**Node.js**

No ano de 2009, Ryan Dahl estava trabalhando em um projeto quando se deparou com um grande problema, descobrir quanto tempo faltava para uma ação de upload ser concluída.

Até então, um método comum utilizado consistia em enviar requisições AJAX a cada determinado intervalo de tempo, perguntando ao servidor quanto do arquivo já tinha sido enviado. Porém, esse método era custoso ao servidor e ao cliente, pois eram feitas muitas requisições até o término do upload. Uma alternativa era o *Long Polling*, que consistia em fazer uma pergunta ao servidor, porém o servidor segurava a resposta até algum evento acontecer. Esse processo é mostrado na figura abaixo.

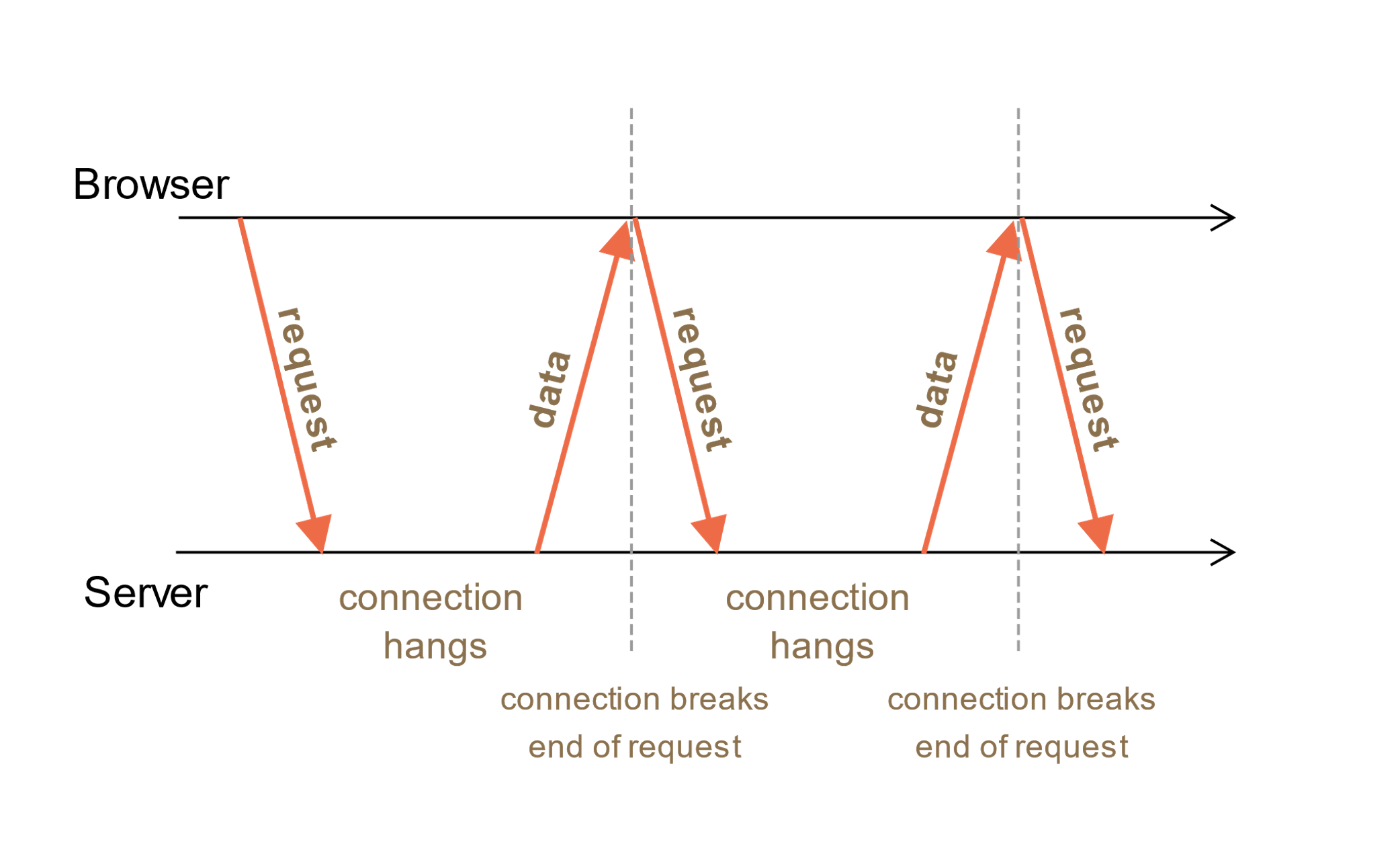


Imagem x - Descrição da Imagem

Dahl queria implementar um servidor que englobasse esse conceito. Um ano antes, em 2008, a empresa Google havia lançado o navegador Google Chrome, que contava com o V8, um poderoso interpretador JavaScript desenvolvido pela mesma.

Aproveitando as novas ferramentas disponíveis, o grande crescimento no mercado, e por não necessitar de bibliotecas ou meios para se implementar assincronismo, escolheu JavaScript como linguagem. O que também evitou a necessidade de criação e gerenciamento de *threads*, perfeito para manter a simplicidade que ele desejava.

JavaScript era simples e leve, permitia ao servidor liberar o uso da CPU pois tinha suporte a operações não-bloqueantes, executava em uma única *thread*, diminuía o uso da CPU e memória, e era orientada a eventos, o que era essencial para implementar o *Long Polling*. Com isso Dahl começou a implementar seu projeto, que foi crescendo, ganhando padrões e hoje é o servidor Web completo que todos conhecem, o Node.js.

**Assincronicidade**

O maior benefício do Node.js, é a facilidade de se implementar assincronismo.

Graças a Libuv, uma biblioteca open source criada originalmente para o Node.js que implementa um Event Loop com todos os recursos necessários, o assincronismo não precisa ser gerenciado pelo desenvolvedor, a própria biblioteca gerencia as threads e os processos para que tudo corra perfeitamente.

O Node.js oferece 3 maneiras de se implementar assincronismo: Callbacks, Promises e utilizando Async/Await (à partir da ES2017).

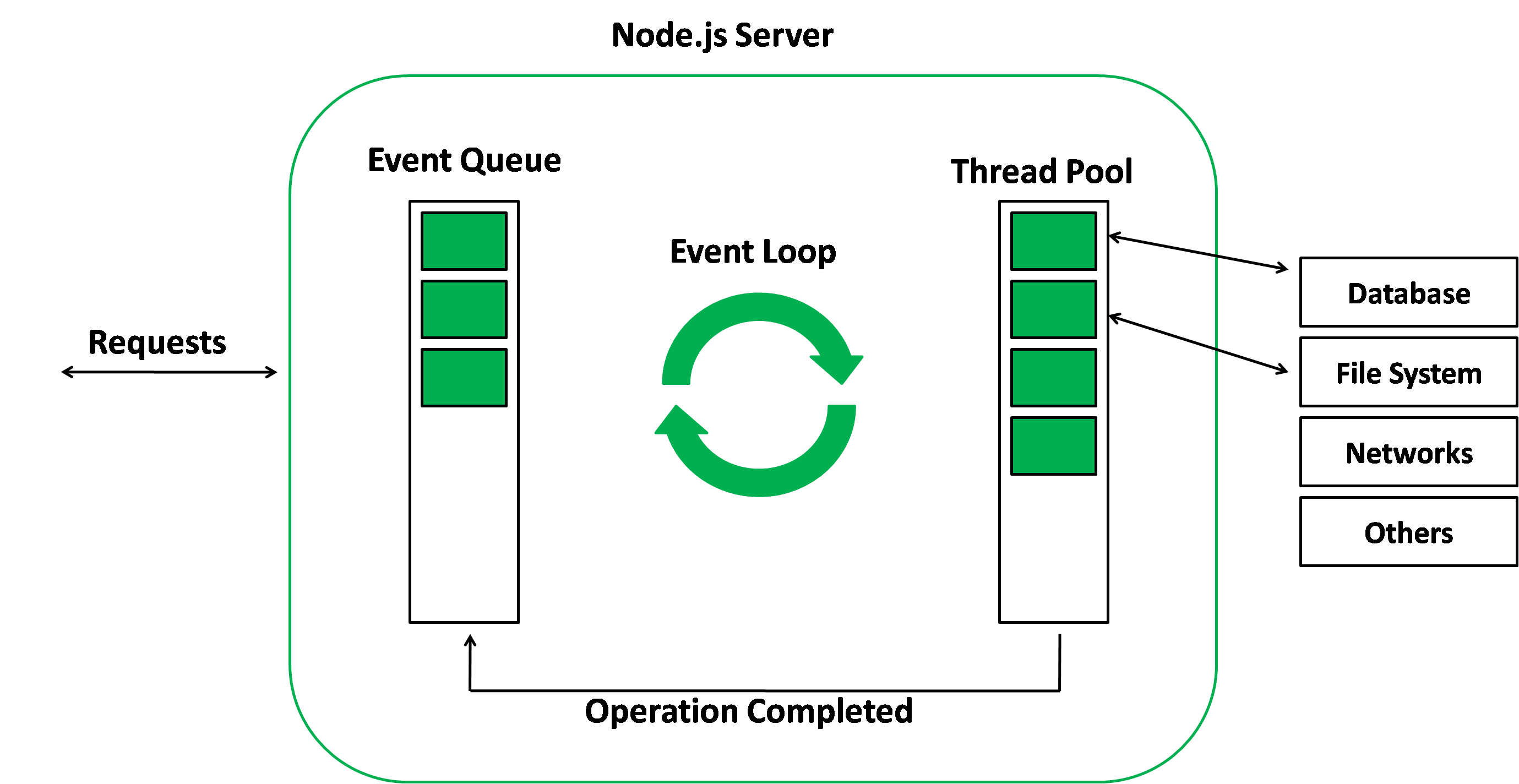


Imagem x – Título da Imagem

**Callbacks**

Callbacks são funções passadas como parâmetro para outra função. A função que o recebe pode executar a callback quando ela achar necessário, normalmente depois de um intervalo de tempo.

Algumas funções internas utilizam callback, como setInterval e setTimeout.

A função setTimeout é o melhor exemplo possível para se entender esse conceito. Caso essa função fosse síncrona, a execução do restante do código ficaria parada por um grande tempo, dependendo do tempo limite que for definido. Mas como ela é assíncrona, recebe uma função callback que será executada após o tempo limite terminar, fazendo assim que o código continue sendo executado sem nenhum bloqueio.

Outro exemplo também é o método .addEventListener(), onde é passado como primeiro parâmetro o tipo de evento que será ouvido, como o “click” por exemplo, e o segundo parâmetro é um callback que será executada somente quando o evento for acionado.

No entanto, utilizar muitos call-backs aninhados pode deixar tudo muito complicado para o desenvolvedor e acabar resultando em um “callback hell”. Como uma opção de substituto aos callbacks, foi introduzido o conceito de promise,

**Promises**

Promise nada mais é do que uma sintaxe mais agradável de callback, foi desenvolvida para ajudar o desenvolvedor a evitar erros como o “callback hell”.

Uma promise pode ter 3 estados:

* Pendente: é o primeiro estado, imediatamente quando a promise é criada, e permanece nesse estado até avançar para resolvida ou rejeitada.
* Resolvida: é quando a operação é concluída com sucesso. Normalmente uma promise resolvida retorna algum tipo de dado com ela.
* Rejeitada: é quando a operação falha. Normalmente uma promise rejeitada retorna algum tipo de erro com ela.

As promises vem integradas com um método .then(), que recebe uma função de retorno como parâmetro. Essa função recebe todos os dados e só será executado quando a promise estiver resolvida.

No caso da promise ser rejeitada, há o método .catch(), que será executado sempre que houver rejeição.

Os métodos .then() e .catch() dão à promise a aparência de um bloco de código try/catch.

**Async/Await**

Diferente das funções normais, funções declaradas com a palavra-chave async permitem se utilizar outra palavra-chave dentro do escopo da função: await.

Quando essa função é chamada, ela retorna uma promise pendente. Se a função retornar um valor, ela resolve a promise com esse valor, se gerar um erro, ela rejeita a promise com o erro gerado.

A palavra-chave await é utilizada para tornar síncronas as funções que normalmente são assíncronas em síncronas. Normalmente é utilizada quando há uma função que retorna uma promise e é necessário pausar a execução da função assíncrona até que a chamada seja concluída.

Essas palavras-chaves podem ajudar muito o desenvolvedor dependendo de suas necessidades.

**Gerenciador de Pacotes (NPM)**

Node Package Manager (NPM) é o gerenciador de pacotes nativo do Node.js. Consiste em três componentes distintos: o site, a interface de linha de comando e o registro.

O site é usado para descobrir pacotes, configurar perfis e gerenciar a experiência com o NPM. A interface de linha de comando é executada no terminal do sistema e é usada para o desenvolvedor interagir com o NPM. O registro é um banco de dados de software JavaScript e as meta-informações que o cercam.

**Servidor Integrado**

Diferente de outras linguagens de programação para Web onde temos que configurar um servidor externo, o Node.js já traz um embutido, que consegue dar suporte até aplicações de médio porte sem grandes dificuldades.

Com poucas linhas de código conseguimos iniciar um servidor na porta que desejamos.

Tela preta com letras brancas em fundo preto

Descrição gerada automaticamente