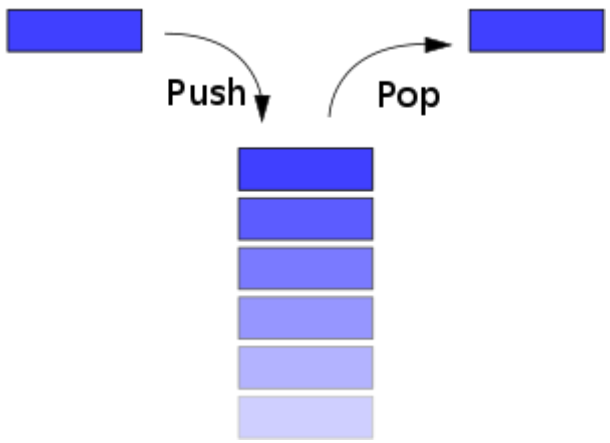
JavaScript 101 - (10) 이벤트 루프와 비동기 동작

▼ 알아두기

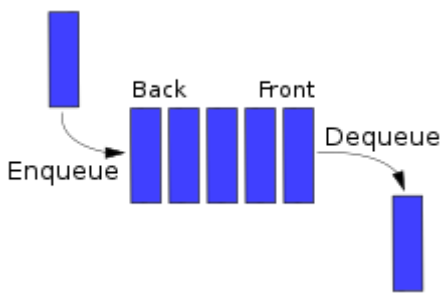
Stack



스택(Stack)은 후입선출(LIFO, Last-In-First-Out) 구조를 가지고 있는 자료구조입니다. 이 구조에서는 가장 마지막에 입력된 데이터가 가장 먼저 출력됩니다. 스택의 가장 위에 있는 항목을 가리키는 포인터를 스택 포인터 또는 top이라고 합니다.

스택은 주로 함수 호출, 재귀 알고리즘, 괄호 검사 등에서 사용됩니다. 예를 들어, 함수를 호출하면 함수 호출 스택에 새로운 스택 프레임이 추가됩니다. 함수가 반환되면 해당 스택 프레임이 스택에서 제거됩니다.

큐



큐(Queue)는 선입선출(FIFO, First-In-First-Out) 구조를 가지고 있는 자료구조입니다. 이 구조에서는 가장 먼저 입력된 데이터가 가장 먼저 출력됩니다. 큐는 주로 데이터가 입력된 순서대로 처리해야 할 때 사용됩니다. 예를 들어, 작업 큐에서는 가장 먼저 들어온 작업부터 순서대로 처리됩니다.

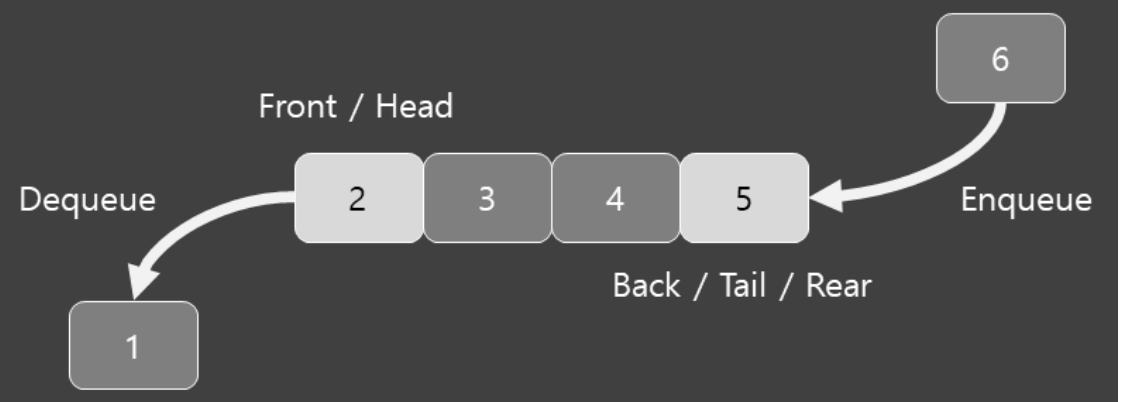
큐는 대기열, 버퍼, 프린터 스풀러 등에서 사용됩니다. 예를 들어, 프린터 스풀러에서는 출력할 문서가 들어오면 큐에 추가되고, 프린터가 이전에 출력한 문서를 모두 출력한 후에 큐에서 다음 문서를 꺼내어 출력합니다.

▼ 그림으로 살펴보는 이벤트루프

[복습] Queue – FIFO 자료구조

FIFO

- First In First Out (선입선출)



[복습] JavaScript의 비동기

JavaScript는

- Single-threaded (싱글 스레드인)
- Never blocking (논 블로킹 방식의)
- Asynchronous (비동기적인)
- Concurrency (동시성의)

언어이다.

[복습] Synchronous vs Asynchronous

Synchronous (동기)

- 순서대로 실행됨
- Call stack에서 코드가 순차적으로 실행될 때
- ex) 기상 -> 세면 -> 출근 -> 퇴근 -> 마트가서 장을 봄 -> 집에서 요리

Asynchronous (비동기)

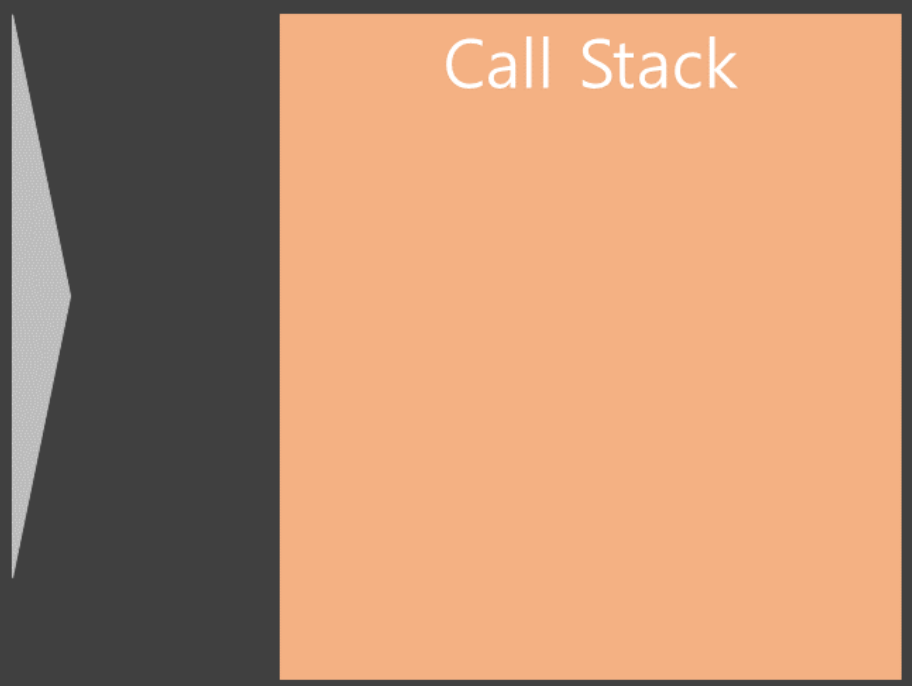
- 동시에 실행되는 것처럼 보임
- 타이머 또는 API를 호출하는 경우
- ex) 눈을 떴는데 알람이 울림, 요기요에서 음식 주문 + B마트에서 음료 주문

JavaScript의 Event Loop: Call Stack

```
function logger(message) {
  console.log(message);
}

function main() {
  logger("a");
}

main();
```

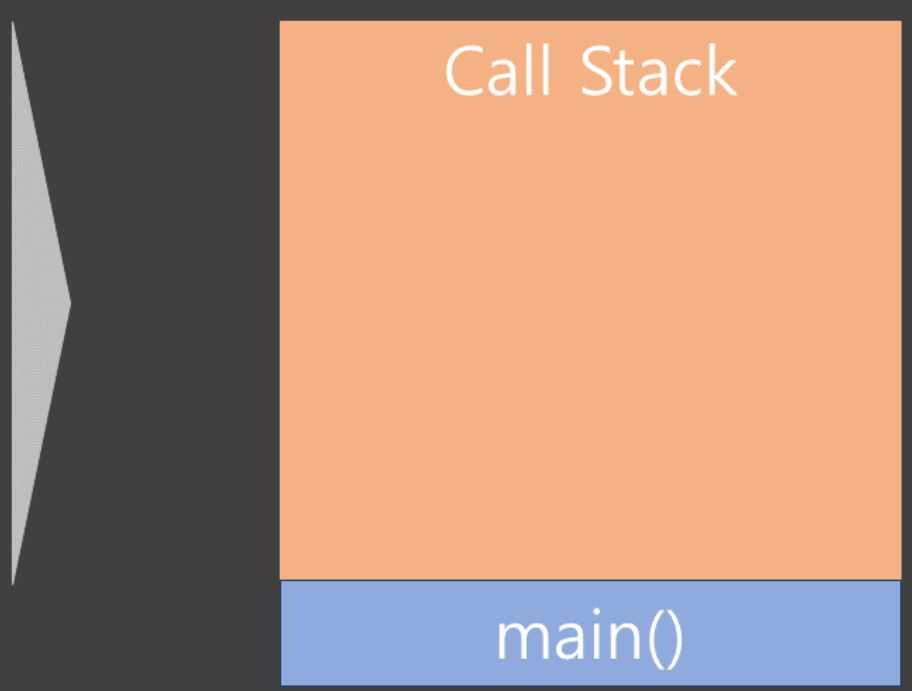
JavaScript의 Event Loop: Call Stack

```
function logger(message) {
  console.log(message);
}

function main() {
  logger("a");
}

main();
```



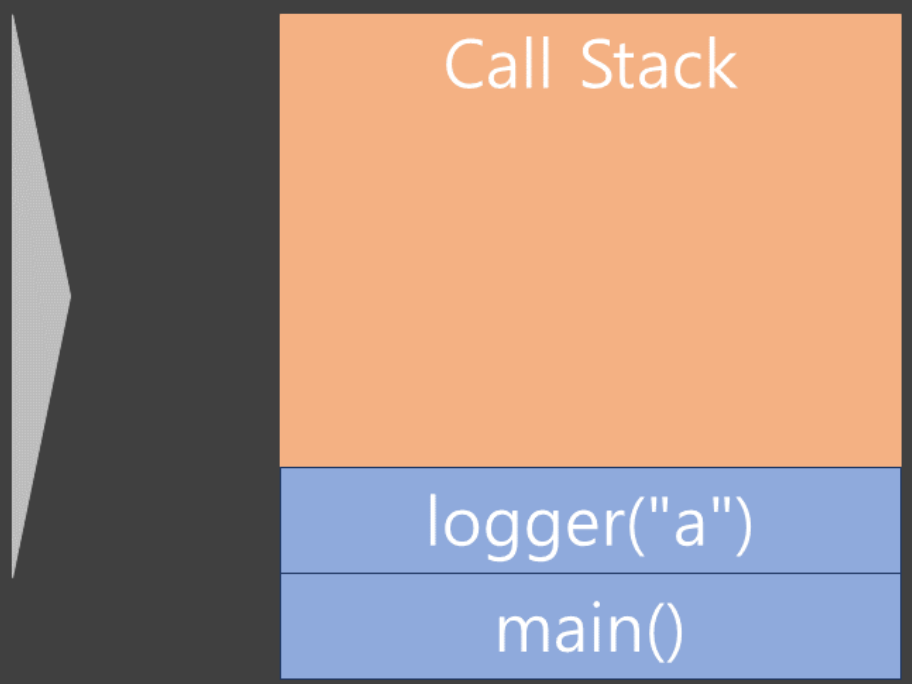
JavaScript의 Event Loop: Call Stack

```
function logger(message) {
  console.log(message);
}

function main() {
  logger("a");
}

main();
```



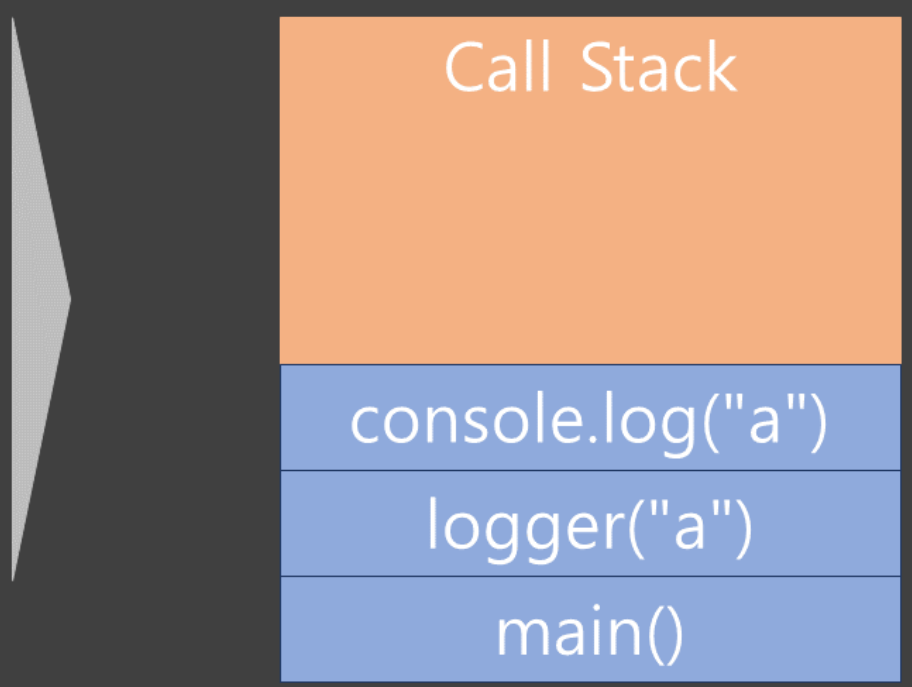
JavaScript의 Event Loop: Call Stack

```
function logger(message) {
  console.log(message);
}

function main() {
  logger("a");
}

main();
```

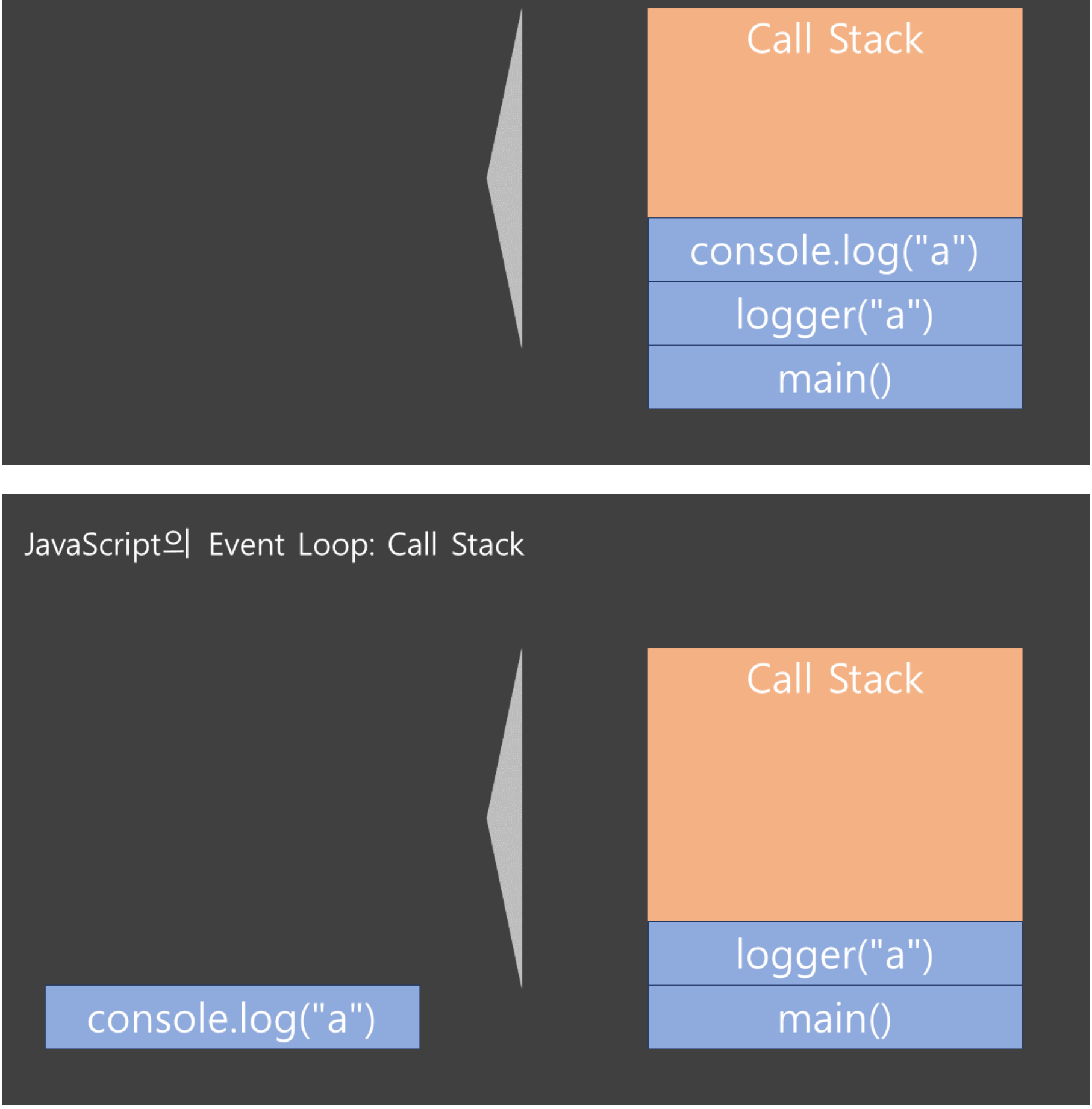


JavaScript의 Event Loop: Call Stack

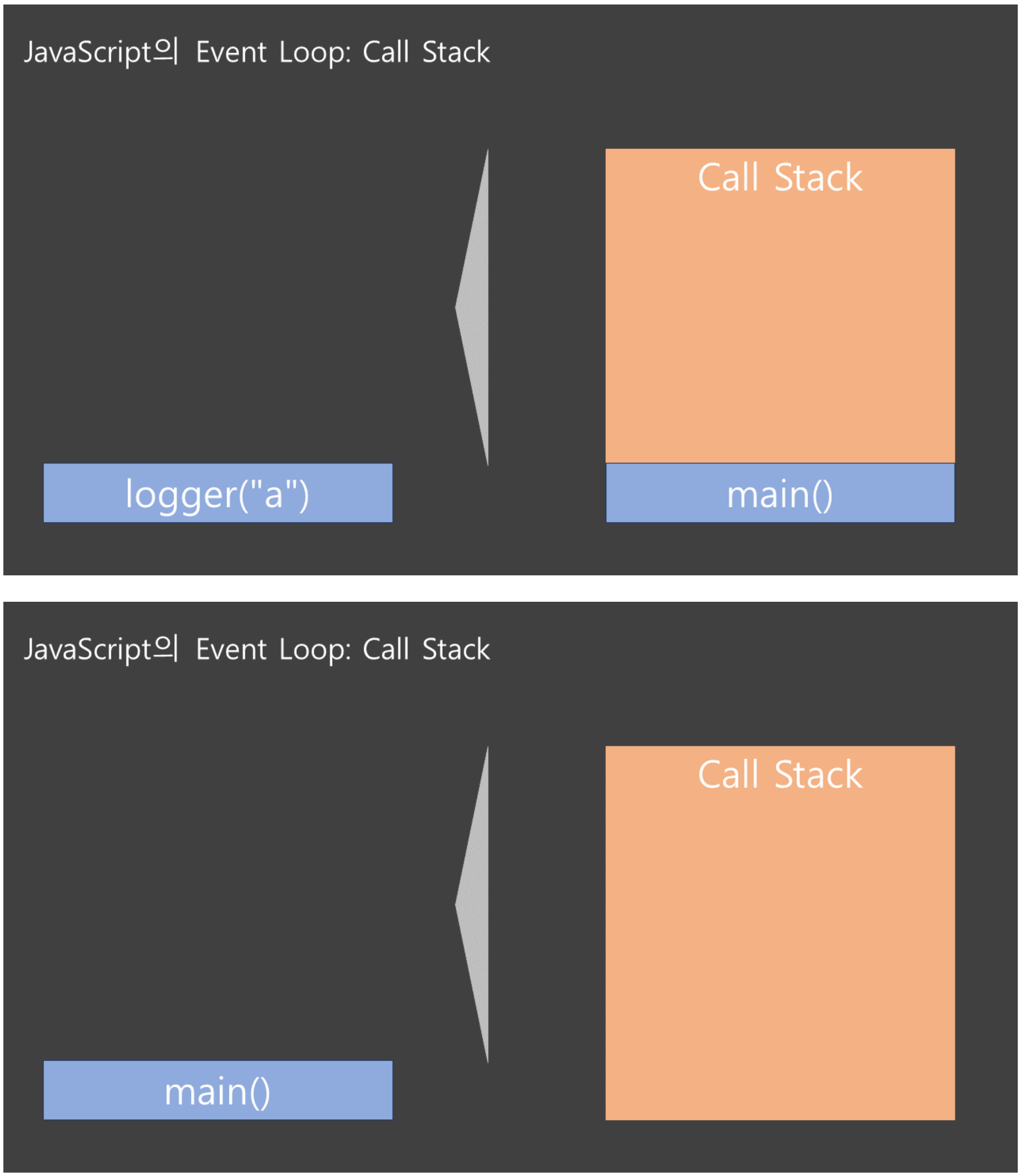
```
function logger(message) {
  console.log(message);
}

function main() {
  logger("a");
}

main();
```



JavaScript의 Event Loop: Call Stack

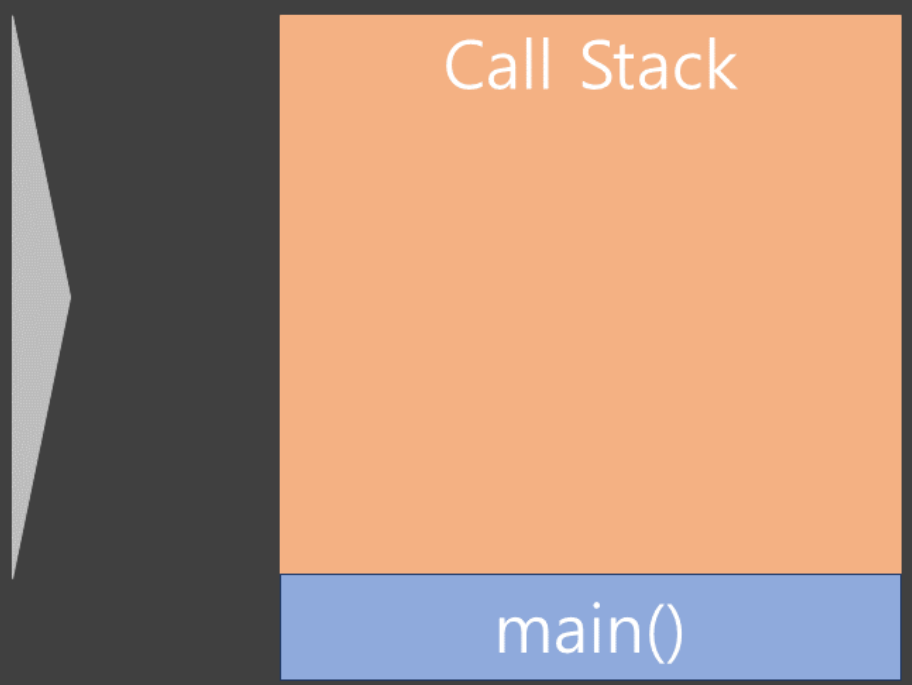


JavaScript의 Event Loop: Call Stack


JavaScript의 Event Loop: Call Stack
Call Stack
logger("a")
main()
JavaScript의 Event Loop: Call Stack
Call Stack
main()

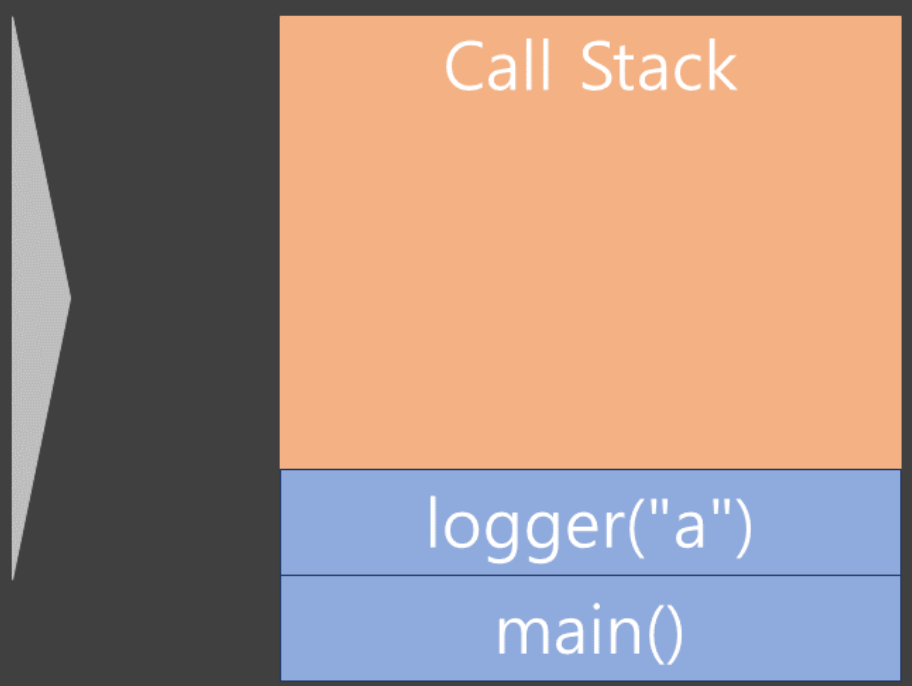
JavaScript의 Event Loop: Asynchronous

```
let timeId;

function logger(message) {
  console.log(message);
}

function main() {
  timeId = setTimeout(() => {
    logger(timeId);
  }, 1000);
  logger("a");
}

main();
```



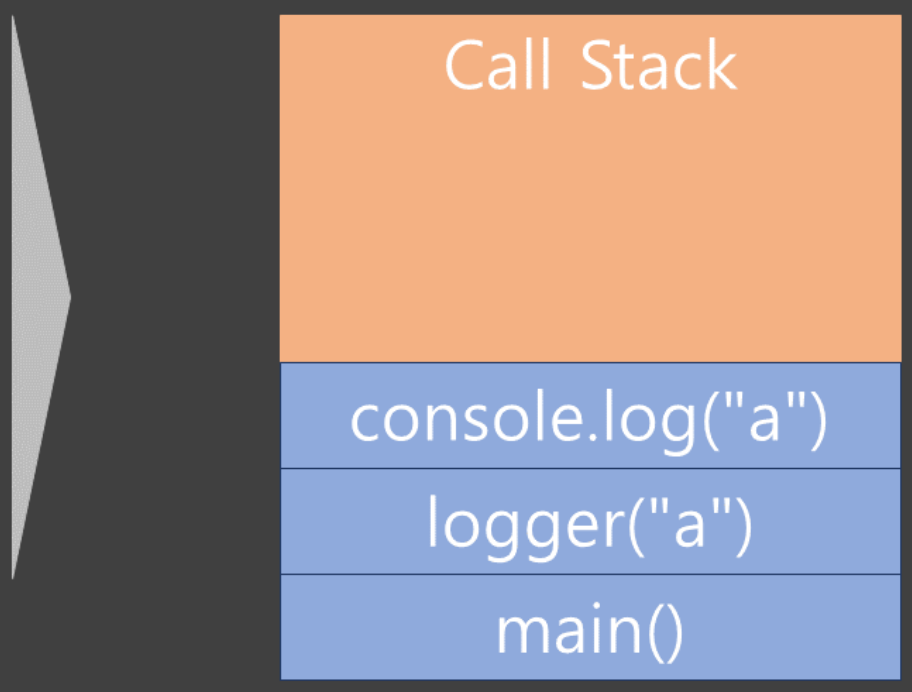
JavaScript의 Event Loop: Asynchronous

```
let timeId;

function logger(message) {
  console.log(message);
}

function main() {
  timeId = setTimeout(() => {
    logger(timeId);
  }, 1000);
  logger("a");
}

main();
```

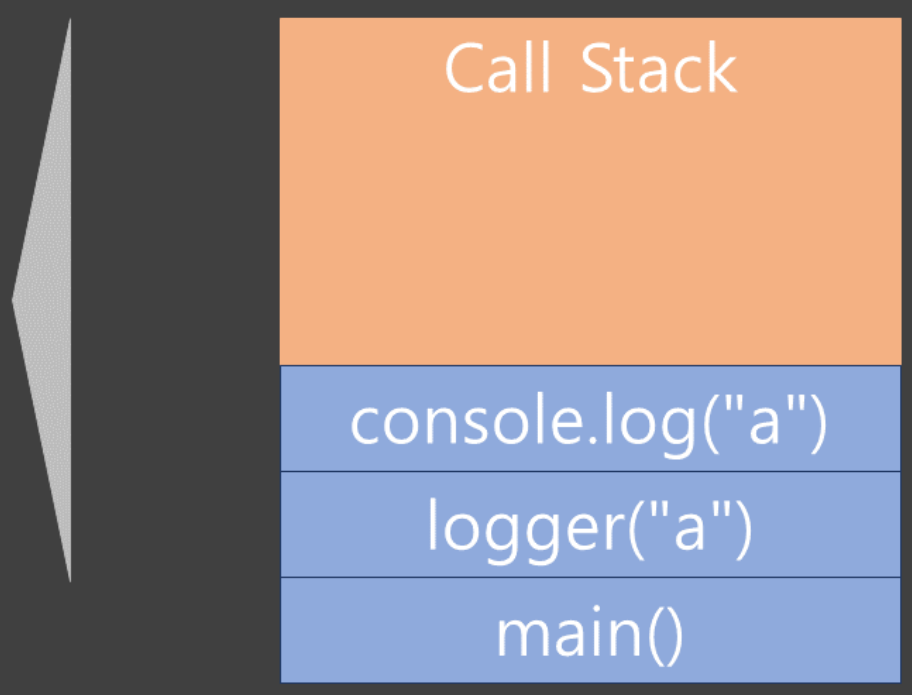


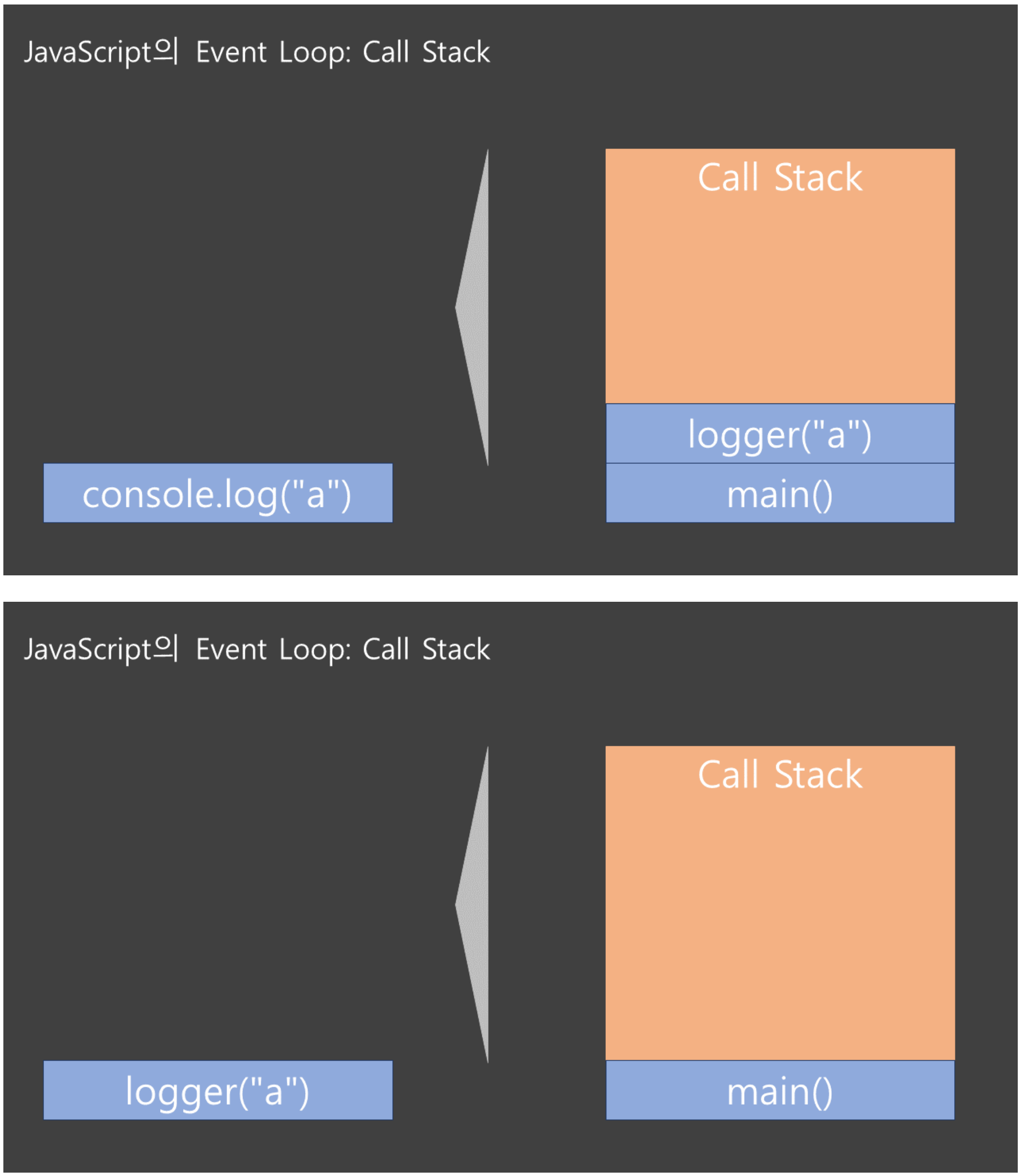
JavaScript의 Event Loop: Asynchronous

```
let timeId;

function logger(message) {
  console.log(message);
}

function main() {
  timeId = setTimeout(() => {
    logger(timeId);
  }, 1000);
  logger("a");
}

main();
```

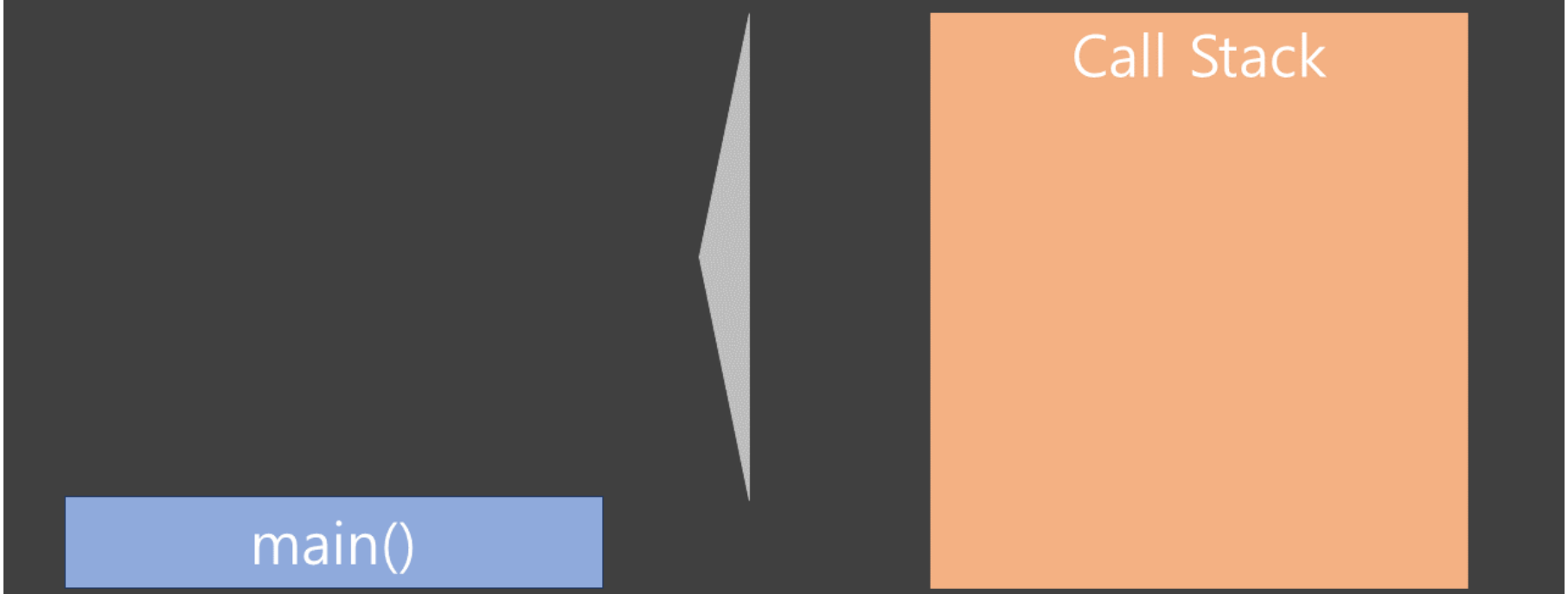


JavaScript의 Event Loop: Call Stack

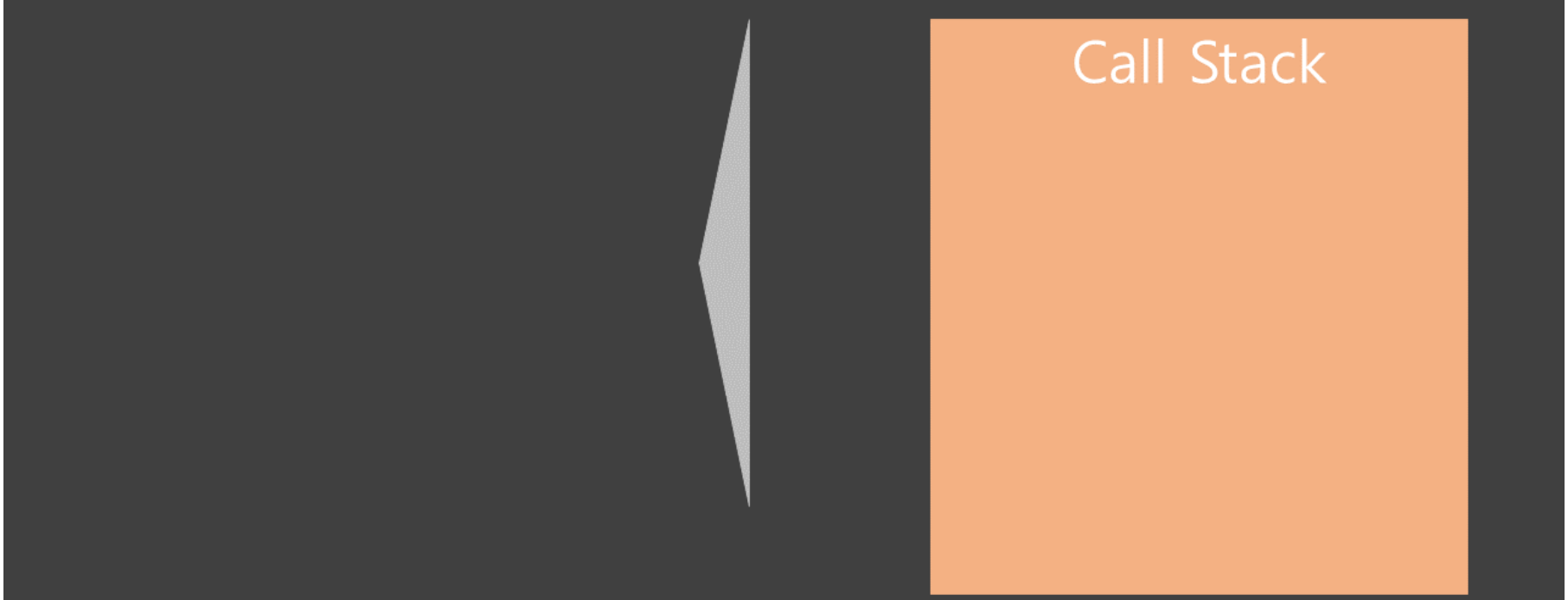



JavaScript의 Event Loop: Call Stack
Call Stack
logger("a")
main()
console.log("a")
JavaScript의 Event Loop: Call Stack
Call Stack
main()
logger("a")

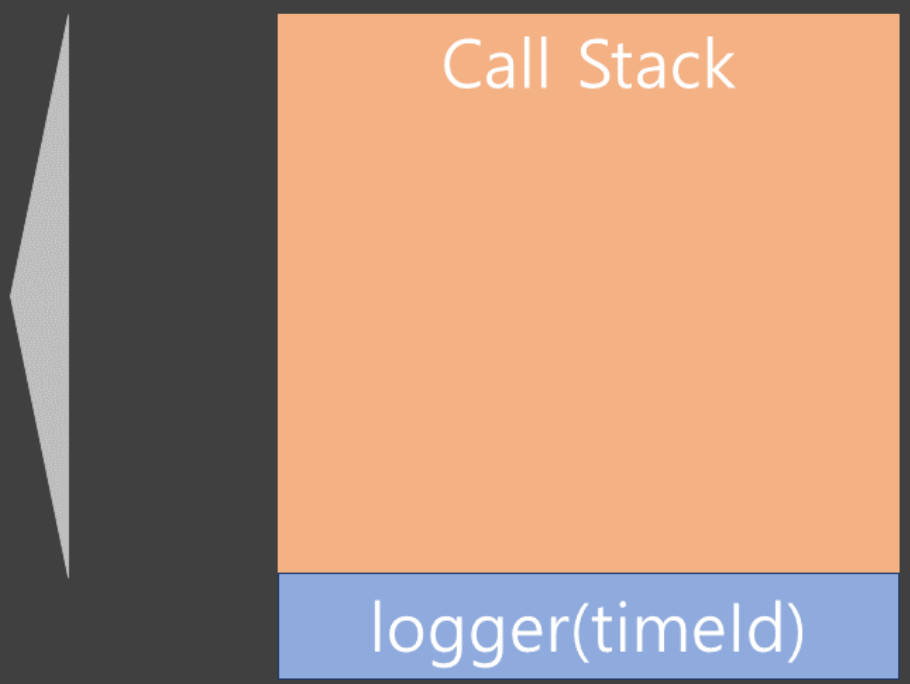
JavaScript의 Event Loop: Call Stack



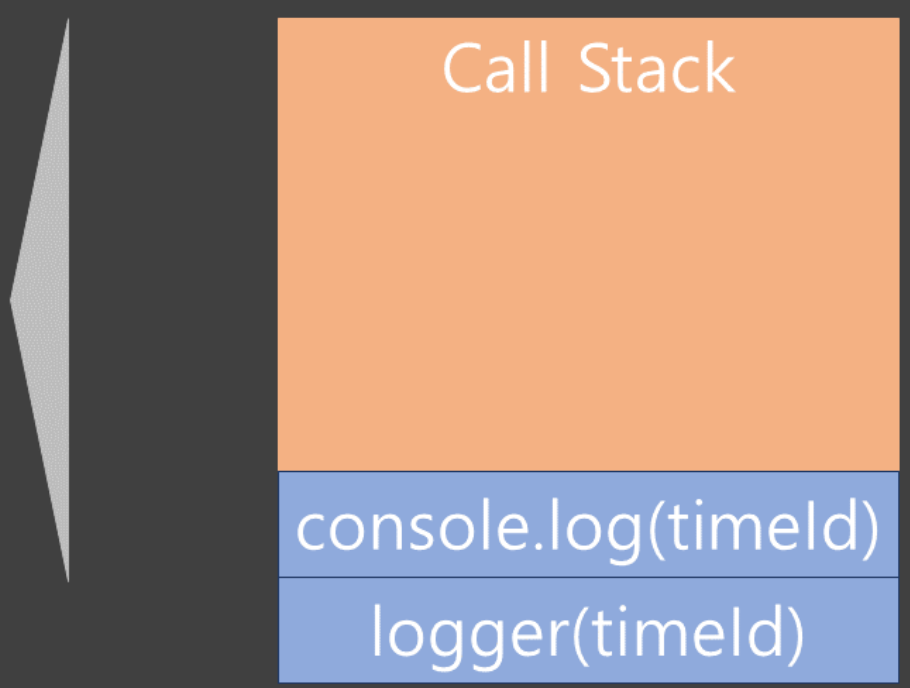
JavaScript의 Event Loop: Call Stack

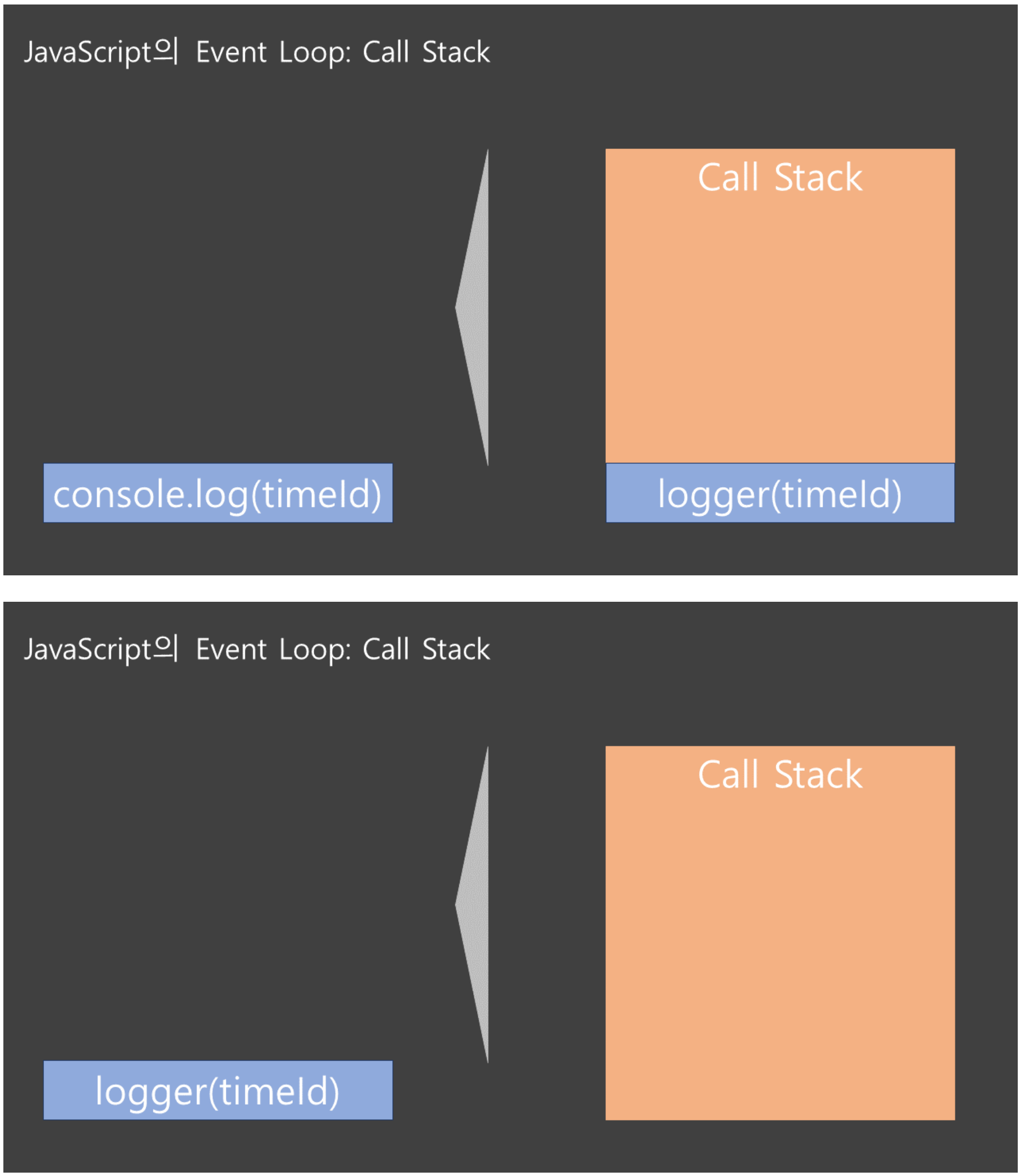


JavaScript의 Event Loop: Call Stack

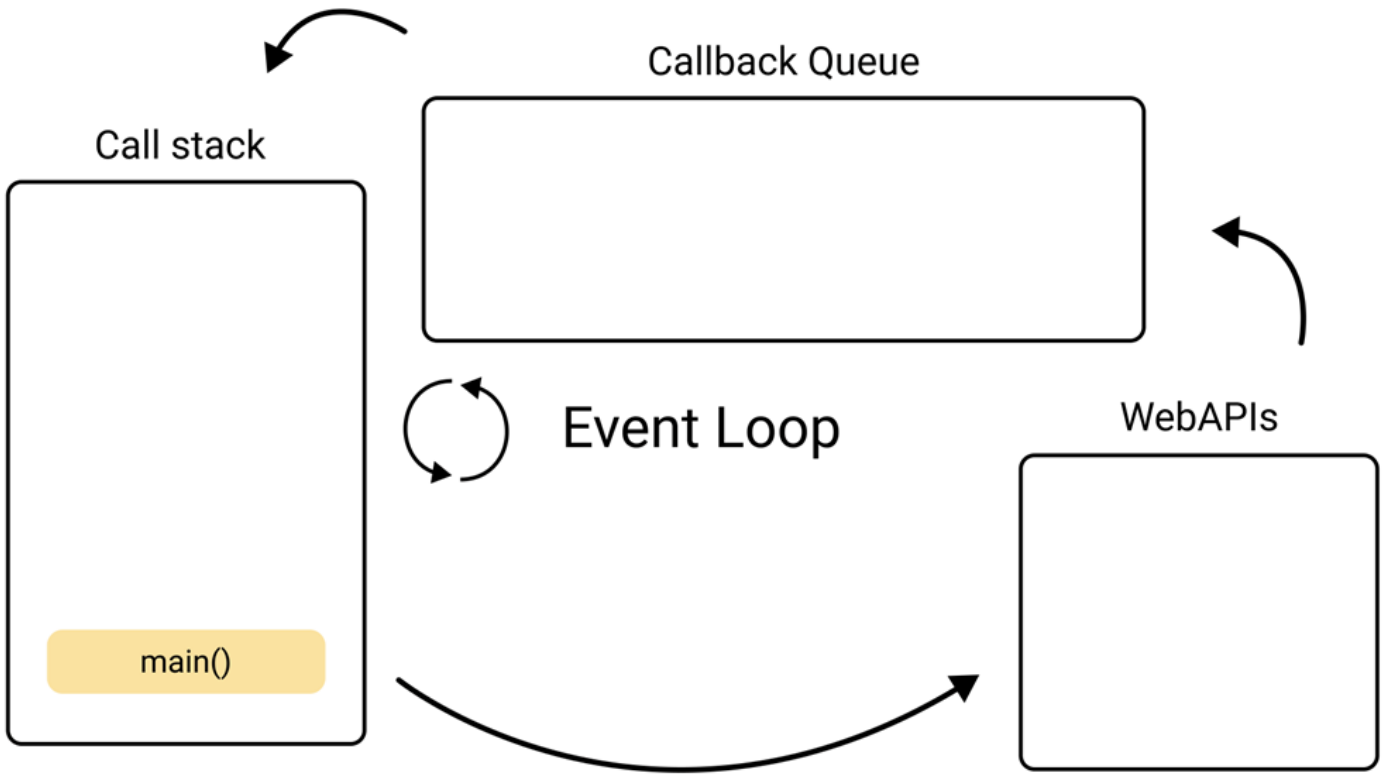


JavaScript의 Event Loop: Call Stack

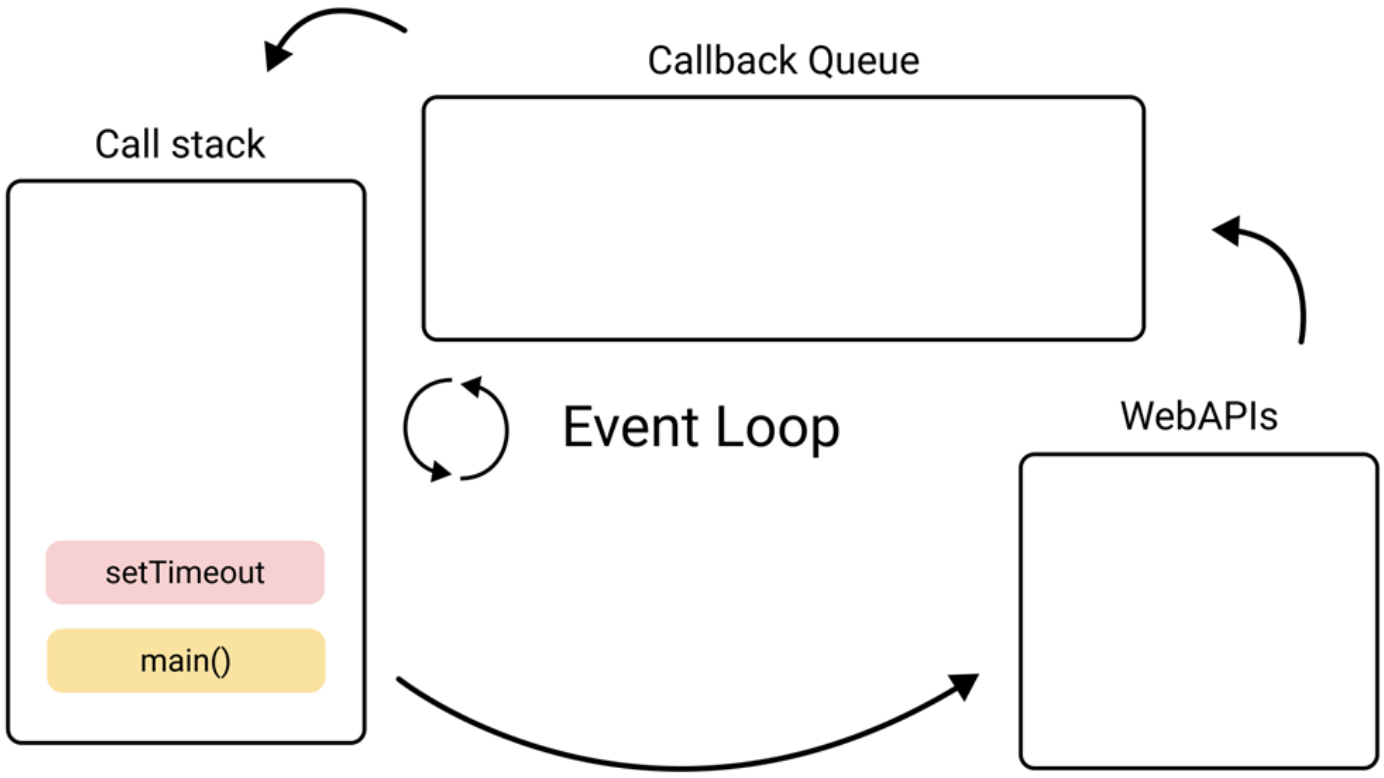



JavaScript의 Event Loop: Call Stack
Call Stack
console.log(timeId)
logger(timeId)
JavaScript의 Event Loop: Call Stack
Call Stack
logger(timeId)

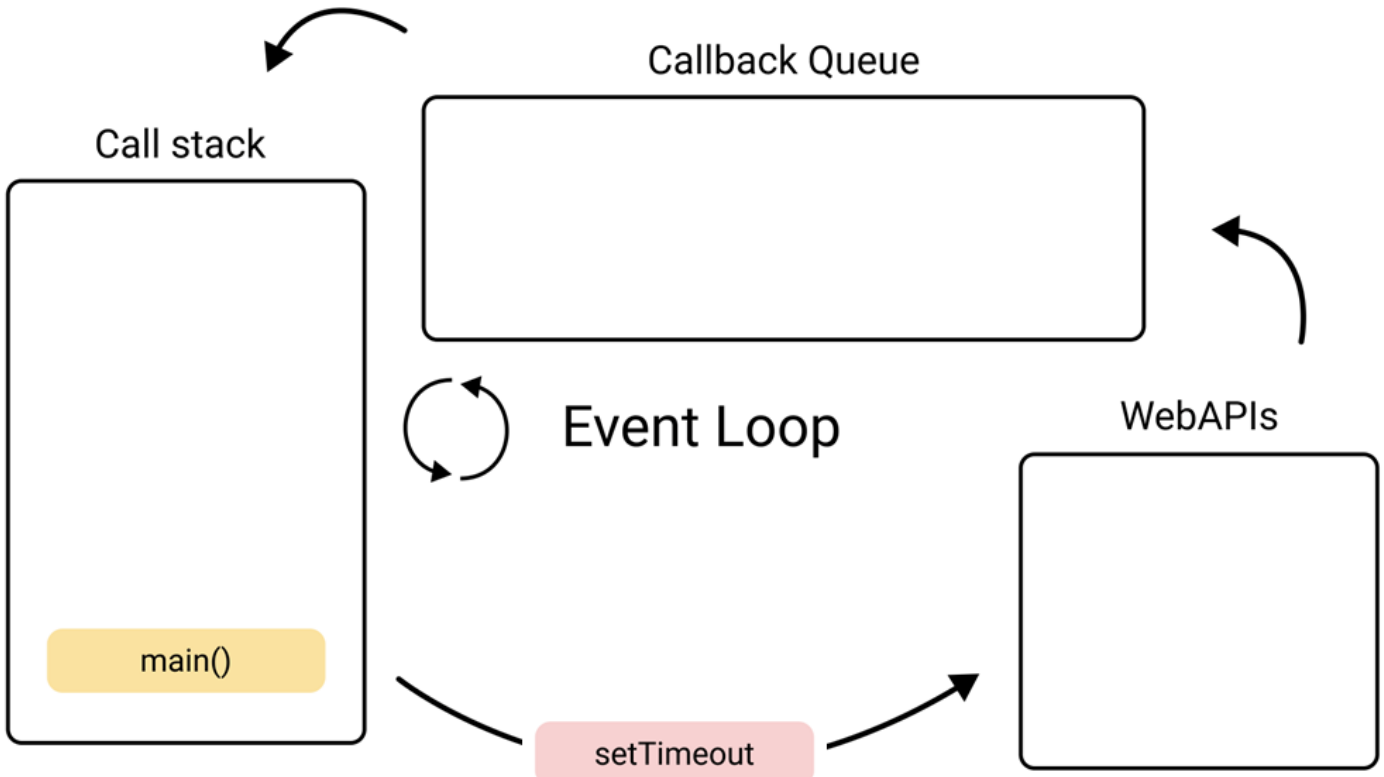
JavaScript의 Event Loop



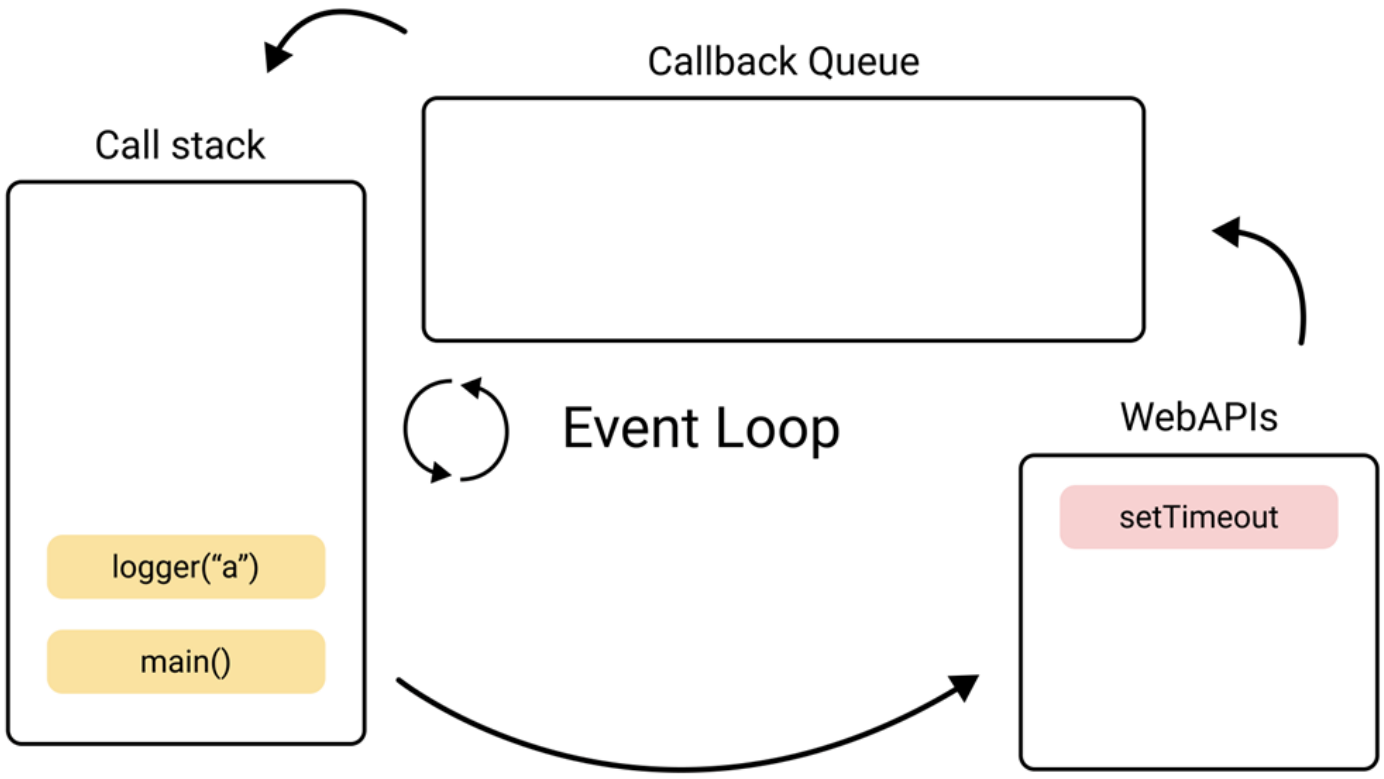
JavaScript의 Event Loop



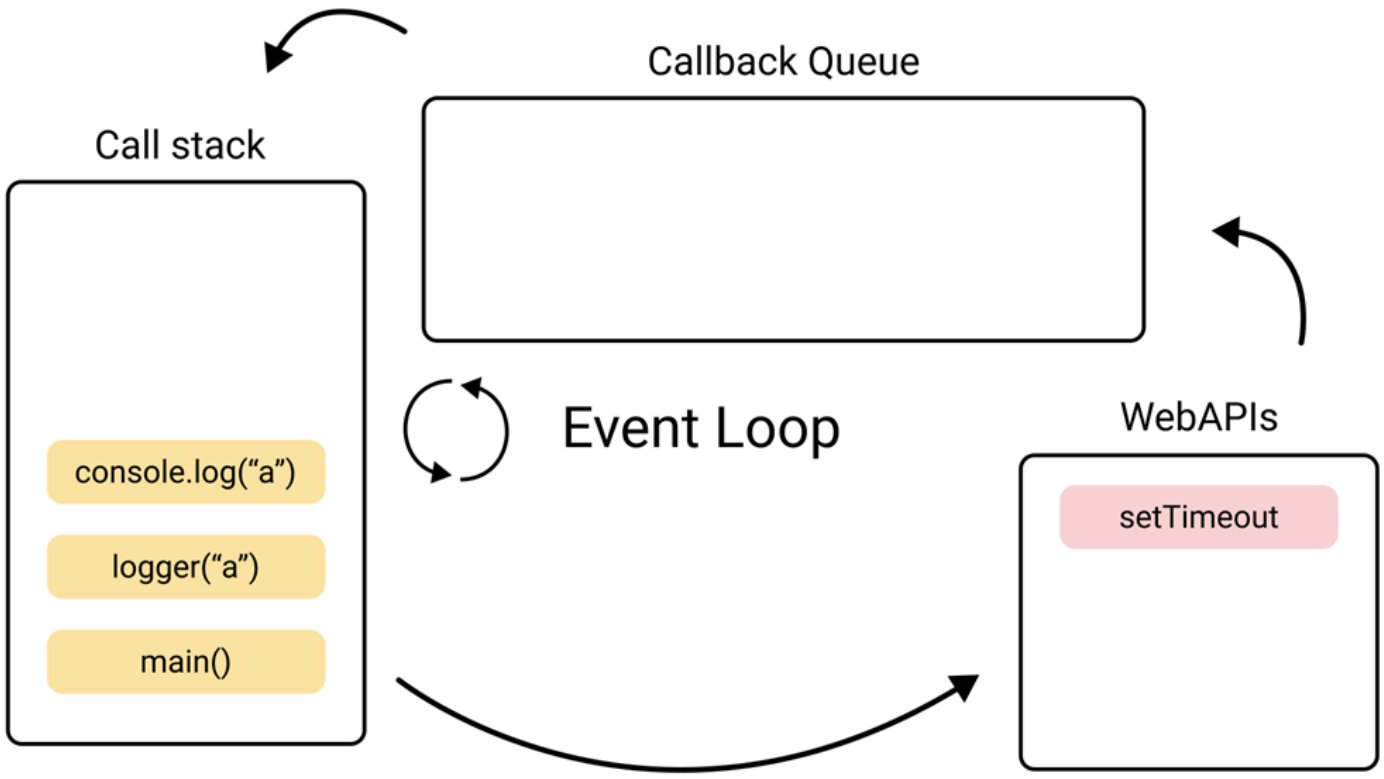
JavaScript의 Event Loop



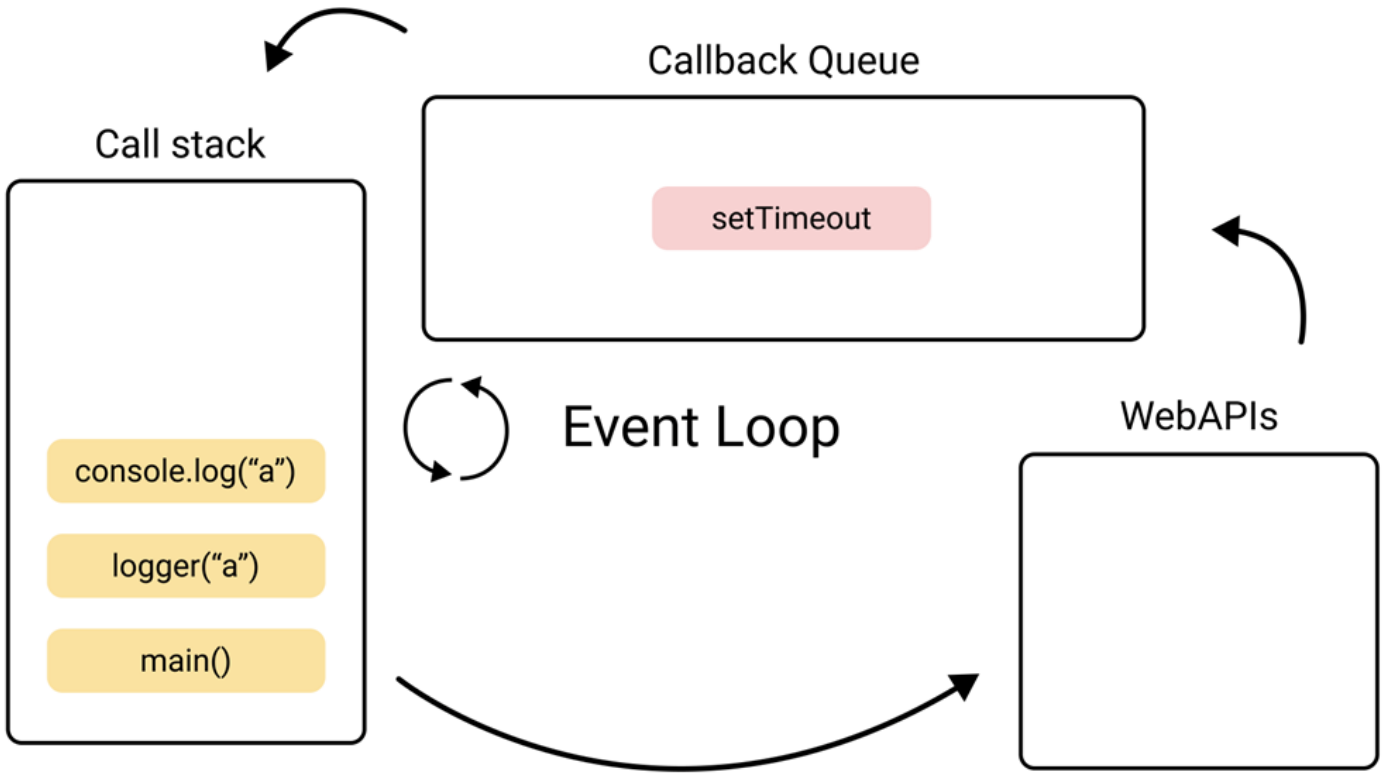
JavaScript의 Event Loop



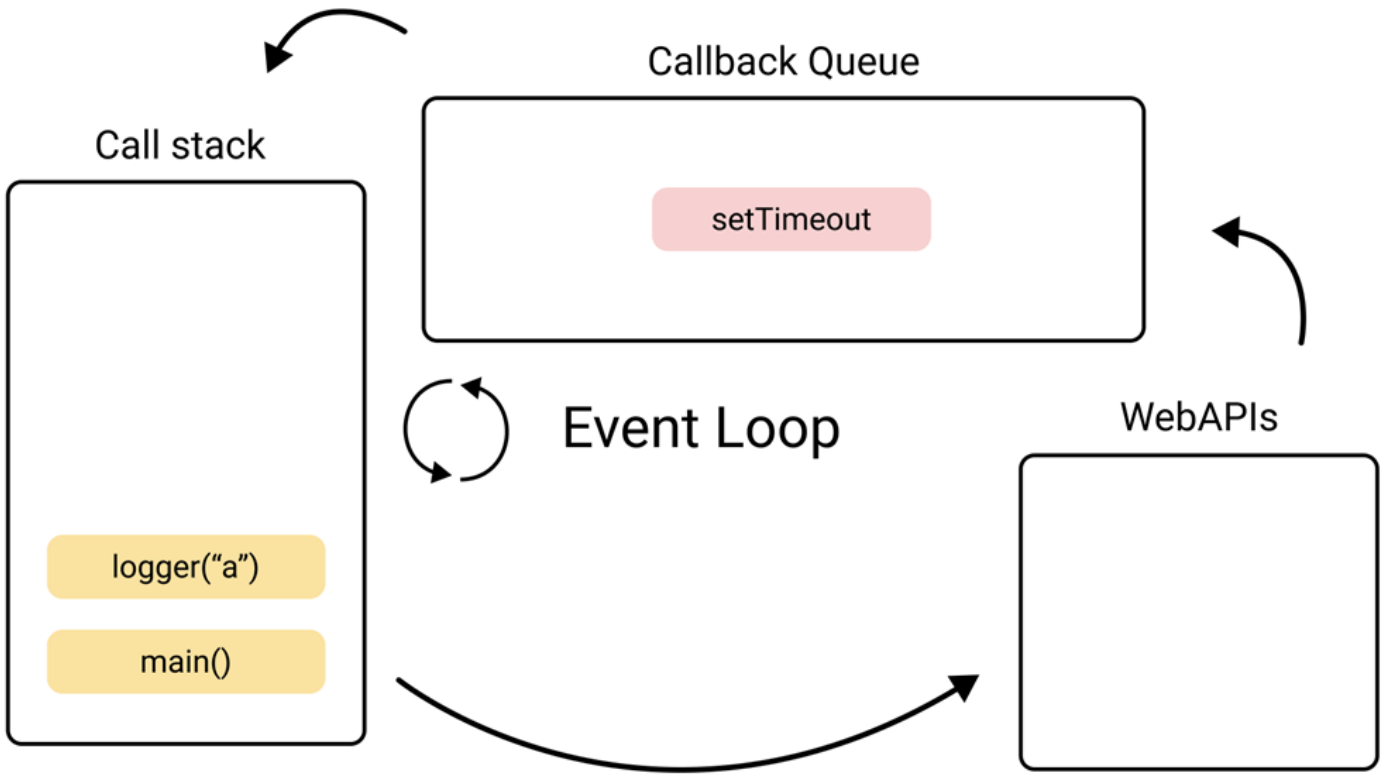
JavaScript의 Event Loop



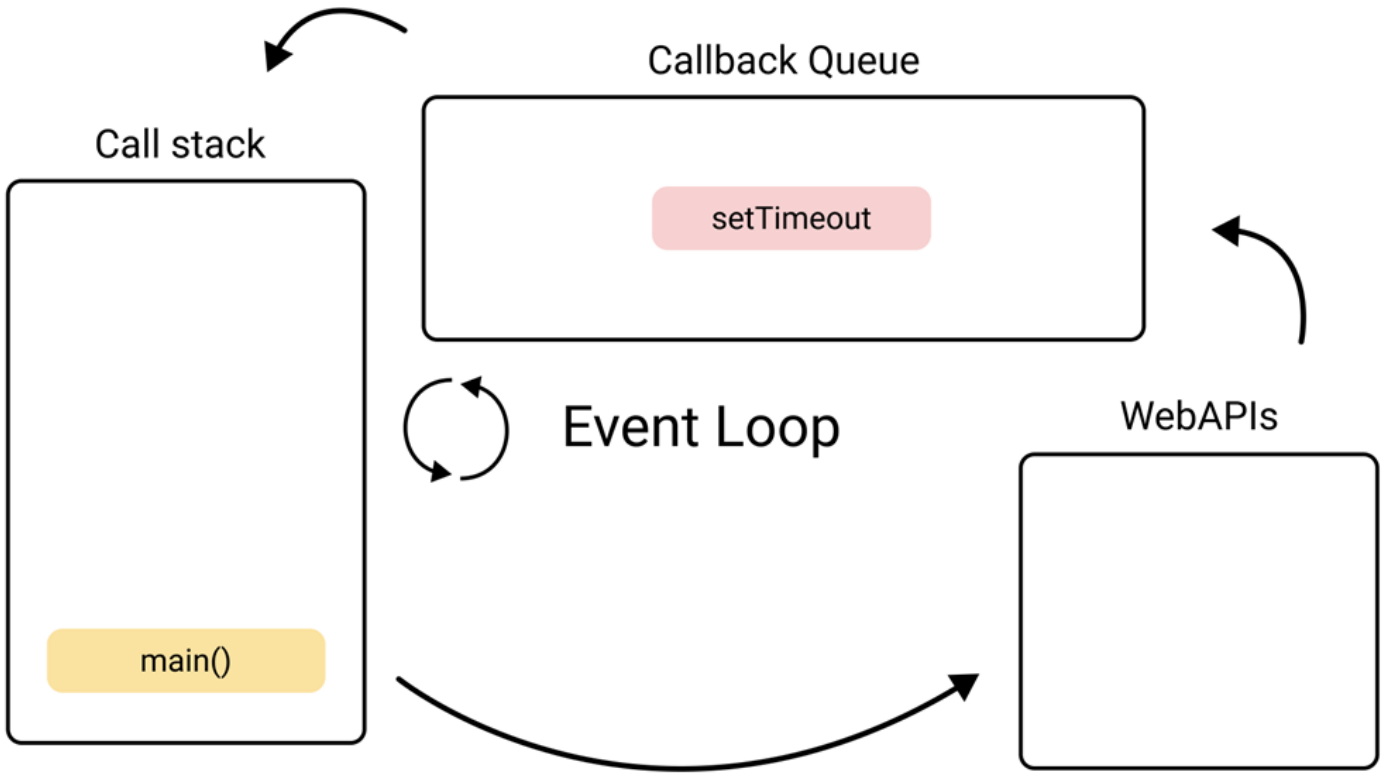
JavaScript의 Event Loop



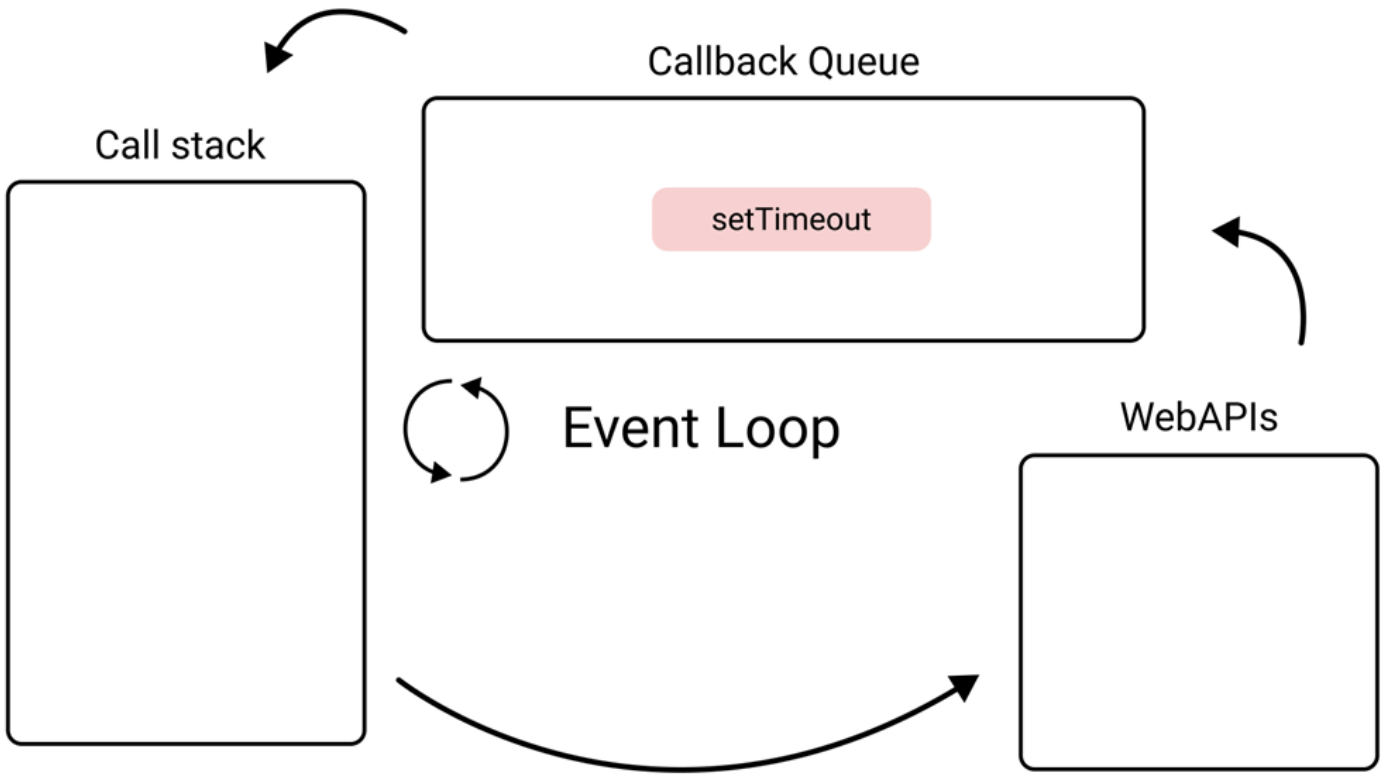
JavaScript의 Event Loop



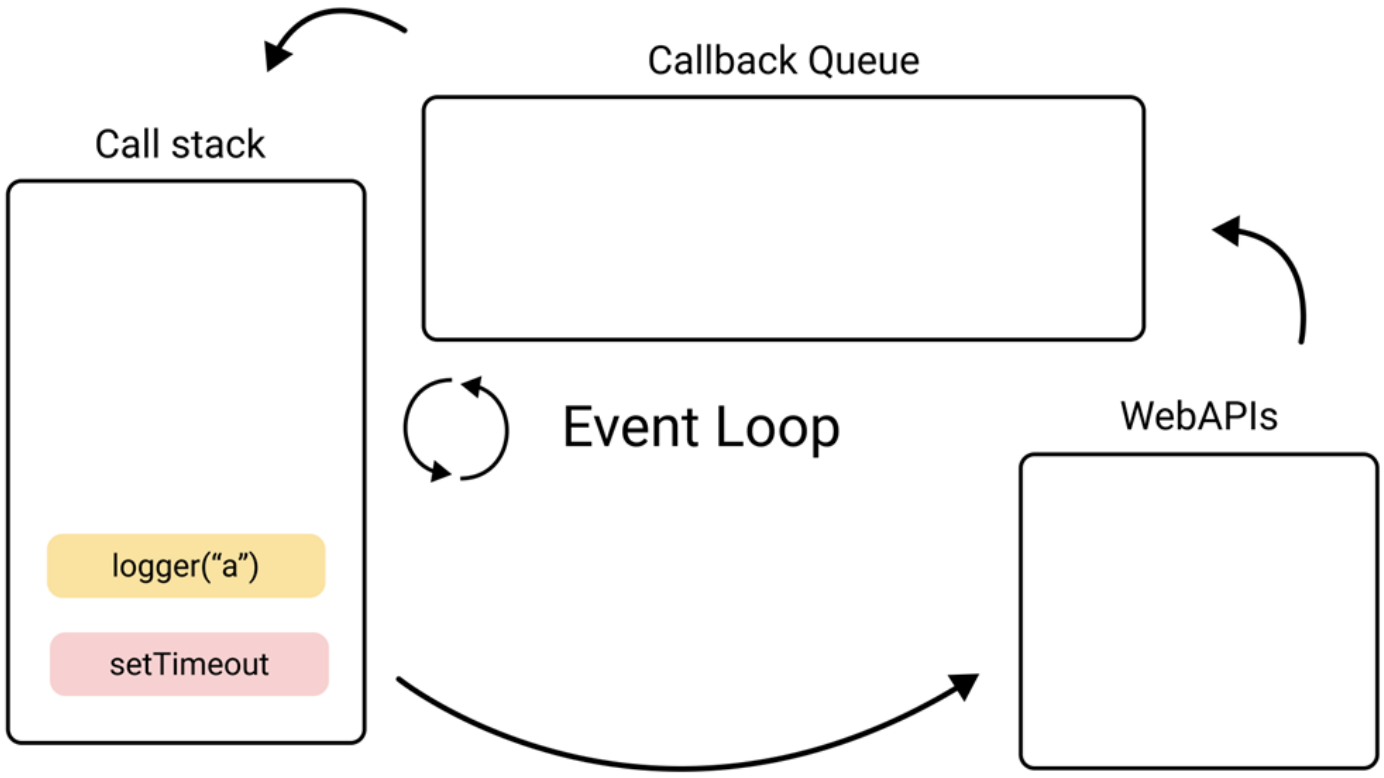
JavaScript의 Event Loop



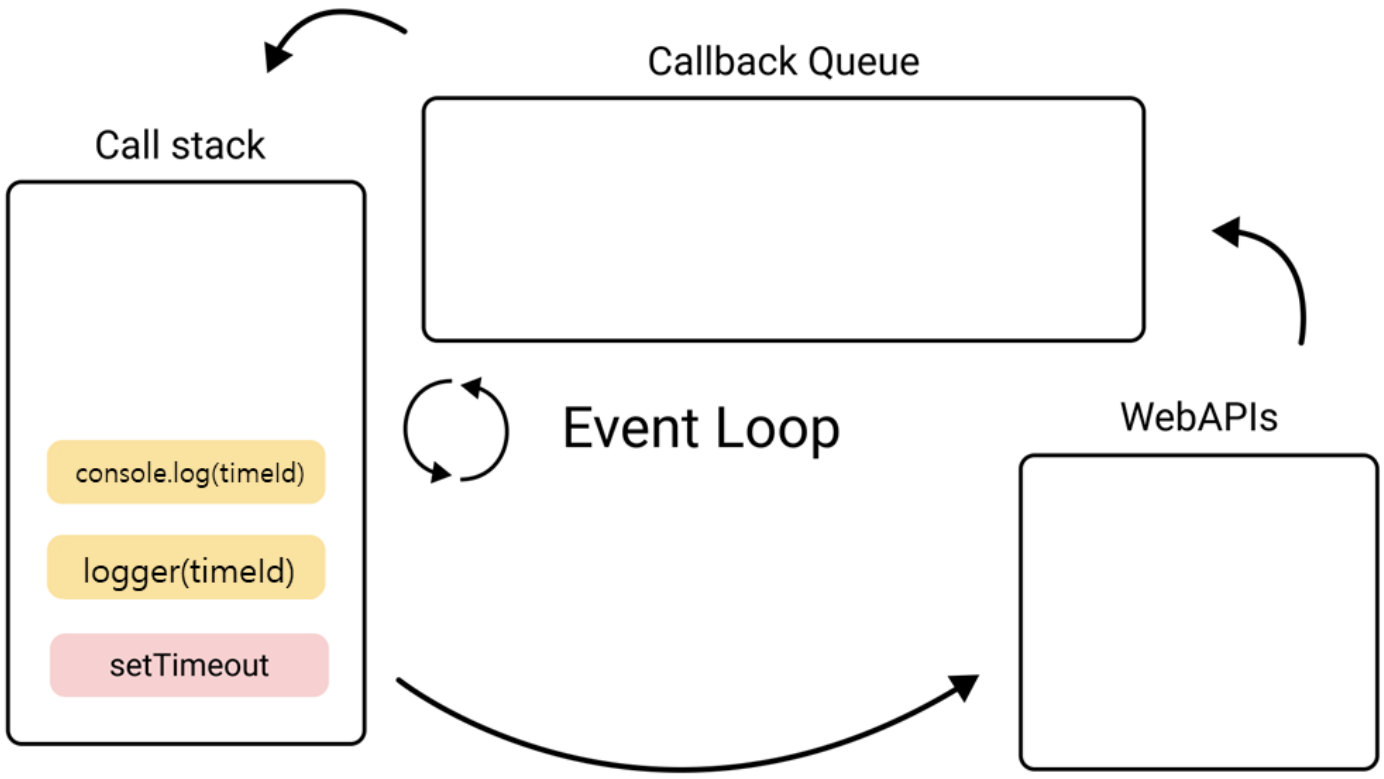
JavaScript의 Event Loop



JavaScript의 Event Loop



JavaScript의 Event Loop



JavaScript는 웹 프론트엔드 개발에서 빼놓을 수 없는 언어 중 하나입니다. 그리고 JavaScript에서 비동기(asynchronous) 동작을 다루는 것은 매우 중요합니다. JavaScript는 단일 스레드(single-thread) 언어로, 한 번에 하나의 작업(task)만 처리할 수 있습니다. 그러나 이벤트 루프(event loop)라는 메커니즘을 사용하여 비동기 동작을 지원합니다.

이벤트 루프는 브라우저나 Node.js와 같은 환경에서 실행되는 JavaScript 코드의 동작 방식을 제어합니다. 이벤트 루프는 큐(queue)에 있는 작업(task)을 순서대로 실행하며, 이 작업이 완료되지 않았을 때 블로킹(blocking)되지 않고 다음 작업을 수행할 수 있습니다.

이벤트 루프는 다양한 요소로 구성되어 있습니다. 먼저 Call Stack이 있습니다. Call Stack은 현재 실행 중인 함수의 정보를 저장하는 공간입니다. 다음으로 Web APIs가 있습니다. Web APIs는 브라우저에서 제공하는 API들입니다. 예를 들어, setTimeout()이나 XMLHttpRequest 객체 등이 여기에 속합니다. Web APIs에서는 비동기 동작이 일어납니다. Web APIs에서 발생한 콜백 함수들이 Callback Queue에 저장됩니다. Callback Queue는 Web APIs에서 발생한 콜백 함수들이 저장되는 공간입니다. 마지막으로, Event Loop가 있습니다. Event Loop는 Call Stack과 Callback Queue를 모니터링하며, Call Stack이 비었을 때 Callback Queue에 있는 콜백 함수를 Call Stack으로 이동시킵니다.

이벤트 루프의 동작 방식을 이해하면, JavaScript에서 비동기 동작을 하는 메서드를 사용할 때 어떻게 동작하는지 이해할 수 있습니다. setTimeout() 메서드를 사용하여 일정 시간이 지난 후에 함수를 실행하도록 할 때, 이벤트 루프는 Web APIs에서 일정 시간을 기다린 후에 콜백 함수를 Callback Queue에 추가합니다. 그리고 Call Stack이 비었을 때, 콜백 함수를 Call Stack으로 이동시켜 실행합니다.

이벤트 루프는 JavaScript에서 비동기 동작을 가능하게 하며, 콜백 함수와 Promise, async/await와 같은 도구들도 이벤트 루프 위에서 동작합니다. 따라서 JavaScript를 사용하는 개발자라면 이벤트 루프의 동작 방식을 이해하고, 비동기 동작을 적절하게 다룰 수 있도록 하는 것이 중요합니다. 이벤트 루프를 이해하는 것은 JavaScript를 더욱 효과적으로 다룰 수 있는 기반을 만드는 것입니다.

비동기 동작

JavaScript에서는 많은 비동기 동작을 다루어야 합니다. 이벤트 루프가 이러한 비동기 동작을 제어하기 위해 사용되는데, 대표적인 비동기 동작으로는 setTimeout(), setInterval(), Promise, async/await 등이 있습니다.

setTimeout()은 일정 시간이 지난 후에 콜백 함수를 실행하는 메서드입니다. 이때, 일정 시간이 지난 후에 실행되는 콜백 함수는 Web APIs에서 실행됩니다. setInterval()은 일정한 간격으로 콜백 함수를 실행하는 메서드입니다. 이 메서드도 setTimeout()과 동일하게 Web APIs에서 실행됩니다.

Promise는 비동기 처리를 위한 객체입니다. Promise를 사용하여 어떤 작업을 비동기적으로 처리하고, 작업이 완료됐을 때 결과를 반환할 수 있습니다. Promise는 미래에 완료될 작업을 대신하여 대기(waiting)하는 객체입니다. Promise를 사용하면 콜백 지옥(callback hell)과 같은 문제를 해결할 수 있습니다.

async/await는 Promise를 더욱 쉽게 사용할 수 있도록 만든 문법입니다. async 함수는 항상 Promise를 반환하며, await 키워드는 Promise가 결과를 반환할 때까지 함수의 실행을 일시 중지합니다. 이를 통해 동기적인 코드와 유사한 구문으로 비동기적인 코드를 작성할 수 있습니다.

JavaScript에서 비동기 동작을 다루는 방법은 다양하지만, 이벤트 루프의 동작 방식을 이해하면 어떤 방법을 사용하더라도 코드를 더욱 효과적으로 작성할 수 있습니다.

결론

이벤트 루프는 JavaScript에서 비동기 동작을 가능하게 하며, JavaScript를 사용하는 개발자라면 이벤트 루프의 동작 방식을 이해하는 것이 중요합니다. 이를 통해 비동기 동작을 적절하게 다룰 수 있고, 코드의 효율성을 높일 수 있습니다. 또한, 이벤트 루프를 이해하면 JavaScript에서 비동기 동작을 다루는 다양한 방법을 보다 쉽게 이해할 수 있습니다.

JavaScript를 사용하는 개발자라면 이벤트 루프를 꼭 학습하고, 비동기 동작을 적절하게 다룰 수 있도록 해야 합니다.