COMP 431/531: Web Development

Lecture 15: Web Servers

Mack Joyner (mjoyner@rice.edu)

https://www.clear.rice.edu/comp431

Announcements & Reminders

- HW #4 (Draft Front-end) due Tue. Oct. 21st at 11:59pm

  classroom hw4 repo: https://classroom.github.com/a/hfLD_40C

- HW #5 (Front-end) due Thu. Nov. 6th at 11:59pm

  classroom hw5 repo: https://classroom.github.com/a/hIFk8cgA

In-Class Exercise 14: Unit Tests

1. Download: https://www.clear.rice.edu/comp431/sample/tictactoe/react/tictactoeSlice.test.js to src/features/game
2. Modify board using dispatch actions cause “X” to win game (remember new state is returned)
3. Test redux global state to verify that “X” won game and “O” lost game
4. Run command: >> *sudo npm run test*
5. If needed, download: https://www.clear.rice.edu/comp431/sample/tictactoe/angular/game.service.spec.ts (and game.service.ts) to src/app/
6. Run command: >> ng test
7. Fix failed test HttpClient errors by adding to TestBed.configureTestingModule: *providers: [provideHttpClient()]*
8. Create an empty board (array) in “should determine game winner” test in game.service.spec.ts
9. Modify board to cause “X” to win game
10. Test board service wonGame function to verify that “X” won game and “O” lost game
11. Run code coverage for unit tests (optional)

Submit game.service.spec.ts and tictactoeSlice.test.js to IC 14

Back End Development

Part II

Web Servers

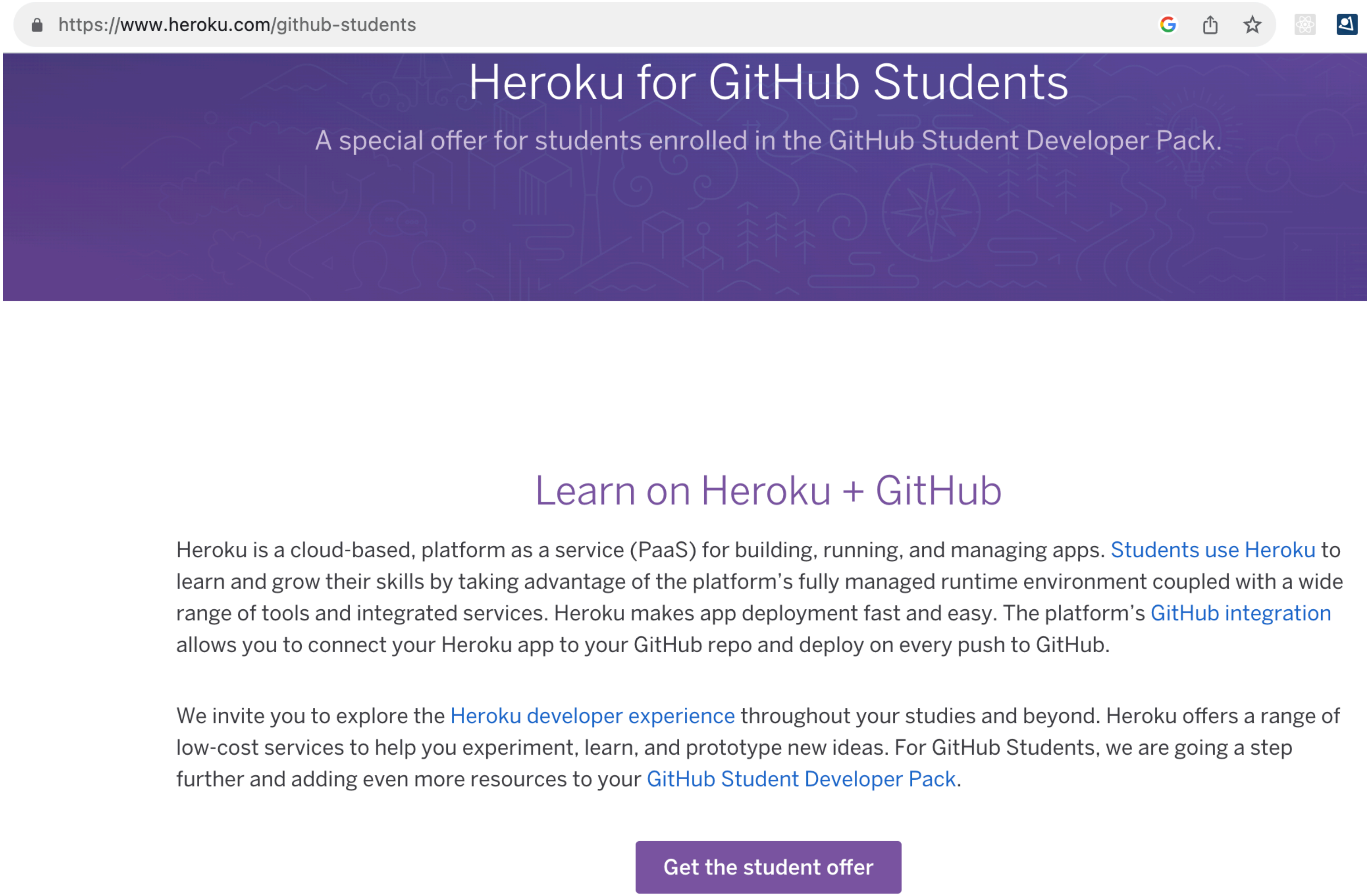
Backend

Architecture

Testing

Web Hosting

Databases



https://www.heroku.com/github-students

Heroku for GitHub Students

A special offer for students enrolled in the GitHub Student Developer Pack.

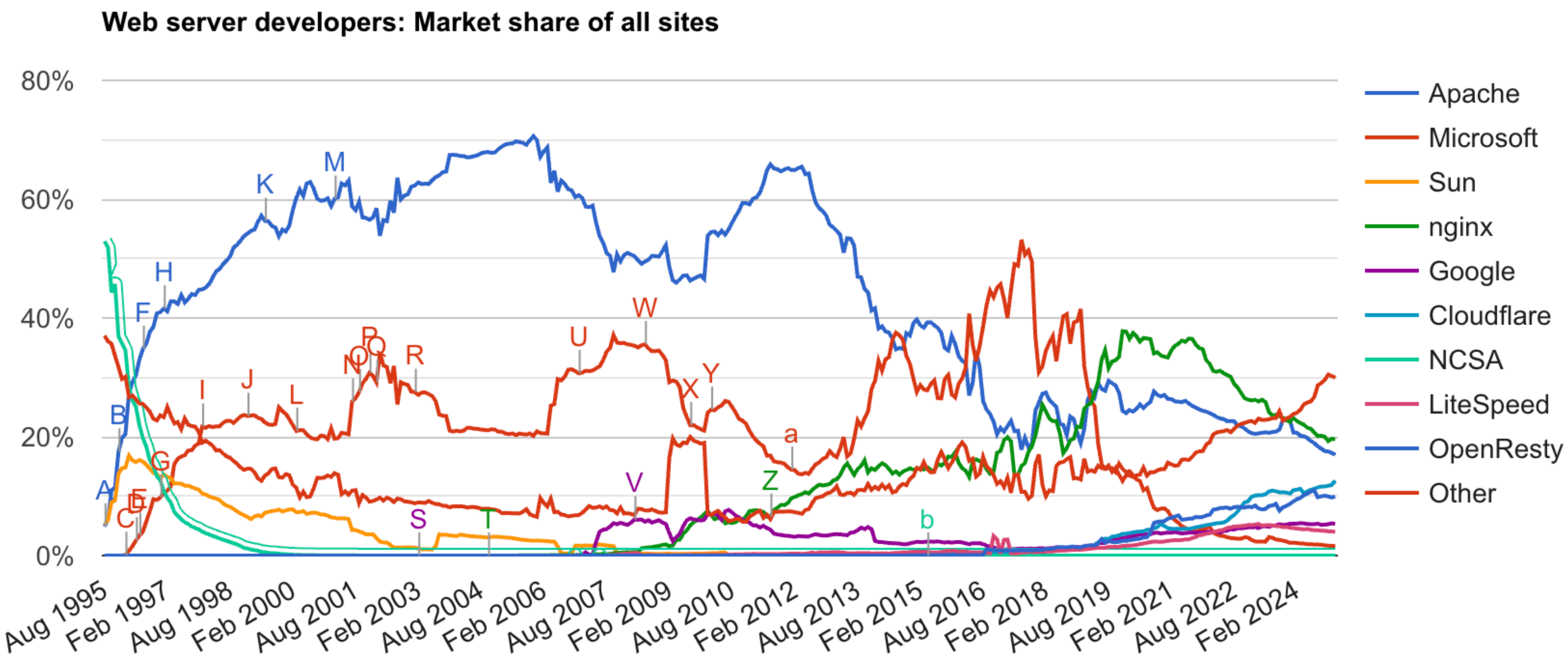
Learn on Heroku + GitHub

Heroku is a cloud-based, platform as a service (PaaS) for building, running, and managing apps. Students use Heroku to learn and grow their skills by taking advantage of the platform's fully managed runtime environment coupled with a wide range of tools and integrated services. Heroku makes app deployment fast and easy. The platform's GitHub integration allows you to connect your Heroku app to your GitHub repo and deploy on every push to GitHub.

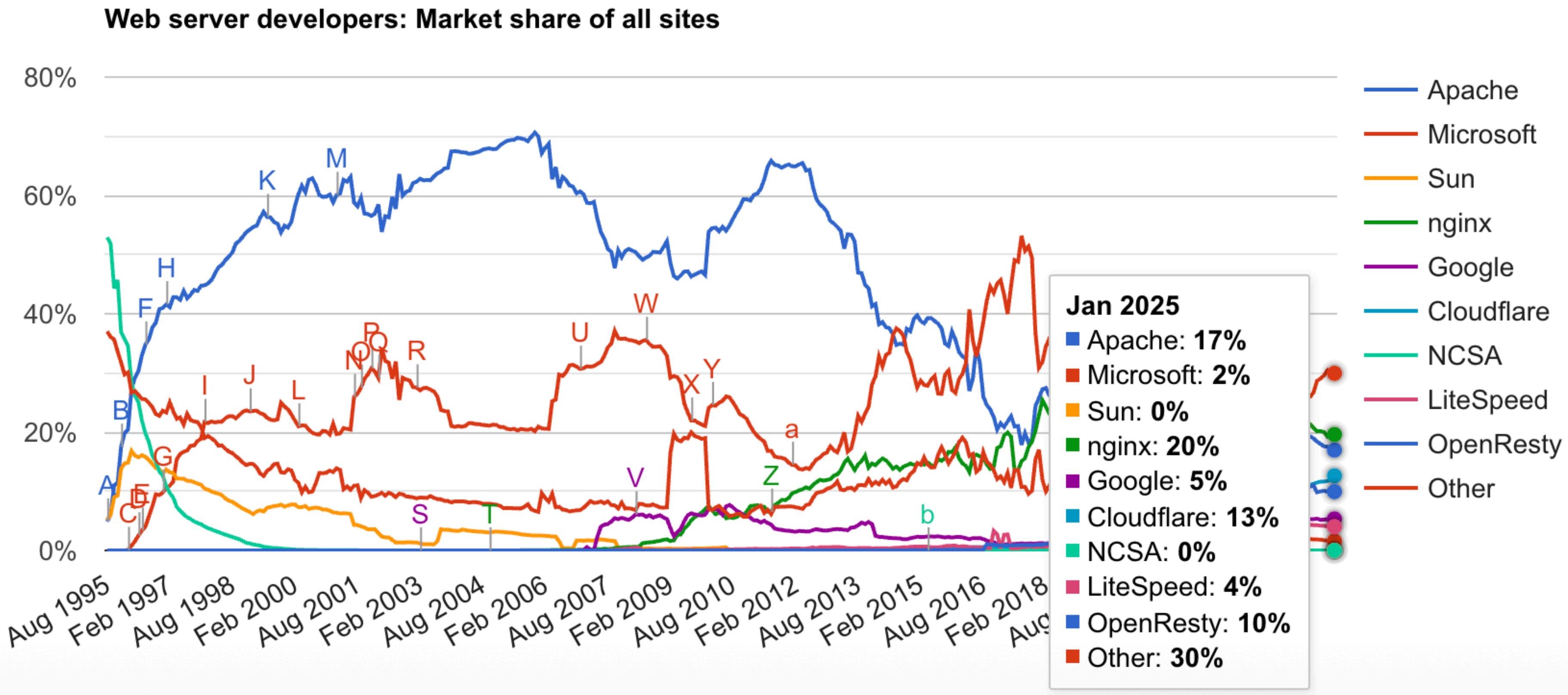
We invite you to explore the Heroku developer experience throughout your studies and beyond. Heroku offers a range of low-cost services to help you experiment, learn, and prototype new ideas. For GitHub Students, we are going a step further and adding even more resources to your GitHub Student Developer Pack.

Web Server History

- 1995 - Apache introduced
- 2002 - Nginx introduced (event-driven approach, non-blocking requests)
- 2009 - Node JS created
- Aug 2011 - Apache serves ~60% of all active web sites
- Jan 2025 - Apache serves now ~17% of all active web sites



https://www.netcraft.com/blog/january-2025-web-server-survey

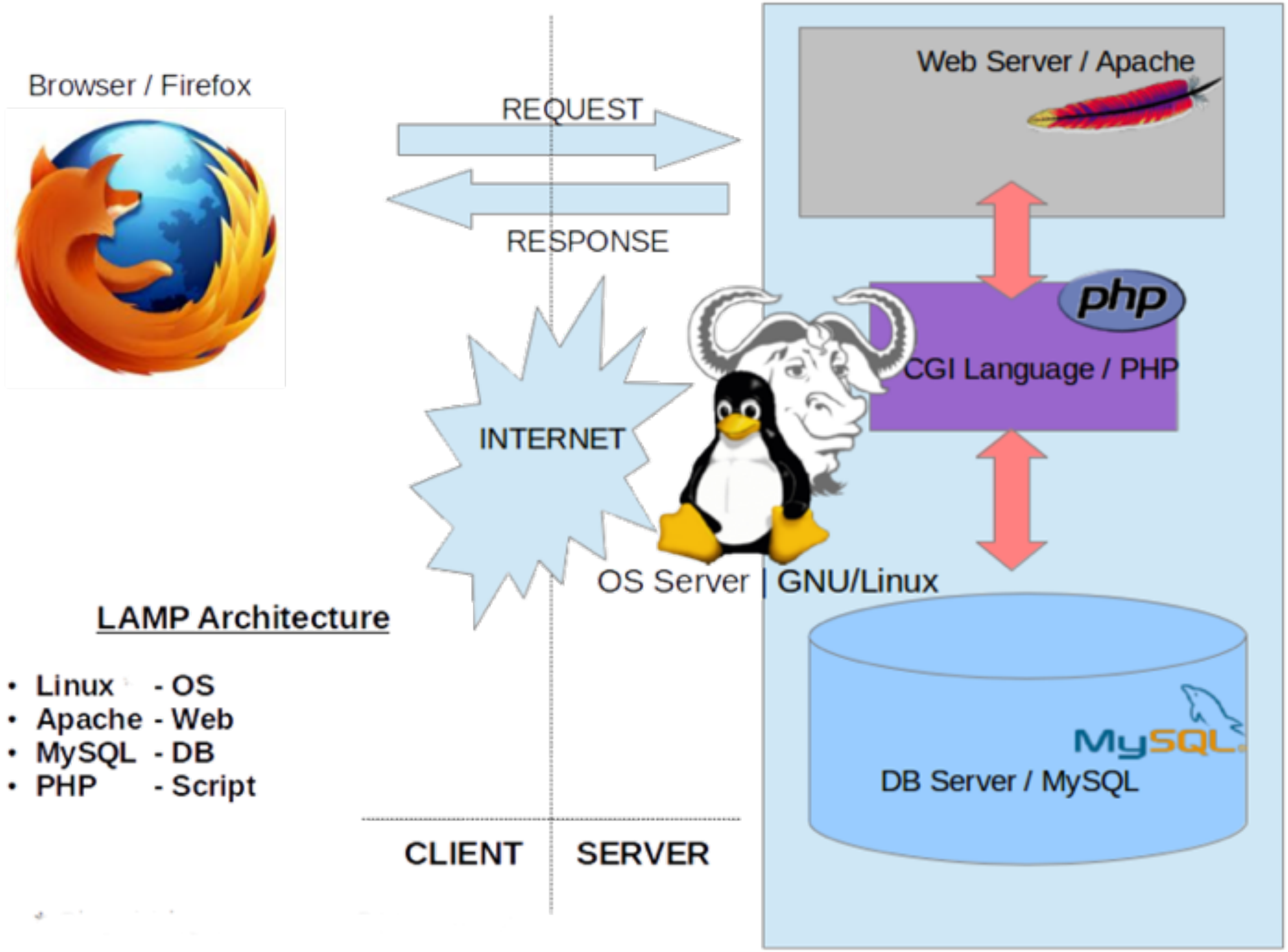


Jan 2025

- Apache: **17%**
- Microsoft: **2%**
- Sun: **0%**
- nginx: **20%**
- Google: **5%**
- Cloudflare: **13%**
- NCSA: **0%**
- LiteSpeed: **4%**
- OpenResty: **10%**
- Other: **30%**

https://www.netcraft.com/blog/january-2025-web-server-survey

LAMP



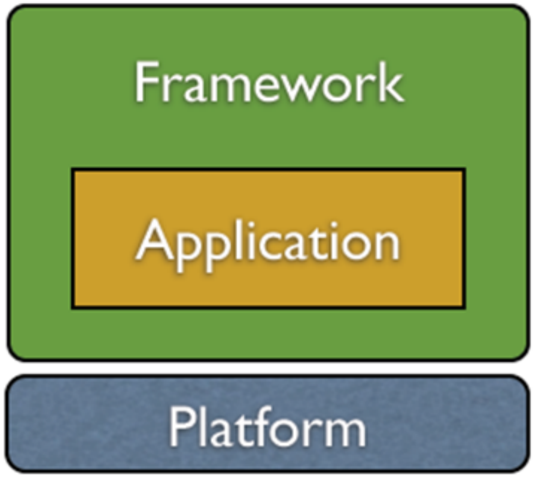
LAMP Architecture

- Linux - OS
- Apache - Web
- MySQL - DB
- PHP - Script

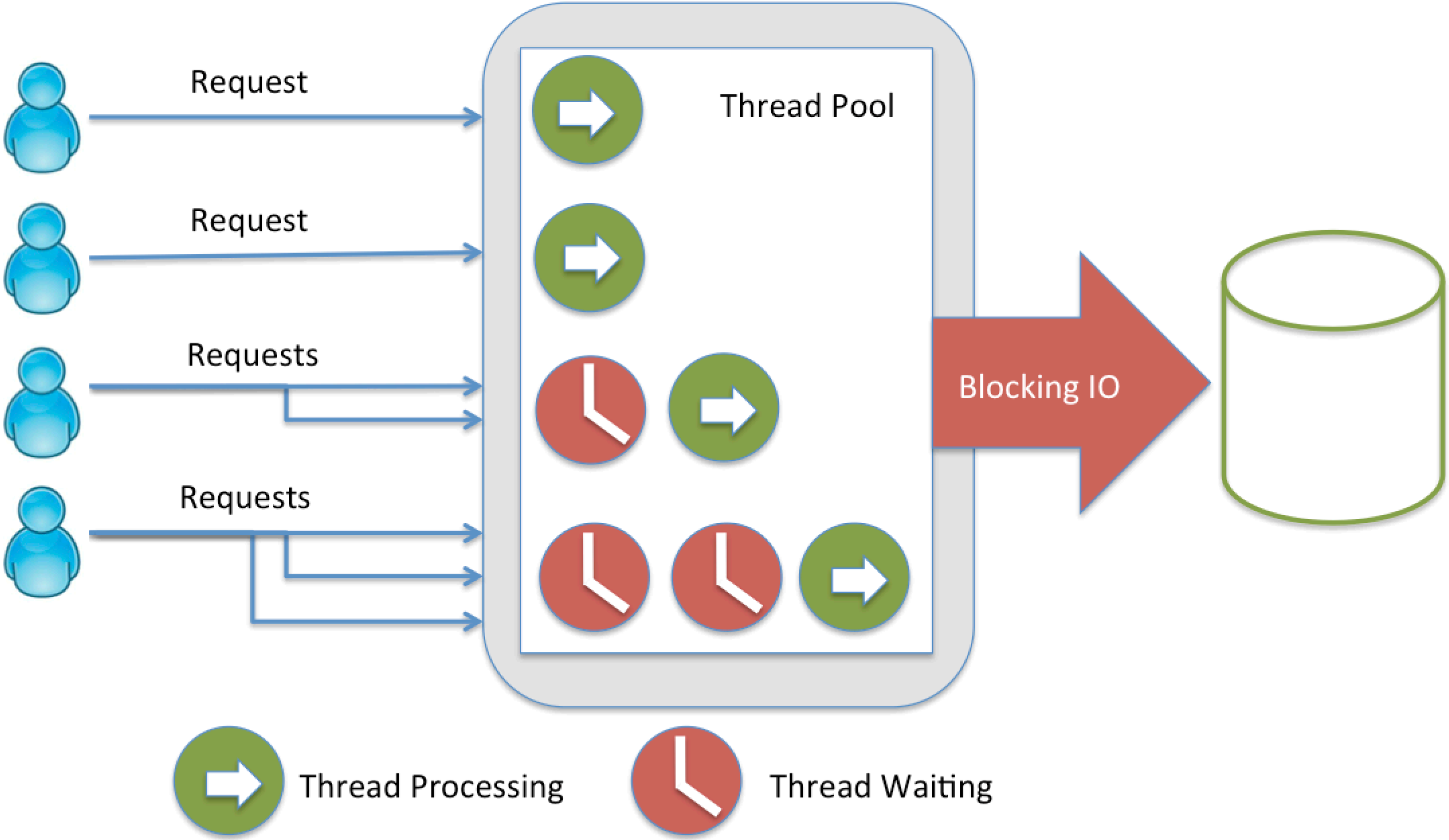
Node JS

- 2009 - Invented by Ryan Dahl
- 2011 - npm created by Isaac Schlueter
- 2015 - Node.js foundation

- Operating System agnostic
- Built on Google's V8 engine
- Asynchronous, event driven, single thread
- Non-blocking and event driven I/O
- Node is a platform (not a framework)



Multi Threaded Server



Request

Request

Requests

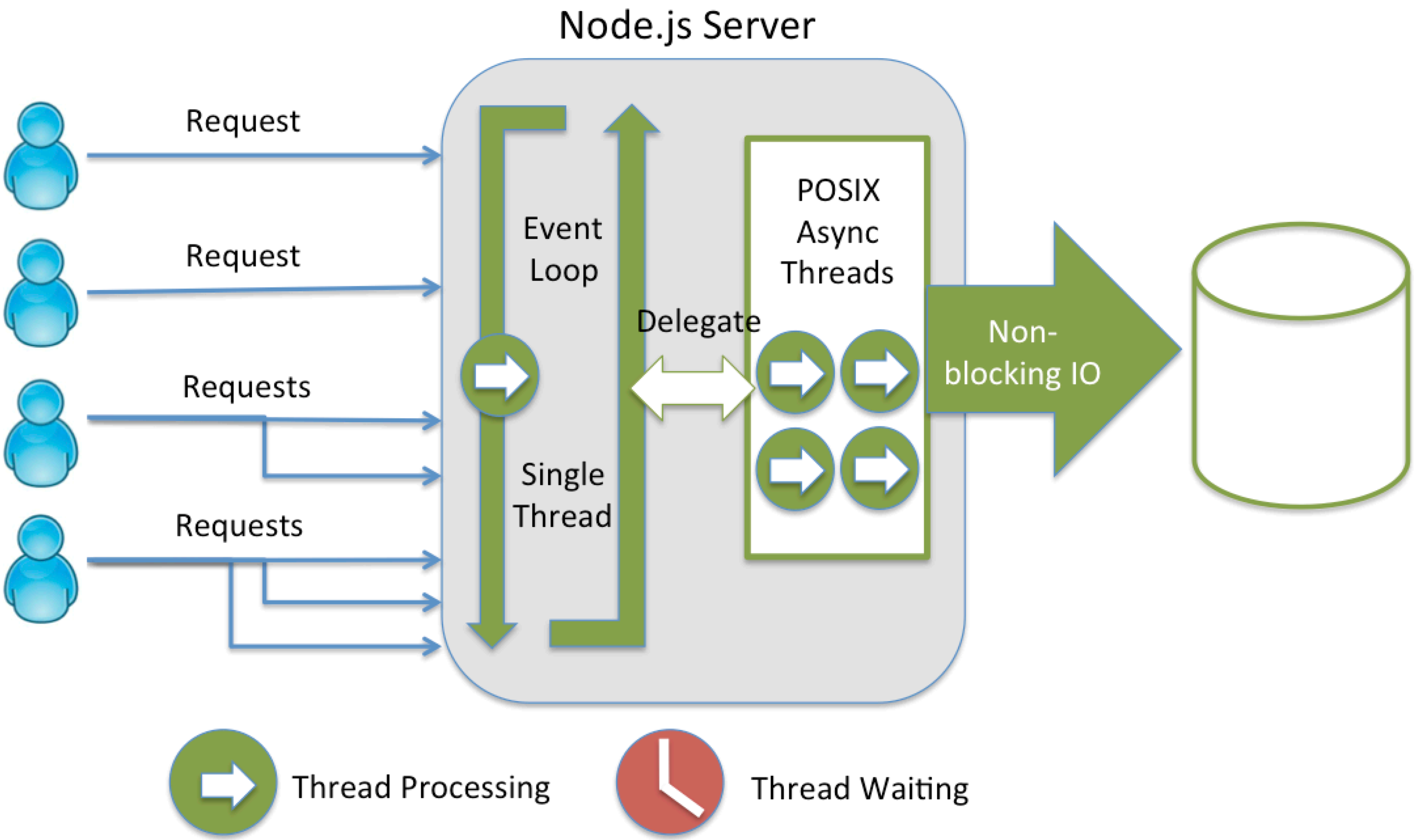
Requests

Thread Pool

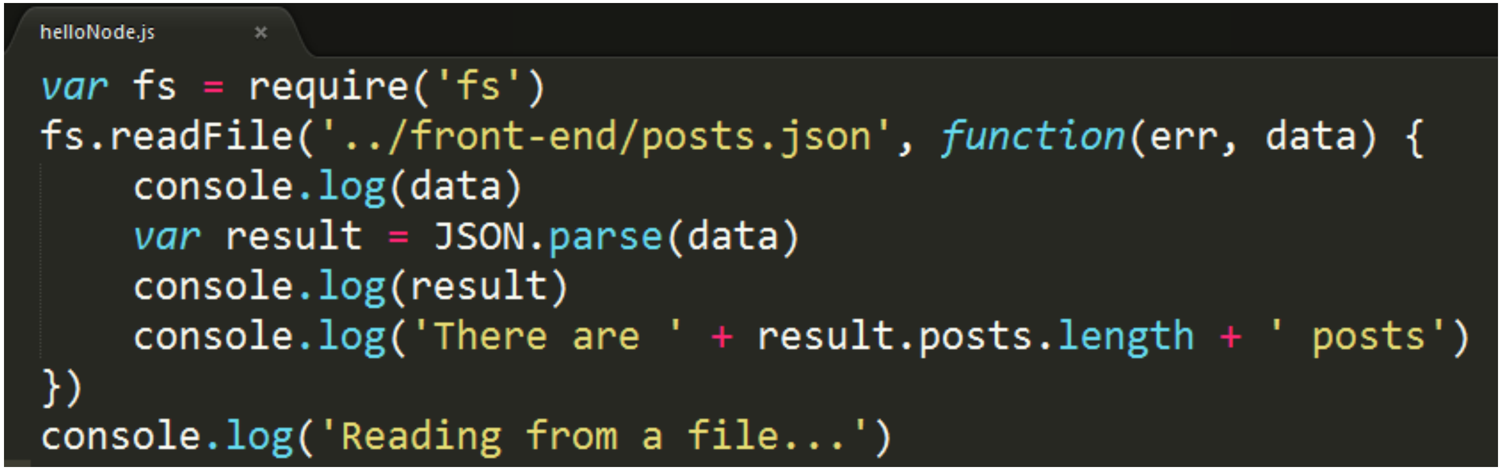
Blocking IO

Thread Processing

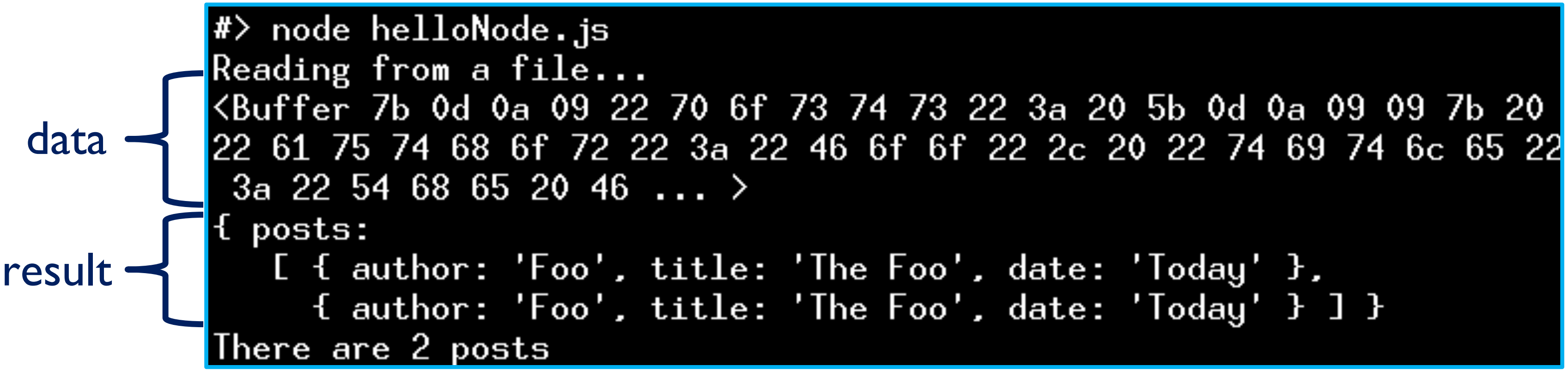
Thread Waiting


Node.js Server
Request
Request
Requests
Requests
Event Loop
Single Thread
Delegate
POSIX Async Threads
Non-blocking IO
Thread Processing
Thread Waiting

Asynchronous I/O

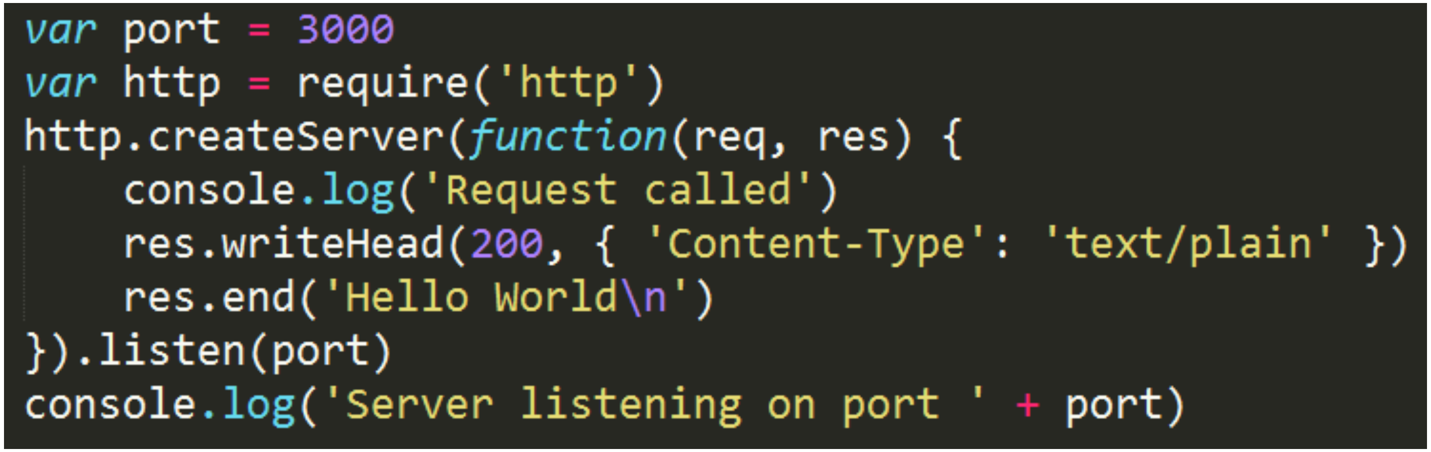


```
var fs = require('fs')
fs.readFile('../front-end/posts.json', function(err, data) {
    console.log(data)
    var result = JSON.parse(data)
    console.log(result)
    console.log('There are ' + result.posts.length + ' posts')
})
console.log('Reading from a file...')
```

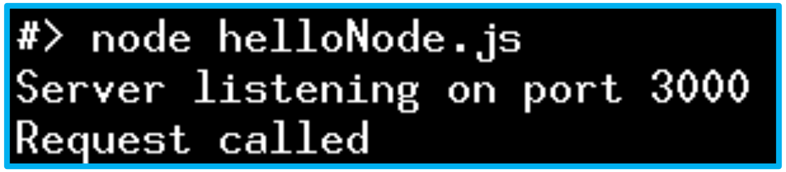


```
#> node helloNode.js
Reading from a file...
<Buffer 7b 0d 0a 09 22 70 6f 73 74 73 22 3a 20 5b 0d 0a 09 09 7b 20
22 61 75 74 68 6f 72 22 3a 22 46 6f 6f 22 2c 20 22 74 69 74 6c 65 22
 3a 22 54 68 65 20 46 ... >
{ posts:
   [ { author: 'Foo', title: 'The Foo', date: 'Today' },
     { author: 'Foo', title: 'The Foo', date: 'Today' } ] }
There are 2 posts
```

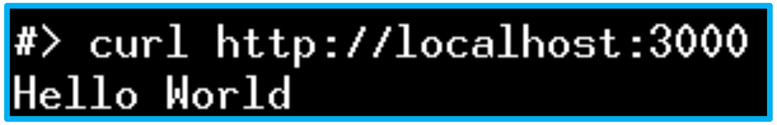
Simple Server



```
var port = 3000
var http = require('http')
http.createServer(function(req, res) {
    console.log('Request called')
    res.writeHead(200, { 'Content-Type': 'text/plain' })
    res.end('Hello World\n')
}).listen(port)
console.log('Server listening on port ' + port)
```



```
#> node helloNode.js
Server listening on port 3000
Request called
```

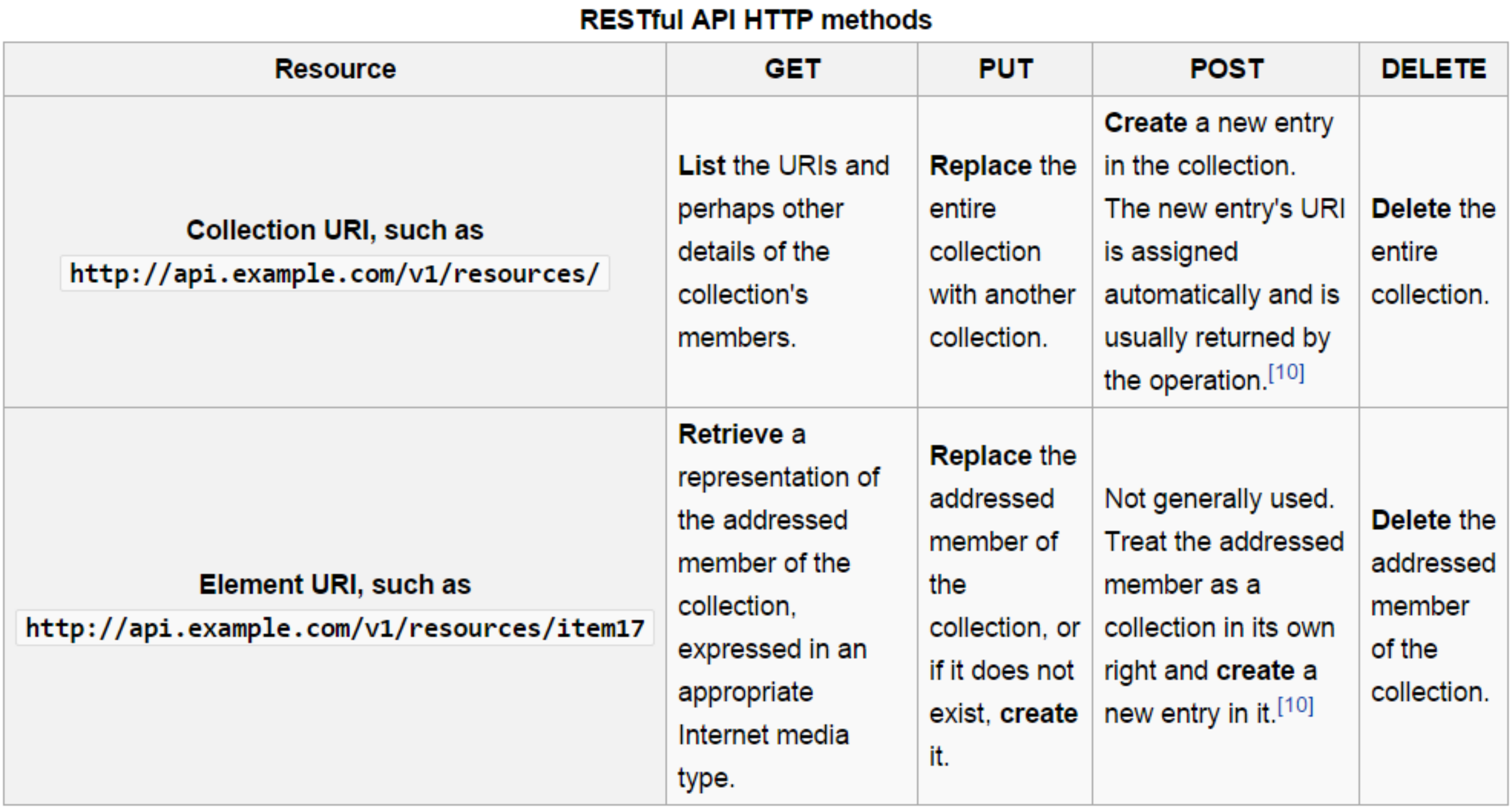


```
#> curl http://localhost:3000
Hello World
```

Representational State Transfer (REST)

- First proposed by Roy Thomas Fielding in 2000 PhD dissertation
- Client-server separation of concerns
  - Simplify component implementation
  - Reduce complexity
  - Increase scalability
- Standard HTTP method verbs
- Typically JSON messages
- REST is stateless

Representational State Transfer Example



RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://api.example.com/v1/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[10]	**Delete** the entire collection.
Element URI, such as `http://api.example.com/v1/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.[10]	**Delete** the addressed member of the collection.

Curl

- Curl is a tool to transfer data to/from a server without user interaction via a browser
- GET is the default request method (-X specifies alt method)
- Common options for POST include:
  - -d (transfer payload)
  - -H (header info to include in request)
  - -i (include header response info in output)
- GET request example:

```
>> curl http://localhost:3333/articles
```

- POST request example:

```
>> curl -d '{"key1": "value1", "key2": "value2"}' -H "Content-Type: application/json" -X POST http://localhost:3333/login
```

Server Http Request

- createServer passes request, response objects as parameters to function argument

req.method	Type of request (GET, POST, etc…)
req.body	Payload for POST request (empty for GET)
req.url	Returns endpoint (e.g. ‘/articles’)
req.on (data, f)	Action to take each time data is seen
req.on (end, f)	Determines action after end of request
JSON.parse(req.body)	Access data as JSON

Server Http Response

- The createServer passes the request and response objects as parameters to its function argument

- res.setHeader(‘Content-type’, ‘application/json’)
- res.statusCode sets the response status (e.g. 200, 404)
- res.end(JSON.stringify(payload)) returns payload
    - payload = { key1: ‘value1’, key2 : ‘value2’}
    - key1, key2 are assumed to be strings (i.e. don’t need quotes)
    - Stringify because *curl* expects a string