Introduction to node.JS

Richard Lee

Taipei Google Technology User Group

console.info(me);

- Richard Lee
- iOS / Rails / JavaScript developer
- Co-founder of Polydice, Inc.

- Email: dlackty@gmail.com
- Twitter: @dlackty

Node.js

- Event-driven I/O framework
- Usually known as server-side JavaScript
- Based on V8 JavaScript Engine by Google
- Great community supported

- However, only on Unix-like platforms

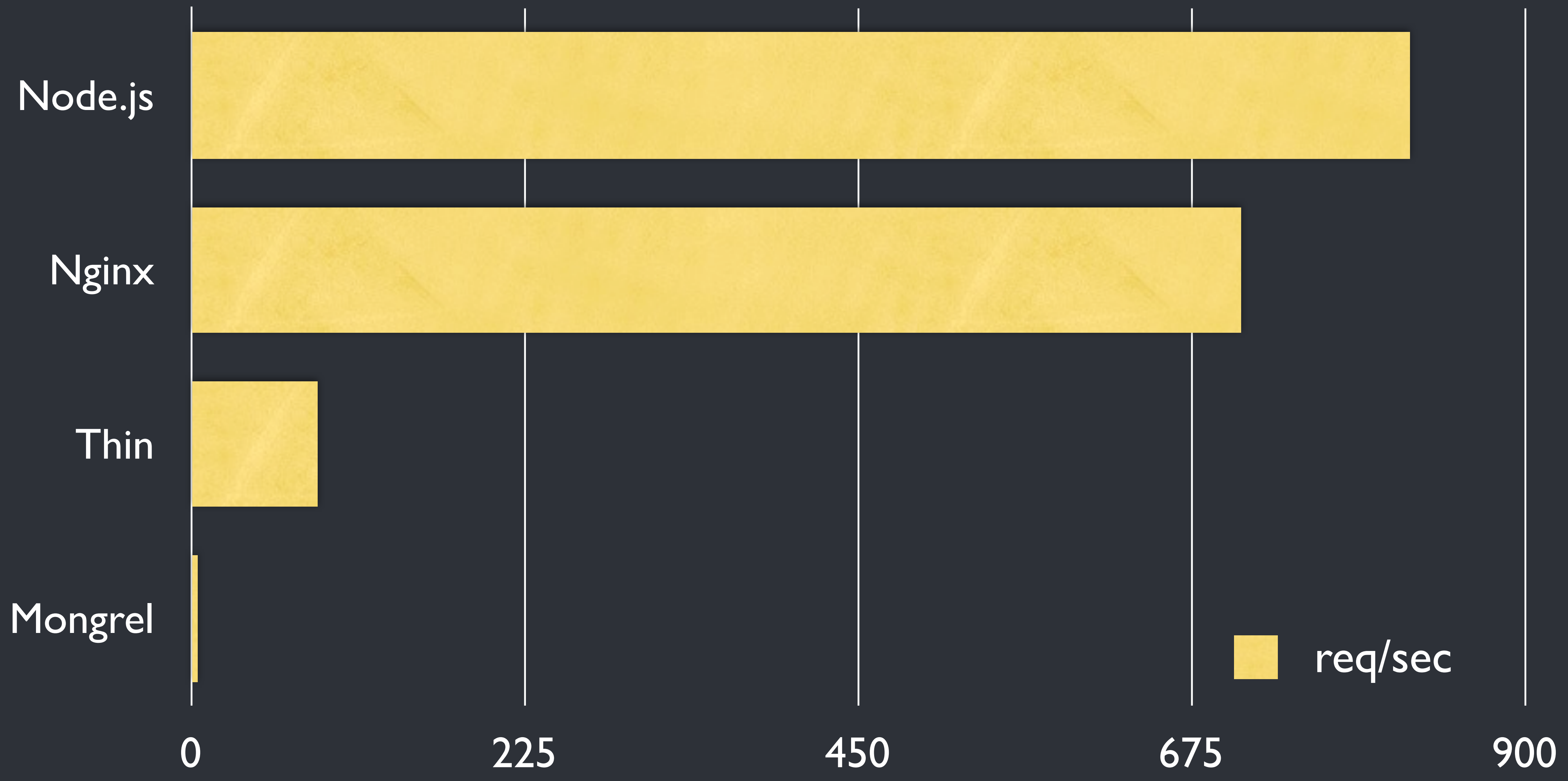
Event-driven I/O framework

- Everything in Node.js is asynchronous
- Doesn’t have to wait for slow file I/O or database operations to continue processing
- Make it insanely fast
- Handle millions of concurrent connections at once

- It’s also known as non-blocking server

Benchmark

100 concurrent clients
1 megabyte response



A synchronous example

```
data = readFromDatabase();
printData(data);

doSomethingUnrelated();
```

- The program get blocked when read from database
- Many CPU cycles are wasted

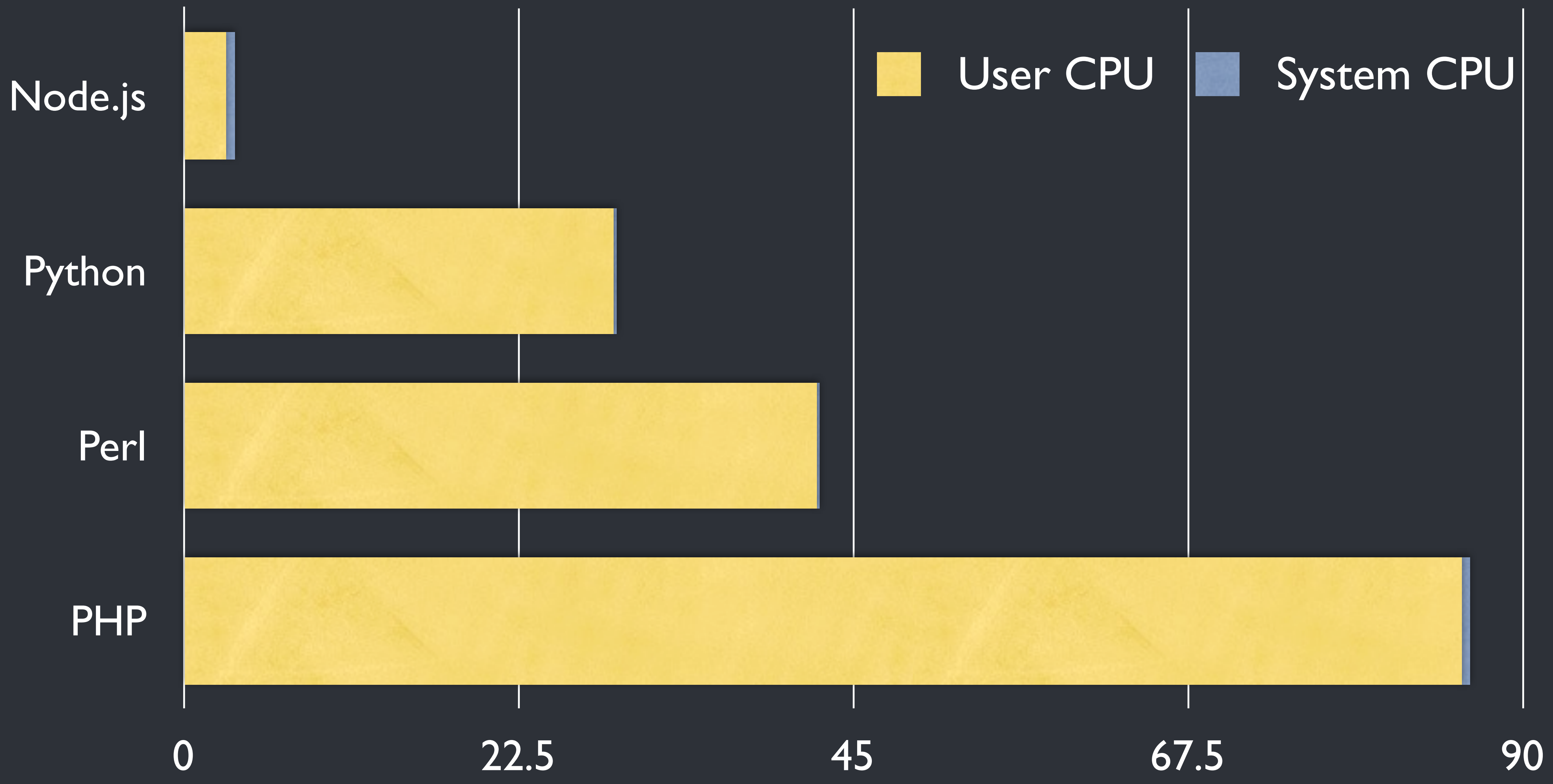
An asynchronous example

```
readFromDatabase(function(data) {
    printData(data);
});

doSomethingUnrelated();
```

- doSomethingUnrelated() get called immediately
- printData(data) will be called when finish reading
- Everything runs in parallel

Benchmark II



http://blog.famzah.net/2010/07/01/cpp-vs-python-vs-perl-vs-php-performance-benchmark/

What is not Node.js

- Node.js is not full stack Web framework
- No built-in MVC architecture
- No built-in database support
- No built-in URL routing system

- But all these thins can be found from modules

Event-loop basis

- All your codes run in an ”event-loop”
- Stop if no event listeners, and event emitters
- That is,
  - a loop waits for event
  - a loop run in single thread
  - but everything else is asynchronous

EventEmitter

- EventEmitter is the core component for Node.js
- Anything related to I/O wrapped with it
- Your own class usually inherits it

Using EventEmitter

```
server.on('connection', function (stream) {
  console.log('someone connected!');
});

server.once('connection', function (stream) {
  console.log('Ah, we have our first user!');
});
```

Create your own emitter

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

eventEmitter.on('someOccurence', function(message){
    console.log(message);
});

eventEmitter.emit('someOccurence', 'Something
happened!');
```

Some pitfalls

- Remember that event-loop is single thread
- Always handle event efficiently, otherwise
  - Event callbacks will get queued in order
  - User response time becomes longer
  - The connection won’t drop, though

Bad example

```
ee = new require('events').EventEmitter();

die = false;
ee.on('die', function() { die = true; });

setTimeout(function() { ee.emit('die'); }, 100);
while(!die);

console.log('done'); // never be called
```

Fix single thread issue

- Spawn new thread if needed
- The communication is also down by events
- Some principle to follow
  - Use event callbacks
  - Avoid share states variables
  - I/O should be handled by built-in function calls

Good, so how?

- Coding in JavaScript are natively asynchronous
- Always need to writes callback functions for I/O
- Organize your codes as
  - I/O components - In event-loop
  - Computational components - Spawn workers

Thus, it's quite common to see

```
doSomething(data, function(data) {
    doAnotherThing(data, function(response) {
        doYetAnotherThing(data, function() {
          doJustYetAnotherThing(data, function()) {
              // kinda of crazy
          });
        });
    });
});

doSomethingUnrelated();
```

HTTP Server

- Node.js has built-in HTTP server library
- Low-level design instead of high level abstraction
- Supports HTTP 1.1
  - Thus, the network connection will persist

HTTP Hello World example

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

http.createServer(function(request, response) {
  request.writeHead(200, {'Content-Type': 'text/
plain'});
  request.end('Hello World\n');
}).listen(8124, "127.0.0.1");

console.log('Server running at http://
127.0.0.1:8124/'
```

HTTP static file server

```
http.createServer(function(request, response) {
    var uri = url.parse(request.url).pathname;
    var filename = path.join(process.cwd(), uri);

    // Then read file

}).listen(8080);
```

HTTP static file server (cont'd)

```
path.exists(filename, function(exists) {
    if(!exists) {
         response.sendHeader(404, {"Content-Type": "text/plain"});
         response.write("404 Not Found\n");
         response.close();
         return;
    }
    fs.readFile(filename, "binary", function(err, file) {
         if(err) {
              response.sendHeader(500, {"Content-Type": "text/plain"});
              response.write(err + "\n");
              response.close();
              return;
         }
         response.sendHeader(200);
         response.write(file, "binary");
         response.close();
    });
});
```

Easy comet example

```
var http = require('http');
var spawn = require('child_process').spawn;

http.createServer(function(request, response) {
  response.writeHead(200, {'Content-Type': 'text/plain'});
  var child_process = spawn('tail', ['-F', '/var/log/system.log']);

  request.connection.on('end', function() {
    child_process.kill();
  });

  child_process.stdout.on('data', function(data) {
    console.log(data.toString());
    response.write(data);
  });

}).listen(8080);
```

Long polling example

```
var http = require("http");
var requests = [];

http.createServer(function(request, response) {
    // store the response so we can respond later
    requests.push(response);
}).listen(8000);

setInterval(function() {
    // respond to each request
    while (requests.length) {
        response = requests.shift();
        response.writeHead(200, { "Content-Type": "text/plain" });
        response.end("Hello, World!");
    }
}, 2000);
```

Great community support

- Reminds me of Ruby community
- GitHub project wiki has lots of resources

- Most of the related projects are on GitHub
- npm is a package manager for node

Some books under working

- Node.js Up and Running
- Mastering Node.js
- The Node Beginner Book

How to Node?

- Pick one of the book or tutorials online
- Do examples of event-driven programming model
- Start to code your project
- Got problem? Read the API, and move on

- Finally, you learn how to node

Thanks for your hearing!