Trabalhar o presente.
Construir o futuro.

#JavaScript-02

SENAI

Introdução JavaScript

SENAI

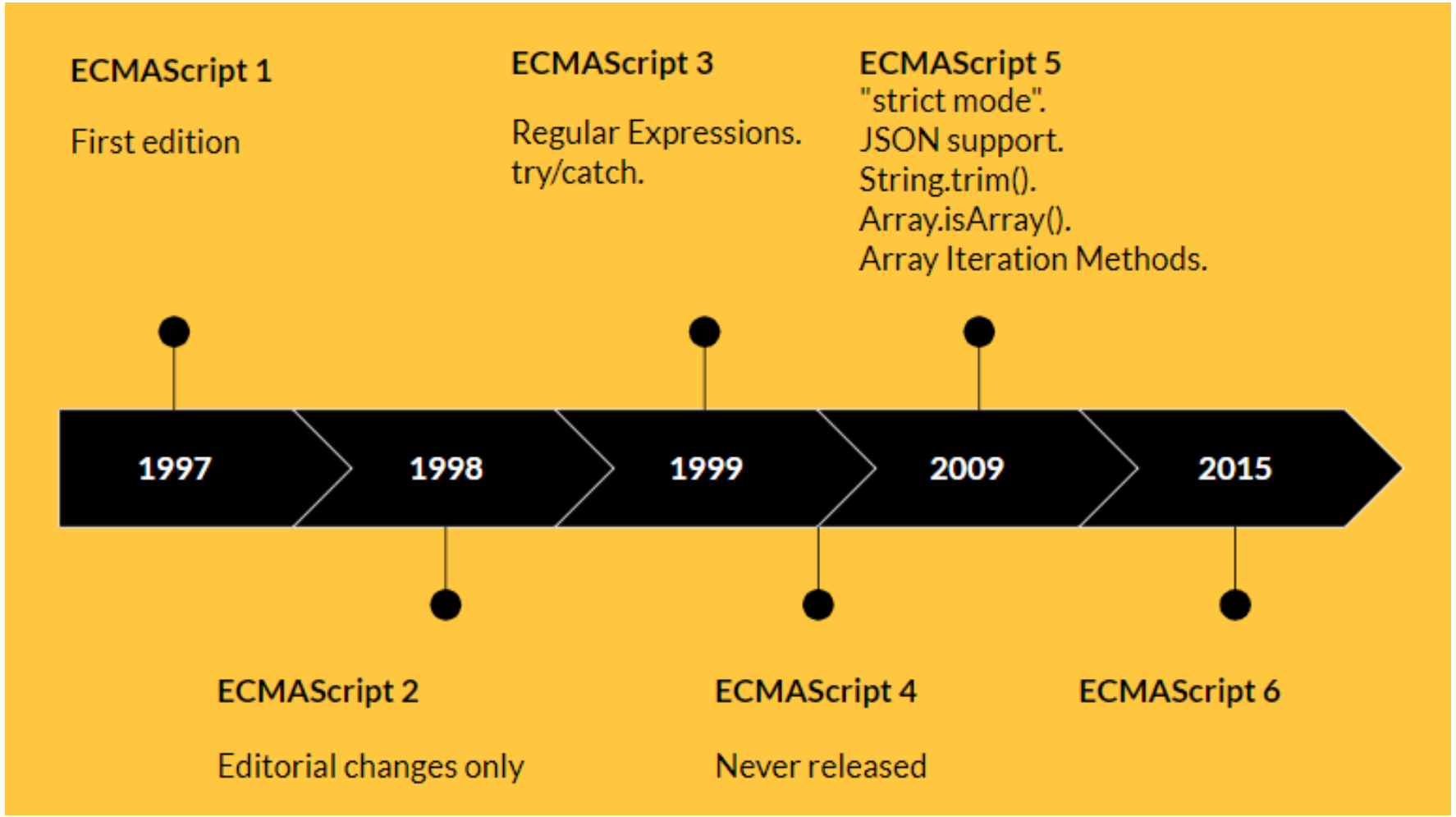
Objetivo Geral

Etapa 1	História
Etapa 2	Evolução JavaScript
Etapa 3	Aplicações
Etapa 4	Document Object Model
Etapa 5	Seleção de Elementos no DOM

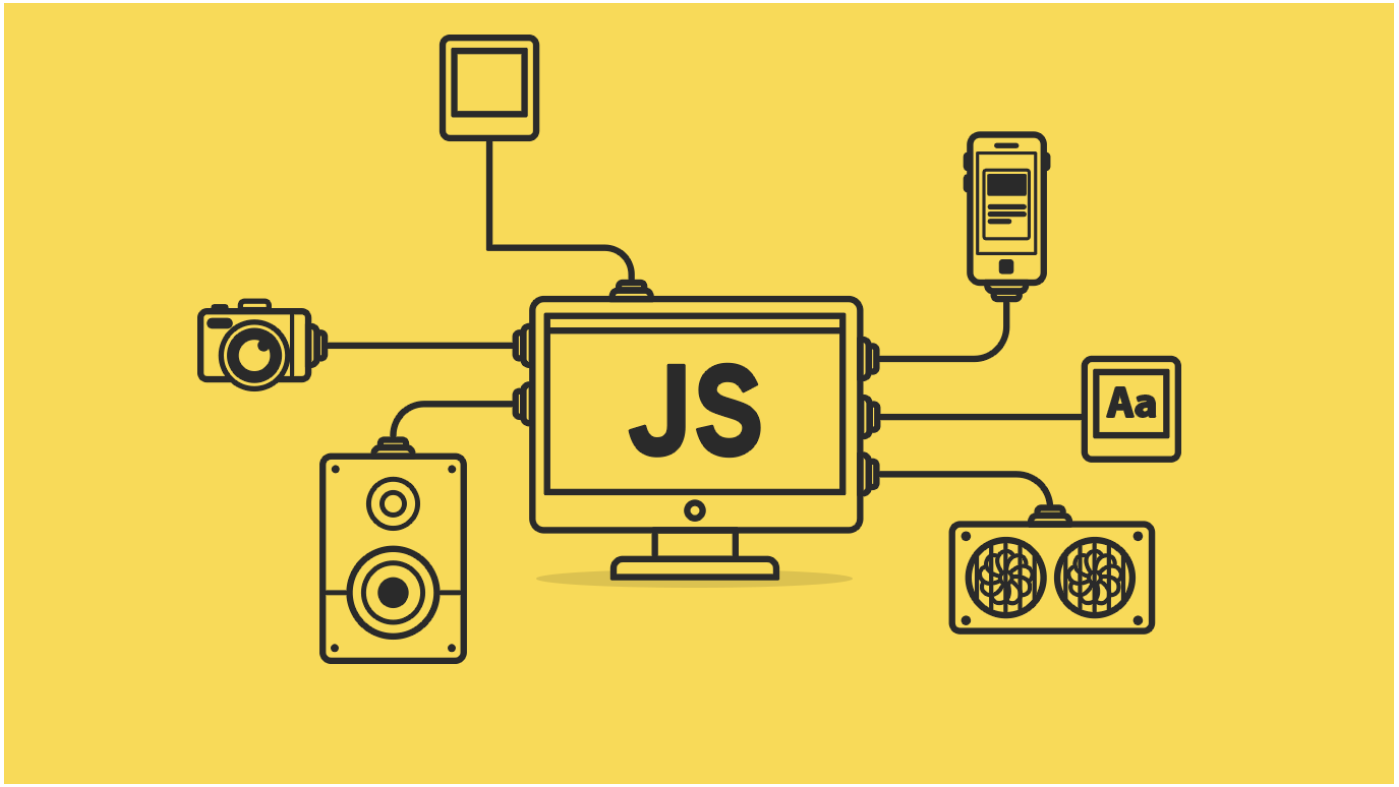
História

- Leve
- Interpretada
- Baseada em Protótipos
- Multiparadigma
- Comumente utilizada em aplicações web client-side
- Segue o padrão ECMAScript

Evolução JavaScript



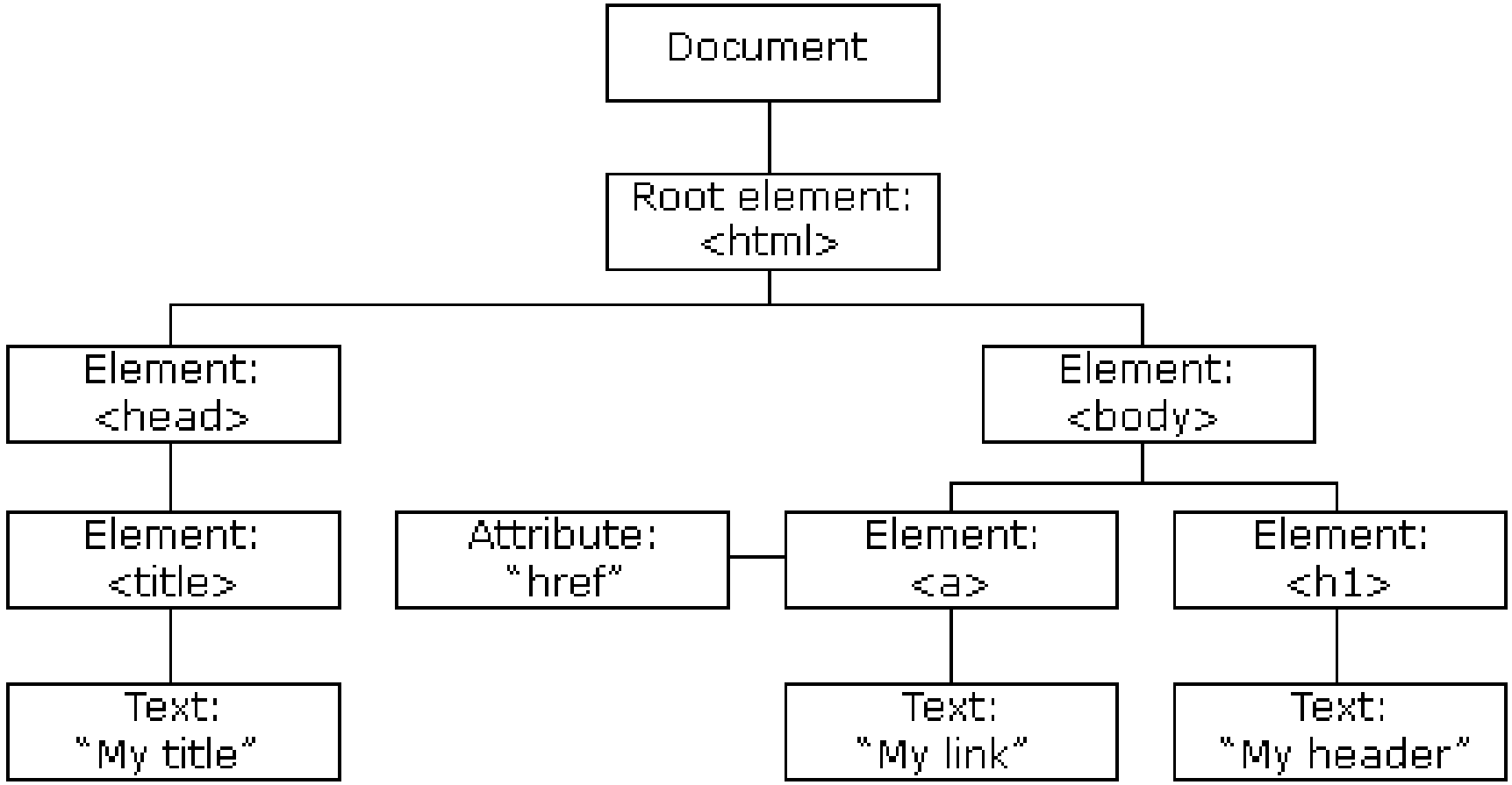
Aplicações



- Web
- Mobile
- Smartwatches
- Games
- Internet of Things
- APIs

DOM

Document Object Model



Seleção de Elementos no DOM

Seleção de Métodos

```html
<html>
  <head>
    <title>Minha página</title>
  </head>

  <body>

    <h1 id="titulo">Minha página</h1>

    <section class="textos">
      <h2>Sobre mim</h2>
      <p>Texto sobre a pessoa aqui.</p>
    </section>

    <section class="textos">
      <h2>Meus projetos</h2>
      <ul>
        <li>Projeto 1</li>
        <li>Projeto 2</li>
        <li>Projeto 3</li>
      </ul>
    </section>

  </body>
</html>
```

```js
document.getElementById('titulo');
// <h1 id="titulo">Minha página</h1>
```

```js
document.getElementsByTagName('li');
/*
[
    <li>Projeto 1</li>,
    <li>Projeto 2</li>,
    <li>Projeto 3</li>
]
*/
```

```js
document.getElementsByClassName('textos');

/*
[
    <section class="textos">
      <h2>Sobre mim</h2>
      <p>Texto sobre a pessoa aqui.</p>
    </section>,

    <section class="textos">
      <h2>Meus projetos</h2>
      <ul>
        <li>Projeto 1</li>
        <li>Projeto 2</li>
        <li>Projeto 3</li>
      </ul>
    </section>
]
*/
```

Seleção de Métodos

```html
<html>
  <head>
    <title>Exemplo querySelectorAll</title>
  </head>

  <body>
    <div class="primeira-classe segunda-classe">
      <ul>
        <li class="opcao">opcao 1</li>
        <li class="opcao">opcao 2</li>
        <li class="opcao">opcao 3</li>
    </div>
  </body>
</html>
```

```javascript
document.querySelectorAll('.primeira-classe .segunda-classe');
/*[
    <div class="primeira-classe segunda-classe">
      <ul>
        <li class="opcao">opcao 1</li>
        <li class="opcao">opcao 2</li>
        <li class="opcao">opcao 3</li>
    </div>
]
*/

document.querySelectorAll('li .opcao');
/*
[
    <li class="opcao">opcao 1</li>,
    <li class="opcao">opcao 2</li>,
    <li class="opcao">opcao 3</li>,
]
```

Propriedades e valores

Adicionar e Deletar

Método	Descrição
document.createElement(element)	Cria um novo elemento HTML
document.removeChild(*element*)	Remove um elemento
document.appendChild(*element*)	Adiciona um elemento
document.replaceChild(*new, old*)	Substitui um elemento

Manipulação de Elementos

❑ Element.classList

```html
<div id="meu-elemento" class="classe">
  <!-- resto do código aqui -->
</div>
```

```javascript
const meuElemento = document.getElementById("meu-elemento")

meuElemento.classList.add("novo-estilo");
// Adiciona a classe "meu estilo"

meuElemento.classList.remove("classe")
// Remove a classe "classe"

meuElemento.classList.toggle("dark-mode")
// Adiciona a classe "dark-mode" caso ela não faça parte da
// lista e remove ela caso faça.
```

Manipulação de Elementos

- Acessar diretamente o CSS de um elemento

```
document.getElementsByTagName("p").style.color = "blue";
```

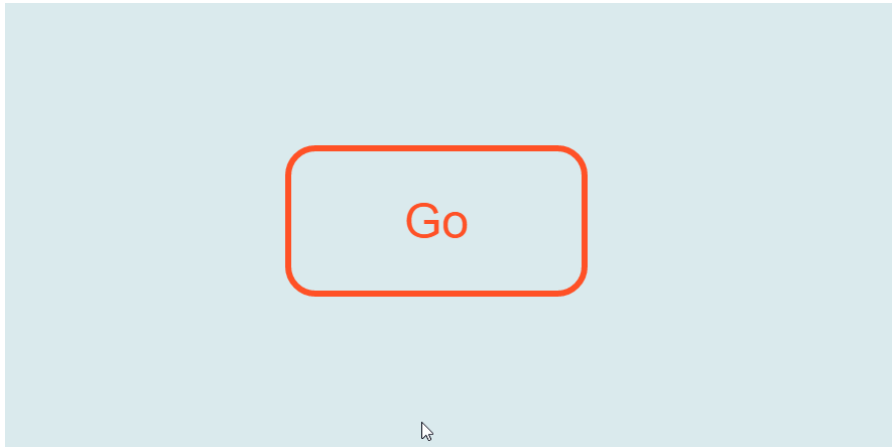
Eventos

Eventos

- Eventos do mouse
- mouseover, mouseout

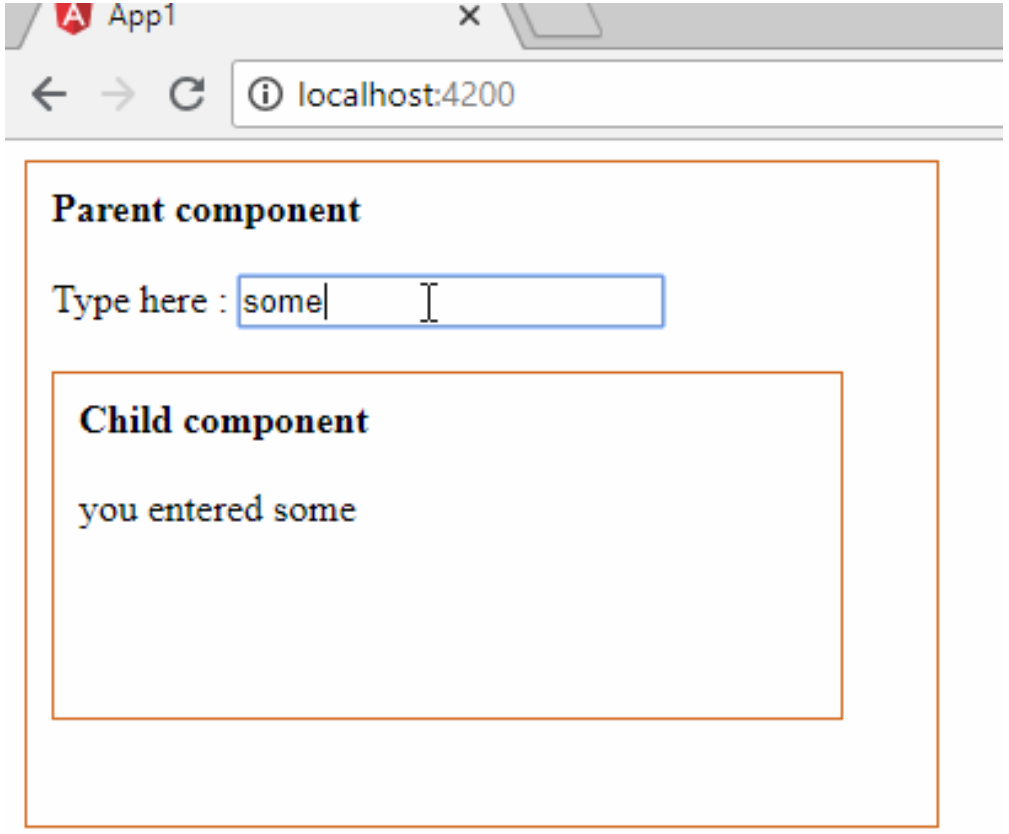
Eventos

- Eventos de clique
- click, dbclick



Eventos

- Eventos de atualização
- Change, load



Adicionar Eventos

- Diretamente no **JavaScript**, cria um evento que vai ser acionado no momento em que o usuário realizar determinada ação.

```
const botao = document.getElementById("meuBotao");

botao.addEventListener("click", outraFuncao);
```

Atributo HTML

Especifica a função a ser chamada diretamente no elemento **HTML**.

```
<html>
<body>

<h1 onclick="mudaTexto(this)"Clique aqui!</h1>

<script>
  function mudaTexto(id) {
    id.innerHTML = "Mudei!";
  }
</script>

</body>
</html>
```

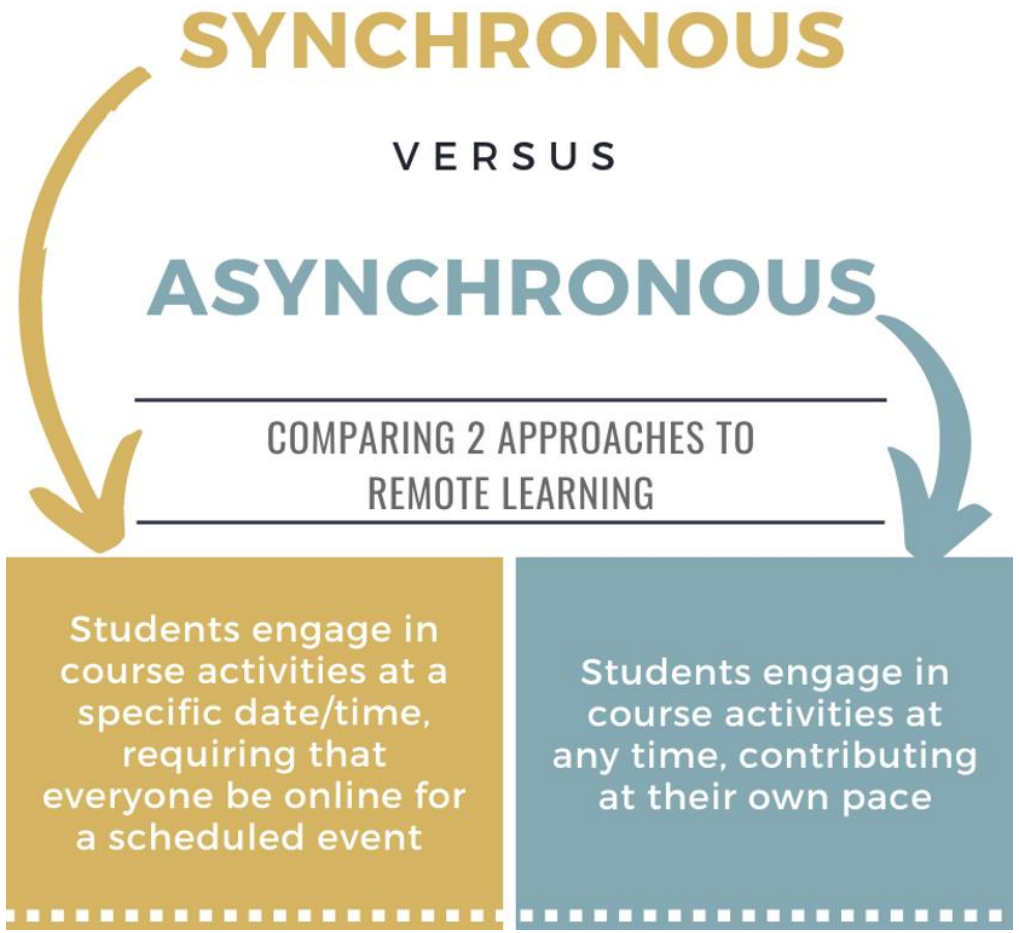
JavaScript Assíncrono

SENAI

- **Assíncrono**

"Que não ocorre ou não se efetiva ao mesmo tempo."

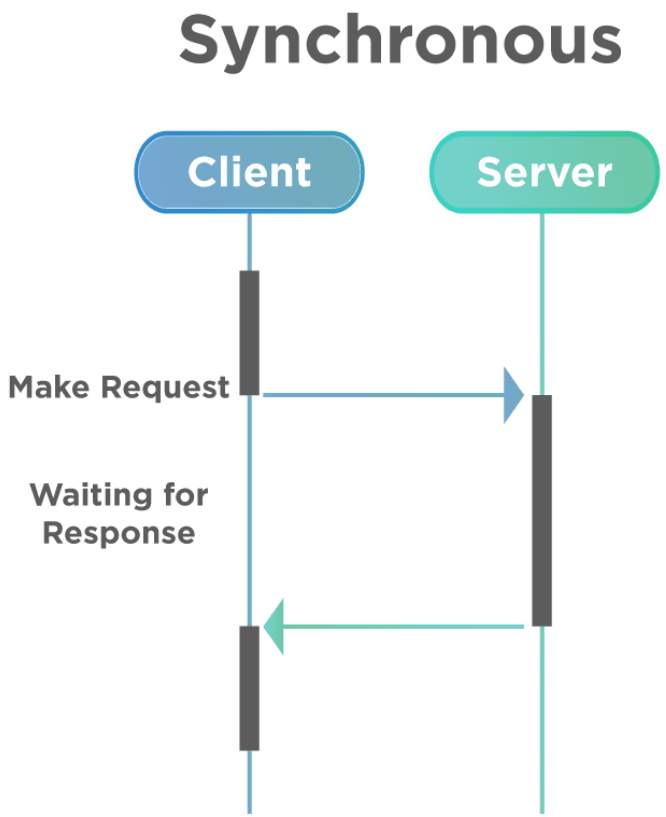
Aplicações

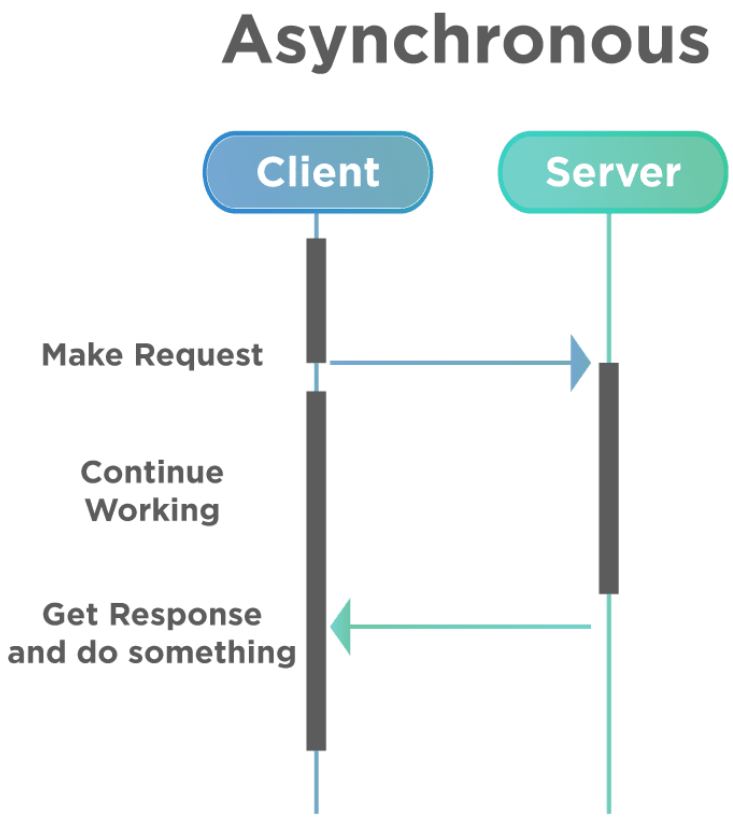


- Assíncrono

"Que não ocorre ou não se efetiva ao mesmo tempo."

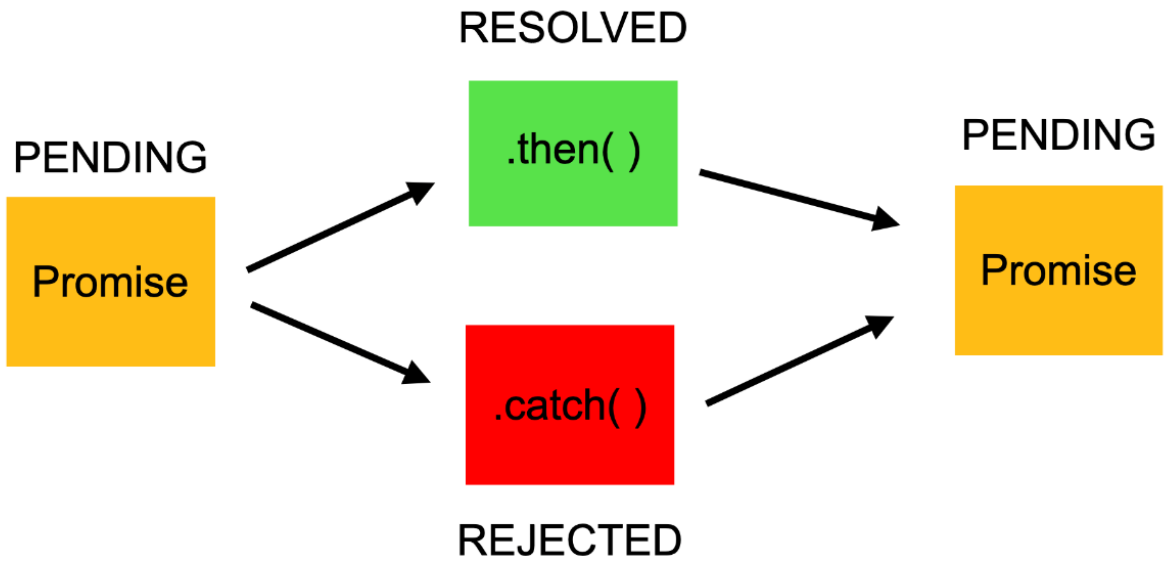
O Javascript roda de maneira síncrona.





Promisses

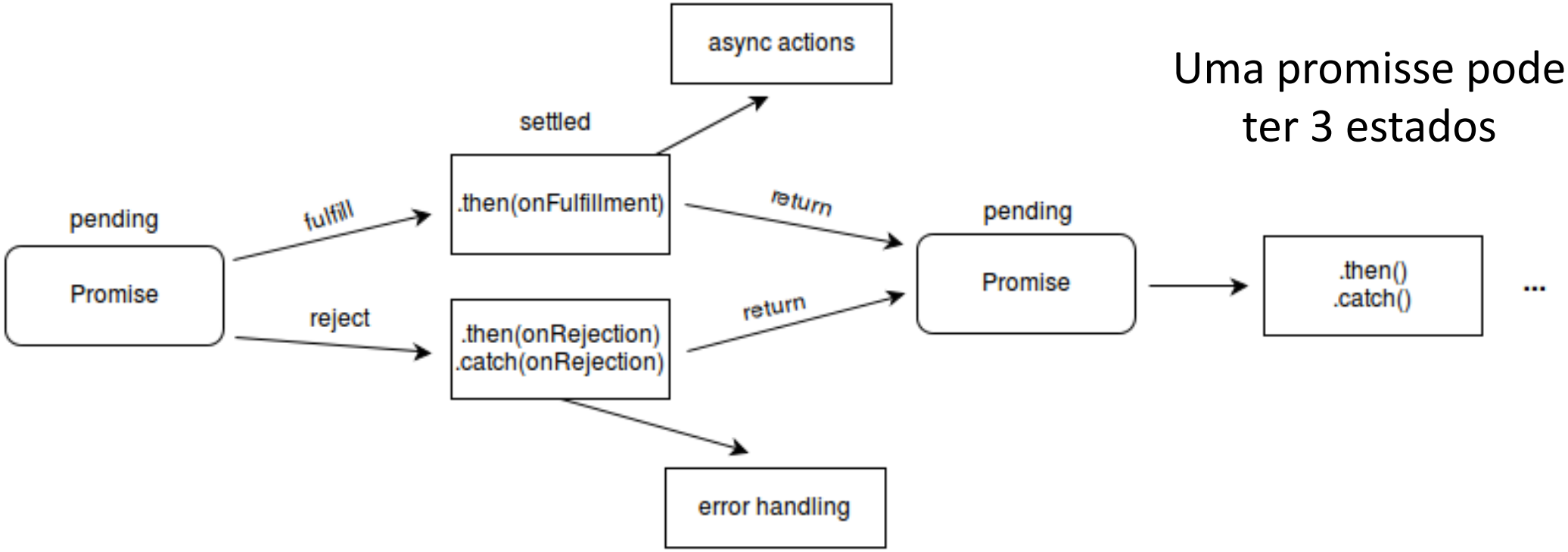
Objeto de processamento assíncrono



- Inicialmente, seu valor é desconhecido.
- Ela pode, então, ser resolvida ou rejeitada.

Promisses - Estados

- **1)** Pending **2)** Fulfilled **3)** Rejected



Uma promisse pode ter 3 estados

Estrutura

```
const myPromise = new Promise((resolve, reject) => {
    window.setTimeout(() => {
        resolve(console.log('Resolvida!'));
    }, 2000);
});
```

Manipulação

```
const myPromise = new Promise((resolve, reject) => {
    window.setTimeout(() => {
        resolve(console.log('Resolvida!'));
    }, 2000);
});
```

Async/Awayt

SENAI

Async/await

- Funções assíncronas precisam dessas duas palavras chaves.

```
async function resolvePromise() {
  const myPromise = new Promise((resolve, reject) => {
    window.setTimeout(() => {
        resolve('Resolvida');
    }, 3000);
});

  const resolved = await myPromise
        .then((result) => result + ' passando pelo then')
        .then((result) => result + ' e agora acabou!')
        .catch((err) => console.log(err.message));

  return resolved;
}
```

Async/await

- Funções assíncronas também retornam Promises!

```
async function resolvePromise() {
  const myPromise = new Promise((resolve, reject) => {
    window.setTimeout(() => {
        resolve('Resolvida');
    }, 3000);
});

  const resolved = await myPromise
        .then((result) => result + ' passando pelo then')
        .then((result) => result + ' e agora acabou!')
        .catch((err) => console.log(err.message));

  return resolved;
}
```

```
> resolvePromise()
<· ▶ Promise {<pending>}
> await resolvePromise()
<· "Resolvida passando pelo then e agora acabou!"
```

Async/await

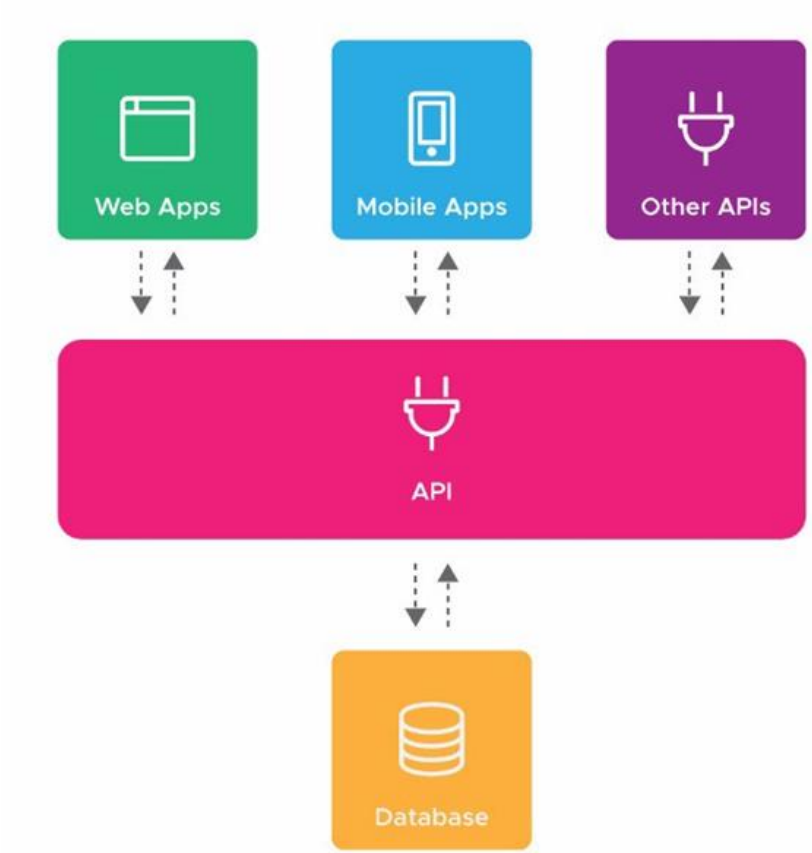
- Utilizando try...catch

```
async function resolvePromise() {
    const myPromise = new Promise((resolve, reject) => {
        window.setTimeout(() => {
            resolve('Resolvida');
        }, 3000);
    });

  let result;

    try {
    result = await myPromise
        .then((result) => result + ' passando pelo then')
        .then((result) => result + ' e agora acabou!')
  } catch(err) {
    result = err.message;
  }

    return result;
}
```

API´s

SENAI

API´s - Application Programming Interface



- Uma API é uma forma de intermediar os resultados do back-end com o que é apresentado no front-end.
- Você consegue acessá-la por meio de URLs.

API´s - JSON: JavaScript Object Notation

bank.json

```
[
  {
    "description": "schema POST bank",
    "type": "object",
    "properties": {
    "id": {
      "type": "number",
      "minimum": 0
    },
    "code": {
      "type": "string"
    },
    "name": {
        "type" : "string"
    }
  }
]
```

- É muito comum que **APIs** retornem seus dados no formato **.json**, portanto precisamos tratar esses dados quando os recebermos.

API´s - Consumindo APIs

```
fetch(url, options)
  .then(response => response.json())
  .then(json => console.log(json))

// retorna uma Promise
```

API´s - Operações no banco (POST, GET, PUT, DELETE, etc)

```
fetch('https://endereco-api.com/', {
  method: 'GET',
  cache: 'no-cache',

})
  .then(response => response.json())
  .then(json => console.log(json))

// retorna uma Promise
```

```
fetch('https://endereco-api.com/', {
  method: 'POST',
  cache: 'no-cache',
  body: JSON.stringify(data)

})
  .then(response => response.json())
  .then(json => console.log(json))

// retorna uma Promise
```

SENAI

COM O **SENAI**,
VOCÊ FAZ O FUTURO.