This work is licensed under CC BY 4.0

The Node.js Platform



What is Node.js?

- ✓ Node.js is an open-source and cross-platform JavaScript runtime environment.
- ✓ Node.js runs the V8 JavaScript engine.
  - JavaScript is generally considered an interpreted language, but JavaScript is internally compiled by V8 with **just-in-time** (JIT) **compilation** to speed up the execution.
  - JavaScript has evolved from a language that was generally executing a few dozens of lines of code to complete applications with thousands to hundred of thousands of lines of code.
- ✓ A Node.js app is run in a single process.
  - Libraries in Node.js are written using non-blocking paradigms.
- ✓ Node.js has small core, but a large userland.
  - The npm registry hosts over 1,000,000 open source packages.
  - Node.js has had support for the CommonJS module system for a long time. Recently, it was added in support of ES modules.



```
import http from 'http' // core module
import cowsays from 'cowsays' // module in the node_module directory
import myModule './my-module.js' // file module relative to the file importi
```

History

- ✓ Node.js is a new kid on the block, appeared first late 2009.
  - JavaScript, appeared first late 1995.
  - The World Wide Web, invented 1989.
  - The Internet, appeared January 1983.
    - ARPAnet shifted to the TCP/IP protocol, which to this date powers the modern Internet.
- ✓ Written by Ryan Dahl.
  - Has regrets about Node.js (such as security) and created Deno, released May 2018.
- ✓ In January 2010 a package manager was introduced for the Node.js platform called npm.
- ✓ In September 2015, Node.js v0.12 and io.js v3.3 were merged back together into Node v4.0.
- ✓ …



- ✓ Original Node.js presentation by Ryan Dahl at JSConf 2009
- ✓ 10 Things I Regret About Node.js by Ryan Dahl at JSConf EU 2018

The Key Principles

- ✓ Small core
  - Node itself has a limited set of functionality. Userland modules can be found on npm.
- ✓ Small modules
  - Do one thing and do it well.
- ✓ Small surface area
  - A limited API that is clearer to use.
- ✓ Simplicity and pragmatism
  - Reduce complexity. Small modules can be composed to create larger applications.



Characteristics

- ✓ Node.js is an asynchronous event-driven JavaScript runtime, designed to build scalable network application.
  - "Single Threaded Event Loop Model" architecture.

```
// app.mjs
import http from 'http'

const hostname = '127.0.0.1'
const port = 3000

const server = http.createServer((req, res) => {
  res.statusCode = 200
  res.setHeader('Content-Type', 'text/plain')
  res.end('Hello World')
})

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`)
})
```

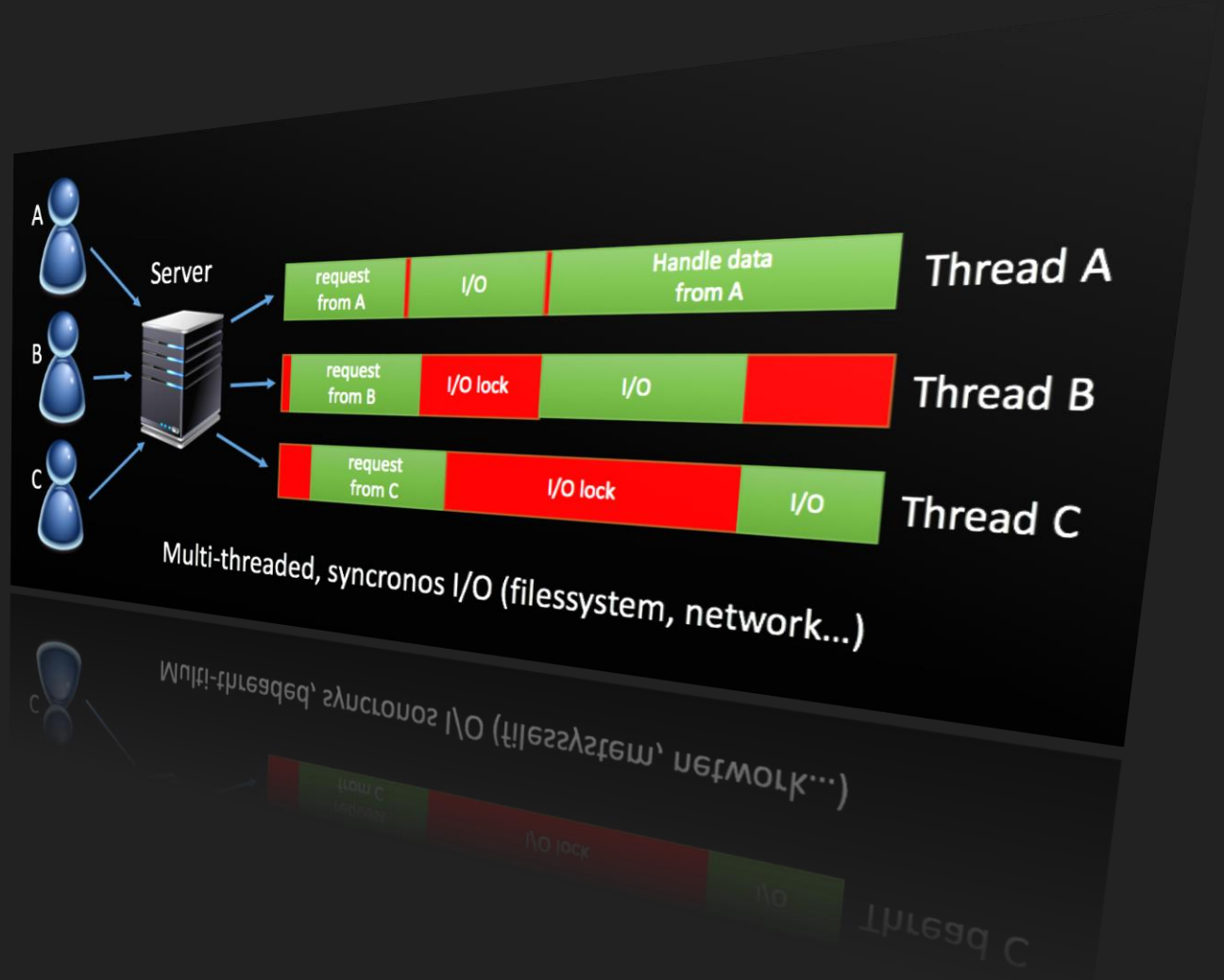
Not having a `package.json` file requires the `.mjs` extension for Node.js to use ES modules.

Node.js can handle several connections concurrently. Upon each connection, the callback is fired, but if there is no work to be done, Node.js will sleep.

- ✓ Single-threaded
- ✓ Non-blocking
- ✓ Asynchronous
- ✓ Event-driven

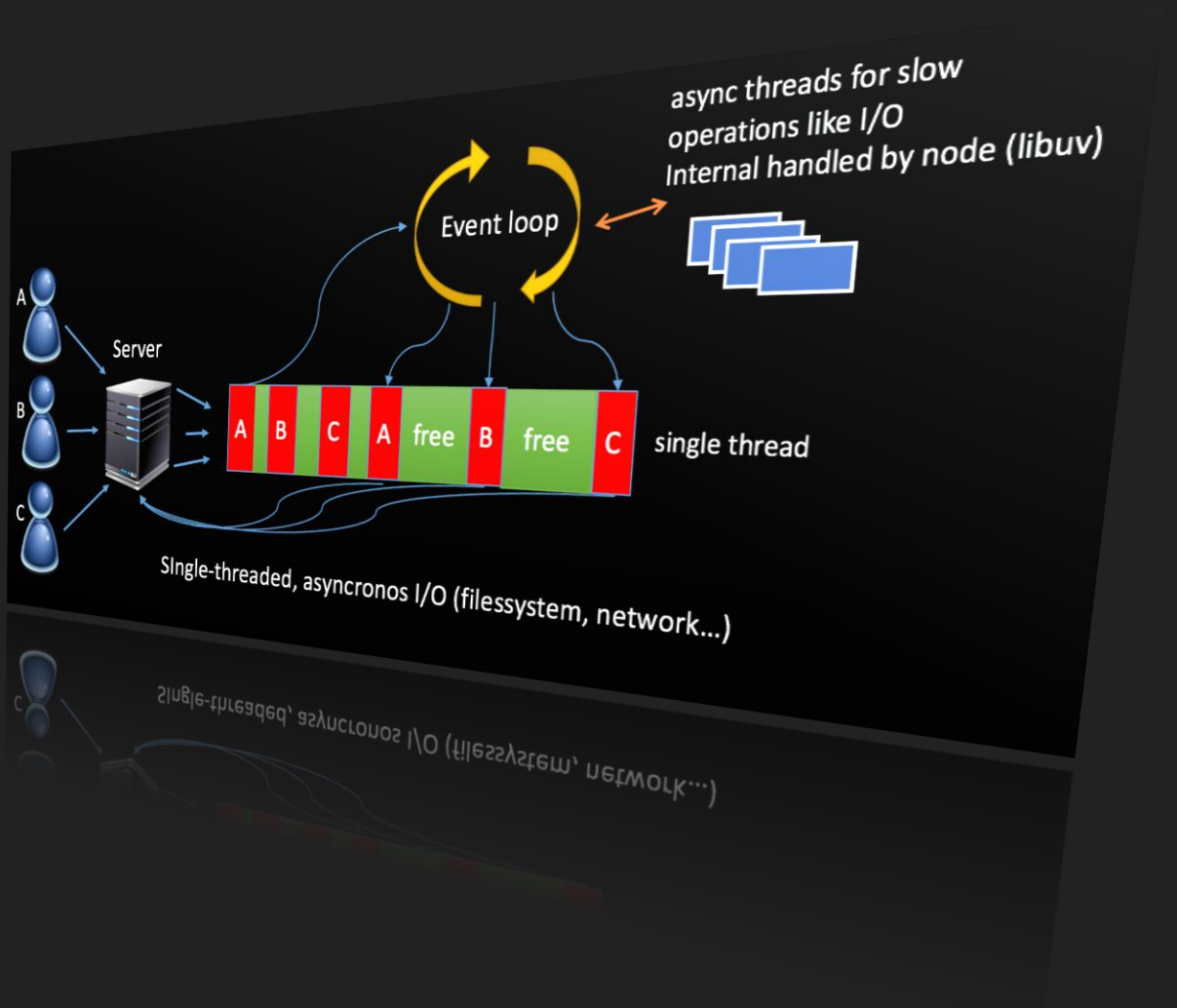
Multi Threaded Web Server Architecture

- ✓ Isolated "process" for each request
  - More traditional web servers like Apache (PHP), Tomcat (Java)
- ✓ Having a thread pool and use a thread for each request
  - Wait for available threads, taking system RAM
  - Lots of idle in each thread, waiting for I/O operations, context switching
  - Synchronous programming model
  - Problem handling many simultaneous requests



“Single Threaded Event Loop Model” architecture

- Incoming requests are handled by a single thread in a Node.js application.
- The main event loop is run on a single thread with all I/O operations running on separate threads because of its non-blocking design to utilize the event loop.
- Node’s single-threaded nature makes programs highly scalable.



Blocking vs Non-blocking

- Blocking methods execute synchronously and non-blocking methods execute asynchronously.
  - The event loop is blocked when the event loop is unable to continue running JavaScript while a blocking operation is occurring.

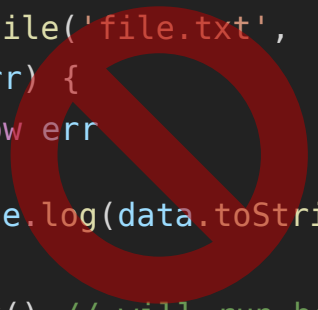
- Synchronous

```
import fs from 'fs'

const data = fs.readFileSync('file.txt') // blocks here until file is read
console.log(data.toString())
moreWork() // will run after console.log
```

- Asynchronous (old school!)

```
fs.readFile('file.txt', (err, data) => {
  if (err) {
    throw err
  }
  console.log(data.toString())
})
moreWork() // will run before console.log
```



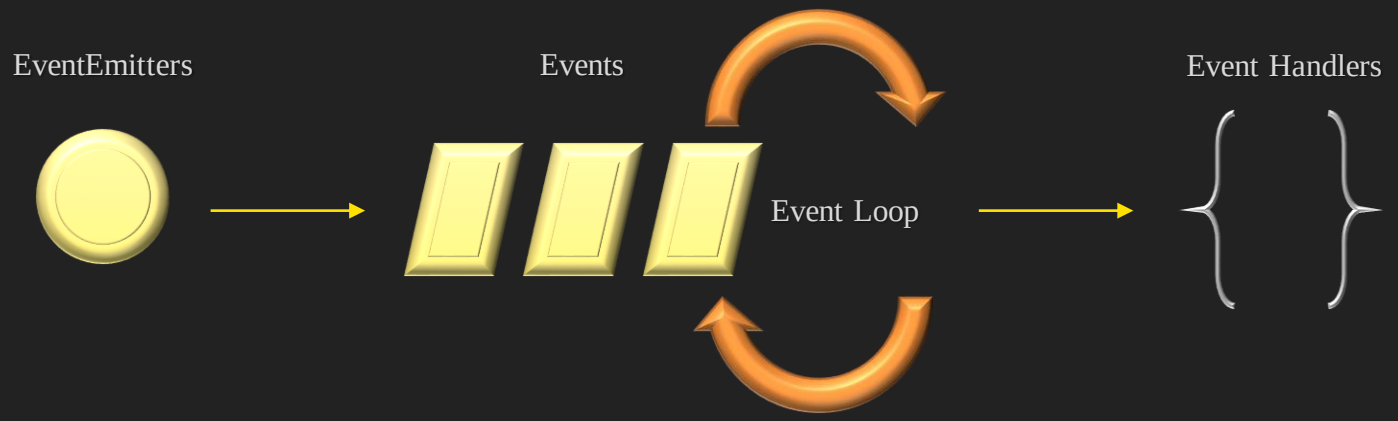
A more contemporary and promising approach!

```
import fs from 'node:fs/promises'

try {
  const data = await fs.readFile('file.txt')
  console.log(data.toString())
} catch (err) {
  console.error(err.message)
}
```

Event-driven Programming

- In an event-driven application, the main loop listens for events, then callbacks are fired, and the events are handled.



- Callbacks are great for simple cases, but the code starts to be complicated when there are many nested callbacks.
  - Promises are one way to deal with asynchronous code without getting stuck in callback hell.
  - Async functions use promises, and with the async/await syntax, asynchronous programming is a bit simpler.

When to use Node.js

- ✓ JavaScript everywhere!
  - Client-side, JSON as data format, Server-side, (from database)
- ✓ Server-side web applications
  - Traditional dynamic web site
  - Backend for Single Page Application
    - Web APIs serving JSON format
    - Front-end servers
- ✓ Realtime applications
  - Handling real time event from server to clients



When not to use Node.js

- ✓ Running code that blocks the thread.
- ✓ Not suited for requests that handle CPU-intensive tasks.
  - Tasks involving heavy calculations.
    - When a time-consuming task is running it blocks the event loop from moving forward for a longer period.
  - Use worker threads or clusters to offload heavy computations.
    - The `worker_threads` module enables the use of threads that execute JavaScript in parallel.
    - The `cluster` module allows creation of child processes.



Package manger

- ✓ npm is the standard package manager for Node.js.
  - Yarn is an alternative to npm.
- ✓ npm manages downloads of dependencies.
- ✓ npm takes care of versioning as well and follows the semantic versioning specification (SemVer).
  - Any version of a package can be specified.



Denna grafik licensieras enligt CC BY-SA

- ✓ By running `npm install` and `npm update`, you can install and update packages.
  - The `package.json` file contains entries defining what packages a project depends on.
  - A package can be installed globally.
- ✓ The `package.json` file has support for command line tasks.
  - `npm start`, `npm run lint`

package.json

- ✓ The package.json file is a kind of manifest for a project.

```
{
  "name": "exercise-promising-web-scraper",
  "version": "1.1.0",
  "description": "1DV026 - Exercise Module A - Promising Web Scarper",
  "type": "module",
  "main": "src/app.js",
  "scripts": {
    "start": "node src/app.js",
    "lint": "npx eslint ./src || exit 0",
    "lint:fix": "npx eslint ./src --fix || exit 0"
  },
  "repository": {
    "type": "git",
    "url": "git@gitlab.lnu.se:1dv026/content/exercises/exercise-promising-web-scraper.git"
  },
  "contributors": [
    "Ellen Nu <en999zz@student.lnu.se>"
  ],
  "license": "MIT",
  "private": true,
  "dependencies": {
    "jsdom": "^18.0.1",
    "node-fetch": "^3.1.0",
    "validator": "^13.7.0"
  },
  "devDependencies": {
    "@lnu/eslint-config": "^1.1.4",
  }
}
```

"name", the application name

"version", the current version

"type": "module", use ES modules

"main", entry point

"dependencies", packages installed as dependencies

"devDependencies", packages installed as development dependencies

Package versions

- ✓ All packages use version numbers like `9.0.1`, semantic versioning, or semver for short.
- ✓ The semver defines which version of a package to download and install.
- ✓ npm applies a set of rules to determine which version a package can be updated too.
  - `^`: It will only do updates that do not change the leftmost non-zero number.
  - `~`: Update to patch releases.
  - `*`: The latest version available.
  - …



package-lock.json

- ✓ The `package-lock.json` file is automatically generated for any operations where npm modifies either the `node_modules` tree, or `package.json`.
- ✓ Commit the `package-lock.json` file!
  - The `npm ci` command use the `package-lock.json` for fixed version install.



Downloading and installing packages

- The `install` command installs a package, and any packages that it depends on.
  - In most cases, the latest version is installed.
- Install packages locally
  - Installs the downloaded packages in the `node_modules` directory.
  - `npm install`
    - Installs all dependencies defined by entries, name and version, in the `package.json` file (*dependencies* and *devDependencies)*.
  - `npm install <package_name>`
    - Installs the package and adds an entry to the `package.json` file *dependencies*.
    - Packages needed in production.
  - `npm install <package_name> --save-dev`
    - Installs the package and adds an entry to the `package.json` file *devDependencies*.
    - Mainly development tools.
- Install packages globally
  - `npm install -g <package_name>`

npx – The package runner

- ✓ npx is an npm package runner that makes it easy to run a node executable.
- ✓ Calling `npx <package_name>` when `<package_name>` is not already installed locally will automatically, and temporarily, install the package from the npm registry, and invoke it.
  - Executes either from
    1. Local `node_modules/.bin`
    2. Global `node_modules/`
       - Use `npm root -g` to check where.
    3. Cache
       - Use `npm config get cache` to check where.

