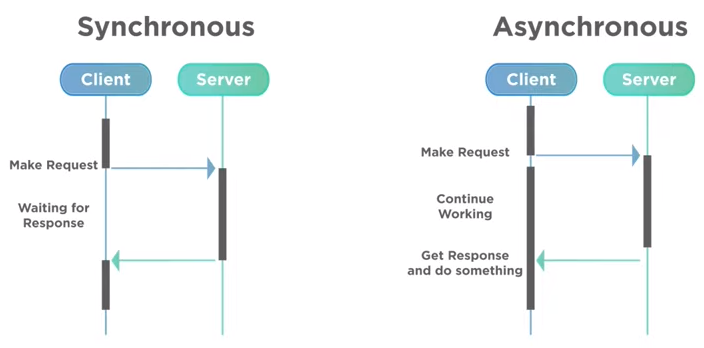
**Assincronicidade**

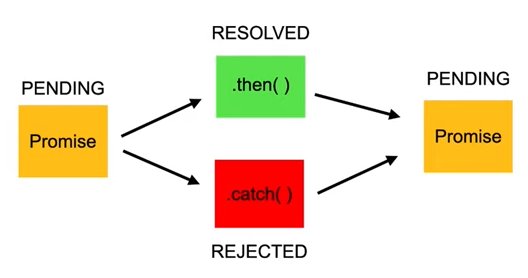
**Assíncrono:**

* que não ocorre ou não se efetiva ao mesmo tempo
* contrário de síncrono, que é algo que acontece simultaneamente
* JavaScript, por padrão, roda de maneira **síncrona**



**Promises**

* objeto de processamento **assíncrono**
* **inicialmente** seu **valor** é **desconhecido**. Ela pode, então, ser **resolvida** ou **rejeitada**



**Promises possui 3 estados:**

1. **Pending** – quando recebe o objeto promise, fica como pendente
2. **Fulfilled** - completada
3. **Rejected** – quando deu errado

**Manipulação:**

const myPromise = **new** *Promise* ((*resolve*, *reject*) => {

    window.setTimeout(() => {

        resolve (console.log ('Resolvida!'));

    }, 2000);

});

await myPromise

    .then ((*result*) => *result* + ' passando pelo then')

    .then ((*result*) => *result* + ' e agora acabou!')

    .catch ((*err*) => console.log(*err*.message));

// após 2 segundos, retornará o valor

// "resolvida passando pelo then e agora acabou!"

**Async/await**

* Funções assíncronas precisam dessas duas palavras chave
* Await é um operador específico para lidar com Promises

async function resolvePromise () {

    const myPromise = **new** *Promise* ((*resolve*, *reject*) => {

        window.setTimeout(() => {

            resolve (console.log ('Resolvida!'));

        }, 3000);

    });

    const resolved = await myPromise   //a palavra "await" é criada especificamente para lidar com Promises

        .then ((*result*) => *result* + ' passando pelo then')

        .then ((*result*) => *result* + ' e agora acabou!')

        .catch ((*err*) => console.log(*err*.message));

    return resolved;

}

* funções assíncronas também retornam Promises!
* também é possível utilizar o try...catch