Node.js를 활용한

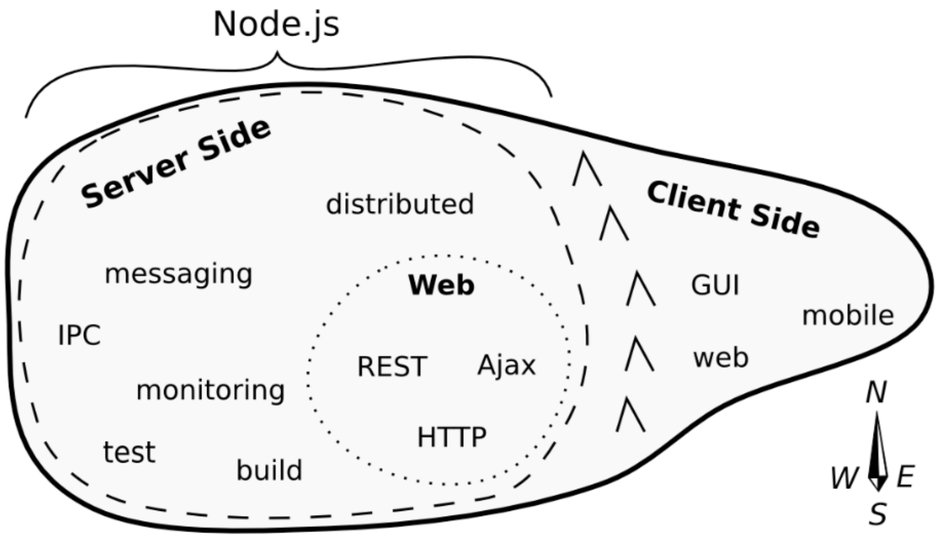
모바일에 특화된 확장 가능한 API 서버 만들기

자료 버전 히스토리

버전	작성자	비고
1.0	박창욱	1.0 배포
1.2	손영수	io / architecture 강화
1.3	신호철	TDD / BDD 강화

Part 1.
Node.js 가볍지 않게 소개하기!

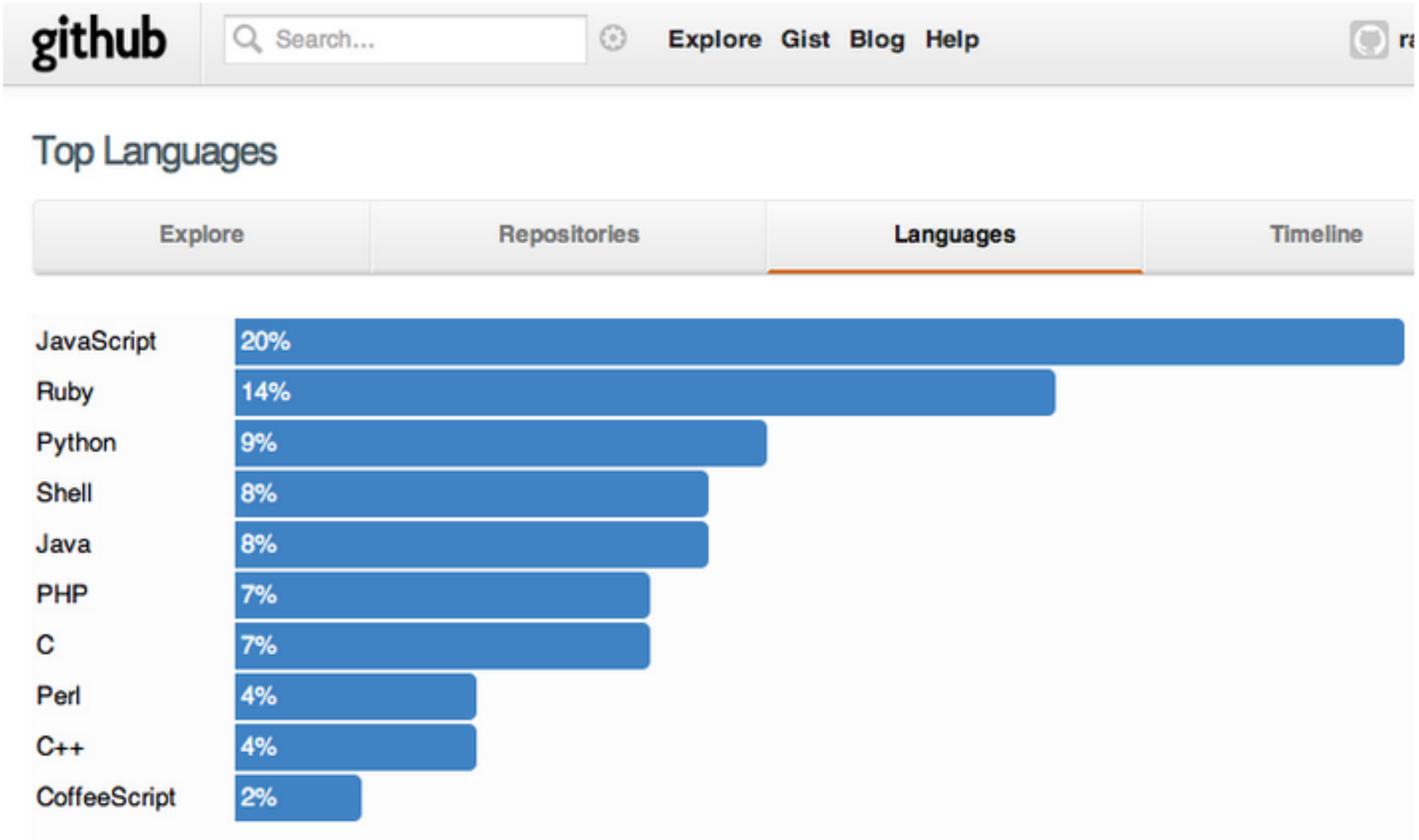
#01. Node.js 소개하기



- Server side Javascript
- Event driven
- Asynchronous
- Non-Blocking I/O
- Single Threaded
- Lightweight
- Fast

Node.js 장점

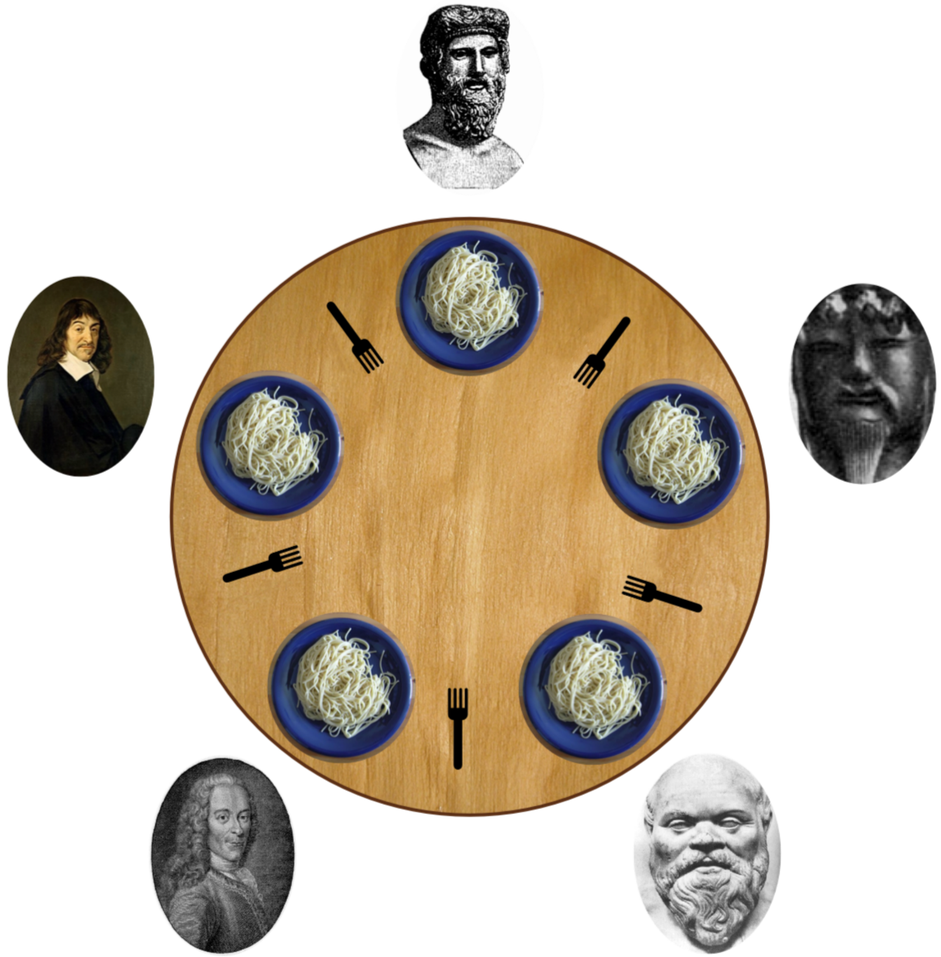
JavaScript는 이미 대세.. in github..



Node.js 장점

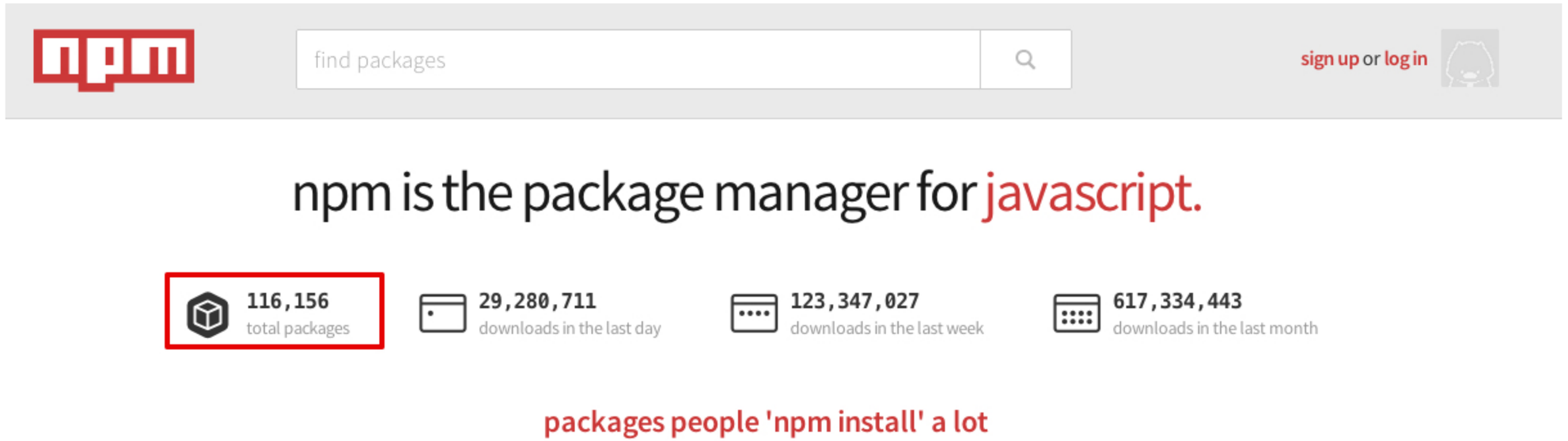
{Json}

Node.js 장점



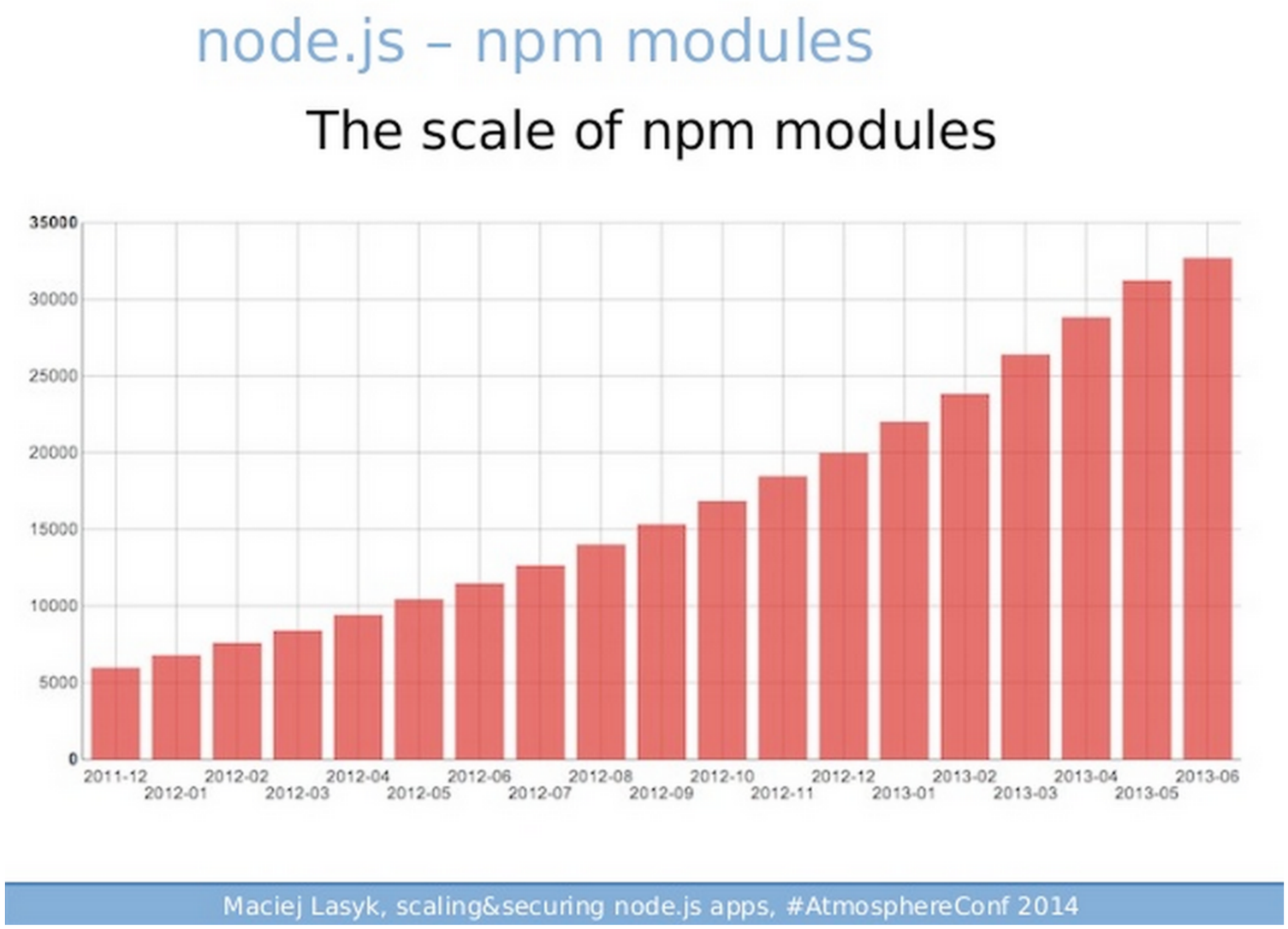
Single thread

Node.js 장점



Node Package Manager

Node.js 장점



Maciej Lasyk, scaling&securing node.js apps, #AtmosphereConf 2014

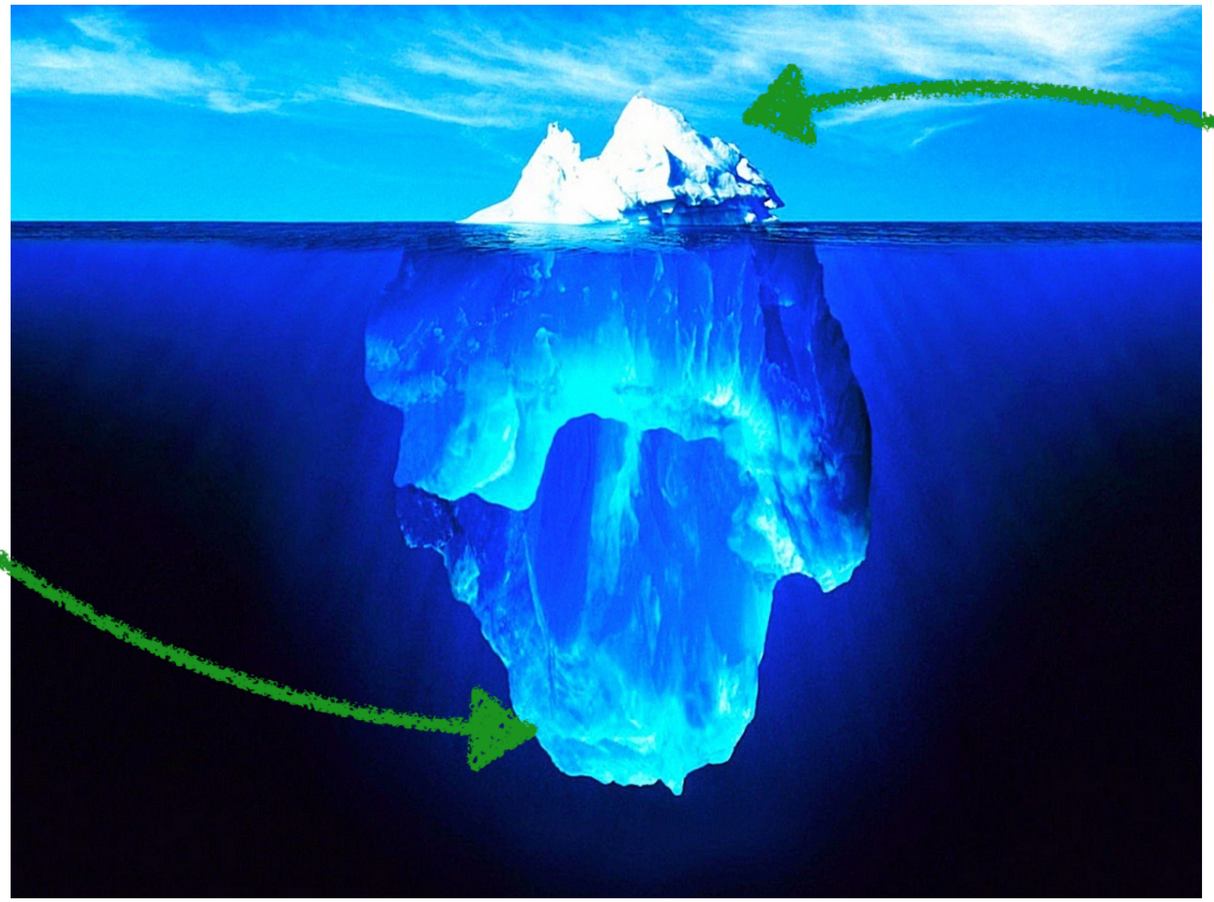
Node.js 단점

```
asncFunction1(function(err, result) {
    asncFunction2(function(err, result) {
        asncFunction3(function(err, result) {
            asncFunction4(function(err, result) {
                asncFunction5(function(err, result) {
                    // do something useful
                })
            })
        })
    })
})
```

모든 비동기가 가지는

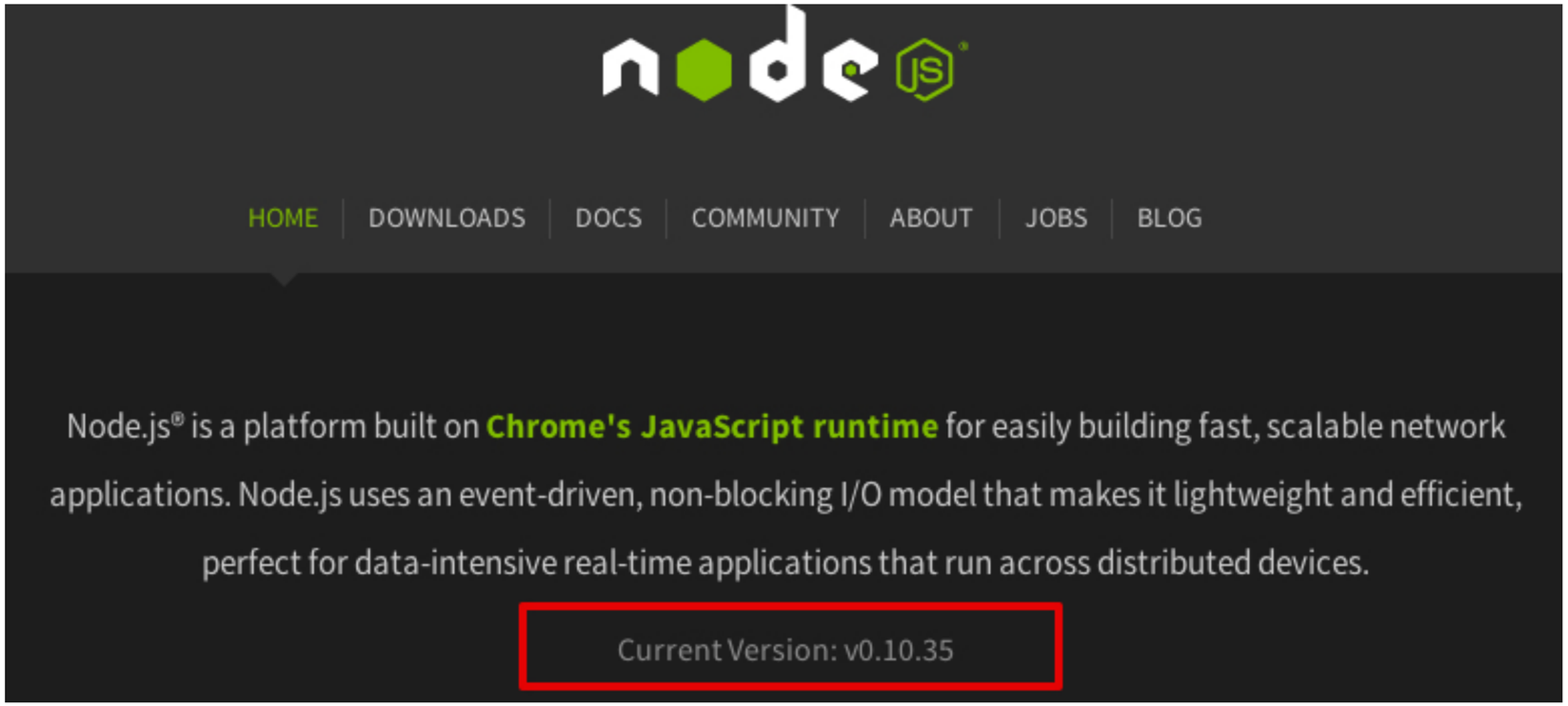
Callback hell!!

Node.js 단점



Javascript..

Node.js 단점



공식버전 v14 ..

올해 7월 정식 버전 v16

Node.js 활용

Network Program

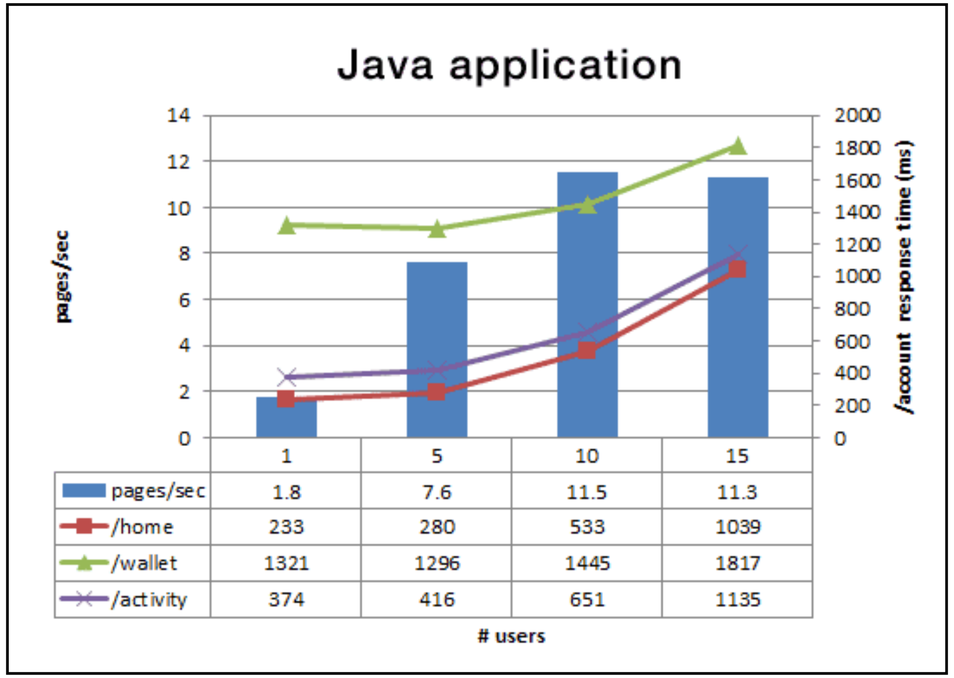
Tools

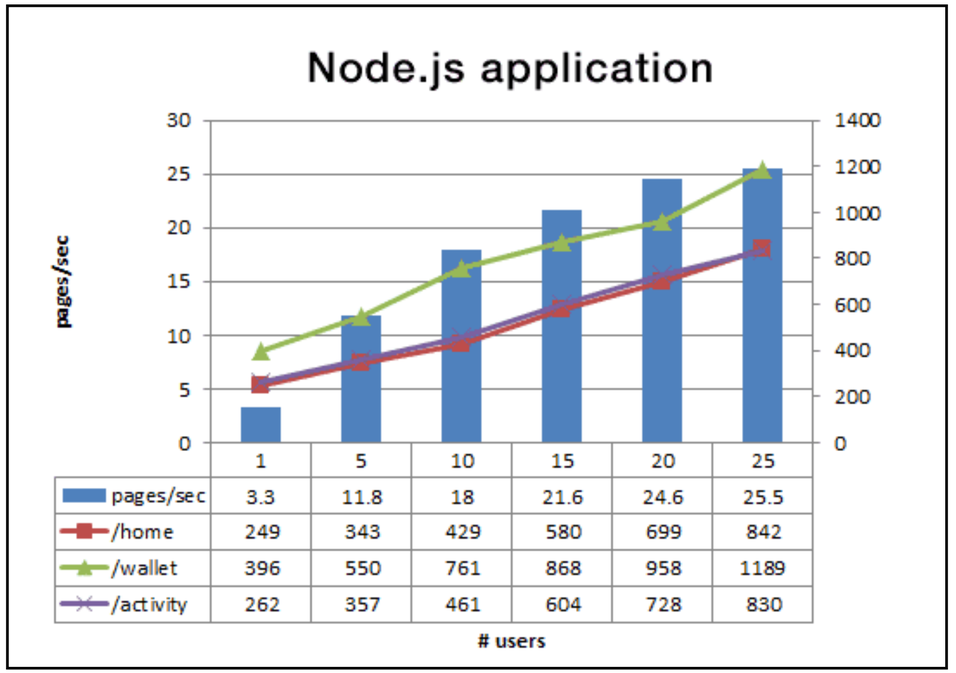
Robot

Desktop Application

Node.js 사용한 어플리케이션

Node.js Paypal



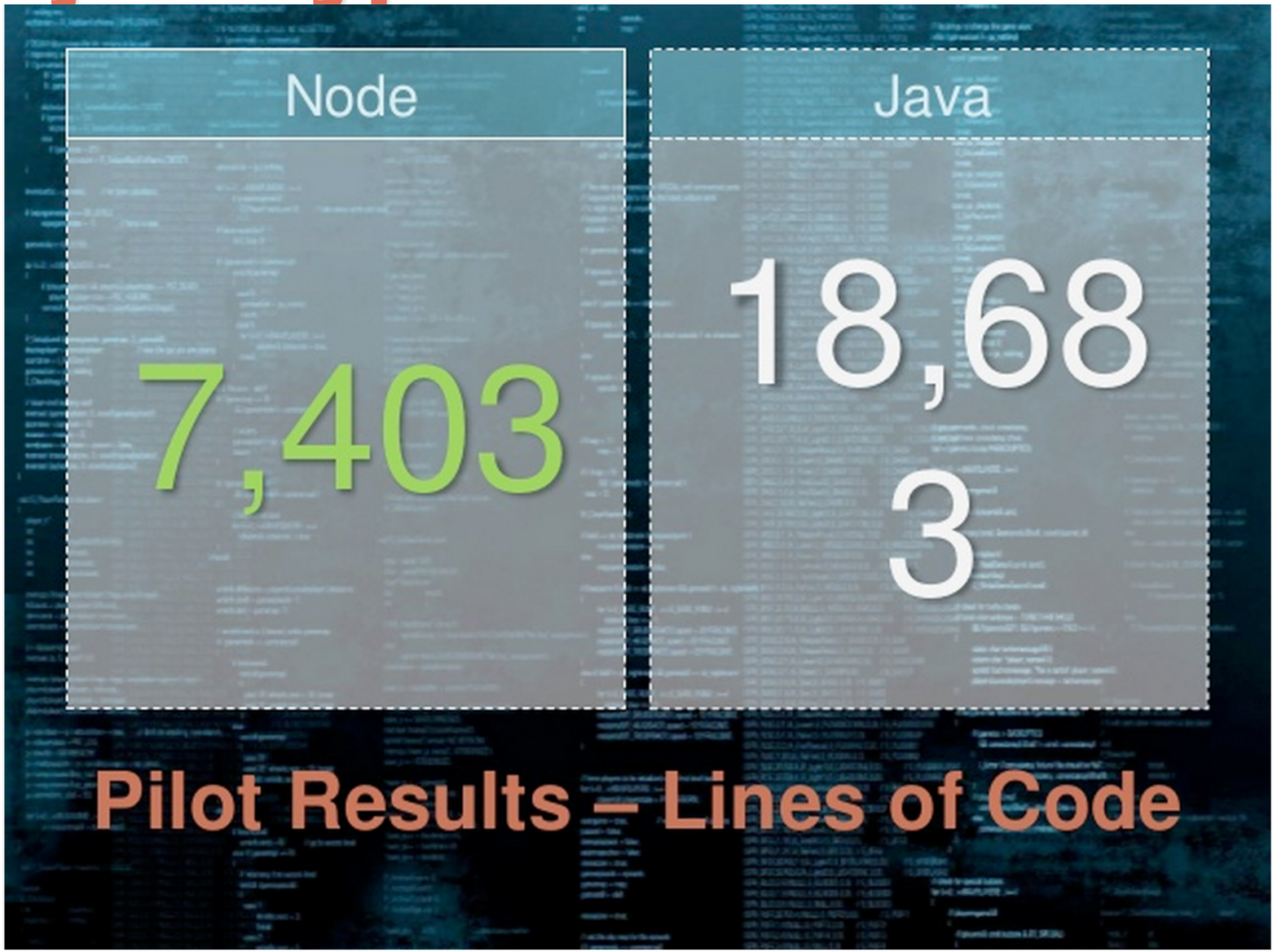


Double the requests per second vs. the Java application. This is even more interesting because our initial performance results were using **a single core for the node.js** application compared to **five cores in Java**. We expect to increase this divide further.

35% decrease in the average response time for the same page. This resulted in the pages being served **200ms faster**— something users will definitely notice.

[출처] https://www.paypal-engineering.com/2013/11/22/node-js-at-paypal/

Node.js Paypal

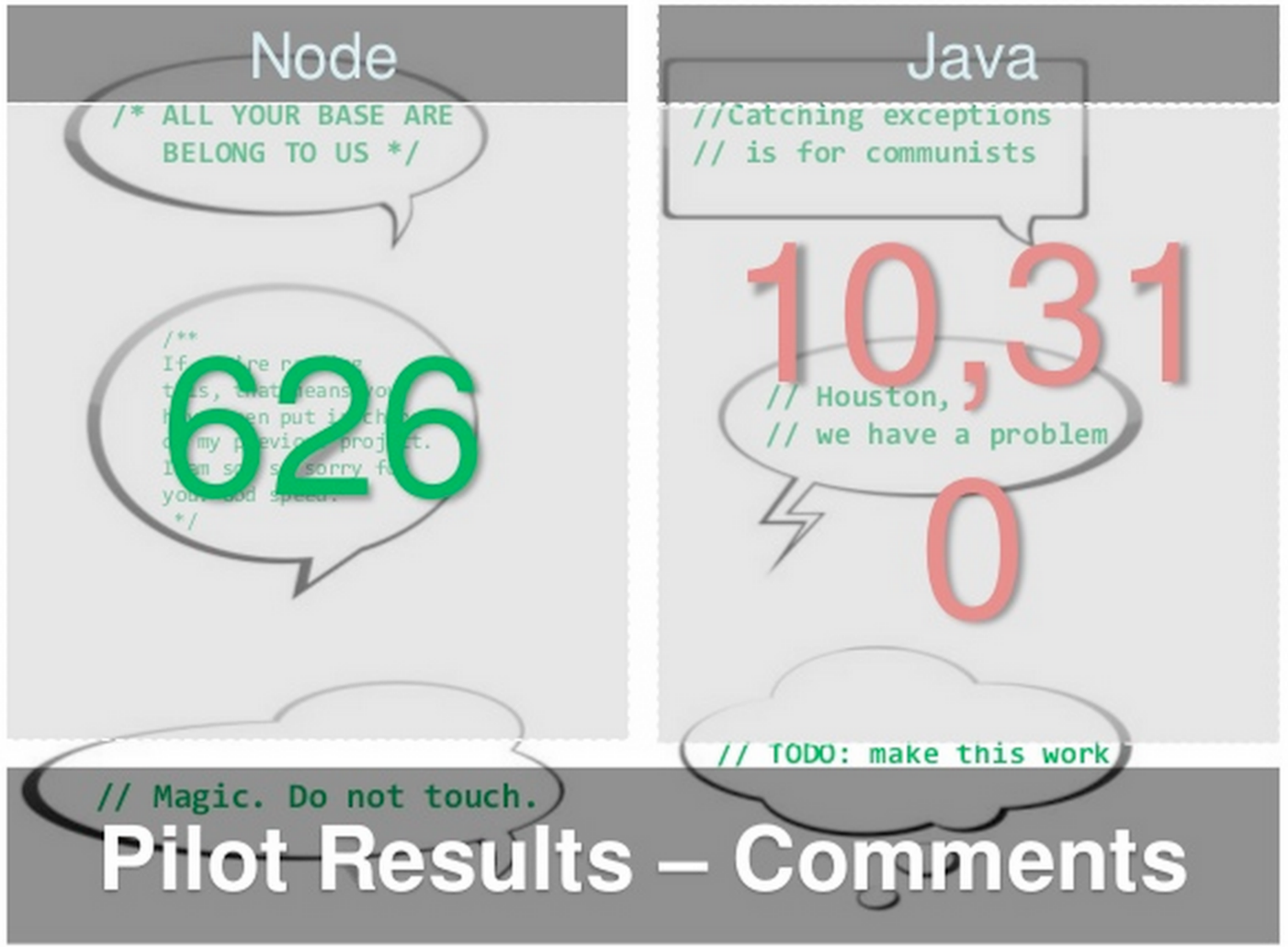


[출처]http://www.slideshare.net/lennymarkus/node-js-at-paypal

Node.js Paypal

[출처]http://www.slideshare.net/lennymarkus/node-js-at-paypal

Node.js Paypal

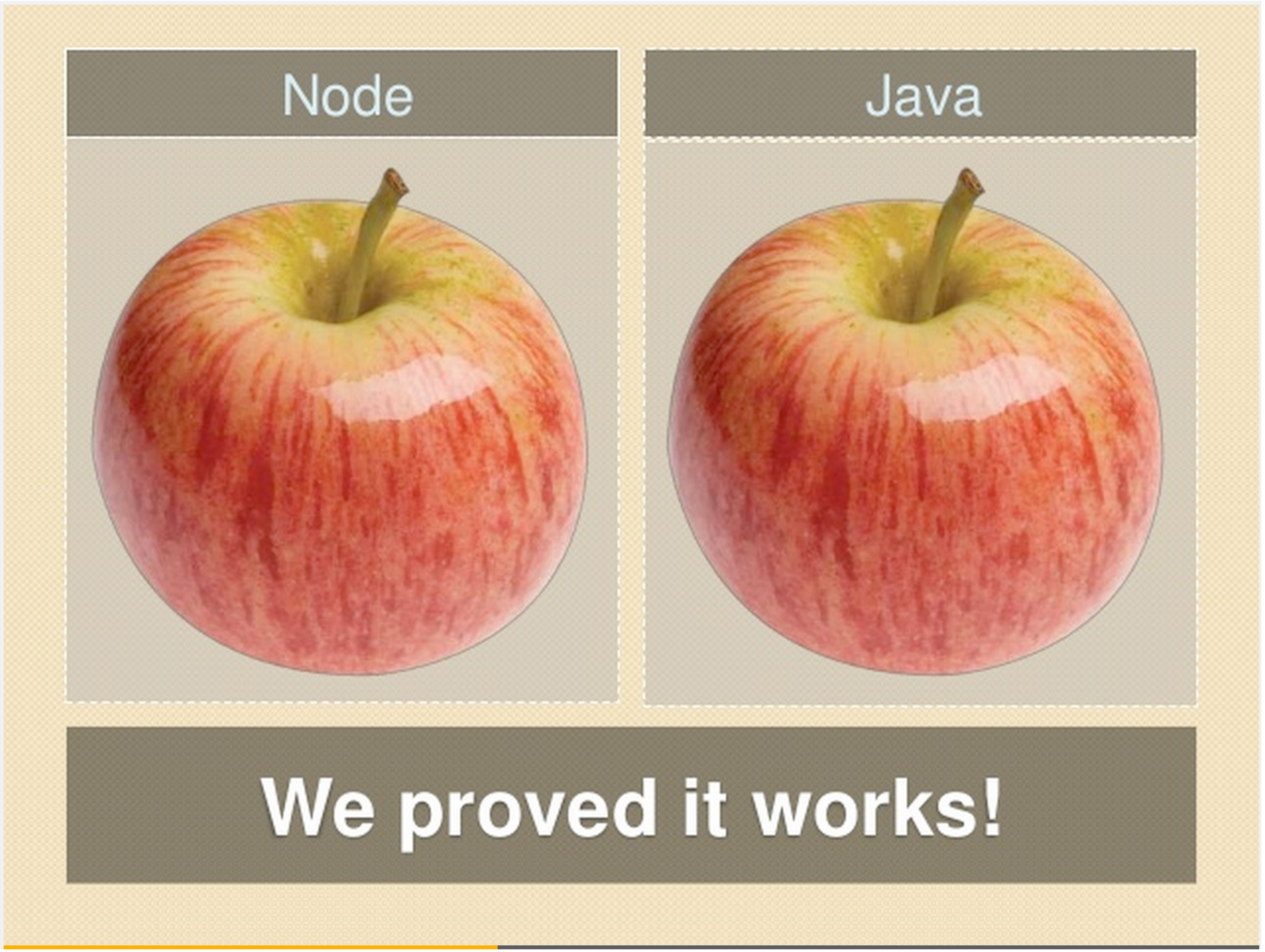


[출처]http://www.slideshare.net/lennymarkus/node-js-at-paypal

Node.js Paypal

[출처]http://www.slideshare.net/lennymarkus/node-js-at-paypal

Node.js Paypal



[출처]http://www.slideshare.net/lennymarkus/node-js-at-paypal

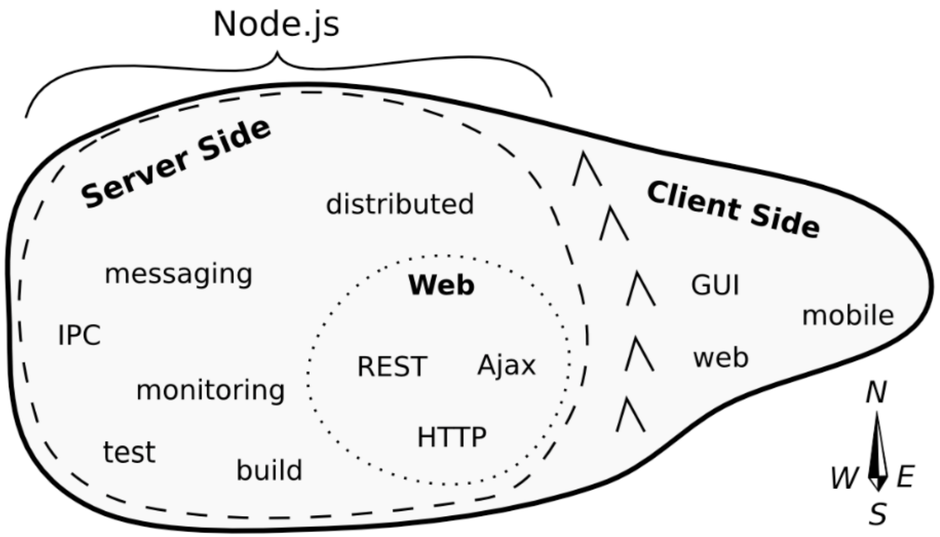
Node.js 좋은 활용

- REST + JSON APIs
- Single-page web app
- Real-time web app
- 빠른 prototyping
- 많은 I/O 처리

Node.js 나쁜 활용

- CPU 부하 걸리는 작업
- Multi-thread app
- 큰데이터 연산
- 복잡한 비지니스 로직

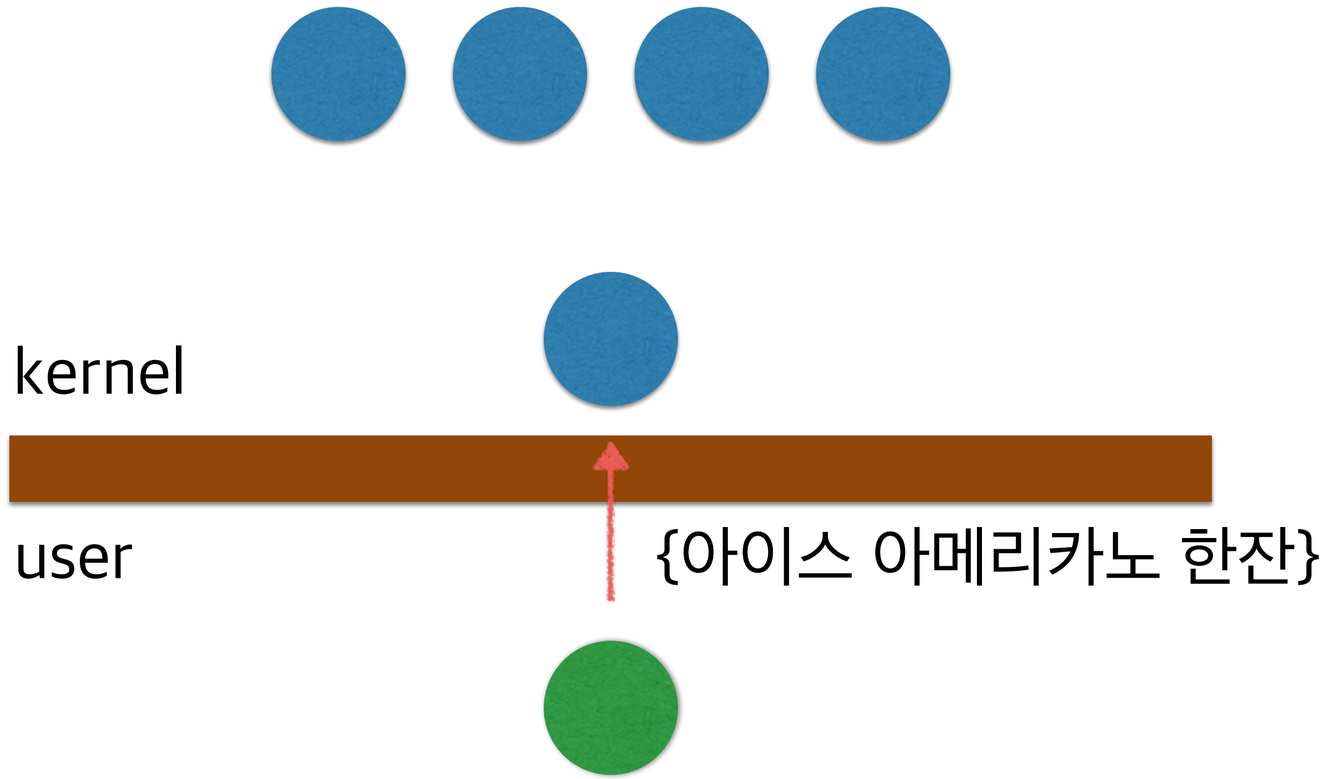
#02. Node.js의 철학..



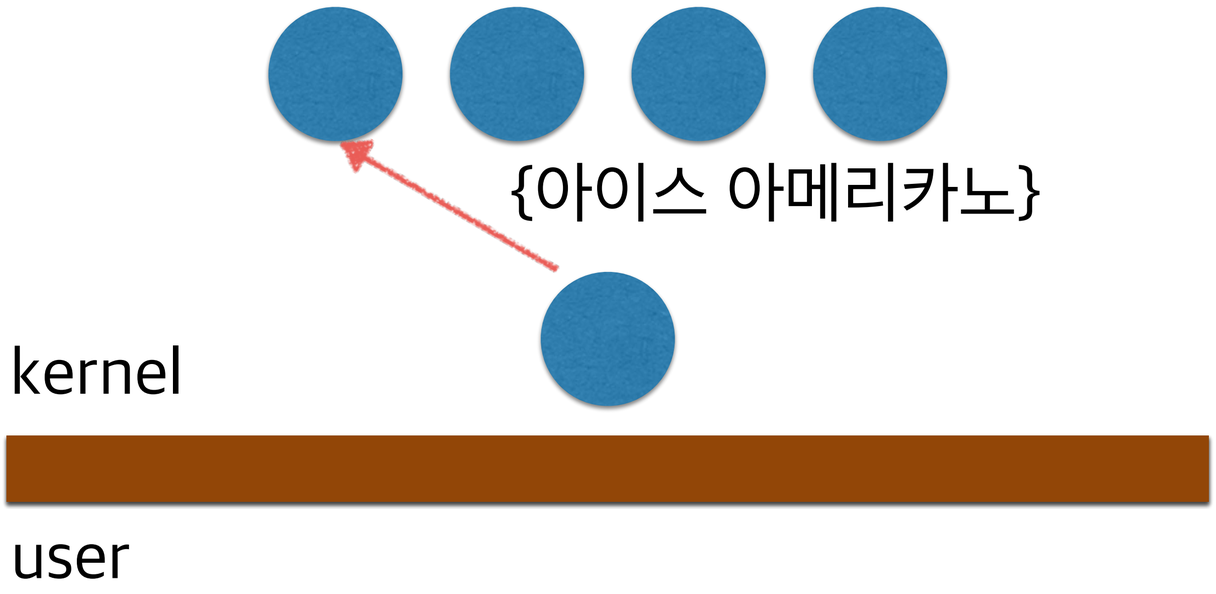
- Server side Javascript
- Event driven
- Asynchronous
- Non-Blocking I/O
- Single Threaded
- Lightweight
- Fast

Event driven

Event driven



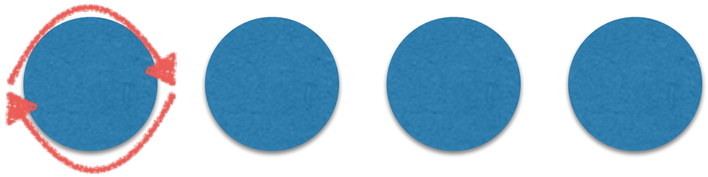
Event driven

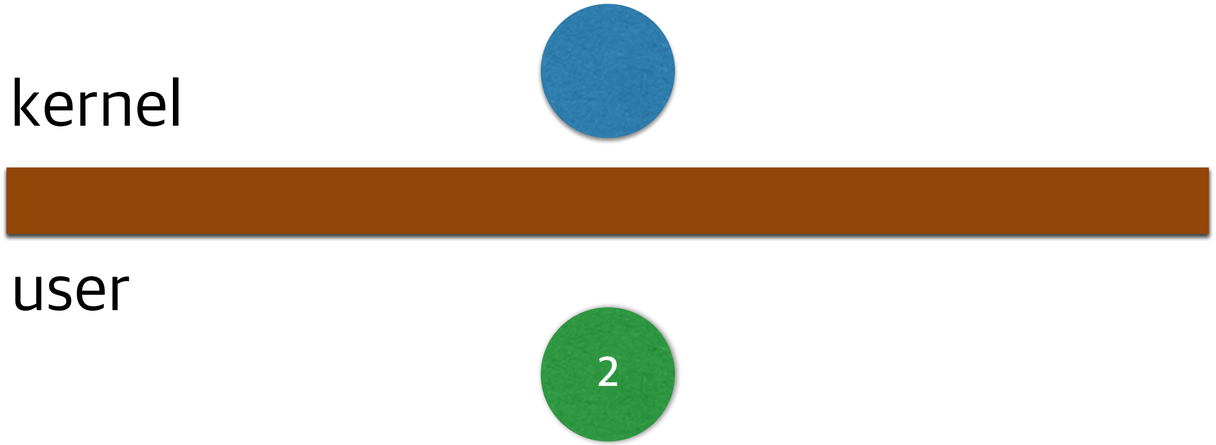


1

Event driven

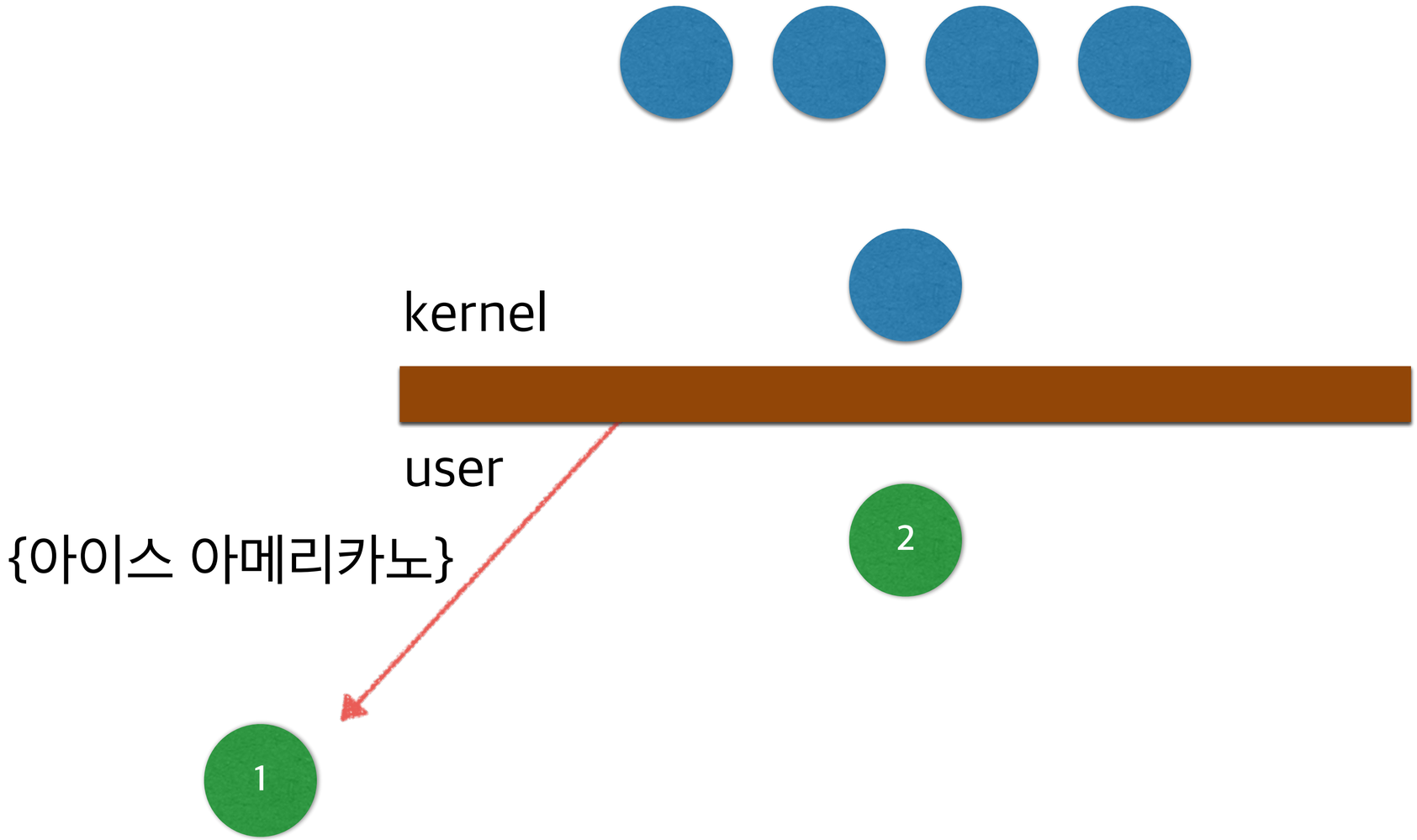
{아이스 아메리카노}





1

Event driven



Asynchronous

(sync_db.js)

```
var db = new Database({host: ‘localhost'});
var conn = db.connect();
var results = conn.find({name: 'pcw'});
console.log(results);
```

(async_db.js)

```
var db = new Database({host: ‘localhost'});
db.connect(function(err, conn) {
    conn.find({name : 'pcw'}, function(err, results) {
        console.log(results);
    });
});
```

Asynchronous

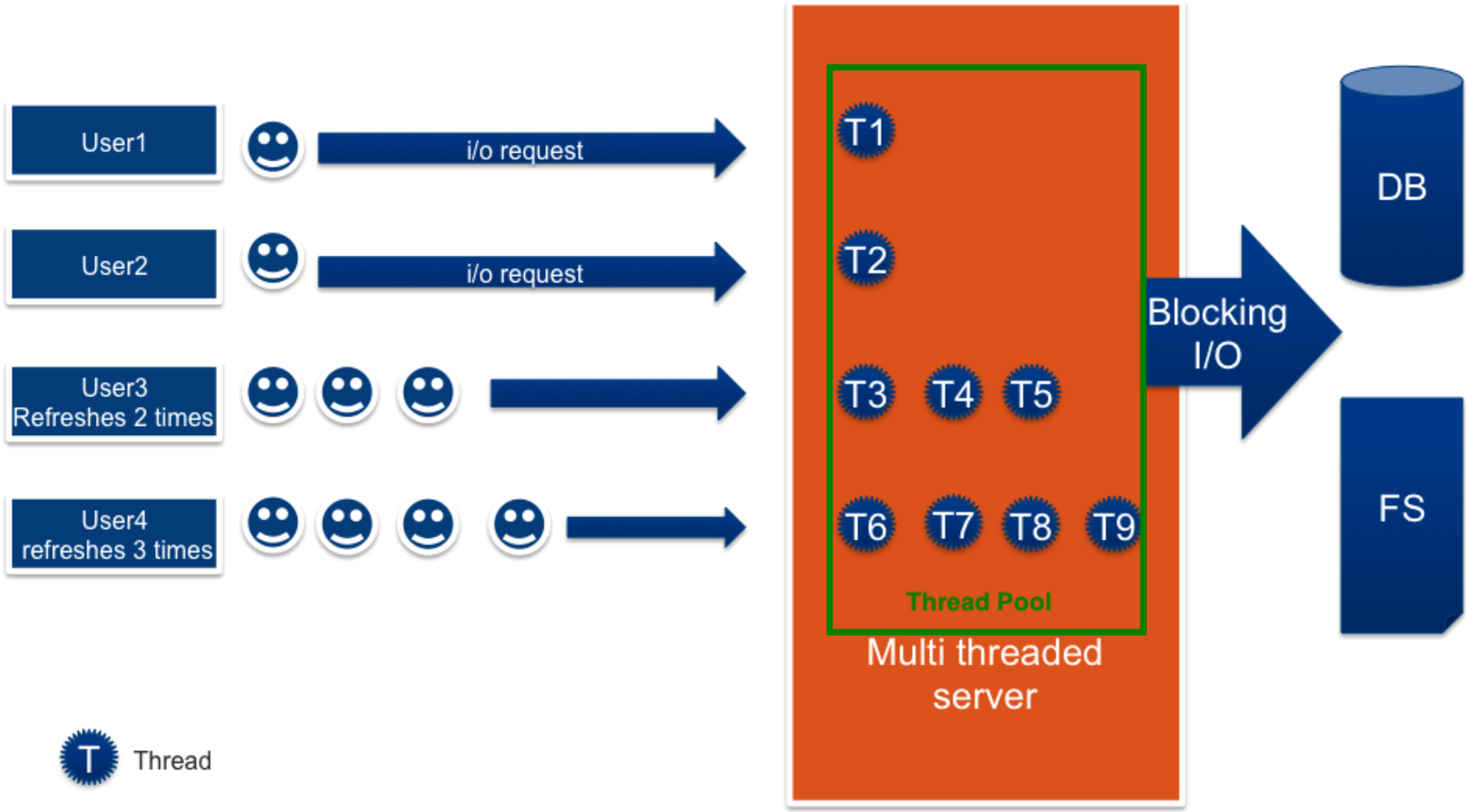
(sync_db.js)

```
var db = new Database({host: ‘localhost'});
var conn = db.connect();
var results = conn.find({name: 'pcw'});
console.log(results);
```

(async_db.js)

```
var db = new Database({host: ‘localhost'});
db.connect(function(err, conn) {
    conn.find({name : 'pcw'}, function(err, results) {
        console.log(results);
    });
});
```

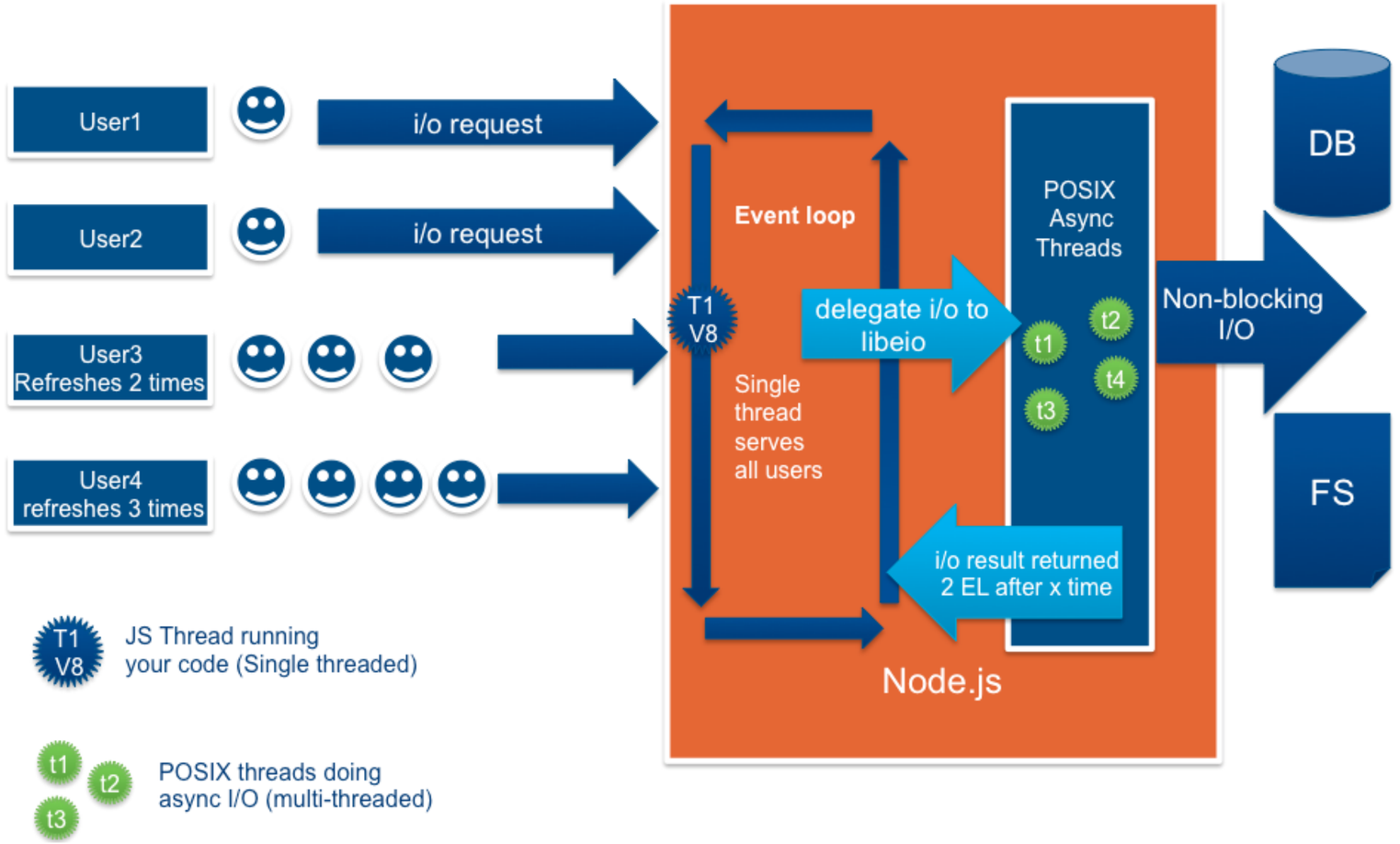
Blocking IO + Multi-Threaded Server



Blocking I/O + direct interface to clients leads to thread-pools w/ larger number threads and more memory requirements

[출처] : http://blog.pivotal.io/cloud-foundry-pivotal/products/future-proofing-your-apps-cloud-foundry-and-node-js

Async-IO + Event 기반의 Server



[출처] : http://blog.pivotal.io/cloud-foundry-pivotal/products/future-proofing-your-apps-cloud-foundry-and-node-js

#03. Node는 왜 이런 구조를 가졌는가..

I/O 비용은 비싸다!

개발자가 알아야할 숫자들..

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from disk	20,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

[출처]Jeff Dean presentation on building large-scale systems at Google http://yoavs.blogspot.kr/2009/10/jeff-dean-presentation-on-building.html

IO를 다루는 방법

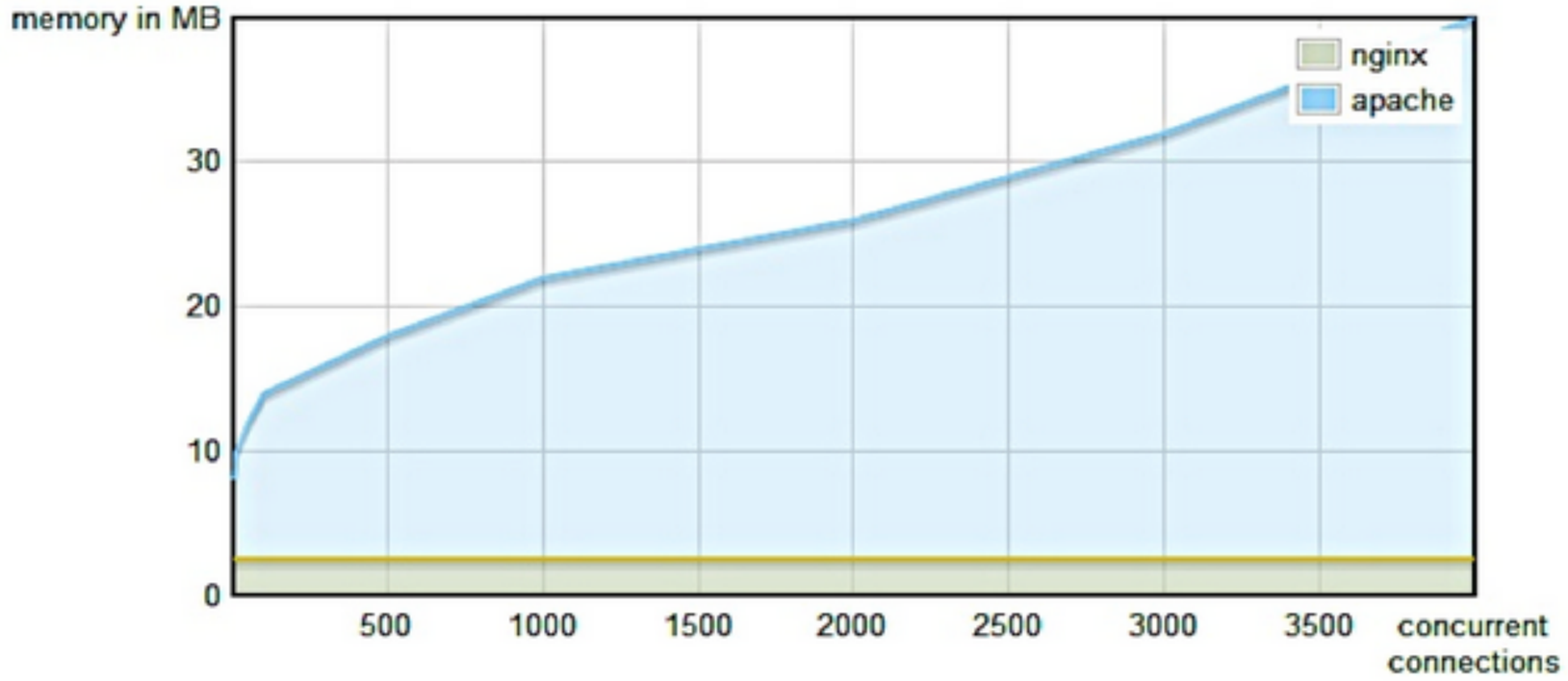
- 동기 : 한번에 하나의 요청만
- Fork : 요청당 새 Process 생성
- Thread : 요청 당 새 Thread 생성

[출처] http://www.nightmare.com/medusa/async_sockets.html

Node는 왜 이런 구조를 가졌는가..

메모리를 적게 사용하자!

Memory Usage : Nginx vs Apache

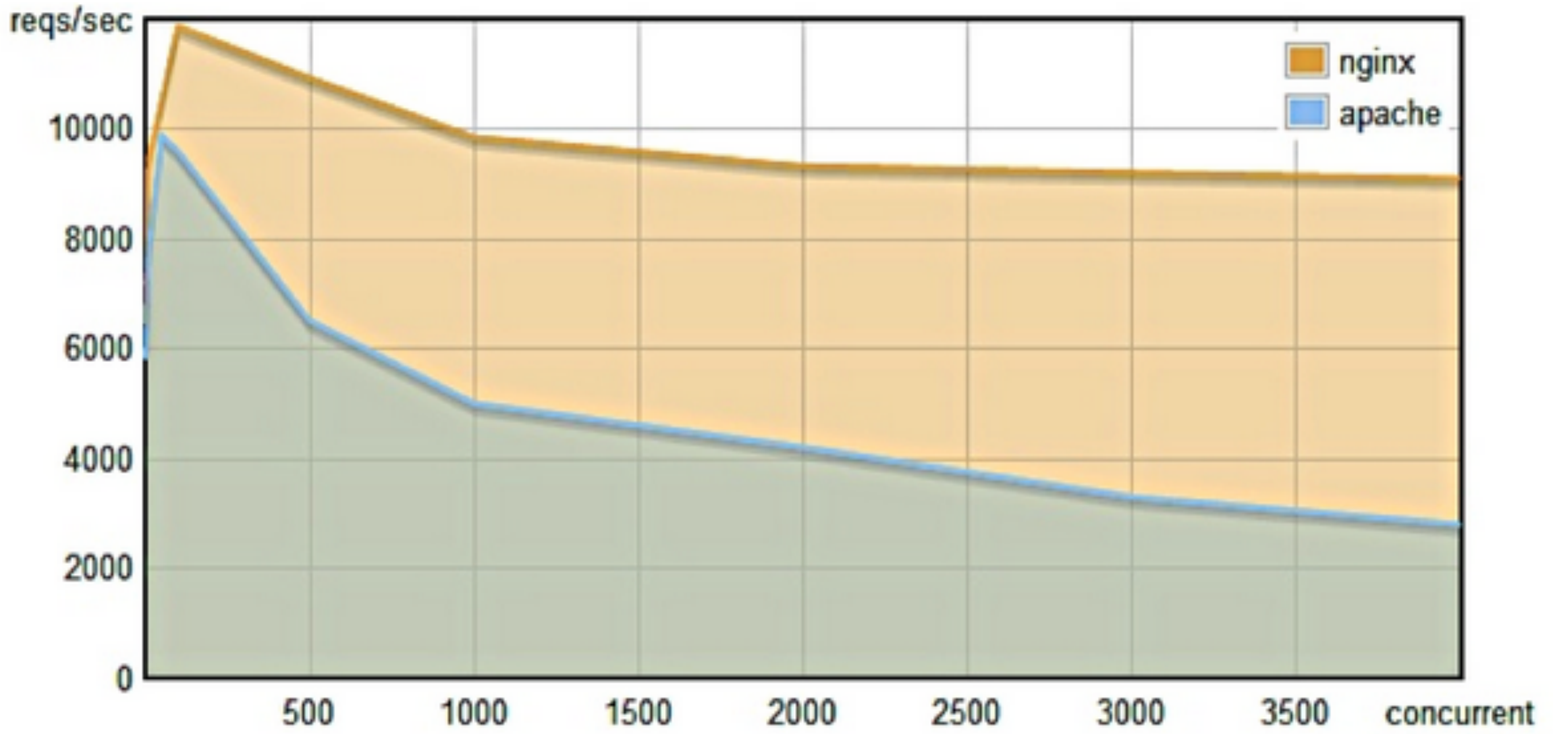


Memory v/s concurrent connections

At ~4000 concurrent connections,
- Nginx uses 3MB memory
- Apache uses 40MB memory

Ref: http://blog.webfaction.com/a-little-holiday-present

Performance : Nginx vs Apache



Reqs/sec v/s concurrent connections

At ~4000 concurrent connections,
- Nginx can serve ~9000 reqs/sec
- Apache can serve ~3000 reqs/sec

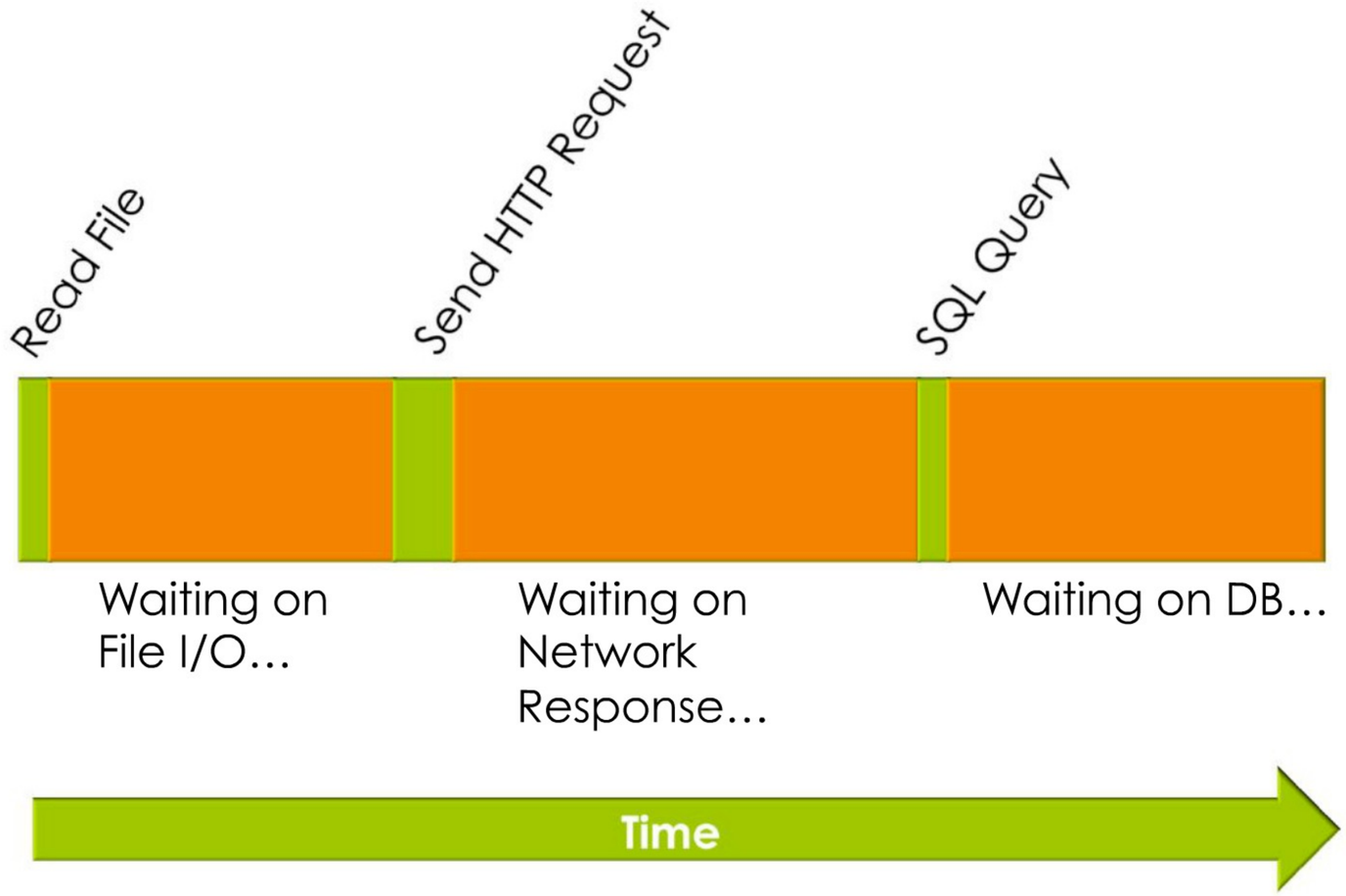
Ref: http://blog.webfaction.com/a-little-holiday-present

#04. 전통적인 Thread vs Eventloop

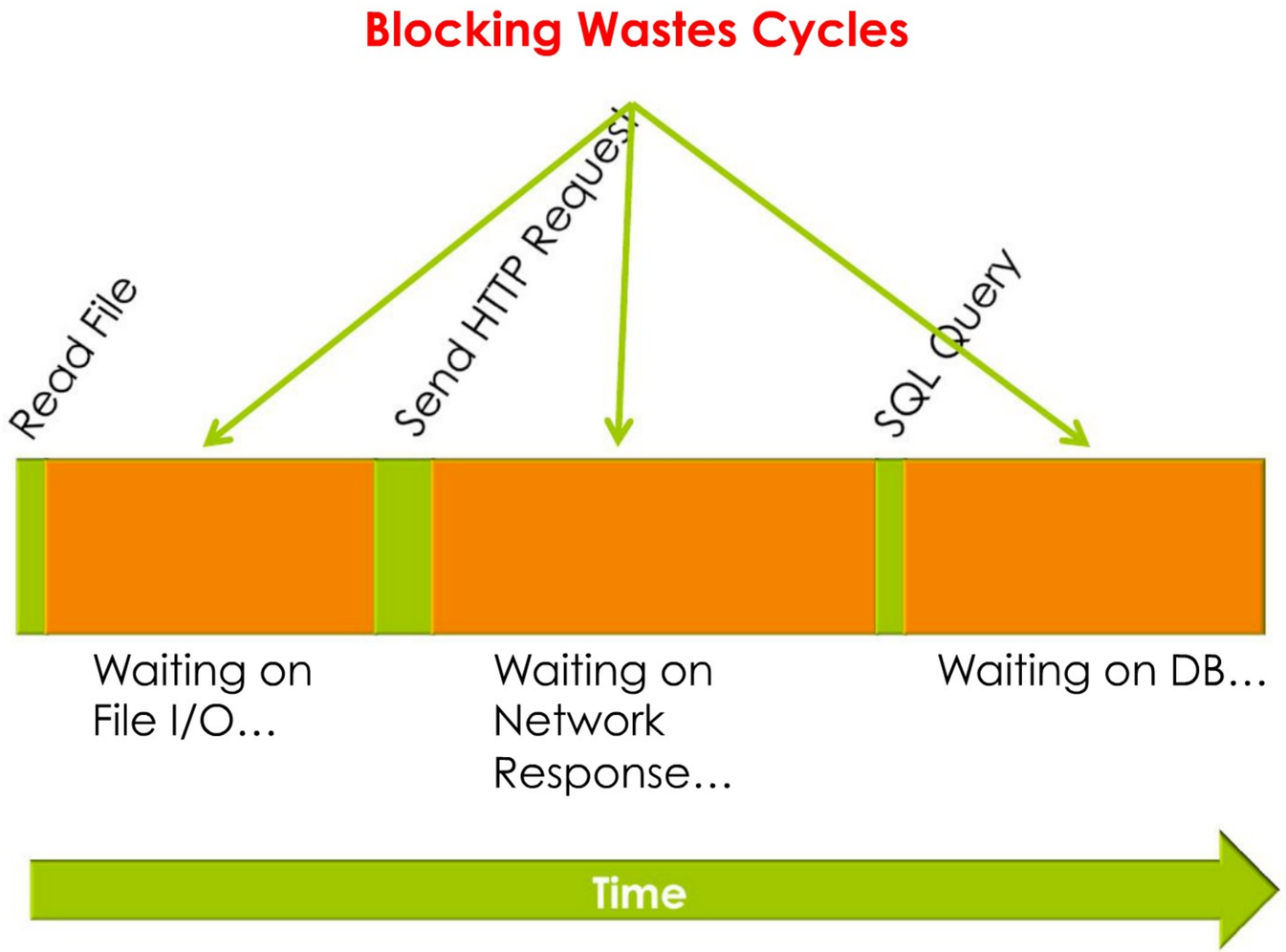
전통적인 Multi Threaded Model

- N 개의 Worker 쓰레드 / 프로세스
- 각각의 Connection 당 Worker가 대응하는 모델
- 모든 Worker가 사용 중이면 다른 Connection 처리 못함
- 예) File IO , DB IO, Network IO 등..

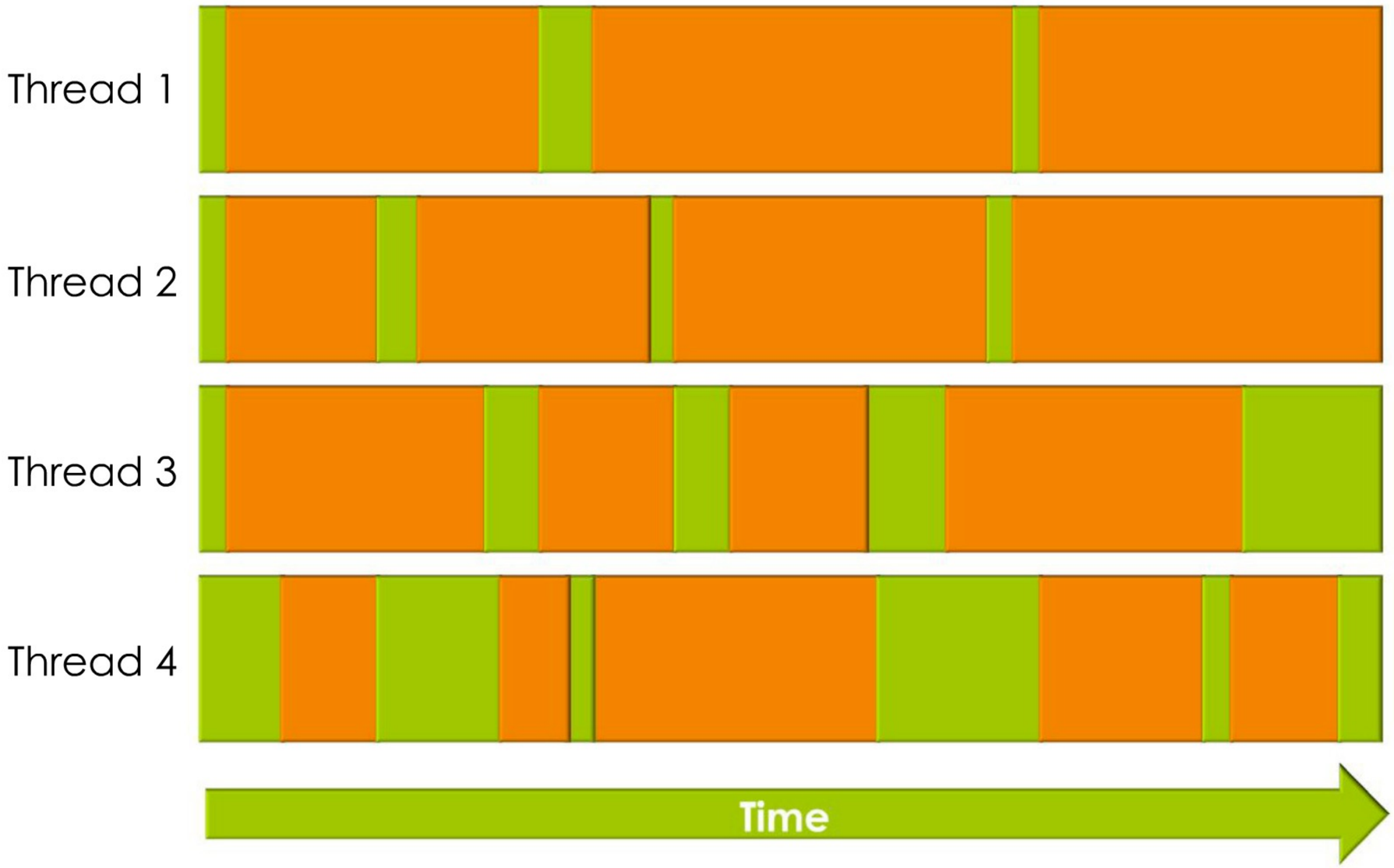
한 Worker의 Lifecycle



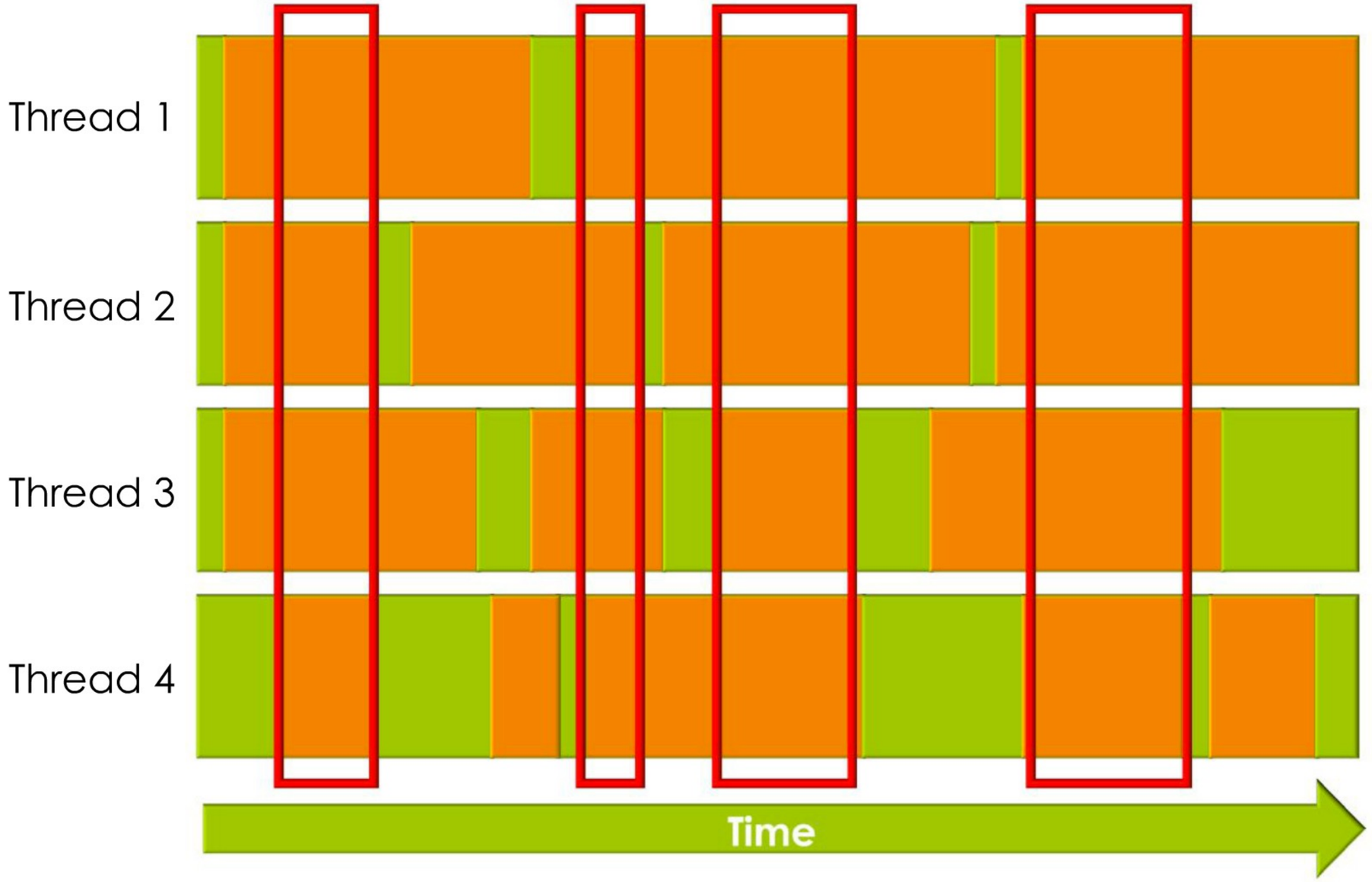
한 Worker의 Lifecycle



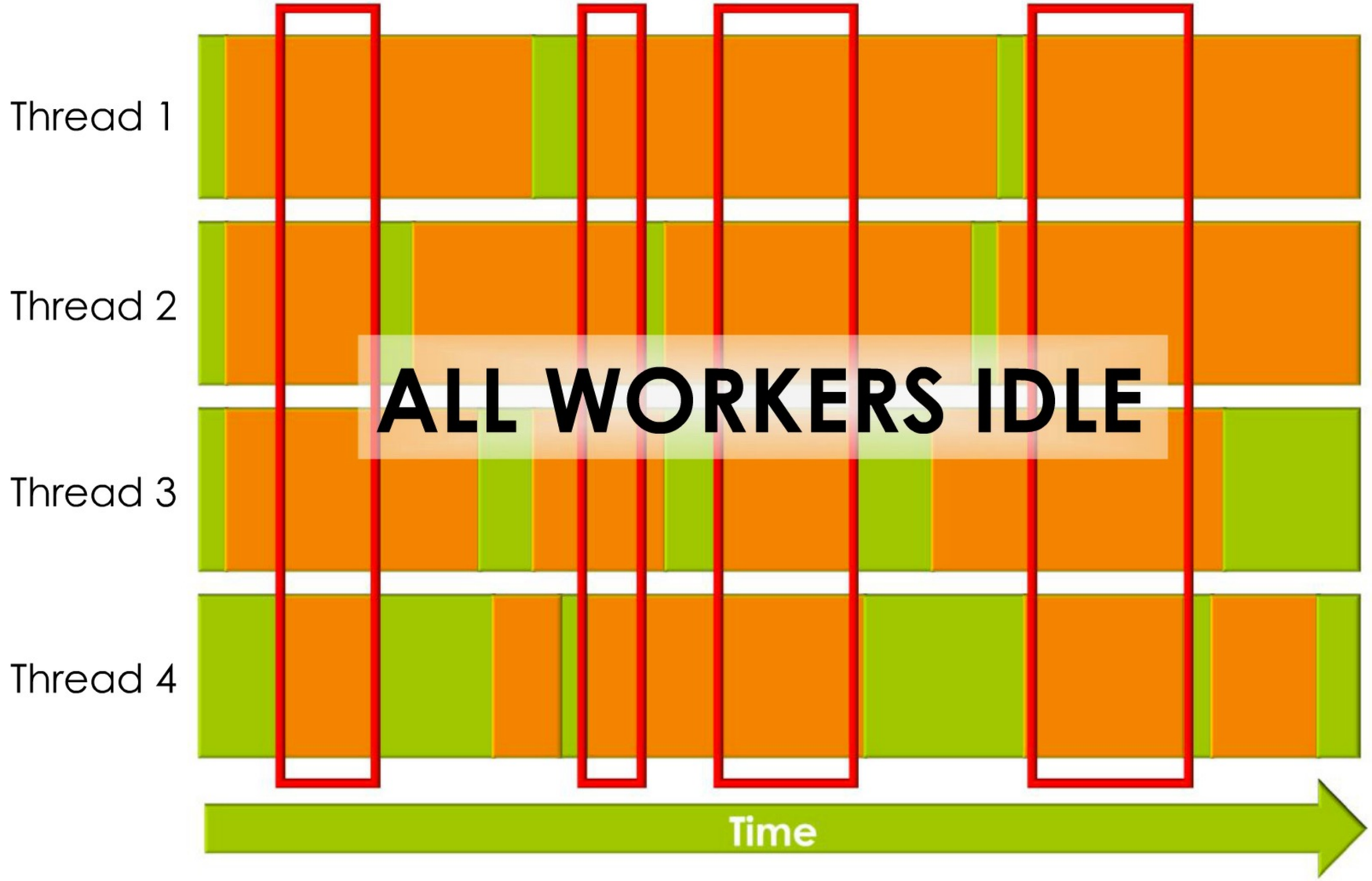
여러개의 Worker라면.



여러개의 Worker라면.



여러개의 Worker라면.



멀티 쓰레드 기반의 코드는

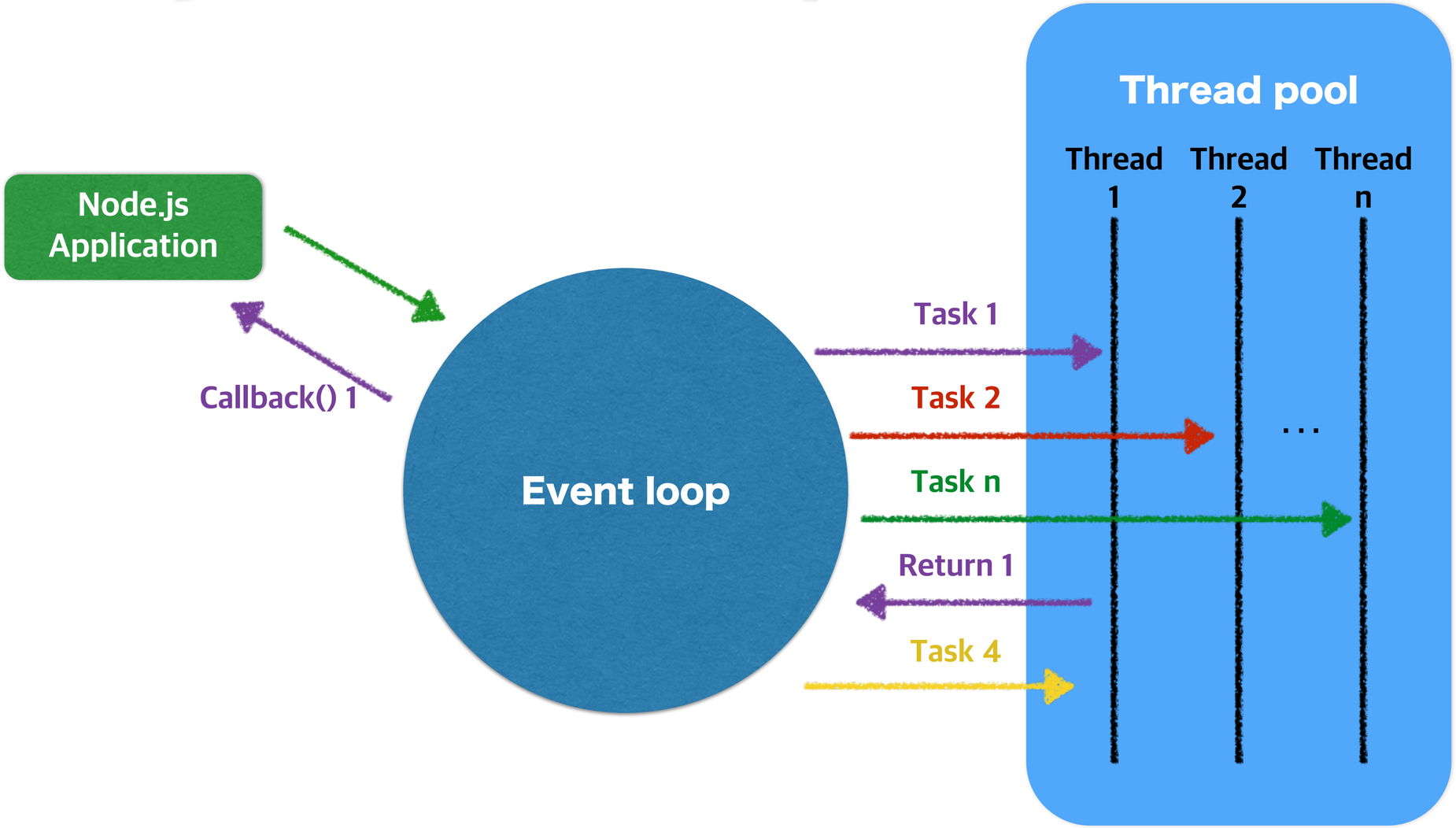
- 효율적으로 작성하기 어렵다..
- 디버그 하기도 어렵다.
- dev / test / maintenance 주기가 나쁜 영향을 줄 수 있다.
- 종종 비효율적인 성능을 야기시킨다.

- 그래서 Node.JS는 Single Thread.

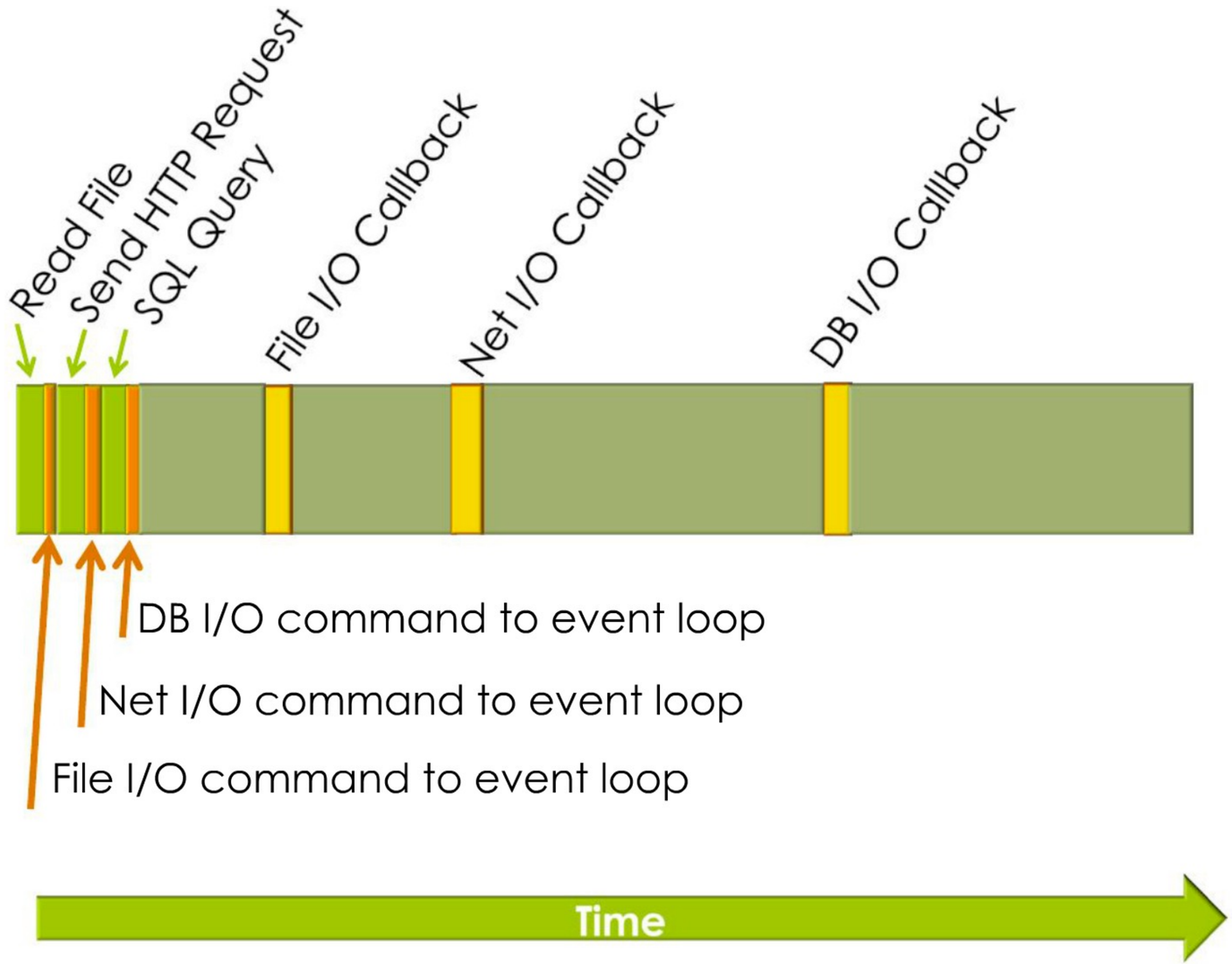
Node.js Style : Event Loop

- IO를 우리가 처리하지 말고, Node Event Loop에 맡겨라.
- 그리하면.. 내부적으로 프로세스 최적화, 동기화 등을 처리 할게!!

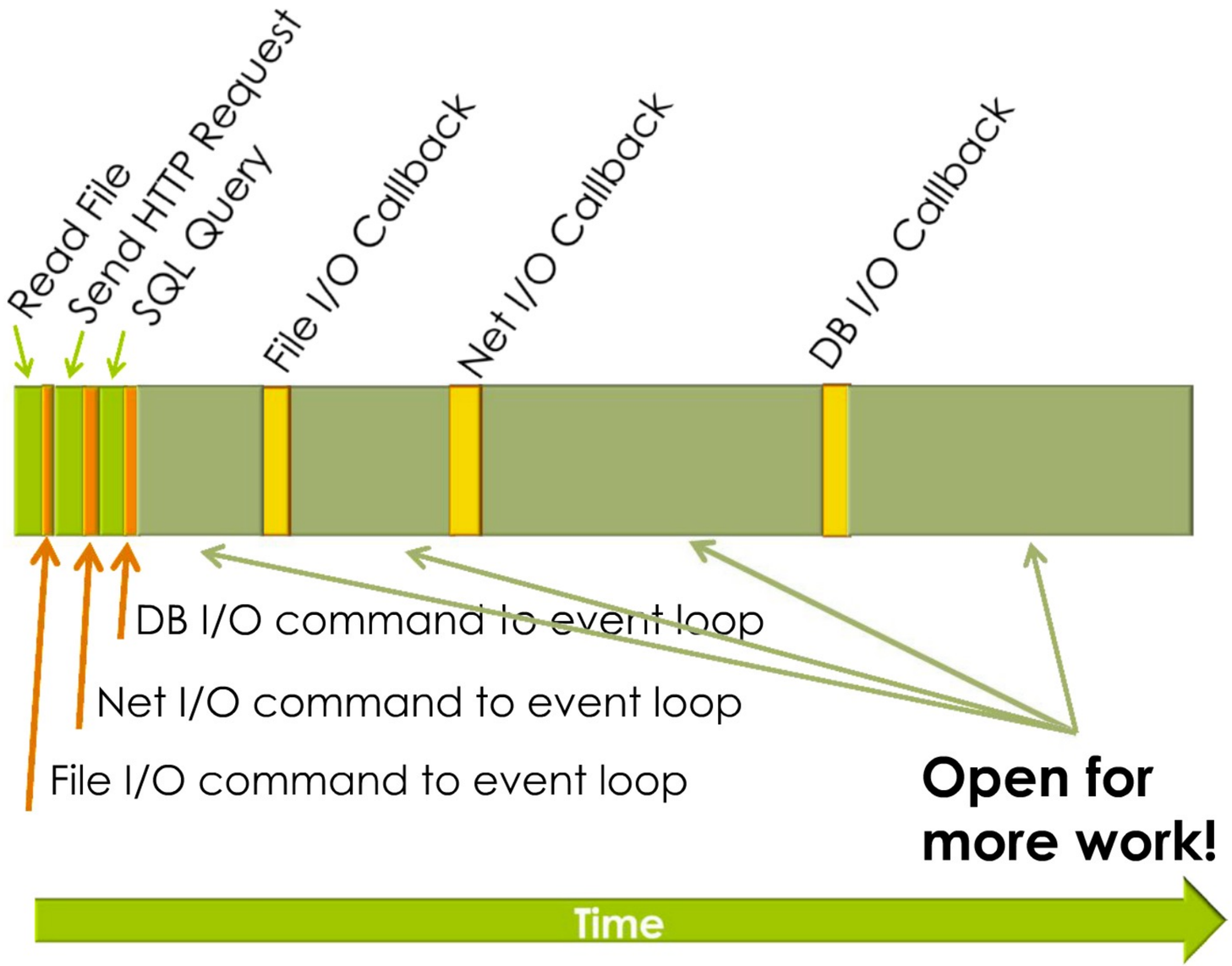
Node.js Event Loop



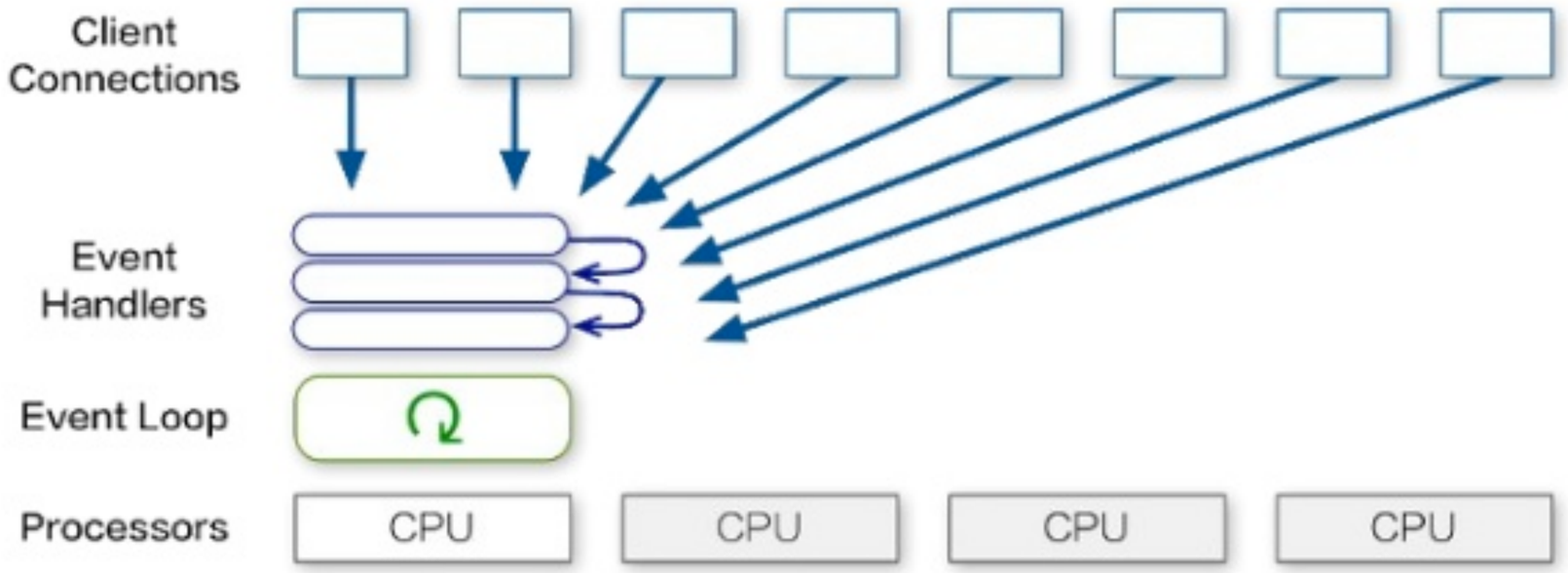
Node.js Event Loop



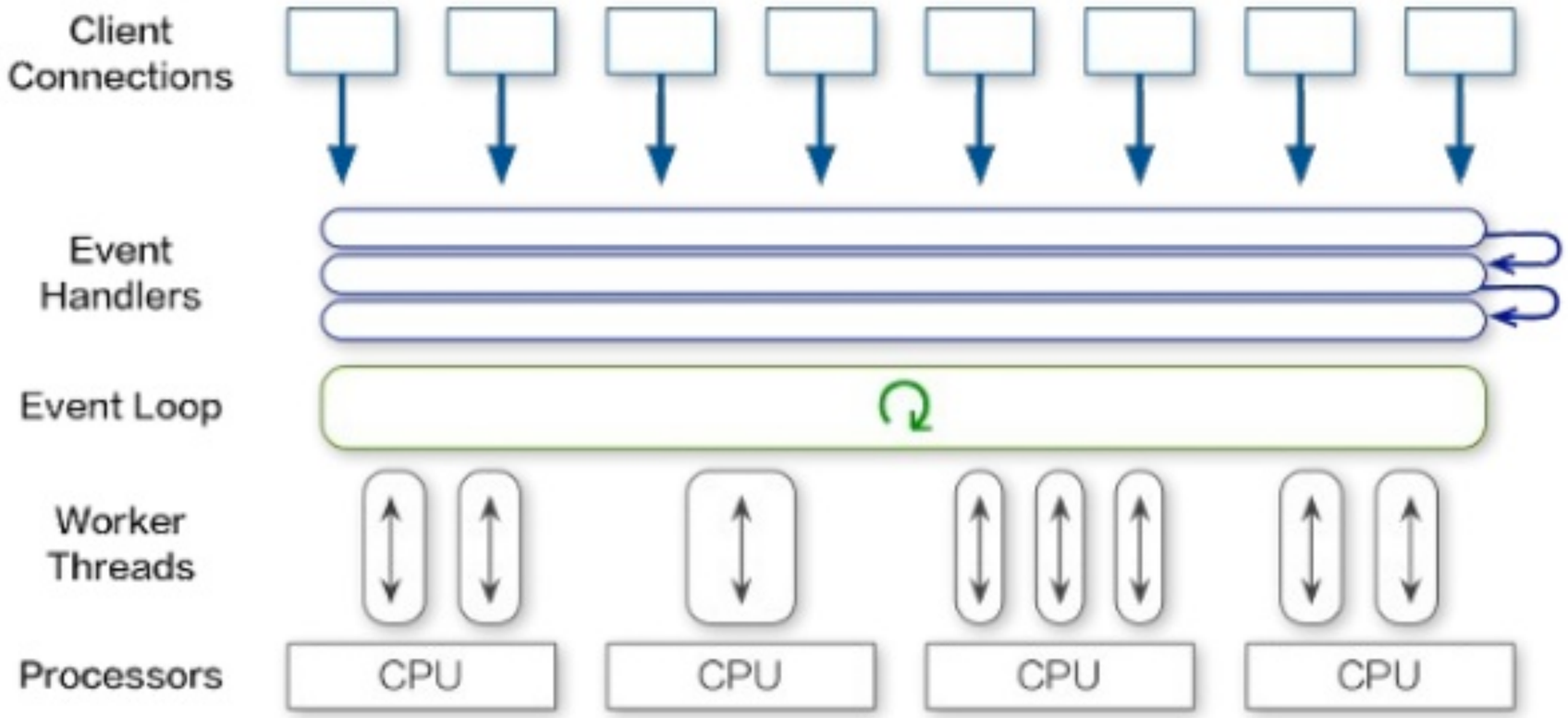
Node.js Event Loop는 더 많은 일을 할수 있다.



Single Thread가 성능이나요?
CPU가 놀지 않나요?

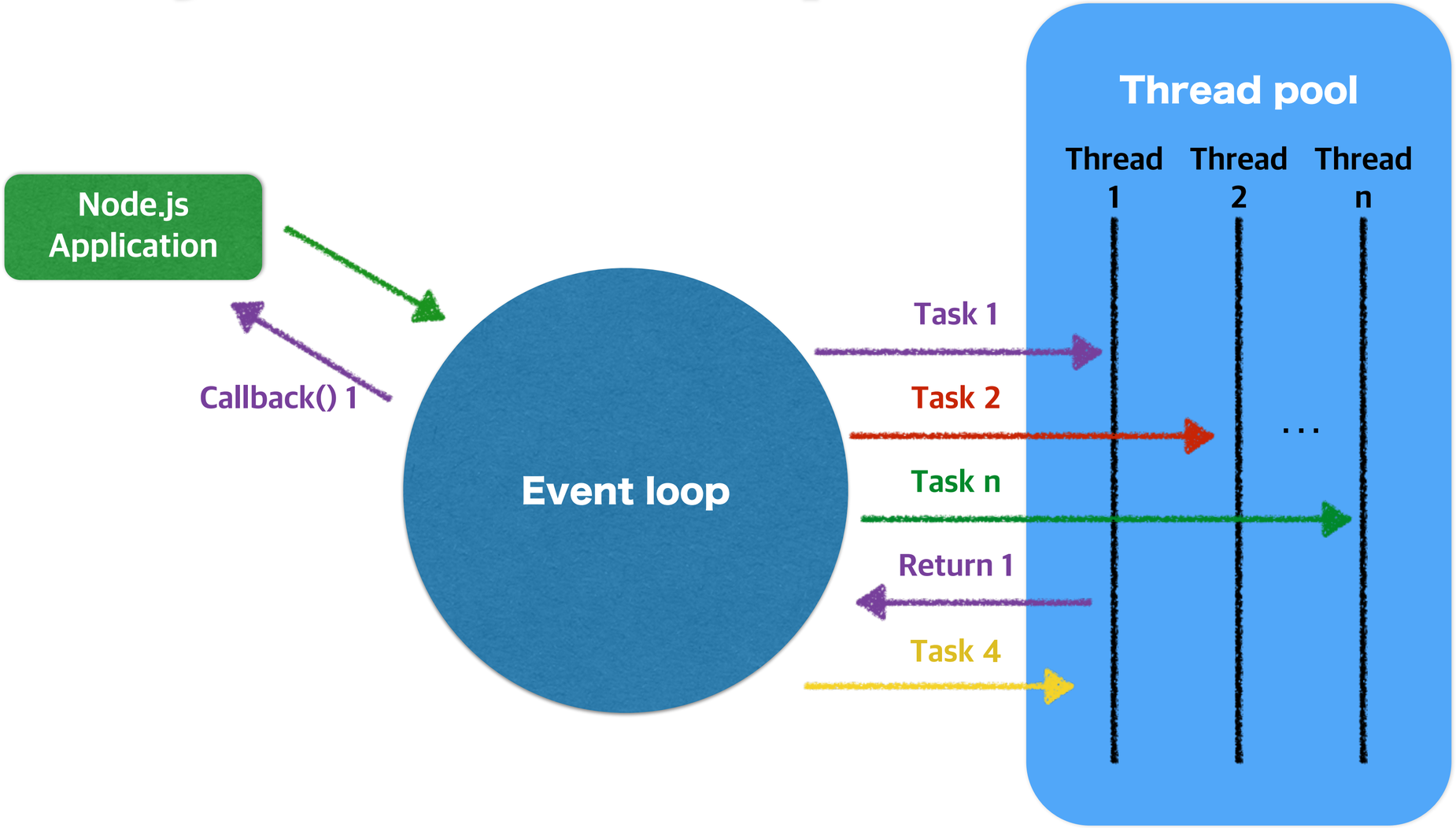


Node.js Production 레벨에서는
CPU 갯수 +@ 만큼 Single Thread를 뛰워요!

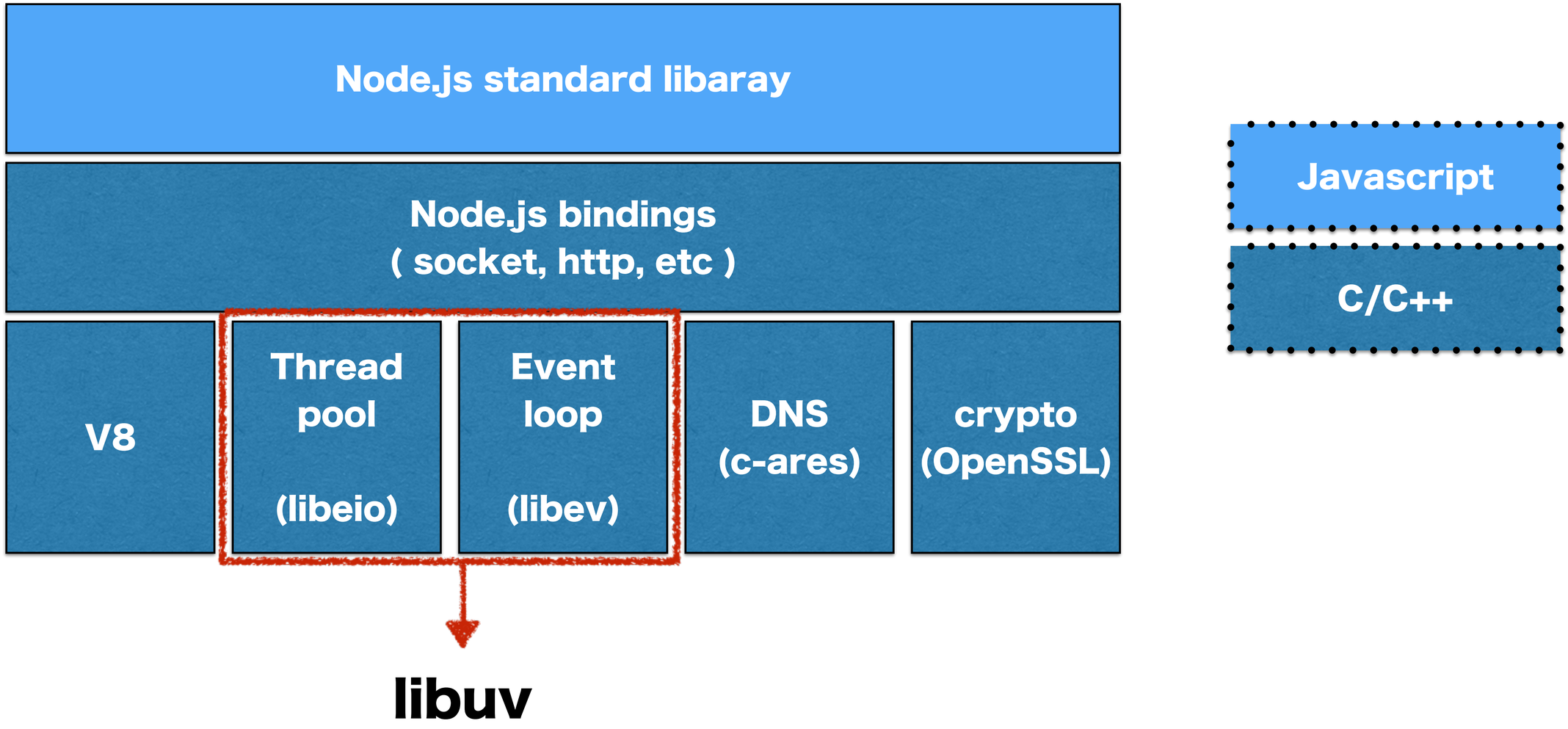


#05. Node.JS 아키텍처

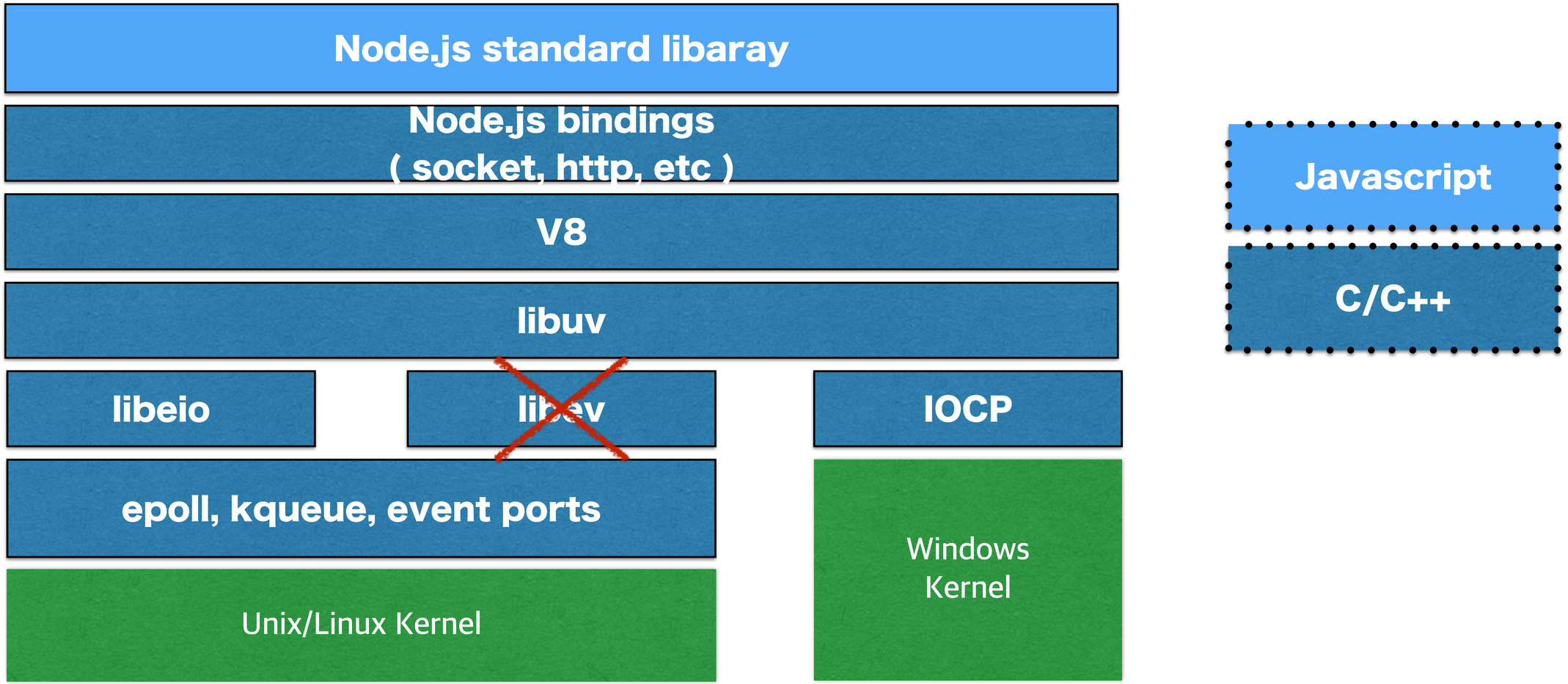
Node.js Event Loop



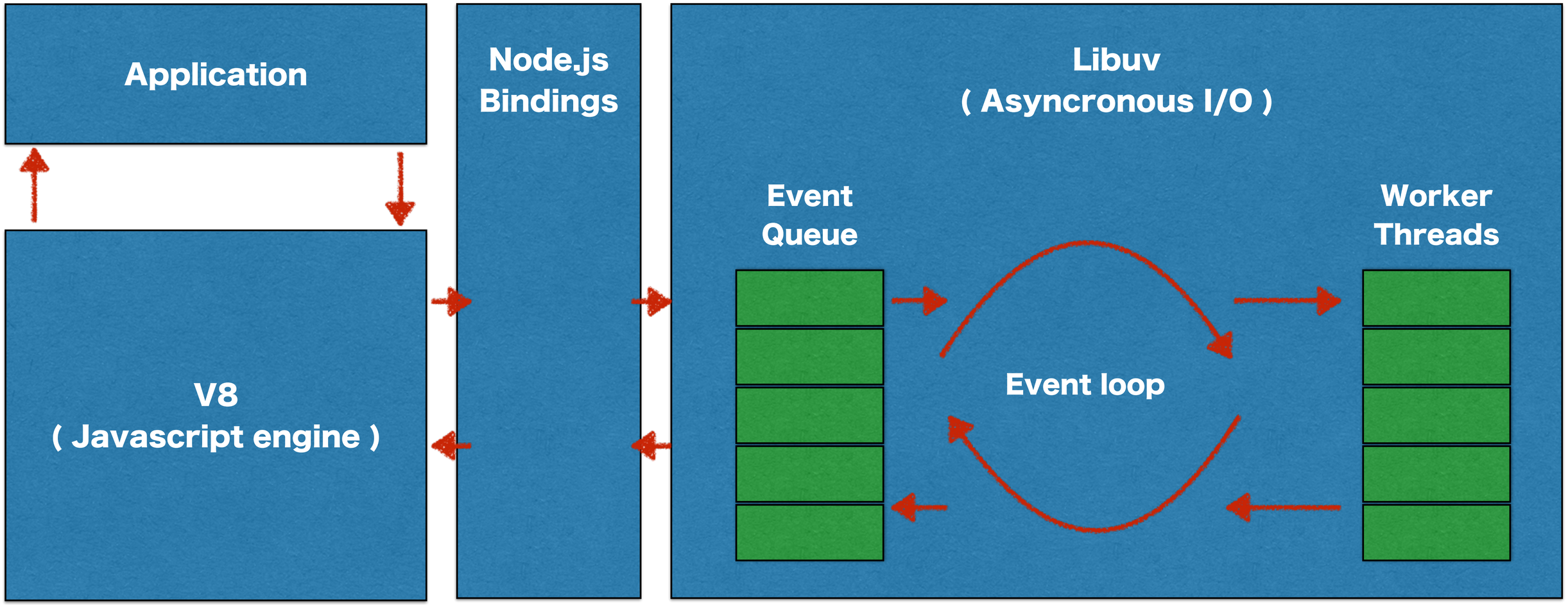
Node.js - Architecture



Node.js - Architecture



Node.js System

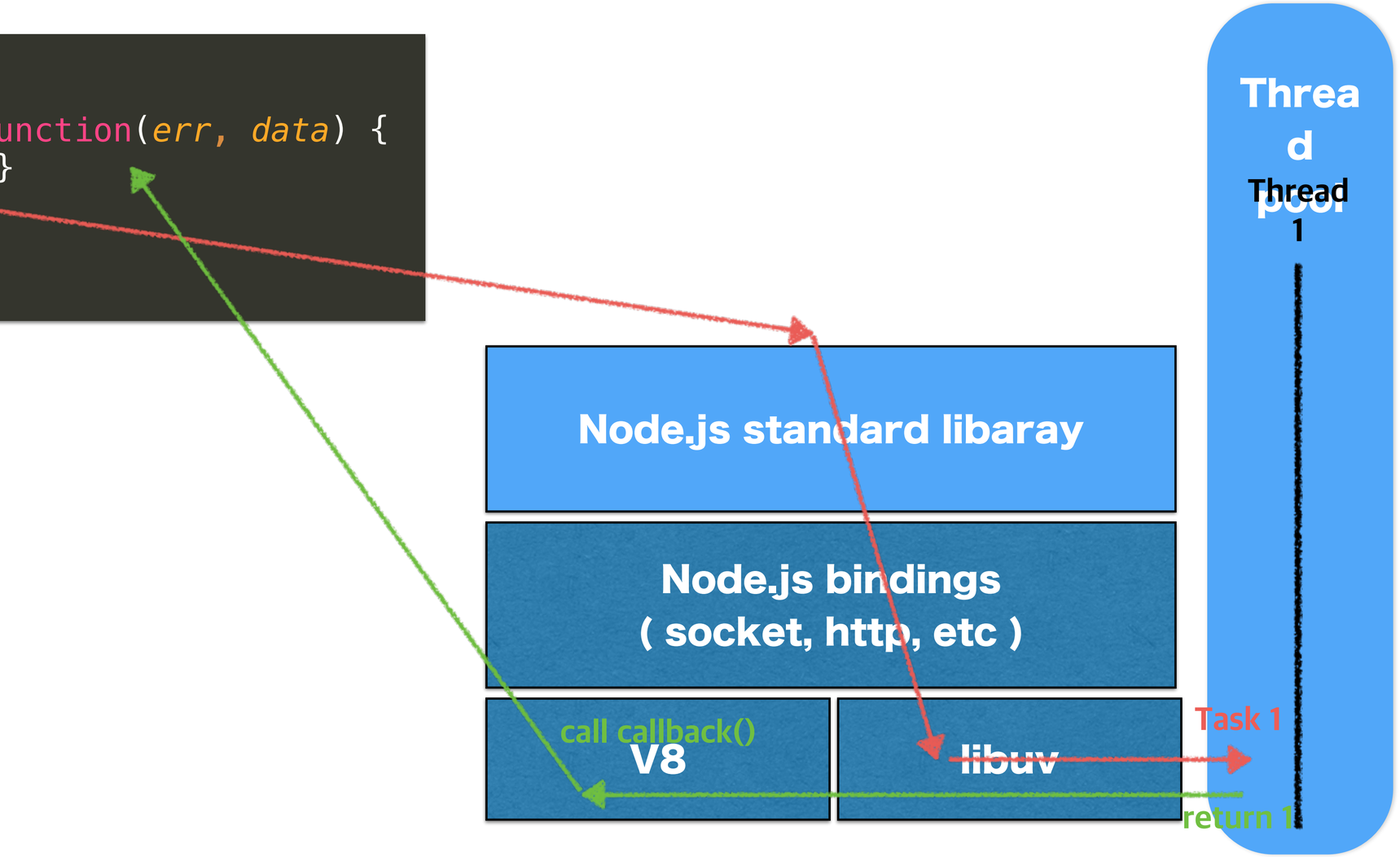


Node.js Event Loop

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

fs.readFile('./data.txt', function(err, data) {
    if( err ) { throw err; }

    console.log(data);
});
```

Node.js Event Loop

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

fs.readFile('./data.txt', function(err, data) {
    if( err ) { throw err; }

    console.log(data);
});
```



이렇게 짜면.. node 쓰지 마라!

```
for(i=0; i<5; i++) {
    sleep(1000);
}
```

EventLoop가 5초동안 멈춥니다!

node 쓸때 고려해라!

- Block 되게 하지 마라!
- 소스 코드 순서대로 순차적 실행되지 않는다.
- Event Loop 의 순서에 주의해라! http://bit.ly/1F4kDz4

node에게 맡겨라!

- Dispatch
- Concurrency
- Asycn Operation

Part 2.
Node.JS 실습

Node.js 설치하기!!

(Node.js, NVM, NPM)

NVM 설치

- Node Version Manager

```
$ ssh root@<<IPADDRESS>>
$ sudo apt-get update
$ sudo apt-get install build-essential libssl-dev
$ curl https://raw.githubusercontent.com/creationix/nvm/v0.22.0/install.sh | sh
$ source ~/.profile
$ nvm ls-remote
```

Node.js 설치

```
$ nvm install 0.10
$ nvm use 0.10.35
$ node -v
$ nvm alias default 0.10.35
$ nvm use default
```

Sublimetext sftp 설정

환경 설정

(Sublimetext - Package Control 설치)

View > Show Console 을 선택해서 Console 창을 열고 아래의 명령어를 입력

Sublimetext2

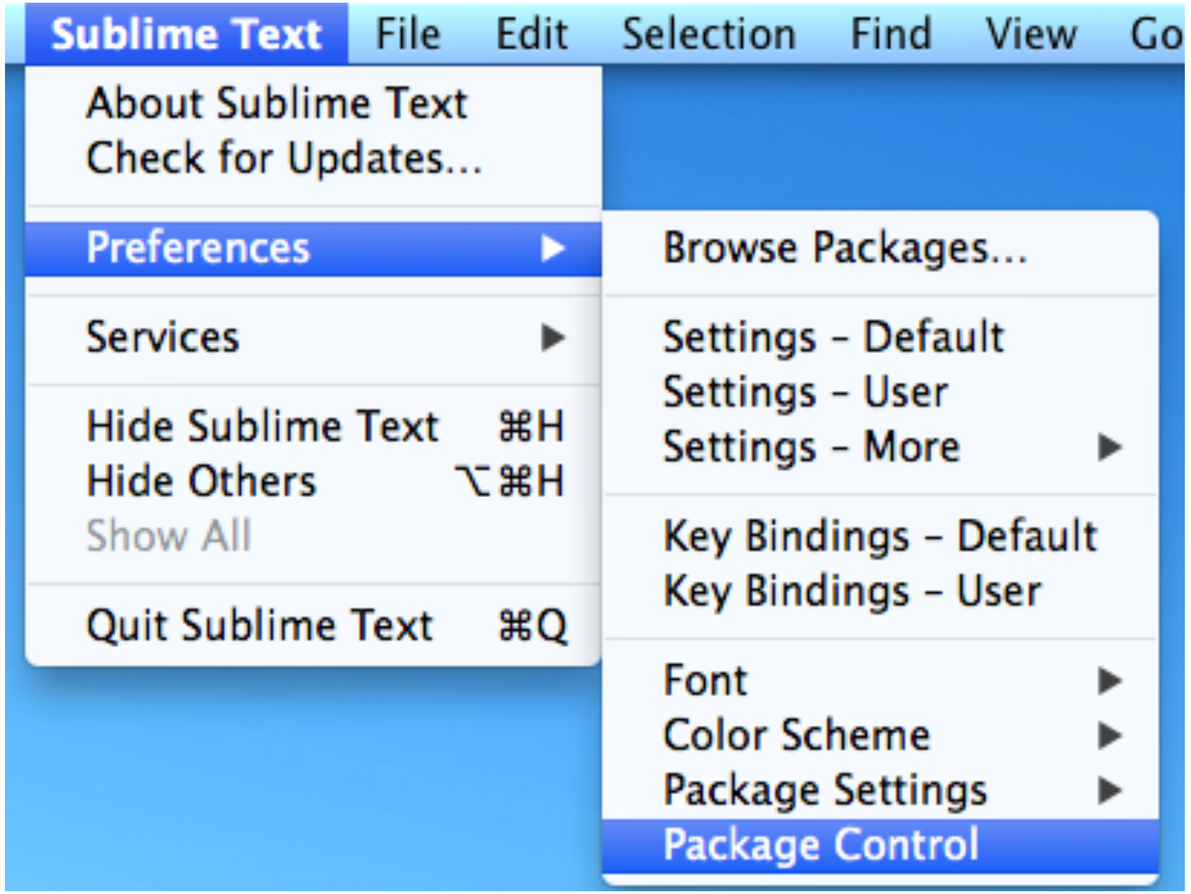
```
import urllib2,os,hashlib; h = '2deb499853c4371624f5a07e27c334aa' + 'bf8c4e67d14fb0525ba4f89698a6d7e1'; pf = 'Package Control.sublime-package'; ipp = sublime.installed_packages_path(); os.makedirs( ipp ) if not os.path.exists(ipp) else None; urllib2.install_opener( urllib2.build_opener( urllib2.ProxyHandler()) ); by = urllib2.urlopen( 'http://packagecontrol.io/' + pf.replace(' ', '%20')).read(); dh = hashlib.sha256(by).hexdigest(); open( os.path.join( ipp, pf), 'wb' ).write(by) if dh == h else None; print('Error validating download (got %s instead of %s), please try manual install' % (dh, h) if dh != h else 'Please restart Sublime Text to finish installation')
```

Sublimetext3

```
import urllib.request,os,hashlib; h = '2deb499853c4371624f5a07e27c334aa' + 'bf8c4e67d14fb0525ba4f89698a6d7e1'; pf = 'Package Control.sublime-package'; ipp = sublime.installed_packages_path(); urllib.request.install_opener( urllib.request.build_opener( urllib.request.ProxyHandler()) ); by = urllib.request.urlopen( 'http://packagecontrol.io/' + pf.replace(' ', '%20')).read(); dh = hashlib.sha256(by).hexdigest(); print('Error validating download (got %s instead of %s), please try manual install' % (dh, h)) if dh != h else open(os.path.join( ipp, pf), 'wb' ).write(by)
```

[출처] http://kr.tutorialboat.com/72

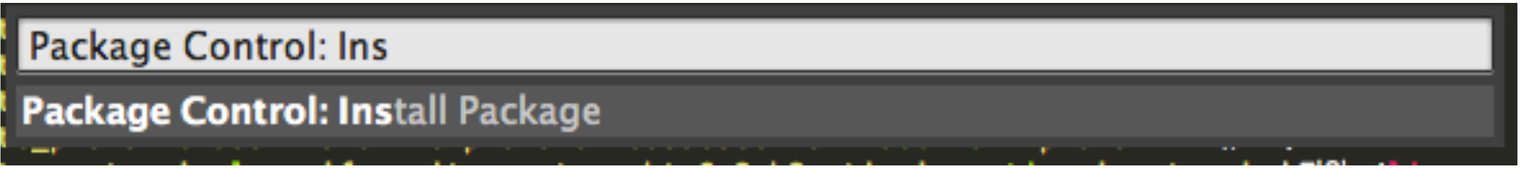
환경 설정
(Sublimetext - sftp 설치)



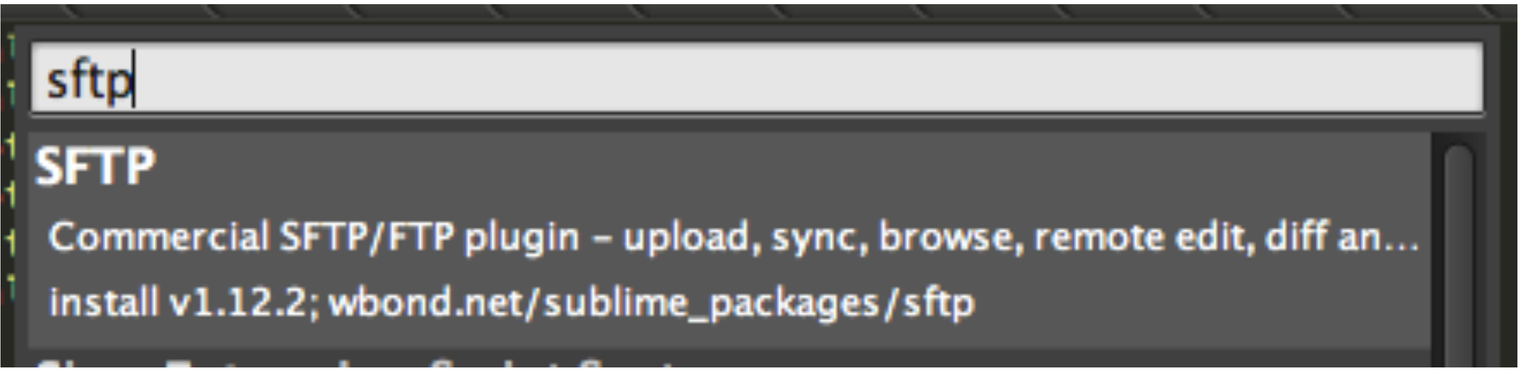
Package Control 실행

[출처] http://kr.tutorialboat.com/72

환경 설정
(Sublimetext - sftp 설치)



Package Control : install package 실행



sftp설치

[출처] http://kr.tutorialboat.com/72

환경 설정

(Sublimetext - sftp 설정)

sftp-config.json

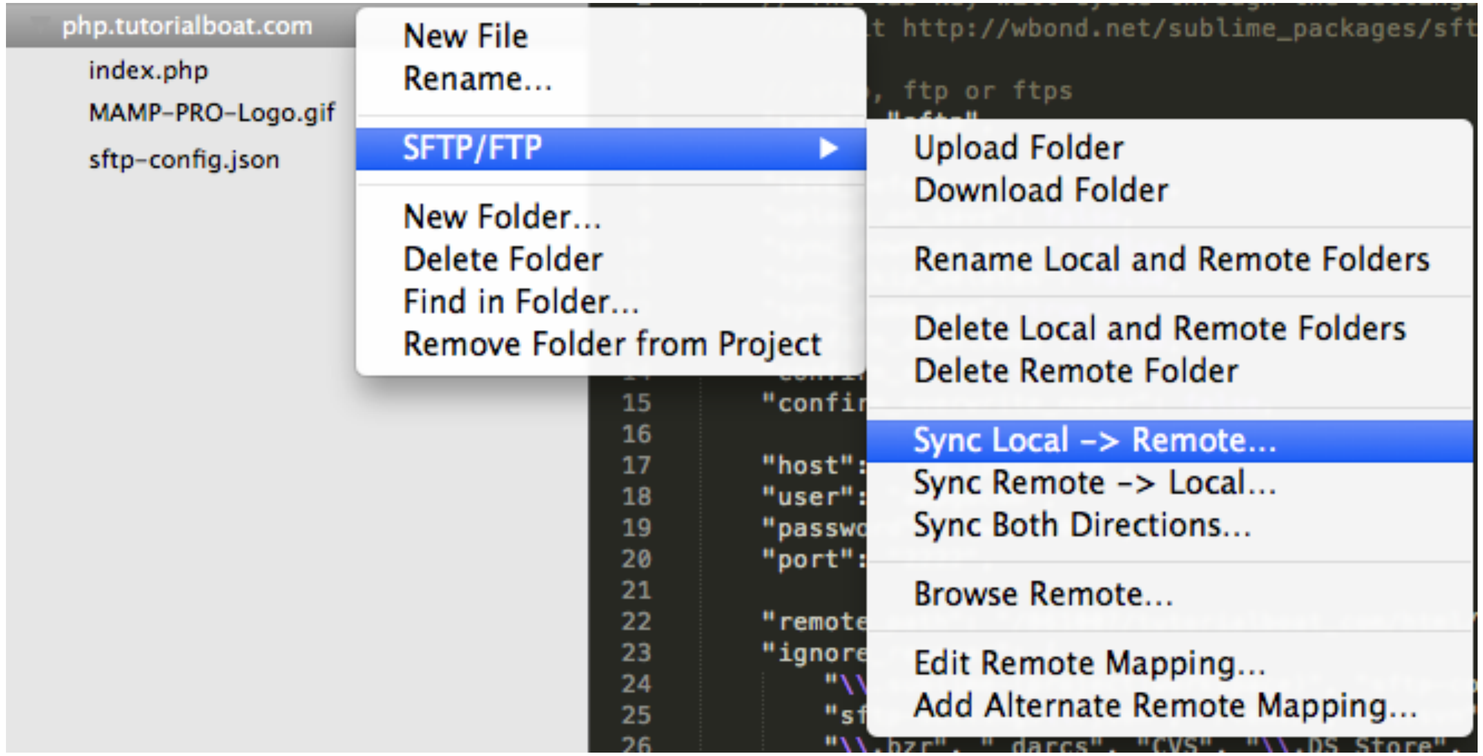
```
{
    // The tab key will cycle through the settings when first created
    // Visit http://wbond.net/sublime_packages/sftp/settings for help

    // sftp, ftp or ftps
    "type": "sftp",

    "save_before_upload": true,
    "upload_on_save": false,
    "sync_down_on_open": false,
    "sync_skip_deletes": false,
    "sync_same_age": true,
    "confirm_downloads": false,
    "confirm_sync": true,
    "confirm_overwrite_newer": false,

    "host": "[redacted]",
    "user": "[redacted]",
    "password": "[redacted]",
    "port": "[redacted]",

    "remote_path": "[redacted]",
    "ignore_regexes": [
        "\\.sublime-(project|workspace)", "sftp-config(-alt\\d?)?\\.json",
        "sftp-settings\\.json", "/venv/", "\\.svn", "\\.hg", "\\.git",
        "\\.bzr", "_darcs", "CVS", "\\.DS_Store", "Thumbs\\.db", "desktop\\.ini"
    ],
    //"file_permissions": "664",
    //"dir_permissions": "775",

    //"extra_list_connections": 0,

    "connect_timeout": [redacted],
    "keepalive": [redacted],
    //"ftp_passive_mode": true,
    //"ssh_key_file": "~/.ssh/id_rsa",
    //"sftp_flags": ["-F", "/path/to/ssh_config"],

    //"preserve_modification_times": false,
    //"remote_time_offset_in_hours": 0,
    //"remote_encoding": "utf-8",
    //"remote_locale": "C",
}
```

“host” : <<IPADDRESS>>,
“user”: “root”,
“password”: “olcolcolc”,

“remote_path”: “/root/”

[출처] http://kr.tutorialboat.com/72

환경 설정
(Sublimetext)



sync 테스트

[출처] http://kr.tutorialboat.com/72

Node.js 실습#0

(hello nodejs)

Node.js 실습#0

```
$ mkdir nodestudy
$ cd nodestudy
```

Node.js 실습#0

(전역객체 - helloConsole.js)

```
console.log('hello nodejs!!');

var json = { msg: ‘Hello nodejs!!'};
console.log('My name is %j', json);

var a = 10;
var b = 20;
console.log('a + b = %d', a+b);

console.time('timeout');
setTimeout(function() {
  console.timeEnd('timeout');
},1000);

console.log('filename : ', __filename);
console.log('dirname : ', __dirname);
```

Node.js 실습#0
(전역객체 - helloConsole.js)

```
$ node helloConsole
```

Node.js 실습#0

(전역객체 - helloConsole.js)

```
console.log('hello nodejs!!');

var json = { msg: ‘Hello nodejs!!’};
console.log('My name is %j', json);

var a = 10;
var b = 20;
console.log('a + b = %d', a+b);

console.time('timeout');
setTimeout(function() {
  console.timeEnd('timeout');
},1000);

console.log('filename : ', __filename);
console.log('dirname : ', __dirname);
```

Node.js 실습#0

(전역객체 - helloProcess.js)

```
console.log('env = ', process.env);

console.log('version = ', process.version);

console.log('versions = ', process.versions);

console.log('platform = ', process.platform);

console.log('memory usage = ', process.memoryUsage());

console.log('up time = ', process.uptime());
```

Node.js 실습#0
(전역객체 - helloProcess.js)

```
$ node helloProcess
```

Node.js 실습#0

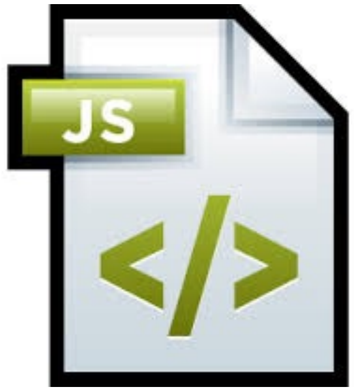
(전역객체 - helloProcess.js)

```
console.log('env = ', process.env);

console.log('version = ', process.version);

console.log('versions = ', process.versions);

console.log('platform = ', process.platform);

console.log('memory usage = ', process.memoryUsage());

console.log('up time = ', process.uptime());
```

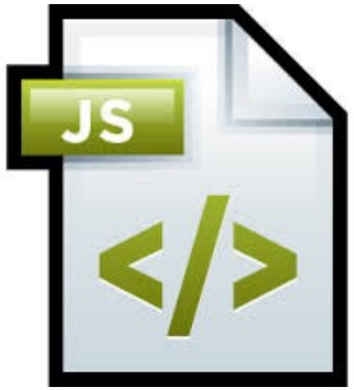
Node.js 실습#1

(모듈과 친해지기)

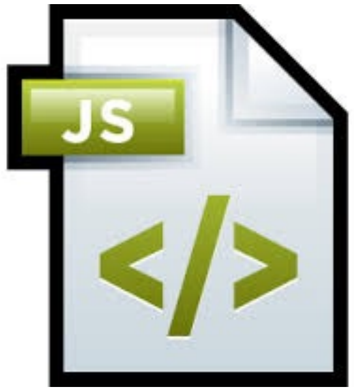
Node.js 실습#1
(module system)



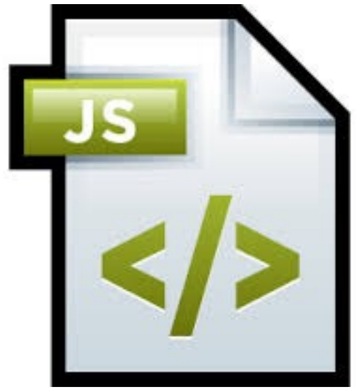
http



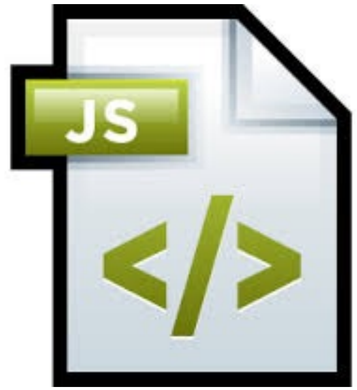
Stream



net



fs



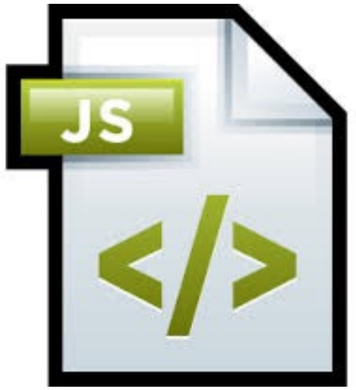
EventEmitter

...

Node.js 실습#1
(module system)

app.js

require('math')



math.js

exports.abs
exports.pow

Node.js 실습#1

(math.js)

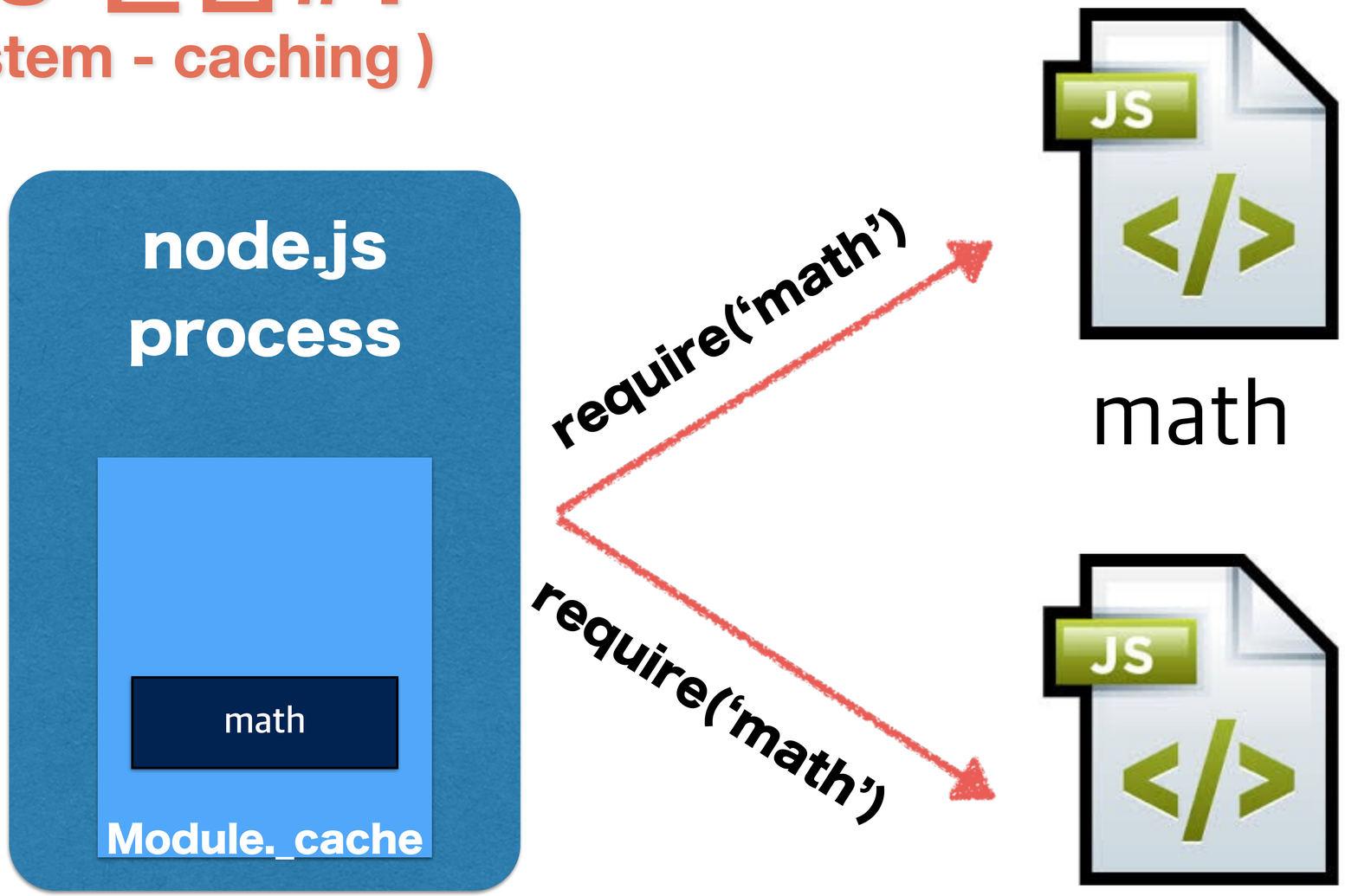
```
exports.abs = function(number) {
    return number < 0 ? -number : number;
};

exports.pow = function(x, y) {
    var value = 1;

    for( var i = 0; i < y; i++ ) {
        value *= x;
    }

    return value;
};
```

(app.js)

```
var math = require('./math');

console.log('math.abs(%d) = %d', -100, math.abs(-100));
console.log('math.pow(%d, %d) = %d', 2, 3, math.pow(2, 3));
```

Node.js 실습#1

(module system - caching)



Node.js 실습#1
(npm)

npm 사용해보기

Node.js 실습#1
(npm)

```
$ npm init
```

Node.js 실습#1
(npm - package.json)

```
{
  "name": "nodestudy",
  "version": "1.0.0",
  "description": "",
  "main": "helloConsole.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Node.js 실습#1
(npm - package.json)

```
{
  "name": "nodestudy",
  "version": "1.0.0",
  "description": "",
  "main": "helloConsole.js",
  "scripts": {
    "start": "node xxxx.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Node.js 실습#1
(npm)

colors 모듈 설치

Node.js 실습#1
(npm)

```
$ npm install colors --save
```

Node.js 실습#1
(npm - package.json)

```
{
  "name": "nodestudy",
  "version": "1.0.0",
  "description": "",
  "main": "helloConsole.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "colors": "^1.0.3"
  }
}
```

Node.js 실습#1
(helloColors.js)

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

console.log('hello'.green);
console.log('i like cake and pies'.underline.red);
console.log('inverse the color'.inverse);
console.log('OMG Rainbows!'.rainbow);
console.log('Run the trap'.trap);
```

Node.js 실습#1
(EventEmitter)

EventEmitter?

Many objects in Node emit events: a net.Server emits an event each time a peer connects to it, a fs.readStream emits an event when the file is opened. **All objects which emit events are instances of events.EventEmitter.** You can access this module by doing: require("events");

Typically, event names are represented by a camel-cased string, however, there aren't any strict restrictions on that, as any string will be accepted.

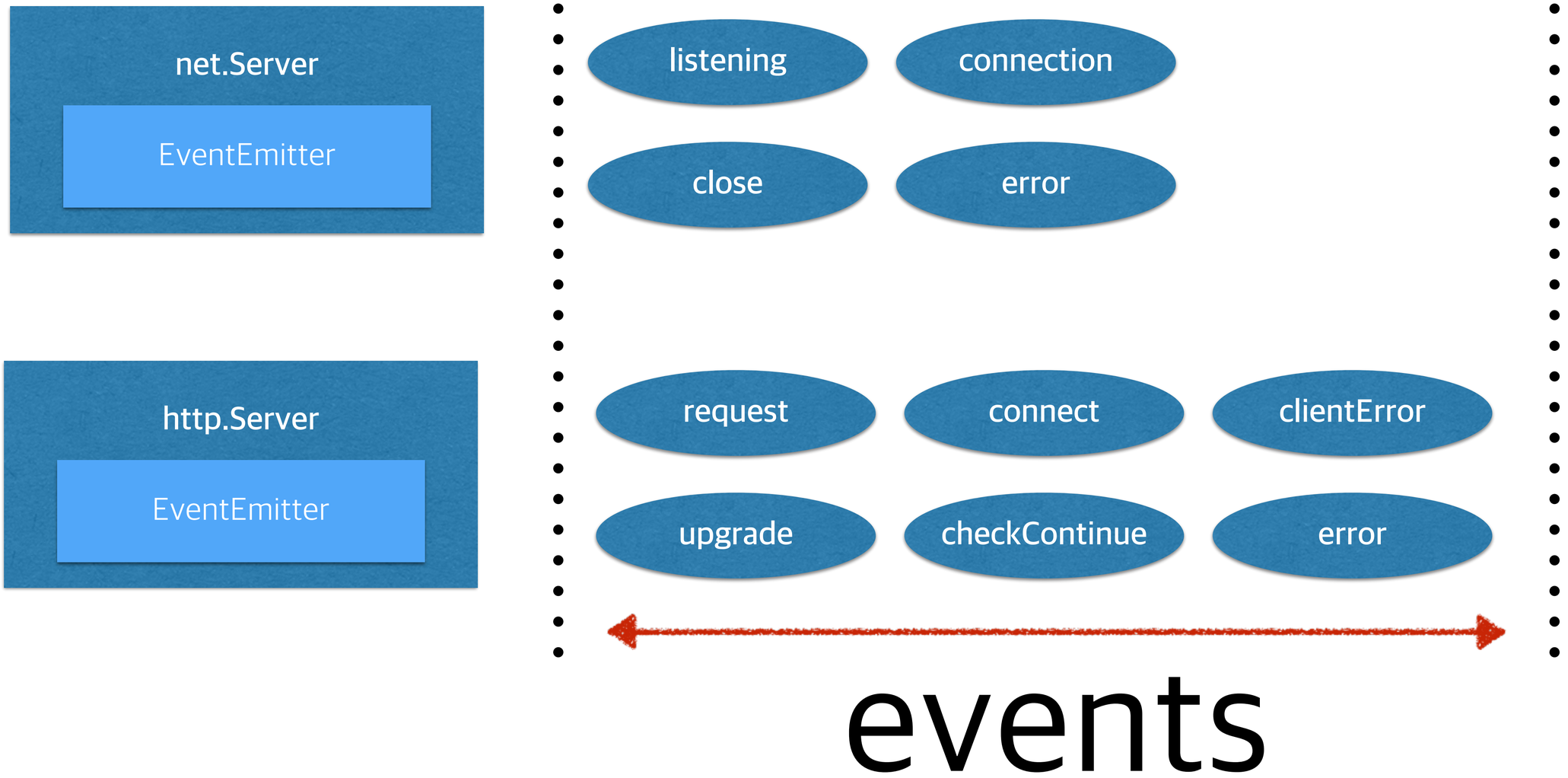
Functions can then be attached to objects, to be executed when an event is emitted. These functions are called listeners. Inside a listener function, this refers to the EventEmitter that the listener was attached to.

[출처] http://nodejs.org/api/events.html

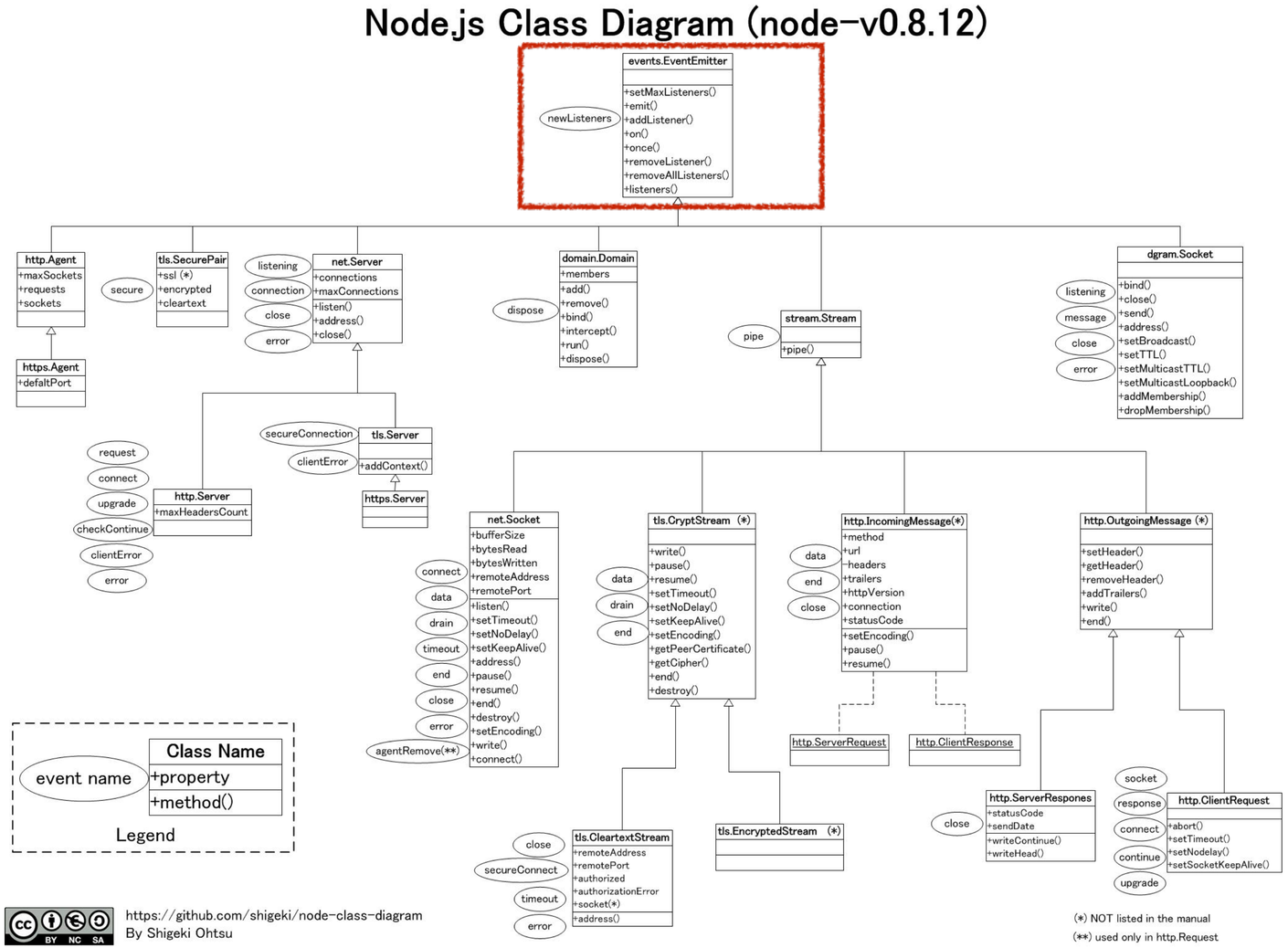
Node.js 실습#1
(EventEmitter)

request

Node.js 실습#1
(EventEmitter)



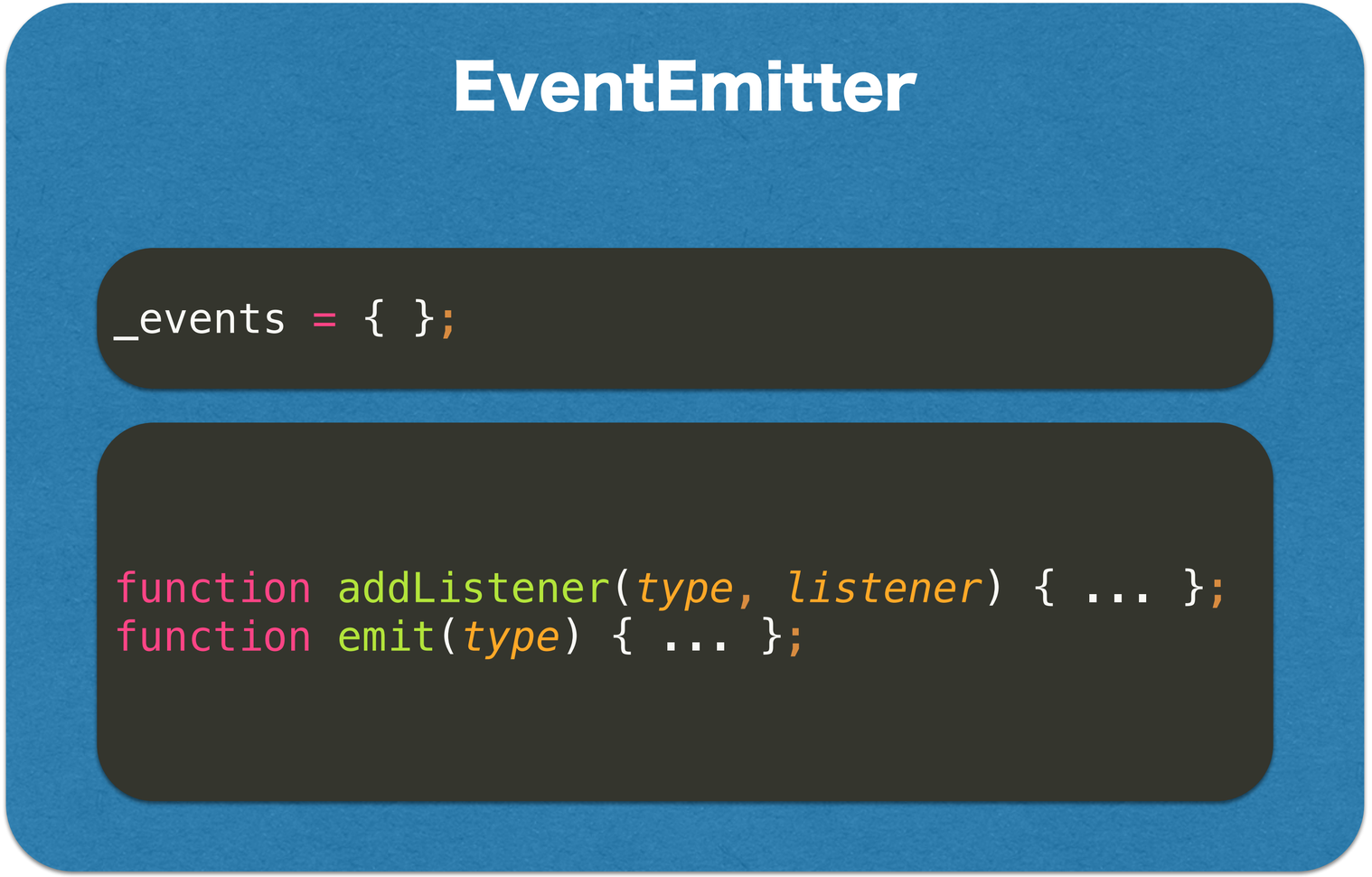
Node.js 실습#1
(EventEmitter)



Node.js 실습#1
(EventEmitter)

```
var EventEmitter = require('events').EventEmitter;
var eventEmitter = new EventEmitter();


eventEmitter.on('customevent', function() {
    console.log('Event Occur!!');
});

eventEmitter.emit('customevent');
```

Node.js 실습#1
(EventEmitter - addListener)



Node.js 실습#1
(EventEmitter - addListener)

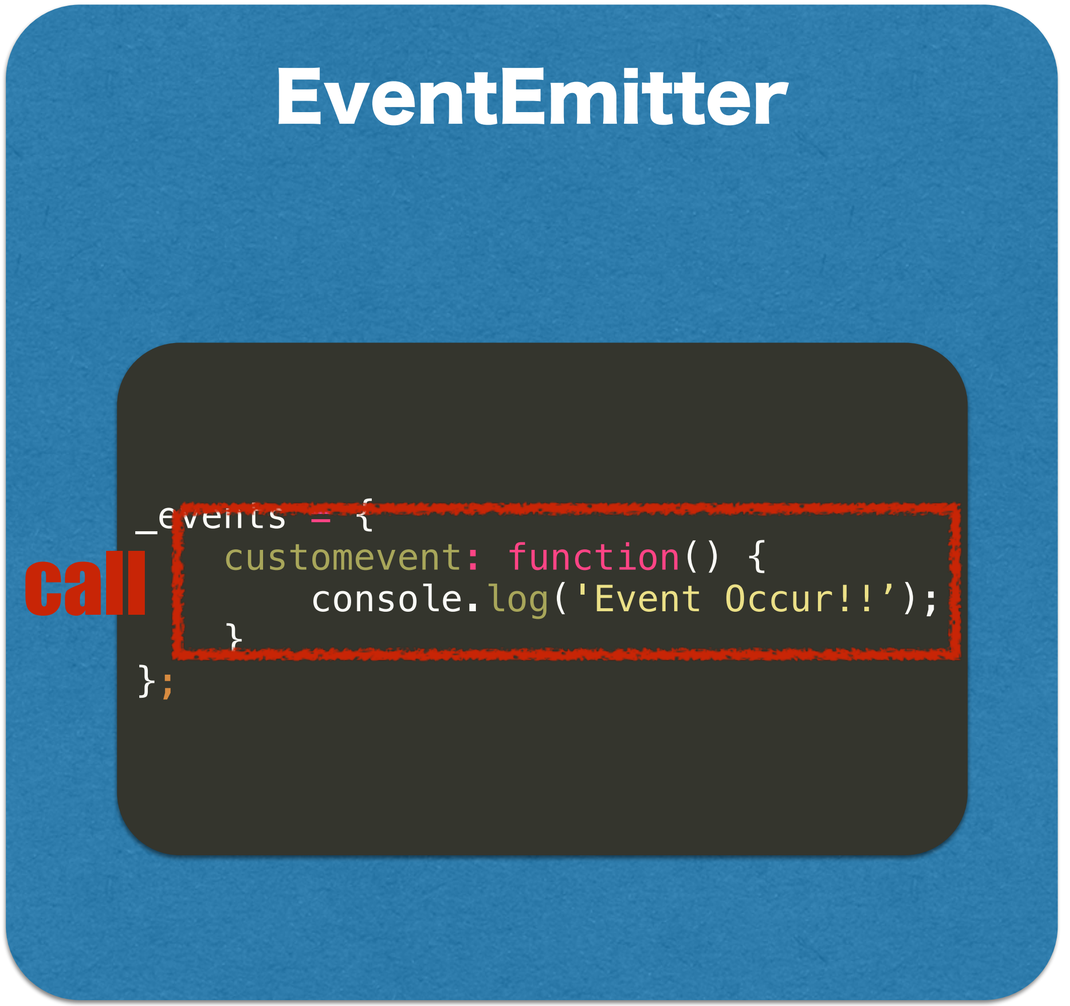
EventEmitter

```
_events = {
    customevent: function() {
        console.log('Event Occur!!');
    }
};
```

```
addListener('customevent', function() {
    console.log('Event Occur!!');
});
```

Node.js 실습#1
(EventEmitter - emit)

```
emit('customevent');
```



Node.js 실습#1
(EventEmitter - timer.js)

```
var EventEmitter = require('events').EventEmitter;
var timer = new EventEmitter();

var count = 0;
var interval = 1000;

timer.on('timed', function(count, uptime) {
    console.log('Event!! count : %d, uptime : %d ms', count, uptime);
});

setInterval(function() {
    timer.emit('timed', ++count, count * interval);
}, interval);
```

Node.js 실습#1
(Stream)

Stream?

데이터가 열을 지어 흐르는 것처럼 입력되는 것. 데이터 통신에서 데이터를 비트의 열로 변환하여 직렬로 전송하는 것.

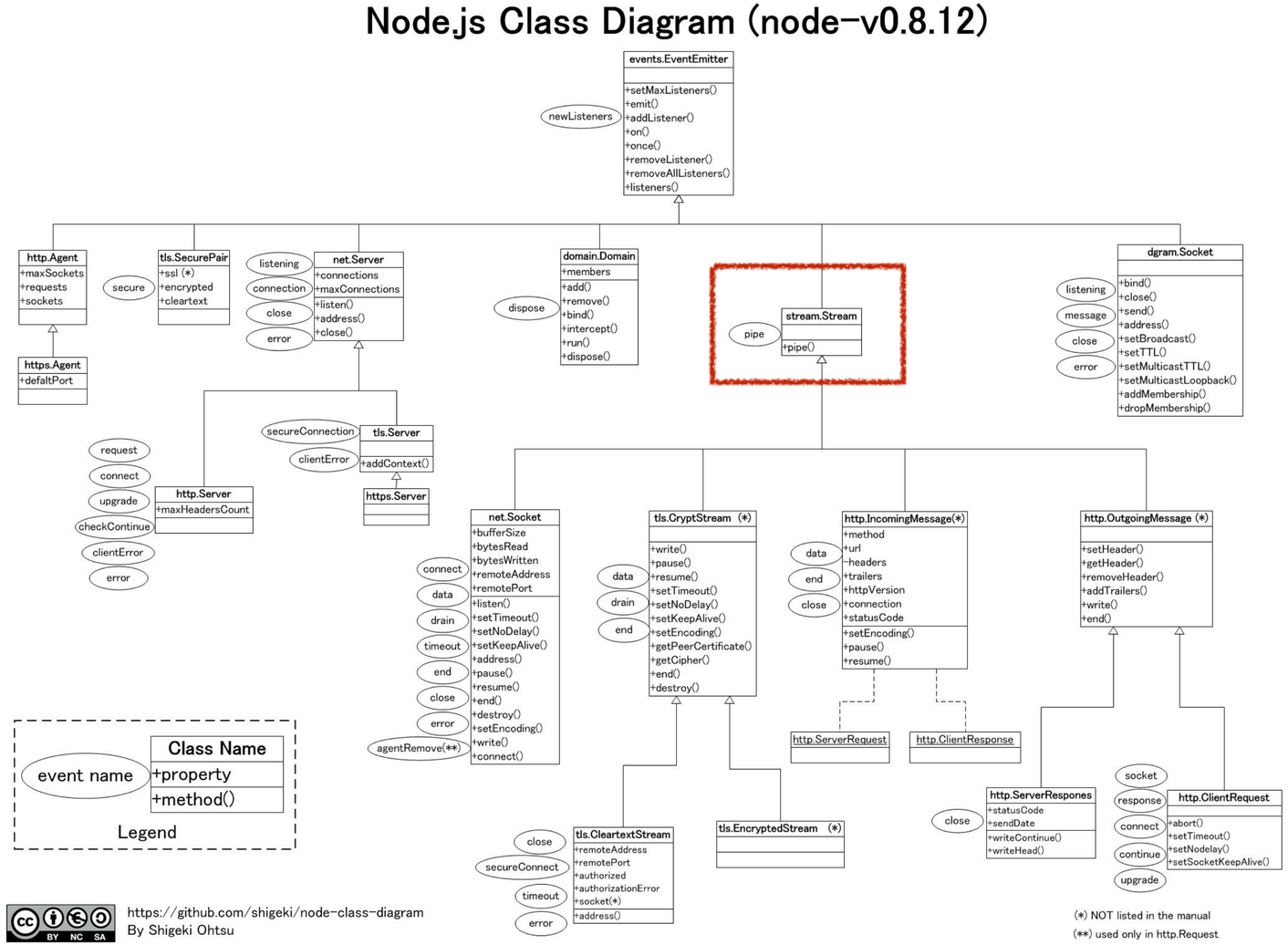
[네이버 지식백과] 데이터 스트림 [data stream] (컴퓨터인터넷IT용어대사전, 2011.1.20, 일진사)

A stream is an **abstract interface** implemented by various objects in Node. For example a request to an HTTP server is a stream, as is stdout. Streams are readable, writable, or both. **All streams are instances of EventEmitter**

[출처] http://nodejs.org/api/stream.html

Node.js 실습#1

(Stream)



Node.js 실습#1
(Stream)

Stream
(Readable)

Stream
(Writable)

Node.js 실습#1
(Stream)

- File 입/출력
- Socket 입/출력
- stdin/stdout

Node.js 실습#1

(Stream - module_stream.js)

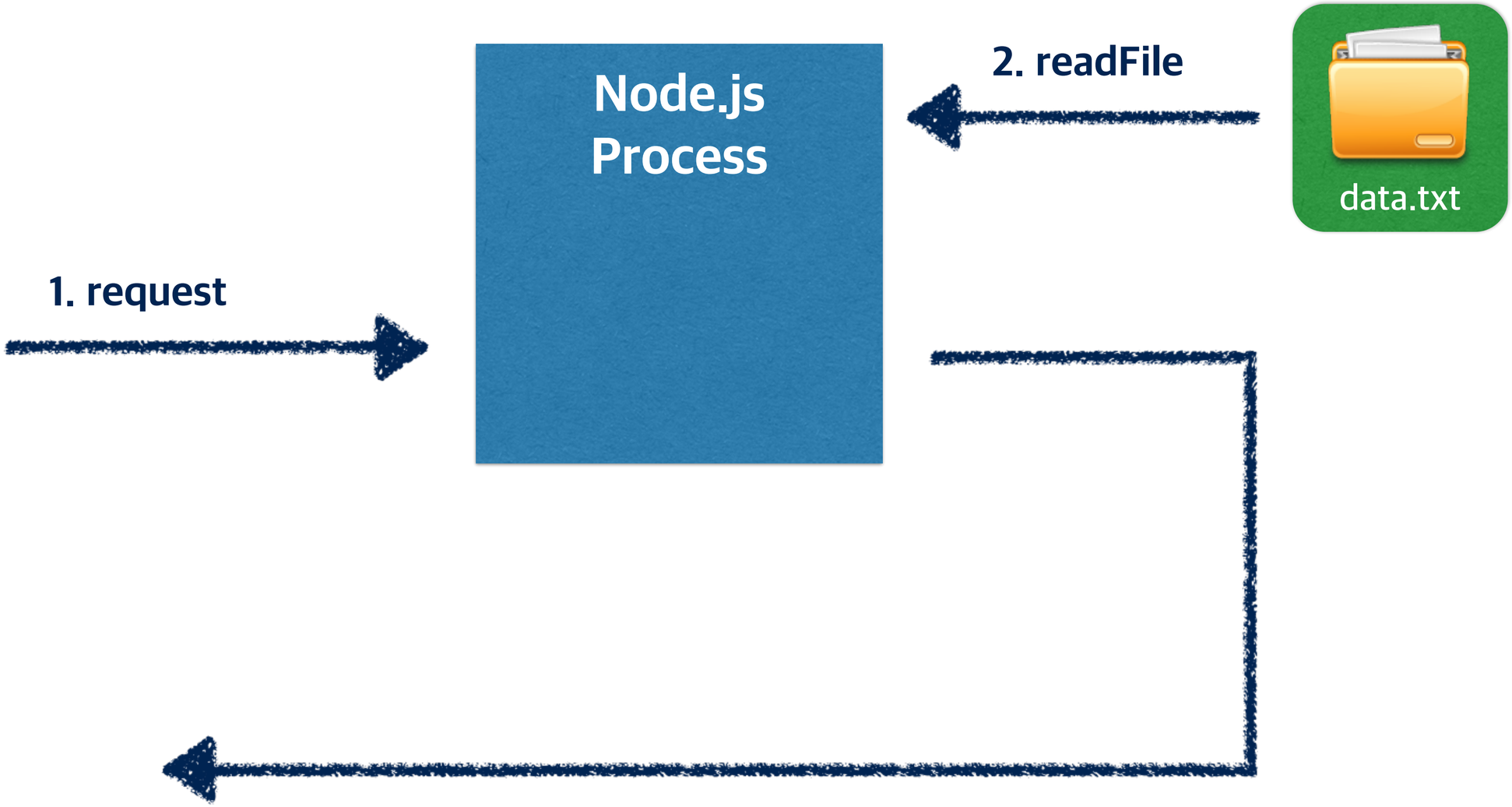
```
var http = require('http');
var fs = require('fs');

var server = http.createServer(function (req, res) {
    // req is an http.IncomingMessage, which is a Readable Stream
    // res is an http.ServerResponse, which is a Writable Stream

    fs.readFile(__dirname + '/data.txt', function (err, data) {
        res.end(data);
    });
});

server.listen(8000);
```

Node.js 실습#1

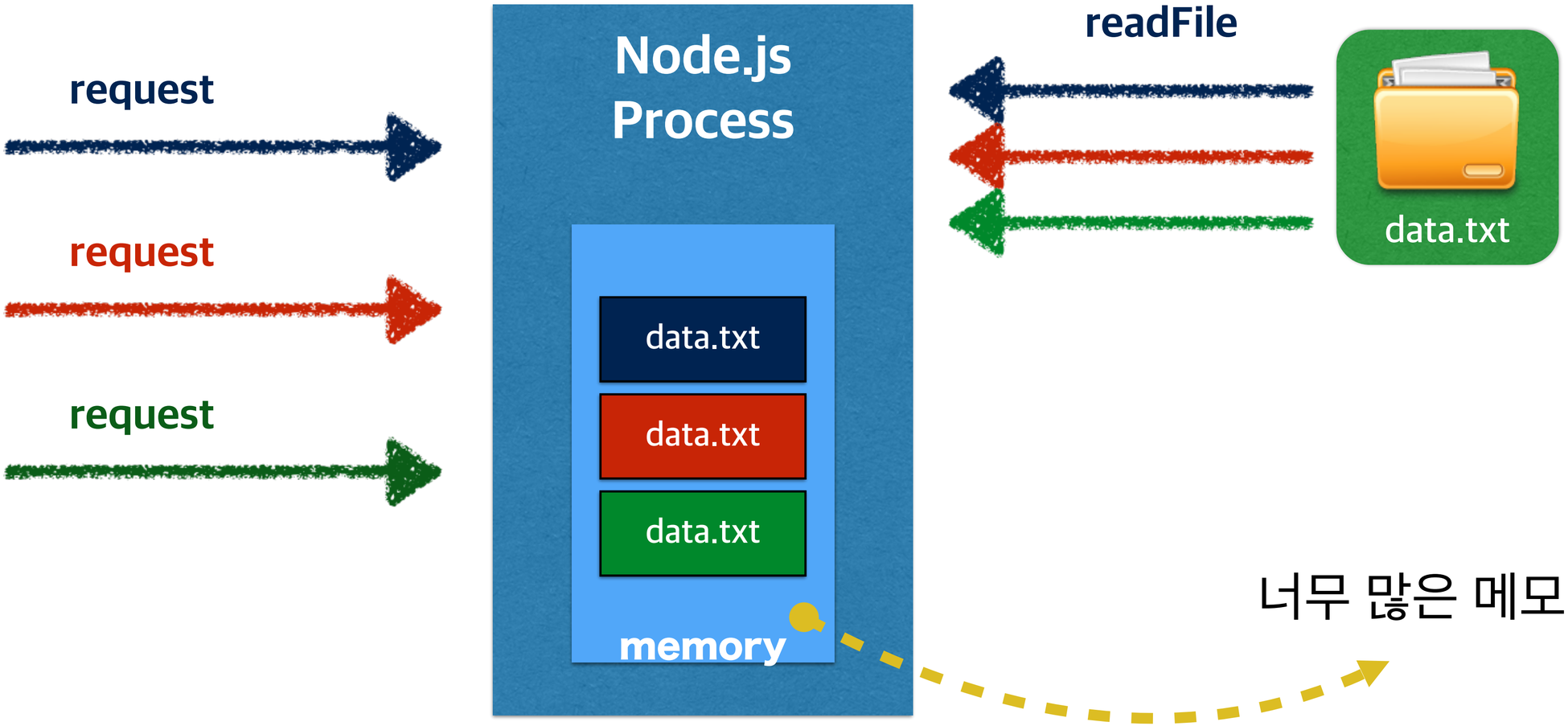
(Stream - module_stream.js)

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

var server = http.createServer(function (req, res) {
    // req is an http.IncomingMessage, which is a Readable Stream
    // res is an http.ServerResponse, which is a Writable Stream

    fs.readFile(__dirname + '/data.txt', function (err, data) {
        res.end(data);
    });
});

server.listen(8000);
```

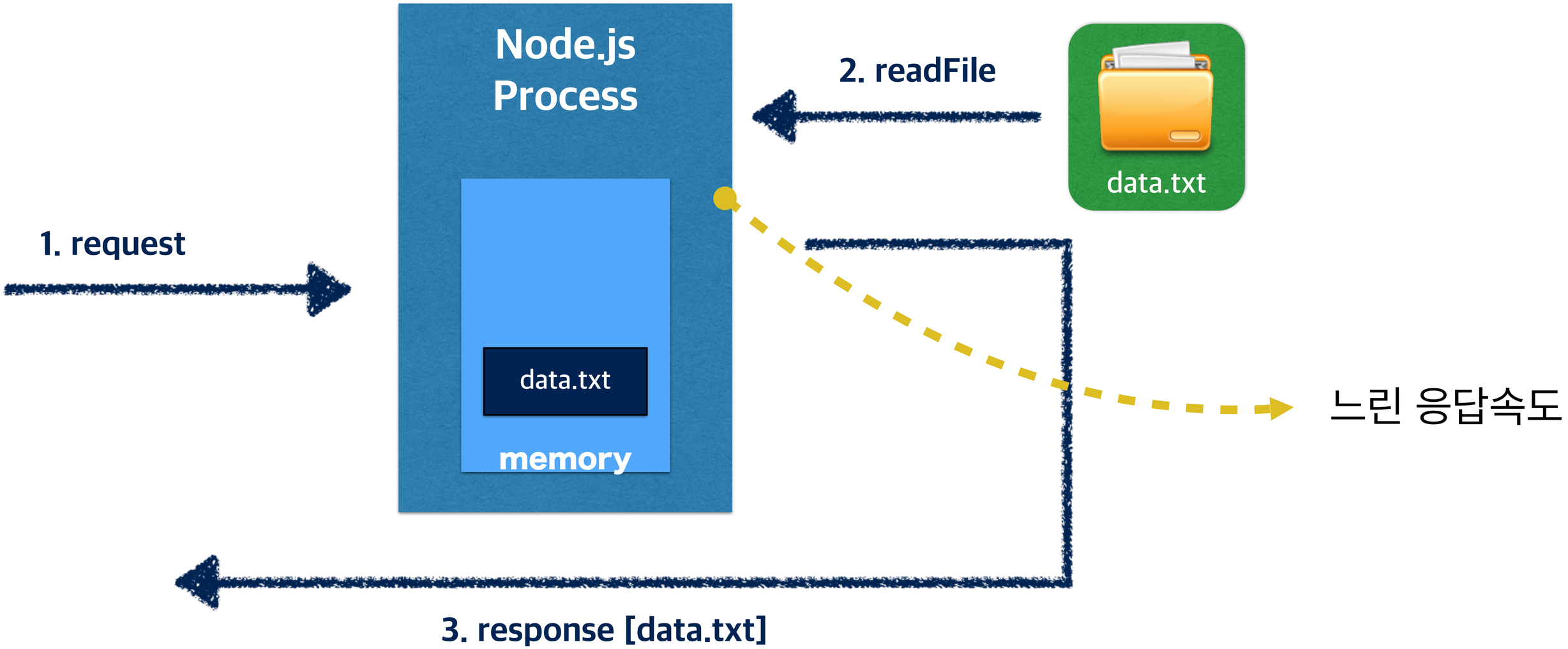
Node.js 실습#1
(Stream - module_stream.js 동작)



Node.js 실습#1
(Stream - module_stream.js 단점-1)



Node.js 실습#1
(Stream - module_stream.js 단점-2)

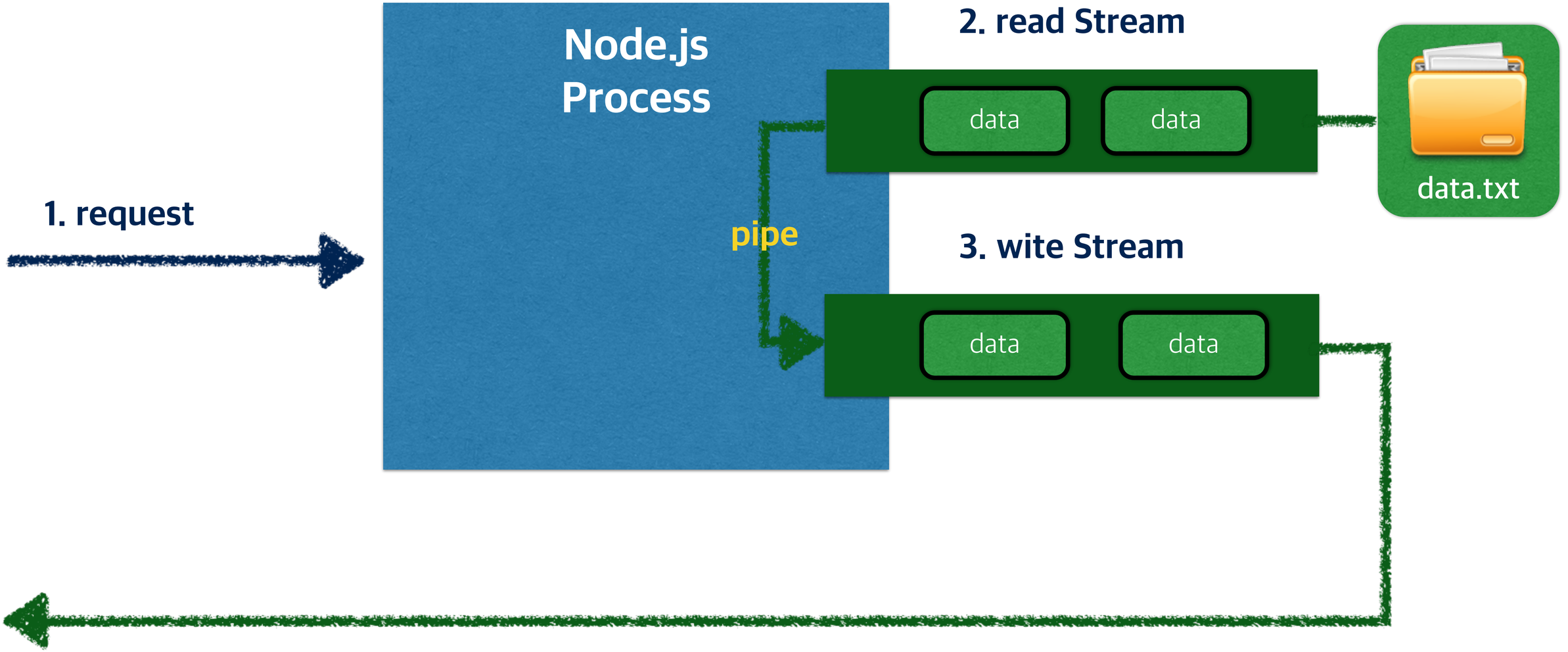


Node.js 실습#1

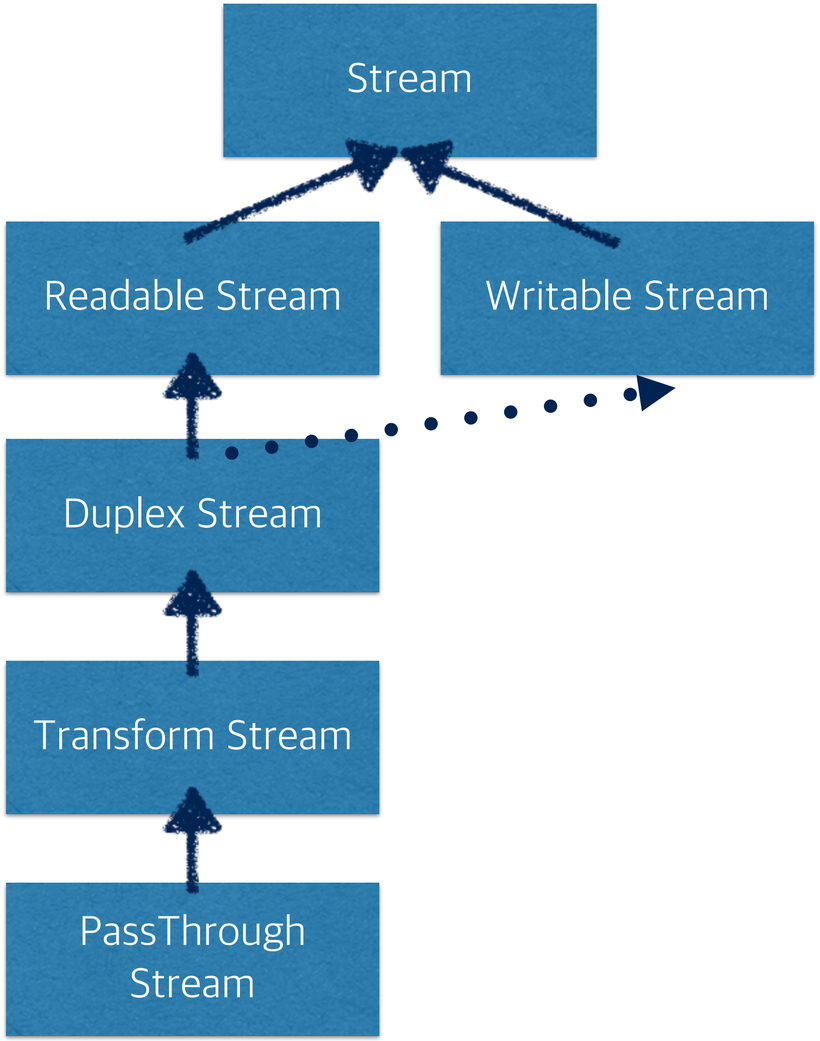
(Stream - module_stream.js)

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

var server = http.createServer(function (req, res) {
    // req is an http.IncomingMessage, which is a Readable Stream
    // res is an http.ServerResponse, which is a Writable Stream

    //fs.readFile(__dirname + '/data.txt', function (err, data) {
    //    res.end(data);
    //});
    var stream = fs.createReadStream(__dirname + '/data.txt');
    stream.pipe(res);
});

server.listen(8000);
```

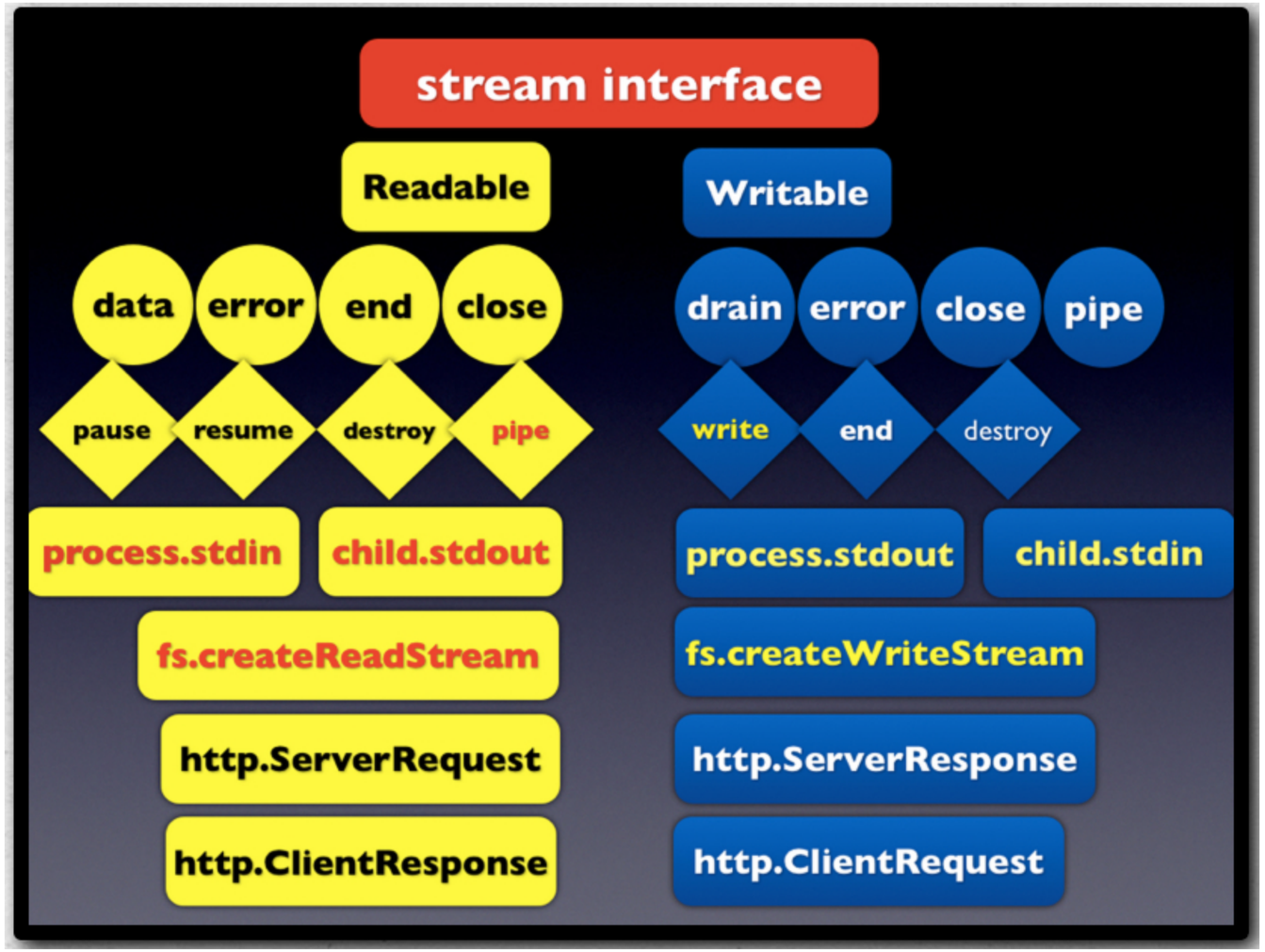
Node.js 실습#1
(Stream - module_stream.js 동작)



Node.js 실습#1
(Stream)



Node.js 실습#1
(Stream)



[출처] http://nodeqa.com/nodejs_ref/61

Node.js 실습#1
(npm)

```
$ sudo npm install socket.io  --save
$ sudo npm install socket.io-client --save
```

Node.js 실습#1
(echoserver.js)

```
var Server = require('socket.io');
var io = Server();

io.on('connect', function(socket){
    console.log('socket connected');

    socket.on('message', function(data) {
        console.log('receive : ', data);
        socket.send('echo : '+ data);
    });

    socket.on('disconnect', function() {
        console.log('socket disconnected');
    })
});

io.listen(3000);
```

Node.js 실습#1

(echoclient.js)

```
var io = require('socket.io-client');
var socket = io('http://127.0.0.1:3000');

socket.on('connect', function() {
    console.log('connected!!');
});
socket.on( 'message', function( data) {
    console.log( "received : ", data);
});
process.stdin.resume();
process.stdin.setEncoding('utf8');
process.stdin.on('data', function (chunk) {
    if( !socket.disconnected ) {
        socket.send(chunk);
        console.log('send : ' + chunk);
    }
});
```

Node.js 실습#2

(RESTful API Server - MemoServer)

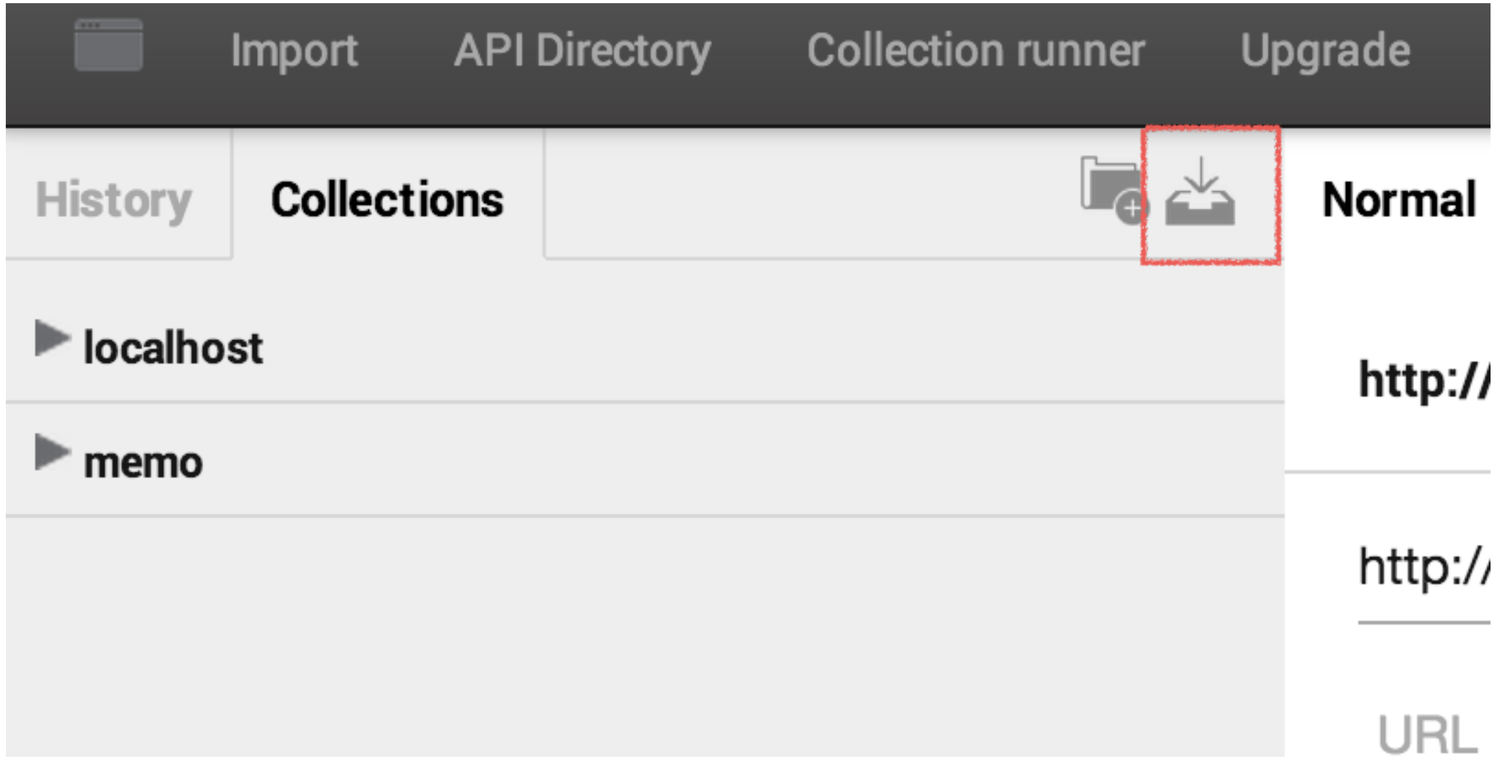
Node.js 실습#2
(Postman 설치)

Postman?

Postman helps you be more efficient while working with APIs. Postman is a scratch-your-own-itch project. The need for it arose while one of the developers was creating an API for his project. After looking around for a number of tools, nothing felt just right. The primary features added initially were a history of sent requests and collections.

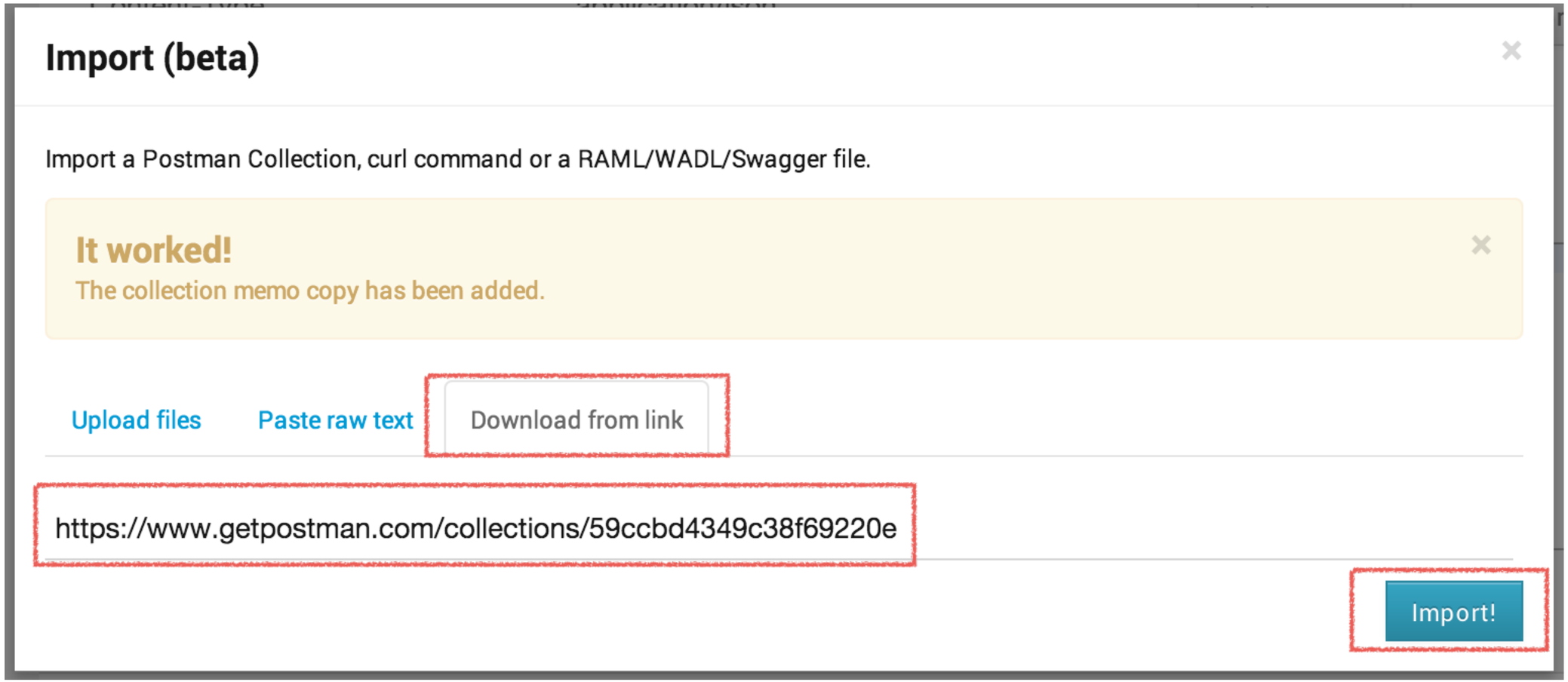
https://chrome.google.com/webstore/detail/postman-rest-client-packa/fhbjgbiflinjbdggehcddcbncdddomop?utm_source=gmail

Node.js 실습#2
(Postman 설치)



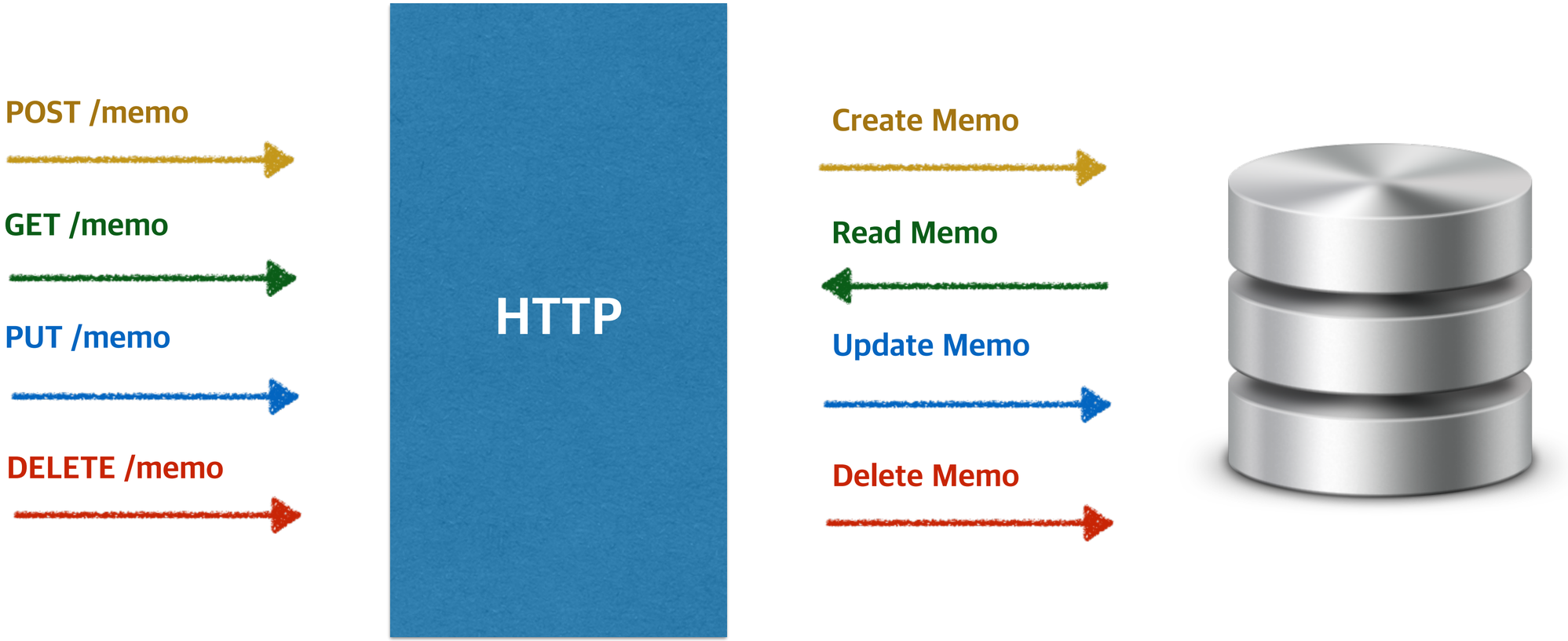
Node.js 실습#2

(Postman 설치)



https://www.getpostman.com/collections/59ccbd4349c38f69220e

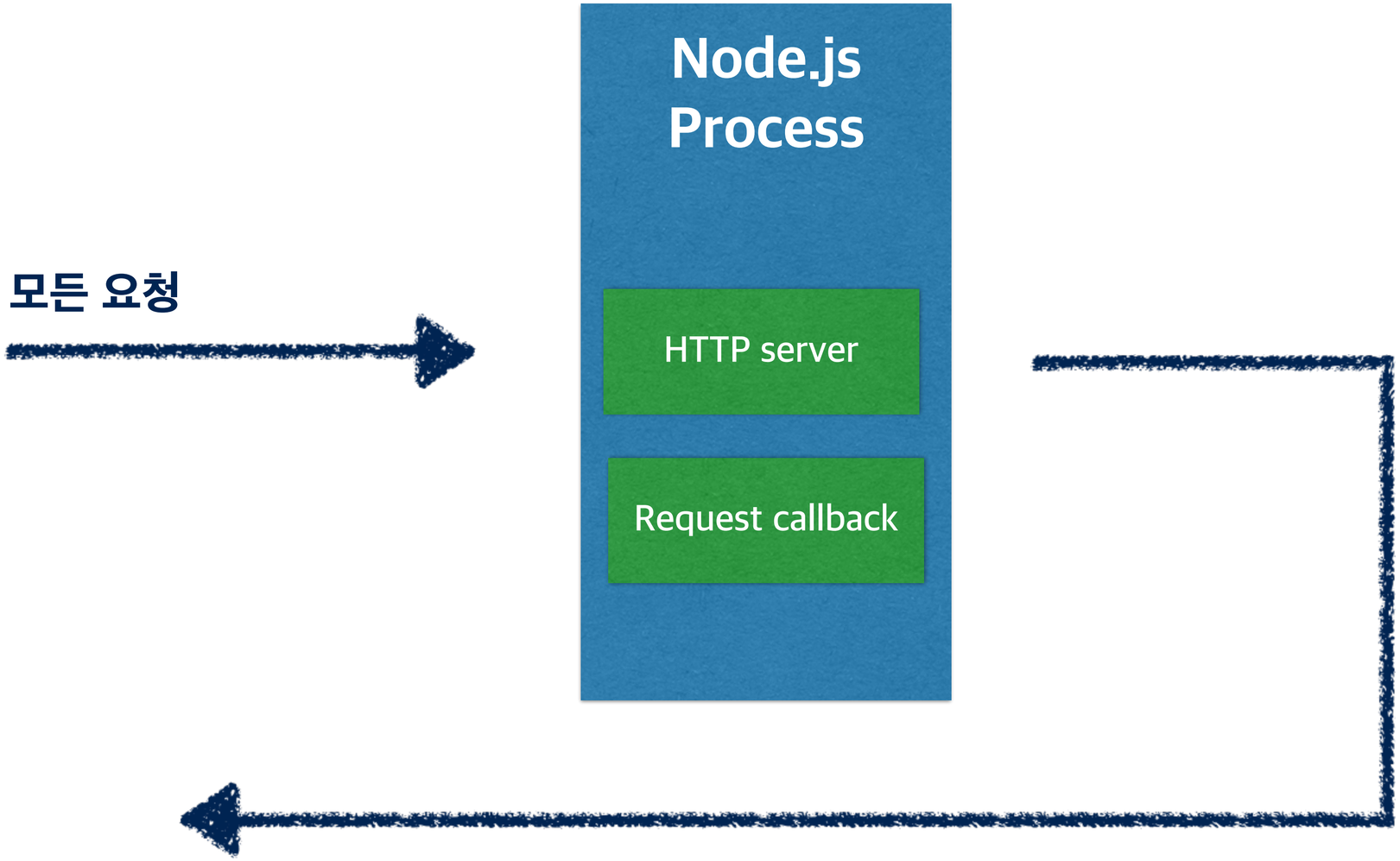
Node.js 실습#2
(MemoServer 기능 설명)



Node.js 실습#2
(기본 server.js)

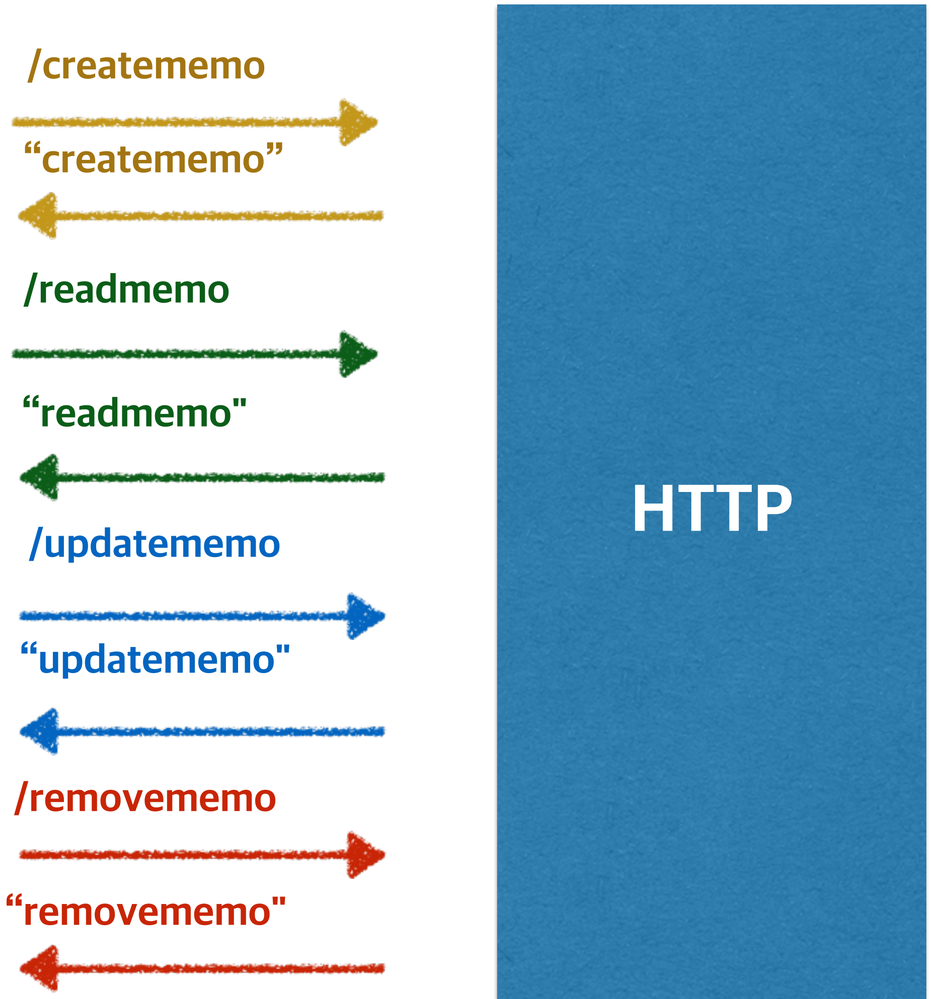
```
var http = require('http');

function onRequest(req, res) {
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.write('hello http server!!');
    res.end();
}

var server = http.createServer(onRequest);
server.listen(8080);

브라우저에 옆 링크 입력 ->  http://localhost:8080
```

Node.js 실습#2
(기본 server.js 동작)



Node.js 실습#2

(기본 server.js - 라우터 추가)



Node.js 실습#2

(기본 server.js - 라우터 추가 1)

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

function onRequest(req, res) {
    var pathname = url.parse(req.url).pathname;

    switch( pathname ) {
        case '/creatememo' :
            res.writeHead(200, {"Content-Type": "text/plain"});
            res.write('creatememo');
            res.end();
            break;
        case '/readmemo' :
            res.writeHead(200, {"Content-Type": "text/plain"});
            res.write('readmemo');
            res.end();
            break;
        case '/updatememo' :
            res.writeHead(200, {"Content-Type": "text/plain"});
            res.write('updatememo');
            res.end();
            break;
        case '/removememo' :
```

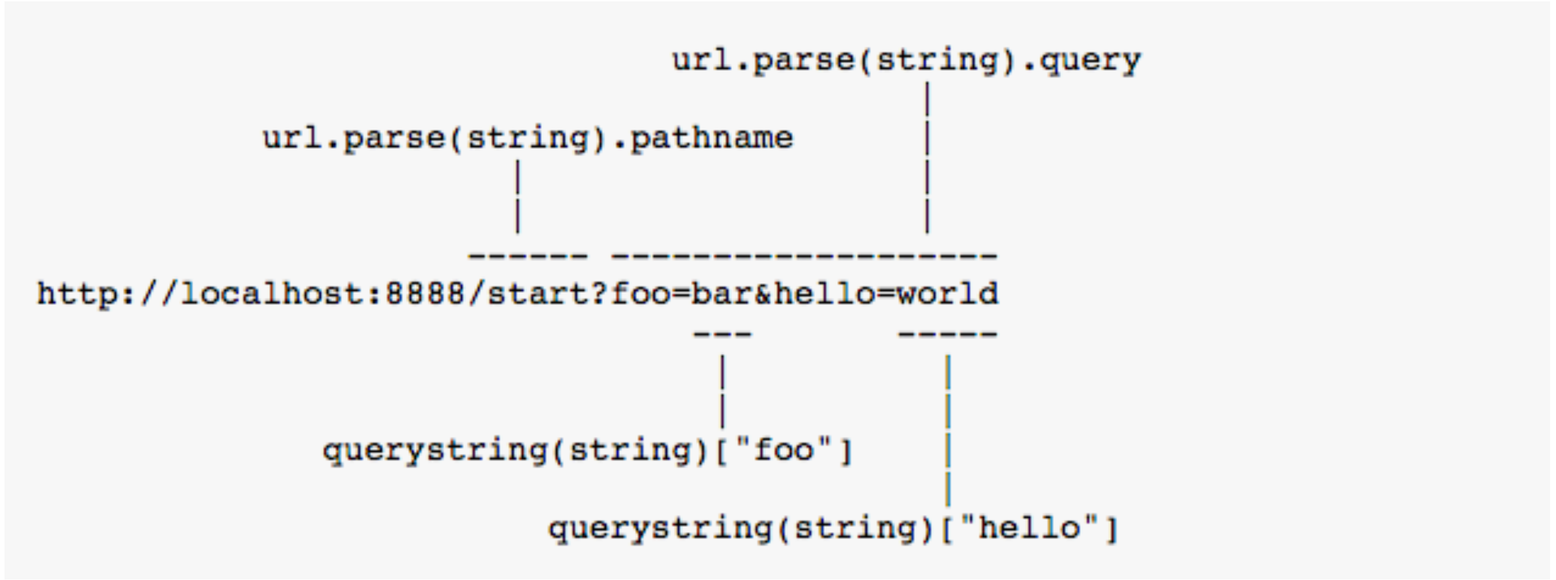
Node.js 실습#2
(기본 server.js - 라우터 추가 1)

```
    case '/removememo' :
            res.writeHead(200, {"Content-Type": "text/plain"});
            res.write('removememo');
            res.end();
            break;
    default :
            res.writeHead(404, {"Content-Type": "text/plain"});
            res.write('pathname error');
            res.end();
            break;
    }
}


var server = http.createServer(onRequest);
server.listen(8080);
```

Node.js 실습#2
(기본 server.js - 라우터 추가 1)

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

function onRequest(req, res) {
    var pathname = url.parse(req.url).pathname;

    …

}


var server = http.createServer(onRequest);
server.listen(3000);
```

Node.js 실습#2
(기본 server.js - 라우터 추가 1)

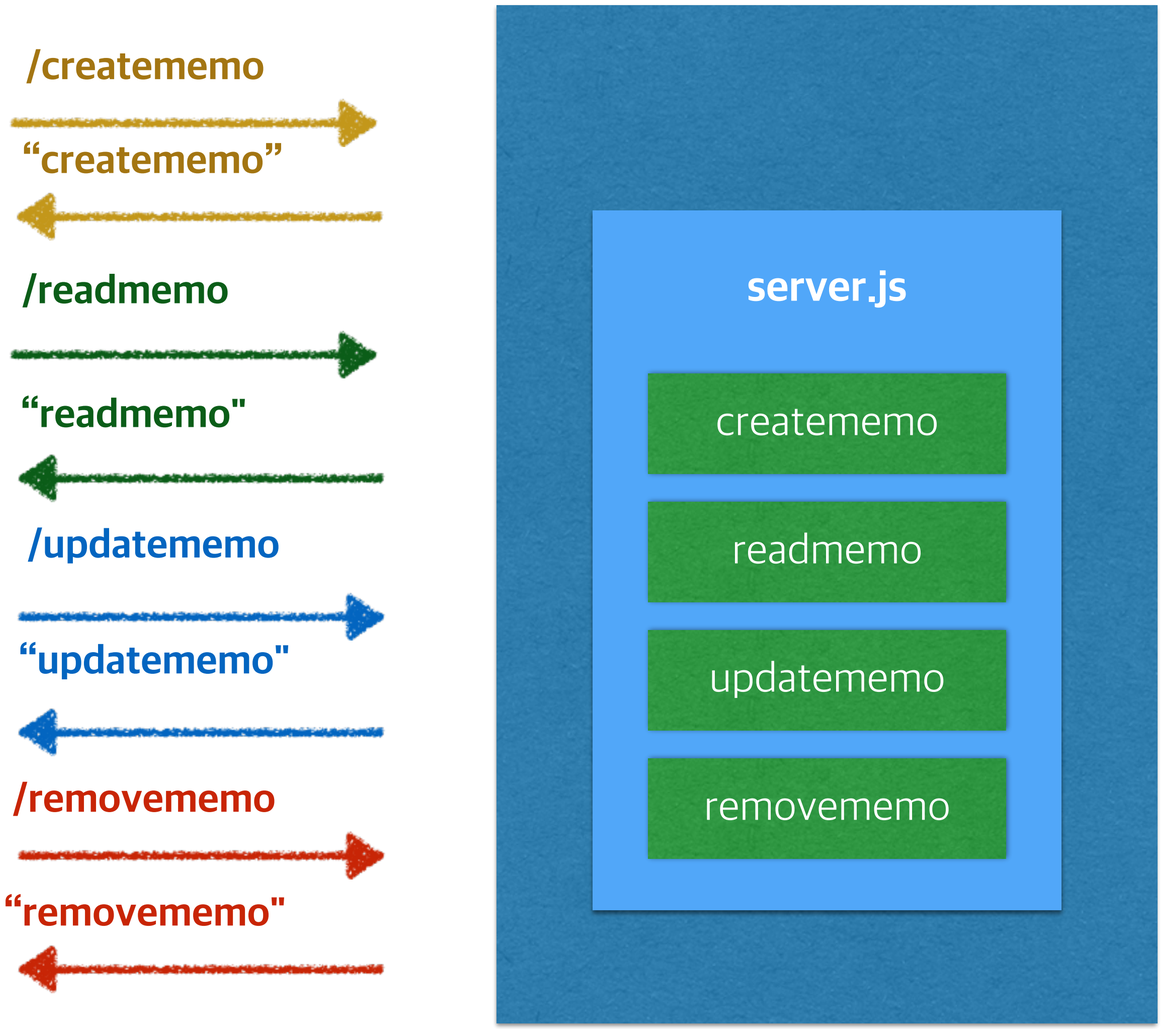


URL
+ href + protocol + slahes + host + auth + hostname + port + pathname + search + path + query + hash
+ parse() + format() + resolve()

http://nodejs.org/api/url.html

Node.js 실습#2

(기본 server.js - 라우터 추가)

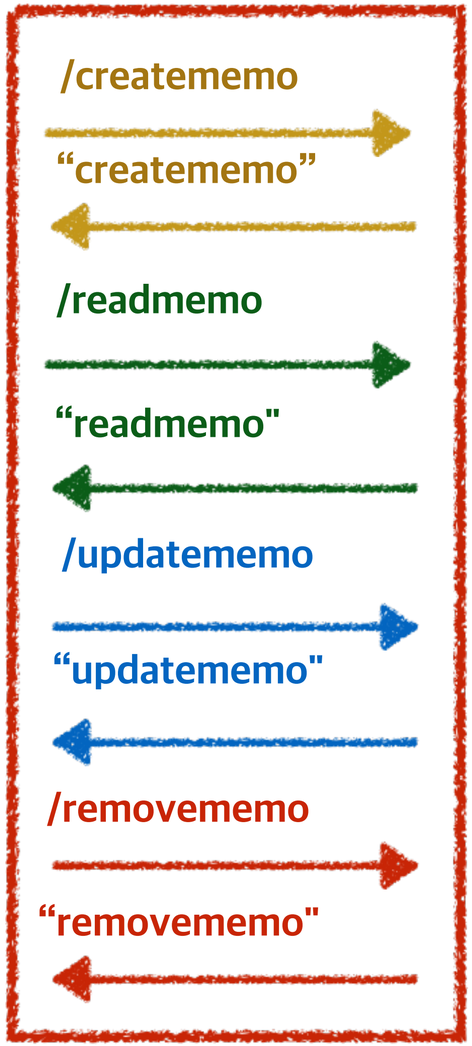


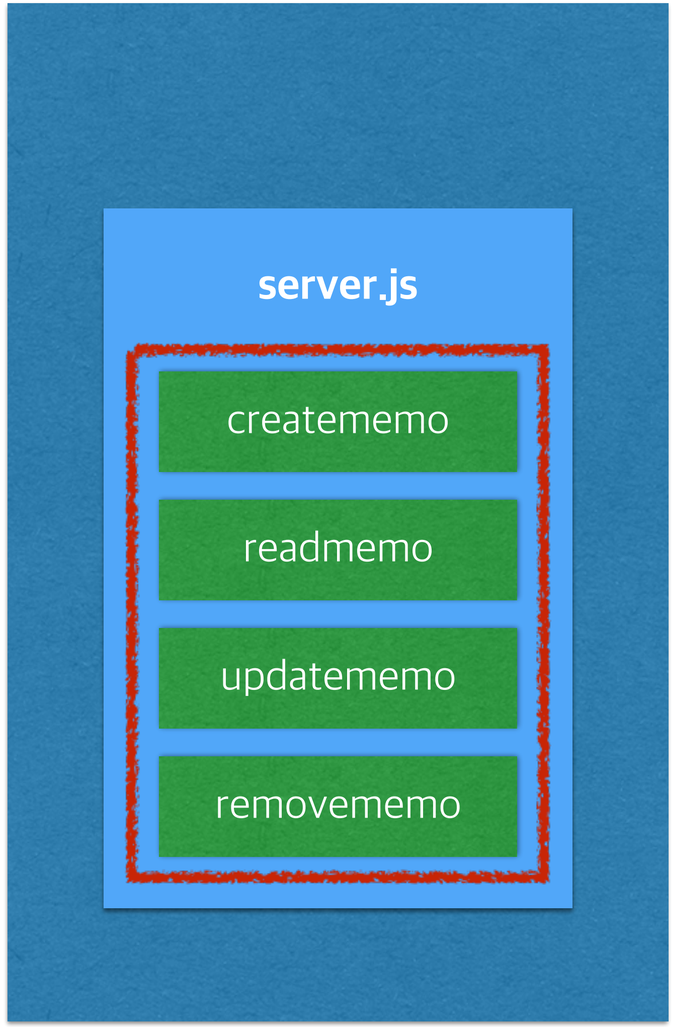
Node.js 실습#2

(기본 server.js - 라우터 추가)

RESTful

- POST
- GET
- DELETE
- PUT





Node.js 실습#2

(기본 server.js - 라우터 추가)

POST /memo

“creatememo”

GET /memo

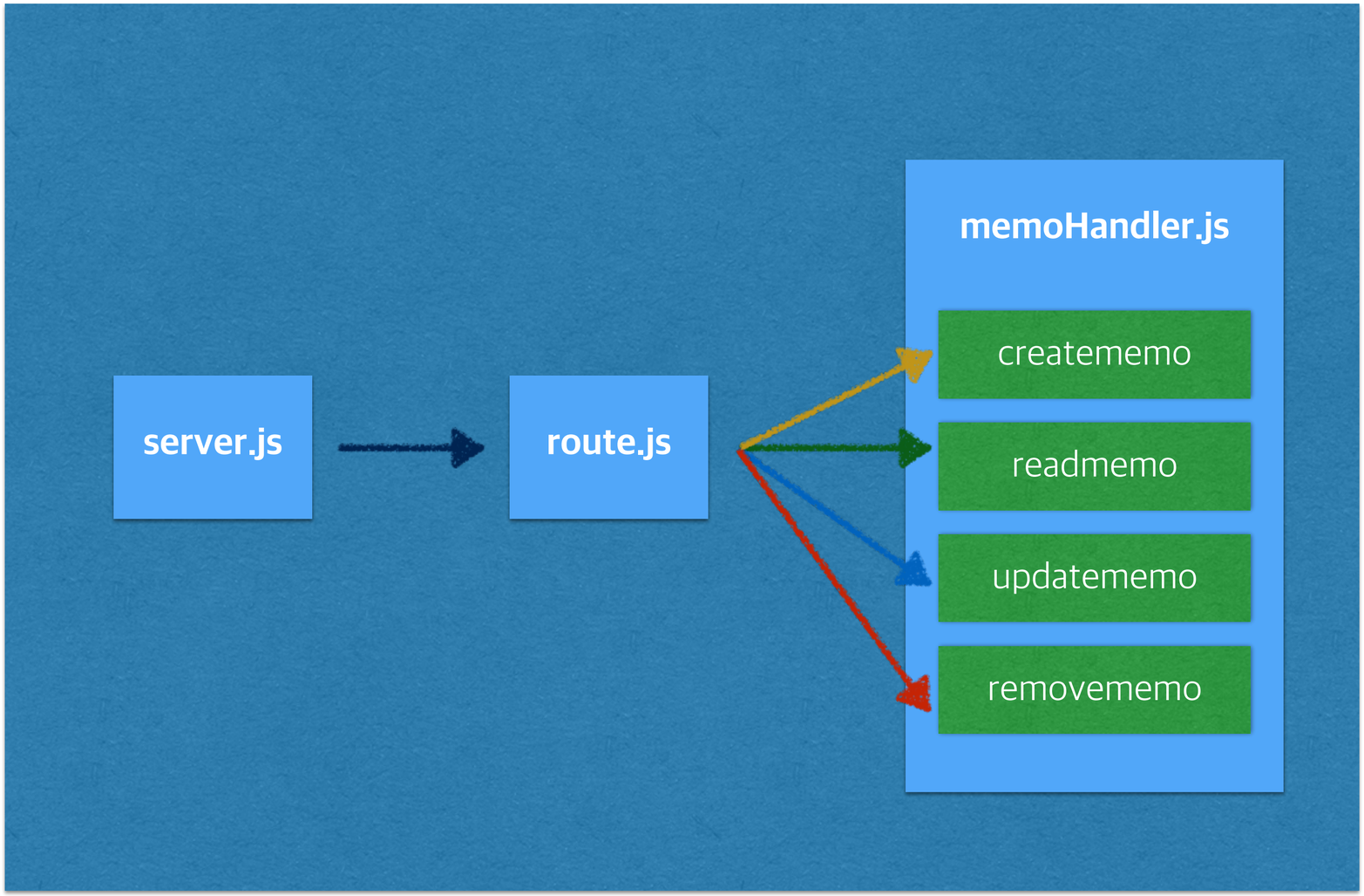
“readmemo"

PUT /memo

“updatememo"

DELETE /memo

“removememo"



Node.js 실습#2

(memoHandler.js)

```
/** memoHandler.js **/

exports.create = function(req, res, body) {
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.write('creatememo');
    res.end();
};

exports.read = function(req, res) {
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.write('readmemo');
    res.end();
};

exports.update = function(req, res, body) {
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.write('updatememo');
    res.end();
};

exports.remove = function(req, res, body) {
    res.writeHead(200, {"Content-Type": "text/plain"});
    res.write('removememo');
    res.end();
};
```

Node.js 실습#2
(route.js)

```
/** route.js **/
var memoHandler = require('./memoHandler.js');
var url = require('url');

exports.route = (function() {

    var handlers = {};

    handlers['/memo'] = {
        POST: memoHandler.create,
        GET: memoHandler.read,
        PUT: memoHandler.update,
        DELETE: memoHandler.remove
    };

    function route(req, res, body) {
        var pathname = url.parse(req.url).pathname;
        var method = req.method.toUpperCase();

        if(typeof handlers[pathname][method] === 'function') {
            handlers[pathname][method](req, res, body);
        } else {
            res.writeHead(404, {"Content-Type": "text/plain"});
            res.write('pathname error');
            res.end();
        }
    }

    return route;
})();
```

```
{ '/memo':
  {
    POST: [Function],
    GET: [Function],
    PUT: [Function],
    DELETE: [Function]
  }
 }
```

Node.js 실습#2
(server.js)

```
var http = require('http');
var route = require('./route.js');


function onRequest(req, res) {
    var body = '';

    req.on('data', function(chunk) {
        body += chunk;
    });

    req.on('end', function() {
        route.route(req, res, body);
    });
}


var server = http.createServer(onRequest);
server.listen(8080);
```

Node.js 실습#2

(기본 server.js - 라우터 추가)

POST /memo

“creatememo”

GET /memo

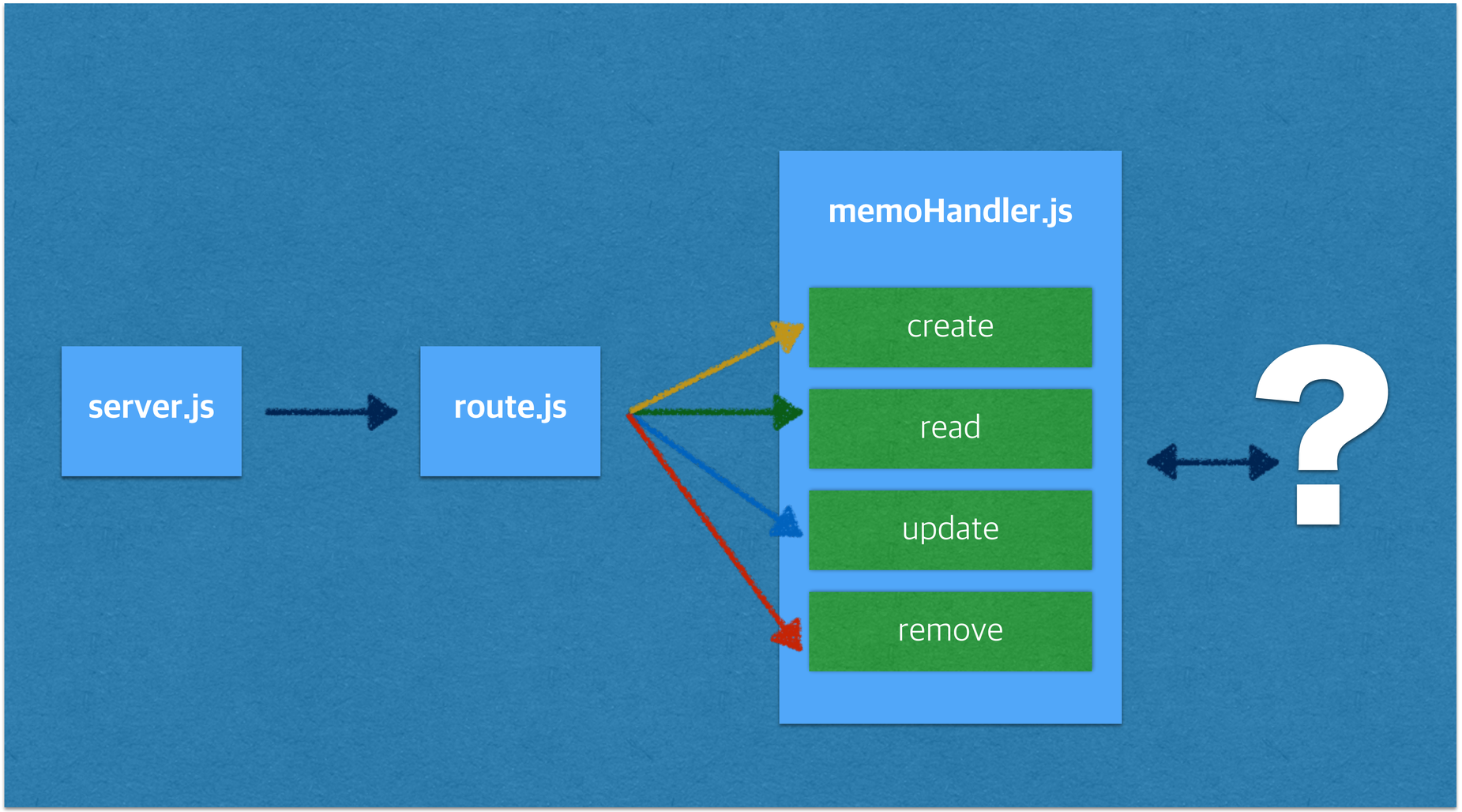
“readmemo"

PUT /memo

“updatememo"

DELETE /memo

“removememo"



Node.js 실습#2
(Database 추가)

https://github.com/louischatriot/nedb

README.md

NeDB (Node embedded database)

Embedded persistent database for Node.js, written in Javascript, with no dependency (except npm modules of course). You can **think of it as a SQLite for Node.js projects**, which can be used with a simple `require` statement. The API is a subset of MongoDB's. You can use it as a persistent or an in-memory only datastore.

NeDB is not intended to be a replacement of large-scale databases such as MongoDB! Its goal is to provide you with a clean and easy way to query data and persist it to disk, for web applications that do not need lots of concurrent connections, for example a continuous integration and deployment server and desktop applications built with Node Webkit.

NeDB was benchmarked against the popular client-side database TaffyDB and NeDB is much, much faster. That's why there is now a browser version, which can also provide persistence.

Check the change log in the wiki if you think nedb doesn't behave as the documentation describes! Most of the issues I get are due to non-latest version NeDBs.

- Document DB
- Written in Javascript
- 설치가 간단
- Mongodb API와 비슷

Node.js 실습#2
(Database - nedb 저장 포맷)

```
{"author":"pcw","memo":"test","date":{"$$date":1421153113134},"_id":"nxi5jUVW2AZe1Hd9"}
{"author":"pcw","memo":"test","date":{"$$date":1421153130192},"_id":"CmOjzsOHwwHgMPo0"}
{"author":"pcw","memo":"test","date":{"$$date":1421153130455},"_id":"JQMJDc7LcKyb3sn5"}
{"author":"pcw","memo":"test","date":{"$$date":1421153130625},"_id":"ijy4xW0sUJfa9GMt"}
{"author":"pcw","memo":"test","date":{"$$date":1421153130832},"_id":"MIGjwM6sUgIoV1YI"}
{"author":"pcw","memo":"test","date":{"$$date":1421153131282},"_id":"nttf0FVySpcM8b5N"}
{"author":"pcw","memo":"test","date":{"$$date":1421153131563},"_id":"oMetRNzbGmgkV8cE"}
```

Node.js 실습#2

(Database 추가)

Installation, tests

Module name on npm is `nedb`.

```
npm install nedb --save   // Put latest version in your package.json

npm test   // You'll need the dev dependencies to test it
```

Node.js 실습#2
(memoHandler.js)

```
/** memoHandler.js **/

var Datastore = require('nedb');
var db = new Datastore({ filename: './data/memo', autoload: true });
var querystring = require('querystring');
var url = require('url');


exports.create = function(req, res, body) {
    _insertMemo( body , function(error, result) {
        res.writeHead(200, {"Content-Type": "application/json"});
        res.write( '{ "result": "creatememo"}');
        res.end();
    });
};

exports.read = function(req, res) {
    _findMemo({}, function(error, results) {
        res.writeHead(200, {"Content-Type": "application/json"});
        res.write(JSON.stringify(results));
        res.end();
    });
};

exports.update = function(req, res, body) {
    var query = url.parse(req.url).query;
    var where = querystring.parse(query);

    _updateMemo(where, body, function(error, results) {
        res.writeHead(200, {"Content-Type": "application/json"});
        res.write('updatememo');
        res.end();
    });
};
```

Node.js 실습#2 (memoHandler.js)

```
exports.remove = function(req, res, body) {
    var query = url.parse(req.url).query;
    var where = querystring.parse(query);

    _removeMemo(where, function(error, results) {
        res.writeHead(200, {"Content-Type": "application/json"});
        res.write( '{"result": "removememo"}');
        res.end();
    });
};

function _insertMemo(body , callback) {
    body = typeof body === 'string' ? JSON.parse(body) : body;

    var memo = {
        author: body.author,
        memo : body.memo,
        date: new Date()
    };
    db.insert(memo, callback);
}

function _findMemo(where, callback) {
    where = where || {};
    db.find(where, callback);
}

function _updateMemo(where, body, callback) {
    body = typeof body === 'string' ? JSON.parse(body) : body;
    db.update(where, {$set: body}, {multi: true}, callback);
}

function _removeMemo(where, callback) {
    db.remove(where, {multi: true}, callback);
}
```

Node.js 실습#2

(기본 server.js - 라우터 추가)

POST /memo

“creatememo”

GET /memo

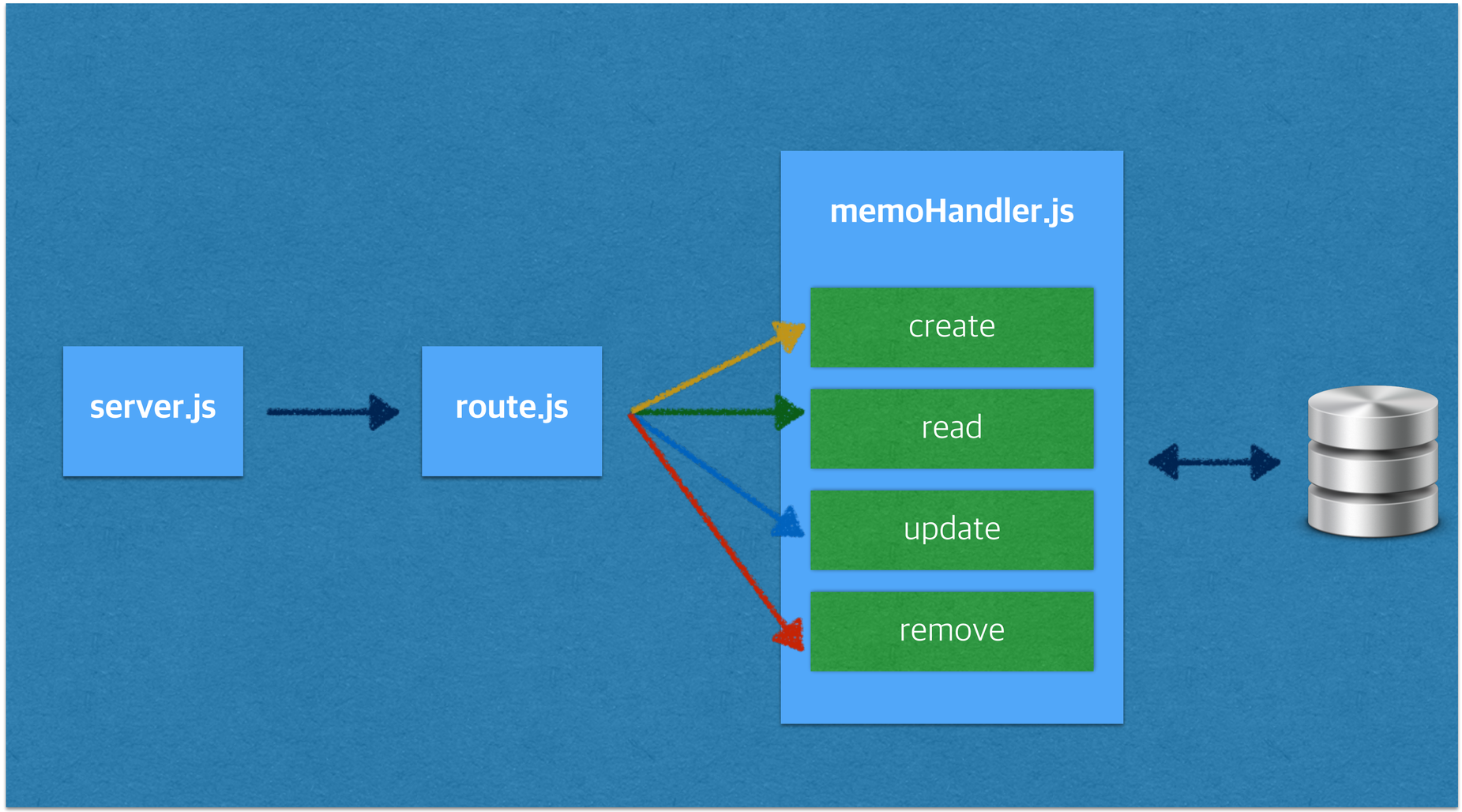
“readmemo"

UPDATE /memo

“updatememo"

DELETE /memo

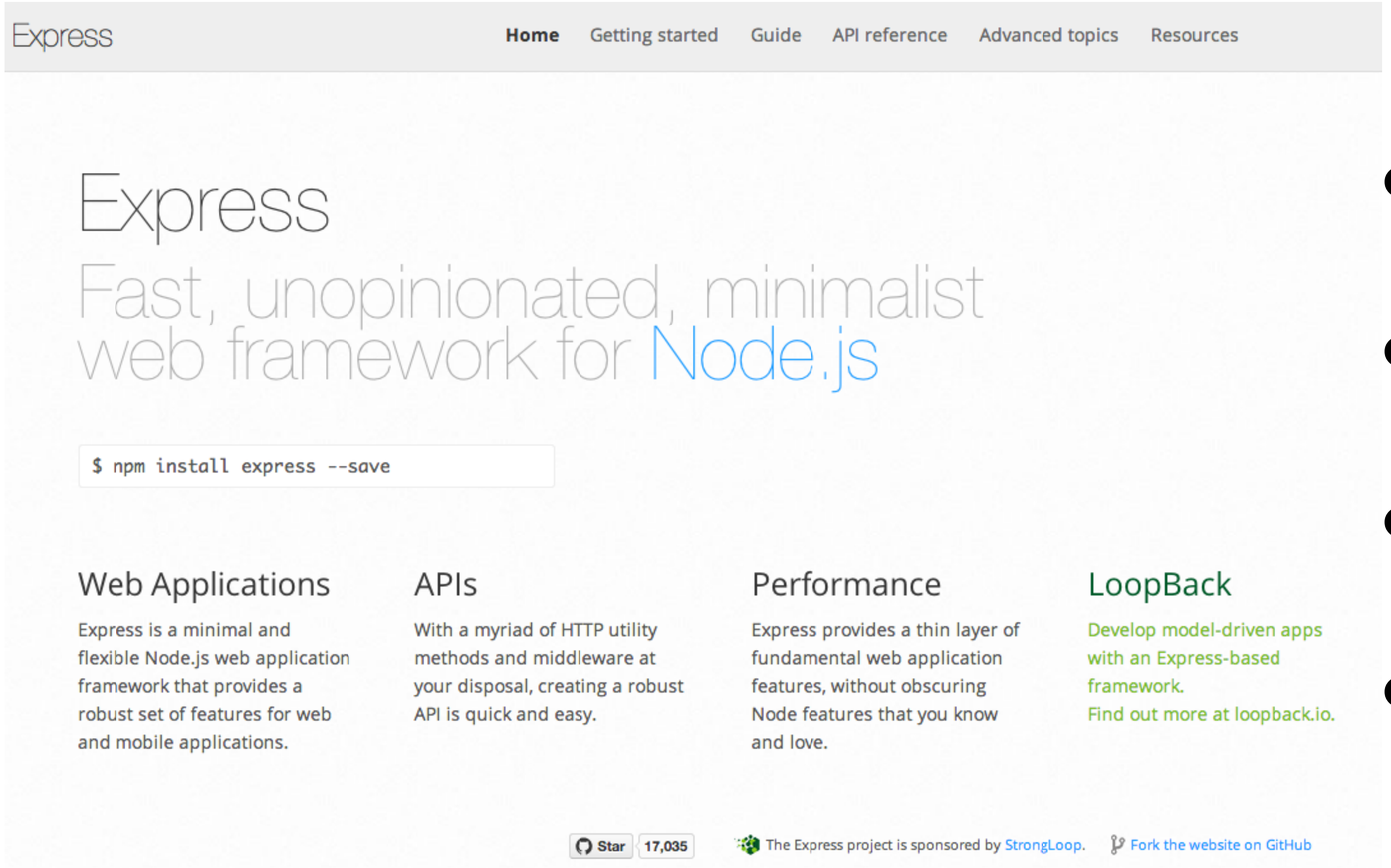
“removememo"



Node.js 실습#3

(express)

Node.js 실습#3
(express)



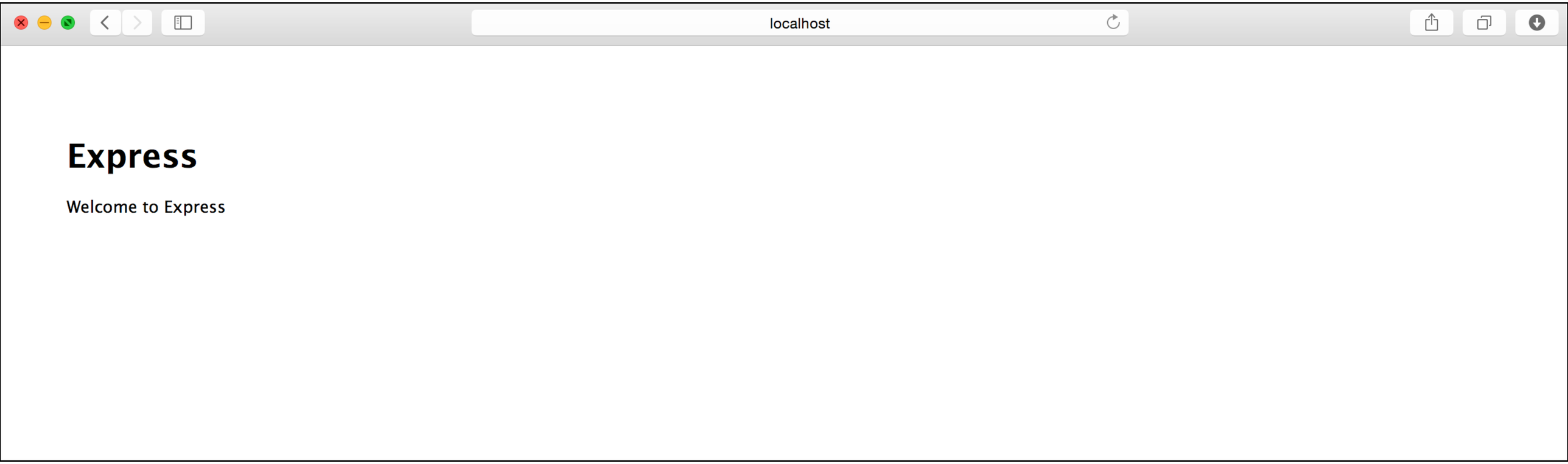
- Web Framework
- Parsing HTTP request
- Managing Session
- Organizing Routes

Node.js 실습#3
(express - 설치)

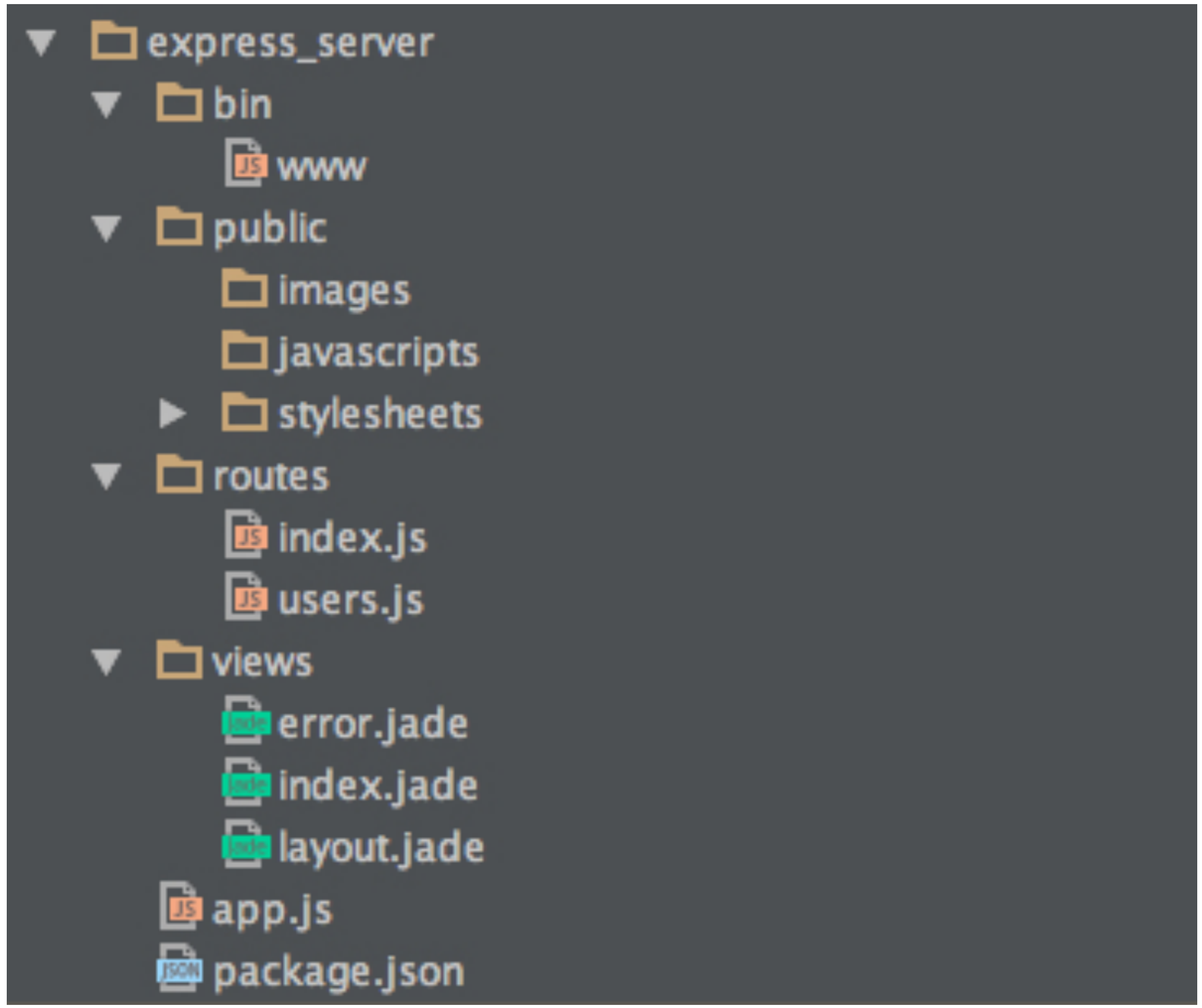
```
$ npm install -g express-generator
$ express express_memoserver
$ cd express_memoserver
$ npm install
$ npm start
```

Node.js 실습#3
(express - 설치)

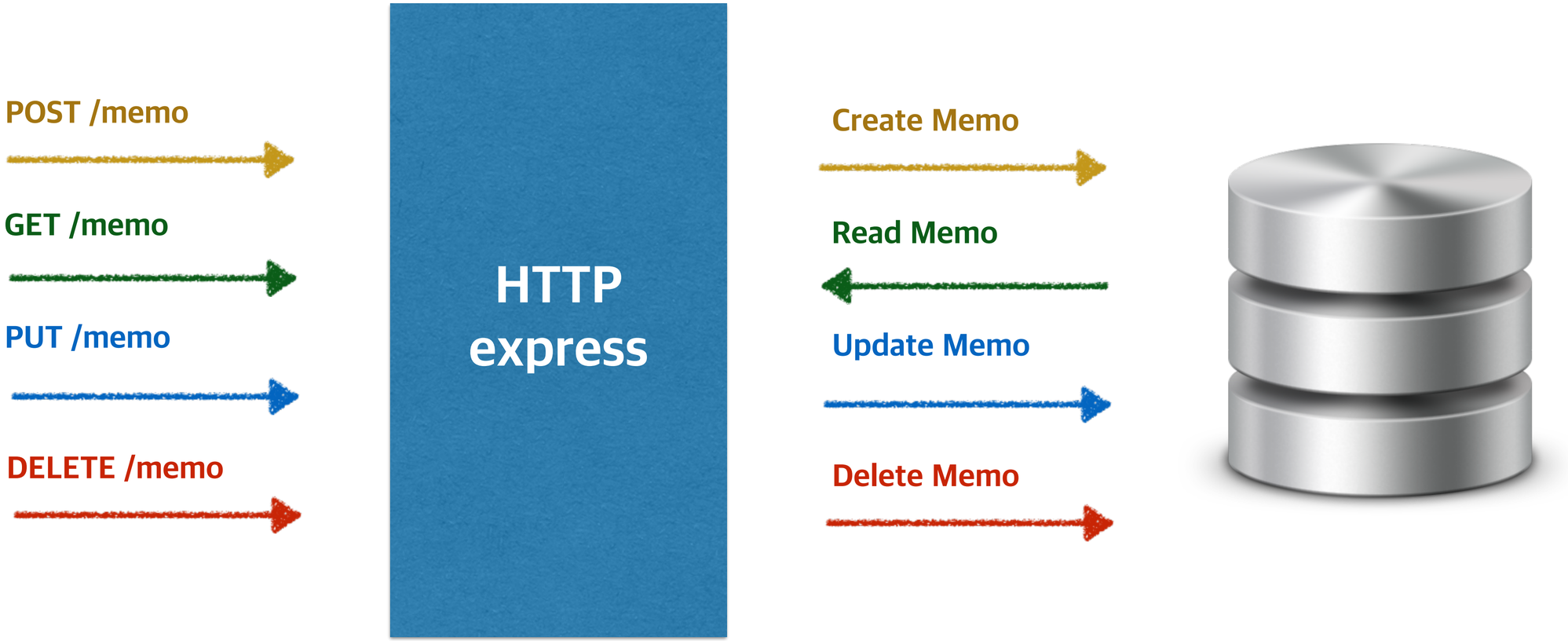
localhost:3000



Node.js 실습#3
(express - 디렉토리 구조)

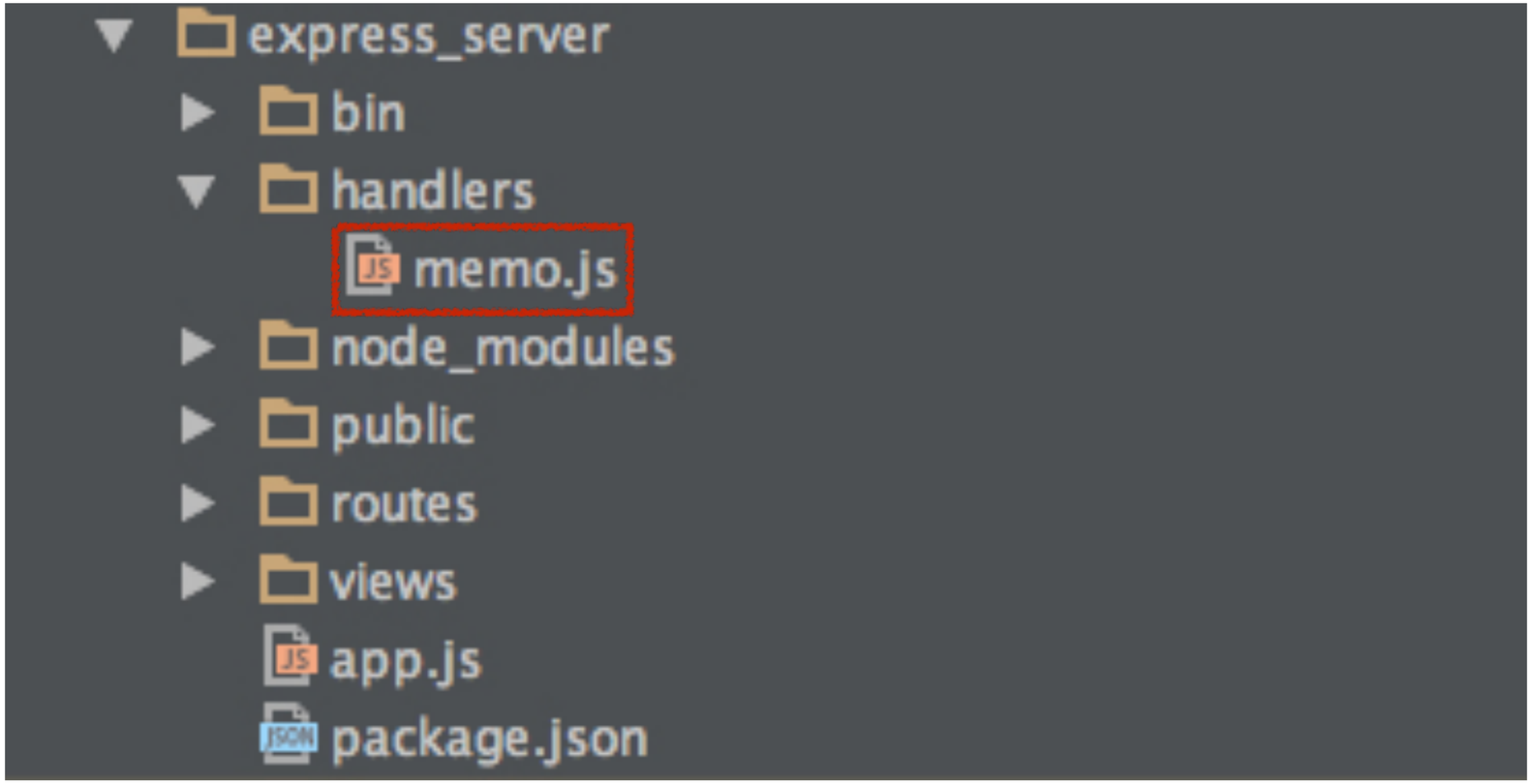


Node.js 실습#3
(express - memoserver)



Node.js 실습#3

(express - handlers/memo.js 추가)



Node.js 실습#3

(express- handlers/memo.js(1/2))

```
/** memo.js **/
var Datastore = require('nedb');
var db = new Datastore({ filename: './data/memo', autoload: true });

exports.create = function(req, res) {
    var body = req.body;

    _insertMemo( body , function(error, result) {
        res.json( {error: error, result: result});
    });
};

exports.read = function(req, res) {
    _findMemo({}, function(error, results) {
        res.json( {error: error, results: results});
    });
};

exports.update = function(req, res) {
    var where = req.query;
    var body = req.body;

    _updateMemo(where, body, function(error, results) {
        res.json( {error: error, results: results});
    });
};

exports.remove = function(req, res) {
    var where = req.query;

    _removeMemo(where, function(error, results) {
        res.json( {error: error, results: results});
    });
};

function _insertMemo(body , callback) {
```

Node.js 실습#3

(express- handlers/memo.js(2/2))

```
function _insertMemo(body , callback) {
    body = typeof body === 'string' ? JSON.parse(body) : body;

    var memo = {
        author: body.author,
        memo : body.memo,
        date: new Date()
    };

    db.insert(memo, callback);
}

function _findMemo(where, callback) {
    where = where || {};
    db.find(where, callback);
}

function _updateMemo(where, body, callback) {
    db.update(where, {$set: body}, {multi: true}, callback);
}

function _removeMemo(where, callback) {
    db.remove(where, {multi: true}, callback);
}
```

Node.js 실습#3

(express - routes/memo.js 추가)

Node.js 실습#3

(express - route/memo.js)

```
var express = require('express');
var router = express.Router();
var memo = require('../handlers/memo');

router.post('/', memo.create);

router.get('/', memo.read);

router.put('/', memo.update);

router.delete('/', memo.remove);

module.exports = router;
```

Node.js 실습#3

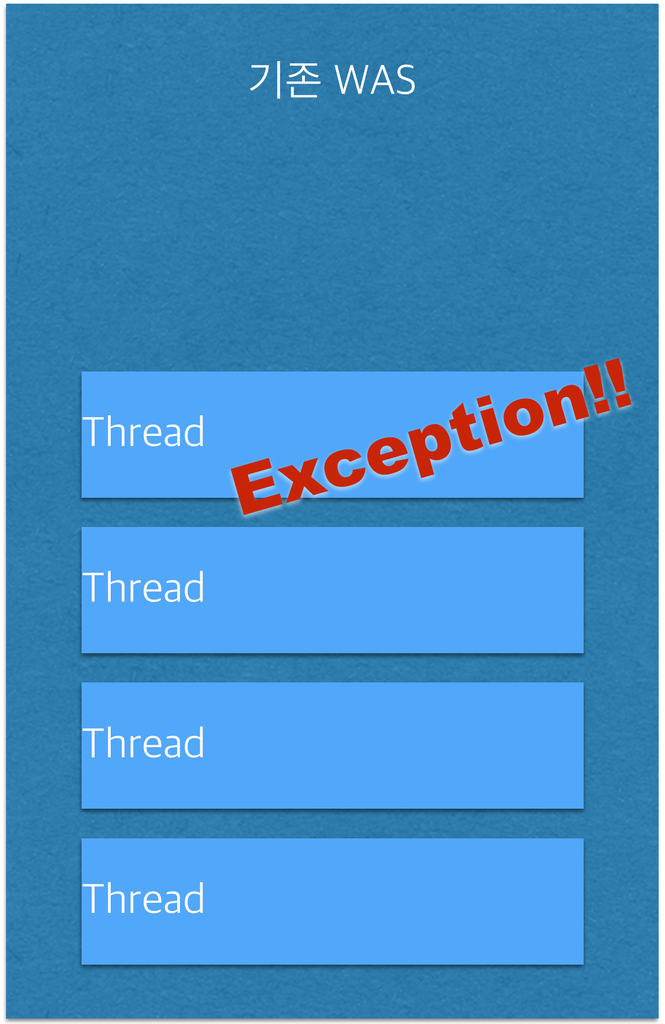
(express - app.js)

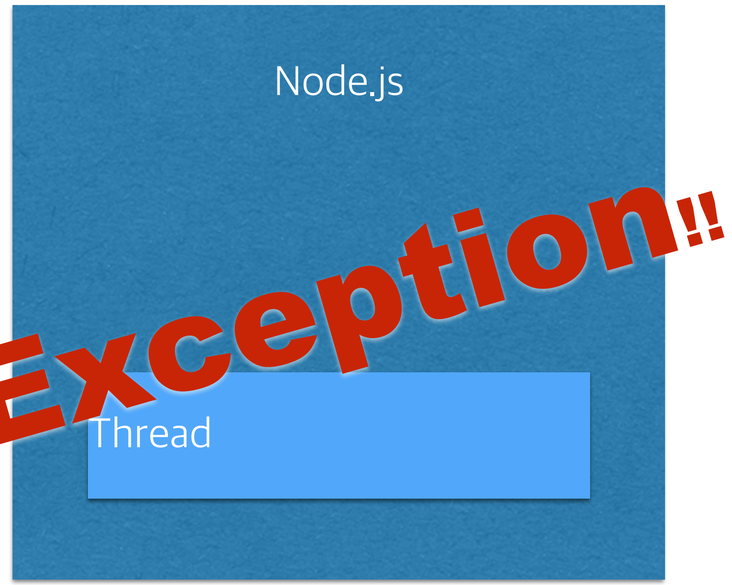
```
var express = require('express');
var path = require('path');
var favicon = require('static-favicon');
var logger = require('morgan');
var cookieParser = require('cookie-parser');
var bodyParser = require('body-parser');

var routes = require('./routes/index');
var users = require('./routes/users');
var memo = require('./routes/memo');


var app = express();

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'jade');

app.use(favicon());
app.use(logger('dev'));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded());
app.use(cookieParser());
app.use(express.static(path.join(__dirname, 'public')));

app.use('/', routes);
app.use('/users', users);
app.use('/memo', memo);


/// catch 404 and forward to error handler
```

Part 3.
Node.JS Production
Leady Go!

- Error handling
- Multi process
- Callback hell
- Logging
- Unit test

Node.js Production Ready
(Error handling)





Node.js Production Ready
(Error handling)

Error handling

- try, catch
- uncaughtException
- forever / pm2

Node.js Production Ready
(Error handling - uncaughtException)

```
function makeException() {
    throw new Error('Custom Exception!!');
}

setTimeout(function() {
    console.log('Hello');
    setTimeout(function(){
        console.log('Wold');
    }, 1000);
}, 1000);

makeException();

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

Node.js Production Ready
(Error handling - uncaughtException)

```
process.on('uncaughtException', function(error) {
    console.log('Error ' + error.stack);
});

function makeException() {
    throw new Error('Custom Exception!!');
}

…
```

Node.js Production Ready
(Error handling - uncaughtException)

```
process.on('uncaughtException', function(error) {
    console.log('Error ' + error.stack);
    process.exit(1);
});


function makeException() {
    throw new Error('Custom Exception!!');
}

…
```

Node.js Production Ready
(Error handling - try,catch)

```
function makeException() {
    throw new Error('Custom Exception!!');
}

setTimeout(function() {
    console.log('Hello');
    setTimeout(function(){
        console.log('Wold');
    }, 1000);
}, 1000);

try{
    makeException();
}catch(err) {
    console.log('try-catch ' + err.stack);
}

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

Node.js Production Ready
(Error handling - Domain)

Domain은 여러개의 다른 IO 연산들을 하나의 그룹으로 처리하는 방법을 제공.
-> 에러가 나면 한 군데 모아서 처리할수 있습니다. .

uncaughtException 핸들러가 에러의 Context 를 잃어버려 제대로 정보를 안 남겨주거나, error code를 남기고 바로 죽는 프로그램들은 Domain을 사용하는 것이 좋습니다.

Express는 좀더 쓰기 쉬운 Domain-Middleware를 사용하시면 callback의 번거러움에 해방될 수 있습니다.

Node.js Production Ready
(Error handling - Domain) - 변경할 것

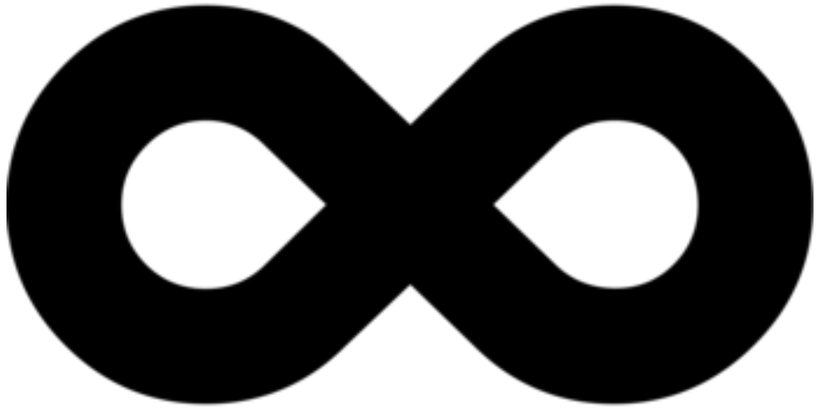
```
var domain = require('domain');
var d = domain.create();
var fs = require('fs');

d.on('error', function(err) {
  console.error(err);
});

d.run(function() {
  fs.readFile('somefile.txt', function (err, data) {
    if (err) throw err;
    console.log(data);
  });
});
```

Node.js Production Ready
(Error handling)

그래도 Node.js는 죽습니다...

그럼 그 다음은??

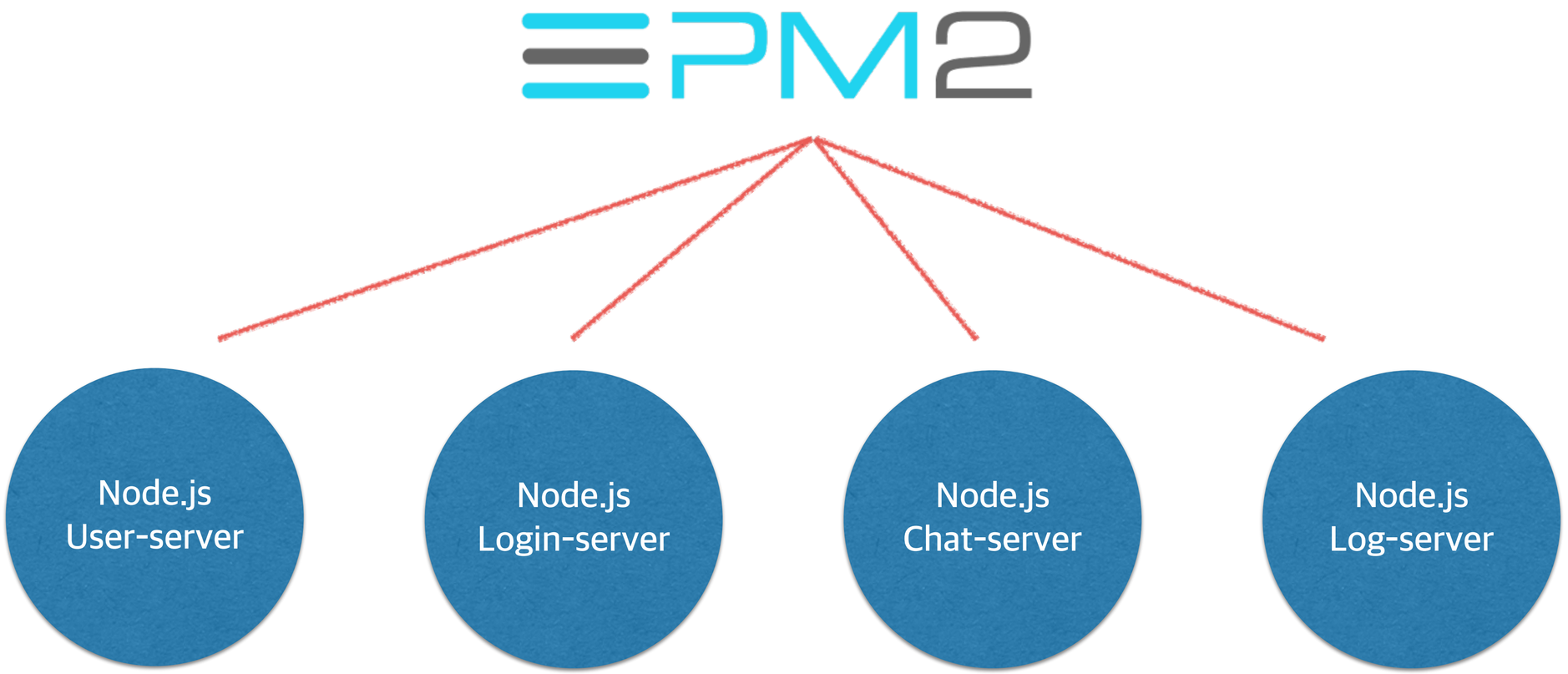
Node.js Production Ready
(Error handling)



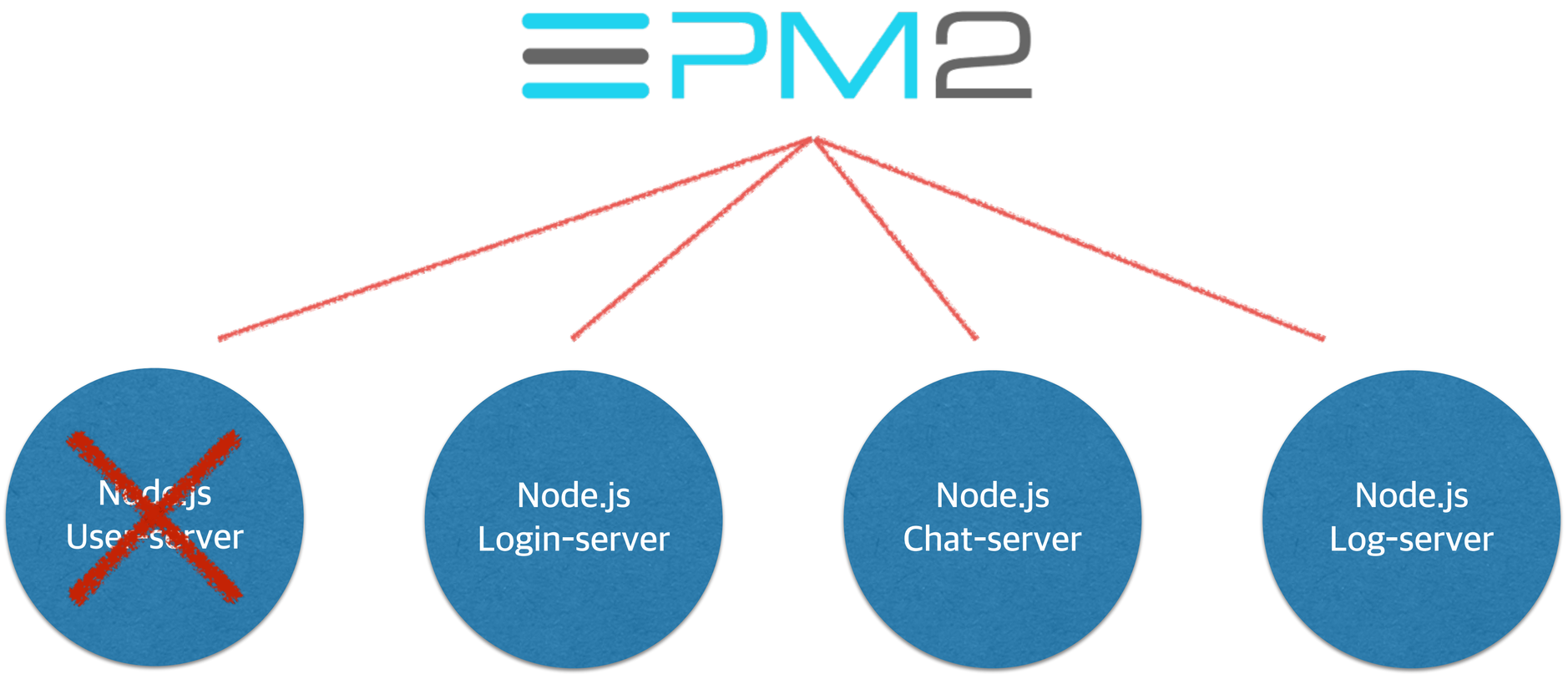
Node.js Production Ready
(Error handling)

PM2 is a production process manager for Node.js applications with a built-in load balancer. It allows you to keep applications alive forever, to reload them without downtime and to facilitate common system admin tasks.

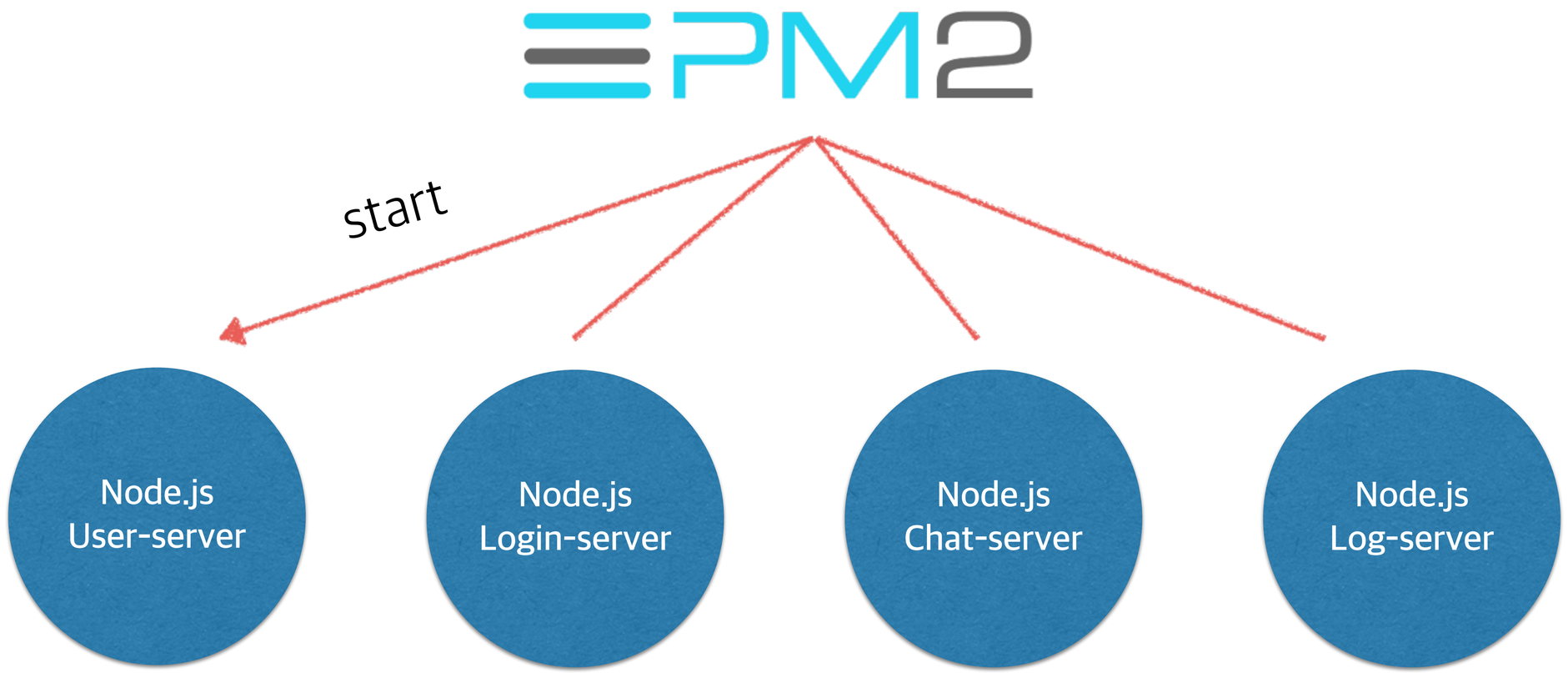
Node.js Production Ready
(Error handling)



Node.js Production Ready
(Error handling)



Node.js Production Ready
(Error handling)



Node.js Production Ready
(Error handling)

```
[tknew:~/Unitech/pm2] master(+84/-121)+* ± pm2 list
PM2  Process listing
```

App Name	id	mode	PID	status	Restarted	Uptime	memory	err logs
bashscript.sh	6	fork	8278	online	0	10s	1.379 MB	/home/tknew/.pm2/logs/bashscript.sh-err.log
checker	5	cluster	0	stopped	0	2m	0 B	/home/tknew/.pm2/logs/checker-err.log
interface-api	3	cluster	7526	online	0	3m	15.445 MB	/home/tknew/.pm2/logs/interface-api-err.log
interface-api	2	cluster	7517	online	0	3m	15.453 MB	/home/tknew/.pm2/logs/interface-api-err.log
interface-api	1	cluster	7512	online	0	3m	15.449 MB	/home/tknew/.pm2/logs/interface-api-err.log
interface-api	0	cluster	7507	online	0	3m	15.449 MB	/home/tknew/.pm2/logs/interface-api-err.log

```
Editing file on Transport
[echokill out (153)] log message from echo auto kill
[echo out (147)] log message from echo.js
[echokill out (154)] log message from echo auto kill
[echo err (1245)] err msg from echo.js
[echo out (148)] log message from echo.js
[echokill out (155)] log message from echo auto kill
[echo out (149)] log message from echo.js
[echokill err (115)] error message, killing my self
[PM2 DAEMON (13)] ["2013-05-29T13:39:43.407Z","Script /home/tknew/Unitech/pm2/examples/echokill.js 166 exited code 255\n"]
[PM2 DAEMON (14)] ["2013-05-29T13:39:43.447Z","/home/tknew/Unitech/pm2/examples/echokill.js - id167 worker online\n"]
[echo err (1246)] err msg from echo.js
[echo out (150)] log message from echo.js
[echokill out (156)] log message from echo auto kill
[echo out (151)] log message from echo.js
[echokill out (157)] log message from echo auto kill
[echo err (1247)] err msg from echo.js
[echo out (152)] log message from echo.js
[echokill out (158)] log message from echo auto kill
```

```
PM2 monitoring :

 ● web-ui                [                    ] 0 %
[5] [cluster_mode]       [                    ] 0 B

 ● check-online          [                    ] 0 %
[4] [fork_mode]          [|||||               ] 1.258 MB

 ● interface-api         [|||||||||||||||||||| ] 73 %
[3] [cluster_mode]       [|||||||||||         ] 35.223 MB

 ● interface-api         [|||||||||||||||||||| ] 78 %
[2] [cluster_mode]       [|||||||||||         ] 35.418 MB

 ● interface-api         [|||||||||||||||||||| ] 90 %
[1] [cluster_mode]       [|||||||||||         ] 35.137 MB

 ● interface-api         [|||||||||||||||||   ] 63 %
[0] [cluster_mode]       [|||||||||||         ] 35.137 MB
```

Node.js Production Ready
(Error handling)

Install PM2

```
$ npm install pm2 -g
```

npm is a builtin CLI when you install Node.js - Installing Node.js with NVM

Node.js 0.11.14 is recommended for cluster mode and reload feature.

Start an application

```
$ pm2 start app.js
$ pm2 start app.js -i max  # Enable load-balancer and cluster features
```

Node.js Production Ready
(Error handling)

```
syntaxfishui-MacBook-Pro:nodejsstudyexample syntaxfish$ pm2 start server.js
[PM2] Process server.js launched
```

App name	id	mode	PID	status	restarted	uptime	memory	watching
server	2	fork	5350	online	0	0s	40.250 MB	disabled

```
 Use `pm2 info <id|name>` to get more details about an app
syntaxfishui-MacBook-Pro:nodejsstudyexample syntaxfish$ kill -9 5350
syntaxfishui-MacBook-Pro:nodejsstudyexample syntaxfish$ pm2 list
```

App name	id	mode	PID	status	restarted	uptime	memory	watching
server	2	fork	5353	online	1	4s	156.613 MB	disabled

Node.js Production Ready
(Multi process)

부하는 늘어나는데...

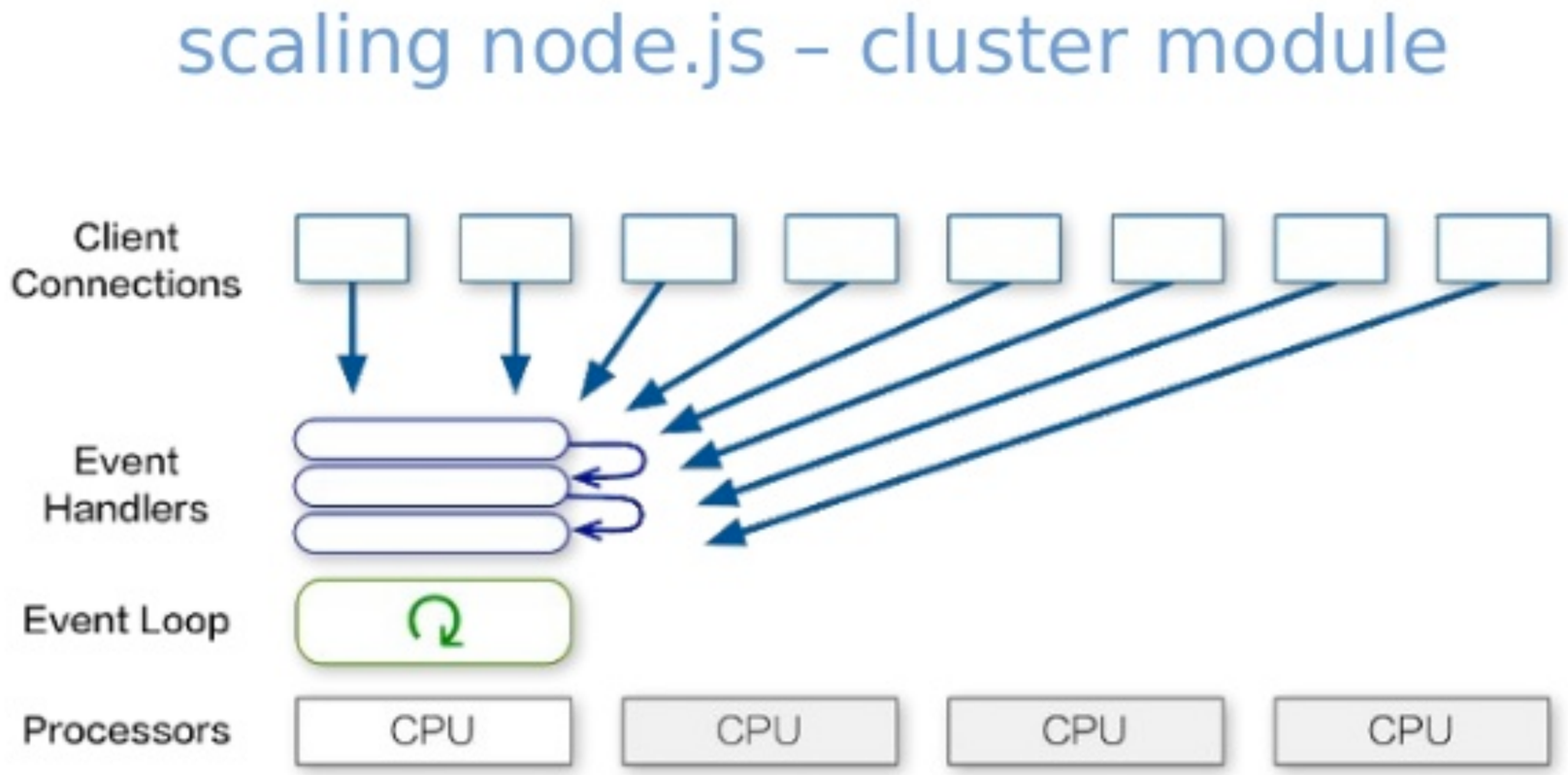
Thread를 늘릴수도 없고...

Node.js Production Ready
(Multi process)

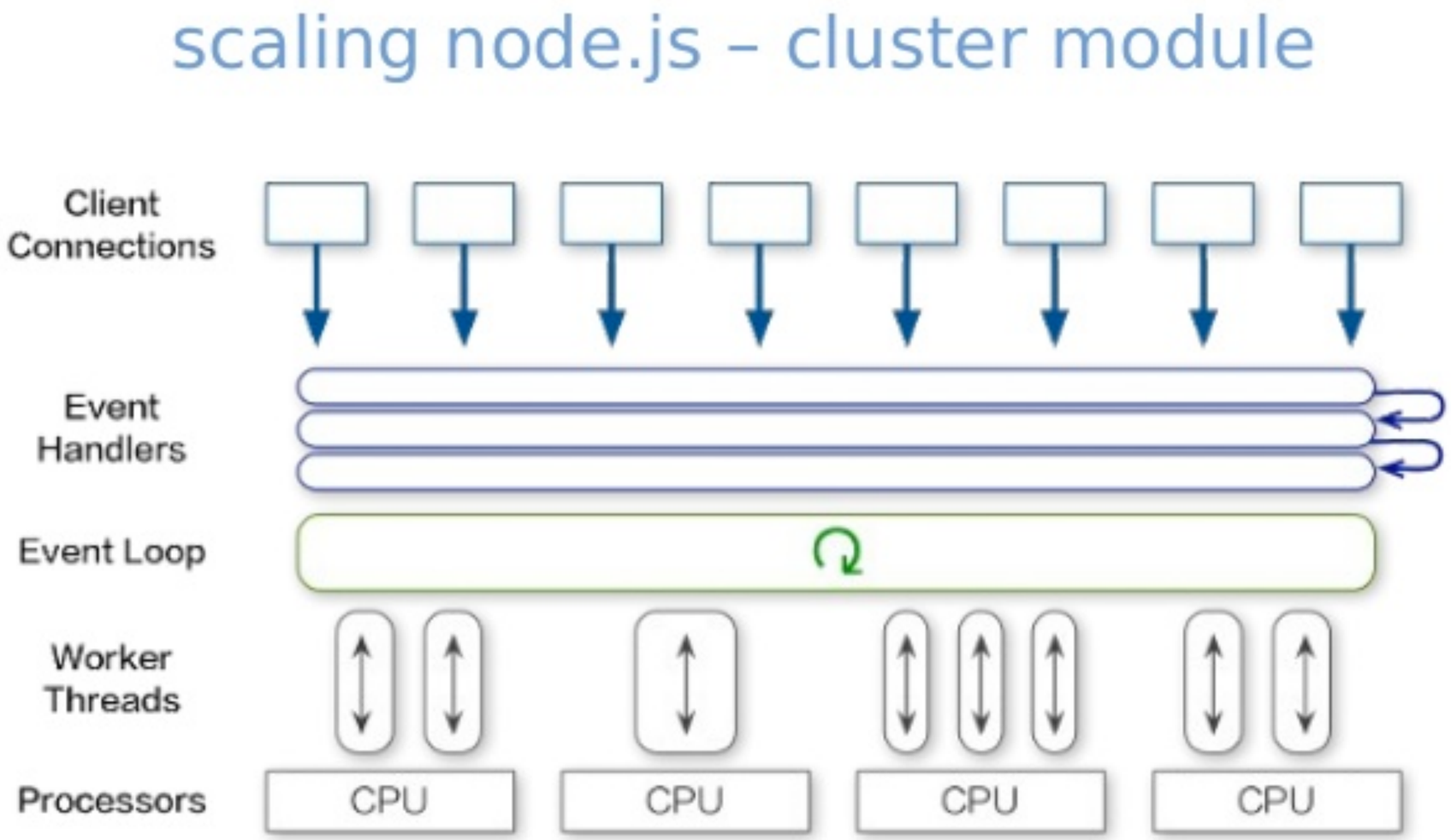
Multi process

- cluster

Node.js Production Ready
(Multi process)



Node.js Production Ready
(Multi process)



Node.js Production Ready
(Multi process)

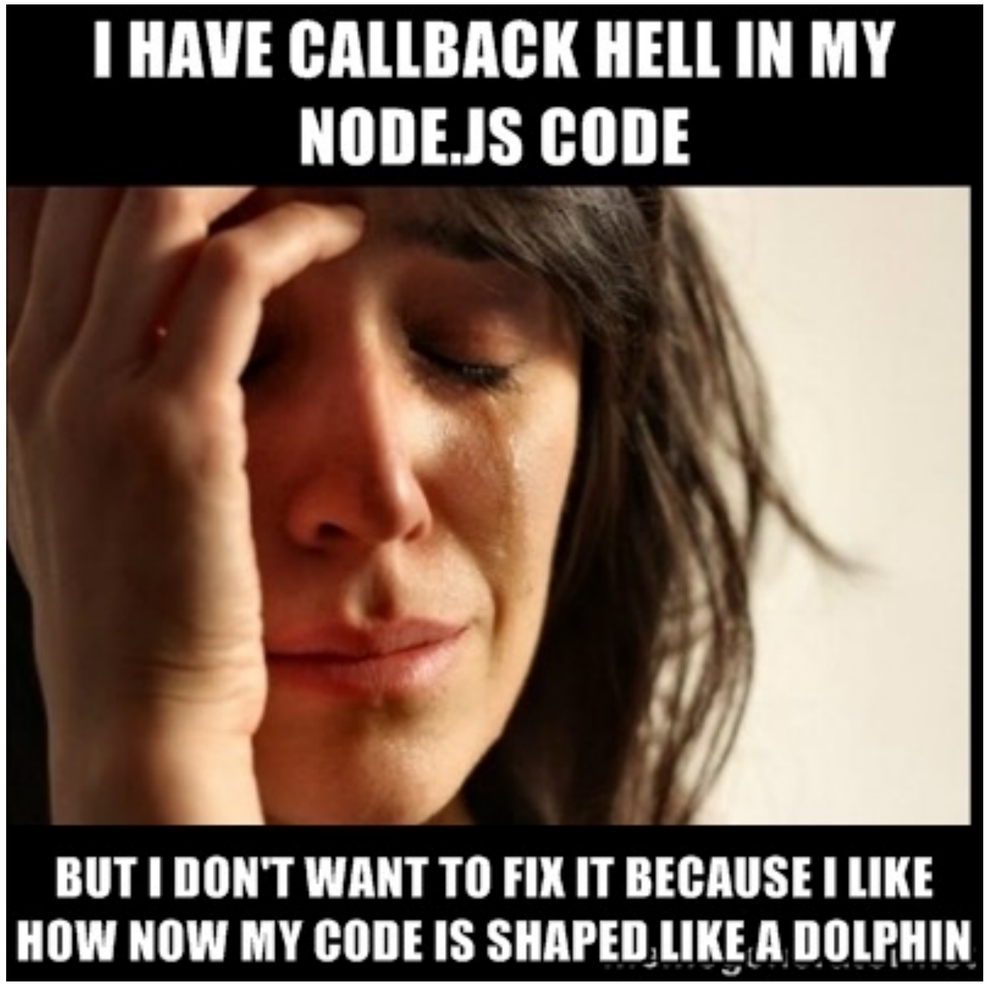
```
var http = require('http');
var cluster = require('cluster');

if( cluster.isMaster ) {
    var cpus = require('os').cpus().length;

    for( var i = 0; i < cpus; i++ ) {
        cluster.fork();
    }
} else {
    function onRequest(req, res) {
        res.writeHead({'Content-type': 'text/plain'});
        res.end('Process on ' + process.pid);

        for( var i = 0; i < 100000; i++ ) {
            console.log('Dummy job');
        }
```

Node.js Production Ready
(Multi process)

```
  };
    var server = http.createServer(onRequest);
    server.listen(8000);
}
```

Node.js Production Ready
(Callback hell)

```
app.get('/some_resources', function (req, res) {
  db.query('SELECT A ...', function (err, a) {
    if (err) return res.end(err);

    db.query('SELECT B ... WHERE a=' + a, function (err, b) {
      if (err) return res.end(err);

      db.query('SELECT C ... WHERE b=' + b, function (err, c) {
        if (err) return res.end(err);

        db.query('SELECT D ... WHERE c=' + c, function (err, d) {
          if (err) return res.end(err);

          res.end(d);
        });
      });
    });
  });
});
```



Node.js Production Ready
(Callback hell)

Callback hell

- async
- promise

Node.js Production Ready
(Callback hell - async)

Async.js

Async is a utility module which **provides straight-forward,** powerful functions for working with asynchronous JavaScript. Although originally designed for use with Node.js and installable via npm install async, it can also be used directly in the browser.

https://github.com/caolan/async

Node.js Production Ready
(Callback hell - async)

Control Flow

- series
- parallel
- parallelLimit
- whilst
- doWhilst
- until
- doUntil
- forever
- waterfall
- compose
- seq
- applyEach
- applyEachSeries
- queue

- queue
- priorityQueue
- cargo
- auto
- retry
- iterator
- apply
- nextTick
- times
- timesSeries

Node.js Production Ready
(Callback hell)

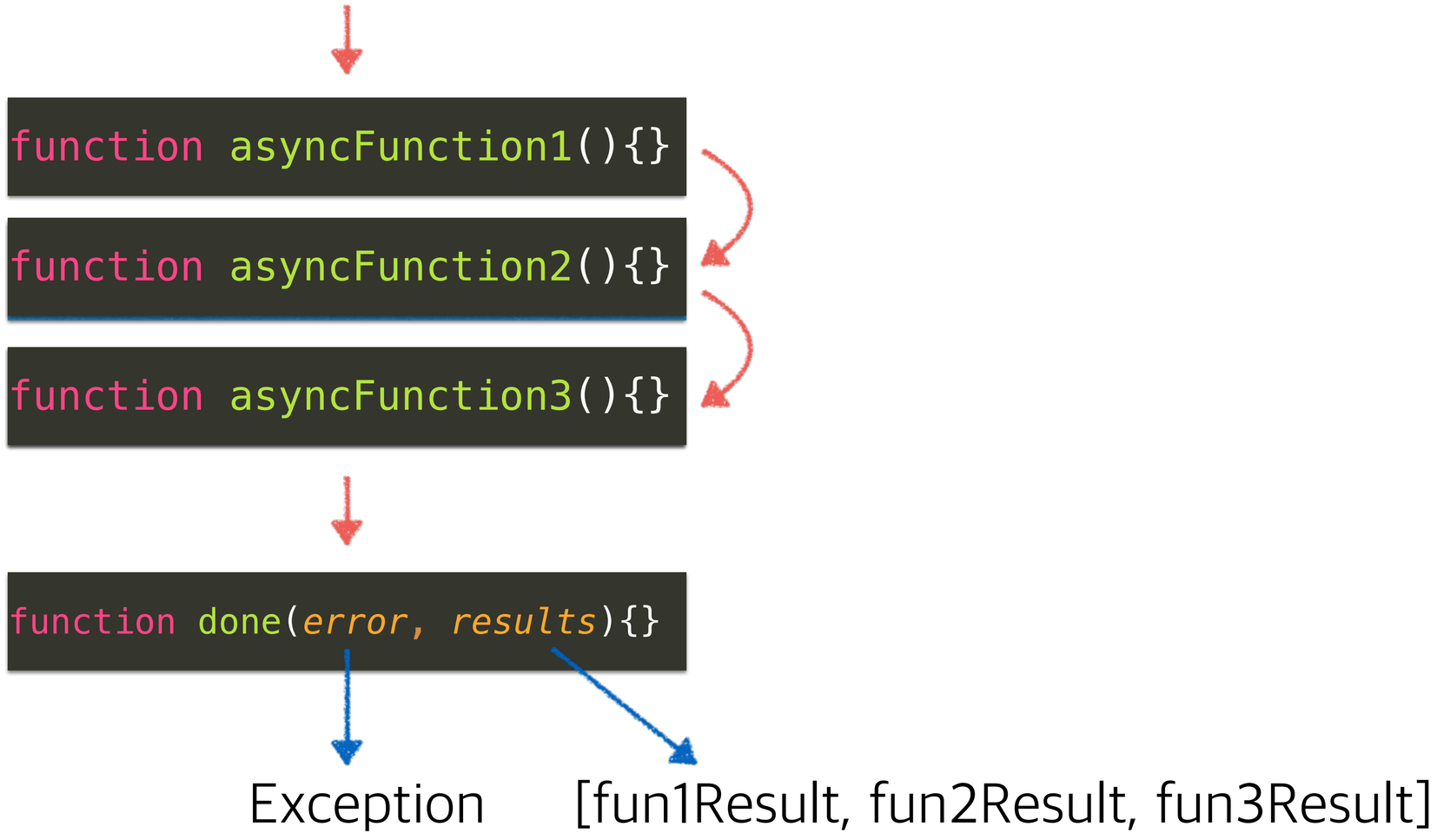
```
asyncFunction1(function(error, results) {
    asyncFunction2(function(error, results) {
        asyncFunction3(function(error, results) {
            done(error, results);
        });
    });
});
```

Node.js Production Ready
(Callback hell - async.series)

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

async.series([
    function asyncFunction1(cb) {
        setTimeout(function(){
            console.log('asyncFunction1');
            cb(null, 'asyncFunction1');
        }, 1000);
    },
    function asyncFunction2(cb) {
        console.log('asyncFunction2');
        cb(null, 'asyncFunction2');
    },
    function asyncFunction3(cb) {
        console.log('asyncFunction3');
        cb(null, 'asyncFunction3');
    }
], function done(error, results) {
    console.log('error: ', error);
    console.log('results: ', results);
});
```

Node.js Production Ready

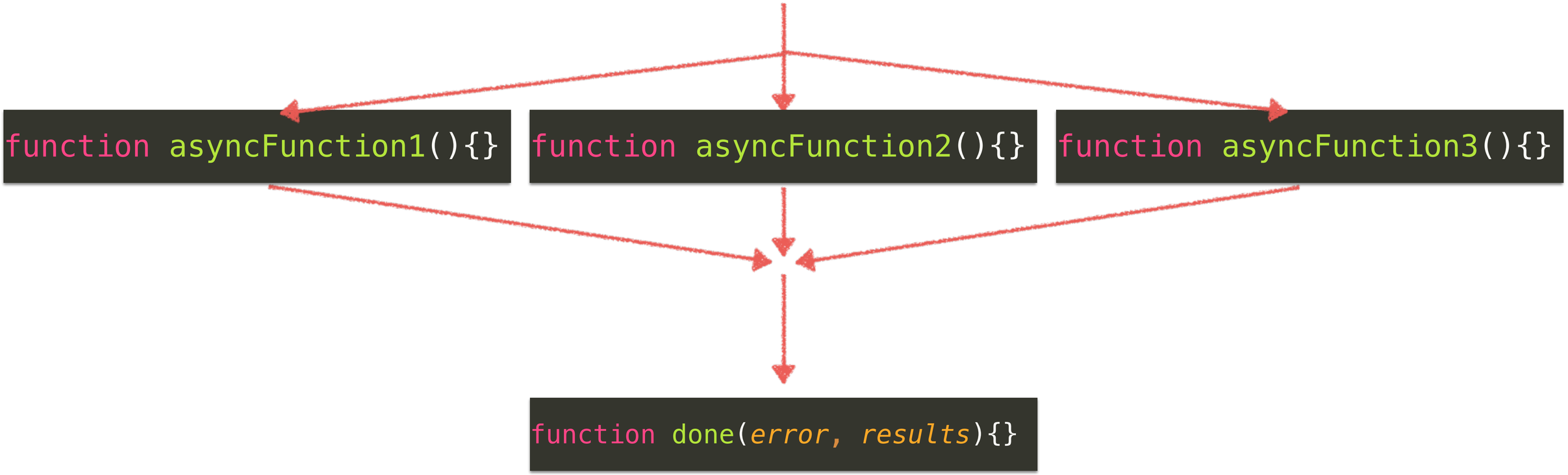
(Callback hell - async.series)



Node.js Production Ready
(Callback hell - async.parallel)

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

async.parallel([
    function asyncFunction1(cb) {
        setTimeout(function(){
            console.log('asyncFunction1');
            cb(null, 'asyncFunction1');
        }, 1000);
    },
    function asyncFunction2(cb) {
        console.log('asyncFunction2');
        cb(null, 'asyncFunction2');
    },
    function asyncFunction3(cb) {
        console.log('asyncFunction3');
        cb(null, 'asyncFunction3');
    }
], function done(error, results) {
    console.log('error: ', error);
    console.log('results: ', results);
});
```

Node.js Production Ready
(Callback hell - async.parallel)

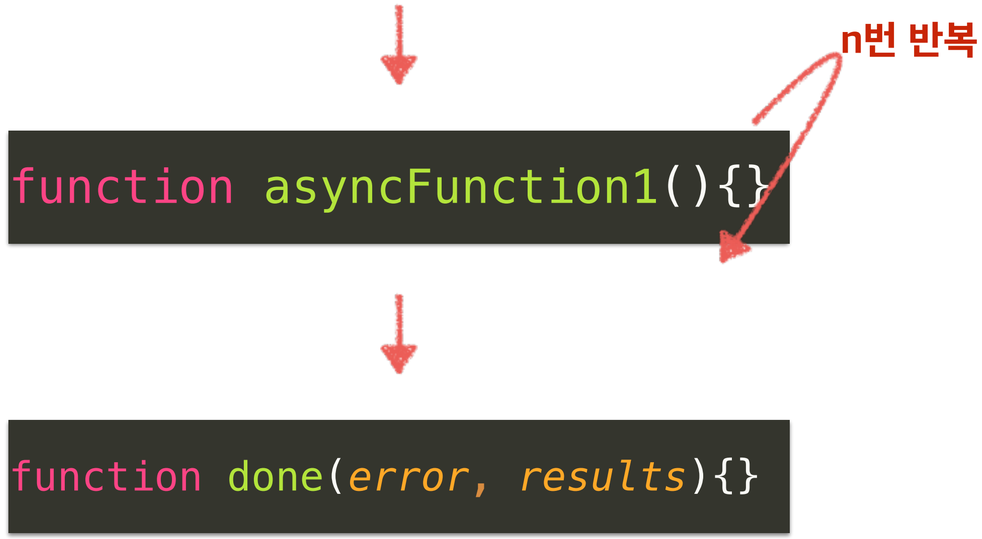


Node.js Production Ready
(Callback hell - async.times)

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

async.times(5, function asyncFunction(n, next) {
    console.log('[%d] asyncFunction', n);
    next(null, 'asyncFunction' + n);
},function done(error, results) {
    console.log('error: ', error);
    console.log('results: ', results);
});
```

Node.js Production Ready

(Callback hell - async.times)



Node.js Production Ready
(Callback hell - async)

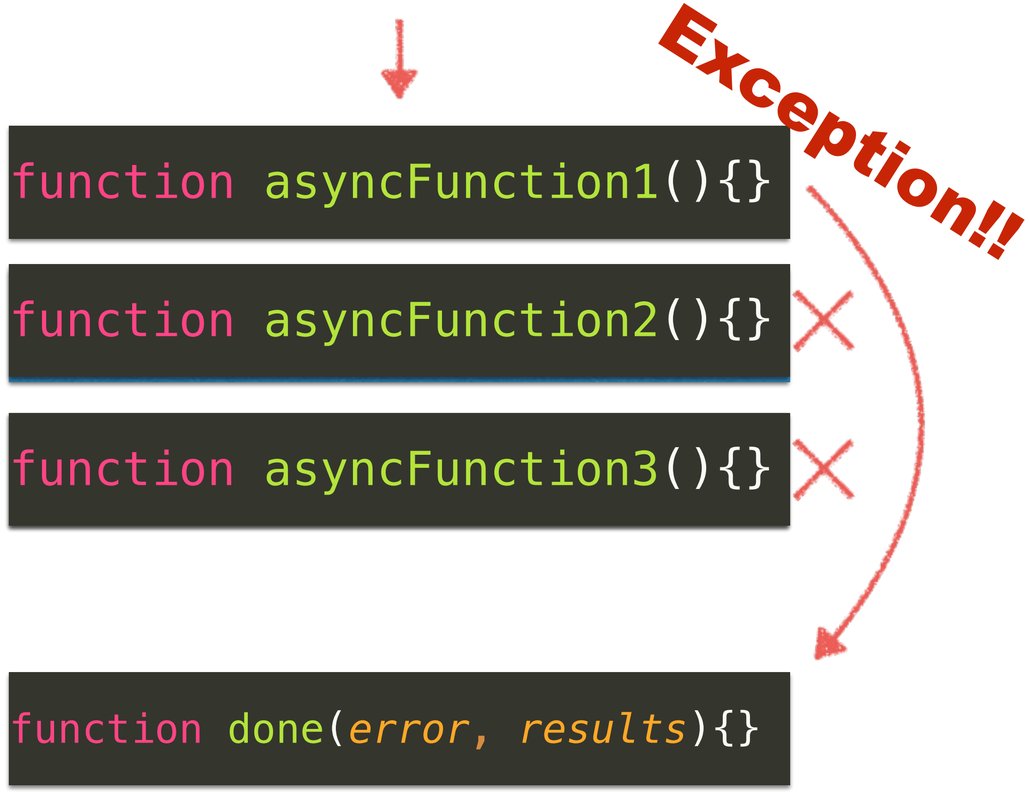
Exception이 발생하면?

Node.js Production Ready
(Callback hell - async.series)

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

async.series([
    function asyncFunction1(cb) {
        setTimeout(function(){
            console.log('asyncFunction1');
            cb(new Error('timeout'), 'asyncFunction1');
        }, 1000);
    },
    function asyncFunction2(cb) {
        console.log('asyncFunction2');
        cb(null, 'asyncFunction2');
    },
    function asyncFunction3(cb) {
        console.log('asyncFunction3');
        cb(null, 'asyncFunction3');
    }
], function done(error, results) {
    console.log('error: ', error);
    console.log('results: ', results);
});
```

Node.js Production Ready

(Callback hell - async.series)



Node.js Production Ready
(Event logging)

Event logging

- winston

Node.js Production Ready
(Event logging - winston)

winston

Winston is designed to be a **simple and universal logging library** with support for multiple transports. A transport is essentially a storage device for your logs. Each instance of a winston logger **can have multiple transports configured at different levels**. For example, one may want error logs to be stored in a persistent remote location (like a database), but all logs output to the console or a local file.

https://github.com/flatiron/winston

Node.js Production Ready
(Event logging - winston)

Installation

```
npm install winston
```

Node.js Production Ready
(Event logging - winston#1)

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

winston.log('info', 'Hello distributed log files!');
winston.info('Hello again distributed logs');
```

Node.js Production Ready
(Event logging - winston#2)

```
var winston = require('winston');
var Logger = winston.Logger;
var File  = winston.transports.File;
var Console = winston.transports.Console;


var options = {
    levels: {
        info: 0,
        normal: 1,
        minor: 2,
        critical: 3
    },
    colors: {
        info: 'green',
        normal: 'blue',
        minor: 'yellow',
        critical: 'red'
    },
```

Node.js Production Ready
(Event logging - winston#2)

```
  },
    transports: [
        new File({
            name : 'log-file',
            filename: 'log-file.log',
            level: 'info'
        }),
        new Console({
            level: 'critical',
            colorize: true
        })
    ]
};
var logger = new Logger(options);
logger.info('info!!');
logger.normal('normal!!');
logger.minor('minor!!');
logger.critical('critical!!');
```

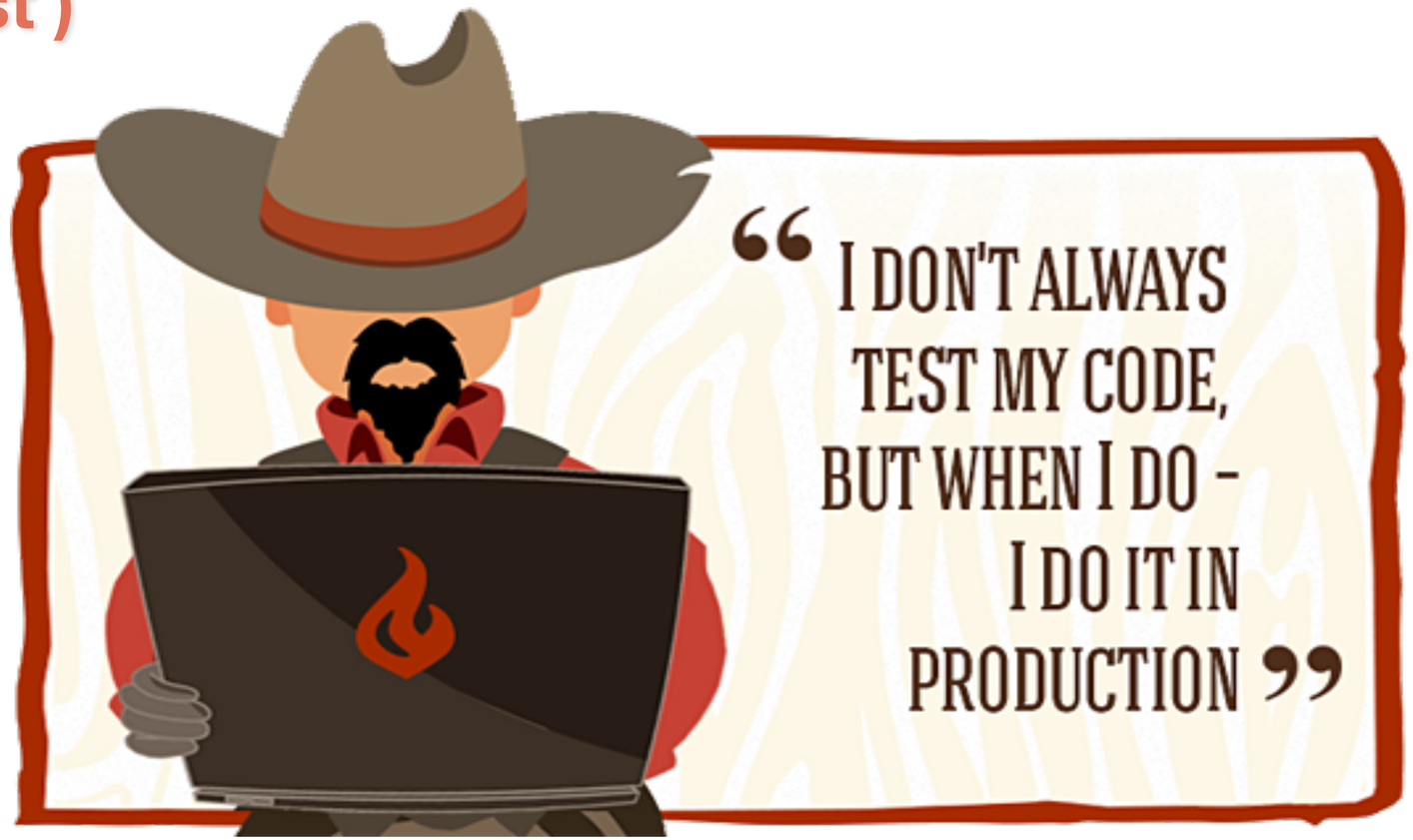
TDD vs BDD

Mocha를 통해 테스트 방식 배우기

목차

1. TDD 소개
2. BDD 소개
3. SuperTest 소개

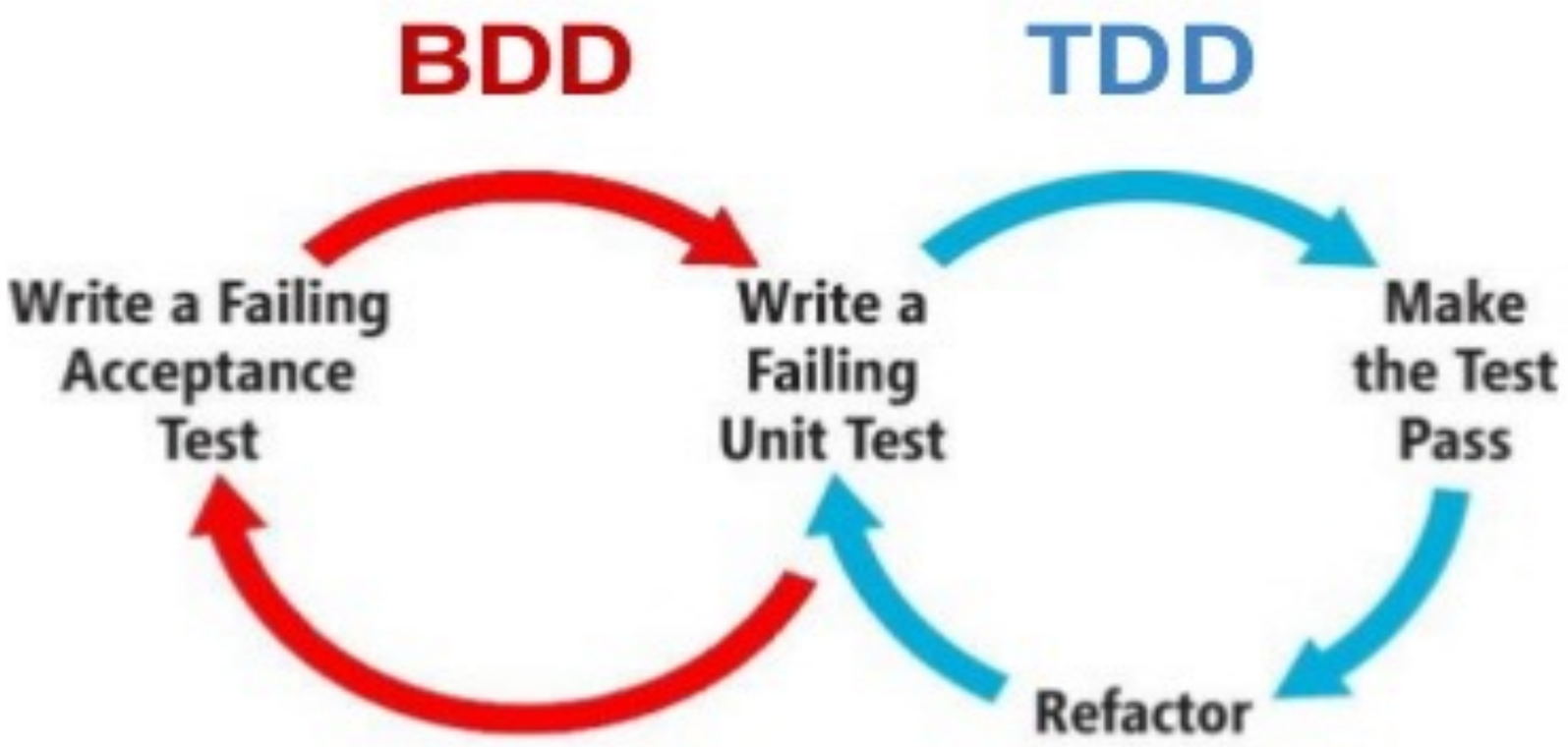
단위 테스트란?
(Unit test)



초기부터 테스트를 하여 견고한 코드를!

대표적 단위 테스트 방법

BDD vs TDD



[출처] http://www.slideshare.net/dimka5/agile-testing-framework-the-art-of-automated-testing

TDD란?

- TDD(Test Driven Development)
  - 작은 단위의 테스트를 초기부터 함

  - 코드 단위(Unit of Code)로
    테스트 하기에 규모가 작음

  - 단계별로 코드를 확실하게 검증

  - 이전 코드를 신뢰하고 코드 추가, 변경을 함

[출처] http://www.slideshare.net/dimka5/agile-testing-framework-the-art-of-automated-testing

TDD 구현 단계

TDD는 다음의 흐름을 따른다.

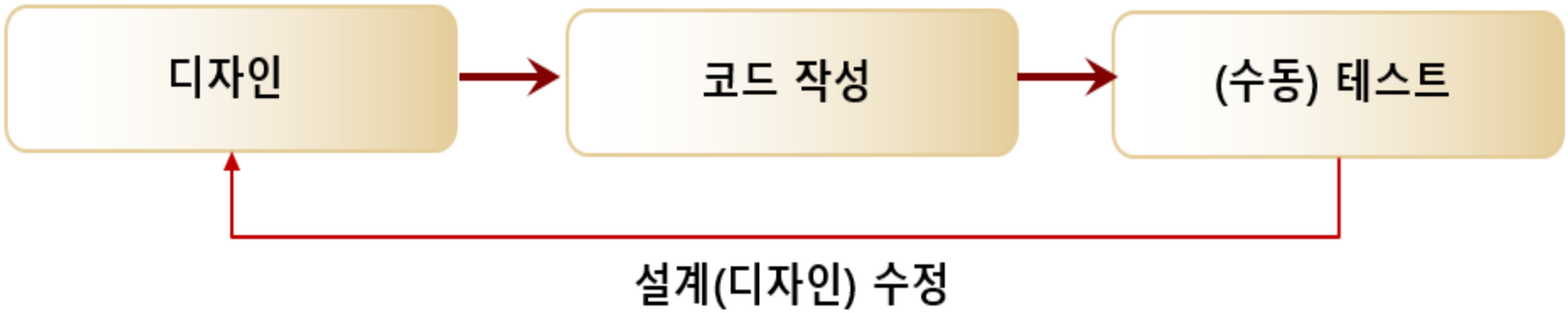
1. define a test set for the unit first
첫번째로 유닛을 위한 테스트 셋을 정의.

2. the implement the unit
유닛을 구현.

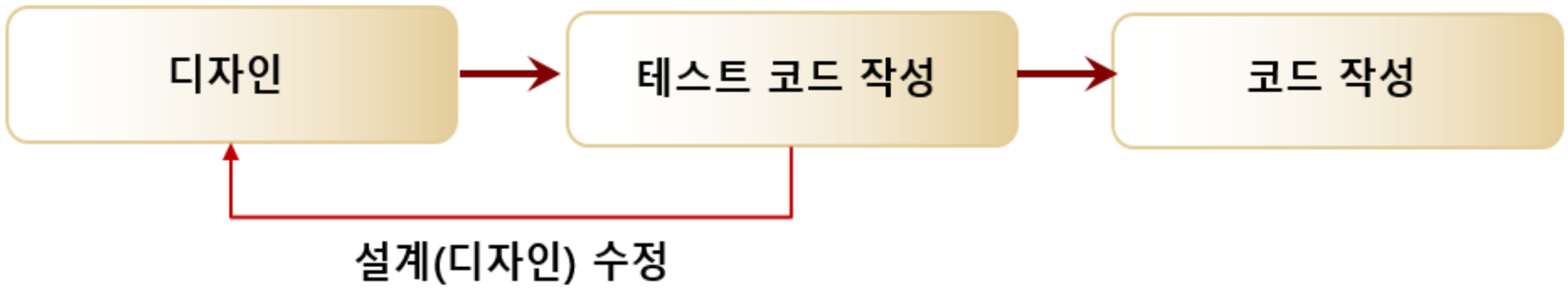
3. finally verify that the implementation of the unit makes the tests success
마지막으로 유닛에 대한 구현이 테스트를 통과하는지 검증.

[출처] http://www.slideshare.net/dimka5/agile-testing-framework-the-art-of-automated-testing

기존 프로세스 vs TDD



기존의 개발 프로세스



TDD 개발 프로세스

[출처] http://www.hoons.net/Lecture/View/644

TDD의 장점

- 테스트 코드 작성 후 코드 작성!

- 기능 하나하나를 구조화할 수 있다!

- 코드 재사용을 고려하기에 튼튼한 코드 작성

- 작은 규모의 테스팅이라 디버깅 용이

[출처] : http://www.hoons.net/Lecture/View/644

TDD의 단점

- TDD는 비즈니스보다 테스트 자체에 집중

- 개발 언어로 쓰여져 개발자 외에는 이해가 어려움

- 문제를 잘못 이해해 엉뚱한 구현으로 이어질 수 있음.

=> 일상적인 언어를 이용한 테스트 방법론이 필요해짐

[출처] http://blog.aliencube.org/ko/2014/04/02/differences-between-bdd-and-tdd/
http://stackoverflow.com/questions/2509/what-are-the-primary-differences-between-tdd-and-bdd#2548

BDD란?

- BDD : Behavior Driven Development
  - 행동, 상황에 중점을 둔 테스트 방식
  - 시나리오를 작성하고 테스트 코드로 바꾼다
  - 일반적으로 쓰이는 언어이기에 이해가 쉽다
  - 일상어라서 테스트 코드와 문제 상황이 자연스럽게 연결

[출처] http://www.slideshare.net/dimka5/agile-testing-framework-the-art-of-automated-testing

User Story

User Story는 대표적인 BDD 방법론 중 하나이다. User가 소프트웨어를 사용하는 상황을 하나의 story로 나타내는 것을 User Story라 부르며 User Story에는 Given, When, Then이 필요하다.

1. 특정 상황이 주어지고 (Given)

2. 어떤 이벤트가 발생했을 때 (When)

3. 그에 대한 결과를 보여준다(Then)

[출처] http://www.slideshare.net/dimka5/agile-testing-framework-the-art-of-automated-testing

User Story의 예

웹사이트에 로그인하는 상황

Story : 유저가 로그인을 한다.
  Given : 유저 이름과 비밀번호가 있을 때
  When : 유저는 이름과 비밀번호로 로그인을 하려 하면
  Then : 로그인 폼이 나타나야 한다.

Given(상황), When(사건), Then(결과)를 통해 자연스럽게
스토리를 나타내는 게 User Story(BDD의 한 방법)이다.

[출처] http://www.slideshare.net/dimka5/agile-testing-framework-the-art-of-automated-testing

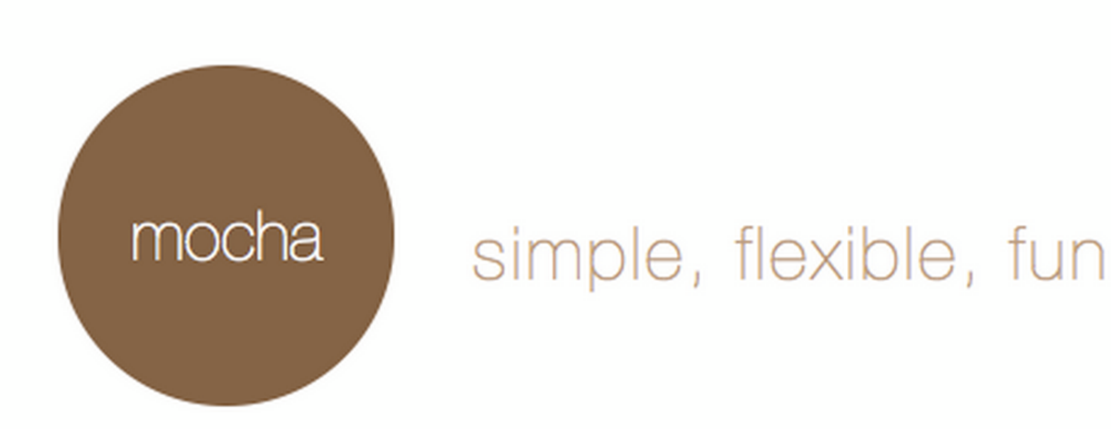
ATM 출금 상황을 User Story로 표현

```
Feature: get cash from an ATM
  Background:
    Given the ATM has 1000
    And the user John is authenticated
    And the user's account has 5000
  Scenario: success
    When the user asks the ATM for 500
    Then the ATM will have 500
    And the user's account will have 4500
    And the ATM will provide 500 in cash
  Scenario: not enough money in the ATM
    When the user asks the ATM for 1500
    Then the ATM will have 1000
    And the user's account will have 5000
    And the ATM will notify the user it does not have enough cash
```

Success 시나리오를 뒤에서 테스트 코드로 바꿔봅니다.

[출처] http://eamodeorubio.github.io/bdd-with-js/#/6

Mocha – 자바스크립트 테스트 프레임워크



Mocha is a feature-rich JavaScript **test framework running on node.js** and the browser, making asynchronous testing simple and fun. Mocha tests run serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Hosted on GitHub.

https://github.com/mochajs/mocha

Mocha 설치법

```
$ npm install -g mocha
$ mkdir test
$ cd test
$ vi atm.js
$ vi mocha_bdd_test.js
```

atm.js (로직)

```
1  var totalDeposit = 0;
2  var Account = function(deposit){
3    totalDeposit = deposit;
4  };
5  Account.prototype.getAccount= function(){
6    return totalDeposit;
7  };
8  Account.prototype.withDraw = function(money){
9      this.money = money;
10     totalDeposit -= money;
11 };
12 Account.prototype.AtmRemainingCash = function(){
13   return this.money;
14 };
15
16 module.exports = Account;
17
```

[출처] http://eamodeorubio.github.io/bdd-with-js/#/15

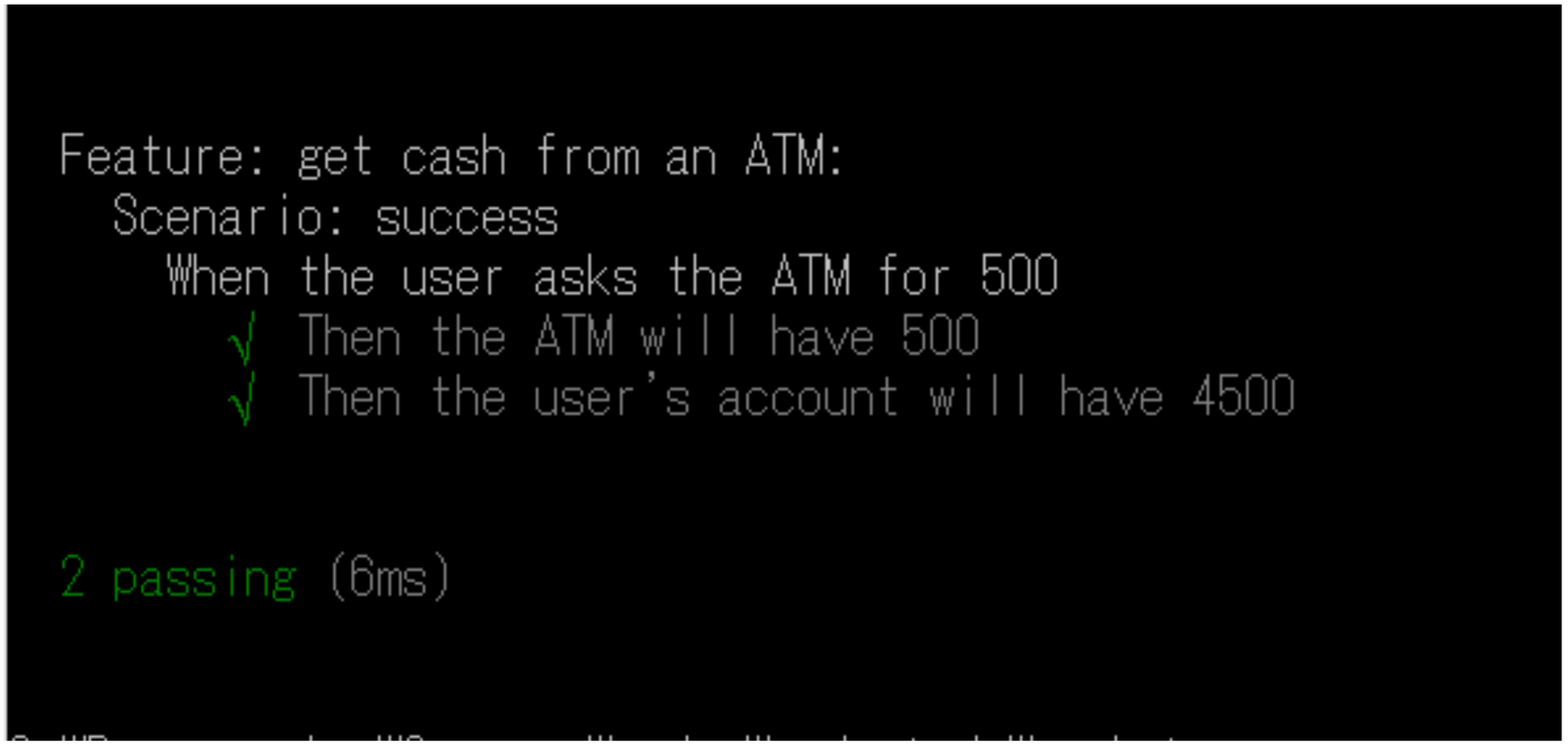
BDD예제 : mocha_bdd_test.js (테스트)

```
// Mocha BDD Interface
// describe 함수는 given, when을 나타내는데 쓰이고
// context 함수는  Scenario를 표현하는 데 쓰인다.
// it 함수는 then, User story의 결론을 보여준다.
var assert = require('assert');
var Atm = require('./atm');
var atm = Atm(5000);
// ATM에서 현금을 인출할 때
describe('Feature: get cash from an ATM:', function() {
  context('Scenario: success', function() { // Scenario : 현금인출 성공한 상황
    describe('When the user asks the ATM for 500', function() { // When : 유저가 ATM에서 500을 꺼내려할 때
      atm.withDraw(500);
      it('Then the ATM will have 500', function() { // Then : ATM은 500을 갖게 된다.
        assert.equal(  atm.AtmRemainingCash() , 500);
      });
      it("Then the user's account will have 4500", function(done) { // Then : 유저의 계좌에는 4500이 남게 된다.
        assert.equal( atm.getAccount(), 4500);
        done();
      });
    });
  });
});
```

[소스] : https://gist.github.com/Hochul822/7e5a250eb36279a4f03d

Mocha, BDD로 테스트하기

```
$ mocha  mocha_bdd_test.js
```



TDD 예제 : ./math.js (로직)

```
1  exports.abs = function(number){
2      return number < 0 ? -number : number;
3  };
4
5  exports.pow = function(x, y){
6      var value = 1;
7
8      for( var i = 0; i < y; i++ ){
9          value *= x;
10     }
11     return value;
12 };
13
14 exports.asyncAbs = function(number, cb){
15     setTimeout(function(){
16         cb(null, exports.abs(number));
17     }, 100);
18 };
```

소스 : https://gist.github.com/Hochul822/da1fd4d547daba3fa203

TDD 예제 : ./test/mocha_tdd_test.js

```
1  var assert = require('assert');
2  var math = require('../math');
3
4  suite('math test', function(){
5      test('math.pow test', function(){
6          assert.deepEqual(4, math.pow(2,2));
7          assert.deepEqual(1, math.pow(1,2));
8      });
9
10     test('math.abs test', function(){
11         assert.deepEqual(10, math.abs(-10));
12         assert.deepEqual(10, math.abs(10));
13     });
14
15     test('math.asyncAbs test', function(done){
16         math.asyncAbs(-10, function(err, number){
17             assert.ifError(err);
18             assert.deepEqual(10, number);
19             done();
20         });
21     });
22 });
23
```

소스 : https://gist.github.com/Hochul822/da1fd4d547daba3fa203

Mocha, TDD로 테스트하기

```
$ mocha -u tdd mocha_tdd_test.js
```

BDD와 TDD의 차이는?

```
describe(‘describe situation that’, function(){
  it('should result that’, function() {
    // write test logic
}});
```

VS

```
suite('#example', function() {
        test('this is a test', function() {
            // write test logic
        });
    });
});
```

BDD는 상황을 그리는 데(describe) 중점을 두는 반면,
TDD는 적합한지(suite)를 판별하려 합니다.

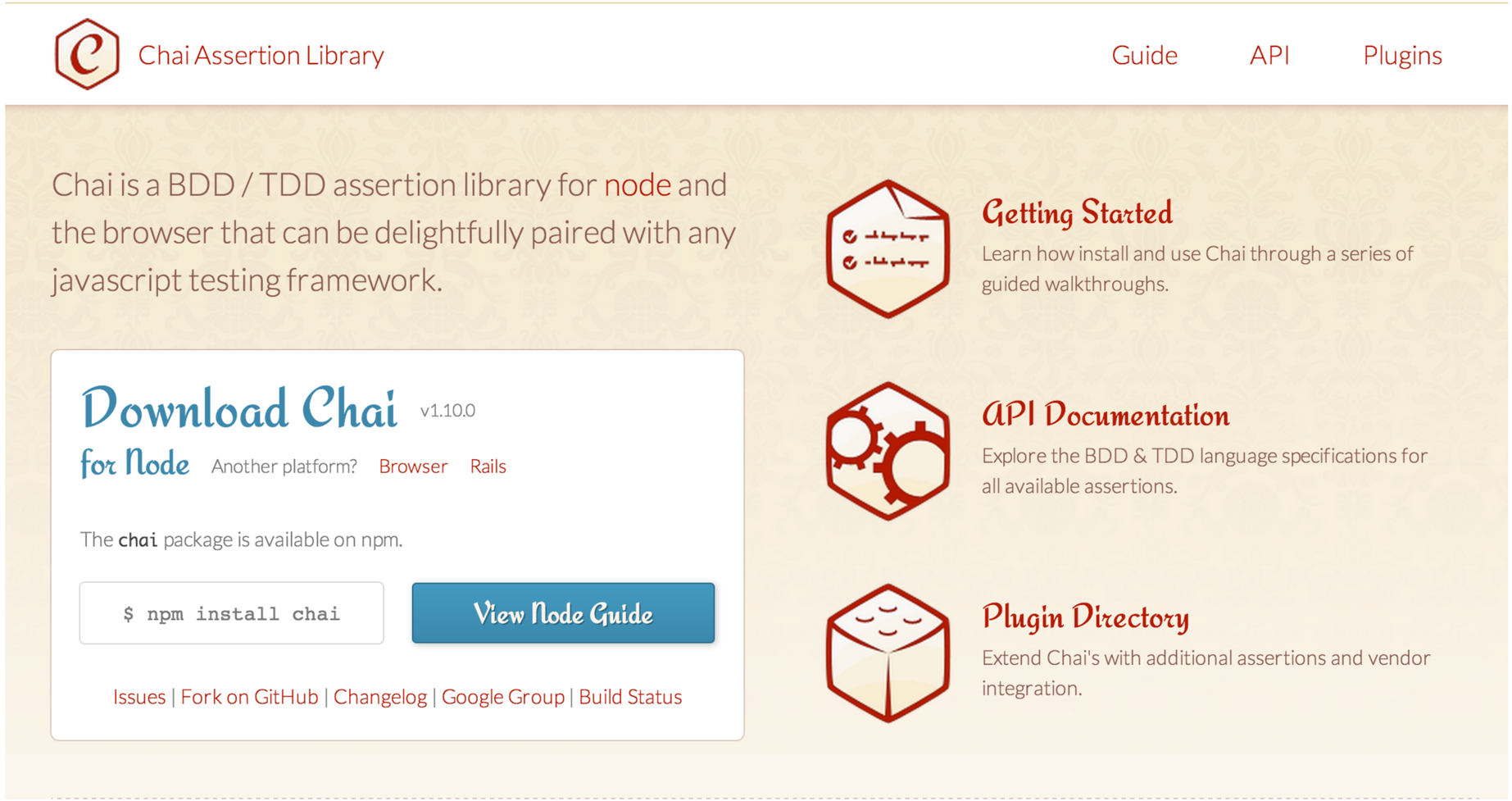
Hooks – Preconditions을 셋업하는 데 도움을 줌

```
suite('TDD Style', function() {
    suiteSetup(function() {
        // executed before test suit
   });

    suiteTeardown(function() {
        // executed after test suit
    });

    setup(function() {
        // executed before every test
    });

    teardown(function() {
        // executed after every test
    });
```

Hooks – Mocha에서 제공하는 테스트 셋업 관련 기능

```
describe('BDD Style', function() {
    before(function() {
        // executed before test describe
    });

    after(function() {
        // executed after test describe
    });

    beforeEach(function() {
        // executed before every test
    });

    afterEach(function() {
        // executed after every test
    });
```

Mocha외의 유용한 모듈 - Chai



http://chaijs.com/

SuperTest

– RestFul API를 테스트 하는 모듈

- 고 레벨 수준에서 Http를 테스트 하는데 유용하다

- SuperTest 및 express 설치(Restful 프레임워크)

```
$ npm install supertest -save-dev

$ npm install -g express-generator
$ express myapp

$ cd myapp

$ npm install
```

Supertest – Restful에서 get method 테스트

```
$ node super_test.js
```

```
1  var request = require('supertest');
2  var express = require('express');
3  var app = express();
4
5  app.get('/users', function(req, res){
6    res.send(200, { name : 'tobi'});
7  });
8  // get 함수를 통해 users에 request한 값이 잘 들어오는지 확인합니다.
9  // expect는 들어온 값들이 예상한(expect)한 값과 일치하는지 확인합니다.
10 request(app)
11   .get('/users')
12   .expect('Content-Type', /json/)  // Content-Type이 json인지 확인
13   .expect('Content-Length', '15')  // Content-Length가 15인지 확인
14   .expect(200) // response code가 200인지 확인합니다.
15   .end(function(err, res){
16     if(err) throw err;
17 });
```

소스 : https://gist.github.com/Hochul822/741bdf49a03c90de3516

정상이면 아무런 에러 없음
문제가 생길때만 뜸

-> 이래서 Supertest는 Mocha와 병행을..

http://chaijs.com/

Supertest + BDD는?
-> Mocha를 이용!

```
$ mocha super_test_bdd.js
```

```
1  var request = require('supertest');
2  var express = require('express');
3  var app = express();
4
5  app.get('/users', function(req, res){
6    res.send(200, { name : 'tobi'});
7  });
8
9  describe('GET /users', function(){
10   it('respond with json', function(done){ //
11     request(app)
12     .get('/users')
13     .expect('Content-Type', /json/)
14     .expect('Content-Length', '15')
15     .expect(200)
16     .end(function(err, res){
17       if(err) throw err;
18       done(); // mocha를 쓸 때는 반드시 done()을 end 함수 안에 넣어줘야합니다.
19     });
20   })
21 })
```

로직

BDD 추가

테스트

소스 : https://gist.github.com/Hochul822/741bdf49a03c90de3516

Q&A