Node js

moshahmed
Azularc 2017

Node.js

1. Created by Ryan Dahl in 2009
2. Development & maintenance sponsored by Joyent
3. Licence MIT
4. Version 6.11 (2017-6)
5. Based on Google Chrome V8 Engine
6. Data is generated on the server, transferred to the client and displayed by the browser.
7. ServerSide: nodejs, PHP, JSP, ASP.net, Rails, Django, java servelets.

Advantages

1. Non Blocking I/O
2. V8 Javascript Engine
3. Single Thread with Event Loop
4. Millions of modules
5. Windows, Linux, Mac
6. Same Language for Frontend and Backend
7. Active community

What is node.js?

- Asynchronous I/O framework
- Core in C++ on top of v8; remaining in js.
- 1000s of concurrent connections with minimal overhead (cpu/memory) on a single process
- NOT a web framework,
- NOT a language.

Getting Started….. Hello World

- Download and install nodejs in c:\tools\nodejs

Install modules/library

$ npm install *modulename* # local to app

$ npm install cheerio –g # for global

$ cat hello.js

```
console.log("Hello World");
```

$ c:/tools/nodejs/node.exe hello.js

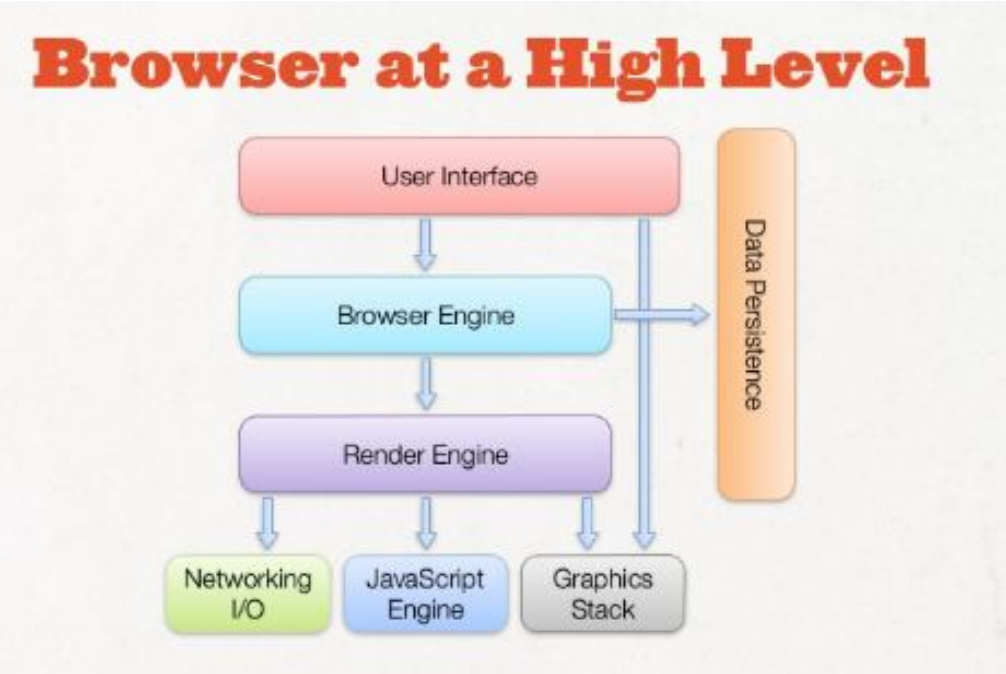
Hello World

JavaScript Engines……

Every browser has **its own VM**

Firefox? **Spidermonkey**
Internet Explorer? **Chakra**
Chrome? **V8**
Safari? **JavaScriptCore**
Opera? **Carakan**

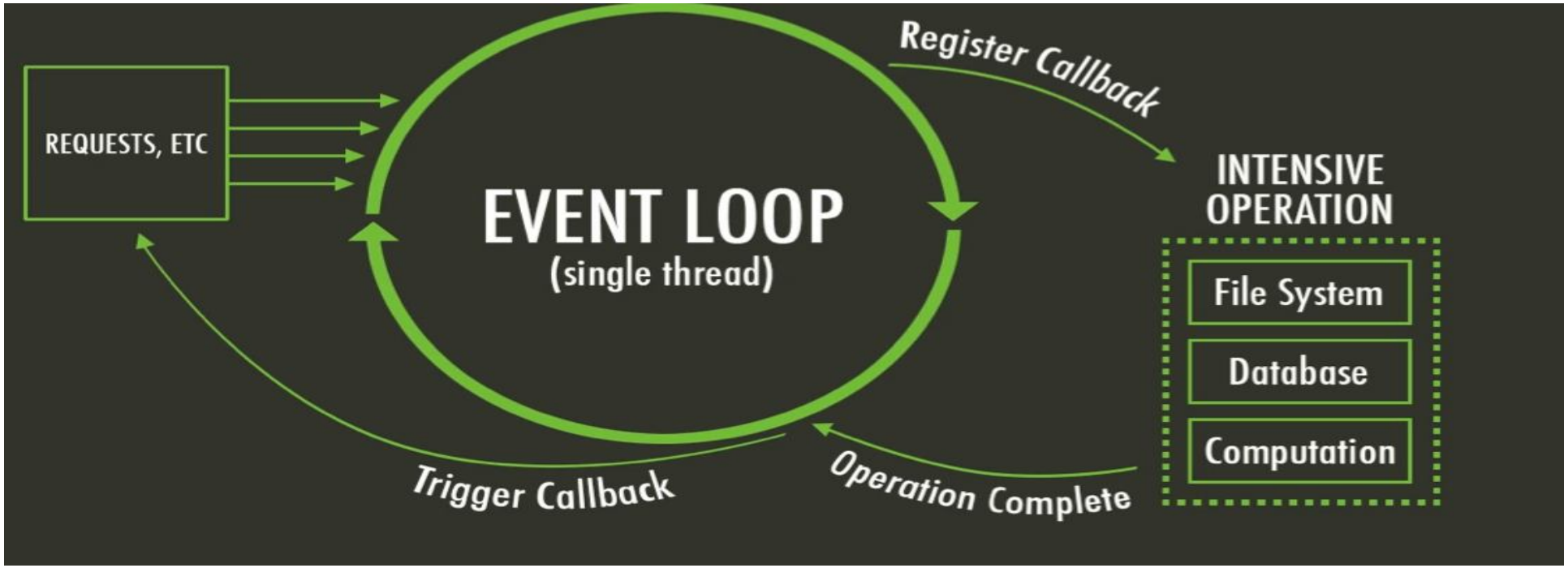
Also **Rhino**, stand alone



The ideas

1. Single thread async-processing, instead of classical multithreading. Minimize overhead & latency, maximize scalability.
2. Scale horizontally instead of vertically
3. Ideal for applications that serve a lot of requests but don't use/need lots of computational power per request.
4. Not for heavy calculations / massive parallel processing.
5. Less problems with concurrency

Node.js Event Loop

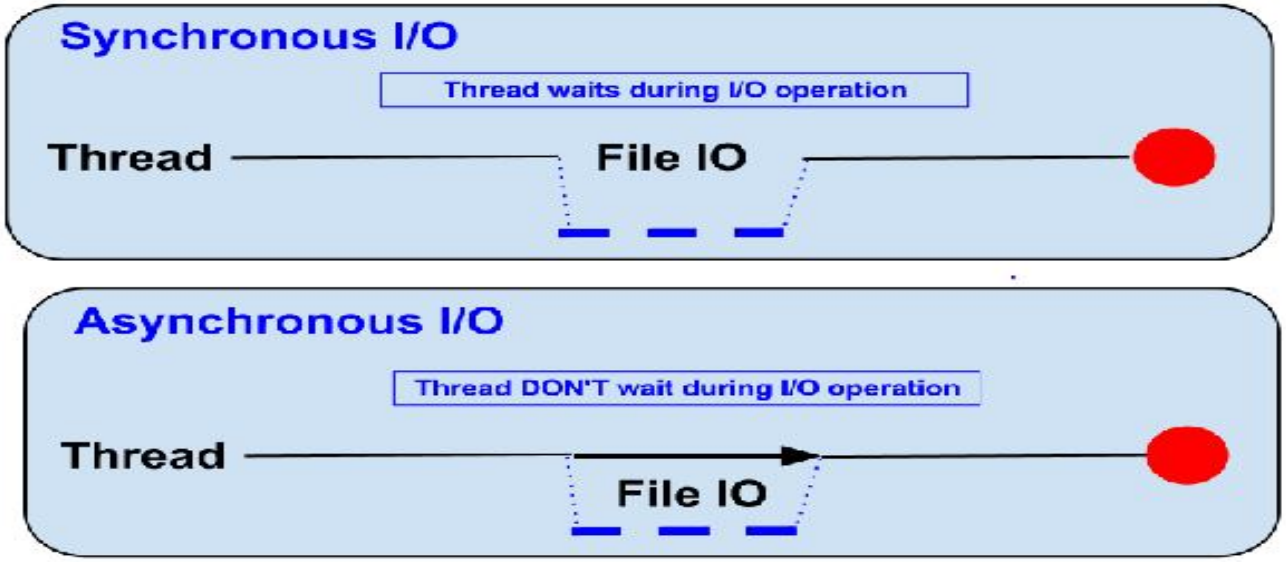


There are a couple of implications of this apparently very simple and basic model

- Avoid synchronous code at all costs because it blocks the event loop
- Which means: callbacks, callbacks, and more callbacks

Blocking vs Non-Blocking……

Example: Read data from file and show data



Using a module in main.js

```
$ npm install http fs express
$ cat main.js
    var http = require('http');
    const fs = require('fs');
    const express = require('express');
```

Blocking IO - file reader

```
// Read data from file and print on console
var data = fs.readFileSync( “test.txt” );
// blocks until file is read.
console.log( data );
```

Non-Blocking file reader with Callback

```
// Continue doing other things
// while waiting for data to be read.
fs.readFile(   “filename.txt”,  // arg1
     function  /* anonymous_callback */ (  err, data ) { // arg2
                    console.log(data);
      }
);
```

When to use it ?

- Chat / Messaging
- Real-time Applications
- Intelligent Proxies
- High Concurrency Applications
- Communication Hubs
- Coordinators

Using a module in mjs (top level async)

```
$ cat  main.mjs
   import * as mylib from './mylib.mjs';
   await console.log(mylib.f1("Hello", "World"));

$ cat mylib.mjs
    export function f1(a, b) {  return `${a} ${b}`; }

$ node main.mjs
  Hello World
```

Webserver

```
$ cat server.js
    const express = require('express')
    const app = express( )
    const port = 3000
    app.all(  '/hello',   /* url served */
        (req, res) => {   /* arrow function */
       console.log( 'serving req', JSON.stringify(req.query))
        res.send('Hello World!')
    } )
    app.listen( port , ( ) => {
      console.log(`App listening on port ${port}.`)
    })
```

Webserver

```
$ npm install express npx nodemon

$ npx nodemon  –ignore *.json  server.js
    App listening on port 3000.
    serving req {"q":"1"}

$ curl localhost:3000/hello?q=1
    Hello World!
```

Linting

```
$ npm install eslint
$ eslint --init
$ eslint main.js
```

Debug:

```
$ node --inspect-brk --inspect=0.0.0.0:3000 main.js
```

Debugging

```
$ node debug main.js
  Debugger listening on [::]:3000
    connecting to 127.0.0.1:3000 ... ok
    break in main.js:4

 debug> help
   Commands: run(r), cont(c), next(n), step(s), out(o),
     backtrace(bt), setBreakpoint(sb), clearBreakpoint(cb),
     watch, unwatch, watchers, repl, exec, restart, kill, list,
     scripts, breakOnException, breakpoints, version
debug> r
…
```

Using Postgres db

```
const pg = require('pg');
 const pool = new pg.Pool({ user: 'postgres', host: 'localhost',
      database: 'mydb', password: process.env.DBPASS, port: 5432, }

 app.get( "/time",
    async /*gettime*/ (req, res) => {
      try {
          var now = await pool.query("SELECT NOW( )");
          res.render(“timer2", { message : now });
        }catch(err){
          console.log(‘error’,err)
          res.render(“timer2”, { message: ‘error’} );
        }
     });
```

EJS (extended JS - HTML rendered server side)

$ cat views/pages/timer2.ejs

```
<div>
<% if (locals.message) { %>
    <%= message %>
<% } else { %>
   Nothing to render!
<% } %>
</div>
```

ES6

- var is hoisted, function/global scope, but value available only after assigned
- let is hoisted, decl only once, block scoped
- let and const are block scoped, only use after decl.

- for..in Loop - iterates over the index in the array.
- for..of Loop - iterates over the object of objects.

- a1 = [“a", “b"]; a2 = [“c", “d"]
- a12 = [... a1, ... a2] // spread operator 3-dots

```
for (const key in obj) { // Iterate over an object { key: value }
  console.log(`${key}: ${obj[key]}`);
}
```

References

- Introduction https://www.tutorialspoint.com/nodejs/nodejs_quick_guide.htm
- Debugging https://www.javatpoint.com/nodejs-debugger