AMIS

Fontys

INTRODUCING NODE

node JS™

Lucas Jellema

15th May 2017, Fontys Hogeschool, Eindhoven

AGENDA

INTRODUCTION OF NODE (FKA NODE.JS)

SERVER SIDE JAVASCRIPT – HELLO WORLD!

THE NODE ECOSYSTEM

HTTP REQUEST HANDLING, EXPRESS & OUTBOUND HTTP CALLS

HANDSON WORKSHOP

SERVER PUSH, EMAILING, PROMISES AND INTERACTING WITH MONGODB & ORACLE

MORE HANDSON WORKSHOP

OVERVIEW

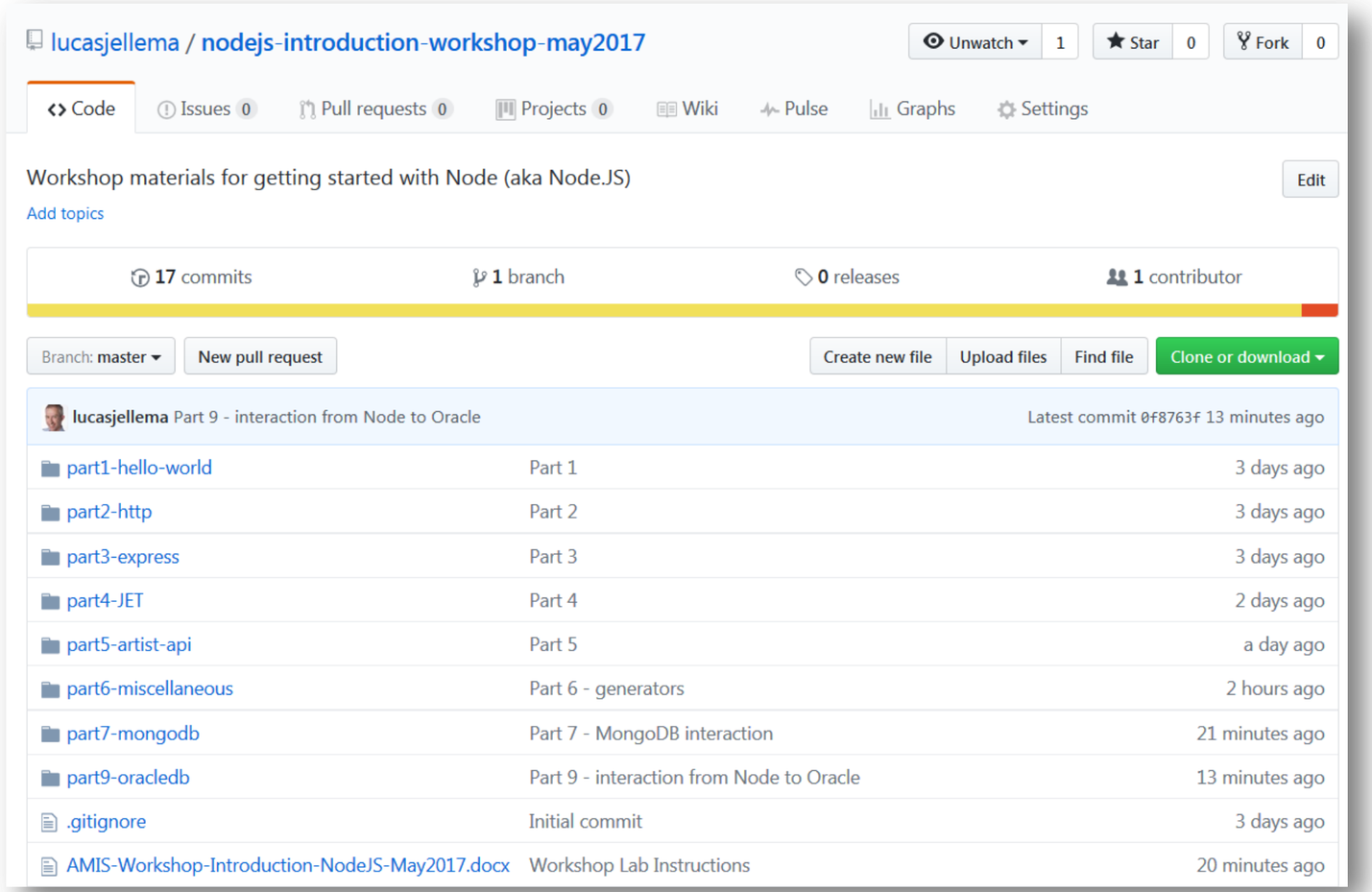
- Get used to it: JavaScript on the Server
- First steps with Node.js and yes: hello world!
- Some language essentials: *functions (*and *closures)*
- Modules and npm

- The Single Thread or: the quest for scalability
- Handling HTTP requests
  - static file serving
  - serving rich client web applications [such as Oracle JET]
  - Implementing REST APIs and mBaaS (and iPaaS)
- Making HTTP calls to external APIs and Services
- Database interaction from Node.js

AMIS

SUPPORTING RESOURCES ON GITHUB

- https://github.com/lucasjellema/nodejs-introduction-workshop-may2017



ARE THESE TERMS FAMILIAR?

Apache Kafka

DevOps

NoSQL [database]

microservices

WebSockets

Angular

git & GitHub

Cloud

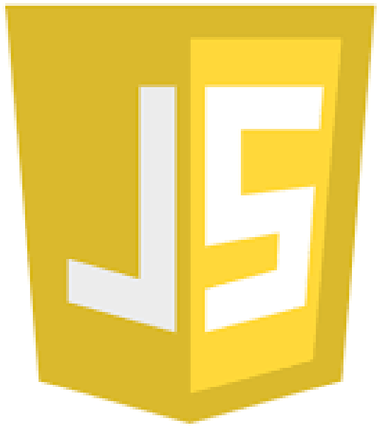
Docker [containers]

JSON

Server less computing
(AWS Lambda/Azure Functions)

ngrok

JAVASCRIPT



AMIS

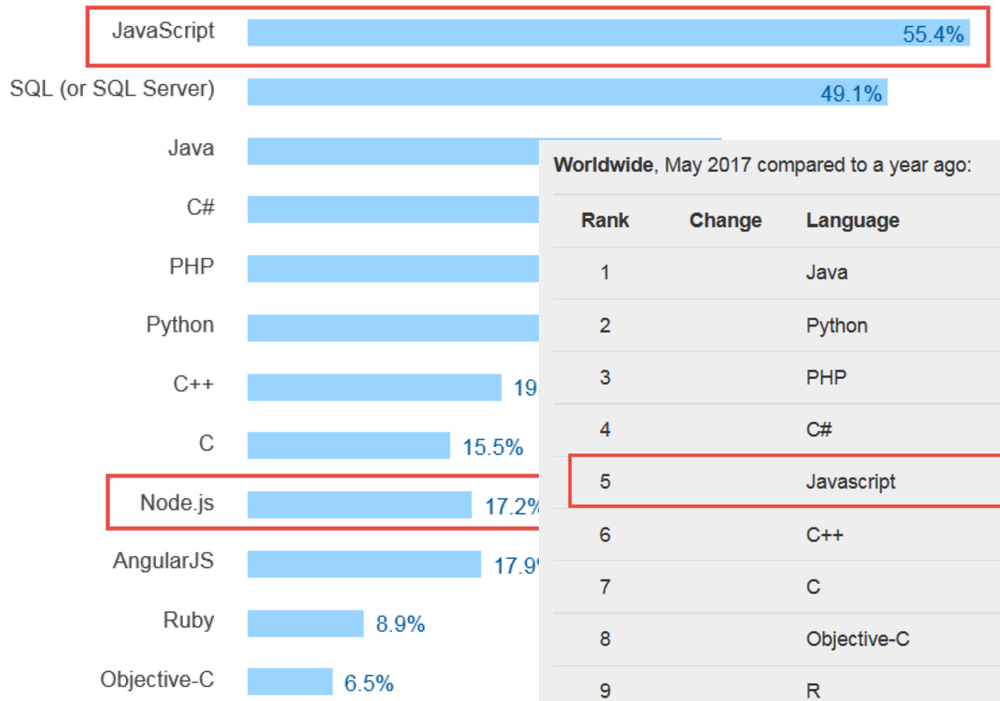
TIOBE – LONG TERM HISTORY

Programming Language	2017	2012	2007	2002	1997	1992	1987
Java	1	1	1	1	14	-	-
C	2	2	2	2	1	1	1
C++	3	3	3	3	2	2	4
C#	4	4	7	14	-	-	-
Python	5	7	6	9	27	-	-
PHP	6	5	4	5	-	-	-
JavaScript	7	9	8	7	20	-	-
Visual Basic .NET	8	21	-	-	-	-	-
Perl	9	8	5	4	4	11	-
Assembly language	10	-	-	-	-	-	-
COBOL	25	31	17	6	3	13	8
Lisp	31	12	14	10	9	9	2
Prolog	33	37	26	13	18	14	3
Pascal	102	13	19	29	8	3	5

POPULARITY RATINGS…

I. Most Popular Technologies

2016 | 2015 | 2014 | 2013

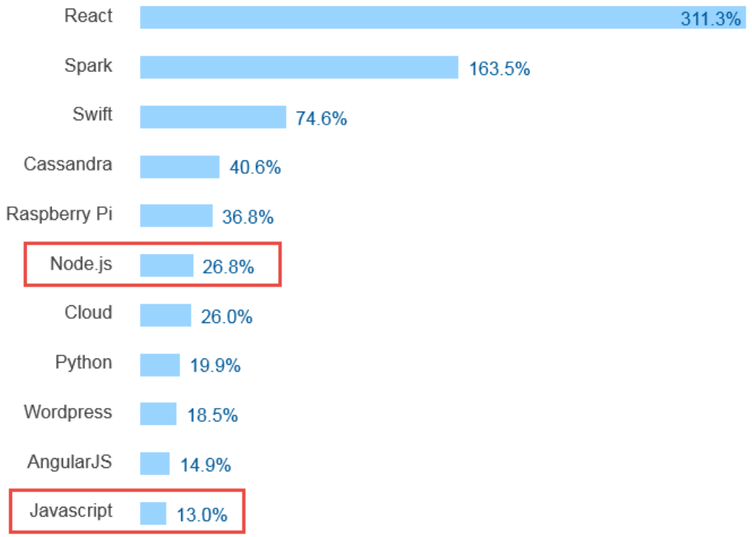


Worldwide, May 2017 compared to a year ago:

Rank	Change	Language	Share	Trend
1		Java	22.7 %	-1.3 %
2		Python	15.7 %	+3.5 %
3		PHP	9.3 %	-1.1 %
4		C#	8.3 %	-0.5 %
5		Javascript	7.9 %	+0.5 %
6		C++	6.9 %	-0.2 %
7		C	6.7 %	-0.1 %
8		Objective-C	3.8 %	-0.9 %
9		R	3.6 %	+0.4 %
10		Swift	2.8 %	-0.1 %
11		Matlab	2.6 %	-0.1 %
12		Ruby	1.9 %	-0.4 %
13		Visual Basic	1.4 %	0.2 %

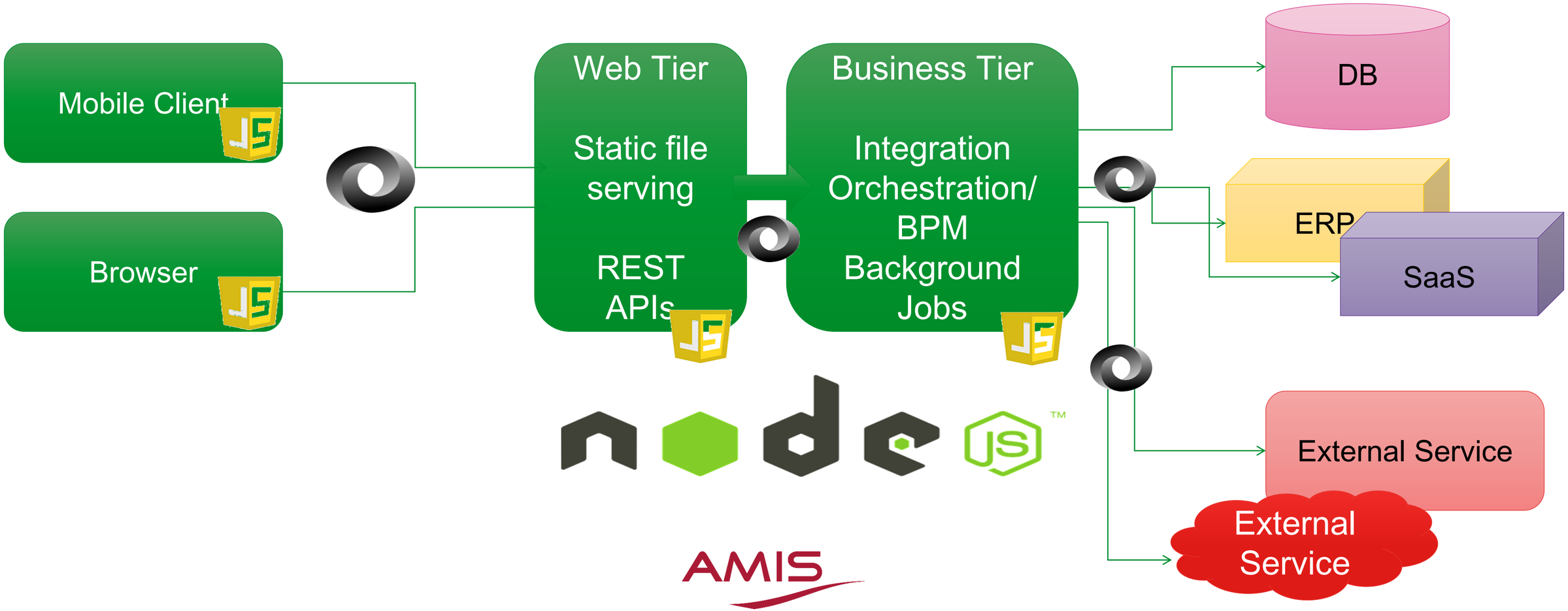
IV. Trending Tech on Stack Overflow

Winners | Losers

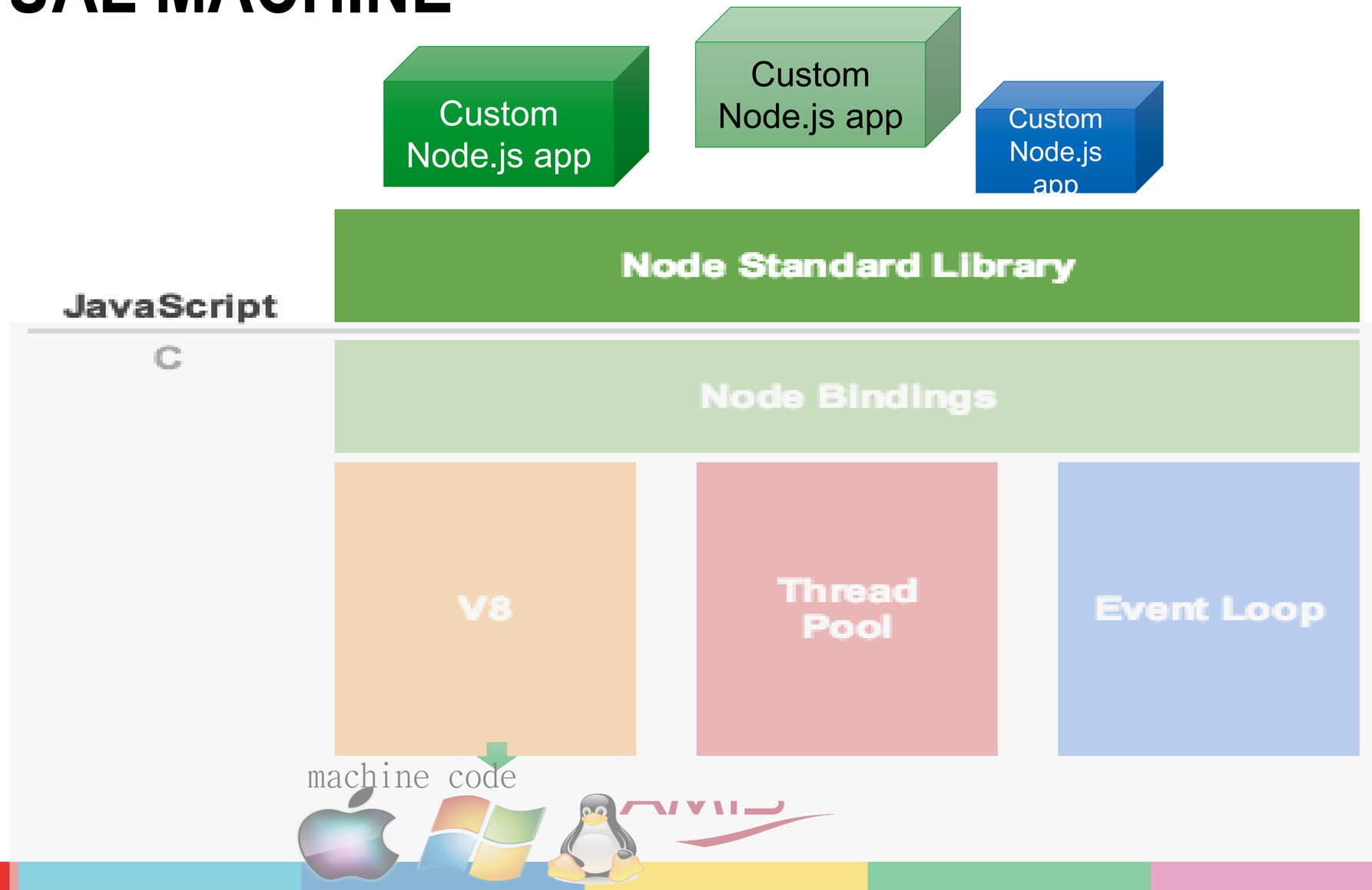


May 2017	May 2016	Change	Programming Language	Ratings	Change
1	1		Java	14.639%	-6.32%
2	2		C	7.002%	-6.22%
3	3		C++	4.751%	-1.95%
4	5	^	Python	3.548%	-0.24%
5	4	v	C#	3.457%	-1.02%
6	10	^^	Visual Basic .NET	3.391%	+1.07%
7	7		JavaScript	3.071%	+0.73%
8	12	^^	Assembly language	2.859%	+0.98%
9	6	v	PHP	2.693%	-0.30%
10	9	v	Perl	2.602%	+0.28%
11	8	v	Ruby	2.429%	+0.09%
12	13	^	Visual Basic	2.347%	+0.52%
13	15	^	Swift	2.274%	+0.68%
14	16	^	R	2.192%	+0.86%

JAVASCRIPT IN ENTERPRISE ARCHITECTURE



NODE.JS IS LIKE A JS-DK – JAVASCRIPT VIRTUAL MACHINE

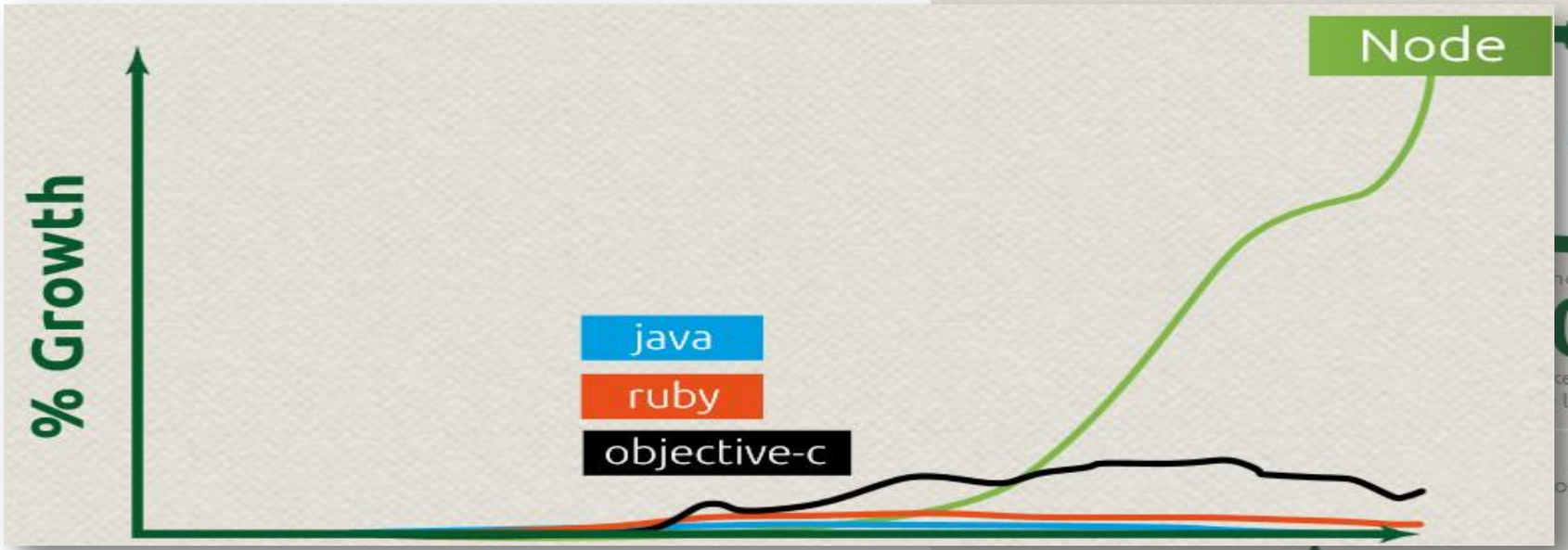


BRIEF HISTORY OF NODE.JS

- 2009 – unveiled by Ryan Dahl, written on Linux, leveraging Google V8
- 2010
  - Initial release of *express* – the default web application server framework
- 2011
  - package manager *npm* introduced – to publish and share open-source Node.js libraries and simplify installation, updating and uninstallation of libraries
  - Native Windows implementation of Node.js released
- 2011-2015 Many Cloud Platforms supporting Node.js
  - Heroku, Joyent, Cloud Foundry, Nodester, NodeJitsu, Microsoft Azure, IBM BlueMix, Google AppEngine, AWS, …

Node.js Goes Mainstream in the Enterprise

THE NUMBER OF NODE PROFESSIONALS IS GROWING FAST!

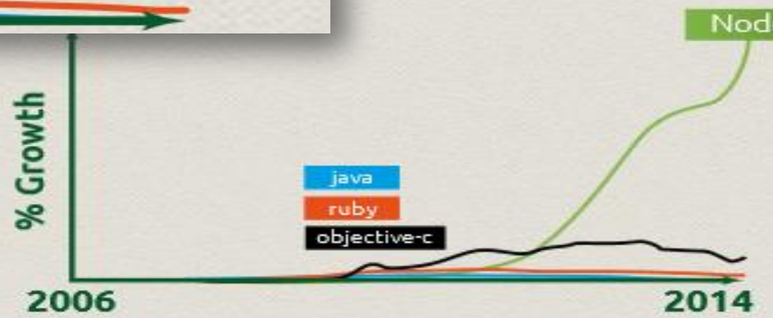


here are over
0,000
edIn members
listed Node as a skill

npm download volume is greater during the workweek

More Node development is happening on company time vs personal time.

obs are growing faster than other languages.



NODE IS IN PRODUCTION AT BIG COMPANIES

Big companies have moved from experimenting with Node to deploying it into production. Companies like...

NBC · Bank of America Merrill Lynch · STAPLES · TARGET · BEST BUY · Cigna. · Bloomberg · Alibaba.com · intuit · FANDANGO · PG&E · WELLS FARGO · Symantec · Go Daddy · ★macy's · DIRECTV

...FOR USE CASES LIKE:

APIs
The "glue" that connects devices and browsers to data and services

Mobile
Backends and full-stack JavaScript hybrid apps

Web
Servers and single-page apps

Internet of Things
Network connected embedded devices and sensors

ENTERPRISE SOFTWARE VENDORS ARE BETTING BIG ON NODE

IBM
Launched NodeRed, an IoT framework plus a JavaScript runtime for IBM platforms.

amazon.com
Announced Lambda, a Node powered event driven computing service for APIs and dynamic applications.

Microsoft
Launched a Node developer center and support for Node in Visual Studio and Azure.

ca technologies
Partnered with StrongLoop to add Node monitoring to their APM.

ORACLE
Launched Nashorn a high-performance JavaScript runtime, enabling Node apps to run on the JVM plus a Node driver for Oracle database.

StrongLoop
Announced Arc, a complete set of tools for developing APIs in Node, from design to production. Maintains the popular Express and LoopBack frameworks.

redhat
Acquired FeedHenry a Node-based mobile backed-as-service.

apigee
Added Node support for API extensibility.

BRIEF HISTORY OF NODE.JS

- 2015
  - Various forks brought back together: Node v4.0 including V8 with ES6 features
  - Open Source under Governance by the Node.js Foundation under Linux Foundation
  - Support for ARM processors
  - Oracle releases node-oracledb Database Driver for Node.js
- 2016/2017
  - **Node** is gradually becoming the new denominator (taking over from Node.JS)
  - Major steps in JavaScript (ES6)
  - More Tool support for Node
  - Node one of primary technologies for Serverless Computing (e.g. AWS Lambda) as well as for microservices
  - Rapid evolution of Node releases
    - Current release: 6.10.3 (LTS) and 7.4.0 (Current)

HELLO WORLD

```
// my first Node.js program

console.log("Hello World!");
```

```
node hello-world.js
Hello World!
```

HELLO WORLD – USING A FUNCTION

```
function greeting() {
  console.log("Hello World!");
}

greeting();
```

```
node hello-world-2.js
Hello World!
```

HELLO WORLD – READ COMMANDLINE ARGUMENTS

```
function greeting(greetedPerson) {
  console.log("Hello "+ greetedPerson + "!");
}

// assume command line: node hello-world-3.js someName
var greetee = process.argv[2];
greeting(greetee);
```

```
node hello-world-3.js Lex
Hello Lex!
```

HELLO WORLD – PASS AROUND A FUNCTION REFERENCE

```
var g = function greeting(greetedPerson) {
          console.log("Hello "+ greetedPerson + "!");
        }

var greetee = process.argv[2];

g(greetee);
```

```
node hello-world-4.js Lex
Hello Lex!
```

HELLO WORLD – HAVE ONE FUNCTION EXECUTE [A REFERENCE TO] ANOTHER

```
var g = function greeting(greetedPerson) {
          console.log("Hello "+ greetedPerson + "!");
        }

var greetee = process.argv[2];

function reception( greetFunction, greetee) {
  greetFunction(greetee); // execute the passed in function
}

reception(g, greetee);
```

```
node hello-world-5.js Lex
Hello Lex!
```

HELLO WORLD – CALLBACK FUNCTIONS AND TIME OUTS – NO STATE YET

```
// callback functions used with timeout
var g = function (greetedPerson) {
          console.log("Hello "+ greetedPerson + "!");
        }

var r = function ( greetFunction, greetee) {
  greetFunction(greetee);
}

for (var i=2;i<process.argv.length;i++) {
  setT
}

console
```

```
node hello-world-7.js Lex Tobias Madelon
Hello Lex!
Hello Tobias!
Hello Madelon!
The Main Program Flow is Done!
```

HELLO WORLD – CLOSURES WITH STATE EXECUTED ON TIME OUT

```
var g = function (greetedPerson) {
          console.log("Hello "+ greetedPerson + "!");
        }

function getGreeter ( greetee, greetFunction) {
  var toGreet = greetee;
  console.log('I will greet '+ greetee + ' in a little while');
  return function () {  greetFunction(toGreet)};  // the closure
}

for (var i=2;i<process.a
  setTimeout( getGreeter
}

console.log('The Main Pr
```

```
node hello-world-8.js Lex Tobias Madelon
I will greet Lex in a little while
I will greet Tobias in a little while
I will greet Madelon in a little while
The Main Program Flow is Done!
Hello Lex!
Hello Tobias!
Hello Madelon!
```

MODULES

- Node programs can be organized in modules
- A module is file that exports a scope that may contain (public) functions and shared objects
- Modules are ‘imported’ through the *require* function
  - Add a line such as *var localRef = require(‘moduleName’);* in a file to get access at runtime to all the goodies from the module
- Node ships with a set of core modules that can be required into any program – without additional installation or configuration; for example:
  - fs: file system operations
  - path: string transformations around file system operations
  - os: access to some operating systems details
  - util: utility functions, for example util.log to log messages to console with timestamp
  - http: for HTTP protocol based network operations

FILE MANIPULATION FROM NODE.JS - USING CORE MODULE FS

```
//write a file with all command line arguments on separate lines

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

process.argv.slice(2).forEach(function (val) {
  fs.appendFileSync("./output.txt", val + "\n");
});

util.log('The Main Program Flow is Done!');
```

HELLO WORLD – USING CORE MODULE UTIL

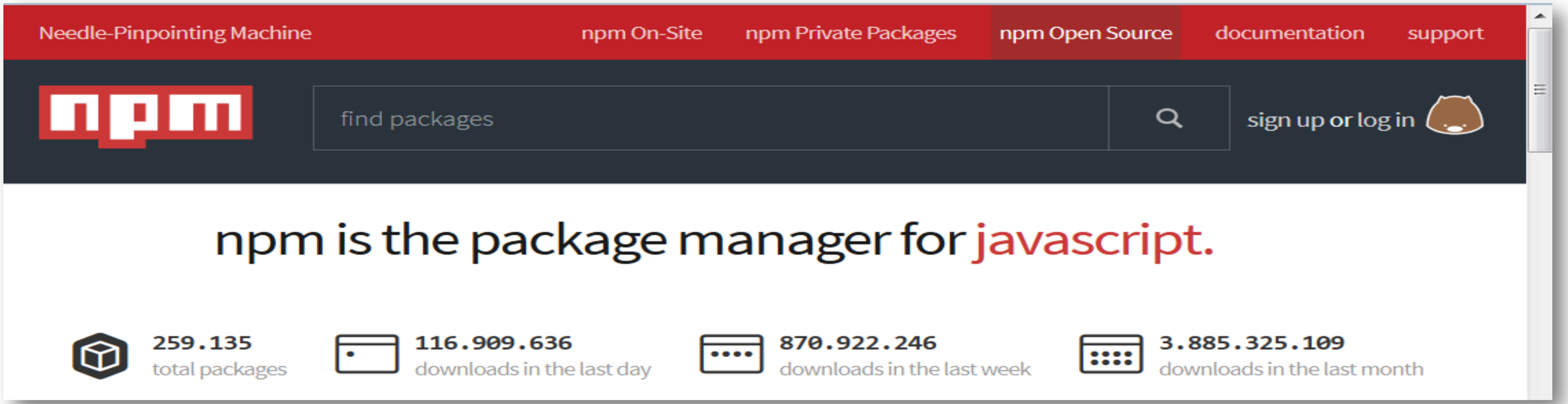
```
var util = require('util');
var g = function (greetedPerson) {
          util.log(util.format('Hello %s!', greetedPerson));
        }

function getGreeter ( greetee, greetFunction) {
  var toGreet = greetee;
  util.log( util.format('I will greet %s in a little while', toGreet));
return function () {  greetFunction(toGreet)};
}

process.argv.slice(2).forEach(function (val, index, array) {
    setTimeout( getGr
});

util.log('The Main Pr
```

```
node hello-world-9.js Lex Tobias Madelon
27 Mar 08:27:47 - I will greet Lex in a little while
27 Mar 08:27:47 - I will greet Tobias in a little while
27 Mar 08:27:47 - I will greet Madelon in a little while
27 Mar 08:27:47 - The Main Program Flow is Done!
27 Mar 08:27:47 - Hello Lex!
27 Mar 08:27:48 - Hello Tobias!
27 Mar 08:27:50 - Hello Madelon!
```

PACKAGES

- A package is a bundle of resources that supports development of a node application
  - Design time (test, code analyze, format, browserify, package, …)
  - Run time
- A package can contain one or more modules that may or may not be intended to be ‘required’ into a *node* application
- Packages are [downloaded and] added to the source folders of a *node* application
- Package management can be tricky
  - An application can depend on multiple packages
  - Packages can depend on multiple [versions of] packages themselves
  - Gathering and keeping up to date all compatible versions of all required packages is potentially complex, time consuming and error prone
  - Enter: *npm*
    - *and package.json – the closest thing to a pom-file*

REUSING NODE MODULES



NPM

- Bring in a package to your current application
  - **npm install <package name>**

    The package is downloaded and installed in folder node_modules
- Create a new application with a neat description
  - **npm init**

    This will create a package.json file that
    describes the [Node.js] application and its dependencies
- Use
  - **npm install <package name> --save**

    To have dependencies on the package added to the package,json file

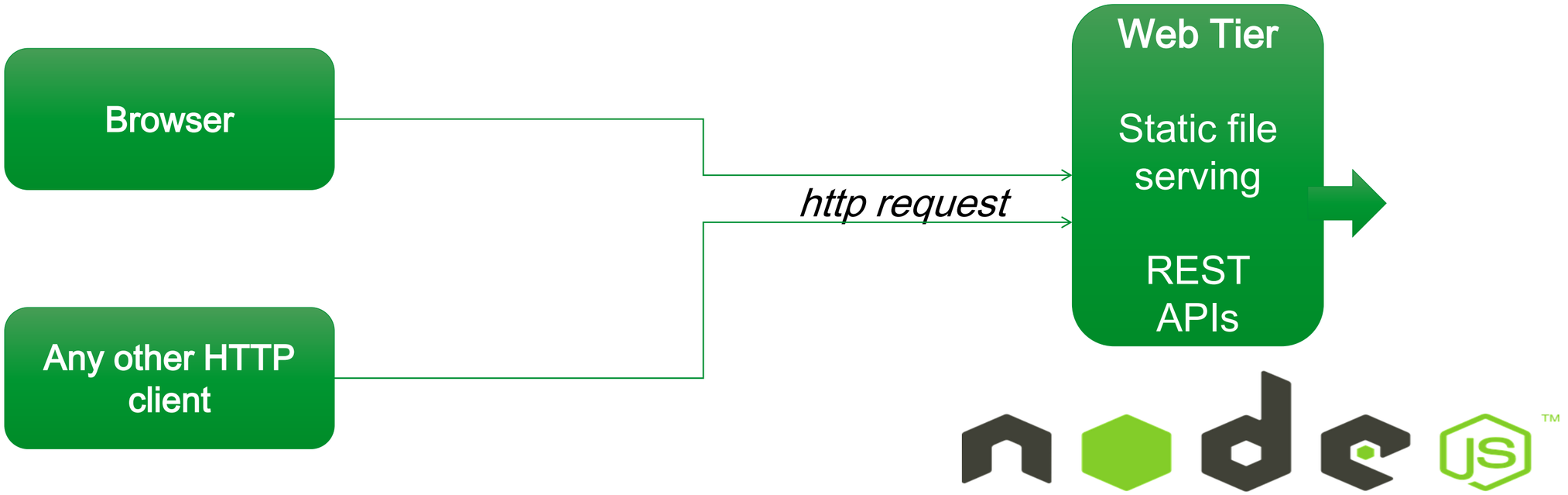
  - **npm install**

    To have all packages listed in package.json installed or updated into the node_modules folder

```
  {
  "name": "part3",
  "version": "1.0.0",
  "description": "Trying out the Express framework",
  "main": "express.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "Lucas Jellema",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.15.0",
    "express": "^4.13.4",
    "serve-static": "^1.10.2"
  }
}
```

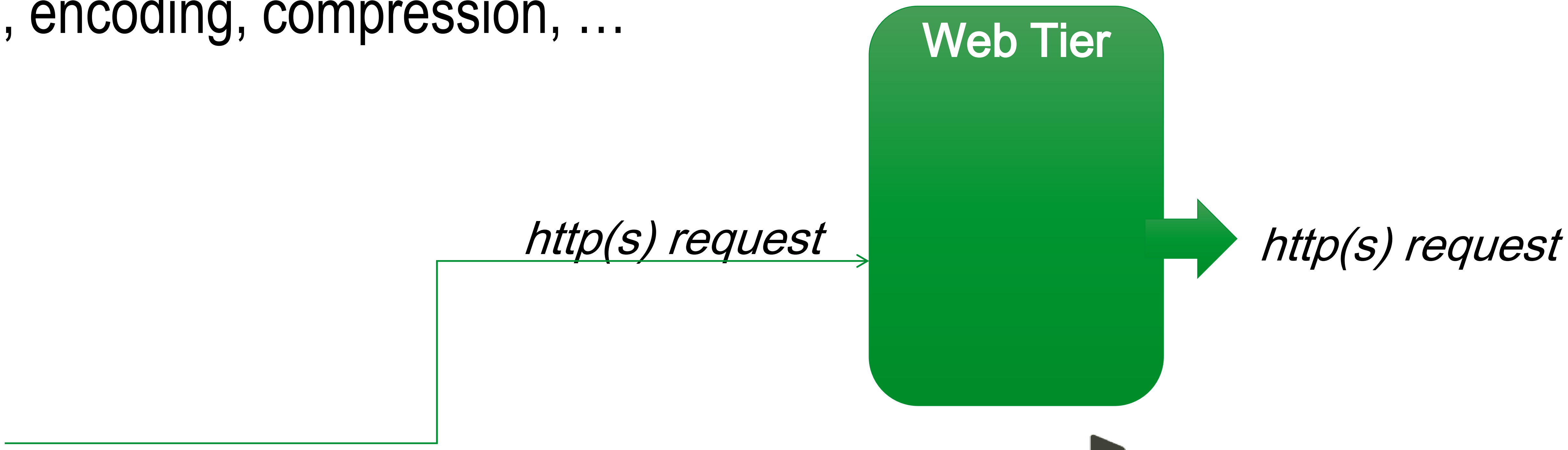
NODE.JS CAN HANDLE INCOMING HTTP REQUESTS

- Similar to a Java Servlet or PHP page, a Node.js application can contain logic to interpret and respond to an HTTP request



CORE MODULES FOR (INBOUND & OUTBOUND) NETWORKING: HTTP, HTTPS, NET, DGRAM

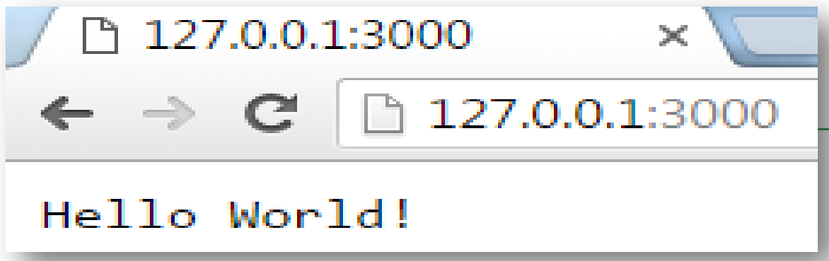
- No separate installation required (included with core Node.js)
- Handle incoming request and/or make outbound request
- Handle URL (path, query parameters, post payload or Form encoding or file upload)
- Handle headers, encoding, compression, …



SIMPLEST FORM OF HTTP HANDLING

```
// invoke from browser or using curl:  curl http://127.0.0.1:3000
var http = require('http');

var server = http.createServer(function handleRequest(req, res) {
    res.write('Hello World!');
    res.end();
}).listen(3000);

console.log('server running on port 3000');
```

```
C:\Users\lucas_j>curl http://127.0.0.1:3000
Hello World!
```



http(s) request

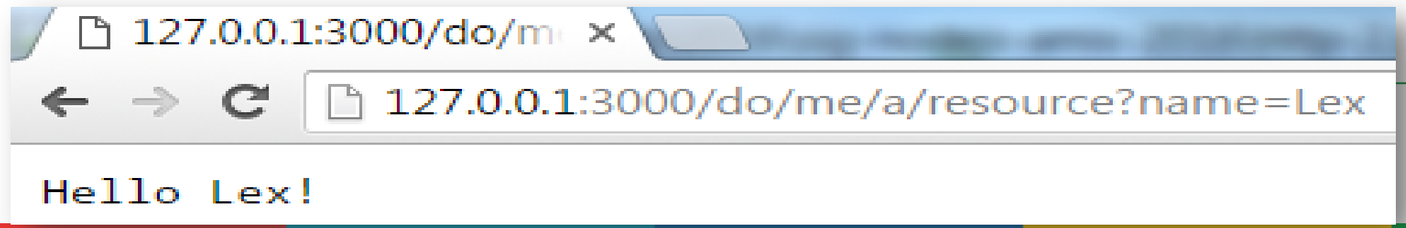
Web Tier

AMIS

node.js

INTERPRET REQUEST AND RESPOND APPROPRIATELY

```
// invoke: curl http://127.0.0.1:3000/do/me/a/resource?name=Lex
var http = require('http')
,   url = require('url') ;
var server = http.createServer(function handleRequest(req, res) {
    console.log('URL '+ req.url);
      var queryObject = url.parse(req.url,true).query;
      var name = queryObject.name;
    console.log('path: '+url.parse(req.url).pathname);
    console.log('queryObject: '+JSON.stringify(queryObject));
    res.write('Hello '+ name + '!');
    res.end();
}).listen(3000);
```



http(s) request

Web Tier

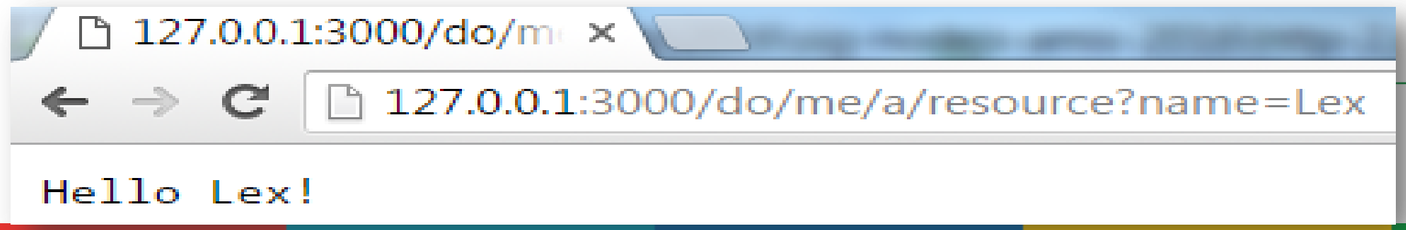
INTERPRET REQUEST AND RESPOND APPROPRIATELY

```
// invoke: curl http://127.0.0.1:3000/do/me/a/resource?name=Lex
var http = require('http')
,   url = require('url') ;
var
    res.write('Hello '+ name + '!');
    res.end();
}).listen(3000);
```

```
server running on port 3000
URL /do/me/a/resource?name=Lex
path: /do/me/a/resource
queryObject: {"name":"Lex"}
URL /favicon.ico
path: /favicon.ico
queryObject: {}
```

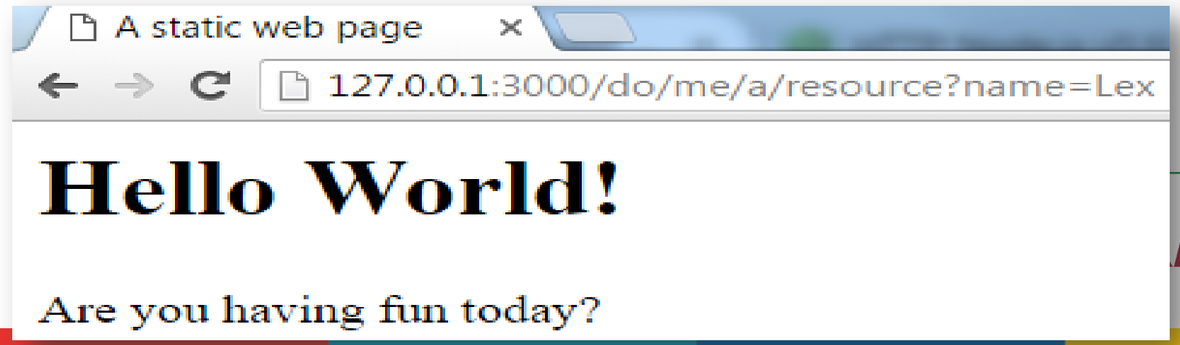
http(s) request

Web Tier



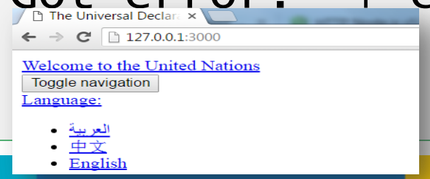
STATIC FILE SERVING

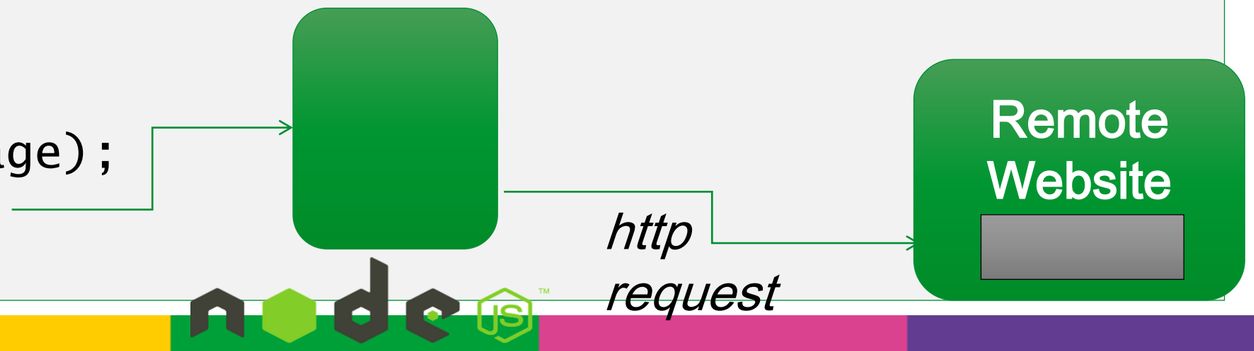
```
// serve static file: /public/index.html
var http = require('http')
,   fs= require('fs');

var server = http.createServer(function handleRequest(req, res) {
    res.writeHead(200, { 'content-type': 'text/html' });
    fs.createReadStream('./public/index.html').pipe(res);
}).listen(3000);

console.log('server running on port 3000');
```



REMOTE RESOURCE SERVING – HTTP IN AND HTTP OUT

```
var http = require('http');
var options = {
  host: 'www.un.org',   path: '/en/universal-declaration-human-rights/' };

var server = http.createServer(function handleRequest(req, res) {
    res.writeHead(200, { 'content-type': 'text/html' });
    http.get(options, function handleRemoteResponse(resp) {
       var body="";
       resp.on("data", function(chunk) { //response returned in chunks
            body = body+chunk;
       });
       resp.on("end", function() { //when response is complete, pass it on
          res.end(body);
       });
    }).on('error', function(e) {
       console.log("Got error: "+ e.message);
    });
  }).listen(3000);
```





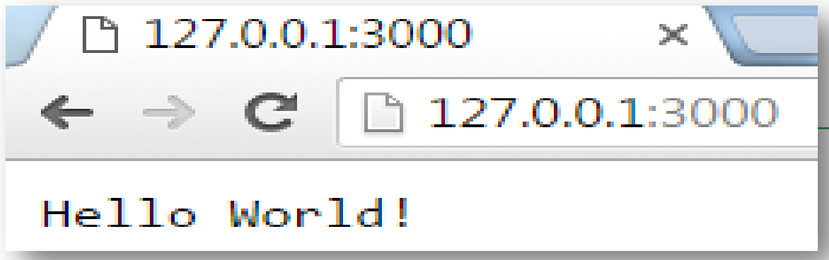
EXPRESS WEB APPLICATION FRAMEWORK

- Fast, unopinionated, minimalist web framework for Node.js
- Framework on top of core Node.js top facilitate incoming HTTP requests
  - “Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.”
- Convenient for
  - Web Server for static files
  - API implementation (REST)
  - Any incoming HTTP request handling
- Express is distributed as npm package
  - npm install express

EXPRESS - SIMPLEST FORM OF HTTP HANDLING

```
// invoke from browser or using curl:  curl http://127.0.0.1:3000
var express = require('express')
,   http = require('http');

var app = express()
          .use(function (req, res, next) {
                res.end('Hello World!');
           });
// Create HTTP server with Express app as the request listener
http.createServer(app)
```

```
C:\Users\lucas_j>curl http://127.0.0.1:3000
Hello World!
```



http(s) request

Web Tier

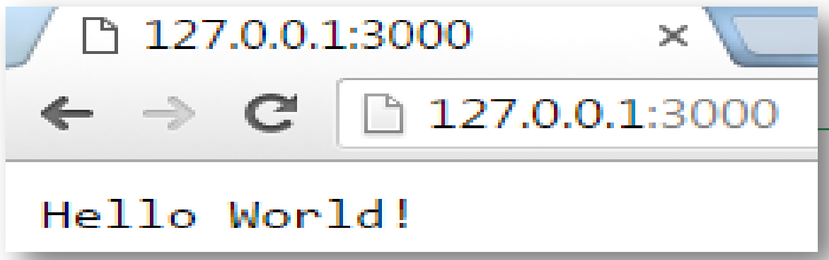
AMIS

node express

EXPRESS – EVEN SIMPLER FORM OF HTTP HANDLING (OR AT LEAST SIMPLER CODE)

```
// invoke from browser or using curl:  curl http://127.0.0.1:3000
var express = require('express');

express().use(function (req, res, next) {
                res.end('Hello World!');
            }).listen(3000);

console.log('server running on port 3000');
```

```
C:\Users\lucas_j>curl http://127.0.0.1:3000
Hello World!
```



http(s) request

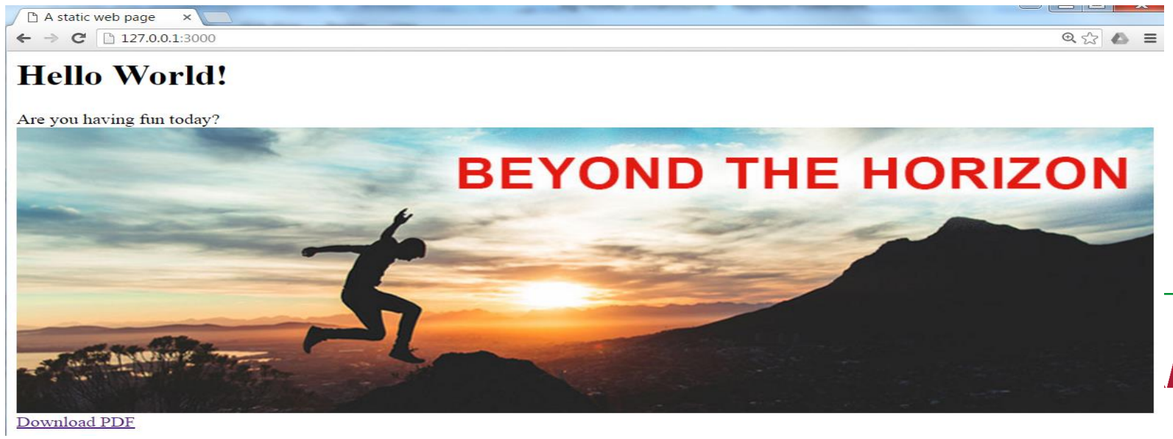
Web Tier

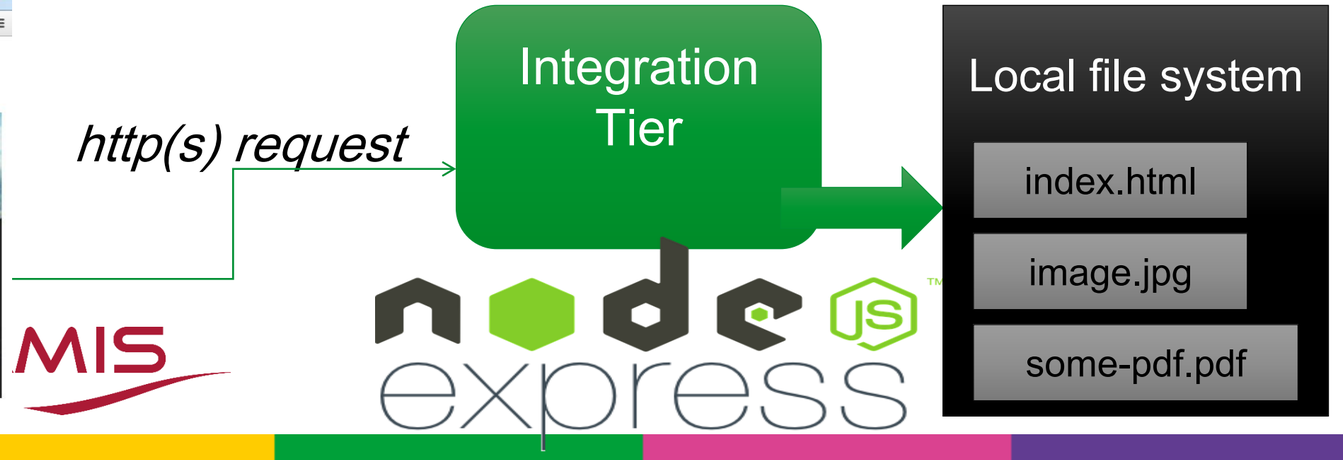
AMIS

node express

STATIC FILE SERVING

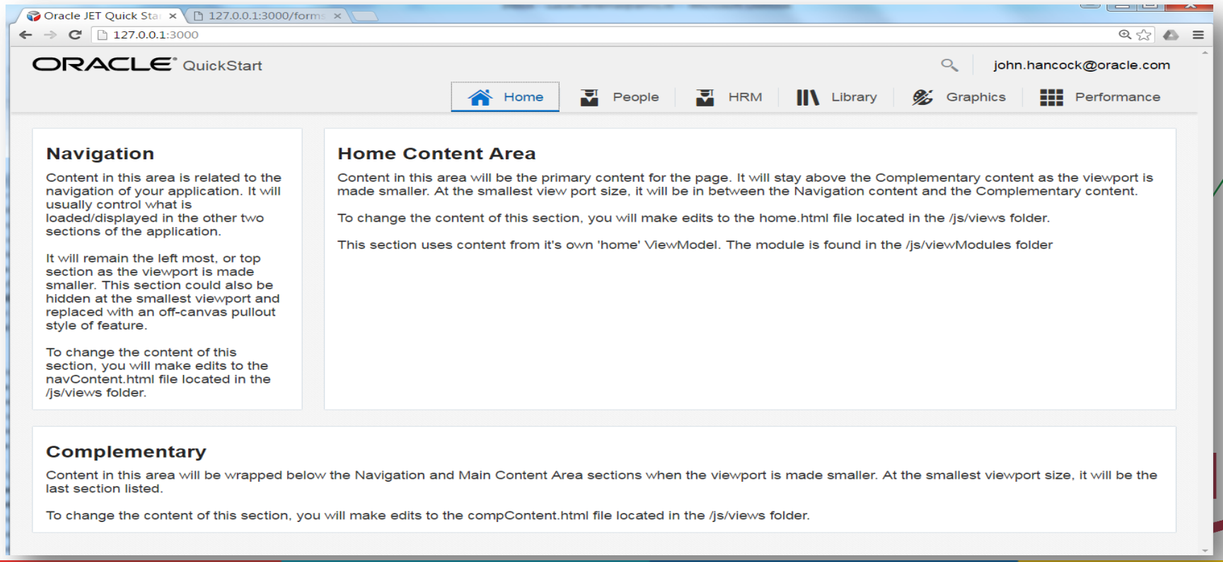
```
// static file server for all resources in public and below
// will server public/index.html when invoked at http://127.0.0.1:3000
var express = require('express'); //npm install express

express().use(express.static(__dirname + '/public'))
    .listen(3000);

console.log('server running on port 3000');
```

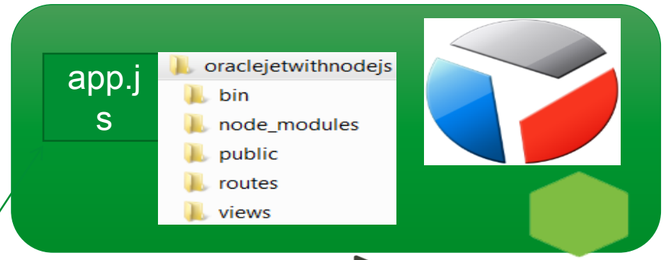




ORACLE JET ON NODE.JS

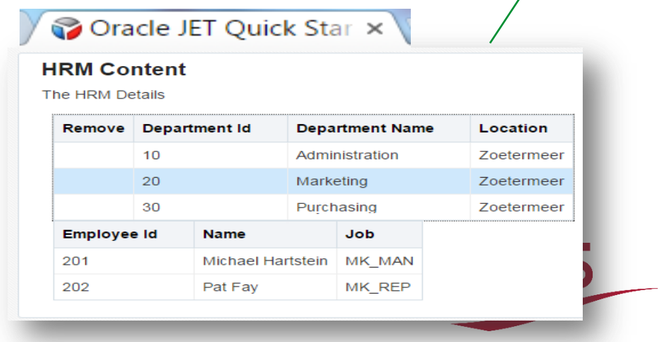
- Oracle JET is a rich client web application
- All the action is on the client
- Role of Node.js: serve the JET application with all its required resources
  - All static HTML, CSS and JS files

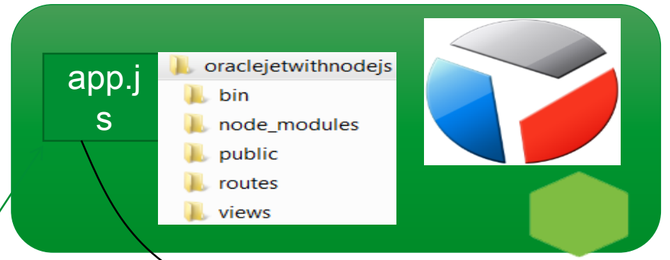


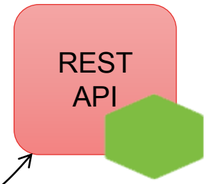


ORACLE JET ON NODE.JS

- Oracle JET is a rich client web application
- All the action is on the client
- Role of Node.js: serve the JET application with all its required resources
  - All static HTML, CSS and JS files
- Additionally: Node.js can provide back end APIs and proxy that JET application leverages

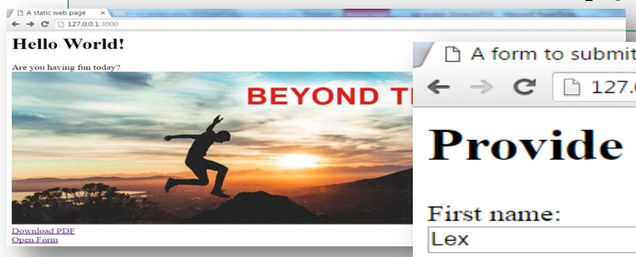


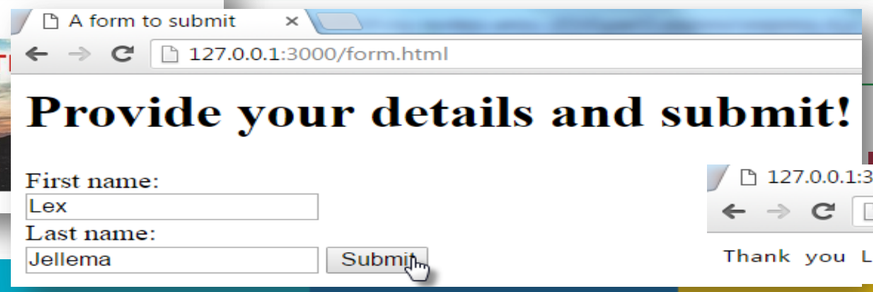




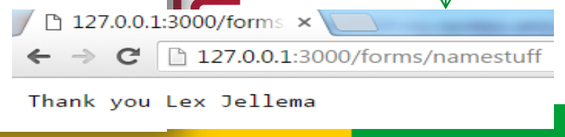
HANDLING FORM SUBMIT AND OTHER HTTP POST REQUESTS

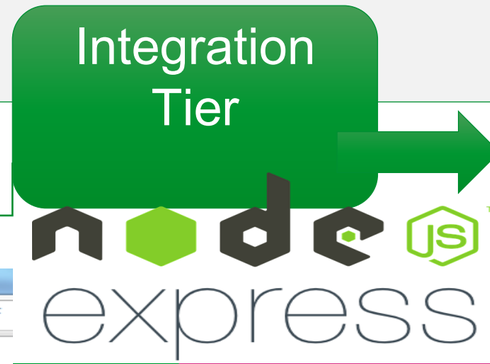
```
// static file server for all resources in public and below
// AND handle forms submission to path /forms/...
var express = require('express'); //npm install express
var bodyParser = require('body-parser'); // npm install body-parser

var app = express()
            .use(bodyParser.urlencoded({ extended: true}))
                  .post('/forms/*', function (req, res) { //process
                     console.log(JSON.stringify(req.body));
                          res.end('Thank you '+ req.body.firstname
                            +' '+req.body.lastname);
                      })
            .use(express.static(__dirname + '/public'))
                  .listen(3000);
```

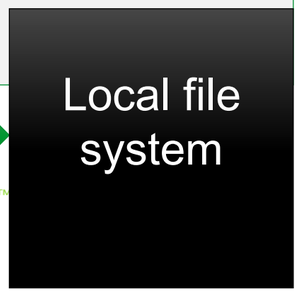




POST request



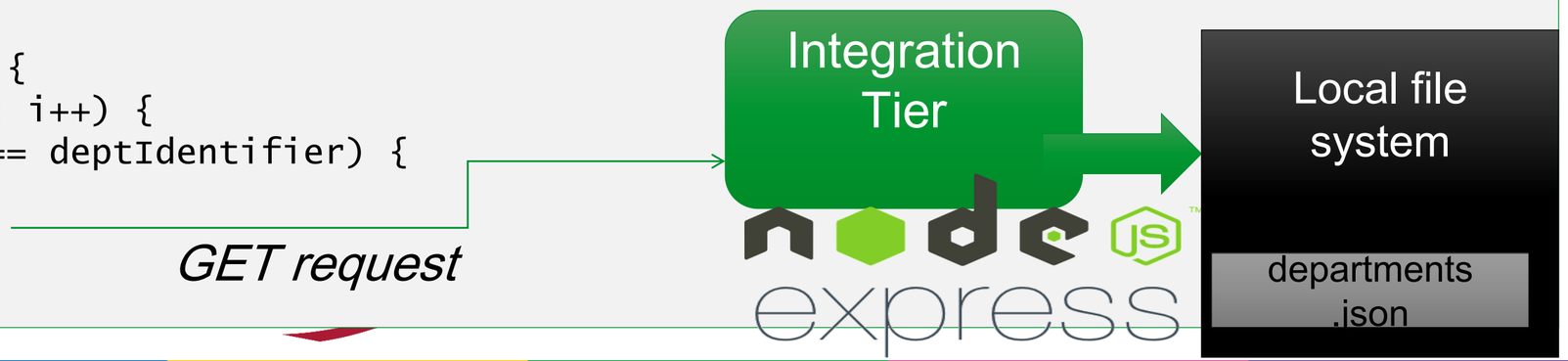




HANDLING REST GET REQUESTS

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

var departments= JSON.parse(fs.readFileSync('departments.json', 'utf8'));
var app = express()
                .get('/departments/:departmentId', function (req, res) {
      var department = getDepartment(req.params['departmentId']);
      console.log(JSON.stringify(department));
      res.send( department);
                 })
                .get('/departments', function (req, res) {
                       res.send( departments);
                 })
           .use(express.static(__dirname + '/public'))
                .listen(3000);
```

```
function getDepartment(depIdentifier) {
  for (var i=0; i< departments.length; i++) {
    if (departments[i].DEPARTMENT_ID == deptIdentifier) {
          return departments[i];
    }//if
  }//for
}// getDepartment
```

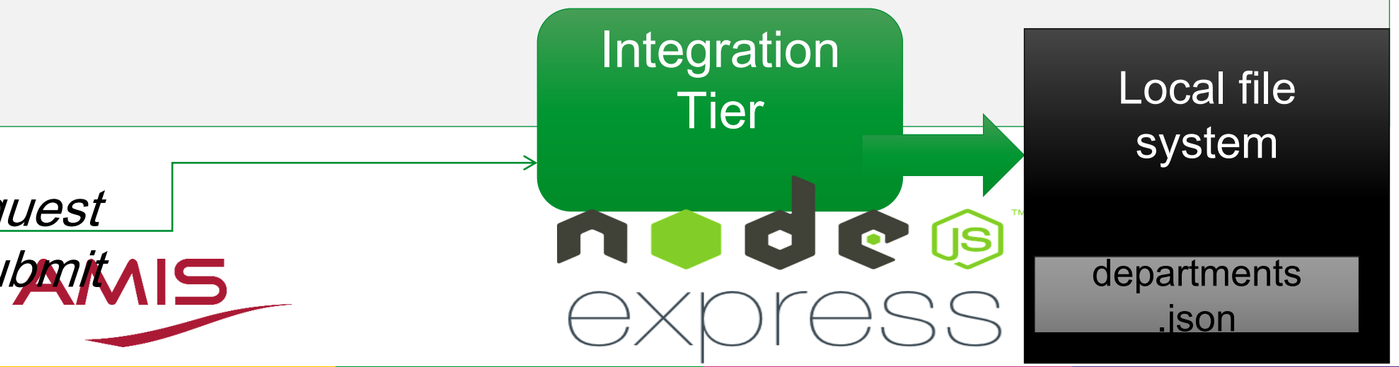


HANDLING REST GET & FORM POST REQUESTS

```
var express = require('express'),bodyParser = require('body-parser')
,   fs = require('fs');

var app = express()
                .use(bodyParser.urlencoded({  extended: true}))
                .post('/forms/department', function (req, res) { //process
                 console.log(JSON.stringify(req.body));
                 departments.push( {"DEPARTMENT_ID":req.body.departmentId
                                   ,"DEPARTMENT_NAME":req.body.departmentName});
                  res.end('Thank you for the new department '+
                        req.body.departmentId+" "+req.body.departmentName);
                   })
          .get('/departments/:departmentId', function (req, res) {
           …
                .listen(3000);
```

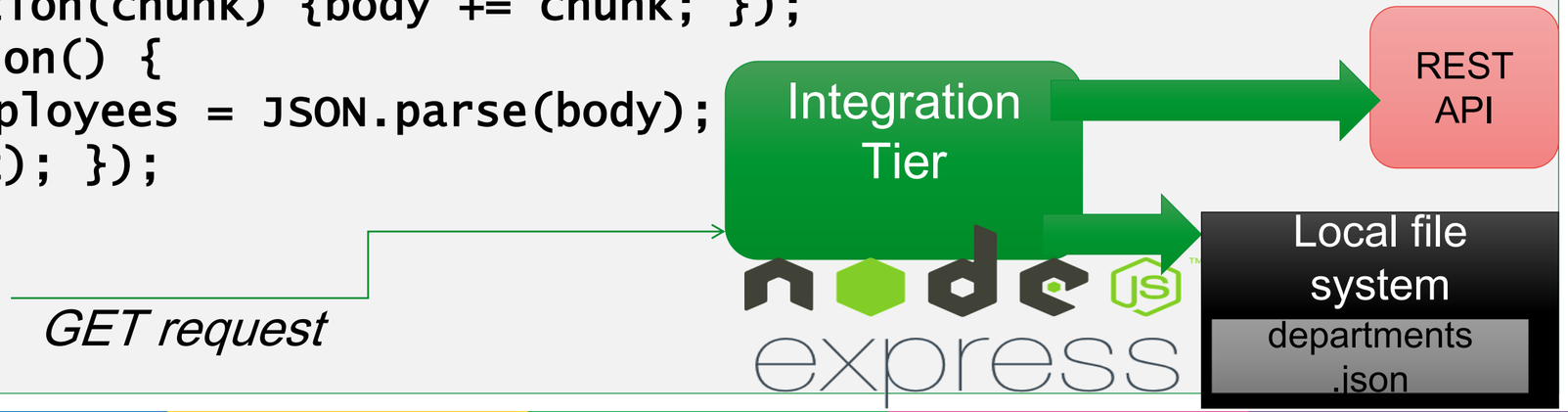
GET request

Form Submit



SERVE DEPARTMENT DETAILS – RETRIEVED FROM EXTERNAL REST API

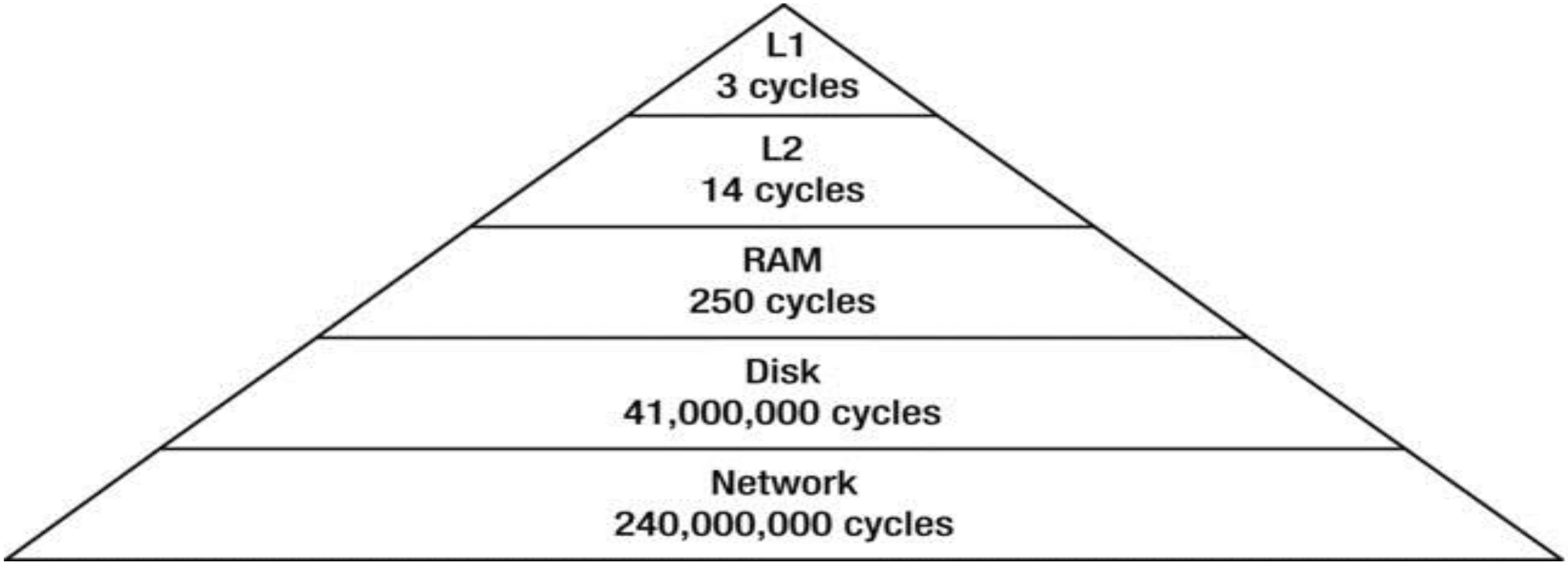
```
var express = require('express'),bodyParser = require('body-parser')
,   fs = require('fs'), https = require('https');
var app = express()
    .get('/departmentdetails/:departmentId', function (req, res) {
          var departmentId = req.params['departmentId'];
          var department = getDepartment(departmentId);
          // get employee details for department from remote API
          https.get({ host: 'data-api-oraclecloud.com', port: 443,
                    path: '/departments/'+departmentId, method: 'GET'
                  }, function handleRemoteResponse(resp) {
        var body="";
        resp.on("data", function(chunk) {body += chunk; });
        resp.on("end", function() {
                department.employees = JSON.parse(body);
          res.send(department); });
        });
      })
    .listen(3000);
```



AND NOW FOR A REAL USP

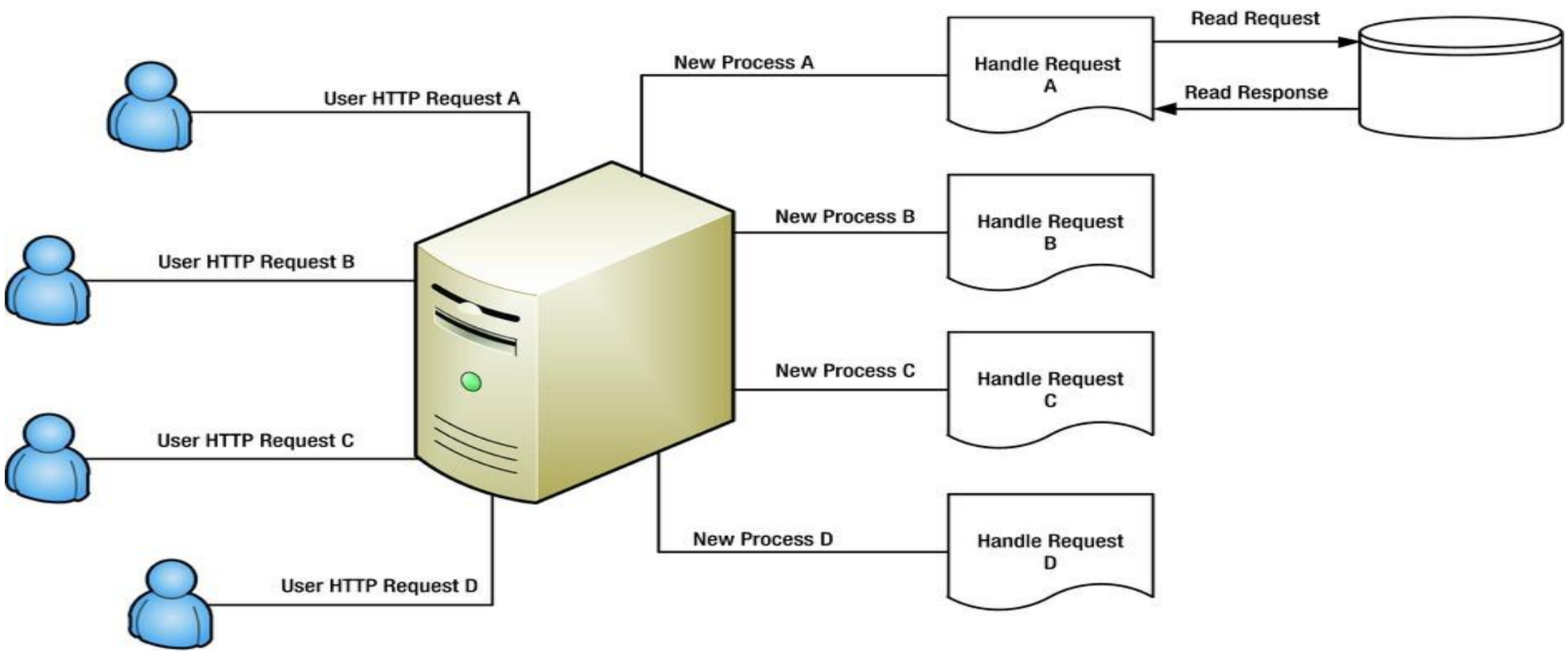
ACCESS SPEED FOR DATA RETRIEVED FROM VARIOUS SOURCES

Source: Beginning Node.js - by Basarat Ali Syed Apress, released: December 2014



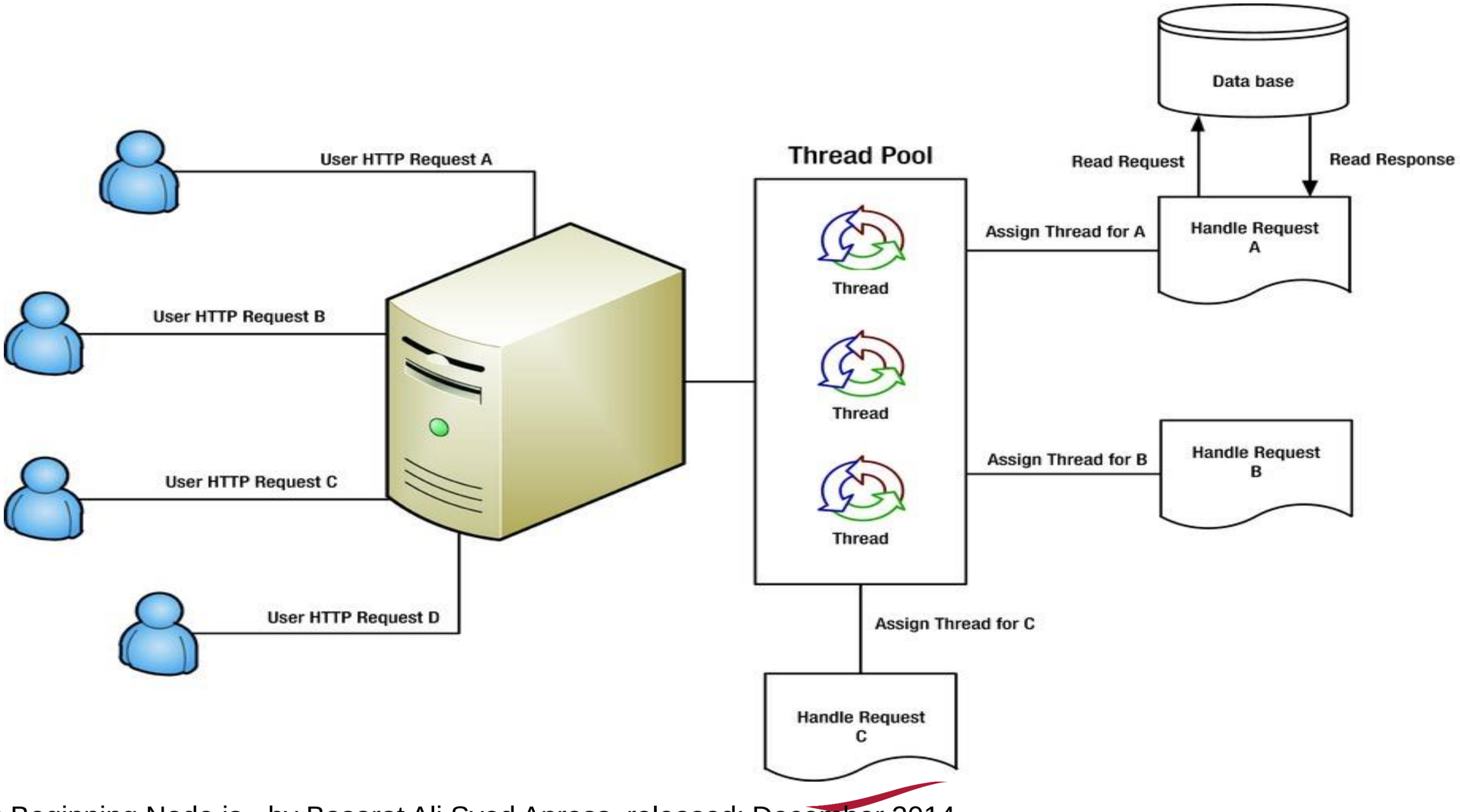
AMIS

THREADS IN WEB APPLICATION HANDLING HTTP-REQUESTS



Source: Beginning Node.js - by Basarat Ali Syed Apress, released: December 2014

READY-TO-RUN THREADS IN WEB APPLICATION HANDLING HTTP-REQUESTS



Source: Beginning Node.js - by Basarat Ali Syed Apress, released: December 2014

shopping area

Warehouse

cash registers

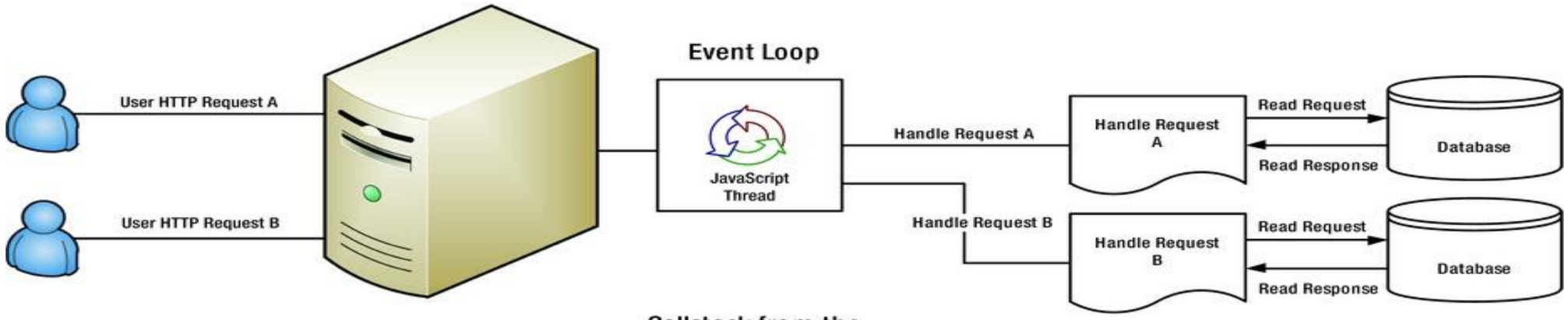
desk to collect large pieces

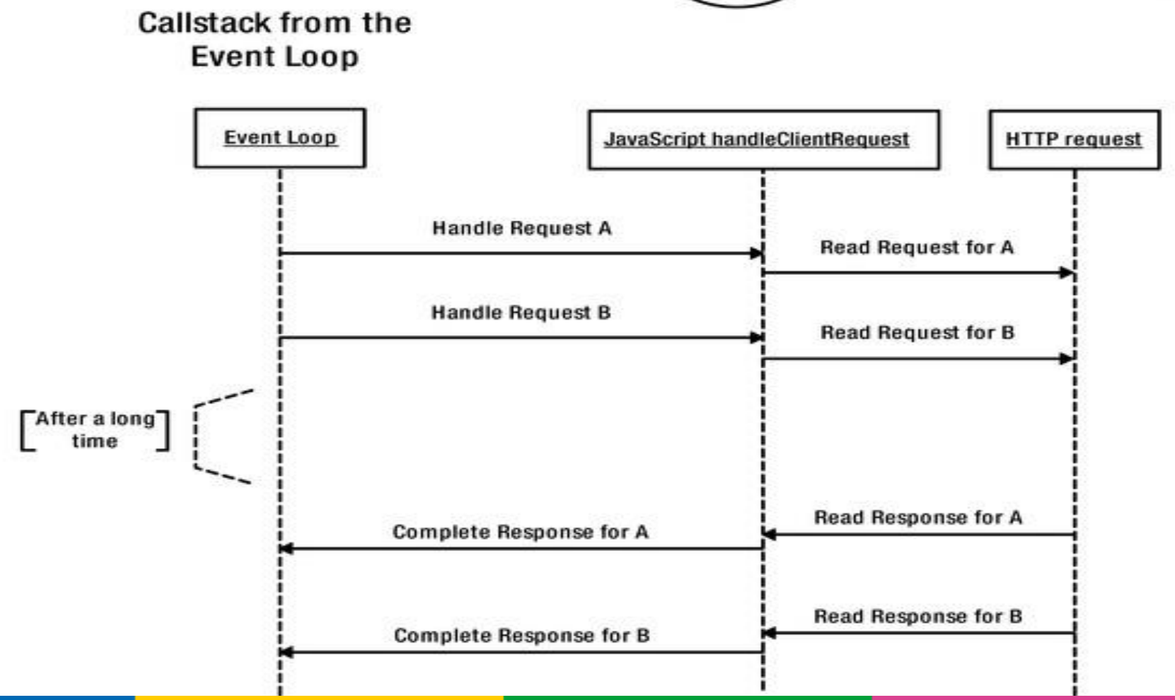
waiting area

Only a single desk to handle all customers who want to have stuff from the warehouse brought to them – compare with over a dozen cash registers

Exit

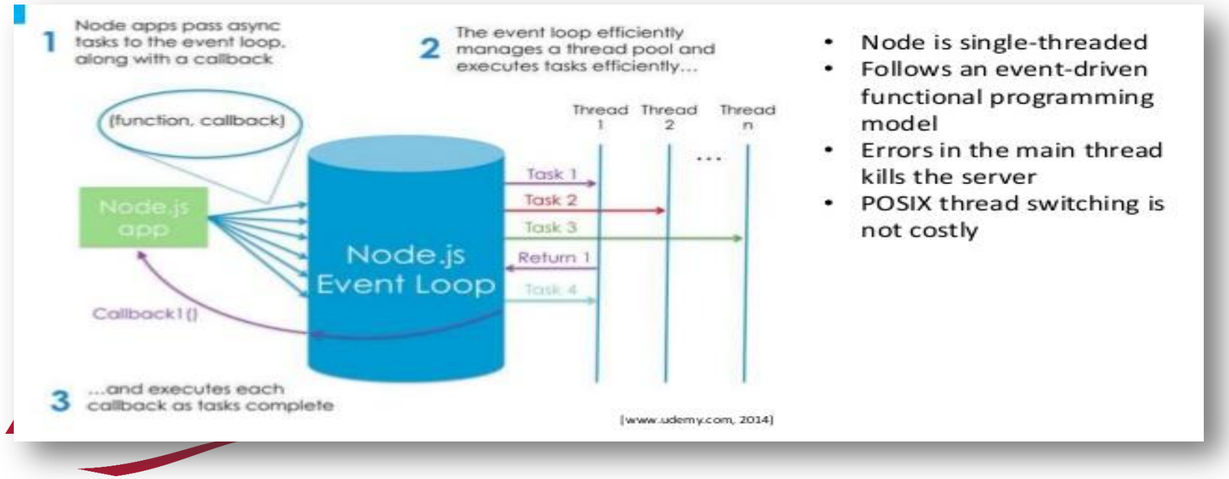
SINGLE THREAD MODEL IN NODE.JS



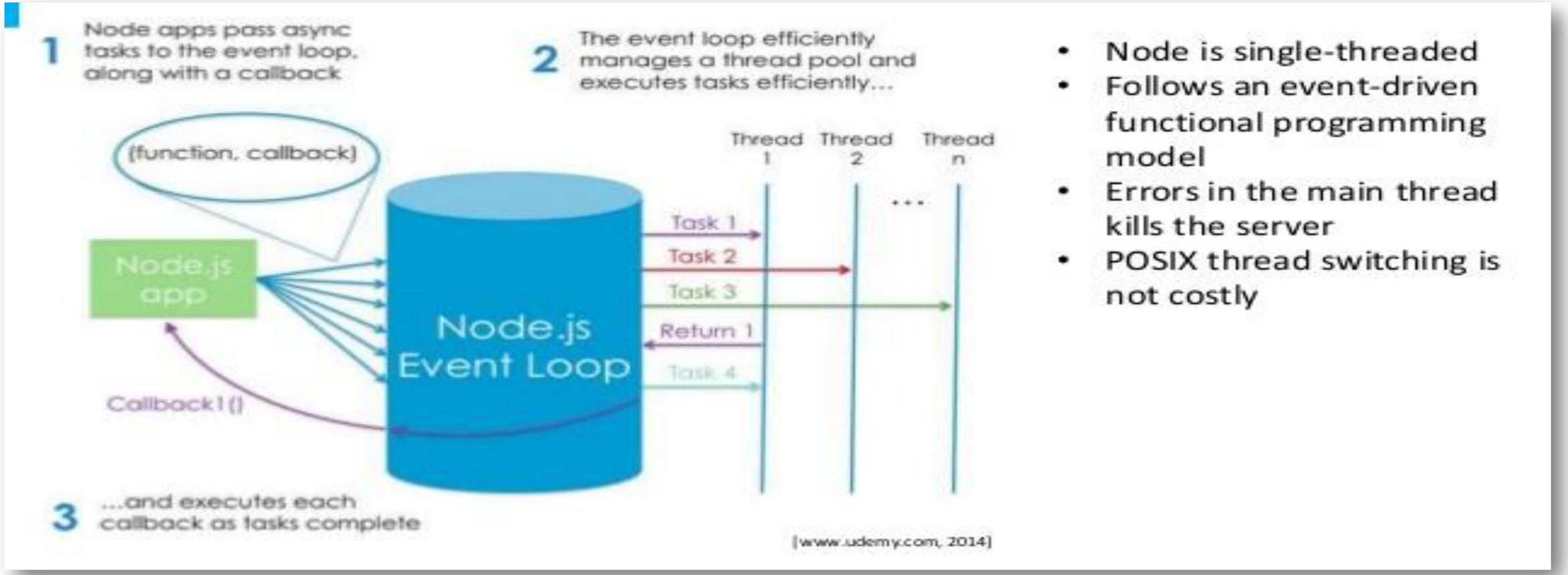


SINGLE THREAD

- No synchronous calls or blocking waits are done on the thread
  - intensive work too is pushed to another process
- After the question is asked, the thread moves
- When a response arrives – as an asynchronous event – the callback function is given the opportuniy to act on it
  - Again, as quickly as possible, not tying up shared resources!



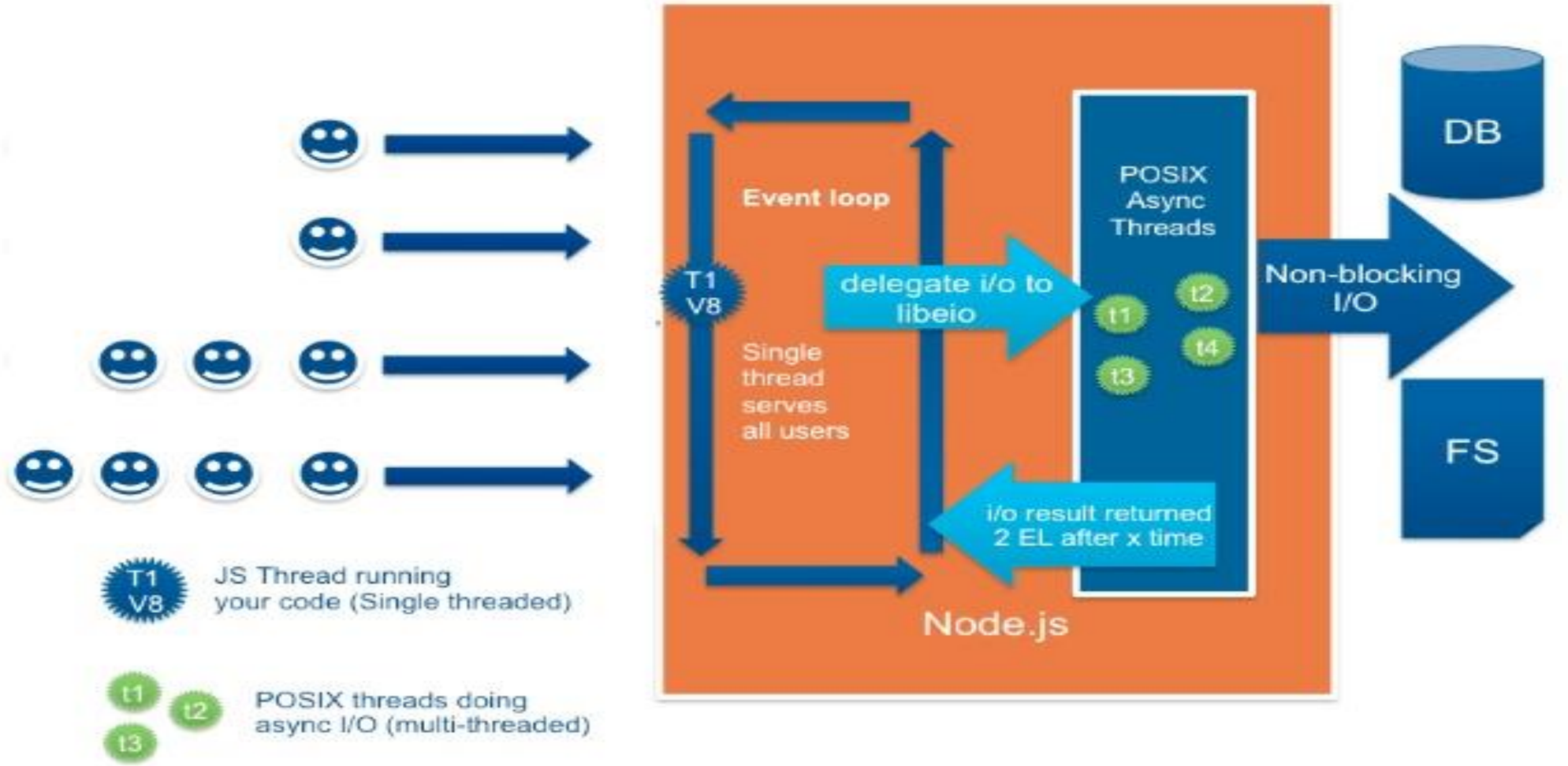
SINGLE THREAD



- Node is single-threaded
- Follows an event-driven functional programming model
- Errors in the main thread kills the server
- POSIX thread switching is not costly

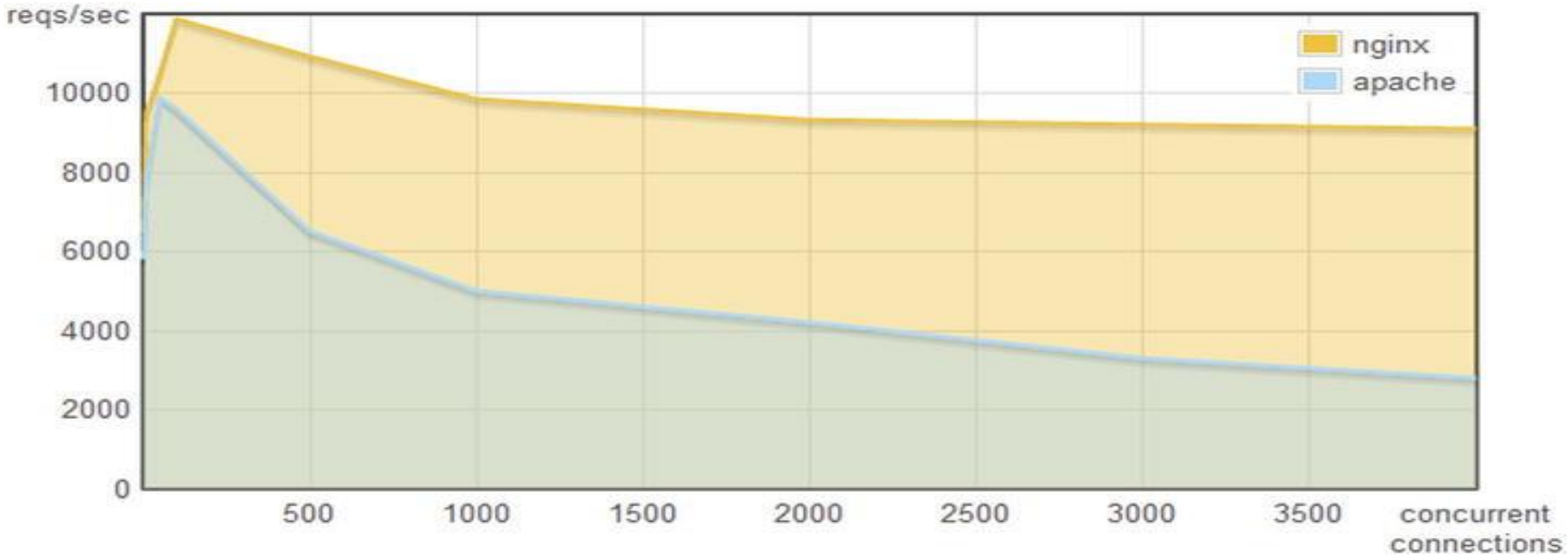
AMIS

Node.js - Single Thread, Event-ed



*Credit: blog.cloudfoundry.com

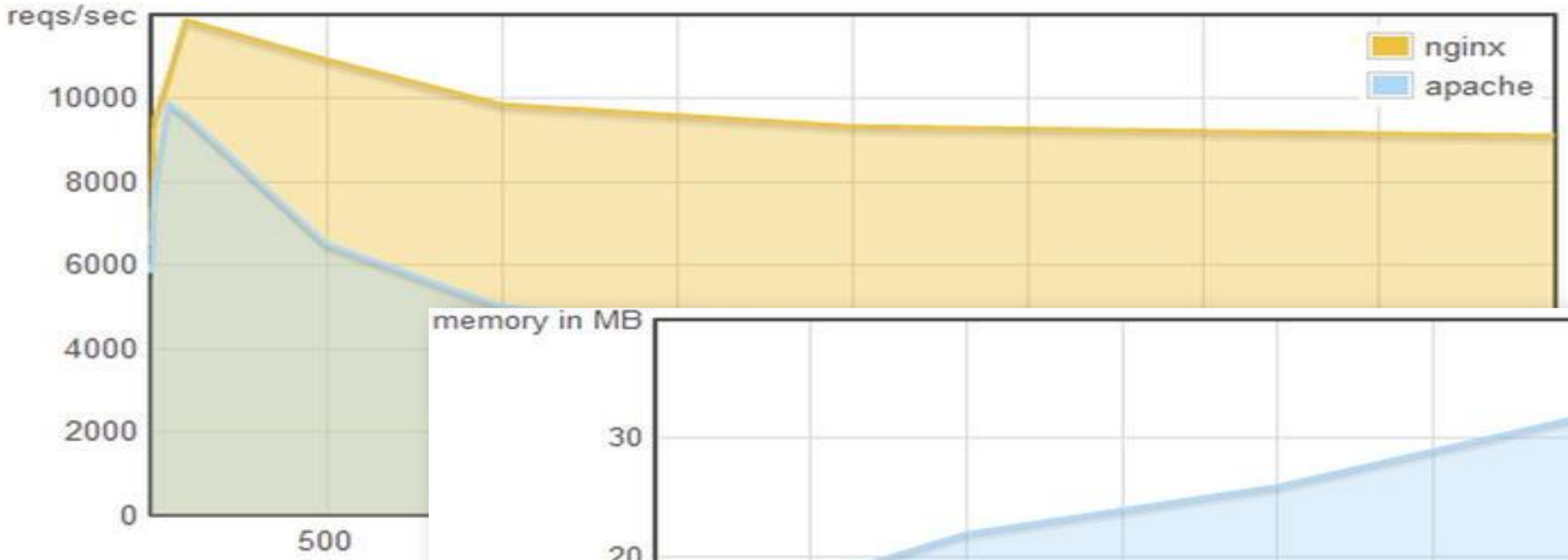
THE SINGLE THREAD WAY: NGINX

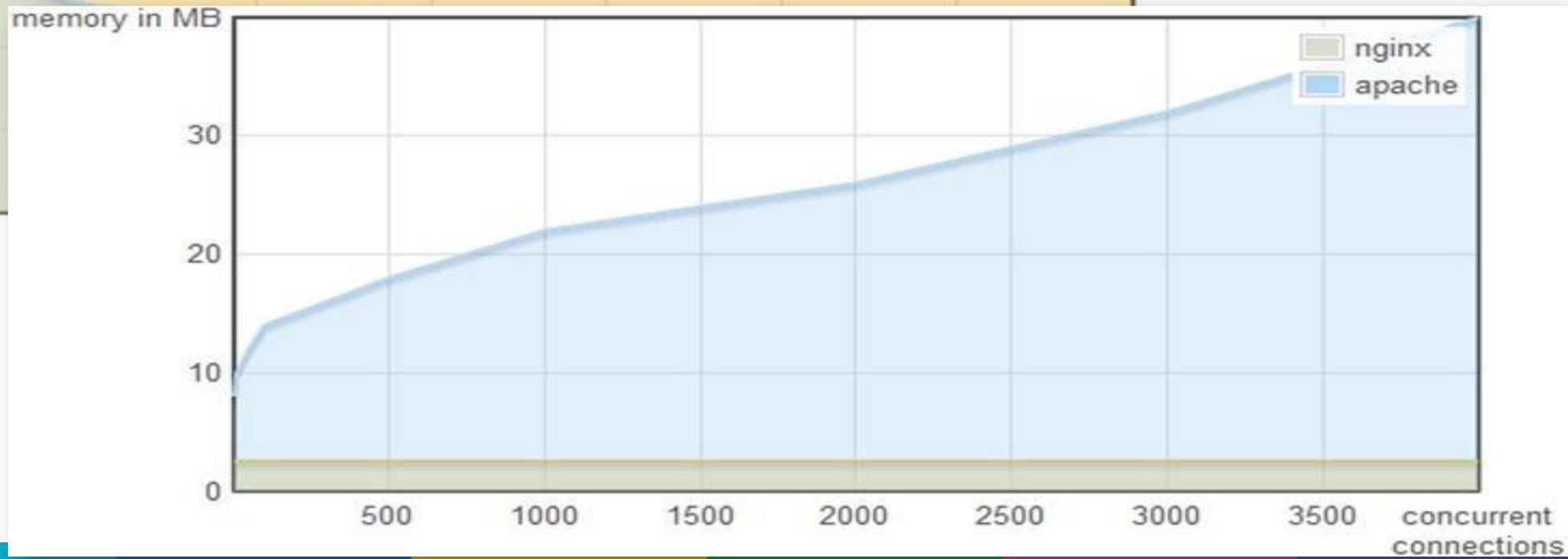


NGINX

AMIS

THE SINGLE THREAD WAY: NGINX





ASYNCHRONOUSLY ASSEMBLING THE HTTP RESPONSE

- Receive HTTP request
  - Possibly leverage cookie to re-establish state/context
- Set work in motion to collect data (remote API call, DB call) and register callback – with original request scope
- Stop processing, vanish, release resources
- When callback is invoked, use response to construct the response
  - Complete and send the response

Callback == Closure

(function to execute with context to execute in)

HANDSON – PART 1

Fetch Sources and Labs from GitHub

Install Node v6 or v7 locally (or in VM or in Docker Container)

Go through:

Part 1 – Hello World

Part 2 – Core HTTP

Part 3 – Express

Part 4 – Rich Client Web Application

AMIS

AGENDA

INTRODUCTION OF NODE (FKA NODE.JS)

SERVER SIDE JAVASCRIPT – HELLO WORLD!

THE NODE ECOSYSTEM

HTTP REQUEST HANDLING, EXPRESS & OUTBOUND HTTP CALLS

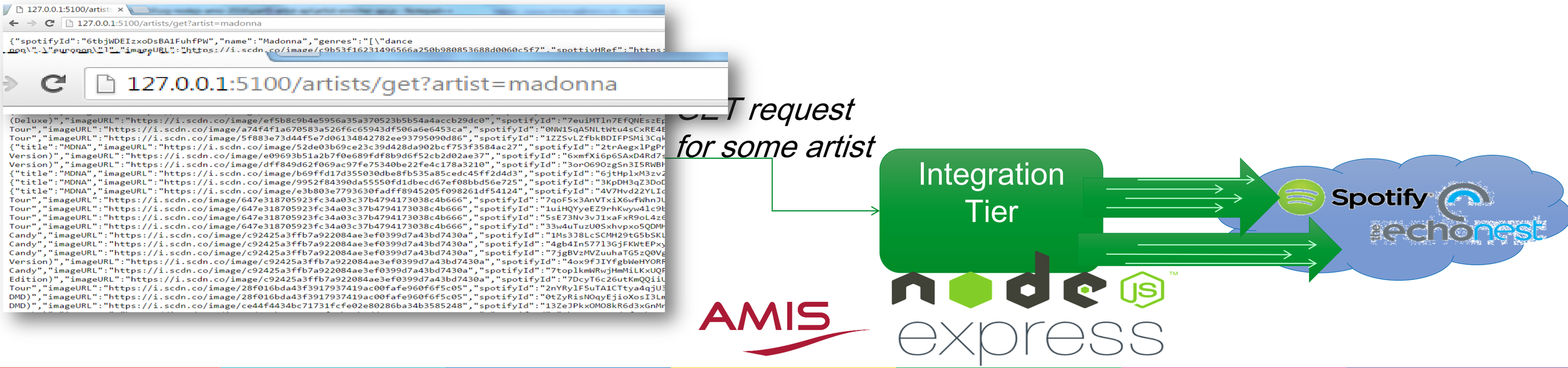
HANDSON WORKSHOP

SERVER PUSH, EMAILING, PROMISES AND INTERACTING WITH MONGODB & ORACLE

MORE HANDSON WORKSHOP

ARTIST API

- Provide rich JSON message for an artist
- External APIs invoked at Spotify
- External calls are made in parallel
  - Waiting is done in parallel
  - Constructing the response is done when all responses are in



ASYNC

- The node module async is a utility module which provides straight-forward, powerful functions for working with asynchronous JavaScript.
- Available for server side Node.js and for use in the browser client
- Program easily with asynchronous interactions
  - Coordinate parallel or sequential calls and deal with errors
- Example operations:
  - Map
  - Reduce
  - Filter
  - Waterfall
  - Parallel
  - ForEachOf
  - Series

ASYNC – EXECUTE IN PARALLEL

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

var obj = {dev: "/dev.json", test: "/test.json", prod: "/prod.json"};
var configs = {};

async.forEachOf(obj, function (value, key, callback) {
       … // work on populating configs


}, function (err) { // when all parallel actions are done – do something
    if (err) { console.error(err.message); return;}
    // configs is now a map of JSON data
    doSomethingWith(configs);
})
```

For each property in the object…

… execute this function

When all parallel branches under forEachOf are done, do this function

ASYNC – EXECUTE IN PARALLEL

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

var obj = {dev: "/dev.json", test: "/test.json", prod: "/prod.json"};
var configs = {};

async.forEachOf(obj, function (value, key, callback) {
    fs.readFile(__dirname + value, "utf8", function (err, data) {
        if (err) return callback(err);
        try {
            configs[key] = JSON.parse(data);
        } catch (e) {
            return callback(e);
        }
        callback();
    });
}, function (err) { // when all parallel actions are done - do something
    if (err) { console.error(err.message); return;}
    // configs is now a map of JSON data
    doSomethingWith(configs);
})
```

For each property in the object…

… execute this function

Asynchronously read the file

.. and when done, call this function

Notify forEachOf of the completion of this branch

When all parallel branches under forEachOf are done, do this function

ASYNC – EXECUTE IN WATERFALL

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

async.waterfall
( [  … // one or more functions that are executed in a chained fashion:
       // - fixed order
       // - passing output from one function as input into the next


  ], function (err, result) {
    // when all chained functions are done – do something
    if (err) { console.error(err.message); return;}
    // do something with result
})// end of waterfall
```

When all steps in the waterfall are complete, do this function

ASYNC – EXECUTE IN WATERFALL

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

async.waterfall
( [ function (callback) {
        … // do something that is potentially asynchronous
        callback( errorIfAny, result);
    },
    function (value, callback) {
        …
        callback(errorIfAny, result);
    },
    function (value, callback) {
        …
        callback(errorIfAny, result);
    }
  ], function (err, result) {
    // when all chained functions are done – do something
    if (err) { console.error(err.message); return;}
    // do something with result
})// end of waterfall
```

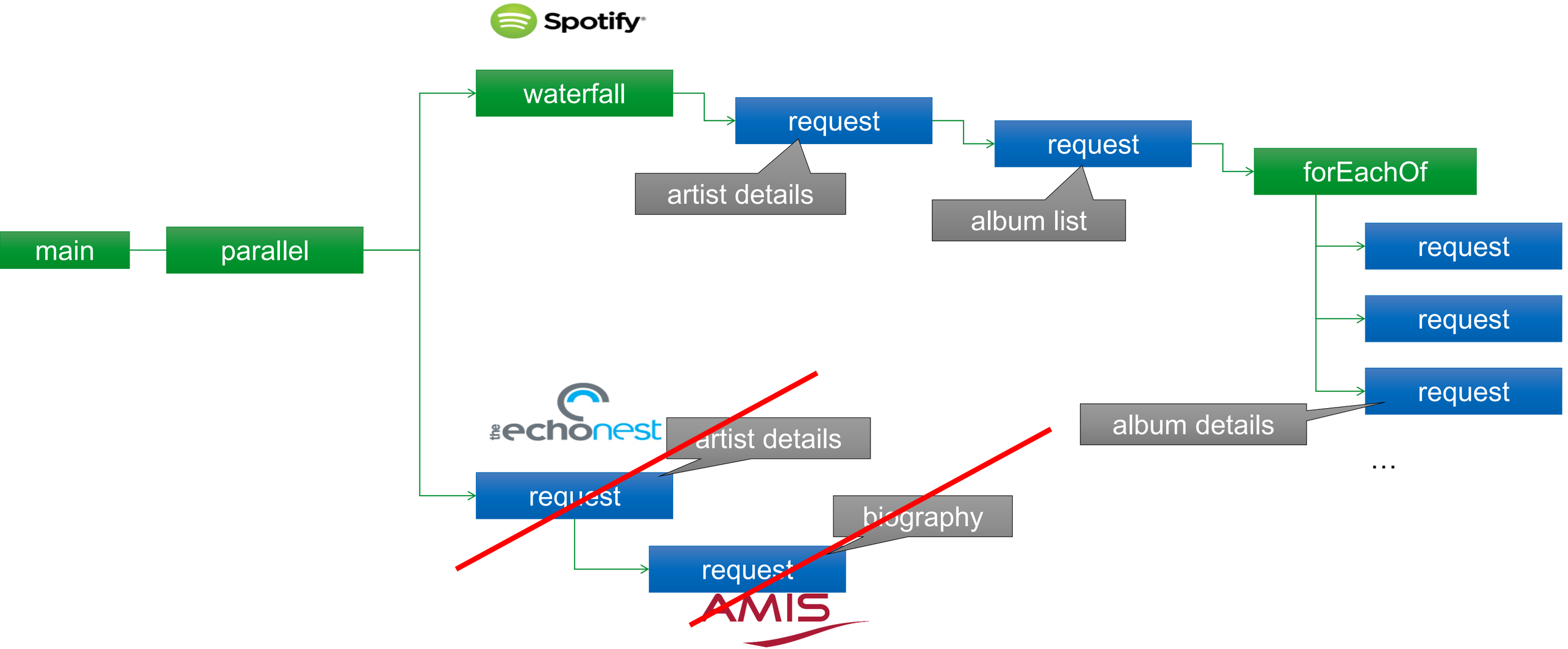
Start with first asynchronous function in in array

When done executing, then invoke callback to give async.waterfall control

When no error was reported, the next function in the waterfall is called

When all steps in the waterfall are complete, do this function

STRUCTURE OF ARTIST API



PROMISES IN JAVASCRIPT

> A promise is an abstraction for asynchronous programming. It's an object that proxies for the return value or the exception thrown by a function that has to do some asynchronous processing. — Kris Kowal on JSJ

- Promises in JavaScript are an alternative for explicit callbacks
  - A *promise* represents the later-to-be-provided-result of a call that will asynchronously be handled

```
function asynchronousAction(param) {
   return new Promise((resolve, reject) => { .. the real work
      … more work
      resolve( resultFromHardWork);
   });
}

asynchronousAction(paramValue)
.then( someOtherAsynchronousAction)
.then( function (result) { … do something with result })
```

PROMISES CAN BE EVALUATED IN PARALLEL AND IN A CHAIN

- Use Promise.all to wait for the resolution of a collection of promises

```
Promise.all( array of promises).then(function (array of results) {})
```

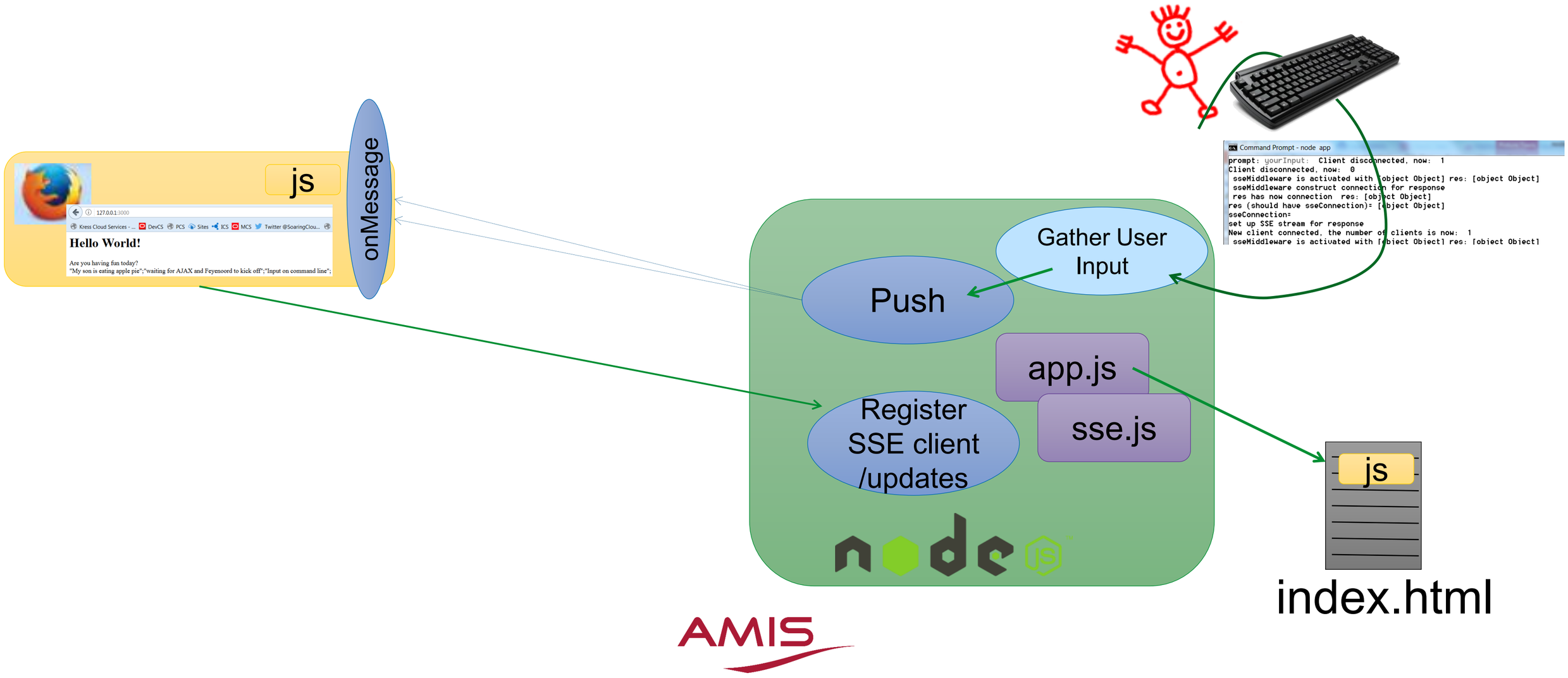
- Use a promise chain to handle a subsequent series of asynchronous actions that depend on each other

```
firstAction(paramValue)
.then( secondAction)
.then( nextAction)
.then( function (result) { … do something with result })
```

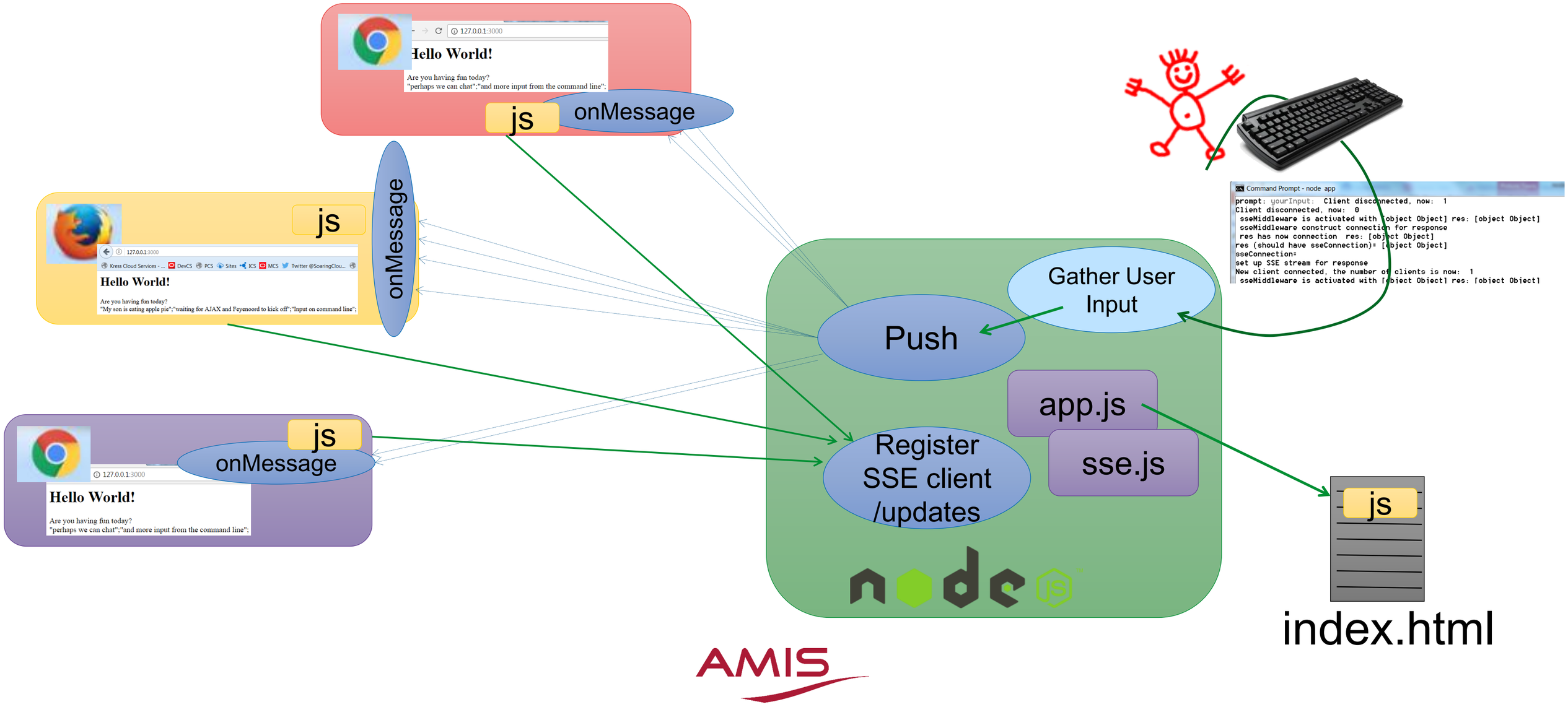
- Promise Chains can be included in Promise.all

AMIS

SERVER SENT EVENTS



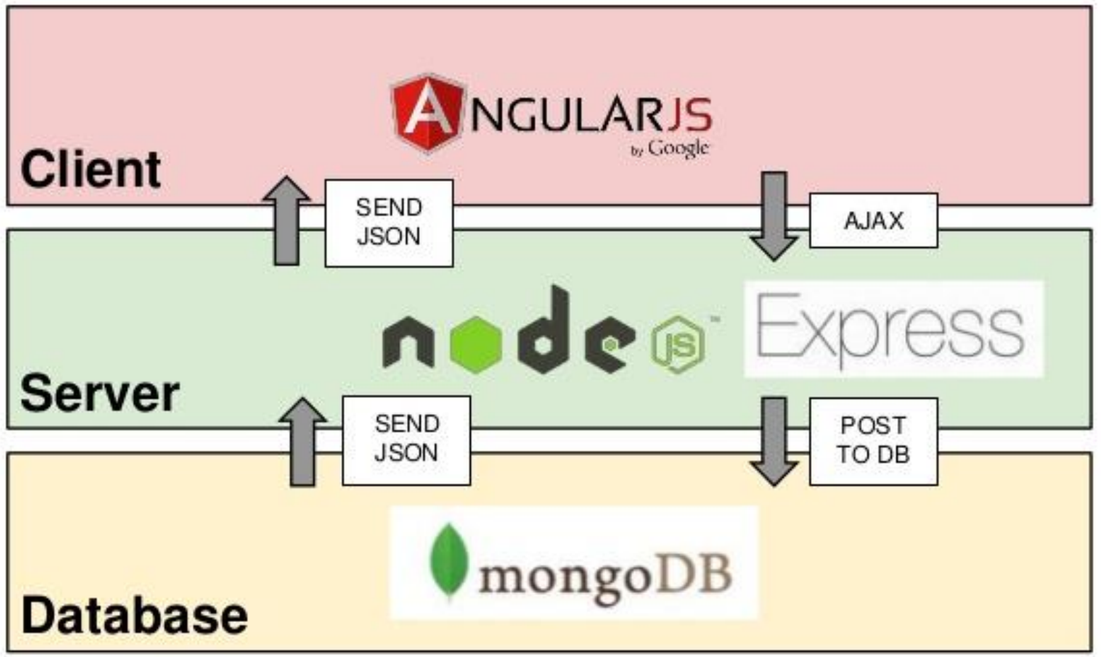
SERVER SENT EVENTS



AMIS

MEAN STACK

- Modelled after LAMP
- End-to-End JavaScript/JSON
- Term coined in 2013 (by MongoDB)

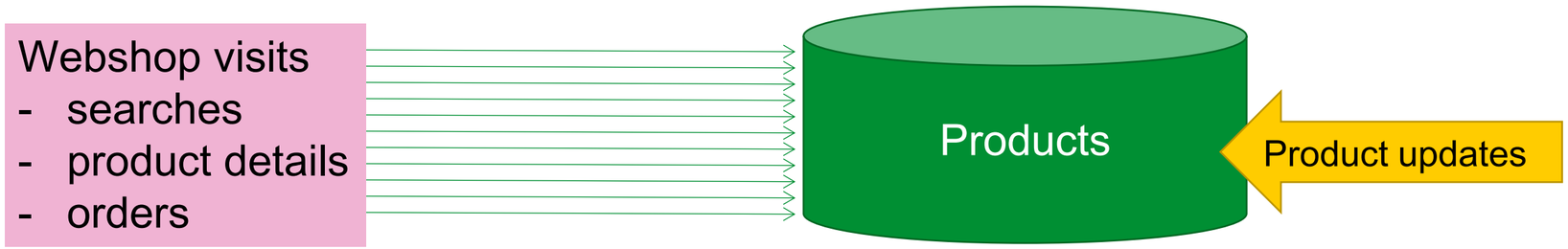


NOSQL – RELATIONAL DATABASES RE-EVALUATED

- Not all use cases require ACID (or can afford it)
  - Read only (product catalog for web shops)
  - Inserts only and no (inter-record) constraints
  - Big Data collected and “dumped” in Data Lake (Hadoop) for subsequent processing
  - High performance demands
- Not all data needs structured formats or structured querying and JOINs
  - Entire documents are stored and retrieved based on a single key
- Sometimes – scalable availability is more important than Consistency – and ACID is sacrificed
  - CAP-theorem states: Consistency [across nodes], Availability and Partition tolerance can not all three be satisfied

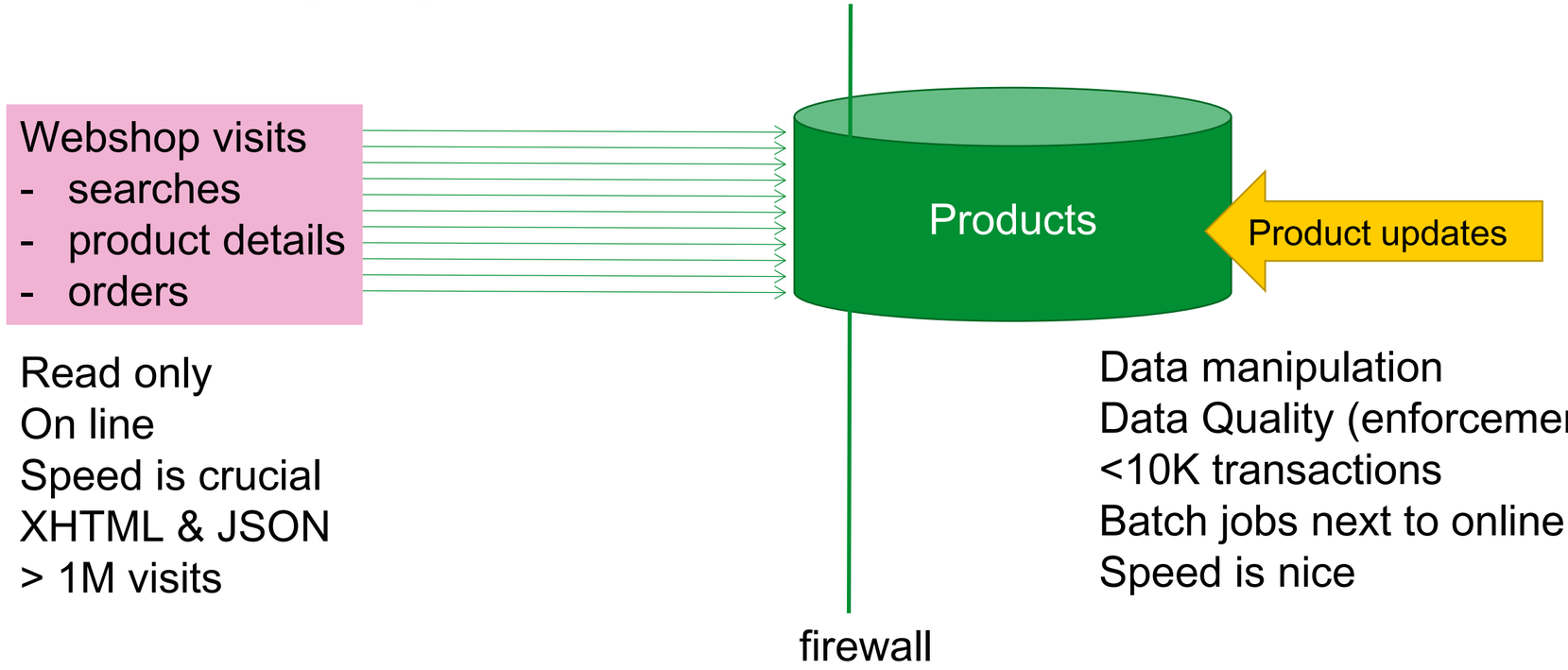
USE CASE: WEBSHOP

- Webshop – 1M visitors per day
- Product catalog consists of millions of records
  - The web shop presents: product description, images, reviews, pricing details, related offerings, stock status
- Products are added and updated and removed every day
  - Although most products do not change very frequently
  - Some vendors do bulk manipulation of product details

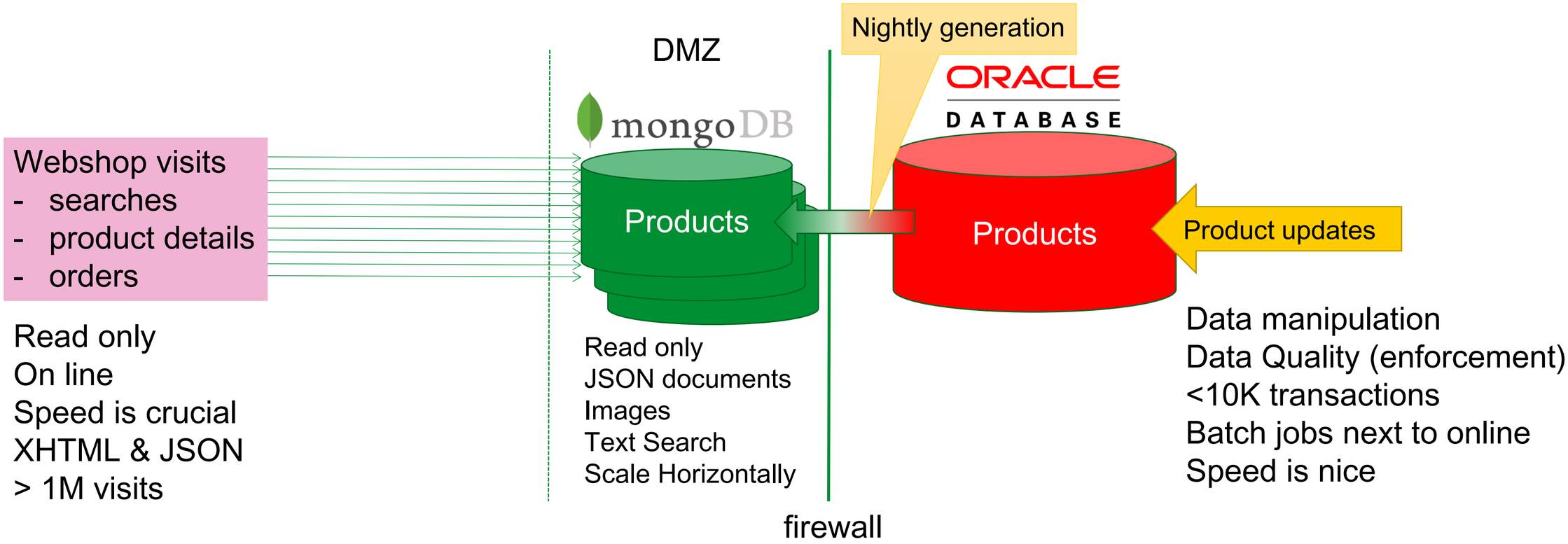


AMIS

USE CASE: WEBSHOP TECHNOLOGY AND ARCHITECTURE CONSIDERATIONS



USE CASE: WEBSHOP TECHNOLOGY AND ARCHITECTURE CONSIDERATIONS



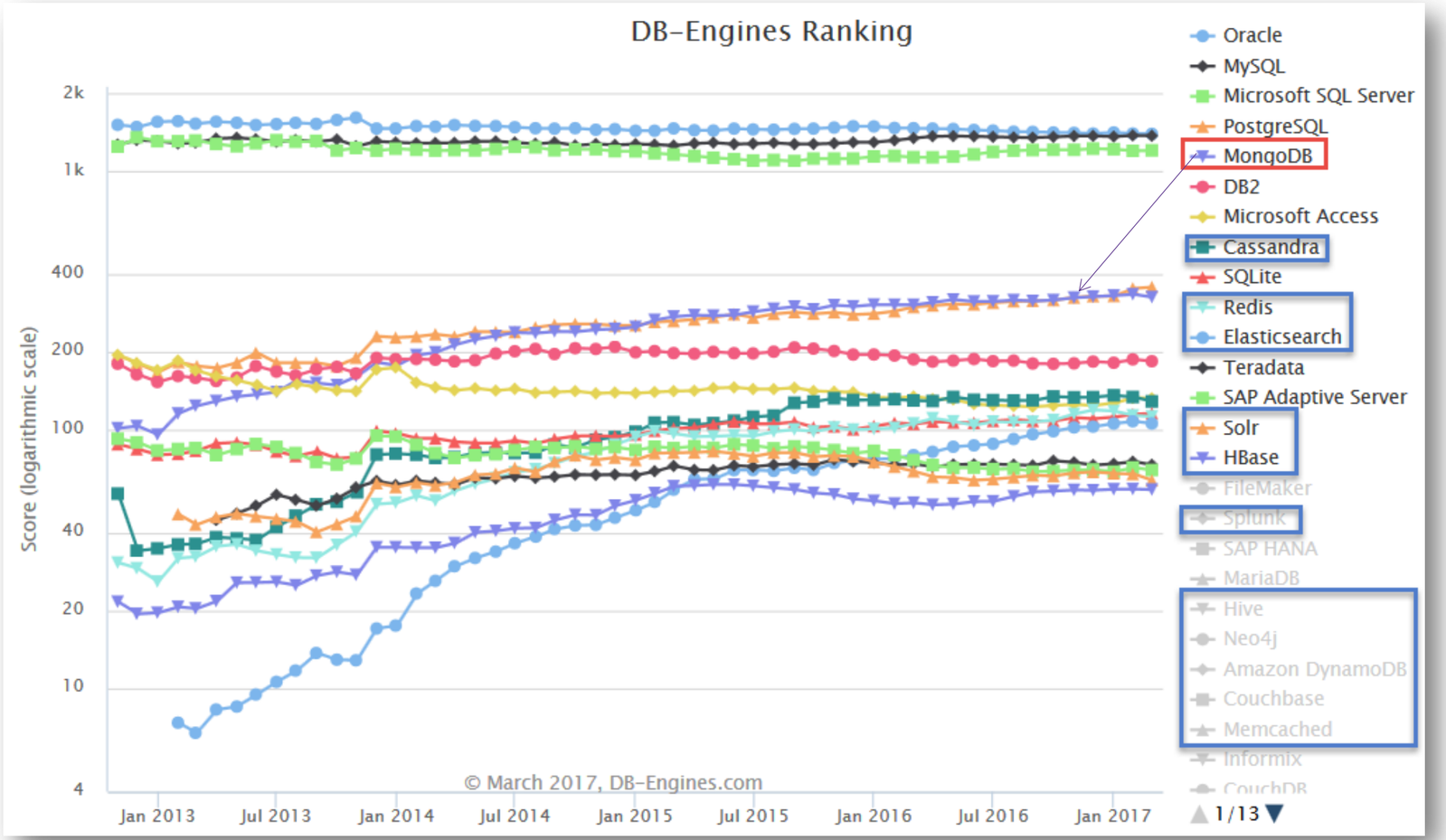
NOSQL AND BASE

- NoSQL arose because of performance and scalability challenges with Web Scale operations
- NoSQL is a label for a wide variety of databases that aspect of a true relational database
  - ACID-ness, SQL, relational model, constraints
- The label has been used since 2009
  - Perhaps NoREL would be more appropriate
- Some well known NoSQL products are
  - Cassandra, MongoDB, Redis, CouchDB, …
- BASE as alternative to ACID:
  - **b**asically **a**vailable, **s**oft state, **e***ventually consistent (after a short duration)*

AMIS

(LEADING) NOSQL DATABASE PRODUCTS

- MongoDB is (one of) the most popular (by any measure)
- Cloud (only):
  - Google BigTable,
  - AWS Dynamo
- Cache (in memory)
  - ZooKeeper, Redis, Memcached, …
- Hadoop/HDFS
- Oracle NoSQL (fka Berkeley DB)



AMIS

HISTORY OF MONGODB

- 10gen – startup from 2007 in New York City
  - Developed database as component in Platform as a Service product
- Initial release of MongoDB: 2009
  - 10gen changed its name to MongoDB Inc. in 2013
  - Offers enterprise support, MongoDB Atlas cloud service, MongoDB Compass, Connectors
- Current stable release: 3.4.2 (3.5.2 is in preview)
- Open Source - GNU Affero General Public License and the Apache License.
- MongoDB == Hu*mongo*us *D*ata*b*ase
- Evolution of MongoDB is still pretty fast
  - Some crucial enterprise database capabilities were added fairly recently or are still lacking
- MongoDB is number one NoSQL database in terms of popularity

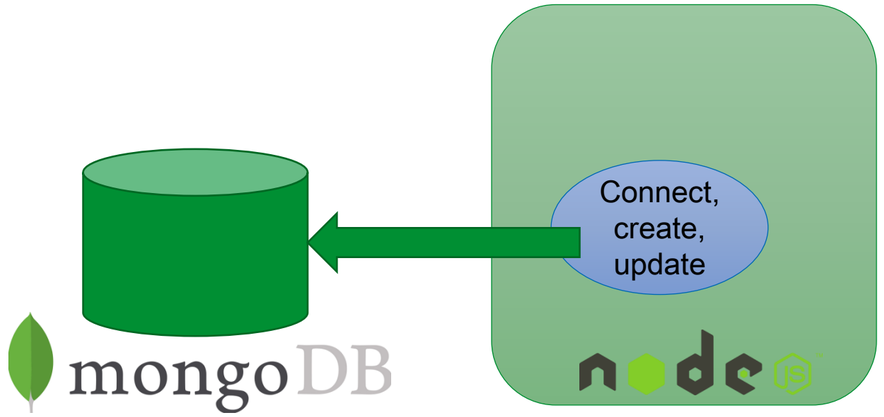
AMIS

CORE ARCHITECTURE

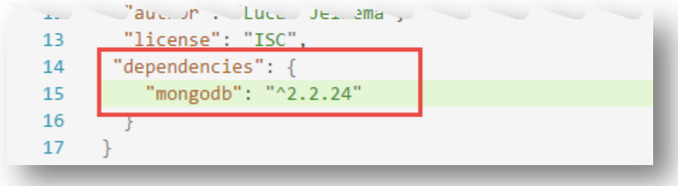
- MongoDB stores JSON documents in a binary format (BSON)
  - Documents are stored in collections (similar to row or records in tables)
- Interaction is through JavaScript
  - taking the place of SQL for DML and query
- Written in C++
- Has the V8 JavaScript engine included
- Runs on Mac OS X, Windows, Solaris, and most flavors of Linux
- Source code is on GitHub
- Has GeoSpatial and Tekst indexes and search capabilities
- MongoDB, Inc. officially supports drivers for C, C++, C#, Erlang, Java, **Node.js**, JavaScript, Perl, PHP, Python, Scala, and Ruby

AMIS

INTERACTING WITH MONGODB FROM NODE.JS



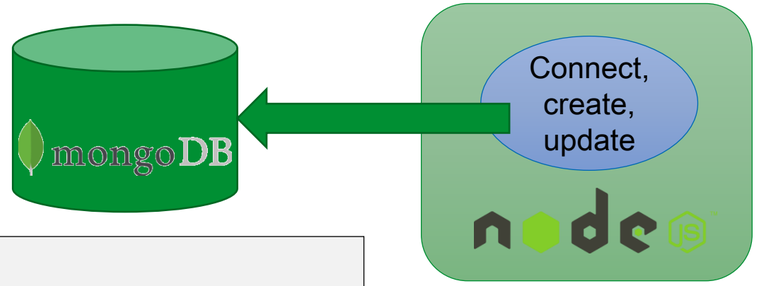
- MongoDB Node.js Driver 2.0
  - Support for ECMAScript 6.0 – Promises for asynch
  - http://mongodb.github.io/node-mongodb-native/2.0/
- Using mongodb in a NodeJS application
  - npm install mongodb –save



  - var MongoClient = require('mongodb').MongoClient;

AMIS

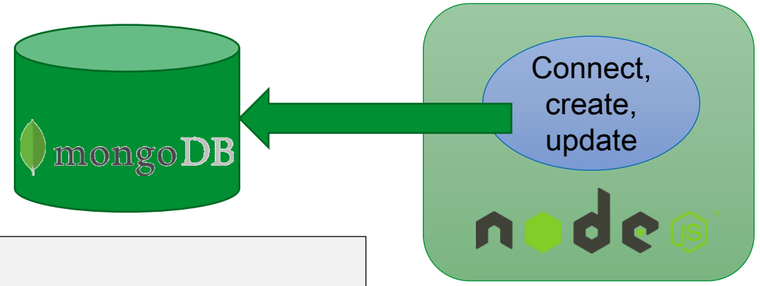
CONNECTING TO SERVER




```
var MongoClient = require('mongodb').MongoClient;

var mongodbHost = '127.0.0.1';
var mongodbPort = '27017';
var mongodbDatabase = 'world';

var url = 'mongodb://'+mongodbHost+':'+mongodbPort + '/' + mongodbDatabase;

MongoClient.connect(url, function(err, db) {
   console.log("Connected correctly to server.");

   // DO YOUR THING WITH MONGODB

   db.close();
   console.log("Connection to database is closed.");
}) //connect()
```

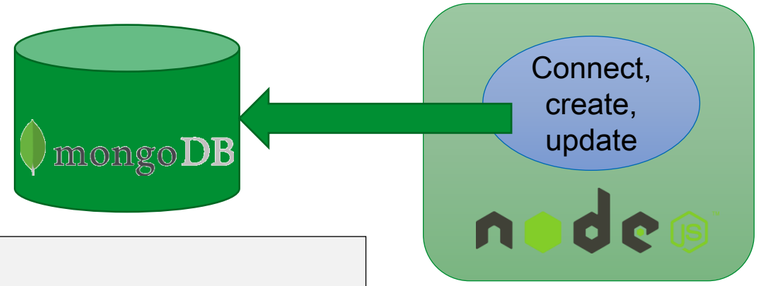
AMIS

RETRIEVE DATA (1A)



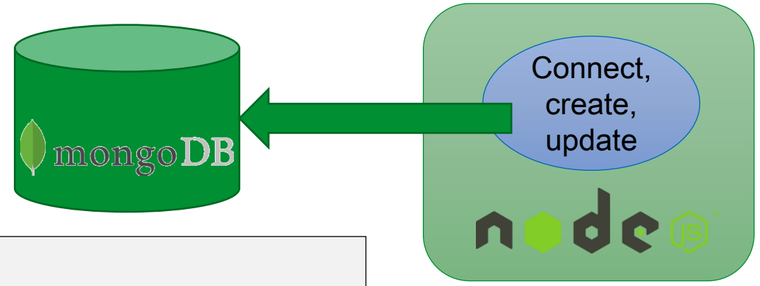

```
var MongoClient = require('mongodb').MongoClient;
var mongodbHost = '127.0.0.1';
var mongodbPort = '27017';
var mongodbDatabase = 'world';
var url = 'mongodb://'+mongodbHost+':'+mongodbPort + '/' + mongodbDatabase;

MongoClient.connect(url, function(err, db) {
   console.log("Connected correctly to server.");
   db.collection('countries').find({},{"sort": [["area",-1]]})
                             .limit(20).toArray(function(err, results){
     console.log("Name of Country Four " +results[3].name
                 + " and size: " +results[3].area);
     …
   }
```

RETRIEVE DATA (1B)



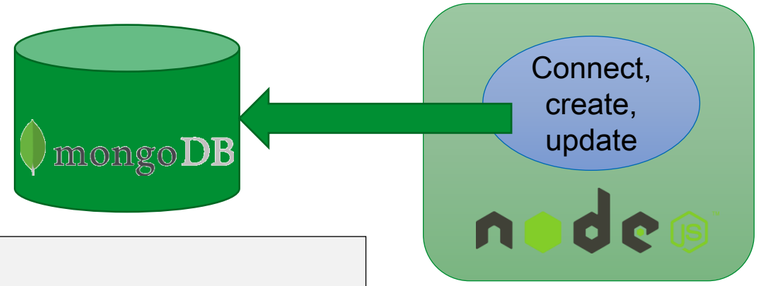

```
var MongoClient = require('mongodb').MongoClient;
var mongodbHost = '127.0.0.1';
var mongodbPort = '27017';
var mongodbDatabase = 'world';
var url = 'mongodb://'+mongodbHost+':'+mongodbPort + '/' + mongodbDatabase;

MongoClient.connect(url, function(err, db) {
   console.log("Connected correctly to server.");
   // using cursors to retrieve data sets in a controlled fashion
   // note: cursor implements NodeJS Stream  - results can be piped
   var cursor = db.collection('countries').find({"continent":"Asia"}, {"sort": "name"});

   cursor.count(function(err, count){
      console.log("Country count in Asia: "+count);
   });

   cursor.each(function(err, country){
     console.log(country.name);
   })//each cursor
```

RETRIEVE DATA (2)



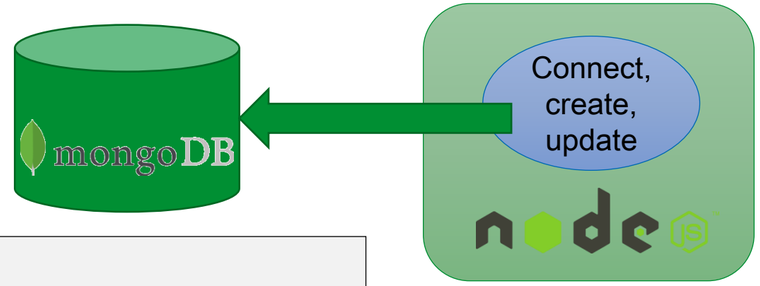

```
var aggquery = [  {$sort: {area : -1}}
                , {$group:{ _id: '$continent'
                           , largestCountry : {$first: "$name"}
                           }
                  }
    ];
    var aggcursor = db.collection('countries').aggregate(aggquery);
    aggcursor.each(function(err, result){
      if (err) {
         console.log(err);
      } else if (result)
         console.log(JSON.stringify(result));
    })  //each aggcursor


var ccursor = db.collection('countries').find({});
// the cursor returned from find and aggregate implements a NodeJS (readable) Stream
ccursor.on('data', function(doc) {
  console.log(doc);
});
ccursor.once('end', function() {
  console.log("Out of countries. Time to move on");
});
```

CONNECTING TO SERVER – ECMA 6



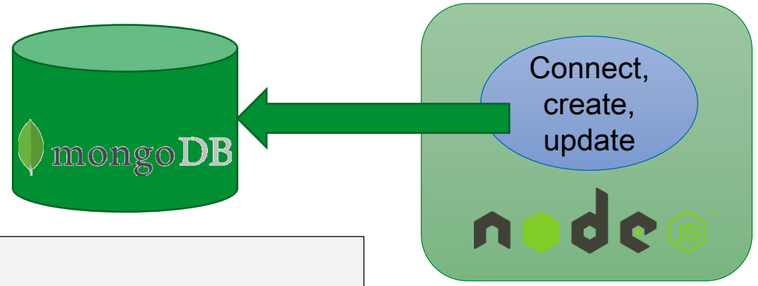

```
var MongoClient = require('mongodb').MongoClient;
var assert = require('assert'),   co = require('co');

var mongodbHost = '127.0.0.1';
var mongodbPort = '27017';
var mongodbDatabase = 'world';
var url = 'mongodb://'+mongodbHost+':'+mongodbPort + '/' + mongodbDatabase;

co(function*() {
  // Use connect method to connect to the Server
  var db = yield MongoClient.connect(url);

  console.log("Connected correctly to server");
  // DO YOUR THING WITH MONGODB

  // Close the connection
  db.close();
}).catch(function(err) {
  console.log(err.stack);
});
```

RETRIEVE DOCUMENTS – ECMA 6



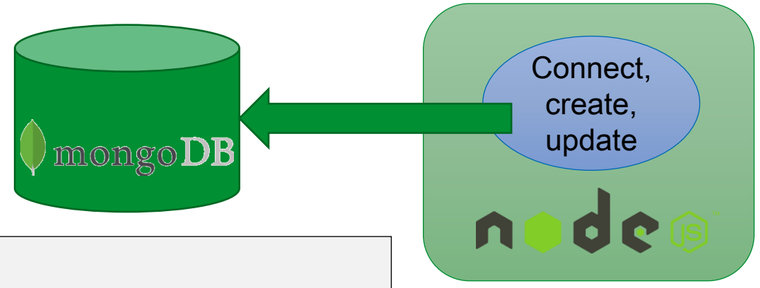

```
co(function*() {
  // Use connect method to connect to the Server
  var db = yield MongoClient.connect(url);

  console.log("Connected correctly to server");
  // find top 20 countries by  size
  var results = yield db.collection('countries').find({}
                          ,{"sort": [["area",-1]]}).limit(20).toArray();
  console.log("Country One " +JSON.stringify(results[0]));
  console.log("Name of Country Four " +results[3].name
                                      + " and size: " +results[3].area);
  // use cursor to get the country count
  var cursor = db.collection('countries').find({"continent":"Asia"}
                                                , {"sort": "name"});
  var count = yield cursor.count();
  console.log("Country count in Asia: "+count);

  while (yield cursor.hasNext()){
    var country = yield cursor.next();
    console.log(country.name);
  }
```

RETRIEVE DOCUMENTS (2) – ECMA 6



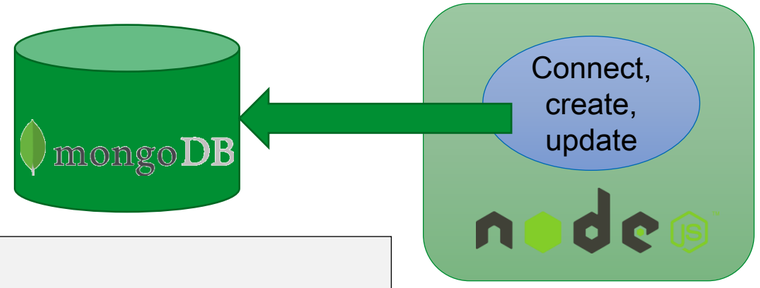

```
// the largest country per continent
    var aggquery = [  {$sort: {area : -1}}
                    ,  {$group:{ _id: '$continent'
                                , largestCountry : {$first: "$name"}
                                }
                       }
     ];
     var aggcursor = db.collection('countries').aggregate(aggquery);
  while (yield aggcursor.hasNext()){
       var result = yield aggcursor.next();
          console.log(JSON.stringify(result));
    }
  // Close the connection
  db.close();
}).catch(function(err) {
  console.log(err.stack);
});//co
```

CREATE DOCS IN MONGODB – ES6



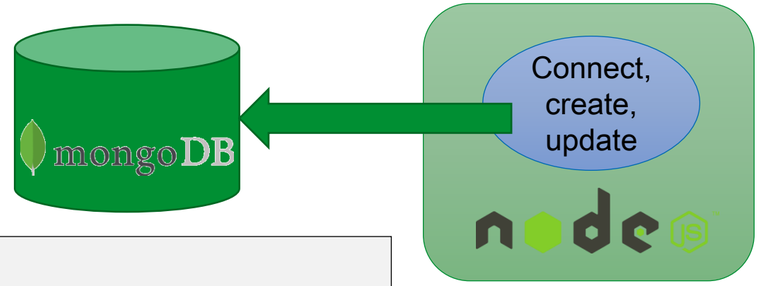

```
// define two documents, any structure you like
  var doc = {
       "continent" : "Europe",
        "children" : [ {"name":"Belgium"}, {"name":"Luxemburg"}],
        "someVariable" : 19123, "andmore" : "2716NK" };
  var doc2 = {
       "continent" : "Asia", "name" : "Johnny",
        "nrs" : [ {"name":"China"}, {"name":"India"}, {"name":"Buthan"}],
       "tree": {"branch": {"twig":{"leaf": 1}}}
     };
  var nameOfCollection = "myDocs"
  var result = yield db.collection(nameOfCollection).insertMany([doc,doc2]);
  console.log(">> "+result.insertedCount
                  +" documents created into collection "+nameOfCollection);
  ...
```

CREATE & UPDATE DOCS – ES6




```
// define two documents, any structure you like
   var doc = {...}, doc2 = {...} , nameOfCollection = "myDocs“;
   var result = yield db.collection(nameOfCollection).insertMany([doc,doc2]);
   console.log(">> "+result.insertedCount +" documents created);
   var cursor = db.collection(nameOfCollection).find();
   while (yield cursor.hasNext()){
       var doc  = yield cursor.next();
       console.log("Document: " +JSON.stringify(doc));
    }// while cursor
   result = yield db.collection(nameOfCollection).updateOne(
    {"tree.branch.twig.leaf":1}, {$set: {name: "Hank", city:"Nieuwegein“
                                  , "location.province":"Utrecht“
                                  , "location.country":"The Netherlands“
                                  , "tree.stem":5}});
    console.log(">> Updated "+result.modifiedCount+" document(s)");
    cursor = db.collection(nameOfCollection).find();
    while (yield cursor.hasNext()){
       var doc  = yield cursor.next();
       console.log("Document: " +JSON.stringify(doc));
    }// while cursor
```

CREATE, UPDATE AND DELETE – ES6




```
// define two documents, any structure you like
var doc = {...}, doc2 = {...} , nameOfCollection = "myDocs“;
var result = yield db.collection(nameOfCollection).insertMany([doc,doc2]);
console.log(">> "+result.insertedCount +" documents created);
var cursor = db.collection(nameOfCollection).find();
while (yield cursor.hasNext()){
    var doc  = yield cursor.next();
    console.log("Document: " +JSON.stringify(doc));
 }// while cursor
result = yield db.collection(nameOfCollection).updateOne(...);
...
result = yield db.collection(nameOfCollection).deleteMany({});
console.log(">> Deleted "+ result.deletedCount+" documents ");
// execute command to drop the collection
yield db.command({drop:nameOfCollection});
console.log(">> Dropped collection "+nameOfCollection);
// Close the connection
db.close();
console.log("Connection to database is closed.");
```

GETTING ACCESS TO A MONGODB INSTANCE

- Install locally
  - 150 MB download, simple install wizard (< 5 minutes)
- Compose a Virtual Machine with MongoDB
- Use Docker Container for MongoDB
  - https://hub.docker.com/r/_/mongo/
- Install MongoDB on Compute Cloud Service
  - Azure, AWS, Google Cloud Platform, Oracle Public Cloud, …
- Leverage MongoDB managed instance (PaaS service)
  - mLab (https://mlab.com), MongoDB Atlas (https://cloud.mongodb.com)
  - Starting with free tier

AMIS

ORACLE AND NODE

- Node for Application Container Cloud Service
- Node for implementing APIs in Mobile Cloud Service
- Node as execution language for Oracle Functions Cloud Service
- Node as deployment platform for Oracle JET applications

- Oracle Database driver for Node.js

AMIS

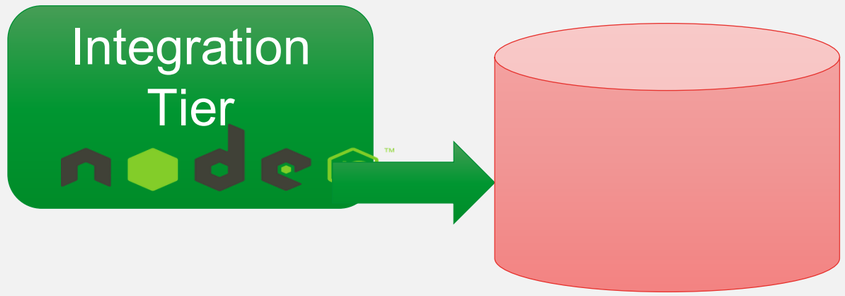
NODE-ORACLEDB DATABASE DRIVER

- The node-oracledb driver connects to Oracle Database for fast and functional applications. It is an open source project with Apache 2.0 license.
- It is maintained as an NPM package by Oracle and is under active development.
- https://github.com/oracle/node-oracledb or npm install node-oracledb
- Note: the Oracle Application Container Cloud preloads this driver to any instance of a Node.js container – ready to connect to DBaaS or on premises database
- Support for SQL and PL/SQL, Transaction Management, CLOBs and BLOBs, Ref Cursors, Types and Collections, authentication, …
  - Leveraging OCI Oracle (Fat) Client Libraries

CONNECT TO ORACLE DATABASE FROM NODE APPLICATION …

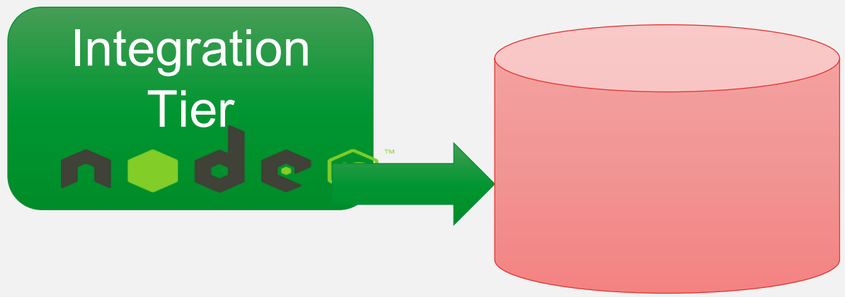
```
var oracledb = require('oracledb');
var dbConfig = require('./dbconfig.js');



oracledb.getConnection(
  {
    user          : dbC
    password      : dbConfig.password,
    connectString : dbConfig.connectString
  },
  function(err, connection)
  {
    if (err) {
      console.error(err.message);
      return;
    }
    connection.execute(
...
```

```
module.exports = {
    user          : process.env.NODE_ORACLEDB_USER || "hr",
    password      : process.env.NODE_ORACLEDB_PASSWORD || "oracle",
    connectString : process.env.NODE_ORACLEDB_CONNECTIONSTRING || "127.0.0.1:1521/orcl
    // Setting externalAuth is optional.  It defaults to false.  See:
    // https://github.com/oracle/node-oracledb/blob/master/doc/api.md#extauth
    externalAuth  : process.env.NODE_ORACLEDB_EXTERNALAUTH ? true : false
};
```



… AND PERFORM SQL OR PL/SQL

```
...
 connection.execute(
      "SELECT department_id, department_name " +
        "FROM departments " +
        "WHERE department_id = :did",
      [180],
      function(err, result)
      {
        if (err) {
          console.error(err.message);
          doRelease(connection);
          return;
        }
        console.log(result.metaData);
        console.log(result.rows);
        doRelease(connection);
      });
  });

function doRelease(connection)
{ connection.release( function(err) {…});}
```



YOUR NEXT STEPS WITH NODE … AND BEYOND

- API Development
- Explore npm packages – 10Ks of them
  - For SOAP, XML, IMDB, Dictionary, Geography, Text-to-Speech, Loopback mBaaS …
- Advanced topics: Web Sockets , security, scaling, H/A, state in a stateless world, queuing and scheduling ‘background jobs’
- Creating your own modules and NPM packages
  - Contribute to existing packages
- IDEs – Visual Studio Code, WebStorm, NetBeans
- Debugging
- Testing
- Deployment
- Microservices
- Rethink mBaaS & iPaaS

AMIS

HANDSON – PART 2

Part 5 – REST API (Spotify)

Part 6 – Send Email, Collect User Input from Command Line, Server Push, Promises

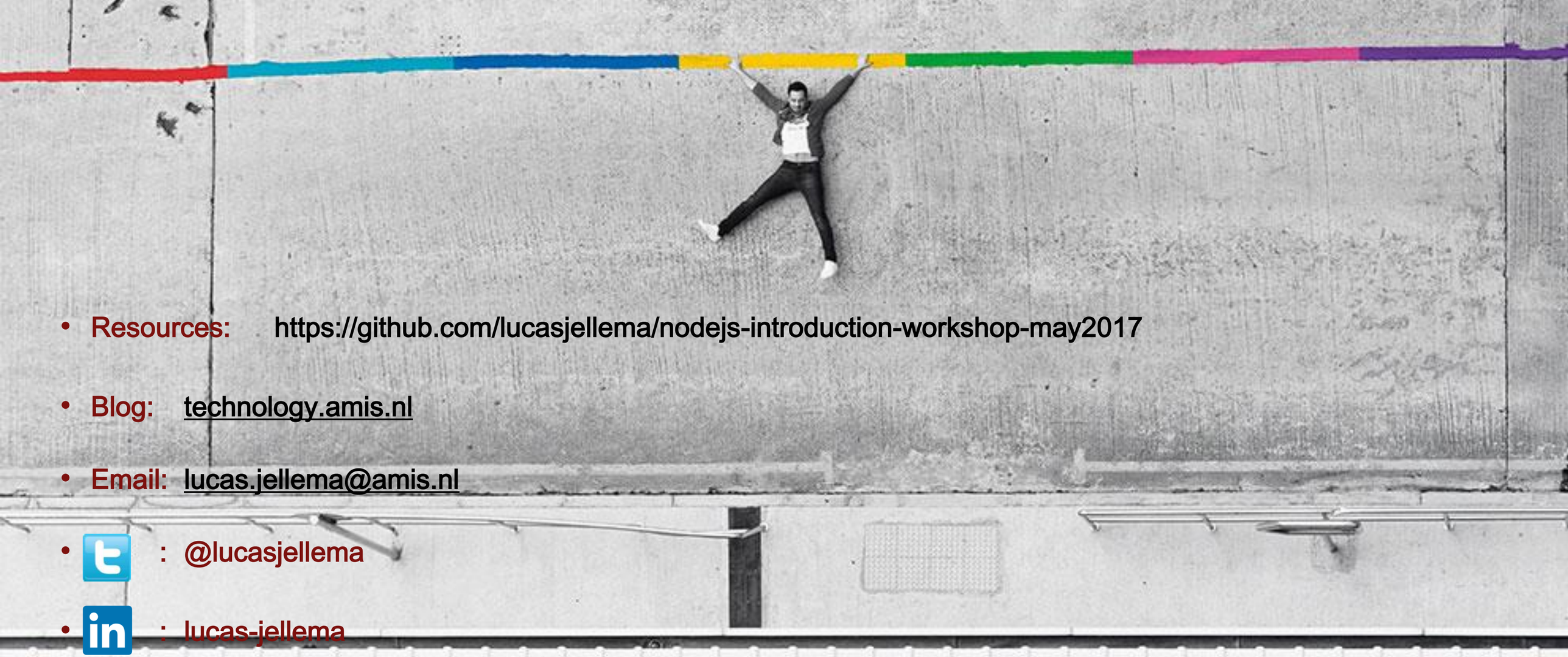
Part 7 – Interaction with MongoDB (MEaN stack)

Part 8 – Node in Docker

Part 9 – Node and Oracle Database

End to End Assignment: Artist Portal (Express, SSE, APIs, MongoDB)

AMIS



- Resources: https://github.com/lucasjellema/nodejs-introduction-workshop-may2017
- Blog: technology.amis.nl
- Email: lucas.jellema@amis.nl
- : @lucasjellema
- : lucas-jellema
- : www.amis.nl, info@amis.nl
  +31 306016000
  Edisonbaan 15,
  Nieuwegein