INSTITUTO FEDERAL DO NORTE DE MINAS GERAIS - CAMPUS JANUÁRIA

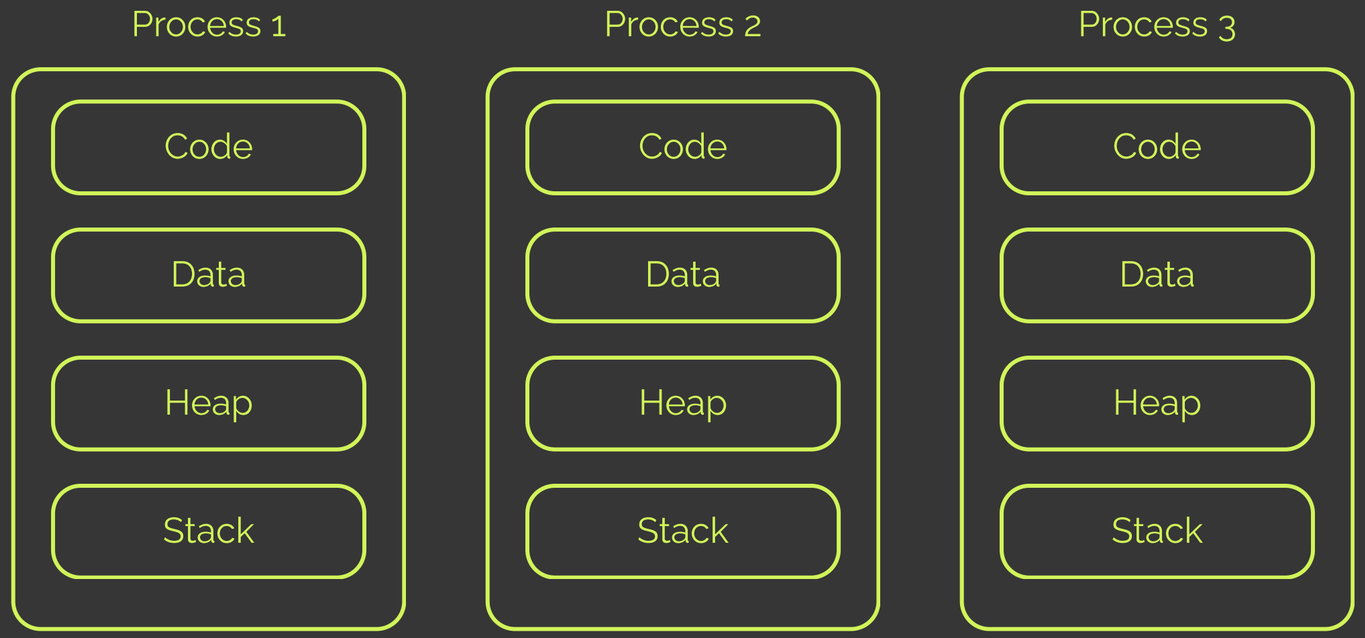
PYTHON THREADING E GLOBAL INTERPRETER LOCK (GIL)

SISTEMAS DISTRIBUÍDOS

OUT 2025

REVISÃO - THREADS VS. PROCESSOS

“Um **processo** é uma instância em execução de um programa, que possui seu próprio espaço de memória, tabelas de arquivos abertos, recursos e contexto de execução.”



REVISÃO - THREADS VS. PROCESSOS

> Um **processo** é uma instância em execução de um programa, que possui seu próprio espaço de memória, tabelas de arquivos abertos, recursos e contexto de execução.”

Contexto

Conjunto de informações e recursos necessários para sua execução

Process Control Block

Armazena todos os dados necessários para iniciar ou retomar um processo

- ID de Processo,
- Estado da CPU do processo,
- Informações de gerenciamento de memória

REVISÃO - THREADS VS. PROCESSOS

Troca de contexto

REVISÃO - THREADS VS. PROCESSOS

> “Uma thread é a menor unidade de execução dentro de um processo. É uma estrutura essencialmente ligada ao seu processo pai, mas possui componentes individuais que a distinguem de outras threads.”

Recursos Compartilhados

O principal objetivo de utilizar threads é **maximizar a utilização dos recursos do computador** e resolver o **problema de bloqueio**

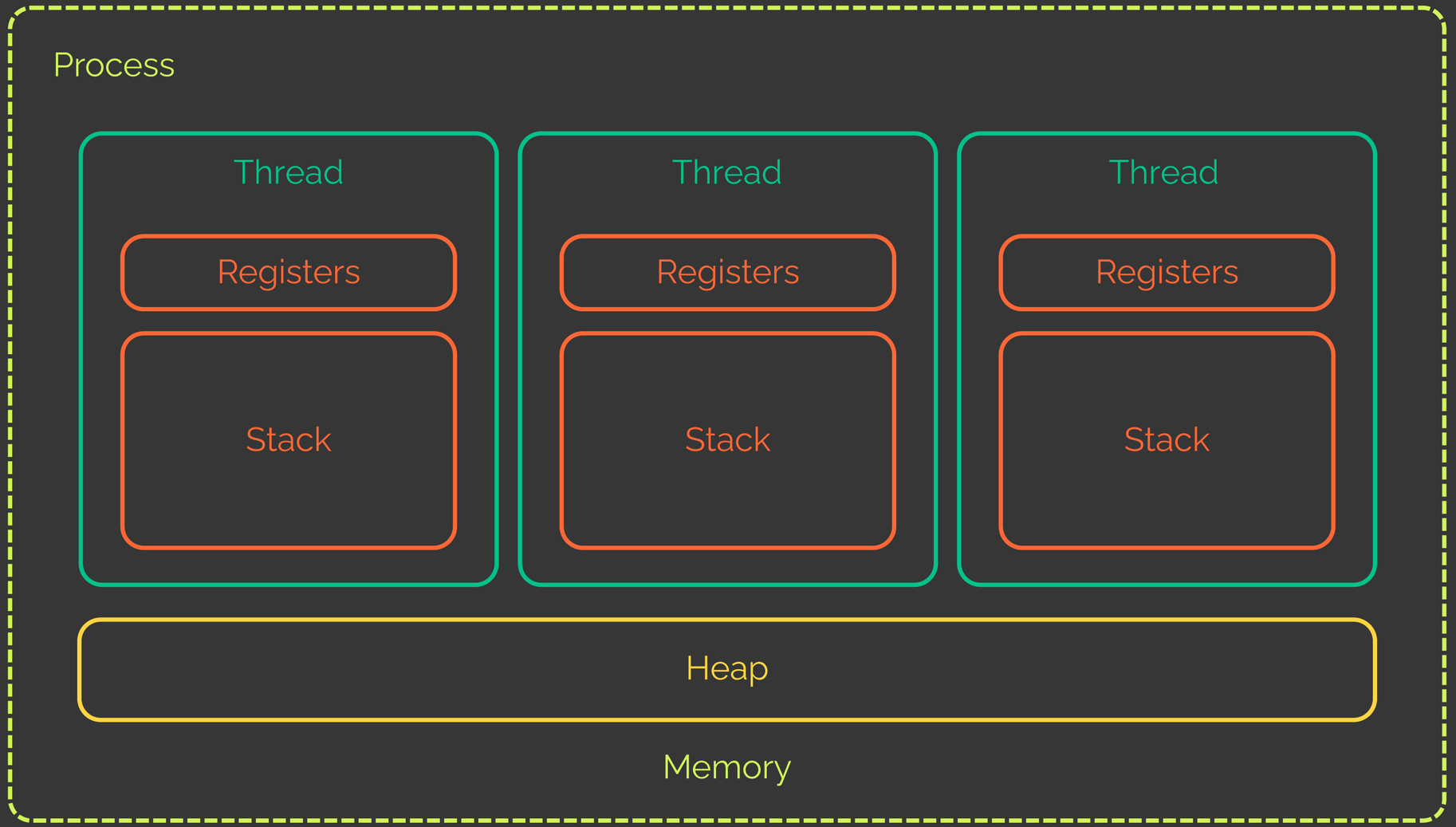
Comunicação

As threads dentro do mesmo processo compartilham todo o espaço de endereçamento, incluindo o código executável, constantes, e a Heap

Sicronização

Quando várias threads acessam o mesmo recurso, pode ocorrer uma condição de corrida

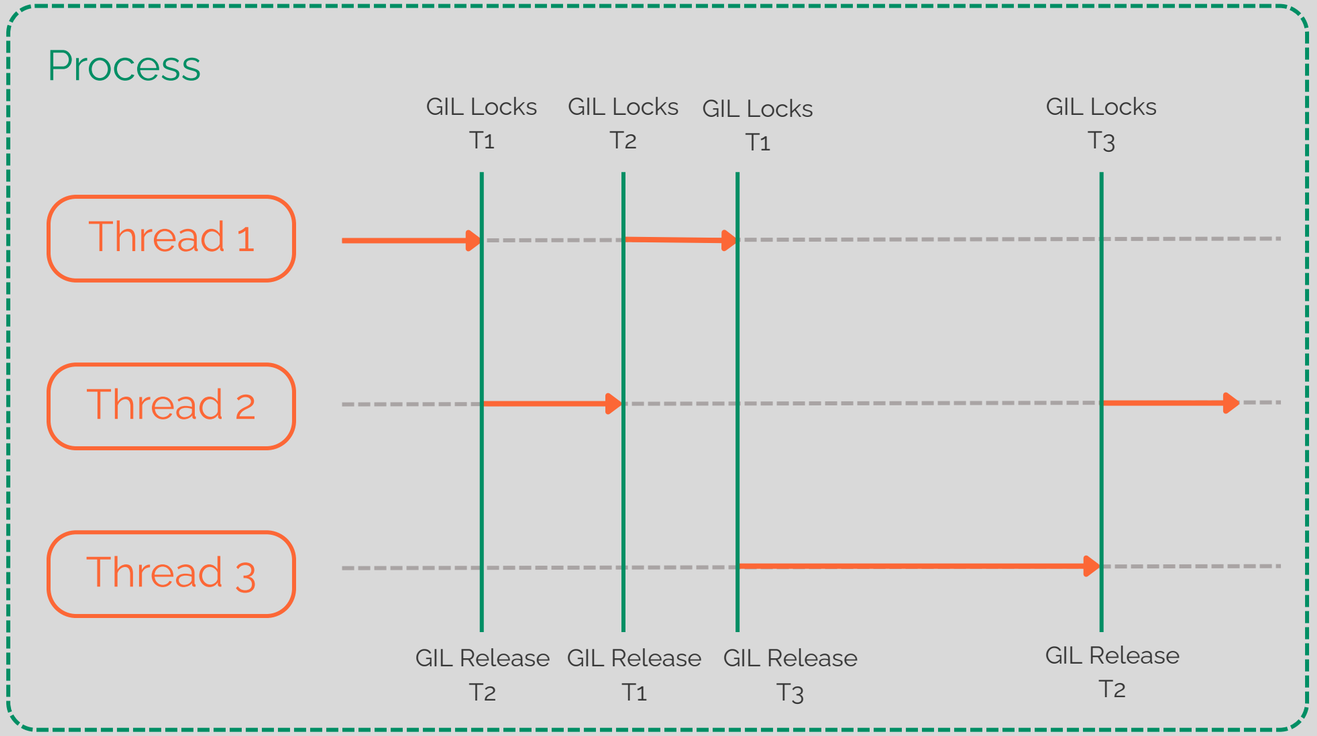
REVISÃO - THREADS VS. PROCESSOS



PYTHON GIL (GLOBAL INTERPRETER LOCK)

> Mecanismo que permite que apenas uma thread execute código Python por vez dentro do mesmo processo, mesmo em sistemas com múltiplos CPUs.

> Em tarefas que esperam por I/O, o GIL é liberado enquanto aguarda, permitindo que outras threads rodem



PYTHON GIL (GLOBAL INTERPRETER LOCK)

> O GIL afeta a programação multithreading sacrificando paralelismo em troca de simplicidade e segurança na gestão de memória.

> Para superar essa limitação, a abordagem recomendada é o uso de múltiplos processos (com o módulo multiprocessing), onde cada um tem seu próprio interpretador Python e não compartilha o GIL, permitindo paralelismo real em múltiplos núcleos.

> Em programas CPU-bound, o GIL introduz um gargalo: mesmo que o sistema tenha múltiplos núcleos

> Em programas I/O-bound, onde o GIL é liberado durante as esperas, a eficiência é maior

THREADING VS. ASYNCIO

THREADING	ASYNCIO
Concorrência baseada em threads	Concorrência cooperativa
O interpretador Python (CPython) usa o GIL para garantir que apenas uma thread execute bytecode por vez	Roda todas as tarefas em uma única thread, dentro de um event loop. Cada tarefa precisa cooperar e liberar o controle (await) quando está esperando I/O.
Têm custo maior de criação e alternância.	Como não cria várias threads reais, o overhead é muito menor

PYTHON THREADING

> “No contexto do Python, threading é um módulo interno que permite que vários threads sejam executados simultaneamente.”

Viabilidade

Situações onde o programa passa a maior parte do tempo esperando por operações de entrada/saída como leitura e escrita em arquivos, chamadas de rede, interação com bancos de dados.

Quando o usuário utiliza uma Interface Gráfica do Usuário (GUI), ela deve permanecer responsiva, apesar das tarefas serem executadas em segundo plano.

PYTHON THREADING

Criando threads em Python

1. Importar o módulo threading
2. Criar um função
3. Criar um novo objeto de thread

Para utilizar a função basta usar a função start()

```
import threading

def function_name():
    # code here


t = threading.Thread(target=function_name)
```

```
t.start()
```

PYTHON THREADING

Classes e Métodos da Biblioteca Threading

Classe Thread

Cria uma thread estendendo a classe Thread e sobrescrevendo o método run(), que define o que a thread executa.

Métodos importantes:

- start(): inicia a thread e chama internamente o método run() em background.
- run(): contém o código que será executado na thread.
- join(): bloqueia a chamada até a thread terminar sua execução.
- is_alive(): retorna se a thread ainda está em execução.

NOVA VERSÃO PYTHON 3.14 E FREE-THREADING

> Python 3.14 foi lançado oficialmente, trazendo consigo a promessa de uma revolução na performance

> A grande estrela desta versão é a opção 'free-threading', que permite desabilitar o Global Interpreter Lock (GIL)

> A mudança pode acelerar aplicações CPU-bound em mais de 300%.

> No Python 3.13, para contornar isso era por meio do multiprocessamento, que permite executar vários processos em todos os núcleos da CPU em vez de em apenas um.

> Cada tarefa executada em uma CPU separada possui seu próprio GIL, permitindo verdadeiro paralelismo

NOVA VERSÃO PYTHON 3.14 E FREE-THREADING

Comparando a atualização

```python
import threading
import time
import sys

print(f"Python version: {sys.version}")

def count():
    count = 0
    for _ in range(10**9):
        count += 1

start = time.time()

t1 = threading.Thread(target=count)
t2 = threading.Thread(target=count)

t1.start()
t2.start()

t1.join()
t2.join()

end = time.time()
print(f"CPU-bound threaded time: {end - start:.2f} seconds")
```

NOVA VERSÃO PYTHON 3.14 E FREE-THREADING

Comparando a atualização

```
(py312) mac:~ $ python threading_test.py
Python version: 3.12.2 (v3.12.2:6abddd9f6a, Feb  6 2024, 17:02:06) [Clang
13.0.0 (clang-1300.0.29.30)]
CPU-bound threaded time: 37.35 seconds
```

```
(py314) mac:~ bmolyneaux$ python threading_test.py
Python version: 3.14.0b4 (main, Jul 21 2025, 19:14:12) [Clang 15.0.0 (clang-
1500.1.0.2.5)]
CPU-bound threaded time: 27.21 seconds
```

OBRIGADA!