

# Pilhas e Filas

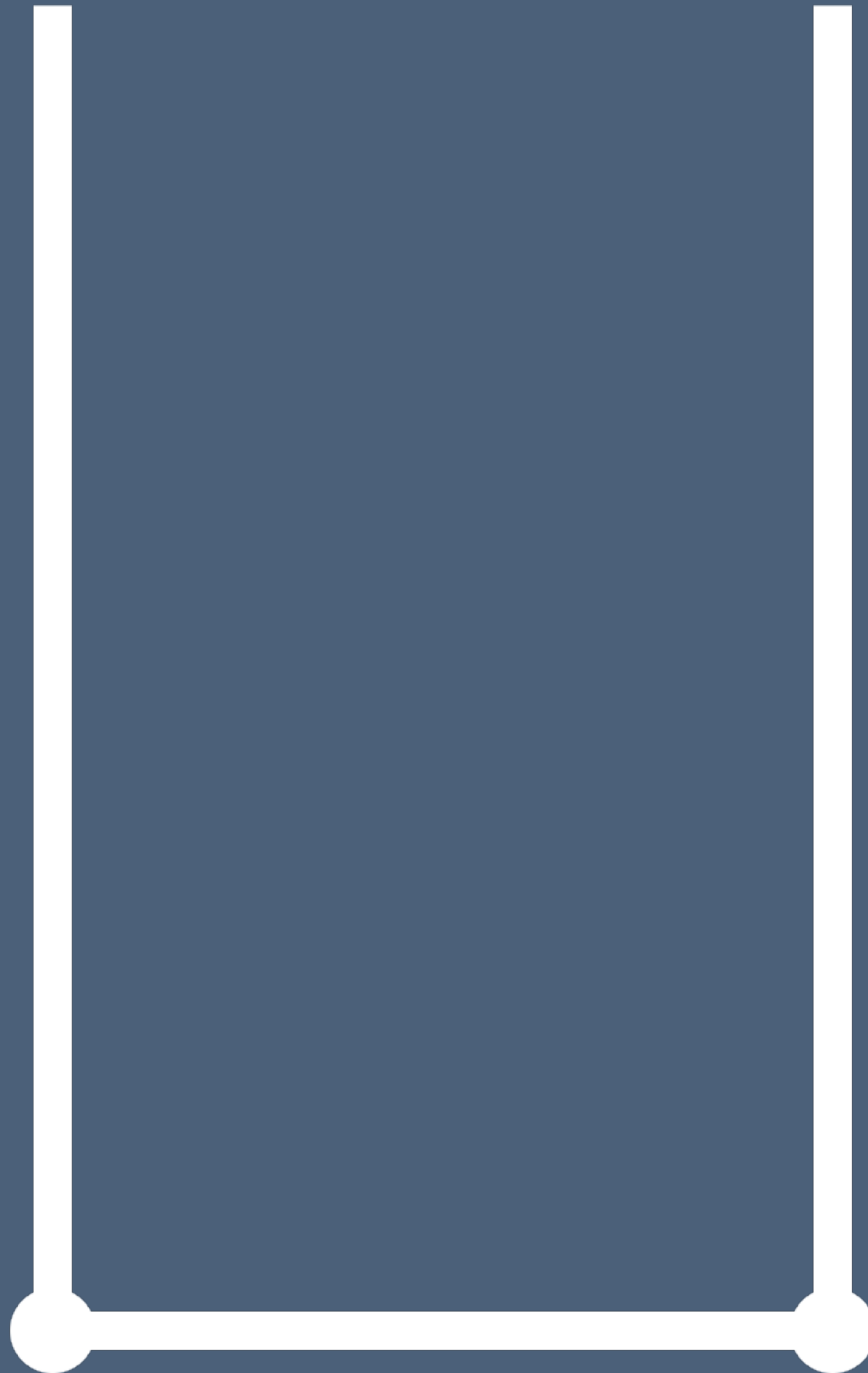
e como isso tem tudo a ver com async/await do Javascript

# Pilhas

- Guardar elementos
- LIFO (Last In, First Out)
- Apenas uma direção de entrada e saída
- **push** e **pop**

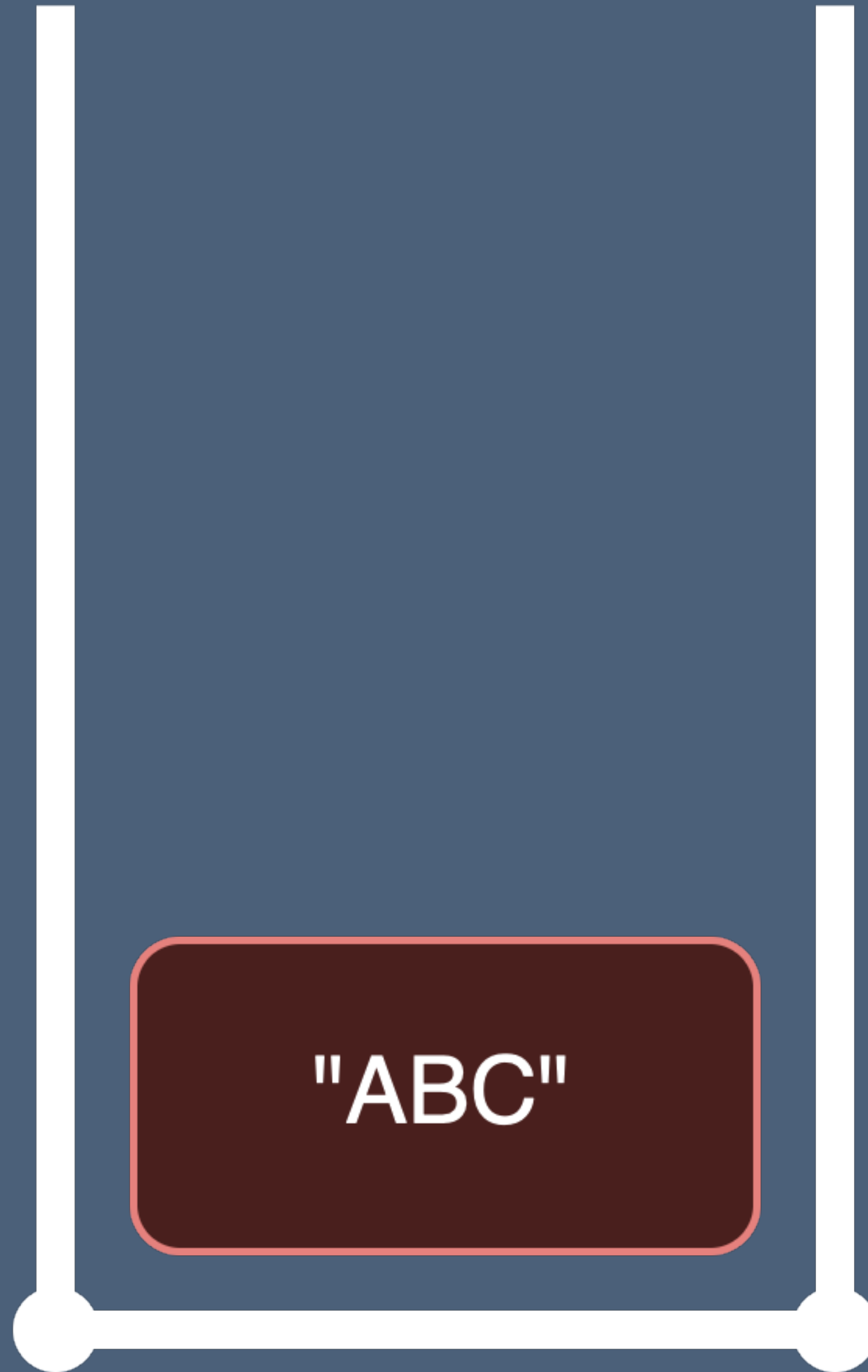


Pilhas



```
push("ABC")
```

# Pilhas



```
push("XYZ")
```

# Pilhas



```
push("FOO")
```



# Pilhas



Só tem uma direção!

pop()

# Pilhas



"FOO"

Só tem uma direção mesmo!

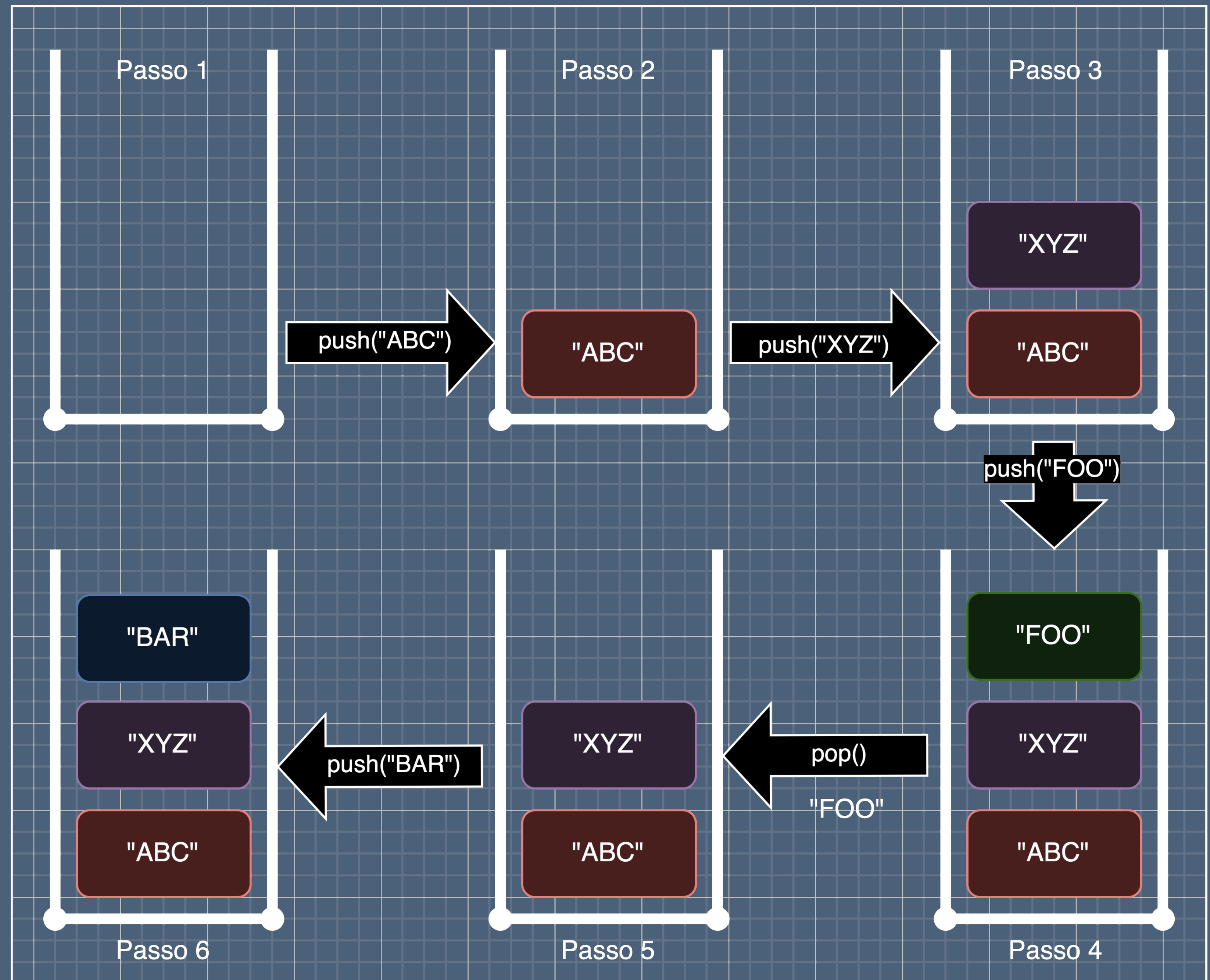
push("BAR")

# Pilhas



# Pilhas

- Guardar elementos
- LIFO (Last In, First Out)
- Apenas uma direção de entrada e saída
- **push** e **pop**





# Pilhas - casos de uso

- **Quero saber para onde voltar**
- Navegação de telas
  - A nova tela é empilhada (push)
  - O voltar faz com que a tela seja desempilhada (pop)
- Execução de métodos!!!! (Vamos ver exemplos)

Curso Alura de estrutura de dados



<https://www.alura.com.br/artigos/estrutura-dados-computacao-na-pratica-com-java>

Mas e o async/await?

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```

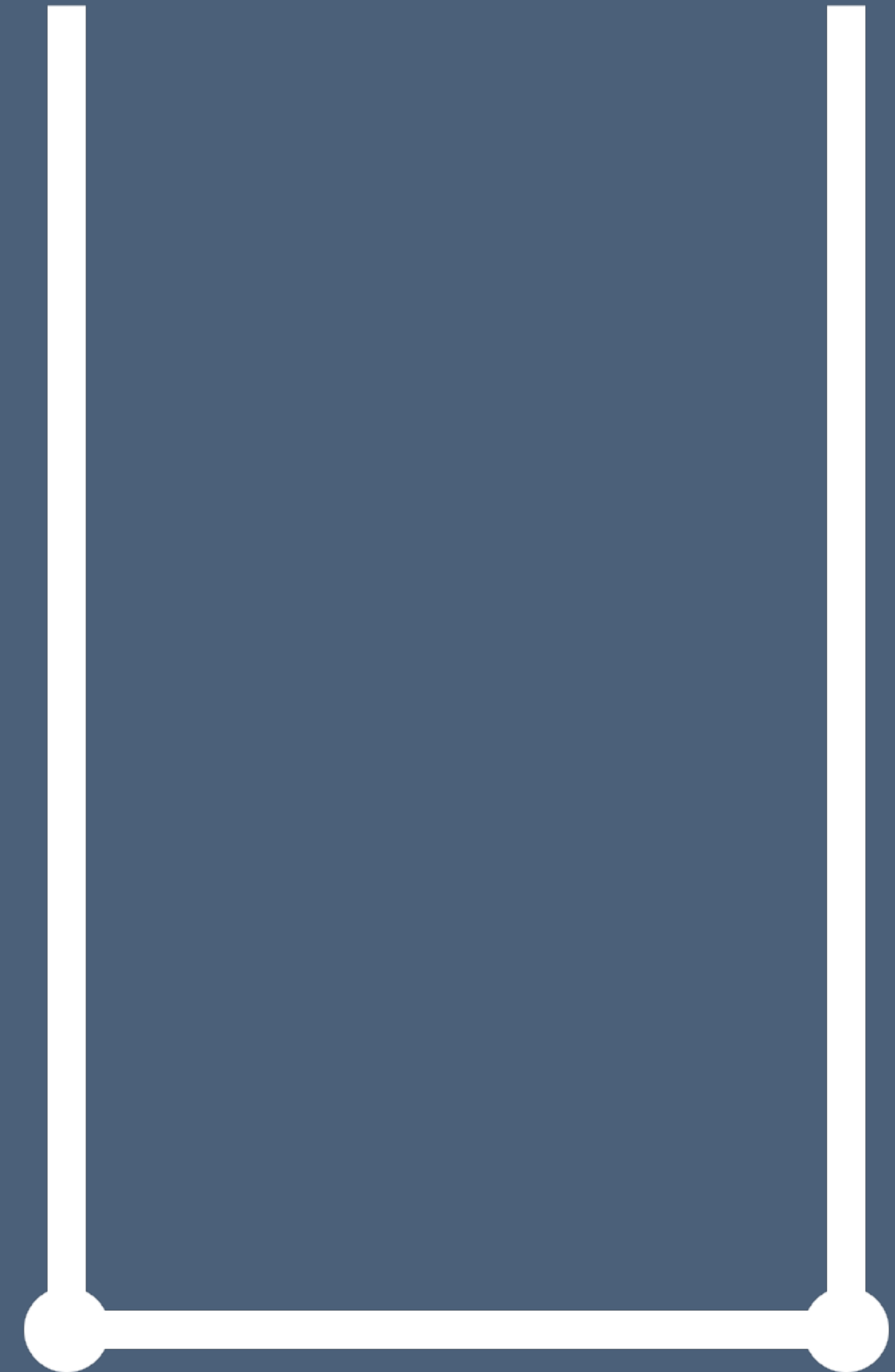
# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}
```

```
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}
```

```
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}
```

1 → `abc();`



# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
1 function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```



The diagram illustrates a function call stack. It consists of a white L-shaped frame with rounded corners. Inside the frame, at the bottom, is a dark red rounded rectangle with a thin red border. The text "abc()" is written in white inside this rectangle. The frame is positioned on the right side of the slide.

abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
1  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```

log()

abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
2 function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
1 function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```

xyz()

abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```

2 →

1 →

log()

xyz()

abc()



# Execução de métodos

```
3 function foo() {  
    console.log("começo F00");  
    //metodo foo  
    console.log("fim F00");  
}  
  
function xyz() {  
    console.log("começo XYZ");  
2   foo();  
    console.log("fim XYZ");  
}  
  
function abc() {  
    console.log("começo ABC");  
1   xyz();  
    console.log("fim ABC");  
}  
  
abc();
```



# Execução de métodos

```
function foo() {  
3 console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
2 console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
1 console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```

log()

foo()

xyz()

abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```

foo()

xyz()

abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
3 console.log("fim F00");  
  
function xyz() {  
  console.log("começo XYZ");  
2 foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
1 xyz();  
  console.log("fim ABC");  
}  
  
abc();
```

log()

foo()

xyz()

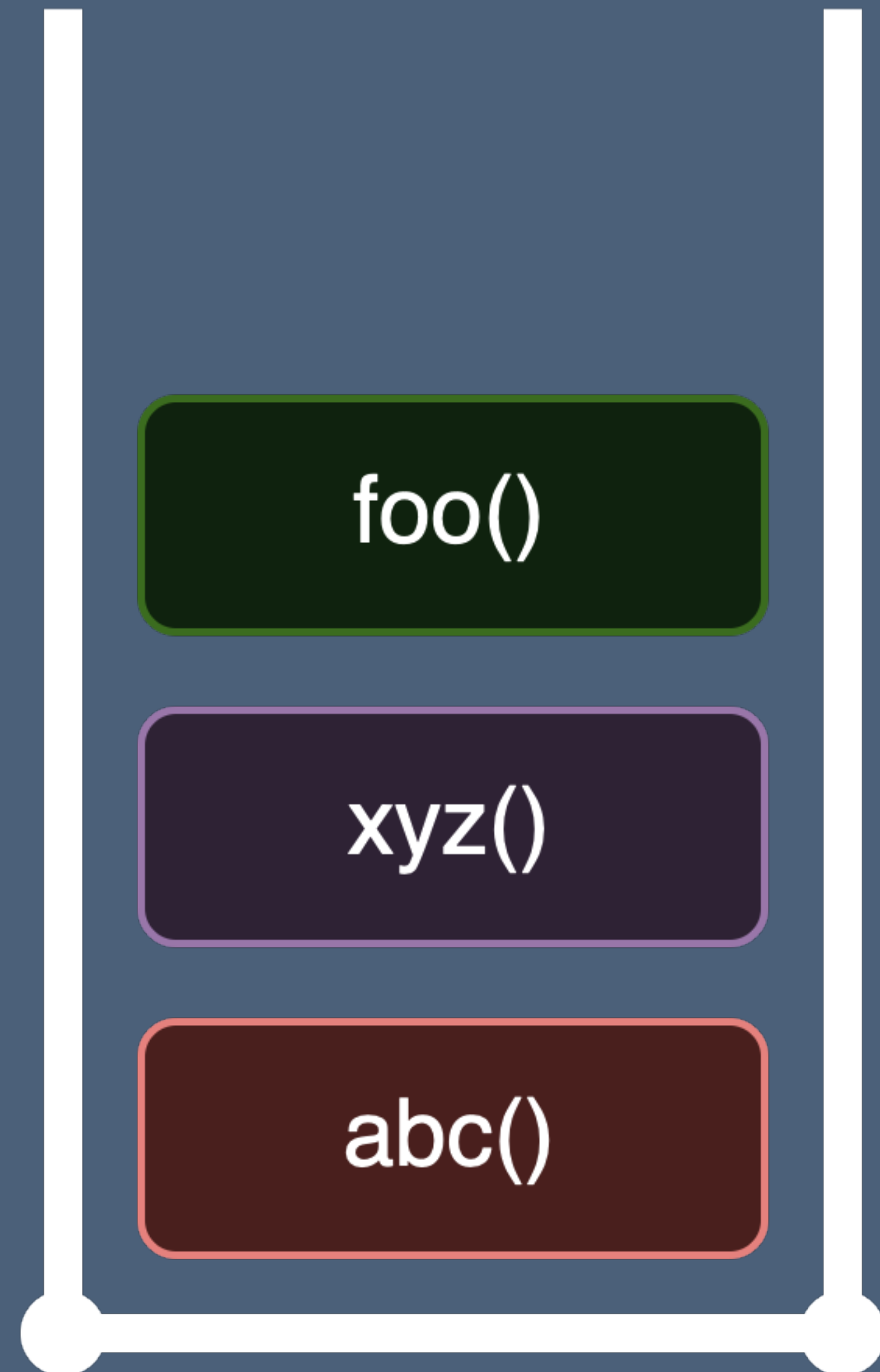
abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
abc();
```

Diagram illustrating the execution flow of the code:

- 1. Execution starts at the call to `abc()`.
- 2. Execution enters the `abc()` function, which calls `xyz()`.
- 3. Execution enters the `xyz()` function, which calls `foo()`.



# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  2 → foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  1 → xyz();  
  console.log("fim ABC");  
}  
  
abc();
```



# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}
```

```
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}
```

```
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}
```

```
abc();
```

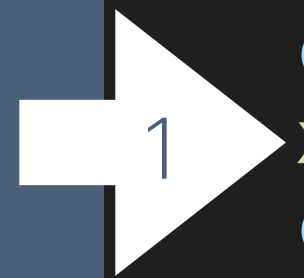
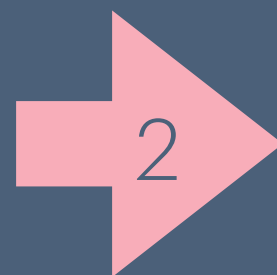
log()

xyz()

abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```





# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  1 xyz();  
  console.log("fim ABC");  
}  
  
abc();
```



abc()

# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}
```

```
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}
```

```
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}
```

1

```
abc();
```

log()

abc()

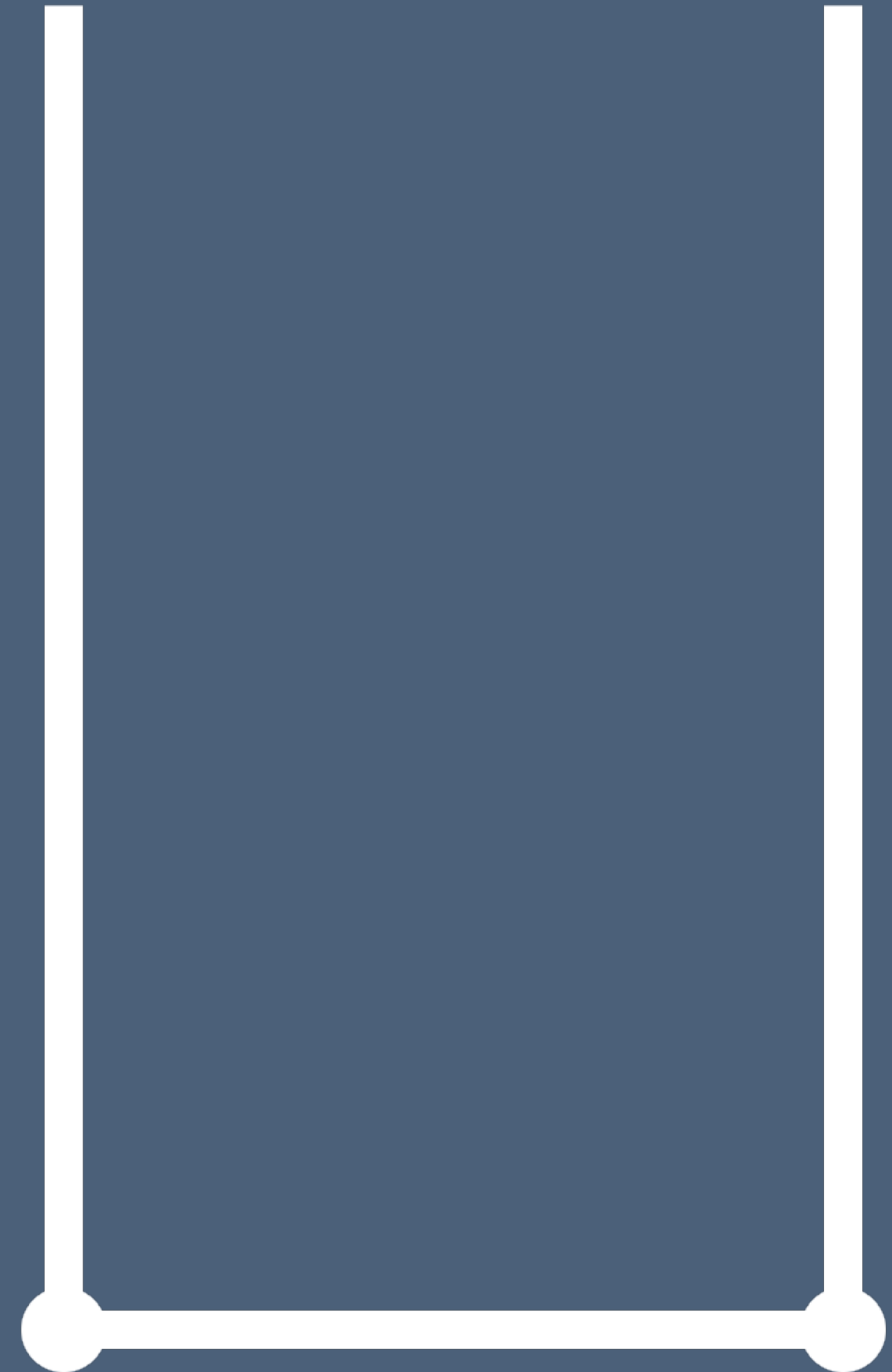
# Execução de métodos

```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
abc();
```



# Execução de métodos

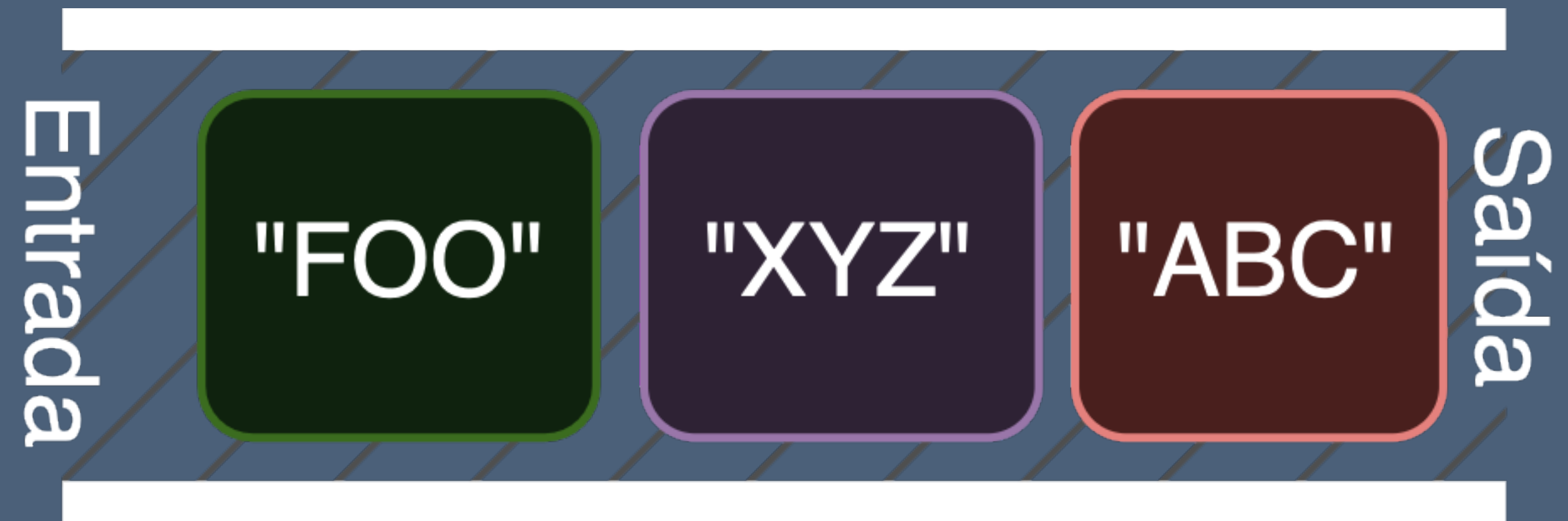
```
function foo() {  
  console.log("começo F00");  
  //metodo foo  
  console.log("fim F00");  
}  
  
function xyz() {  
  console.log("começo XYZ");  
  foo();  
  console.log("fim XYZ");  
}  
  
function abc() {  
  console.log("começo ABC");  
  xyz();  
  console.log("fim ABC");  
}  
  
abc();
```



Mas e o async/await?

# Filas

- FIFO (First In, First Out)
- Uma direção de entrada e OUTRA de saída
- `queue(...)` e `dequeue()`



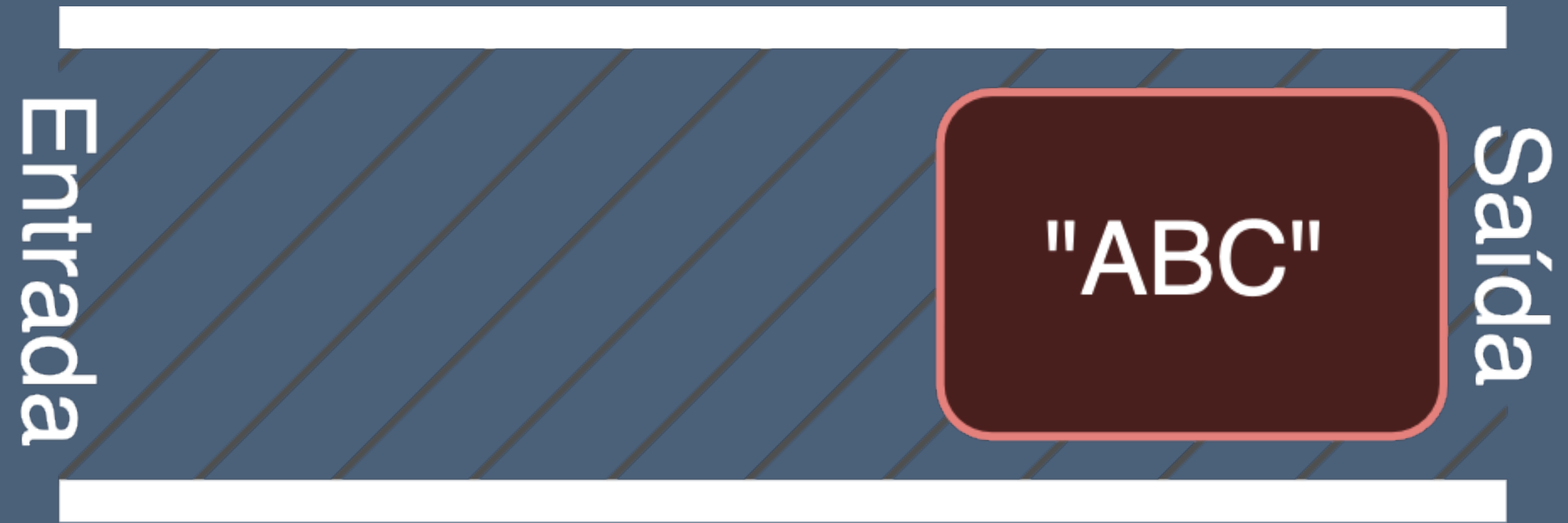
# Filas



```
enqueue("ABC")
```



# Filas



```
enqueue("XYZ")
```

# Filas



```
enqueue("FOO")
```

# Filas



Agora tem a outra direção

dequeue()

# Filas





# Pilhas



# Comparando com pilhas



"FOO"

# Filas



# Filas



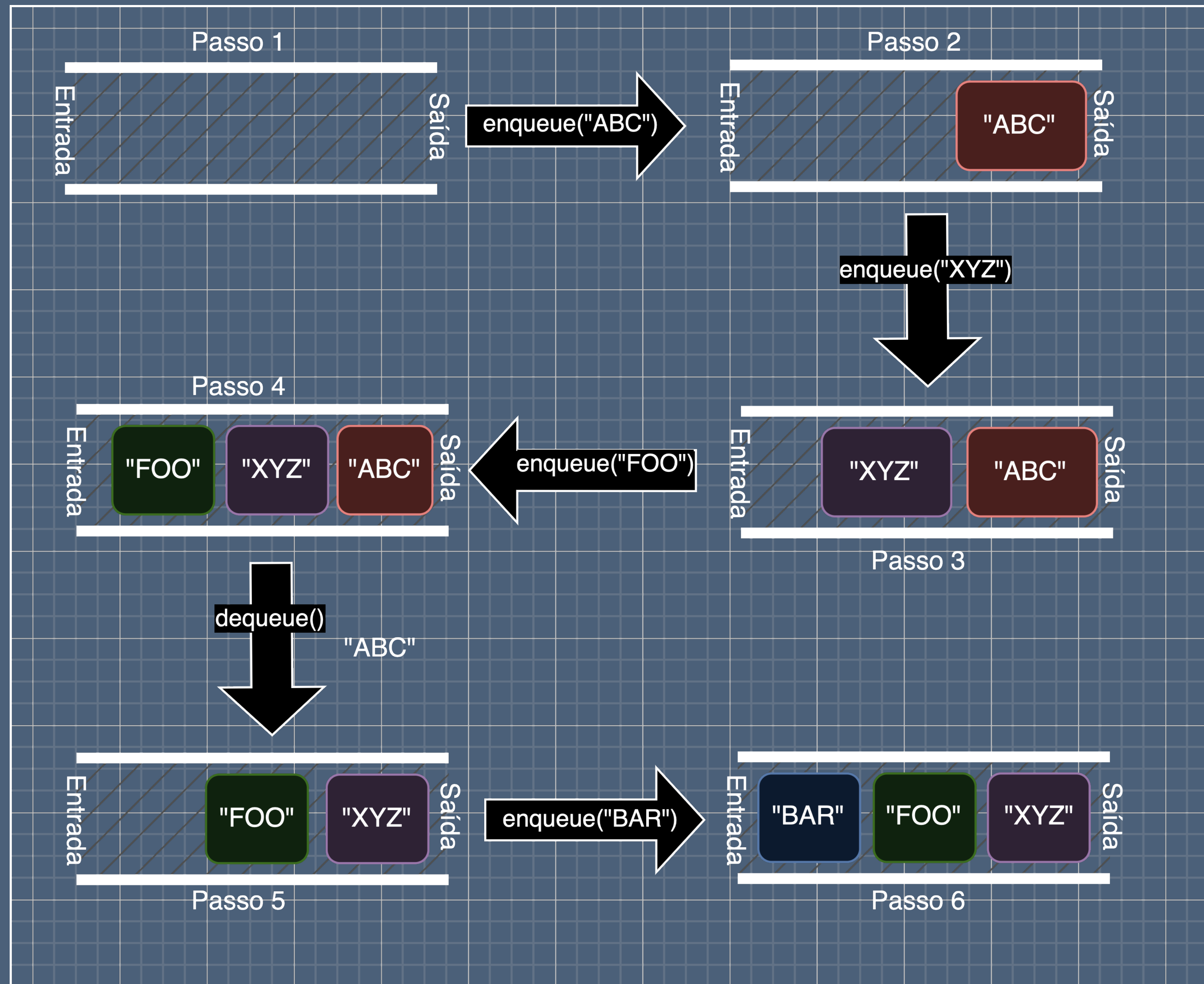
enqueue("BAR")

# Filas



# Filas

- FIFO (First In, First Out)
- Uma direção de entrada e OUTRA de saída
- queue(...) e dequeue()



# Filas - casos de uso

- **Quem é o próximo?**
- Processamento de mensagens (na ordem)
- Próximo código assíncrono a ser executado



Antes vamos ver um pouco de  
código assíncrono

# Execução de métodos assíncronos

```
function xpto() {  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
}  
  
xpto();
```

```
$ node fila_01.js  
Mensagem 1  
Mensagem 2
```

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```

O `setTimeout(..)` agenda uma execução para daqui X ms

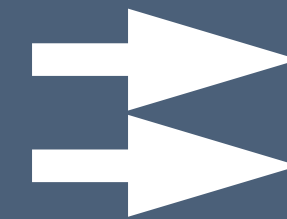
# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```

```
$ node fila_02.js  
Começo XPT0  
Final XPT0  
Mensagem 1  
Mensagem 2
```

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
    → 100);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
    → 50);  
  console.log("Final XPT0");  
}  
  
xpto();
```



```
$ node fila_03.js  
Começo XPT0  
Final XPT0  
Mensagem 2  
Mensagem 1
```

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 0);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 0);  
  console.log("Final XPT0");  
}  
  
xpto();
```

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 0);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 0);  
  console.log("Final XPT0");  
}  
  
xpto();
```

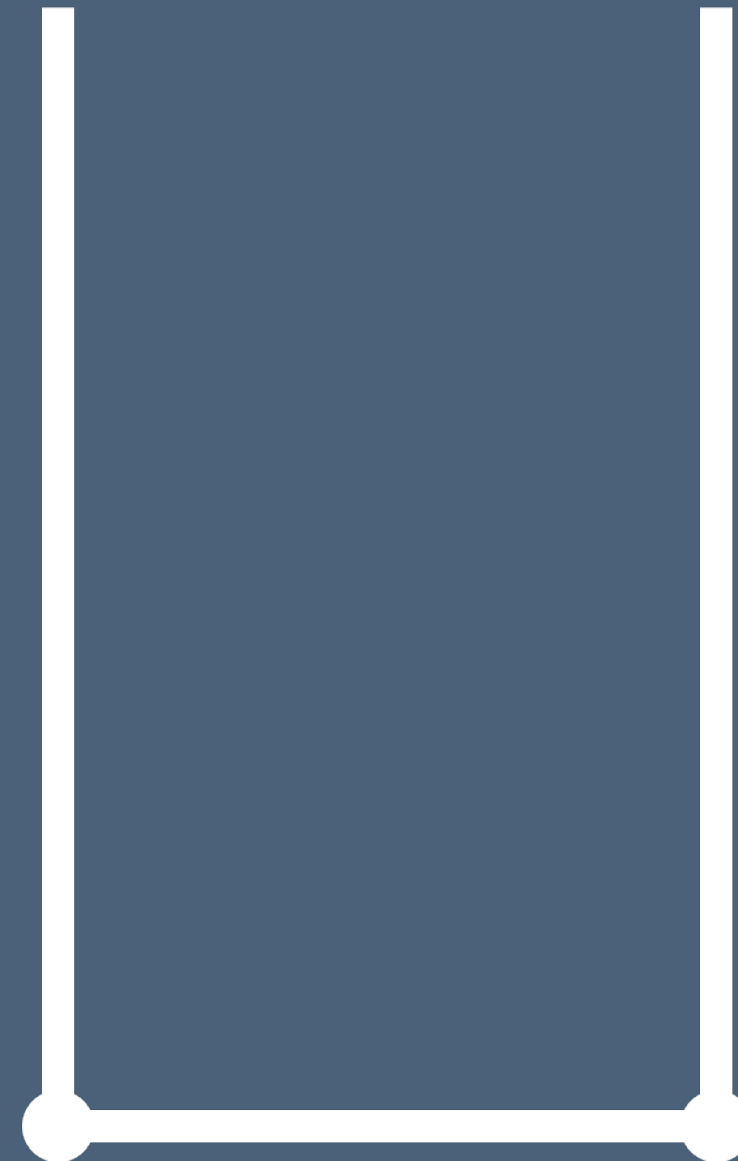
```
$ node fila_04.js  
Começo XPT0  
Final XPT0  
Mensagem 1  
Mensagem 2
```




Vamos entender o  
setTimeout(..) a fundo e o que  
significa o JS ser Single-thread

# Execução de métodos assíncronos

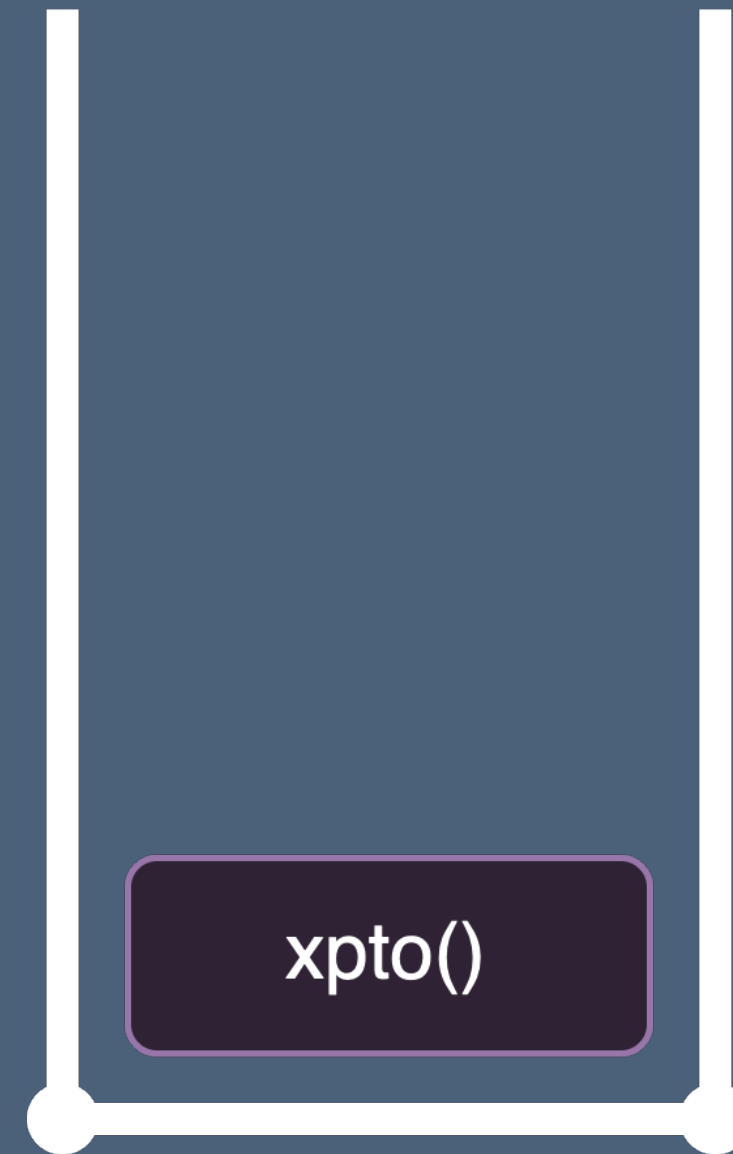
```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



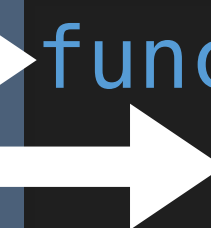
# Execução de métodos assíncronos



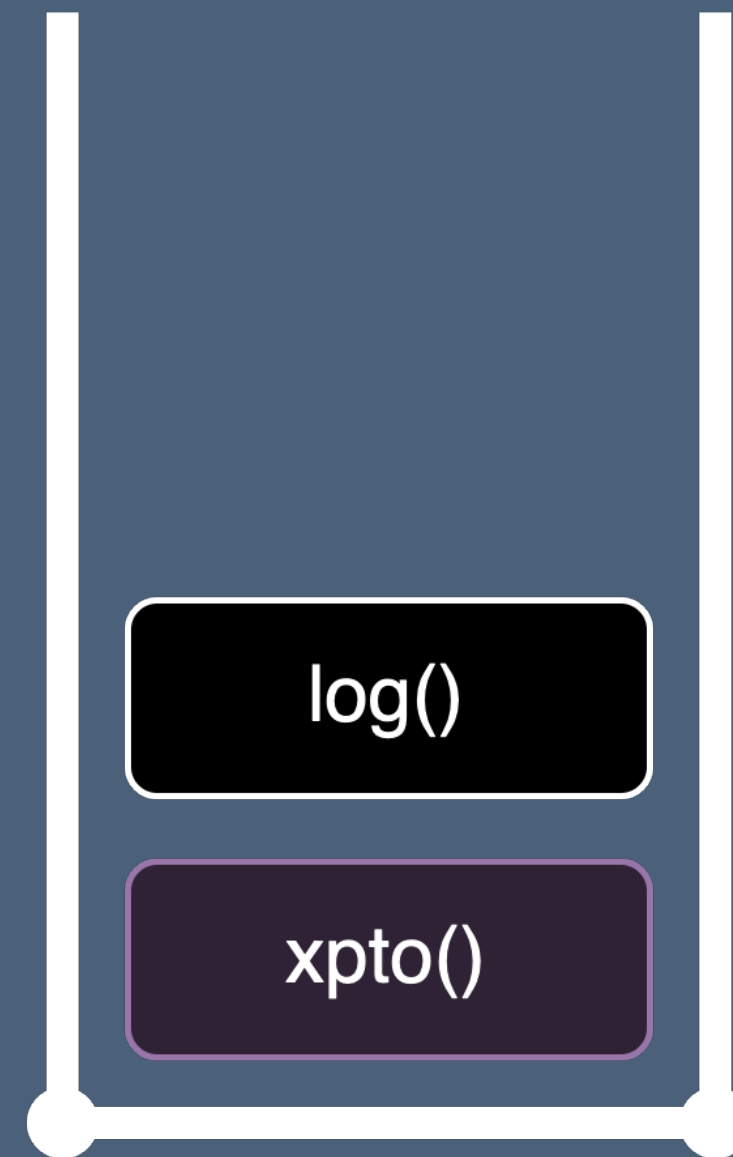
```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



# Execução de métodos assíncronos

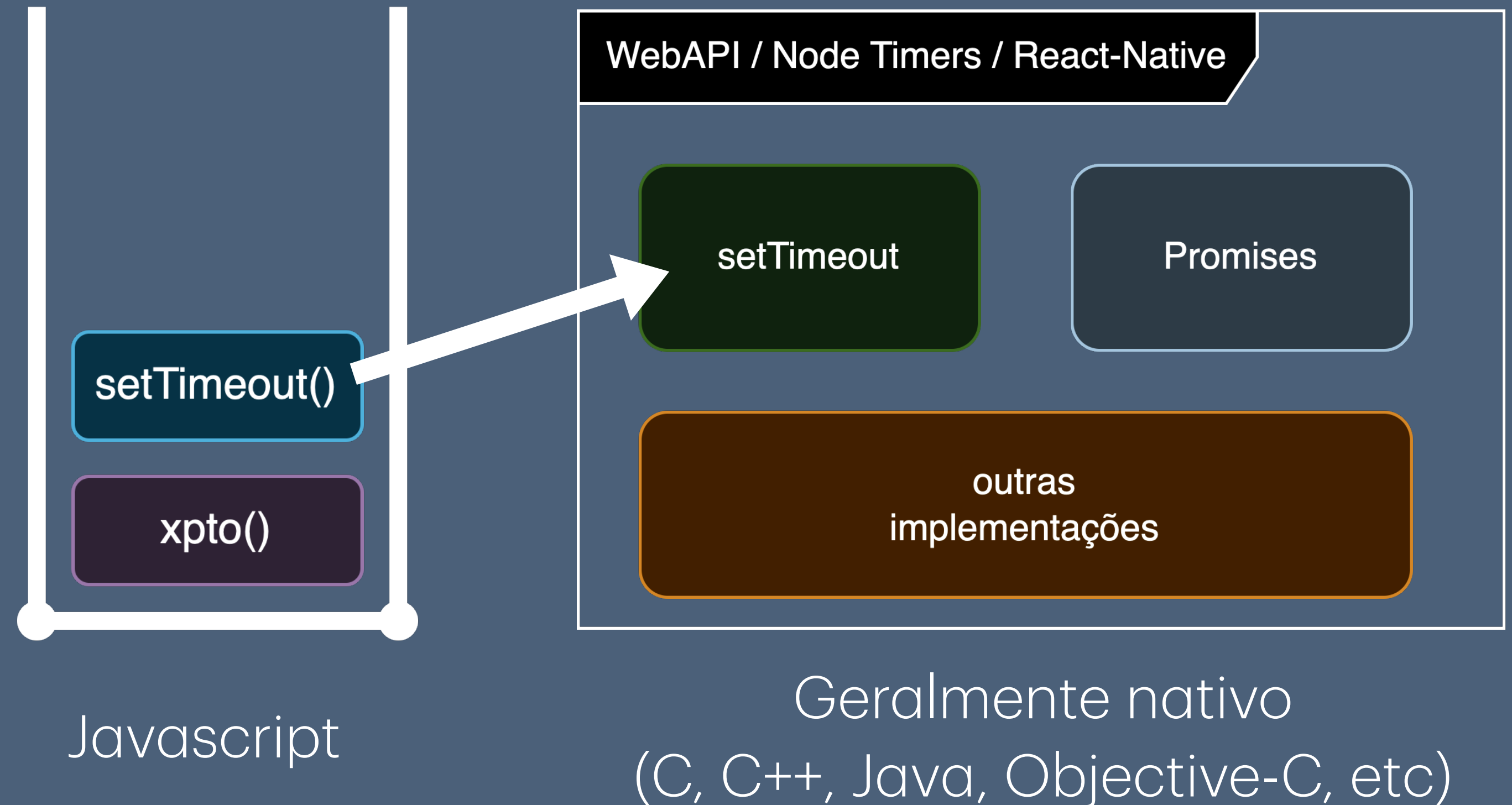


```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



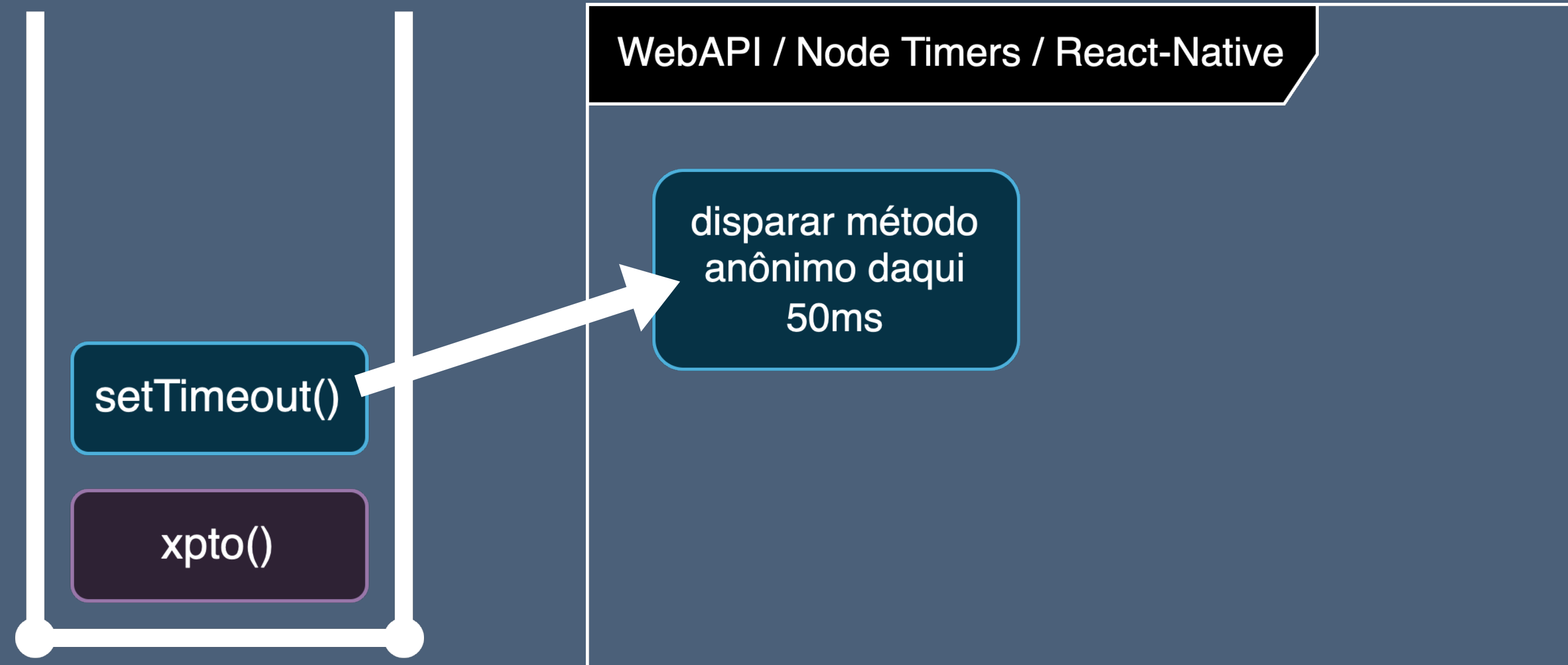
# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



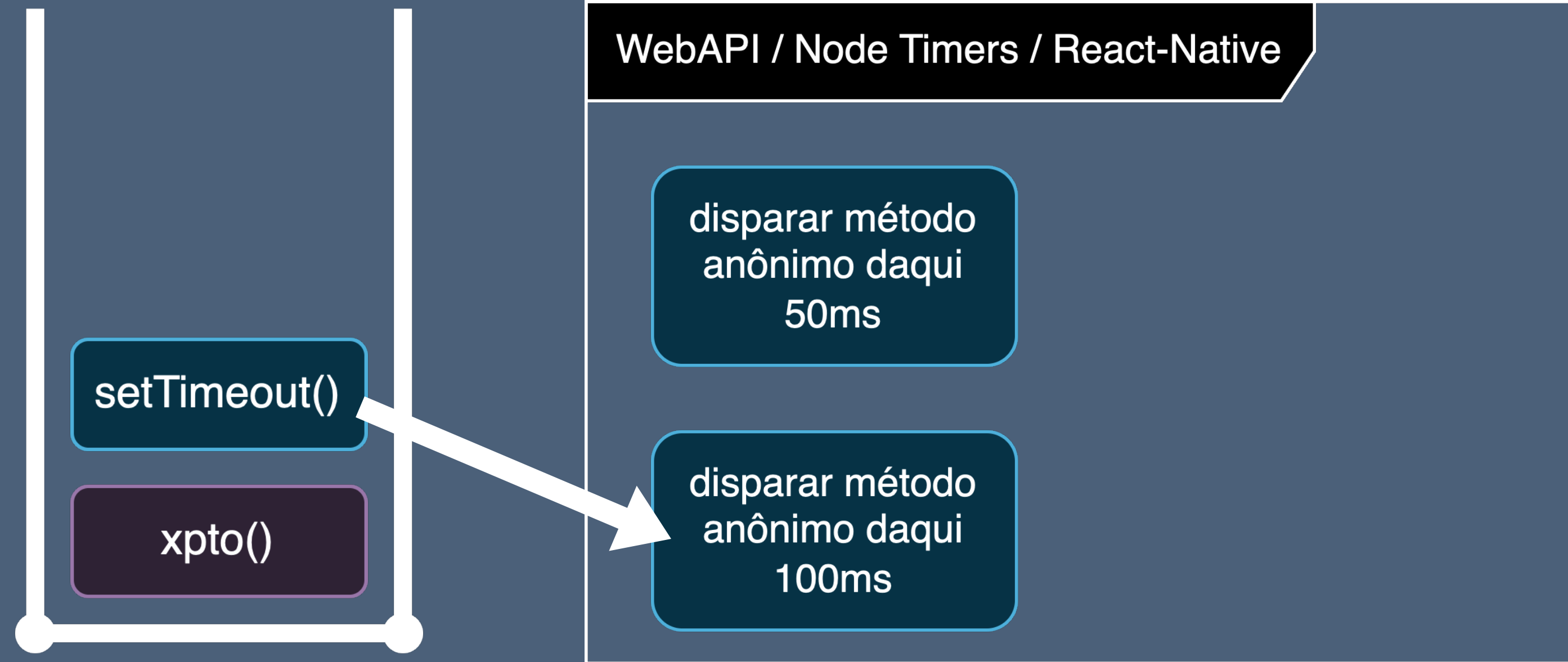
# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



WebAPI / Node Timers / React-Native

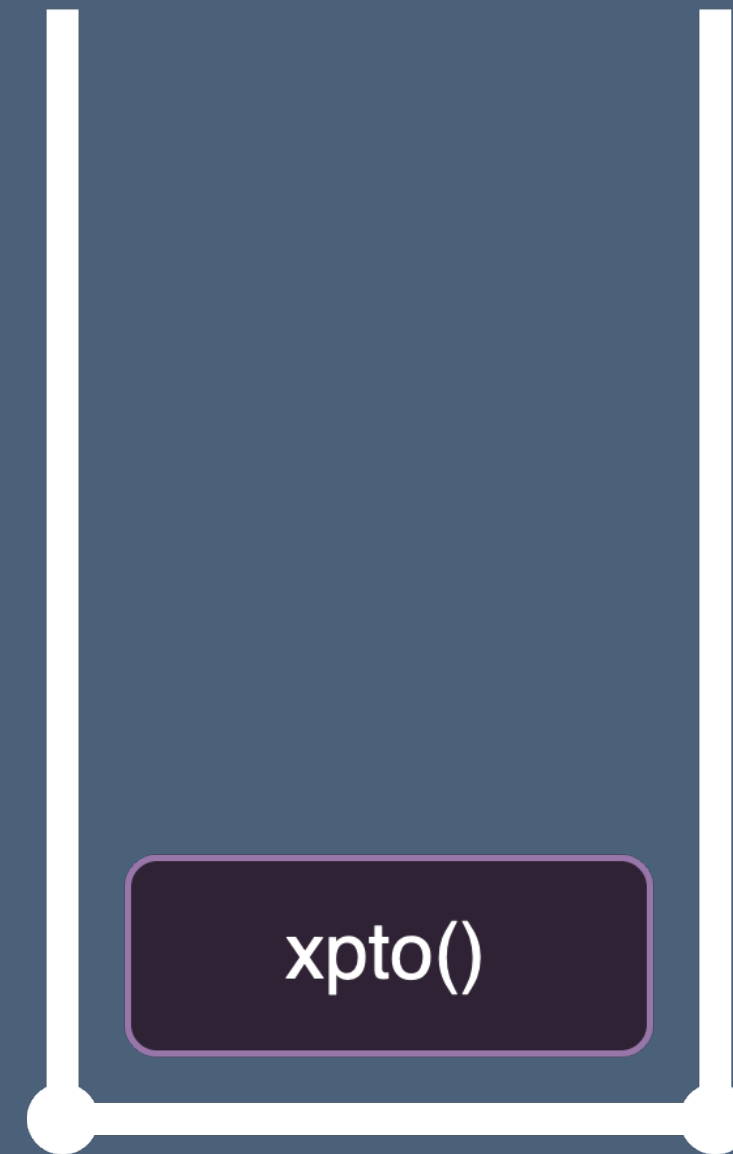
disparar método  
anônimo daqui  
50ms

disparar método  
anônimo daqui  
100ms



# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
xpto();
```



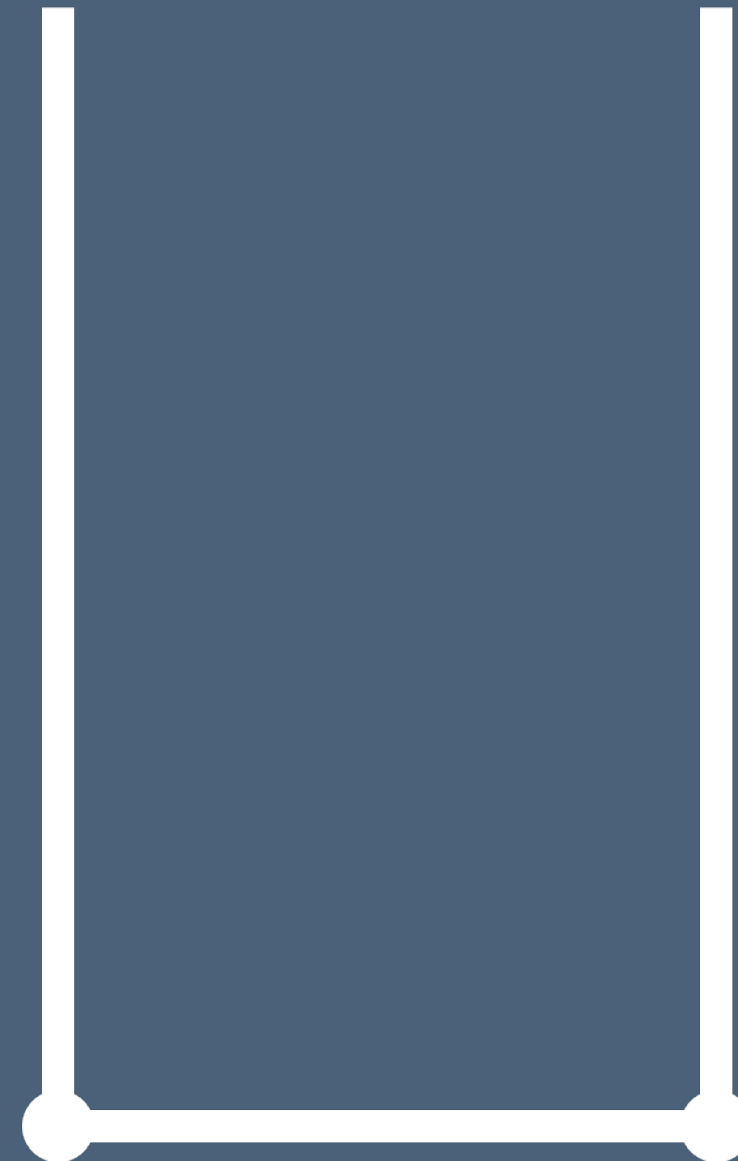
WebAPI / Node Timers / React-Native

disparar método  
anônimo daqui  
50ms

disparar método  
anônimo daqui  
100ms

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



WebAPI / Node Timers / React-Native

disparar método  
anônimo daqui  
50ms

disparar método  
anônimo daqui  
100ms

E depois dos 50ms e dos 100ms?

Fila!

# Task Queue

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



WebAPI / Node Timers / React-Native

disparar método  
anônimo daqui  
50ms

disparar método  
anônimo daqui  
100ms

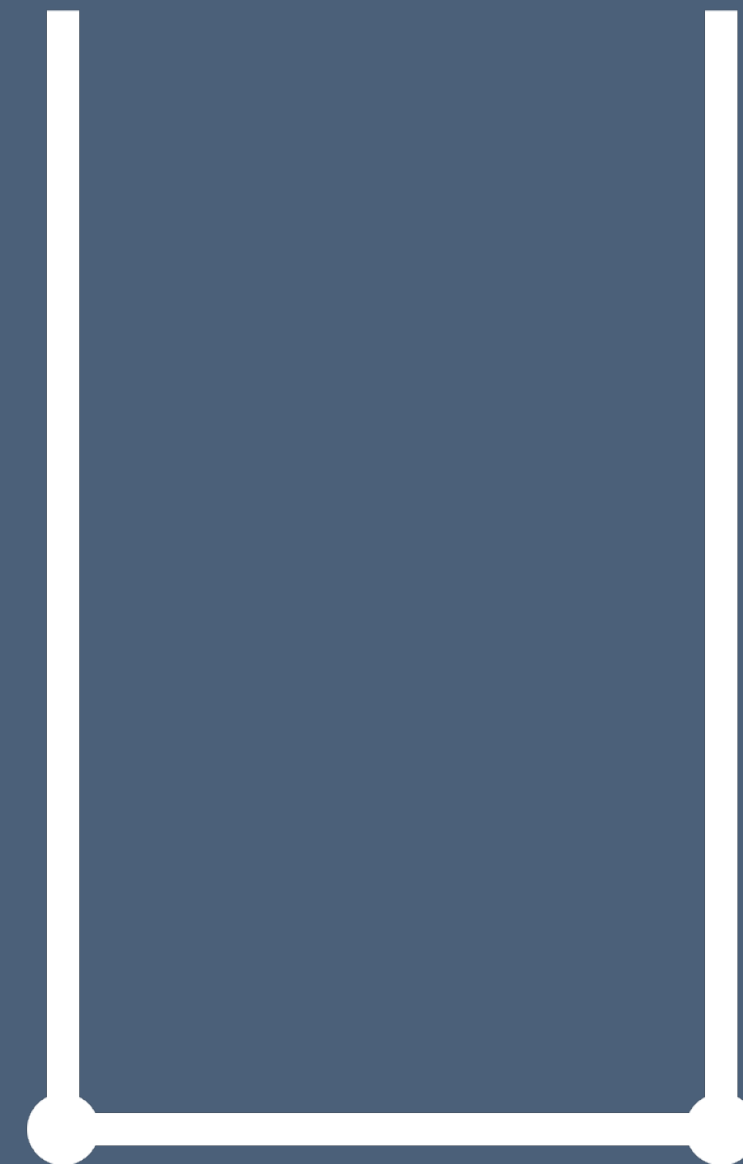
Saída

Entrada

Task Queue

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



WebAPI / Node Timers / React-Native

disparar método  
anônimo daqui  
50ms

Saída

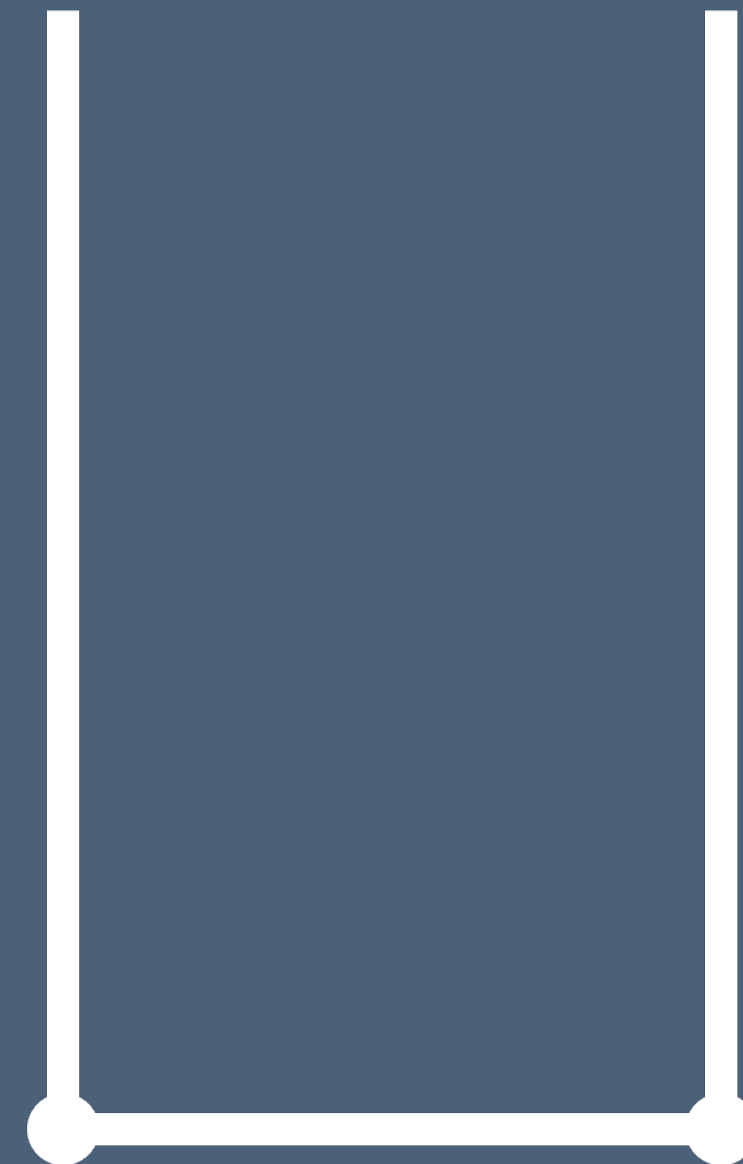
```
()=>{  
  console.log("Mensagem 1");  
}
```

Entrada

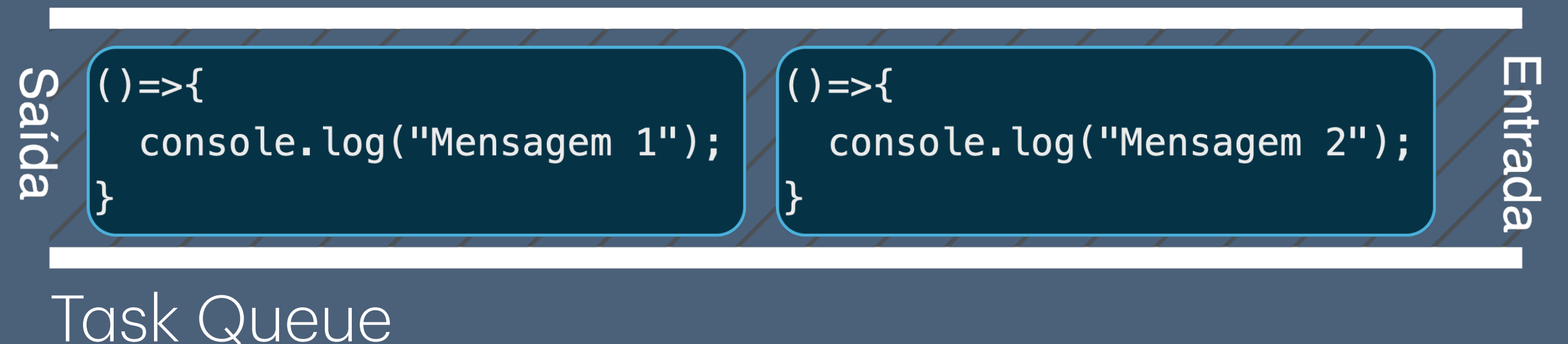
Task Queue

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```



WebAPI / Node Timers / React-Native



# Event Loop



# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 50);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 100);  
  console.log("Final XPT0");  
}  
  
xpto();
```

Pega algo da task  
queue para executar

Event  
Loop

A pilha está vazia?

Saída

```
()=>{  
  console.log("Mensagem 1");  
}
```

```
()=>{  
  console.log("Mensagem 2");  
}
```

Entrada

Task Queue

WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
()=>{  
  console.log("Mensagem 1");  
}
```

anonymous()

WebAPI / Node Timers / React-Native


Saída

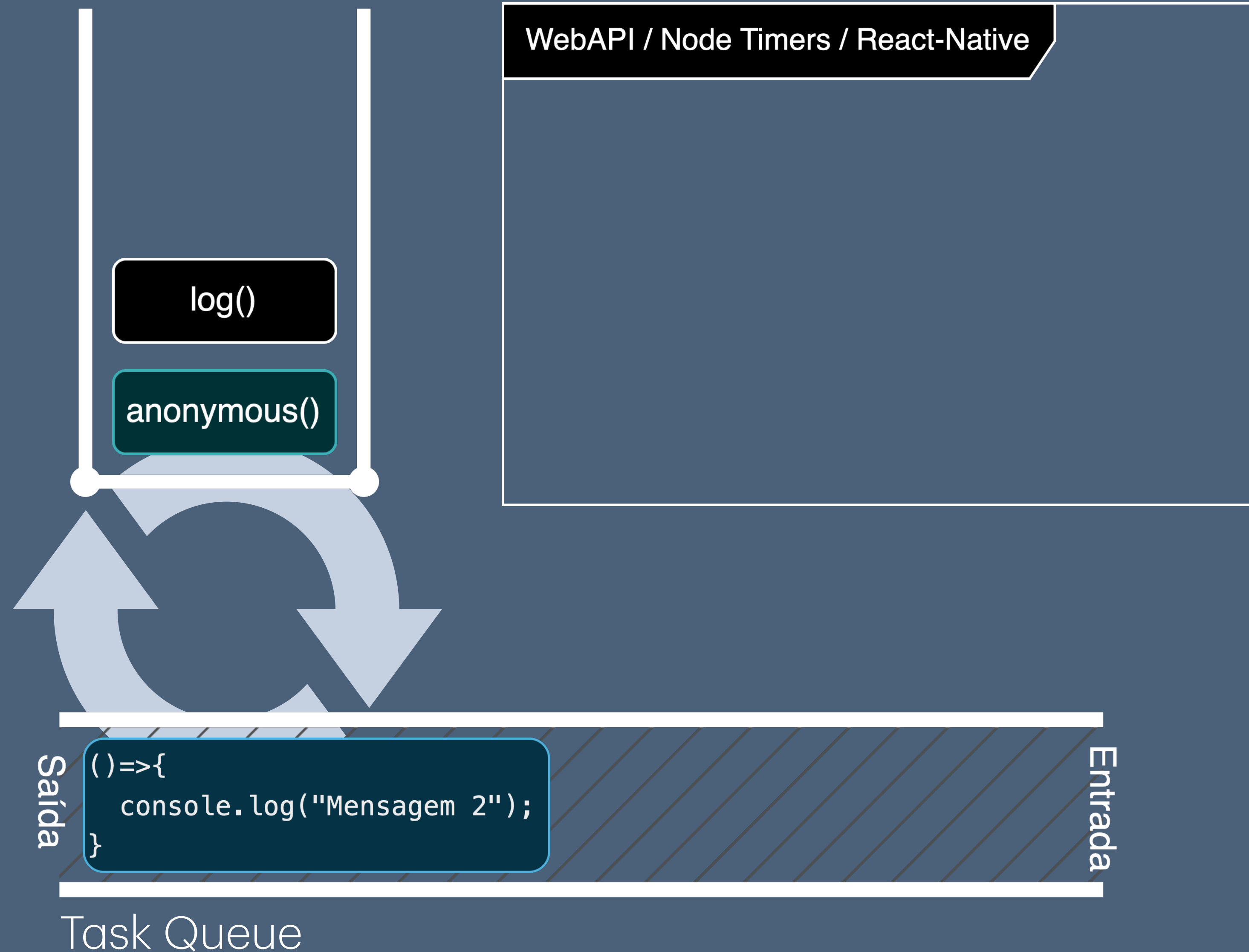
```
()=>{  
  console.log("Mensagem 2");  
}
```

Entrada

Task Queue

# Execução de métodos assíncronos

```
()=>{  
  console.log("Mensagem 1");  
}
```



# Execução de métodos assíncronos

```
()=>{  
  console.log("Mensagem 1");  
}
```

anonymous()

WebAPI / Node Timers / React-Native

Saída

```
()=>{  
  console.log("Mensagem 2");  
}
```

Entrada

Task Queue

# Execução de métodos assíncronos

```
()=>{  
  console.log("Mensagem 2");  
}
```

anonymous()

WebAPI / Node Timers / React-Native

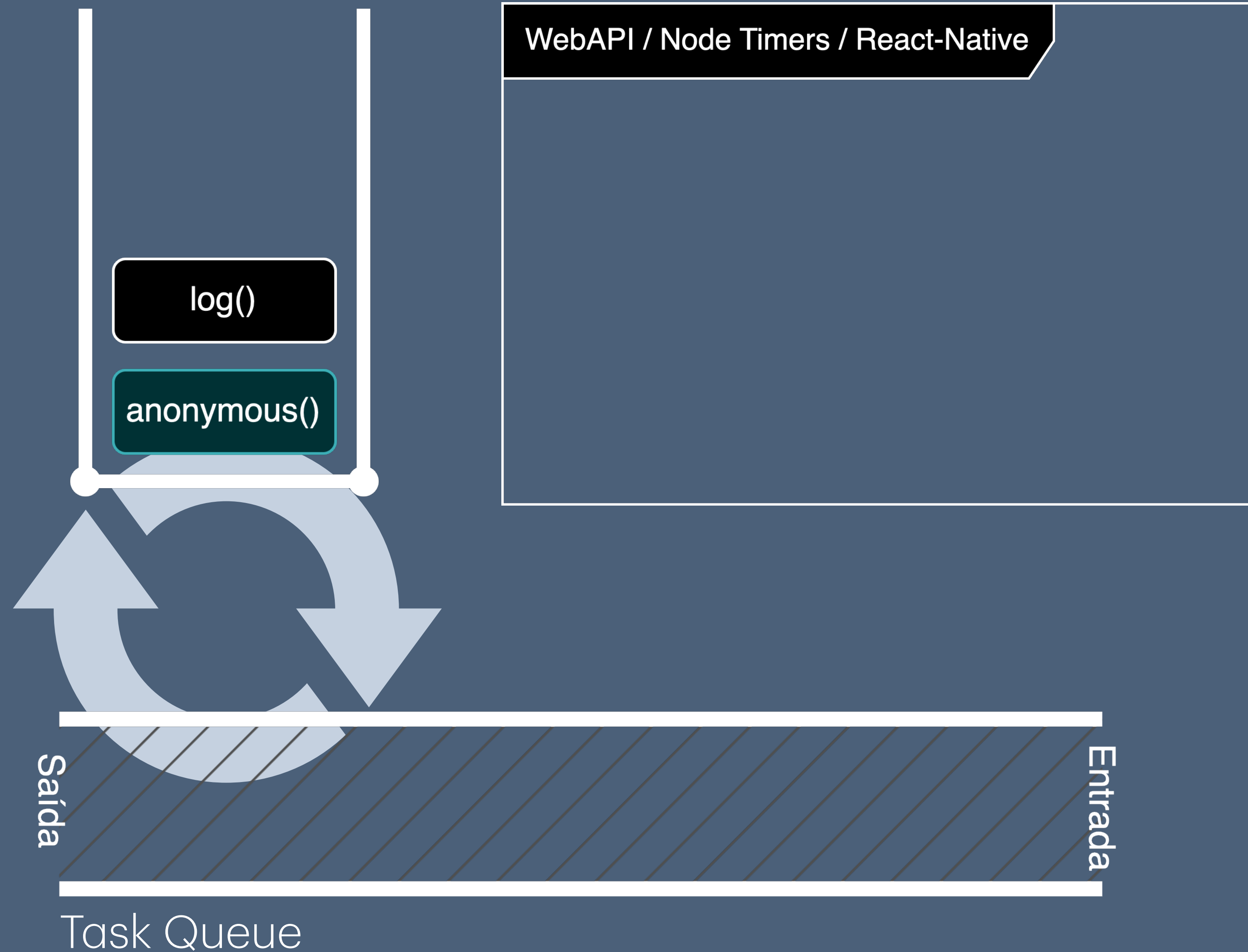
Saída

Entrada

Task Queue

# Execução de métodos assíncronos

```
→ () => {  
→ console.log("Mensagem 2");  
}
```



# Execução de métodos assíncronos

```
()=>{  
  console.log("Mensagem 2");  
}
```

anonymous()

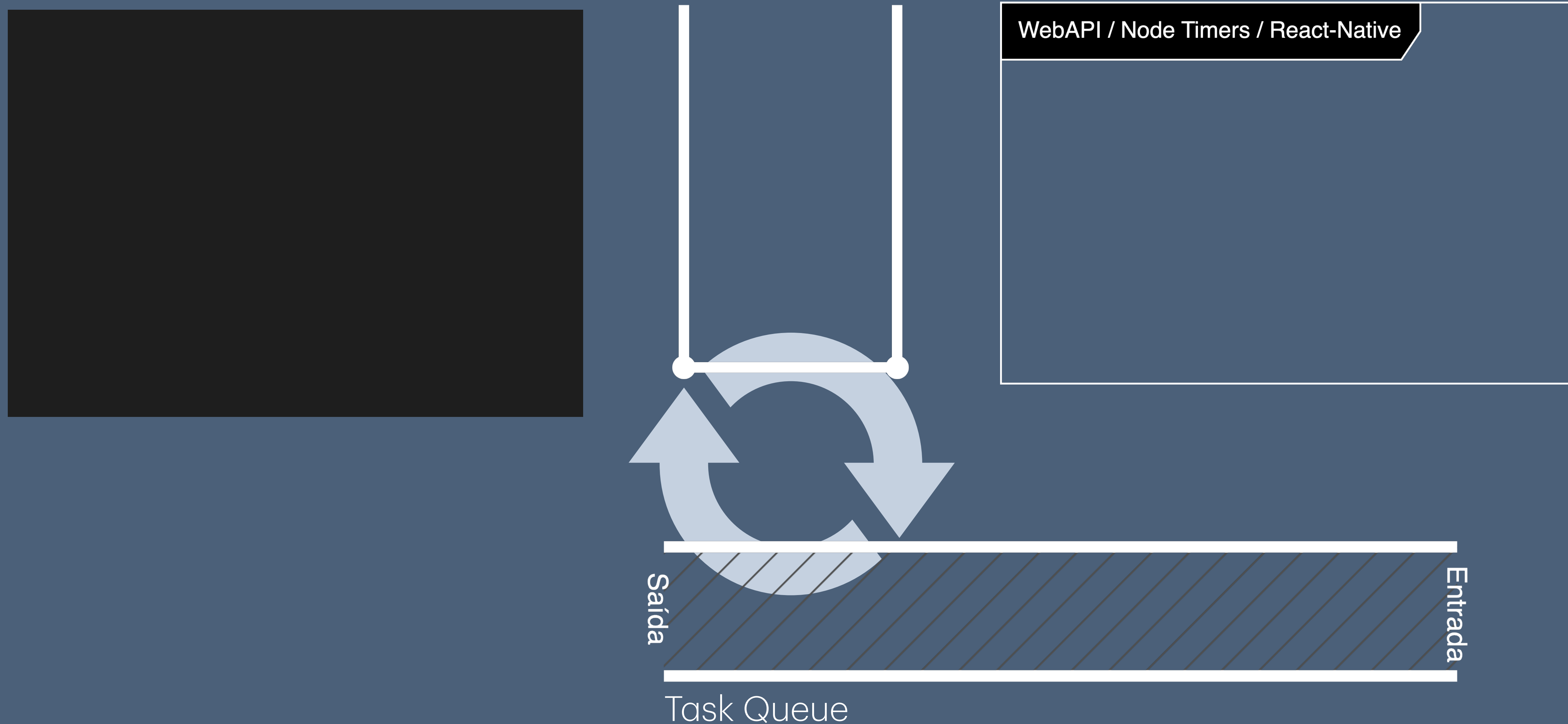
WebAPI / Node Timers / React-Native

Saída

Entrada

Task Queue

# Execução de métodos assíncronos





E as Promises?

São muito parecidas com  
setTimeout!

# Execução dos setTimeouts

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(()=>{  
    console.log("Mensagem 1");  
  }, 0);  
  setTimeout(()=>{  
    console.log("Mensagem 2");  
  }, 0);  
  console.log("Final XPT0");  
}  
  
xpto();
```

```
$ node fila_02.js  
Começo XPT0  
Final XPT0  
Mensagem 1  
Mensagem 2
```

# Execução das promises

```
function xpto() {  
  console.log("Começo XPT0");  
  Promise.resolve(true).then(()=>{  
    console.log("Mensagem 1");  
  });  
  Promise.resolve(true).then(()=>{  
    console.log("Mensagem 2");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```


```
$ node promises_01.js  
Começo XPT0  
Final XPT0  
Mensagem 1  
Mensagem 2
```

# Execução de setTimeout e Promise

```
function xpto() {  
  console.log("Começo XPT0");  
  Promise.resolve(true).then(()=>{  
    console.log("Mensagem Promise");  
  });  
  setTimeout(()=>{  
    console.log("Mensagem setTimeout");  
  }, 0);  
  console.log("Final XPT0");  
}  
  
xpto();
```

```
$ node promises_02.js  
Começo XPT0  
Final XPT0  
Mensagem Promise  
Mensagem setTimeout
```

# Execução de setTimeout e Promise



```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```

```
$ node promises_03.js  
Começo XPT0  
Final XPT0  
Mensagem Promise  
Mensagem setTimeout
```

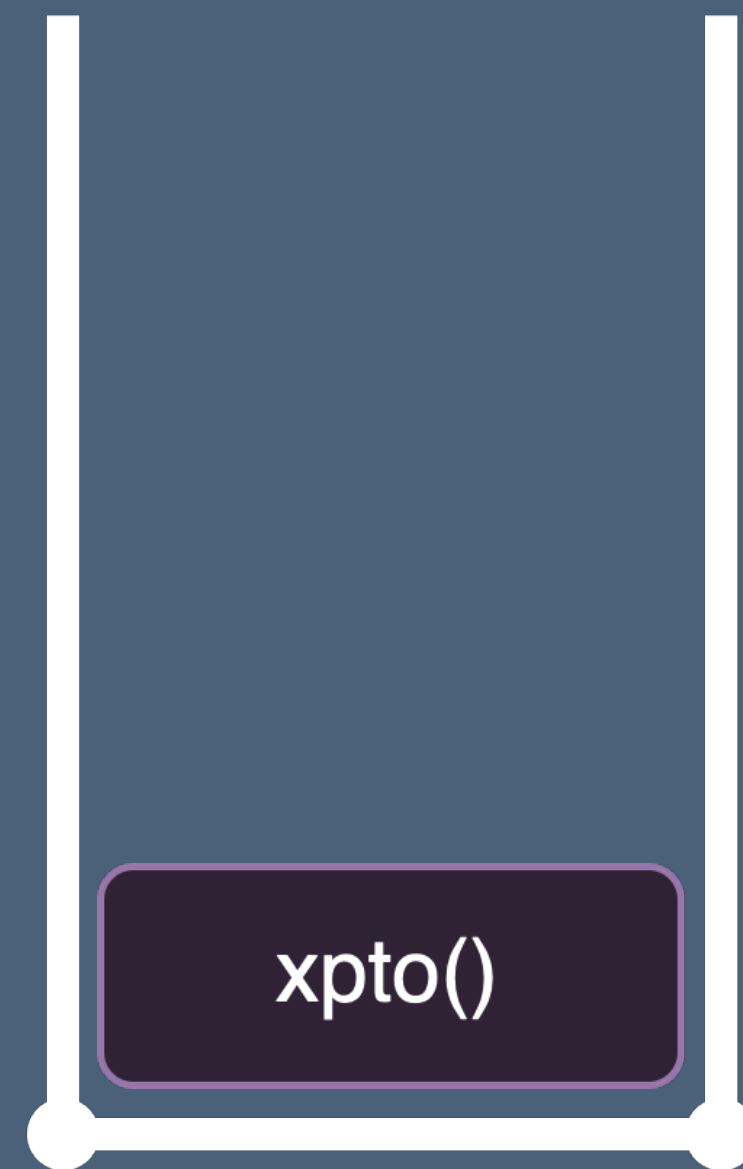
Por que isso aconteceu?

Porque ao invés de Task Queue,  
as promises têm a Job Queue



# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```



Job Queue



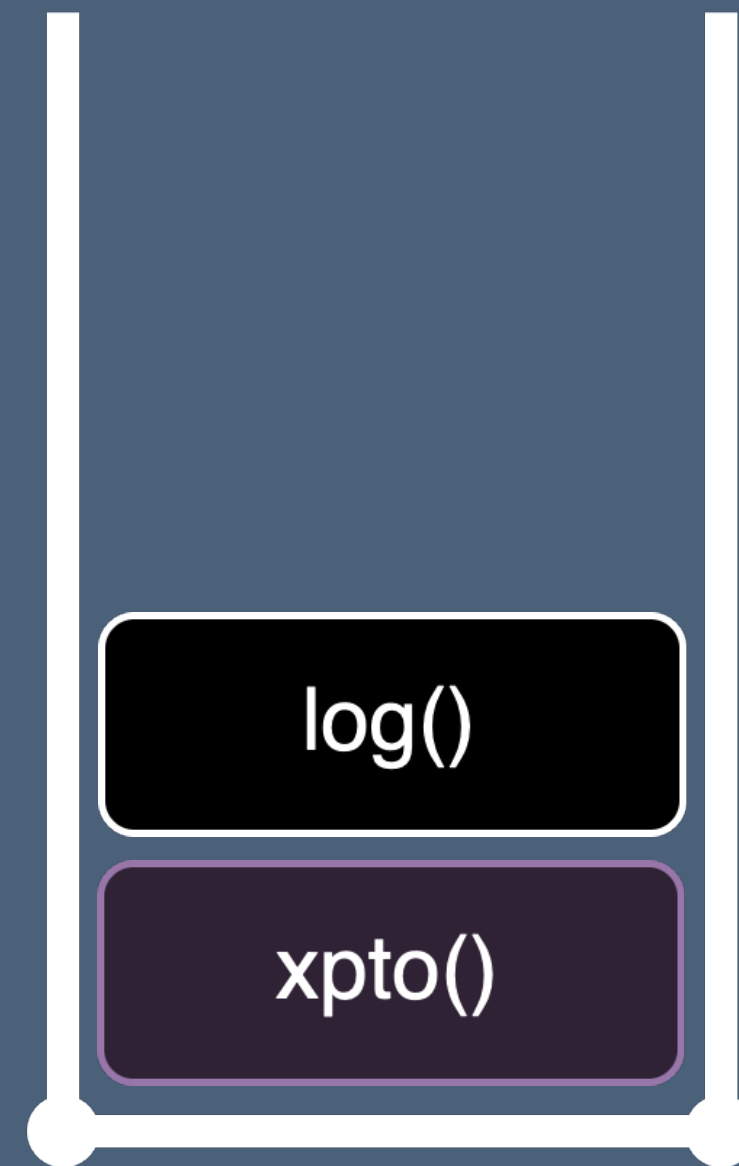
Task Queue



WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```



Job Queue



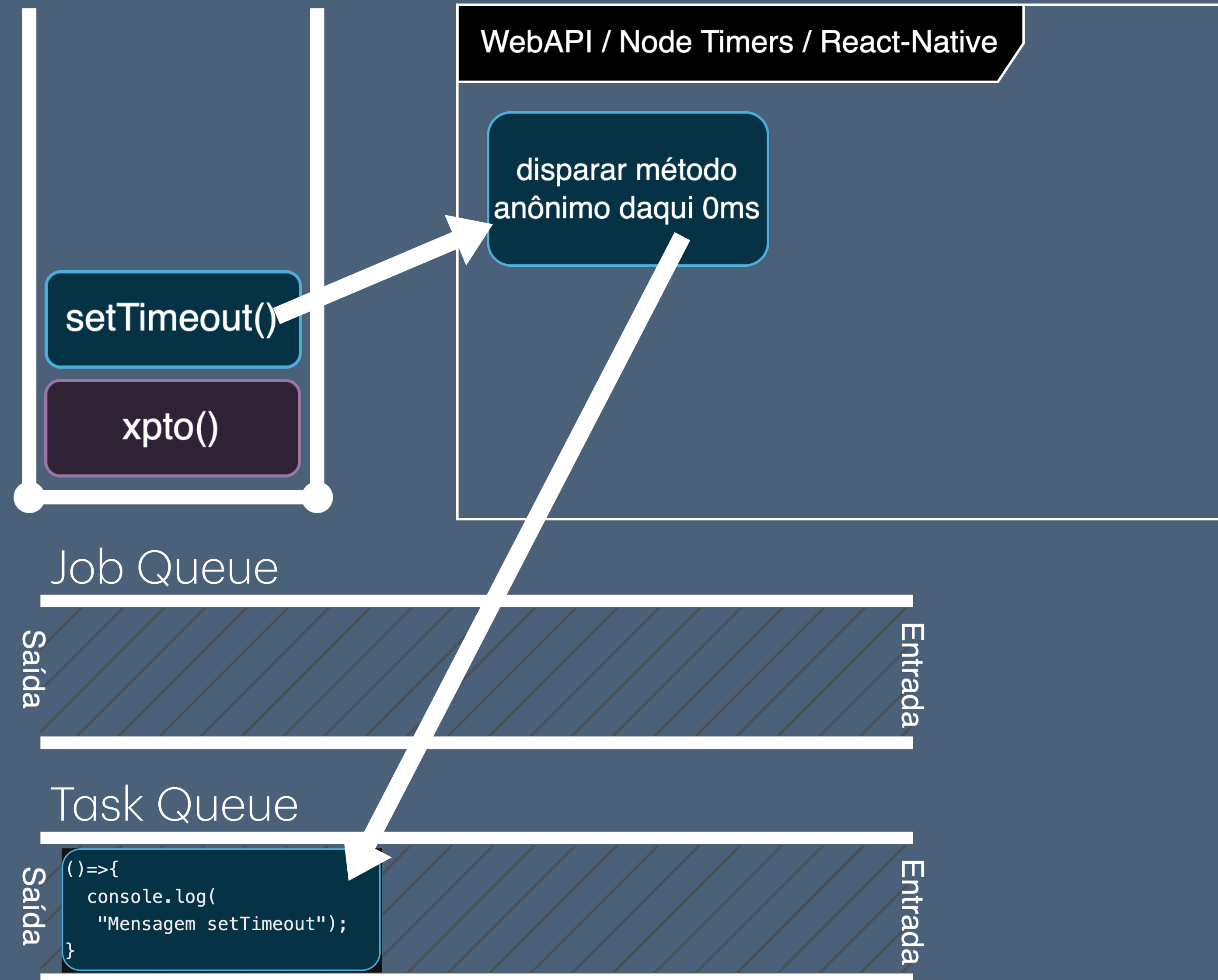
Task Queue



WebAPI / Node Timers / React-Native

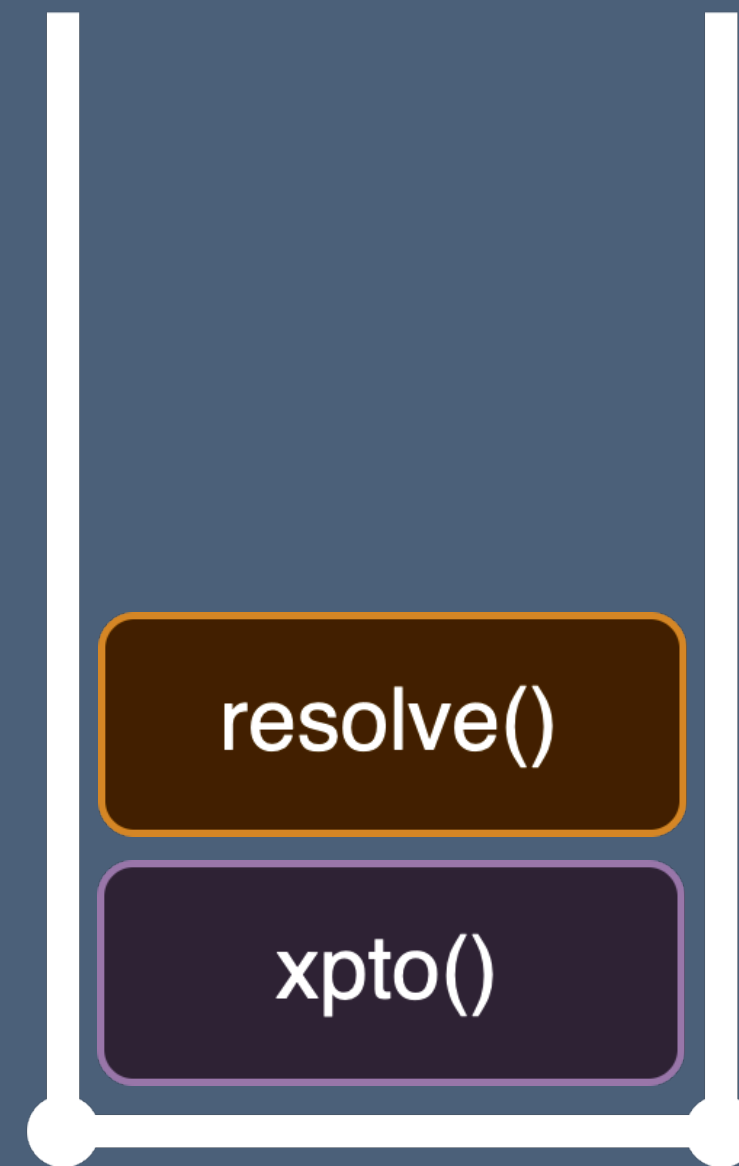
# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```



# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```



Job Queue

Saída

Entrada

Task Queue

Saída

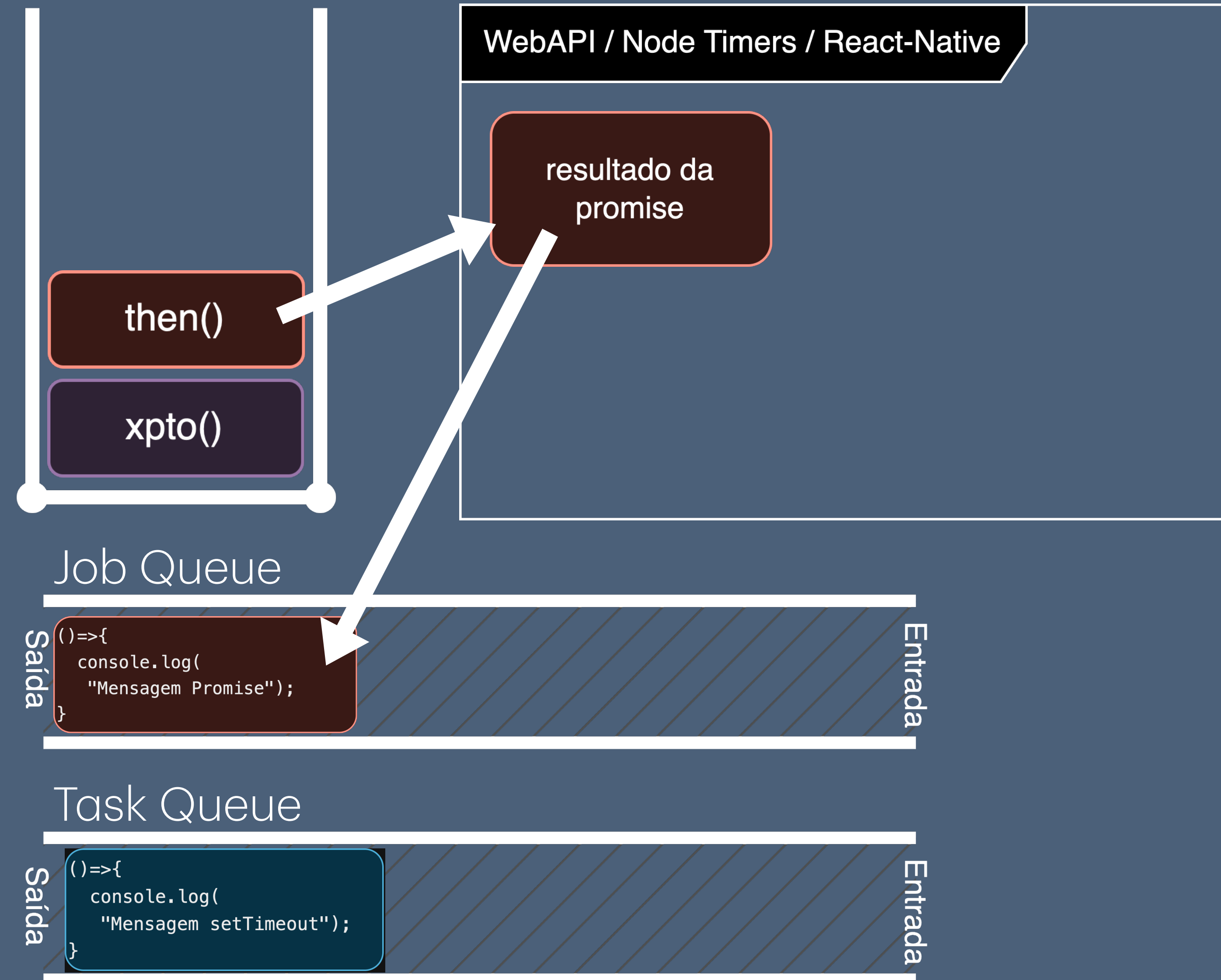
Entrada

```
()=>{  
  console.log(  
    "Mensagem setTimeout");  
}
```

WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

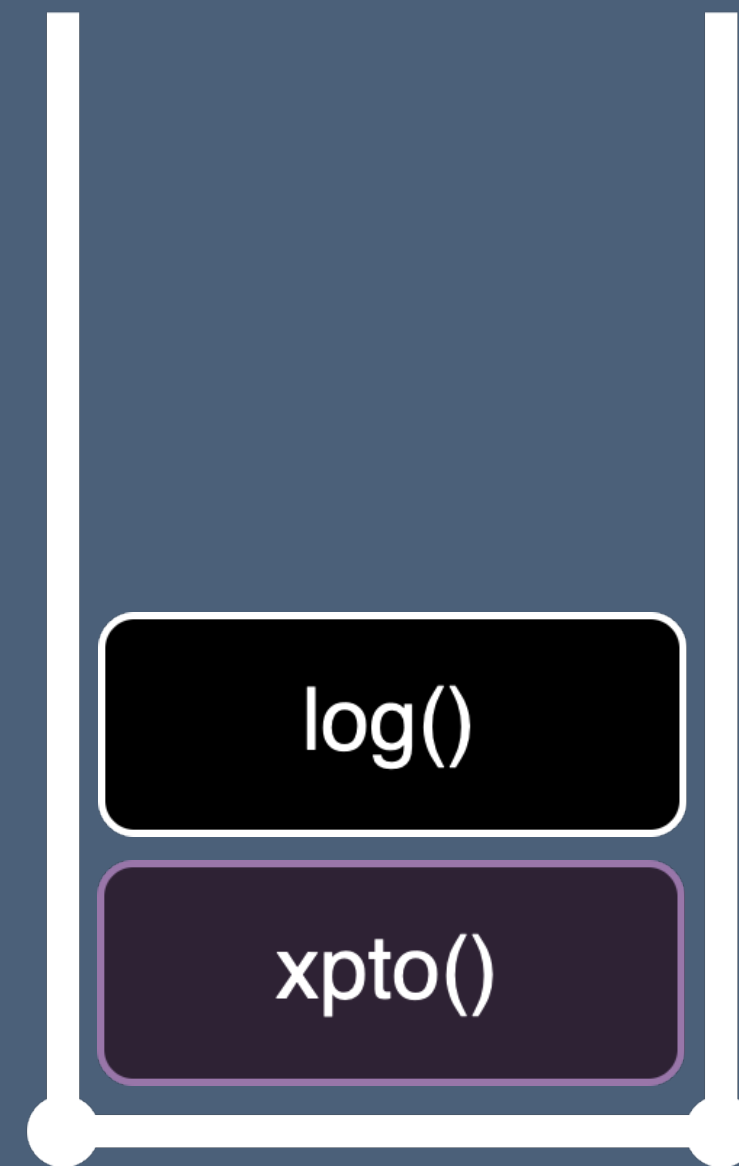
```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```





# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```



Job Queue



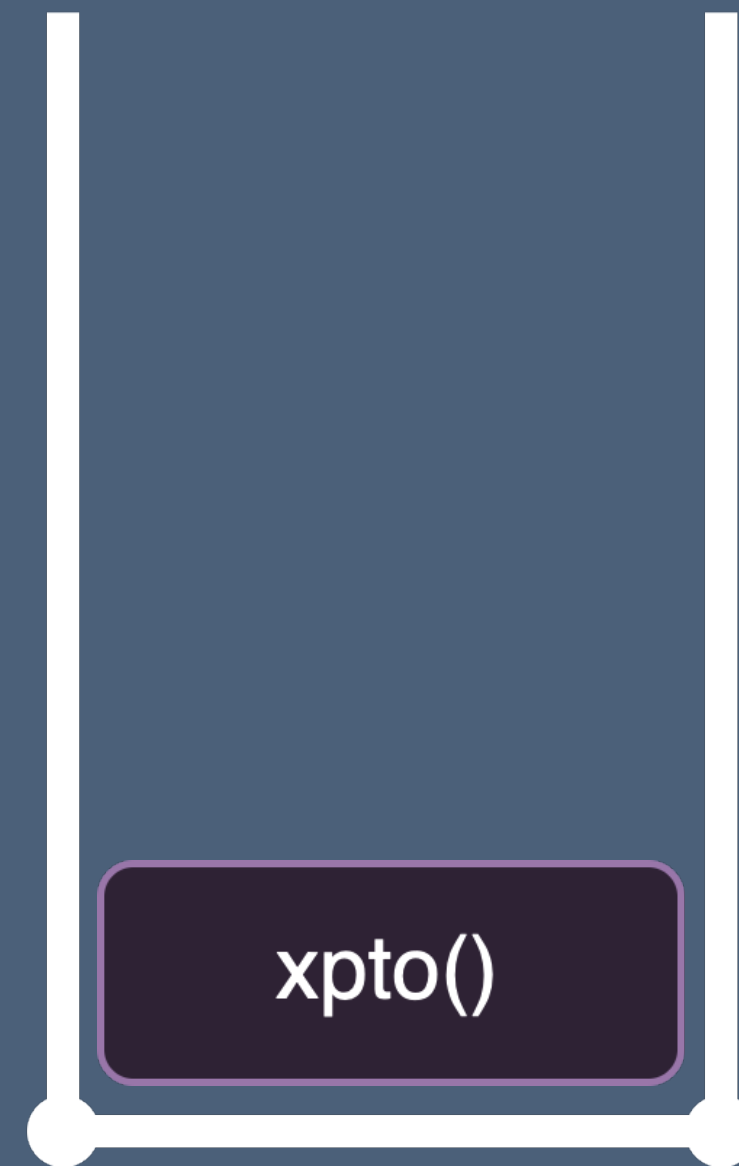
Task Queue



WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
xpto();
```



Job Queue



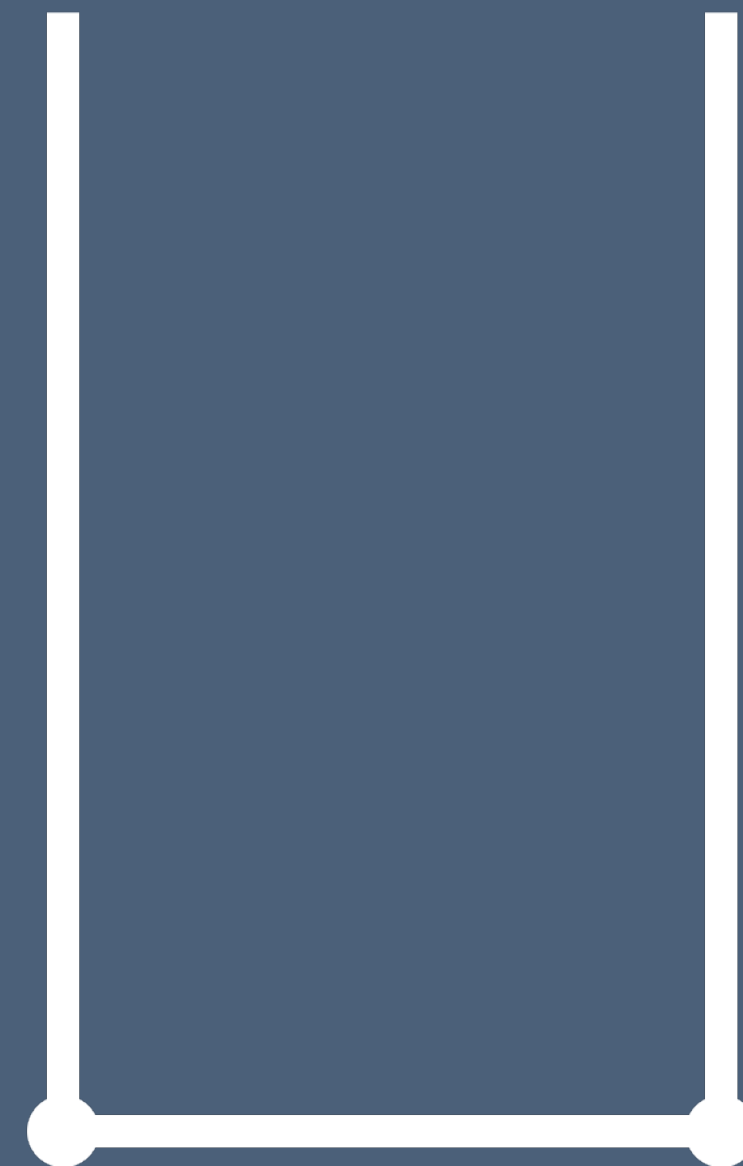
Task Queue



WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
function xpto() {  
  console.log("Começo XPT0");  
  setTimeout(() => {  
    console.log("Mensagem setTimeout");  
  }, 0);  
  Promise.resolve(true).then(() => {  
    console.log("Mensagem Promise");  
  });  
  console.log("Final XPT0");  
}  
  
xpto();
```



WebAPI / Node Timers / React-Native

Job Queue

Saída

```
()=>{  
  console.log(  
    "Mensagem Promise";  
  }  
}
```

Entrada

Task Queue

Saída


```
()=>{  
  console.log(  
    "Mensagem setTimeout";  
  }  
}
```

Entrada

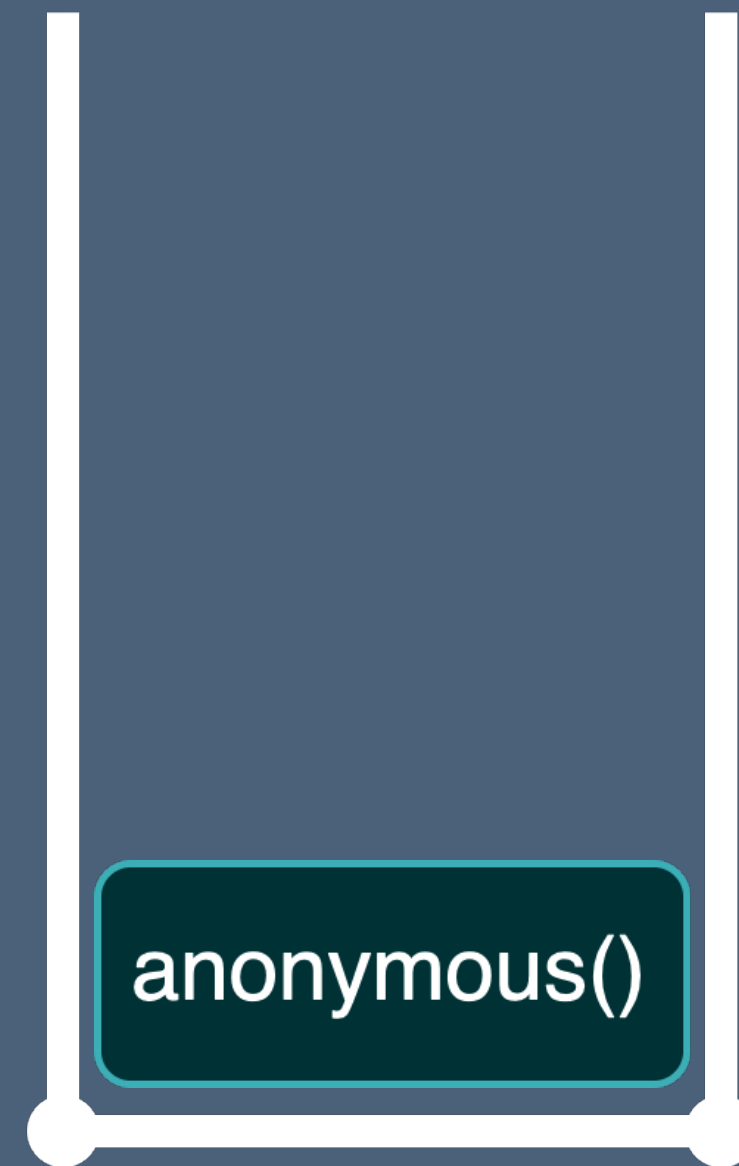


Primeiro o Event Loop checa a  
Job Queue

# Execução de métodos assíncronos



```
() => {  
  console.log("Mensagem Promise");  
}
```



Job Queue

Saída

Entrada

Task Queue

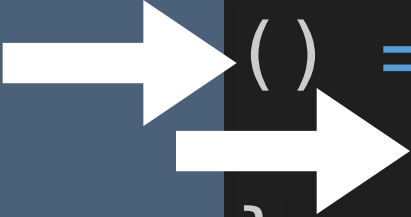
Saída

Entrada

```
()=>{  
  console.log(  
    "Mensagem setTimeout");  
}
```

WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
() => {  
  console.log("Mensagem Promise");  
}
```



Job Queue

Saída

Entrada

Task Queue


Saída

Entrada

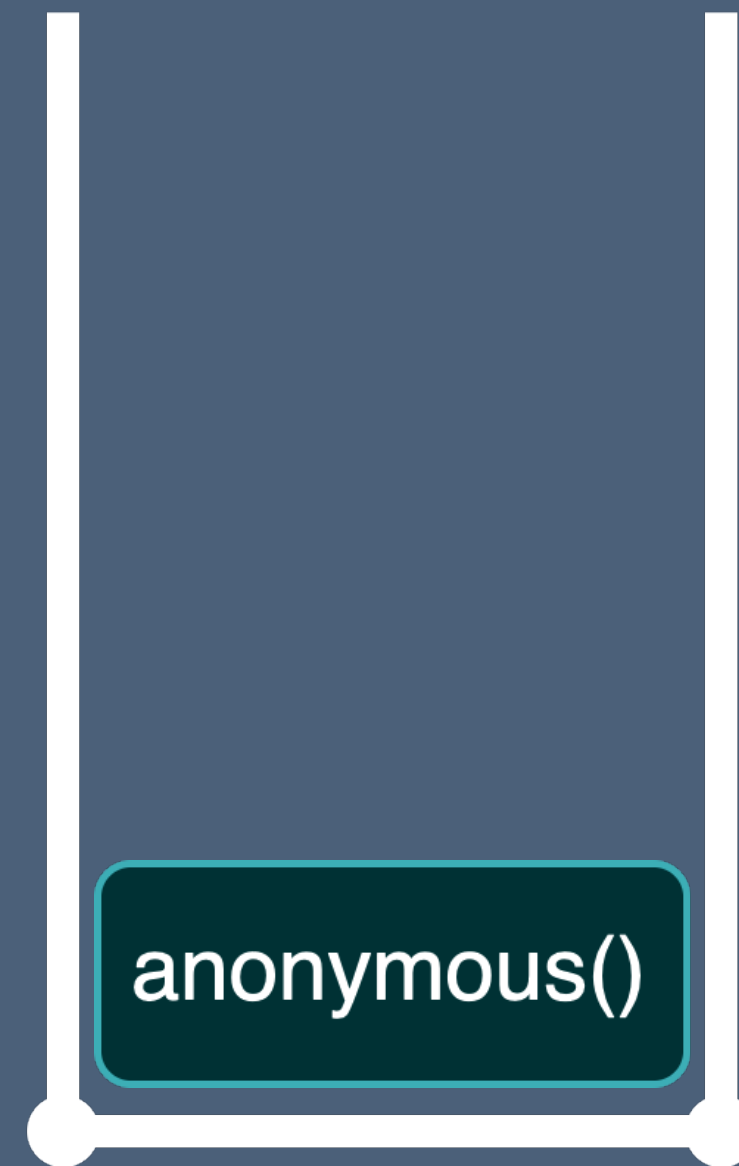
```
()=>{  
  console.log(  
    "Mensagem setTimeout");  
}
```

WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos



```
() => {  
  console.log("Mensagem Promise");  
}
```



Job Queue



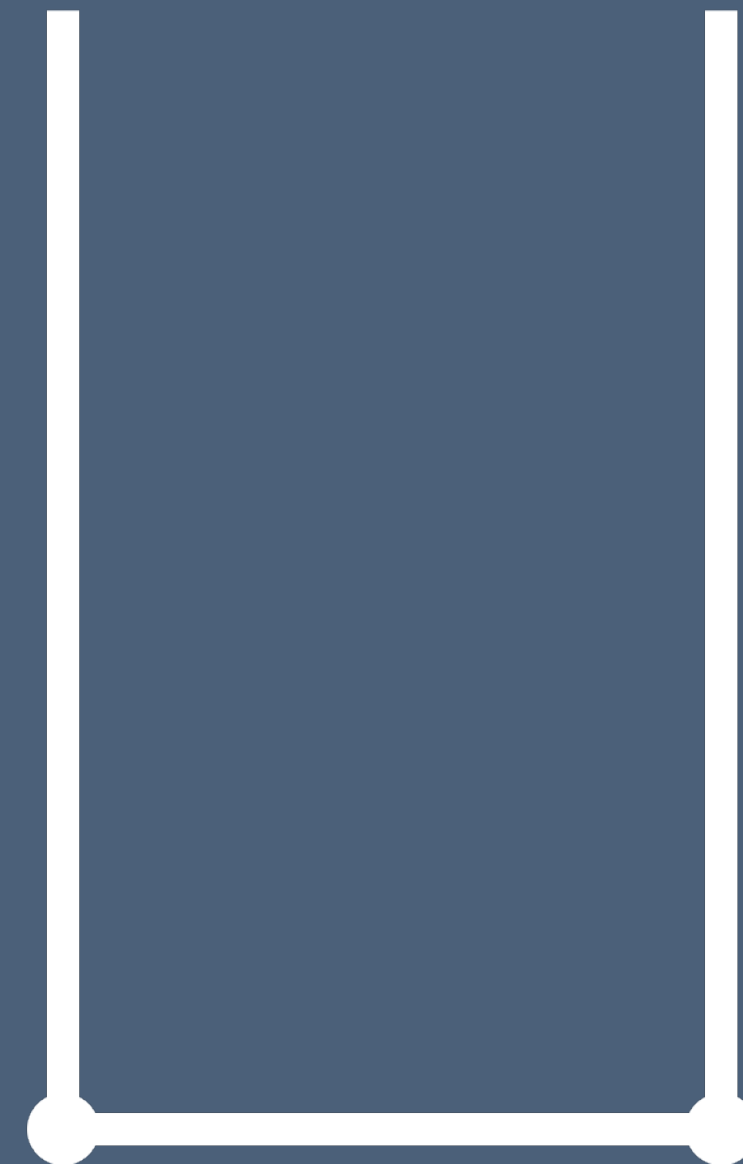
Task Queue



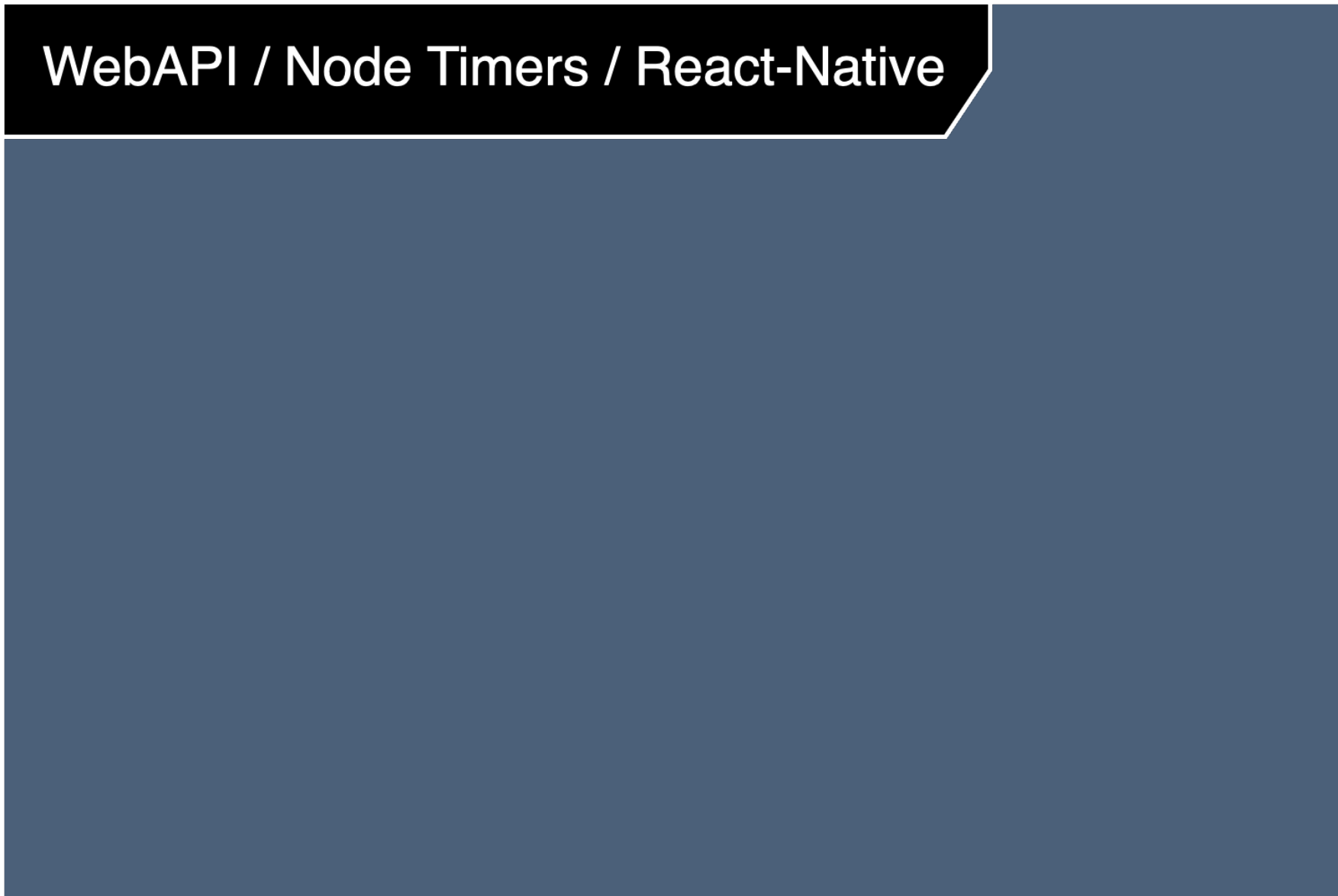
WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
() => {  
  console.log("Mensagem Promise");  
}
```



Job Queue



Saída

Entrada

Task Queue

Saída

Entrada

```
()=>{  
  console.log(  
    "Mensagem setTimeout");  
}
```

A Job Queue está vazia, agora  
pode checar a Task Queue

# Execução de métodos assíncronos



```
() => {  
  console.log("Mensagem setTimeout");  
}
```



Job Queue





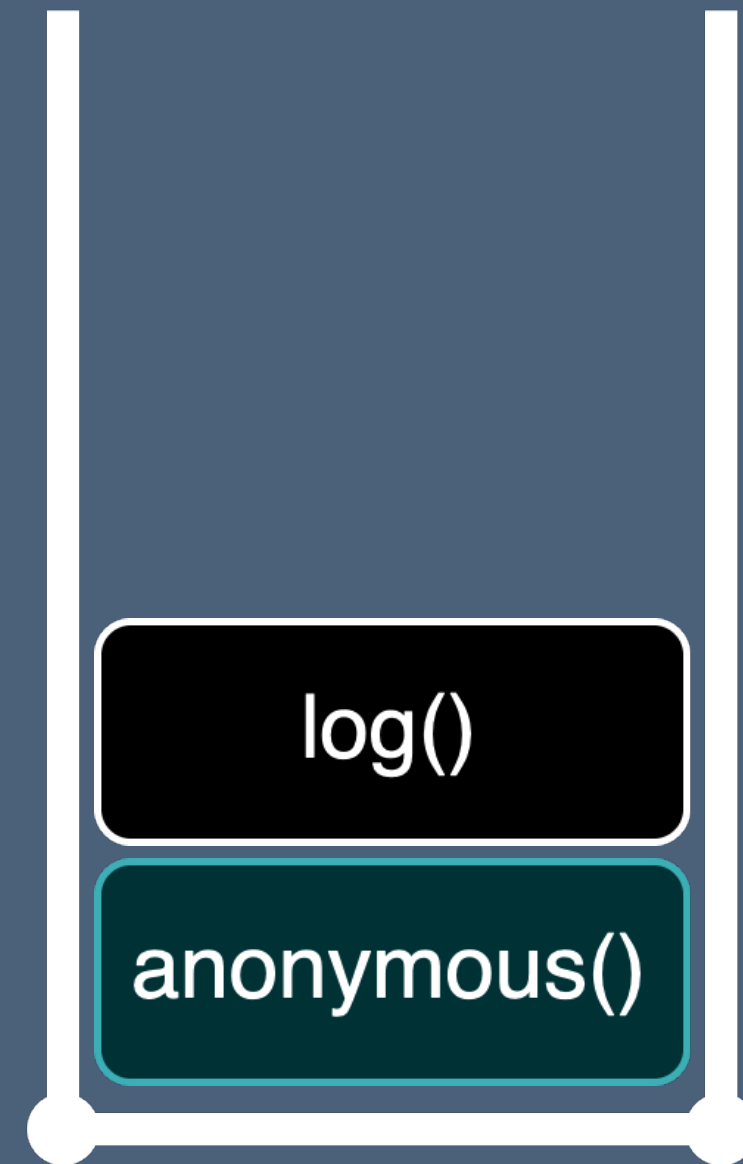
Task Queue



WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
 () => {  
 console.log("Mensagem setTimeout");  
}
```



Job Queue



Task Queue



WebAPI / Node Timers / React-Native



# Execução de métodos assíncronos



```
() => {  
  console.log("Mensagem setTimeout");  
}
```



Job Queue

Saída

Entrada

Task Queue

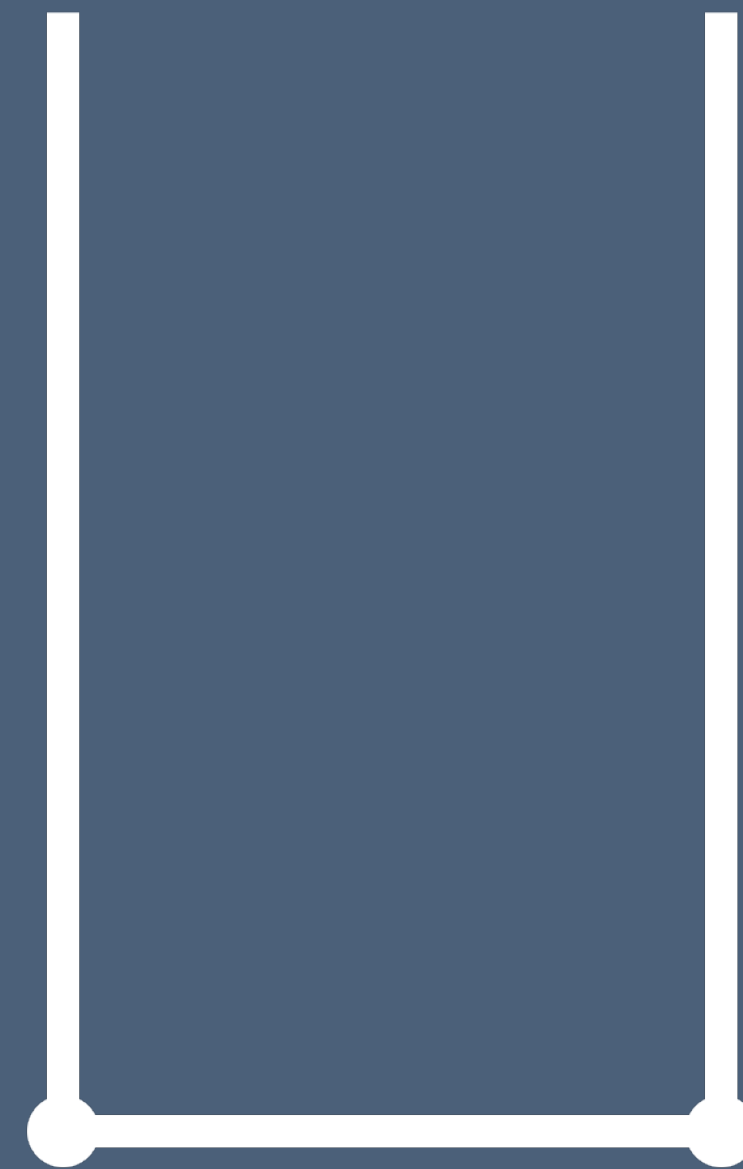
Saída

Entrada

WebAPI / Node Timers / React-Native

# Execução de métodos assíncronos

```
() => {  
  console.log("Mensagem setTimeout");  
}
```



Job Queue



Task Queue



WebAPI / Node Timers / React-Native

# Assíncrono

- Muitas vezes obrigatório
- Operações pesadas que podem ser executadas em background
  - acesso ao disco
  - requisições web
  - parse de JSON
  - Acesso à bridge

Curso Alura de estrutura de dados



<https://unibb.alura.com.br/course/javascript-crud-assincrono>

Mas e o async/await?

async/await são apenas syntax  
sugar de promises

# Async/await vs Promise

```
const URL_TO_FETCH = "https://swapi.dev/api/films/1/";

async function xpto() {
  const resp = await fetch(URL_TO_FETCH);
  const jsonResp = await resp.json();
  console.log(jsonResp.title);
}

xpto();
```

```
const URL_TO_FETCH = "https://swapi.dev/api/films/1/";

function xpto() {
  return fetch(URL_TO_FETCH).then(resp => {
    return resp.json().then((jsonResp => {
      console.log(jsonResp.title);
    }));
  });
}

xpto();
```

Obrigado!

# Referências

- <https://developer.mozilla.org/en-US/docs/Web/API/setTimeout>
- <https://jakearchibald.com/2015/tasks-microtasks-queues-and-schedules/>
- <https://swapi.dev/documentation>