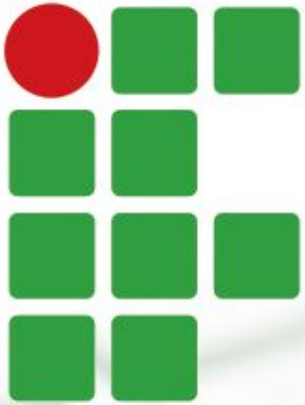


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sistemas Distribuídos

- Threading -

Concorrência

- Mecanismos para implementação de **Concorrência**

Corrotinas

Libs de alto nível fornecidas pela linguagem de programação

Threads

Processos

Mecanismos nativos do Sistema Operacional

(Independe de linguagem)

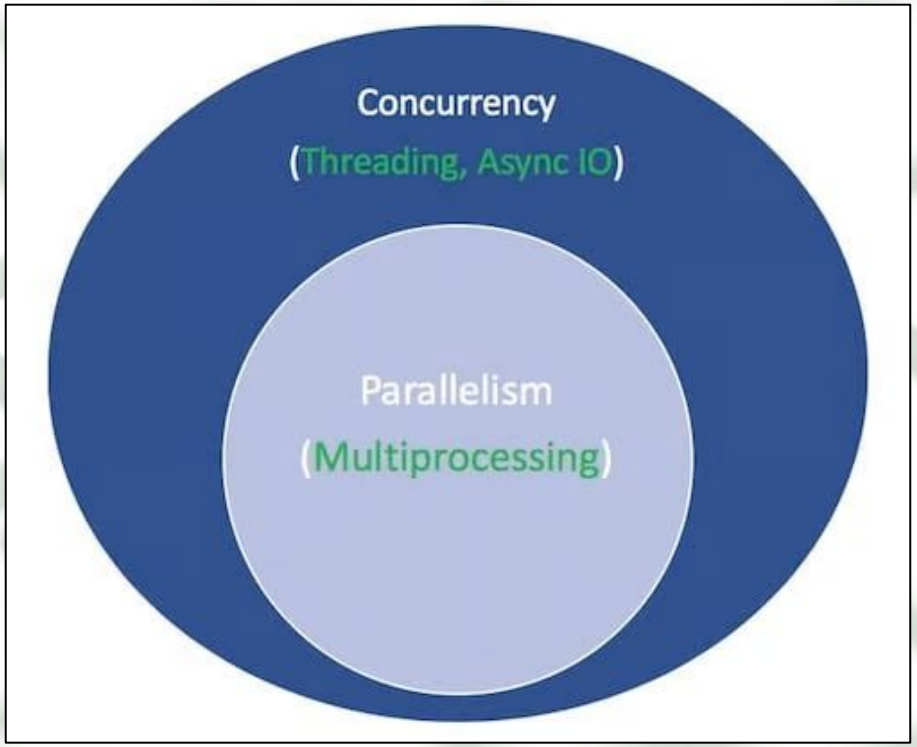
Concorrência

- Em Python…

Corrotinas
asyncio, trio, curio, …

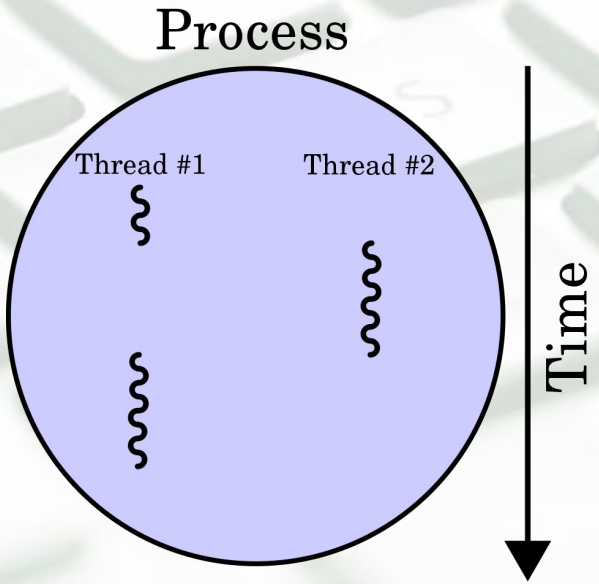
Threads
threading

Processos
multiprocessing



Threads

- **Objetivo: separar o conceito de Processo em execução do conceito de Linha de Execução.**
- Imagine as ***threads*** como **linhas de execução distintas e concorrentes** dentro de **um mesmo processo**.

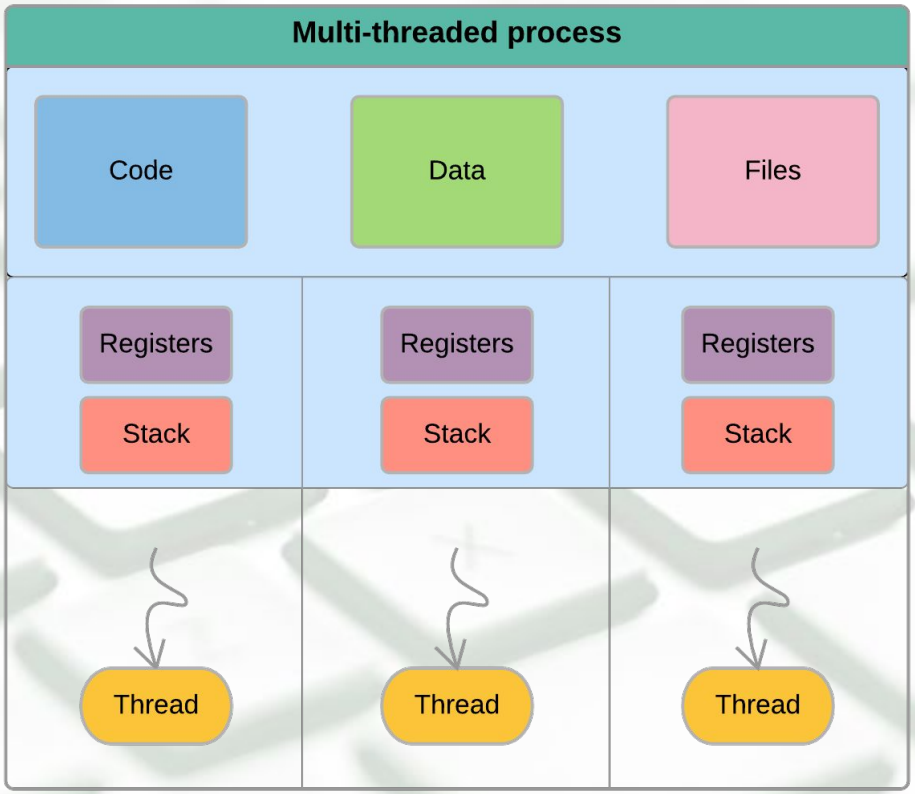


Threads

- Assim como em um processo, uma **thread** executa uma porção bem definida de código, e também **independente** de outras threads.
- Entretanto, no caso de *threads,* o S.O. não se preocupa com alto grau de transparência na concorrência entre essas threads.
- As **threads** passam a ser a unidade de escalonamento no contexto de um processo.
- Cada CPU (core) executa um processo por vez, e dentro deste contexto, **uma thread por vez**.

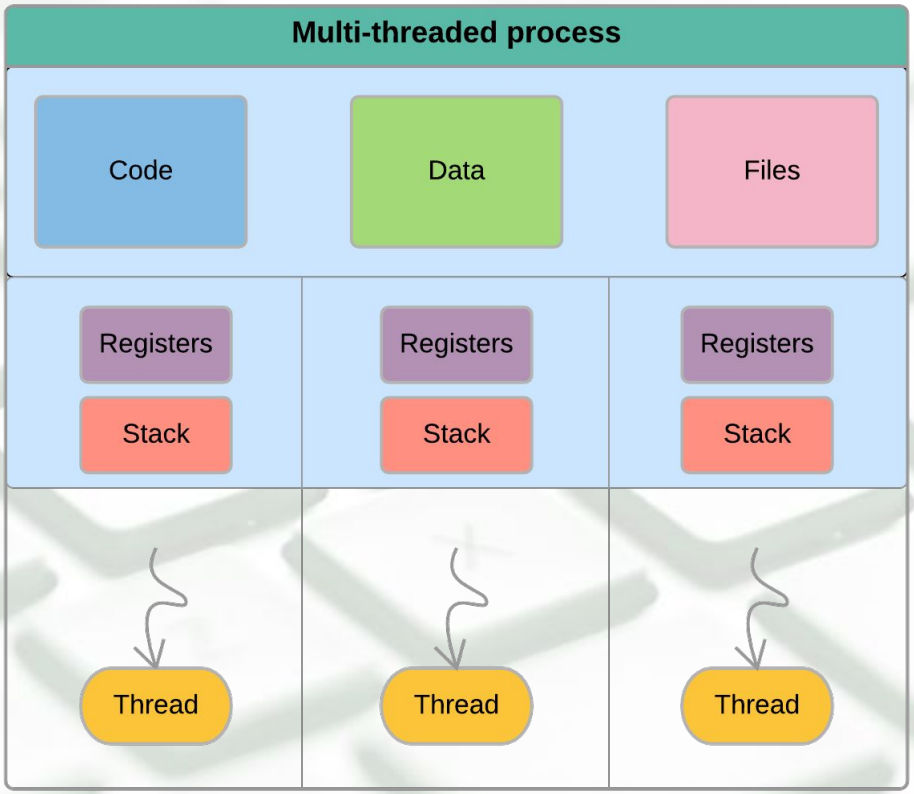
Threads

- Cada **processo** contém, no mínimo, uma thread (*Main Thread*).
- Processos podem porém, implementar um conjunto de outras **threads** (controles de fluxos distintos) que serão executadas **concorrentemente** no mesmo escopo do processo.



Threads

- Perceba que as *threads* **compartilham código, arquivos e dados (variáveis)** do processo como um todo.



- É algo POSITIVO:

  Economia e Praticidade

- E NEGATIVO:

  Segurança e Sincronização

NEGATIVO???

- Imagine que um sistema bancário utiliza *threads* para atender as requisições de movimentação financeira de uma conta-corrente.

- Em que aspecto isso se tornaria um grande problema?

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Acompanhe...

Saldo Atual da Conta: R$ 1.000	
Thread A	**Thread B**
Consulta Saldo da Conta	
	Consulta Saldo da Conta
Realiza Saque de R$ 1.000,00	
	Paga conta de R$ 500,00
Saldo Atual da Conta: R$???	

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Acompanhe...

Saldo Atual da Conta: R$ 1.000	
Thread A	**Thread B**
Consulta Saldo da Conta	
	Consulta Saldo da Conta
Realiza Saque de R$ 1.000,00	
	Paga conta de R$ 500,00
Saldo Atual da Conta: R$???	

Vocês já devem ter estudado o conceito de **ACID** em outras disciplinas...

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Acompanhe...

Saldo Atual da Conta: R$ 1.000	
Thread A	**Thread B**
Consulta Saldo da Conta	
	Consulta Saldo da Conta
Realiza Saque de R$ 1.000,00	
	Paga conta de R$ 500,00
Saldo Atual da Conta: R$???	

Vocês já devem ter estudado o conceito de **ACID** em outras disciplinas...
ATOMICIDADE, CONSISTÊNCIA, ISOLAMENTO E DURABILIDADE

Threads

- Contudo, as ***threads*** são muito importantes para a grande maioria das aplicações e **essenciais** para o desenvolvimento de Sistemas Distribuídos…

- **Vantagens:**
  - Facilidade para desenvolvimento de concorrência.
  - Sobreposição de operações de I/O e computação.
  - Melhor aproveitamento da CPU.
  - …

Implementação de Threads

- Existem duas formas de implementação de **Threads**
  - Kernel Level Threads
  - User Level Threads (*light-weight threads*)

Kernel Level Threads

- **Kernel Level Threads**
  - Também chamado de *Lightweight Process* (LWP)
- Criadas através de *System Calls* do próprio S.O.
- Neste modelo, **as threads são reconhecidas pelo Sistema Operacional**.
- O S.O. faz escalonamento das threads (e não dos processos vinculados)
- Consegue resolver problema de justiça entre processos:
  - P.Ex.: P1 tem 2 threads e P2 tem 10 threads. A CPU possui 2 cores. Qual forma mais justa de escalonar?

User Level Threads

- **User Level Threads** - *Lightweight Threads* (LWT)
  - Implementado através de Bibliotecas específicas.
- Threads são invisíveis ao Sistema Operacional.
- **Criação de threads, troca de contexto e sincronização é feito por chamadas de funções do próprio processo.**

- Por não depender do S.O., são mais leves e rápidas.
- Desvantagem: Escalonamento das threads. (O S.O não possui controle sobre elas)

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Asyncio

slide03.01a_asyncio.py

```
import asyncio
import time

async def task1():
    print("Task1 Iniciada")
    time.sleep(6)
    print("Task1 Concluída")

async def task2():
    print("Task2 Iniciada")
    time.sleep(4)
    print("Task2 Concluída")

async def main():
    print('Aplicação Iniciada')
    inicio = time.time()
    t1 = asyncio.create_task(task1())
    t2 = asyncio.create_task(task2())
    await t1,t2
    print(f'Execução Finalizada em {time.time()-inicio:.2f} segs.')

asyncio.run(main())
```

Lembram-se do problema de chamadas a funções síncronas em funções assíncronas no asyncio?

T == 10 segundos

Threading

slide03.01b_threading.py

```
def task1():
  print("Task1 Iniciada")
  time.sleep(6)
  print("Task1 Concluída")

def task2():
  print("Task2 Iniciada")
  time.sleep(4)
  print("Task2 Concluída")

def main():
  print('Aplicação Iniciada')
  inicio = time.time()
  t1 = threading.Thread(target=task1)
  t2 = threading.Thread(target=task2)
  t1.start()
  t2.start()
  t1.join()
  t2.join()
  print(f'Execução Finalizada em {time.time()-inicio:.2f} segs.')

main()
```

Isso não é um problema para o módulo threading.

T == 6 segundos

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Asyncio  Thread

slide03.01c_asyncio to_thread

```python
import asyncio
import time

async def task1():
    print("Task1 Iniciada")
    await asyncio.to_thread(time.sleep,6)
    print("Task1 Concluída")

async def task2():
    print("Task2 Iniciada")
    await asyncio.to_thread(time.sleep,6)
    print("Task2 Concluída")

async def main():
    print('Aplicação Iniciada')
    inicio = time.time()
    t1 = asyncio.create_task(task1())
    t2 = asyncio.create_task(task2())
    await t1,t2
    print(f'Execução Finalizada em {time.time()-inicio:.2f} segs.')

asyncio.run(main())
```

Abordagem Híbrida: Asyncio + Thread

T == 6 segundos

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Asyncio

e o problema de Tarefas CPU-Bound?

```
async def main():
    number = 9_999_999_999_999_917
    efeito = asyncio.create_task(
                msg_blink(f'Verificando se {number} é Primo...')
             )
    result = await verificar_primo(number)
    efeito.cancel()
    if result:
        print(f'\n\n{number} é PRIMO')
    else:
        print(f'\n\n{number} NÃO É PRIMO')

asyncio.run(main())
```

slide03.02a_cpubound_asyncio.py

Threading

Também não é problema para threading...

```
def main():
    number = 9_999_999_999_999_917
    signal = threading.Event()
    efeito = threading.Thread(target=msg_blink,
                              args=(f'Verificando se {number} é Primo...',signal)
             )
    efeito.start()
    result = verificar_primo(number)
    signal.set()
    if result:
        print(f'\n\n{number} é PRIMO')
    else:
        print(f'\n\n{number} NÃO É PRIMO')


main()
```

slide03.02b_cpubound_threading.py

Asyncio  Thread

Abordagem Híbrida: Asyncio + Thread

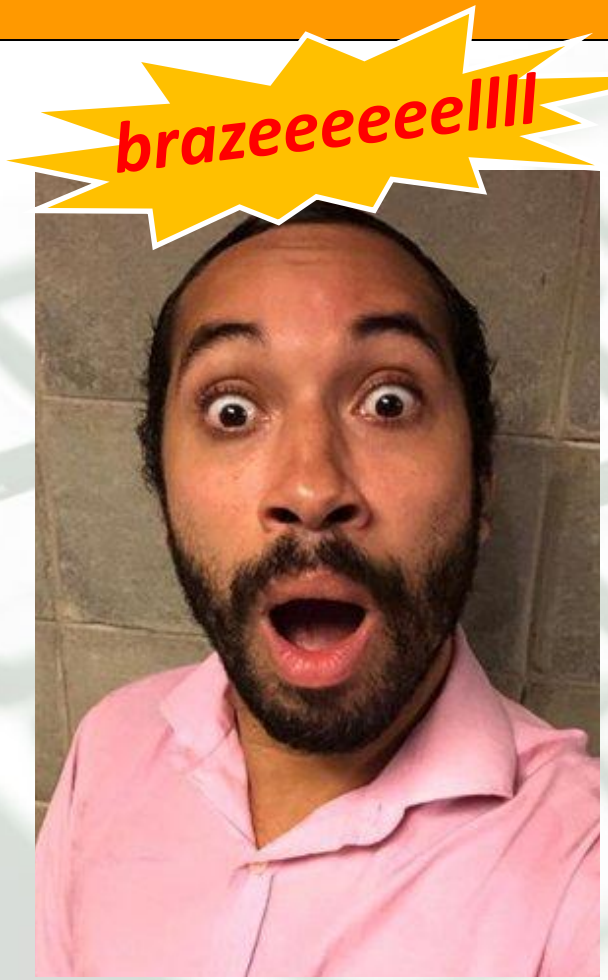
```
async def main():
    number = 9_999_999_999_999_917
    efeito = asyncio.create_task(
                msg_blink(f'Verificando se {number} é Primo...')
             )
    result = await asyncio.to_thread(verificar_primo,number)
    efeito.cancel()
    if result:
        print(f'\n\n{number} é PRIMO')
    else:
        print(f'\n\n{number} NÃO É PRIMO')

asyncio.run(main())
```

slide03.02c_cpubound_asyncio_to_thread.py

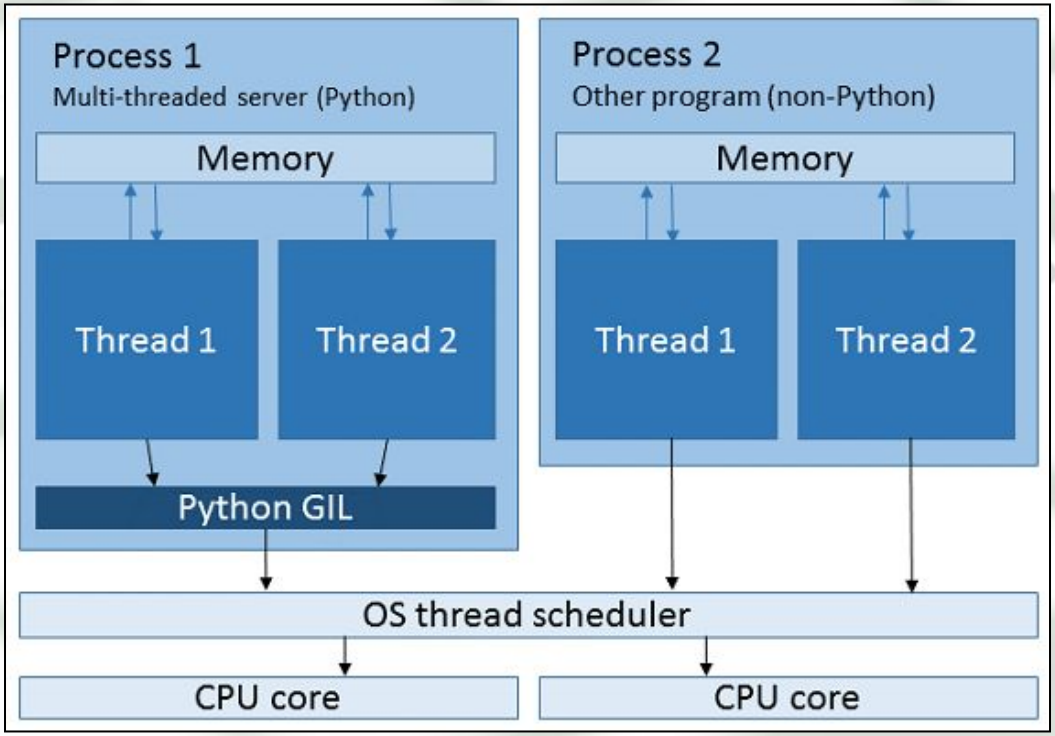
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Threading

Mas nem tudo são flores...



*O **GIL (Global Interpreter Locker)** obriga que todas threads de um processo python (<3.14) opere em modo **concorrente (sem paralelismo real)**.*



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

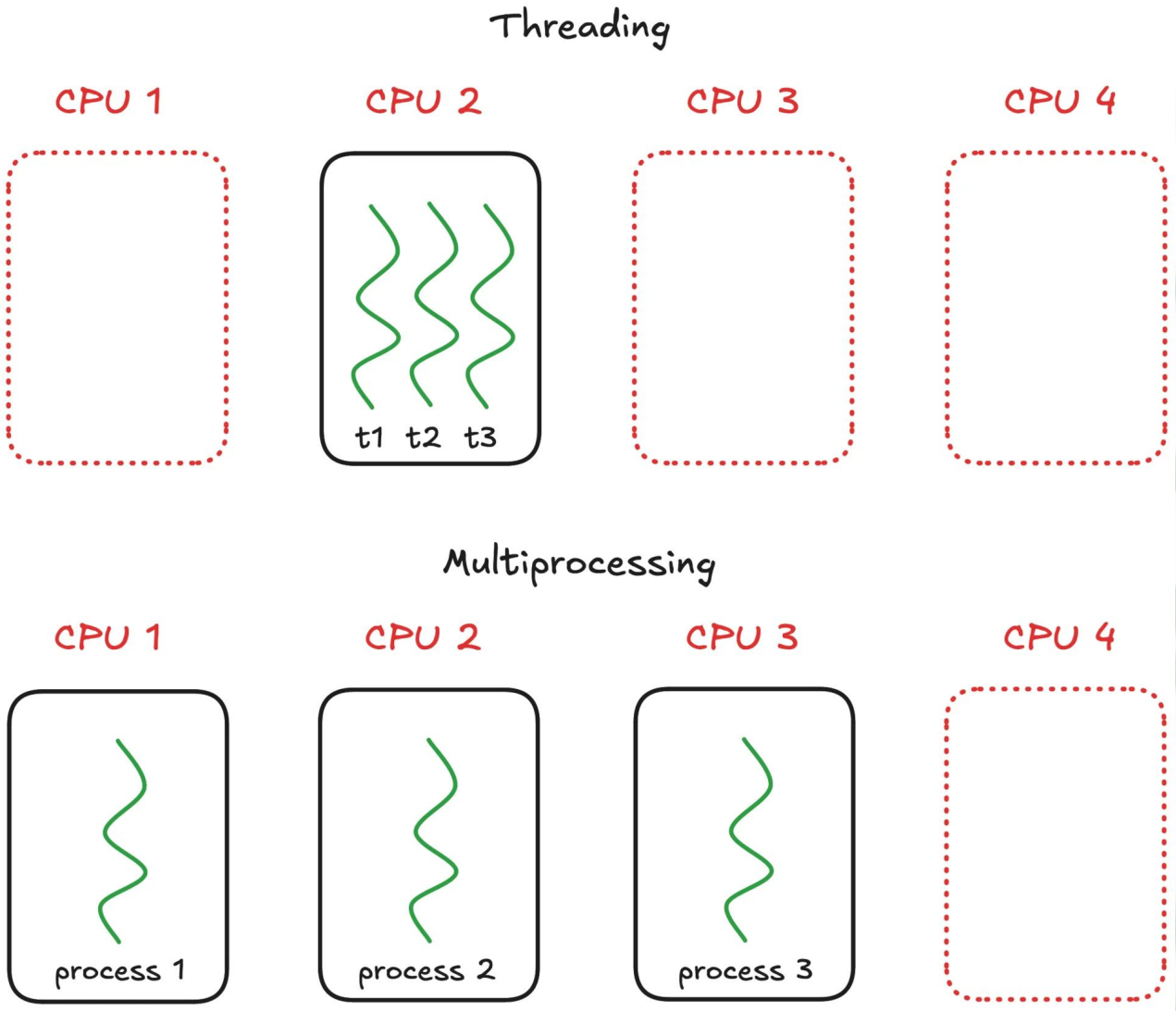
Threading

slide03.03_threading_gil.py

```
def main():
    numeros = [4_444_444_444_444_423,
               5_000_111_000_222_021,
               6_666_666_666_666_719,
               9_999_999_999_999_917]

    inicio = time.time()
    tasks = [threading.Thread(target=is_prime, args=(i,))
             for i in numeros]
    for t in tasks:
        t.start()

    for t in tasks:
        t.join()

    print(f'Total Threading = {time.time() - inicio:.2f} segs.')

main()
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Threading vs Multiprocessing



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Threading vs Multiprocessing

Threading

```
4444444444444423 é Primo
6666666666666719 é Primo
500011100022021 é Primo
9999999999999917 é Primo
Total Threading = 52.87 segs.
```

Concorrência

Multiprocessing

```
500011100022021 é Primo
4444444444444423 é Primo
6666666666666719 é Primo
9999999999999917 é Primo
Total Multiprocessing = 6.24 segs.
```

Paralelismo

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

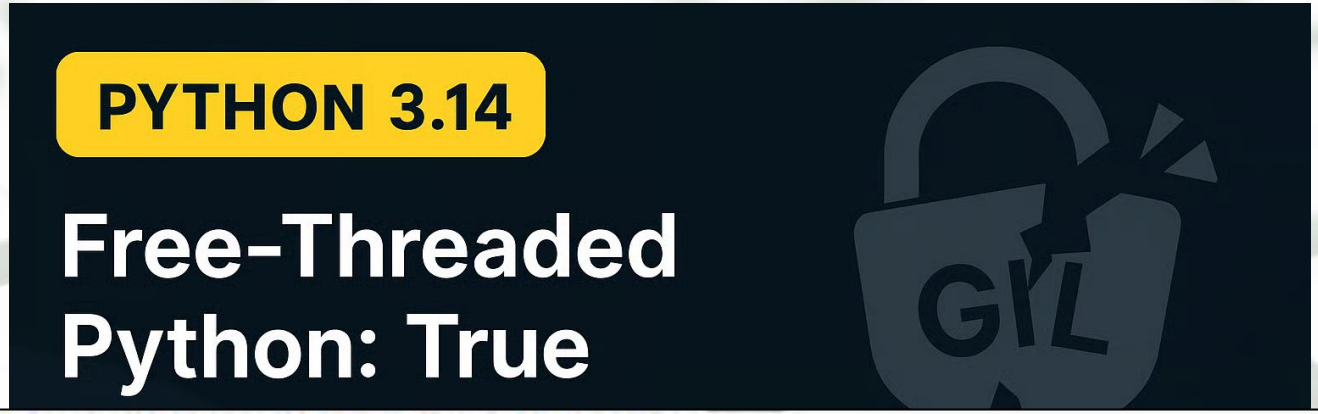
Threading UPDATE!

- **Python 3.14 (Outubro/2025)**
  - **Free-threaded Python is officially supported**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Threading UPDATE!

- **Python 3.14 (Outubro/2025)**
  - **Free-threaded Python is officially supported**




```
500011000222021 é Primo
4444444444444423 é Primo
6666666666666719 é Primo
9999999999999917 é Primo
Total Python 3.14 Free-Threaded = 6.56 segs.
```

Threading com Paralelismo

Python Threading

- Módulo **Threading** no **Python.**

```
import threading

def ola_mundo(idThread):
  print(f'Olá Mundo! Sou a thread {idThread}')

for i in range(5):
  t = threading.Thread(target=ola_mundo, args=(i+1,))
  t.start()
```

Laboratório 03-01

- Implemente cada modificação a seguir, em etapas (em cada modificação, execute várias vezes a solução e observe o comportamento):

1. Faça com que cada thread imprima a mesma mensagem 10 vezes.
2. Faça com que o tempo entre uma impressão e outra seja de 1 segundo (use `time.sleep`)
3. Na criação das threads, use os argumentos:

```
(target=ola_mundo, args=(i+1,), daemon=True)
```

4. Retire o passo anterior, e faça com que o programa termine toda a execução com a frase "**ATÉ MAIS**"

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

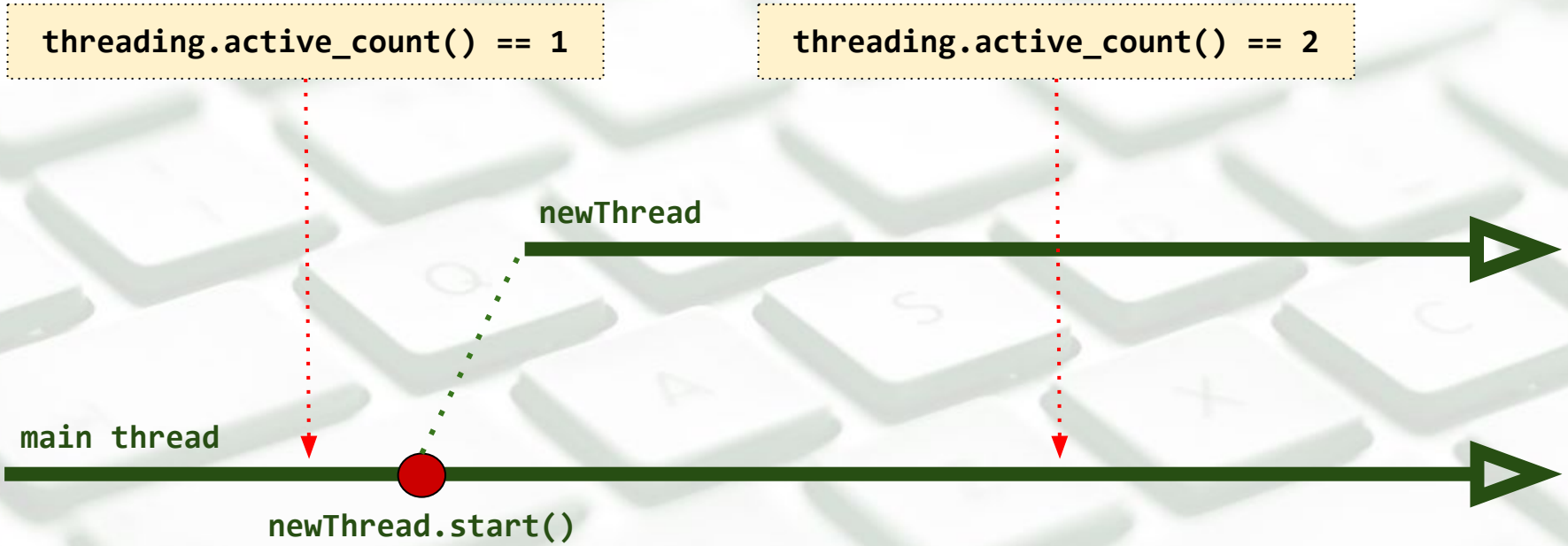
Python Threading

- Comportamento *multithreading*.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

O que acontece com o programa se a **mainThread** finaliza a sua execução antes da **newThread**?

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

O que acontece com o programa se a **mainThread** finaliza a sua execução antes da **newThread**?

Em modo padrão, o programa só encerra quando **todas as threads finalizam** sua execução!

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

O que acontece com o programa se a **mainThread** finaliza a sua execução antes da **newThread**?

É um problema quando precisamos de uma thread em loop infinito, p.ex.: aceitar requisições de conexão…

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

O que acontece com o programa se a **mainThread** finaliza a sua execução antes da **newThread**?

Threads com parâmetro daemon=True são encerradas assim que a main thread é finalizada.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

O que acontece com o programa se a **mainThread** finaliza a sua execução antes da **newThread**?

MAS... Como garantir agora que a newThread fez seu trabalho até o fim???

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

O que acontece com o programa se a **mainThread** finaliza a sua execução antes da **newThread**?

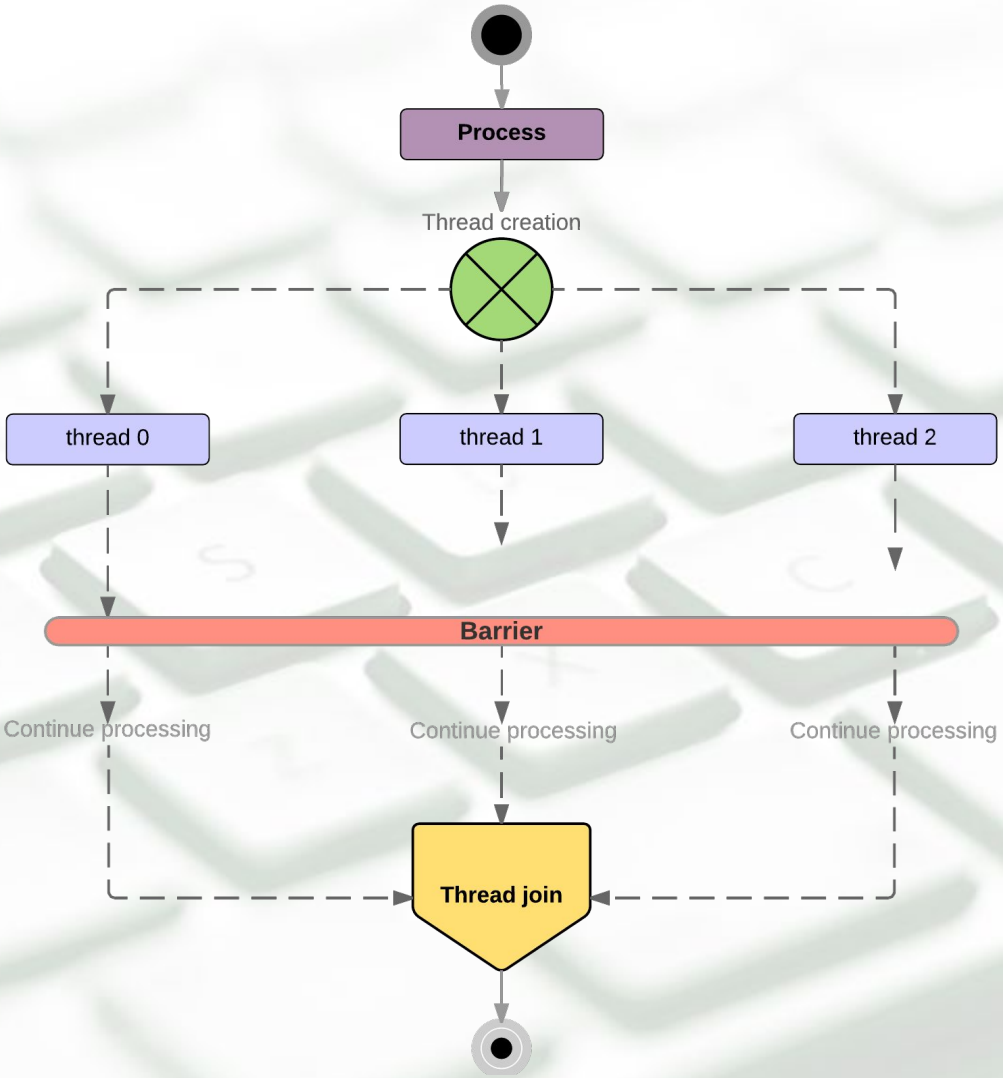
É necessário técnicas de sincronização!

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Método Join( )

Join é uma técnica de sincronização de threads que estabelece um **ponto de barreira.**

Se uma **thread t1** executa a instrução **t2.join()**, **t1** será bloqueado até a finalização de **t2**.

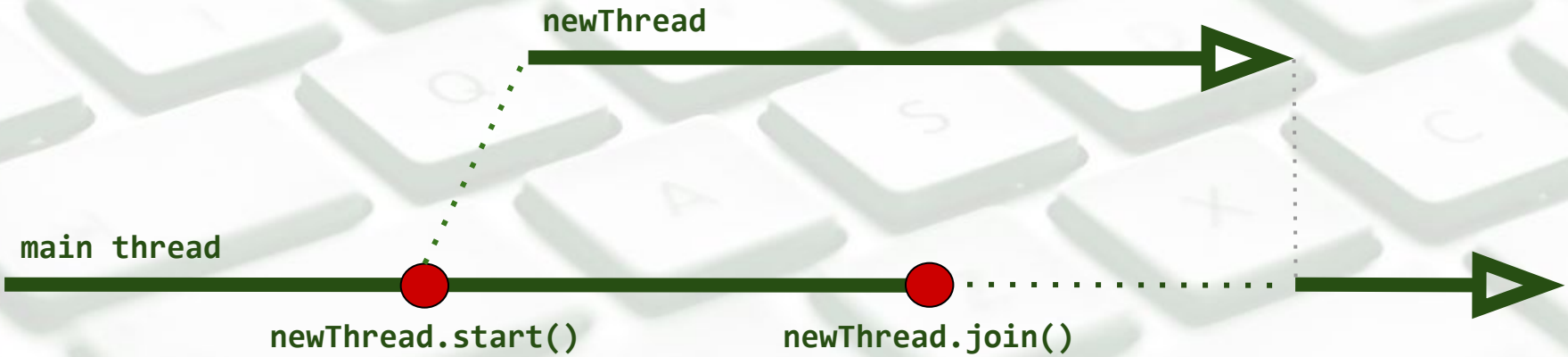


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

Sincronização com Join( )



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

Sincronização com Event( )

Laboratório 03-02

- **Threads Race**
  - Desenvolva um game que simula a corrida entre 5 threads.
  - Cada thread percorre a distância de 1 casa com tempo aleatório uma das outras - use random.random( ).
  - Atualize a tela constantemente (0.1s) para visualizar a corrida e acompanhar a evolução dos competidores.
  - Ganha a thread que percorrer primeiro 70 casas.
  - Ao final, seu programa deve informar a thread vencedora, ou se houve um empate.

Laboratório 03-02

- **Threads Race**

```
************************************************************ (61)
************************************************************** (66)
**************************************************************** (68)
****************************************************************** (70)
************************************************************** (66)
Thread 4 venceu!
```

```
********** (10)
******** (8)
********* (9)
********** (10)
********** (10)
Houve Empate
```

Laboratório 03-03

- **Laboratórios Práticos**
  - Faça uma (**única**) aplicação que consome a API…
    https://api.thecatapi.com/v1/images/search?limit=10
  - A partir dos links obtidos, baixe as imagens dos *gatíneos* para sua máquina, de duas formas:
    - Em **modo serial** (sem concorrência)
    - Em **modo concorrente**, utilizando ***threading***.
  - Meça o tempo final de execução em cada modo, compare e analise.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório 03-03

- **Laboratórios Práticos**

```
{'id': '2us', 'url': 'https://cdn2.thecatapi.com/images/2us.jpg', 'width': 2048, 'height': 1536}
{'id': '3eg', 'url': 'https://cdn2.thecatapi.com/images/3eg.jpg', 'width': 500, 'height': 332}
{'id': '4lp', 'url': 'https://cdn2.thecatapi.com/images/4lp.gif', 'width': 300, 'height': 225}
{'id': '7f4', 'url': 'https://cdn2.thecatapi.com/images/7f4.jpg', 'width': 681, 'height': 1024}
{'id': '9gg', 'url': 'https://cdn2.thecatapi.com/images/9gg.jpg', 'width': 600, 'height': 427}
{'id': 'aak', 'url': 'https://cdn2.thecatapi.com/images/aak.jpg', 'width': 467, 'height': 700}
{'id': 'da5', 'url': 'https://cdn2.thecatapi.com/images/da5.jpg', 'width': 640, 'height': 427}
{'id': 'MTY2MTMzMQ', 'url': 'https://cdn2.thecatapi.com/images/MTY2MTMzMQ.jpg', 'width': 500, 'height': 539}
{'id': 'MjA2NjAyOA', 'url': 'https://cdn2.thecatapi.com/images/MjA2NjAyOA.jpg', 'width': 960, 'height': 638}
{'id': 'nqS9tUT3i', 'url': 'https://cdn2.thecatapi.com/images/nqS9tUT3i.jpg', 'width': 1000, 'height': 1000}
Tempo Serial: 9.16 segundos
{'id': '2us', 'url': 'https://cdn2.thecatapi.com/images/2us.jpg', 'width': 2048, 'height': 1536}
{'id': '3eg', 'url': 'https://cdn2.thecatapi.com/images/3eg.jpg', 'width': 500, 'height': 332}
{'id': '4lp', 'url': 'https://cdn2.thecatapi.com/images/4lp.gif', 'width': 300, 'height': 225}
{'id': '7f4', 'url': 'https://cdn2.thecatapi.com/images/7f4.jpg', 'width': 681, 'height': 1024}
{'id': '9gg', 'url': 'https://cdn2.thecatapi.com/images/9gg.jpg', 'width': 600, 'height': 427}
{'id': 'aak', 'url': 'https://cdn2.thecatapi.com/images/aak.jpg', 'width': 467, 'height': 700}
{'id': 'da5', 'url': 'https://cdn2.thecatapi.com/images/da5.jpg', 'width': 640, 'height': 427}
{'id': 'MTY2MTMzMQ', 'url': 'https://cdn2.thecatapi.com/images/MTY2MTMzMQ.jpg', 'width': 500, 'height': 539}
{'id': 'MjA2NjAyOA', 'url': 'https://cdn2.thecatapi.com/images/MjA2NjAyOA.jpg', 'width': 960, 'height': 638}
{'id': 'nqS9tUT3i', 'url': 'https://cdn2.thecatapi.com/images/nqS9tUT3i.jpg', 'width': 1000, 'height': 1000}
Tempo Threading: 0.28 segundos
```

compare e analise.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório 03-04a

- 03 ~~*amigos*~~ threads se reuniram para contar colaborativamente do número 1 ao número 100, tal como descrito na função abaixo…

```
def contar(id: int, num: int):
    while num<=100:
        print(f'Thread {id} contou número: {num}')
        num += 1
        time.sleep(1)
```

- Faça um programa que lance as 03 threads e interprete o que acontece…

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório 03-04b

- Os amigos perceberam que estão utilizando um **objeto imutável**, e decidiram alterá-lo para um **objeto mutável**...
- Pesquise a diferença entre objetos **mutáveis** e **imutáveis**.
- Faça a alteração no código e **interprete** o que acontece...

```
def contar(id: int, num: list):
    while num[0]<=100:
        print(f'Thread {id} contou número: {num[0]}')
        num[0] += 1
        time.sleep(1)
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório

- **Algo errado não está certo!**
- **Precisamos contar corretamente…**

```
Thread 1 contou número: 1
Thread 2 contou número: 2
Thread 3 contou número: 3
Thread 1 contou número: 4
Thread 2 contou número: 5
Thread 3 contou número: 6
Thread 1 contou número: 7
Thread 2 contou número: 8
Thread 3 contou número: 9
Thread 1 contou número: 10
Thread 3 contou número: 11
Thread 2 contou número: 12
Thread 1 contou número: 13
Thread 3 contou número: 14
Thread 2 contou número: 15
Thread 1 contou número: 16
Thread 3 contou número: 17
Thread 2 contou número: 18
Thread 1 contou número: 19
Thread 3 contou número: 20
Thread 2 contou número: 21
Thread 1 contou número: 22
Thread 2 contou número: 23
Thread 3 contou número: 24
Thread 1 contou número: 25
Thread 2 contou número: 26
Thread 3 contou número: 27
Thread 1 contou número: 28
Thread 2 contou número: 29
Thread 3 contou número: 30
Thread 1 contou número: 31
Thread 2 contou número: 32
Thread 3 contou número: 33
Thread 1 contou número: 34
Thread 2 contou número: 35
Thread 3 contou número: 36
Thread 1 contou número: 37
Thread 2 contou número: 38
Thread 3 contou número: 39
Thread 1 contou número: 40
```