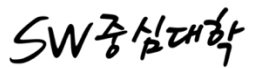
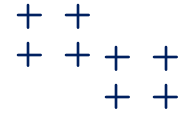
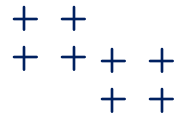


# 전북대 소중해유(You)

## Node.js 소개: Node.js의 기능과 사용 사례



# 목차

1/ Node.js란?

2/ Node.js 기술 및 실행 원리

3/ Node.js 사용 예제

## Node.js란?

- 본래 Javascript는 태생이 브라우저에서만 동작하도록 태어남
- 크롬, 파이어폭스, 사파리 같은 브라우저에는 자바스크립트 엔진이 있어 자바스크립트를 실행할 수 있음
- 하지만 브라우저를 벗어난 환경에서는 자바스크립트를 사용할 수 없었음
  - => 브라우저 외의 환경에서도 js를 쓰자!
  - => Node.js 탄생!

## Node.js란?

### • Node.js란?

- 서버사이드 측 Javascript 기반 프레임워크
- 비동기 이벤트 기반 아키텍처로 높은 성능 제공 (**Non-blocking I/O** 방식)
- CommonJS 모듈 시스템 (자바스크립트의 모듈화 지원)

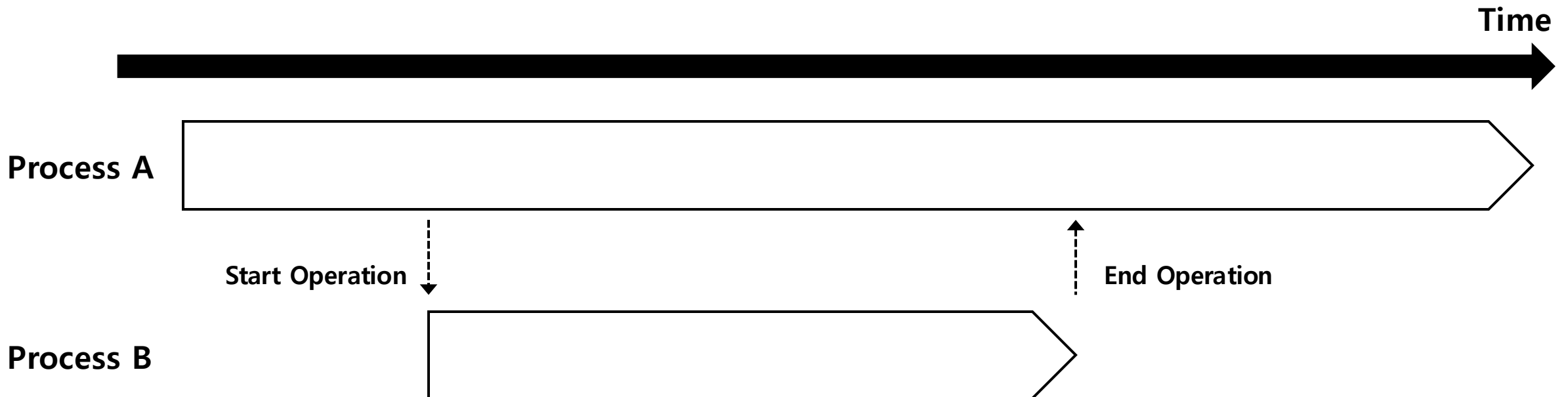
### • 역사

- 2009년 라이언 달에 의해 오픈소스로 공개됨
- 브라우저 밖에서 자바스크립트를 실행할 수 있도록 구글의 V8 엔진을 사용
- I/O에 대한 새로운 관점을 제시! → 프로그래밍 발전에 중요한 기여!

## Node.js란?

### • Non-blocking?

- 어떤 작업을 시작했을 때, 그 작업이 완료될 때까지 기다리지 않고, 다른 작업을 계속해서 수행할 수 있는 방법
- 시스템의 효율성을 높이고, 동시에 여러 작업을 수행할 수 있게 하여, 최근, Non-blocking은 주로 I/O 작업이나 네트워크 요청과 같이 시간이 오래 걸릴 수 있는 작업에 많이 사용 됨.



## Node.js란? (Non-Blocking)

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

setTimeout(() => {
  console.log('This is executed after 2 seconds');
}, 2000);

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

Console 예제

```
const fs = require('fs');

console.log('Start reading file');

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) {
    console.error('Error reading file:', err);
    return;
  }
  console.log('File content:', data);
});

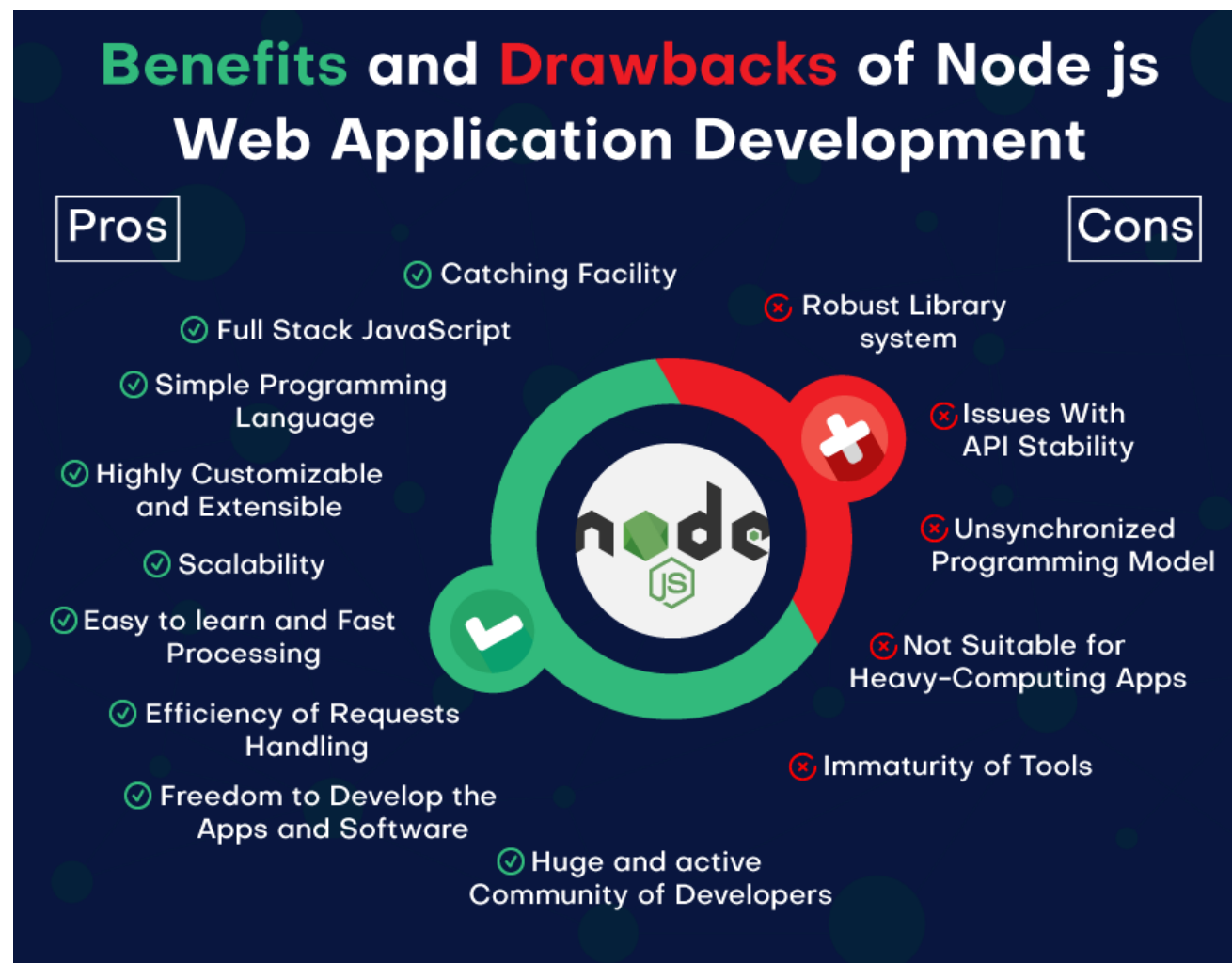
console.log('End of script');
```

File I/O 예제

## Node.js란? (Non-Blocking)

	Blocking	Non-Blocking
<b>Synchronous (동기)</b>	<ol style="list-style-type: none"><li>1. A 작업을 시작</li><li>2. B 작업을 시작</li><li>3. B 작업이 완료될 때까지 기다림</li><li>4. B 작업 완료 후 A 작업 시작</li></ol>	<ol style="list-style-type: none"><li>1. A 작업을 시작</li><li>2. A 작업 완료 여부와 관계없이 프로그램의 다른 부분 실행 가능</li></ol>
<b>Async (비동기)</b>	<ol style="list-style-type: none"><li>1. A 작업을 시작</li><li>2. A 작업이 완료될 때까지 기다림</li><li>3. 동시에 다른 작업 시작 가능</li><li>4. A 작업 완료 후 결과 처리</li></ol>	<ol style="list-style-type: none"><li>1. A 작업을 시작</li><li>2. A 작업 완료 여부와 관계없이 프로그램의 다른 부분 실행 가능</li><li>3. A 작업 완료 여부와 관계없이 B 작업 시작</li><li>4. A 작업 완료 후 결과 처리</li></ol>

## Node.js의 장단점





## Node.js란?

### • Node.js의 장점

- 가벼운 프레임워크 (간단한 코드로 서버를 실행할 수 있음)
- 빠른 속도 및 트래픽 제어 가능
  - 25만 동시 접속 처리 가능
  - 초당 20,000번의 트래픽 처리 가능
  - Apache에 비해 최소 3배 이상 속도가 빠름
- 대규모 오픈소스 및 라이브러리 (모듈) 지원

### • Node.js의 단점?

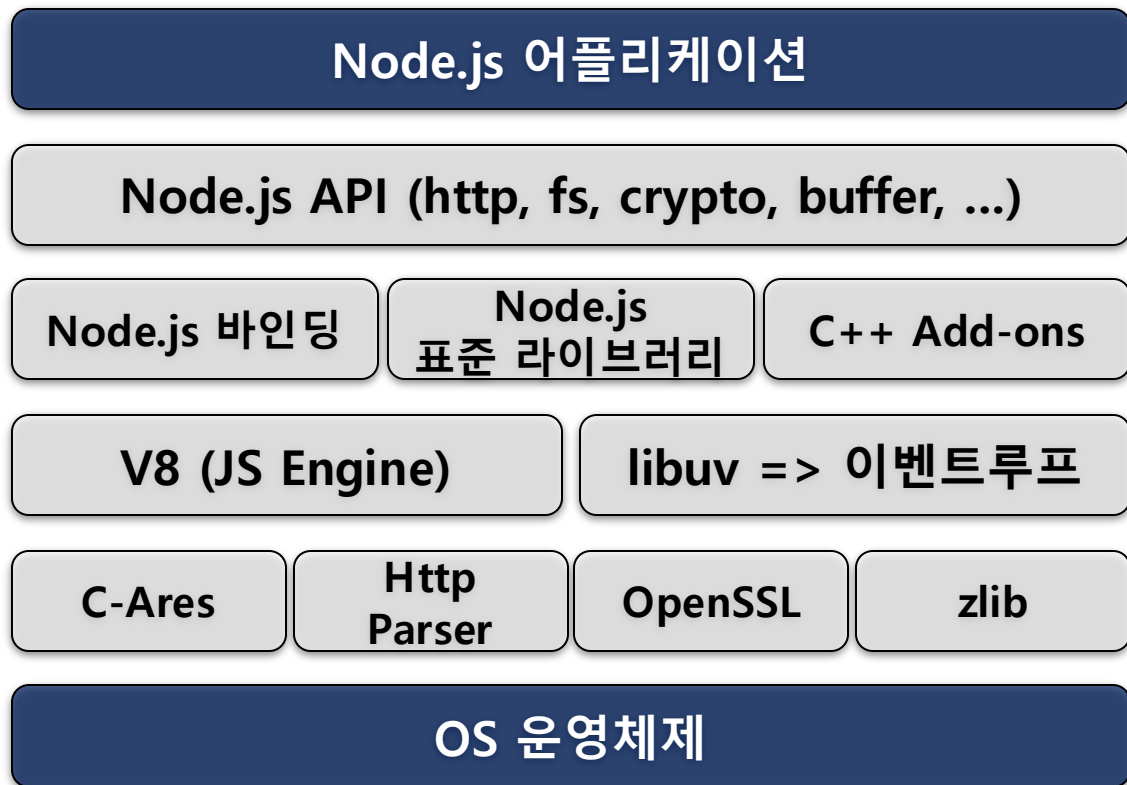
- Callback style의 구현 어려움/ 속도 자체가 CPU 개수 + Memory 용량 등에 선형적으로 영향을 받지 않음.

## Node.js의 장단점

장점	단점
<ul style="list-style-type: none"><li>• 비동기 이벤트 기반 I/O 사용 =&gt; 여러 요청을 다루기 용이 함</li><li>• Javascript 기반 =&gt; 확장성/ 배우기 쉬움/ 코드 재사용</li><li>• 개발 생태계가 잘 구성되어 있으며, <b>package manager</b>를 사용 할 수 있음 =&gt; 진입장벽 낮음/ 개발 난이도 낮음</li><li>• V8 엔진이 JIT 컴파일러 기반으로 서버 기동이 매우 빠름 (vs SpringBoot/ Flask 등)</li></ul>	<ul style="list-style-type: none"><li>• <b>CPU를 하나만 사용</b> (멀티코어 사용을 위해서는 별도 작업 필요)</li><li>• 비동기 작업을 지원하지 않는 I/O 요청에서는 CPU 관리에 주의</li><li>• Callback 사용 주의해야 함! (가독성/ 디버깅 등)</li><li>• 이벤트 기반 프레임워크로, 상대적으로 배우기 어려울 수 있음</li></ul>

## Node.js란?

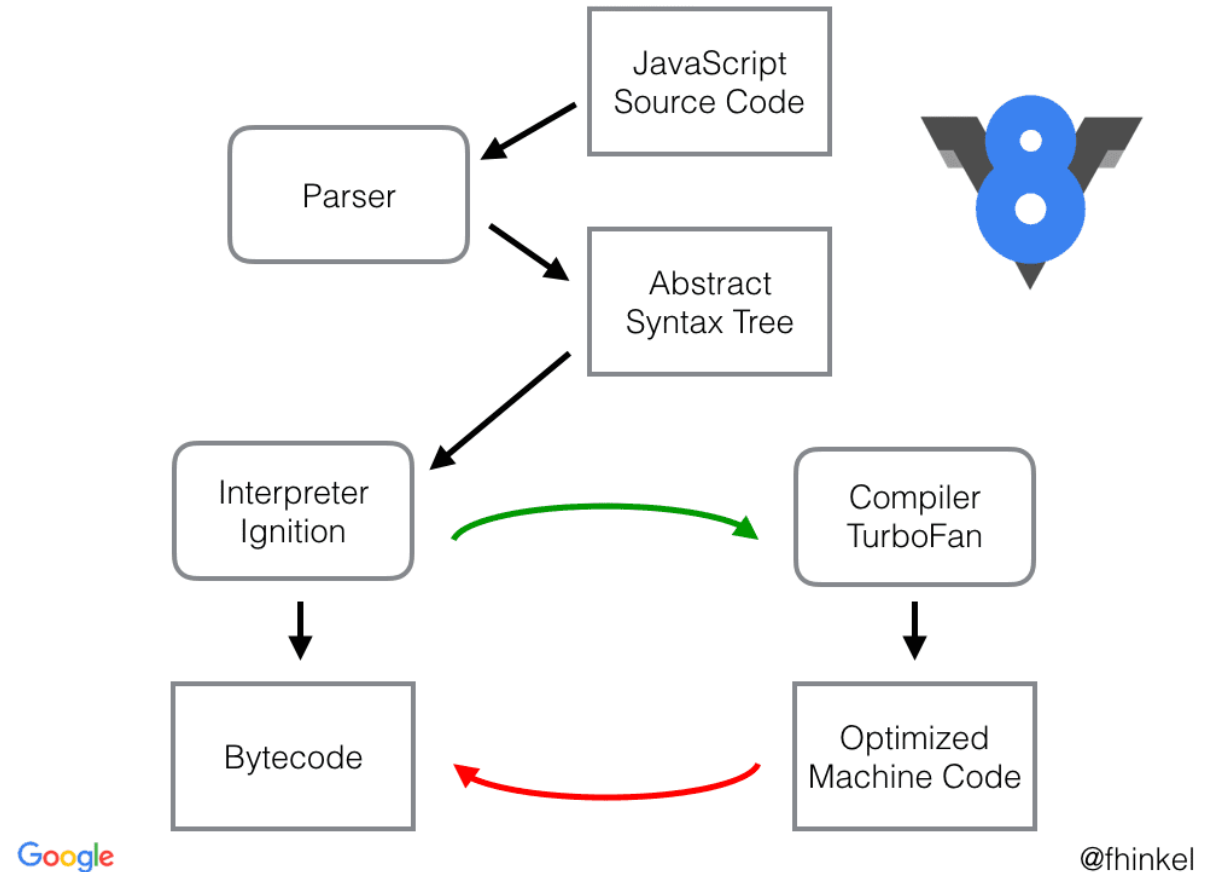
### • Node.js의 구성 요소



- **Node.js API (Javascript)**
  - 자바스크립트 라이브러리
- **Node.js 바인딩 (Javascript & C++)**
  - 자바스크립트에서 C/C++ 함수 호출을 도와줌
- **Node.js 표준 라이브러리 (C++)**
  - OS와 관련된 함수들, 타이머, 파일시스템, 네트워크 등 사용
- **C/C++ 에드온**
  - Node.js에서 C/C++ 소스 실행을 도와줌
- **V8 (C++)**
  - Javascript Engine으로, JS Parsing 및 interpreter, compiler 최적화
- **libuv (C++)**
  - 비동기 I/O에 초점을 맞춘 멀티 플랫폼을 지원하는 라이브러리

### V8 엔진

- V8은 C++로 만든 오픈 소스 자바스크립트 엔진
- 엔진은 parser, compiler, interpreter, garbage collector, call-stack, heap으로 구성
- Ignition 및 TurboFan 등 최신 기법 적용
- 최근 잘 짜여진 Javascript는 C++에 근사하는 성능을 낼 수 있다고 함!! (인터프리터 언어인데?!)

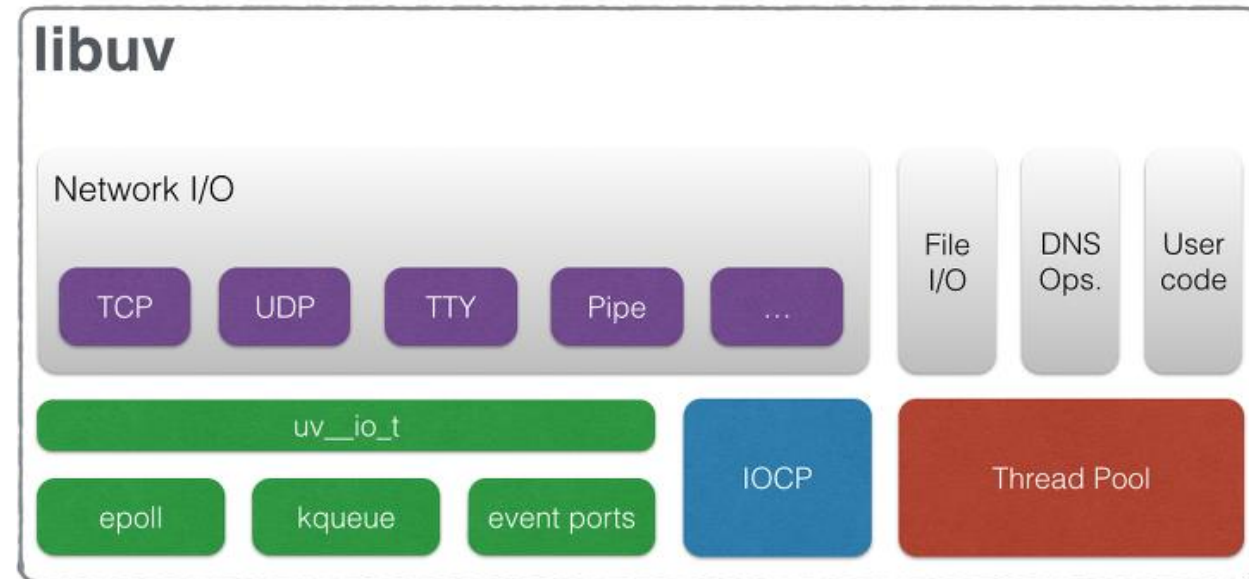


## Node.js 기술 및 실행 원리

### libuv

- libuv: 비동기 입출력, 이벤트 기반에 초점을 맞춘 라이브러리로, libuv는 비동기, 논블로킹 스타일을 사용 => **커널의 비동기IO를 이용**
- 윈도우와 리눅스 플랫폼의 비동기 IO인터페이스를 추상화 => 크로스플랫폼 역할 수행
- 즉, libuv를 사용하면 각 플랫폼의 가장 빠른 비동기IO인터페이스로 통일된 코드로 돌릴 수 있음!
- Node.js에서 HTTP, 파일, 소켓 통신, I/O 등 자바스크립트에 없는 기능 => libuv(C++) 라이브러리를 사용!

<http://docs.libuv.org>

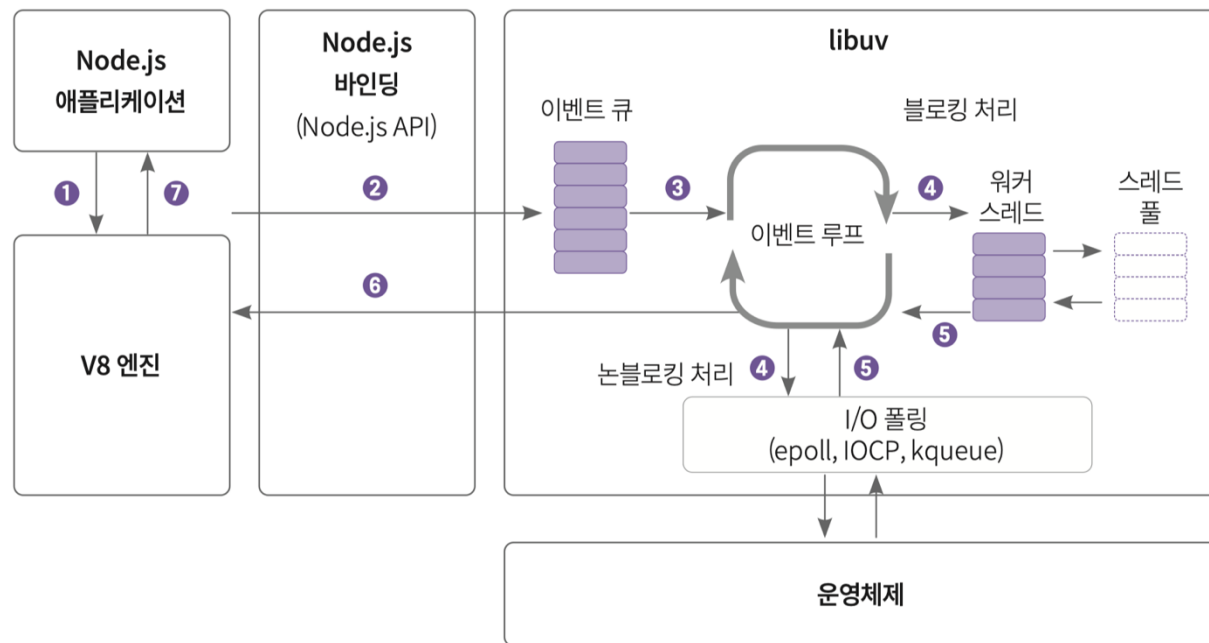


## Node.js 기술 및 실행 원리

### Node.js 아키텍처

- 1 애플리케이션에서 요청이 발생 후, V8 엔진은 자바스크립트 코드로 된 요청을 바이트 코드나 기계어로 변경
- 2 자바스크립트로 작성된 Node.js의 API는 C++로 작성된 코드를 사용
- 3 V8 엔진은 이벤트 루프로 libuv를 사용하고 전달된 요청을 libuv 내부의 이벤트 큐 추가
- 4 이벤트 큐에 쌓인 요청은 이벤트 루프에 전달되고, 운영체제 커널에 비동기 처리를 전달. 운영체제 내부적으로 비동기 처리가 힘든 경우(DB, DNS 룩업, 파일 처리 등)는 워커 스레드에서 처리
- 5 운영체제의 커널 또는 워커 스레드가 완료한 작업은 다시 이벤트 루프로 전달
- 6 이벤트 루프에서는 콜백으로 전달된 요청에 대한 완료 처리를 하고 전달
- 7 완료 처리된 응답을 Node.js 애플리케이션으로 전달

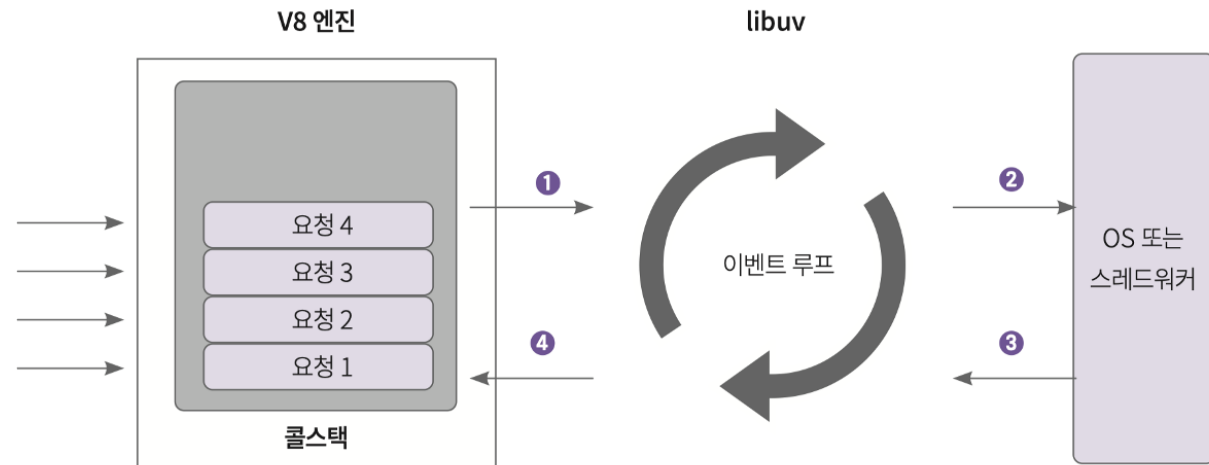
▼ Node.js 아키텍처



# Node.js 이벤트 처리

- 1 V8에 쌓이고 I/O 처리가 필요한 코드는 이벤트 루프로 전달
- 2 이벤트 루프에서는 말그대로 루프를 실행하면서 운영 체제 또는 스레드 워커에 I/O 처리를 전달/수행
- 3 스레드 워커와 운영체제는 받은 요청에 대한 결과를 이벤트 루프로 돌려주고 (callback)
- 4 이벤트 루프에서는 결과값에 대한 코드를 콜 스택에 다시 추가

▼ Node.js의 이벤트 기반 아키텍처



<https://wikidocs.net/223233>

## Node.js 사용 예제 (기초 http 서버)

server.js

```
1  const http = require('http');
2
3  const server = http.createServer((req, res) => {
4    res.statusCode = 200;
5    res.setHeader('Content-Type', 'text/plain');
6    res.end('Hello, this is your Node.js server!\n');
7  });
8
9  const port = 3000;
10 server.listen(port, () => {
11   console.log(`Server running at http://localhost:${port}/`);
12 });
13
```

1. Node.js 설치 (<https://nodejs.org>)
2. server.js 파일 만들기/저장
3. "node server.js " 명령어 실행
4. 웹브라우저 (크롬, 엣지, 사파리 등)에서 <http://localhost:3000>를 통해 접속
5. 내용 확인!



## Node.js 사용 예제 (기초 http 서버)

server.js

```
1  const http = require('http');
2
3  const server = http.createServer((req, res) => {
4    res.statusCode = 200;
5    res.setHeader('Content-Type', 'text/plain');
6    res.end('Hello, this is your Node.js server!\n');
7  });
8
9  const port = 3000;
10 server.listen(port, () => {
11   console.log(`Server running at http://localhost:${port}/`);
12 });
13
```

← → ↻ 🏠 ⓘ localhost:3000

Hello, this is your Node.js server!

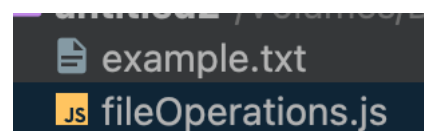
1. Node.js 설치 (<https://nodejs.org>)
2. server.js 파일 만들기/저장
3. "node server.js " 명령어 실행
4. 웹브라우저 (크롬, 엣지, 사파리 등)에서 <http://localhost:3000>를 통해 접속
5. 내용 확인!

## Node.js 사용 예제 (파일 읽기 쓰기)

### fileOperations.js

```
1  const fs = require('fs');
2
3  // 파일 쓰기
4  fs.writeFile('example.txt', 'This is an example text.', (err) => {
5    if (err) throw err;
6    console.log('File has been saved!');
7
8    // 파일 읽기
9    fs.readFile('example.txt', 'utf8', (err, data) => {
10      if (err) throw err;
11      console.log('File content:', data);
12    });
13  });
14
```

```
(base) ksl@MacStudio untitled2 % node fileOperations.js
File has been saved!
File content: This is an example text.
```



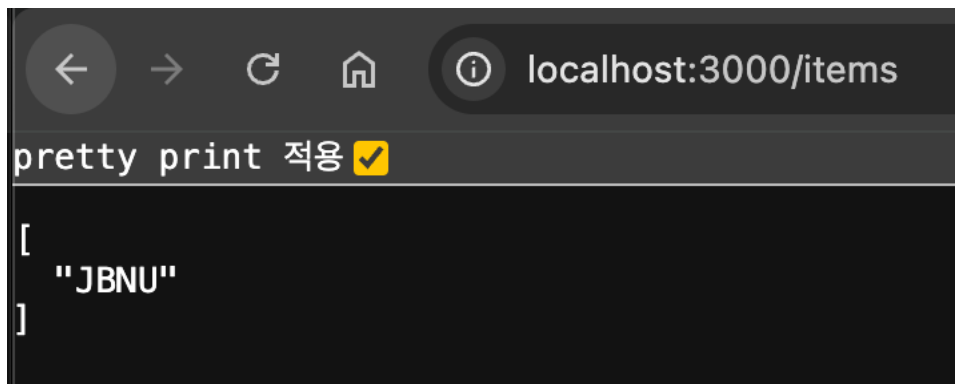
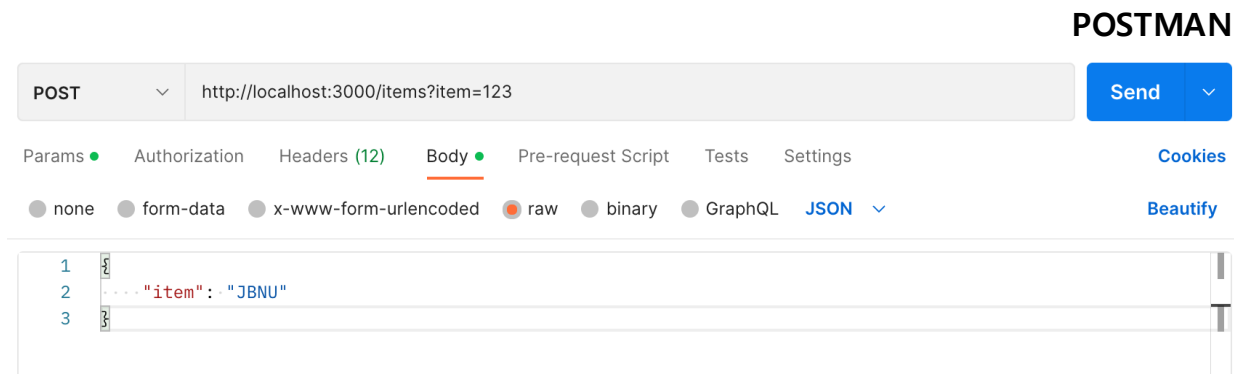
A screenshot of a file explorer window showing two files: 'example.txt' with a document icon and 'fileOperations.js' with a JavaScript icon.

## Node.js 사용 예제 (간단한 REST API 만들기)

api.js

```
1 const express = require('express');
2 const bodyParser = require('body-parser');
3
4 const app = express();
5 app.use(bodyParser.json());
6
7 let items = [];
8
9 app.get('/items', (req, res) => {
10   res.json(items);
11 });
12
13 app.post('/items', (req, res) => {
14   const newItem = req.body.item;
15   items.push(newItem);
16   res.status(201).json({ message: 'Item added successfully', item: newItem });
17 });
18
19 const port = 3000;
20 app.listen(port, () => {
21   console.log(`API server running at http://localhost:${port}/`);
22 });
23
```

```
npm init -y
npm install express body-parser
```

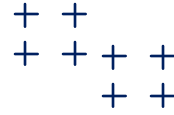


## Node.js 사용 예제 (간단한 REST API 만들기)

asyncExample.js

```
1 function asyncTaskPromise() {
2     return new Promise((resolve, reject) => {
3         setTimeout(() => {
4             resolve('Task completed with Promise');
5         }, 2000);
6     });
7 }
8
9 asyncTaskPromise()
10 .then((message) => {
11     console.log(message);
12 })
13 .catch((error) => {
14     console.error(error);
15 });
16
```

```
1 function asyncTask(callback) {
2     setTimeout(() => {
3         callback('Task completed');
4     }, 2000);
5 }
6
7 asyncTask((message) => {
8     console.log(message);
9 });
10
```



# 감사합니다.

- 본 온라인 콘텐츠는 2024년도 과학기술 정보통신부 및 정보통신기획평가원의 'SW중심대학사업' 지원을 받아 제작되었습니다.
- 본 결과물의 내용을 전재할 수 없으며, 인용(재사용)할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원이 지원한 'SW중심대학'의 결과물이라는 출처를 밝혀야 합니다.

