NodeJS

środowisko i technologia ServerSide

PAWEŁ ŁUKASZUK

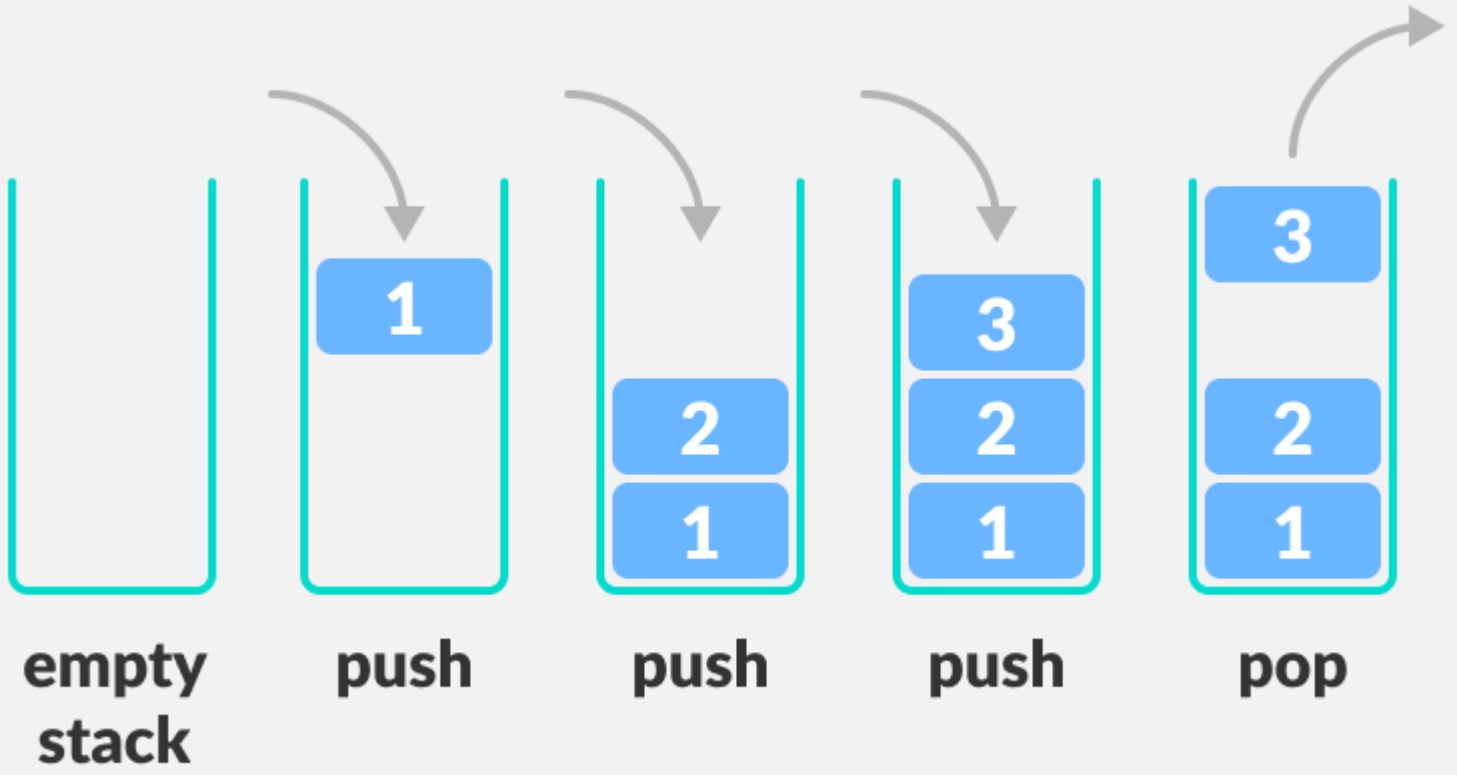

1064

Stack

In computer science, a stack is an abstract data structure that serves as a collection of elements, with two main principal operations:

- push, which adds an element to the collection
- pop, which removes the most recently added element that was not yet removed.

Stack



LIFO

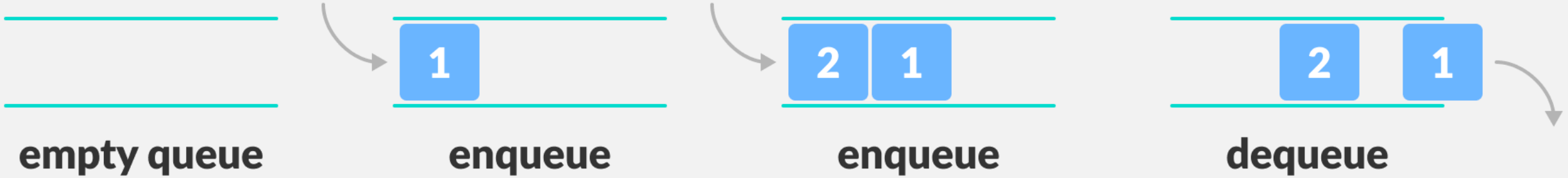
LAST IN – FIRST OUT

Queue

In computer science, a queue is an abstract data structure that serves as a collection of elements, with two main principal operations:

- enqueue, which adds an element to the end collection
- dequeue, which removes the last recently added element

Queue



FIFO

FIRST IN – FIRST OUT

Call stack

A call stack is a mechanism for an interpreter to keep track of its place in a script that calls multiple functions — what function is currently being run and what functions are called from within that function, etc.

- when a script calls a function, the interpreter adds it to the call stack and then starts carrying out the function
- any functions that are called by that function are added to the call stack further up, and run where their calls are reached
- when the current function is finished, the interpreter takes it off the stack and resumes execution where it left off in the last code listing

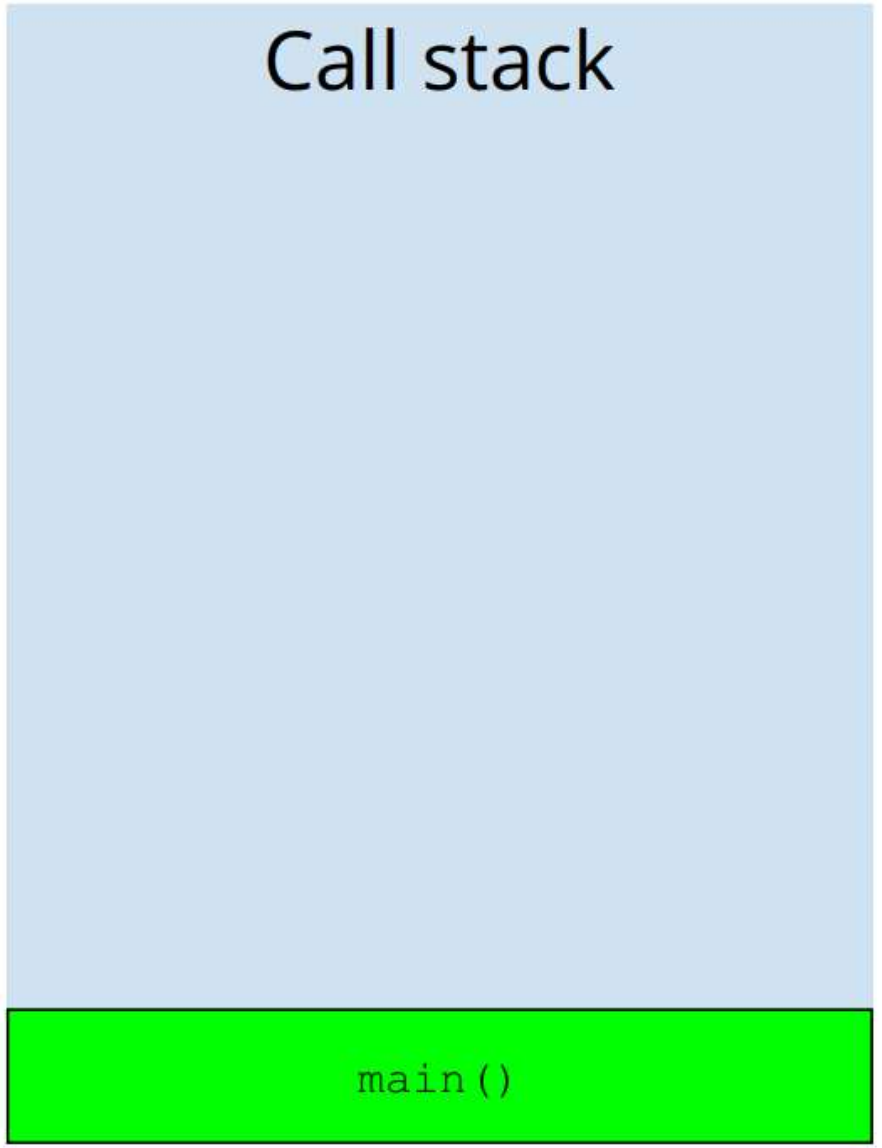
Stack – basic example

```
const a = 3;

const b = a + 4;

console.log('b = ', b);
```

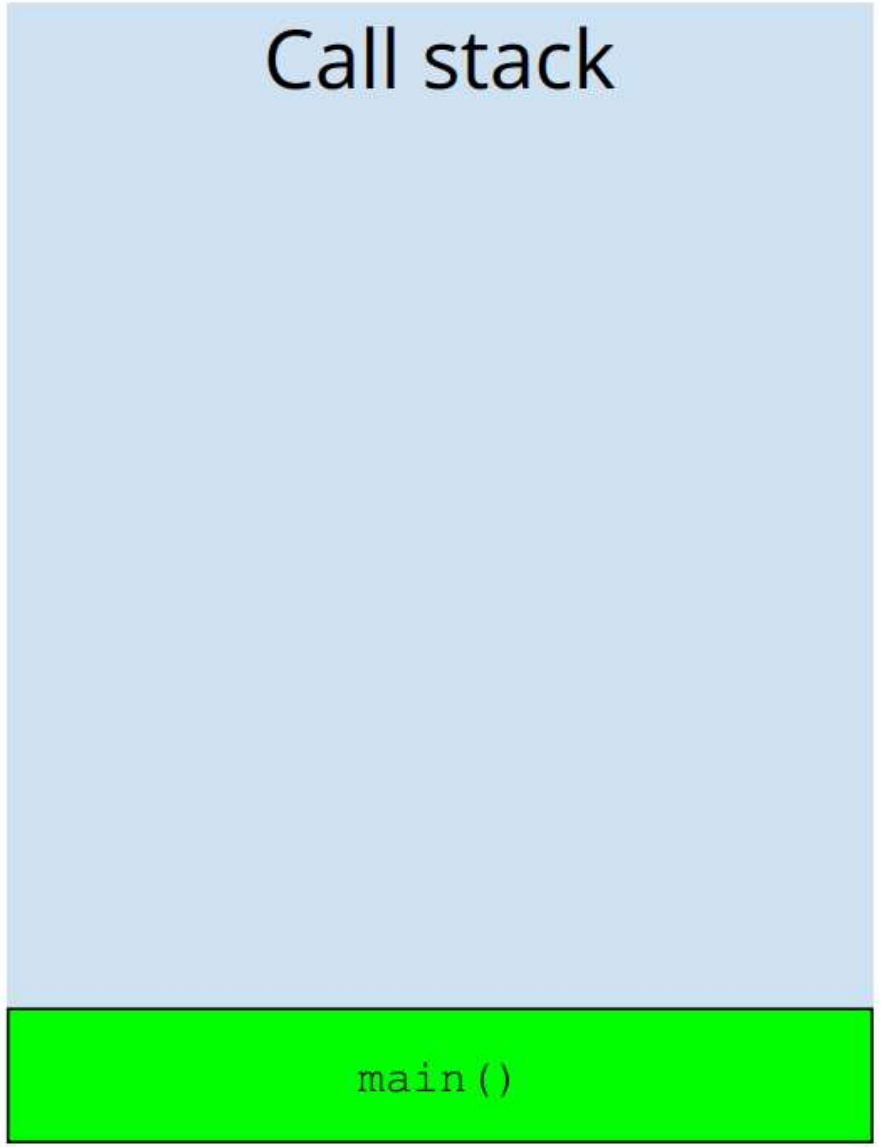
```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```

Call stack

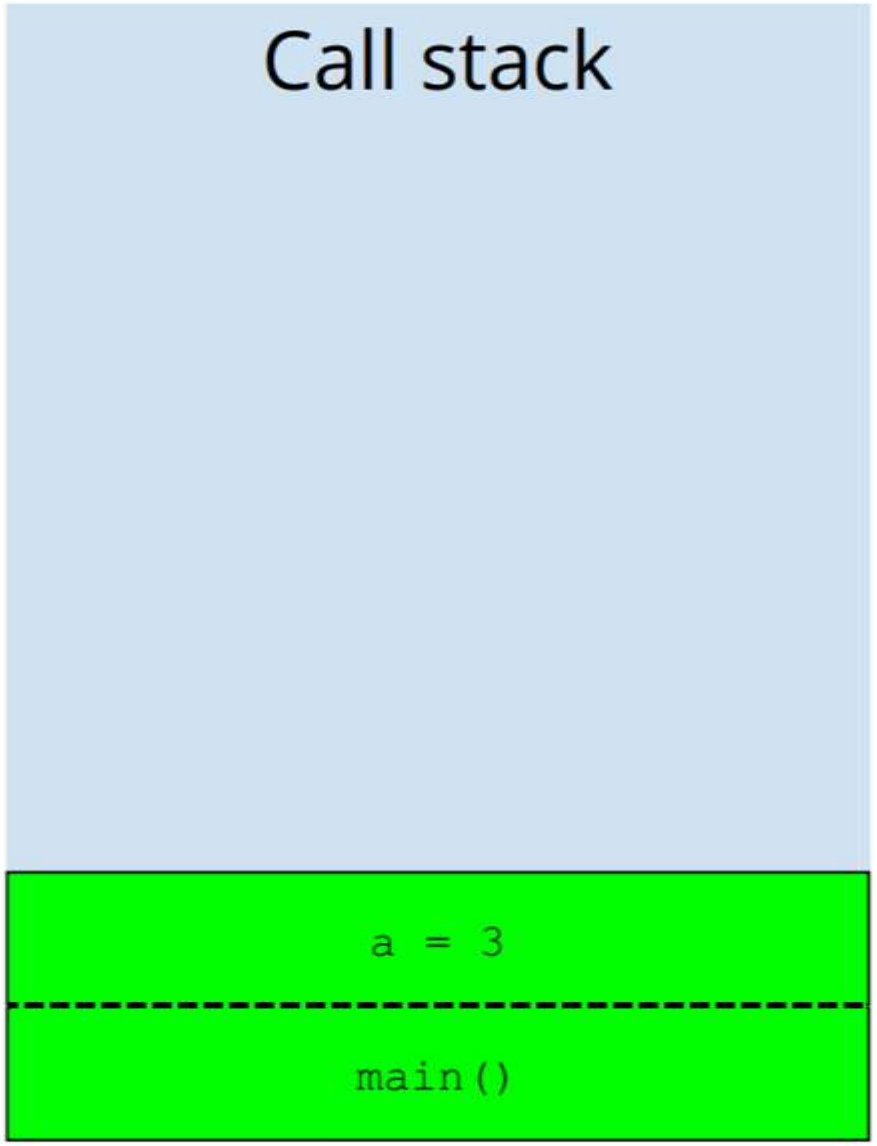
```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



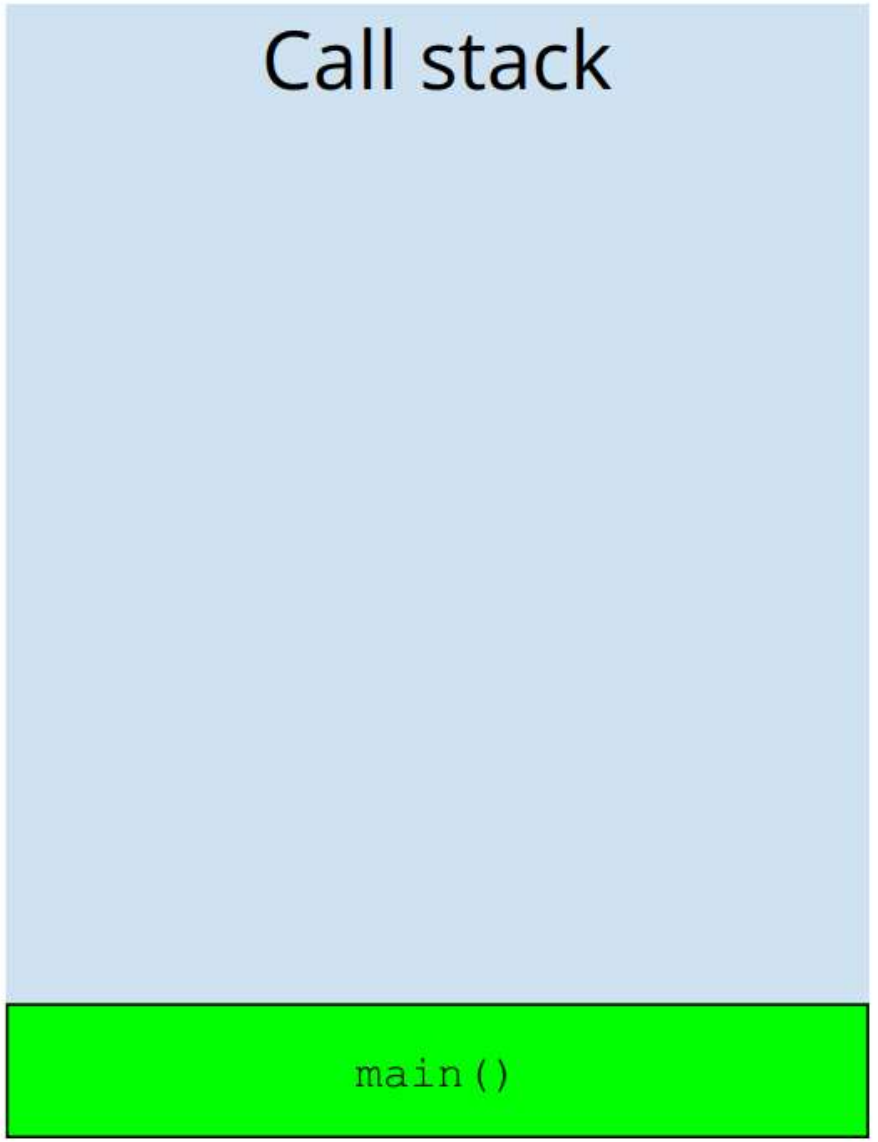
```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



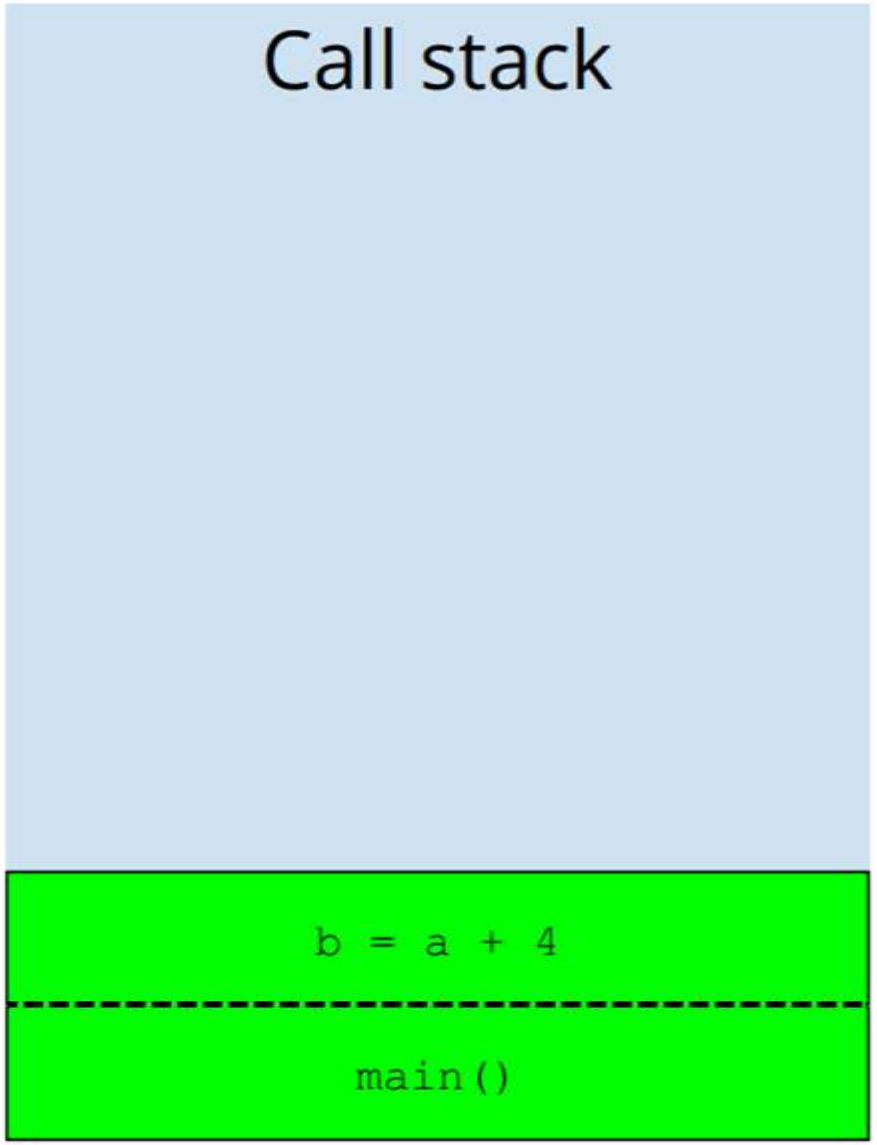
```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



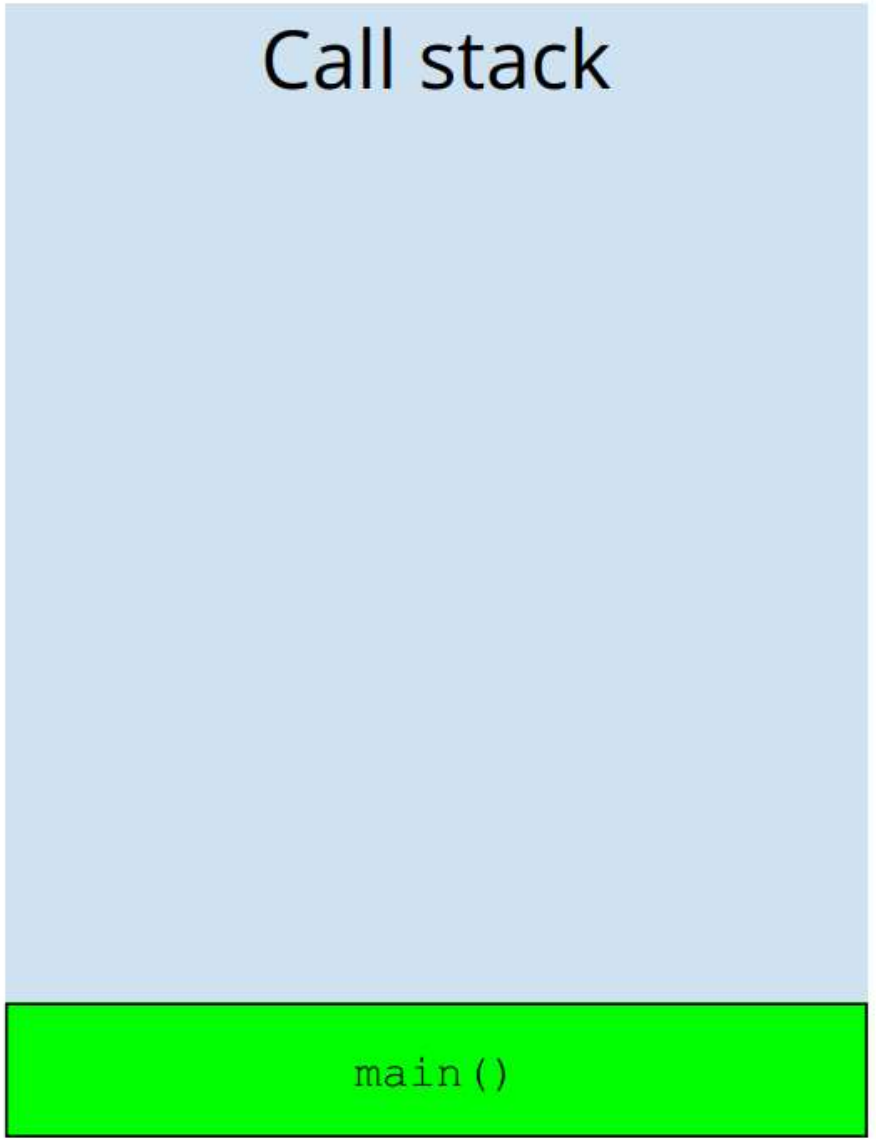
```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



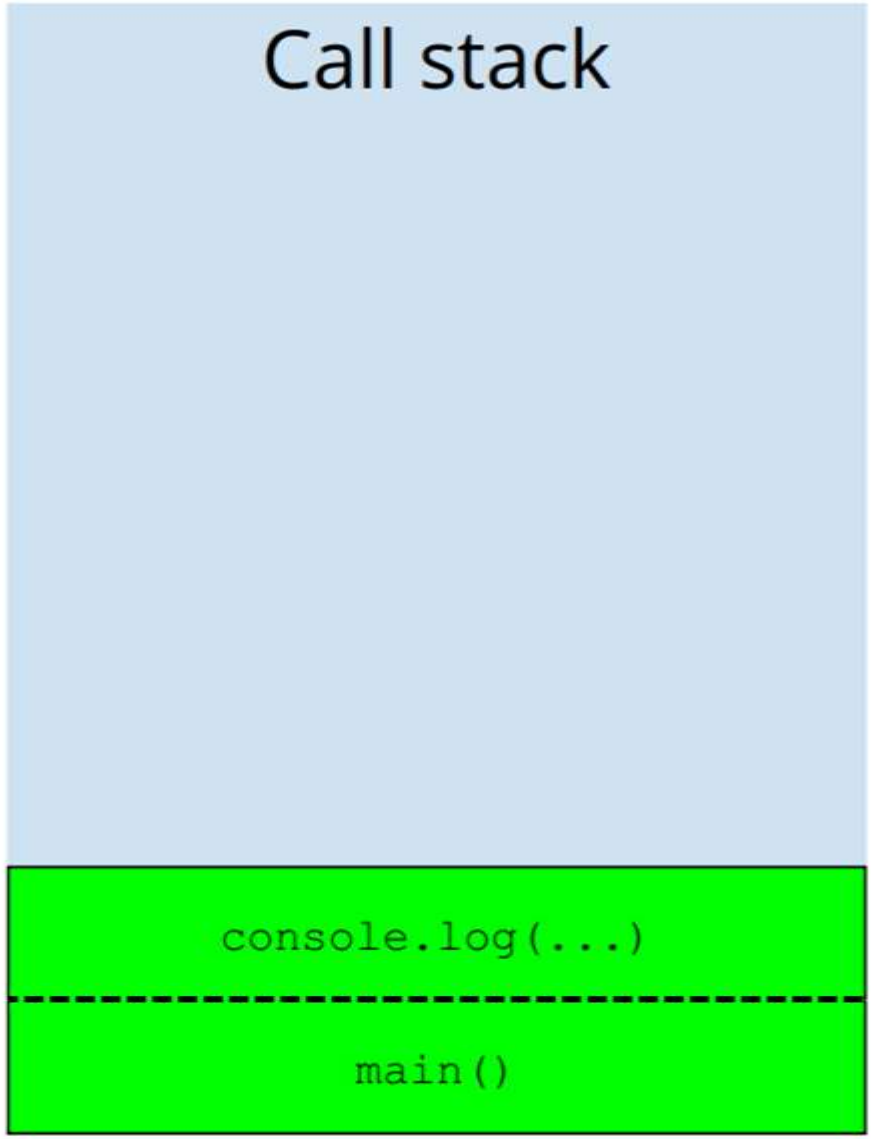
```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



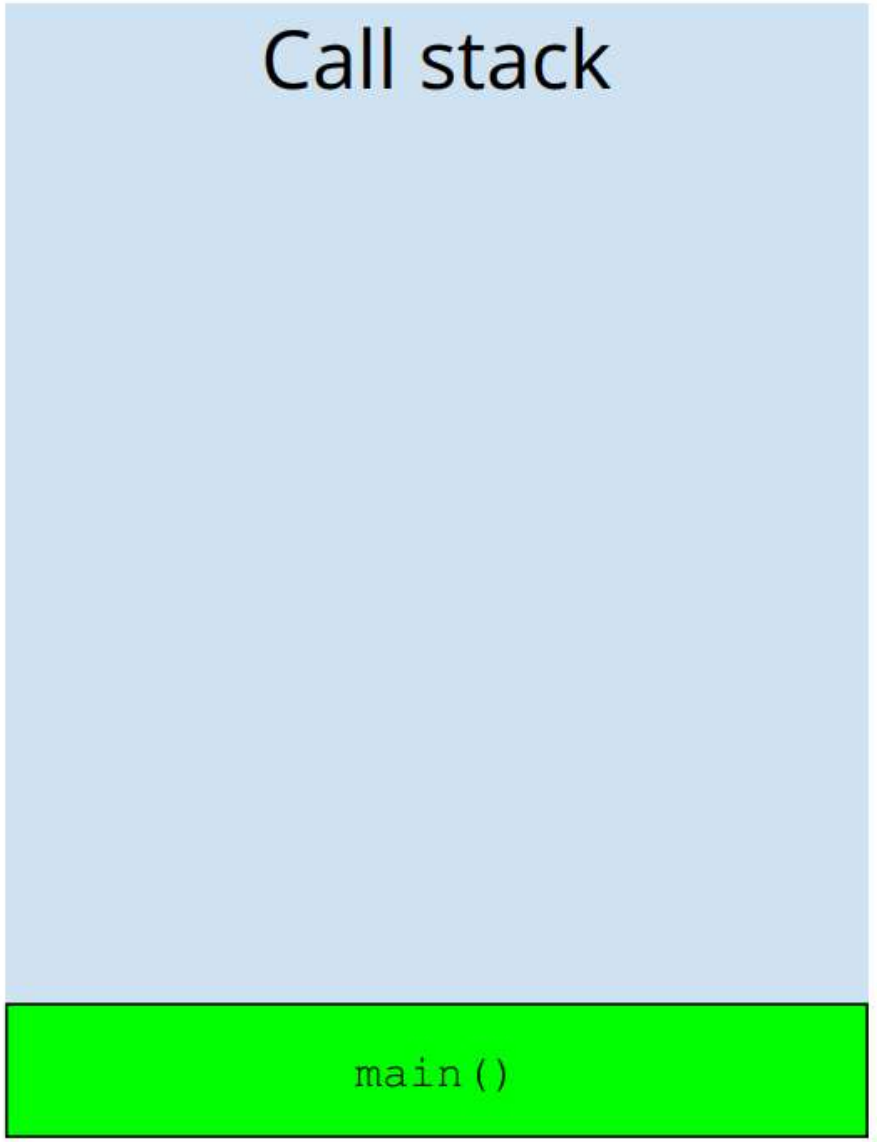
```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```



```
1 const a = 3;
2
3 const b = a + 4;
4
5 console.log('b = ', b);
```

Call stack

Friendly reminder

The Call stack is

not an infinite resource

Stack Overflow

If the stack takes up more space than it had assigned to it, it results in a "stack overflow" error.

Stack Overflow

```
RangeError: Maximum call stack size exceeded
```

Stack trace

```
function abc() {
    throw new Error("help!");
}

function def(){
    abc();
}

function ghi(){
    def();
}

ghi();
```

```
C:\code\Nodejs21-22\Semestr1\03\13\app.js:2
      throw new Error("help!");
      ^

Error: help!
    at abc (C:\code\Nodejs21-22\Semestr1\03\13\app.js:2:13)
    at def (C:\code\Nodejs21-22\Semestr1\03\13\app.js:5:7)
    at ghi (C:\code\Nodejs21-22\Semestr1\03\13\app.js:8:7)
    at Object.<anonymous> (C:\code\Nodejs21-22\Semestr1\03\13\app.js:10:3)
    at Module._compile (node:internal/modules/cjs/loader:1101:14)
    at Object.Module._extensions..js (node:internal/modules/cjs/loader:1153:10)
    at Module.load (node:internal/modules/cjs/loader:981:32)
    at Function.Module._load (node:internal/modules/cjs/loader:822:12)
    at Function.executeUserEntryPoint [as runMain] (node:internal/modules/run_main:81:12)
    at node:internal/main/run_main_module:17:47
```

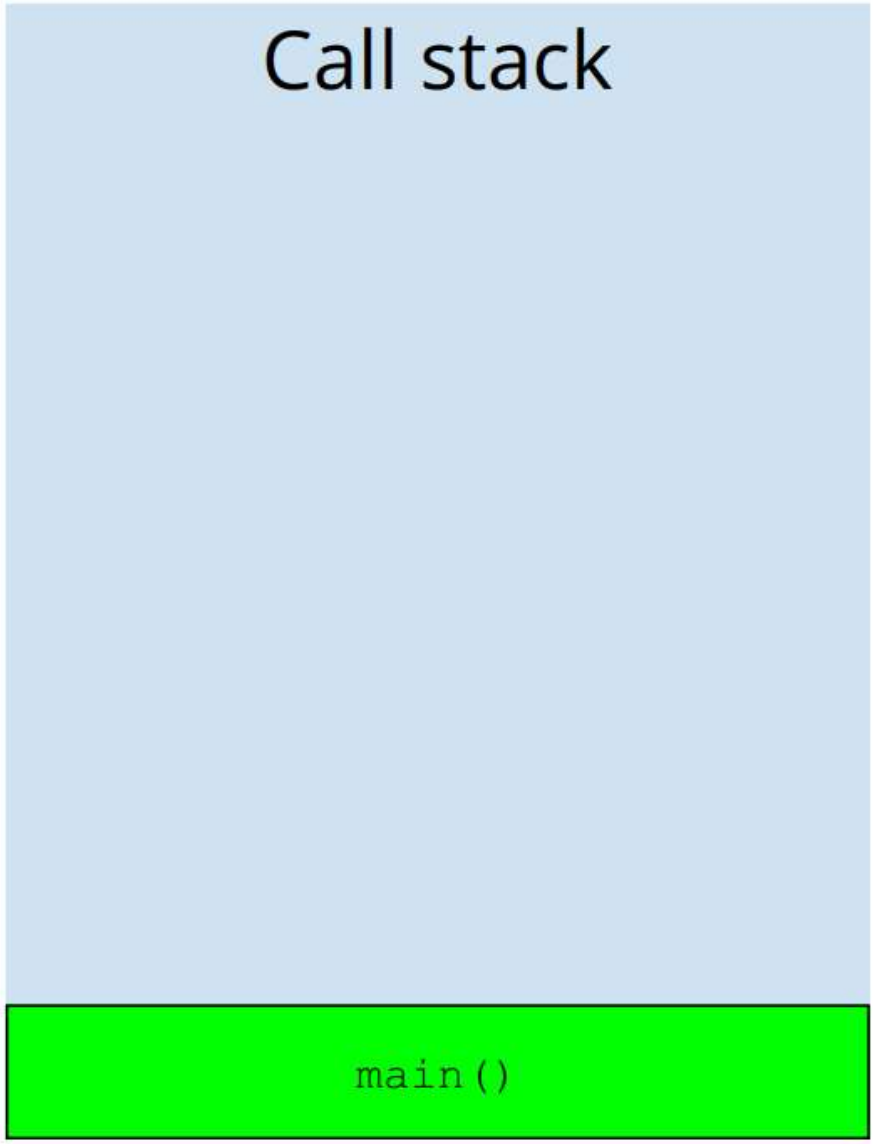
Stack – advanced example

```
function multiply(a, b) {
    const result = a * b;
    return result;
}

const someNumber = multiply(5, 4);
console.log(someNumber);
```

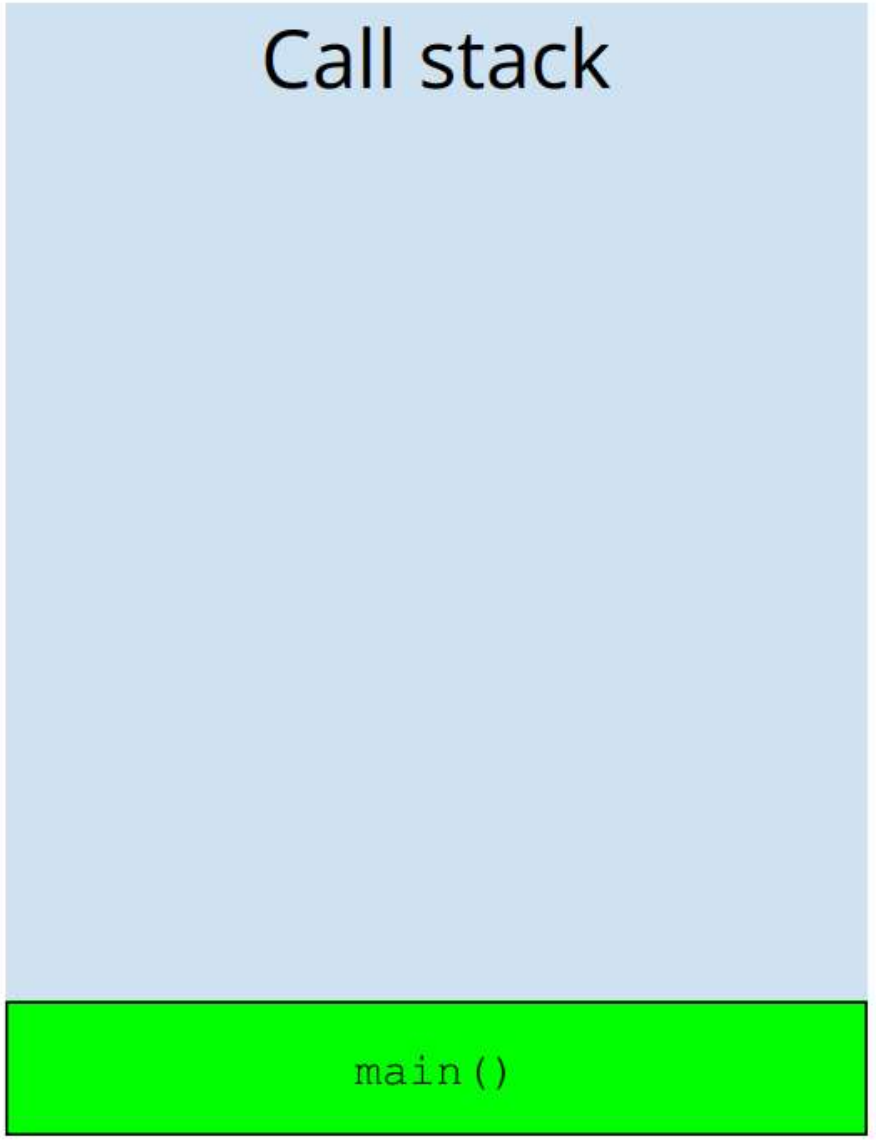
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```

Call stack

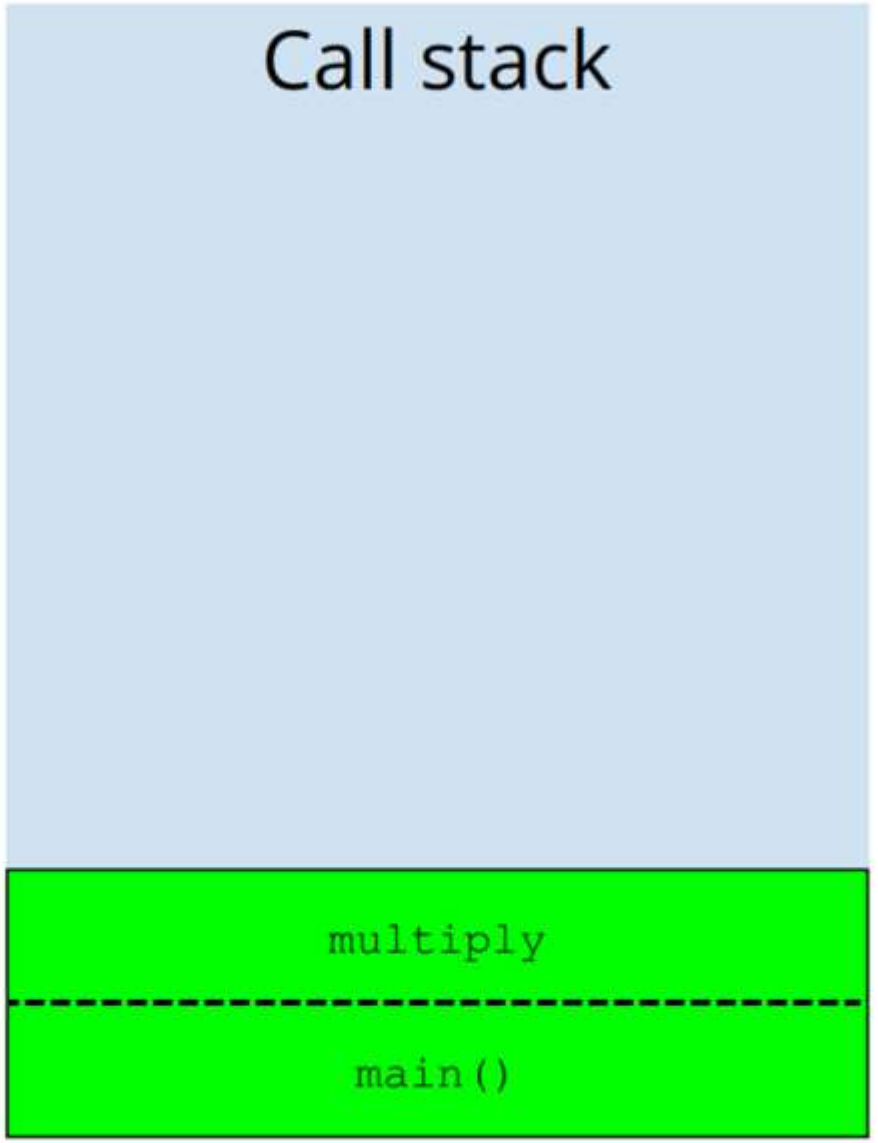
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



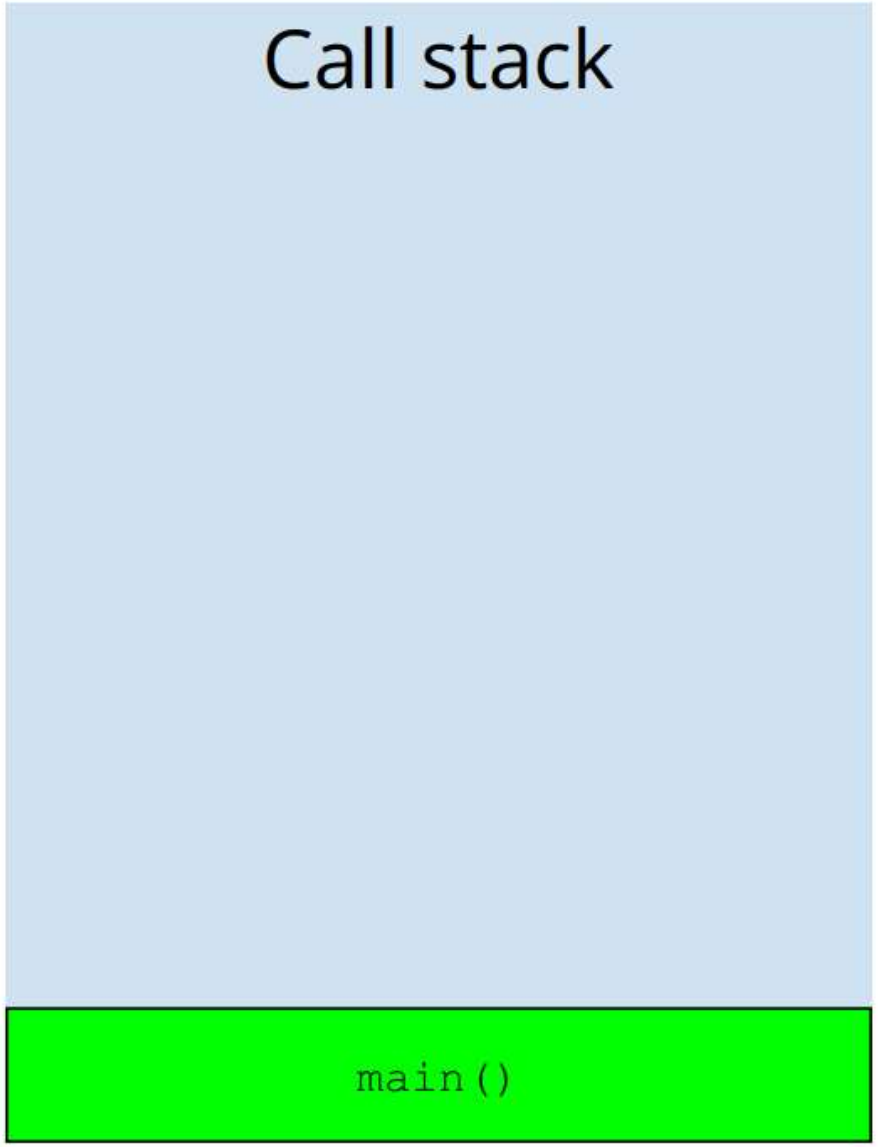
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



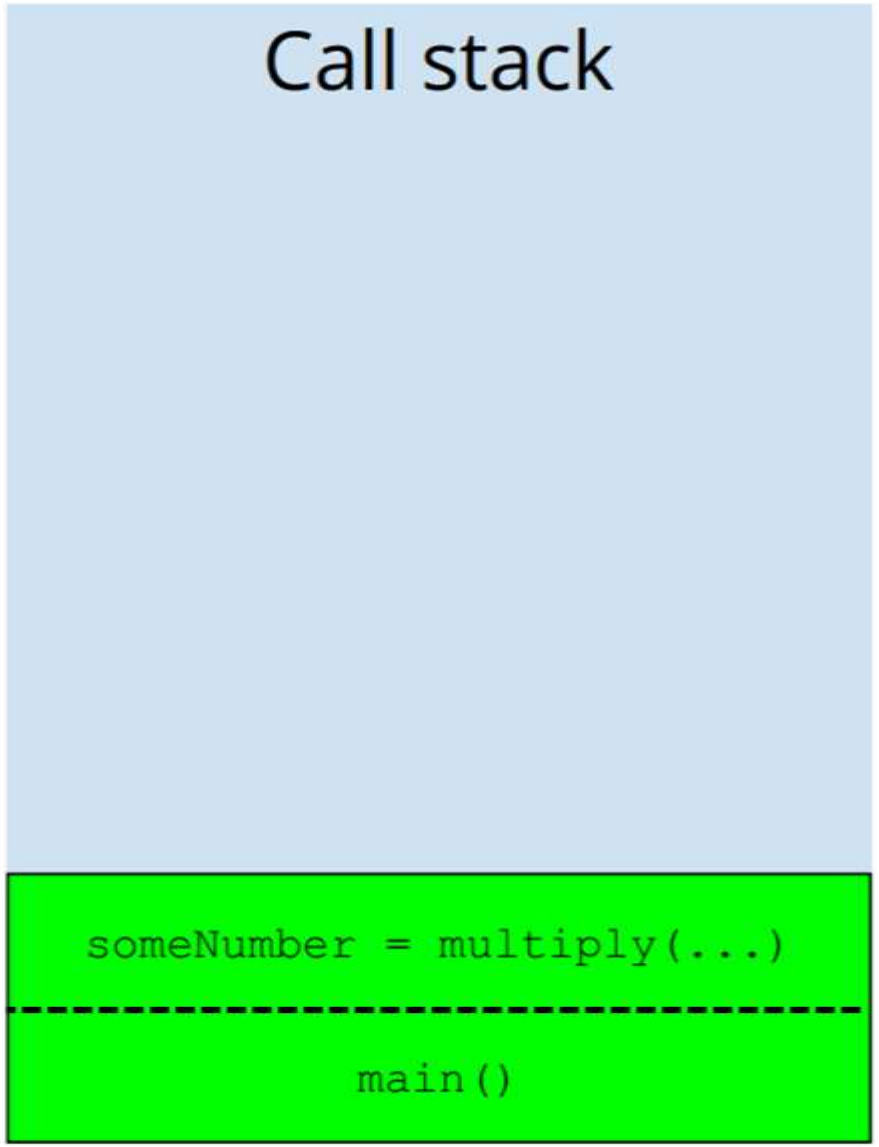
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



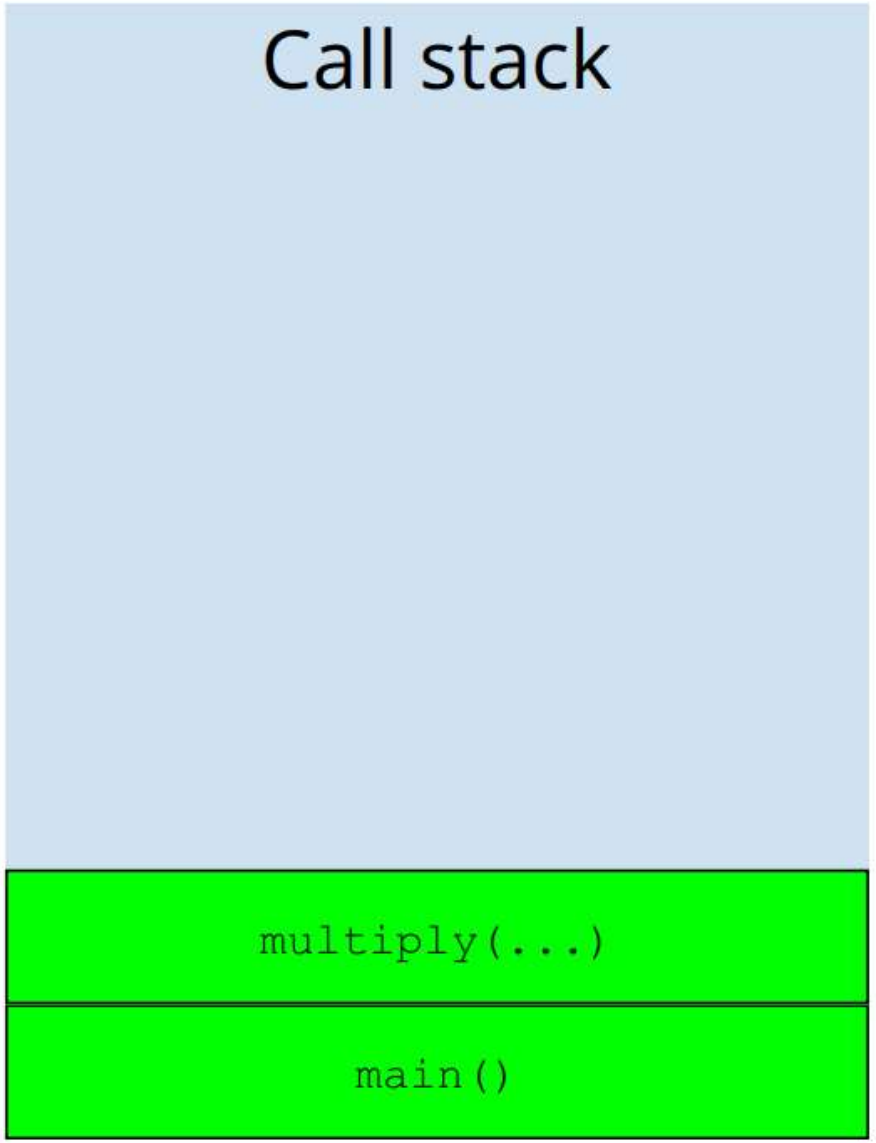
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



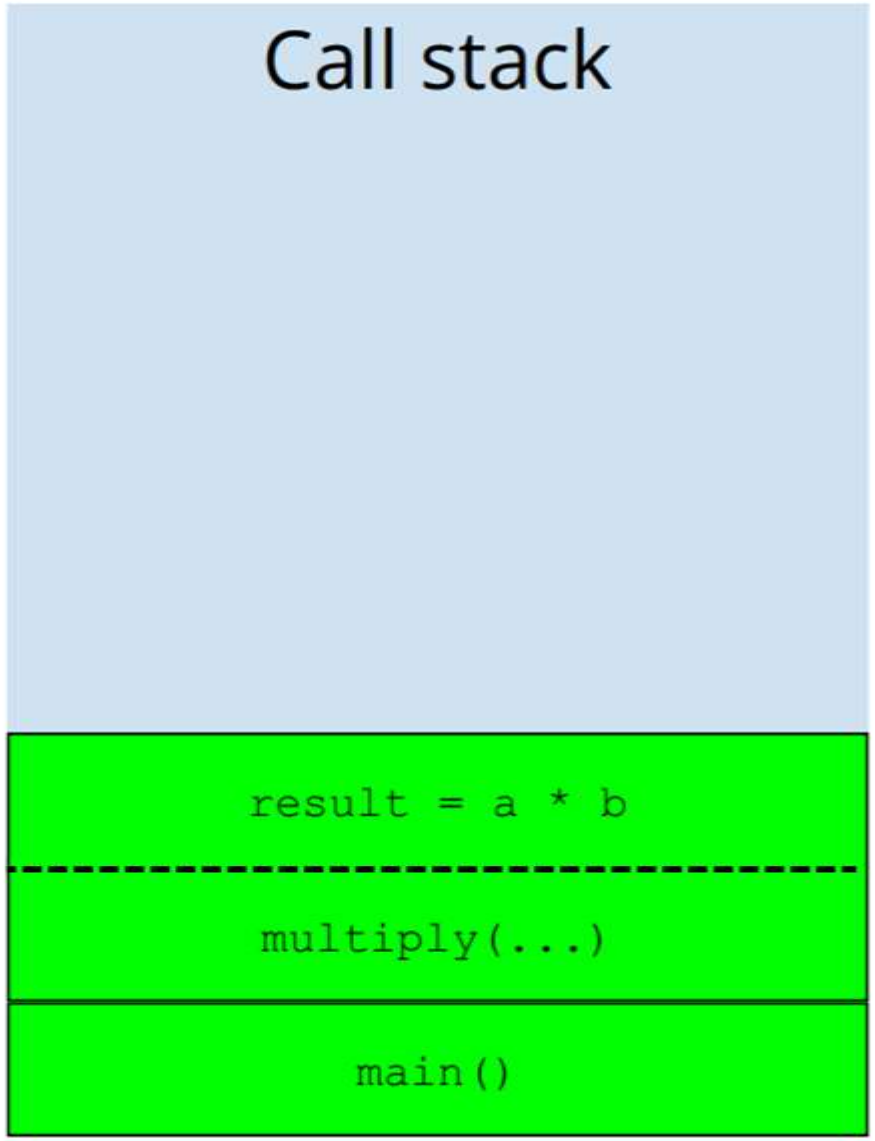
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



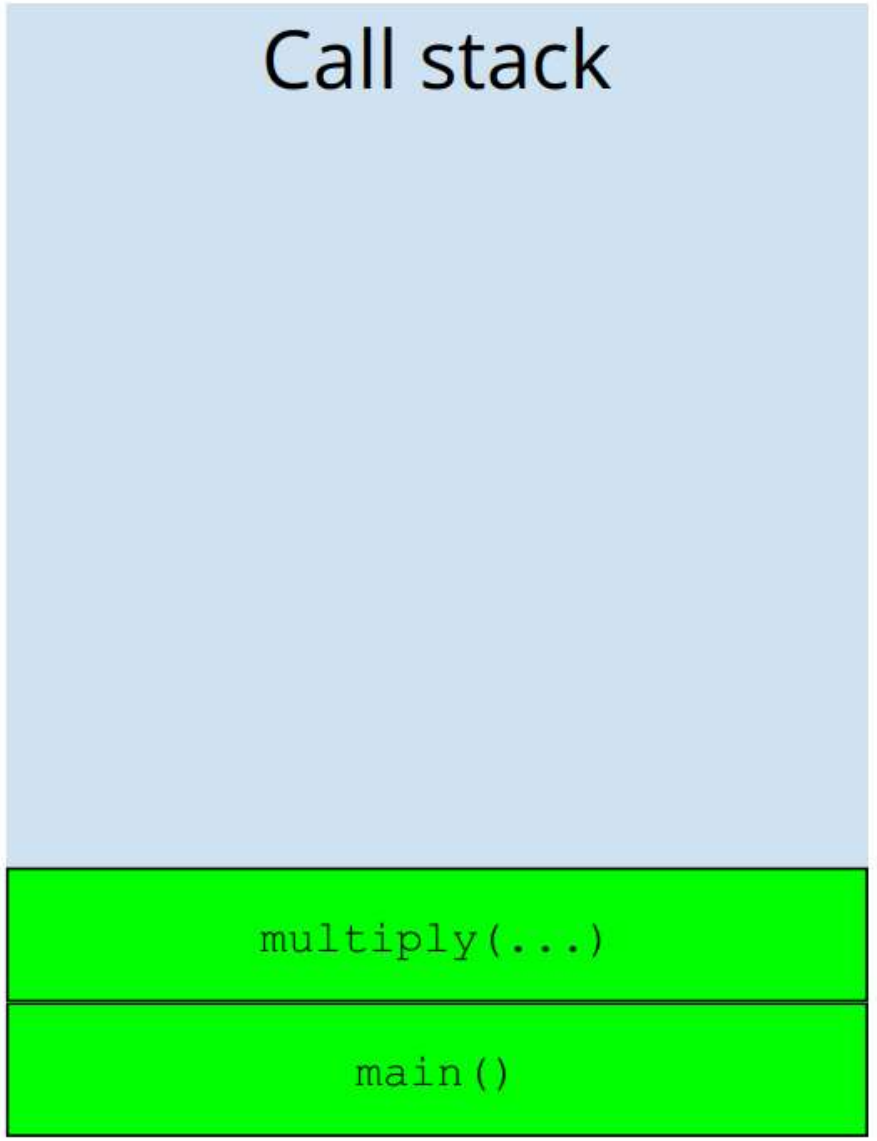
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



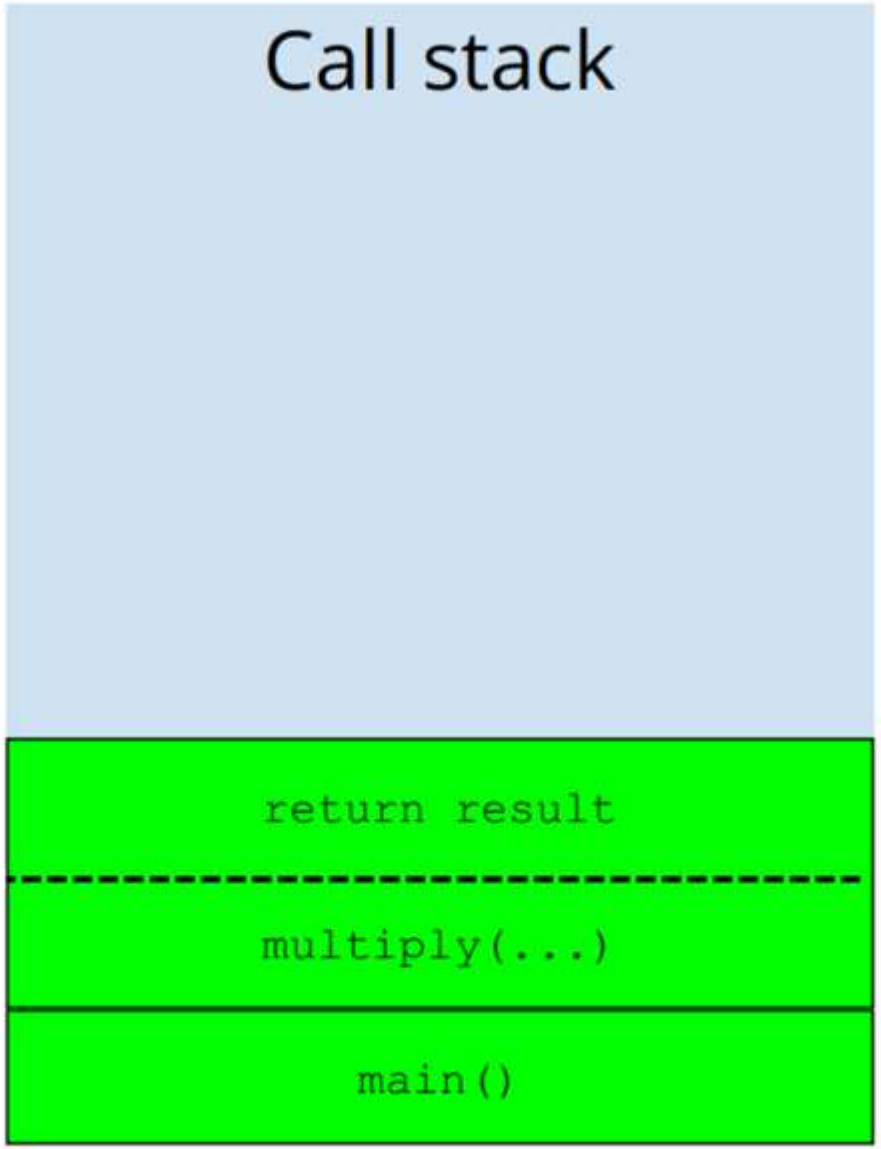
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



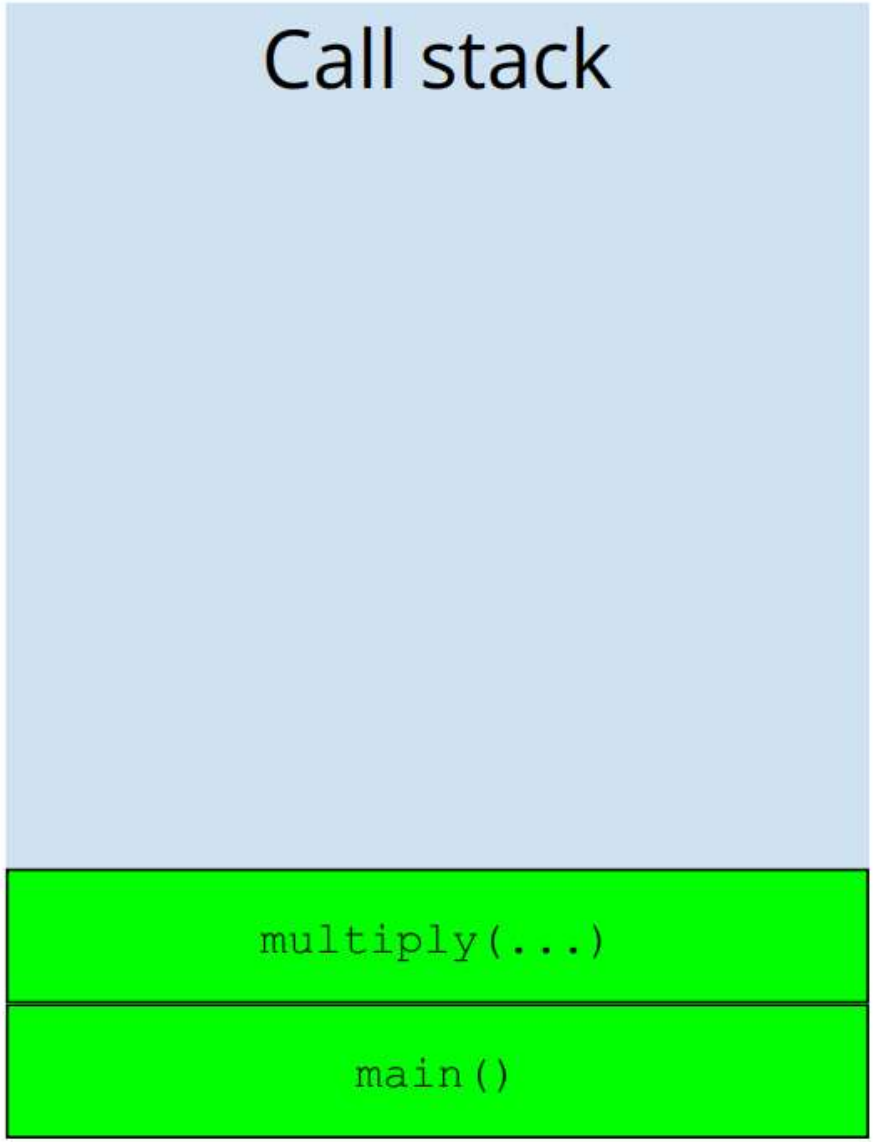
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



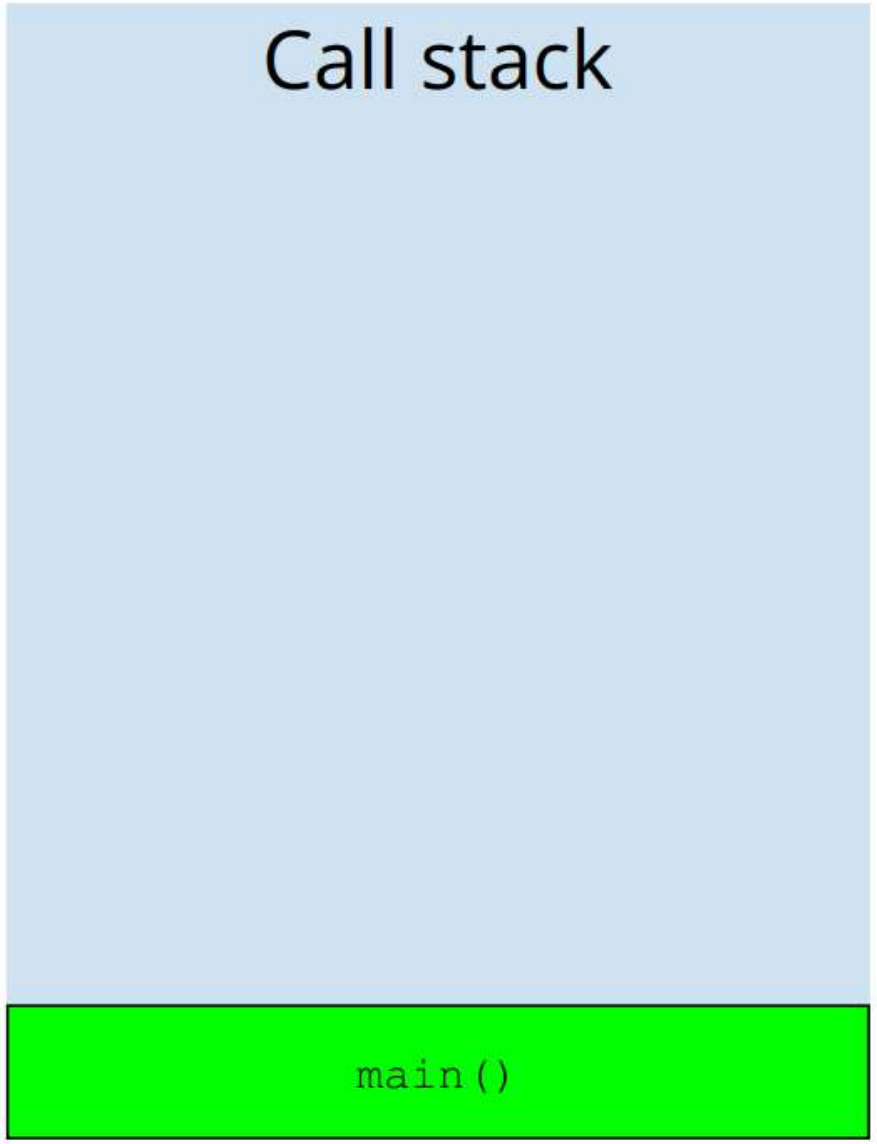
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



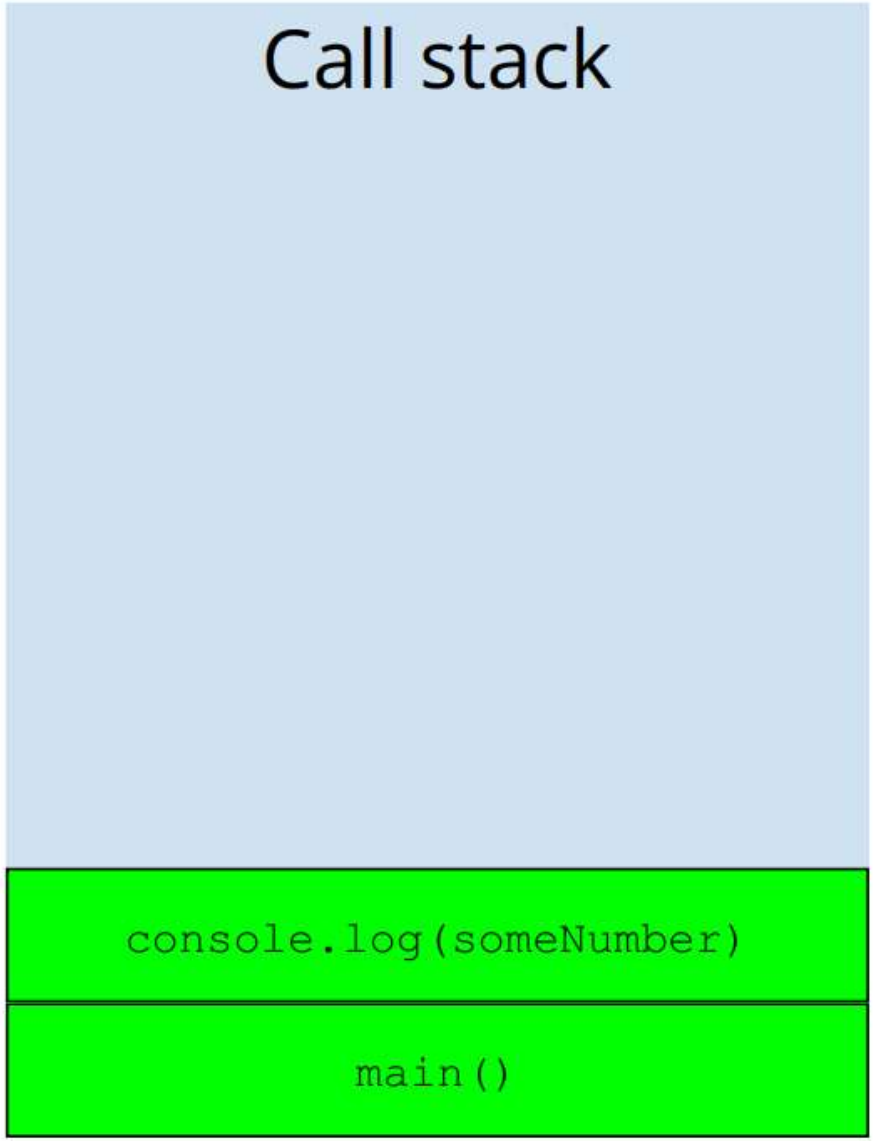
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



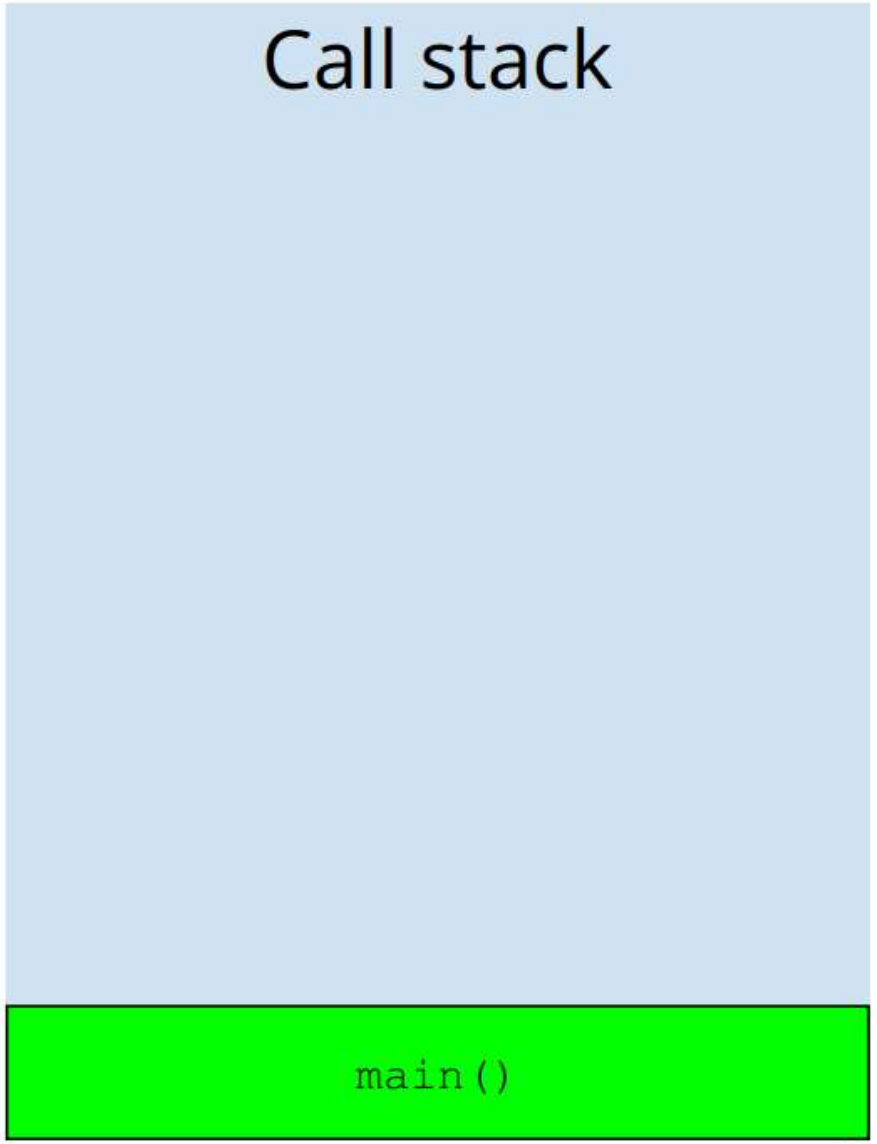
```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```



```
1 function multiply(a, b) {
2   const result = a * b;
3   return result;
4 }
5
6 const someNumber = multiply(5, 4);
7
8 console.log(someNumber);
```

Call stack

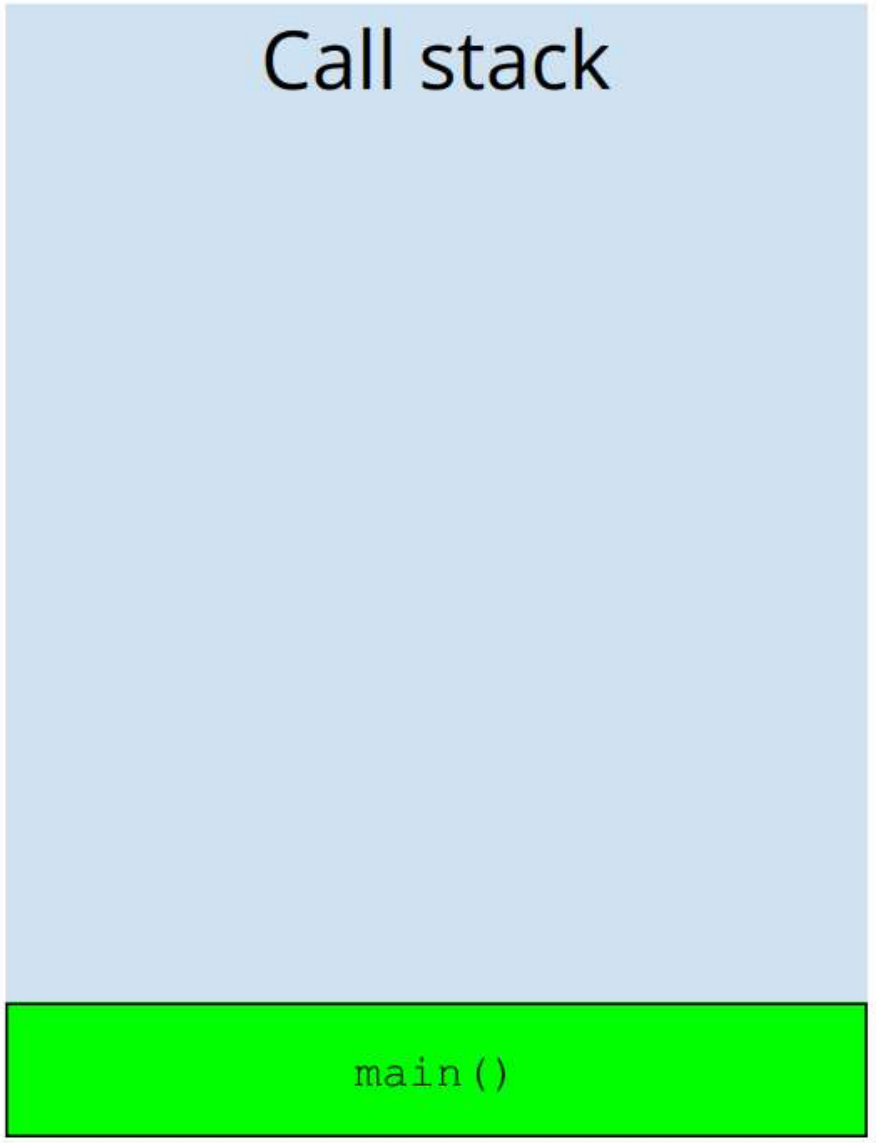
Stack – more advanced example

```
const fs = require('fs’);
const file1 = fs.readFileSync('file1.txt’);
const file2 = fs.readFileSync('file2.txt’);
const file3 = fs.readFileSync('file3.txt’);
console.log(file1);
console.log(file2);
console.log(file3);
```

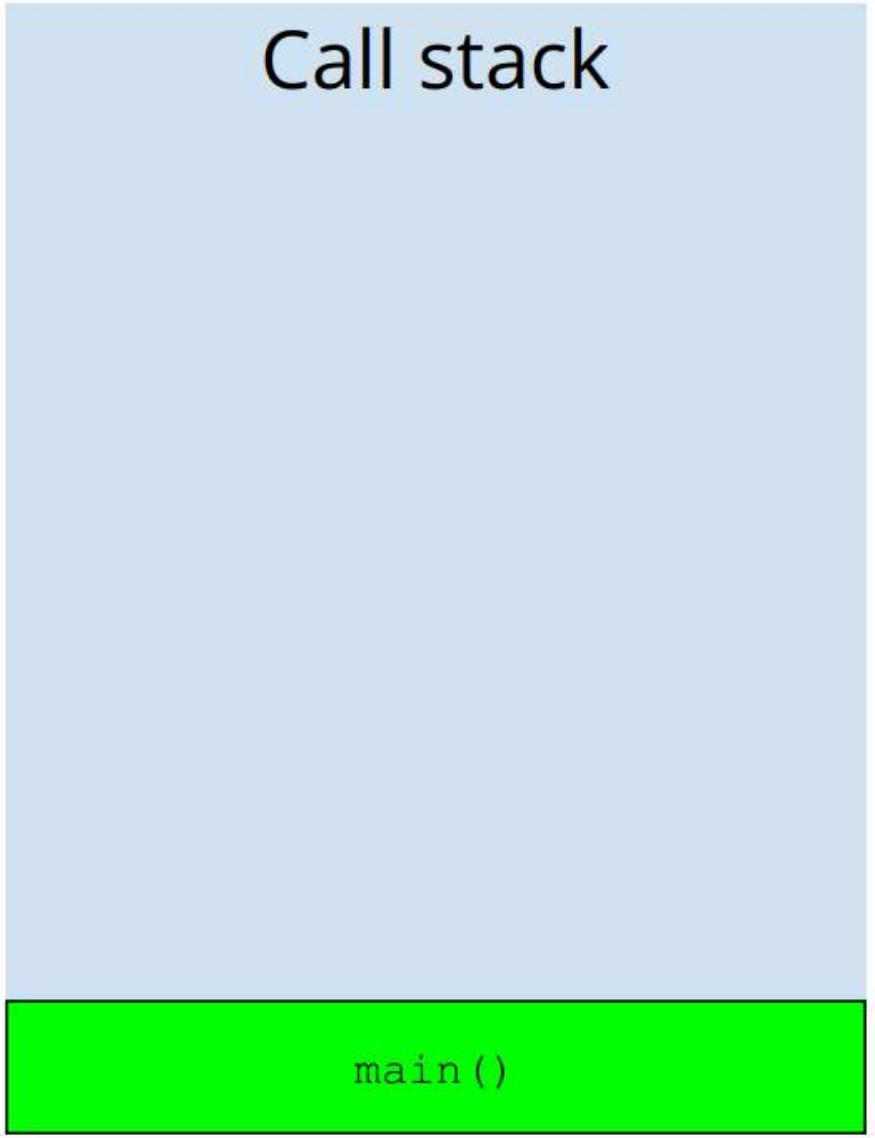
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```

Call stack

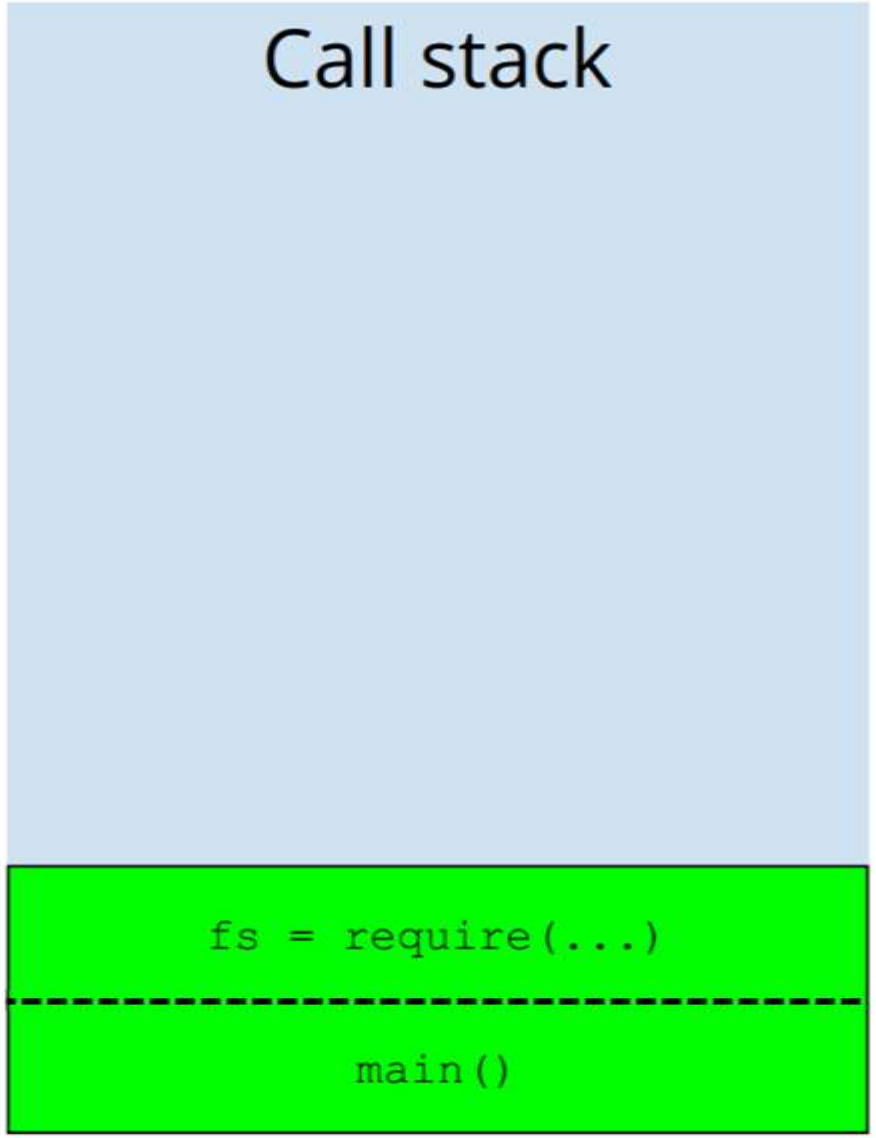
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



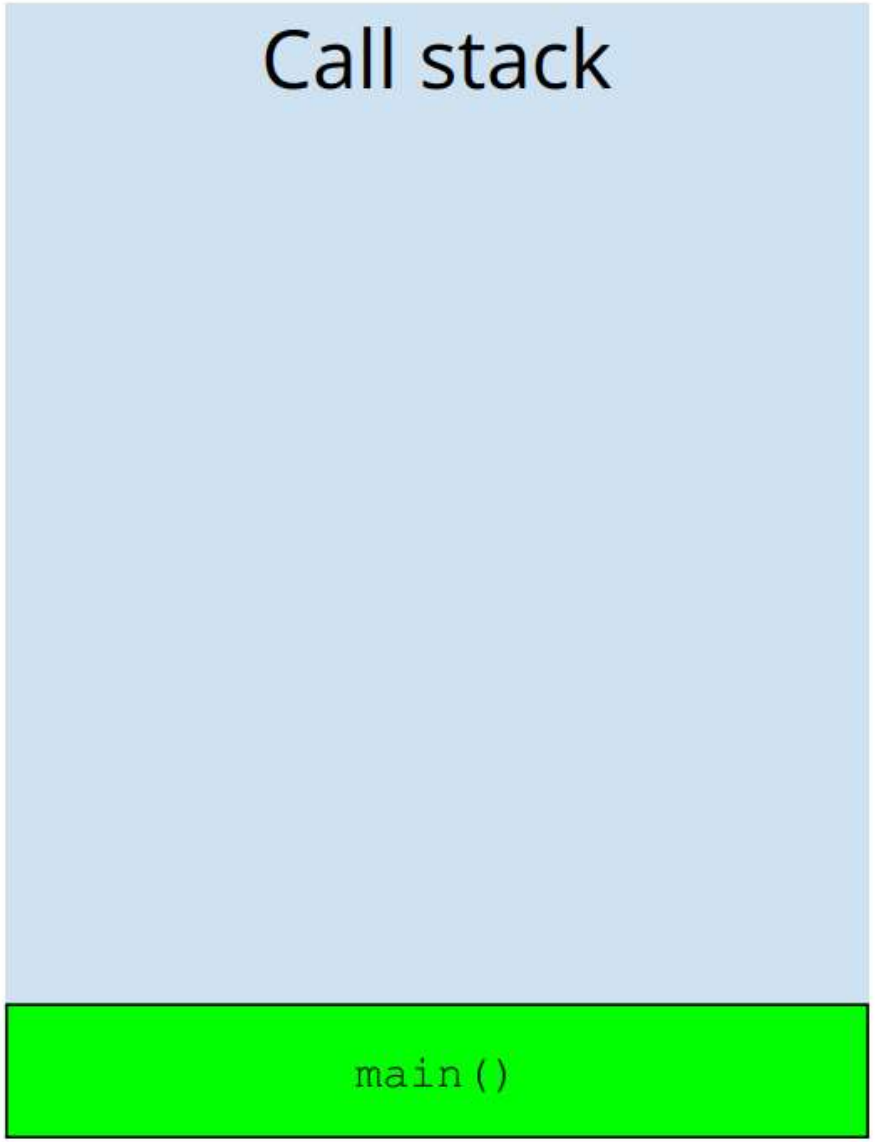
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



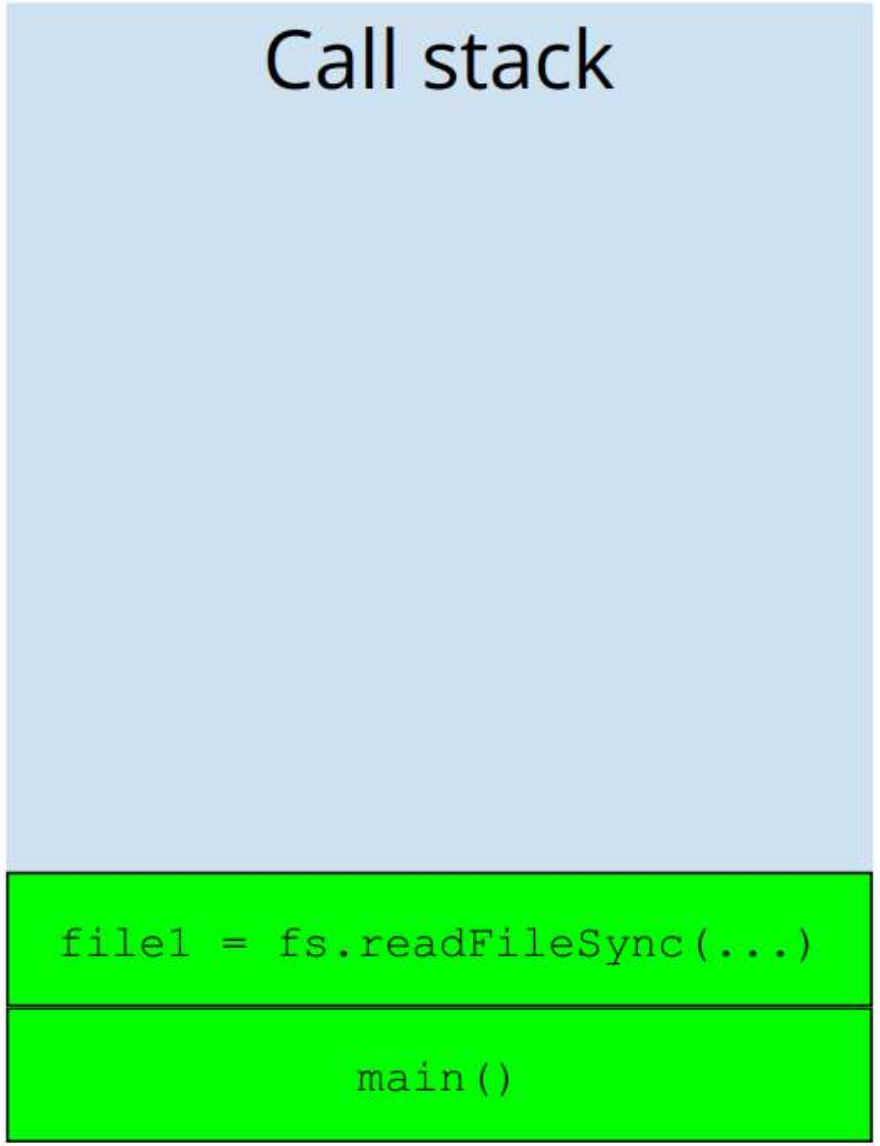
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



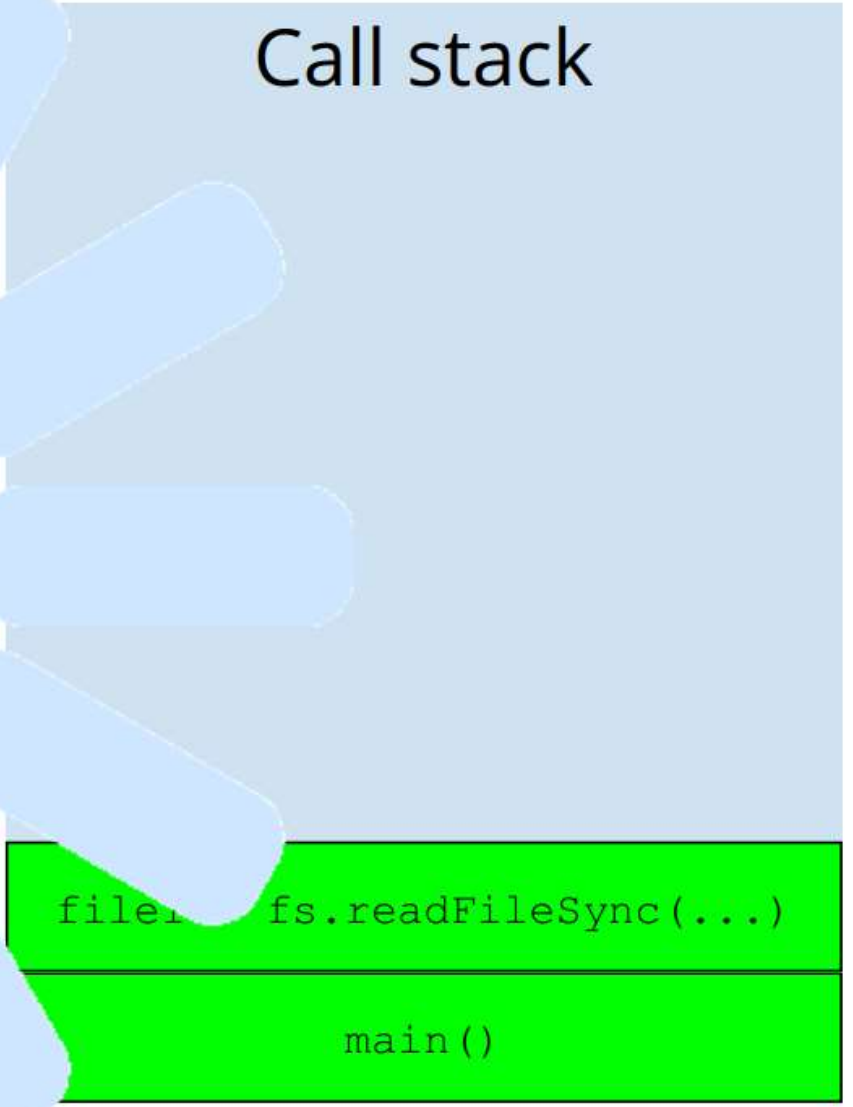
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



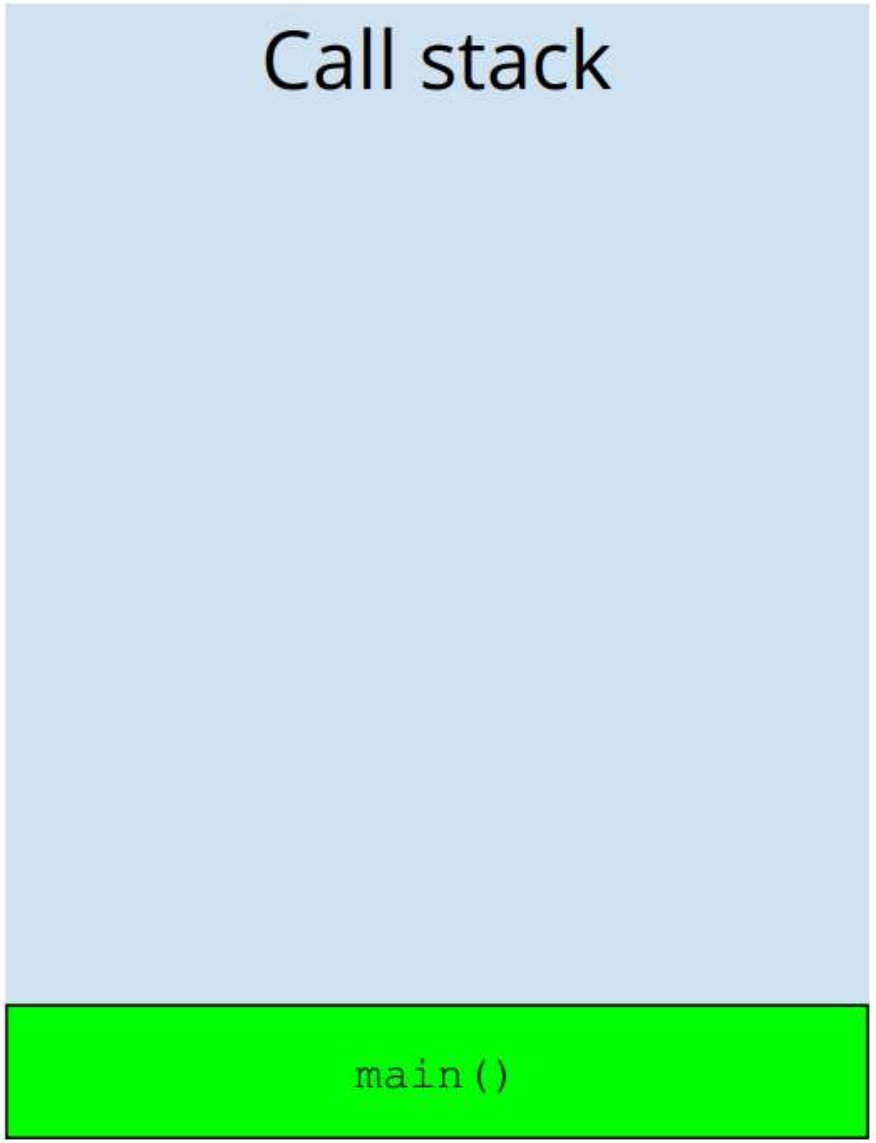
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



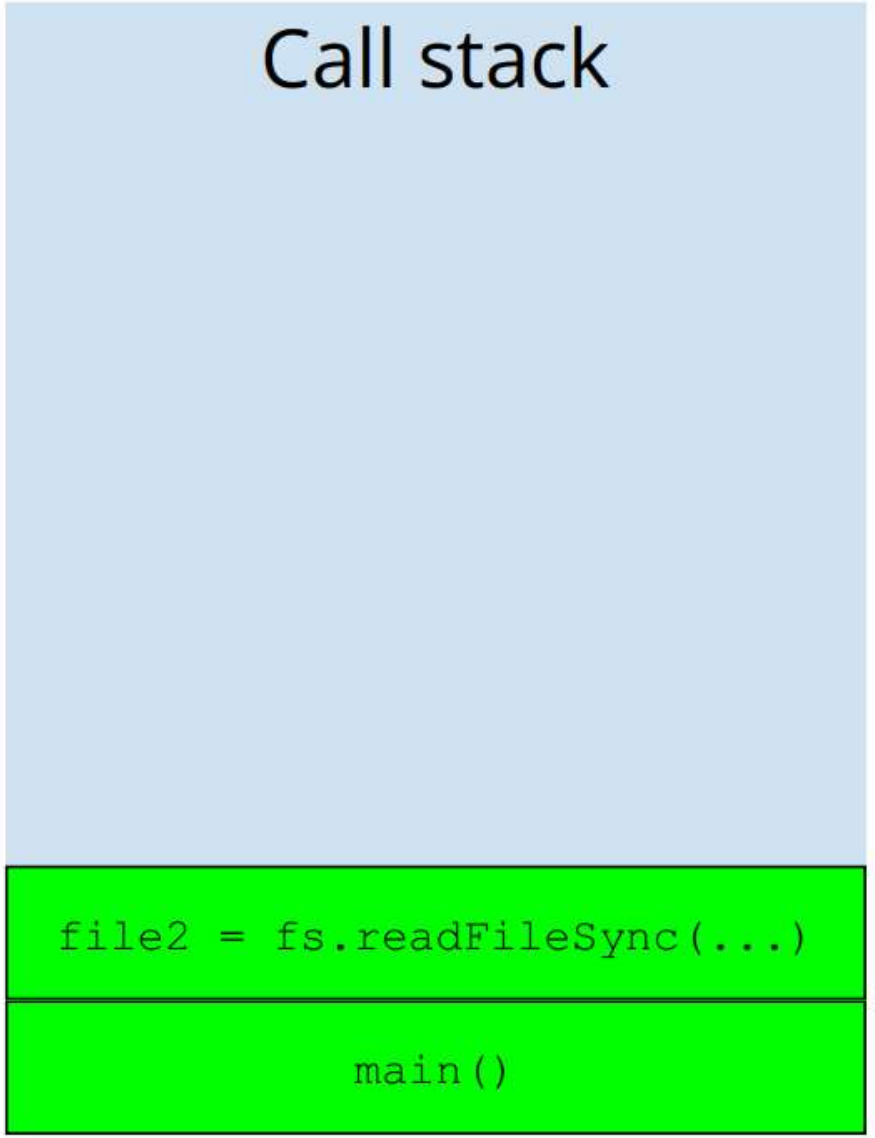
```
1  const fs = require('fs');
2
3
4  const file1 = fs.r                c('  e1.txt');
5
6  const file2 = fs.readFil          'file2.txt');
7
8  const file3 = f                   c('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3
```



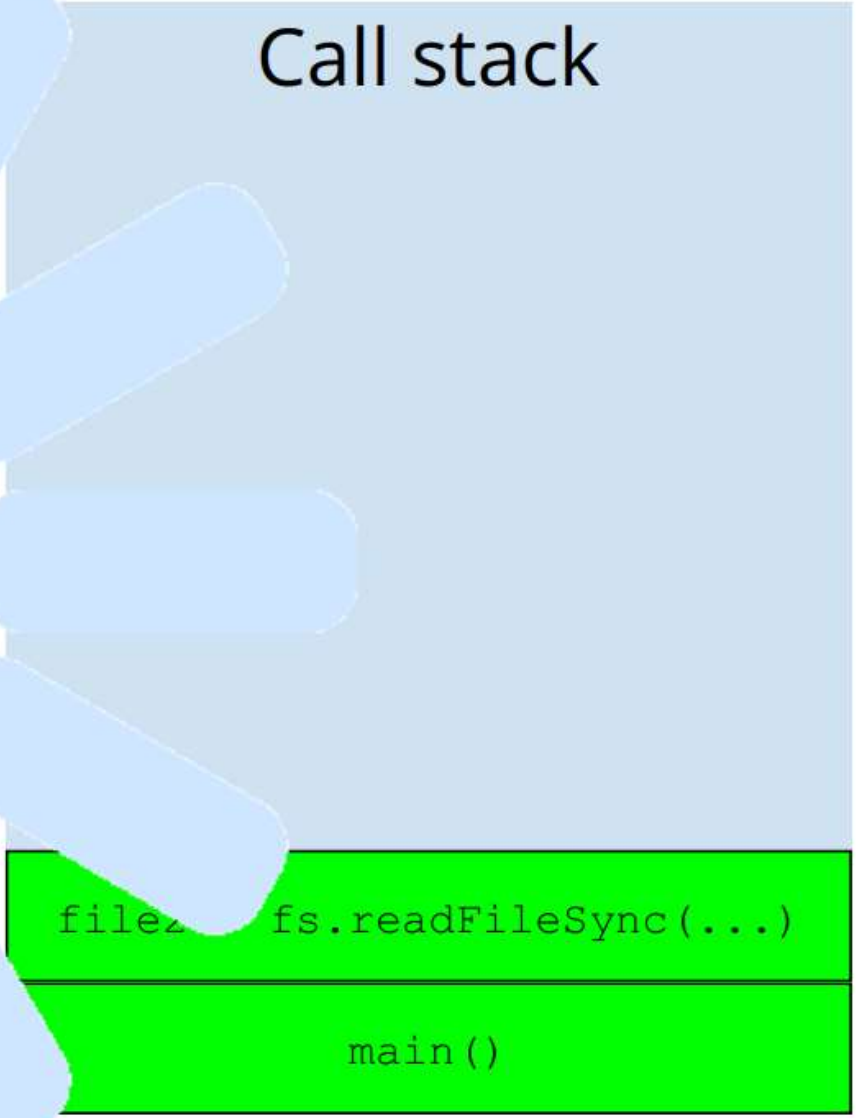
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



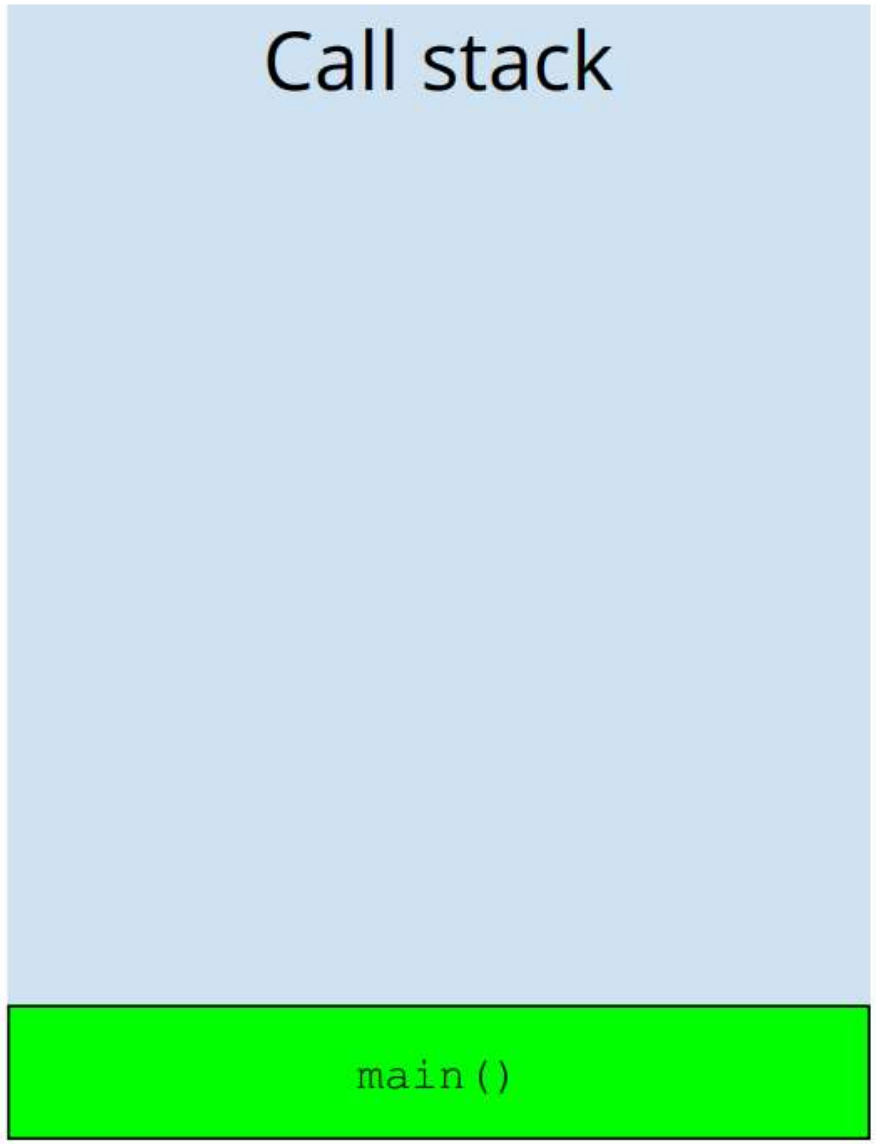
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



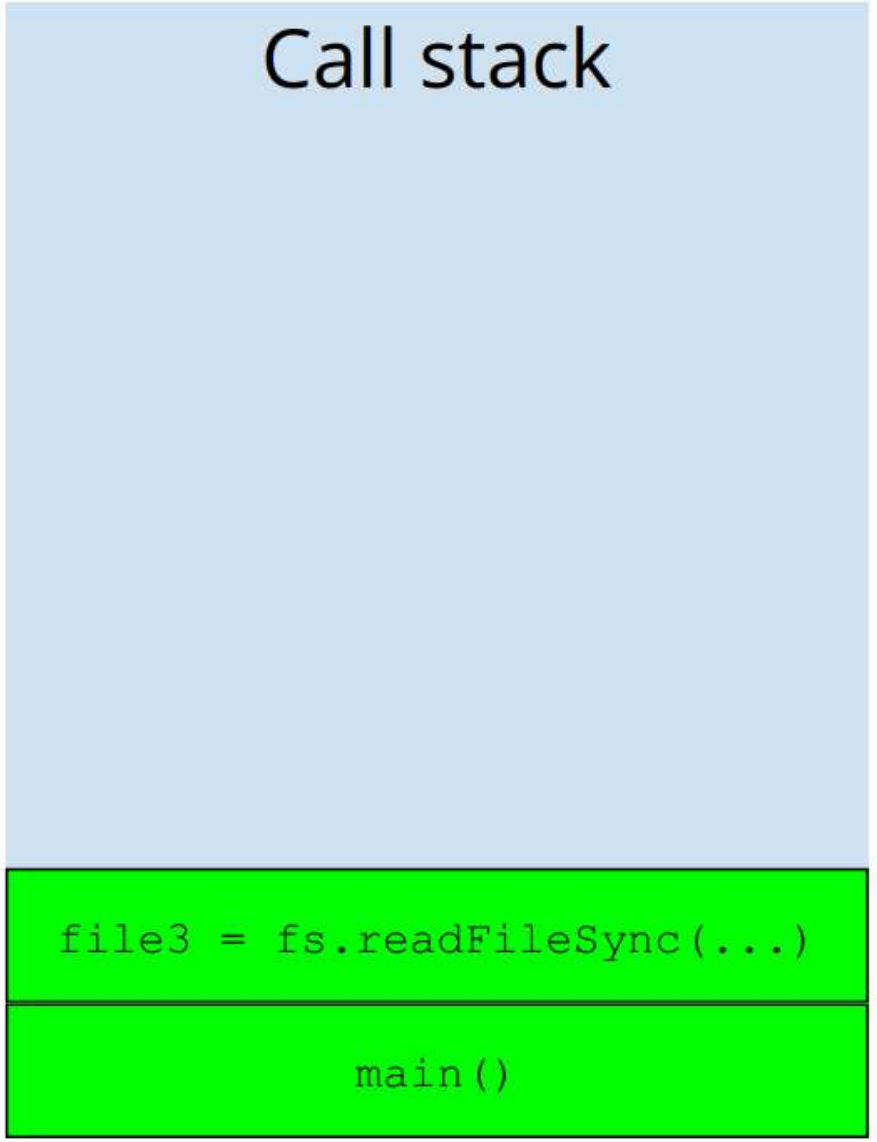
```
1  const fs = require('fs');
2
3
4  const file1 = fs.r                c('  e1.txt');
5
6  const file2 = fs.readFil          'file2.txt');
7
8  const file3 = f                   c('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3
```



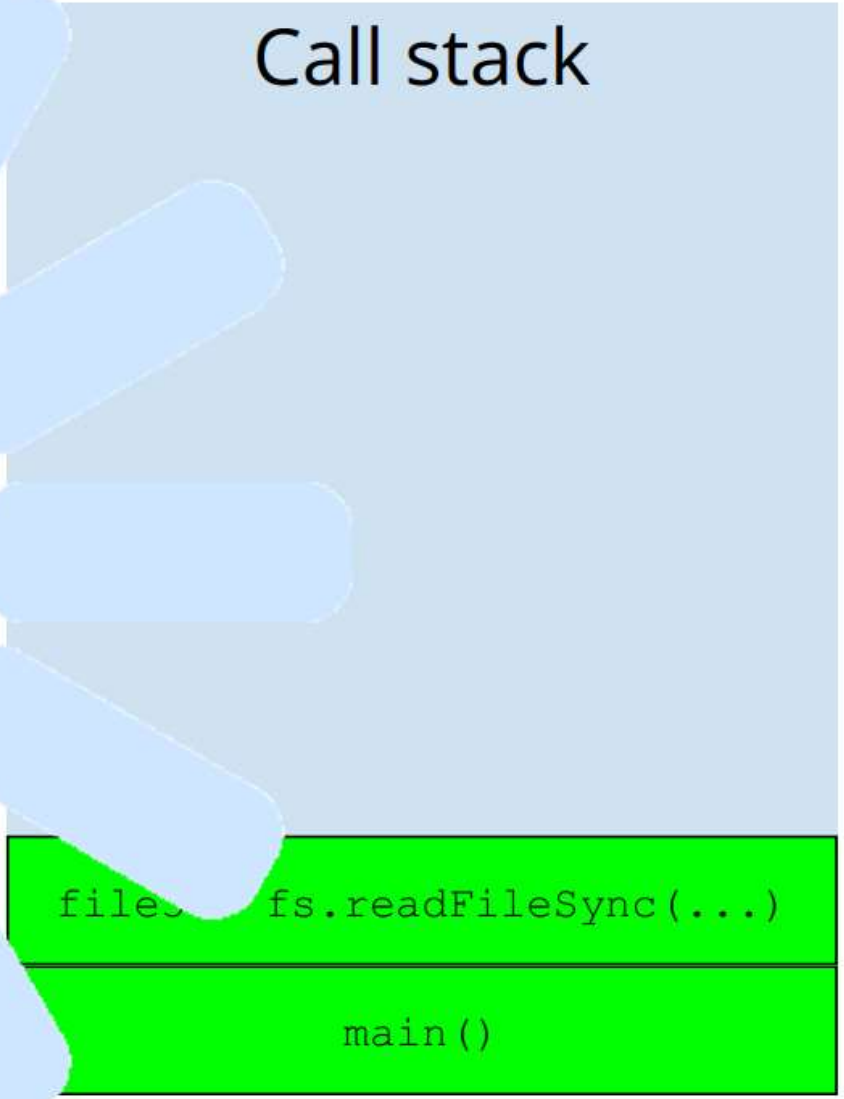
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



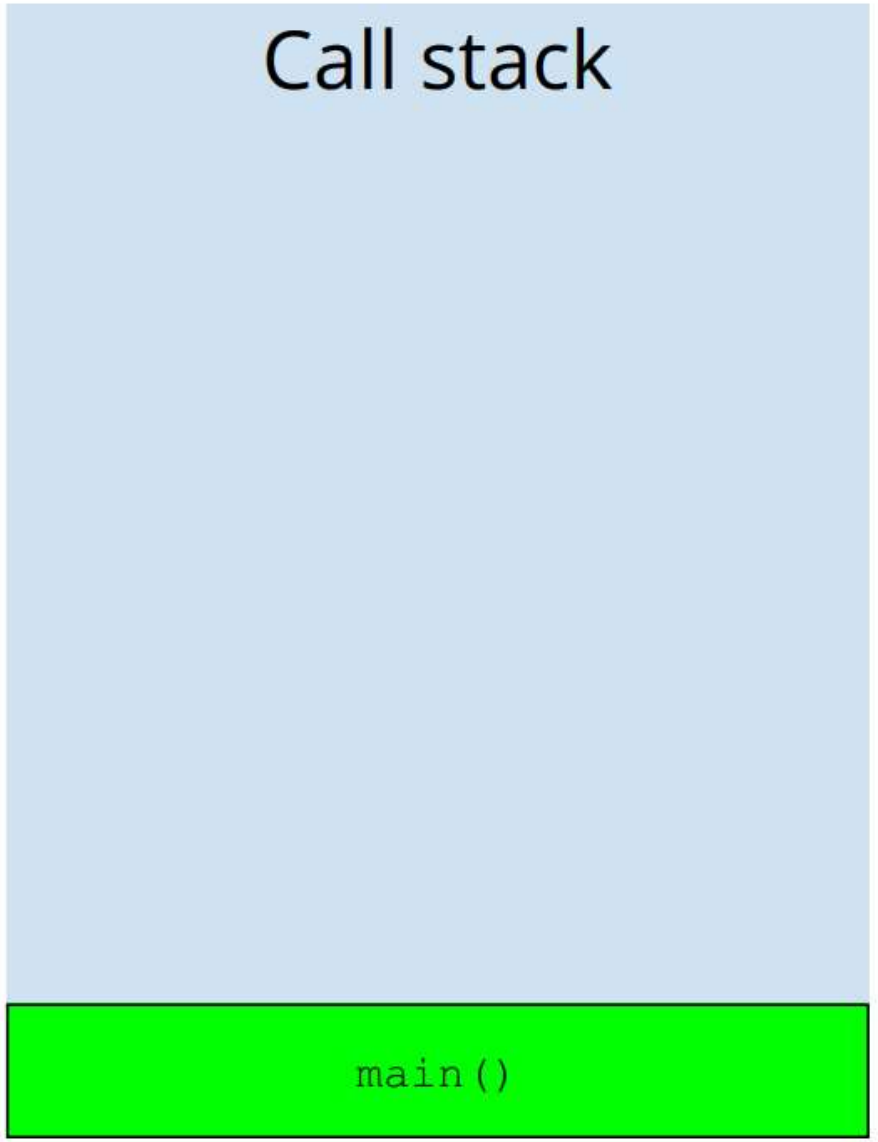
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



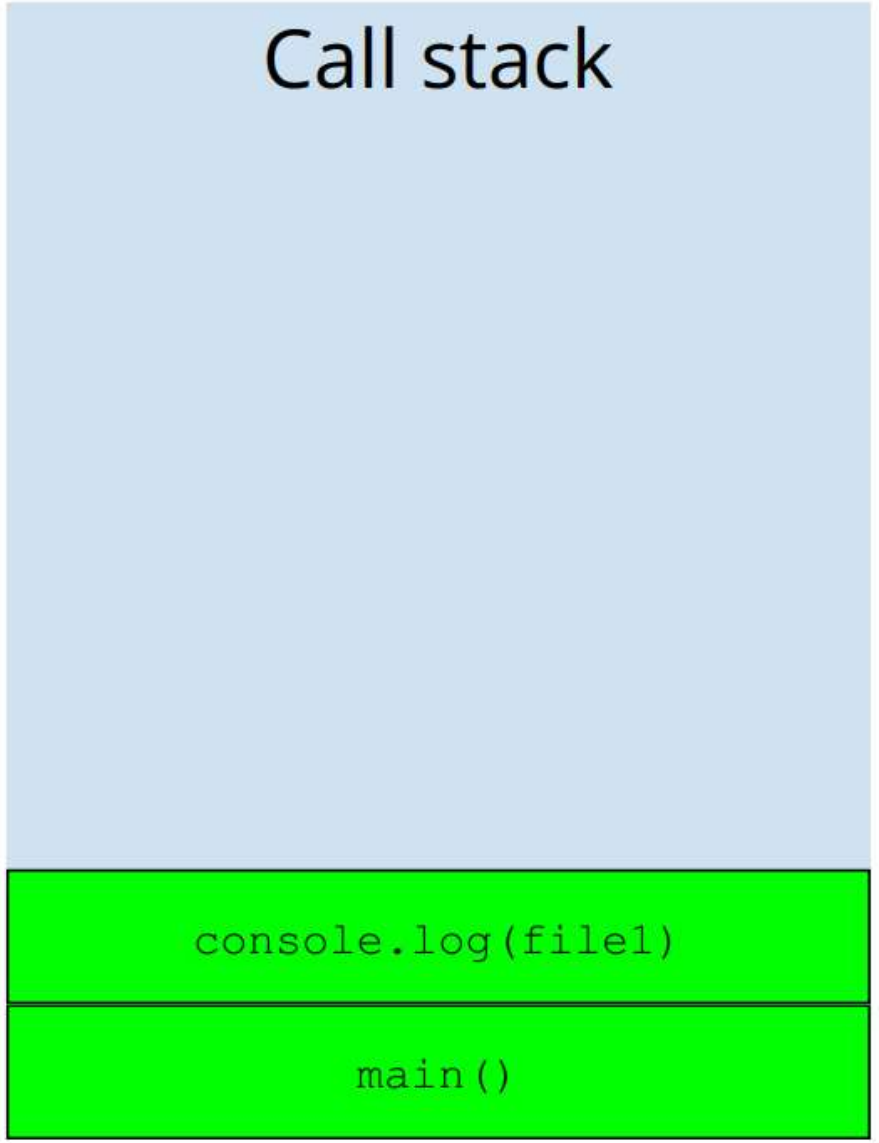
```
1  const fs = require('fs');
2
3
4  const file1 = fs.
5
6  const file2 = fs.readFi
7
8  const file3 =
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3
```



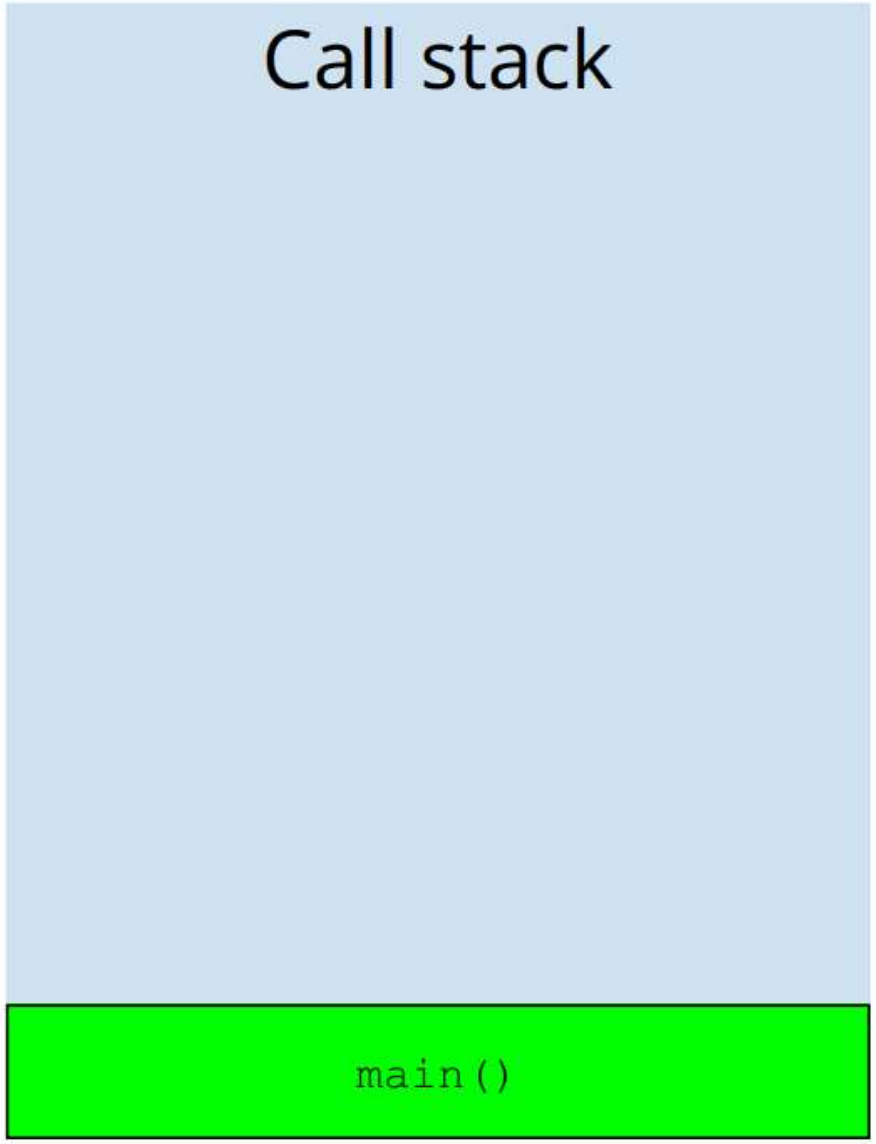
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



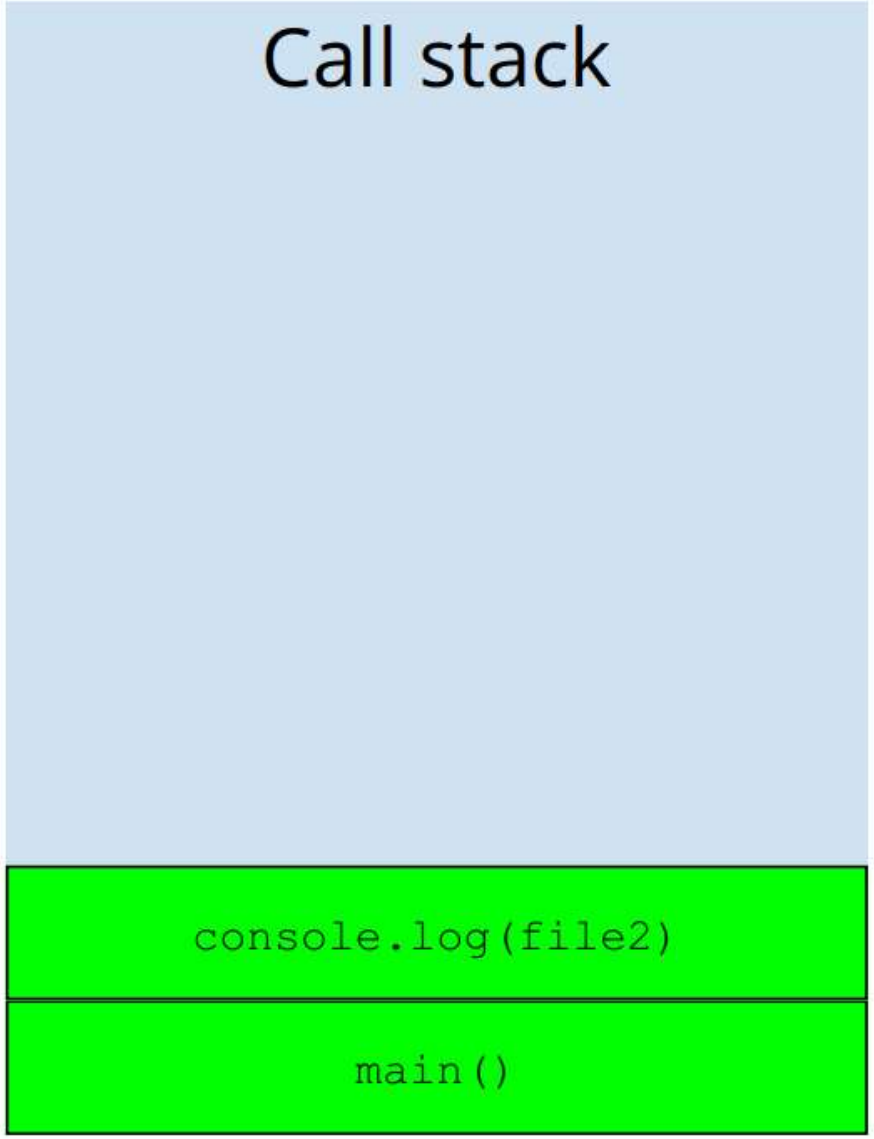
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



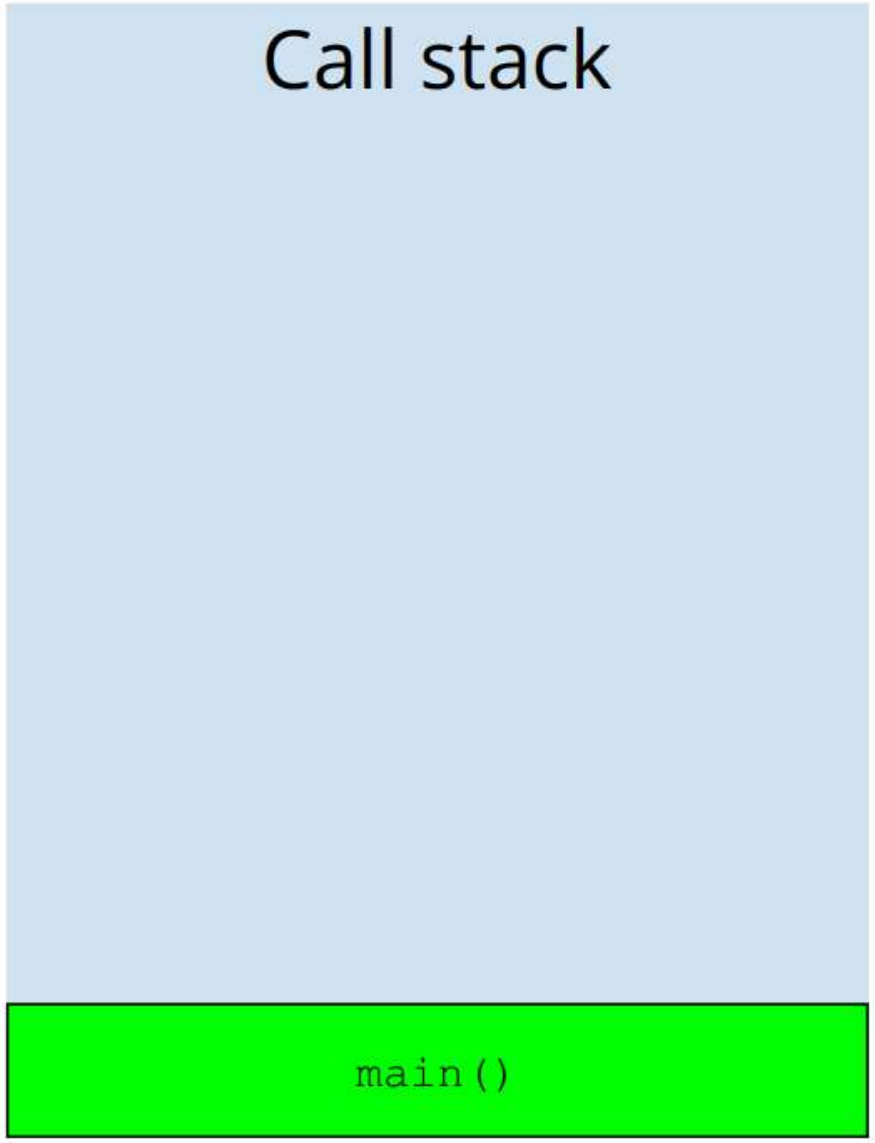
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



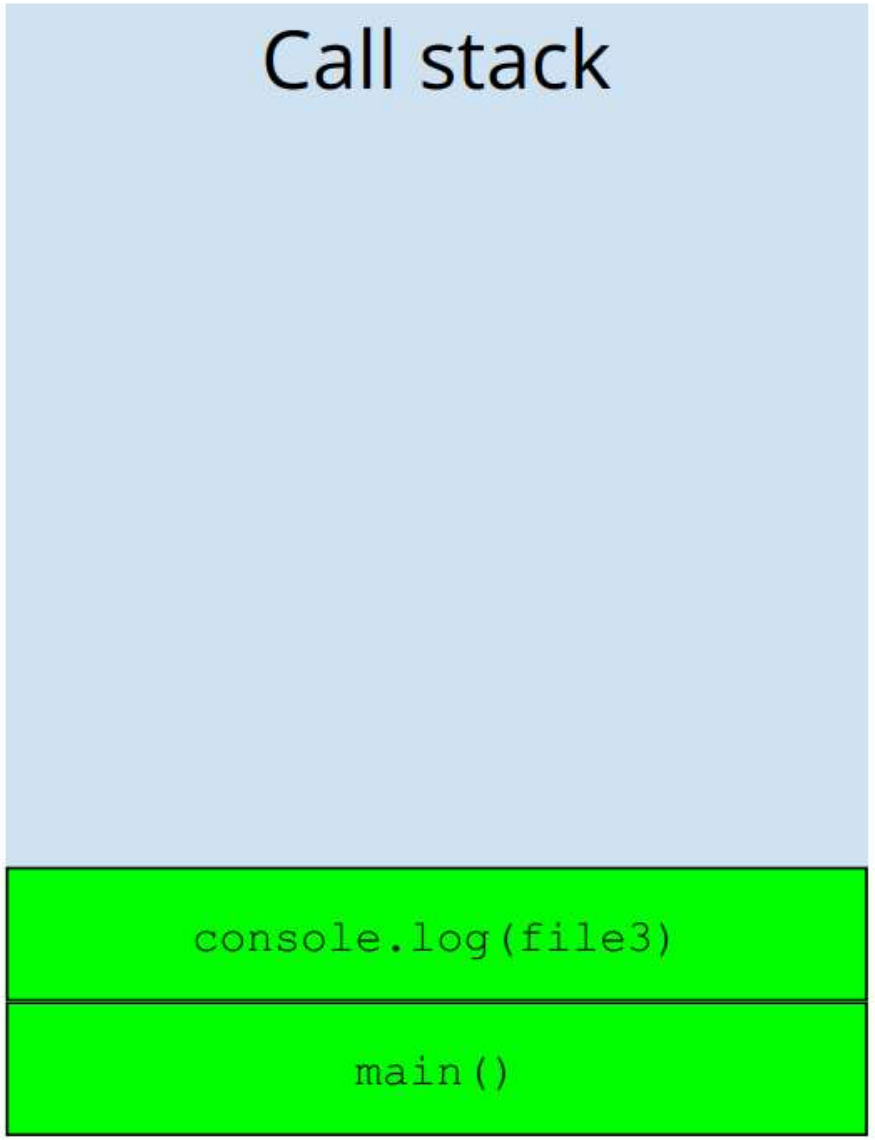
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



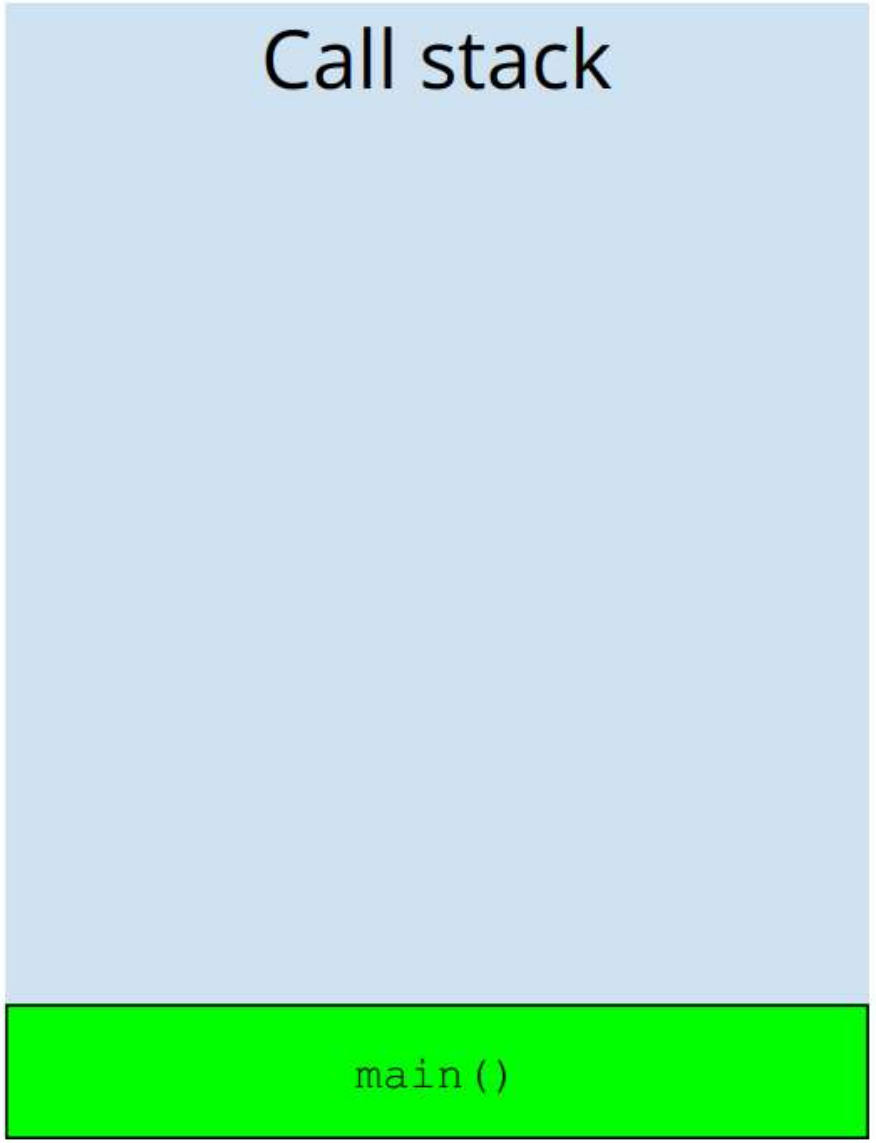
```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```



```
1  const fs = require('fs');
2
3
4  const file1 = fs.readFileSync('file1.txt');
5
6  const file2 = fs.readFileSync('file2.txt');
7
8  const file3 = fs.readFileSync('file3.txt');
9
10
11 console.log(file1);
12
13 console.log(file2);
14
15 console.log(file3);
```

Call stack

Callback

A Callback is simply a function passed as an argument to another function which will then use it (call it back)

Callback function allows other code to run in the meantime.

Node.js makes heavy use of callbacks - all the APIs of Node.js are written in such a way that they support callbacks. It allows Node.js to process a large number of requests without waiting for any function to return the result which makes Node.js highly scalable.

For example: In Node.js, when a function start reading file, it returns the control to execution environment immediately so that the next instruction can be executed. Once file I/O gets completed, callback function will get called to avoid blocking or wait for File I/O.

SetTimeout

```
setTimeout(function () {
    console.log('my function callback');
}, 5000);
```

Callback

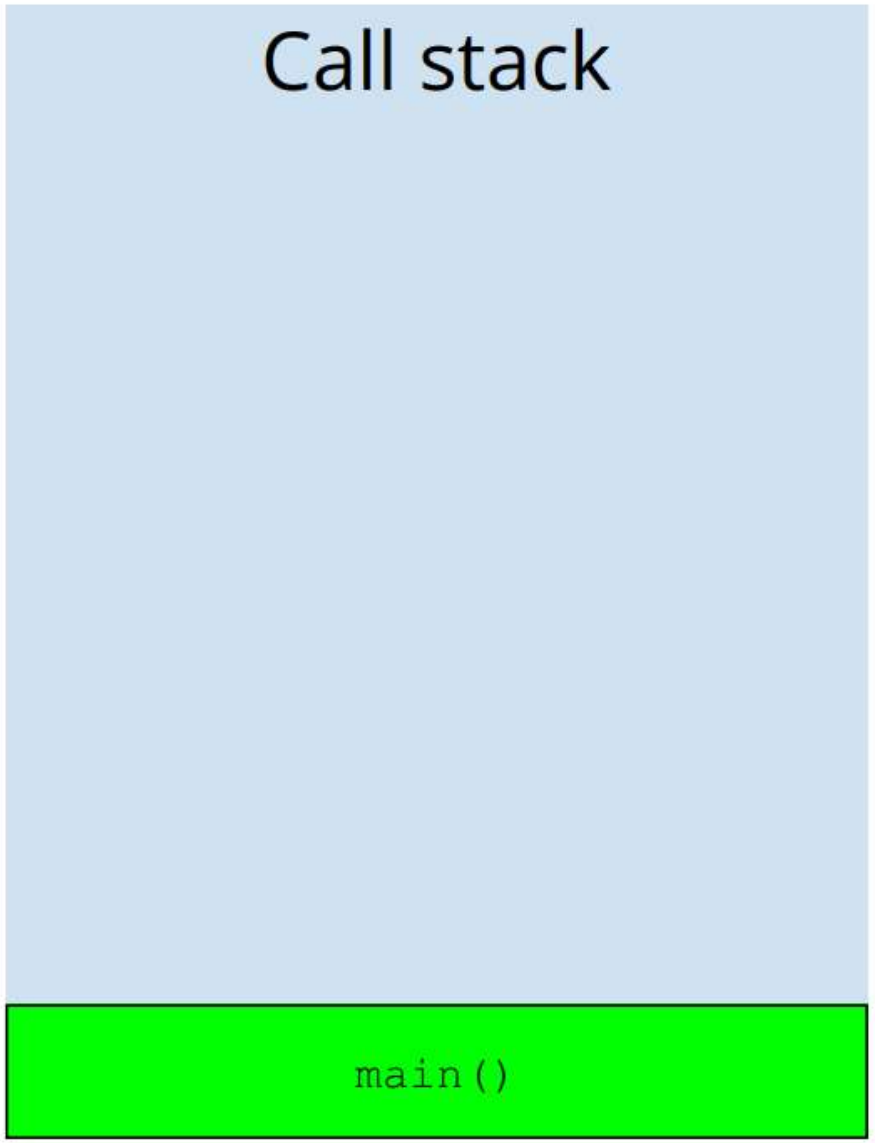
```
console.log('starting app');

setTimeout(function () {
    console.log('my function callback');
}, 5000);

console.log('end app');
```

```
C:\code\Nodejs21-22\Semestr1\03 [master ≡ +8 ~0 -0 !]> node .\w3.js
starting app
end app
my function callback
```

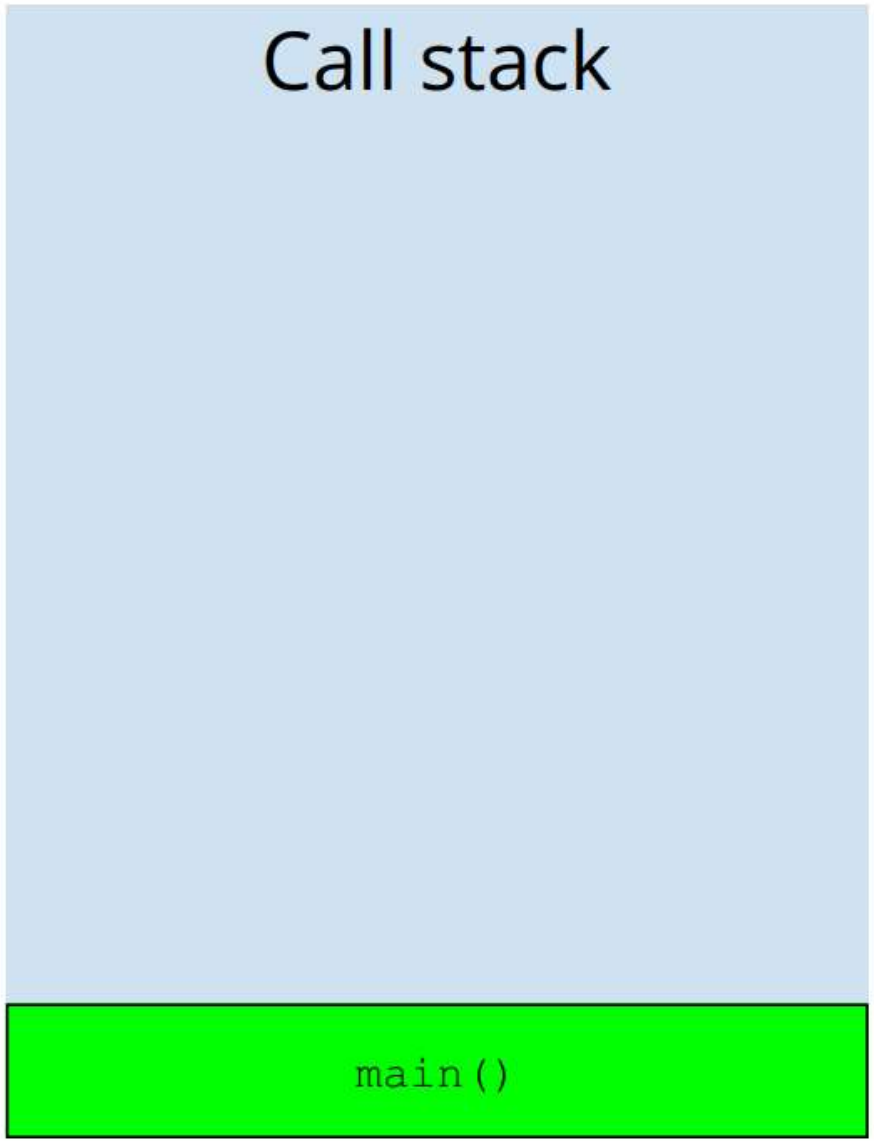
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```

Call stack

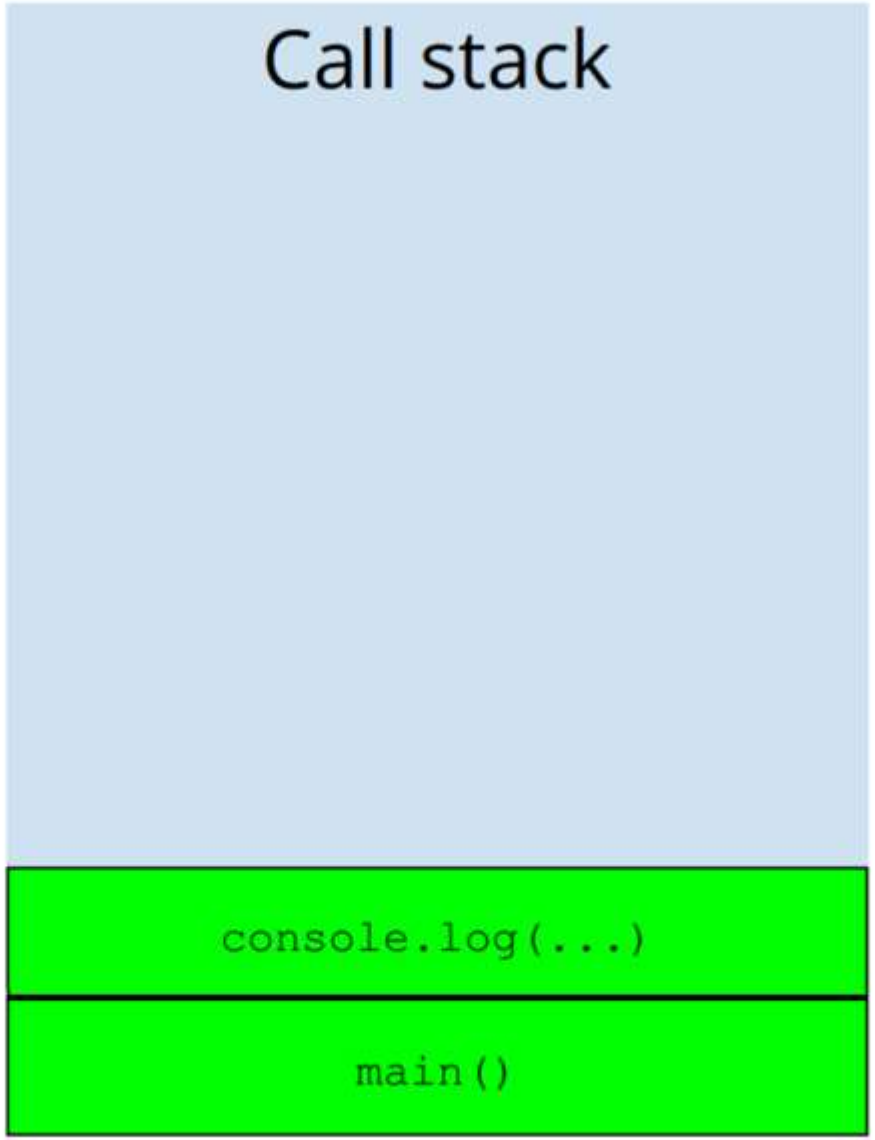
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



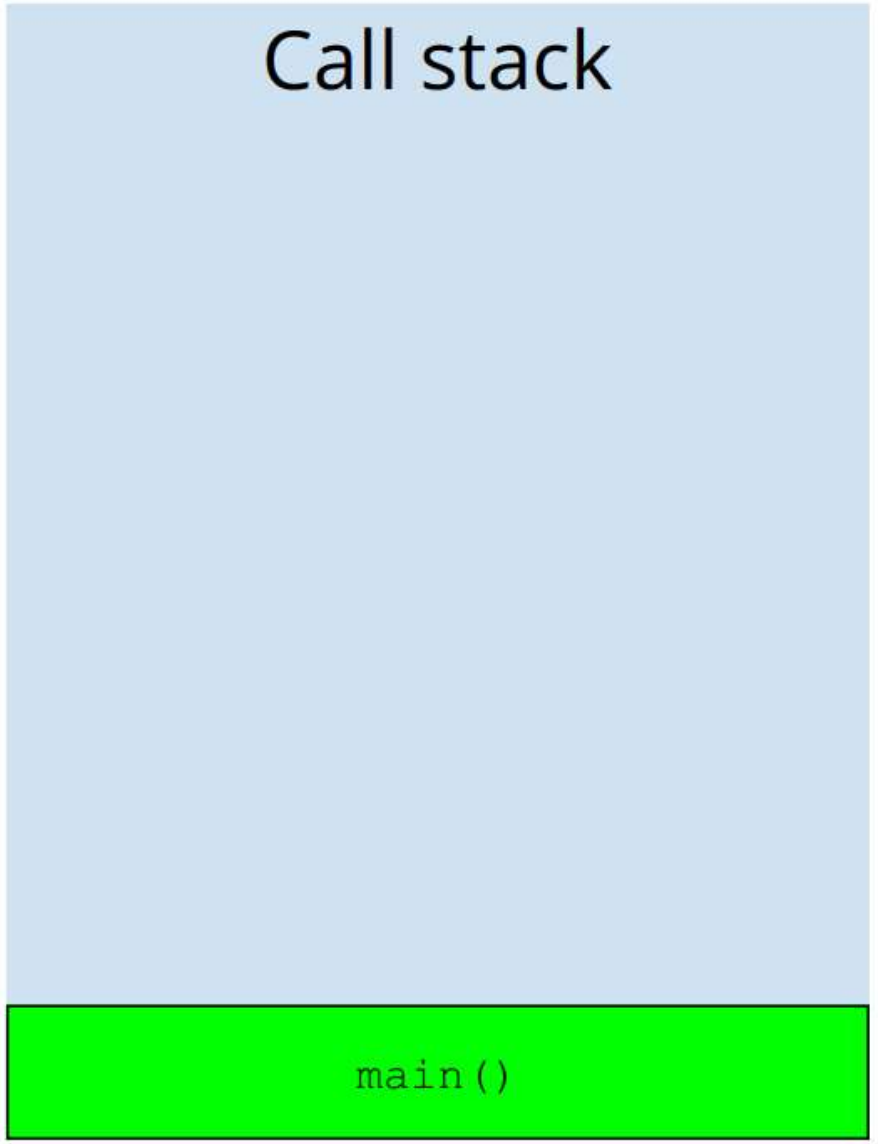
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



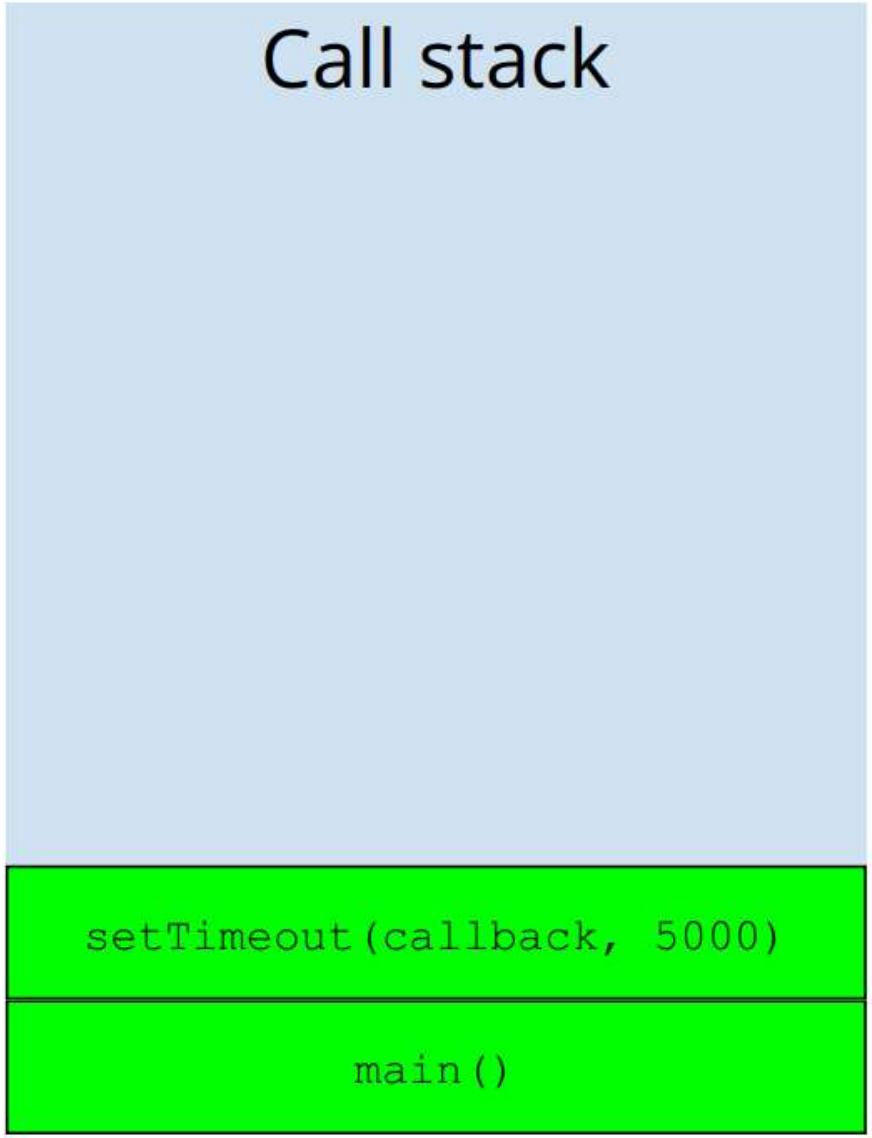
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



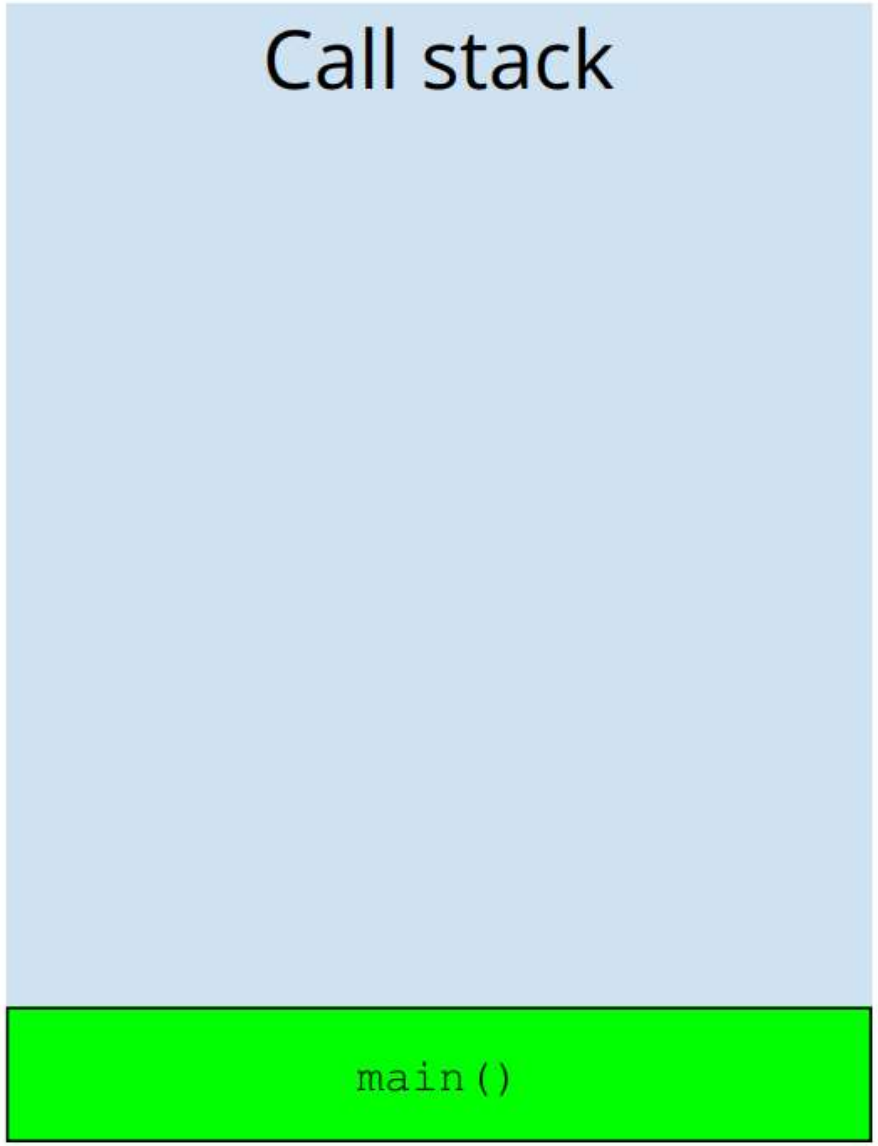
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



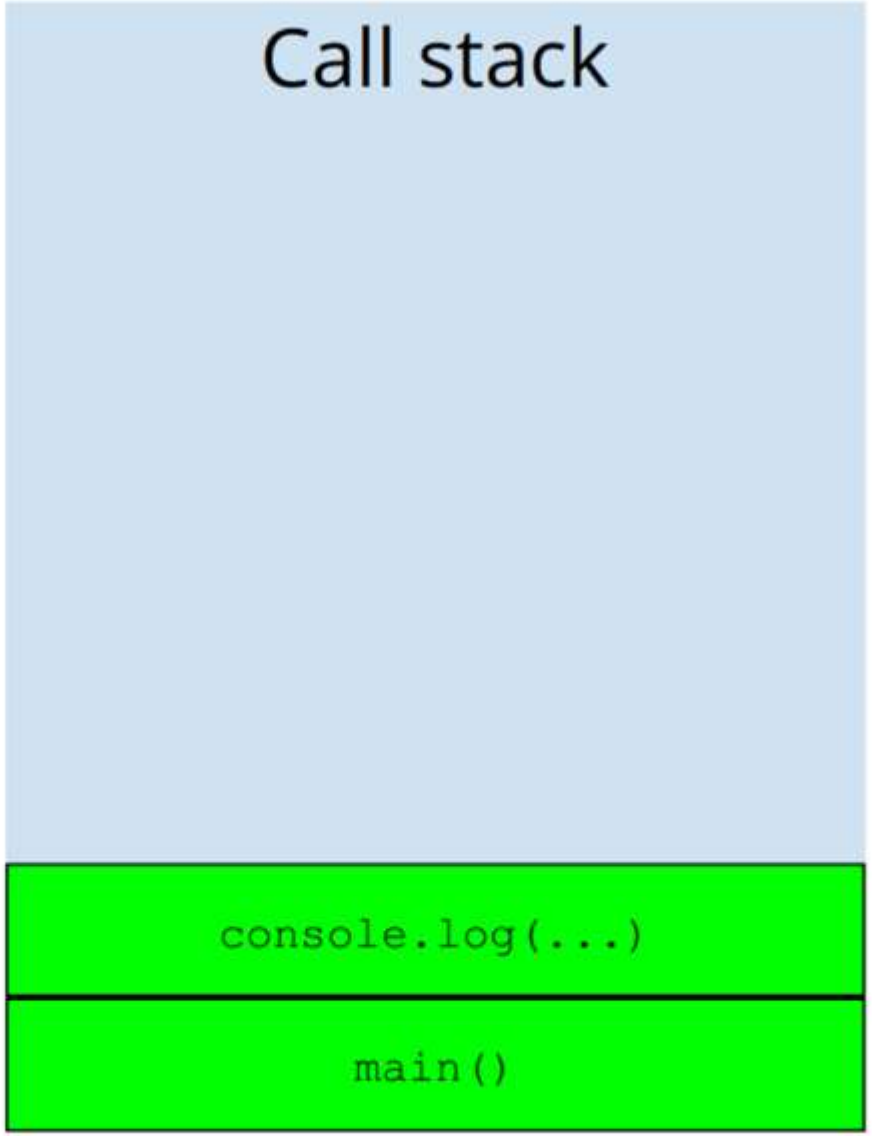
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



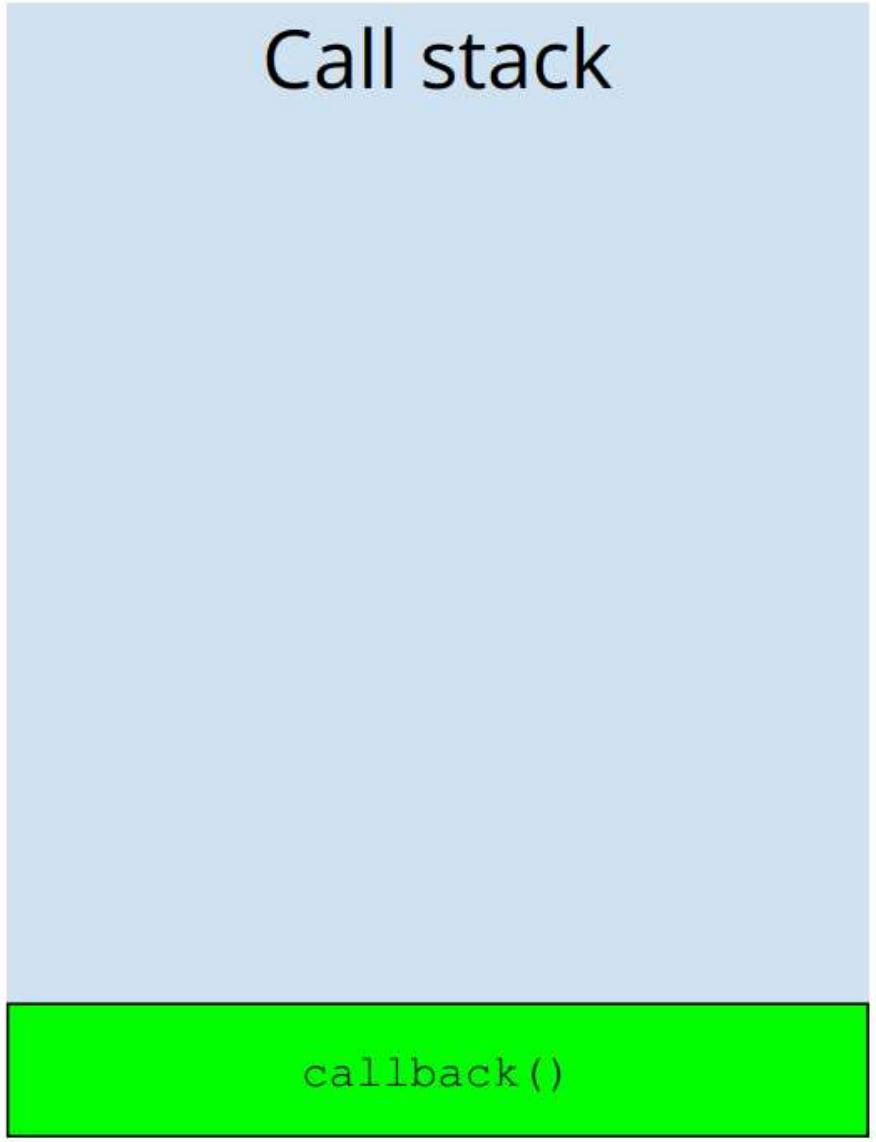
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



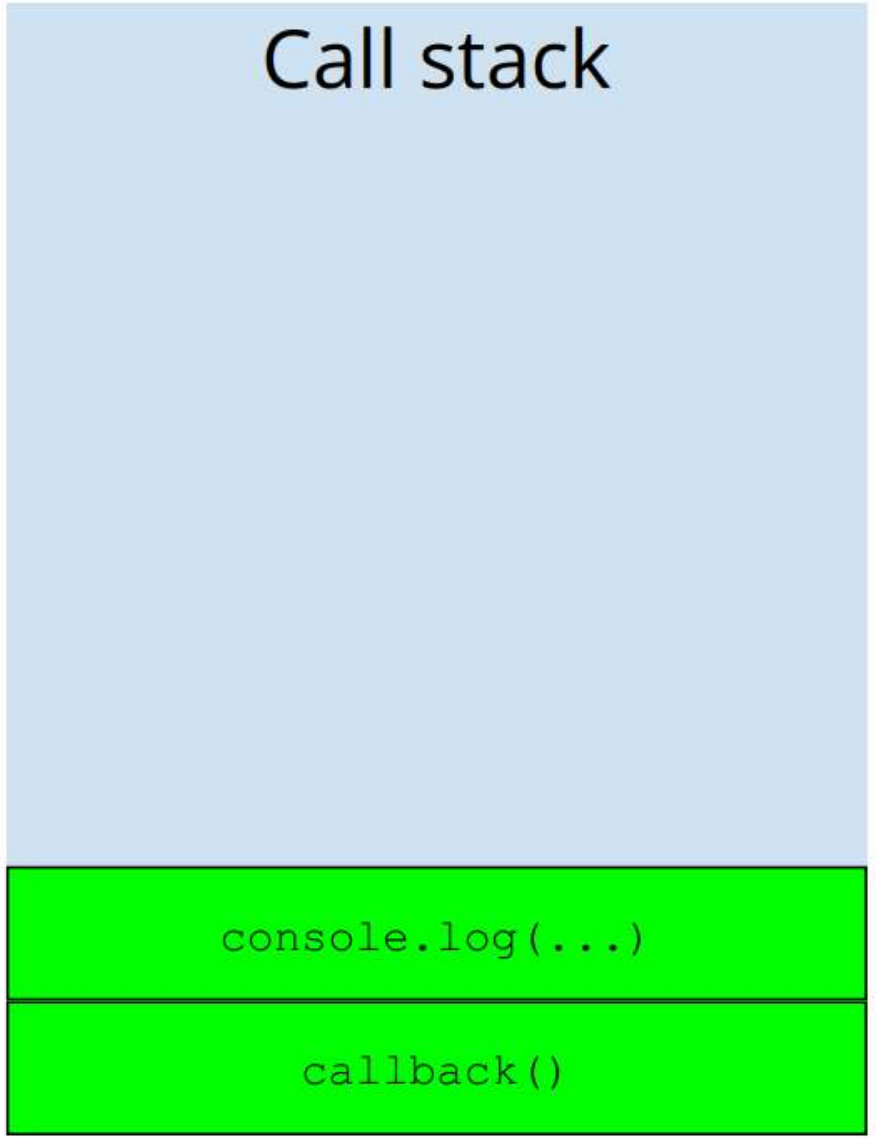
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```

Call stack

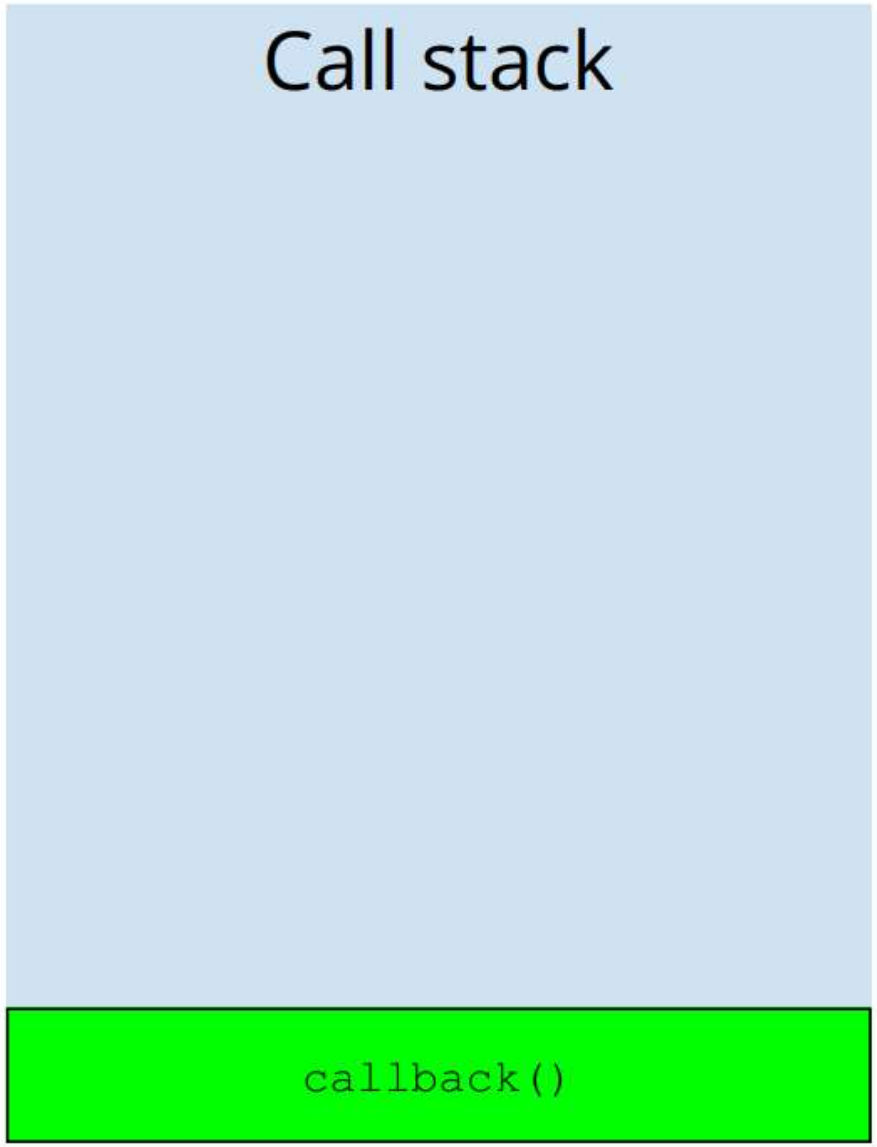
```
1  console.log('starting app');
2
3  setTimeout(function callback() {
4
5    console.log('my function callback');
6
7  }, 5000);
8
9   console.log('end app');
```



```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```

Call stack

Callback

```
console.log('starting app');

setTimeout(function () {
    console.log('my function callback');
}, 0);

console.log('end app');
```

```
C:\code\Nodejs21-22\Semestr1\03 [master ≡ +8 ~0 -0 !]> node .\w4.js
starting app
end app
my function callback
```

API

Application Programming Interface

An API is a defined specification of possible interactions with a software component.

API doesn’t have to explain what happens inside. Can describe only inputs and returned values.

Node.js APIs

Timers

File System

Network

…

Task/Callback/Message queue

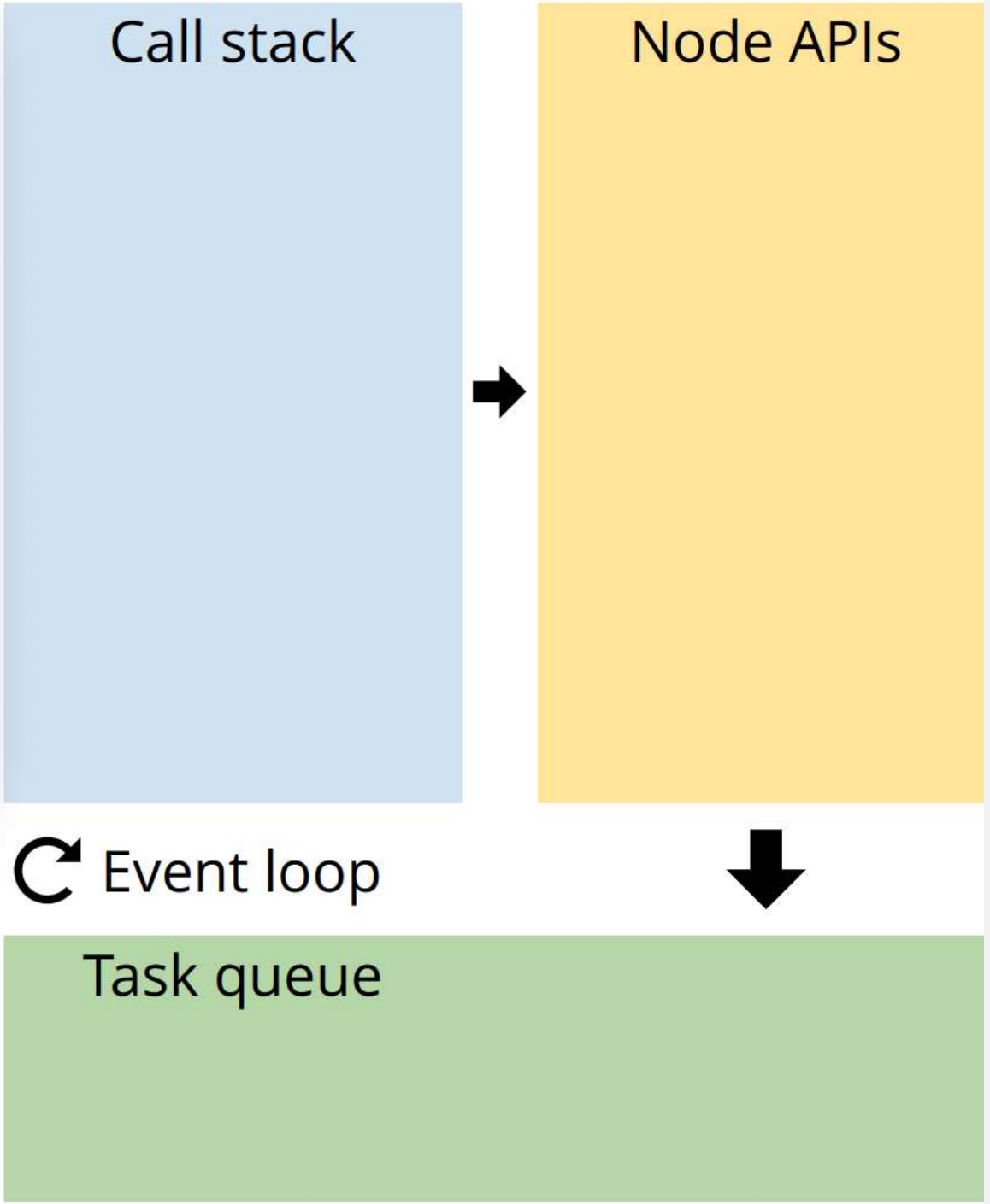
Event loop

The event loop is what allows Node.js to perform non-blocking I/O operations by offloading operations to the system kernel whenever possible.

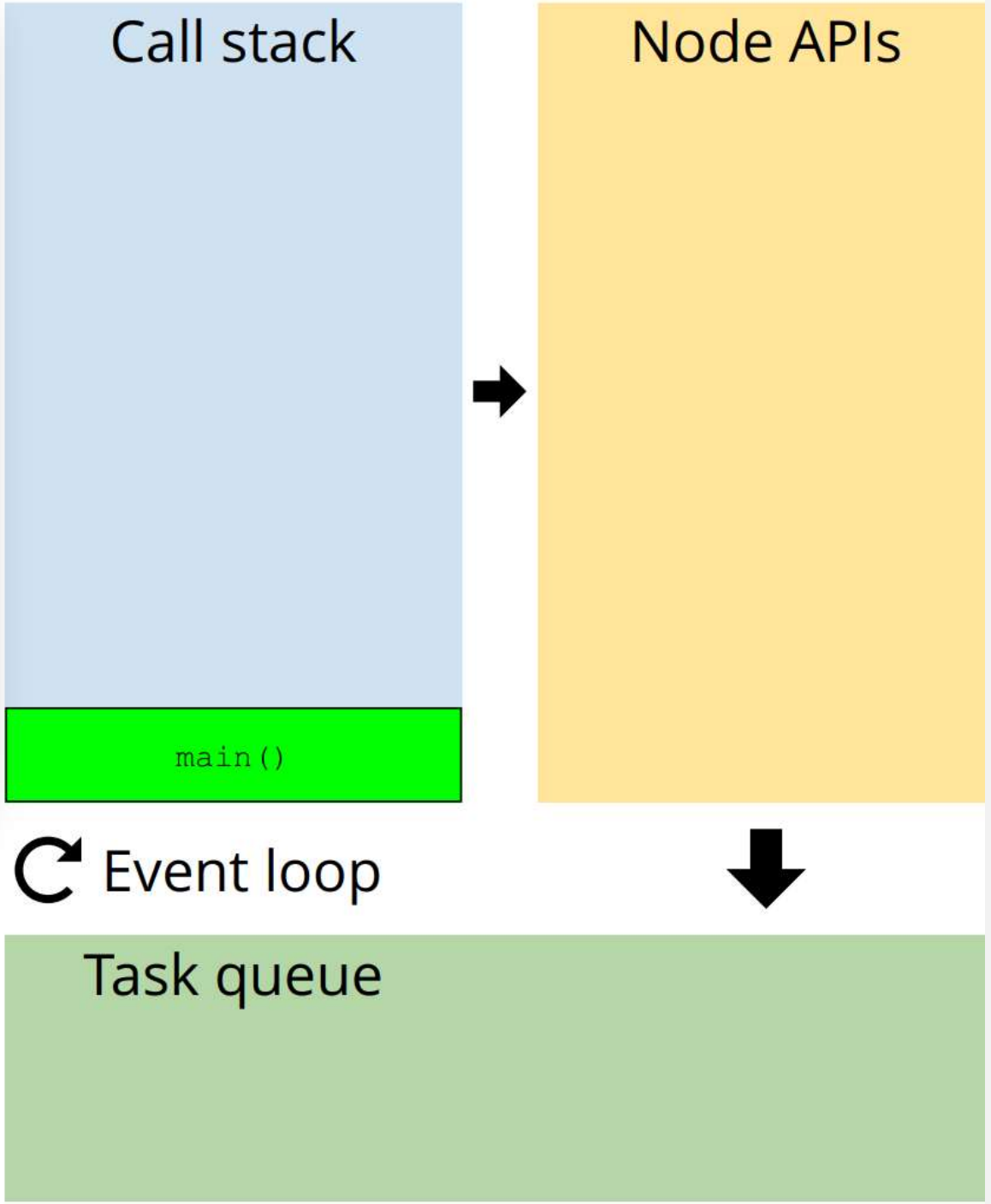
When Node.js starts, it initializes the event loop, processes the provided input script (or drops into the REPL) which may make async API calls, schedule timers then begins processing the event loop.

The loop gives priority to the call stack, and it first processes everything it finds in the call stack, and once there's nothing in there, it goes to pick up things in the task queue.

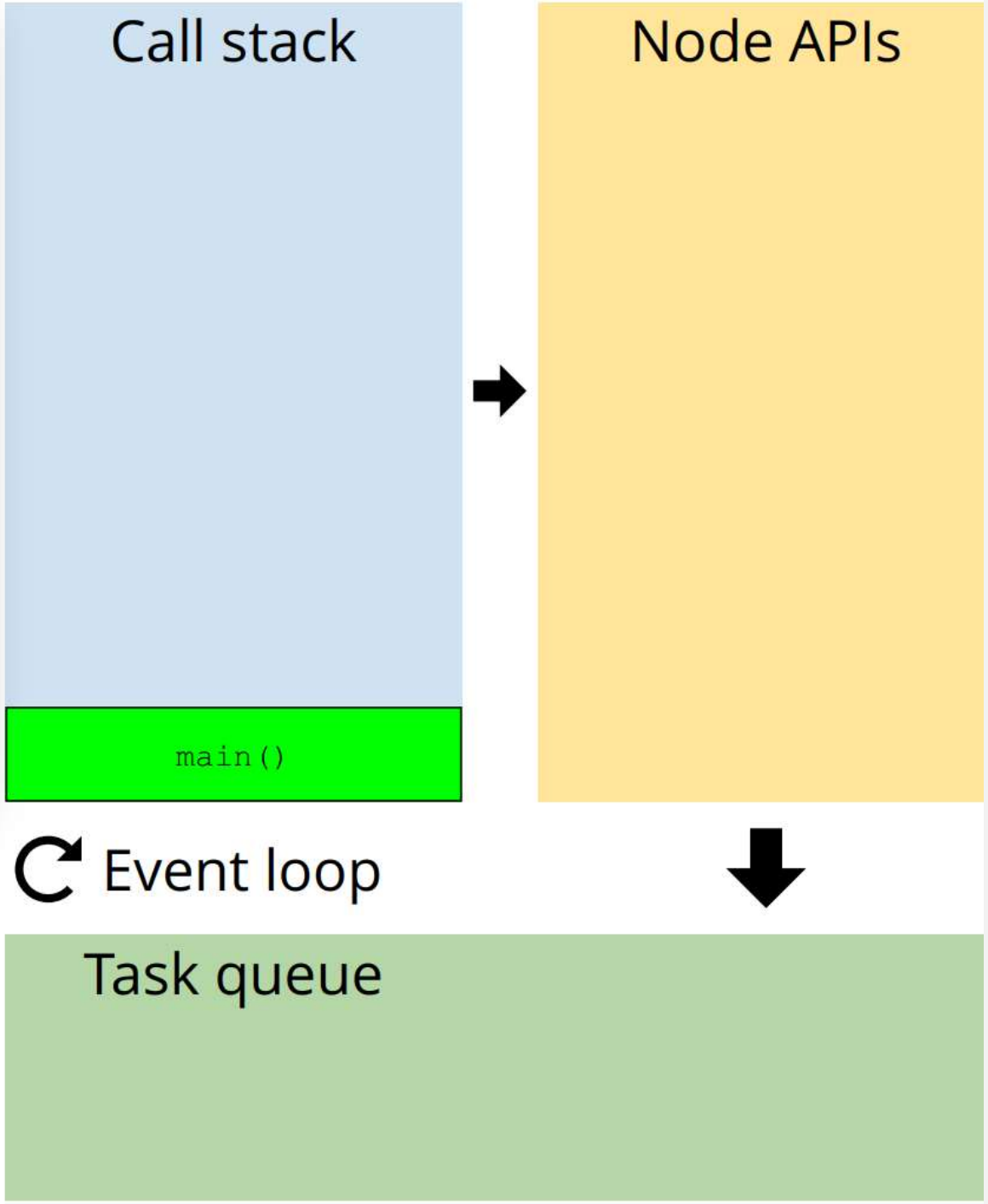
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



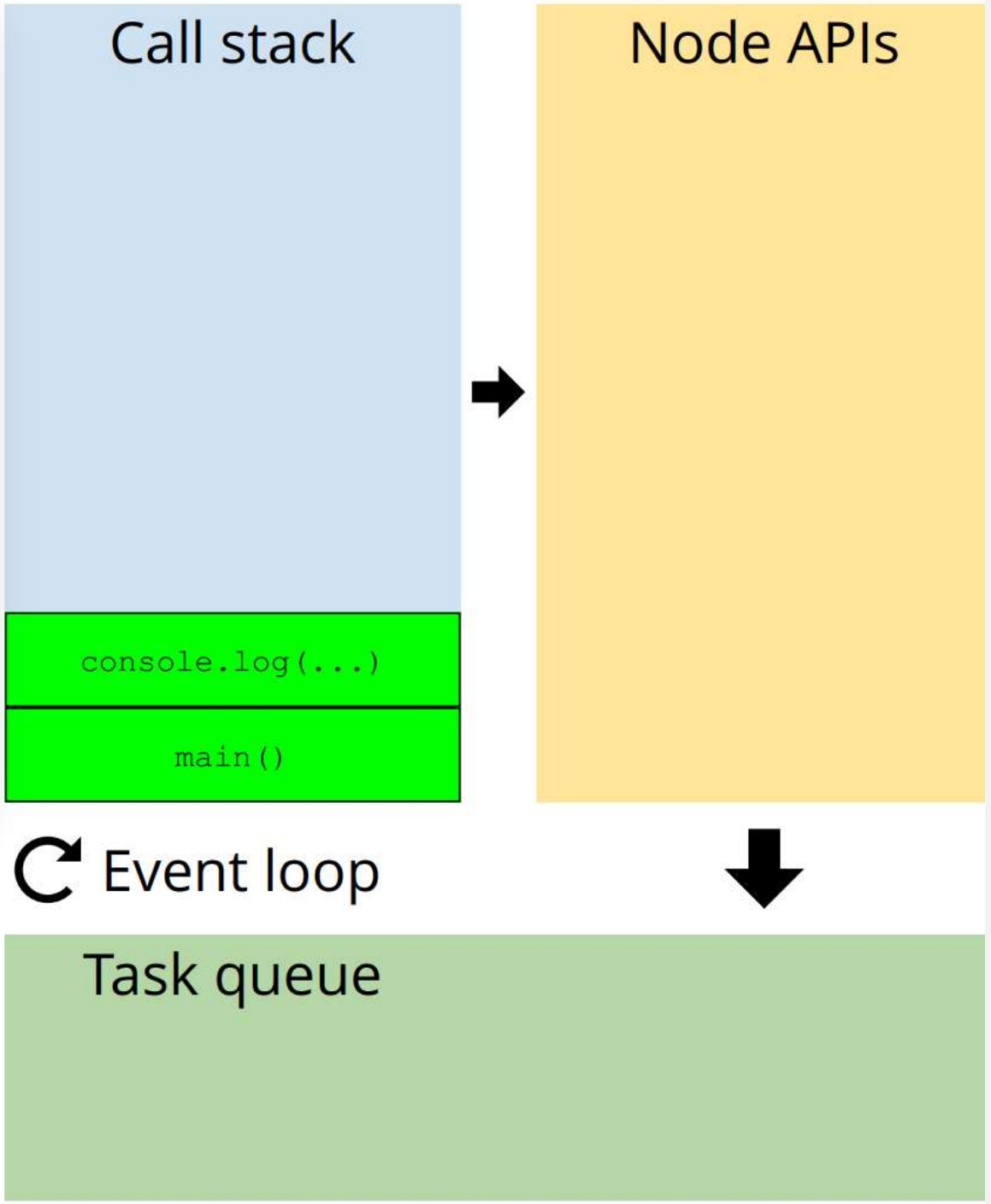
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



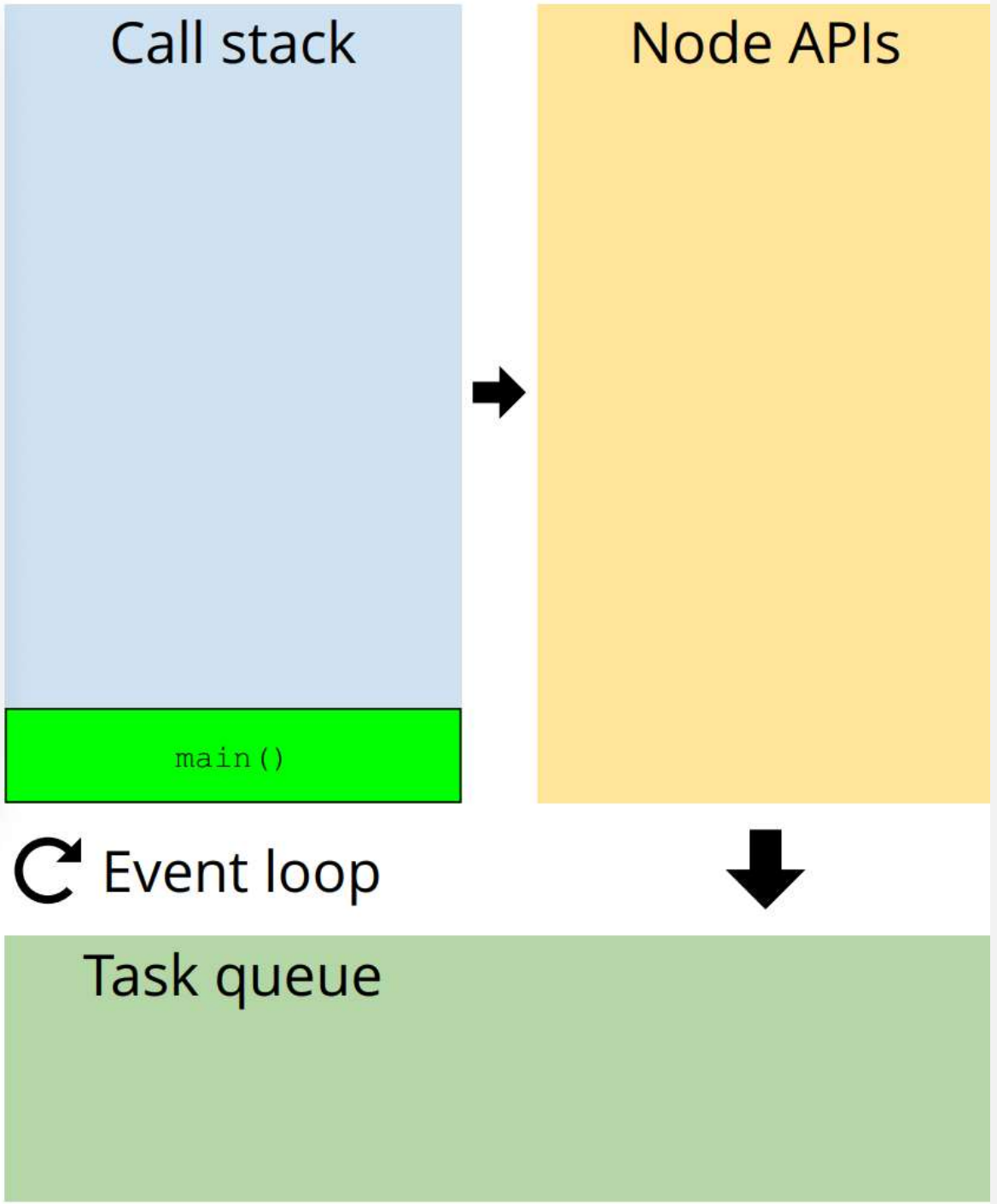
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



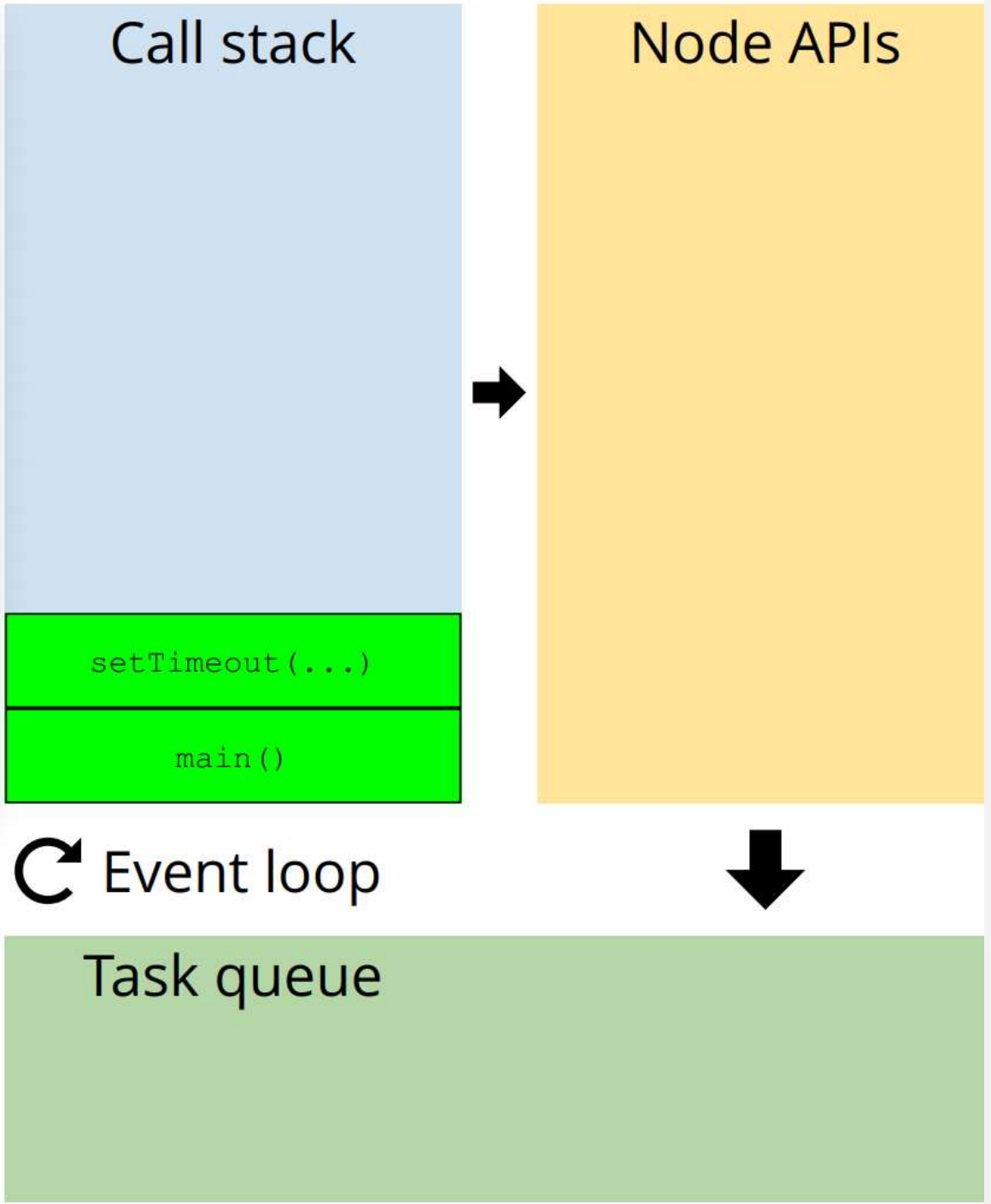
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



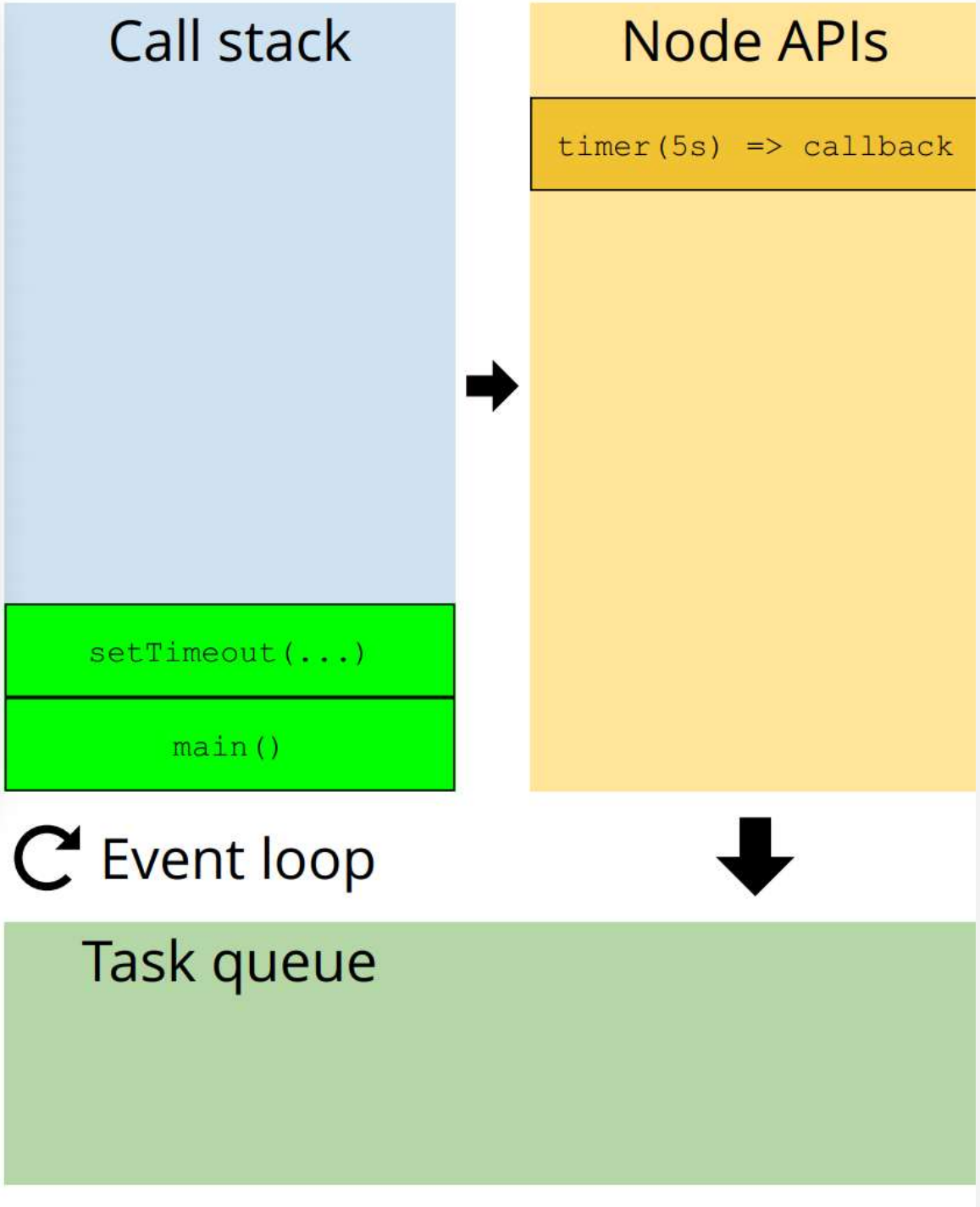
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



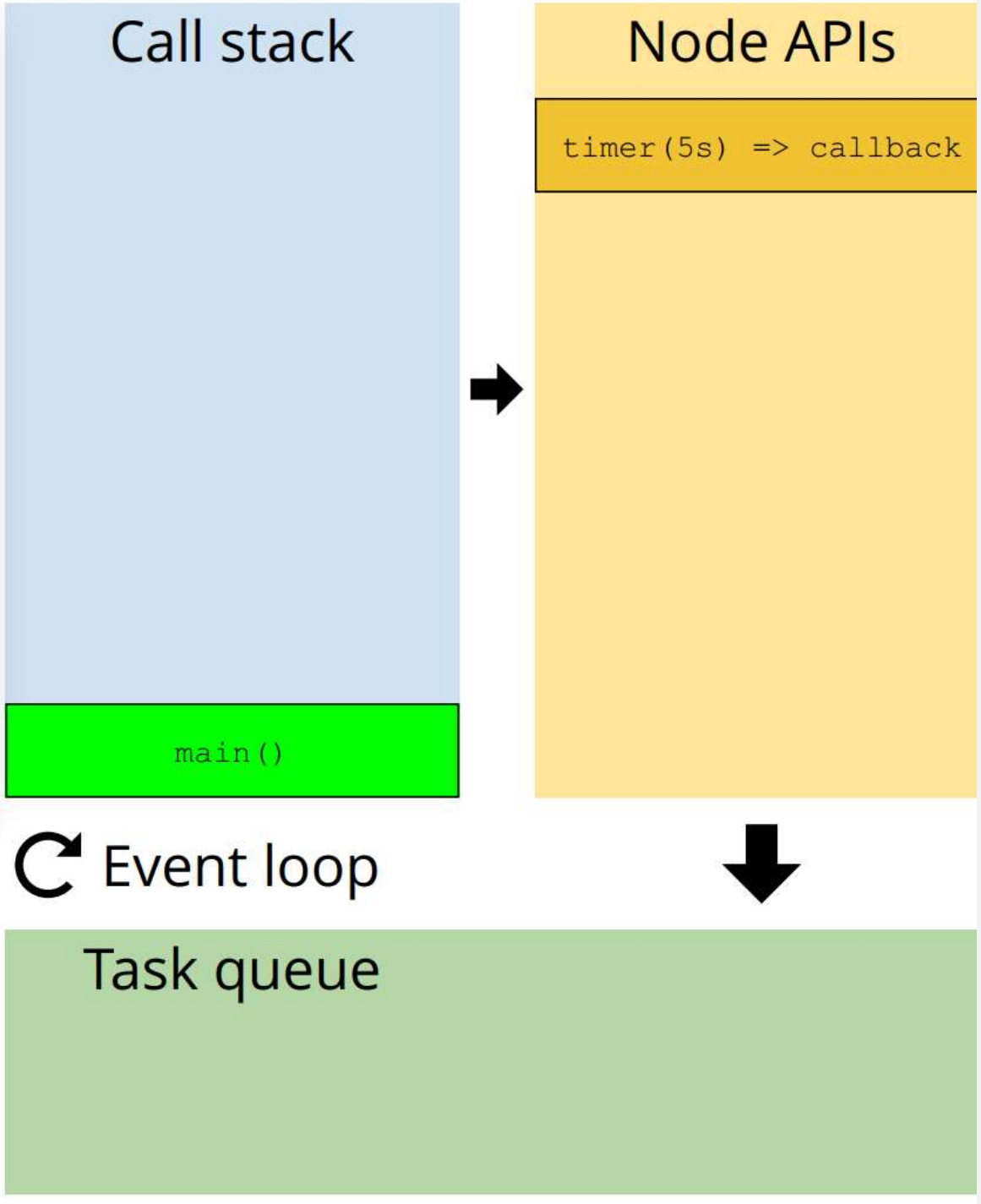
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



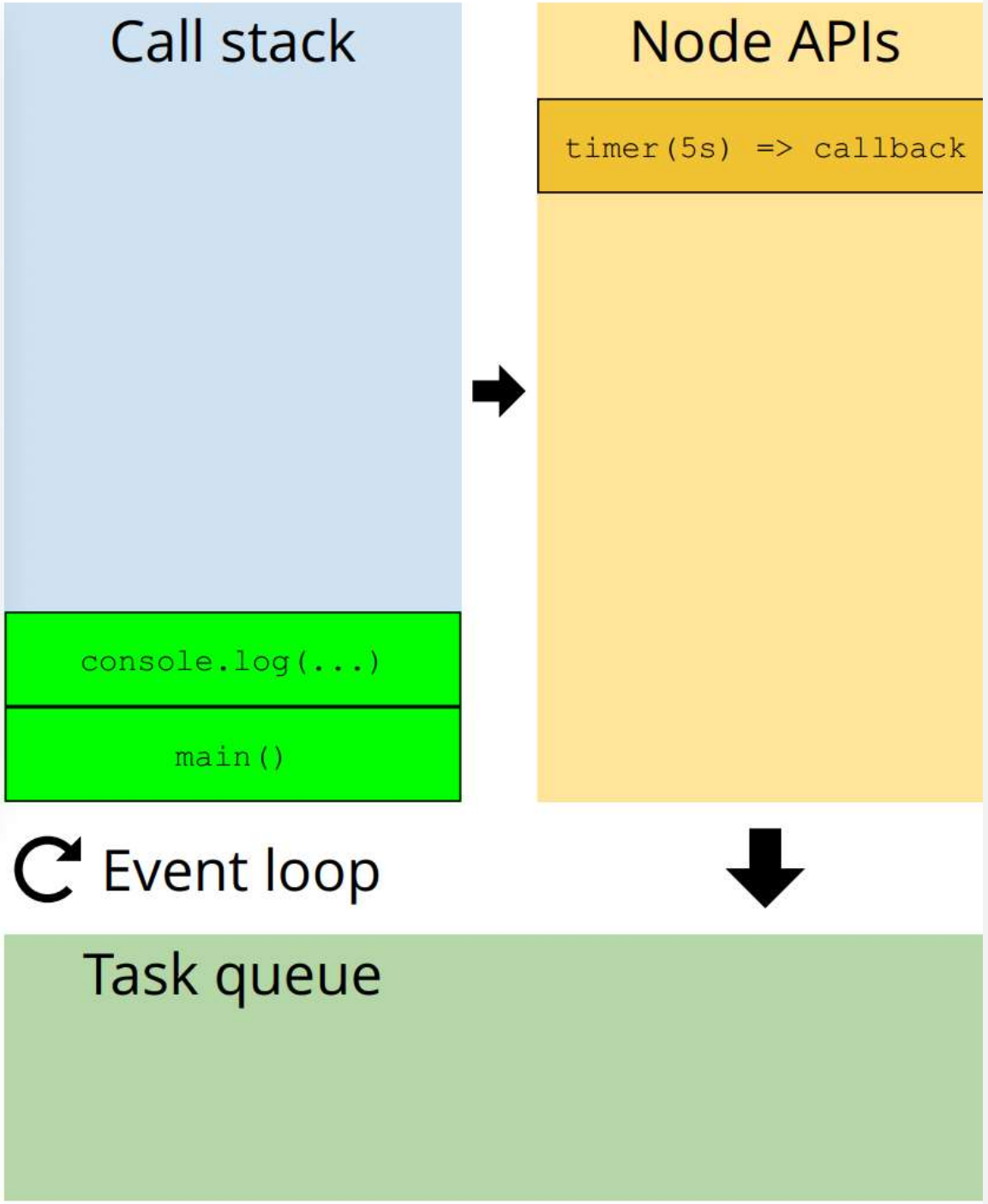
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



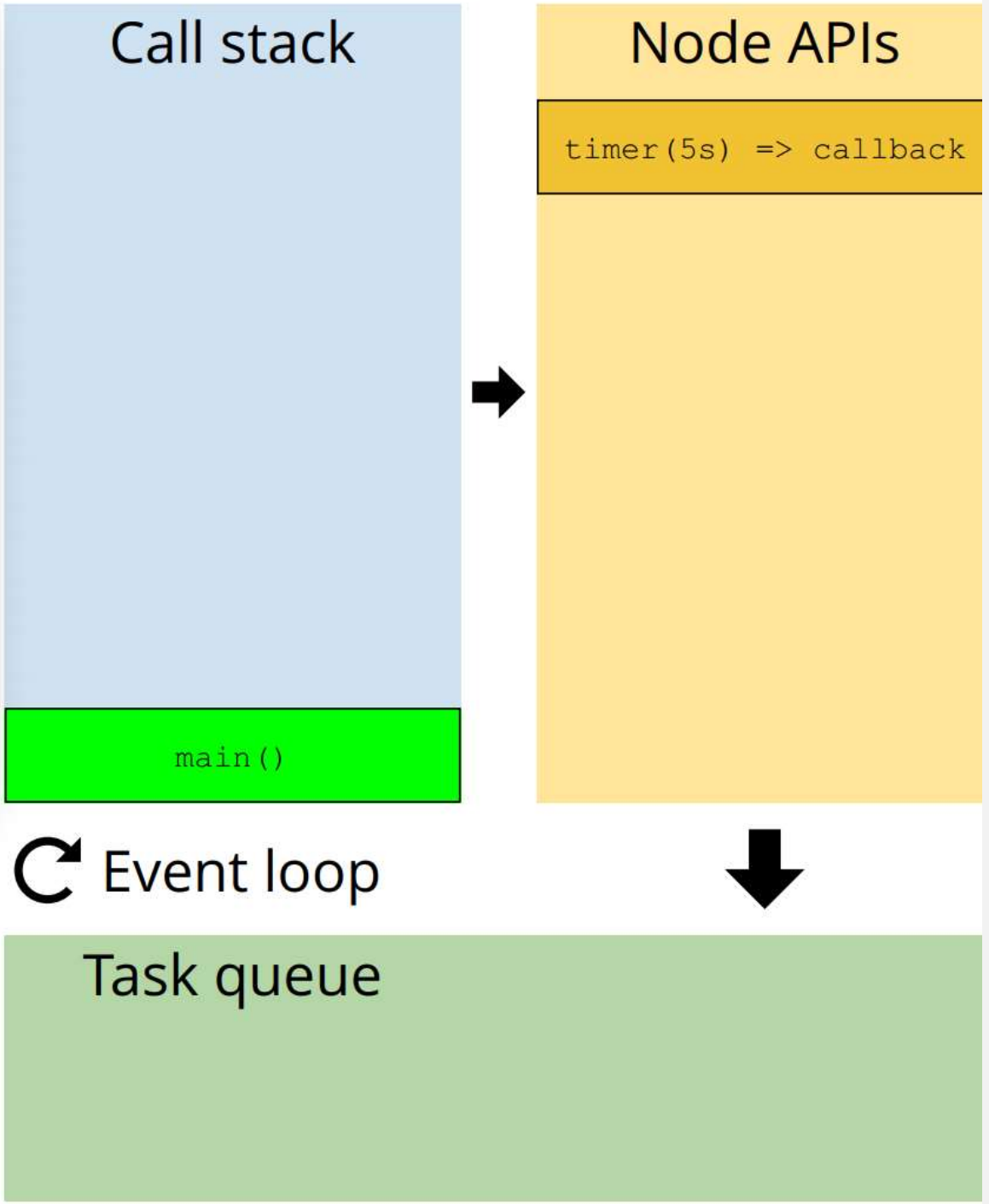
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



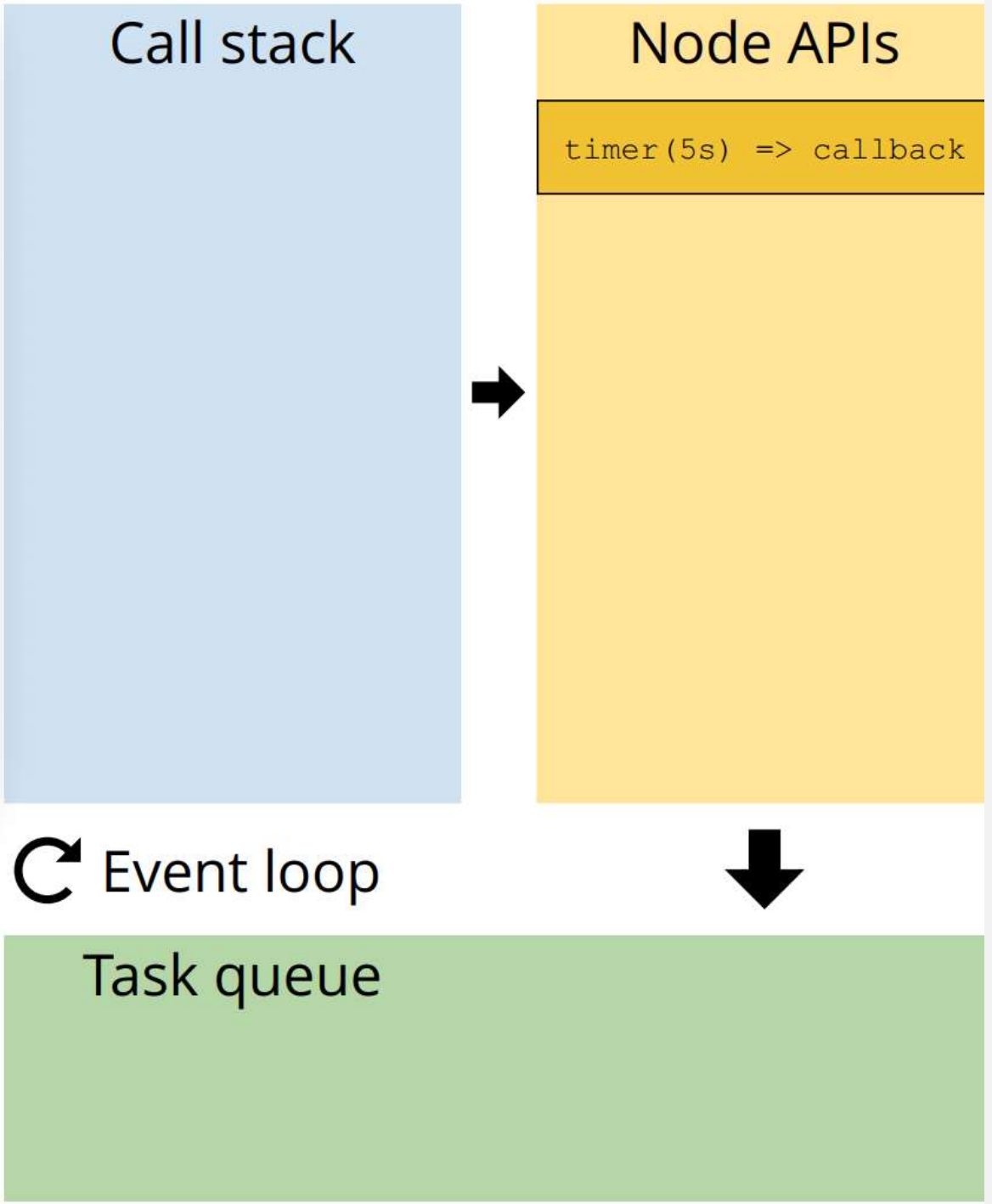
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



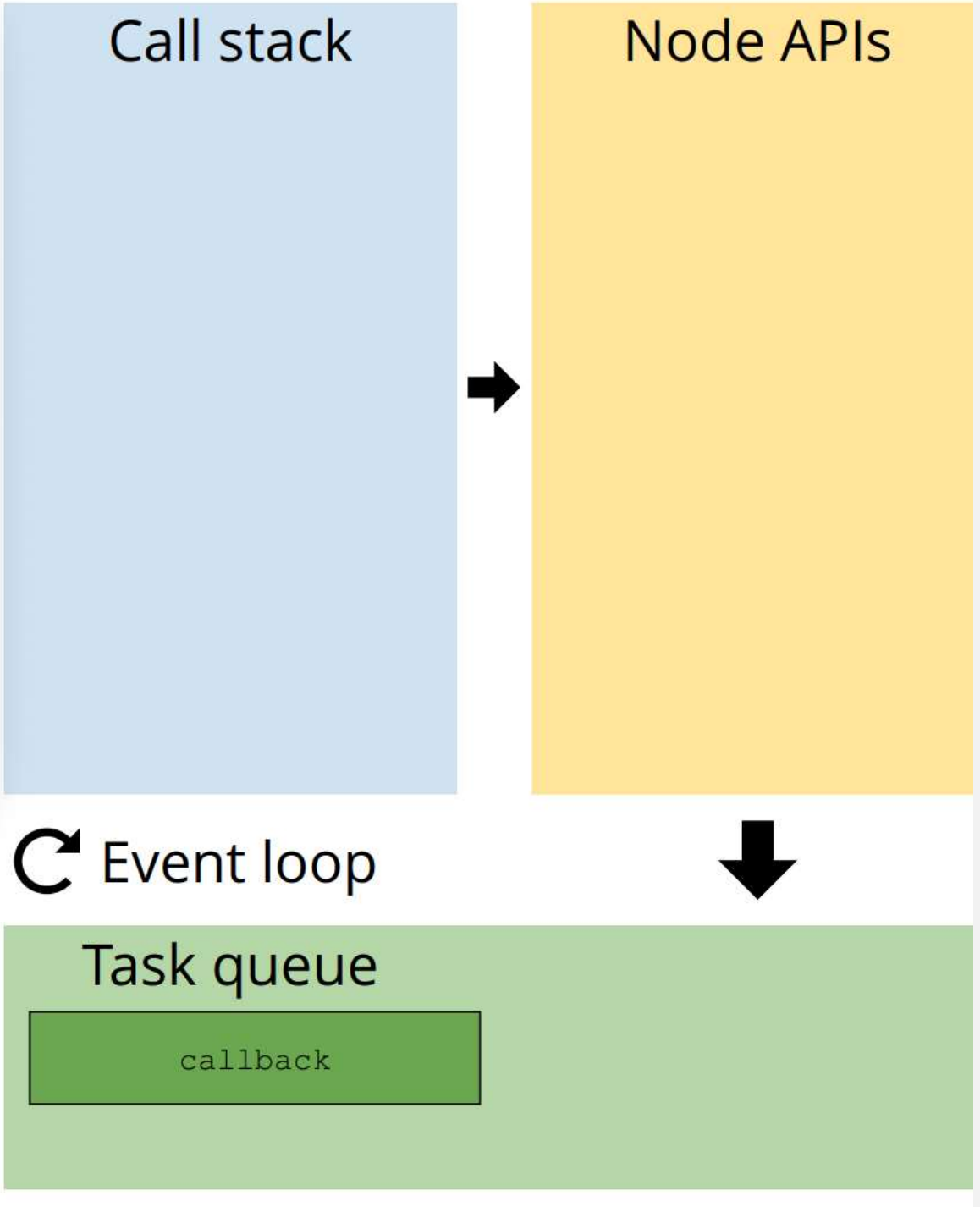
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



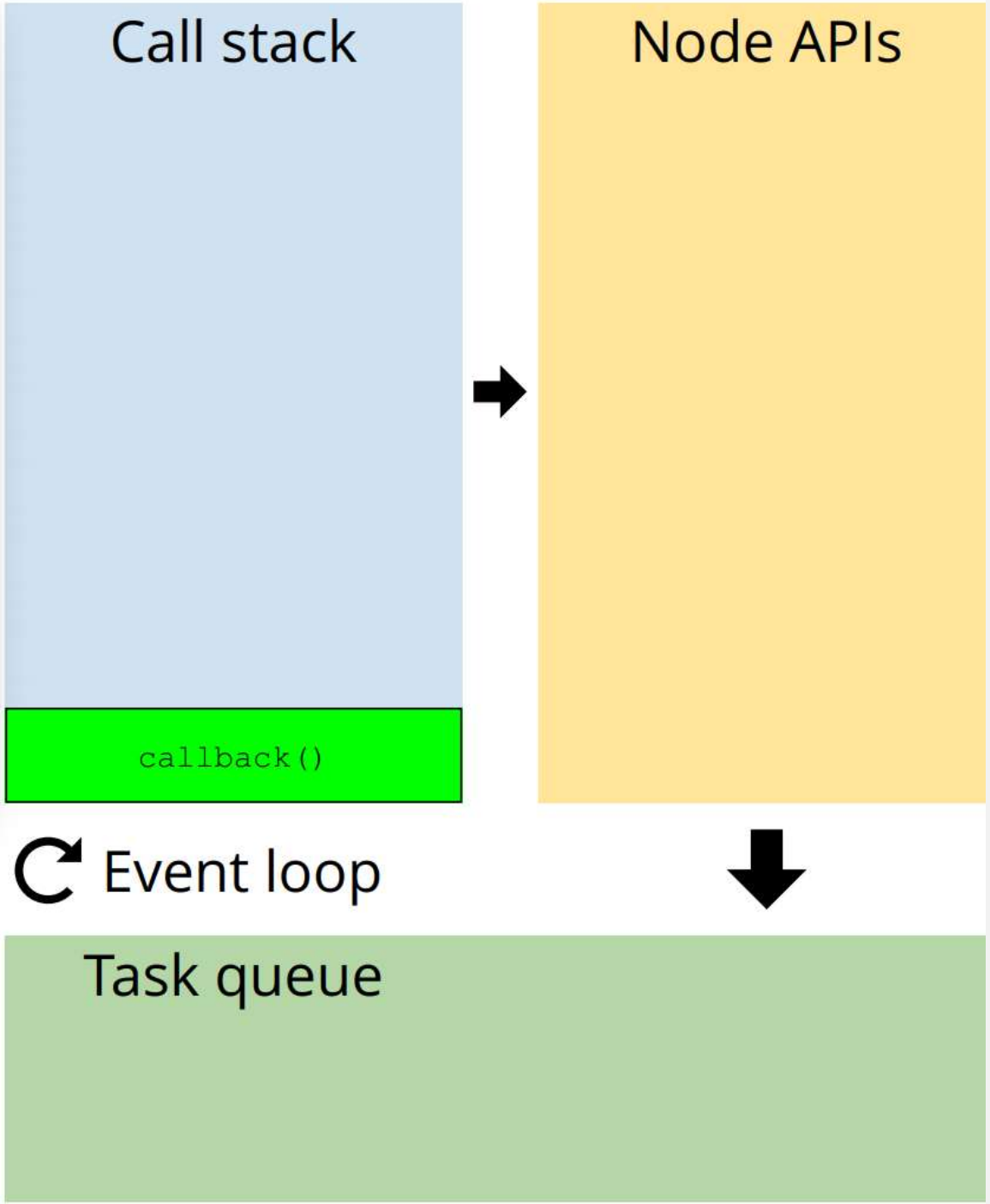
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



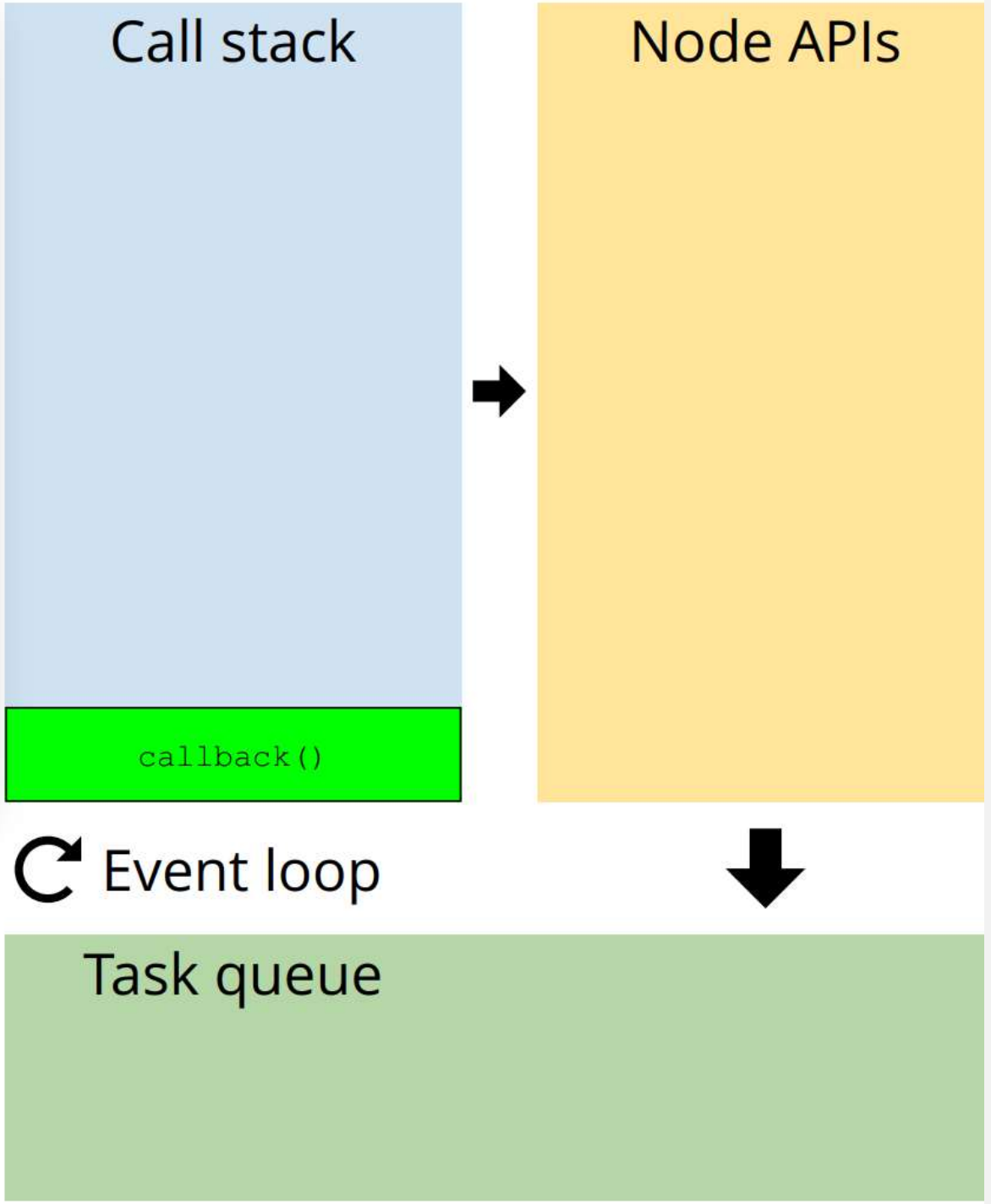
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



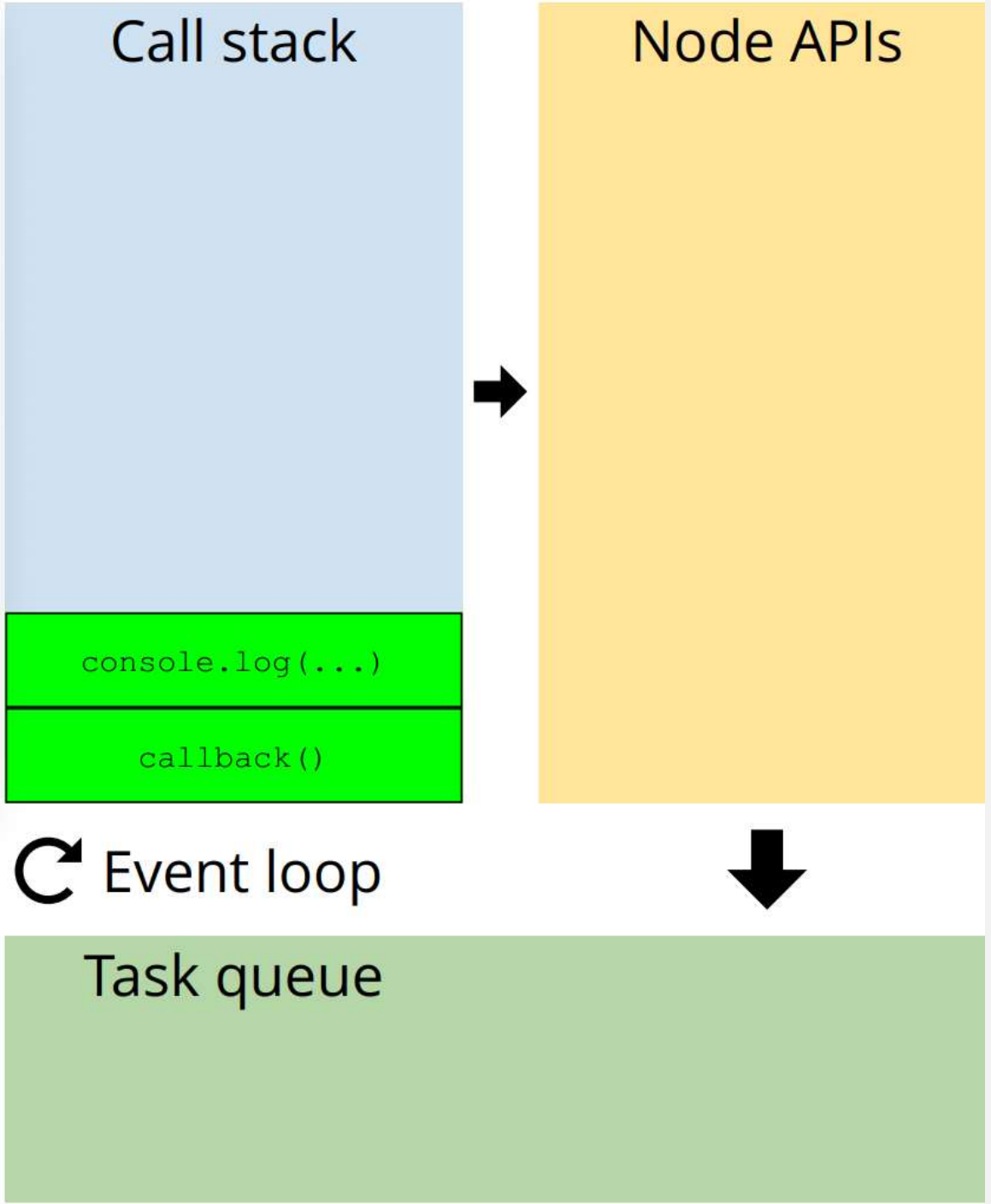
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



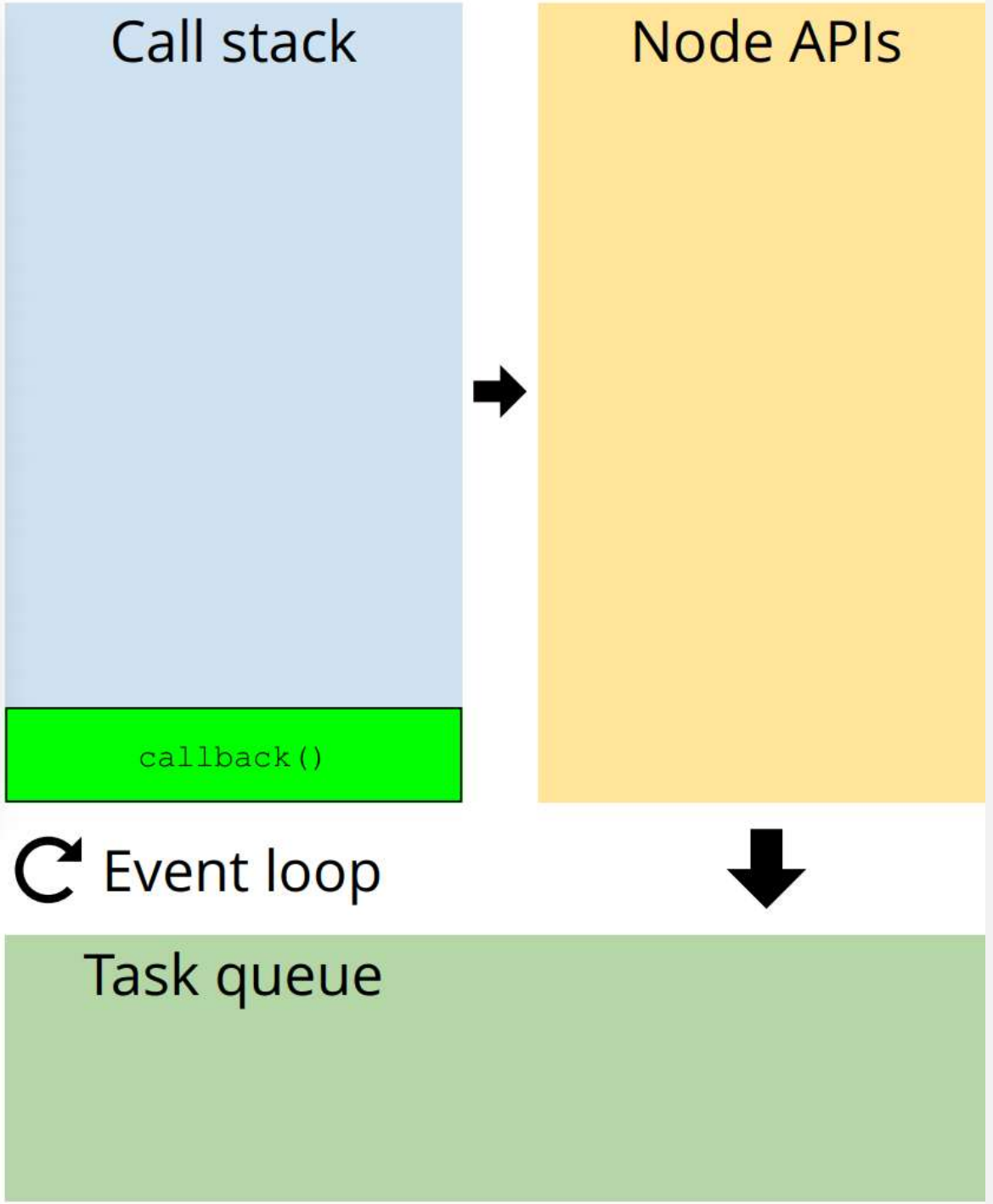
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



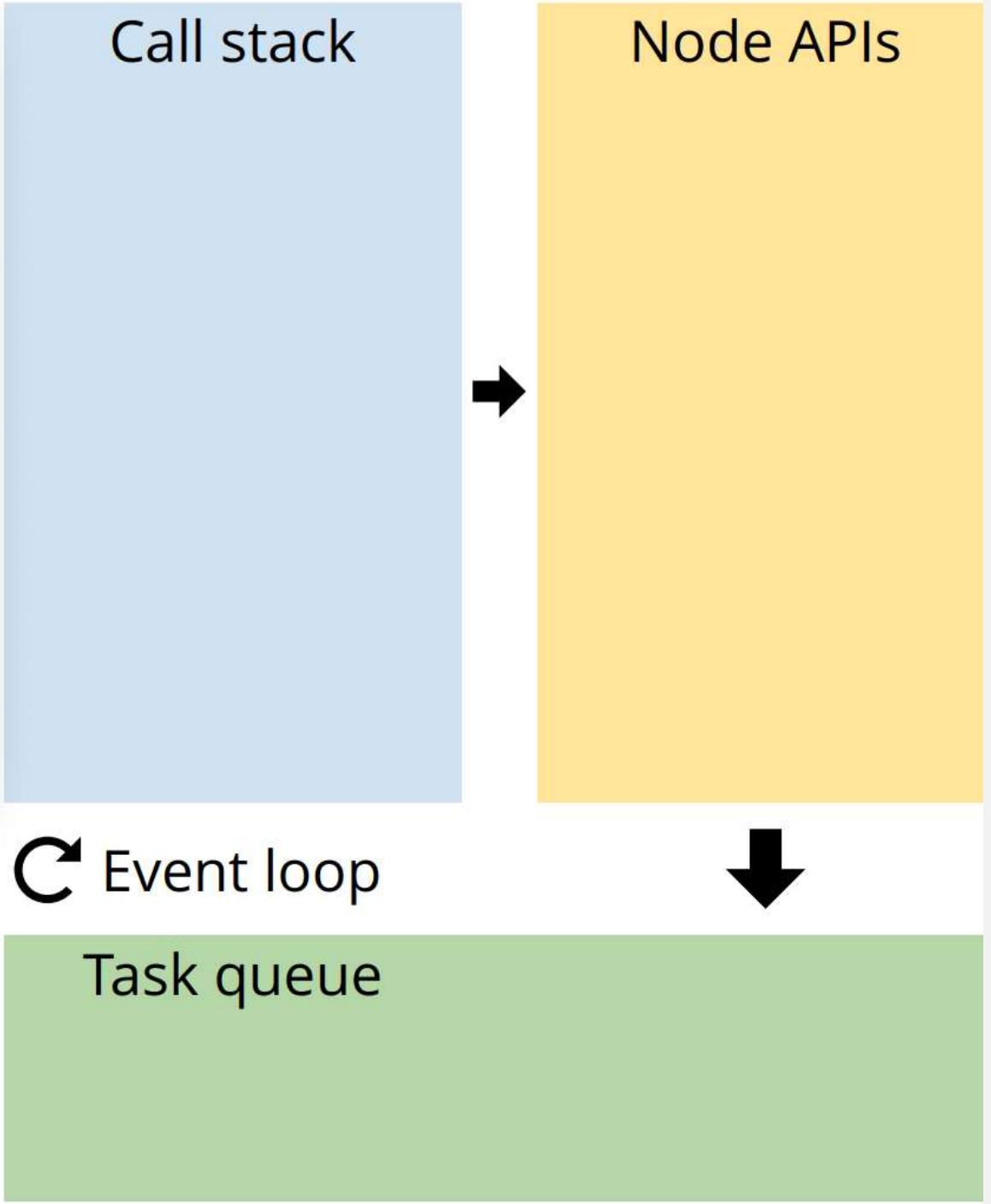
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



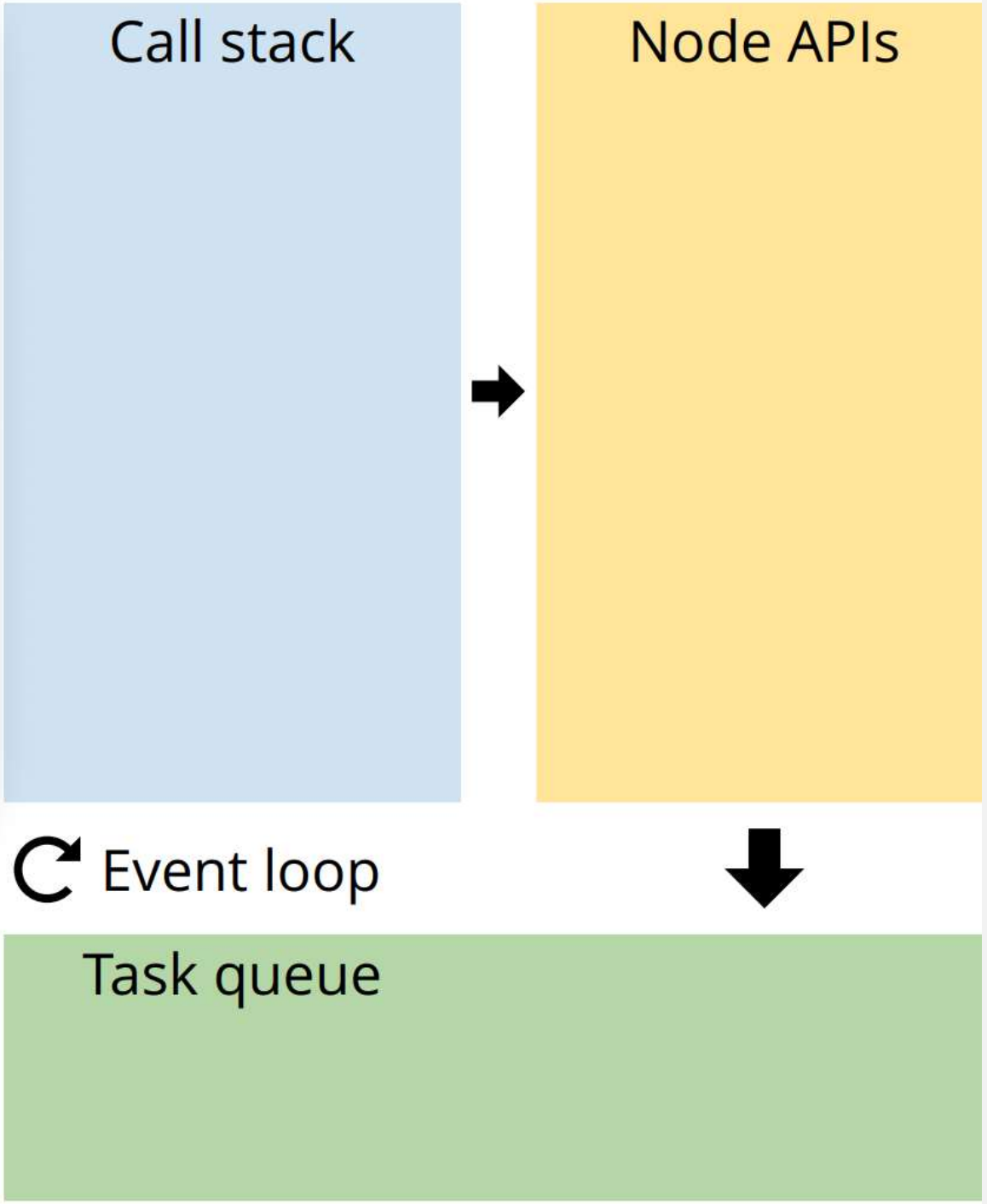
```
1 console.log('starting app');
2
3 setTimeout(function callback() {
4
5   console.log('my function callback');
6
7 }, 5000);
8
9  console.log('end app');
```



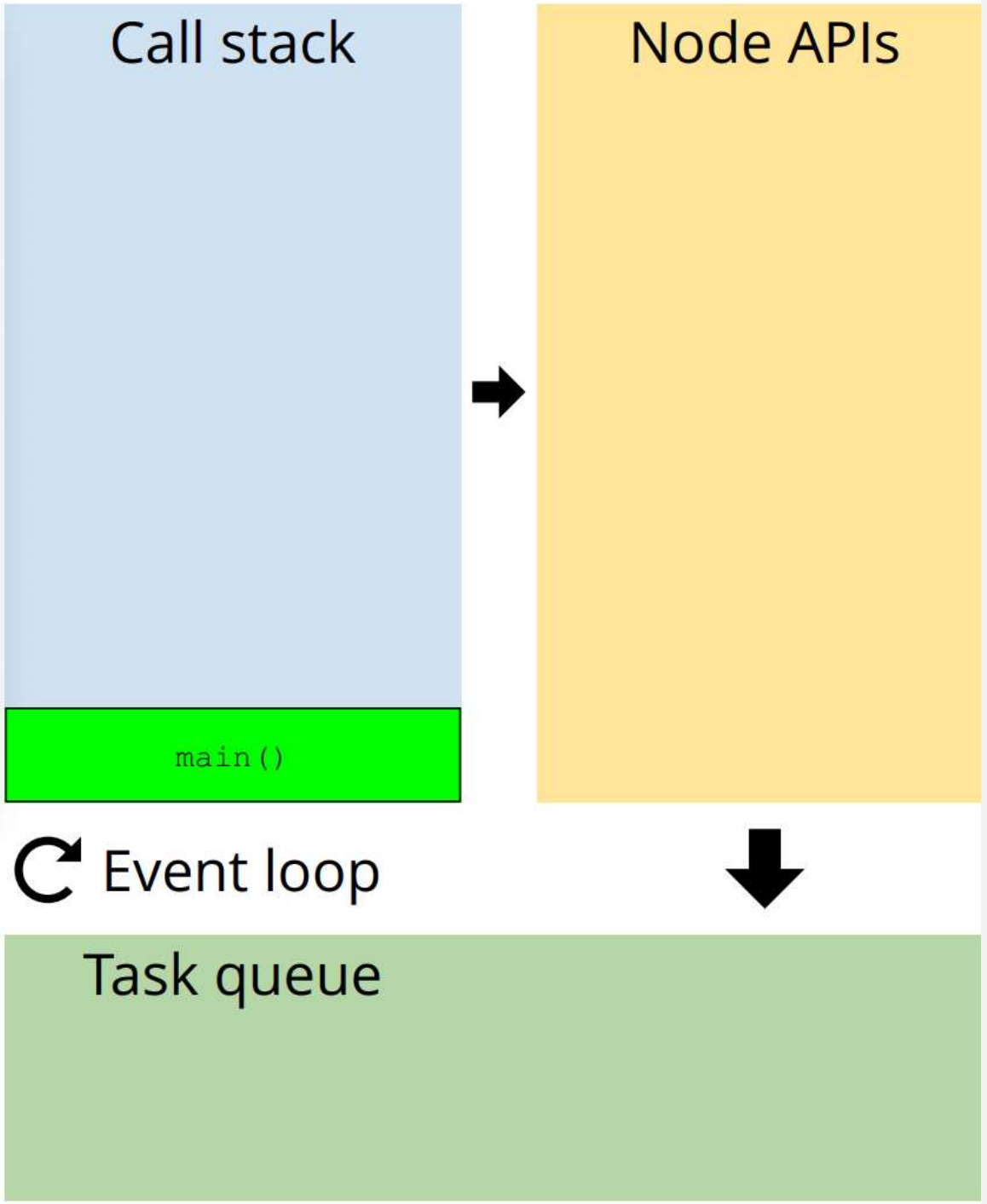
Callback

```
console.log('starting app’);

setTimeout(function callback1() {
    console.log('my function callback1')
}, 5000);

setTimeout(function callback2() {
    console.log('my function callback2')
}, 0);

console.log('end app')
```

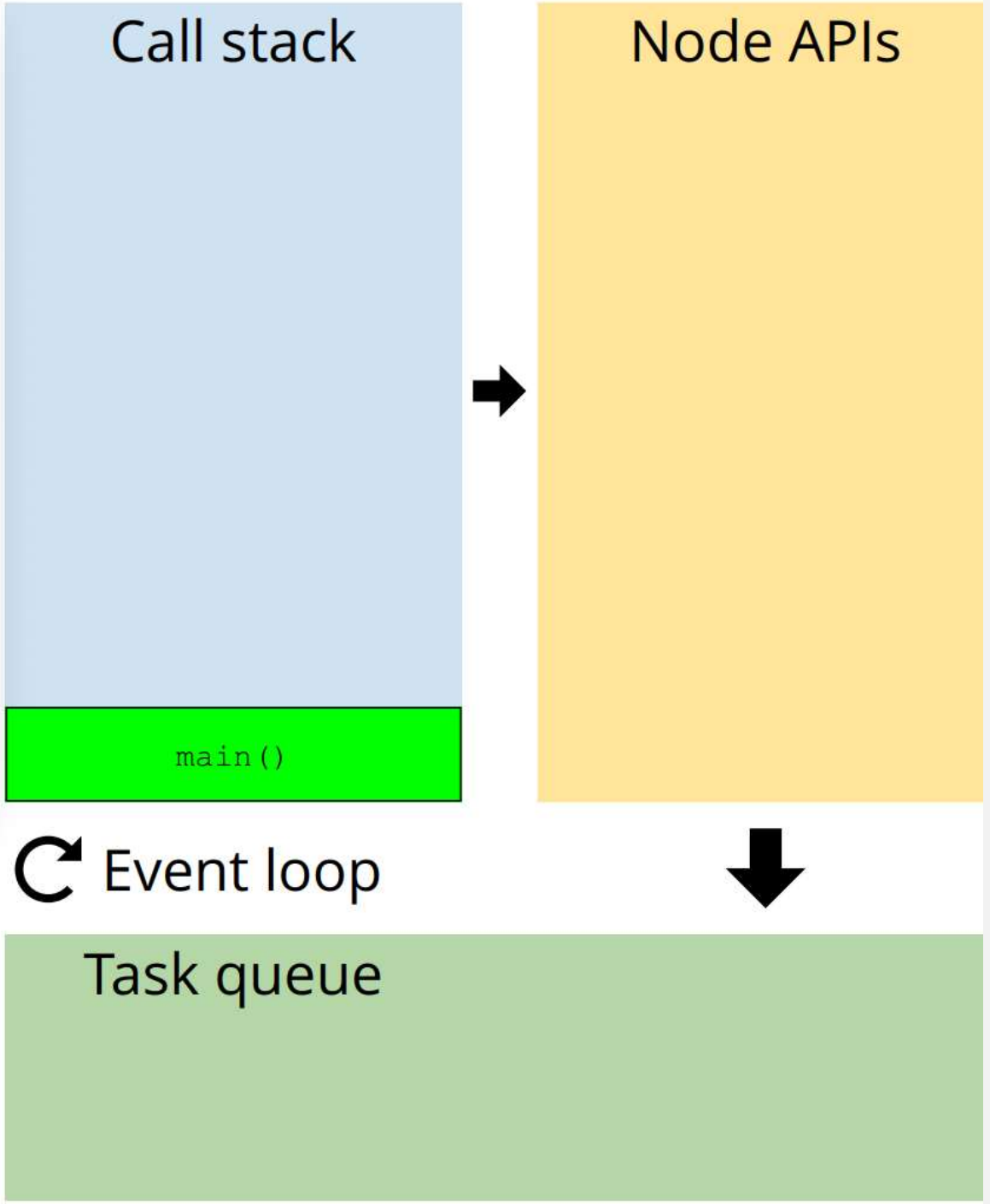
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



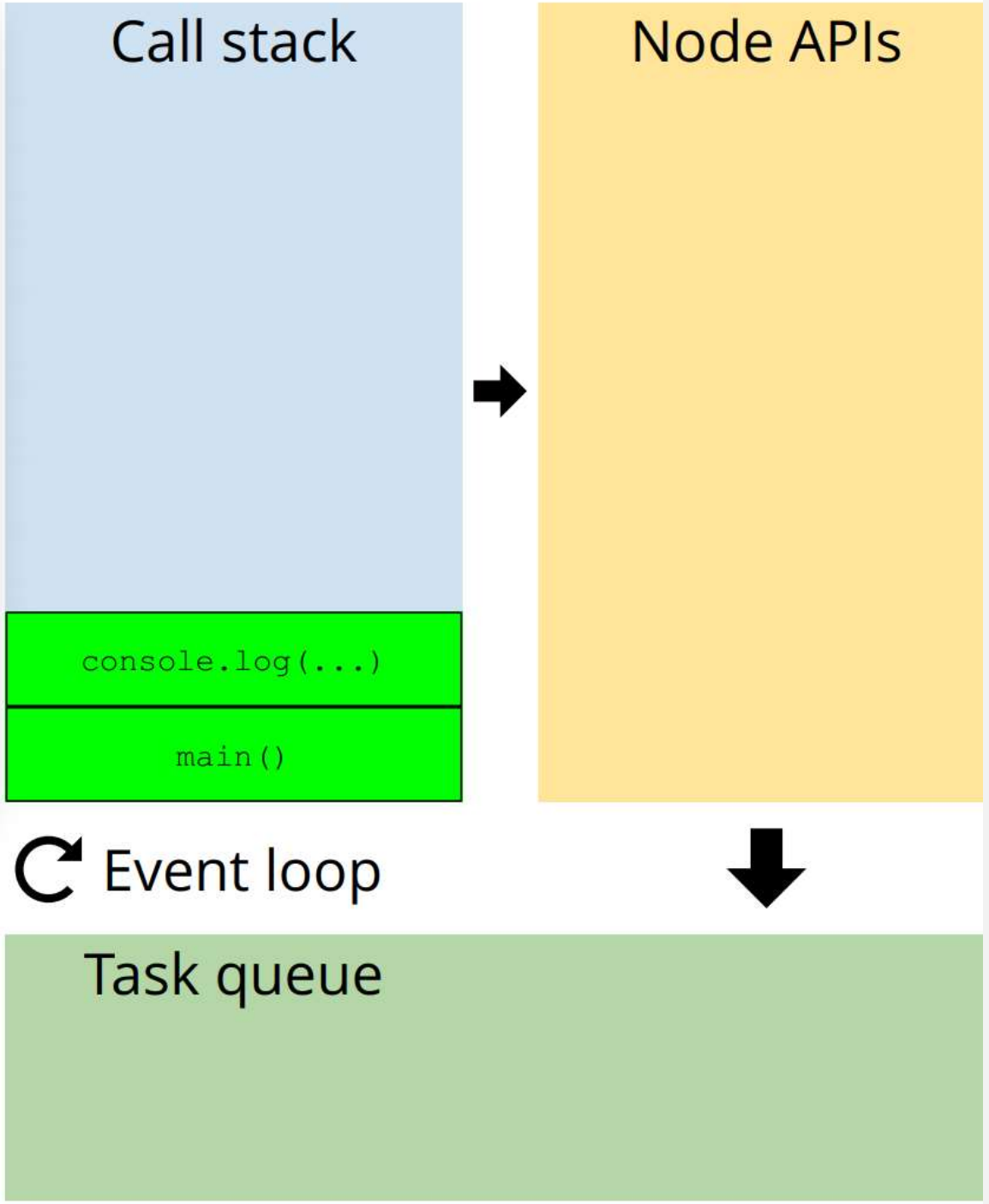
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



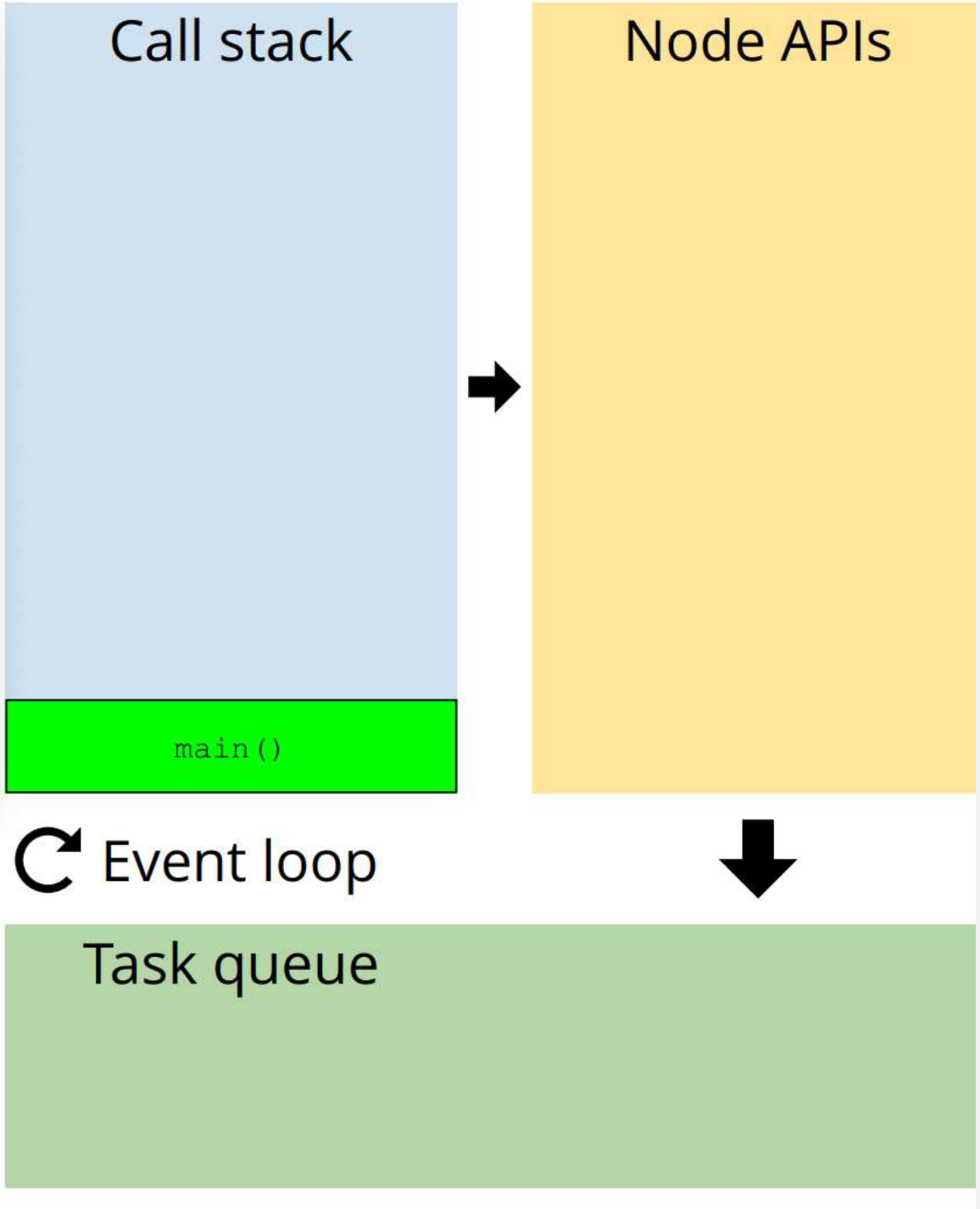
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



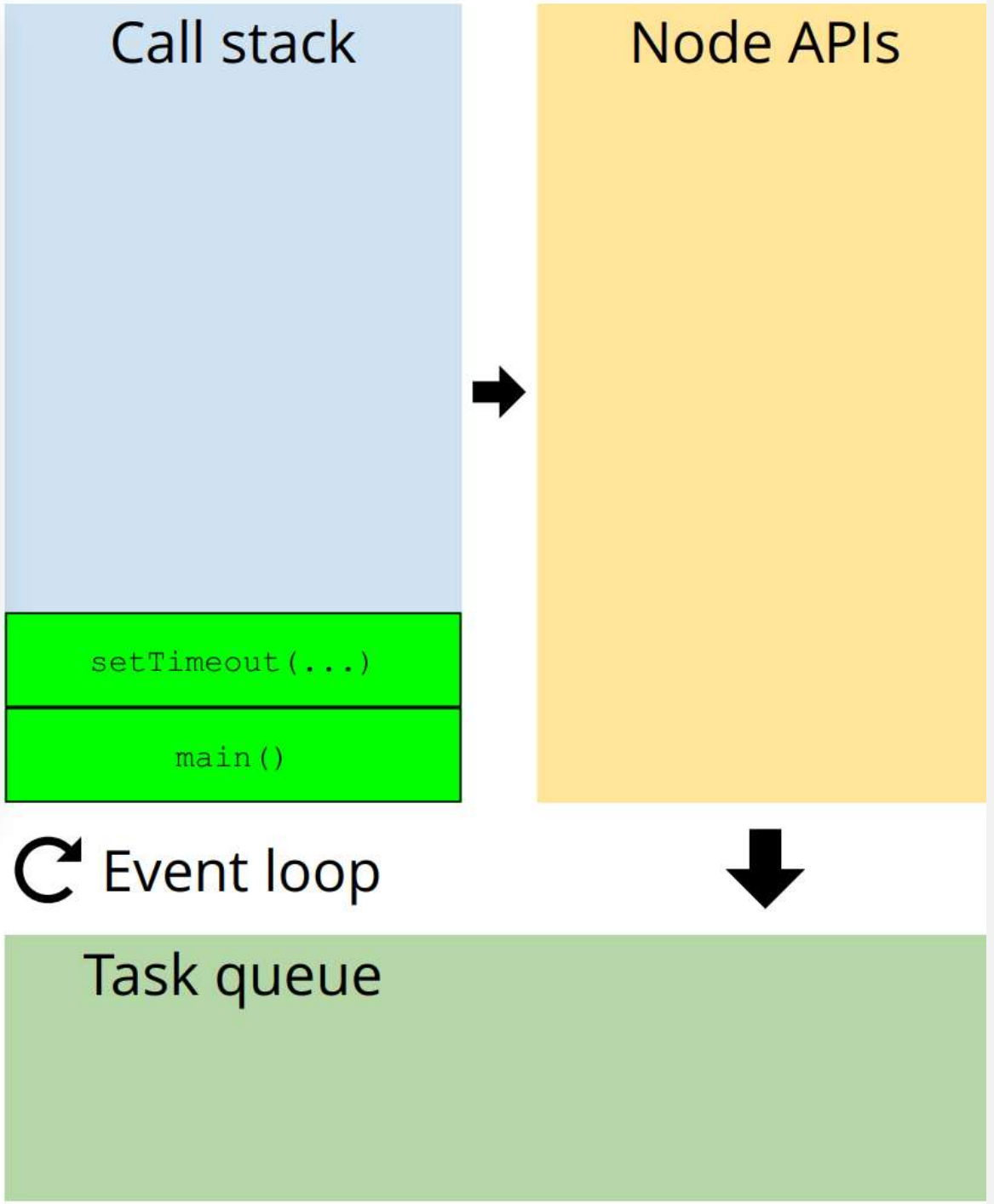
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



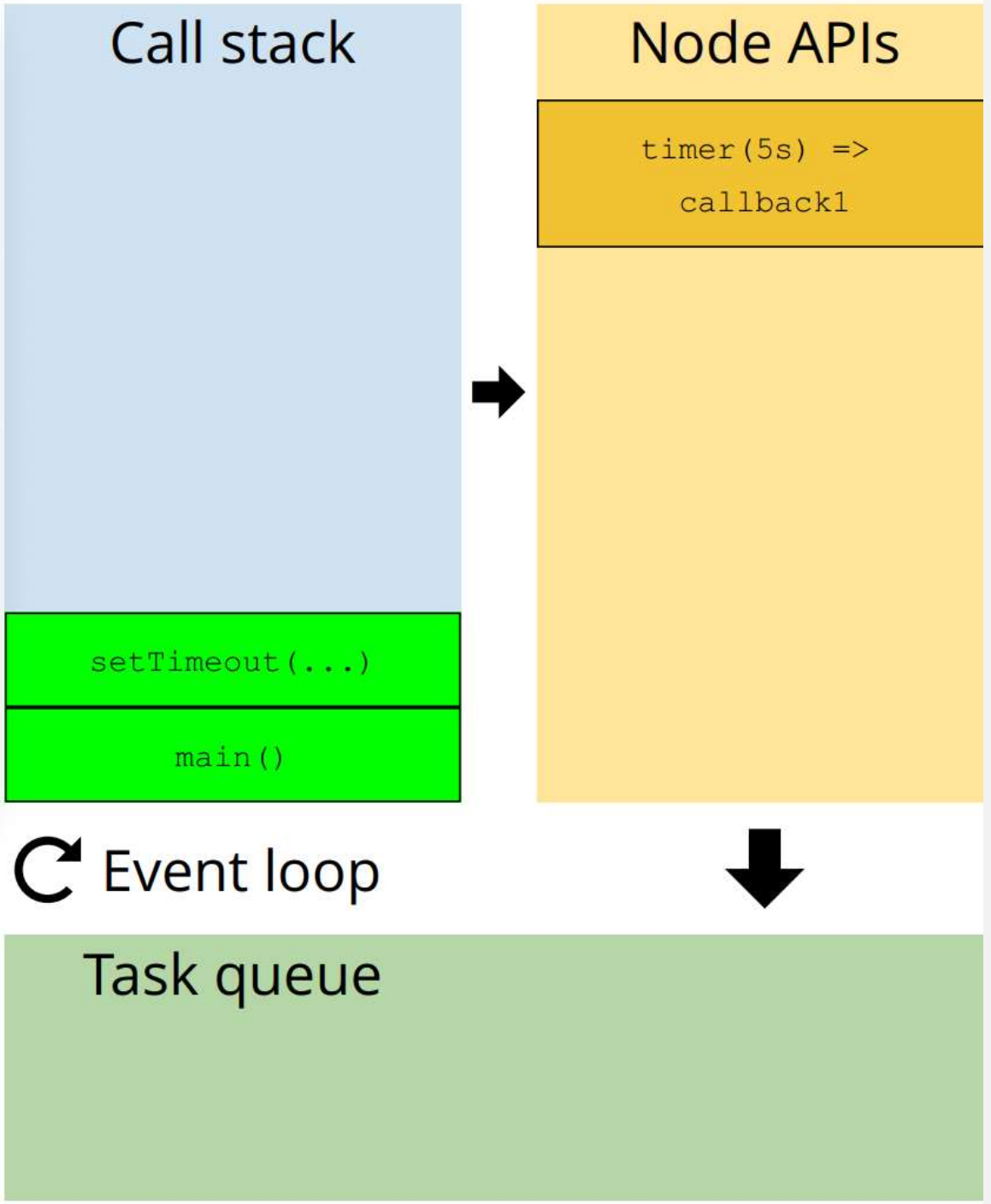
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



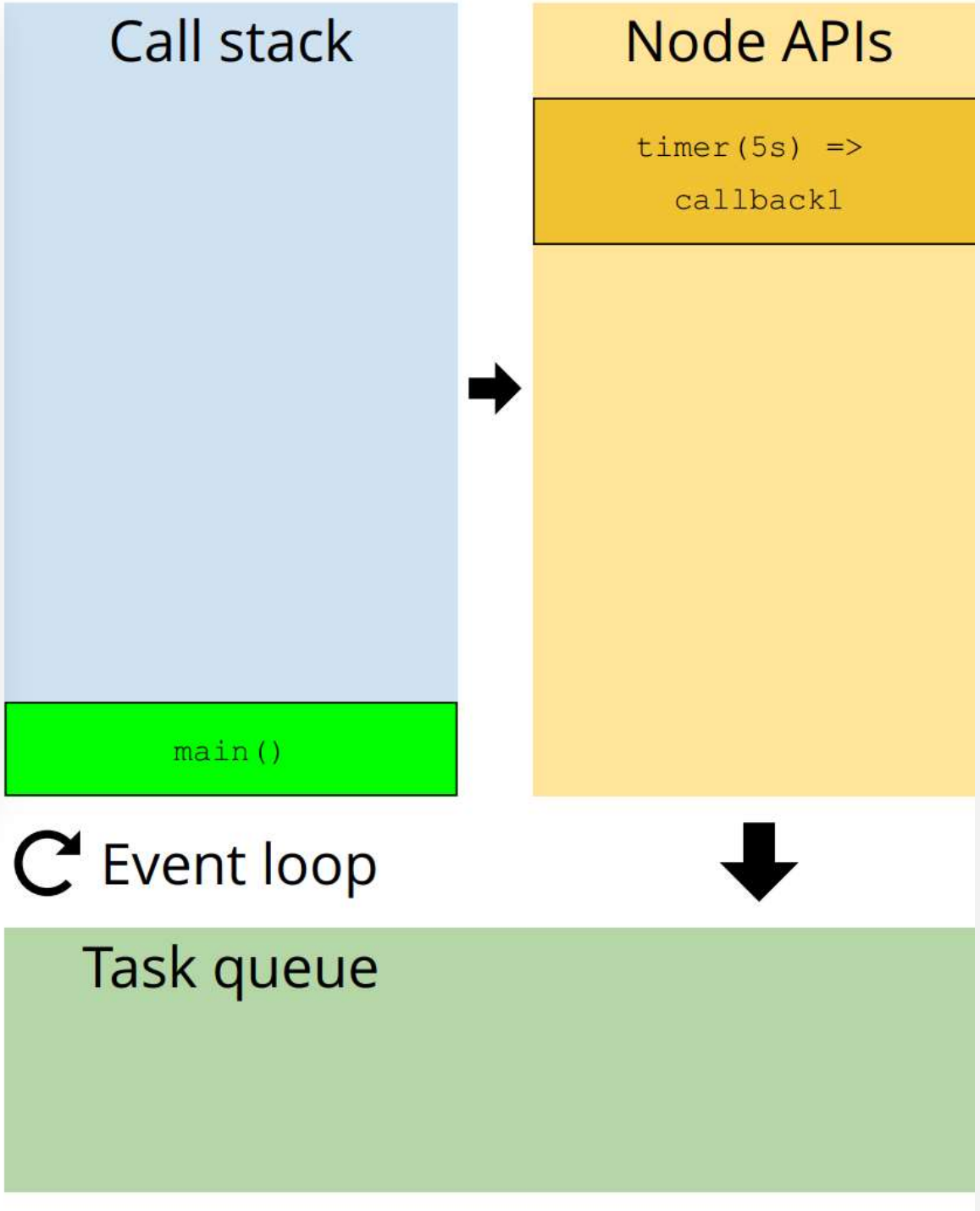
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



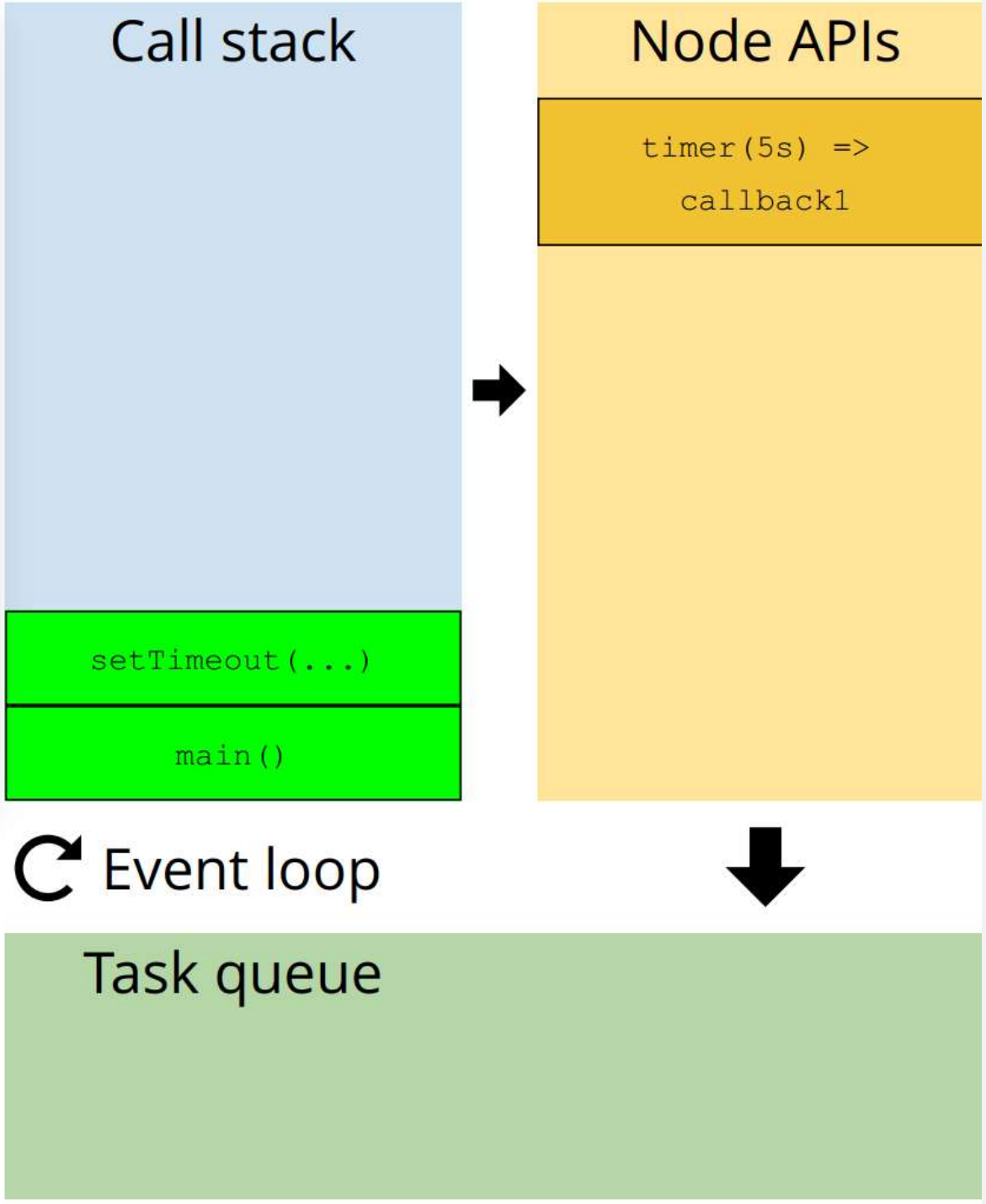
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



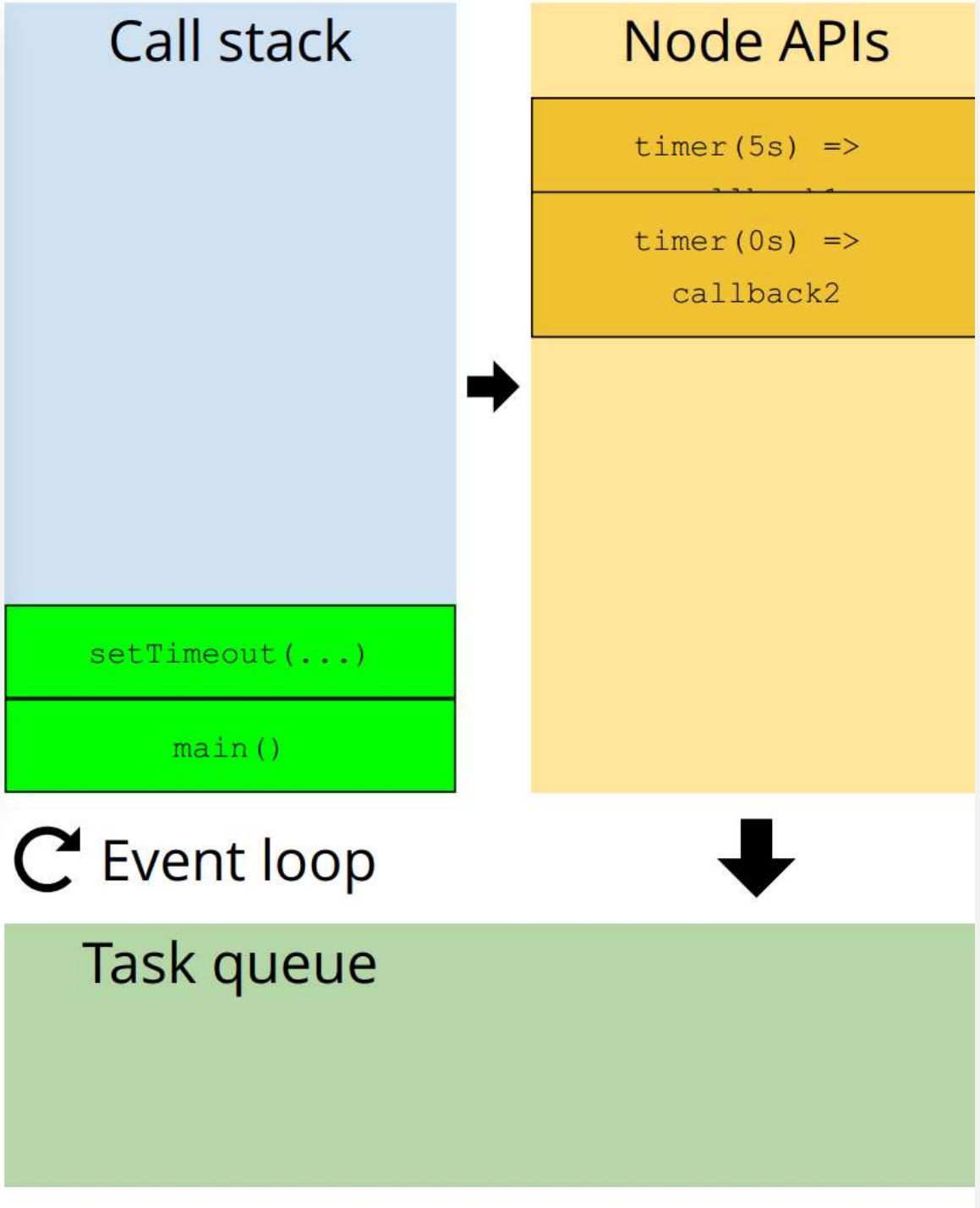
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



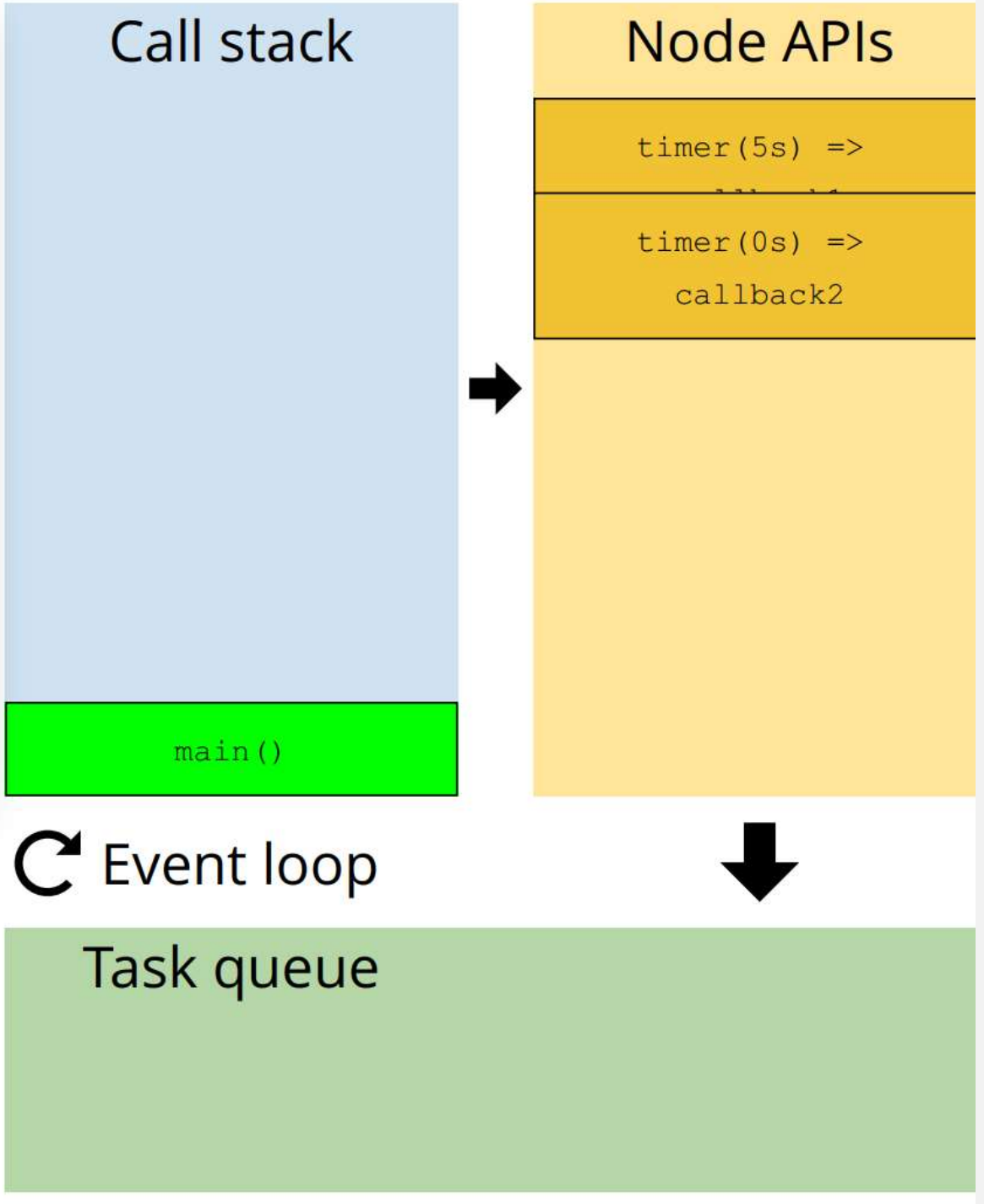
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



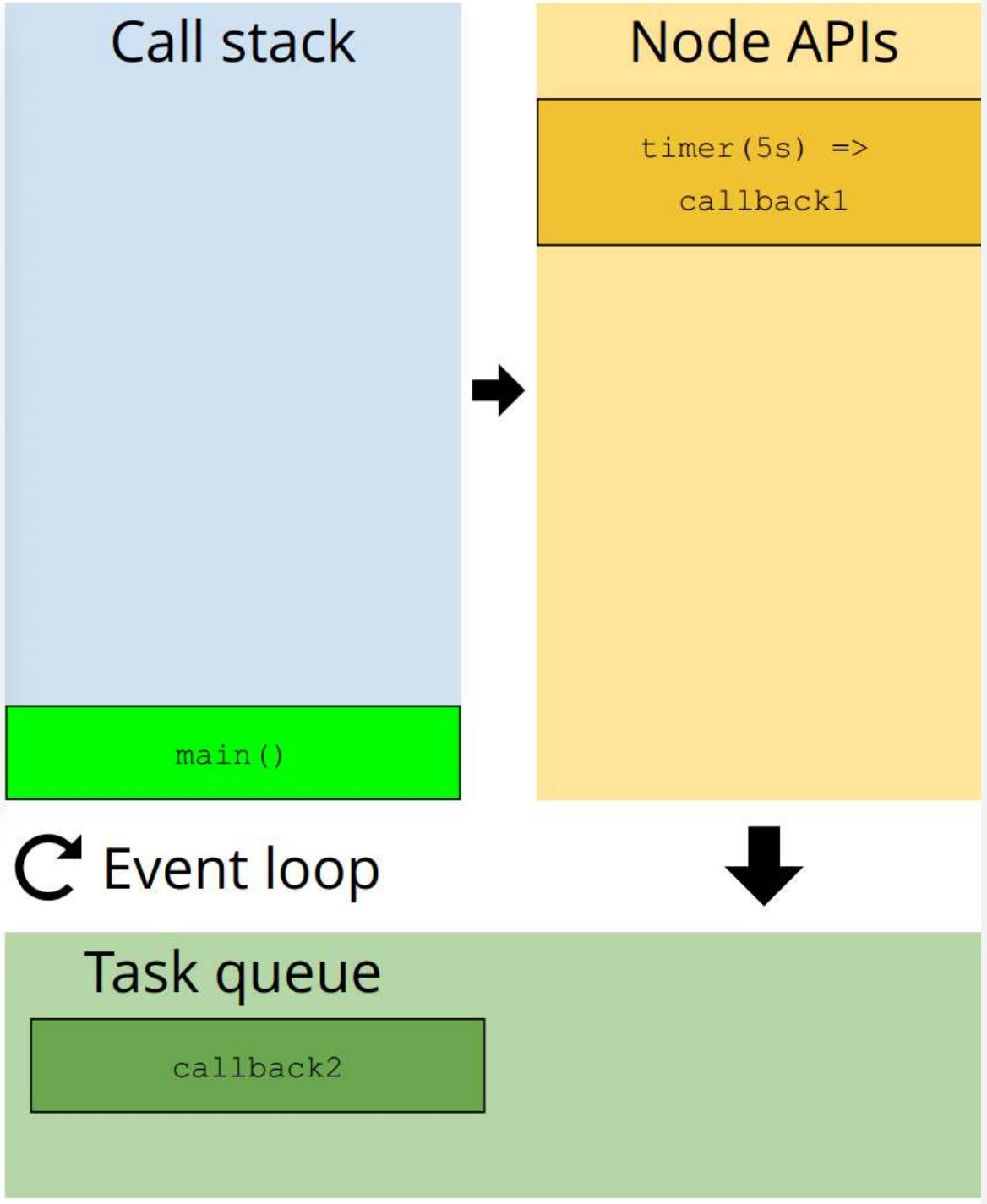
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



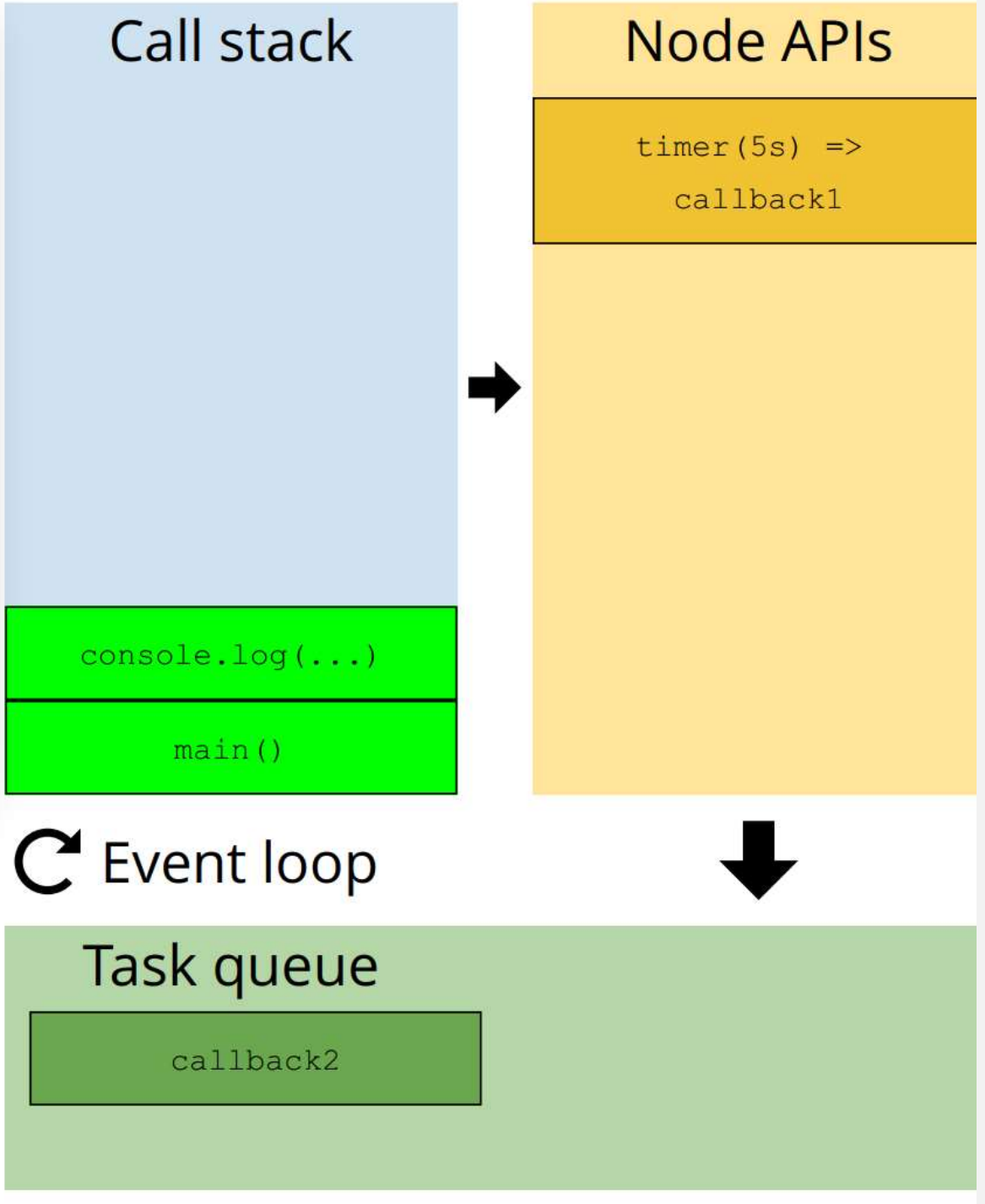
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



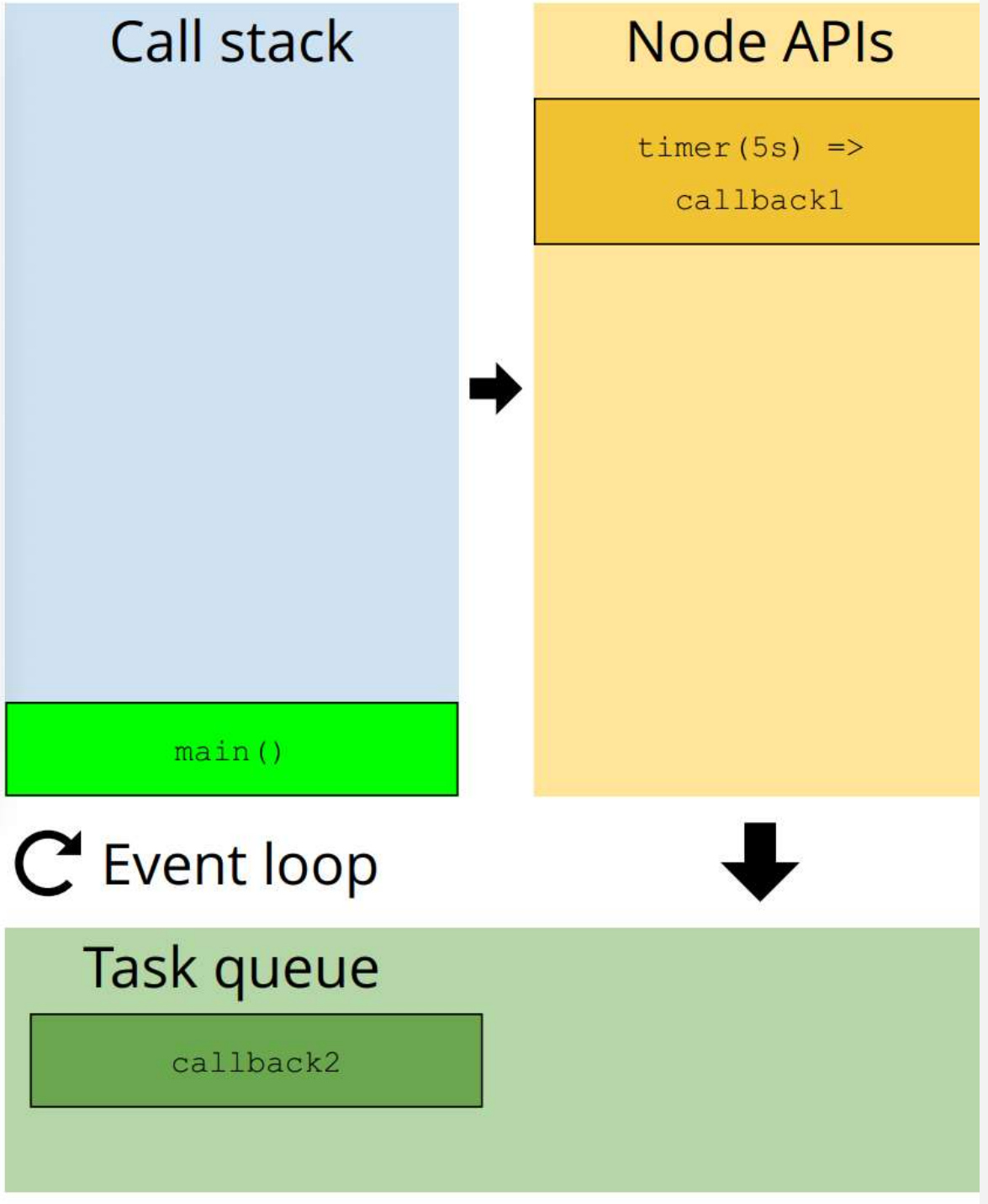
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



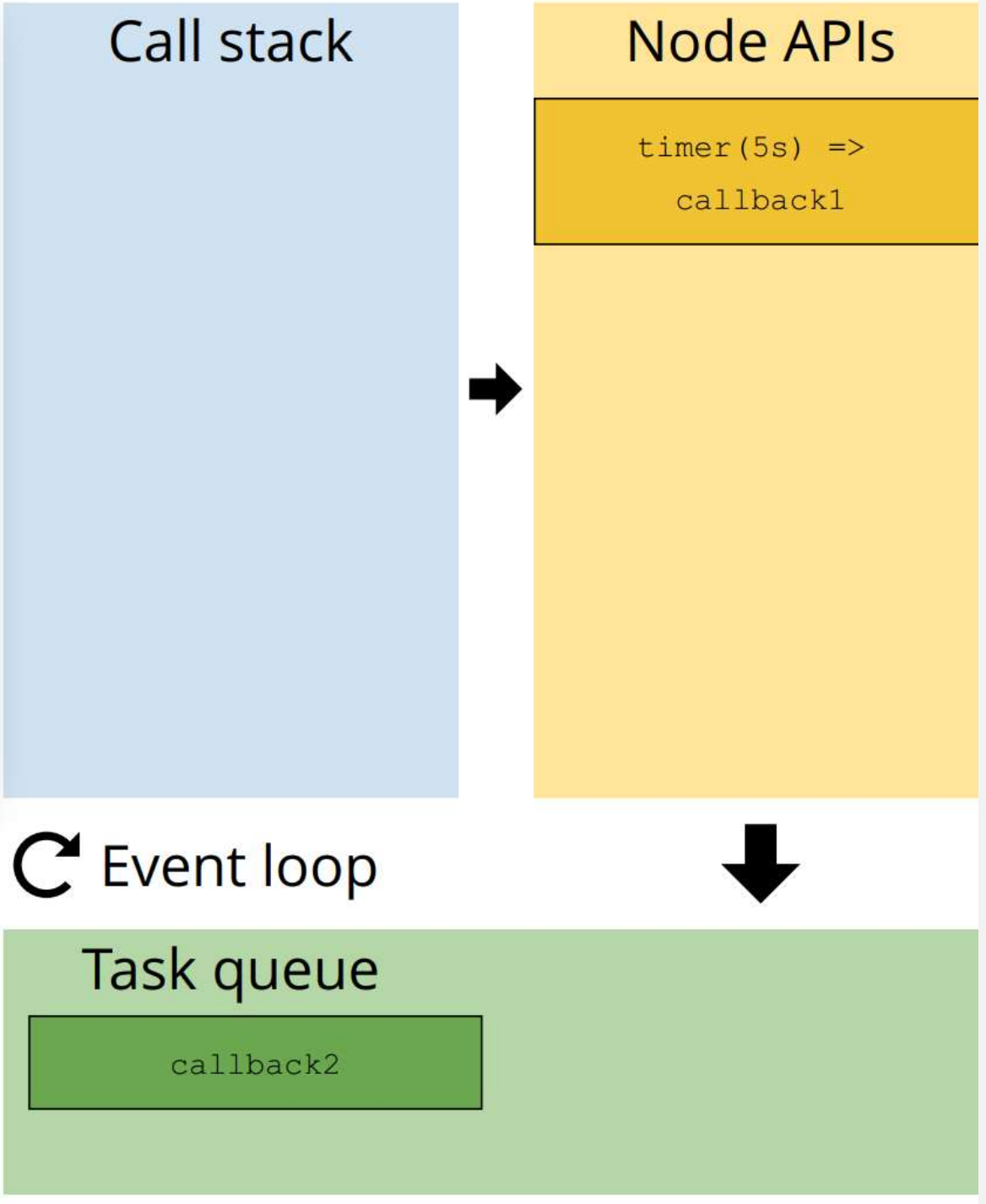
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



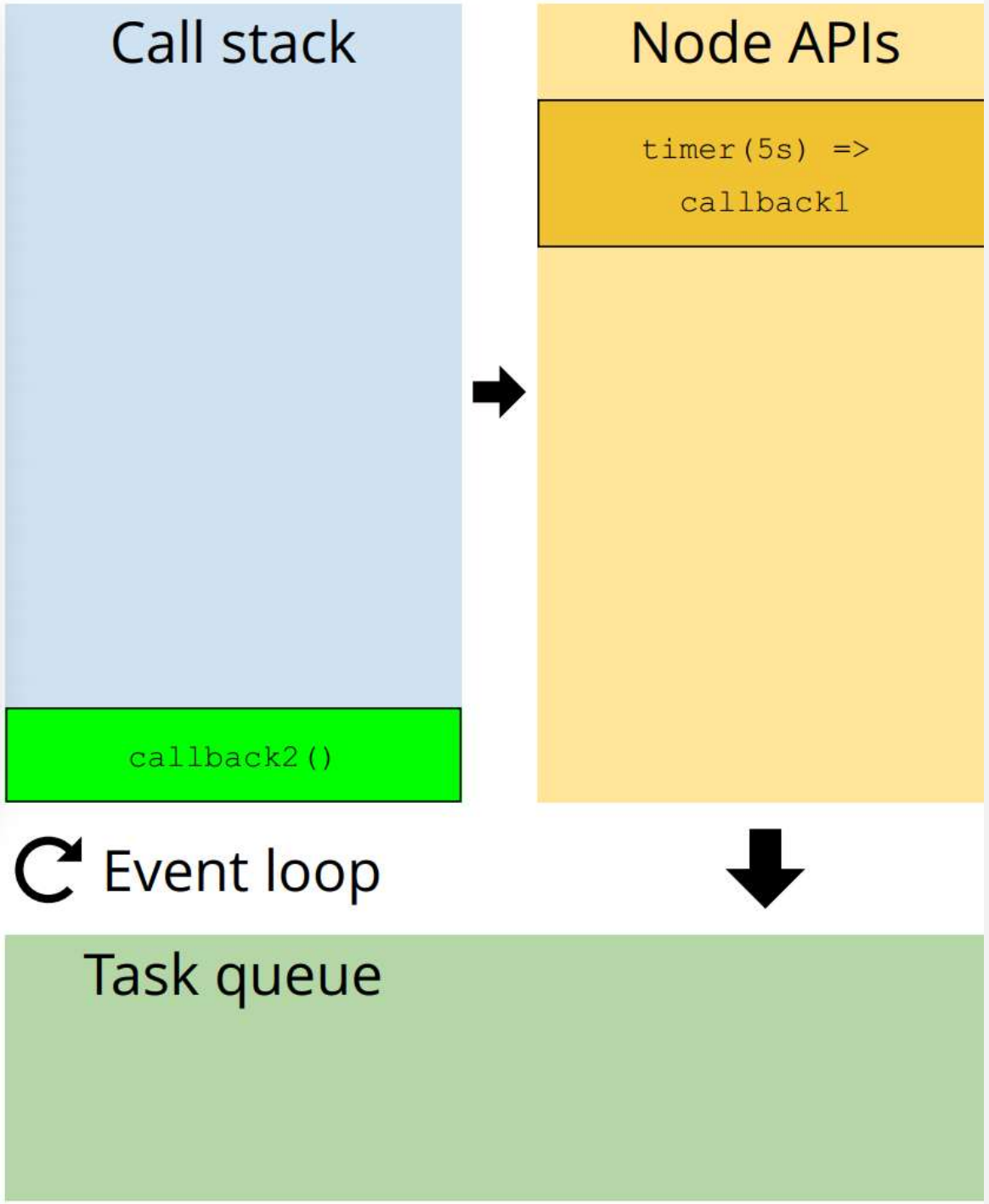
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



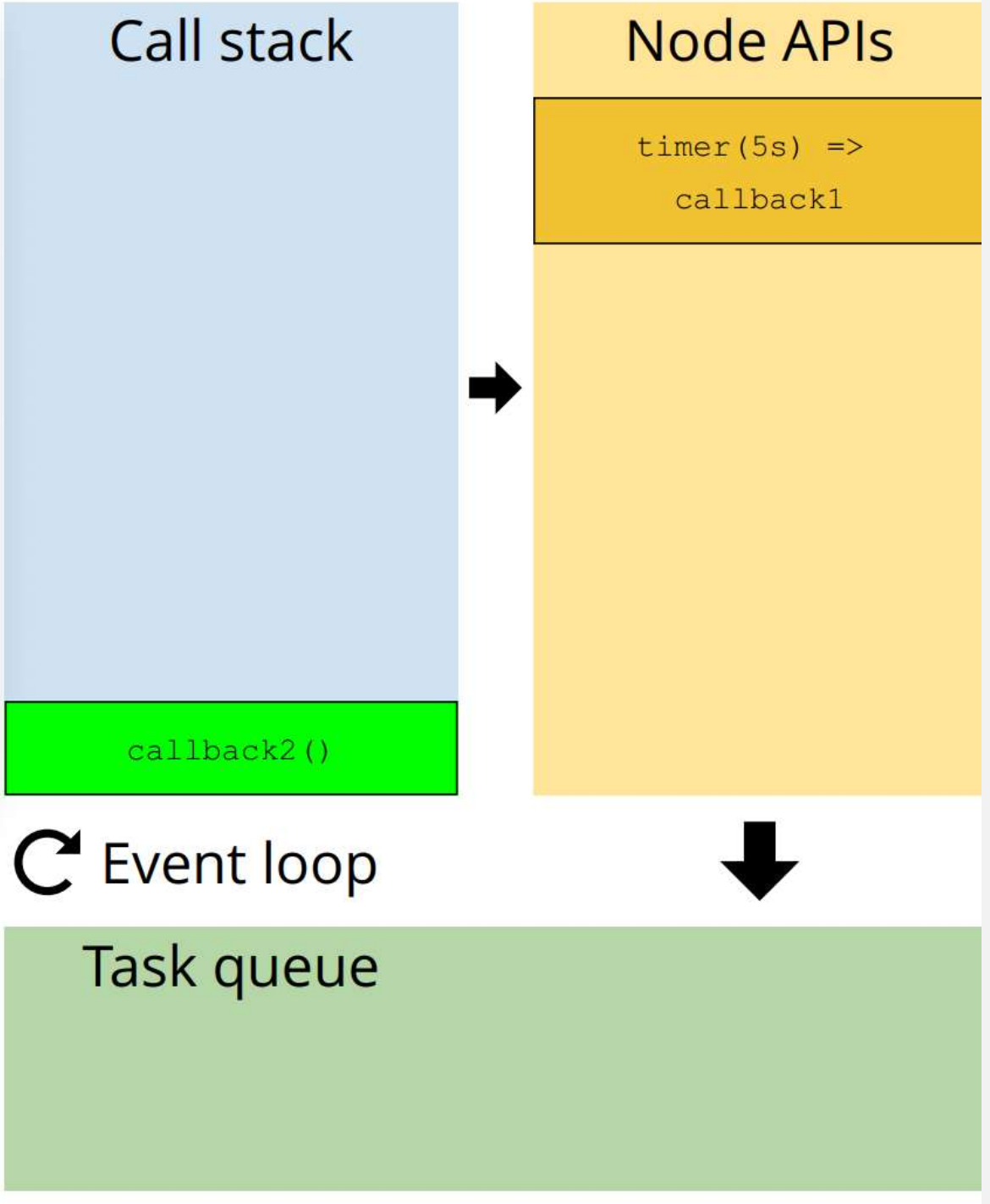
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



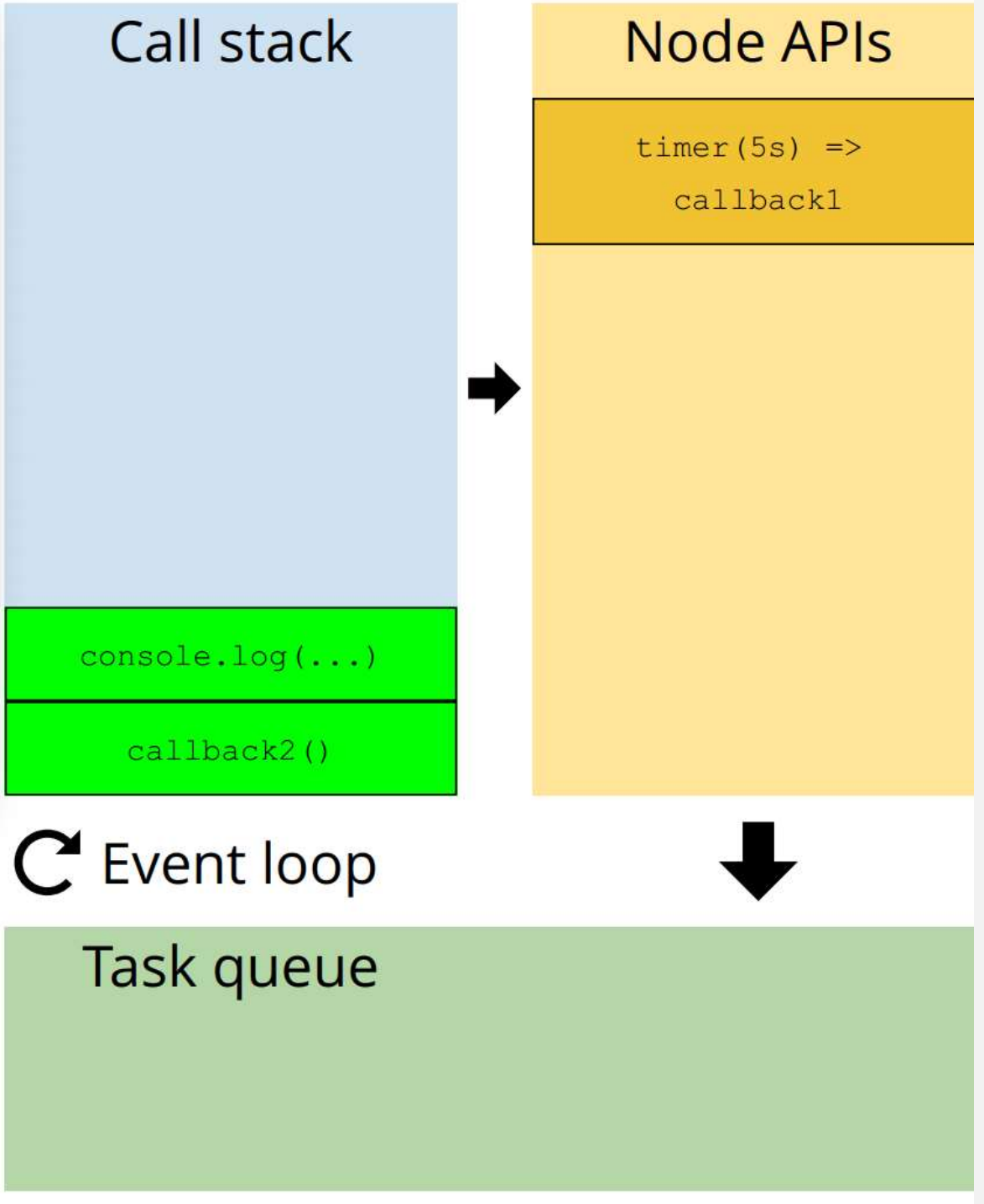
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



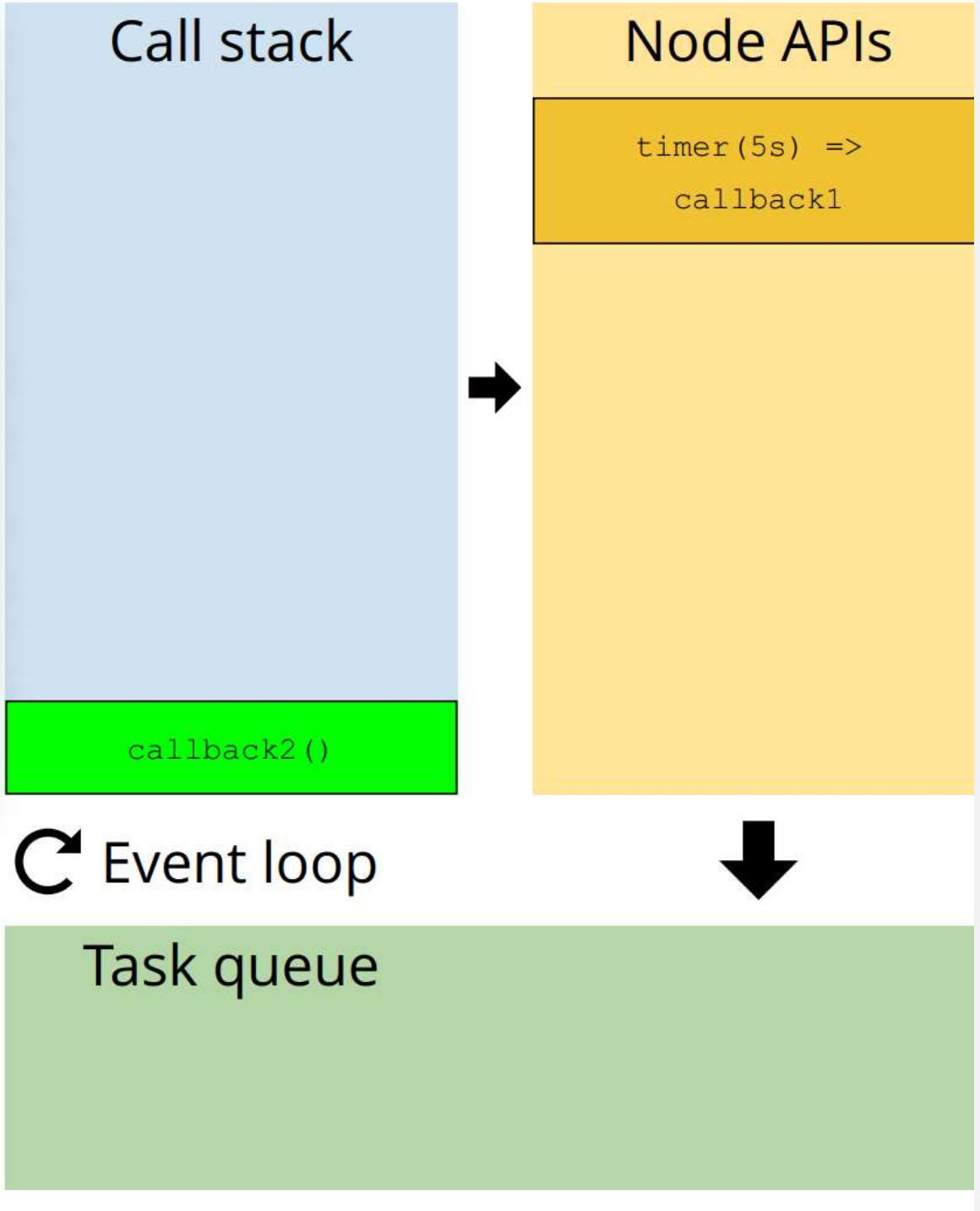
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



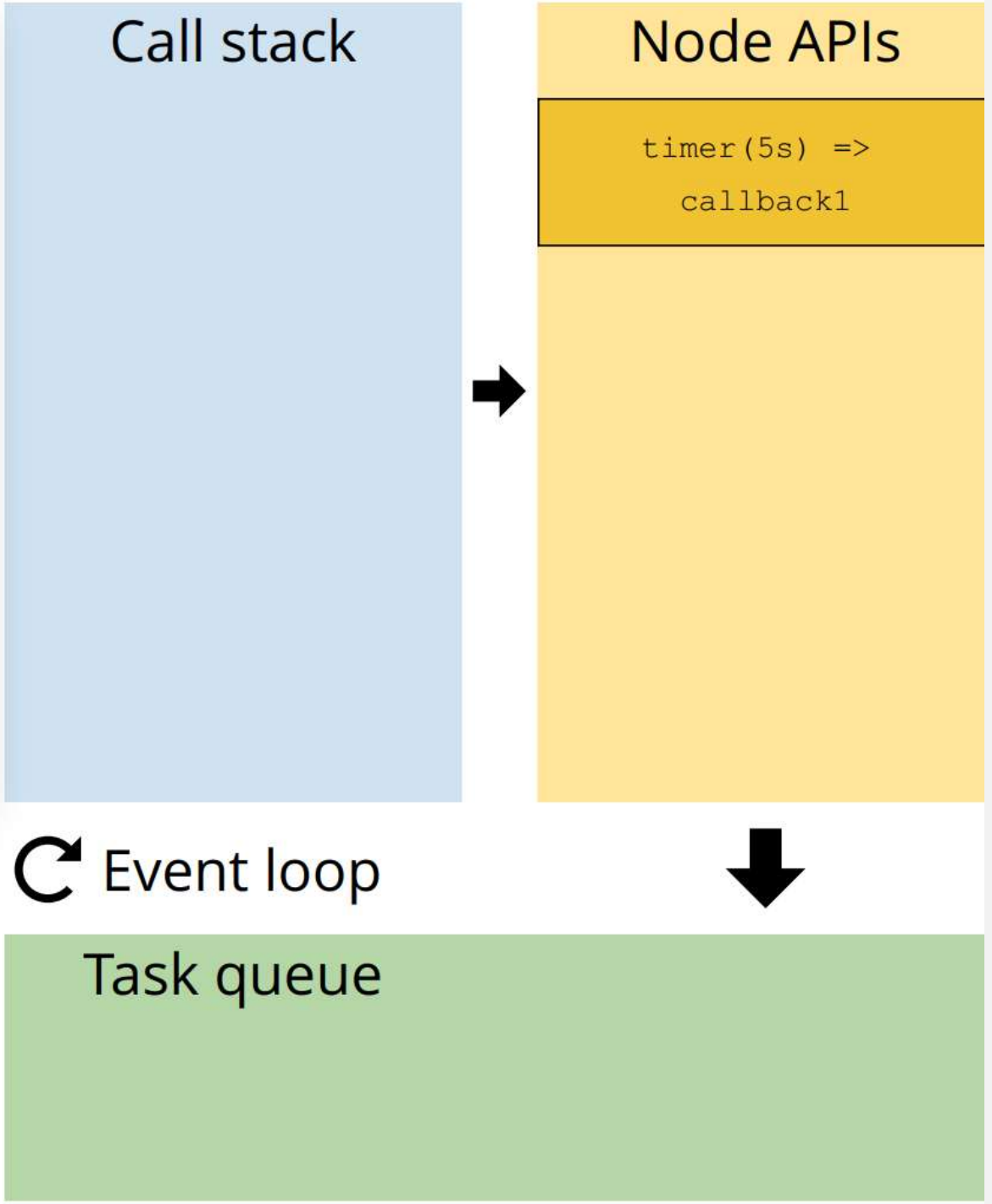
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



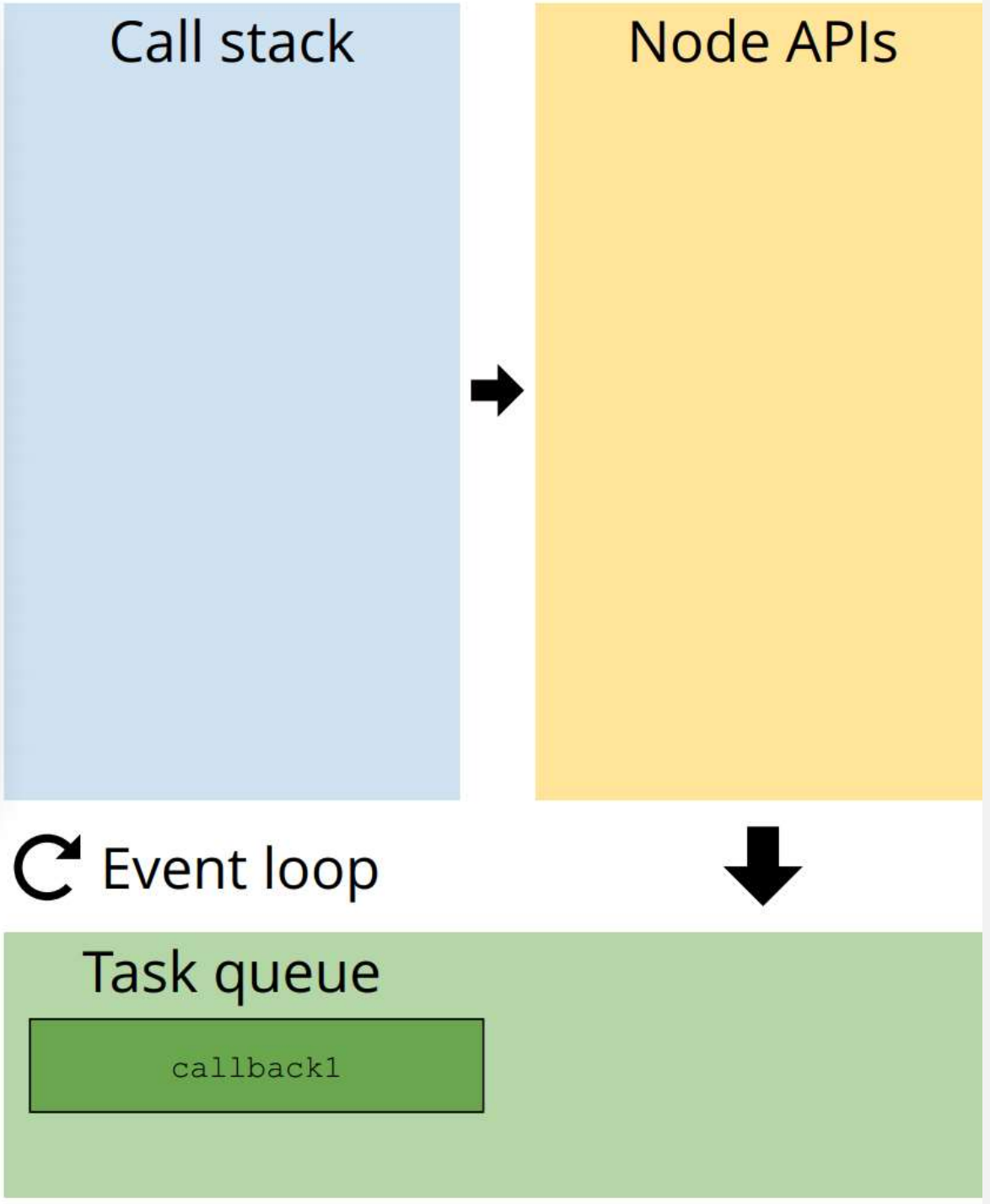
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



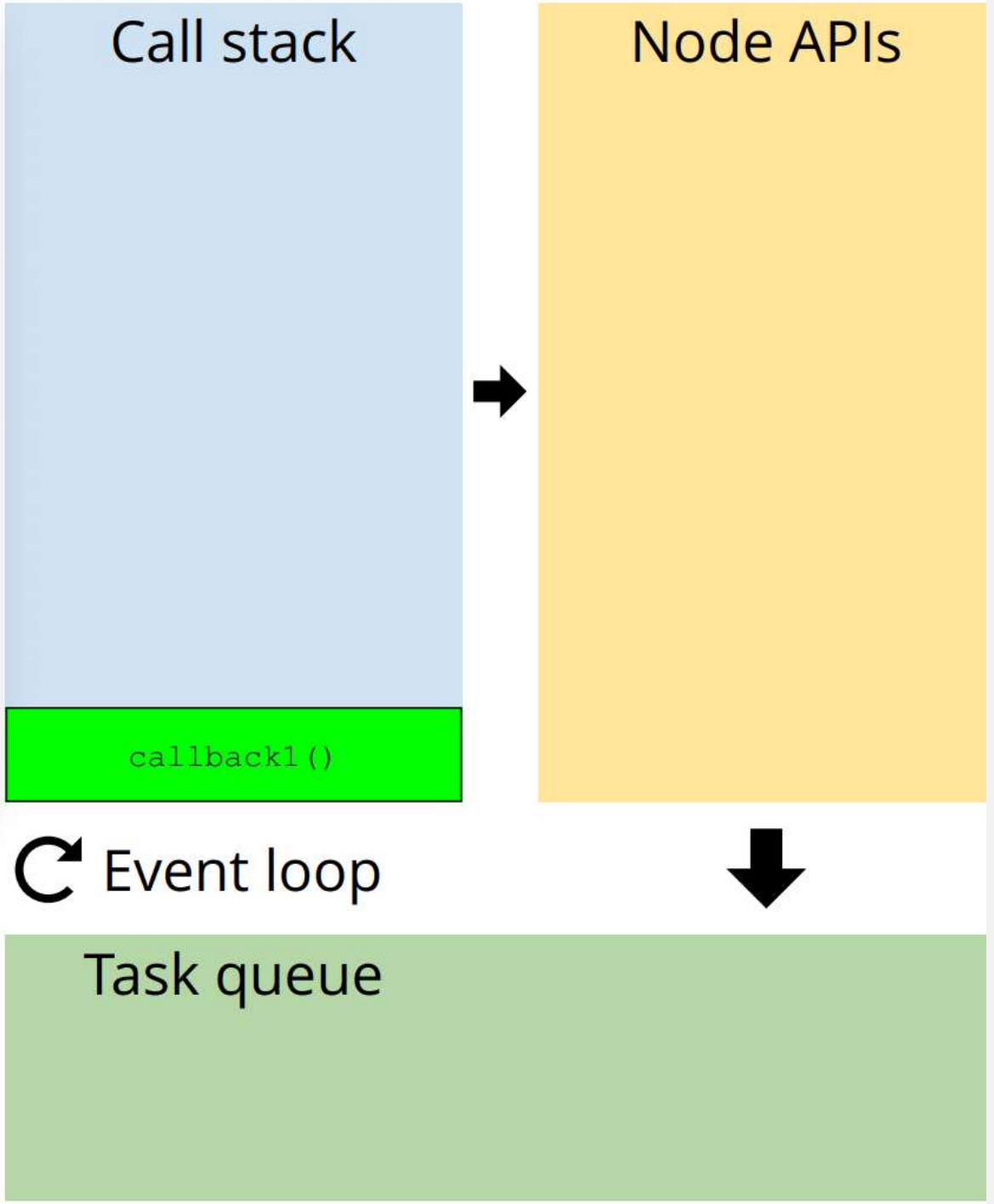
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



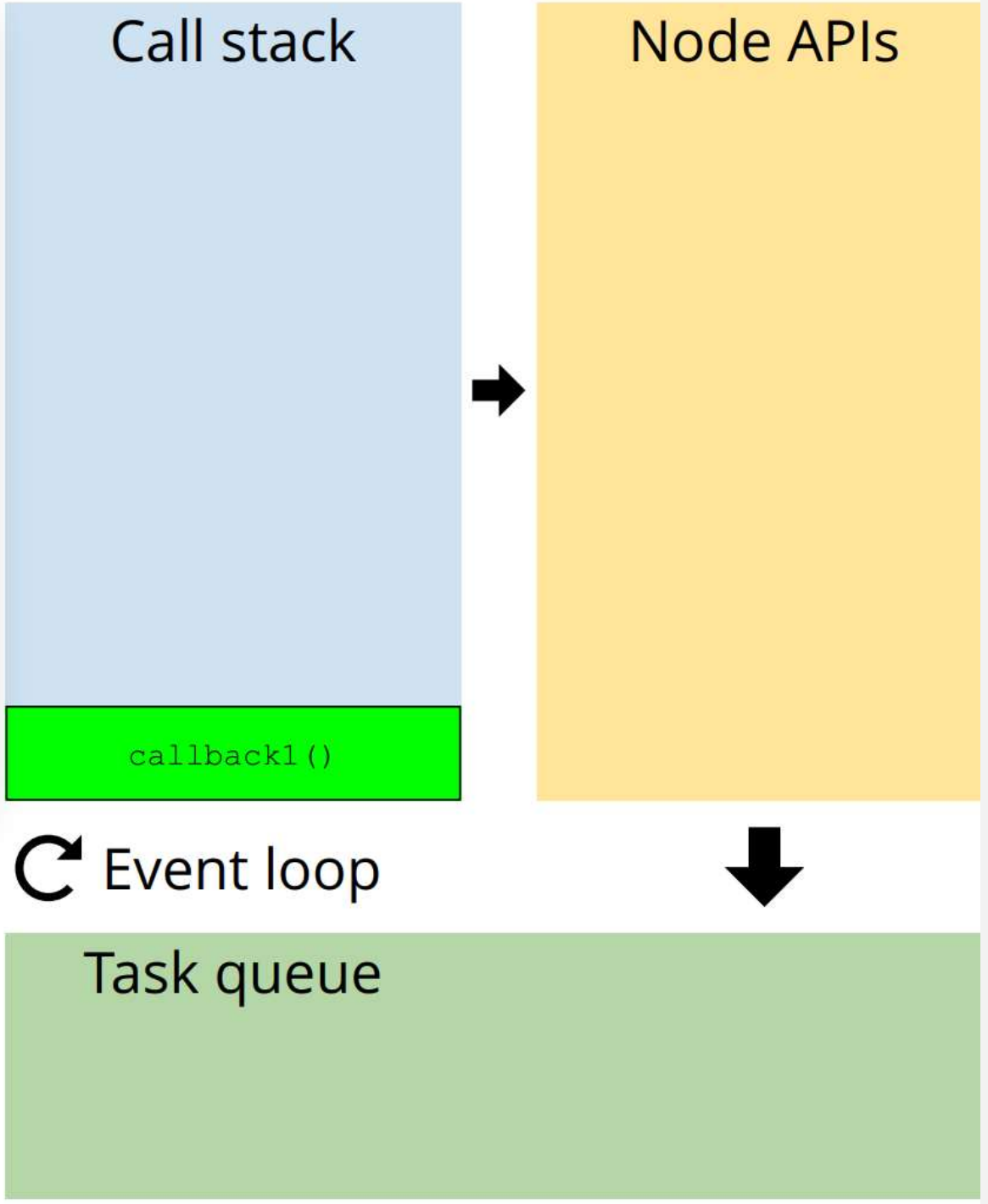
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



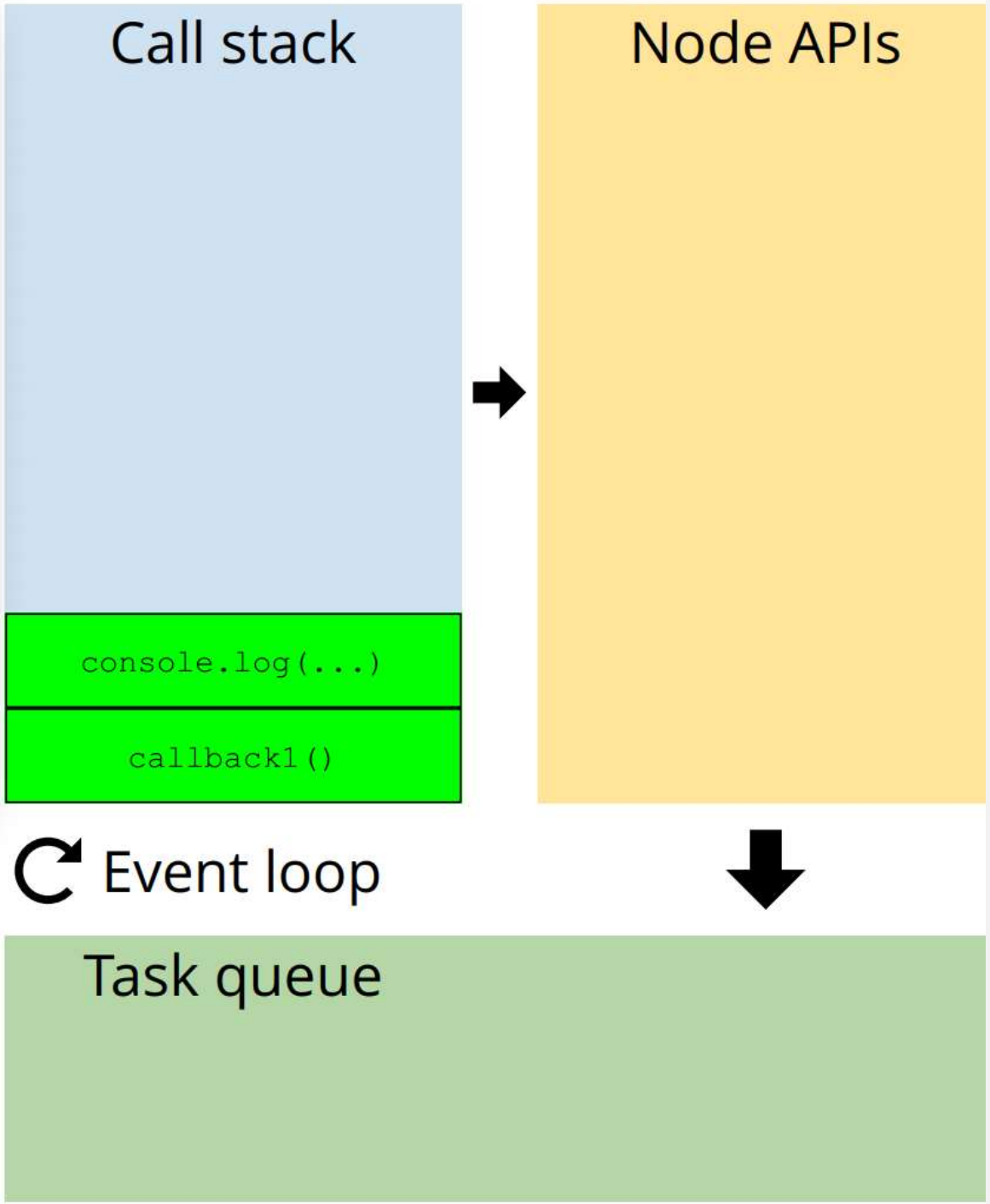
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



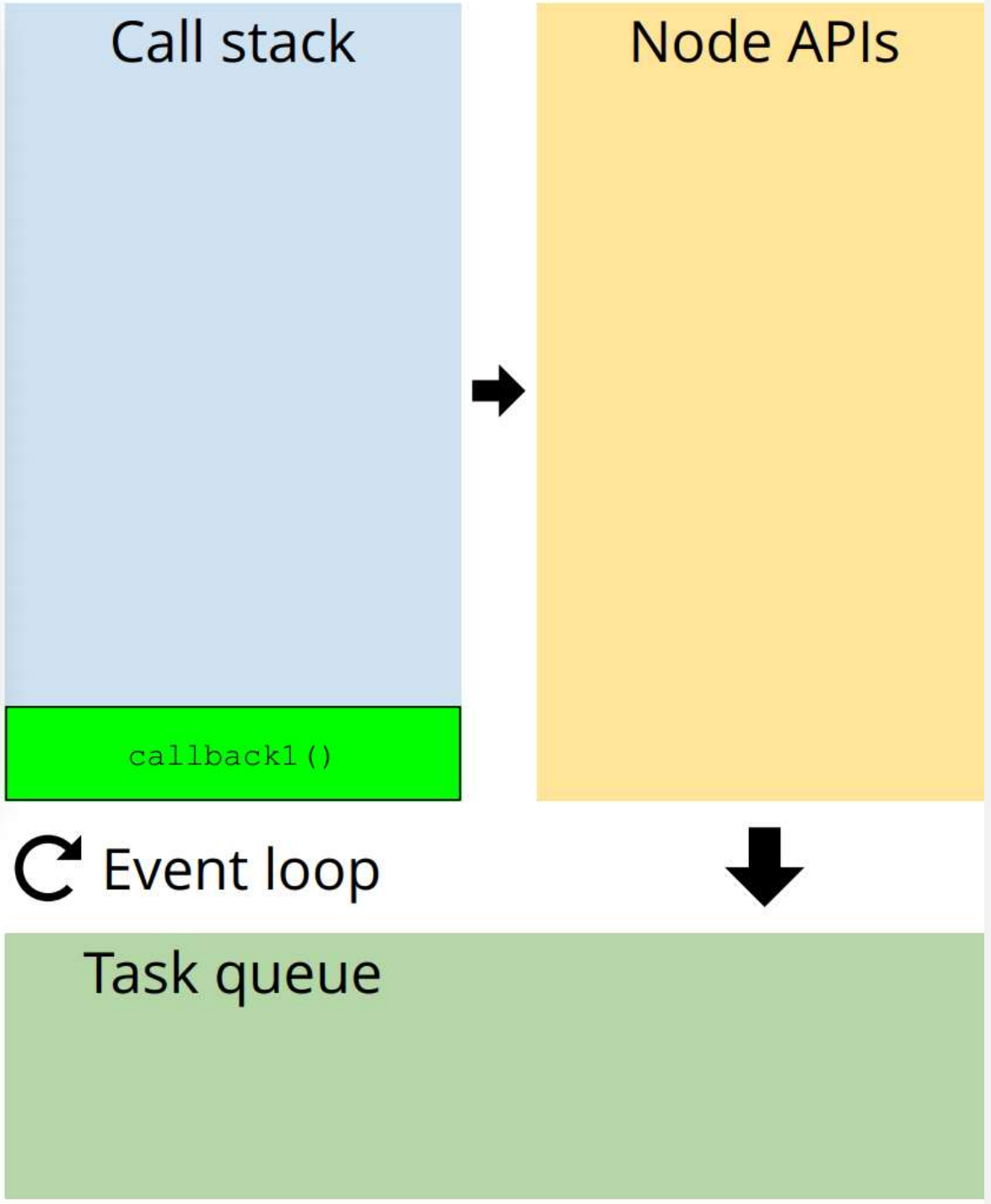
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



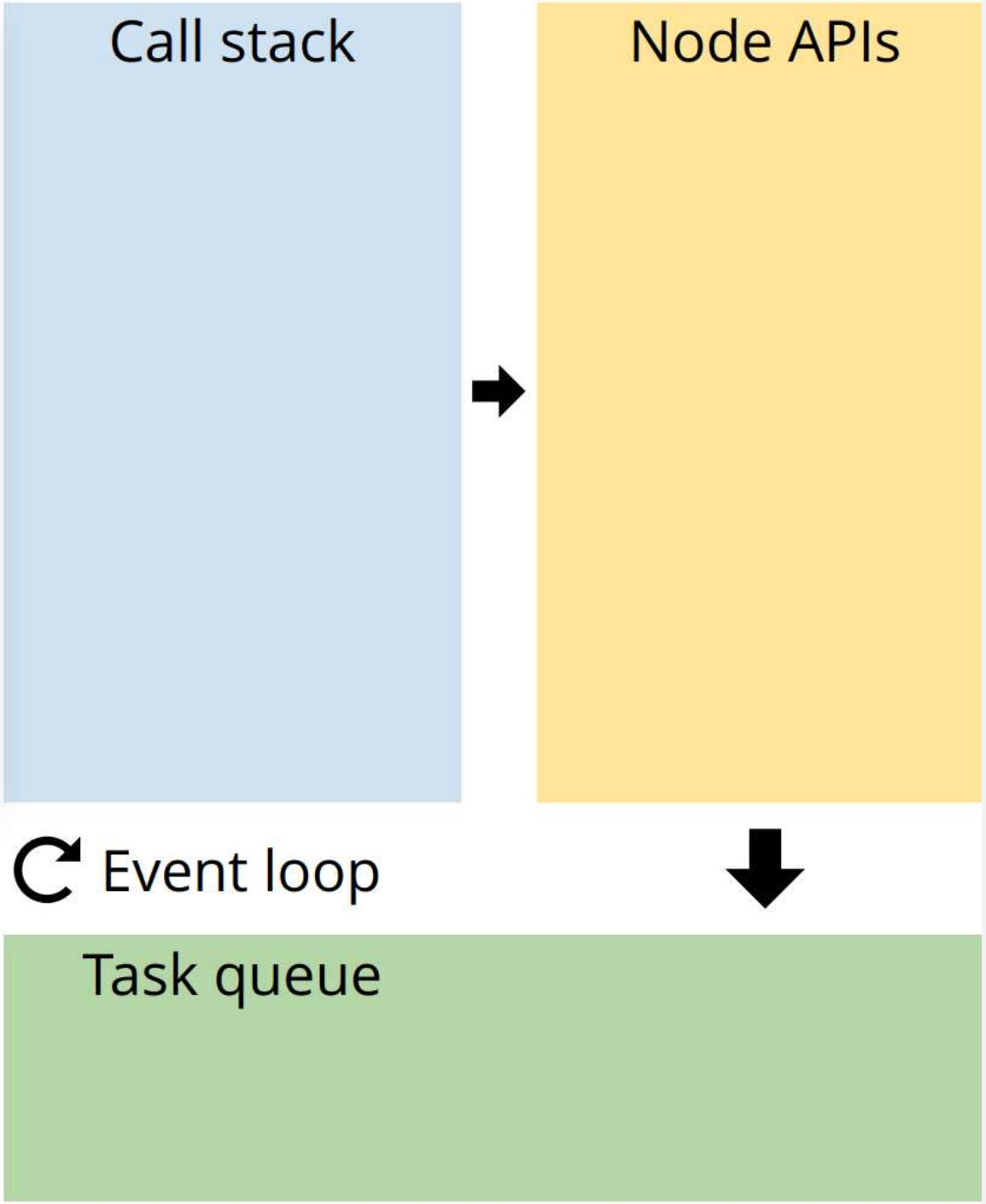
```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```



```
1  console.log('starting app');
2
3  setTimeout(function callback1() {
4
5    console.log('my function callback1')
6
7  }, 5000);
8
9  setTimeout(function callback2() {
10
11   console.log('my function callback2')
12
13 }, 0);
14
15  console.log('end app');
```

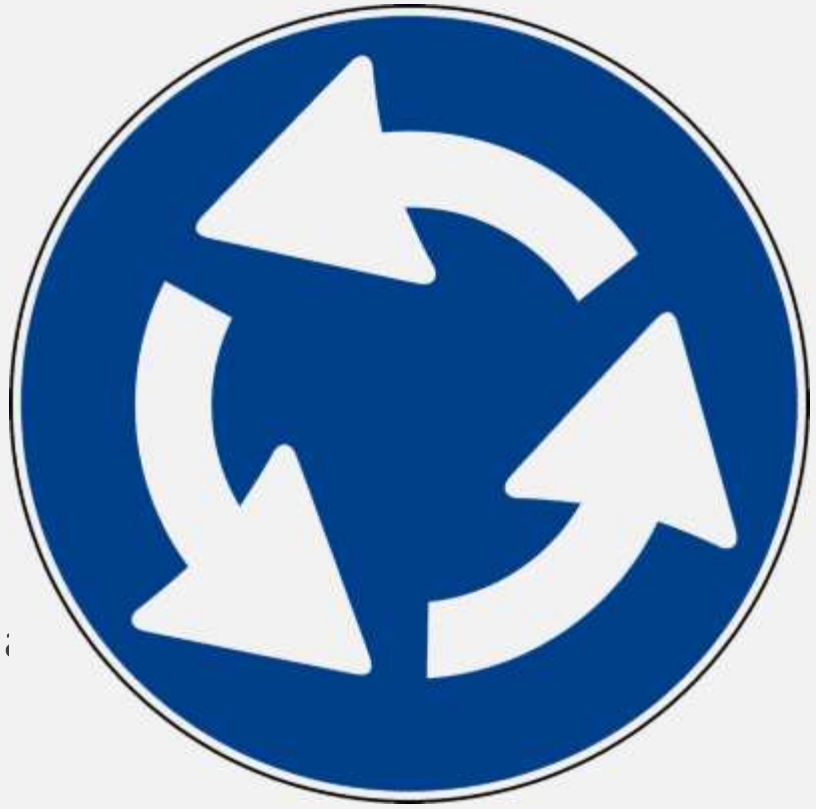


Multiple callbacks

```
console.log('starting app’);

setTimeout(function callback1() {
    console.log('my function callback1')
}, 0);

setTimeout(function callback2() {
    console.log('my function callback2')
}, 5000);

console.log('end app’);
```

Event loop

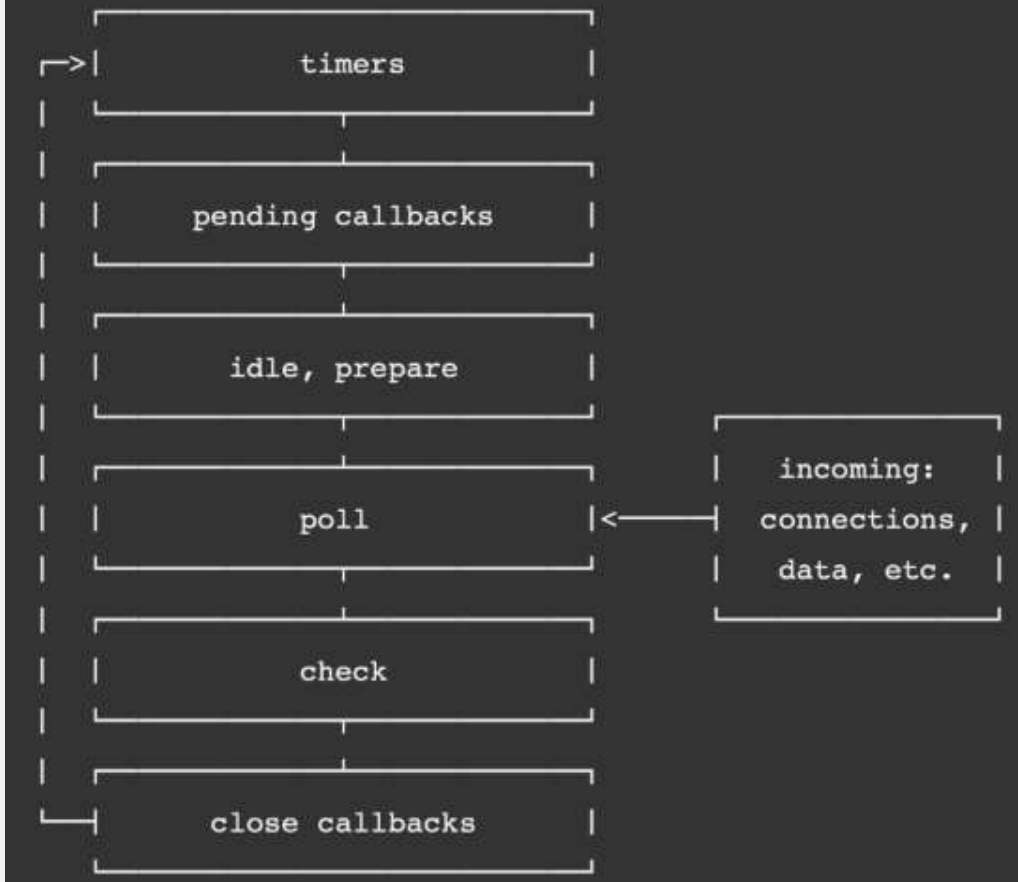
- endless loop (end with program)
- initialized on Node.js start
- executes tasks only when the call stack is empty
- waits for tasks, executes them and then sleeps until it receives more tasks
- allows us to use callbacks and promises
- executes the tasks starting from the oldest first (oldest that ready to process)



Event loop in details

Deep dive into event loop (not required for this course)

https://nodejs.org/en/docs/guides/event-loop-timers-and-nexttick/



Summary

Node.js is a single-threaded non-blocking asynchronous concurrent technology

=

Node.js has:

call stack, event loop, callback queue

and some other APIs stuff

Callback example

```
const fs = require("fs");

var myCallbackFunction = function (err, data) {
    console.log(data.toString());
}

fs.readFile("input.txt", myCallbackFunction);

console.log("Program Ended");
```

Callback example #2

```
const fs = require("fs");

fs.readFile("input.txt", function (err, data) {
   console.log(data.toString());
});

console.log("Program Ended");
```

Error-First callback pattern

In Node.js, it is considered standard practice to handle errors in asynchronous functions by returning them as the first argument to the current function's callback.

If there is an error, the first parameter is passed an Error object with all the details. Otherwise, the first parameter is null.

```
var callback = function (error, retval) {
    if (error) {    // something went wrong
        console.log(error); // log error
        return; // and leave
    }

    console.log(retval); // ok, we can process returned value
}
```

Callback Hell / Pyramide of Doom

```
console.log("Pyramid of doooooooooom!😵😉")

step1(function (value1) {
    step2(function (value2) {
        step3(function (value3) {
            step4(function (value4) {
                step5(function (value5) {
                    step6(function (value6) {
                        step7(function (value7) {
                            //Do something with value 4
                            console.log(value7)
                        });
                    });
                });
            });
        });
    });
});
```