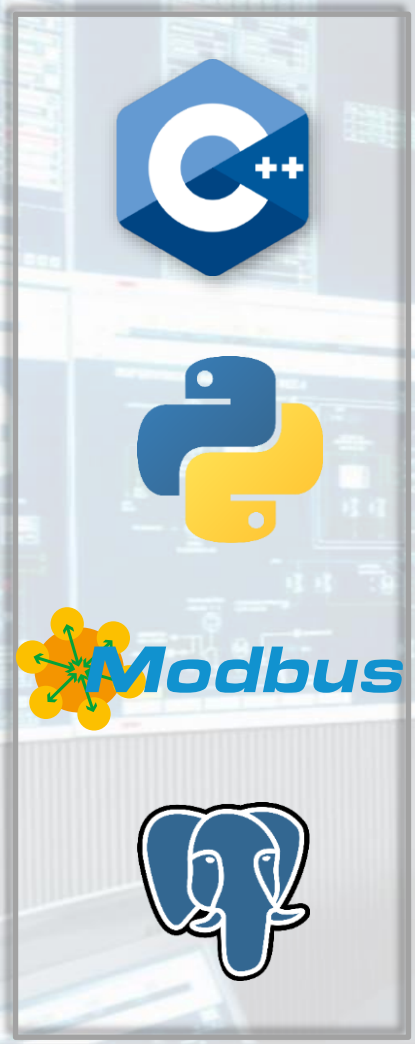



# Informática Industrial

## Introdução à Programação Concorrente

Prof. Guilherme Márcio Soares, Dr. Eng.
guilherme.marcio@ufjf.edu.br

ufjf
Universidade Federal de Juiz de Fora

# Processos e Threads

- A **execução** de um determinado **programa** de computador demanda **um conjunto de recursos**.
- Um **processo** é um programa em execução, que necessita de vários recursos, dentre eles um código binário, registradores, uma pilha (stack) e o *program counter*.
- Cada processo no computador é executado de **maneira independente**, possuindo o seu próprio conjunto de recursos.

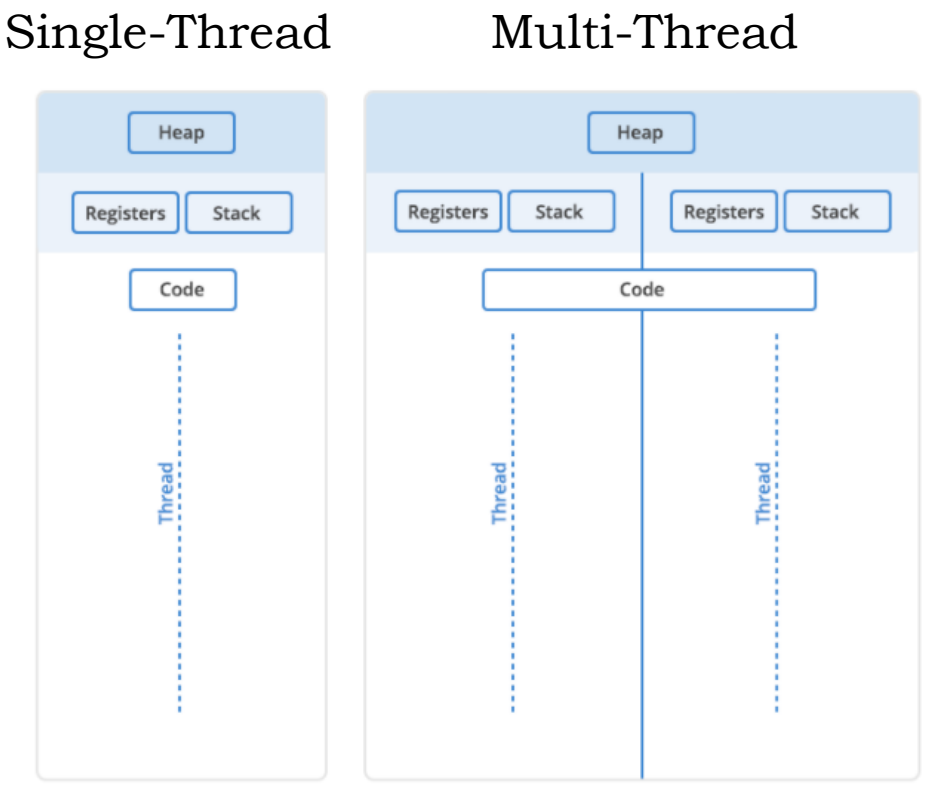
Um processo de computador


Fonte: https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/

# Processos e Threads

- Uma **thread** é um fluxo de execução independente dentro de um processo.
- Cada processo possui **no mínimo 1 thread**, mas pode possuir mais.
- Cada thread possui seus próprios registradores e stacks, mas outros recursos são **compartilhados** entre threads de um mesmo processo.
- O **compartilhamento de recursos** e variáveis é mais fácil e rápido entre threads do que entre processos.

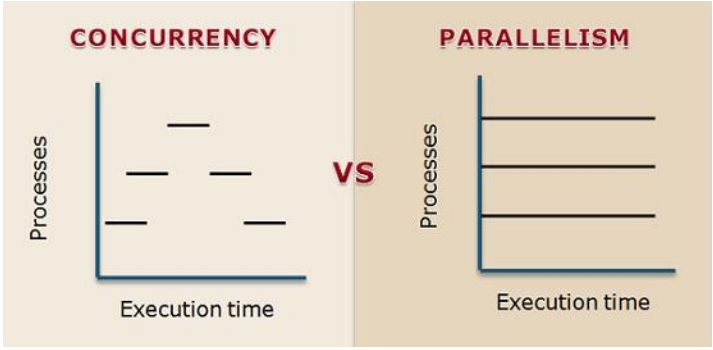
# Processos e Threads




Fonte: https://www.backblaze.com/blog/whats-the-diff-programs-processes-and-threads/

# Concorrência e Paralelismo

- Em computação, **concorrência** é a execução de diversas tarefas dividindo um mesmo recurso (CPU).
- O desenvolvimento de **programas concorrentes** permite um **uso melhor** do **hardware** disponível no computador.
- No **paralelismo** as tarefas são executadas **ao mesmo tempo**.
- O **paralelismo** permite **aumentar** a **velocidade** em que uma tarefa é executada.

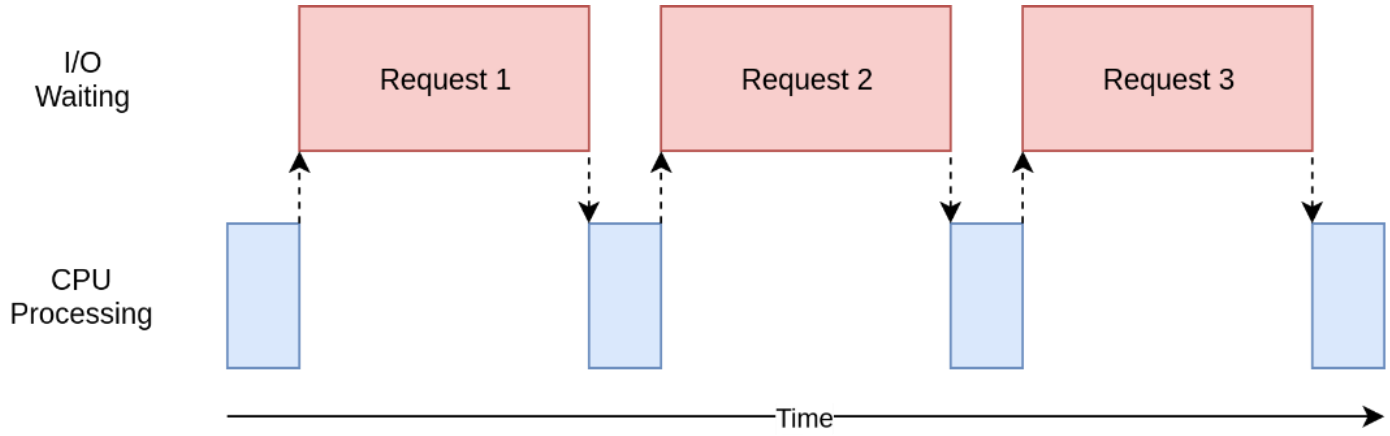



Fonte: https://techdifferences.com/difference-between-concurrency-and-parallelism.html

# Concorrência e Paralelismo

- Para entender quando deve-se utilizar a concorrência e o paralelismo, é importante entender os dois tipos básicos de problemas em computação:

- **Programas com muitas operações de E/S (I/O-bound):** são programas que demanda grande uso de operações de entrada e saída, como gravação de arquivos, comunicação em rede e atendimento a usuários.

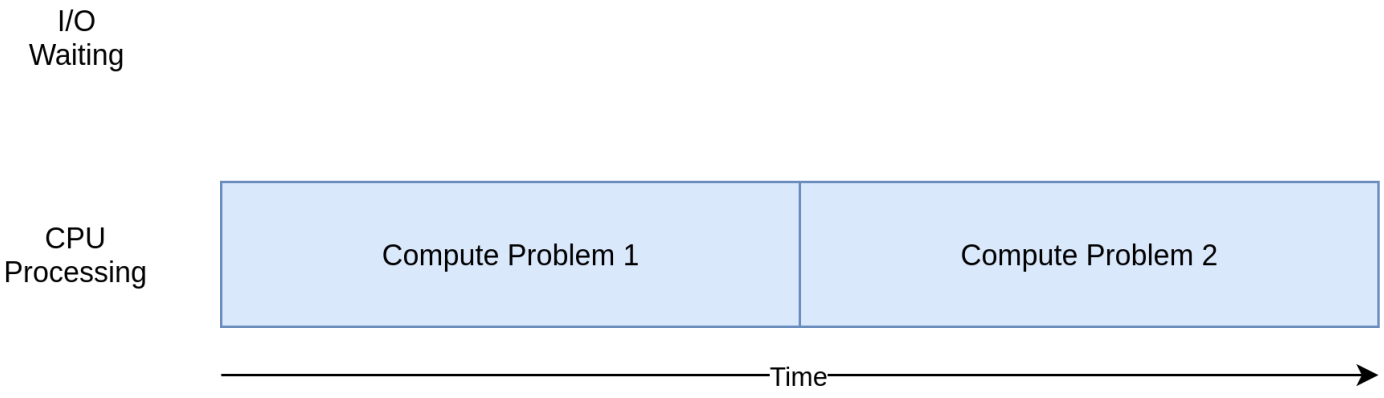



Fonte: https://realpython.com/python-concurrency/

# Concorrência e Paralelismo

- Para entender quando deve-se utilizar a concorrência e o paralelismo, é importante entender os dois tipos básicos de problemas em computação:

- **Programas computacionalmente intensivos (CPU-bound):** programas com alta demanda da CPU, como realização de cálculos matemáticos, etc.




Fonte: https://realpython.com/python-concurrency/

# Concorrência em Python

- Em Python, existem basicamente 3 módulos que permitem a criação de programas concorrentes ou paralelos:

| Módulo | Tipo de Concorrência | Nº CPUs | Mais indicado para |
|---|---|---|---|
| **Multithreading** | **Multitarefa preemptivo** | **1** | **I/O-bound** |
| asyncio | Multitarefa cooperativo | 1 | I/O-bound |
| Multiprocessing | Multiprocessos | Vários | CPU-bound |

Deve-se analisar o tipo de aplicação antes de escolher a melhor opção. Dependendo da escolha o desempenho da aplicação pode até diminuir.

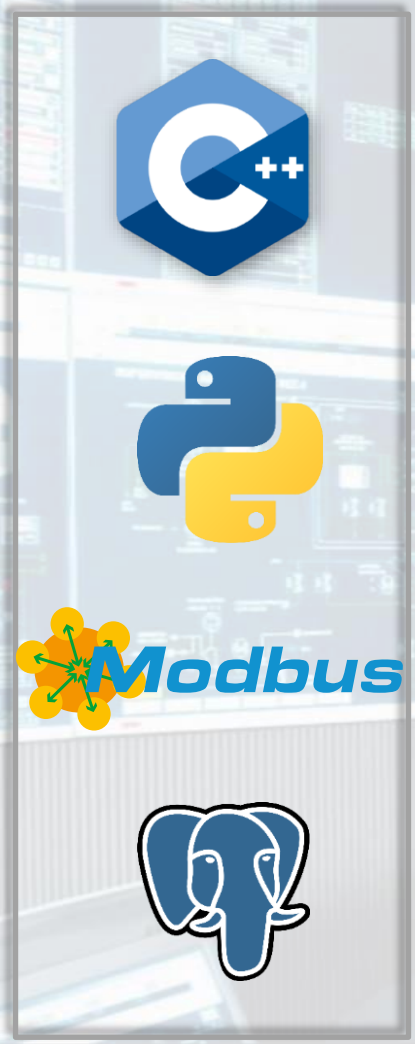
# Módulo Multithreading

- **Multitarefa preemptivo:** o sistema operacional decide quando executar uma thread ou outra com base em seu algoritmo de escalonamento.

- Adequado para problemas do tipo **IO-bounds**, que são muito comuns em Informática Industrial.

- Documentação:

https://docs.python.org/3/library/threading.html#module-threading




# Informática Industrial

## Introdução à Programação Concorrente: Mecanismos de Sincronismo

Prof. Guilherme Márcio Soares, Dr. Eng.
guilherme.marcio@ufjf.edu.br

UFJF
Universidade Federal de Juiz de Fora

# Compartilhamento de Recursos

- Muitas vezes as diversas **threads** de um programa podem compartilhar recursos;

- Dependendo das características do programa, estas diversas linhas de execução podem tentar acessar um recurso de forma simultânea e possivelmente pode **corrompê-lo** ou fazer com que o programa não se comporte da maneira esperada.

# Compartilhamento de Recursos

- Desta forma é necessário **proteger recursos compartilhados**.
- Códigos **thread-safe** garantem a **consistência** da execução em programas **multi-thread**.
- Existem diversas formas de se fazer sincronismo em sistemas concorrentes, tais como ***lock*** e **semáforos**. Estes mecanismos permitem tornar o objeto ou parte do código **thread-safe**.

# Seção Crítica

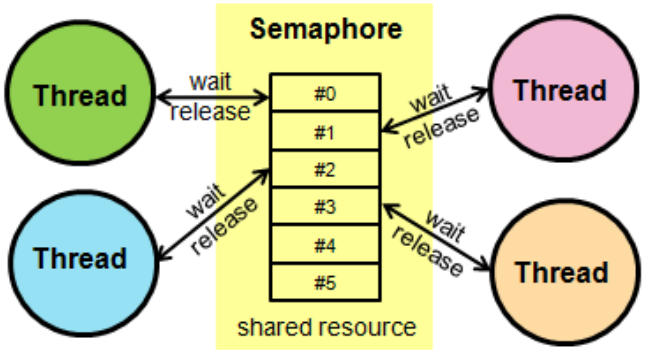
- Os compartilhados que devem ser protegidos podem ser agrupados em regiões chamadas **seções críticas**.
- Estas seções críticas devem possuir **comportamento atômico**, ou seja, as instruções não podem ser interrompidas por outra thread.

# Lock

- O objeto **lock** permite implementar o conceito de seção crítica.
- **lock.acquire** permite **bloquear** a seção crítica, de modo que as demais threads somente conseguirão entrar nela depois que o objeto estiver liberado.
- **lock.release** libera o recurso para que outras threads possam entrar na seção crítica.

# Semáforo

- O objeto **semaphore** permite o controle do número de threads ao objeto compartilhado.
- Normalmente são utilizados para proteger recursos com **capacidade limitada**.



# Tarefa

- Continue o desenvolvimento da última tarefa e mostre a capacidade do servidor em atender múltiplos clientes simultaneamente.