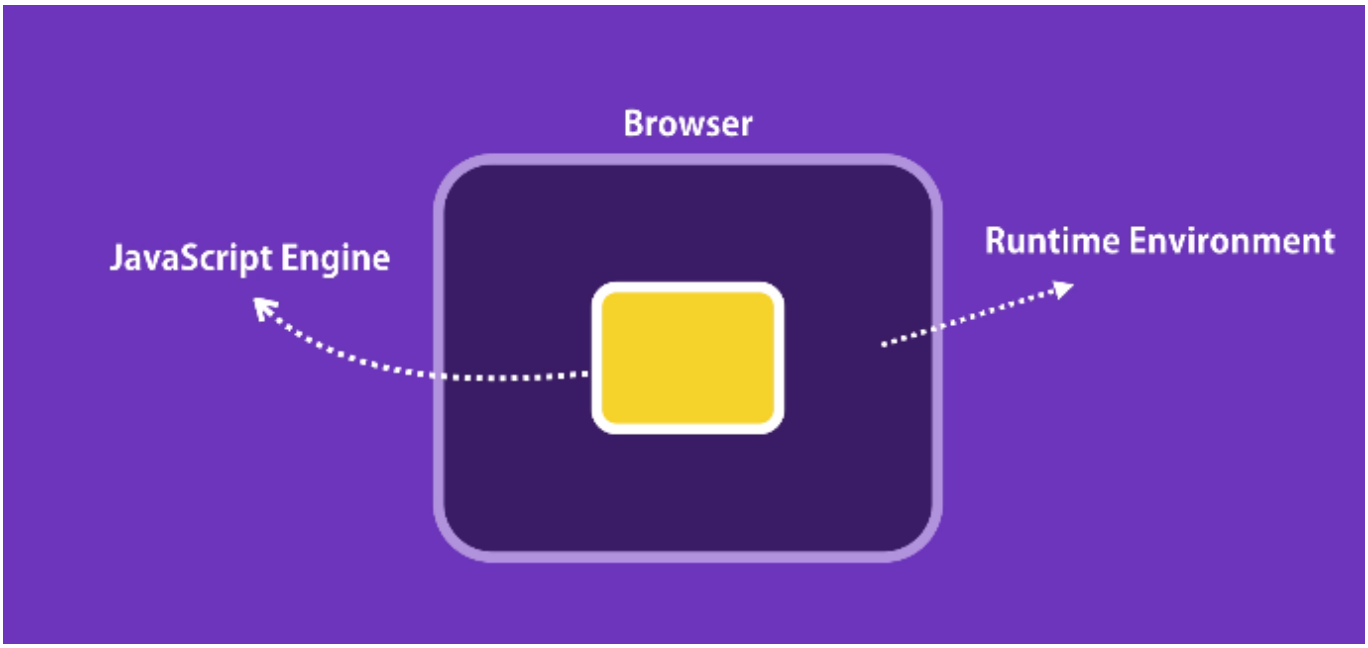
node

É um **Runtime Environment** (Ambiente de Execução) que permite o uso de Javascript **fora do navegador**

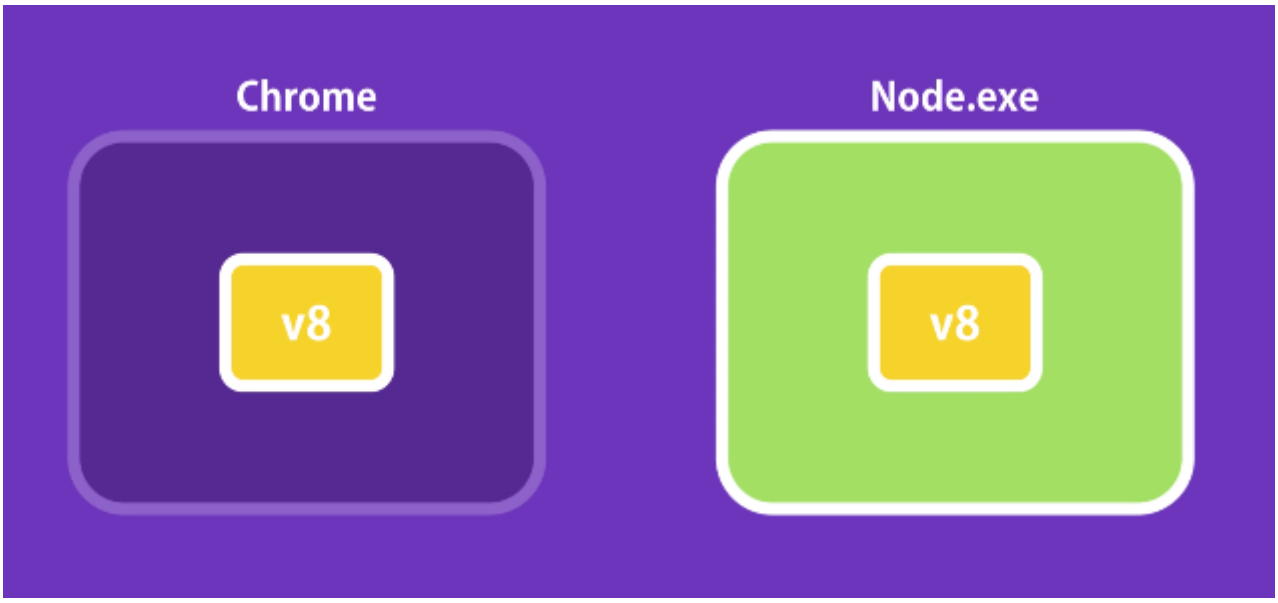
Javascript em Browsers vs Node

Originalmente, javascript só podia ser executado via navegador. Para tal, cada um possui sua própria **JS Engine** (Ou **interpretador javascript**), que interpreta o código em **código de máquina** e o executa, o que faz de JS uma linguagem **interpretada**. O próprio **interpretador é quem se comunica com a CPU e executa o código de máquina**, ao contrário de C (Que é uma linguagem compilada) onde o GCC/**compilador apenas compila o código para código de máquina**, que deve ser executado posteriormente pelo próprio usuário. Exemplos de JS engines: Chackra(Edge) SpiderMonkey(Firefox) e **V8(Google Chrome)**.



O **runtime Environment não é a Engine**, mas todo o **ambiente e funções providas** pelo mesmo que complementam o funcionamento de Engine e permitem um uso mais completo e específico da linguagem. Portanto, **navegadores também possuem Runtime Environment** para Javascript, que são **limitados pelo escopo do Browser** (Podem acessar elementos HTML pelo id ou poppar caixas de alerta, procedimentos que fazem sentido no contexto do navegador, mas não podem, por exemplo, acessar arquivos do HD do usuário, bem como a maioria das funções de sistema no geral).

Node utiliza apenas o **V8 do Chrome**, envolvendo-o em **código c++** para criar um runtime environment **alheio ao navegador** com **acesso a funções do sistema**, desta maneira permitindo uma execução **nativa** e com **funções essenciais** para **servidores**.



Ou seja, o Chrome e o Node possuem **a mesma JS Engine**, mas promovem **diferentes ambientes de execução** para javascript.

Por exemplo:

```
let my_div = document.getElementById("div_principal")
```

É um programa que **funciona apenas em navegadores**, pois `document.getElementById` é definida pelo runtime environment dos mesmos. Já:

```
const fs = require('fs')

fs.readFile('/Users/joe/test.txt', (err, data) => {
  if (err) {
    console.error(err)
    return
  }
  console.log(data)
})
```

Utiliza o

Servidor em node vs servidor "tradicional"

Node utiliza a abordagem **non-blocking** para atingir baixa latência mesmo com altas taxas de transferência (throughput).

Em **servidores "tradicionais"**, cada nova requisição gera (spawns) uma **nova thread**, ou mesmo clona (forks) o processo atual em um **novo processo**, que lidam e respondem a mesma. Na prática, a criação de novas threads é menos custosa (Em memória e CPU Overhead) que a criação de novos processos, mas, ainda assim, os ciclos gastos com **thread scheaduling** e **context switching** podem gerar sobrecarga em

sistemas com muitas threads, o que aumenta a latência, impondo limites em escalabilidade e taxa de transferência.

Fontes:

.1 https://www.infoworld.com/article/3210589/what-is-nodejs-javascript-runtime-explained.html

.2 https://www.udemy.com/course/nodejs-master-class