pycon china PyCon China · 2019

The Past and Future of GIL

Jiayuan Zhang

Who Am I?

Who Am I?

- Full-Stack Developer at IQIYI.Inc

Who Am I?

- Full-Stack Developer at IQIYI.Inc
- Python, JavaScript, Lisp and (Rust)

Who Am I?

- Full-Stack Developer at IQIYI.Inc
- Python, JavaScript, Lisp and (Rust)
- Open source contributor, werkzeug, requests, doom-emacs, etc.

Who Am I?

- Full-Stack Developer at IQIYI.Inc
- Python, JavaScript, Lisp and (Rust)
- Open source contributor, werkzeug, requests, doom-emacs, etc.
- Coding with Emacs, organizing my life with org-mode

Outline

Outline

- What is GIL

Outline

- What is GIL
- How GIL works

Outline

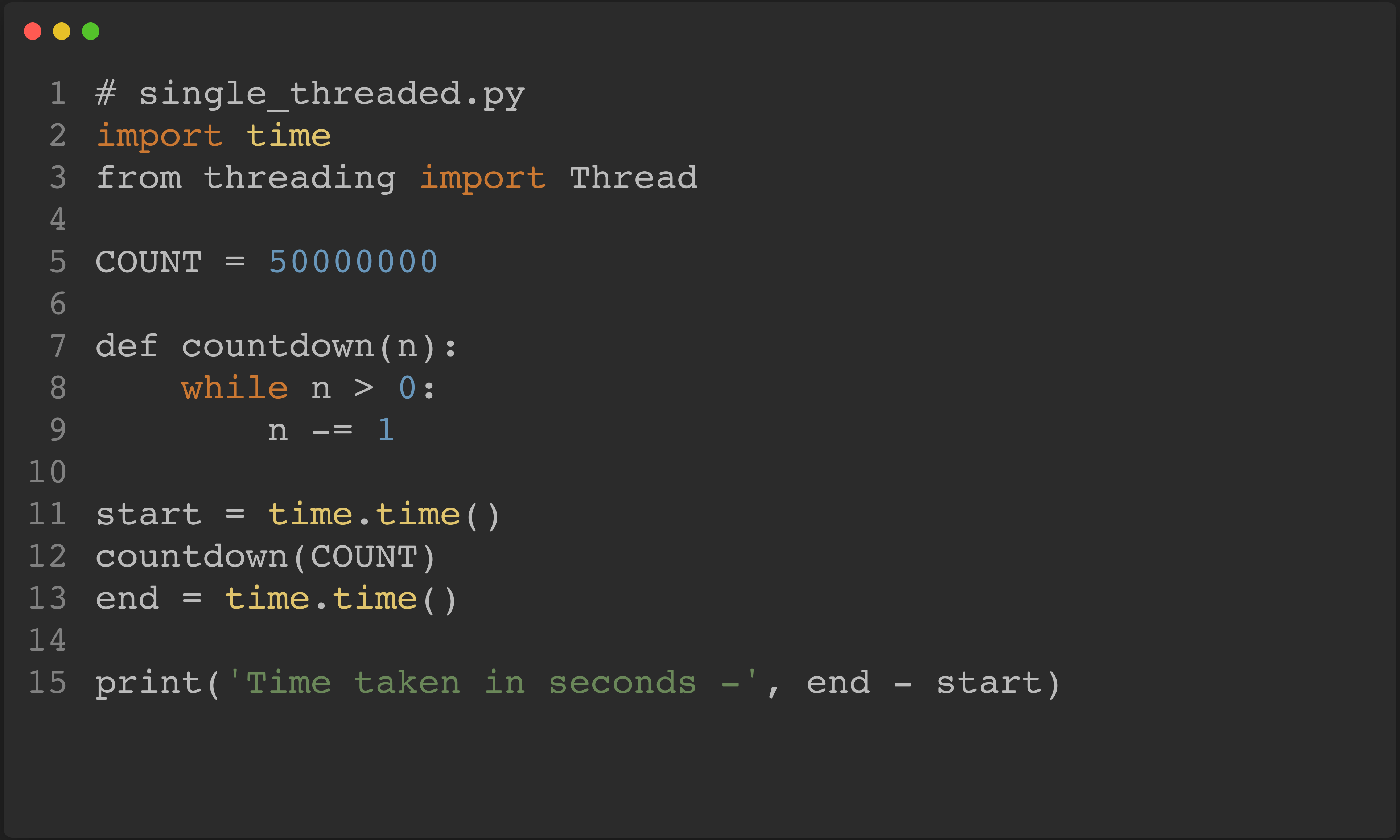
- What is GIL
- How GIL works
- Remove GIL

Outline

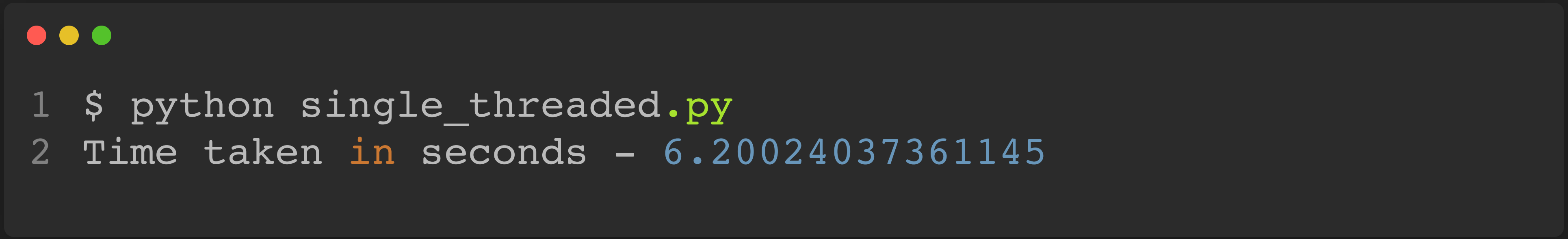
- What is GIL
- How GIL works
- Remove GIL
- The future

Part I

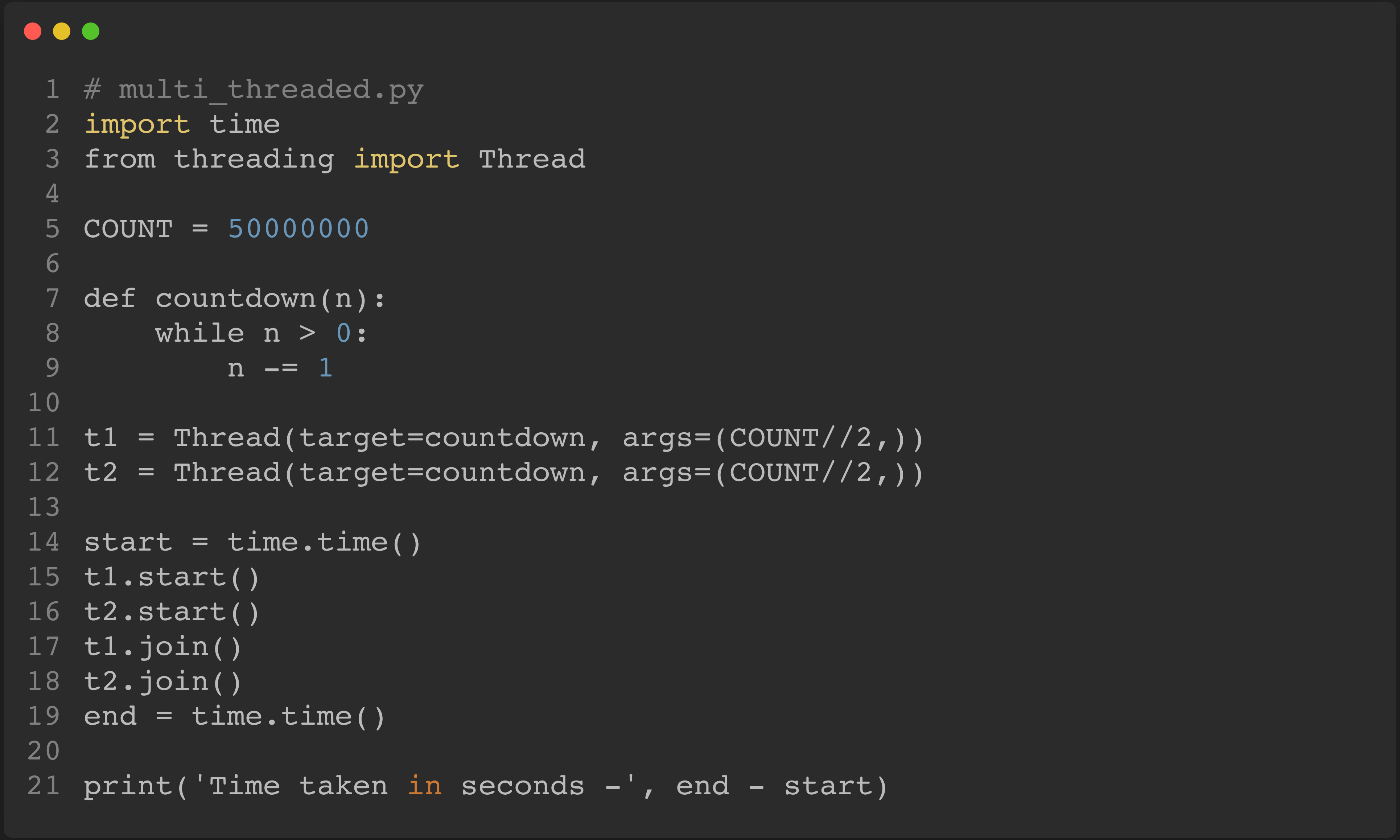
What is GIL?



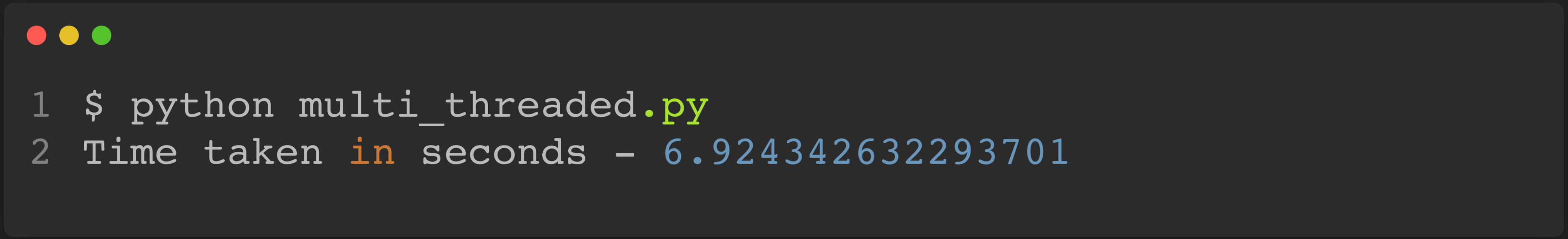
```python
# single_threaded.py
import time
from threading import Thread

COUNT = 50000000

def countdown(n):
    while n > 0:
        n -= 1

start = time.time()
countdown(COUNT)
end = time.time()

print('Time taken in seconds -', end - start)
```



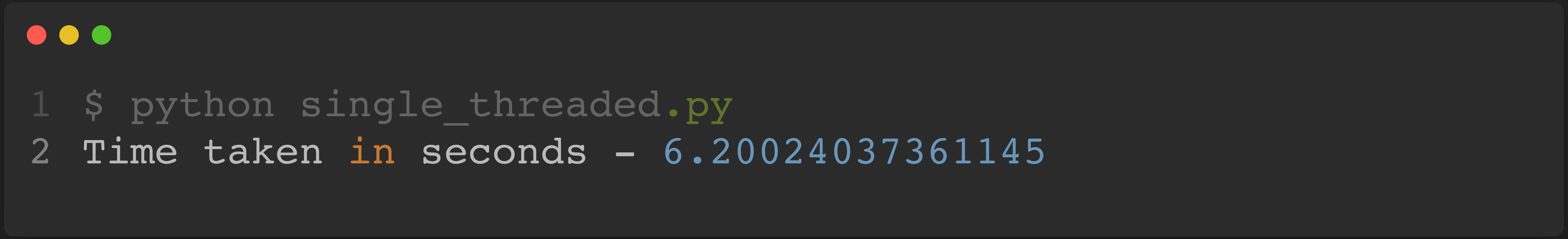
```
$ python single_threaded.py
Time taken in seconds - 6.20024037361145
```



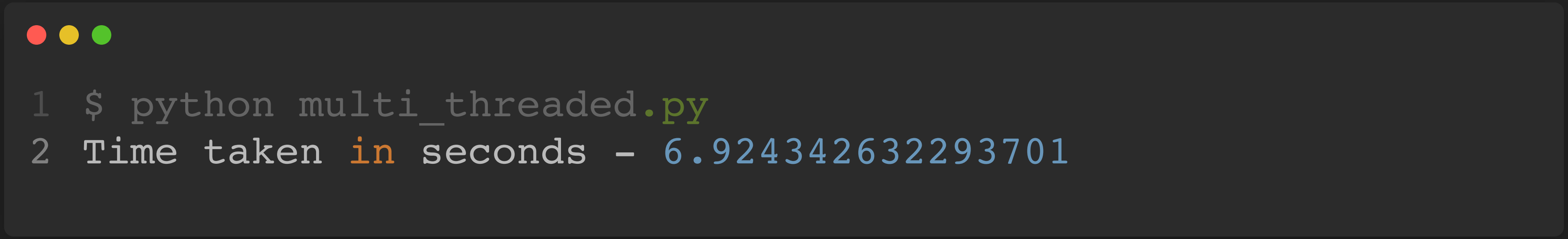
```python
# multi_threaded.py
import time
from threading import Thread

COUNT = 50000000

def countdown(n):
    while n > 0:
        n -= 1

t1 = Thread(target=countdown, args=(COUNT//2,))
t2 = Thread(target=countdown, args=(COUNT//2,))

start = time.time()
t1.start()
t2.start()
t1.join()
t2.join()
end = time.time()

print('Time taken in seconds -', end - start)
```



```
$ python multi_threaded.py
Time taken in seconds - 6.924342632293701
```



```
$ python single_threaded.py
Time taken in seconds - 6.20024037361145
```



```
$ python multi_threaded.py
Time taken in seconds - 6.924342632293701
```

What is GIL ?

What is GIL ?

- Global Interpreter Lock

What is GIL ?

- Global Interpreter Lock
- Mutex, pthread (Linux) or win thread (Windows), controlled by OS

What is GIL ?

- Global Interpreter Lock
- Mutex, pthread (Linux) or win thread (Windows), controlled by OS
- Allows only one thread to execute Python code at any point in time

What is GIL ?

- Global Interpreter Lock
- Mutex, pthread (Linux) or win thread (Windows), controlled by OS
- Allows only one thread to execute Python code at any point in time
- Bottleneck in CPU-bound and multi-threaded code

What is GIL in Depth

What is GIL in Depth

- Is Python thread safe?

What is GIL in Depth

- Is Python thread safe?
- "Lock" on what?

Is Python Thread Safe?

Is Python Thread Safe?

```
L.append(x)
L1.extend(L2)
x = L[i]
x = L.pop()
L1[i:j] = L2
L.sort()
x = y
x.field = y
D[x] = y
D1.update(D2)
D.keys()
```

Is Python Thread Safe?

```python
L.append(x)
L1.extend(L2)
x = L[i]
x = L.pop()
L1[i:j] = L2
L.sort()
x = y
x.field = y
D[x] = y
D1.update(D2)
D.keys()
```

```python
i = i+1
L.append(L[-1])
L[i] = L[j]
D[x] = D[x] + 1
```

Is Python Thread Safe?

```python
L.append(x)
L1.extend(L2)
x = L[i]
x = L.pop()
L1[i:j] = L2
L.sort()
x = y
x.field = y
D[x] = y
D1.update(D2)
D.keys()
```

Safe

```python
i = i+1
L.append(L[-1])
L[i] = L[j]
D[x] = D[x] + 1
```

Is Python Thread Safe?

```python
L.append(x)
L1.extend(L2)
x = L[i]
x = L.pop()
L1[i:j] = L2
L.sort()
x = y
x.field = y
D[x] = y
D1.update(D2)
D.keys()
```

Safe

```python
i = i+1
L.append(L[-1])
L[i] = L[j]
D[x] = D[x] + 1
```

Not Safe

Is Python Thread Safe?

```python
L.append(x)
L1.extend(L2)
x = L[i]
x = L.pop()
L1[i:j] = L2
L.sort()
x = y
x.field = y
D[x] = y
D1.update(D2)
D.keys()
```

Safe

```python
i = i+1
L.append(L[-1])
L[i] = L[j]
D[x] = D[x] + 1
```

Not Safe

What kinds of global value mutation are thread-safe?

Why?

How Python Runs?

How Python Runs?

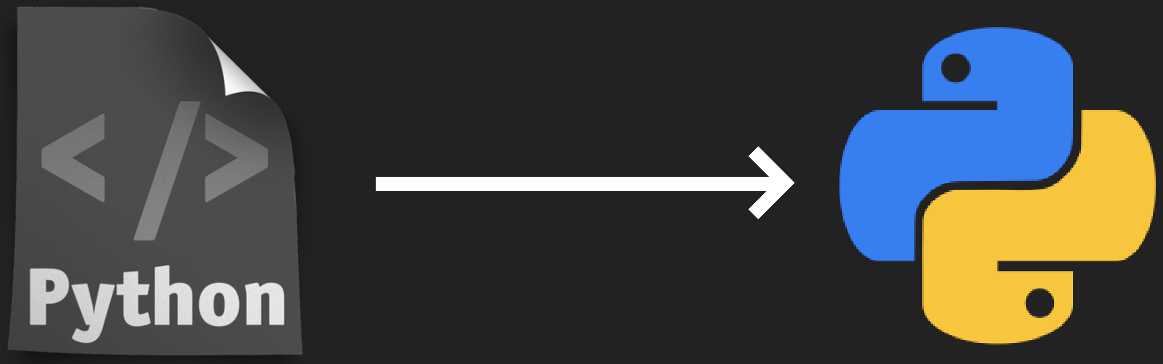
How Python Runs?

Python Source
Code (.py files)

How Python Runs?

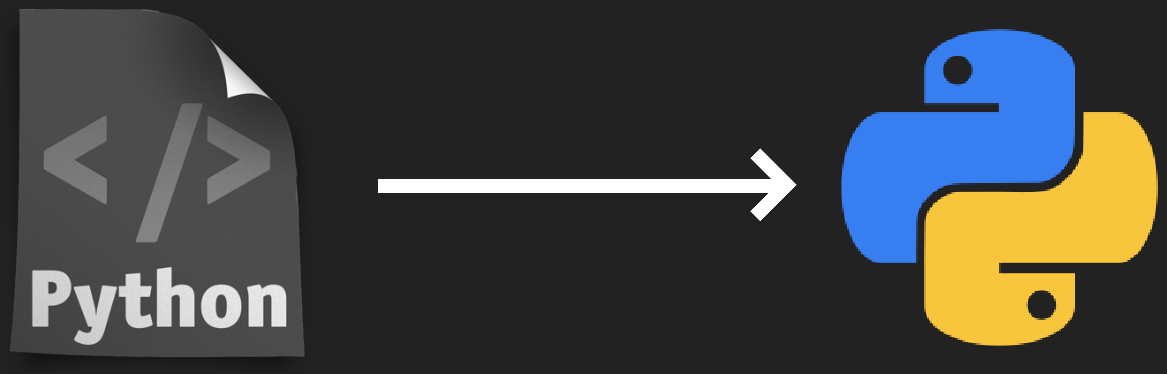
Python Source
Code (.py files)

How Python Runs?



Python Source
Code (.py files)

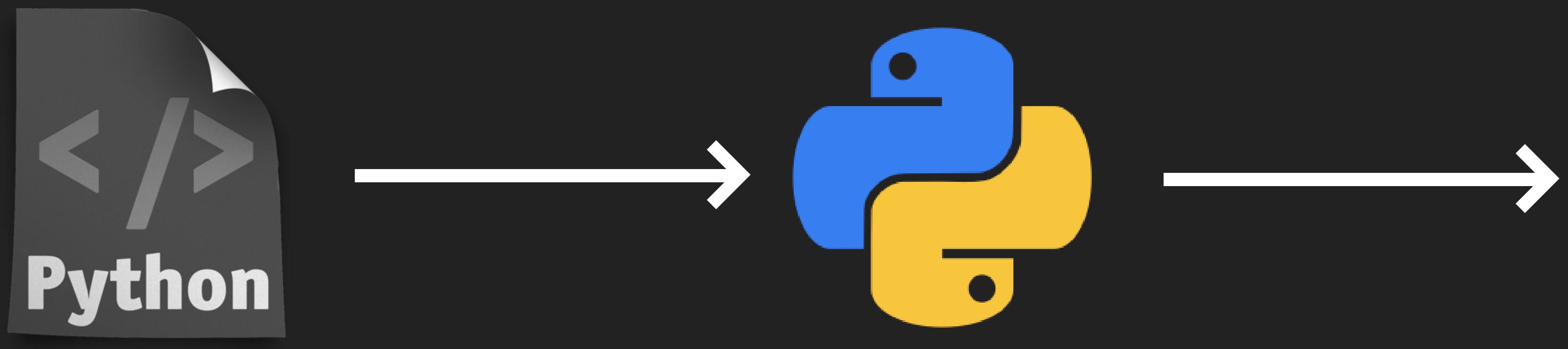
How Python Runs?



Python Source Code (.py files)

Python Interpreter

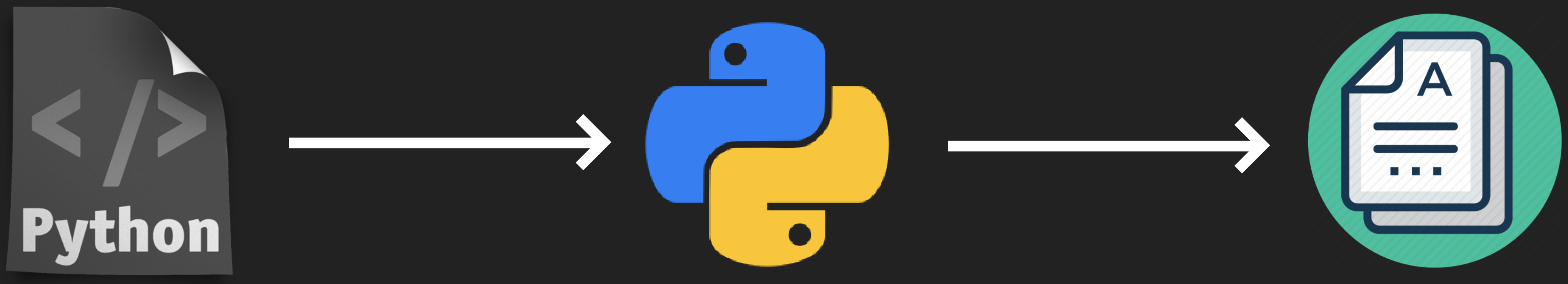
How Python Runs?



Python Source Code (.py files)

Python Interpreter

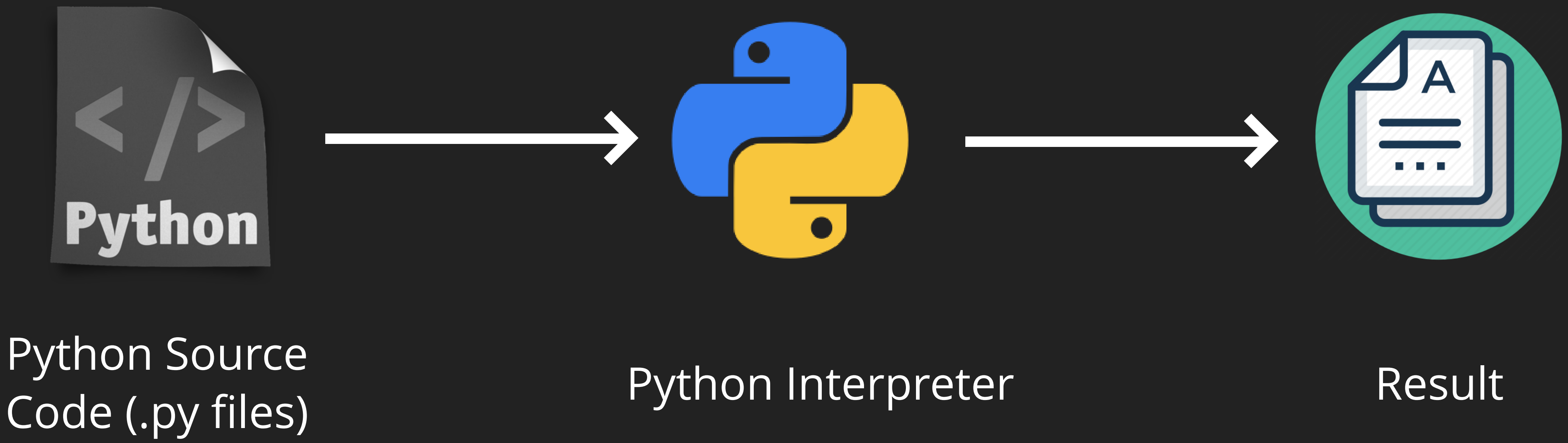
How Python Runs?



Python Source Code (.py files)

Python Interpreter

How Python Runs?



Python Source Code (.py files)

Python Interpreter

Result

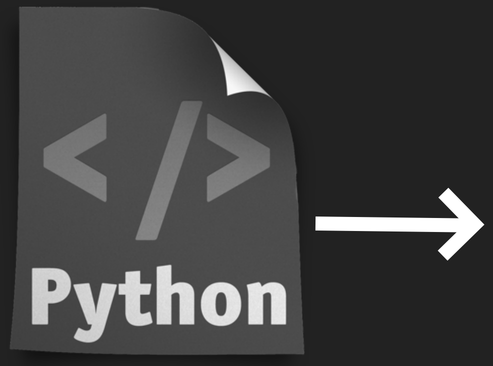
What is Python Interpreter?

What is Python Interpreter?

What is Python Interpreter?

Python Source
Code (.py files)

What is Python Interpreter?

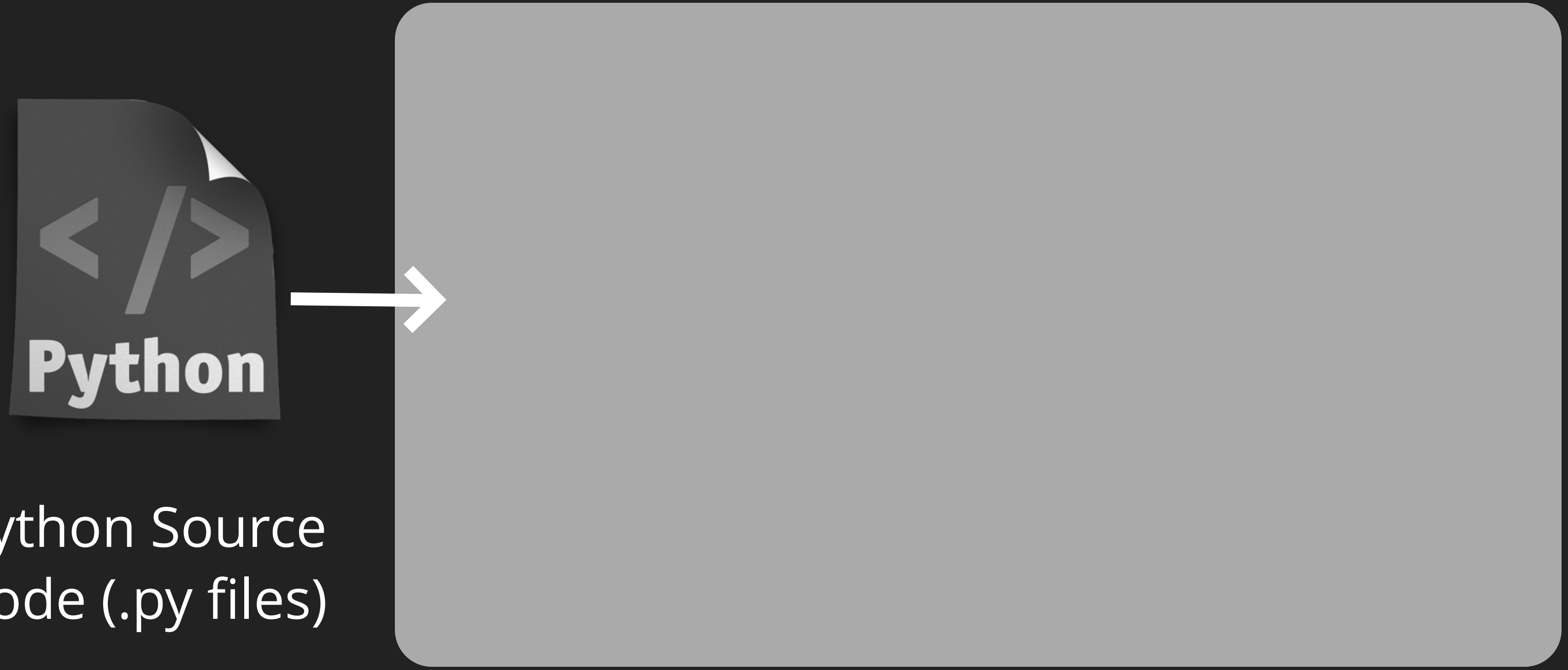


Python Source
Code (.py files)

What is Python Interpreter?

Python Source Code (.py files)

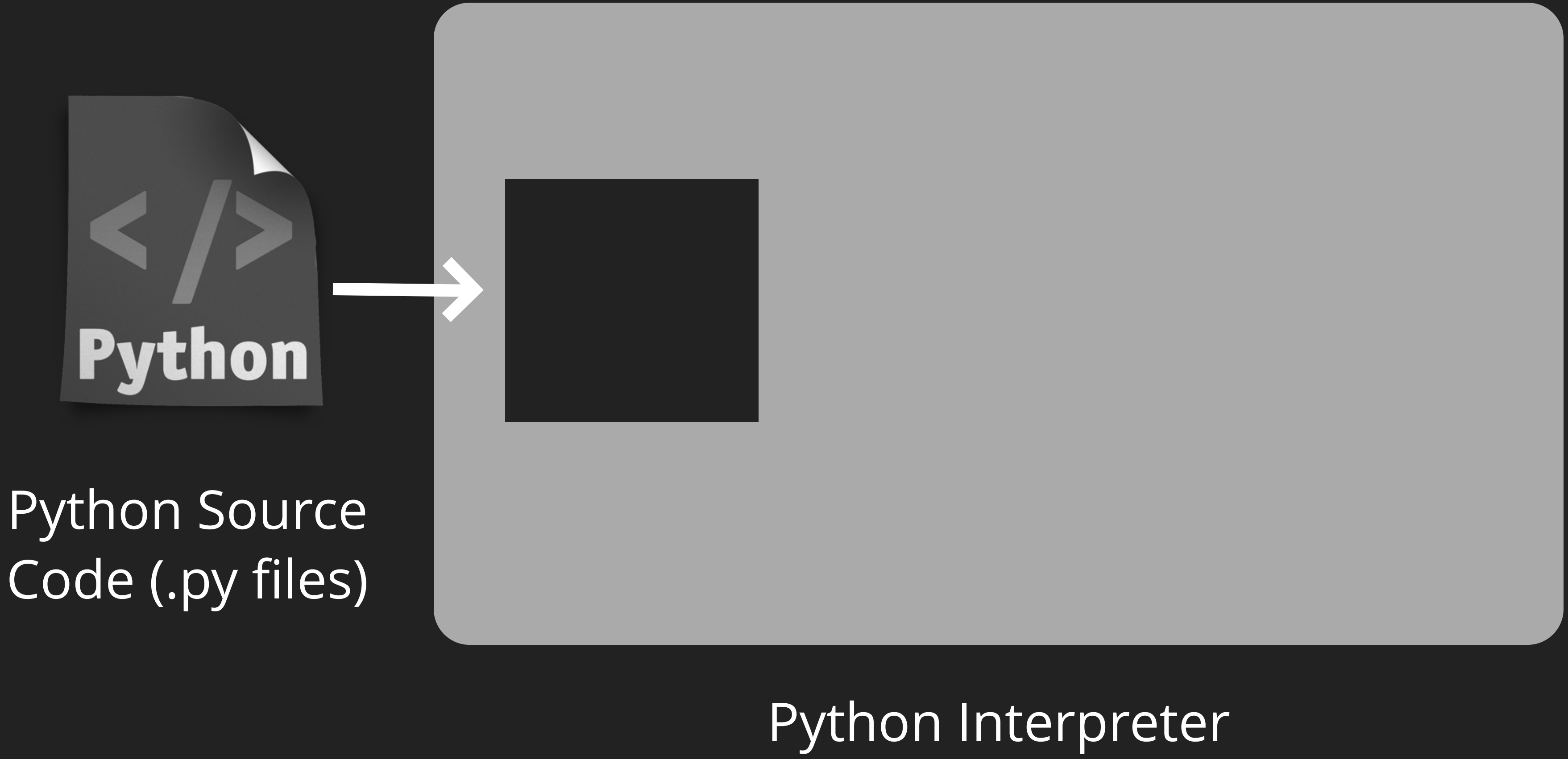
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

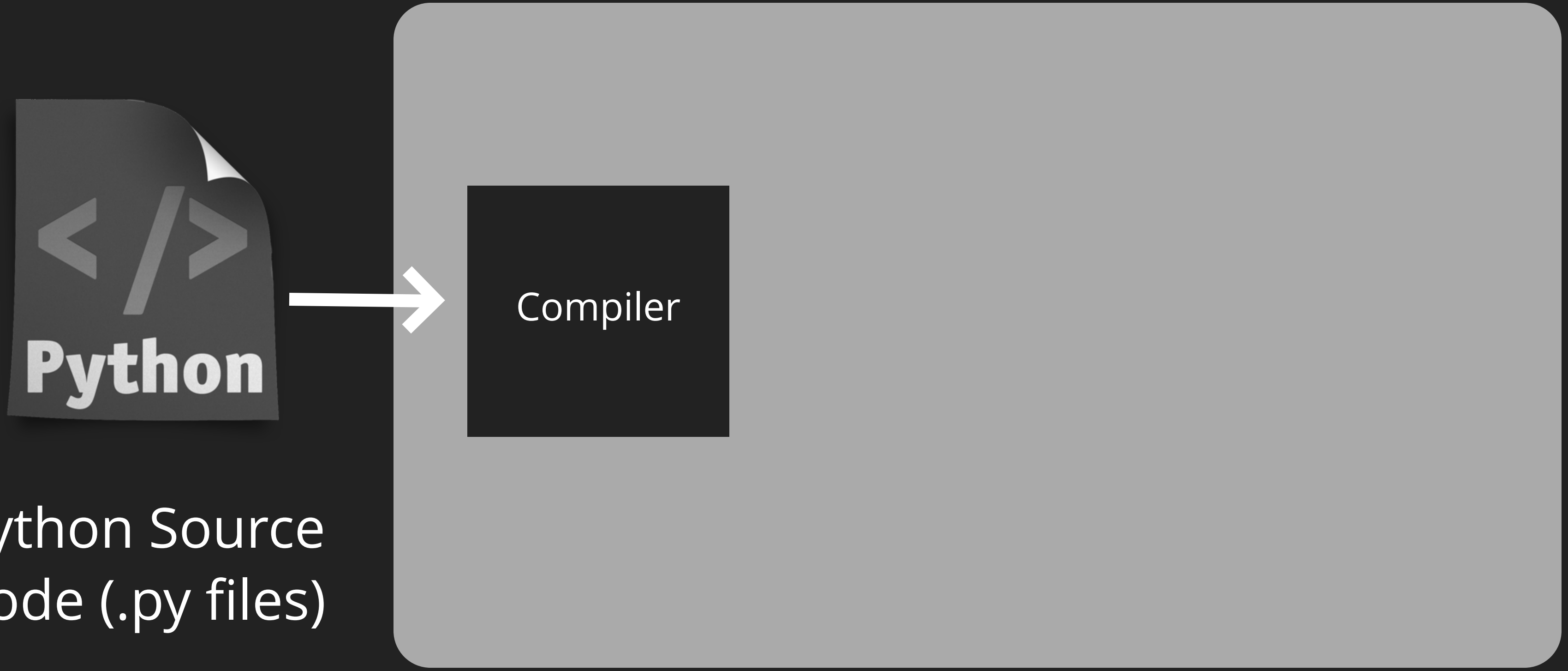
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

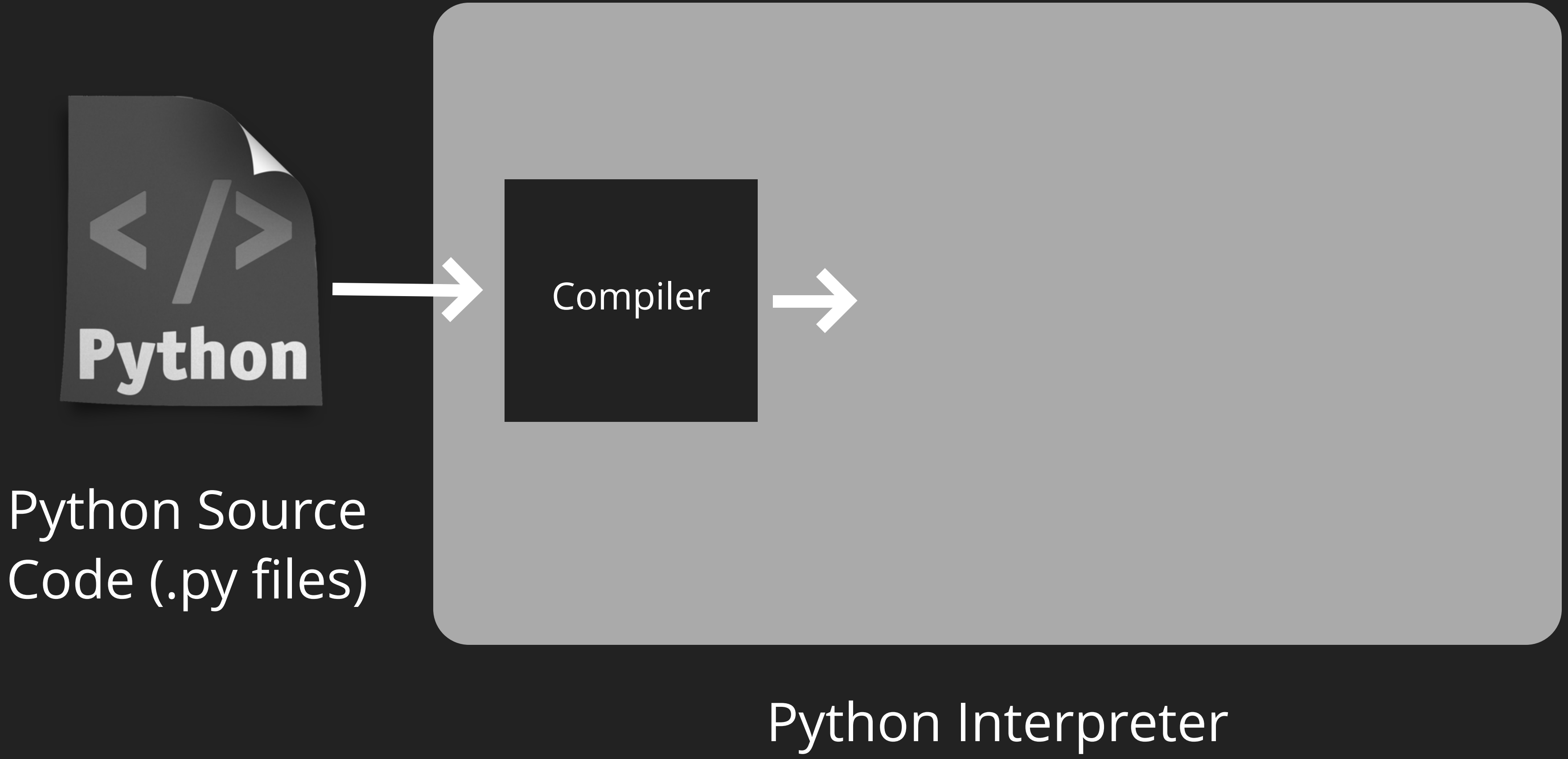
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

What is Python Interpreter?

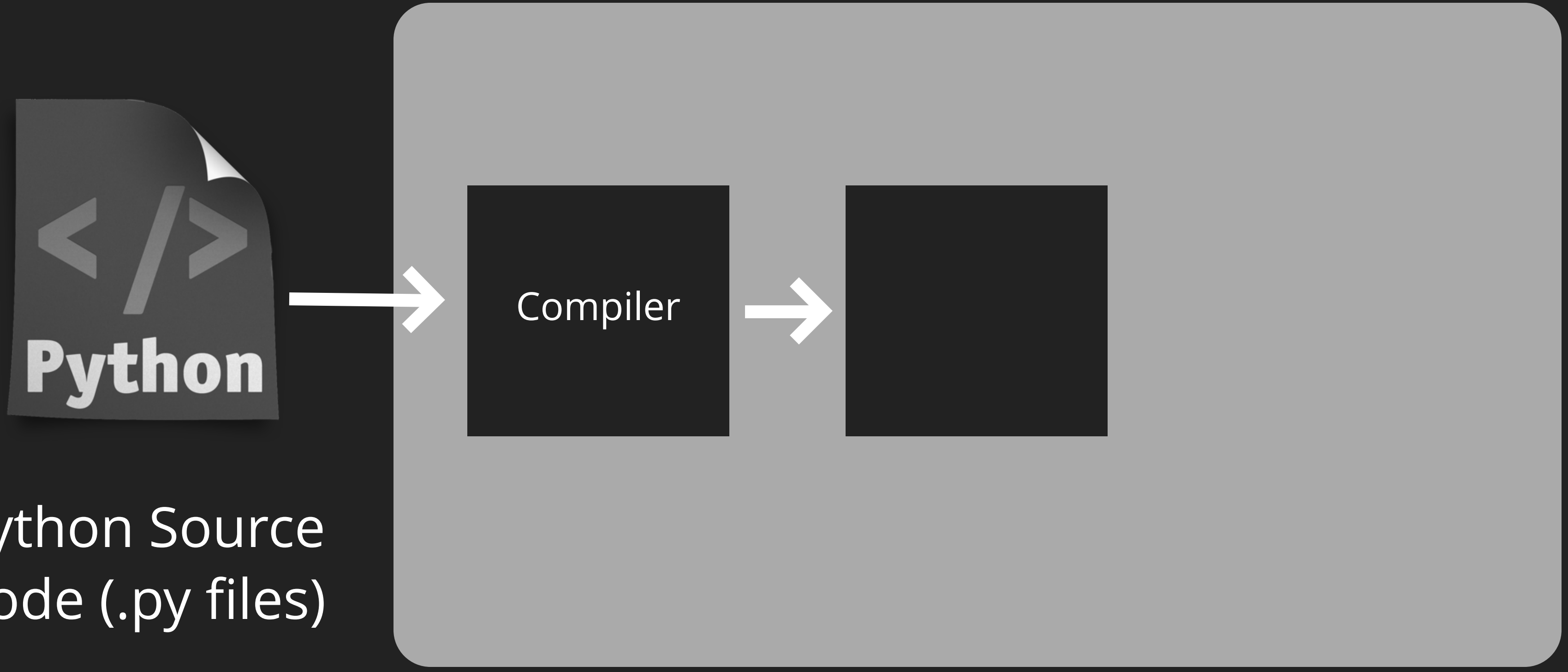


Compiler

Python Source Code (.py files)

Python Interpreter

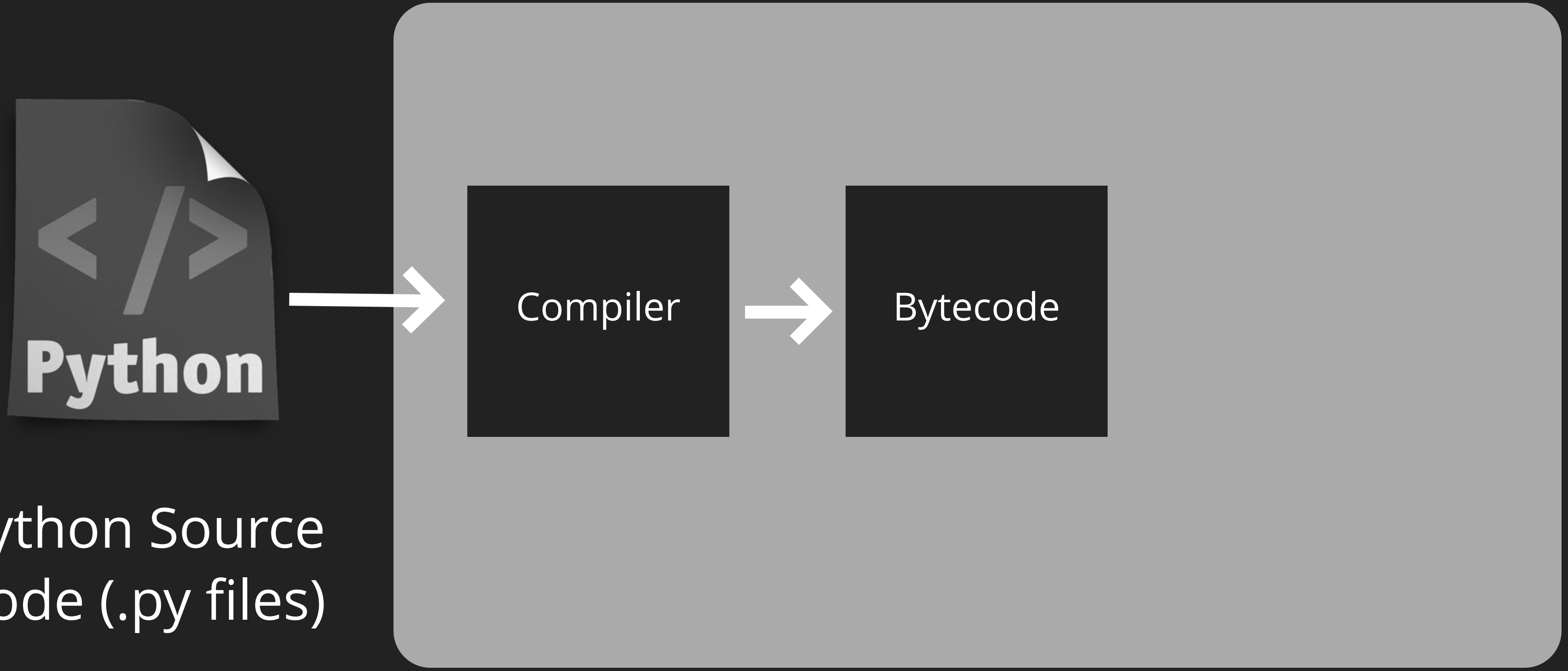
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

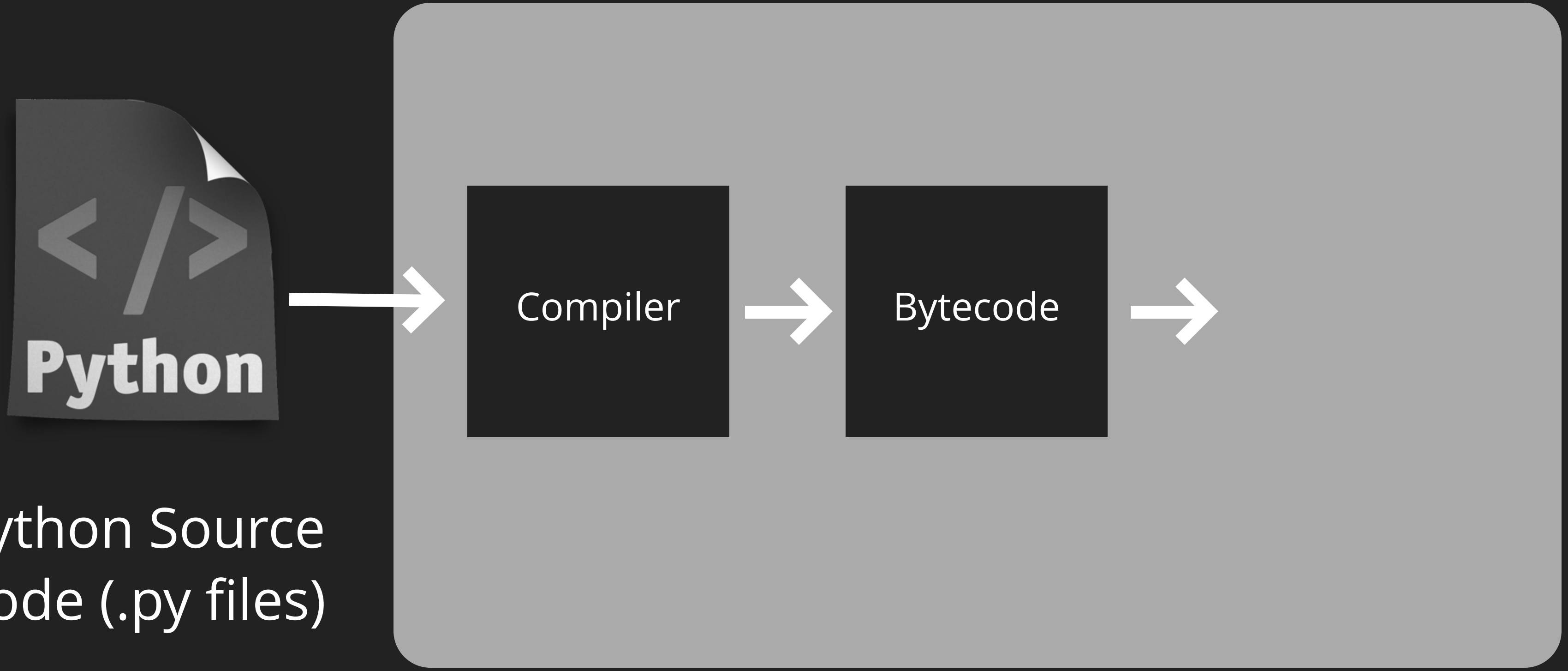
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

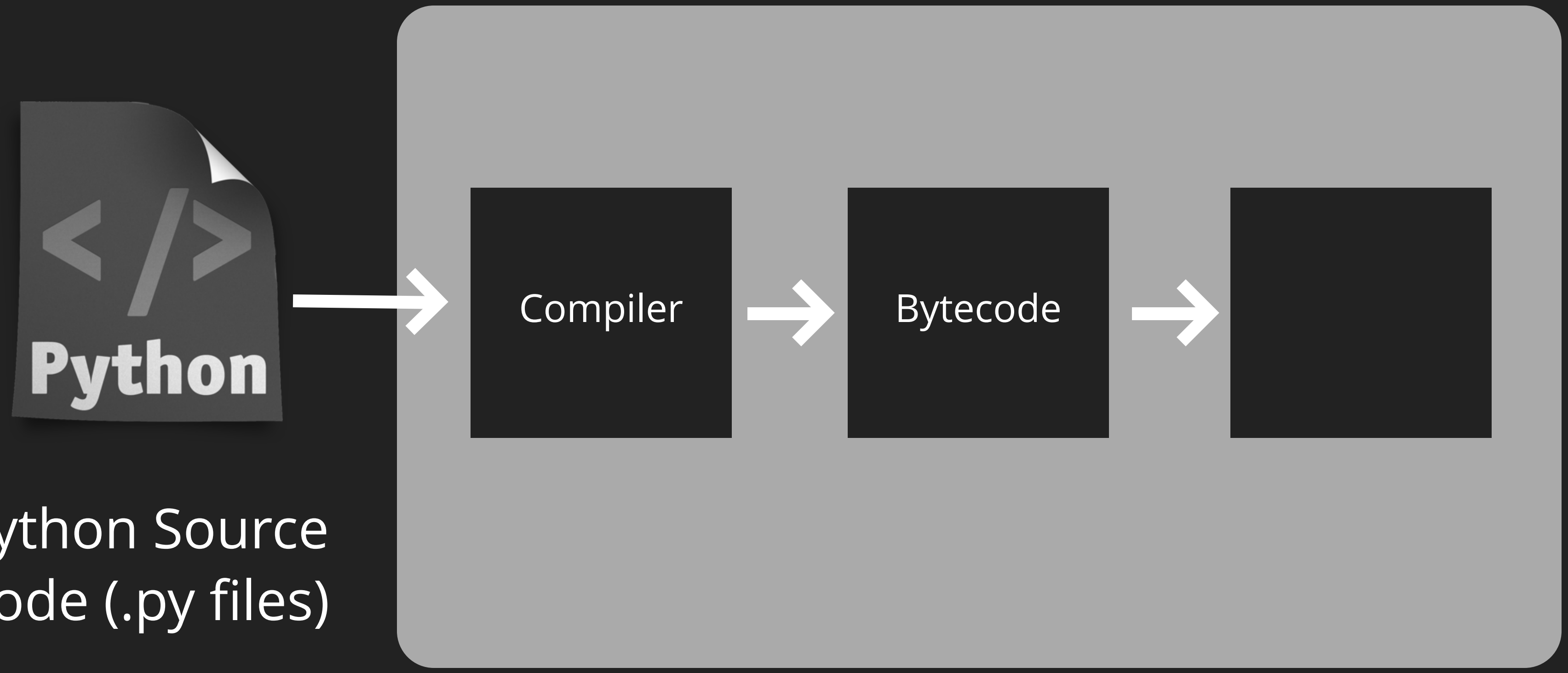
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

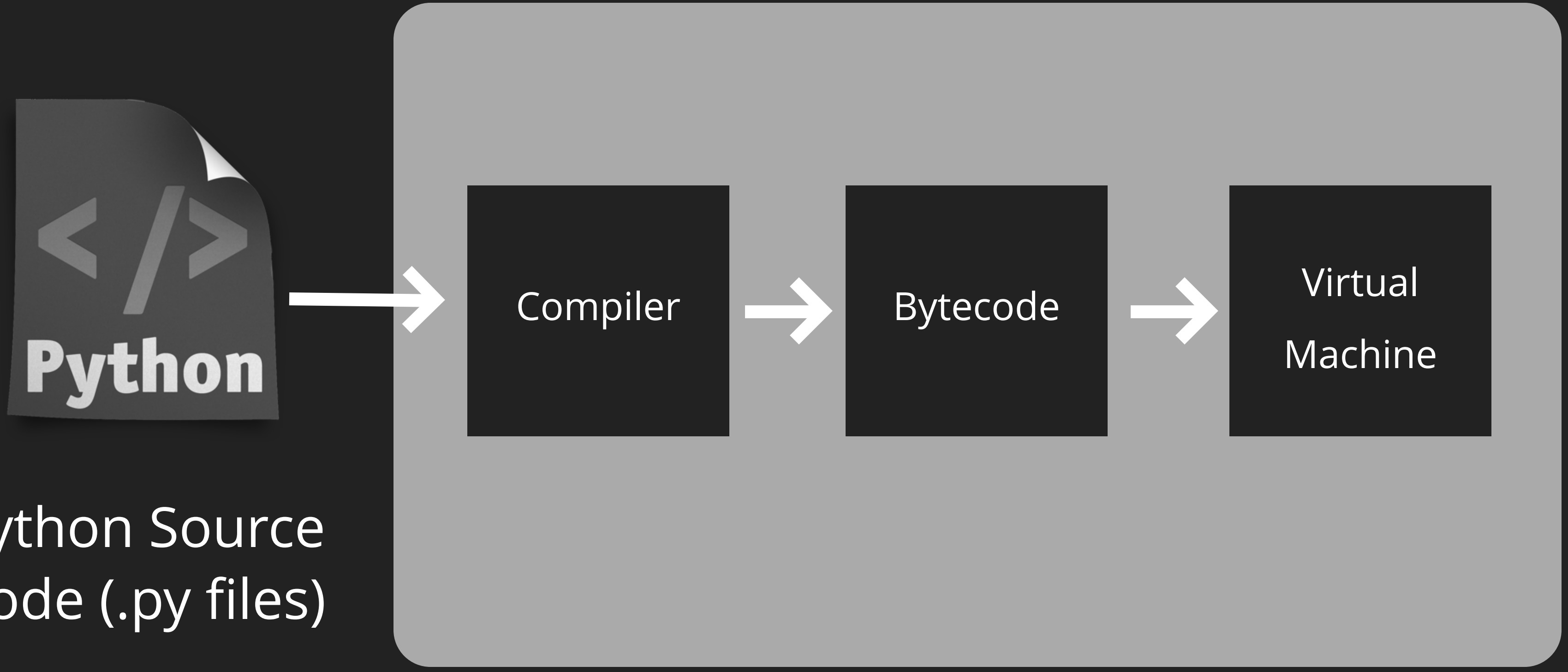
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

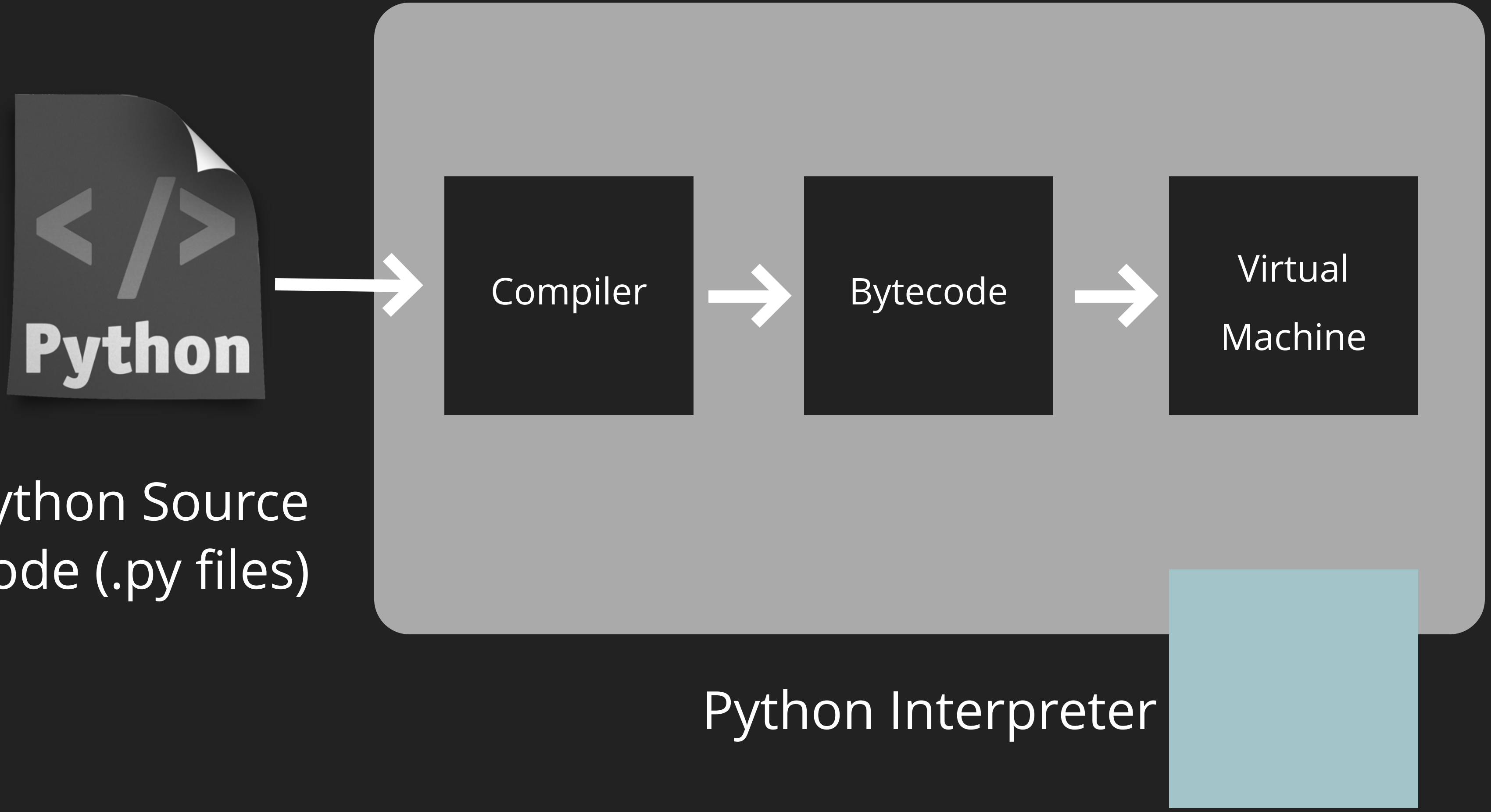
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

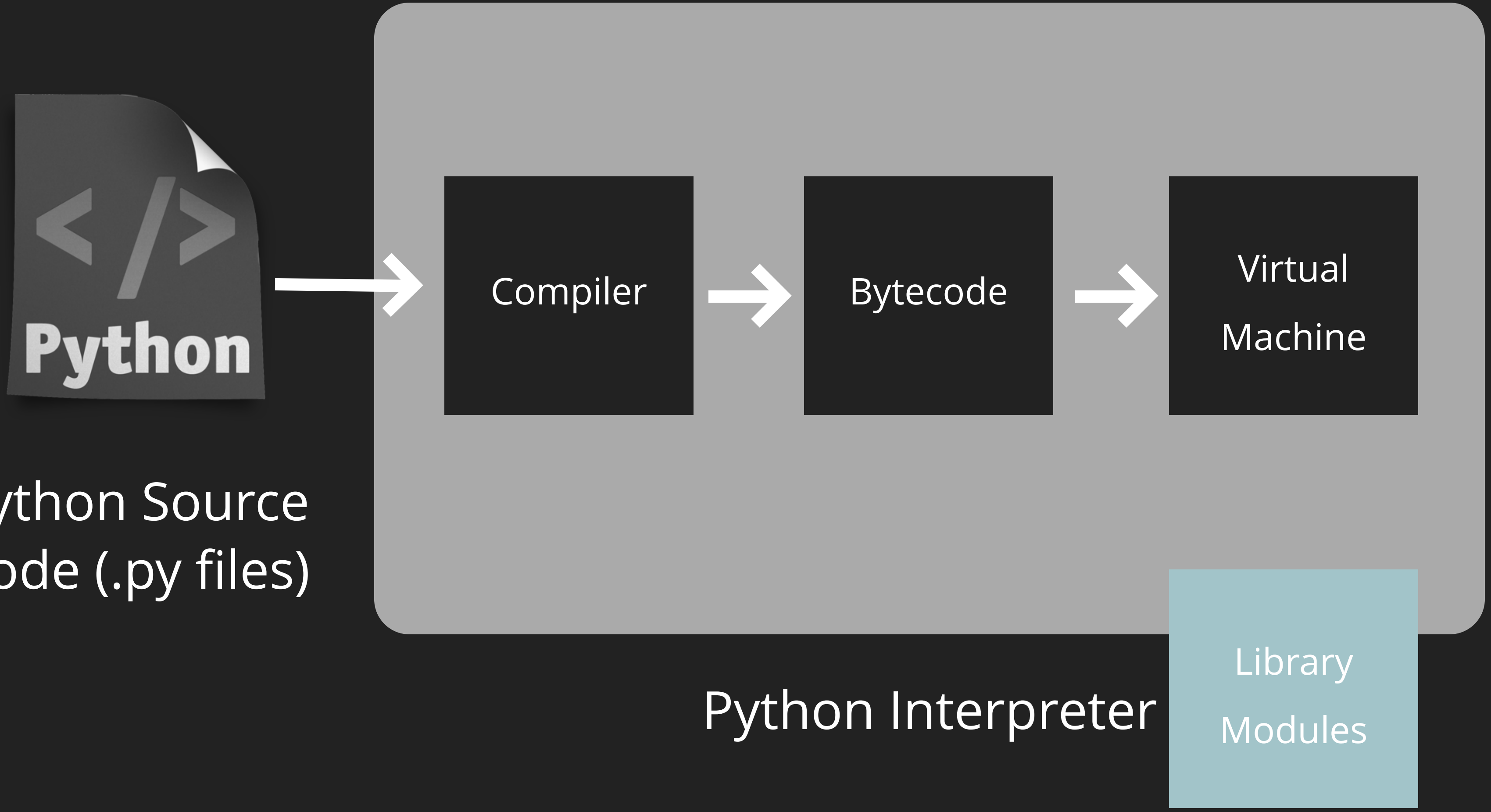
What is Python Interpreter?



Python Source Code (.py files)

Python Interpreter

What is Python Interpreter?



Python Source Code (.py files)

Compiler

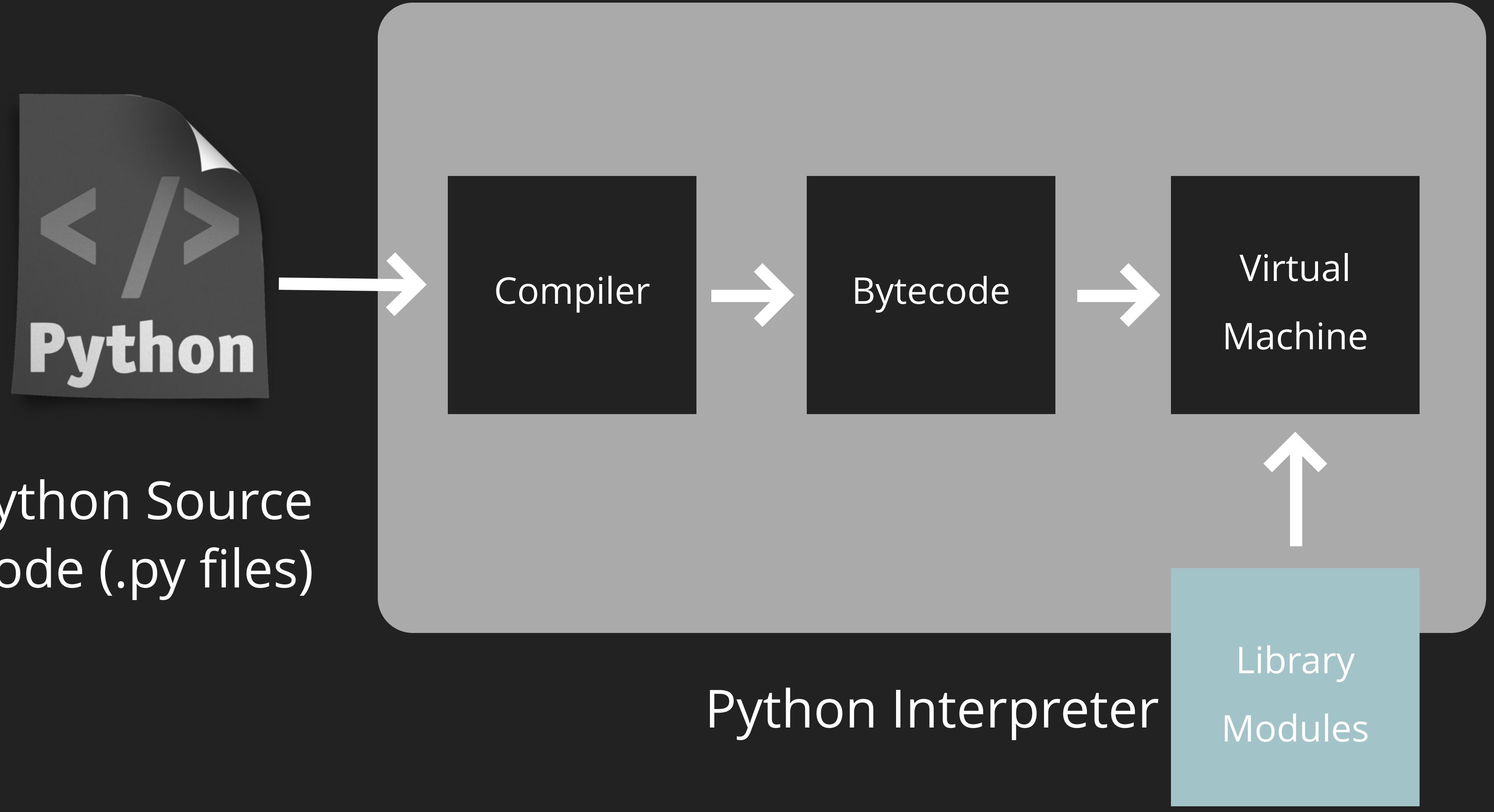
Bytecode

Virtual Machine

Library Modules

Python Interpreter

What is Python Interpreter?



Compiler

Bytecode

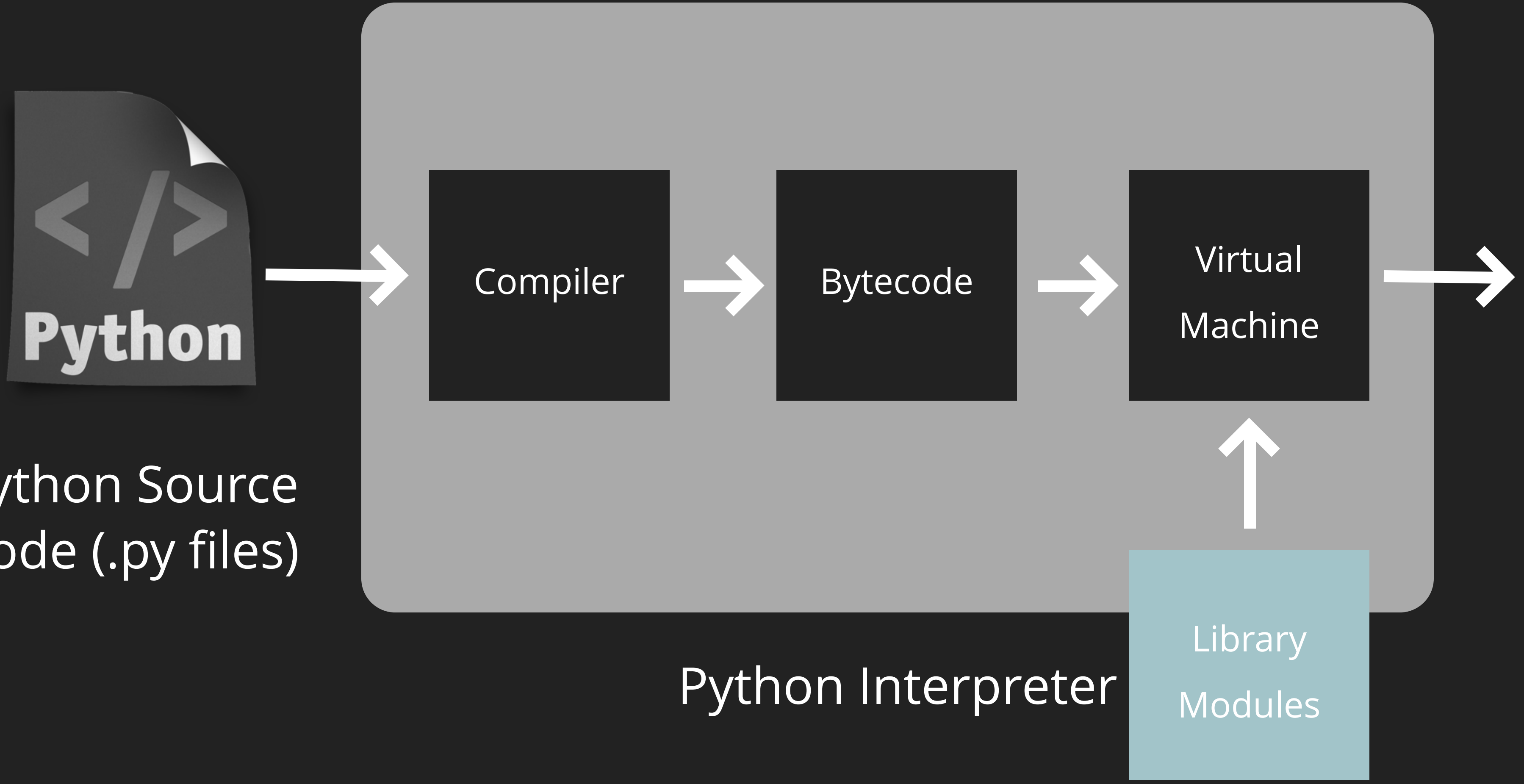
Virtual Machine

Library Modules

Python Source Code (.py files)

Python Interpreter

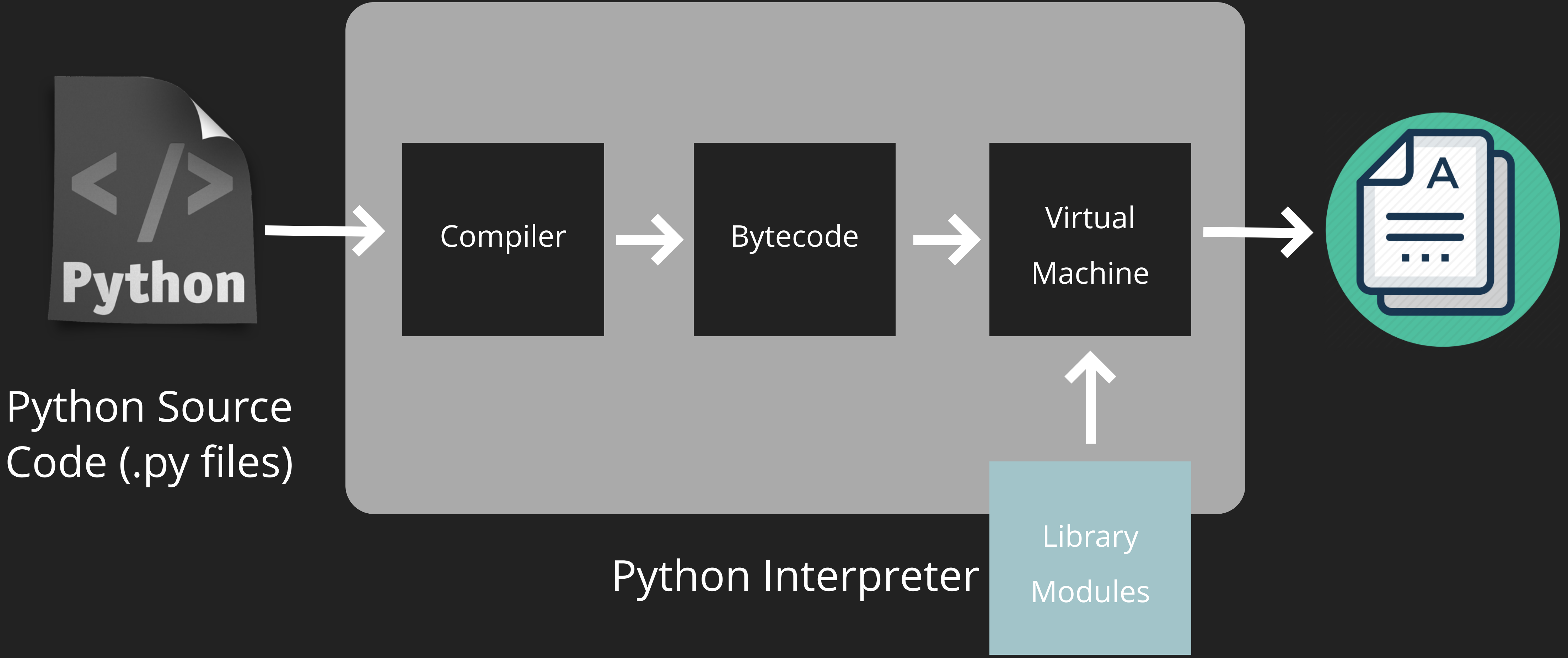
What is Python Interpreter?



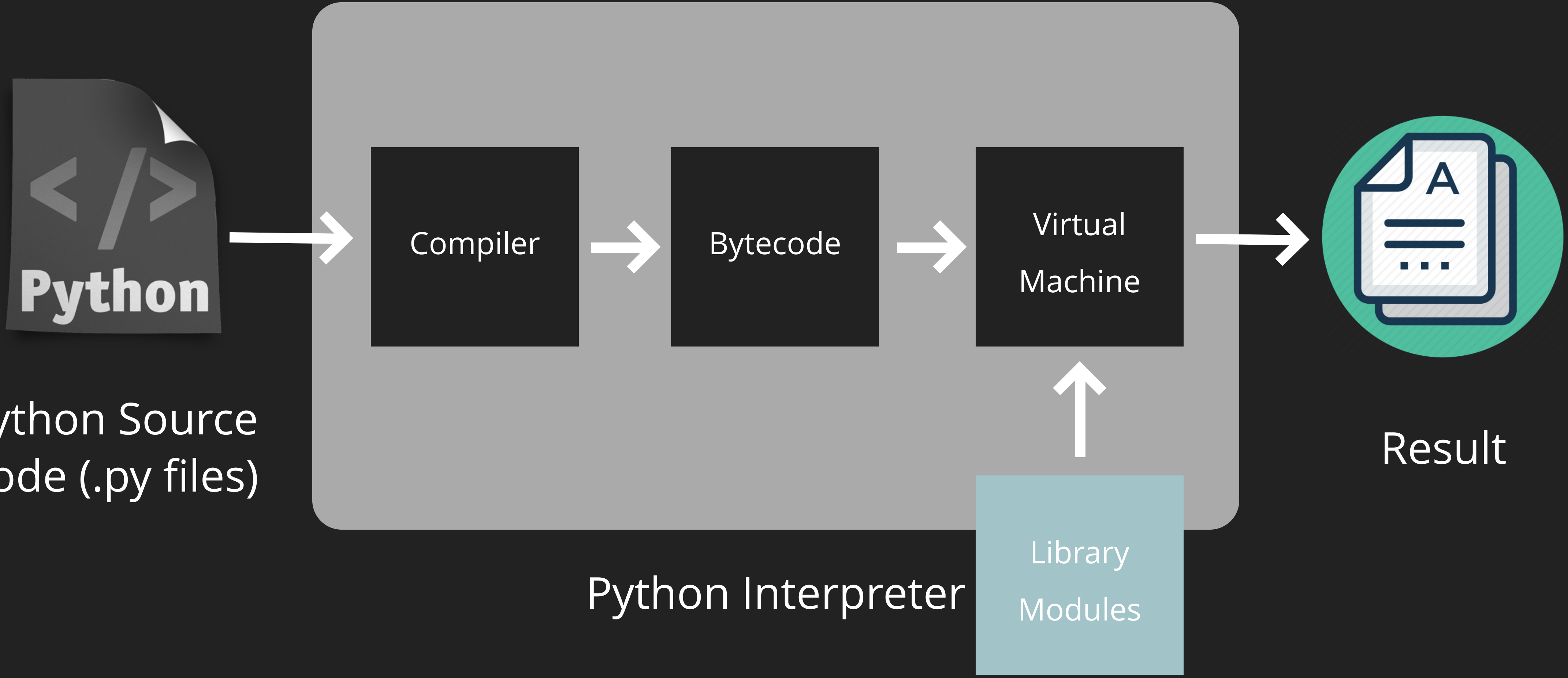
Python Source Code (.py files)

Python Interpreter

What is Python Interpreter?



What is Python Interpreter?



Python Source Code (.py files)

Compiler

Bytecode

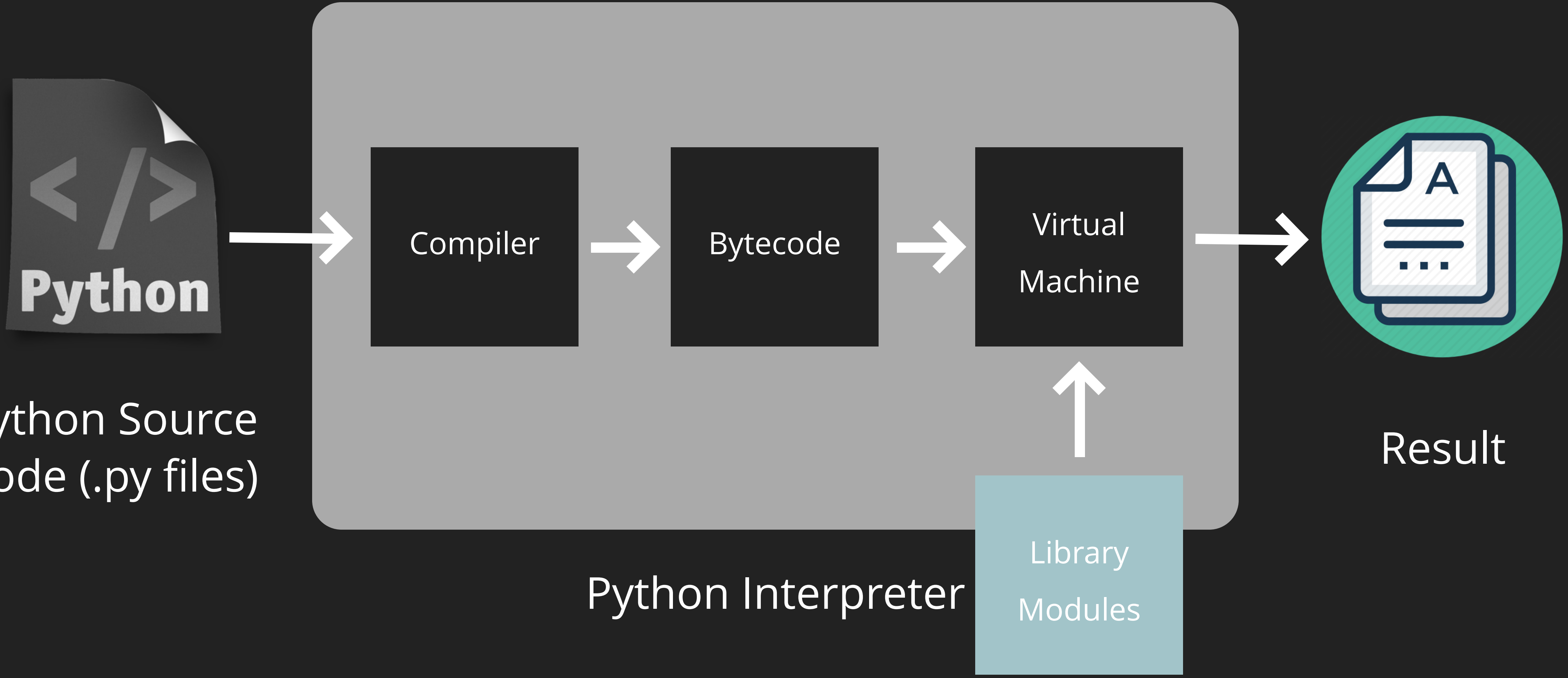
Virtual Machine

Library Modules

Python Interpreter

Result

"Lock" on What?

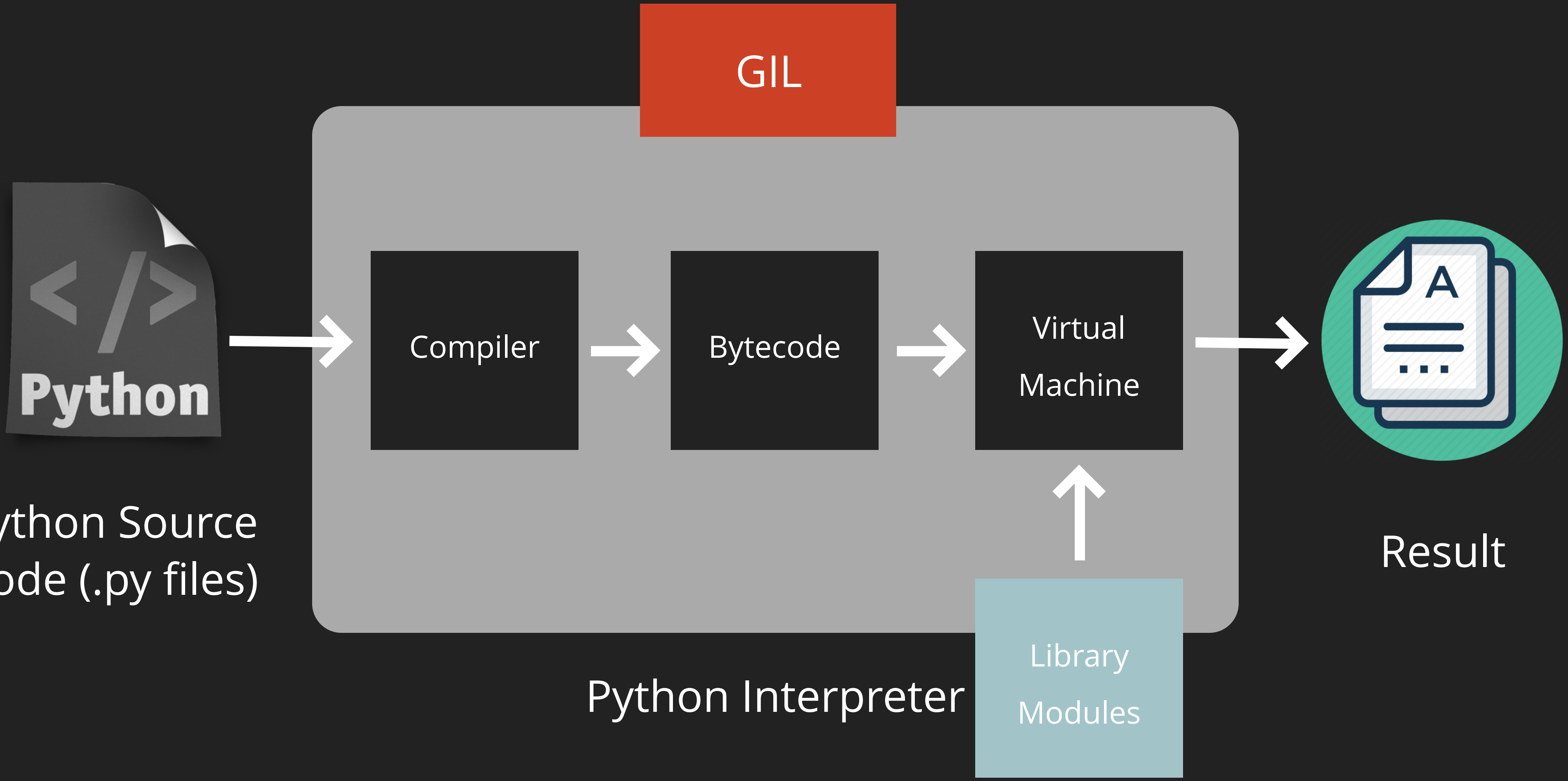


Python Source Code (.py files)

Python Interpreter

Result

"Lock" on What?



GIL

Python

Python Source Code (.py files)

Compiler

Bytecode

Virtual Machine

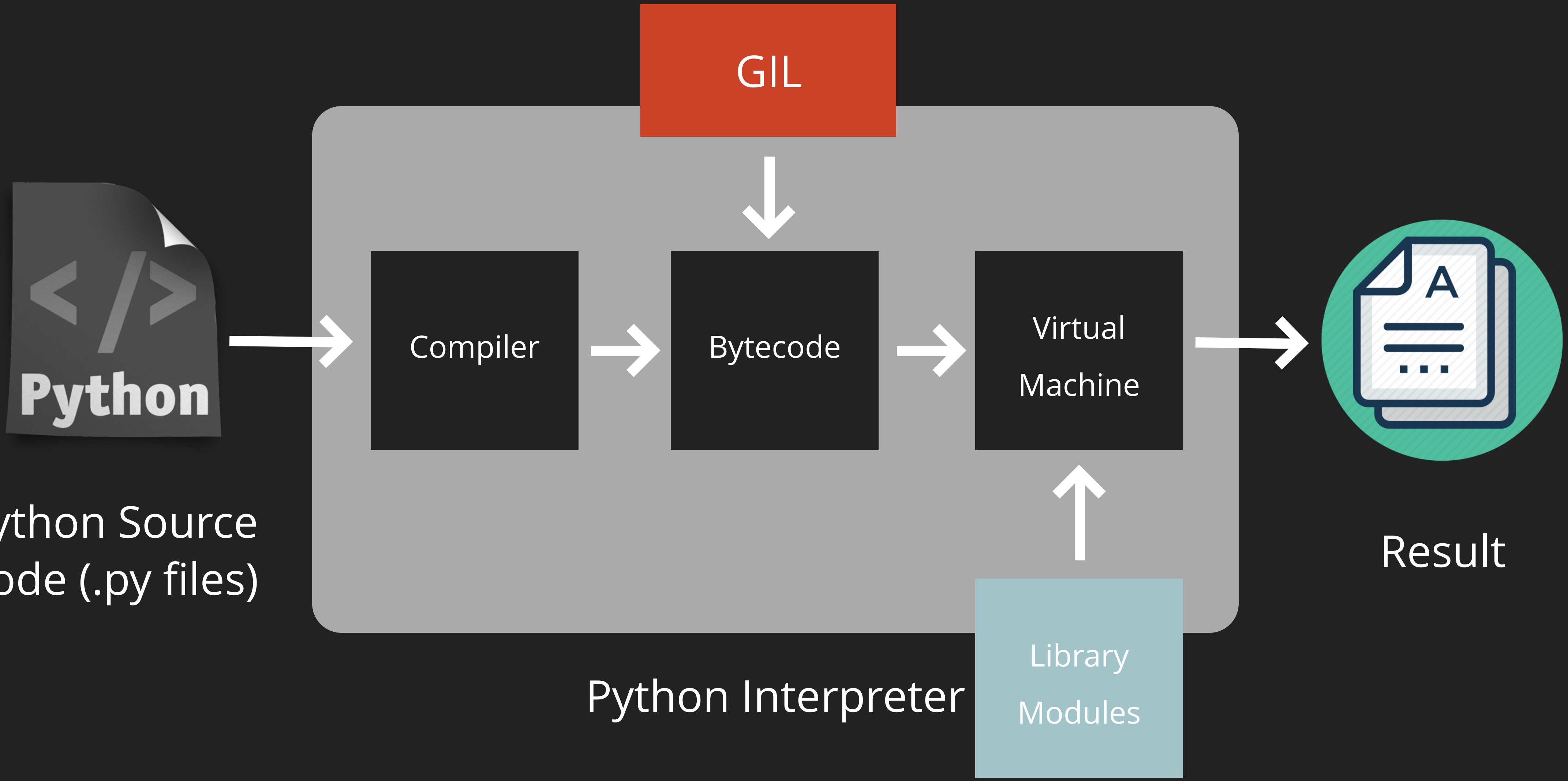
Library Modules

Python Interpreter

A

Result

"Lock" on What?



Python Source Code (.py files)

Python Interpreter

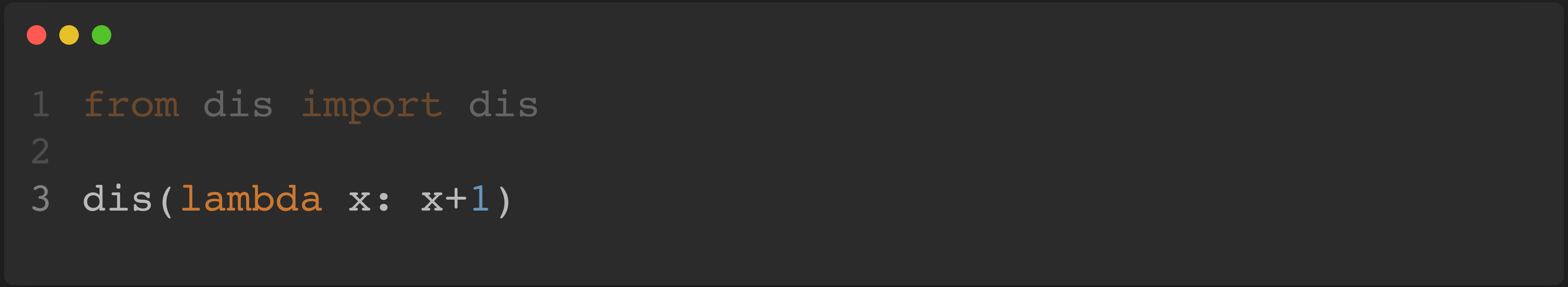
Result

Try Some Bytecode

Is `number += 1` thread safe?

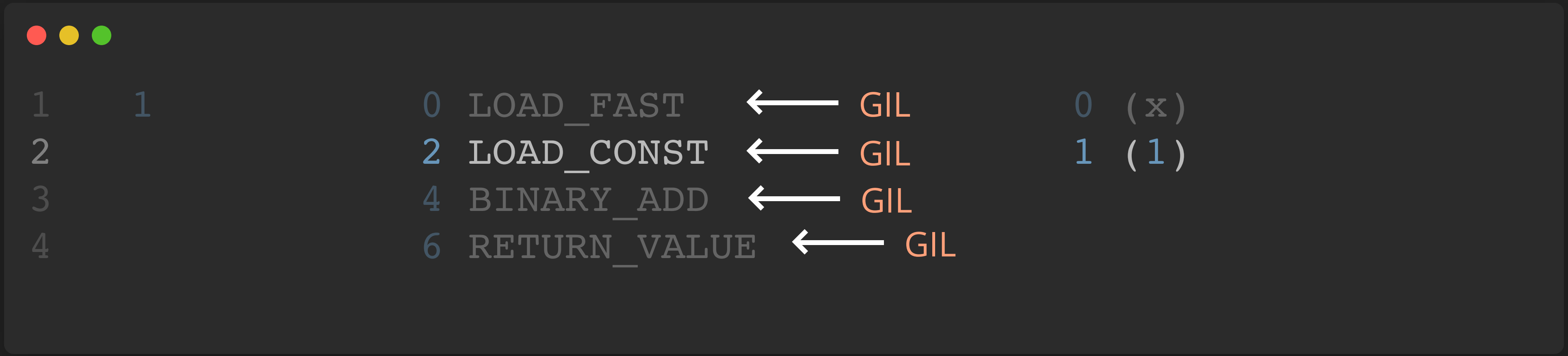
Try Some Bytecode

Is `` `number += 1` `` thread safe?



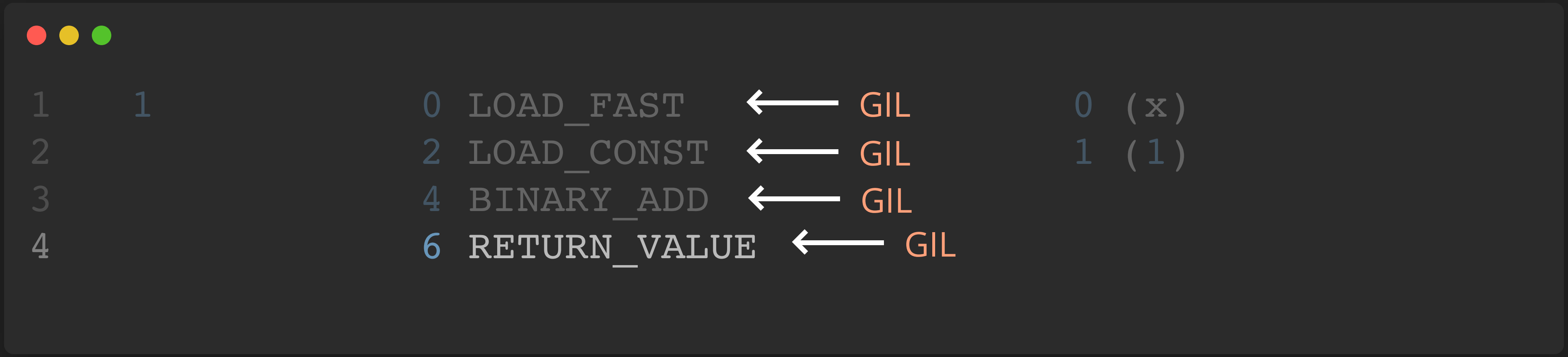
```python
from dis import dis

dis(lambda x: x+1)
```

Try Some Bytecode



```
1           0 LOAD_FAST    ⟵ GIL      0 (x)
            2 LOAD_CONST   ⟵ GIL      1 (1)
            4 BINARY_ADD   ⟵ GIL
            6 RETURN_VALUE  ⟵ GIL
```

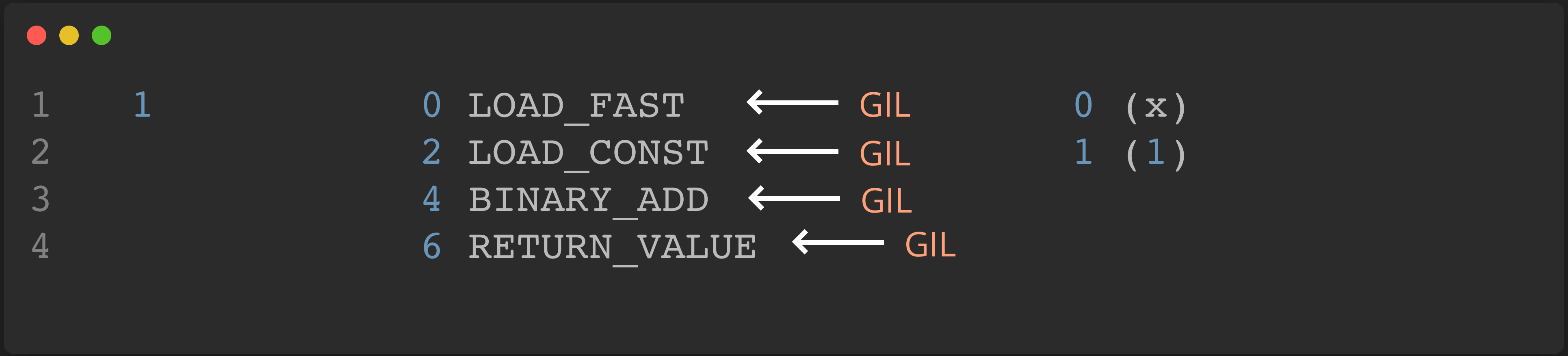
Try Some Bytecode



```
1          0 LOAD_FAST     ⟵ GIL         0 (x)
           2 LOAD_CONST    ⟵ GIL         1 (1)
           4 BINARY_ADD    ⟵ GIL
           6 RETURN_VALUE   ⟵ GIL
```

Not Thread Safe!!!

Try Some Bytecode



```
1           0 LOAD_FAST      ← GIL          0 (x)
            2 LOAD_CONST     ← GIL          1 (1)
            4 BINARY_ADD     ← GIL
            6 RETURN_VALUE     ← GIL
```

Not Thread Safe!!!

Part II

How GIL works?

I/O Bound Module

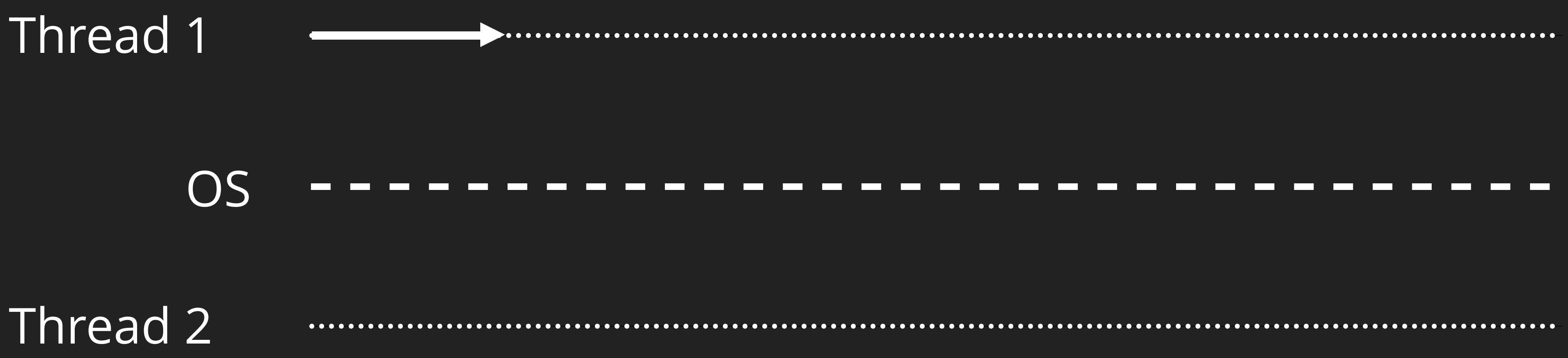
I/O Bound Module

Thread 1

OS

Thread 2

I/O Bound Module

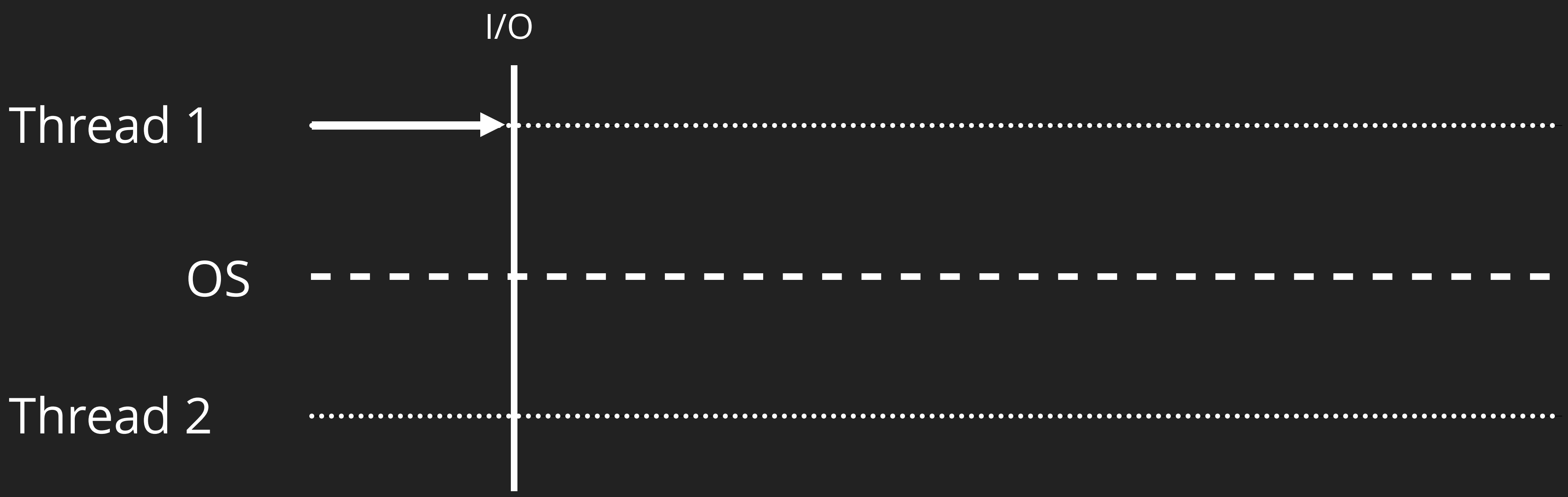


Thread 1

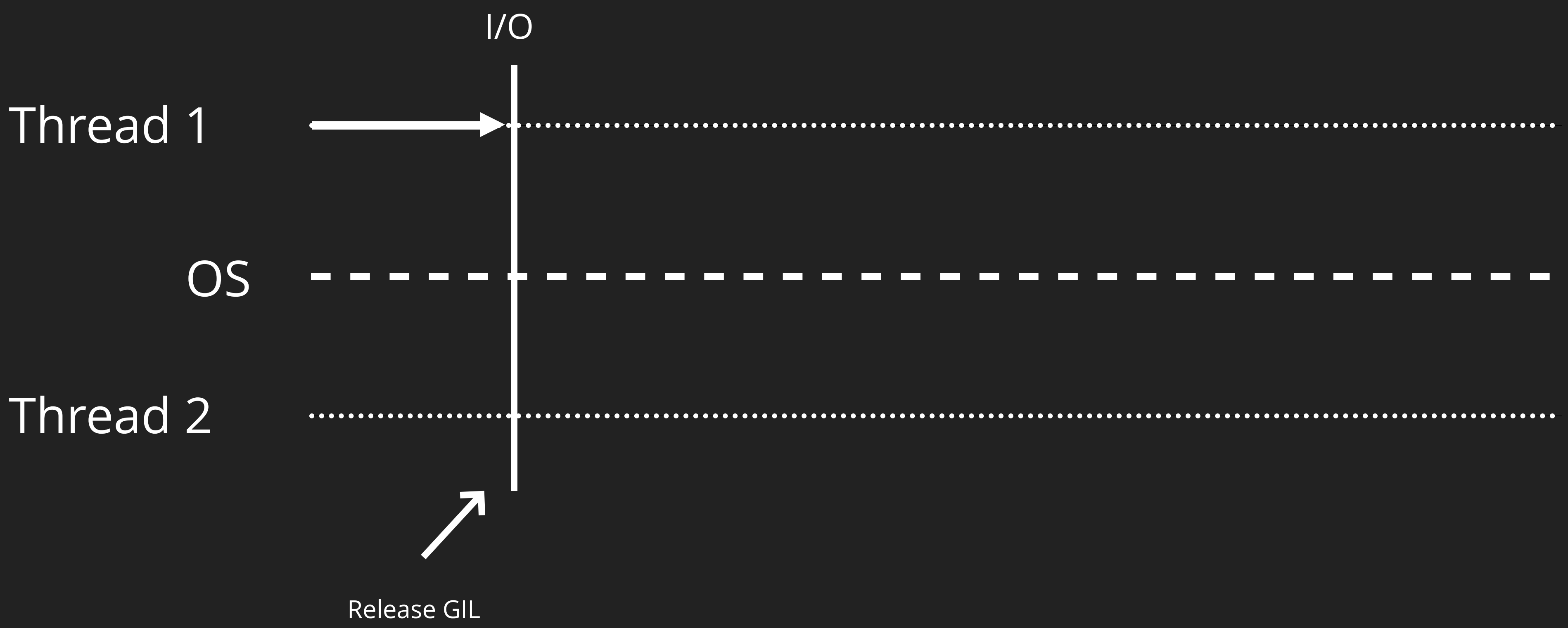
OS

Thread 2

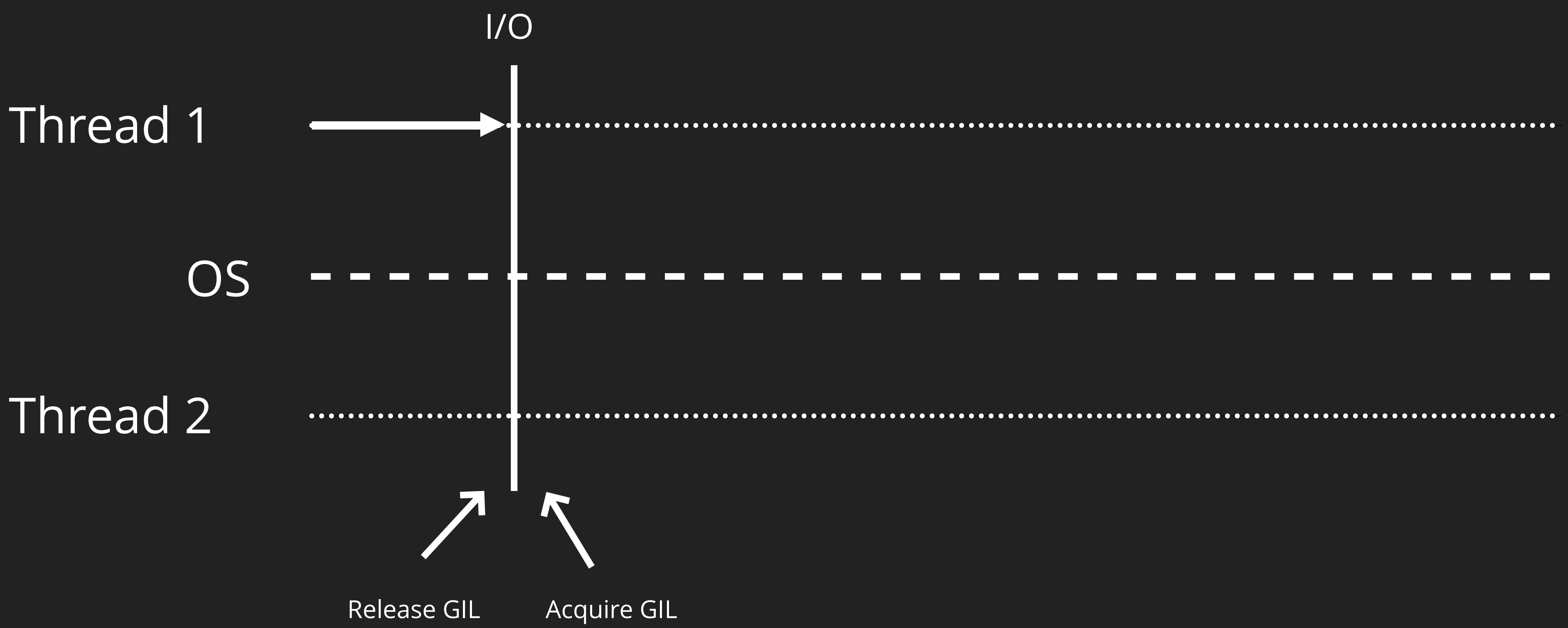
I/O Bound Module



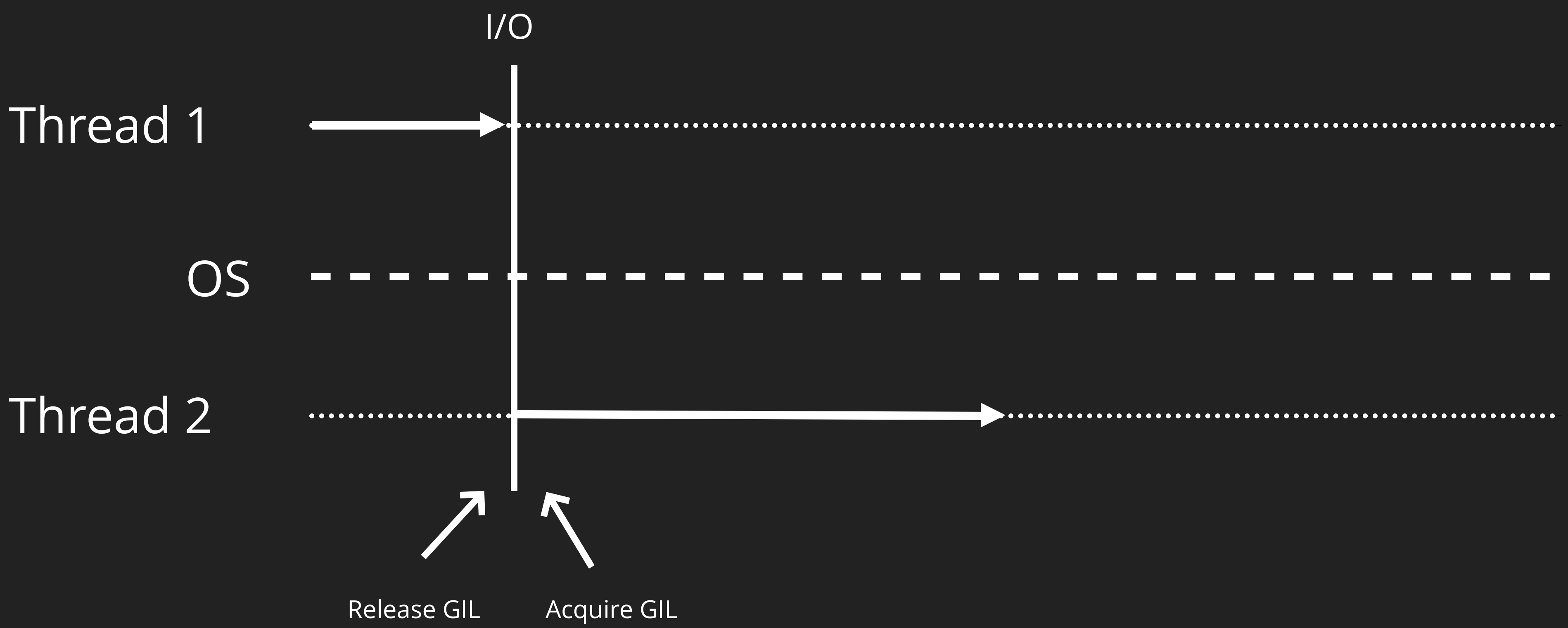
I/O Bound Module



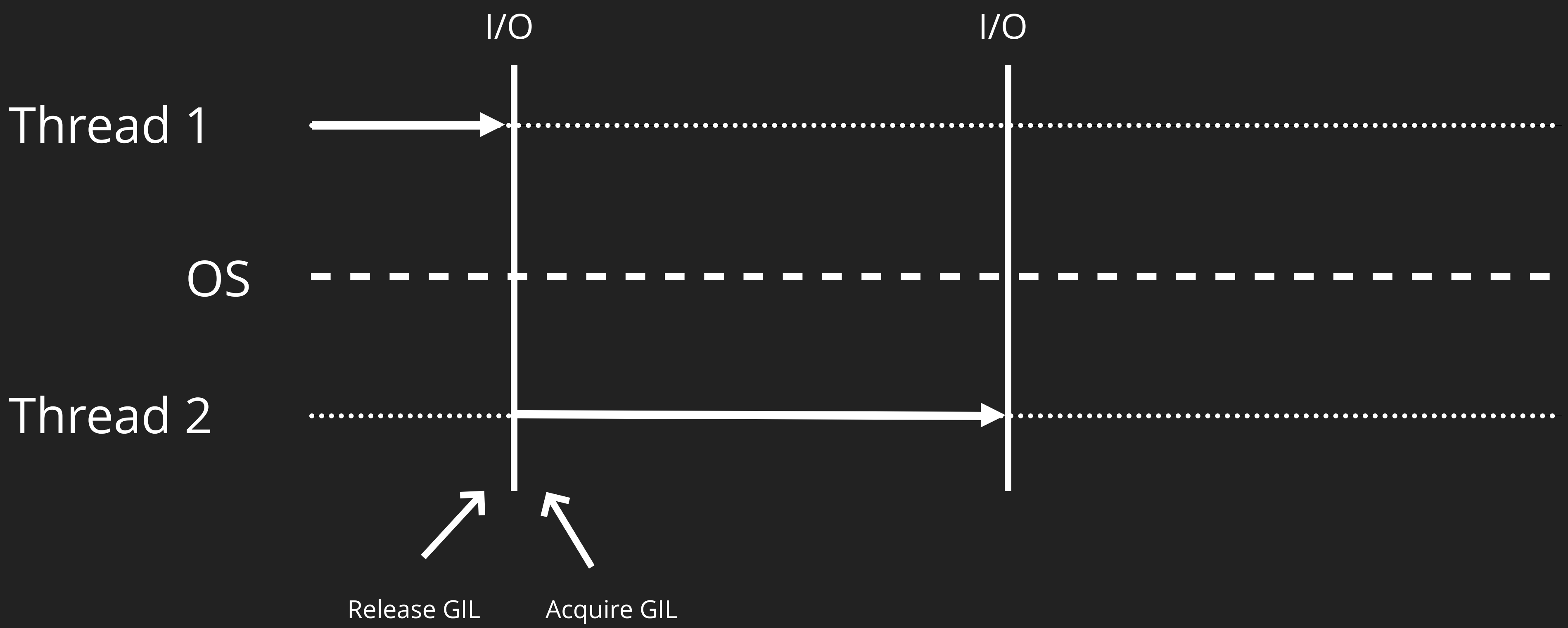
I/O Bound Module



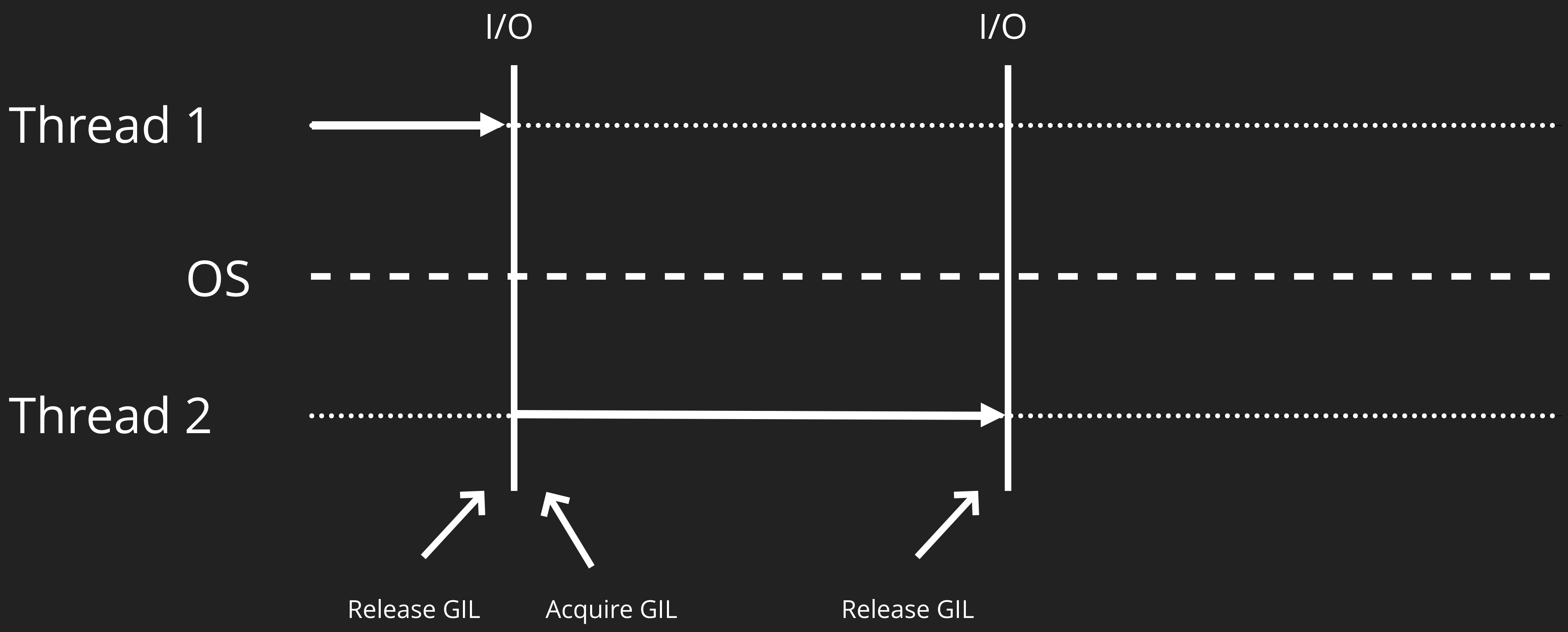
I/O Bound Module



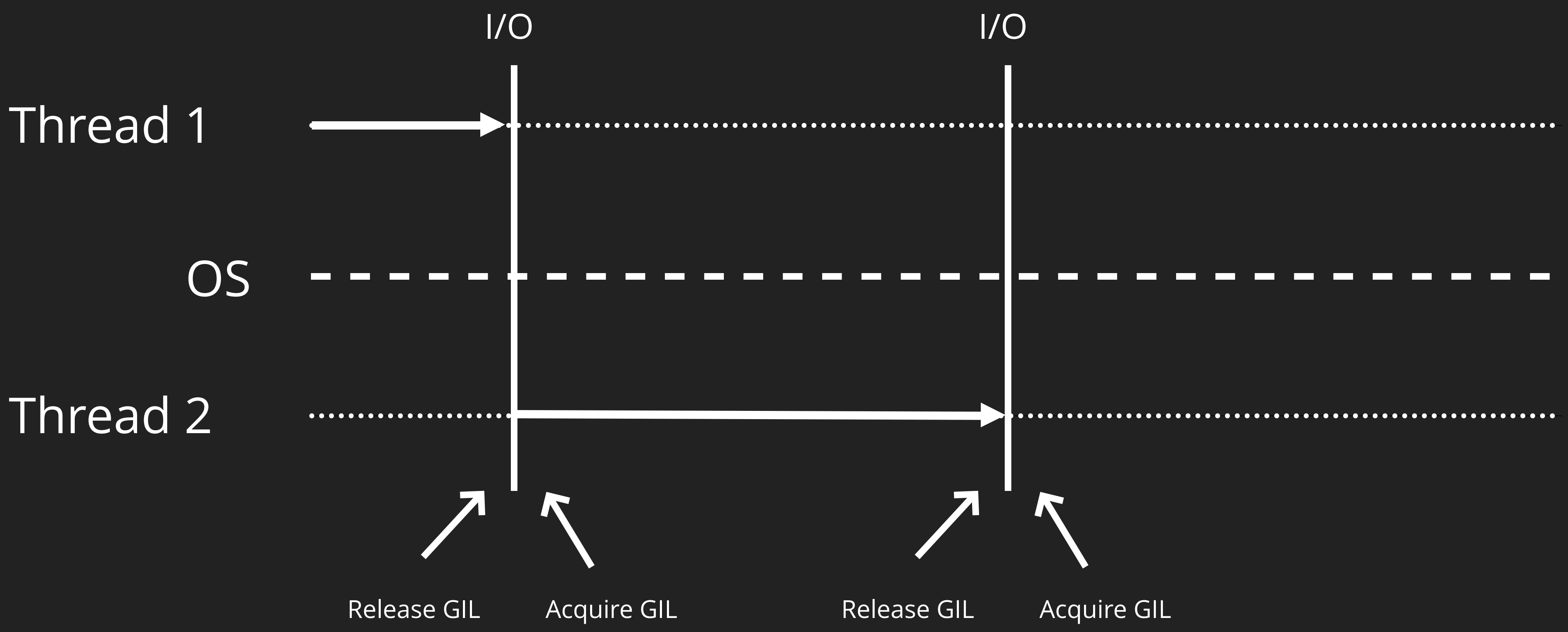
I/O Bound Module



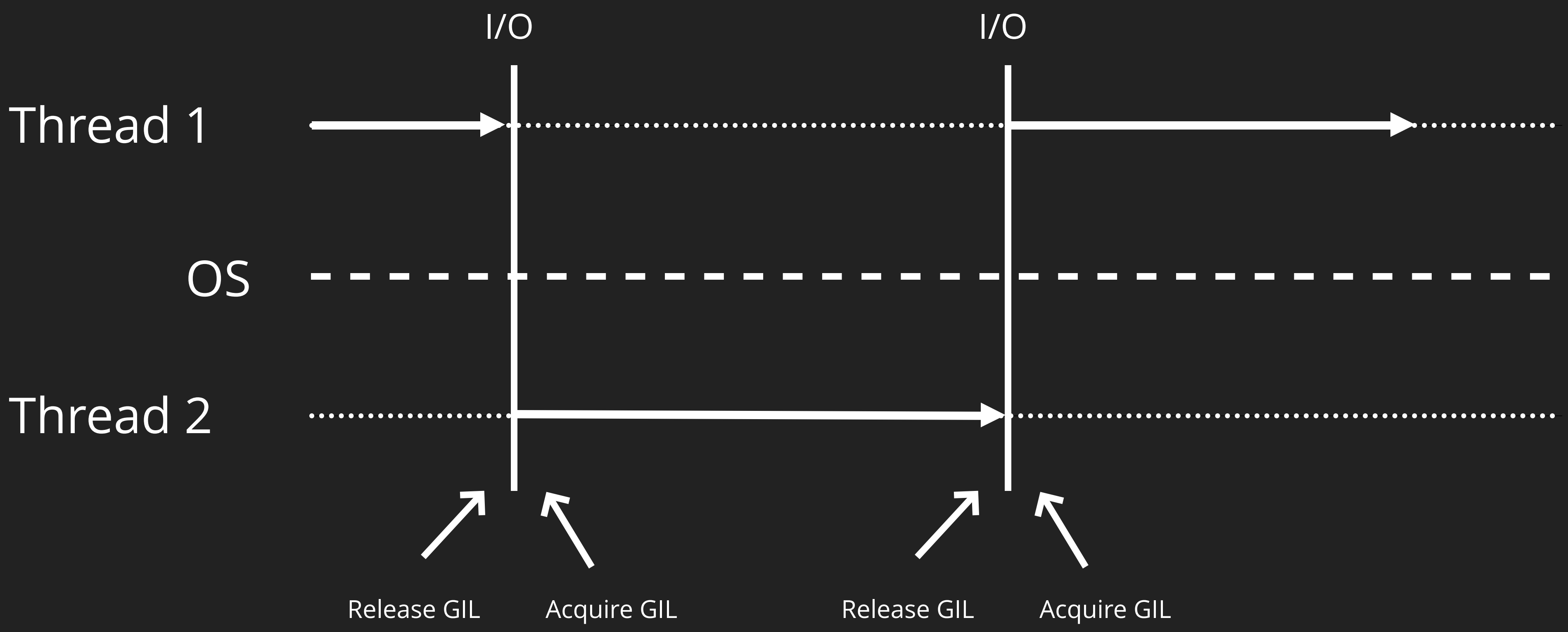
I/O Bound Module



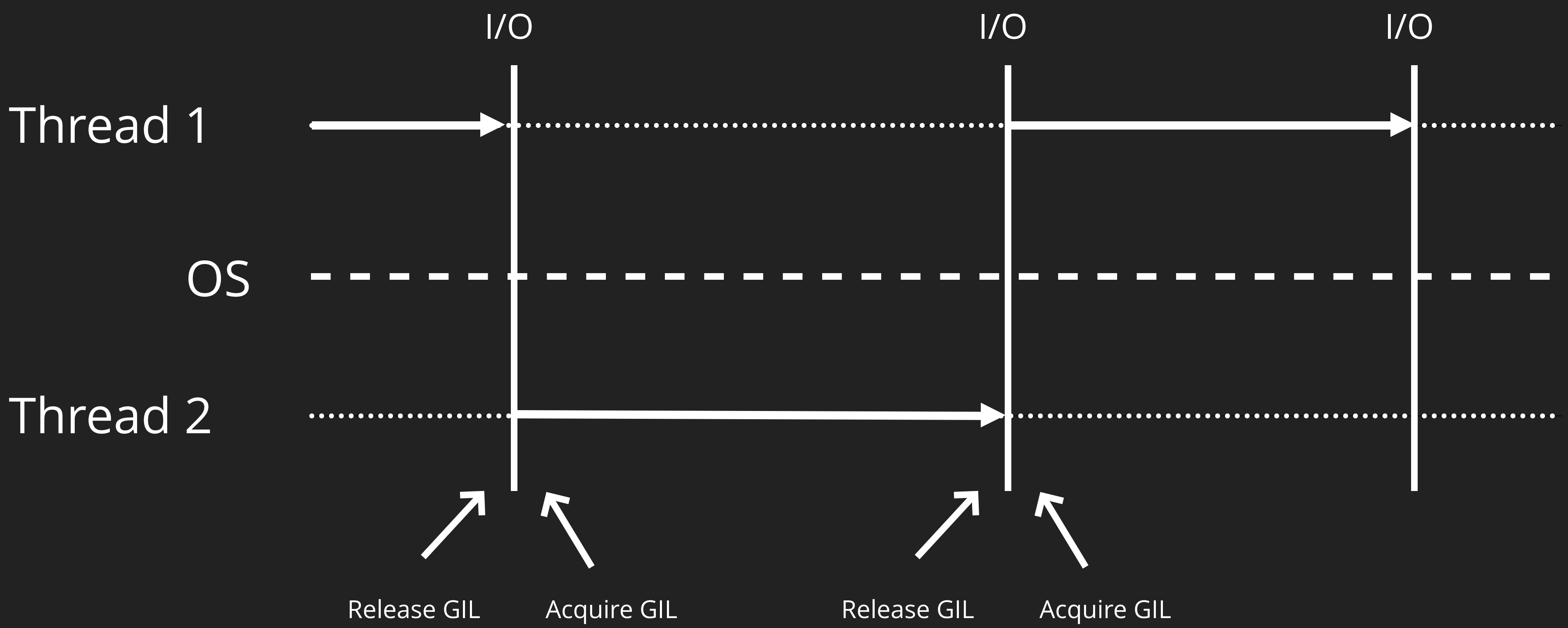
I/O Bound Module



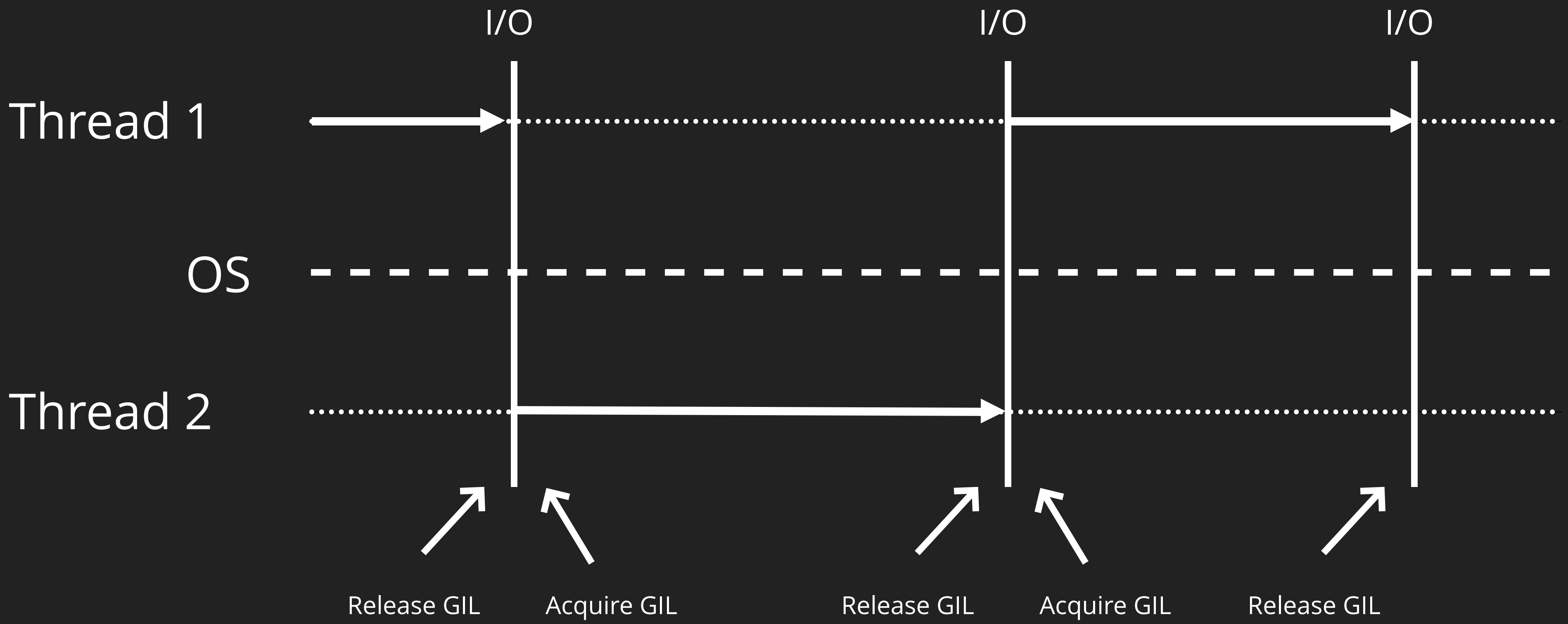
I/O Bound Module



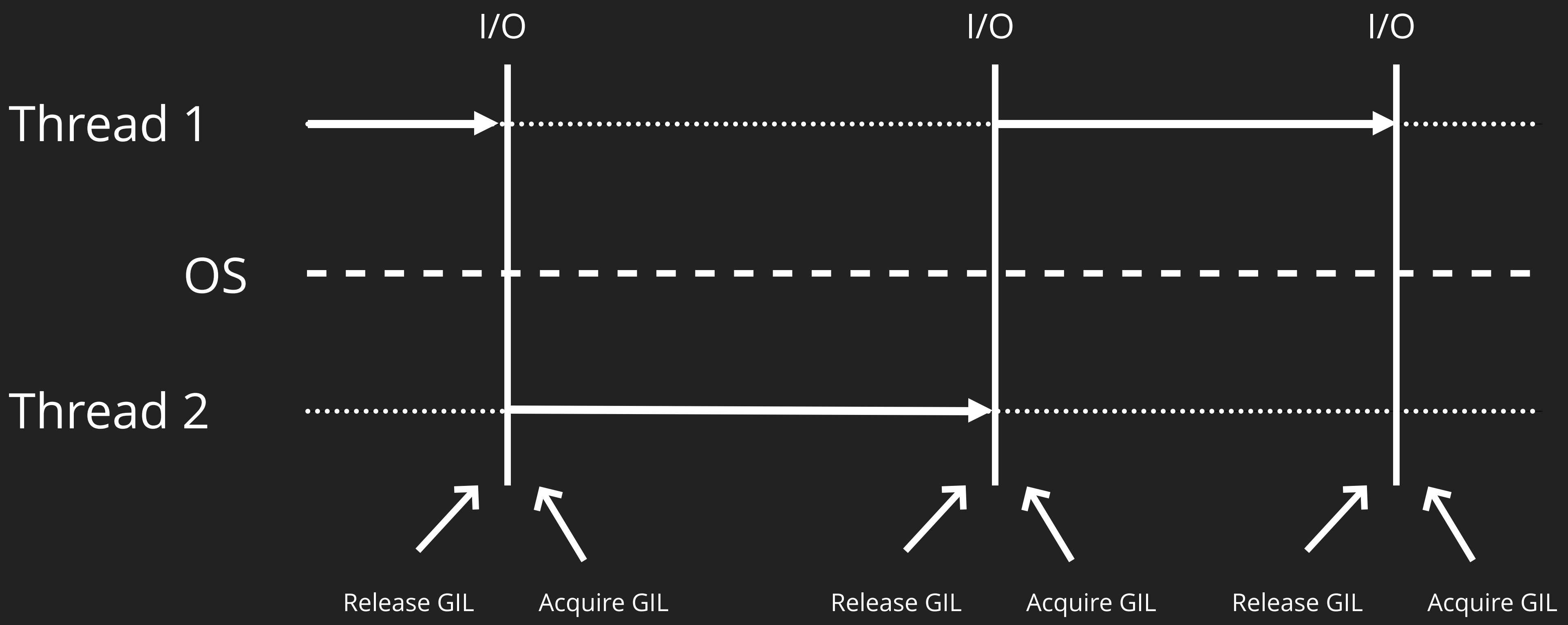
I/O Bound Module



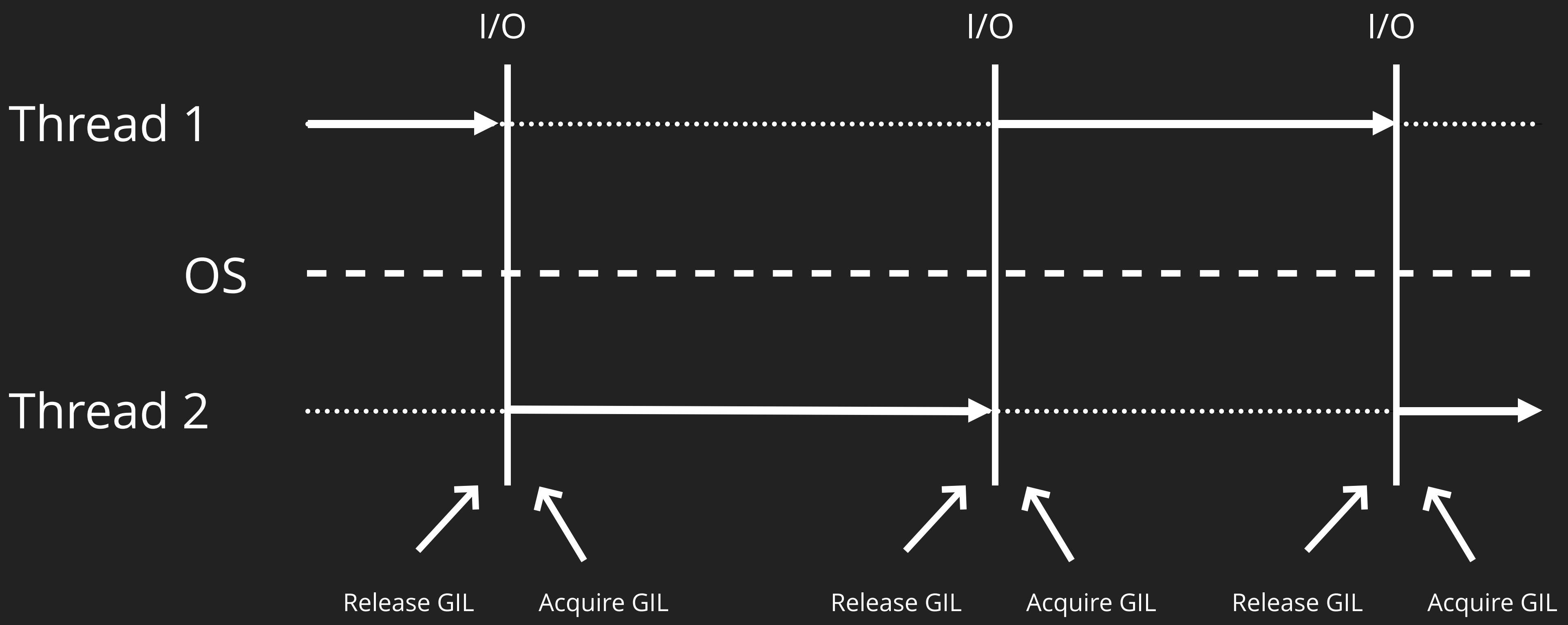
I/O Bound Module



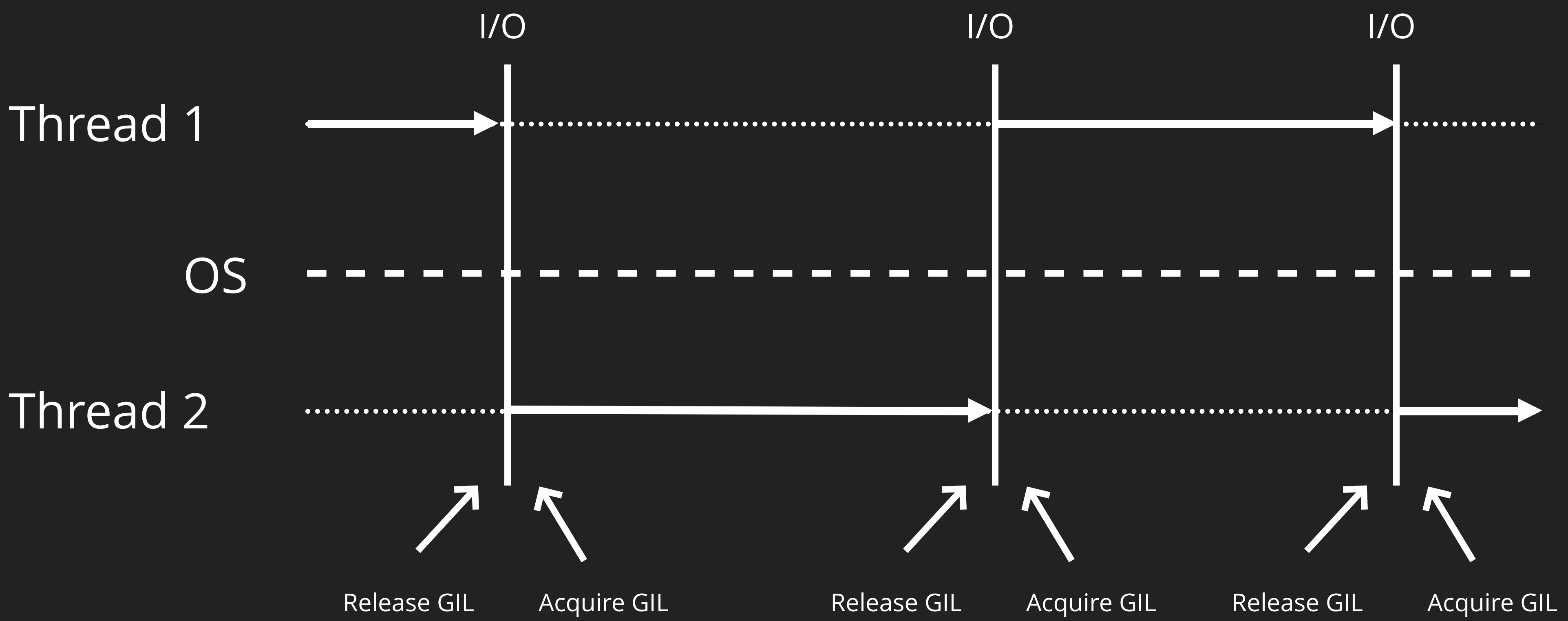
I/O Bound Module



I/O Bound Module

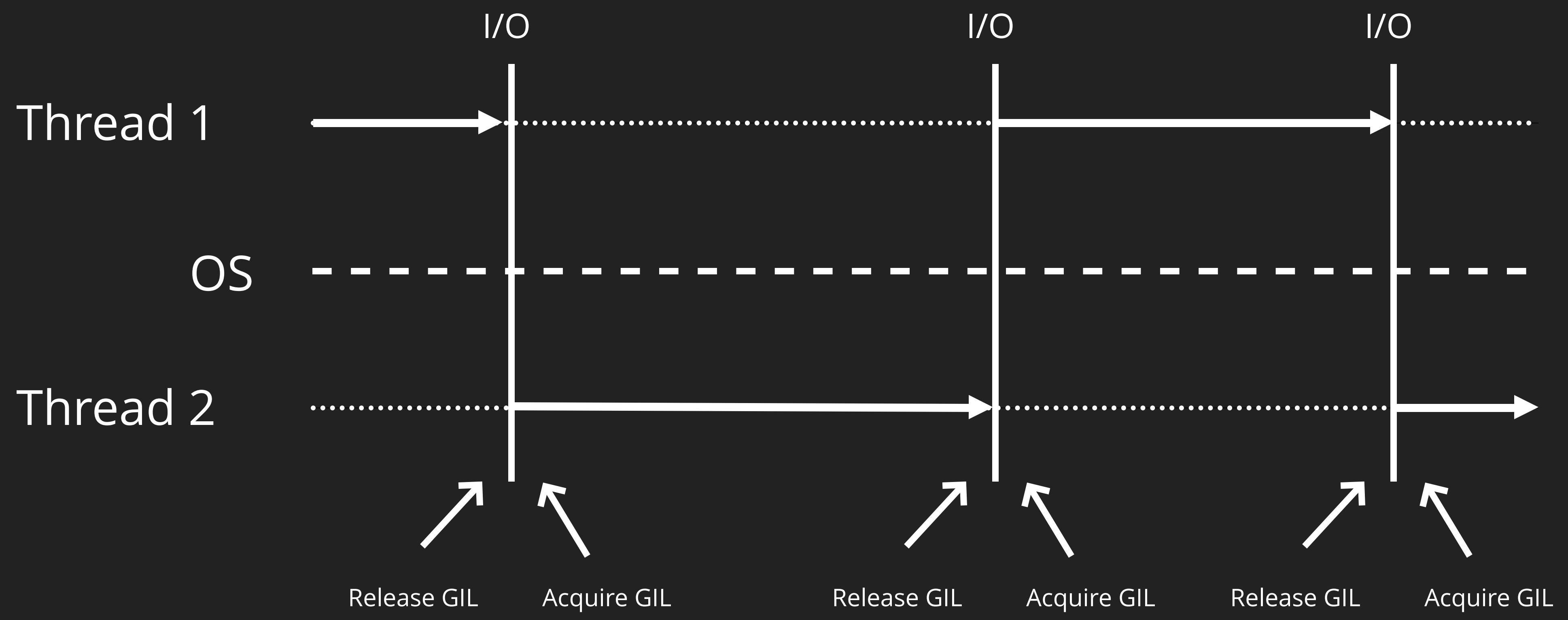


I/O Bound Module



- When a thread is running, it holds the GIL

I/O Bound Module



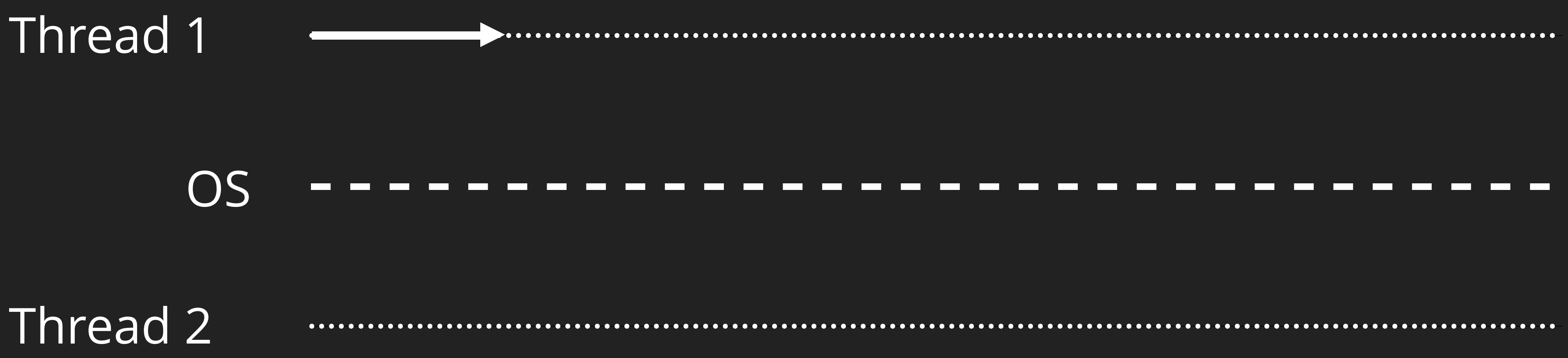
- When a thread is running, it holds the GIL
- GiL released on I/O (read, write, send, recv, etc.)

CPU Bound Module

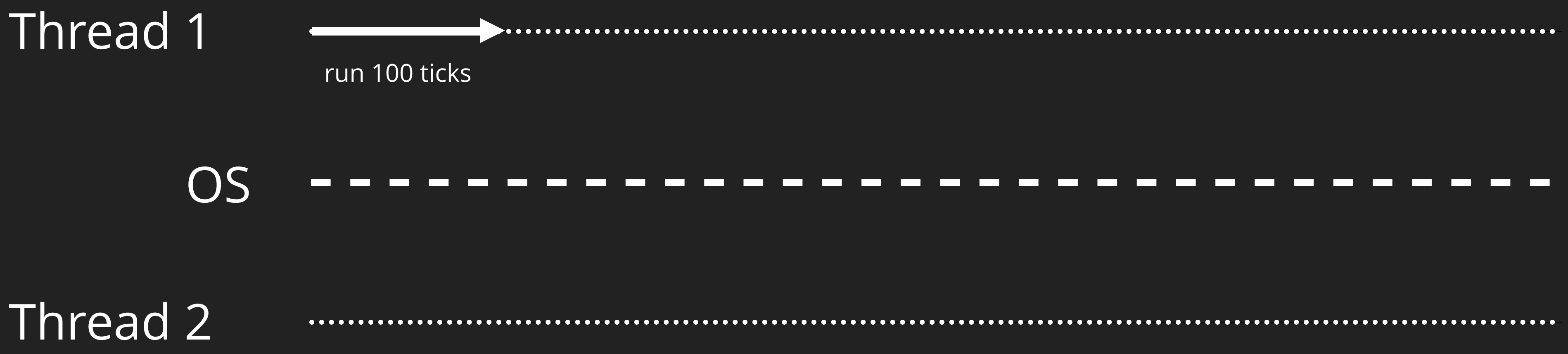
CPU Bound Module

Thread 1
OS
Thread 2

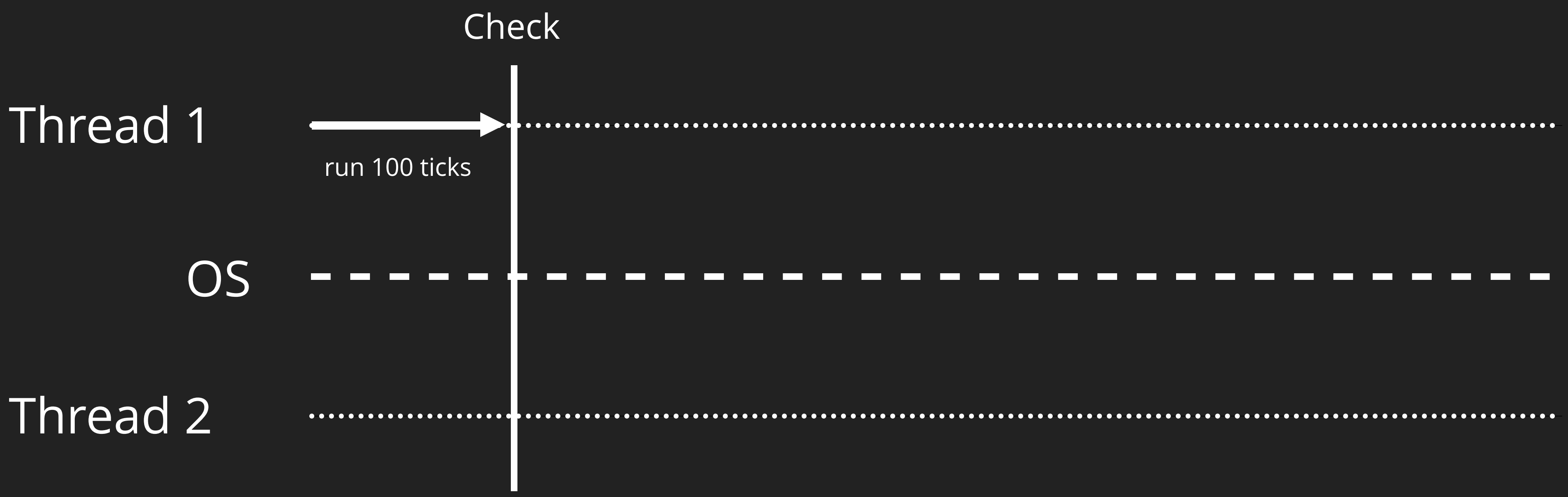
CPU Bound Module



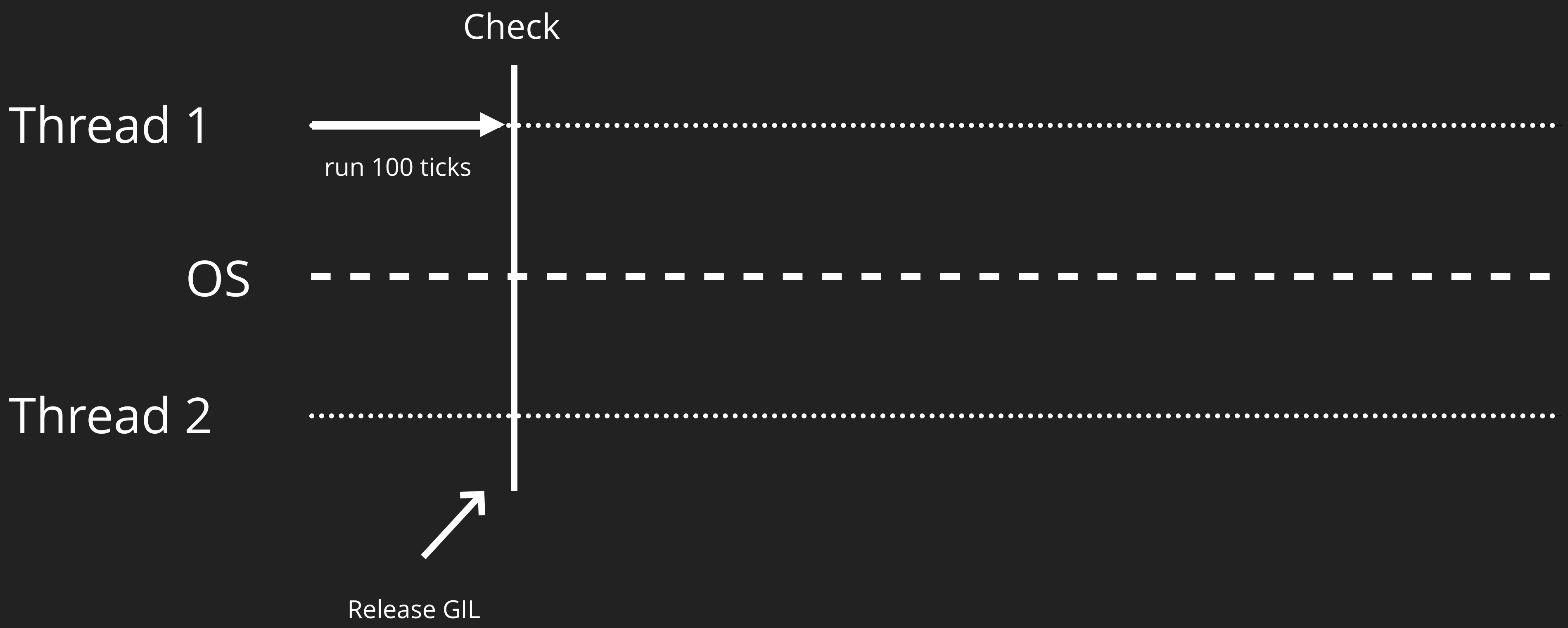
CPU Bound Module



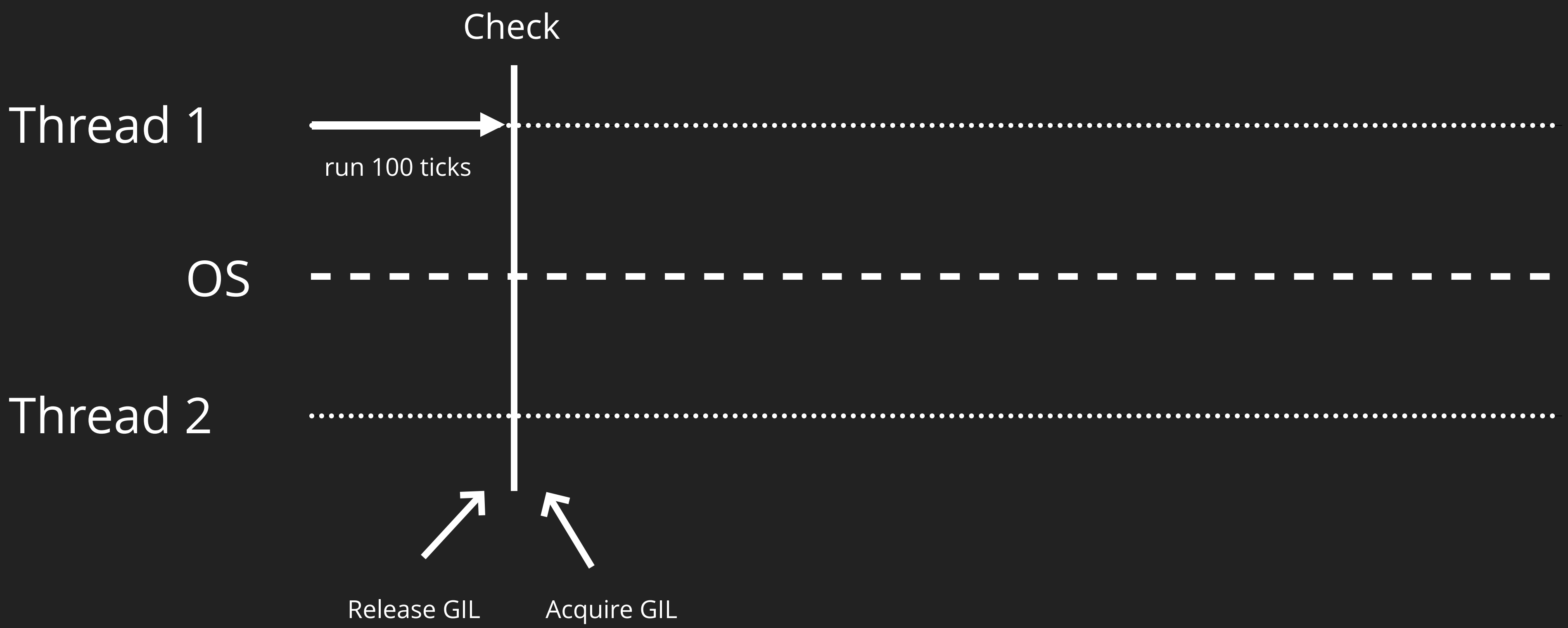
CPU Bound Module



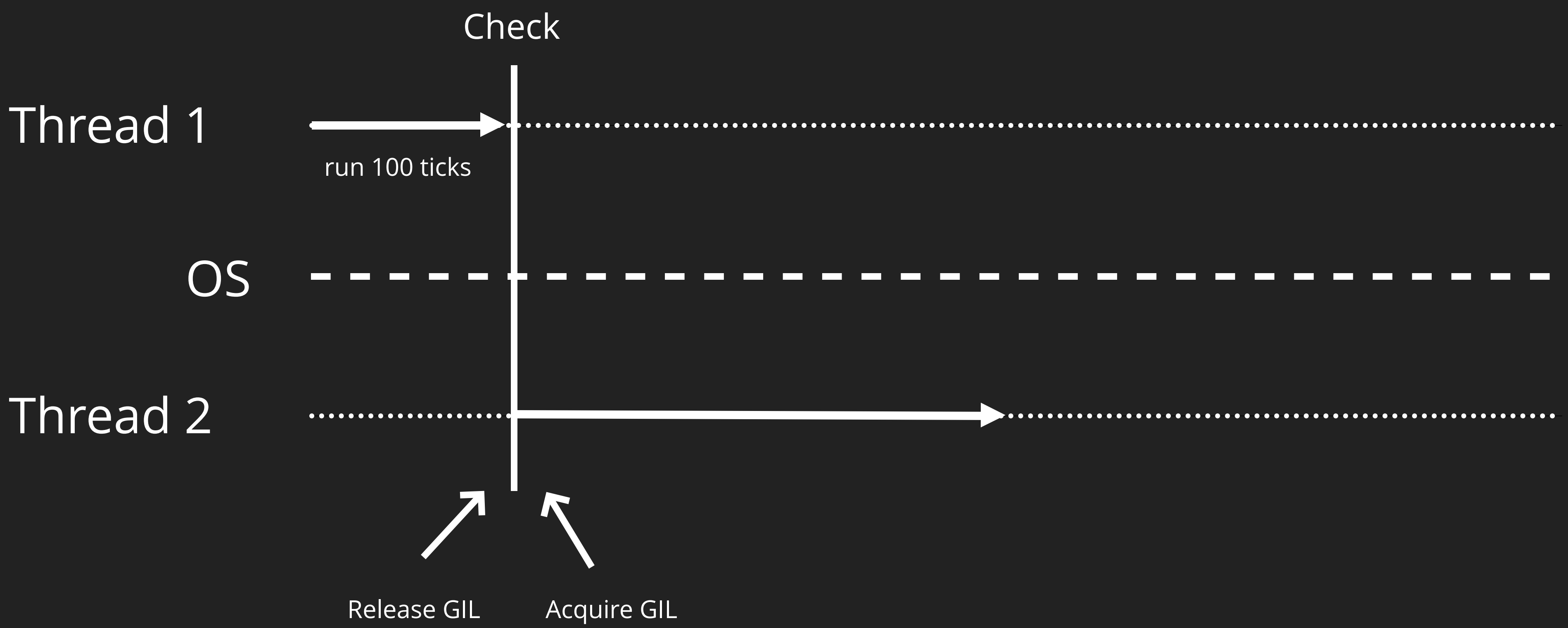
CPU Bound Module



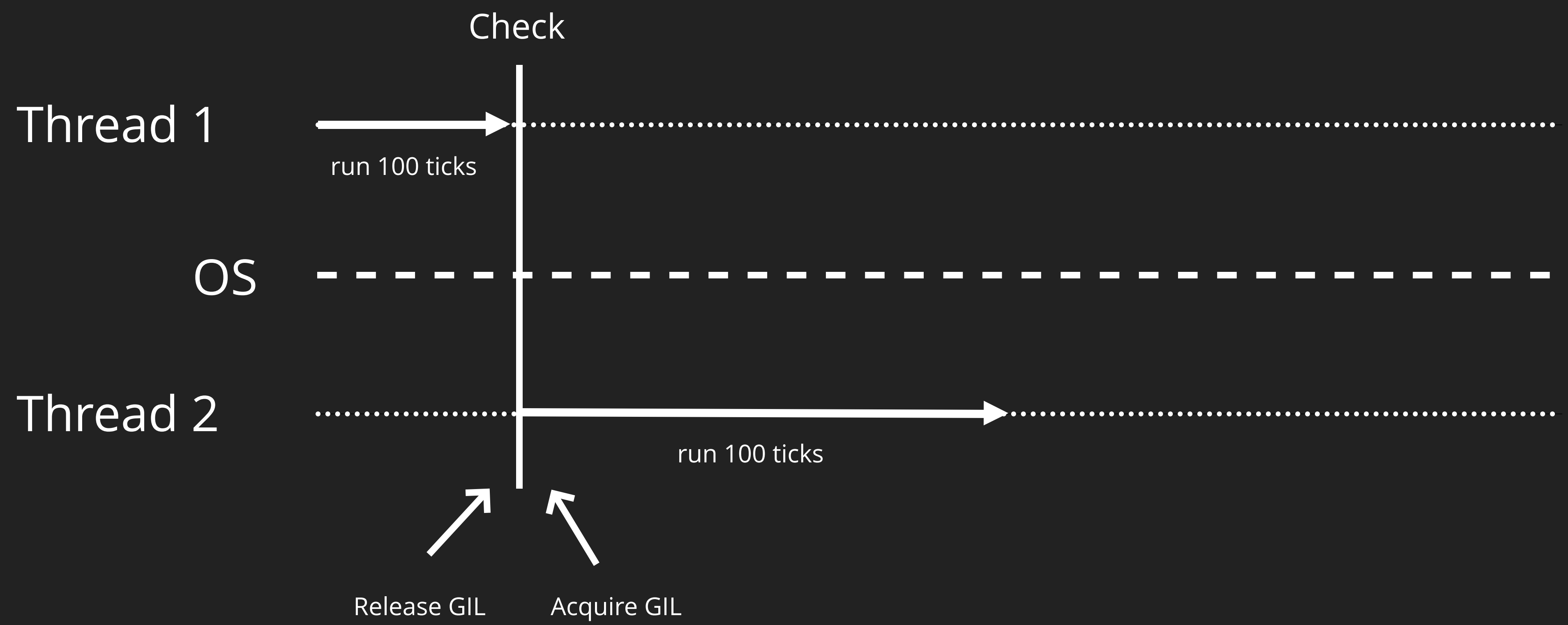
CPU Bound Module



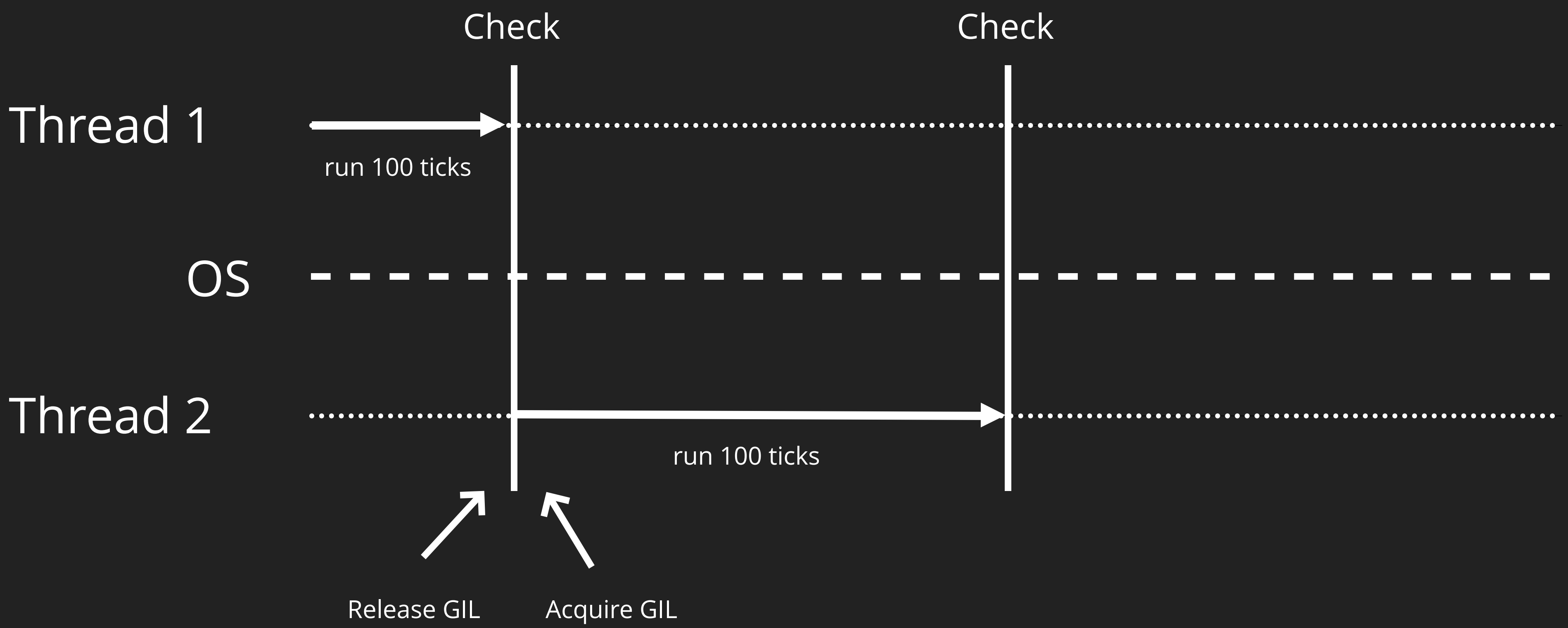
CPU Bound Module



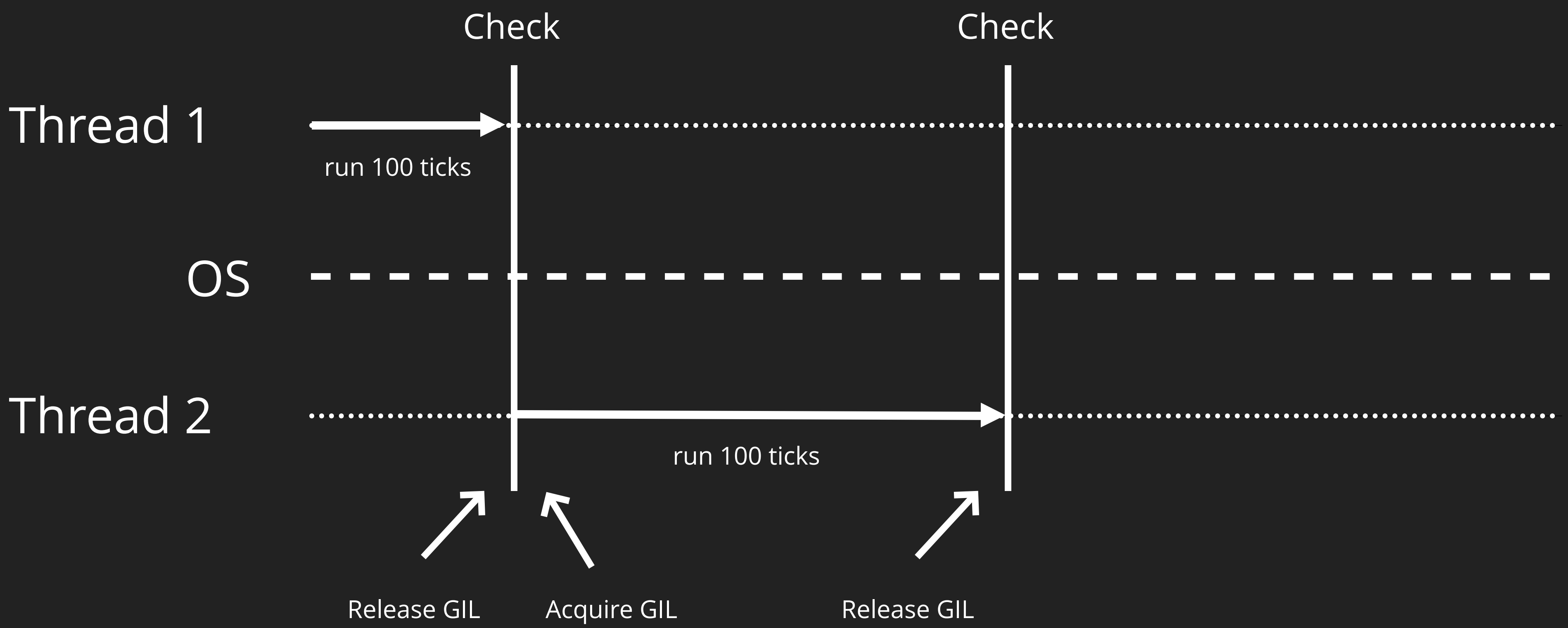
CPU Bound Module



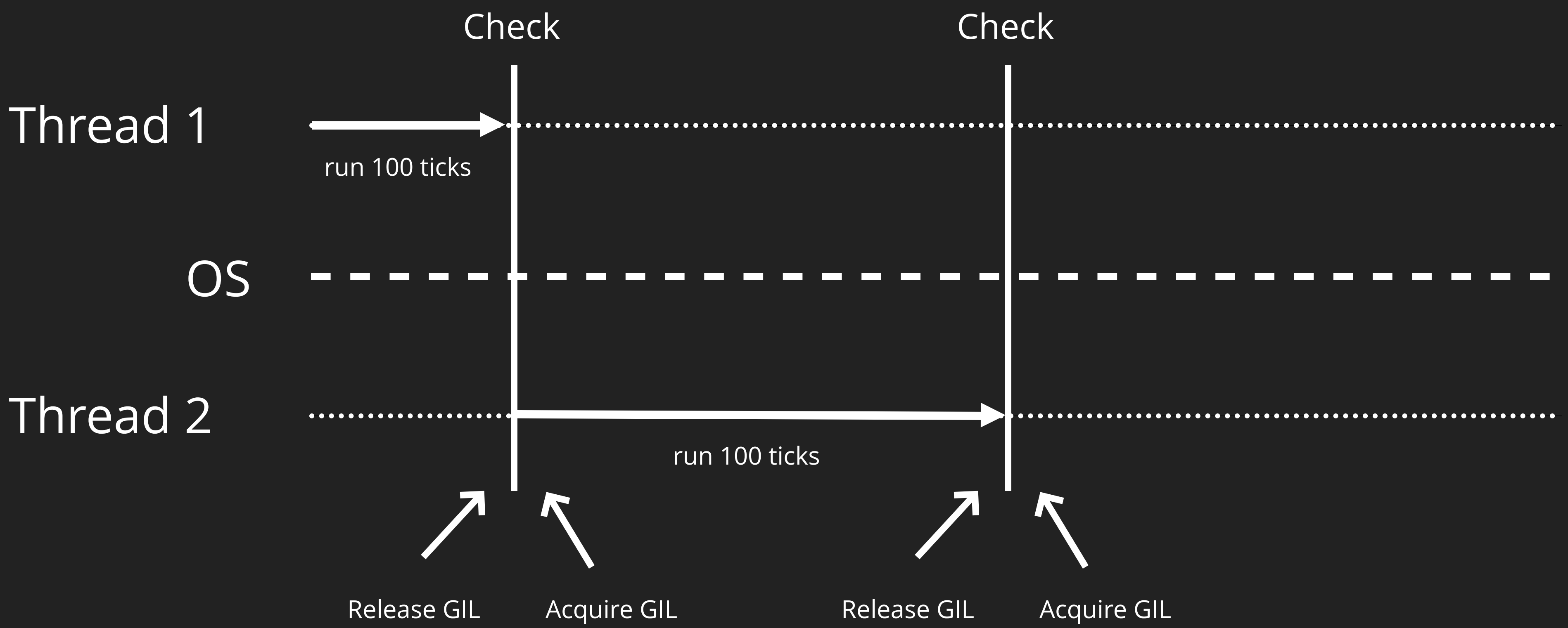
CPU Bound Module



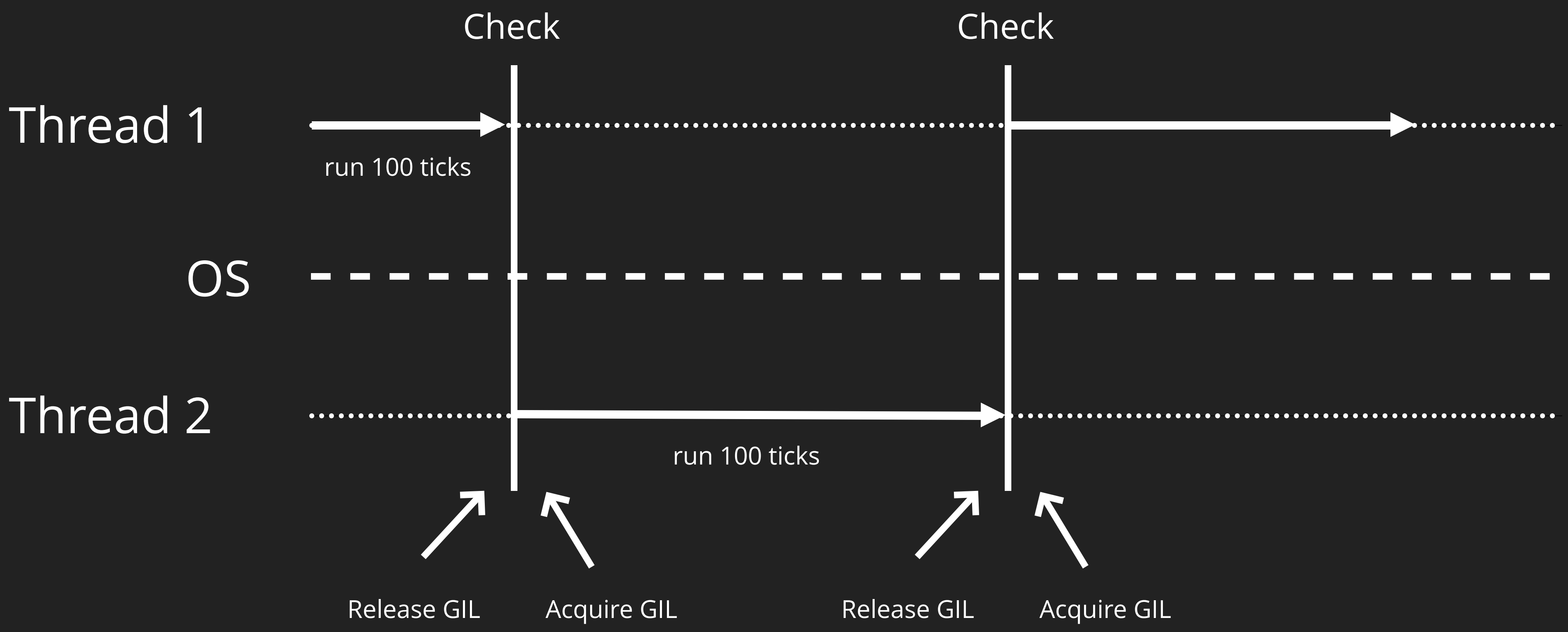
CPU Bound Module



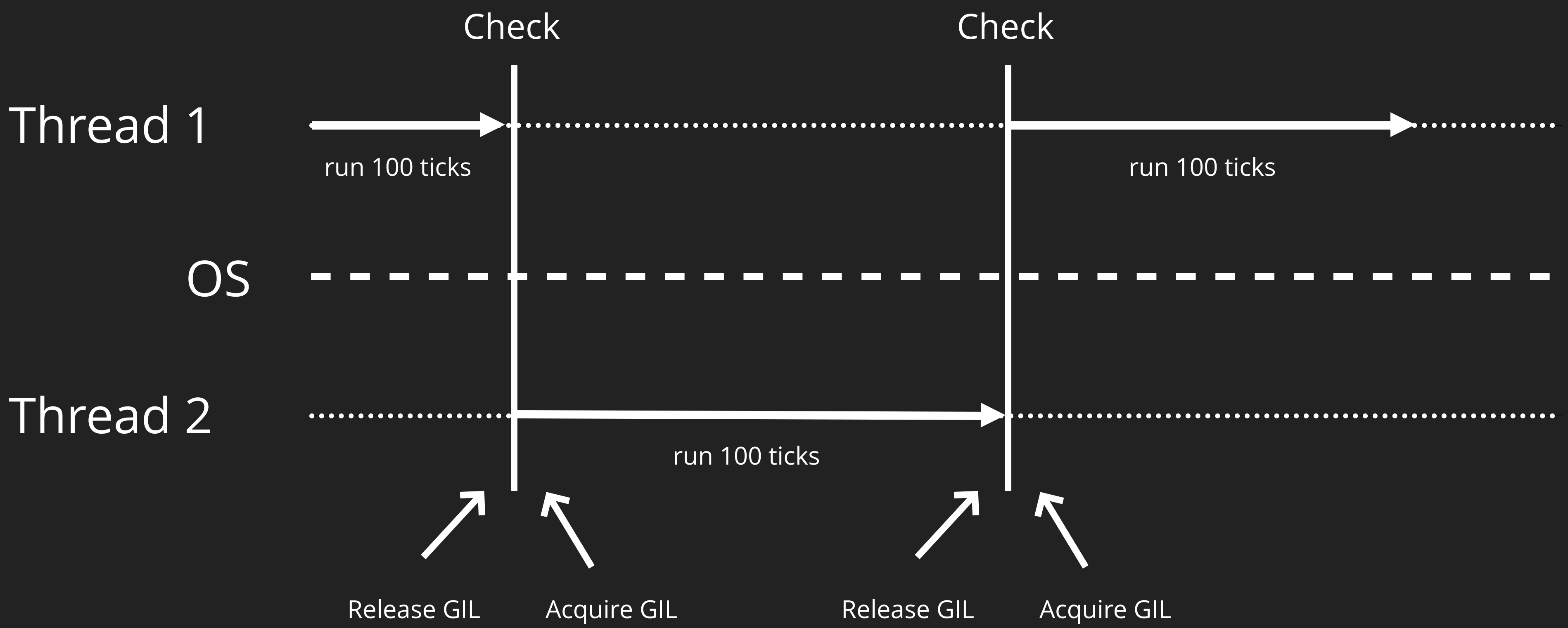
CPU Bound Module



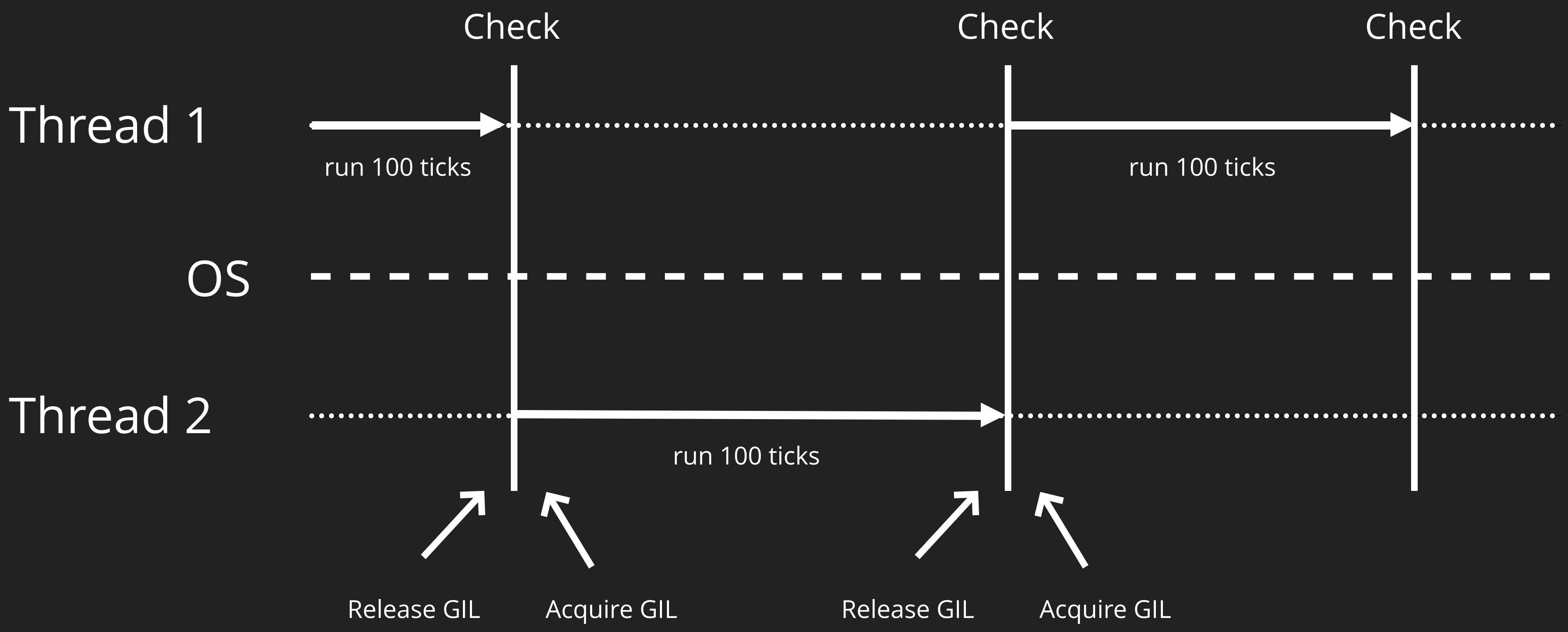
CPU Bound Module



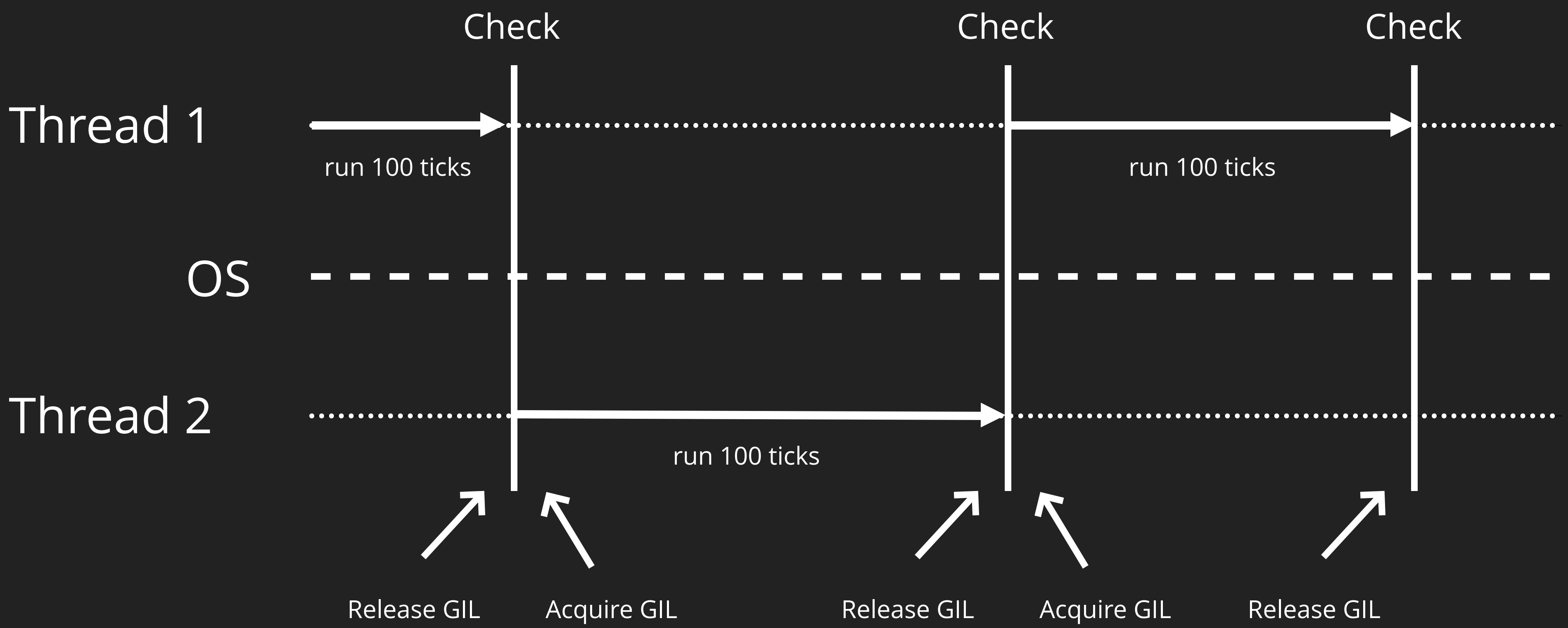
CPU Bound Module



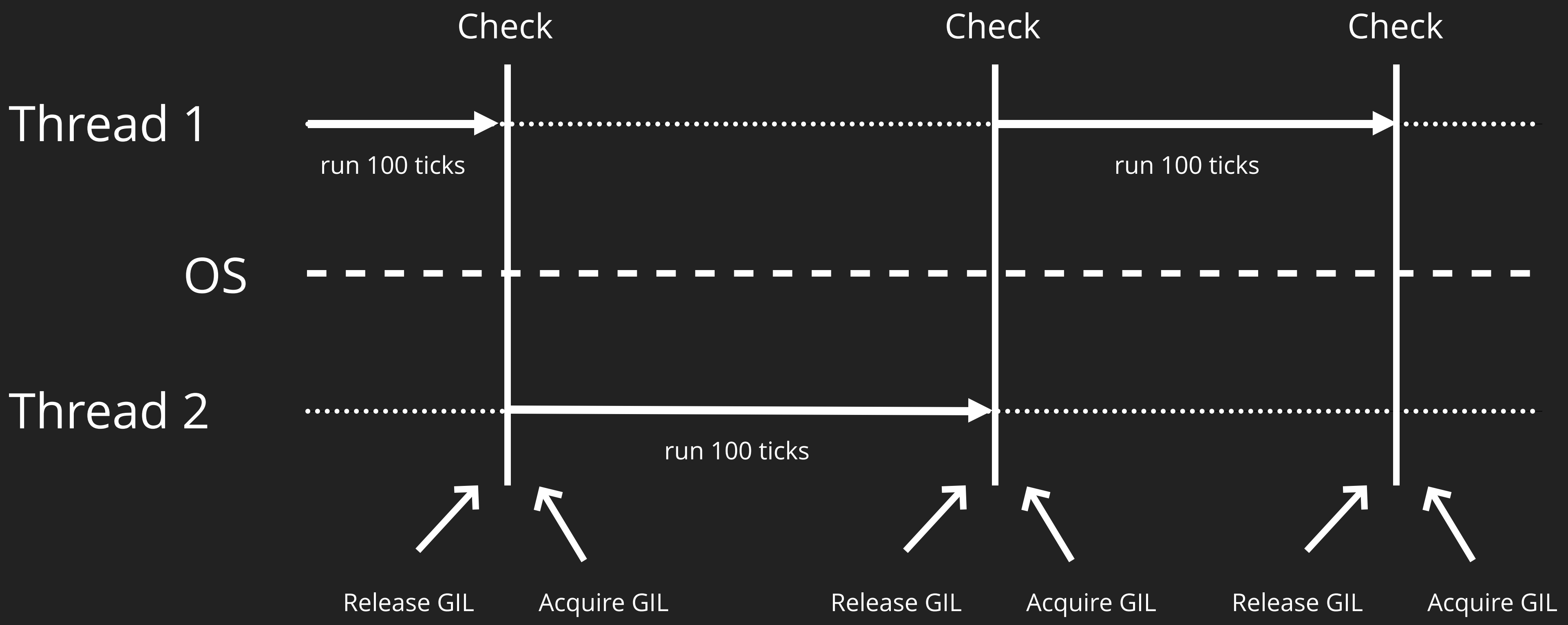
CPU Bound Module



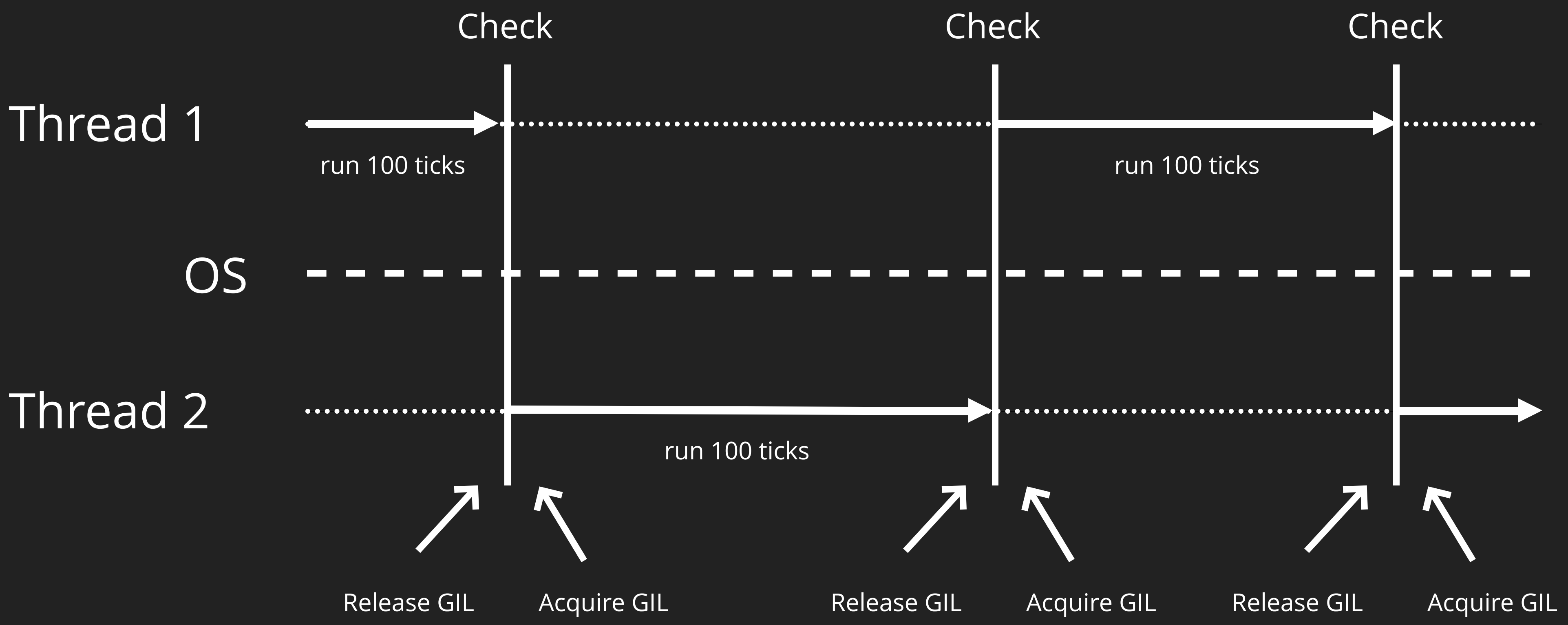
CPU Bound Module



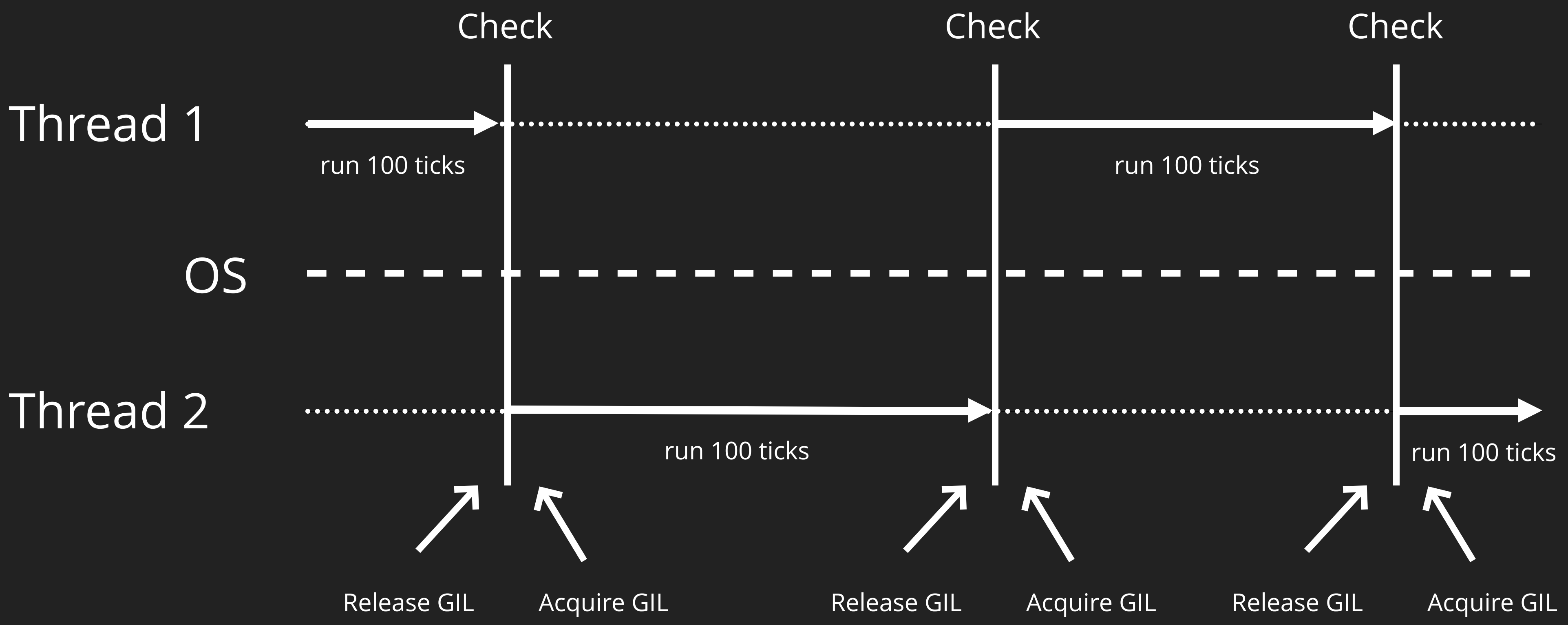
CPU Bound Module



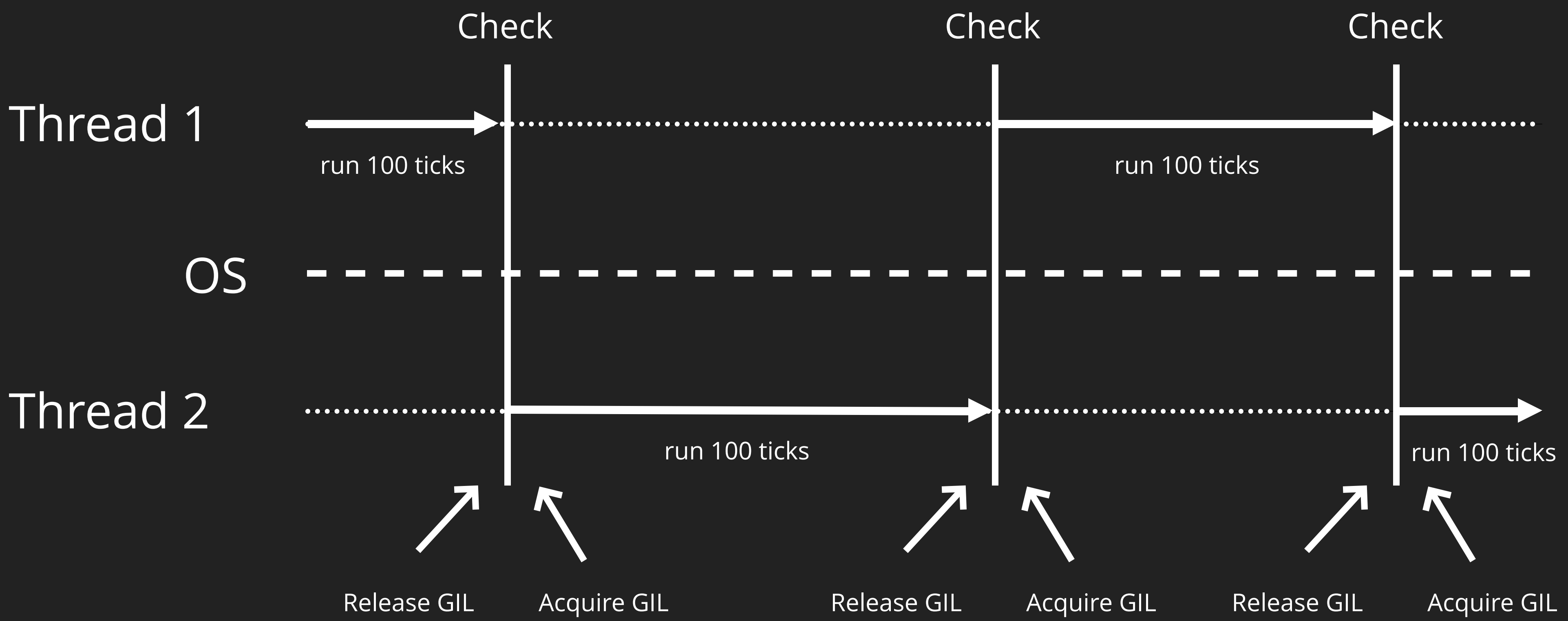
CPU Bound Module



CPU Bound Module

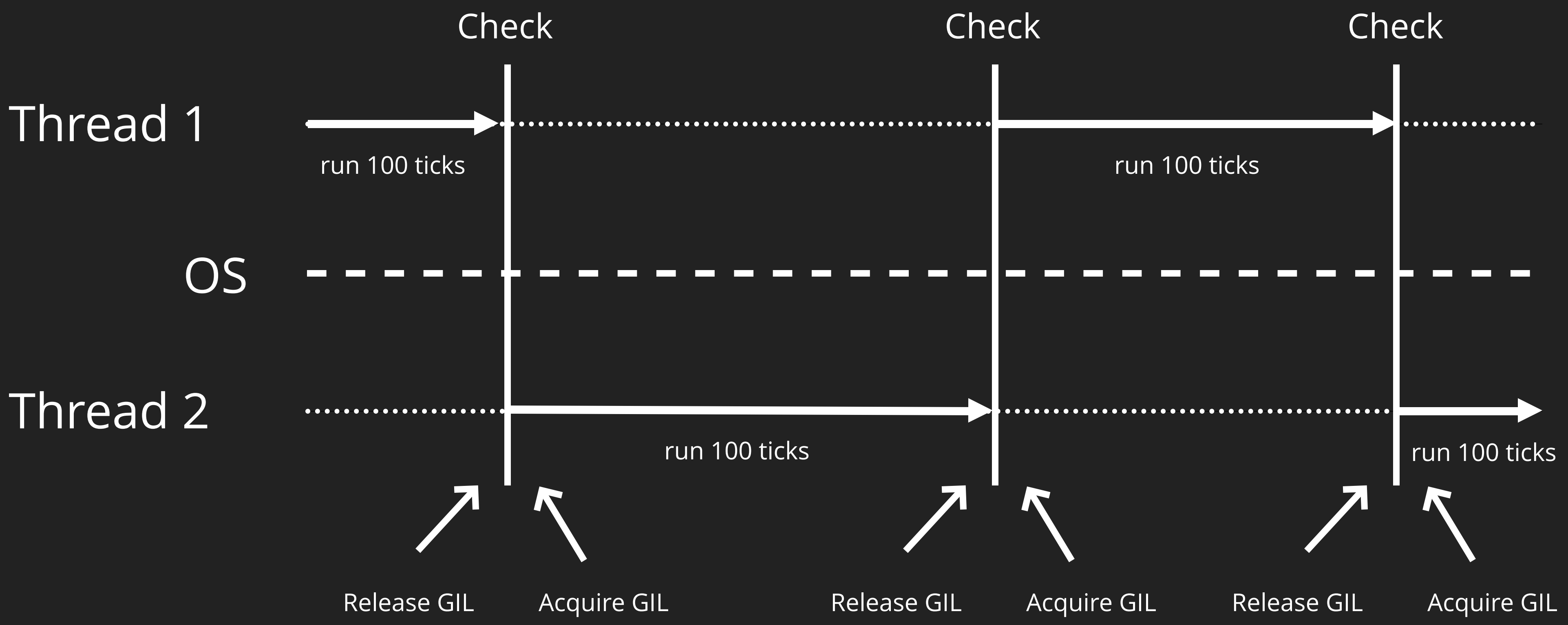


CPU Bound Module



- CPU bound threads that never perform I/O are handled as a special case

CPU Bound Module



- CPU bound threads that never perform I/O are handled as a special case
- A "check" occurs every 100 "ticks"

Some OS Stuff

Some OS Stuff

- The operating system has a priority queue of threads/processes ready to run

Some OS Stuff

- The operating system has a priority queue of threads/processes ready to run
- Signaled threads simply enter that queue

Some OS Stuff

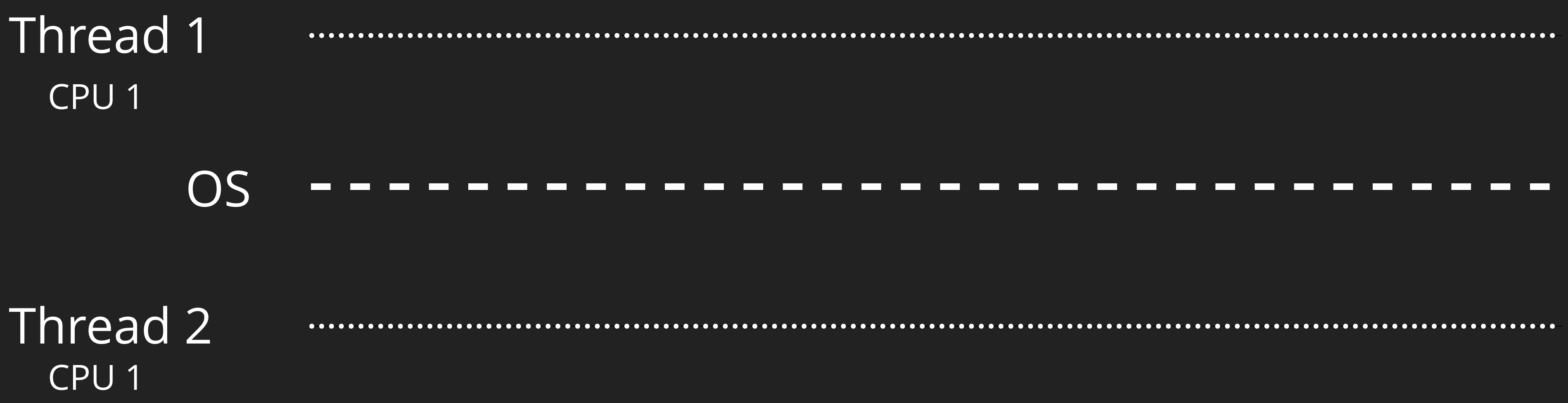
- The operating system has a priority queue of threads/processes ready to run
- Signaled threads simply enter that queue
- The operating system then runs the process or thread with the highest priority

Some OS Stuff

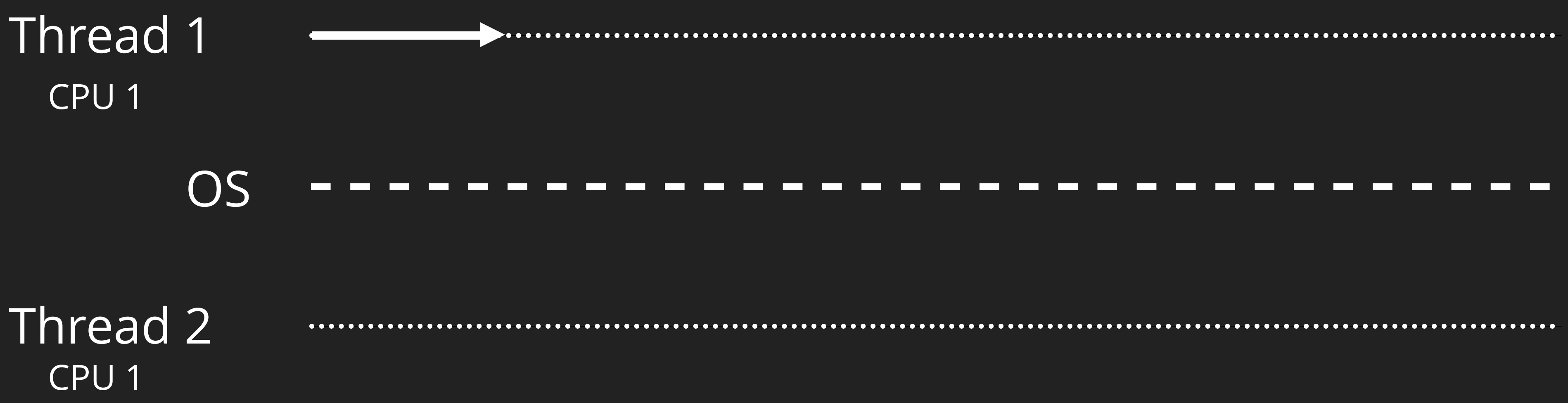
- The operating system has a priority queue of threads/processes ready to run
- Signaled threads simply enter that queue
- The operating system then runs the process or thread with the highest priority
- It may or may not be the signaled thread

Real CPU Bound Thread Situation

Real CPU Bound Thread Situation



Real CPU Bound Thread Situation



Thread 1

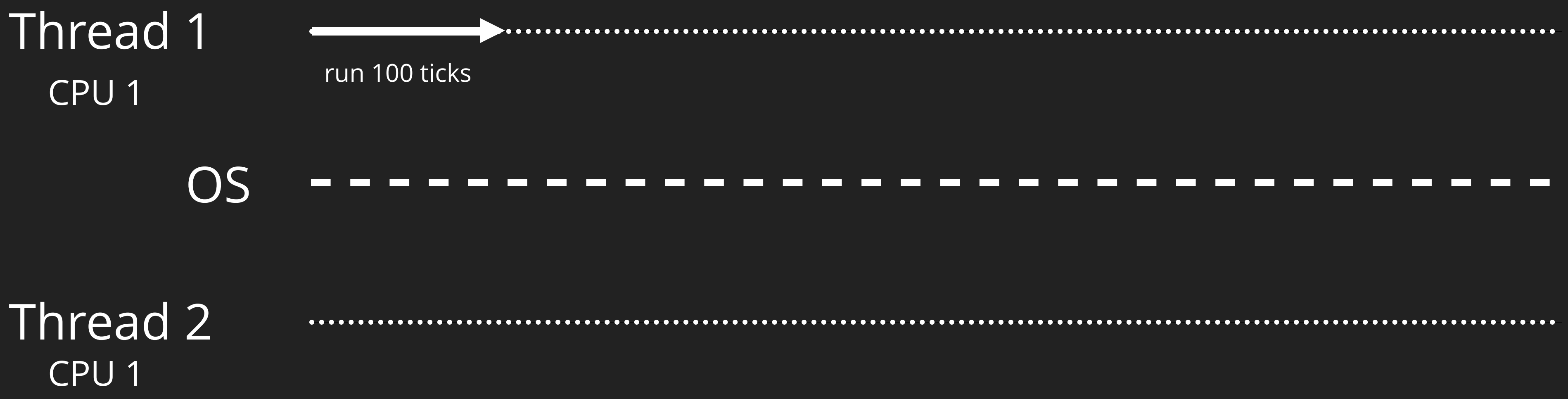
CPU 1

OS

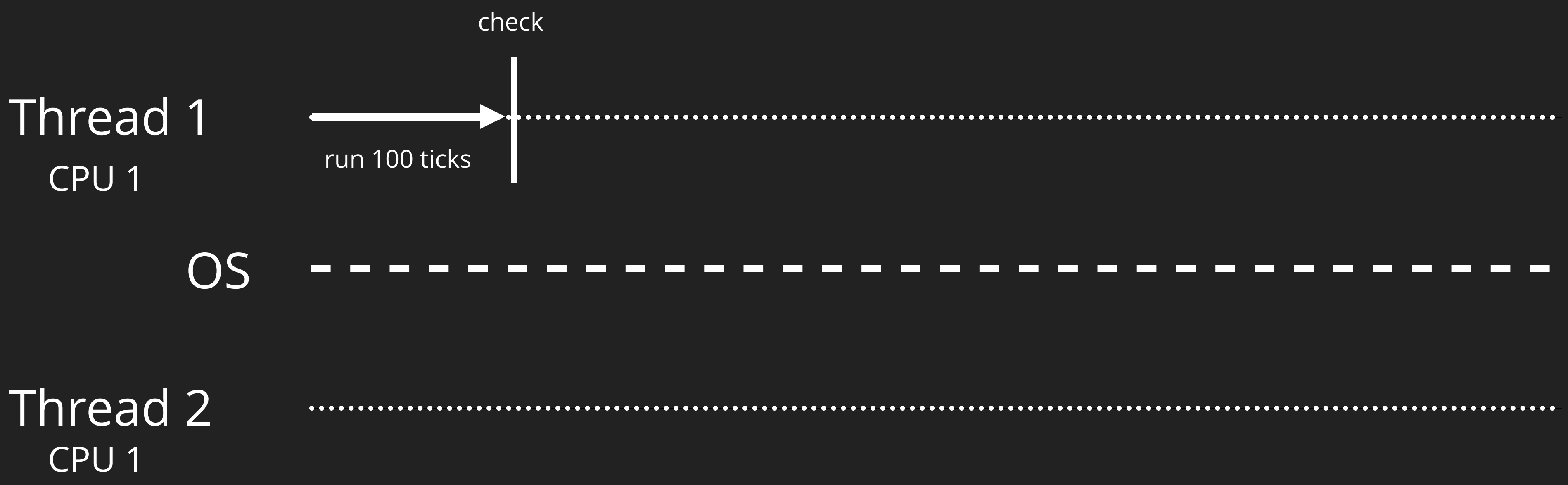
Thread 2

CPU 1

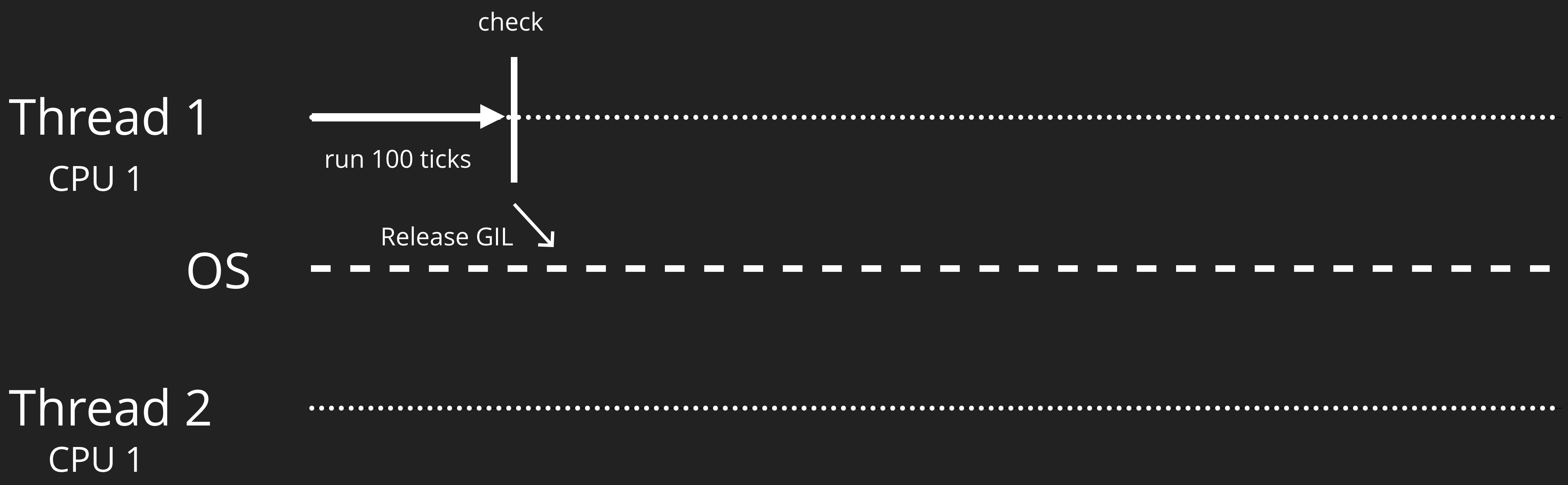
Real CPU Bound Thread Situation



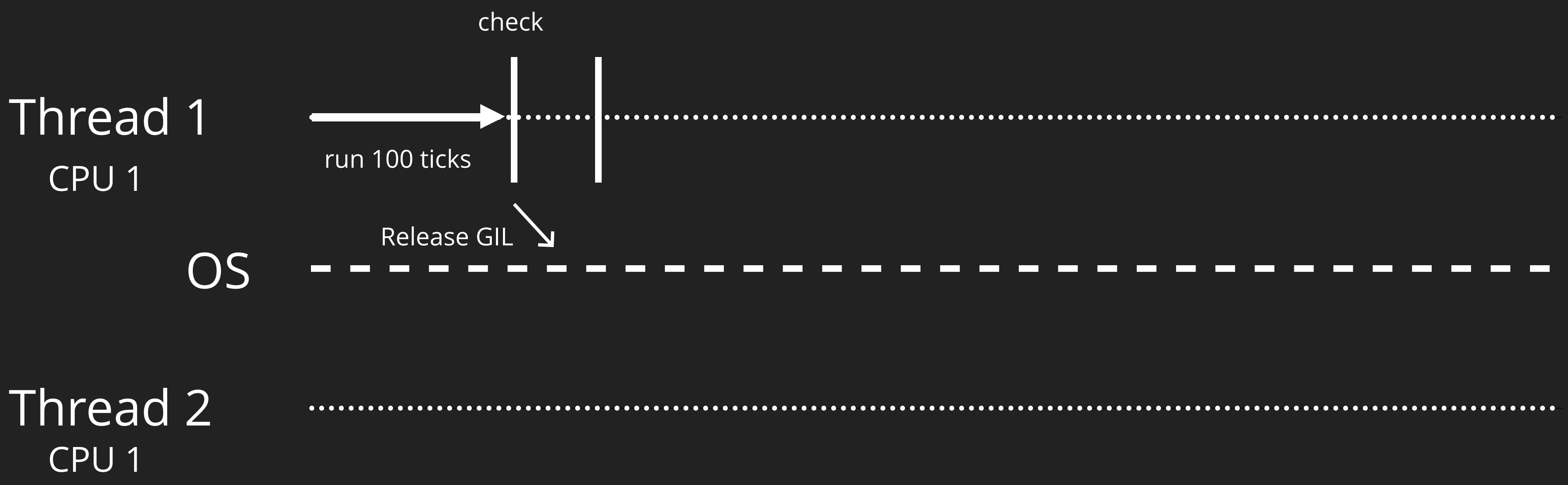
Real CPU Bound Thread Situation



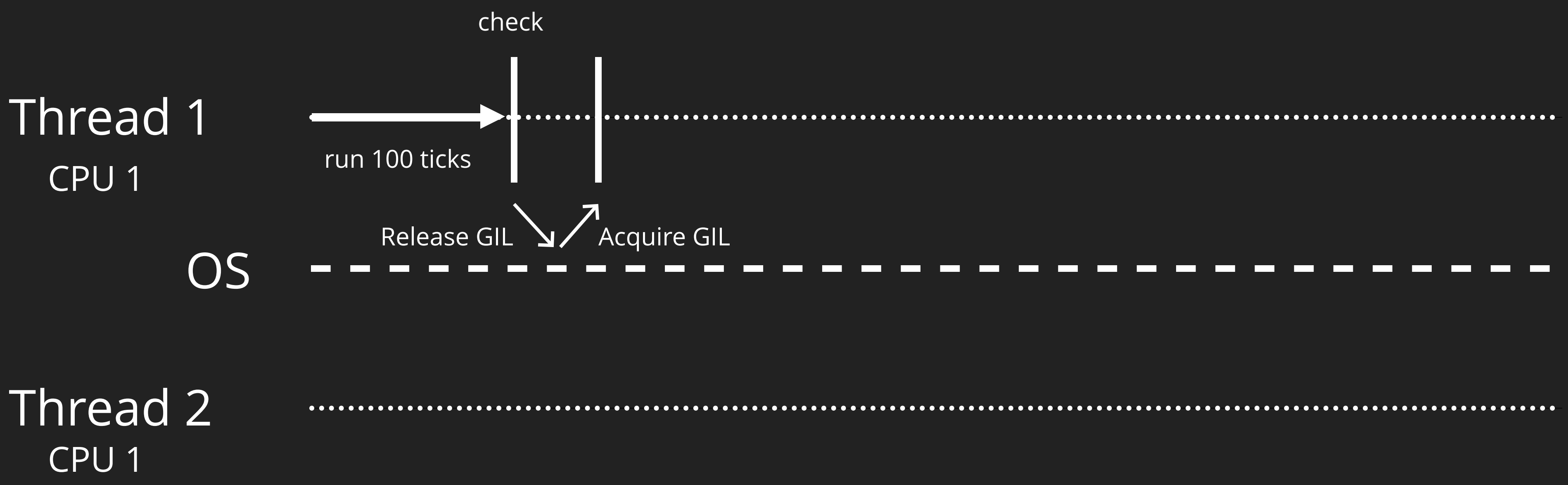
Real CPU Bound Thread Situation



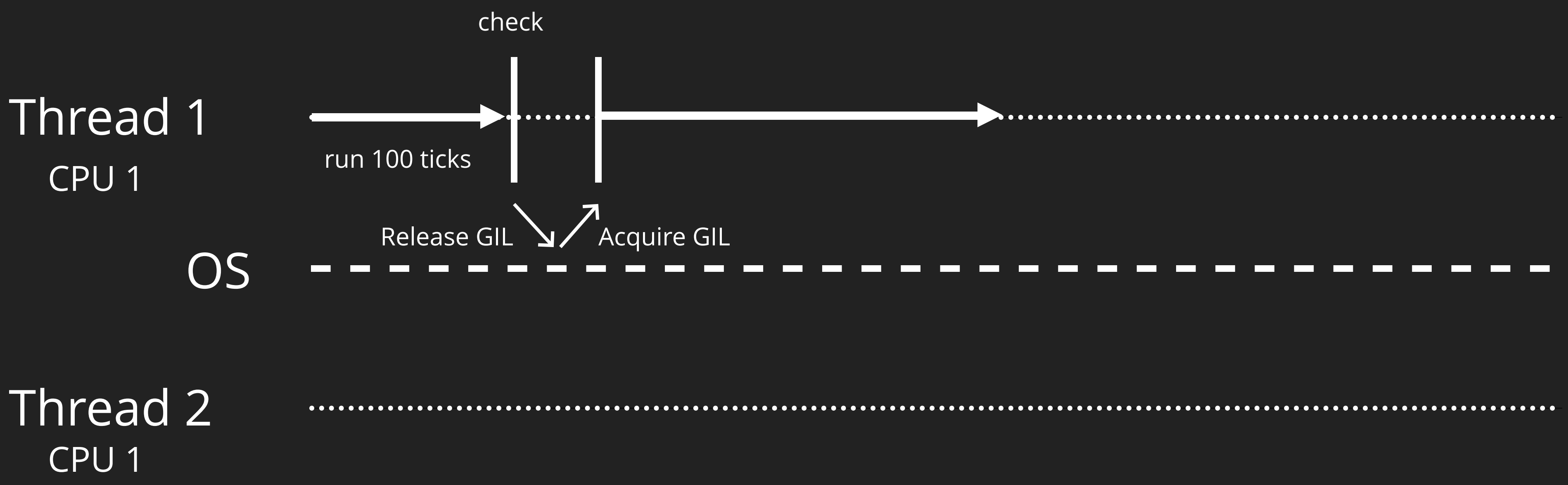
Real CPU Bound Thread Situation



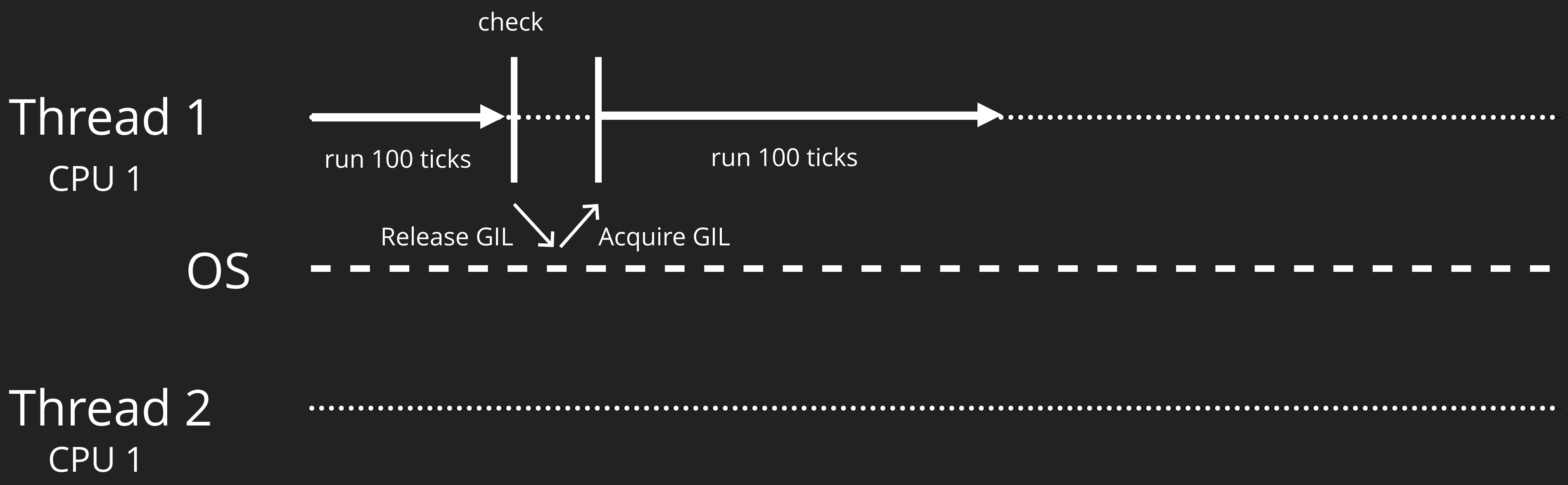
Real CPU Bound Thread Situation



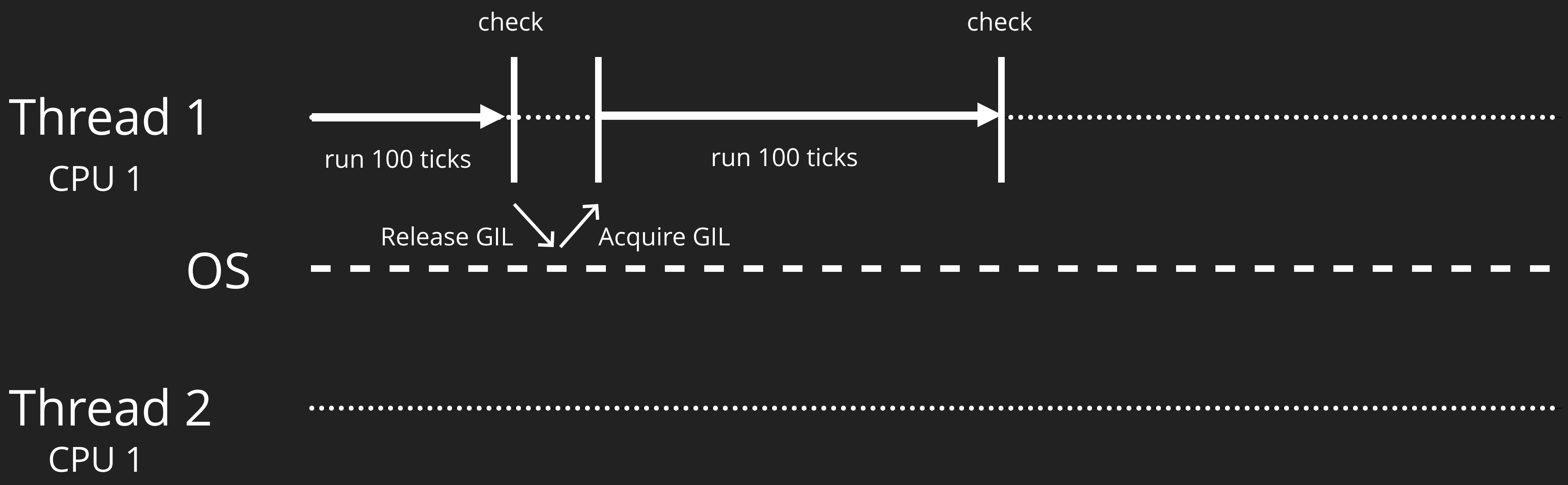
Real CPU Bound Thread Situation



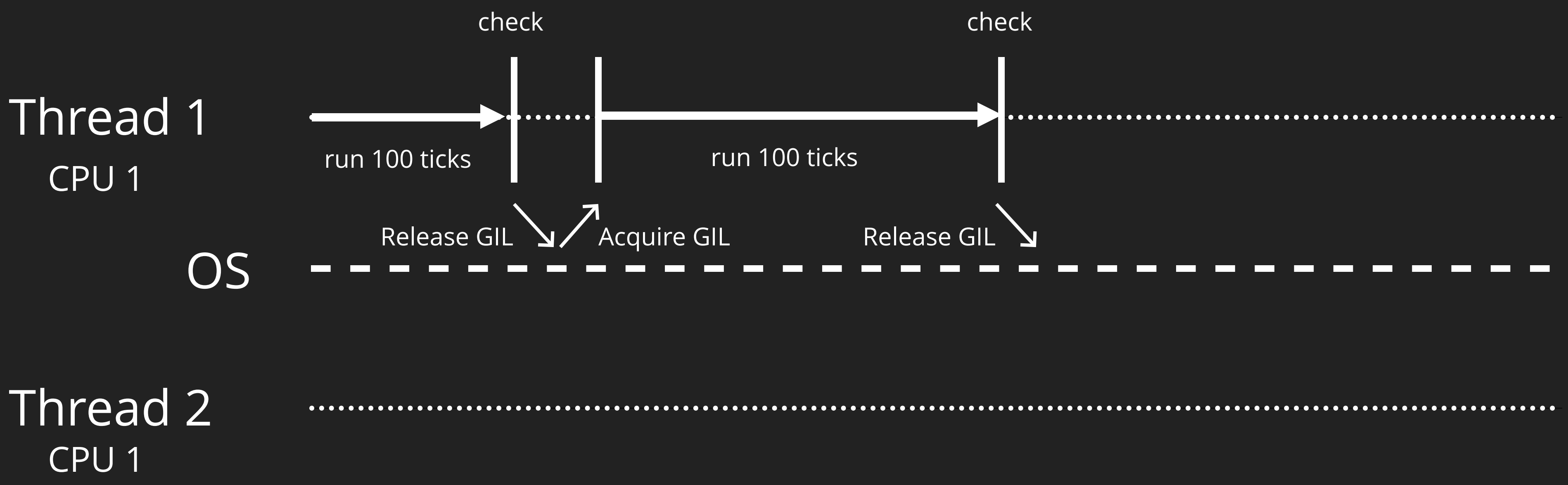
Real CPU Bound Thread Situation



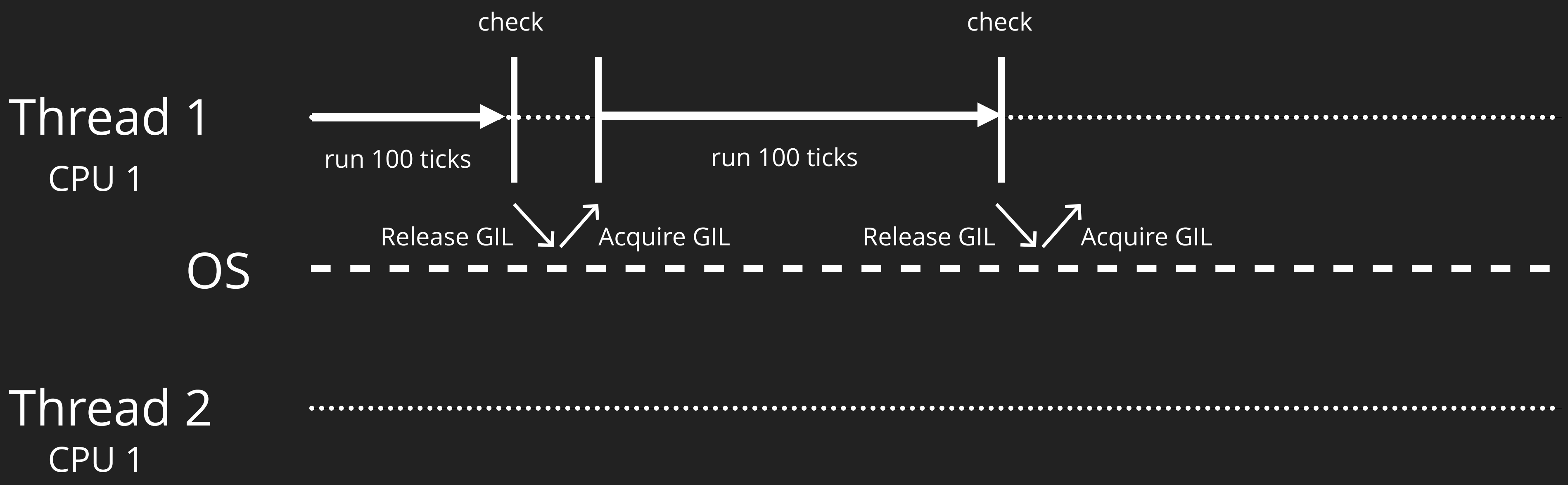
Real CPU Bound Thread Situation



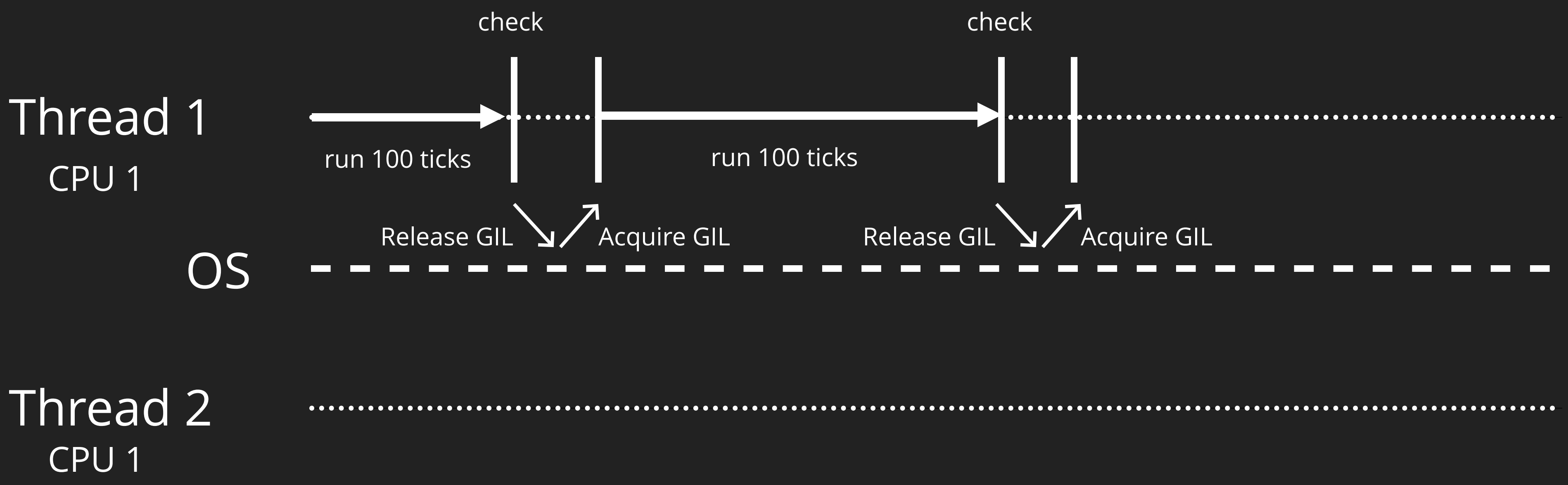
Real CPU Bound Thread Situation



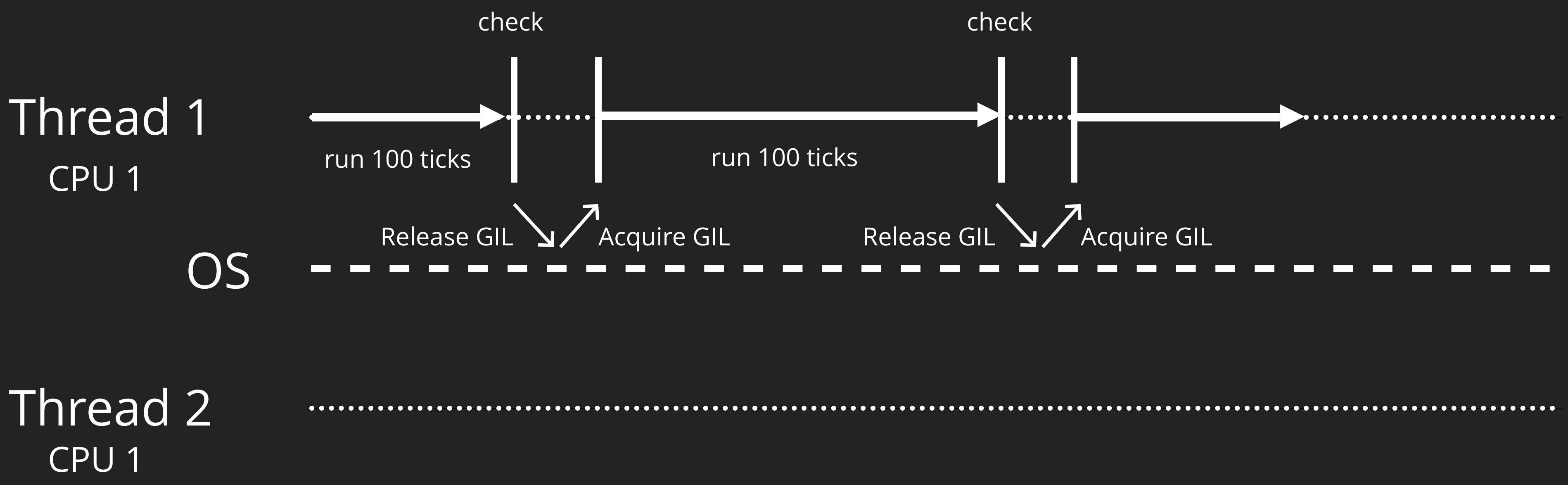
Real CPU Bound Thread Situation



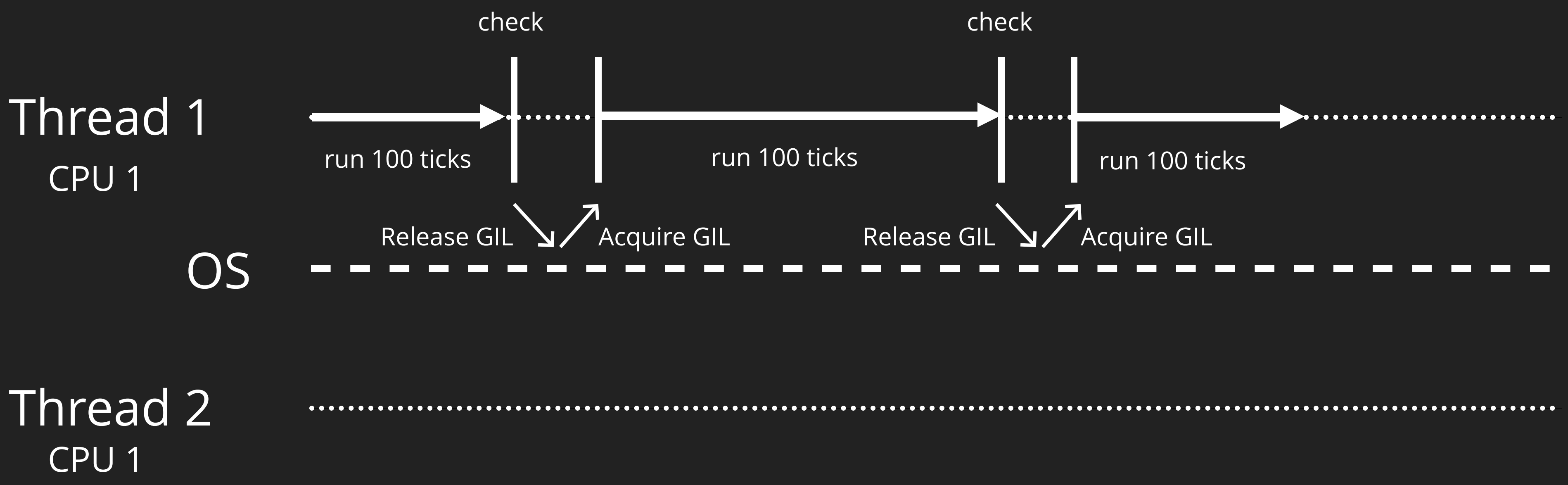
Real CPU Bound Thread Situation



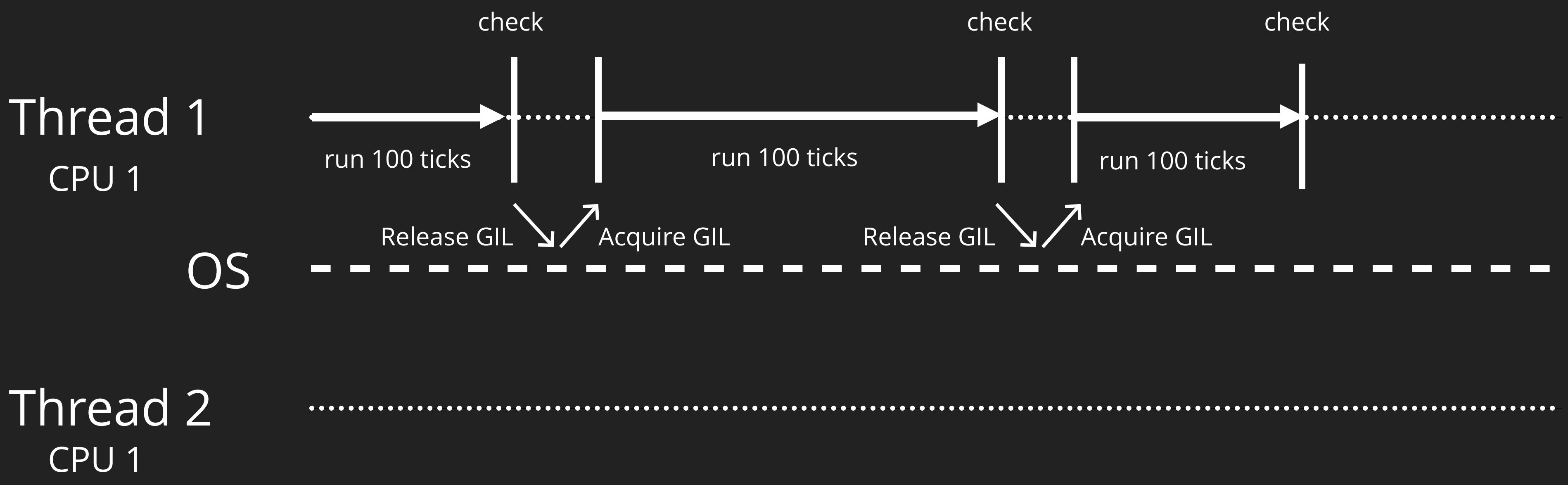
Real CPU Bound Thread Situation



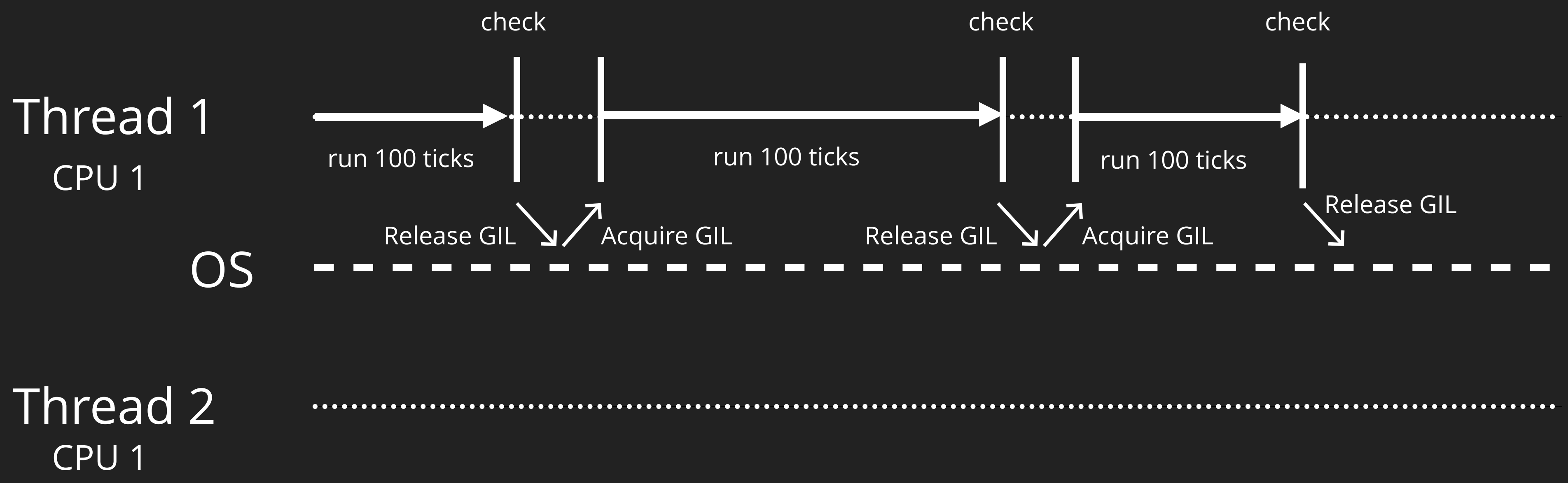
Real CPU Bound Thread Situation



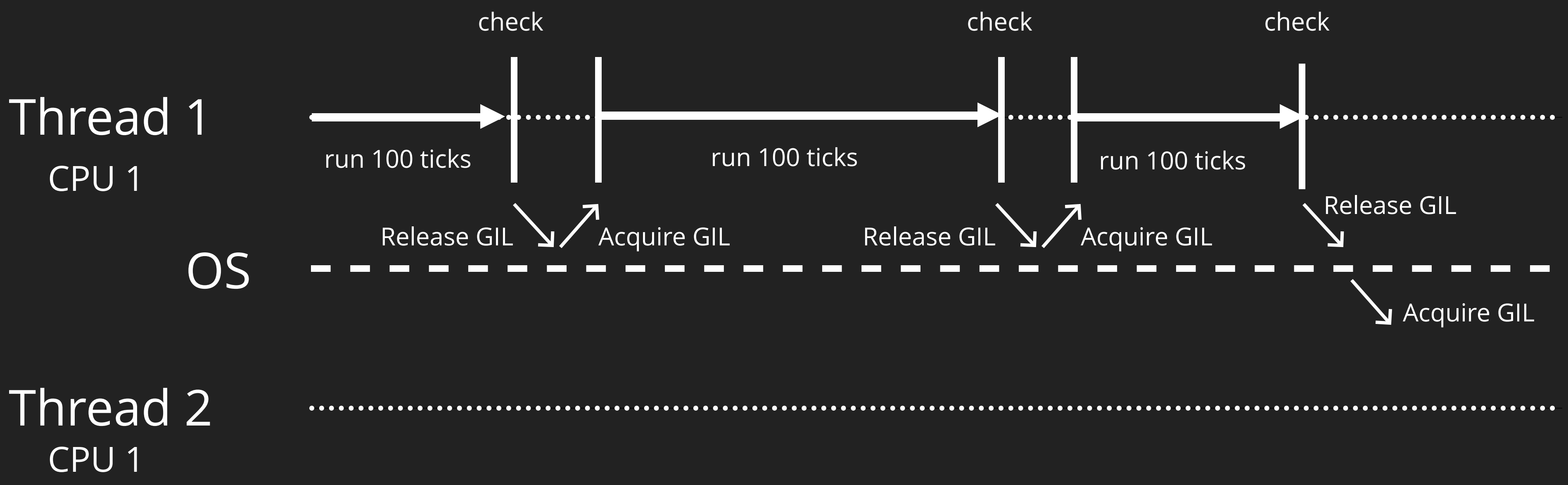
Real CPU Bound Thread Situation



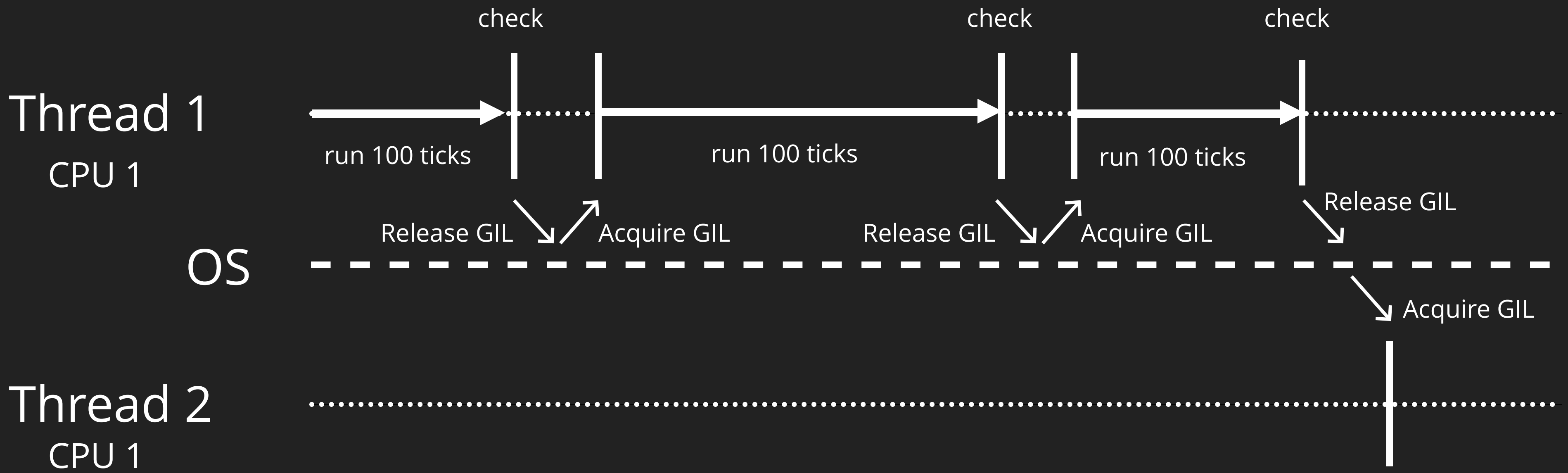
Real CPU Bound Thread Situation



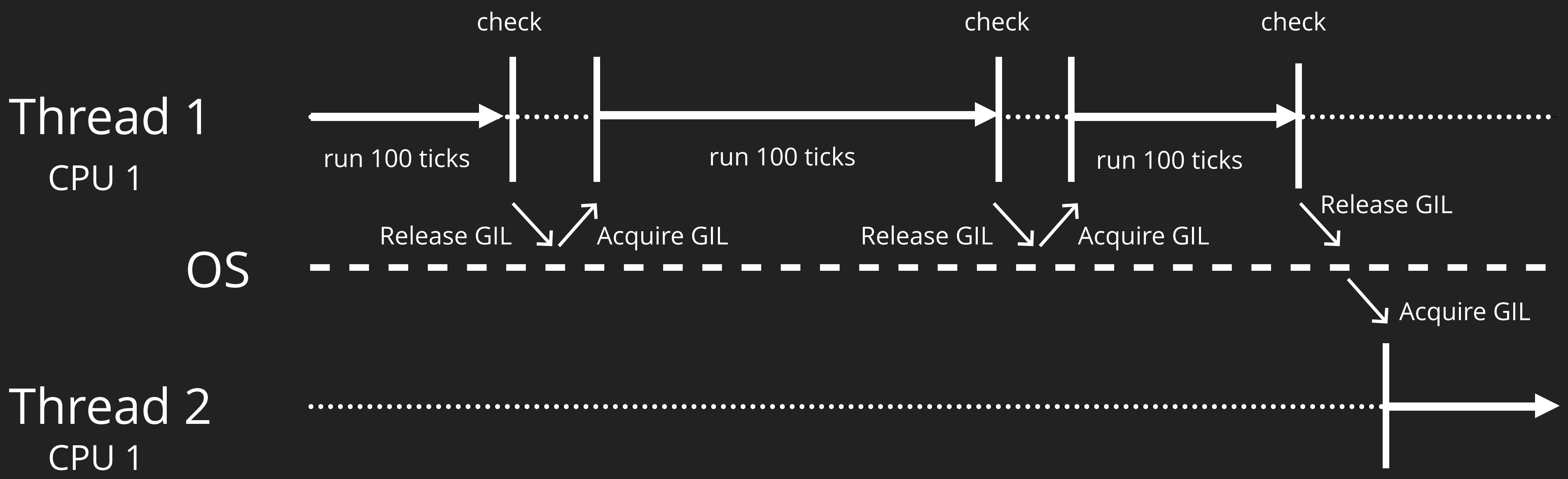
Real CPU Bound Thread Situation



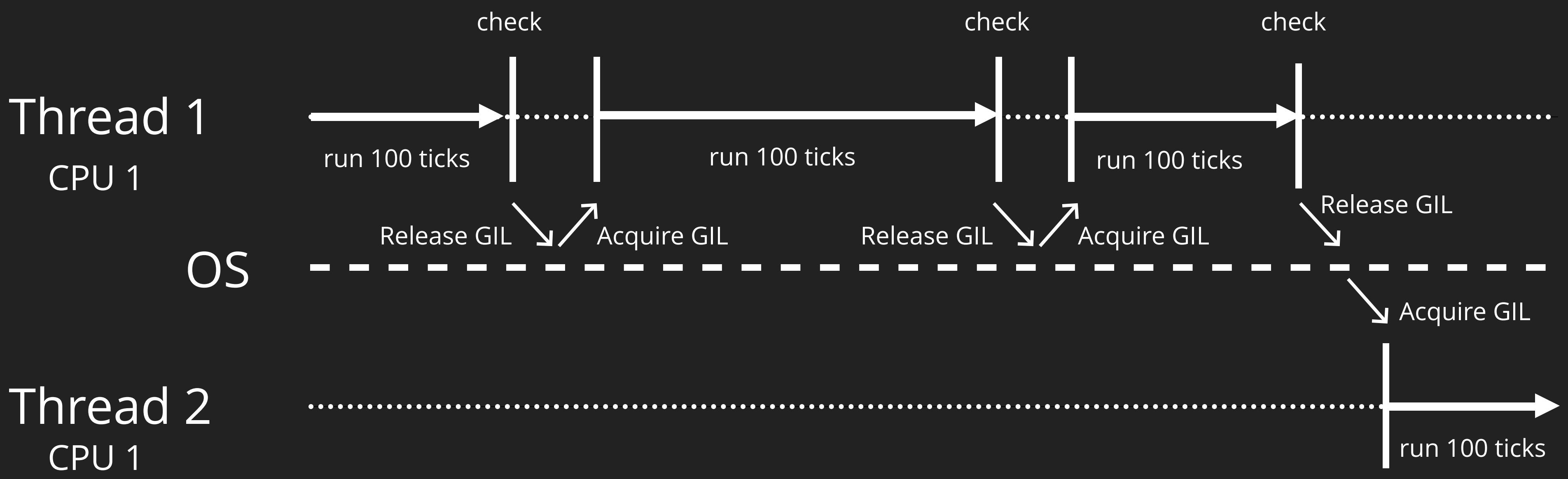
Real CPU Bound Thread Situation



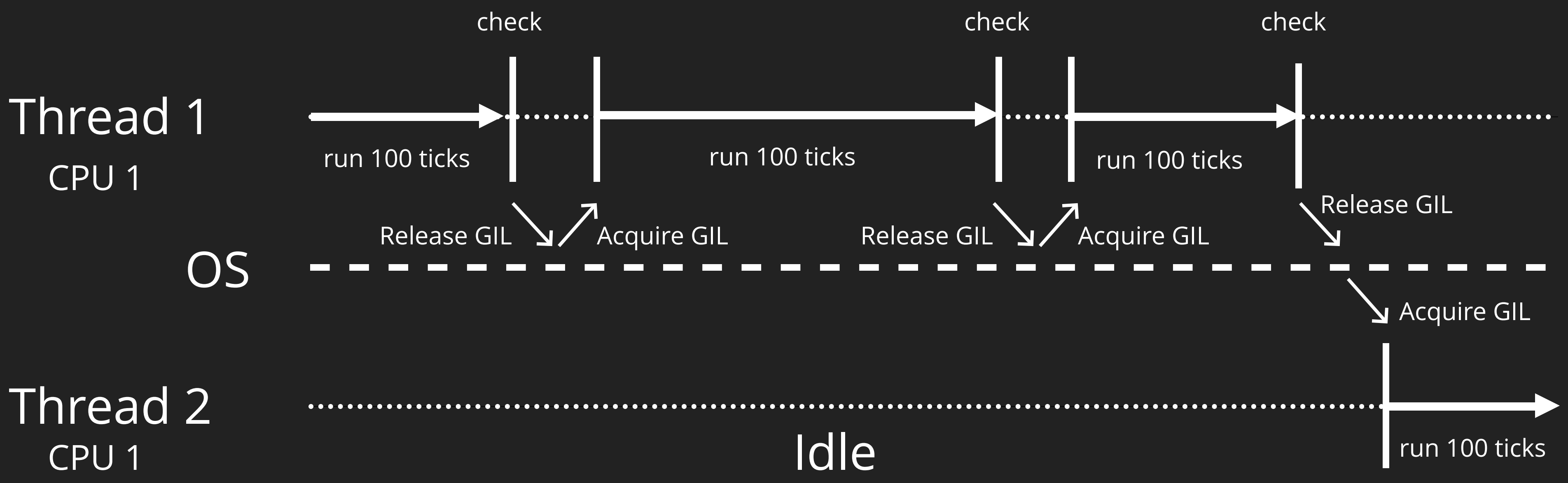
Real CPU Bound Thread Situation



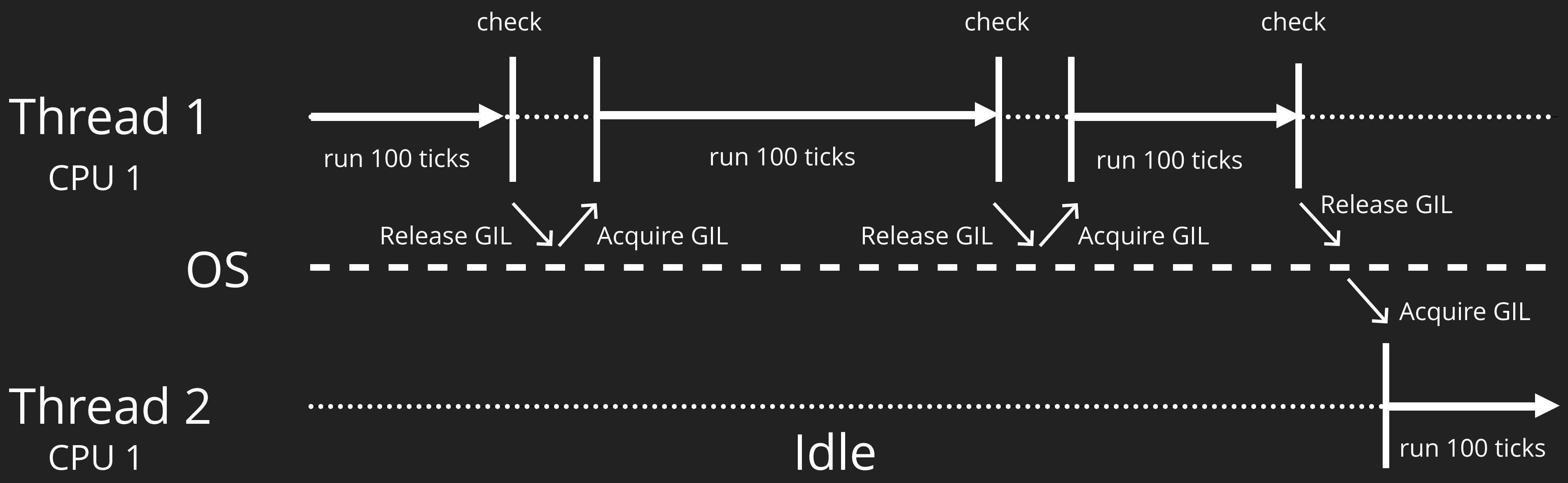
Real CPU Bound Thread Situation



Real CPU Bound Thread Situation



Real CPU Bound Thread Situation



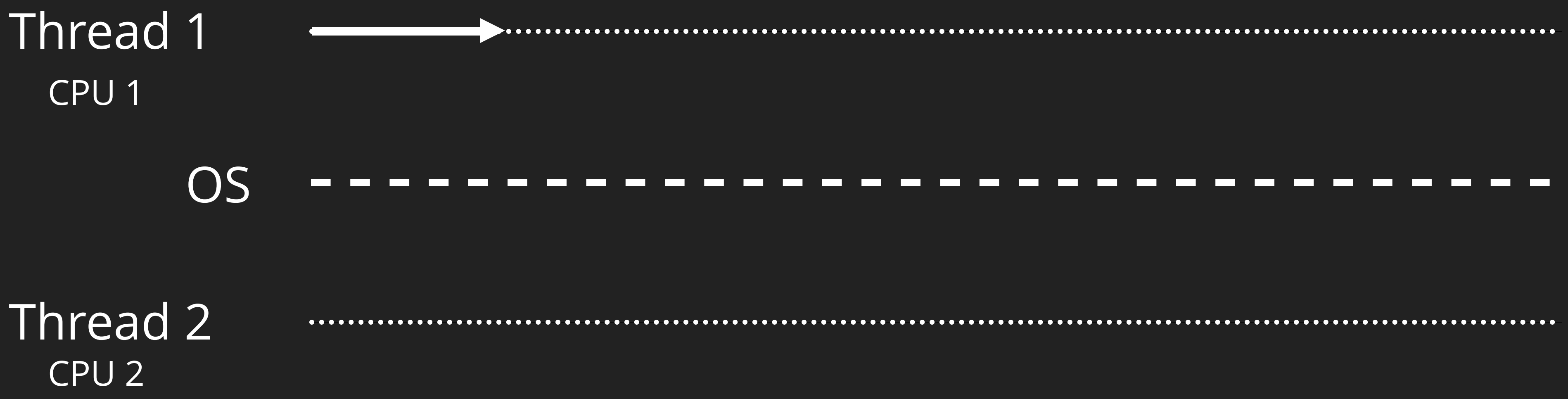
- Hundreds to thousands of checks might occur before a thread context switch

Thread Thrashing

Thread Thrashing

Thread 1
CPU 1
OS
Thread 2
CPU 2

Thread Thrashing



Thread 1

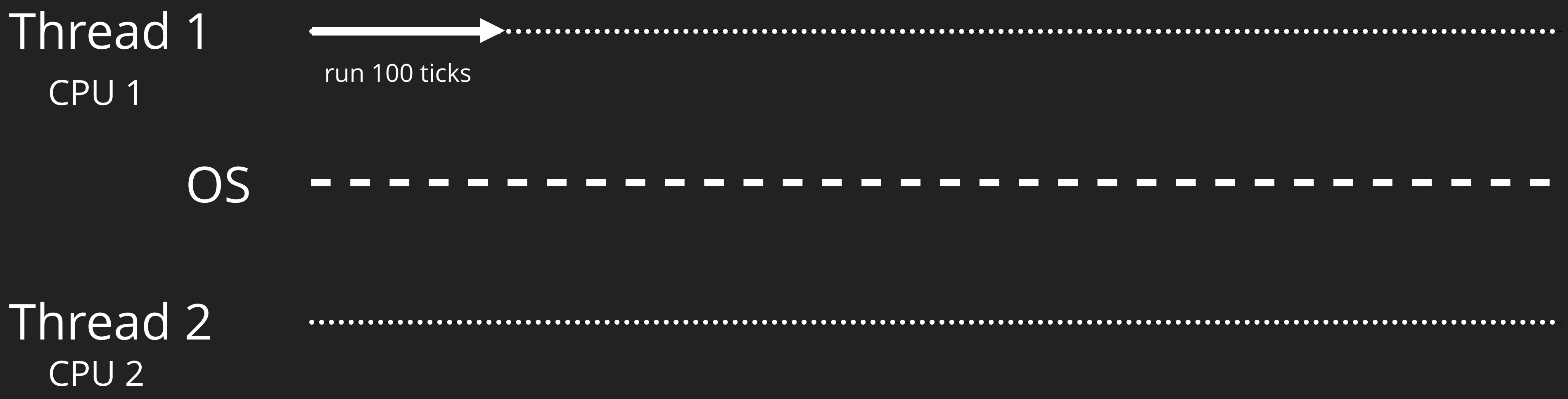
CPU 1

OS

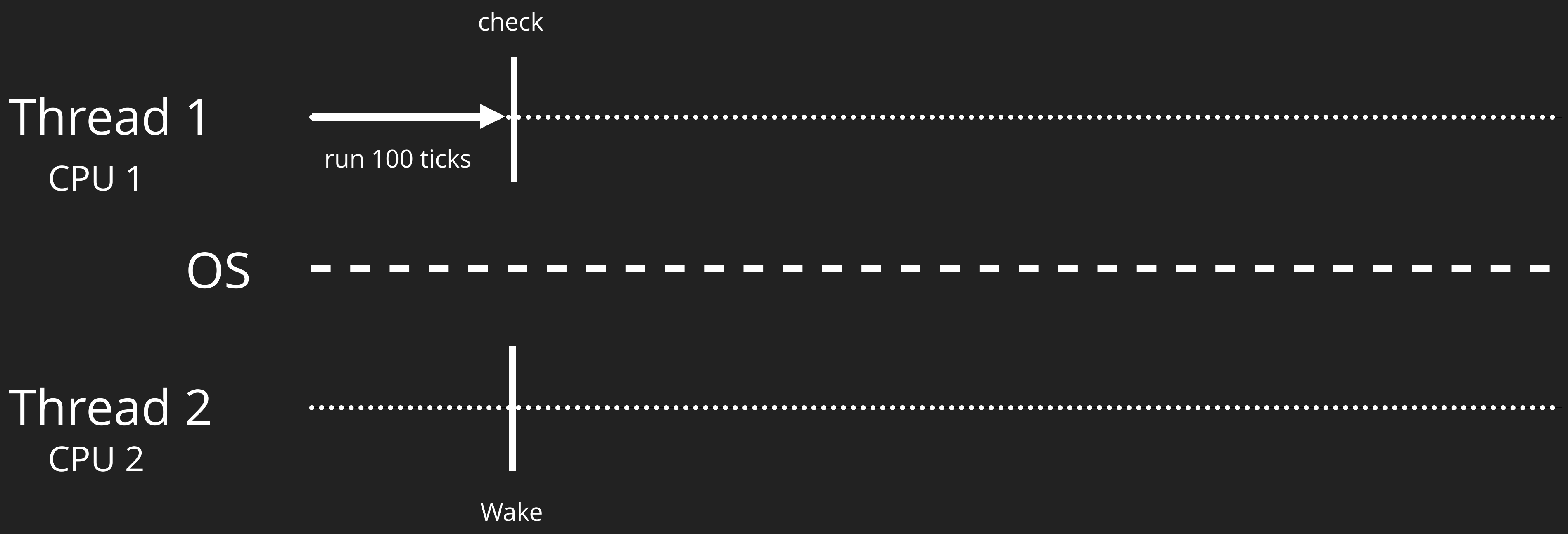
Thread 2

CPU 2

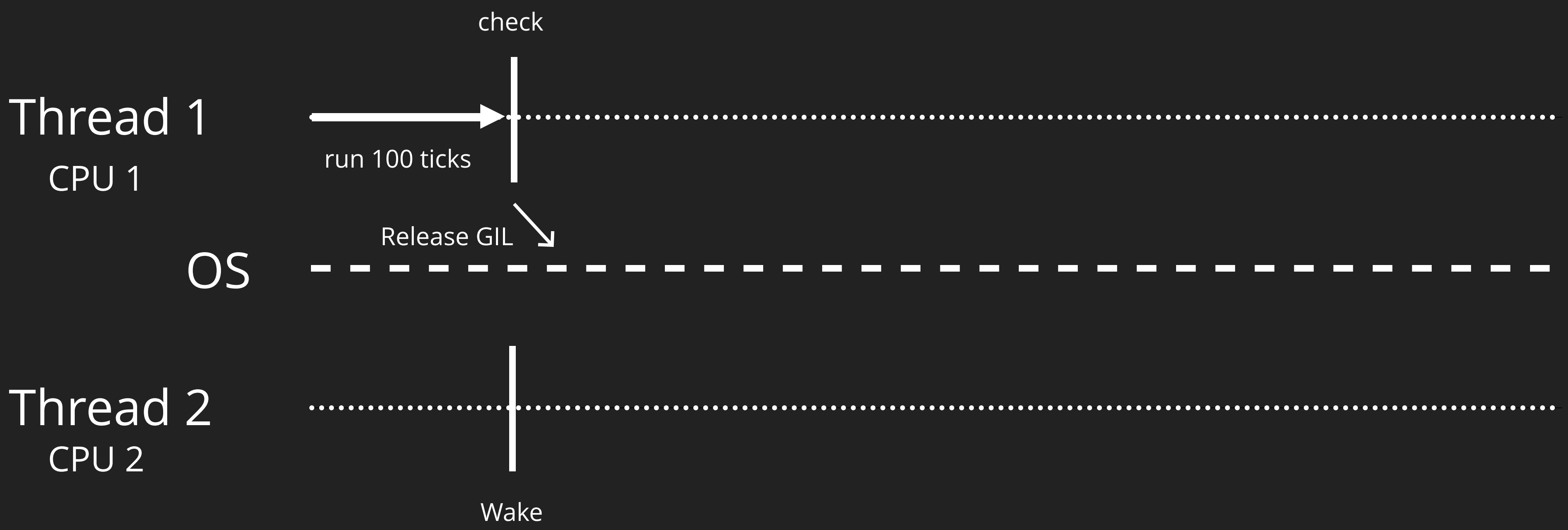
Thread Thrashing



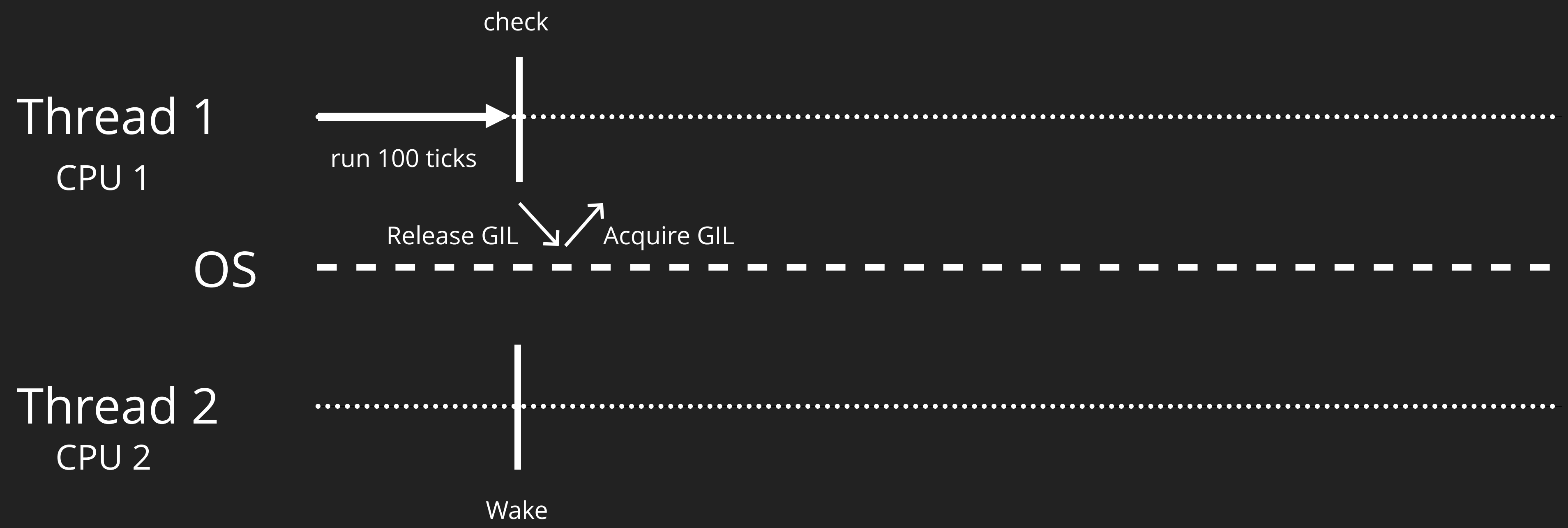
Thread Thrashing



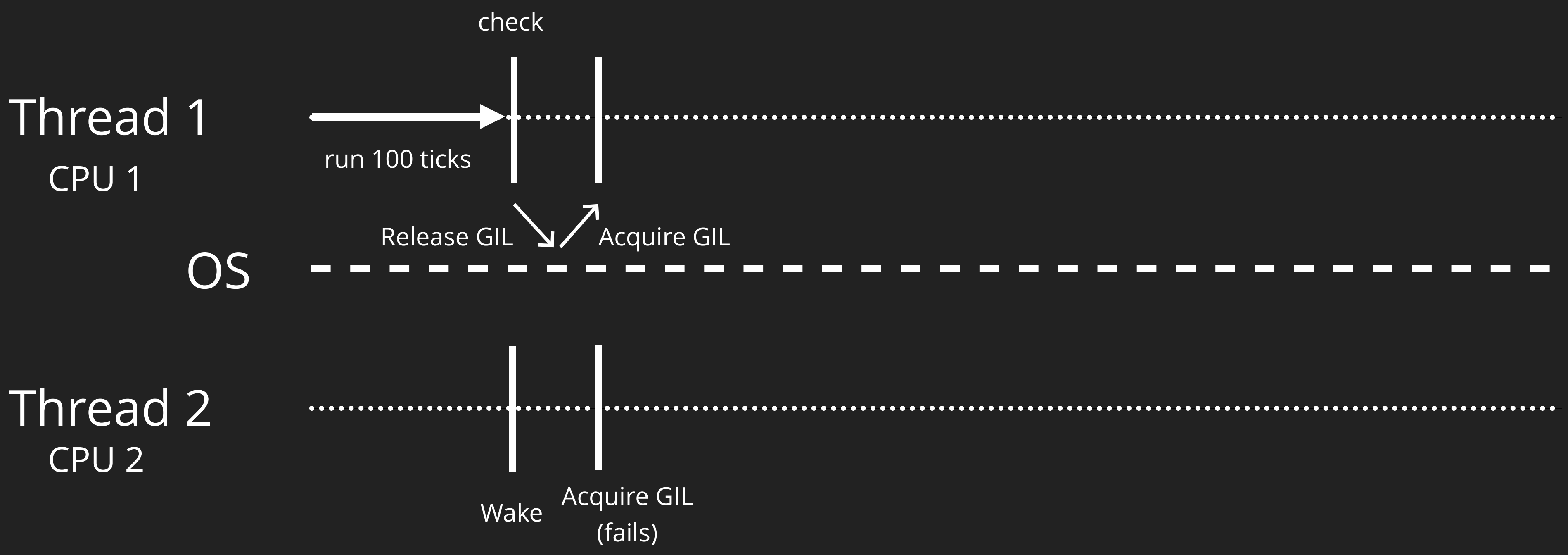
Thread Thrashing



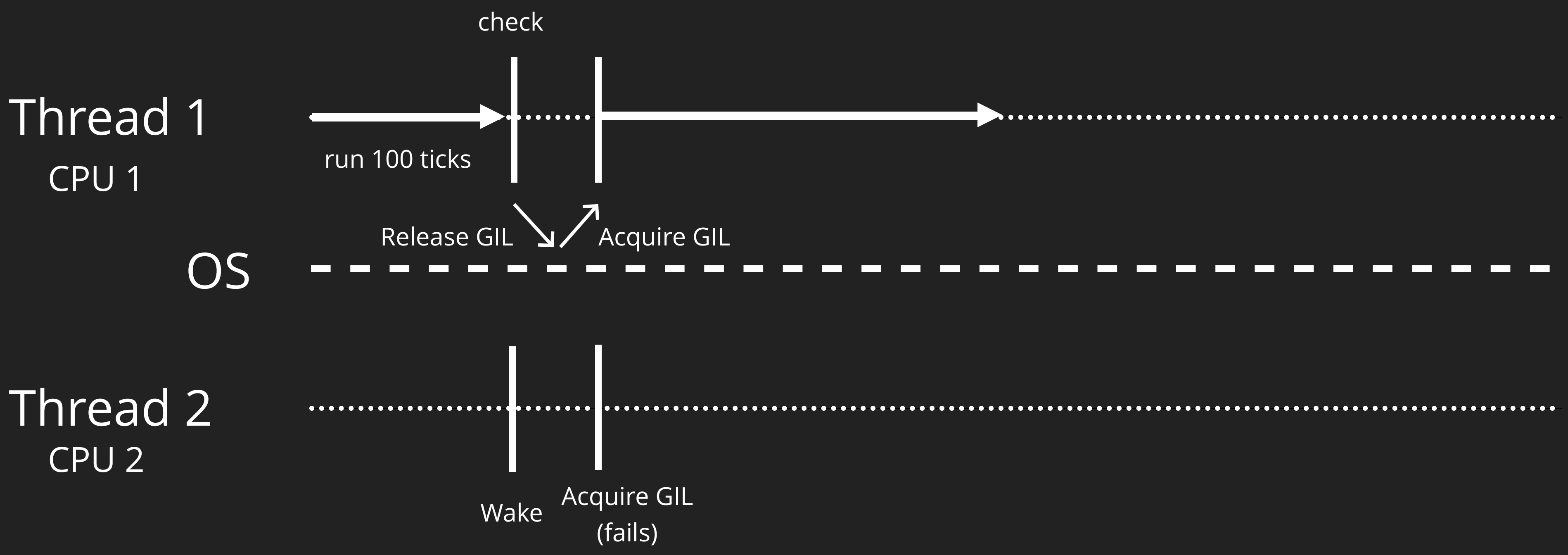
Thread Thrashing



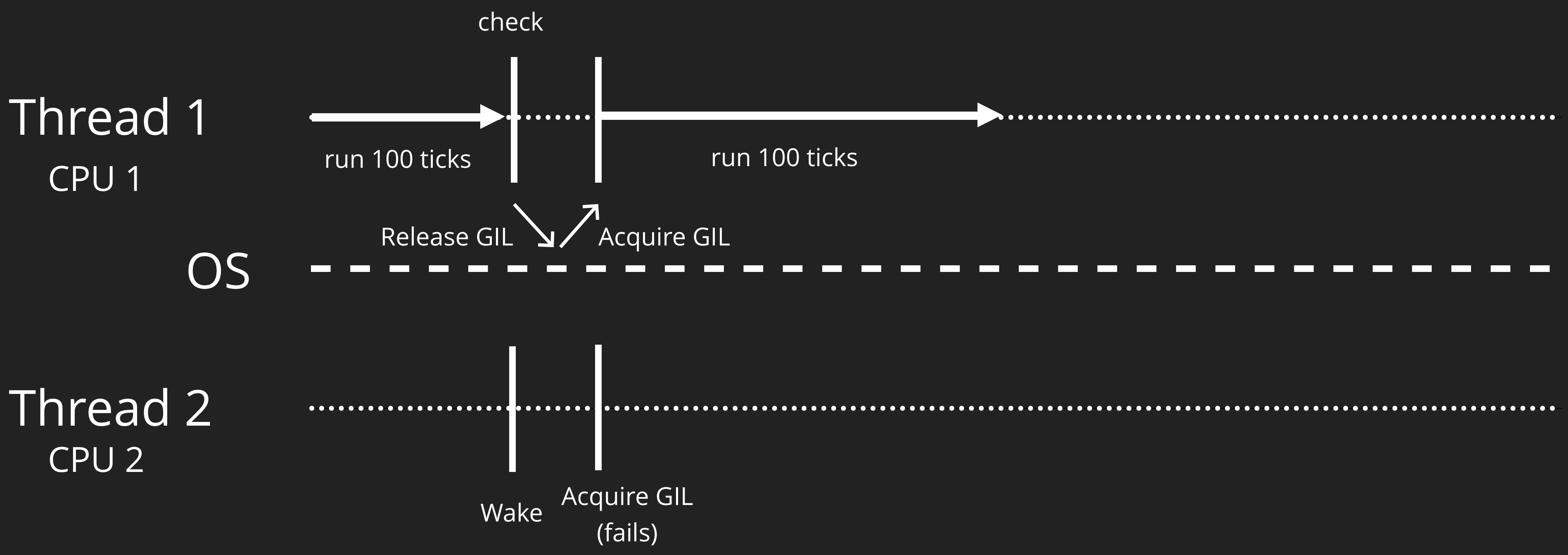
Thread Thrashing



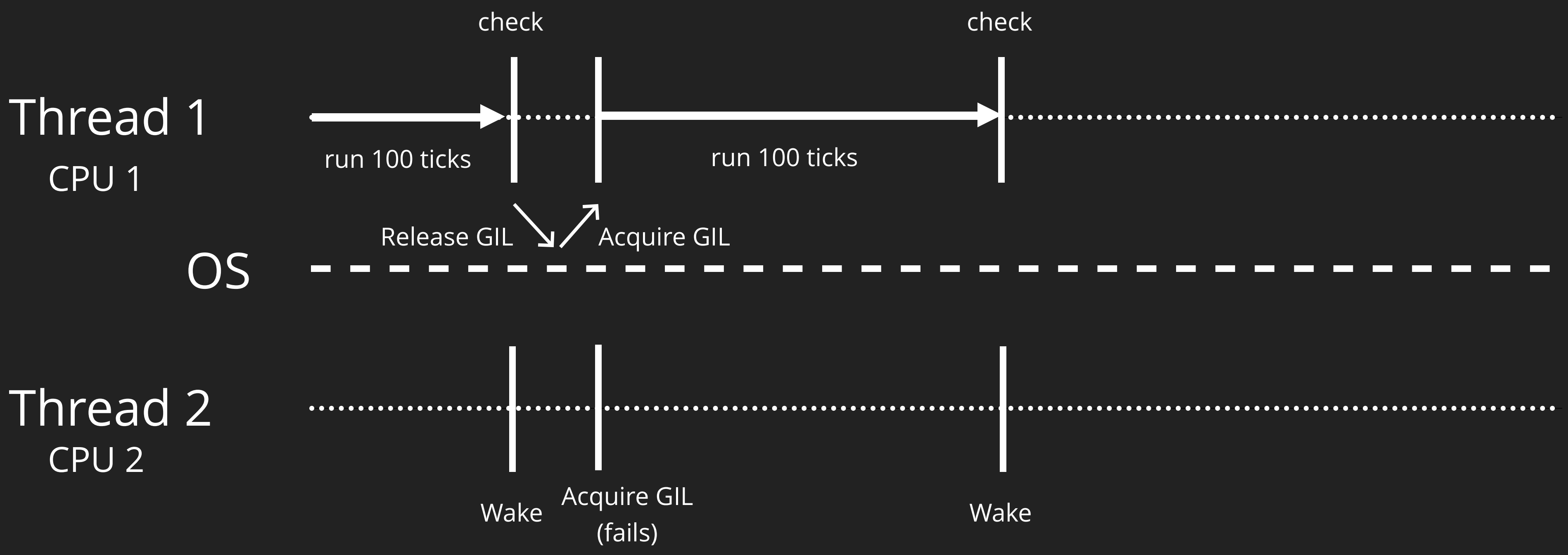
Thread Thrashing



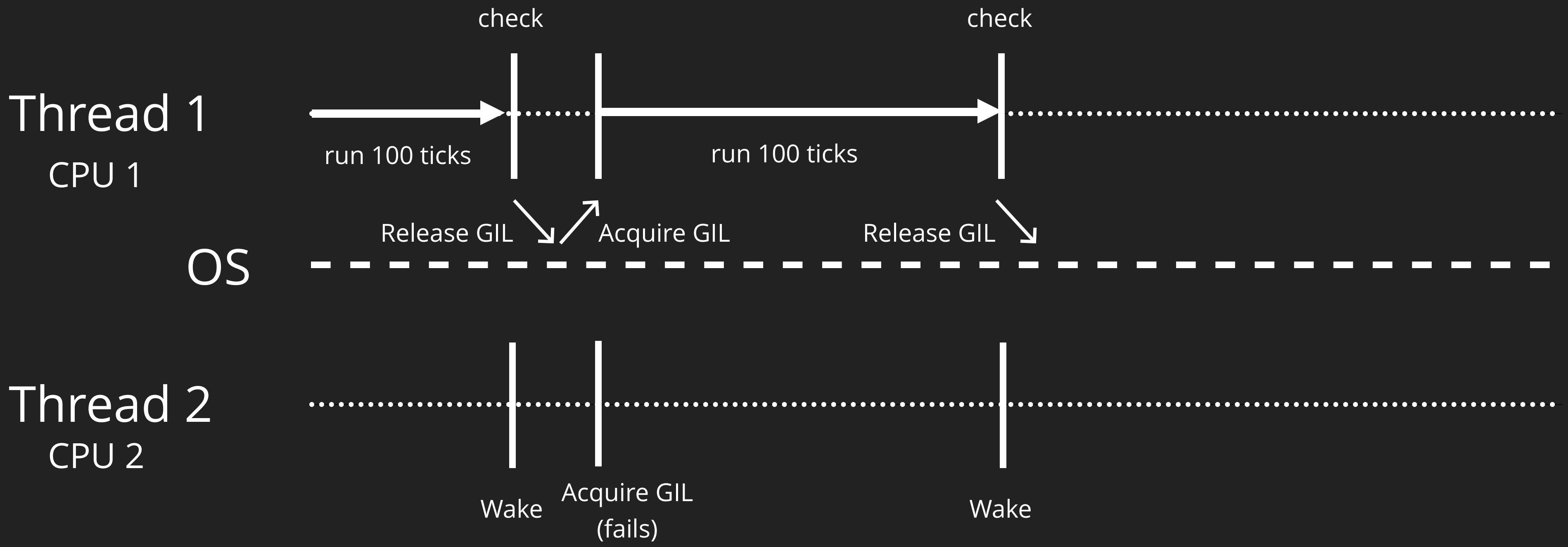
Thread Thrashing



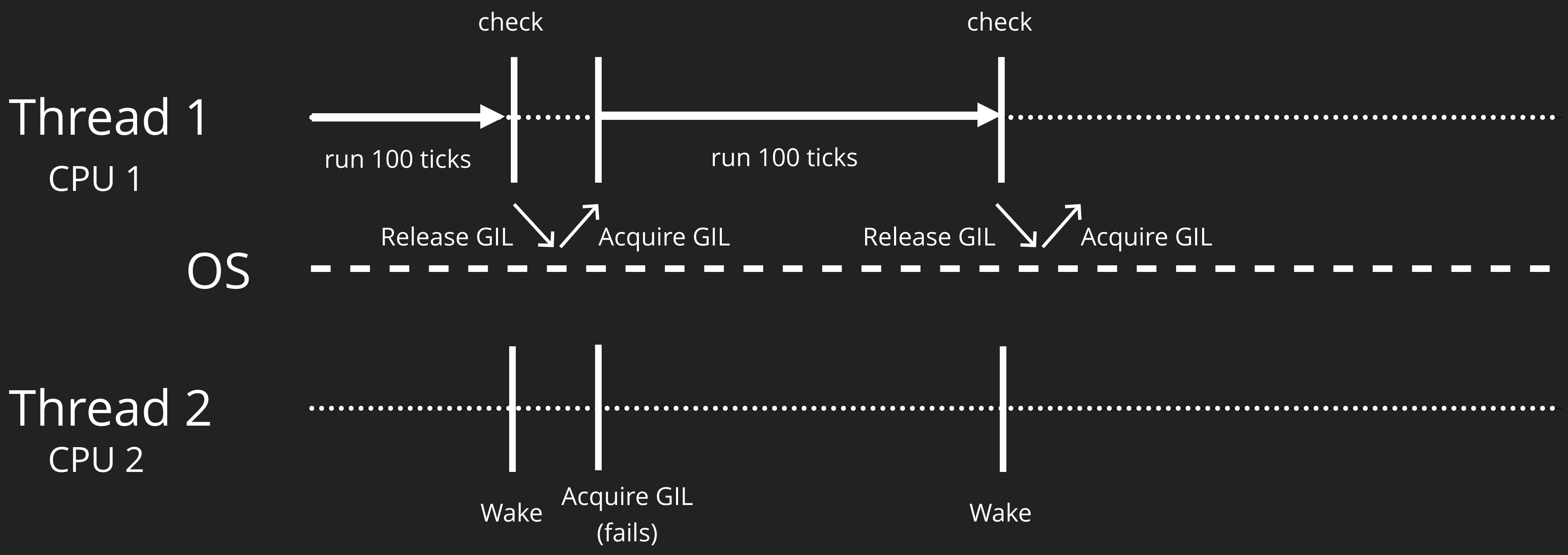
Thread Thrashing



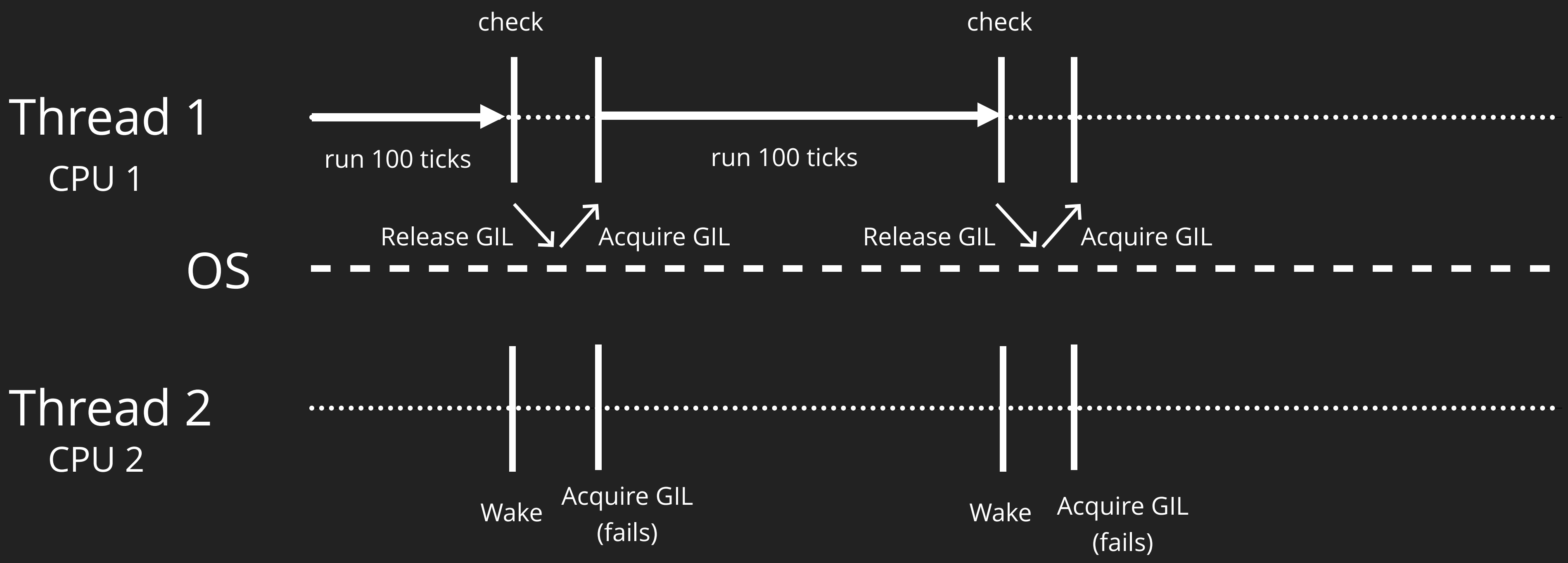
Thread Thrashing



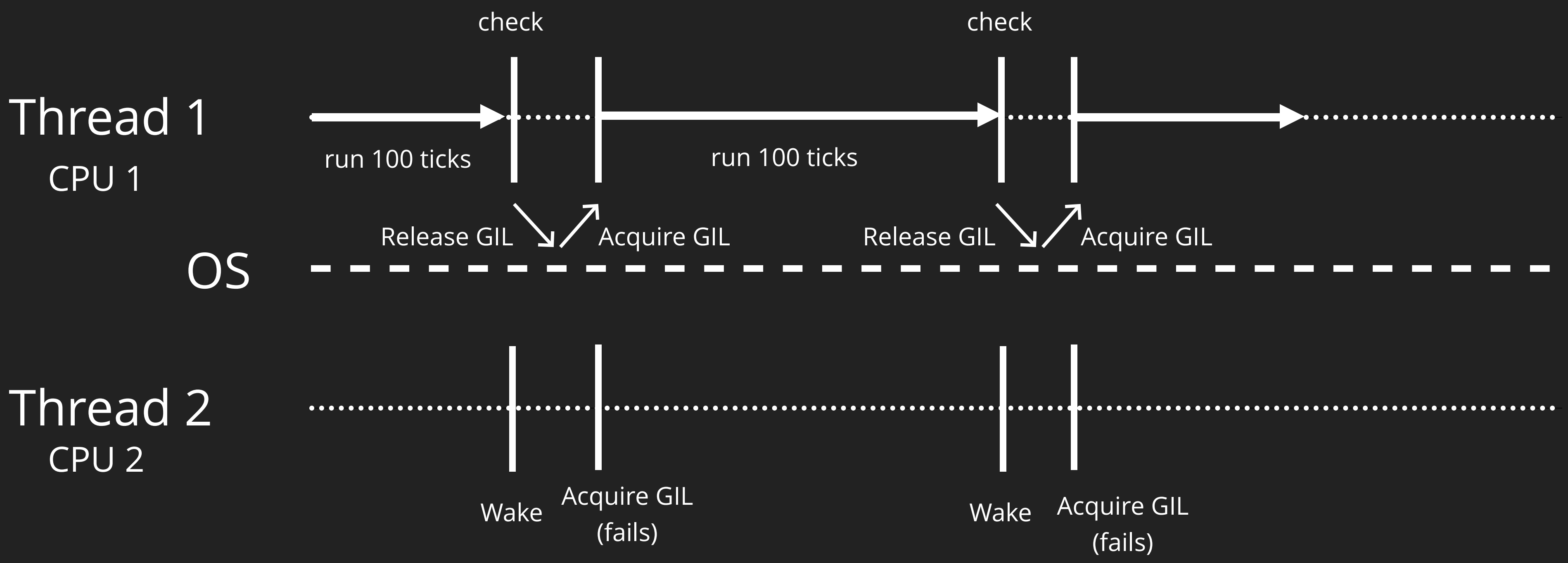
Thread Thrashing



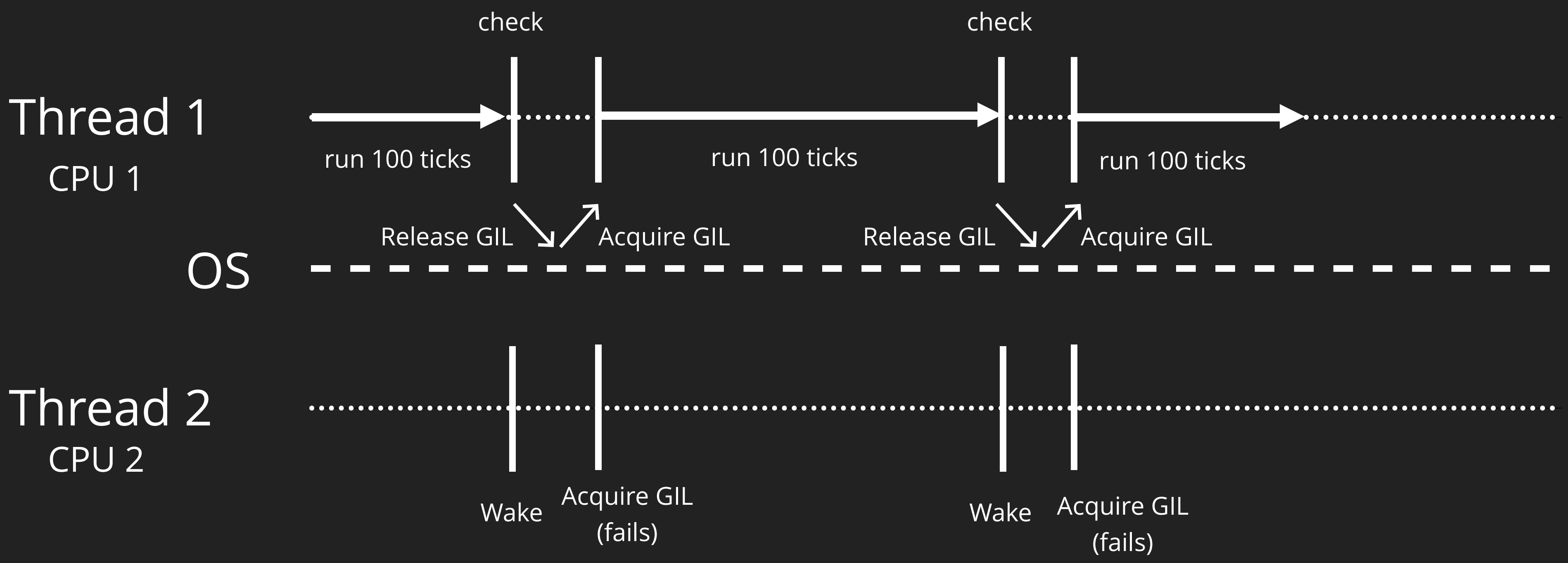
Thread Thrashing



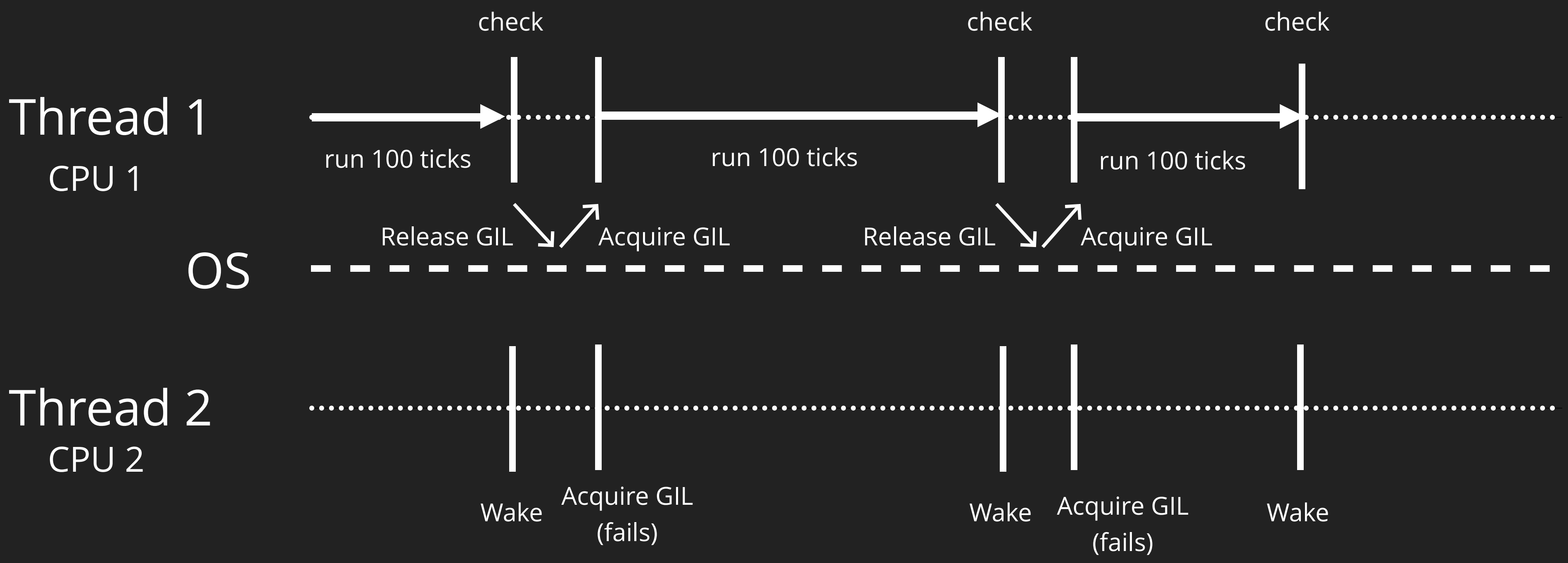
Thread Thrashing



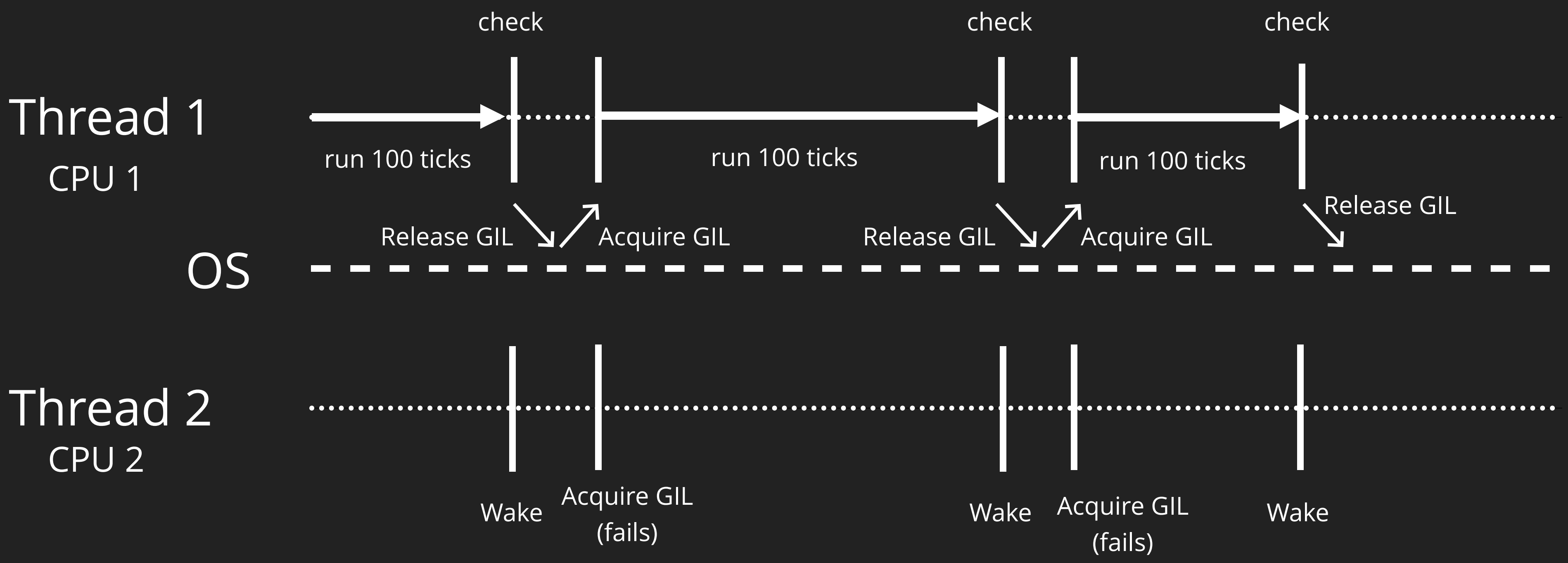
Thread Thrashing



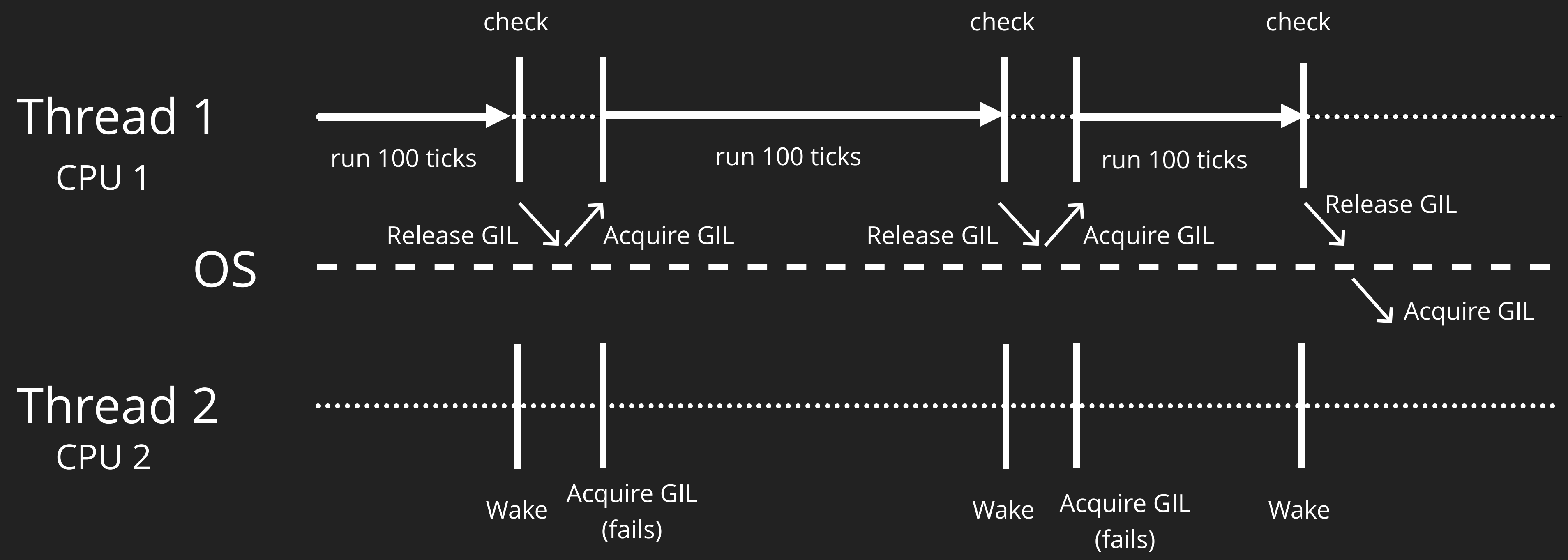
Thread Thrashing



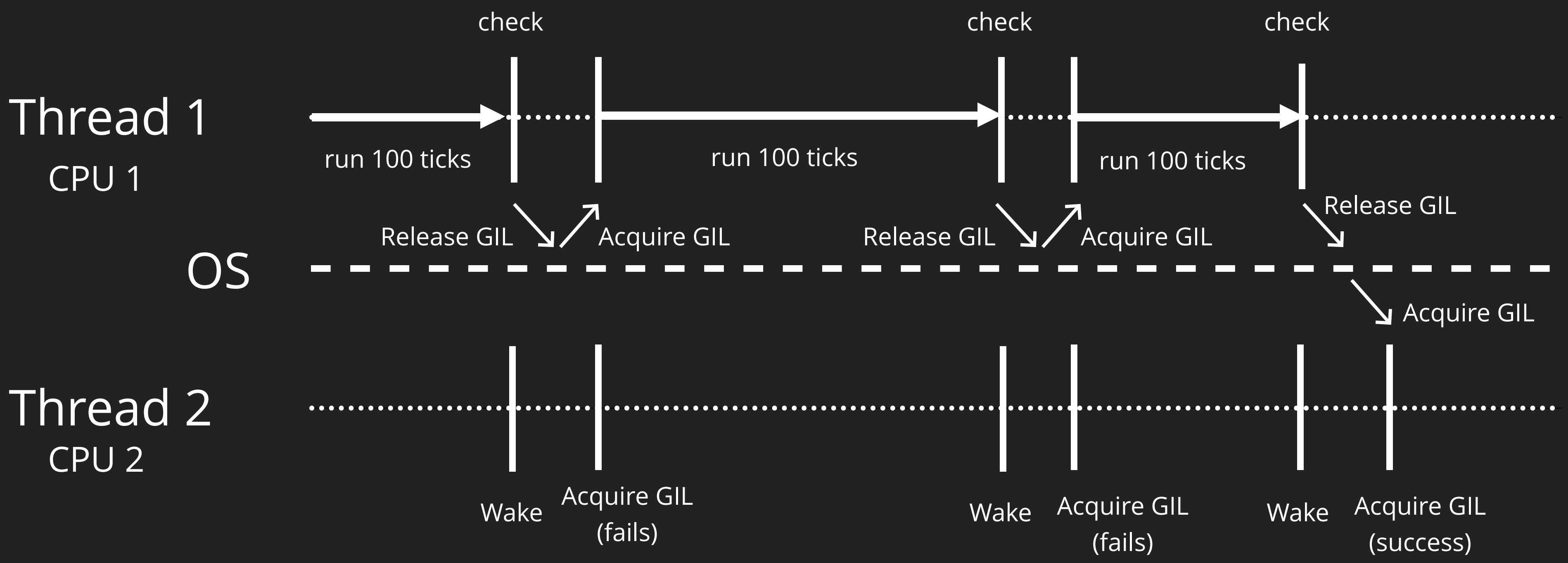
Thread Thrashing



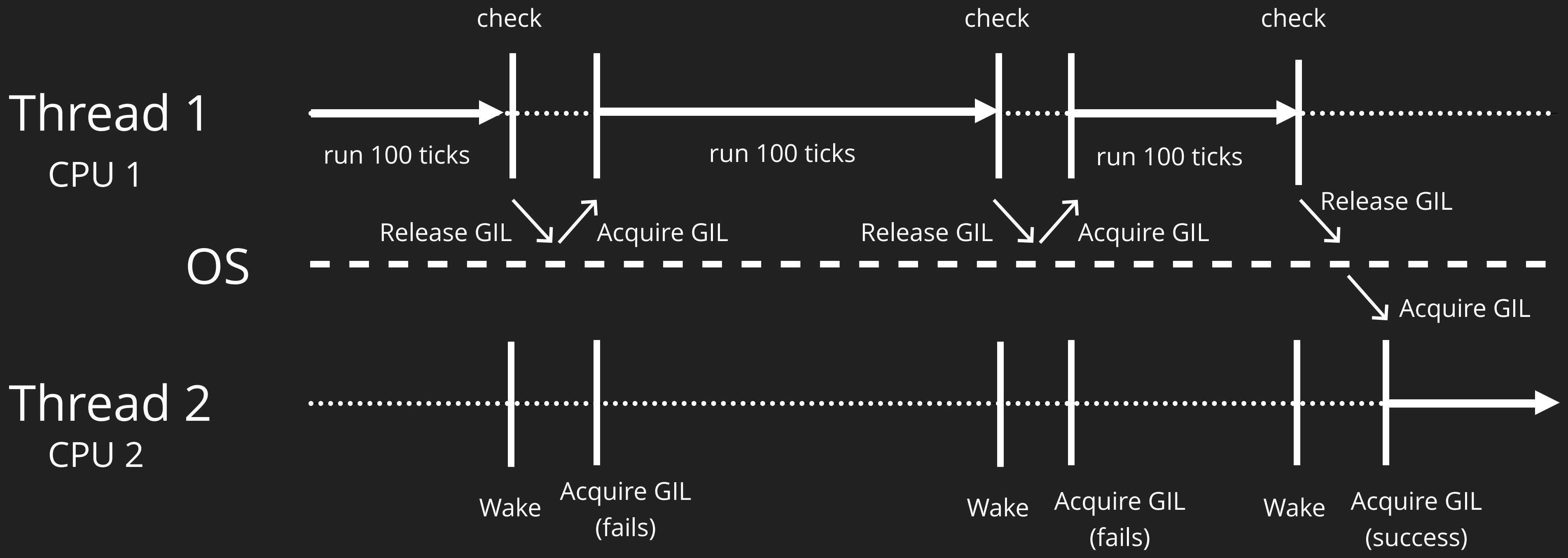
Thread Thrashing



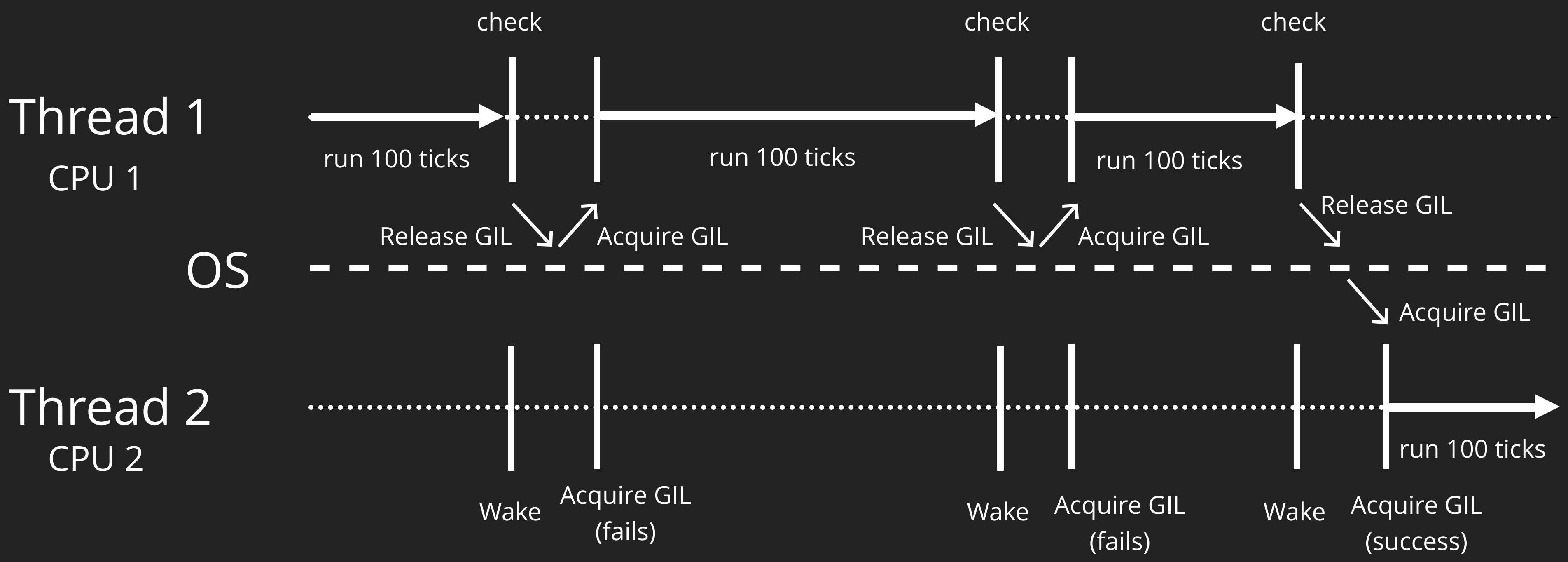
Thread Thrashing



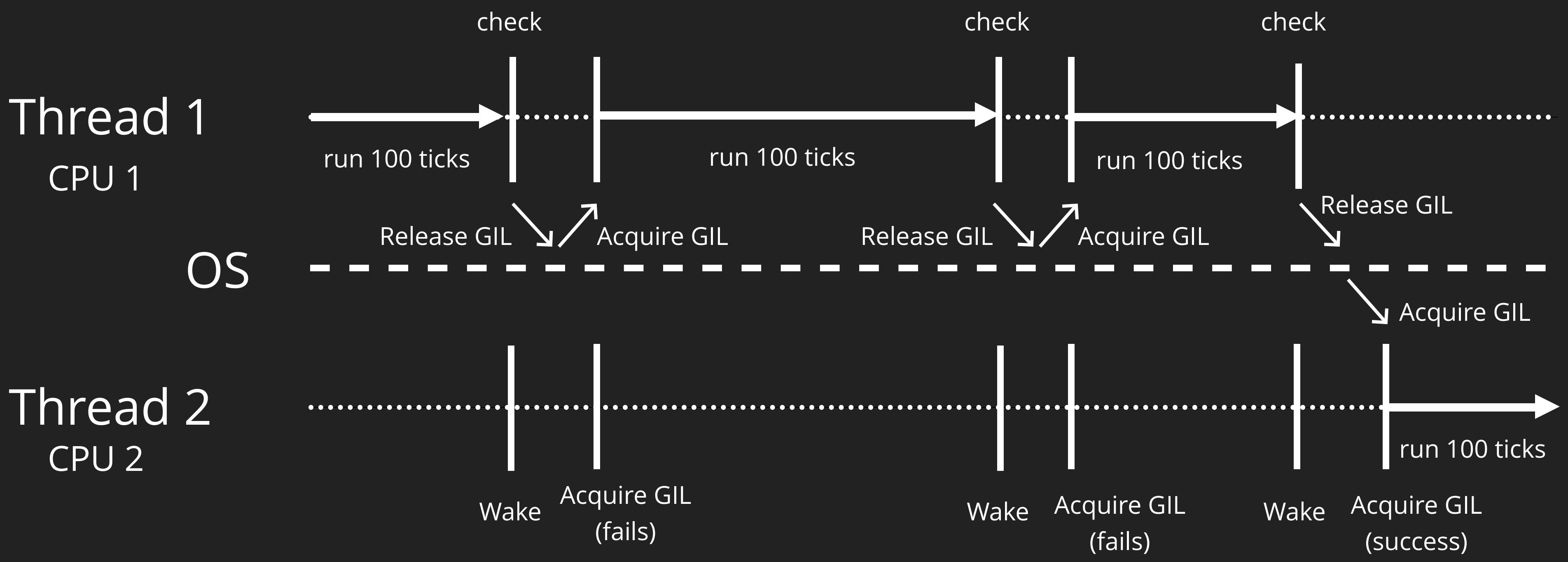
Thread Thrashing



Thread Thrashing



Thread Thrashing



- When thread 2 wakes up, the GIL is already gone

A Better GIL after Python 3.2

A Better GIL after Python 3.2

- It aims to fix thread thrashing

A Better GIL after Python 3.2

- It aims to fix thread thrashing
- Current thread will voluntarily release the GIL if it runs out of TIMEOUTs

A Better GIL after Python 3.2

- It aims to fix thread thrashing
- Current thread will voluntarily release the GIL if it runs out of TIMEOUTs
- If other thread acquire the GIL, the current thread will release the GIL after 5ms

A Better GIL after Python 3.2

- It aims to fix thread thrashing
- Current thread will voluntarily release the GIL if it runs out of TIMEOUTs
- If other thread acquire the GIL, the current thread will release the GIL after 5ms
- A thread runs until `gil_drop_request` gets set to 1

But, the GIL is still there…

Part III

Remove GIL?

Why GIL?

Why GIL?

- There was no multi-core computer when Python was created

Why GIL?

- There was no multi-core computer when Python was created
- Python is designed to be easy-to-use, so you don't need to care about the memory stuff

Why GIL?

- There was no multi-core computer when Python was created
- Python is designed to be easy-to-use, so you don't need to care about the memory stuff
- GIL prevents deadlocks (as there is only one lock)

Why GIL?

- There was no multi-core computer when Python was created
- Python is designed to be easy-to-use, so you don't need to care about the memory stuff
- GIL prevents deadlocks (as there is only one lock)
- GIL provides a performance increase to single-threaded programs as only one lock needs to be managed

Why GIL?

- There was no multi-core computer when Python was created
- Python is designed to be easy-to-use, so you don't need to care about the memory stuff
- GIL prevents deadlocks (as there is only one lock)
- GIL provides a performance increase to single-threaded programs as only one lock needs to be managed
- CPython uses Reference Counting

Before Removing the GIL…

Before Removing the GIL…

1. Reference Counting

Before Removing the GIL…

1. Reference Counting
2. Globals and statics in the interpreter

Before Removing the GIL…

1. Reference Counting
2. Globals and statics in the interpreter
3. The C extension parallelism and reentrancy issues need to be handled as do places in the code where atomicity is required

Before Removing the GIL...

1. Reference Counting
2. Globals and statics in the interpreter
3. The C extension parallelism and reentrancy issues need to be handled as do places in the code where atomicity is required
4. You can't breaking all of the C extensions

Before Removing the GIL...

1. Reference Counting
2. Globals and statics in the interpreter
3. The C extension parallelism and reentrancy issues need to be handled as do places in the code where atomicity is required
4. You can't breaking all of the C extensions

> *“ I'd welcome a set of patches into Py3k only if the performance for a single-threaded program (and for a multi-threaded but I/O-bound program) does not decrease.*
> *-- Guido van Rossum*

Related Works

Related Works

- 1995, Greg Stein, a fork of Python 1.5

Related Works

- 1995, Greg Stein, a fork of Python 1.5
- Larry Hastings' Gilectomy (on hold)

Related Works

- 1995, Greg Stein, a fork of Python 1.5
- Larry Hastings' Gilectomy (on hold)
- Many other implementations...

It's Hard!!!

But, Who Cares?

But, Who Cares?

- Users with threaded, CPU bound Python code

But, Who Cares?

- Users with threaded, CPU bound Python code

But, Who Cares?

- Users with threaded, CPU bound Python code

Use C extensions!

But, Who Cares?

- Users with threaded, CPU bound Python code
- Basically no one else

Use C extensions!

Other Solutions

- Multi-processing (not recommend)
- C extension modules
  - Rewrite CPU bound code in C (you need to control memory by yourself)
  - Release the GIL around that code
- Corountine

Part IV

The Future

PEP 554

PEP 554

- Multiple Interpreters in the Stdlib

PEP 554

- Multiple Interpreters in the Stdlib
- By Eric Snow (to GIL or not to GIL: the Future of Multi-Core CPython)

PEP 554

- Multiple Interpreters in the Stdlib
- By Eric Snow (to GIL or not to GIL: the Future of Multi-Core CPython)
- Will release in CPython 3.9 (maybe)

PEP 554

- Multiple Interpreters in the Stdlib
- By Eric Snow (to GIL or not to GIL: the Future of Multi-Core CPython)
- Will release in CPython 3.9 (maybe)
- CPython has supported multiple interpreters in the same process since version 1.5 (1997) via the C-API

PEP 554

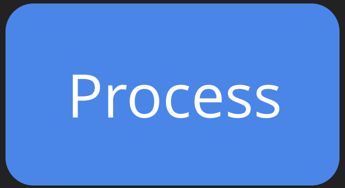
- Multiple Interpreters in the Stdlib
- By Eric Snow (to GIL or not to GIL: the Future of Multi-Core CPython)
- Will release in CPython 3.9 (maybe)
- CPython has supported multiple interpreters in the same process since version 1.5 (1997) via the C-API
- Introduce the stdlib interpreters modules, high-level interface to subinterpreters

PEP 554

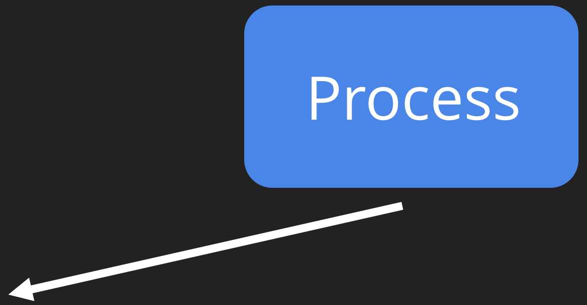
- Multiple Interpreters in the Stdlib
- By Eric Snow (to GIL or not to GIL: the Future of Multi-Core CPython)
- Will release in CPython 3.9 (maybe)
- CPython has supported multiple interpreters in the same process since version 1.5 (1997) via the C-API
- Introduce the stdlib interpreters modules, high-level interface to subinterpreters
- Functionality for sharing data between interpreters (channel)

Subinterpreters Model

Subinterpreters Model

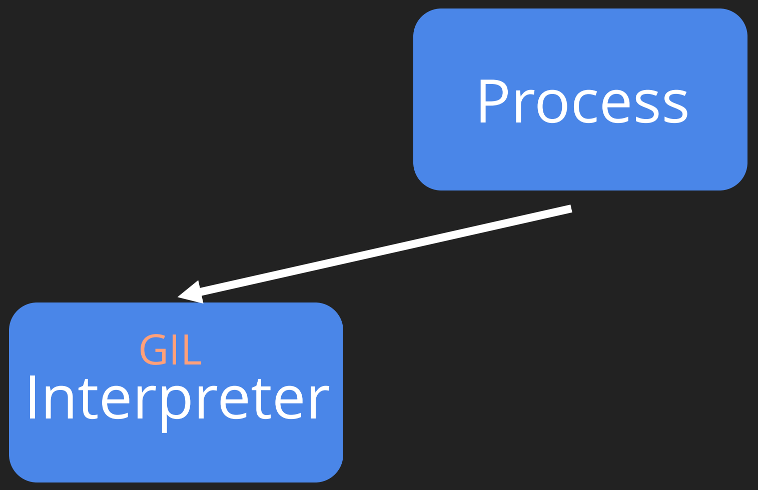


Subinterpreters Model

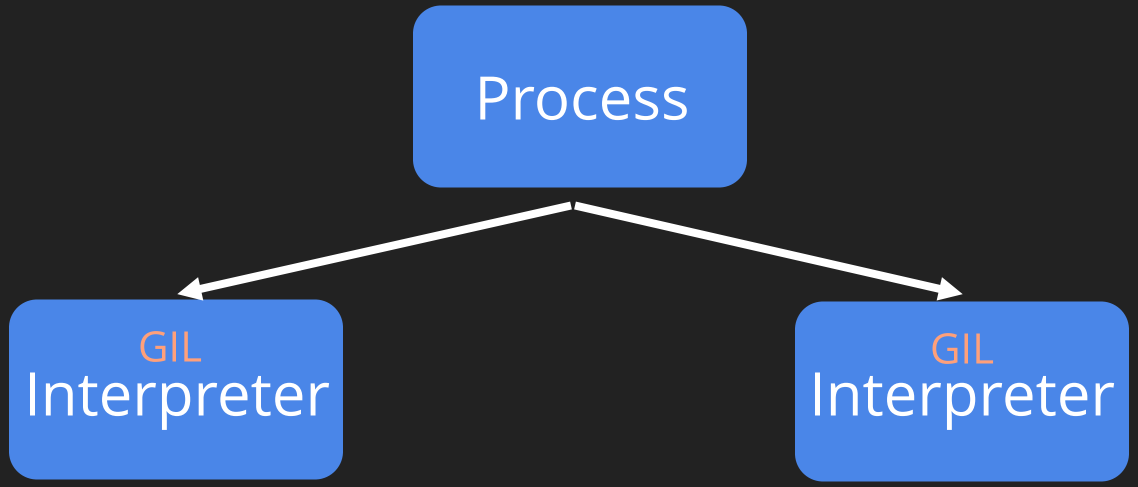


Process

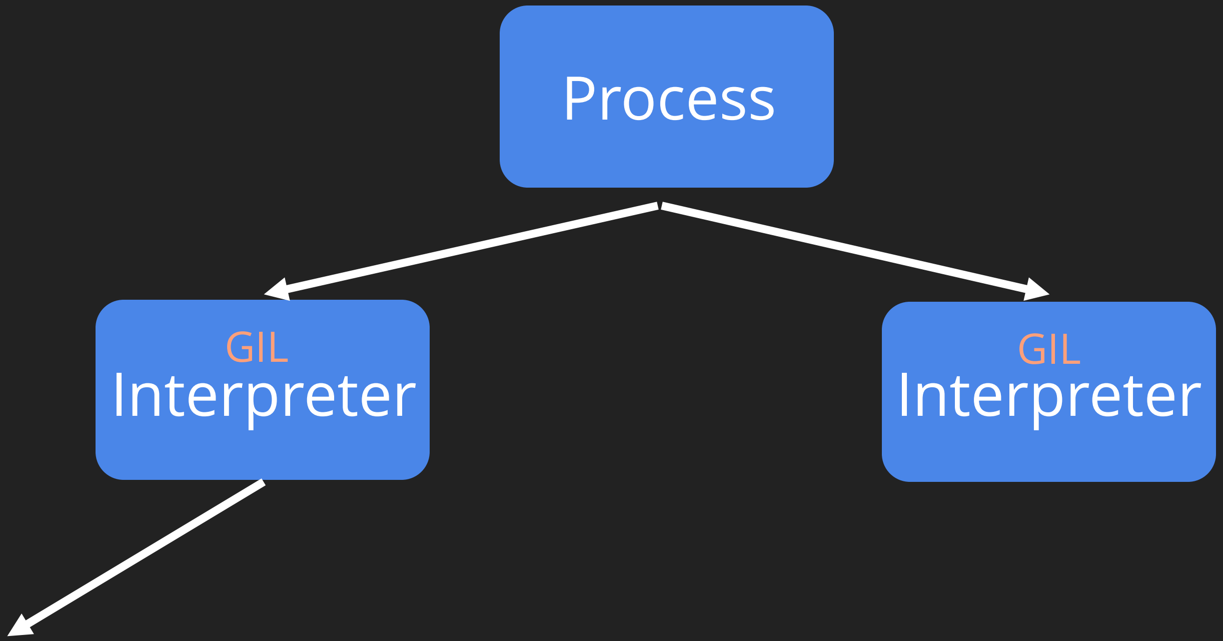
Subinterpreters Model



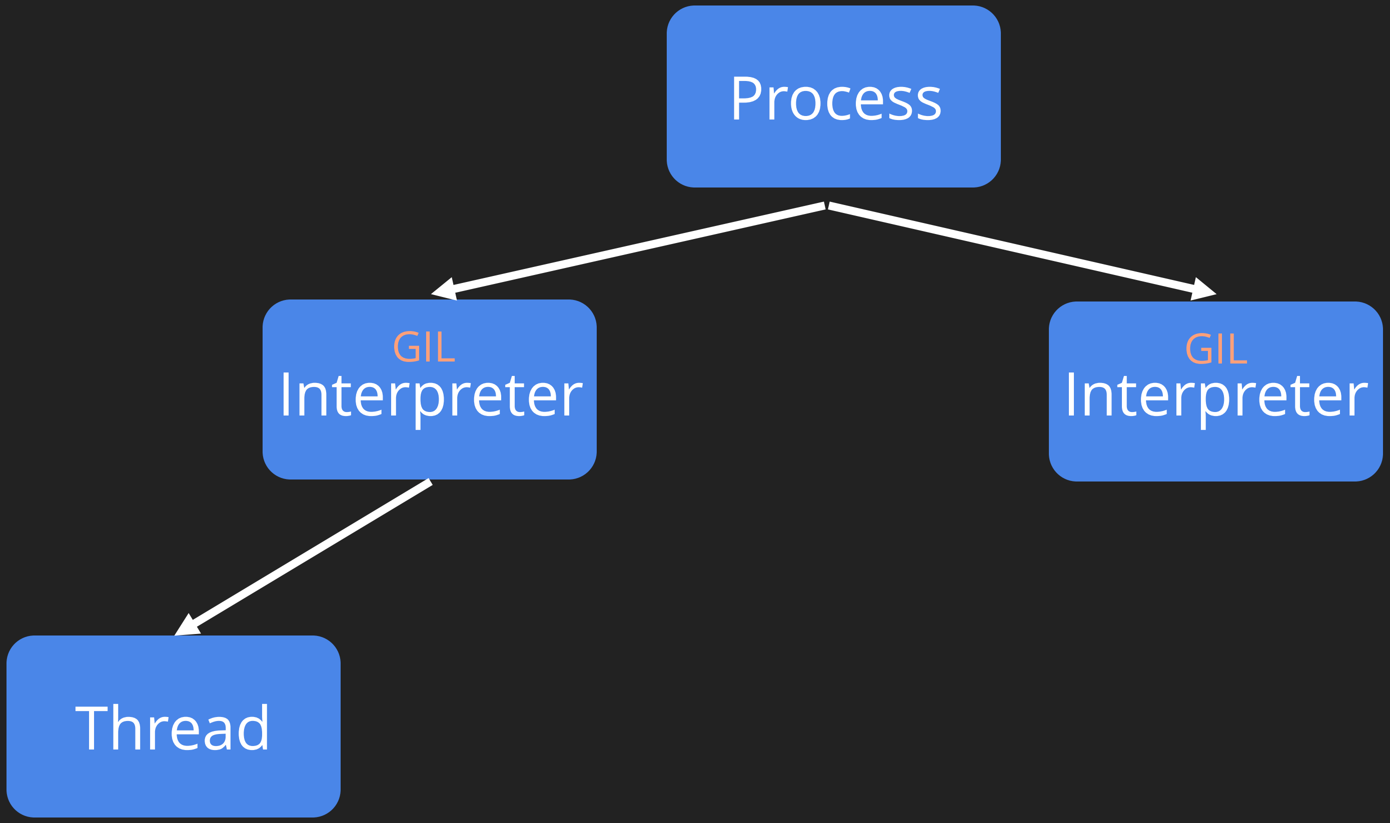
Subinterpreters Model



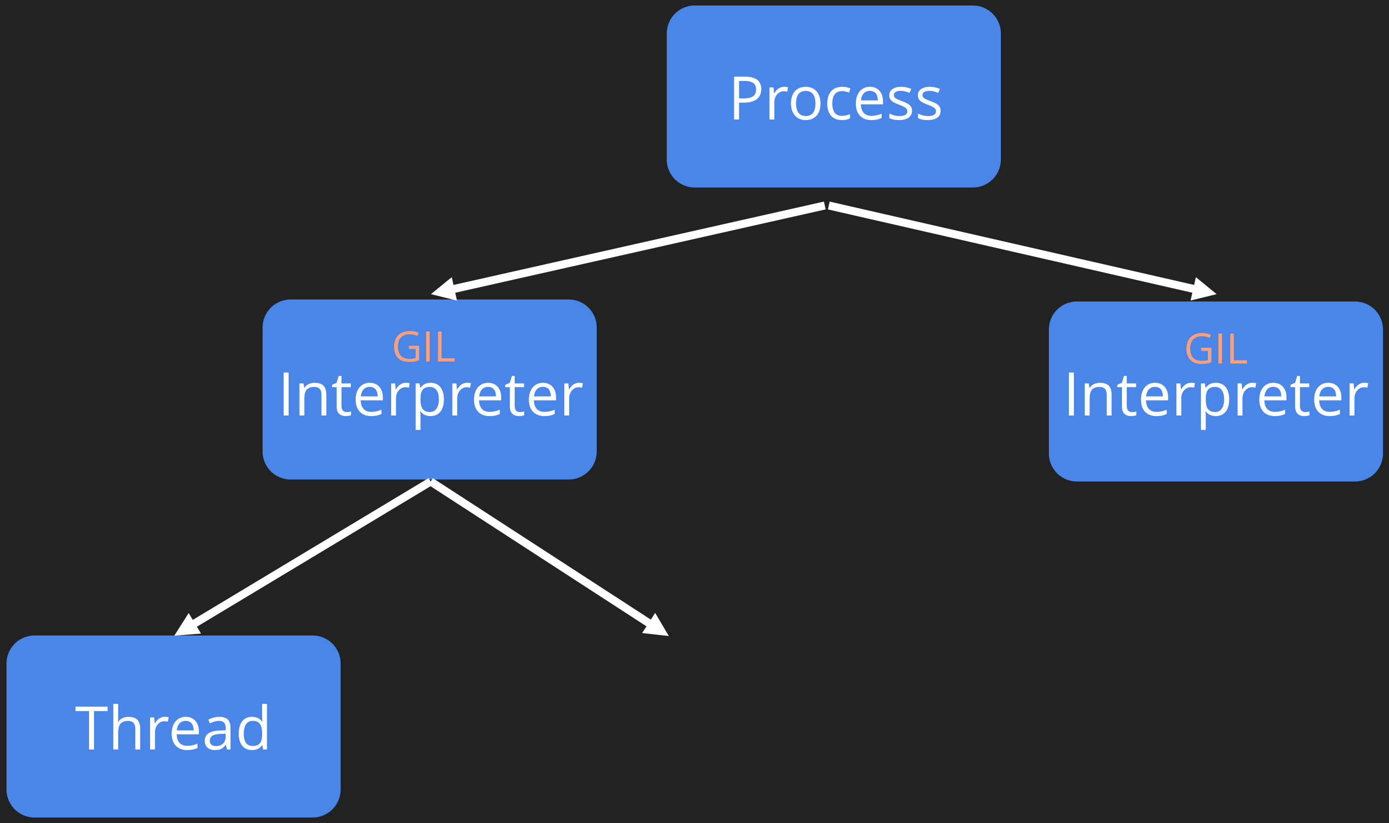
Subinterpreters Model



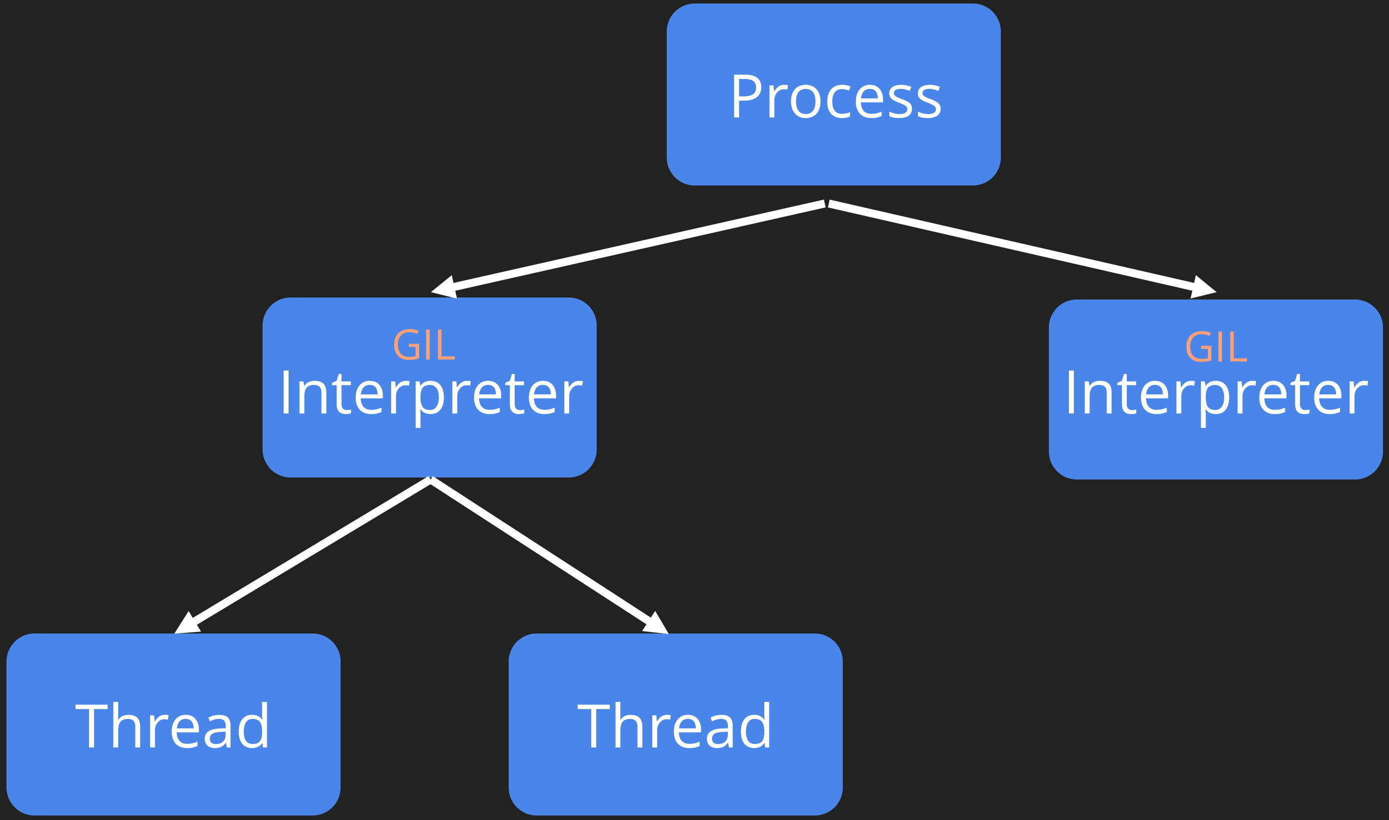
Subinterpreters Model



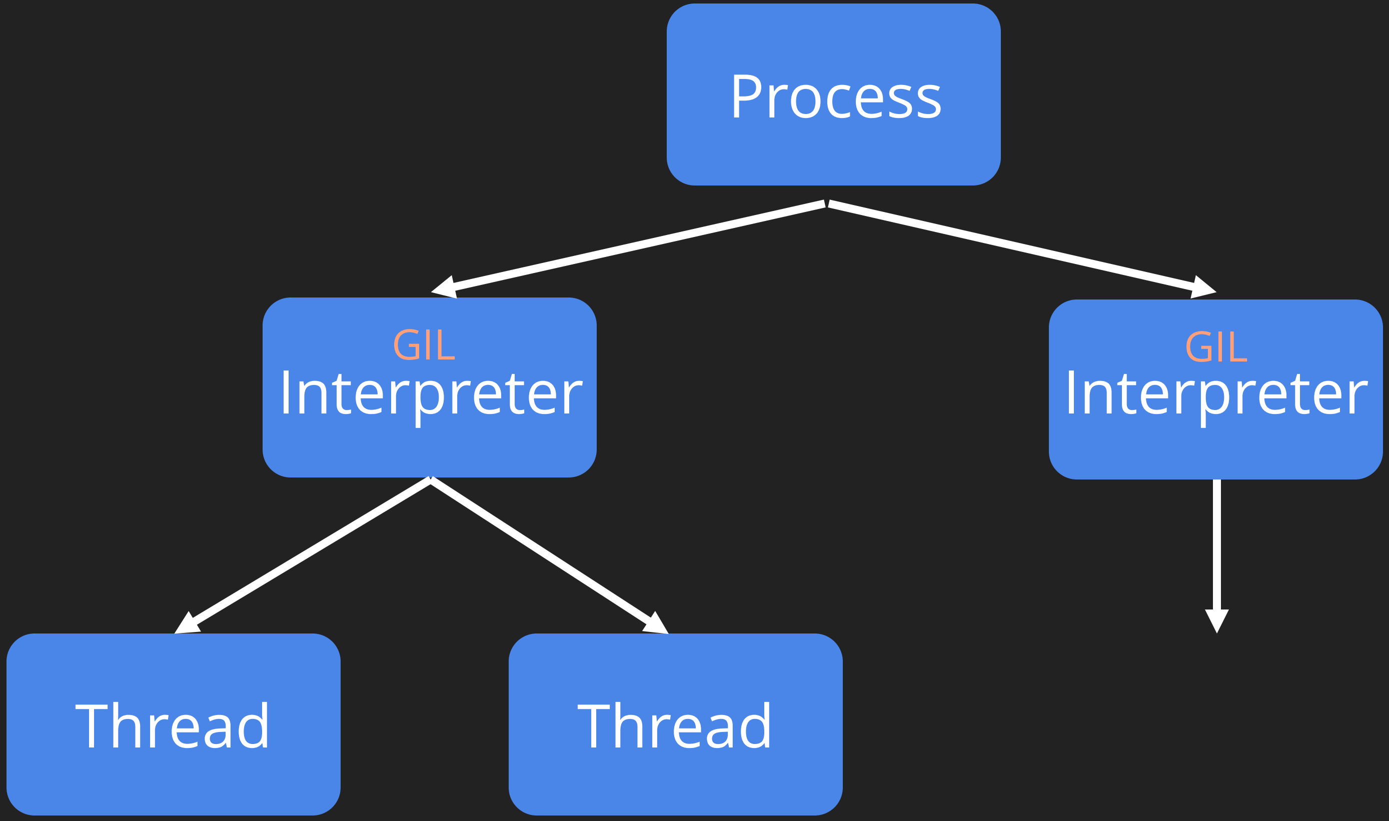
Subinterpreters Model



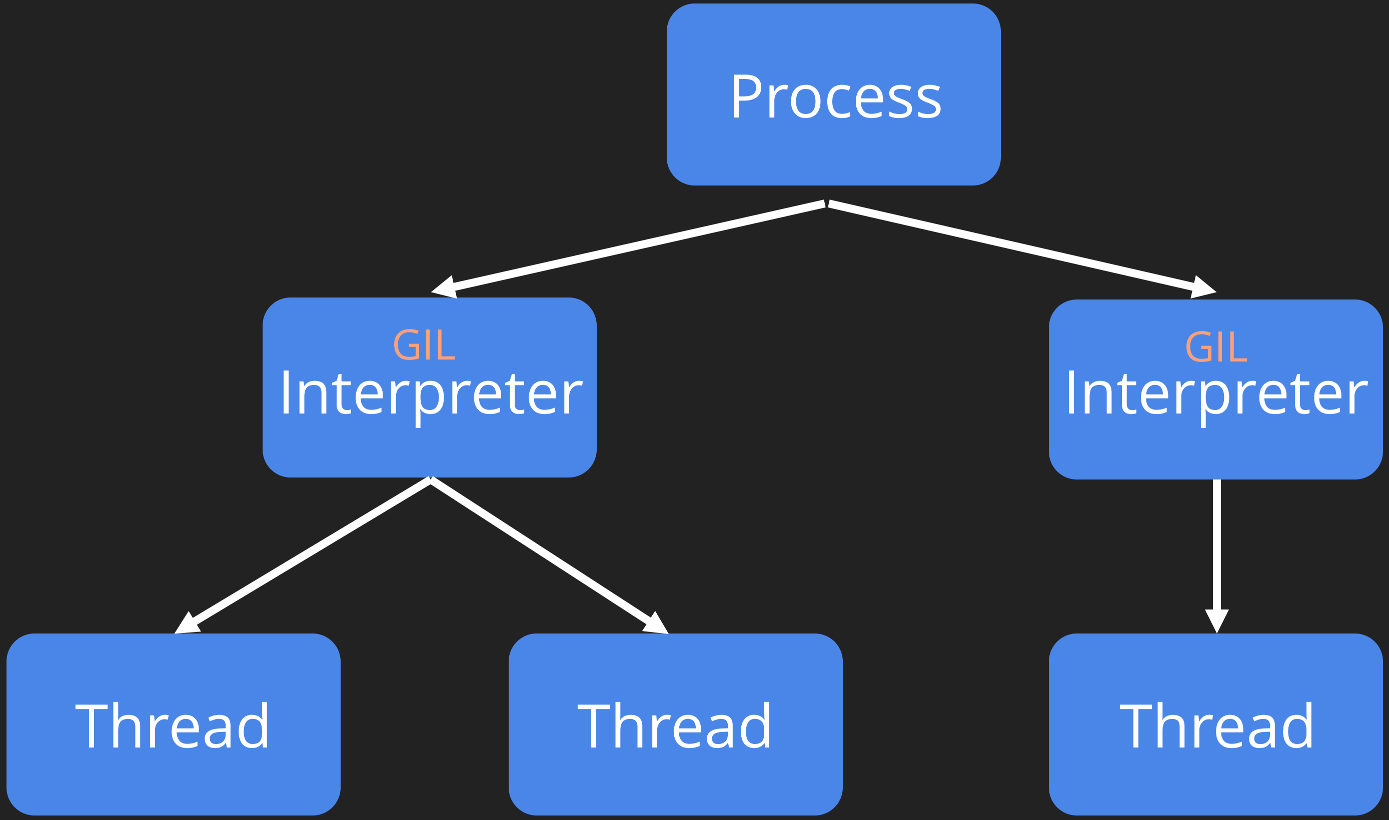
Subinterpreters Model



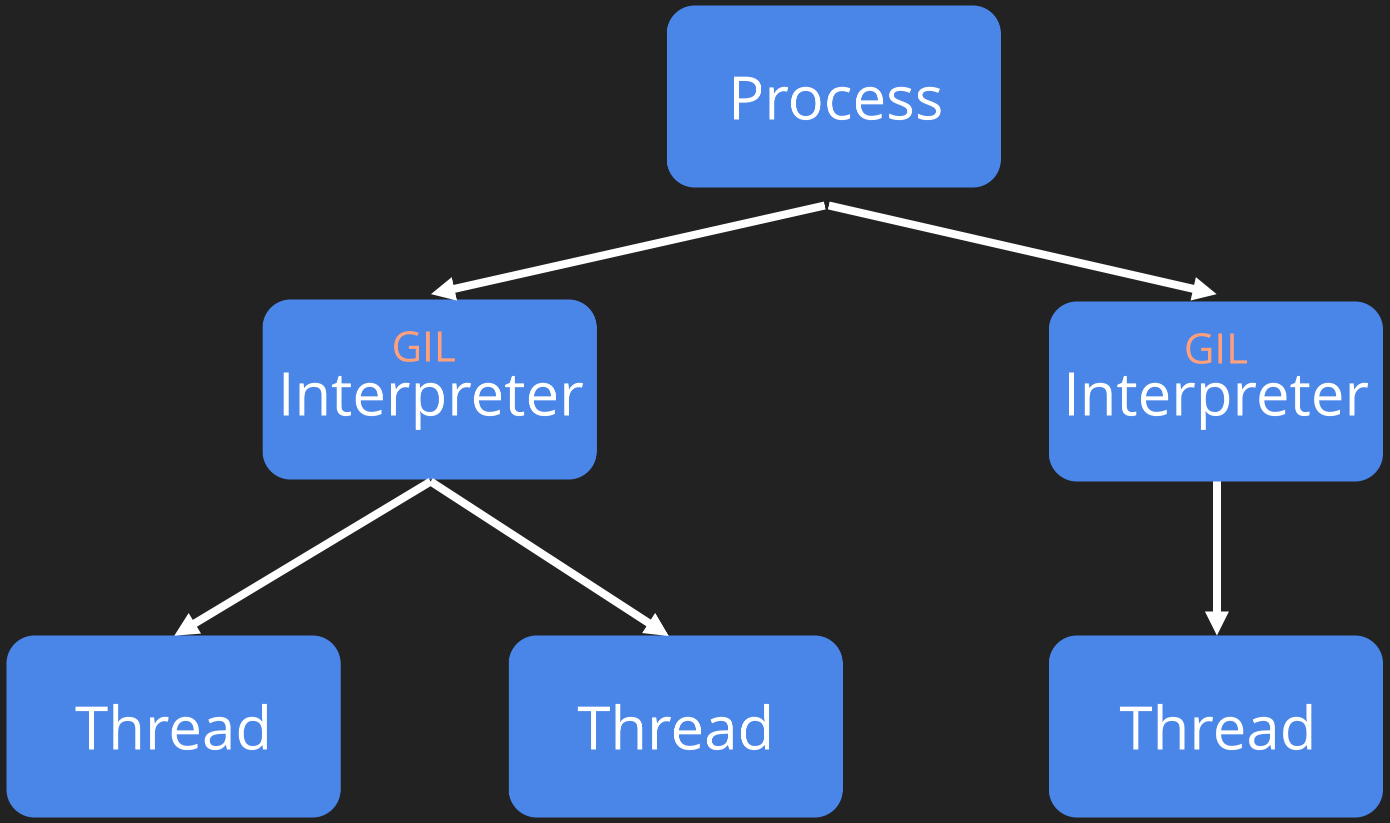
Subinterpreters Model



Subinterpreters Model



Subinterpreters Model



- Sub-interpreter can't access other sub-interpreters' variables

How to Sharing Data?

How to Sharing Data?

- A mechanism centers around "channels"

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:
    - None

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:
    - None
    - bytes

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:
  - None
  - bytes
  - str

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:
    - None
    - bytes
    - str
    - int

How to Sharing Data?

- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:
  - None
  - bytes
  - str
  - int
  - PEP 3118 buffer objects

How to Sharing Data?

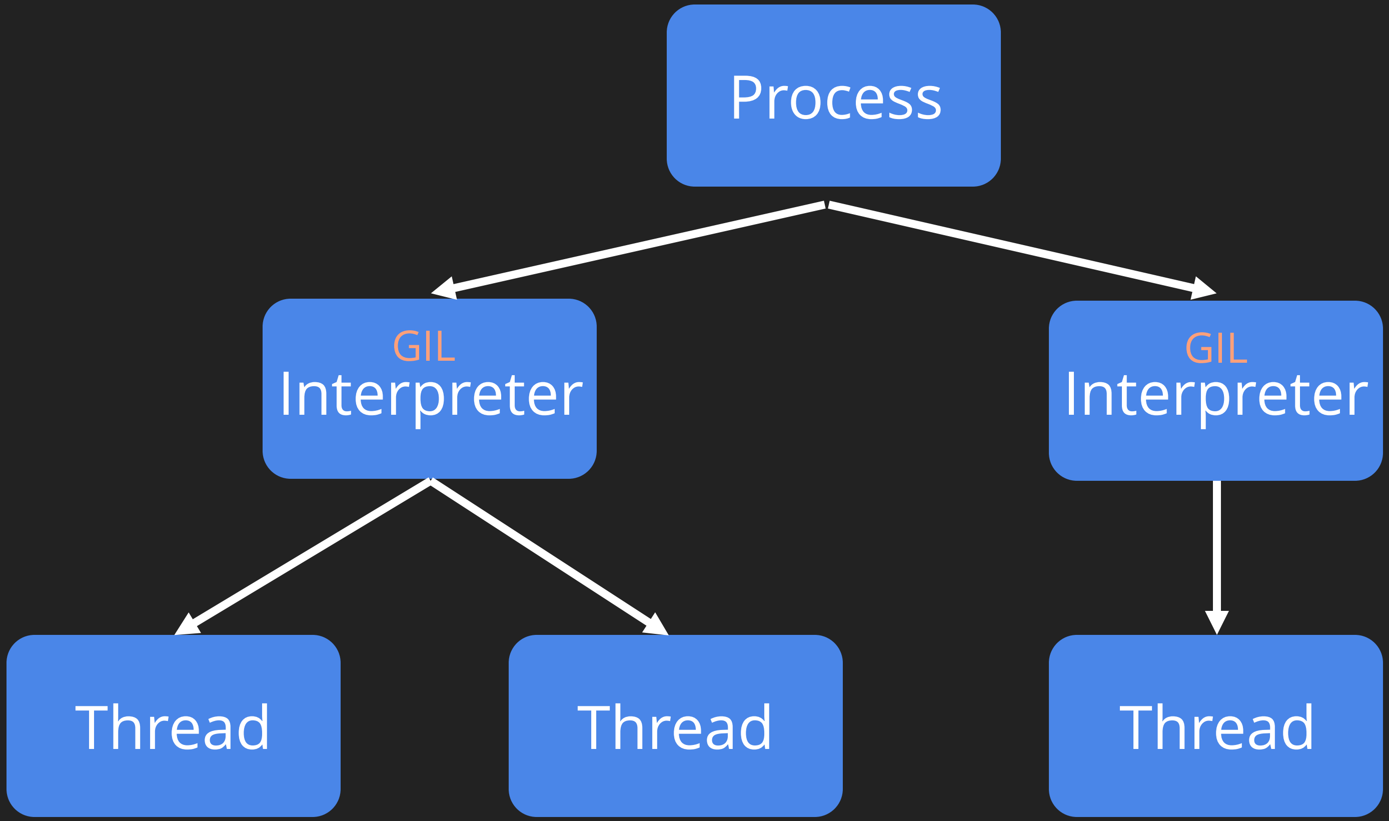
- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:
  - None
  - bytes
  - str
  - int
  - PEP 3118 buffer objects
  - PEP 554 channels

How to Sharing Data?

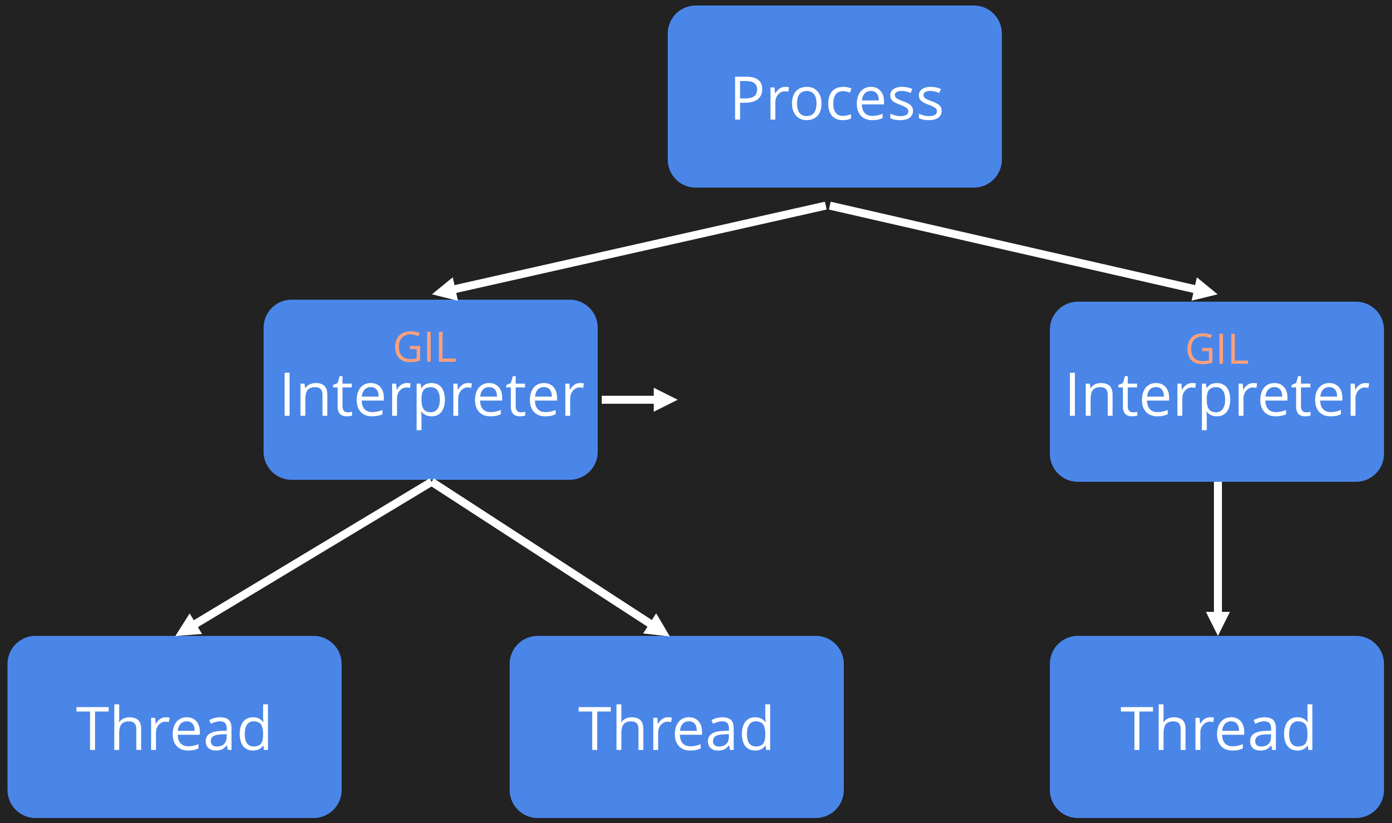
- A mechanism centers around "channels"
- Similiar to queues and pipes
- Objects are not shared between iterpreters since they are tied to the interpreter in which they were created
- Only the following types will be supported fo sharing:
  - None
  - bytes
  - str
  - int
  - PEP 3118 buffer objects
  - PEP 554 channels

Support for other basic types (e.g. bool, float) will be added later

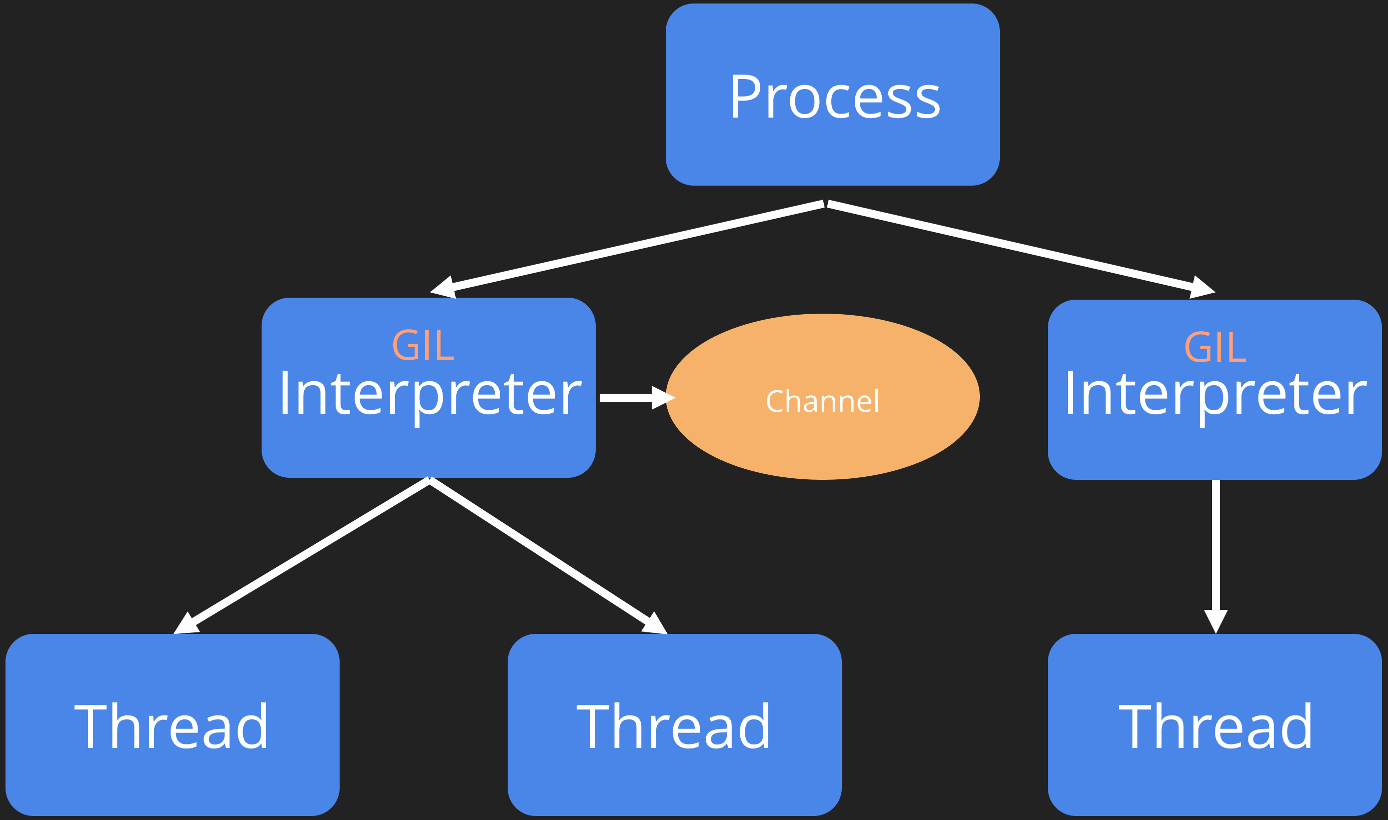
How to Sharing Data



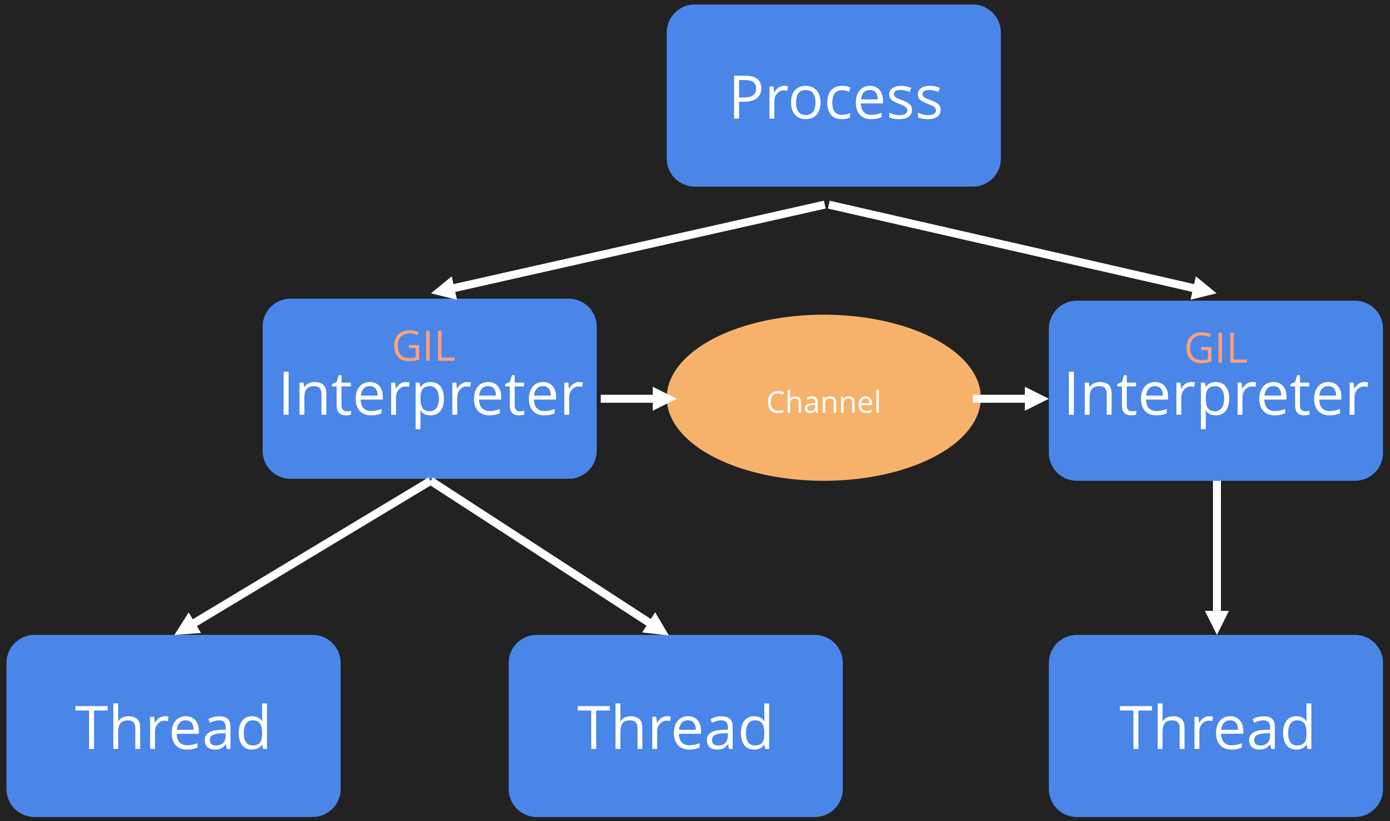
How to Sharing Data



How to Sharing Data



How to Sharing Data



Advantages

Advantages

Advantages

- Isolation

Advantages

- Isolation
  - Each interpreter has its own copy of all modules, classes, functions, and variables

Advantages

- Isolation
  - Each interpreter has its own copy of all modules, classes, functions, and variables
  - But process-global state remains shared (file descriptors, builtin types...)

Advantages

- Isolation
  - Each interpreter has its own copy of all modules, classes, functions, and variables
  - But process-global state remains shared (file descriptors, builtin types...)
- Potentially performance

Advantages

- Isolation
  - Each interpreter has its own copy of all modules, classes, functions, and variables
  - But process-global state remains shared (file descriptors, builtin types...)
- Potentially performance
  - multiprocessing < subinterpreter < threads ?

Advantages

- Isolation
  - Each interpreter has its own copy of all modules, classes, functions, and variables
  - But process-global state remains shared (file descriptors, builtin types...)
- Potentially performance
  - multiprocessing < subinterpreter < threads ?
- Provide a direct route to an alternate concurrency mode

In Progress

In Progress

- https://github.com/ericsnowcurrently/multi-core-python

In Progress

- https://github.com/ericsnowcurrently/multi-core-python
- Resolve bugs

In Progress

- https://github.com/ericsnowcurrently/multi-core-python
- Resolve bugs
- Deal with C globals

In Progress

- https://github.com/ericsnowcurrently/multi-core-python
- Resolve bugs
- Deal with C globals
- Move some runtime state into the interpreter state

In Progress

- https://github.com/ericsnowcurrently/multi-core-python
- Resolve bugs
- Deal with C globals
- Move some runtime state into the interpreter state
    - Including the GIL

That's All. Thanks!!!

Contacts

- Homepage: http://jiayuanzhang.com/
- Blog: http://blog.jiayuanzhang.com/
- GitHub: https://github.com/forrestchang
- Twitter: https://twitter.com/tisoga

Wechat

（请备注公司/学校 + 姓名）