# 자바스크립트 이벤트 루프

Node.js의 **이벤트 루프(Event Loop)**는 JavaScript 런타임 환경에서 매우 중요한 역할을 합니다. JavaScript는 기본적으로 **단일 스레드**로 동작하는 언어인데, 그럼에도 불구하고 Node.js는 **비동기 I/O** 작업을 효율적으로 처리할 수 있습니다. 이는 **이벤트 루프** 덕분입니다. 이벤트 루프는 Node.js가 비동기 작업을 관리하고 처리하는 메커니즘으로, 서버가 많은 요청을 처리할 수 있게 도와줍니다.

## 1. 이벤트 루프란?

이벤트 루프는 Node.js의 비동기 작업을 처리하는 핵심적인 구성 요소로, 자바스크립트가 단일 스레드에서도 많은 I/O 작업을 처리할 수 있도록 하는 메커니즘입니다. 이벤트 루프는 콜백 함수나 비동기 작업을 큐에 쌓아두고, 메인 스레드가 유휴 상태일 때 그 작업들을 순차적으로 처리합니다.

## 2. 동작 방식

Node.js는 다음과 같은 단계로 이벤트 루프를 통해 비동기 작업을 처리합니다:

### 1) Call Stack (호출 스택)

- 자바스크립트 코드가 실행되면 함수 호출이 스택에 쌓입니다.
- 스택에 있는 작업이 완료되면, 이벤트 루프는 작업 큐에서 다음 작업을 처리할 준비가 된 콜백을 호출합니다.

### 2) Event Queue (이벤트 큐)

- 비동기 작업(파일 읽기, 네트워크 요청 등)을 처리하는 콜백 함수들이 완료되면, 해당 콜백은 이벤트 큐에 대기합니다.
- 이때 이벤트 큐는 완료된 비동기 작업을 트래킹하면서 처리할 준비가 된 콜백을 대기시킵니다.

### 3) 이벤트 루프의 동작

- 이벤트 루프는 호출 스택이 비어 있는지 확인하고, 스택이 비어 있으면 이벤트 큐에서 대기 중인 콜백을 가져와 실행합니다.
- 비동기 작업이 처리되면, 해당 작업과 연결된 콜백 함수가 이벤트 큐에 들어가고, 차례가 되면 이벤트 루프가 이를 처리합니다.

https://vimeo.com/856341807

## 3. 이벤트 루프 단계

이벤트 루프는 크게 몇 가지 단계로 나뉘어 작동합니다:

- **Timers**: `setTimeout()` 과 `setInterval()` 로 예약된 콜백이 이 단계에서 실행됩니다.
- **Pending Callbacks**: 일부 I/O 작업이 완료되었을 때 실행되는 콜백이 이 단계에서 처리됩니다.

- **Idle, Prepare**: 내부적으로 사용되는 단계로 일반적으로는 신경 쓰지 않아도 되는 부분입니다.
- **Poll**: I/O 작업이 완료될 때까지 대기하거나, 완료된 I/O 작업의 콜백을 실행하는 단계입니다.
- **Check**: `setImmediate()`로 예약된 콜백이 이 단계에서 실행됩니다.
- **Close Callbacks**: 소켓과 같은 리소스가 닫혔을 때 실행되는 콜백이 이 단계에서 처리됩니다.

## 4. 예시: 비동기 코드와 이벤트 루프

```
console.log('Start');

setTimeout(() => {
    console.log('Timeout callback');
}, 0);

console.log('End');
```

1. **'Start'**가 호출 스택에 올라가고 즉시 실행됩니다.
2. **`setTimeout()`** 은 타이머를 설정한 후 비동기 작업으로 분리되어 이벤트 큐에 콜백이 들어갑니다.
3. **'End'**가 스택에 쌓이고 실행됩니다.
4. 스택이 비게 되면 이벤트 루프는 이벤트 큐에서 콜백을 가져와 실행합니다. 이때 타이머가 만료된 `setTimeout`의 콜백이 실행됩니다.

출력 결과:

```
Start
End
Timeout callback
```

이 예시는 비동기 작업이 이벤트 큐에 대기하다가, 메인 스레드가 유휴 상태일 때 이벤트 루프에 의해 처리되는 방식을 보여줍니다.

## 5. Node.js가 이벤트 루프를 통해 효율성을 가지는 이유

- **논블로킹 I/O**: I/O 작업(파일 읽기/쓰기, 네트워크 요청 등)은 블로킹되지 않으며, 이벤트 루프에 의해 비동기로 처리됩니다. 이렇게 하면 자원을 효율적으로 사용할 수 있습니다.
- **단일 스레드에서 다중 요청 처리**: 단일 스레드에서 많은 요청을 처리할 수 있어 스레드 풀이나 복잡한 멀티스레딩 모델을 구현하지 않아도 됩니다.

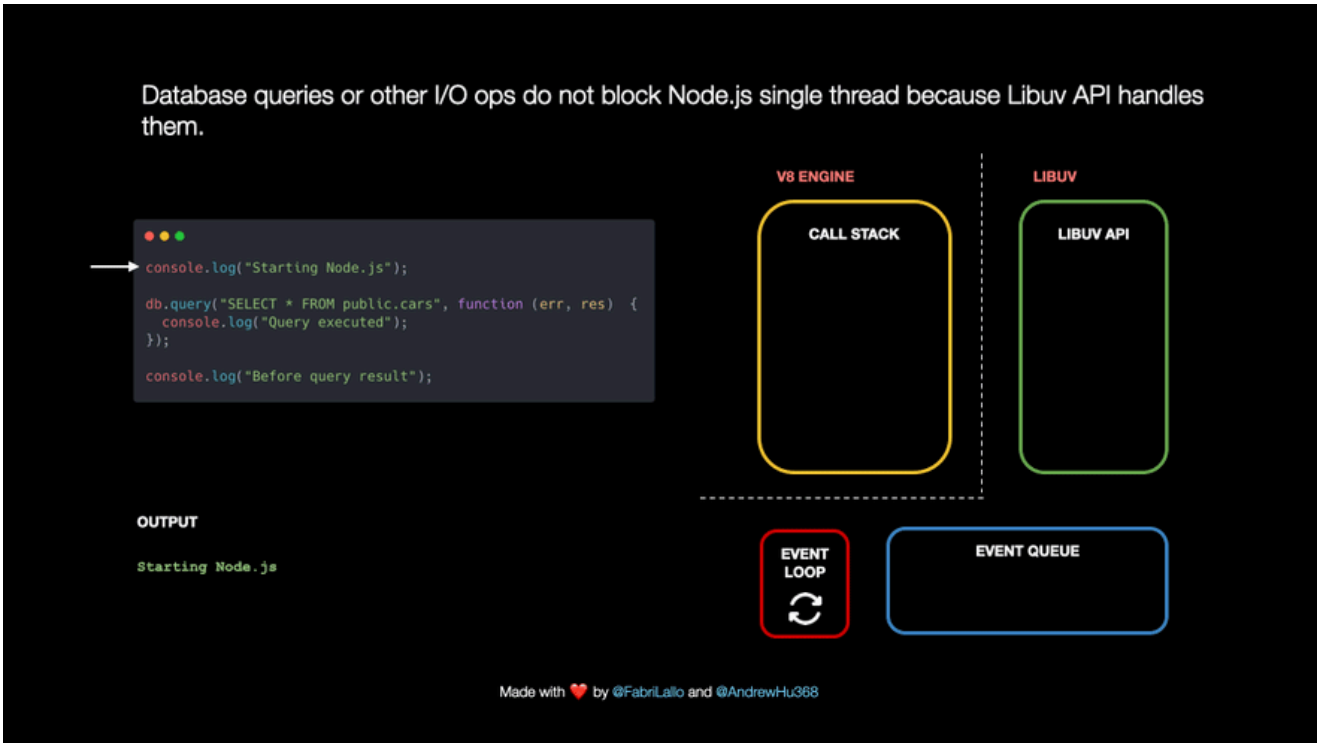
이러한 특성 덕분에 Node.js는 서버 측 개발에서 특히 높은 성능을 발휘할 수 있습니다.

## 6. **이벤트 루프 동작방식

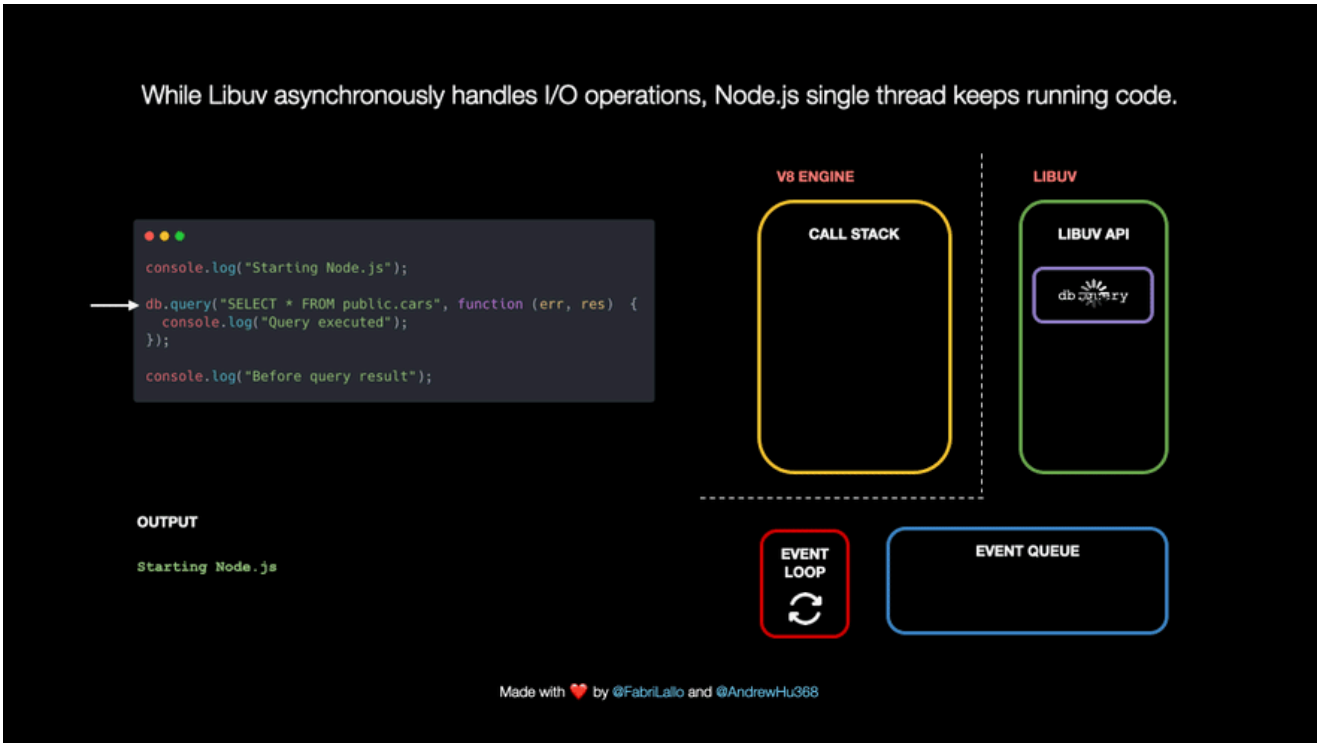
https://medium.com/@emadelmogy/node-js-event-loop-introduction-be4da9a1c0b2

다음 코드 줄은 데이터베이스 쿼리입니다. 이러한 작업은 시간이 오래 걸릴 수 있으므로 즉시 팝오프됩니다. Libuv에 전달되고 Libuv는 백그라운드에서 비동기적으로 이를 처리합니다. 동시에 Node.js는 단일 스레드를 차단하지 않고 다른 코드를 계속 실행할 수 있습니다.

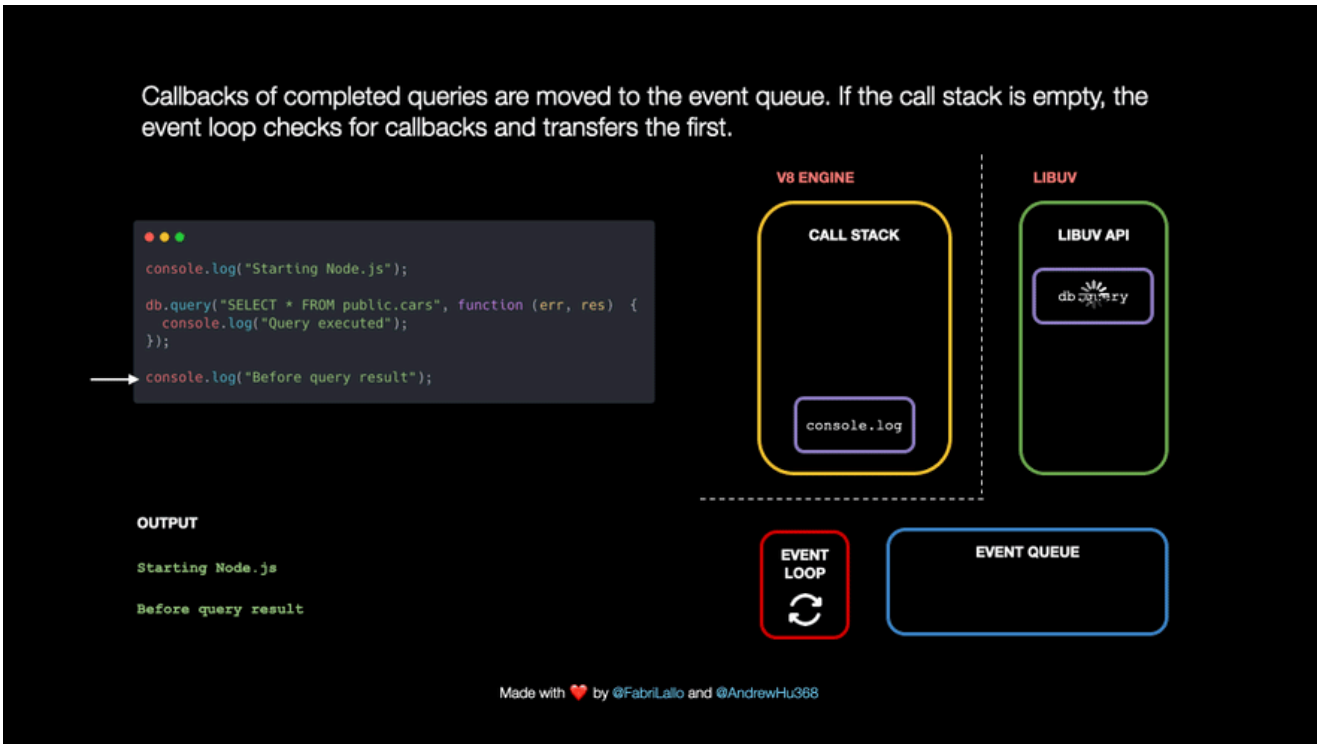
앞으로 Node.js는 작업 결과나 오류를 처리하기 위한 지침과 함께 콜백 함수를 연결했기 때문에 쿼리를 어떻게 처리할지 알게 될 것입니다. 우리의 경우, 그것은 간단 `console.log` 하지만, 프로덕션 애플리케이션에서는 복잡한 비즈니스 로직이나 데이터 처리가 될 수 있습니다.




Libuv가 백그라운드에서 쿼리를 처리하는 동안 JavaScript는 차단되지 않고 계속 작업할 수 있습니다 `console.log("Before query result")`.



쿼리가 완료되면 콜백이 I/O 이벤트 큐에 푸시되어 곧 실행됩니다 . * * 이벤트 루프는 큐와 호출 스택을 연결합니다. 호출 스택이 비어 있는지 확인하고 실행을 위해 첫 번째 큐 항목을 이동합니다.




# 결론

이벤트 루프, 위임, 비동기 처리 메커니즘은 Node.js의 비밀 요소로, 수천 개의 연결을 처리하고, 거대한 파일을 읽고 쓰고, 코드의 다른 부분을 작업하는 동안 타이머를 처리합니다.

이 글에서 우리는 Libuv의 중요한 역할과 잠재적으로 오래 실행되는 수많은 작업을 처리하는 능력을 살펴보았습니다. 동시에, 우리는 이벤트 루프와 I/O 이벤트 큐의 비동기 작업 콜백과 호출 스택 간의 브리지/커넥터로서의 역할을 살펴보았습니다. 다음 글에서는 **타이머** , **I/O** , **약속** , **틱** 이 이벤트 루프의 여러 단계에서 어떻게 처리되는지 자세히 살펴보겠습니다.