Programação na Internet

Turma i52d

Aula 9
Node Intro

Módulos/Packages

Common Js

Node.JS

V8 + Sistema de Gestão de packages (biliotecas, frameworks, etc)

V8 + npm (Repo central de packages + Ferramenta gerir dependencias)

Npm - Gestão de dependencias:

1. Download das bibliotecas
2. Configurar quais as bibliotecas de que depende uma aplicação

(registo em package.json – meta dados que descrevem a aplicação)

NPM

Npm

- **`npm init`** -- inicializar o package.json
- **`npm install`** – instala todas as bibliotecas referidas no package.jon
- **`npm install <nome módulo>`** - nome unico || instalação local à directoria
- **`npm install -g …`** - instala globalmente
- **`npm install --save`** - grava no package json informação do **módulo**

Módulos para pedidos HTTP

Alternativas:

- **`http`** – modulo standard do node
- **`fetch`** – funcionalidade nativa nos Browsers para pedidos HTTP
- **`node-fetch`** – a versão do fetch do Browser para node
- **`urllib`**
- **`request`**
- **`then-request`**
- Etc…

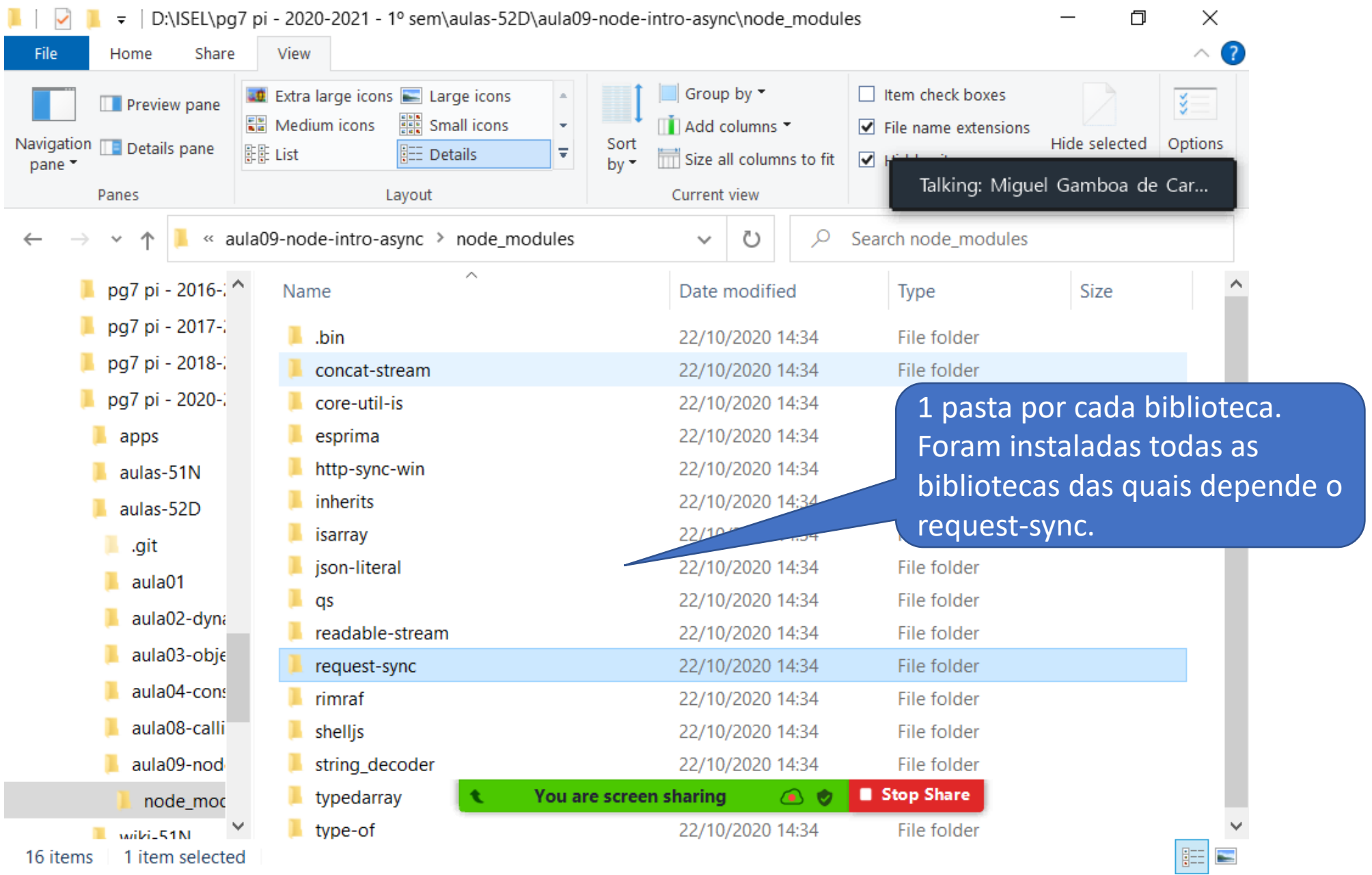
Npm e.g.

npm install --save request-sync

```
{
  …
  "dependencies": {
    "sync-request": "^6.1.0"
  }
}
```

Semantic Versioning:

- Major => quebra de compatibilidade
- Minor => nova funcionalidade
- Patch => correcção de Bugs


D:\ISEL\pg7 pi - 2020-2021 - 1º sem\aulas-52D\aula09-node-intro-async\node_modules
File
Home
Share
View
Navigation pane
Preview pane
Details pane
Panes
Extra large icons
Large icons
Medium icons
Small icons
List
Details
Layout
Sort by
Group by
Add columns
Size all columns to fit
Current view
Item check boxes
File name extensions
Hide selected
Options
Talking: Miguel Gamboa de Car...
aula09-node-intro-async
node_modules
Search node_modules
pg7 pi - 2016-
pg7 pi - 2017-
pg7 pi - 2018-
pg7 pi - 2020-
apps
aulas-51N
aulas-52D
.git
aula01
aula02-dyna
aula03-obje
aula04-cons
aula08-calli
aula09-nod
node_mod
wiki-51N
Name
Date modified
Type
Size
.bin 22/10/2020 14:34 File folder
concat-stream 22/10/2020 14:34 File folder
core-util-is 22/10/2020 14:34 File folder
esprima 22/10/2020 14:34 File folder
http-sync-win 22/10/2020 14:34 File folder
inherits 22/10/2020 14:34 File folder
isarray
json-literal 22/10/2020 14:34 File folder
qs 22/10/2020 14:34 File folder
readable-stream 22/10/2020 14:34 File folder
request-sync 22/10/2020 14:34 File folder
rimraf 22/10/2020 14:34 File folder
shelljs 22/10/2020 14:34 File folder
string_decoder 22/10/2020 14:34 File folder
typedarray
type-of 22/10/2020 14:34 File folder
1 pasta por cada biblioteca. Foram instaladas todas as bibliotecas das quais depende o request-sync.
You are screen sharing
Stop Share
16 items
1 item selected

Utilização do modulo em Javascript

Common JS – importar um modulo:

- **`require(<string nome do módulo>)`** => retorna um objecto

(**`package`** = modulo distribuido no repositorio central NPM)

Single-thread

Concorrente

Assíncrono

Node.js

- **Single-thread**

- **Concorrente – Simultaneadade – Execução de operações simultaneas**

- Assíncrono != Sync

(resultado != retorna) (resultado função = retorno => bloqueio)

CONCORRENCIA != PARALELISMO

Não bloquear != (Executar Tarefas em Paralelo e.g. Stream.parallel(), thread pool, etc…)

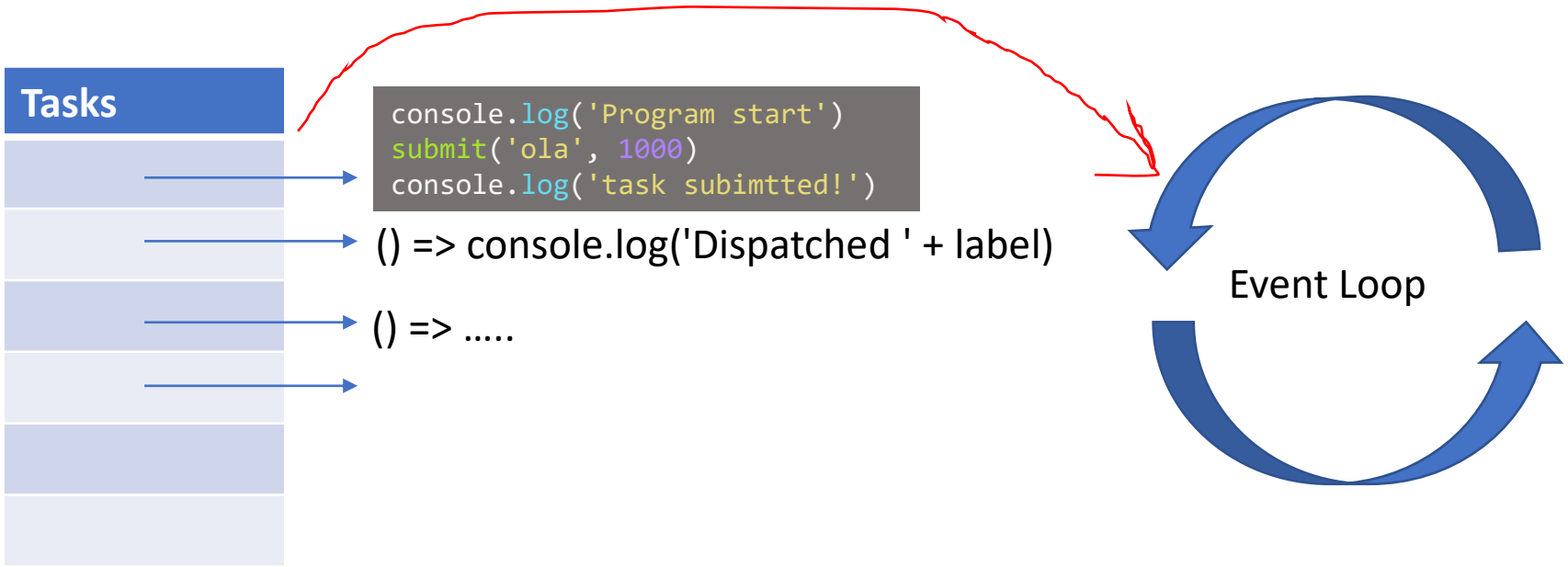
Node.js

Node.js:

- Google's **V8 JavaScript engine**
- ****event loop**** (e.g. setTimeout())
- ****low-level I/O API**** => não bloqueante / assíncrona

Event loop

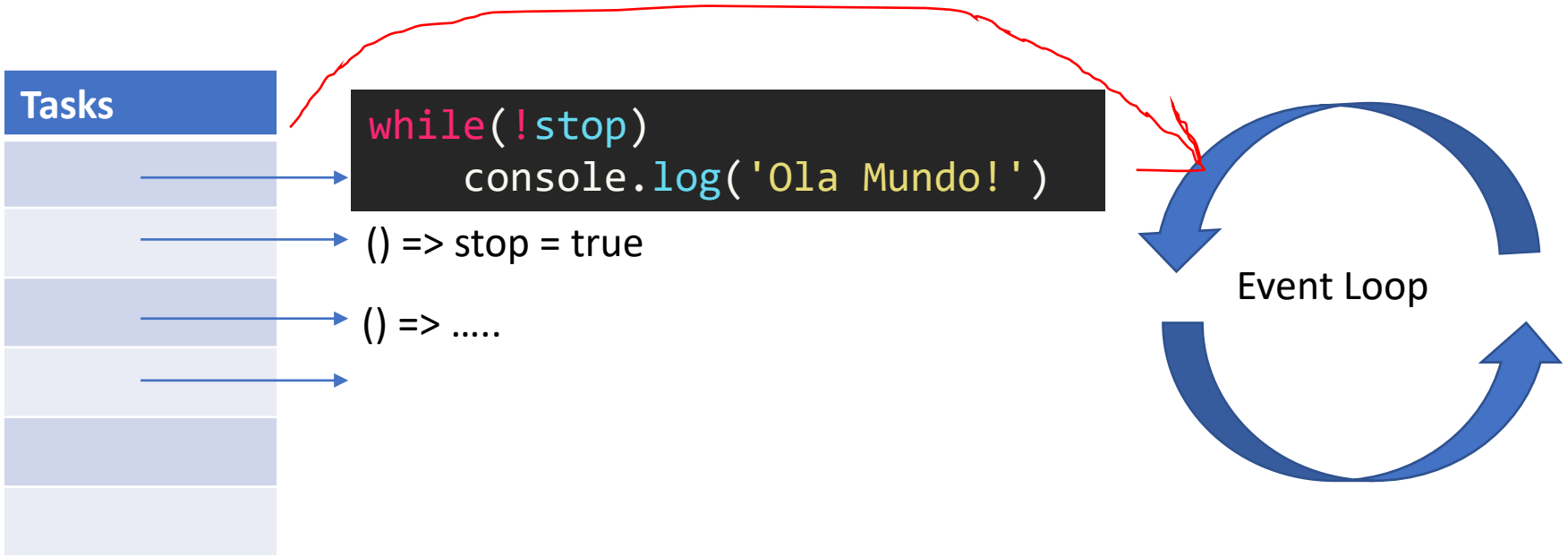
Aplicação termina quando não existirem mais Tarefas



E.g. setTimeout(…)

Event loop

Aplicação termina quando não existirem mais Tarefas



E.g. setTimeout(…)

Node

- Felix Geisendörfer: “***everything runs in parallel except your code.***”
- Vários **callbacks** podem responder a um mesmo evento, **mas só um** é executado em cada instante – **sequencialmente**.
- Os **callbacks NÃO são interrompidos por outros callbacks**, nem correm em paralelo com outros callbacks.
- Do ponto de vista da linguagem JavaScript o Node.js oferece um **ambiente single-threaded**.
- Todas as operações de IO do Node.js podem ser executadas em simultaneo e.g. ler ficheiros; pedidos HTTP,….
- As operações de IO usam técnicas **non-blocking**.

Sync <versus> Async

Síncrono => Resultado de F() ==== retorno de F

Assíncrona => Resultado de F() != retorno de F

Assíncrono idiomas:

- Callbacks – génese do Javascript
- Promises (aka CompletableFuture em Java e Task em .net)
- async await (except em Java)
- Suspend functions (e.g. Kotlin)

Callbacks em Javascript

- **Callback** é uma função
- **Callback** é o ultimo parametro da função (convenção)
- **callback** recebe como 1º argumento erro (convenção):

```
(err, data, …) => { if(err) return….  }
```

> SUCESSO – err é undefined

> ERRO – err refere um objecto Error

!=

Sync:

- Resultado = retorno da função.
- Erro = Excepção