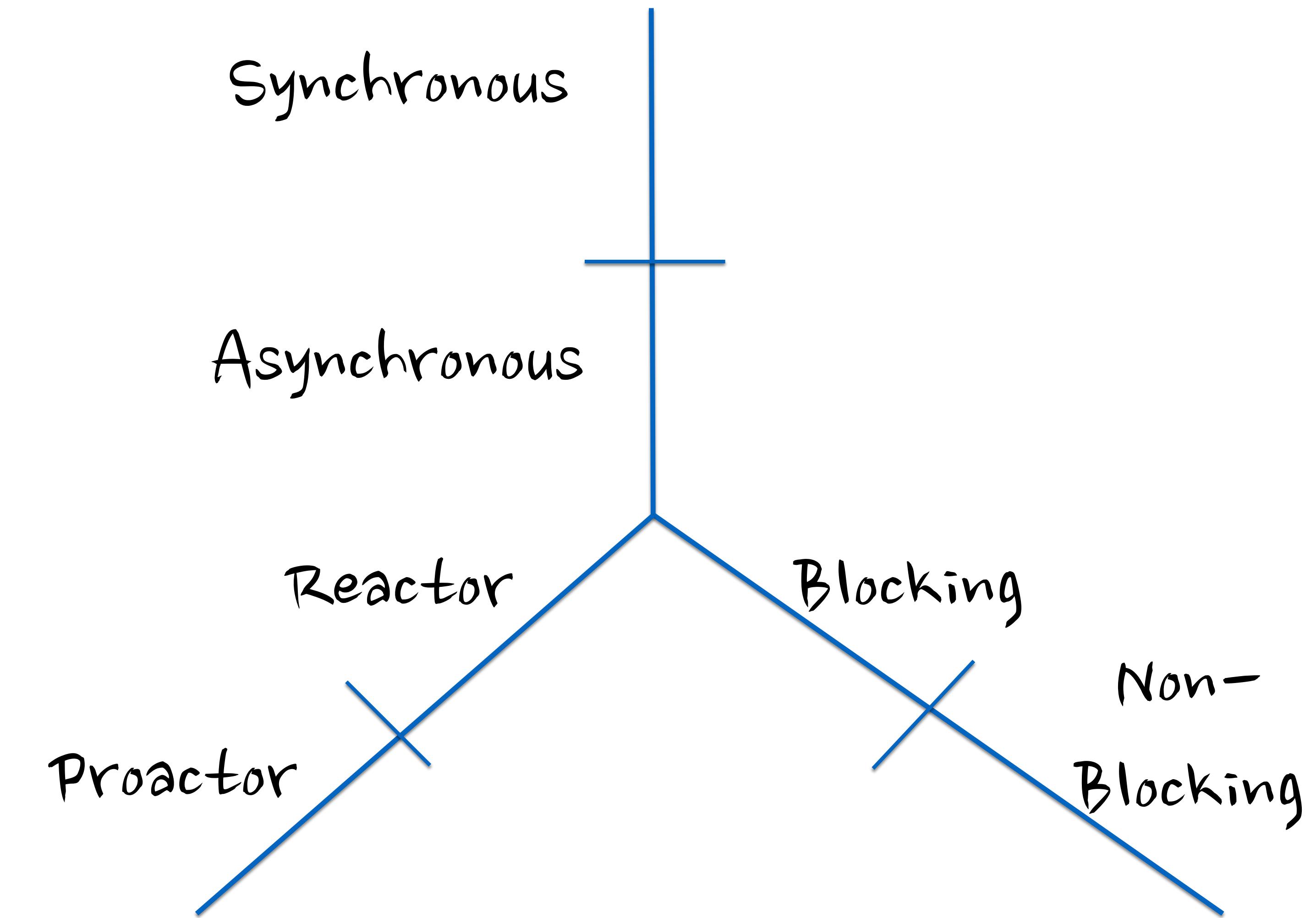


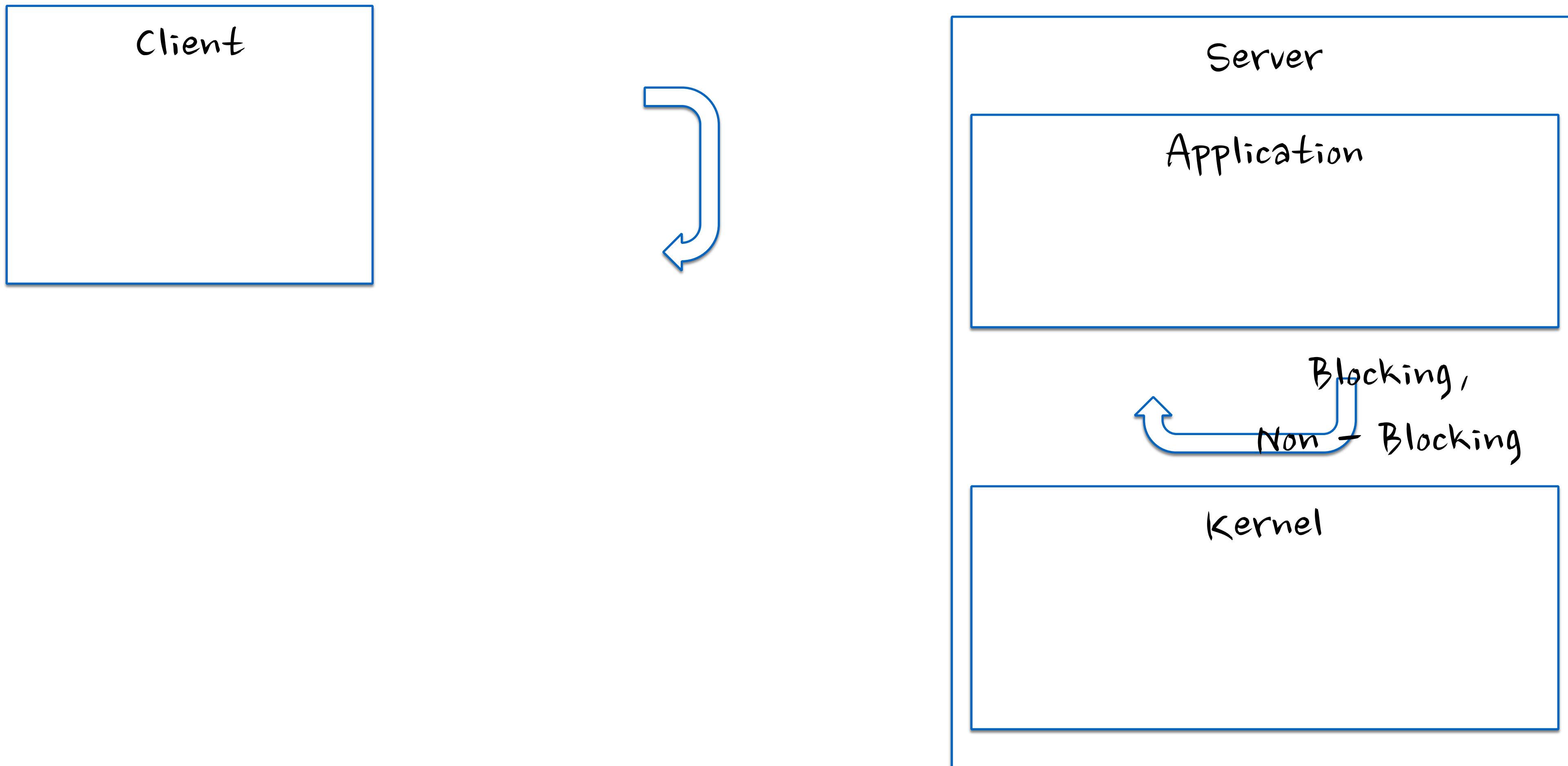
Node.js

베이엔드 개발을 가라며..
다 아시겠지만...

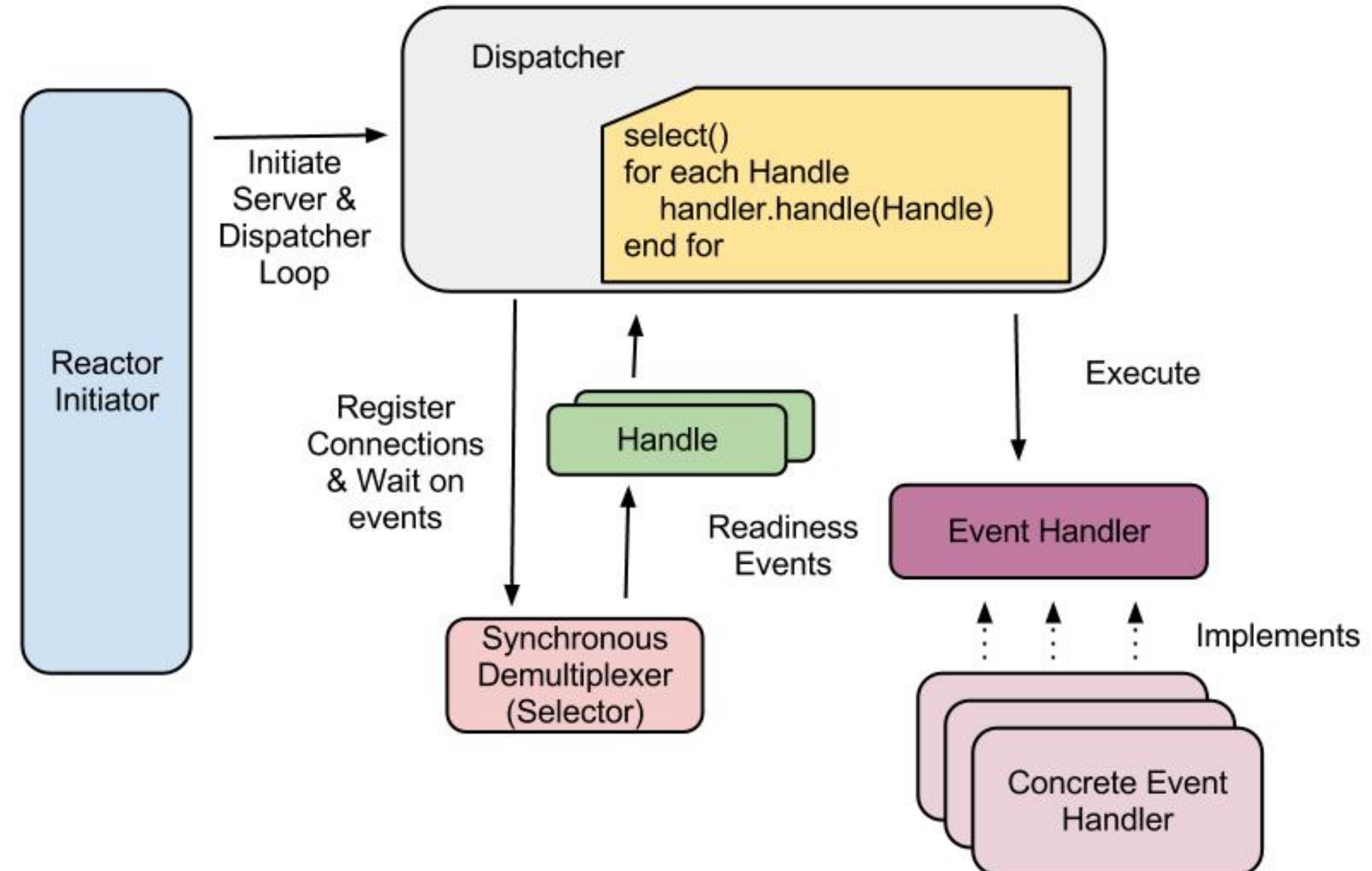
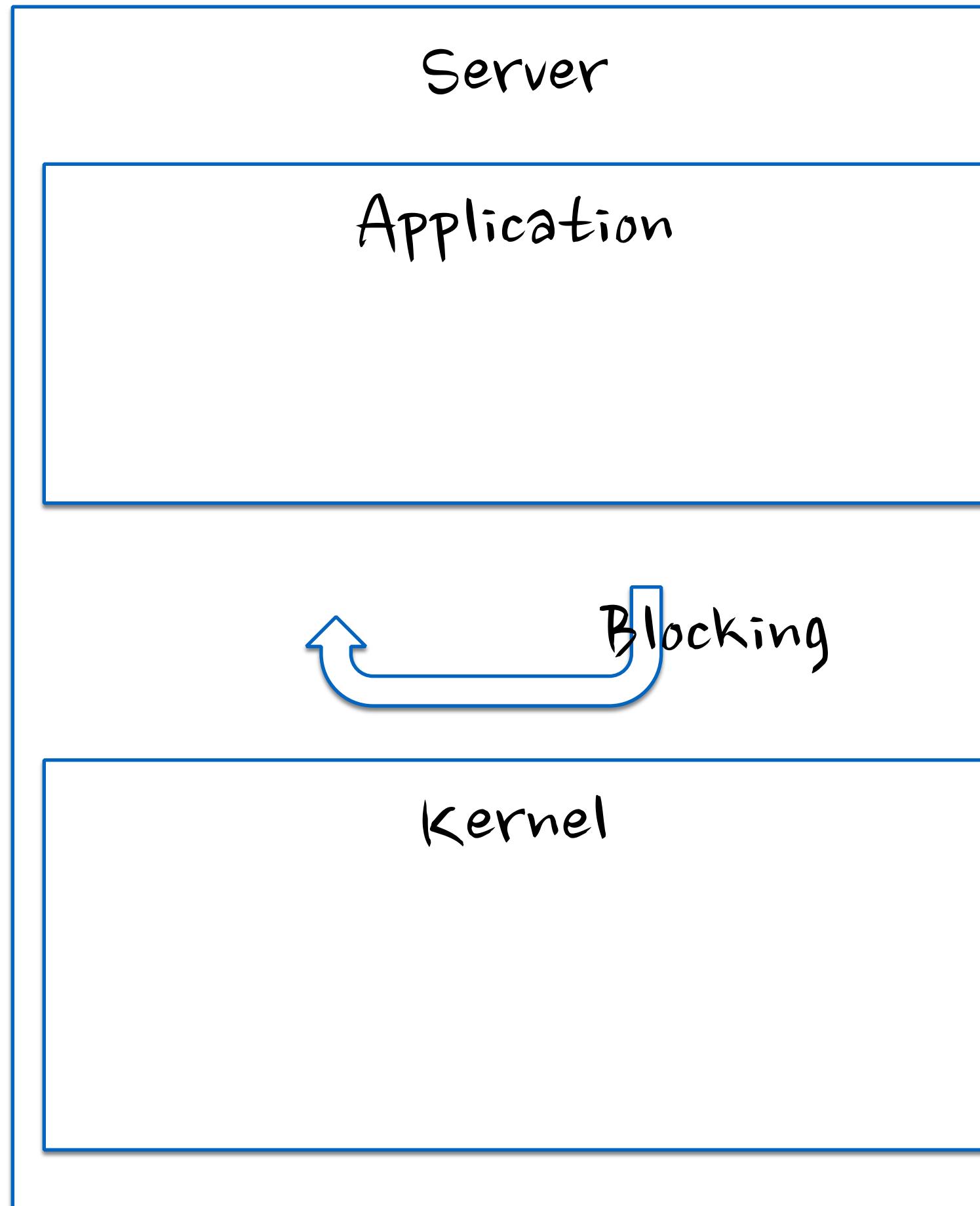




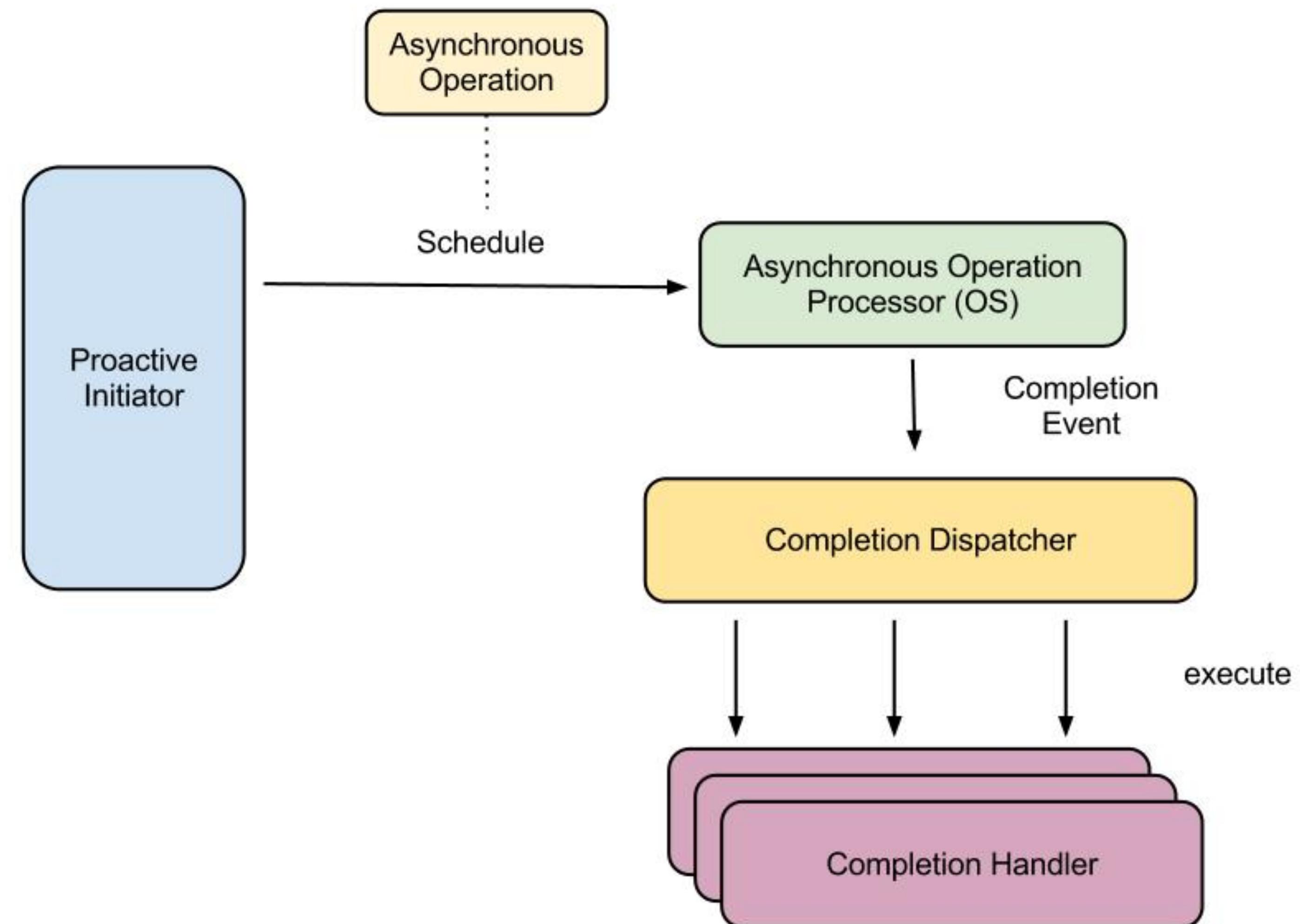
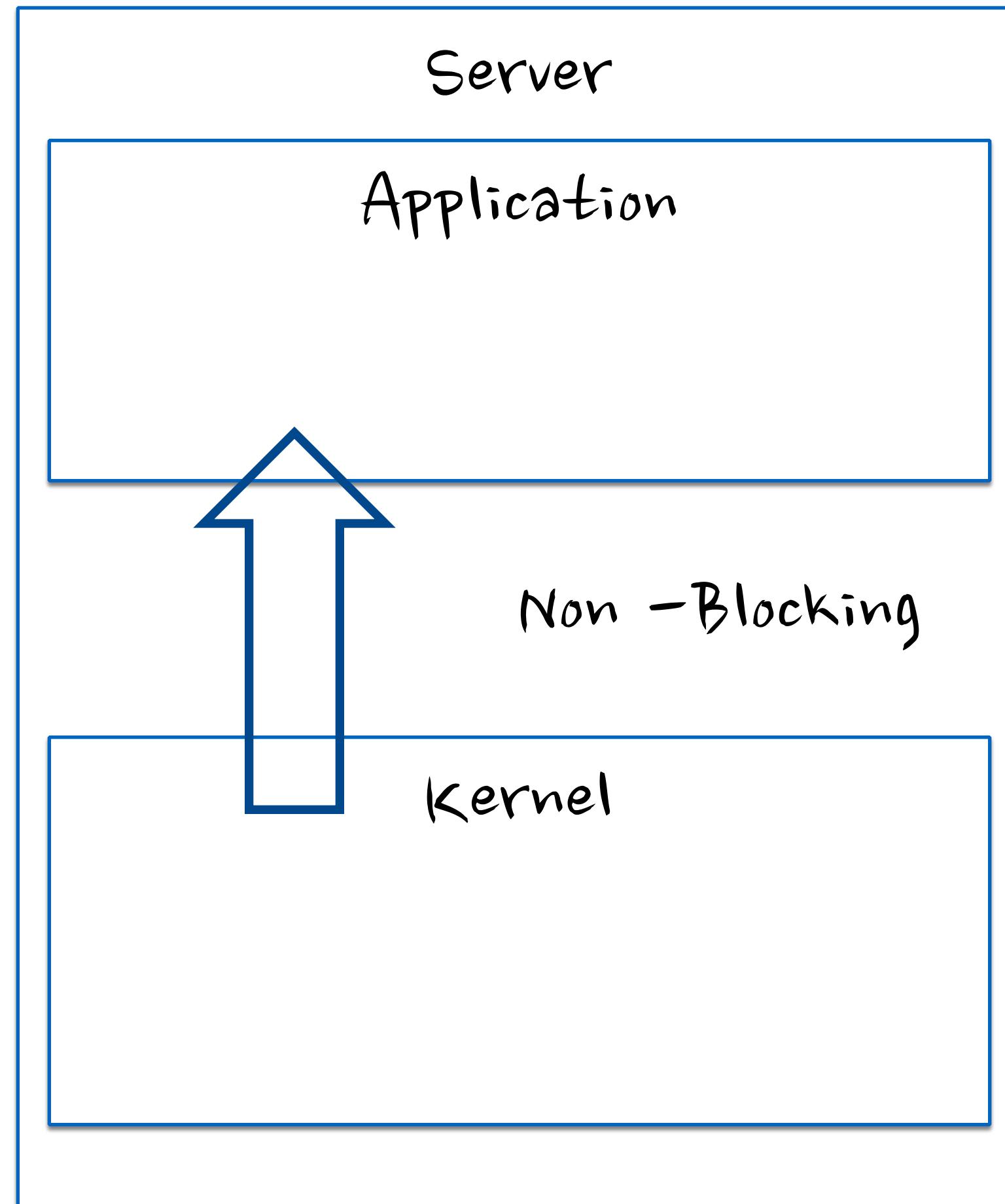
Sync/Async



Reactor

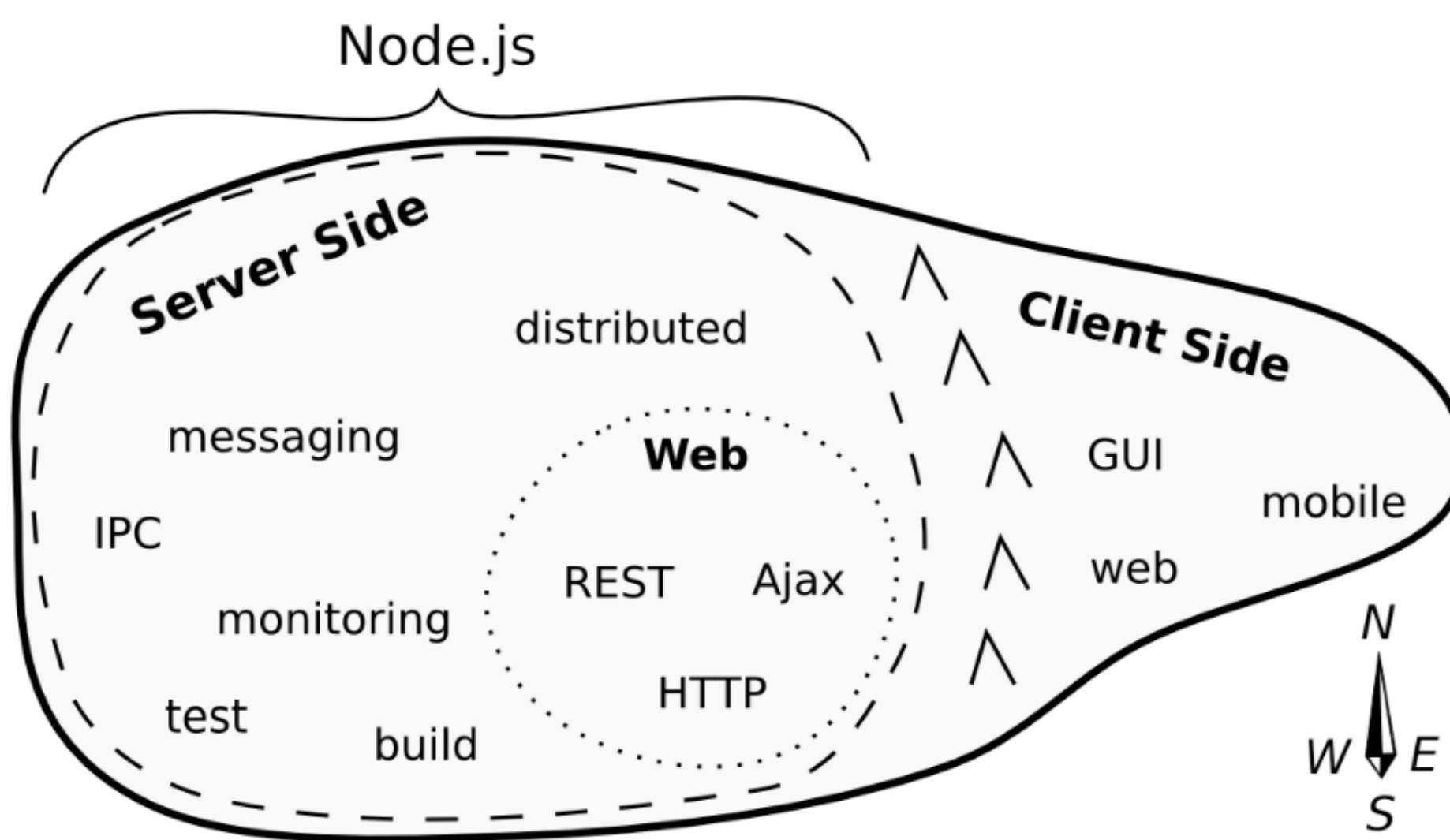


Proactor



Part 1. Node.js Intro!

#01. Node.js Introduction



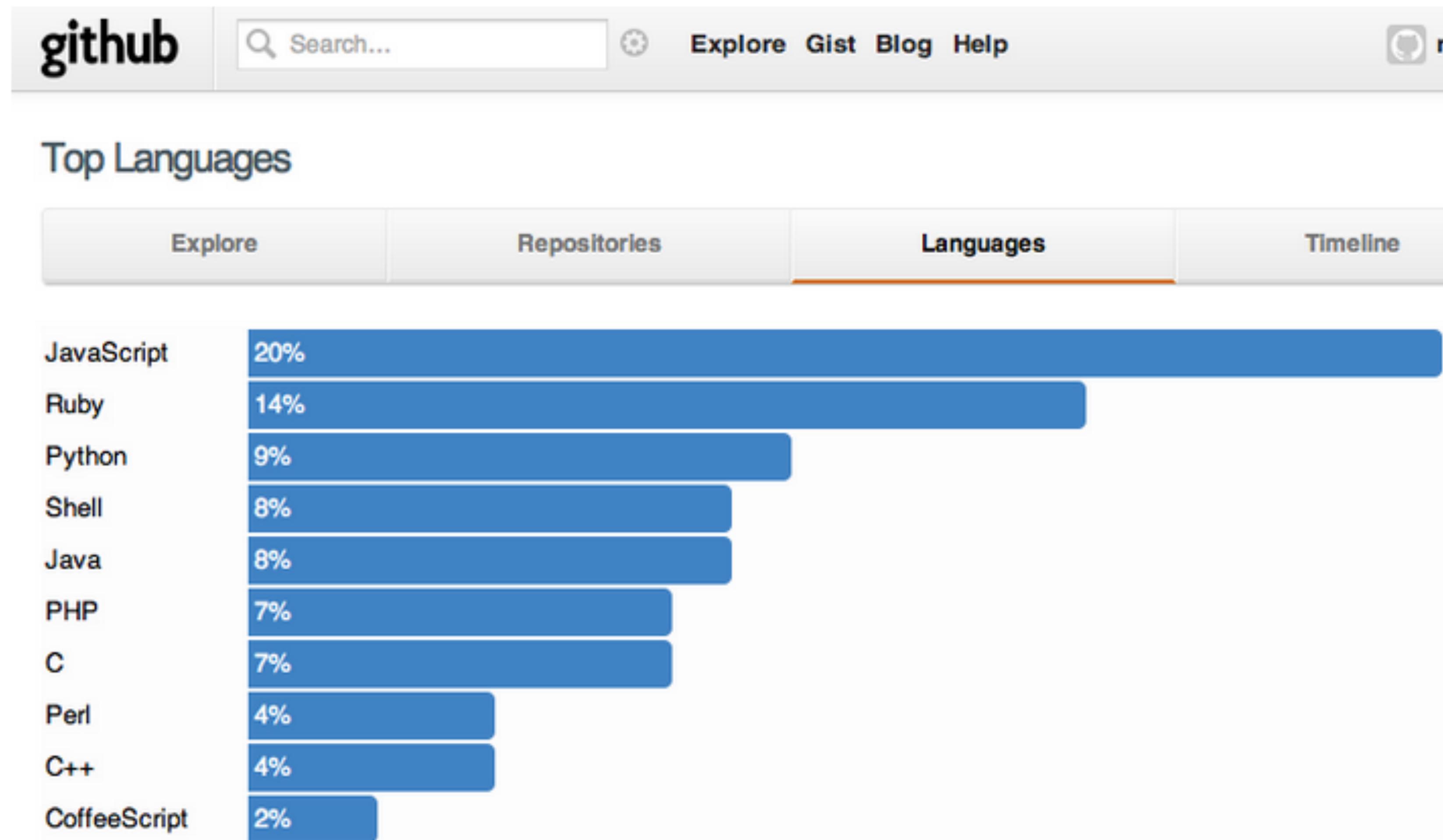
- Server side Javascript
- Event driven
- Asynchronous
- Non-Blocking I/O
- Single Threaded
- Lightweight
- Fast

Node.js Benefits #1



JavaScript

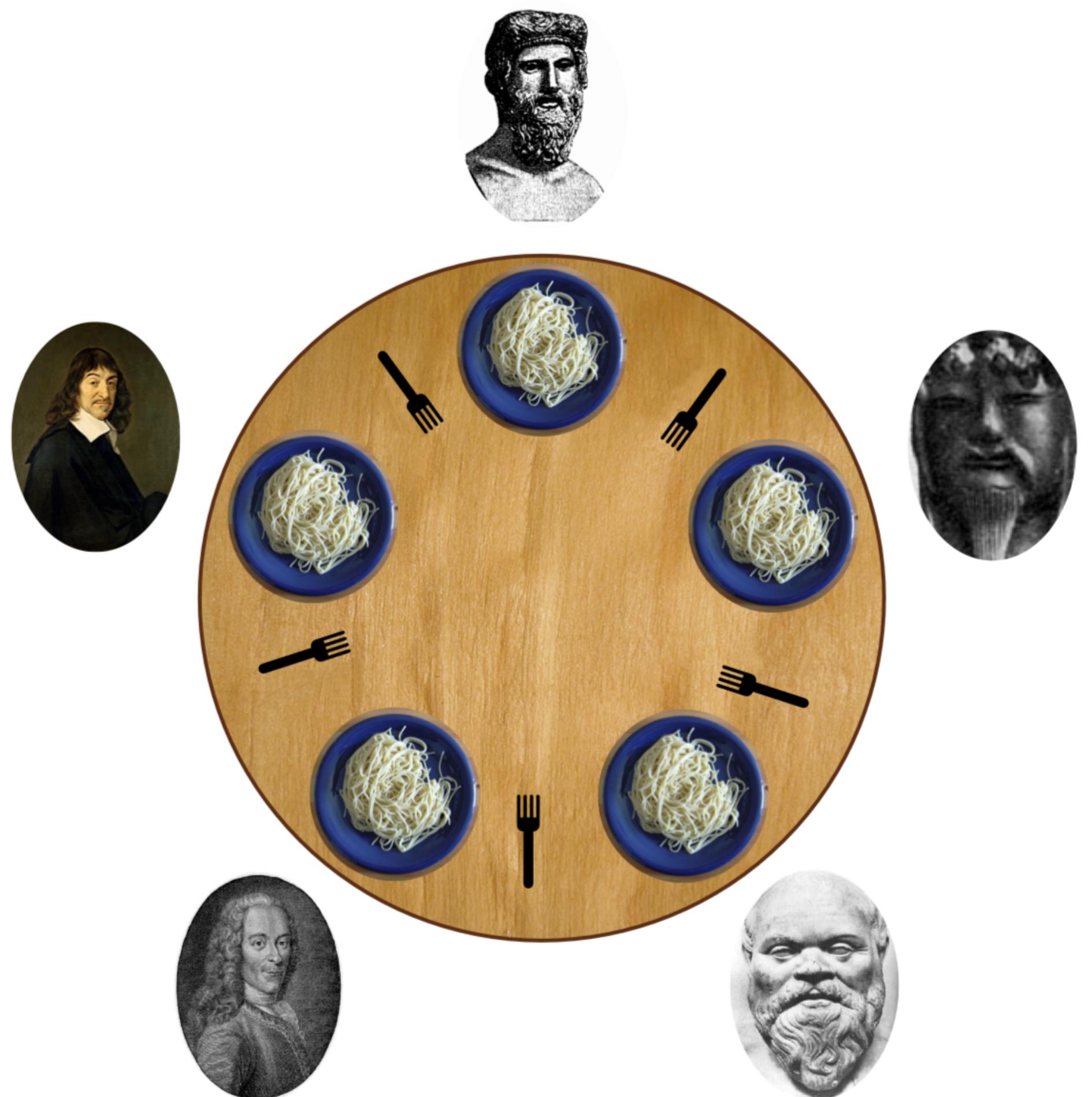
JavaScript is Mainstream.. in github..



Node.js Benefits #2 -> MEAN Stack

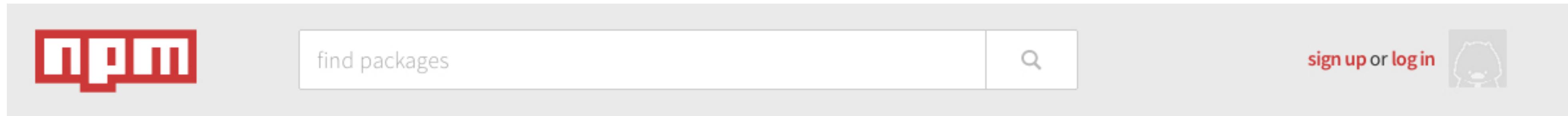


Node.js Benefits #3



Single thread

Node.js Benefits #4



npm is the package manager for javascript.

116,156
total packages

29,280,711
downloads in the last day

123,347,027
downloads in the last week

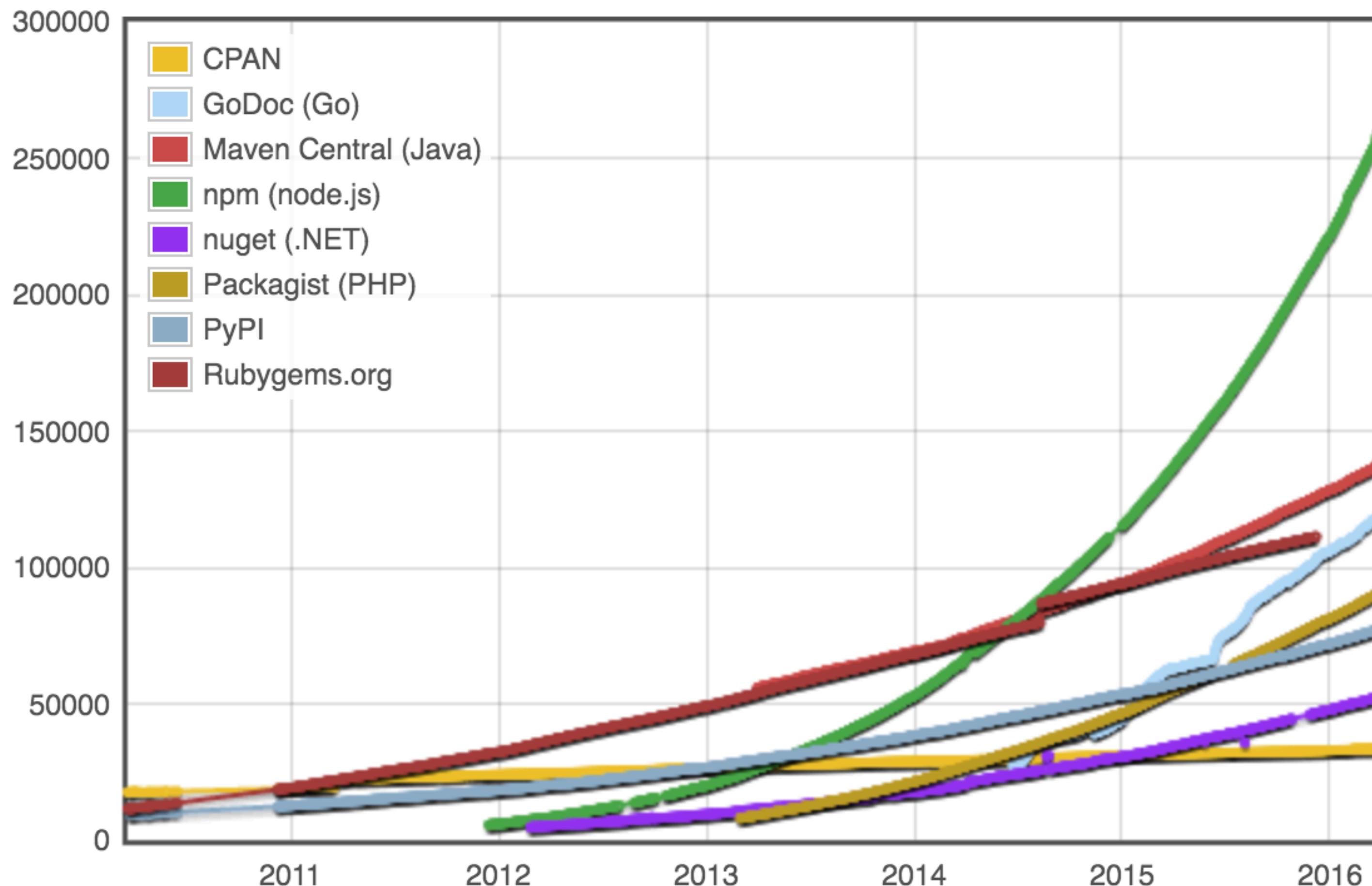
617,334,443
downloads in the last month

packages people 'npm install' a lot

Node Package Manager

Node.js Benefits #4

Module Counts



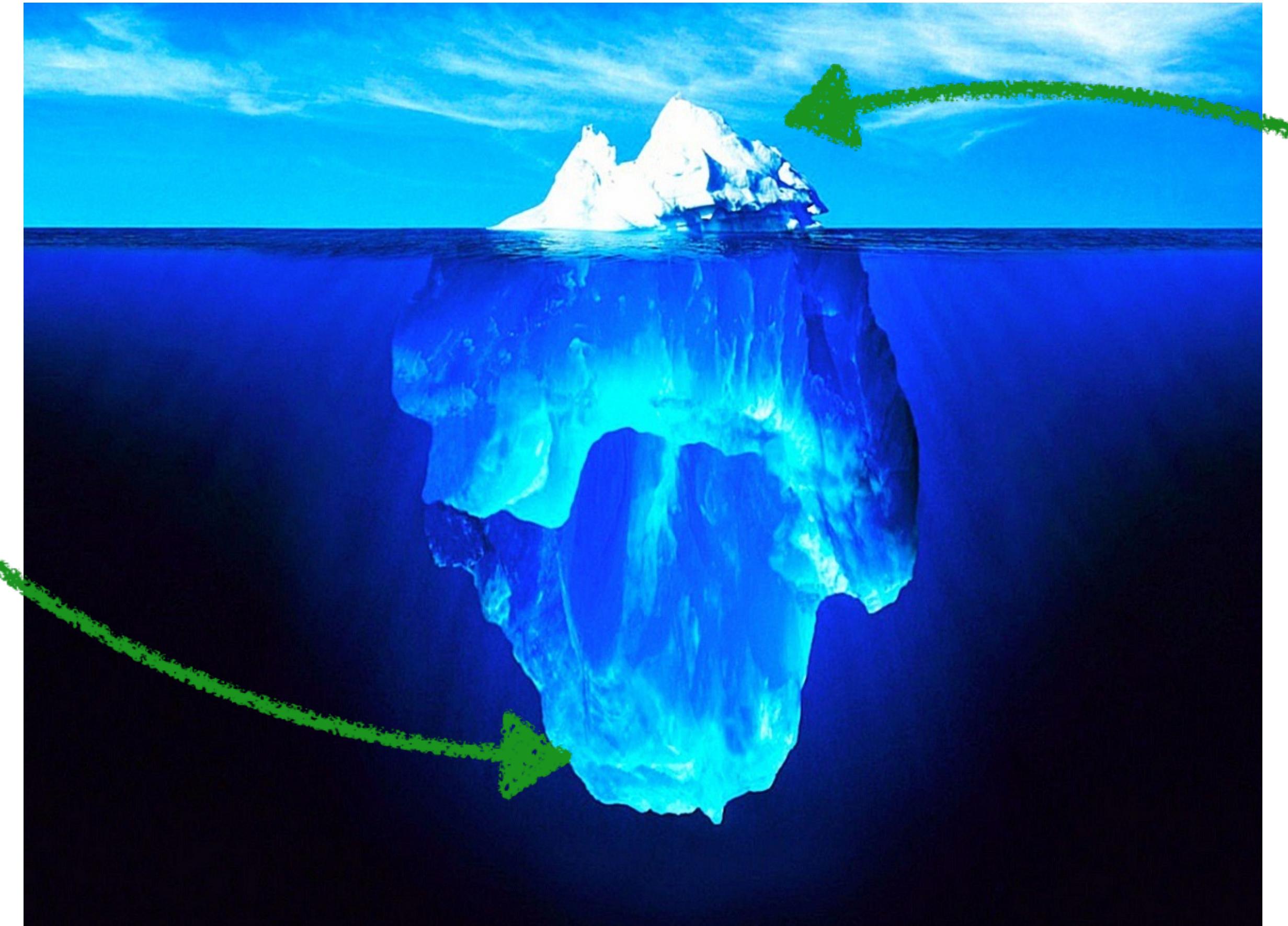
Node.js Drawbacks #1

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

Callback hell!!

Node.js Drawbacks #2

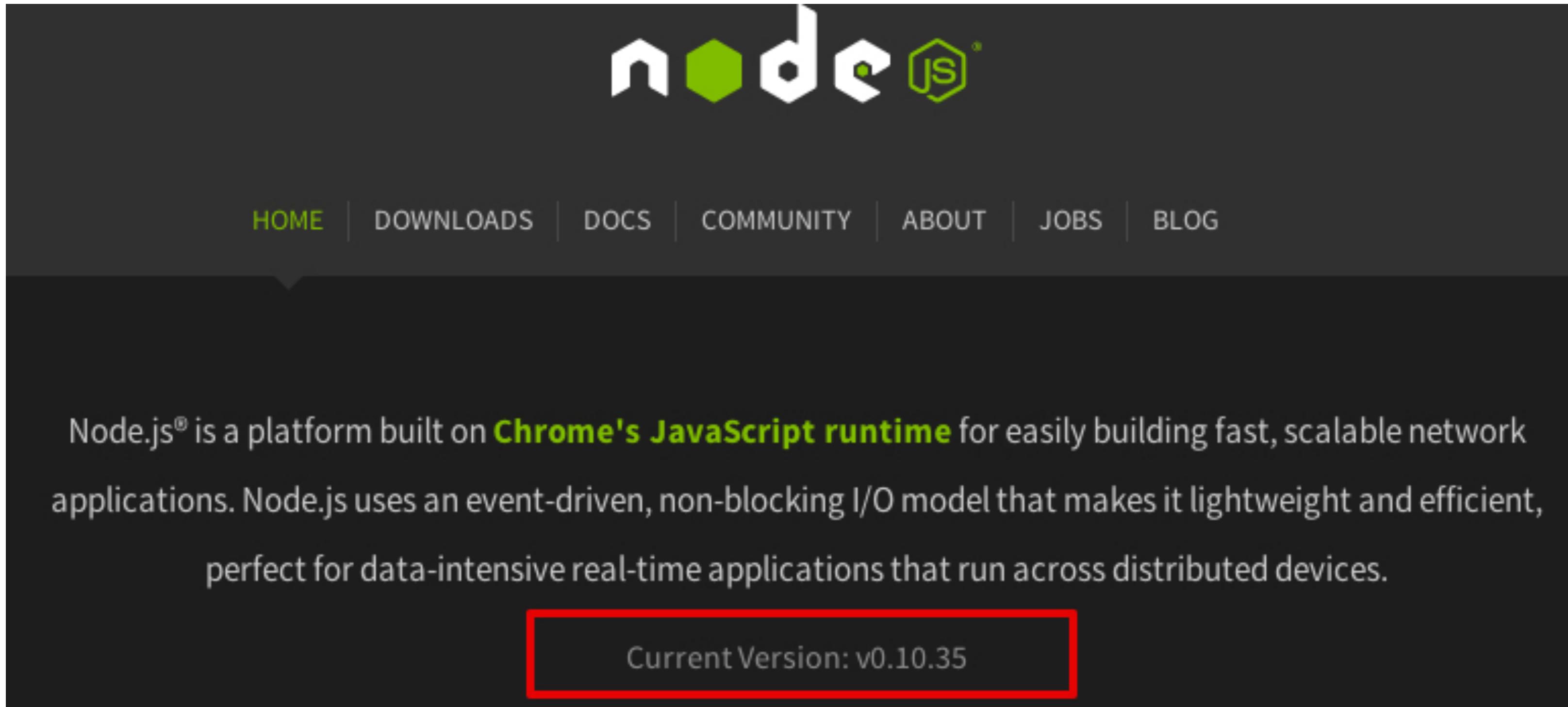
JS is not Java!!



Java??

Javascript..

Node.js Drawbacks #3



v0.10, v0.12, v4.x, v6.x

Node.js Applications

Network
Program

Tools

Robot



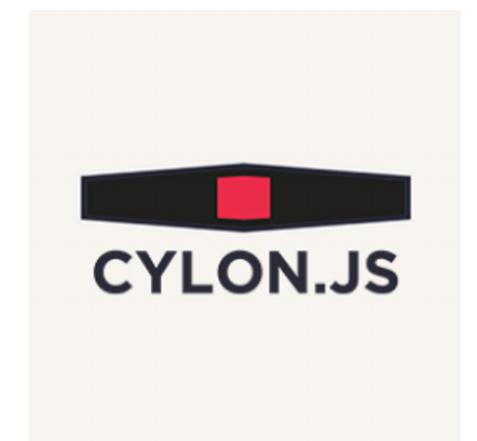
YO



GRUNT



BOWER



Desktop Application



Node.js References

YAHOO!

yammerTM
The Enterprise Social Network

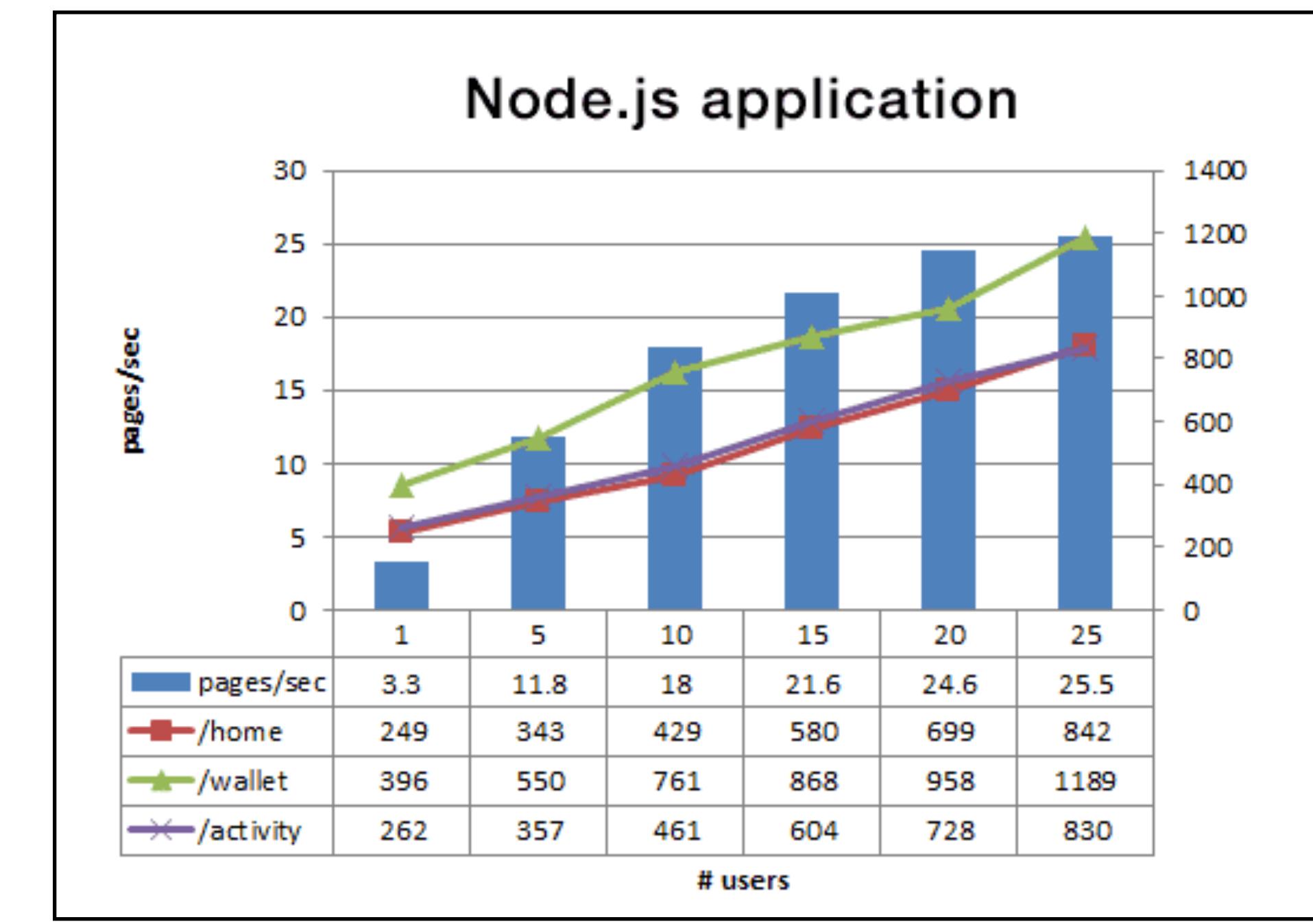
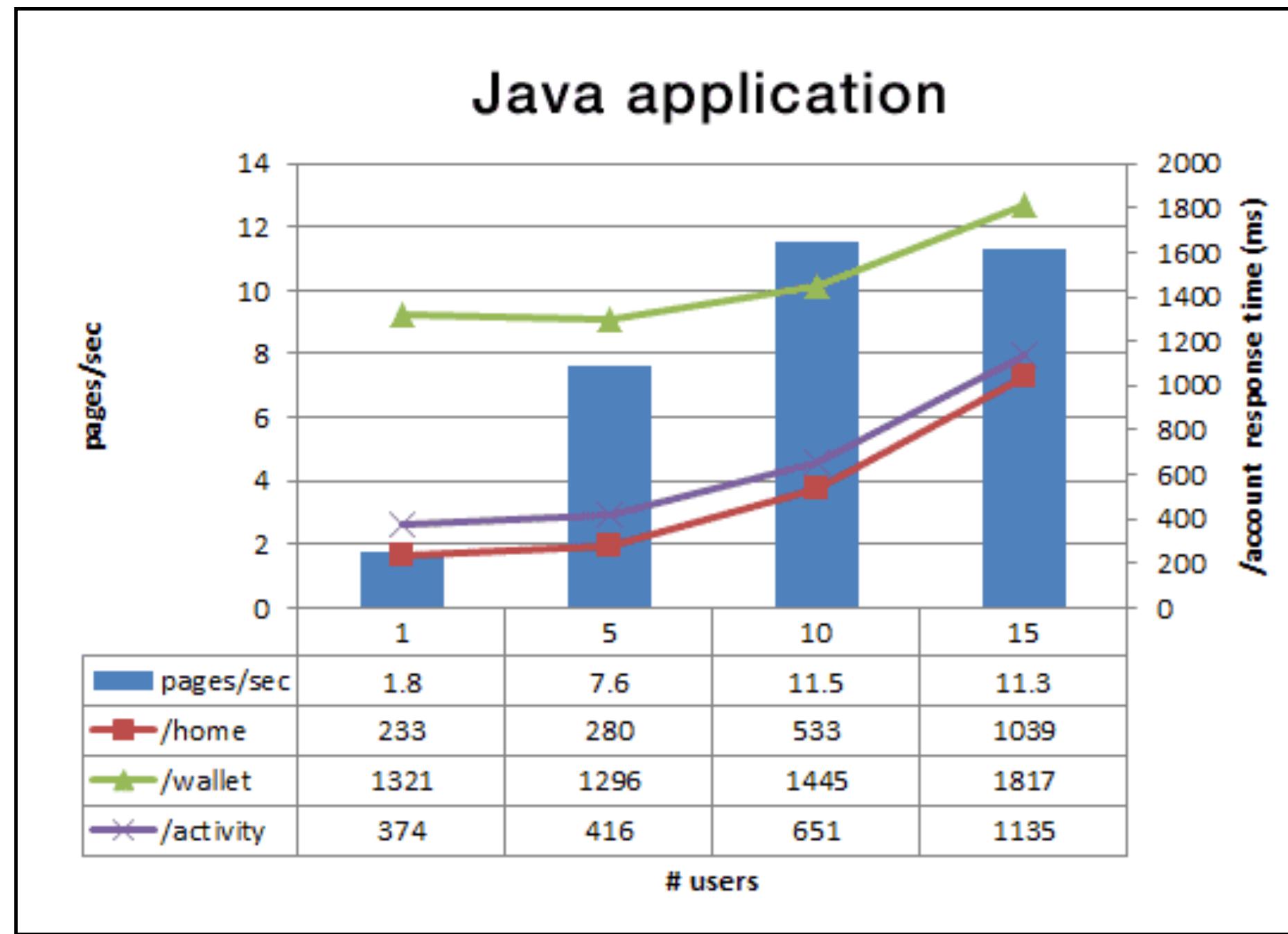
LinkedInTM

U B E R

Windows® AzureTM

PayPalTM

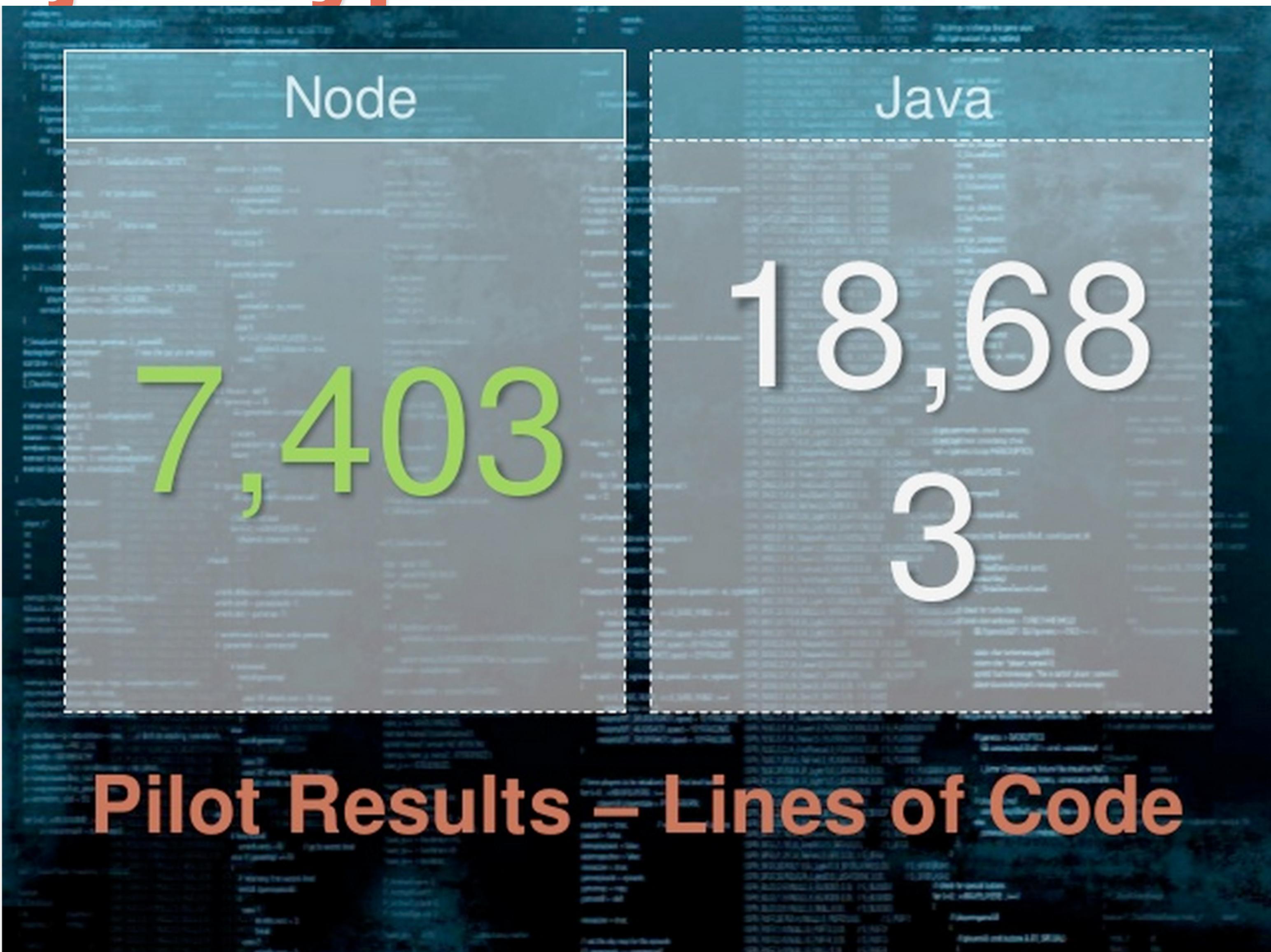
Node.js Paypal



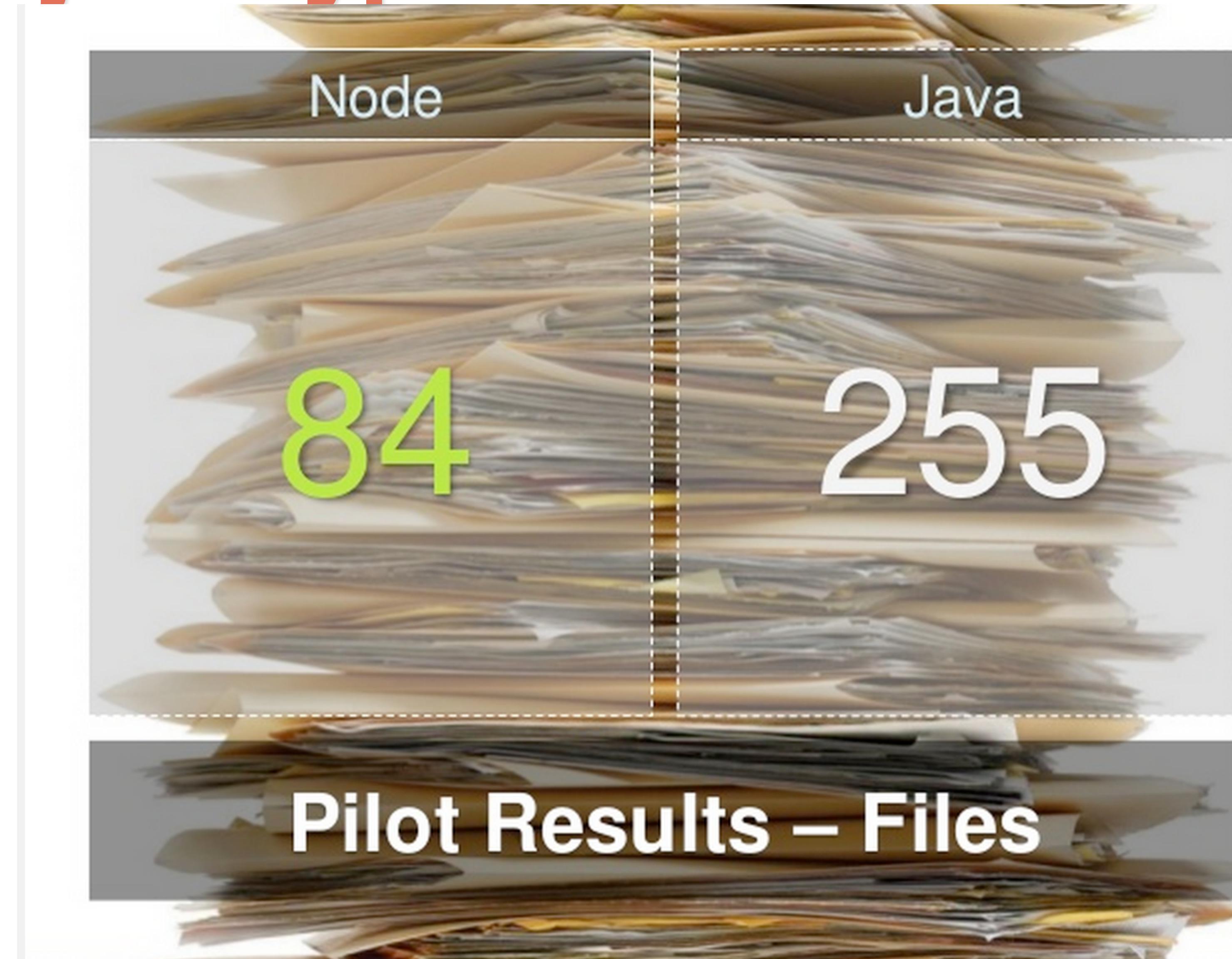
Double the requests per second vs. the Java application. This is even more interesting because our initial performance results were using **a single core for the node.js** application compared to **five cores in Java**. We expect to increase this divide further.

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.

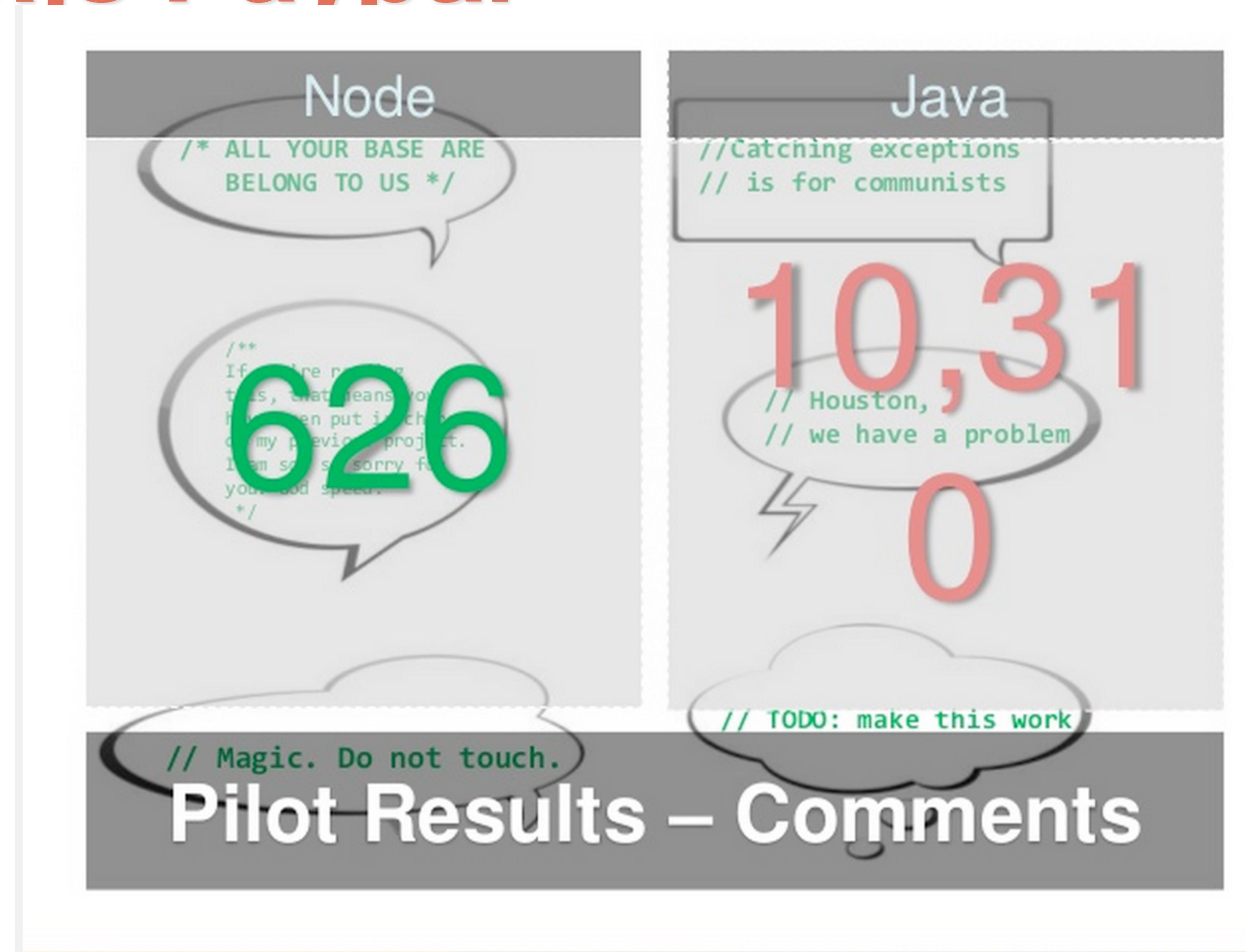
Node.js Paypal



Node.js Paypal



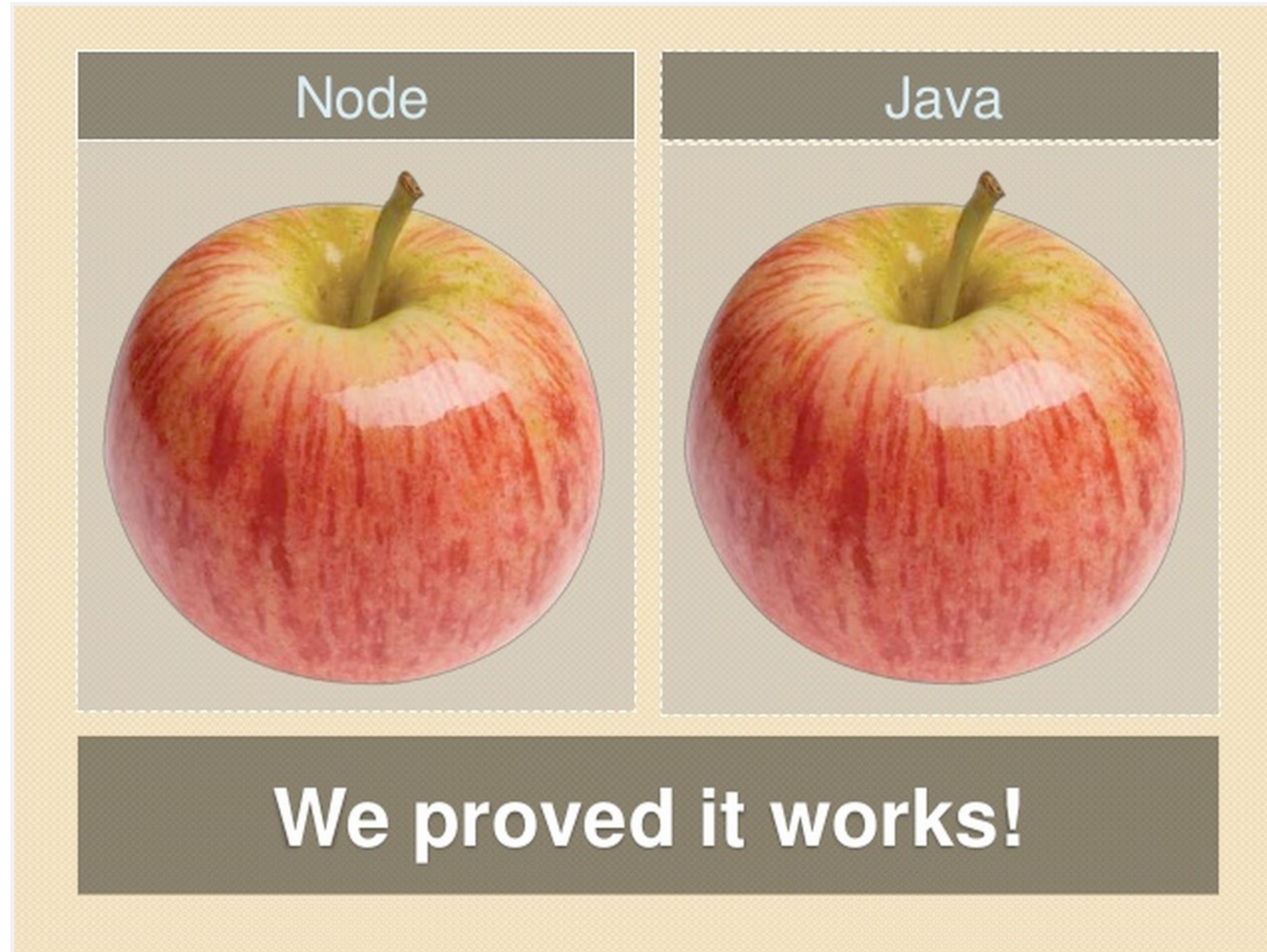
Node.is Paypal



Node.js Paypal



Node.js Paypal



Node.js Good Points

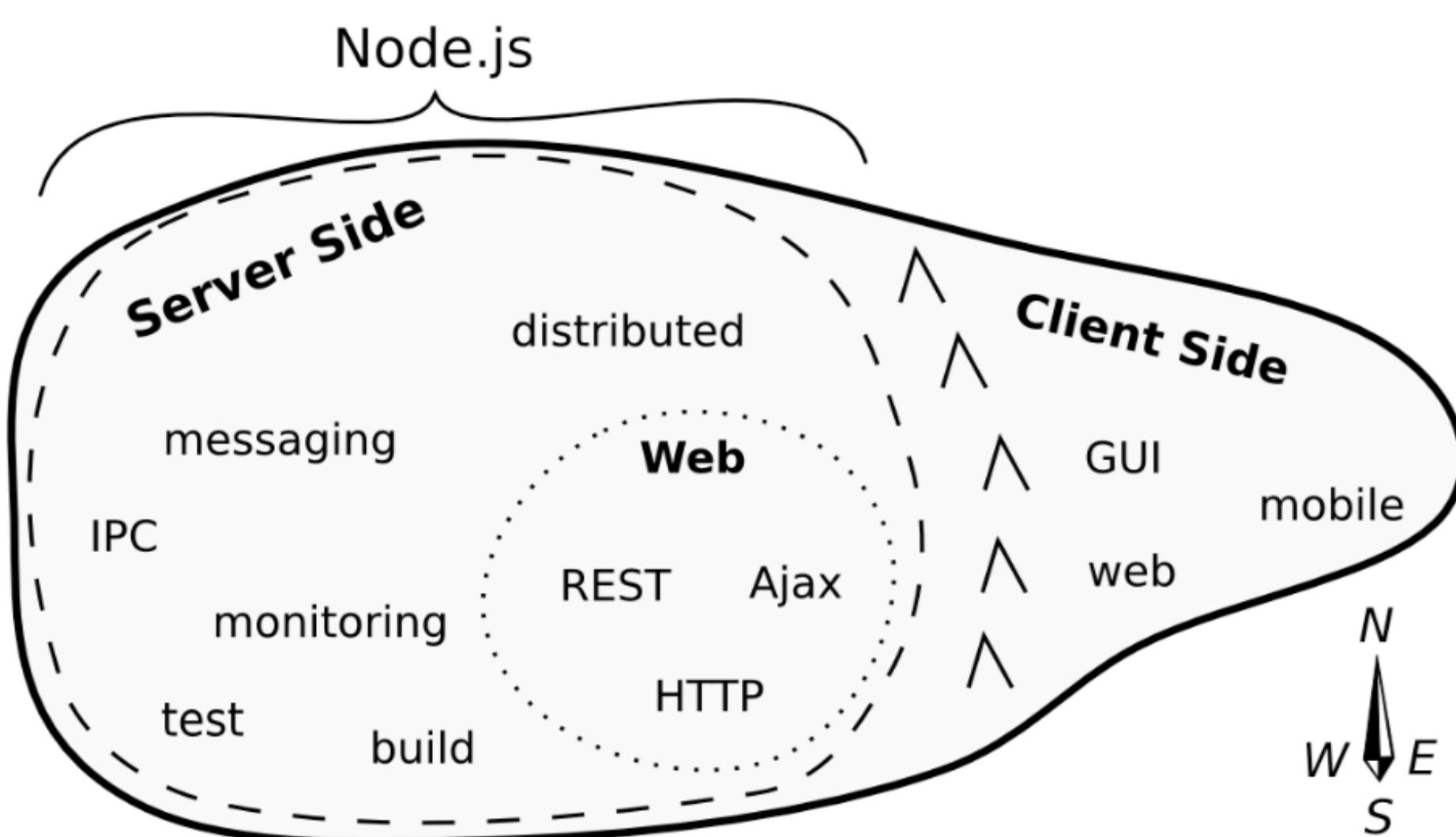
- REST + JSON APIs
- Single-page web app
- Real-time web app
- Rapid Prototyping
- I/O Processing

Node.js Bad Points

- CPU Intensive Job
- Multi-thread app
- Big Data
- Complex Business Logic

#02. Node.js's Phillosophy

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



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);
  });
});
```

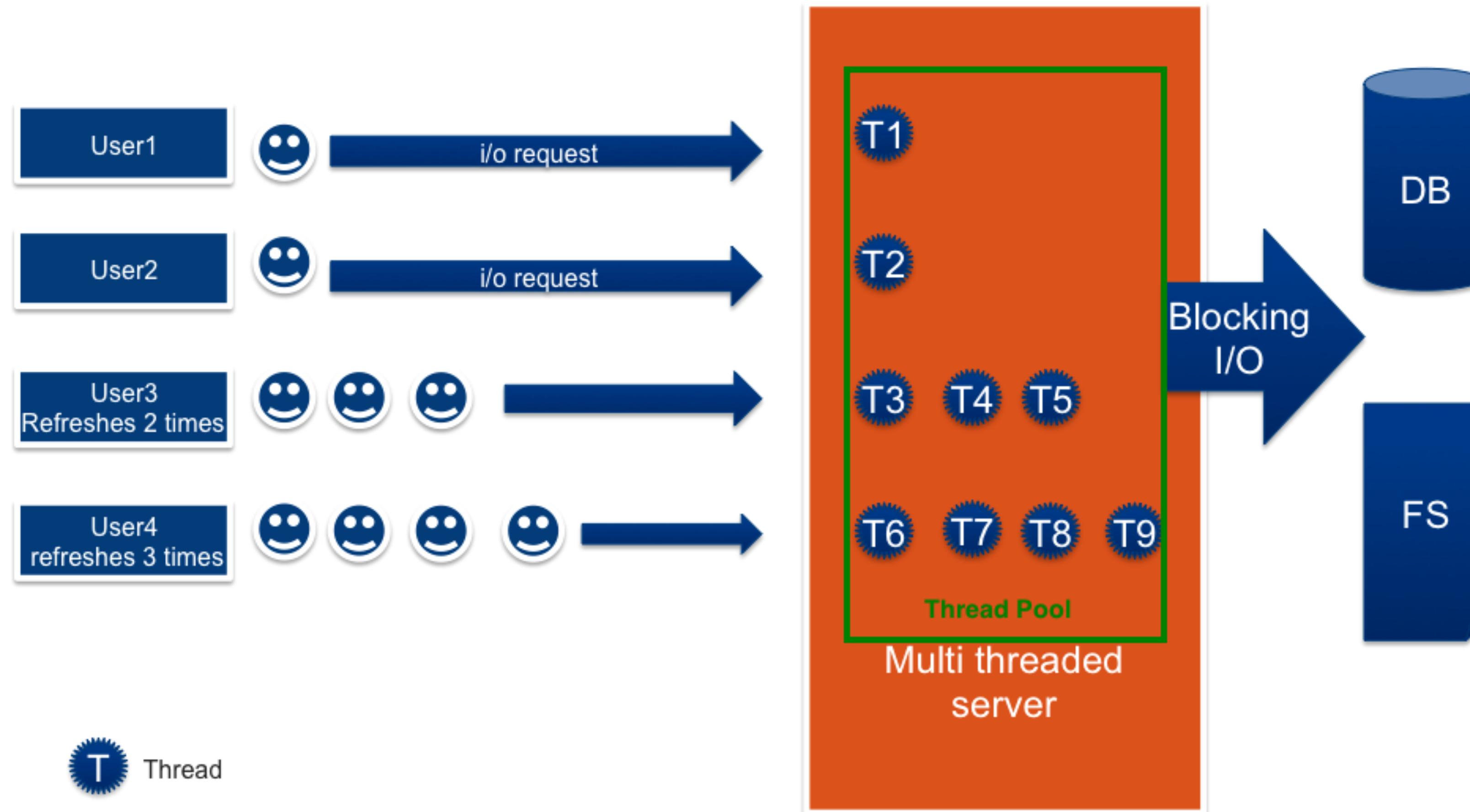
Asynchronous

(sync_db.js)

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

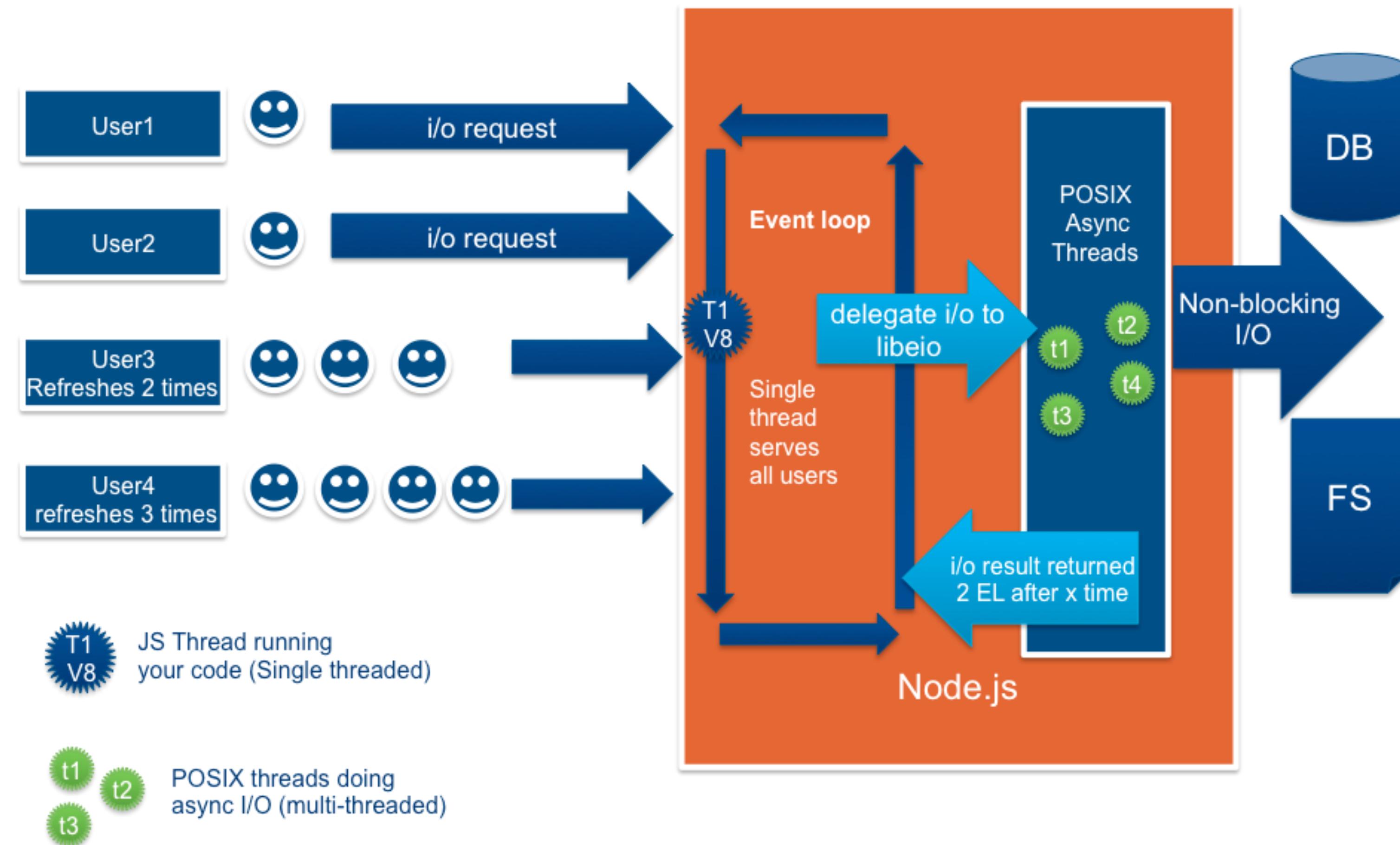
```
( async_db.js )
var db = new Database({host: 'localhost'});
db.connect(function(err, conn) {
  conn.find({name : 'pcw'}, function(err, results) {
    console.log(results);
  });
});
```

Blocking IO + Multi-Threaded Server

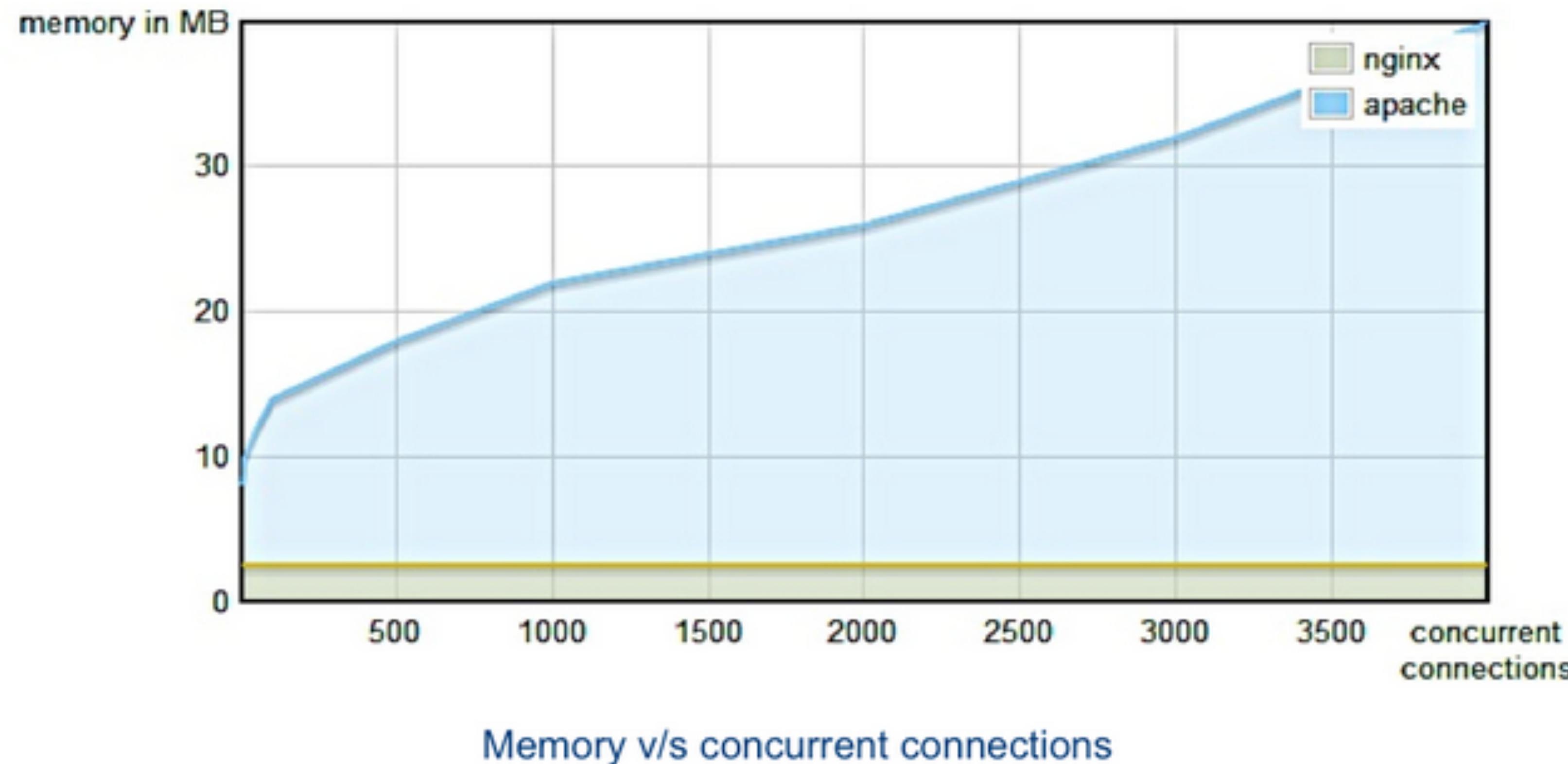


Blocking I/O + direct interface to clients leads to thread-pools w/ larger number threads and more memory requirements

Async-IO + Event based Server

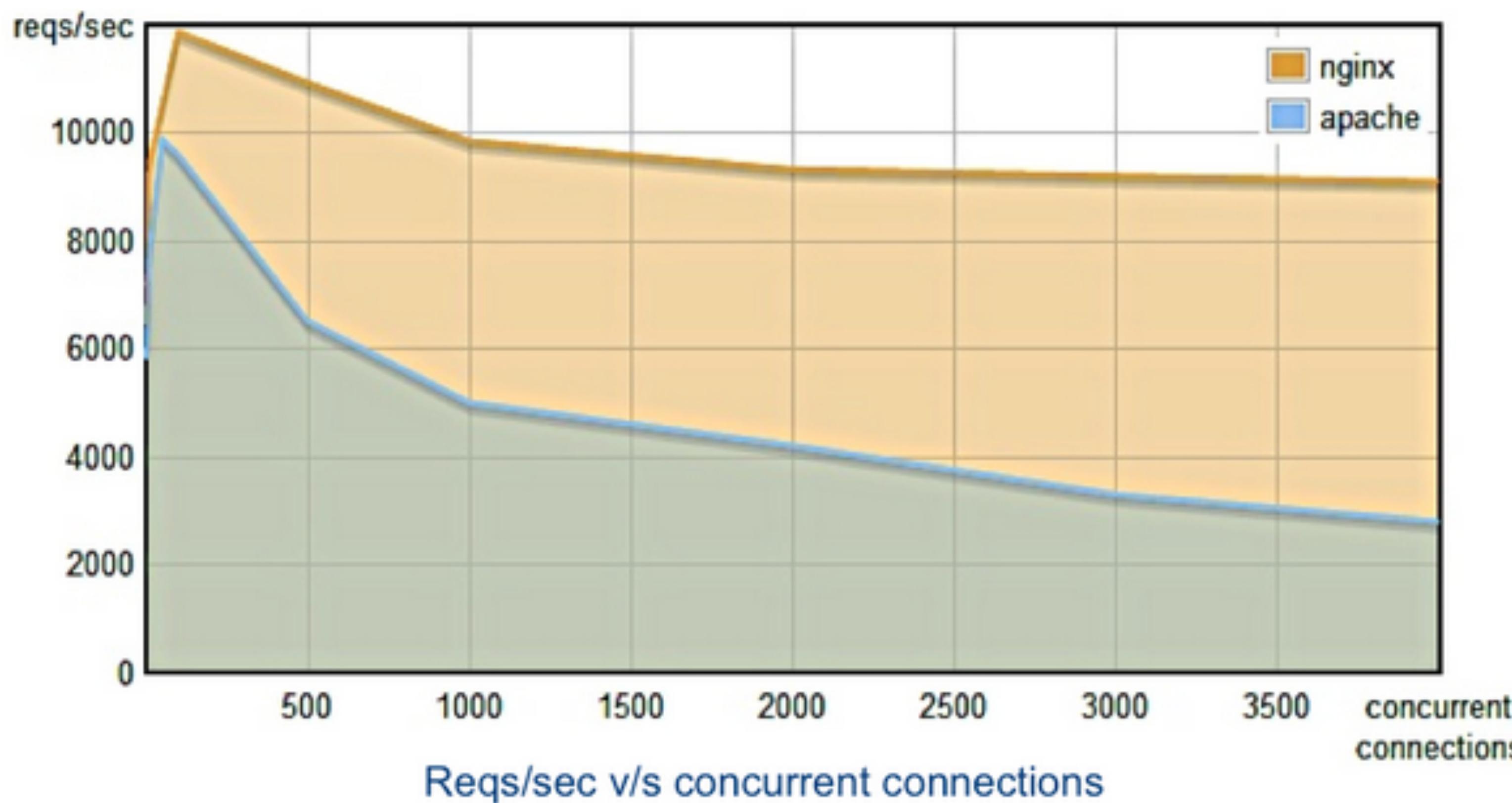


Memory Usage : Nginx vs Apache



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

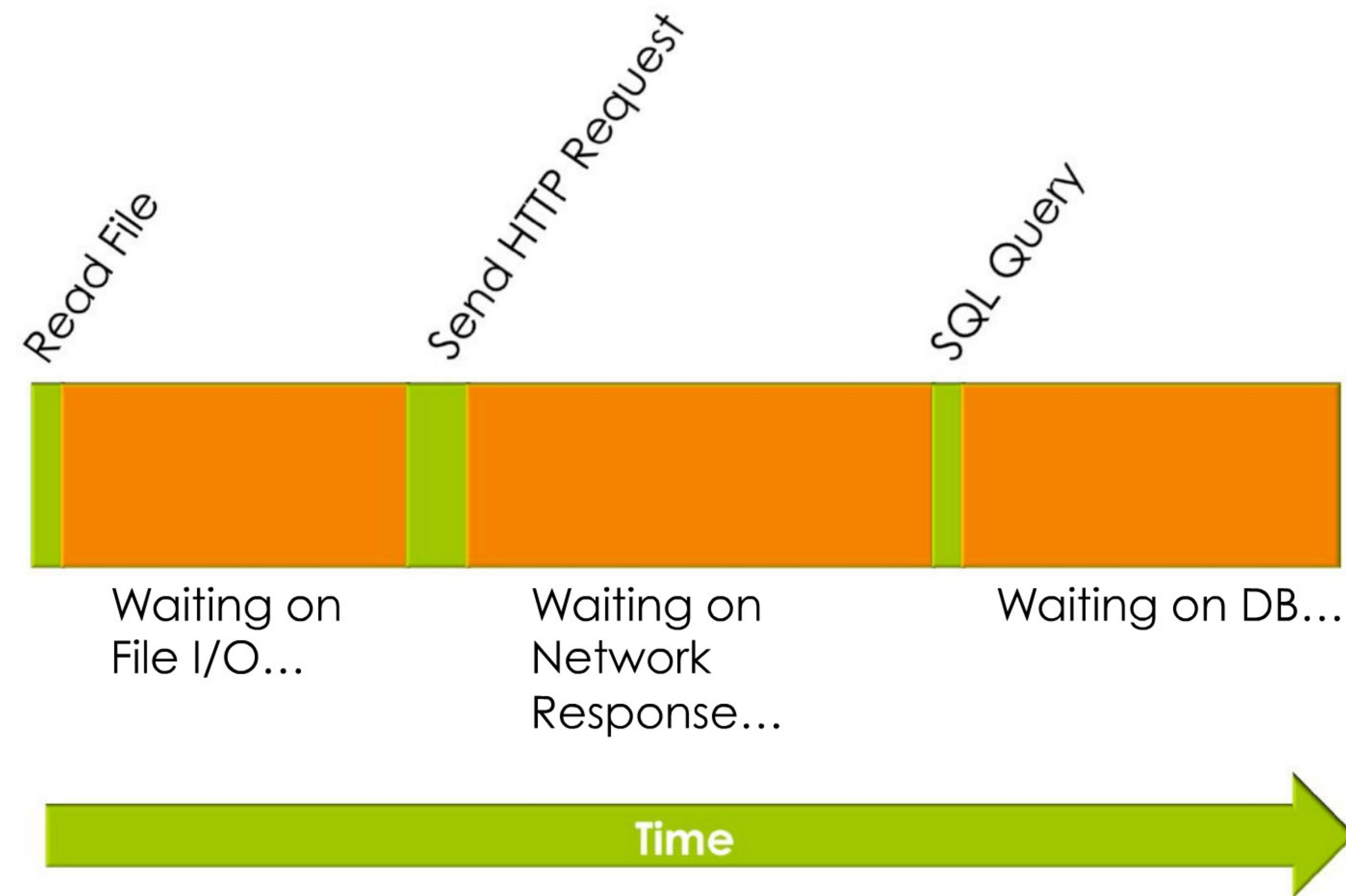
Performance : Nginx vs Apache



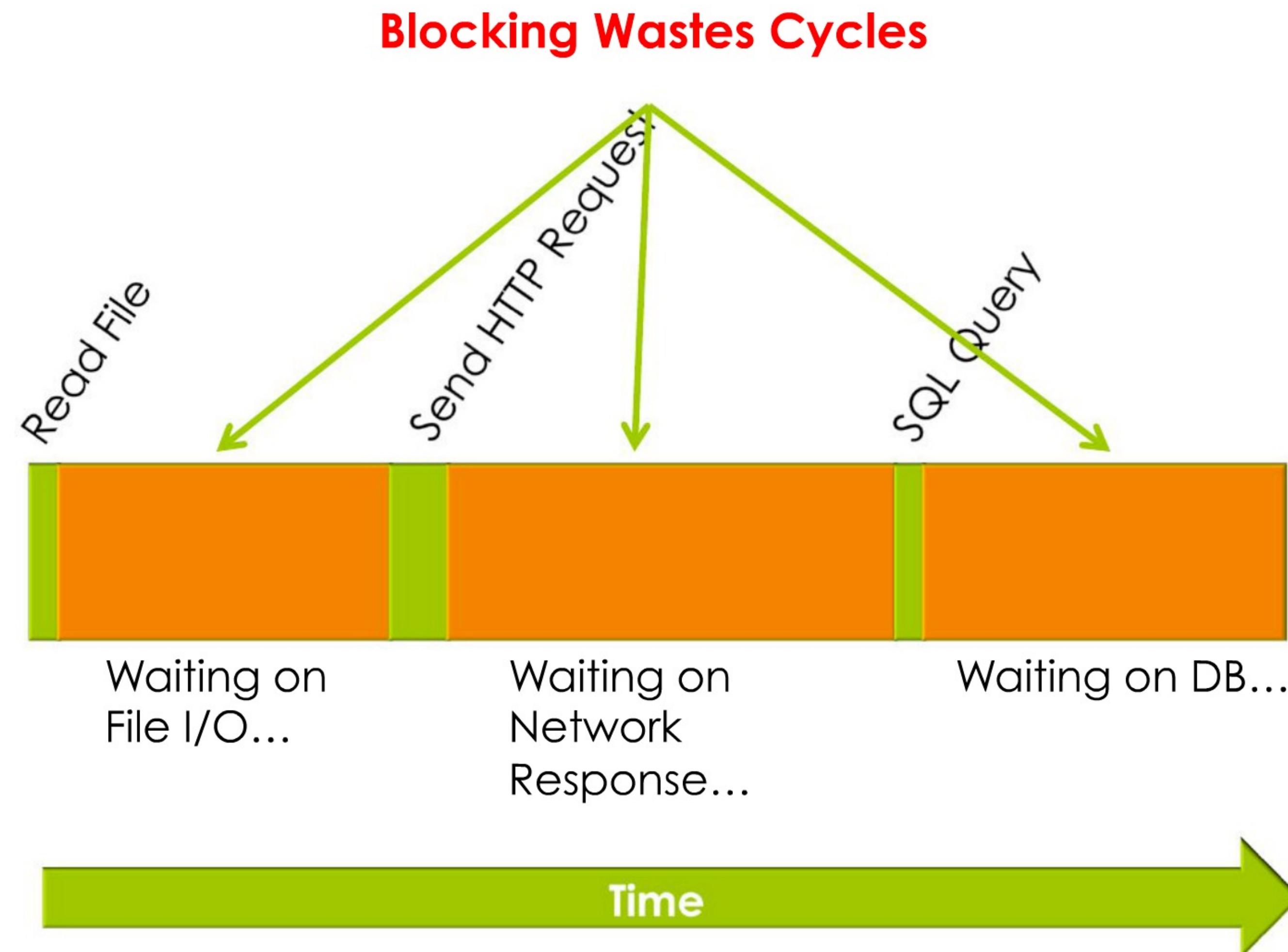
At ~4000 concurrent connections,
- Nginx can serve ~9000 reqs/sec
- Apache can serve ~3000 reqs/sec

#04. Traditional Thread vs Eventloop

One Worker's Lifecycle



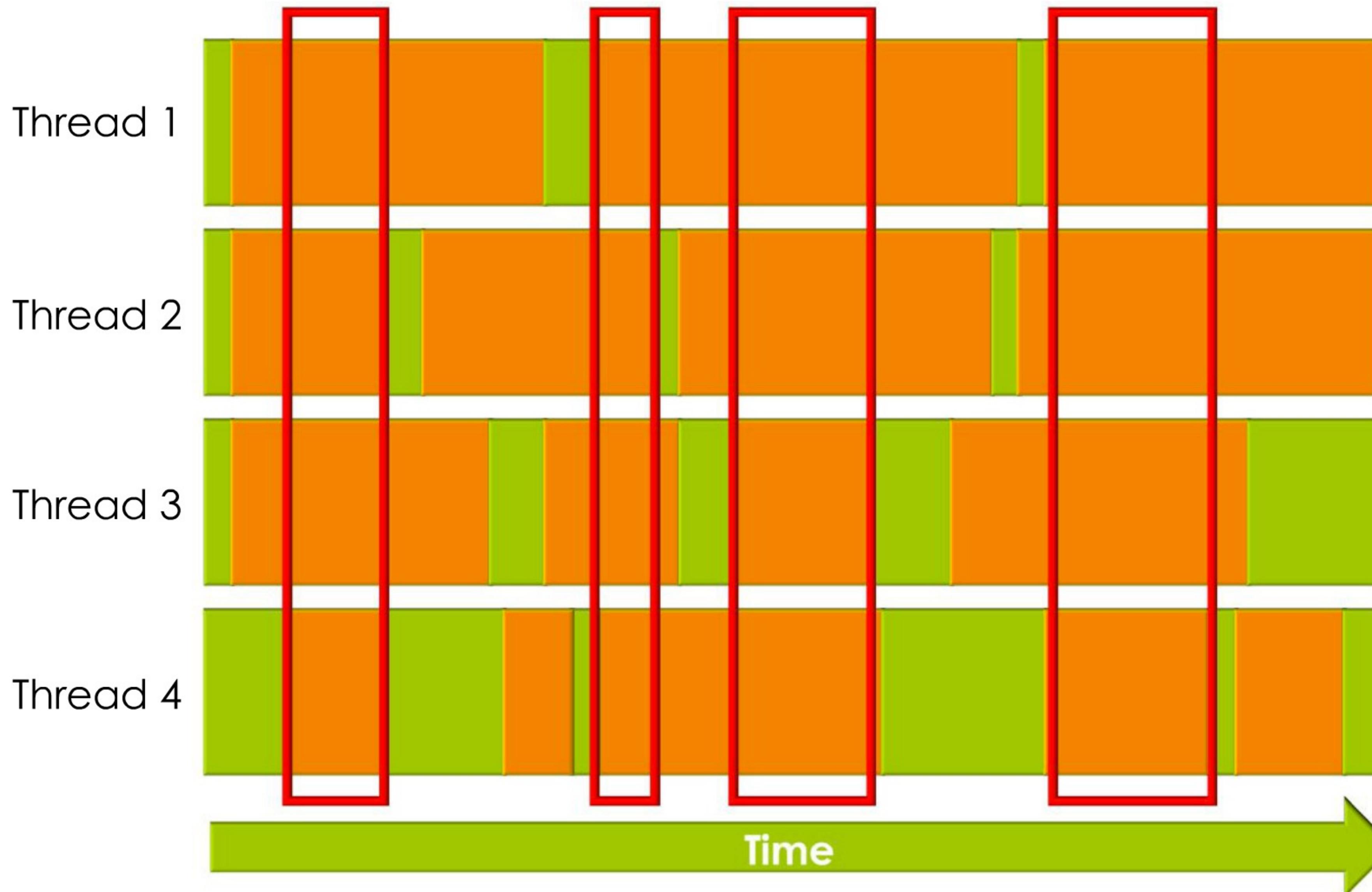
One Worker's Lifecycle



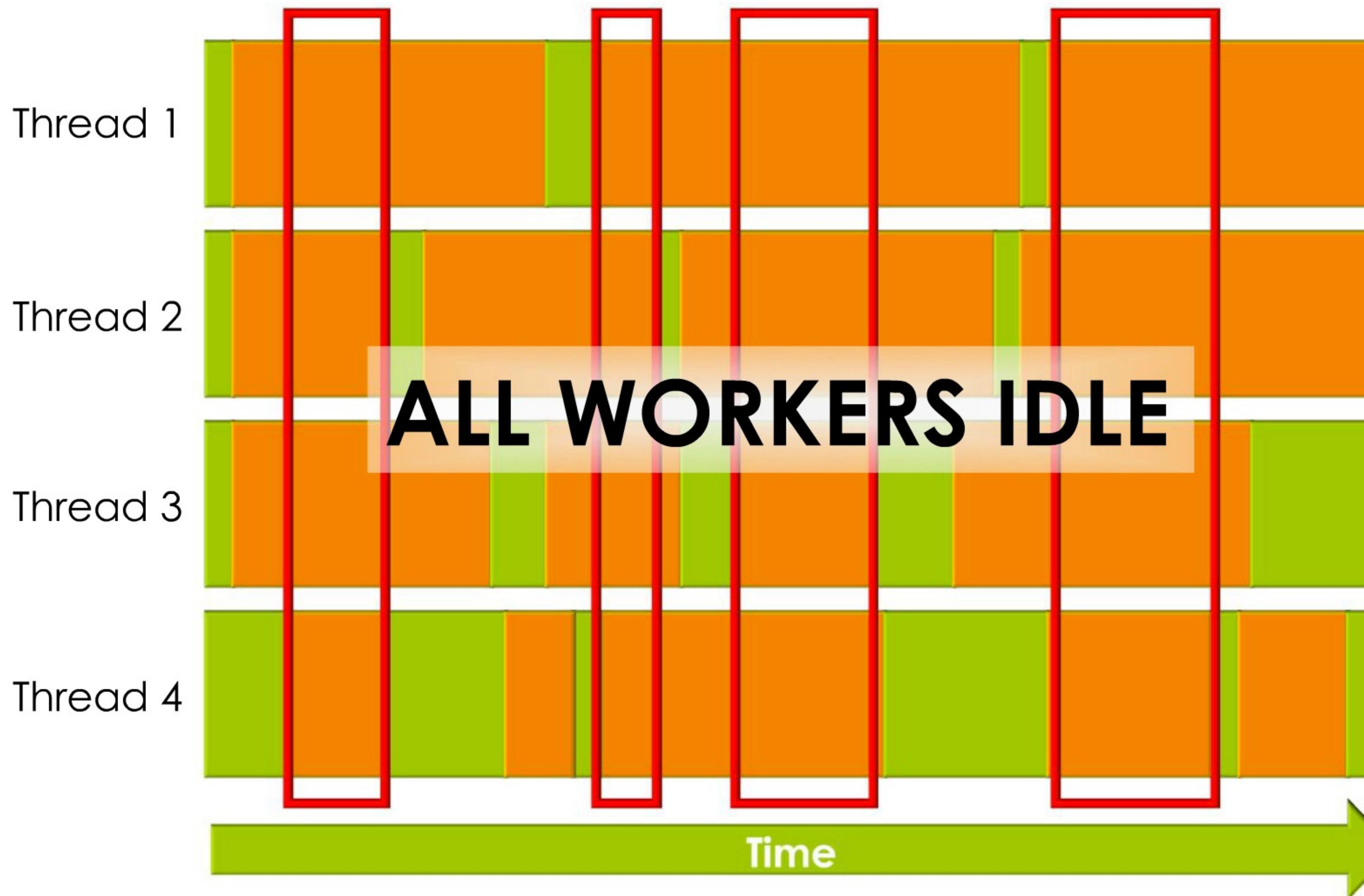
Multiple Workers..



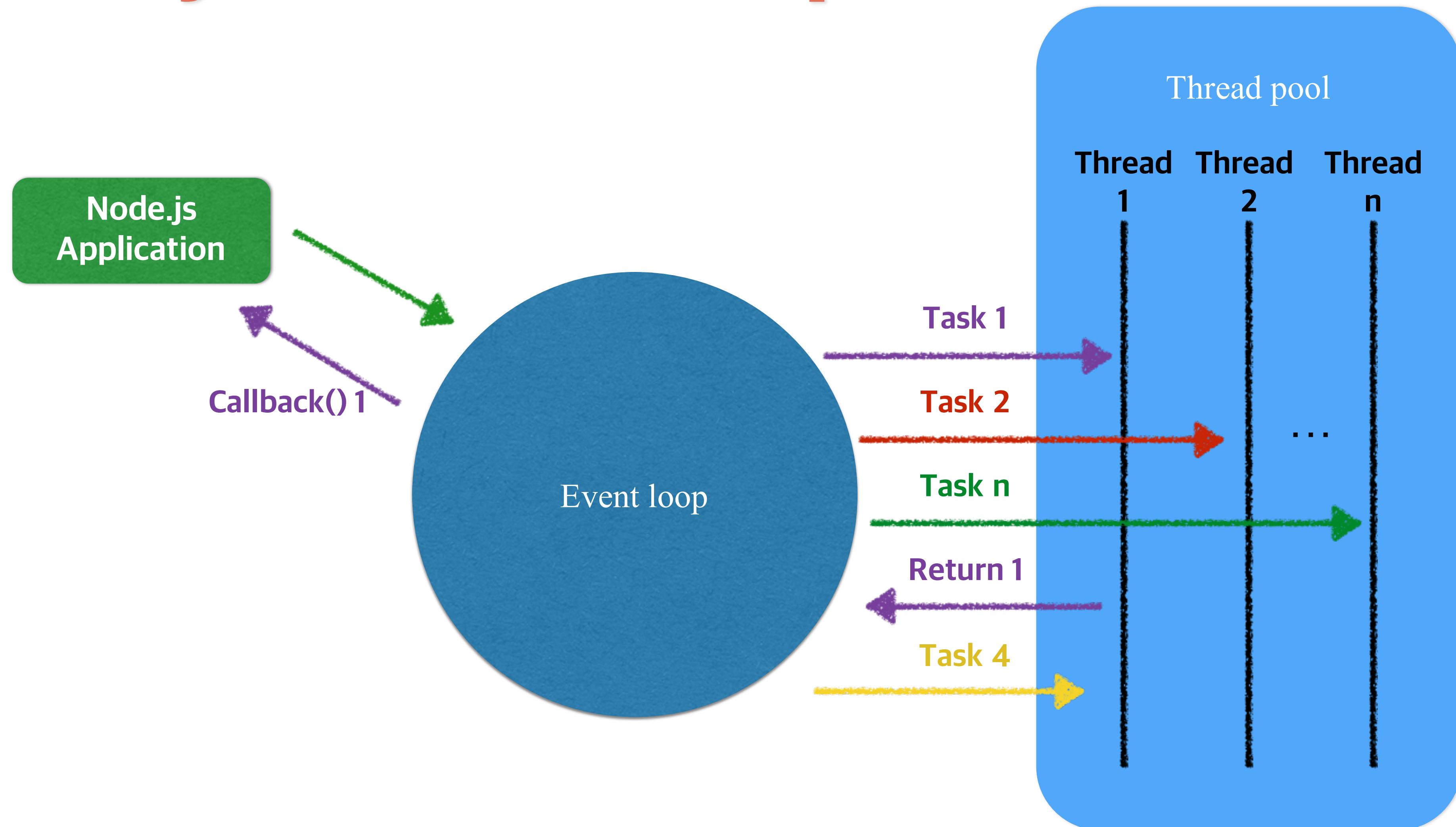
Multiple Workers..



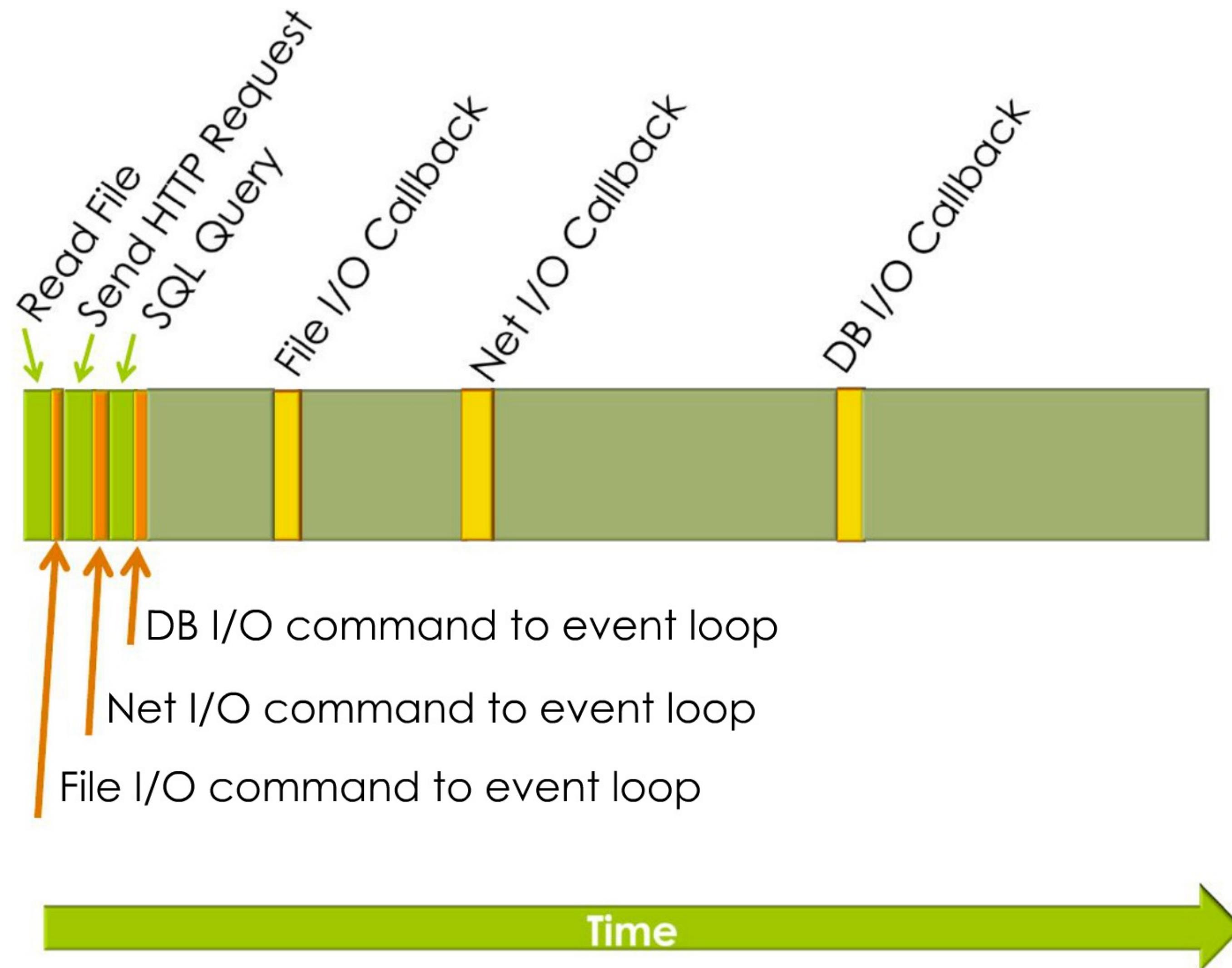
Multiple Workers..



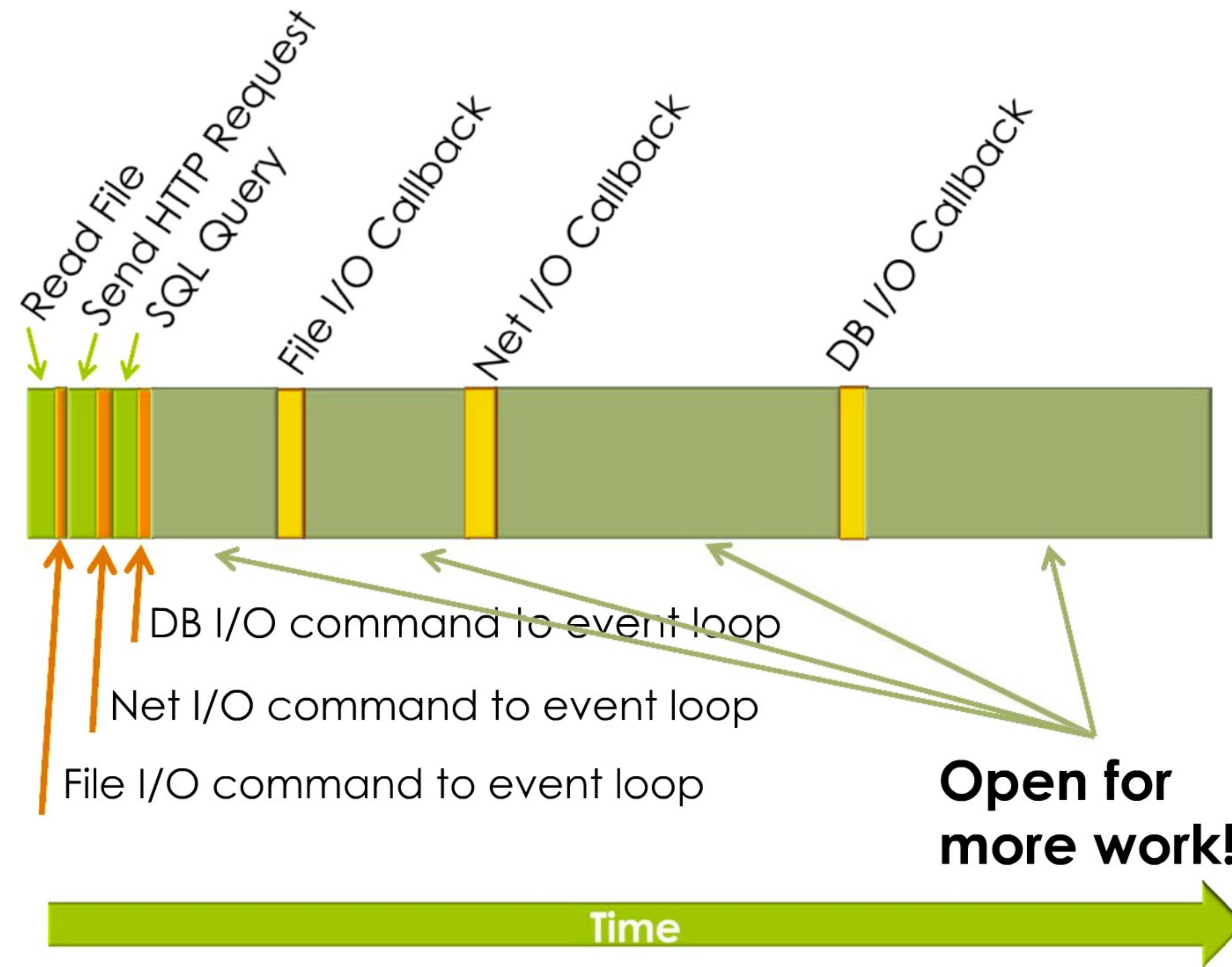
Node.js Event Loop



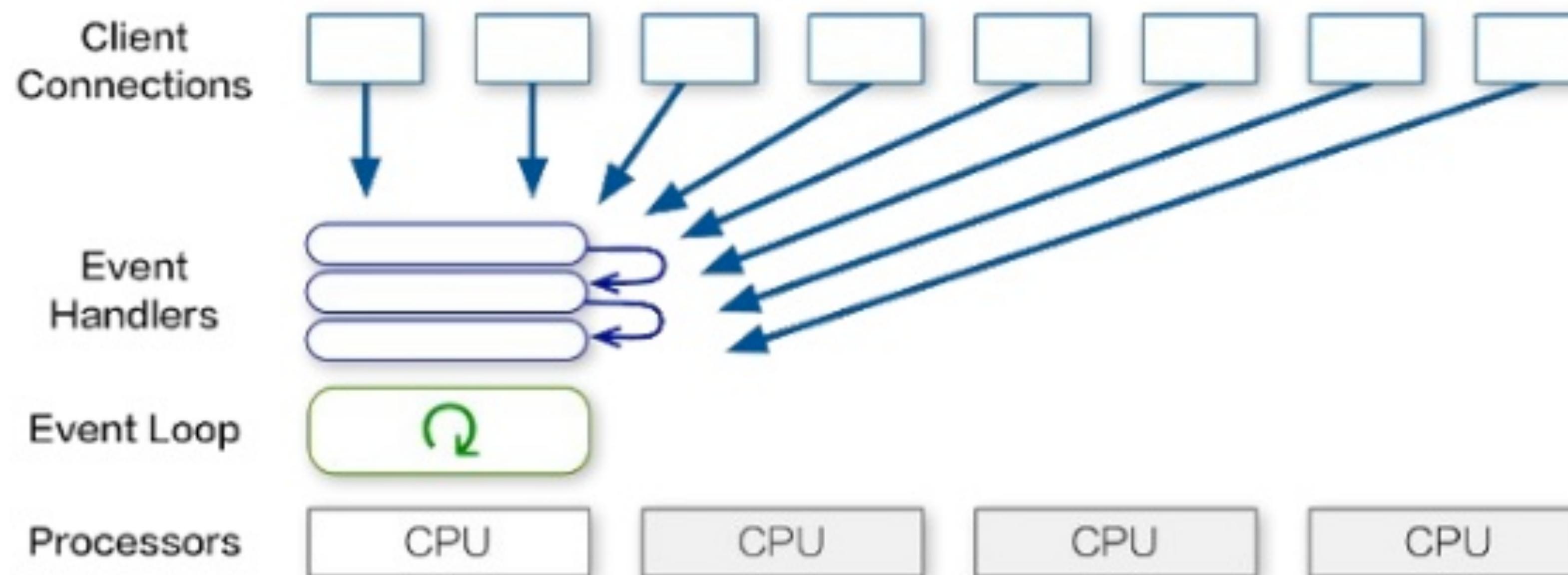
Node.js Event Loop



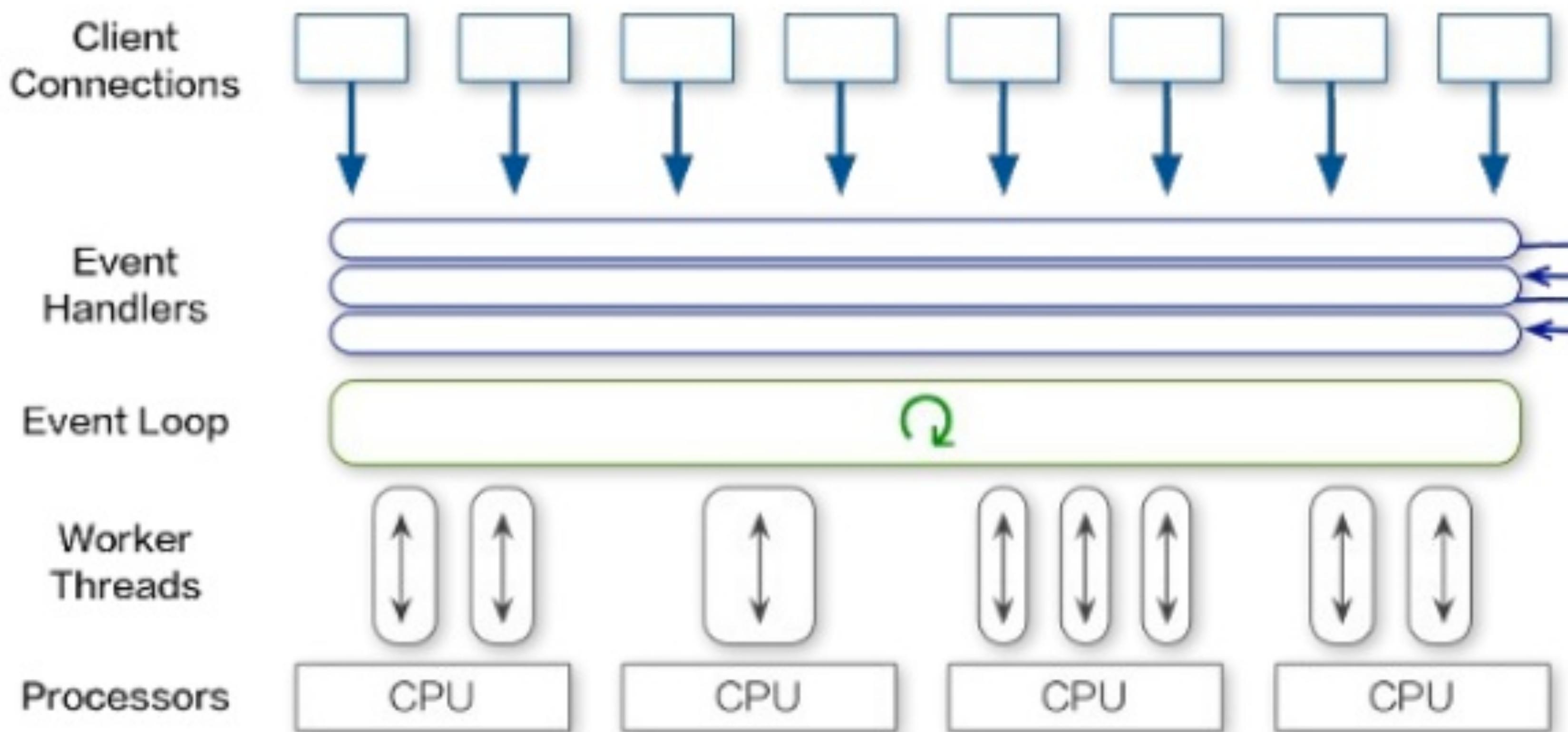
Node.js Event Loop do More Work!!



Single Thread??

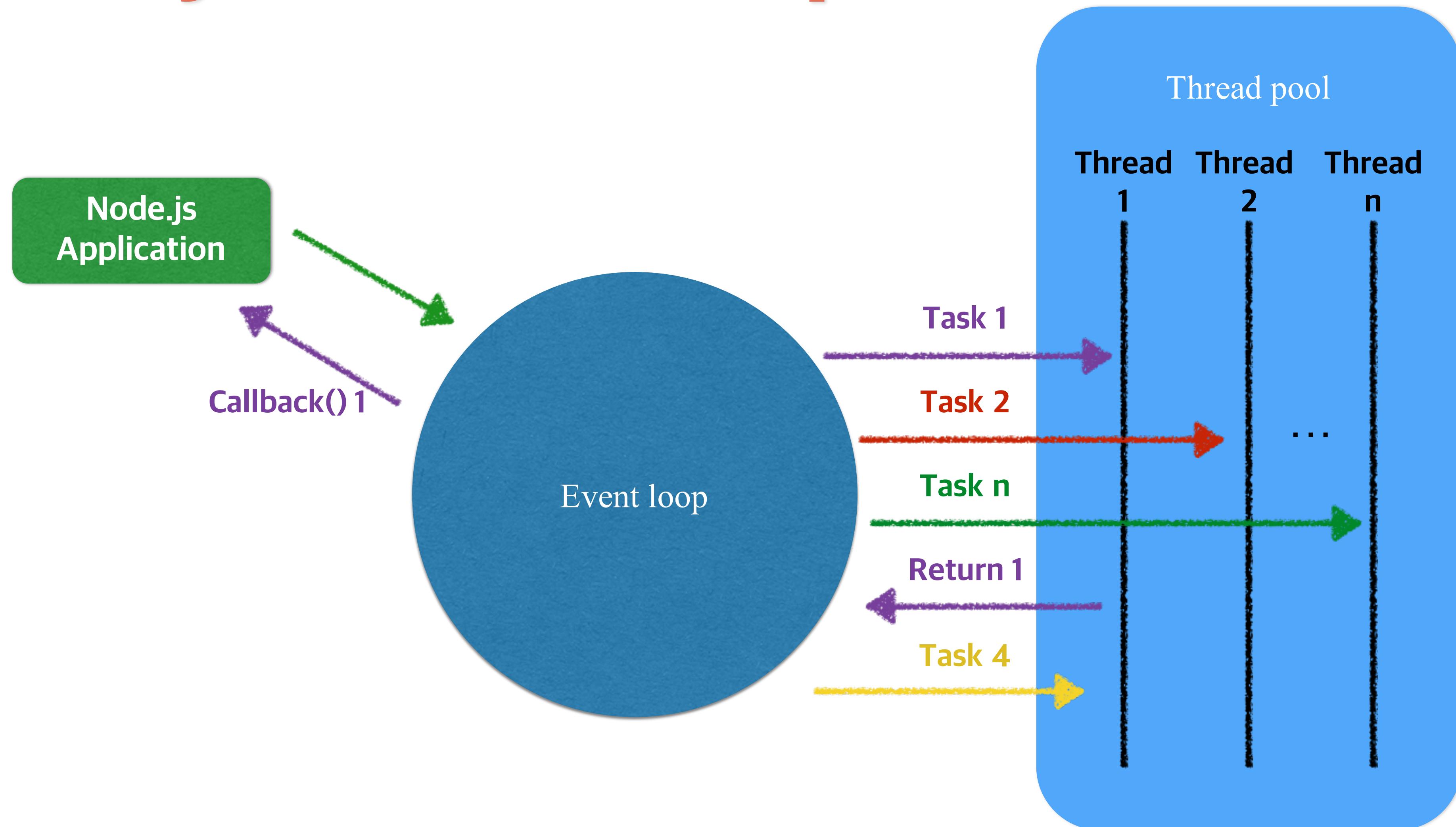


Multi-Thread without locking.

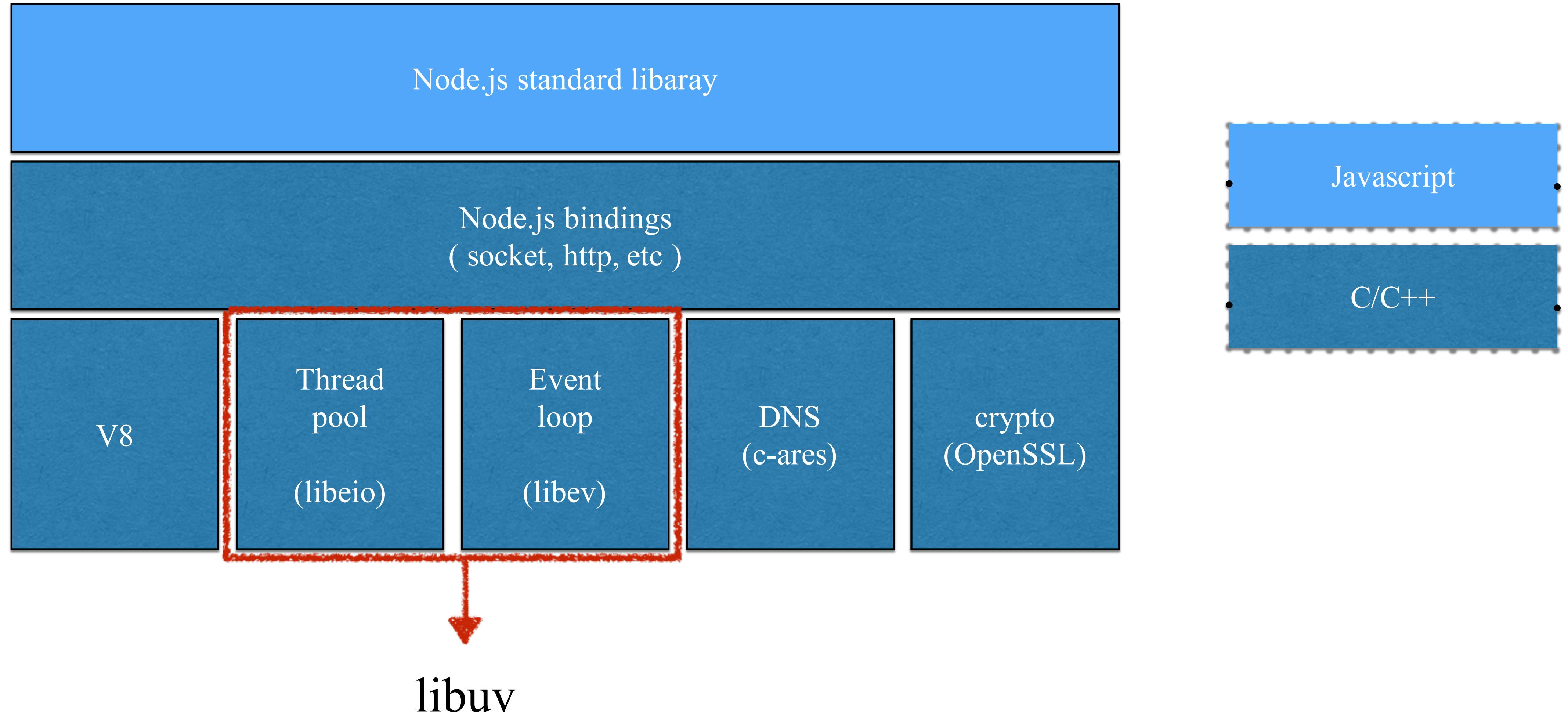


#05. Node.JS Architecture

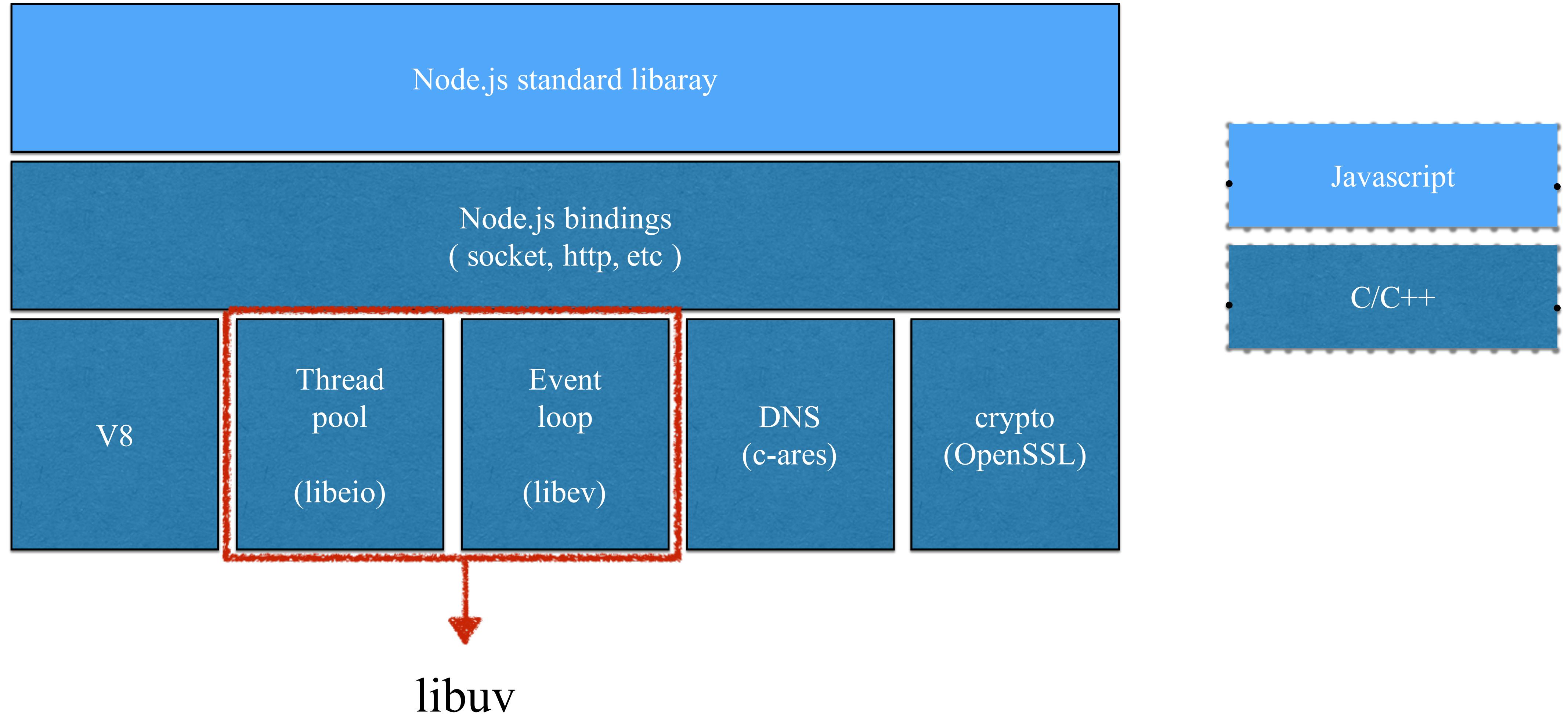
Node.js Event Loop



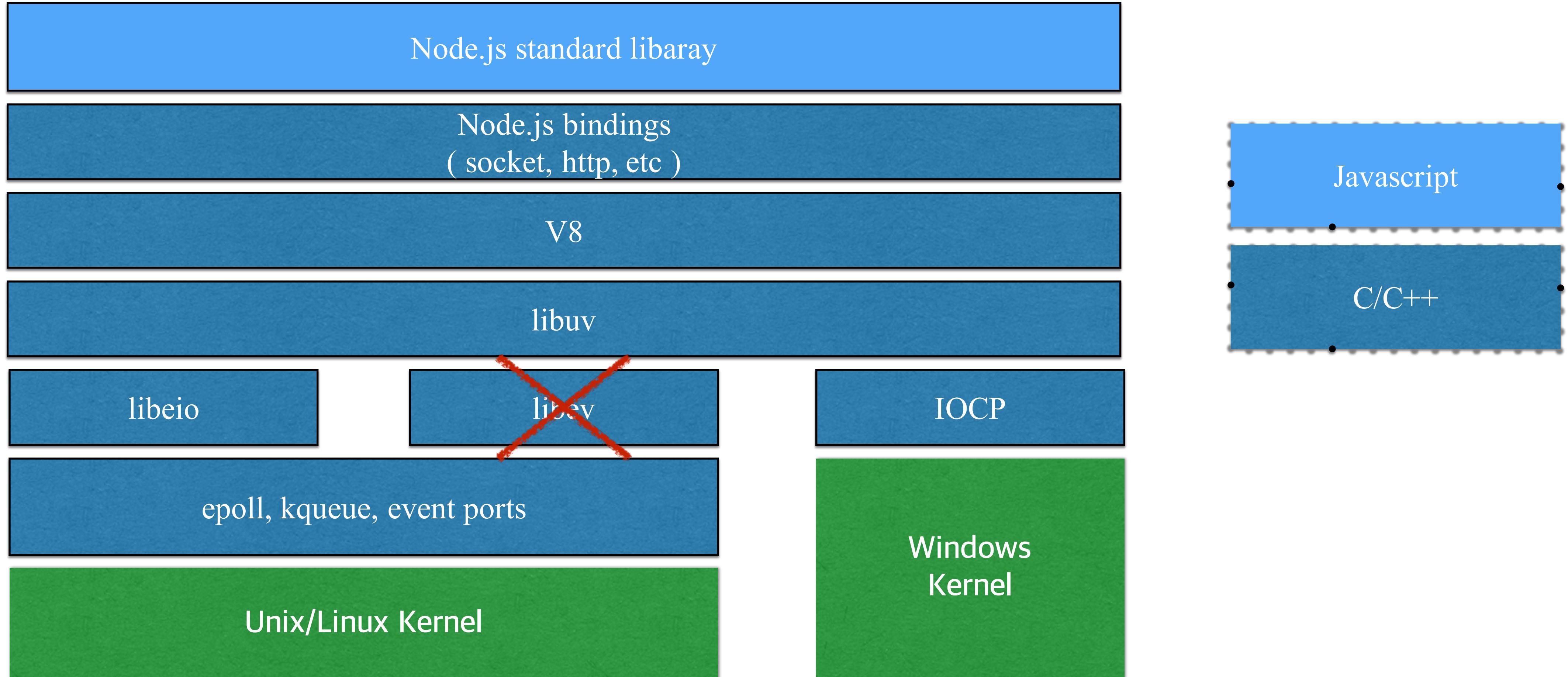
Node.js - Architecture



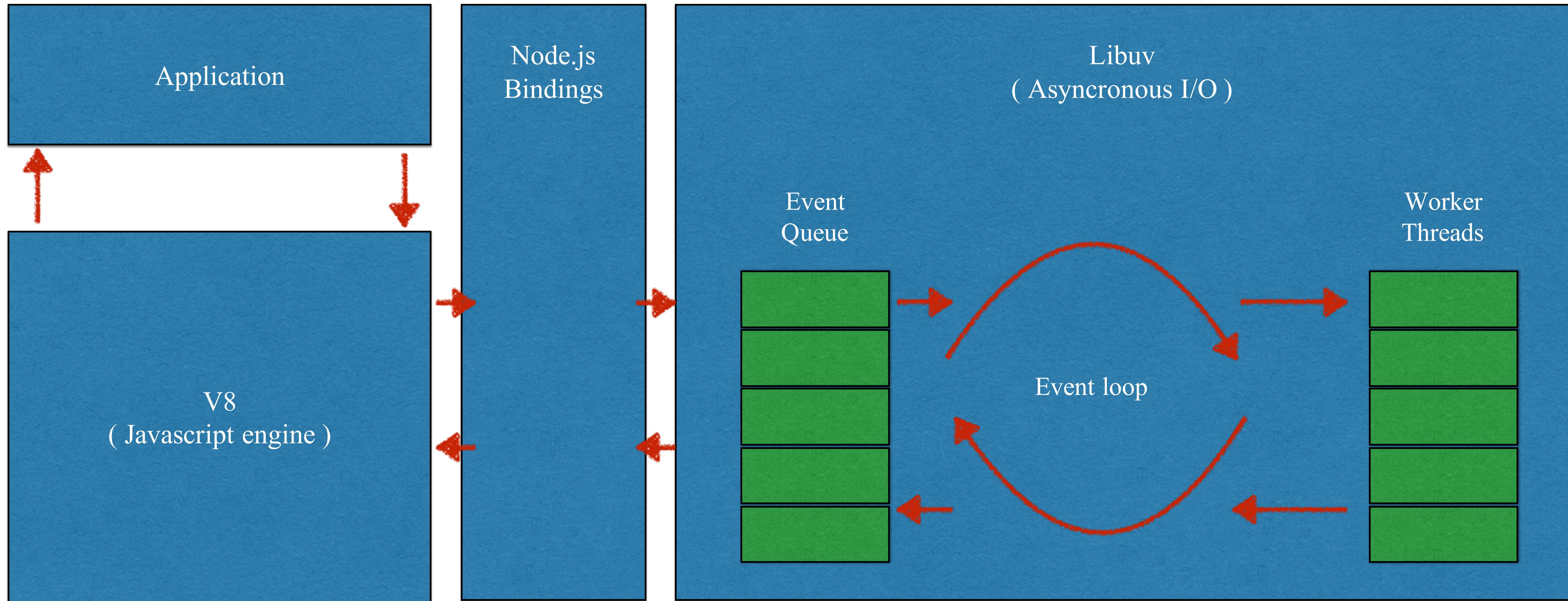
Node.js - Architecture



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; }

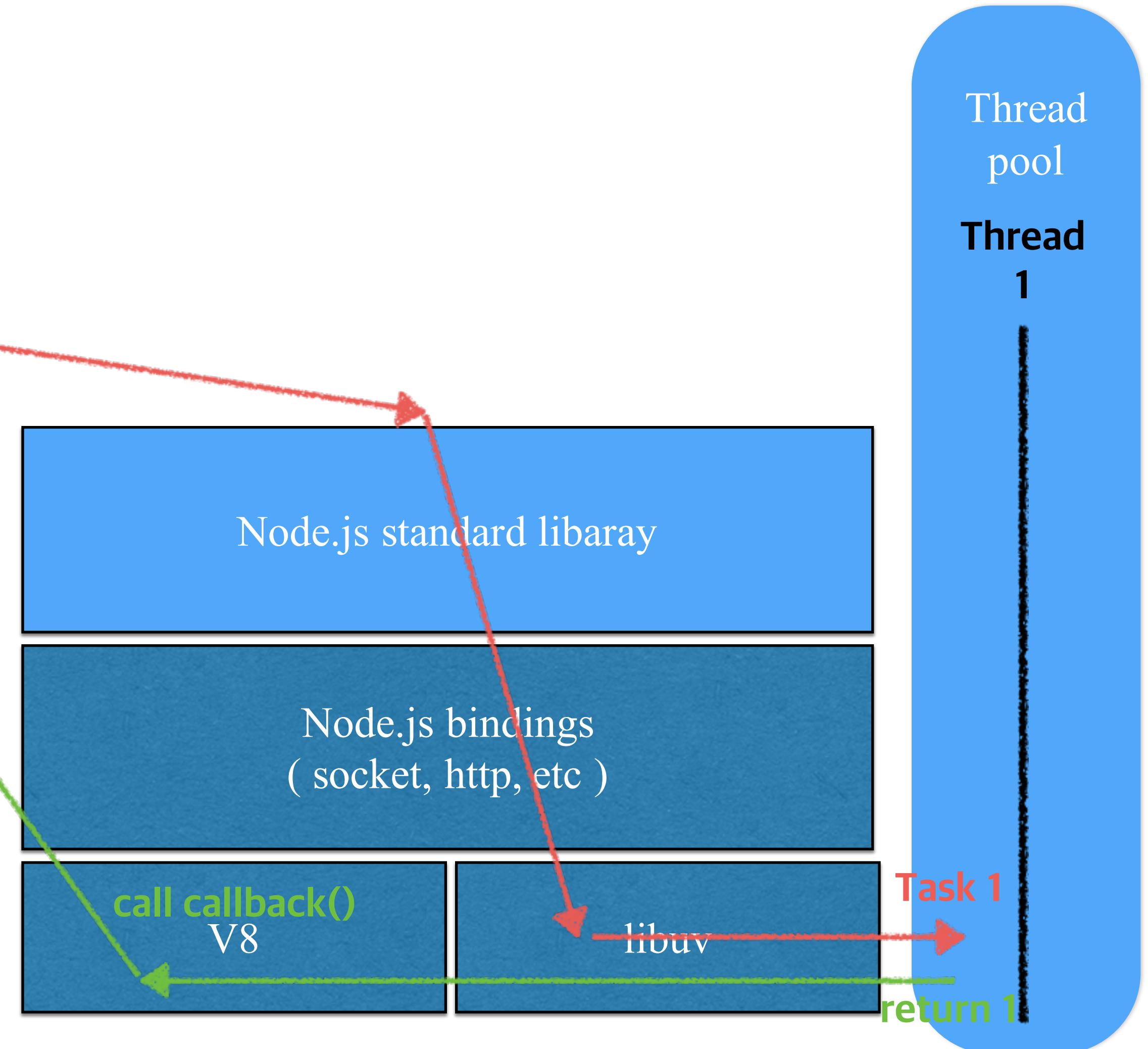
  console.log(data);
});
```

Node.js Event Loop

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

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

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



Part 2.

Node.JS Workshop

Install Node.js

(Node.js, NVM, NPM)

Node.js Workshop#0

(hello nodejs)

Node.js WS #0

```
$ mkdir nodestudy
```

```
$ cd nodestudy
```

Node.js WS#0

(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 WS#0

(helloConsole.js)

```
$ node helloConsole
```

Node.js WS#0

(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 WS#0

(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 WS#0

(helloProcess.js)

```
$ node helloProcess
```

Node.js WS#0

(helloProcess.js)

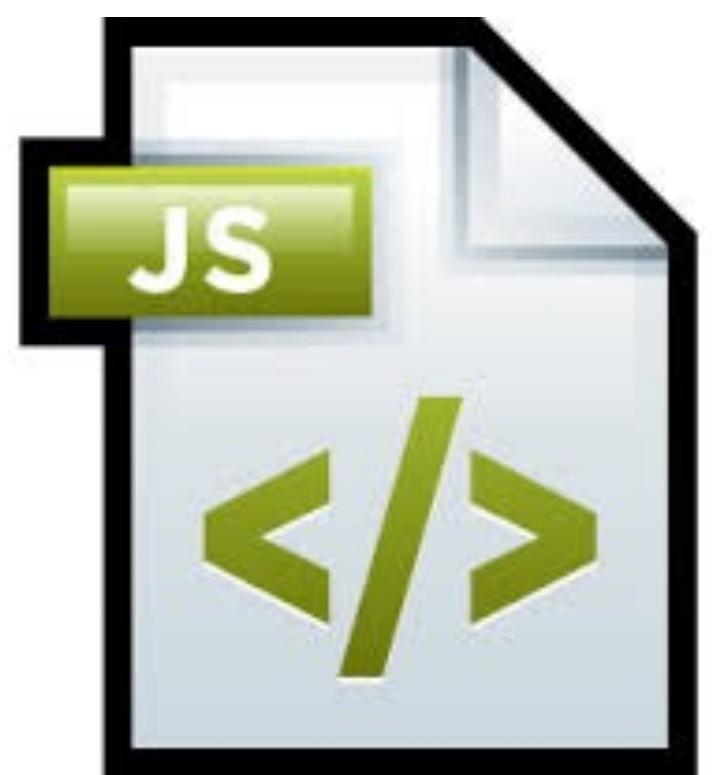
```
console.log('env = ', process.env);
console.log('version = ', process.version);
console.log('versions = ', process.versions);
console.log('platform = ', process.platform);
console.log('memory usage = ', process.memoryUsage());
console.log('up time = ', process.uptime());
```

Node.js Workshop #1

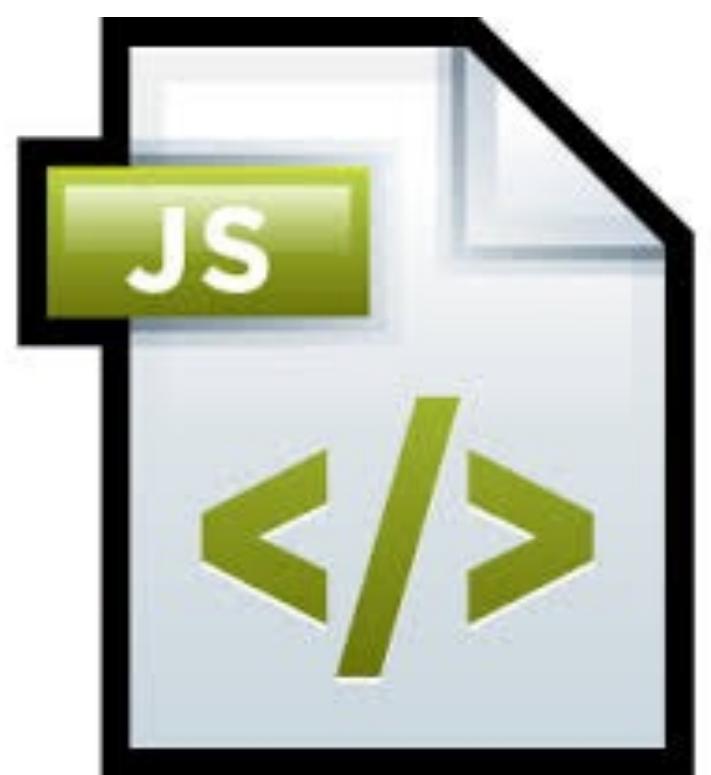
(Handling Modules)

Node.js WS #1

(module system)



http



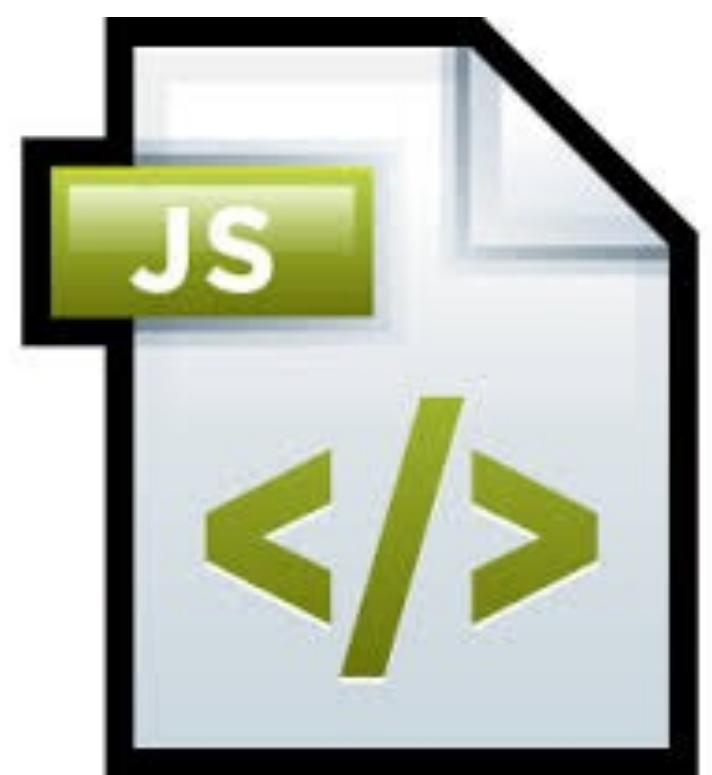
Stream



net



fs



EventEmitter

...

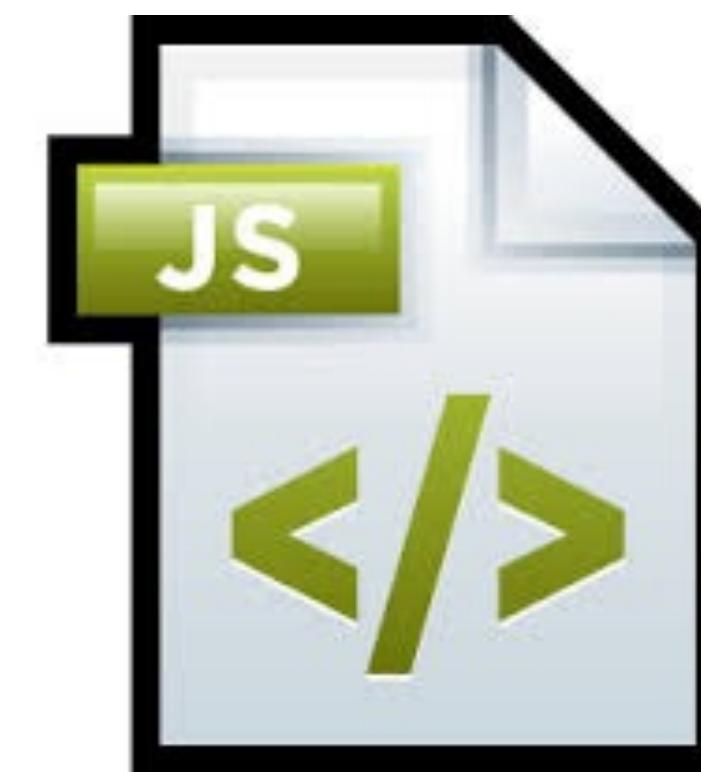
Node.js WS #1

(module system)



app.js

`require('math')`



math

`exports.abs`
`exports.pow`

Node.js WS #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;
};
```

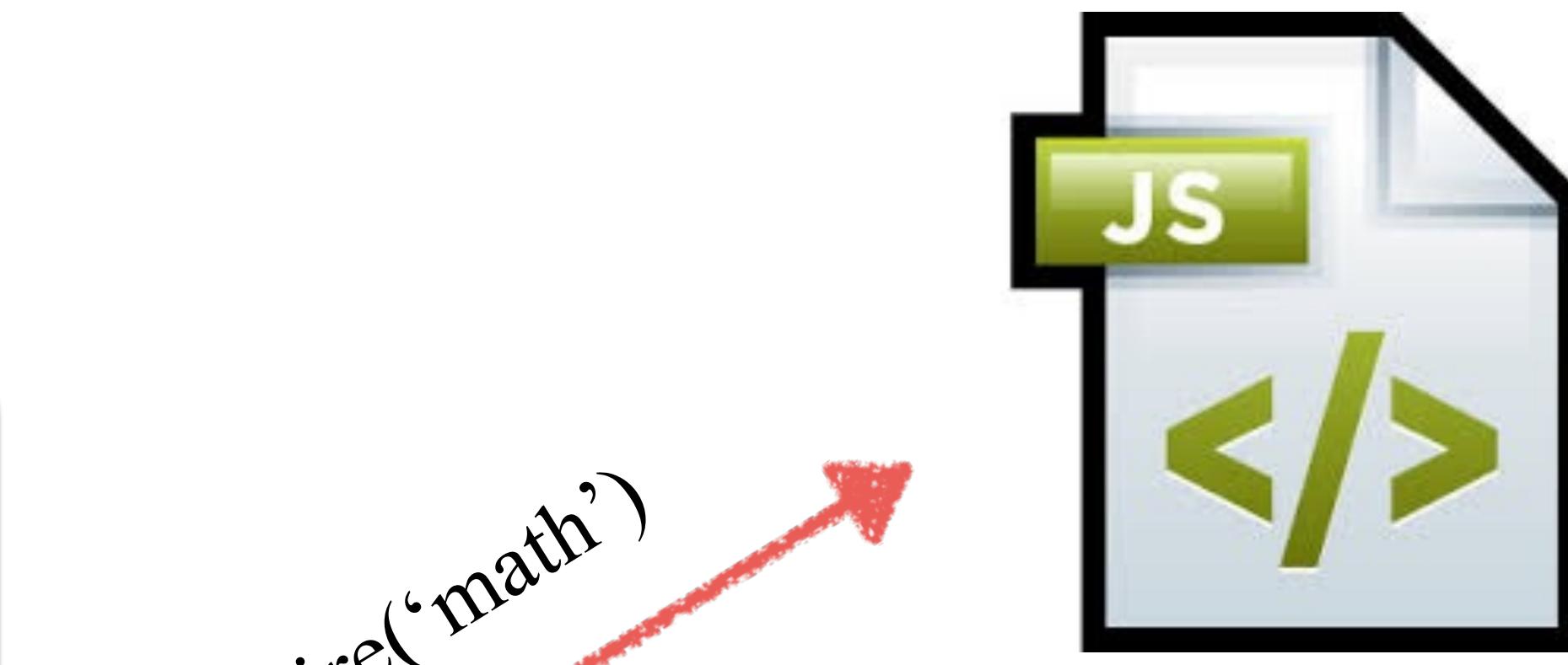
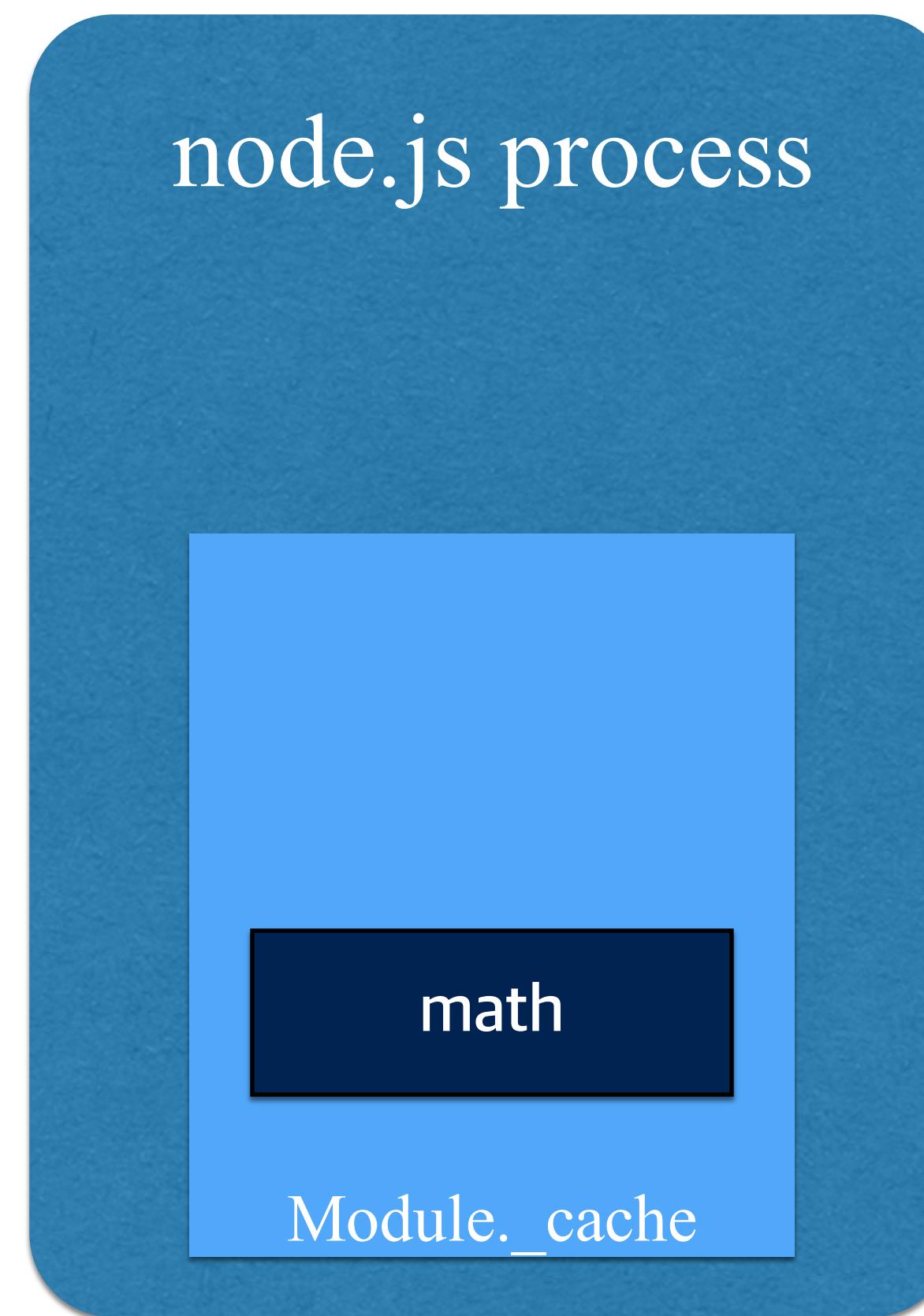
(app.js)

```
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 WS #1

(module system - caching)



require('math')



math



math

Node.js WS #1

(npm)

How to use npm

Node.js WS #1

(npm)

```
npm init
```

Node.js WS #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 WS #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 WS #1

(npm)

Install colors module

Node.js WS #1

(npm)

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

Node.js WS #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 WS #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 WS #1

(EventEmitter)

EventEmitter?

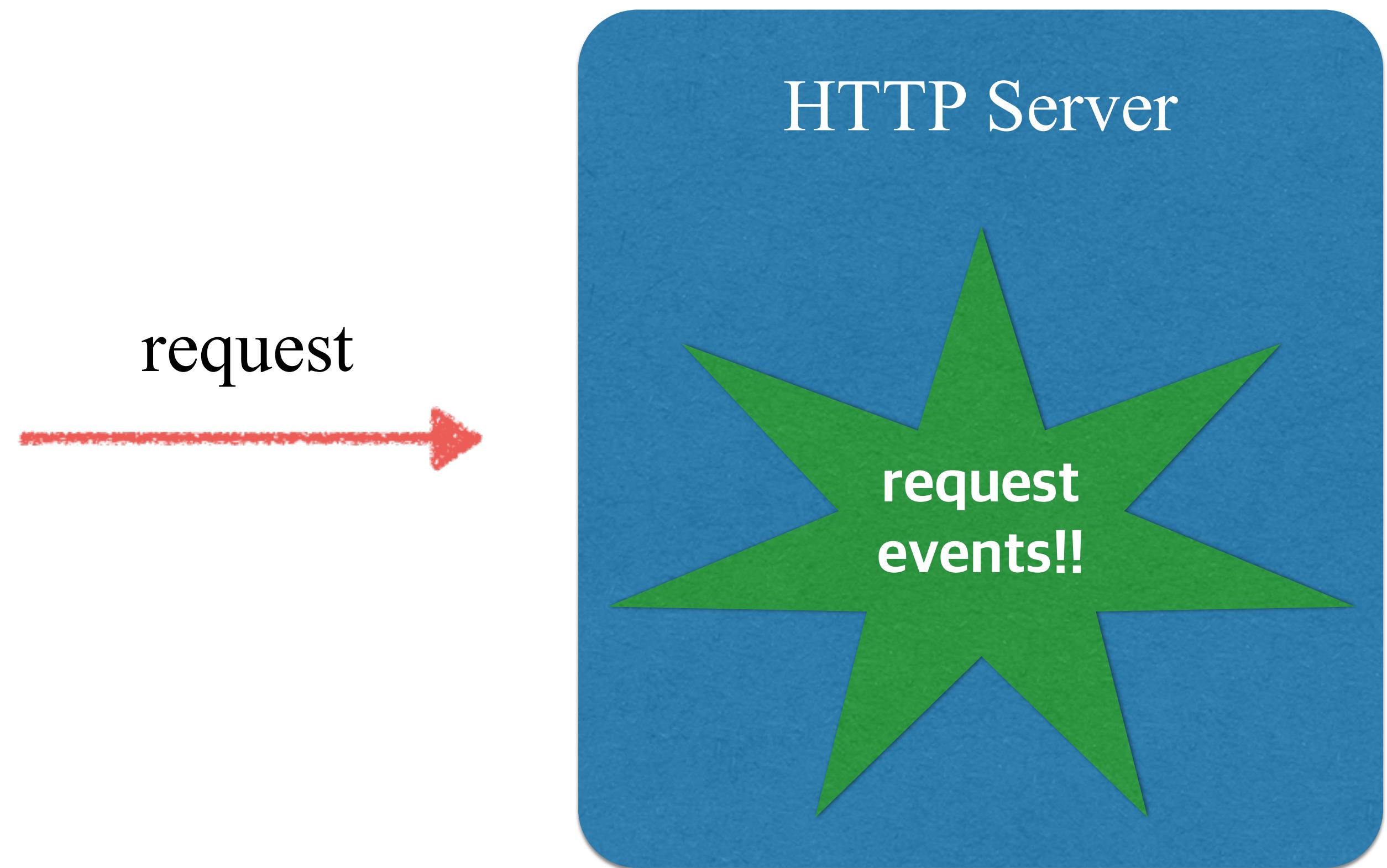
Many objects in Node emit events: a net.Server emits an event each time a peer connects to it, a fs.readStream emits an event when the file is opened. **All objects which emit events are instances of events.EventEmitter.** You can access this module by doing: require("events");

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.

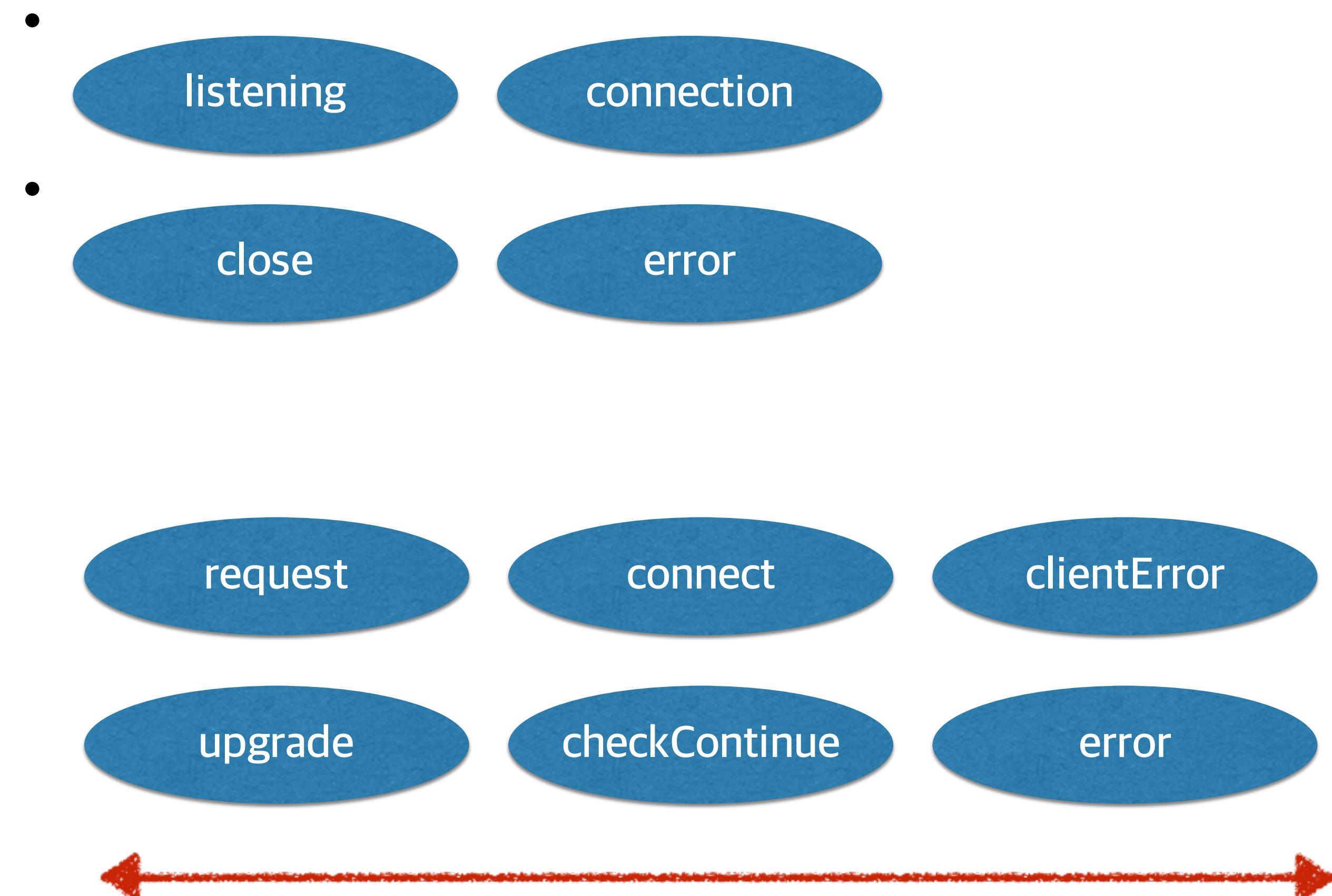
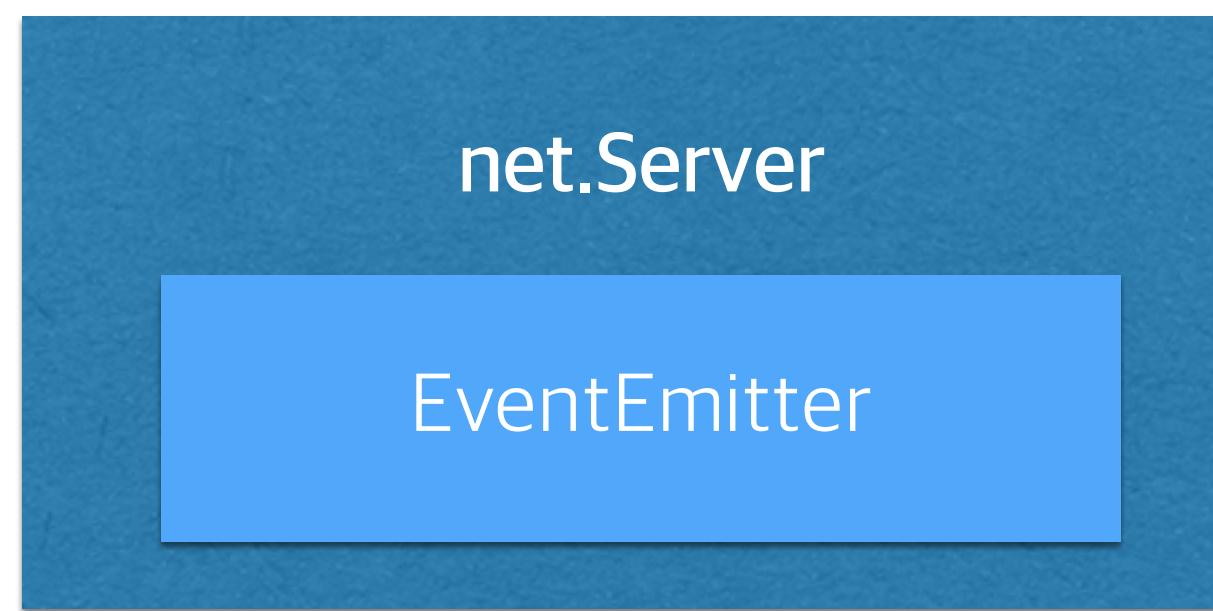
Node.js WS #1

(EventEmitter)



Node.js WS #1

(EventEmitter)

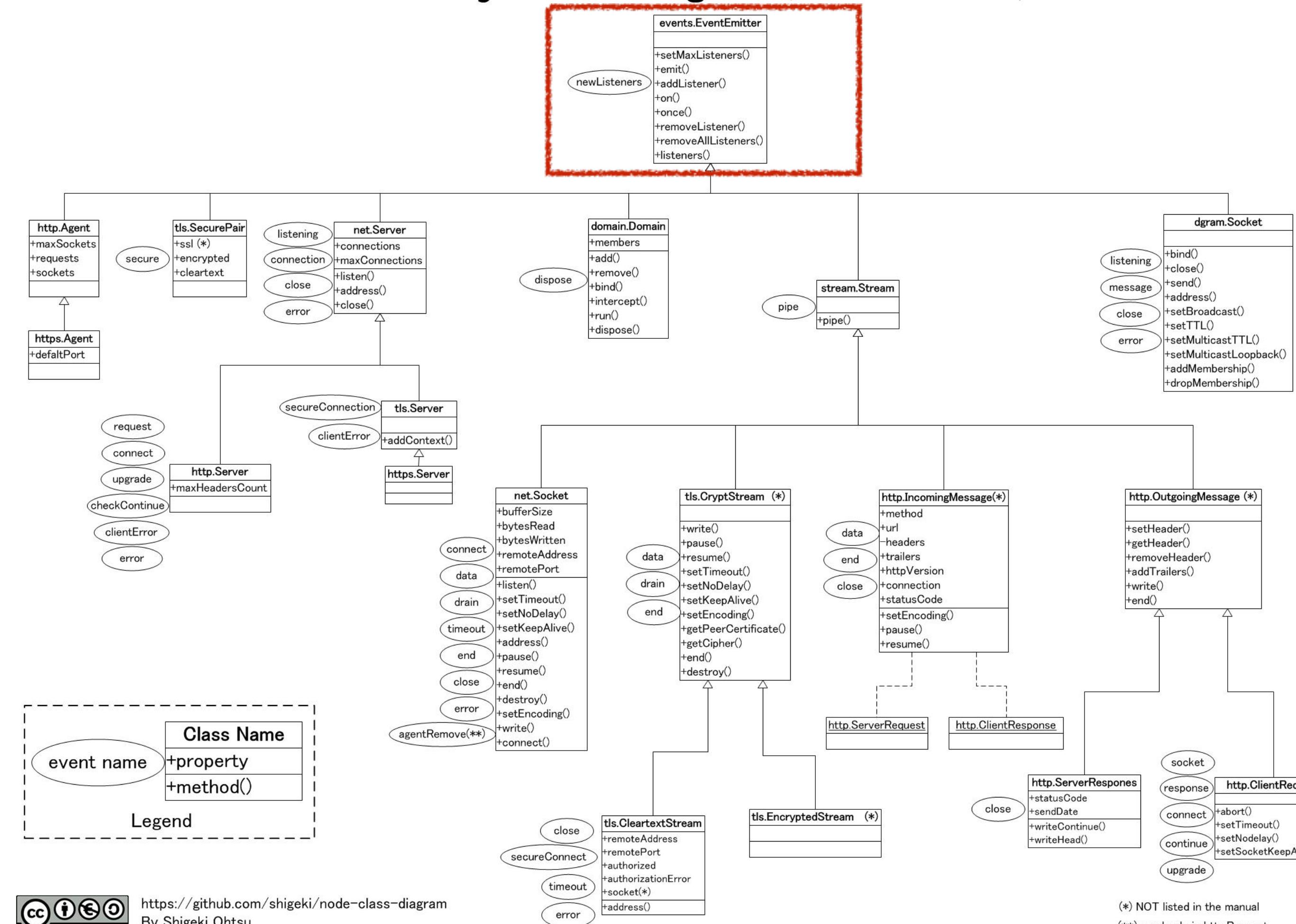


events

Node.js WS #1

(EventEmitter)

Node.js Class Diagram (node-v0.8.12)



Node.js WS #1

(EventEmitter)

```
var EventEmitter = require('events').EventEmitter;  
var eventEmitter = new EventEmitter();  
  
eventEmitter.on('customevent', function() {  
    console.log('Event Occur!!');  
});  
  
eventEmitter.emit('customevent');
```

Node.js WS #1

(EventEmitter - addListener)

EventEmitter

```
_events = { };
```

```
function addListener(type, listener) { ... };  
function emit(type) { ... };
```

Node.js WS #1

(EventEmitter - addListener)

EventEmitter

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



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

Node.js WS #1

(EventEmitter - emit)

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



EventEmitter

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

Node.js WS #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 WS #1

(Stream)

Stream?

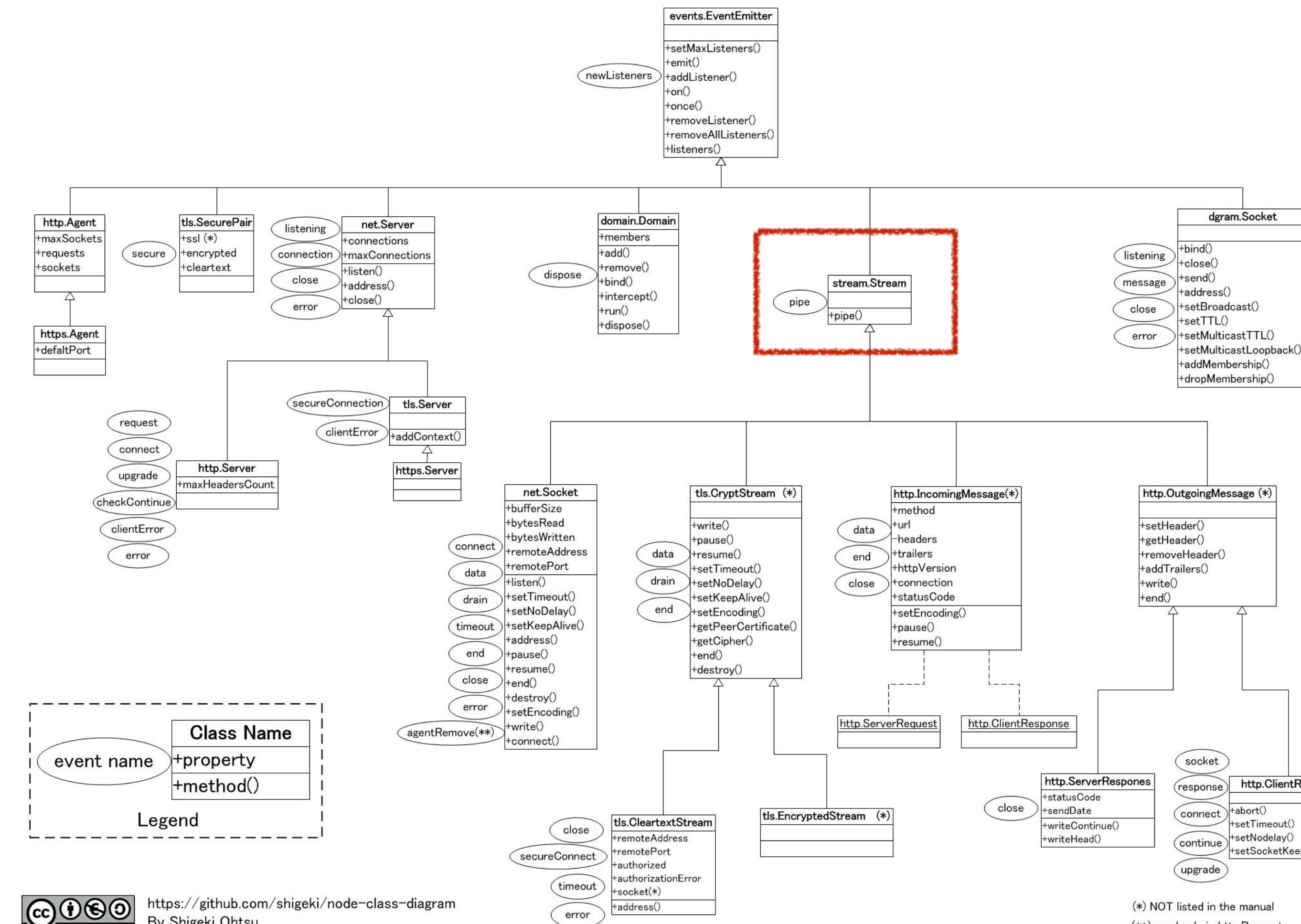
A stream is an **abstract interface** implemented by various objects in Node. For example a request to an HTTP server is a stream, as is stdout. Streams are readable, writable, or both. **All streams are instances of EventEmitter**

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

Node.js WS #1

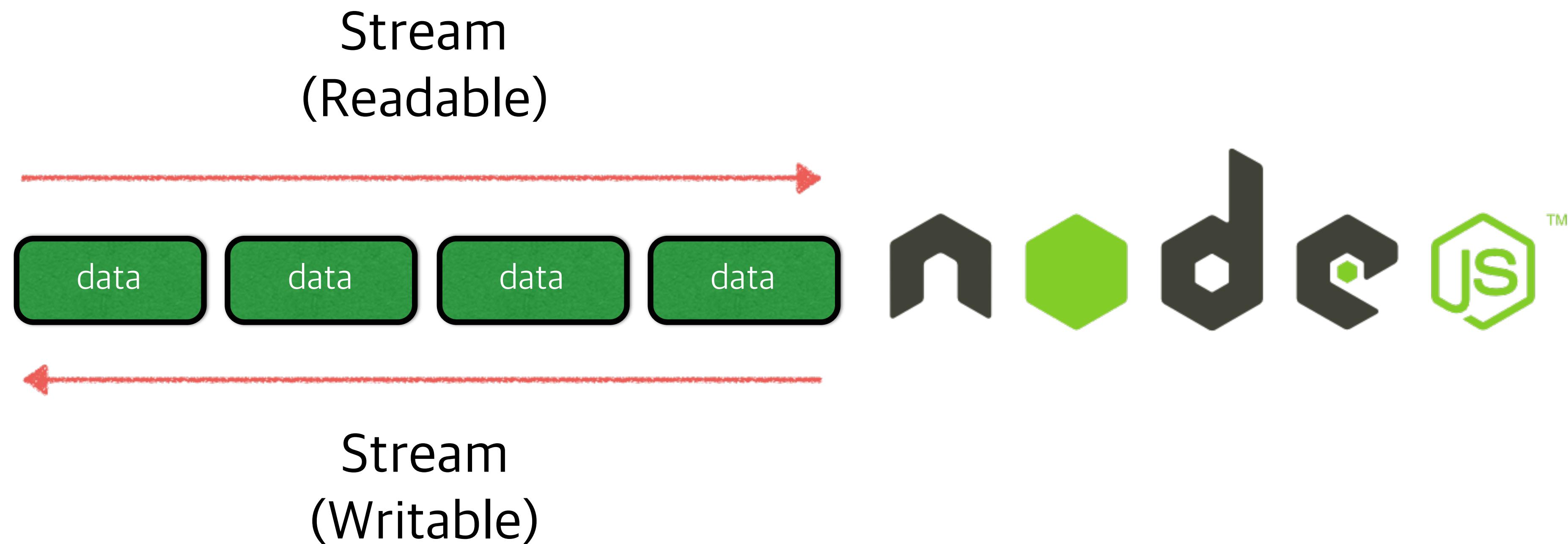
(Stream)

Node.js Class Diagram (node-v0.8.12)



Node.js WS #1

(Stream)



Node.js WS #1

(Stream)



- File IO
- Socket IO
- stdin/stdout

Node.js WS #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 WS #1

(Stream - module_stream.js)

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

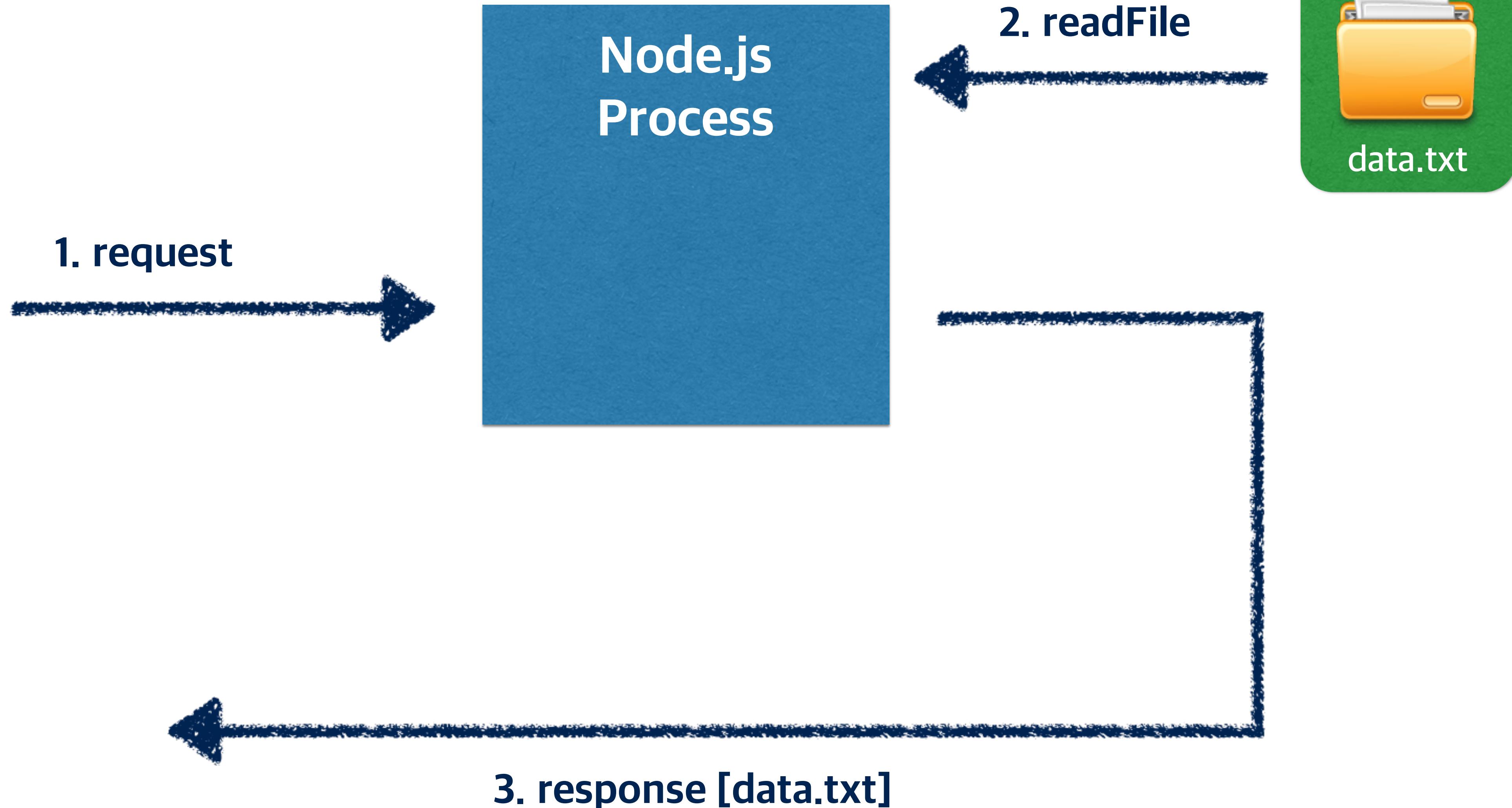
var server = http.createServer(function (req, res) {
    // req is an http.IncomingMessage, which is a Readable Stream
    // res is an http.ServerResponse, which is a Writable Stream

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

server.listen(8000);
```

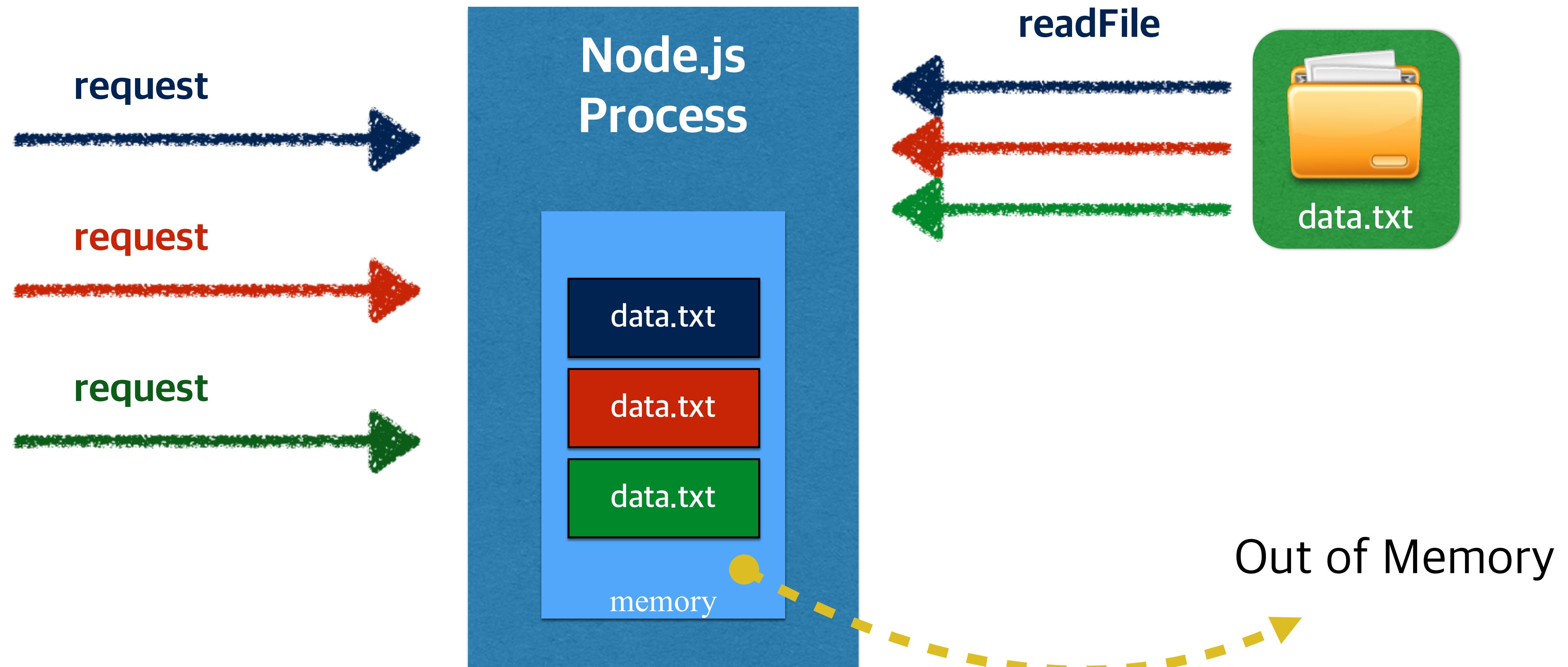
Node.js WS #1

(Stream – how to work module _stream.js)



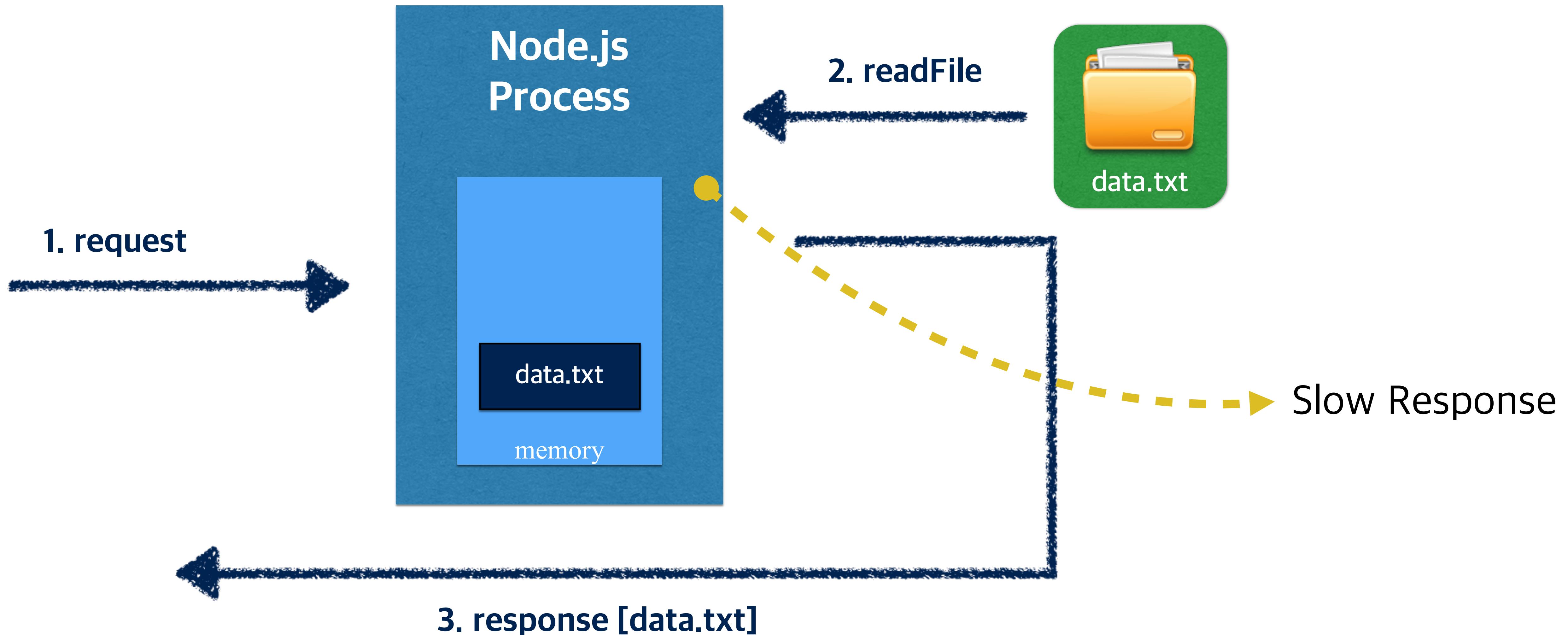
Node.js WS #1

(Stream - module_stream.js drawbacks-1)



Node.js WS #1

(Stream - module_stream.js drawbacks-2)



Node.js WS #1

(Stream - module_stream.js)

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

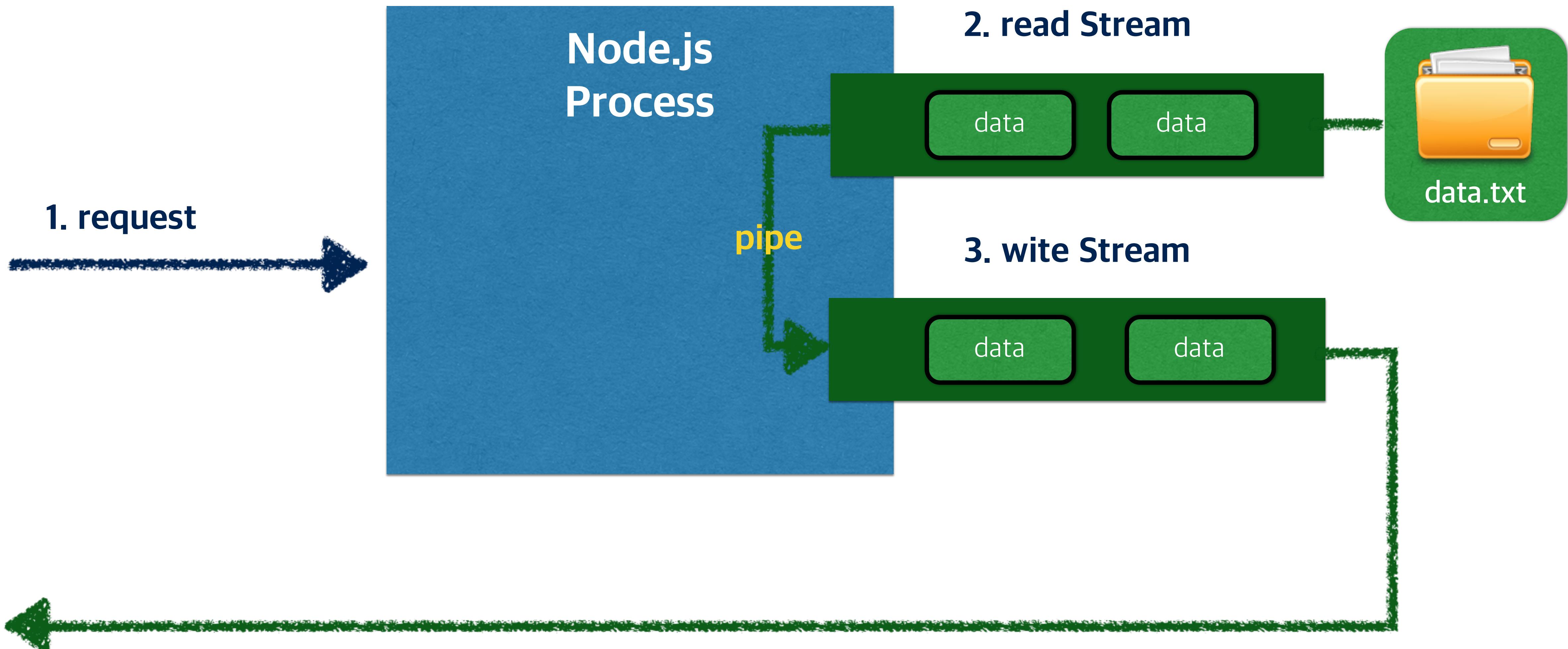
var server = http.createServer(function (req, res) {
  // req is an http.IncomingMessage, which is a Readable Stream
  // res is an http.ServerResponse, which is a Writable Stream

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

server.listen(8000);
```

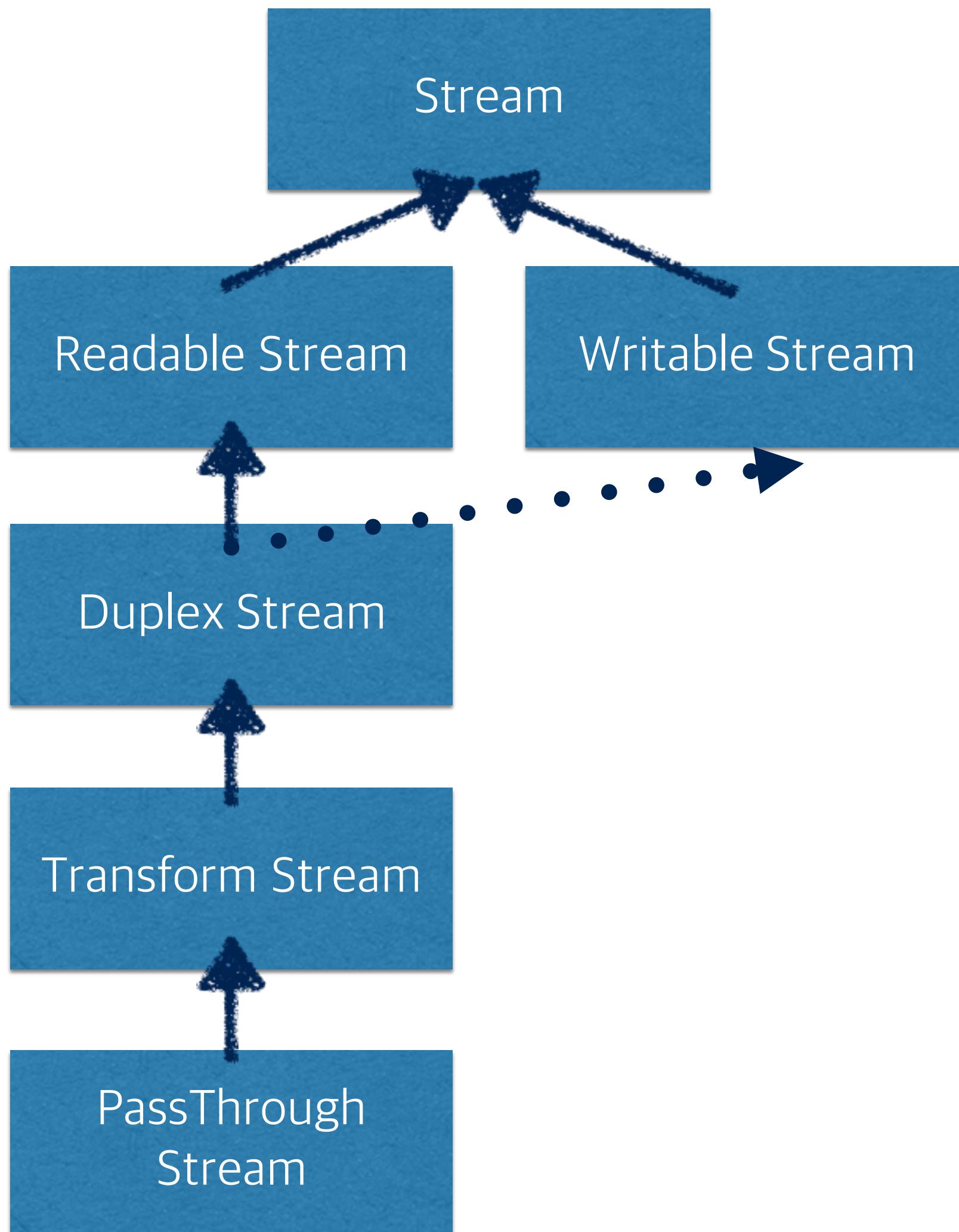
Node.js WS #1

(Stream – how to work module _stream.js)



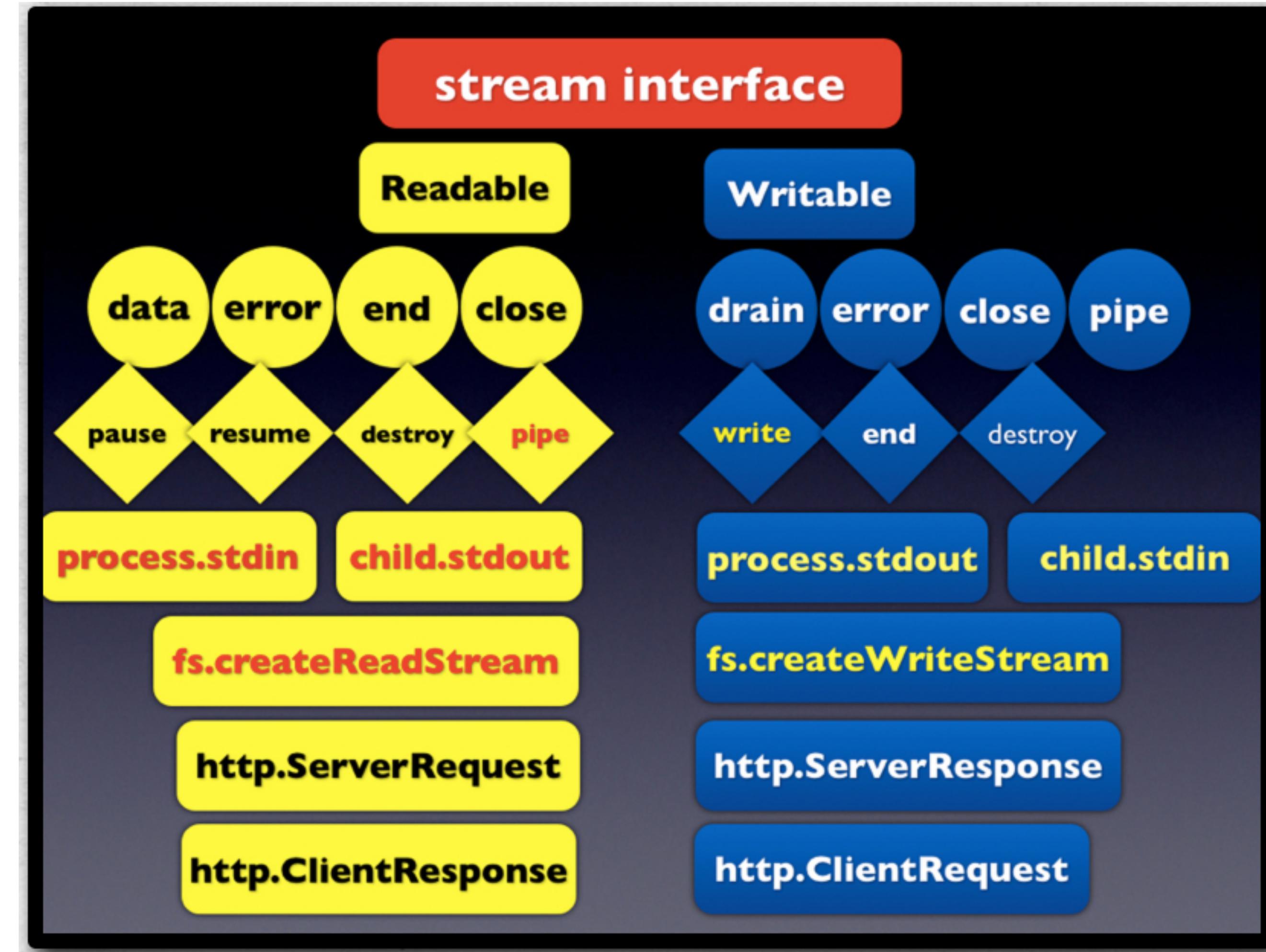
Node.js WS #1

(Stream)



Node.js WS #1

(Stream)



Node.js WS #1

(npm)

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

Node.js WS #1

(echoserver.js)

```
var Server = require('socket.io');
var io = new 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 WS #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 Workshop #2

(RESTful API Server - MemoServer)

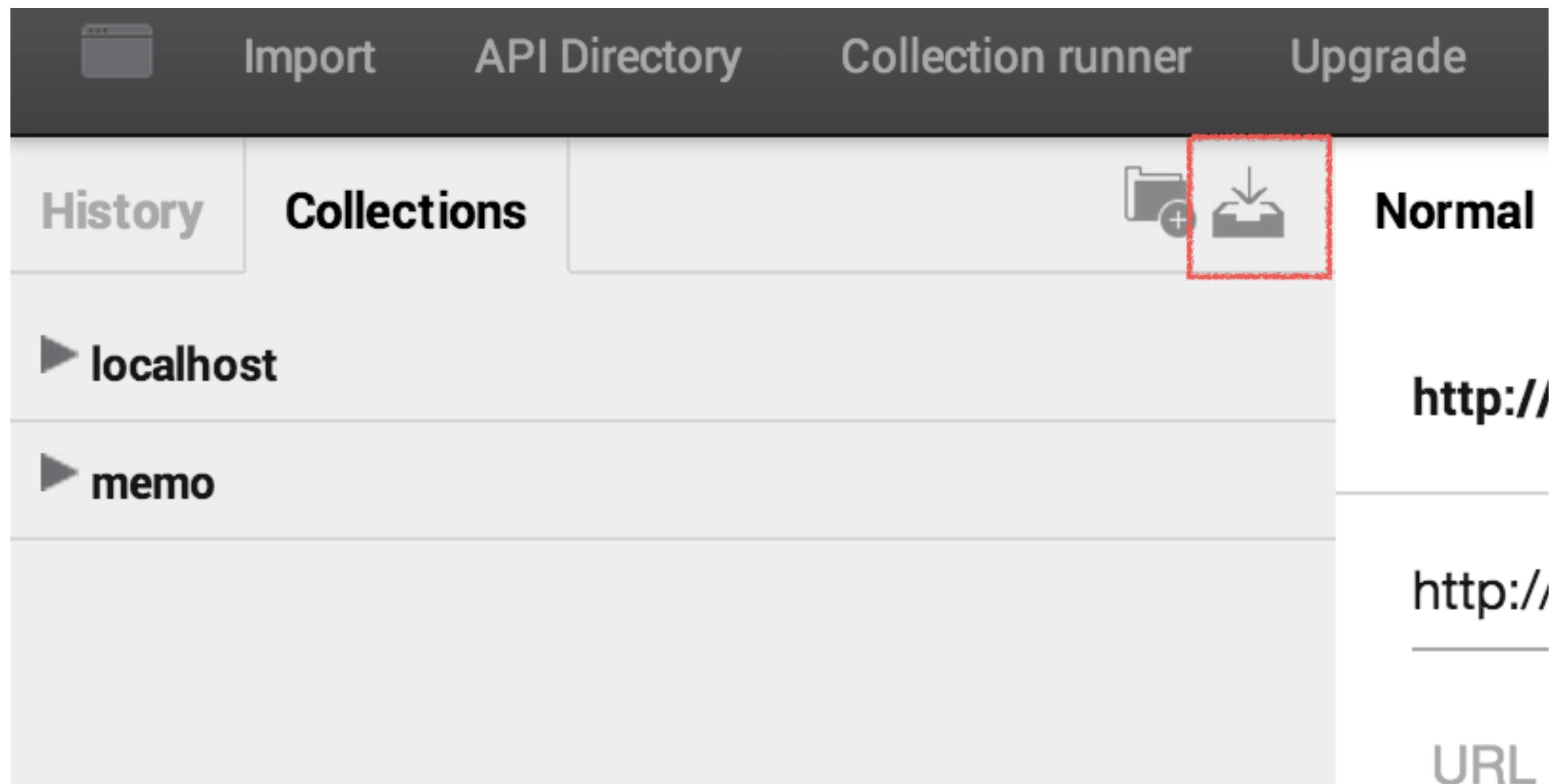
Node.js WS #2

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/fbjgbiflinjbdggehcddcbncddomop?utm_source=gmail

Node.js WS #2



Node.js WS #2

Import (beta)

Import a Postman Collection, curl command or a RAML/WADL/Swagger file.

It worked!
The collection memo copy has been added.

[Upload files](#) [Paste raw text](#) [Download from link](#)

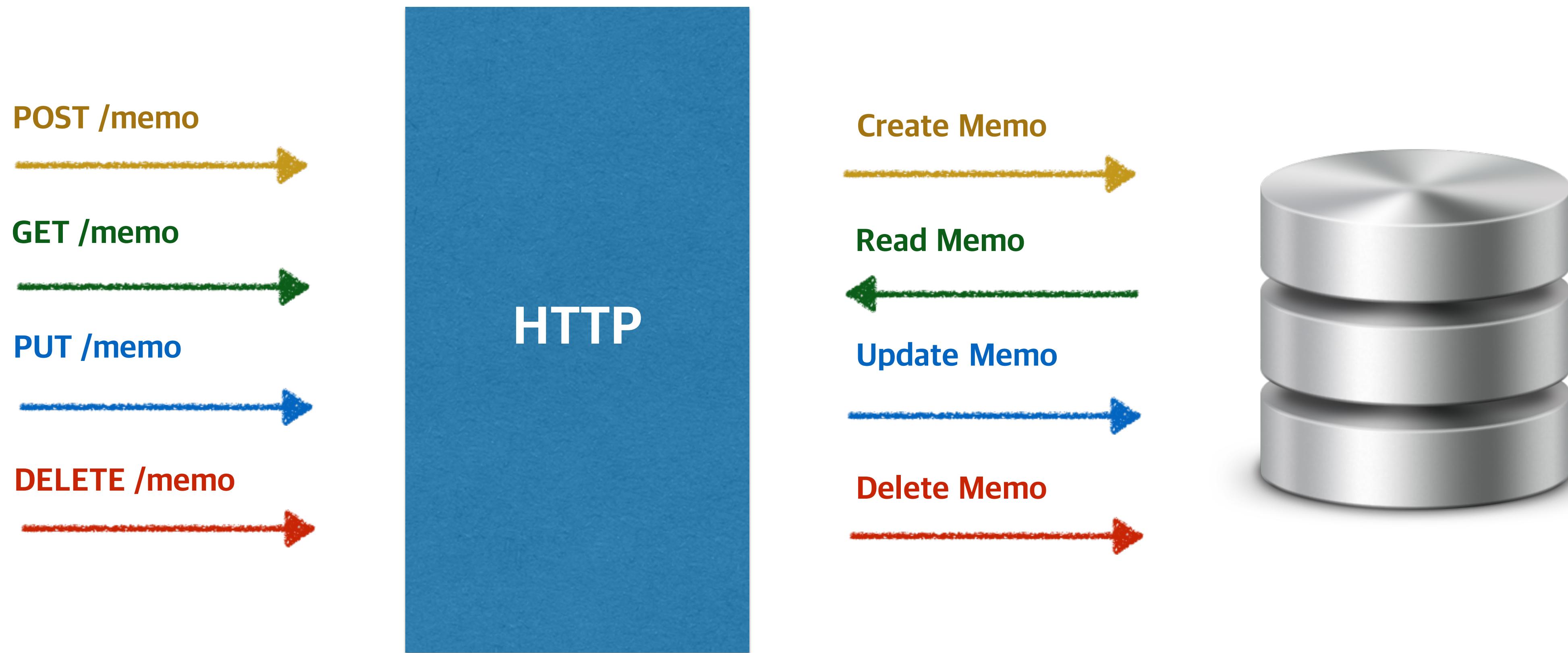
<https://www.getpostman.com/collections/59ccbd4349c38f69220e>

[Import!](#)

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

Node.js WS #2

(MemoServer Introduction)



Node.js WS #2

(server.js)

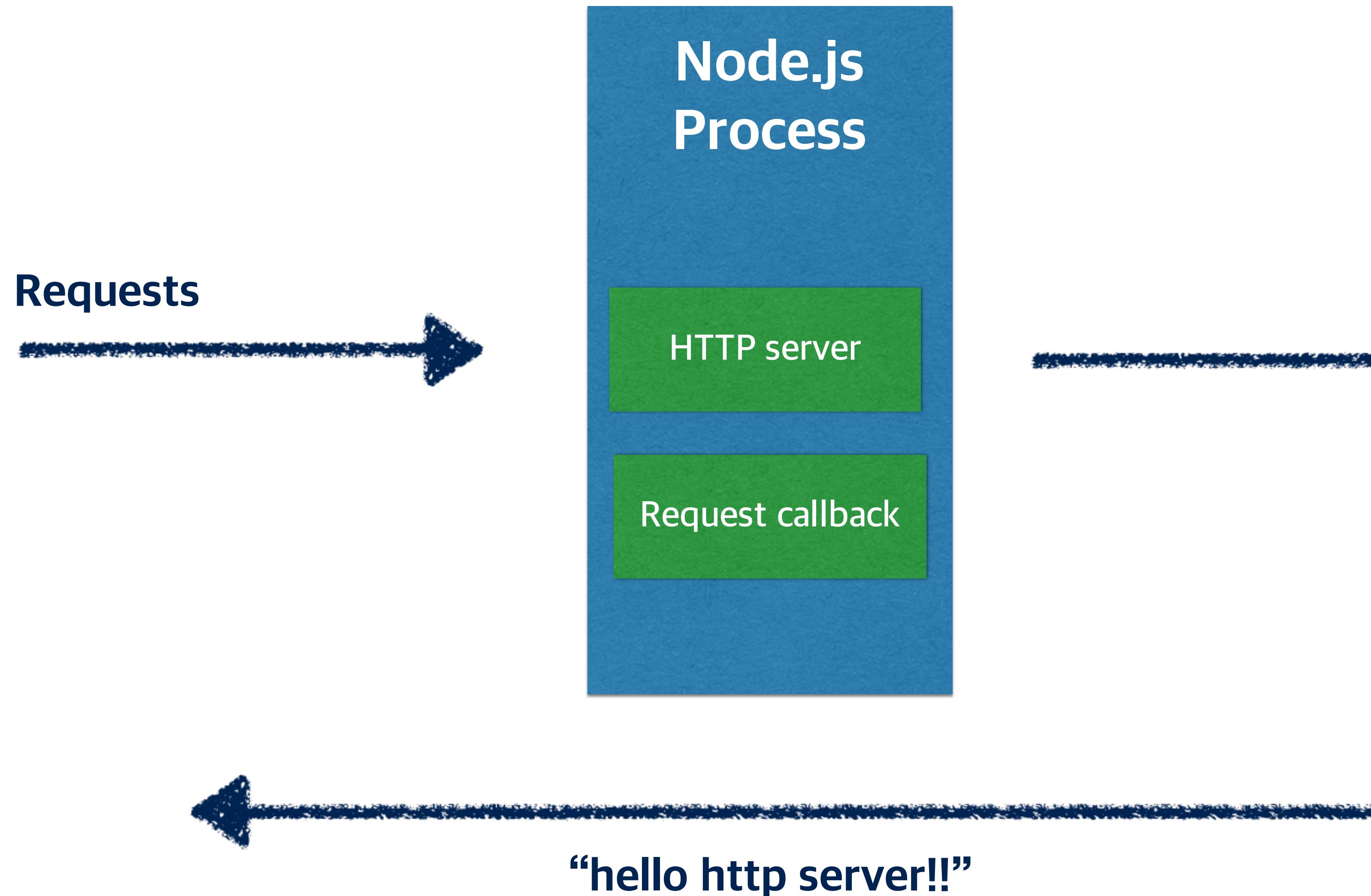
```
var http = require('http');

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);
```

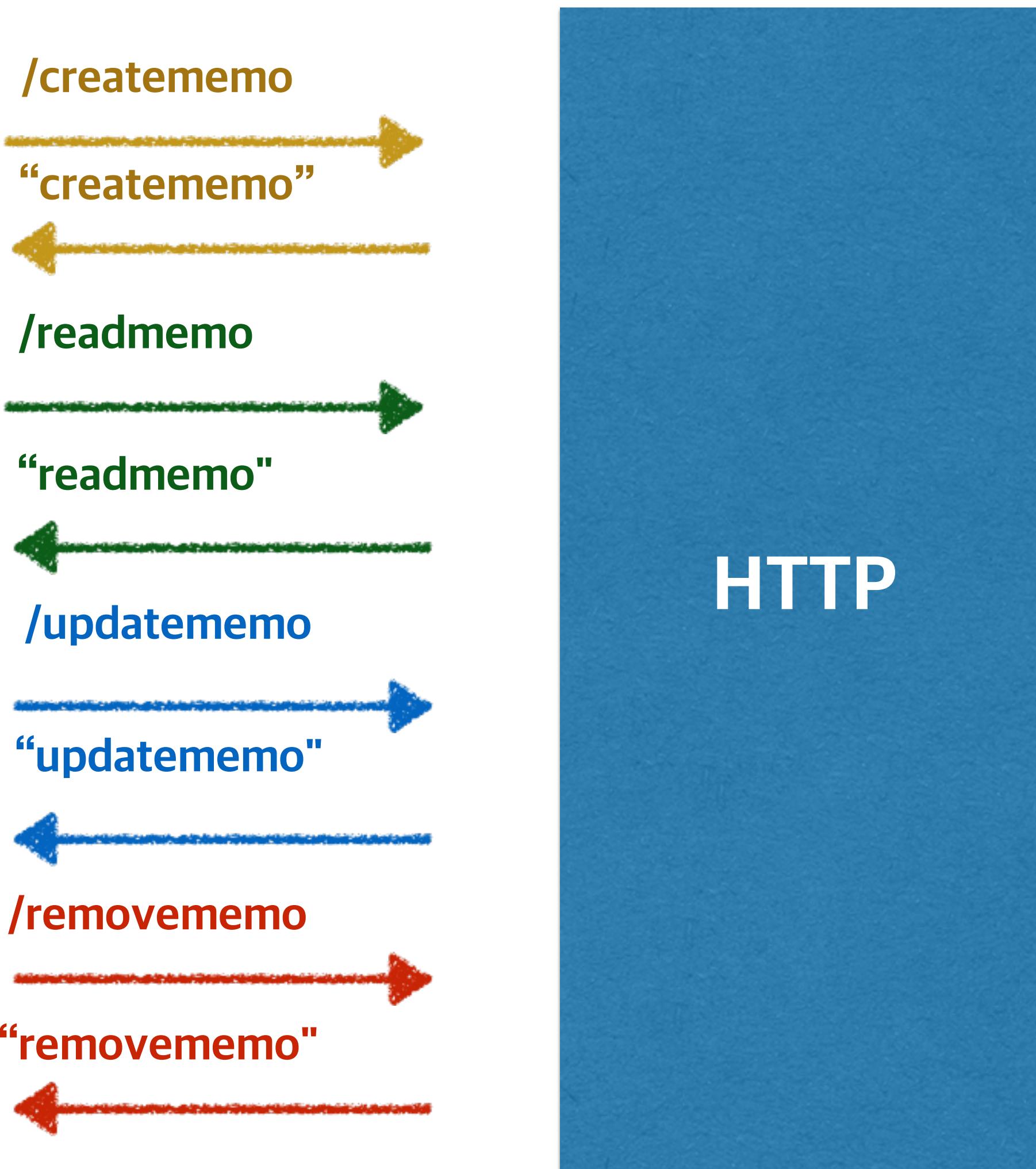
Node.js WS #2

(server.js interaction)



Node.js WS #2

(server.js - Add Router)



Node.js WS #2

(server.js - Add Router)

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

Node.js WS #2

(server.js - Add Router)

```
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 WS #2

(server.js - Add Router)

```
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 WS #2

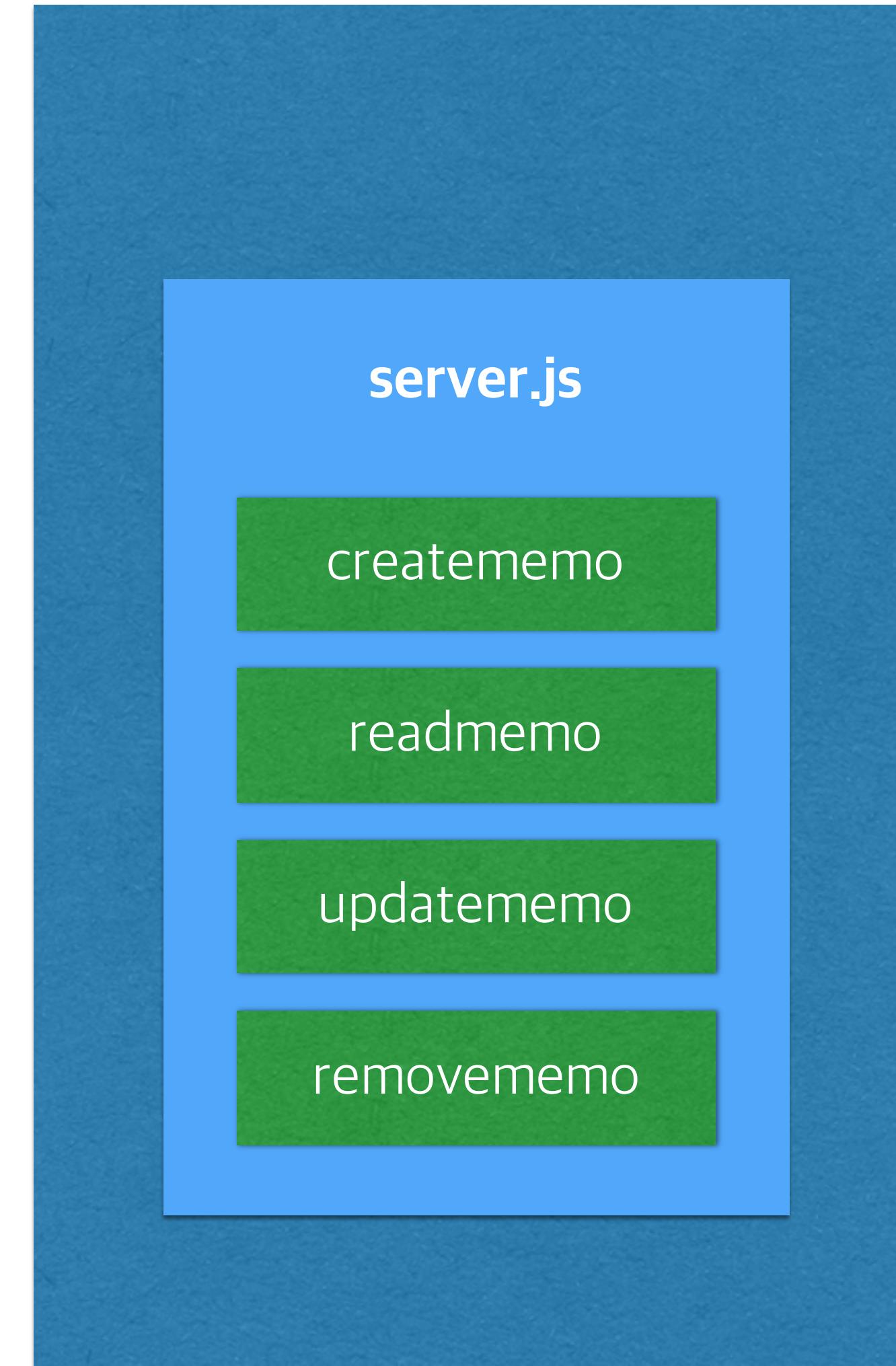
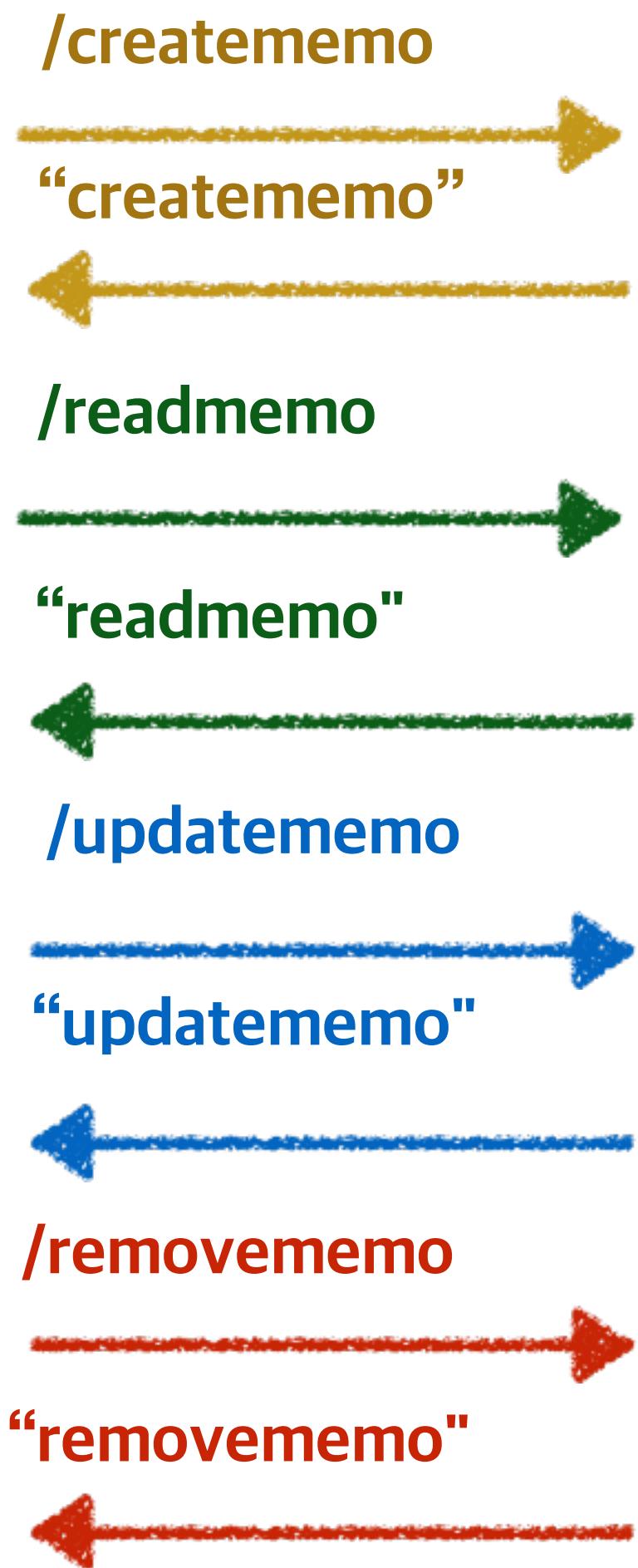
(server.js - Add Router)

```
url.parse(string).query  
url.parse(string).pathname  
  
-----  
http://localhost:8888/start?foo=bar&hello=world  
-----  
querystring(string)[ "foo" ]  
querystring(string)[ "hello" ]
```

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

Node.js WS #2

(server.js - Add Router)

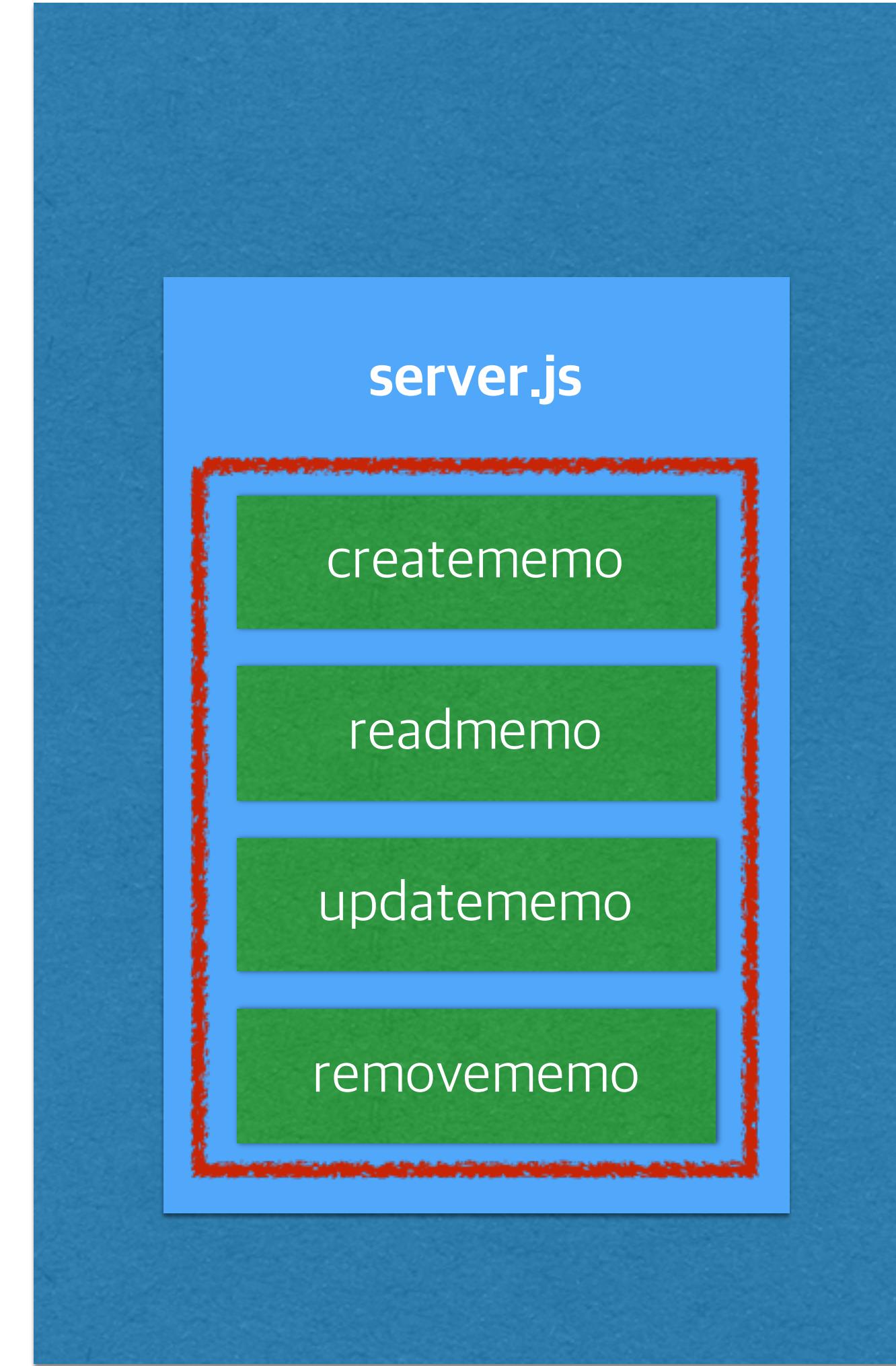
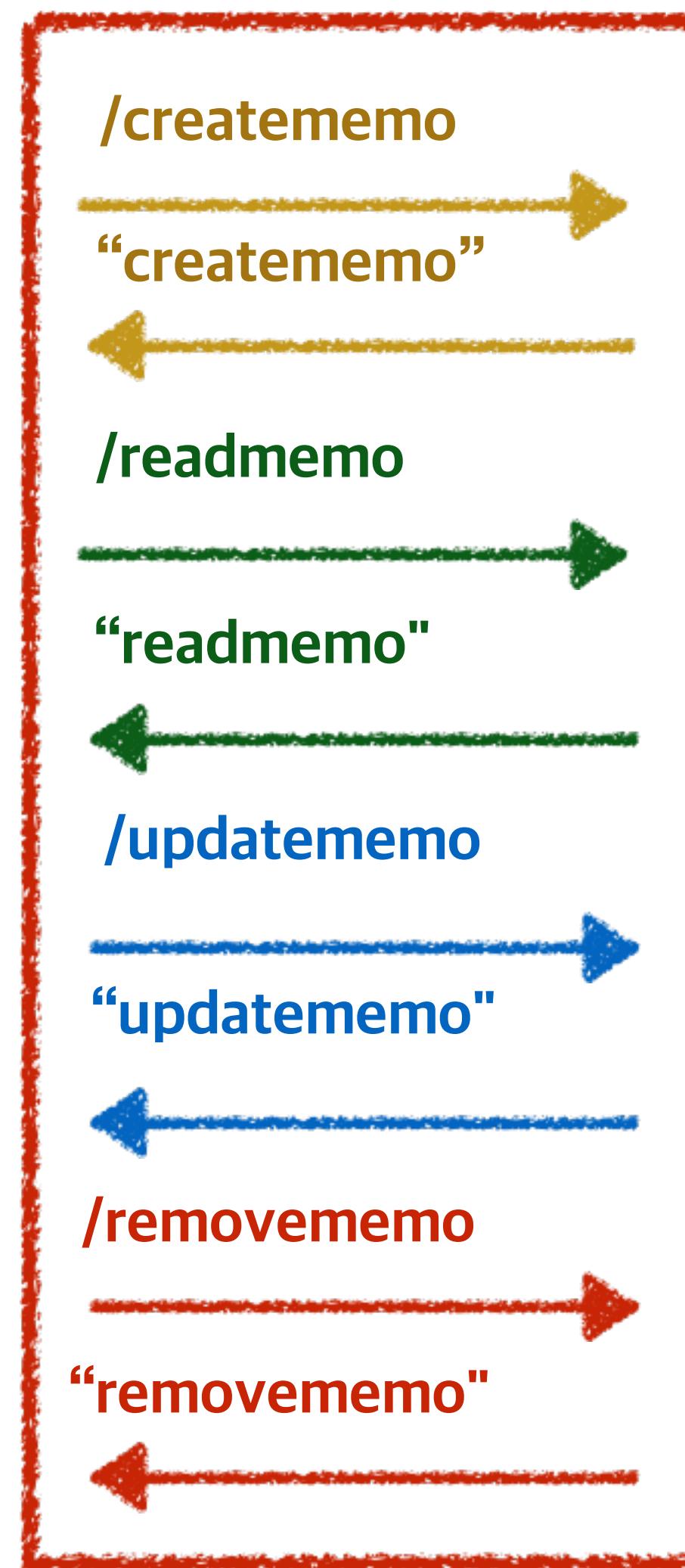


Node.js WS #2

(server.js - Add Router)

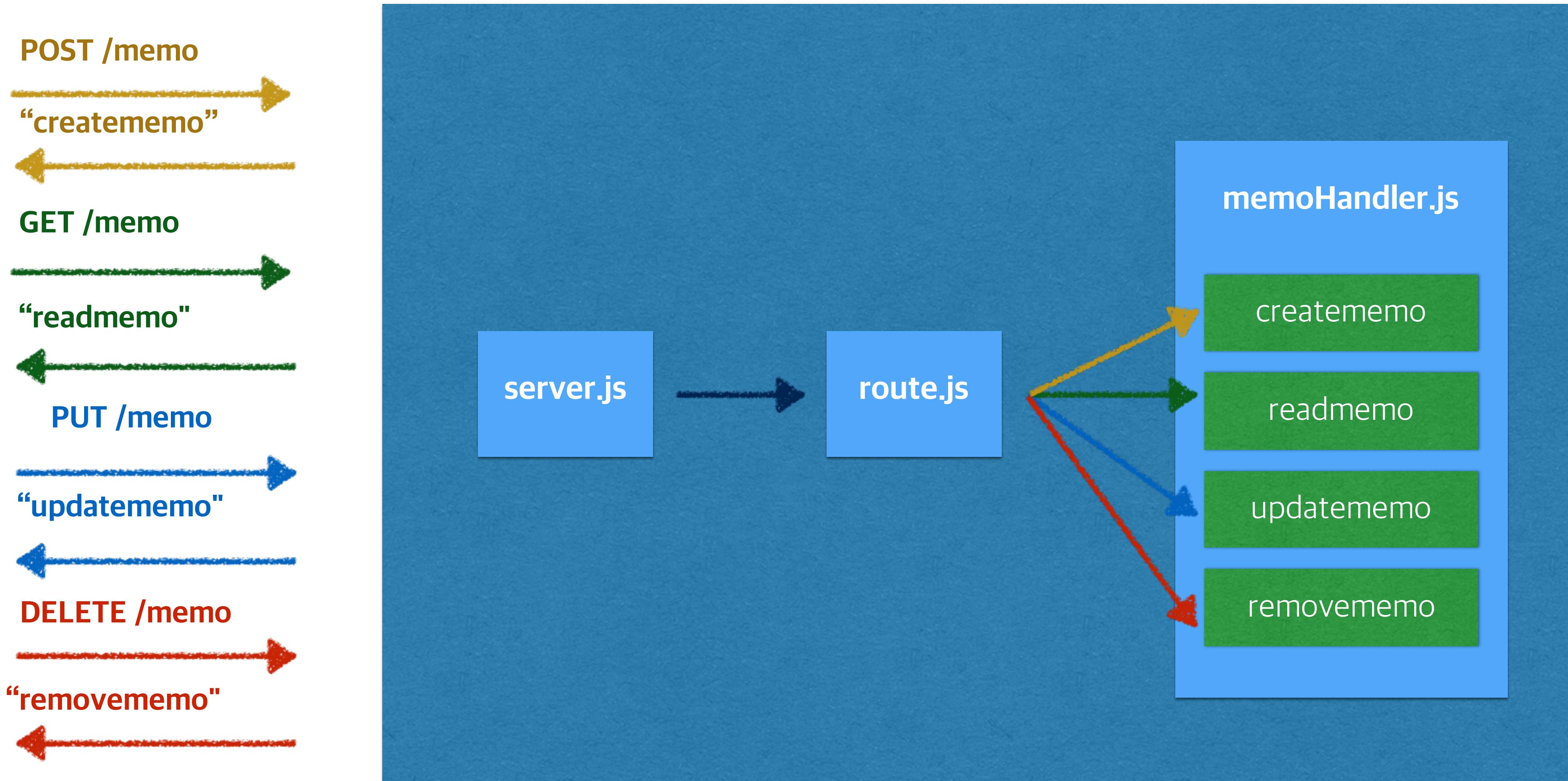
RESTful

- POST
- GET
- DELETE
- PUT



Node.js WS #2

(server.js - Add Router)



Node.js WS #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 WS #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 WS #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;
    });

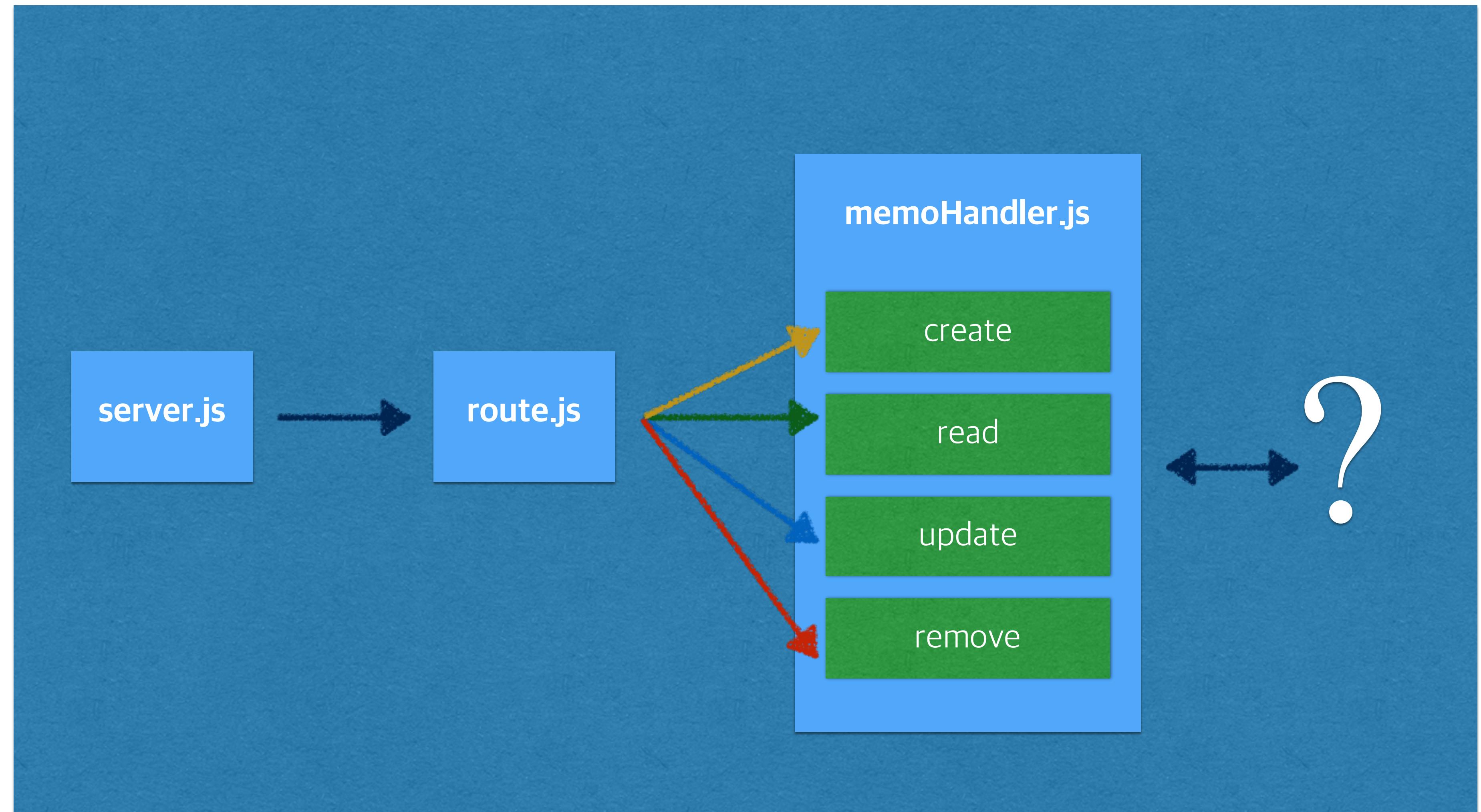
    req.on('end', function() {
        route.route(req, res, body);
    });
}

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

Node.js WS #2

(server.js - Add Router)

POST /memo
"creatememo"
GET /memo
"readmemo"
PUT /memo
"updatememo"
DELETE /memo
"removememo"



Node.js WS #2

(Database)

<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
- East Install
- Subset of Mongodb API

Node.js WS #2

(Database - nedb format)

```
{"author":"pcw","memo":"test","date":{"$$date":1421153113134},"_id":"nxi5jUVW2AZe1Hd9"}  
{"author":"pcw","memo":"test","date":{"$$date":1421153130192},"_id":"Cm0jzs0HwwHgMPo0"}  
{"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 WS #2

(Database)

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 WS #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 WS #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);
}

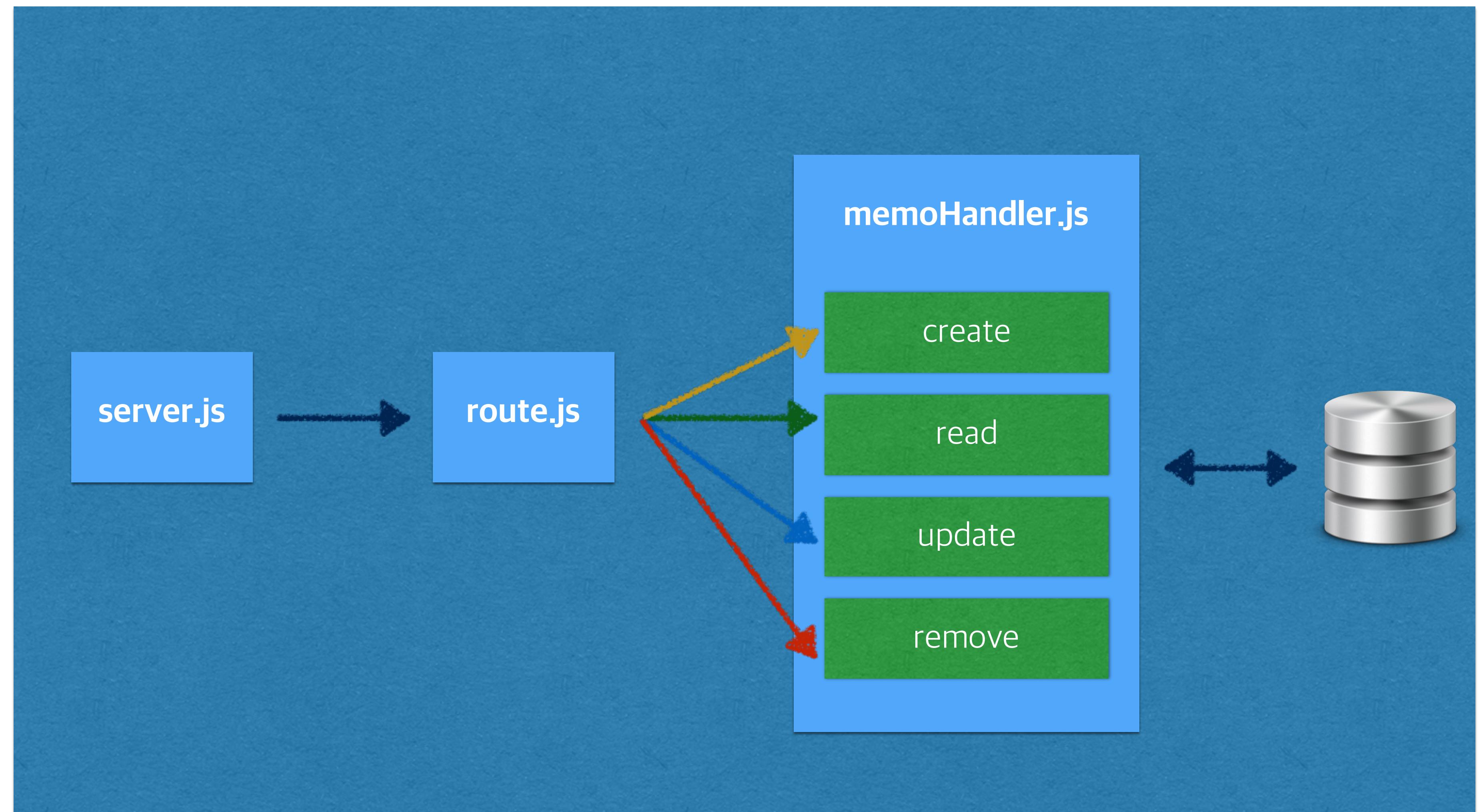
function _updateMemo(where, body, callback) {
  body = typeof body === 'string' ? JSON.parse(body) : body;
  db.update(where, {$set: body}, {multi: true}, callback);
}

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

Node.js WS #2

(server.js – Add Router)

POST /memo
"creatememo"
GET /memo
"readmemo"
UPDATE /memo
"updatememo"
DELETE /memo
"removememo"

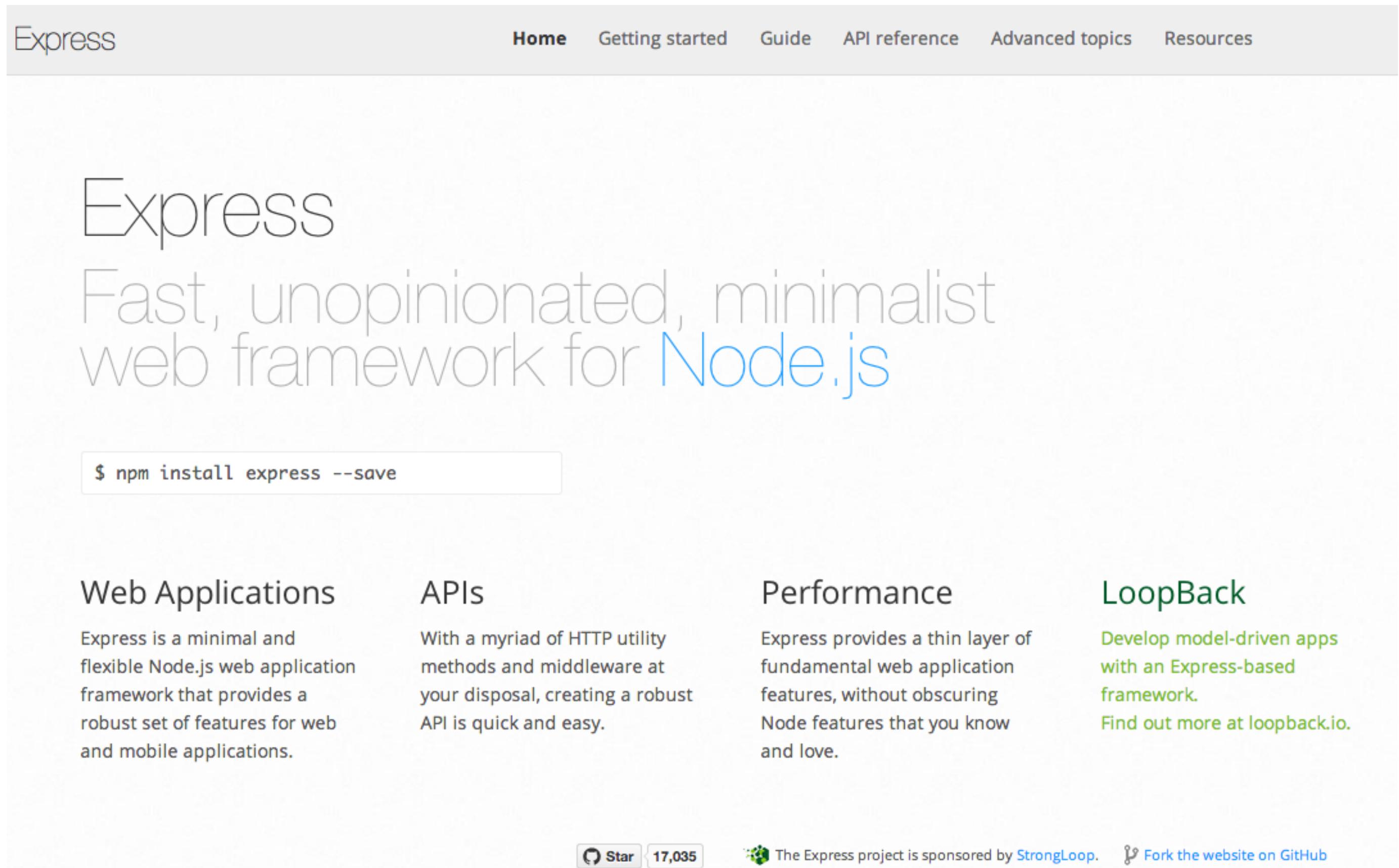


Node.js WS #3

(express)

Node.js WS #3

(express)



The screenshot shows the official Express.js website. At the top, there's a navigation bar with links for Home, Getting started, Guide, API reference, Advanced topics, and Resources. The main title "Express" is prominently displayed. Below it, a subtitle reads "Fast, unopinionated, minimalist web framework for Node.js". A command-line interface (CLI) box contains the text "\$ npm install express --save". The page is divided into several sections: "Web Applications", "APIs", "Performance", and "LoopBack". Each section has a brief description and a link to more information. At the bottom, there are GitHub statistics ("Star 17,035") and a note about StrongLoop sponsorship.

Express

Home Getting started Guide API reference Advanced topics Resources

Express

Fast, unopinionated, minimalist web framework for Node.js

```
$ npm install express --save
```

Web Applications

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node features that you know and love.

LoopBack

Develop model-driven apps with an Express-based framework. Find out more at [loopback.io](#).

Star 17,035

The Express project is sponsored by [StrongLoop](#). Fork the website on [GitHub](#)

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

Node.js WS #3

(express)

```
$ npm install -g express-generator@4
$ express express_memoserver
$ cd express_memoserver
$ npm install
$ npm start
```

Node.js WS #3

(express)

localhost:3000

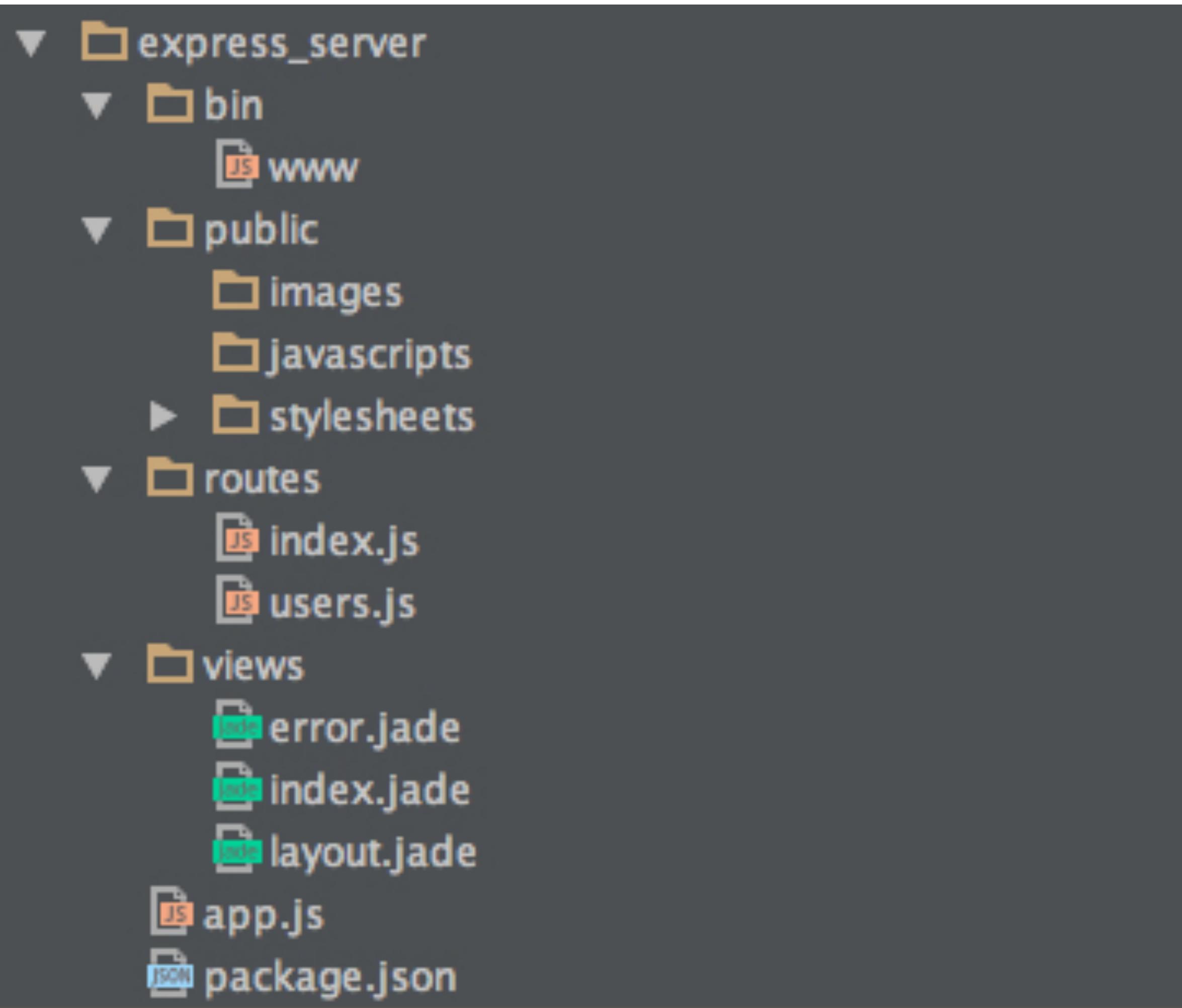


Express

Welcome to Express

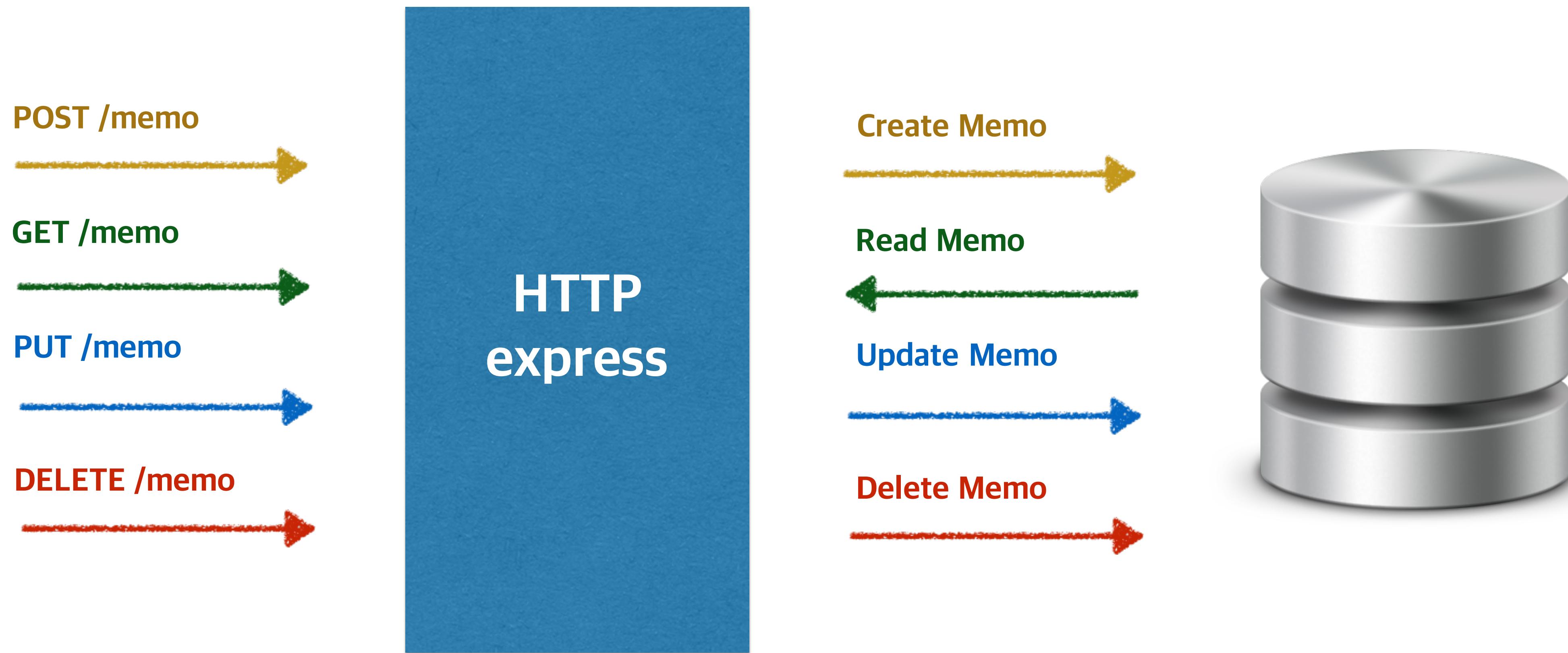
Node.js WS #3

(express - Directory Structure)



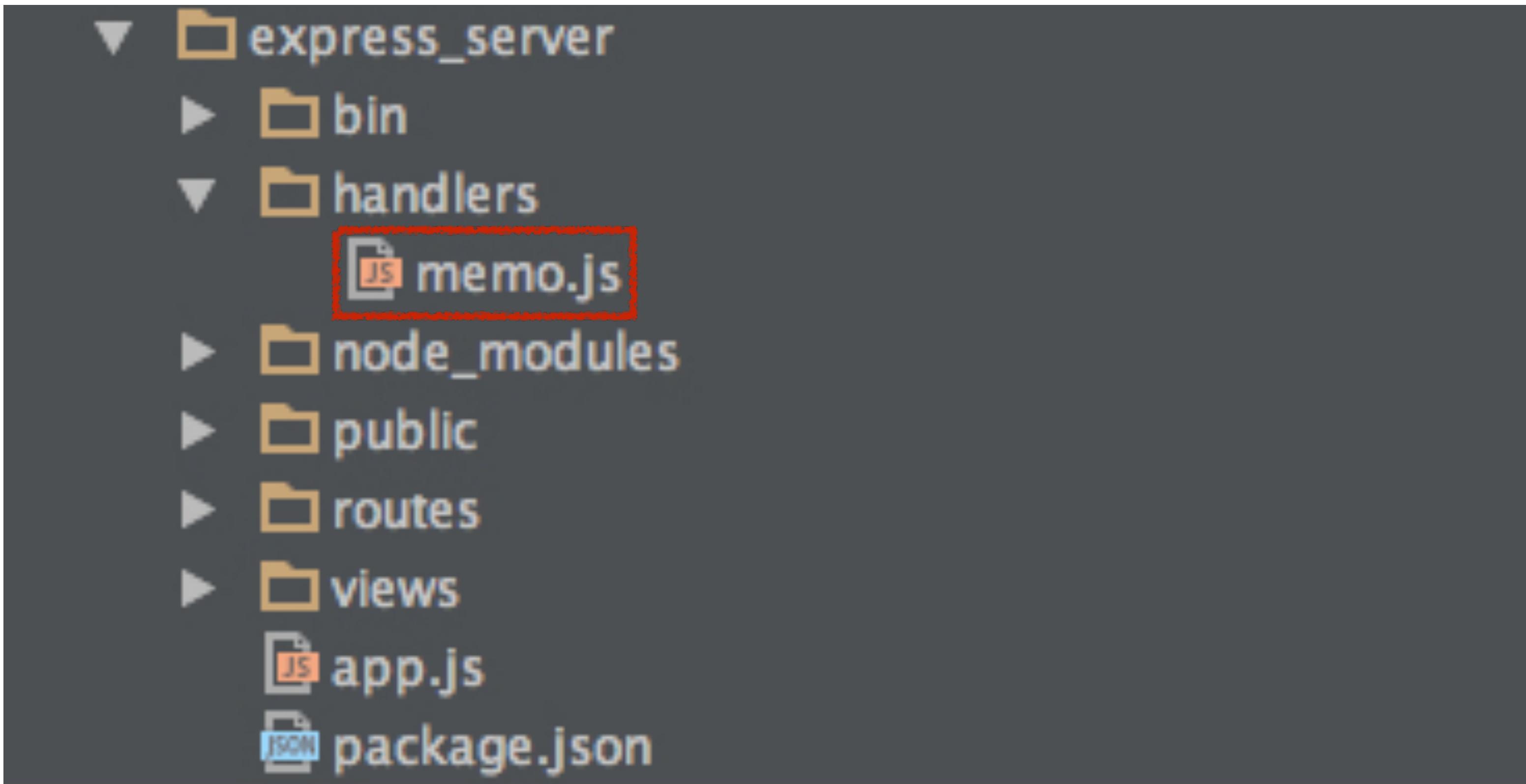
Node.js WS #3

(express - memoserver)



Node.js WS #3

(express - handlers/memo.js)



Node.js WS #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 WS #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 WS #3

(express - routes/memo.js)



Node.js WS #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 WS #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')));

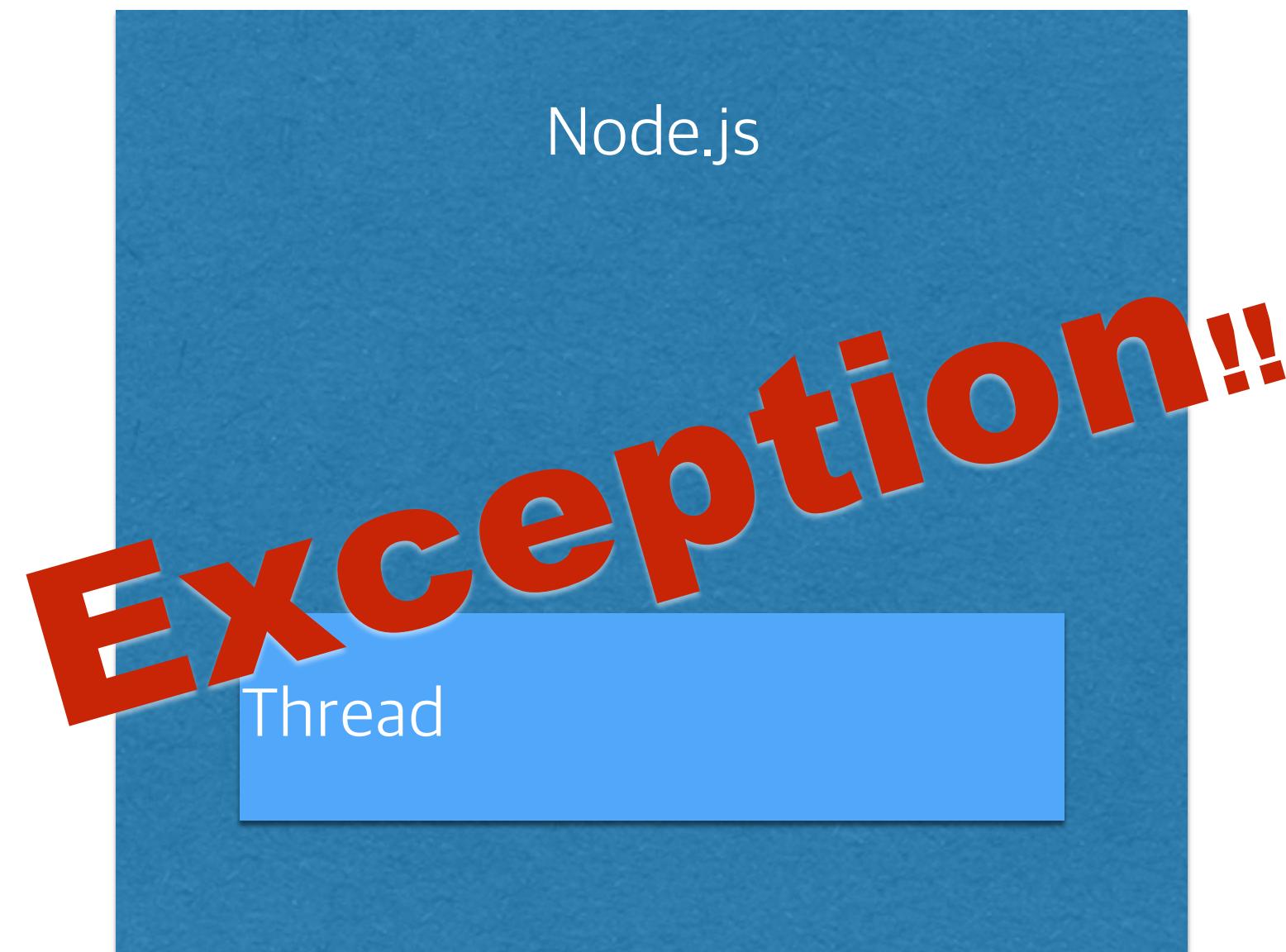
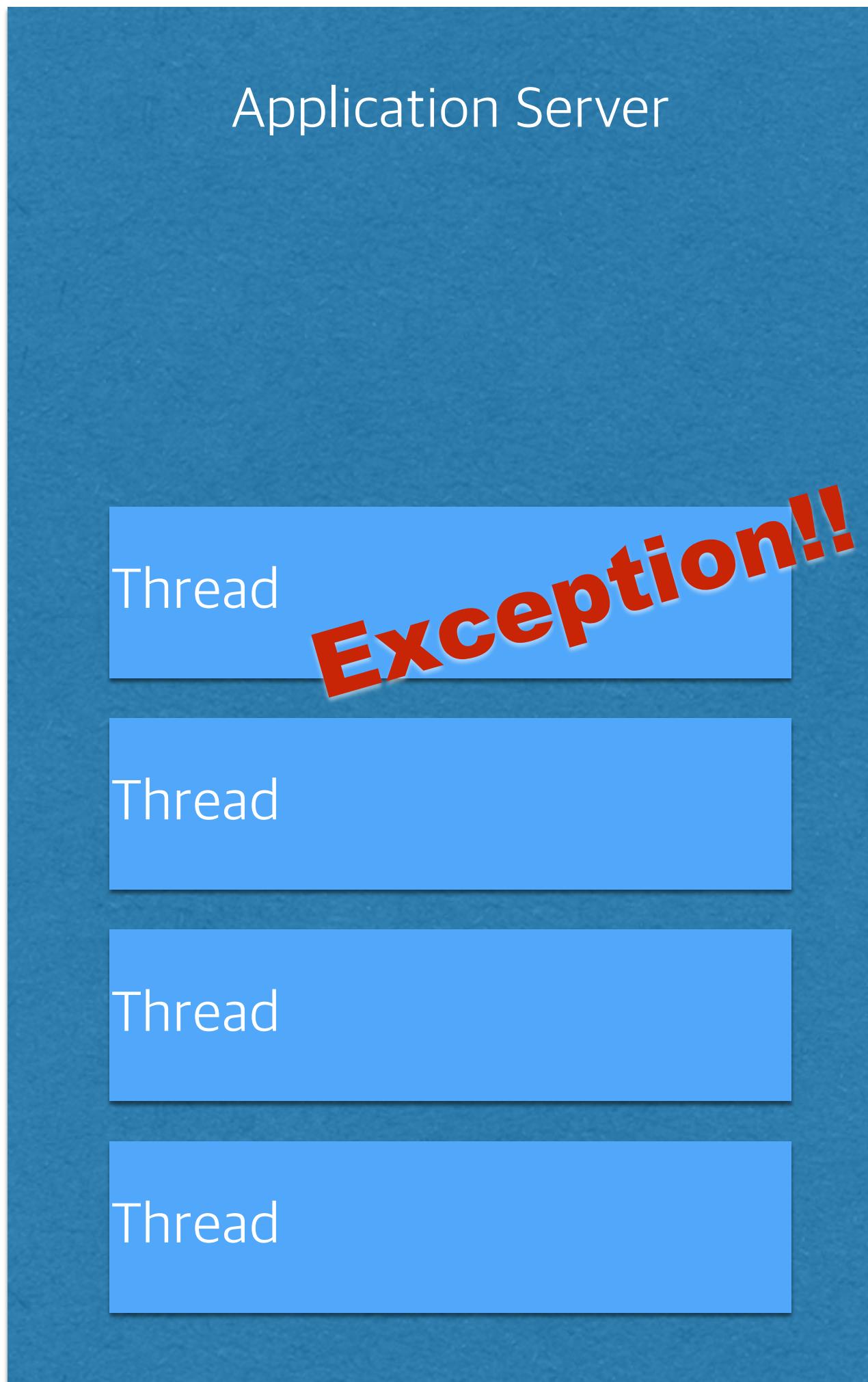
app.use('/', routes);
app.use('/users', users);
app.use('/memo', memo);
```

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`
- `domain`
- `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)

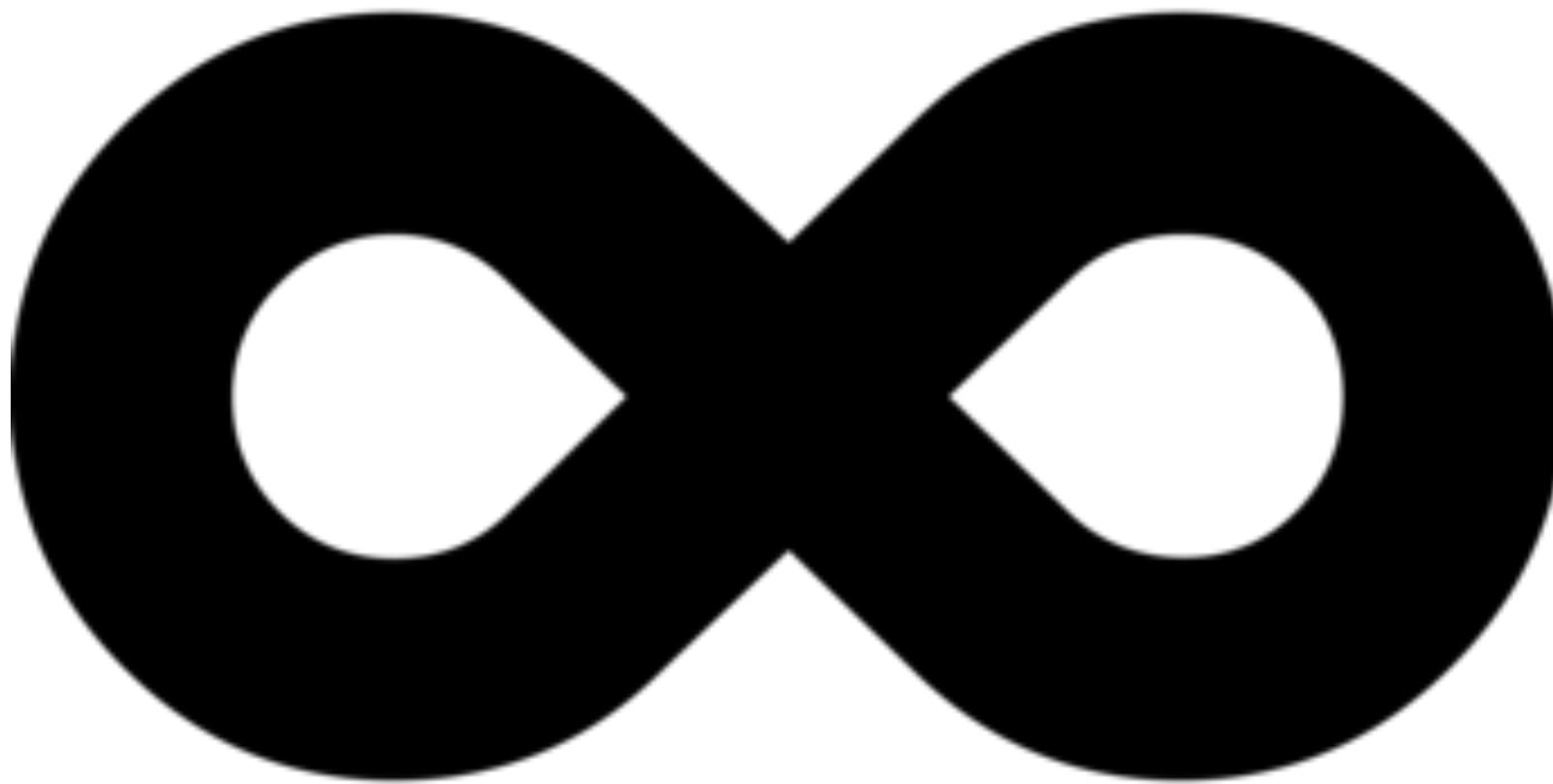
```
var domain = require('domain');
var d = domain.create();
var fs = require('fs');

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)



Node.js Production Ready

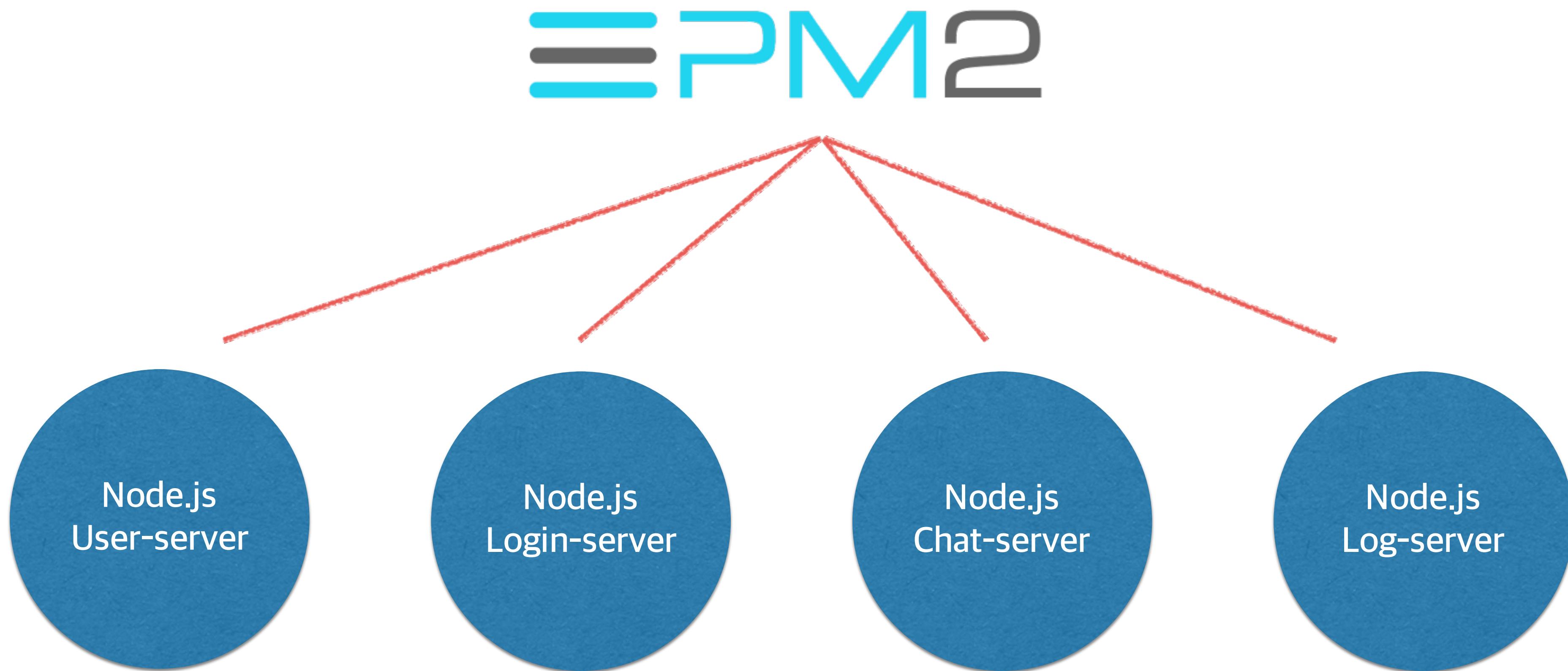
(Error handling)



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.

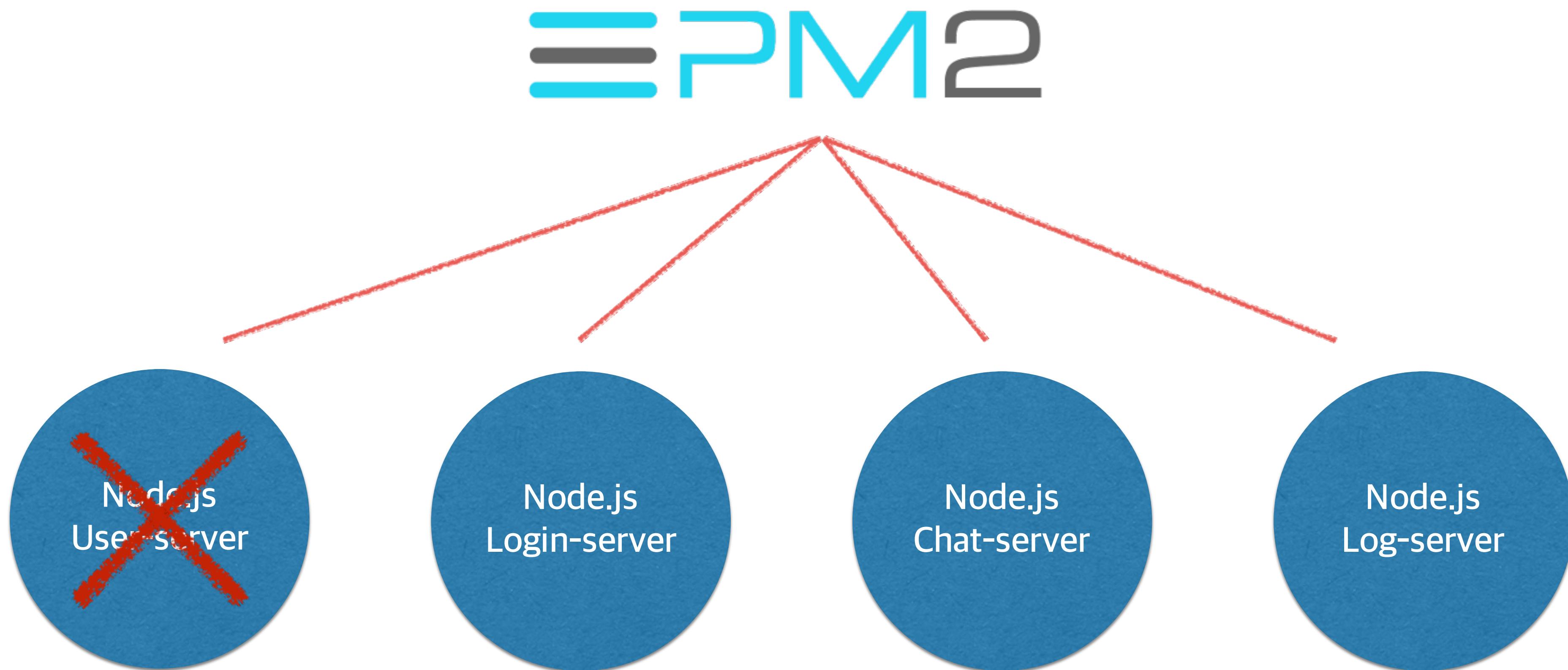
Node.js Production Ready

(Error handling)



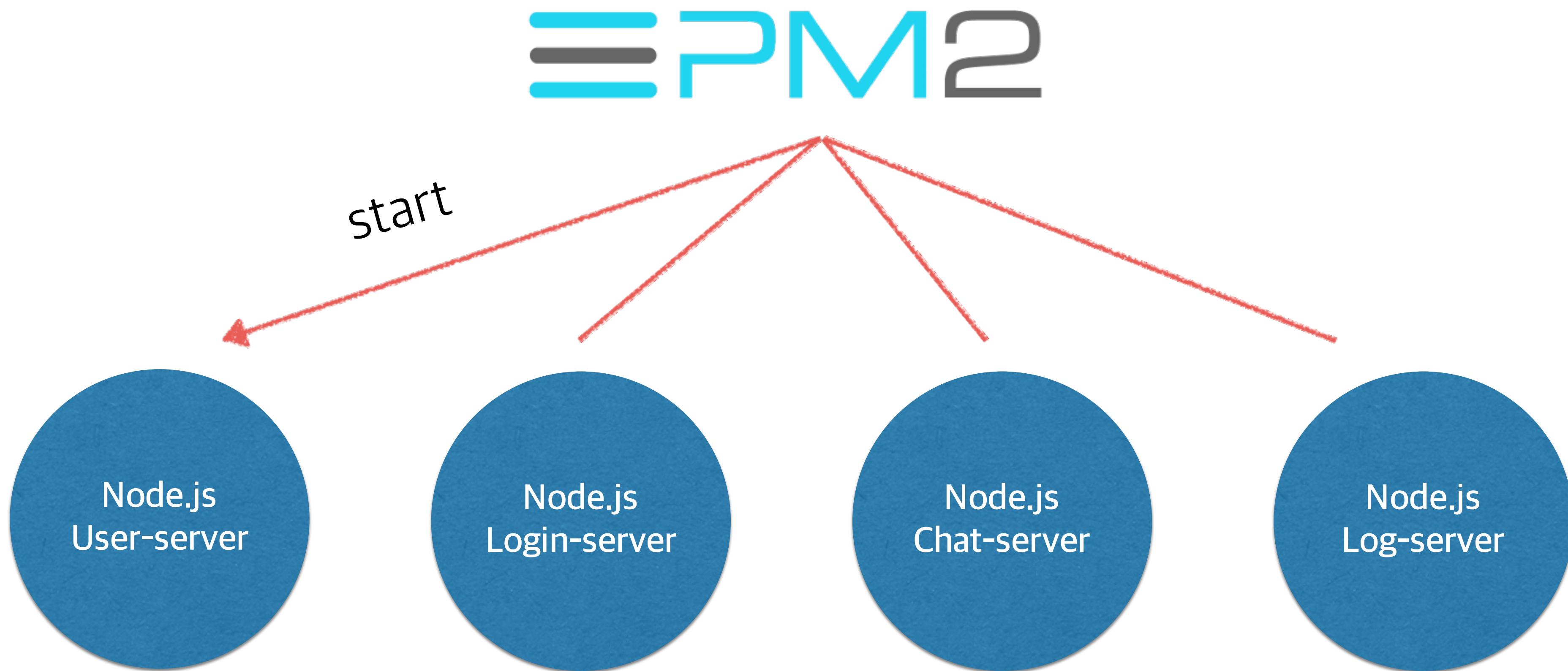
Node.js Production Ready

(Error handling)



Node.js Production Ready

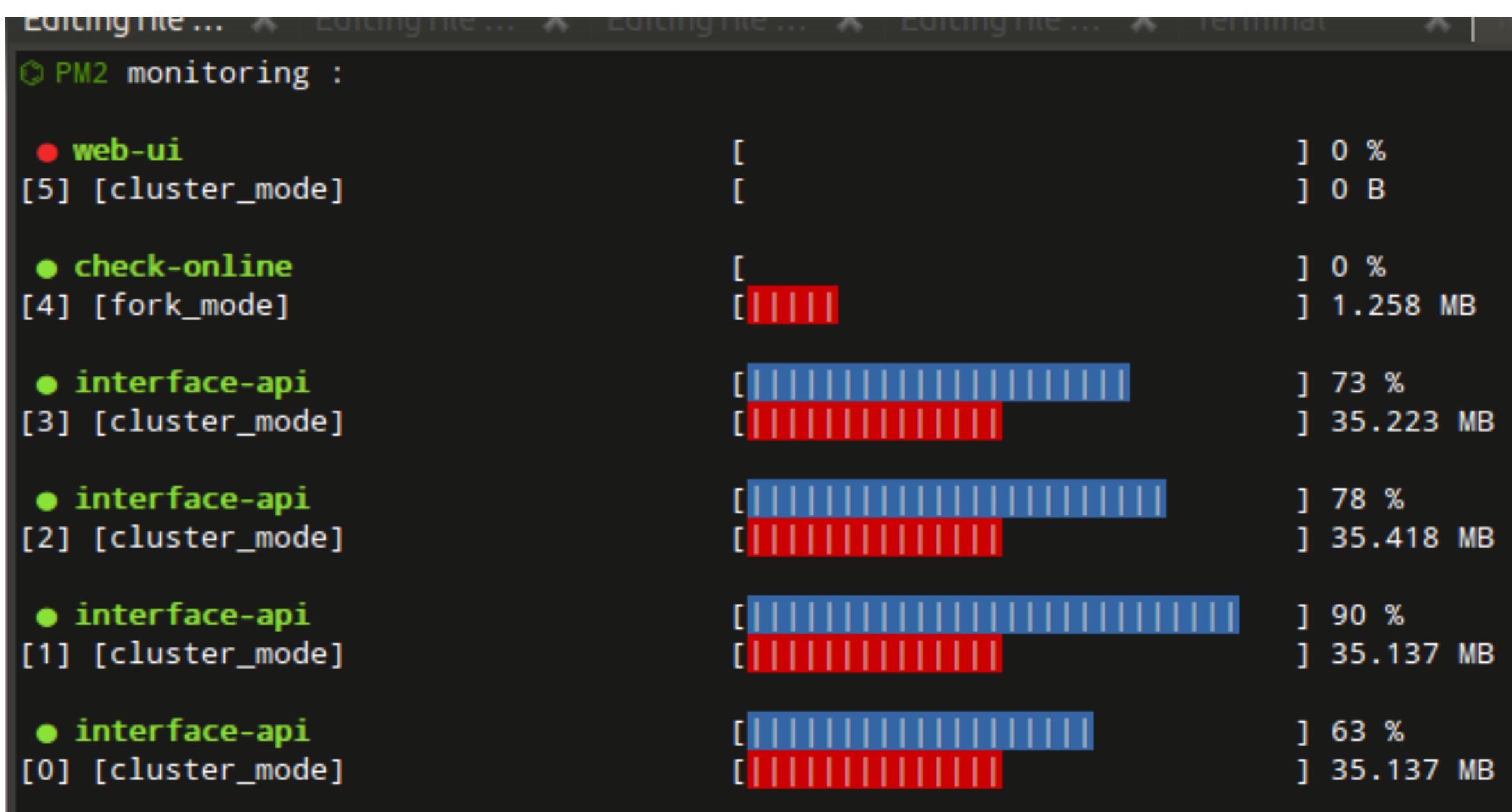
(Error handling)



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 tknew@Transport tknew@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
```



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)

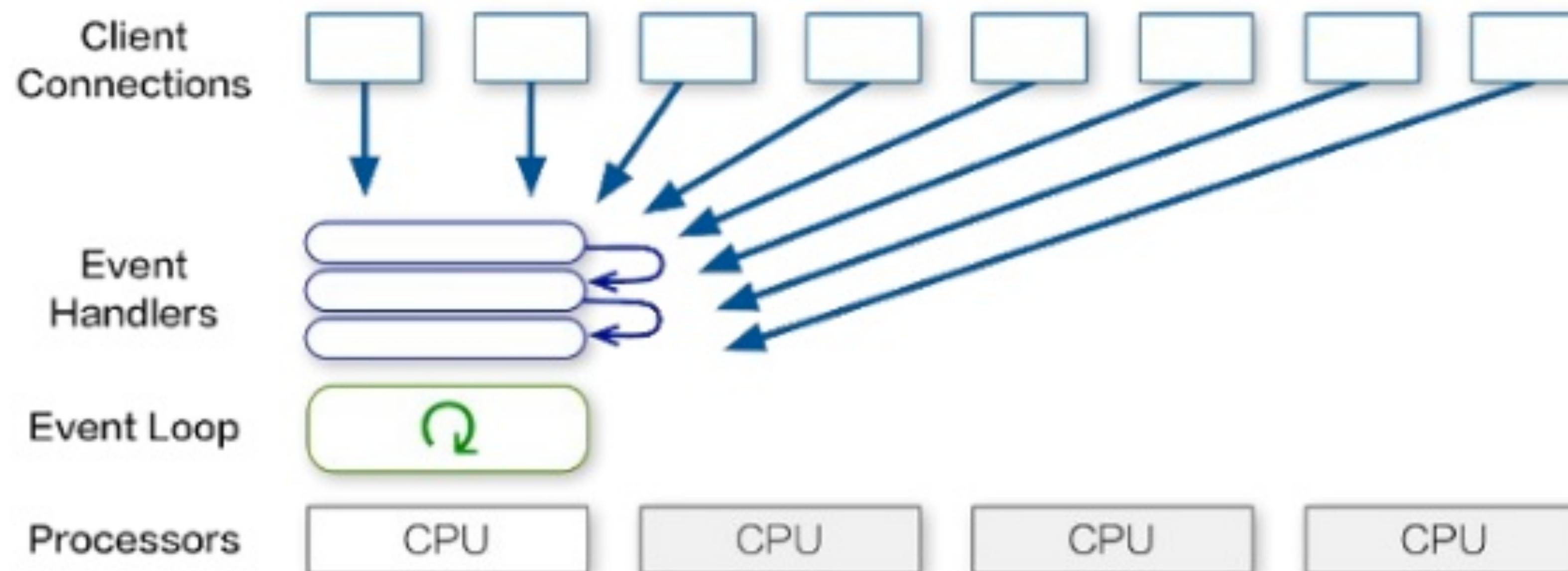
Multi process

- cluster

Node.js Production Ready

(Multi process)

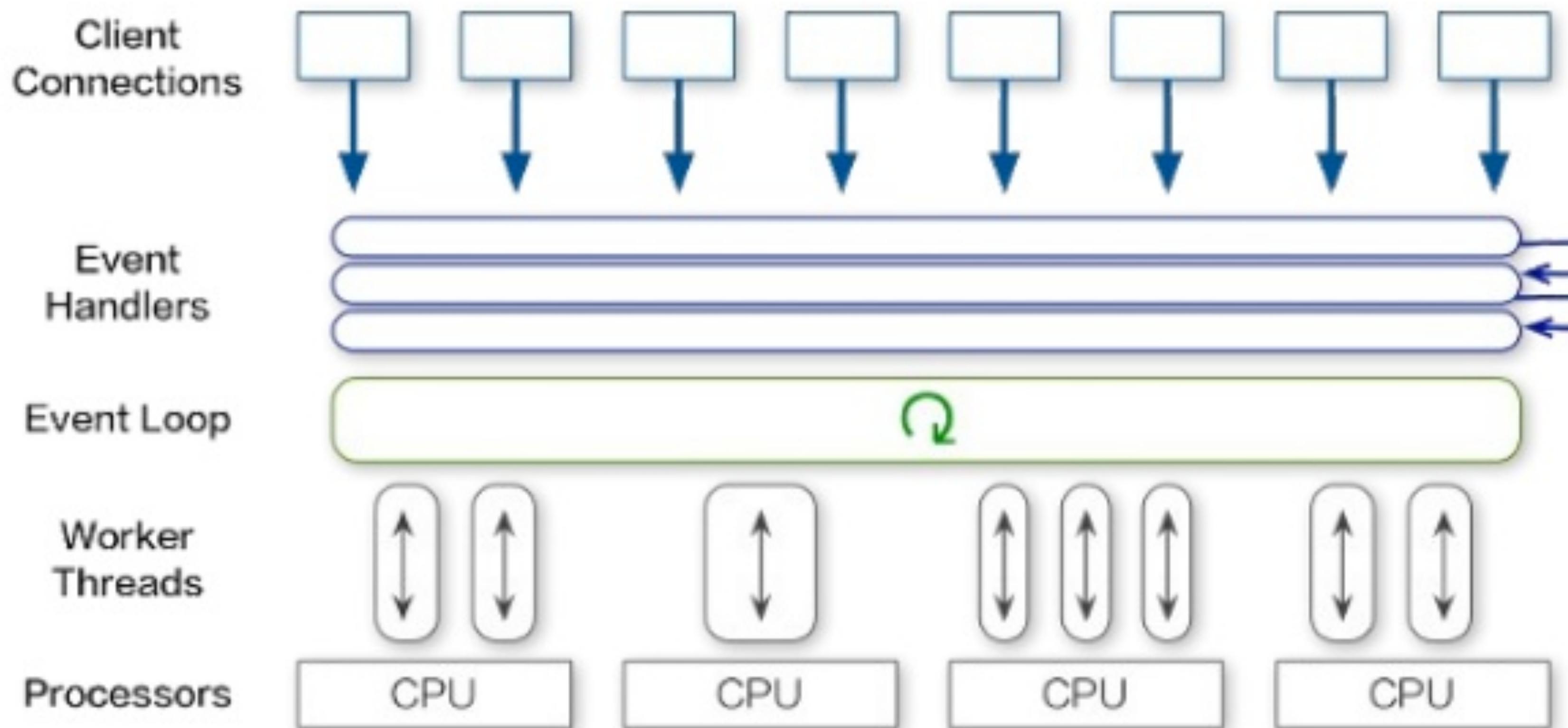
scaling node.js – cluster module



Node.js Production Ready

(Multi process)

scaling node.js - cluster module



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

Node.js Production Ready

(Multi process)

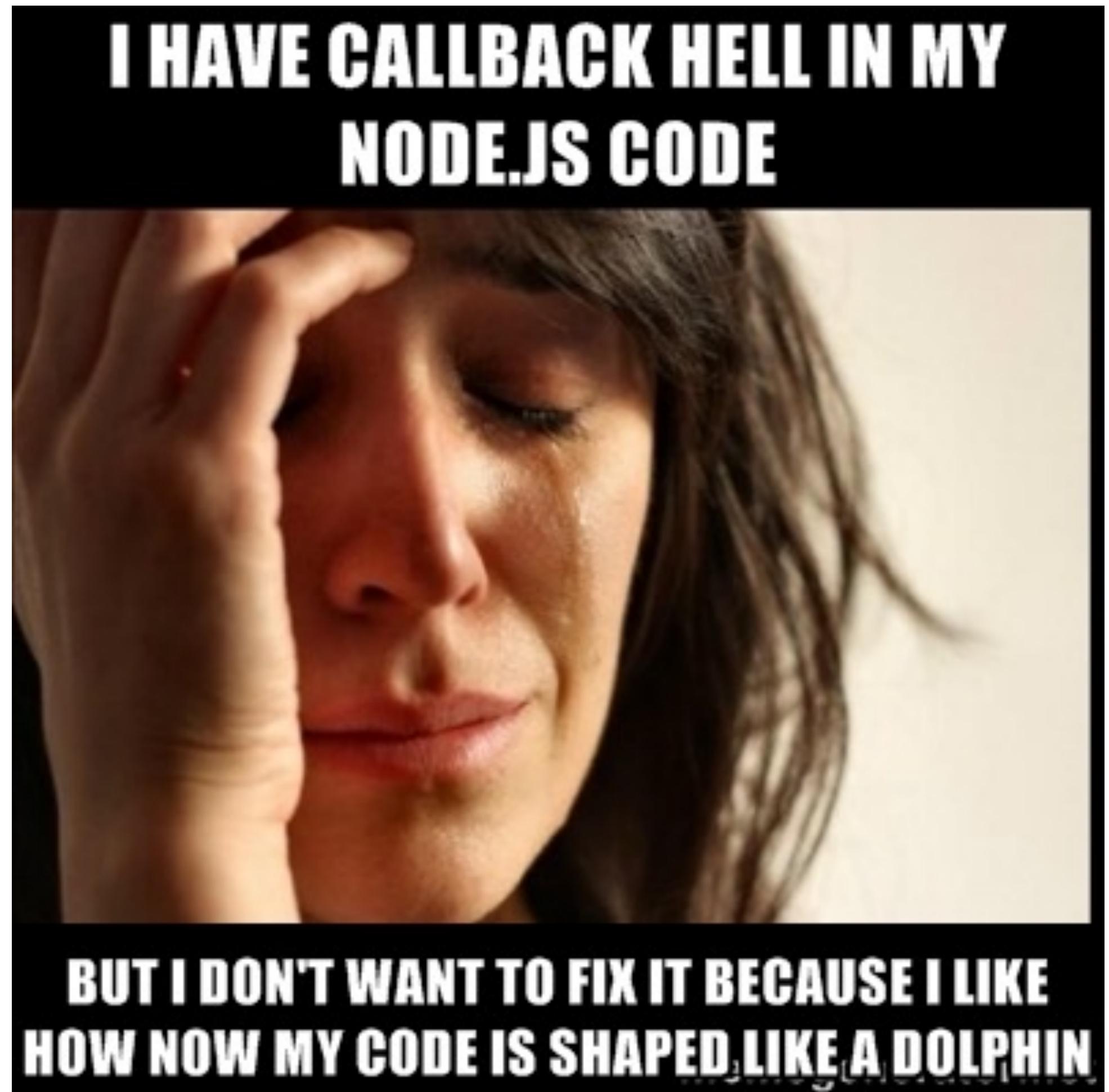
```
};

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

}
```

Node.js Production Ready (Callback hell)

```
1 app.get('/some_resources', function (req, res) {  
2   db.query('SELECT A ...', function (err, a) {  
3     if (err) return res.end(err);  
4  
5     db.query('SELECT B ... WHERE a=' + a, function (err, b) {  
6       if (err) return res.end(err);  
7  
8       db.query('SELECT C ... WHERE b=' + b, function (err, c) {  
9         if (err) return res.end(err);  
10  
11       db.query('SELECT D ... WHERE c=' + c, function (err, d) {  
12         if (err) return res.end(err);  
13  
14         res.end(d);  
15       });  
16     });  
17   });  
18 });  
19});
```



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

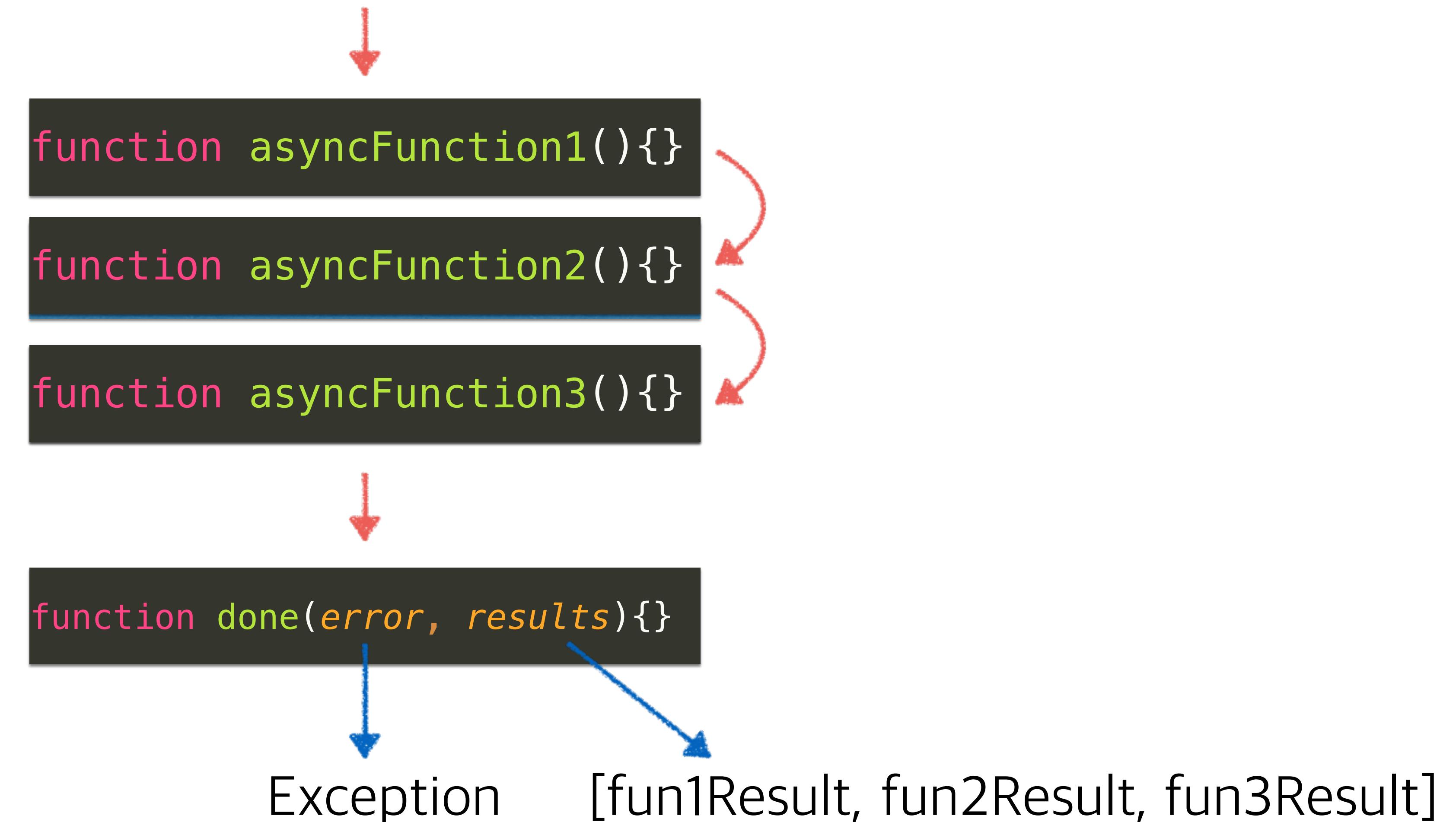
(Callback hell - `async.series`)

```
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

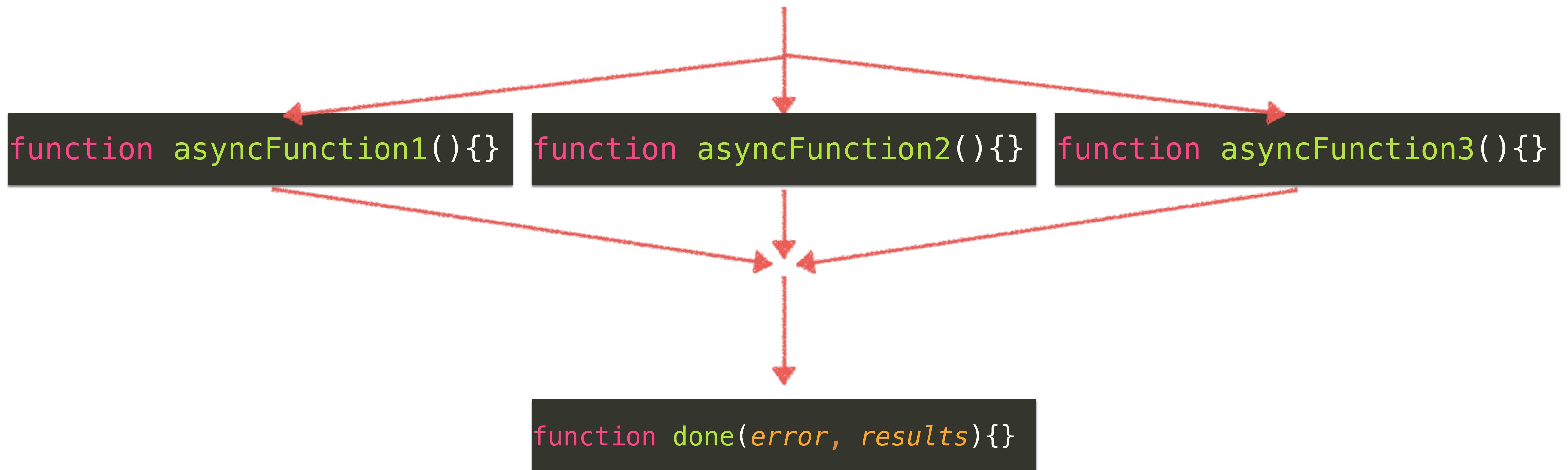
(Callback hell - `async.parallel`)

```
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

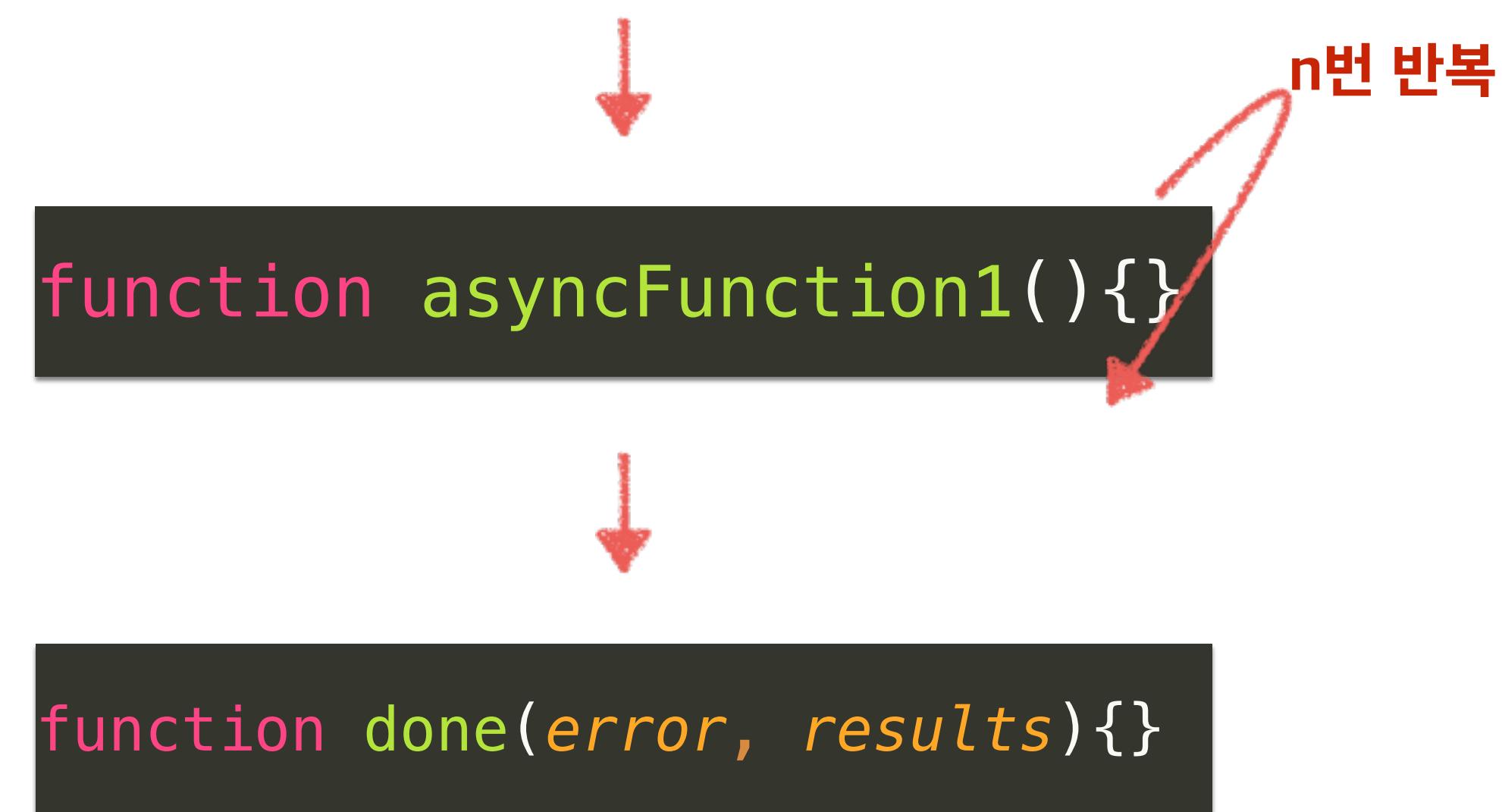
(Callback hell - `async.times`)

```
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

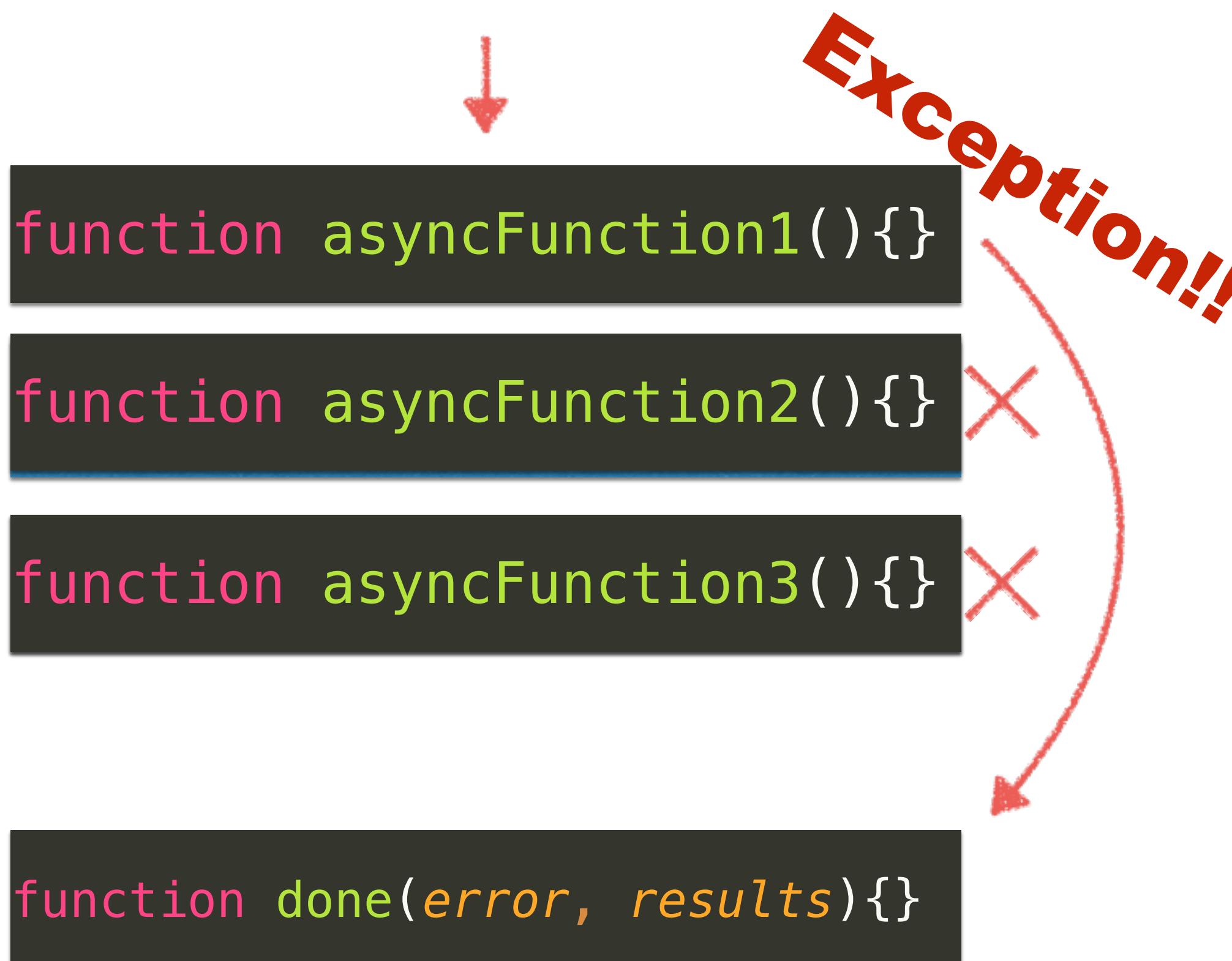
(Callback hell - `async.series`)

```
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!!');
```

Q & A