이벤트 루프

Status	Not started
Assign	웹

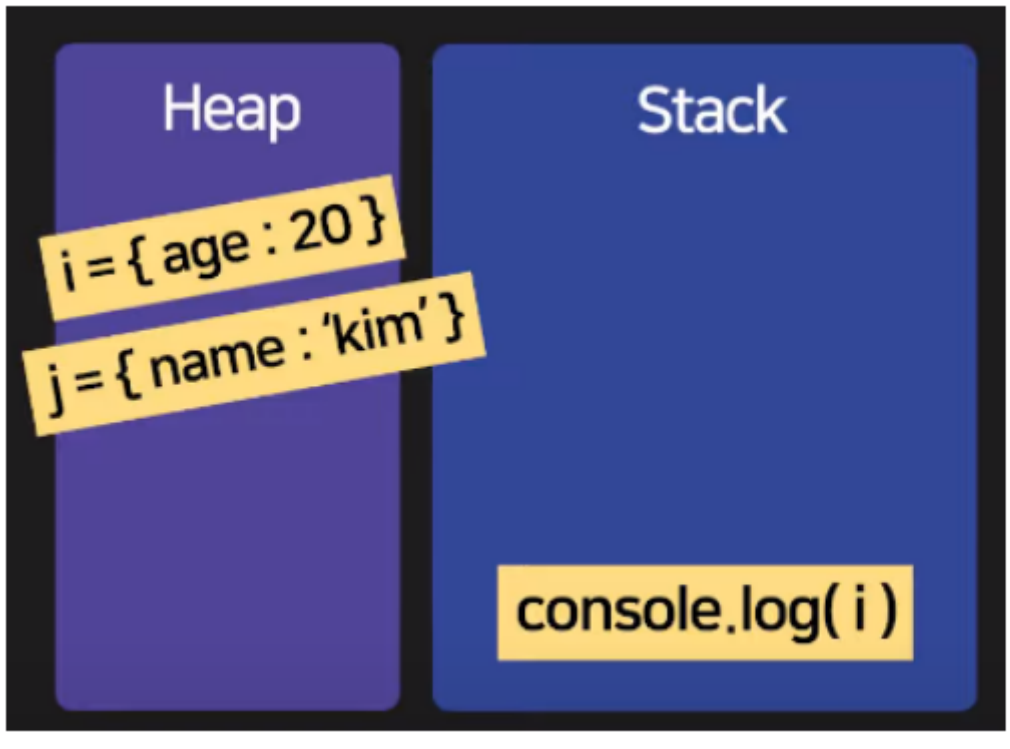
이벤트 루프

싱글스레드(Single Thread)

자바스크립트의 큰 특징 중 하나가 `Single Thread` 기반의 언어라는 점

스레드가 하나라는 말은 동시에 하나의 작업만을 처리할 수 있다는 말이 된다. 하지만 실제로 자바스크립트가 사용되는 환경을 생각해 보면 많은 작업이 동시에 처리되고 있는 걸 볼 수 있다. 예를 들면, 웹 브라우저는 HTTP요청이 진행되면서 다른 함수가 작동하는 걸 많이 봐왔다. Node.js에서는 동시에 여러 개의 HTTP요청을 처리하기도 한다.

자바스크립트 엔진



Memory Heap

프로그래밍을 할 때 선언된 변수, 함수가 담겨지는 메모리 할당이 일어나는 곳입니다. 메모리 저장소 입니다.

```
const i = { age : 20 };
const j = { name : 'kim'}
```

Call Stack

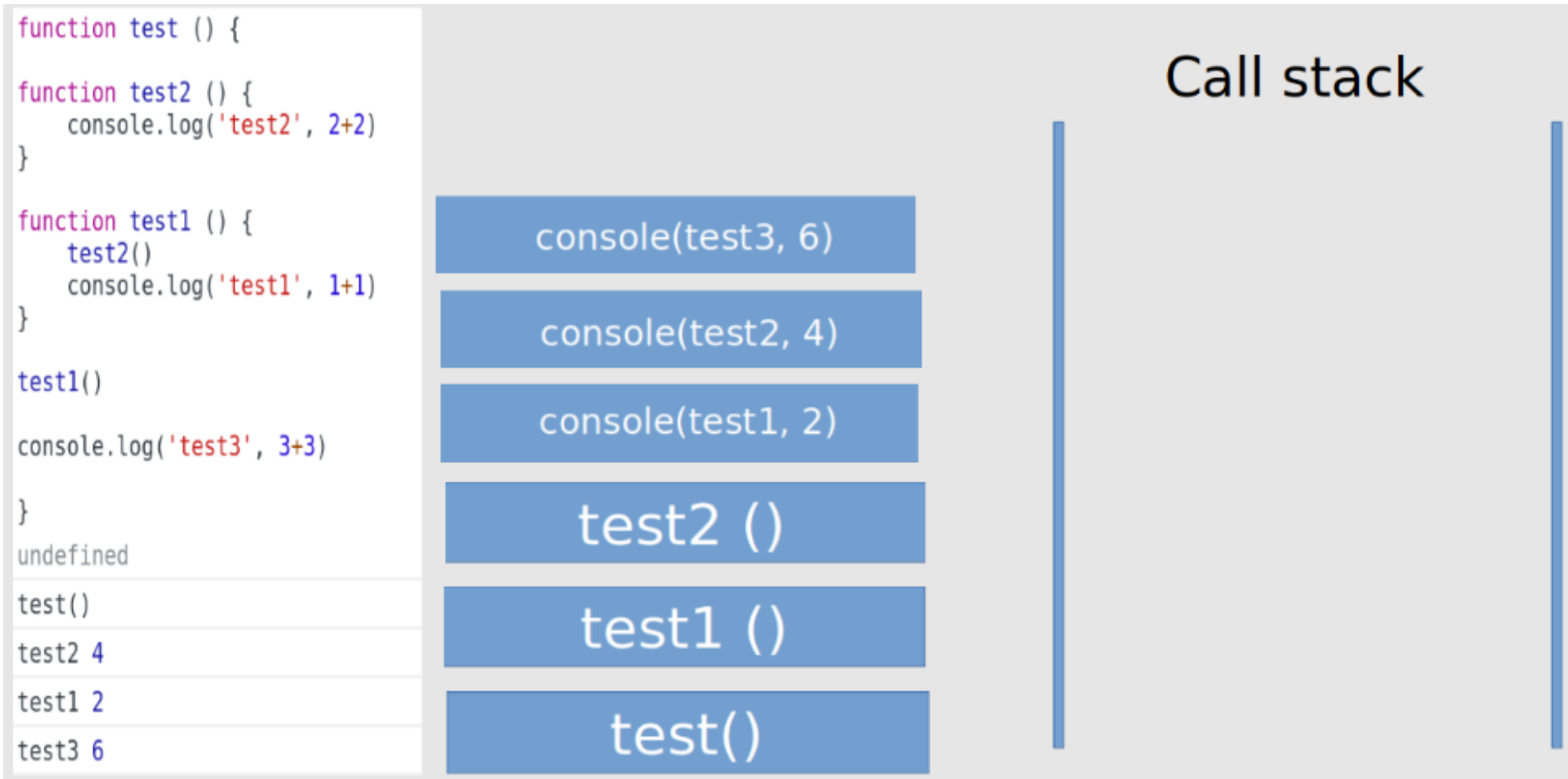
`call stack`은 자바스크립트에서 코드가 실행될 때 쌓이는 곳이고 stack의 형태로 코드를 싱행한다. Last In First Out, 마지막에 들어온게 먼저 나는 구조이다.

함수를 실행하게 되면 우선 stack에 쌓인 뒤 코드를 읽고 return코드를 만나면 함수가 종료가 되는데, 여기서 `call stack`이 비었다는 말로 설명 해놔야 합니다. 이 말은 더 이상 실행을 할 함수가 없다는 의미입니다.

자바스크립트에서 `call stack`은 하나만 있으며 자바스크립트언어는 단일 스레드 프로그래밍 언어이기 때문입니다.

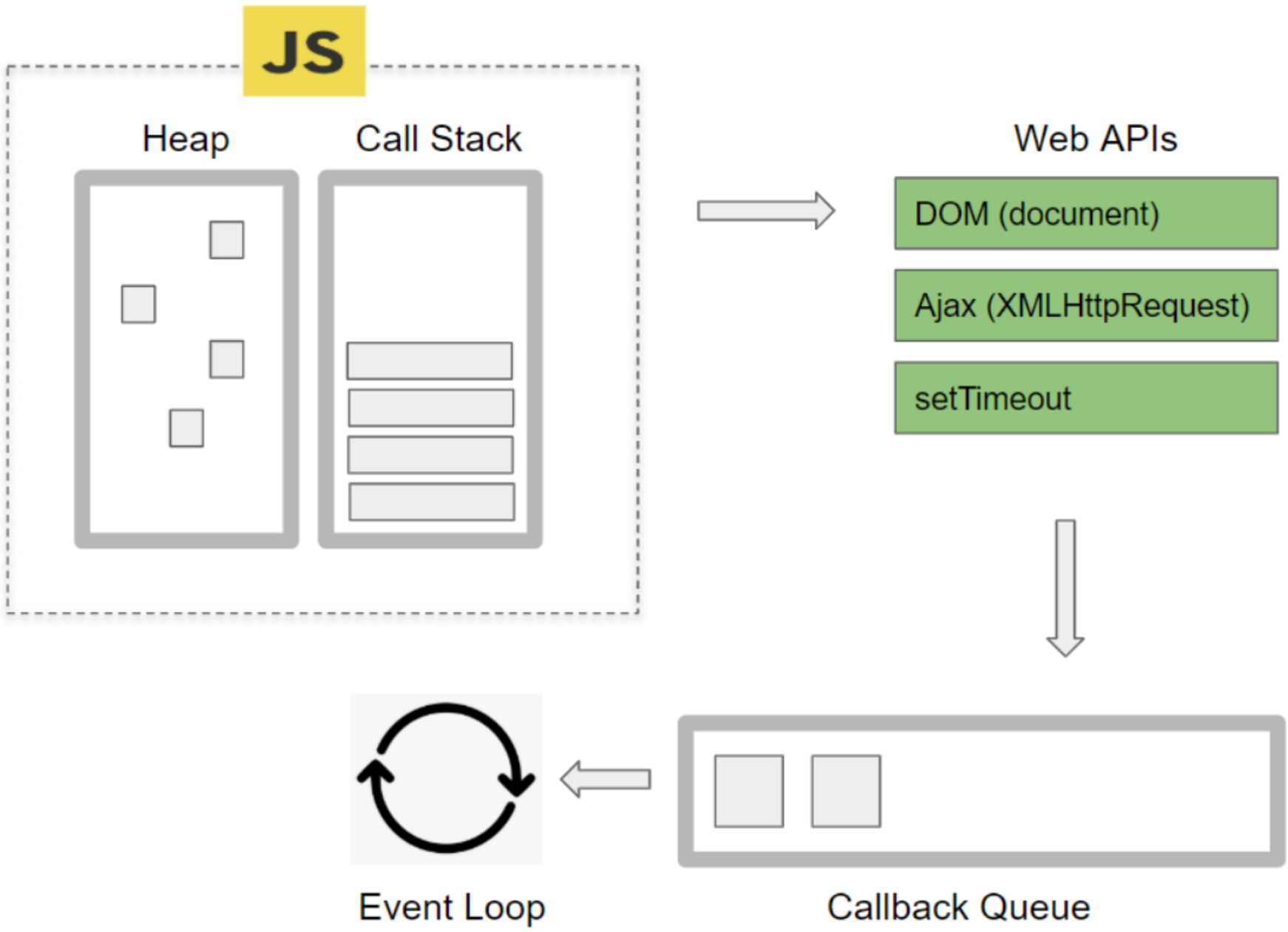
```
function returnAgeandName(i, j) {
  return i, j
}

returnAgeandName()
```



이렇게 자바스크립트언어는 call stack에 실행할 함수를 stack으로 쌓여 가장 마지막에 들어온 함수가 가장 먼저 실행하며 처음에 들어온 함수에 경우 가장 마지막에 실행된다는 점이다. 여기서 문제점은, 서버와의 통신을 할 때 비동기 호출을 이용한다.

이벤트 루프



웹 브라우저는 자바스크립트 엔진 말고도 WebAPI, 콜백큐(Callback Queue), 이벤트루프(Event Loop) 이런 것들을 가지고 있다.

WebAPIs

- 대표적으로 `HTTP요청메서드` `setTimeout`, `Dom메서드(addEventLisner(function()))`
- 이 메서드들은 비동기 메서드이기 때문에 작동이 마치면 콜백함수를 콜백 큐에 집어 넣습니다.

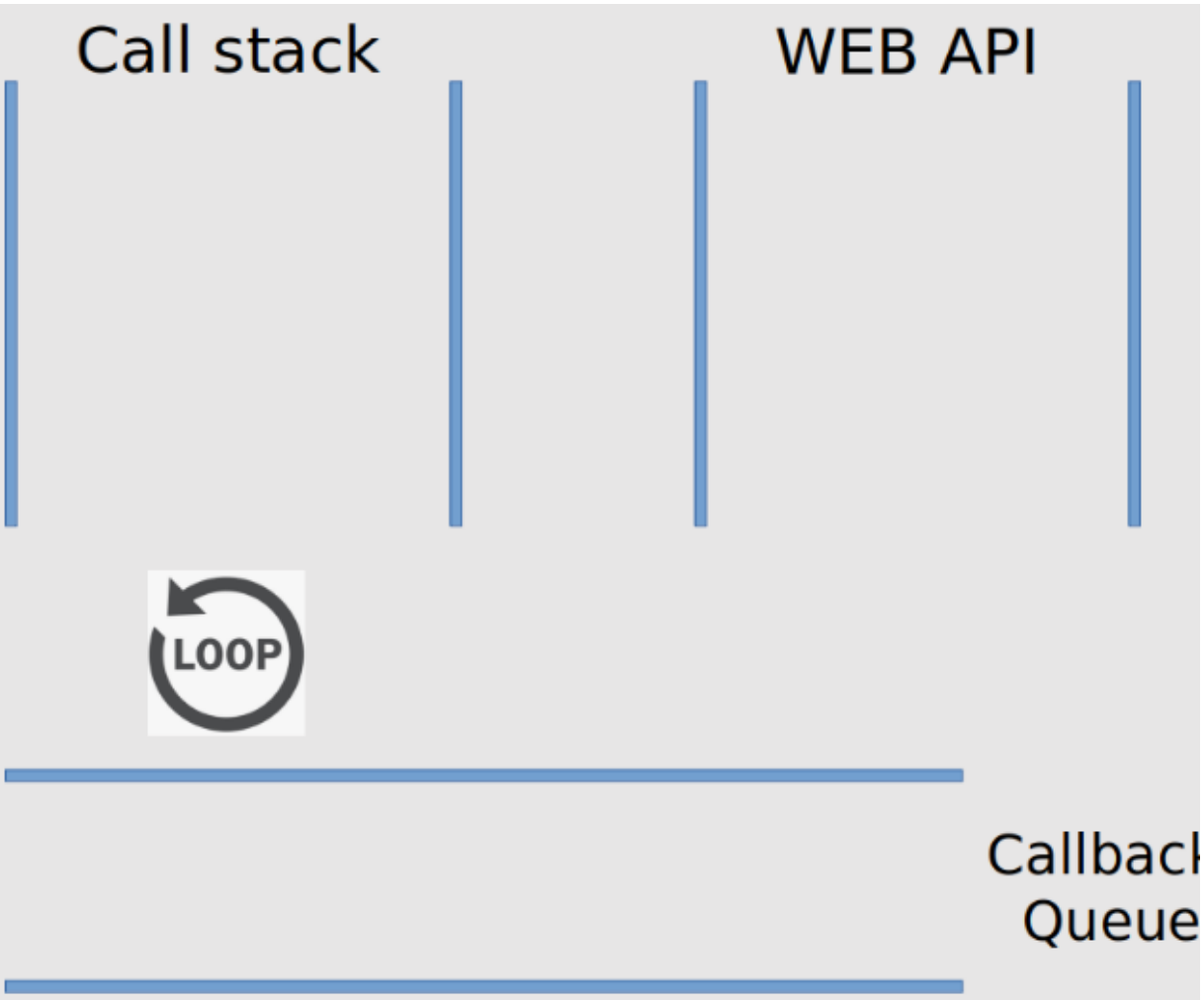
Callback Queue

- 콜백 큐에 들어간 콜백 함수들은 순차적으로 실행을 대기한다.

WebAPI & Callback Queue & EventLoop 동작 과정

```
function second() {
    setTimeout(function() {
        console.log('2+2')
    }, 2000);
}

function first () {
    console.log('1+1')
    second();
    console.log('3+3')
}
undefined
first ()
1+1
3+3
undefined
2+2
```



1. `first()` 함수가 실행 된다.
2. `console.log('1+1')` 이 실행된다.
3. `second()` 함수가 실행 되지만 비동기메서드인 setTimeout는 타이머를 webAPI에서 생성하게 된다. 이 타이머를 생성하는 걸로 setTimeout의 역할은 끝이 나고 callback함수가 callback Queue에 들어가며 이때 `이벤트루프`가 실행된다.
4. `console.log('3+3')` 이 실행된다.
5. `console.log('2+2')` 이 실행된다.

`이벤트루프`는 call stack과 callback queue를 계속 주시하고 있다. call stack이 비어 있으면, 먼저 들어온 순서대로 callback queue에 있는 콜백 함수들을 call stack으로 집어 넣습니다.

callback queue를 지켜보다가 콜백함수가 들어온 경우 call stack이 비어있는지 확인 후 비어있다면 call stack으로 옮겨주는 것을 이벤트 루프(**event loof)** 입니다

```
console.log('Start')

setTimeout(() => {
  console.log('TimeOut!')
}, 0)

Promise.resolve('Promise!')
.then(res => console.log(res))

console.log('End!')

// 출력 결과

Start!
End!
Promise!
TimeOut!
```

https://talkwithcode.tistory.com/89