44-563: Unit 05

Developing Web Applications and Services

Includes

- Schedule
- Node.js
- Client-Server Communications
- Using Node.js
- Node Apps
- Exam 01 Friday

Schedule

Wk	Topics
1	Intro, static pages, HTML5, CSS3
2	Holiday Day (no class M) Responsive design
3	JavaScript
4	DOM, JQuery, **M04, W04, W05, A02, some workshops began on Friday**
5	Node.js, **Exam 1 Friday**

Exam 1 covers first 4 weeks:
HTML / CSS / Responsive design /
BootStrap / JS / DOM / JQuery

Exam 01 Friday

- Exam 01
- See Exam 01 Review Guide (go back to Welcome page)
- 100 points; 50 questions; all online
- Understand the coding assignments - questions will be similar to a lab exam.
- You may bring one 8-1/2 x 11 sheet of paper with **handwritten** notes (front and back).

Node.js

Node.js is for networks

An easy way to build **scalable network** programs.

http://www.slideshare.net/the_undefined/nodejs-as-a-networking-tool

Node.js

- Complete software **platform** for scalable, server-side and networking applications
- Open-source MIT licence
- Comes with a JavaScript interpreter (REPL) 💗
- Runs on **all major OSes** (Linux, Windows, Mac OS)

Benjamin San Souci & Maude Lemaire, *Node.js*, https://mcgill-csus.github.io/student_projects/NodeJSPresentation.pdf

Node.js on a Timeline

OCTOBER 1, 2009

FIRST VERY EARLY PREVIEW OF NPM, THE NODE PACKAGE MANAGER

Huge Success

Benjamin San Souci & Maude Lemaire, *Node.js*, https://mcgill-csus.github.io/student_projects/NodeJSPresentation.pdf

Why so popular?

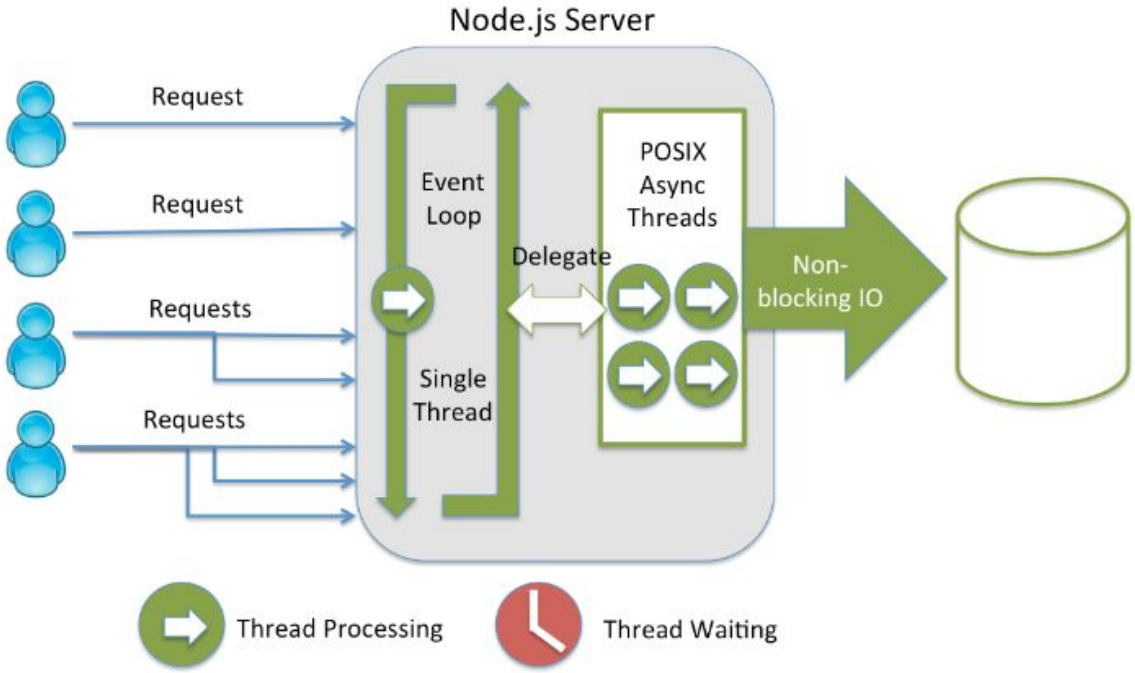
- Until recently, web was **stateless** (no memory).
- Interactive features were encapsulated within Flash or Java Applets
- Node enables **real-time, two-way** connections!
- It's fast:

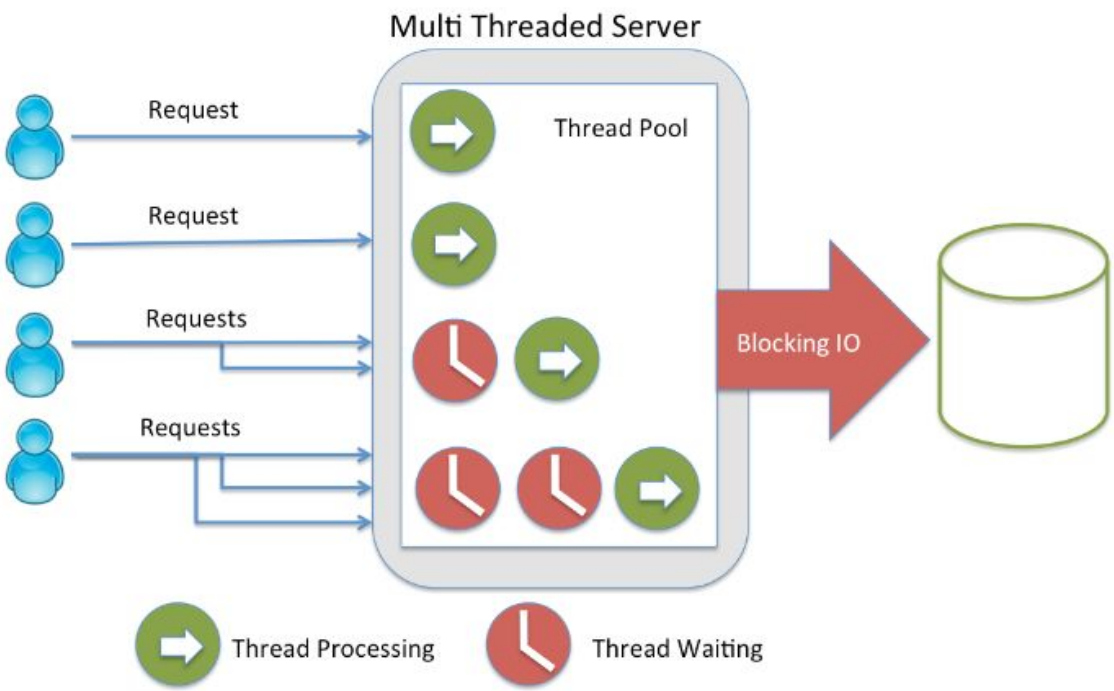
Number of iterations	Node.js	PHP
100	2.00	0.14
10'000	3.00	10.53
1'000'000	15.00	1119.24
1'000'000'000	11118.00	1036272.19

Platform	Number of request per second
PHP (via Apache)	3187,27
Static (via Apache)	2966,51
Node.js	5569,30

Benjamin San Souci & Maude Lemaire, *Node.js*, https://mcgill-csus.github.io/student_projects/NodeJSPresentation.pdf

Team 61, Team 51 (IU)





https://strongloop.com/strongblog/node-js-is-faster-than-java/

How it works

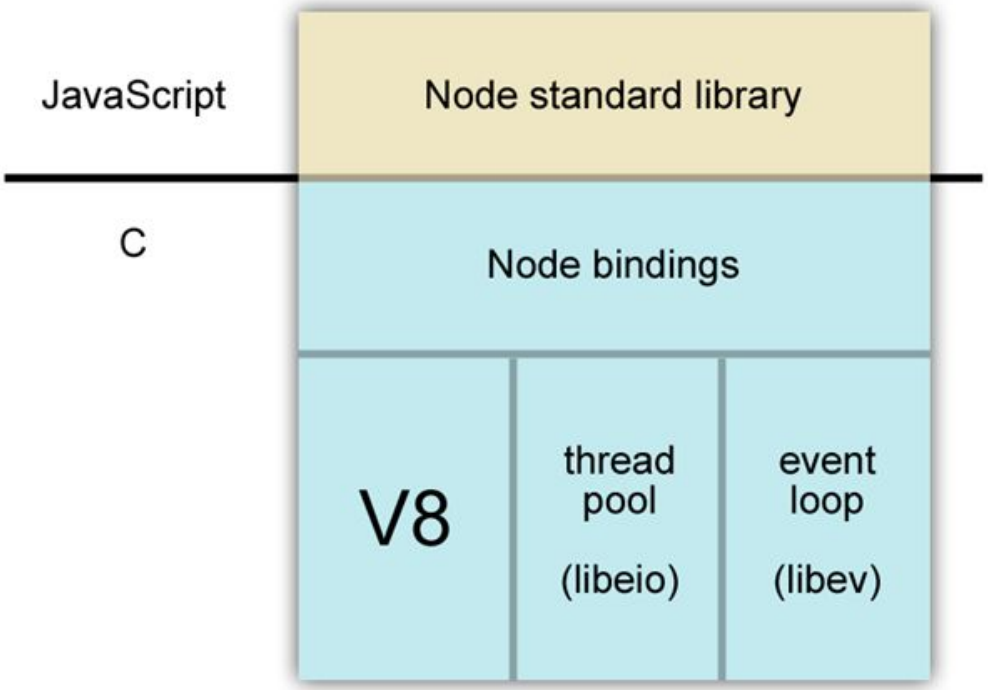
- Built on Chrome's **V8 JavaScript runtime** for easily building fast, scalable network applications
- Uses an **event-driven**, **non-blocking** I/O model that makes it lightweight and efficient, perfect for data intensive real-time applications across **distributed** devices

Benjamin San Souci & Maude Lemaire, *Node.js*, https://mcgill-csus.github.io/student_projects/NodeJSPresentation.pdf

Overall Structure

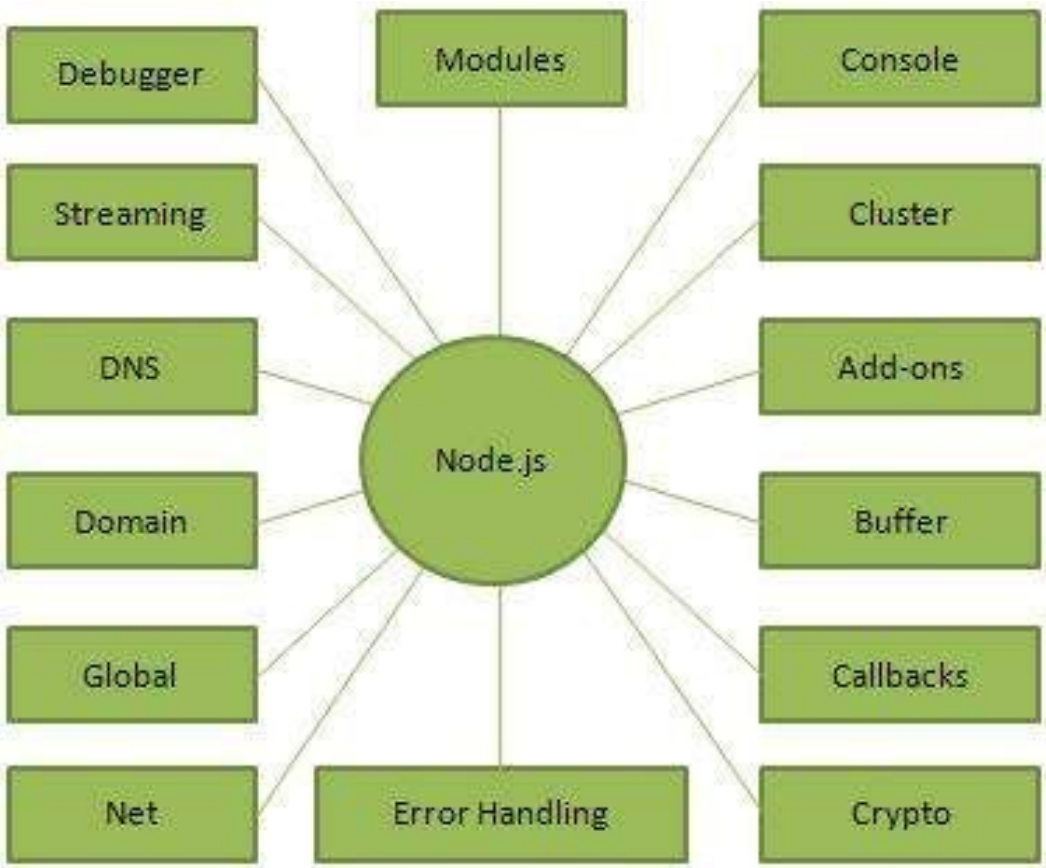
Two major components:

- Main Core, written in **C and C++**
- Modules, such as Libuv library and **V8 JavaScript runtime** engine (also written in C++)



Benjamin San Souci & Maude Lemaire, *Node.js*, https://mcgill-csus.github.io/student_projects/NodeJSPresentation.pdf

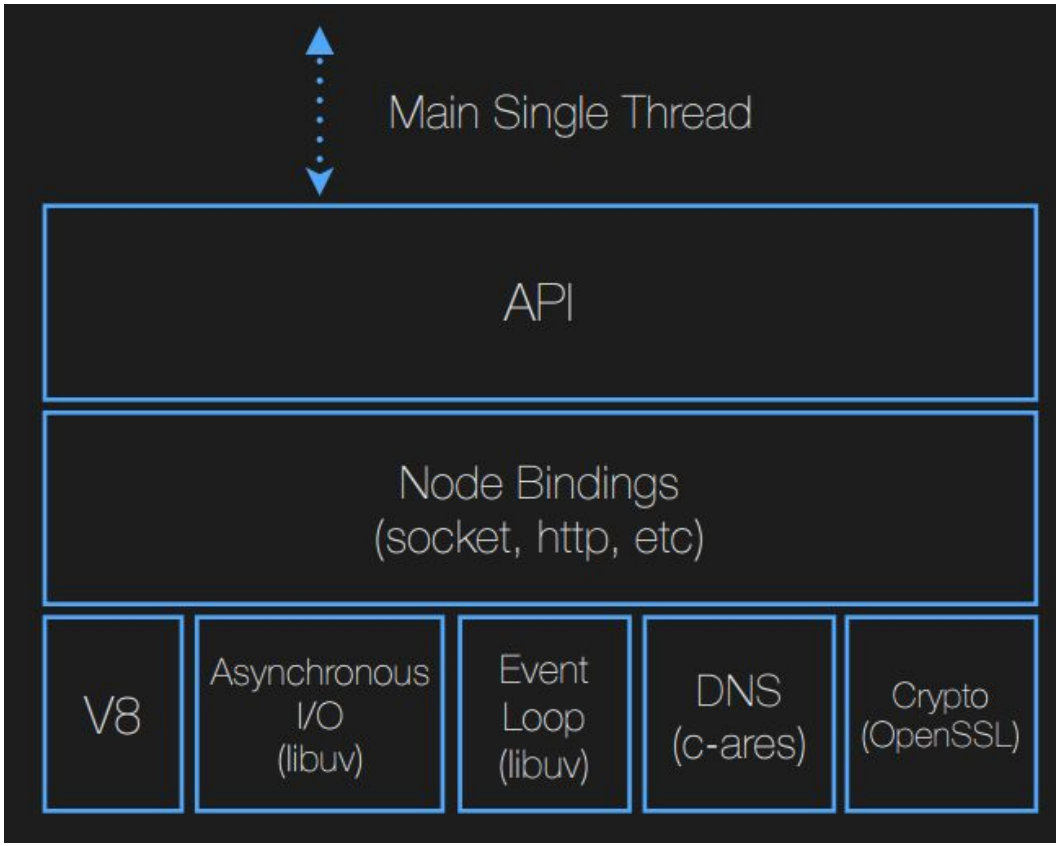
Some key components



https://www.tutorialspoint.com/nodejs/nodejs_introduction.htm

Architecture

All requests handled by the Main Single Thread



- **API** in JavaScript
- Node bindings allow for server operations
- **Libuv** responsible for both asynchronous I/O & **event loop**

https://mcgill-csus.github.io/student_projects/NodeJSPresentation.pdf

Blocking I/O

```
db.query ('SELECT A')        // wait until each statement finishes
console.log('query A done')

db.query ('SELECT B')
console.log('query B done')
```

Time = **SUM**(A, B)

Felix Geisendörfer, *Node.js - As a networking tool*, LinkedIn SlideShare

Non-Blocking I/O

```
db.query ('SELECT A', function(result) {  // start & proceed
   console.log('query A done')
})
db.query ('SELECT B', function(result) {  // doesn't wait to start
   console.log('query B done')
})
```

Time = **MAX**(A, B)

Callbacks

A **callback** function is called at the completion of a given task.

```
db.query ('SELECT A', function(result) {
   console.log('query A done')
})
```

Non-Blocking I/O

- Close to ideal for high **concurrency** / high **throughput** single execution stack
- Feels like Ajax in the browser
- Forces you to write more efficient, concurrent code that can take advantage of **parallel processing of** your I/O

Non-blocking IO: A Metaphor

A typical experience at a restaurant would be something like this:

1. You sit at a table and the server grabs your **drink order.**
2. The server goes back to the bar and passes your order to a **bartender**.
3. While the bartender is working on your drink, the server moves on to grab another table's **drink order**.
4. The server goes back to the **bartender** and passes along the other table's order.
5. Before the server brings back your drinks, you order some **food**.
6. Server passes your food order to the **kitchen**.
7. Your drinks are ready now, so the server picks up your **drinks** and brings them back to your table.
8. The other table's **drinks** are ready, so the server picks them up and takes them to the other table.
9. Finally your **food** is ready, so server picks it up and brings it back to your table.

The restaurant server can only process one request at a time (they only have two hands!), just like your application code. They turn the request over to the kitchen (the OS), but they do not need to wait for the task to be done (non-blocking).

https://www.codeschool.com/blog/2014/10/30/understanding-node-js/

Non-Blocking IO: A Metaphor

```
// requesting drinks for table 1 and waiting...
var drinksForTable1 = requestDrinksBlocking(['Coke', 'Tea', 'Water']);
// once drinks are ready, then server takes order back to table.
serveOrder(drinksForTable1);
// once order is delivered, server moves on to another table.

// requesting drinks for table 2 and waiting...
var drinksForTable2 = requestDrinksBlocking(['Beer', 'Scotch', 'Vodka']);
// once drinks are ready, then server takes order back to table.
serveOrder(drinksForTable2);
// once order is delivered, server moves on to another table.

// requesting food for table 1 and waiting..
var foodForTable1 = requestFoodBlocking(['Burger', 'Salad', 'Pizza']);
// once food is ready, then server takes order back to table.
serveOrder(foodForTable1);
// once order is delivered, server moves on to another table.
```

https://www.codeschool.com/blog/2014/10/30/understanding-node-js/

Non-Blocking IO: A Metaphor

```
// requesting drinks for table 1 and moving on...
requestDrinksNonBlocking(['Coke', 'Tea', 'Water'], function(drinks){
  return serveOrder(drinks);
});

// requesting drinks for table 2 and moving on...
requestDrinksNonBlocking(['Beer', 'Scotch', 'Vodka'], function(drinks){
  return serveOrder(drinks);
});

// requesting food for table 1 and moving on...
requestFoodNonBlocking(['Burger', 'Salad', 'Pizza'], function(food){
  return serveOrder(food);
});
```

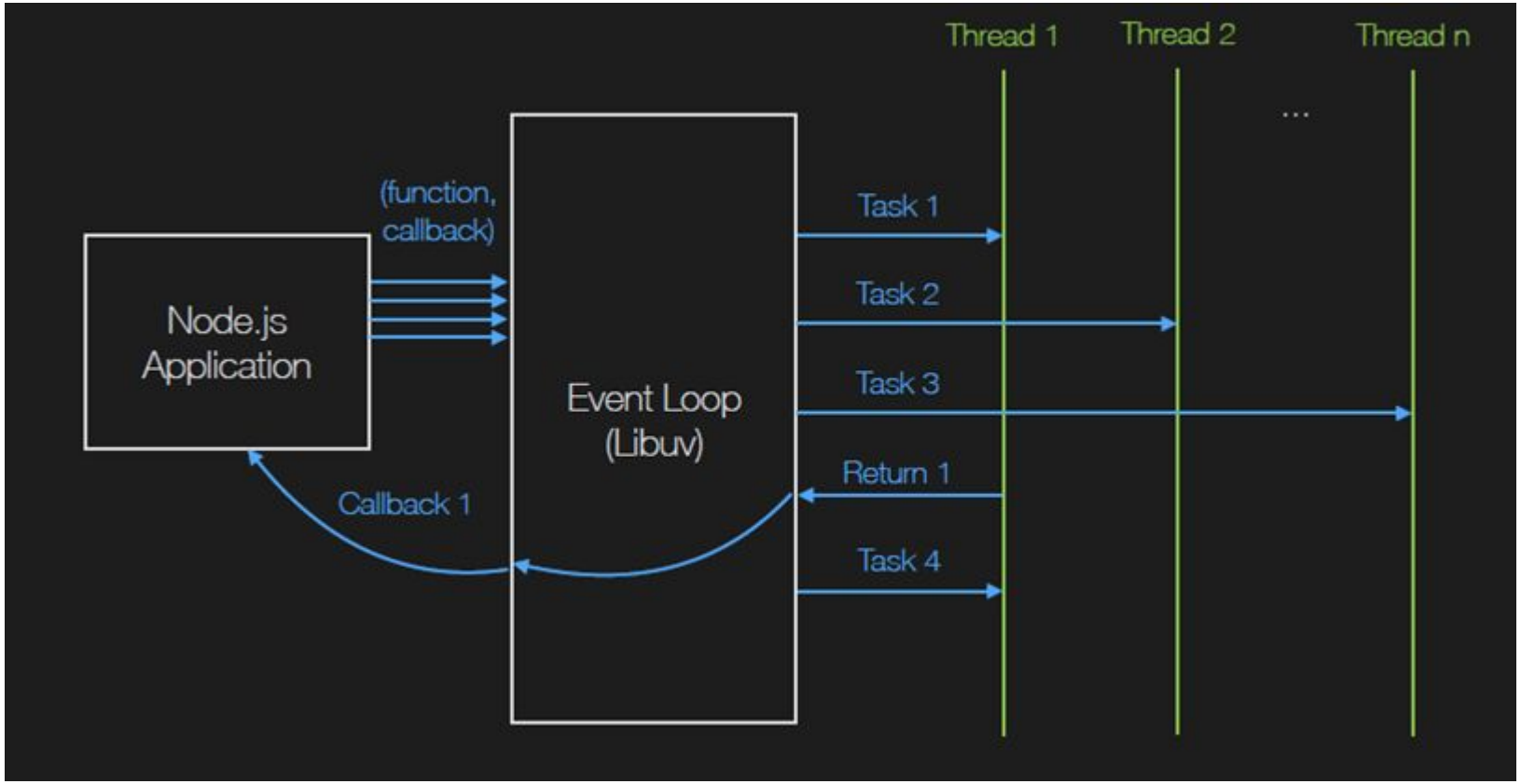
https://www.codeschool.com/blog/2014/10/30/understanding-node-js/

libuv - infinite loop

- high-performance event-based **I/O library**
- same API on Windows and Unix
- polls the operating system (OS) for **events**
- lets us register watchers / callbacks for the events you care about
- Earlier versions were libev, libevents
- More info: An introduction to libuv at https://nikhilm.github.io/uvbook/index.html

Image: https://github.com/libuv/libuv

libuv - infinite loop



M05

Let's Try It (M05)

- Go to https://nodejs.org/en/
- Install Node.js for Windows (or macOS, or Linux, as appropriate)
- Select latest Long-Term Support LTS version recommended for most users
- To verify, open Windows command prompt (or PowerShell or Terminal)
  - node -v
  - npm -v
- Open the REPL (read-eval-print-loop)
  - node
  - > console.log("hello, world!")
  - > 5 * 5

Let's Try It (M05)

```
> var x = 10
undefined
> var y = 20
undefined
> x + y
30
> var sum = _
undefined
> console.log(sum)
30
undefined
>
```

Type just the lines in bold

Underscore is short-hand for "last result"

REPL commands

- **ctrl + c** - terminate the current command.
- **ctrl + c twice** - terminate the Node REPL.
- **ctrl + d** - terminate the Node REPL.
- **Up/Down Keys** - see command history and modify previous commands.
- **tab Keys** - list of current commands.
- **.help** - list of all commands.
- **.break** - exit from multiline expression.
- **.clear** - exit from multiline expression
- **.save *filename*** - save current Node REPL session to a file.
- **.load *filename*** - load file content in current Node REPL session.

https://www.tutorialspoint.com/nodejs/nodejs_repl_terminal.htm <--- Click here for things to try

For Windows Users

Add "**Open Command Window here as Administrator**" to your Windows Explorer menu.

Opens a command prompt in the given folder.

http://www.sevenforums.com/tutorials/47415-open-command-window-here-administrator.html

Client Server Communication

Communicating with HTTP

HTTP is a stateless protocol.

Stateless protocols **do not require** server to store information about each user over multiple requests.

Each request appears independent, but developers can use other methods to maintain information (e.g. cookies, server sessions, hidden elements in HTML, and url parameters, e.g. sessionID).

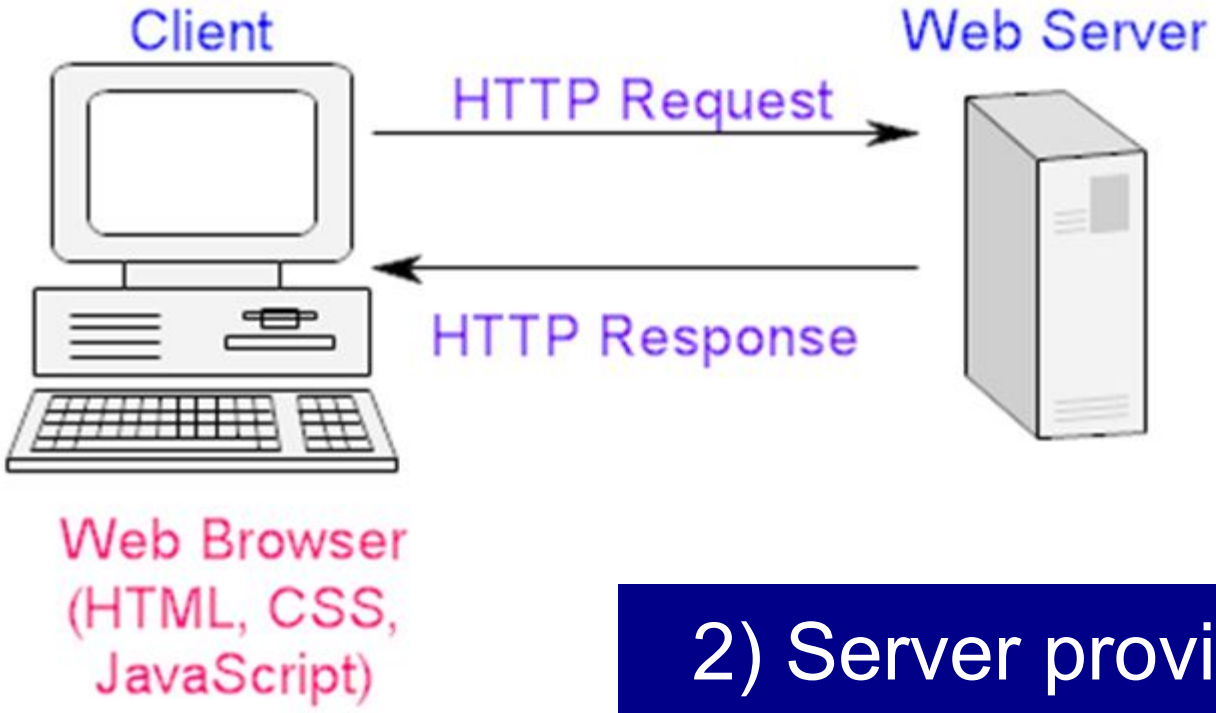
What can you do with HTTP? A Spectrum of HTTP-Based Communication Strategies

1. **Polling** - a client sends requests to the server -- hey, do you have anything for me? If not, the server closes the connection, the client waits a bit (e.g., 1 s), and then requests again. This a type of "pull" communication.
2. **Long-Polling** - Instead of immediately closing the connection, it keeps the connection open until it *does* have data, and then it closes the connection. Also "pull" based.
3. **Server-Sent Events** - Instead of closing the connection after one message, the connection remains open, and other messages can be sent over that connection. The server is basically in control, hence the name. This is a type of "push" communication.
4. **WebSockets** - This is a full duplex connection - "push" and "pull". The client sends a request with an HTTP header that says the client can handle websockets. If the server can do so as well, then a websocket is established. Both the client and server can send data to the other, whenever.

https://jersey.java.net/documentation/latest/sse.html

Traditional Client Side pull

1) Client makes a request

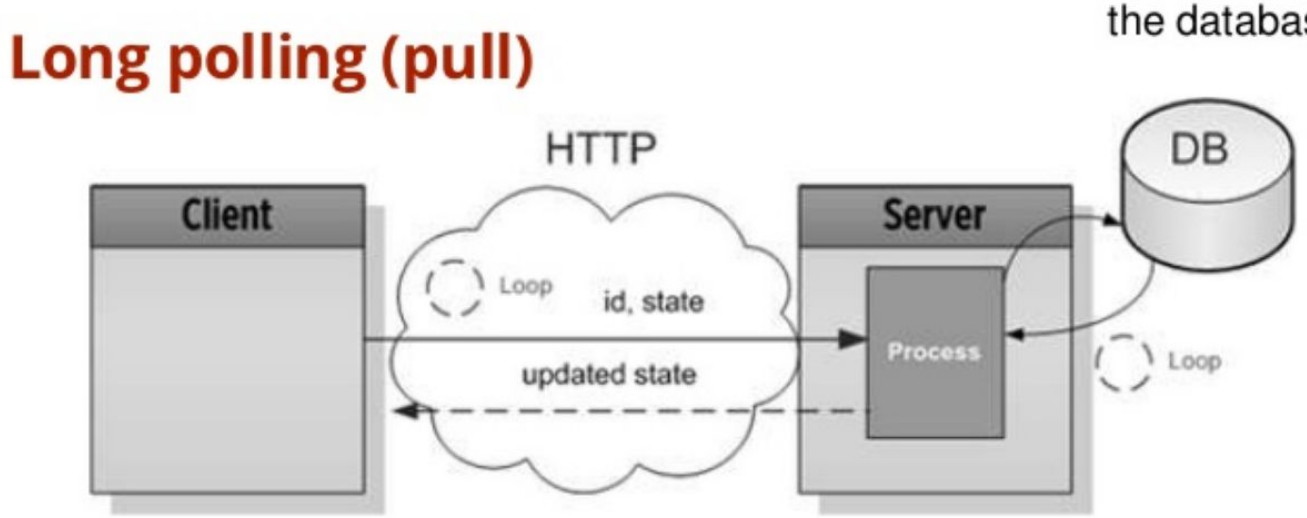


2) Server provides a response

HTTP Long Polling

1) Client makes a request

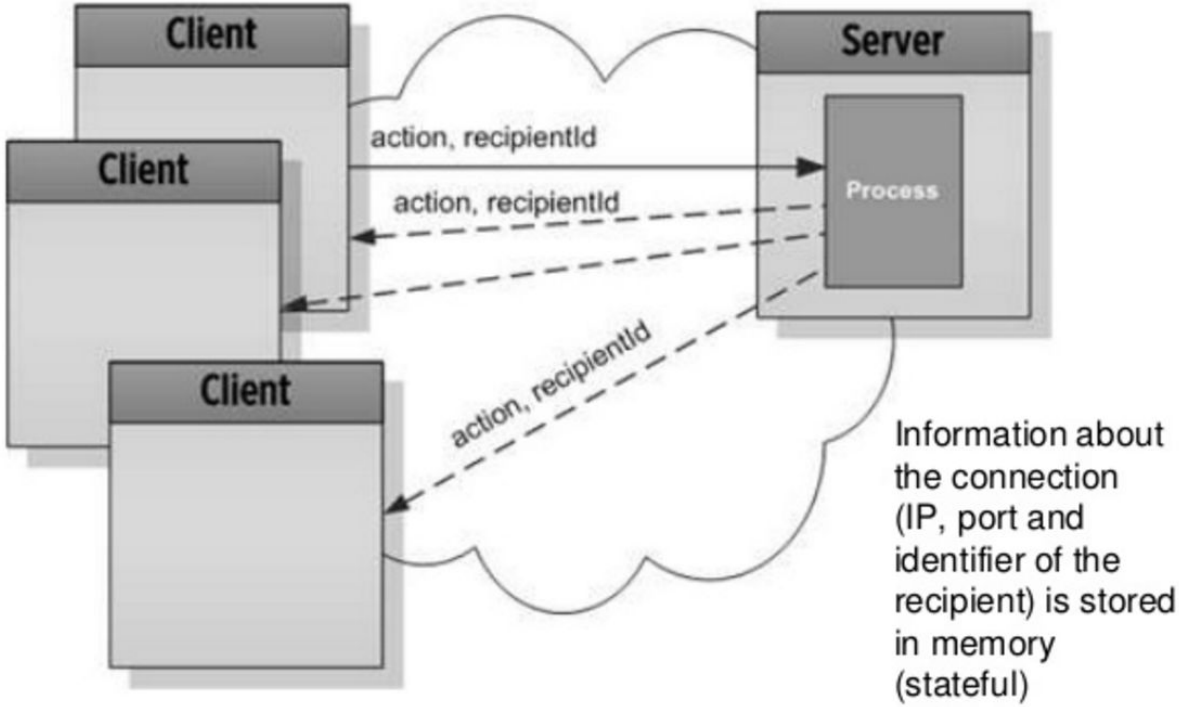
Since HTTP is stateless, the Session and other data might need to be read from the database



http://www.slideshare.net/OutSystems/enterprise-application-performance-with-server-push-technologies-web-sockets

2) Server holds request open until new data is available, then responds. This loop continues.

WebSocket protocol (push/pull)



Server can originate communications

WebSocket Protocol

- Message-based transport.
- Bidirectional, **full-duplex**, long-lived.
- Uses initial HTTP-friendly handshake, and ports 80 and 443
- After initial handshake, very little message overhead.
- Messages are handled asynchronously.

See this discussion for an interesting comparison of Long Polling v. WebSockets

Full Duplex v. Half Duplex

Full duplex communication:

- Both sides can transmit & receive simultaneously.
- Cell phone conversations are full duplex.

Half duplex communication:

- Only one side can transmit / receive at a time.
- Walkie-talkie conversations are half duplex.

Node.js: enables push

- Communication initiated by publisher/server or client
- Persistent connection between browser / server
- Very hard / awkward on traditional stacks (e.g. PHP)
- Hard to scale on traditional stacks

Using Node.js

Node.js Use Cases

- **WebSockets/push applications**
- Processing data streams (e.g. Twitter)
- Parallelizing I/O
- Not much CPU processing
- Spawning other programs (processes) to do work

Node.js Non-Use Cases

- Lots of computations / high CPU requirements
- CRUD apps (create-read-update-delete)
- Heavy-weight real-time systems
- Number crunching / huge in-memory datasets

npm Package Manager

- Default package manager for Node.js (manages all the dependencies - over 5,000 options, easily installed)
- Automatically included when you install Node.js
- Most starred packages: https://www.npmjs.com/browse/star

Some npm factoids

- Node comes with ~30 packages built-in; you will often install others.
- npm installs packages (inside a **node_modules** folder) that give a node project extra functionality. We install locally in class so all have similar environments.
- When you require('a_module'), it returns the built-in **module.exports** object.

Examples:

```
npm install package      # installs package locally, this project only (preferred)
npm install package -g   # installs package globally for all projects
npm install              # installs all dependencies listed in package.json
                         # if global, be sure to note it in the README
```

Try it:

```
npm install chalk
npm install cool-ascii-faces
```

```
var chalk = require('chalk')
console.log(chalk.red("I am red text"))
I am red text
var cool = require('cool-ascii-faces')
undefined
> console.log(cool())
(ง°ل͜°)ง
```

package.json

Node projects have a **package.json** file in the root folder. When you install a package with --save option, it adds the dependency to package.json.

Reinstalling all dependencies is done with:

```
npm install
```

To **create** one of these mythical beasties, just write:

```
npm init
```

and follow the prompts.

To make your *own* package that other people can use, here is [one way to get started](#).

```
{
  "name": "treesforever",
  "version": "0.0.1",
  "description": "A node project involving trees that will
probably take ... forever",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://xx@bitbucket.org/xx/treesforever.git"
  },
  "keywords": [
    "trees",
    "forever"
  ],
  "author": "Somebody or other",
  "license": "ISC",
  "homepage":
"https://bitbucket.org/xx/treesforever#readme",
"dependencies": {
    "chalk": "^1.1.3",
    "cool-ascii-faces": "^1.3.4"
  }
}
```

Socket.IO

- JavaScript **library** for real-time web apps
- Client-side library that runs in the browser
- Server-side library for Node.js
- Both have nearly the same API
- Event-driven (like Node.js)
- It *uses* WebSockets, but has other features (e.g., can broadcast to multiple clients, store data with clients)
- If working with .NET, **SignalR** is an alternative.

Wikipedia Socket.IO article at https://en.wikipedia.org/wiki/Socket.IO
Ultra-Cool Demo: https://socket.io/demos/chat/

A Brief Look at HTTP

HTTP Message Examples

Request:

```
GET /doc/test.html HTTP/1.1
Host: www.test101.com
Accept: image/gif, image/jpeg, */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0
Content-Length: 35

bookId=12345&author=Tan+Ah+Teck
```

Request Line

Request Headers

Request Message Header

A blank line separates header & body

Request Message Body

Response:

```
HTTP/1.1 200 OK
Date: Sun, 08 Feb xxxx 01:11:12 GMT
Server: Apache/1.3.29 (Win32)
Last-Modified: Sat, 07 Feb xxxx
ETag: "0-23-4024c3a5"
Accept-Ranges: bytes
Content-Length: 35
Connection: close
Content-Type: text/html

<h1>My Home page</h1>
```

Status Line

Response Headers

Response Message Header

A blank line separates header & body

Response Message Body

https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/images/HTTP_RequestMessageExample.png
https://www.ntu.edu.sg/home/ehchua/programming/webprogramming/images/HTTP_ResponseMessageExample.png

More Details Coming Later ...

Node App: Hello World

Main.js (named function)

```
var http = require("http")  // similar to includes

http.createServer(requestListener).listen(8081)

function requestListener (request, response) {

   // Send the HTTP header
   // HTTP Status: 200 : OK
   // Content Type: text/plain
   response.writeHead(200, {'Content-Type': 'text/plain'})

   // Prepare and send the response body
   response.end('Hello World!\n')
}

console.log('Server running at http://127.0.0.1:8081/')
```

Open command window in folder, type **node main.js**
Or just **node main**

```
C:\44563\W05>node main.js
Server running at http://127.0.0.1:8081/
```

Explore your application

// Execute the main.js to start the server:

```
$ node main.js
```

// Verify the Output. Server has started

```
Server running at http://127.0.0.1:8081/
```

Make a request to Node.js server by opening

http://127.0.0.1:8081/

in any browser.

http://127.0.0.1:8081/

Main.js (anonymous)

```
var http = require("http")  // similar to includes

http.createServer(function (request, response) {

   response.writeHead(200, {'Content-Type': 'text/plain'})

   response.end('Hello World!\n')

}).listen(8081)

// Console will print the message
console.log('Server running at http://127.0.0.1:8081/')
```

In JavaScript, writing anonymous functions is more common.
Try it: Open in a browser & click **view page source**

Request listener function

```
var http = require("http")  // similar to includes

http.createServer(function (request, response) {

   response.writeHead(200, {'Content-Type': 'text/plain'})

   response.end('Hello World!\n')

}).listen(8081)
```

The argument for createServer is an optional request listener function. It is a **closure** - a method that has access to the surrounding state. (It may or may not be anonymous.) It includes an incoming message request object and provides an outgoing server response.

Many ways to code

```
var http = require("http")  // similar to includes

http.createServer(function (request, response) {

   response.writeHead(200, {'Content-Type': 'text/plain'})

   response.end('Hello World!\n')

}).listen(8081)
```

```
var http = require("http")  // similar to includes

const server = http.createServer(function (request, response) {

   response.writeHead(200, {'Content-Type': 'text/plain'})

   response.end('Hello World!\n')

})
server.listen(8081)
```

Main.js (writing html)

```
var http = require("http")

http.createServer(requestListener).listen(8081)

function requestListener(request, response) {
    response.writeHead(200, { 'Content-Type': 'text/html' })
    response.write("<!DOCTYPE html>")
    response.write("<html><head><title>W05</title></head>")
    response.write("<body><h1>Hello World!!</h1></body></html>")
    response.end()
}

console.log('Server running at http://127.0.0.1:8081/')
```

Try it: Open in a browser & click **view page source**
Optional: Use **\n** to create newlines to format the html.

Node App: Weather

https://bitbucket.org/professorcase/weathernode/src

Creating a Weather app

weatherReader.js

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

function printWeather(city, weather) {
    console.log('In ' + city + ', it is ' + weather + ' degrees C.')
}
function printError(error) {  console.error(error.message) }

module.exports = function get(city){
    var request = http.get('http://api.openweathermap.org/data/2.5/weather?q='+
    city + '&units=metric&apikey=c184205bc1fcbcdc42c4b37ccf710de3', responseFunction)

    function responseFunction(response) {
        var body = ''
        response.on('data', function(chunk) { body += chunk }) //On getting data, do this
        response.on('end', function() {  // on completion, do this
            if (response.statusCode === 200) {
                try {
                    var weatherAPI = JSON.parse(body)
                    printWeather(weatherAPI.name, weatherAPI.main.temp)
                } catch(error) {  printError(error)   }
            } else {
                printError({message: 'Error getting weather from ' + city + '. (' +
                                     http.STATUS_CODES[response.statusCode] + ')'})
            }
        })
  }
  request.on('error', printError)  // // on getting an error, do this
}
```

https://bitbucket.org/professorcase/weathernode

Creating a Weather app

getWeather.js

```
var getWeather = require('./weatherReader.js')('Maryville, Missouri')

// calls weatherReader.js module.exports function and passes in one argument (city)

// If we wanted to do this in two lines instead, we would write:

var getWeather = require('./weatherReader.js')
getWeather('Maryville, Missouri')
```

```
C:\44563\W05>node getWeather
In Maryville, it is 6.73 degrees C.
```

https://bitbucket.org/professorcase/weathernode

W05

W05

A more detailed introduction can be found at:
https://openclassrooms.com/courses/ultra-fast-applications-using-node-js/creating-your-first-app-with-node-js

Or try the introduction at
http://nodeguide.com/beginner.html

Read and work through examples. Customize the message you send back and then customize an HTML response.

Exam 1
Friday

References

Express in Action: Writing, building, and testing Node.js applications by Evan Hahn

Web development with Node and Express by Ethan Brown

Felix Geisendörfer, *Node.js - As a networking tool*, LinkedIn SlideShare,
http://www.slideshare.net/the_undefined/nodejs-as-a-networking-tool

Felix Geisendörfer, *Node.js - A practical introduction (v2)*, LinkedIn SlideShare,
http://www.slideshare.net/the_undefined/nodejs-a-practical-introduction-v2

TutorialsPoint.com, *Node.js content*

Benjamin San Souci & Maude Lemaire, *Node.js*,
https://mcgill-csus.github.io/student_projects/NodeJSPresentation.pdf