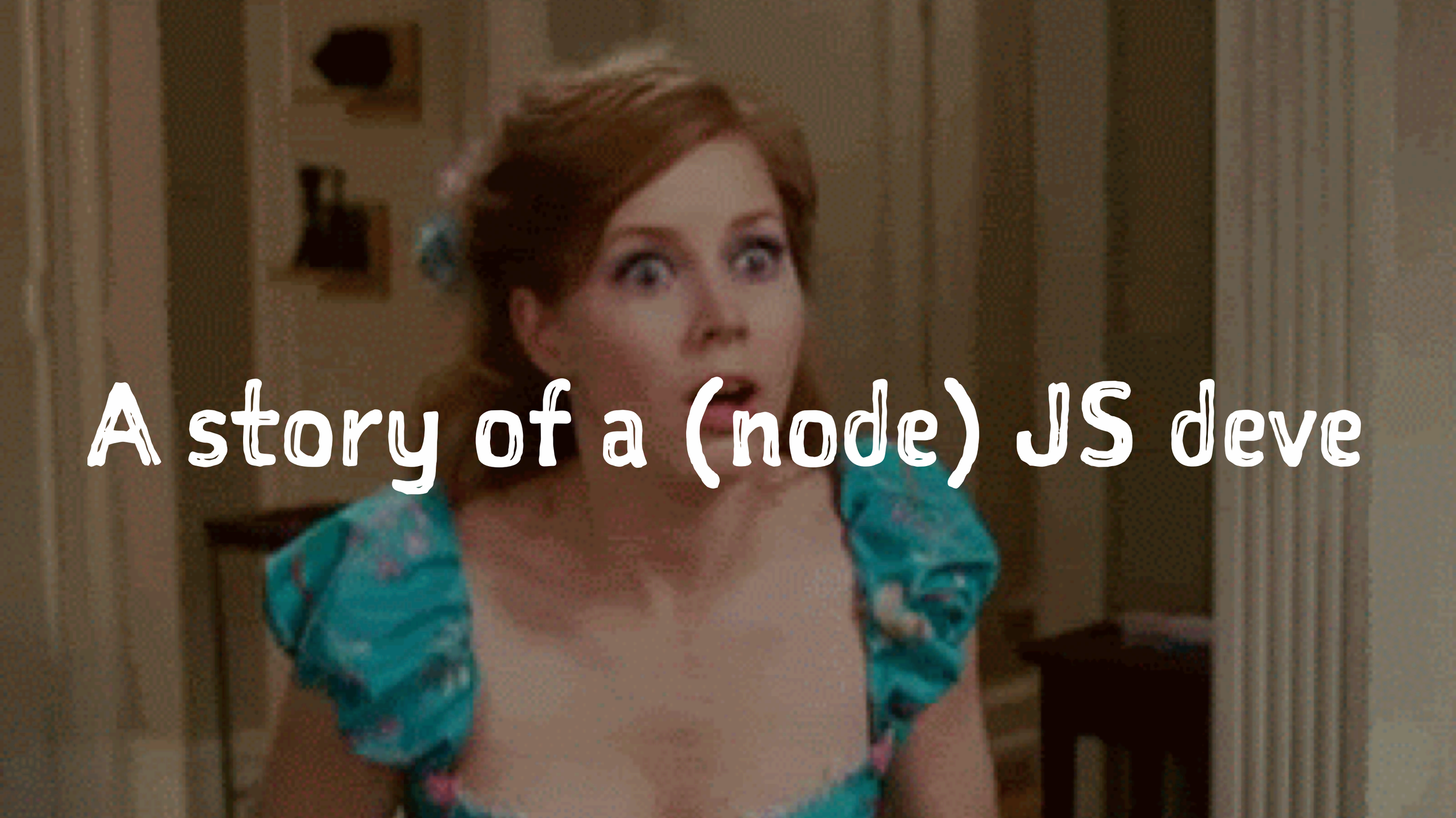
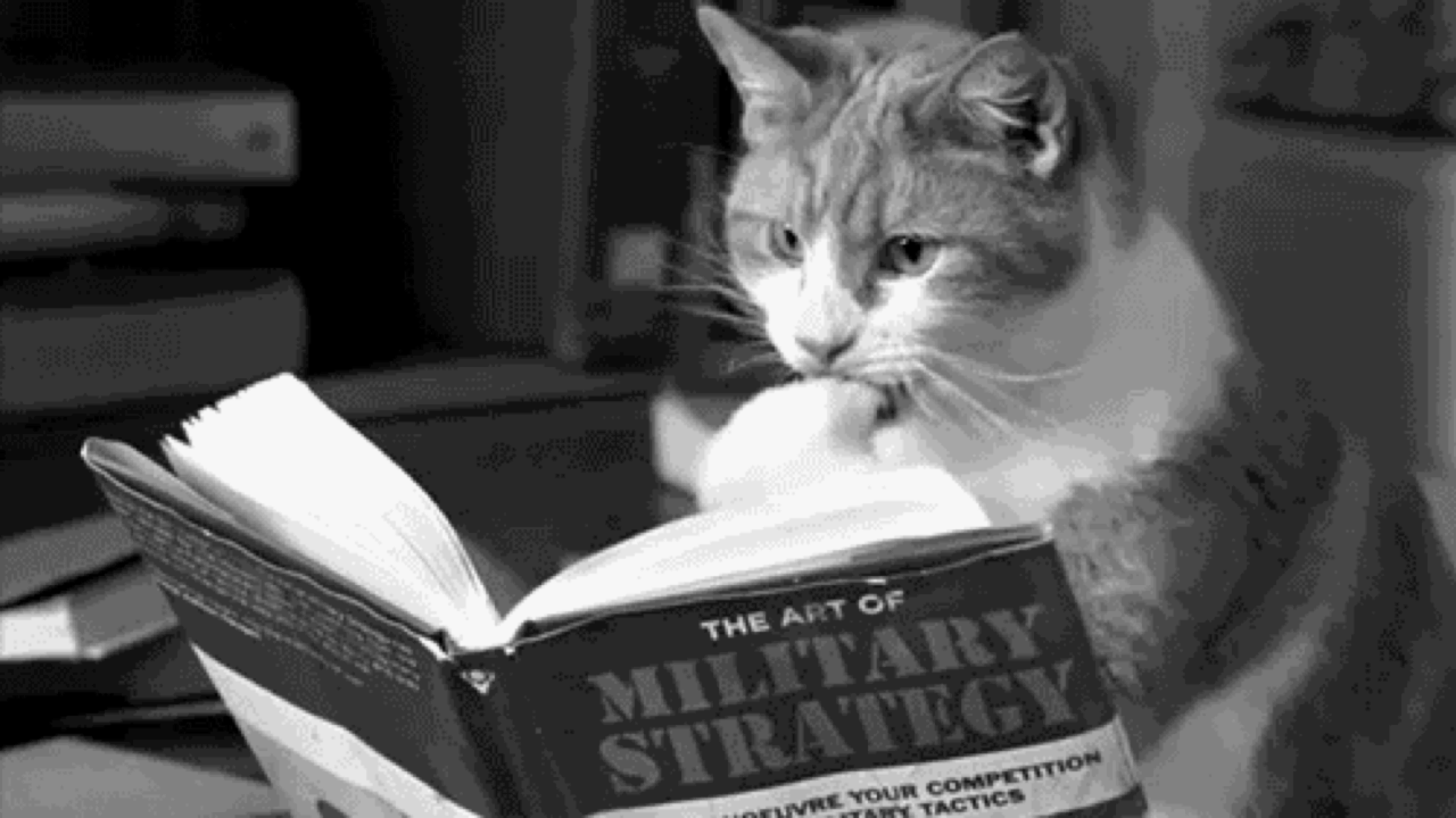
Let me tell you a story...

A story of a developer...


A story of a (node) JS deve

Simple


THE ART OF
MILITARY
STRATEGY

hmm.. that's very easy!

```
function read(filename){
  return fs.readFileSync(filename
}
```

Okies....sau the file is like

```
function read(filename, callback){
  fs.readFile(filename, 'utf8', function (err, res){
    if (err) {
        return callback(err);
    }
    callback(null,res);
  });
}
```

OK, not bad...now process that file.

```
function process(file){
  /* Some processing stuff */
}

function read(filename, callback){
  fs.readFile(filename, 'utf8', function (err, res){
    if (err) {
        return callback(err);
    }
    try {
      callback(null, process(res));
    } catch (ex) {
      callback(ex);
    }
```

```
function read(filename, callback){
  fs.readFile(filename, 'utf8', function (err, res){
    if (err) {
        return callback(err);
    }
    try {
      res = process(res);
    } catch (ex) {
      return callback(ex);
    }
    callback(null, res);
  });
}
```

Time passes by....

```
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);
        });
      });
    });
  });
});
```

So, how do we avoid callback hell?

- Name your functions.
- Keep your code shallow.
- Modularize!
- Binding `this`

When you know...then you know!

Let us make a promise!

- fulfilled
- rejected
- pending
- settled

```
var promise = new Promise(function(resolve, reject) {
  // Some async...
  if (/* Allz well*/) {
    resolve("It worked!");
  }
  else {
    reject(Error("It did not work :'("));
  }

promise.then(function(result) {
  console.log(result); // "It worked!"
}, function(err) {
  console.log(err); // Error: "It don not work :'("
})
```

```
async1().then(function() {
  return async2();
}).then(function() {
  return async3();
}).catch(function(err) {
  return asyncHeal1();
}).then(function() {
  return asyncHeal4();
}, function(err) {
  return asyncHeal2();
}).catch(function(err) {
  console.log("Ignore them");
}).then(function() {
```

lie-fs

```
$ npm install lie-fs
```

```
fs.stat('/tmp', function(err,data){
    if(!err){
        console.log(data);
    } else {
        console.error(err);
    }
```

```
var fsp = require('lie-fs');
```

async

```
async.map(['file1','file2','file3'], fs.stat, function
    // results is now an array of stats for each file
});

async.filter(['file1','file2','file3'], fs.exists, fun
    // results now equals an array of the existing fil
});

async.parallel([
    function(){ ... },
```


Happ(Y)ily ever after...

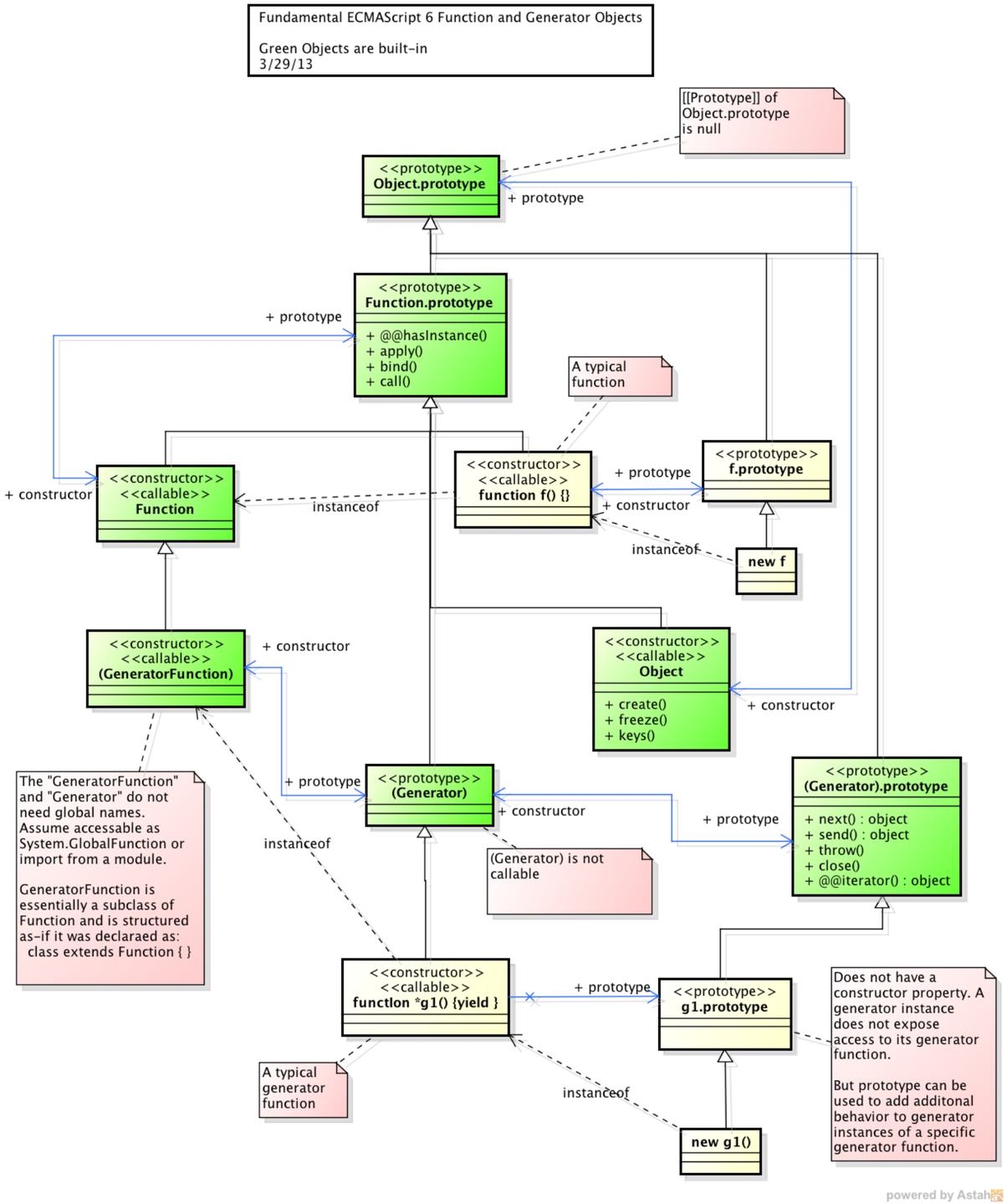
Streams are broken, callbacks are not great to work with, errors are vague, tooling is not great, community convention is sort of there..

- you may get duplicate callbacks
- you may not get a callback at all (lost in limbo)
- you may get out-of-band errors
- emitters may get multiple “error” events
- missing “error” events sends everything to hell
- often unsure what requires “error” handlers
- “error” handlers are very verbose
- callbacks suck

Welcome to the new gen of

"Harmony" is the name TC3

[[harmony:generators]]



- First-class coroutines.
- Represented as objects encapsulating suspended execution contexts.
- Prior art: Python, Icon, Lua, Scheme, Smalltalk.


nick

Every generator object has the following internal properties:

* [[Prototype]] : the original value of Object.prototy
* [[Code]] : the code for the generator function body.
* [[ExecutionContext]] : either null or an execution c
* [[Scope]] : the scope chain for the suspended execut
* [[Handler]] : a standard generator handler for perfo

Show me the code!

```
function *Counter(){
    let n = 0;
    while(1<2) {
      yield n;
      n = n + 1;
    }

let CountIter = Counter();

CountIter.next();
// Would result in { value: 0, done: false }

// Again
CountIter.next();
```

```
function *fibonacci() {
    let [prev, curr] = [0, 1];
    for (;;) {
        [prev, curr] = [curr, prev + curr];
        yield curr;
    }
}

for (fib of fibonacci()) {
    if (fib === 42)
        break;
    console.log(fib);
}
```

```
function *powPuff() {
  return Math.pow((yield "x"), (yield "y"));
}

let puff = powPuff()

puff.next();

puff.next(2);

puff.next(3); // Guess ;)
```

```
function* menu(){
  while (true){
    var val = yield null;
    console.log('I ate:', val);
  }
}


let meEat = menu();

meEat.next();

meEat.next("Poori");
```

meEat.throw(new Error("E

```
function* menu(){
  while (true){
    try{
      var val = yield null;
      console.log('I ate: ', val);
    }catch(e){
      console.log('Good, now pay the bill :P');
    }
  }
}

meEat.throw(new Error("Burp!"));
```

We can deligate!

```
var inorder = function* inorder(node) {
  if (node) {
    yield* inorder(node.left);
    yield node.label;
    yield* inorder(node.right);
  }
}
```


Deeper you must go! Hmmm

```
function *theAnswer() {
  yield 42;
}

var ans = theAnswer();

ans.next();
```

So....

```
function theAnswer() {
  var state = 0;
  return {
    next: function() {
      switch (state) {
        case 0:
          state = 1;
          return {
            value: 42, // The yielded value.
            done: false
          };
        case 1:
          return {
```

But. this has not uet solved

Let us assume a function na

```
run(genFunc){
    /*
          _ __ ___    __ _  __ _(_) ___
         | '_ ` _ \ / _` |/ _` | |/ __|
         | | | | | | (_| | (_| | | (__
         |_| |_| |_|\__,_|\__, |_|\___|
                             |___/


run(function *(){
  var data = yield read('package.json');
  var result = yield process(data);
  console.log(data);
  console.log(result);
```

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

function run(fn) {
  var gen = fn();

  function next(err, res) {
    var ret = gen.next(res);
    if (ret.done) return;
    ret.value(next);
  }

  next();
}
```

THUNKS!

```
let timeoutThunk = (me) =>
```

```
function readFile(path) {
    return function(callback) {
        fs.readFile(path, callback);
```

Instead of :

```
readFile(path, function(err, result) {
```

We now have:

```
readFile(path)(function(err, result) {
```

So that:

```
var data = yield read('package.json');
```

Baaazinga!

More usecases! Generator-t

```
$ npm install thunkify
```

Turn a regular node function into one which returns a thunk!

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

var read = thunkify(fs.readFile);

read('package.json', 'utf8')(function(err, str){

});
```

```
$ npm install co
```

Write non-blocking code in a nice-ish way!

```
var co = require('co');
var thunkify = require('thunkify');
var request = require('request');

co(function *(){
  try {
    var res = yield get('http://badhost.invalid');
    console.log(res);
  } catch(e) {
    console.log(e.code) // ENOTFOUND
```

```
var urls = [/* Huge list */];

// sequential

co(function *(){
  for (var i = 0; i < urls.length; i++) {
    var url = urls[i];
    var res = yield get(url);
    console.log('%s -> %s', url, res[0].statusCode);
  }
})()

// parallel
```

Intresting? What more?

```
$ npm install co-sleep

var sleep = require('co-sleep');
var co = require('co');

co(function *() {
  var now = Date.now();
  // wait for 1000 ms
  yield sleep(1000);
  expect(Date.now() - now).to.not.be.below(1000);
})();
```

```
$ npm install co-ssh

var ssh = require('co-ssh');

var c = ssh({
  host: 'n.n.n.n',
  user: 'myuser',
  key: read(process.env.HOME + '/.ssh/some.pem')
});

yield c.connect();
yield c.exec('foo');
yield c.exec('bar');
```

```
var monk = require('monk');
var wrap = require('co-monk'); // co-monk!
var db = monk('localhost/test');


yield users.remove({});

// Par||el!
yield [
  users.insert({ name: 'Tom',   species: 'Cat' }),
  users.insert({ name: 'Jerry', species: 'Rat' }),
  users.insert({ name: 'Goffy', species: 'Dog' })
```

```
$ npm install suspend

var suspend = require('suspend'),
    resume = suspend.resume;

suspend(function*() {
    var data = yield fs.readFile(__filename, 'utf8', r
    console.log(data);

var readFile = require('thunkify')(require('fs').readF

suspend(function*() {
    var package = JSON.parse(yield readFile('package.j
    console.log(package.name);
```

Is that all?

Koa FTW?!

```
$ npm install koa

var koa = require('koa');
var app = koa();

// logger

app.use(function *(next){
  var start = new Date;
  yield next;
  var ms = new Date - start;
  console.log('%s %s - %s', this.method, this.url, ms)
});
```

koa

```
// x-response-time

app.use(function(next){
  return function *responseTime(){
    var start = new Date;
    yield next;
    var ms = new Date - start;
    this.set('X-Response-Time', ms + 'ms');
  }
});

// logger

app.use(function(next){
  return function *logger(){
    var start = new Date;
    yield next;
    var ms = new Date - start;
    console.log('%s %s - %s', this.method, this.url, ms);
  }
});

// content-length

app.use(function(next){
  return function *contentLength(){
    yield next;
    if (!this.body) return;
    this.set('Content-Length', Buffer.byteLength(this.body));
  }
});

// response

app.use(function(next){
  return function *response(){
    yield next;
    if ('/' != this.url) return;
    this.body = 'Hello World';
  }
});
```

Something for the client?

Task.js

generators + promises = tasks

```
<script type="application/javascript" src="task.js"></

<!-- 'yield' and 'let' keywords require version opt-in
<script type="application/javascript;version=1.8">
function hello() {
    let { spawn, sleep } = task;
    spawn(function() { // Firefox does not yet use the
        alert("Hello...");
```

Sweet and simple!

```
spawn(function*() {
    try {
        var [foo, bar] = yield join(read("foo.json"),
                                    read("bar.json")).
        render(foo);
        render(bar);
    } catch (e) {
        console.log("read failed: " + e);
```

```
var foo, bar;
var tid = setTimeout(function() { failure(new Error("t

var xhr1 = makeXHR("foo.json",
                   function(txt) { foo = txt; success(
                   function(err) { failure() });
var xhr2 = makeXHR("bar.json",
                   function(txt) { bar = txt; success(
                   function(e) { failure(e) });

function success() {
    if (typeof foo === "string" && typeof bar === "str
        cancelTimeout(tid);
```

ES7 async/await! (("Propo

The idea is to have more sugar!

```
async function <name>?<argumentlist><body>

=>
```

Example of animating elements with Promise:

```
function chainAnimationsPromise(elem, animations) {
    var ret = null;
    var p = currentPromise;
    animations.forEach(function(anim){
      p = p.then(function(val) {
            ret = val;
            return anim(elem);
        });
```

Same example with task.js:

```
function chainAnimationsGenerator(elem, animations) {
    return spawn(function*() {
        var ret = null;
        try {
            for(var anim of animations) {
                ret = yield anim(elem);
            }
        } catch(e) { /* ignore and keep going */ }
        return ret;
    });
```

Same example with async/await:

```
async function chainAnimationsAsync(elem, animations)
    var ret = null;
    try {
        for(var anim of animations) {
            ret = await anim(elem);
        }
    } catch(e) { /* ignore and keep going */ }
    return ret;
}
```

Another example from the draft:

```
async function getData() {
  var items = await fetchAsync('http://example.com/use
  return await* items.map(async(item) => {
    return {
      title: item.title,
      img: (await fetchAsync(item.userDataUrl)).img
    }
  }
```

Performance review!

```
function * Counter() {
    var n = 0;
    while (1 < 2) {
        yield n;
        ++n
    }
}

function funcStat(fn) {
    var optimizations = {
      1 : 'is optimized',
      2 : 'is not optimized',
      3 : 'is always optimized'
```

Function is not (yet) optimized! ~_~

```
[deoptimize global object @ 0x4af7380f3f1]
[disabled optimization for 0x33530e0470d9 <SharedFunct
[marking 0x4af73814041 <JS Function IN (SharedFunction
[disabled optimization for 0x4af73812439 <SharedFuncti

Function: is not optimized
```

```
var tools = window.tools = {};

// thunk
tools.setTimeoutThunk = (ms) => (cb) => setTimeout(cb,

tools.delegated = function*() {
  yield tools.setTimeoutThunk(1);
  yield tools.setTimeoutThunk(1);
};

tools.delegator = function*() {
  yield* tools.delegated();
}.
```

Testing in Firefox 33.0 on OS X 10.9

Test		Ops/sec

Without delegation	`// async test`
	`tools.runParallel(function(cb) {`
	`  var gen = tools.delegated(),`
	`  done = false;`
	
	`  while (!done) {`
	`  done = gen.next().done;`
	`  }`
	
	`  cb();`
	`}, function() {`
	`  deferred.resolve();`
	`});`
		6.93
		±9.92%
		fastest

With delegation	`// async test`
	`tools.runParallel(function(cb) {`
	`  var gen = tools.delegator(),`
	`  done = false;`
	
	`  while (!done) {`
	`  done = gen.next().done;`
	`  }`
	
	`  cb();`
	`}, function() {`
	`  deferred.resolve();`
	`});`
		4.12
		±9.57%
		40% slower

```
Benchmark.prototype.setup = function() {
    var arr = Array(100).join(Math.random()).split('.'
    function* generator() {
      for(var
        i = 0; i < this.length;
        yield this[i++]
      );
    }
    var genvar0 = 0;
    function genNext() {
        var i = genvar0;
        genvar0 = i + 1;
        if (i <= this.length) {
```

Testing in Firefox 33.0 on OS X 10.9		
Test		**Ops/sec**
generator	`var values = (i for (i in generator.call(arr)));` `console.log(values.length);`	49,282 ±9.89% 100% slower

simulated generator	`var values = [];`
	`while (true) {`
	`  var value = genNext.call(arr);`
	`  if (value) {`
	`  values.push(value);`
	`  } else {`
	`  break;`
	`  }`
	`}`
		16,759,226
		±2.44%
		fastest

Moral of the story?

http://h3manth.com

http://nmotw.in

http://github.com/hemanth

@gnumanth

Demos -> h3manth.com/es6-generators

THANK YOU! :-)