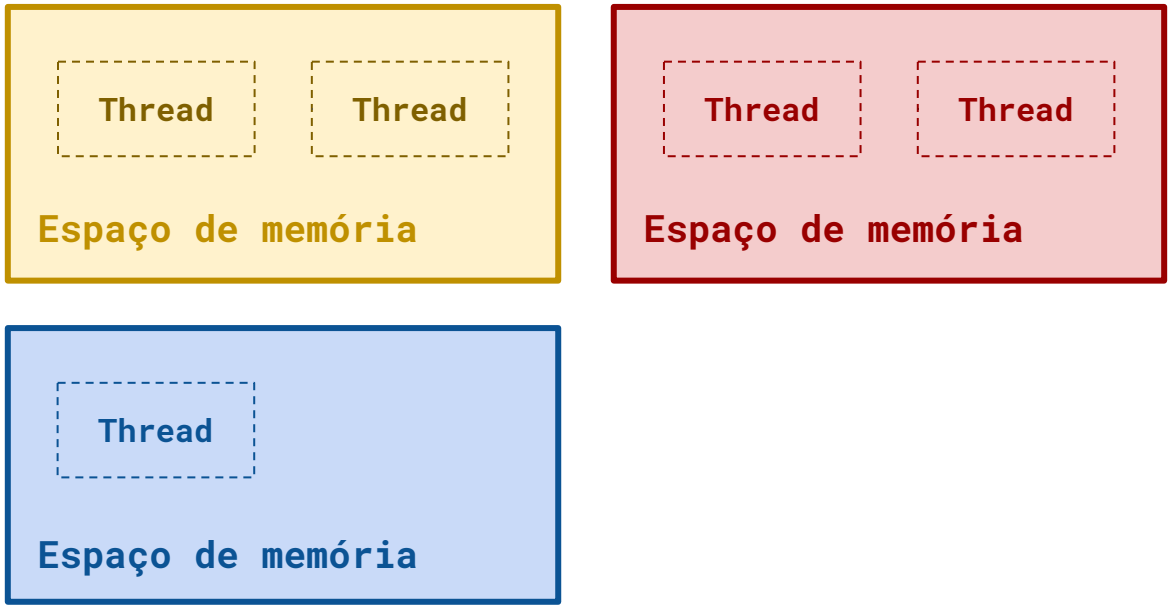
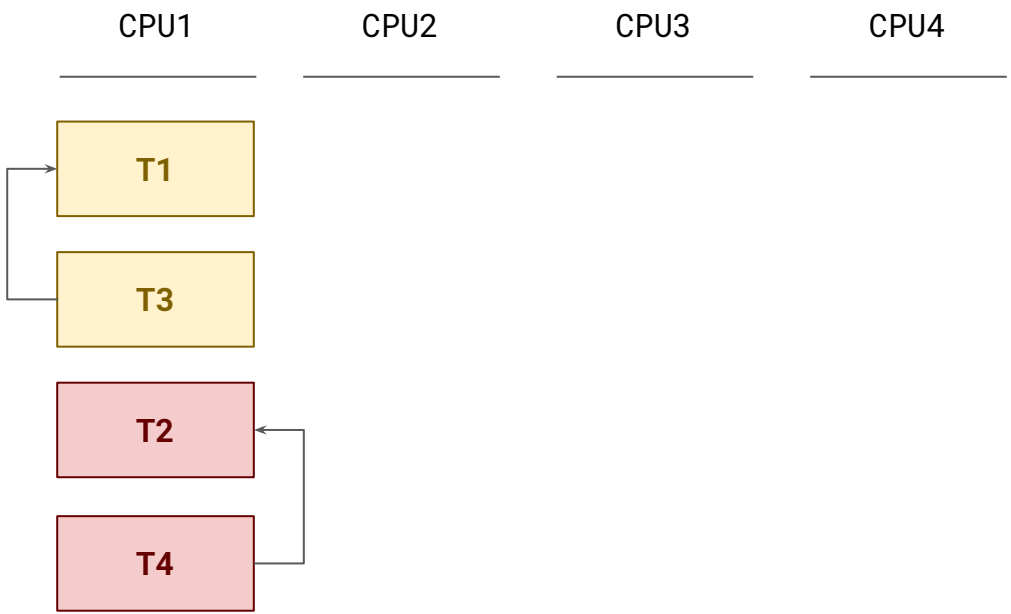
Paralelismo e Python

juliana.orod@gmail.com

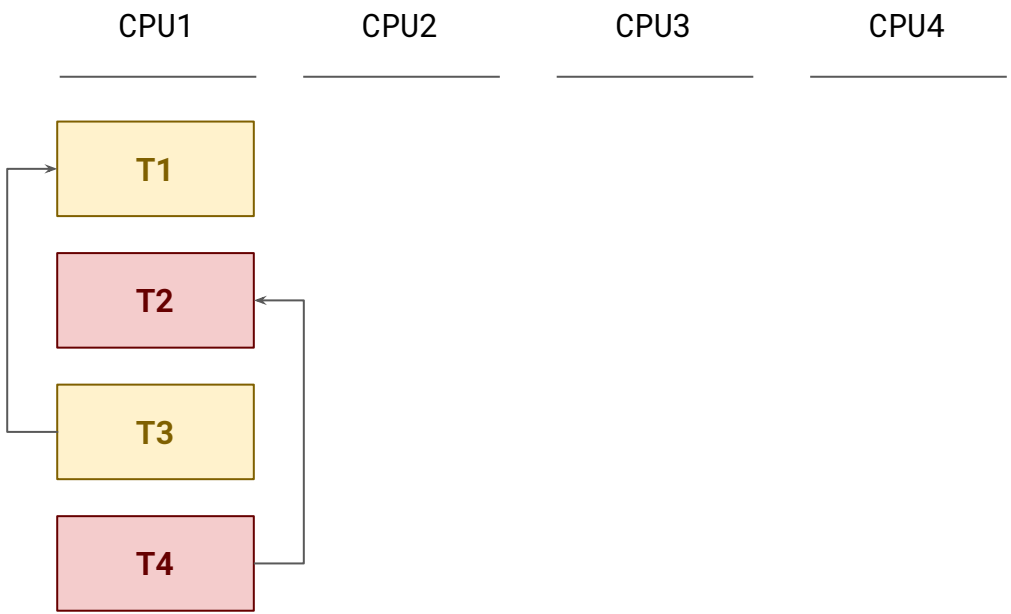
Threads e Processos



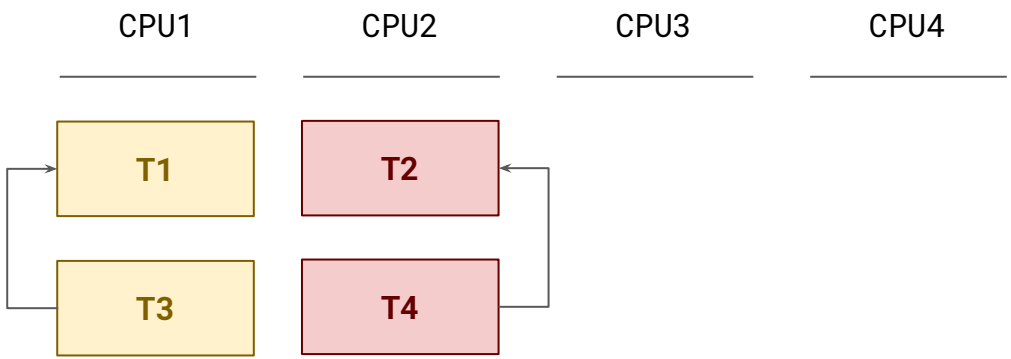
Paralelismo ou Concorrência



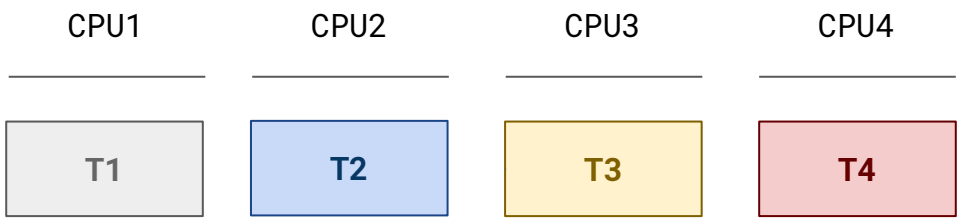
Paralelismo ou Concorrência



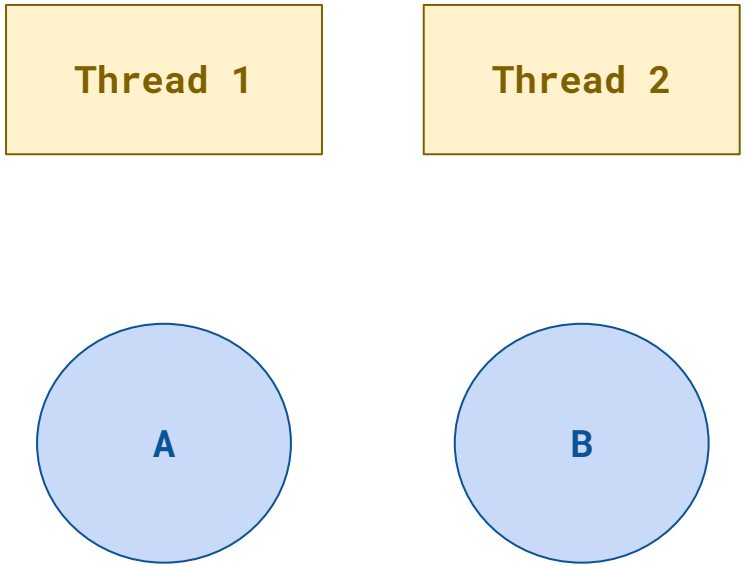
Paralelismo ou Concorrência



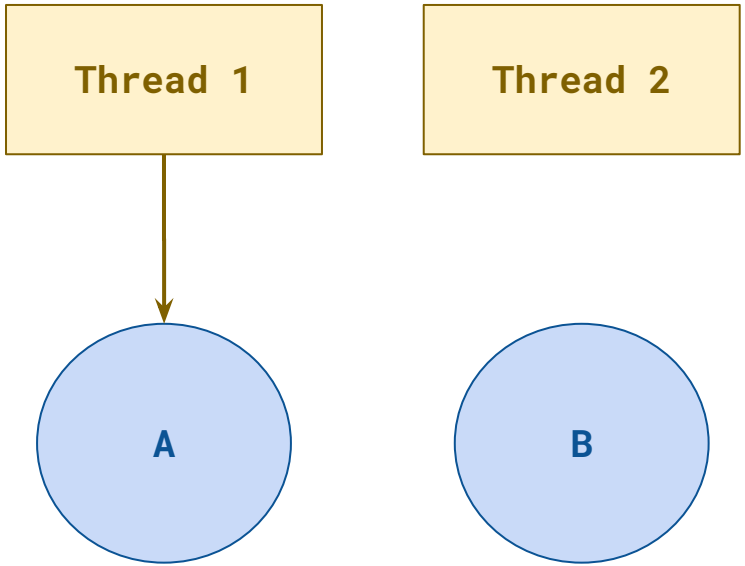
Paralelismo ou Concorrência



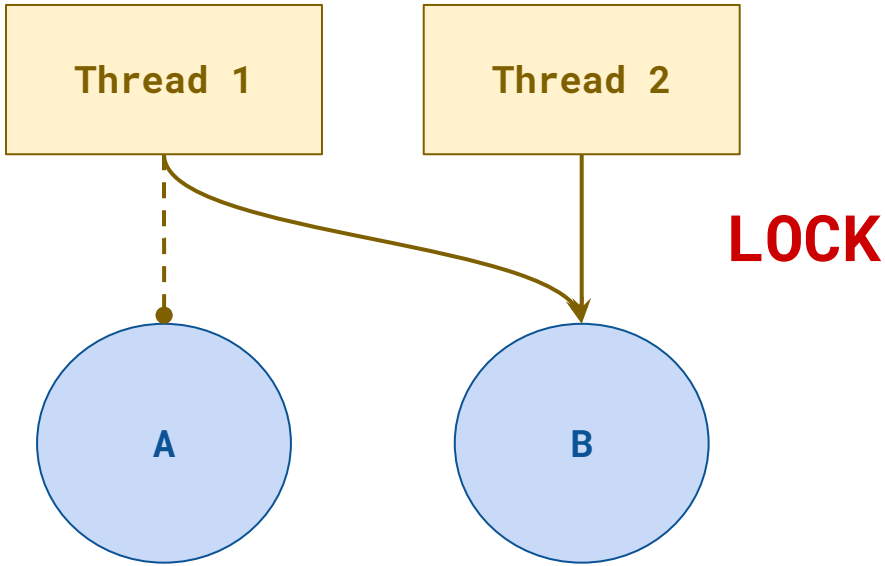
Locks e Deadlocks



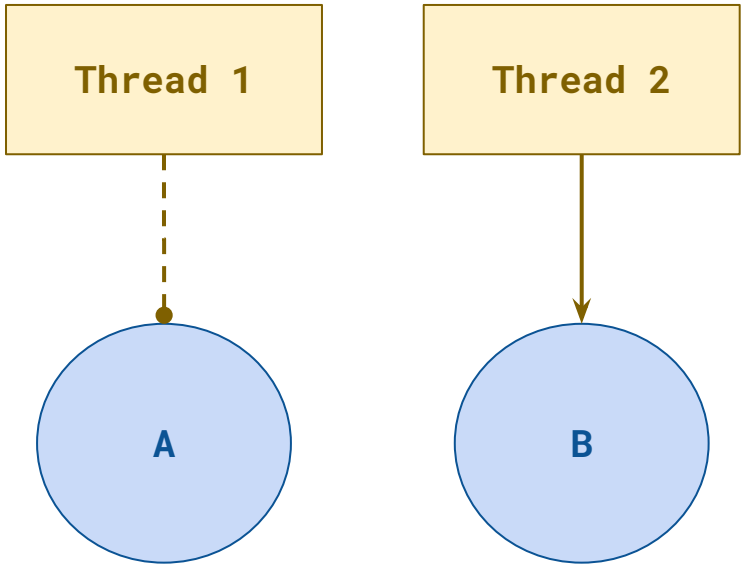
Locks e Deadlocks



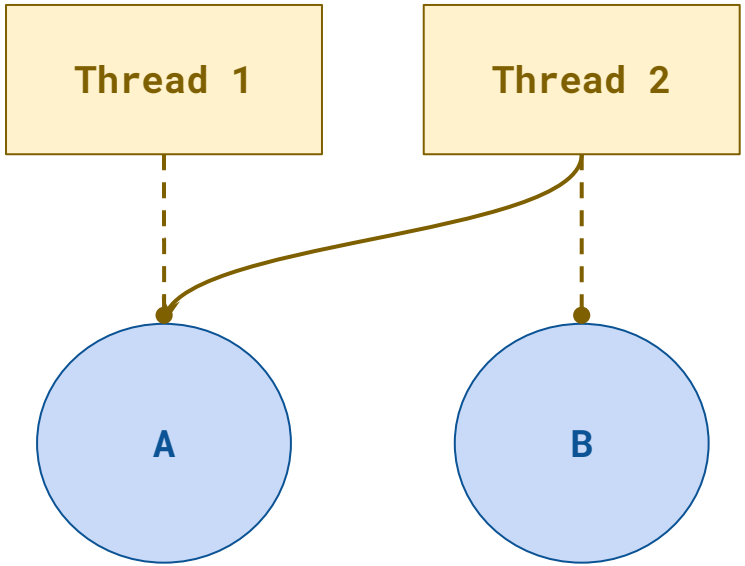
Locks e Deadlocks



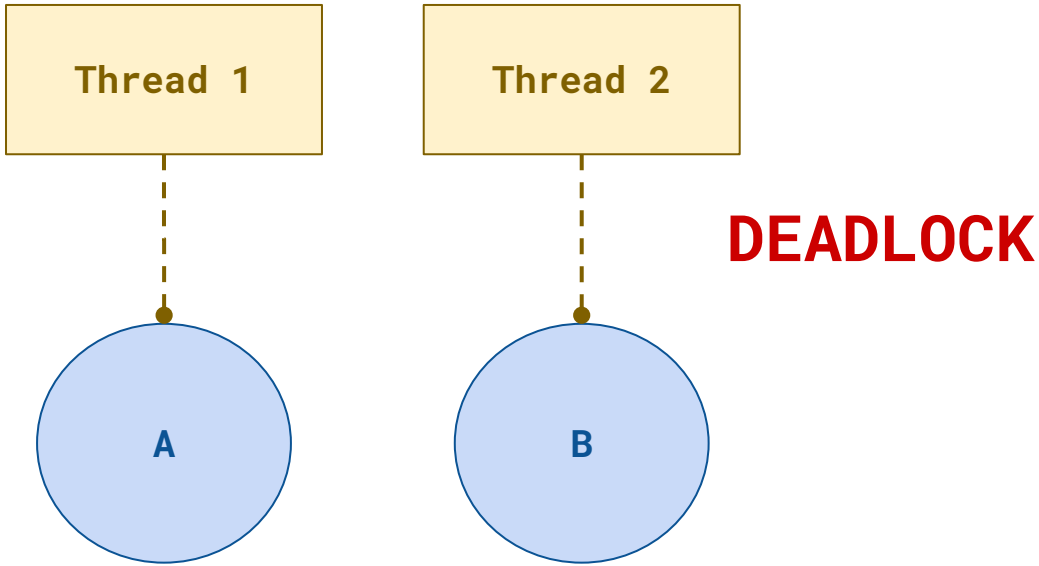
Locks e Deadlocks



Locks e Deadlocks

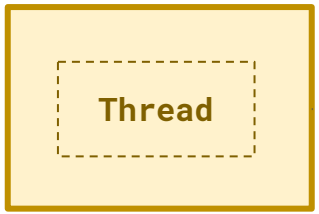


Locks e Deadlocks



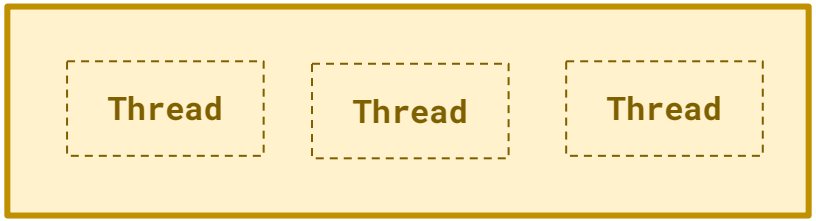
```
curl -s 0.0.0.0:5000/hello
{ “hello world!” }
```

app.run()



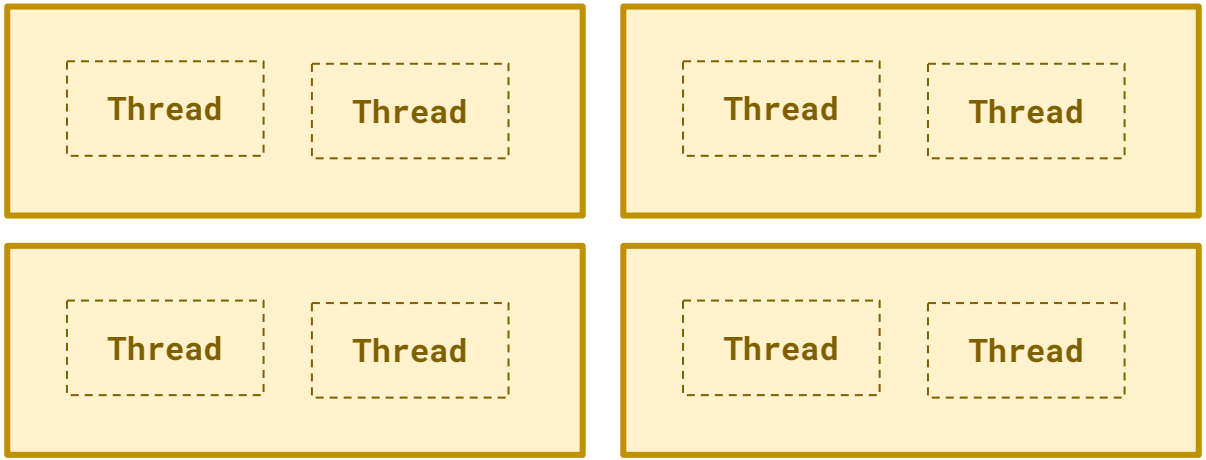
```
curl -s 0.0.0.0:5000/hello
{ “hello world!” }
```

```
app.run(threaded=True)
```



```
curl -s 0.0.0.0:5000/hello
{ “hello world!” }
```

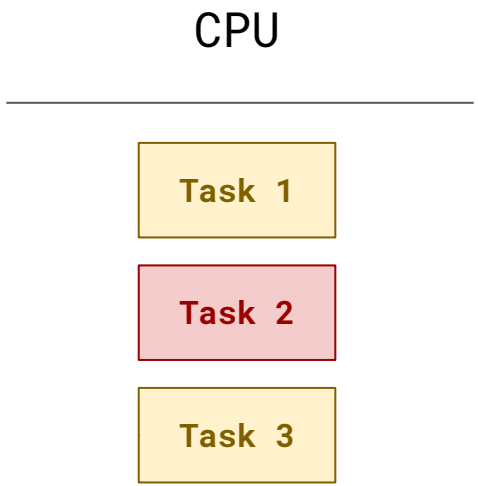
```
gunicorn server:app -w 4 --threads 2
```



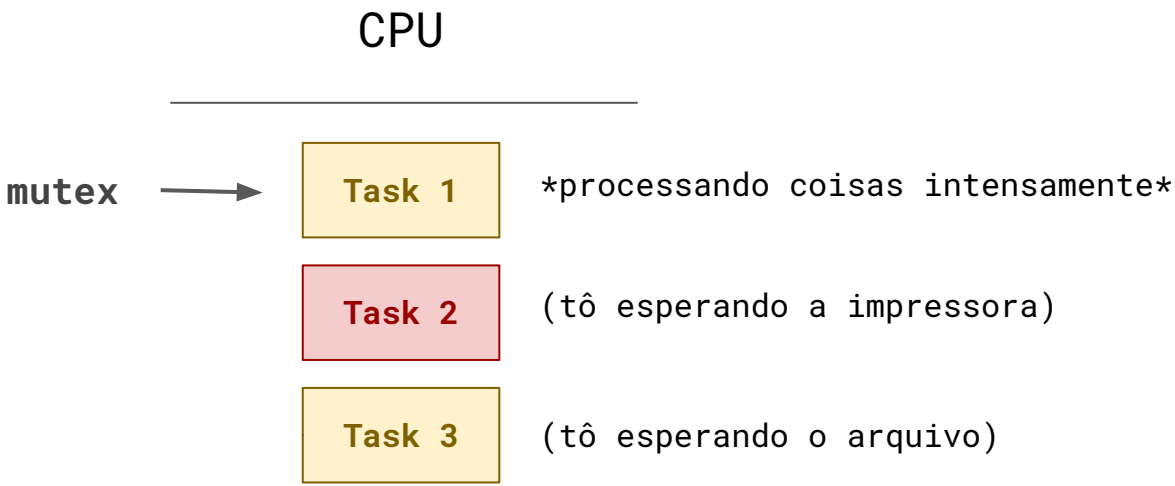
Voltando aos anos 90

CPU

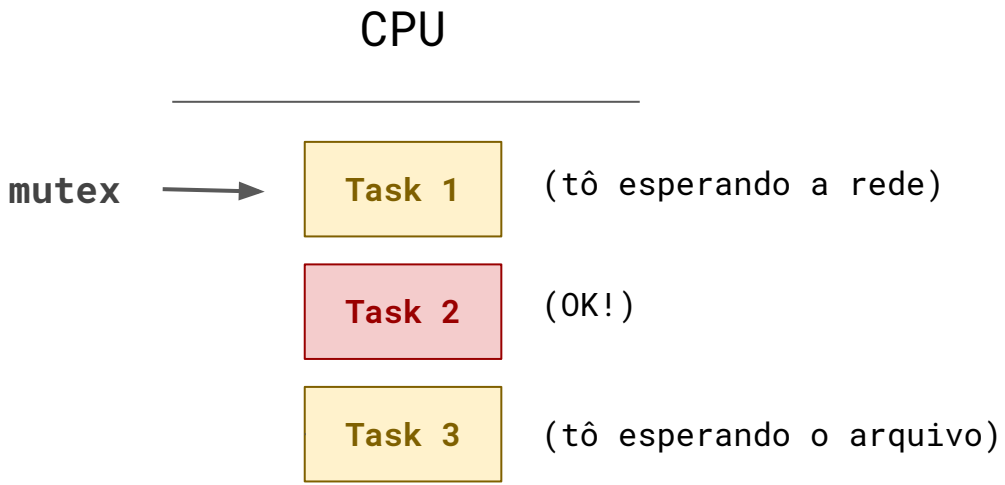
Voltando aos anos 90



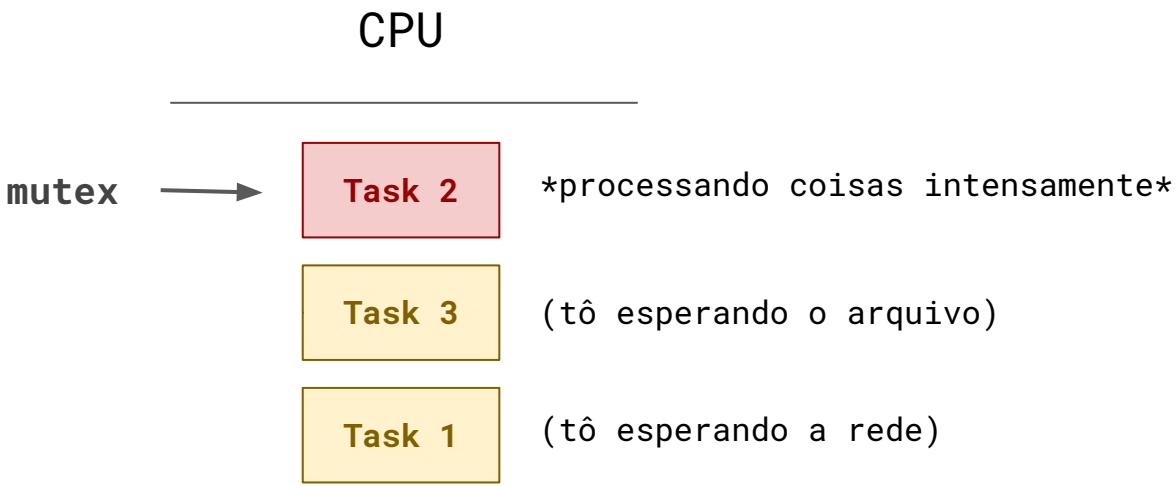
Voltando aos anos 90



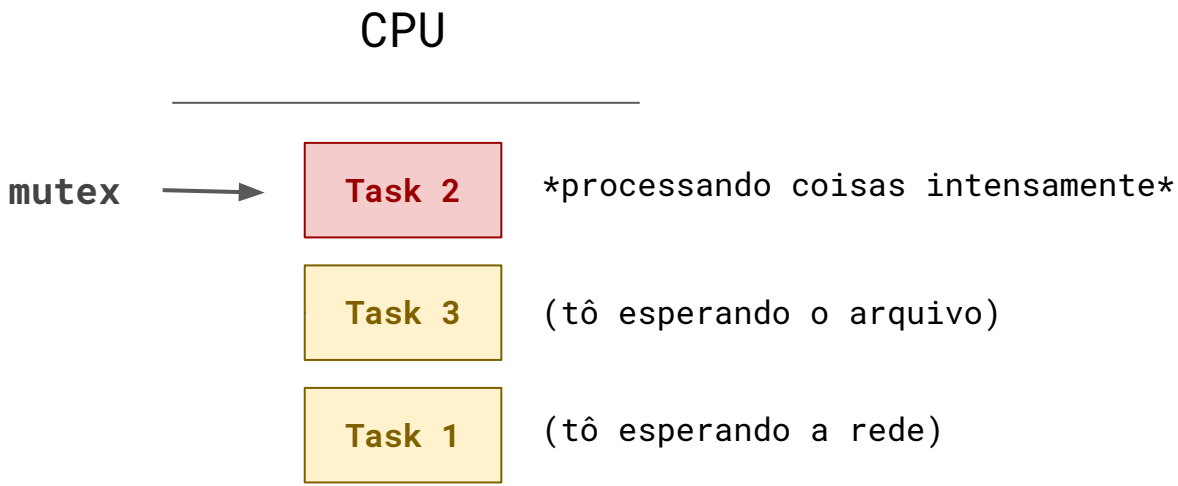
Voltando aos anos 90



Voltando aos anos 90



Voltando aos anos 90



{COOPERATIVE,PREEMPTIVE} MULTITASKING

Python roda em cima de um interpretador

Uma thread roda Python, enquanto N dormem ou esperam I/O

Python roda em cima de um interpretador

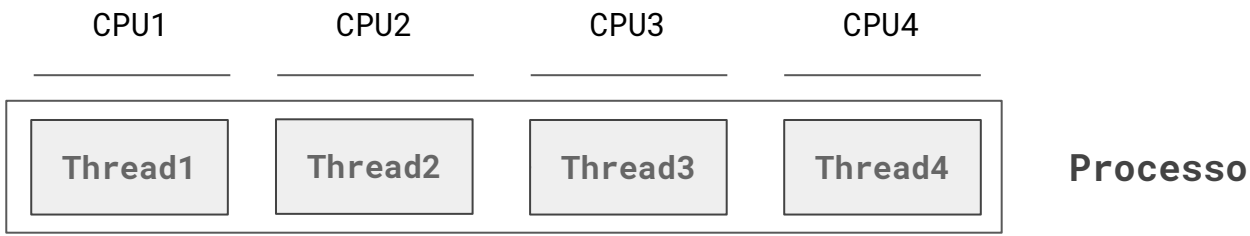
Uma thread roda Python, enquanto N dormem ou esperam I/O

GLOBAL INTERPRETER LOCK (GIL)

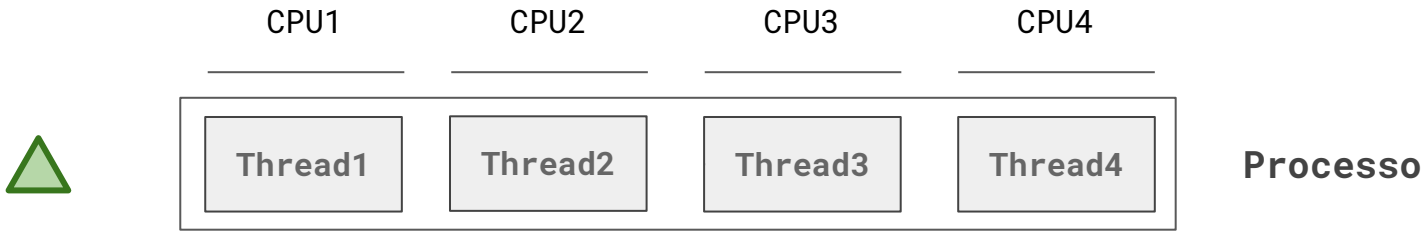
Python roda em cima de um interpretador... em C

```
#include "pythread.h"
#define ENTER_HASHLIB(obj) \
    if ((obj)->lock) { \
        if (!PyThread_acquire_lock((obj)->lock, 0)) { \
            Py_BEGIN_ALLOW_THREADS \
            PyThread_acquire_lock((obj)->lock, 1); \
            Py_END_ALLOW_THREADS \
        } \
    }
#define LEAVE_HASHLIB(obj) \
    if ((obj)->lock) { \
        PyThread_release_lock((obj)->lock); \
    }
```

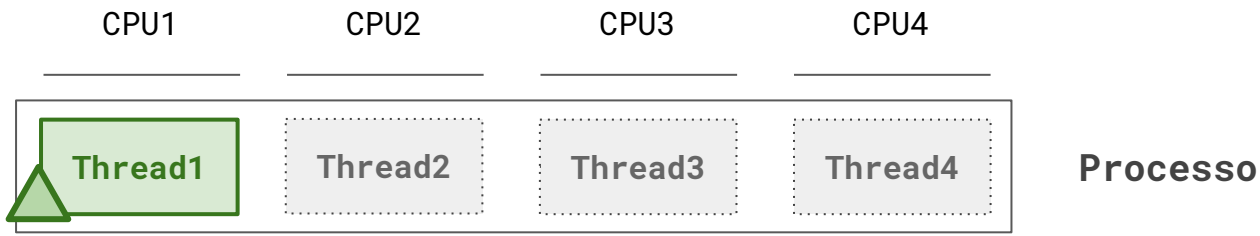
Trazendo isso pra hoje



Trazendo isso pra hoje



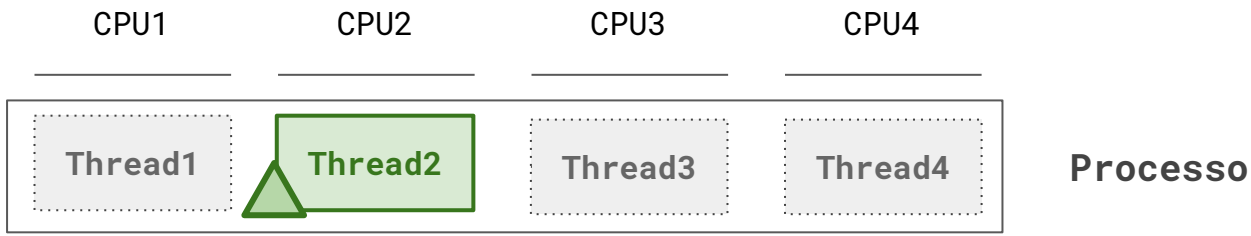
Trazendo isso pra hoje



A cada 100 instruções o interpretador...

1. Trava o mutex
2. Sinaliza isso através de uma variável, que as outras threads estão sempre lendo.
3. Por conta das outras threads estarem sempre esperando a vez delas, syscalls adicionais são usadas pra mandar sinais pra elas.

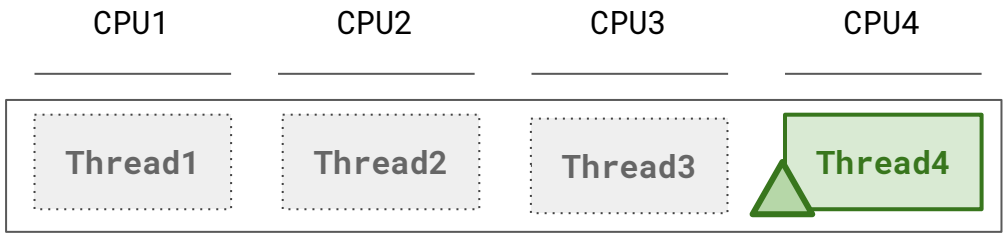
Trazendo isso pra hoje



A cada 100 instruções o interpretador...

1. Trava o mutex
2. Sinaliza isso através de uma variável, que as outras threads estão sempre lendo.
3. Por conta das outras threads estarem sempre esperando a vez delas, syscalls adicionais são usadas pra mandar sinais pra elas.

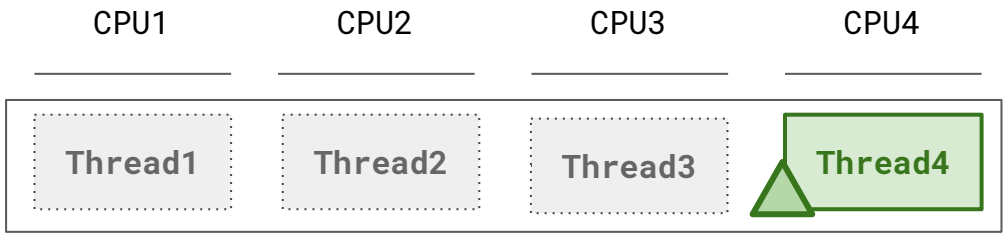
Trazendo isso pra hoje



A cada 100 instruções o interpretador...

1. Trava o mutex
2. Sinaliza isso através de uma variável, que as outras threads estão sempre lendo.
3. Por conta das outras threads estarem sempre esperando a vez delas, syscalls adicionais são usadas pra mandar sinais pra elas.

Trazendo isso pra hoje

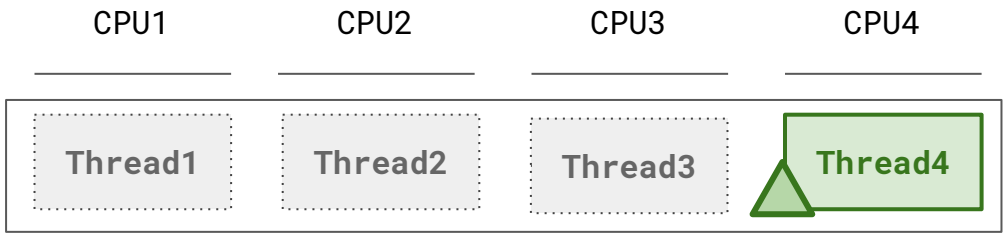


A cada 100 instruções o interpretador...

```
while thread.locked:
    thread.wait()
```

1. Trava o mutex
2. Sinaliza isso através de uma variável, que as outras threads estão sempre lendo.
3. Por conta das outras threads estarem sempre esperando a vez delas, syscalls adicionais são usadas pra mandar sinais pra elas.

Trazendo isso pra hoje




```
def count(n):
  while n > 0:
    n -= 1
```

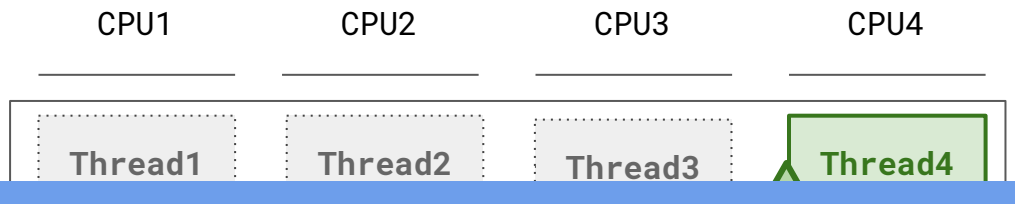
Sequential Execution (OS-X, 1 CPU)
736 Unix system calls
117 Mach System Calls

Two CPU-bound threads (OS-X, 1 CPU)
1149 Unix system calls
~ **3.3 Million** Mach System Calls

```
count(100000000)
```

-x1.8

Trazendo isso pra hoje



Como resolver essa treta?

```
while n > 0:
    n -= 1
```

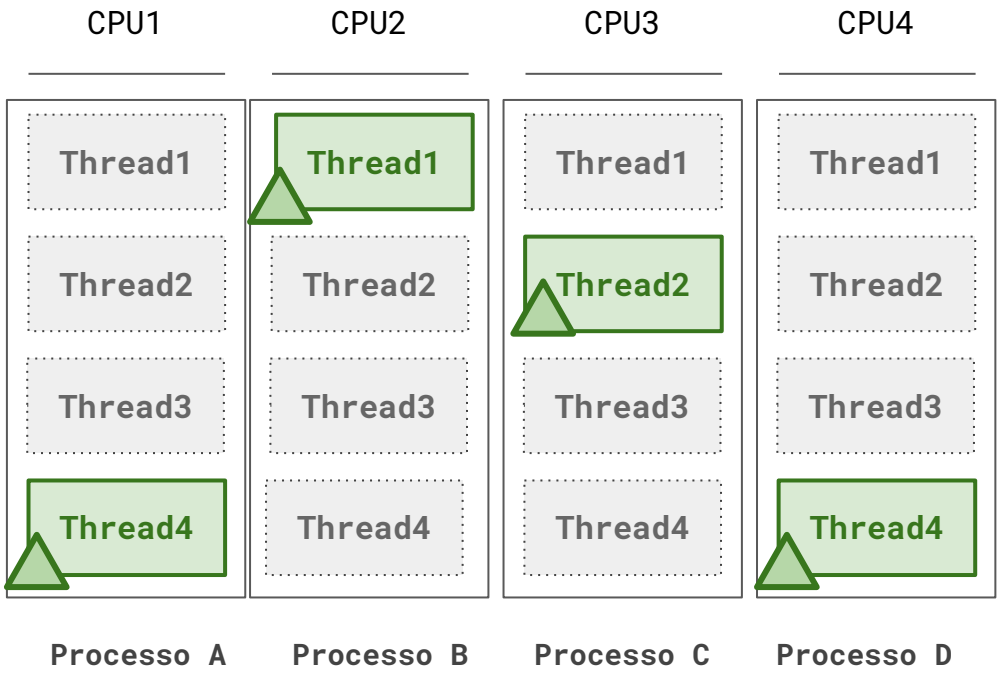
117 Mach System Calls

```
count(100000000)
```

Two CPU-bound threads (OS-X, 1 CPU)
1149 Unix system calls
~ **3.3 Million** Mach System Calls

-x1.8

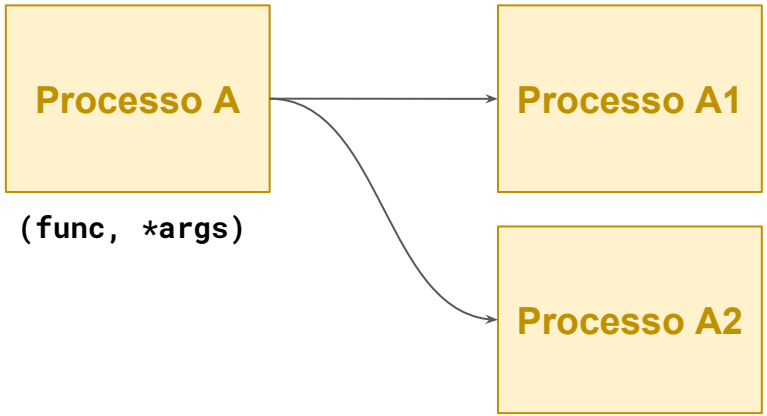
Multiprocessing

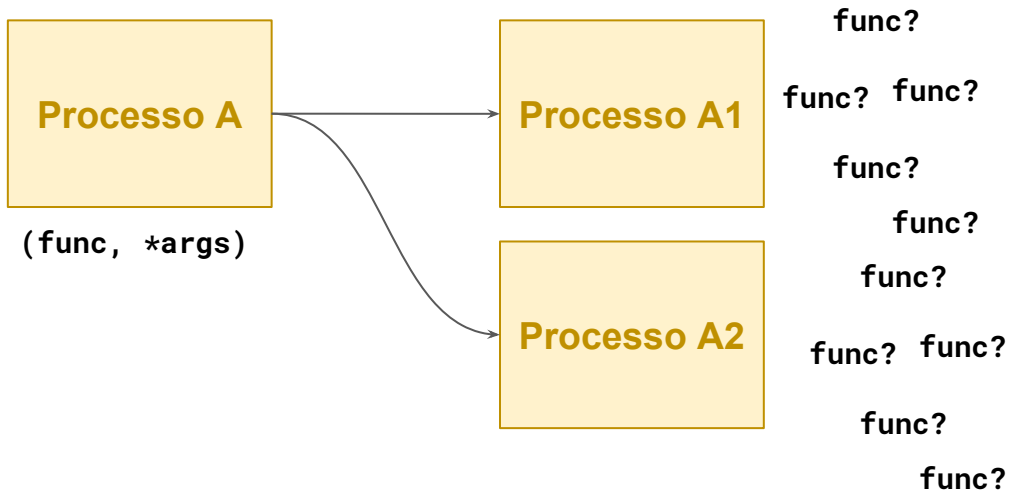


Multiprocessing

< comparação >

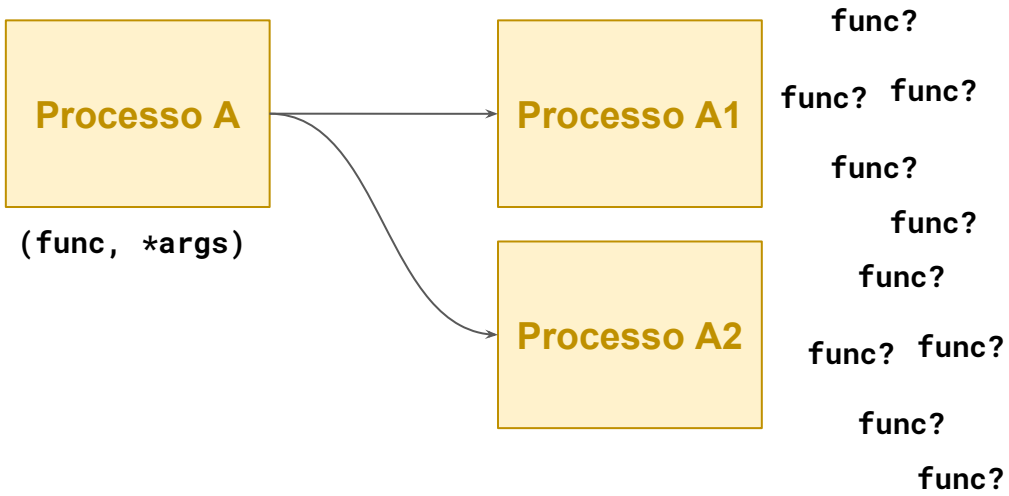
`(func, *args)`


Processo A
(func, *args)
Processo A1
Processo A2


Processo A
(func, *args)
Processo A1
Processo A2
func?
func?
func?
func?
func?
func?
func?
func?
func?
func?

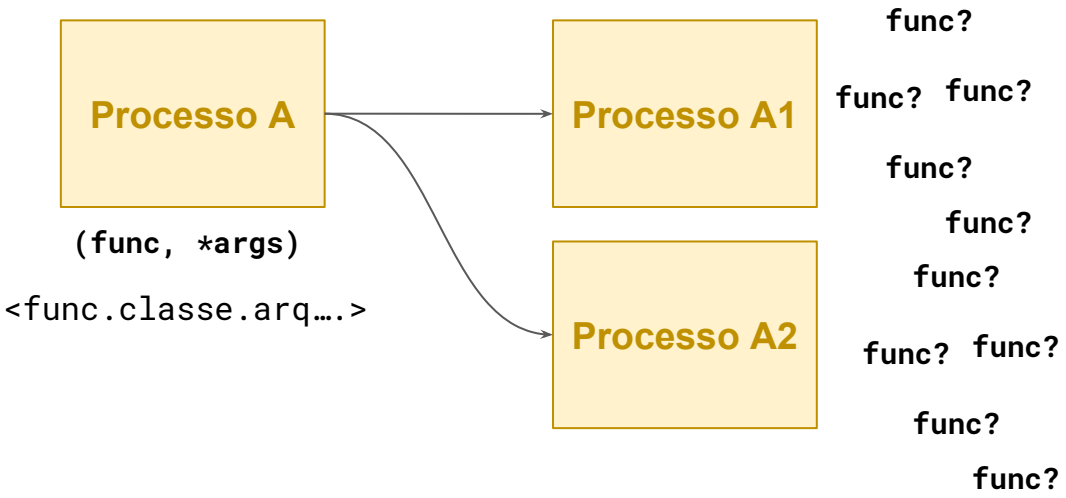
Serialização

Pickle



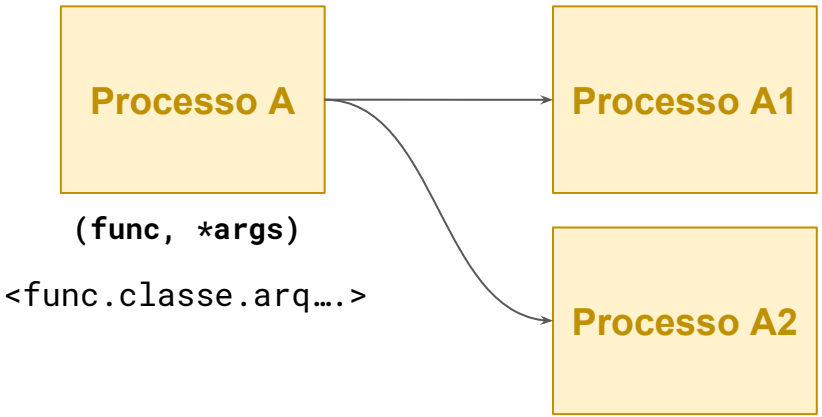
Serialização

Pickle



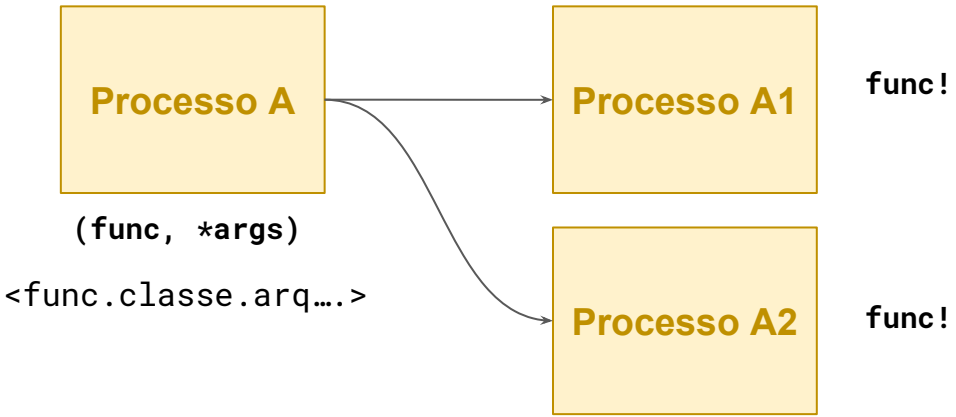
Serialização

Pickle



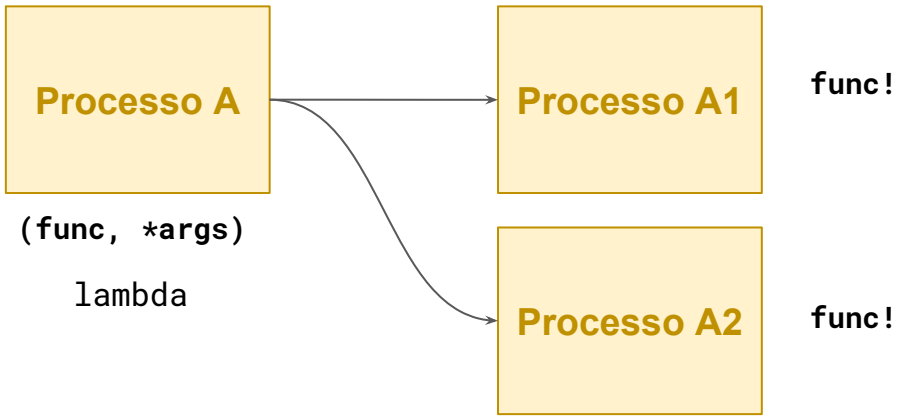
Serialização

Pickle



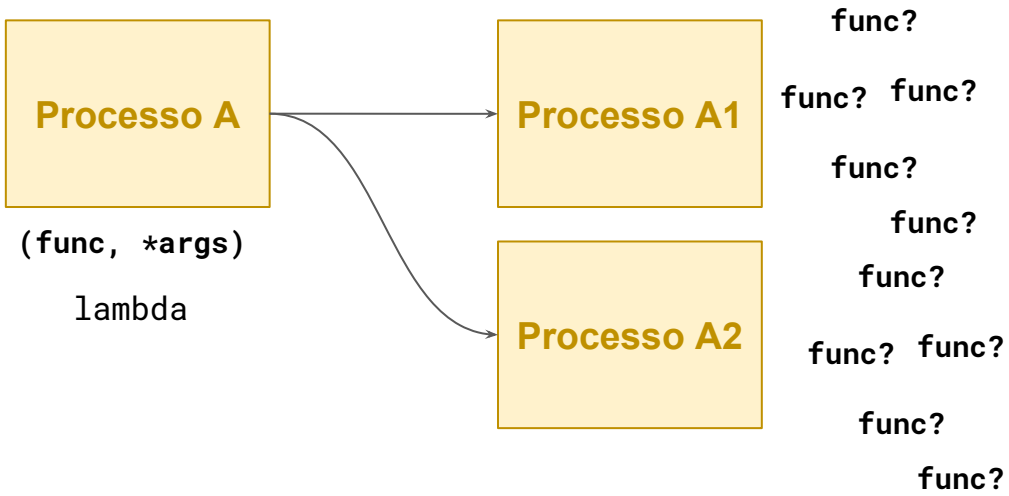
Serialização

Pickle



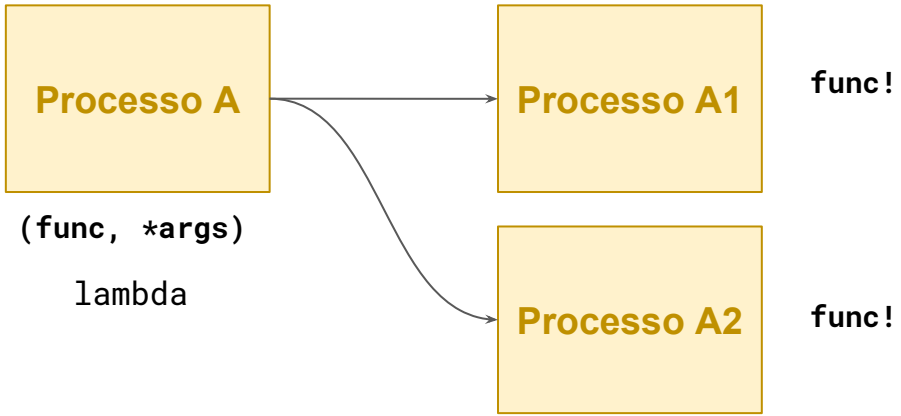
Serialização

Pickle



Serialização

DILL



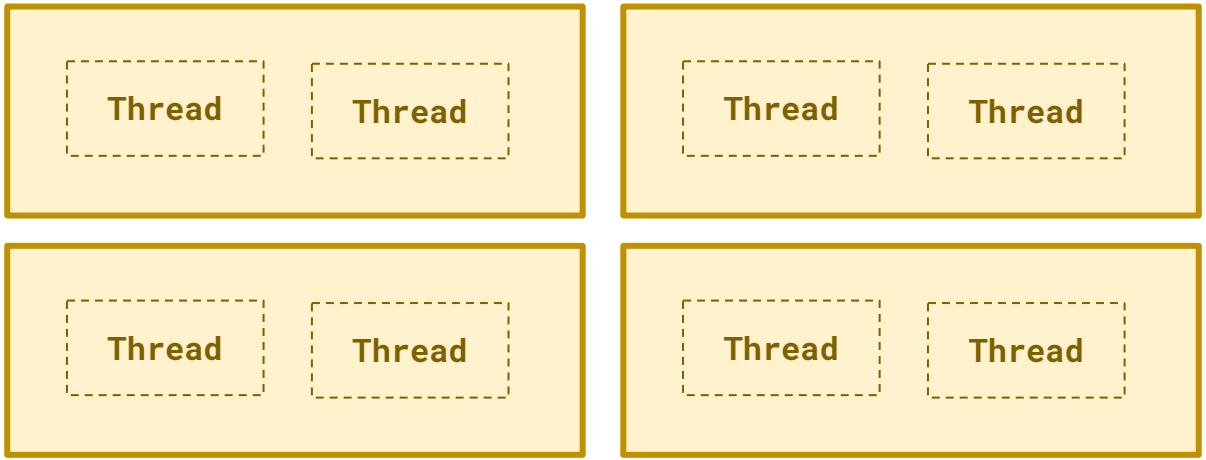
Multiprocessing

PATHOS: Dill + Multiprocessing

a framework for parallel graph management
and execution in heterogeneous computing

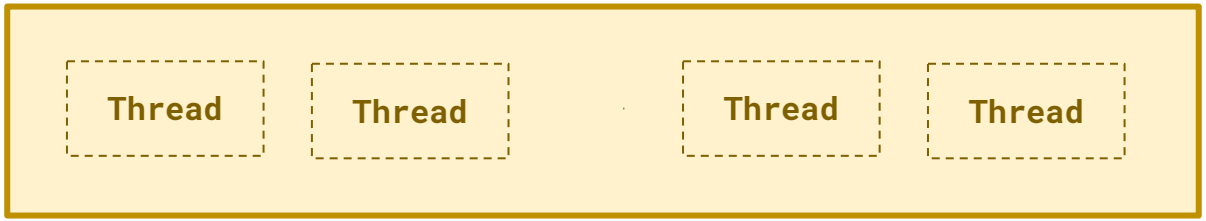
```
curl -s 0.0.0.0:5000/hello
{ “hello world!” }
```

```
gunicorn server:app -w 4 --threads 2
```



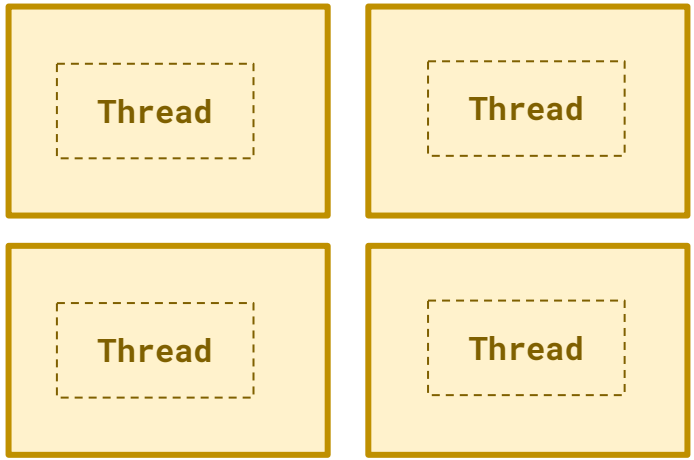
```
curl -s 0.0.0.0:5000/hello/4
{ ‘hello ’ + prime(4) }
```

```
gunicorn server:app -w 1 --threads 4
```



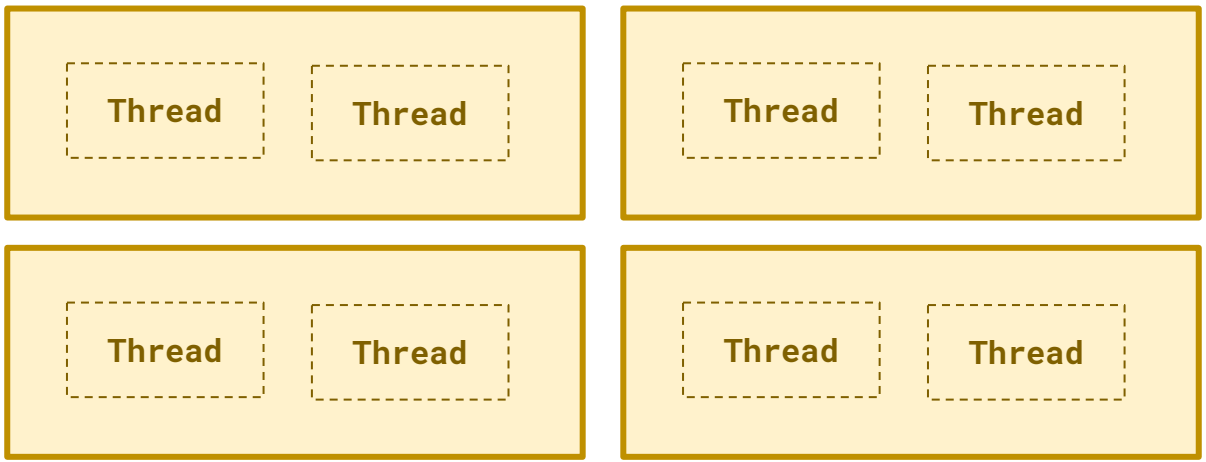
```
curl -s 0.0.0.0:5000/hello/4
{ ‘hello ’ + prime(4) }
```

```
gunicorn server:app -w 4 --threads 1
```



```
curl -s 0.0.0.0:5000/hello/4
{ ‘hello ’ + prime(4) }
```

```
gunicorn server:app -w 4 --threads 2
```



Perguntas?