Parallel Python

Embracing the Future with Sub-Interpreters and Free Threading

Shekhar Koirala

PyCon Ireland 2024

About me

- Working as ML Engineer at Identv LLC
- Studying Masters at Georgia Tech
- Volunteer at EuroPython and other Python/ PyData Conferences

Before …….

1. Don’t Blindly go with the Benchmarks

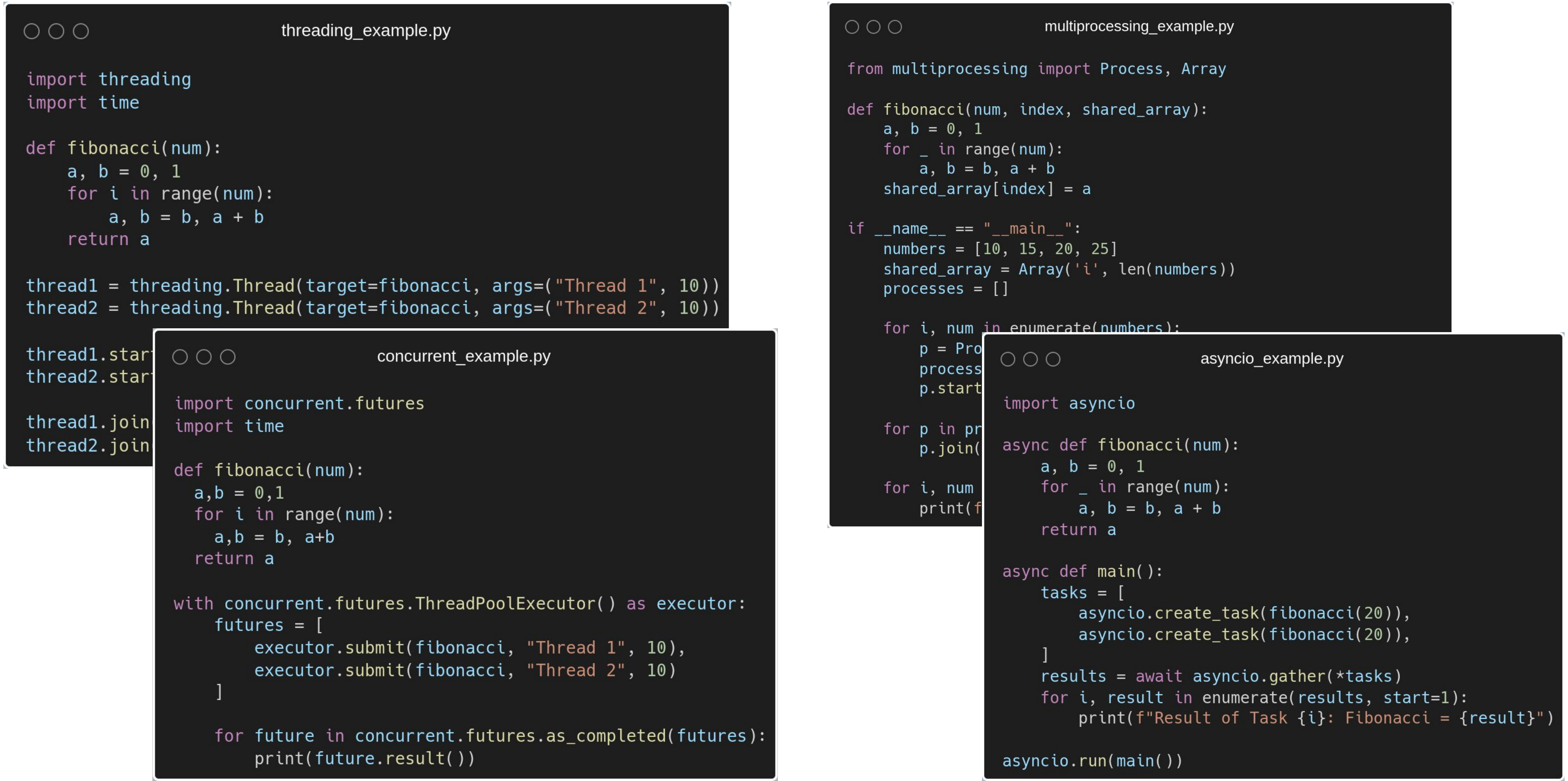
2. Play around with releases but wait for safety

3. Embrace the Incremental Process*

* https://peps.python.org/pep-0703/ [703]
* https://peps.python.org/pep-0684/ [684]
* https://peps.python.org/pep-0734/ [734]

Why care about Parallelism ?

Approaches for Concurrency and Parallelism



threading_example.py

```python
import threading
import time

def fibonacci(num):
    a, b = 0, 1
    for i in range(num):
        a, b = b, a + b
    return a

thread1 = threading.Thread(target=fibonacci, args=("Thread 1", 10))
thread2 = threading.Thread(target=fibonacci, args=("Thread 2", 10))

thread1.star
thread2.star

thread1.join
thread2.join
```

concurrent_example.py

```python
import concurrent.futures
import time

def fibonacci(num):
  a,b = 0,1
  for i in range(num):
    a,b = b, a+b
  return a

with concurrent.futures.ThreadPoolExecutor() as executor:
    futures = [
        executor.submit(fibonacci, "Thread 1", 10),
        executor.submit(fibonacci, "Thread 2", 10)
    ]

    for future in concurrent.futures.as_completed(futures):
        print(future.result())
```

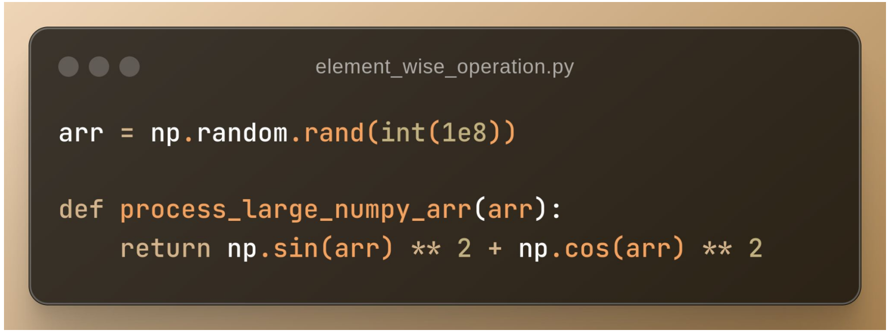
multiprocessing_example.py

```python
from multiprocessing import Process, Array

def fibonacci(num, index, shared_array):
    a, b = 0, 1
    for _ in range(num):
        a, b = b, a + b
    shared_array[index] = a

if __name__ == "__main__":
    numbers = [10, 15, 20, 25]
    shared_array = Array('i', len(numbers))
    processes = []

    for i, num in enumerate(numbers):
        p = Pro
        process
        p.start

    for p in pr
        p.join(

    for i, num
        print(f
```

asyncio_example.py

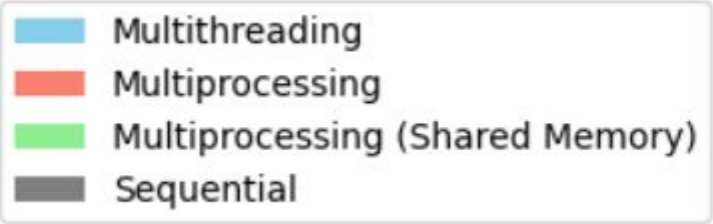
```python
import asyncio

async def fibonacci(num):
    a, b = 0, 1
    for _ in range(num):
        a, b = b, a + b
    return a

async def main():
    tasks = [
        asyncio.create_task(fibonacci(20)),
        asyncio.create_task(fibonacci(20)),
    ]
    results = await asyncio.gather(*tasks)
    for i, result in enumerate(results, start=1):
        print(f"Result of Task {i}: Fibonacci = {result}")

asyncio.run(main())
```

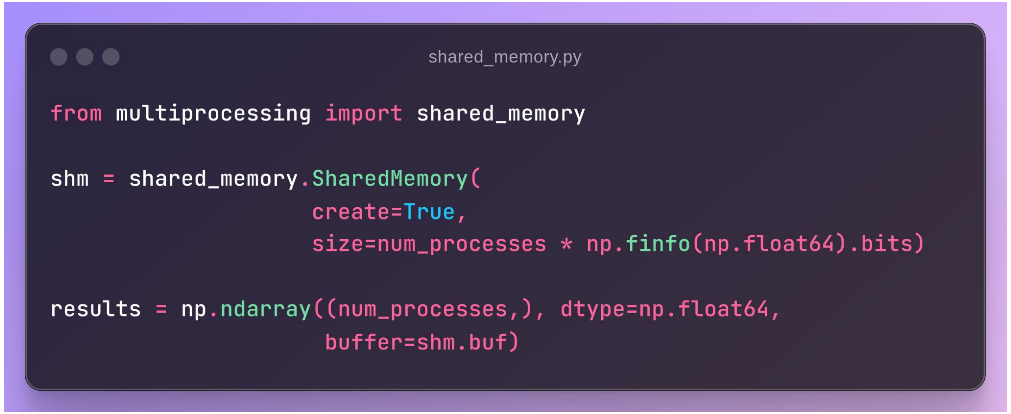
Quiz Time



element_wise_operation.py

```python
arr = np.random.rand(int(1e8))

def process_large_numpy_arr(arr):
    return np.sin(arr) ** 2 + np.cos(arr) ** 2
```



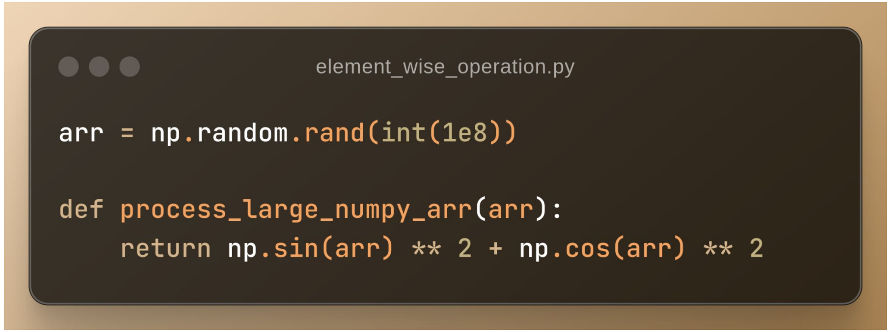


shared_memory.py

```python
from multiprocessing import shared_memory

shm = shared_memory.SharedMemory(
                create=True,
                size=num_processes * np.finfo(np.float64).bits)

results = np.ndarray((num_processes,), dtype=np.float64,
                 buffer=shm.buf)
```

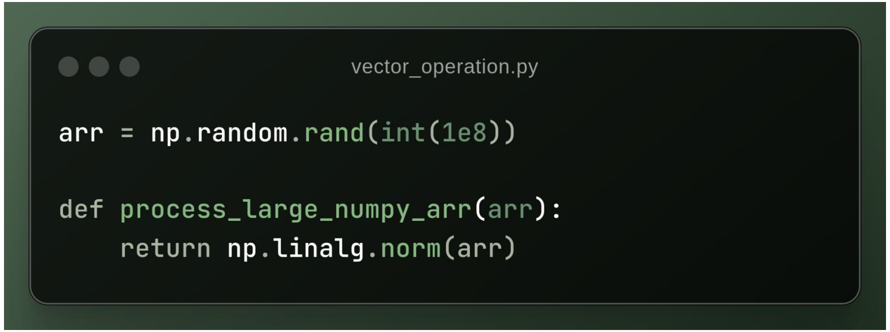
* issue: https://github.com/python/cpython/issues/82300

*https://medium.com/@turbopython/multithreading-vs-multiprocessing-with-numpy-which-one-is-faster-afebf027a0fa



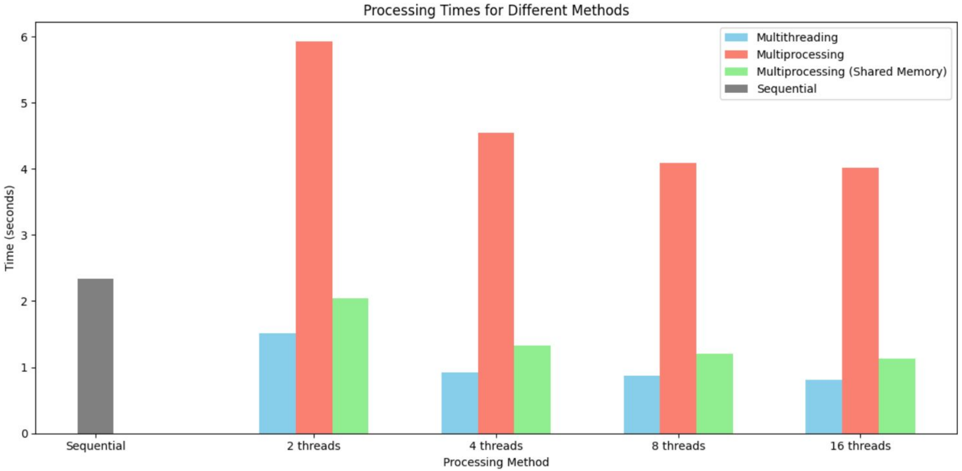
element_wise_operation.py

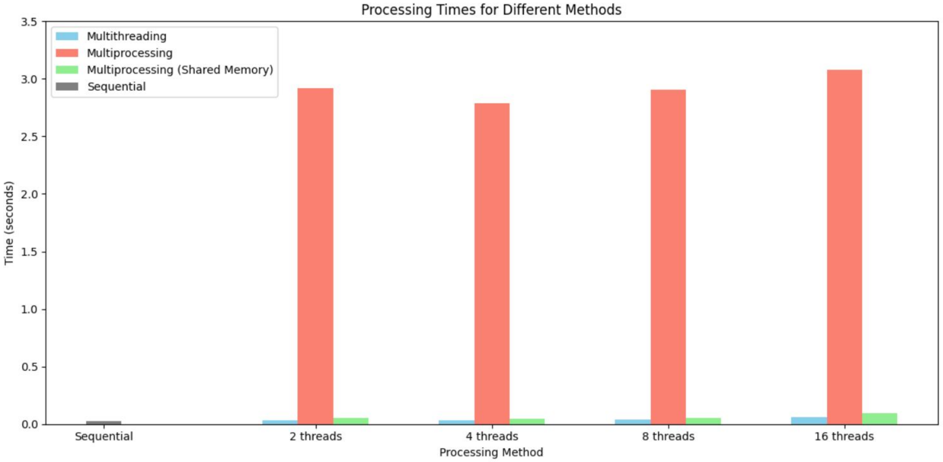
```python
arr = np.random.rand(int(1e8))

def process_large_numpy_arr(arr):
    return np.sin(arr) ** 2 + np.cos(arr) ** 2
```



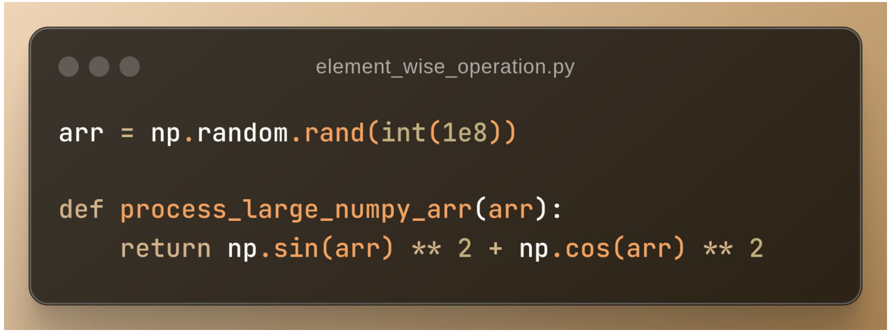
vector_operation.py

```python
arr = np.random.rand(int(1e8))

def process_large_numpy_arr(arr):
    return np.linalg.norm(arr)
```



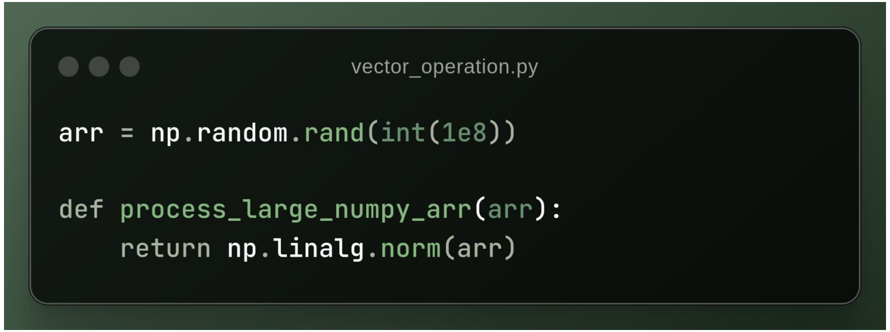


*https://medium.com/@turbopython/multithreading-vs-multiprocessing-with-numpy-which-one-is-faster-afebf027a0fa



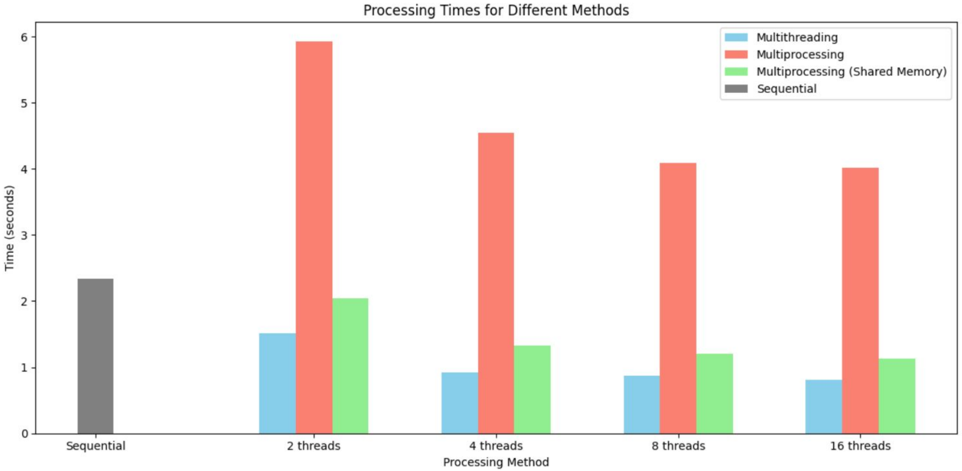
element_wise_operation.py

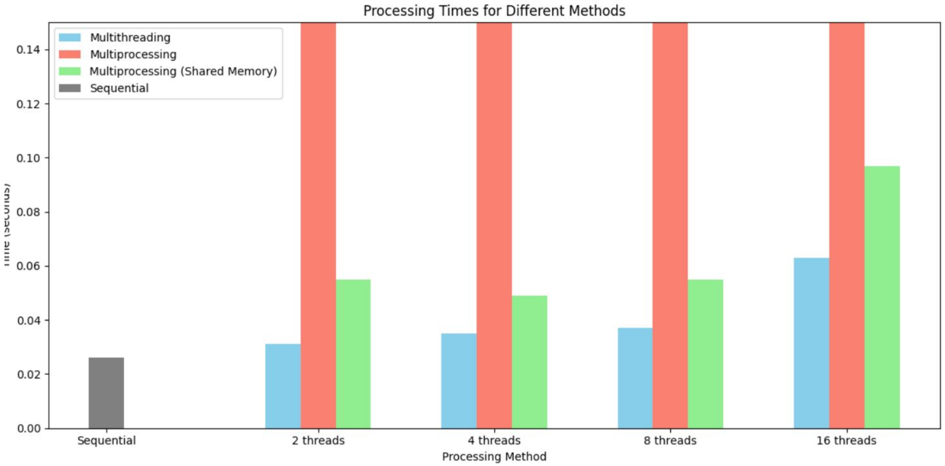
```python
arr = np.random.rand(int(1e8))

def process_large_numpy_arr(arr):
    return np.sin(arr) ** 2 + np.cos(arr) ** 2
```



vector_operation.py

```python
arr = np.random.rand(int(1e8))

def process_large_numpy_arr(arr):
    return np.linalg.norm(arr)
```

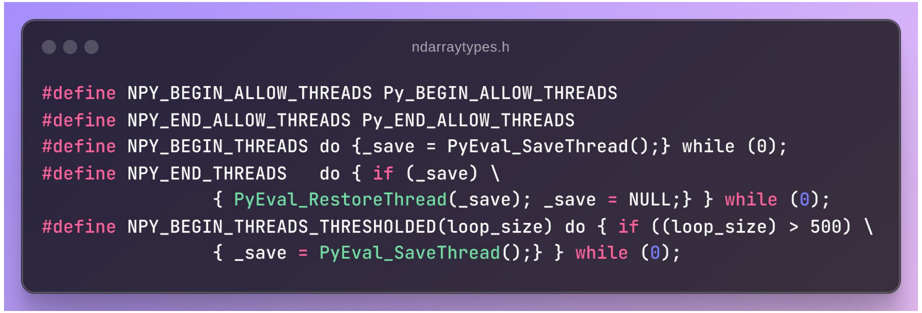




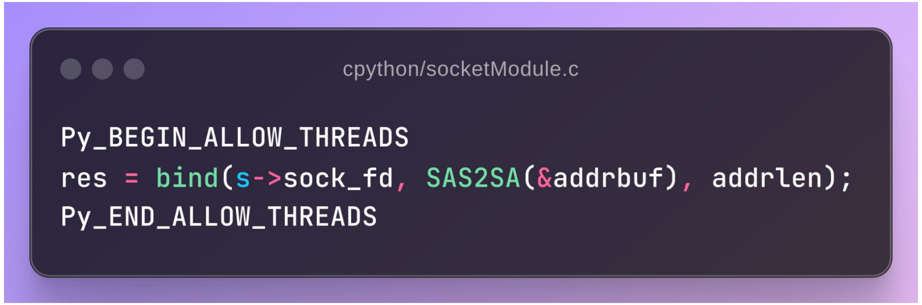
Why it happened ?

Note that operations that *do not* release the GIL will see no performance gains from use of the **threading** module,
and instead might be better served with **multiprocessing**.
In particular, operations on arrays with dtype=object do not release the GIL.

- * https://numpy.org/devdocs/reference/thread_safety.html

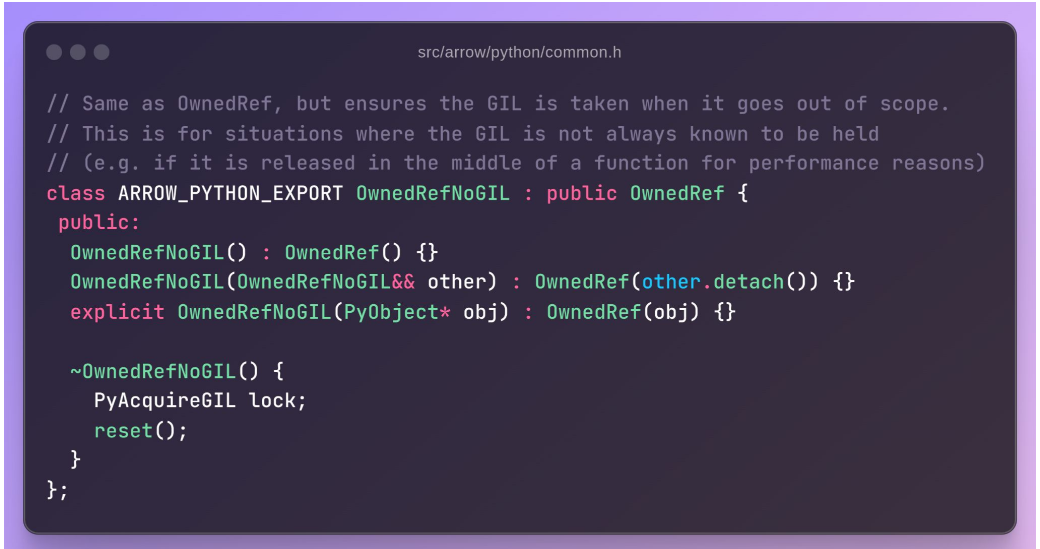


```
#define NPY_BEGIN_ALLOW_THREADS Py_BEGIN_ALLOW_THREADS
#define NPY_END_ALLOW_THREADS Py_END_ALLOW_THREADS
#define NPY_BEGIN_THREADS do {_save = PyEval_SaveThread();} while (0);
#define NPY_END_THREADS   do { if (_save) \
                { PyEval_RestoreThread(_save); _save = NULL;} } while (0);
#define NPY_BEGIN_THREADS_THRESHOLDED(loop_size) do { if ((loop_size) > 500) \
                { _save = PyEval_SaveThread();} } while (0);
```

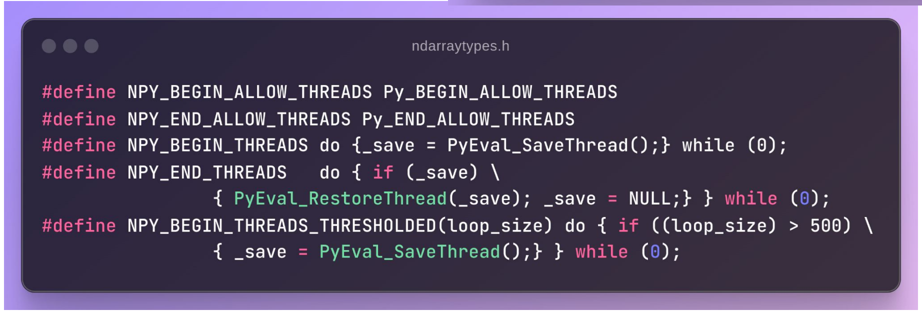


```
Py_BEGIN_ALLOW_THREADS
res = bind(s->sock_fd, SAS2SA(&addrbuf), addrlen);
Py_END_ALLOW_THREADS
```

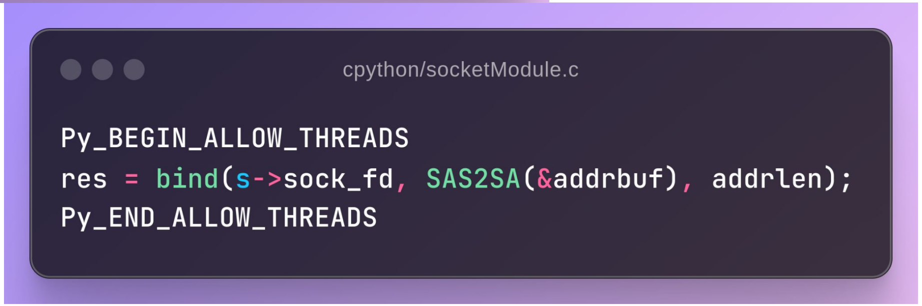
Why it happened ?



```cpp
// Same as OwnedRef, but ensures the GIL is taken when it goes out of scope.
// This is for situations where the GIL is not always known to be held
// (e.g. if it is released in the middle of a function for performance reasons)
class ARROW_PYTHON_EXPORT OwnedRefNoGIL : public OwnedRef {
 public:
  OwnedRefNoGIL() : OwnedRef() {}
  OwnedRefNoGIL(OwnedRefNoGIL&& other) : OwnedRef(other.detach()) {}
  explicit OwnedRefNoGIL(PyObject* obj) : OwnedRef(obj) {}

  ~OwnedRefNoGIL() {
    PyAcquireGIL lock;
    reset();
  }
};
```

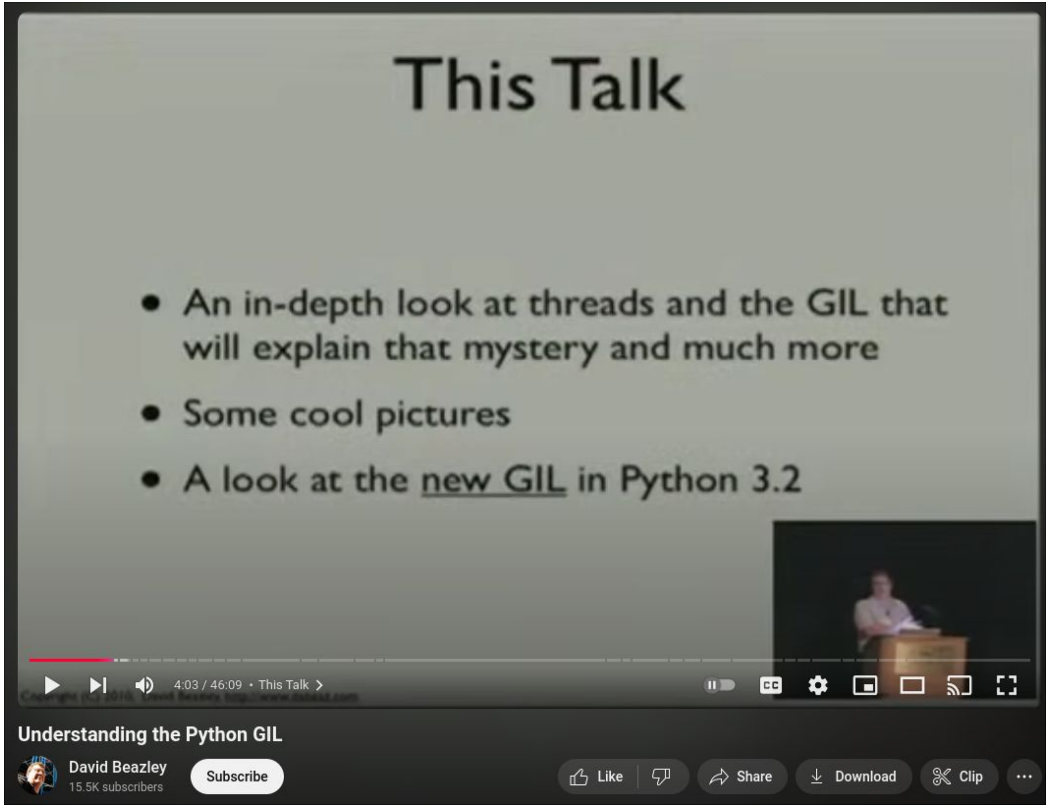


```c
#define NPY_BEGIN_ALLOW_THREADS Py_BEGIN_ALLOW_THREADS
#define NPY_END_ALLOW_THREADS Py_END_ALLOW_THREADS
#define NPY_BEGIN_THREADS do {_save = PyEval_SaveThread();} while (0);
#define NPY_END_THREADS   do { if (_save) \
                { PyEval_RestoreThread(_save); _save = NULL;} } while (0);
#define NPY_BEGIN_THREADS_THRESHOLDED(loop_size) do { if ((loop_size) > 500) \
                { _save = PyEval_SaveThread();} } while (0);
```



```c
Py_BEGIN_ALLOW_THREADS
res = bind(s->sock_fd, SAS2SA(&addrbuf), addrlen);
Py_END_ALLOW_THREADS
```

GIL



This Talk

- An in-depth look at threads and the GIL that will explain that mystery and much more
- Some cool pictures
- A look at the new GIL in Python 3.2

Copyright (C) 2010, David Beazley, http://www.dabeaz.com

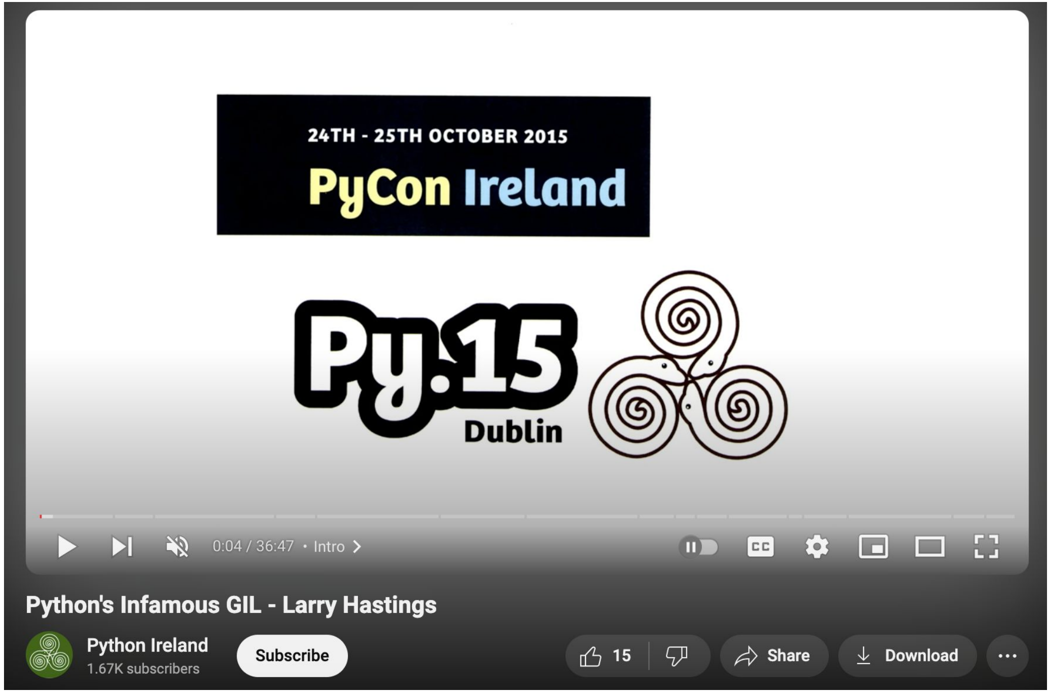
Understanding the Python GIL

David Beazley

15.5K subscribers

*https://www.youtube.com/watch?v=Obt-vMVdM8s

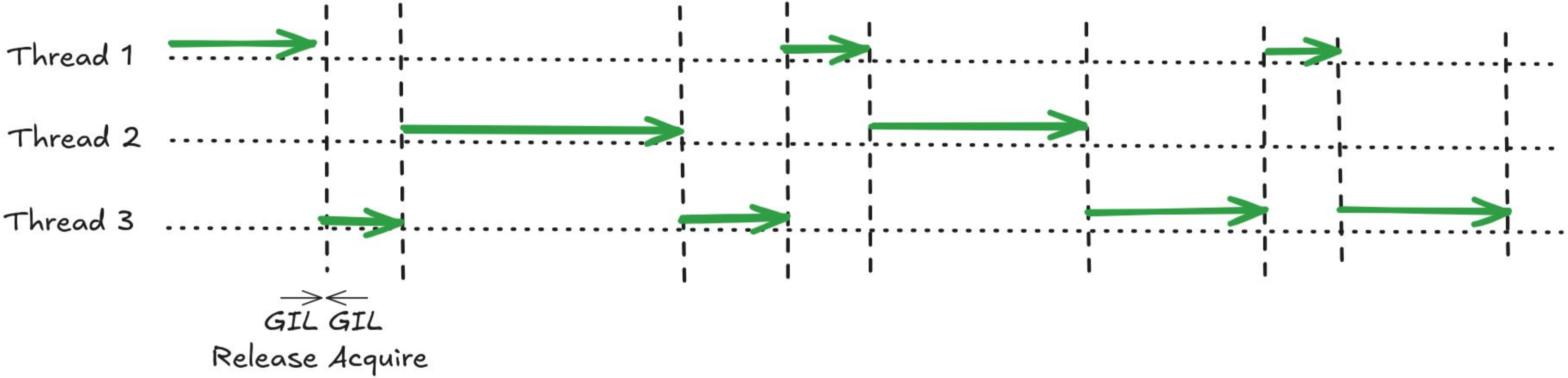
GIL

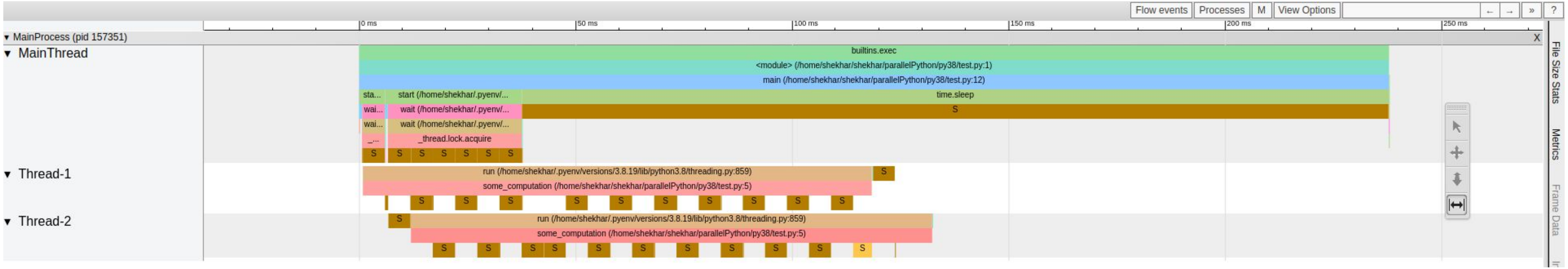


Python's Infamous GIL - Larry Hastings

*https://www.youtube.com/watch?v=sxMl4DsYgpw

Visualizing GIL




Flow events
Processes
M
View Options
?
0 ms
50 ms
100 ms
150 ms
200 ms
250 ms
MainProcess (pid 157351)
MainThread
builtins.exec
<module> (/home/shekhar/shekhar/parallelPython/py38/test.py:1)
main (/home/shekhar/shekhar/parallelPython/py38/test.py:12)
sta...
start (/home/shekhar/.pyenv/...
time.sleep
wai...
wait (/home/shekhar/.pyenv/...
S
wai...
wait (/home/shekhar/.pyenv/...
_...
_thread.lock.acquire
S S S S S S S
Thread-1
run (/home/shekhar/.pyenv/versions/3.8.19/lib/python3.8/threading.py:859)
S
some_computation (/home/shekhar/shekhar/parallelPython/py38/test.py:5)
S S S S S S S S S S
Thread-2
S
run (/home/shekhar/.pyenv/versions/3.8.19/lib/python3.8/threading.py:859)
some_computation (/home/shekhar/shekhar/parallelPython/py38/test.py:5)
S S S S S S S S S S S
File Size Stats
Metrics
Frame Data

*https://www.maartenbreddels.com/perf/jupyter/python/tracing/gil/2021/01/14/Tracing-the-Python-GIL.html

More on GIL

- Lock used by Python runtimes to protect Global states and variables
- Prevents race condition when running its ByteCode.
- Protect C extension module
- This was in the 90’s where many devices where single core.
- Gil was attempted to removed but it was slower than single thread.*

*https://www.youtube.com/watch?v=P3AyI_u66Bw

GIL was/is hard to remove

- Lots of C extensions assumes GIL exists
- ABI changes

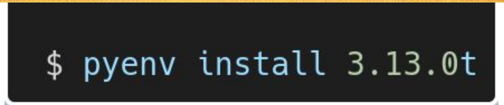
* No GIL but use Locks or immortal objects* !!!

* https://peps.python.org/pep-0683/

Other Parallelism Project

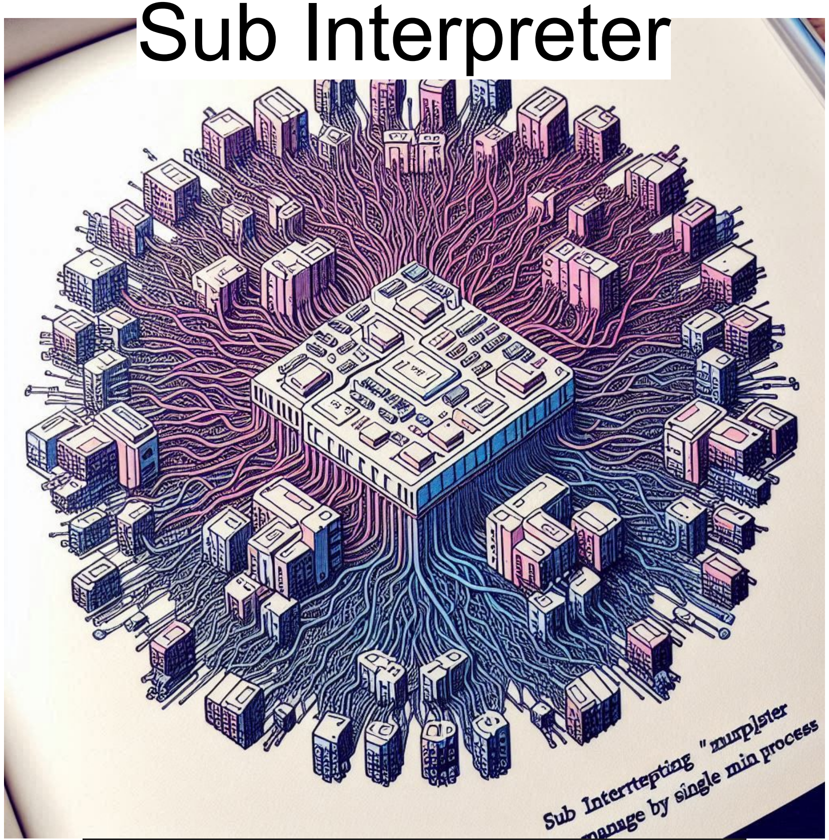
- CPython alternative : Jython / IronPython
- Gilectomy [“no-gil” project]
- PyParallel Project
- Other parallelism tool : Dask/ Taichi
- MultiProcessing
- Give Up Multi Core , ASYNCIO …. !!!

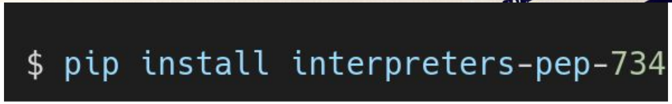
Free Threading



```
$ pyenv install 3.13.0t
```

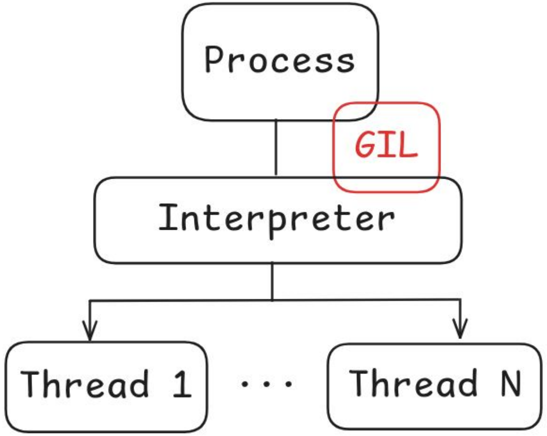
Sub Interpreter



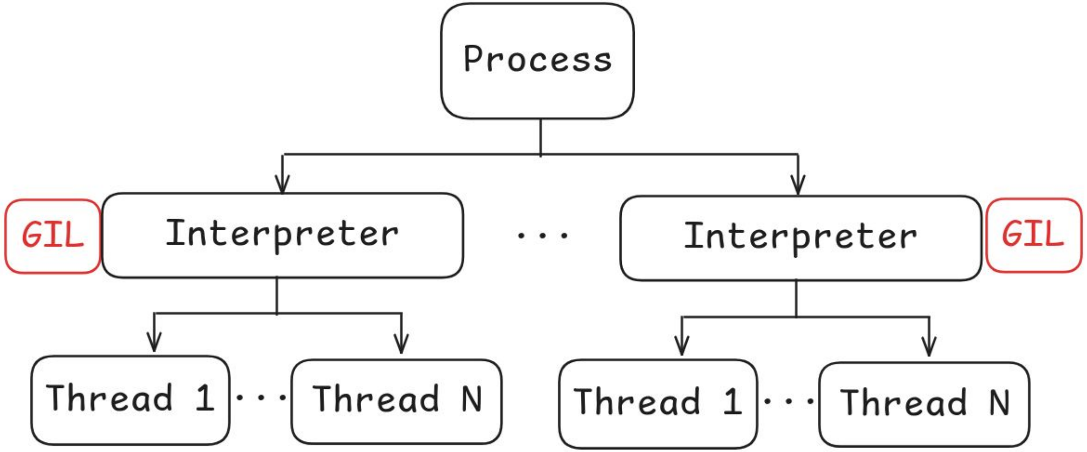


```
$ pip install interpreters-pep-734
```

Sub interpreters



Python <=3.11



*Python >=3.12

* https://youtu.be/z4MaqTbM_gY?si=VIwHv72WkwQCApxG Anthony Shaw

Differences

Feature	Multiprocessing	Sub Interpreter
Execution	Separate OS Processes	Multiple Interpreters within one process
Memory Isolation	Separate Memory per process	Same Memory space but separate Python states
Communication	Inter Process Communication [Pipe, Queue, Manager]	Shared Memory Access [Queues Channel] *
Overhead / creation	Higher to create	Lower to create [less memory footprint]

*https://www.youtube.com/watch?v=fwRMdncVOnA

Start-up Time in Sub Interpreter

Around 5x improvement compared to multiprocessing.*

```
multiprocessing.set_start_method("spawn", force=True)
```

Speed of starting up sub interpreters lies between the “fork” and “spawn” method.

*https://discuss.python.org/t/expected-performance-characteristics-of-subinterpreters/53251

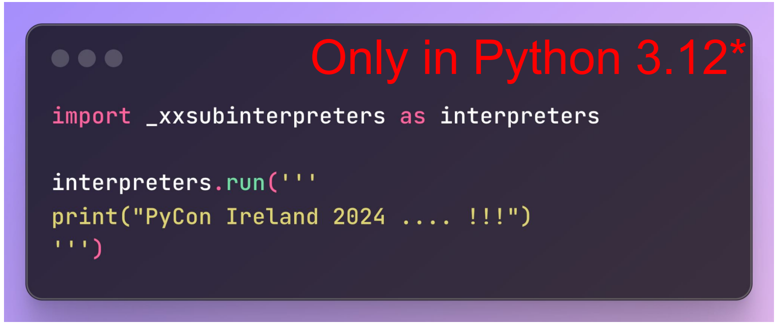
Sharable Objects in Sub Interpreter

- str
- bytes
- int
- float
- bool (True/False)
- None
- tuple (only with shareable items)
- interpreters.Queue
- Memoryview (underlying buffer actually shared)

IMMUTABLE

→Synchronization using locks can be used for Thread safety.

Interpreter.get_current("situation")

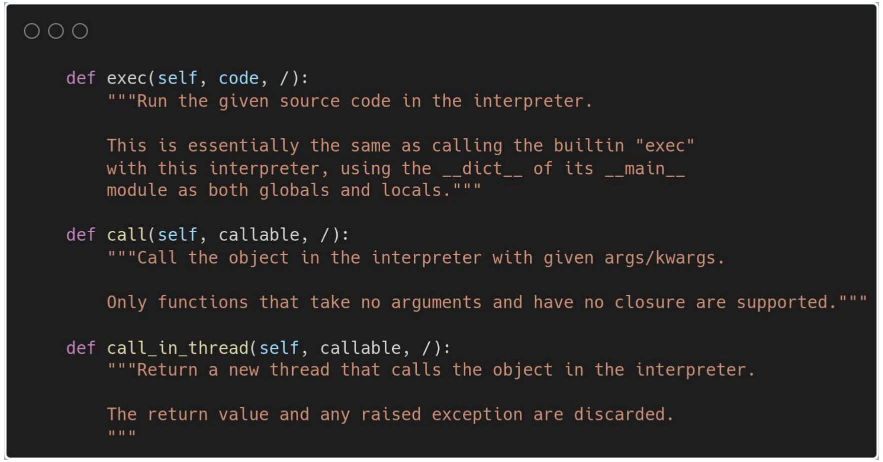


Only in Python 3.12*

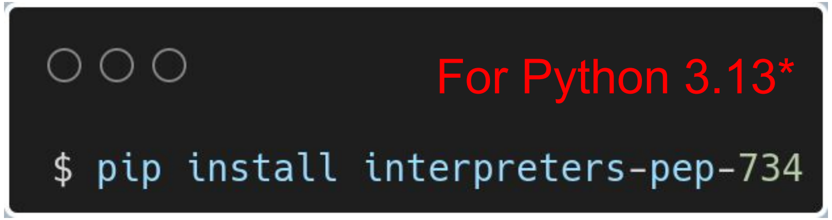
```python
import _xxsubinterpreters as interpreters

interpreters.run('''
print("PyCon Ireland 2024 .... !!!")
''')
```

PEP 734 – Multiple Interpreters in the Stdlib

Status: Deferred



```python
    def exec(self, code, /):
        """Run the given source code in the interpreter.

        This is essentially the same as calling the builtin "exec"
        with this interpreter, using the __dict__ of its __main__
        module as both globals and locals."""

    def call(self, callable, /):
        """Call the object in the interpreter with given args/kwargs.

        Only functions that take no arguments and have no closure are supported."""

    def call_in_thread(self, callable, /):
        """Return a new thread that calls the object in the interpreter.

        The return value and any raised exception are discarded.
        """
```



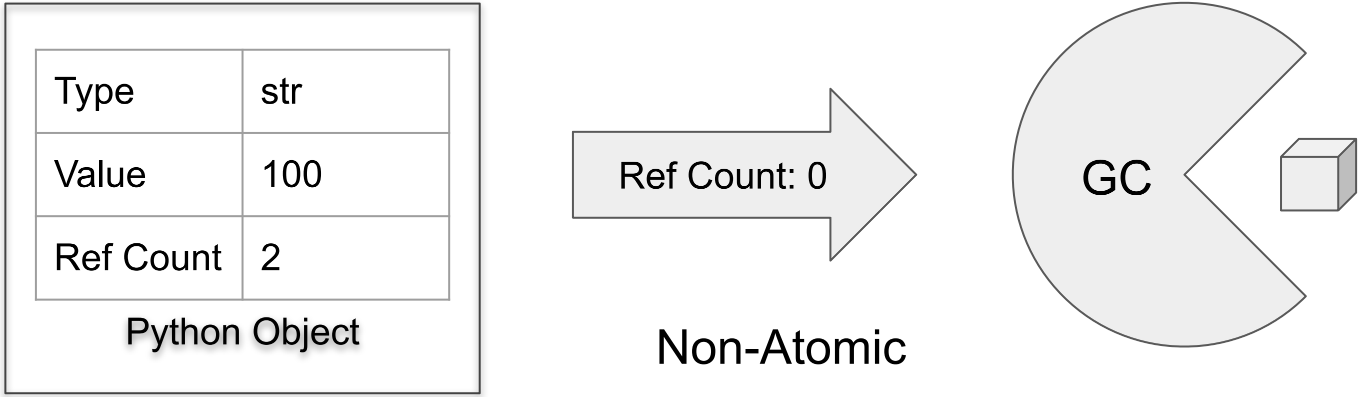
For Python 3.13*

```
$ pip install interpreters-pep-734
```

*will be added in concurrent.futures.InterpreterPoolExecutor

Free Threading

Reference Counting



Type	str
Value	100
Ref Count	2

Python Object

Non-Atomic

Biased reference count = sum(local ref count + shared ref count)

Free Threading

Reference Counting

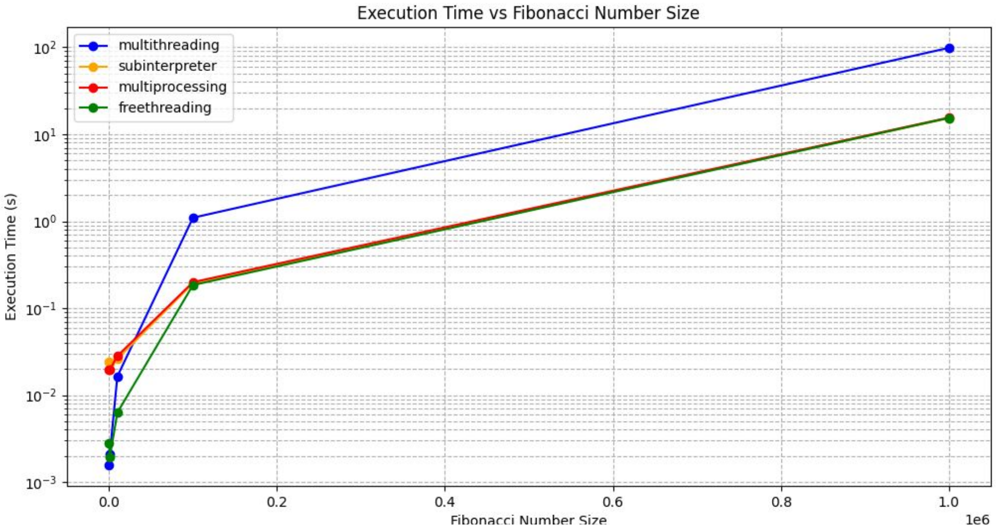
Memory Allocation

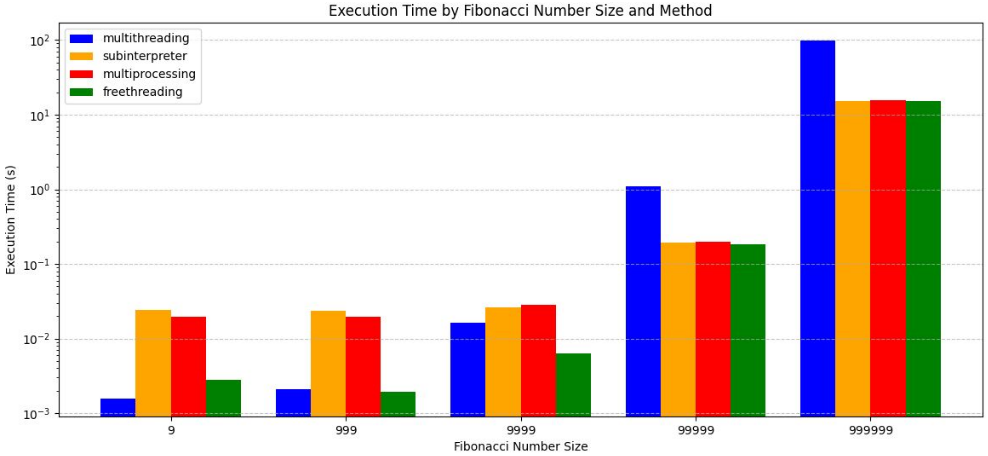
Thread Safety

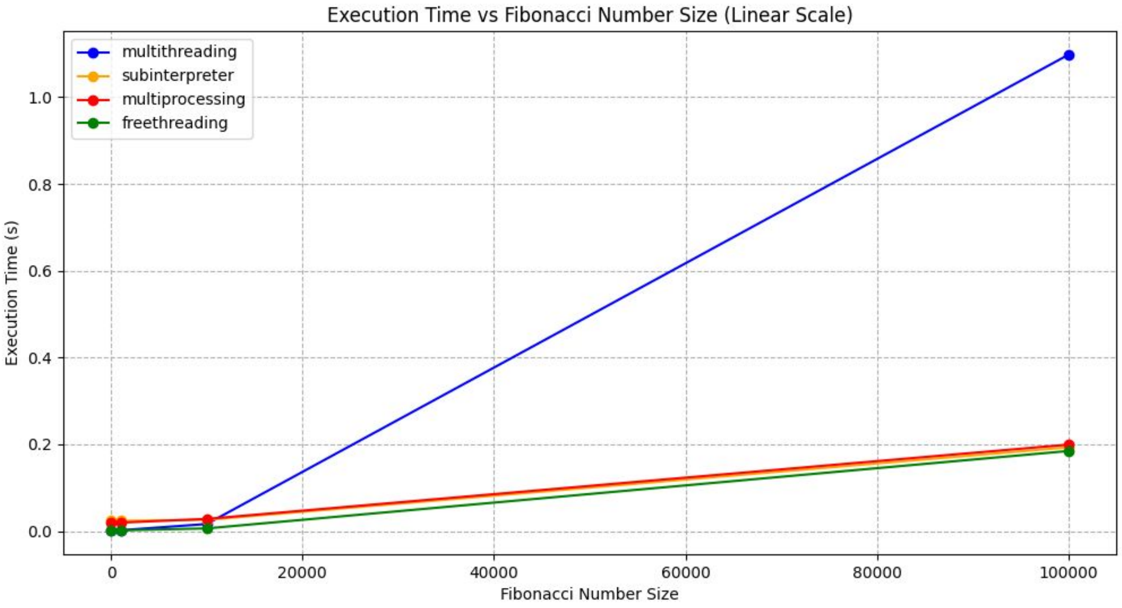
Special wheel and special build of Cpython

Introduces new ABI

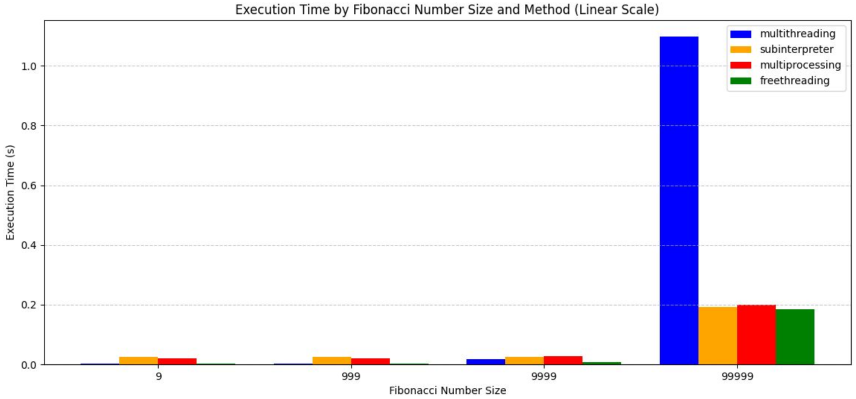
Benchmarks


Execution Time vs Fibonacci Number Size
multithreading
subinterpreter
multiprocessing
freethreading
Execution Time (s)
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
0.0
0.2
0.4
0.6
0.8
1.0
1e6
Fibonacci Number Size


Execution Time by Fibonacci Number Size and Method
multithreading
subinterpreter
multiprocessing
freethreading
Execution Time (s)
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
9
999
9999
99999
999999
Fibonacci Number Size

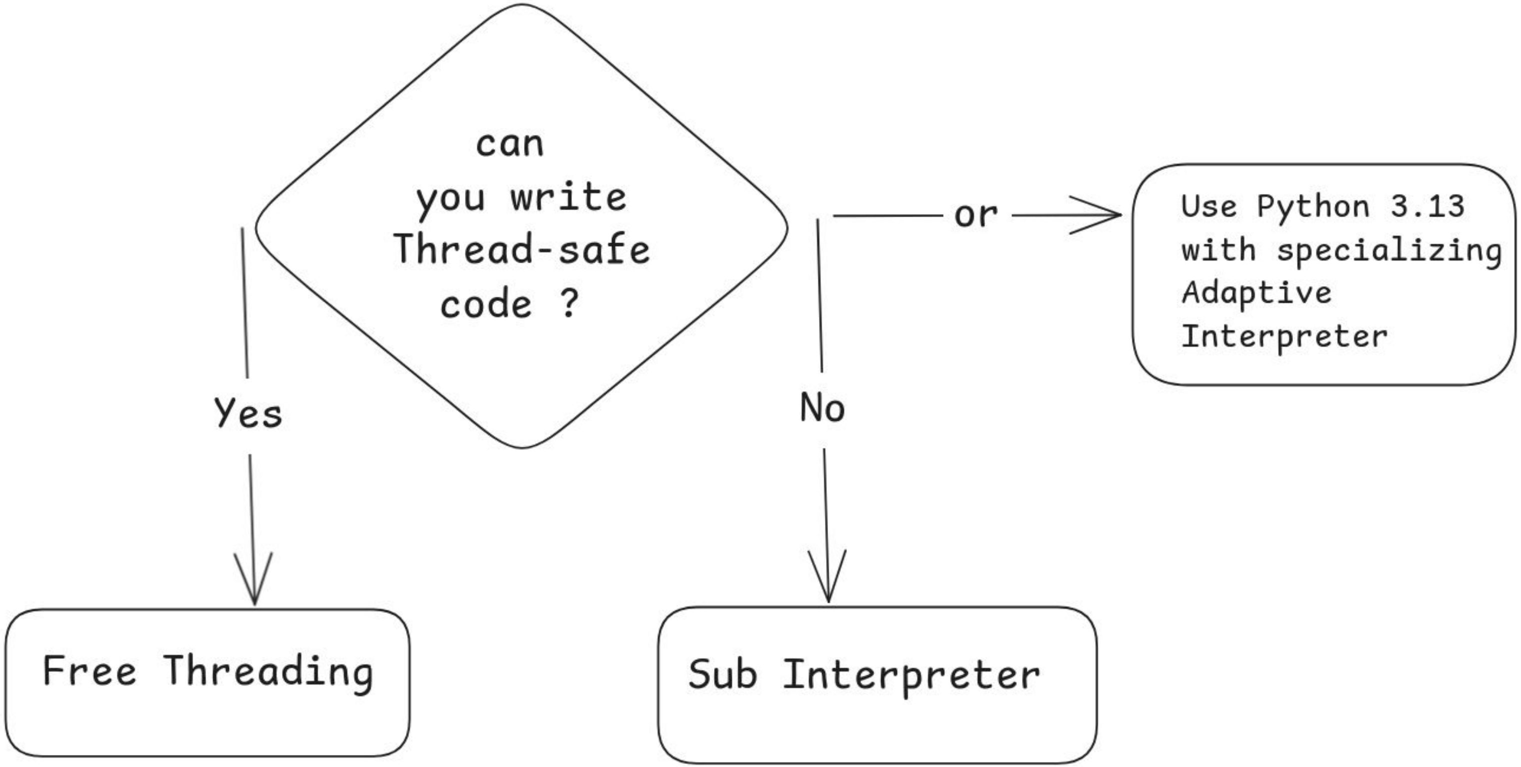


Execution Time vs Fibonacci Number Size (Linear Scale)



Execution Time by Fibonacci Number Size and Method (Linear Scale)

Free Threading or Sub Interpreter ?



can you write Thread-safe code ?

Yes

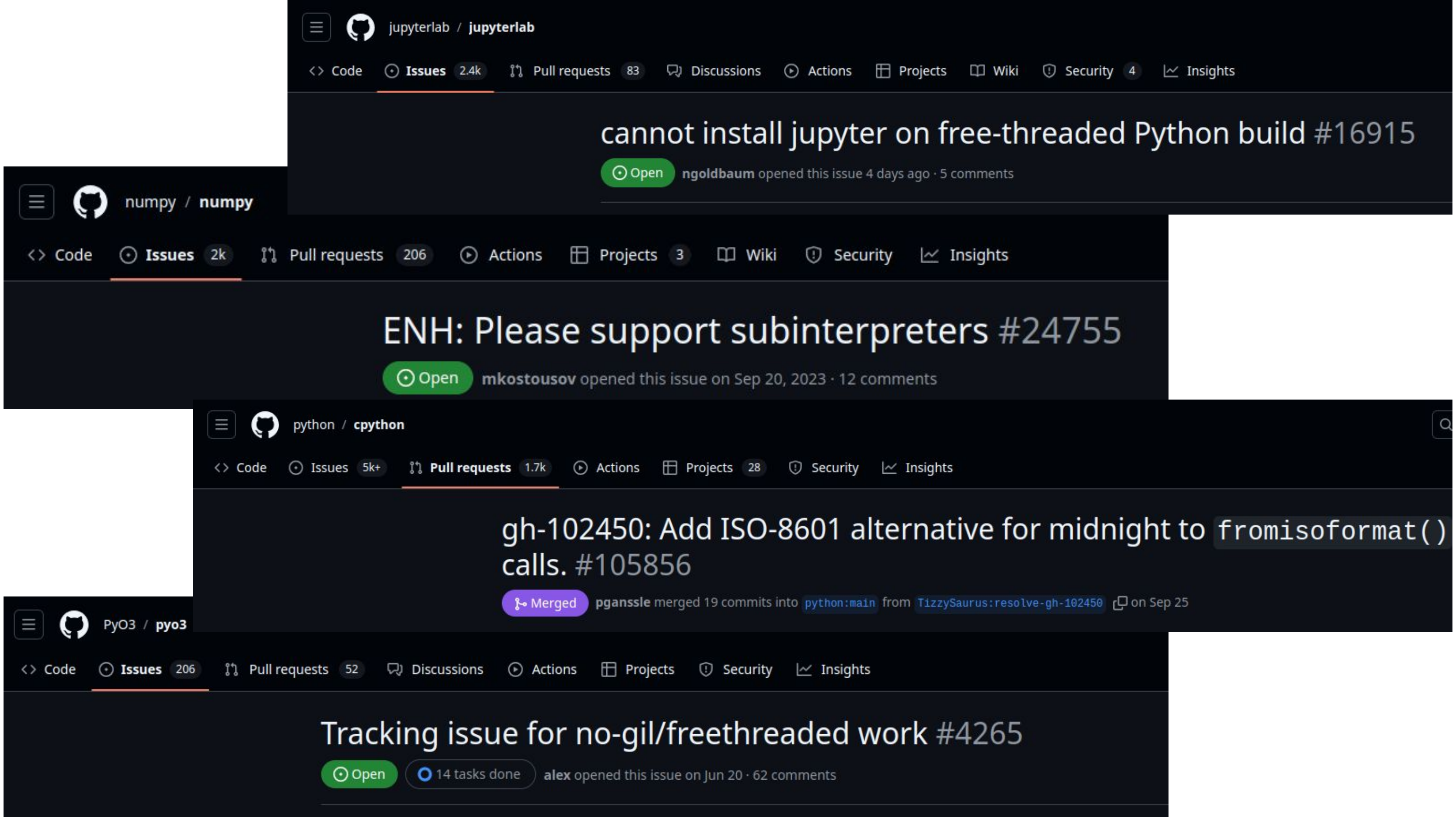
Free Threading

No

Sub Interpreter

or

Use Python 3.13 with specializing Adaptive Interpreter



cannot install jupyter on free-threaded Python build #16915

Open **ngoldbaum** opened this issue 4 days ago · 5 comments

ENH: Please support subinterpreters #24755

Open **mkostousov** opened this issue on Sep 20, 2023 · 12 comments

gh-102450: Add ISO-8601 alternative for midnight to `fromisoformat()` calls. #105856

Merged **pganssle** merged 19 commits into `python:main` from `TizzySaurus:resolve-gh-102450` on Sep 25

Tracking issue for no-gil/freethreaded work #4265

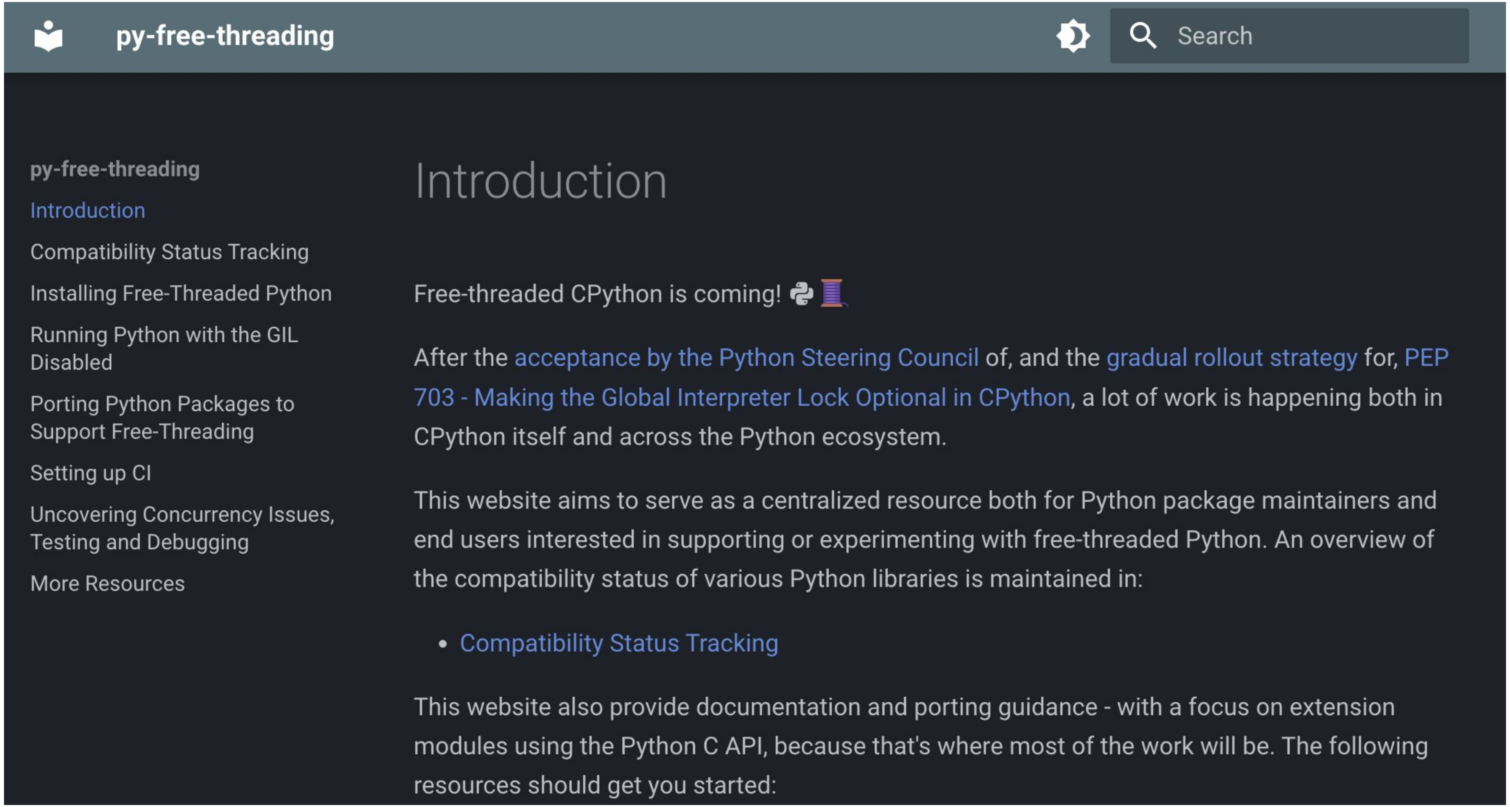
Open 14 tasks done **alex** opened this issue on Jun 20 · 62 comments

* https://github.com/numpy/numpy/issues/24755
* https://github.com/jupyterlab/jupyterlab/issues/16915
* https://github.com/PyO3/pyo3/issues/4265
* https://github.com/python/cpython/pull/105856

Check few things before using Free Threading

Free-Threading Info : https://py-free-threading.github.io

Py-free-threading



py-free-threading

Introduction

Compatibility Status Tracking

Installing Free-Threaded Python

Running Python with the GIL Disabled

Porting Python Packages to Support Free-Threading

Setting up CI

Uncovering Concurrency Issues, Testing and Debugging

More Resources

Introduction

Free-threaded CPython is coming! 🐍🧵

After the acceptance by the Python Steering Council of, and the gradual rollout strategy for, PEP 703 - Making the Global Interpreter Lock Optional in CPython, a lot of work is happening both in CPython itself and across the Python ecosystem.

This website aims to serve as a centralized resource both for Python package maintainers and end users interested in supporting or experimenting with free-threaded Python. An overview of the compatibility status of various Python libraries is maintained in:

- Compatibility Status Tracking

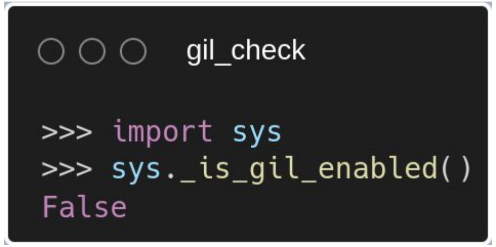
This website also provide documentation and porting guidance - with a focus on extension modules using the Python C API, because that's where most of the work will be. The following resources should get you started:

Check few things before using Free Threading

Free-Threading Info : https://py-free-threading.github.io

All of the underlying libraries supports Free Threading ?

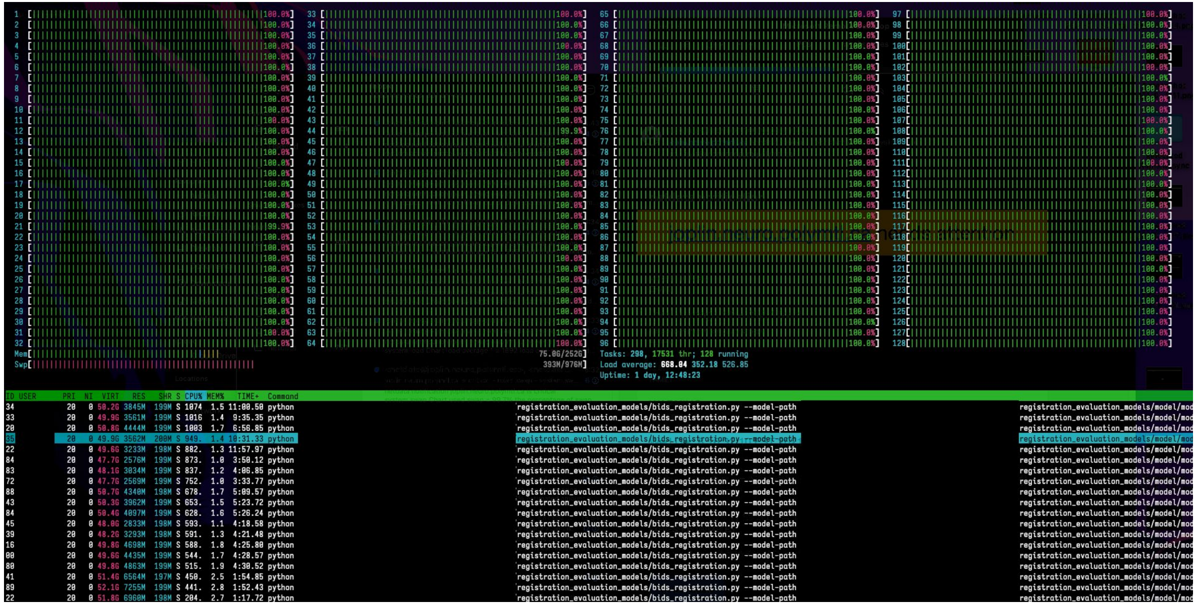
Test Yourself using : pytest-run-parallel / pytest-freethreaded



```
>>> import sys
>>> sys._is_gil_enabled()
False
```

* https://github.com/Quansight-Labs/pytest-run-parallel
* https://github.com/tonybaloney/pytest-freethreaded

Thank You



```
  1 [||||||||||||||||||||100.0%]  33 [||||||||||||||||||||100.0%]  65 [||||||||||||||||||||100.0%]  97 [||||||||||||||||||||100.0%]
  2 [||||||||||||||||||||100.0%]  34 [||||||||||||||||||||100.0%]  66 [||||||||||||||||||||100.0%]  98 [||||||||||||||||||||100.0%]
  3 [||||||||||||||||||||100.0%]  35 [||||||||||||||||||||100.0%]  67 [||||||||||||||||||||100.0%]  99 [||||||||||||||||||||100.0%]
  4 [||||||||||||||||||||100.0%]  36 [||||||||||||||||||||100.0%]  68 [||||||||||||||||||||100.0%]  100[||||||||||||||||||||100.0%]
  5 [||||||||||||||||||||100.0%]  37 [||||||||||||||||||||100.0%]  69 [||||||||||||||||||||100.0%]  101[||||||||||||||||||||100.0%]
  6 [||||||||||||||||||||100.0%]  38 [||||||||||||||||||||100.0%]  70 [||||||||||||||||||||100.0%]  102[||||||||||||||||||||100.0%]
  7 [||||||||||||||||||||100.0%]  39 [||||||||||||||||||||100.0%]  71 [||||||||||||||||||||100.0%]  103[||||||||||||||||||||100.0%]
  8 [||||||||||||||||||||100.0%]  40 [||||||||||||||||||||100.0%]  72 [||||||||||||||||||||100.0%]  104[||||||||||||||||||||100.0%]
  9 [||||||||||||||||||||100.0%]  41 [||||||||||||||||||||100.0%]  73 [||||||||||||||||||||100.0%]  105[||||||||||||||||||||100.0%]
 10 [||||||||||||||||||||100.0%]  42 [||||||||||||||||||||100.0%]  74 [||||||||||||||||||||100.0%]  106[||||||||||||||||||||100.0%]
 11 [||||||||||||||||||||100.0%]  43 [||||||||||||||||||||100.0%]  75 [||||||||||||||||||||100.0%]  107[||||||||||||||||||||100.0%]
 12 [||||||||||||||||||||100.0%]  44 [|||||||||||||||||||| 99.9%]  76 [||||||||||||||||||||100.0%]  108[||||||||||||||||||||100.0%]
 13 [||||||||||||||||||||100.0%]  45 [||||||||||||||||||||100.0%]  77 [||||||||||||||||||||100.0%]  109[||||||||||||||||||||100.0%]
 14 [||||||||||||||||||||100.0%]  46 [||||||||||||||||||||100.0%]  78 [||||||||||||||||||||100.0%]  110[||||||||||||||||||||100.0%]
 15 [||||||||||||||||||||100.0%]  47 [||||||||||||||||||||100.0%]  79 [||||||||||||||||||||100.0%]  111[||||||||||||||||||||100.0%]
 16 [||||||||||||||||||||100.0%]  48 [||||||||||||||||||||100.0%]  80 [||||||||||||||||||||100.0%]  112[||||||||||||||||||||100.0%]
 17 [||||||||||||||||||||100.0%]  49 [||||||||||||||||||||100.0%]  81 [||||||||||||||||||||100.0%]  113[||||||||||||||||||||100.0%]
 18 [||||||||||||||||||||100.0%]  50 [||||||||||||||||||||100.0%]  82 [||||||||||||||||||||100.0%]  114[||||||||||||||||||||100.0%]
 19 [||||||||||||||||||||100.0%]  51 [||||||||||||||||||||100.0%]  83 [||||||||||||||||||||100.0%]  115[||||||||||||||||||||100.0%]
 20 [||||||||||||||||||||100.0%]  52 [||||||||||||||||||||100.0%]  84 [||||||||||||||||||||100.0%]  116[||||||||||||||||||||100.0%]
 21 [|||||||||||||||||||| 99.9%]  53 [||||||||||||||||||||100.0%]  85 [||||||||||||||||||||100.0%]  117[||||||||||||||||||||100.0%]
 22 [||||||||||||||||||||100.0%]  54 [||||||||||||||||||||100.0%]  86 [||||||||||||||||||||100.0%]  118[||||||||||||||||||||100.0%]
 23 [||||||||||||||||||||100.0%]  55 [||||||||||||||||||||100.0%]  87 [||||||||||||||||||||100.0%]  119[||||||||||||||||||||100.0%]
 24 [||||||||||||||||||||100.0%]  56 [||||||||||||||||||||100.0%]  88 [||||||||||||||||||||100.0%]  120[||||||||||||||||||||100.0%]
 25 [||||||||||||||||||||100.0%]  57 [||||||||||||||||||||100.0%]  89 [||||||||||||||||||||100.0%]  121[||||||||||||||||||||100.0%]
 26 [||||||||||||||||||||100.0%]  58 [||||||||||||||||||||100.0%]  90 [||||||||||||||||||||100.0%]  122[||||||||||||||||||||100.0%]
 27 [||||||||||||||||||||100.0%]  59 [||||||||||||||||||||100.0%]  91 [||||||||||||||||||||100.0%]  123[||||||||||||||||||||100.0%]
 28 [||||||||||||||||||||100.0%]  60 [||||||||||||||||||||100.0%]  92 [||||||||||||||||||||100.0%]  124[||||||||||||||||||||100.0%]
 29 [||||||||||||||||||||100.0%]  61 [||||||||||||||||||||100.0%]  93 [||||||||||||||||||||100.0%]  125[||||||||||||||||||||100.0%]
 30 [||||||||||||||||||||100.0%]  62 [||||||||||||||||||||100.0%]  94 [||||||||||||||||||||100.0%]  126[||||||||||||||||||||100.0%]
 31 [||||||||||||||||||||100.0%]  63 [||||||||||||||||||||100.0%]  95 [||||||||||||||||||||100.0%]  127[||||||||||||||||||||100.0%]
 32 [||||||||||||||||||||100.0%]  64 [||||||||||||||||||||100.0%]  96 [||||||||||||||||||||100.0%]  128[||||||||||||||||||||100.0%]
 Mem[||||||||||||||||||||                      75.0G/252G]  Tasks: 298, 17531 thr; 128 running
 Swp[||||||||||||||||||||                       393M/976M]  Load average: 668.04 352.18 526.85
                                                             Uptime: 1 day, 12:48:23

ID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
34            20   0 50.2G 3845M  199M S 1074  1.5 11:00.50 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
33            20   0 49.9G 3561M  199M S 1016  1.4  9:35.35 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
20            20   0 50.8G 4444M  199M S 1003  1.7  6:56.85 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
35            20   0 49.9G 3562M  200M S 949.  1.4 10:31.33 python   registration_evaluation_models/bids_registration.py --model-path    registration_evaluation_models/model/mod
22            20   0 49.6G 3233M  198M S 882.  1.3 11:57.97 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
84            20   0 47.7G 2576M  199M S 873.  1.0  3:50.12 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
83            20   0 48.1G 3034M  199M S 837.  1.2  4:06.85 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
72            20   0 47.7G 2569M  199M S 752.  1.0  3:33.77 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
88            20   0 50.7G 4340M  198M S 678.  1.7  5:09.57 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
43            20   0 50.3G 3962M  199M S 653.  1.5  5:23.72 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
84            20   0 50.4G 4097M  199M S 628.  1.6  5:26.24 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
45            20   0 48.0G 2833M  198M S 593.  1.1  4:18.58 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
39            20   0 48.2G 3293M  198M S 591.  1.3  4:21.48 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
16            20   0 49.8G 4698M  199M S 588.  1.8  4:25.80 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
00            20   0 49.6G 4435M  199M S 544.  1.7  4:28.57 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
80            20   0 49.8G 4863M  199M S 515.  1.9  4:30.52 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
41            20   0 51.4G 6564M  197M S 450.  2.5  1:54.85 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
89            20   0 52.1G 7255M  199M S 441.  2.8  1:52.43 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
22            20   0 51.8G 6960M  198M S 204.  2.7  1:17.72 python   'registration_evaluation_models/bids_registration.py --model-path   registration_evaluation_models/model/mod
```

https://www.linkedin.com/in/shekharkoirala/