
GIL

Google: can Python into threads?

"CPython doesn't support multi-threading"

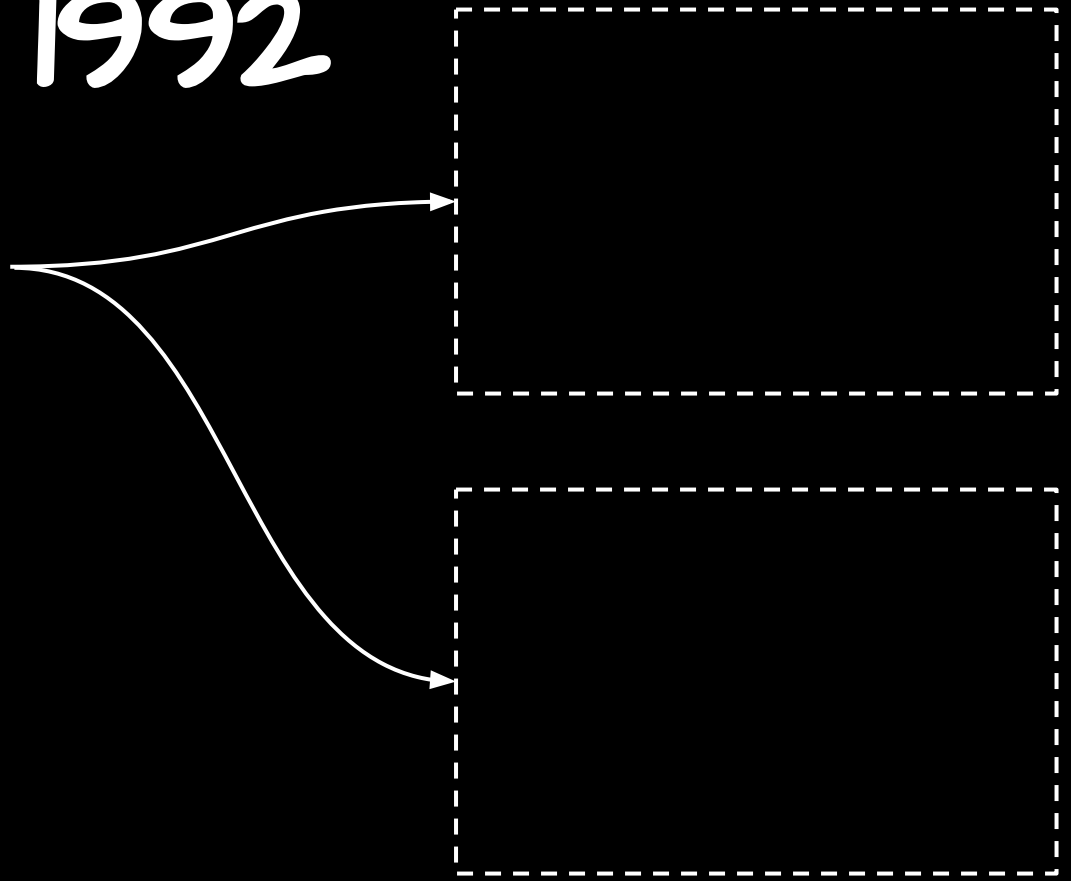
1992

1992

- me

1992

- me
- Java & JavaScript

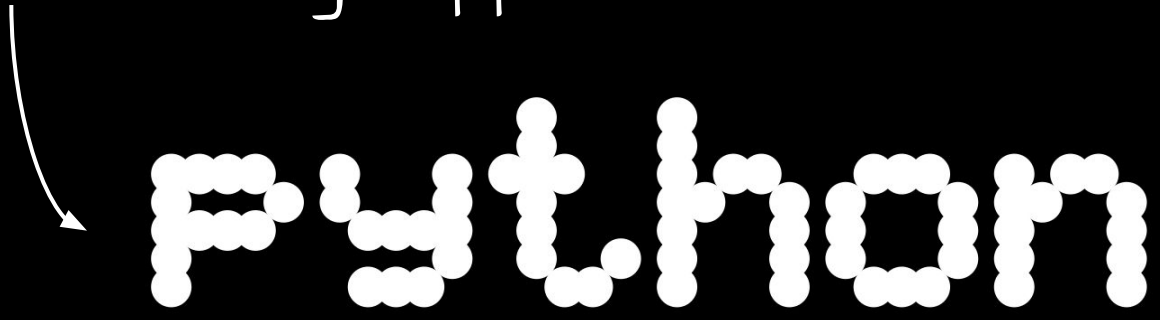


1992

- me
- Java & JavaScript
- consumer-grade multi-core CPUs

1992

- me
- Java & JavaScript
- consumer-grade multi-core CPUs
- Python (with threading support)



× „CPython doesn't support multi-threading"

× „CPython doesn't support multi-threading"

„CPython can run only on single core"

- × „CPython doesn't support multi-threading"
- × „CPython can run only on single core"

- × „CPython doesn't support multi-threading"
- × „CPython can run only on single core"

  "CPython process can execute Python bytecode in one thread at the time"

× „CPython doesn't support multi-threading"
× „CPython can run only on single core"
✓ "CPython process can execute Python bytecode in one thread at the time"

kolodziejj.info
@unit03


allegro


allegro

GIL

GIL: prerequisites

Threads

Threads

- process

Threads

- process:
  - instance of an application

Threads

- process:
  - instance of an application
  - one or more threads

Threads

- process:
  - instance of an application
  - one or more threads
  - shared memory

Threads

- process:
  - instance of an application
  - one or more threads
  - shared memory
- `import threading`

Threads

- process:
  - instance of an application
  - one or more threads
  - shared memory
- `import threading`
  - system (kernel/native/Posix) threads

Threads

- process:
  - instance of an application
  - one or more threads
  - shared memory
- `import threading`
  - system (kernel/native/Posix) threads
  - vs. green threads, greenlets, coroutines etc.

Threads

- process:
  - instance of an application
  - one or more threads
  - shared memory
- `import threading`
  - system (kernel/native/Posix) threads
  - vs. green threads, greenlets, coroutines etc.
- thread state:
  - ready/runnable
  - running
  - blocked/waiting (e.g. for system calls, e.g. I/O)

Parallelism

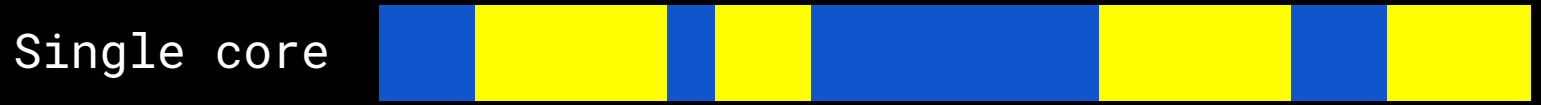
- parallelism - a form of concurrency

- parallelism - a form of concurrency

- parallelism - a form of concurrency

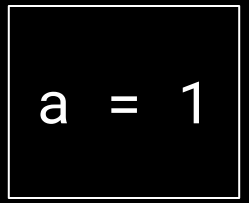
- parallelism - a form of concurrency

- other forms:



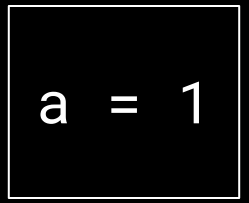
Race conditions and locks

Race conditions



Race conditions

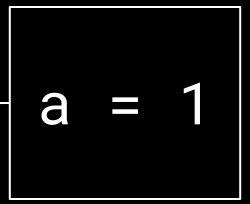
```
Thread 0
a = a - 1
```




```
Thread 1
a = a - 1
```

Race conditions

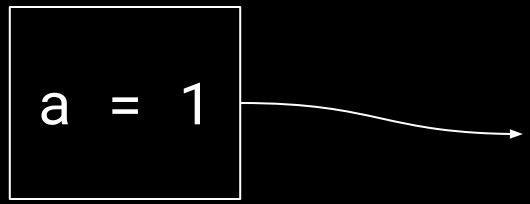
```
Thread 0
a = a - 1
a = 1 - 1
```




```
Thread 1
a = a - 1
```

Race conditions

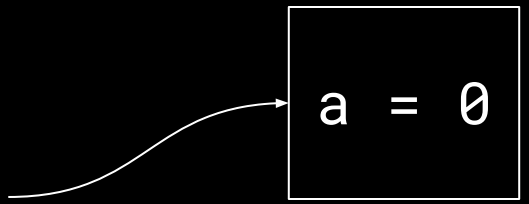
```
Thread 0
a = a - 1
a = 1 - 1
```




```
Thread 1
a = a - 1
a = 1 - 1
```

Race conditions

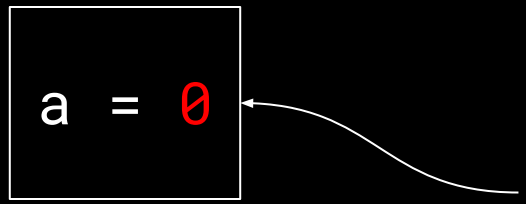
```
Thread 0
a = a - 1
a = 1 - 1
a = 0
```




```
Thread 1
a = a - 1
a = 1 - 1
```

Race conditions

```
Thread 0
a = a - 1
a = 1 - 1
a = 0
```




```
Thread 1
a = a - 1
a = 1 - 1
a = 0
```

Locks

Locks

- or: mutex - **mut**ual **ex**clusion

Locks

- or: mutex - **mut**ual **ex**clusion
- tool for preventing race-conditions

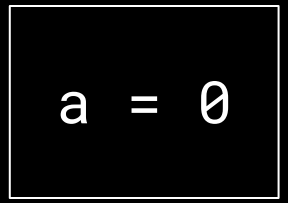
Locks

- or: mutex - **mut**ual **ex**clusion
- tool for preventing race-conditions
- lock (hold) and unlock (release)

Locks

```
lock = threading.Lock()
```

```
Thread 0
with lock:
    a = a - 1
```

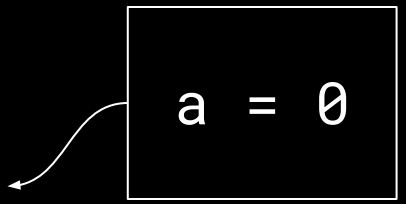



```
Thread 1
with lock:
    a = a - 1
```

Locks

```
lock = threading.Lock()
```

```
Thread 0
with lock:
    a = a - 1
    a = 1 - 1
```

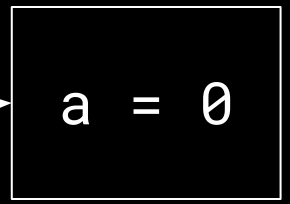



```
Thread 1
with lock:
    a = a - 1
```

Locks

```
lock = threading.Lock()
```

```
Thread 0
with lock:
    a = a - 1
    a = 1 - 1
    a = 0
```

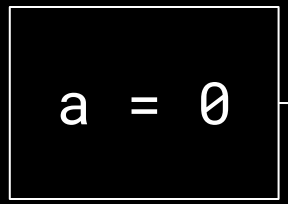



```
Thread 1
with lock:
    a = a - 1
```

Locks

```
lock = threading.Lock()
```

```
Thread 0
with lock:
    a = a - 1
    a = 1 - 1
    a = 0
```

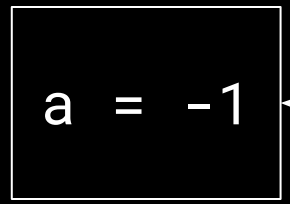



```
Thread 1
with lock:
    a = a - 1
    a = 0 - 1
```

Locks

```
lock = threading.Lock()
```

```
Thread 0
with lock:
    a = a - 1
    a = 1 - 1
    a = 0
```




```
Thread 1
with lock:
    a = a - 1
    a = 0 - 1
    a = -1
```

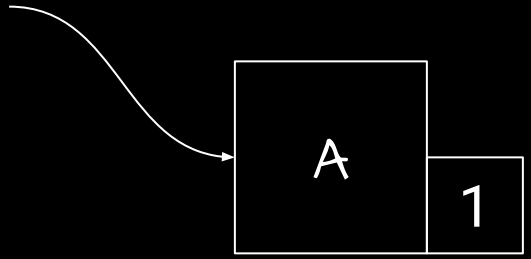
Memory management

Memory management

- CPython: reference counting

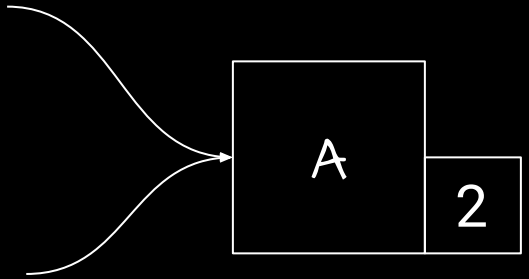
Memory management

- CPython: reference counting



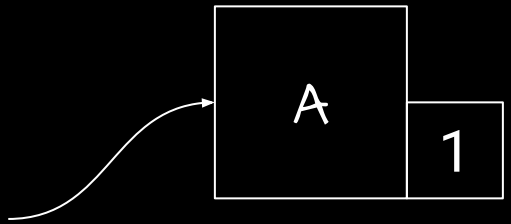
Memory management

- CPython: reference counting



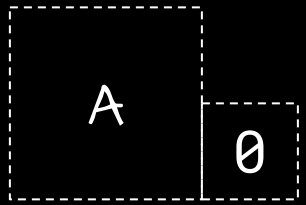
Memory management

- CPython: reference counting



Memory management

- CPython: reference counting

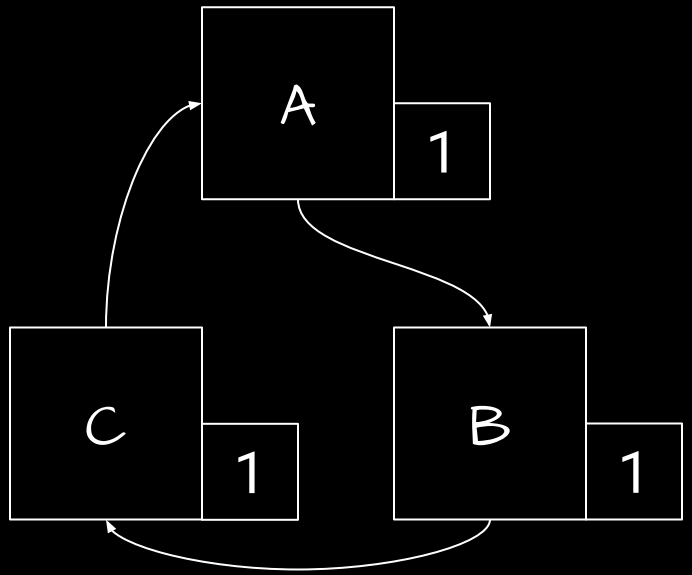


Memory management

- CPython: reference counting
  - shared resources - need locking!

Memory management

- CPython: reference counting
  - shared resources - need locking!
  - + garbage collector **for circular references**



Memory management

- CPython: reference counting
  - shared resources - need locking!
  - + garbage collector **for circular references**
- JVM, C#, ...: **tracing** garbage collectors

Memory management

- CPython: reference counting
  - shared resources - need locking!
  - + garbage collector **for circular references**
- JVM, C#, ...: **tracing** garbage collectors
  - more **robust**

Memory management

- CPython: reference counting
  - shared resources - need locking!
  - + garbage collector **for circular references**
- JVM, C#, ...: **tracing** garbage collectors
  - more **robust**
  - but more **complex**

Python bytecode

Python bytecode

- Python code -> compilation -> bytecode

Python bytecode

- Python code -> compilation -> bytecode

```
a = 1

print(a)
```

Python bytecode

- Python code -> compilation -> bytecode

```
a = 1                   2 LOAD_CONST    1 (1)
                        4 STORE_FAST    0 (a)
print(a)                6 LOAD_GLOBAL   1 (NULL + print)
                       18 LOAD_FAST     0 (a)
                       20 PRECALL       1
                       24 CALL          1
```

Python bytecode

- Python code -> compilation -> bytecode

```
a = 1                   2 LOAD_CONST    1 (1)
                        4 STORE_FAST    0 (a)
print(a)                6 LOAD_GLOBAL   1 (NULL + print)
                       18 LOAD_FAST     0 (a)
                       20 PRECALL       1
                       24 CALL          1
```

- bytecode -> interpreter -> execution


finally
GIL

GIL

- Global Interpreter Lock

GIL

- Global Interpreter Lock
- protects against race conditions

GIL

- Global Interpreter Lock
- protects against race conditions:
  - reference counters

GIL

- Global Interpreter Lock
- protects against race conditions:
  - reference counters
  - "C-mutable" Python data structures (dicts, lists, ...)

GIL

- Global Interpreter Lock
- protects against race conditions:
  - reference counters
  - "C-mutable" Python data structures (dicts, lists, strings, tuples, integers etc.)

GIL

- Global Interpreter Lock
- protects against race conditions:
  - reference counters
  - "C-mutable" Python data structures (dicts, lists, strings, tuples, integers etc.)
  - internal global state

GIL

- Global Interpreter Lock
- protects against race conditions:
  - reference counters
  - "C-mutable" Python data structures (dicts, lists, strings, tuples, integers etc.)
  - internal global state
  - atomic APIs

GIL

- Global Interpreter Lock
- protects against race conditions:
  - reference counters
  - "C-mutable" Python data structures (dicts, lists, strings, tuples, integers etc.)
  - internal global state
  - atomic APIs
- C extensions/binary modules assumes GIL

GIL

- holding the GIL not needed when:

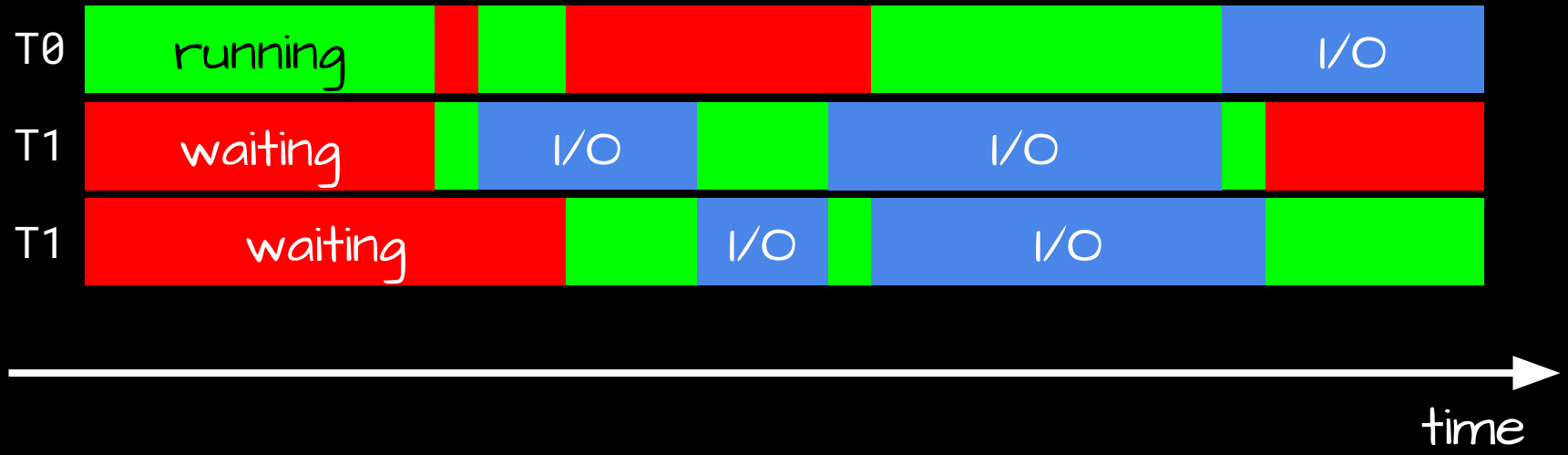
GIL

- holding the GIL not needed when:
  - waiting for I/O

GIL

- holding the GIL not needed when:
  - waiting for I/O
  - executing code that doesn't access Python objects


T0
running
T1
waiting
T1
waiting
time


T0
running
I/O
T1
waiting
I/O
I/O
T1
waiting
I/O
I/O
time

GIL: implementation

- "ceval" loop: Python/ceval.c
  - the GIL part: Python/ceval_gil.c
- GIL data structure: Include/internal/pycore_gil.h


GIL

De-GIL'ing

De-GIL'ing expectations

De-GIL'ing expectations

1. Performance:

De-GIL'ing expectations

1. Performance:
    - for multi-threaded code: must scale with number of cores

De-GIL'ing expectations

1. Performance:
   - for multi-threaded code: must scale with number of cores
   - single-threaded code: not slower

De-GIL'ing expectations

1. Performance:
   - for multi-threaded code: must scale with number of cores
   - single-threaded code: not slower
2. Compatible with C-extensions

De-GIL'ing expectations

1. Performance:
   - for multi-threaded code: must scale with number of cores
   - single-threaded code: not slower
2. Compatible with C-extensions
3. CPython codebase: not significantly more complex than with the GIL

De-GIL'ing attempts

De-GIL'ing attempts

- 1996: free-threading patch

De-GIL'ing attempts

- 1996: free-threading patch
- 2008: python-safethread

De-GIL'ing attempts

- 1996: free-threading patch
- 2008: python-safethread
- 2016: GILectomy

- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython:

- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython:
  - **Already available in 3.13!**

- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython:
  - **Already available in 3.13!**
  - changes in reference counting, garbage collection, thread-safety of container types, memory allocation... (go read the PEP)

- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython:
  - **Already available in 3.13!**
  - changes in reference counting, garbage collection, thread-safety of container types, memory allocation... (go read the PEP)
  - `--disable-gil` build-time flag, separate builds

- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython:
  - **Already available in 3.13!**
  - changes in reference counting, garbage collection, thread-safety of container types, memory allocation... (go read the PEP)
  - `--disable-gil` build-time flag, separate builds
  - https://py-free-threading.github.io/tracking/

De-GIL'ing expectations

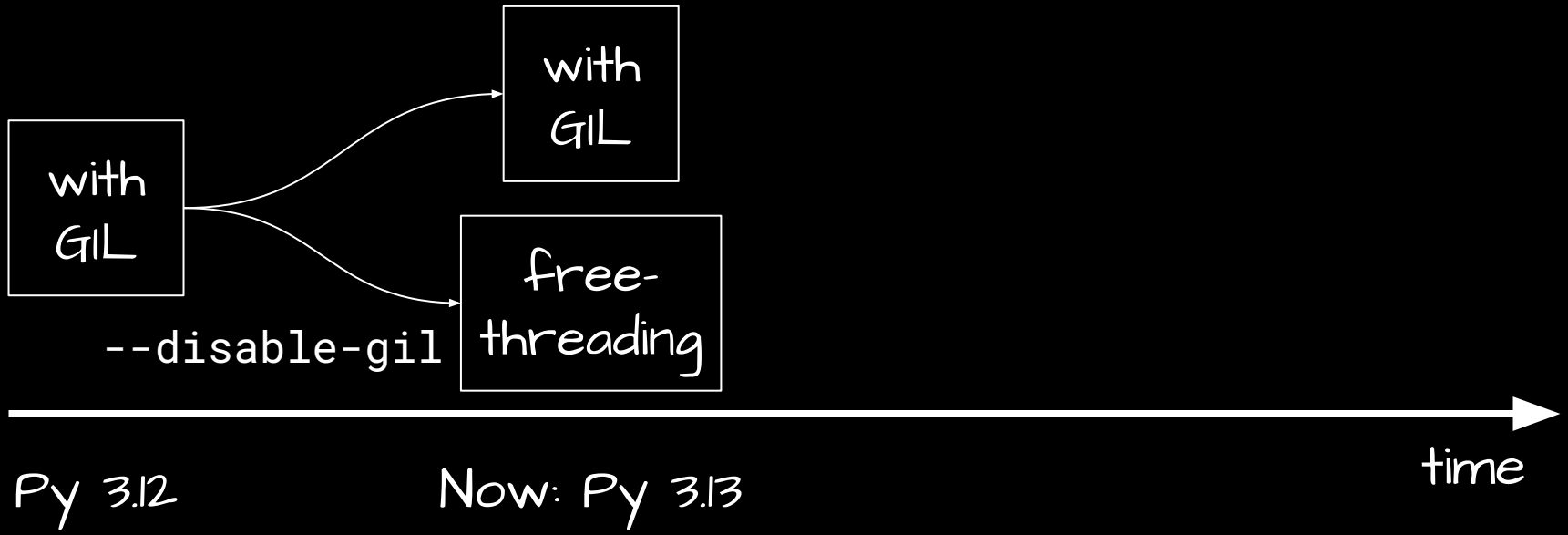
1. Performance:
   - for multi-threaded code: must scale with number of cores
   - single-threaded code: not slower
2. Compatible with C-extensions
3. CPython codebase: not significantly more complex than with the GIL

De-GIL'ing expectations

1. Performance:
   - for multi-threaded code: must scale with number of cores
   - single-threaded code: not significantly slower
2. Gradual migration path for C-extensions
3. CPython codebase: not significantly more complex than with the GIL

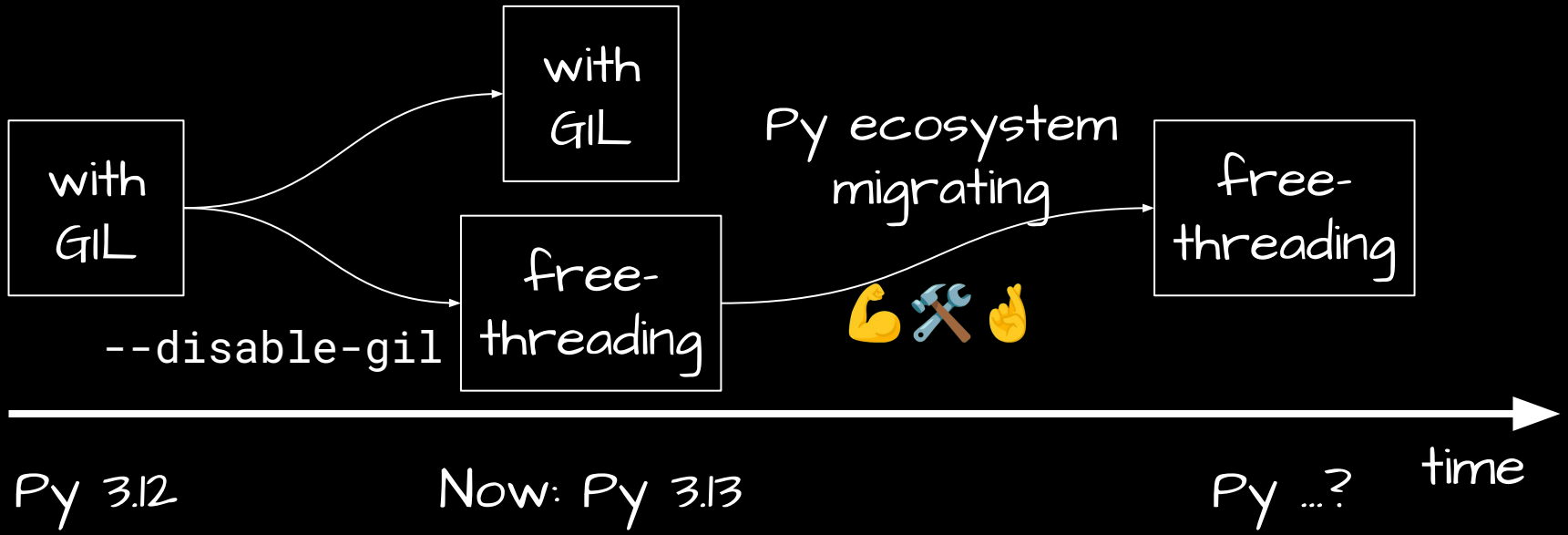
- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython

- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython



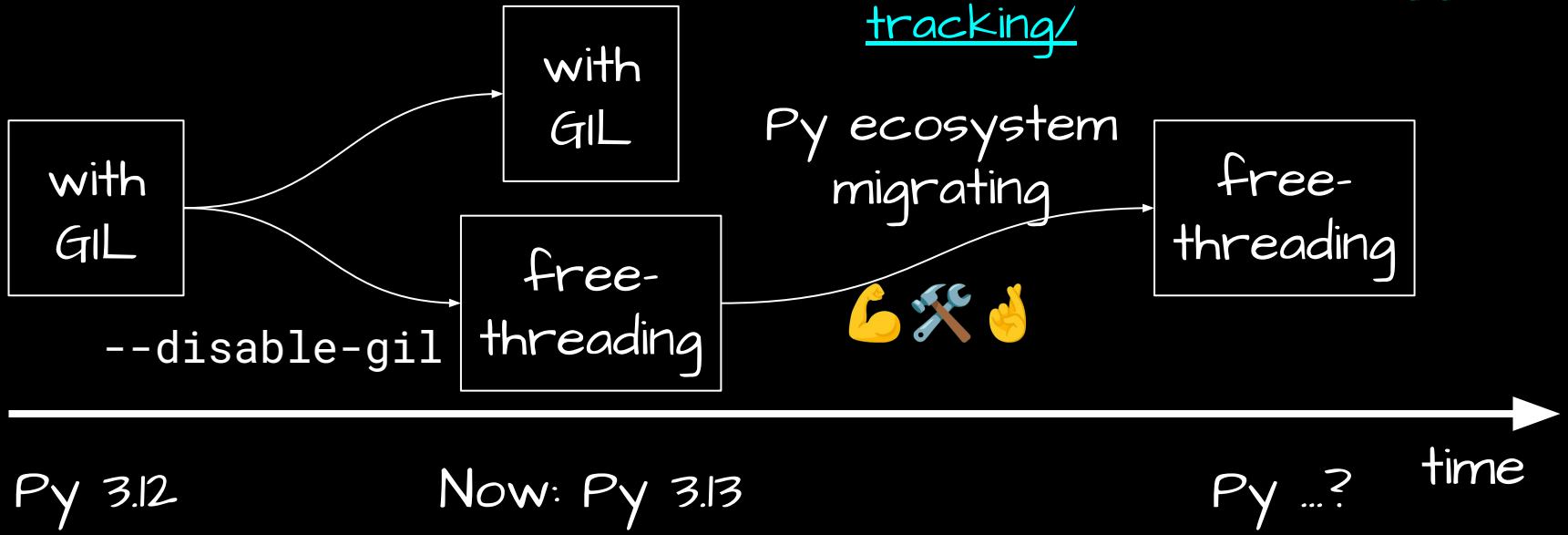
- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython

- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython



- 2023: PEP 703 - Making the Global Interpreter Lock Optional in CPython

https://py-free-threading.github.io/tracking/



In the meantime: what to do?

What to do?

- single-threaded, pure Python: nothing

What to do?

- single-threaded, pure Python: nothing
- single-threaded Python with multi-threaded C-extensions:
  - may use all the cores already
  - profile

What to do?

- single-threaded, pure Python: nothing
- single-threaded Python with multi-threaded C-extensions:
  - may use all the cores already
  - profile
- multi-threaded, I/O-bound:
  - profile
  - add more threads

What to do?

- asynchronous, I/O-bound:

What to do?

- asynchronous, I/O-bound:
  - run multiple processes

What to do?

- asynchronous, I/O-bound:
  - run multiple processes
  - profile

What to do?

- asynchronous, I/O-bound:
  - run multiple processes
  - profile
- asynchronous with thread pools for synchronous code:

What to do?

- asynchronous, I/O-bound:
  - run multiple processes
  - profile
- asynchronous with thread pools for synchronous code:
  - profile
  - add more threads

What to do?

- multi-threaded, CPU-bound:

What to do?

- multi-threaded, CPU-bound:
  - profile! (cProfile, perf...)

What to do?

- multi-threaded, CPU-bound:
  - profile!
  - benchmarks warning!

What to do?

- multi-threaded, CPU-bound:
  - profile!
  - benchmarks warning!
  - multiprocessing

What to do?

- multi-threaded, CPU-bound:
  - profile!
  - benchmarks warning!
  - multiprocessing
  - subinterpreters ✨

What to do?

- multi-threaded, CPU-bound:
  - profile!
  - benchmarks warning!
  - multiprocessing
  - subinterpreters ✨
  - C-extension for CPU-intensive work

What to do?

- multi-threaded, CPU-bound:
  - profile!
  - benchmarks warning!
  - multiprocessing
  - subinterpreters ✨
  - C-extension for CPU-intensive work
  - Cython's `nogil`

What to do?

- multi-threaded, CPU-bound:
  - profile!
  - benchmarks warning!
  - multiprocessing
  - subinterpreters ✨
  - C-extension for CPU-intensive work
  - Cython's `nogil`
  - non-Python alternatives

What to do?

- multi-threaded, CPU-bound:
  - profile!
  - benchmarks warning!
  - multiprocessing
  - subinterpreters ✨
  - C-extension for CPU-intensive work
  - Cython's `nogil`
  - non-Python alternatives
  - weigh all pros and cons!

- 🤞 for free-threaded CPython adoption

- 🤞 for free-threaded CPython adoption
- multi-threaded, CPU-bound: 🧰

- 🤞 for free-threaded CPython adoption
- multi-threaded, CPU-bound: 🧰
- otherwise: 🤷

- 🤞 for free-threaded CPython adoption
- multi-threaded, CPU-bound: 🧰
- otherwise: 🤷
- profile, profile, profile (cProfile, perf...) 👌

Thank you! :)
Questions?

kolodziejj.info/talks/gil