Web Application Programming

Introduction to Node.js

Node.js Lecture 1

These slides were drawn from the following references:

https://www.w3schools.com/nodejs/

https://www.tutorialsteacher.com/nodejs/nodejs-tutorials

What is Node.js

- Node.js is an open-source server environment
- Node.js is free
- Node.js runs on various platforms (Windows, Linux, Unix, Mac OS X, etc.)
- Node.js uses JavaScript on the server

Opening a File

A common task for a web server can be to open a file on the server and return the content to the client. Here is how PHP or ASP handles a file request:

1. Sends the task to the computer's file system.
2. Waits while the file system opens and reads the file.
3. Returns the content to the client.
4. Ready to handle the next request.

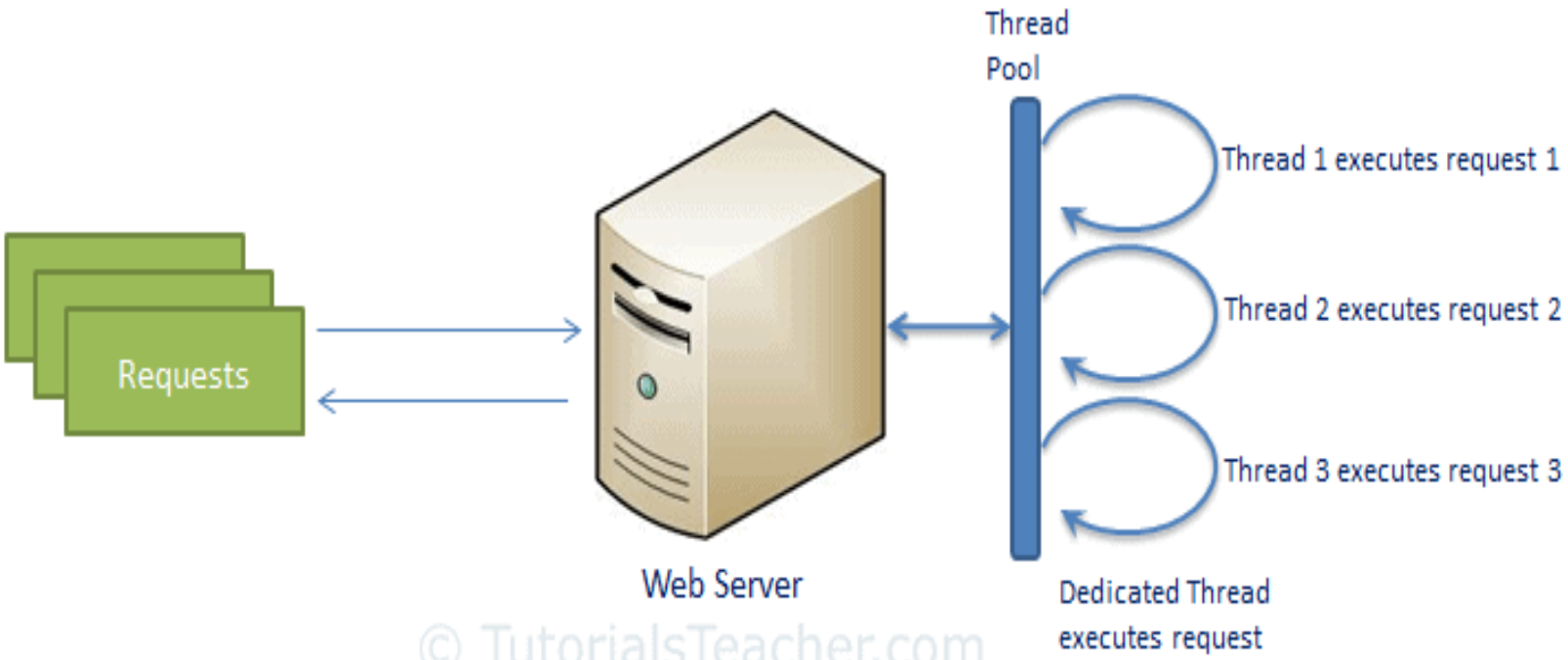
Node.js is Asynchronous

Here is how Node.js handles a file request:

1. Sends the task to the computer's file system.
2. Ready to handle the next request.
3. When the file system has opened and read the file, the server returns the content to the client.

Node.js eliminates the waiting, and simply continues with the next request.
Node.js runs single-threaded, non-blocking, asynchronous programming, which is very memory efficient.

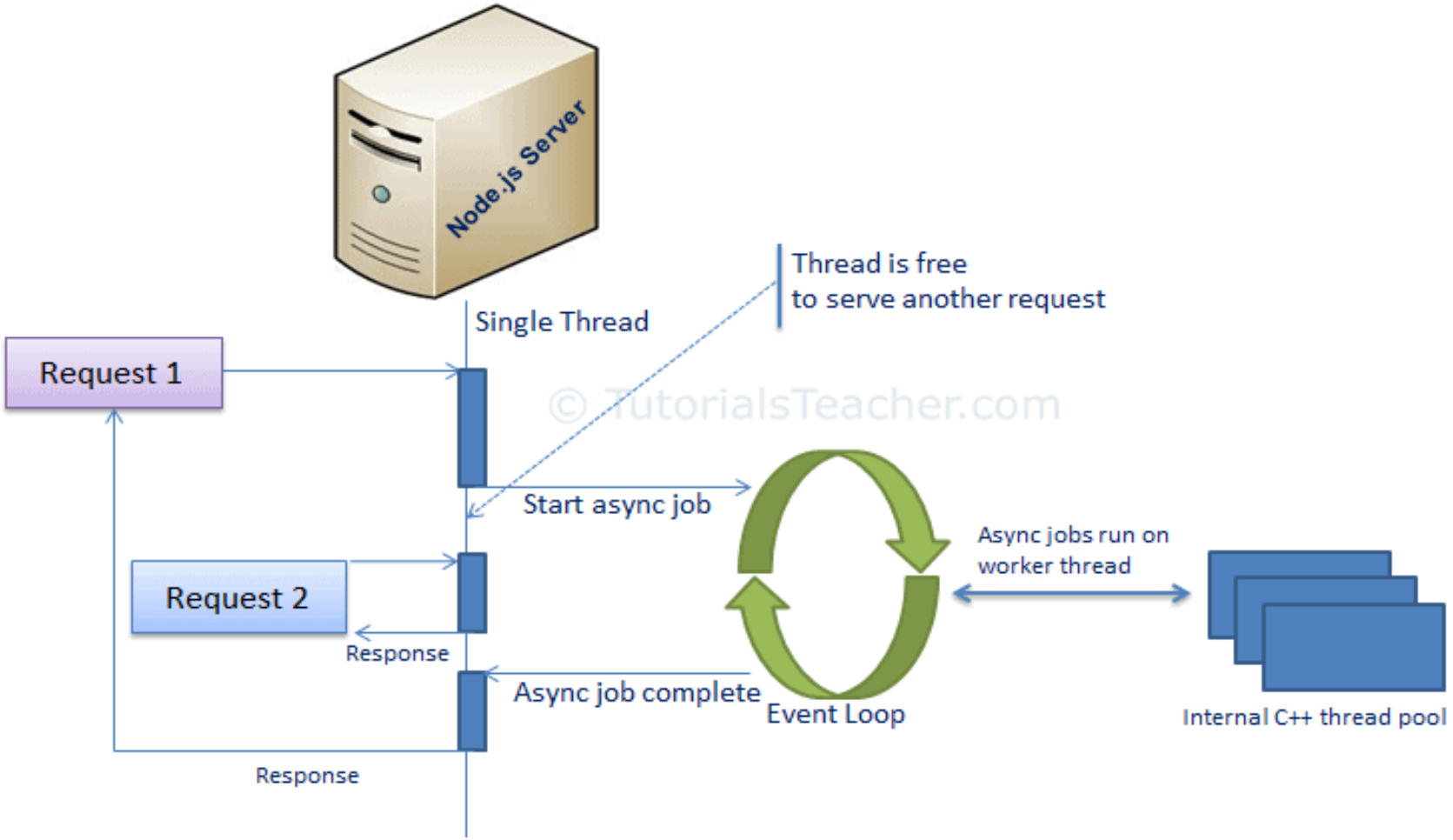
Traditional Web Server



Traditional Web Server Model

- Each request is handled by a dedicated thread from the thread pool.
- If no thread is available in the thread pool at any point of time, then the request waits until the next available thread.
- Dedicated thread executes a particular request and does not return to thread pool until it completes the execution and returns a response.

Node.js Process Model



Node.js Process Model

- Node.js processes user requests differently when compared to a traditional web server model.
- Node.js runs in a single process, and the application code runs in a single thread, and thereby needs less resources than other platforms.

Node.js Single Thread

- All the user requests to your web application will be handled by a single thread.
- All the I/O work or long running job is performed asynchronously for a particular request.
- This single thread doesn't have to wait for the request to complete and is free to handle the next request.
- When asynchronous I/O work completes, then it processes the request further and sends the response.

Node.js Performance

- Node.js process model increases the performance and scalability with some exceptions.
- Node.js is not fit for an application which performs CPU-intensive operations like image processing or other heavy computation work because it takes time to process a request and thereby blocks the single thread.

Exercise N1

- Download Node.js from the official Node.js website and install it: https://nodejs.org.
- Create a folder for your Node.js files (**`NodeFolder`**).
- Save the Hello World program in a file using VS Code:

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

http.createServer(function (req, res) {
  res.writeHead(200,
          {'Content-Type': 'text/html'});
  res.end('Hello World!');
}).listen(8080);
```

Exercise N1 (continued)

- Open a Command Window
- Run command:

```
C:NodeFolder> npm init
```

- Run the Hello World program:

```
C:NodeFolder> node HelloWorld.js
```

- In your Browser, enter the URL:

http://localhost:8080

Modules

- Modules are like JavaScript libraries.
- A *Module* is a set of functions you want to include in your application.
- Node.js has a set of built-in modules which you can use without any further installation.
- To include a module, use the **`require()`** function with the name of the module:

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

Types of Modules

- Built-in Modules
  - Node.js has a set of built-in modules which you can use without any further installation.
  - buffer, fs, http, path, etc.
  - Built-in Modules Reference
- 3rd party modules on www.npmjs.com
- Your own Modules
  - Simply create a normal JS file it will be a module *(without affecting the rest of other JS files and without messing with the global scope).*

Include Modules – `require()` function

- The basic functionality of `require` is that it reads a JavaScript file, executes the file, and then proceeds to return the `module.exports` object.
  - `const path = require('path');`
  - `const config = require('./config');`

- Rules:
  - if the file doesn't start with "**./**" , then it is considered a **core module** (and the local Node path is checked).
  - If the file starts with "**./**" it is considered a relative file to the file that called `require`.

How `require('/path/to/file')` works

- Node goes through the following sequence of steps:
  1. Resolve: to find the absolute path of the file
  2. Load: to determine the type of the file content
  3. Wrap: to give the file its private scope
  4. Evaluate: This is what the VM actually does with the loaded code
  5. Cache: when we require this file again, don't go over all the steps.

- **Note**: Node core modules return immediately (no resolve)

What's the wrapper?

- `node -p "require('module').wrapper"`

1. Node will wrap your code into:

```
(function (exports, require, module, __filename, __dirname){
   // This is why you can use require, and module in your code
   // Node will initialize them and pass them as parameters to
   //    this wrapper function
    return module.exports;
});
```

2. Node will run the function using .apply()
3. Node will return the following:

```
return module.exports;
```

module.exports

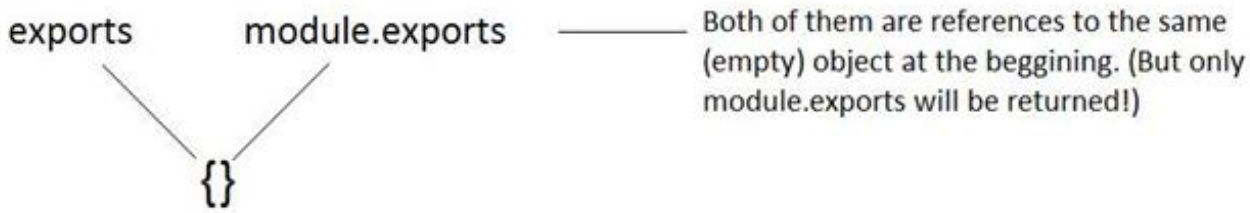
- Think of this object (`module.exports`) as a return statement.

```
// helloModule.js
var sayHi = function(){
              console.log('hi');
            }

module.exports = sayHi;
```

```
// app.js
var hello = require('./helloModule');

hello();
```

`exports` vs `module.exports`

- `exports` **object is a reference to the** `module.exports`**, that is shorter to type**



- **At the end,** `module.exports` **will be returned.**

Create your own module

play folder

```
// play/violin.js
const play = function() { console.log("First Violin is playing!"); }
module.exports = play;
```

```
// play/clarinet.js
const play = function() { console.log("Clarinet is playing!"); }
module.exports = play;
```

```
// play/index.js
const violin = require('./violin');
const clarinet = require('./clarinet');
module.exports = { 'violin': violin, 'clarinet': clarinet };
```

```
// app.js
const play = require('./play');
play.violin();
play.clarinet();
```

Main Point

- Node implements the CommonJS module specification. This allows the programmer to include additional libraries (.js code files) into their code using the **require()** function, which returns the **module.exports** object defined in the specified .js file.

- *Science of Consciousness*: Knowledge can be gained from inside and outside

Creating New Modules (Exercise N2)

```
myDate function () {
   return Date();
};
exports.myDate = myDate;
```

- Use the *exports* keyword to make properties and methods available outside the module file.
- Save this in a file called **`myModule.js`**

Exercise N2 (continued)

- Use the module *myModule* in a Node.js file:

```
var http = require('http');
var dt = require('./myModule');

http.createServer(function (req, res) {
  res.writeHead(200,
          {'Content-Type': 'text/html'});
  res.write("The date and time are
      currently: " + dt.myDate());
  res.end();
}).listen(8080);
```

Creating New Modules

```
myDate function () {
   return Date();
};
exports.myDate = myDate;
```

- A node.js file may contain function definitions, variable declarations, and object declarations.
- By default, all of these declarations are **private**.
- The exports object is used to make any of these public.
- In the above example, function myDate( ) will become public.

Using a New Module

```
var dt = require('./myModule');
```

- The *require* operation needs the file name of the module (including the path from the currently executing node.js file).
- To use the public methods of the module, the variable name *dt* is used:

```
dt.myDate();
```

Another Module Example

```
const getName = () => {
  return 'Jim';
};

const getLocation = () => {
  return 'Munich';
};

const dateOfBirth = '12.01.1982';

exports.getName = getName;
exports.getLocation = getLocation;
exports.dob = dateOfBirth;
```

Using the New Module

```
const user = require('./userfile');
console.log(user.getName() + ' lives in ' + user.getLocation() + ' and was born on ' + user.dob);
```

- The *require* operation needs the file name of the module.
- The constant *user* accesses the public methods.

HTTP Module

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

//create a server object:
http.createServer(function (req, res) {
  //write a response to the client
  res.write('Hello World!');
  res.end(); //end the response
}).listen(8080);
//the server object listens on port 8080
```

HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

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

http.createServer(function (req, res) {
  res.writeHead(200,
      {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

Client Request

- The *req* argument represents the request from the client.
- The property *url* holds the part of the url that comes after the domain name.

```
var http = require('http');
http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

Exercise N3

Run the code shown below and then enter the following in your Browser:

http://localhost:8080/apples
http://localhost:8080/pears
http://localhost:8080/strawberries

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  res.write(req.url);
  res.end();
}).listen(8080);
```

Query String

The following code reads the values from the query string corresponding to the names *year* and *month*:

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

http.createServer(function (req, res) {
  res.writeHead(200, {'Content-Type': 'text/html'});
  var q = url.parse(req.url, true).query;
  var txt = q.year + " " + q.month;
  res.end(txt);
}).listen(8080);
```

HTTP Header

If the response from the HTTP server is supposed to be displayed as HTML, you should include an HTTP header with the correct content type:

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

http.createServer(function (req, res) {
  res.writeHead(200,
      {'Content-Type': 'text/html'});
  res.write('Hello World!');
  res.end();
}).listen(8080);
```

Node.js File System

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

Common use for the File System module:

- Read files
- Create files
- Update files
- Delete files
- Rename files

Exercise N4: Simple Adder

Opening HTML page
(**SimpleAdder.html**)

```
<form action =
            "http://localhost:8080/add.js">
  <div>Enter two numbers
    <input type = "text" name="first"/><br/>
    <input type = "text" name="second"/><br/>
    <input type="submit" value="Click"/>
  </div>
</form>
```

Module to Perform Addition

addmod.js

```
exports.add = function (req,res,vals) {
    var sum = parseInt(vals.first) + parseInt(vals.second);
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write("<!DOCTYPE html>");
    res.write("<html>");
    res.write("<head><meta charset=\"utf-8\"/>");
    res.write("<title>Calculator Web Site</title>");
    res.write("</head>");
    res.write("<body>");
    res.write("<p>The sum is: ");
    res.write(String(sum));
    res.write("</p>");
    res.write("</body>");
    res.write("</html>");
    return res.end();
};
```

Using Template Literals

addmod.js

```
exports.add = function (req,res,vals) {
var sum = parseInt(vals.first) + parseInt(vals.second);
    res.writeHead(200, {'Content-Type': 'text/html'});
    res.write(`<!DOCTYPE html>
      <html>
       <head><meta charset=\"utf-8\"/>
        <title>Calculator Web Site</title>
       </head>
       <body>
         <p>The sum is:  ${String(sum)}</p>
       </body>
      </html> ` );
     return res.end();
};
```

Web Server for the Adder

AdderWebServer.js

```
var http = require('http');
var url = require('url');
var fs = require('fs');
var addmod = require('./addmod.js');

http.createServer(function (req, res) {
  var q = url.parse(req.url, true);
  var filename = "." + q.pathname;
  if (q.pathname=="/add.js")
    addmod.add(req,res,q.query)
  else
  fs.readFile(filename, function(err, data) {
    if (err) {
      res.writeHead(404, {'Content-Type': 'text/html'});
      return res.end("404 Not Found");
    }
    res.writeHead(200);  // Content-Type not included
    res.write(data);
    return res.end();
  });
}).listen(8080);};
```

Run the Simple Adder

- To run the Simple Adder website, use the following URL:

  http://localhost:8080/SimpleAdder.html

End of Exercise N4

Reading a File

Use fs.readFile() method to read the physical file asynchronously.

Signature:

fs.readFile(fileName [,options], callback)

Parameter Description:

- filename: Full path and name of the file as a string.
- options: The options parameter can be an object or string which can include encoding and flag. The default encoding is utf8 and default flag is "r".
- callback: A function with two parameters *err* and *fd*. This will get called when *readFile* operation completes.

Read File Example

```
var fs = require('fs');
fs.readFile('TestFile.txt',
     function (err, data) {
        if (err) throw err;
        console.log(data);
      }
);
```