The What, Why and How of NodeJS

An introduction for Rubyists

Glen Mailer

- Web Developer for about 10 years
- JavaScript, Ruby, Python, PHP
- Working for SkyBet.com
  - Mainly a PHP Web Stack
  - Increasing use of NodeJS

The What and Why…

Reactor Pattern

in **JavaScript**

on **V8**

Reactor Pattern

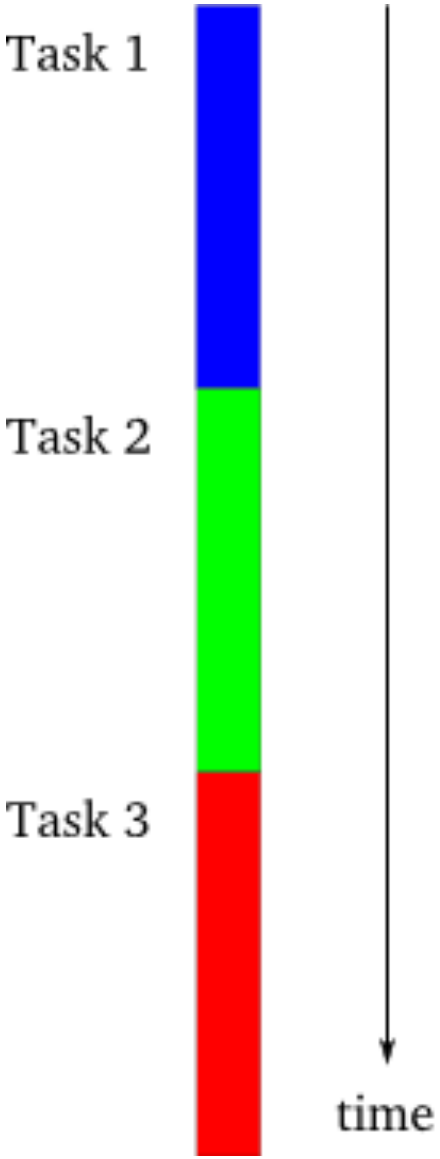
Python: Twisted

Ruby: EventMachine

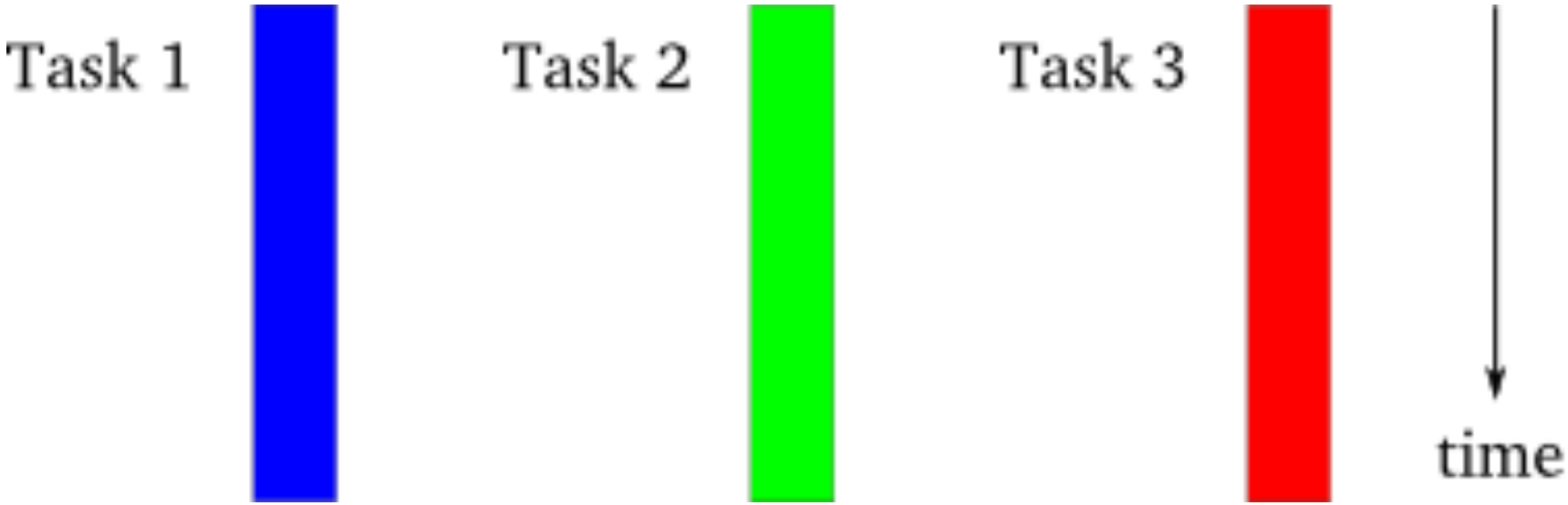
Pretty much any GUI

Do something

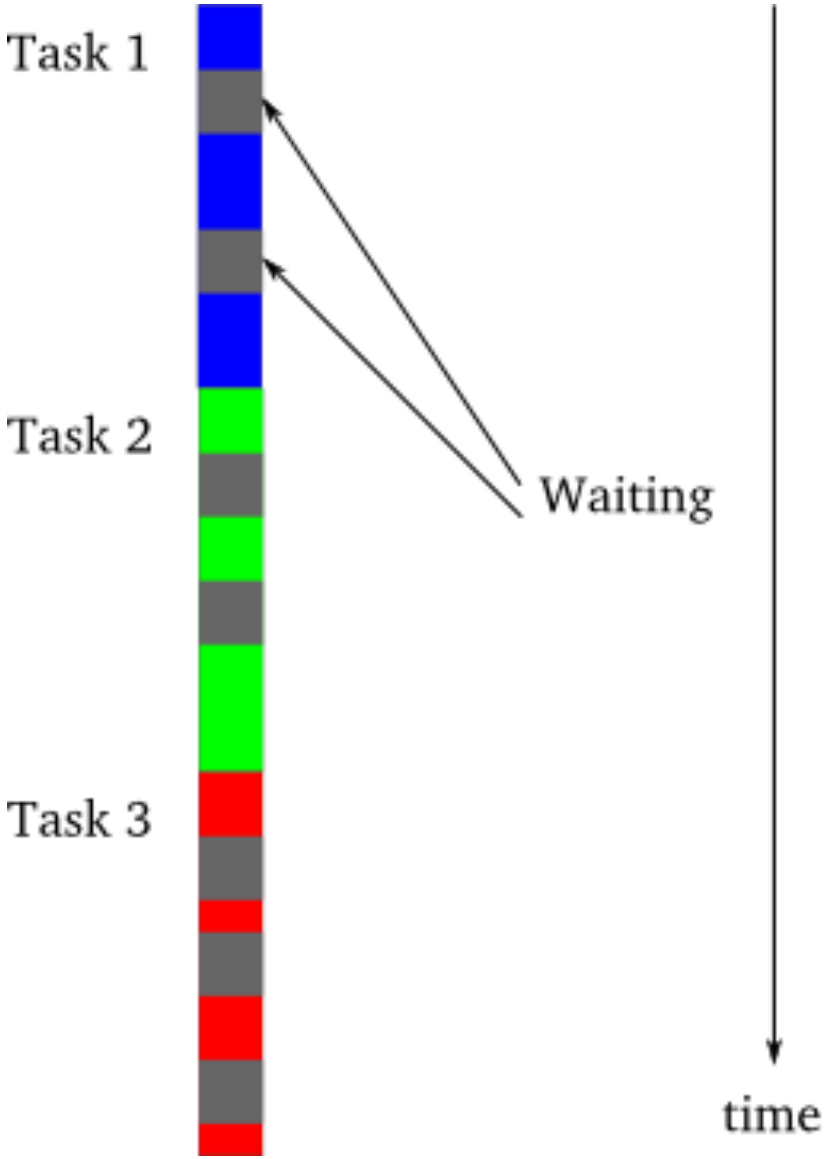
react after IO



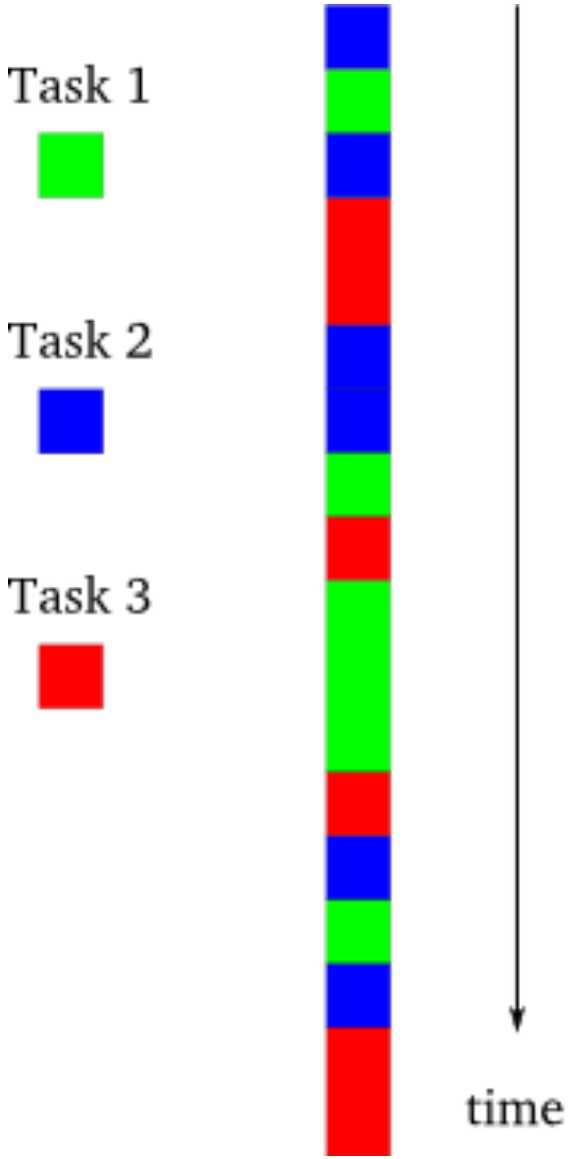
Images stolen from http://krondo.com/?p=1209



True concurrency



Wasted time waiting for IO



Reactor Pattern approach

Unusual program flow

Don’t block the loop!

JavaScript

Loose Dynamic Typing

Prototypal Inheritance

Lots and lots of closures

No standard library

Everyone knows JavaScript…

“It is the language that people use without bothering to learn it first”

Douglas Crockford, Javascript: The Good Parts

V8

Began the JS arms race

JIT Compilation

ECMAScript 5

“use strict”

Array.forEach

Object.keys

Function.bind

Object.defineProperty

1 GB Heap Limit!

The How…

The REPL

```
> node
> 1 + 1;
2
> [1, 2, 3].join(" ")
'1 2 3'
> "string".method()
TypeError: Object string has no method 'method'
    at [object Context]:1:10
    at Interface.<anonymous> (repl.js:171:22)
    at Interface.emit (events.js:64:17)
    at Interface._onLine (readline.js:153:10)
    at Interface._line (readline.js:408:8)
```

The Node REPL

Modules

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

var async = require('async');

var module = require('./my-module');
var magic = require('./magic')('stuff');
```

Requiring other modules

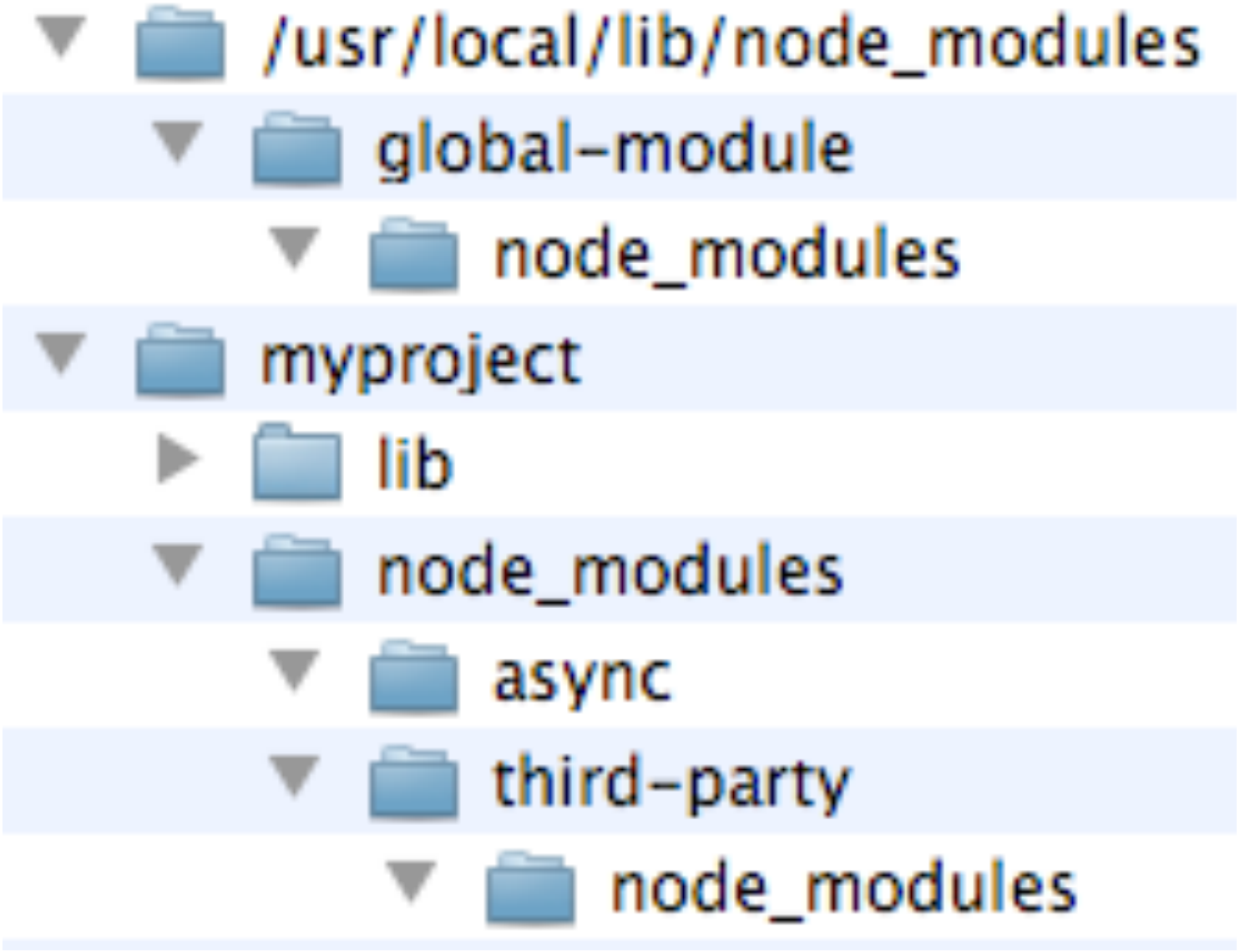
```
/* my-module.js */
exports.publicFunction = function() {
    return 'something';
}

exports.anotherFunction = function() {
    return 'else';
}
```

Defining your own modules

```
/* magic.js */
module.exports = function(arg) {
    return {
        func: function() { return arg; }
    };
}
```

Re-defining the exports object



Load path using node_modules

npm

http://npmjs.org

npm install (-g) <package>

Callbacks

```
function fetch_all(database, table, callback) {
    db.connect(function(err) {
        if (err) {
            callback(err);
            return;
        }
        db.useDatabase(database, function(err) {
            if (err) {
                callback(err);
                return;
            }
            db.query("SELECT * FROM " + table, callback);
        });
    });
}
```

Using callbacks to wait for IO

```
function expensive_operation(a, b, callback) {
    var c = a + b;
    process.nextTick(function() {
        callback(null, c);
    })
}

expensive_operation(3, 4, function(err, result) {
    if (err) {
        console.warn(err);
    } else {
        console.log(result);
    }
});
```

process.nextTick to release the loop

EventEmitter

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

var emitter = new events.EventEmitter();

emitter.on('event', console.log);
emitter.on('event', function(arg) {
    console.log("Multiple listeners");
});
emitter.on('error', console.warn);

emitter.emit('event', 'argument');
emitter.emit('error', 'whoops');
```

EventEmitter example

The basis of most stdlib modules

Prototypal Inheritance

Not Classical Inheritance

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

function Animal(legs) {
    this.legs = legs;
}

util.inherits(Dog, Animal);
function Dog(name) {
    this.constructor.super_.call(this, 4);
    this.name = name;
}
```

Defining “classes”

```
var fido = new Dog('fido');

console.log(fido.name === 'fido');
// => true
console.log(fido.legs === 4);
// => true
```

Instance and properties

```
Dog.prototype.bark = function() {
    return this.name + ' barks';
}
console.log(fido.bark());
// => 'fido barks'
```

Method on a “class”

```
Animal.prototype.speed = function() {
    return (this.legs * 5) + 'm/s';
}
console.log(fido.speed());
// => 20m/s
```

Method on a parent “class”

```
i = [];
for (var f in fido) {
    i.push(f);
}
console.log(i);
// => ['legs', 'name',
//     'bark', 'speed']
```

Iterating over an instance

```
Object.defineProperty(
    Animal.prototype, 'speed',
    {
        value: function() {
            return (legs * 5) + 'm/s';
        },
        enumerable: false
    }
);
```

Controlling iteration behaviour

HTTP

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

http.createServer(function(req, resp) {
    req.setEncoding('utf8');
    var data = '';
    req.on('data', function(chunk) { data += chunk; });
    req.on('end', function() {
        console.log("Request for: " +
                    req.method + " " + req.url);
        console.log("Body: " + data);
        resp.writeHead(418, {'Content-Type': 'text/plain',
                             'Content-Length': 12});
        resp.write("I'm a teapot");
        resp.end();
    });
}).listen(8888);
```

Simple HTTP Server

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

var options = { host: "localhost", port: 8888,
                method: "POST", path: "/path" }

var req = http.request(options, function(resp) {
    resp.setEncoding('utf8');
    console.log(resp.statusCode);
    var data = '';
    resp.on('data', function(chunk) { data += chunk });
    resp.on('end', function() { console.log(data) });
});
req.write("POST BODY");
req.end();
```

Simple HTTP Client

Control Flow

async

https://github.com/caolan/async

```
function fetch_all(database, table, callback) {
    db.connect(function(err) {
        if (err) {
            callback(err);
            return;
        }
        db.useDatabase(database, function(err) {
            if (err) {
                callback(err);
                return;
            }
            db.query("SELECT * FROM " + table, callback);
        });
    });
}
```

Function nesting without async

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

function fetch_all_neatly(database, table, callback) {
    async.series({
        connect: db.connect.bind(db),
        use:     db.useDatabase.bind(db, database),
        query:   db.query.bind(db, "SELECT * FROM " + table)
    }, function(err, result) {
        callback(err, result.query);
    });
}
```

Tidied up with async

```
async.parallel({
    'socket':    initSocket,
    'amqp':      initAmqp,
    'db':        initDB,
    'interface', initInterface,
    'plugins',   initPlugins
},
function(err, results) {
    if (err) {
        console.warn("Initialisation failed: " +
                     err.message);
        process.exit(1);
    }
    startWorker(results);
});
```

Parallel initialisation tasks

```
async.forEach(files,
    function iterator(file, next) {
        file.save(next);
    },
    function finished(err) {
        console.log("All files have been saved");
    }
);

async.forEachSeries(files,
    //...
);
```

Iterating and calling async functions

async.map[Series]

async.filter[Series]

async.reduce[Series]

async.detect[Series]

async.queue

async.auto

Web Frameworks

Connect

http://senchalabs.github.com/connect/

```
var connect = require('connect');
var app = connect();

app.use(connect.logger('dev'));
app.use(connect.responseTime());
app.use(connect.basicAuth('user', 'pass'));
app.use(connect.directory(ROOT, {icons: true}));
app.use(connect.static(ROOT));

app.listen(8888);
```

Simple connect example

express

http://expressjs.com

```
var app = module.exports = express.createServer();
app.configure(function(){
  app.set('views', __dirname + '/views');
  app.set('view engine', 'jade');
  app.use(express.bodyParser());
  app.use(express.methodOverride());
  app.use(app.router);
  app.use(express.static(__dirname + '/public'));
});
```

Express app initialisation

```
app.configure('development', function(){
  app.use(express.errorHandler({
    dumpExceptions: true,
    showStack: true
  }));
});

app.configure('production', function(){
  app.use(express.errorHandler());
});
```

Express understands environments

```
app.get('/', function(req, res){
  res.render('index', { title: 'Express' });
});

app.param('userName', function(req, res, next, username) {
  User.lookup(username, function(err, user) {
    req.user = user;
    next(err);
  });
});
app.get('/profile/:userName', function(req, res) {
  res.render('profile', { title: 'Profile',
                          user: req.user });
});
```

Handling requests

```
// layout.jade
!!!
html
  head
    title= title
    link(rel='stylesheet',
         href='/stylesheets/style.css')
  body!= body

// profile.jade
h2 Welcome #{user.name}
```

Templating with Jade

```
<html>
    <head>
        <title>Profile</title>
        <link href="/stylesheets/style.css"
              rel="stylesheet">
    </head>
    <body>
        <h2>Welcome else</h2>
    </body>
</html>
```

The output from the Jade template

Testing

```
exports.Calculator  = Calculator
function Calculator(initial) {
    this.number = initial || 0;
    this.operation = null;
}
Calculator.prototype.add = function() {
    this.operation = function(a, b) { return a + b };
}
Calculator.prototype.minus = function() {
    this.operation = function(a, b) { return a - b };
}
```

Sample code we'll be testing

```
Calculator.prototype.type = function(number) {
    this.number = this.operation(this.number, number);
}
Calculator.prototype.equals = function(callback) {
    var answer = this.number;
    process.nextTick(function() {
        callback(null, answer)
    });
}
```

With some contrived asynchronous aspect

Scenarios to Test

- New instance has methods:
  - add
  - minus
  - type
  - equals
- Can initialise with default value
- Can calculate: 2 + 2 = 4 + 2 = 6

Vows

http://vowsjs.org

```
vows.describe("Calculator").addBatch({
    "new Calculator()": {
        topic: new calculator.Calculator,
        "has add() method": function(calc) {
            assert.equal(typeof calc.add, 'function');
            assert.equal(calc.add.length, 0);
        }
    }
}).export(module);
```

Basic Vows usage

```
function hasMethod(name, length) {
    length = length || 0;
    return function(calc) {
        assert.equal(typeof calc[name], 'function');
        assert.equal(calc[name].length, length);
    }
}

    "has minus() method":    hasMethod('minus'),
    "has type(num) method":  hasMethod('type', 1),
    "has equals(cb) method": hasMethod('equals', 1)
```

Define your own sugar

```
"new Calculator(1).equals()": {
    topic: function() {
        new calculator.Calculator(1).equals(this.callback);
    },
    "should callback 1": function(answer) {
        assert.equal(answer, 1);
    }
},
```

Async testing with Vows

```
"2 + 2": {
    topic: function() {
        var calc = new calculator.Calculator();
        return calc.type(2).add().type(2);
    },
    "=": {
        topic: function(calc) {
            calc.equals(this.callback);
        },
        "4": assertEquals(4),
        "+ 2 =": {
            topic: function(four, calc) {
                calc.add().type(2).equals(this.callback);
            },
            "6": assertEquals(6)
        }
    }
}
```

Nested async contexts with Vows

Vows Summary

Upsides

- Expressive DSL-like syntax
- Good range of formatters
- Coverage via jsCoverage

Downsides

- Unusual execution model
- Forced to separate evaluation & assertion
- Topic not clean per vow
- Only one formatter per run
- Commas!

nodeunit

https://github.com/caolan/nodeunit

```
exports['instance methods'] = function (test) {
    var calc = new calculator.Calculator();
    test.expect(8);
    test.equal(typeof calc.add, 'function');
    test.equal(calc.add.length, 0);
    test.equal(typeof calc.minus, 'function');
    test.equal(calc.minus.length, 0);
    test.equal(typeof calc.type, 'function');
    test.equal(calc.type.length, 1);
    test.equal(typeof calc.equals, 'function');
    test.equal(calc.equals.length, 1);
    test.done();
};
```

Basic nodeunit usage

```
exports['default value of 1'] = function (test) {
    test.expect(2);
    var calc = new calculator.Calculator(1);
    calc.equals(function(err, value) {
        test.ifError(err);
        test.equal(value, 1);
        test.done();
    });
};
```

Async testing with nodeunit

```
exports['2 + 2 = 4 + 2 = 6'] = function (test) {
    test.expect(4);
    var calc = new calculator.Calculator();
    calc.type(2).add().type(2);
    calc.equals(function(err, value) {
        test.ifError(err);
        test.equal(value, 4);
        calc.add().type(2);
        calc.equals(function(err, value) {
            test.ifError(err);
            test.equal(value, 6);
            test.done();
        })
    })
};
```

Nested async with nodeunit

Nodeunit Summary

Upsides

- Lightweight
- Track number of assertions
- Doesn't force a particular style
- Range of formatters

Downsides

- No code coverage
- Have to roll your own structure
- No nested contexts

nodespec

https://github.com/glenjamin/nodespec

```
nodespec.describe("Calculator", function() {
    this.subject("calc", function() {
        return new calculator.Calculator(this.initial);
    });
    this.describe("constructor", function() {
        this.example("should have add() method", function() {
            this.assert.equal(typeof this.calc.add, 'function');
            this.assert.equal(this.calc.add.length, 0);
        });
    });
});
```

Testing with nodespec

```
function shouldHaveMethod(group, name, length) {
    length = length || 0;
    group.example("should have "+name+"() method", function() {
        this.assert.equal(typeof this.calc[name], 'function');
        this.assert.equal(this.calc[name].length, length);
    });
}

    shouldHaveMethod(this, 'minus');
    shouldHaveMethod(this, 'type',   1);
    shouldHaveMethod(this, 'equals', 1);
```

Sugaring tests with nodespec

```
this.subject("calc", function() {
    return new calculator.Calculator(this.initial);
});
this.describe("constructor", function() {
    this.example("default 1, equals should callback 1", function(test) {
        test.expect(2);
        test.initial = 1;
        test.calc.equals(function(err, answer) {
            test.assert.ifError(err);
            test.assert.equal(answer, 1);
            test.done();
        });
    });
    shouldEqualGivenDefault(this, 7);
    shouldEqualGivenDefault(this, -5);
});
```

Async testing and subjects with nodespec

```
this.describe("Calculating", function() {
    this.example("2 + 2 = 4 + 2 = 6", function(test) {
        test.expect(4);
        test.calc.type(2).add().type(2);
        test.calc.equals(function(err, answer) {
            test.assert.ifError(err);
            test.assert.equal(answer, 4);
            test.calc.add().type(2);
            test.calc.equals(function (err, answer) {
                test.assert.ifError(err);
                test.assert.equal(answer, 6);
                test.done();
            })
        });
    });
});
```

Chaining async with nodespec

Why nodespec?

- Take best aspects from vows and nodeunit
- Stick to native nodeJS stdlib assertions
- Some RSpec-esque sugar
  - Nested contexts
  - Before/after blocks
  - Lazily evaluated subjects

Still work in progress

Done

- Nested contexts
- Friendly sync/async syntax
- Before/after hooks
- Lazy subjects
- Basic formatter
- Cukes!

Todo

- More formatters – including multiple outputs from one run
- Command-line runner
- Code coverage

Questions?

Your Turn

Challenge: Parallel Searching

- Install node + npm, and clone https://github.com/glenjamin/node-intro-challenge
- Make requests in **parallel** then output first 100 chars
- Modify to make requests **one at a time**
- Now go back to parallel, and implement:

```
function search(term, callback(err, results))
results = {google: {data: 'html…', time: 1.34},
           yahoo: {data: 'html…', time: 2.01}, … }
```

Further Reading

- MDC – JavaScript 1.5 Reference https://developer.mozilla.org/en/JavaScript/Reference
- NodeJS homepage http://nodejs.org/
- My blog post on prototypes http://blog.glenjamin.co.uk/i-think-i-finally-really-understand-javascrip