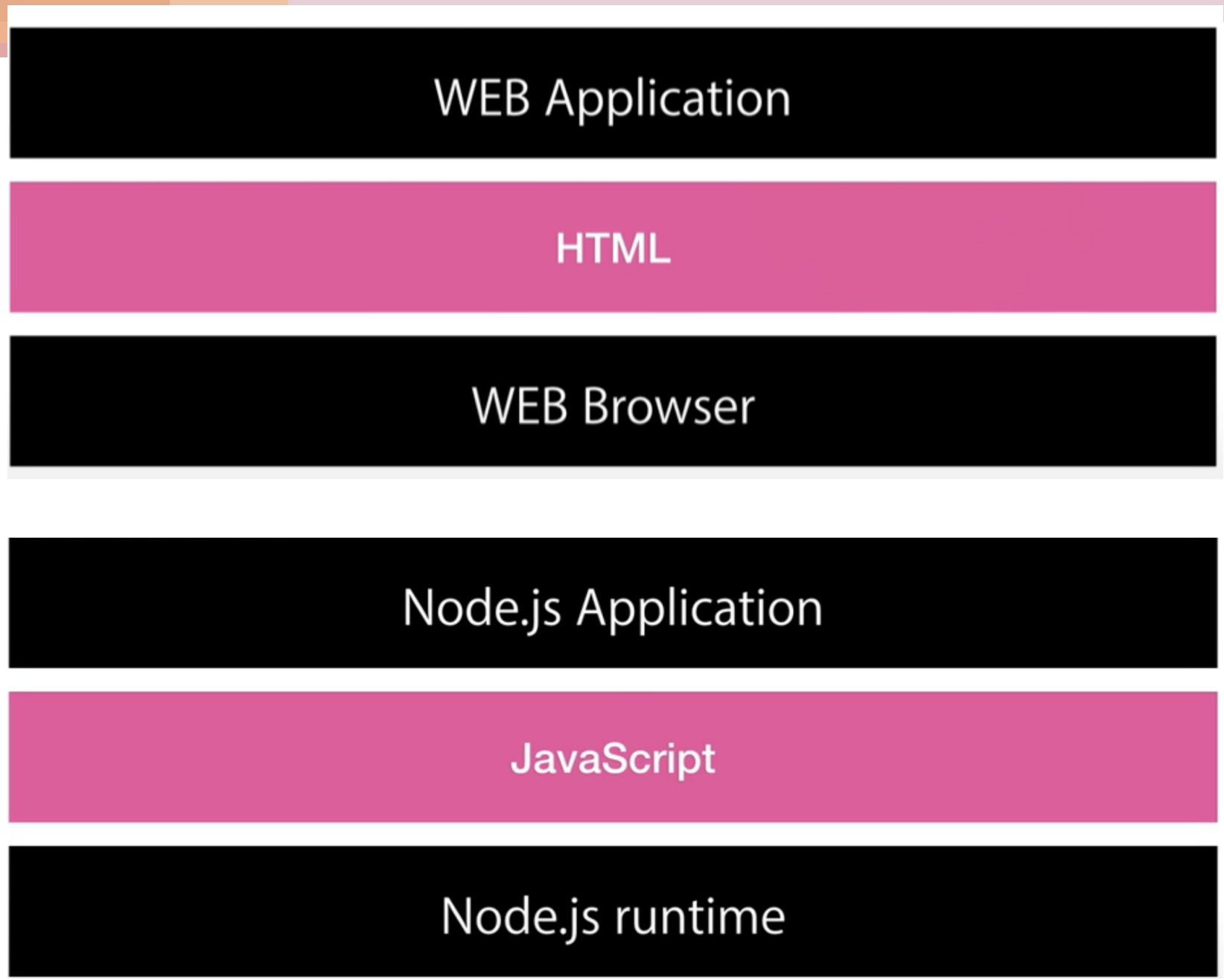
Nodejs - Introduction

Sung-Dong Kim,
School of Computer Engineering,
Hansung University

Contents

Background


WEB Application
HTML
WEB Browser
Node.js Application
JavaScript
Node.js runtime

Introudction

Introduction

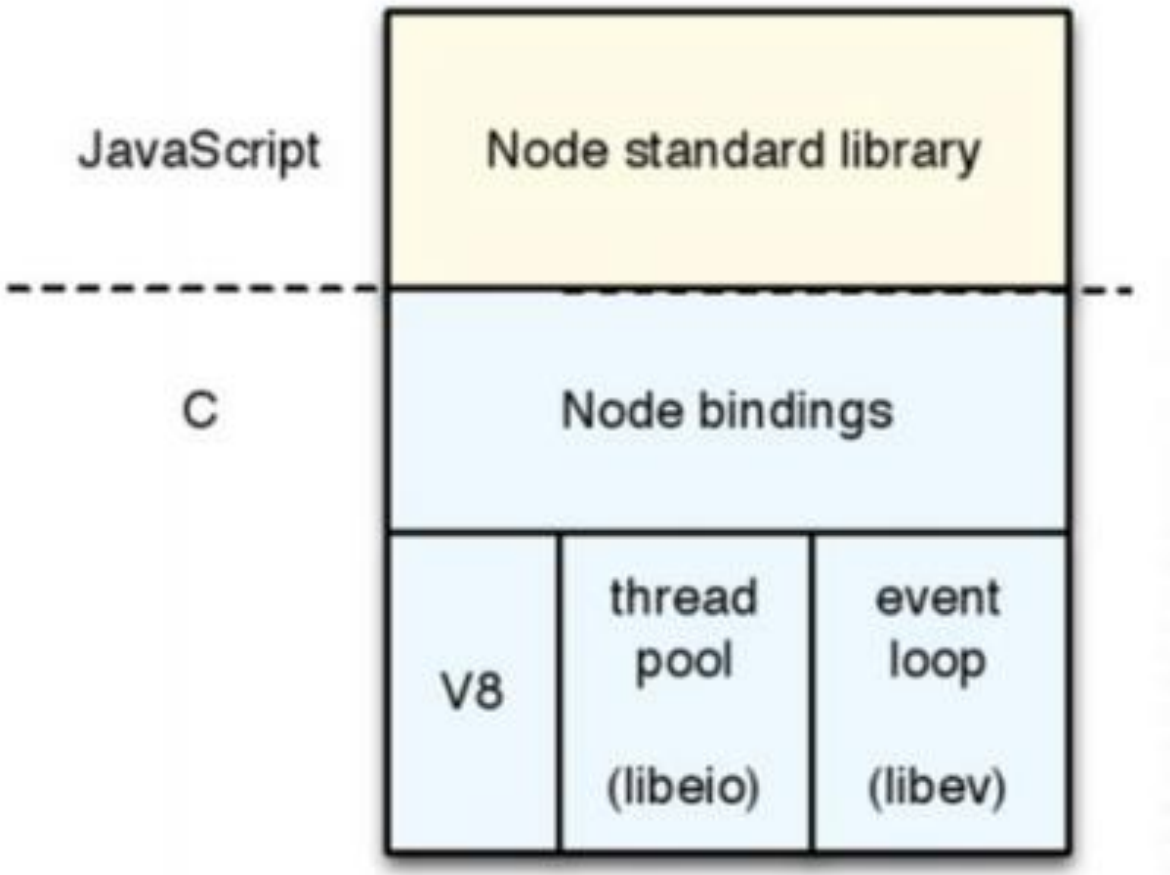
- 2009, Ryan Dhal – 네트워크를 이용하는 application 개발을 위한 platform
- 크롬 V8 JavaScript 엔진 (= JavaScript runtime) 포함
- 웹 브라우저에서 사용하는 Javascript 실행 엔진을 다른 곳(server, pc, …)에서 사용할 수 있게 됨
- 여러 platform에서 동작 (Windows, Linux, Mac OS X, ...)
- Server에서 JavaScript 이용 가능

Introduction

- 비동기(asynchronous) IO + 이벤트 기반
- single-thread, non-blocking
- 함수형 프로그래밍 지향 → 복잡한 기능 개발할 때, 명령형 언어에 대한 좋은 대안
- 설치: https://nodejs.org

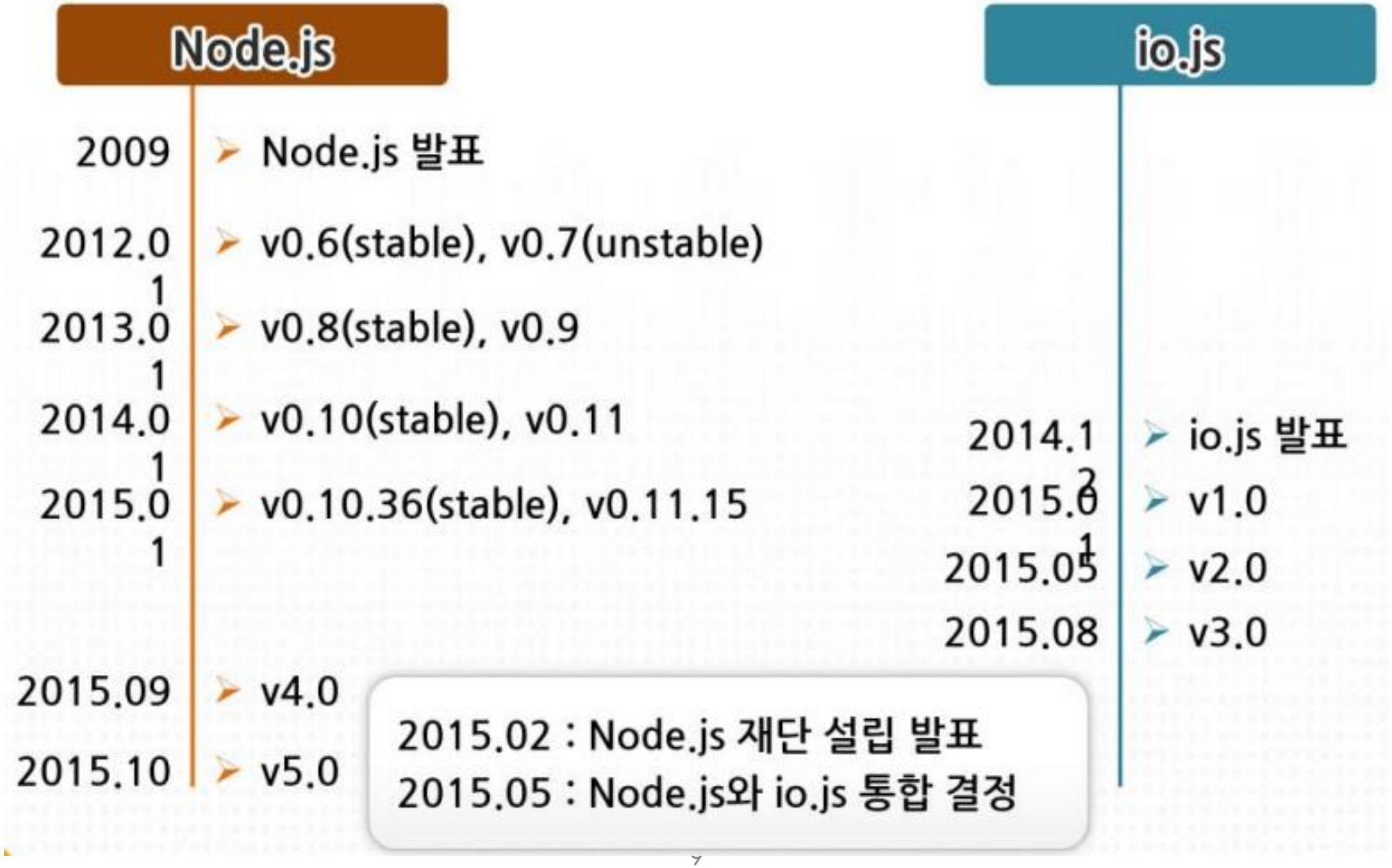
Introduction

- 아키텍처



Introduction

- 역사



Introduction

- 버전 구성과 지원
  - 두 단계로 진행
  - 기존: 짝수 버전 (stable)과 홀수 버전 (unstable)
  - 4.x이후: Stable, LTS
  - LTS (Long Term Support)
    - 짝수 버전 6개월 이후 LTS로 전환
    - LTS 18개월, 그 후 12개월 maintaia 상태
    - 매년 새로운 메이저 버전의 LTS 시작

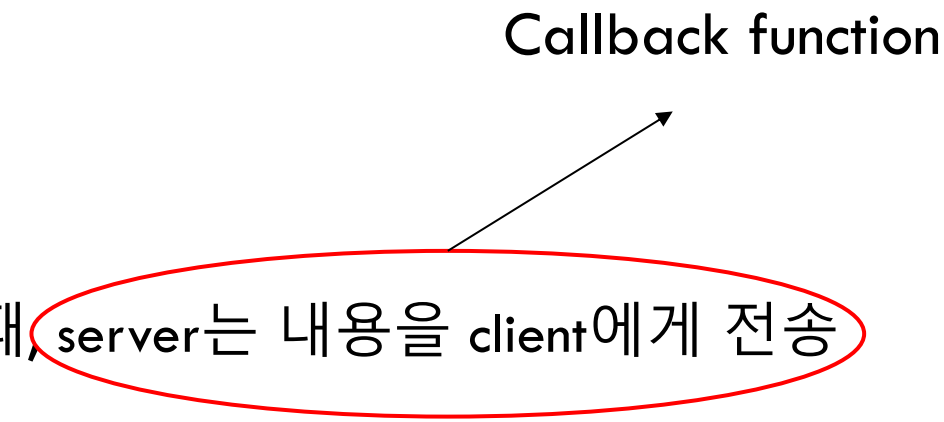
비동기 입출력

비동기 입출력

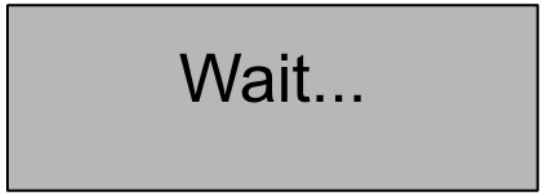
- file request 과정 예
  - file system에 작업 전송
  - file system이 file을 열고 읽을 동안 기다림 = 동기식 (synchronous)
  - client에게 내용 전송
  - 다음 request를 처리할 준비가 됨

비동기 입출력

- Node.js가 request를 다루는 방법
  - file system에게 작업 전송
  - 다음 request를 처리할 준비가 됨
  - file system이 file을 열어서 읽을 때, server는 내용을 client에게 전송
- 종합
  - 요청 처리가 끝나기 전에 다른 요청을 동시에 처리
  - 파일 읽기 요청 전에 callback function 등록
  - 읽기 작업이 끝나면 callback function 자동 호출



```
var contents = file.read('a.txt');
```




```
doShow(contents);
var result = doAdd(10, 10);
```

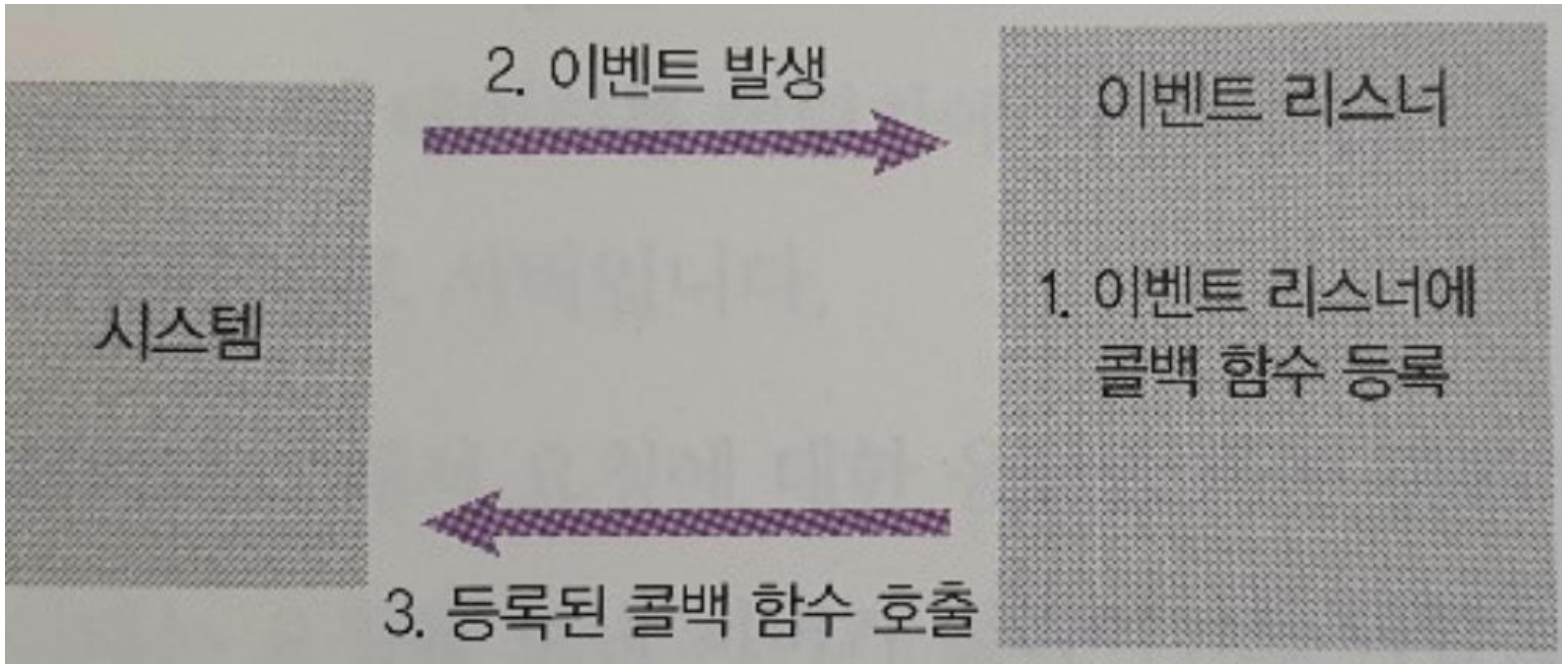
```
file.read('a.txt', function(contents) {
    doShow(contents);
});

var result = doAdd(10, 10);
```

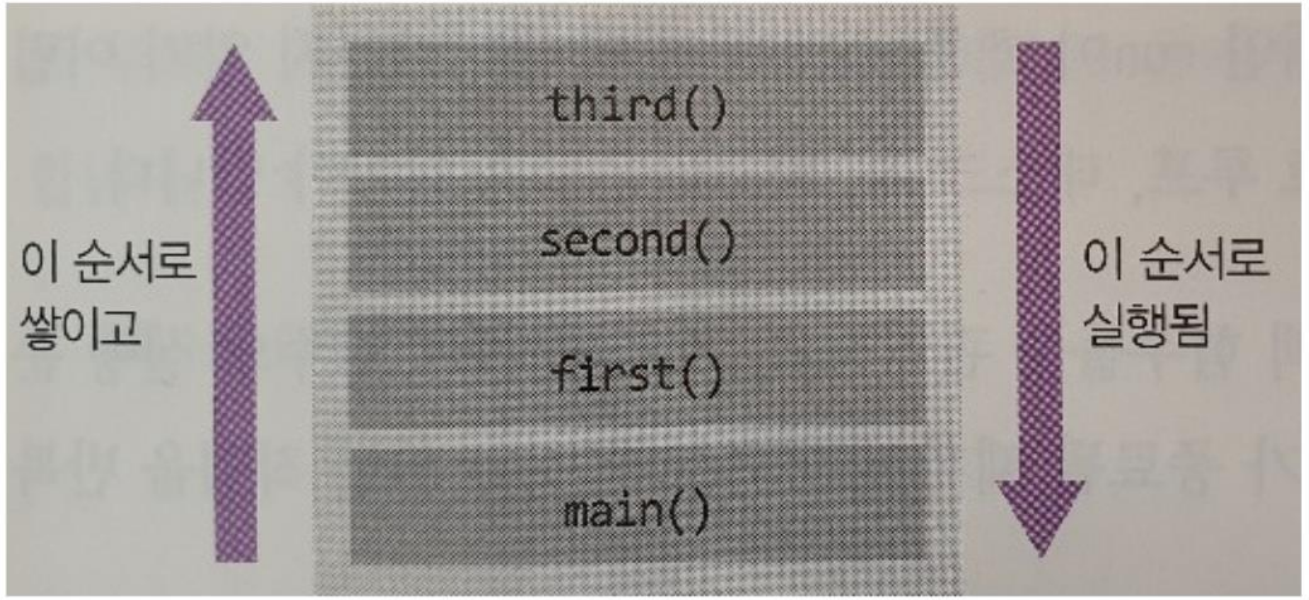
이벤트 기반

이벤트 기반

- 이벤트가 발생할 때 미리 지정된 작업을 수행
- 이벤트 리스너에 callback function 등록 = 이벤트가 발생할 때 무엇을 할지 미리 등록하는 것 !!!
- 호출 스택



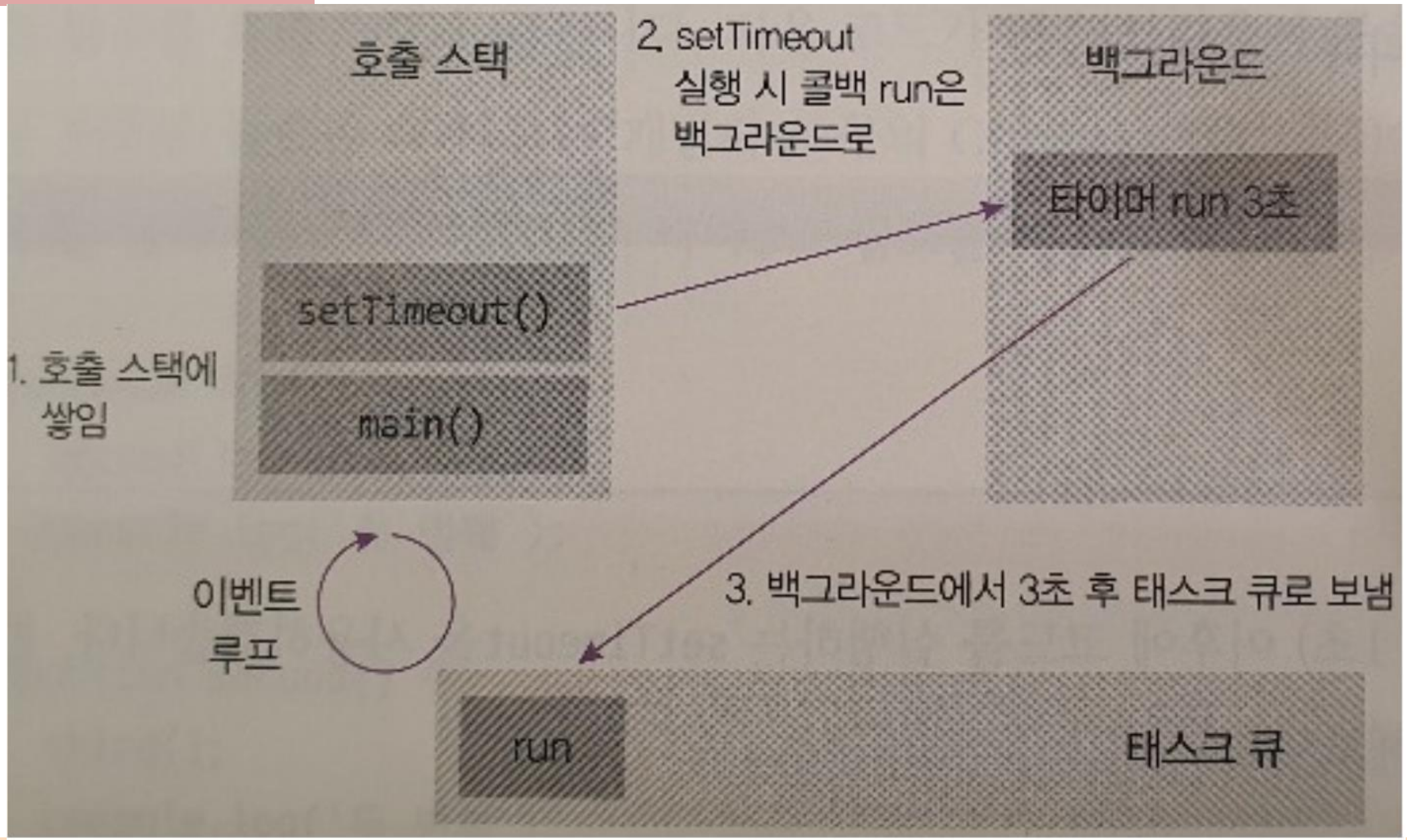
```
function first() {
    second();
    console.log('first');
}
function second() {
    third();
    console.log(second);
}
function third() {
     console.log('third');
}
first();
```

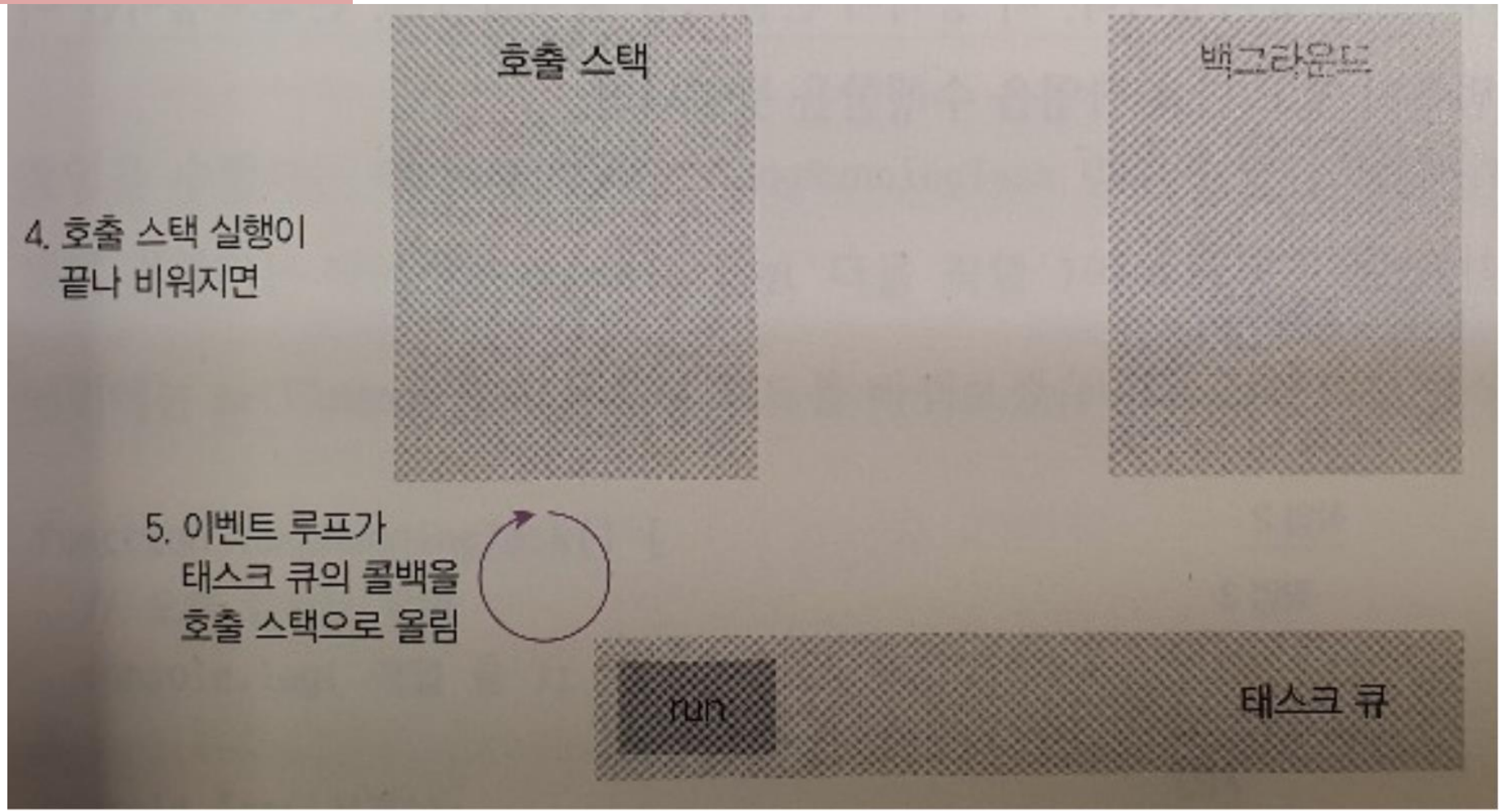


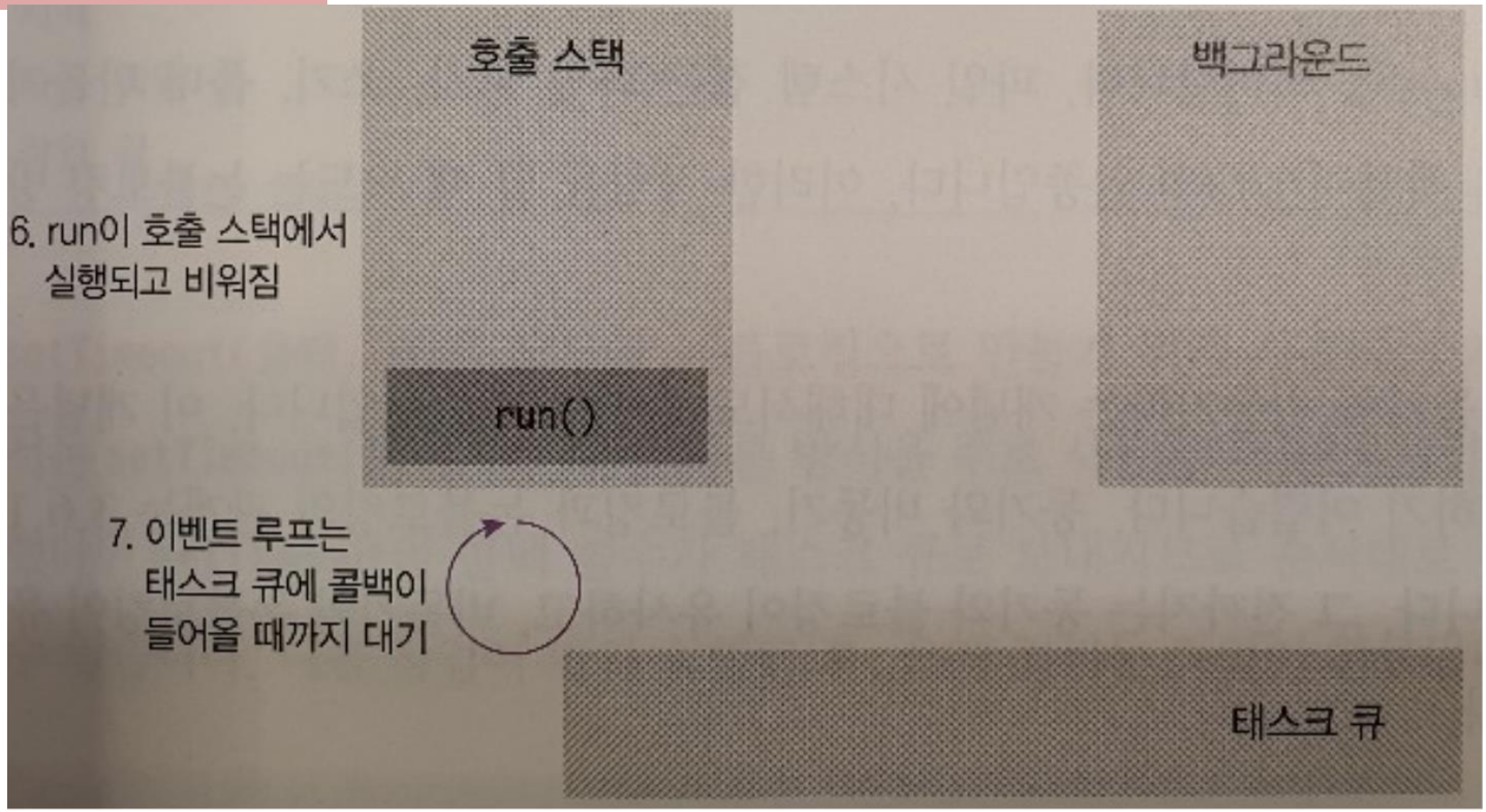
이벤트 기반

- 이벤트 루프 (event loop)
  - 이벤트 발생 시 호출할 callback 함수 관리
  - 호출된 callback 함수 실행 순서 결정
  - Node 종료까지 이벤트 처리를 위한 작업 반복 = loop
- 태스크 큐: 이벤트 발생 후 호출되어야 할 callback 함수들이 기다리는 공간 (= callback queue)
- 백그라운드: 타이머, I/O 작업 콜백, 이벤트 리스너들이 대기하는 곳

```
function run() {
    console.log('execute after 3 seconds');
}
console.log('start');
setTimeout(run, 3000);
console.log('end');
```


호출 스택
2. setTimeout
실행 시 콜백 run은
백그라운드로
백그라운드
타이머 run 3초
setTimeout()
main()
1. 호출 스택에
쌓임
이벤트
루프
3. 백그라운드에서 3초 후 태스크 큐로 보냄
run
태스크 큐


호출 스택
백그라운드
4. 호출 스택 실행이
끝나 비워지면
5. 이벤트 루프가
태스크 큐의 콜백을
호출 스택으로 올림
run
태스크 큐


호출 스택
백그라운드
6. run이 호출 스택에서
실행되고 비워짐
run()
7. 이벤트 루프는
태스크 큐에 콜백이
들어올 때까지 대기
태스크 큐

Non-Blocking I/O

특징

- 오래 걸리는 함수를 백그라운드로 보내고 다음 코드 먼저 실행
- 백그라운드에 보낸 함수가 태스크 큐를 거쳐 호출 스택으로 올라오기를 기다림
- 이전 작업에 완료될 때까지 멈추지 않고 다음 작업을 수행함
- 파일 시스템 접근, 네트워크 요청 등의 작업을 할 때 non-blocking 방식으로 동작

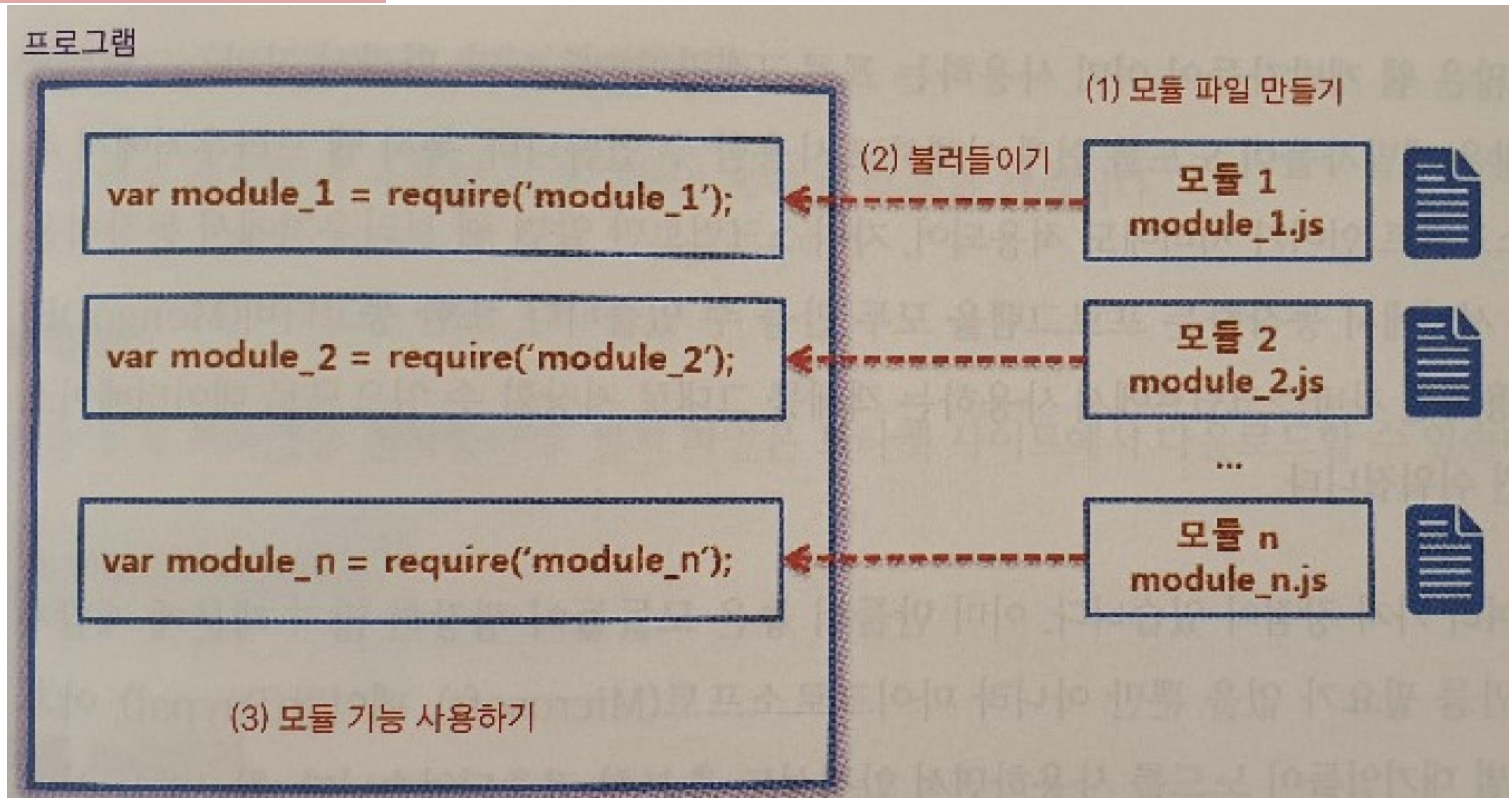
```
function longRunningTask() {
    console.log('end');
}
console.log('start');
longRunningTask();
console.log('next job');
```

```
function longRunningTask() {
    console.log('end');
}
console.log('start');
setTimeout(longRunningTask, 0);
console.log('next job');
```

Module

Module

- JavaScript file의 일부를 따로 떼어 별도의 파일로 만든 것
- package: 여러 모듈을 합친 것
- npm: node package manager
- 사용
  - 독립적 기능을 별로 파일로 분리 → 모듈
  - 여러 개의 파일로 나누어 개발
  - 모듈 사용: exports 전역 객체 이용
  - require() 메소드


프로그램
(1) 모듈 파일 만들기
var module_1 = require('module_1');
(2) 불러들이기
모듈 1
module_1.js
var module_2 = require('module_2');
모듈 2
module_2.js
...
var module_n = require('module_n');
모듈 n
module_n.js
(3) 모듈 기능 사용하기

ex1.js

```
let calc = require('./calc1');

console.log(calc.add(10, 20));
```

ex2.js

```
var calc = require('./calc2');

console.log(calc.add(10, 20));
```

calc1.js

```
exports.add = function(a, b) {
  return a + b;
}

exports.multiply = function(a, b) {
  return a * b;
}
```

calc2.js

```
let calc = { };
calc.add = function(a, b) {
  return a + b;
}
calc.multipy = function(a, b) {
  return a * b;
}
module.exports = calc;
```

Module

외장 모듈 사용하기

```
var nconf = require('nconf');

nconf.env();


console.log('OS env variable = %s', nconf.get('OS'));
```

Module

- 내장 모듈 사용하기
  - OS module
    - system information
    - hostname(), totalmem(), freemem(), cpus(), networlnterfaces(), ...
  - Path module
    - 파일 패스를 다룸
    - join(), dirname(), basename(), extname(), ...
  - 내장 모듈 정보: nodejs.org/api

```
var os = require('os');


console.log('hostname:', os.hostname());
console.log('memory: %d / %d', os.freemem(), os.totalmem());
console.log('CPU info:\n');
console.dir(os.cpus());
console.log('network interface:\n' );
console.dir(os.networkInterfaces());
```

```
let path = require('path');


let directories = ['users', 'mike', 'docs'];
let docsDirectory = directories.join(path.sep);
console.log('Document Directory: %s', docsDirectory);


let curPath = path.join('/Users/mike', 'nodepad.exe');
console.log('File path: ', curPath);
```

```
const filename = “C:\\Users\\mike\\notepad.exe”;

const dirname = path.dirname(filename);

const basename = path.basename(filename);

const extname = path.extname(filename);


console.log(“dir name = ”, dirname);

console.log(“base name = ”, basename);

console.log(“ext name = ”, extname);
```

HTTP 객체

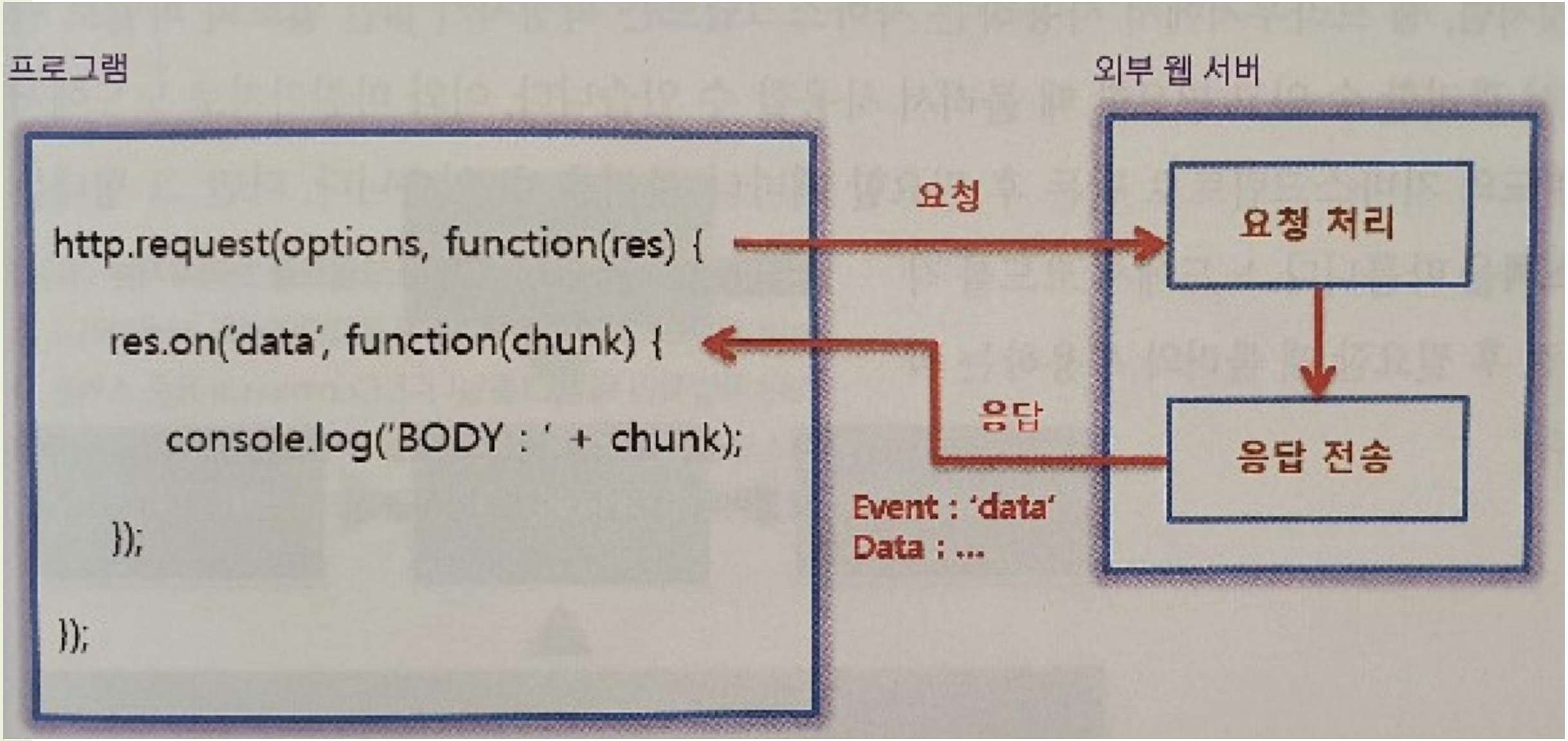
HTTP 객체

- http protocol로 웹 서버에 데이터 요청 기능: request()
- 응답을 받으면 callback function 자동 호출: res 객체
  - data event
  - on(): [event – callback function] binding

```
var http = require('http');


http.createServer(function (req, res) {
      res.writeHead(200, {'Content-Type': 'text/html'});
      res.end('Hello World!');
}).listen(8080);
```

browser - localhost:8080


프로그램
외부 웹 서버
http.request(options, function(res) {
res.on('data', function(chunk) {
console.log('BODY : ' + chunk);
});
});
요청
요청 처리
응답
응답 전송
Event : 'data'
Data : ...

전역 객체

전역 객체

- console: console 창에 결과를 보여주는 객체
- process: process 실행에 대한 정보를 다루는 객체
- exports: 모듈을 다루는 객체

console

- method
  - dir(object): 객체 속성 출력
  - time(id): 실행 시간 측정을 위해 시작 시간 기록
  - timeEnd(id): 실행 시간 측정을 위해 끝 시간 기록

```
console.log('숫자 보여주기: %d', 10);
console.log('문자열 보여주기: %s', "Hansung University");
console.log('JSON 객체 보여주기: %j', { name: 'Hansung' });
```

```
let result = 0;
console.time('duration');
for (let i = 1; i <= 1000; i++) {
    result += i;
}
console.timeEnd('duration');
console.log('1 ~ 1000: %d', result);
```

```
let school = {name: 'Hansung', district: 'Seongbuk-gu'};
console.dir(school);
```

process

- property/method
  - argv
  - env
  - exit();

```
console.log('argv # of parameters:', process.argv.length);
console.dir(process.argv);

process.argv.forEach(function(item, index) {
    console.log(index + ' : ', item);
});

console.dir(process.env);
```

Others

싱글 스레드

- 스레드 (thread): 작업을 처리하는 일손
- 예: 식당
  - 손님 여러 명, 점원 1명 → 싱글 스레드
  - 손님 여러 명, 손님 1명당 점원 1명 → 멀티 스레드
- Node는 싱글 스레드, 넌블로킹 모델

서버로서의 노드

- Single thread
  - 컴퓨터 자원을 적게 사용
  - CPU core를 하나만 사용
  - 하나뿐인 thread를 에러로 멈추지 않도록 잘 관리해야 함 → 서버 전체가 멈출 수 있음
- Web server가 내장되어 있음
- JavaScript 언어 사용 → 큰 장점

서버로서의 노드

- I/O가 많은 작업에 적합
  - Libuv 라이브러리를 사용하여 I/O 작업을 non-blocking 방식으로 처리
  - CPU 연산을 많이 요구하면 성능 저하
- Nginx 등보다 느림
- 용도
  - 개수는 많지만 크기는 작은 데이터를 실시간으로 주고 받을 때
  - 네트워크, 데이터베이스, 디스크 작업 등의 I/O에 특화
  - 실시간 채팅 app, 주식 차트, JSON 데이터를 제공하는 API 서버
- 안 좋은 경우: 이미지/비디오 처리, 대규모 데이터 처리

서버 외의 노드

- 웹, 모바일, 데스크톱 애플리케이션 개발에 사용됨
- 노드 기반 웹 프레임워크
  - Angular: 구글 진영에서 front end app 개발
  - React: 페이스북 진영
  - Vue, Meteor

공부한 것들

공부한 것들

- Background
- What
- 비동기 입출력
- 이벤트 기반
- Non-Blocking I/O
- Module
- HTTP 객체
- 전역 객체
- Others