

YOU DON'T KNOW NODE QUICK INTRO TO 6 CORE FEATURES

BETTER APPS -
BETTER LIFE

ABOUT PRESENTER

AZAT MARDAN



TWITTER: @AZAT_CO
EMAIL: HI@AZAT.CO
BLOG: WEBAPPLOG.COM

ABOUT PRESENTER

- > WORK: TECHNOLOGY FELLOW AT CAPITAL ONE
- > EXPERIENCE: FDIC, NIH, DOCU SIGN, HACKREACTOR AND STORIFY
- > BOOKS: REACT QUICKLY, PRACTICAL NODE.JS, PRO EXPRESS.JS, EXPRESS.JS API AND 8 OTHERS
 - > TEACH: NODEPROGRAM.COM



STARTING WITH BASICS: WHY USE NODE?

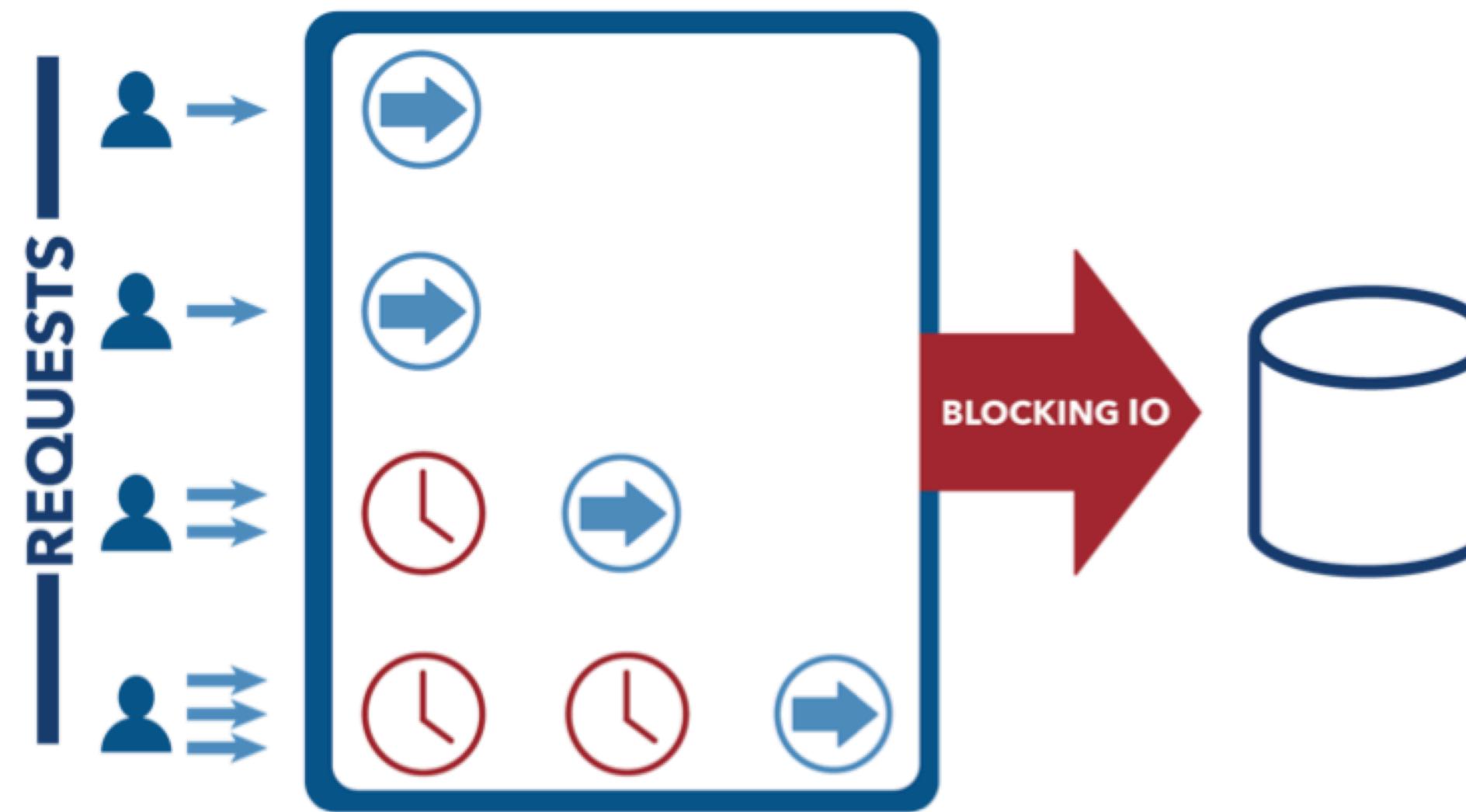
WAY TO RUN
INPUT/OUTPUT
WHICH ARE THE
MOST

JAVA SLEEP

```
System.out.println("Step: 1");
System.out.println("Step: 2");
Thread.sleep(1000);
System.out.println("Step: 3");
```



TECH | BLOCKING I/O



NODE 'SLEEP'

```
console.log('Step: 1')
setTimeout(function () {
  console.log('Step: 3')
}, 1000)
console.log('Step: 2')
```

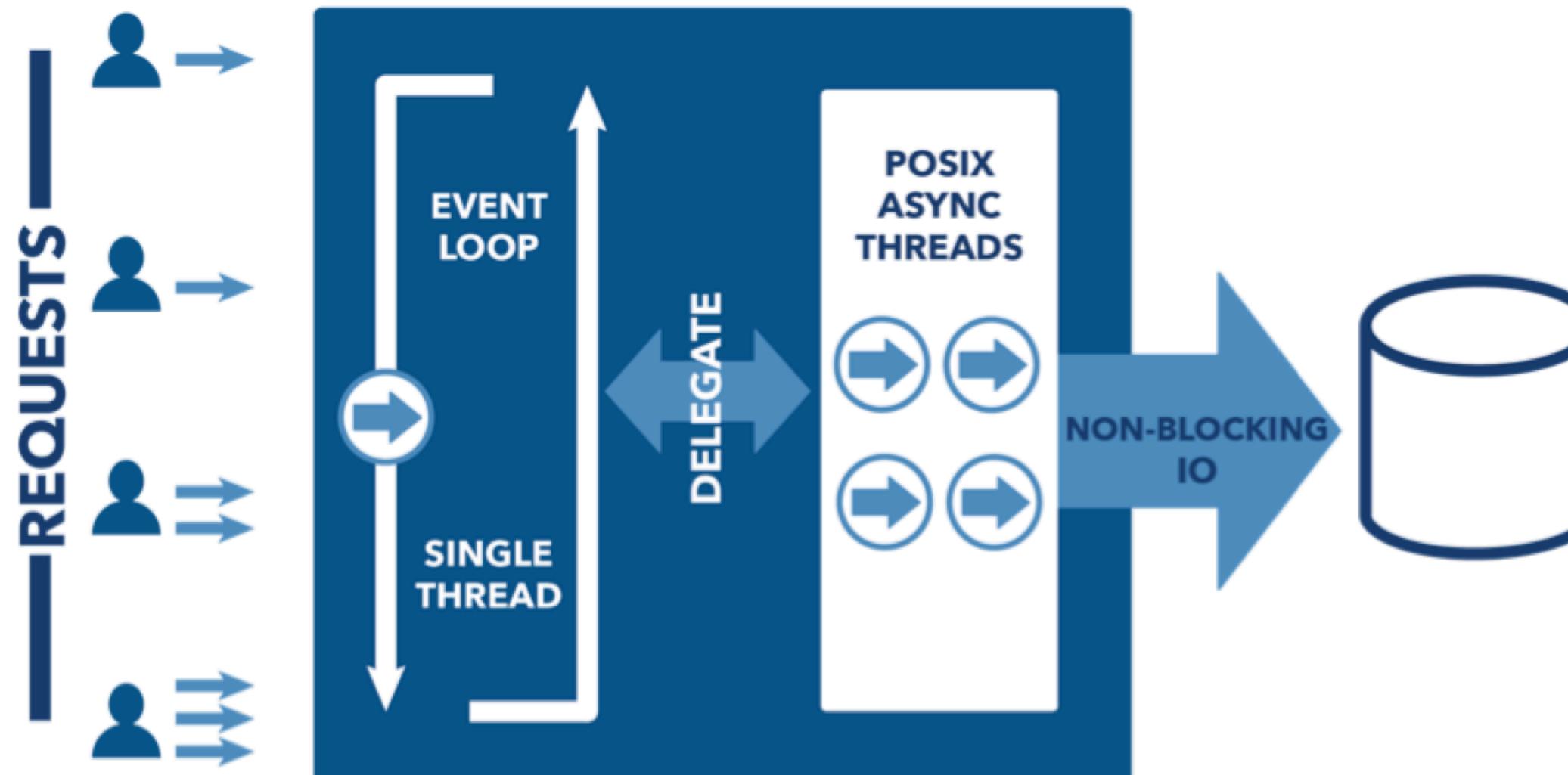
PROCESS MULTIPLE TASKS

```
console.log('Step: 1')
setTimeout(function () {
  console.log('Step: 3')
  // console.log('Step 5')
}, 1000);
console.log('Step: 2')
// console.log('Step 4')
```

EVENT LOOP



NON-BLOCKING I/O



[MULTI-THREADING] IS THE SOFTWARE EQUIVALENT OF A NUCLEAR DEVICE BECAUSE IF IT IS USED INCORRECTLY, IT CAN BLOW UP IN YOUR FACE.

[HTTP://BLOG.CODINGHORROR.COM/THREADING-CONCURRENCY-
AND-THE-MOST-POWERFUL-PSYCHOKINETIC-EXPLOSIVE-IN-THE-
UNIV](http://blog.codinghorror.com/threading-concurrency-and-the-most-powerful-psychokinetic-explosive-in-the-univ)

SINGLE THREAD
- NO WORRIES





TECH | NON-BLOCKING I/O



IT'S STILL POSSIBLE TO
WRITE BLOCKING CODE IN
NODE.JS.



BLOCKING NODE.JS CODE

```
// blocking.js
console.log('Step: 1')
for (var i = 1; i<1000000000; i++) {
    // This will take 100-1000ms
}
console.log('Step: 2')
```

BLOCKING NODE.JS CODE

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

```
var contents = fs.readFileSync('accounts.txt', 'utf8')
console.log(contents)
console.log('Hello Ruby\n')
```

```
var contents = fs.readFileSync('ips.txt', 'utf8')
console.log(contents)
console.log('Hello Node!')
//data1->Hello Ruby->data2->Hello NODE!
```

NON-BLOCKING NODE.JS CODE

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

var contents = fs.readFile('accounts.txt','utf8', function(err,contents){
    console.log(contents)
})
console.log('Hello Python\n')

var contents = fs.readFile('ips.txt','utf8', function(err,contents){
    console.log(contents)
})
console.log("Hello Node!")
//Hello Python->Hello Node->data1->data2
```

MOST OF NODE IS JAVASCRIPT

- ARRAY
- STRING
- PRIMITIVES
- FUNCTIONS
- OBJECTS

NODE!
BROWSER
JAVASCRIPT

HOW TO CREATE GLOBAL VARIABLES (NO window IN NODE)?

GLOBAL

global . fi
lename

global.di
rname

global.modu
le

global require()

global.proc
ess

- > HOW TO ACCESS CLI INPUT, OS, PLATFORM, MEMORY USAGE, VERSIONS, ETC.?
- > WHERE TO STORE PASSWORDS?

PROCESS

process.pid

process. ver
sions

process.arc
h

process.arg
v

process.env

process.upt
ime()

```
process.mem  
oryUsage()
```

process.cwd
()

process.exe
t()

process.on(
))

WHO LIKES AND
UNDERSTANDS
CALLBACKS?



HTTP://CALLBACKHELL.COM

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

CALLBACKS ARE NOT
VERY DEVELOPMENTAL
SCALABLE



ME WHEN WORKING WITH DEEPLY NESTED CALLBACKS



EVENTS

EVENTS

IN NODE.JS AN EVENT CAN BE DESCRIBED SIMPLY AS A STRING WITH A CORRESPONDING CALLBACK.

```
emitter.on('done', function(results) {  
    console.log('Done: ', results)  
})
```

EVENT HANDLING IN NODE USES THE OBSERVER PATTERN

AN EVENT, OR SUBJECT, KEEPS TRACK
OF ALL FUNCTIONS THAT ARE
ASSOCIATED WITH IT

THESE ASSOCIATED FUNCTIONS, KNOWN AS OBSERVERS, ARE EXECUTED WHEN THE GIVEN EVENT IS TRIGGERED

USING EVENT EMITTERS

```
var events = require('events')
var emitter = new events.EventEmitter()

emitter.on('knock', function() {
  console.log('Who\'s there?')
})

emitter.on('knock', function() {
  console.log('Go away!')
})

emitter.emit('knock')
```

INHERITING FROM EVENTEMITTER

```
// job.js
var util = require('util')
var Job = function Job() {
  // ...
  this.process = function() {
    // ...
    job.emit('done', { completedOn: new Date() })
  }
}

util.inherits(Job, require('events').EventEmitter)
module.exports = Job
```

INHERITING FROM EVENTEMITTER

```
// weekly.js
var Job = require('./job.js')
var job = new Job()

job.on('done', function(details){
  console.log('Job was completed at', details.completedOn)
  job.removeAllListeners()
})

job.process()
```

LISTENERS

```
emitter.listeners(eventName)  
emitter.on(eventName, listener)  
emitter.once(eventName, listener)  
emitter.removeListener(eventName, listener)
```

PROBLEMS WITH LARGE DATA

- SPEED: TOO SLOW
- BUFFER LIMIT: ~1GB

STREAMS ABSTRACTIONS FOR CONTINUOUS CHUNKING OF DATA

NOT UNLUCKY
WAIT FOR THE
ENTIRE
RESOURCE TO

TYPES OF STREAMS

- > READABLE
- > WRITABLE
- > DUPLEX
- > TRANSFORM

STREAMS INHERIT FROM EVENT EMITTER

STREAMS ARE EVERYWHERE!

- > HTTP REQUESTS AND RESPONSES
- > STANDARD INPUT/OUTPUT (STDIN&STDOUT)
- > FILE READS AND WRITES

READABLE STREAM EXAMPLE

`process.stdin`

STANDARD INPUT STREAMS CONTAIN DATA GOING INTO APPLICATIONS.

THIS IS
ACHIEVED VIA A
READ
OPERATION.

TYPICALLY
COMES FROM
THE KEYBOARD
USED TO START

TO LISTEN IN ON DATA FROM STDIN, USE THE `data` AND `end` EVENTS:

```
// stdin.js
process.stdin.resume()
process.stdin.setEncoding('utf8')

process.stdin.on('data', function (chunk) {
  console.log('chunk: ', chunk)
})

process.stdin.on('end', function () {
  console.log('--- END ---')
})
```

DEMO

\$ node stdin.js

NEW INTERFACE read()

```
var readable = getReadableStreamSomehow()
readable.on('readable', () => {
  var chunk
  while (null !== (chunk = readable.read())) {
    console.log('got %d bytes of data', chunk.length)
  }
})
```

WRITABLE STREAM EXAMPLE

process.stdout

STANDARD OUTPUT STREAMS CONTAIN DATA GOING OUT OF THE APPLICATIONS.

THIS IS DONE
VIA A WRITE
OPERATION.

DATA WRITTEN
TO STANDARD
OUTPUT IS
VISIBLE ON THE

WRITABLE STREAM

TO WRITE TO `stdout`, USE THE `write` FUNCTION:

```
process.stdout.write('A simple message\n')
```

WHAT ABOUT HTTP?

```
const http = require('http')
var server = http.createServer( (req, res) => {
  var body = ''
  req.setEncoding('utf8')
  req.on('data', (chunk) => {
    body += chunk
  })
  req.on('end', () => {
    var data = JSON.parse(body)
    res.write(typeof data)
    res.end()
  })
})
server.listen(1337)
```

PIPE

```
var r = fs.createReadStream('file.txt')
var z = zlib.createGzip()
var w = fs.createWriteStream('file.txt.gz')
r.pipe(z).pipe(w)
```

WHAT DATA TYPE TO USE FOR BINARY DATA?

BUFFERS

BINARY DATA TYPE. TO CREATE:

- new Buffer(size)
- new Buffer(array)
- new Buffer(buffer)
- new Buffer(str[, encoding])

DOCS: [HTTP://BIT.LY/1IEACZ1](http://bit.ly/1IEACZ1)

WORKING WITH BUFFER

```
// buf.js
var buf = new Buffer(26)
for (var i = 0 ; i < 26 ; i++) {
  buf[i] = i + 97 // 97 is ASCII a
}
console.log(buf) // <Buffer 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76 77 78 79 7a>
console.log(buf.toString('utf8')) // abcdefghijklmnopqrstuvwxyz
```

BUFFER CONVENTION

```
buf.toString('ascii') // outputs: abcdefghijklmnopqrstuvwxyz  
buf.toString('ascii', 0, 5) // outputs: abcde  
buf.toString('utf8', 0, 5) // outputs: abcde  
buf.toString(undefined, 0, 5) // encoding defaults to 'utf8', outputs abcde
```

REMEMBER FS?

```
fs.readFile('/etc/passwd', function (err, data) {  
  if (err) return console.error(err)  
  console.log(data)  
});
```

data IS BUFFER!

DEMO

```
$ node server-stream
```

STREAMS AND BUFFER DEMO

```
// server-stream.js
app.get('/stream', function(req, res) {
  var stream = fs.createReadStream(largeImagePath)
  stream.pipe(res)
})

$ node server-stream
```

[HTTP://LOCALHOST:3000/STREAM](http://localhost:3000/stream)
[HTTP://LOCALHOST:3000/NON-STREAM](http://localhost:3000/non-stream)

RESULTS IN DEVTOOLS

/stream **RESPONDS FASTER!**

X-Response-Time
~300ms vs. 3-5s

STREAM RESOURCES

[HTTPS://GITHUB.COM/SUBSTACK/STREAM-ADVENTURE](https://github.com/Substack/stream-adventure)

```
$ sudo npm install -g stream-adventure  
$ stream-adventure
```

[HTTPS://GITHUB.COM/SUBSTACK/STREAM-HANDBOOK](https://github.com/Substack/stream-handbook)

HOW TO SCALE A SINGLE THREADED SYSTEM?

CLUSTER USAGE

- > MASTER: STARTS WORKERS
- > WORKER: DO THE JOB. E.G., HTTP SERVER

NUMBER OF PROCESSES = NUMBER OF CPUS

CLUSTERS

```
var cluster = require('cluster')
if (cluster.isMaster) {
  for (var i = 0; i < numCPUs; i++) {
    cluster.fork()
  }
} else if (cluster.isWorker) {
  // your server code
})
```

CLUSTER DEMO

- 1. RUN code/cluster.js WITH NODE (\$ node cluster.js).**
- 2. INSTALL loadtest WITH NPM: \$ npm install -g loadtest**
- 3. RUN LOAD TESTING WITH: \$ loadtest http://localhost:3000 -t 20 -c 10**

PRESS CONTROL+C ON THE SERVER TERMINAL

CLUSTER LIBRARIES

- > CORE CLUSTER: LEAN AND MEAN
- > STRONG-CLUSTER-CONTROL ([HTTPS://GITHUB.COM/STRONGLOOP/STRONG-CLUSTER-CONTROL](https://github.com/strongloop/strong-cluster-control)). OR `\$ SLC RUN` : GOOD CHOICE
- > PM2 ([HTTPS://GITHUB.COM/UNITECH/PM2](https://github.com/unitech/pm2)): GOOD CHOICE

PM2

[HTTPS://GITHUB.COM/UNITECH/PM2](https://github.com/unitech/pm2)

[HTTP://PM2.KEYMETRICS.IO](http://pm2.keymetrics.io)

ADVANTAGES:

- › LOAD-BALANCER AND OTHER FEATURES
- › OS RELOAD DOWN-TIME, I.E., FOREVER ALIVE

PM2 DEMO: TYPICAL EXPRESS SERVER

```
var express = require('express')
var port = 3000
global.stats = {}
console.log('worker (%s) is now listening to http://localhost:%s',
  process.pid, port)
var app = express()
app.get('*', function(req, res) {
  if (!global.stats[process.pid]) global.stats[process.pid] = 1
  else global.stats[process.pid] += 1;
  var l ='cluser '
    + process.pid
    + ' responded \n';
  console.log(l, global.stats)
  res.status(200).send(l)
})
app.listen(port)
```

PM2 DEMO

USING server.js:

```
$ pm2 start server.js -i 0
```

IN A NEW WINDOW:

```
$ loadtest http://localhost:3000 -t 20 -c 10  
$ pm2 list
```

SPAWN VS FORK VS EXEC

- › `require('child_process').spawn()` - **LARGE DATA.
STREAM, NO NEW V8 INSTANCE**
- › `require('child_process').fork()` - **NEW V8
INSTANCE, MULTIPLE WORKERS**
- › `require('child_process').exec()` - **BUFFER.
ASYNC. ALL THE DATA AT ONCE**

SPAWN EXAMPLE

```
fs = require('fs')
process = require('child_process')
var p = process.spawn('node', 'program.js')
p.stdout.on('data', function(data)) {
  console.log('stdout: ' + data)
})
```

FORK EXAMPLE

```
fs = require('fs')
process = require('child_process')
var p = process.fork('program.js')
p.stdout.on('data', function(data)) {
  console.log('stdout: ' + data)
})
```

EXEC EXAMPLE

```
fs = require('fs')
process = require('child_process')
var p = process.exec('node program.js', function (error, stdout, stderr) {
  if (error) console.log(error.code)
})
```

HOW TO HANDLE ASYNC ERRORS?

HANDLING ASYNC ERRORS

EVENT LOOP: ASYNC ERRORS ARE HARDER TO HANDLE/DEBUG.
BECAUSE SYSTEM LOSES CONTEXT OF THE ERROR. THEN.
APPLICATION CRASHES.

TRY/CATCH IS NOT GOOD ENOUGH.

SYNCHRONOUS ERROR IN NODE

```
try {  
    throw new Error('Fail!')  
} catch (e) {  
    console.log('Custom Error: ' + e.message)  
}
```

FOR SYNC ERRORS TRY/CATCH WORKS FINE.

ASYNC ERROR EXAMPLE

```
try {  
    setTimeout(function () {  
        throw new Error('Fail!')  
    }, Math.round(Math.random()*100))  
} catch (e) {  
    console.log('Custom Error: ' + e.message)  
}
```

THE APP CRASHES!

ME WHEN ASYNC ERROR'S THROWN



ASYNC ERRORS

HOW TO DEAL WITH IT?



BEST PRACTICES FOR ASYNC ERRORS?

- > LISTEN TO ALL 'ON ERROR' EVENTS
- > LISTEN TO uncaughtException
- > USE domain (SOFT DEPRECATED) OR ASYNCWRAP
 - > LOG, LOG, LOG & TRACE
 - > NOTIFY (OPTIONAL)
- > EXIT & RESTART THE PROCESS

ON('ERROR')

ANYTHING THAT INHERITS FROM OR CREATES AN INSTANCE OF THE
ABOVE: EXPRESS, LOOPBACK, SAILS, HAPI, ETC.

```
server.on('error', function (err) {  
  console.error(err)  
})
```

ON('ERROR') CHAINED METHOD EXAMPLE

```
var http = require('http')
var server = http.createServer(app)
.on('error', function(e) {
  console.log('Failed to create server')
  console.error(e)
  process.exit(1)
})
```

ON('ERROR') NAMED VARIABLE EXAMPLE

```
var req = http.request(options, function(res) {  
  // ... processing the response  
})  
  
req.on('error', function(e) {  
  console.log('problem with request: ' + e.message)  
})
```

UNCAUGHTEXCEPTION

uncaughtException IS A VERY CRUDE MECHANISM FOR EXCEPTION HANDLING. AN UNHANDLED EXCEPTION MEANS YOUR APPLICATION – AND BY EXTENSION NODE.JS ITSELF – IS IN AN UNDEFINED STATE. BLINDLY RESUMING MEANS ANYTHING COULD HAPPEN.

UNCAUGHTEXCEPTION

ALWAYS LISTEN TO uncaughtException!

```
process.on('uncaughtException', handle)
```

OR

```
process.addListener('uncaughtException', handle)
```

UNCAUGHTEXCEPTION EXPANDED EXAMPLES

```
process.on('uncaughtException', function (err) {  
  console.error('uncaughtException: ', err.message)  
  console.error(err.stack)  
  process.exit(1)  
})
```

OR

```
process.addListener('uncaughtException', function (err) {  
  console.error('uncaughtException: ', err.message)  
  console.error(err.stack)  
  process.exit(1)
```

DOMAIN

THIS MODULE IS SOFTLY DEPRECATED IN 4.0 (MOST LIKEY WILL BE SEPARATE FROM CORE MODULE). BUT THERE'S NO ALTERNATIVES IN CORE AS OF NOW.

DOMAIN EXAMPLE

```
var domain = require('domain').create()
domain.on('error', function(error){
  console.log(error)
})
domain.run(function(){
  throw new Error('Failed!')
})
```

DOMAIN WITH ASYNC ERROR DEMO

DOMAIN-ASYNC.JS:

```
var d = require('domain').create()
d.on('error', function(e) {
  console.log('Custom Error: ' + e)
})
d.run(function() {
  setTimeout(function () {
    throw new Error('Failed!')
  }, Math.round(Math.random()*100))
});
```

C++ ADDONS

HOW TO WRITE C/C++ BINDING FOR YOUR IOT. HARDWARE, DRONE, SMARTDEVICE, ETC.?

NODE AND C++

CREATE THE hello.cc FILE:

```
#include <node.h>

namespace demo {

using v8::FunctionCallbackInfo;
using v8::HandleScope;
using v8::Isolate;
using v8::Local;
using v8::Object;
using v8::String;
using v8::Value;
```

NODE AND C++

CREATE THE hello.cc FILE:

```
void Method(const FunctionCallbackInfo<Value>& args) {
  Isolate* isolate = args.GetIsolate();
  args.GetReturnValue().Set(String::NewFromUtf8(isolate, "capital one"));
}

void init(Local<Object> exports) {
  NODE_SET_METHOD(exports, "hello", Method);
}

NODE_MODULE(addon, init)

} // namespace demo
```

CREATING binding.gyp

CREATE binding.gyp:

```
{  
  "targets": [  
    {  
      "target_name": "addon",  
      "sources": [ "hello.cc" ]  
    }  
  ]  
}
```

NODE-GYP

```
$ npm install -g node-gyp
```

[HTTPS://GITHUB.COM/NODEJS/NODE-GYP](https://github.com/nodejs/node-gyp)

CONFIGURING AND BUILDING

```
$ node-gyp configure  
$ node-gyp build
```

CHECK FOR COMPILED .node FILES IN BUILD/RELEASE/

C++ ADDONS EXAMPLES

[HTTPS://GITHUB.COM/NODEJS/NODE-ADDON-EXAMPLES](https://github.com/nodejs/node-addon-examples)

INCLUDING ADDON

CREATE hello.js AND INCLUDE YOUR C++ ADDON:

```
var addon = require('./build/Release/addon')
console.log(addon.hello()) // 'capital one'
```

RUN

```
$ node hello.js
```

CAPITAL ONE

WE USE NODE A LOT!

[HTTPS://WWW.YOUTUBE.COM/WATCH?V=BJPELJHV1lC](https://www.youtube.com/watch?v=BJPELJHV1lC)



30-SECOND SUMMARY

1. EVENT EMITTERS
2. STREAMS
3. BUFFERS
4. CLUSTERS
5. C++ ADDONS
6. DOMAIN

THE END



Atwood's Law

Any application that *can* be written in JavaScript,
will eventually be written in JavaScript.

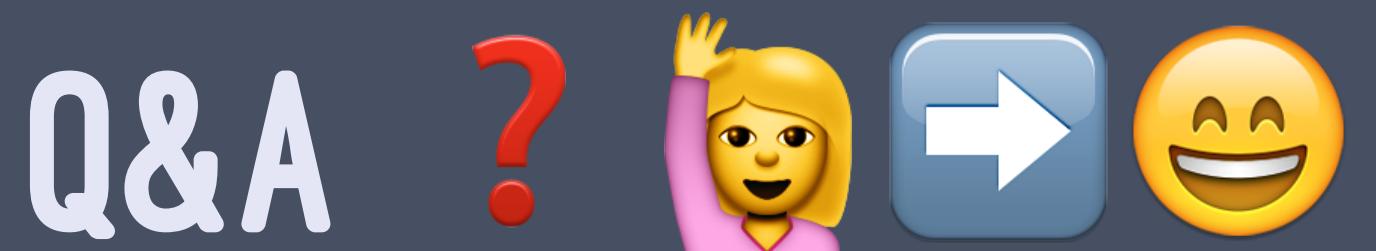
SLIDES & CODE



[HTTPS://GITHUB.COM/AZAT-CO/YOU-DONT-KNOW-NODE](https://github.com/azat-co/you-dont-know-node)

OR

PDF: [HTTP://BIT.LY/1VJWPQK](http://bit.ly/1VJWPQK)



TWITTER: @AZAT_CO
EMAIL: HI@AZAT.CO