CAMPINAS TECH
TALENTS

REALIZAÇÃO:

CAMPINAS TECH
novofuturo tech
share rh
Valor compartilhado em recursos humanos

PATROCÍNIO:

sensedia

Como funciona o NodeJS

Event loop

O código javascript do NodeJS é single thread, ou seja, apenas uma coisa pode ser executada por vez

Essa desvantagem reduz os problemas de simultaneidade por gerenciar apenas um processo, por exemplo;

O Javascript possui um conceito de eventos que é extremamente simples, porém é compilado para C++

Cada evento fica armazenado em uma fila (event queue)

Event loop

Nesse meio tempo de execução temos:

→ Os eventos do javascripts (são denominados binds)

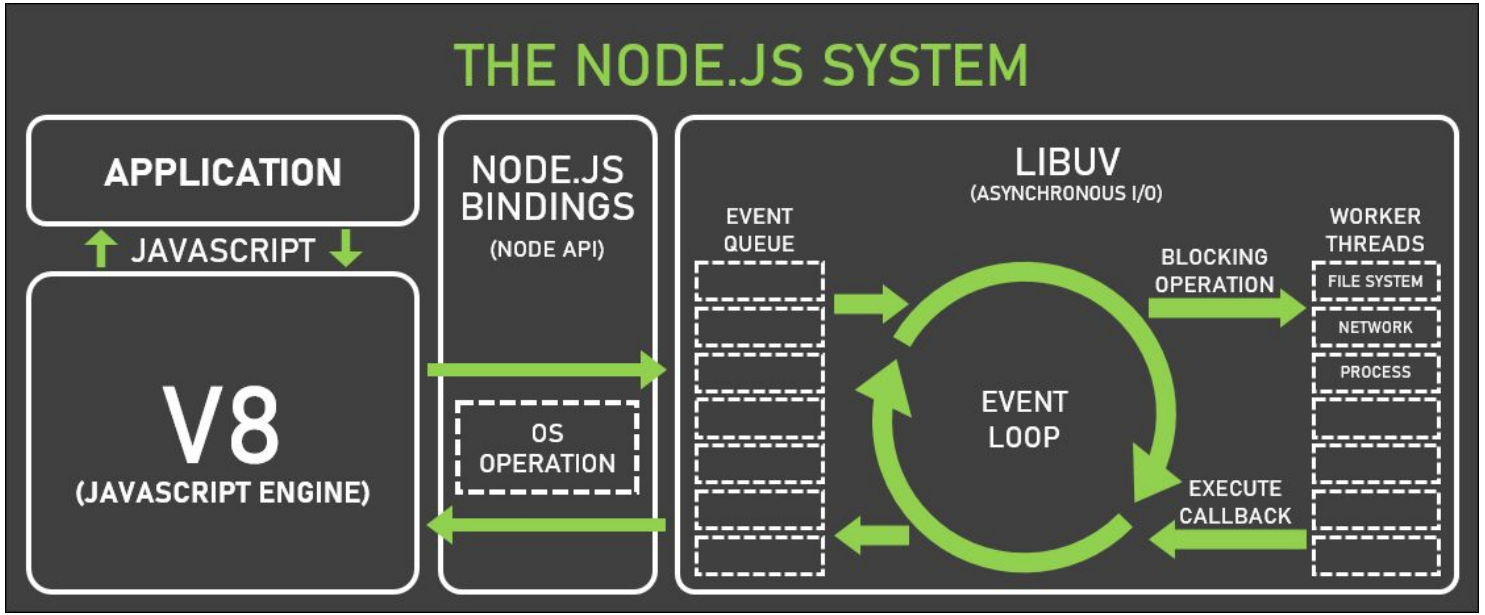
→ A camada de aplicação que manda a requisição

→ O motor V8 que executa o Javascript

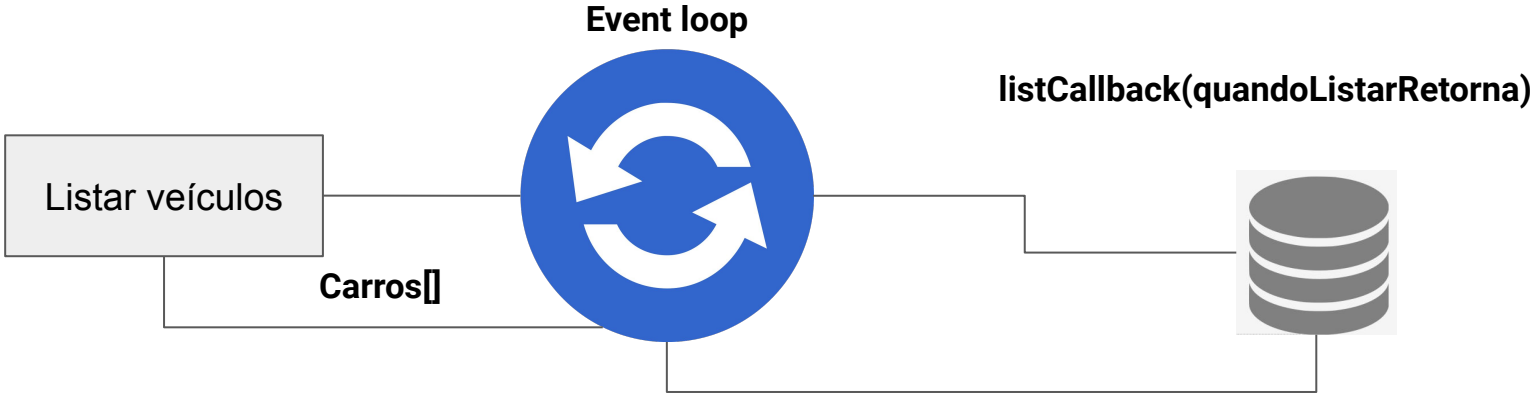
→ A camada de transpilação de javascript para C++

→ Que executa então a partir daí a LibUv com foco em I/O assíncrono

Fluxo de eventos

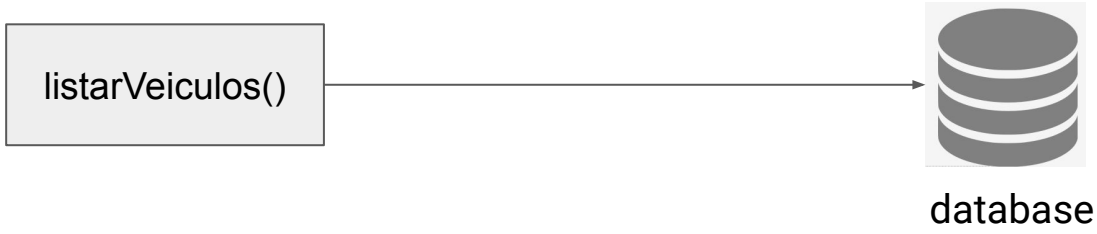


Trazendo pro mundo real =)



Ciclo de vida do Javascript

→ Todas as chamadas que dependem de interação com o mundo externo serão executadas em background



Background (segundo plano)

A forma com que ele é escrito é diferente de como ele é executado nesse caso!

Explicando na prática

Arrow functions

Arrow functions

→ É uma forma mais curta que agiliza a hora de programar;

→ São menor verbosas;

→ é conhecida como "arrow" pela declaração "**=>**"

→ Simplifica o escopo de blocos de código;

→ Não precisa declarar **function**, nem **return;**

Explicando na prática

Revisando interações de array

Explicando na prática

Exercícios

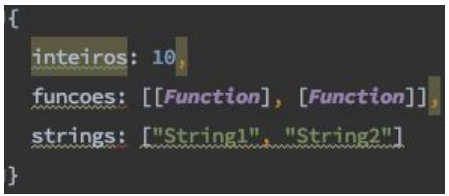
Nível I:

1 - Iniciar um novo projeto com o nome exercicios-de-array;
2 - Instalar o pacote nodemon;
3 - Criar o script "dev" para executar a rotina "nodemon index.js";
4 - Em **index.js:**
5 - Criar uma função chamada "preencherArray()" que retorne a criação de um array com n de posições passadas no array;
6 - listar a quantidade de posições do array;
7 - Criar uma função chamada "parOuImpar()" que passe a variável "tipo" e permita filtrar valores de um array entre par ou impar;
8 - Chamar a função par ou impar, e mostrar todos os valores em apenas uma linha;
9 - Criar um array chamado vemDeTudo. Crie uma função chamda "jogaTudoPraDentro()" que permita armazenar qualquer tipo de elemento, objeto, número ou função dentro do array vemDeTudo;

Nível II:

10 - Pegando o Array "vemDeTudo", adicionar posições nele no formato inteiro, string e funções. Criar uma nova função chamada processamento, onde:

> Se for número, deverá multiplicar os valores,
> Se for Array, armazená-lo em um array,
> Se for uma função, armazená-lo em outro array.
> O retorno desse processamento deve ser retornado em um objeto, como na imagem



Promises

Promises

→ Promises (promessas), traduzindo pro melhor português possível é que: vamos fazer uma promessa que em algum momento será cumprida e teremos um retorno xpto.

→ É um objeto javascript que ajuda nosso código assíncrono ser menos complexo e verboso

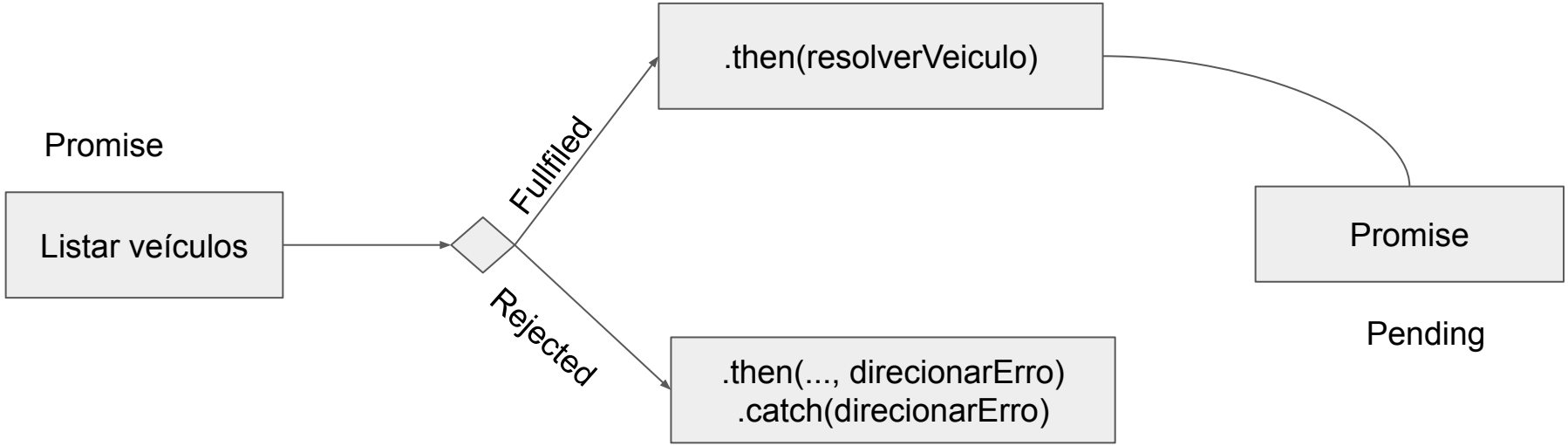
Ciclo de vida de uma Promise:

→ Ao instanciar uma Promise, ela sempre entra como o estado inicial **"PENDING",** logo ela não terminou ou não foi rejeitada

→ Ao executar as operações com sucesso, a Promise retorna pro código o status "**FULLFILED**"

→ Caso de falha, o retorno sempre será "**REJECTED**"

Trazendo pro mundo real =)



Promises Async/await

Promises async/await

→ O async/await é uma nova forma de tratar Promises evitando o cascateamento com **.then**

→ Ou seja, o async/await garante que o comportamento seja assíncrono baseado em Promises;

→ O código fica mais simples por conta de encadeamento;

→ Async/await não altera a performance de sua aplicação;

→ Toda função await poderá ser utilizada apenas em funções com a palavra chave **async**;

→ Deverá ser utilizado quando dependermos de uma resposta por uma função terceira;

```
async function index() {
    const veiculo = await listarVeiculo();
    listarPlaca(veiculo);
}
```