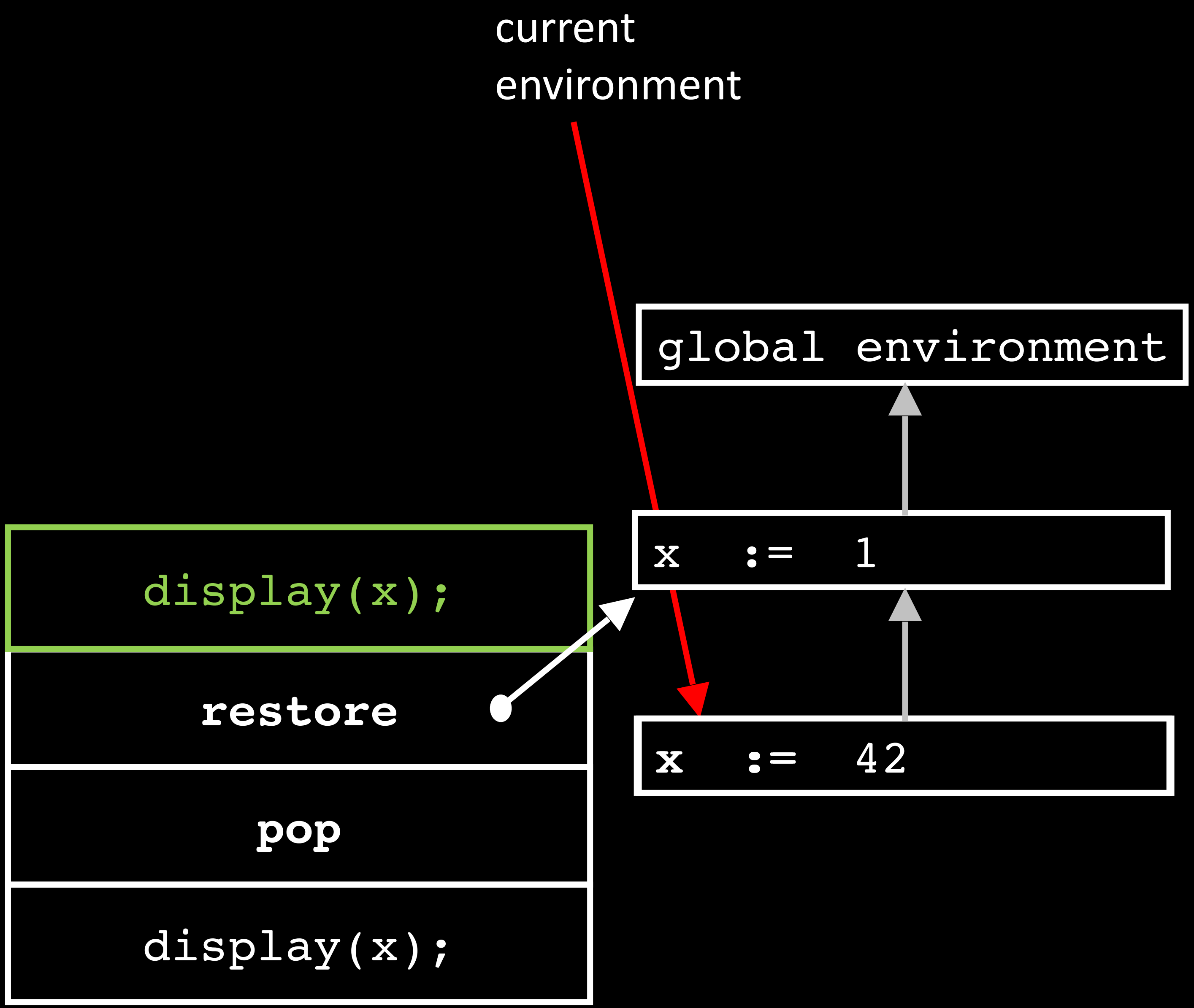

current environment
assign x
pop
display(x);
restore
pop
display(x);
global environment
x := 1
x :=
42

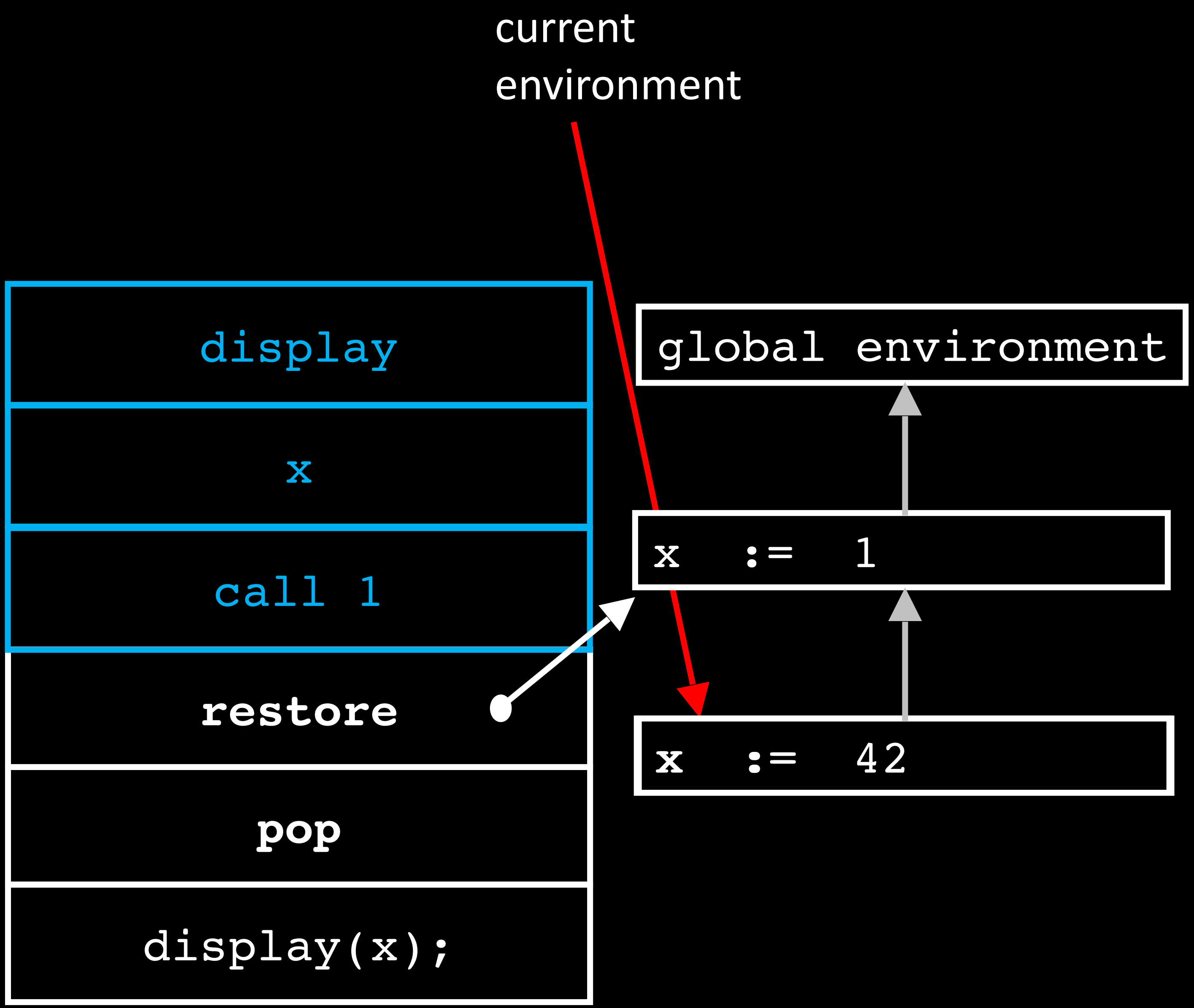

current environment
assign x
pop
display(x);
restore
pop
display(x);
global environment
x := 1
x :=
42

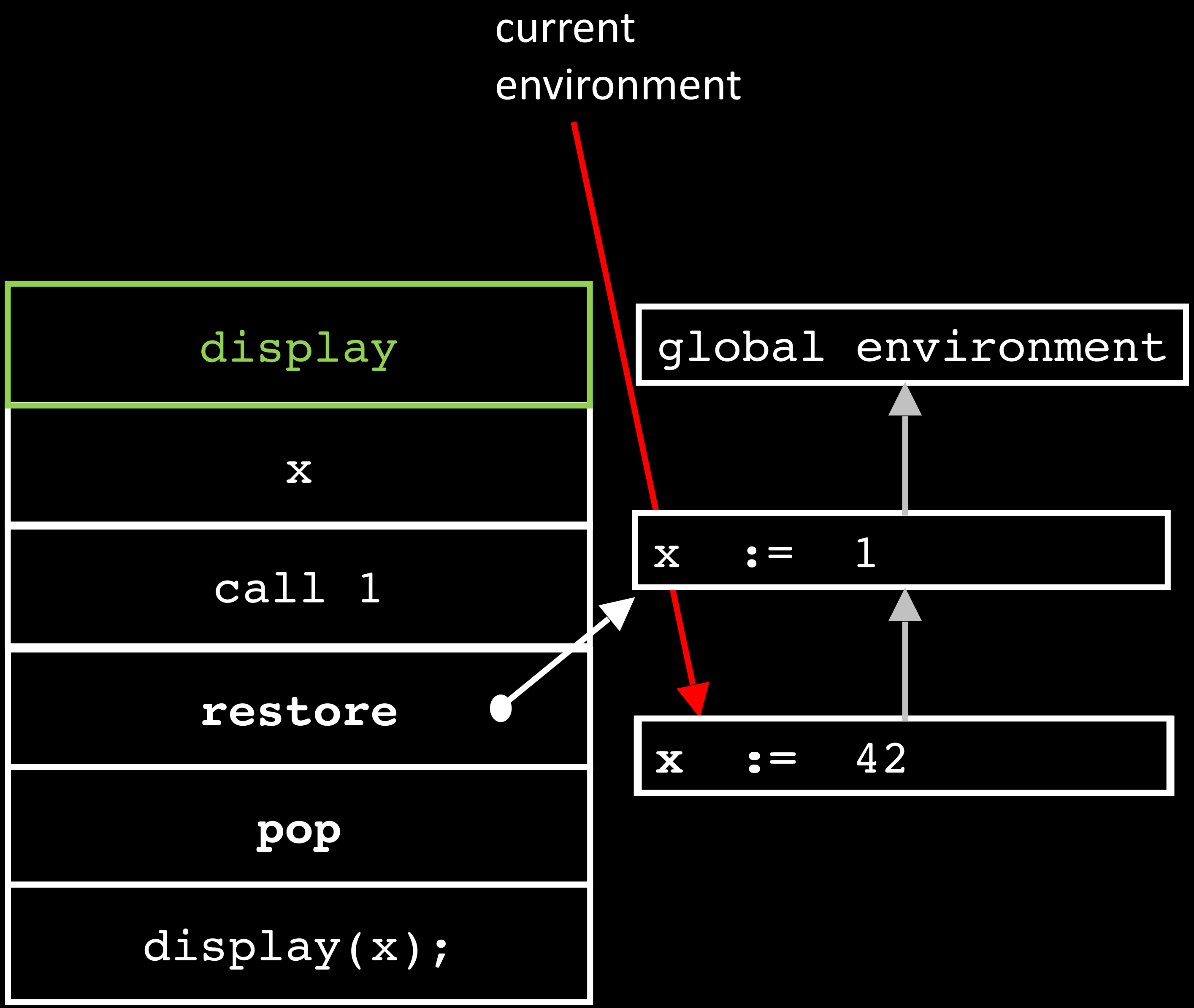

current
environment
global environment
pop
display(x);
restore
pop
display(x);
x := 1
x := 42
42

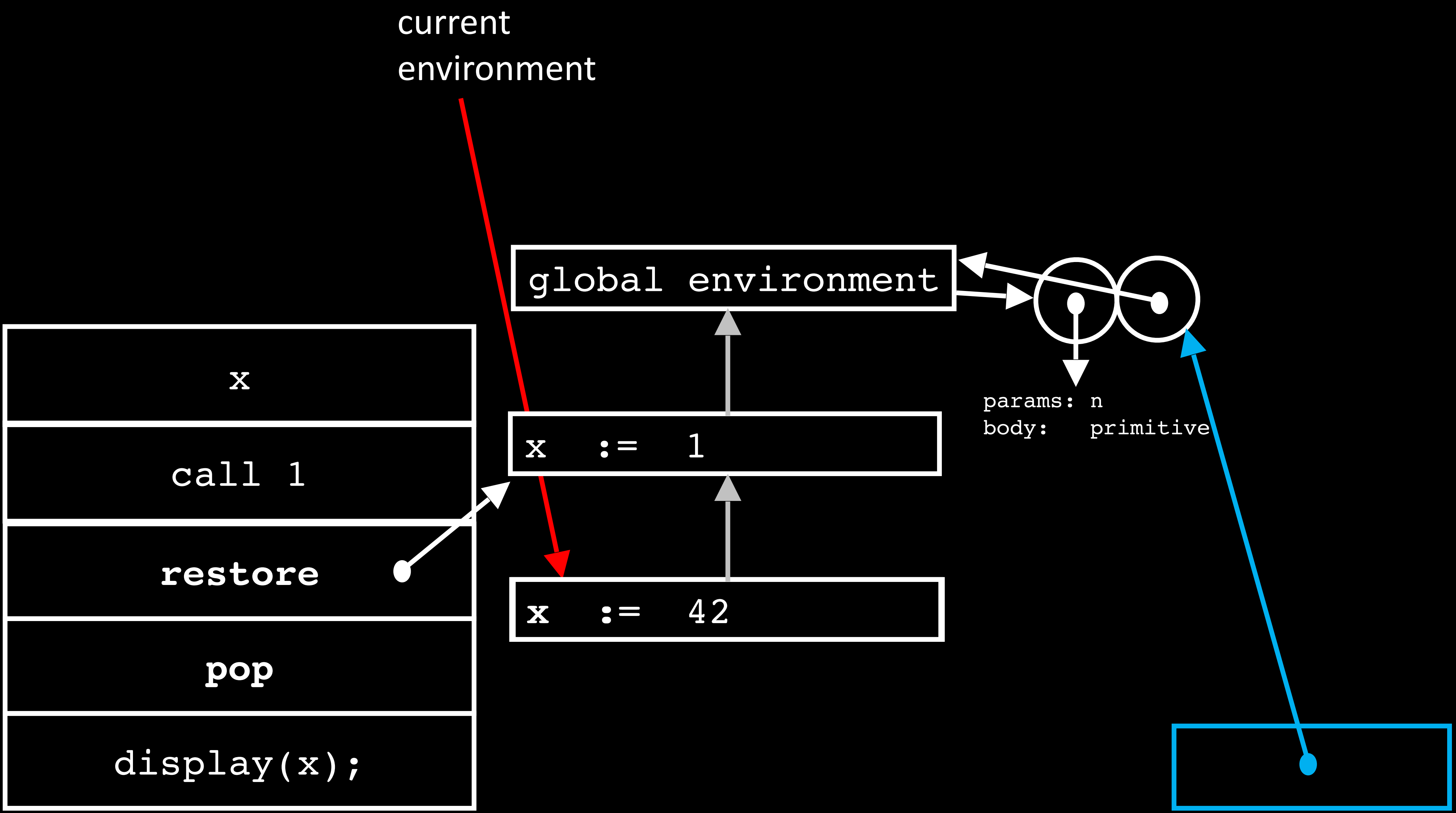

current environment
global environment
x := 1
x := 42
pop
display(x);
restore
pop
display(x);
42


current
environment
global environment
x := 1
x := 42
display(x);
restore
pop
display(x);

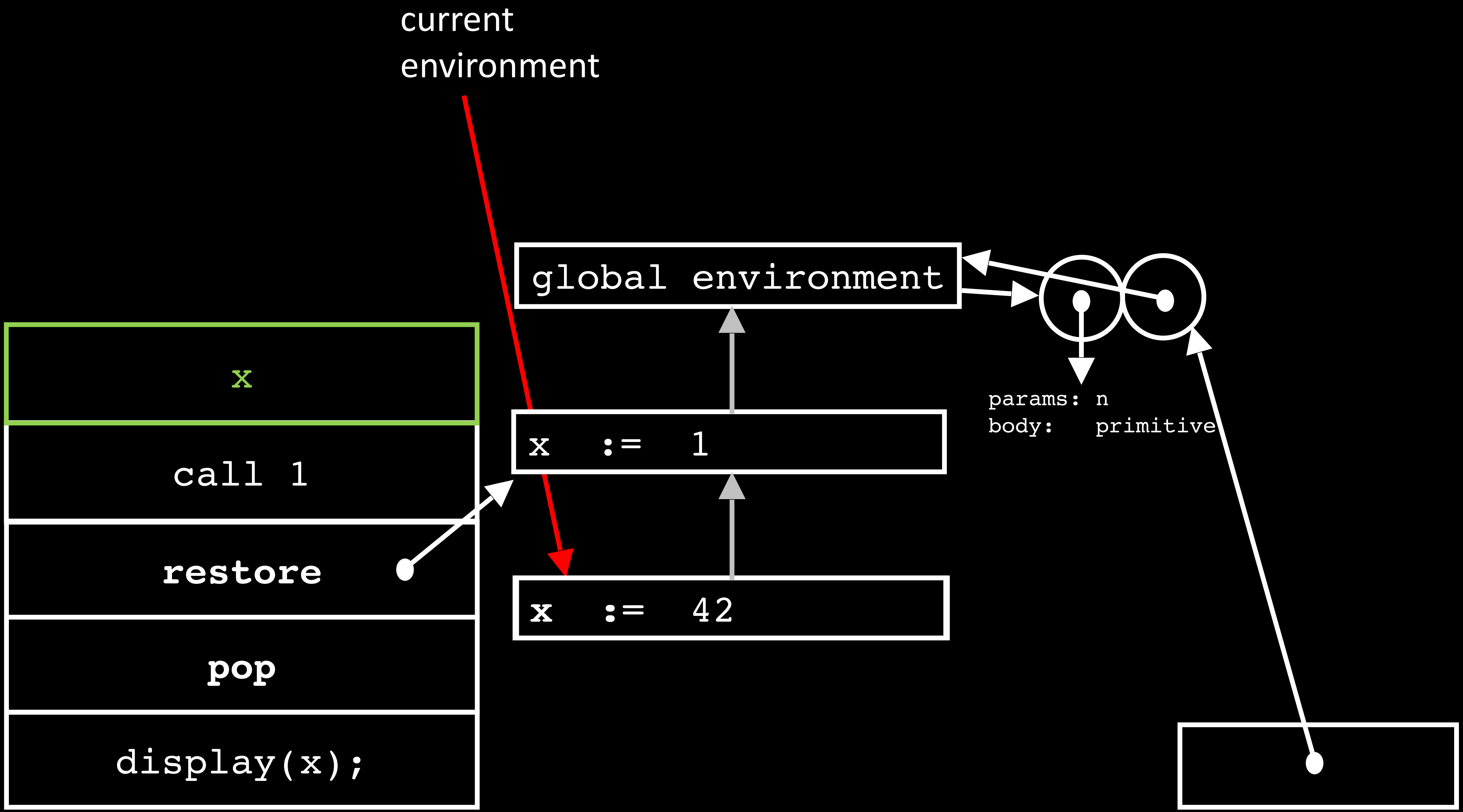

current environment
global environment
x := 1
x := 42
display(x);
restore
pop
display(x);

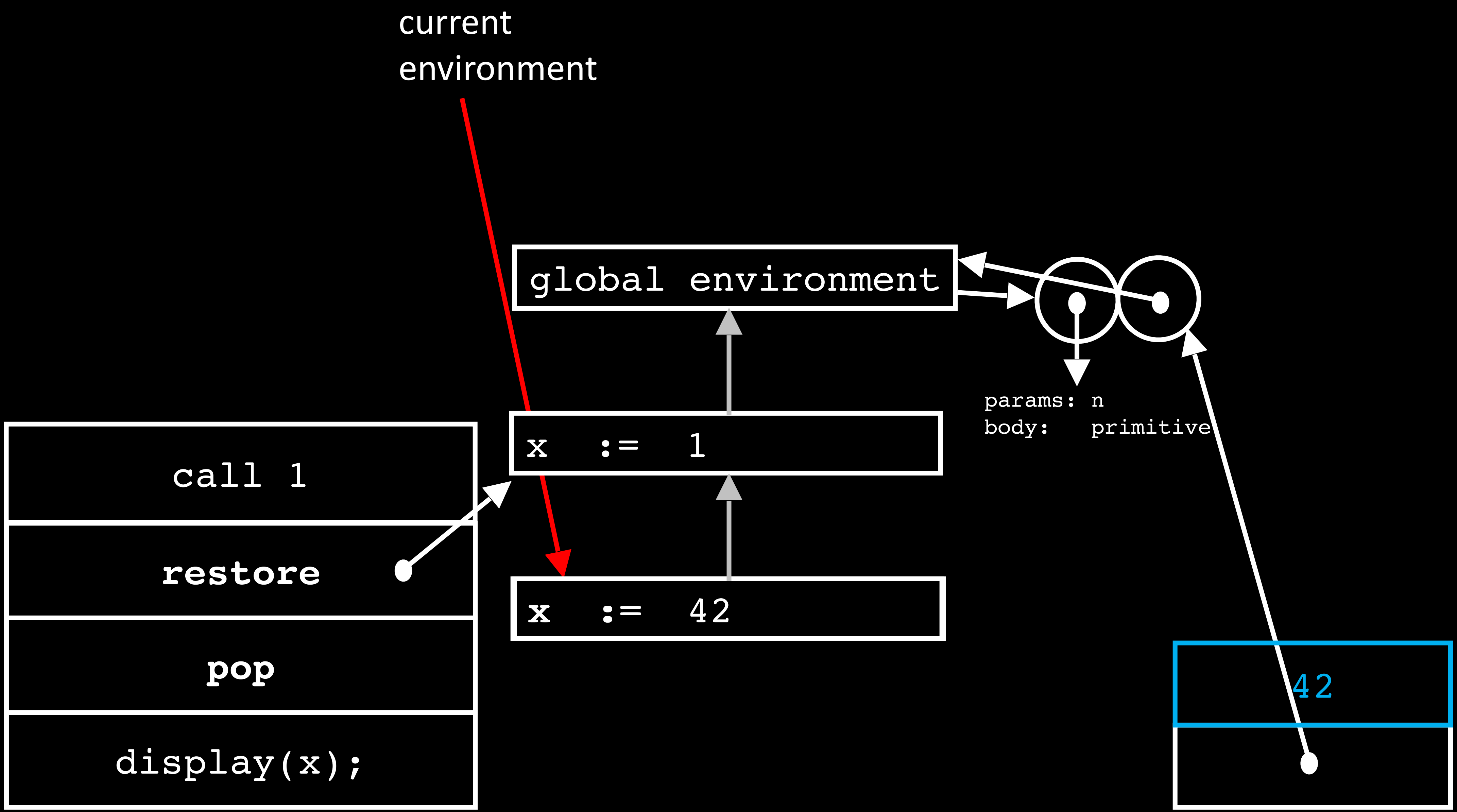

current environment
display
x
call 1
restore
pop
display(x);
global environment
x := 1
x := 42

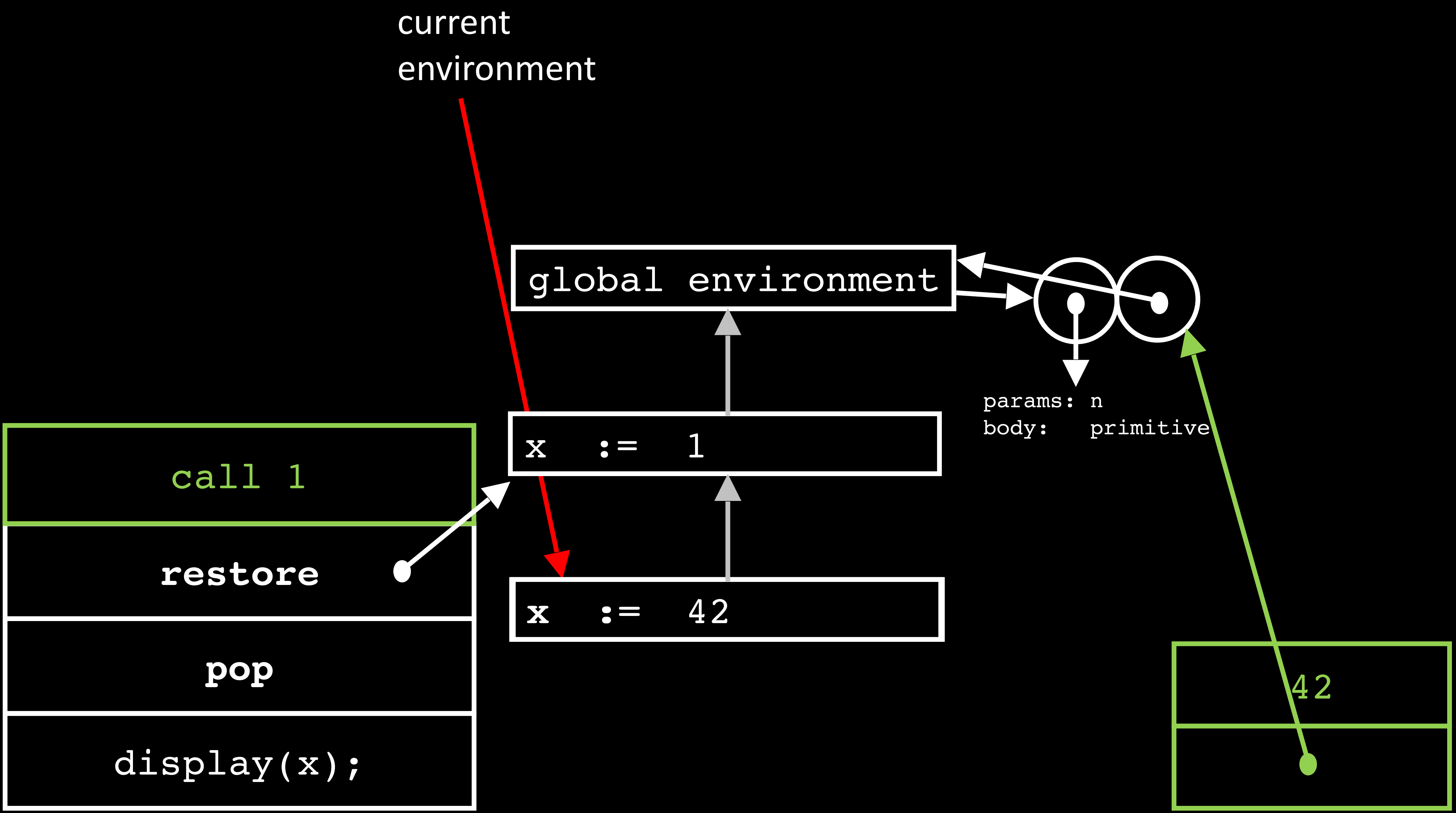

current environment
display
x
call 1
restore
pop
display(x);
global environment
x := 1
x := 42

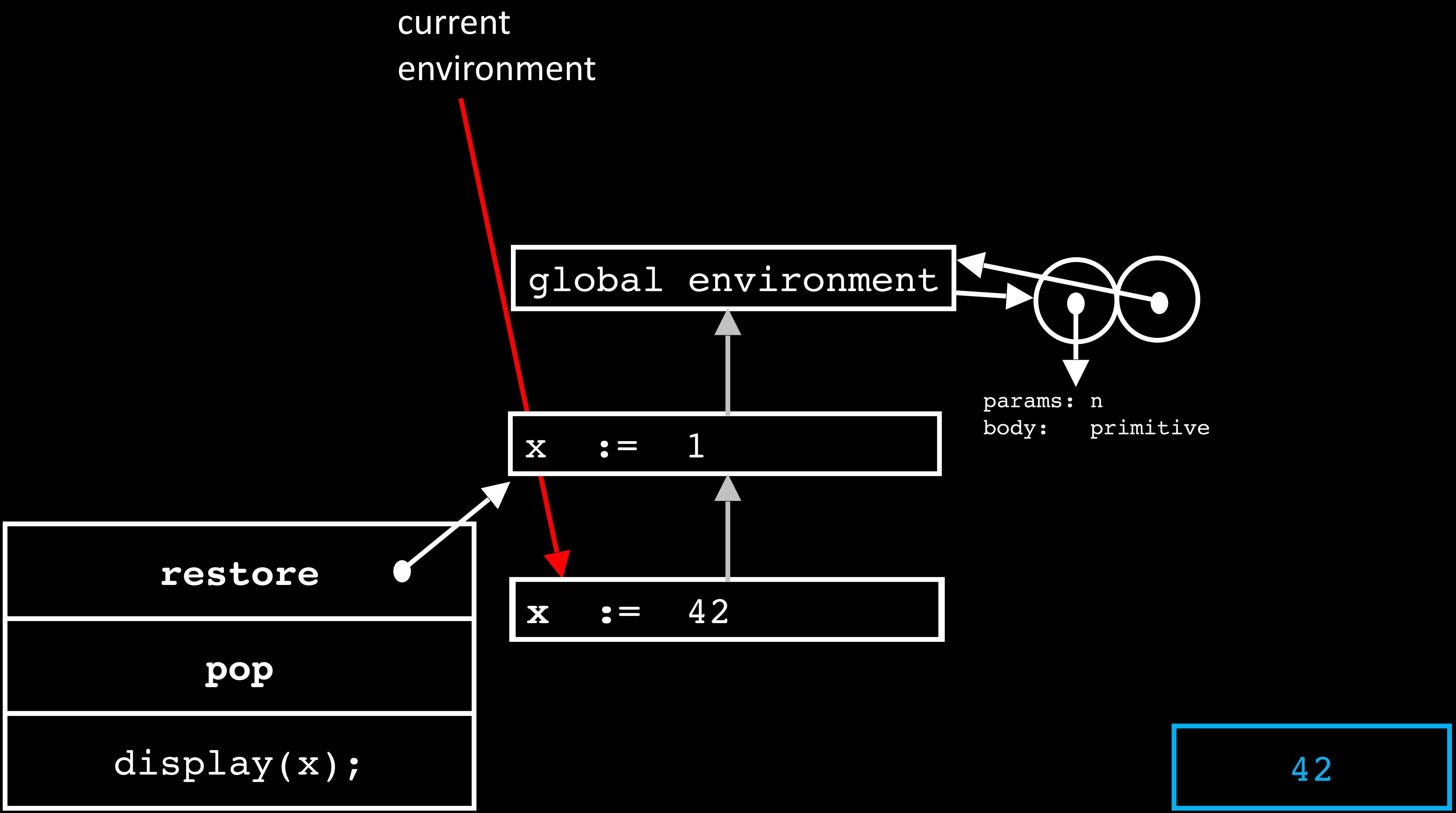


current environment
global environment
x
call 1
restore
pop
display(x);
x := 1
x := 42
params: n
body: primitive

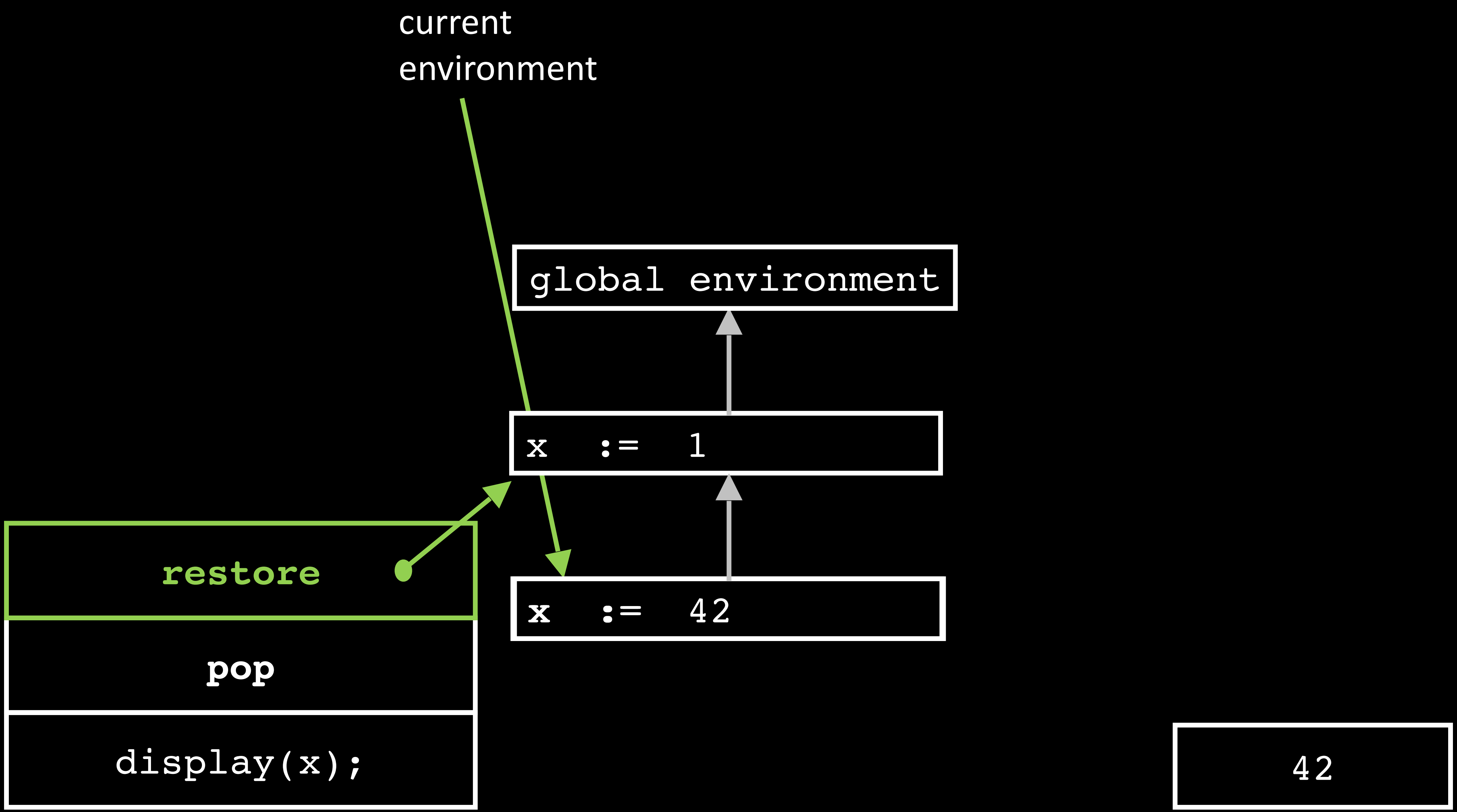

current
environment
global environment
x := 1
x := 42
params: n
body: primitive
x
call 1
restore
pop
display(x);

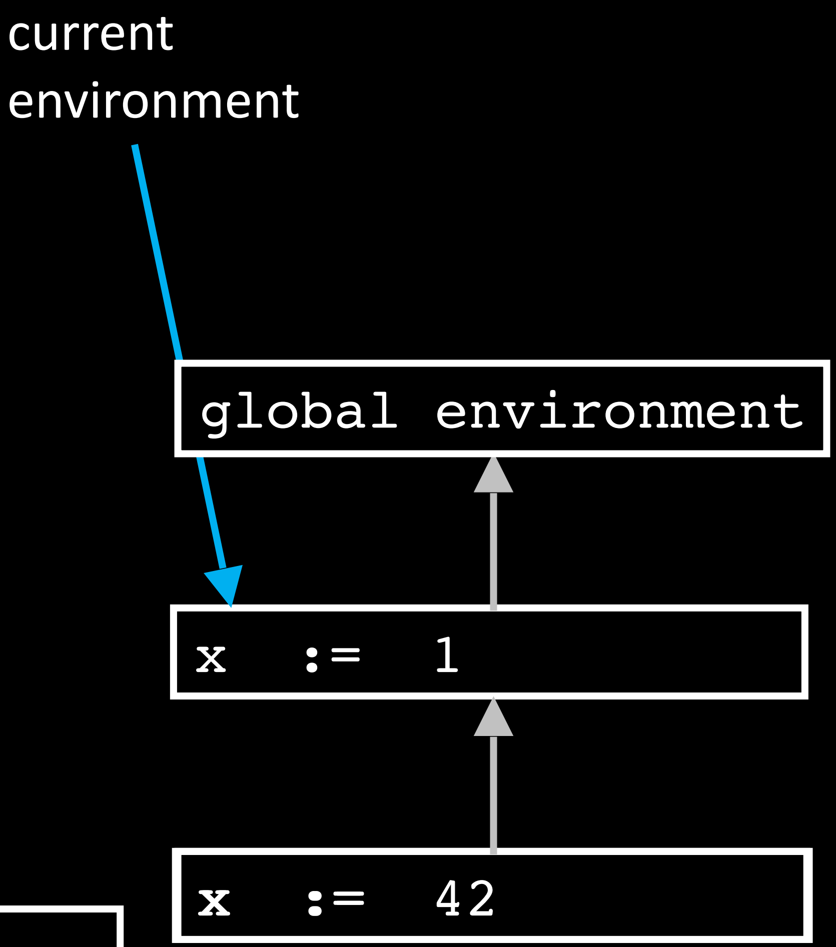

current environment
global environment
x := 1
x := 42
params: n
body: primitive
call 1
restore
pop
display(x);
42


current environment
global environment
params: n
body: primitive
x := 1
x := 42
call 1
restore
pop
display(x);
42



current environment
global environment
params: n
body: primitive
x := 1
x := 42
restore
pop
display(x);
42

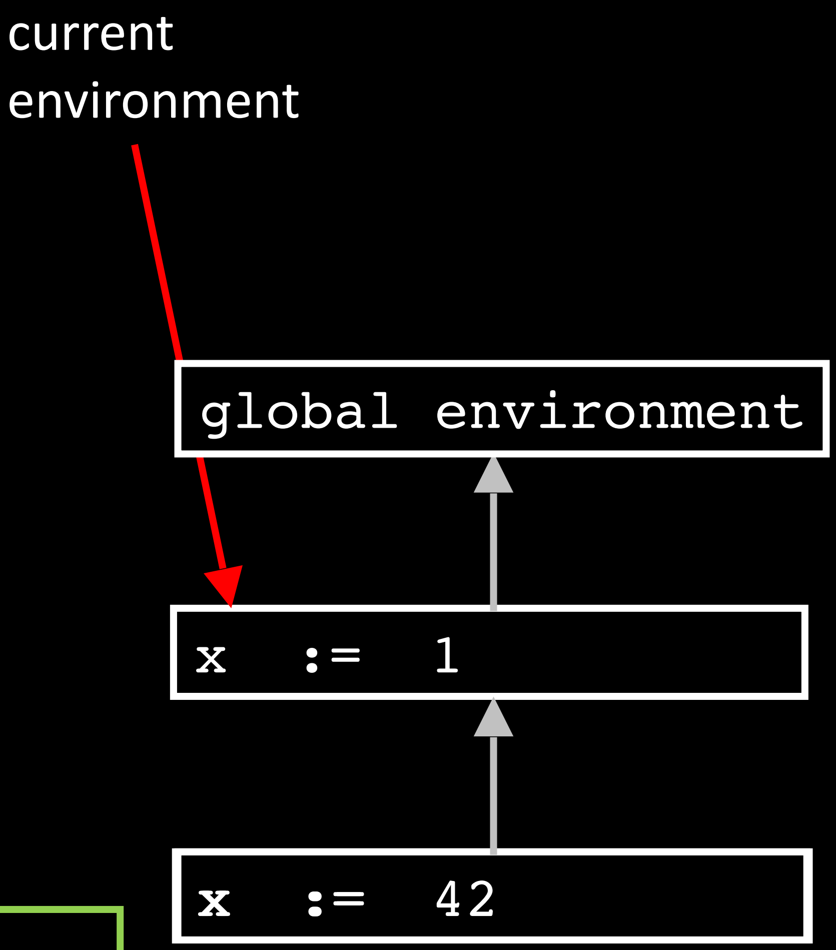

current environment
global environment
x := 1
x := 42
restore
pop
display(x);
42



pop

```
display(x);
```

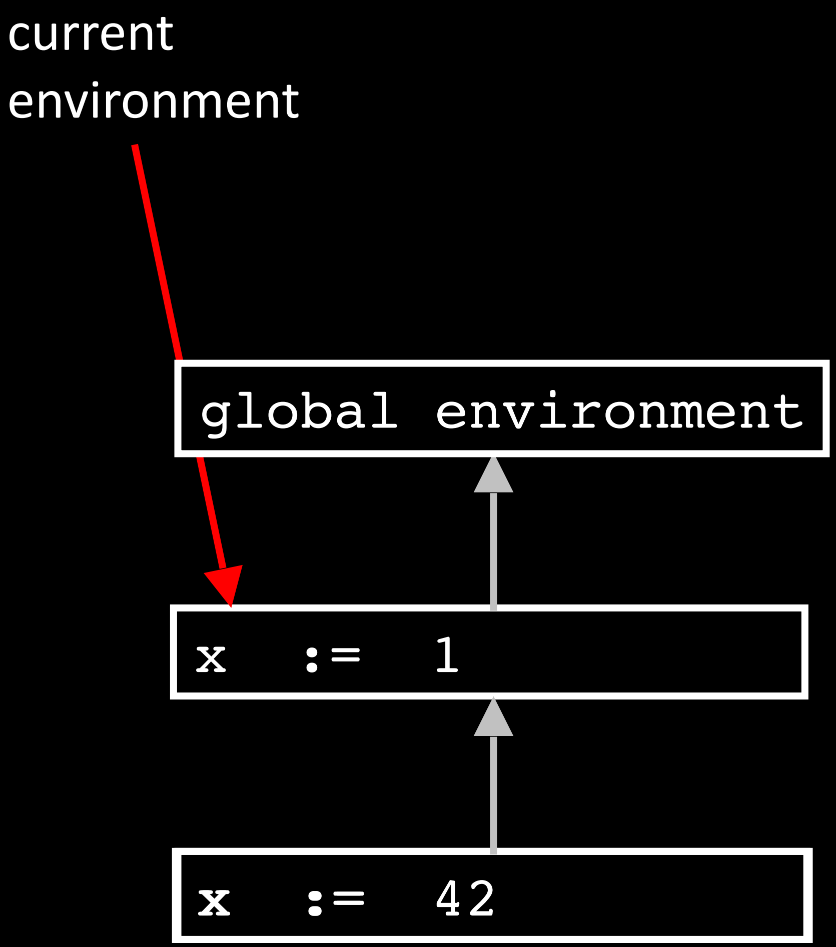
```
42
```



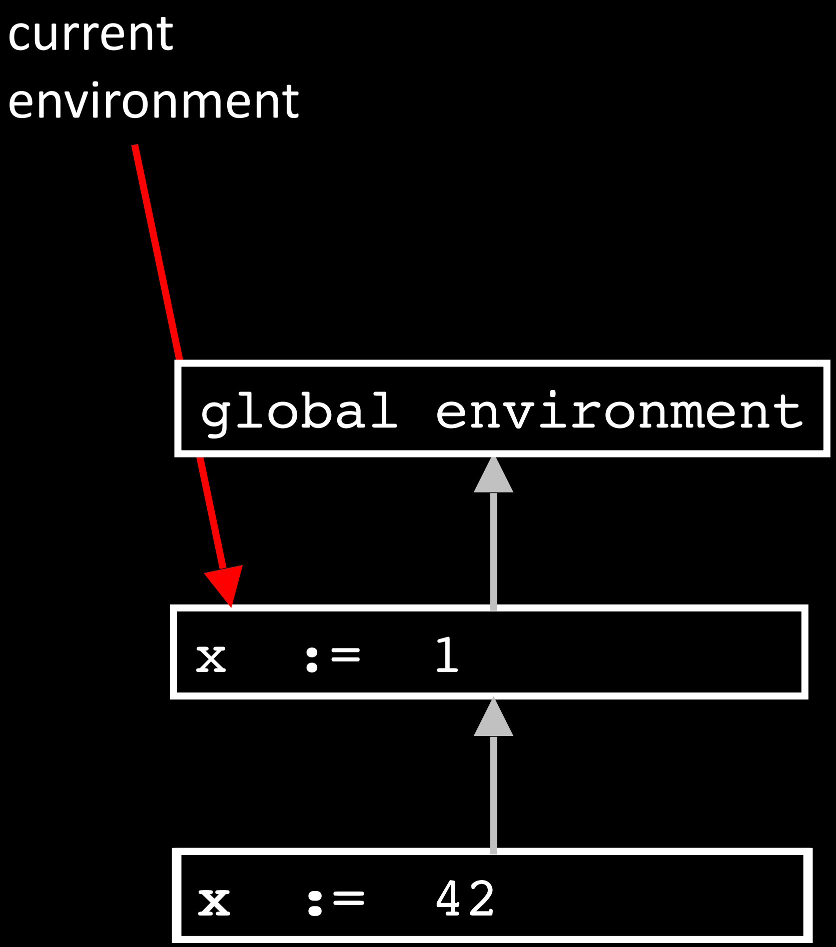
pop

```
display(x);
```

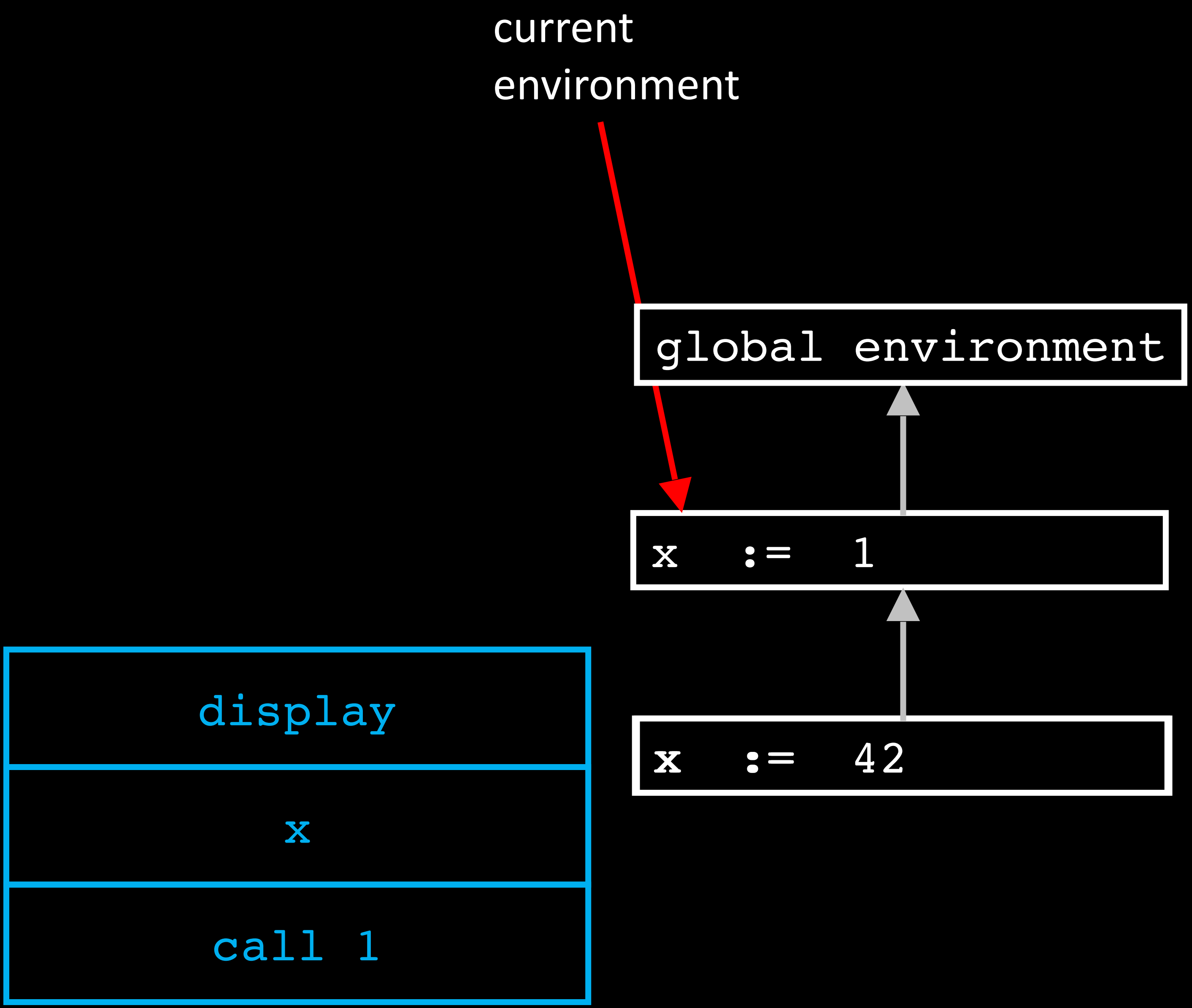
```
42
```

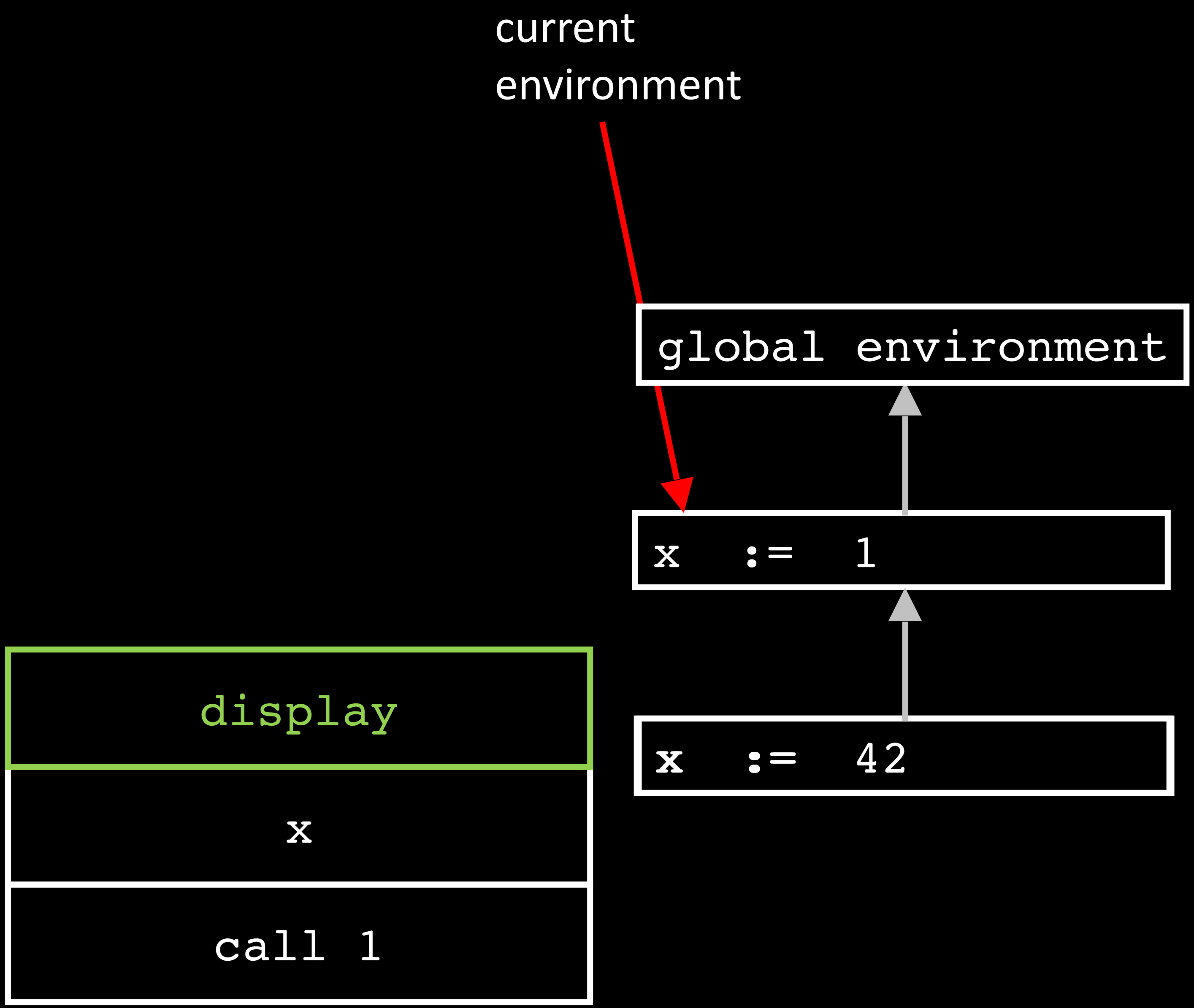


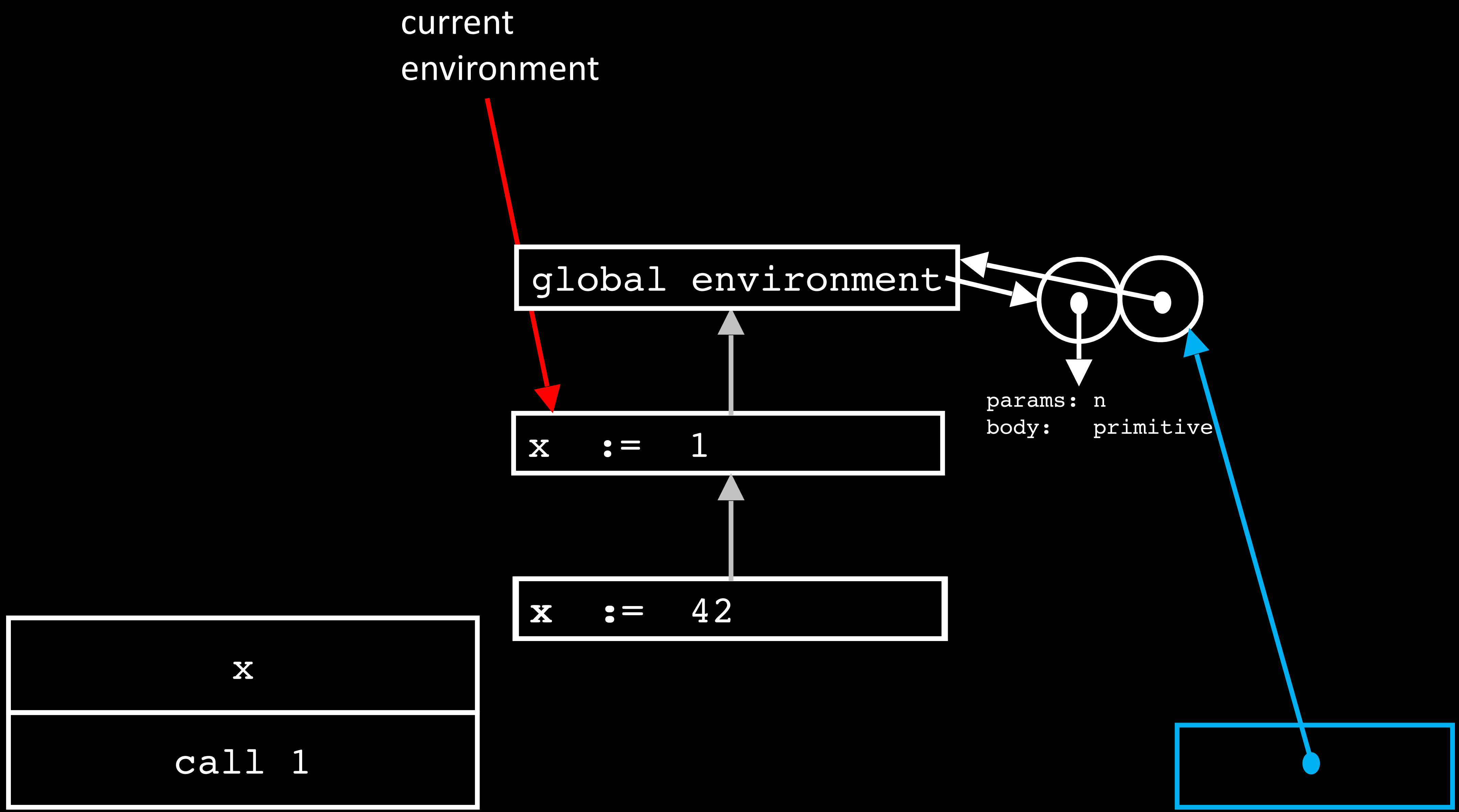
```
display(x);
```

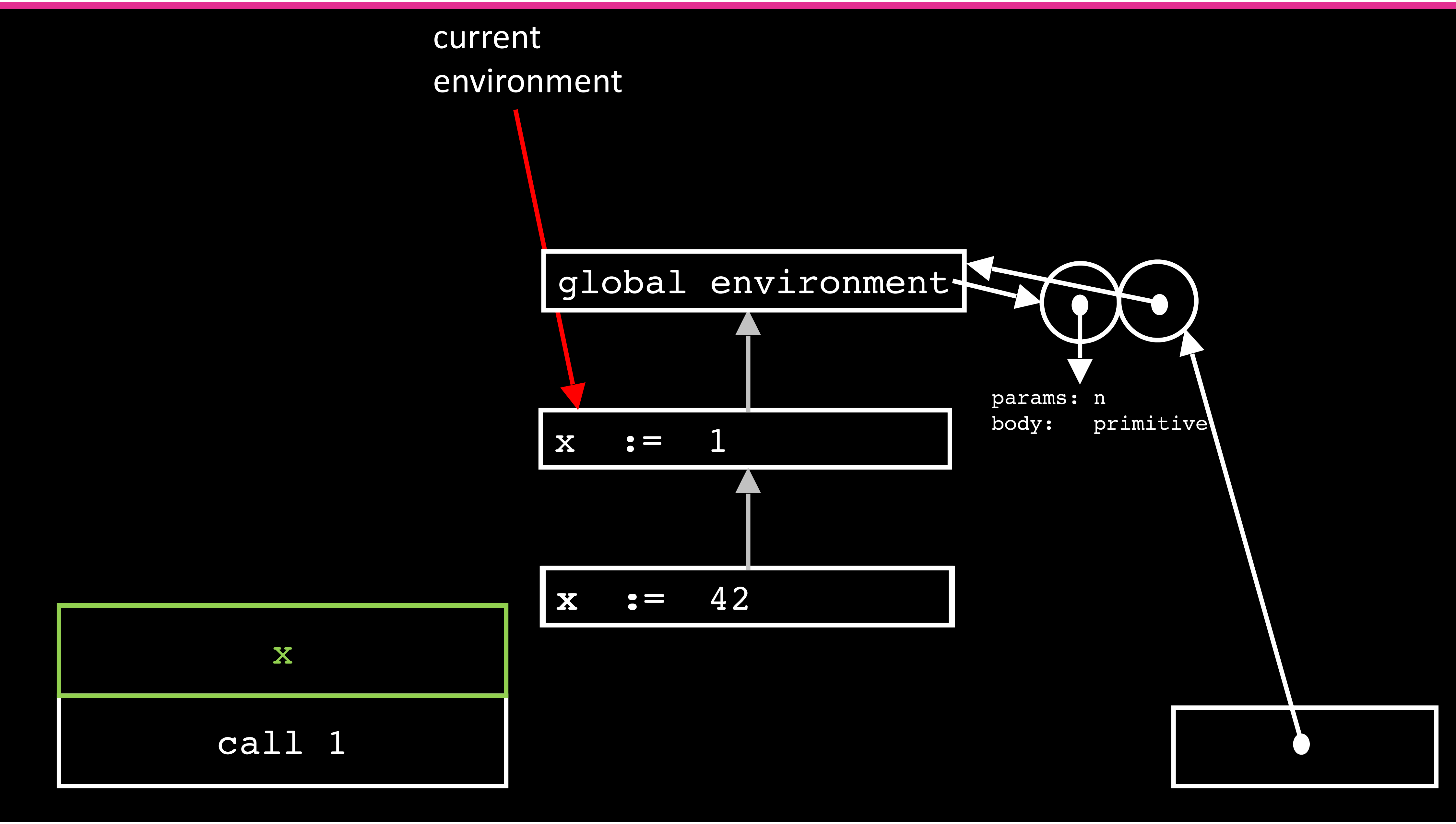


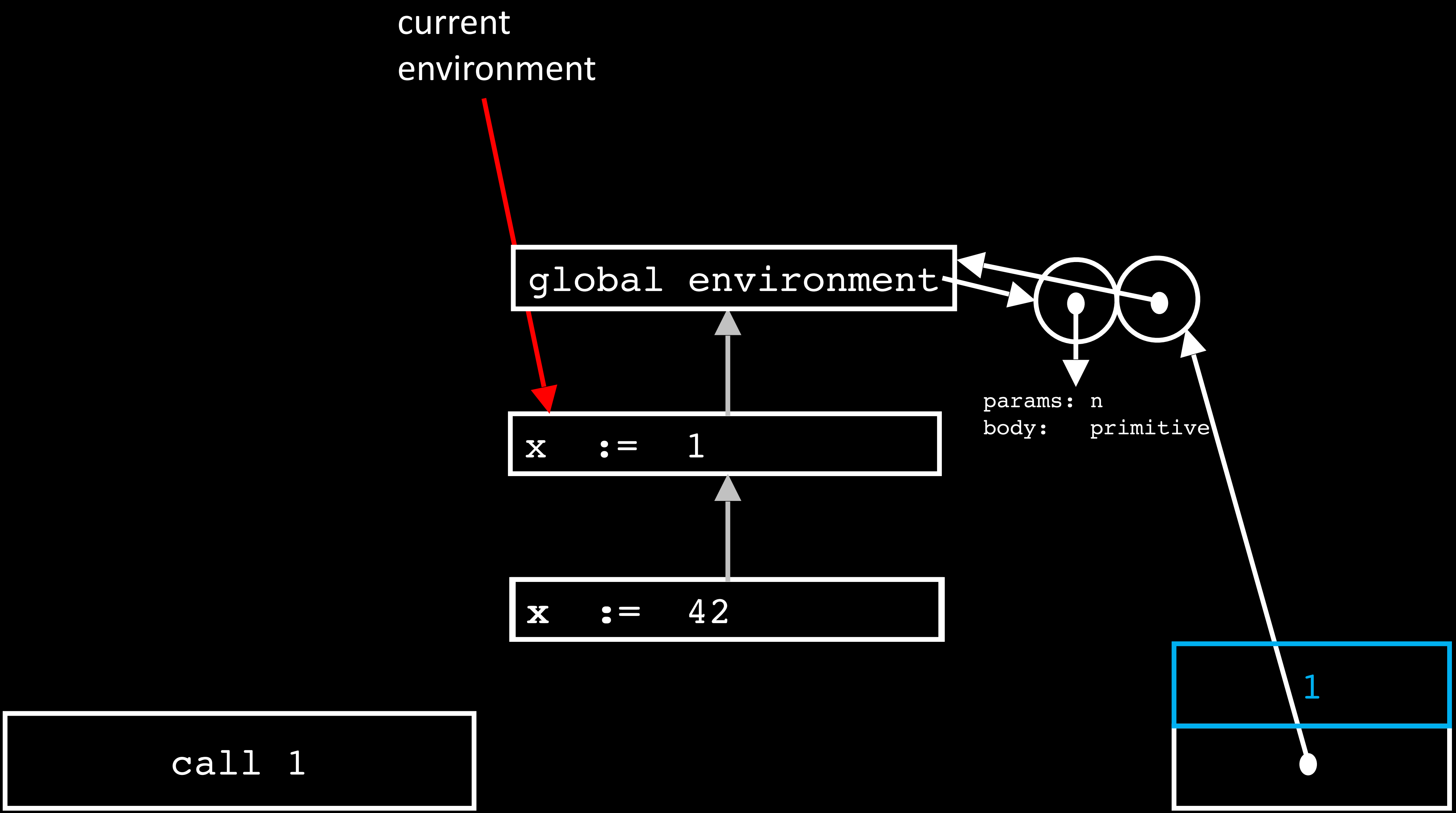
```
display(x);
```

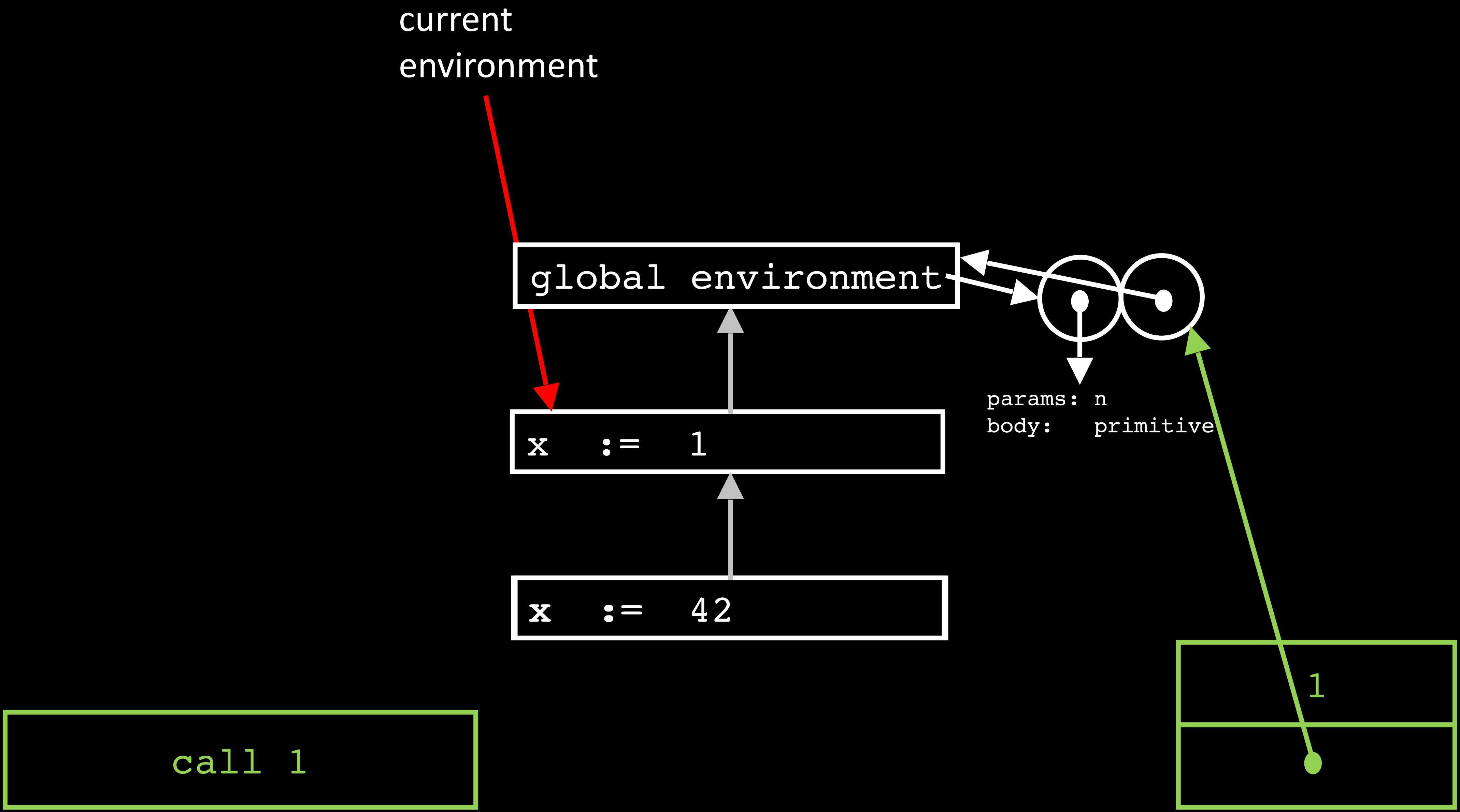

current environment
global environment
x := 1
x := 42
display
x
call 1

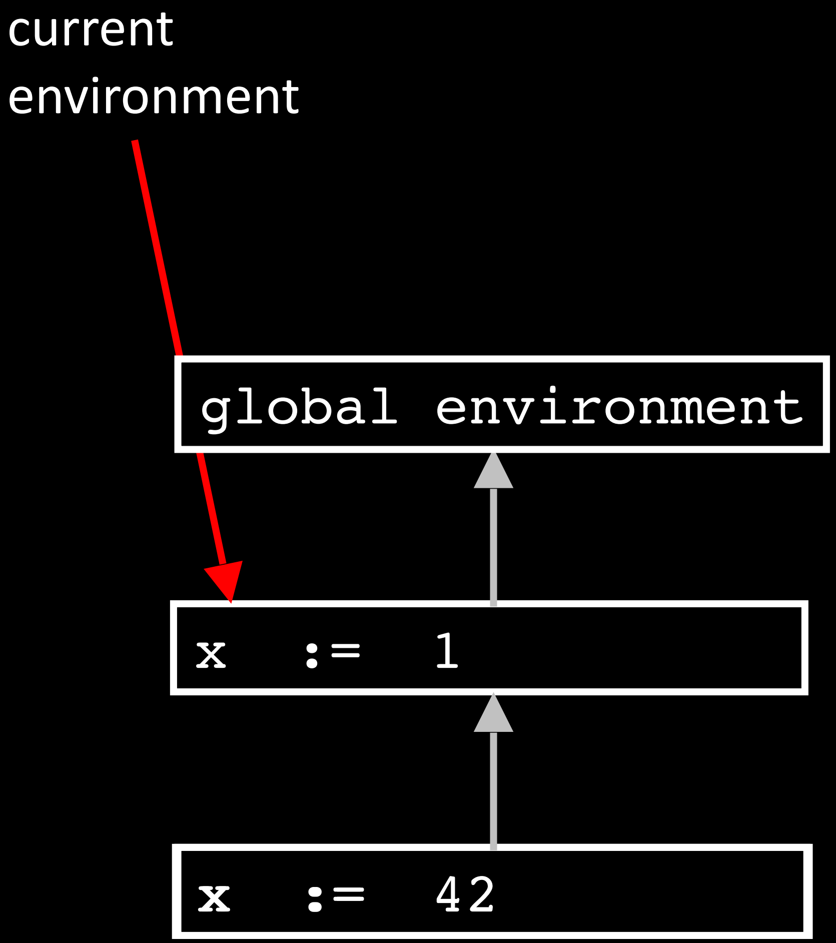

current environment
global environment
x := 1
x := 42
display
x
call 1


current
environment
global environment
x := 1
x := 42
params: n
body: primitive
x
call 1


current environment
global environment
params: n
body: primitive
x := 1
x := 42
x
call 1

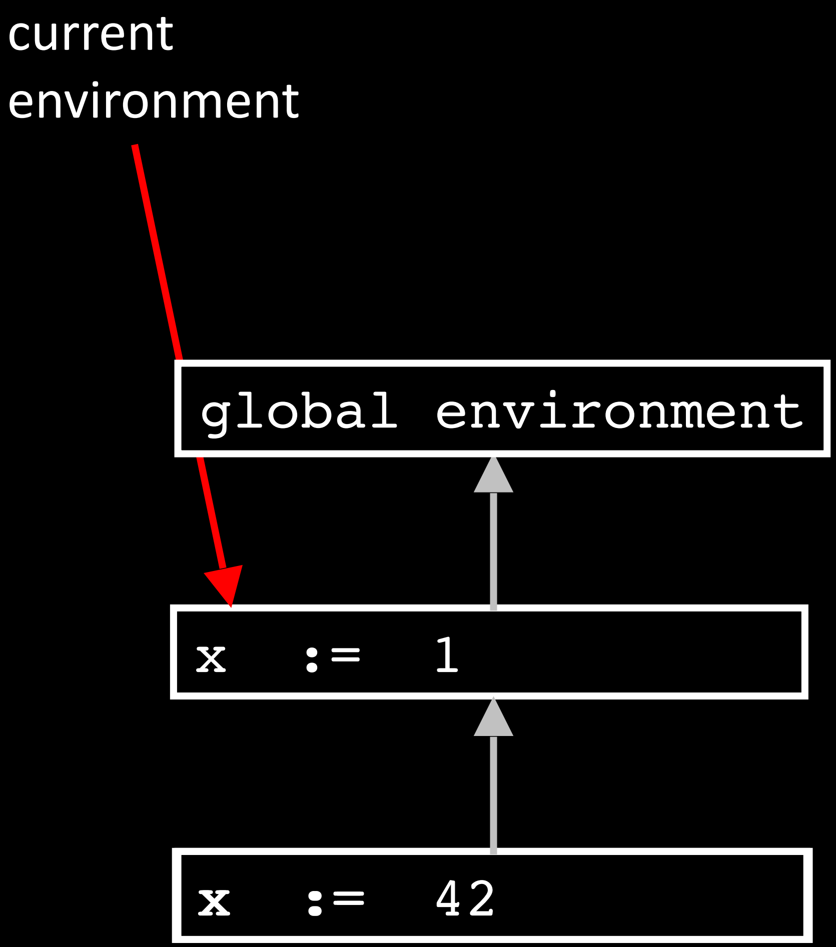

current
environment
global environment
params: n
body: primitive
x := 1
x := 42
1
call 1


current
environment
global environment
params: n
body: primitive
x := 1
x := 42
call 1
1



1

Done



1

Recall: The need for preserving environments (2)

```
const n = 42;

function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}

fact(4) + n;
```

Recall: The need for preserving environments (2)

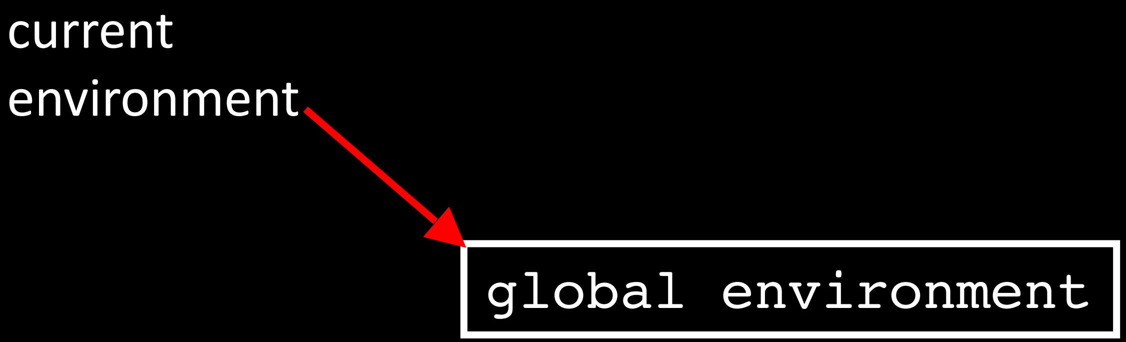
```
const n = 42;

function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}

fact(4) + n;
```

- After returning from the recursive call, we need the previous **n**.

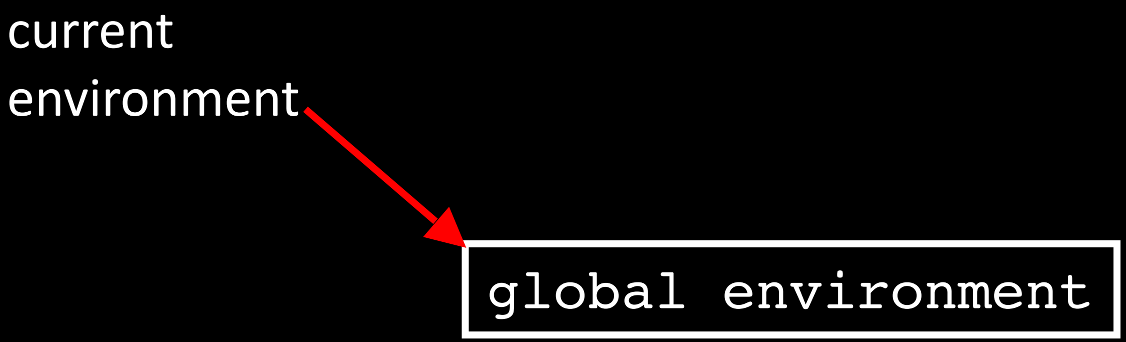
Recall: The need for preserving environments (2)

```
const n = 42;

function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}

fact(4) + n;
```

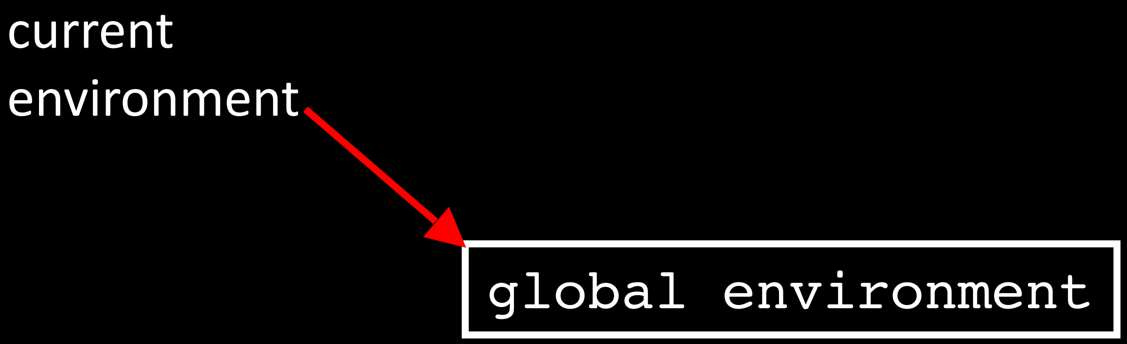
- After returning from the recursive call, we need the previous **n**.
- After fact(4), we need the **n** of the program environment.



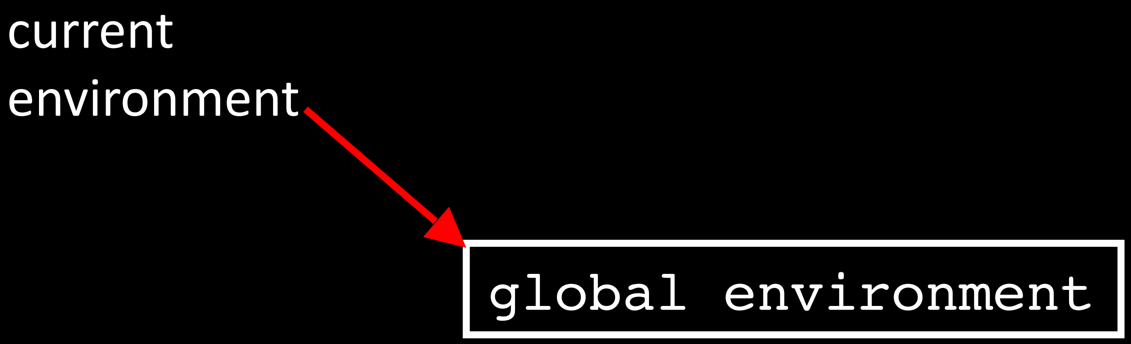
```
const n = 42;
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
fact(4) + n;
```



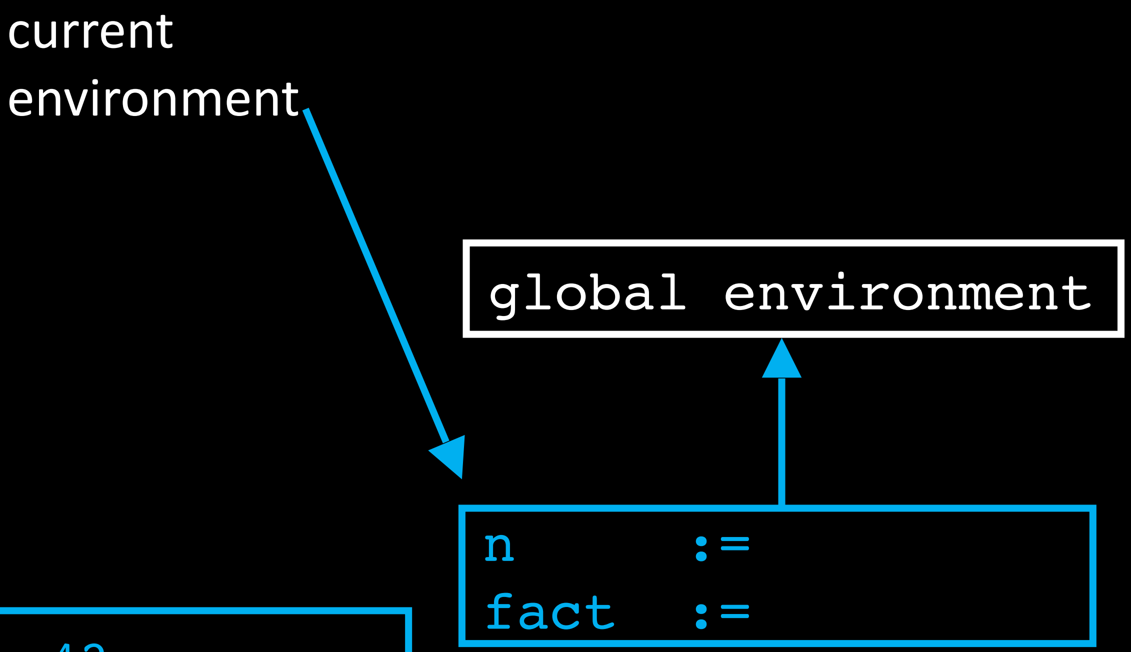
```
const n = 42;
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
fact(4) + n;
```



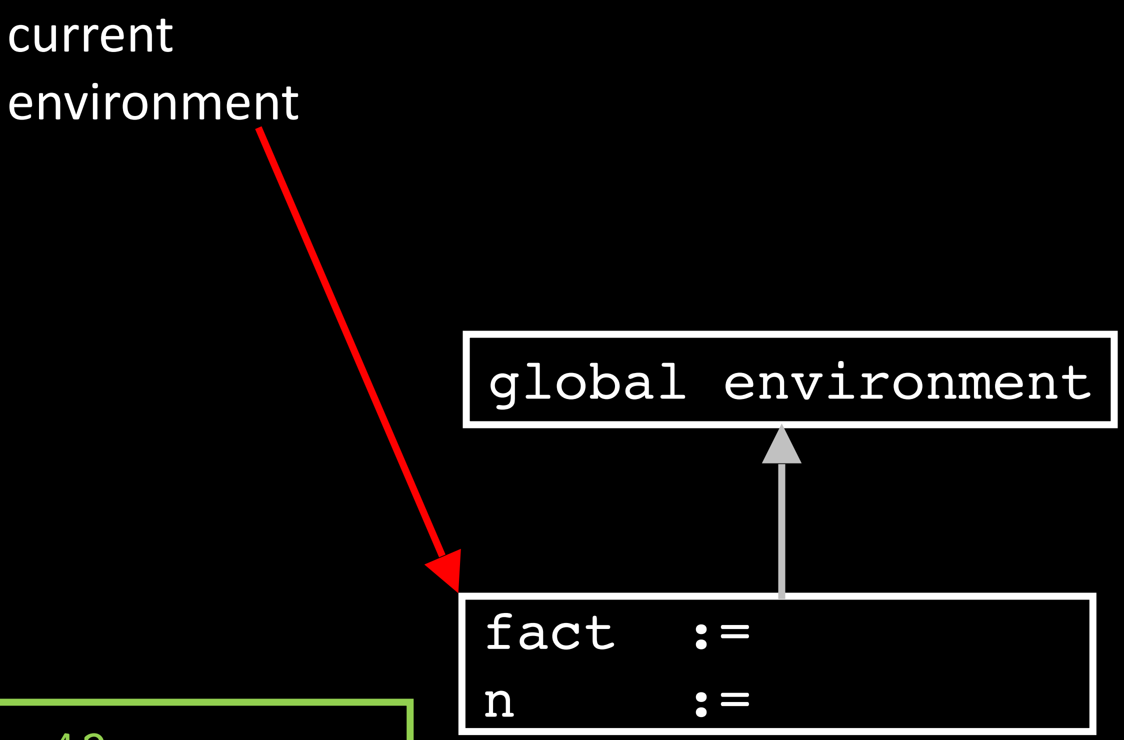
```
{
 const n = 42;
 function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
 }
 fact(4) + n;
}
```



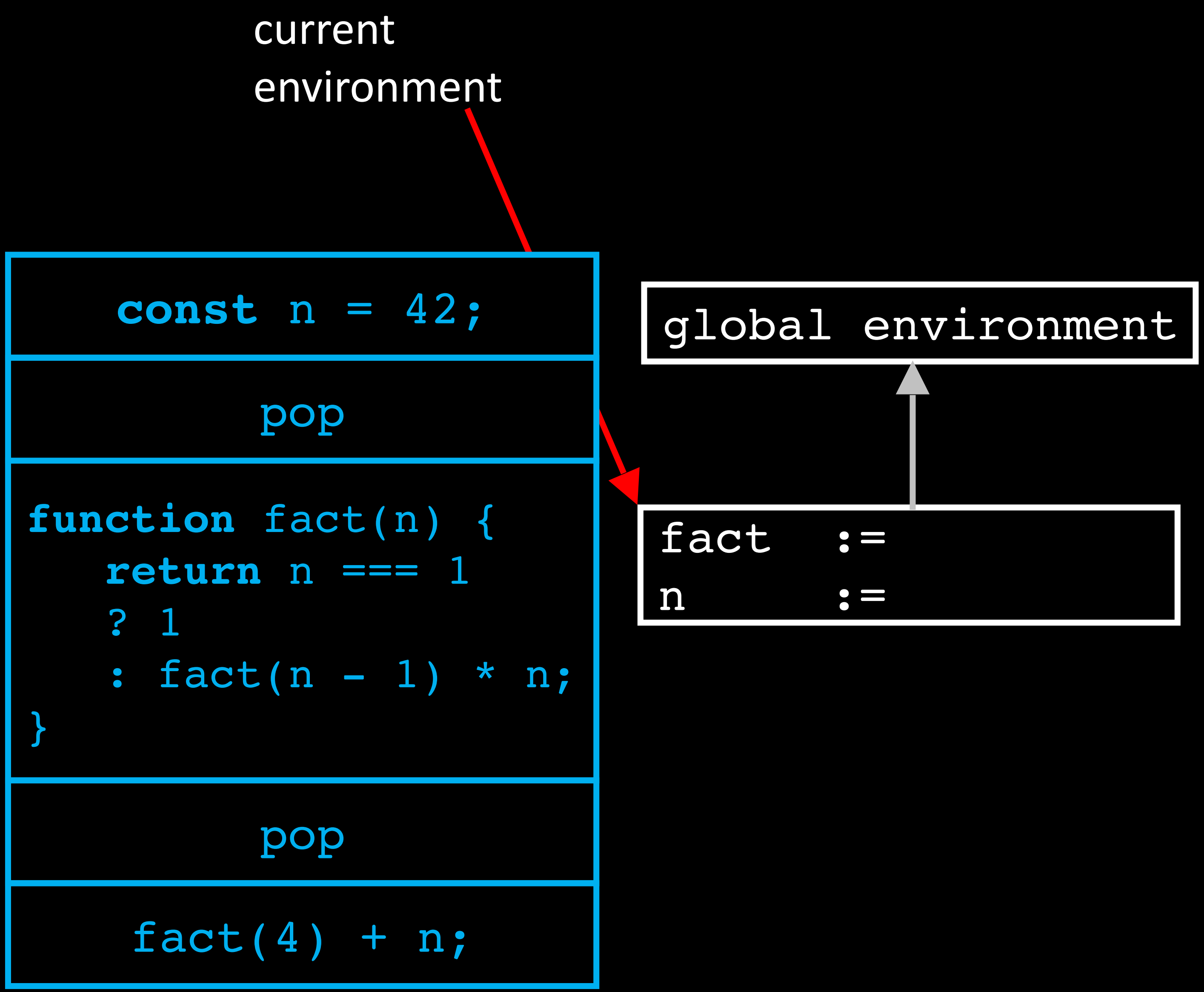
```
{
 const n = 42;
 function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
 }
 fact(4) + n;
}
```



```
const n = 42;
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
fact(4) + n;
```



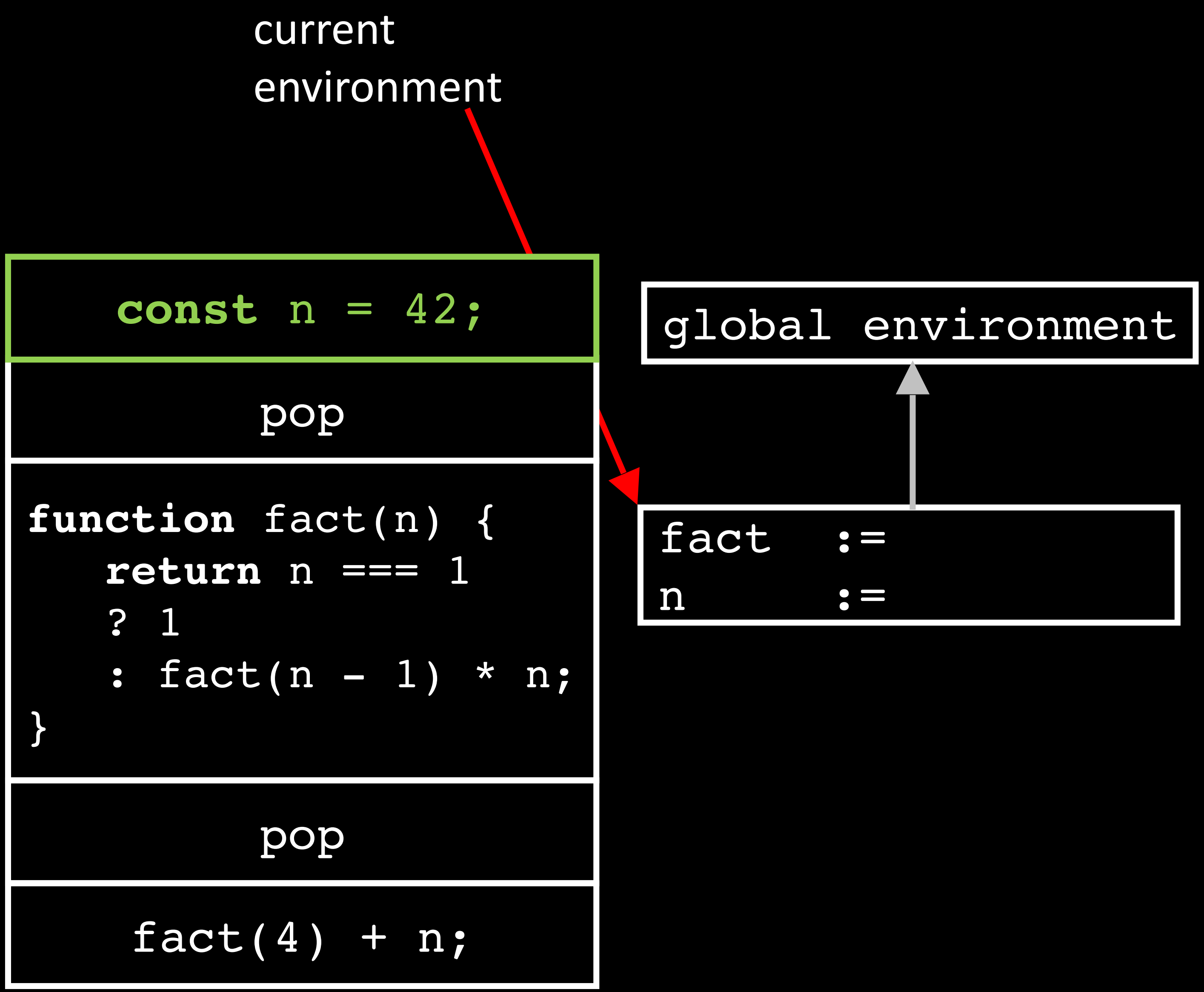
```
const n = 42;
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
fact(4) + n;
```



```
const n = 42;
```

```
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
```

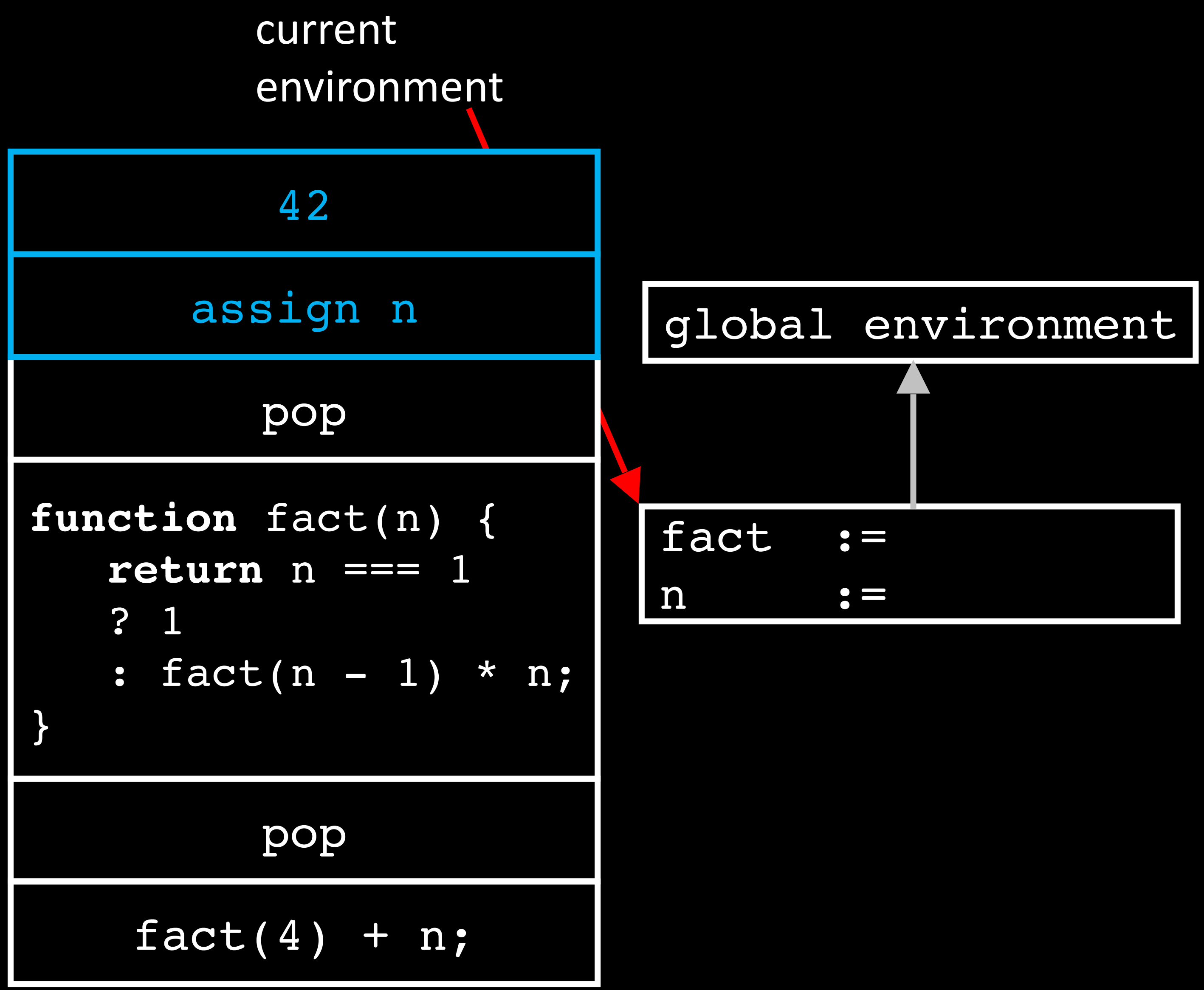
```
fact(4) + n;
```



```
const n = 42;
```

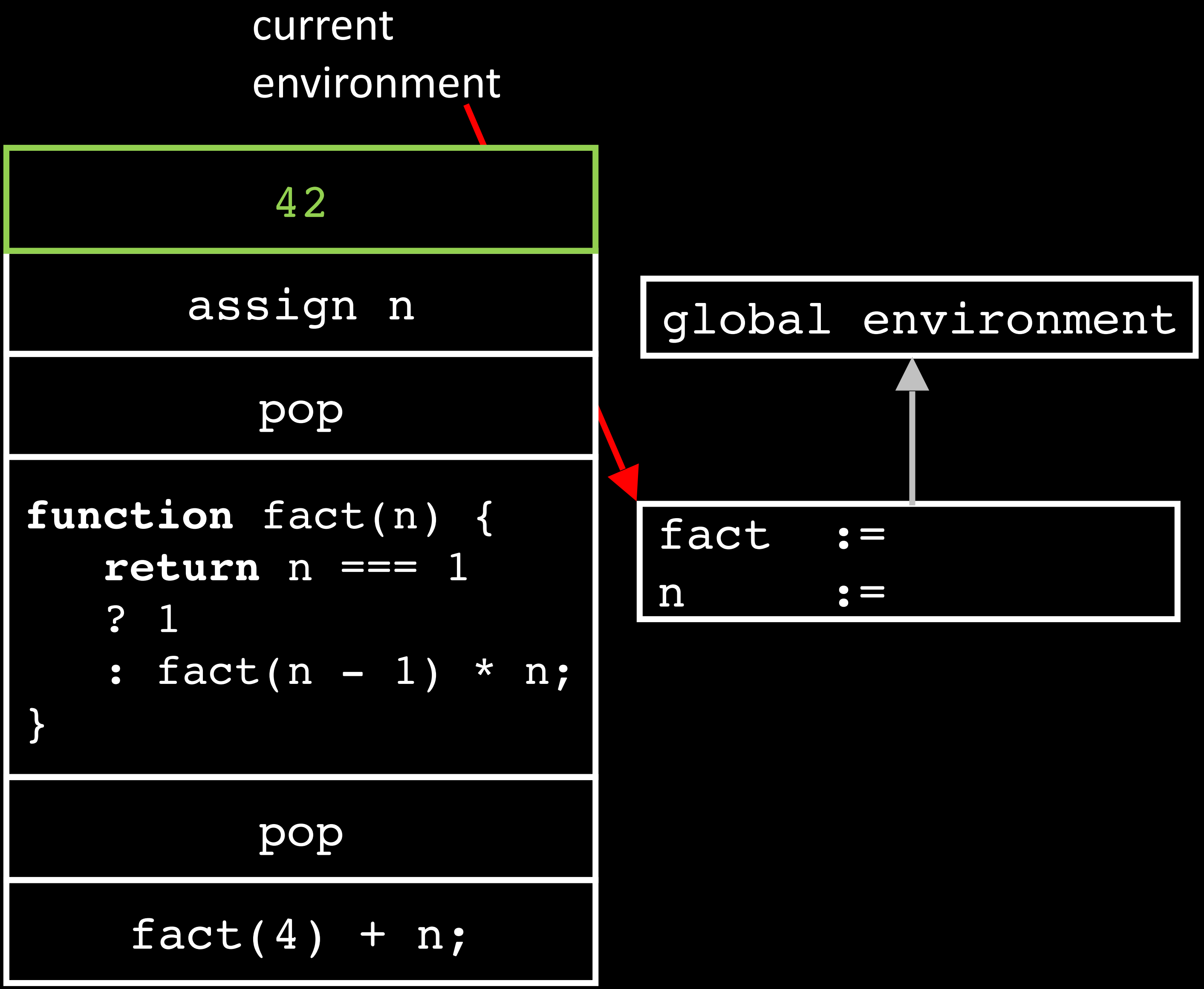
```
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
```

```
fact(4) + n;
```



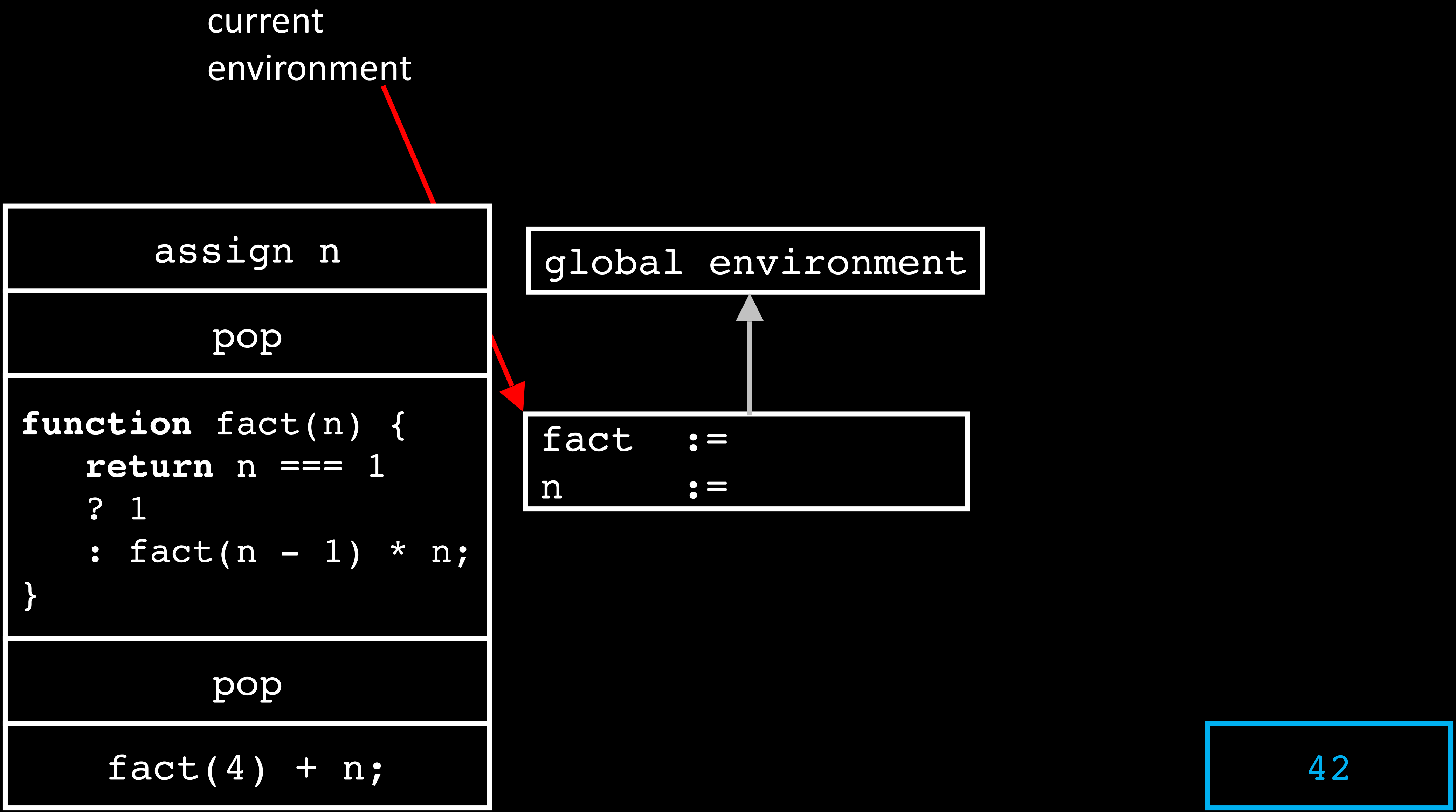
```
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
```

```
fact(4) + n;
```

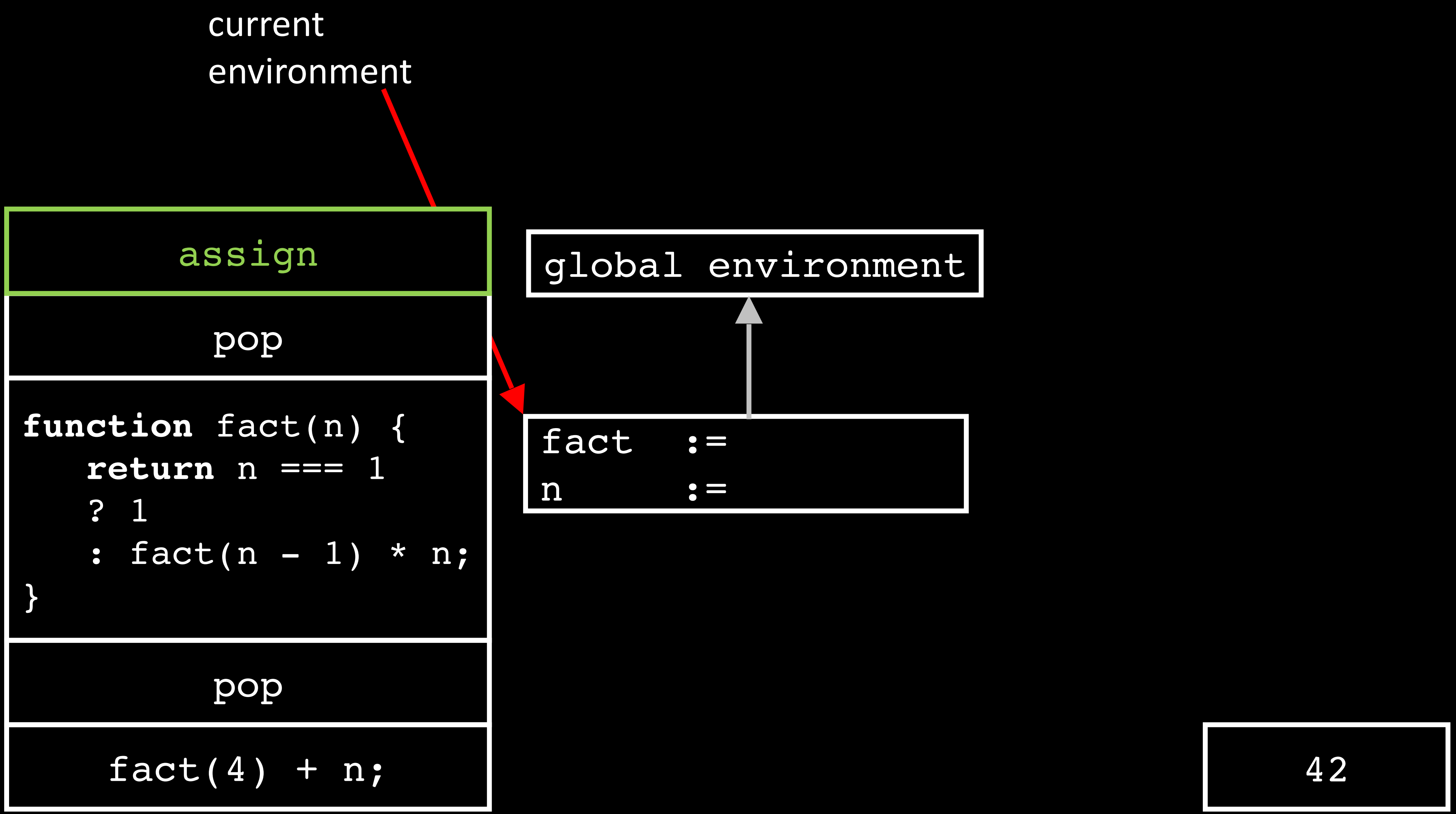


```
function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}
```

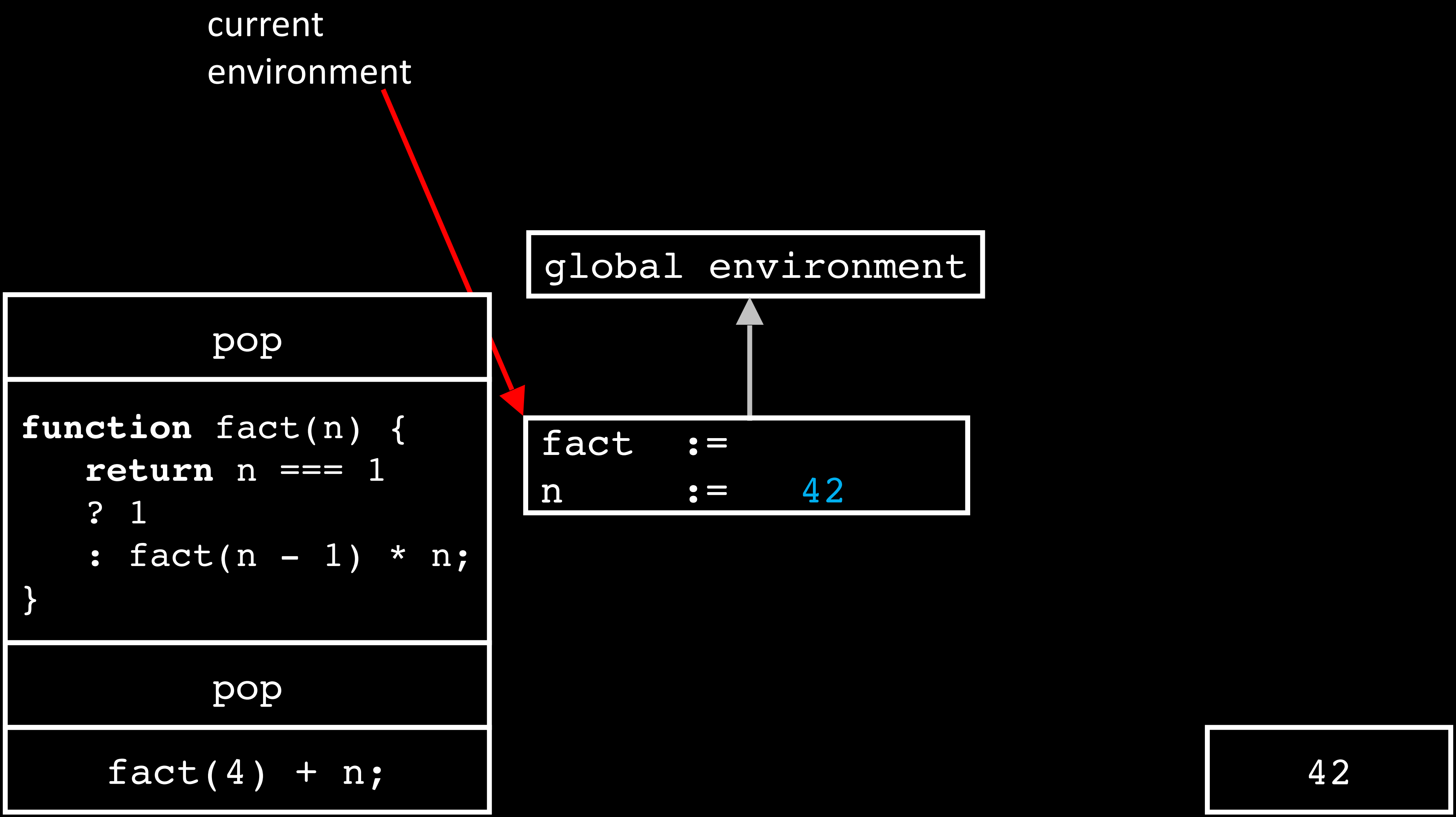
```
fact(4) + n;
```



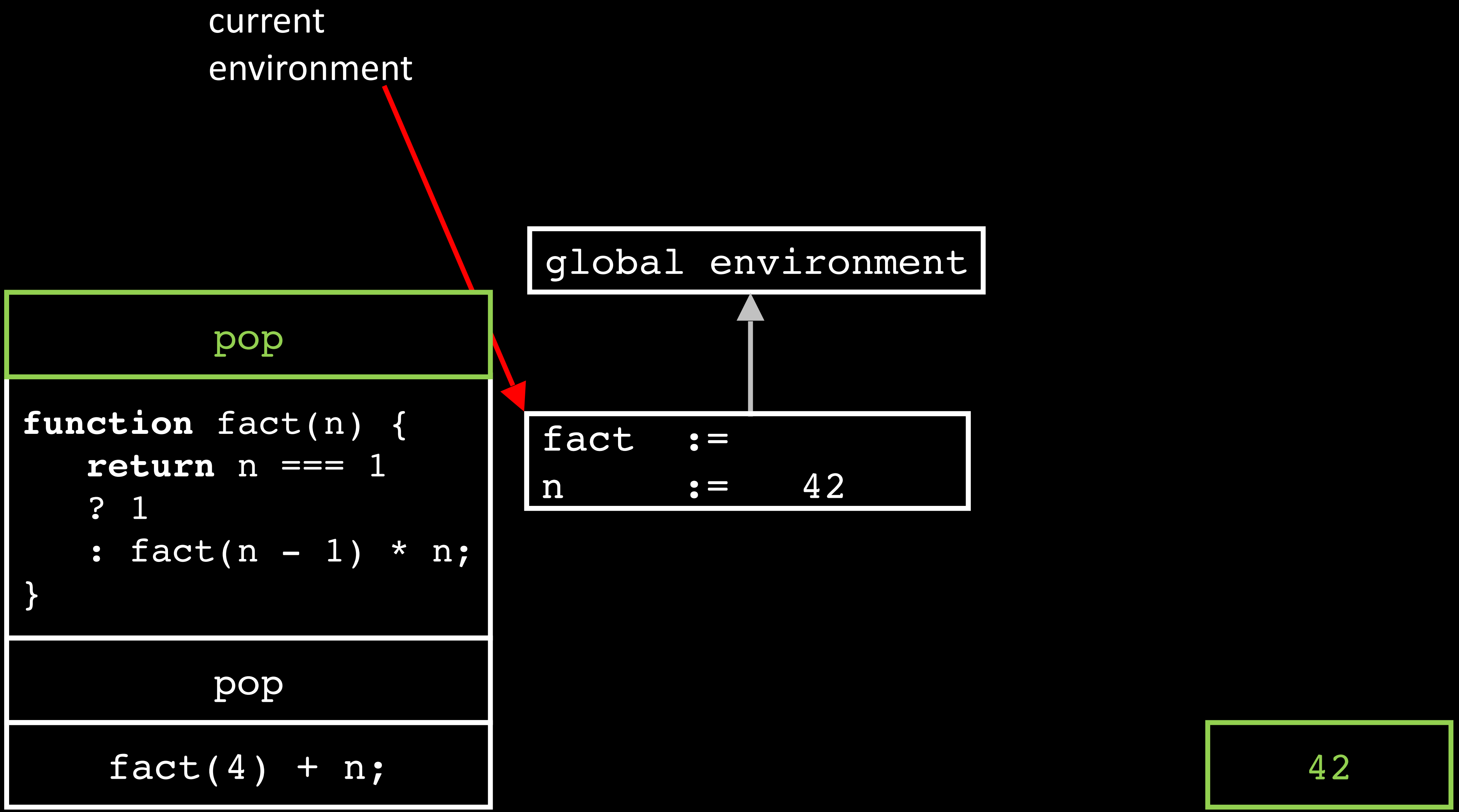
```
function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}
```



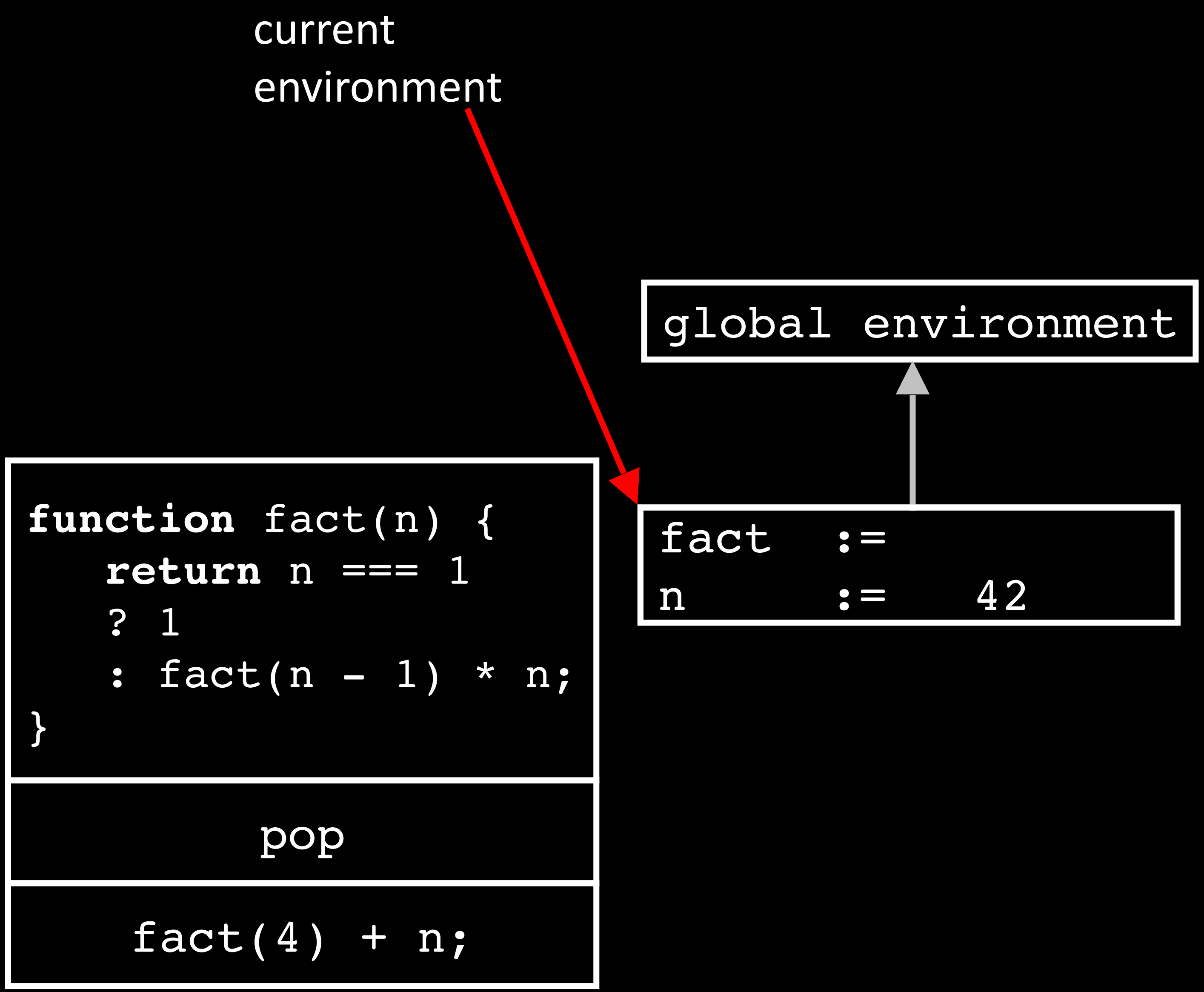
```
function fact(n) {
    return n === 1
    ? 1
    : fact(n - 1) * n;
}
```



```
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
```



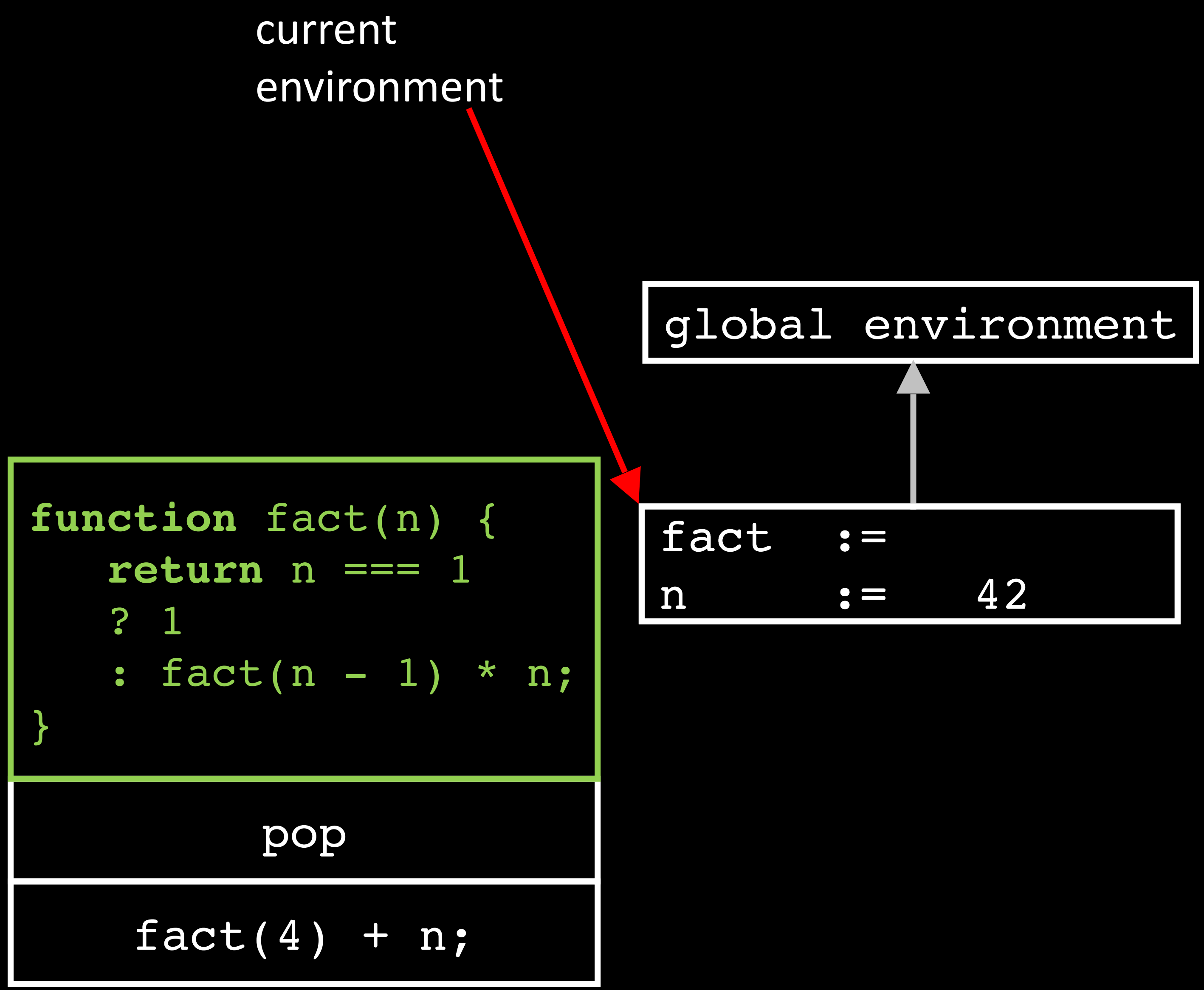
```
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
```



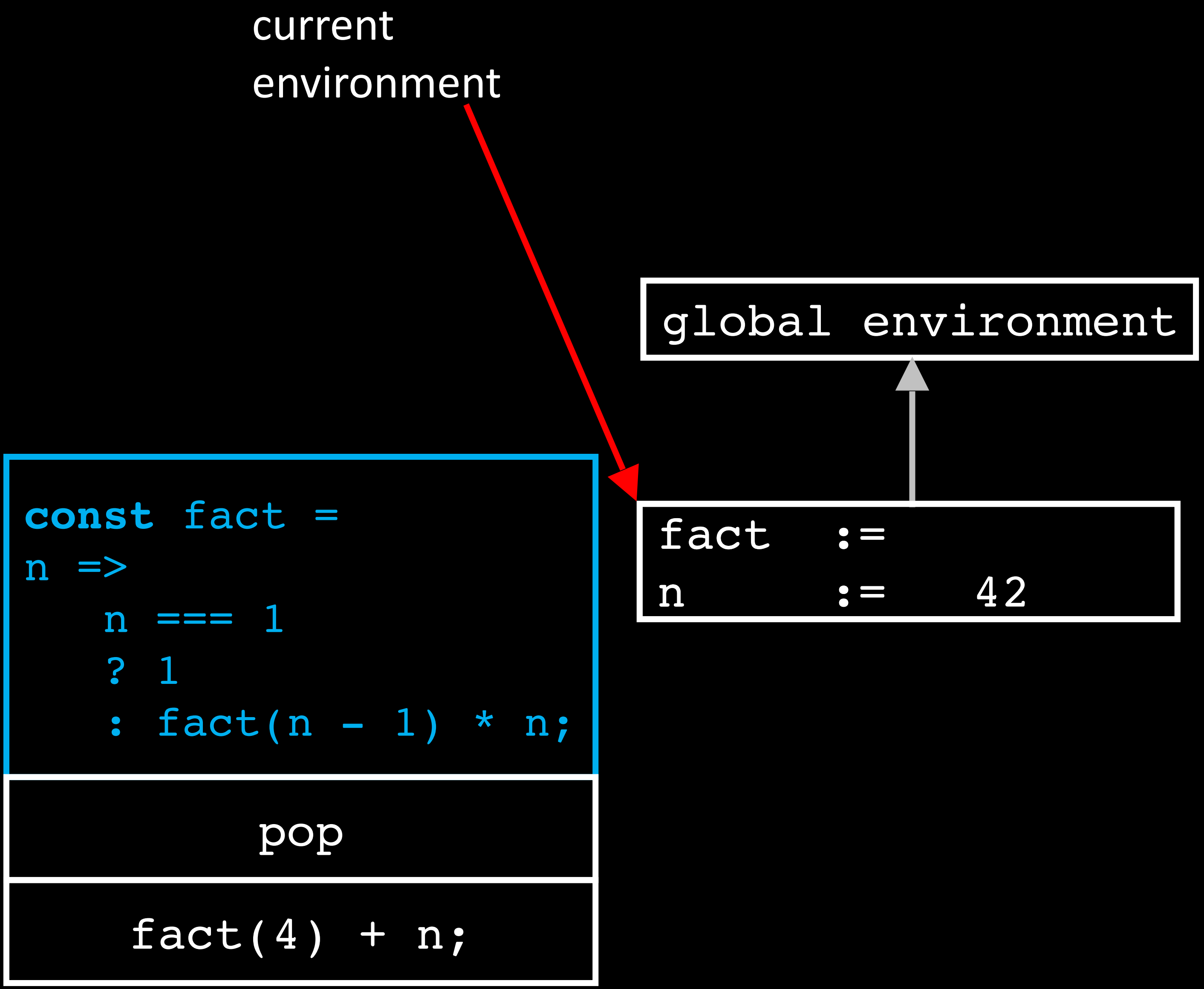
```
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
```

```
pop
```

```
fact(4) + n;
```

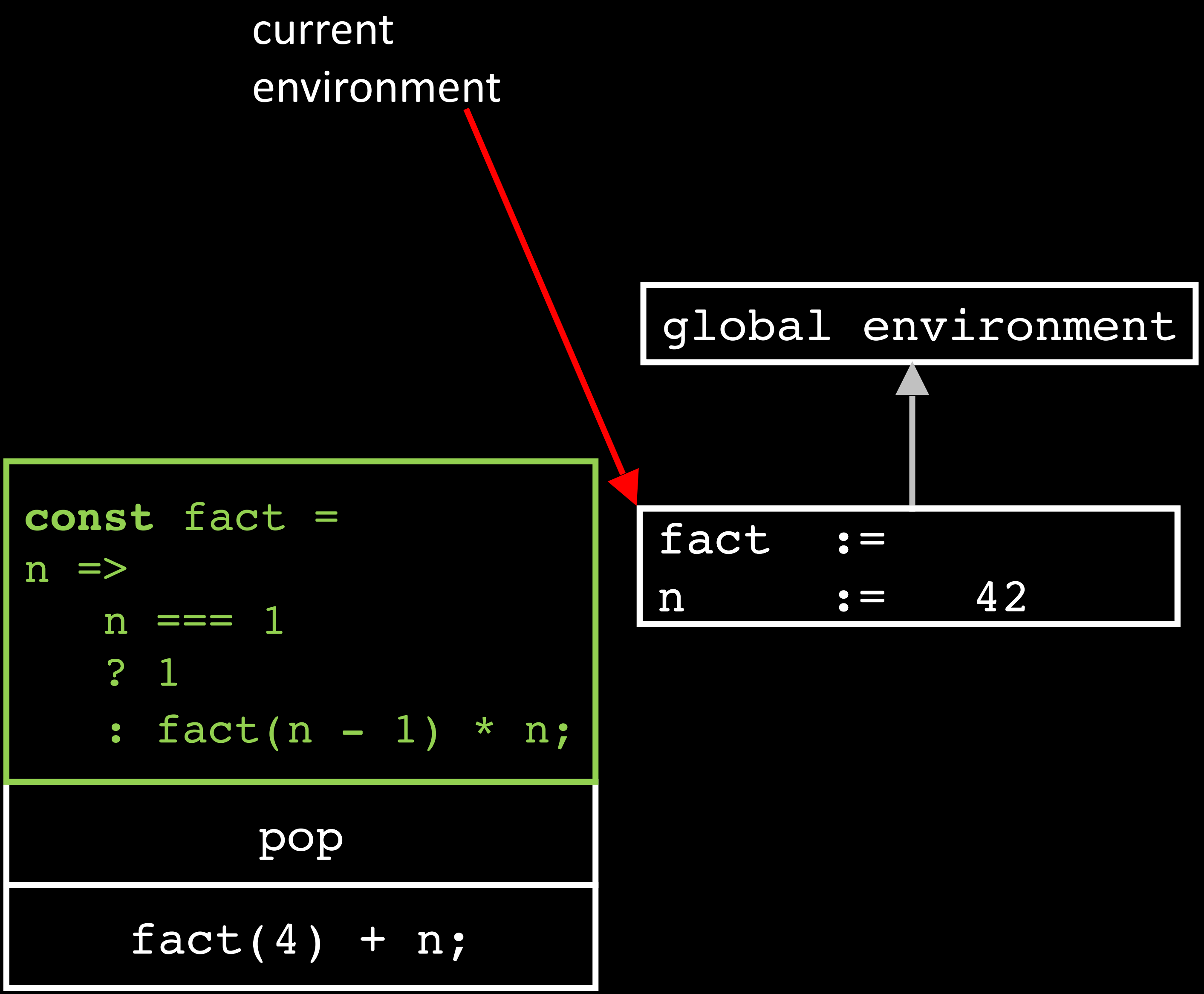


```
function fact(n) {
   return n === 1
   ? 1
   : fact(n - 1) * n;
}
```

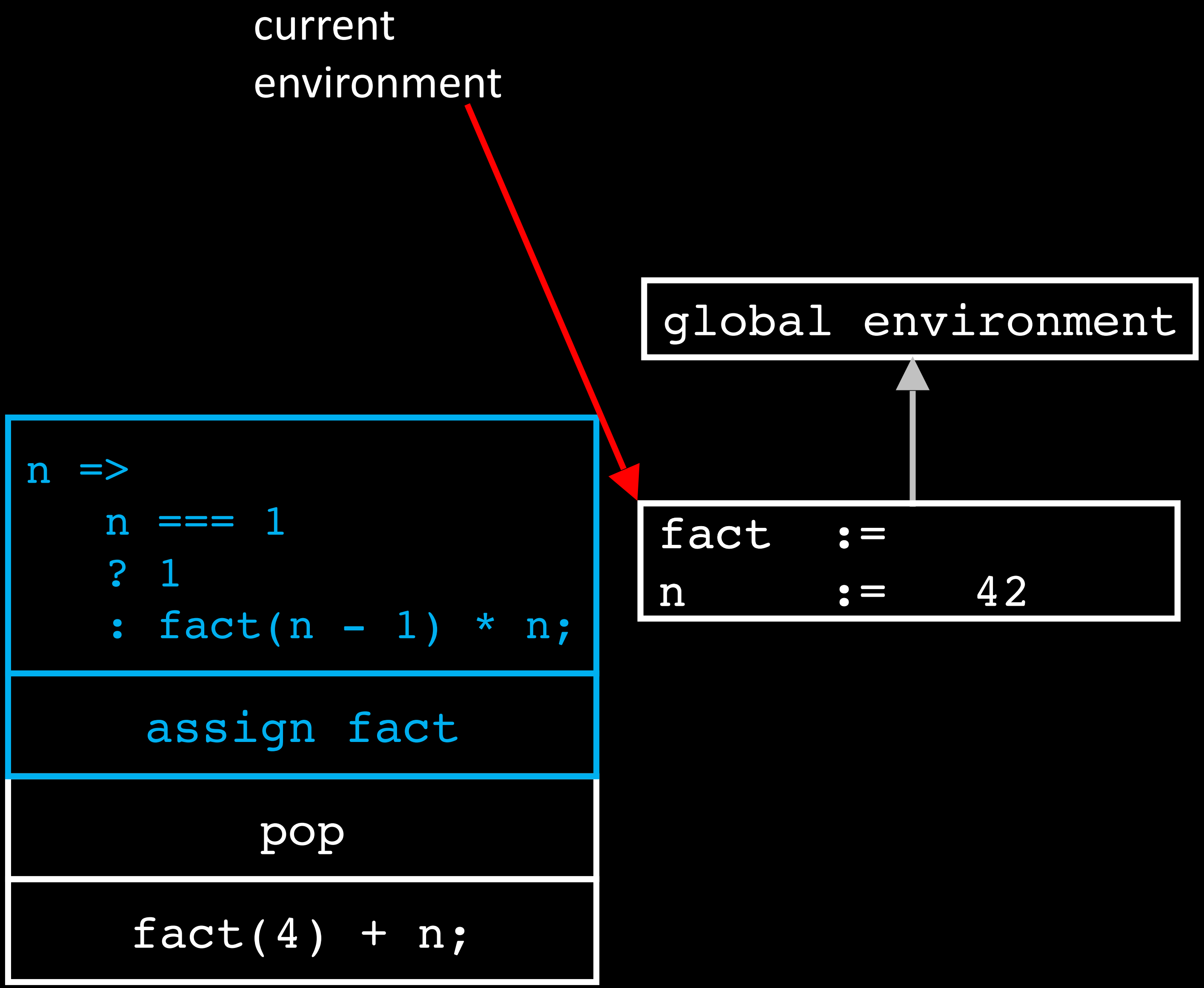


```
const fact =
n =>
    n === 1
    ? 1
    : fact(n - 1) * n;
```

```
fact(4) + n;
```



```
const fact =
n =>
   n === 1
   ? 1
   : fact(n - 1) * n;
```

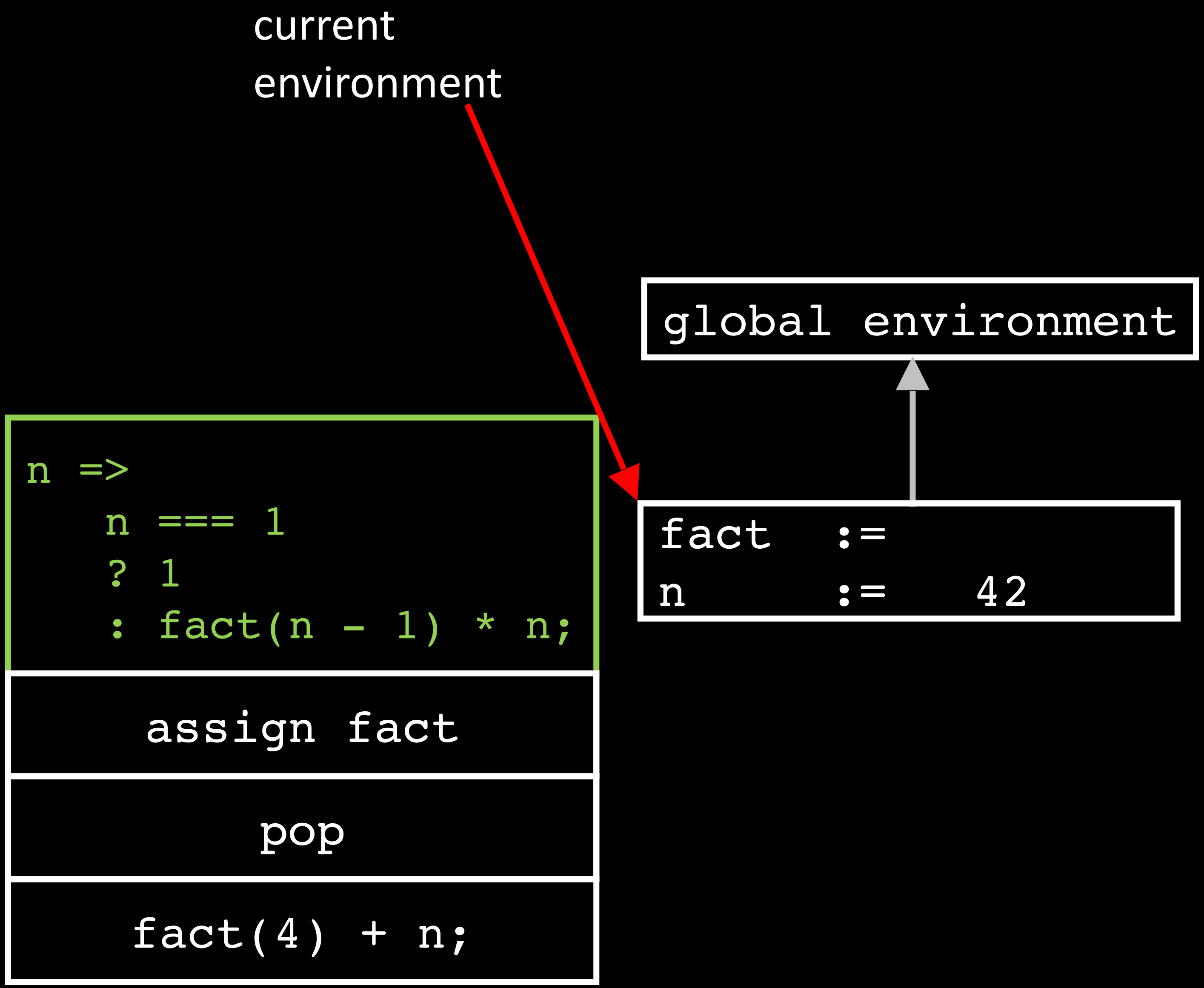


```
n =>
   n === 1
   ? 1
   : fact(n - 1) * n;
```

```
assign fact
```

```
pop
```

```
fact(4) + n;
```

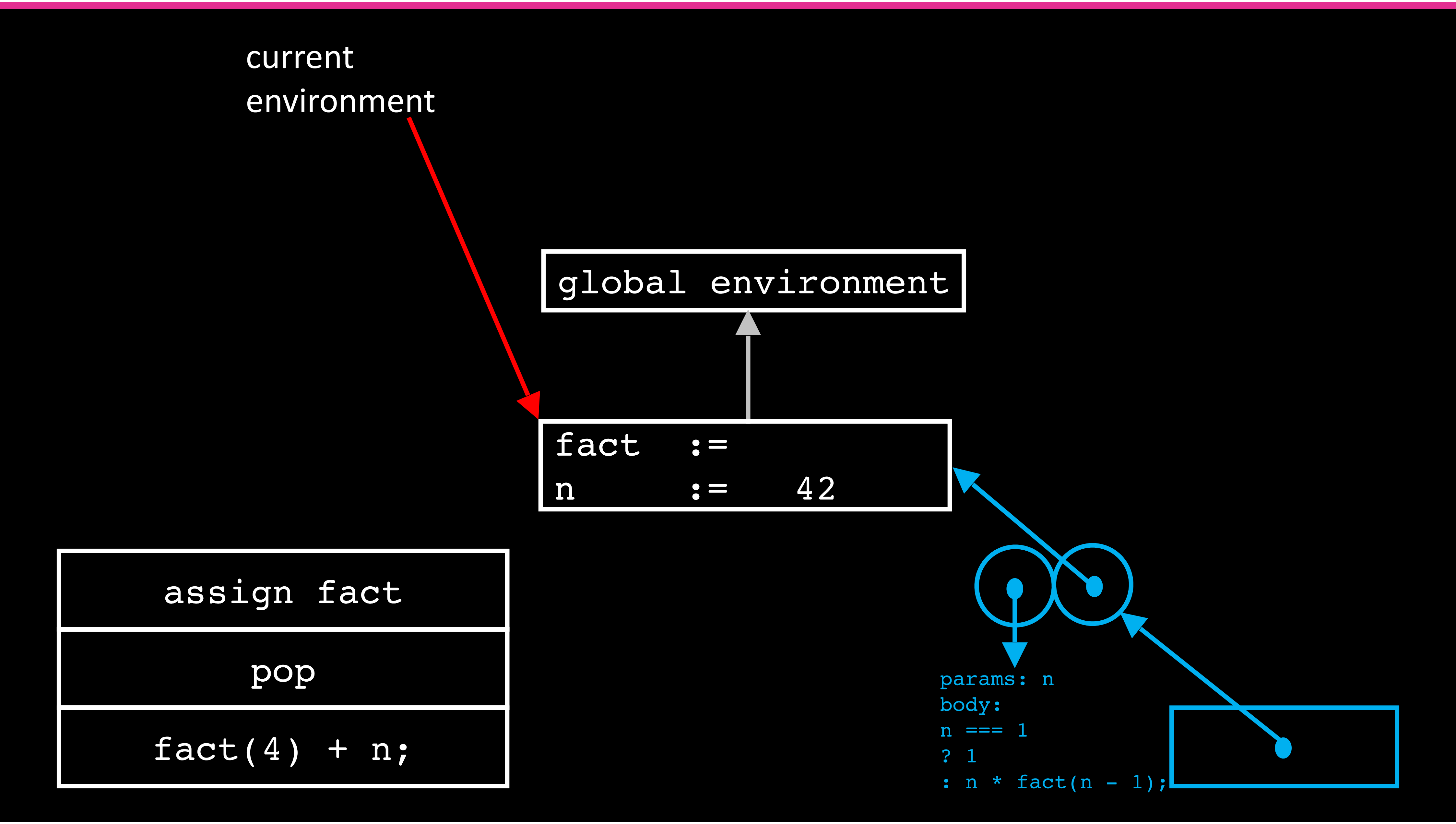


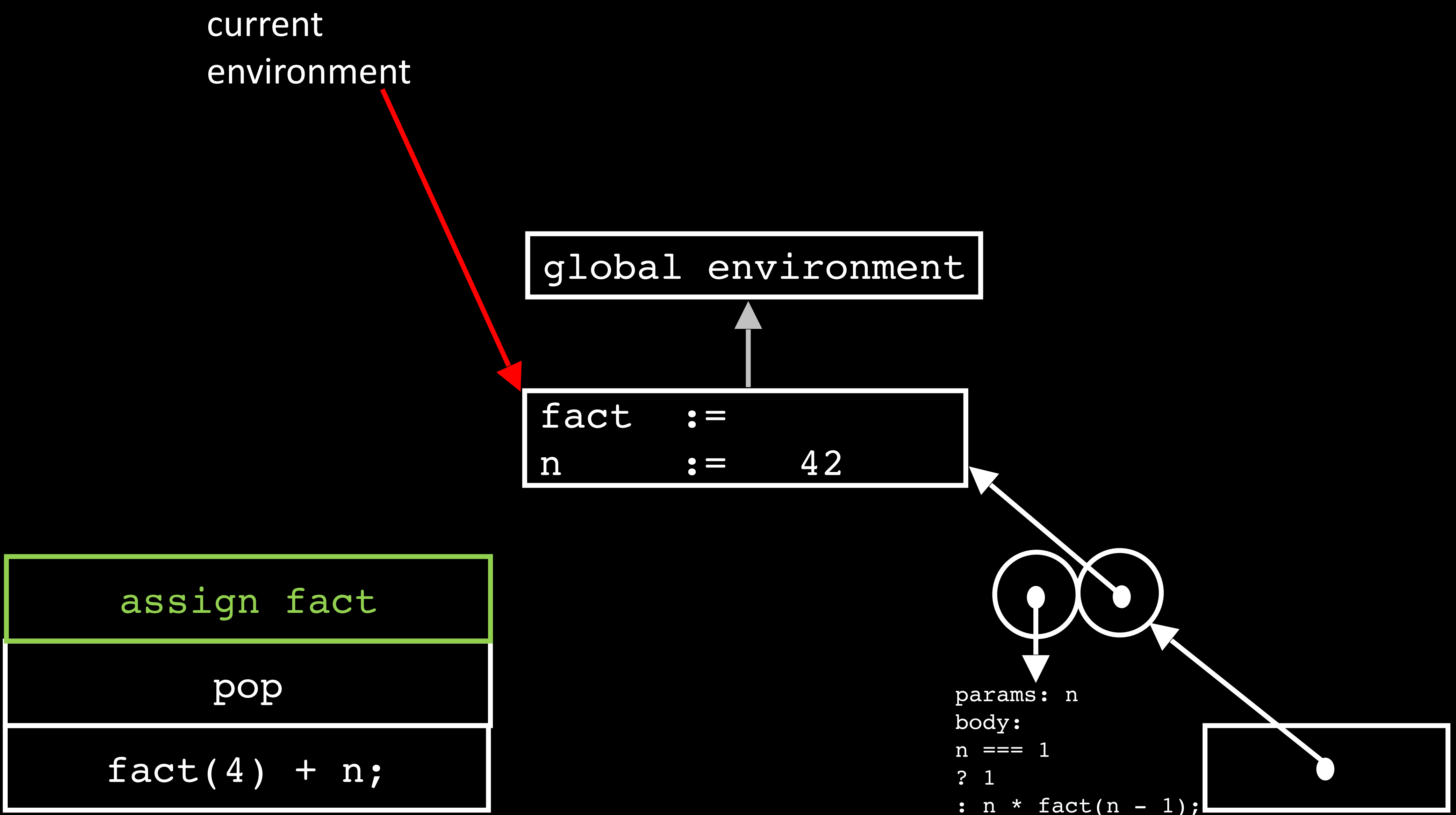
```
n =>
   n === 1
   ? 1
   : fact(n - 1) * n;
```

```
assign fact
```

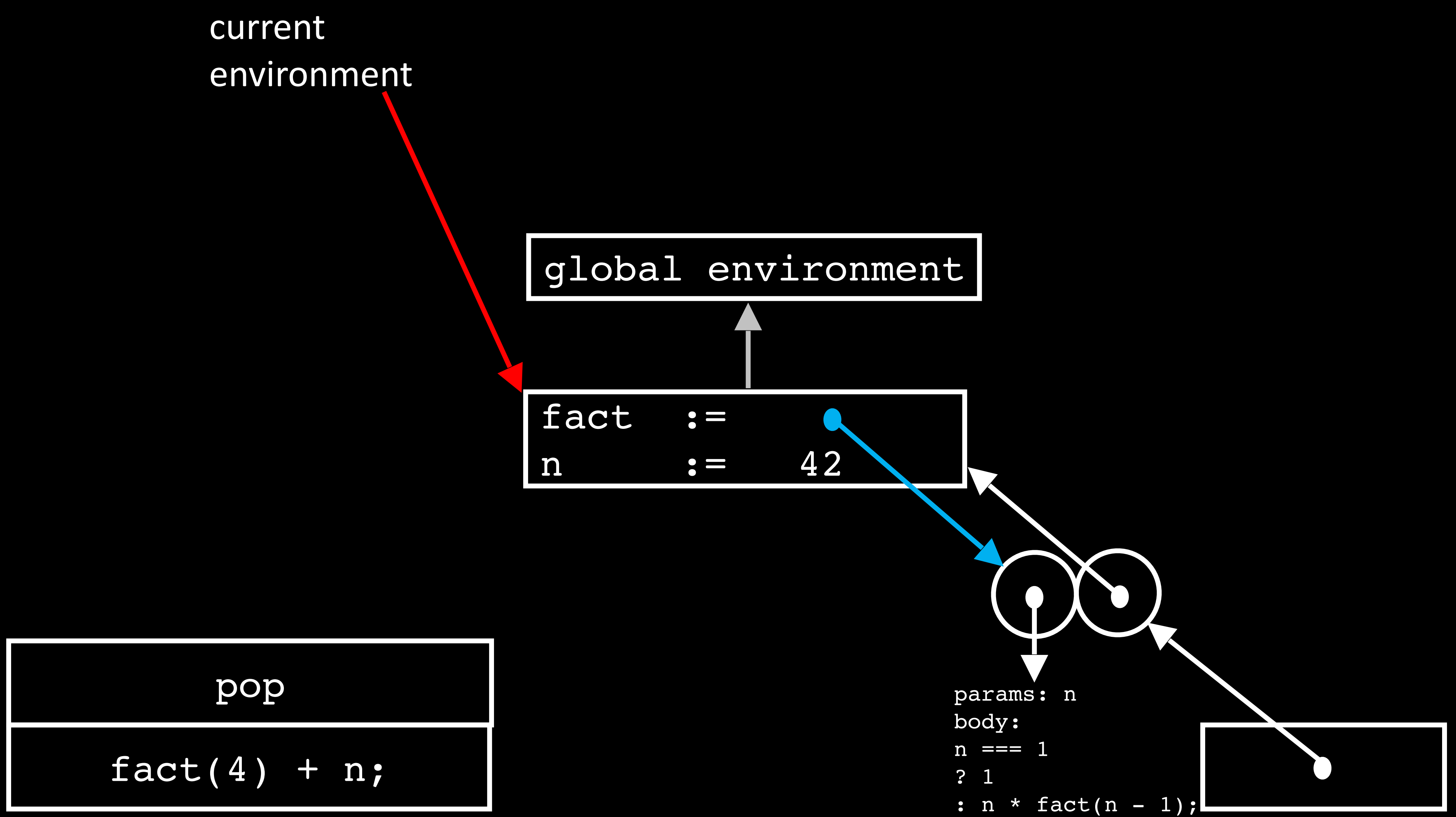
```
pop
```

```
fact(4) + n;
```

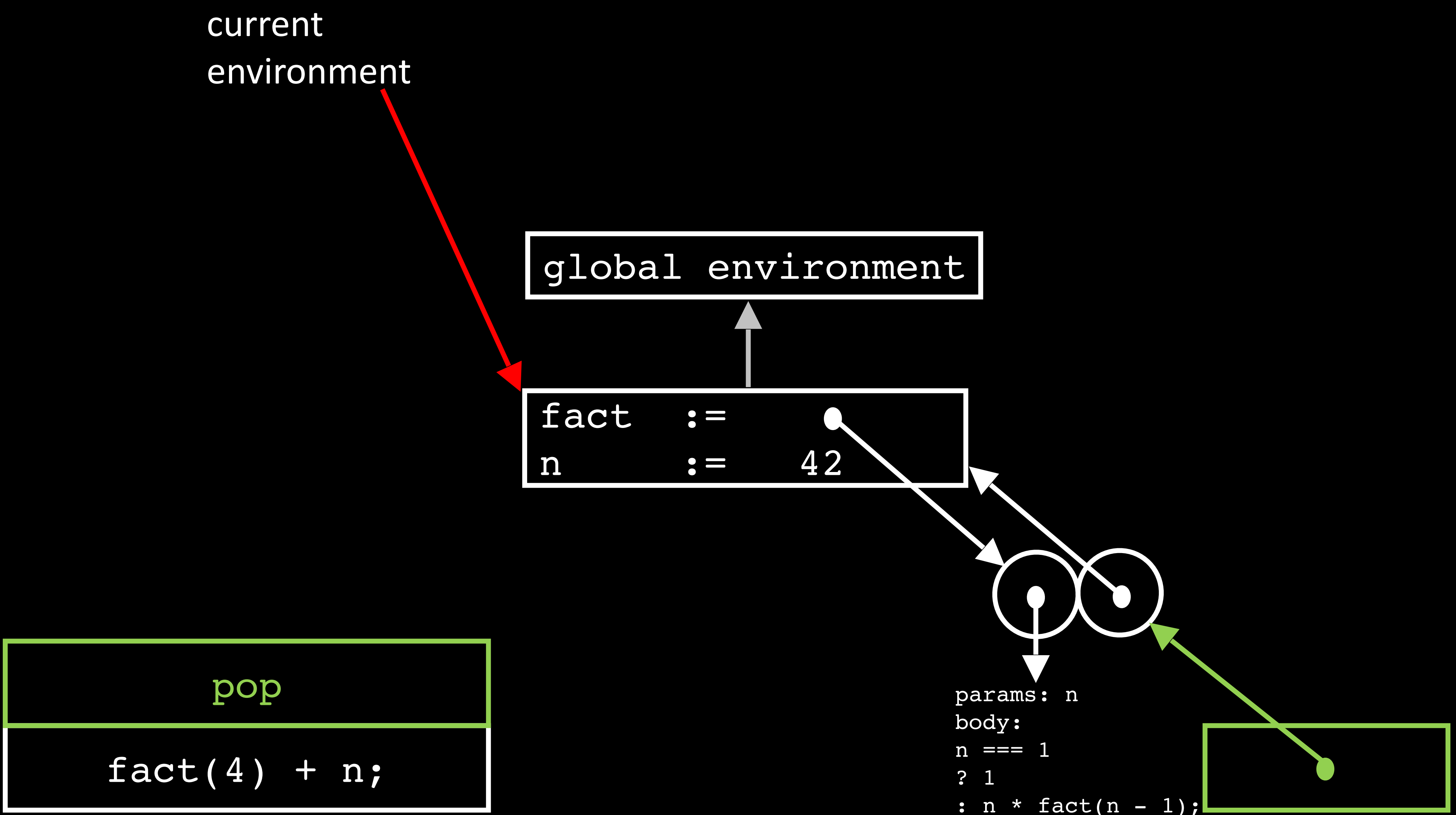

current
environment
global environment
fact :=
n := 42
assign fact
pop
fact(4) + n;
params: n
body:
n === 1
? 1
: n * fact(n - 1);



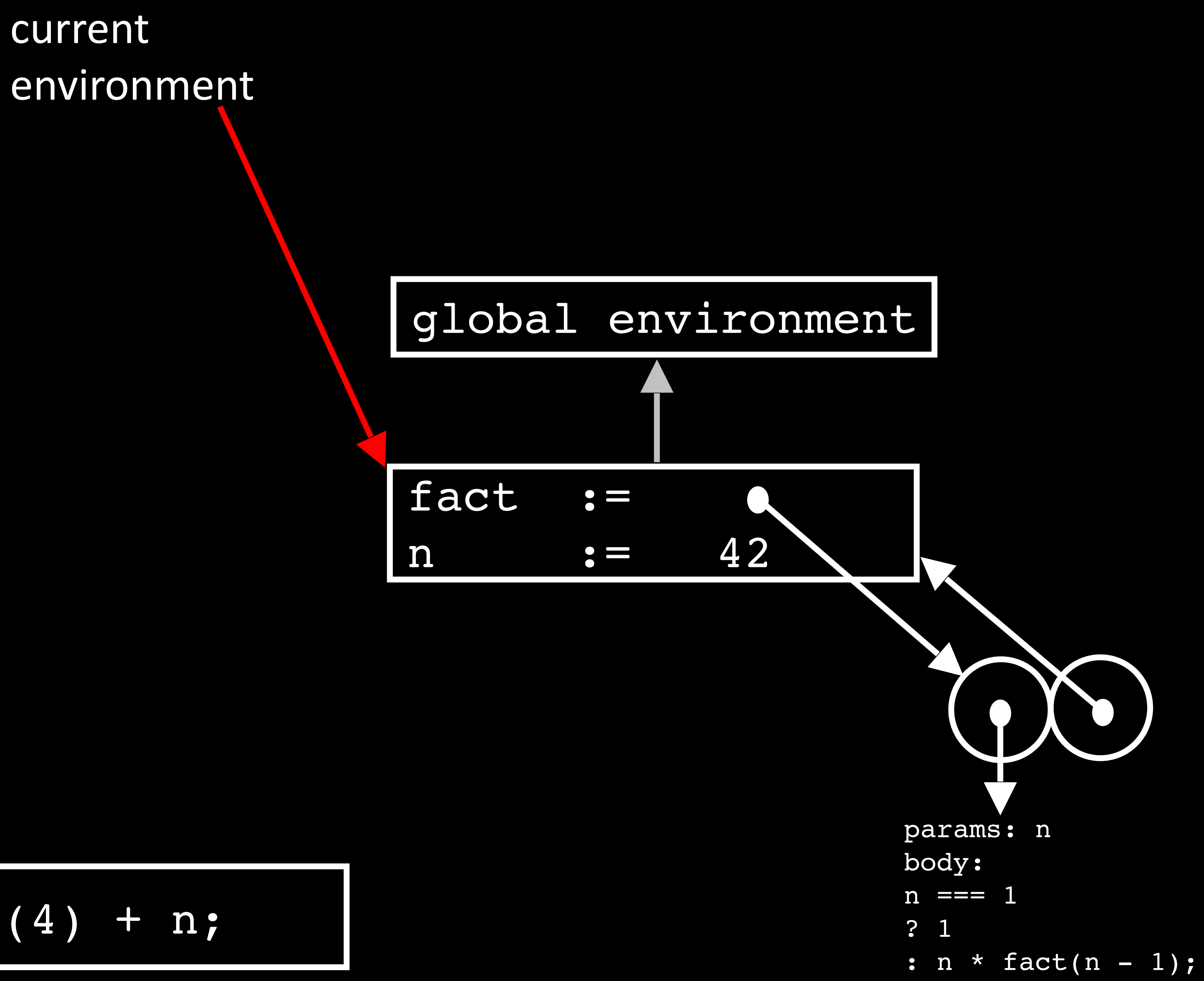
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



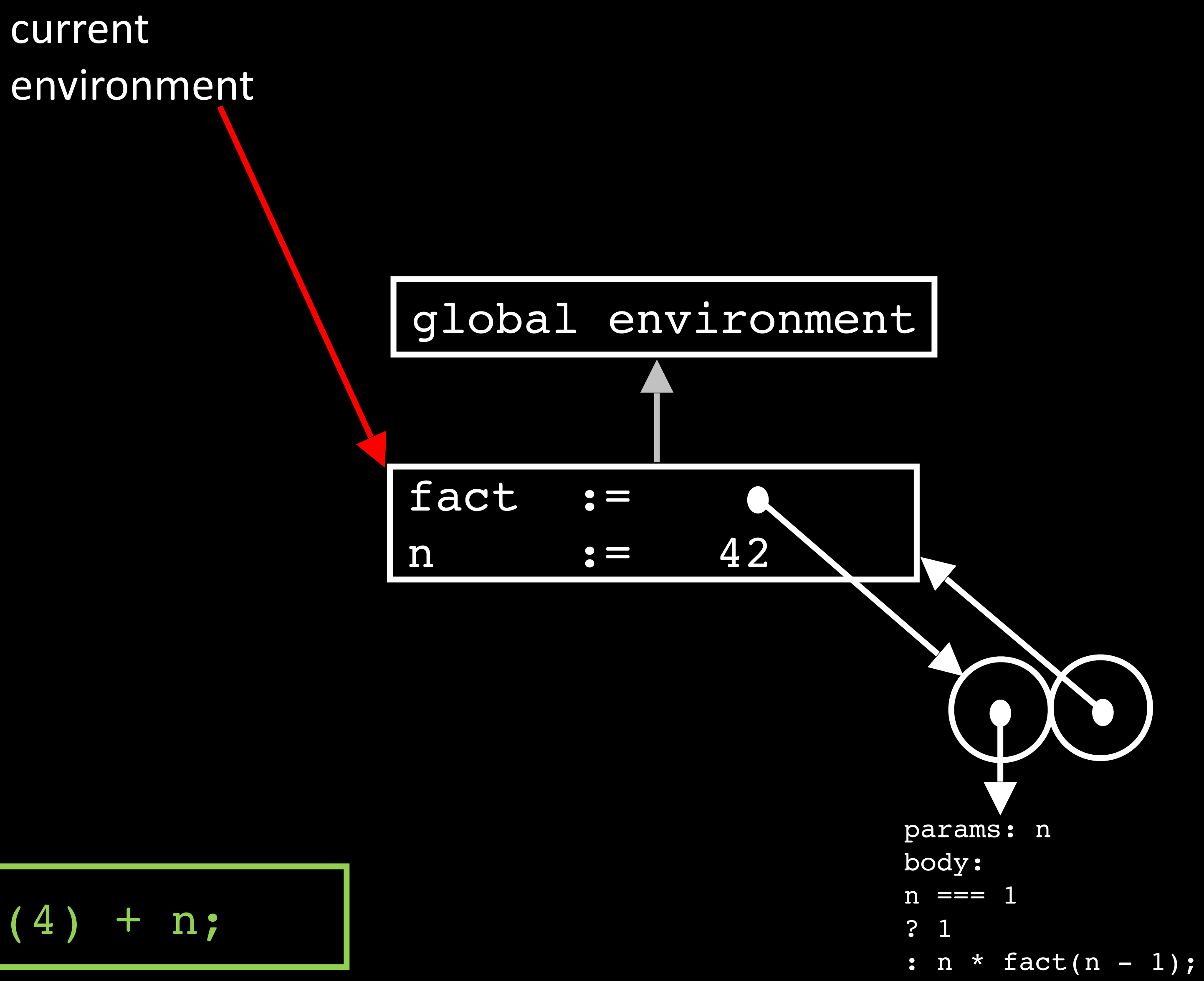
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



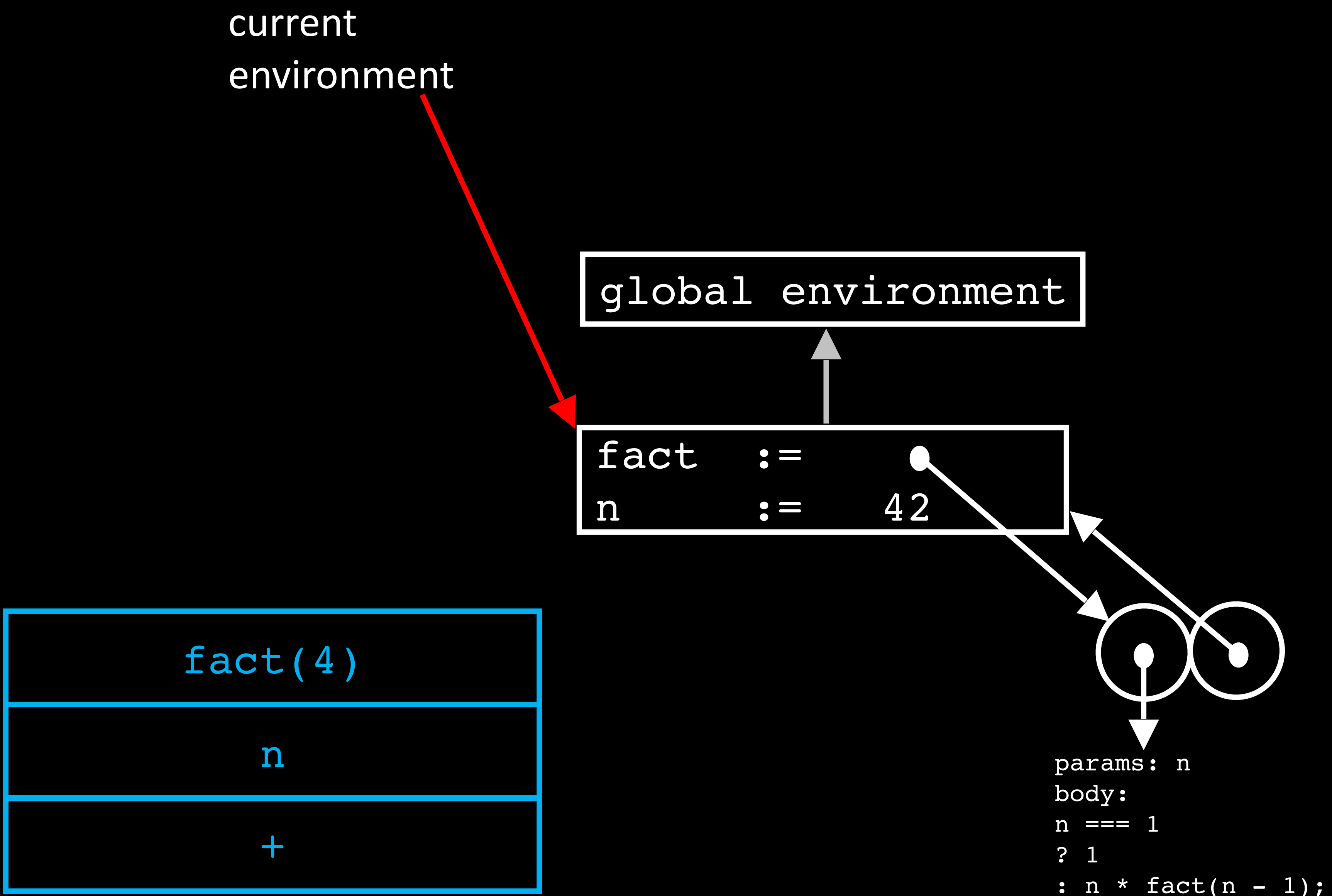
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```


current
environment
global environment
fact :=
n := 42
params: n
body:
n === 1
? 1
: n * fact(n - 1);

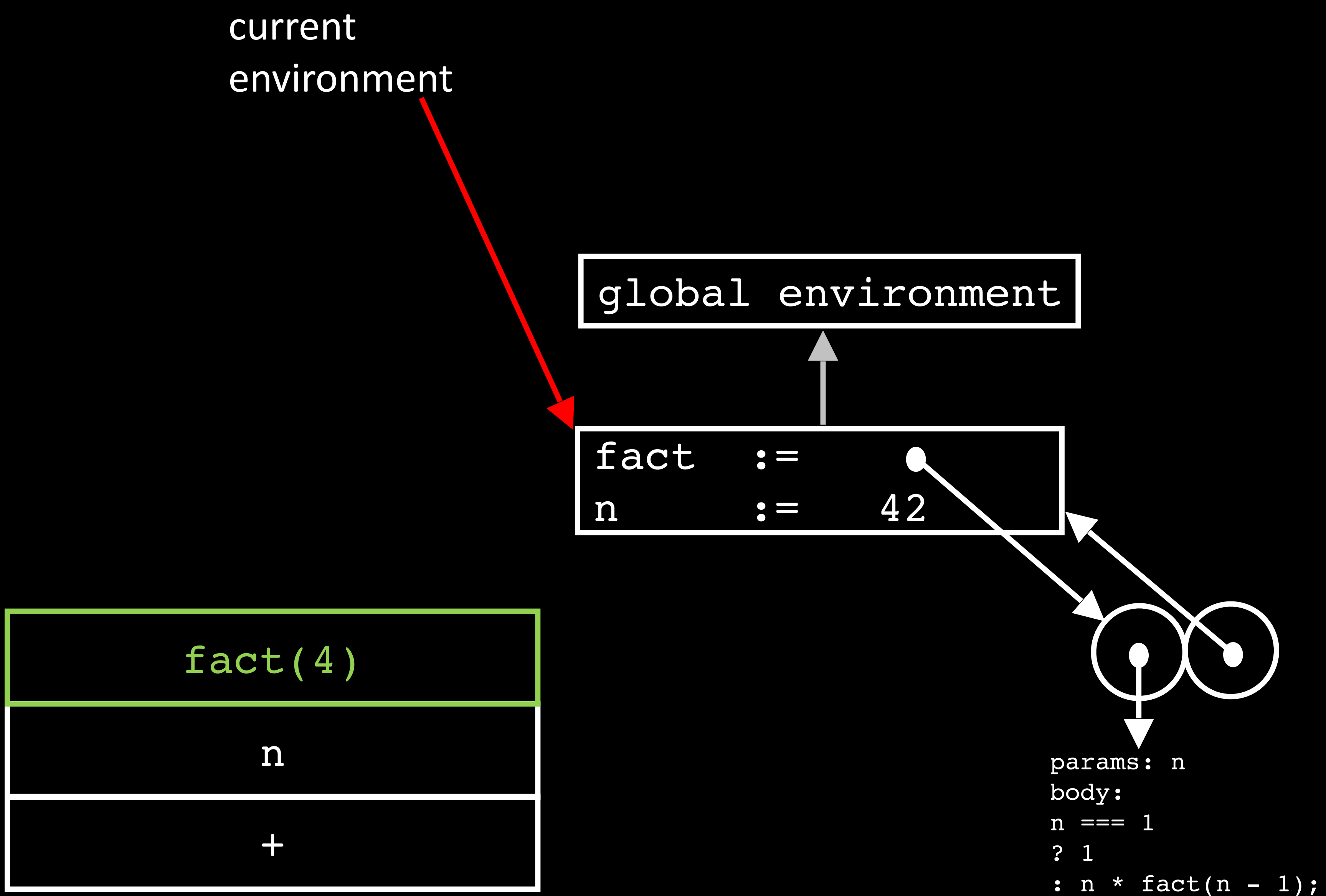
```
fact(4) + n;
```


current environment
global environment
fact :=
n := 42
params: n
body:
n === 1
? 1
: n * fact(n - 1);

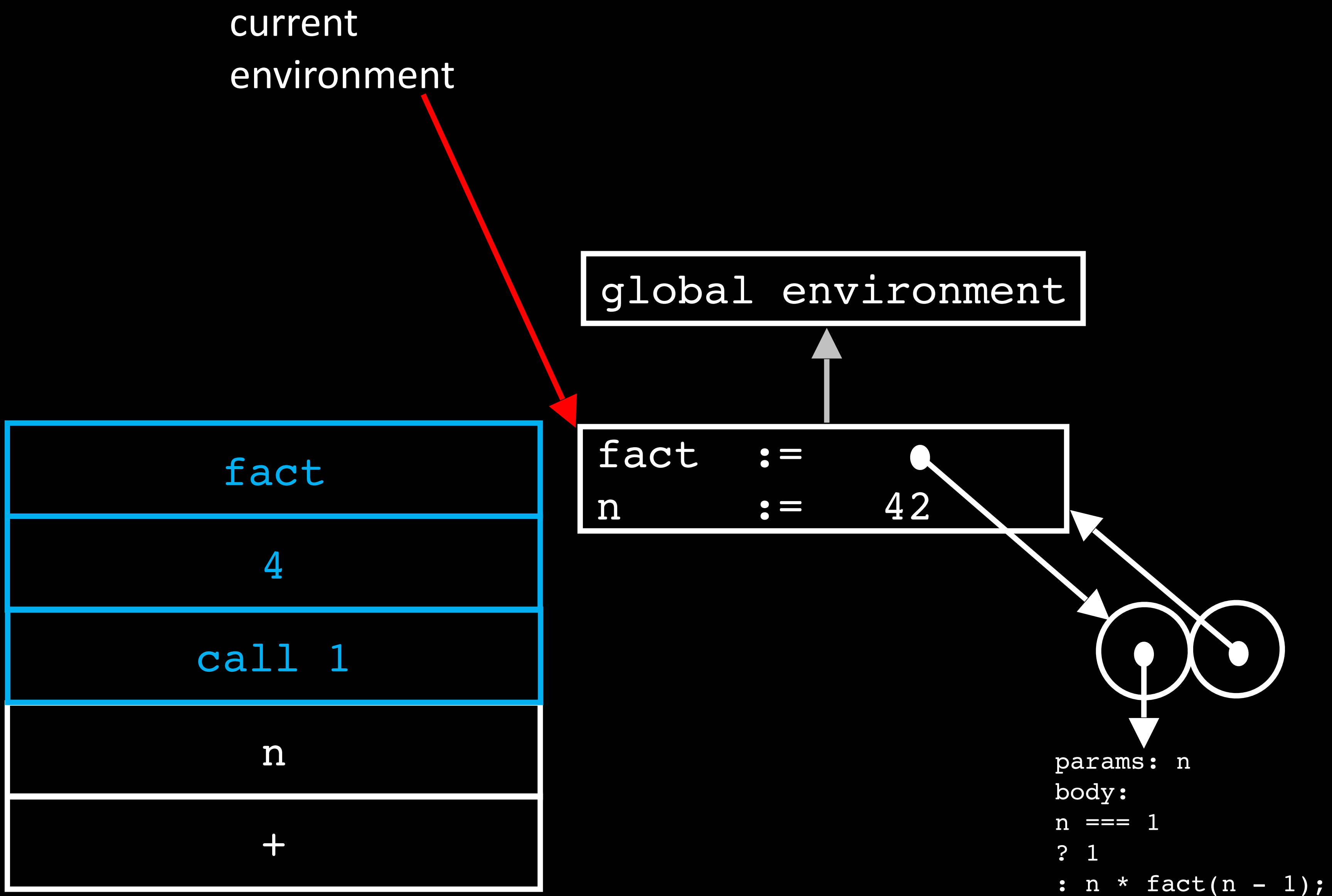
```
fact(4) + n;
```



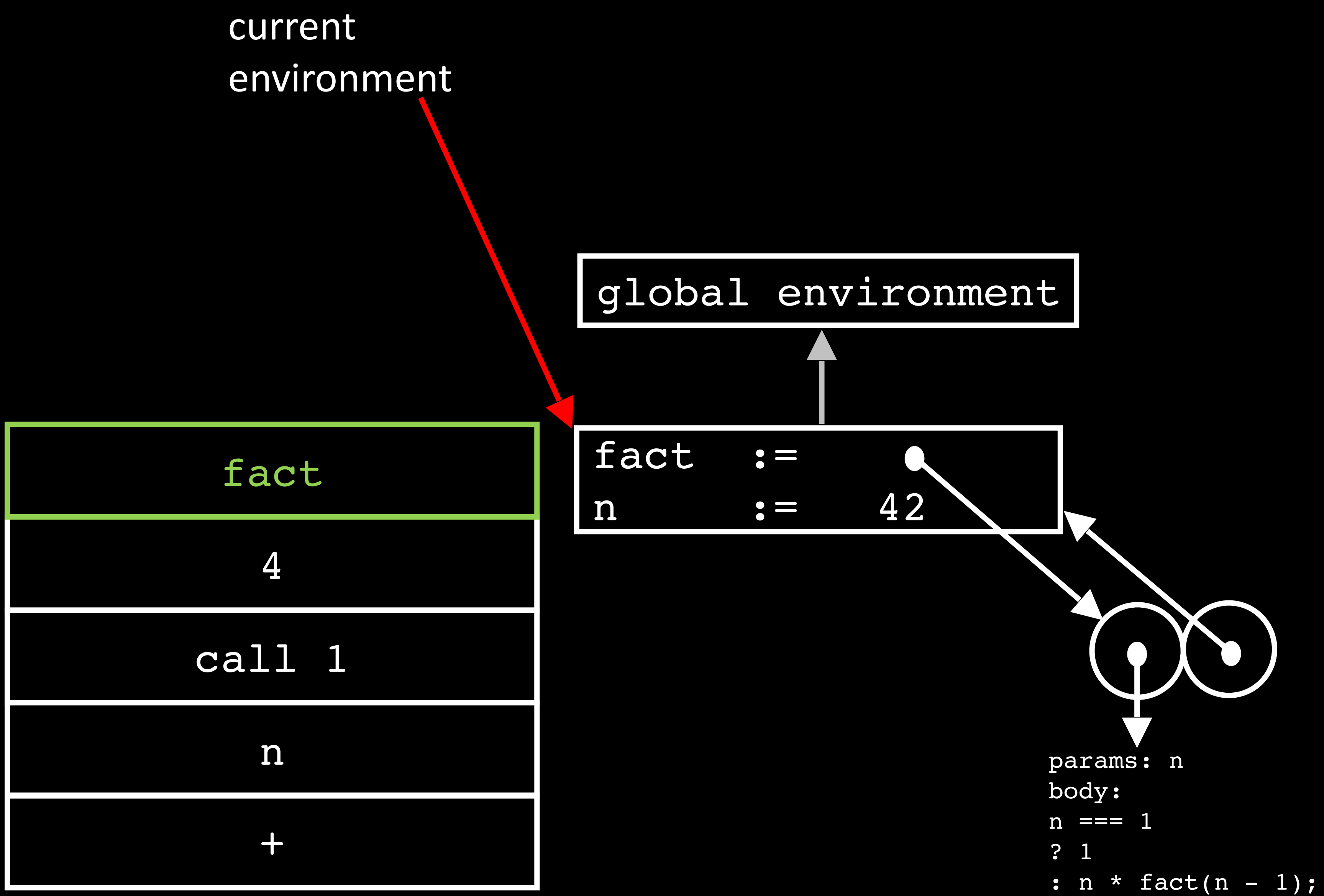
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



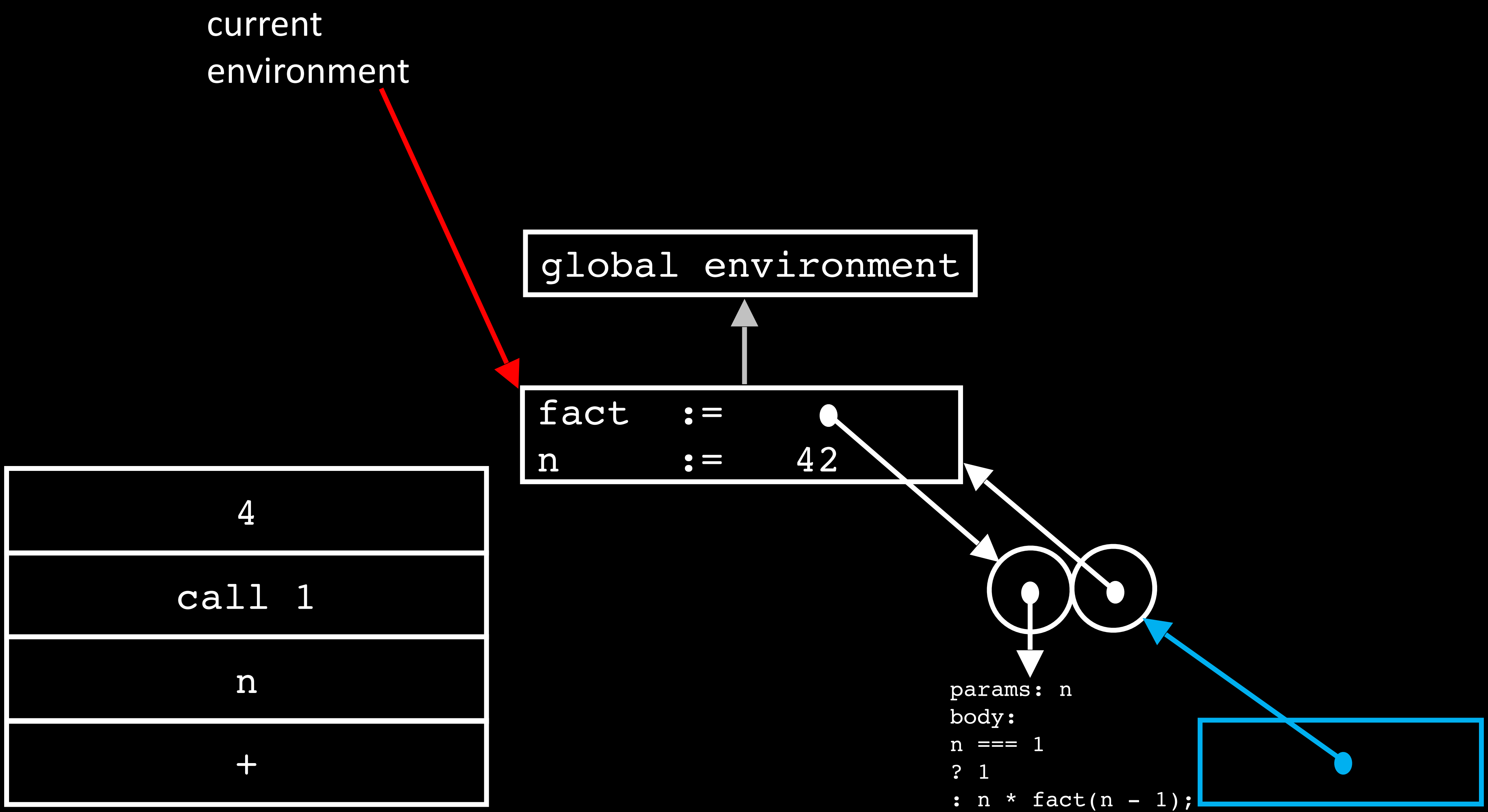
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

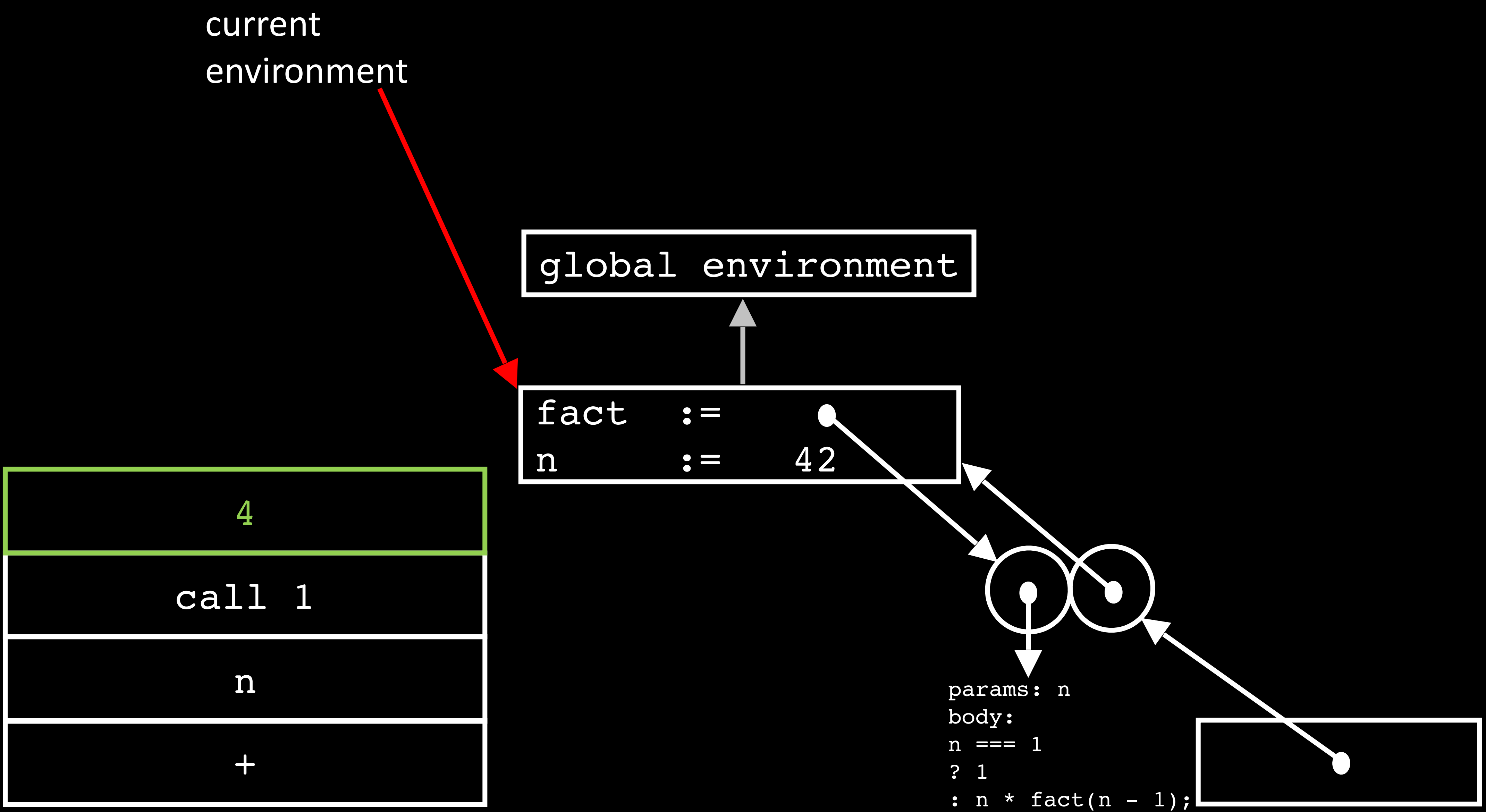


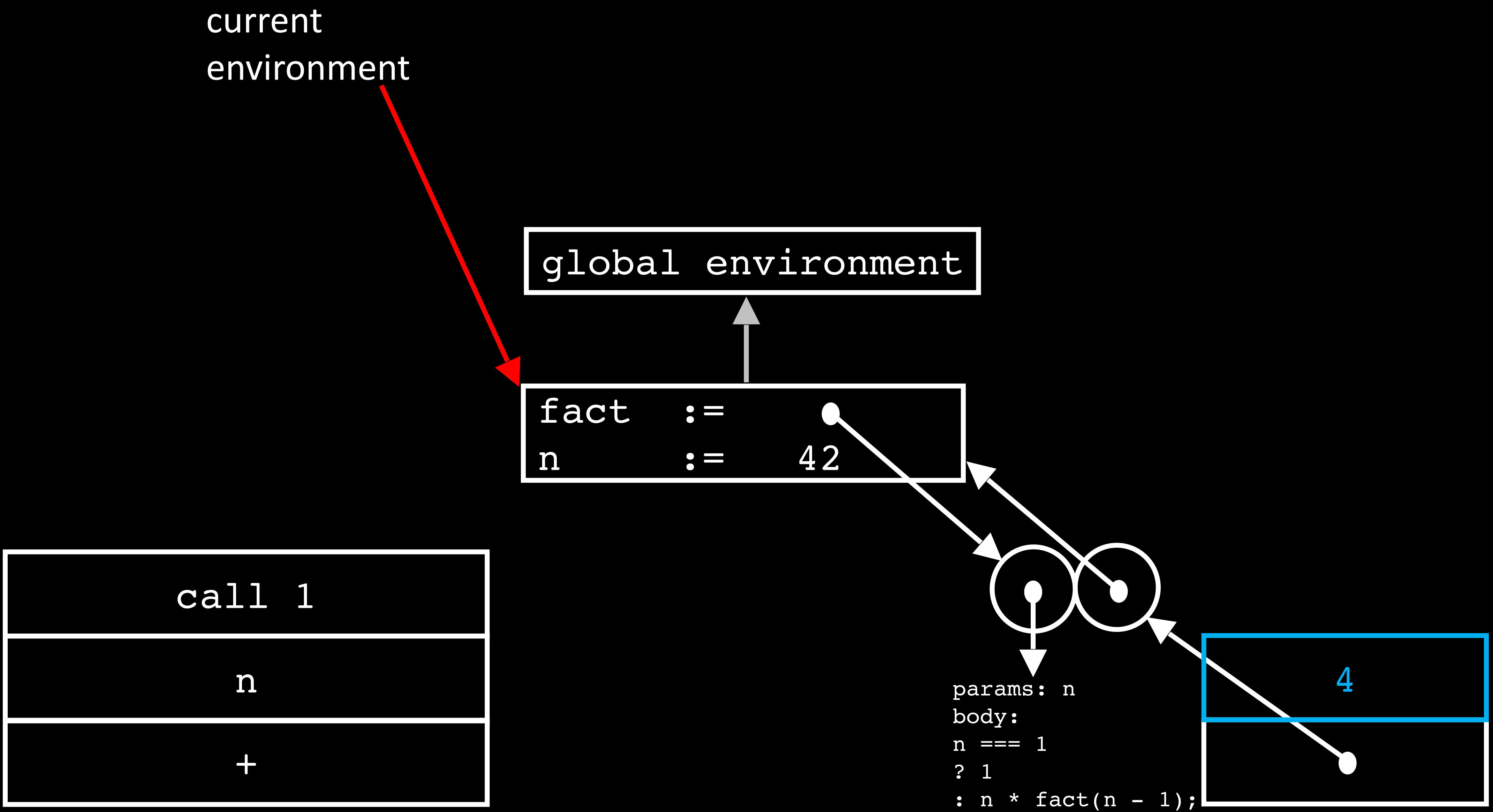
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



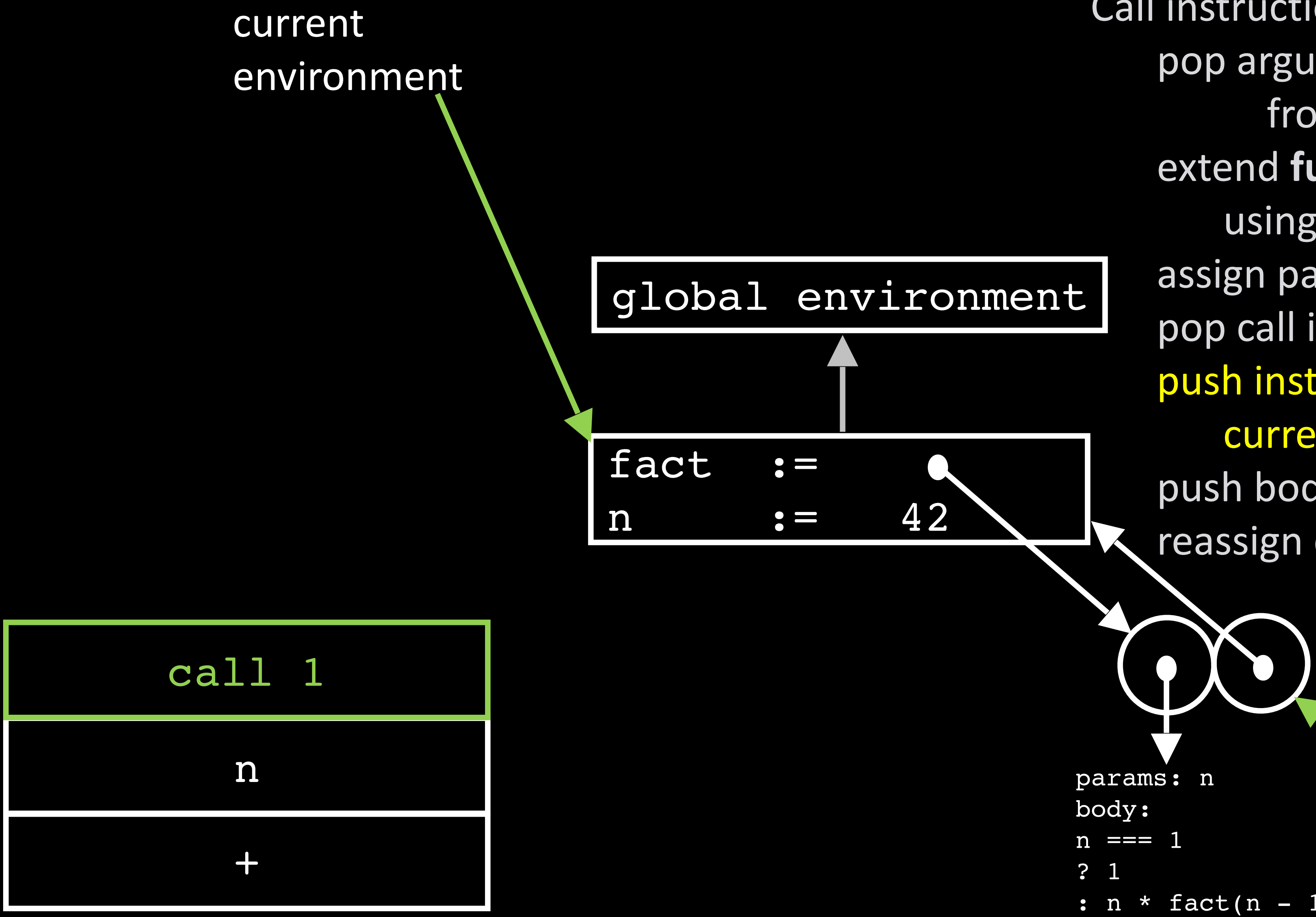
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```


current environment
global environment
fact :=
n := 42
4
call 1
n
+
params: n
body:
n === 1
? 1
: n * fact(n - 1);


current environment
global environment
fact :=
n := 42
4
call 1
n
+
params: n
body:
n === 1
? 1
: n * fact(n - 1);

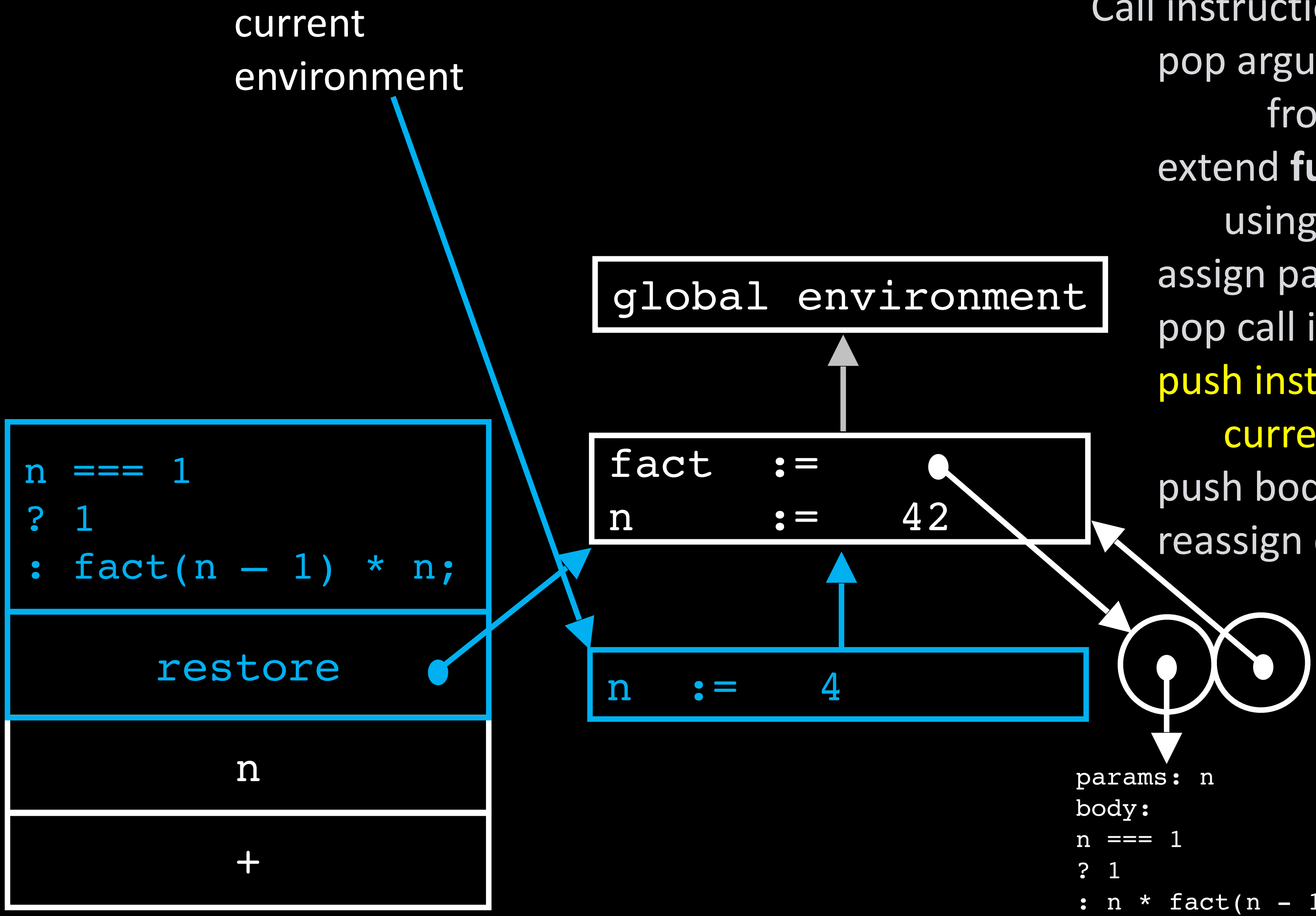


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



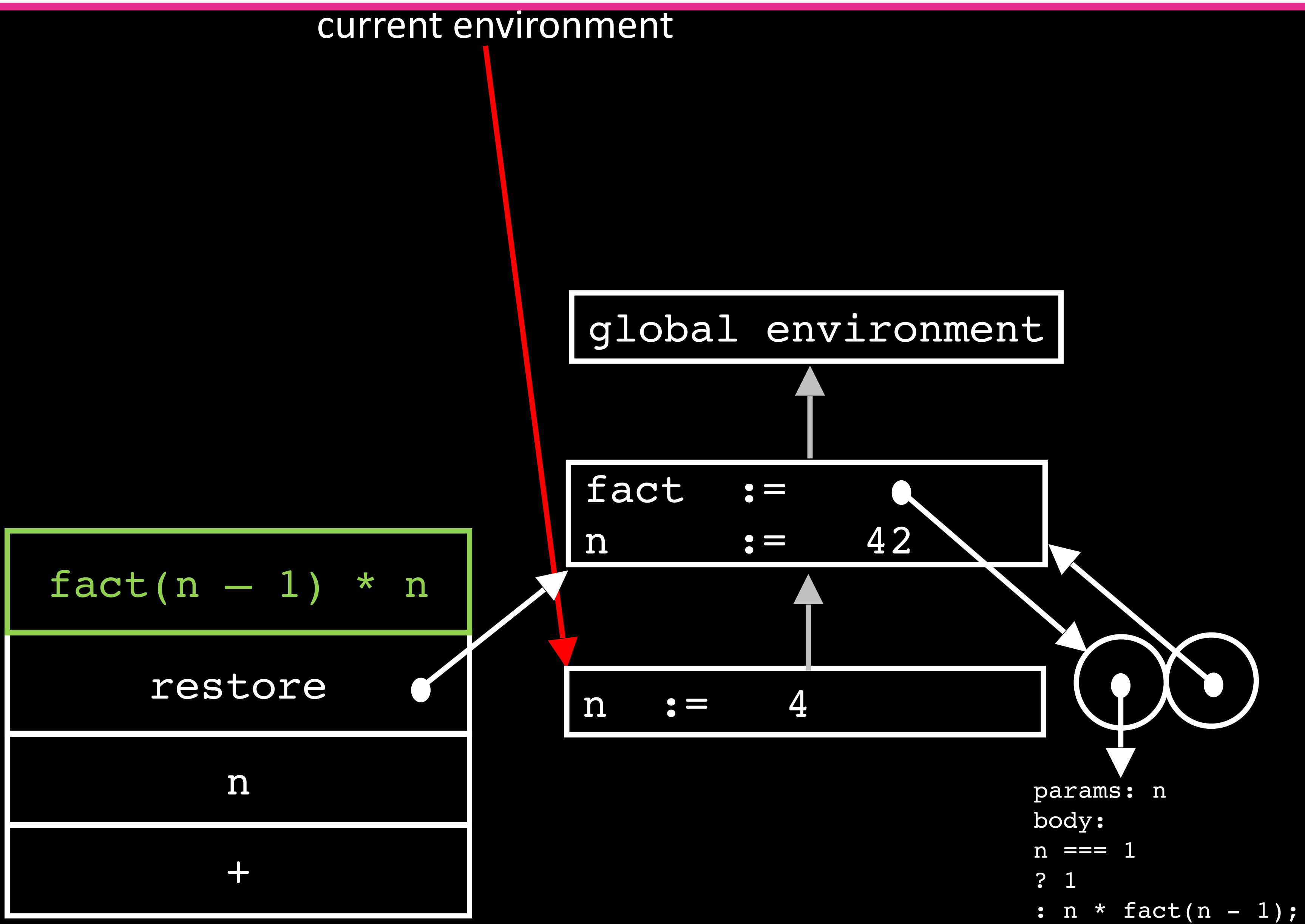
Call instruction:

- pop arguments and function from stash
- extend **function's** env using parameters
- assign parameters to args
- pop call instr from agenda
- push instruction to **restore** current environment
- push body on agenda
- reassign current environment



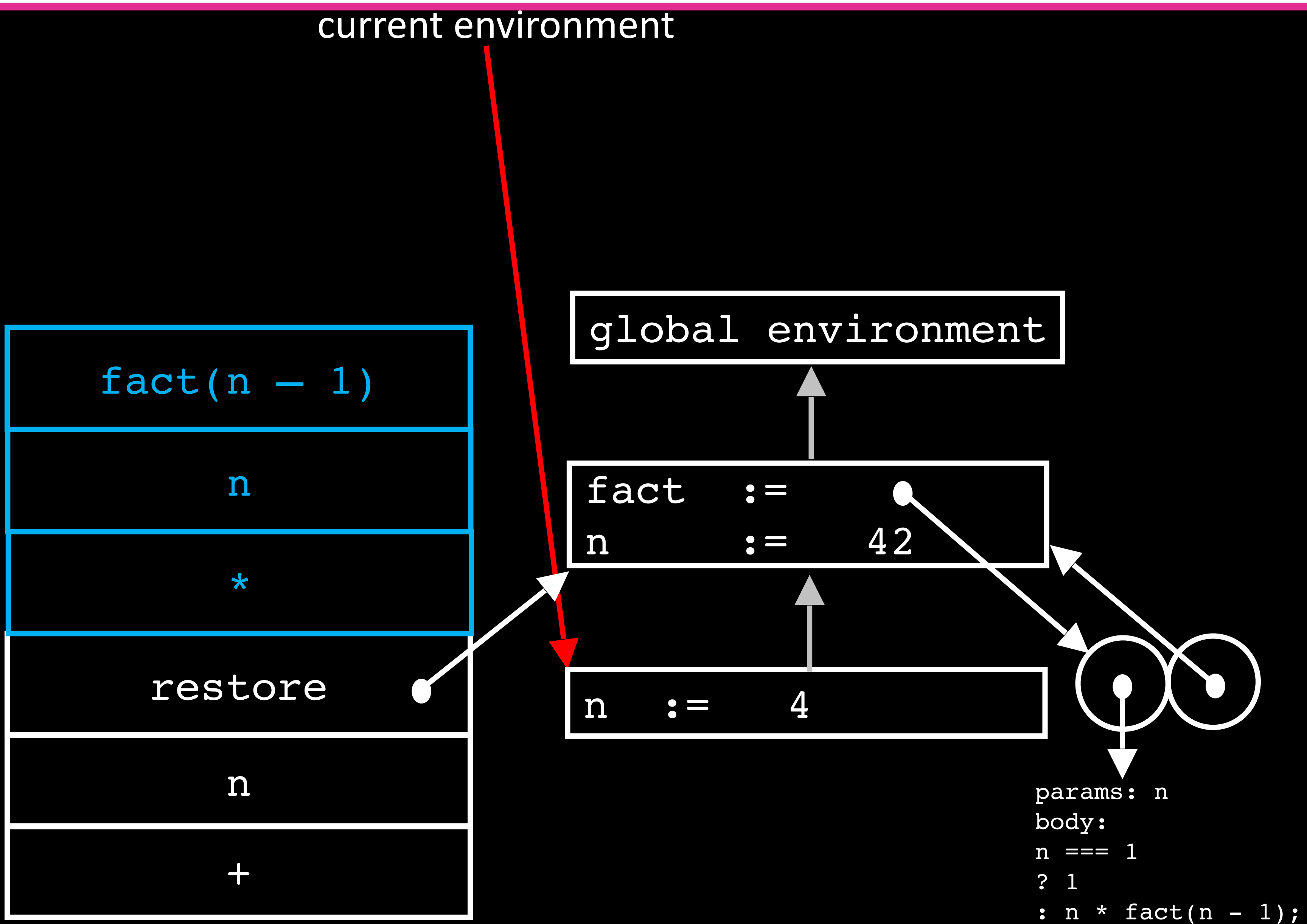
Call instruction:
- pop arguments and function from stash
- extend **function's** env using parameters
- assign parameters to args
- pop call instr from agenda
- push instruction to **restore** current environment
- push body on agenda
- reassign current environment

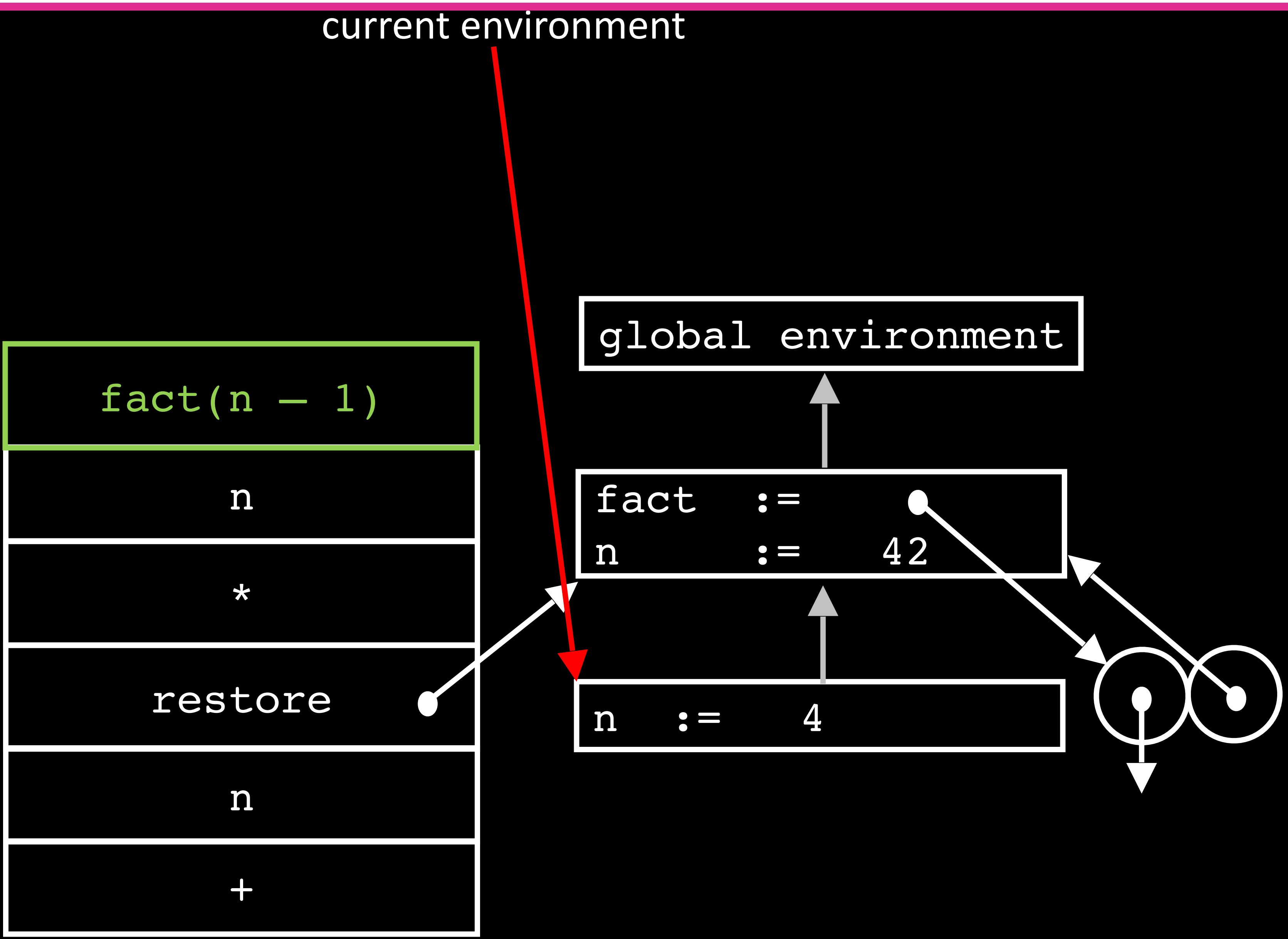
...after a while...

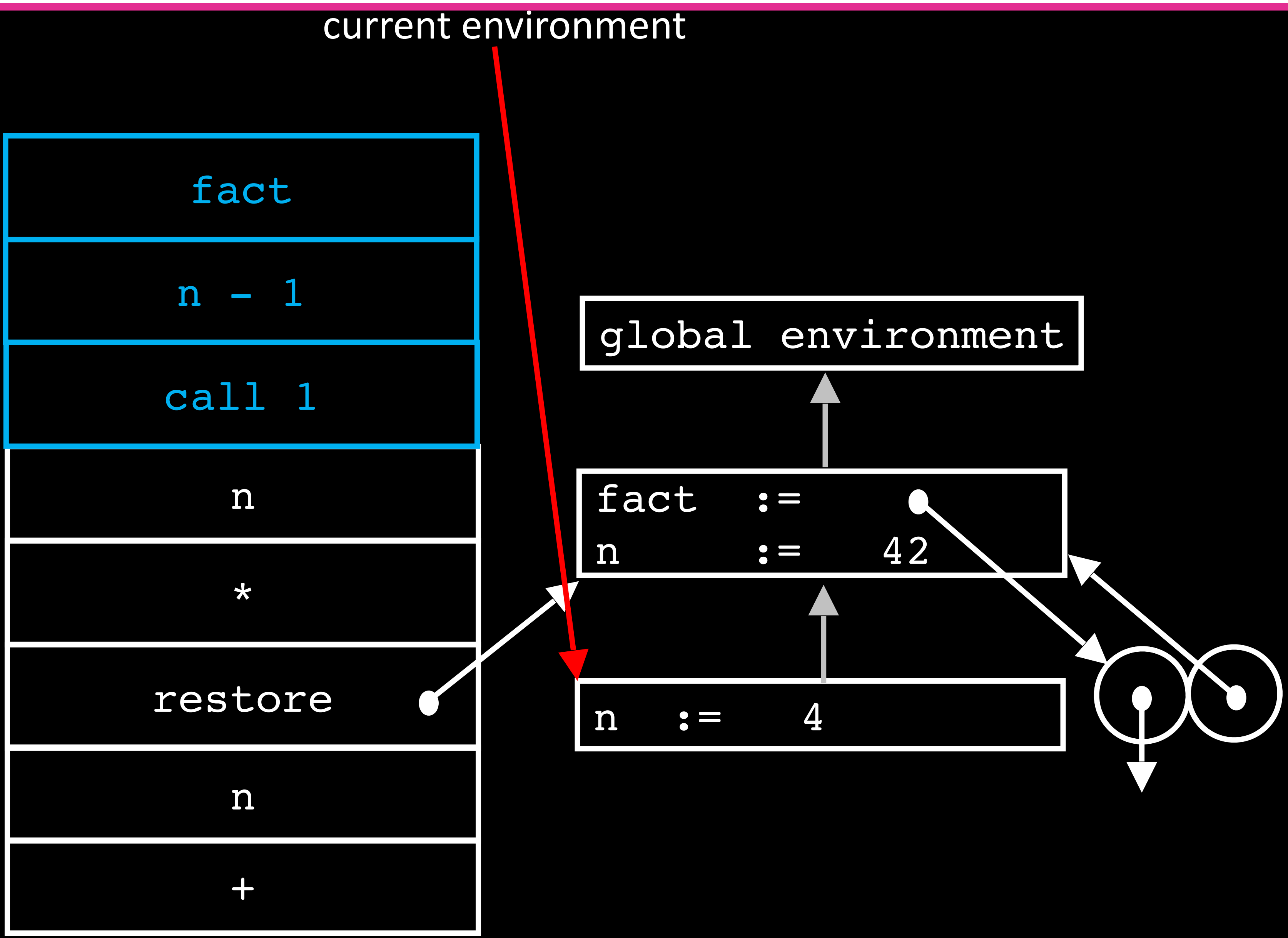


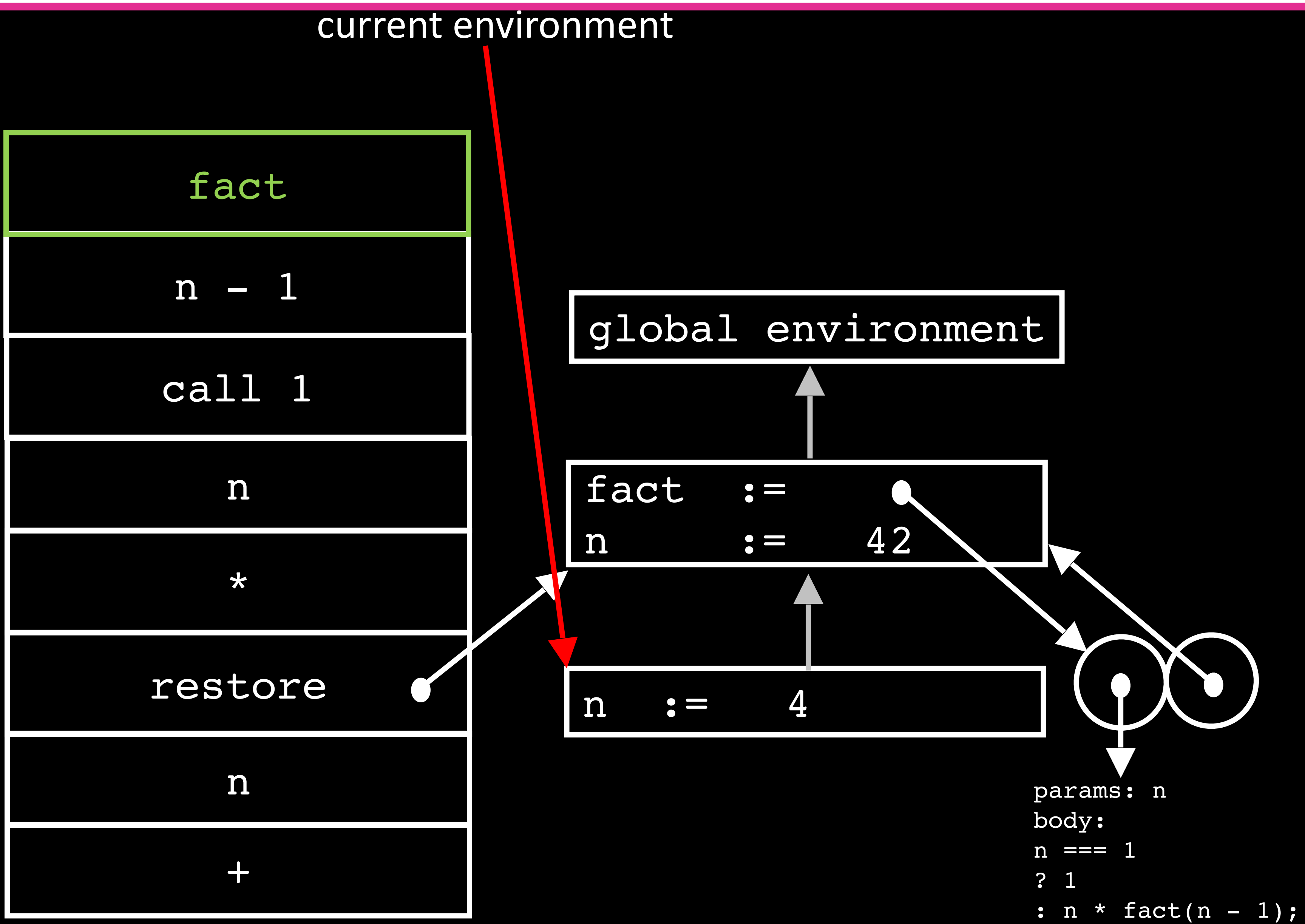
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

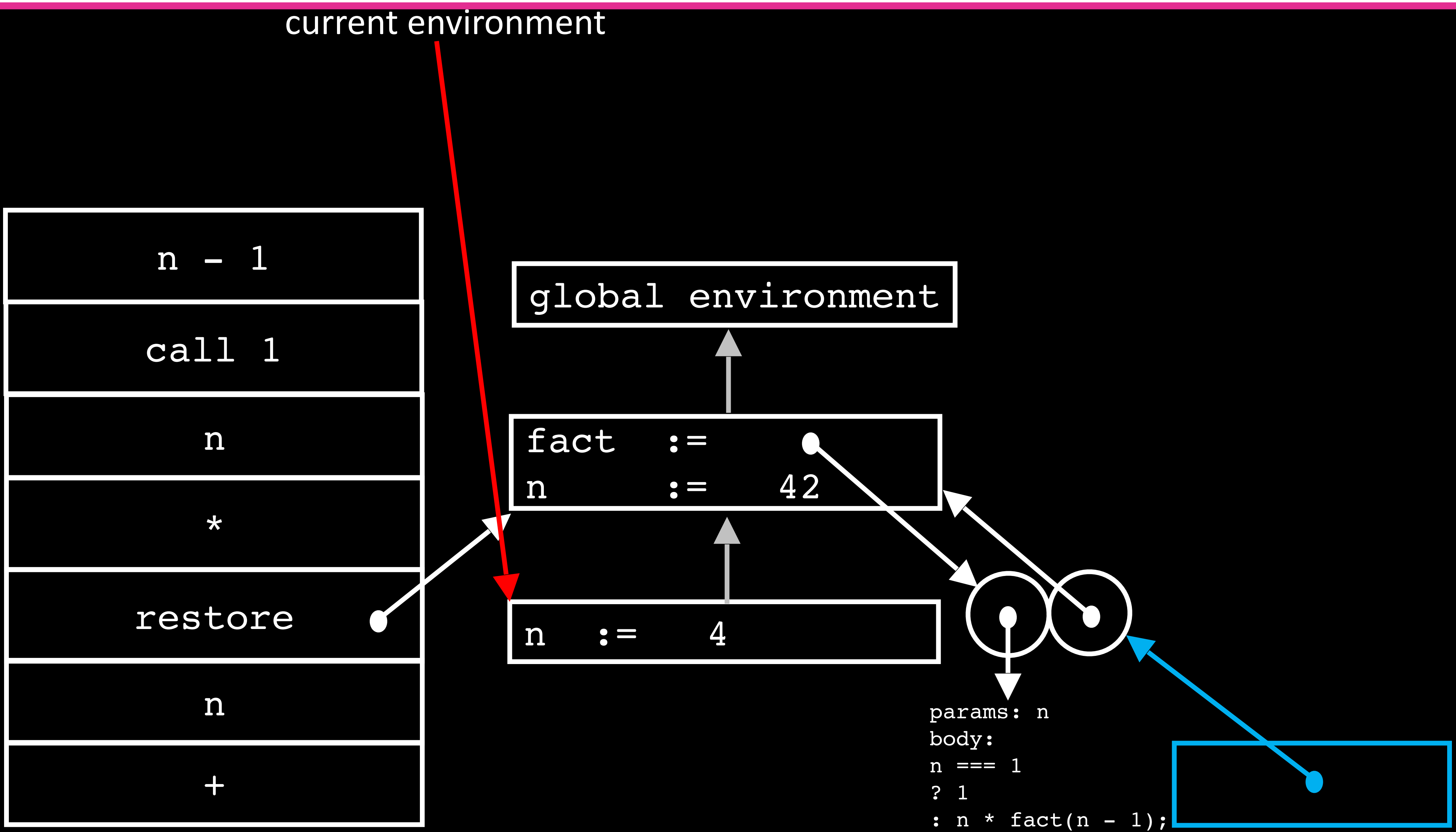
…after a while…

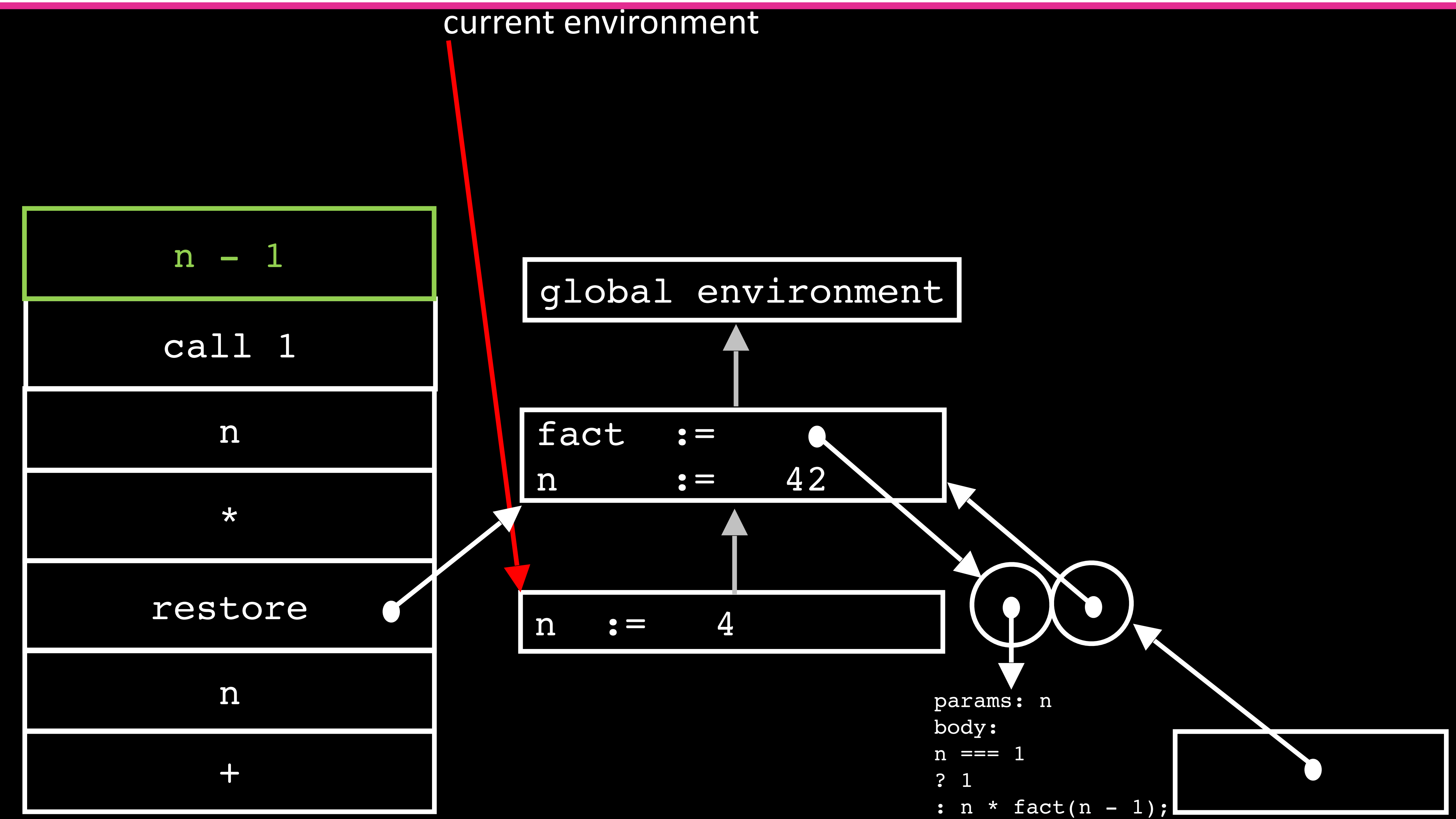


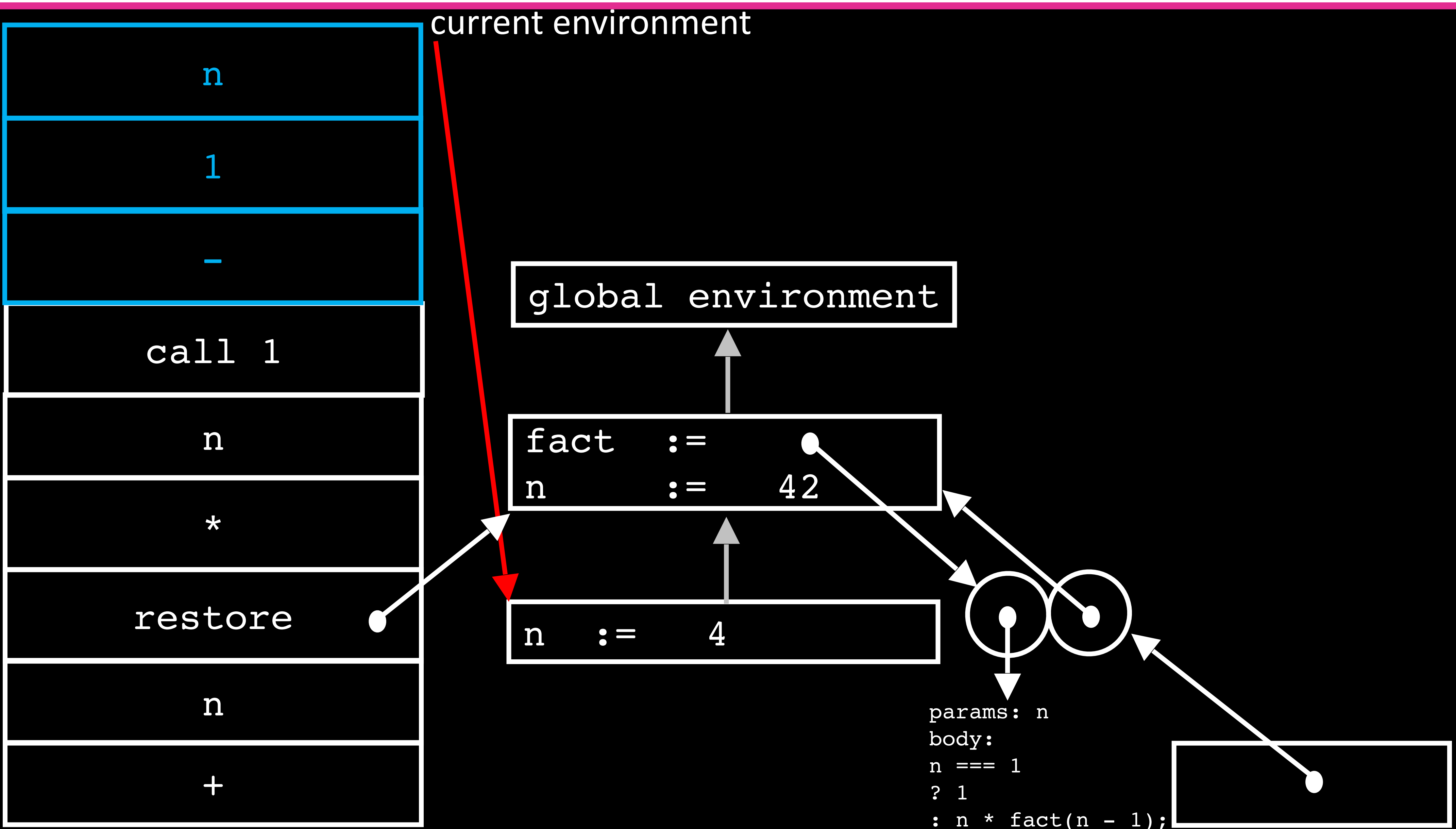

current environment
fact(n – 1)
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4

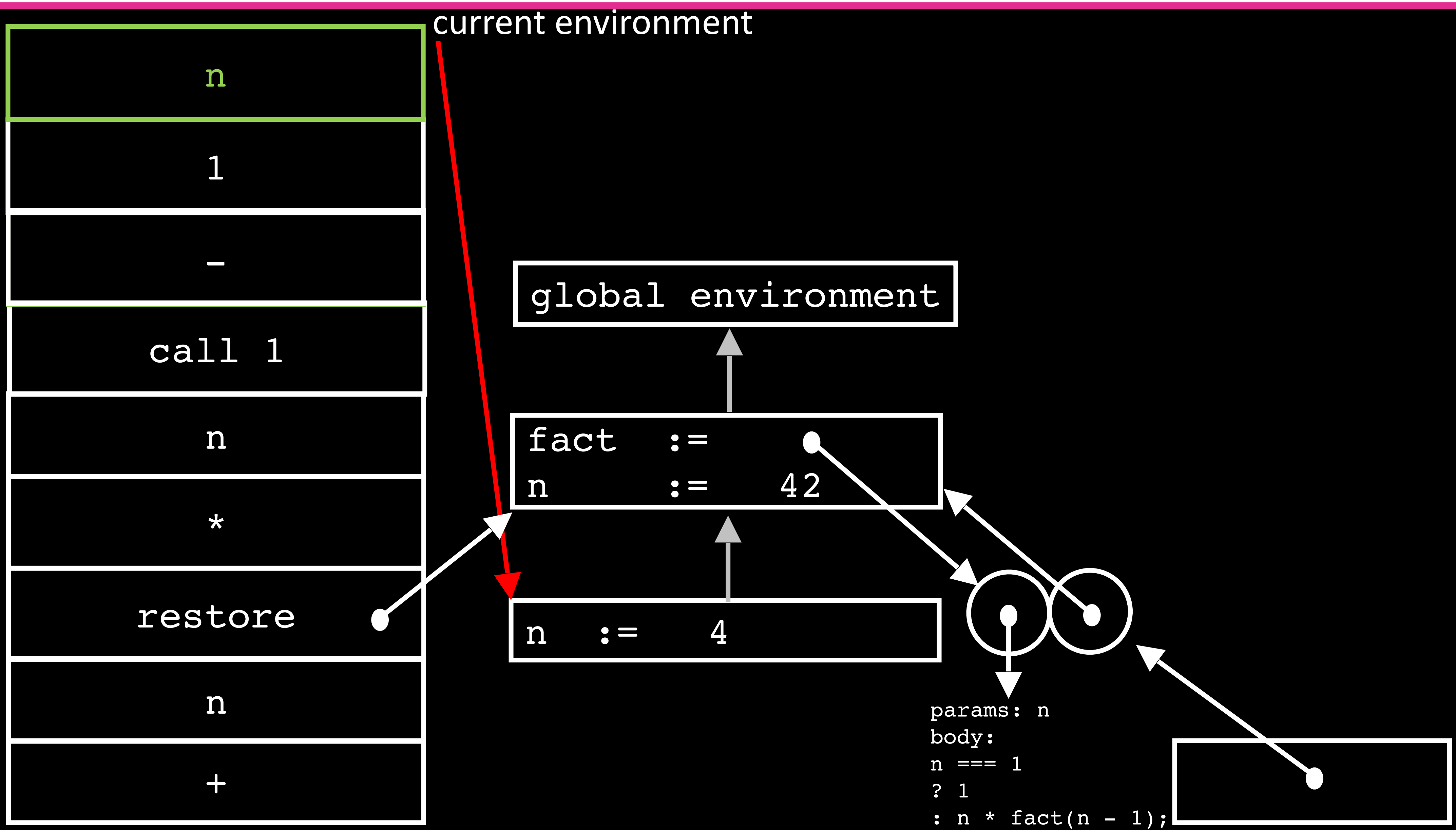

current environment
fact
n - 1
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4

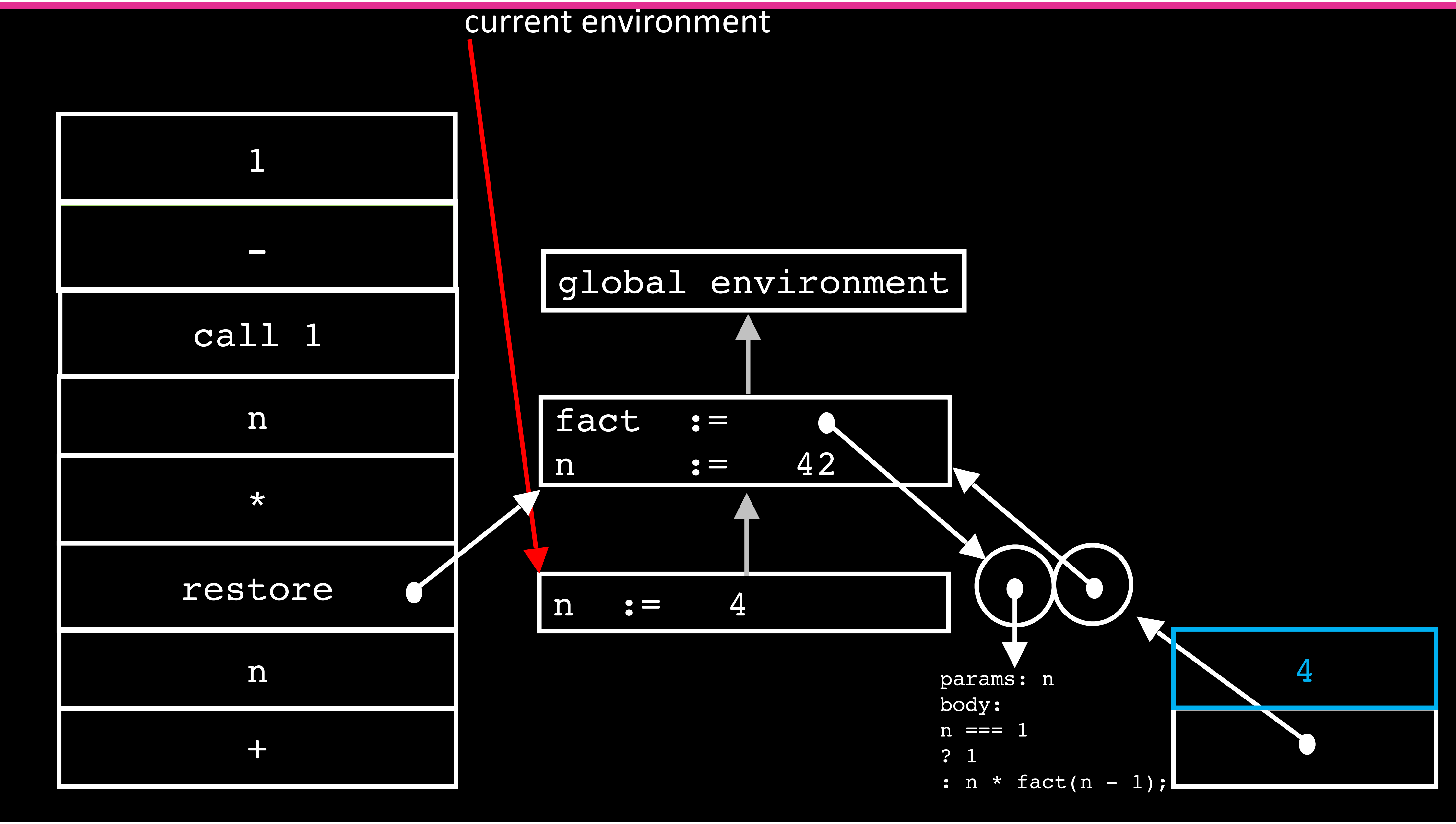

current environment
fact
n - 1
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);


current environment
n - 1
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);

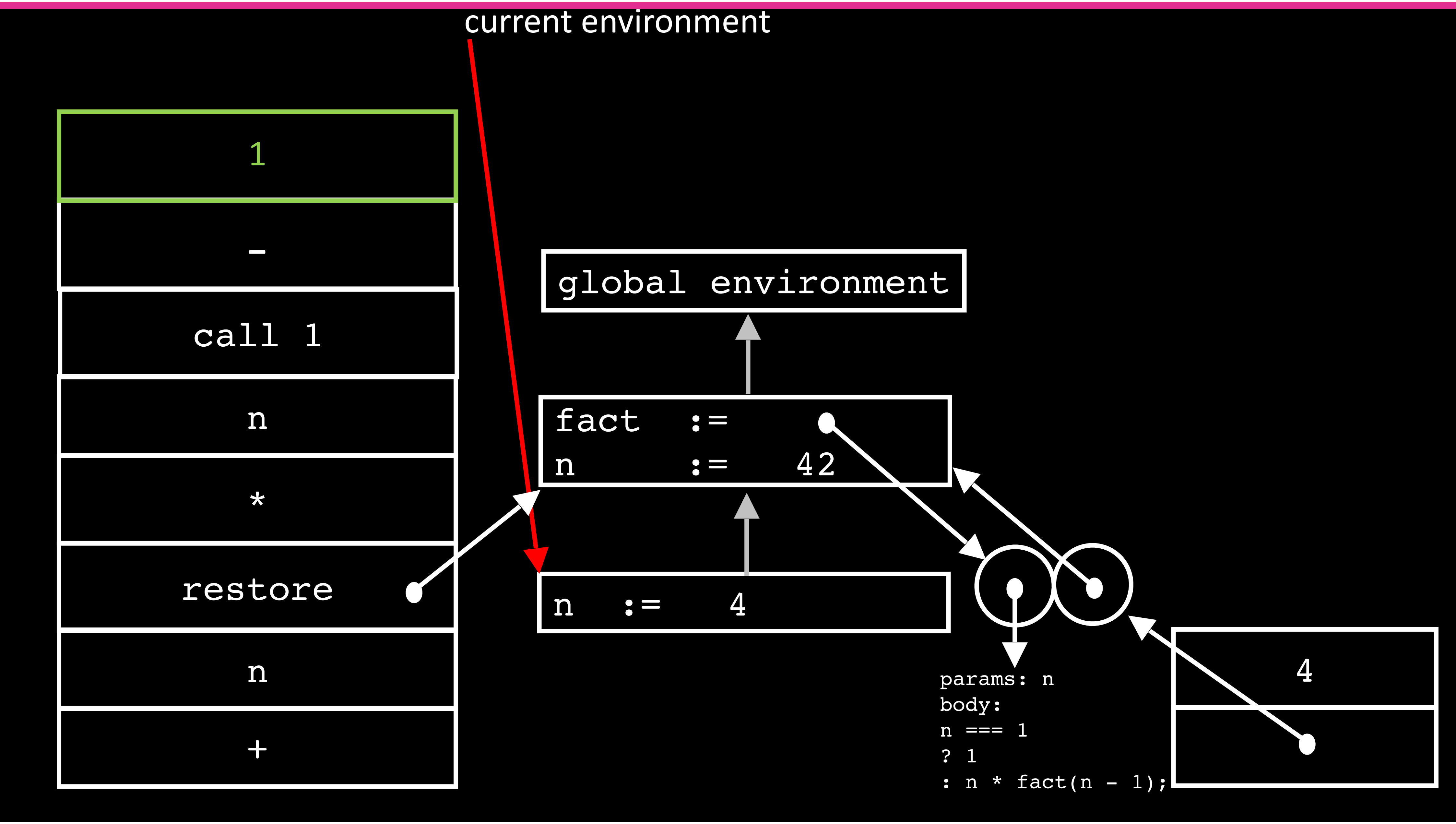

current environment
n - 1
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);

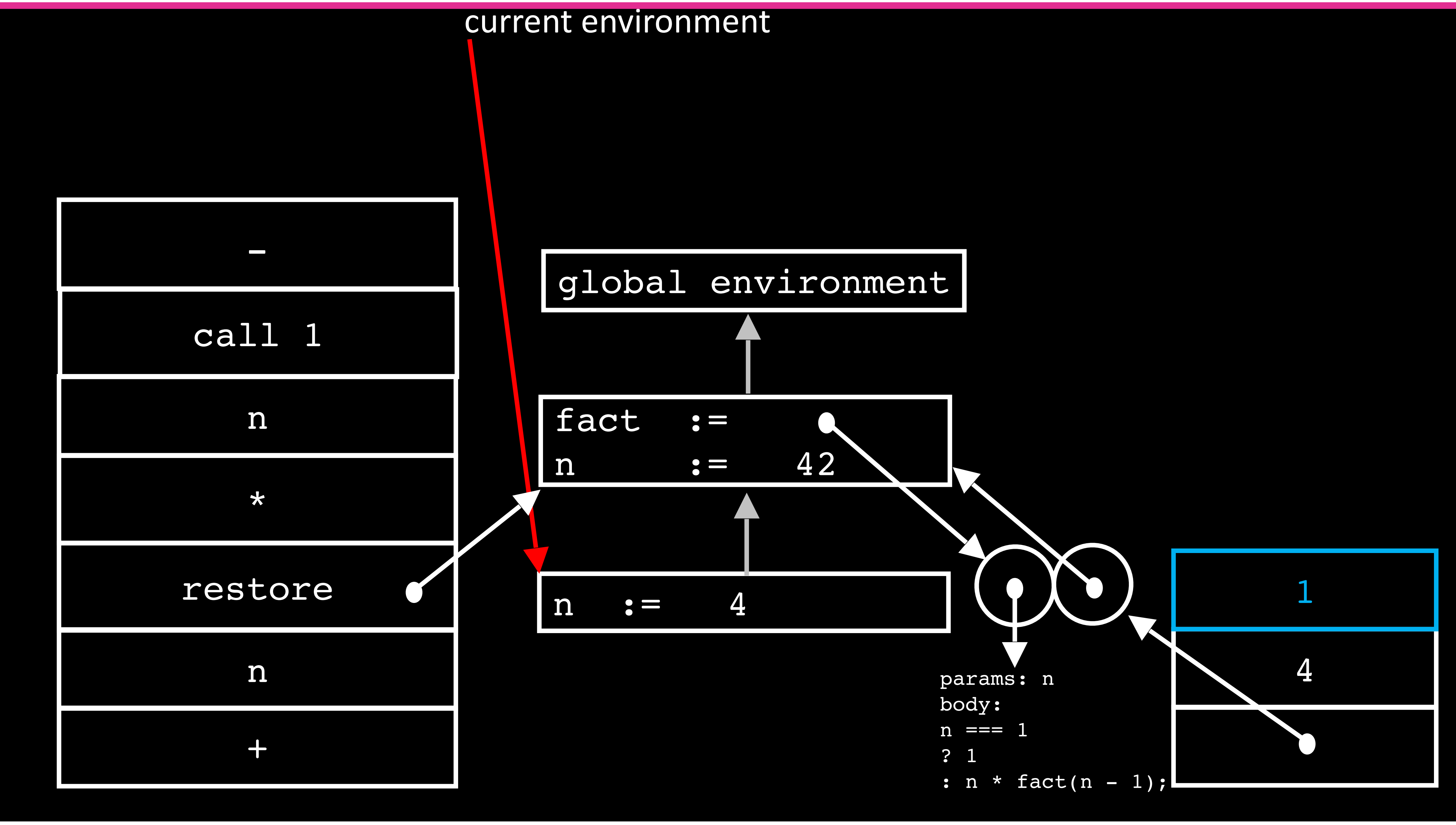

current environment
n
1
-
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);


current environment
n
1
-
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);

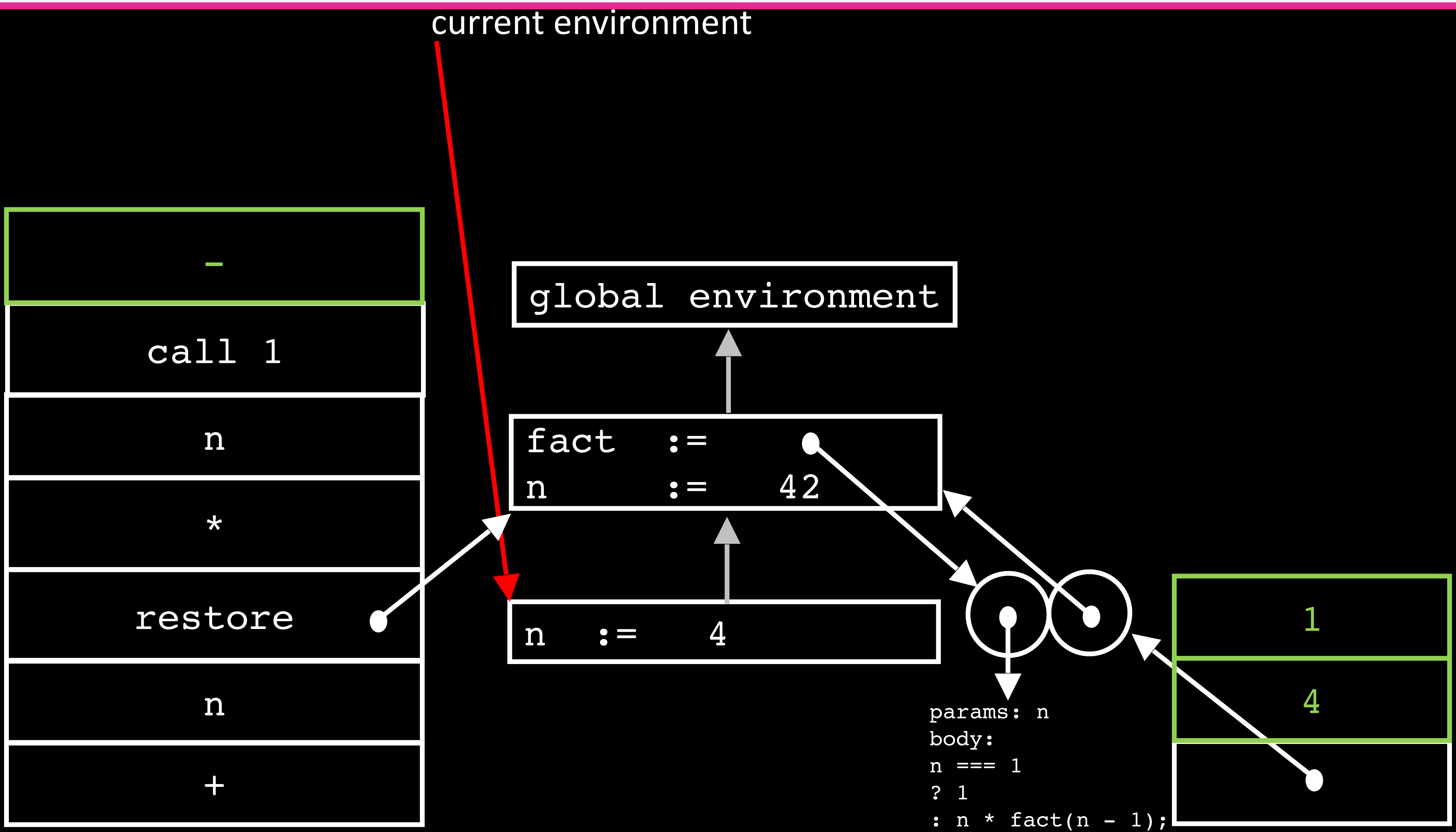


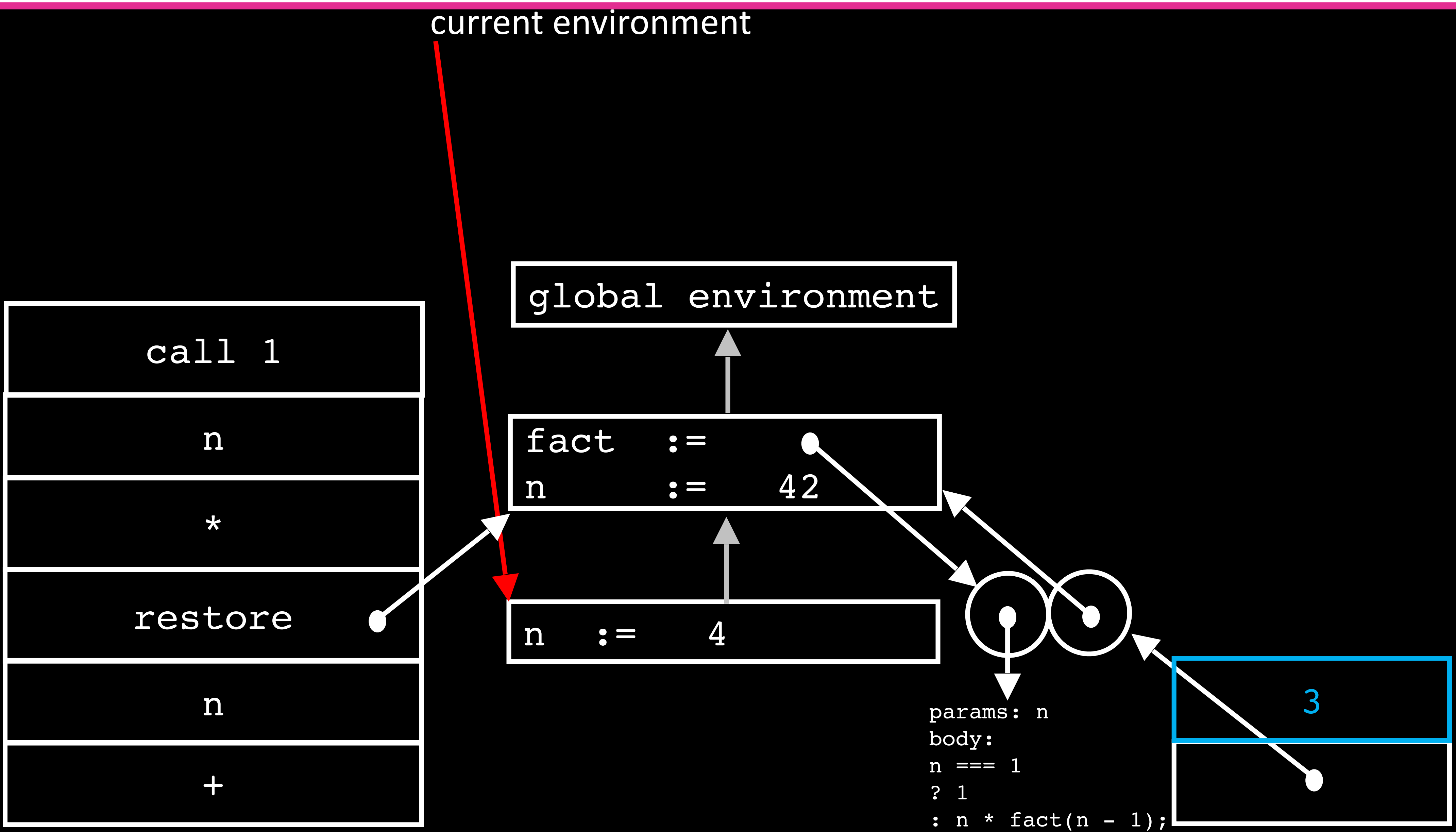
current environment
1
-
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
4


current environment
1
-
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
4

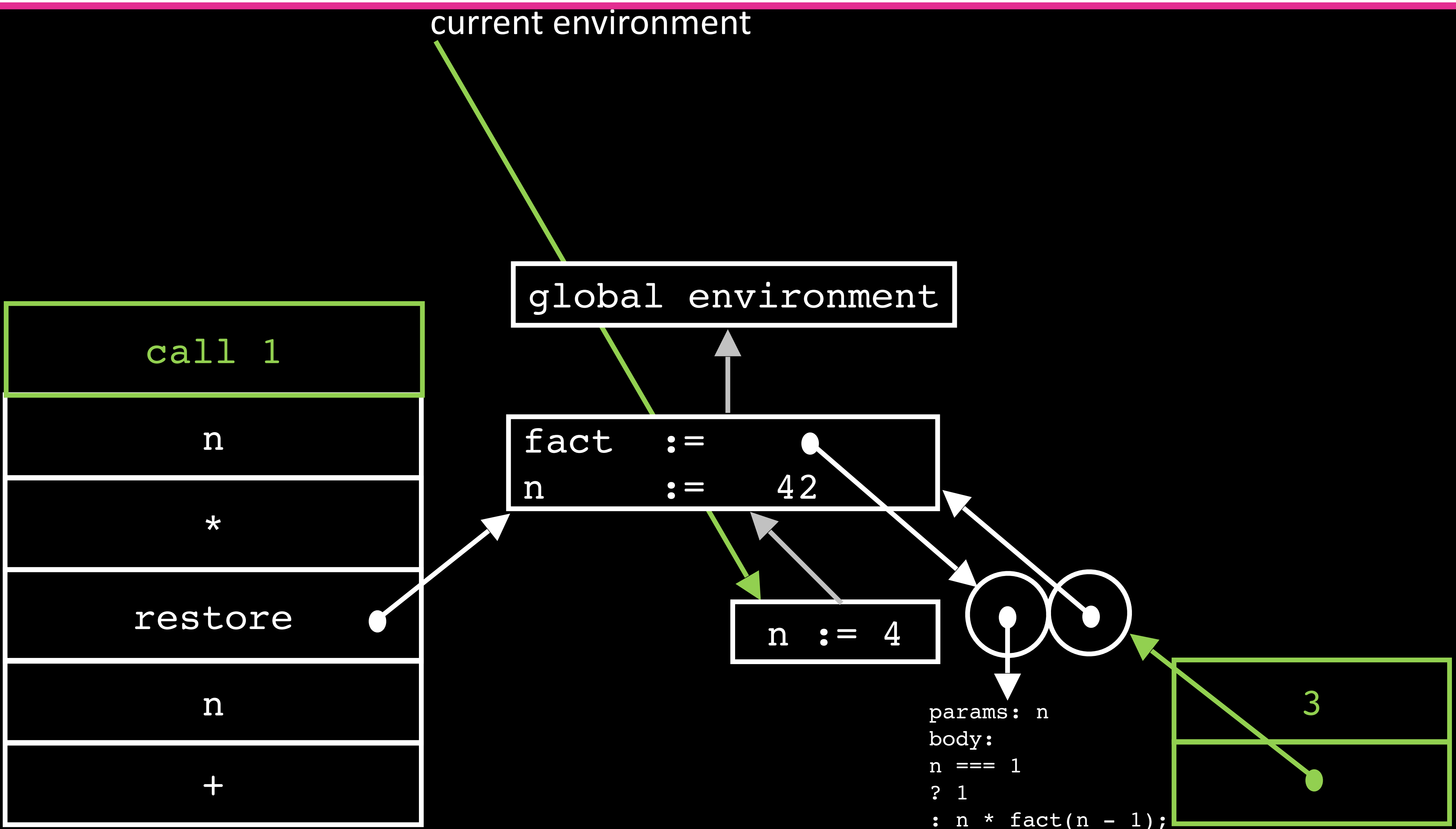


current environment
-
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
1
4

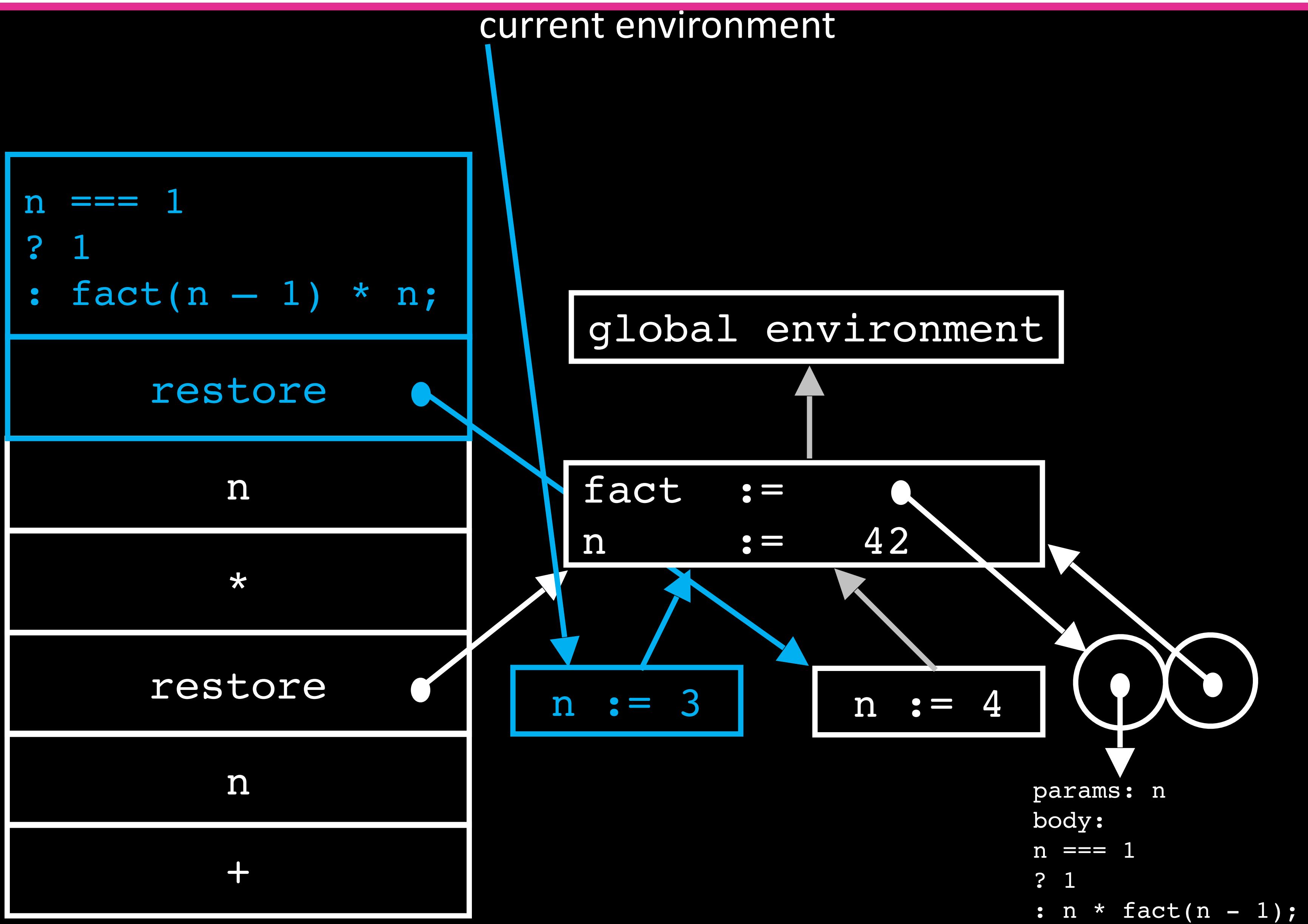

current environment
-
call 1
n
*
restore
n
+
global environment
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
1
4



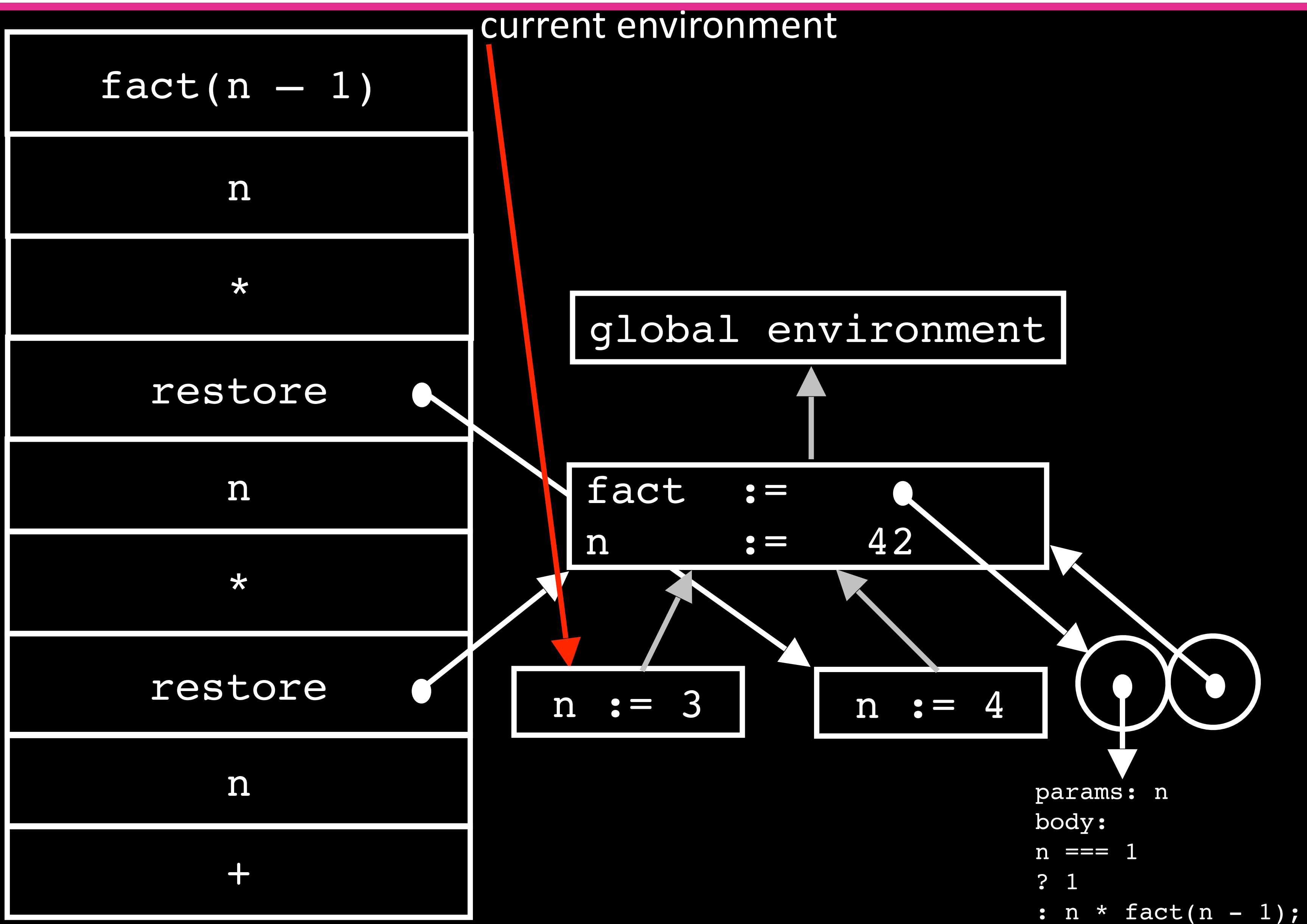
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



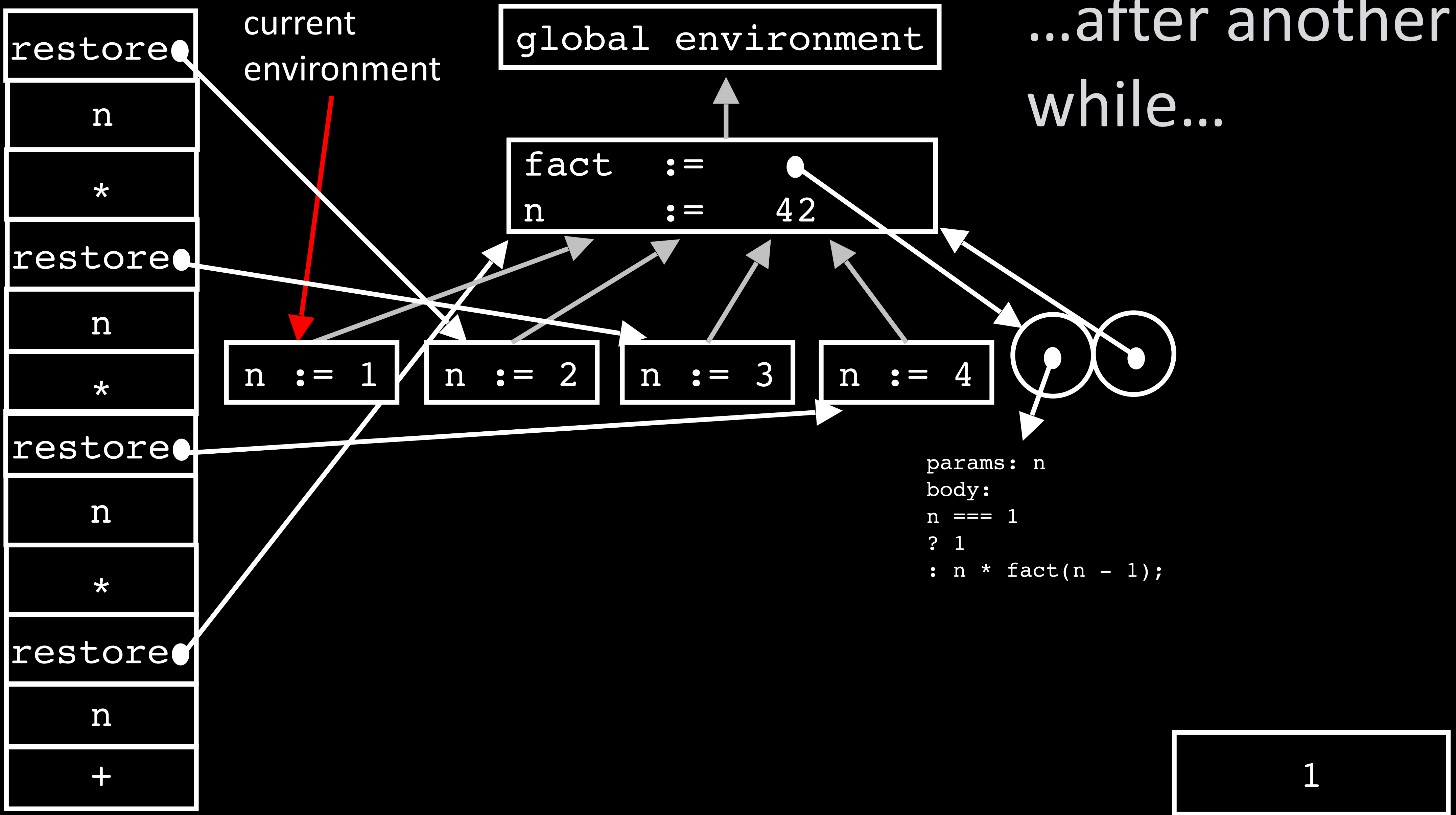
current environment
global environment
call 1
n
*
restore
n
+
fact :=
n := 42
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
3


current environment
n === 1
? 1
: fact(n – 1) * n;
restore
n
*
restore
n
+
global environment
fact :=
n := 42
n := 3
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);

…after a while…

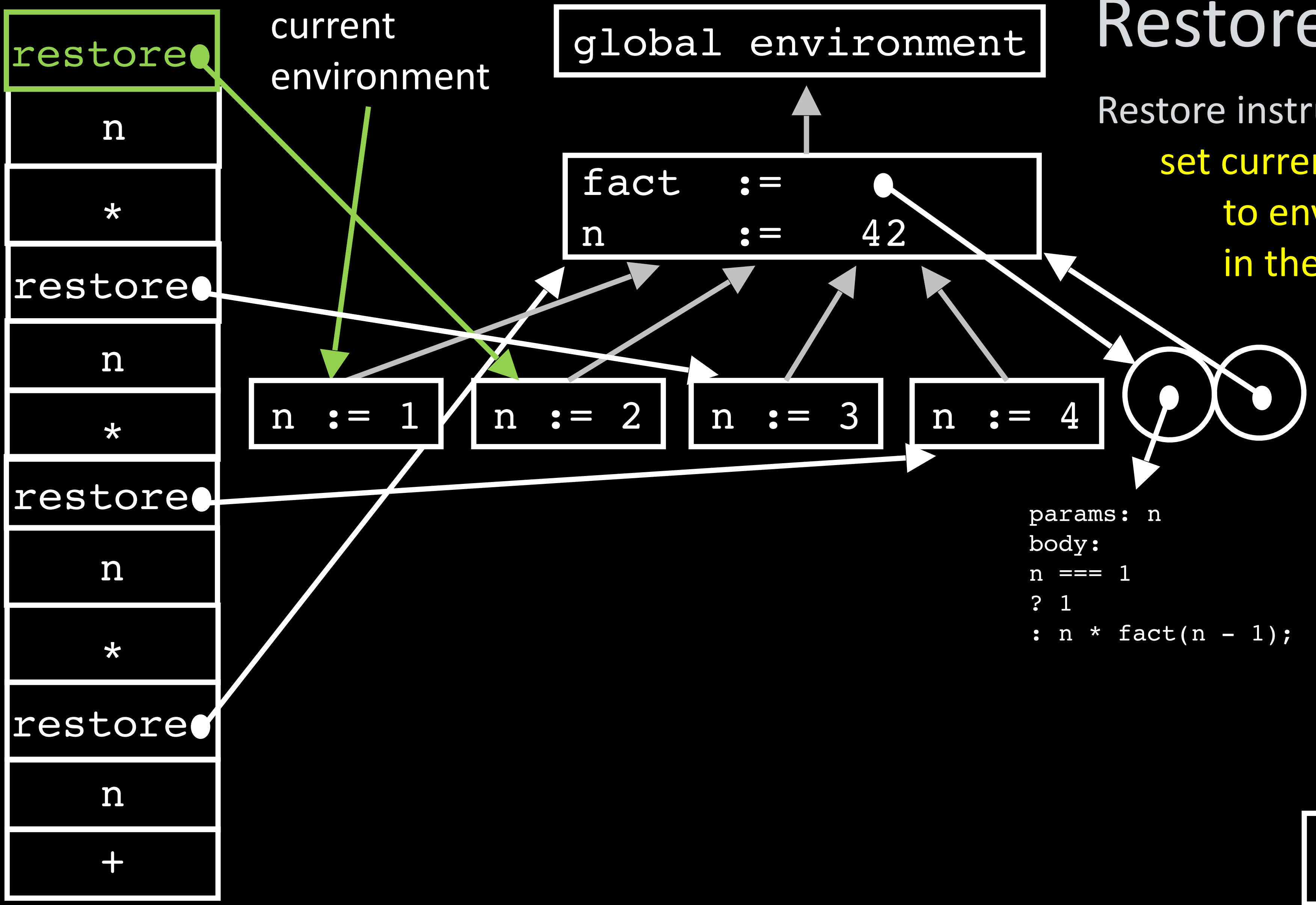


…after another while…



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Restore instr.

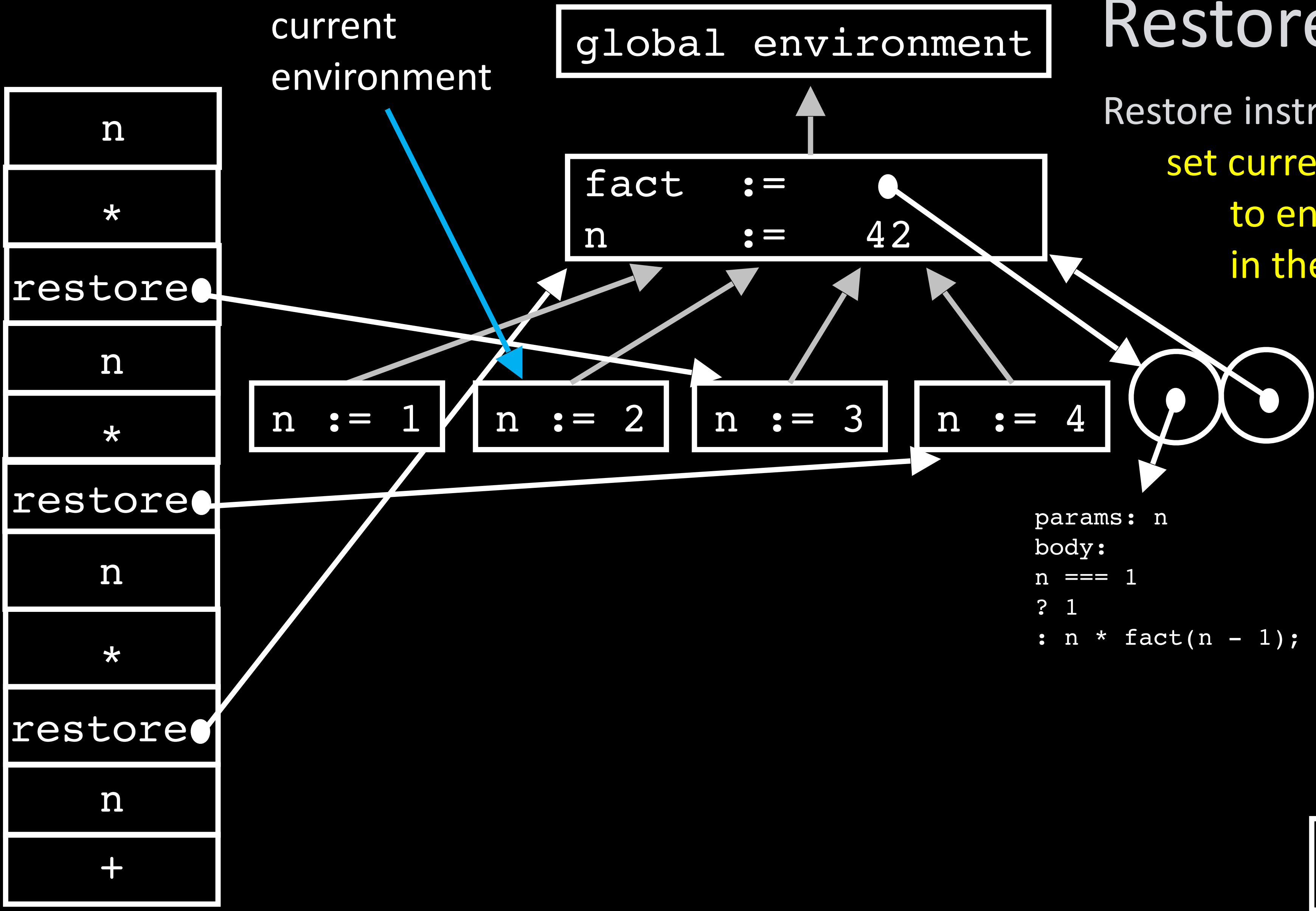


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Restore instruction:

set current environment
to environment stored
in the instruction

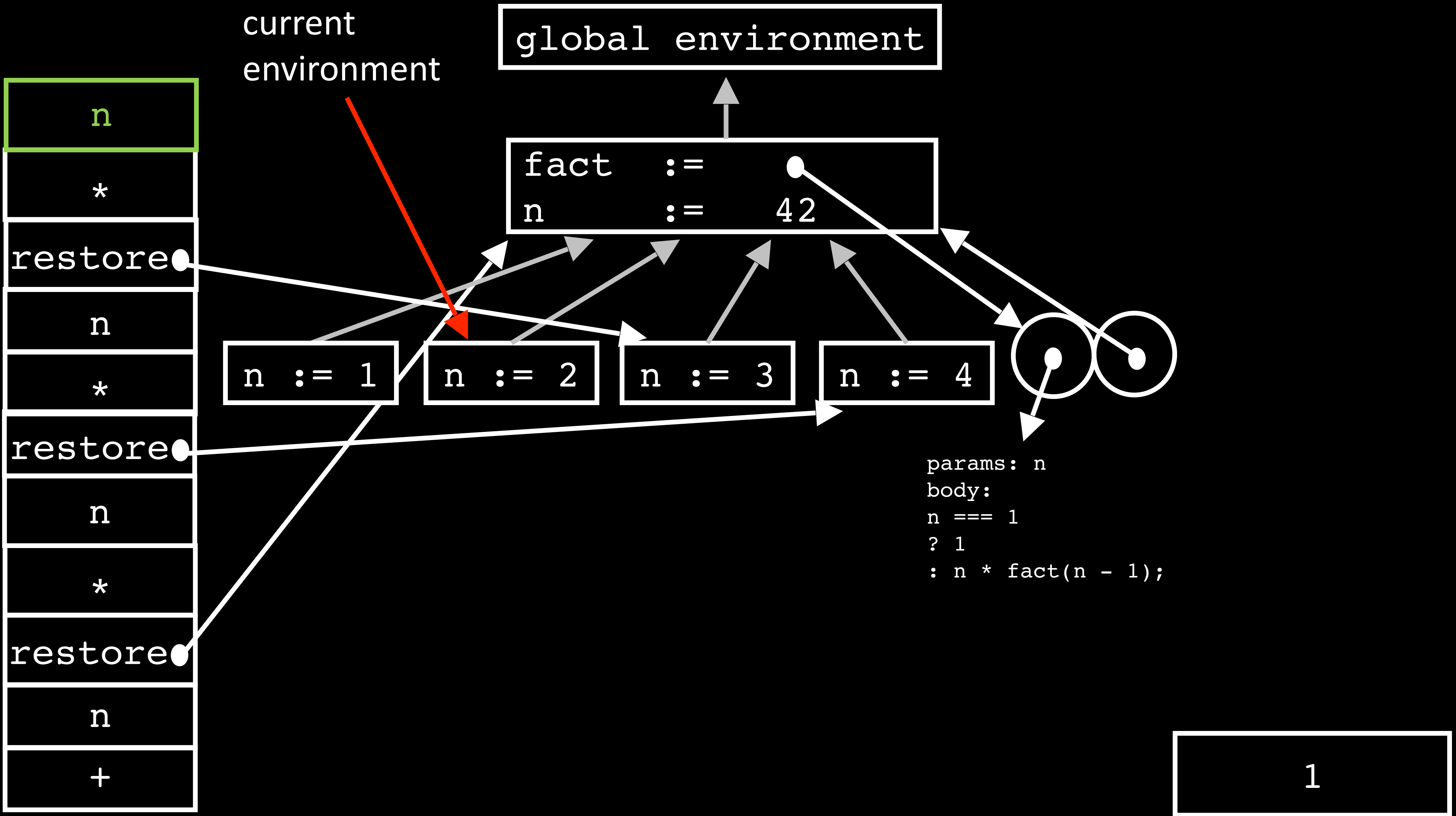
Restore instr.



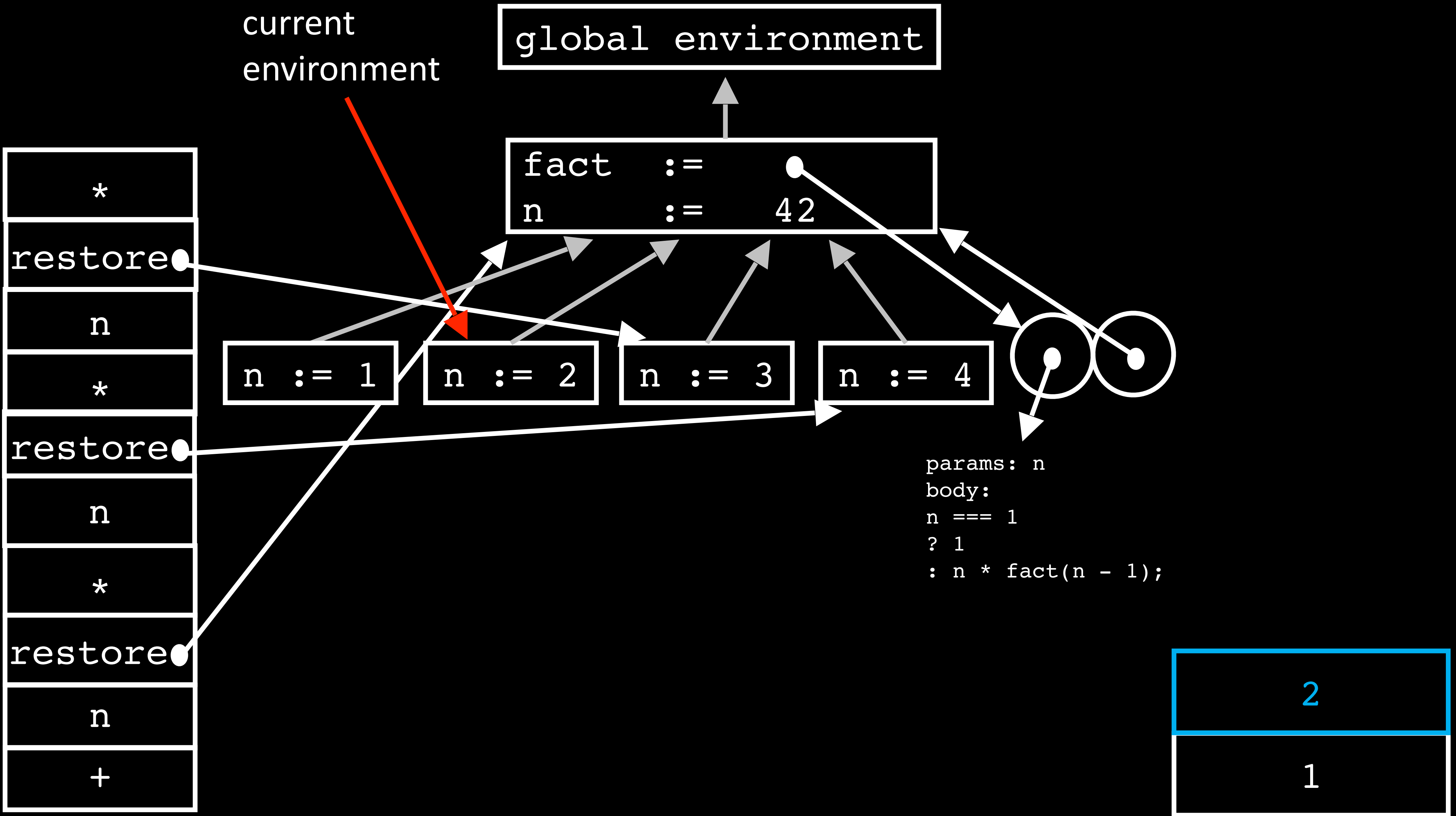
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Restore instruction:

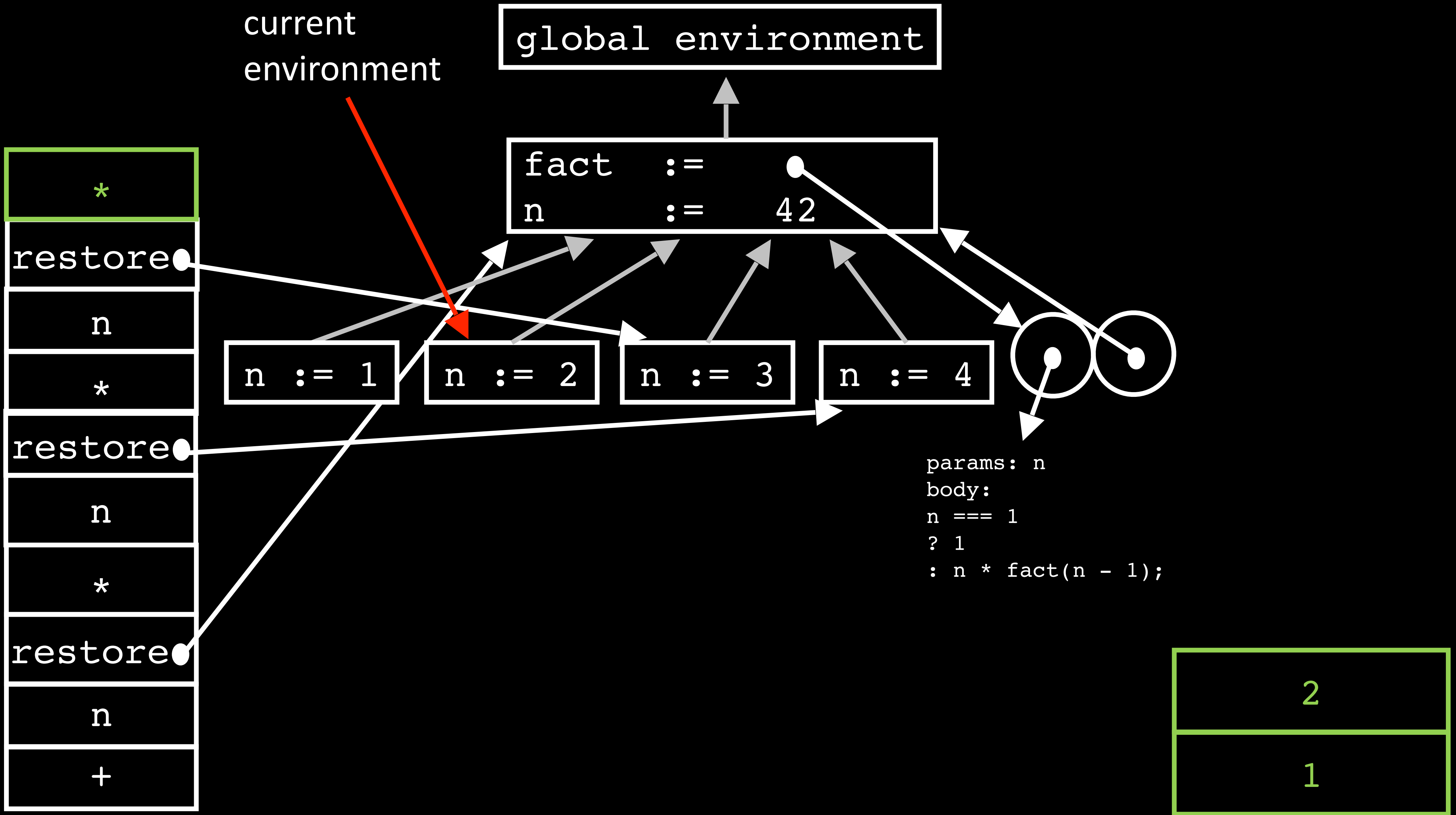
set current environment
to environment stored
in the instruction



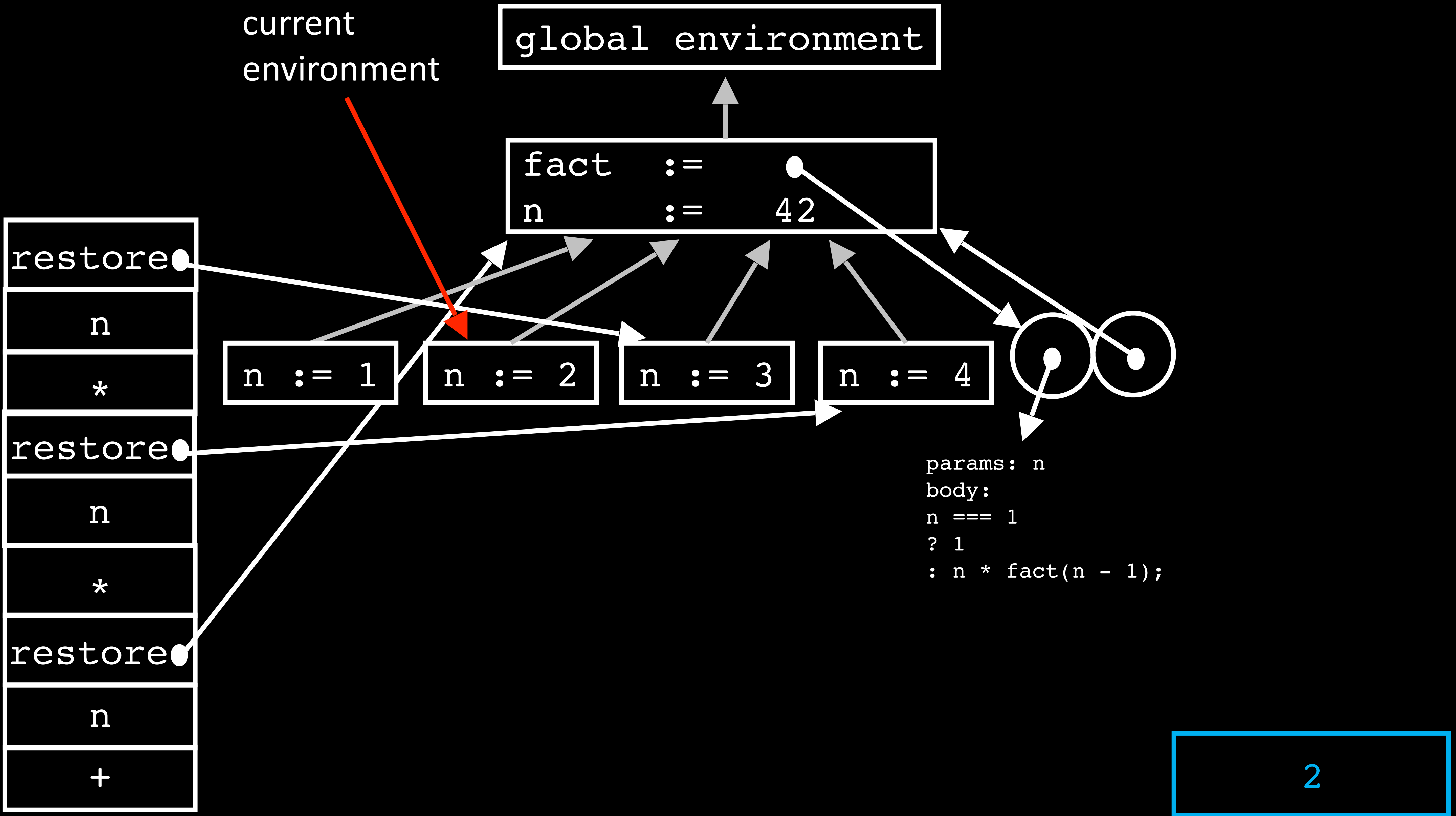
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

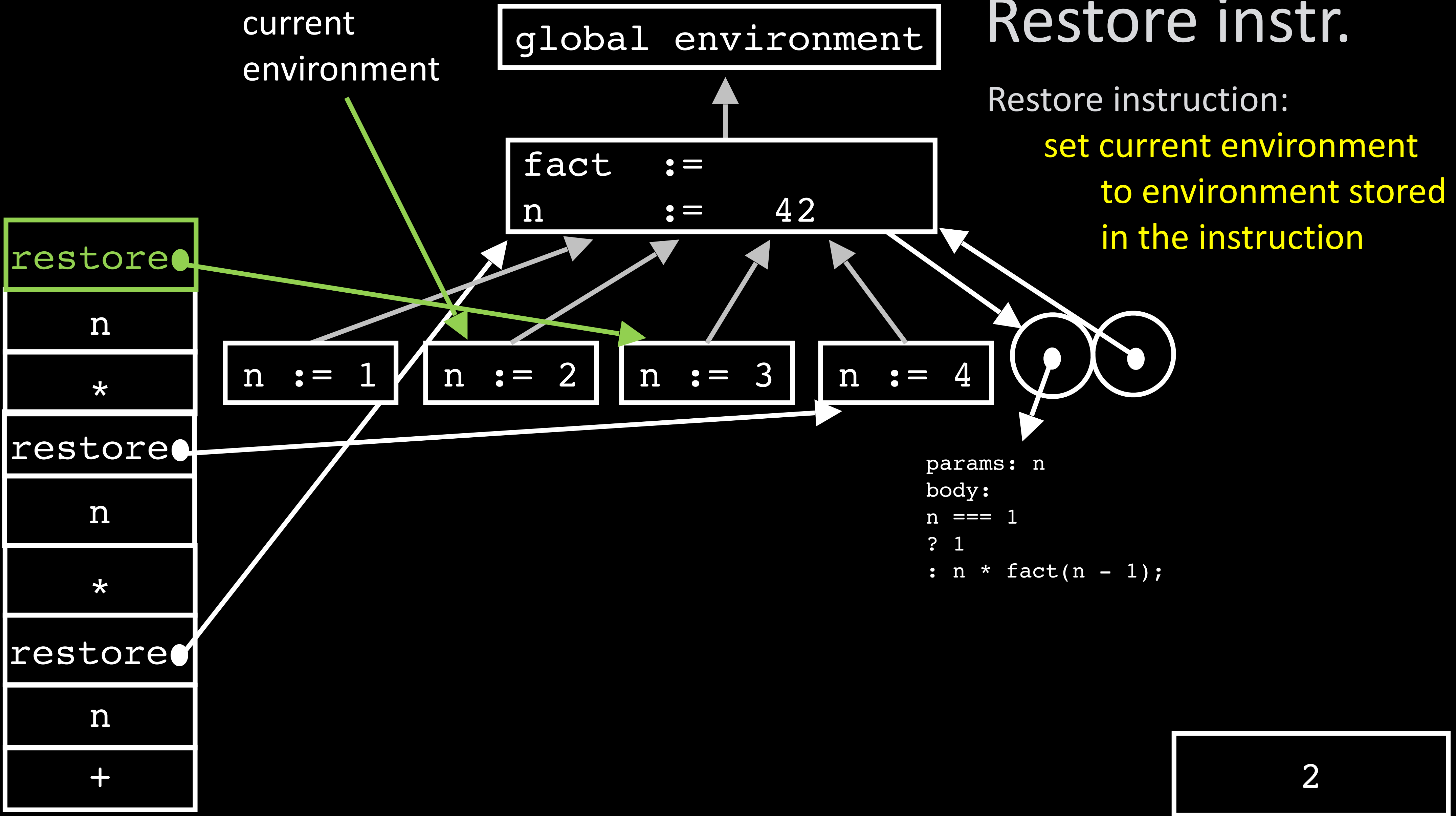


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Restore instr.

Restore instruction:

set current environment to environment stored in the instruction

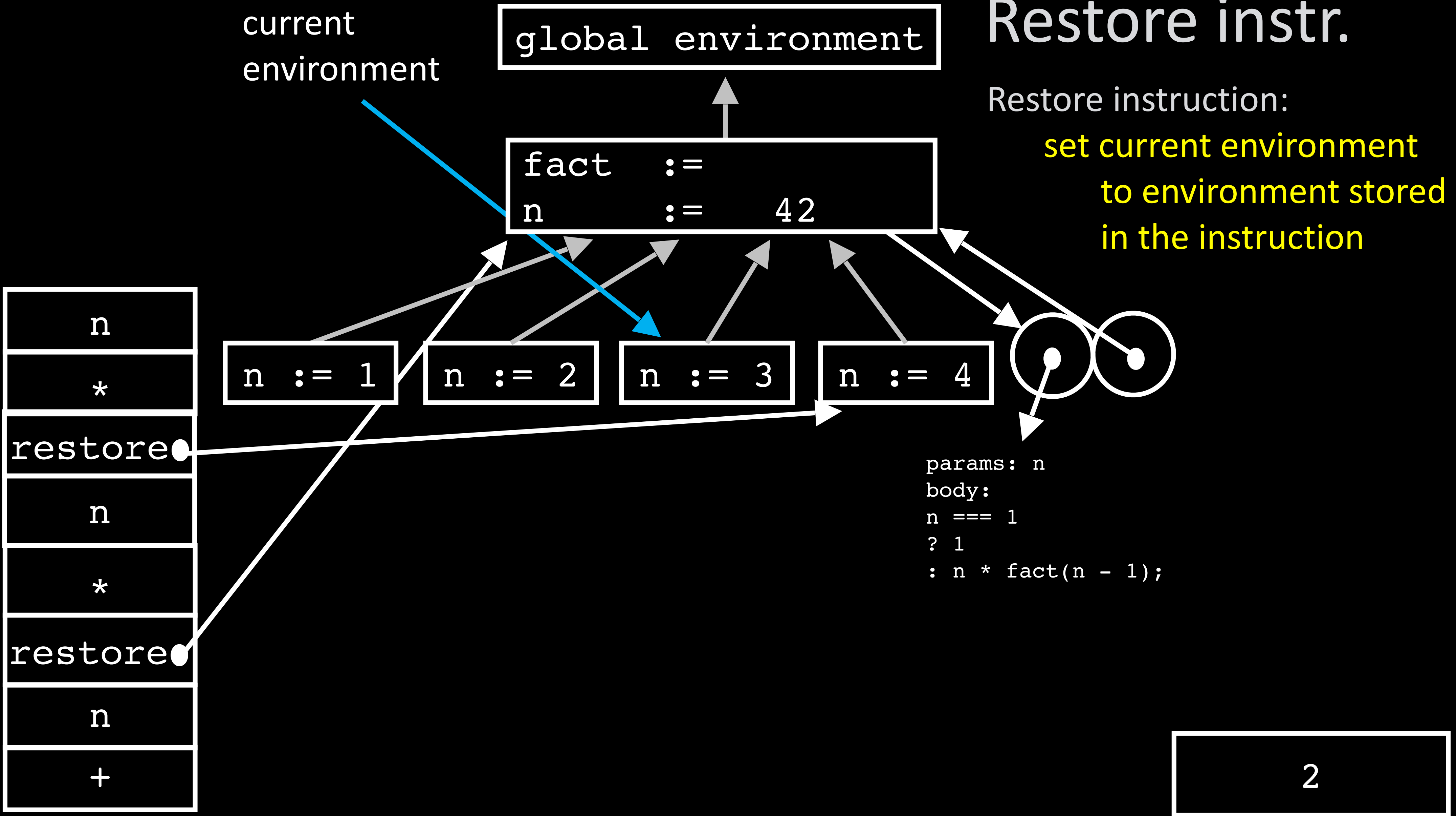


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

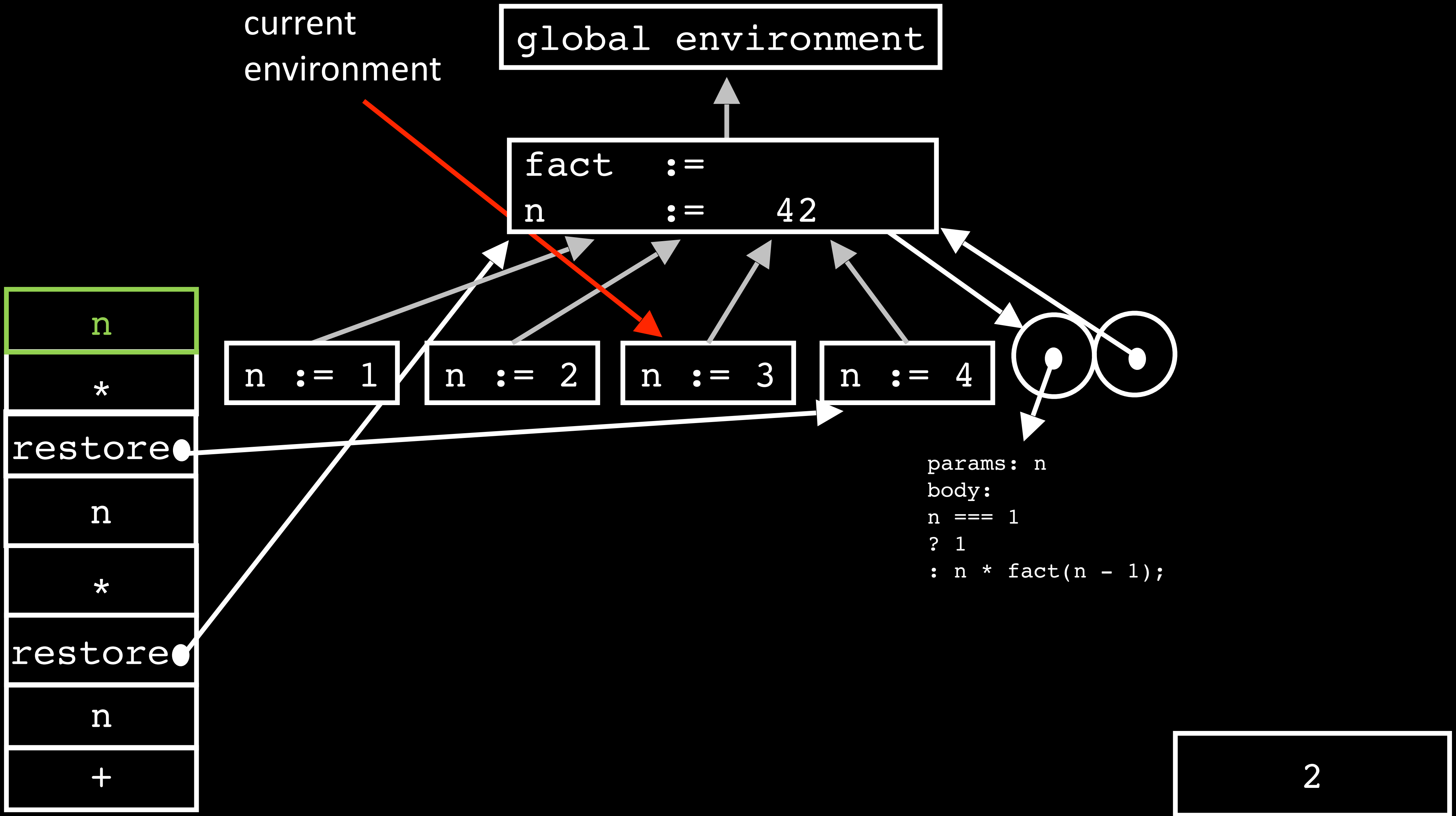
Restore instr.

Restore instruction:

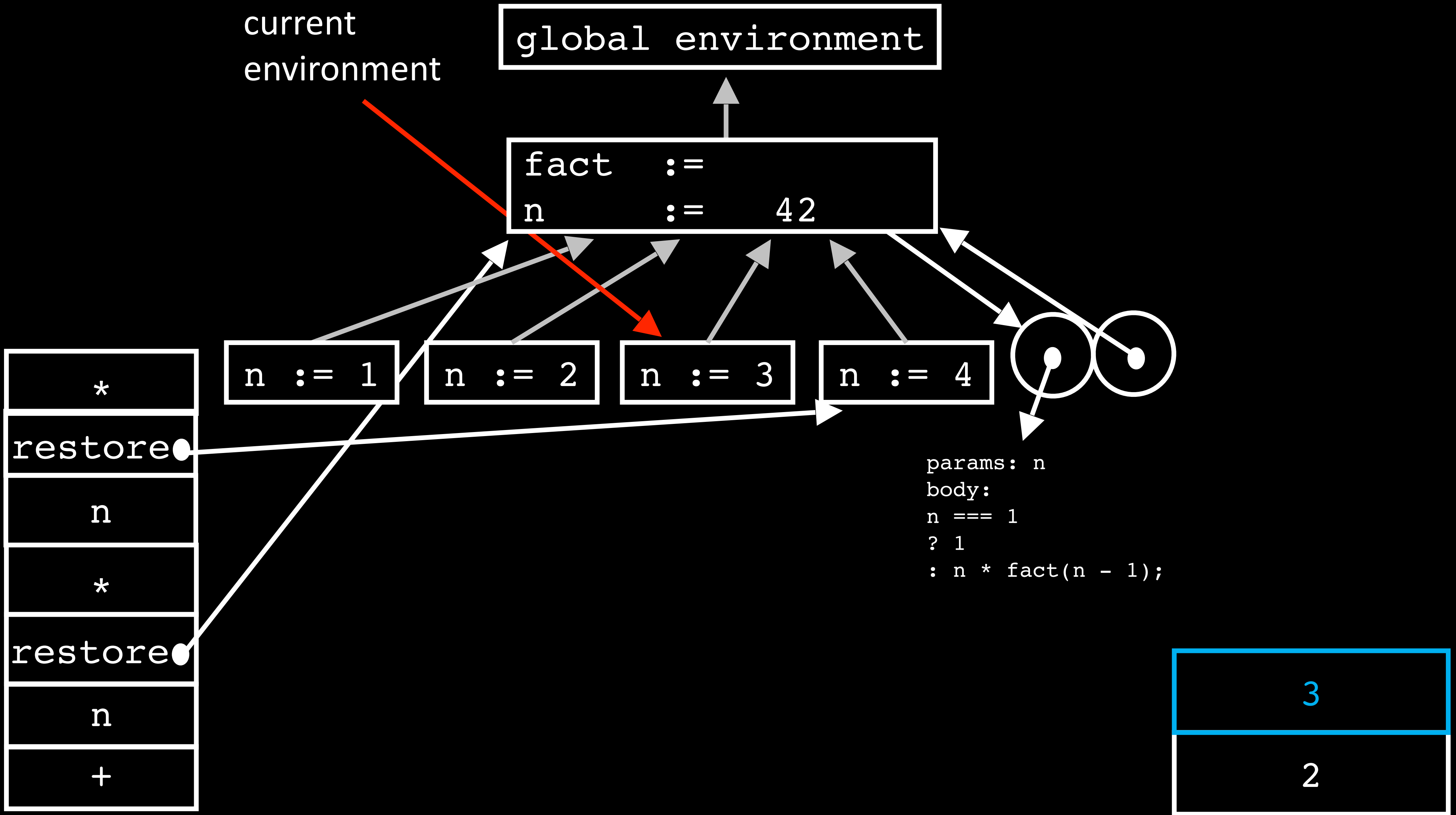
set current environment
to environment stored
in the instruction



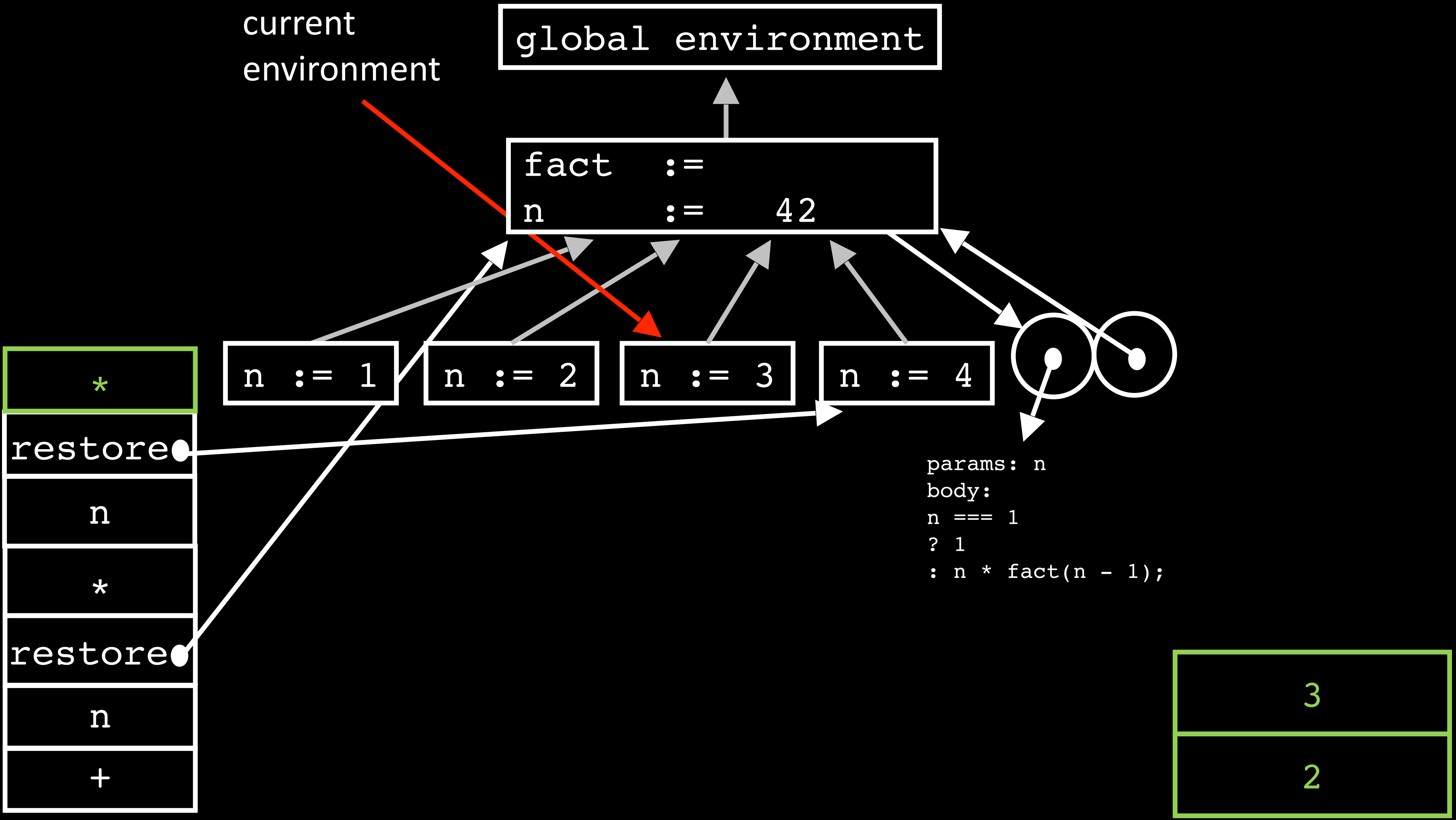
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



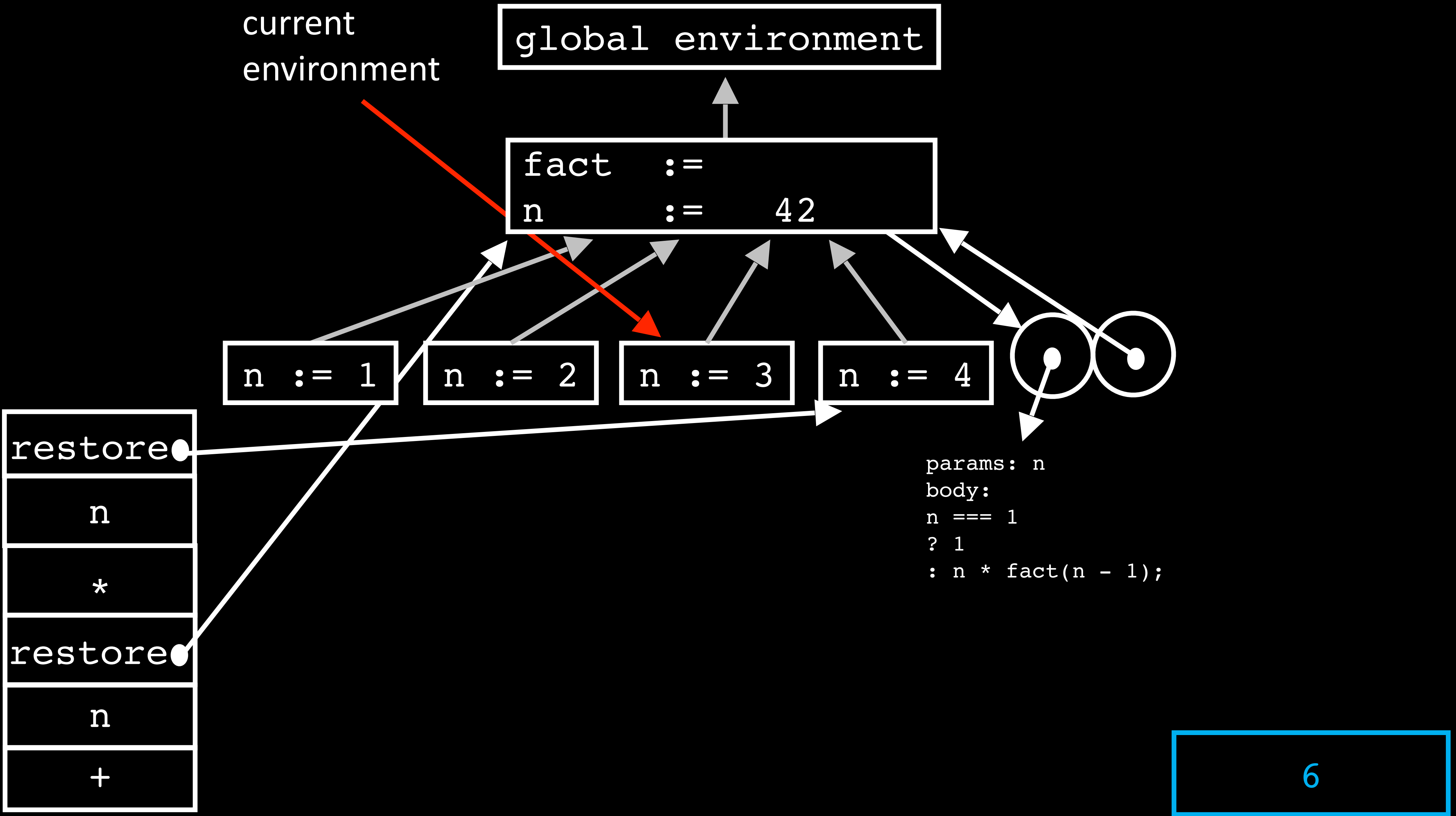
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



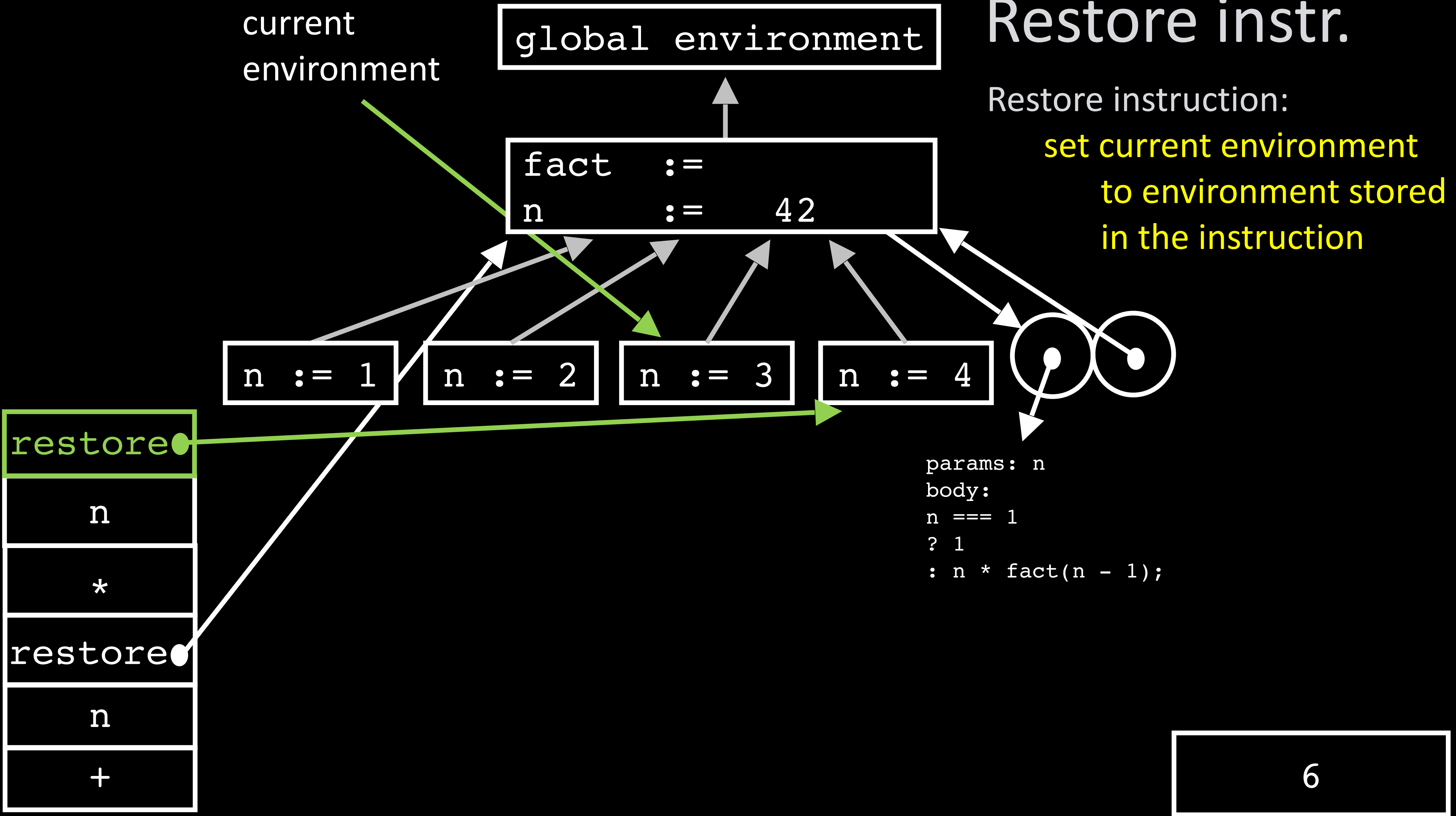
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```


current environment
global environment
fact :=
n := 42
n := 1
n := 2
n := 3
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
restore
n
*
restore
n
+
6

Restore instr.

Restore instruction:

set current environment
to environment stored
in the instruction

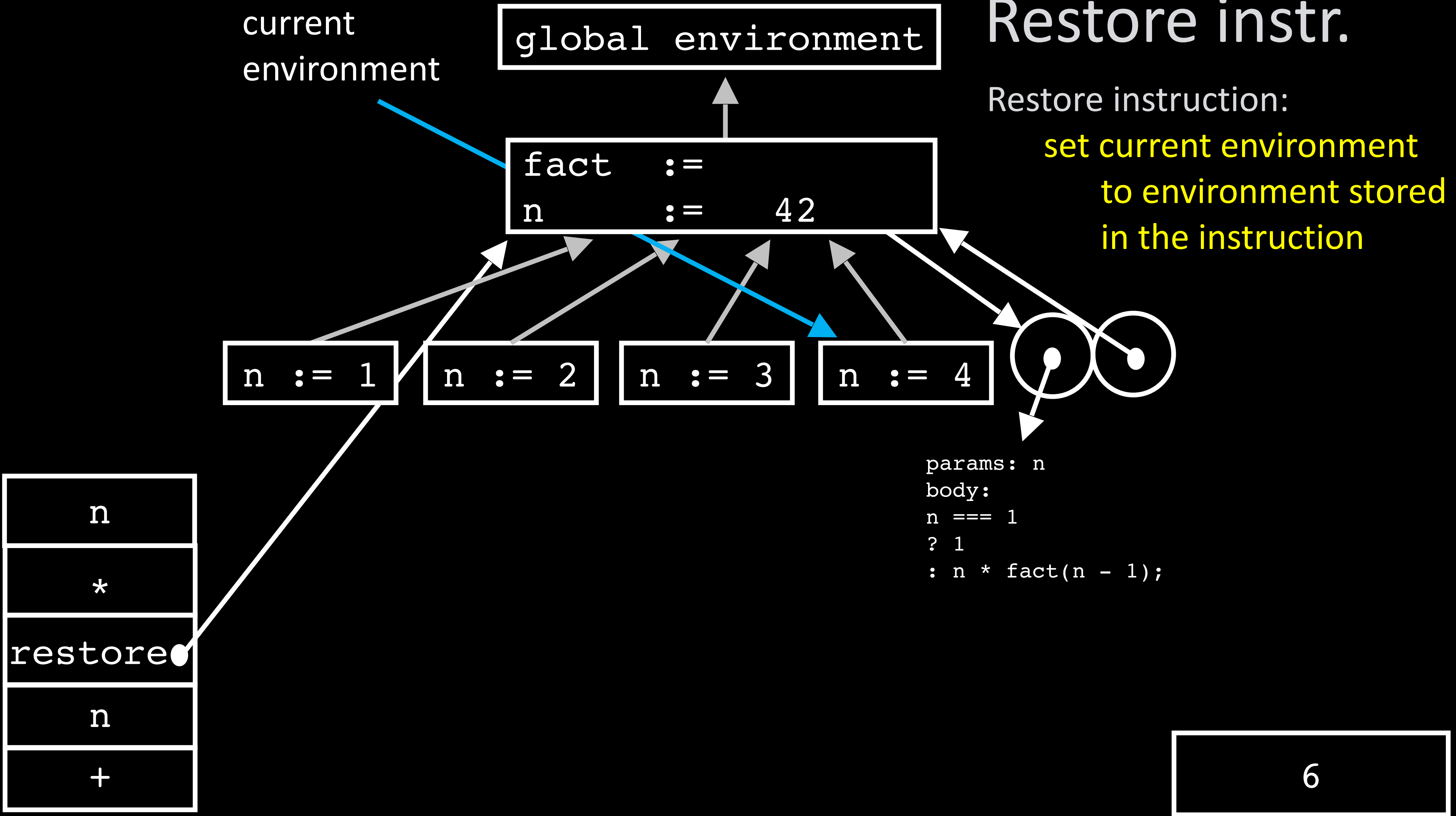


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

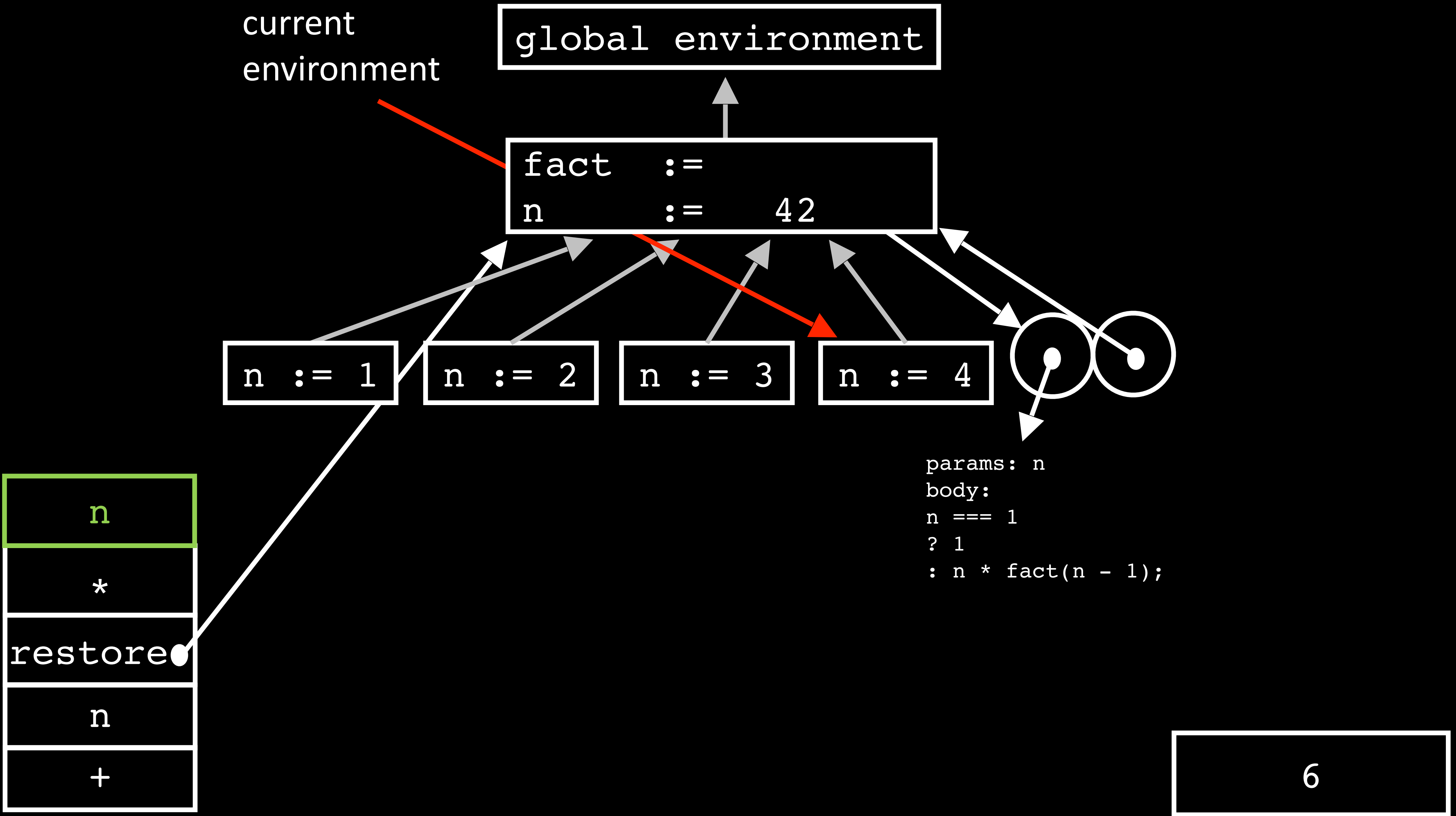
Restore instr.

Restore instruction:

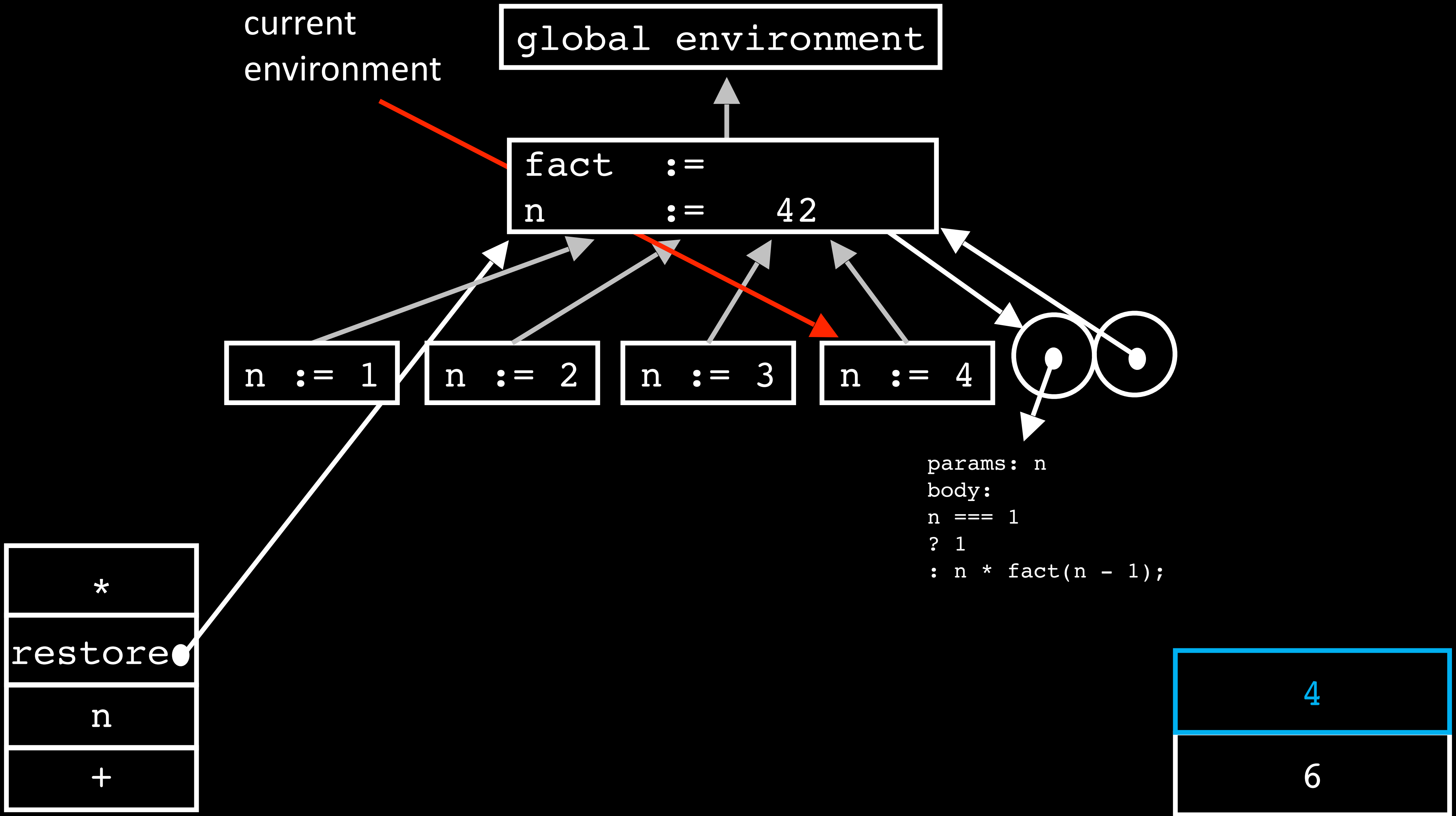
set current environment
to environment stored
in the instruction



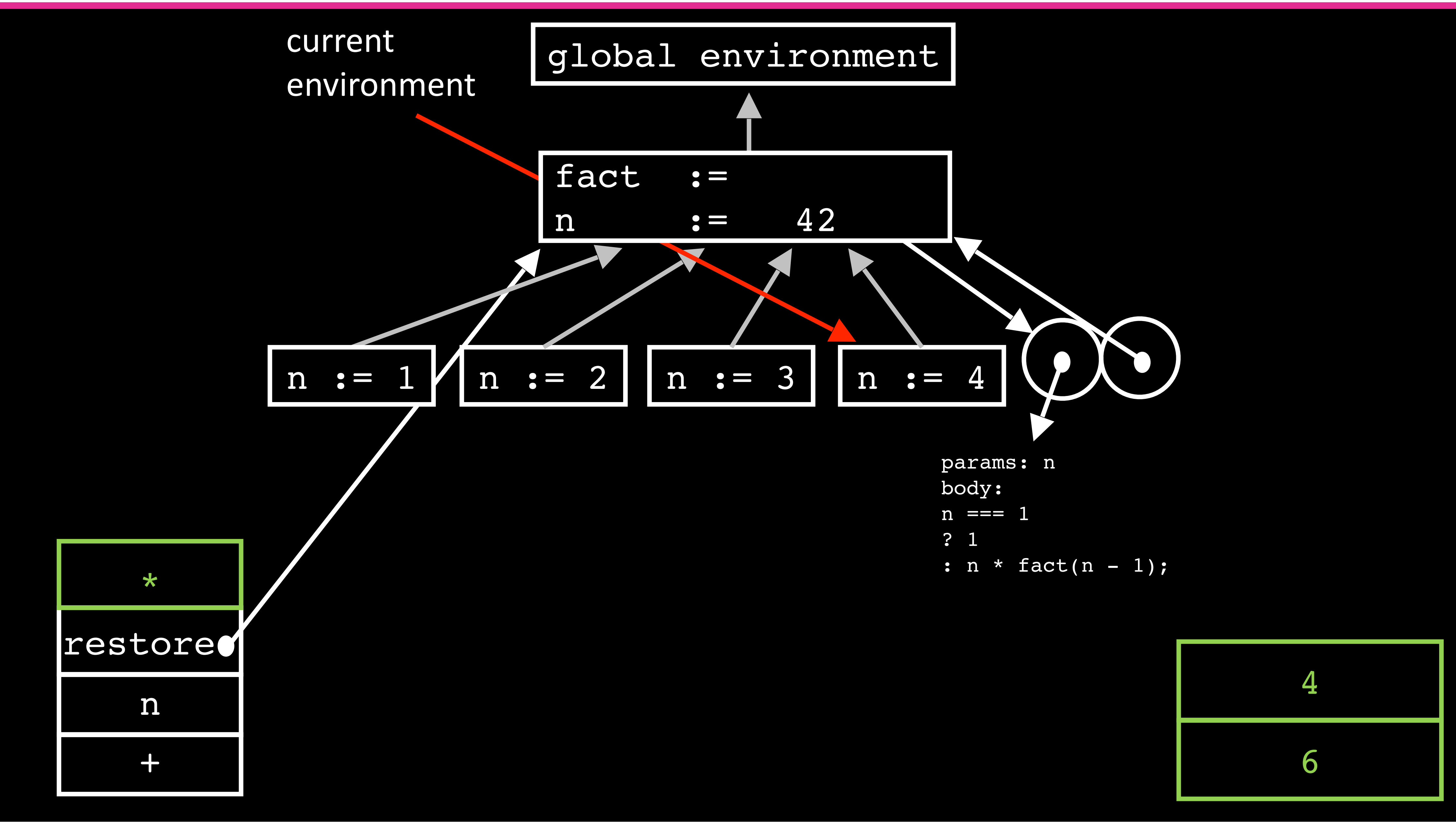
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



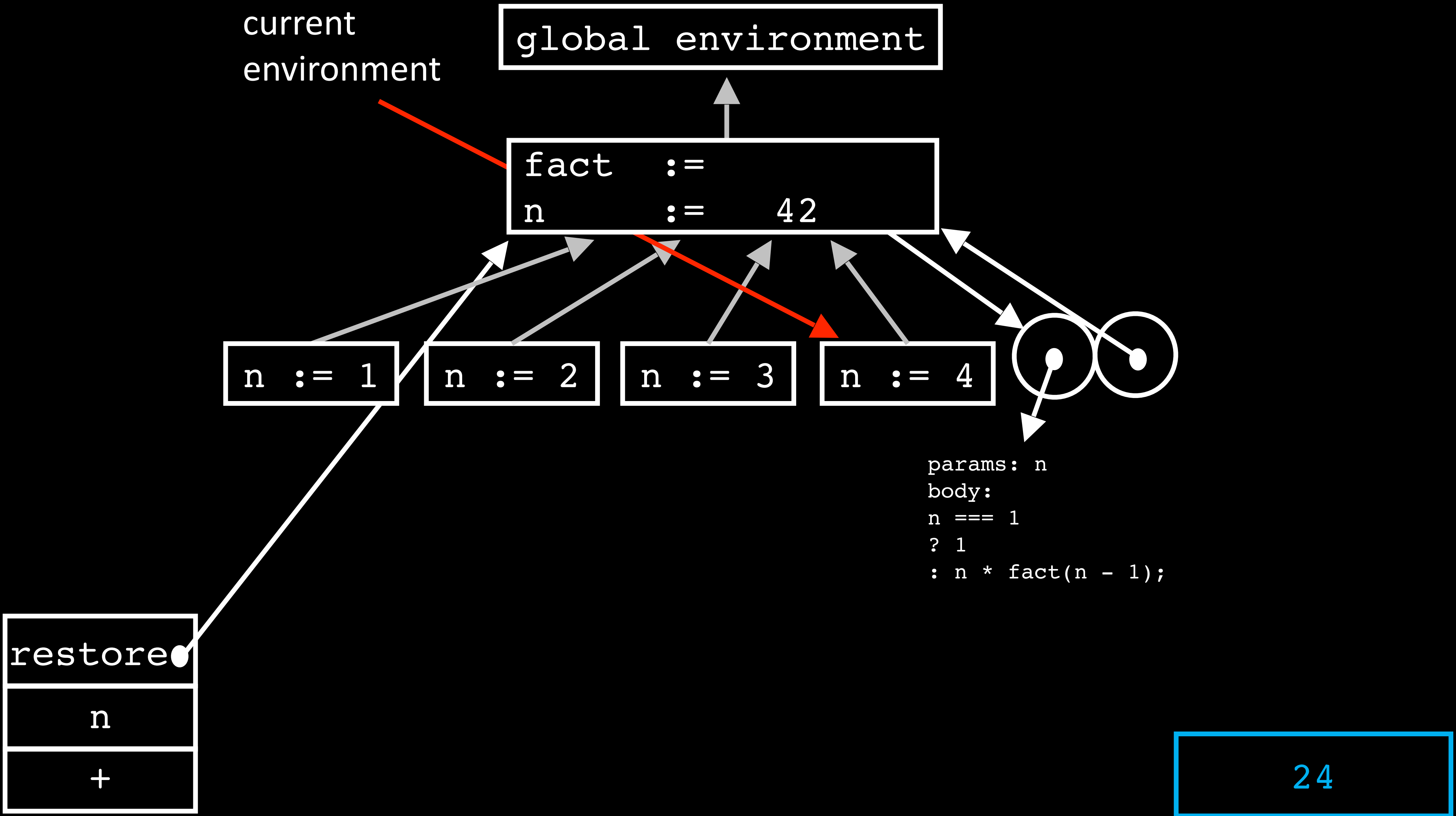
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

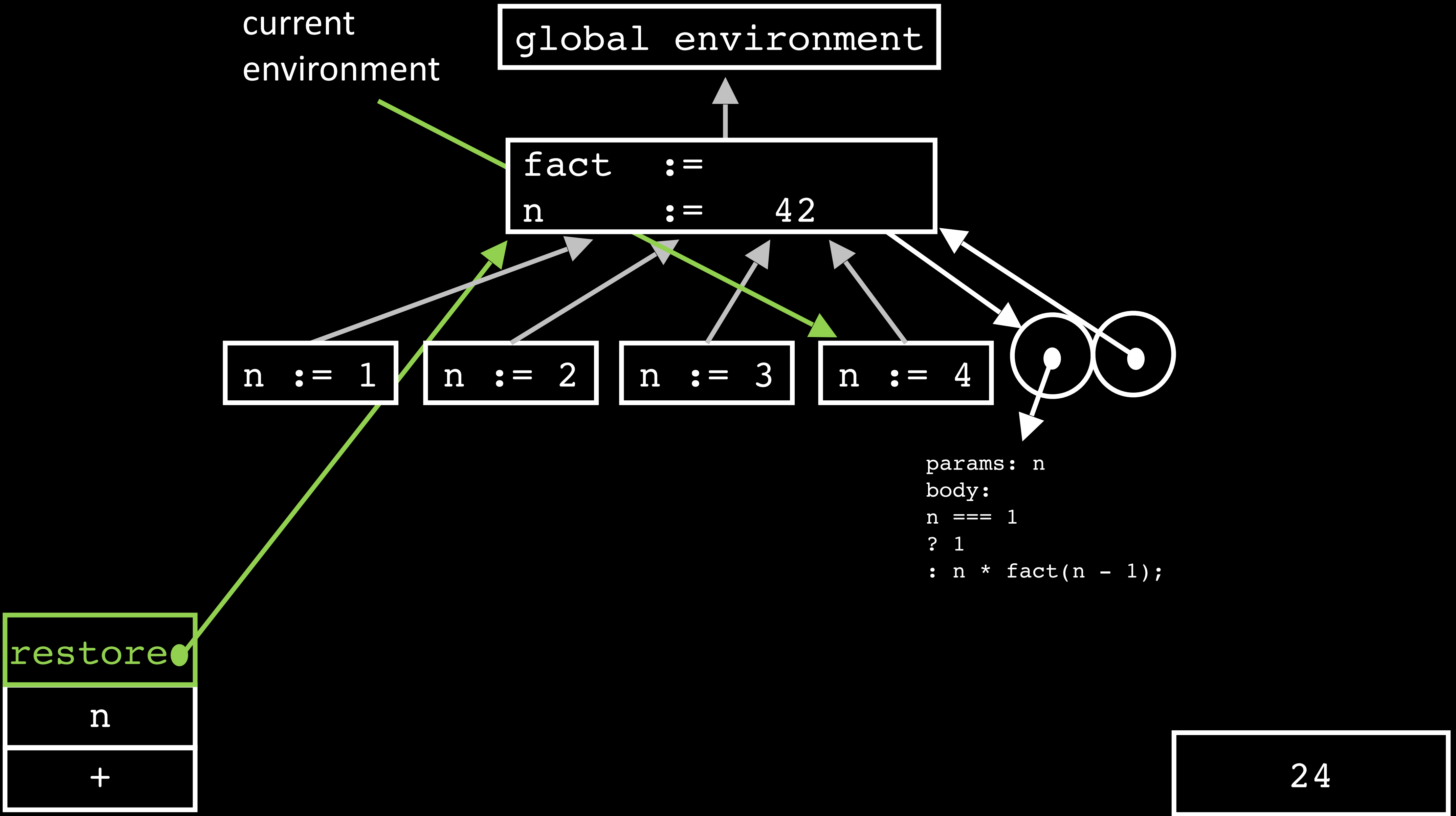


current environment
global environment
fact :=
n := 42
n := 1
n := 2
n := 3
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
*
restore
n
+
4
6

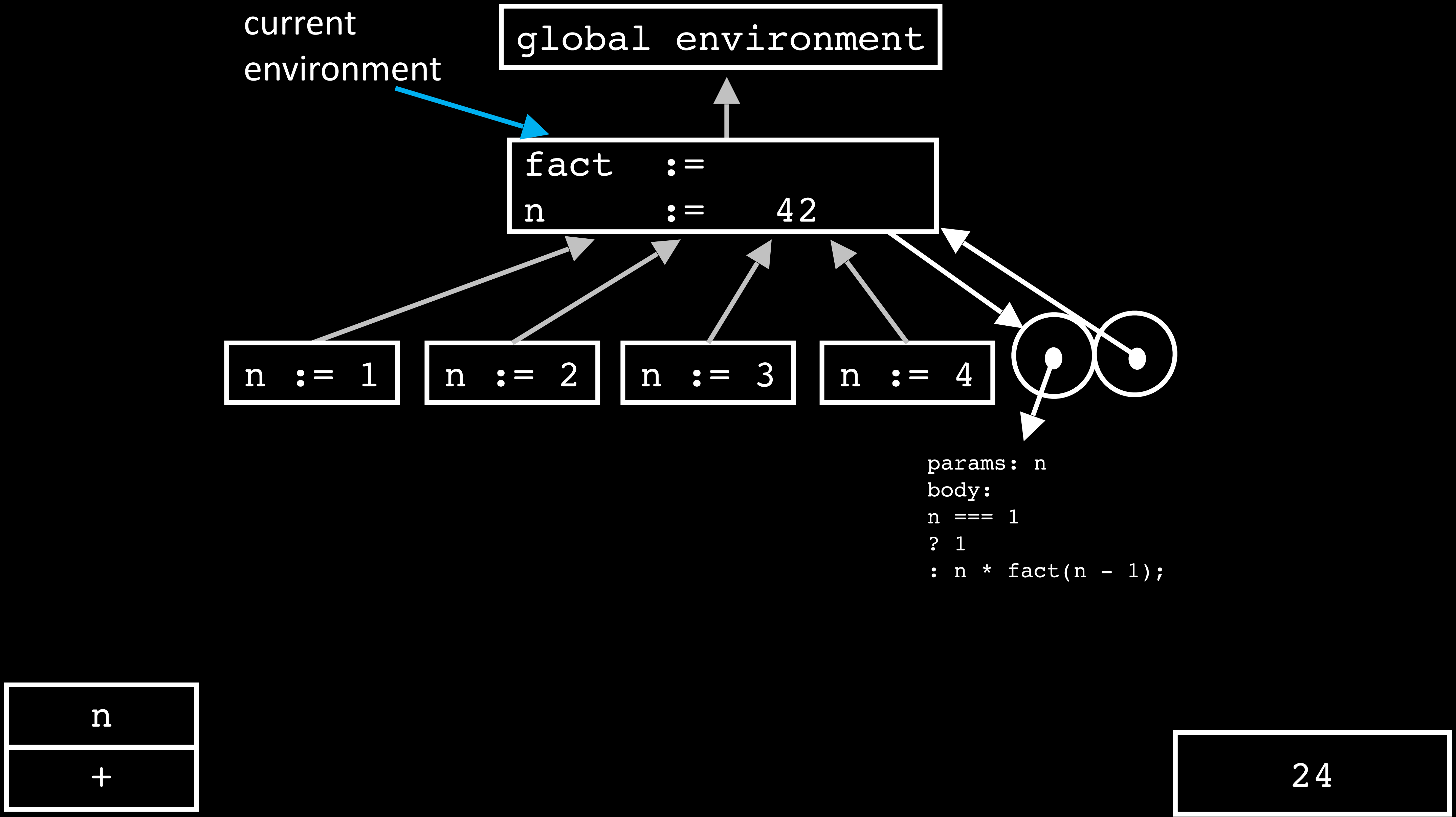


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

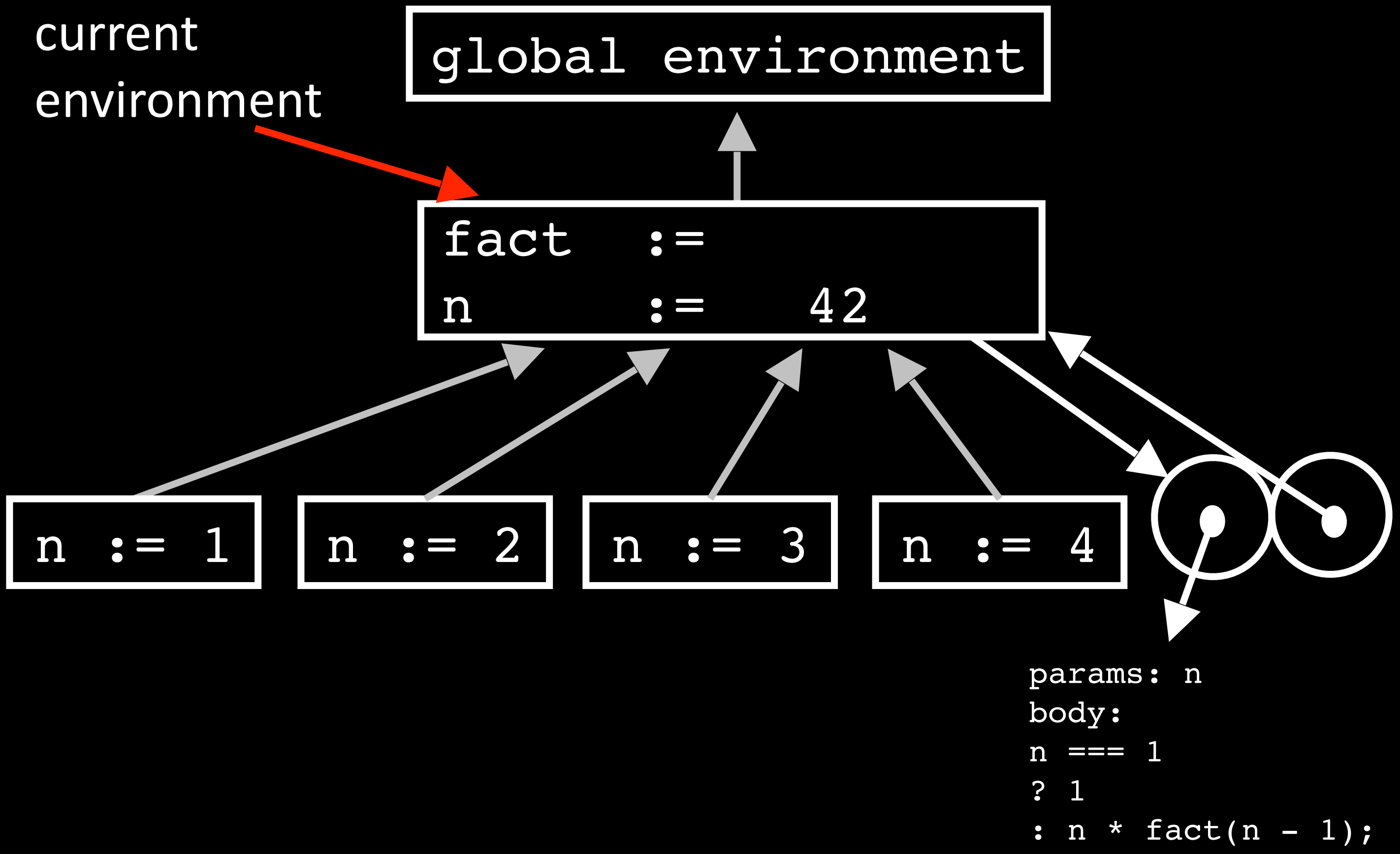

current
environment
global environment
fact :=
n := 42
n := 1
n := 2
n := 3
n := 4
params: n
body:
n === 1
? 1
: n * fact(n - 1);
restore
n
+
24



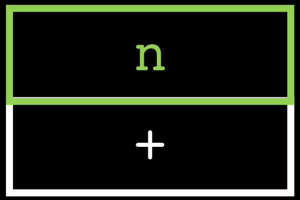
```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

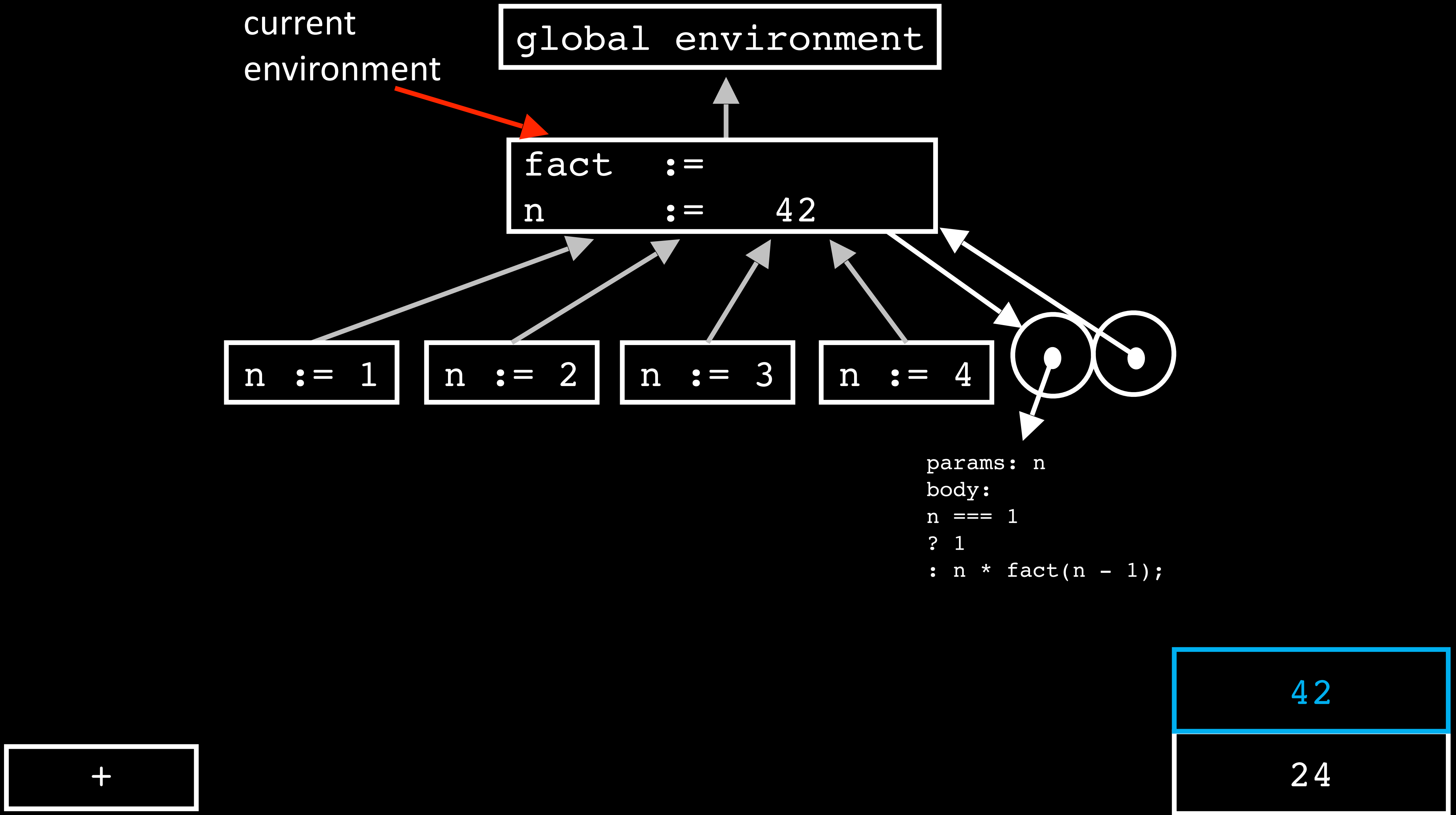


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

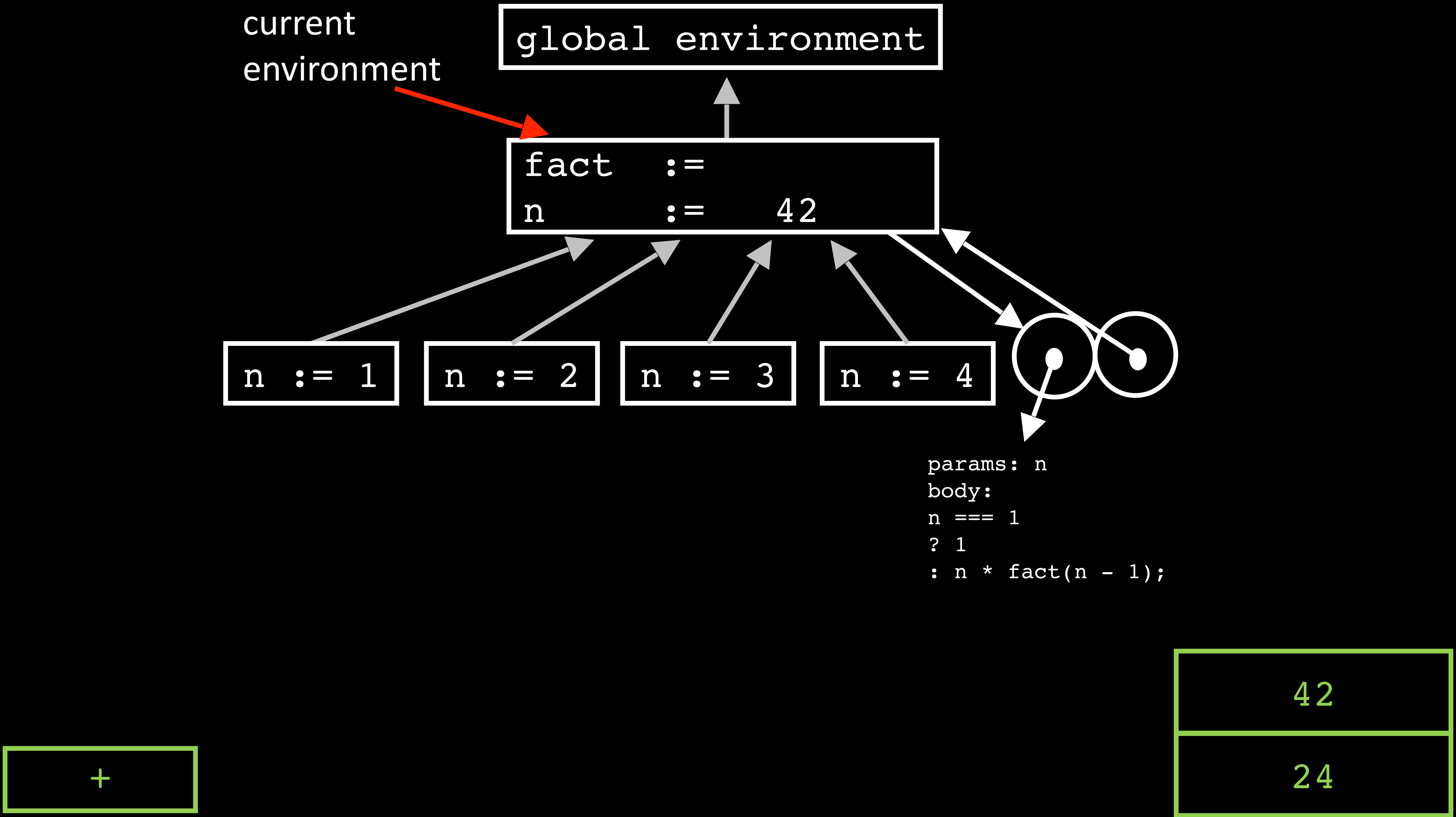


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



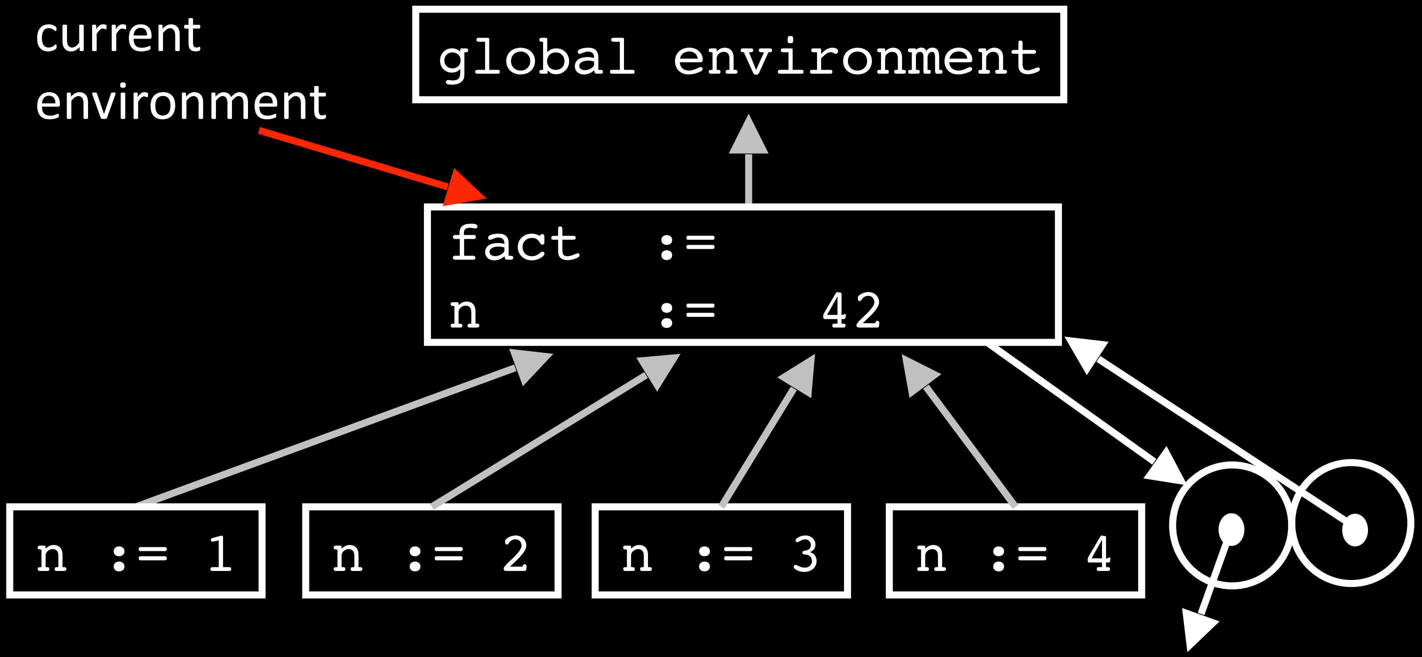


```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

Done!



```
params: n
body:
n === 1
? 1
: n * fact(n - 1);
```

The journey

- Calculator language
- Add conditionals, Booleans, sequences
- Add blocks, declarations, names
- Add function declaration and application (simple return)
- Restoring environments
- Further language features

Further language features

- General return statements
- Tail recursion
- Exception handling

The issue with general return statements

The issue with general return statements

- So far, the body of a function consisted of a single return statement.

The issue with general return statements

- So far, the body of a function consisted of a single return statement.
- Call instructions simply pushed the return expression on the agenda.

The issue with general return statements

- So far, the body of a function consisted of a single return statement.
- Call instructions simply pushed the return expression on the agenda.
- When evaluation of the return expression is done, the result is on the stash, where it should be, so the caller can use it.

The issue with general return statements

- So far, the body of a function consisted of a single return statement.
- Call instructions simply pushed the return expression on the agenda.
- When evaluation of the return expression is done, the result is on the stash, where it should be, so the caller can use it.
- With return statements anywhere in the body, there can be agenda items from the body that must be skipped, to get to the caller agenda.

The solution

The solution

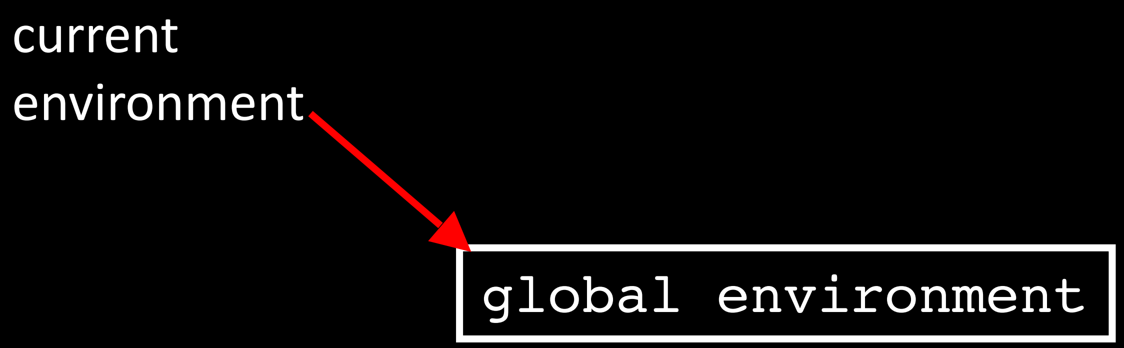
- Calling complex functions pushes a marker on the agenda.

The solution

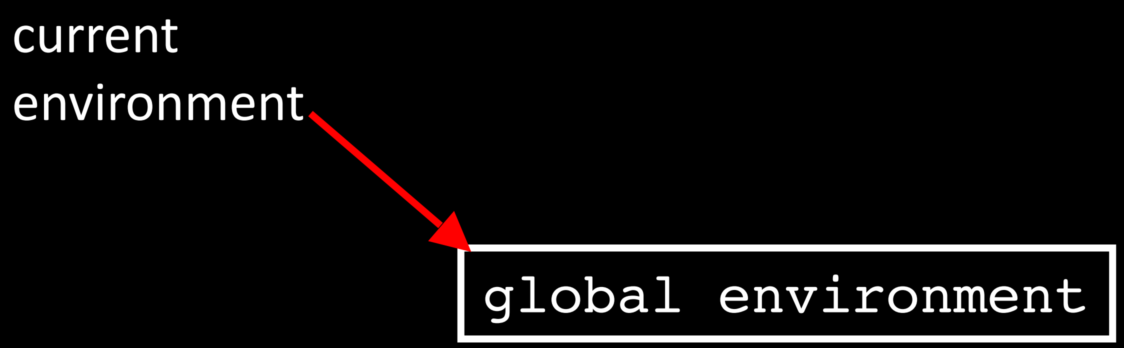
- Calling complex functions pushes a marker on the agenda.
- Any return statement in a complex function pushes a reset instruction that removes all agenda items that were pushed after the last marker.

Example

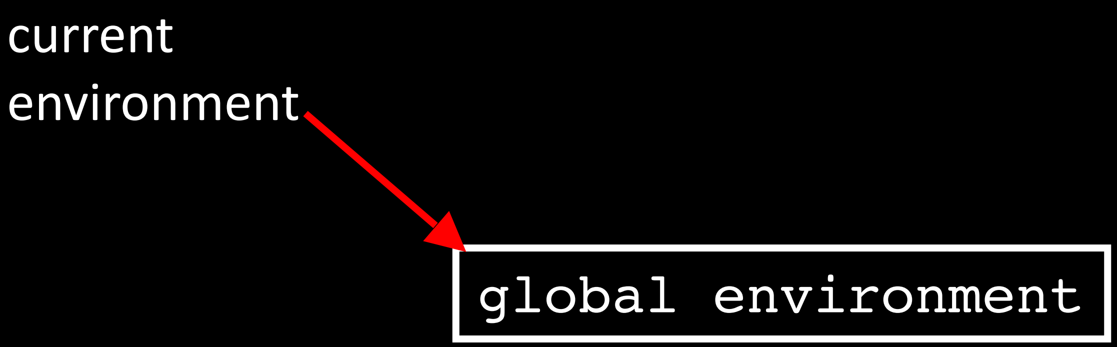
```
function f(b, x) {
    if (b) {
        return x + 1;
    }
    return x - 1;
}
f(true, 7);
```



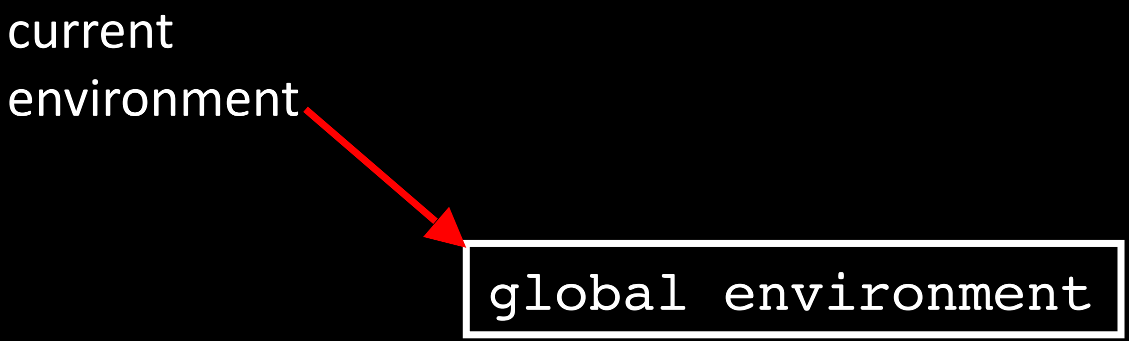
```
function f(b, x) {
    if (b) {
     return x + 1;
 }
 return x - 1;
}
f(true, 7);
```



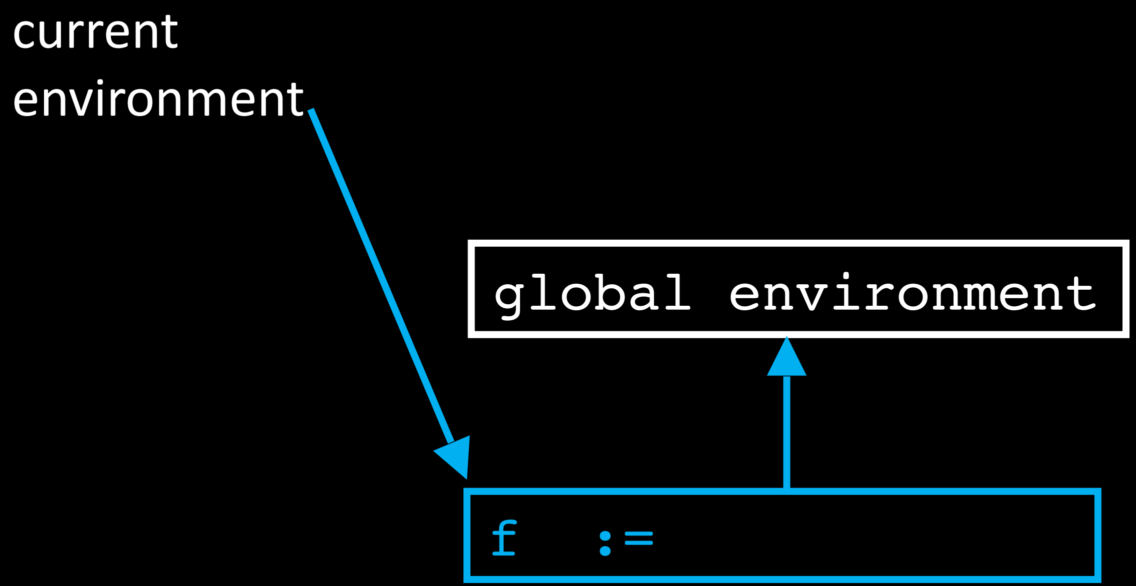
```
function f(b, x) {
   if (b) {
      return x + 1;
   }
   return x - 1;
}
f(true, 7);
```



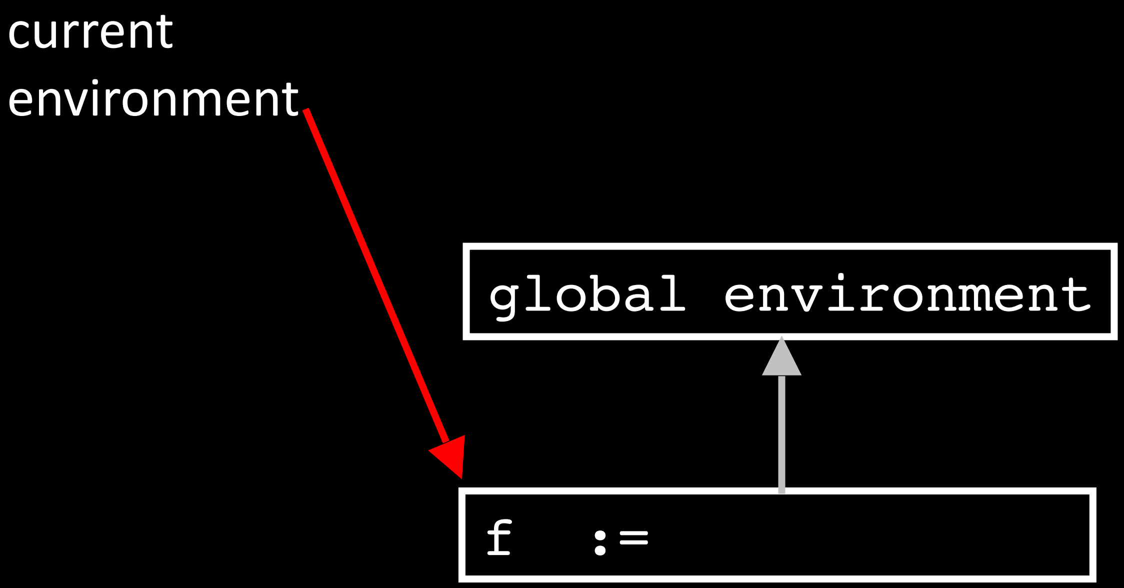
```
{
 function f(b, x) {
    if (b) {
       return x + 1;
  }
    return x - 1;
 }
 f(true, 7);}
}
```



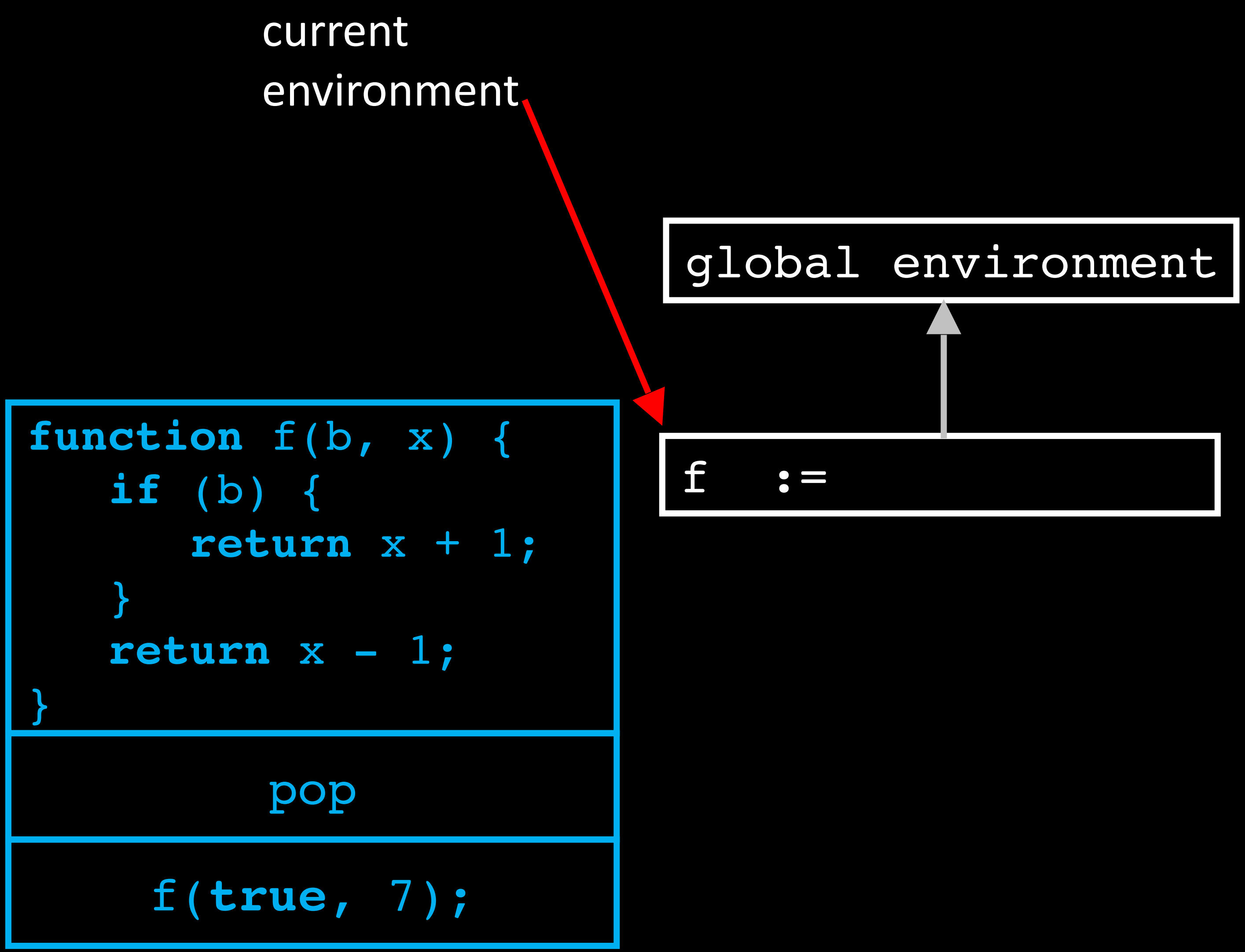
```
{
  function f(b, x) {
      if (b) {
          return x + 1;
      }
      return x - 1;
  }
  f(true, 7);
}
```



```
function f(b, x) {
   if (b) {
      return x + 1;
   }
   return x - 1;
}
f(true, 7);
```



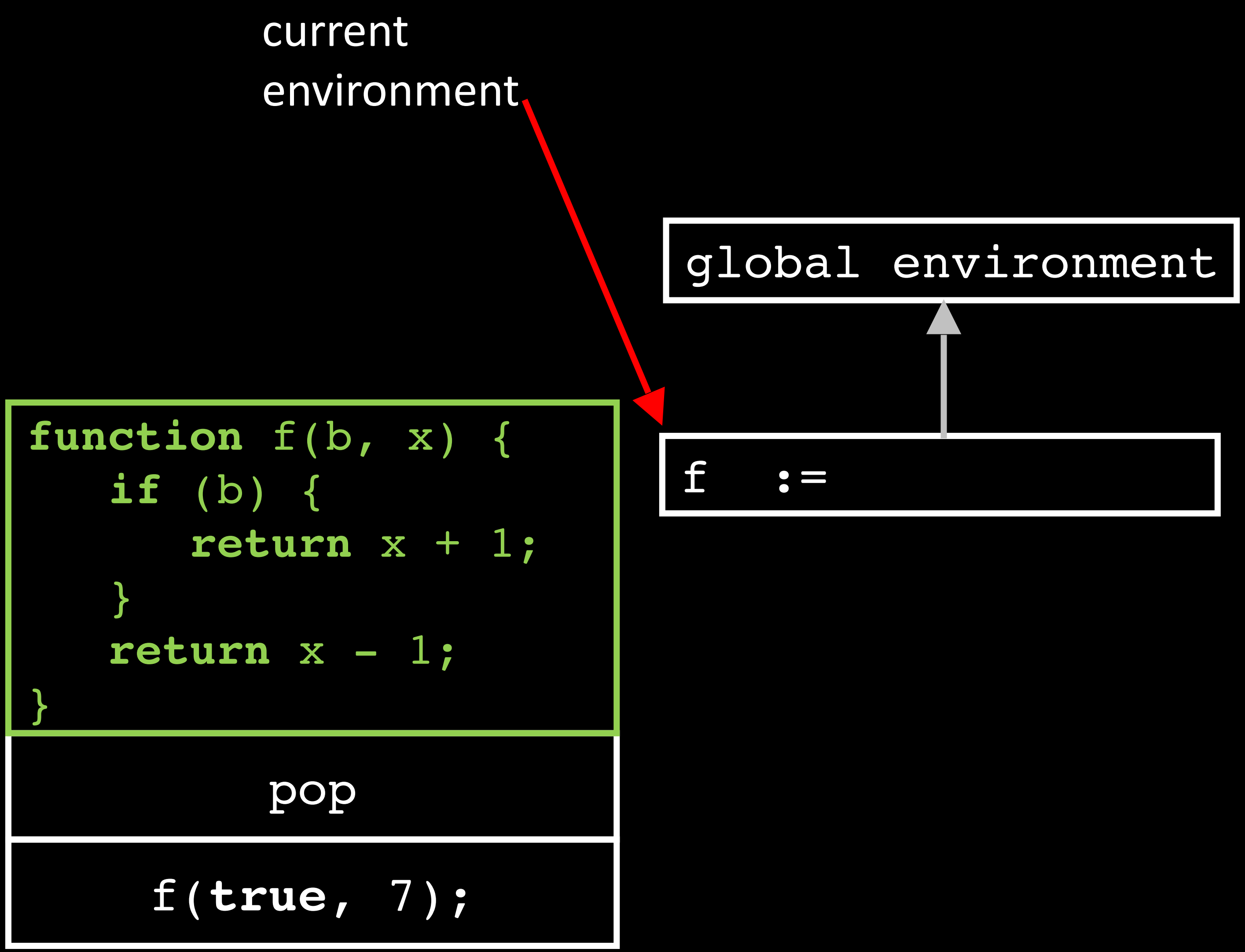
```
function f(b, x) {
    if (b) {
        return x + 1;
    }
    return x - 1;
}
f(true, 7);
```



```
function f(b, x) {
    if (b) {
        return x + 1;
    }
    return x - 1;
}
```

```
pop
```

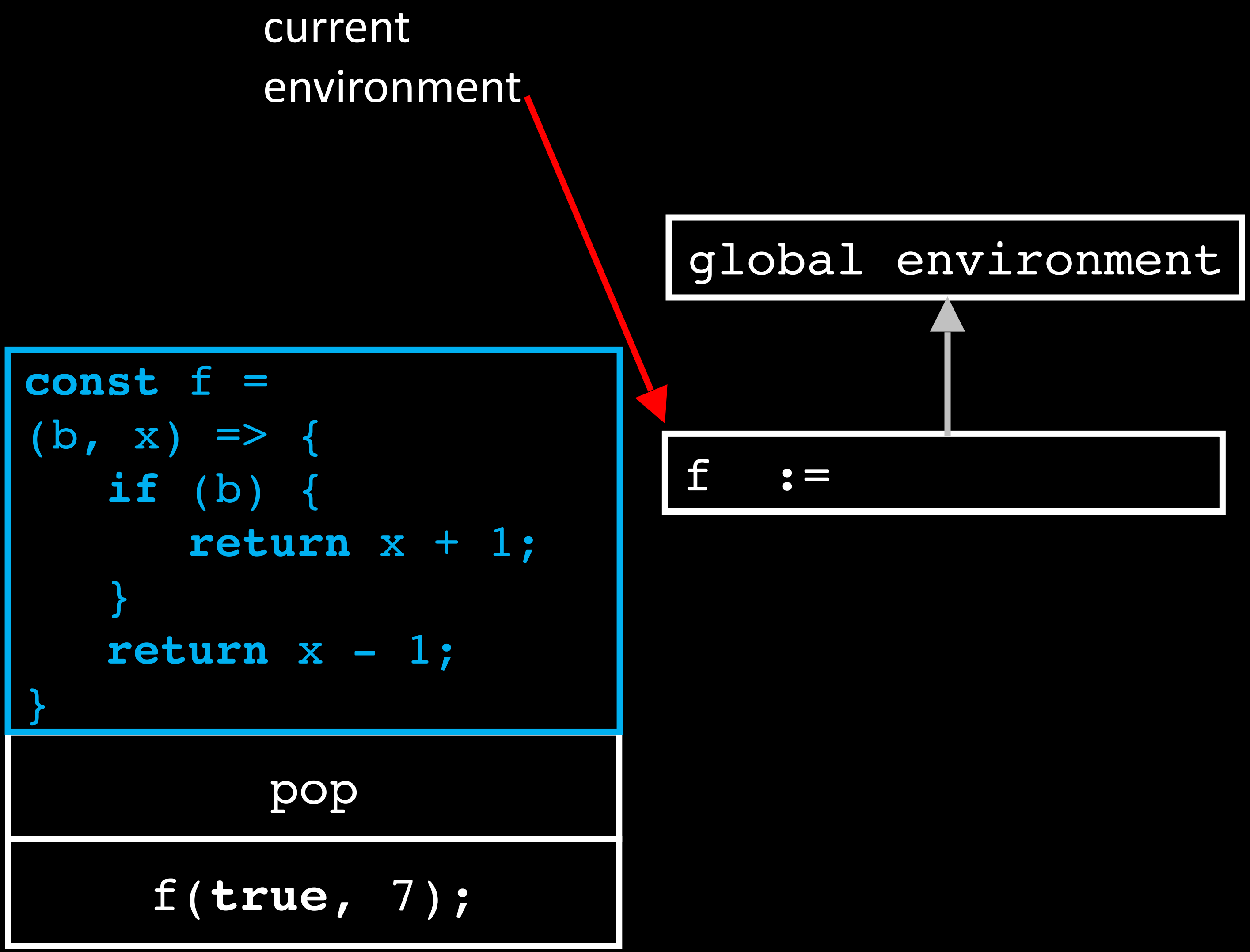
```
f(true, 7);
```



```
function f(b, x) {
    if (b) {
        return x + 1;
    }
    return x - 1;
}
```

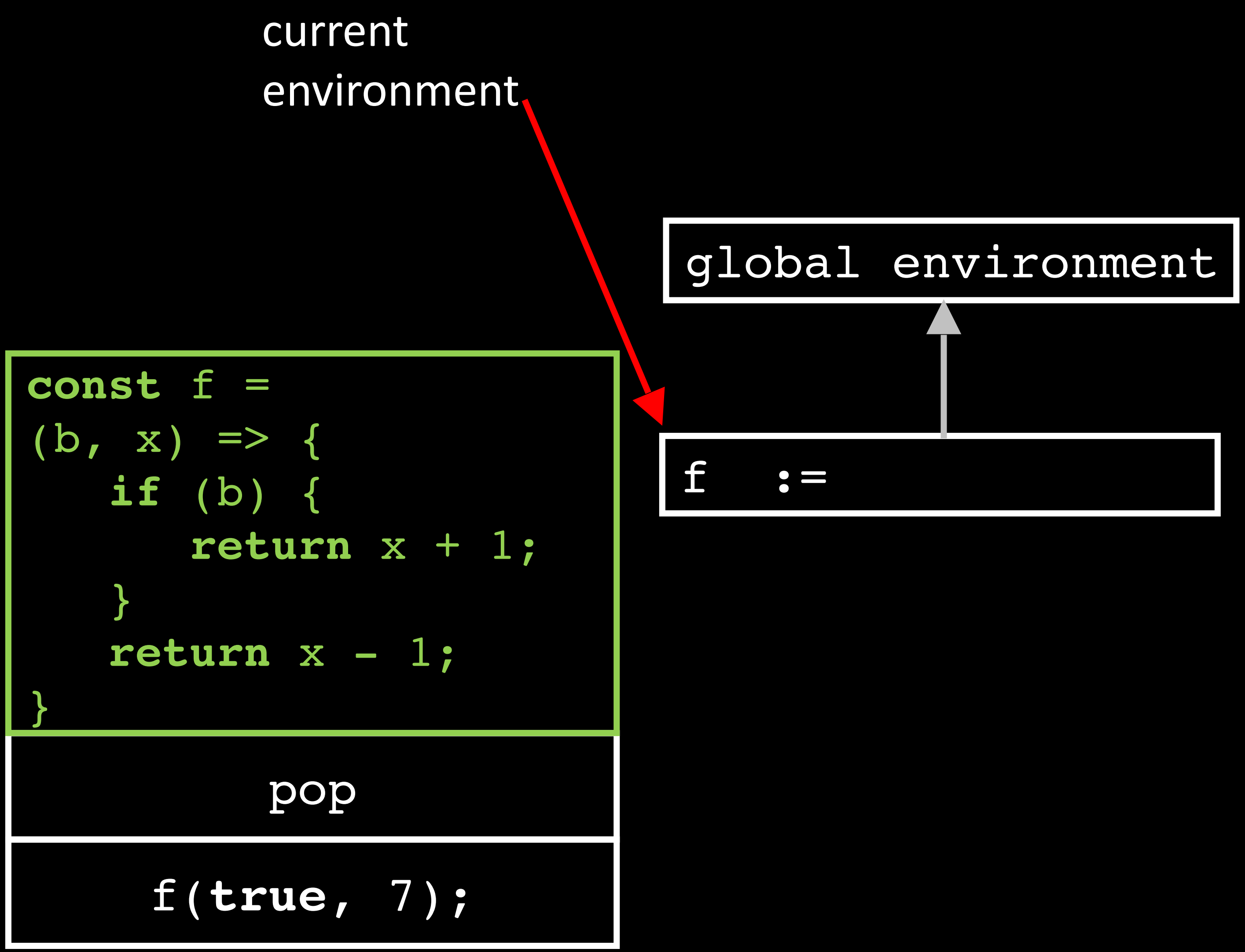
```
pop
```

```
f(true, 7);
```



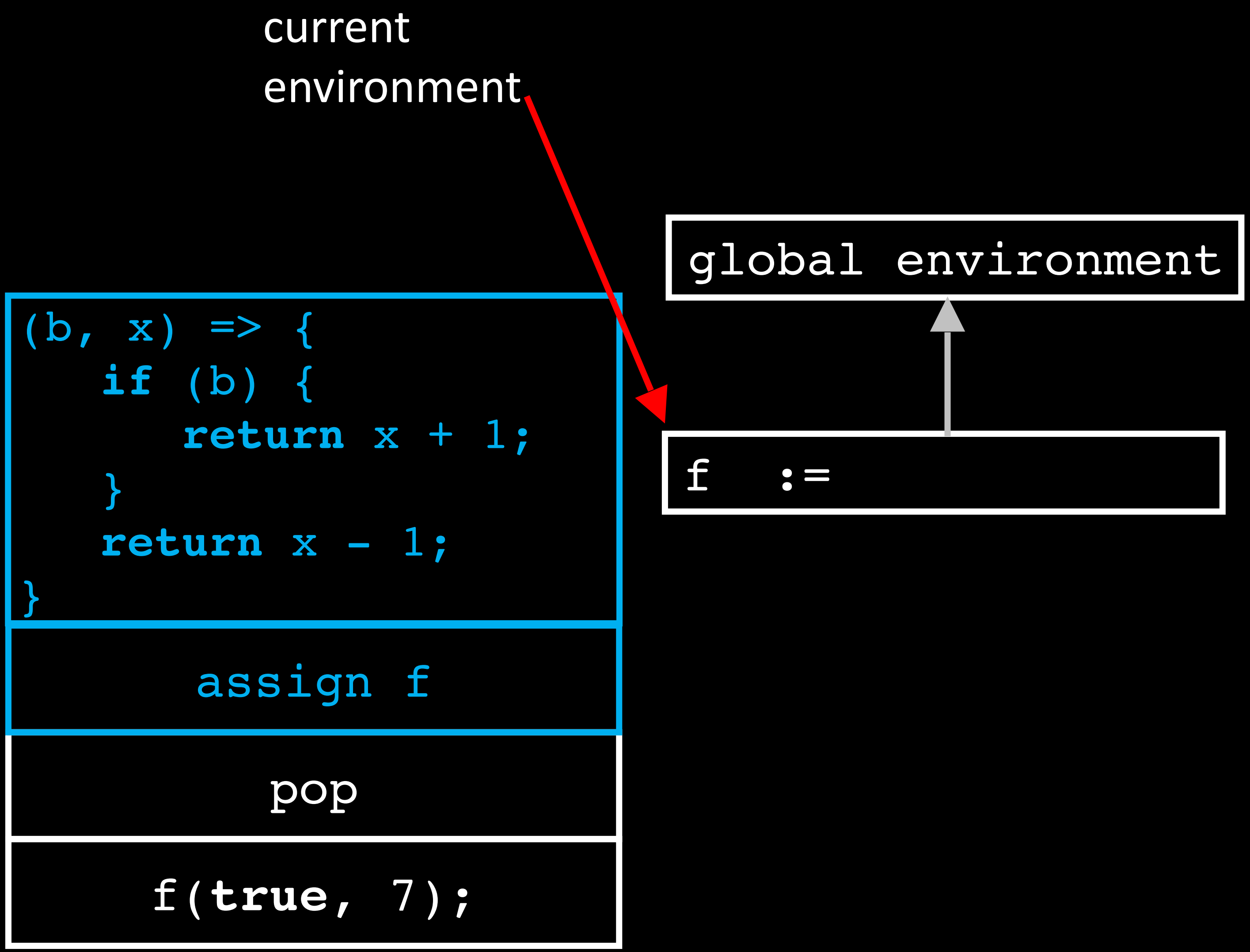
```
const f =
(b, x) => {
   if (b) {
      return x + 1;
   }
   return x - 1;
}
```

```
f(true, 7);
```



```
const f =
(b, x) => {
    if (b) {
        return x + 1;
    }
    return x - 1;
}
```

```
f(true, 7);
```

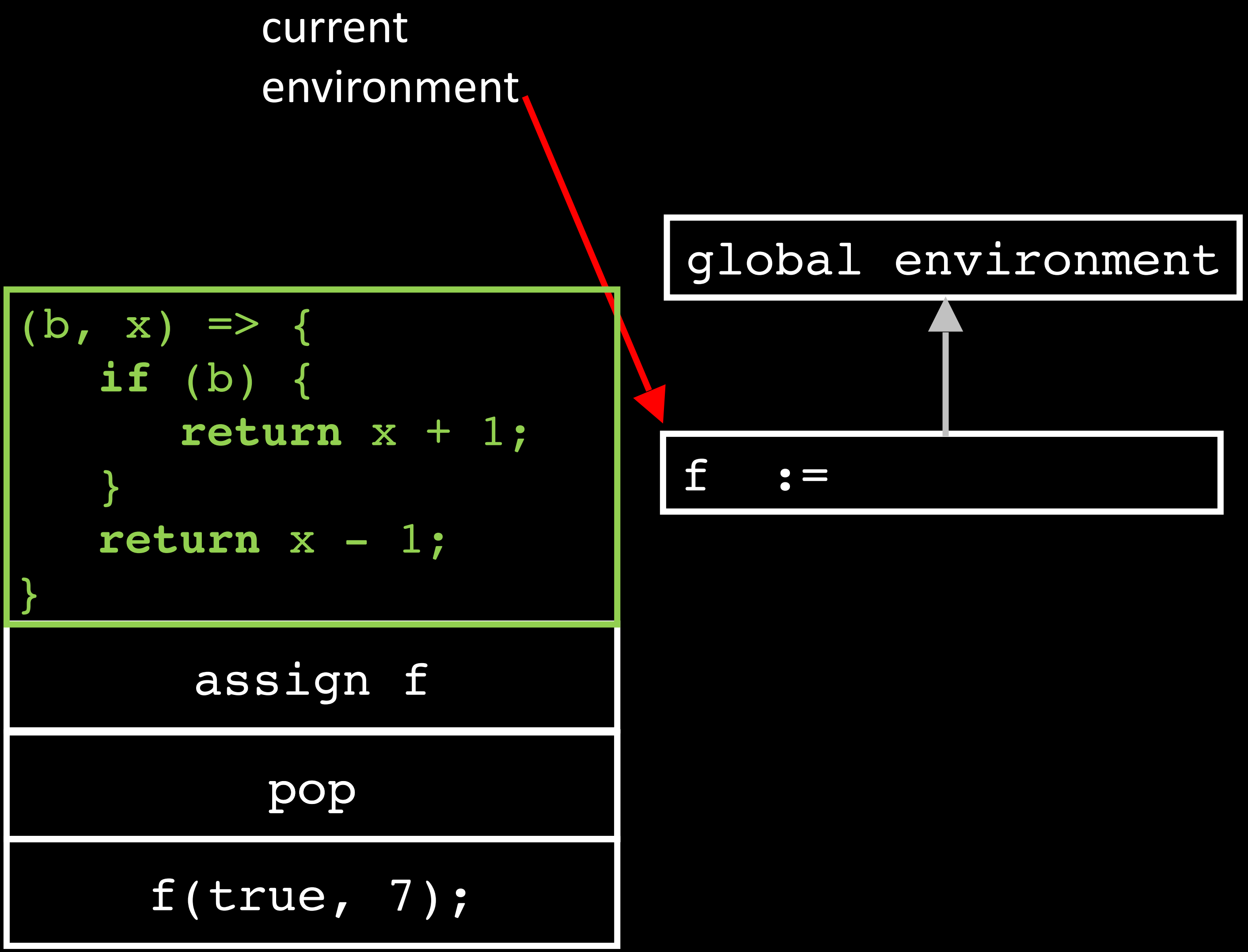


```
(b, x) => {
    if (b) {
        return x + 1;
    }
    return x - 1;
}
```

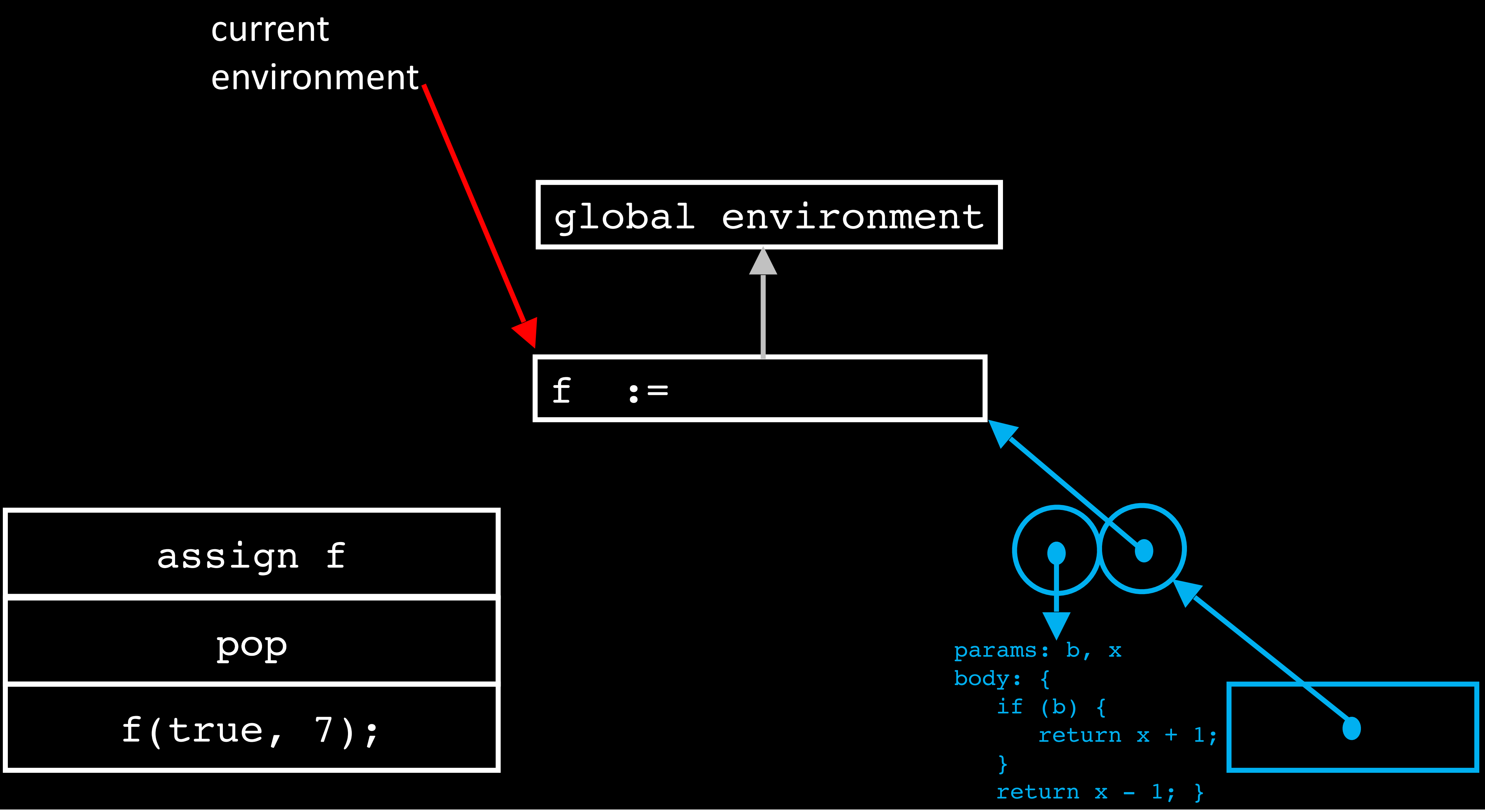
```
assign f
```

```
pop
```

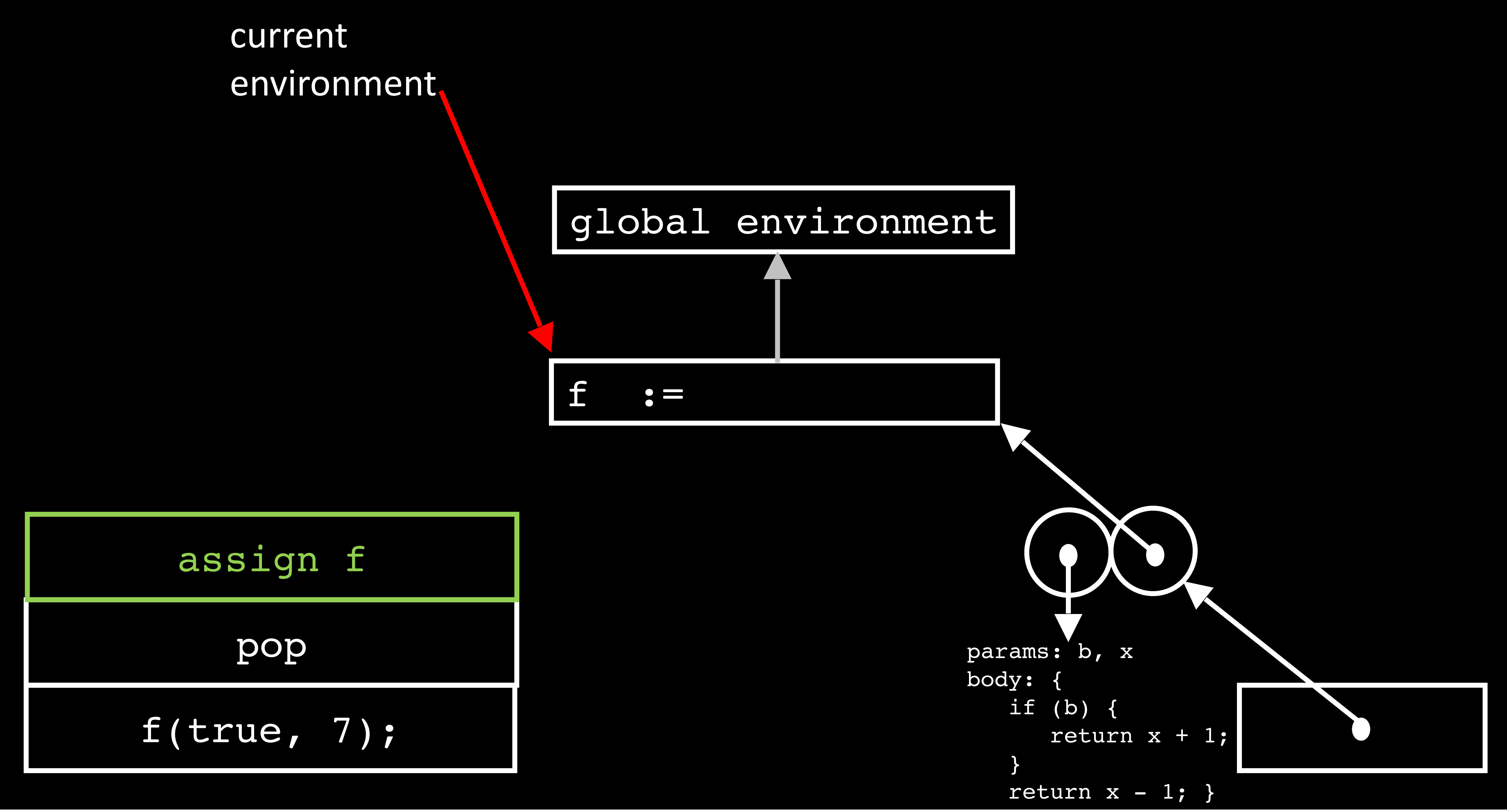
```
f(true, 7);
```



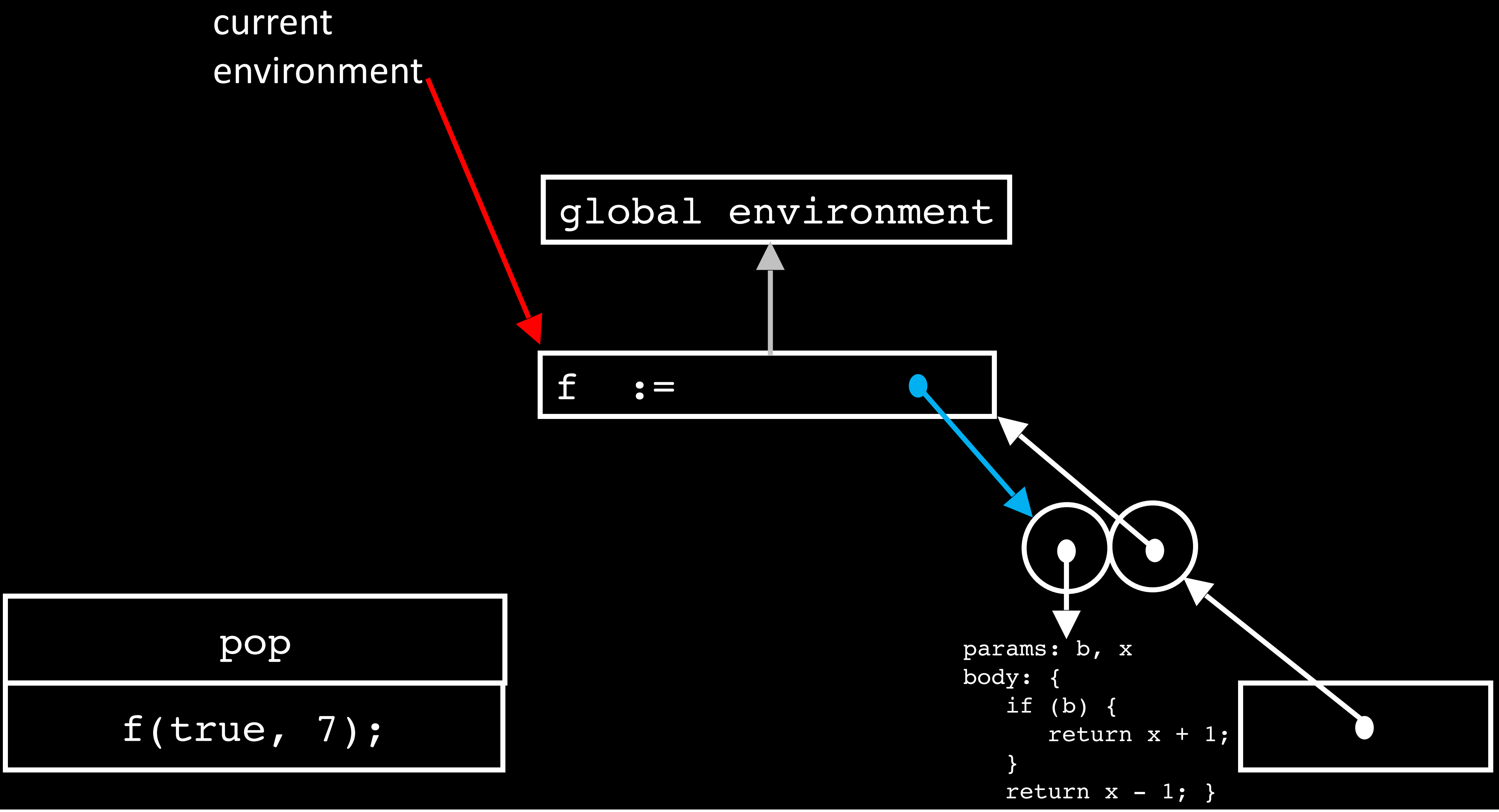
```
(b, x) => {
   if (b) {
      return x + 1;
   }
   return x - 1;
}
```



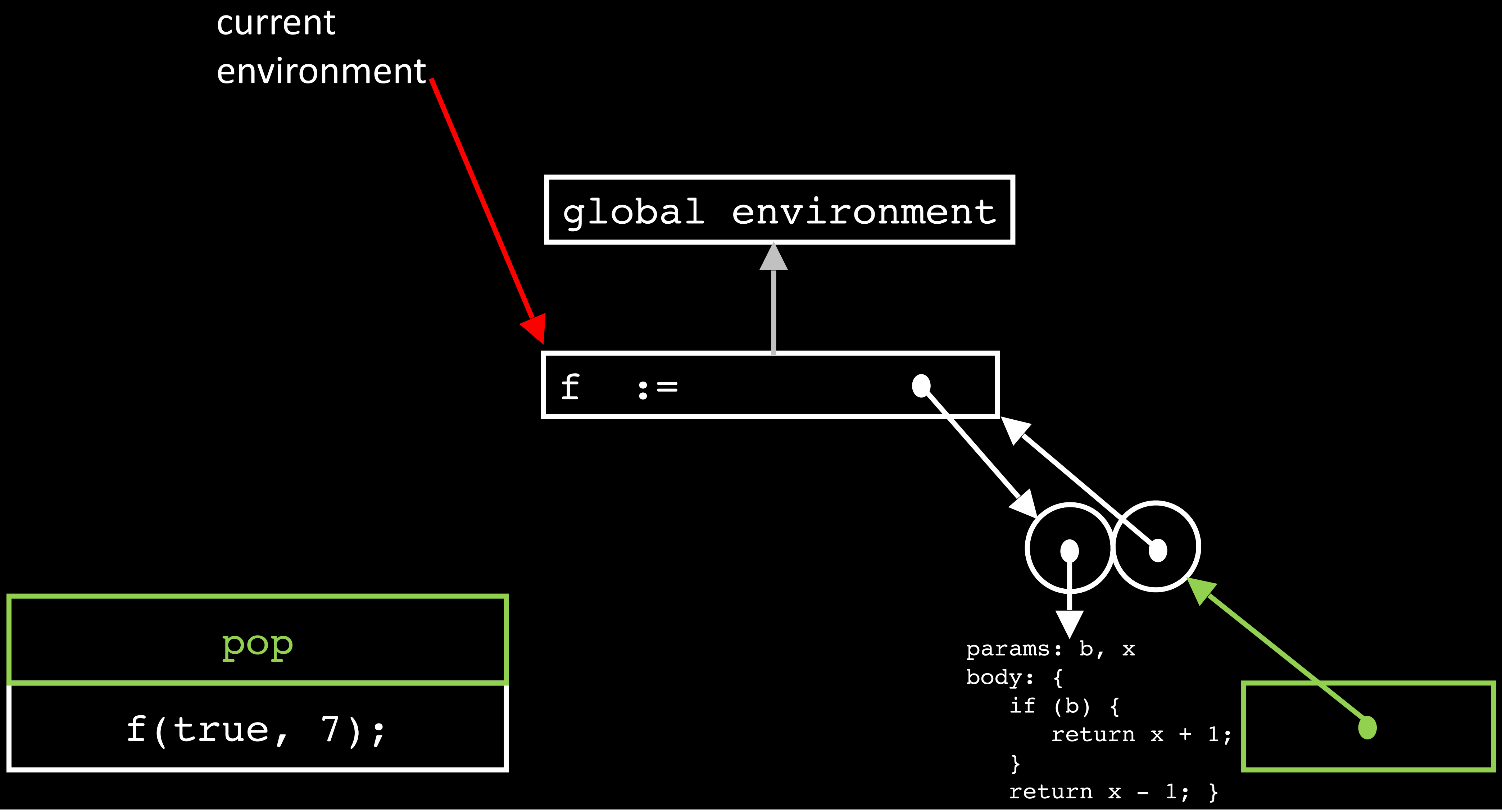
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



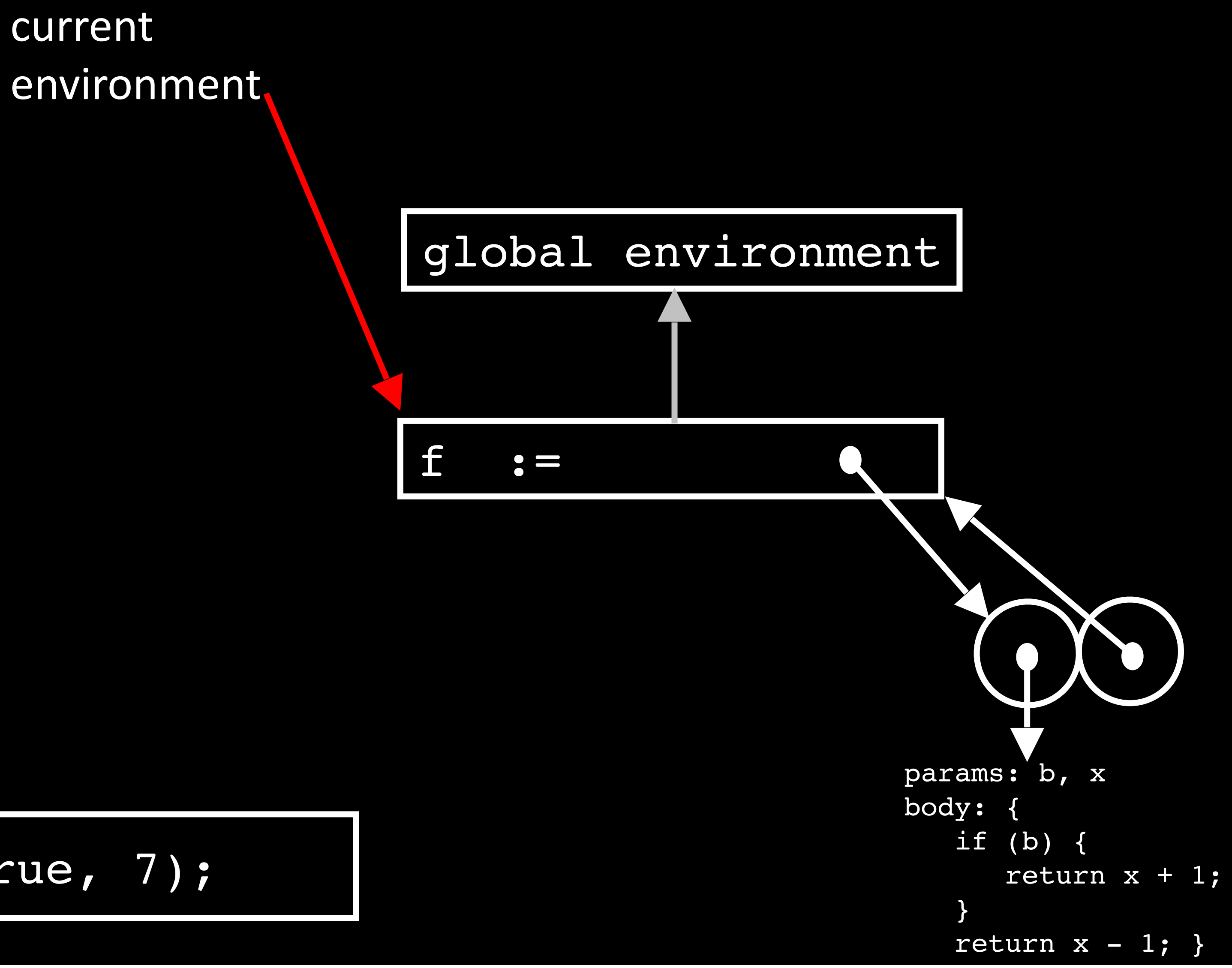
```
params: b, x
body: {
   if (b) {
      return x + 1;
   }
   return x - 1; }
```



```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

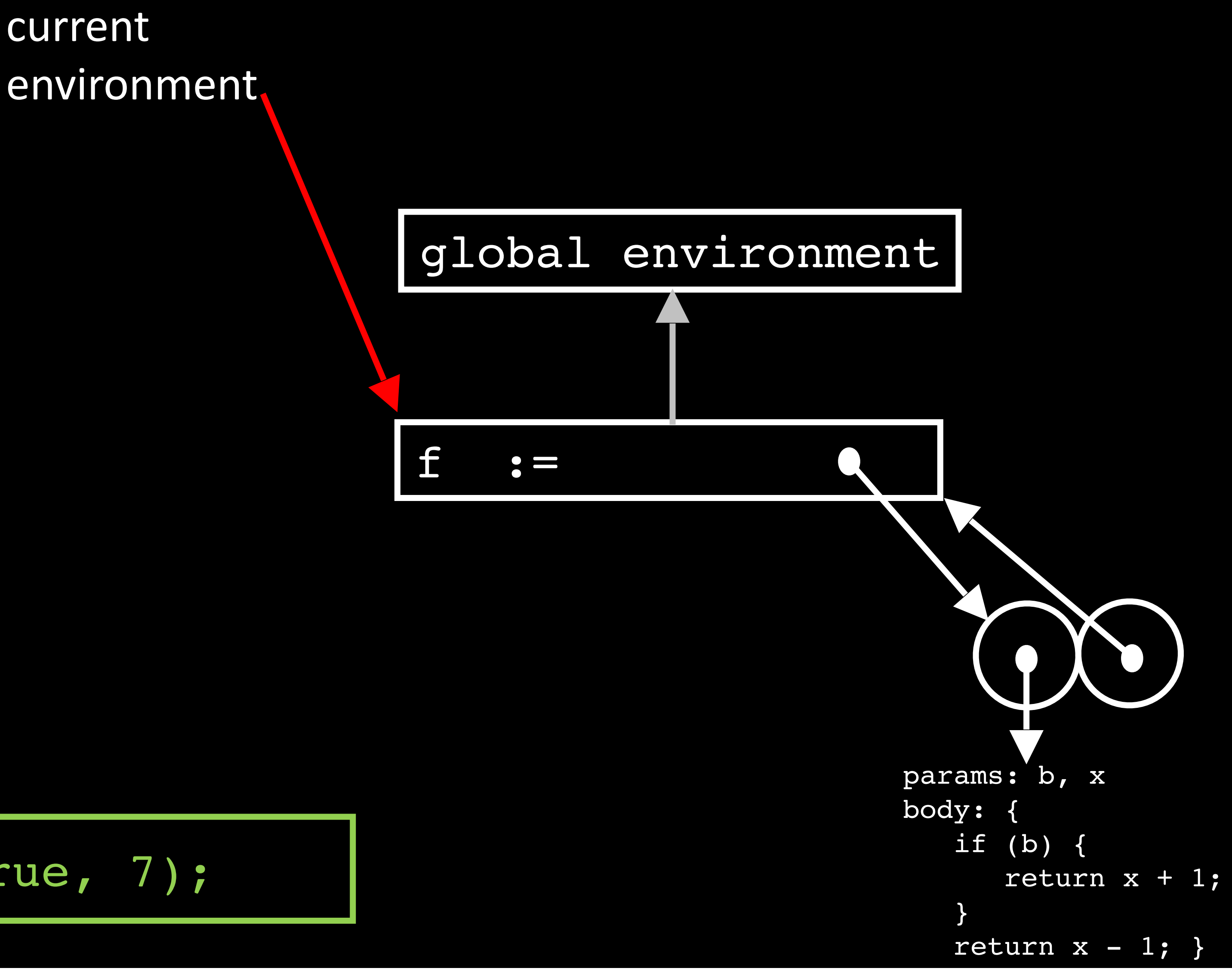


```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



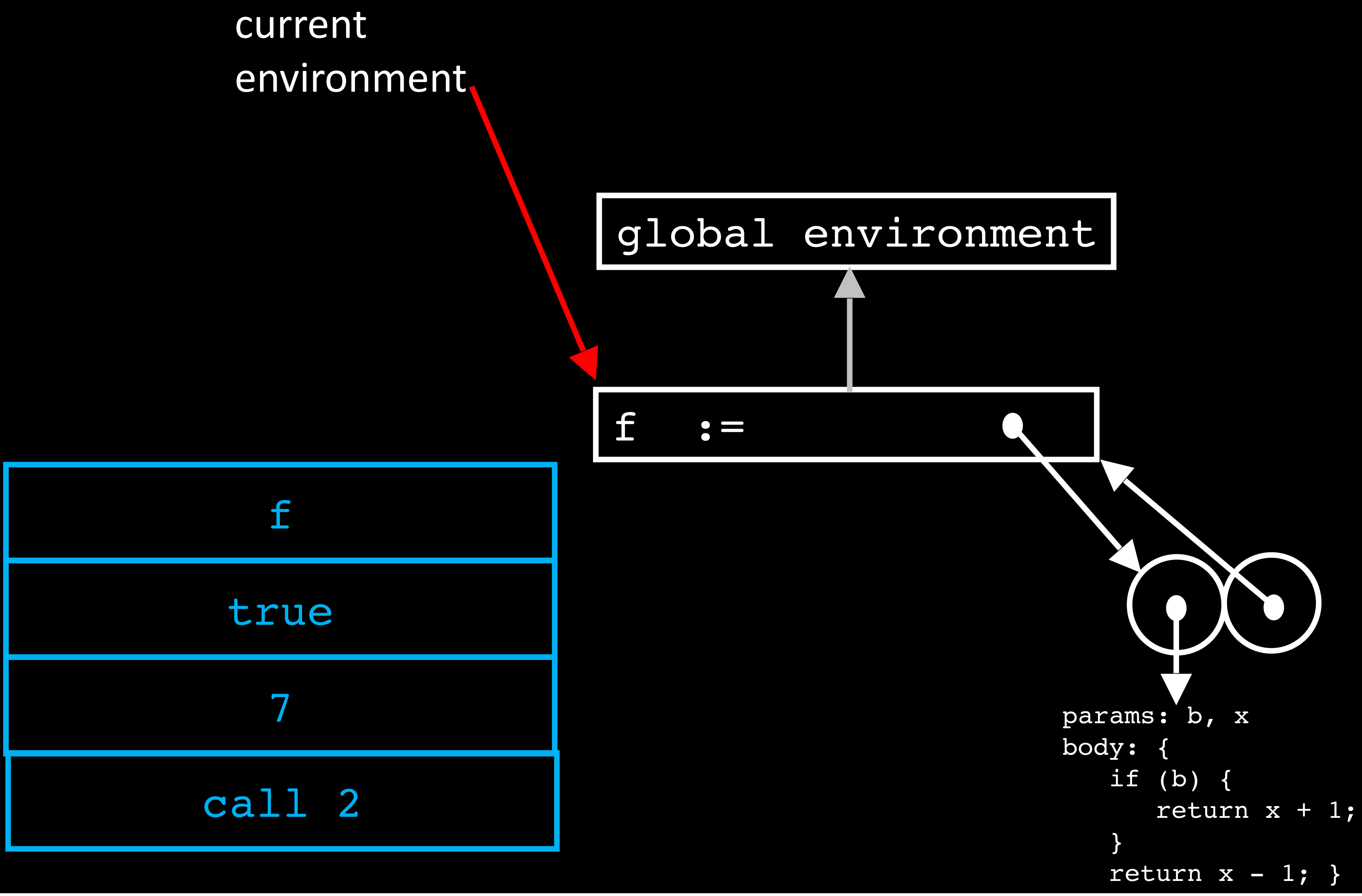
```
params: b, x
body: {
   if (b) {
      return x + 1;
   }
   return x - 1; }
```

```
f(true, 7);
```

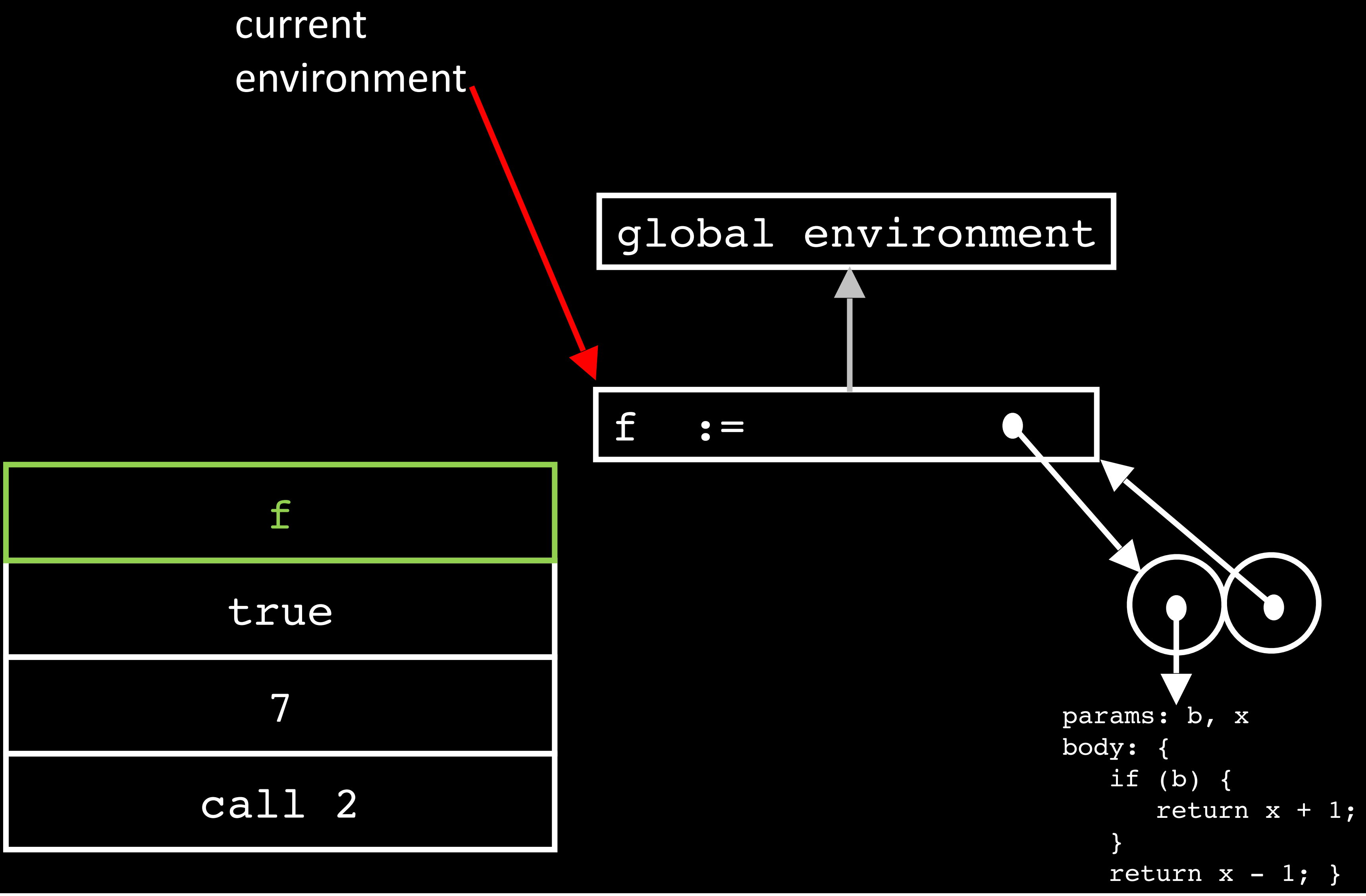


```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

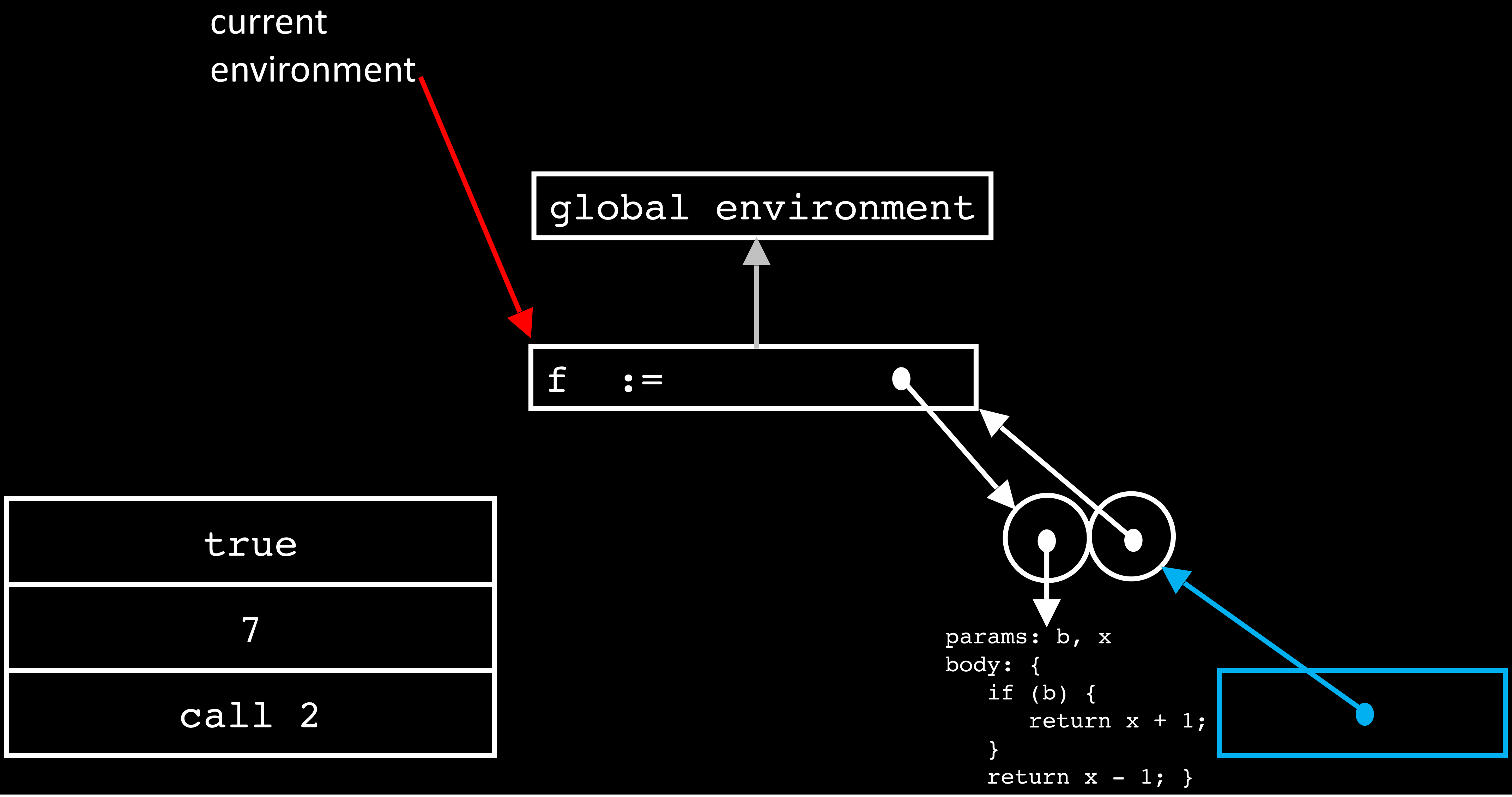
```
f(true, 7);
```



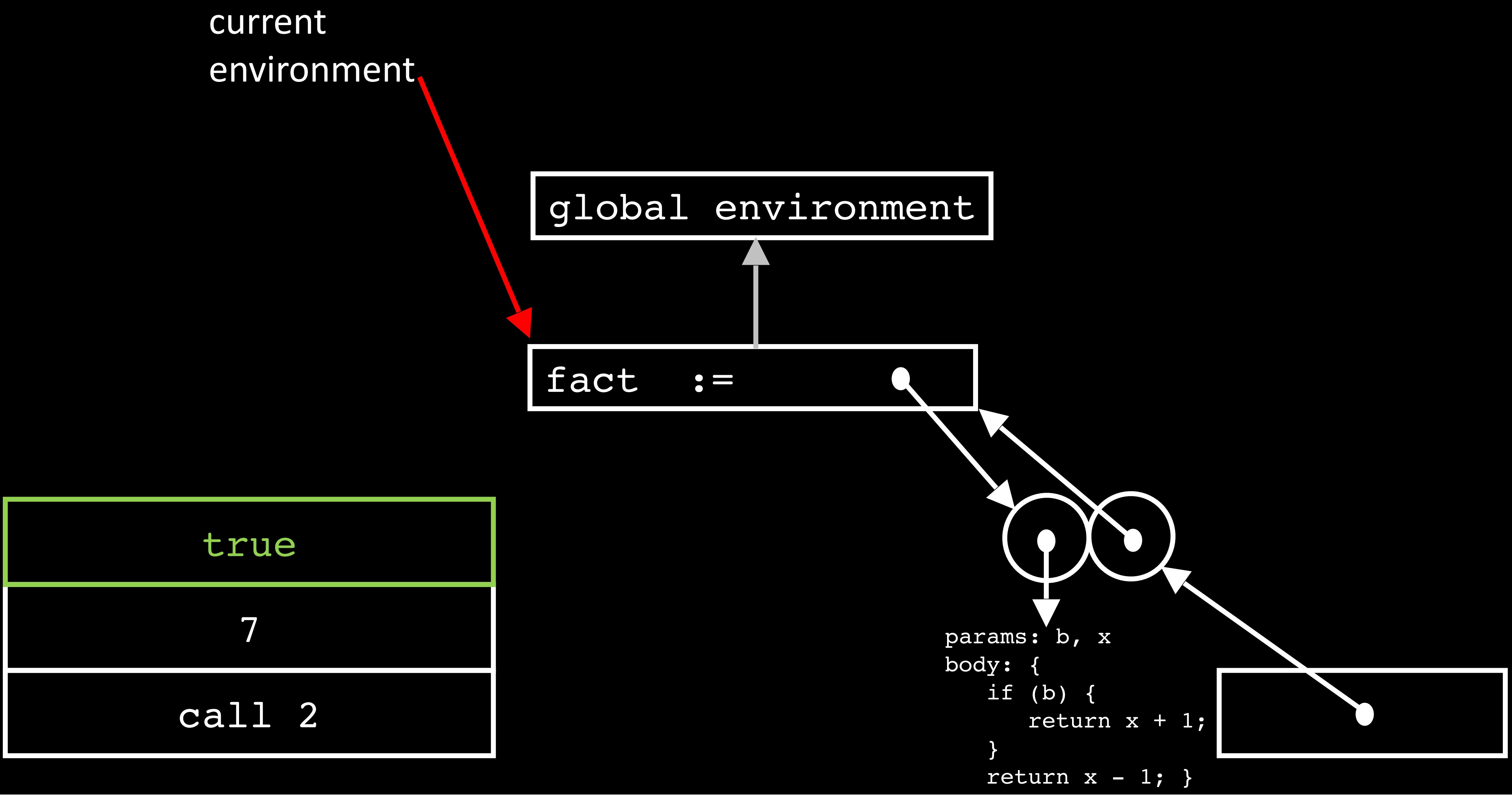
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



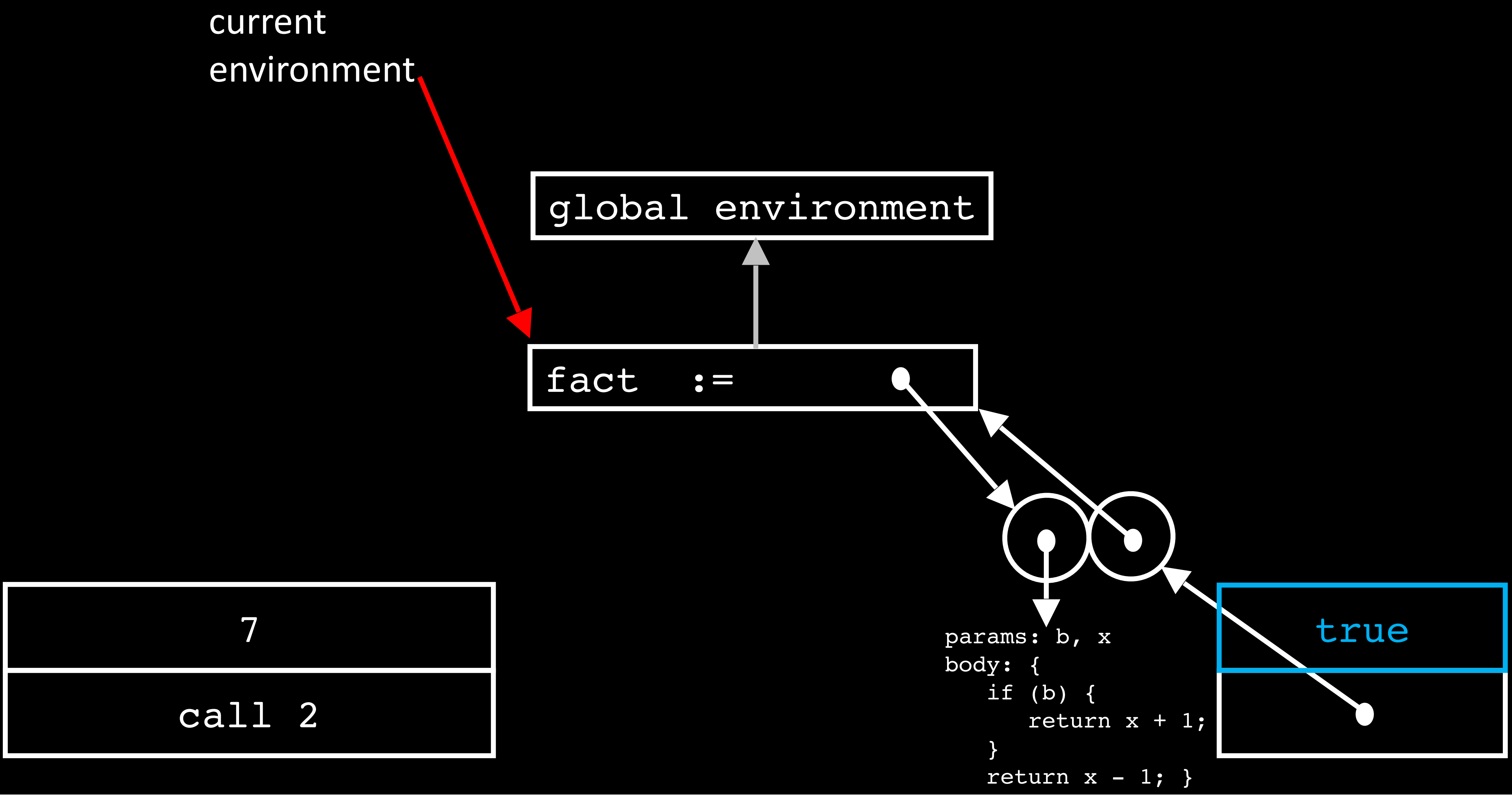
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



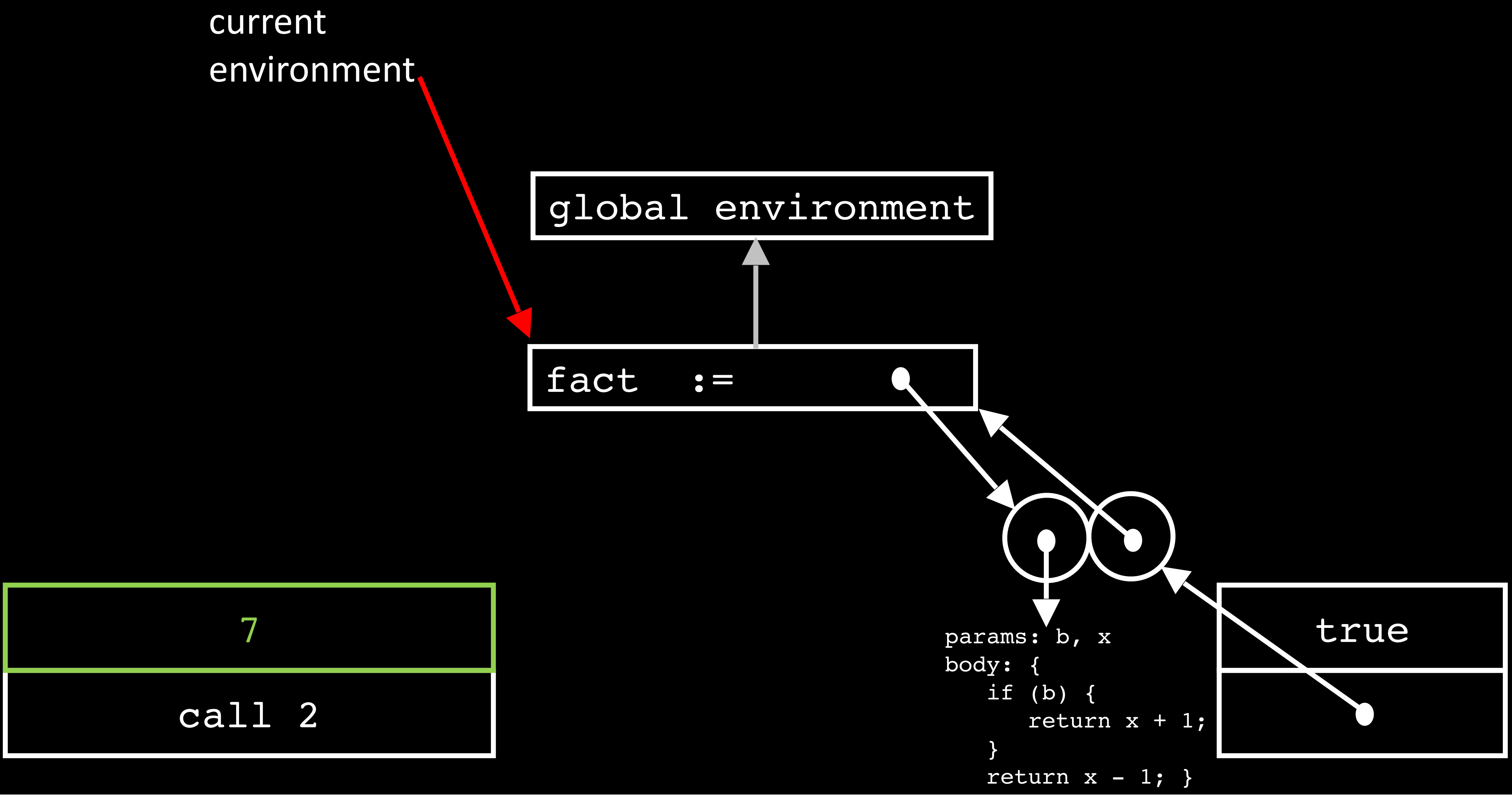
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



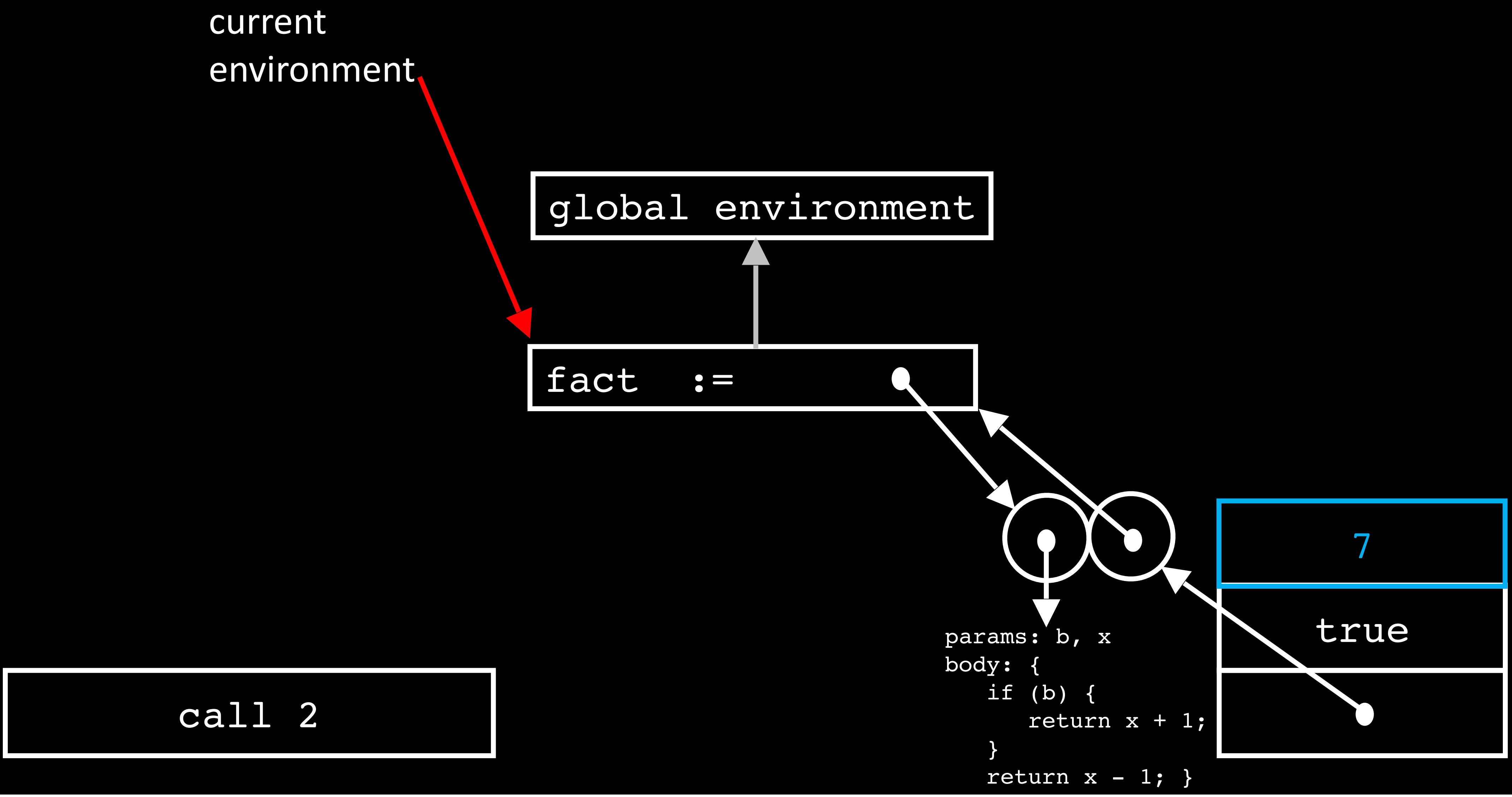
```
params: b, x
body: {
   if (b) {
      return x + 1;
   }
   return x - 1; }
```



```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

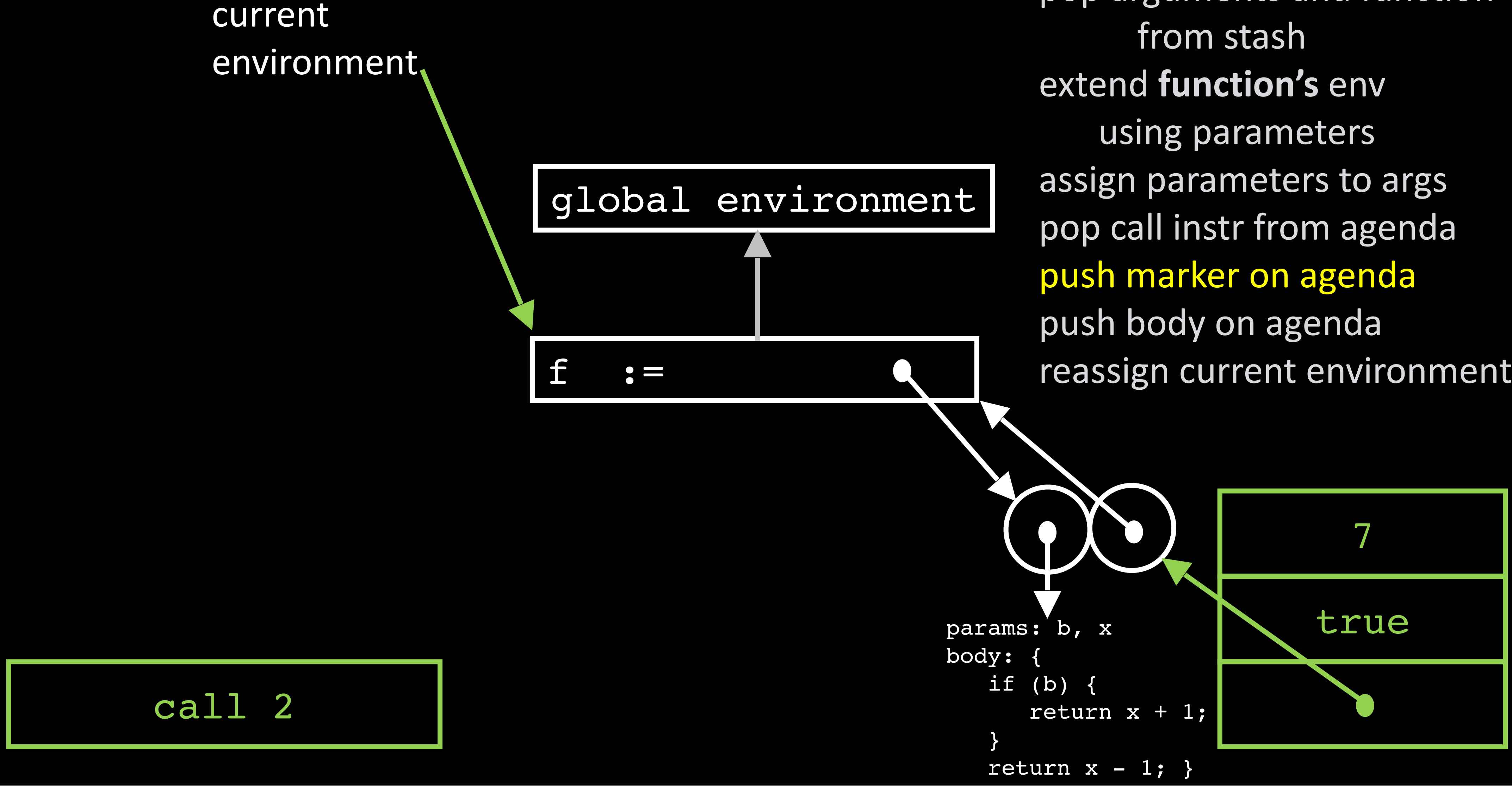


```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



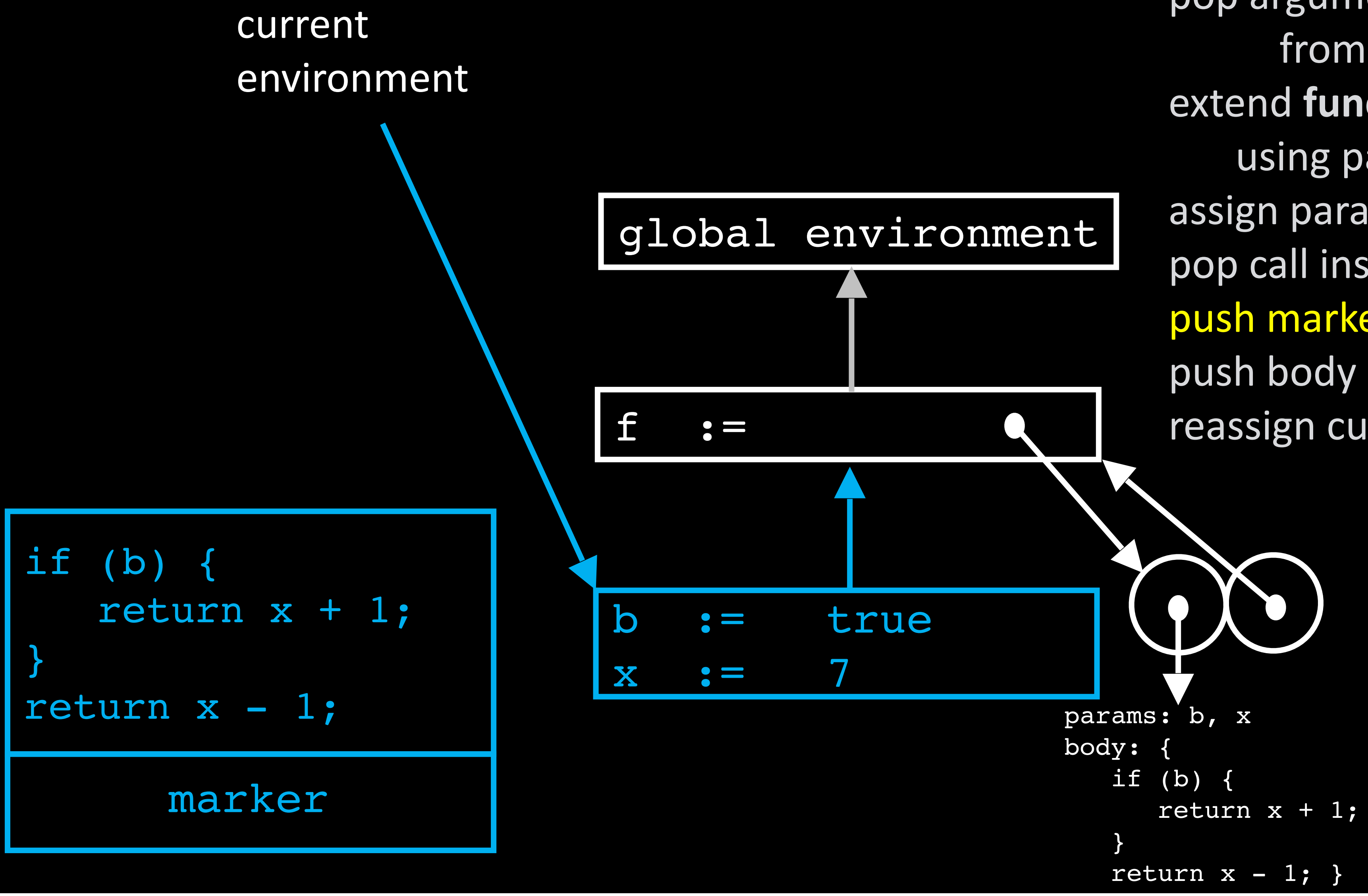
```
params: b, x
body: {
   if (b) {
      return x + 1;
   }
   return x - 1; }
```

Call instruction



Call instruction:
- pop arguments and function from stash
- extend **function's** env using parameters
- assign parameters to args
- pop call instr from agenda
- push marker on agenda
- push body on agenda
- reassign current environment

Call instruction

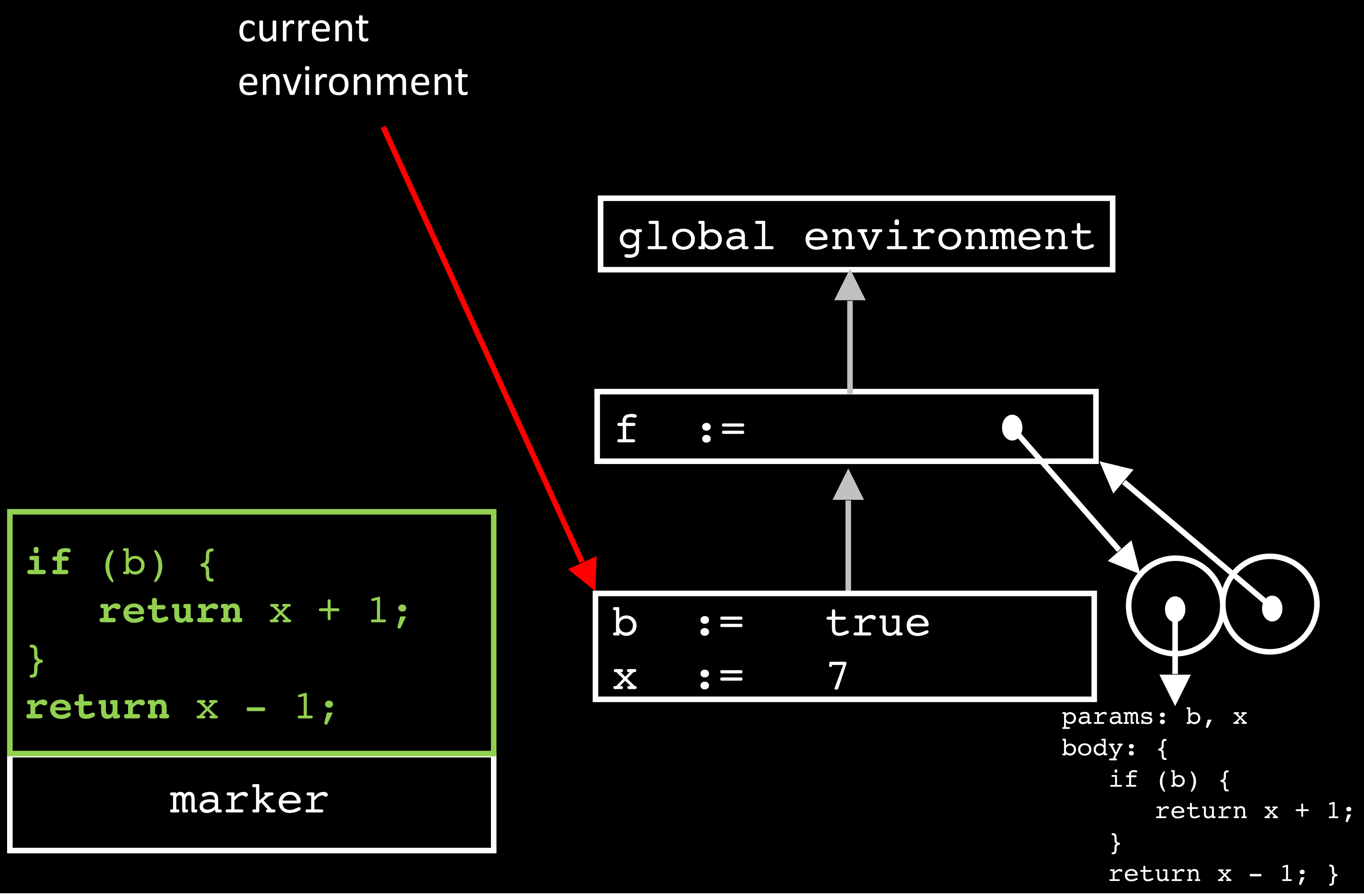


```
if (b) {
   return x + 1;
}
return x - 1;
```

```
params: b, x
body: {
   if (b) {
      return x + 1;
   }
   return x - 1; }
```

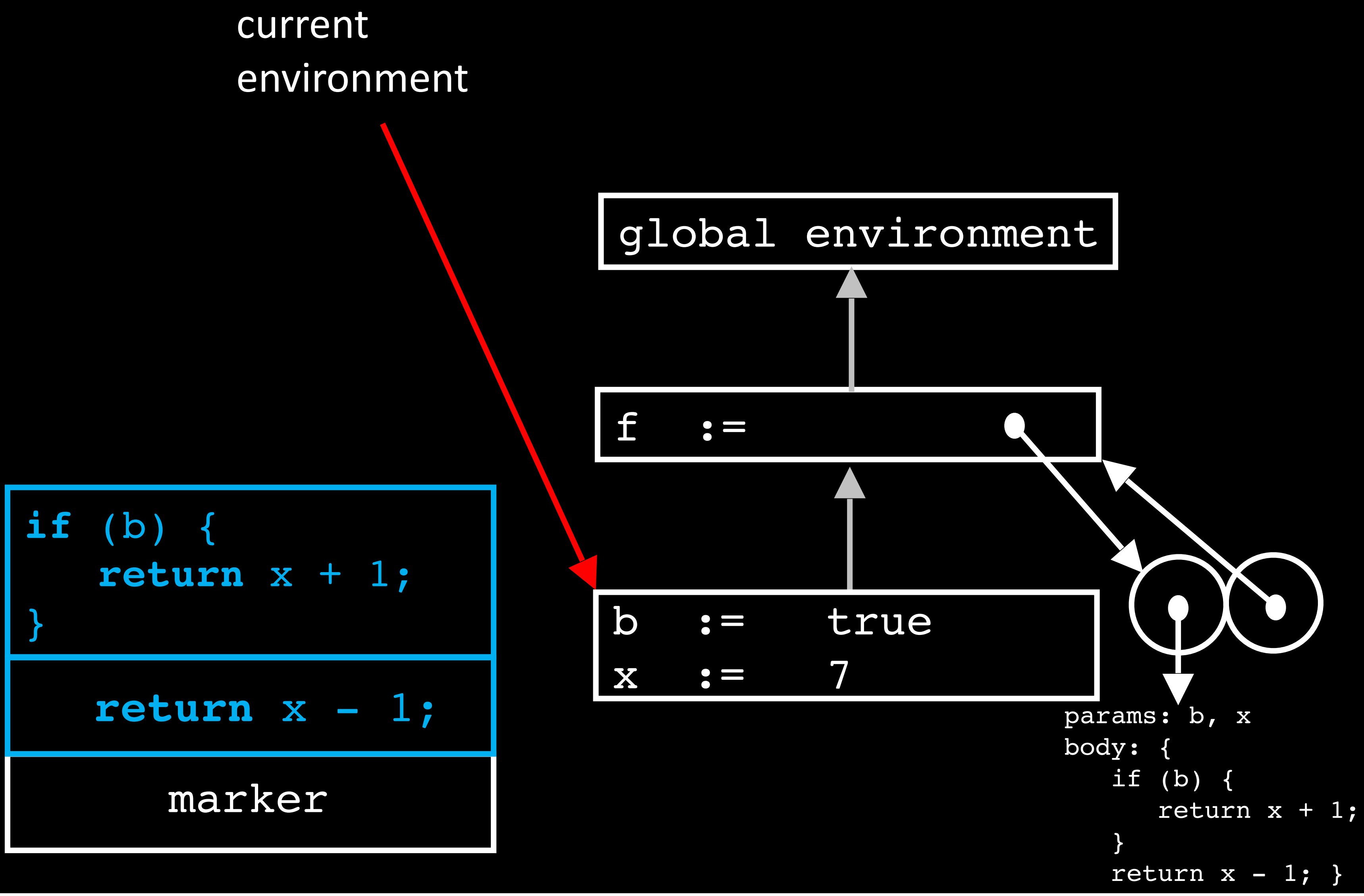
Call instruction:

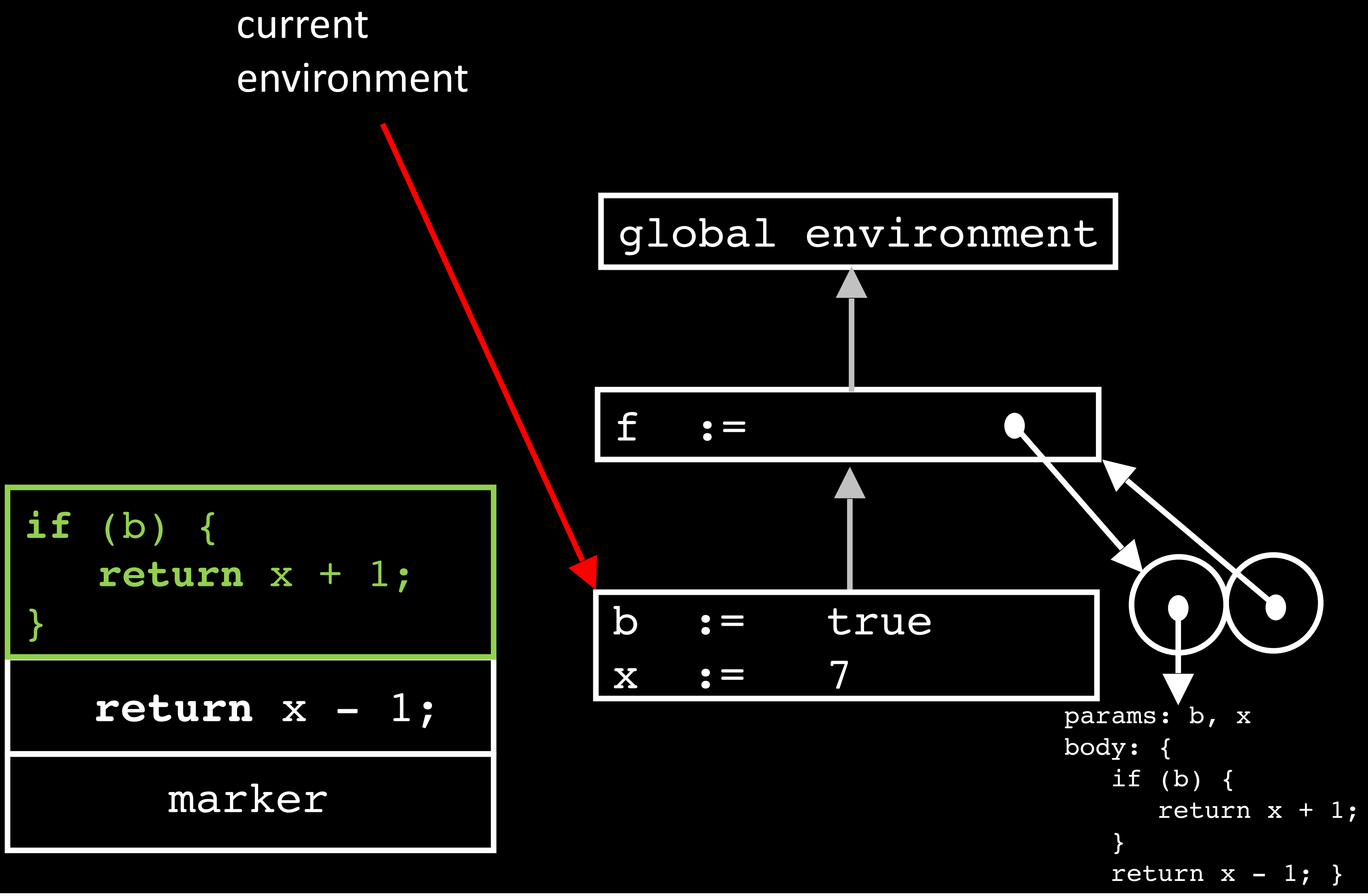
- pop arguments and function from stash
- extend **function's** env using parameters
- assign parameters to args
- pop call instr from agenda
- push marker on agenda
- push body on agenda
- reassign current environment



```
if (b) {
   return x + 1;
}
return x - 1;
```

```
params: b, x
body: {
   if (b) {
      return x + 1;
   }
   return x - 1; }
```

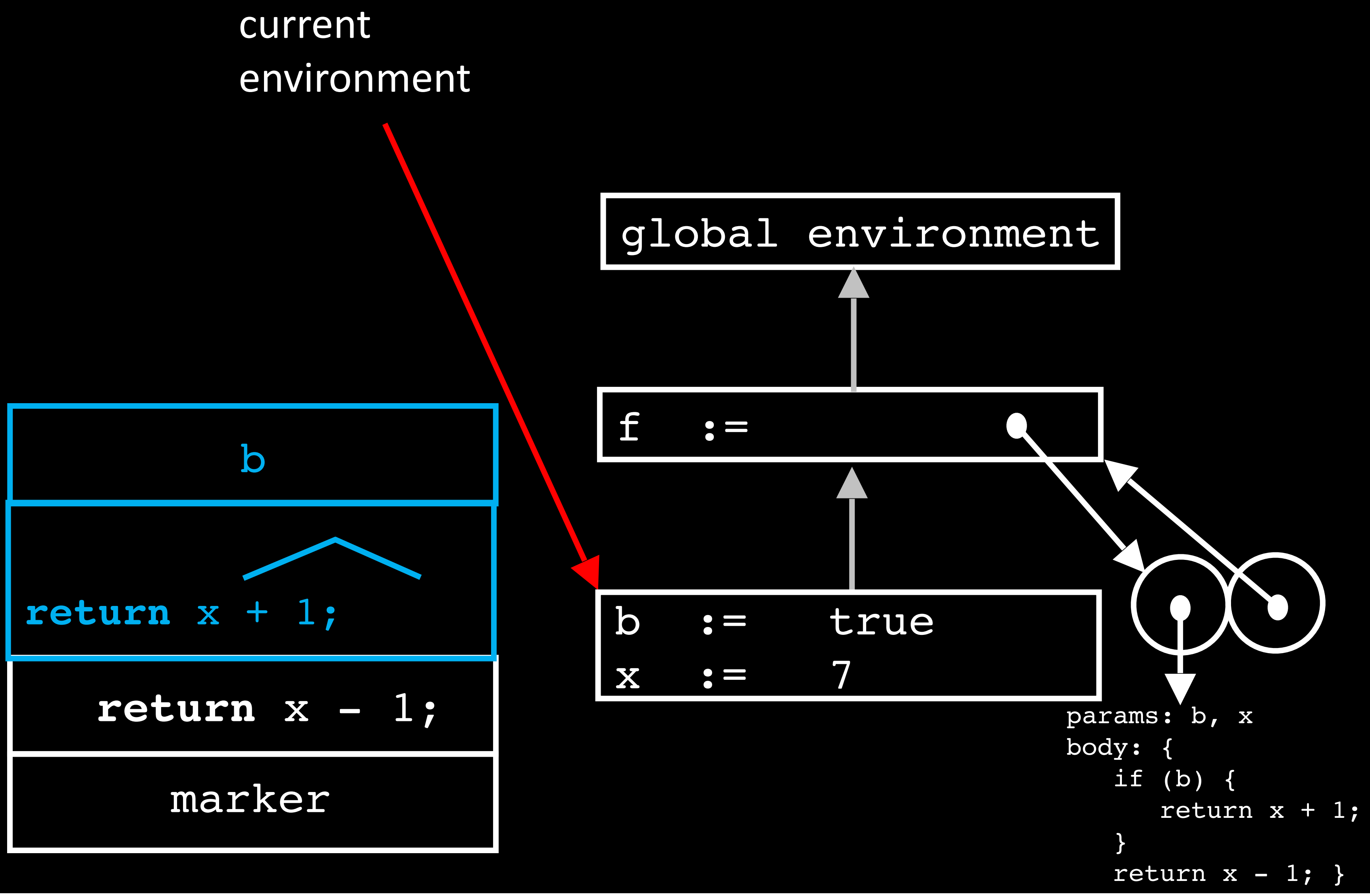

current
environment
global environment
f :=
b := true
x := 7
params: b, x
body: {
if (b) {
return x + 1;
}
return x - 1; }
if (b) {
return x + 1;
}
return x - 1;
marker



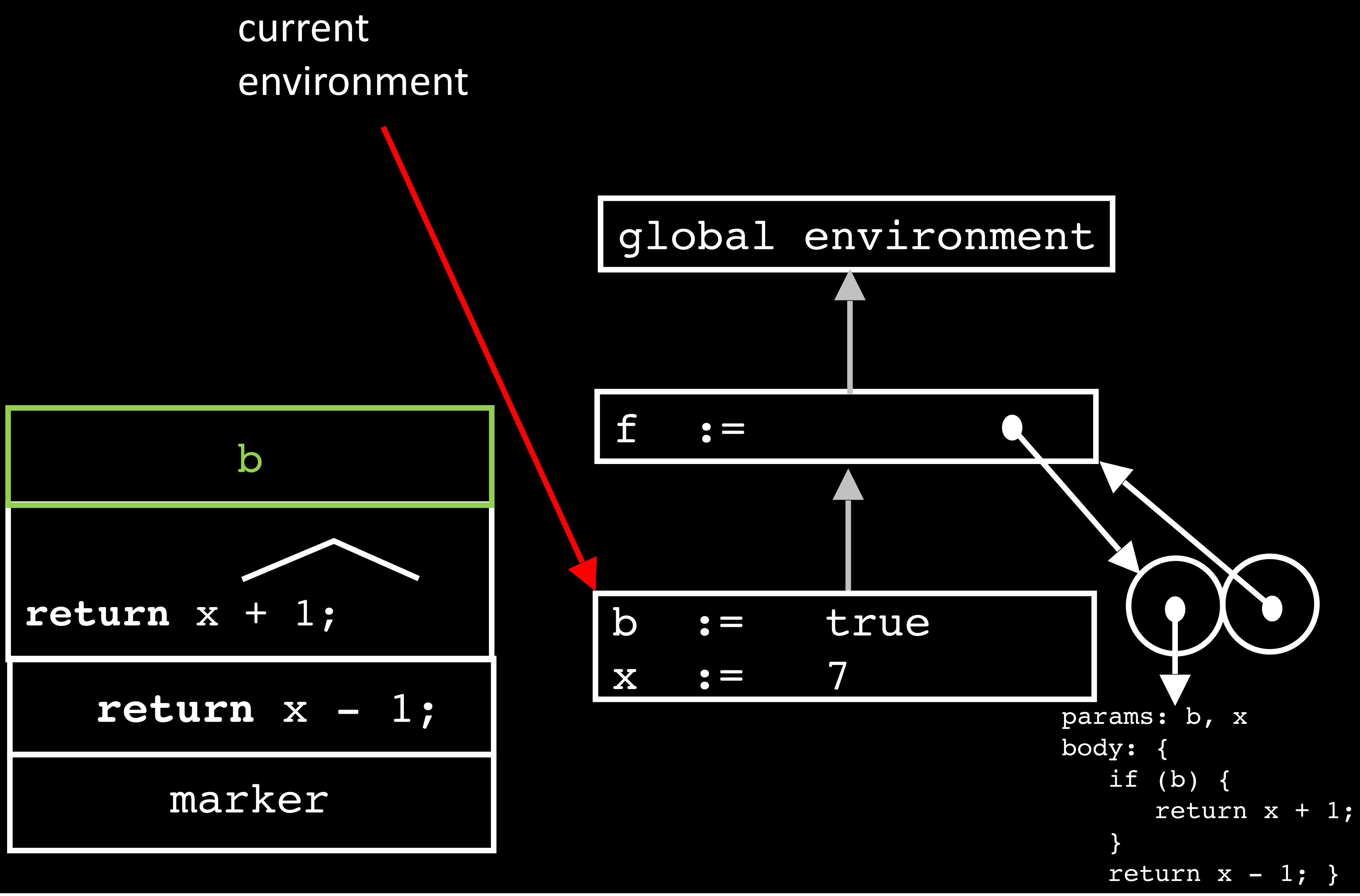
```
if (b) {
   return x + 1;
}
```

```
return x - 1;
```

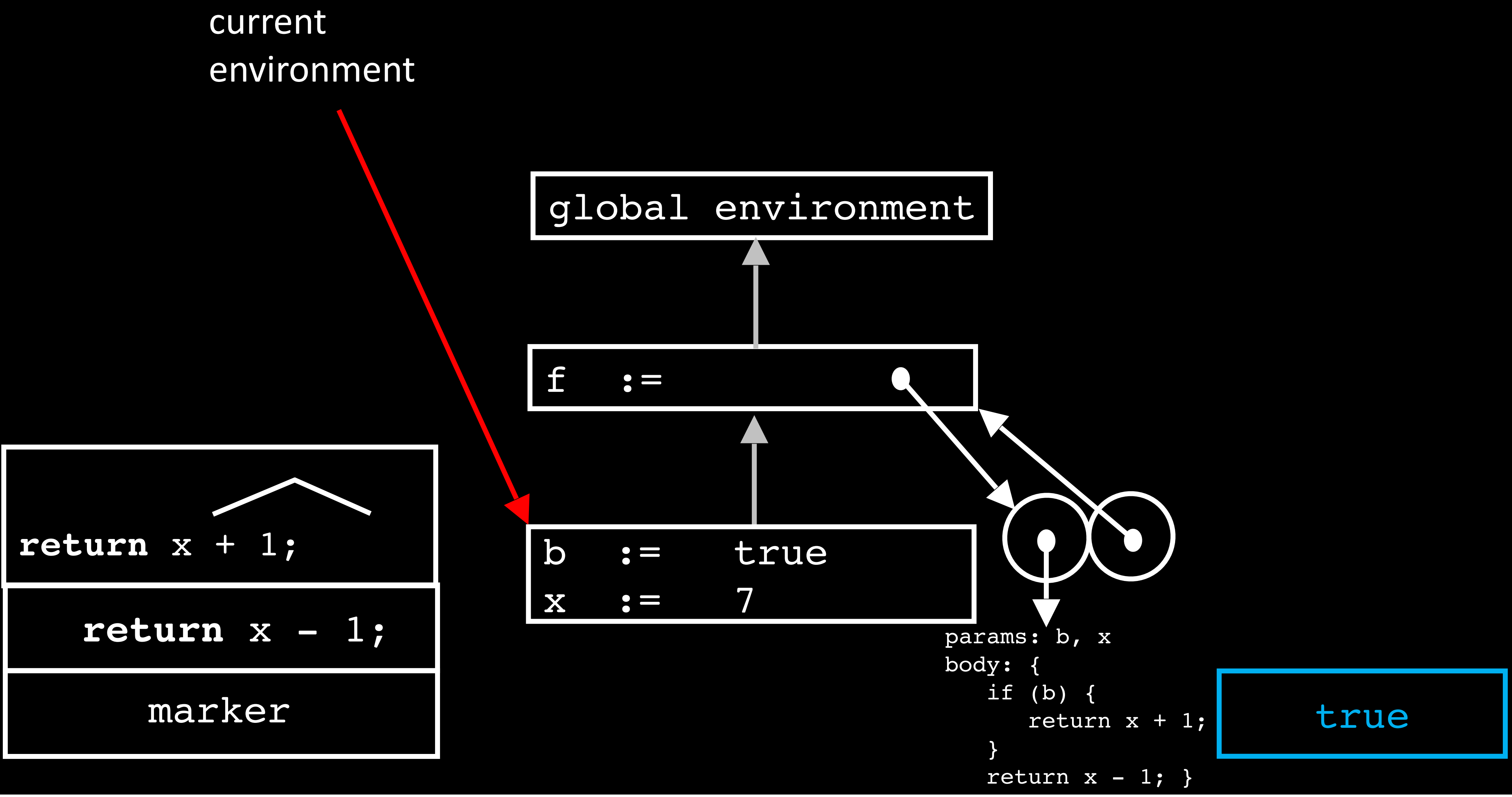
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



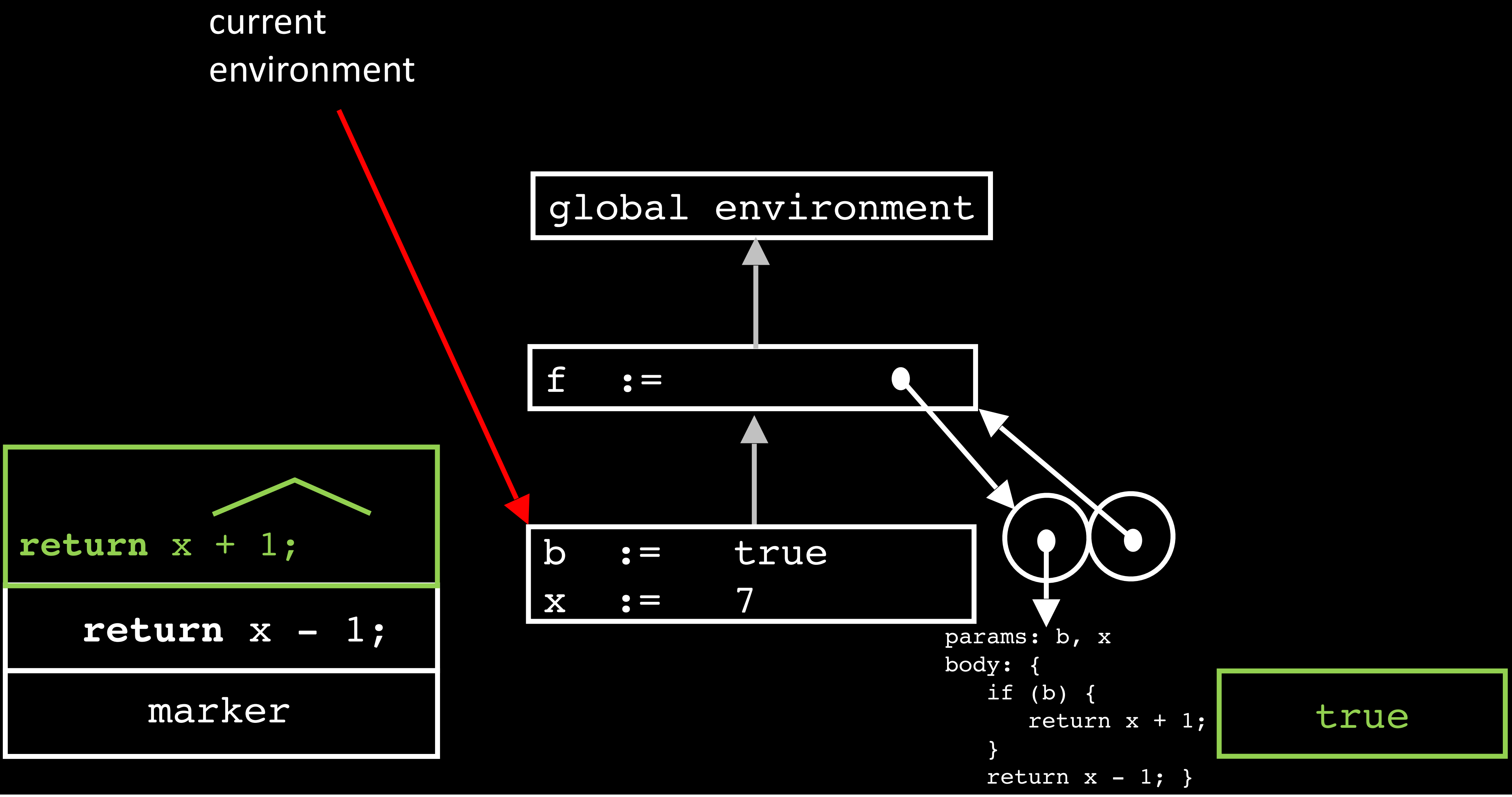
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



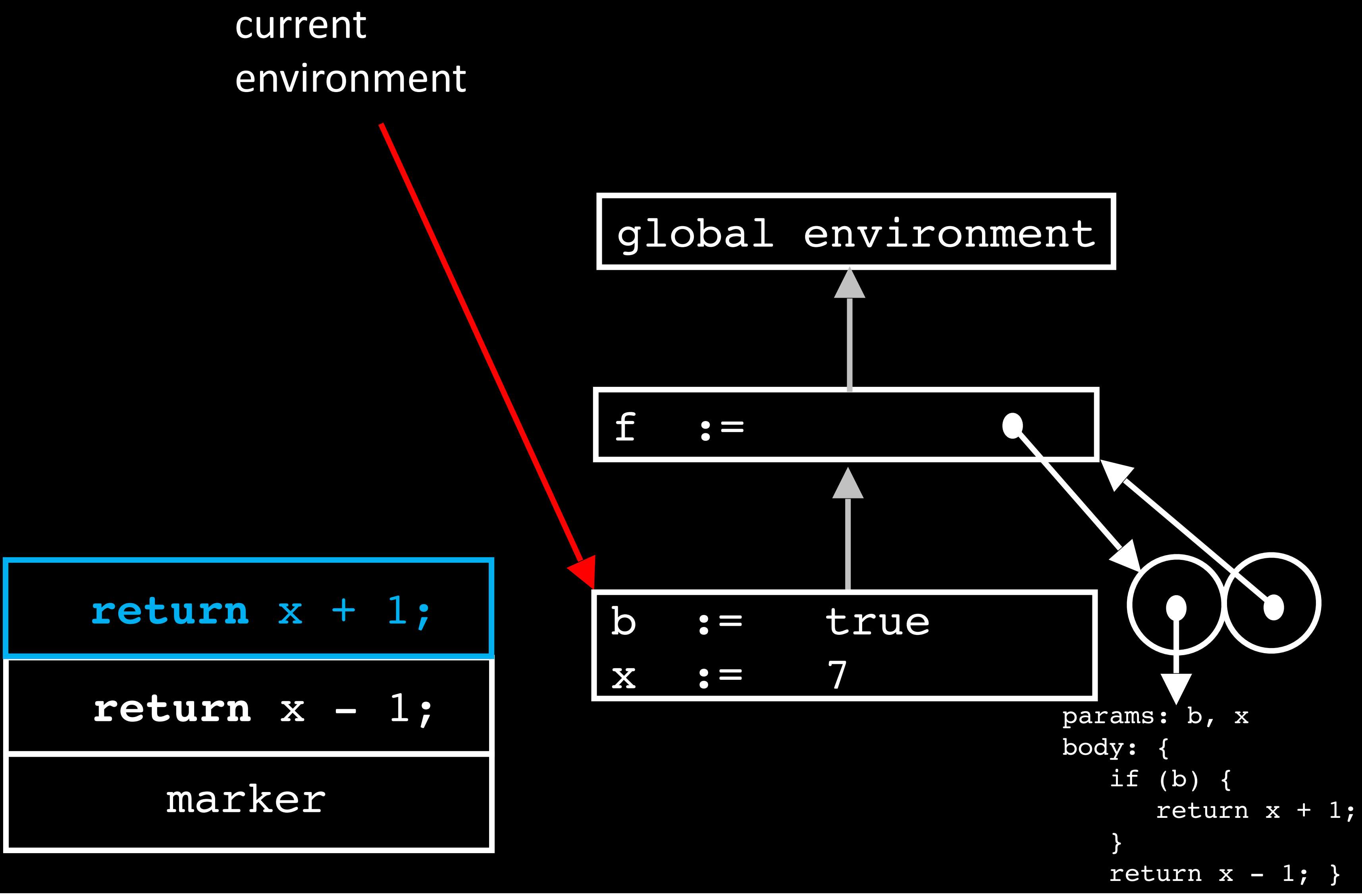
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

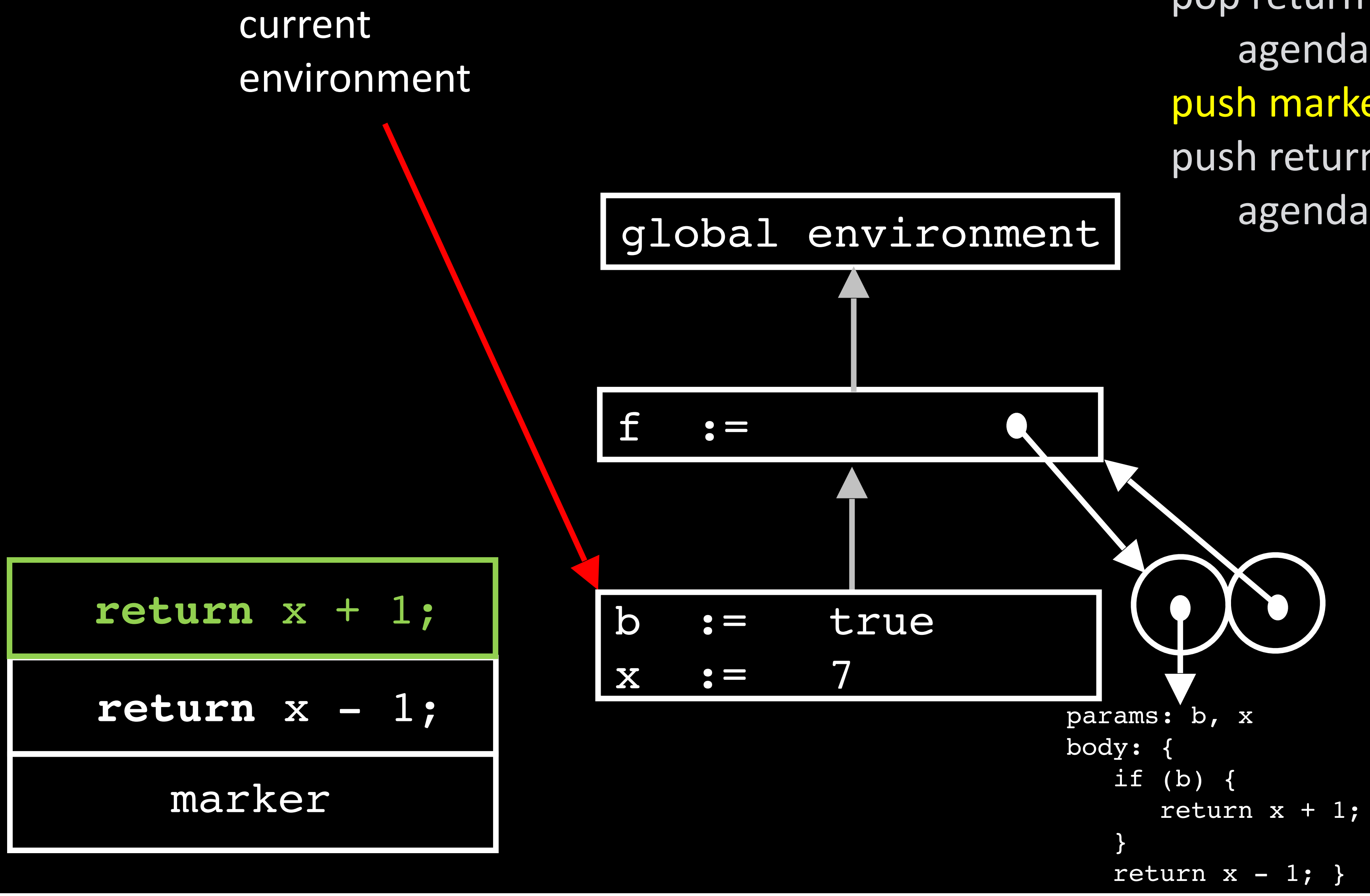


```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

Return statement

Return statement:

- pop return statement from agenda
- push marker on agenda
- push return expression on agenda

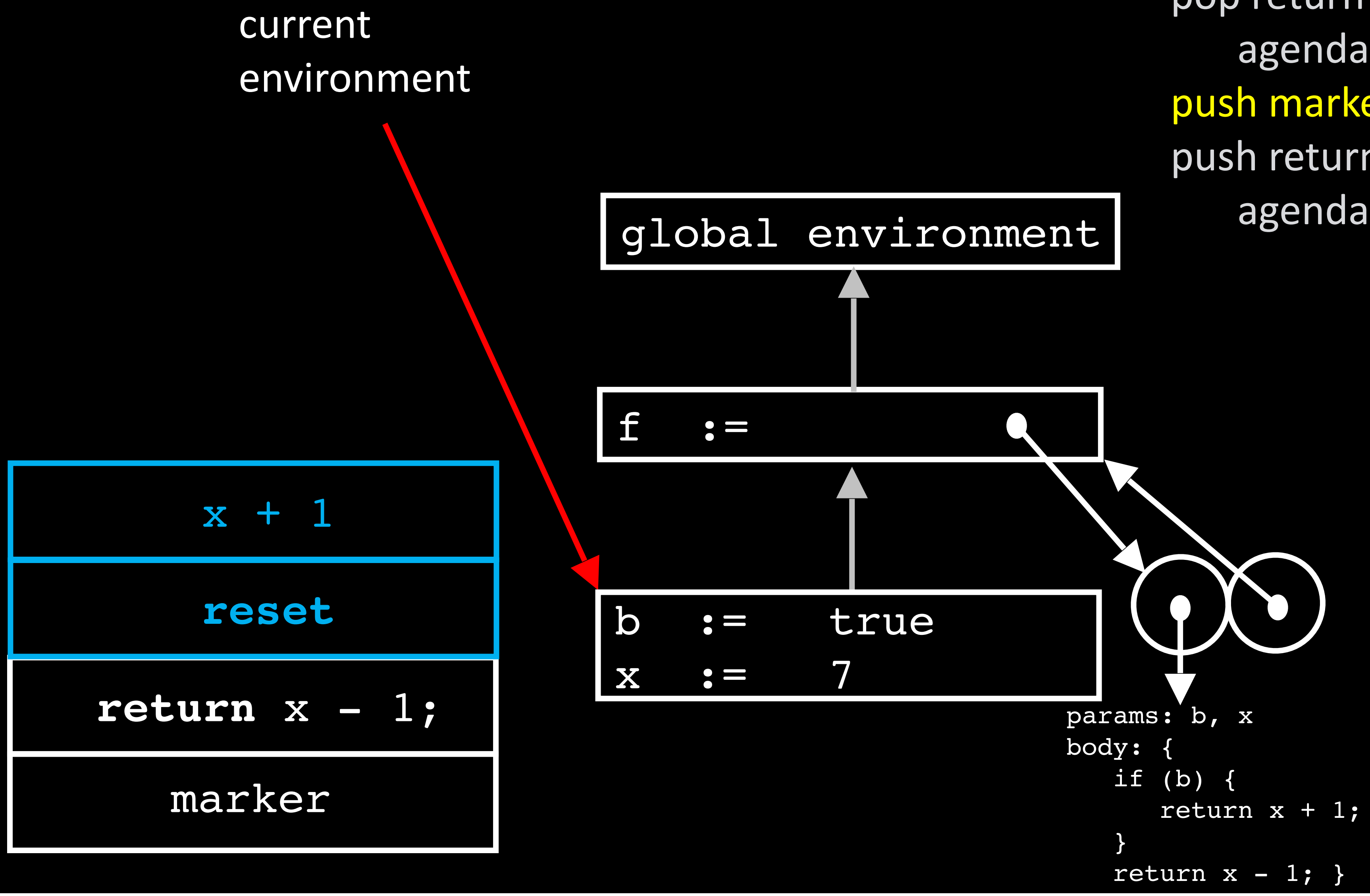


```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

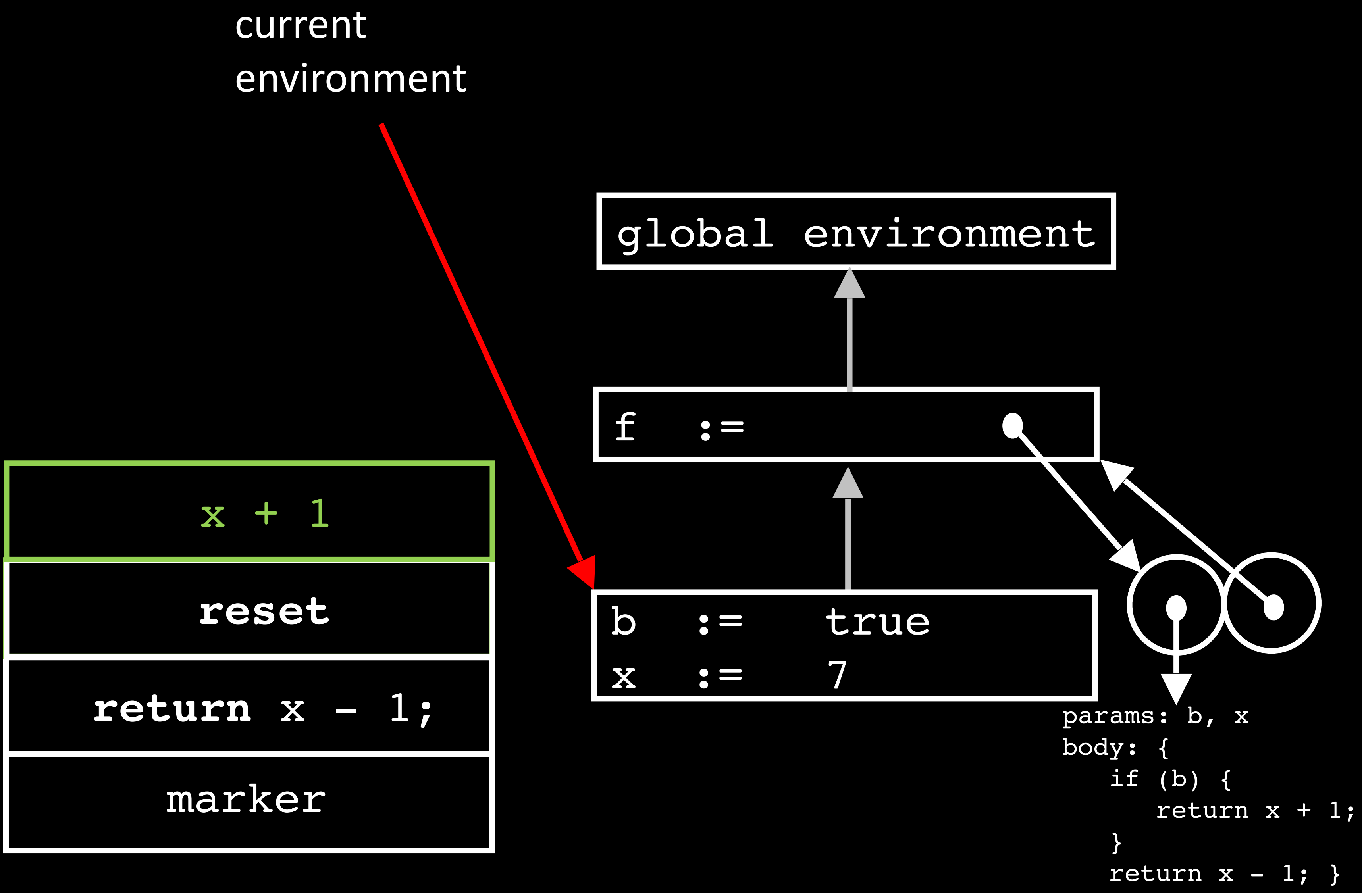
Return statement

Return statement:

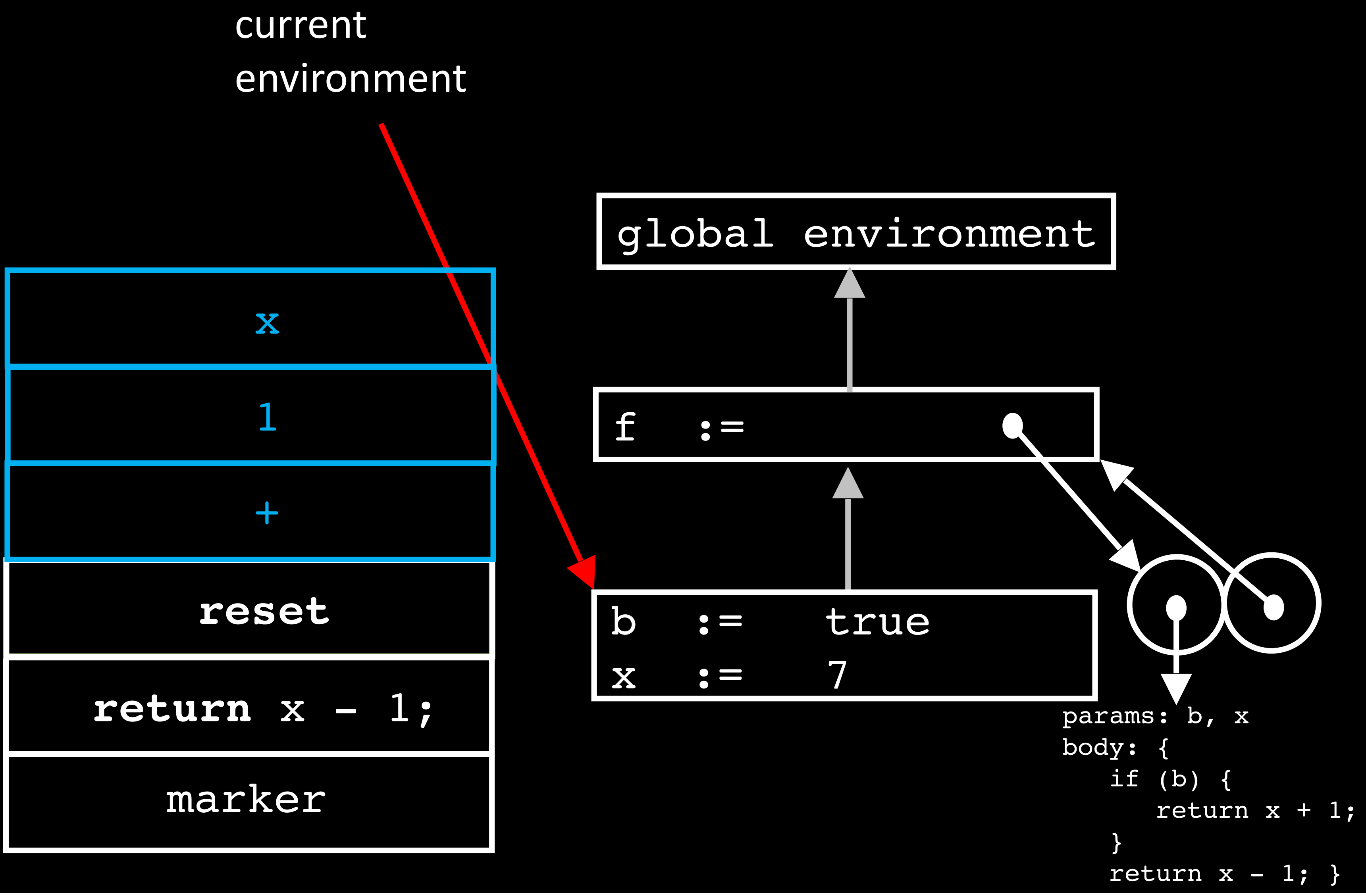
- pop return statement from agenda
- push marker on agenda
- push return expression on agenda



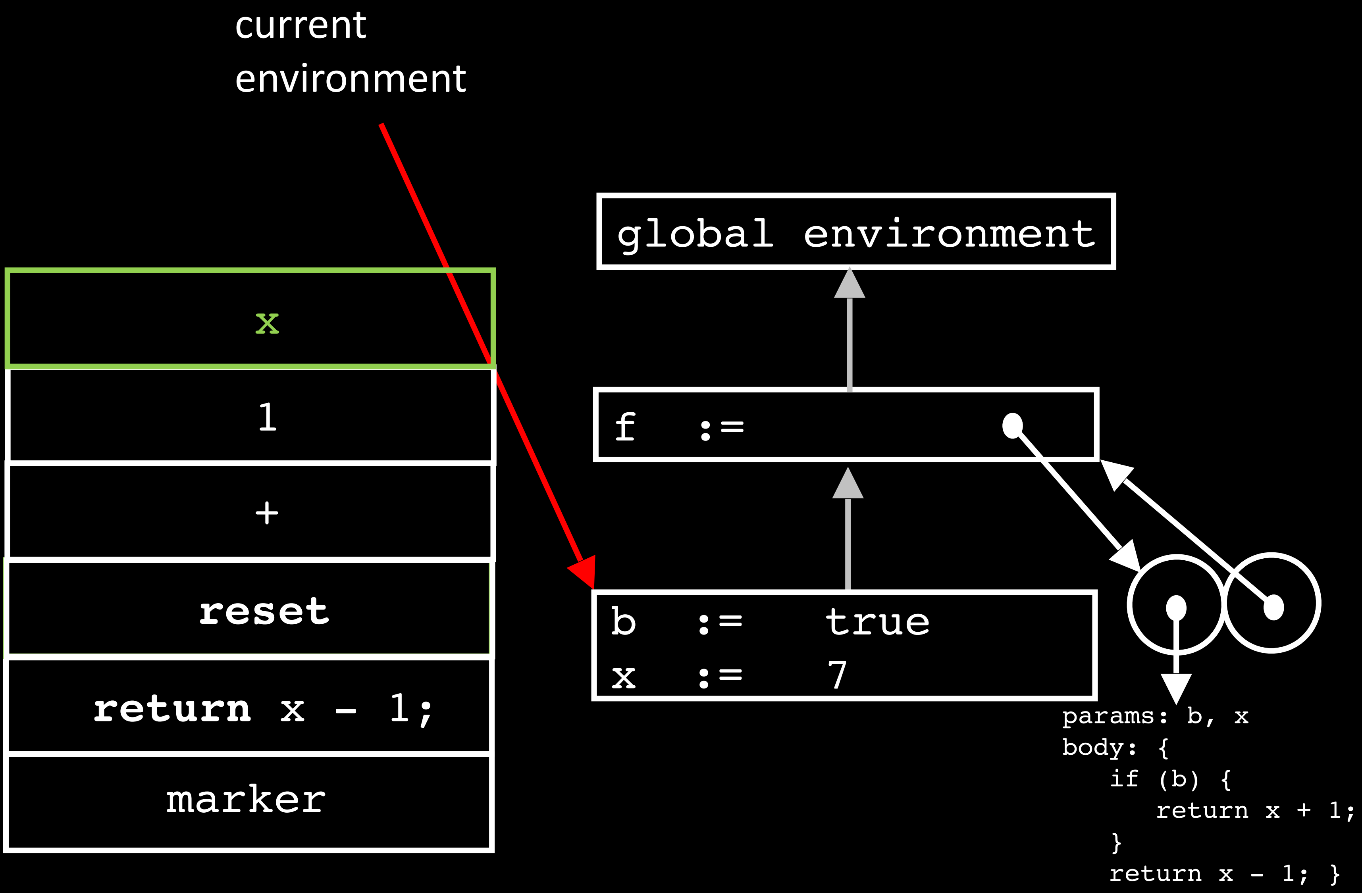
```
params: b, x
body: {
    if (b) {
       return x + 1;
    }
    return x - 1; }
```



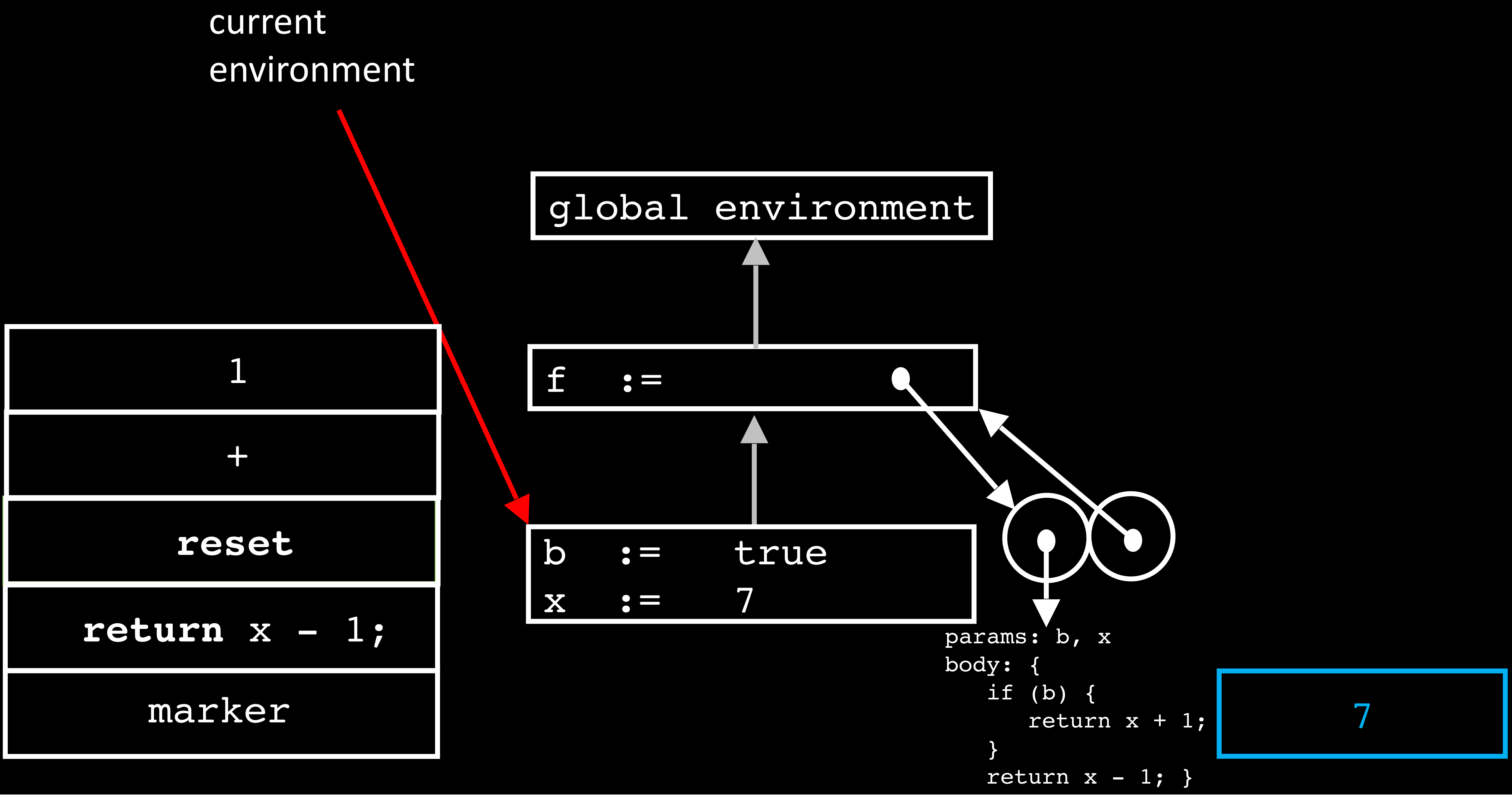
```
params: b, x
body: {
    if (b) {
       return x + 1;
    }
    return x - 1; }
```



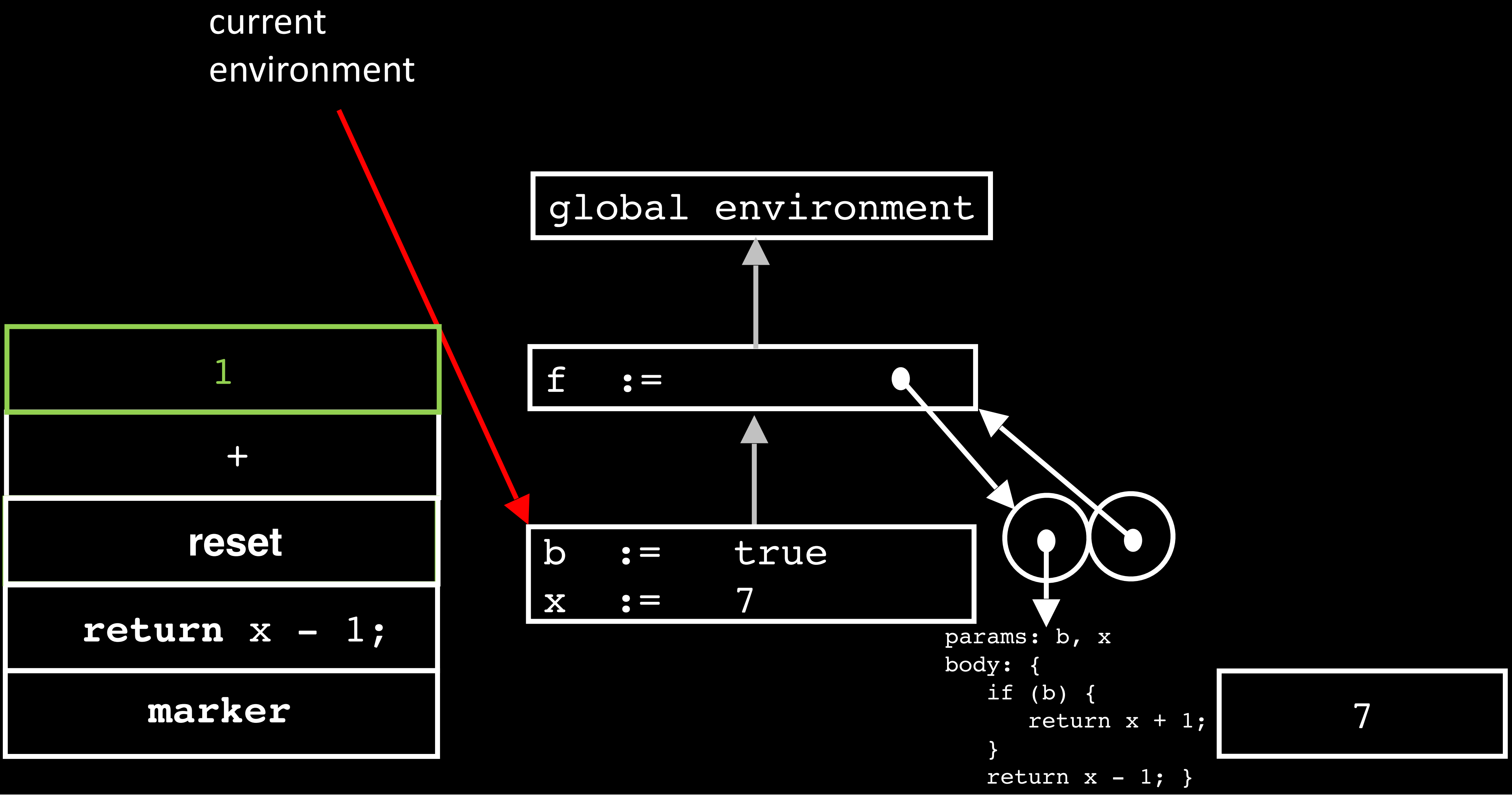
```
params: b, x
body: {
    if (b) {
       return x + 1;
    }
    return x - 1; }
```



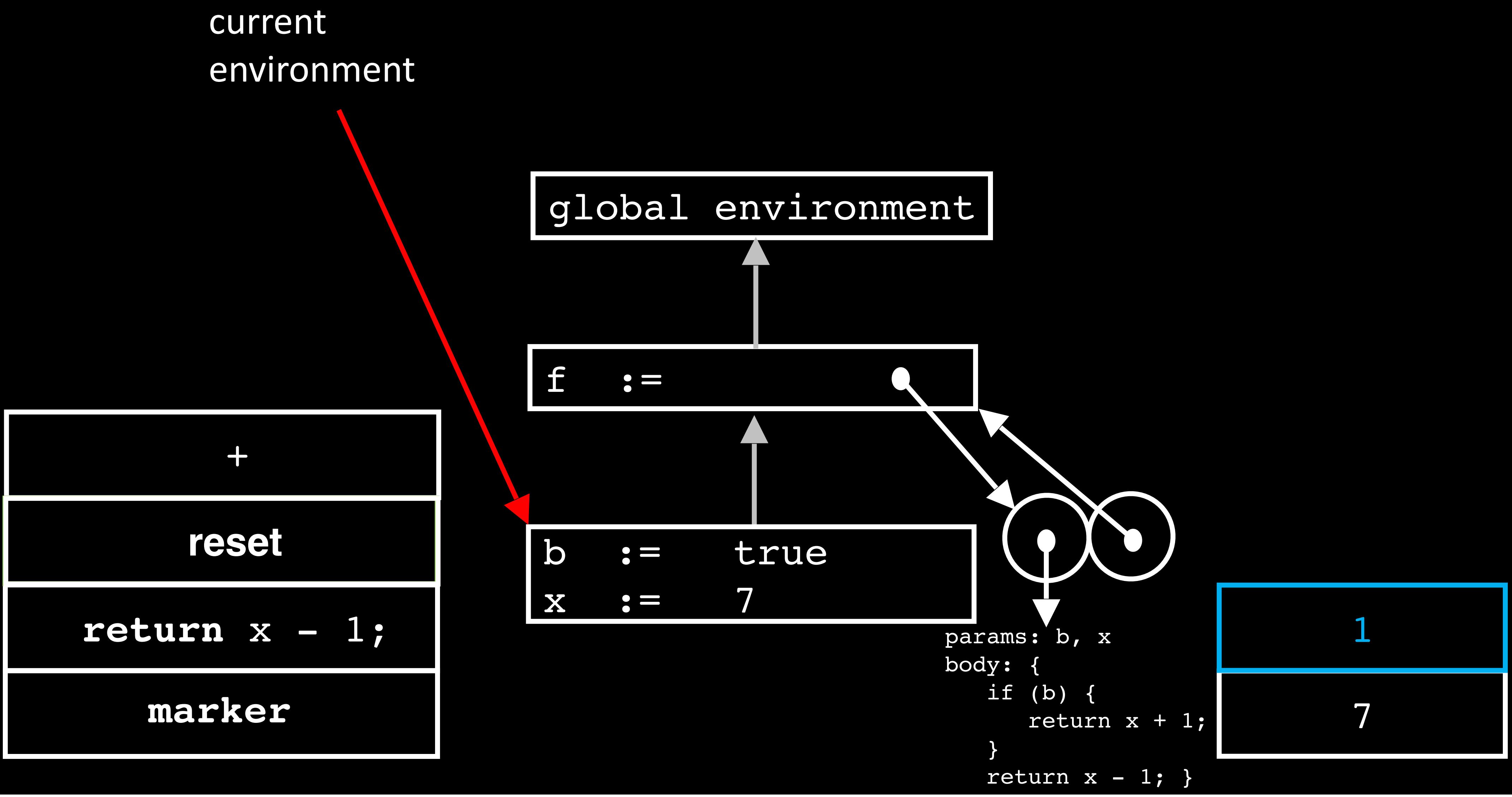
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



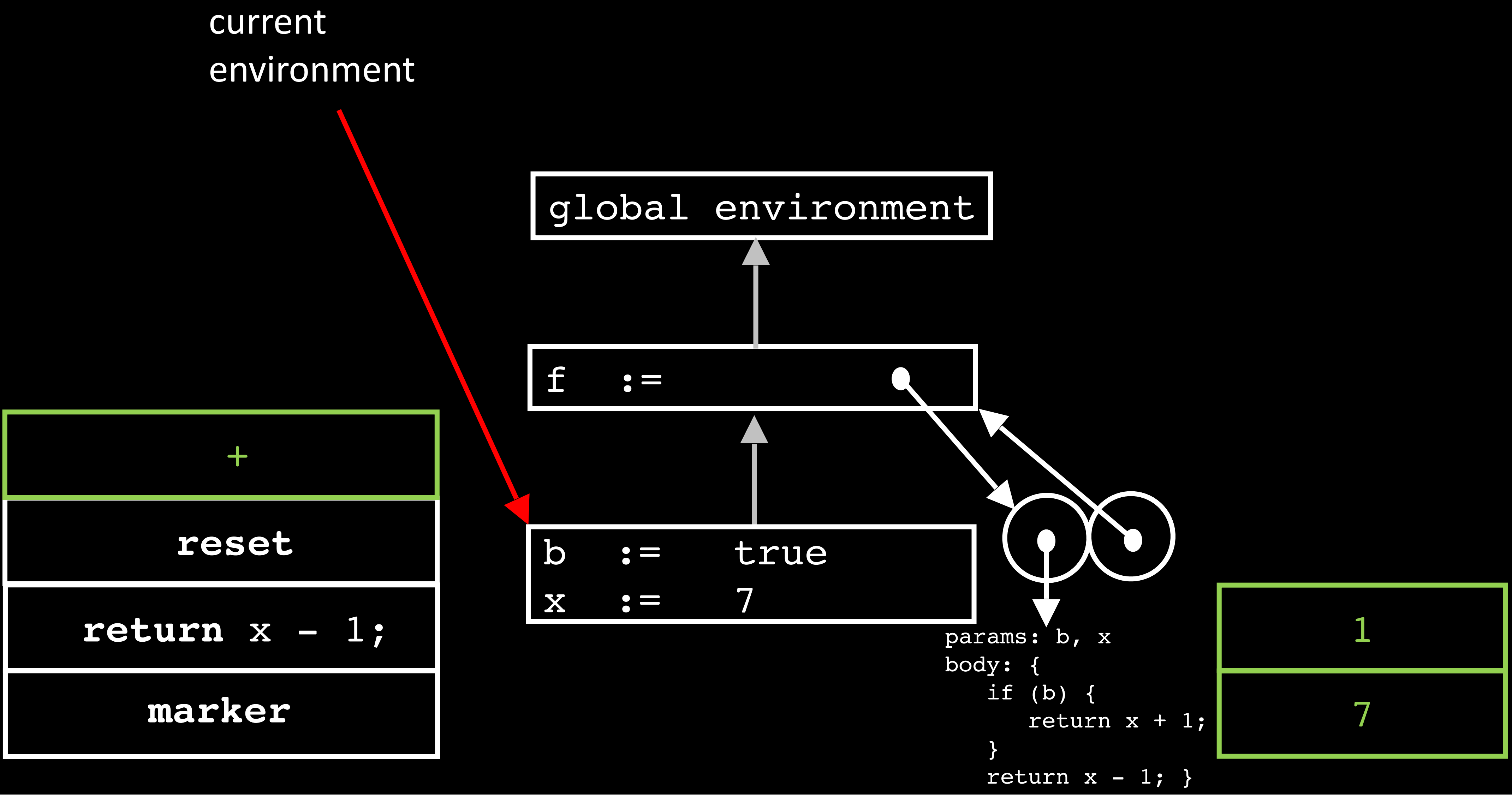
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



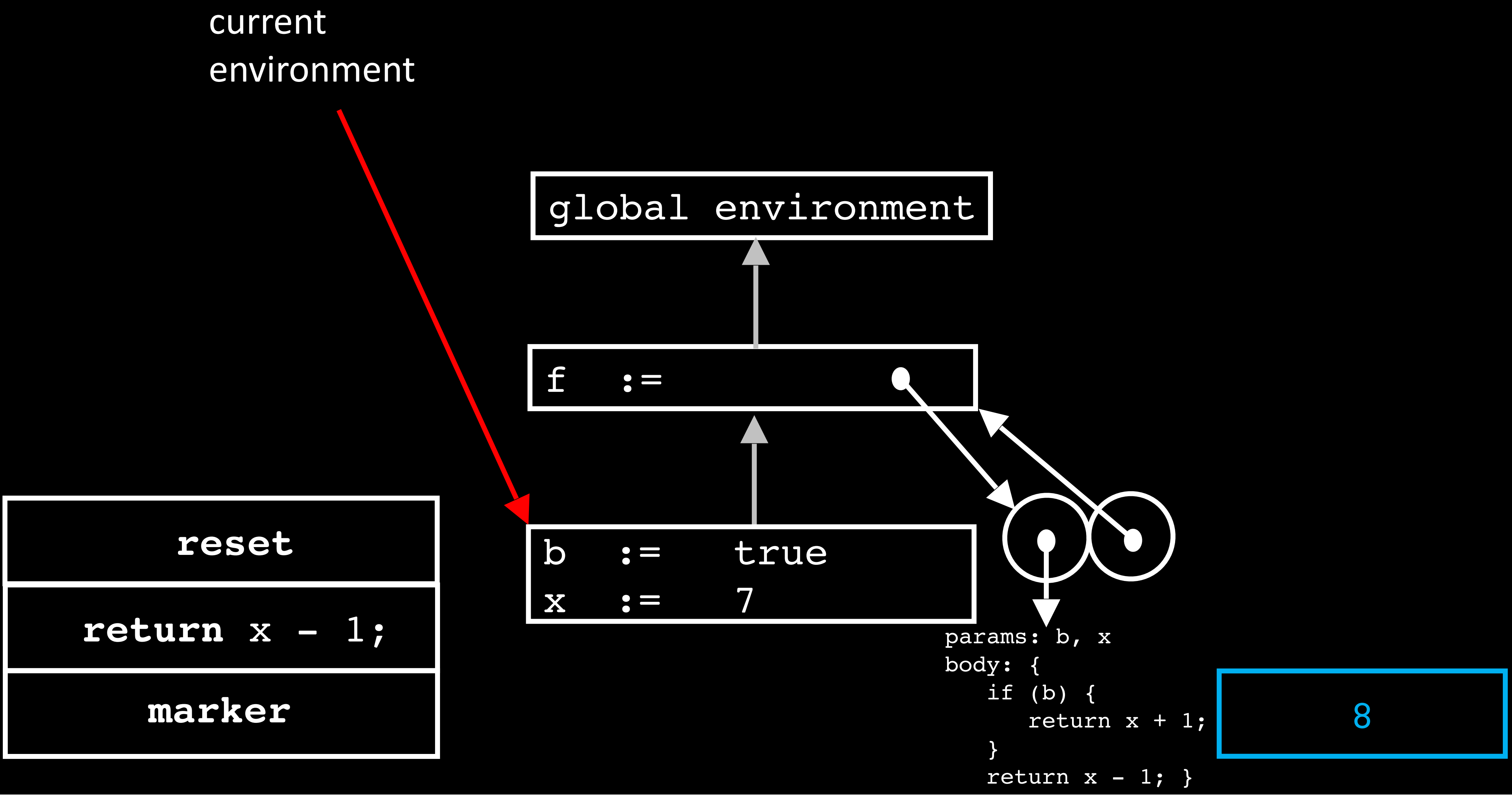
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```



```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

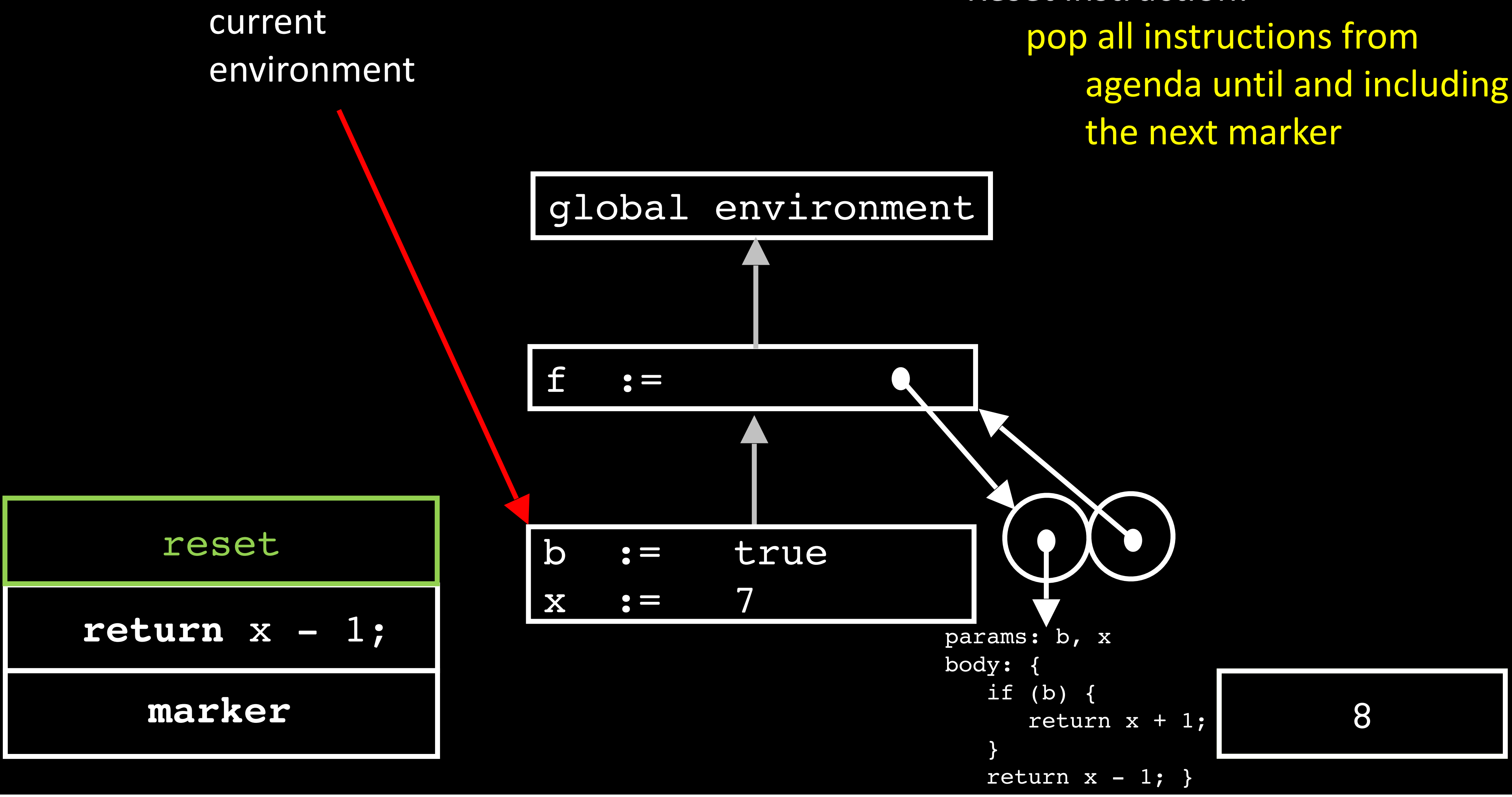


```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

Reset instruction

Reset instruction:

pop all instructions from agenda until and including the next marker



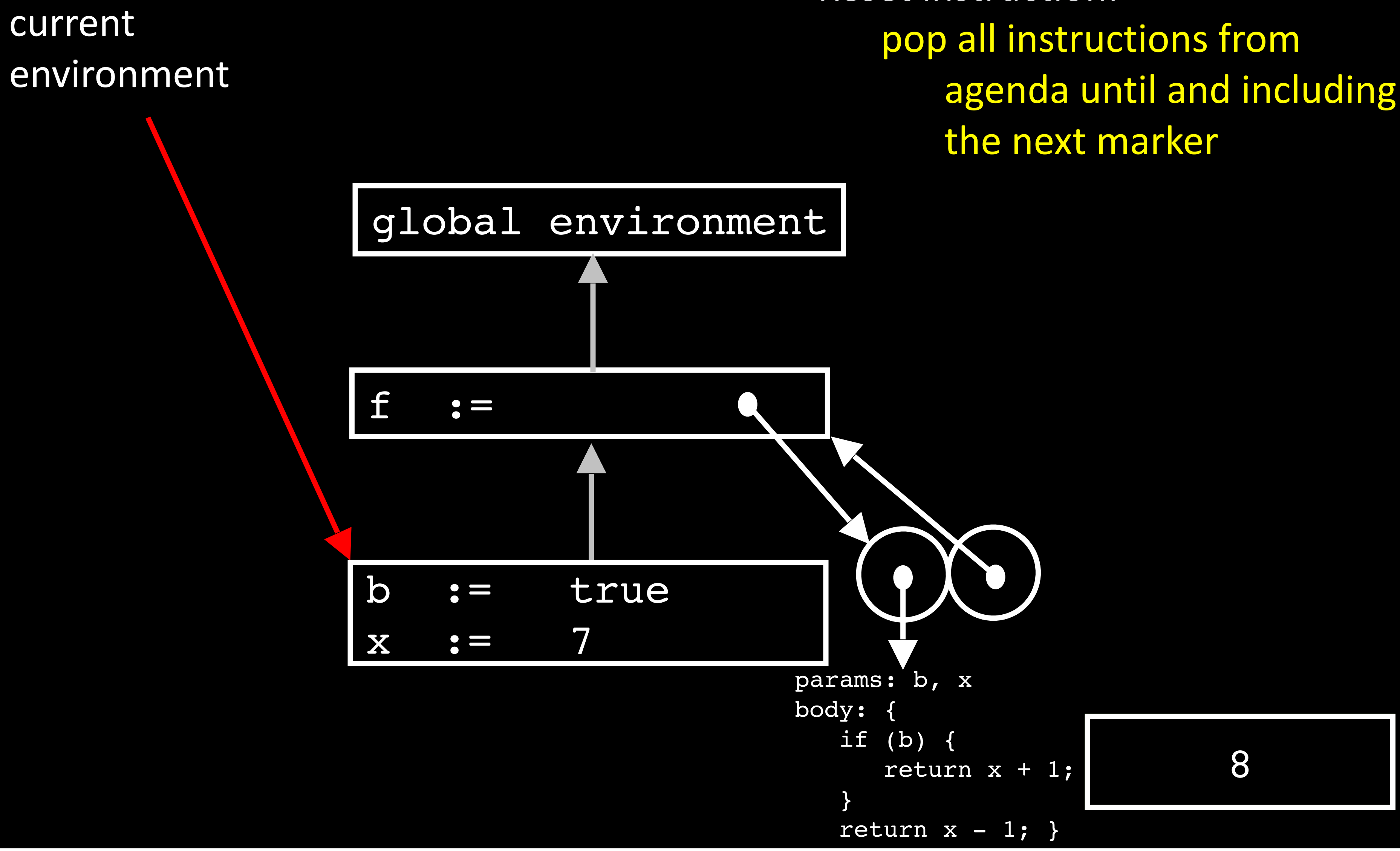
```
params: b, x
body: {
    if (b) {
        return x + 1;
    }
    return x - 1; }
```

Reset instruction

Done!

Reset instruction:

pop all instructions from agenda until and including the next marker



Iterative Processes

Iterative Processes

- Some markers and restore instructions are not needed when the function gives rise to an iterative process

Iterative Processes

- Some markers and restore instructions are not needed when the function gives rise to an iterative process
- Simple check suffices to avoid mark and restore

Iterative Processes

- Some markers and restore instructions are not needed when the function gives rise to an iterative process
- Simple check suffices to avoid mark and restore
- We say: Our notional machine is “naturally tail recursive”

Summary of Extended Environment Model

Summary of Extended Environment Model

- EEM captures full state of computation

Summary of Extended Environment Model

- EEM captures full state of computation
- Agenda keeps track of what actions to take in the future

Summary of Extended Environment Model

- EEM captures full state of computation
- Agenda keeps track of what actions to take in the future
- Stash keeps track of intermediate values in expression evaluation

Summary of Extended Environment Model

- EEM captures full state of computation
- Agenda keeps track of what actions to take in the future
- Stash keeps track of intermediate values in expression evaluation
- Environment keeps track of name bindings