FROM ZERO TO 80

YOUR FIRST NODE.JS WEB APPLICATION

Andrew Theken

Andrew Theken

@atheken

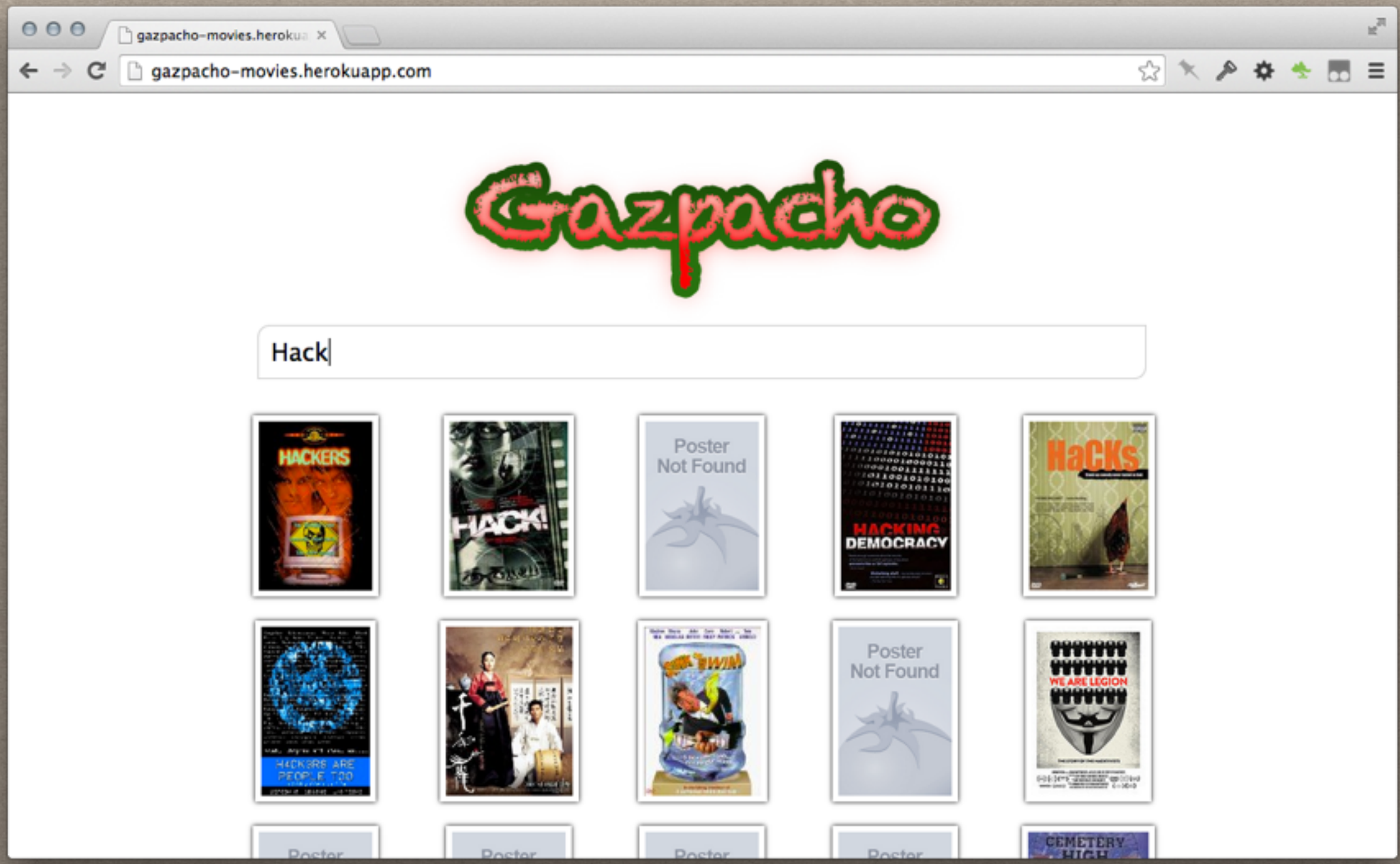
andrewtheken.com

Curtis Herbert

WHAT WE'LL COVER

- What's Node?
- The reasons to use (or not use) Node.js
- Getting Node.js installed, and getting comfortable in in the environment.
- Modules
- Events
- Streams
- Creating an HTTP server
- Middleware
- Node.js "express"
- HTML templating using Jade.

WHAT WE'LL BUILD



WHAT'S NODE.JS?

Node.js is a platform built on Chrome's JavaScript runtime for easily building fast, scalable network applications. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient, perfect for data-intensive real-time applications that run across distributed devices.

- nodejs.org

WHY SHOULD YOU USE NODE.JS?

- One language everywhere: Javascript.
- Tons of modules available via NPM.
- Easy to deploy/run
- More than just HTTP Servers: CLI-apps, networked services, embedded systems, desktop GUI apps (really!)
- Active, fantastic community.
- It’s Fun!

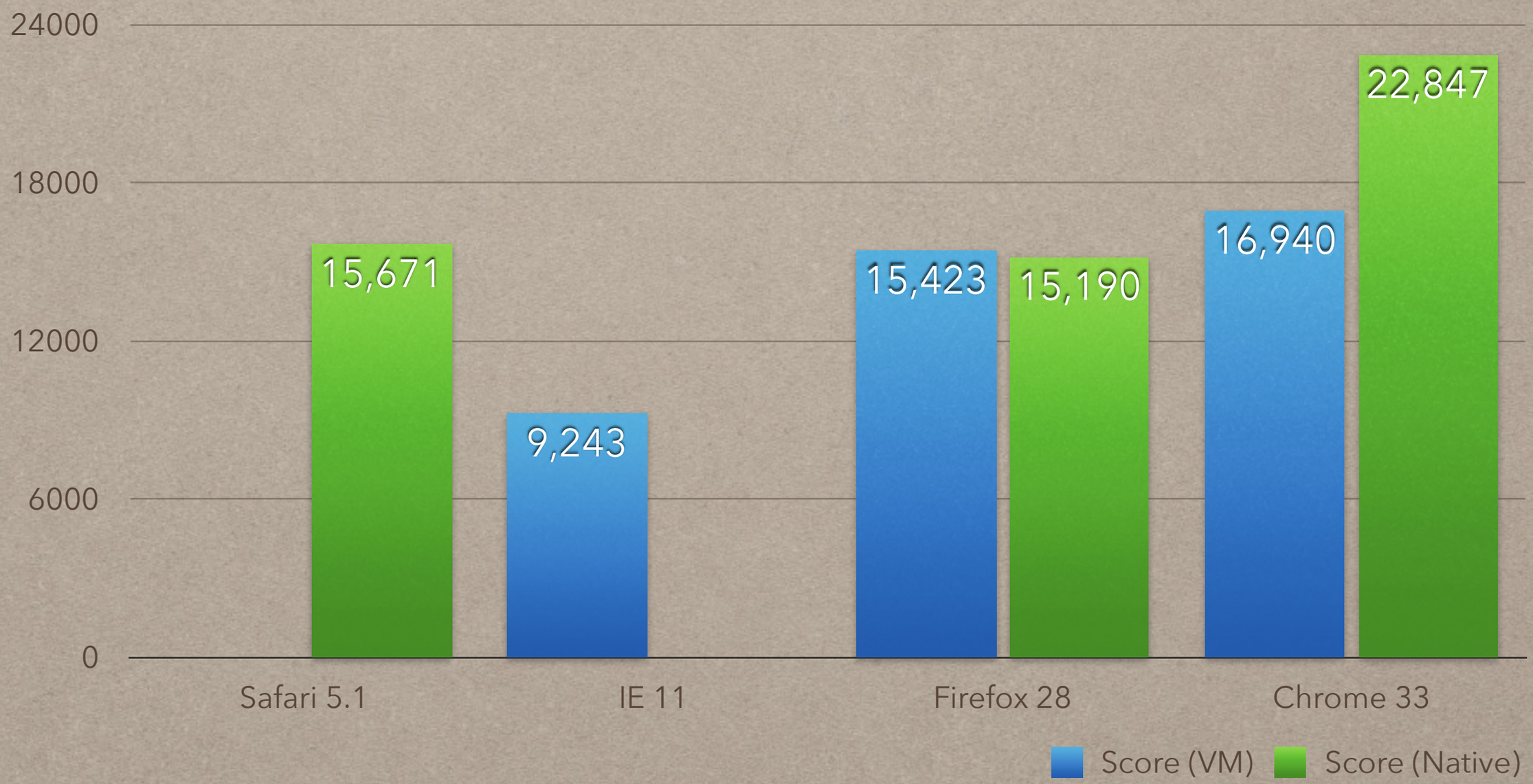
WHY SHOULDN'T YOU USE NODE.JS?

- Your app is CPU-intensive (graphics processing, for example)*
- Your app needs high-precision math capabilities.
- You expect to have a large-scale codebase (50kLoC+). This can be done, but requires practice and discipline.

** One could farm out CPU intensive work to a C++ process; "Think async."*

WHY V8?

Chrome's V8 Javascript Engine is *FAST*.



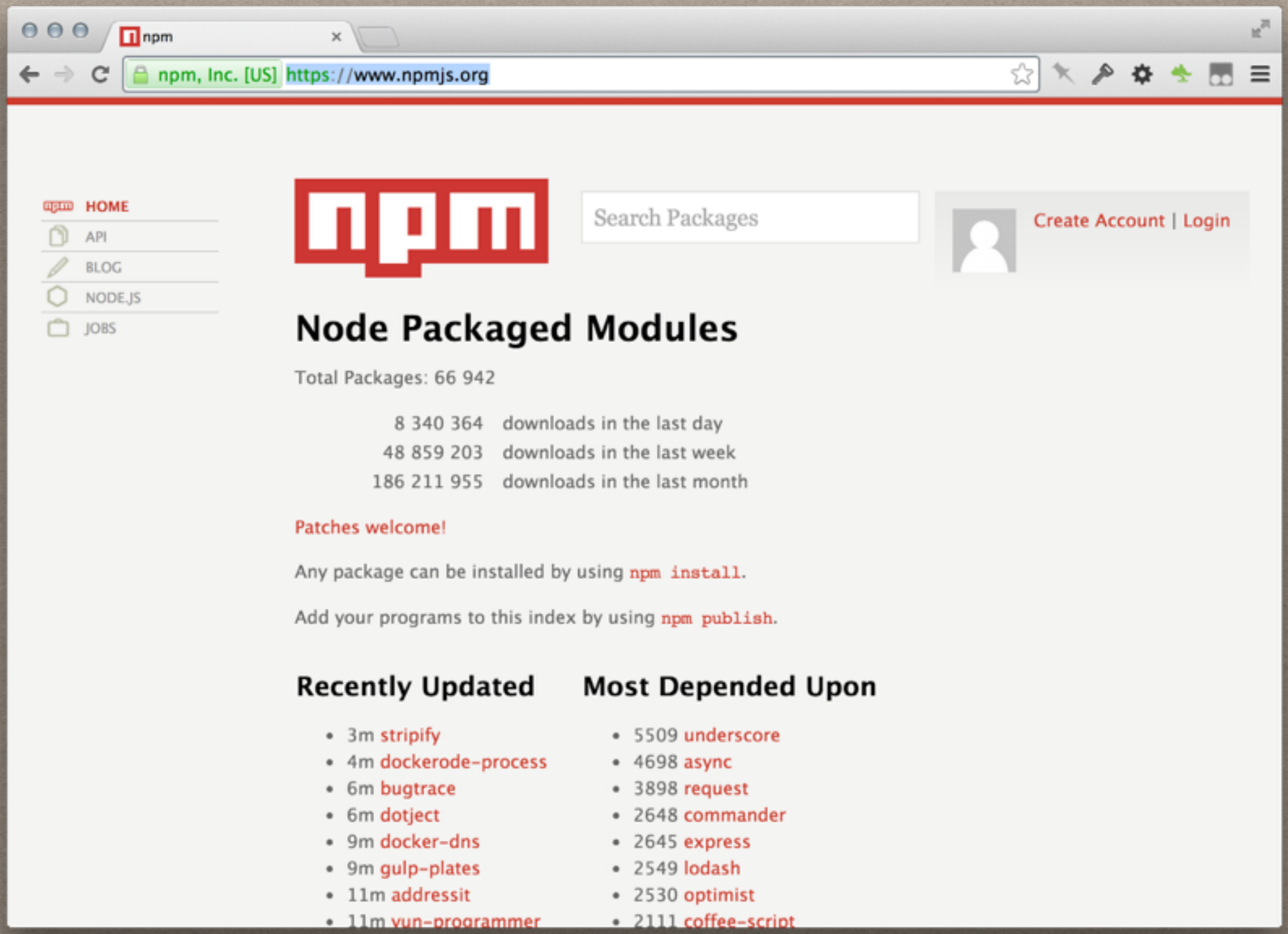
octane-benchmark.googlecode.com/svn/latest/index.html

Exercise 1: Getting to know node

- Install Node.js
- Run the Node.js Prompt
- Print “hello world”
- Create your first node script.
- Explore the “globals” available in the node console.

http://bit.ly/1mRtVHu

NPM *http://npmjs.org*



Exercise 2: Doing something useful(?)

```
➜  exercise2 git:(master) ✗ node colorize.js
The quick brown fox jumps over the lazy dog.
➜  exercise2 git:(master) ✗
```

- Search npmjs.org for “ansi-colorizer”
- Review docs on “ansi-colorizer”
  - install “ansi-colorizer” using NPM.
  - check out the “node_modules” directory
  - use “require” to load the ansi-colorizer
  - Colorize vowels in a string.

EVENT-DRIVEN?

React to changes rather than *"poll"* for them.

*Imperative (**Active**):*

```
while(true){
    if(isNewFileInDirectory(“./uploads”)){
        processFile();
    }
}
```

*Declarative (**Passive**):*

```
var fileWatcher = new FileWatcher(“./uploads”);

fileWatcher.on(‘newfile’, processFile);
```

NON-BLOCKING I/O

I/O Is **slow**. Like, really, ***really***, slow.

Reading 1Mb from traditional hard drive takes **60,000,000** CPU Cycles

Event	Latency	Scaled
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access (DRAM, from CPU)	120 ns	6 min
Solid-state disk I/O (flash memory)	50–150 µs	2–6 days
Rotational disk I/O	1–10 ms	1–12 months
Internet: San Francisco to New York	40 ms	4 years
Internet: San Francisco to United Kingdom	81 ms	8 years
Internet: San Francisco to Australia	183 ms	19 years
TCP packet retransmit	1–3 s	105–317 years
OS virtualization system reboot	4 s	423 years
SCSI command time-out	30 s	3 millennia
Hardware (HW) virtualization system reboot	40 s	4 millennia
Physical system reboot	5 m	32 millennia

Table 2.2 Example Time Scale of System Latencies

From: *Systems Performance: Enterprise and the Cloud, p. 20*

NON-BLOCKING I/O

Blocking I/O (**Synchronous, Active**):

```
console.log(‘Dictionary data loading. . .’);
//Open and read a 1MB file from disk.
var content = fileSystem.readFileSync(‘./dictionary.txt’);

//“1 year” later. . .
console.log(‘Dictionary loaded.’);
```

Non-Blocking I/O (**Asynchronous, Passive**):

```
//Send request for file.
fileSystem.readFile('./dictionary.txt', function(err, data){
    if(!err){
        console.log(‘Dictionary loaded.’);
    }
});

//CPU is ready for more work immediately.
console.log('Dictionary data loading...’);
```

SYNC VS. ASYNC

Synchronous

"I'll wait here while you get this done"

Asynchronous

"Give me a job, get in, get out."

HOW DOES "ASYNC" WORK?

JAVASCRIPT THREAD

```
var fs = require('fs');
function cb(err, data){...};
fs.readFile('file.txt', cb);
```

BACKGROUND WORKER THREADS (C++)

```
retrieveFile('file.txt');
```

HOW DOES "ASYNC" WORK?

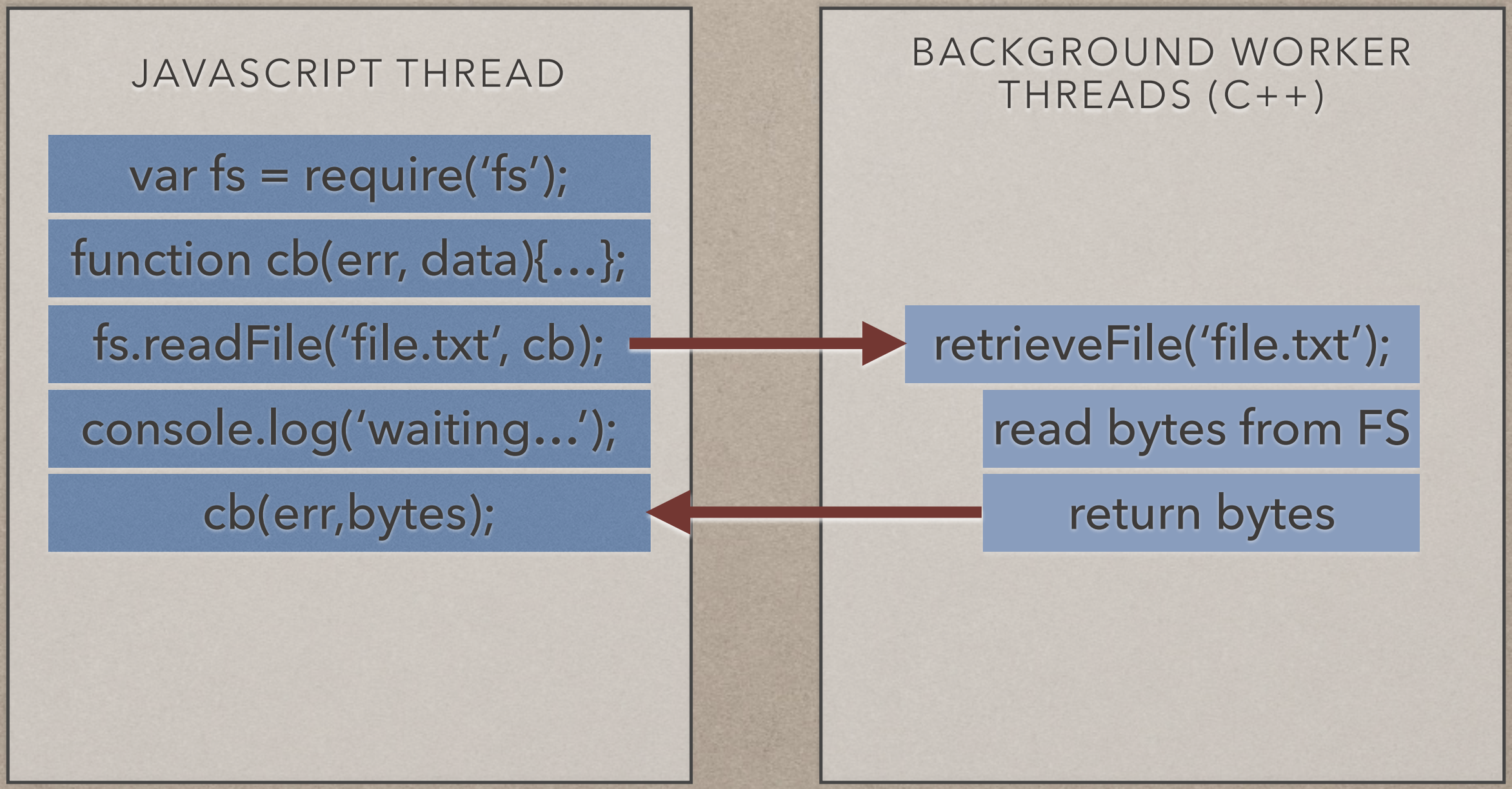
JAVASCRIPT THREAD

```
var fs = require('fs');
function cb(err, data){...};
fs.readFile('file.txt', cb);
console.log('waiting...');
```

BACKGROUND WORKER THREADS (C++)

```
retrieveFile('file.txt');
read bytes from FS
```

HOW DOES "ASYNC" WORK?



BIG DATA?

Congratulations! You work for wikipedia.

Your job is to find articles that could be enhanced by the inclusion of a cute cat pictures like the one on the right:

STREAMS

You decide that any article that mentions *cats, kittens, or felis* should include a kitten picture at the top.

There are **4,485,774** articles in the English version of Wikipedia. The full text of Wikipedia is **multiple terabytes.** **Loading all of these entries into memory at once is not an option.**

But, you're smart, and a node.js expert, so you recognize you can use ***streams*** to incrementally process the ginormous database of articles, one at a time.

Exercise 3: Data River

- require('fs'), check out the node docs for FileSystem
- Open utf-8 read file stream for "dictionary.txt", and listen for "data" events.
- Colorize vowels in the file and write them to process.stdout as data becomes available.

FOLLOWUP: KITTEN OVERLOAD

There have 1,000,000,000+ edits to Wikipedia.

How can we efficiently track & update new or modified articles that might benefit from kittenificationation?

That's it for the "core" node concepts.

Let's do something fun!

HTTP WITH NODE

- Node's web server is built-in!
- An HTTP "transaction" is an input *stream* (request), and an output *stream* (response):
  - Requests have headers, parameters, urls, bodies, and verbs.
  - Responses have headers, bodies, and statuses.

THE (ALMOST) SIMPLEST WEB SERVER POSSIBLE

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

// create a server respond to all requests
// with the current date.
var server = http.createServer(
    function(request, response){
        response.write(new Date().toString());
        response.end();
});

//start the server on port 4514
server.listen(4514);
```

Exercise 4: Time and Temperature (ok, just time)

- Create a basic web server.
- Respond to any request with the current date and time on port 4514
- Open http://localhost:4514/
- Enhance the server to respond with message of ‘Not Found. Sorry.’ for any url that isn’t to ‘/datetime’

COMMON HTTP SERVER TASKS

- Logging requests
- Serve “static” files
- Compress responses
- Store posted files
- Parse request parameters and cookies
- Authenticate/ Authorize Requests
- Send non-2xx responses (404,302, 500, etc.)
- Set cache headers
- Serve dynamic requests

Break for Lunch!

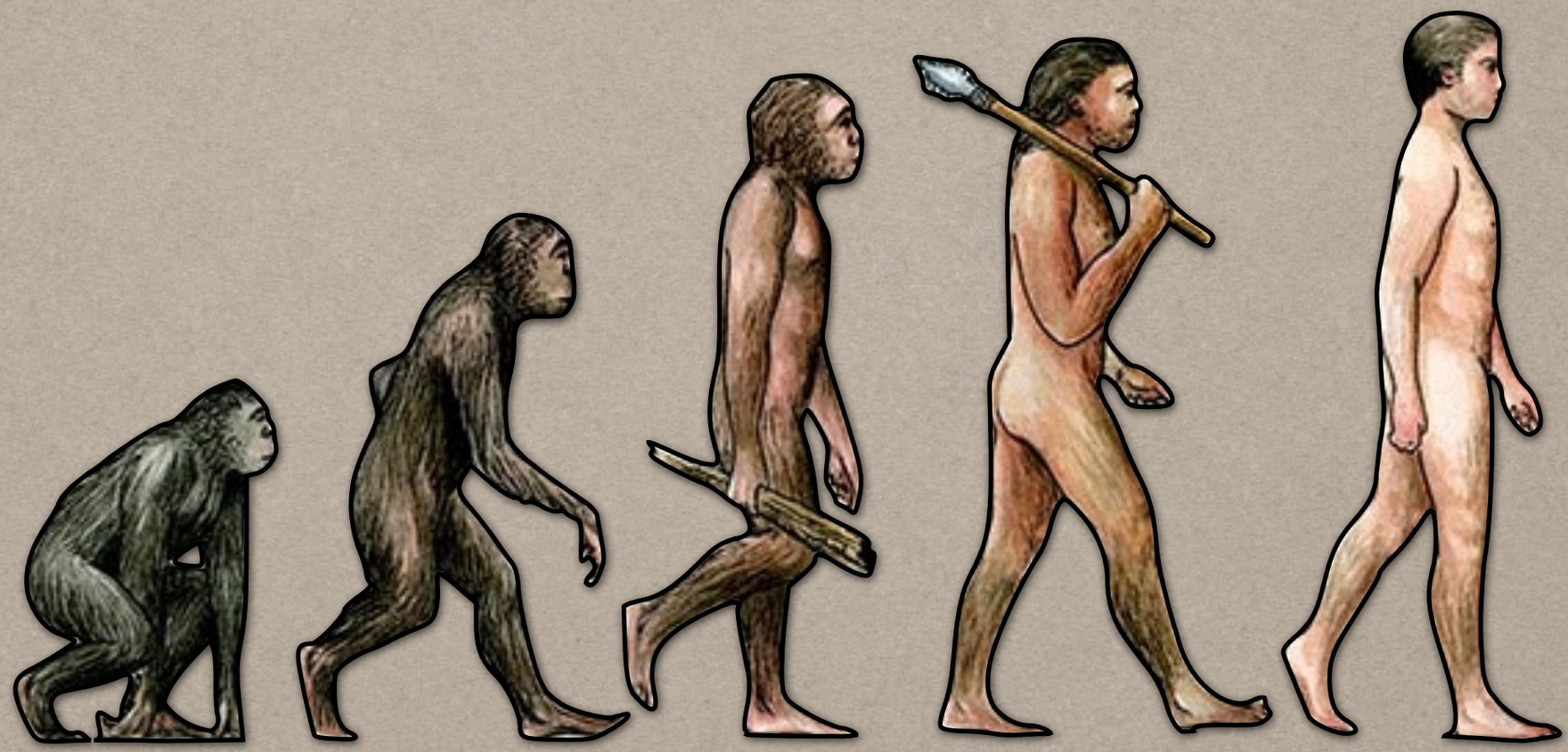
REVIEWING A SIMPLE HTTP SERVER

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

// create a server respond to all requests
// with the current date.
var server = http.createServer(
    function(request, response){
        if(request.url == '/datetime'){
            response.write(new Date().toString());
        }else{
            response.write('Not Found. Sorry.');
        }
        response.end();
});

//start the server on port 4514
server.listen(4514);
```

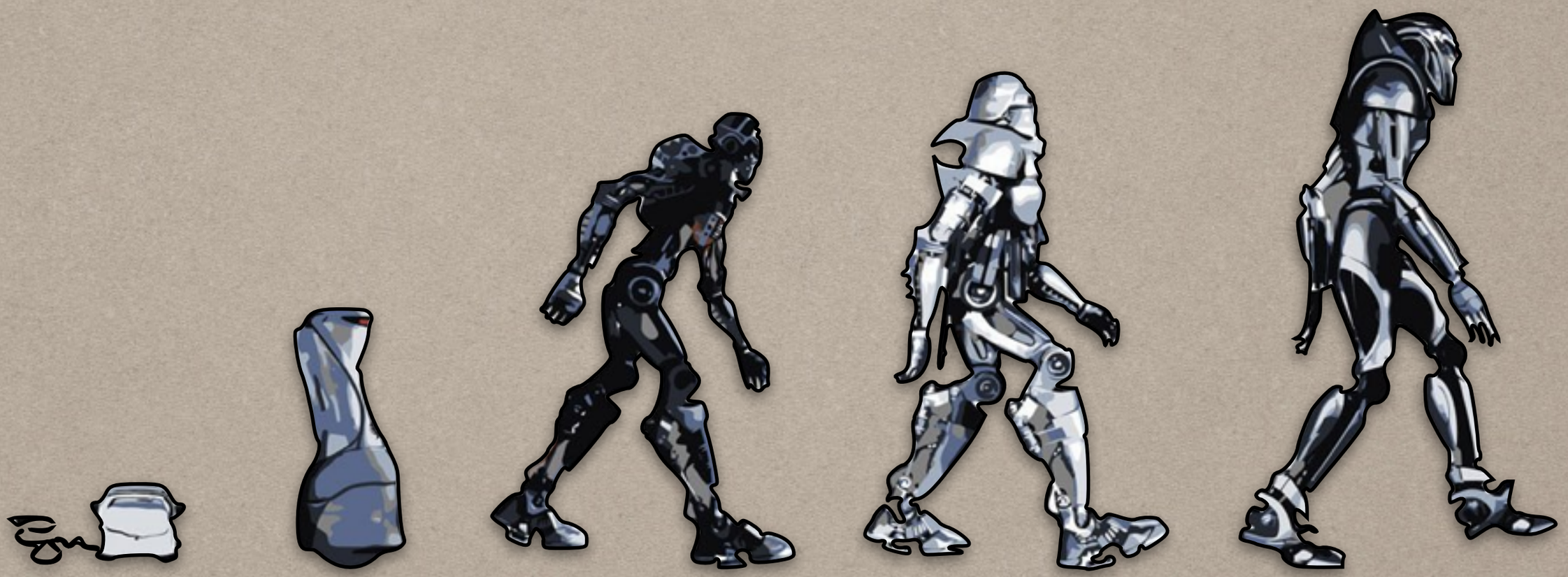
MIDDLEWARE



"Middleware is so simple,
you'd almost overlook explaining it."

- me, just now

MIDDLEWARE



"Middleware is so simple,
you'd almost overlook explaining it."

- me, just now

THE MIDDLEWARE FUNCTION

Middleware Prototype:

```
function(request, response, done){
    //inspect request
    //manipulate response
    //or, do nothing
}
```

THE MIDDLEWARE FUNCTION

Middleware Prototype:

```
function(request, response, done){
    //inspect request
    //manipulate response
    //or, do nothing
}
```

Log Request:

```
//A simple request logger.
function(request, response, done){
    console.log(request.path);
    done();
}
```

THE MIDDLEWARE FUNCTION

Log Request:

```
//A simple request logger.
function(request, response, done){
    console.log(request.path);
    done();
}
```

Send content:

```
//A simple date-time response
function(req, res, done){
    if(req.url === '/datetime'){
        res.write(new Date());
        res.end();
    }
    else{
        done();
    }
}
```

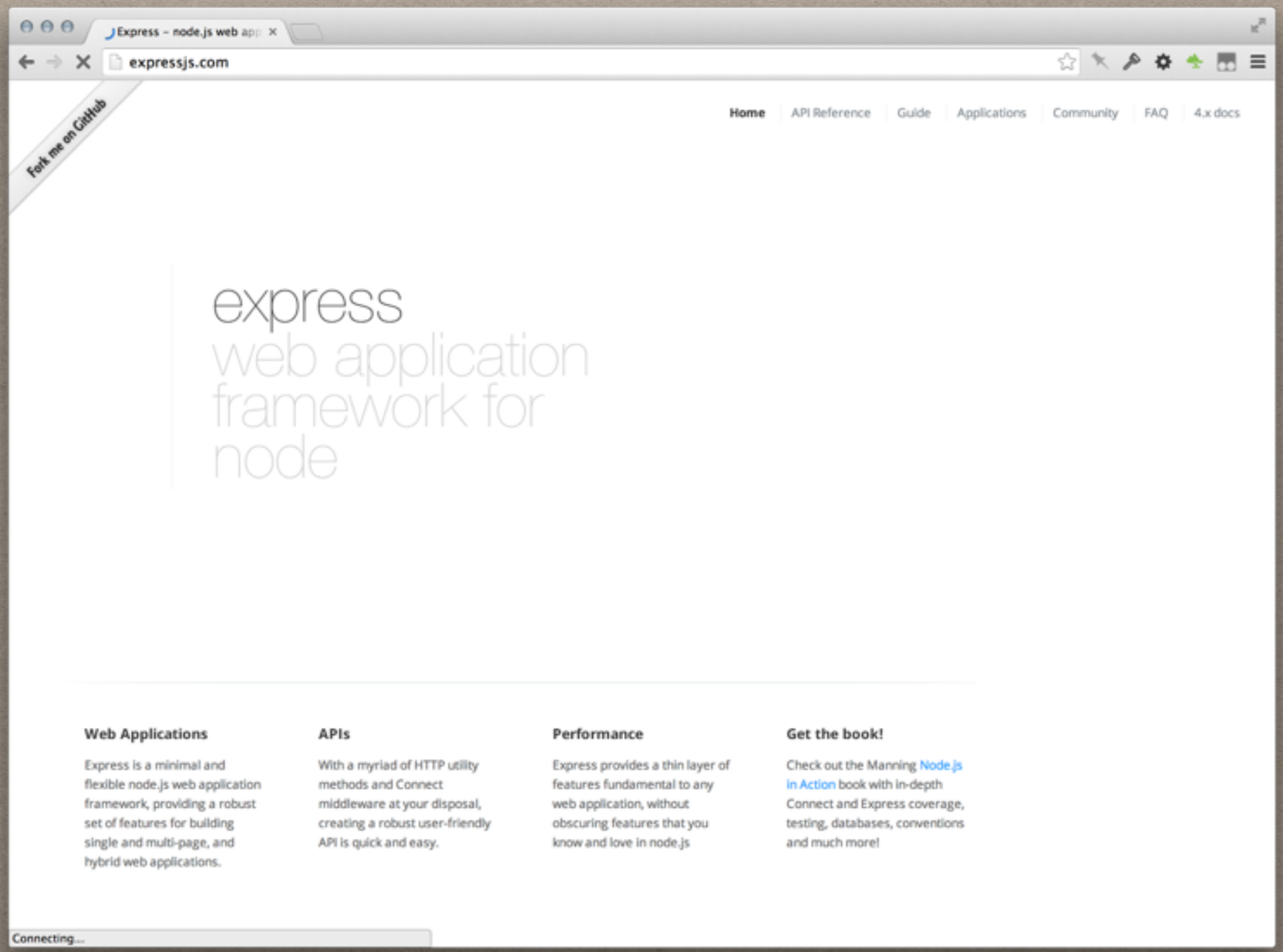
COMMON HTTP SERVER TASKS

- Logging requests
- Serve “static” files
- Compress responses
- Store posted files
- Parse request parameters and cookies
- Authenticate/ Authorize Requests
- Send non-2xx responses (404,302, 500, etc.)
- Set cache headers
- Serve dynamic requests

HTTP SERVER FRAMEWORKS

- Connect
- Express
- Restify
- Hapi
- Union

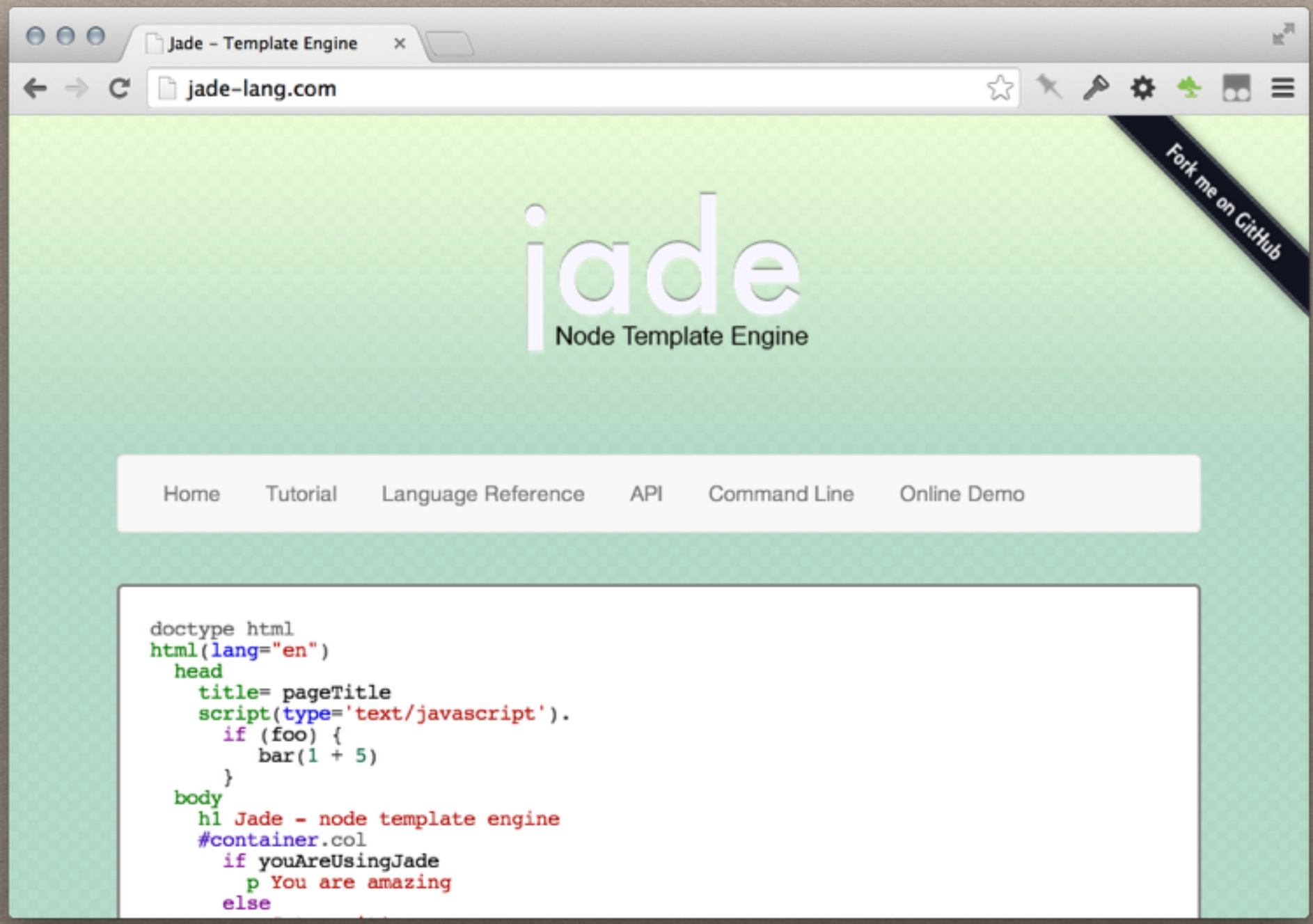
EXPRESS http://expressjs.com



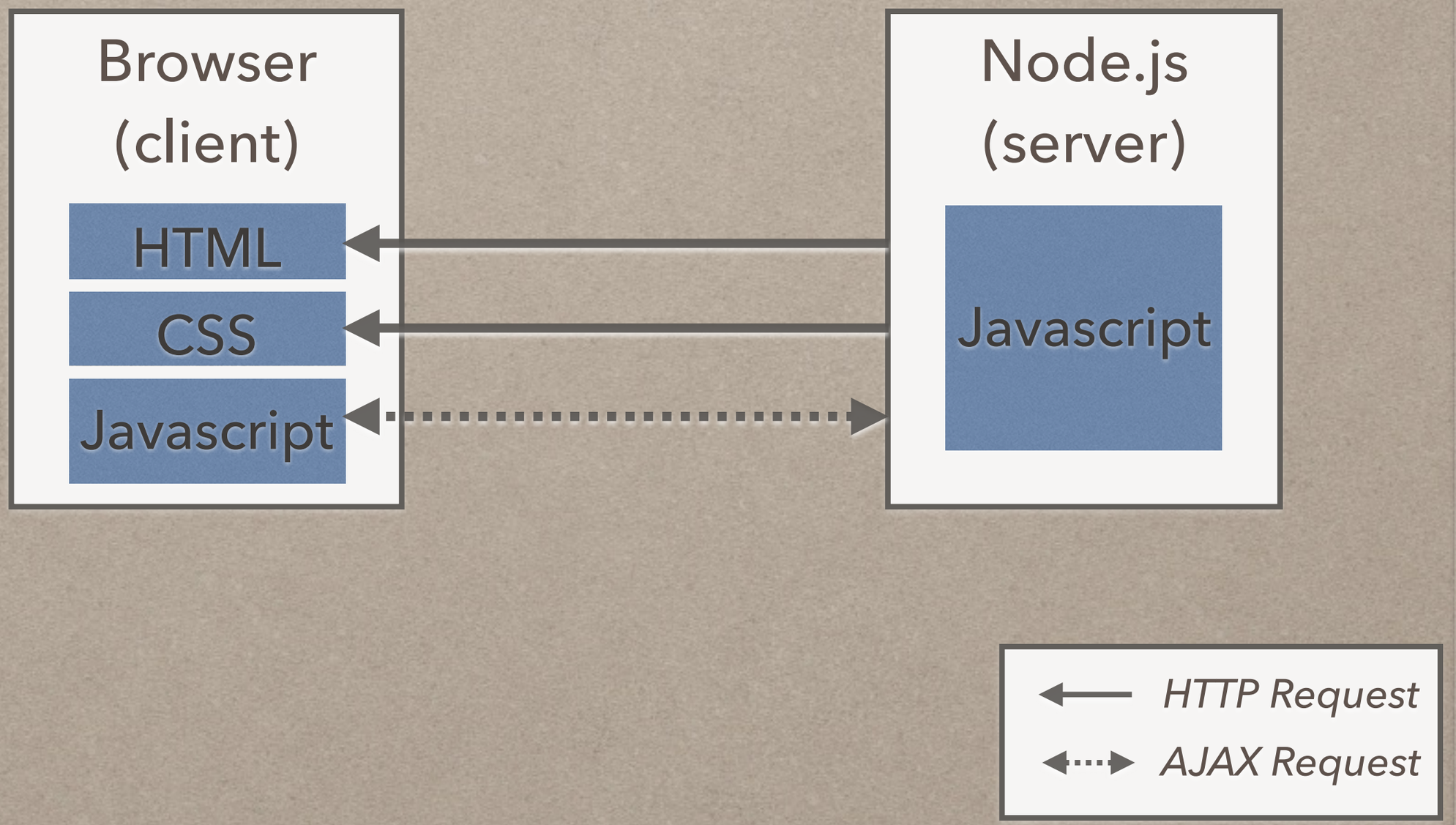
Exercise 5: All Aboard!

- `npm install express`
- Create file “server.js”
- Create express app, create a logging middleware.
- Create a “date time” middleware.
- Leverage express-provided middleware instead of our primitive middleware
- Serve static files from “public” folder.

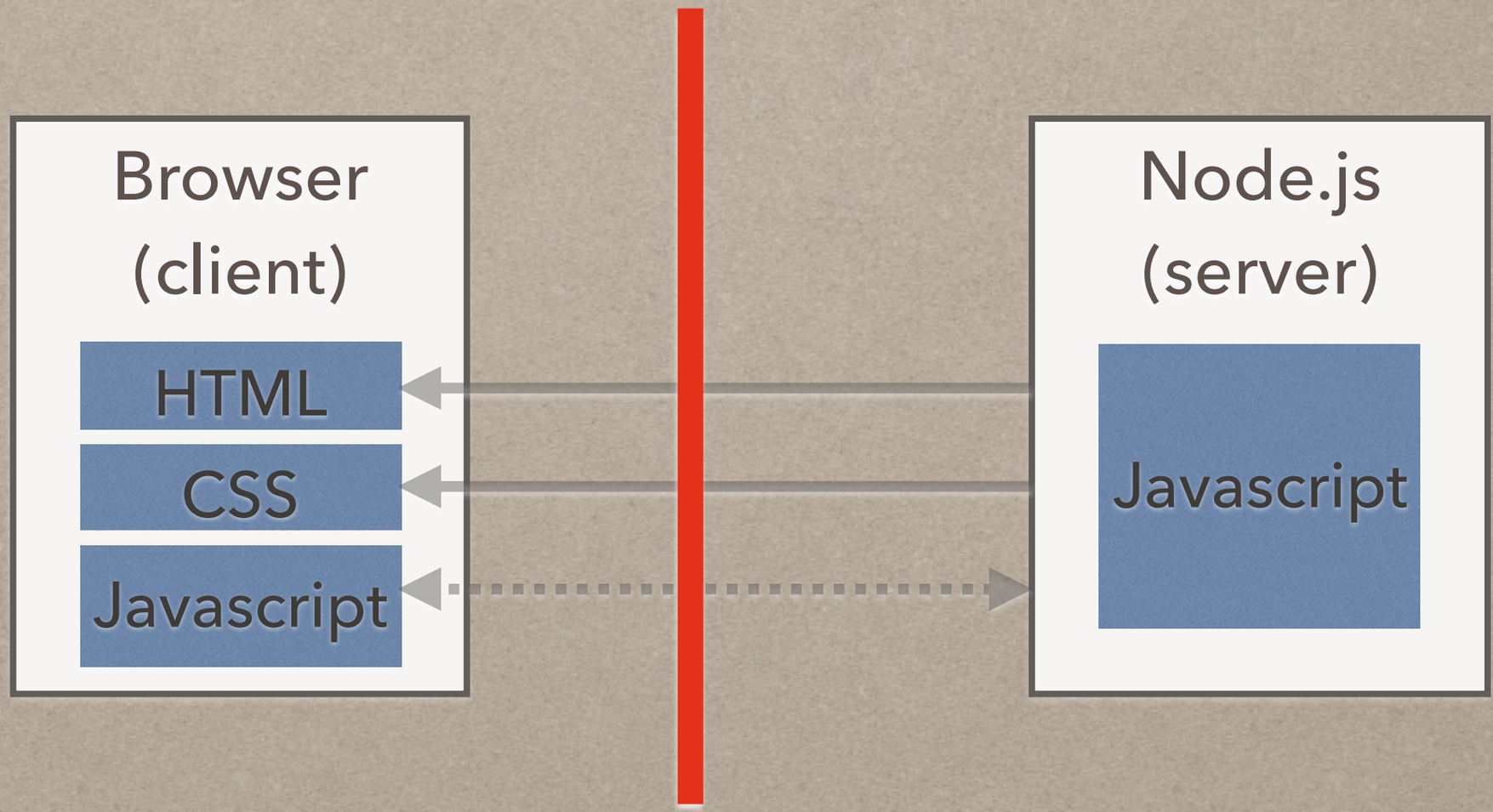
JADE http://jade-lang.com



A NOTE ABOUT CLIENT-SERVER COMMUNICATION



A NOTE ABOUT CLIENT-SERVER COMMUNICATION



The browser and Node.js
DO NOT share program state.

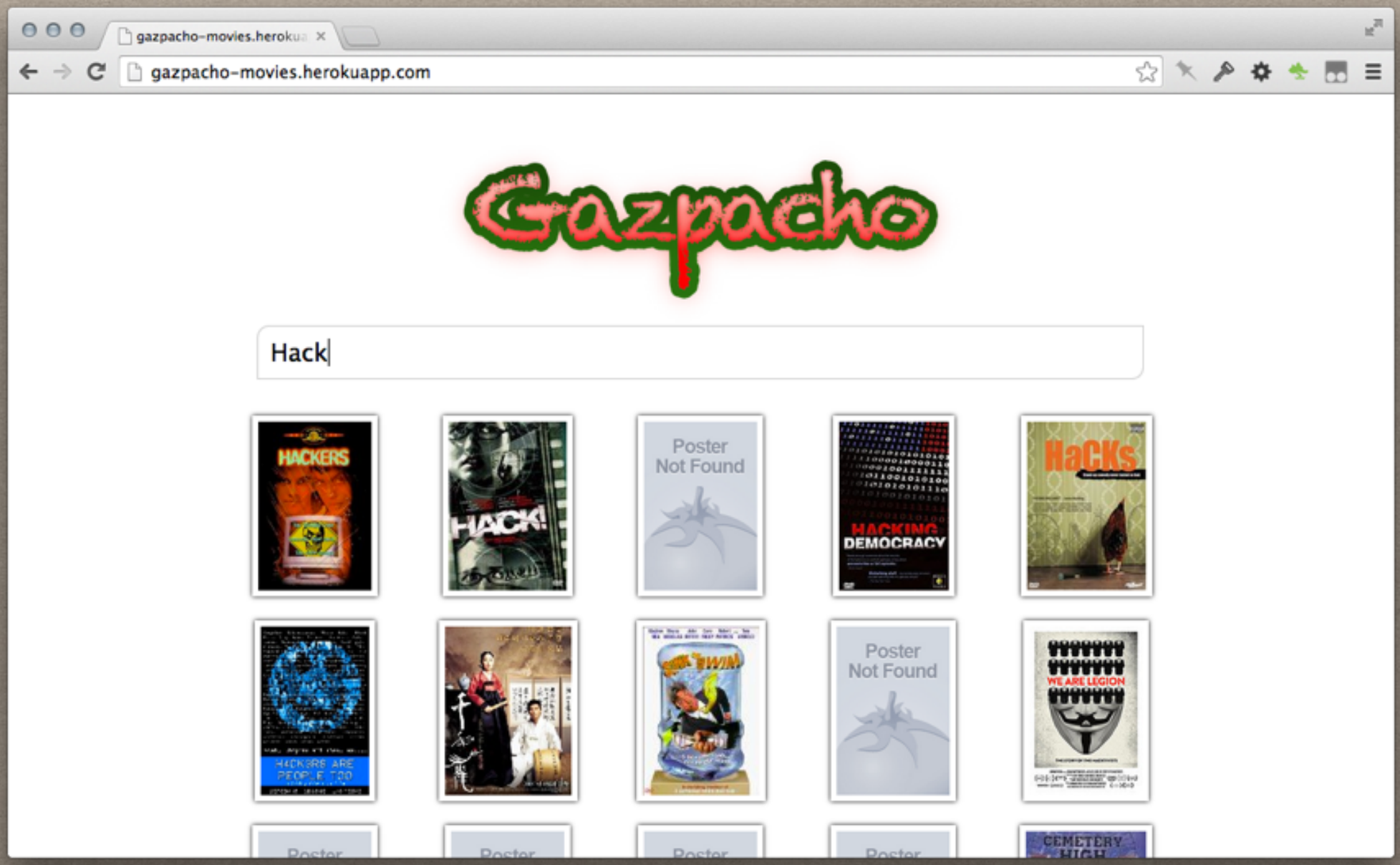
HTTP Request

AJAX Request

Exercise 6: I'm Jaded, and you will be, too.

- `npm install jade`
- Configure jade as a rendering engine.
- Create a *jade* view with a model.

GAZPACHO: A (PRETTY?) BROWSER FOR ROTTEN TOMATOES



Exercise 7: Glue it all together

- `npm install rotten-api`
- Add ‘/listmovies’ API middleware
- Add search box to index.jade
- On keyup, request movies from server.
- Render movie titles as list on client.
- BONUS: Create “favoriting” functionality.

MISCELLANEOUS

PERSISTENCE MODULES

- couchdb
- elasticsearch
- redis
- mongodb
- mysql
- riak
- postgres
- oracle
- voltdb
- aws
- azure
- parse
- csv
- sqlite
- memcached
- cassandra
- sql server

TOOLS

- node-inspector (GUI debugging interface)
- supervisor (Continuously reload node as files are changed)
- “async” and “Seq” (avoid “boomerang” code)
- Grunt (automate repetitive tasks)
- JSHint (Identify syntax errors quickly).
- Bower (easily integrate client-side libraries)
- Yeoman (scaffold new projects)
- Browserify (use *some* node modules in the browser)
- Jasmine (unit testing)
- git (Source control, very popular in node community)

DEPLOYING

- Heroku (free*)
- Nodjitsu
- Joyent
- Runnable
- c9.io
- Nitrous.io
- Azure
- Web Faction
- Digital Ocean
- Linode
- Rackspace

RESOURCES

- nodeschool.io
- Read *JavaScript: The Good Parts*
  - http://amzn.com/B0026OR2ZY
- Node Philly: http://node.ph/
- Node.JS Philly Meetup: http://www.meetup.com/nodejs-philly/

Andrew Theken

@atheken

andrewtheken.com

Curtis Herbert