JavaScript Node.js

Async programming

Url: https://github.com/obiSerra/laboratorio-nodejs-2022

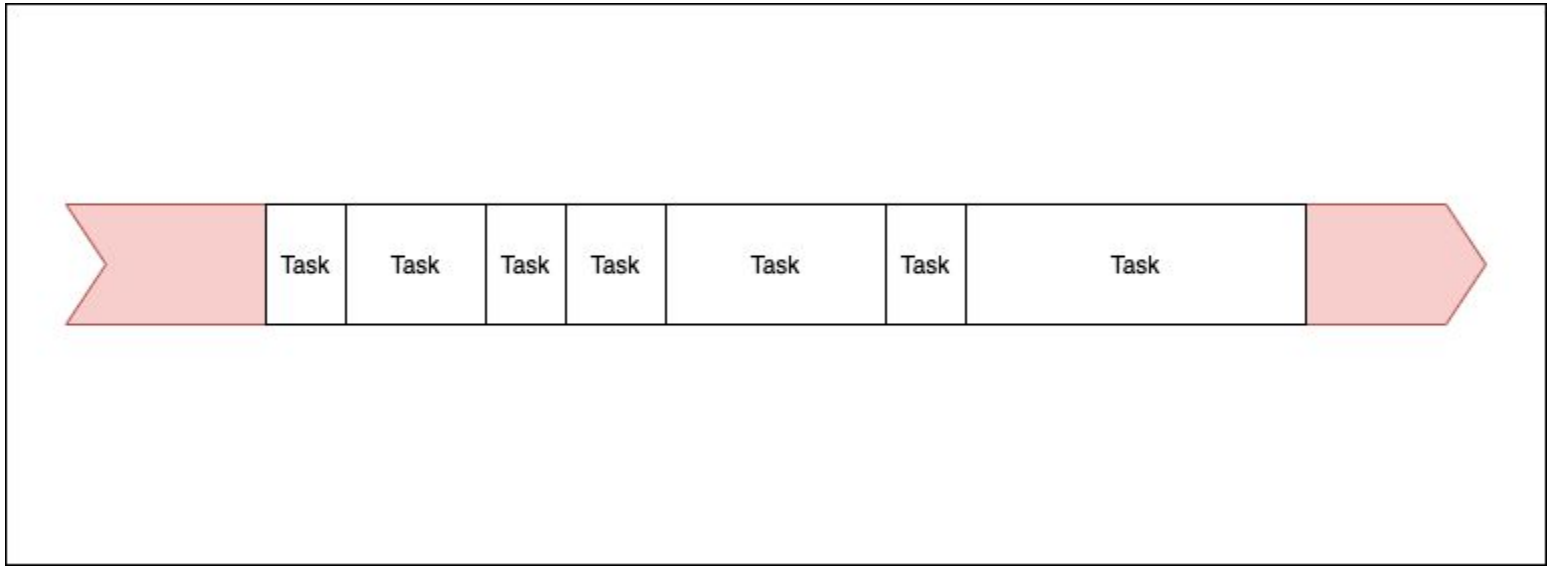
Breve riepilogo

- In NodeJs i moduli sono estremamente importanti
- Esistono moduli forniti da Node.js (path, os, fs...), moduli locali e moduli scaricati da npm
- Lo standard di Node per creare moduli si chiama CommonJs e prevede l'utilizzo di
  - *require("...")*
  - *module.exports = ...* (con la s finale)
- Esistono diversi pattern per creare moduli
  - named export
  - substack pattern
  - class export
  - instance export

Programmazione sincrona

- E’ la classica esecuzione, dove le istruzioni vengono eseguite una dopo l’altra (*lezione3/sync/index.js*)
- Task lunghi o che utilizzano risorse esterne bloccano il processo

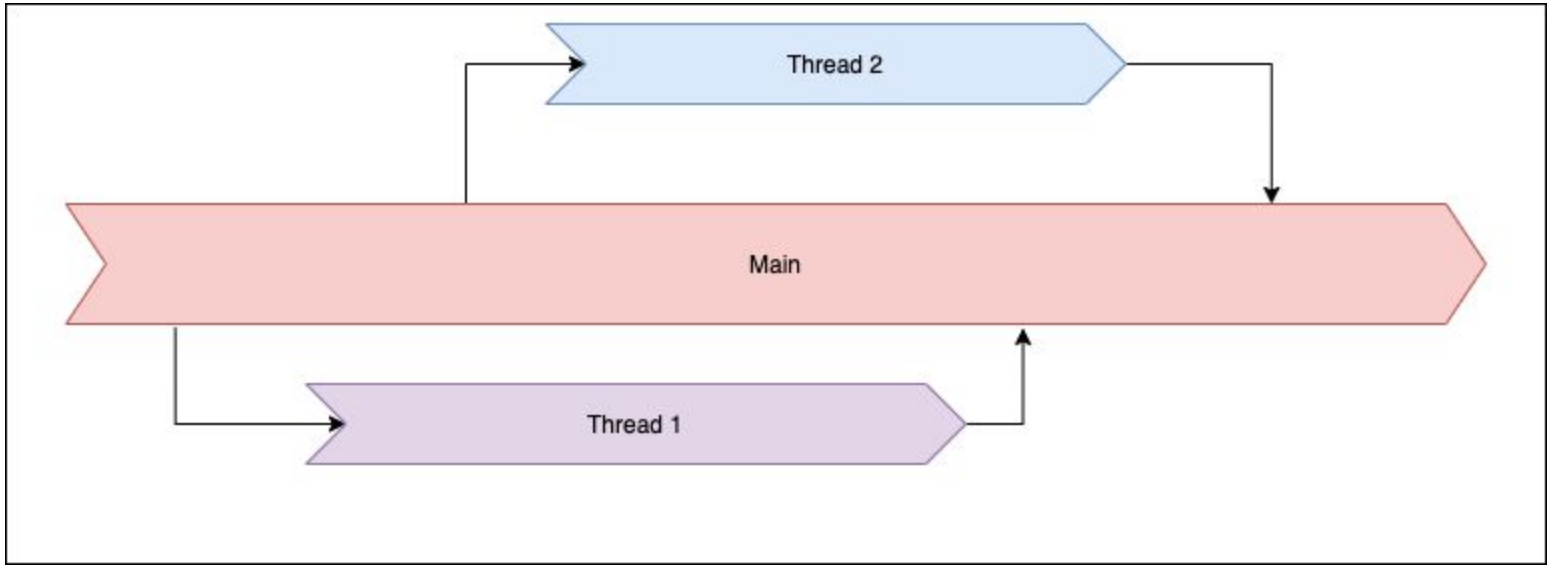
Programmazione sincrona



Threads (piccolo ripasso)

- In Computer Science un thread e’ una “sequenza” di istruzioni che possono essere eseguite all’interno di un processo
- Alcune runtime permettono di creare multipli thread (ma e’ solitamente una cosa molto complicata)
- Node invece e’ una runtime single-thread

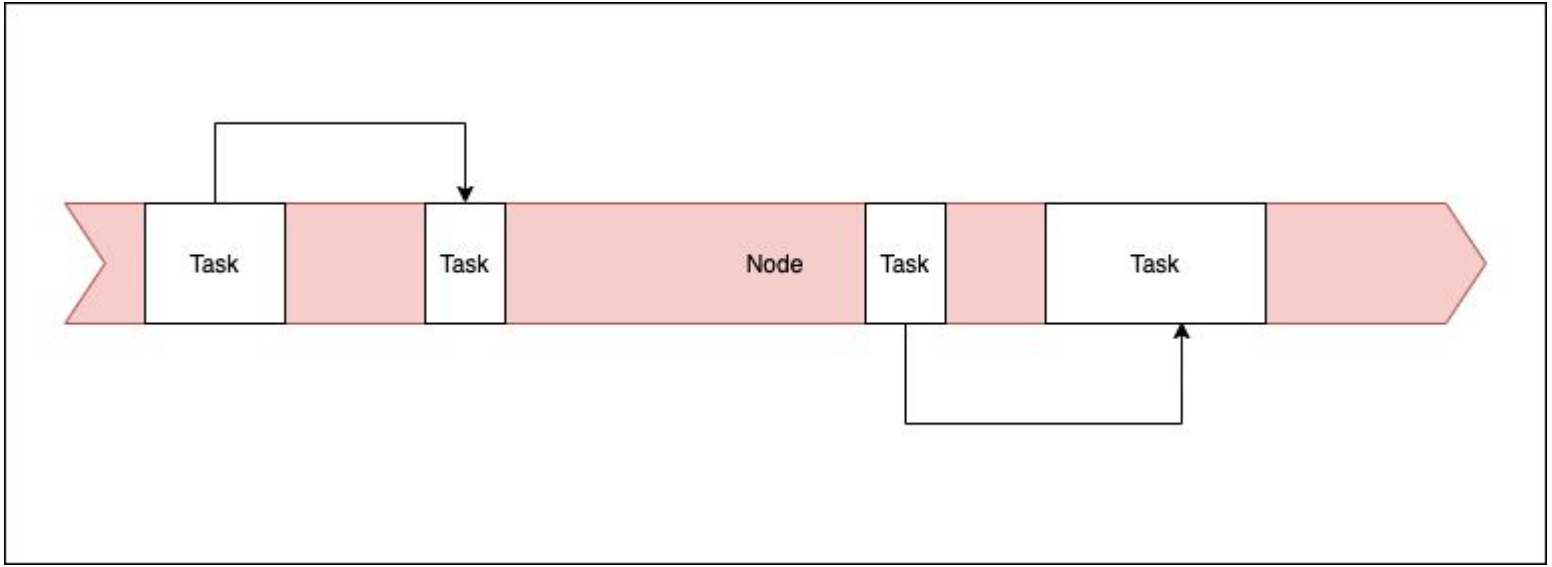
Multi-threads



NodeJs

- In Node non possiamo creare nuovi thread
- Libuv pero’ a basso livello utilizza una thread pool
- NodeJS e’ un sistema *event-driven* perche’ permette di agganciare azioni a particolare eventi (es: alla fine dell’esecuzione di un task)

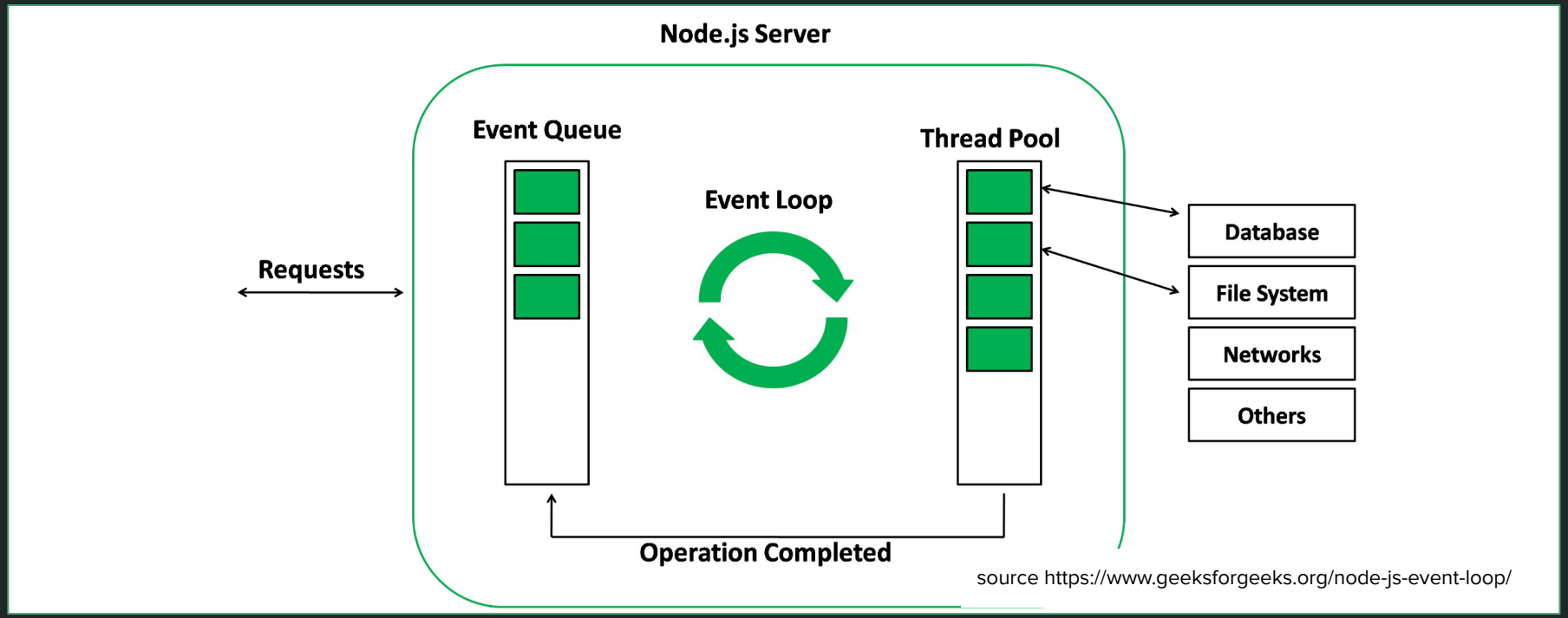
NodeJS



Event Loop

- Node (come JS lato browser) e’ basato su un meccanismo chiamato Event Loop
- L’ Event Loop e’ quello che permette a Node di eseguire *non-blocking operations*
- Si tratta di un loop eseguito continuamente e composto da diverse fasi

Node event loop



source https://www.geeksforgeeks.org/node-js-event-loop/

Programmazione asincrona

- Nonostante Node.js sia single-thread, possiamo sfruttare processi esterni in maniera “trasparente”
- In Node la programmazione asincrona sfrutta l’event loop per “schedulare” un comportamento (ad esempio una funzione) in modo che venga eseguita in un secondo momento, senza bloccare il processo principale

Callback

- Sono il metodo base per eseguire codice asincrono
- Possono essere sincrone o asincrone (attenzione a non mischiarle)
- In Node (ma non in JS) e' convenzione passare la cb come ultimo parametro
- Se il codice diventa molto complesso possiamo trovarci davanti a quella che viene chiamata la *pyramid of doom* (ovvero a un codice estremente “nested” e poco mantenibile)

Troppa teoria 0

Iniziamo a creare un tool per leggere il contenuto di un file di testo e contare quante volte appare una determinata parola

Il nostro tool

$ npm run counter <file> <world>

Troppa teoria 1

- Creiamo un nuovo progetto “counter”
- Struttura:
  - index.js
  - package.json
  - src
    - fileReader.js
- ./src/fileReader.js esporta una classe FileReader con un costruttore che prende 1 parametro (file) ed ha un metodo readFile a cui possiamo passare una callback
- index.js utilizza quella callback per contare le occorrenze del secondo parametro

Events

- *ref lezione3/esempi/events.js*
- Possono essere usati estendendo la classe Events (https://nodejs.org/api/events.html)
- E’ possibile combinare callback ed eventi (*ref lezione3/esempi/callback-events.js)*
- E’ possibile avere multipli eventi e multipli subscribers
- (Gli eventi sono molto in moduli di default o scaricati, ma e’ abbastanza raro dover creare dei custom events)

Promise

- *ref lezione3/esempi/promise.js*
- Sono nate per risolvere il problema del *callback hell* e delle *pyramid of doom (ref lezione3/esempi/pyramid.js)*
- Permettono il chaining di then
- Gestione degli errori con *.catch*
- Possono diventare complicate da gestire quando ci sono diverse “catene” in parallelo
- Promise.all fornisce un semplice modo gestire multiple risorse asincrone

Ancora troppa teoria

- Modifichiamo il nostro world counter per ritornare una Promise invece di usare una callback

Async/Await

- *ref lezione3/esempi/async-await.js*
- Ultima aggiunta al panorama di JS (e Node)
- Permettono di scrivere codice asincrono come fosse sincrono
- E’ basato sulle promise, quindi supporta anche funzionalita’ come *Promise.all()*
- Gestione degli errori con *try/catch*
- *await* puo’ essere chiamato solo in una funzione dichiarata con *async*
- *await* blocca l’esecuzione del codice che segue rendendolo di fatto sincrono