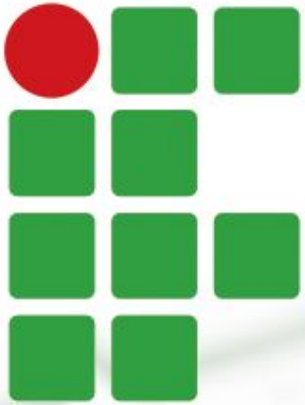


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Sistemas Distribuídos

- Paralelismo -

Avalie a Afirmação

Avalie a seguinte afirmação:

TODO SISTEMA CONCORRENTE (QUE IMPLEMENTA *THREADS*) É MAIS EFICIENTE DO QUE SISTEMAS SEQUENCIAIS.

Laboratório Prático

- Dada a seguinte função:

```
def contar(max):
    n = 0
    while n < max:
        n += 1
```

- Implemente três versões, sendo:
  - Programação Sequencial (sem *multithreading*)
    - Contar até 500 Milhões
  - Utilizando 2 Threads (cada thread contará 250 Milhões)
  - Utilizando 5 Threads (cada thread contará 100 Milhões)
- Meça o tempo de execução nas três versões e analise…

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório Prático

- Dada a seguinte função:

```
def contar(max):
    n = 0
    while n < max:
        n += 1
```

- Implemente três versões, sendo:
  - Utilizando 2 Threads (cada thread contará 250 Milhões)
- Meça o tempo de execução nas três versões e analise...

```
TEMAS DISTRIBUÍDOS/__source$ python3 lab03.05_contarMilhoes.py
Tempo Gasto - Sequencial: 21.223316431045532
Tempo Gasto - 2 Threads: 36.62672805786133
Tempo Gasto - 5 Threads: 37.88188123703003
```

Thread Única (Processo)	2 Threads	5 Threads
21.22 segundos	36.62 segundos (+72%)	37.88 segundos (+79%)

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório Prático

- Dada a seguinte funcão:

```
def contar(max):
    n = 0
    while n < max:
        n += 1
```

Execução concorrente gastou mais tempo!!! Porquê?!

- Implemente três versões, sendo:

```
TEMAS DISTRIBUÍDOS/__source$ python3 lab03.05_contarMilhoes.py
Tempo Gasto - Sequencial: 21.223316431045532
Tempo Gasto - 2 Threads: 36.62672805786133
Tempo Gasto - 5 Threads: 37.88188123703003
```

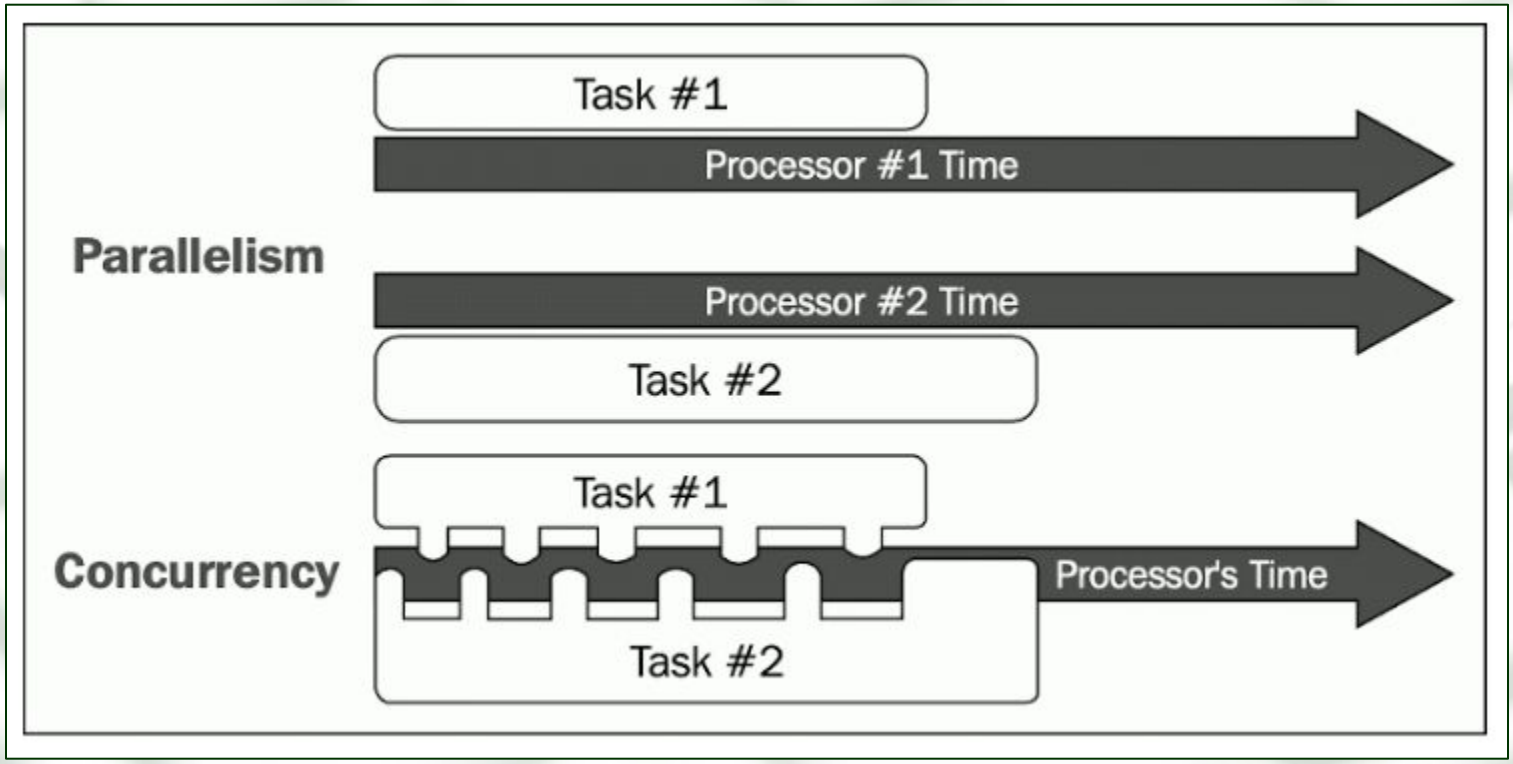
- Utilizando 2 Threads (cada thread contará 250 Milhões)

Thread Única (Processo)	2 Threads	5 Threads
21.22 segundos	36.62 segundos (+72%)	37.88 segundos (+79%)

- Meça o tempo de execução nas três versões e analise...

Não esqueça...

CONCORRÊNCIA != PARALELISMO



Filosofando...

- ***Concorrência*** *é sobre* ***lidar*** *com várias coisas ao mesmo tempo...*

- ***Paralelismo*** *é sobre* ***fazer*** *várias coisas ao mesmo tempo...*

Sendo mais formal...

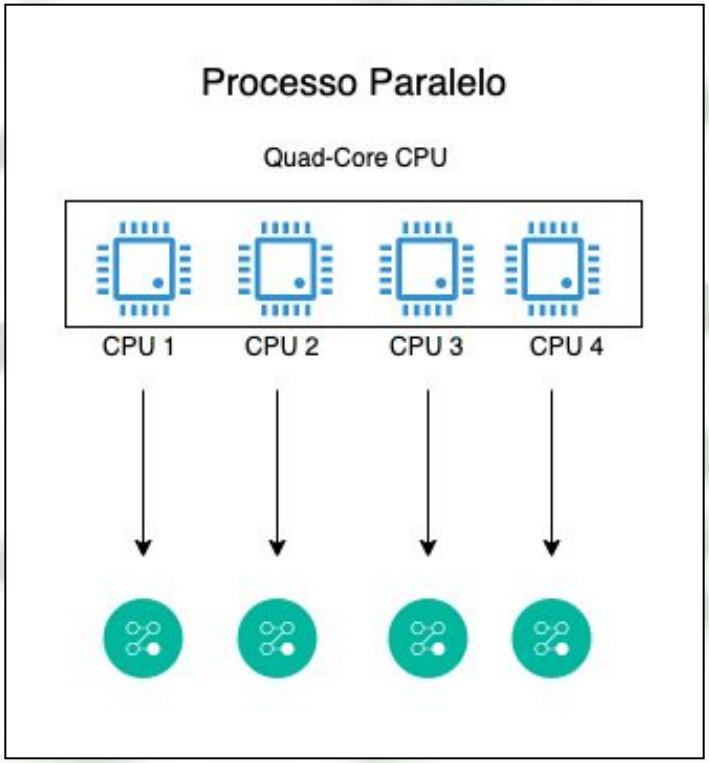
- ***Concorrência*** *é a capacidade de se executar duas ou mais tarefas em* ***um período de tempo****;*

- ***Paralelismo*** *é a capacidade de se executar duas ou mais tarefas de* ***ao mesmo tempo****;*

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Processamento Paralelo

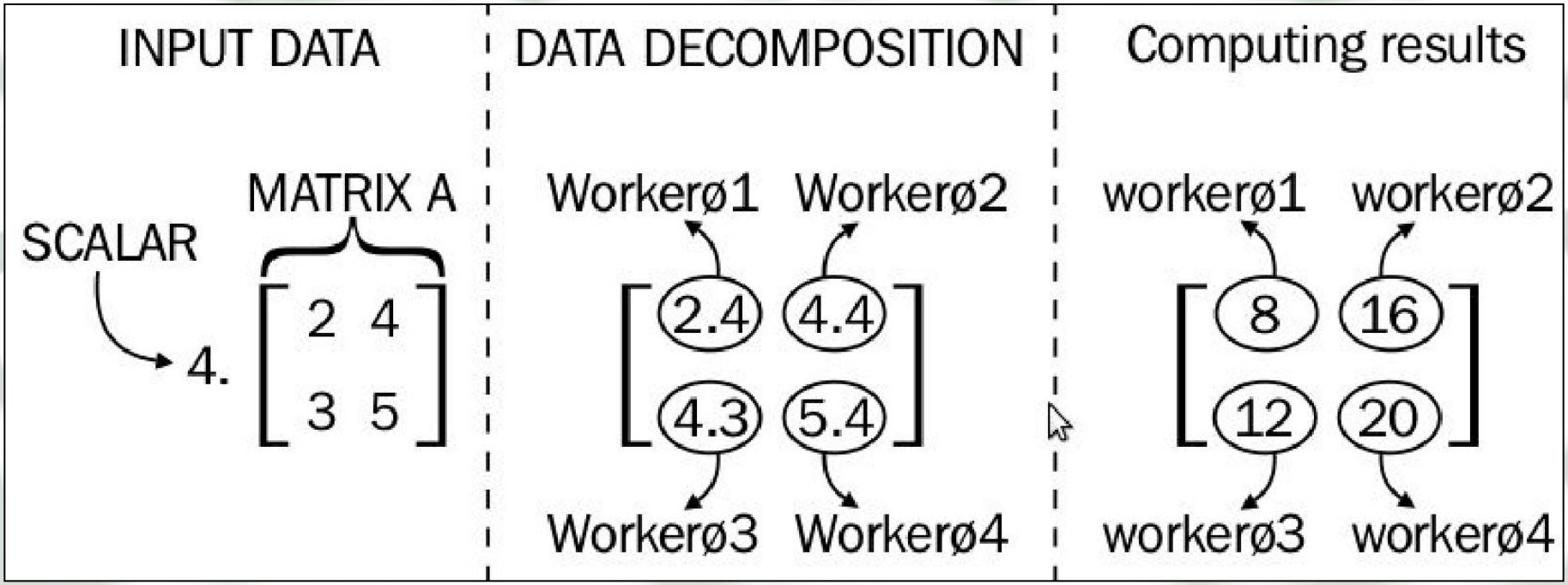
- **Processamento Paralelo** só faz sentido quando há disponibilidade de duas ou mais CPUs (ou Cores).



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Exemplos de Algoritmos

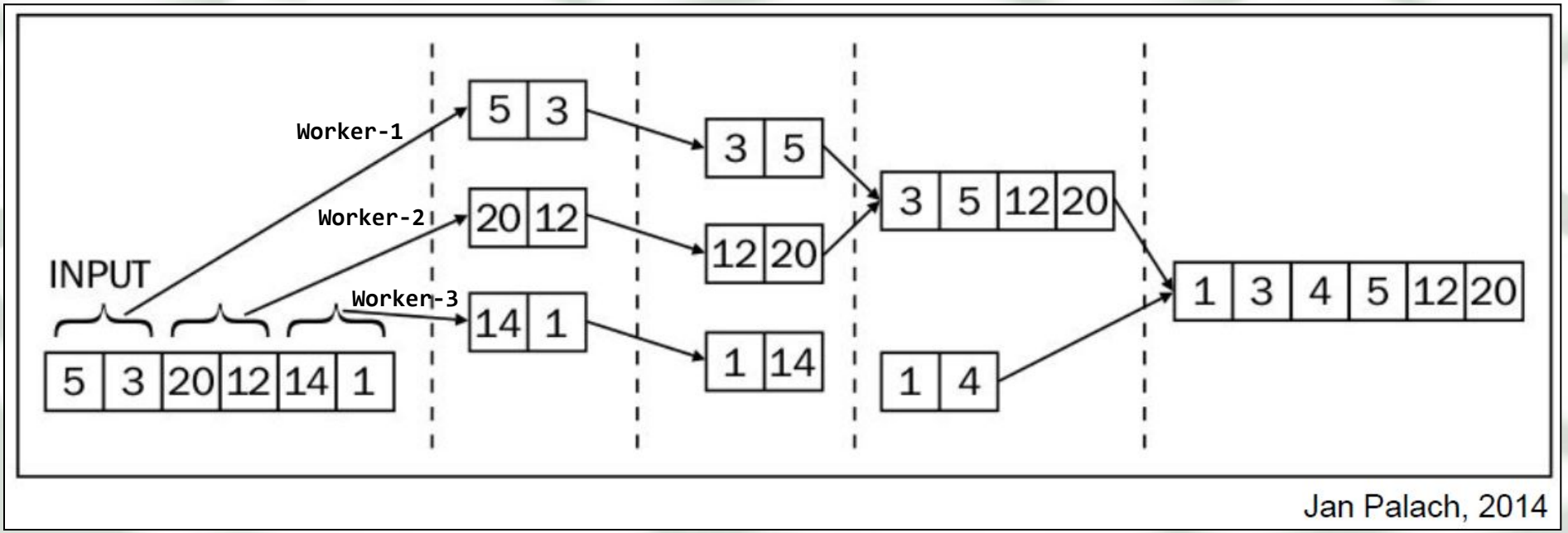
- Decomposição
  - Ex.: Multiplicação Escalar



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

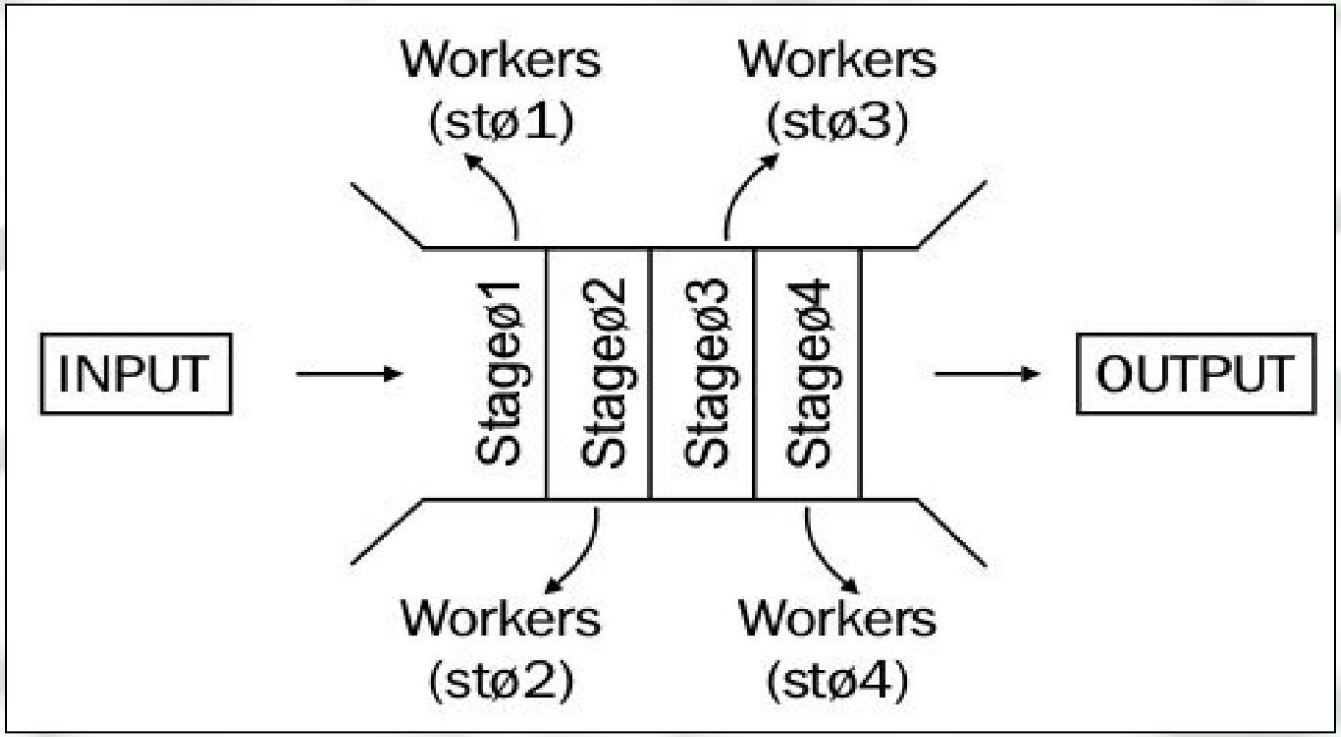
Exemplos de Algoritmos

- Dividir e Conquistar (*Divide and Conquer*)
  - Ex.: Ordenação



Exemplos de Algoritmos

- Pipeline
  - Ex.: Tratamento de Imagens (I.A.)



Módulos Python

- **Módulos Python:**

threading => Implementa **Concorrência**

multiprocessing => Implementa **Paralelismo**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Exemplo Prático...

```
TEMAS DISTRIBUÍDOS/__source$ python3 lab03.05_contarMilhoes.py
Tempo Gasto - Sequencial: 21.223316431045532
Tempo Gasto - 2 Threads: 36.626728057861 33
Tempo Gasto - 5 Threads: 37.88188123703003
```

Thread Única (Processo)	2 Threads	5 Threads
21.22 segundos	36.62 segundos (+72%)	37.88 segundos (+79%)

Exemplo Prático...

e se usássemos **multiprocessing**?

```
TEMAS DISTRIBUÍDOS/__source$ python3 lab03.05_contarMilhoes.py
Tempo Gasto - Sequencial: 21.223316431045532
Tempo Gasto - 2 Threads: 36.626728057861 33
Tempo Gasto - 5 Threads: 37.88188123703003
```

Thread Única (Processo)	2 Threads	5 Threads
21.22 segundos	36.62 segundos (+72%)	37.88 segundos (+79%)

Exemplo Prático...

e se usássemos **multiprocessing**?

```
TEMAS DISTRIBUÍDOS/__source$ python3 lab03.05_contarMilhoes.py
Tempo Gasto - Sequencial: 21.223316431045532
Tempo Gasto - 2 Threads: 36.626728057861 33
Tempo Gasto - 5 Threads: 37.88188123703003
```

```
TEMAS DISTRIBUÍDOS/__source$ python3 lab03.05_contarMilhoes_multipro
Tempo Gasto - Sequencial: 20.937788009643555
Tempo Gasto - 2 Processos: 12.816991567611694
Tempo Gasto - 5 Processos: 7.200292587280273
```

Thread Única (Processo)	2 Processos	5 Processos
20.93 segundos	12.81 segundos (-38%)	7.20 segundos (-66%)

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

O que muda na prática?!

adriano@adriano-notebook: ~

Arquivo Editar Ver Pesquisar Terminal Ajuda

```
  1  [||||||||||||                    33.5%]   Tasks: 196, 1013 thr, 107 kthr; 2 running
  2  [||||||||||||                    33.3%]   Load average: 0.72 0.80 0.80
  3  [|||||||||                       25.8%]   Uptime: 01:49:52
  4  [|||||||||||                     28.6%]
  Mem[|||||||||||||||||||||5.09G/7.63G]
  Swp[|||||||                       402M/2.00G]

  S      PID USER       PRI  NI   VIRT   RES    SHR S CPU% MEM%   TIME+  Command
  S    35382 adriano     20   0   377M  9200   6348 S 99.0  0.1  1:06.60 python3 lab03.05_c
  S    35407 adriano     20   0   377M  9200   6348 S 22.4  0.1  0:01.56 python3 lab03.05_c
  S    35408 adriano     20   0   377M  9200   6348 S 21.1  0.1  0:01.77 python3 lab03.05_c
  S    35409 adriano     20   0   377M  9200   6348 S 20.4  0.1  0:01.43 python3 lab03.05_c
  S    35411 adriano     20   0   377M  9200   6348 S 18.5  0.1  0:01.40 python3 lab03.05_c
  R    35410 adriano     20   0   377M  9200   6348 R 16.0  0.1  0:01.47 python3 lab03.05_c
  S      833 root        20   0   432M 60060  30048 S  3.2  0.8  3:07.53 /usr/lib/xorg/Xorg
  S    23137 adriano     20   0  7420M  1712M 32608 S  2.6 21.9  6:44.86 /snap/pycharm-comm
  R    35168 adriano     20   0 11656  5044   3176 R  1.3  0.1  0:06.34 htop
  S    35087 adriano     20   0   456M 41872  30900 S  1.3  0.5  0:02.17 /usr/libexec/gnome
  S     2391 adriano     20   0  3771M  108M 59996 S  1.3  1.4  4:14.49 cinnamon --replace
  S     1178 root        20   0   432M 60060  30048 S  1.3  0.8  0:27.01 /usr/lib/xorg/Xorg
  S    35397 adriano     20   0   375M 35004  27700 S  1.3  0.4  0:00.24 /usr/bin/gnome-scr
  S    23837 adriano     20   0  7420M  1712M 32608 S  0.6 21.9  1:12.36 /snap/pycharm-comm
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

O que muda na prática?!

adriano@adriano-notebook: ~/PycharmProjects/pythonProject

Arquivo Editar Ver Pesquisar Terminal Ajuda

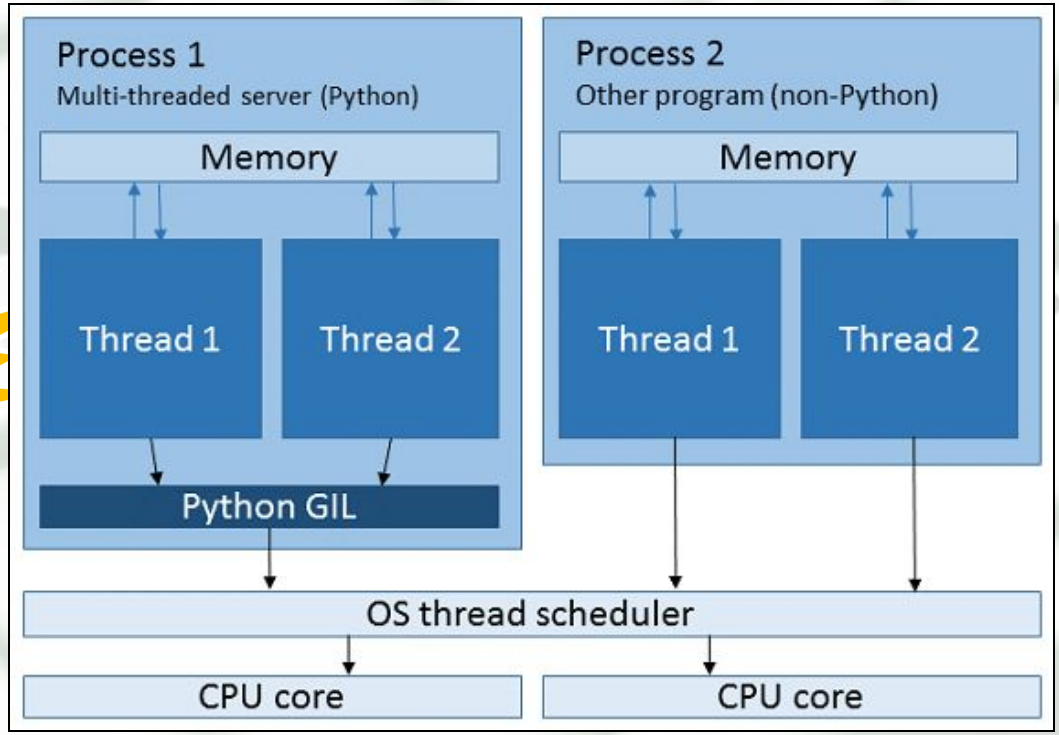
```
  1  [||||||||||||||||||||||||||100.0%]   Tasks: 175, 647 thr, 98 kthr; 4 running
  2  [||||||||||||||||||||||||||100.0%]   Load average: 4.45 3.53 2.52
  3  [||||||||||||||||||||||||||100.0%]   Uptime: 01:43:39
  4  [||||||||||||||||||||||||||100.0%]
  Mem[|||||||||||||||||||2.86G/7.63G]
  Swp[|                    51.5M/2.00G]

  S     PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  R   15609 adriano    20   0  110M 28216  4408 R 100.  0.4  1:14.99 python3 lab04.04_prim
  R   15608 adriano    20   0  110M 28220  4408 R 100.  0.4  1:15.17 python3 lab04.04_prim
  R   15613 adriano    20   0  110M 28220  4408 R 99.0  0.4  1:15.92 python3 lab04.04_prim
  R   15610 adriano    20   0  110M 28220  4408 R 99.0  0.4  1:14.81 python3 lab04.04_prim
  S   15606 adriano    20   0  101M 21764  6680 S  1.3  0.3  0:10.20 python3 lab04.04_prim
  R   14757 adriano    20   0 11404  4936  3304 R  1.3  0.1  0:21.22 htop
  S   15607 adriano    20   0  101M 21764  6680 S  0.7  0.3  0:02.87 python3 lab04.04_prim
  S    2703 adriano    20   0 32.3G  131M 95468 S  0.7  1.7  1:25.77 /opt/google/chrome/ch
  S     805 root       20   0  415M 95208 53732 S  0.0  1.2  4:02.42 /usr/lib/xorg/Xorg -c
  S   14922 adriano    20   0 1009M 60352 34996 S  0.0  0.8  0:03.04 /usr/bin/gnome-screen
  S   11279 adriano    20   0  1.1T  363M  125M S  0.0  4.6  1:37.54 /opt/google/chrome/ch
  S    2094 adriano    20   0 3774M  138M 71324 S  0.0  1.8  3:43.22 cinnamon --replace
  S    2627 adriano    20   0 32.6G  413M  186M S  0.0  5.3  5:31.87 /opt/google/chrome/ch
  S    2678 adriano    20   0 32.6G  413M  186M S  0.0  5.3  0:32.16 /opt/google/chrome/ch
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Observação Importante!!!

- Essa arquitetura ***multithreading*** (concorrente mas sem paralelismo) é uma característica **nativa do Python**, não adotada necessariamente em outras tecnologias.

*O **GIL** (Global Interpreter Locker) obriga que todas threads de um processo operem em modo **concorrente**.*



Multiprocessing

- Para implementar **paralelismo**, o módulo Python **multiprocessing** adota uma arquitetura ***process-based*** através da instanciação de **subprocessos**.
- Subprocessos são processos criados com **vínculo hierárquico**, ou seja: o processo criador é chamado de pai e, novos processos, filhos.
- Subprocessos também podem ter outros Subprocessos.

Multiprocessing

- Para implementar **paralelismo**, o módulo Python **multiprocessing** adota uma arquitetura ***process-based*** através da instanciação de **subprocessos**.
- Subprocessos são processos criados com **vínculo hierárquico**, ou seja: o processo criador é chamado de pai e, novos processos, filhos.
- Subprocessos também podem ter outros Subprocessos.

Multiprocessing é melhor que **Threading** então?!

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing

- Para implemen o Python **multiprocessin** ***process-based*** através da insta **s**.
- Subprocessos s **vínculo hierárquico**, ou é chamado de pai e, novos proces
- Subprocessos t s Subprocessos.

Multiprocessing é melhor que **Threading** então?!

Multiprocessing

- A criação e gerência de processos e **subprocessos** é sempre uma **operação custosa** para qualquer sistema operacional.

- Diferente das **threads**, é preciso alocar **contexto de hardware**, **contexto de software** e **espaço em memória, exclusivos para cada um**.

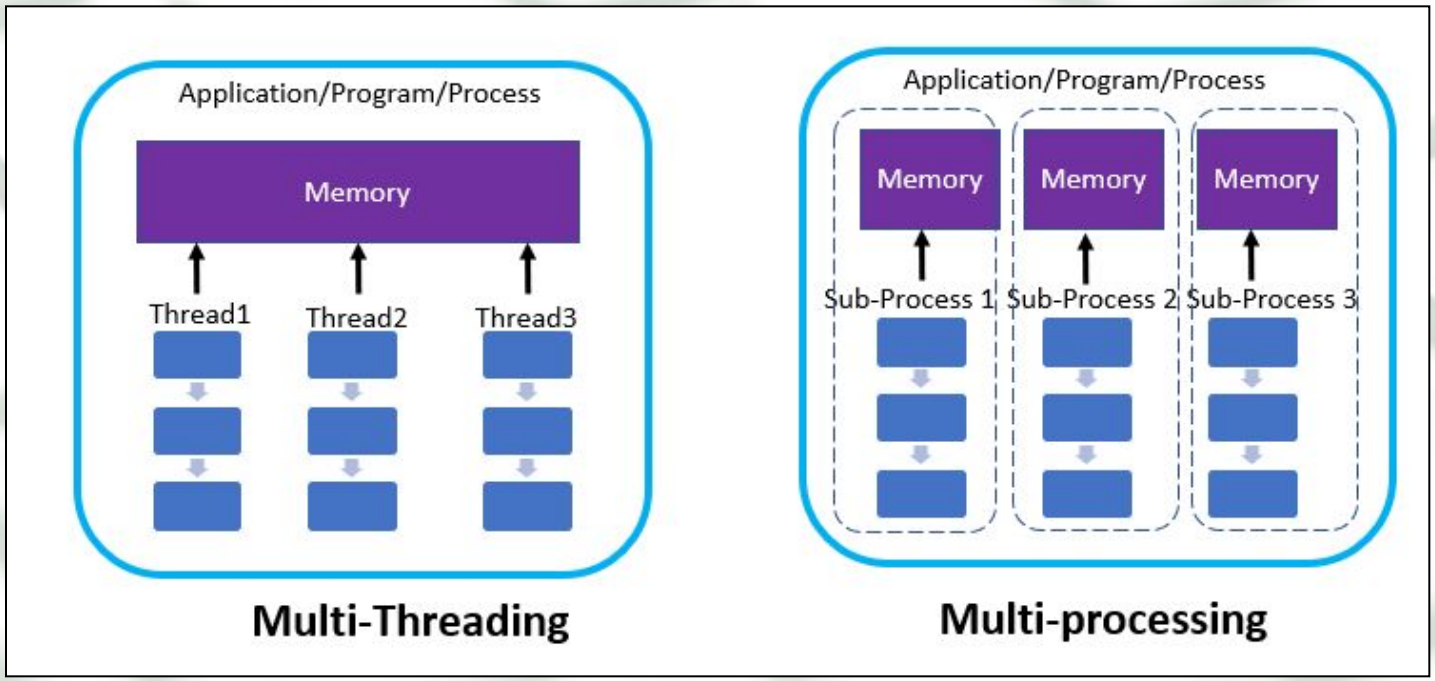
- Lembram do **PCB**?!

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing

Outro detalhe importante...

- Processos (e subprocessos) **não compartilham memória entre si**.



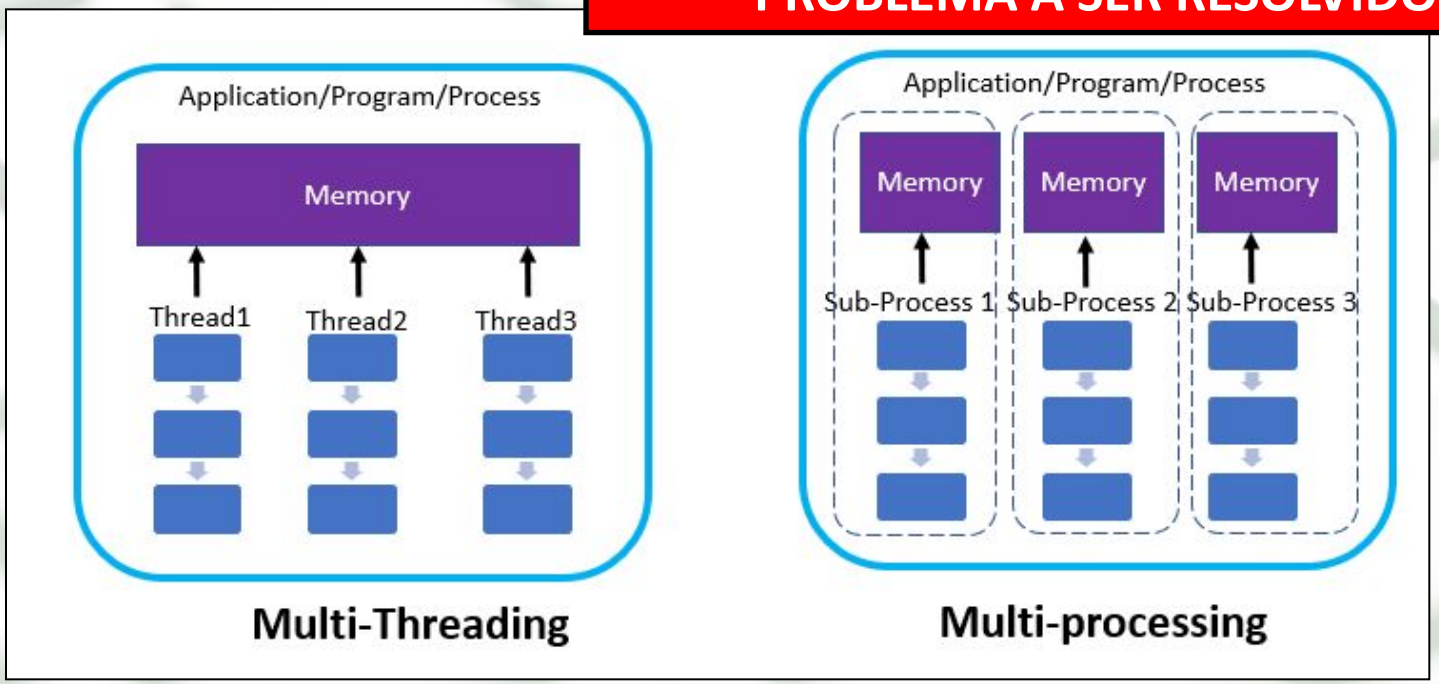
INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing

Outro detalhe importante...

- Processos (e subprocessos) **não compartilham memória entre si**.

COMUNICAÇÃO INTER-PROCESSOS É UM PROBLEMA A SER RESOLVIDO!



Laboratório #04.01

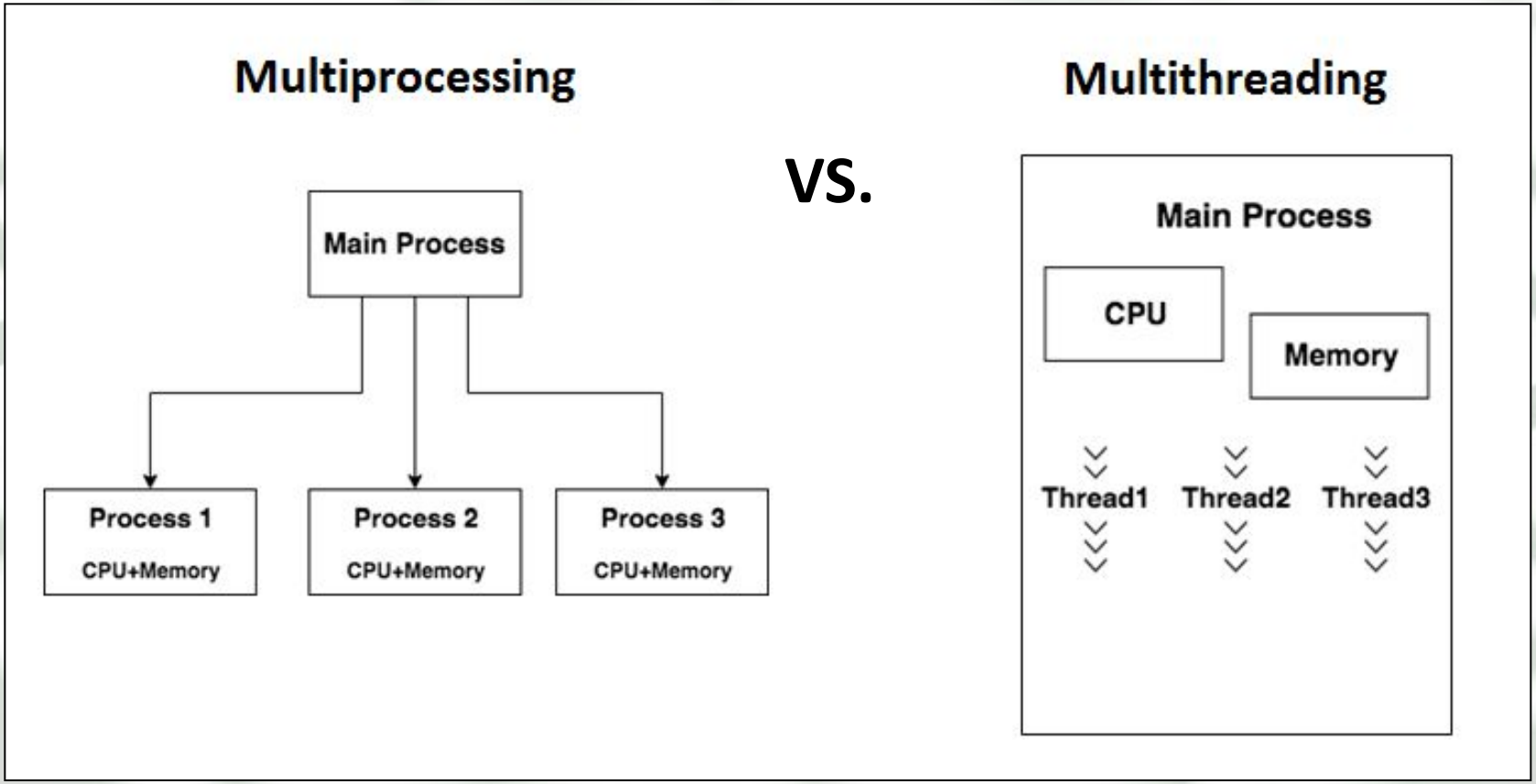
- Implemente uma aplicação que crie um subprocesso (módulo *multiprocessing*) que altera o conteúdo de uma lista enviada por argumento no Python.

- Altere o módulo para *threading* e veja se algo muda.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing

- *Mas, e ai? É Vantagem ou Desvantagem? Qual escolher?*



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing

- *Mas, e ai? É Vantagem ou Desvantagem? Qual escolher?*

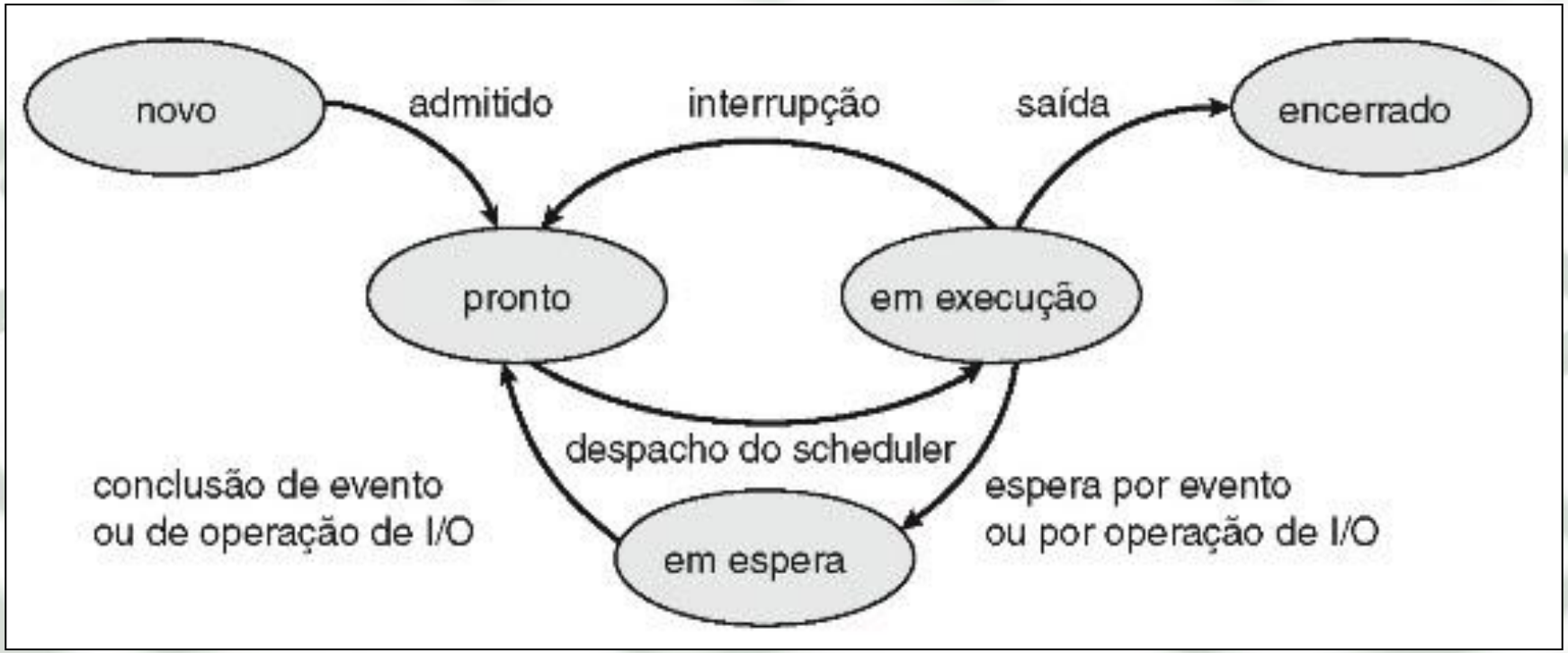
Multiprocessing

- *Mas, e ai? É Vantagem ou Desvantagem? Qual escolher?*

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Estados de um Processo

- *Lembram dos estados de um Processo???*

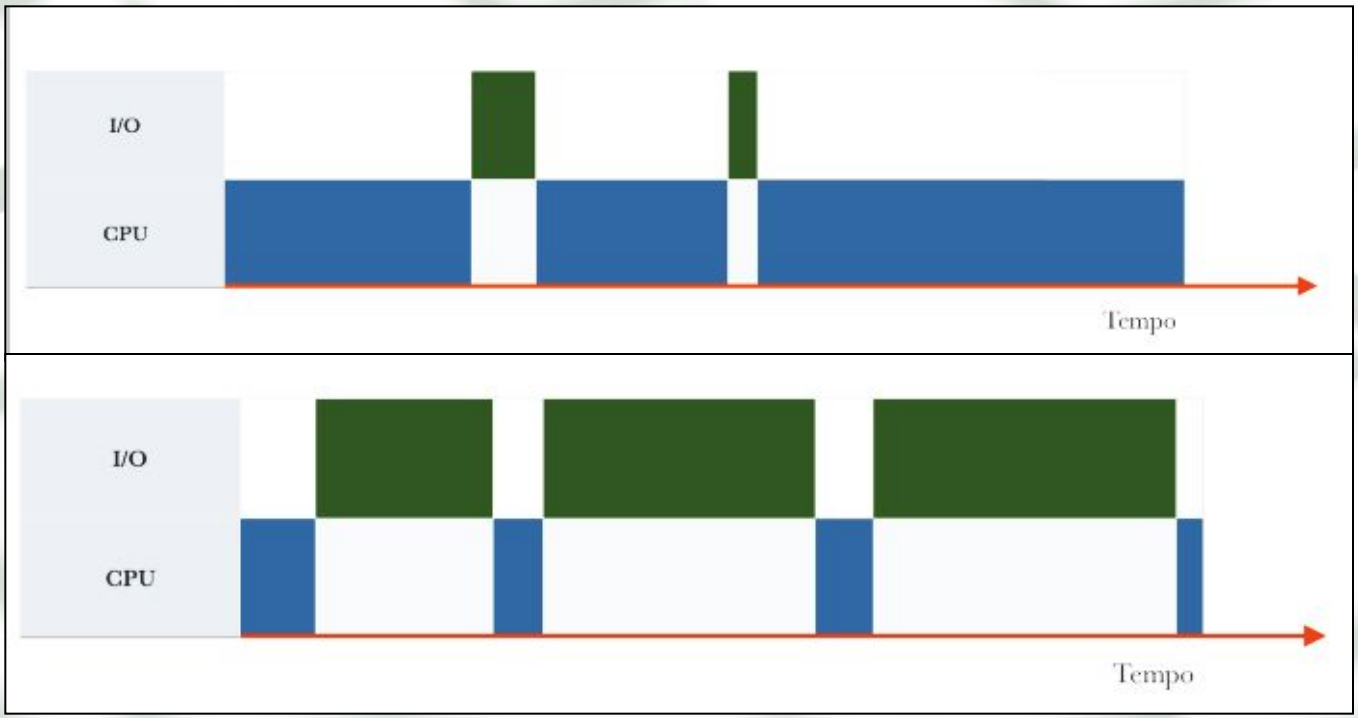


CPU Bound vs. I/O Bound

- Processos podem ser classificados como:
  - **CPU Bound**
    - Termo utilizado para designar processos que fazem uso intensivo da CPU, ou seja, **passa a maior parte do tempo em estado de execução**.
  - **I/O Bound**
    - Termo utilizado para designar processos que, comparativamente, passa a maior parte do tempo no estado de espera, pois **realiza um elevado número de operações de Entrada e Saída (I/O)**.

CPU Bound vs. I/O Bound

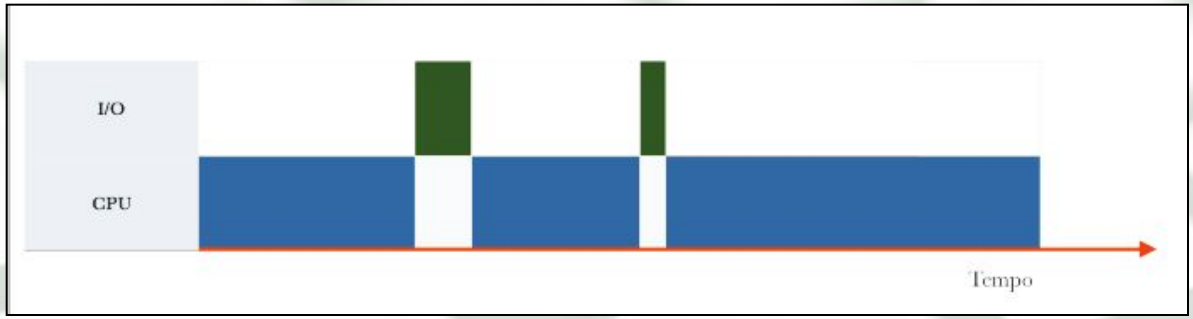
- Saber distinguir as duas características é muito importante para desenvolver adequadamente a solução de um problema...



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

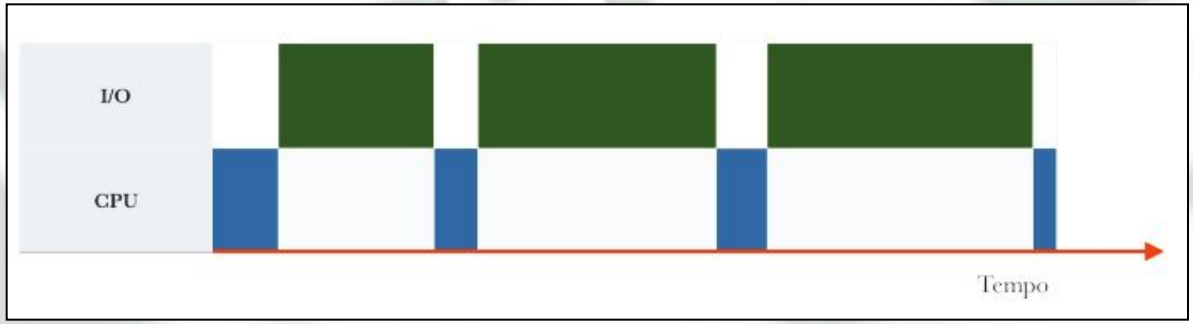
CPU Bound vs. I/O Bound

- Custos vs. Benefícios... **O que você escolheria**?



Concorrência (Threading)?

Paralelismo (Multiprocessing)?



Concorrência (Threading)?

Paralelismo (Multiprocessing)?

Além disso…

- **Temos mais probleminhas para resolver…**
  - **Sincronização**
    - Mecanismos de Coordenação das ações entre Threads/Processos concorrentes.

JÁ ESTUDAMOS ✓

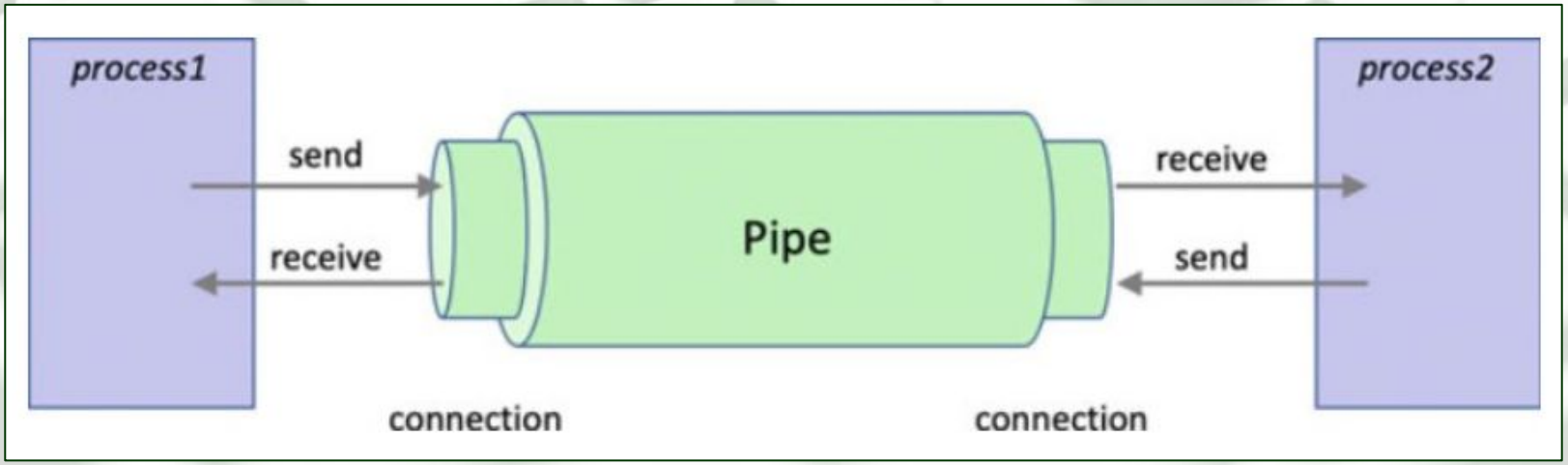
  - **Comunicação**
    - **Mecanismos para troca de mensagens entre os Processos.**

Comunicação entre Processos

- **IPC (*Inter-Process Communication*)**
- Principais Técnicas:
  - **Sistemas Centralizados**
    - Pipes
    - Queues
  - **Sistemas Distribuídos**
    - Sockets
    - RPC

Pipe

- **Pipe** *(tradução: Cano)* constitui um modelo de comunicação básico e unidirecional entre processos.
- Consiste na criação (pelo S.O.) de dois *endpoints* aos quais dois processos se conectam para enviar e receber dados de forma síncrona.



INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Pipe

- **Pipe** *(tradução: Cano)* constitui um modelo de comunicação básico e unidirecional entre processos.
- Consiste na criação (pelo S.O.) de dois *endpoints* aos quais dois processos se conectam para enviar e receber dados de forma

Exemplo de Pipe implementado pelo próprio Shell do Linux

```
Arquivo  Editar  Ver  Pesquisar  Terminal  Ajuda
adriano@adriano-notebook:~$ ls
 a.out                 Downloads      Modelos          snap          Vídeos
'Área de Trabalho'     Dropbox        Música           teste         wget-log
 Desktop               Imagens        Público          teste.c
 Documentos            missfont.log   PycharmProjects  teste.tar.gz
adriano@adriano-notebook:~$ ls | grep *.c
teste.c
```

processo a => “ls”

processo b => “grep”

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing Pipe

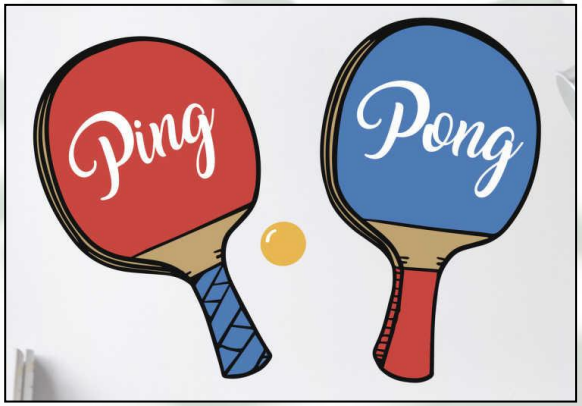
```
conn_a, conn_b = multiprocessing.Pipe()
```


```
conn_a.send('Olá Mundo')
conn_b.recv()
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
conn_b.send('Até Mais')
conn_a.recv()
```

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório #04.02

- **Laboratório Pipe** - Problema *Ping-Pong*
  - Faça um programa que crie dois subprocessos, um chamado PING e outro chamado PONG.
  - Cada um dos processos, deve imprimir a mensagem recebida pelo Pipe e responder a palavra 'PING' ou 'PONG' (respectivamente).
  - **Ping-Pong** deve ocorrer até o usuário teclar `Enter`.



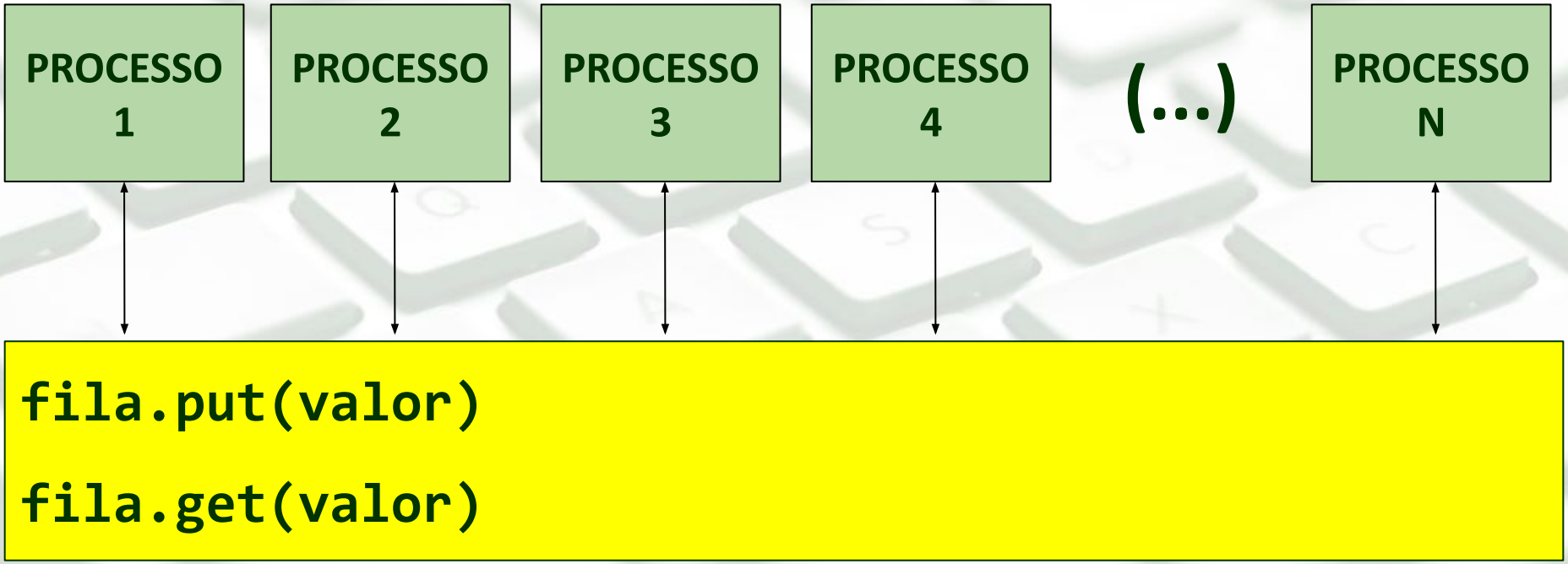
Análise...

- Imagine o problema Produtor-Consumidor visto anteriormente, contudo, agora o **Consumidor é um CPU-BOUND.**
- Nesta hipótese, seria interessante trabalhar com o módulo **`multiprocessing`**.
- Mas como fazer a comunicação **Produtor ⇔ Consumidor**?
- Pipes possuem apenas 2 endpoints...
  - **Vários produtores e vários consumidores?**
  - **Criar um Pipe para cada processo?**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing Queue

```
fila = multiprocessing.Queue(MAX_BUFF)
```

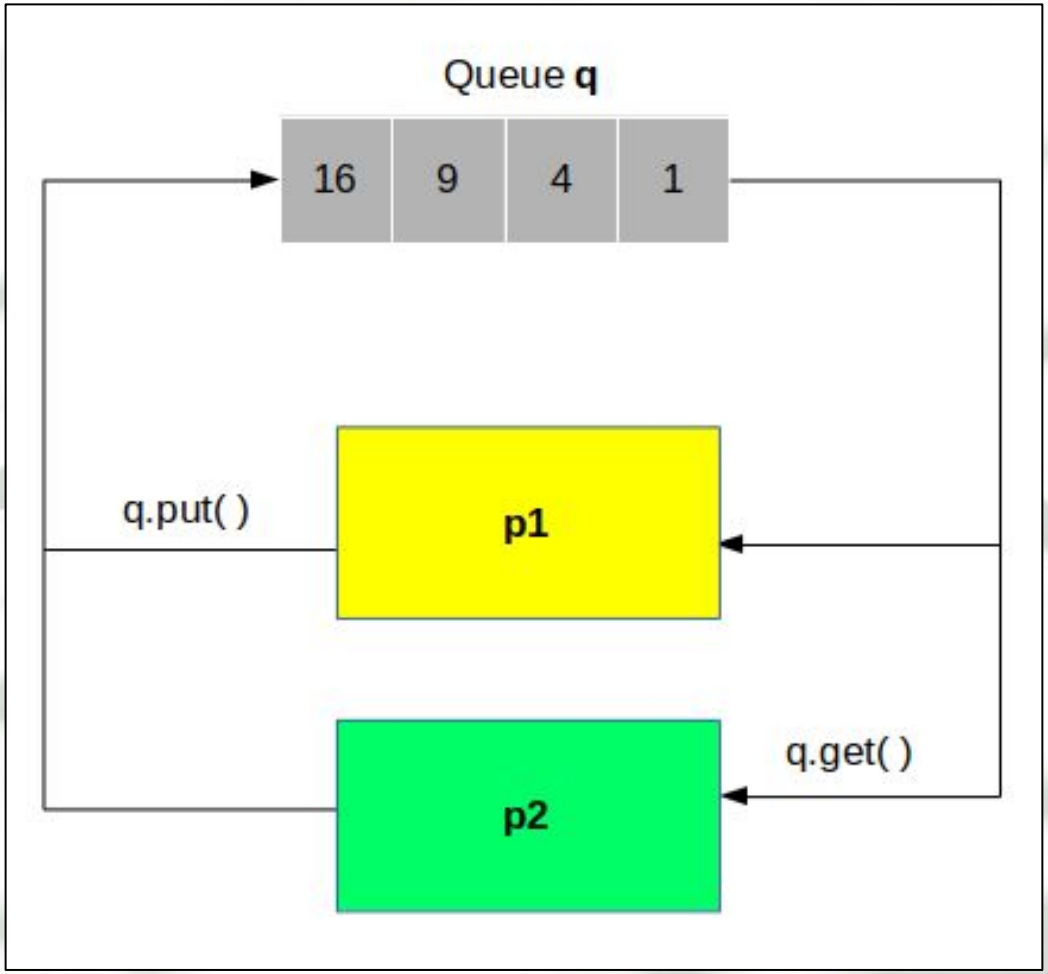



```
fila.put(valor)

fila.get(valor)
```

Multiprocessing Queue

A classe `multiprocessing.Queue()` é **CONCURRENT-SAFE**

Isso quer dizer que a própria classe já implementa os mecanismos de controle de acesso à seção crítica nos métodos PUT e GET.

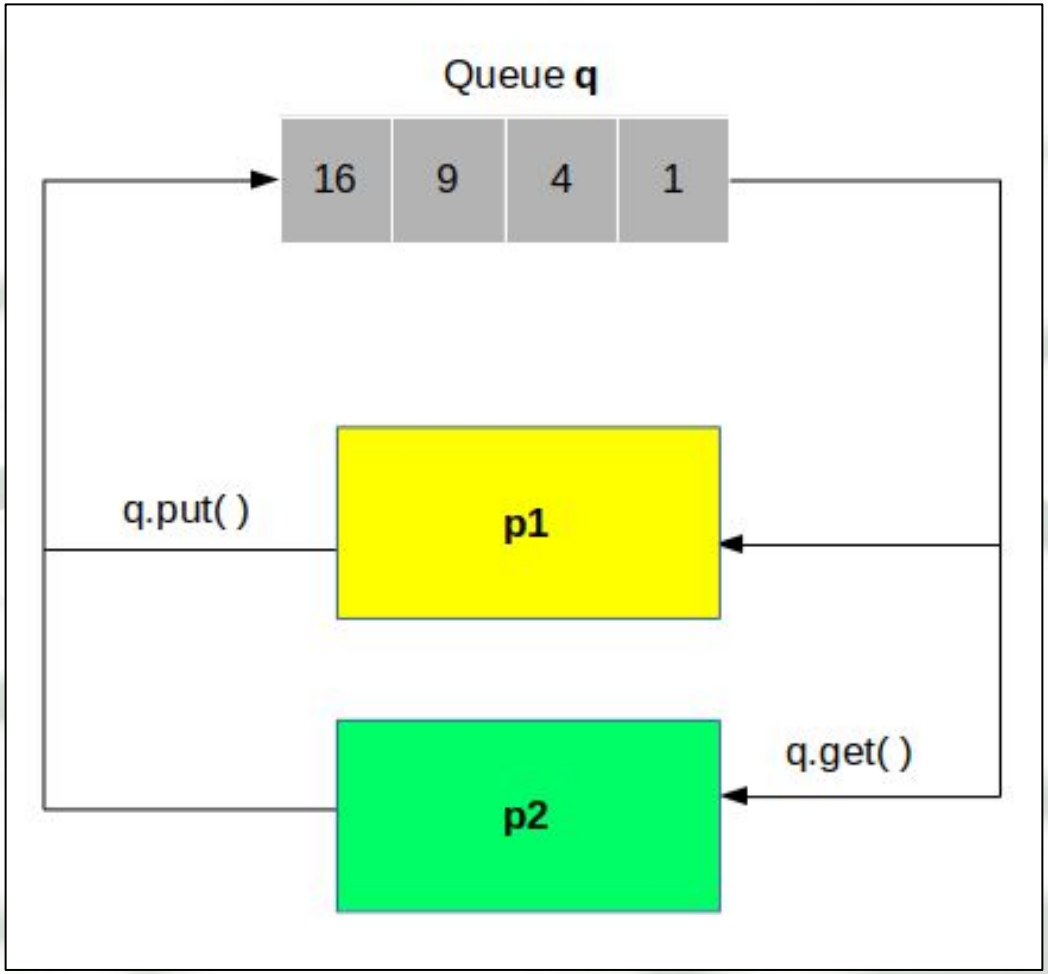


INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Multiprocessing Queue

A classe `multiprocessing.Queue()` é **CONCURRENT-SAFE**

Portanto, a classe Queue( ) encapsula toda a complexidade inerente ao problema de Exclusão Mútua no acesso à Seção Crítica.



Laboratório #04.03

- **Laboratório Queue**
  - Faça um programa que crie um processo produtor, que alimente uma Queue limitada em 4 mensagens, com valores sequenciais (sleep de 1 segundo / produção).
  - Implemente 03 processos consumidores que captam valores, aguardam 10 segundos (simulação CPU-Bound) e imprime o valor coletado e sua raiz quadrada.
  - Deverão ser produzidos 10 valores no total.
  - Imprima “FINALIZADO” ao final da execução.
  - Analise o resultado.

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

Laboratório #04.03

- **Laboratório Queue**
  - Faça um programa
    alimente uma Que
    valores sequencia
  - Implemente 03 pr
    valores, aguardam
    imprime o valor c
  - Deverão ser produ
  - Imprima “FINALIZ
  - Analise o resultad

```
Valor Produzido: 1
Consumidor 3 coletou: 1 => result: 1.0
Valor Produzido: 2
Consumidor 2 coletou: 2 => result: 1.41
Valor Produzido: 3
Consumidor 1 coletou: 3 => result: 1.73
Valor Produzido: 4
Valor Produzido: 5
Valor Produzido: 6
Valor Produzido: 7
Valor Produzido: 8
Consumidor 3 coletou: 4 => result: 2.0
Valor Produzido: 9
```

Laboratório #04.04

- **Laboratório Joinable Queue**
  - Faça um programa que implemente 2 Queues:
  - **Queue de Tarefas** (alimentada com todos os números ímpares existentes com 06 dígitos).
  - Implemente **`os.cpu_count()` subprocessos** para, a partir da Queue de Tarefas, processe e alimente uma **Queue de Resultados**.
  - A Queue de Resultados deve conter todos os números primos de 6 dígitos existentes na Queue de Tarefas.
  - **Após término da Queue de Tarefas, imprimir toda a Queue de Resultados.**

INSTITUTO FEDERAL
Norte de Minas Gerais
Campus Januária

- **Laboratório Joinable Qu**
  - Faça um programa que
  - **Queue de Tarefas** (alim
    ímpares existentes com
  - Implemente **os.cpu_c**
    partir da Queue de Tare
    Queue de Resultados.
  - A Queue de Resultados
    primos de 6 dígitos exis
  - **Após término da Queu**
    Queue de Resultados.

```
998861
998917
998947
998957
999007
999029
999067
999091
999133
999199
999233
999331
999529
999563
999683
999769
999853
999907
999917
999931
999959
FINALIZADO! 68906 números primos encontrados.
```