

# Event Loop



J099 송진현

Event Loop


Callback Queue



1. 이벤트 루프란?
2. 이벤트 루프의 동작 과정
3. Q&A



## 1. 이벤트 루프란?



## 1. 이벤트 루프란?



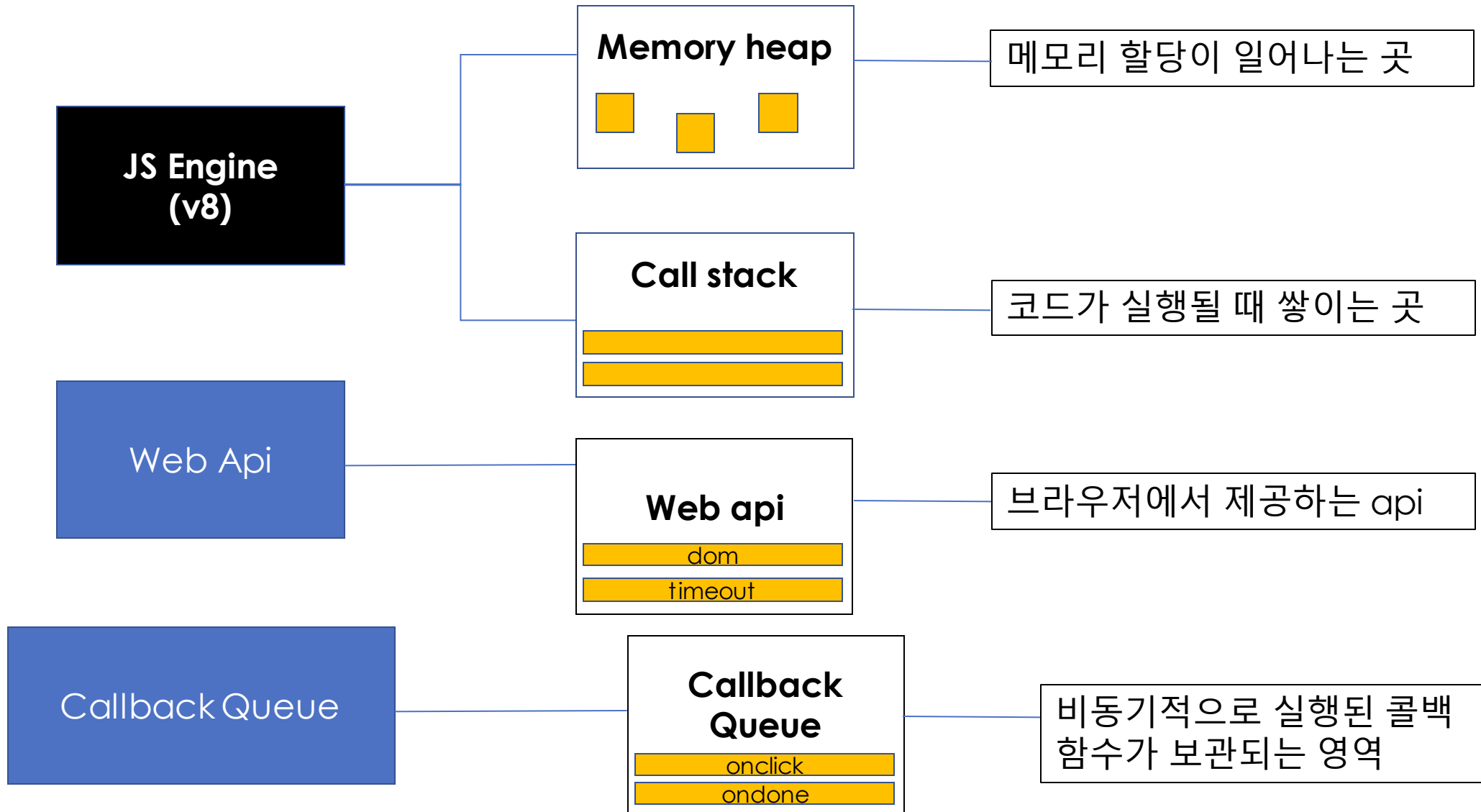
1. 자바스크립트는 단일스레드이기에 한 작업밖에 진행을 못하지만 많은 작업을 할 수 있도록 도와주는 장치
2. call stack이 다 비워지면 callback queue에 존재하는 함수를 하나씩 호출 스택으로 옮기는 역할을 하는 장치

## 1. 이벤트 루프란?



그러면 어떤 식으로 도와주지??

# 1. 이벤트 루프란?



## 2. 이벤트 루프의 동작과정



```
1 console.log("start");  
2 |  
3 setTimeout(function () {  
4 |   console.log("5초 후에");  
5 | }, 5000);  
6  
7 Promise.resolve().then(function () {  
8 |   console.log("promise1");  
9 | });  
10  
11 console.log("end");
```

## 2. 이벤트 루프의 동작과정



```
start  
end  
promise1  
5초 후에
```



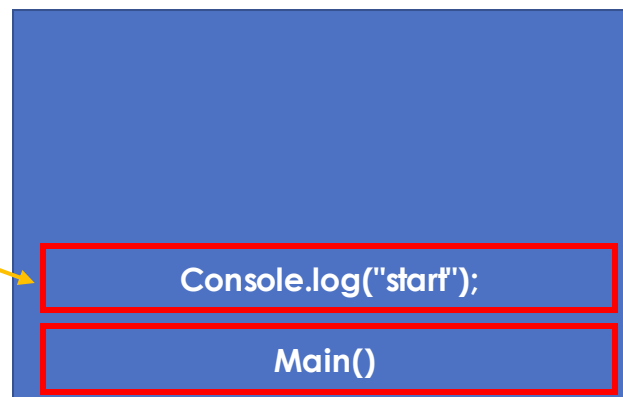
## 2. 이벤트 루프의 동작과정



Source Code

```
1 console.log("start");
2
3 setTimeout(function () {
4   console.log("5초 후에");
5 }, 5000);
6
7 Promise.resolve().then(function () {
8   console.log("promise1");
9 });
10
11 console.log("end");
```

Call Stack



Result

```
start
end
promise1
5초 후에
```

## 2. 이벤트 루프의 동작과정



Source Code

```
1 console.log("start");  
2  
3 setTimeout(function () {  
4   console.log("5초 후에");  
5 }, 5000);  
6  
7 Promise.resolve().then(function () {  
8   console.log("promise1");  
9 });  
10  
11 console.log("end");
```

Call Stack



Web API



Result

```
start  
end  
promise1  
5초 후에
```

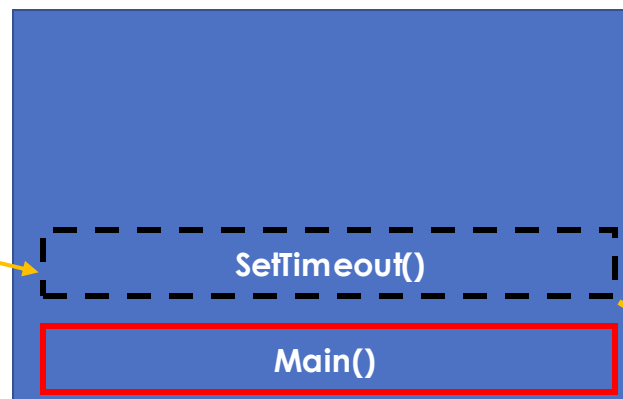
## 2. 이벤트 루프의 동작과정



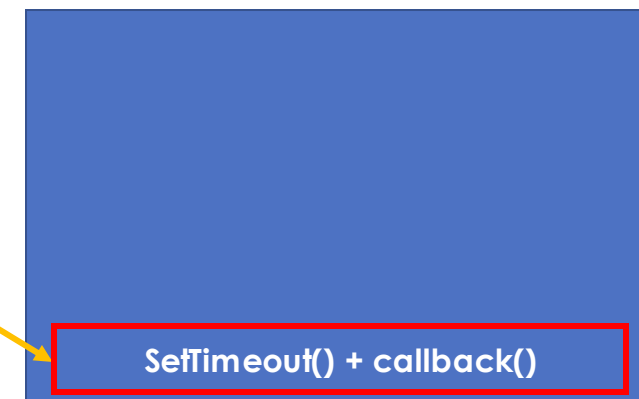
Source Code

```
1 console.log("start");  
2  
3 setTimeout(function () {  
4   console.log("5초 후에");  
5 }, 5000);  
6  
7 Promise.resolve().then(function () {  
8   console.log("promise1");  
9 });  
10  
11 console.log("end");
```

Call Stack



Web API



Result

```
start  
end  
promise1  
5초 후에
```

## 2. 이벤트 루프의 동작과정



Source Code

```
1 console.log("start");
2
3 setTimeout(function () {
4   console.log("5초 후에");
5 }, 5000);
6
7 Promise.resolve().then(function () {
8   console.log("promise1");
9 });
10
11 console.log("end");
```

Call Stack



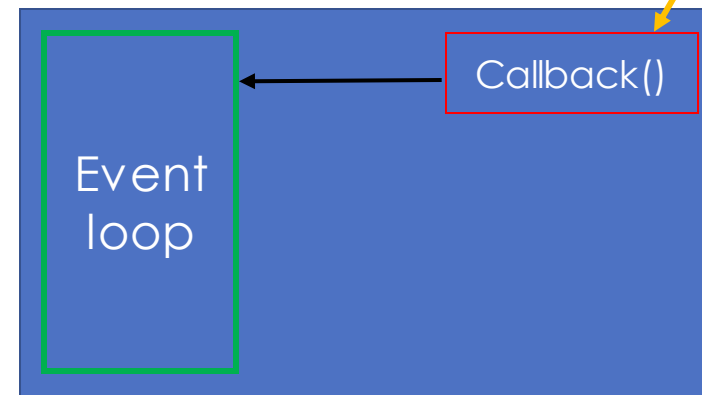
Web API



Result

```
start
end
promise1
5초 후에
```

Callback Queue



5초 후

Task queue

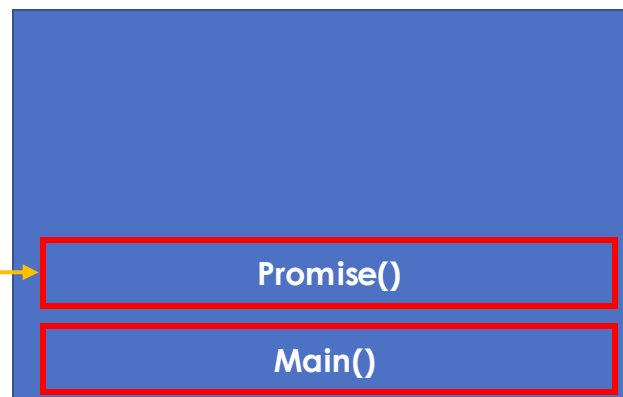
## 2. 이벤트 루프의 동작과정



Source Code

```
1 console.log("start");  
2 |  
3 setTimeout(function () {  
4   console.log("5초 후에");  
5 }, 5000);  
6 |  
7 Promise.resolve().then(function () {  
8   console.log("promise1");  
9 });  
10 |  
11 console.log("end");
```

Call Stack



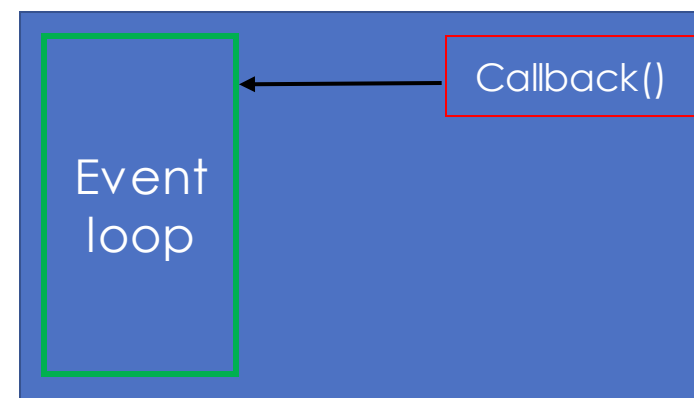
Web API



Result

```
start  
end  
promise1  
5초 후에
```

Callback Queue



Task queue

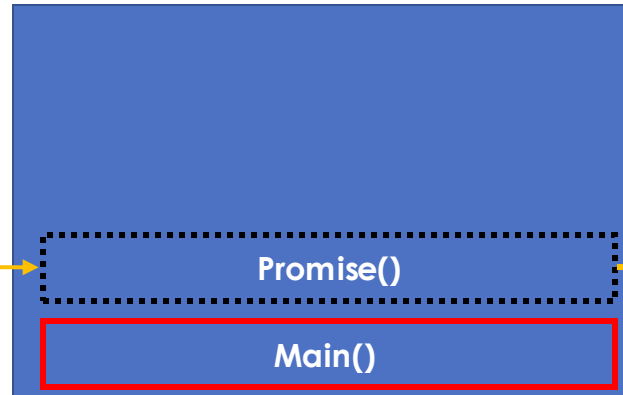
## 2. 이벤트 루프의 동작과정



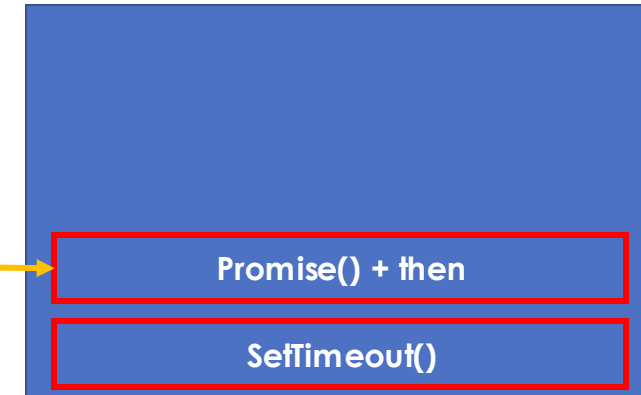
Source Code

```
1 console.log("start");
2
3 setTimeout(function () {
4   console.log("5초 후에");
5 }, 5000);
6
7 Promise.resolve().then(function () {
8   console.log("promise1");
9 });
10
11 console.log("end");
```

Call Stack



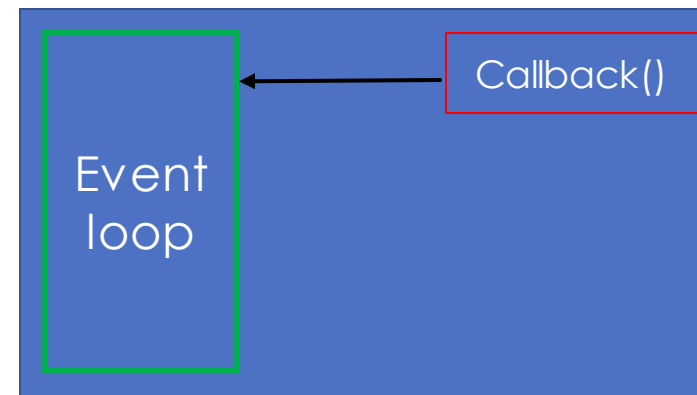
Web API



Result

```
start
end
promise1
5초 후에
```

Callback Queue



Task queue

## 2. 이벤트 루프의 동작과정



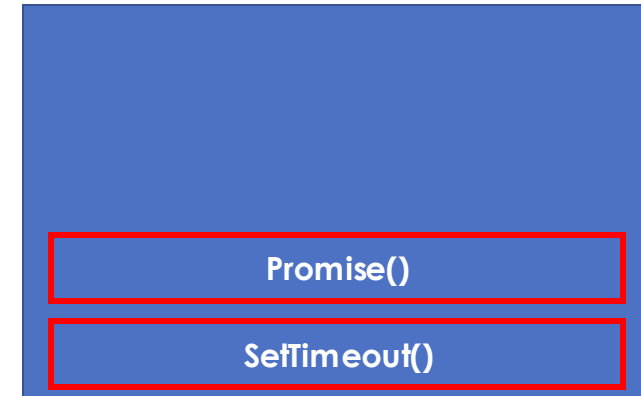
Source Code

```
1 console.log("start");
2
3 setTimeout(function () {
4   console.log("5초 후에");
5 }, 5000);
6
7 Promise.resolve().then(function () {
8   console.log("promise1");
9 });
10
11 console.log("end");
```

Call Stack



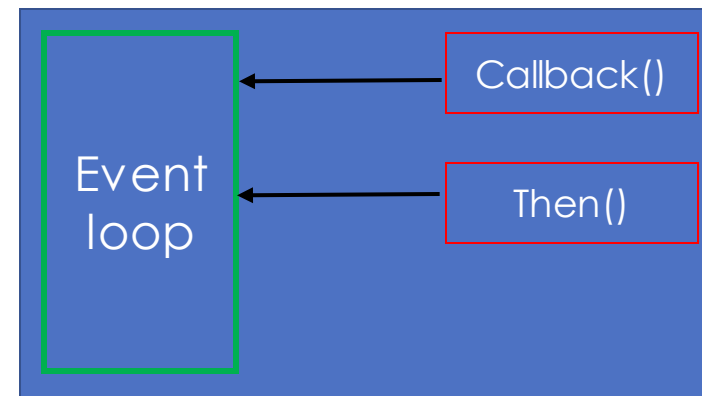
Web API



Result

```
start
end
promise1
5초 후에
```

Callback Queue



Task queue

Microtask queue

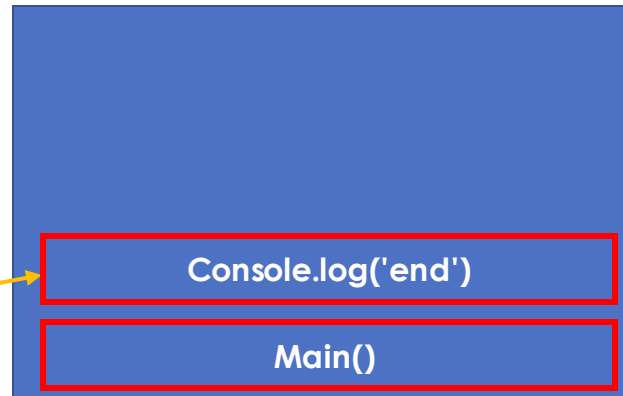
## 2. 이벤트 루프의 동작과정



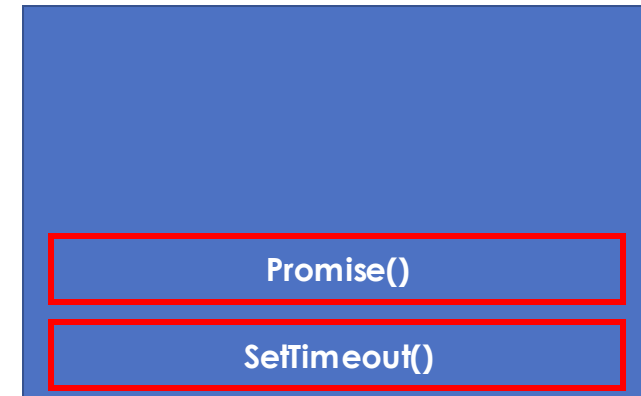
Source Code

```
1 console.log("start");
2 |
3 setTimeout(function () {
4 |   console.log("5초 후에");
5 | }, 5000);
6 |
7 Promise.resolve().then(function () {
8 |   console.log("promise1");
9 | });
10 |
11 console.log("end");
```

Call Stack



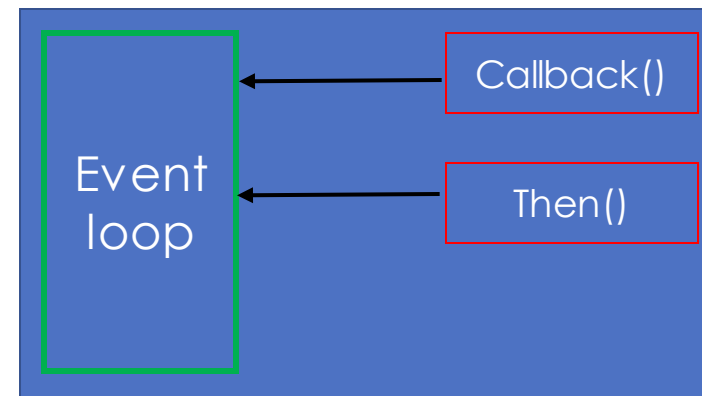
Web API



Result

```
start
end
promise1
5초 후에
```

Callback Queue



Task queue

Microtask queue



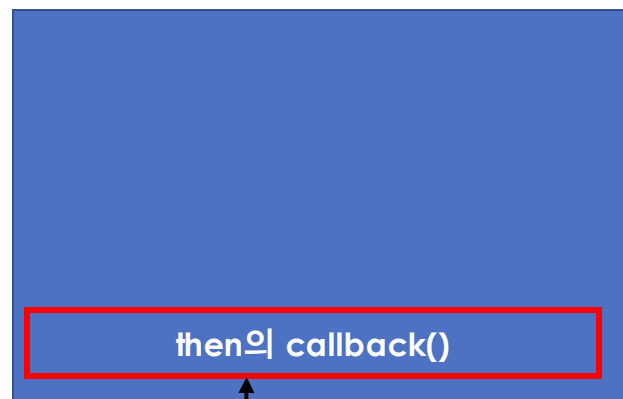
## 2. 이벤트 루프의 동작과정



### Source Code

```
1 console.log("start");
2
3 setTimeout(function () {
4   console.log("5초 후에");
5 }, 5000);
6
7 Promise.resolve().then(function () {
8   console.log("promise1");
9 });
10
11 console.log("end");
```

### Call Stack



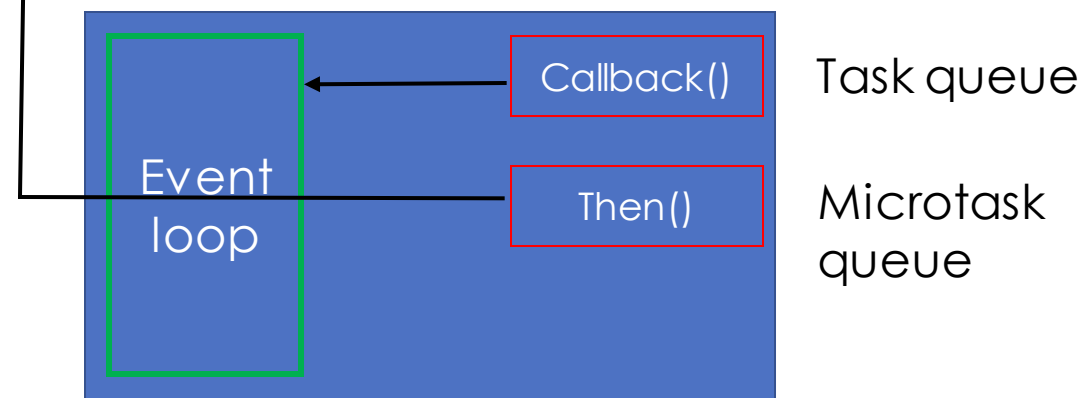
### Web API



### Result

```
start
end
promise1
5초 후에
```

### Callback Queue

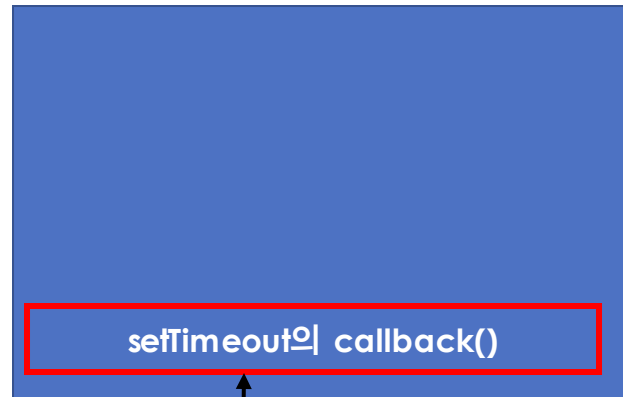


## 2. 이벤트 루프의 동작과정

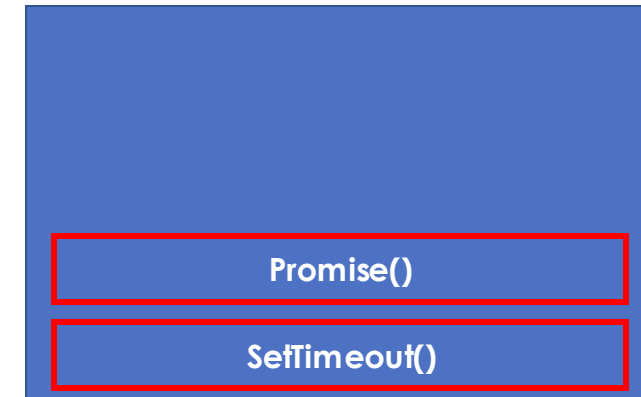
### Source Code

```
1 console.log("start");
2
3 setTimeout(function () {
4   console.log("5초 후에");
5 }, 5000);
6
7 Promise.resolve().then(function () {
8   console.log("promise1");
9 });
10
11 console.log("end");
```

### Call Stack



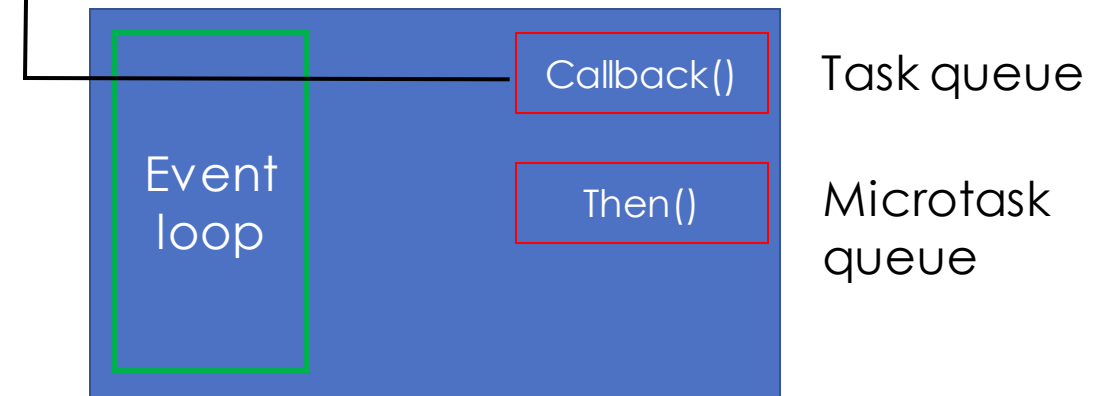
### Web API



### Result

```
start
end
promise1
5초 후에
```

### Callback Queue



## 2. 이벤트 루프의 동작과정



### 결론

1. 자바스크립트는 call stack으로만 봤을 때는 싱글 스레드이지만 이벤트 루프로 인해서 멀티 스레드와 같은 동작을 브라우저에서 할 수 있습니다.
2. 이벤트를 관리할 때 여러가지 종류의 큐가 있고, 큐마다 우선순위가 있다.  
(micro taskqueue > task queue) 참고로 requestanimation queue도 있습니다.
3. 이벤트 루프를 사용하는 것은 자바스크립트의 성능을 극대화할 수 있는 좋은 방법이라고 생각합니다.

## 2. 이벤트 루프의 동작과정



출처

<http://sculove.github.io/blog/2018/01/18/javascriptflow/>  
자바스크립트 비동기 처리 과정과 **RxJS Scheduler**

<https://meetup.toast.com/posts/89>  
자바스크립트와 이벤트 루프

### 3. Q&A



3. Q&A

다음은 브라우저