

Introduction to Node.js

Overview, Modules, Web Server, Request and Response



SoftUni Team
Technical Trainers



SoftUni



Software University

<https://softuni.bg>

sli.do

#js-back-end

Table of Contents

1. Introduction to **Node.js**
2. **Event** Loop
3. **Modules**
4. **Node.js** Web Server
5. **Request** and **Response Wrapper**





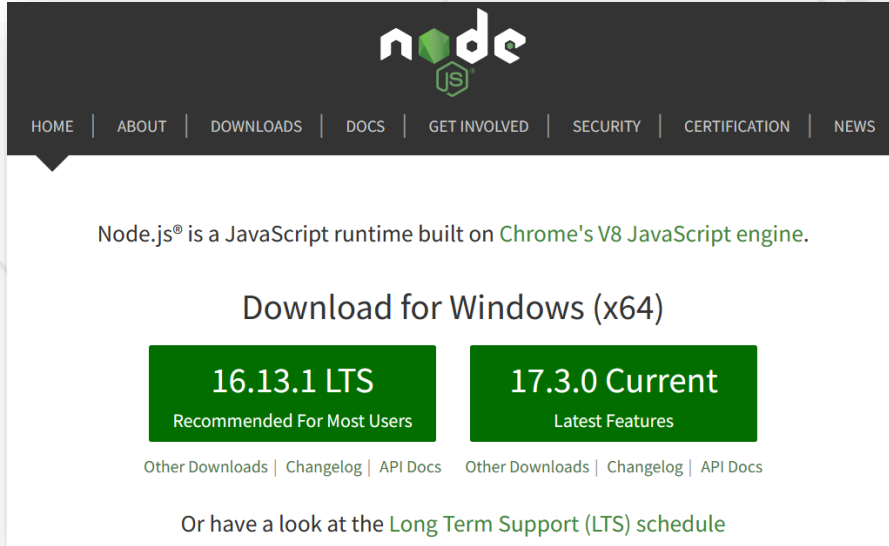
Introduction to Node.js

Node.js Overview

- A runtime environment for JS that runs on the server
- Advantages
 - **One language** for server and client
 - **Asynchronous** and **Event** Driven
 - Very **fast**
 - Efficient **package manager**



- Go to <http://nodejs.org> and install the latest version



- To check the currently installed version of the node, type in the **command prompt / terminal**:

```
node -v
```

- From the **terminal**

```
node           // Starts REPL  
let a = 5  
let b = 3  
a + b         // 8
```

- Interpret code from a **file**
 - Save the script to **index.js**
 - Execute from the terminal:

```
node index.js
```



- Node.js **projects** are usually set up as **NPM packages**
 - From the **terminal**, inside the **target directory**

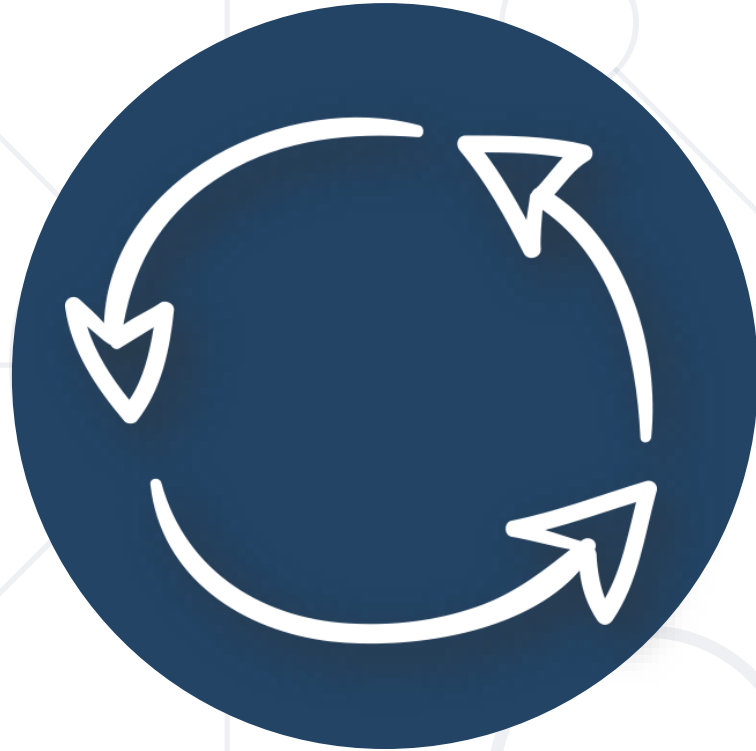
```
npm init
```

- Answer **questions** to initialize the project
- A **package.json** file will be created with initial configuration
- To bypass all questions (take default values):

```
npm init -y
```

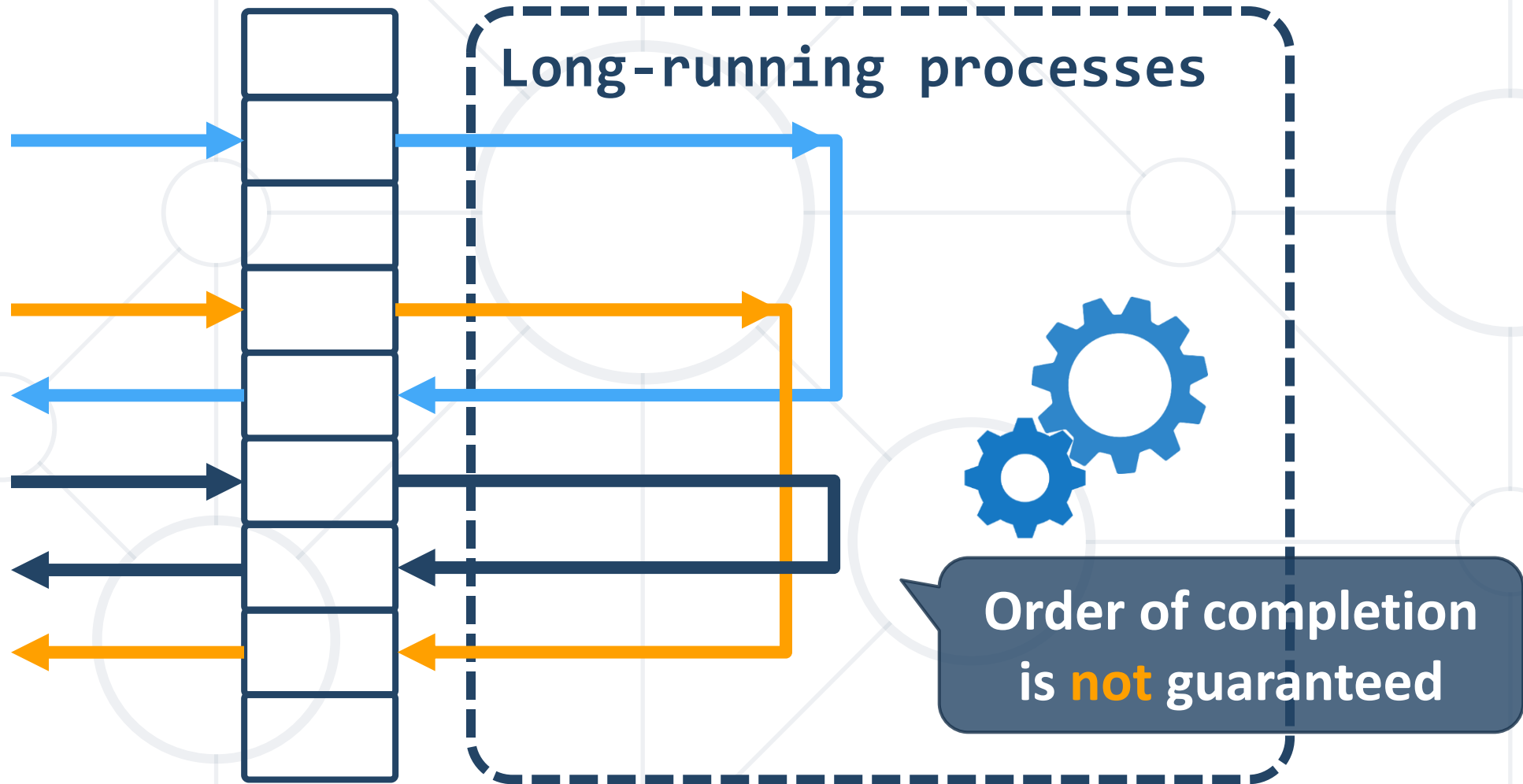

Configuration (Package.json)

```
{
  "name": "demo",
  "version": "1.0.0",
  "description": "Node.js demo project",
  "main": "index.js",
  "engines": {                // Sets versions of Node.js
    "node": ">= 6.0.0",        and other commands
    "npm": ">= 3.0.0" },
  "scripts": {                // Defines a set of node scripts
    "start": "node index.js" },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```



Event Loop

The Event Loop



Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

Stack

Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

Stack

bar(8)



Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

Stack

foo(10)

bar(8)



Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

Stack

return

bar(8)



Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

Stack

bar(8)

Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

Stack

return

Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

GC

Stack calls

```
function foo(x) {  
    return x * x;  
}  
function bar(y) {  
    return foo(y + 2);  
}  
  
bar(8);
```

Stack

```
function init(e1){  
  e1.addEventListener(  
    "click",  
    handler  
  );  
}
```

Stack

A diagram of a stack structure. It consists of a light gray rectangular box containing several circles of different sizes connected by thin gray lines. The circles are arranged in a way that suggests a hierarchical or interconnected network, with some circles acting as central nodes and others as peripheral nodes.

Browser APIs

Hidden implementation



Software
University

Stack

init

Browser APIs

Hidden implementation



Software
University

Stack

addEventListener

init

Browser APIs

Hidden implementation



Software
University

Stack

addEventListener

init

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

return

init

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

return

Browser APIs

Hidden implementation

Event Callback



Software
University

GC

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

A diagram of a stack structure. It consists of a light gray rectangular box containing several circles of varying sizes connected by thin gray lines. The circles are arranged in a way that suggests a hierarchical or interconnected network, with some circles acting as central nodes and others as peripheral nodes.

Browser APIs

Hidden implementation

Event Callback



Stack



The diagram shows a light gray rectangular area labeled 'Stack'. Inside, there is a network of faint gray lines and circles. At the bottom of this area is a dark blue horizontal bar containing three white dots '....'.

Message Queue



An orange rectangular box with a thin orange border, containing the text 'Message Queue' in dark blue.

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

....

Event Loop

Message Queue



Software
University

Browser APIs

Hidden implementation

Event Callback

Stack

....

Event Loop

Message Queue

H

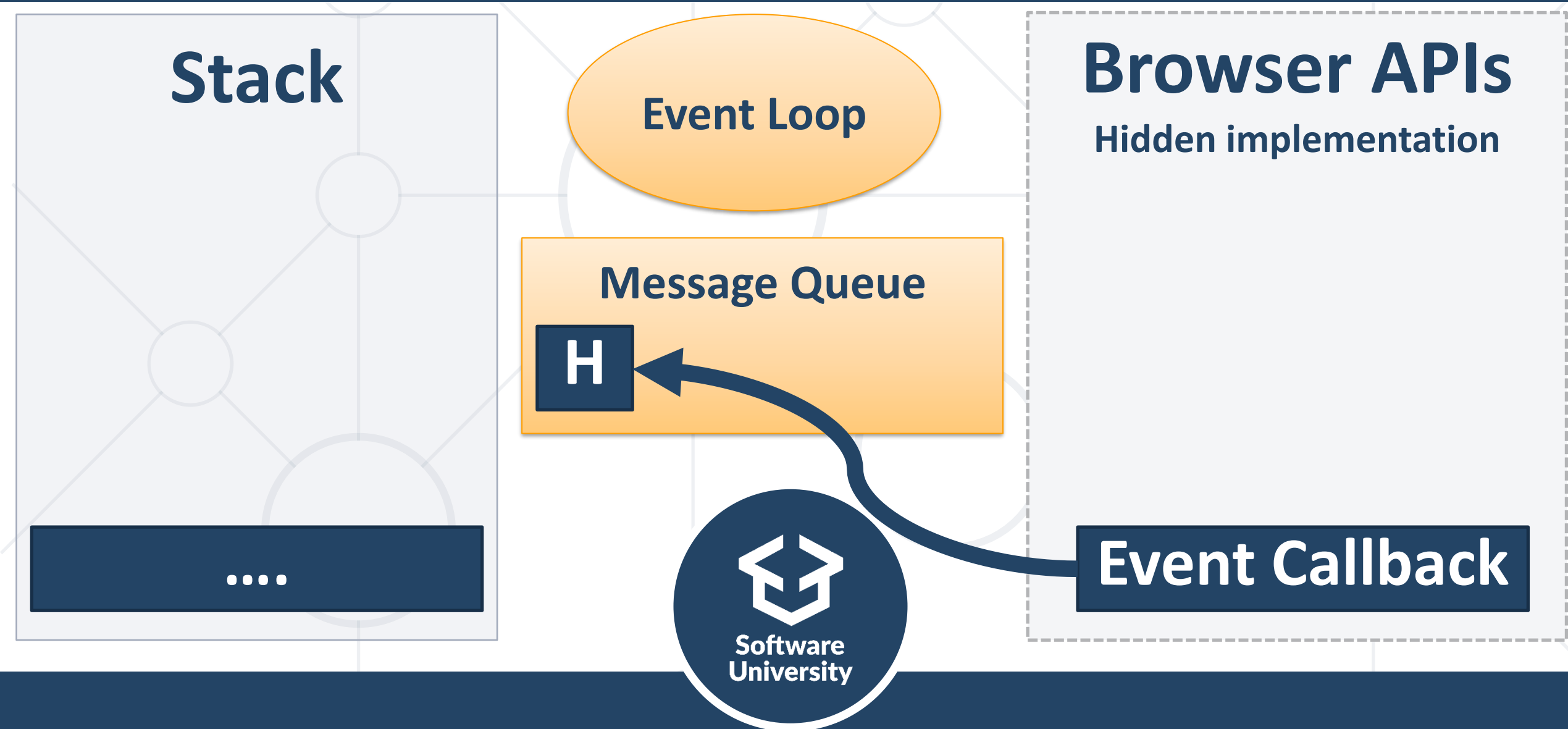
Browser APIs

Hidden implementation

Event Callback



Software
University



Stack

....

Event Loop

Message Queue

H

H

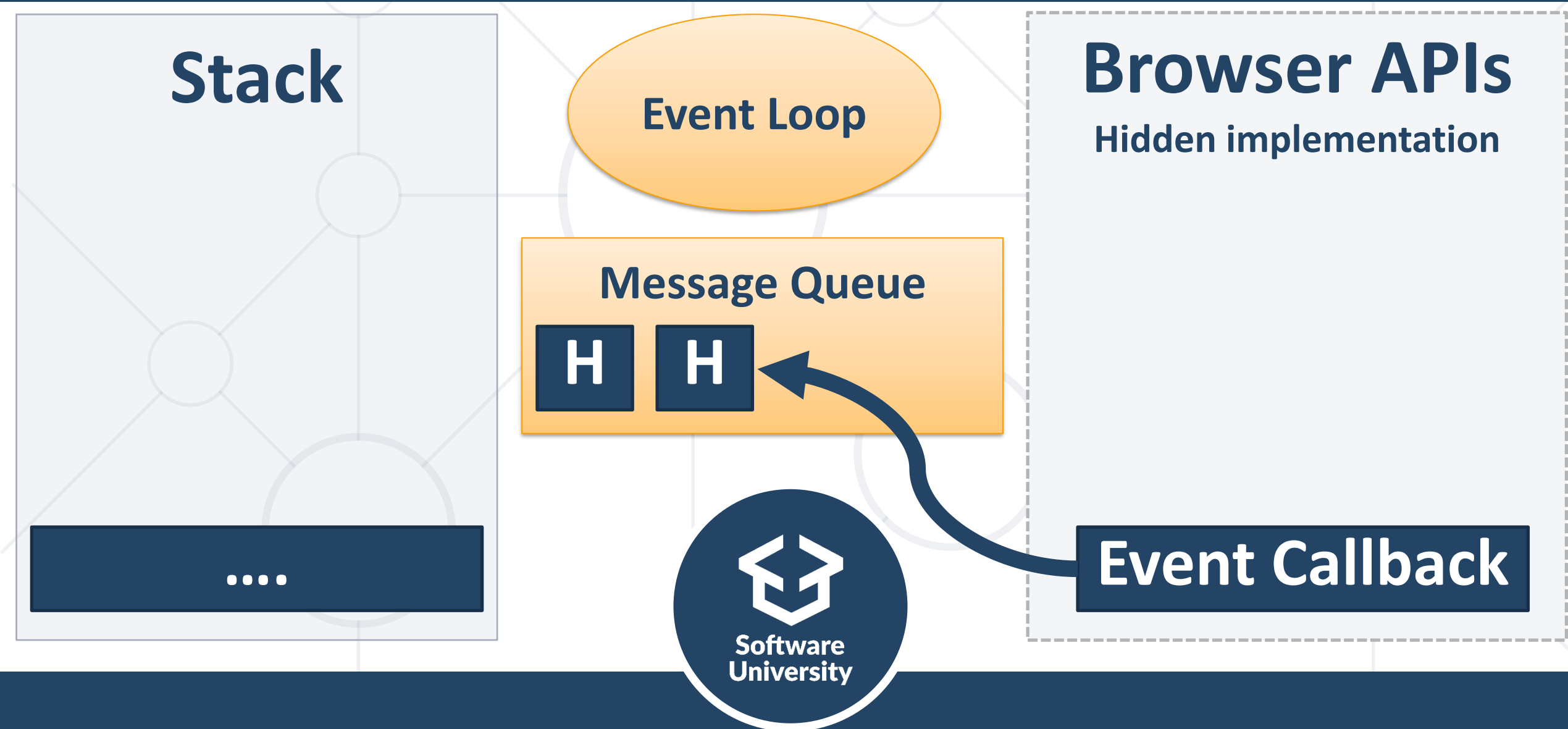
Browser APIs

Hidden implementation

Event Callback



Software
University



Stack

Event Loop

Message Queue

H

H

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

Event Loop

Message Queue

H

H

handler

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

Event Loop

Message Queue

H

handler

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

return

Event Loop

Message Queue

H

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack calls

GC

Event Loop

Message Queue

H



Software
University

Browser APIs

Hidden implementation

Event Callback

Stack

Event Loop

Message Queue

H

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

Event Loop

Message Queue

H

handler

Browser APIs

Hidden implementation

Event Callback



Software
University

Stack

handler

Event Loop

Message Queue



Software
University

Browser APIs

Hidden implementation

Event Callback

Stack

return

Event Loop

Message Queue



Software
University

Browser APIs

Hidden implementation

Event Callback

GC

Event Loop

Message Queue



Software
University

Browser APIs

Hidden implementation

Event Callback

Stack



A diagram of a stack data structure, represented as a vertical container with three circles inside, connected by lines, indicating a Last In, First Out (LIFO) arrangement.

Event Loop



An orange oval shape representing the Event Loop, which is part of the Node.js runtime.

Message Queue



An orange rectangular shape representing the Message Queue, which is part of the Node.js runtime.



Software
University

Browser APIs

Hidden implementation

Event Callback



Modules

Modules

- Allow larger **apps** to be **split** and **organized**
- Each module has its **own context**
 - It **cannot pollute** the **global scope**
- Node.js includes **three types** of modules
 - **Core** Modules
 - **Local** Modules
 - **Third-Party** Modules



Local Modules

- Created **locally** in the Node.js application
- Include **different functionalities** in **separate** folders
- Use **module.exports** to expose a **function, object** or **variable**

```
module.exports = myModule
```

- Loaded using the **require()** function

```
const myModule = require('./myModule.js');
```



Third-Party Modules

- Installed from Node Package Manager (**NPM**)
- Run from the terminal

```
npm install express --save-exact
```

- To use in your code

```
const express = require('express');
```

- To install globally (for use from the terminal)

```
npm install mocha -g
```

Core Modules

- Includes all **functionalities** of Node.js
- Load **automatically** when Node.js process starts
- Need to be **imported** in order to be used

```
const module = require('module');
```

- Commonly used modules are
 - **http** - used to create Node.js server
 - **url, querystring, path, fs**



URL Module

- Provides utilities for URL **resolution** and **parsing**

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

- Parses an address with the **parse()** function
 - Returns an **object** with **info** about the **url**

```
let urlObj = url.parse(req.url);
```

- **Splits** web address into **readable** parts



URL Parts

- Host '**localhost:8080**'

```
let host = urlObj.host
```

- Path '**/home**'

```
let path = urlObj.pathname
```

- Search/query '**?year=2017&month=february**'

```
let query = urlObj.query
```

```
let search = urlObj.search
```



Query String Module

- Provides utilities for **parsing** and **formatting** URL query strings

```
const queryString = require('queryString');
```

- Parses a query string into an object

```
const qs = queryString  
.parse('year=2017&month=february');
```

```
const year = qs.year;           // 2017
```

```
const month = qs.month;        // february
```





Node.js Web Server

Web Servers

- All **physical** servers have **hardware**
- The hardware is controlled by the **operating system**
- **Web servers** are **software** products that use the operating system to **handle web requests**
 - Web servers **serve** Web content
- The requests are **redirected to other software** products (ASP.NET, PHP, etc.), depending on the webserver **settings**



- Creating a simple Node.js web server

```
const http = require('http');  
  
http.createServer((req, res) => {  
  res.write('Hi!');  
  res.end();  
}).listen(1337);  
  
console.log('Node.js server running on port 1337');
```



Request & Response Wrappers

The Request Wrapper

- Used to **handle** incoming http requests
- Properties
 - **httpVersion** - '1.1' or '1.0'
 - **headers** - object for request headers
 - **method** - 'GET', 'POST', etc
 - **url** - the URL of the request



Request Wrapper Example

```
const http = require('http');
const url = require('url');
const port = 1337;

http.createServer((req, res) => {
  let path = url.parse(req['url']).pathname;
  if (path === '/') {
    // TODO: Send 'Welcome to home page!'
  }
}).listen(port);
```

The Response Wrapper

- Used to **retrieve** a **response** to the **client**
- Functions
 - Create **response header**
 - Send the actual **content** to the **client**
 - **End** the response



Response Wrapper Example

```
const http = require('http');
const port = 3000;

http.createServer((req, res) => {
  res.writeHead(200, { // Response Status Code
    'Content-Type': 'text/plain'
  });
  res.write('Hello from Node.js'); // UTF-8 Encoding
  res.end(); // Always End the Response
}).listen(port);
```

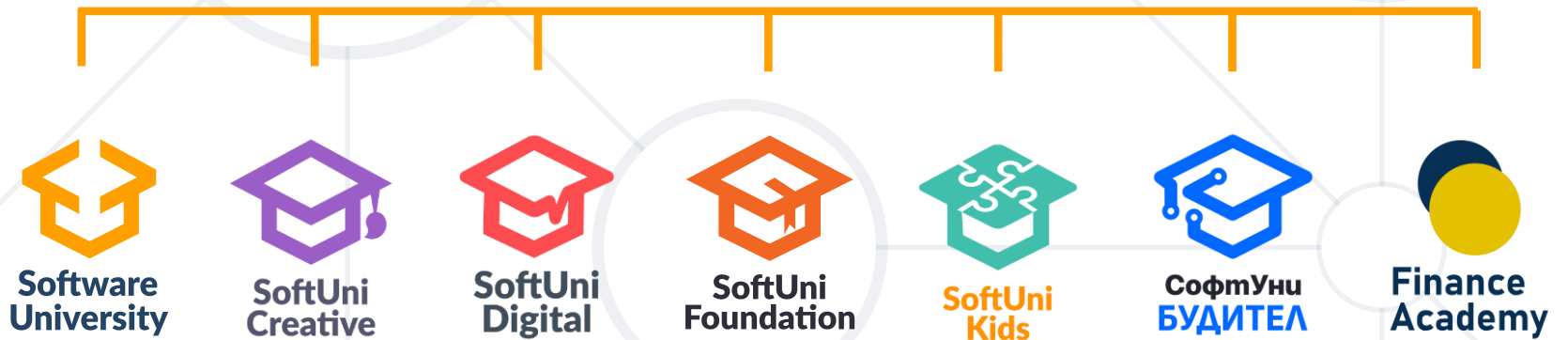
- Node.js is a **fast** and **asynchronous** efficient **package manager**
- Applications can be **organized** using **module**
- NPM allows quick access to **external modules**
- **Web Servers** transfer resources to the **Client**
- The **Request / Response** Wrappers



Questions?



SoftUni



SoftUni Diamond Partners



- Software University – High-Quality Education, Profession and Job for Software Developers

- softuni.bg

- Software University Foundation

- softuni.foundation

- Software University @ Facebook

- facebook.com/SoftwareUniversity



- This course (slides, examples, demos, exercises, homework, documents, videos and other assets) is **copyrighted content**
- Unauthorized copy, reproduction or use is illegal
- © SoftUni – <https://about.softuni.bg/>
- © Software University – <https://softuni.bg>

