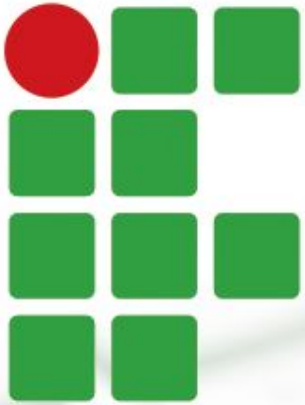


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

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)

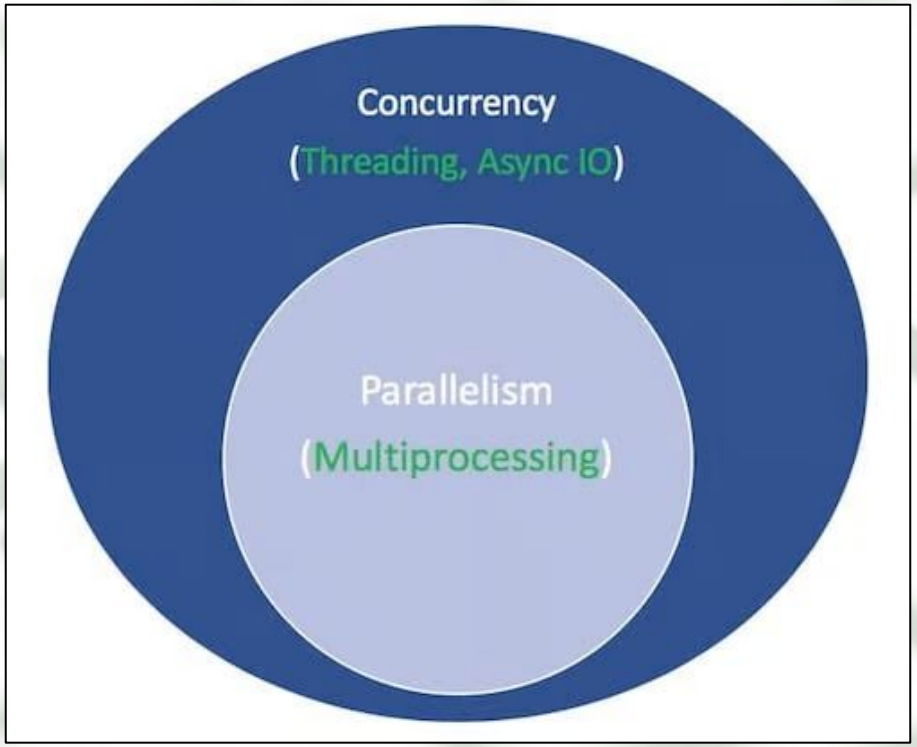
Concorrência

- Em Python…

Corrotinas
asyncio, trio, curio, …

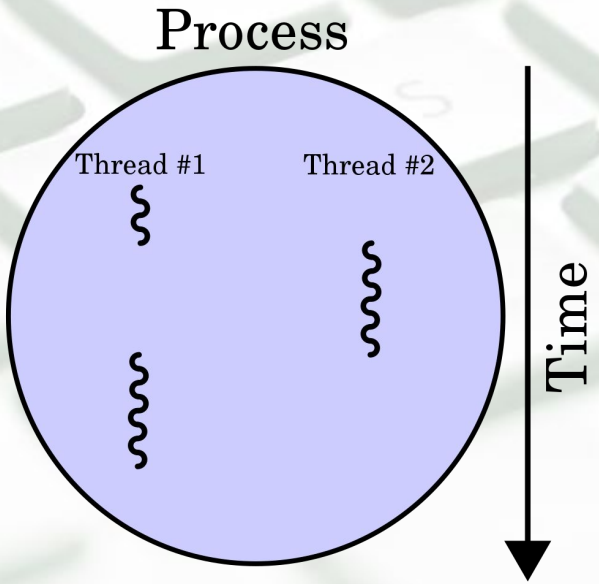
Threads
threading

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**.

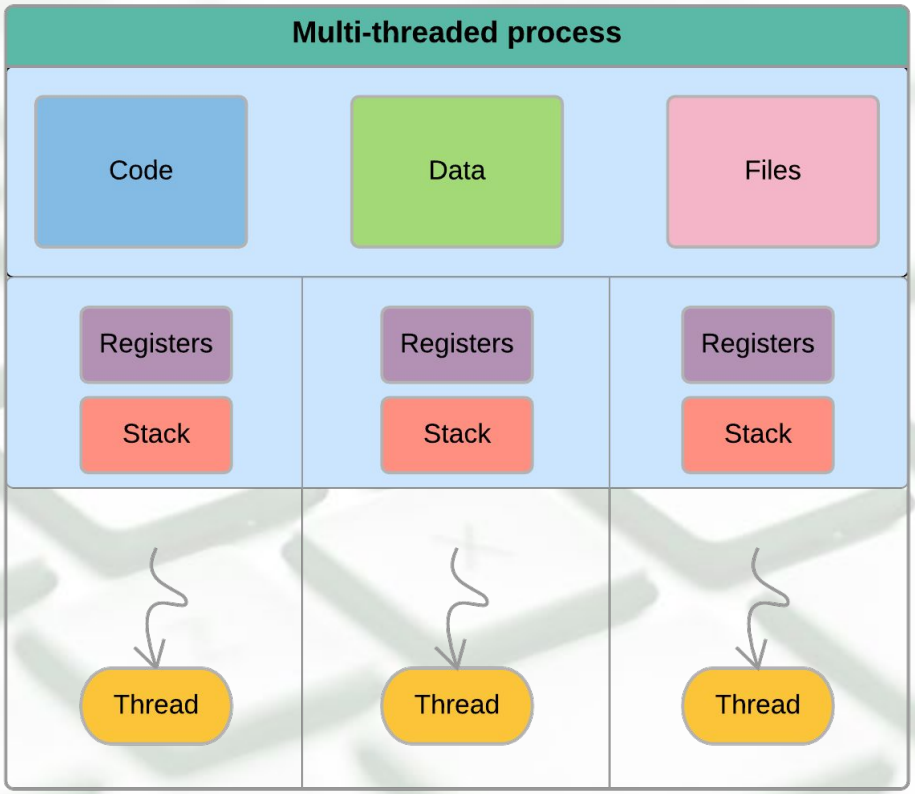


Threads

- 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**.

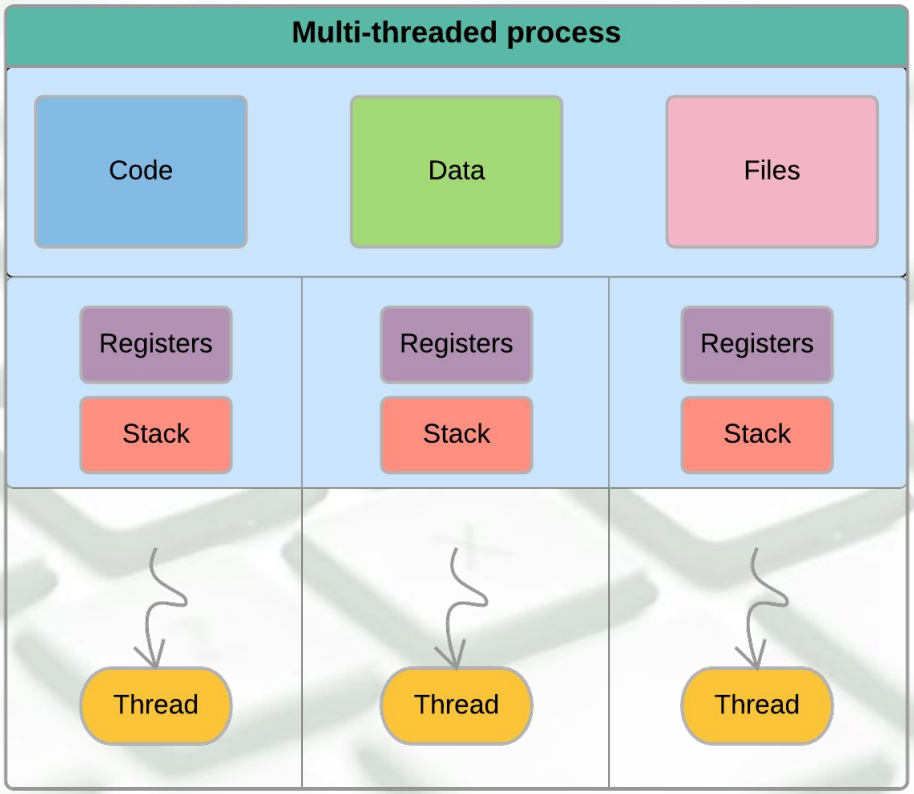
Threads

- 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)

Asyncio

```
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}')

asyncio.run(main())
```

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

T == 10 segundos

Threading

```
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}')

main()
```

Isso não é um problema para o módulo threading.

T == 6 segundos

Python Threading

- Módulo **Threading** no **Python.**

```
import threading

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

threads = []

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

Python Threading

- 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 2 segundos (use 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

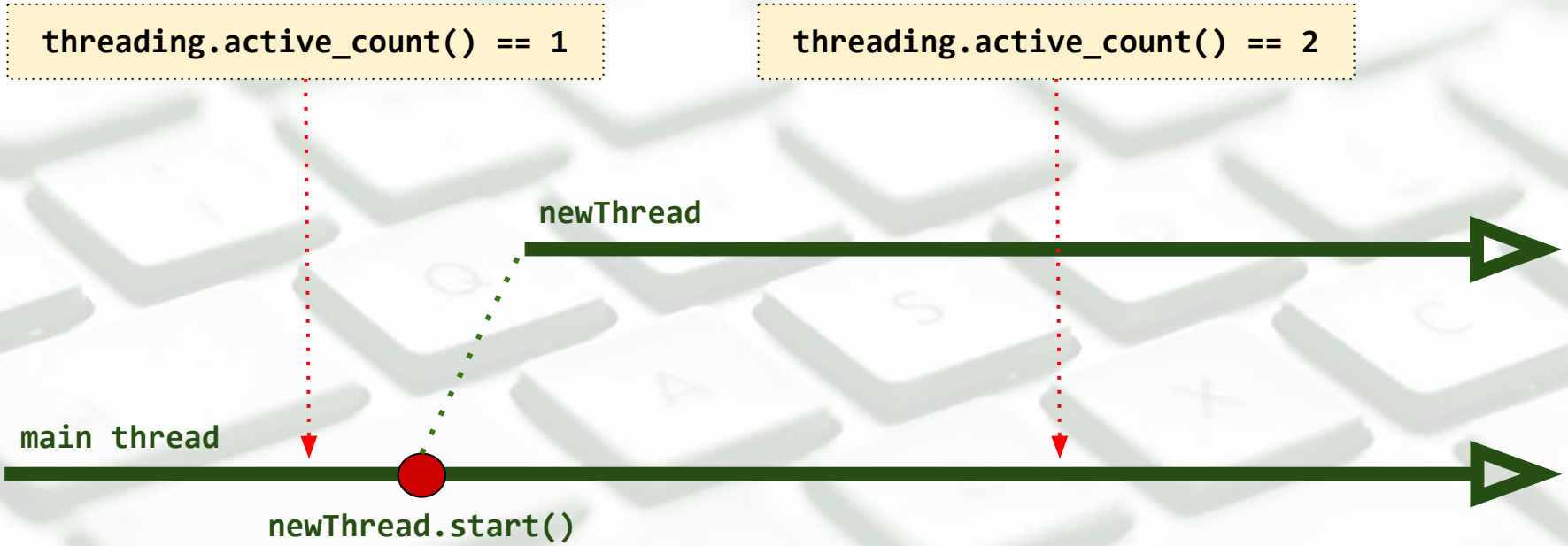
Python Threading

- 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**?

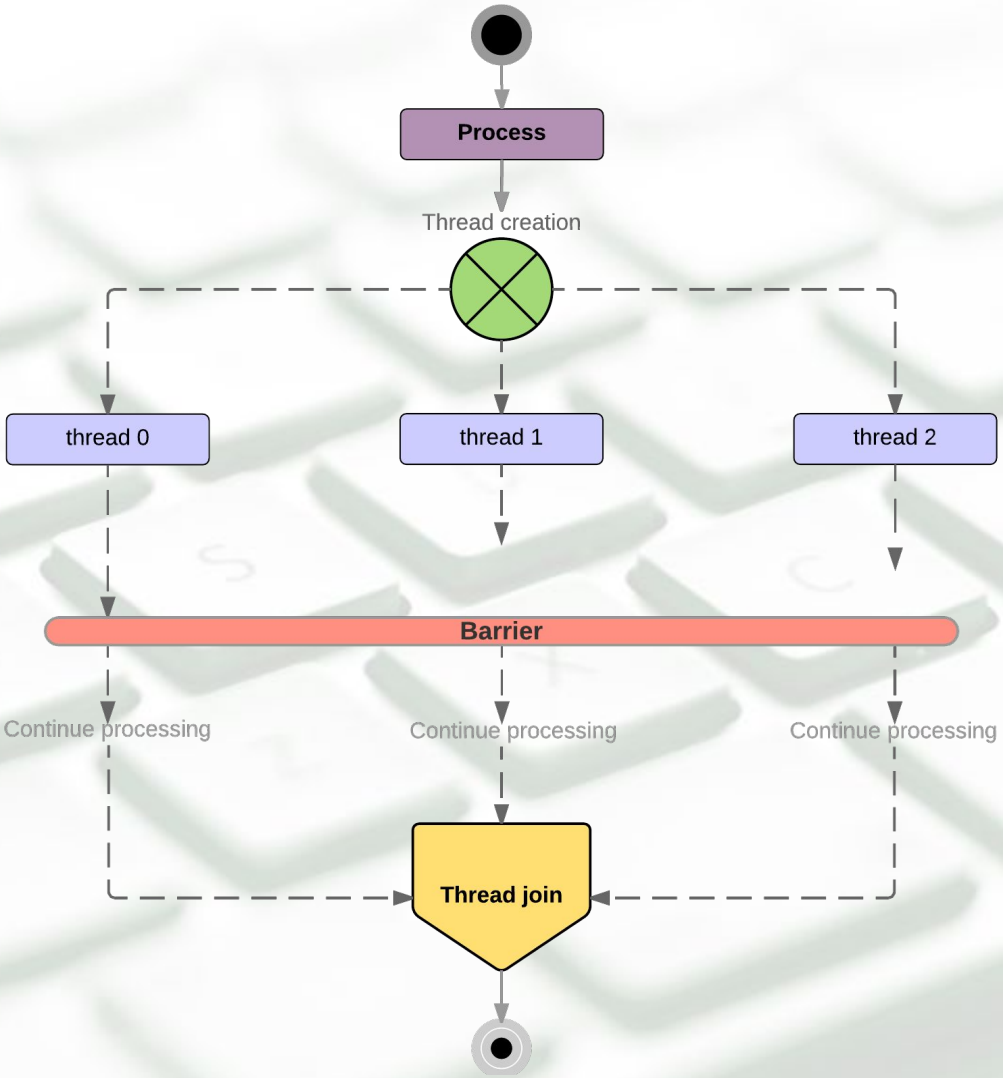
É necessário técnicas de sincronização!

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**.

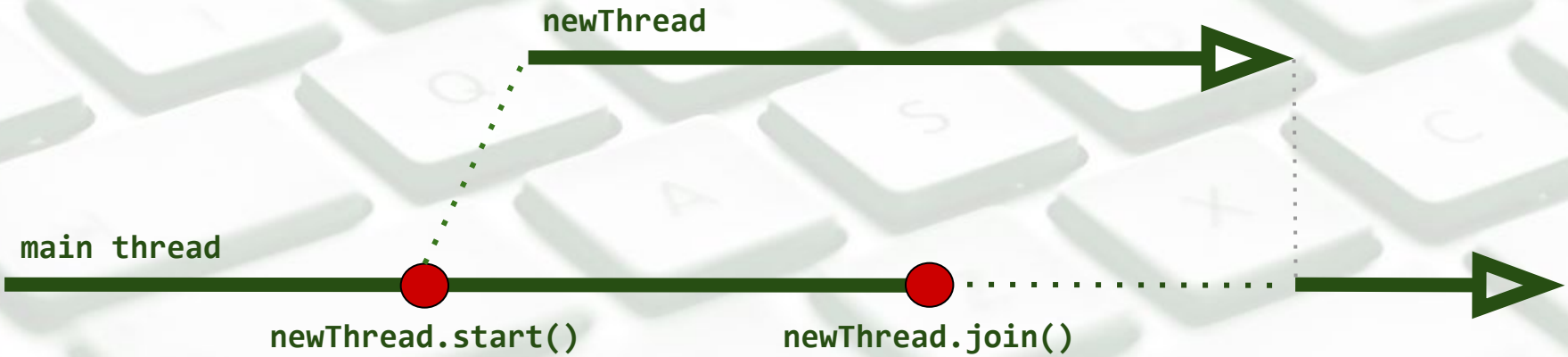


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

Sincronização com Join( )



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Python Threading

- Comportamento *multithreading*.

Sincronização com Event( )

Laboratório 03-01

- **Corrida Maluca**
  - 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 para visualizar a corrida e acompanhar a evolução dos competidores.
  - Ganha a thread que chegar primeiro à casa 80.
  - Ao final, seu programa deve informar a thread vencedora, ou se houve um empate.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório 03-01

```
**********************************************************************************
**********************************************************************
**************************************************************************
**********************************************************************************************
**************************************************************
THREAD 4 VENCEDOR

Process finished with exit code 0
```

```
**********************************************************************************
**********************************************************************************
**************************************************************************
**********************************************************************
**************************************************************
HOUVE EMPATE

Process finished with exit code 0
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório 03-02

- **Laboratórios Práticos**
  - Faça uma (**única**) aplicação que consome a API…
    https://api.thecatapi.com/v1/images/search?limit=10
  - Baixe as imagens contidas no objeto recebido, de duas formas (use o id obtido como nome do arquivo):
    - 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

- Imagine uma pizza gigante, com 128 pedaços disponíveis.
- Imagine que 3 threads que irão consumir toda essa pizza, começando do primeiro pedaço, até o último.
- Cada thread demora aproximadamente 3 segundos para comer cada pedaço.
- Cada pedaço de pizza é ÚNICO.
- Faça a implementação de uma aplicação em Python que simula esse cenário…

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório 03-03

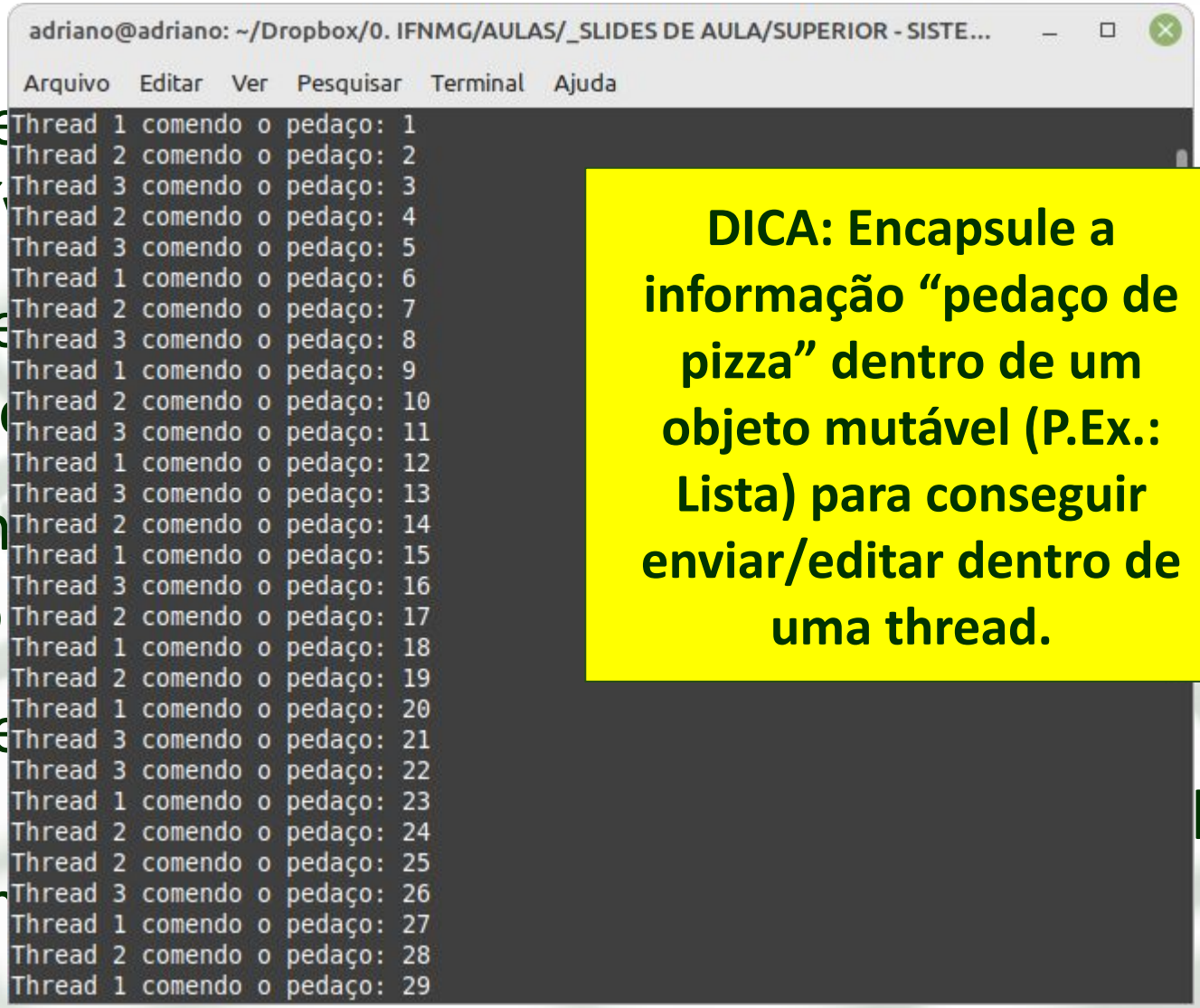
- Imagine
  disponí
- Imagine ssa
  pizza, c mo.
- Cada th dos
  para co
- Cada pe
- Faça a i hon
  que sim

```
adriano@adriano: ~/Dropbox/0. IFNMG/AULAS/_SLIDES DE AULA/SUPERIOR - SISTE...
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
Thread 1 comendo o pedaço: 1
Thread 2 comendo o pedaço: 2
Thread 3 comendo o pedaço: 3
Thread 2 comendo o pedaço: 4
Thread 3 comendo o pedaço: 5
Thread 1 comendo o pedaço: 6
Thread 2 comendo o pedaço: 7
Thread 3 comendo o pedaço: 8
Thread 1 comendo o pedaço: 9
Thread 2 comendo o pedaço: 10
Thread 3 comendo o pedaço: 11
Thread 1 comendo o pedaço: 12
Thread 3 comendo o pedaço: 13
Thread 2 comendo o pedaço: 14
Thread 1 comendo o pedaço: 15
Thread 3 comendo o pedaço: 16
Thread 2 comendo o pedaço: 17
Thread 1 comendo o pedaço: 18
Thread 2 comendo o pedaço: 19
Thread 1 comendo o pedaço: 20
Thread 3 comendo o pedaço: 21
Thread 3 comendo o pedaço: 22
Thread 1 comendo o pedaço: 23
Thread 2 comendo o pedaço: 24
Thread 2 comendo o pedaço: 25
Thread 3 comendo o pedaço: 26
Thread 1 comendo o pedaço: 27
Thread 2 comendo o pedaço: 28
Thread 1 comendo o pedaço: 29
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório 03-03

- Imagine
  disponí
- Imagine
  pizza, c
- Cada th
  para co
- Cada pe
- Faça a i
  que sim



DICA: Encapsule a informação “pedaço de pizza” dentro de um objeto mutável (P.Ex.: Lista) para conseguir enviar/editar dentro de uma thread.