NodeJS For Beginner

By Apaichon Punopas
apaichon@hotmail.com
https://www.facebook.com/apaichon.pup

Objective

- Understand What is NodeJS ?
- Understand Blocking I/O and Non Blocking I/O.
- Setup NodeJS
- Node Package Manager
- Hello World Application
- List of NodeJS Library
- Express Library

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.
- It is used by Groupon, SAP , LinkedIn , Microsoft, Yahoo , Walmart , Paypal.

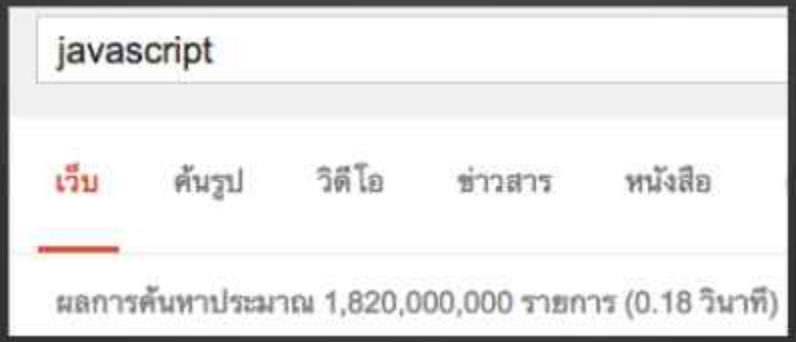
What is JavaScript ?

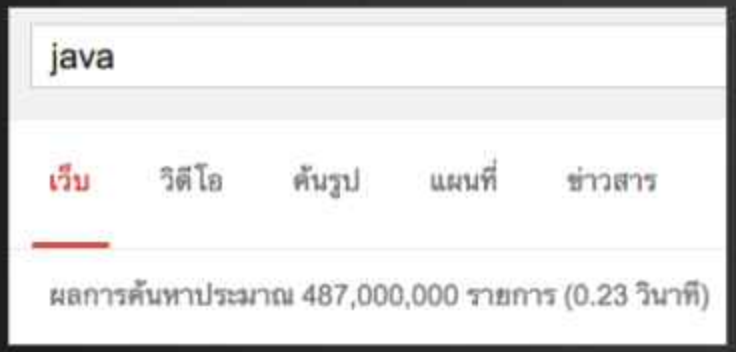
- It is dynamic programming language.
- It is client side script to interact with user.
- It most commonly used as a part of web browser.

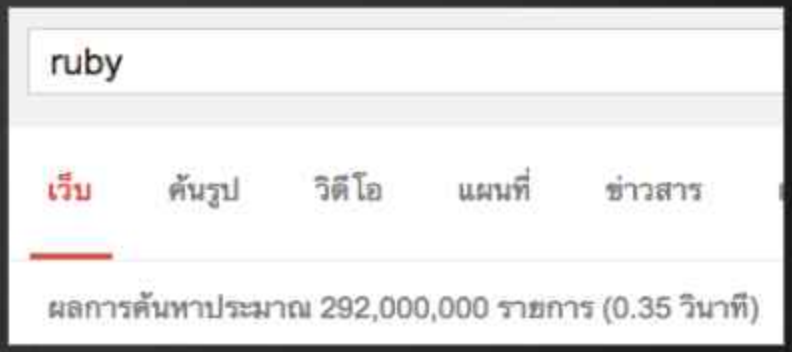
JavaScript

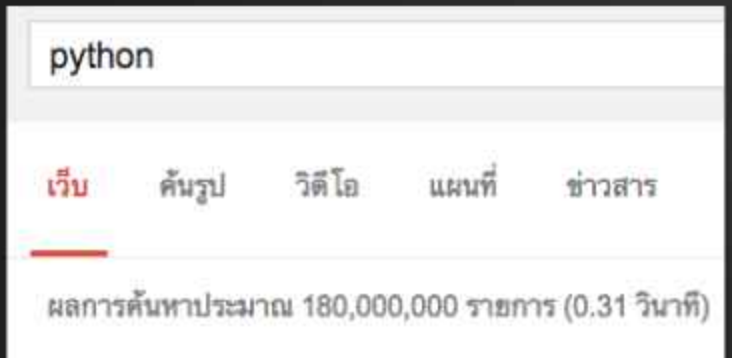
Why JavaScript ?

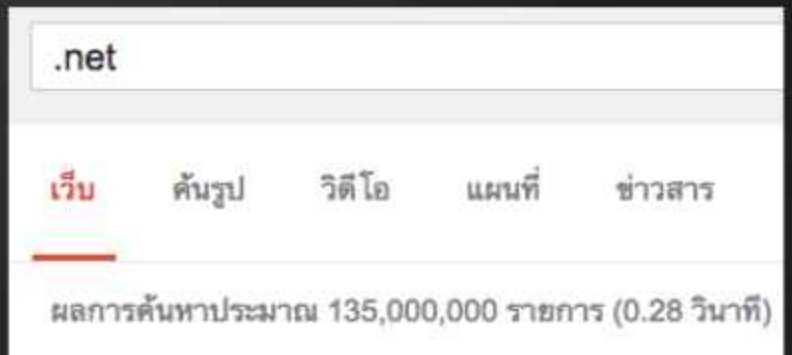
As of 14/1/2015











http://cdn2.carlcheo.com/wp-content/uploads/2014/12/which-programming-language-should-i-learn-first-infographic.png

THE LORD OF THE RINGS ANALOGY TO PROGRAMMING LANGUAGES

ava

andalf

DIFFICULTY

nts peace & works with everyone irtable)

ry popular on all platforms, OS, and vices due to its portability

e of the most in demand & highest ying programming languages

gan: write once, work everywhere

OPULARITY

USED TO BUILD

Gmail, Minecraft, Most Android Apps, Enterprise applications

'G. SALARY

102,000

C

One Ring

DIFFICULTY

The power of C is known to them all

Everyone wants to get its Power

Lingua franca of programming language

One of the oldest and most widely used language in the world

Popular language for system and hardware programming

A subset of C++ except the little details

POPULARITY

USED TO BUILD

Operating systems and hardware

AVG. SALARY

$102,000

C++

Saruman

DIFFICULTY

Everyone thinks that he is the good guy

But once you get to know him, you will realize he wants the power, not good deeds

Complex version of C with a lot more features

Widely used for developing games, industrial and performance-critical applications

Learning C++ is like learning how to manufacture, assemble, and drive a car

Recommended only if you have a mentor to guide you

POPULARITY

USED TO BUILD

Operating systems, hardware, and browsers

AVG. SALARY

$104,000

JavaScript

Hobbit

DIFFICULTY

Frequently underestimated (powerful)

Well-known for the slow, gentle life of the Shire (web browsers)

"Java and Javascript are similar like Car and Carpet are similar" - Greg Hewgill

Most popular clients-side web scripting language

A must learn for front-end web developer (HTML and CSS as well)

One of the hottest programming language now, due to its increasing popularity as server-side language (node.js)

POPULARITY

USED TO BUILD

Paypal, front-end of majority websites

AVG. SALARY

$99,000

C#

Elf

DIFFICULTY

Beautiful creature (language), but stays in their land, Rivendell (Microsoft Platform)

A popular choice for enterprise to create websites and Windows application using .NET framework

Can be used to build website with ASP.NET, a web framework from Microsoft

Similar to Java in basic syntax and some features

Learn C# instead of Java if you are targeting to work on Windows platform only

POPULARITY

USED TO BUILD

Enterprise and Windows applications

AVG. SALARY

$94,000

Ruby

Man (Middle Earth)

DIFFICULTY

Very emotional creature

They (some Ruby developers) feel they are superior & need to rule the Middle Earth

Mostly known for its popular web framework, Ruby on Rails

Focuses on getting things done

Designed for fun and productive coding

Best for fun and personal projects, startups, and rapid development

POPULARITY

USED TO BUILD

Hulu, Groupon, Slideshare

AVG. SALARY

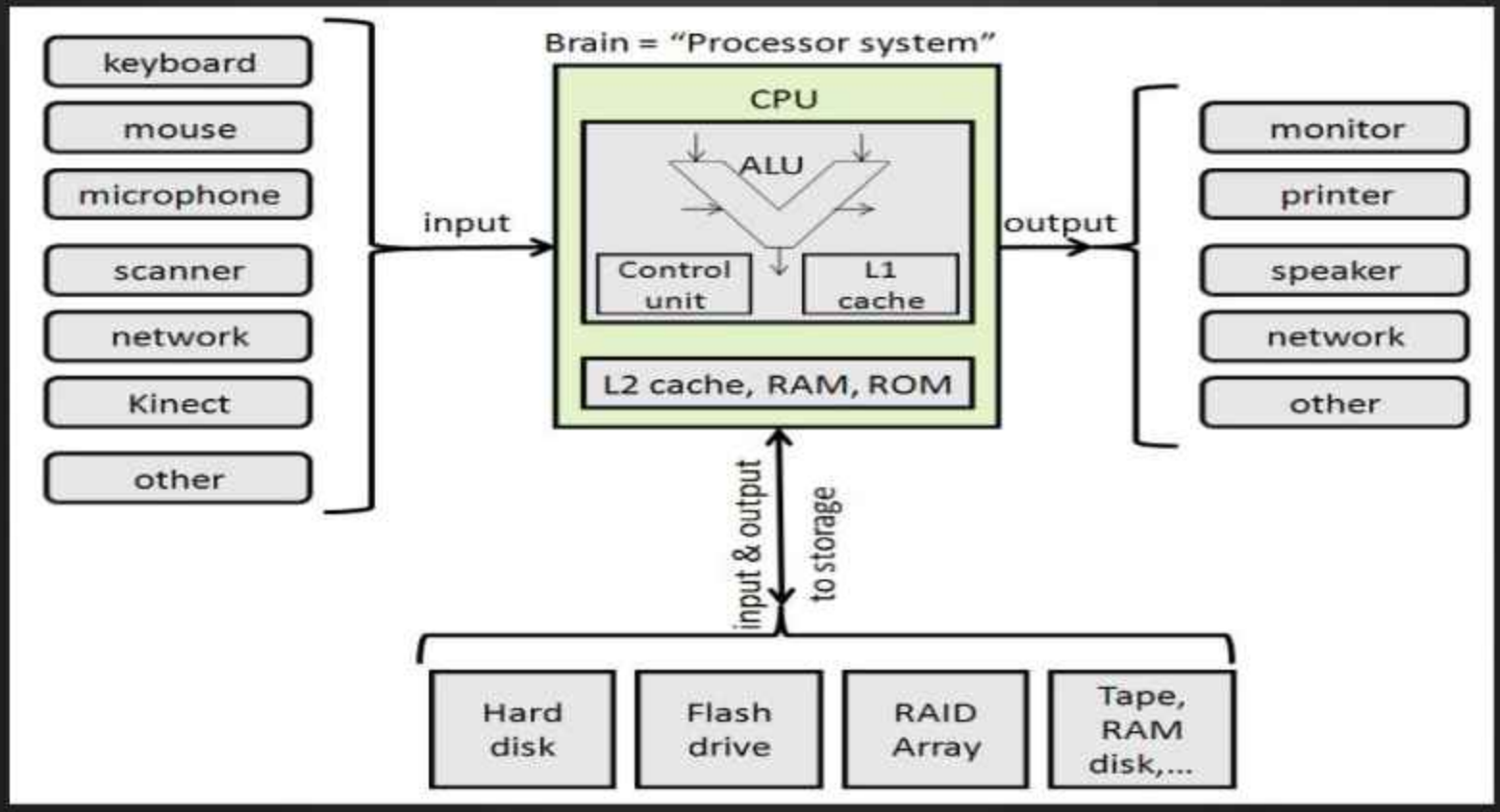
$107,000

JavaScript Basic

```
//Declare variable
var name ="PUP" , pi = 3.14 , isActive =true;
//Declare function
function cal(){
var a =999 , b = 123.343;
console.log(a+b);
}
//Declare object (JSON)
var animal ={type:"dog",maxAge:15
,bark: function(){alert("Hong Hong");}
}
```

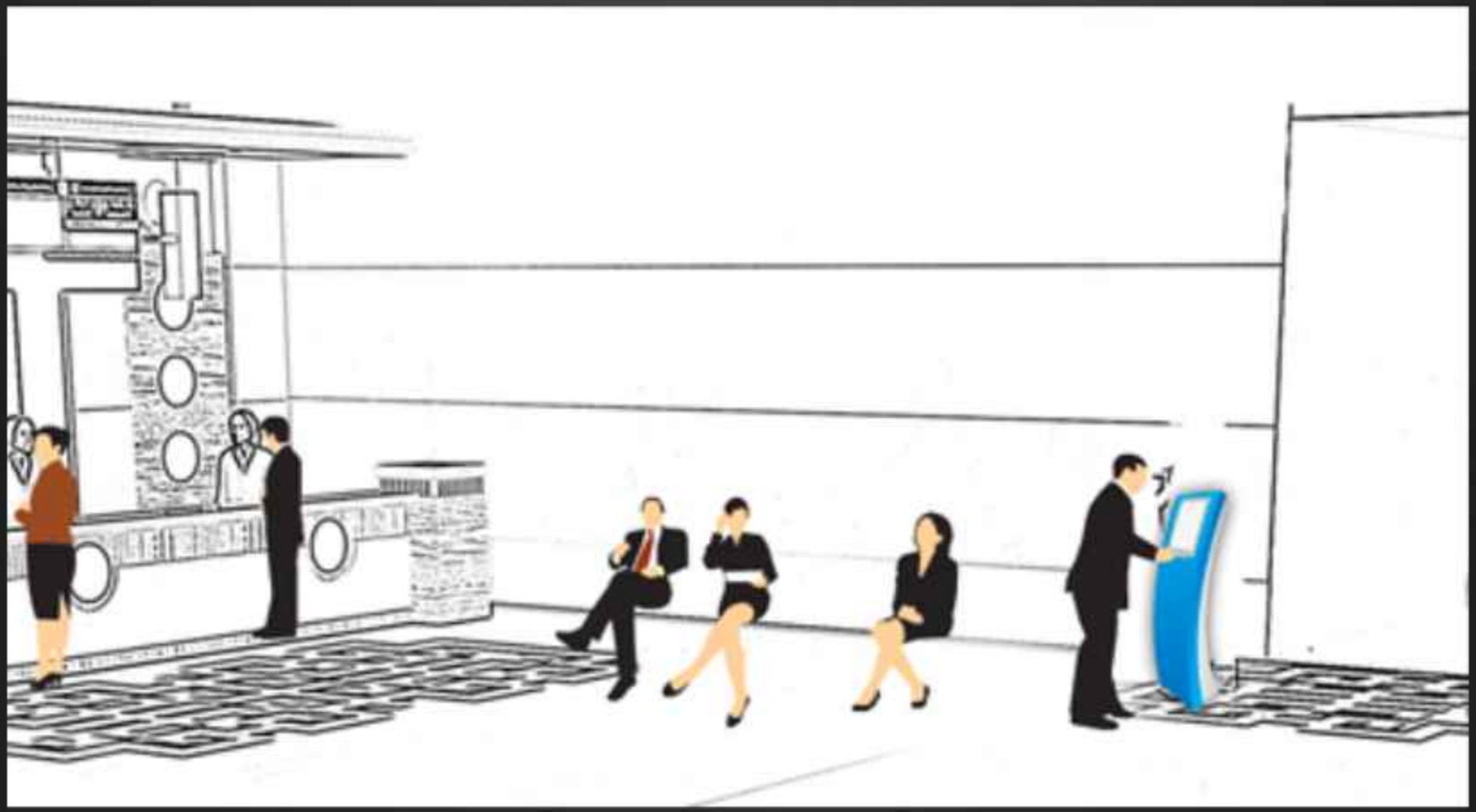
What's I/O ?

- Distance of data travel from input to output.

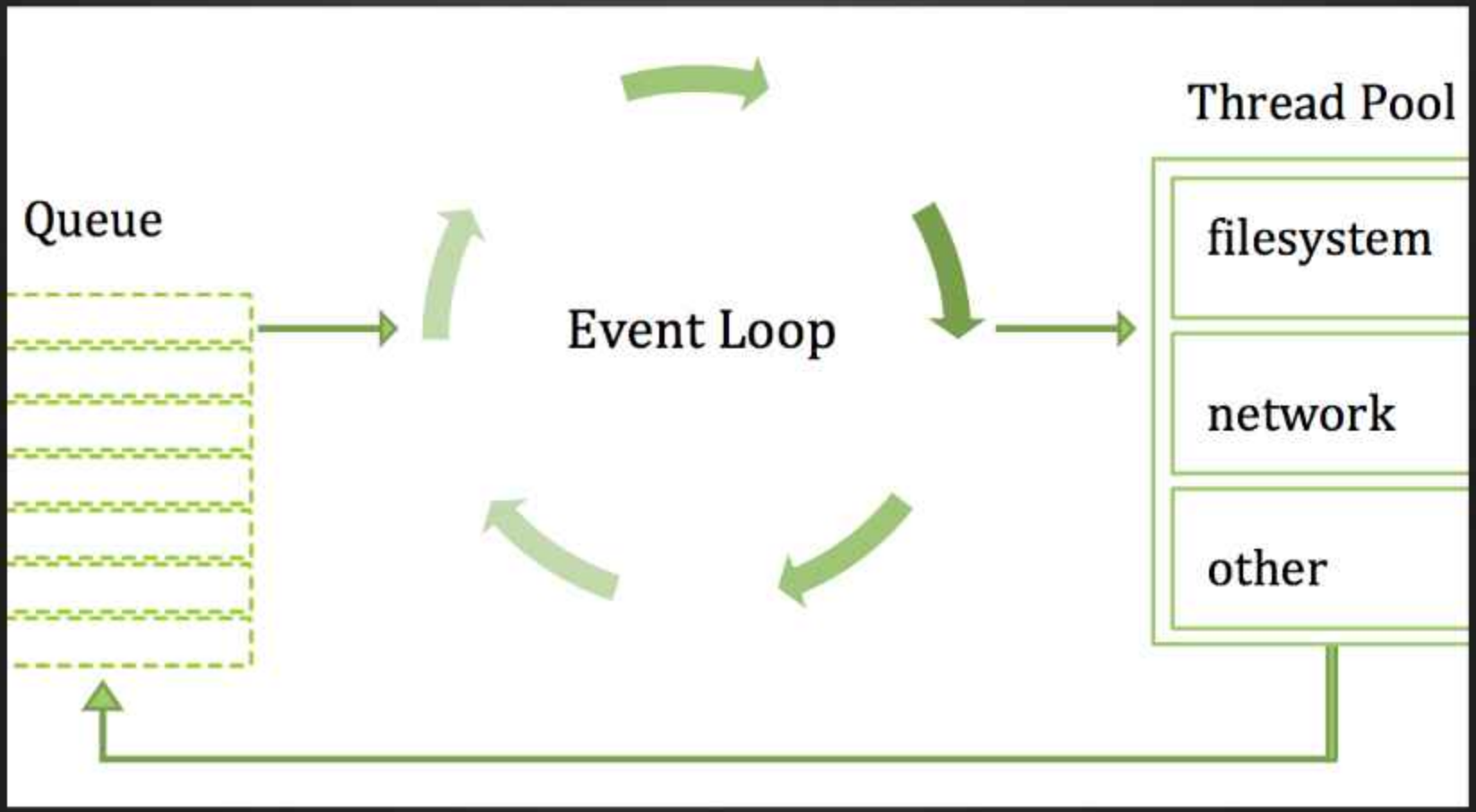


Blocking I/O

Non Blocking I/O



NodeJS Event Loop



JavaScript V8 Engine

- The V8 JavaScript Engine is an open source JavaScript engine developed by Google for the Google Chrome web browser.
- V8 compiles JavaScript to native machine code (IA-32, x86-64, ARM, or MIPS ISAs) before executing it.

Reasons to use NodeJS

- It is very lightweight and fast. There has been over 200,000 visits on this website in three weeks and minimal server resources has been able to handle it all.
- The counter is really easy to make to be real time.
- 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.

NodeJS Setup

- Download at http://nodejs.org/download/
- Set Environtment variable (Last version automatic set)
- Open command line and type following.

```
node -v
```

First App with Hello World

- Create example.js file and write code following.

```
var http = require('http');
http.createServer(function (request, response) {
  response.writeHead(200,
    {'Content-Type': 'text/plain'});
     response.end('Hello World\n');
}).listen(8124);
console.log('Server running at http://127.0.0.1:8124/');
```

- Open Terminal then type node example.js

Global Objects

- These objects are available in all modules.
  - process - In computing, a process is an instance of a computer program that is being executed

```
process.on('uncaughtException', function(err) {
console.log('Caught exception: ' + err);
});
```

  - console - For printing to stdout and stderr.
  - require - refer to external library.

Global Objects #2

- __filename – return current file's name
- __dirname – return current directory.
- module.exports is used for defining what a module exports and makes available through require().
- setTimeout(cb, ms) – Run callback cb after at least ms milliseconds. The timeout must be in the range of 1-2,147,483,647 cannot span more than 24.8 days.
- clearTimeout(t) – Stop a timer that was previously created with setTimeout().

```
console.log(__filename);
console.log(__filename);
var t=setTimeout(helloWorld,3000);
clearTimeout(t);
```

Global Objects #3

- setInterval(cb, ms) – Run callback cb repeatedly every ms milliseconds.
- clearInterval(t) - Stop a timer that was previously created with setInterval().

```
var i =0;
var tCounter = setInterval(counter,2000);

function counter(){
    i++;
    console.log(i);
}

setTimeout(function(){
    clearInterval(tCounter);
}, 10000);
```

Modules

- It is external library and available refer to use by require().

```
var circle =require('./circle.js');
console.log(circle.area(50));

// circle.js
var PI = Math.PI;

exports.area = function (r) {
  return PI * r * r;
};
```

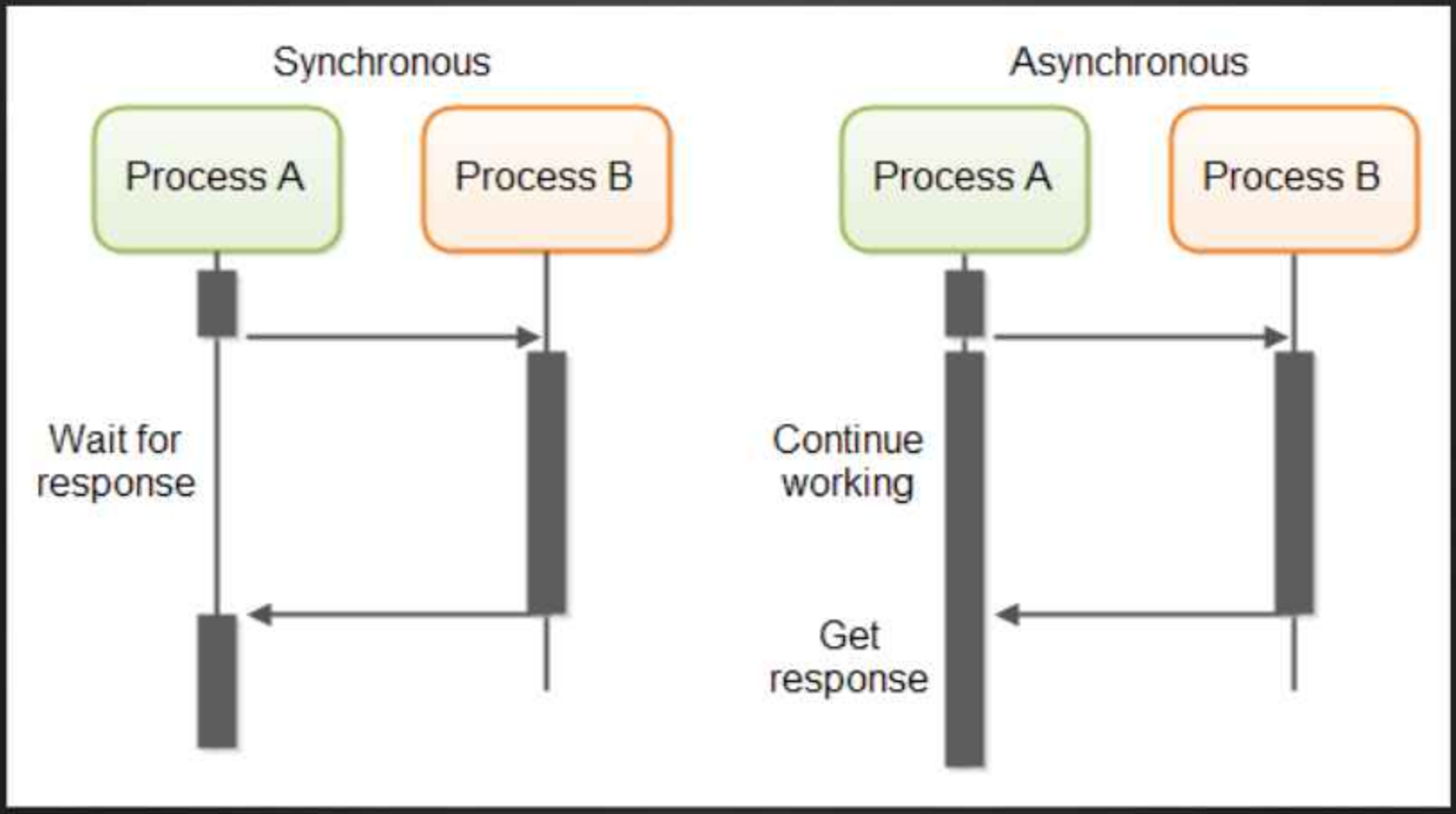
- All libraries in node_modules no need to refer path ‘./’ or ‘../’.

npm – Node Package Manager

- npm is the package manager for javascript and more.
- https://www.npmjs.com/

```
npm install moduleName -option
```

Synchronous - Asynchronous



JavaScript Callback

```
var util = require('util');
var fs = require('fs');
function readFile(str, callback){
    var lines =[];
    fs.readFile(str, function (err, data) {
      if (err) throw err;
      lines.push(data);
      console.log(data.toString());
    });
    callback(lines);
}
var a, b ;
readFile('./20.txt', function(data){a= data;});
readFile('./10.txt',function(data){b= data;});
util.log("a"+ a);
util.log("b" +b);
```

Core Module

- Assertion Testing
- Buffer
- C/C++ Addons
- Child Processes
- Cluster
- Crypto
- Debugger
- DNS
- Domain
- Events
- File System
- HTTP/HTTPS
- Net
- OS
- Path
- Process
- Punycode
- Query Strings
- REPL
- Stream

Core Module#2

String Decoder

Timers

TLS/SSL

TTY

UDP/Datagram

URL

Utilities

VM

ZLIB

Assertion Testing

- This module is used for writing unit tests for your applications, you can access it with require('assert').

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

function add (a, b) {
  return a + b;
}

var expected = add(1,2);
assert( expected === 4, 'one plus two is three');
```

Buffer

- Pure JavaScript is Unicode friendly but not nice to binary data. When dealing with TCP streams or the file system, it's necessary to handle octet streams. Node has several strategies for manipulating, creating, and consuming octet streams.

```
buf = new Buffer(10);
buf.write("abcdefghj", 0, "ascii");
console.log(buf.toString('base64'));
buf = buf.slice(0,5);
console.log(buf.toString('utf8'));
```

C/C++ Add on

Addons are dynamically linked shared objects. They can provide glue to C and C++ libraries.

- Create hello.cc
- Create binding.gyp
- Run node-gyp configure
- Run node-gyp build
- Create hello.js
- Run node hello.js

Child Process

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

```
var exec = require('child_process').exec
exec('ls', function (err, stdout, stderr){
    if (err) {
        console.log("child processes failed with error
code: " +
            err.code);
    }
    console.log(stdout);
});
```

Cluster

- A single instance of Node runs in a single thread. To take advantage of multi-core systems the user will sometimes want to launch a cluster of Node processes to handle the load.

```
var cluster = require('cluster');
if (cluster.isMaster) {
  var numCPUs = require('os').cpus().length;
  for (var i = 0; i < numCPUs; i++) {
    cluster.fork();
  }
  Object.keys(cluster.workers).forEach(function(id) {
    console.log(cluster.workers[id].process.pid);
  });
}
```

Crypto

- 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.

```
var crypto = require('crypto');
var fs = require('fs');
var shasum = crypto.createHash('sha1');
var s = fs.ReadStream('file.txt');
s.on('data', function(d) {
  shasum.update(d);
});
s.on('end', function() {
  var d = shasum.digest('hex');
  console.log(d + '  file.txt' );
});
```

Debugger

- V8 comes with an extensive debugger which is accessible out-of-process via a simple TCP protocol. Node has a built-in client for this debugger. To use this, start Node with the debug argument; a prompt will appear:

```
//node debug debugger.js
for (var i=0;i<10;i++){
    console.log(i);
}
```

DNS

- This module contains functions that belong to two different categories:
  - Functions that use the underlying operating system facilities to perform name resolution.

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

dns.lookup('www.google.com', function onLookup(err,
addresses, family) {
  console.log('addresses:', addresses);
});
```

DNS#2

- Functions that connect to an actual DNS server to perform name resolution.

```
var dns = require('dns');
dns.resolve4('www.google.com', function (err,
addresses) {
  if (err) throw err;
  console.log('addresses: ' +
      JSON.stringify(addresses));
  addresses.forEach(function (a) {
    dns.reverse(a, function (err, domains) {
      if (err) {throw err;}
      console.log('reverse for ' + a + ': ' +
                JSON.stringify(domains));
    });
  });
});
```

Domain

Domains provide a way to handle multiple different IO operations as a single group. If any of the event emitters or callbacks registered to a domain emit an error event, or throw an error, then the domain object will be notified, rather than losing the context of the error.

Events

- Event is an action or occurrence detected by the program that may be handled by the program.

```
var events =require('events').EventEmitter;
var ee = new events();
ee.items = [1,2,3,4,5,6,7];
ee.on("changedItem", function () {
    console.log("event has occured");
    console.log("index:" + this.lastIndexChanged + "
     value:" +this.items[this.lastIndexChanged]);
});
function setItem(index,value){
   ee.items[index] = value;
   ee.lastIndexChanged = index;
   ee.emit("changedItem");
}
setItem(3,"Hello World!");
```

File System

- File I/O is provided by simple wrappers around standard POSIX functions. To use this module do require('fs'). All the methods have asynchronous and synchronous forms.

```
var fs = require('fs');
fs.writeFile('message.txt', 'Hello Node', function
(err) {
  if (err) throw err;
  console.log('It\'s saved!');
});
```

HTTPS

- HTTPS is the HTTP protocol over TLS/SSL. In Node this is implemented as a separate module.

```
var https = require('https');
var fs = require('fs');

var options = {
  key: fs.readFileSync('./localhost.key'),
  cert: fs.readFileSync('./localhost.cert')
};

https.createServer(options, function (req, res) {
  res.writeHead(200);
  res.end("hello world\n");
}).listen(8000);
```

Net

- Creates a new TCP server. The connectionListener argument is automatically set as a listener for the 'connection' event.options is an object with the following defaults:

```
var net = require('net');
var server = net.createServer(function(c)
{ //'connection' listener
  console.log('client connected');
  c.on('end', function() {
    console.log('client disconnected');
  });
  c.write('hello\r\n');
  c.pipe(c);
});
server.listen(8124, function() { //'listening' listener
  console.log('server bound');
});
```

Net#2

```
//client
var net = require('net');
var client = net.connect({port: 8124},
    function() { //'connect' listener
  console.log('connected to server!');
  client.write('world!\r\n');
});
client.on('data', function(data) {
  console.log(data.toString());
  client.end();
});
client.on('end', function() {
  console.log('disconnected from server');
});
```

OS

- Provides a few basic operating-system related utility functions.

```
var os = require('os');
console.log(os.hostname());
console.log(os.type());
console.log(os.platform());
console.log(os.arch());
console.log(os.release());
console.log(os.uptime());
console.log(os.loadavg());
```

Path

- This module contains utilities for handling and transforming file paths. Almost all these methods perform only string transformations

```
var path= require('path');
var result ="";
result+= path.normalize('/foo/bar//baz/asdf/quux/..');
result+="\n" + path.join('/foo', 'bar', 'baz/asdf',
'quux', '..');
result+="\n"+path.resolve('foo/bar', '/tmp/file/',
'..', 'a/../subfile');
console.log(result);
```

Punycode

- Library for convert standard character code to readable or convert back to standard code.

```
var punycode = require('punycode');
console.log(punycode.decode('maana-pta')); // 'mañana'
punycode.decode('--dqo34k'); // '💩-⌘'
// encode domain name parts
punycode.encode('mañana'); // 'maana-pta'
punycode.encode('💩-⌘'); // '--dqo34k'
// decode domain names
punycode.toUnicode('xn--maana-pta.com'); //
'mañana.com'
punycode.toUnicode('xn----dqo34k.com'); // '💩-⌘.com'
// encode domain names
punycode.toASCII('mañana.com'); // 'xn--maana-pta.com'
punycode.toASCII('💩-⌘.com'); // 'xn----dqo34k.com'
```

Query String

- This module provides utilities for dealing with query strings.

```
var querystring =require('querystring');
var result =querystring.stringify({ foo: 'bar', baz:
['qux', 'quux'], corge: '' });
result += "\n" +querystring.stringify({foo: 'bar', baz:
'qux'}, ';', ':');
result += "\n"
+querystring.parse('foo=bar&baz=qux&baz=quux&corge')
console.log(result);
```

readline

- To use this module, do require('readline'). Readline allows reading of a stream (such as process.stdin) on a line-by-line basis.

```
var readline = require('readline');
var rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout
});
rl.question("What do you think of node.js? ",
function(answer) {
  // TODO: Log the answer in a database
  console.log("Thank you for your valuable
feedback:", answer);
  rl.close();
});
```

Stream

- A stream is an abstract interface implemented by various objects in Node.

```
var fs = require('fs');
var zlib = require('zlib');
var r = fs.createReadStream('file.txt');
var z = zlib.createGzip();
var w = fs.createWriteStream('file.txt.gz');
r.pipe(z).pipe(w);
```

String Decoder

- To use this module, do require('string_decoder'). StringDecoder decodes a buffer to a string. It is a simple interface to buffer.toString() but provides additional support for utf8.

```
var StringDecoder =
require('string_decoder').StringDecoder;
var decoder = new StringDecoder('utf8');

var cent = new Buffer([0xC2, 0xA2]);
console.log(decoder.write(cent));

var euro = new Buffer([0xE2, 0x82, 0xAC]);
console.log(decoder.write(euro));
```

Timers

- All of the timer functions are globals.
  - setTimeout(callback, delay, [arg], [...])
  - clearTimeout(timeoutObject)
  - setInterval(callback, delay, [arg], [...])
  - clearInterval(intervalObject)
  - unref()
  - ref()
  - setImmediate(callback, [arg], [...])
  - clearImmediate(immediateObject)

UDP

- UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection.

```
//Server
var dgram = require("dgram");
var server = dgram.createSocket("udp4");
server.on("error", function (err) {
  console.log("server error:\n" + err.stack);
  server.close();
});
server.on("message", function (msg, rinfo) {
  console.log("server got: " + msg + " from " +
    rinfo.address + ":" + rinfo.port);
});
server.on("listening", function () {
  var address = server.address();
  console.log("server listening " +
      address.address + ":" + address.port);
});
server.bind(41234);
```

UDP #2 Client

```
var dgram = require('dgram');
var message = new Buffer("Some bytes");
var client = dgram.createSocket("udp4");
client.send(message, 0, message.length,
41234, "localhost", function(err, bytes)
{
  client.close();
});
```

URL

This module has utilities for URL resolution and parsing. Call require('url') to use it.

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

var result =url.parse('http://
user:pass@host.com:8080/p/a/t/h?
query=string#hash');
console.log(result);
```

Utilities

- Utility Library help convert and validate format of value.

```
var util  = require('util');
console.log(util.format('%s:%s', 'foo', 'bar',
'baz')); // 'foo:bar baz'
console.log(util.format('{%j:%j}', 'name', 'pup'));
console.log(util.format('%d', 1, 2,3,"4"));
util.debug('message on stderr');

console.log(util.isArray([]));
console.log(util.isArray(new Array));
util.print(util.isArray({}));

util.log(util.isDate(new Date()));
```

vm

- In computing, a virtual machine (VM) is an emulation of a particular computer system. Virtual machines operate based on the computer architecture and functions of a real or hypothetical computer, and their implementations may involve specialized hardware, software, or a combination of both.

```
var util = require('util');
var vm = require('vm');
var myContext = {
   hello: "nobody"
}
vm.runInNewContext('hello = "world";',
myContext);
util.log('Hello ' + myContext.hello);
```

Thank you