자바스크립트 비동기 마스터하기

Web/Mobile 공소나

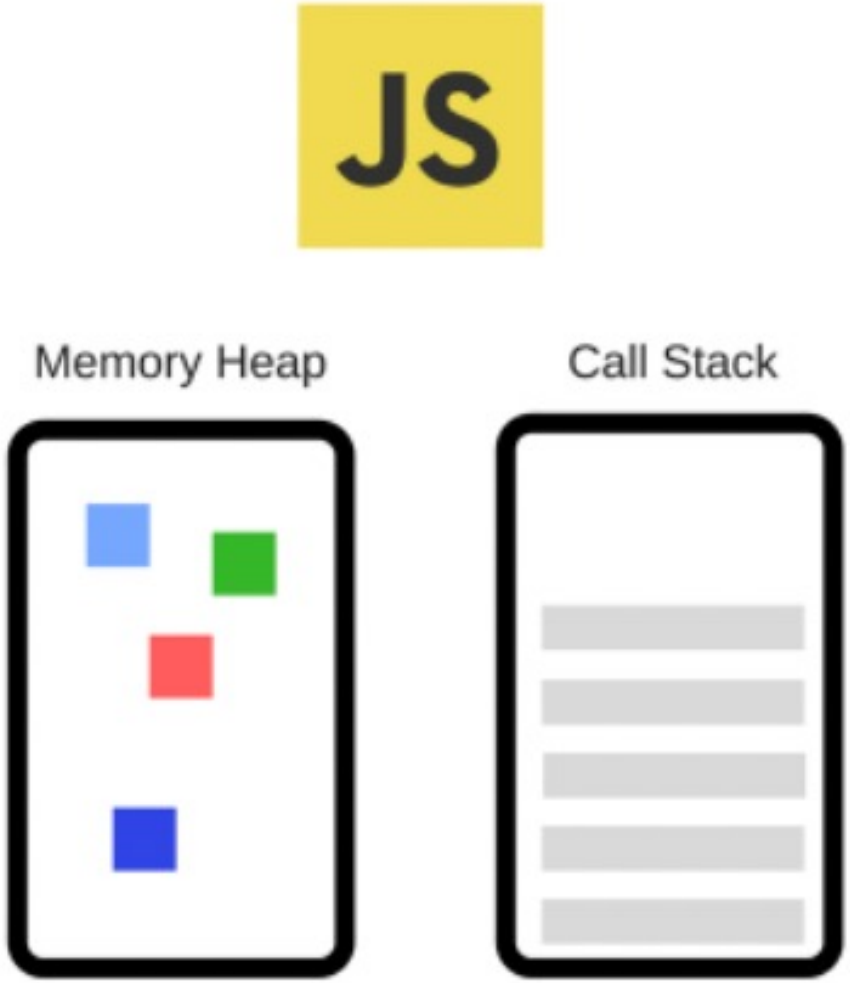
동기 ? 비동기?

자바스크립트 작동 방식

비동기 처리

자바스크립트

한번에 하나의 작업만 처리하는 방식으로 동작


JS
Memory Heap
Call Stack

code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

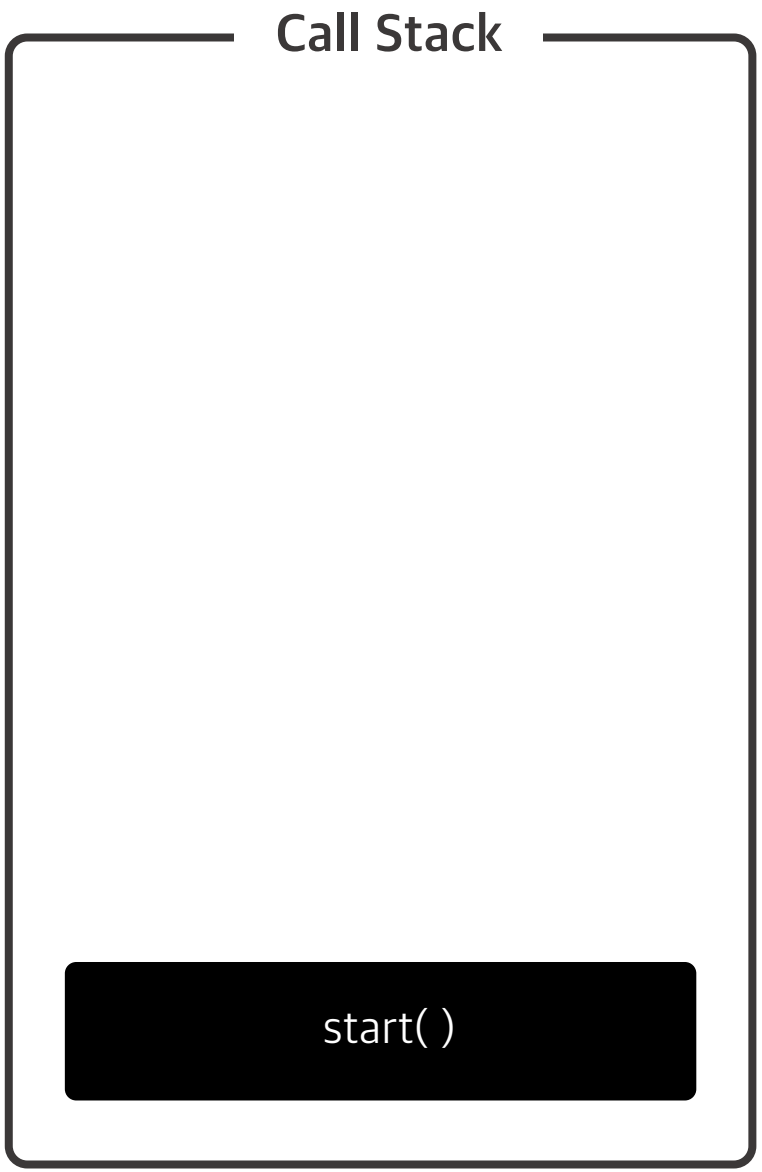
console

Call Stack

code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

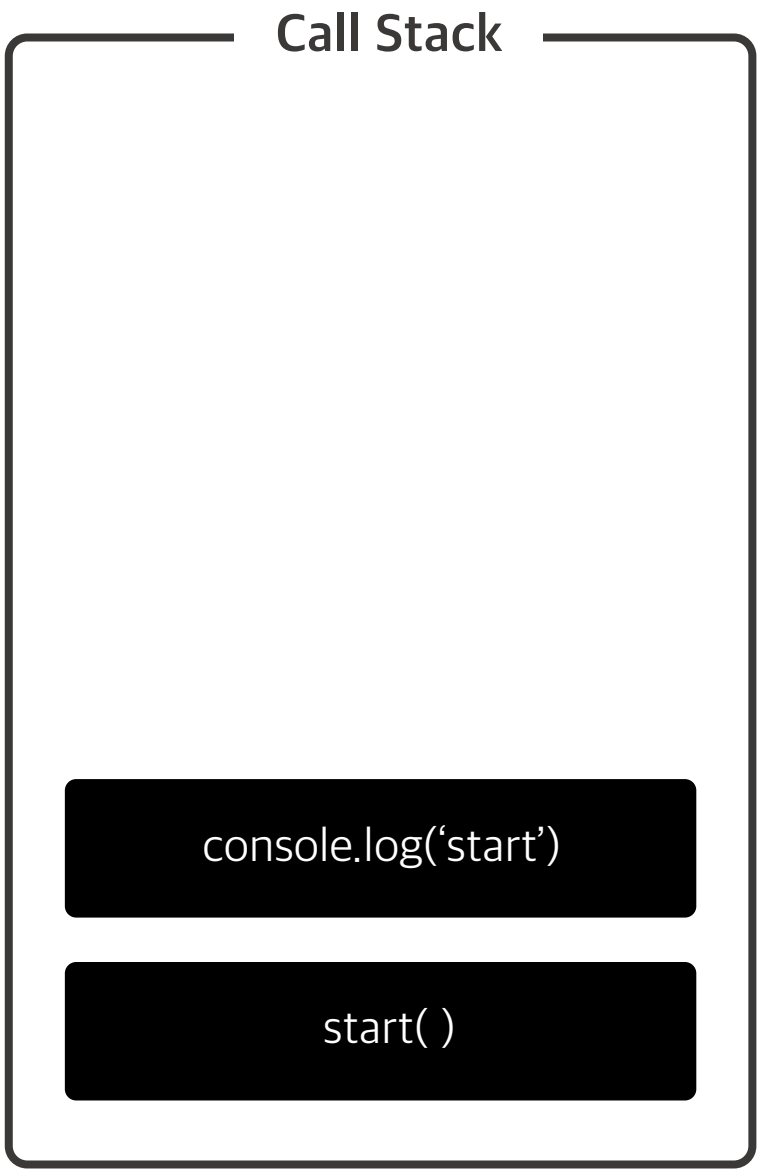
console



code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

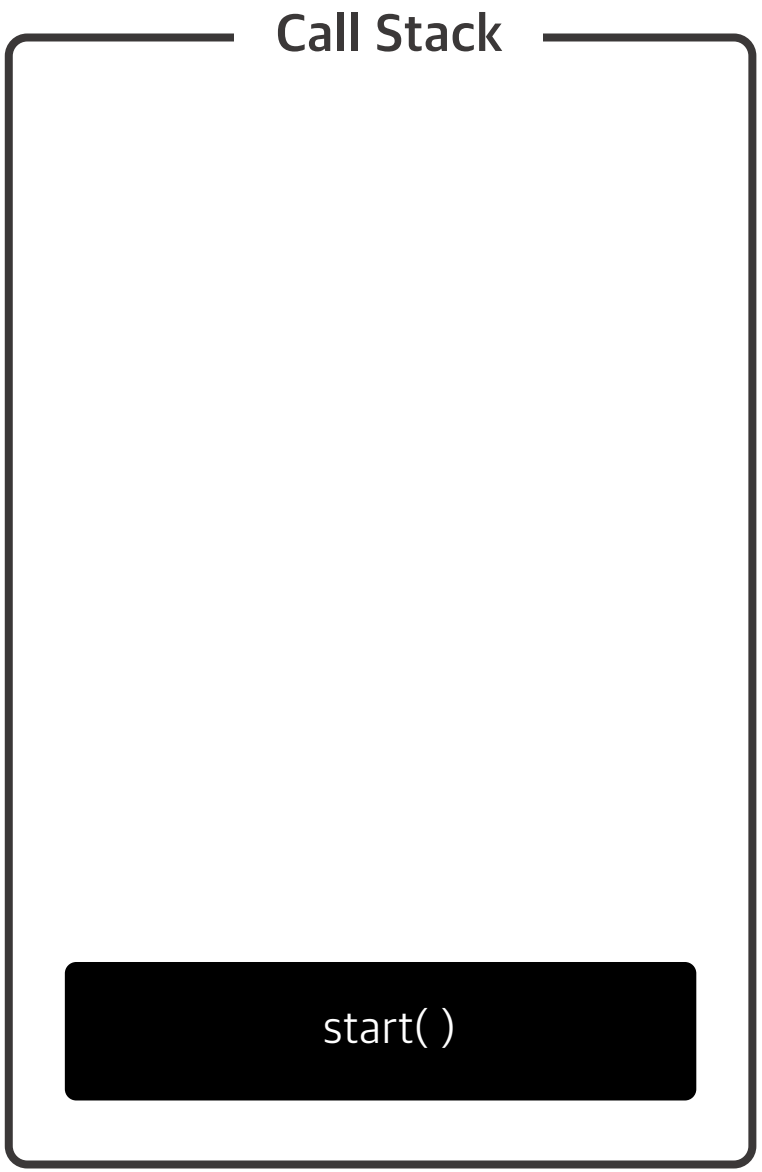


code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

```
start
```



code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

```
start
```

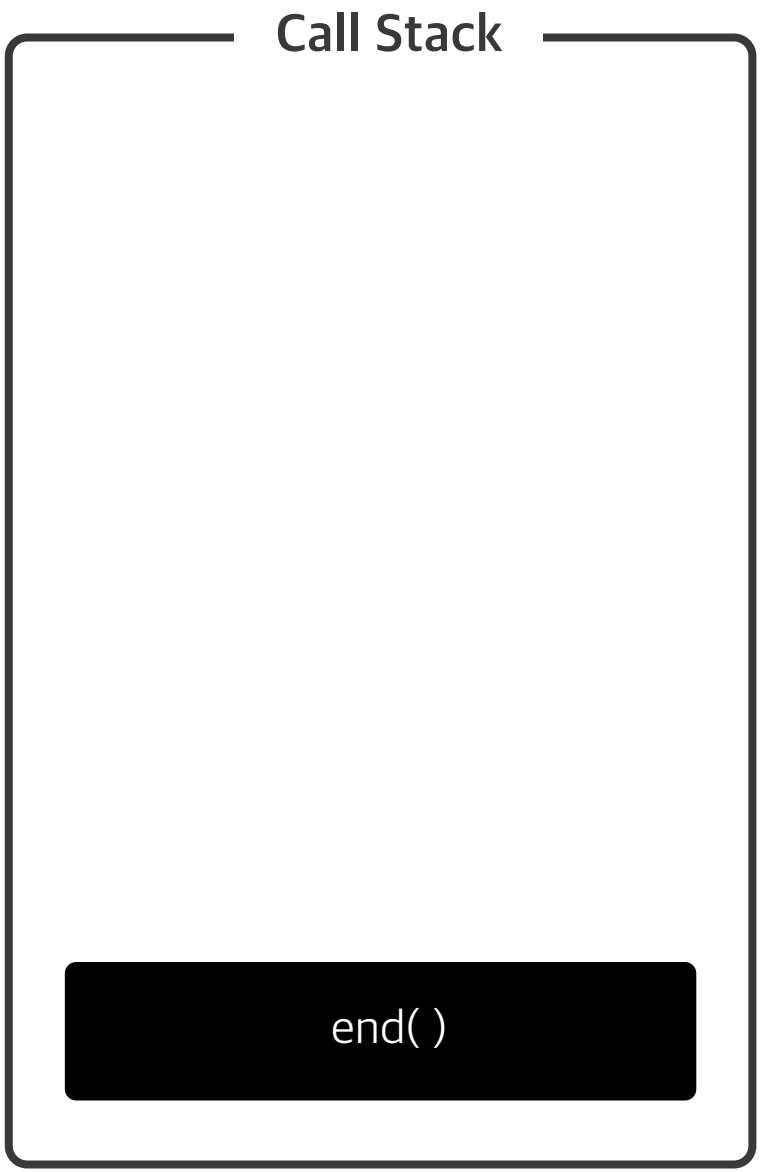
Call Stack

code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

```
start
```

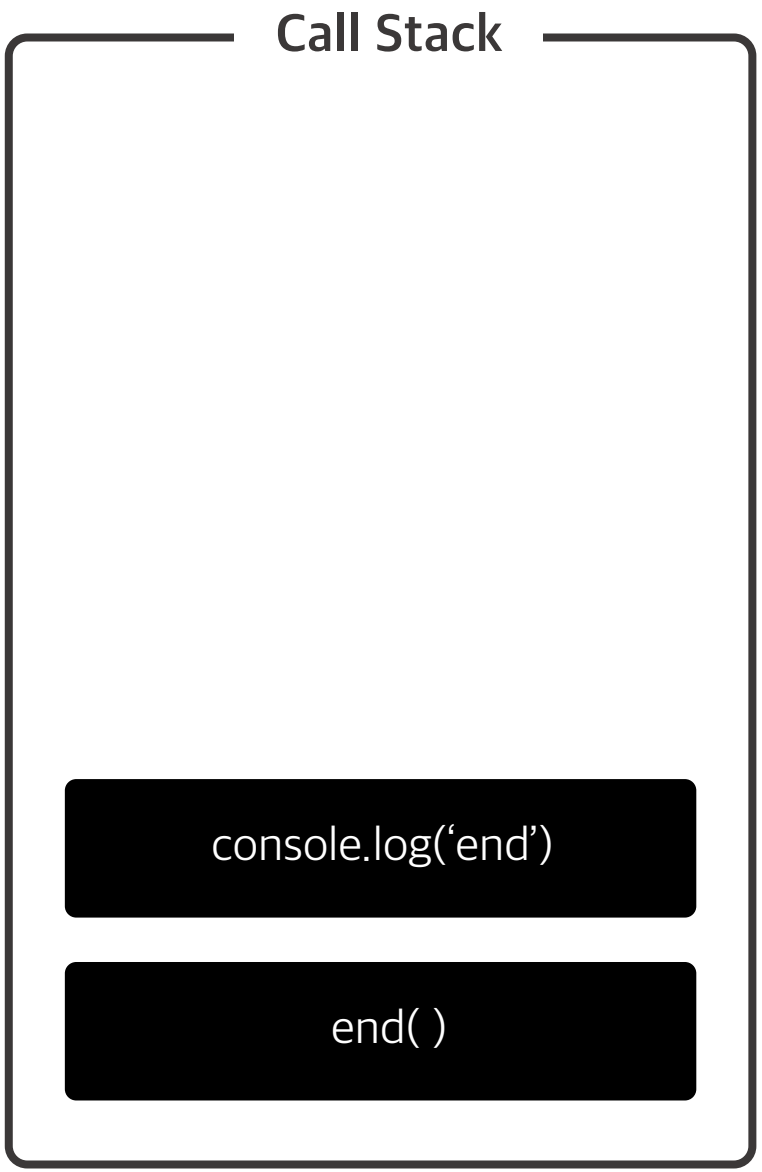


code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

```
start
```

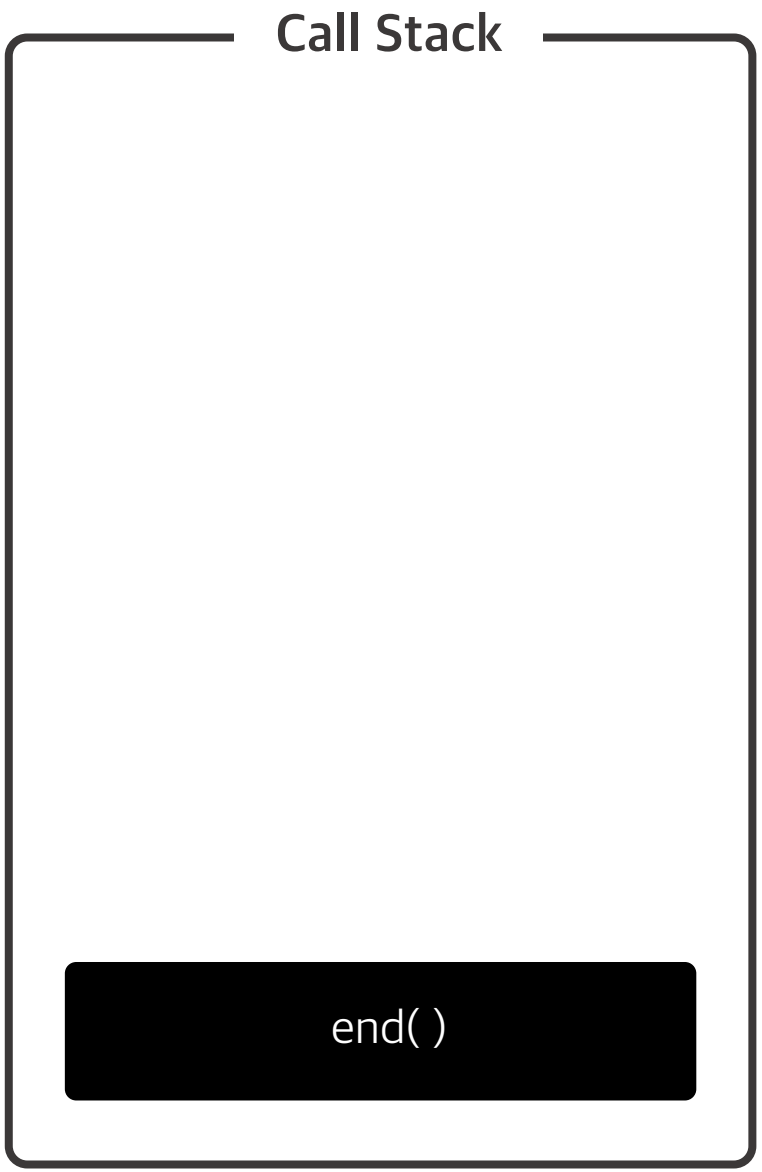


code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

```
start
end
```



code

```
function start() {
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

```
start
end
```

Call Stack

한번에 하나의 작업만 수행한다
= 동기적으로 동작한다

code

```
function start() {
    longlongTime();
    console.log('start');
}

function end() {
    console.log('end');
}

start();
end();
```

console

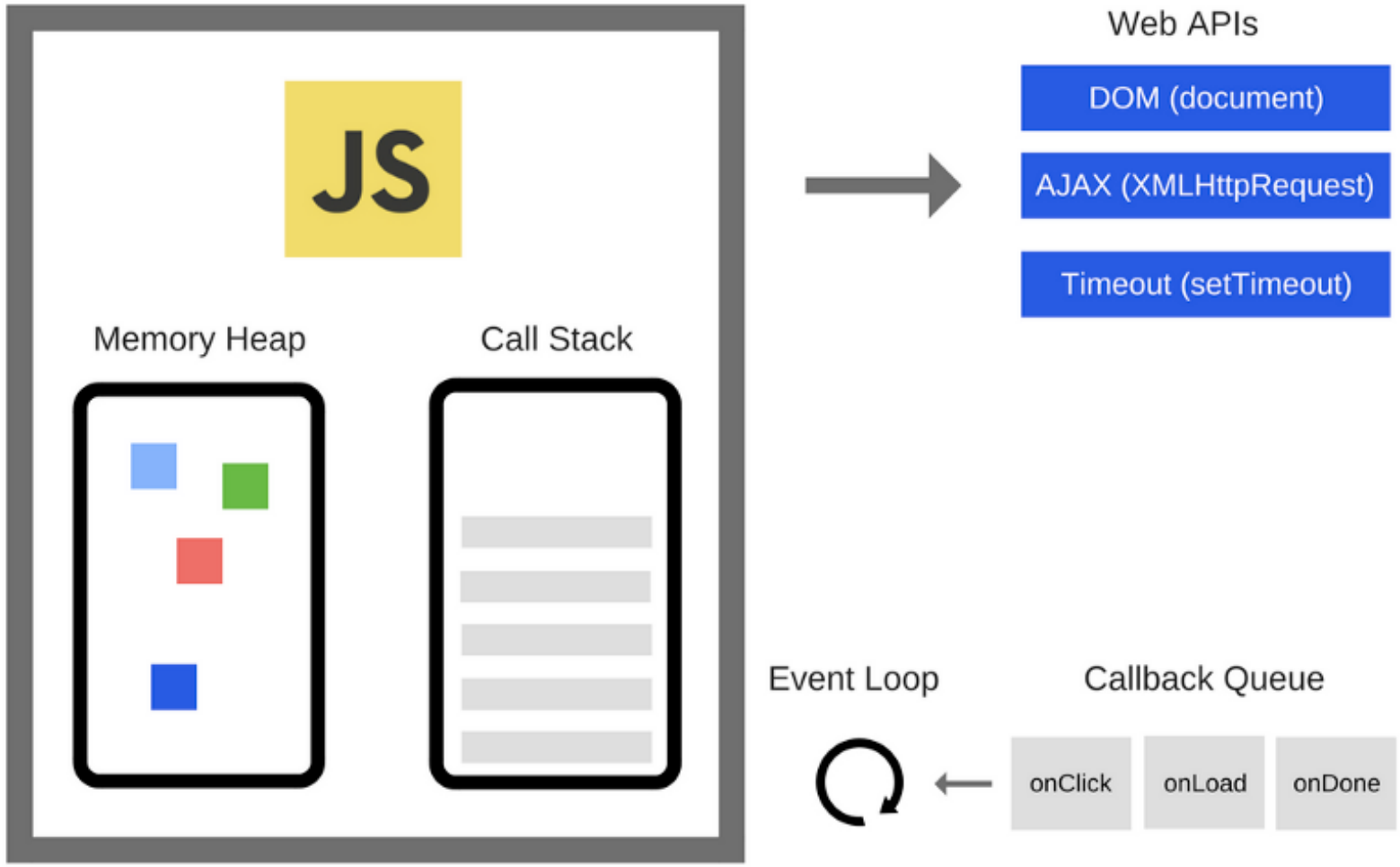
```
start
```

Call Stack

longlongTime( )

start( )

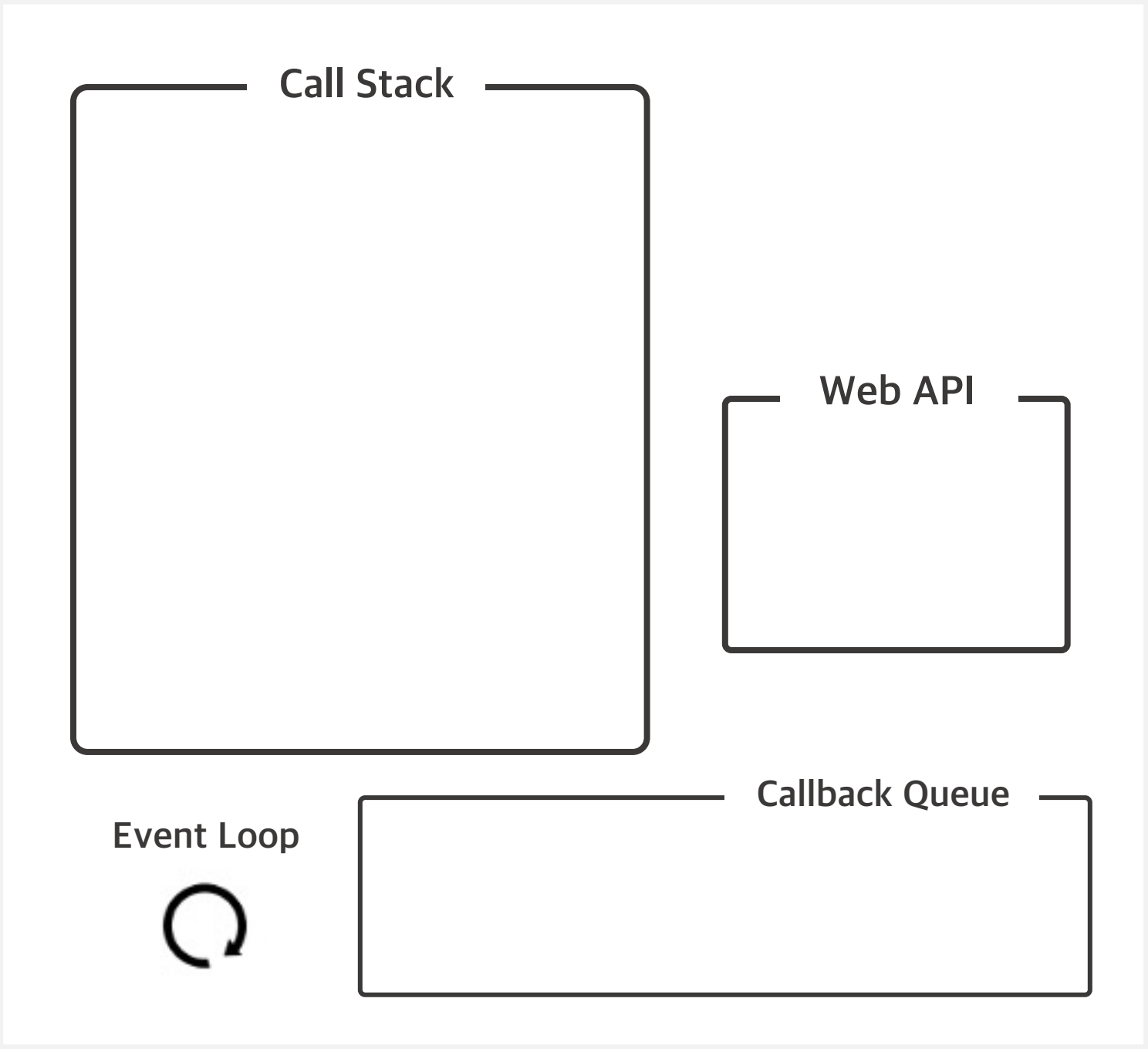
자바스크립트 런타임 환경은
자바스크립트를 비동기적으로 동작하게 만든다.


JS
Memory Heap
Call Stack
Web APIs
DOM (document)
AJAX (XMLHttpRequest)
Timeout (setTimeout)
Event Loop
Callback Queue
onClick
onLoad
onDone

code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

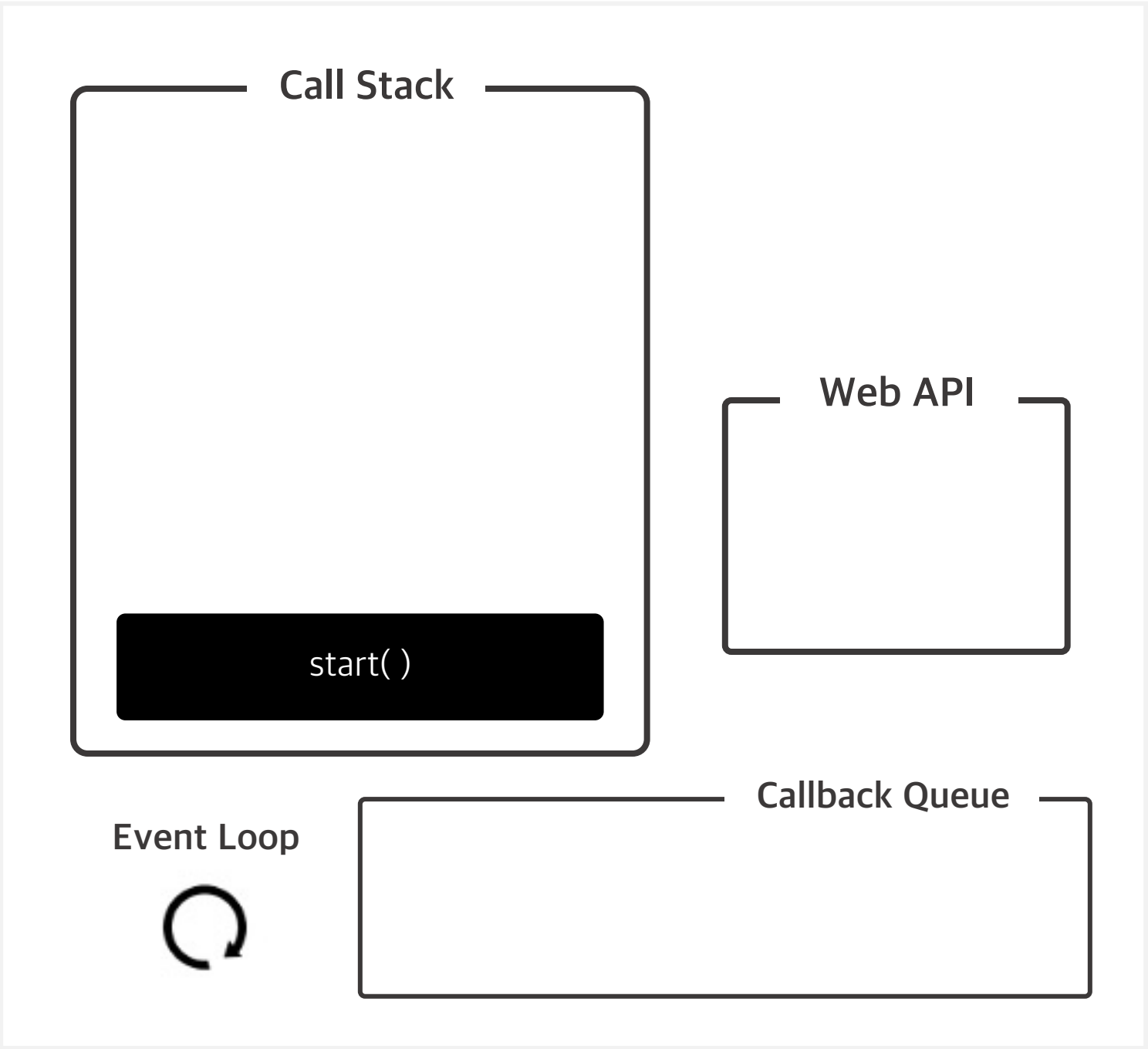
console



code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

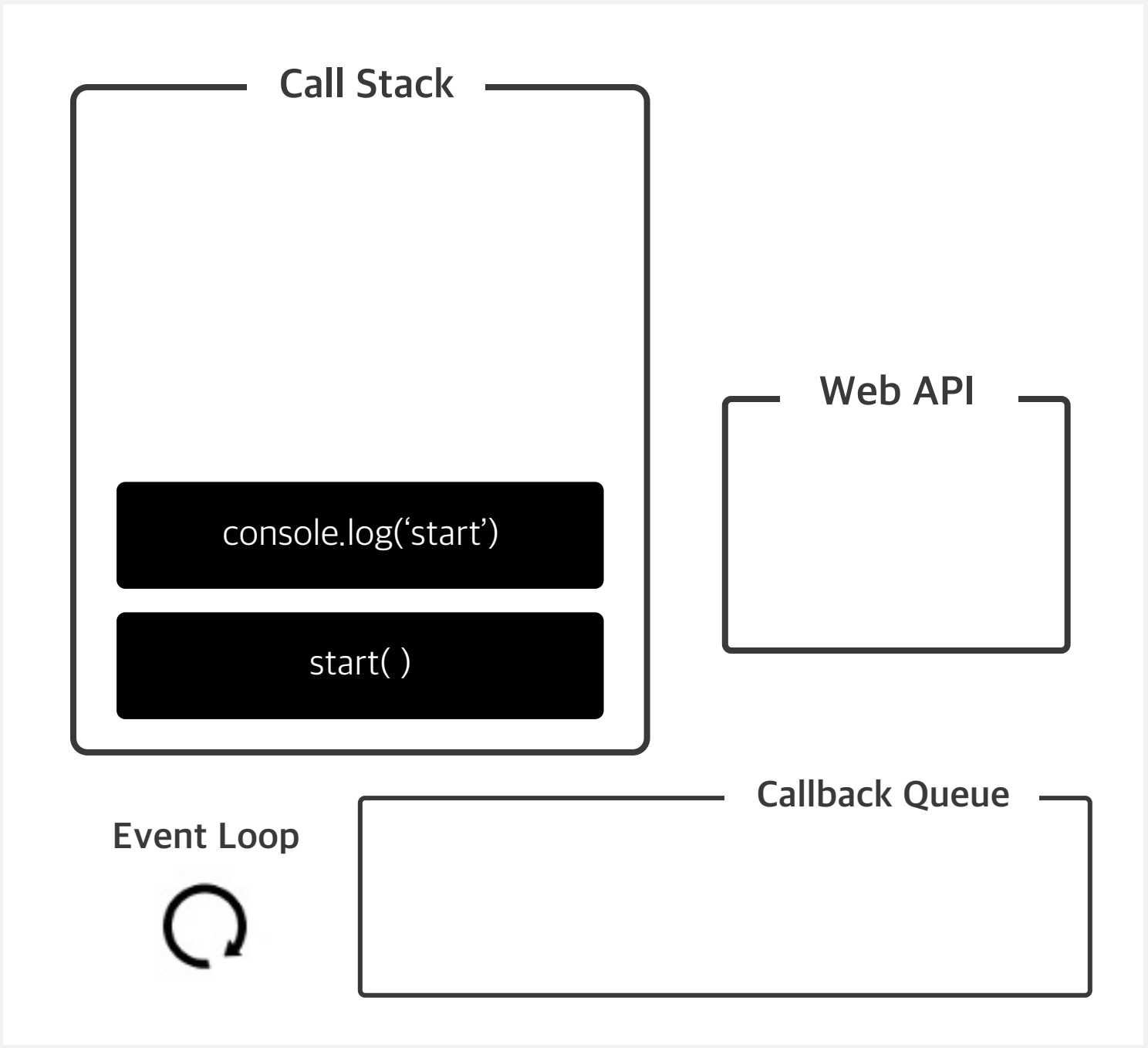
console



code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

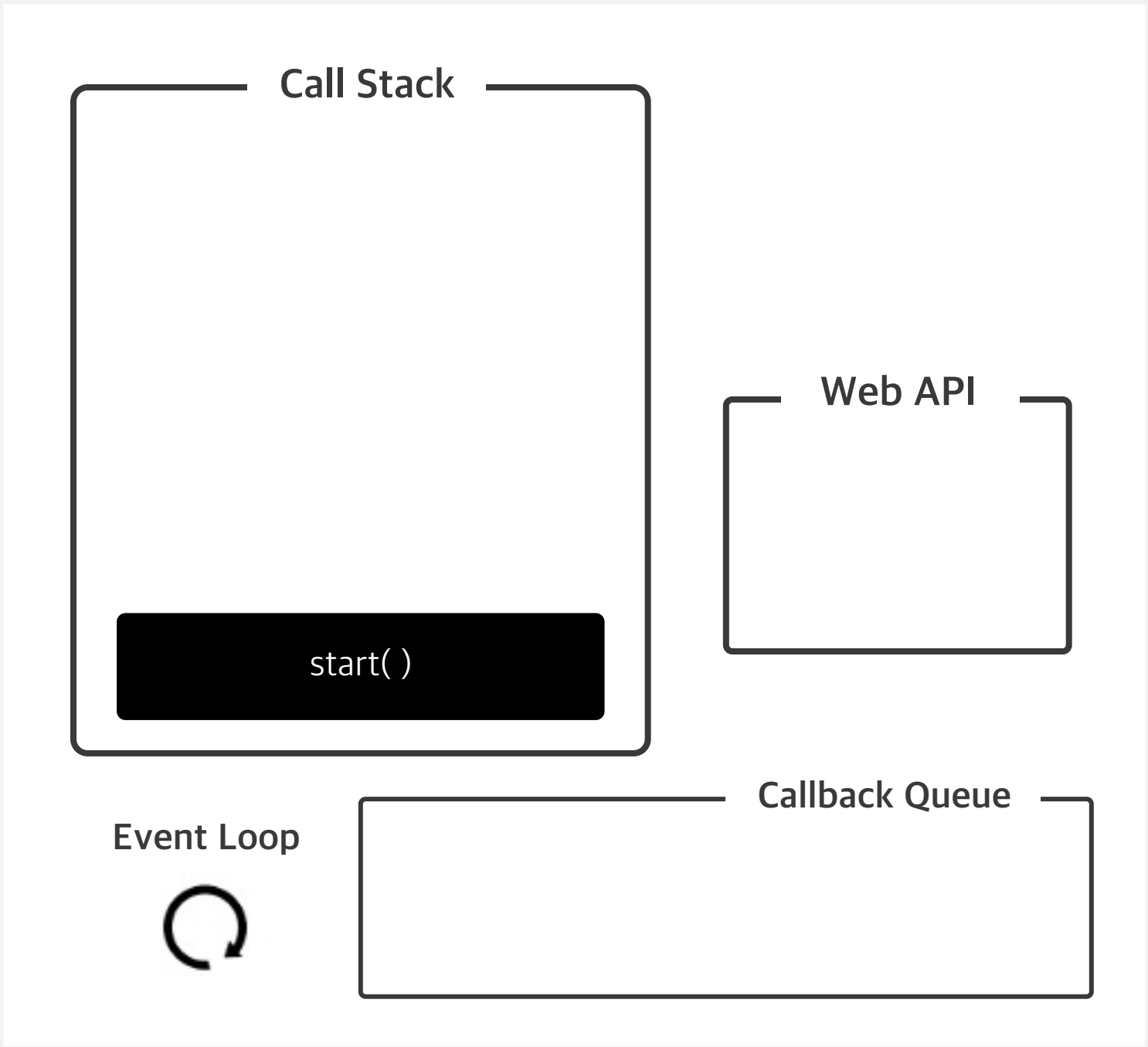


code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```

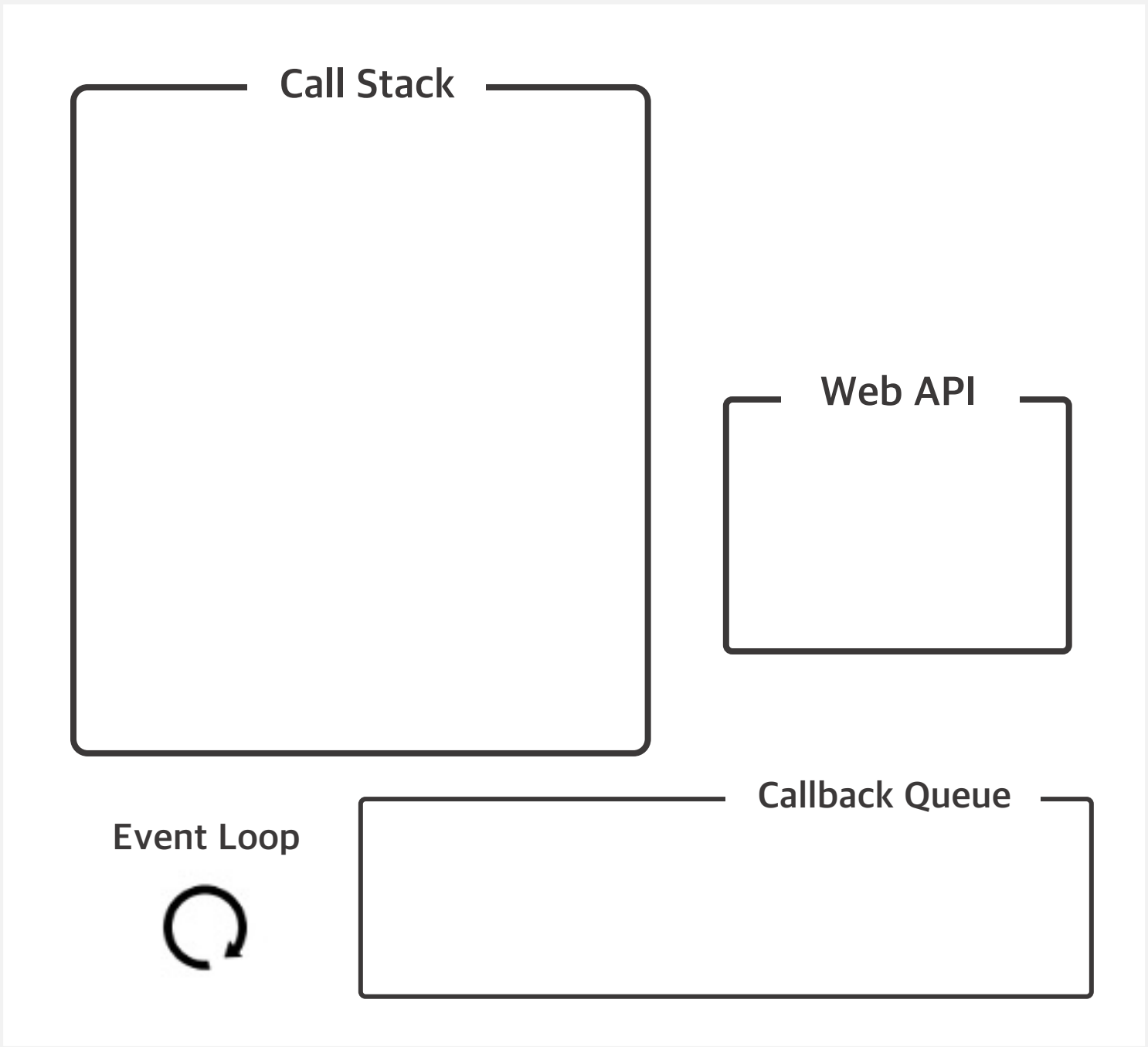


code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```

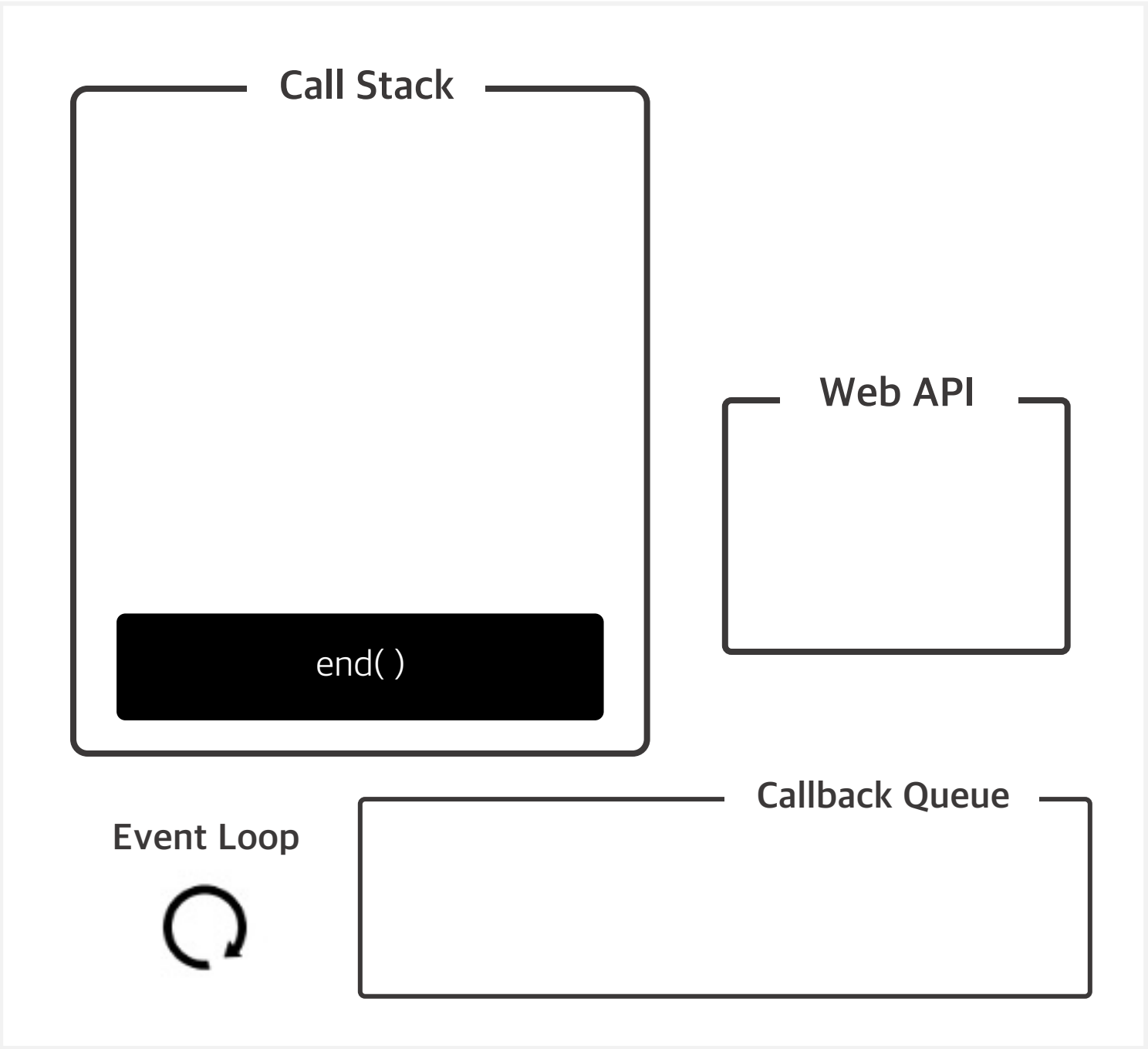


code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```

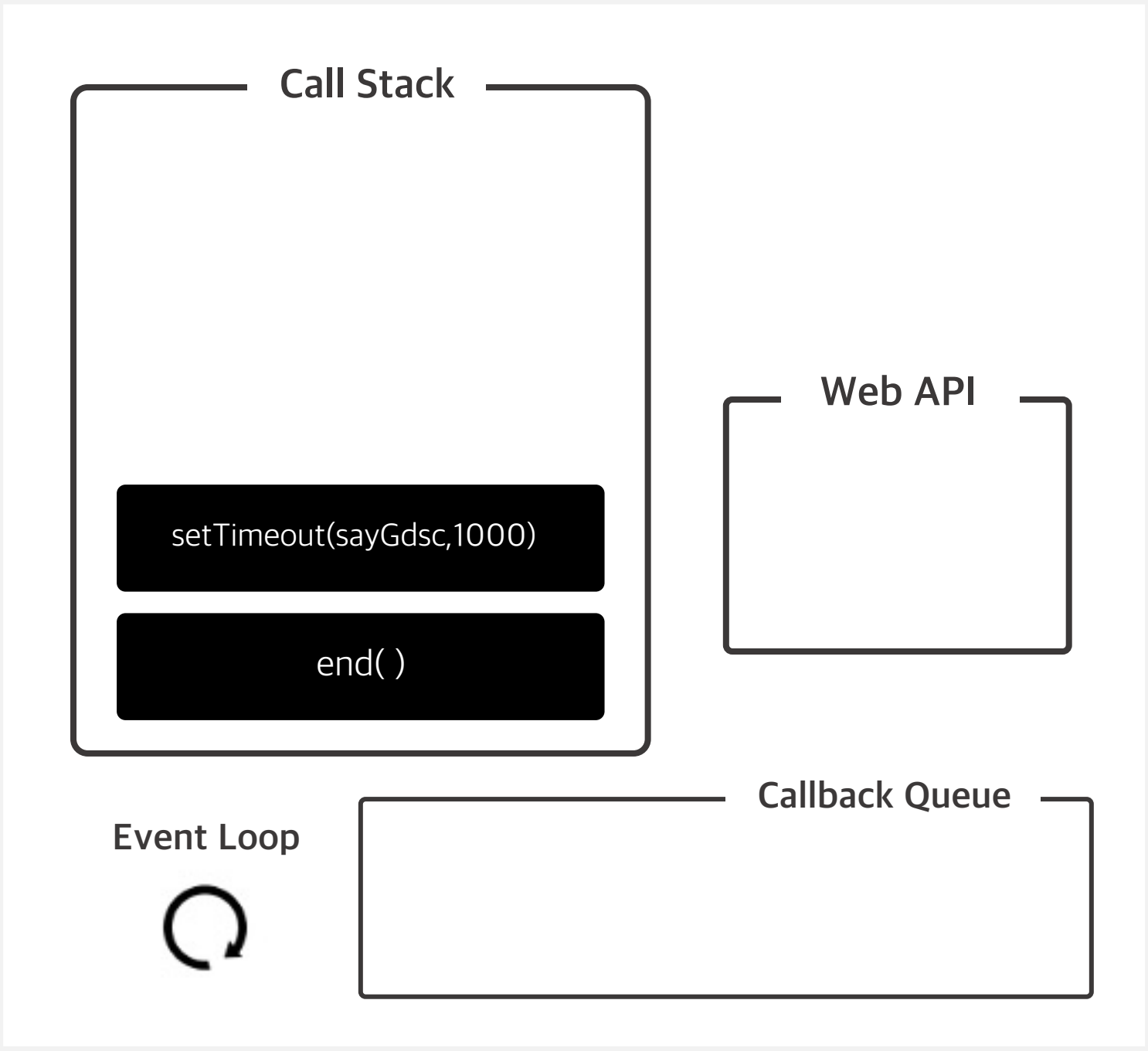


code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```

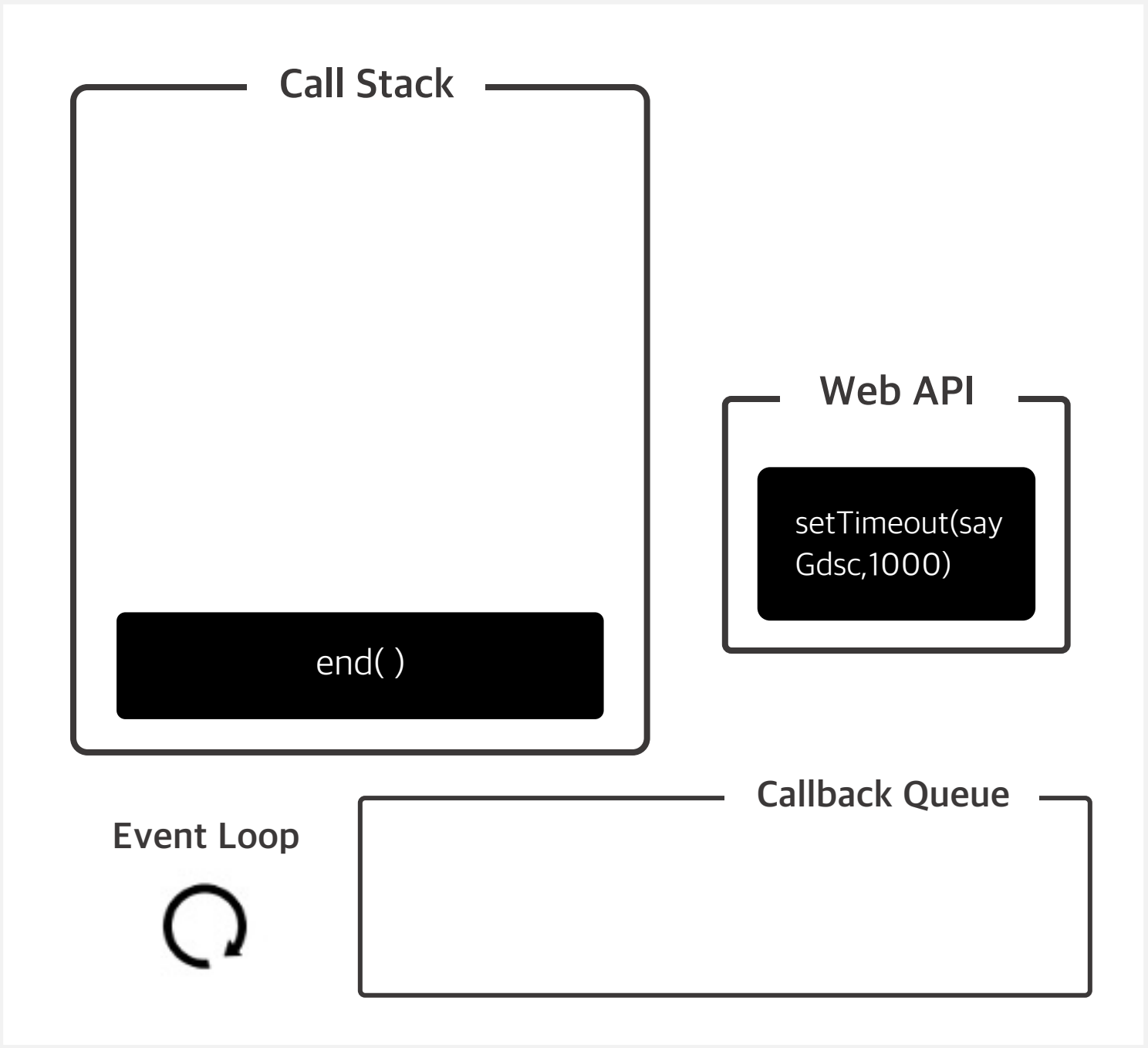


code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```



code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```

Call Stack

Web API

Event Loop

Callback Queue

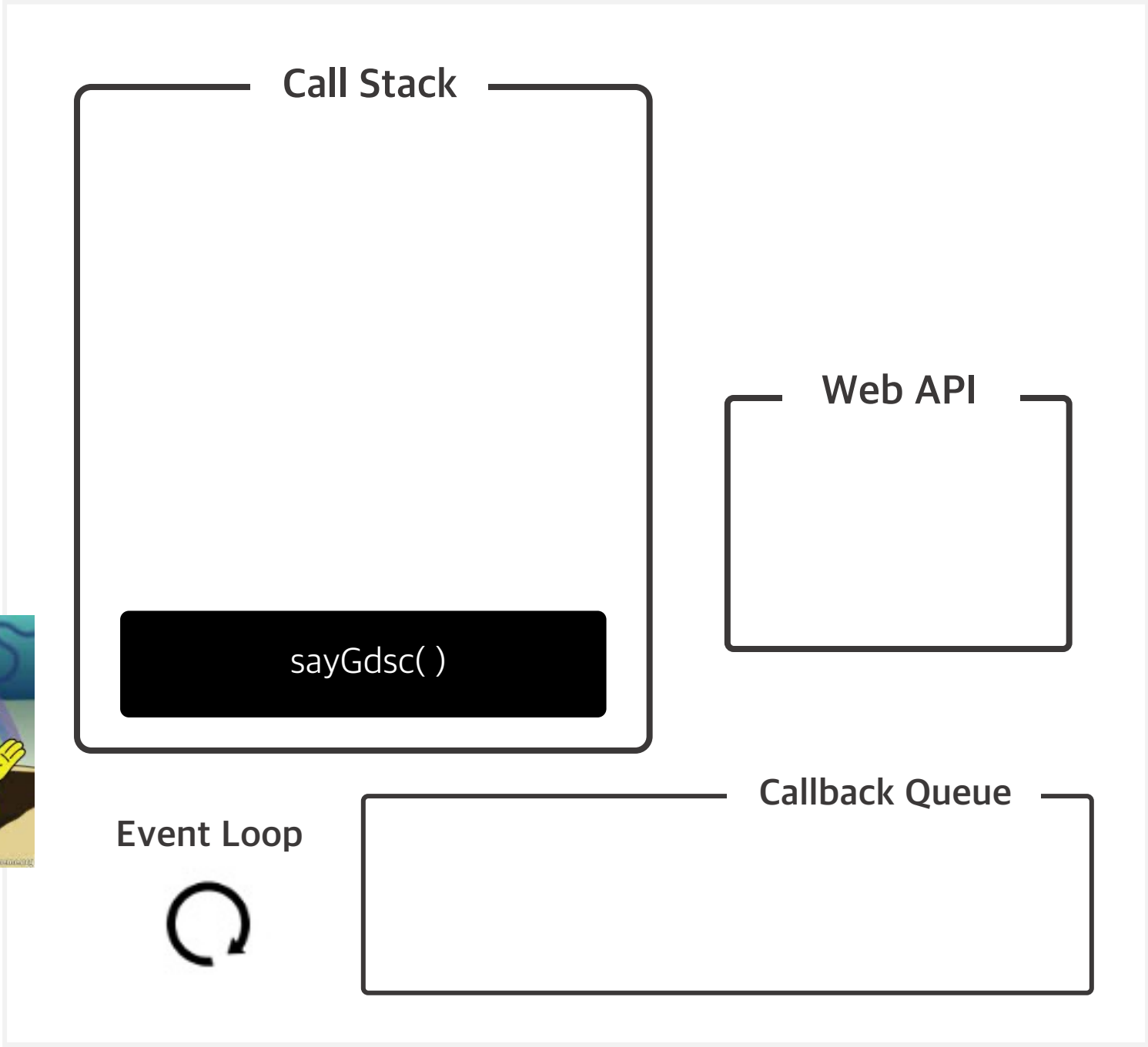
sayGdsc( )

code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```

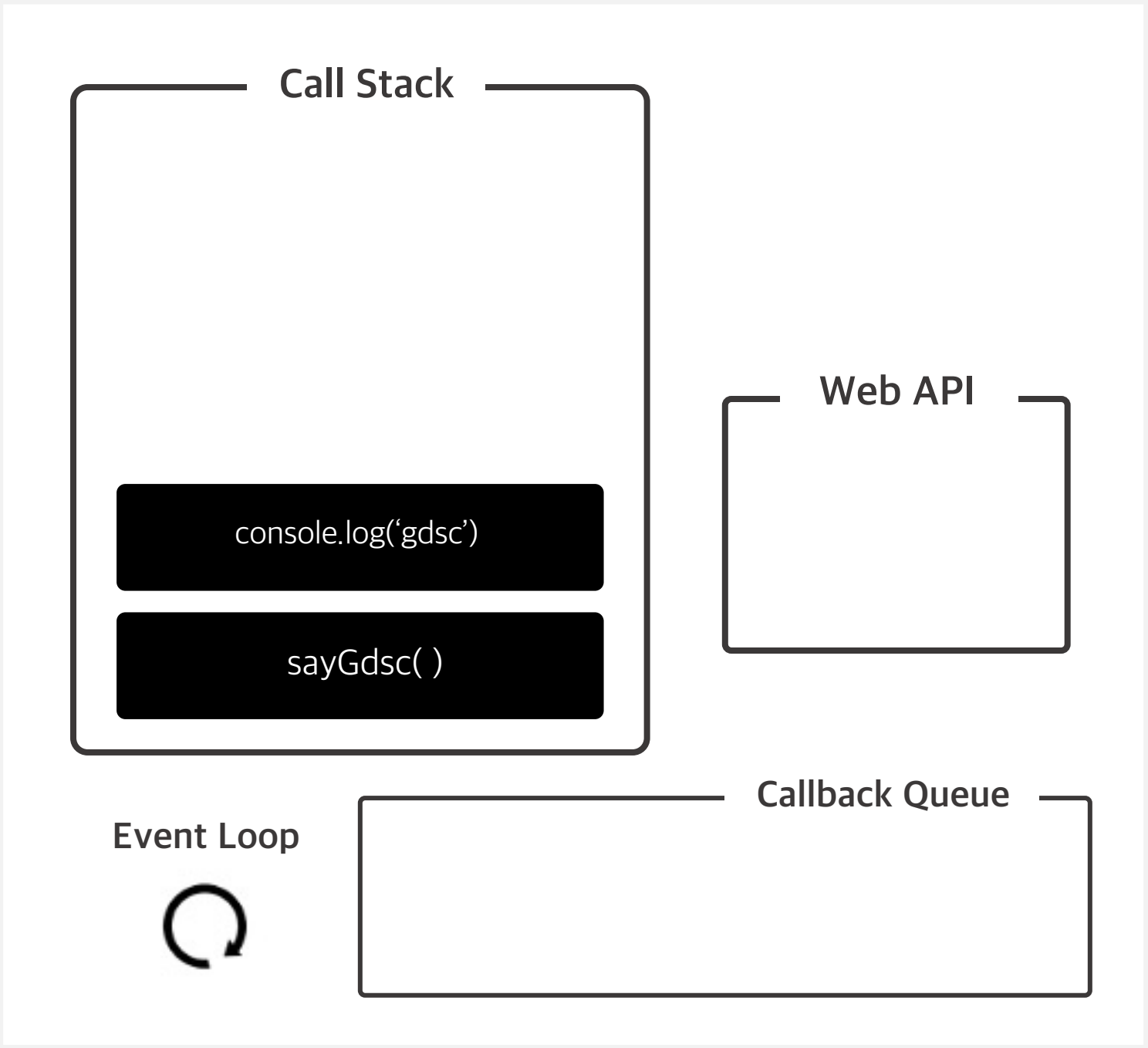


code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
```

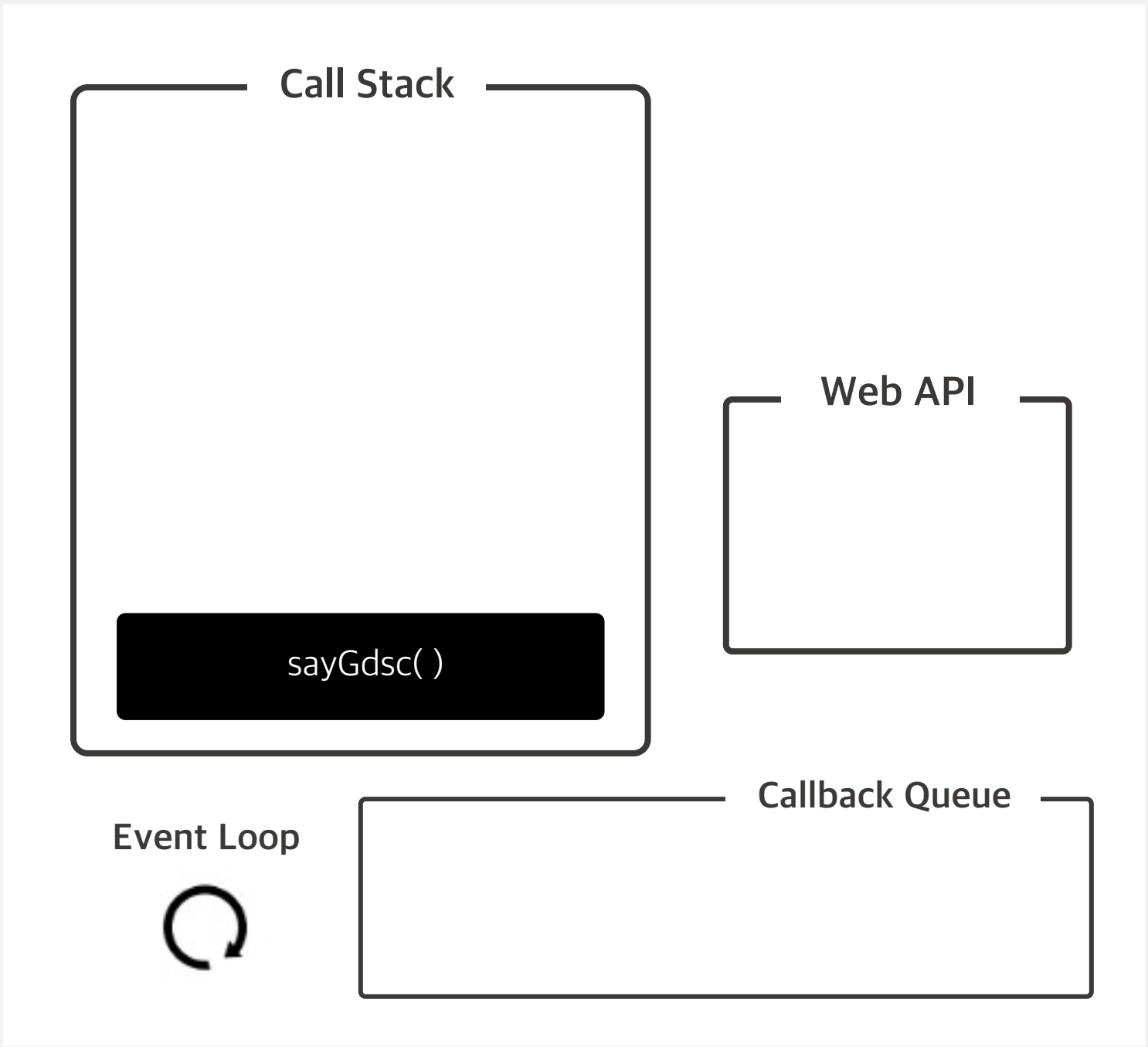


code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
gdsc
```

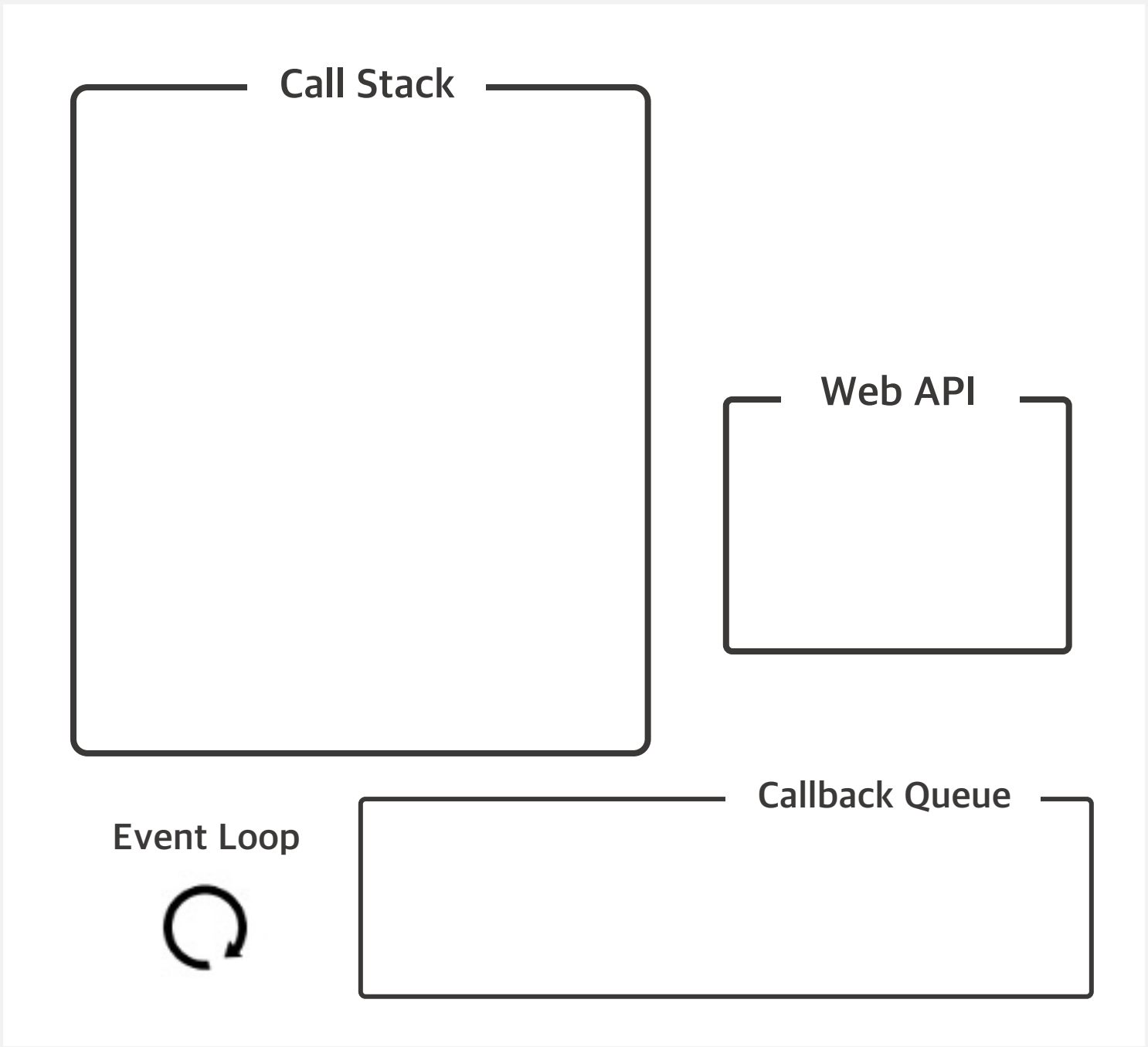


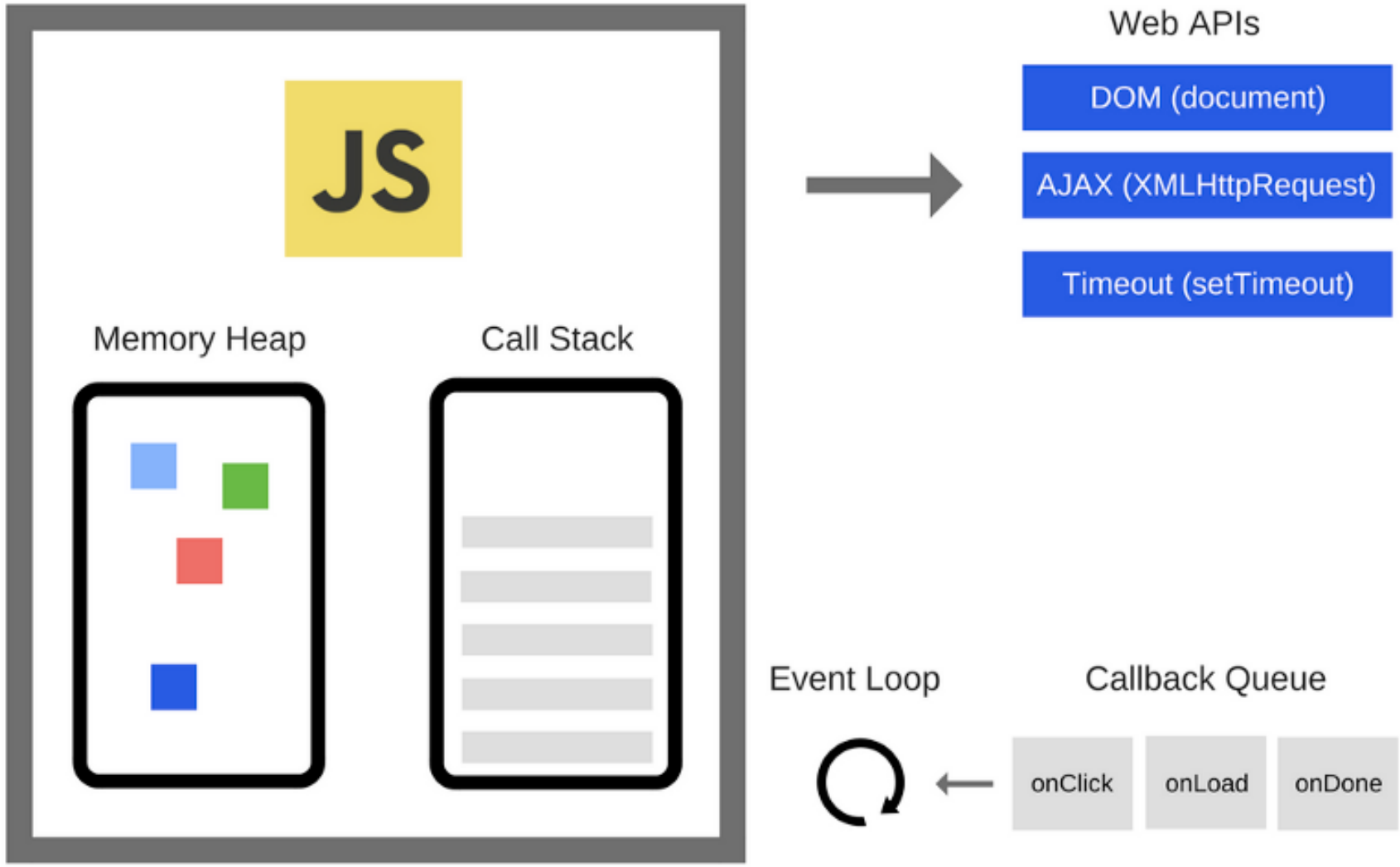
code

```
function start() {
    console.log('start');
}

function sayGdsc(){
    console.log('gdsc');
}

function end() {
    setTimeout(sayGdsc,1000);
}

start();
end();
```

console

```
start
gdsc
```




JS
Memory Heap
Call Stack
Web APIs
DOM (document)
AJAX (XMLHttpRequest)
Timeout (setTimeout)
Event Loop
Callback Queue
onClick
onLoad
onDone

1. setTimeout(callback,0)

```
function A(){
  console.log('A');
}

function B(){
  setTimeout(function(){
    console.log('B')
  },0)
}

function C(){
  console.log('C');
}

A();
B();
C();
```

```
CONSOLE ×
A
C
B
```

2. setTimeout(callback,2000) but..

loupe

help

Edit Rerun Pause Resume

```
function A(){
  console.log('A');
}

function B(){
  setTimeout(function(){
    console.log('B')

  },2000)

}

function C(){
  console.log('C');
}

function D(){
  console.log('D');
}

A();
B();
C();
D();
```

Click me!

Edit

Call Stack

console.log('A')

A()

Web Apis

Callback Queue

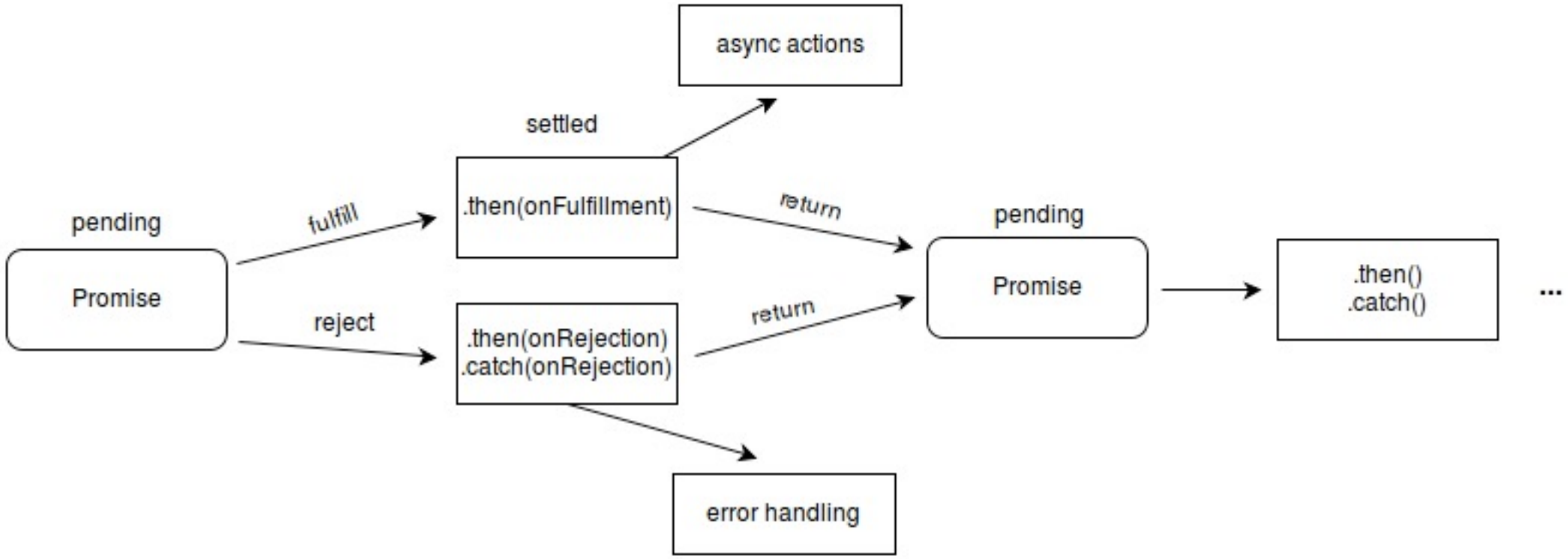
자바스크립트 비동기 처리하는 3가지 방법

1. 콜백 함수

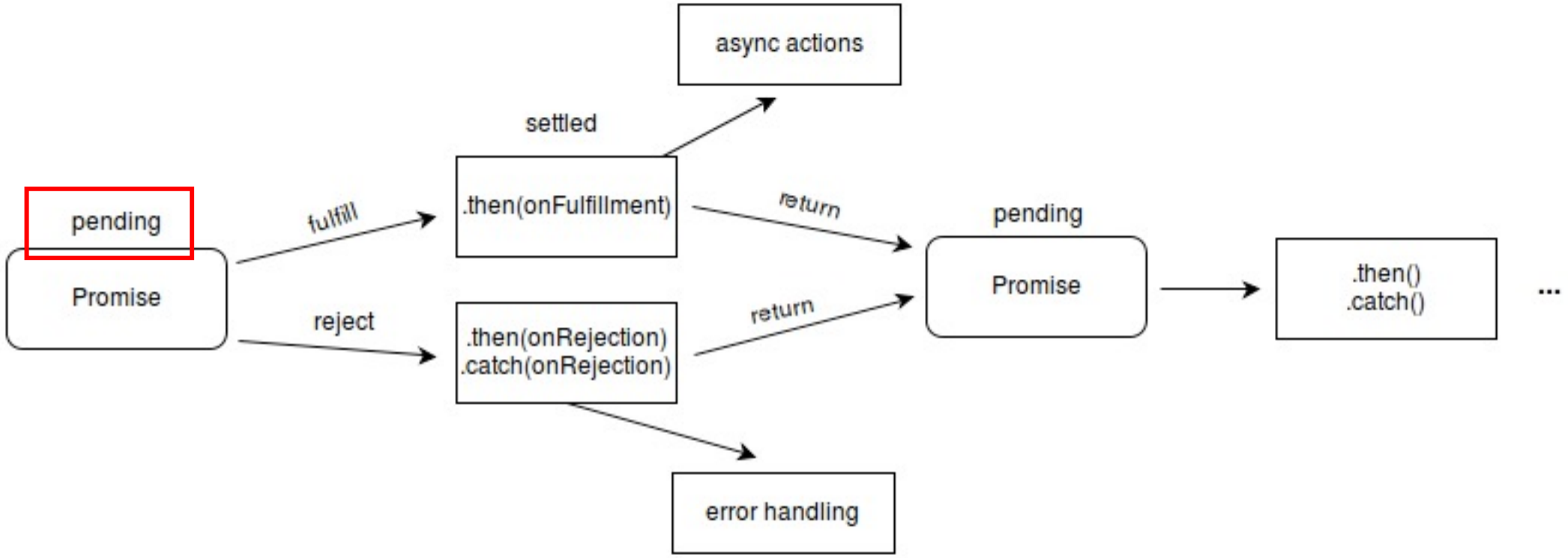
콜백 지옥

```
// Callback Hell


a(function (resultsFromA) {
    b(resultsFromA, function (resultsFromB) {
        c(resultsFromB, function (resultsFromC) {
            d(resultsFromC, function (resultsFromD) {
                e(resultsFromD, function (resultsFromE) {
                    f(resultsFromE, function (resultsFromF) {
                        console.log(resultsFromF);
                    })
                })
            })
        })
    })
});

```

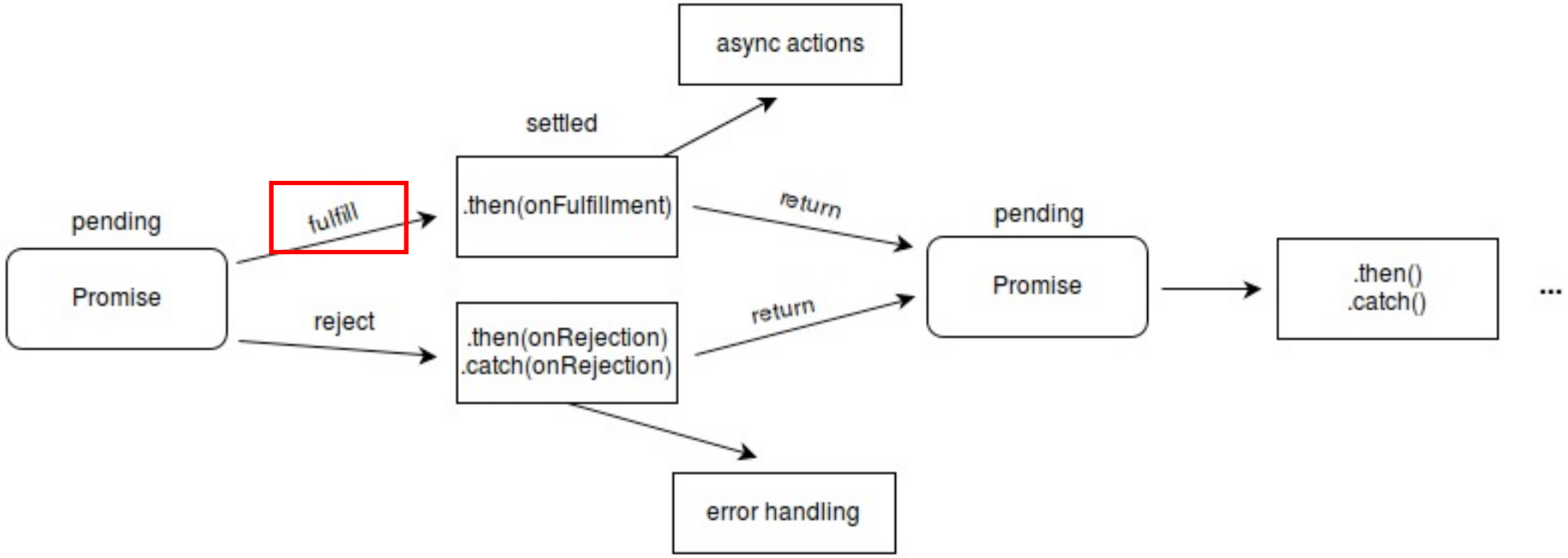
2. Promise


pending
Promise
fulfill
reject
settled
.then(onFulfillment)
.then(onRejection)
.catch(onRejection)
async actions
error handling
return
return
pending
Promise
.then()
.catch()
...

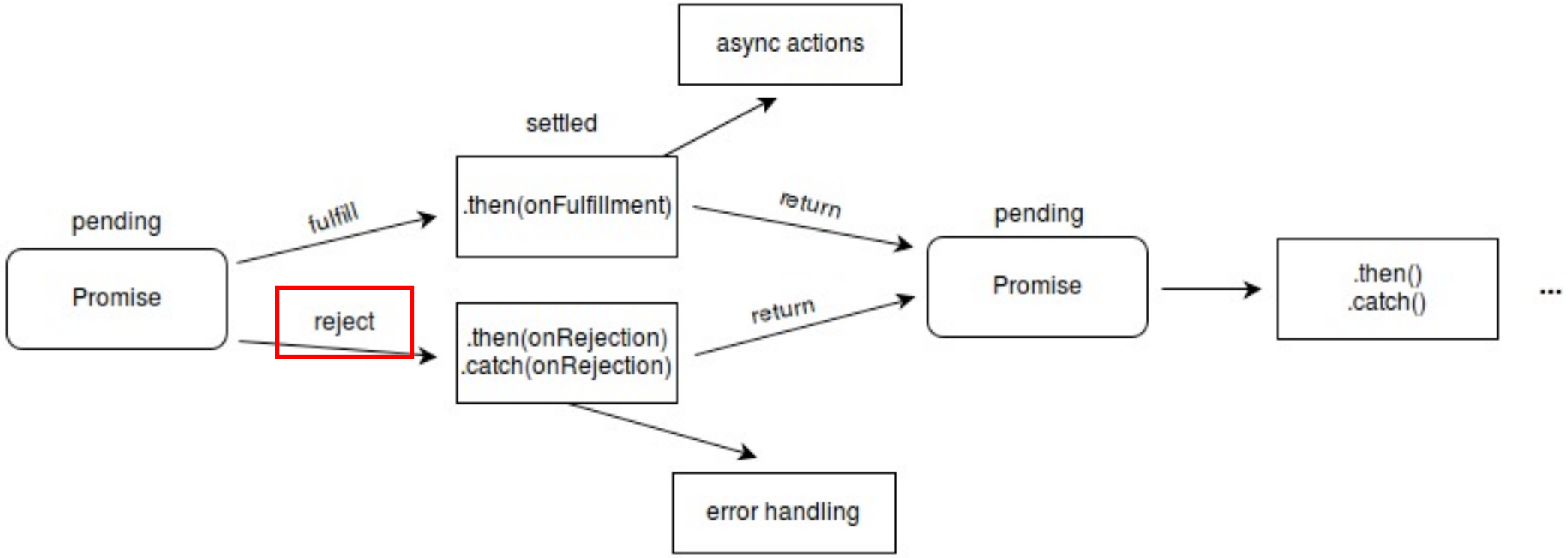
- pending : 비동기 처리 로직이 아직 미완료 상태로 대기 중



- pending : 비동기 처리 로직이 아직 미완료 상태로 대기 중
- fullfilled : 비동기 처리 로직이 완료되어 promise가 결과 값을 반환한 상태



- pending : 비동기 처리 로직이 아직 미완료 상태로 대기 중
- fullfilled : 비동기 처리 로직이 완료되어 promise가 결과 값을 반환한 상태
- rejected : 비동기 처리를 실패하거나 오류가 발생한 상태



1. new Promise( )

```
function getData(){
  return new Promise(function(resolve,reject){
    const data = 10;
    resolve(data);
    reject(new Error('just error'))
  })
}
getData()
.then(function(data){console.log(data)})
.catch(function(err){console.log(err)})
```

2. Promise.resolve()

```
function getData2(){
  const data = 10;
  return Promise.resolve(data)
}

getData2().then(function(data){
  console.log(data)
})
```

* Promise.reject()도 같은 맥락에서 사용된다.

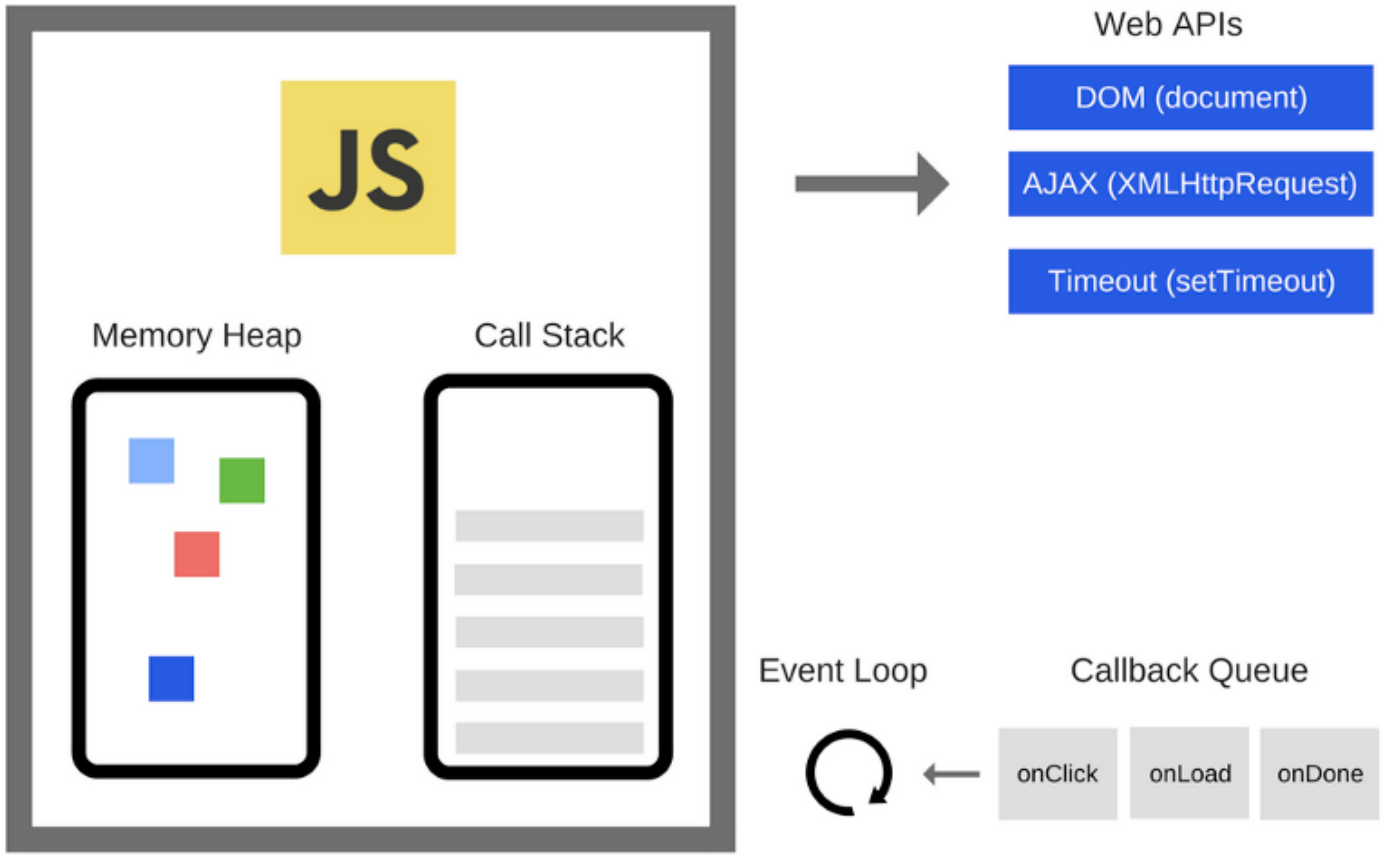
setTimeout 과 Promise..

```
setTimeout(() => console.log("timeout"));

Promise.resolve()
  .then(() => console.log("promise"));

console.log("code");
```

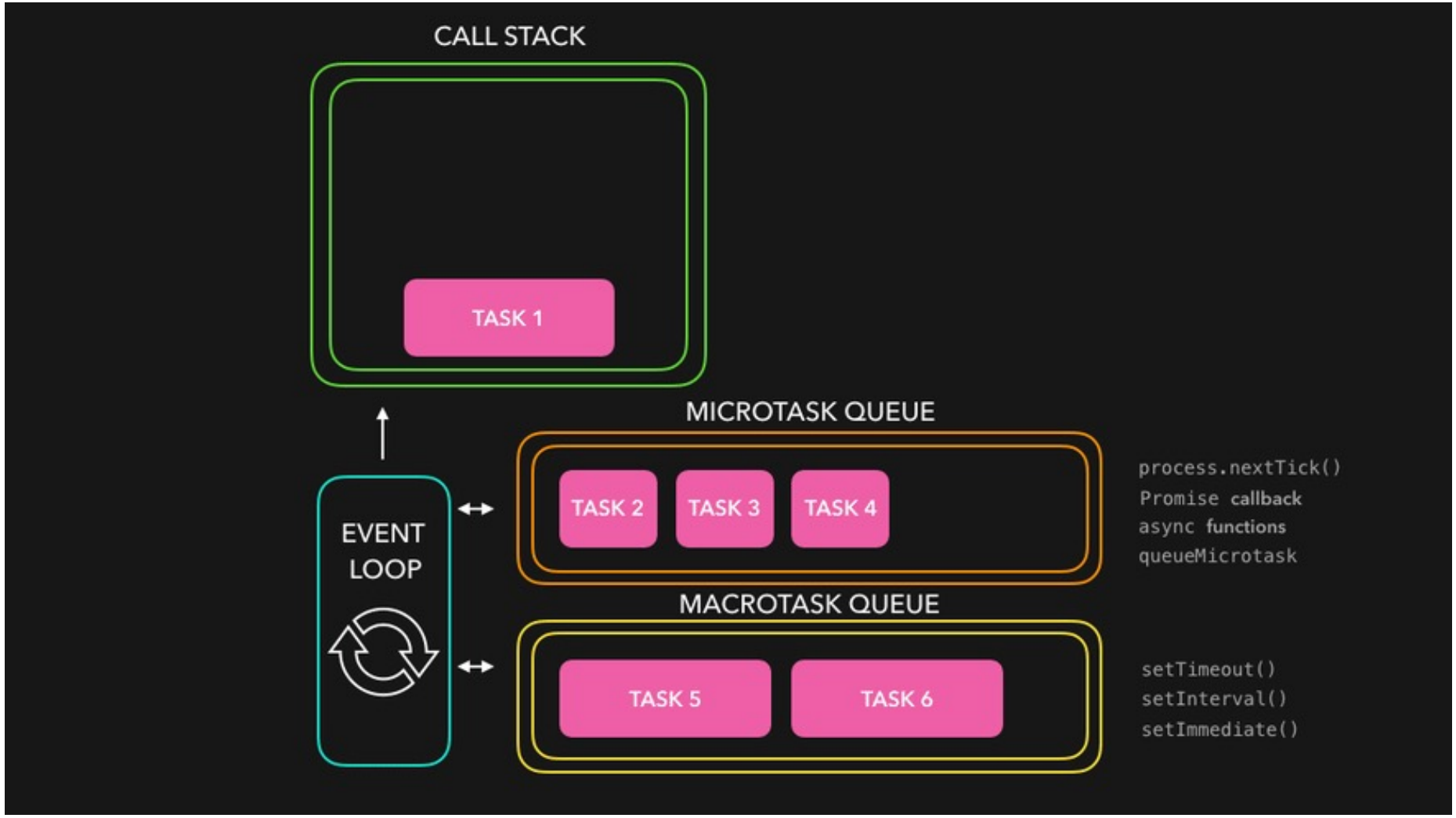
setTimeout 과 Promise..

```
setTimeout(() => console.log("timeout"));

Promise.resolve()
  .then(() => console.log("promise"));

console.log("code");
```

결과 : code -> promise -> timeout


JS
Memory Heap
Call Stack
Web APIs
DOM (document)
AJAX (XMLHttpRequest)
Timeout (setTimeout)
Event Loop
Callback Queue
onClick
onLoad
onDone

microtask queue 와 macrotask queue



microtask queue > macrotask queue

```
getData()
  .then(function(data) {
    // ...
  })
  .then(function() {
    // ...
  })
  .then(function() {
    // ...
  });
```

3. await/async

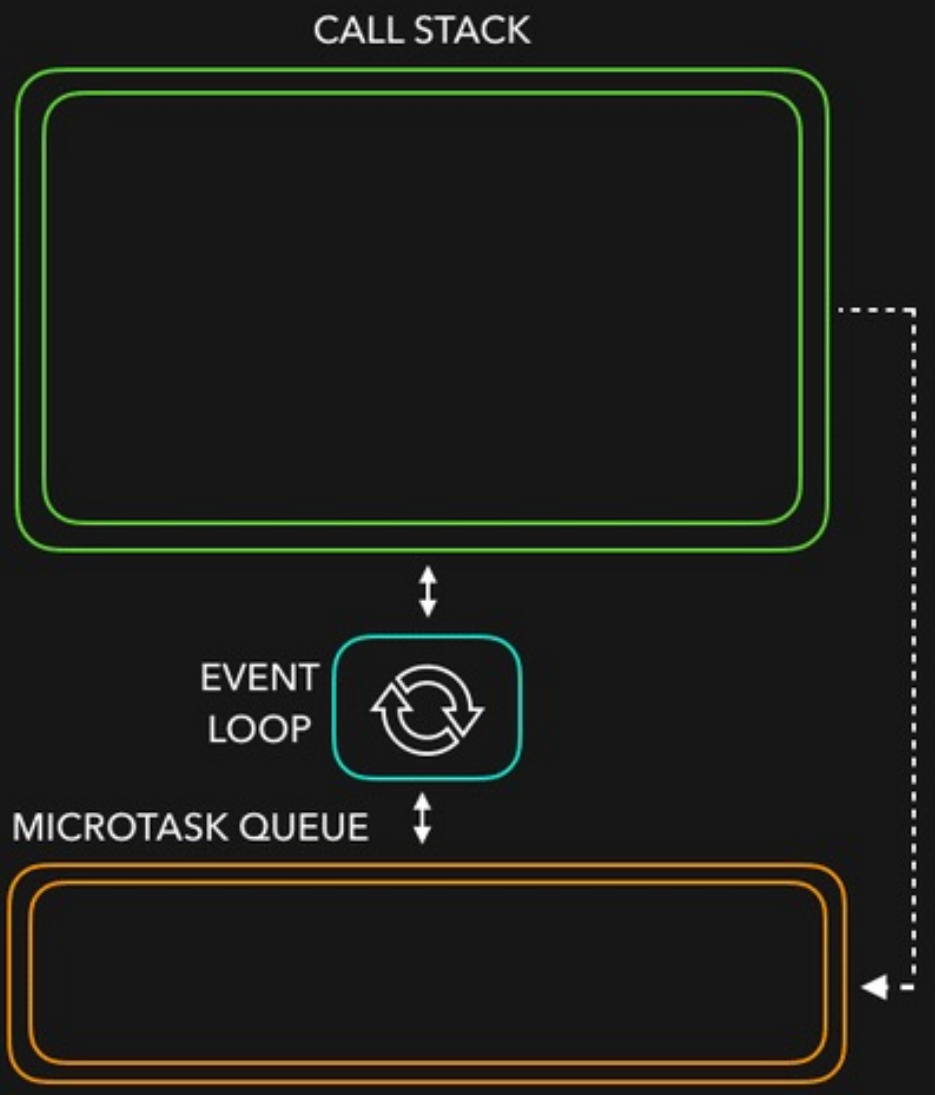
```javascript
function fetchUser() {
  return fetch(url).then(function(response) {
    return response.json();
  });
}

async function getData(){
  const data = await fetchUser();
  console.log(data);
}
```

- async : 함수 앞에 기재 -> Promise 객체 반환
- await : async 함수 일시 중지 -> promise 결과값 기다린 후 함수 실행 재개
- await 는 async 함수 내에서만 사용할 것!

```
const one = () => Promise.resolve('One!')

async function myFunc() {
  console.log('In function!')
  const res = await one()
  console.log(res)
}

console.log('Before function!')
myFunc();
console.log('After function!')
```

node

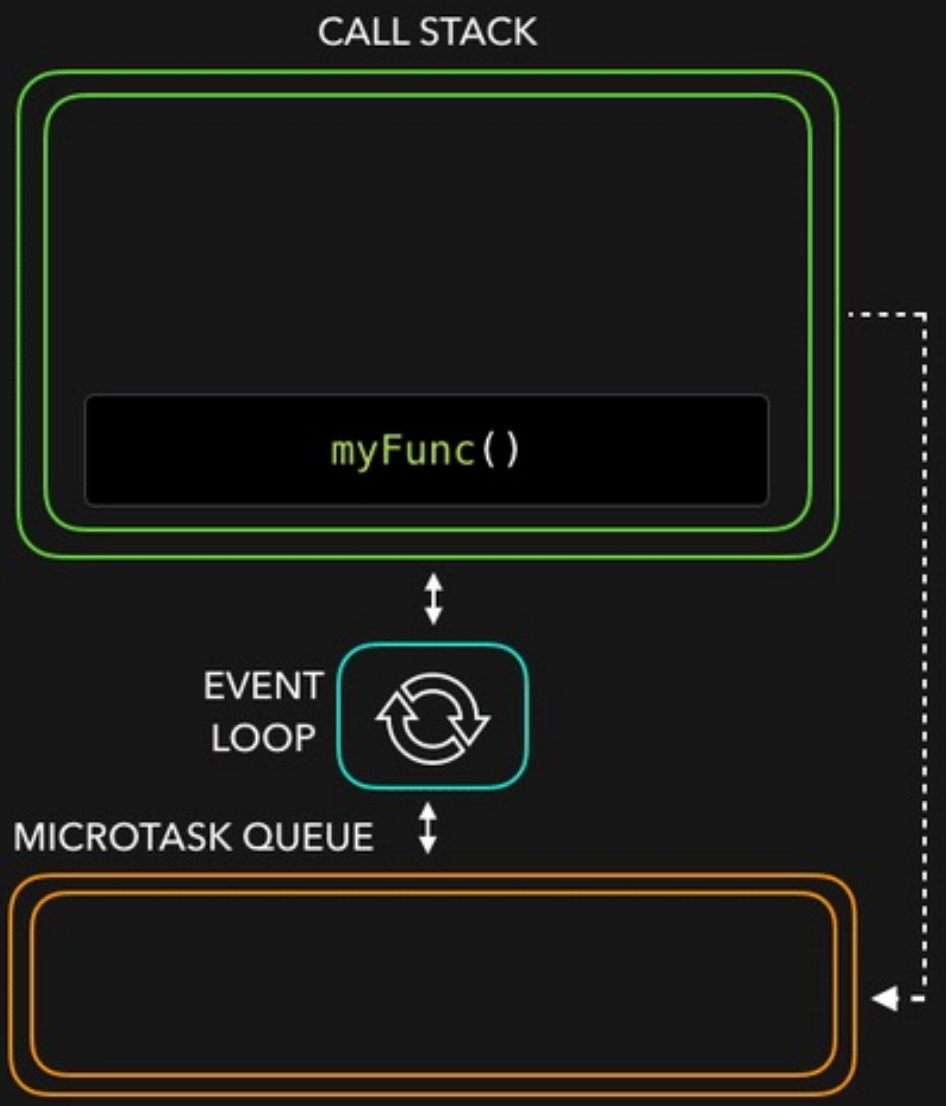
```
Before function!
```



```
const one = () => Promise.resolve('One!')

async function myFunc() {
  console.log('In function!')
  const res = await one()
  console.log(res)
}

console.log('Before function!')
myFunc();
console.log('After function!')
```

node

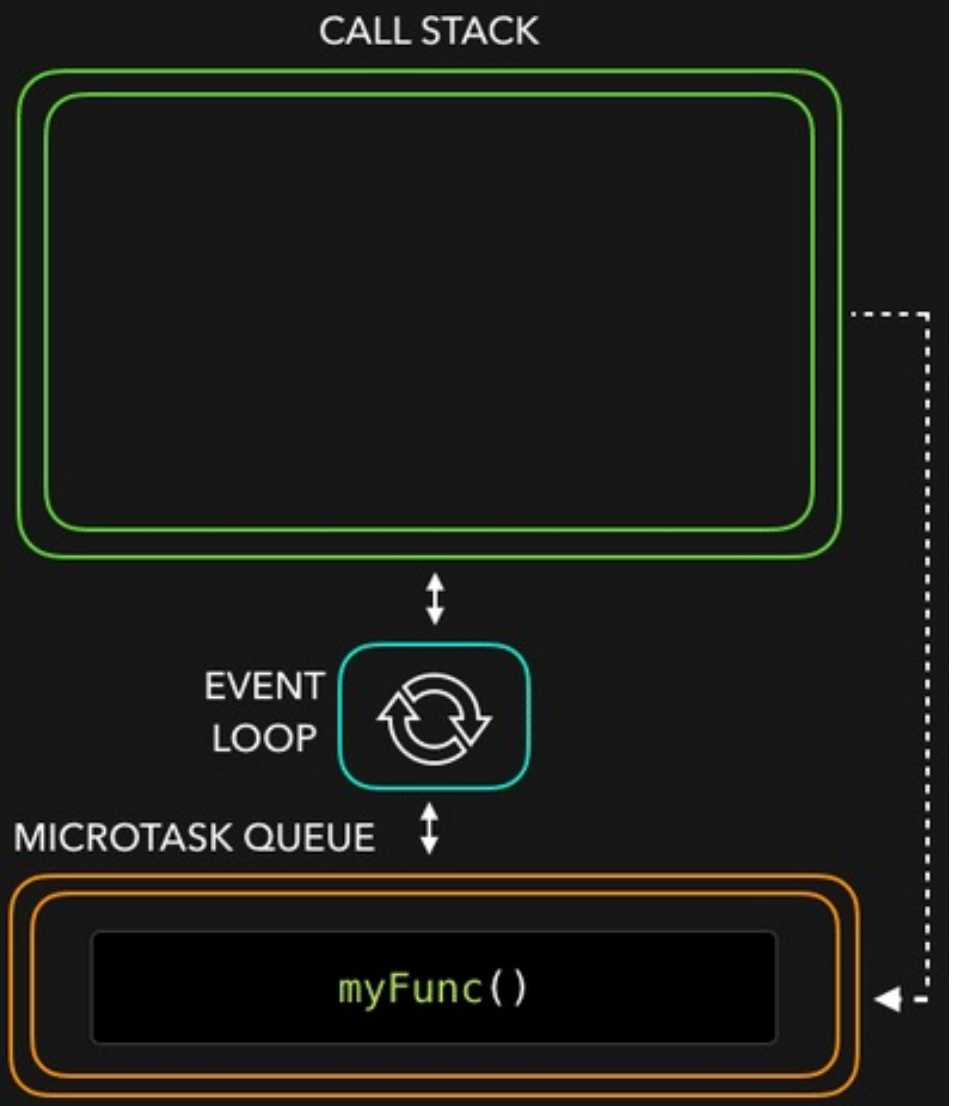
```
Before function!
In function!
```



```
const one = () => Promise.resolve('One!')

async function myFunc() {
  console.log('In function!')
  const res = await one()
  console.log(res)
}

console.log('Before function!')
myFunc();
console.log('After function!')
```

node

```
Before function!
In function!
```



```
const one = () => Promise.resolve('One!')

async function myFunc() {
  console.log('In function!')
  const res = await one()
  console.log(res)
}

console.log('Before function!')
myFunc();
console.log('After function!')
```

node

```
Before function!
In function!
```