

# 클라우드보안 과정:

## Node.js의 기본 개념

성결대학교 정보통신공학과

정복래

brjung@sungkyul.ac.kr

# 1.1 노드의 정의

# 1. 노드의 정의

- 공식 홈페이지 설명:
  - Node.js는 크롬 V8 자바스크립트 엔진으로 빌드된 자바스크립트 런타임입니다. Node.js는 이벤트 기반, 논블로킹 I/O 모델을 사용해 가볍고 효율적입니다. Node.js의 패키지 생태계인 npm은 세계에서 가장 큰 오픈 소스 라이브러리 생태계이기도 합니다.
- 런타임? 이벤트 기반? 논블로킹 I/O? npm?
- 노드는 서버 아닌가요?
  - 서버의 역할도 수행할 수 있는 자바스크립트 런타임

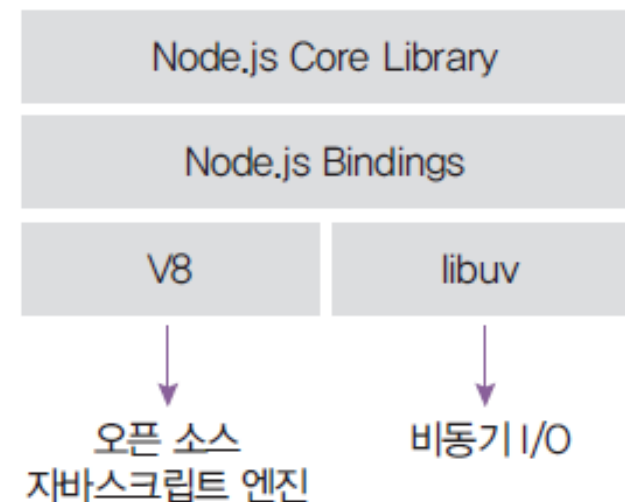
## 2. 런타임

- 노드: 자바스크립트 런타임
  - 런타임: 특정 언어로 만든 프로그램들을 실행할 수 있게 해주는 가상 머신(크롬의 V8 엔진 사용)의 상태
  - ∴ 노드: 자바스크립트로 만든 프로그램들을 실행할 수 있게 해 줌
  - 다른 런타임으로는 웹 브라우저가 있음
  - 노드 이전에도 자바스크립트 런타임을 만들기 위한 많은 시도
  - But, 엔진 속도 문제로 실패

### 3. 내부 구조

- 2008년 V8 엔진 출시, 2009년 노드 프로젝트 시작
- 노드는 V8과 libuv를 내부적으로 포함
  - V8 엔진(오픈 소스 자바스크립트 엔진) → 속도 문제 개선
  - libuv: 노드의 특성인 이벤트 기반, 논블로킹 I/O 모델을 구현한 라이브러리

▼ 그림 1-3 노드의 내부 구조

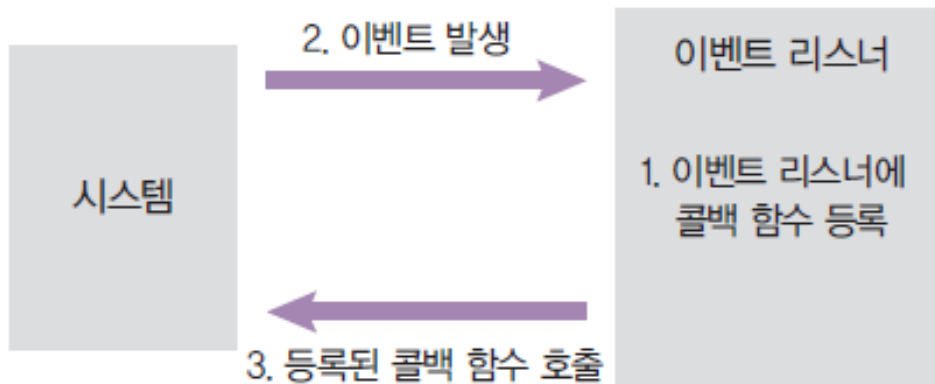


## 1.2 노드의 특성

# 1. 이벤트 기반

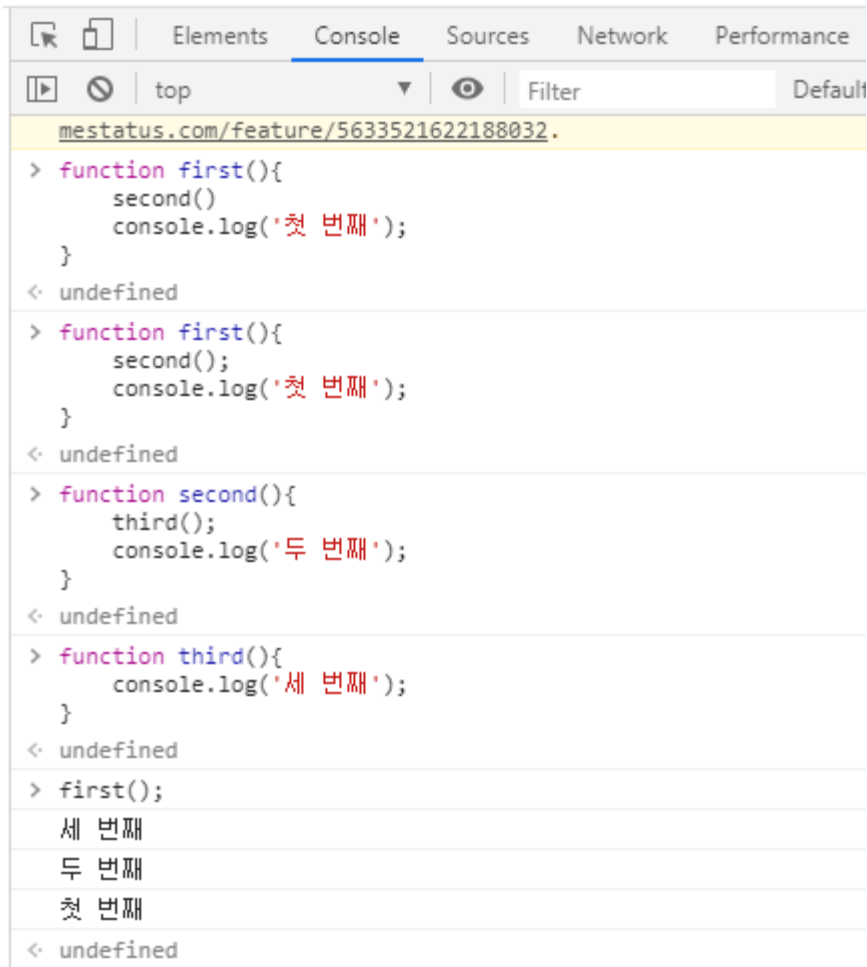
- 이벤트가 발생할 때 미리 지정해둔 작업을 수행하는 방식
  - 이벤트의 예: 클릭, 네트워크 요청, 타이머 등
  - 이벤트 리스너: 이벤트를 등록하는 함수
  - 콜백 함수: 이벤트가 발생했을 때 실행될 함수

▼ 그림 1-4 이벤트 기반

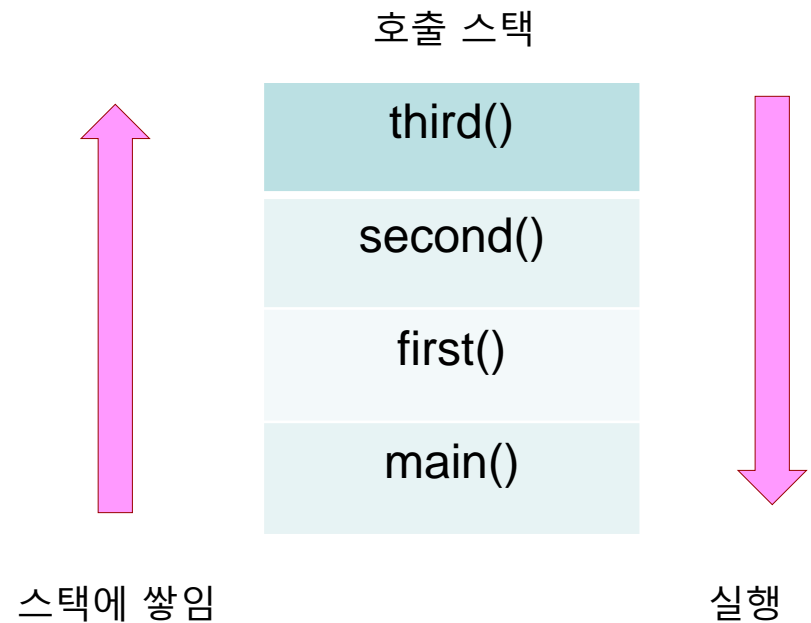


# 1. 이벤트 기반

● 크롬브라우저 → F12(개발자도구) → Console



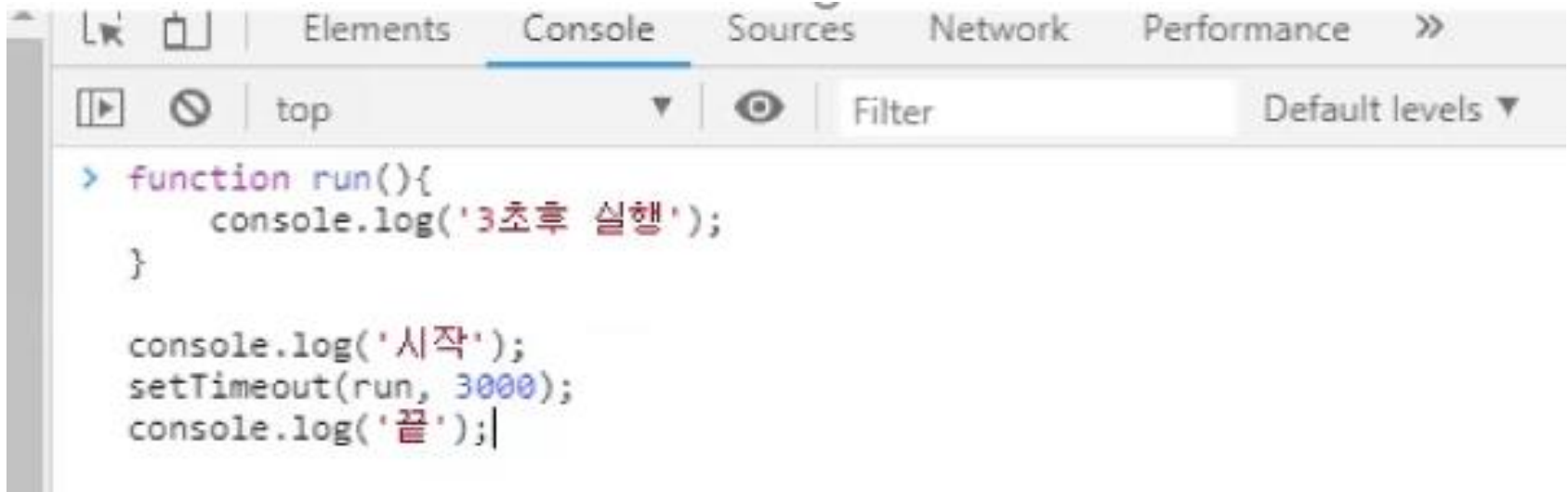
```
function first(){
  second()
  console.log('첫 번째');
}
undefined
function first(){
  second();
  console.log('첫 번째');
}
undefined
function second(){
  third();
  console.log('두 번째');
}
undefined
function third(){
  console.log('세 번째');
}
undefined
> first();
세 번째
두 번째
첫 번째
undefined
```





# 이벤트의 이해

- 크롬 -> F12(개발자모드) -> 콘솔
  - 메모장에 아래 코드를 타이핑 한 후, 콘솔에 붙여넣고 바로 실행

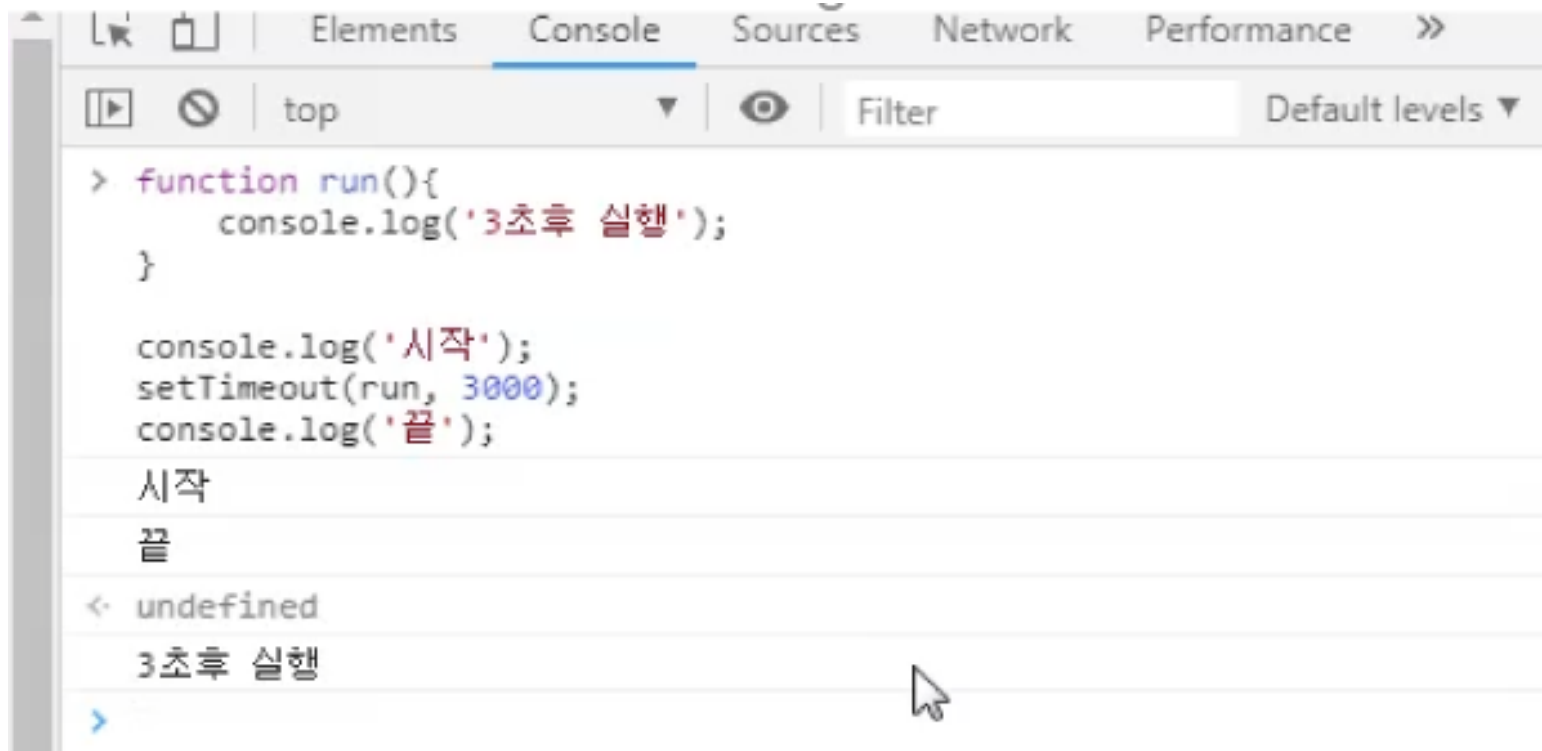
A screenshot of the Chrome DevTools Console. The 'Console' tab is selected. The code entered is: 

```
> function run(){
    console.log('3초후 실행');
}

console.log('시작');
setTimeout(run, 3000);
console.log('끝');
```

# 이벤트의 이해

## 예상한 결과...?



```
> function run(){
    console.log('3초후 실행');
}

console.log('시작');
setTimeout(run, 3000);
console.log('끝');
```

시작

끝

< undefined

3초후 실행

# 1. 이벤트 기반

## ● 이벤트 루프

- 이벤트 발생시 호출할 콜백 함수 관리, 콜백 함수의 실행 순서 결정
- 노드 종료까지 이벤트 처리를 위한 작업 반복

## ● 태스크 큐 (=콜백 큐)

- 이벤트 발생 후 호출되어야 할 콜백 함수들을 기다리는 공간
- 콜백 큐라고도 함
  - 콜백들이 이벤트 루프가 정한 순서대로 대기

## ● 백그라운드

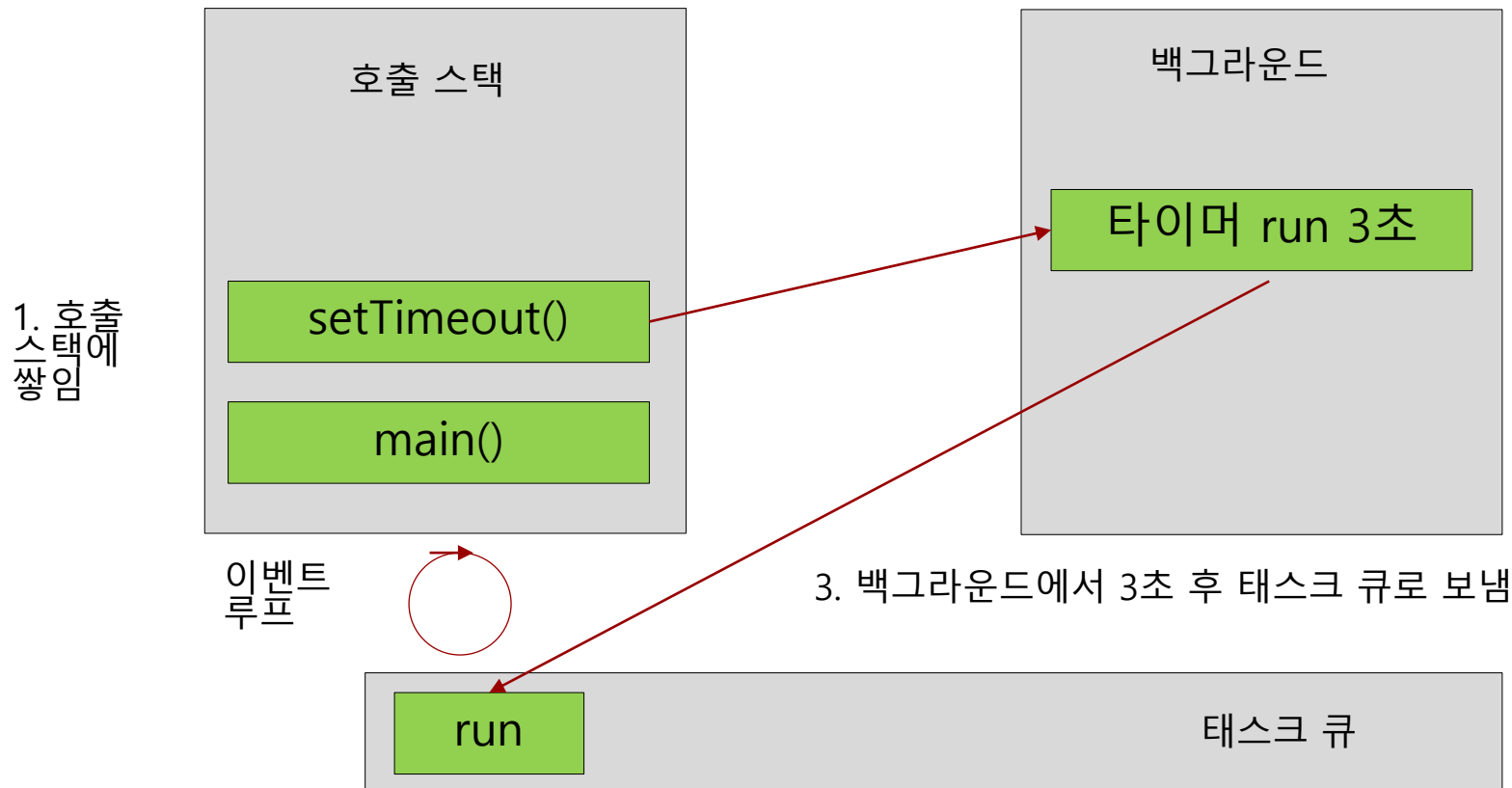
- 타이머, I/O작업 콜백, 이벤트 리스너들이 대기하는 곳

# 1. 이벤트 기반

## ● 이벤트 루프(Event Loop)

```
> function run(){  
    console.log('3초후 실행');  
}  
  
console.log('시작');  
setTimeout(run, 3000);  
console.log('끝');
```

2. setTimeout 실행시 콜백  
run은 백그라운드로

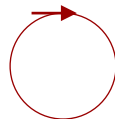
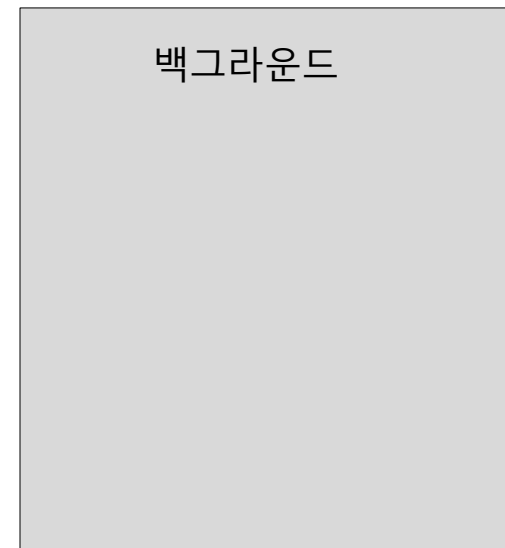
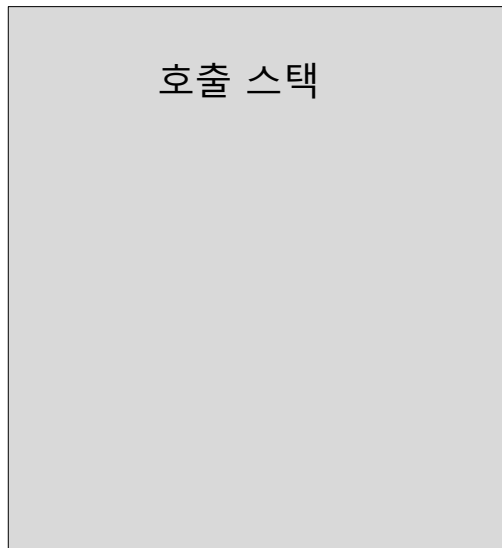


# 1. 이벤트 기반

## ● 이벤트 루프(Event Loop)

```
> function run(){  
    console.log('3초후 실행');  
}  
  
console.log('시작');  
setTimeout(run, 3000);  
console.log('끝');
```

4. 호출 스택 실행이 끝나 비워지면



5. 이벤트 루프가 태스크 콜백을 호출 스택으로 올림

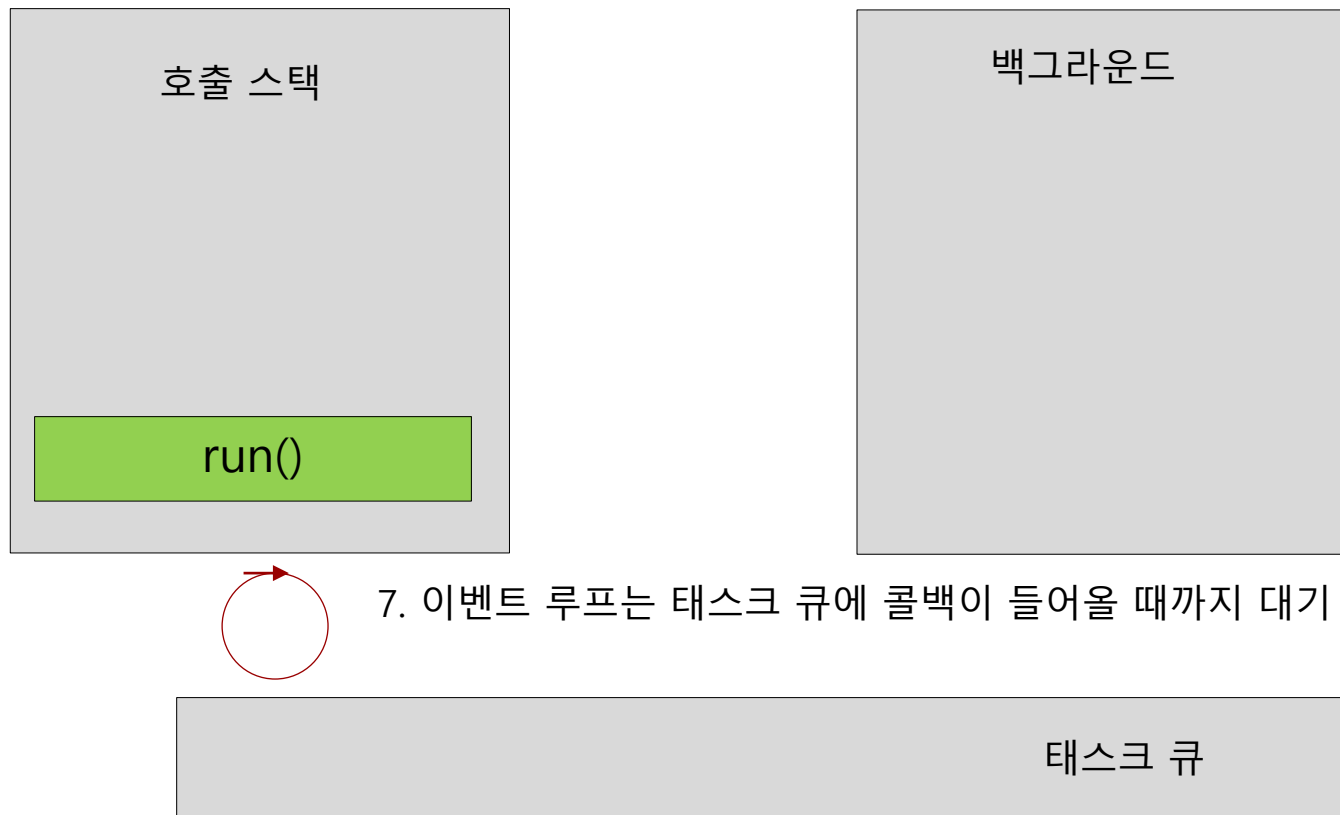


# 1. 이벤트 기반

## ● 이벤트 루프(Event Loop)

```
> function run(){  
    console.log('3초후 실행');  
}  
  
console.log('시작');  
setTimeout(run, 3000);  
console.log('끝');
```

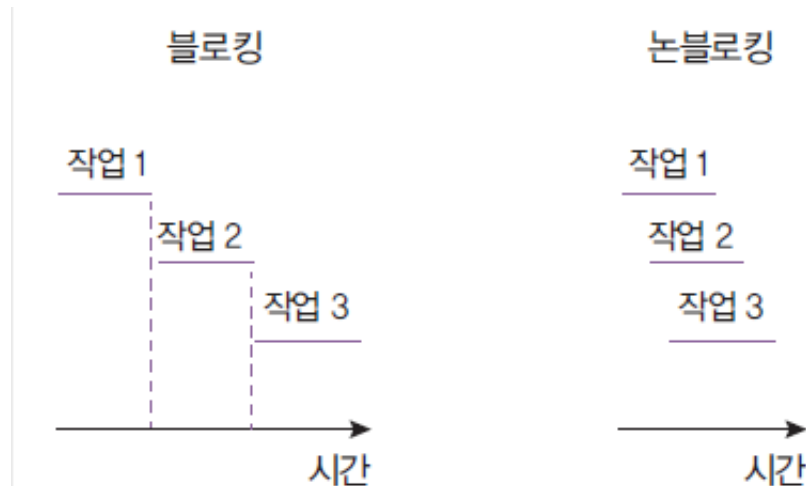
6. run이 호출 스택에서 실행된 후 비워짐



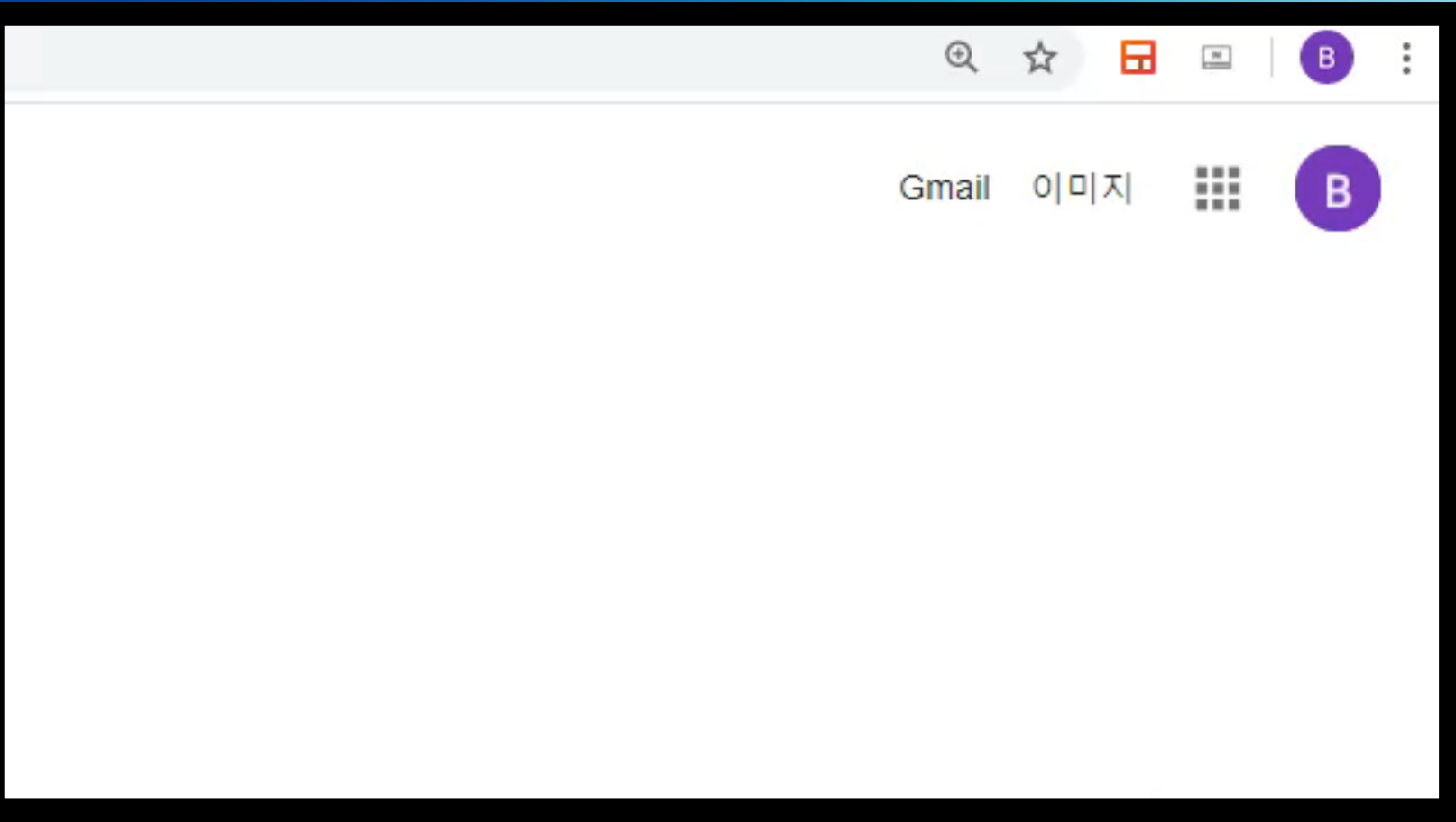
## 2. 논블로킹 I/O

### ● 논블로킹(Non-blocking)

- 오래 걸리는 함수를 백그라운드로 보내서 다음 코드가 먼저 실행되게 하고, 나중에 오래 걸리는 함수를 실행
  - 논블로킹 방식 하에서 일부 작업은 백그라운드에서 병렬로 실행됨
  - 일부 작업:
    - 네트워크 I/O(네트워크 연결 및 닫기, Http 요청 및 응답)
    - 파일 I/O(파일 읽기 및 쓰기)
    - 프로세스 및 시스템 작업(자식 프로세스 생성 및 실행, 시스템 명령 실행, 타이머 및 알람 설정)
  - 나머지 작업은 블로킹 방식으로 실행됨
  - ∴ I/O 작업이 많을 때 노드 활용성이 극대화



# 블록킹 방식 예제

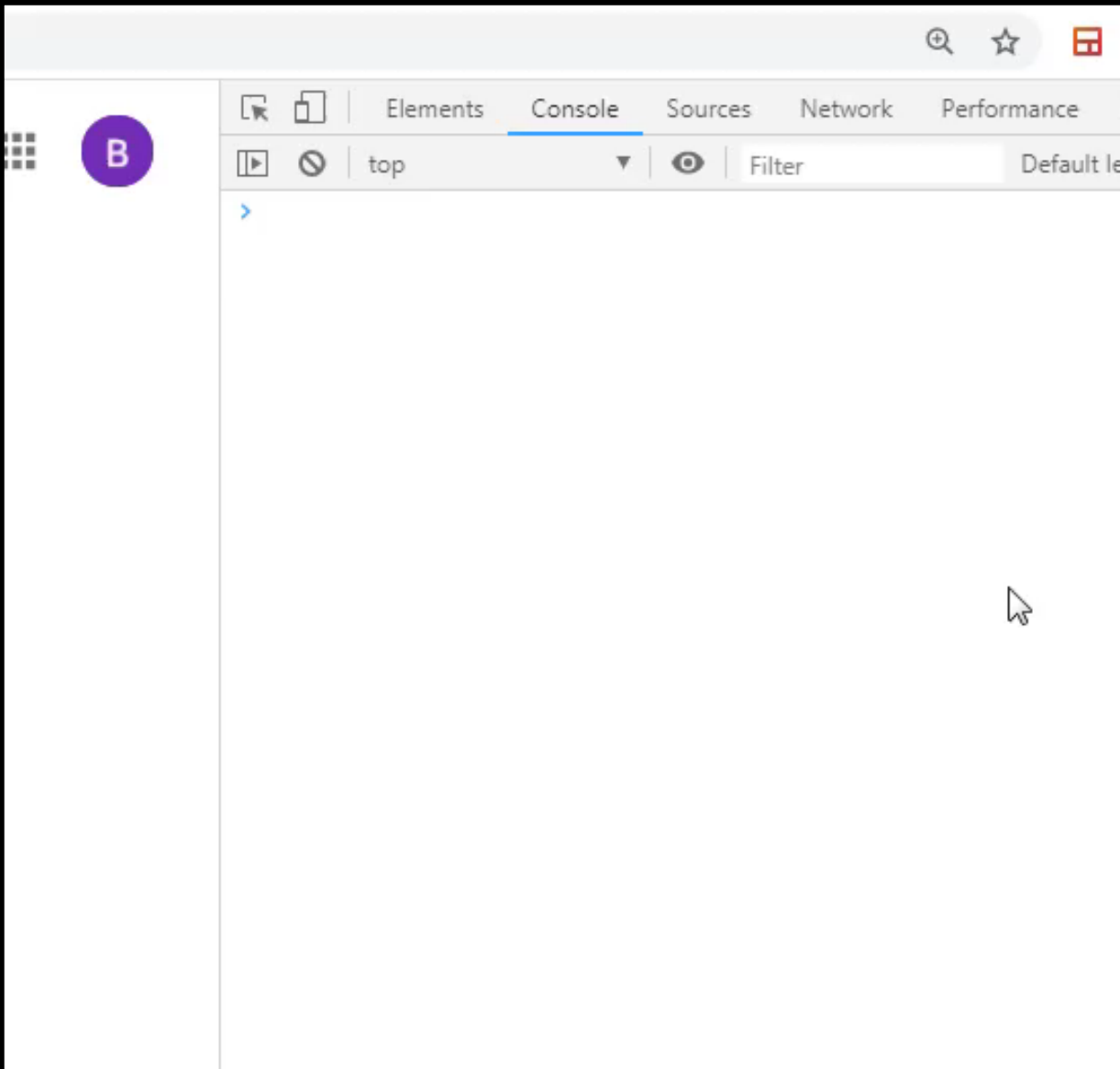


```
function longRunningTask(){  
  //오래 걸리는 작업  
  console.log('작업 끝');  
}
```

```
console.log('시작');  
longRunningTask();  
console.log('다음 작업');
```



# 논블록킹 방식 예제



```
function longRunningTask(){  
  //오래 걸리는 작업  
  console.log('작업 끝');  
}
```

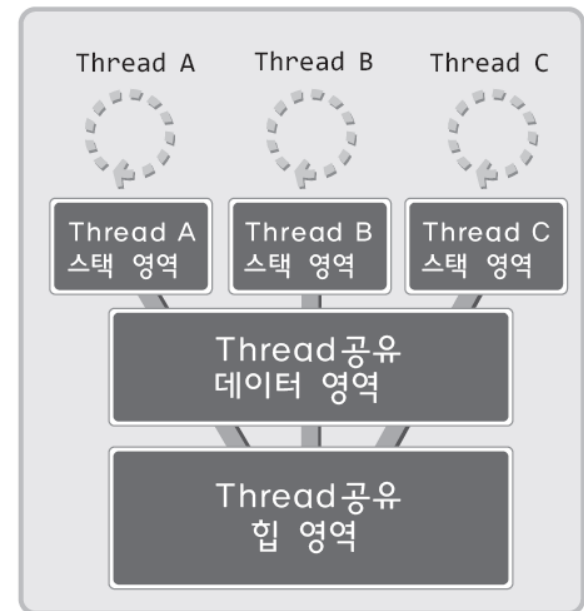
```
console.log('시작');  
setTimeout(longRunningTask,0);  
console.log('다음 작업');
```

setTimeout(callback, 0)  
: HTML5 브라우저는 4ms, 노드는 1ms 시간  
지연이 존재

### 3. 프로세스 vs 스레드

- 프로세스와 스레드
  - 프로세스: 운영체제에서 할당하는 작업의 단위, 프로세스 간 자원 공유X
  - 스레드: 프로세스 내에서 실행되는 작업의 단위, 부모 프로세스 자원 공유
- 노드 프로세스는 멀티 스레드이지만 직접 다룰 수 있는 스레드는 하나이기 때문에 싱글 스레드라고 표현(노드 10에서 멀티 스레드 기능이 추가되었지만 불안정)
- 노드는 멀티 스레드 대신 멀티 프로세스 활용

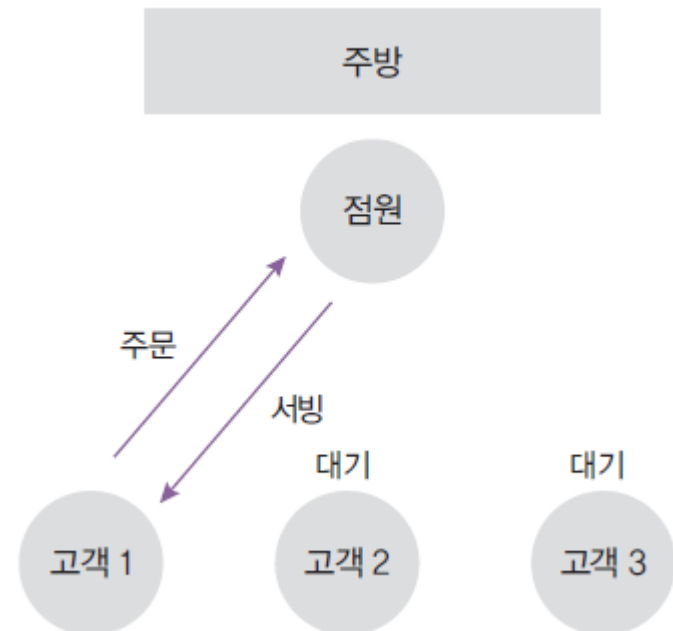
▼ 그림 1-13 스레드와 프로세스



## 4. 싱글 스레드

- 싱글 스레드라 주어진 일을 하나밖에 처리하지 못함
  - 블로킹이 발생하는 경우 나머지 작업은 모두 대기해야 함 → 비효율 발생
- 주방에 비유
  - 점원: 스레드
  - 주문: 요청
  - 서빙: 응답

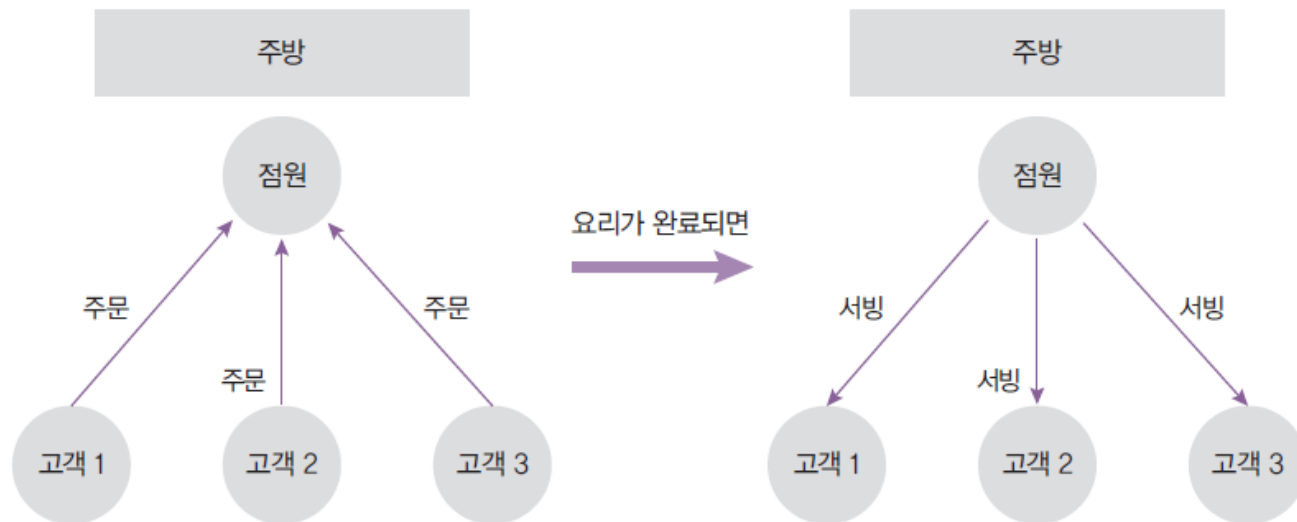
▼ 그림 1-10 싱글 스레드, 블로킹 모델



## 4. 싱글 스레드

- 대신 논블로킹 모델을 채택하여 일부 코드(I/O)를 백그라운드(다른 프로세스)에서 실행 가능
  - 요청을 먼저 받고, 완료될 때 응답함
  - I/O 관련 코드가 아닌 경우 싱글 스레드, 블로킹 모델과 같아짐

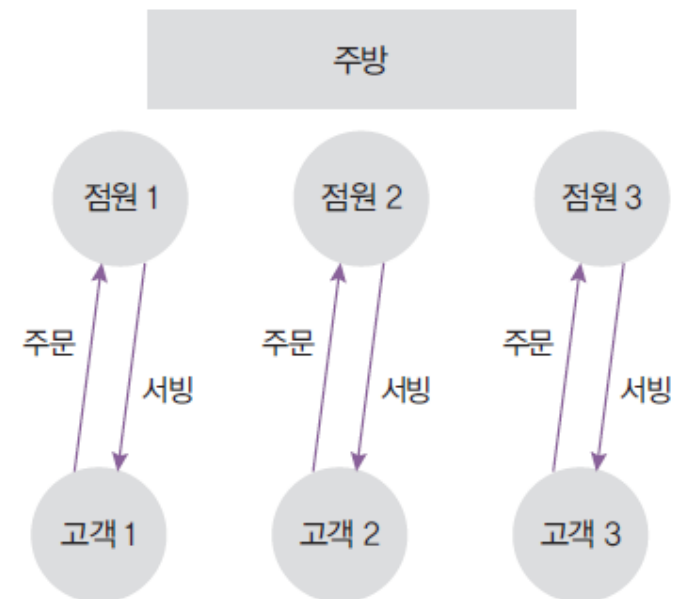
▼ 그림 1-11 싱글 스레드, 논블로킹 모델



## 5. 멀티 스레드 모델과의 비교

- 싱글 스레드 모델은 에러를 처리하지 못하는 경우 멈춤
  - 프로그래밍 난이도 쉽고, CPU, 메모리 자원 적게 사용
- 멀티 스레드 모델은 에러 발생 시 새로운 스레드를 생성하여 극복
  - 단, 새로운 스레드 생성이나 놓고 있는 스레드 처리에 비용 발생
  - 프로그래밍 난이도 어려움
  - 스레드 수만큼 자원을 많이 사용함
- 점원: 스레드, 주문: 요청, 서빙: 응답

▼ 그림 1-12 멀티 스레드, 블로킹 모델

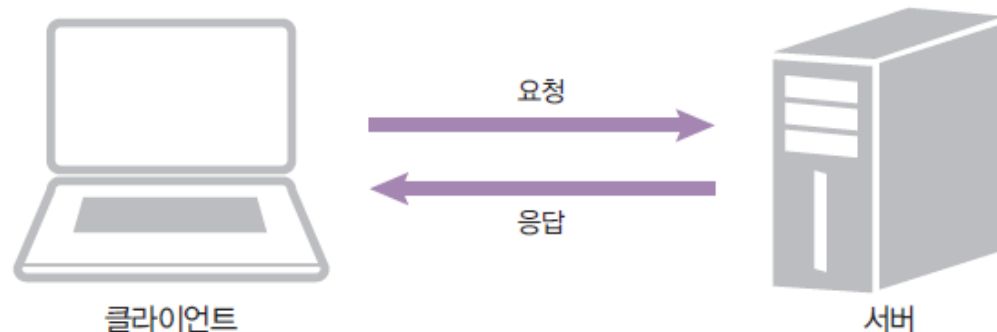


## 1.3 노드의 역할

# 1. 서버로서의 노드

- 서버:
  - 네트워크를 통해 클라이언트에 정보나 서비스를 제공하는 컴퓨터 또는 프로그램
- 클라이언트:
  - 서버에 요청을 보내는 주체(브라우저, 데스크탑 프로그램, 모바일 앱, 다른 서버에 요청을 보내는 서버)
- 예시
  - 브라우저(클라이언트, 요청)가 학교 웹사이트(서버, 응답)에 접속
- 노드 != 서버
- But, 노드는 서버를 구성할 수 있게 하는 모듈(4장에서 설명)을 제공

▼ 그림 1-2 클라이언트와 서버



## 2. 서버로서의 노드

### ● 노드 서버의 장단점

▼ 표 1-1 노드의 장단점

장점	단점
멀티 스레드 방식에 비해 컴퓨터 자원을 적게 사용함	싱글 스레드라서 CPU 코어를 하나만 사용함
I/O 작업이 많은 서버로 적합	CPU 작업이 많은 서버로는 부적합
멀티 스레드 방식보다 쉬움	하나뿐인 스레드가 멈추지 않도록 관리해야 함
웹 서버가 내장되어 있음	서버 규모가 커졌을 때 서버를 관리하기 어려움
자바스크립트를 사용함	어중간한 성능
JSON 형식과 호환하기 쉬움	

- CPU 작업을 위해 AWS Lambda나 Google Cloud Functions같은 별도 서비스 사용
- 페이팔, 넷플릭스, 나사, 월마트 등에서 메인 또는 서브 서버로 사용



### 3. 서버 외의 노드

- 자바스크립트 런타임이기 때문에 용도가 서버에만 한정되지 않음
- 웹, 모바일, 데스크탑 애플리케이션에도 사용
  - 웹 프레임워크: Angular, React, Vue, Meteor 등
  - 모바일 앱 프레임워크: React Native, Ionic, NativeScript
  - 데스크탑 개발 도구: Electron(Atom, Slack, VSCode, Discord 등 제작)
- 위의 프레임워크가 노드 기반으로 동작함

▼ 그림 1-15 노드 기반의 개발 도구

