Universidad del Valle

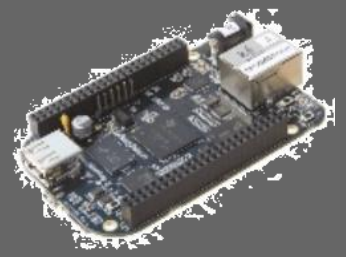
PSI

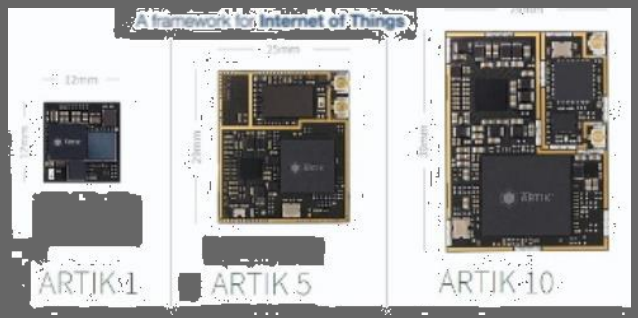
Percepción y Sistemas Inteligentes

Introduction to NodeJS

Professor:
Bladimir Bacca Cortes Ph.D.
Bladimir.bacca@correounivalle.edu.co

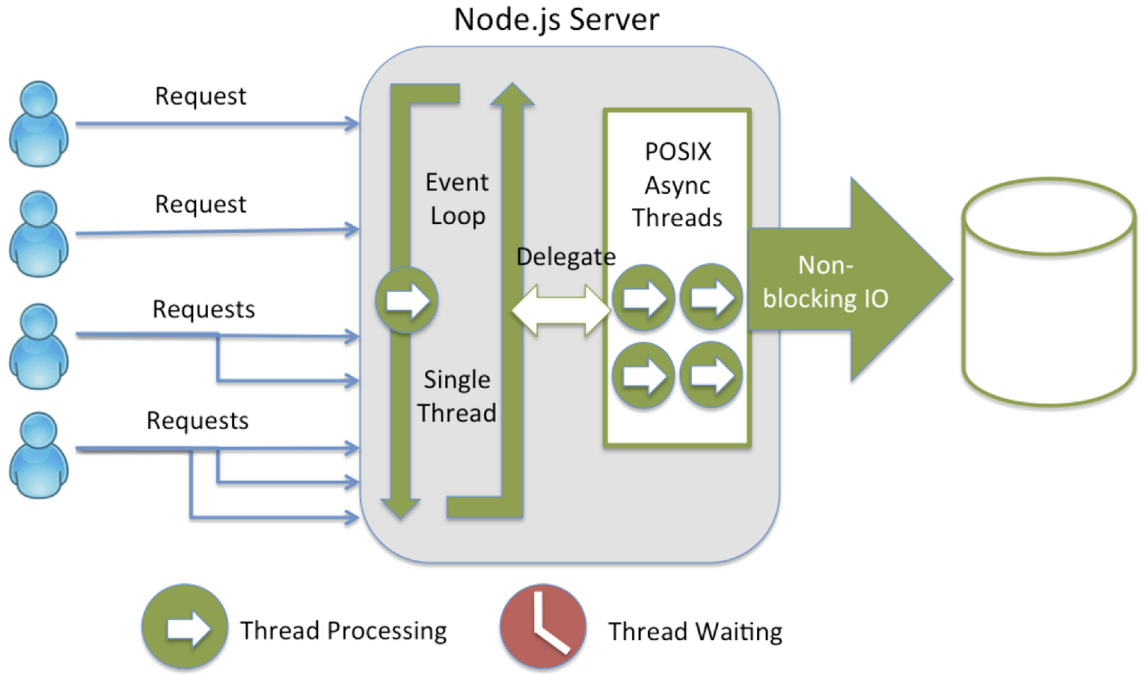
Grupo de Investigación en Percepción y Sistemas Inteligentes.





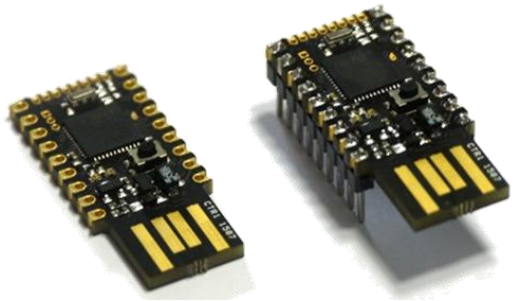
Contents

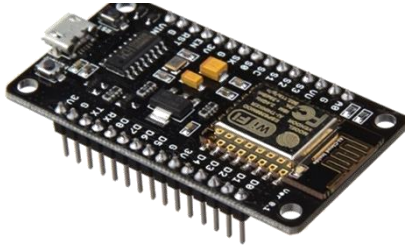
- Introduction.
- Environment setup.
- Architecture and First applications.
- REPL terminal
- NPM
- Callbacks and Events
- Buffers and Streams
- File System.
- Global objects
- Utility modules



Introduction

- **What is NodeJS?**
  - NodeJS is an open source , cross platform runtime environment for server side and networking application.
  - It is written in JavaScript and can run on Linux , Mac , Windows , FreeBSD.
  - It provided an event driven architecture and a non blocking I/O that optimize and scalability. These technology uses for real time application.
  - It used Google JavaScript V8 Engine to Execute Code.
  - V8 compiles JavaScript to native machine code (IA-32, x86-64, ARM, or MIPS ISAs) before executing it.
  - It is used by Groupon, SAP , LinkedIn , Microsoft,Yahoo ,Walmart ,Paypal





Beaglebone

Edison

Introduction

- **Why to use NodeJS?**
  - It is very lightweight and fast
  - Node.js was easy to configure
  - There are lots of modules available for free. For example, I found a Node.js module for PayPal.
  - NodeJS work with NoSQL as well
- **When not use NodeJS**
  - Your server request is dependent on heavy CPU consuming algorithm/Job.
  - Node.JS itself does not utilize all core of underlying system and it is single threaded by default, you have to write logic by your own to utilize multi core processor and make it multi threaded.

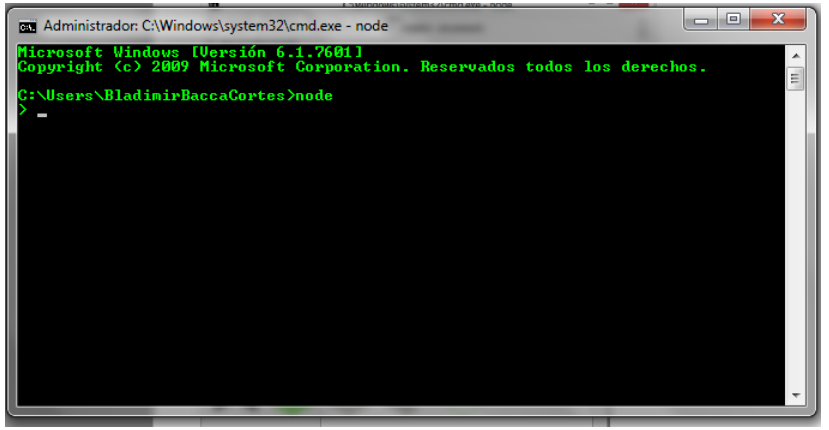
$$Node.JS = RuntimeEnvironment + JavaScriptLibrary$$

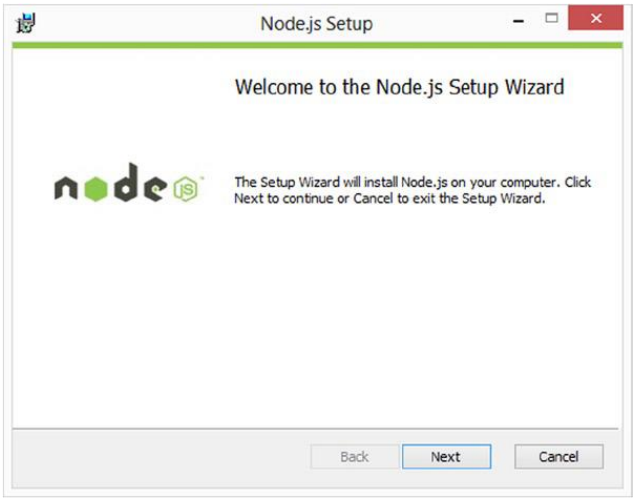
Introduction

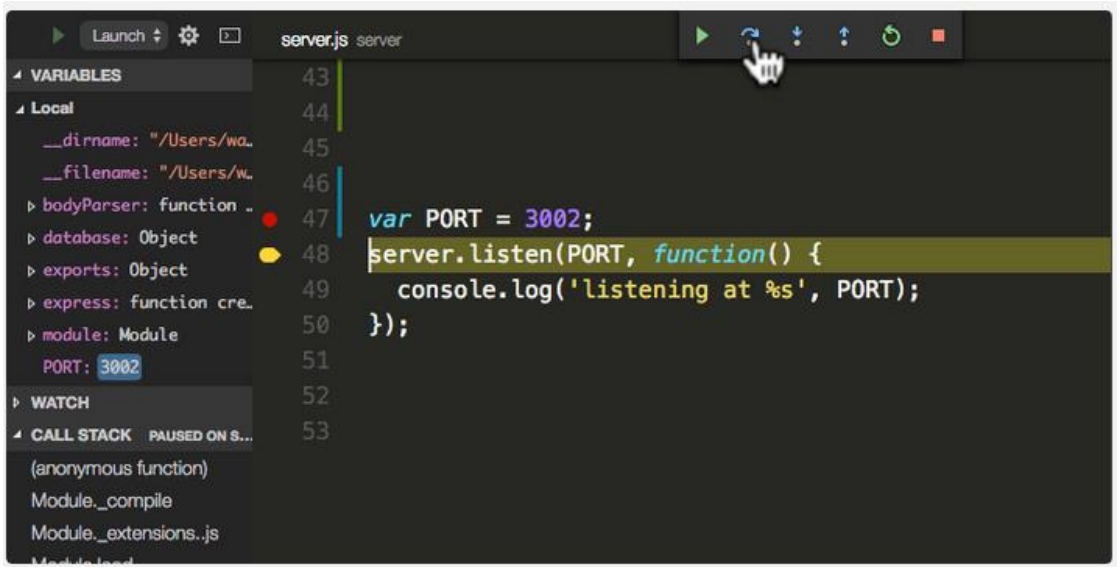
- **Features of NodeJS**
  - *Asynchronous and Event Driven* - All APIs of Node.js library are asynchronous, that is, non-blocking. It essentially means a Node.js based server never waits for an API to return data.
  - Node.js uses an *event-driven, non-blocking I/O model*, which makes it lightweight.
  - It makes use of *event-loops via JavaScript's callback* functionality to implement the non-blocking I/O
  - *Single Threaded but Highly Scalable* - Node.js uses a single threaded model with event looping
  - *No Buffering* - Node.js applications never buffer any data. These applications simply output the data in chunks
  - *License* - Node.js is released under the MIT license
  - In not-so-simple words Node.js is a *high performance network applications framework*, well optimized for high concurrent environments.

Environment Setup

- **Download**:
  - From https://nodejs.org/en/download/
- **Check the installation:**
  - Open a command terminal.
  - Type *node*.
- **IDE – Visual Studio Code**
  - Download: https://code.visualstudio.com/download

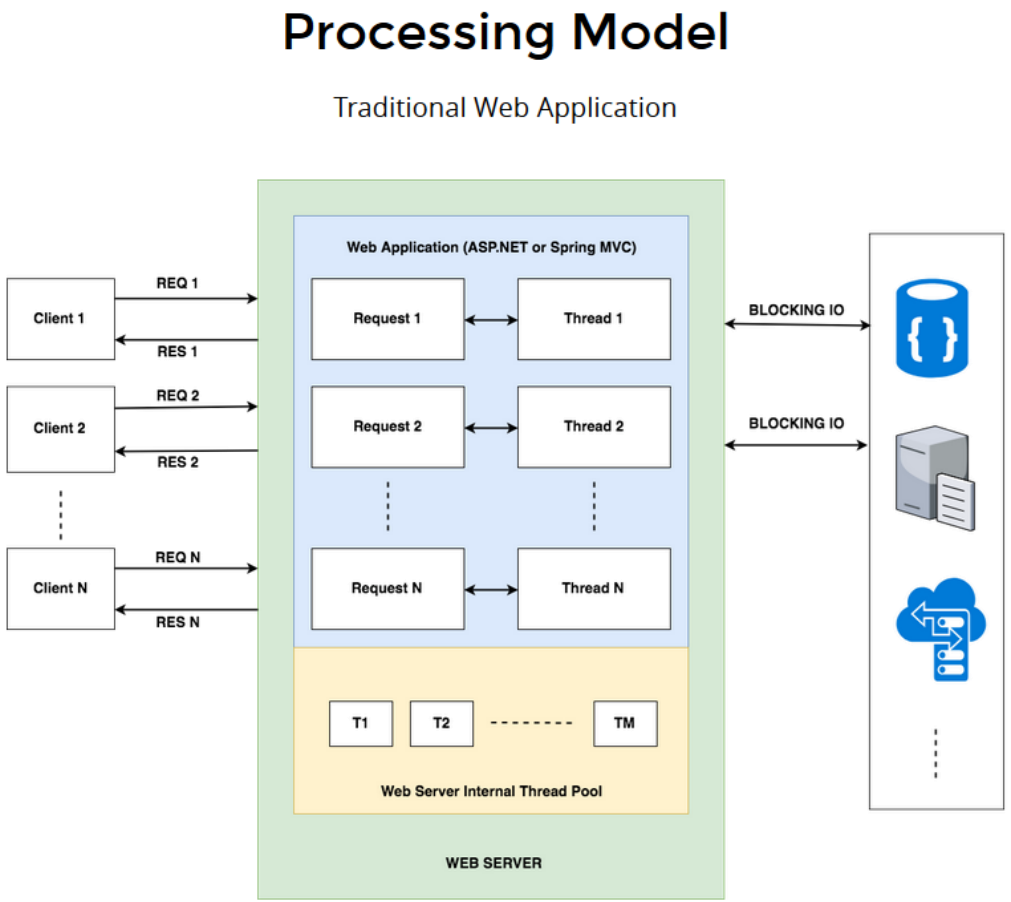


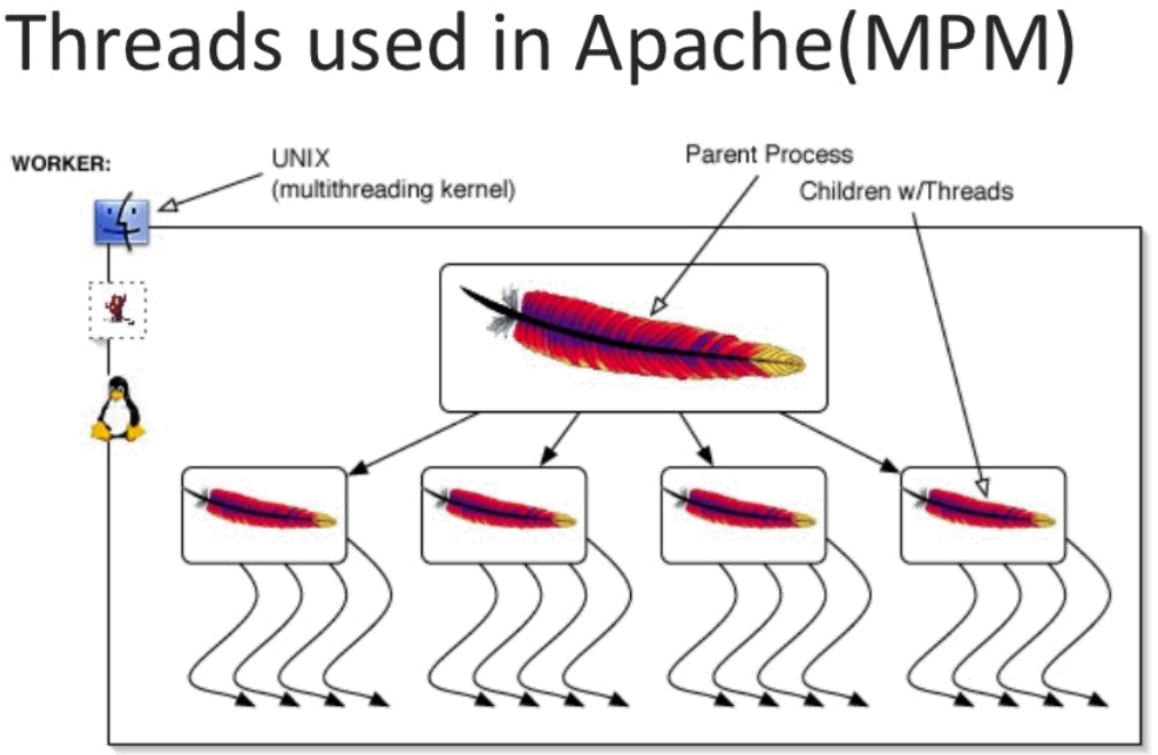




Architecture and First Applications

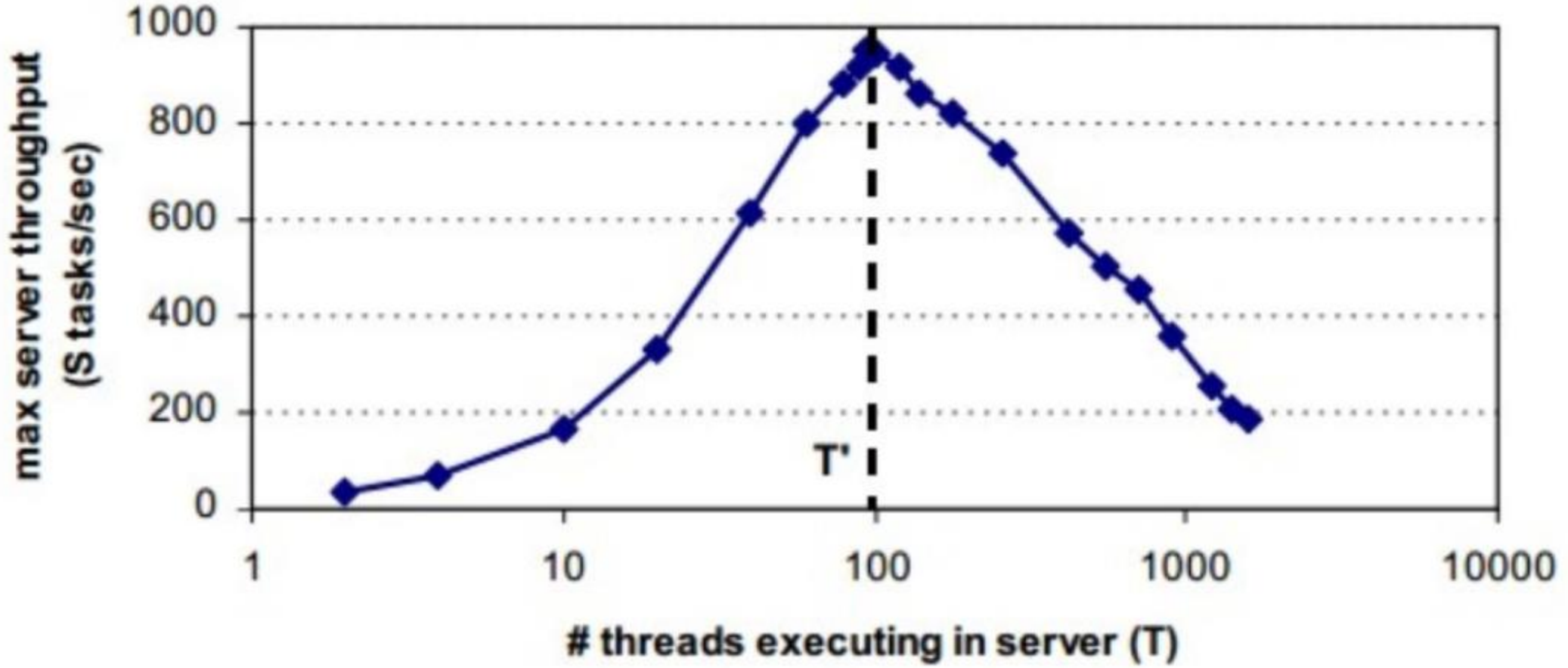
- **Architecture** – *Traditional processing model*





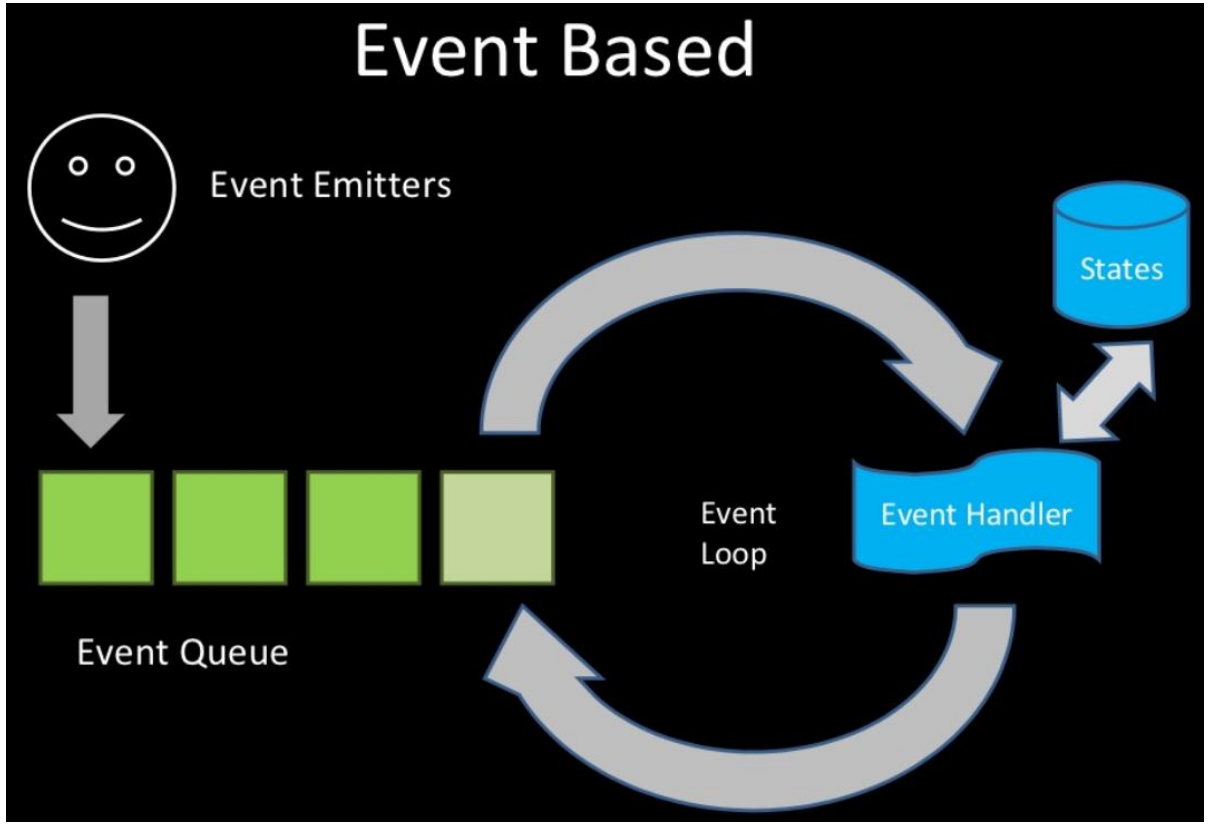
Architecture and First Applications

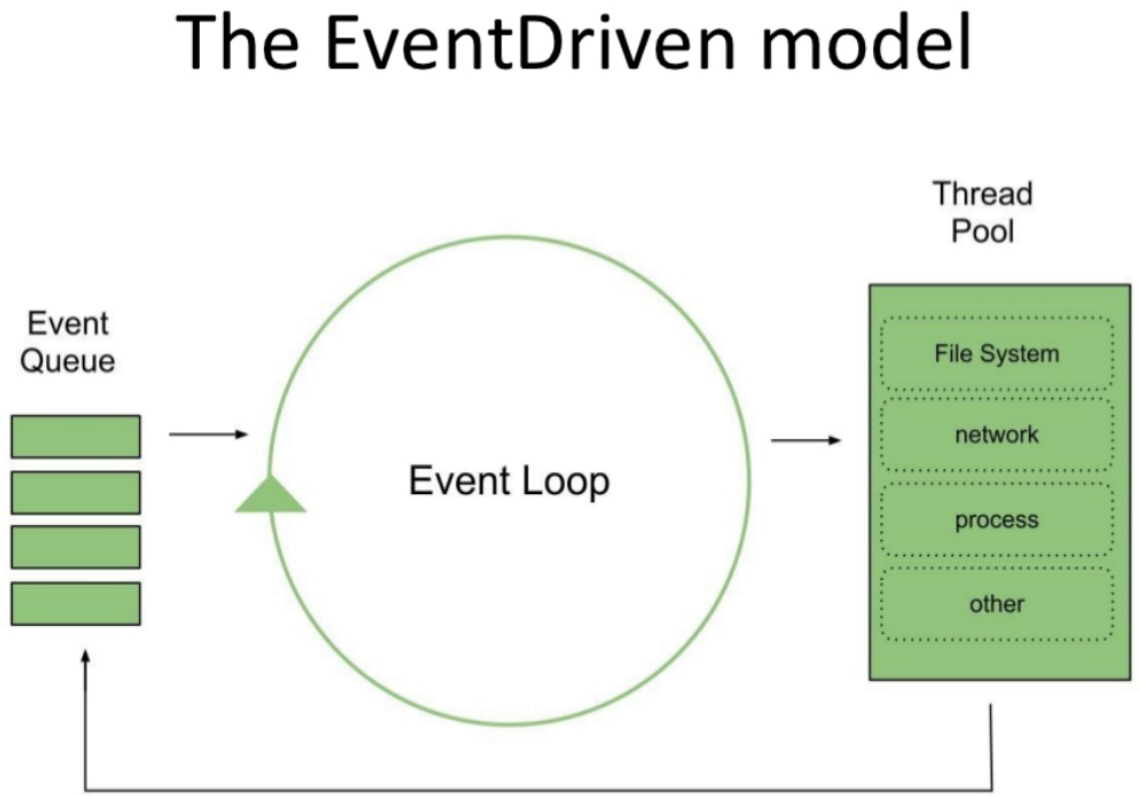
- **Architecture** – *Traditional processing model – Scalability Issues*



Architecture and First Applications

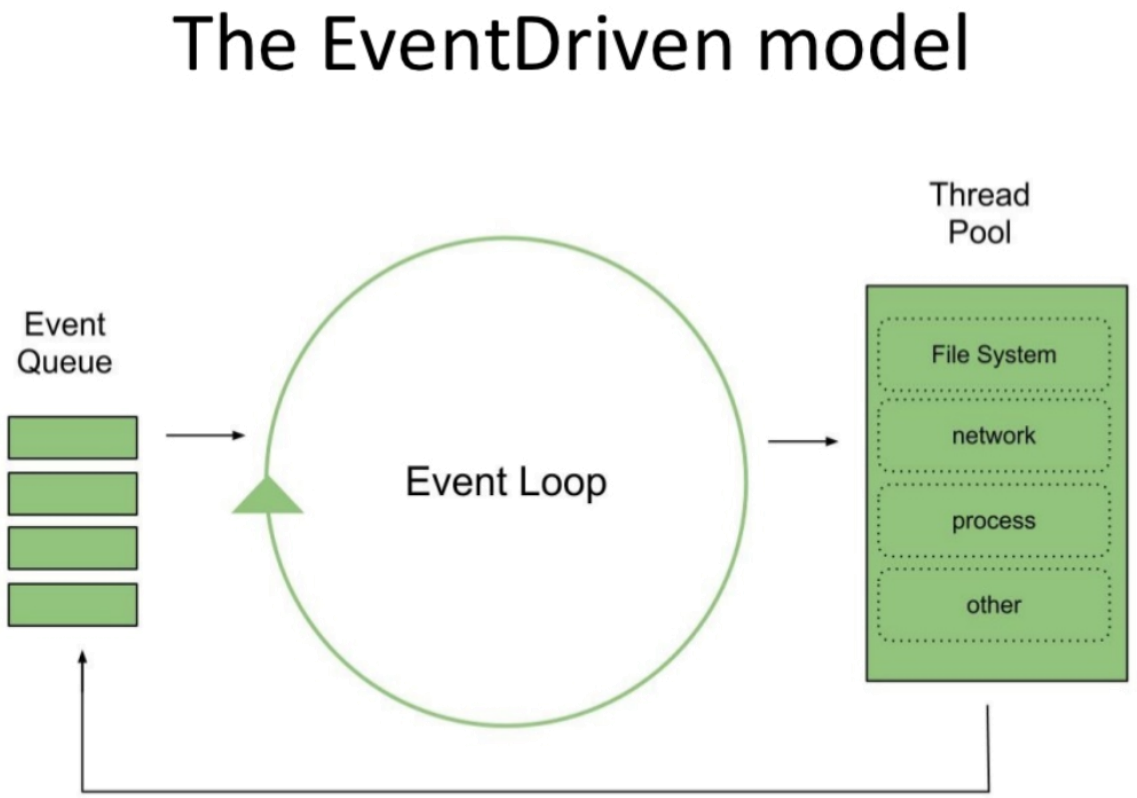
- **Architecture** – *NodeJS*

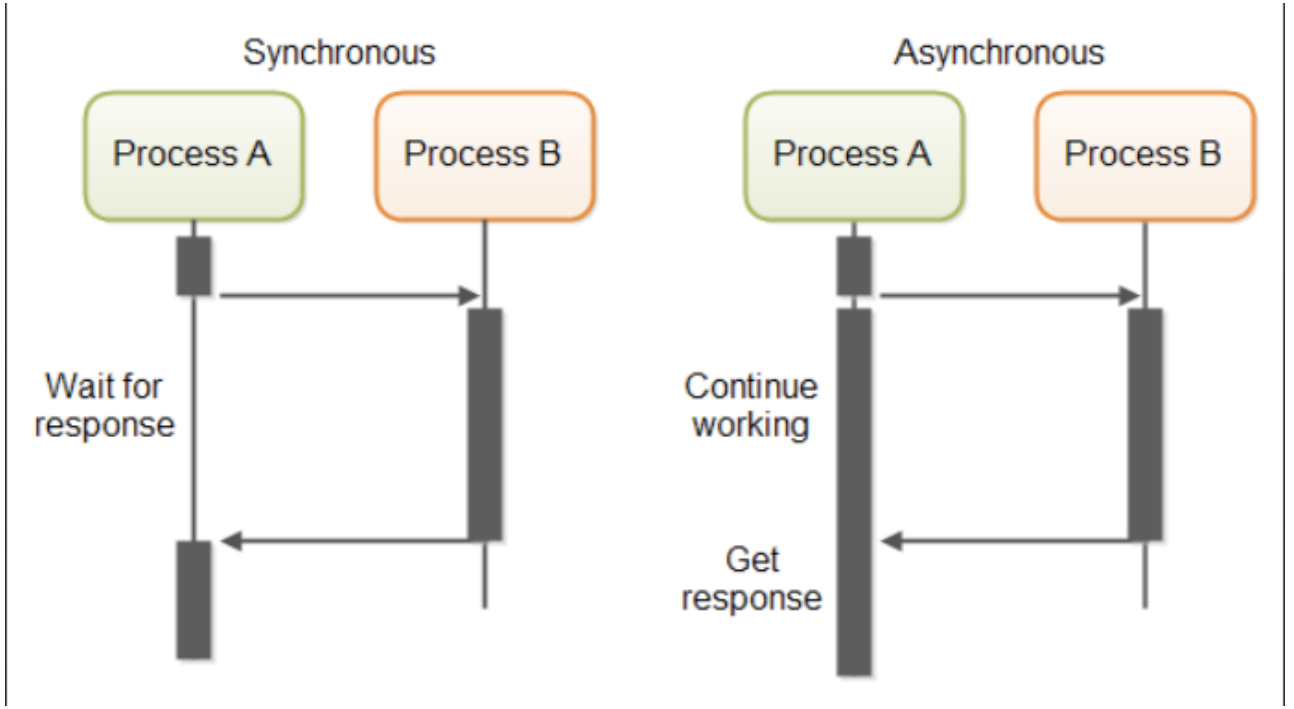




Architecture and First Applications

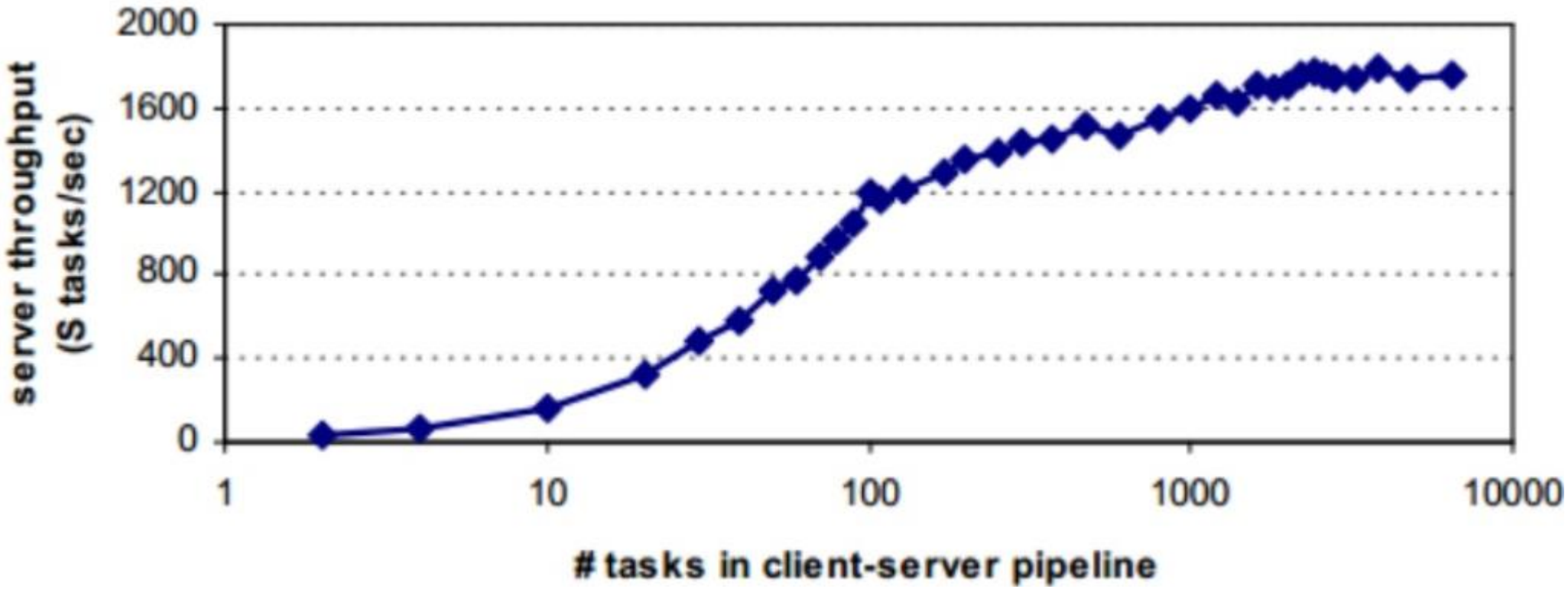
- **Architecture** – *NodeJS – Synchronous /Asynchronous*





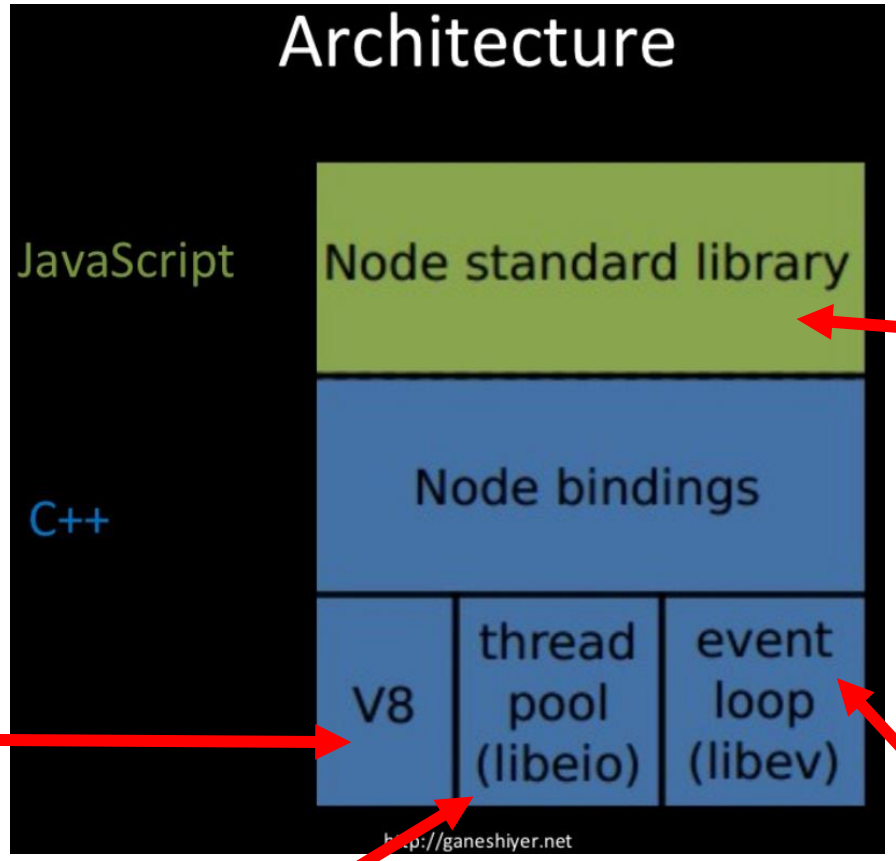
Architecture and First Applications

- **Architecture** – *NodeJS – Scalability Issues*



Architecture and First Applications

- **Architecture** – *NodeJS*



Exposes OS features such as file handling, sockets, etc. Functionality for HTTP, DNS etc.

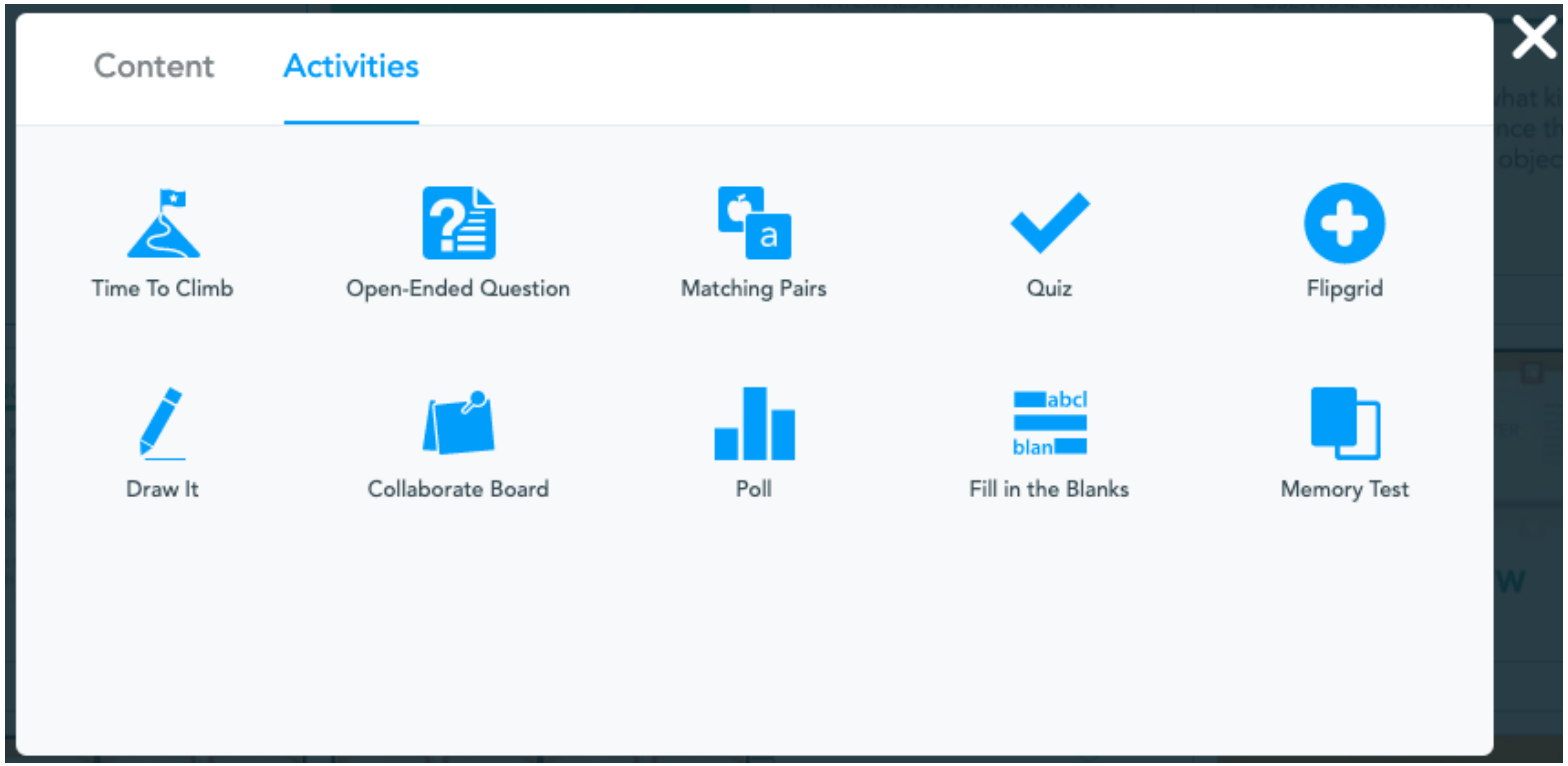
JavaScript Engine

Event loop

Event-based fully asynchronous library

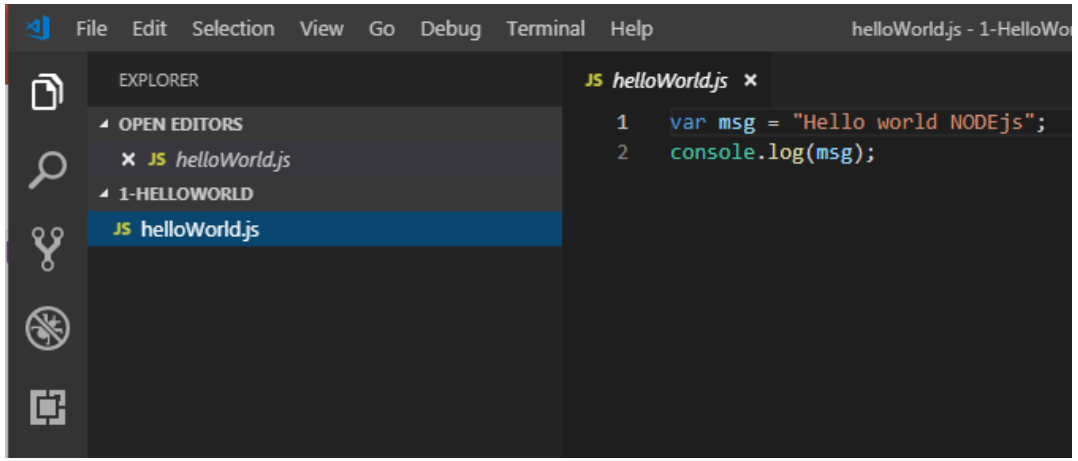
Nearpod Activity

- Please go to the Nearpod link shared in the chat.
- Fulfil the Nearpod activity.
- Analyze the results with your teacher.

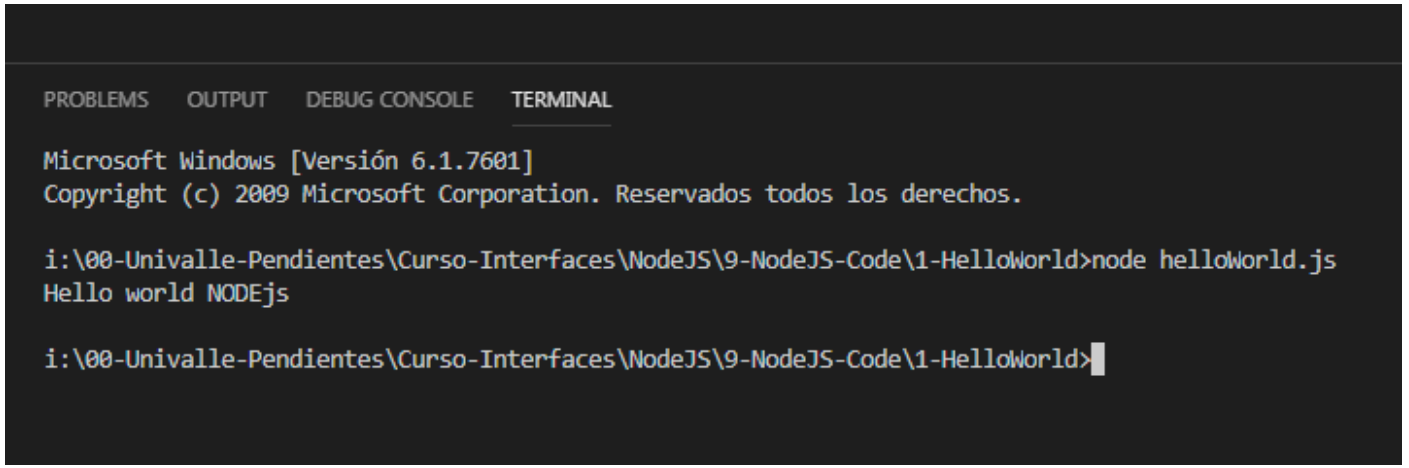


Architecture and First Applications

- **First Application – *helloWorld.js***
  - Open Visual Studio Code.
  - Press the *Explorer* icon.
  - Press the *Open Folder* button, and selects the *1-HelloWorld* folder.

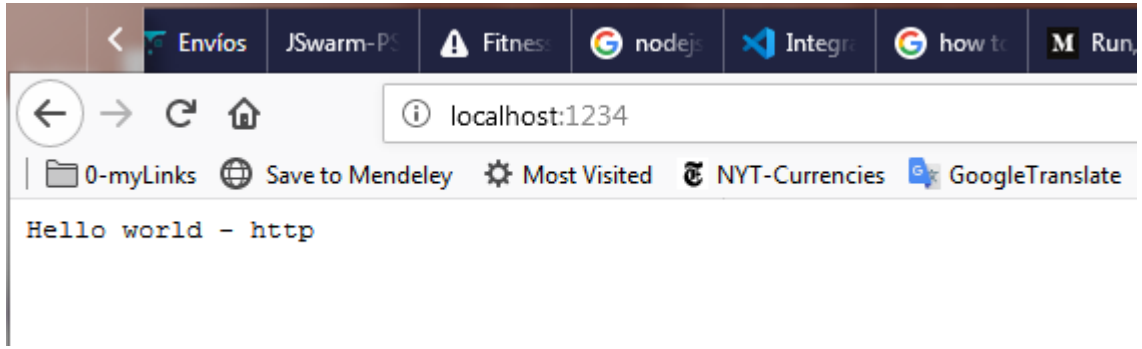


- To execute it, press right button of mouse on the *helloWorld.js* file.
- Then, select *Open in Terminal* option and command line session is opened in the Workspace path.
- Then, type *node helloWorld.js*



Architecture and First Applications

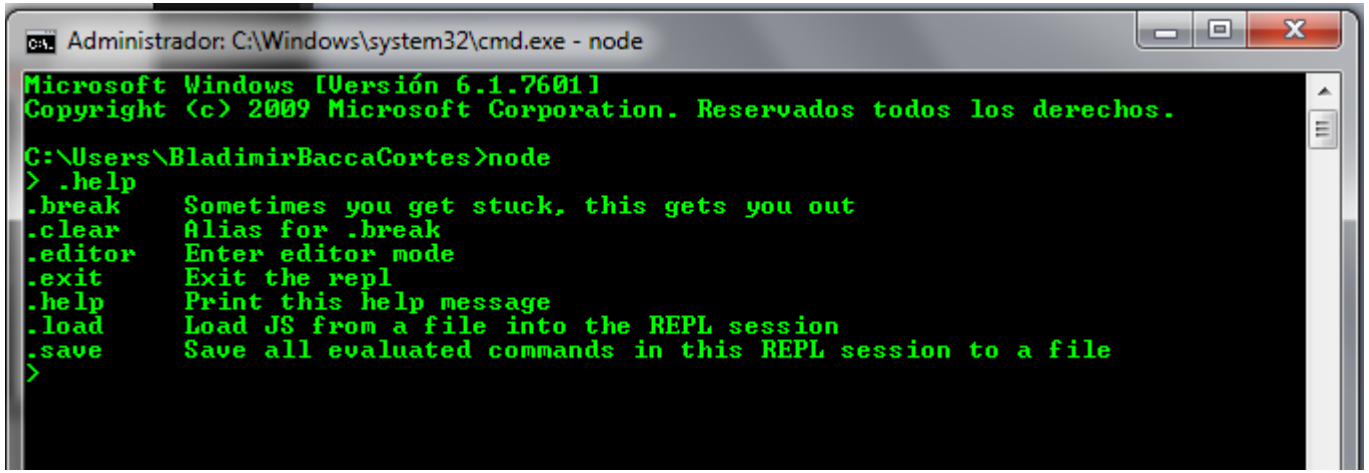
- **First Application – *main.js***
  - Open Visual Studio Code.
  - Press the *Explorer* icon.
  - Press the *Open Folder* button, and selects the 2-*SimpleServer* folder.




```
JS main.js
1  var http = require("http");
2
3  http.createServer(
4      function (request, response)
5      {
6          // Send the HTTP header
7          // HTTP Status: 200 : OK
8          // Content Type: text/plain
9          response.writeHead(200, {'Content-Type': 'text/plain'});
10
11         // Send the response body as "Hello World"
12         response.end("Hello world - http \n");
13     }
14 ).listen(1234);
15
16 // Console message
17 console.log("Hello server running at localhost:1234 ...");
18

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\2-SimpleServer>node main.js
Hello server running at localhost:1234 ...
```

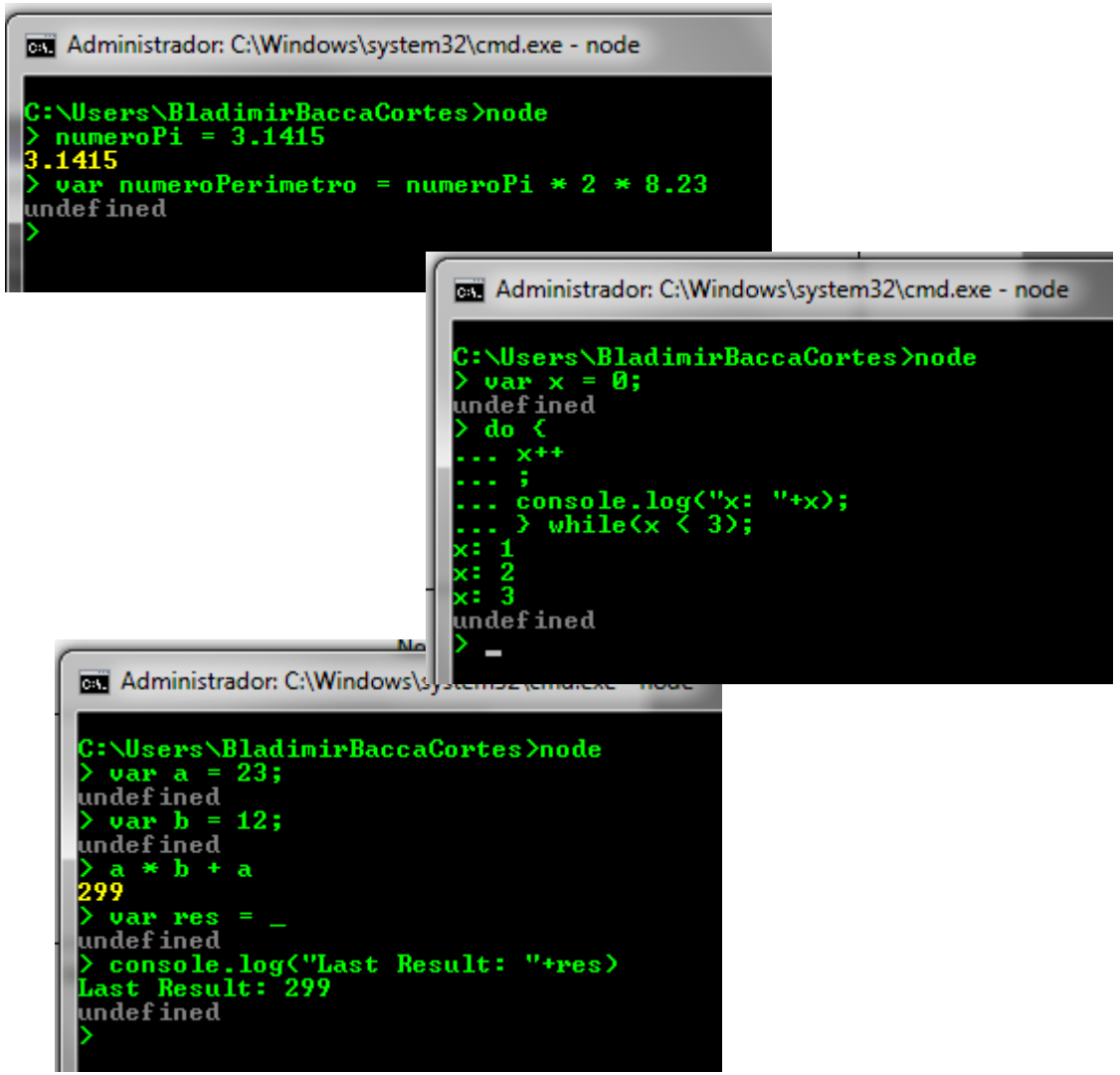
REPL Terminal

- **REPL** stands for Read Eval Print Loop and it represents a computer environment like a Windows console or Unix/Linux shell where a command is entered and the system responds with an output in an interactive mode.
- **Tasks**:
  - **Read** - Reads user's input, parses the input into JavaScript data-structure, and stores in memory.
  - **Eval** - Takes and evaluates the data structure.
  - **Print** - Prints the result.
  - **Loop** - Loops the above command until the user presses ctrl-c twice.
- **How to invoke it?**
  - Open a command terminal.
  - Type *node*.



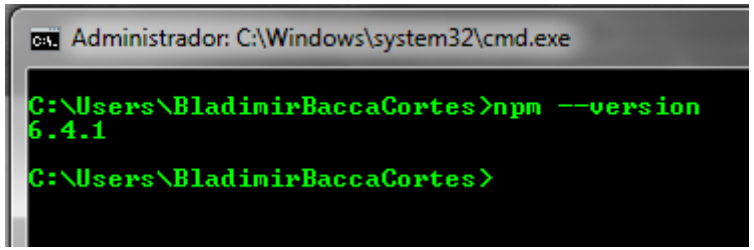
REPL Terminal

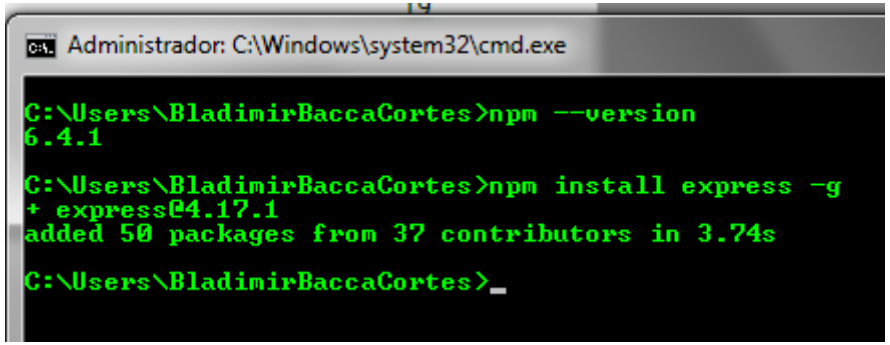
- **Some examples**
  - **Variables**: if *var* keyword is used, variables are not printed.
  - **Multi-line**: *Node* checks continuity of expressions, and then it adds '…'
  - **Last result**: it is stored in a variable named '_'
- **Other features:**
  - ctrl + c - terminate the current command.
  - 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 the current Node REPL session to a file.
  - .load filename - load file content in current Node REPL session

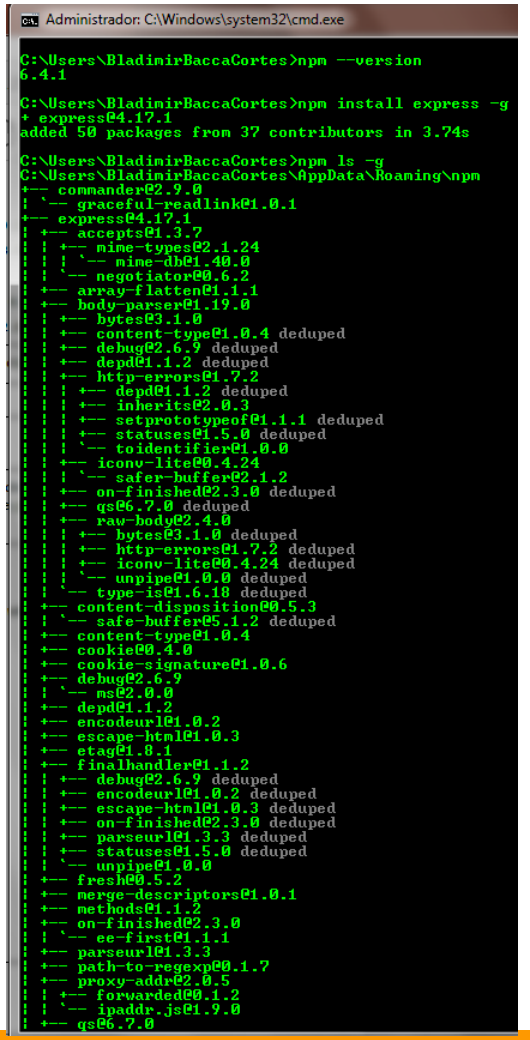


NPM – Package Manager

- **Functionalities**:
  - Online repositories for node.js packages/modules which are searchable on http://search.nodejs.org
  - Command line utility to install Node.js packages, do version management and dependency management of Node.js packages.
- **How to use it?**
  - Open a command terminal
  - Type *npm –version*
- **How to install modules?**
  - Type: *npm install <MODULE_NAME>*
  - Example: *npm install express -g*
  - *Local installation* (without –g option): modules will be installed in the current folder solution.
    - JS files: “var express = require(‘express’);”
  - *Global installation* (with –g option): modules will be installed in system folder.
- **Checking global installed modules:**
  - *npm ls -g*

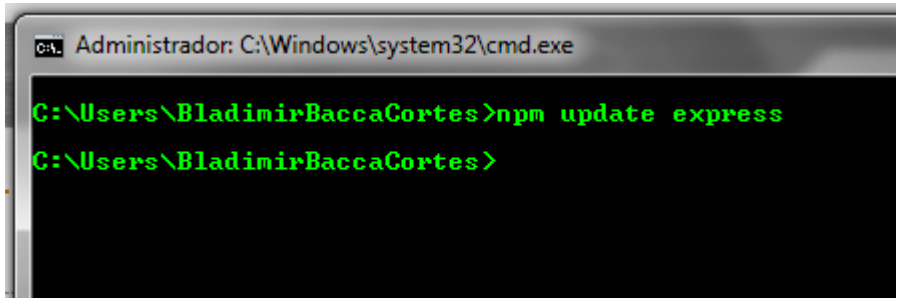


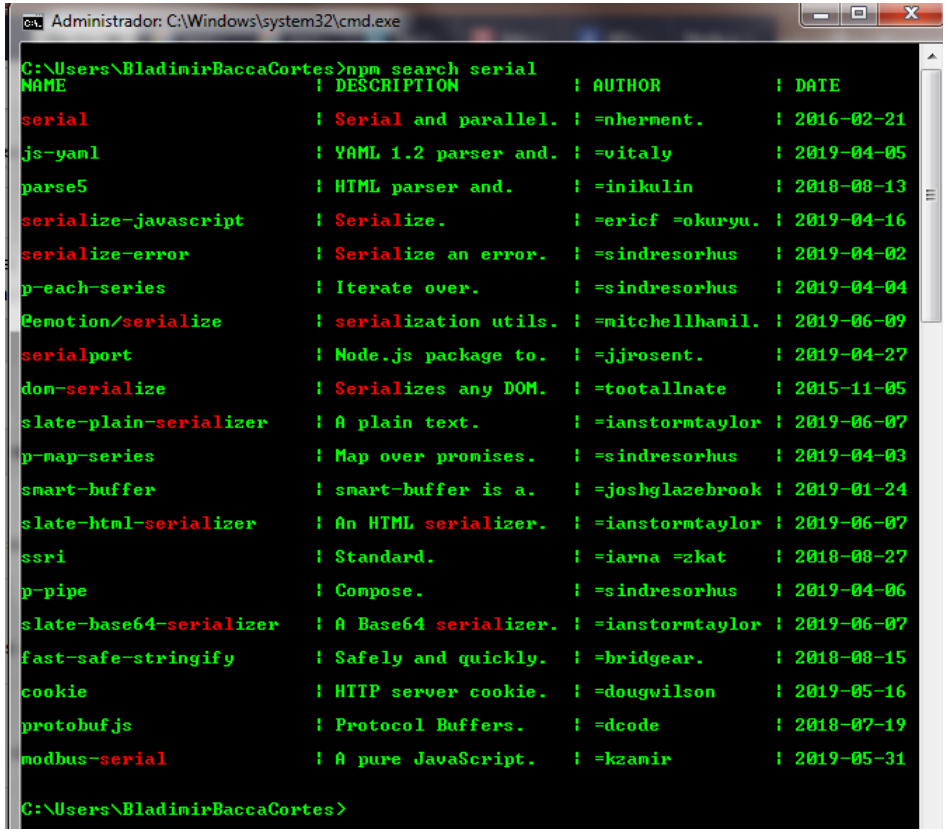




NPM – Package Manager

- **Other commands:**
  - Updating modules:
    - Type: *npm update <MODULE_NAME>*
  - Uninstalling modules:
    - Type: *npm uninstall <MODULE_NAME>*
  - Search for a module:
    - Type: *npm search <MODULE_NAME>*





Variables in NodeJS

- Variables are defined in JavaScript using the **var** keyword
- They support all types of variables without specifying it on the declaration.
- **They support**
  - Numbers
  - Arrays
  - Booleans
  - Object literals
  - Functions.

- Open folder: ***3-BasicNodeJS***
- Observe and run: ***checkVarialbes.js***

checkVariables.js

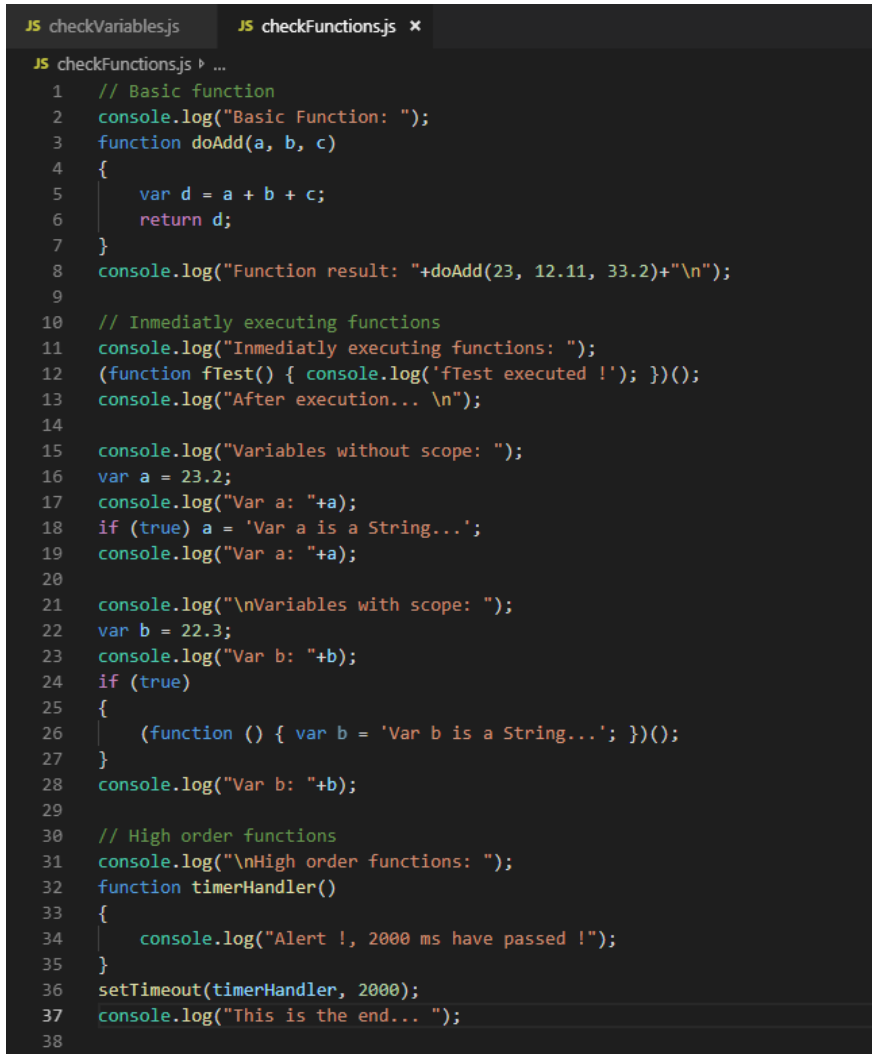
```
// Numbers
console.log("Operations with numbers: ");
var a = 23.34;
var b = 0.234;
console.log("Sum: "+(a + b));
console.log("Mult.: "+(a * b));
console.log("Division: "+(a / (b*10)));
console.log("**********************");

// Booleans
console.log("Operations with booleans: ");
var x = true;
var y = false;
console.log("x * y: "+(x && y));
console.log("x + y: "+(x || y));
console.log("Not x: "+(!x));
console.log("**********************");

// Arrays
console.log("Arrays: ");
var r = [1, 2.3, 4.3];
console.log("r: "+r+"\n");

r.push(5.3);
console.log("Pushing ... r: "+r+"\n");

var lastR = r.pop();
console.log("Pop ... r: "+r);
console.log("Last R: "+lastR+"\n");

r.unshift(0.2);
console.log("Unshifting ... r: "+r+"\n");

var firstR = r.shift();
console.log("Shifting ... r: "+r);
console.log("First R: "+firstR+"\n");

var delItem = r.splice(2, 1);
console.log("Splice ... r: "+r+"\n");
```

Functions in NodeJs

- **Syntaxis**:

```
function functionName (parameter_list)
{
  // Function body
  return expression; // optional
}
```

- Most of the power of JavaScript comes from the way it handles the function type.
- All functions return a value in JavaScript. In the absence of an explicit return statement, a function returns undefined.
- **Immediately executing functions**: The reason for having an immediately executing function is to create a new variable scope. An if, else, or while does not create a new variable scope in JavaScript.
- **Anonymous functions**: A function without a name is called an anonymous function. In JavaScript, you can assign a function to a variable. If you are going to use a function as a variable, you don't need to name the function
- **High-order functions**: Functions that take functions as arguments are called higher-order functions.
- Run ***checkFunctions.js***, in the ***3-BasicsNodeJS*** folder.

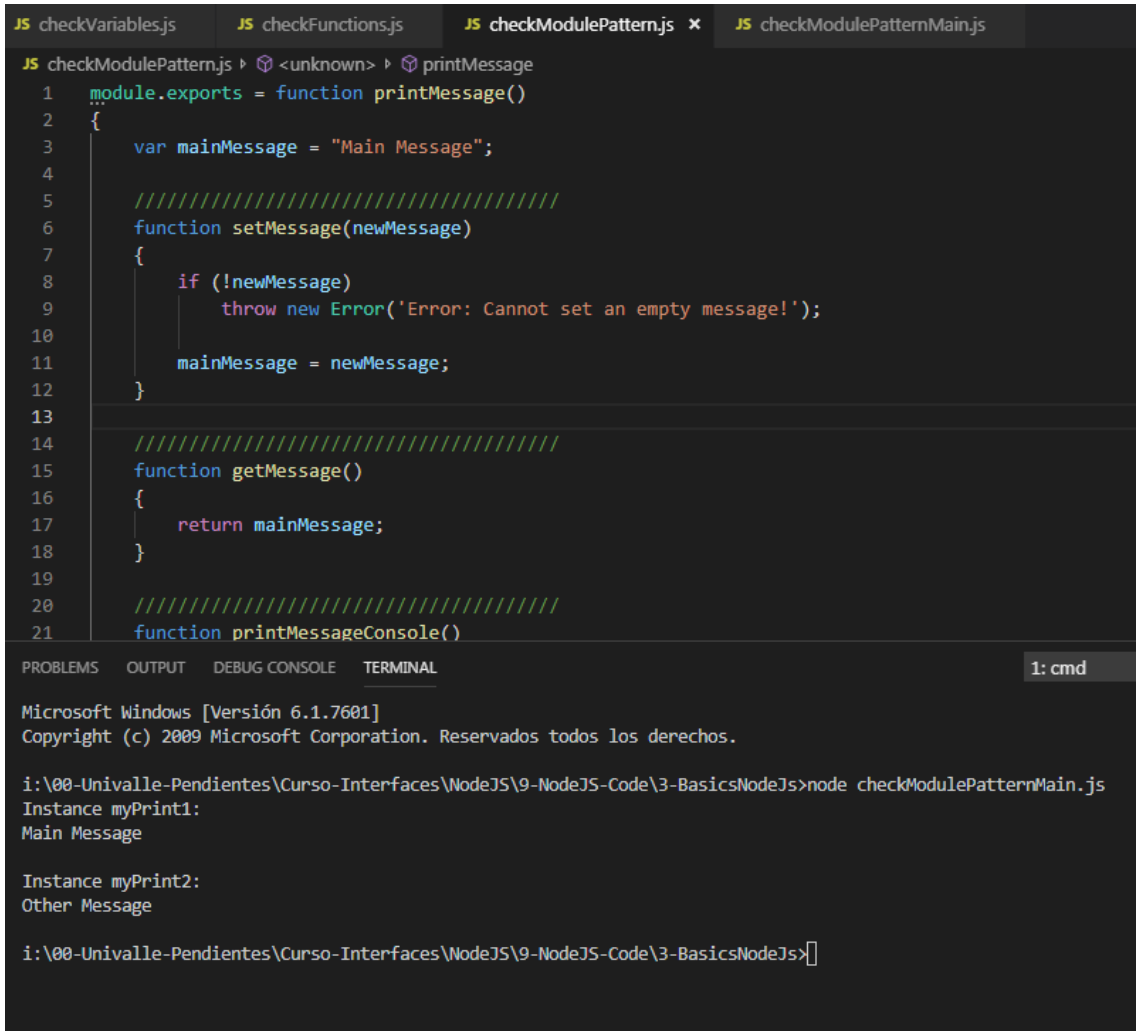


Module Pattern and Exports

- Functions that return objects are a great way to create similar objects.
- An object here means data and functionality bundled into a nice package, which is the most basic form of Object Oriented Programming (OOP) that one can do.
- At the heart of the revealing module pattern is JavaScript's support for closures and ability to return arbitrary (function + data) object literals.

- **Module system properties**:
  - Each file is its own module.
  - Each file has access to the current module definition using the module variable.
  - The export of the current module is determined by the module.exports variable.
  - To import a module, use the globally available require function

- Open and run ***checkModulePattern.js***



Callbacks and Events

- **Definition of Callback**: It is an asynchronous equivalent for a function. A callback function is called at the completion of a given task. Node makes heavy use of callbacks. All the APIs of Node are written in such a way that they support callbacks.

- **For example:**
  - A function to read a file may start reading a file and return the control to the execution environment immediately so that the next instruction can be executed.
  - Once file I/O is complete, it will call the callback function while passing the callback function, the content of the file as a parameter.
  - So there is no blocking or wait for File I/O.
  - This makes Node.js highly scalable, as it can process a high number of requests without waiting for any function to return results.

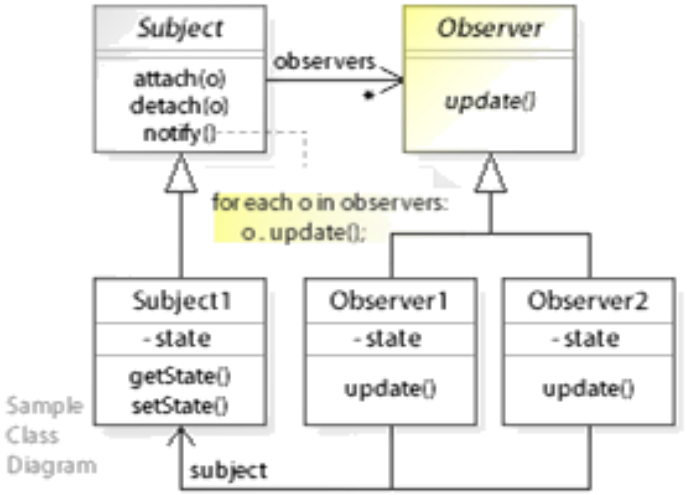
- Open folder **4-CallBacks**, and run **fsBlocking.js** and **fsNoBlocking.js**.

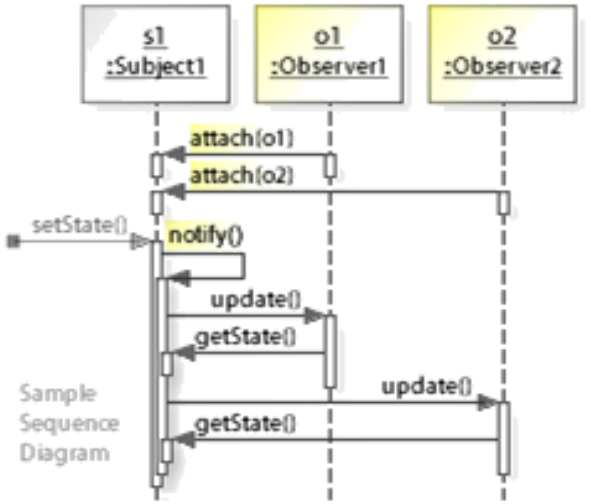
```
PROBLEMS 1  OUTPUT  DEBUG CONSOLE  TERMINAL

Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\4-CallBacks>node fsBlocking.js
Start reading...
Interfaces Course.

NodeJS class Introduction.

Nice things are easy!!


End...

i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\4-CallBacks>
```

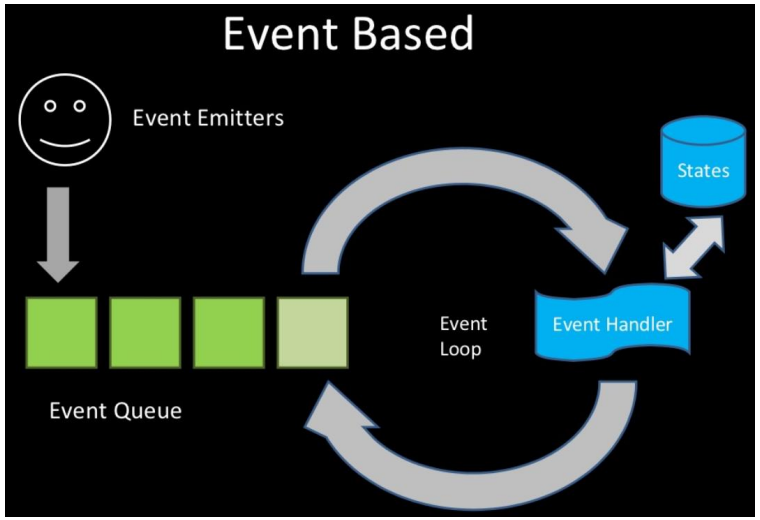
```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\4-CallBacks>node fsNoBlocking.js
Start reading...
End ...
Interfaces Course.

NodeJS class Introduction.

Nice things are easy!!


i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\4-CallBacks>
```

Callbacks and Events

- Node.js is a single-threaded application, but it can support concurrency via the concept of **event** and **callbacks.**
- Every API of Node.js is asynchronous and being single-threaded, they use **async function calls** to maintain concurrency.
- **Node uses observer pattern**: It is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods.
- As soon as Node starts its server, it simply initiates its variables, declares functions, and then simply waits for the event to occur.
- In an event-driven application, there is generally a main loop that listens for events, and then triggers a callback function when one of those events is detected.
- Although events look quite similar to callbacks, the difference lies in the fact that callback functions are called when an asynchronous function returns its result, whereas event handling works on the observer pattern. The functions that listen to events act as Observers







Callbacks and Events

- **Steps to implement events:**

1. Load events module: *events*.
2. Instance emitter event class: *EventEmitter()*
3. Attaching event handler with events.
4. Firing events.

- Open folder **5-Events**, then run **testEvents.js**.

- All objects which emit events are the instances of *events.EventEmitter*.
- When an EventEmitter instance faces any error, it emits an 'error' event.
- When a new listener is added, 'newListener' event is fired
- When a listener is removed, 'removeListener' event is fired.

testEvents.js

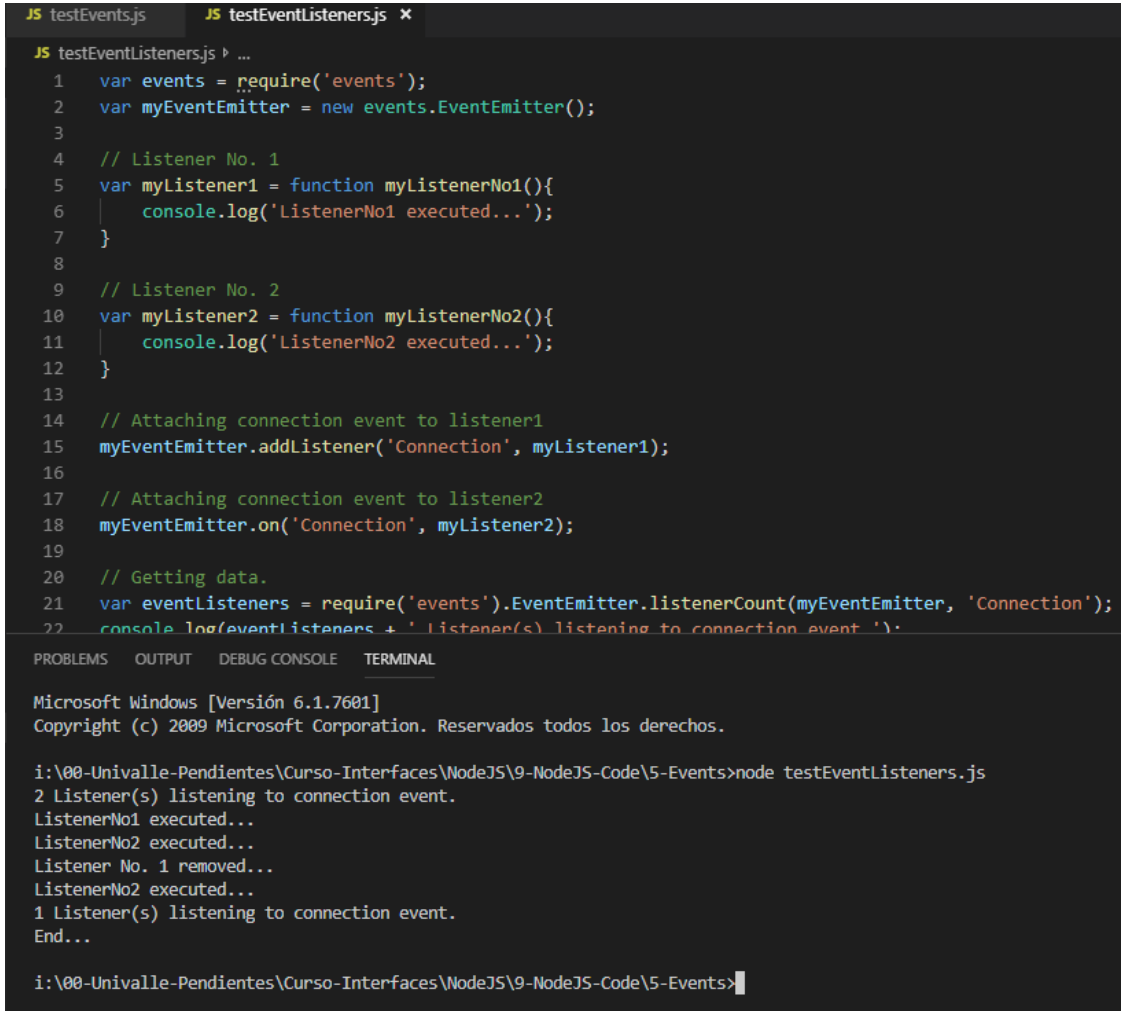
```
// Import module of events
var events = require('events');

// Create event emitter
var myEventEmitter = new events.EventEmitter();

// Create event handler
var myConnectHandler = function connectHandler() {
    console.log('Ok - Connection succesful !');

    // Fire data received by event.
    myEventEmitter.emit('Data_Received');
}

// Attaching the connection event with the handler
myEventEmitter.on('connection', myConnectHandler);

// Attaching Data_received event with an anonymous function
myEventEmitter.on('Data_Received', function(){
    console.log('Data received successfully !');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\5-Events>node testEvents.js
Ok - Connection succesful !
Data received successfully !
End...

i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\5-Events>
```

Callbacks and Events

- **Event emitter methods:**
  - *addListener(event, listener):* Adds a listener at the end of the listeners array for the specified event
  - *on(event, listener):* Adds a listener at the end of the listeners array for the specified event. But it cannot be removed.
  - *once(event, listener):* Adds a one-time listener to the event.
  - *removeListener(event, listener):* Removes a listener from the listener array for the specified event.
  - *removeAllListeners([event]):* Removes all listeners, or those of the specified event.
  - *setMaxListeners(n)* By default, EventEmitters will print a warning if more than 10 listeners are added for a particular event.
  - *listeners(event)* Returns an array of listeners for the specified event.



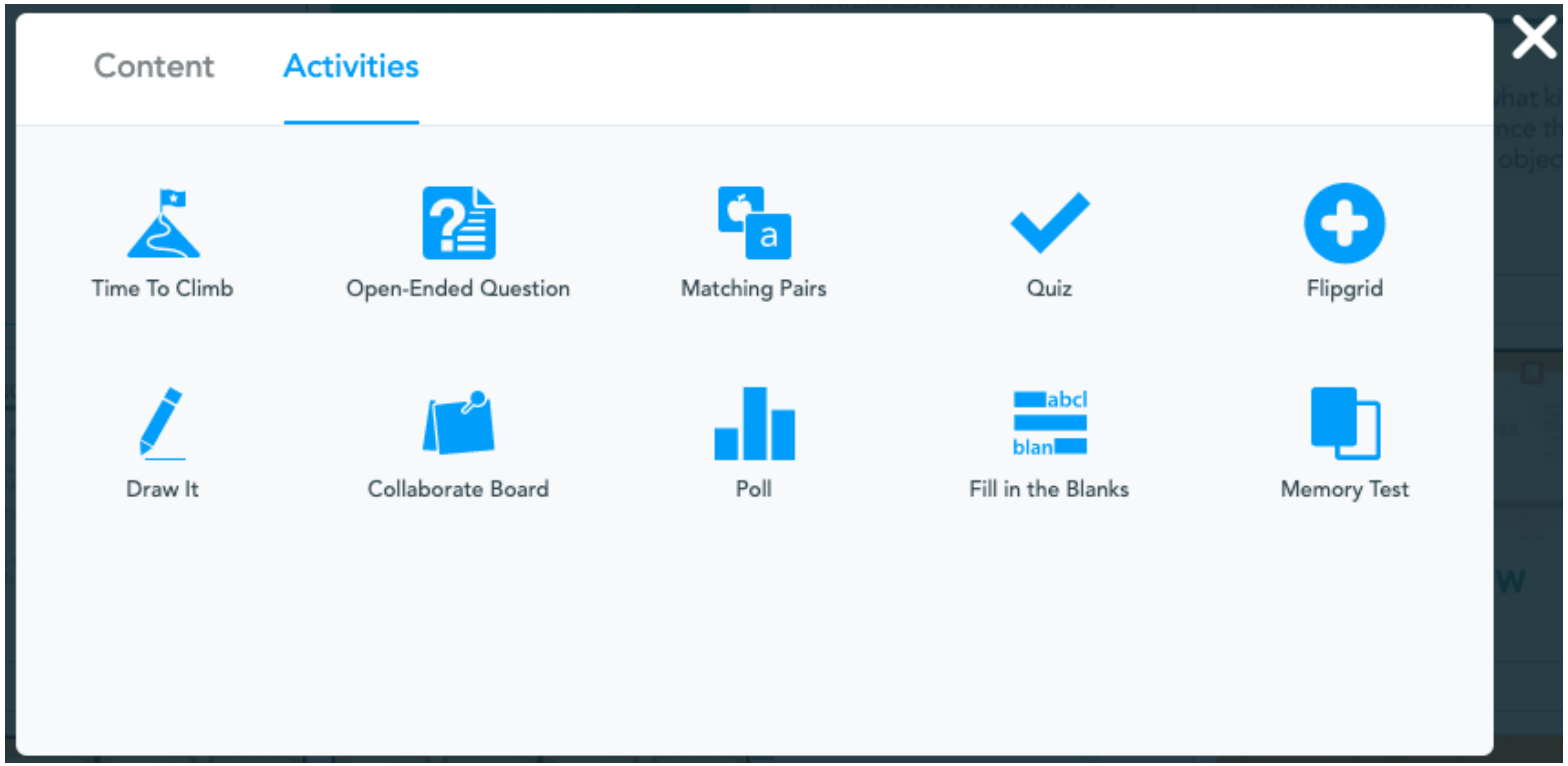
NodeJS Modules

- **Core modules**

- Assertion Testing
- File System
- Buffer
- HTTP/HTTPS
- C/C++ Addons
- Net
- Child Processes
- OS
- Cluster
- Path
- Crypto
- Process
- Debugger
- Punycode
- DNS
- Query Strings
- Domain
- REPL
- Events
- Stream
- String Decoder
- Timers
- TLS/SSL
- TTY
- UDP/Datagram
- URL
- Utilities
- VM
- ZLIB

Nearpod Activity

- Please go to the Nearpod link shared in the chat.
- Fulfil the Nearpod activity.
- Analyze the results with your teacher.



Buffers and Streams

- **Buffers**:

- While dealing with TCP streams or the file system, it's necessary to handle octet streams.

- Node provides Buffer class which provides instances to store raw data similar to an array of integers but corresponds to a raw memory allocation outside the V8 heap

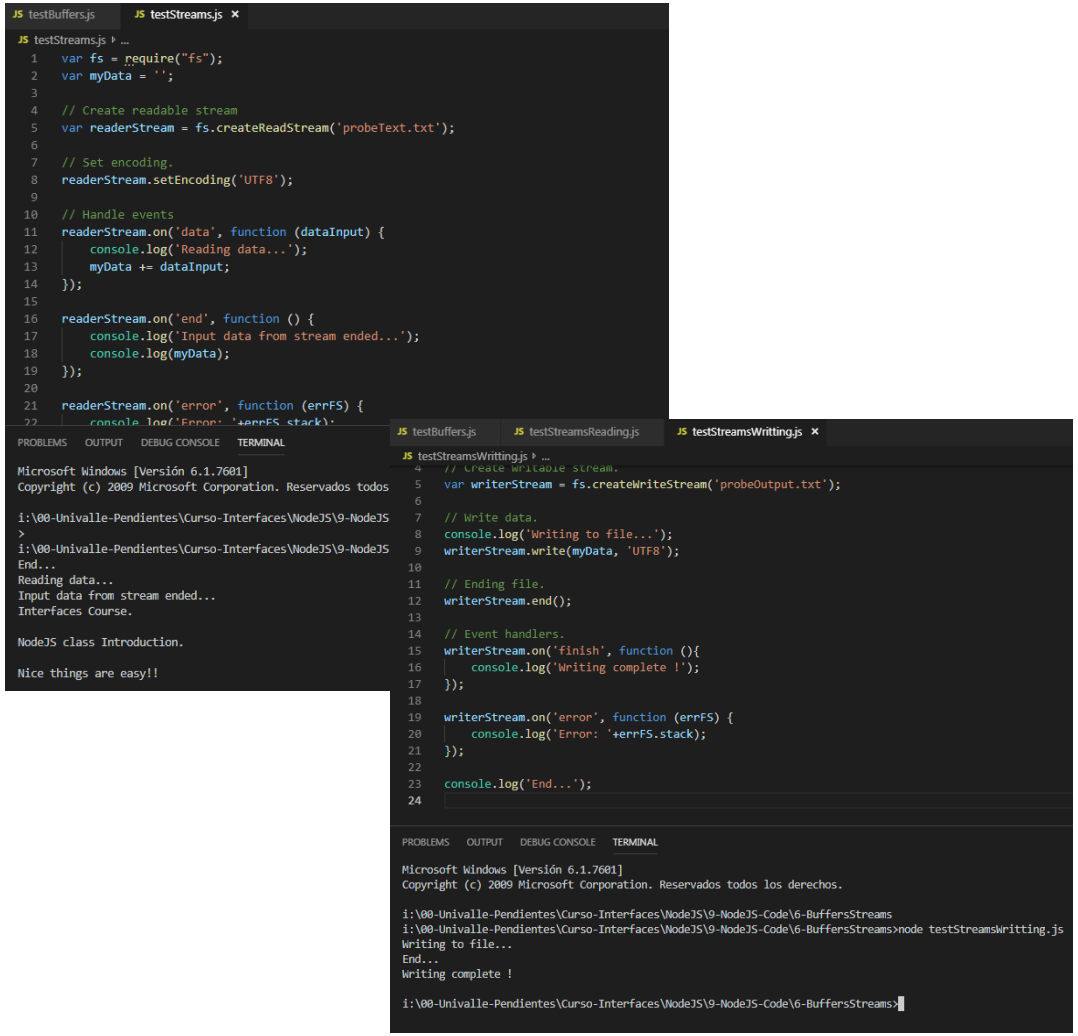
- Buffer class is a global class that can be accessed in an application without importing the buffer module.

- Open **6-BuffersStreams** folder, and run **testBuffers.js**.

Buffers and Streams

- **Streams**:

- Streams are objects that let you read data from a source or write data to a destination in continuous fashion. In Node.js, there are four types of streams:
  - **Readable** – It is used for read operation.
  - **Writable** – It is used for write operation.
  - **Duplex** – It can be used for both read/write operation.
  - **Transform** – A type of duplex stream where the output is computed based on input.
- Each type of Stream is an **EventEmitter** instance and throws several events at different instance of times:
  - **data** – It is fired when there is data is available to read.
  - **end** – It is fired when there is no more data to read.
  - **error** – It is fired when there is any error receiving or writing data.
  - **finish** – It is fired when all the data has been flushed to underlying system.
- Open **6-BuffersStreams** folder, and run **testStreamsReading.js** and **testStreamsWritting.js**.



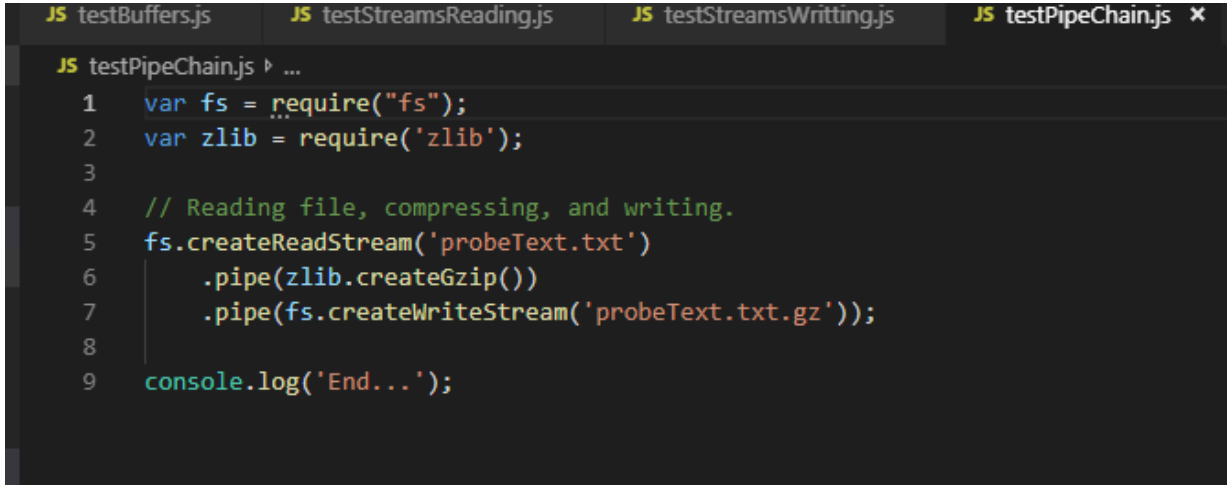
Buffers and Streams

- **Piping and Channing:**

- Piping is a mechanism where we provide the output of one stream as the input to another stream.
- It is normally used to get data from one stream and to pass the output of that stream to another stream.
- There is no limit on piping operations

- Chaining is a mechanism to connect the output of one stream to another stream and create a chain of multiple stream operations.
- It is normally used with piping operations

- Open **6-BufferStream** folder, and run **testPipeChain.js**.



File System

- Every method in the fs module has synchronous as well as asynchronous forms.
- Asynchronous methods take the last parameter as the completion function callback and the first parameter of the callback function as error

- **Methods**:

- *Open files*: fs.open(path, flags[, mode], callback)
- *File info*: fs.stat(path, callback)
- *Writing*: fs.writeFile(filename, data[, options], callback)
- *Reading*: fs.read(fd, buffer, offset, length, position, callback)
- *Closing*: fs.close(fd, callback)
- *Create directory*: fs.mkdir(path[, mode], callback)
- *Read directory*: fs.readdir(path, callback)

- Open folder **7-FileSystem**, and run **testFS.js**.

File System

- **Read and write flags (open):**

- r - Open file for reading.
- r+ - Open file for reading and writing
- rs - Open file for reading in synchronous mode
- rs+ - Open file for reading and writing, asking the OS to open it synchronously.
- w - Open file for writing.
- wx - Like 'w' but fails if the path exists
- w+ - Open file for reading and writing.
- wx+ - Like 'w+' but fails if path exists
- a - Open file for appending
- ax - Like 'a' but fails if the path exists.
- a+ - Open file for reading and appending
- ax+ - Like 'a+' but fails if the path exists

- **File information:**

- *stats.isFile()* - Returns true if file type of a simple file.
- *stats.isDirectory()* - Returns true if file type of a directory.
- *stats.isBlockDevice()* - Returns true if file type of a block device.
- *stats.isCharacterDevice()* - Returns true if file type of a character device.
- *stats.isSymbolicLink()* - Returns true if file type of a symbolic link.
- *stats.isFIFO()* - Returns true if file type of a FIFO.
- *stats.isSocket()* - Returns true if file type of asocket.

Global objects

- Node.js global objects are global in nature and they are available in all modules.
- We do not need to include these objects in our application, rather we can use them directly.

- **Objects**:
- *__filename* – It represents the filename of the code being executed.
- *__dirname* – It represents the name of the directory that the currently executing script resides in.
- *setTimeout(cb, ms)* – It is a global function used to run callback cb after at least ms milliseconds.
- *clearTimeout(t)* – It is a global function used to stop a timer that was previously created with setTimeout().
- *setInterval(cb, ms)* – It is global function used to run callback cb repeatedly after at least ms milliseconds.
- *process.argv* – Command line parameters.

- Open **8-GlobalObjects** folder, and run **testGlobalObj.js**.

testGlobalObj.js

```
// __filename.
console.log('Current file: '+__filename);

// __dirname.
console.log('Current folder: '+__dirname);

// Set timer.
console.log('\nSetting up a timer...');
function timerHandler (){
    console.log('Introduction to NodeJS - Interfaces course.');
}

setTimeout(timerHandler, 1000);
console.log('In 1000ms a message will appear...');

// Clearing a timer.
console.log('\nClearing a timer...');
var myTimer = setTimeout(timerHandler, 2000);
console.log('In 2000ms a message will appear...');
clearTimeout(myTimer);
console.log('Timer cleared... no message.');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Setting up a timer...
In 1000ms a message will appear...

Clearing a timer...
In 2000ms a message will appear...
Timer cleared... no message.

 Interval...
End...
Introduction to NodeJS - Interfaces course.
Introduction to NodeJS - Interfaces course.
Introduction to NodeJS - Interfaces course.
Introduction to NodeJS - Interfaces course.
Clearing interval...
Introduction to NodeJS - Interfaces course.

i:\00-Univalle-Pendientes\Curso-Interfaces\NodeJS\9-NodeJS-Code\8-GlobalObjects>
```

Utility Modules

- *OS module*: Provides a few basic operating-system related utility functions.

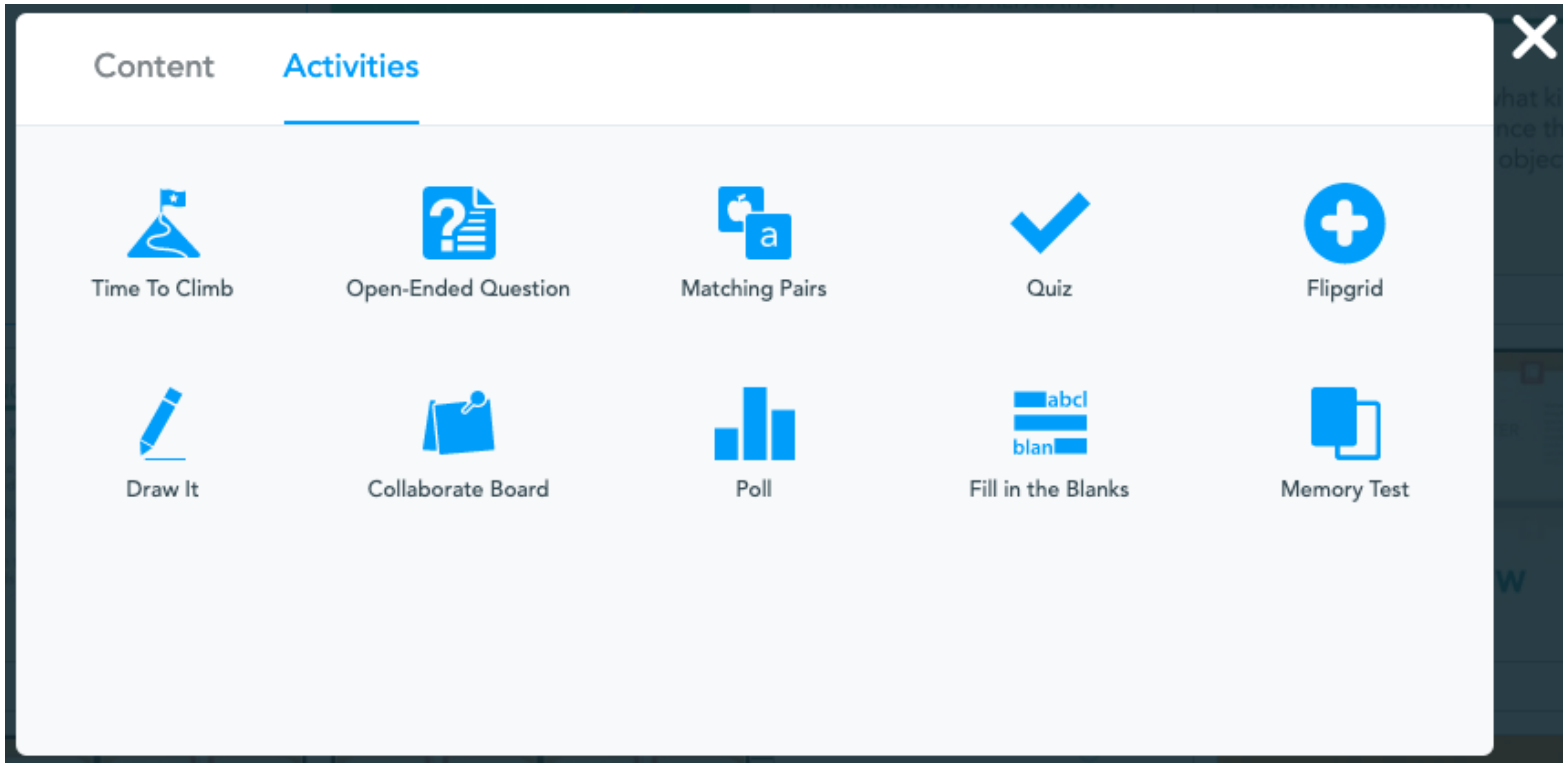
- *Crypto module*: The crypto module offers a way of encapsulating secure credentials to be used as part of a secure HTTPS net or http connection. It also offers a set of wrappers for OpenSSL's hash, hmac, cipher, decipher, sign and verify methods.

- *Child process*: It is possible to stream data through a child's stdin, stdout, and stderr in a fully non-blocking way.

- Open folder **8-GlobalObjects**, and run **testUtilities.js**.

Nearpod Activity

- Please go to the Nearpod link shared in the chat.
- Fulfil the Nearpod activity.
- Analyze the results with your teacher.



¿Questions?