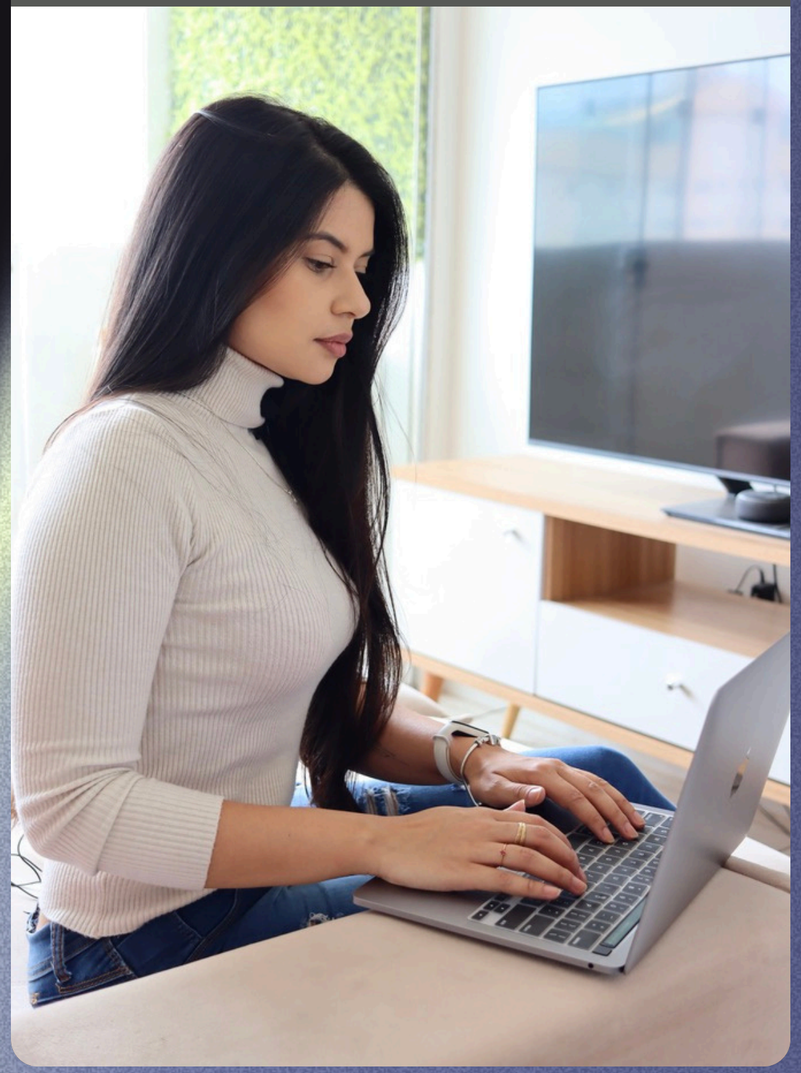
Concorrência e Threads no Node.js: além do Event Loop

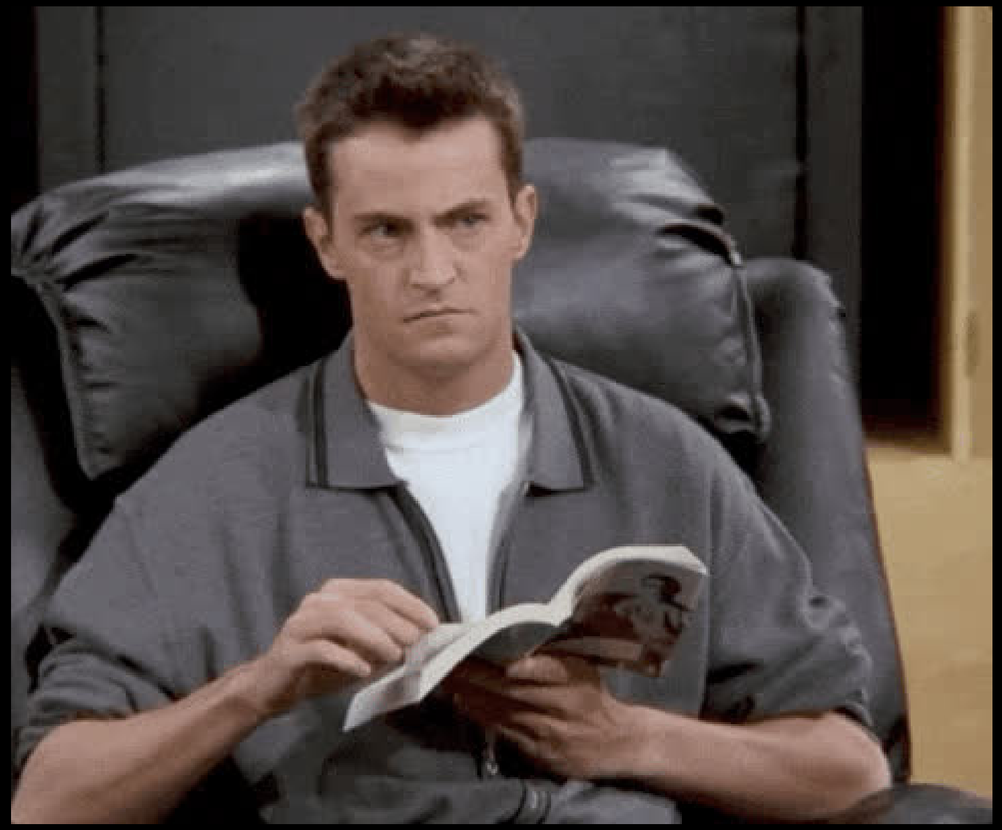
•



console.info('Alexia Kattah', me)

- Engenheira de Computação formada pela Universidade Estadual de Minas Gerais
- Software Engineer Atria Institute 🇺🇸
- AWC Certified
- Há 10 anos de jornada como desenvolvedora
- Stack atual: React.js, node.js, next.js, nest.js, GCP e AWS

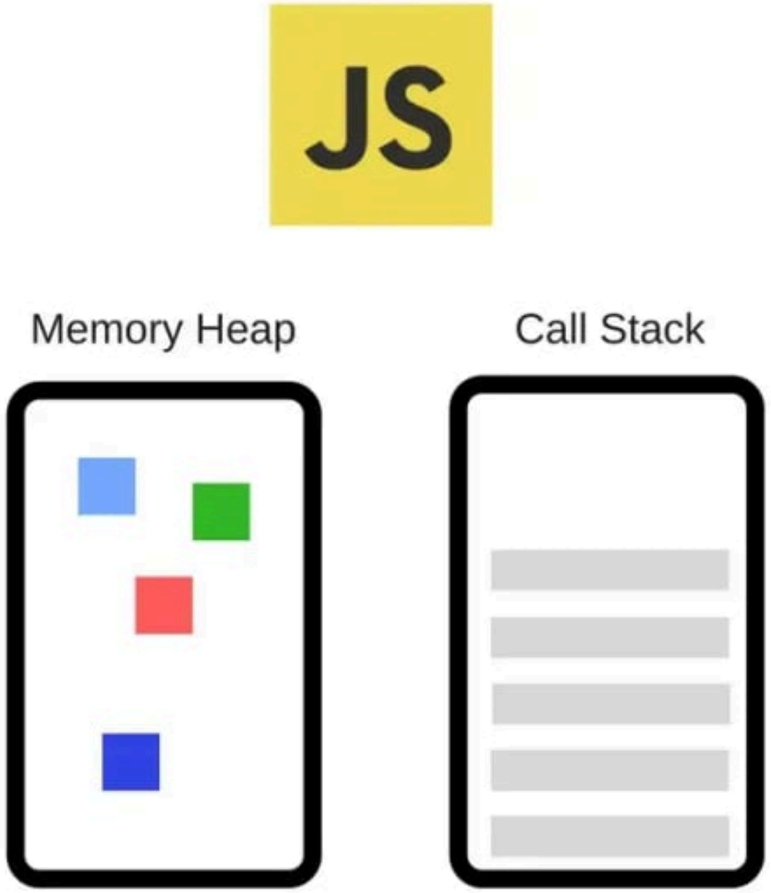
Você sabe como o Node.js funciona?



Let's find out

Vamos entender o que é o V8 e Call Stack

Javascript engine V8

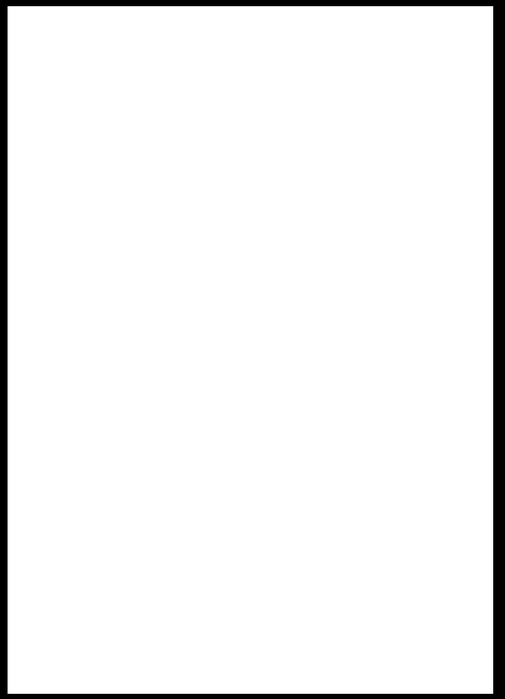


Javascript engine V8

```
1
2  function sum(a, b) {
3    return a + b;
4  }
5
6  function execute(a, b) {
7   const value =  sum(a, b);
8    console.log(value);
9  }
10
11 execute(1, 2);
```

JS

CallStack



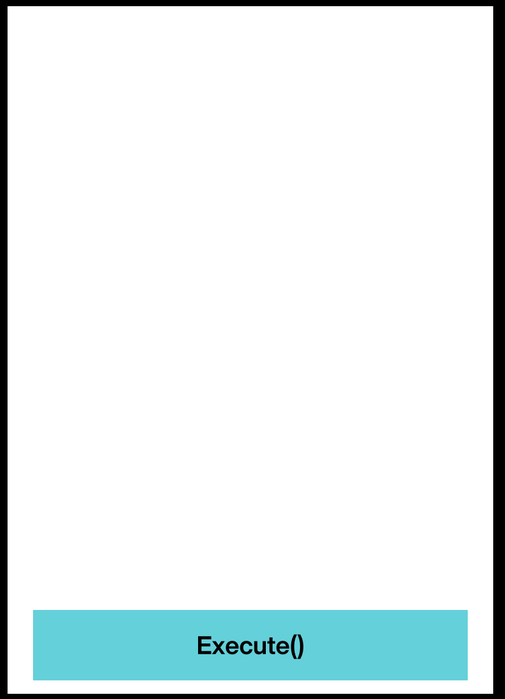
console

Javascript engine V8

```
1
2  function sum(a, b) {
3    return a + b;
4  }
5
6  function execute(a, b) {
7   const value =  sum(a, b);
8    console.log(value);
9  }
10
11 execute(1, 2);
```

JS

CallStack



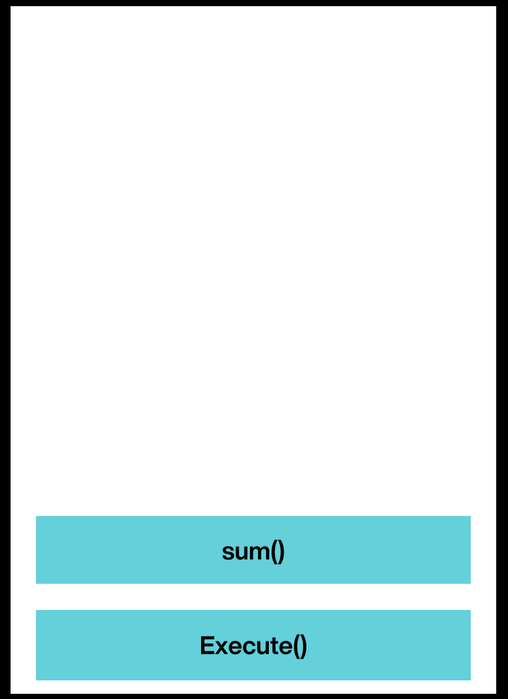
console

Javascript engine V8

```
1
2  function sum(a, b) {
3    return a + b;
4  }
5
6  function execute(a, b) {
7   const value =  sum(a, b);
8    console.log(value);
9  }
10
11 execute(1, 2);
```

JS

CallStack



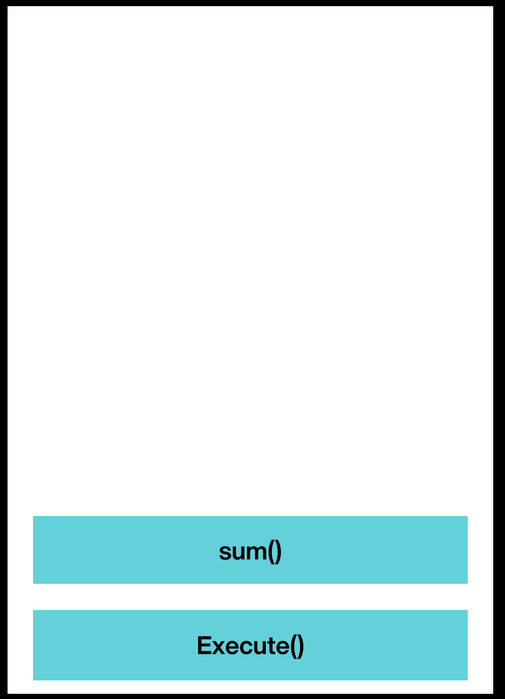
console

Javascript engine V8

```
1
2  function sum(a, b) {
3    return a + b;
4  }
5
6  function execute(a, b) {
7   const value =  sum(a, b);
8    console.log(value);
9  }
10
11 execute(1, 2);
```

JS

CallStack



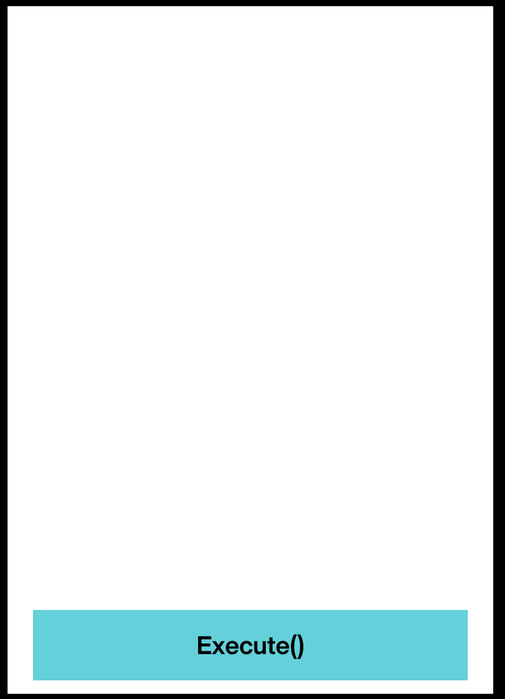
console

Javascript engine V8

```
1
2  function sum(a, b) {
3    return a + b;
4  }
5
6  function execute(a, b) {
7   const value =  sum(a, b);
8    console.log(value);
9  }
10
11 execute(1, 2);
```

JS

CallStack



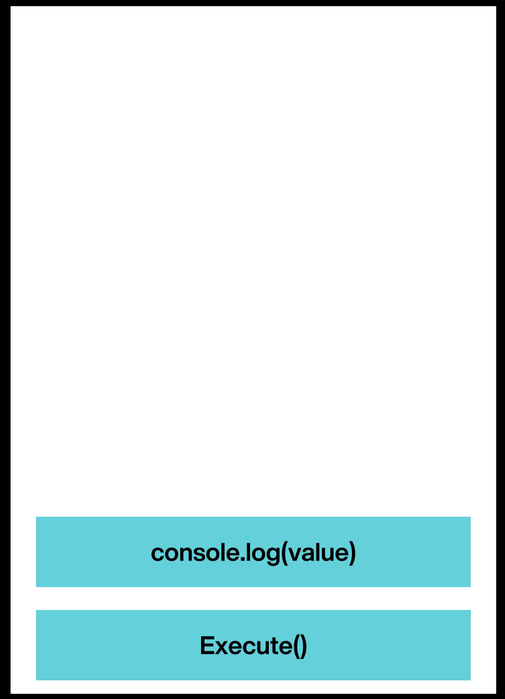
console

Javascript engine V8

```
1
2  function sum(a, b) {
3    return a + b;
4  }
5
6  function execute(a, b) {
7   const value =  sum(a, b);
8    console.log(value);
9  }
10
11 execute(1, 2);
```

JS

CallStack



console

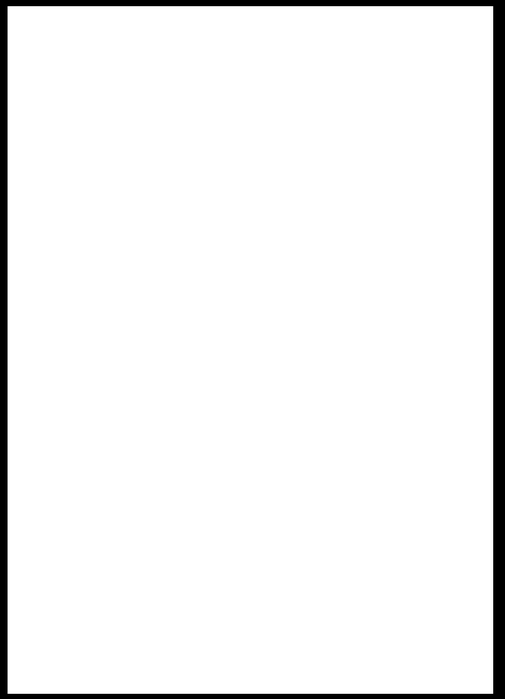
3

Javascript engine V8

```
1
2  function sum(a, b) {
3    return a + b;
4  }
5
6  function execute(a, b) {
7   const value =  sum(a, b);
8    console.log(value);
9  }
10
11 execute(1, 2);
```

JS

CallStack

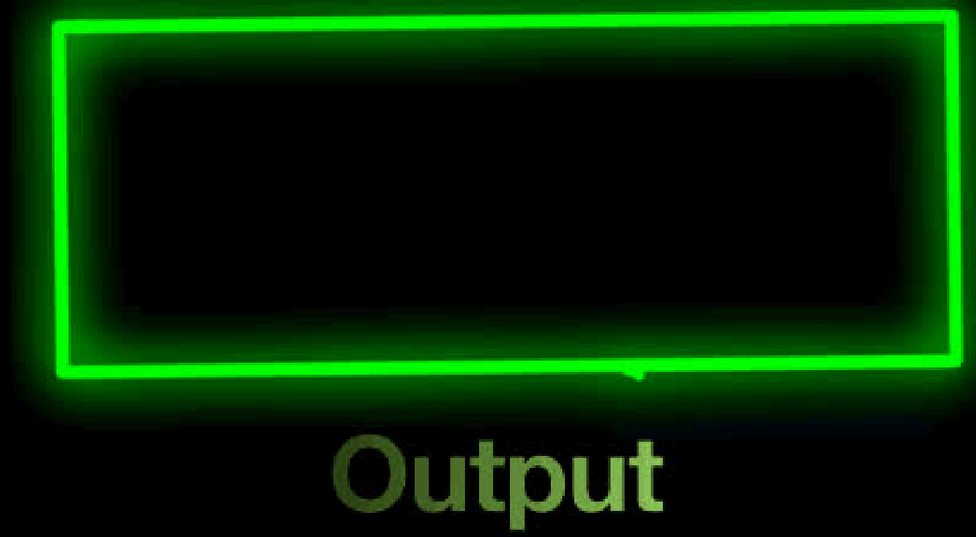


console

```
3
```

Agora vamos entender o event loop


```
function a() {
    return console.log(b())
}
function b() {
    console.log("Called b()")
    return "you can call function a()"
}
a()
```



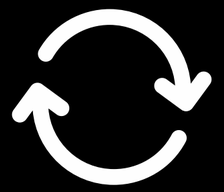
E quando temos tarefas pesadas de CPU, como compressão de arquivos, hash, ou leitura de grandes volumes de dados?

Se a operação for sincrona e muito pesada, ela bloqueia o Event Loop.
Para evitar isso, podemos aproveitar o Thread Pool do libuv para algumas APIs nativas

CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) => {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```



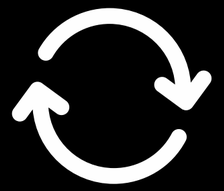
Task Queue

CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) ⇒ {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```

console.log()



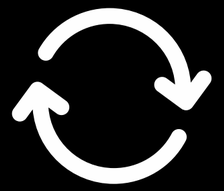
Task Queue

CallStack

readFile()

```
function blockMainThread(ms) {
  const start = Date.now();
  while (Date.now() - start < ms) {
      // Block the main thread
  }
  console.log(`Main thread blocked for ${ms}ms`);
}

console.log("Start program");

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log("File read complete");
  console.log(data);
});

console.log("Started reading the file");

blockMainThread(2000);

console.log("End program");
```

Backgroud Threads (libuv)

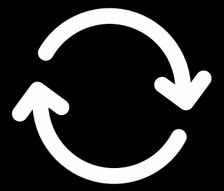


Task Queue

CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) => {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```



Task Queue

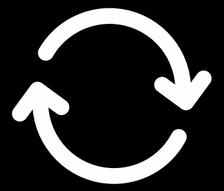
CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) => {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```

readFile()

console.log()



Task Queue

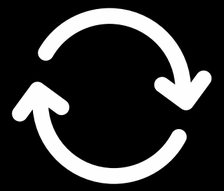
CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) ⇒ {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```

readFile()

blockMainThread(2000)



Task Queue

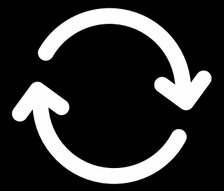
CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) ⇒ {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```

readFile()

blockMainThread(2000)



readFileCallback()

Task Queue

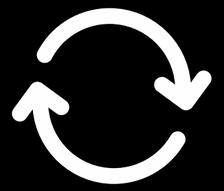
CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) ⇒ {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```

readFile()

blockMainThread(2000)



readFileCallback()

readFileCallback()

Task Queue

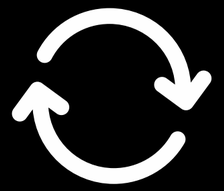
CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) ⇒ {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```

readFile()

blockMainThread(2000)



readFileCallback()

readFileCallback()

readFileCallback()

Task Queue

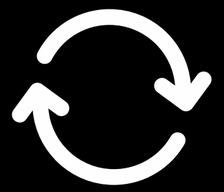
CallStack

Backgroud Threads (libuv)

```
1  function blockMainThread(ms) {
2    const start = Date.now();
3    while (Date.now() - start < ms) {
4        // Block the main thread
5    }
6    console.log(`Main thread blocked for ${ms}ms`);
7  }
8
9  console.log("Start program");
10
11 fs.readFile('example.txt', 'utf8', (err, data) ⇒ {
12   if (err) throw err;
13   console.log("File read complete");
14   console.log(data);
15 });
16
17 console.log("Started reading the file");
18
19 blockMainThread(2000);
20
21 console.log("End program");
```

blockMainThread(2000)

readFile()



readFileCallback()

readFileCallback()

readFileCallback()

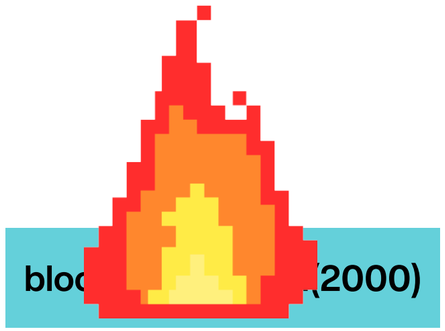
readFileCallback()

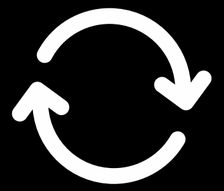
Task Queue

CallStack

Backgroud Threads (libuv)

```
function blockMainThread(ms) {
  const start = Date.now();
  while (Date.now() - start < ms) {
      // Block the main thread
  }
  console.log(`Main thread blocked for ${ms}ms`);
}

console.log("Start program");

fs.readFile('example.txt', 'utf8', (err, data) => {
  if (err) throw err;
  console.log("File read complete");
  console.log(data);
});

console.log("Started reading the file");

blockMainThread(2000);

console.log("End progr  ");
```





Task Queue

Nunca bloqueie o event loop

Além do Event Loop, o Node tem um Thread Pool do libuv. É aqui que caem operações nativas como fs e crypto. Por isso readFile não bloqueia.

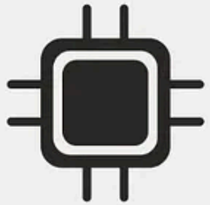
Se eu preciso rodar CPU pesado que não é nativo, entro com Worker Threads! E para escala, uso pool de workers

Worker Threads

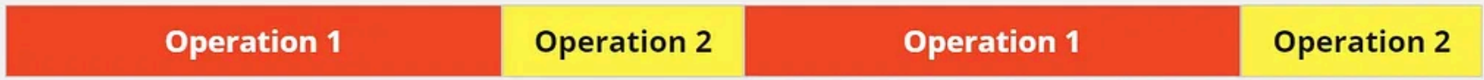
Quando usar:

- Processamento CPU-bound que bloqueia o Event Loop (ex.: criptografia, compressão, cálculos complexos).
- Quando é necessário paralelismo real dentro de um único processo Node.js.

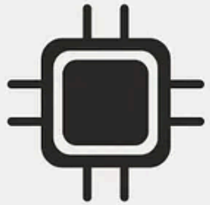
```
const {
  Worker, isMainThread, parentPort, workerData,
} = require('node:worker_threads');

if (isMainThread) {
  module.exports = function parseJSAsync(script) {
    return new Promise((resolve, reject) => {
      const worker = new Worker(__filename, {
        workerData: script,
      });
      worker.on('message', resolve);
      worker.on('error', reject);
      worker.on('exit', (code) => {
        if (code !== 0)
          reject(new Error(`Worker stopped with exit code ${code}`));
      });
    });
  };
} else {
  const { parse } = require('some-js-parsing-library');
  const script = workerData;
  parentPort.postMessage(parse(script));
}
```



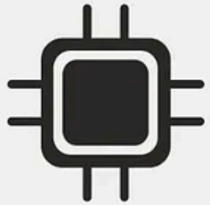
Main thread



Concurrency



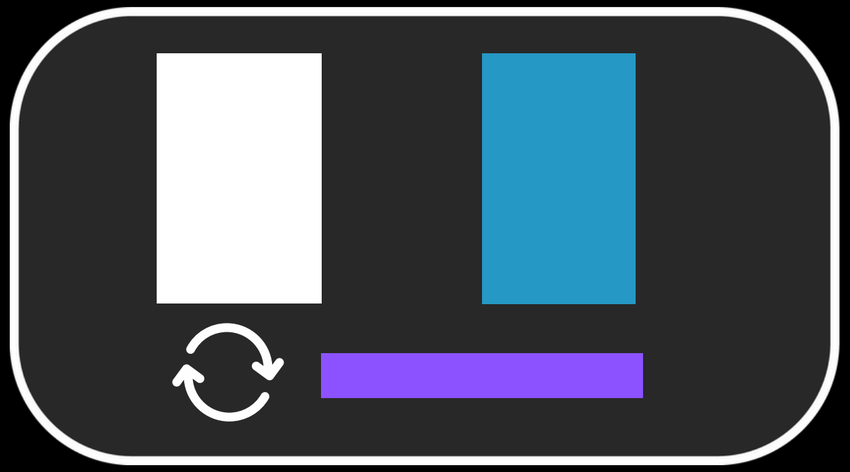
Main thread

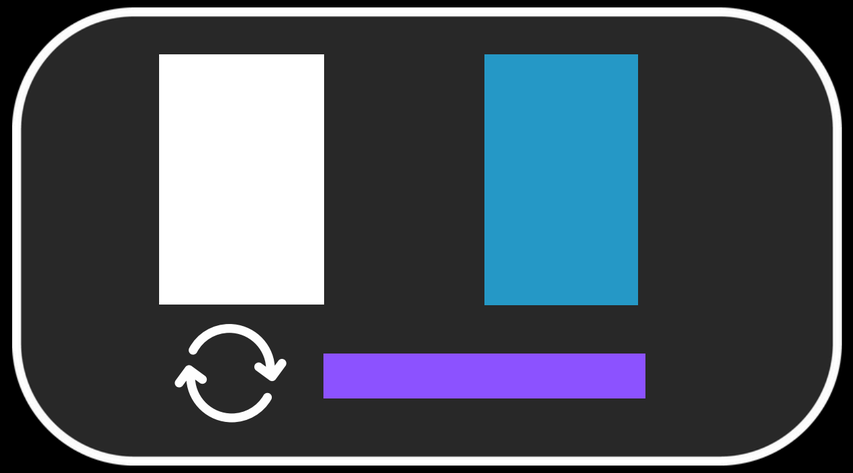


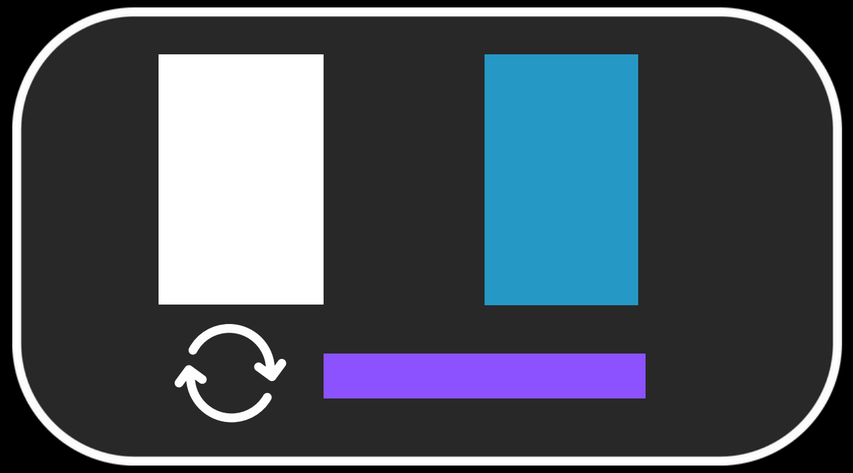
Worker Thread

Parallelism

Cluster







Pool de workers

Pool de workers

- **Reutilização de Workers:** Um pool permite reutilizar workers existentes em vez de criar e destruir um worker para cada tarefa, o que reduz a sobrecarga de criação e destruição de threads

- **Controle de Recursos:** Com um pool, você pode limitar o número de threads simultâneas, evitando o uso excessivo de recursos do sistema.

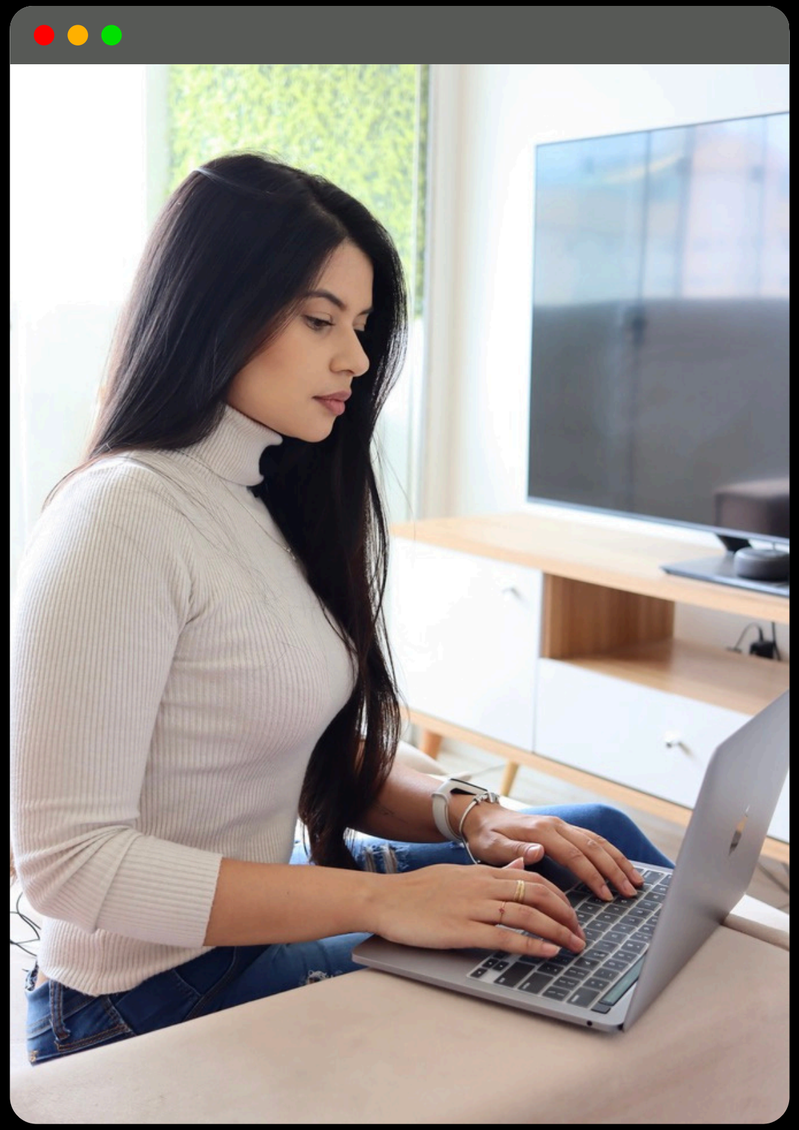
- **Distribuição de Tarefas:** Um pool pode distribuir tarefas de maneira equilibrada entre os workers disponíveis, garantindo que a carga de trabalho seja dividida de forma equitativa.

- **Evitar Sobrecarga:** Um pool ajuda a evitar a criação de um número excessivo de workers que pode sobrecarregar o sistema, proporcionando um controle mais granular sobre o número de threads em execução.

Hora da demo!

No Node.js, concorrência não é só sobre requests assíncronas. É sobre usar o hardware de forma inteligente.

- Worker Threads — docs Node.js (paralelismo CPU-bound, memória compartilhada). Node.js
- Cluster — docs Node.js (multi-processo, balanceamento, IPC). Node.js
- Child Process — docs Node.js (spawn/exec/fork, integração com binários). Node.js
- Event Loop / timers / nextTick — guia oficial (ordem de fases). Node.js+1
- Não bloqueie o Event Loop (armadilhas comuns, REDOS). Node.js
- Thread Pool do libuv (padrão 4; configurar UV_THREADPOOL_SIZE). libuv Documentation
- pool de Worker Threads — GitHub
- autocannon (benchmark HTTP) — GitHub. GitHub
- Crypto/PBKDF2 usa o thread pool — docs oficiais de crypto. Node.js
- Demo GDG - BH – (github.com/alexiakattah/gdg-bh) Github



Obrigada!

/alexiakattah

/alexiakattah

/alexiakattah

Vamos nos conectar!