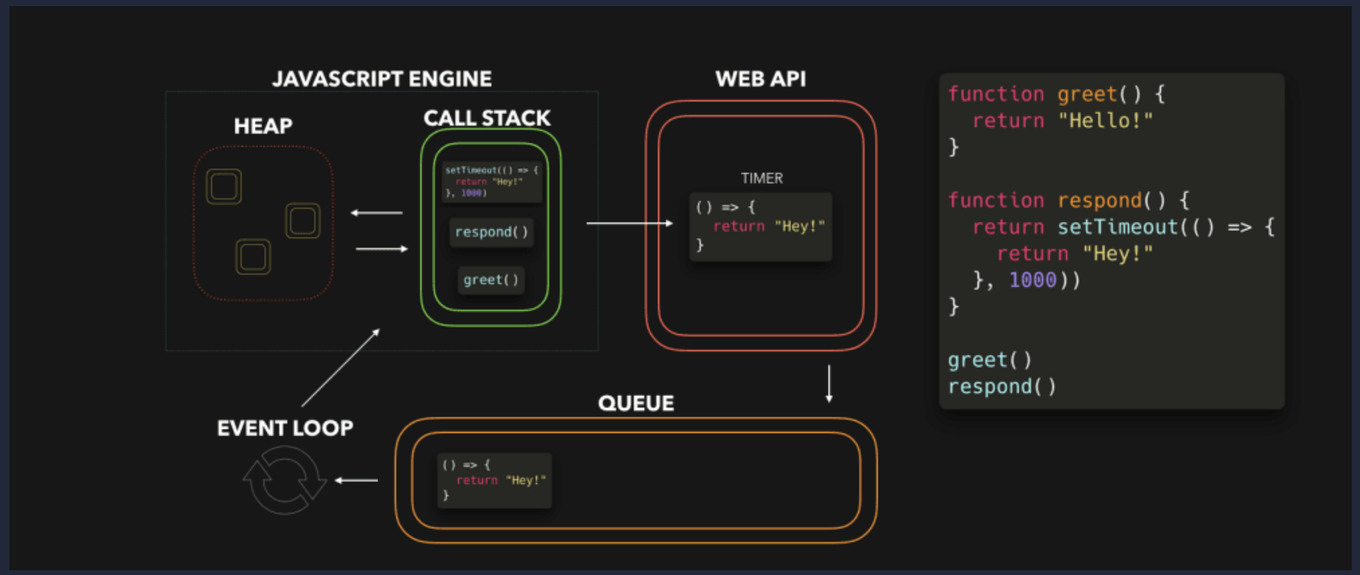
# Event Loop

이벤트 루프

WITH
YAMOO9
DERESA KIM

다음 그림에 대해 설명해보세요.




```
function greet() {
  return "Hello!"
}

function respond() {
  return setTimeout(() => {
    return "Hey!"
  }, 1000))
}

greet()
respond()
```

OR

WITH
YAMOO9
DERESA KIM

# 이벤트 루프란 무엇이며 왜 관심을 가져야 할까?

JavaScript는 **싱글 스레드(Single Thread)로 작동**됩니다. 즉, **한 번에 하나의 작업만 실행 가능**합니다.
일반적으로 큰 문제는 아니지만, 30초 걸리는 작업이 있다고 생각해봤을 때 이 작업이 완료되는
30초 동안 UI는 멈추게 되고 대기 상태가 됩니다.

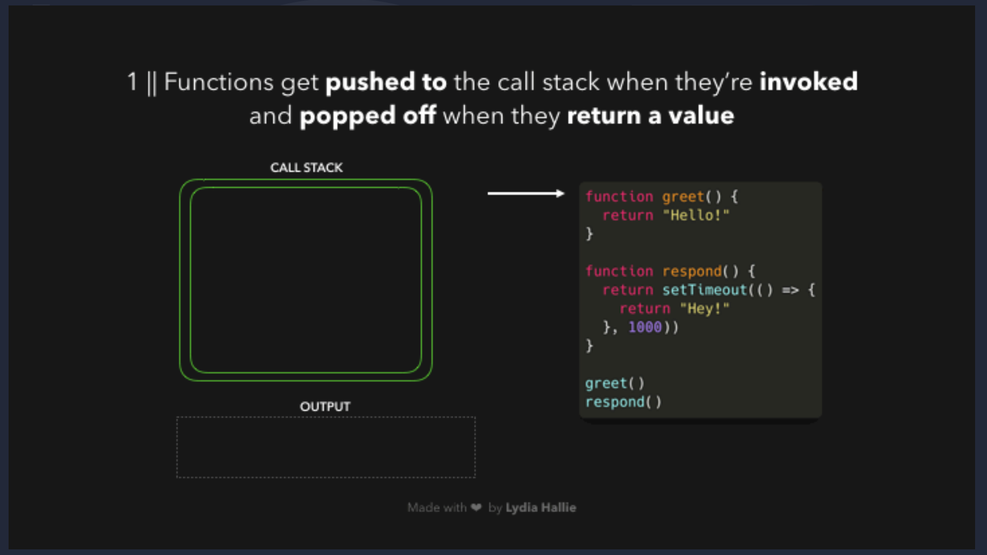
다행히 웹 브라우저 환경은 JavaScript 엔진이 제공하지 않는 **웹 API 기능을 제공**합니다.
(**DOM API**, **setTimeout**, **HTTP Request** 등) 이 기능은 앞서 이야기 한 JavaScript의
싱글 스레드 문제를 문제 없이 해결하는데 도움을 줍니다. (비동기 프로그래밍)

WITH
YAMOO9
DERESA KIM

# 콜 스택(Call Stack)

JavaScript **함수를 호출(Call)** 하면 **호출 스택(Call Stack)에 쌓이게 됩니다.** 호출 스택은 JavaScript 엔진의 일부이며 브라우저 마다 구현이 다릅니다.

스택은 "쌓인다" 또는 "포개진다", "채운다"는 의미를 가지며 함수가 호출되면 쌓이는 더미(예: 팬 케이크)라고 이해하면 좋습니다. **함수가 실행되어 값을 반환하면 콜 스택에서 제거 됩니다.**

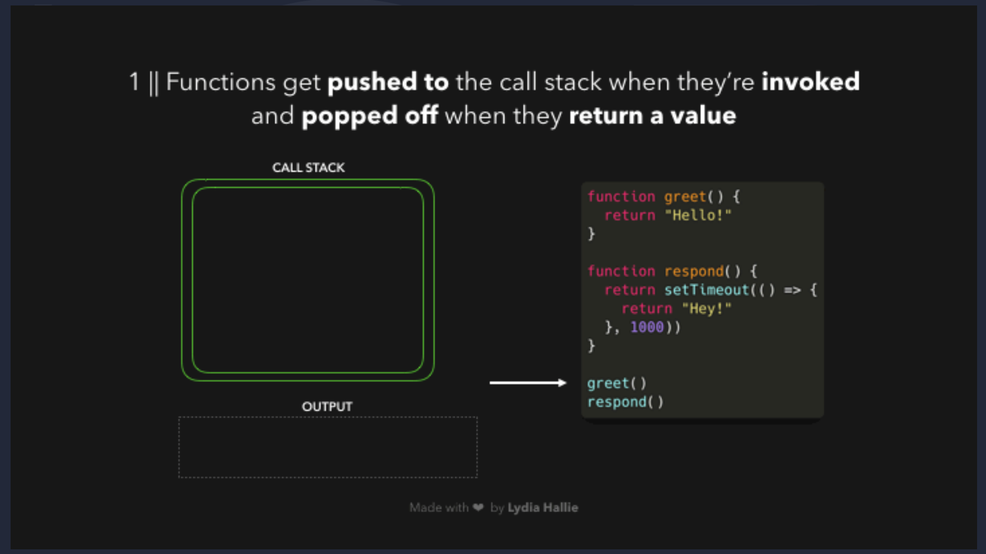



WITH
YAMOO9
DERESA KIM

# 타임아웃 설정(setTimeout)

**respond() 함수**는 **setTimeout() 함수를 반환**합니다.
(setTimeout() 함수는 Web API에 의해 제공)

setTimeout()은 **메인 스레드(Main Thread)를**
**차단하지 않고, 작업을 일정 시간 지연**시킬 수 있습니다.

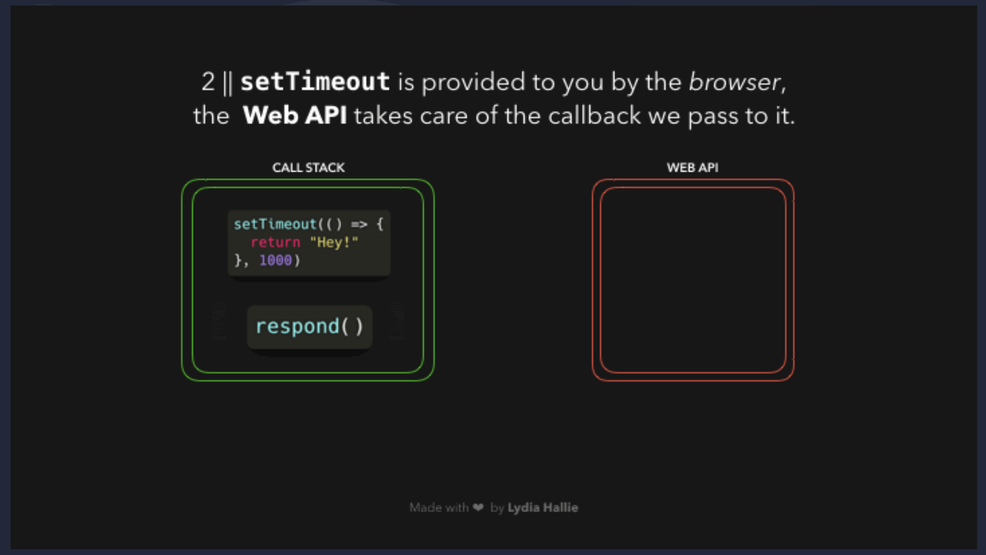



WITH
YAMOO9
DERESA KIM

# 웹(Web) API

setTimeout() 함수의 **첫 번째 인자로 전달된 콜백 함수**는 **Web API에 추가**됩니다. 그 동안 **setTimeout() 함수**와 **respond() 함수가 콜 스택에서 값을 반환한 후 제거** 됩니다.

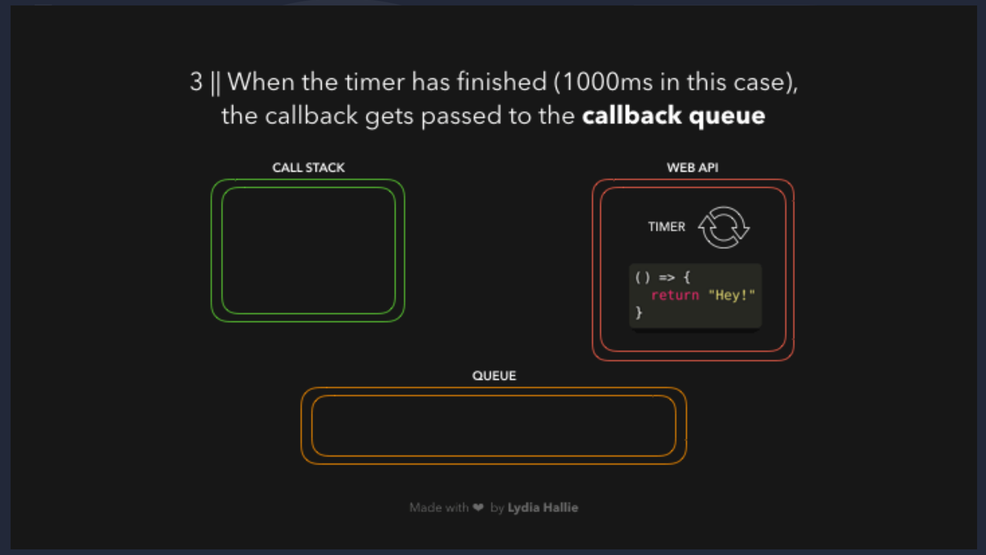
Web API에 추가된 콜백 함수는 setTimeout() 함수의 2번째 인자인 1,000ms 동안 타이머(Timer)가 실행됩니다.




WITH
YAMOO9
DERESA KIM

# 큐(Queue)

**타이머가 완료되면 콜백 함수는 호출 스택으로 바로 이동하지 않고, 큐(대기열)로 전달**됩니다. 타이머가 완료 되었다고해서 바로 호출 스택에 쌓이는 것이 아니라, **큐라고 불리는 대기열에 추가된다는 점에 유의**하세요. 함수는 등록된 큐에서 차례를 기다립니다.

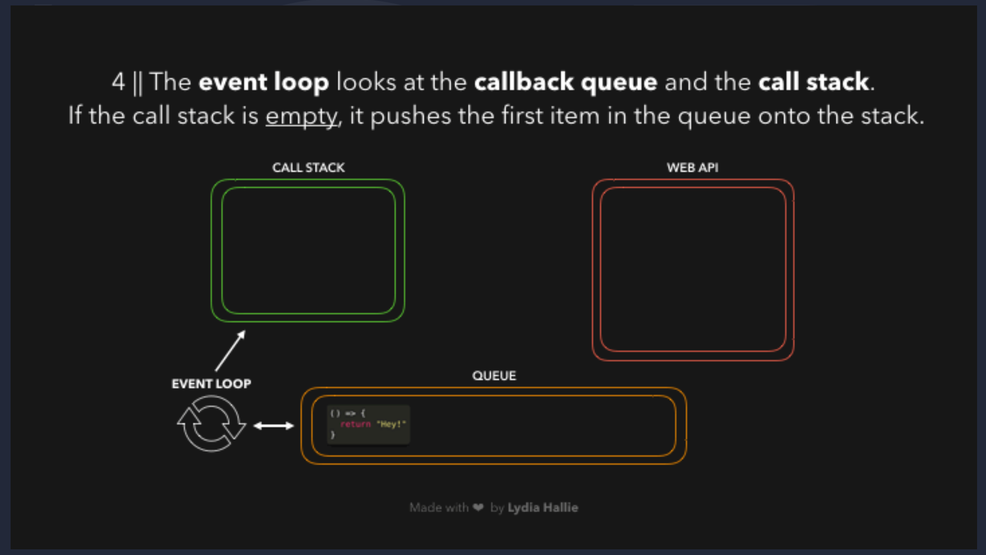



WITH
YAMOO9
DERESA KIM

# 이벤트 루프(Event Loop)

**이벤트 루프가 하는 유일한 일은 호출 스택과 큐(대기열)를 연결하는 것**입니다. **호출 스택이 비어 있으면, 큐에서 차례를 기다리고 있는 콜백 함수가 호출 스택에 추가**됩니다.

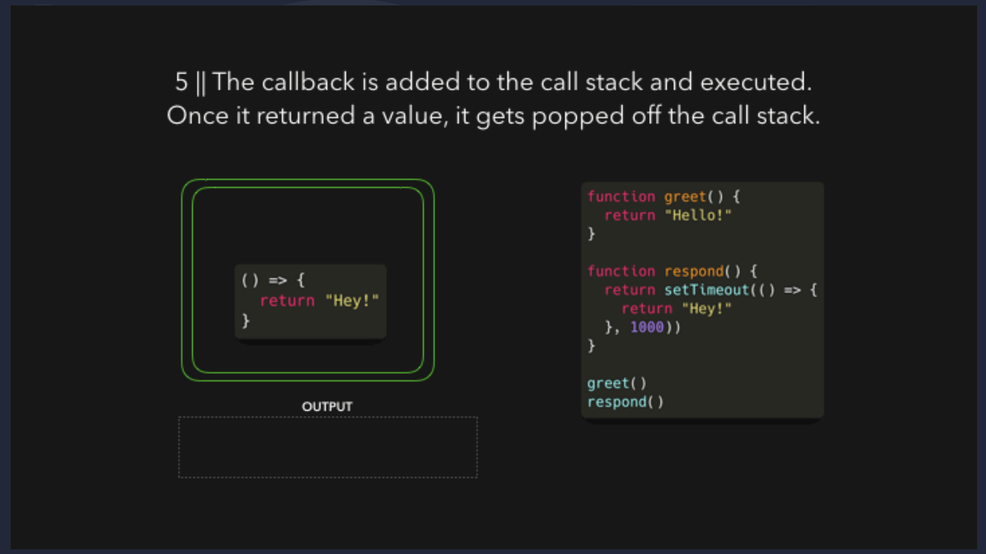
이 경우 다른 함수는 호출되지 않습니다. 다시 말해 콜백 함수가 큐의 첫 번째 항목일 때까지 호출 스택은 비어있습니다.




WITH
YAMOO9
DERESA KIM

# 실행 후, 콜 스택에서 제거 (Popped Off)

호출 스택에 추가된 **콜백 함수는 실행된 후, 값을 반환**하고
호출 스택에서 **제거**됩니다.

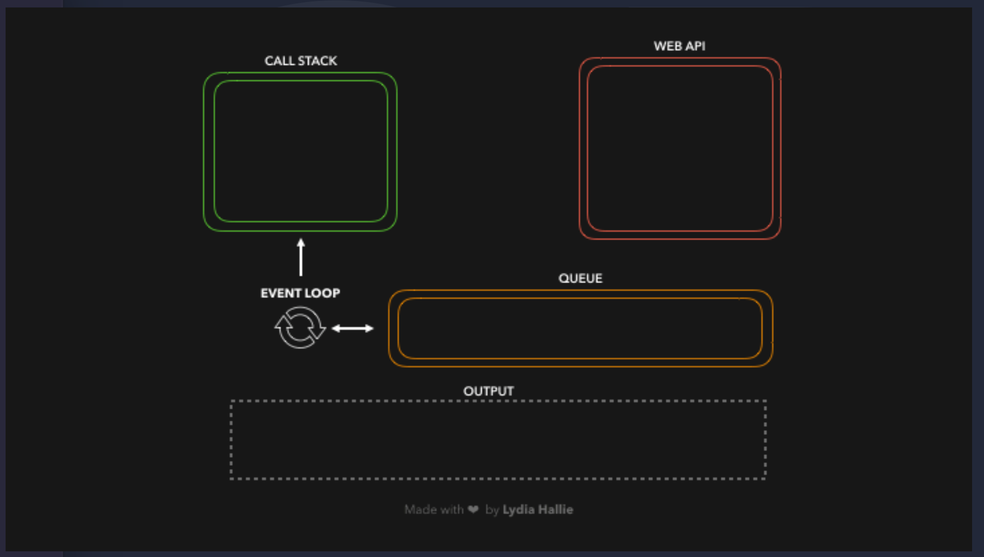



WITH
YAMOO9
DERESA KIM

# 다음 코드의 실행 결과를 유추해보면?

```
const foo = () ⇒ console.log("First");
const bar = () ⇒ setTimeout(
                    () ⇒ console.log("Second"), 500
                  );
const baz = () ⇒ console.log("Third");

// 함수 실행 결과는? 🤔 "Second" → "First" → "Third"?
bar();
foo();
baz();
```




WITH
YAMOO9
DERESA KIM

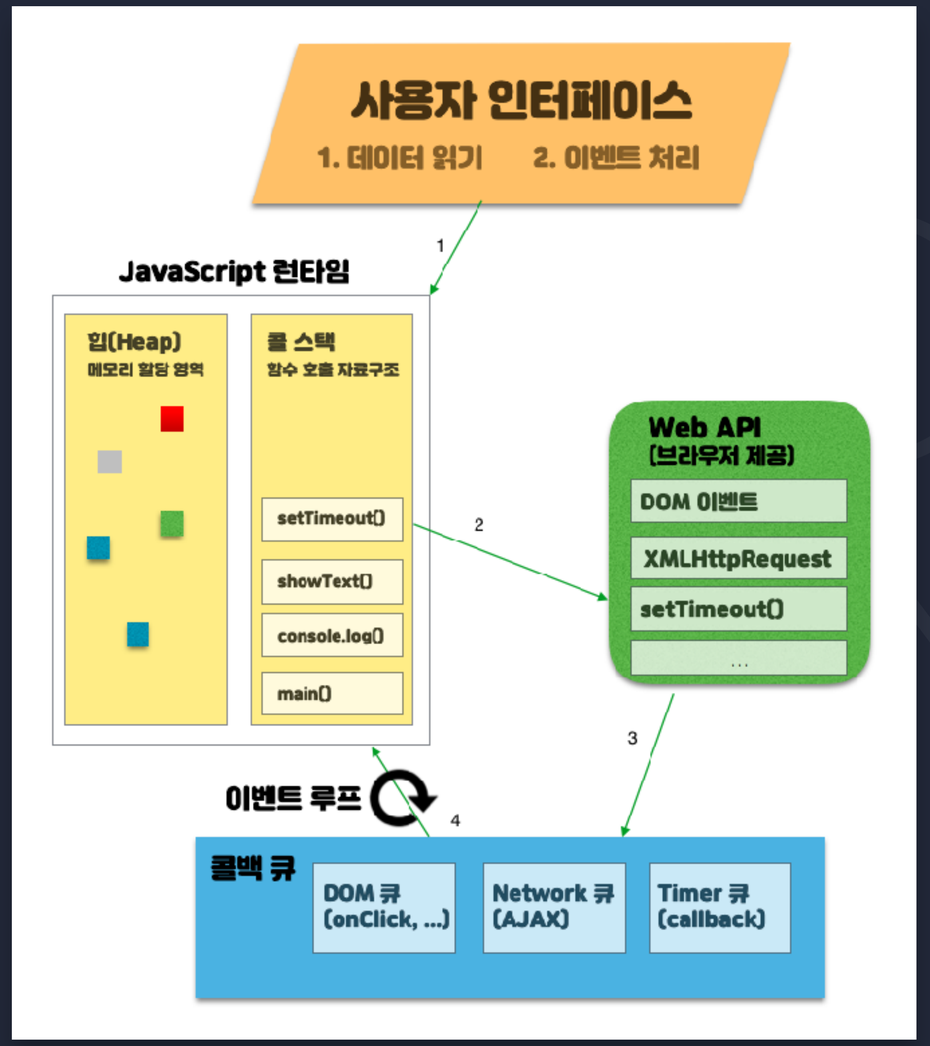
# ✨♻️ 결론(Conclusion)

JavaScript는 단 하나의 스레드로 1개의 동시성(Concurrency) 만을 다루는 프로그래밍 언어입니다. 이것이 의미하는 바는 JavaScript가 단 1개의 작업만 다룰 수 있다는 점입니다.

JavaScript는 1개의 호출 스택(Call Stack)과 큐(Queue)를 갖습니다.

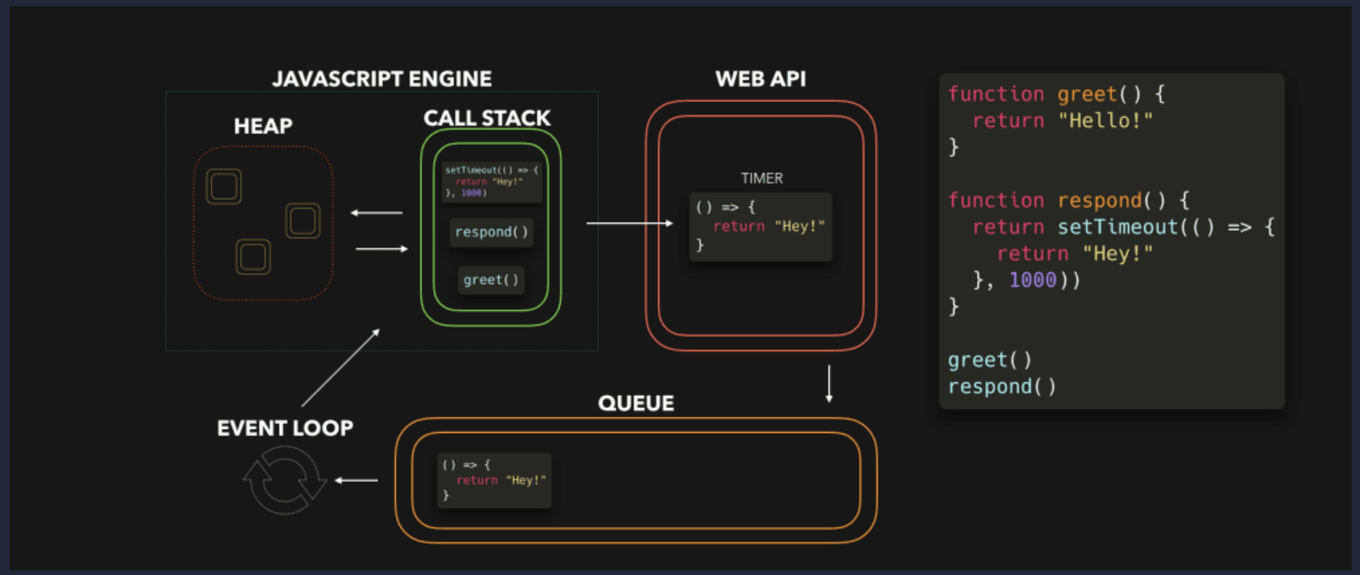
호출 스택은 함수의 호출을 기록하는 자료구조 입니다.
큐는 실행 될 콜백 함수나 실행 될 메시지가 담긴 리스트 입니다.

동시성 모델이 가진 지연 처리 문제를 Web API의 비동기 프로그래밍을 활용하면 일정 시간 뒤에 실행 될 콜백 함수를 큐에 등록할 수 있어 JavaScript는 비동기 프로그래밍으로 해당 문제를 해결합니다.




WITH
YAMOO9
DERESA KIM

자! 그럼 다시 아래 그림에 대해 설명해보세요.




```
function greet() {
  return "Hello!"
}

function respond() {
  return setTimeout(() => {
    return "Hey!"
  }, 1000))
}

greet()
respond()
```

OR

WITH
YAMOO9
DERESA KIM