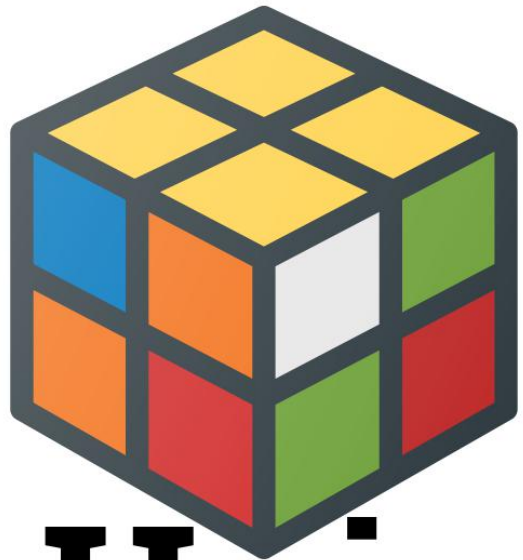


Geek University

Evolua seu lado geek!

www.geekuniversity.com.br

Eventos Multitarefas com Loops de Eventos e Corrotinas

python™

Eventos Multitarefas com Loops de Eventos e Corrotinas

Python Event Loop

É através do Loop de Eventos (Event Loop) que executamos multitarefas assíncronas em Python.

Este recurso foi incorporado em Python 3.4 mas somente no Python 3.5 em diante que se tornou estável e utilizado até hoje através do módulo *asyncio*

Eventos Multitarefas com Loops de Eventos e Corrotinas

Python Event Loop

Fazemos uso de um event loop através de *asyncio.get_event_loop()*

Este método retorna um objeto do tipo *AbstractEventLoop* que como o próprio nome indica é um objeto abstrato.

Através deste objeto, podemos executar o método *obj.run_forever()* ou *obj.run_until_complete(future)**

Ao executar o *obj.run_forever() ou obj.run_until_complete(future)* podemos parar a execução com *obj.stop()* e fechar o evento com *obj.close()*

* Um ‘future’ nada mais é do que uma função assíncrona que pode, ou não, devolver um resultado futuro, ou seja, quando for completada.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Multitarefas Cooperativas

Em Python, ao executar uma função (tarefa) ela mesmo se autosuspende para permitir que outras tarefas (funções) possam ser executadas.

Fazendo uso de Event Loop, este pode reativar uma tarefa (função) quando um evento de IO (Input/Output) é completado.

Isso é chamado de multitarefas cooperativas e é aqui que entram as ***corrotinas***.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Corrotinas

Estudamos corrotinas no nosso curso de Python.

Mas vale lembrar que quando falamos em Corrotinas podemos falar de dois objetos principais:

- Função Corrotina;
- Objeto Corrotina;

Eventos Multitarefas com Loops de Eventos e Corrotinas

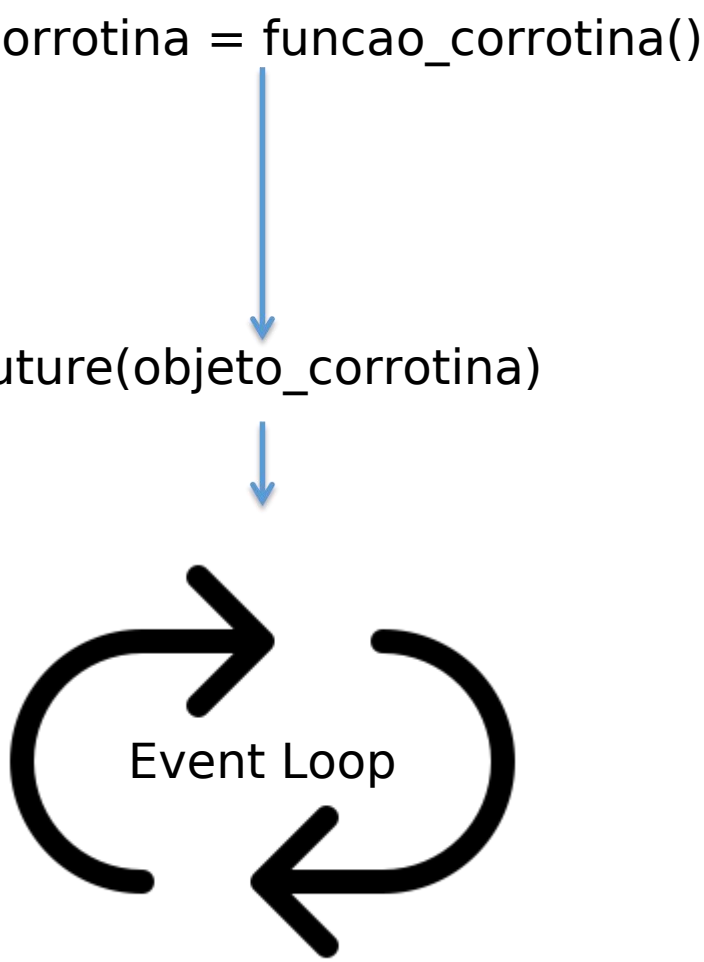
Função Corrotinas

A função corrotina é uma função especial que dá ao seu executor o controle do estado desta função, ou seja, a função pode executar, pausar, reexecutar e parar/finalizar.

Diferente da forma que aprendemos a criar corrotinas no curso de Python, a partir da versão 3.5+ podemos transformar qualquer função comum em corrotina acrescentando a palavra ‘async’ na declaração da função.

A execução de uma função corrotina retorna um objeto corrotina.

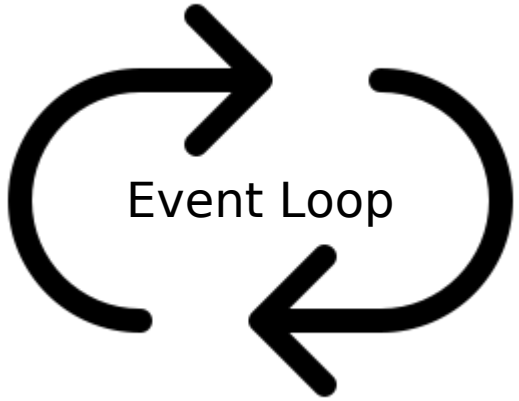
Eventos Multitarefas com Loops de Eventos e Corrotinas



Eventos Multitarefas com Loops de Eventos e Corrotinas

objeto_corrotina = funcao_corrotina()

Future(objeto_corrotina)



Vamos ver isso no código...

Eventos Multitarefas com Loops de Eventos e Corrotinas

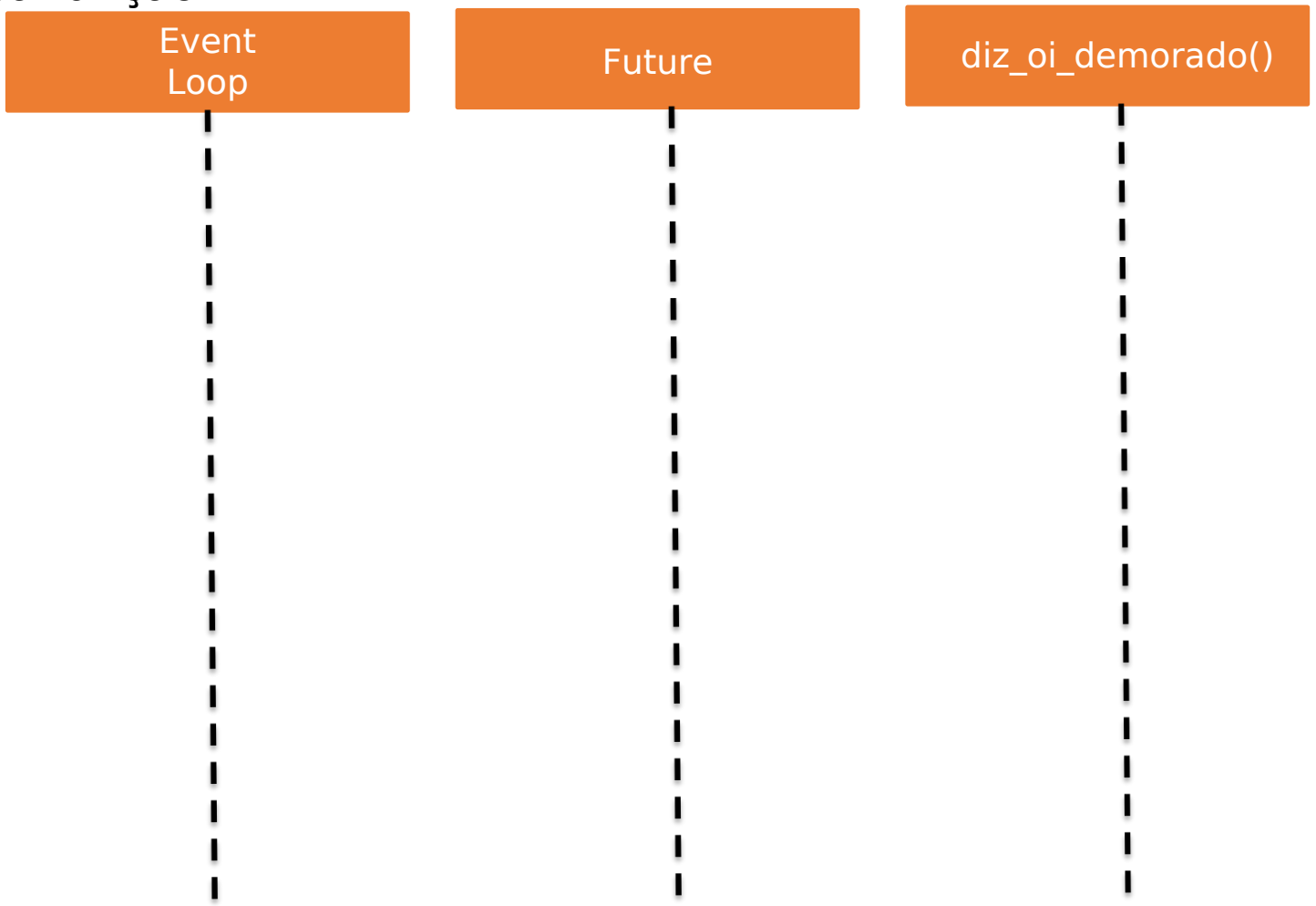
Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')




el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()
```

Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

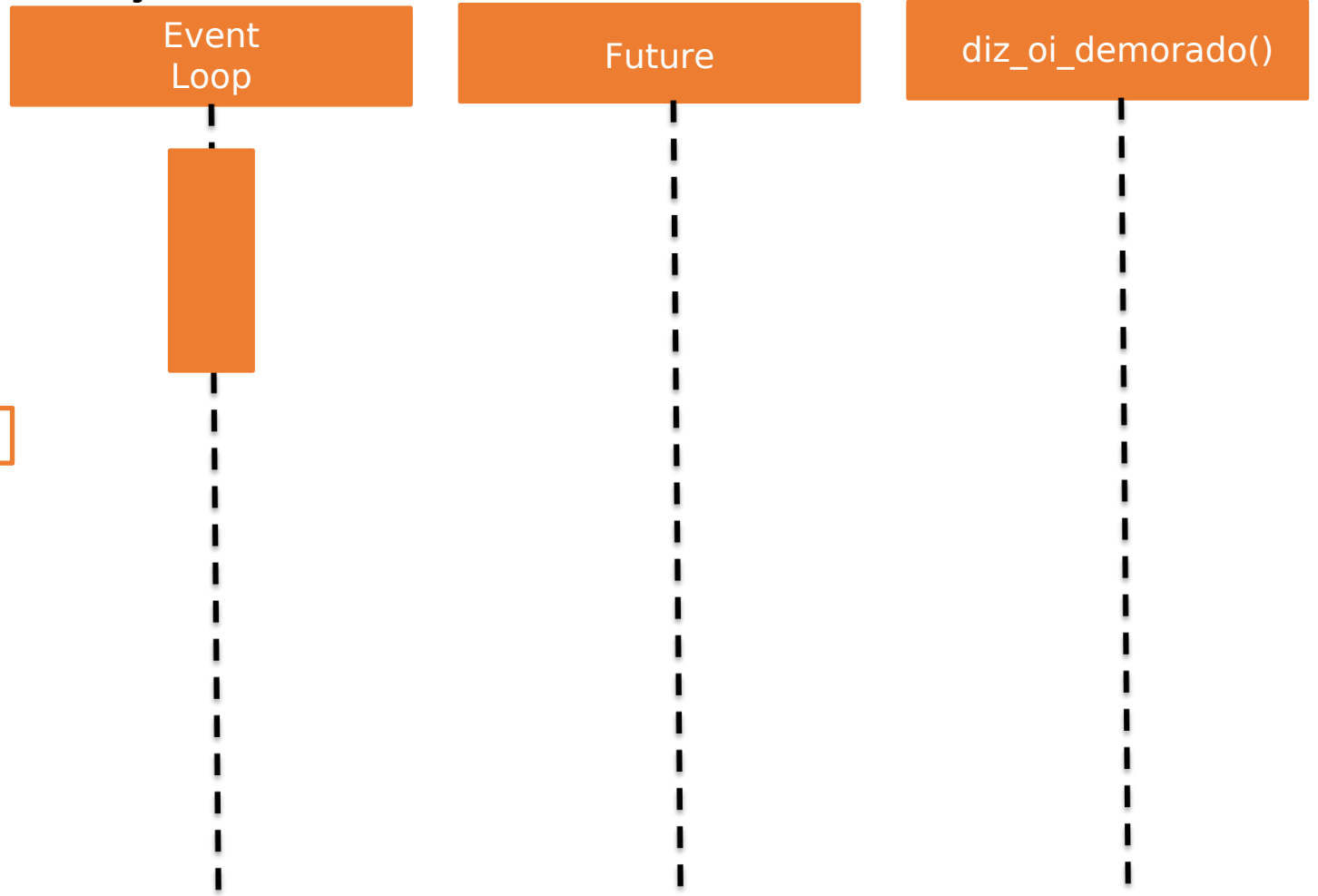
```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()

```



Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```
1  import asyncio
2
3
4  async def diz_oi_demorado():
5      print('Oi...')
6      await asyncio.sleep(2)
7      print('todos...')
8
9
10
11
12 el = asyncio.get_event_loop()
13 el.run_until_complete(diz_oi_demorado())
14 el.close()
15
```

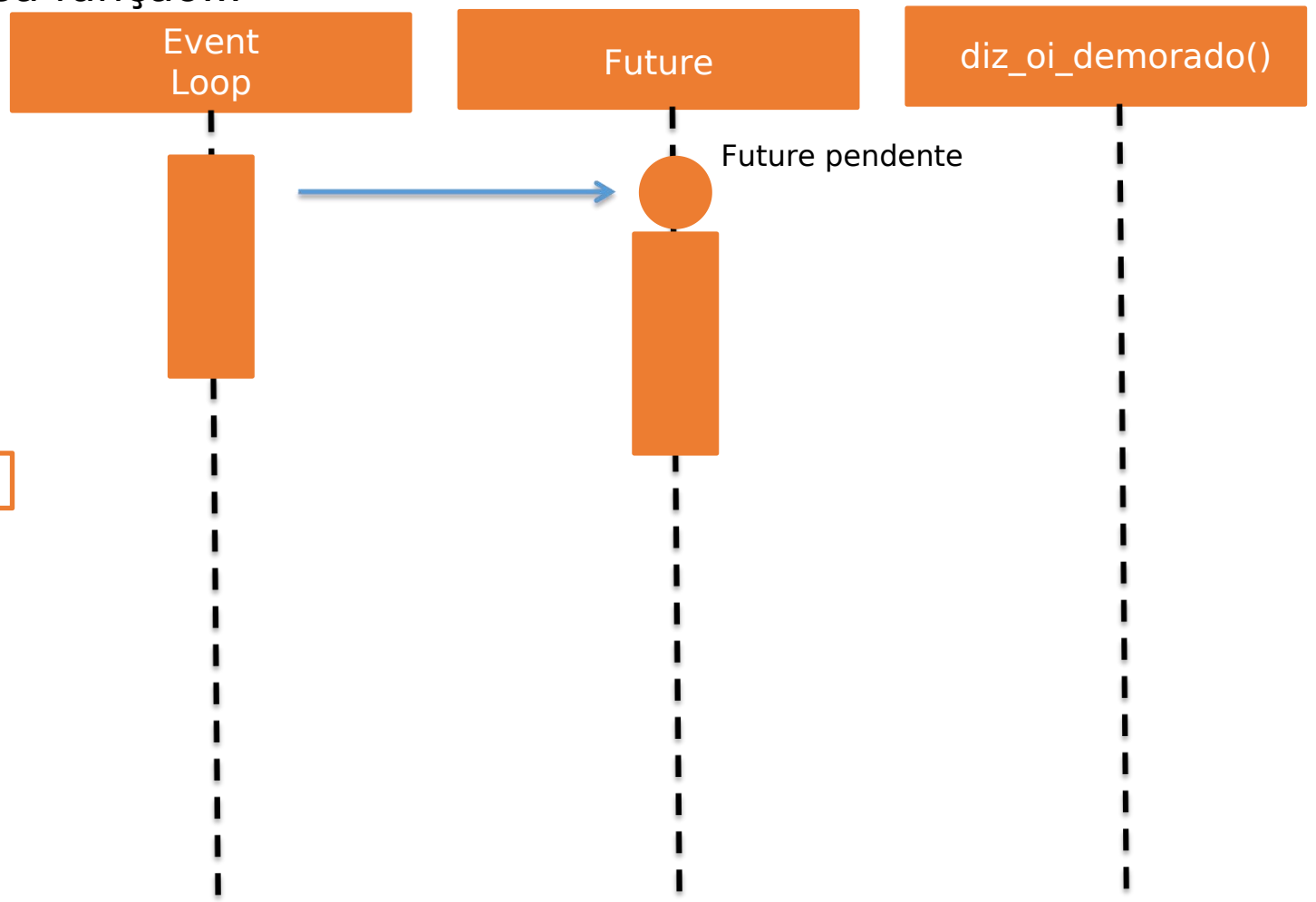


Note que o programa inicia na linha 12, gerando o event loop.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```
1  import asyncio
2
3
4  async def diz_oi_demorado():
5      print('Oi...')
6      await asyncio.sleep(2)
7      print('todos...')
8
9
10
11
12 el = asyncio.get_event_loop()
13 el.run_until_complete(diz_oi_demorado())
14 el.close()
15
```

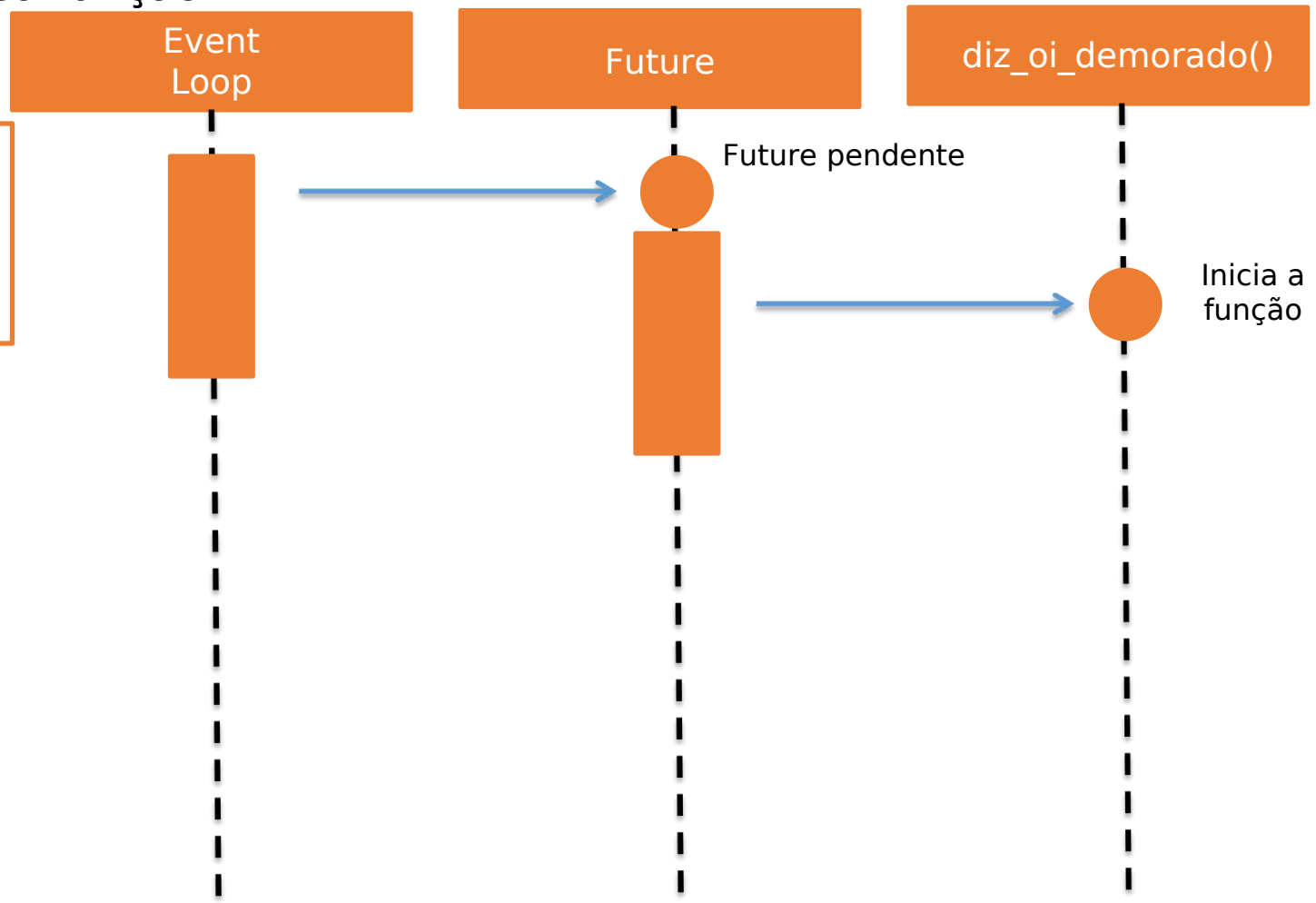


Então na linha 13 o método run_until_complete passa a função para ser executada e um Future fica pendente.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()

```

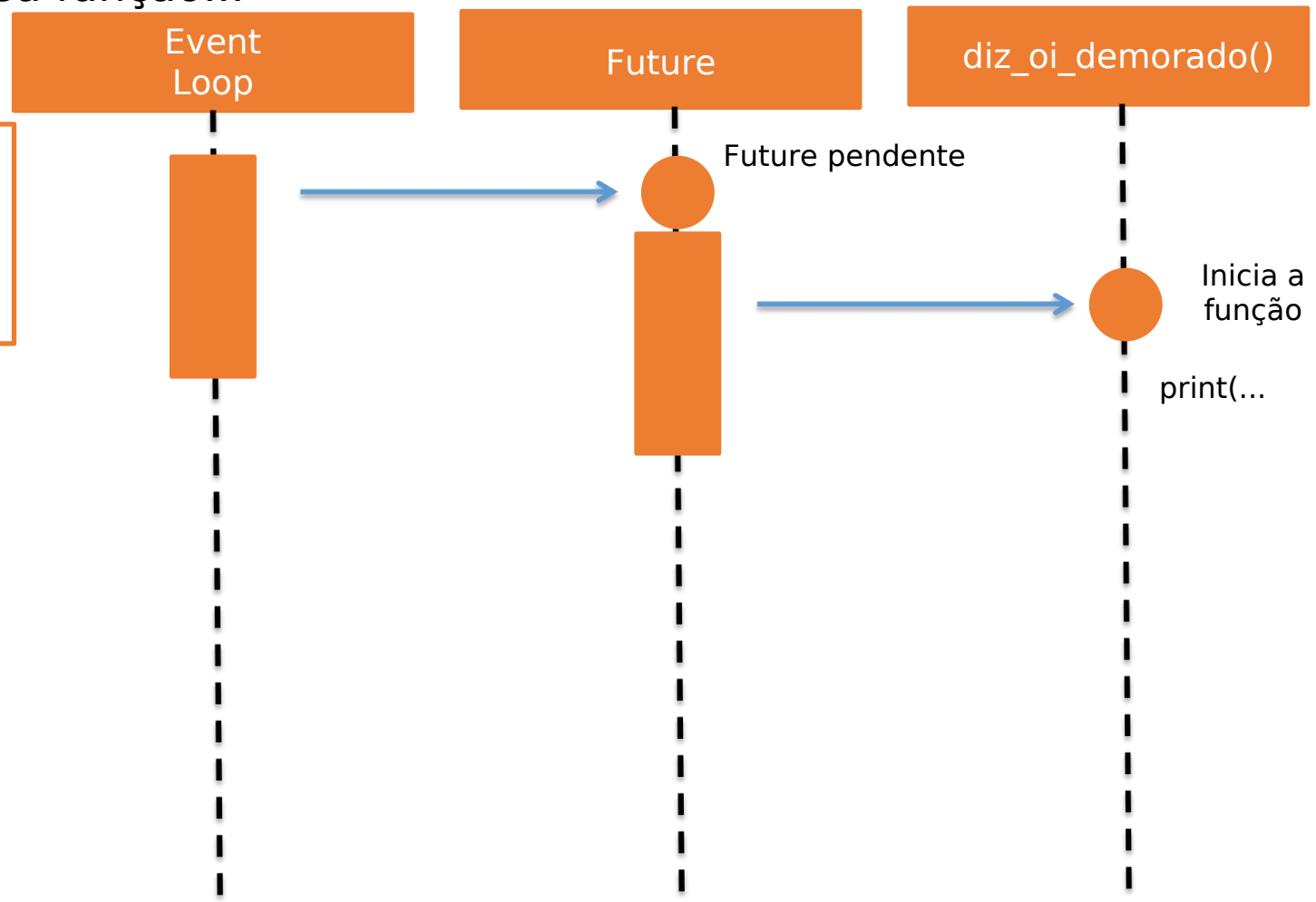


A função diz_oi_demorado() então é executada.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()

```

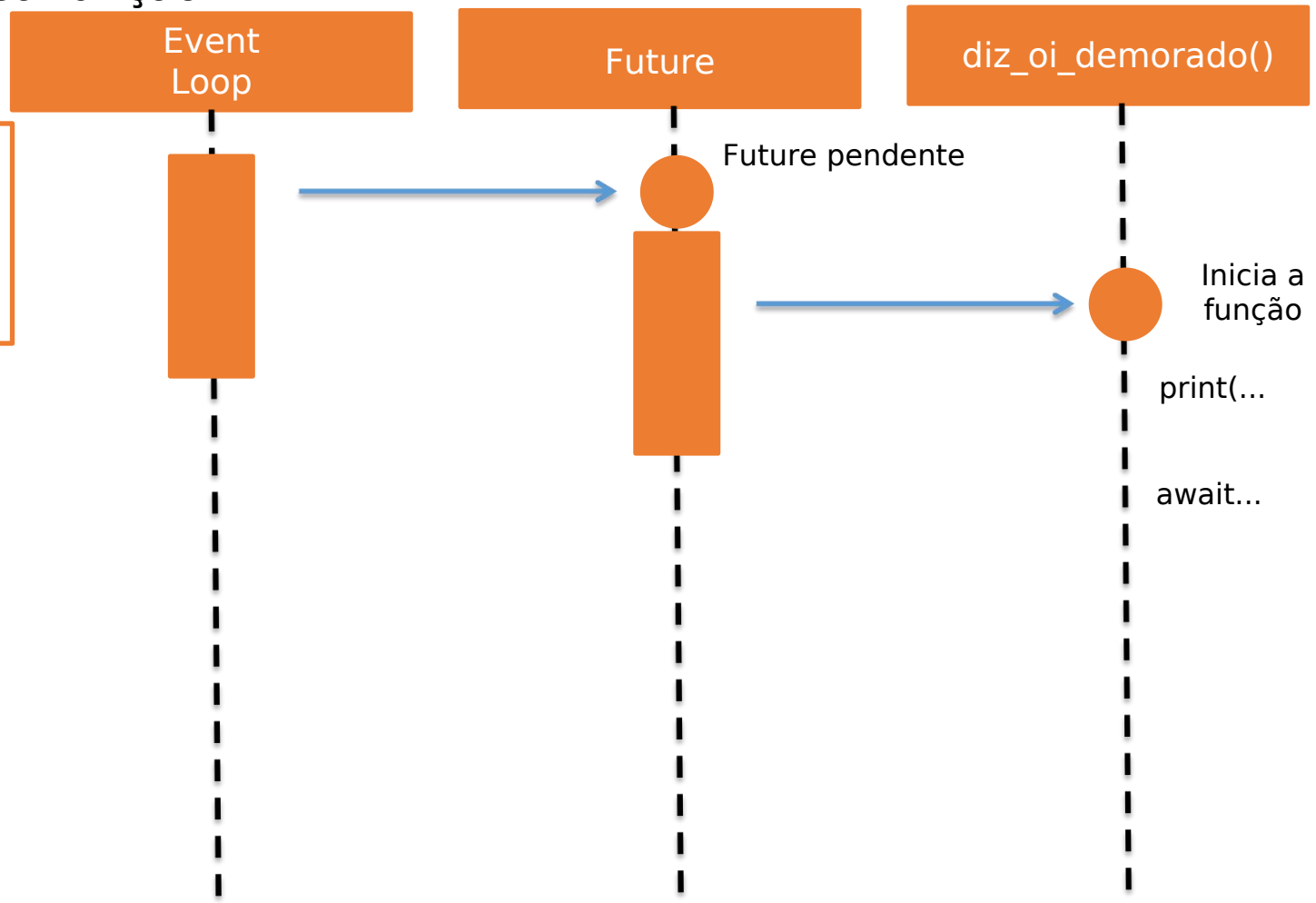


A função diz_oi_demorado() então é executada.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()

```

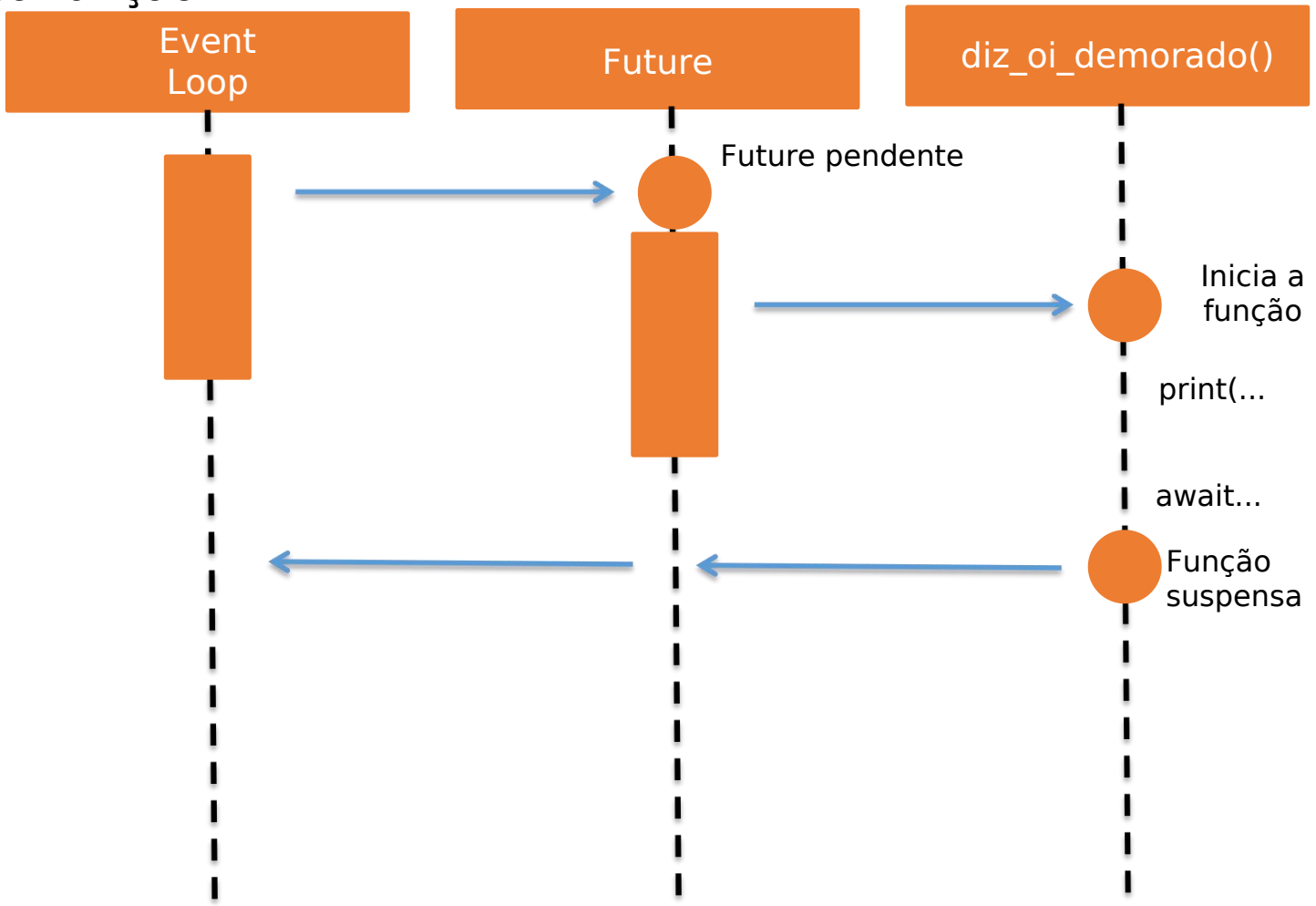


A função diz_oi_demorado() então é executada.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()
```



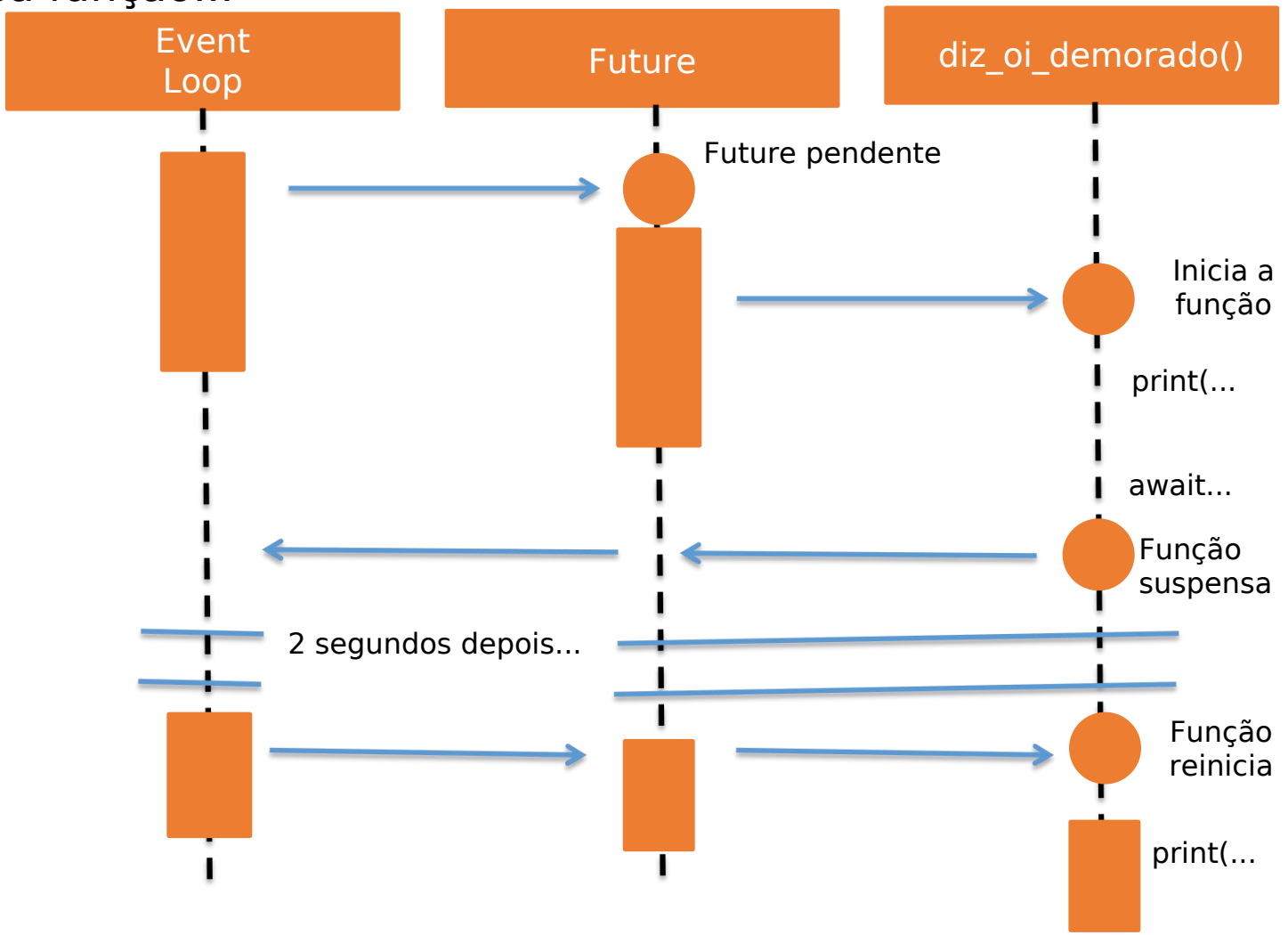
Ao chegar no await, a execução da função diz_oi_demorado() é suspensa e o controle de execução volta para o Event Loop.

Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()
```

Após os 2 segundos, o Event Loop reinicia a função que estava suspensa e o restante da execução é realizada.

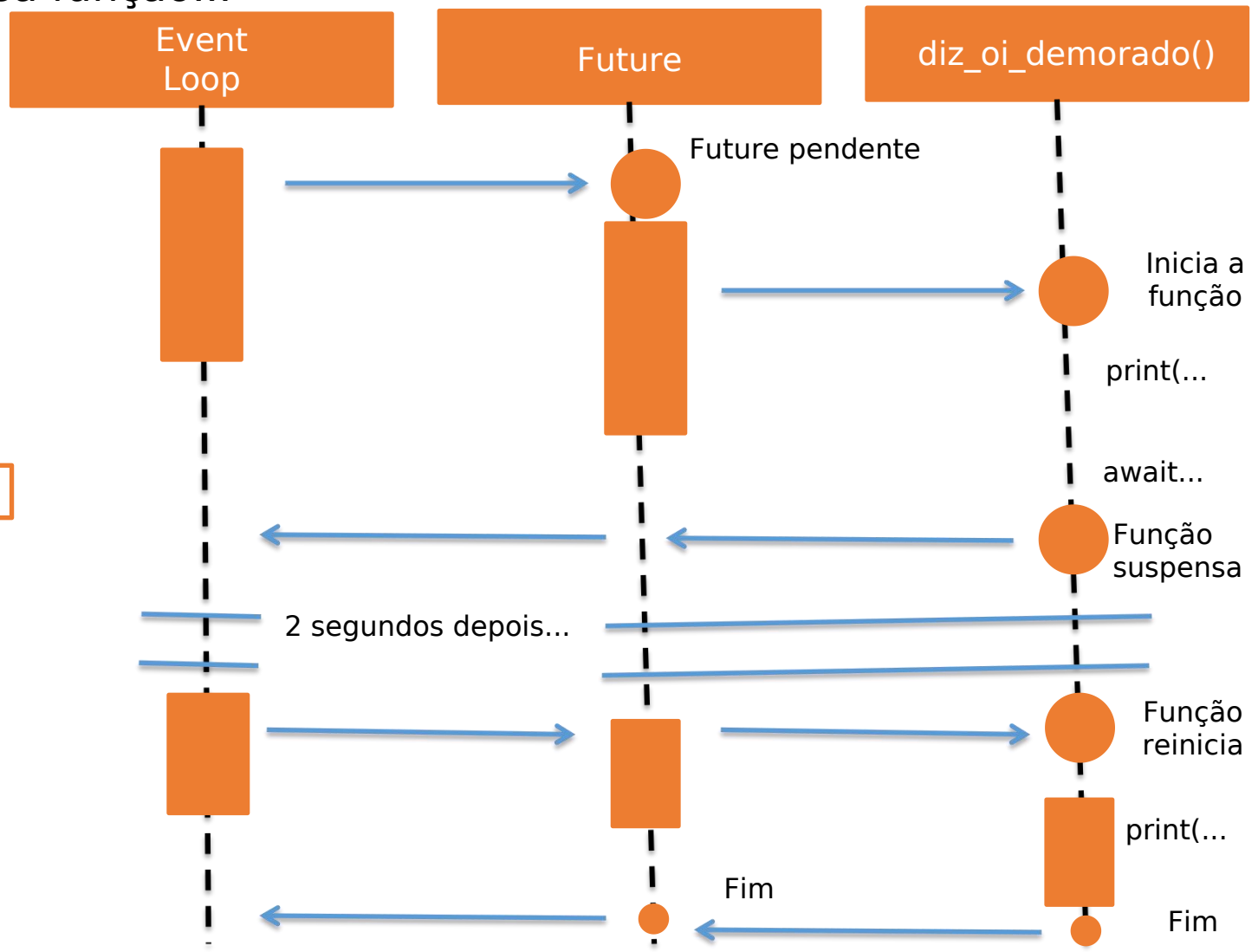


Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()
```

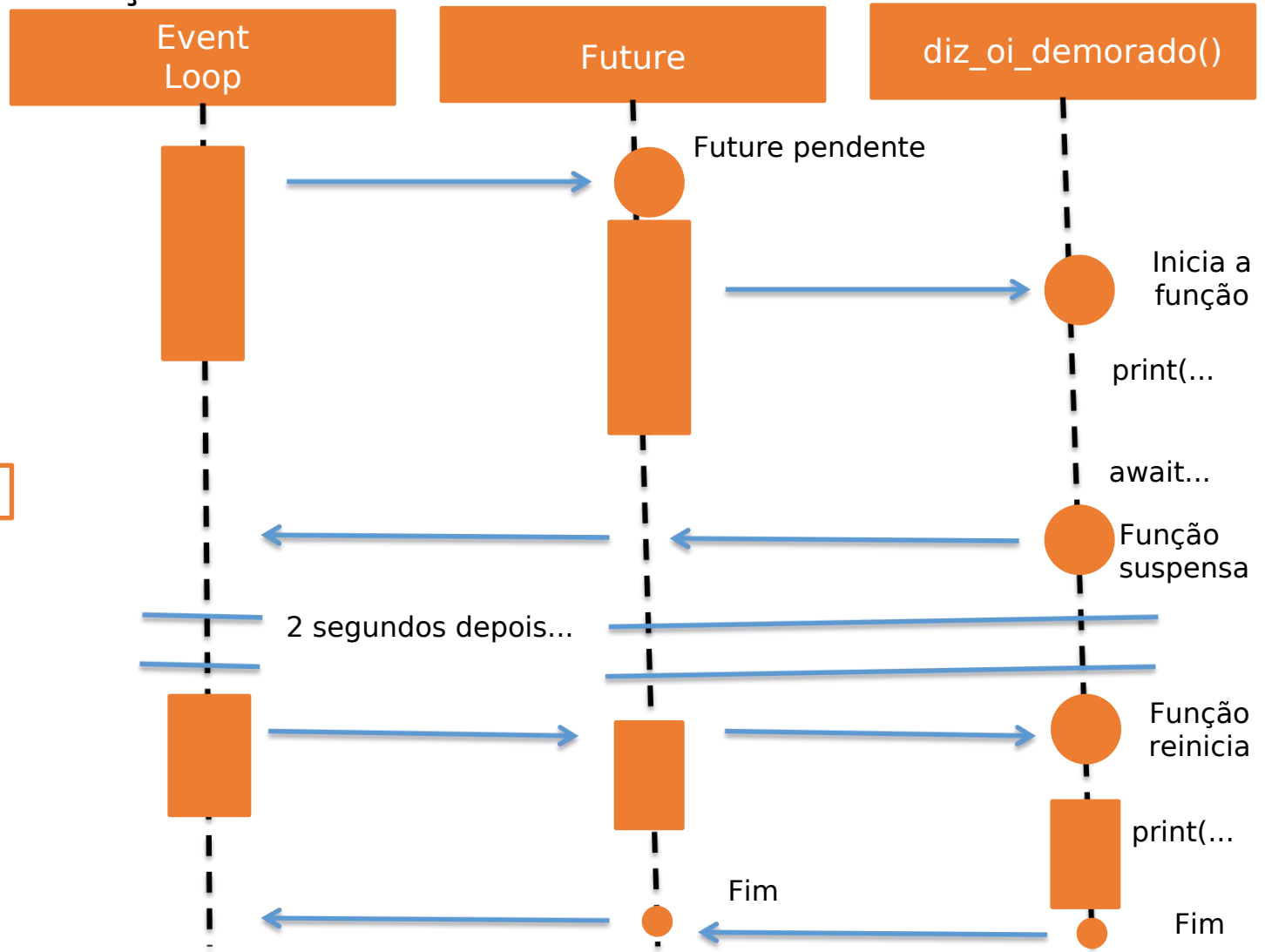
Por fim, a função diz_oi_finaliza(), a Future finaliza e o Event Loop é encerrado.



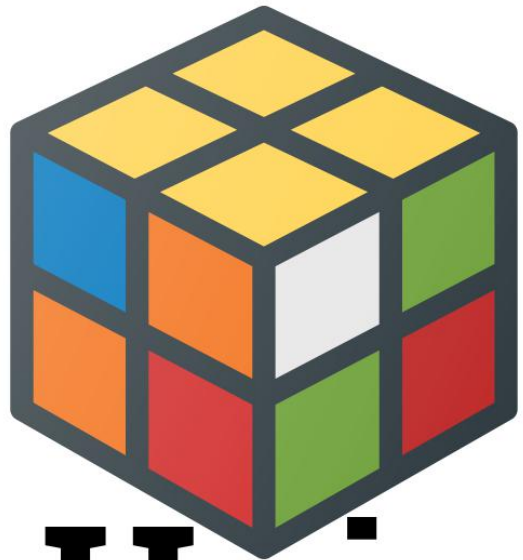
Eventos Multitarefas com Loops de Eventos e Corrotinas

Vamos analisar o fluxo de execução da nossa função...

```python
import asyncio


async def diz_oi_demorado():
    print('Oi...')
    await asyncio.sleep(2)
    print('todos...')



el = asyncio.get_event_loop()
el.run_until_complete(diz_oi_demorado())
el.close()

```



Na próxima aula iremos estudar um pouco mais sobre o Asyncio...



Geek University

Evolua seu lado geek!

www.geekuniversity.com.br