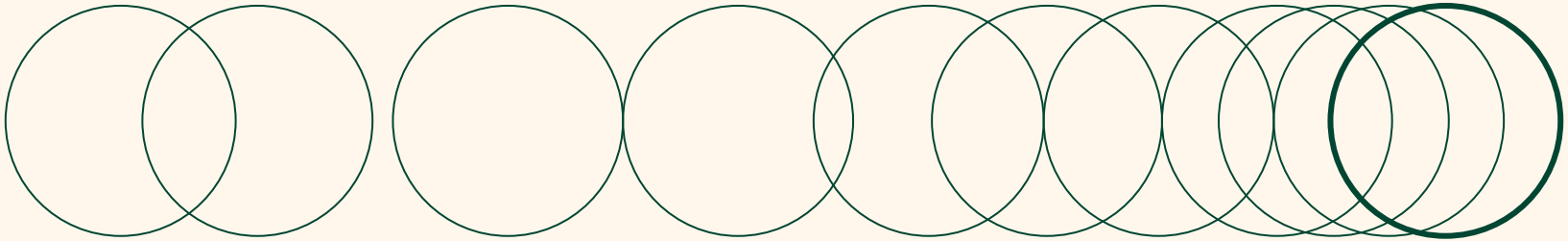
PyLadies Meetup

# Python GIL: What is it and why everyone wants to remove it?

Andressa Valadares



# ABOUT ME

Pylady since 2018

Multi-hobbies enthusiast

Game Server Engineer @ DIGIT Games

# OUTLINE

01

## BASIC CONCEPTS

What is GIL, reference counting and more

02

## MORE ABOUT GIL

Why is it so infamous?

03

## ALTERNATIVES

Not everyone is using GIL

05

## REMOVING GIL

Why it isn't that easy?

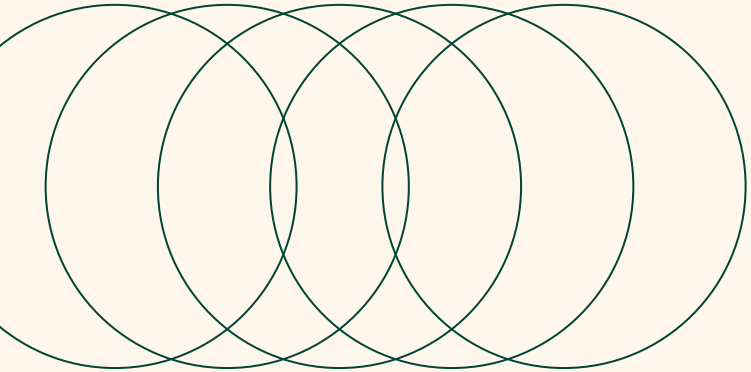
04

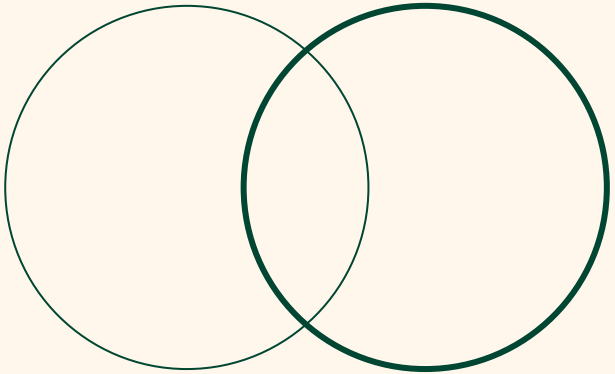
## FUTURE OF GIL

More GILs or NOGIL that is the question

06

## CONCLUSIONS

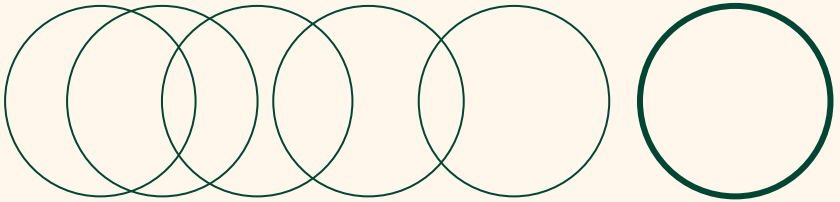




# 01 BASIC CONCEPTS

What is GIL, reference counting and more

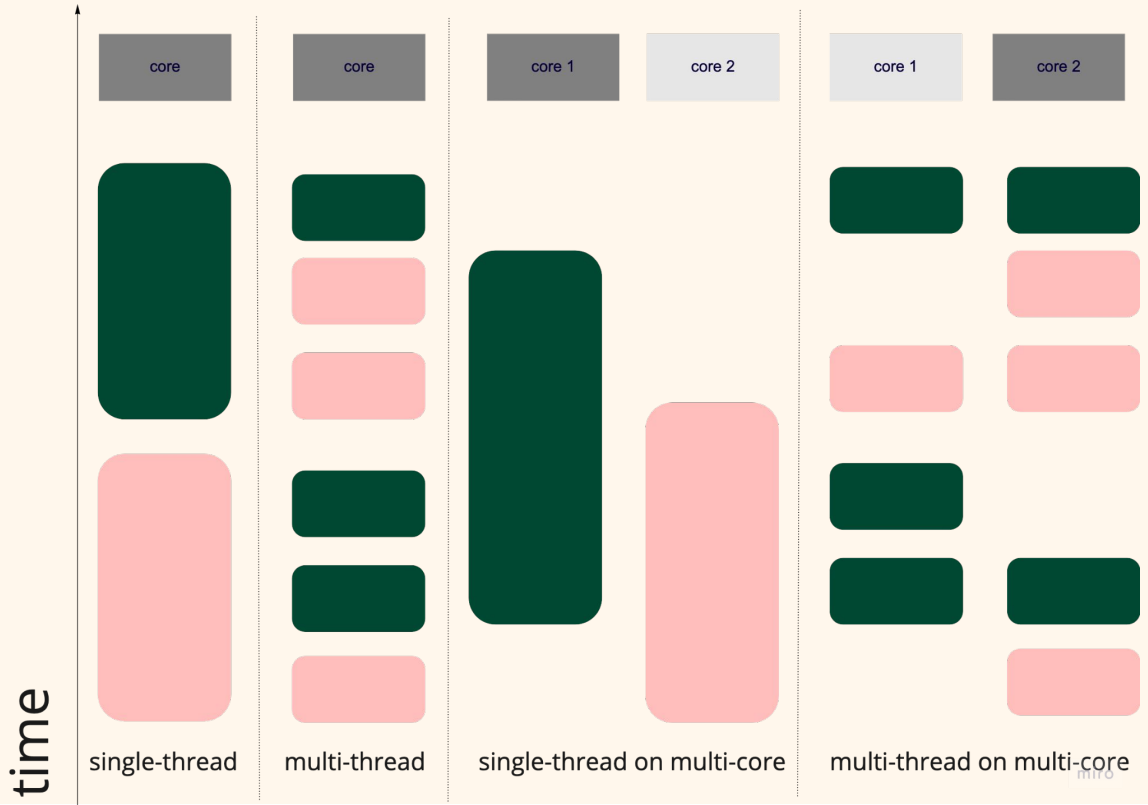
# GIL: GLOBAL INTERPRETER LOCK



It is a mutex that protects access to Python objects
CPython's memory management is not thread-safe

- Prevents multiple threads from executing Python bytecodes at once
- Prevents race conditions and ensures thread safety
- It was implemented in the 90s, when threads were getting popular but multi-cores machines were rarity

# QUICK RECAP: MULTIPROCESSING



# BASICS REFERENCE COUNTING

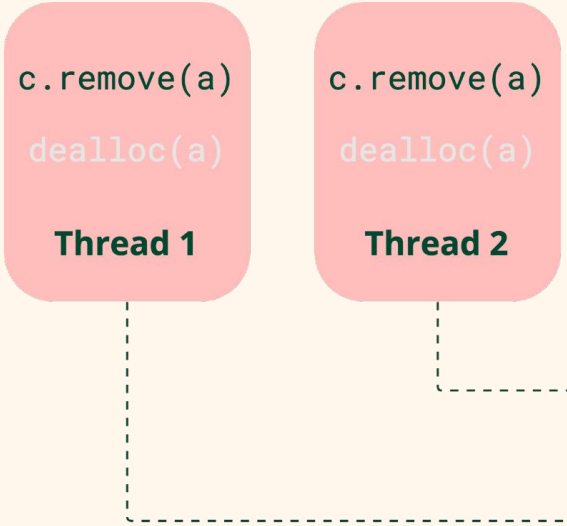
**Interpreter keeps a counter that tracks every reference to an object**

The counter increases with new references and decrease when we remove an object

```
>>> import sys
>>> a = '123456'
>>> c = [a, a]
>>> sys.getrefcount(a)
4
>>> c.remove(a)
>>> c.remove(a)
>>> sys.getrefcount(a)
2
```

# REFERENCE COUNTING + THREADS

## Where the trouble begin




```
load ob_recnt, ax
decrement ax
store ax, ob_refcnt
if ax > 0 jump ...
call ob_type -> dealloc
```

# GIL IN ACTION

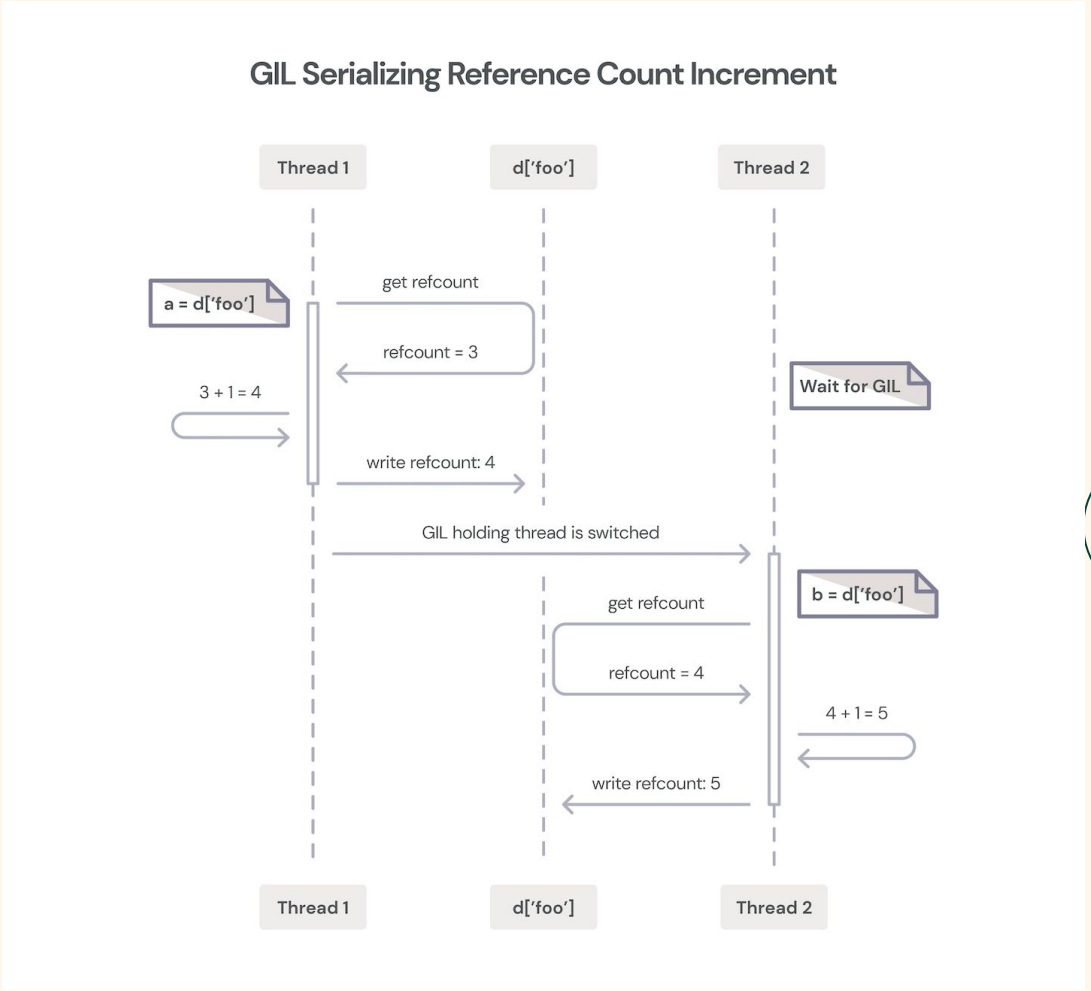
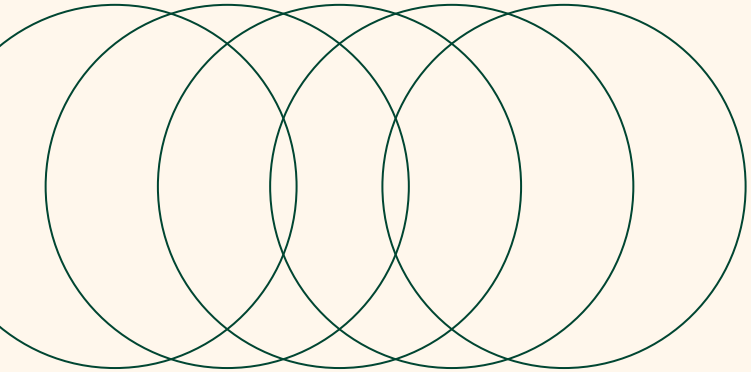
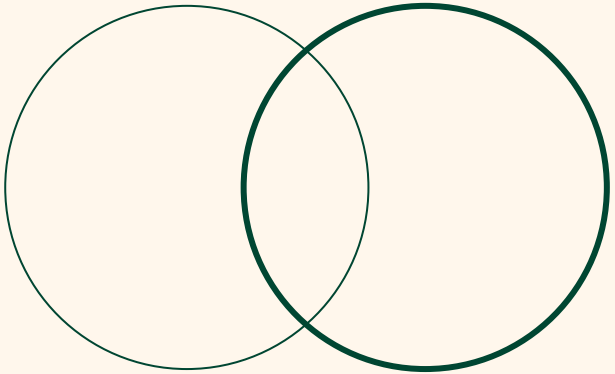



Image from https://www.backblaze.com/blog/the-python-gil-past-present-and-future/



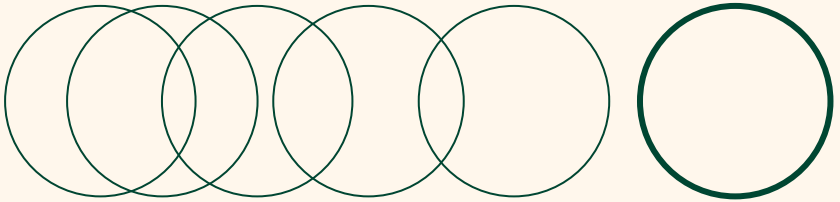


02

# MORE ABOUT GIL

Why is GIL so infamous?

# GIL: THE GLOBAL INFAMOUS LOCK

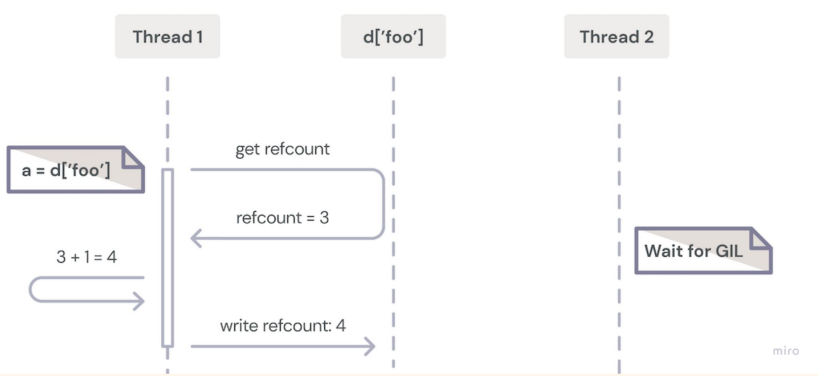


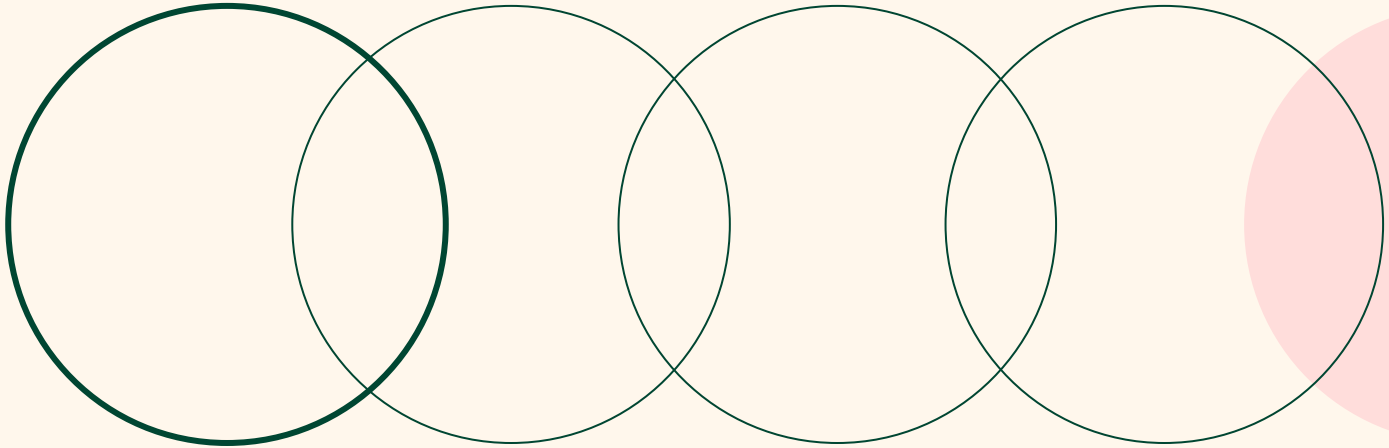
- Threads + Reference counting  = Trouble
- A Global Interpreter Lock is implemented to avoid that two or more threads accesses the Python objects at the same time
- Compatible with non-thread safe extensions
- Helped on the popularization of the language
- Performs good also on single-thread

# GIL: THE GLOBAL INFAMOUS LOCK

Okay, GIL solves a couple of problems but also cause others:

- We cannot run tasks simultaneously even with multicore
- Prevents multithreaded CPython programs from taking full advantage of multiprocessor systems in certain situations.





03

# ALTERNATIVES

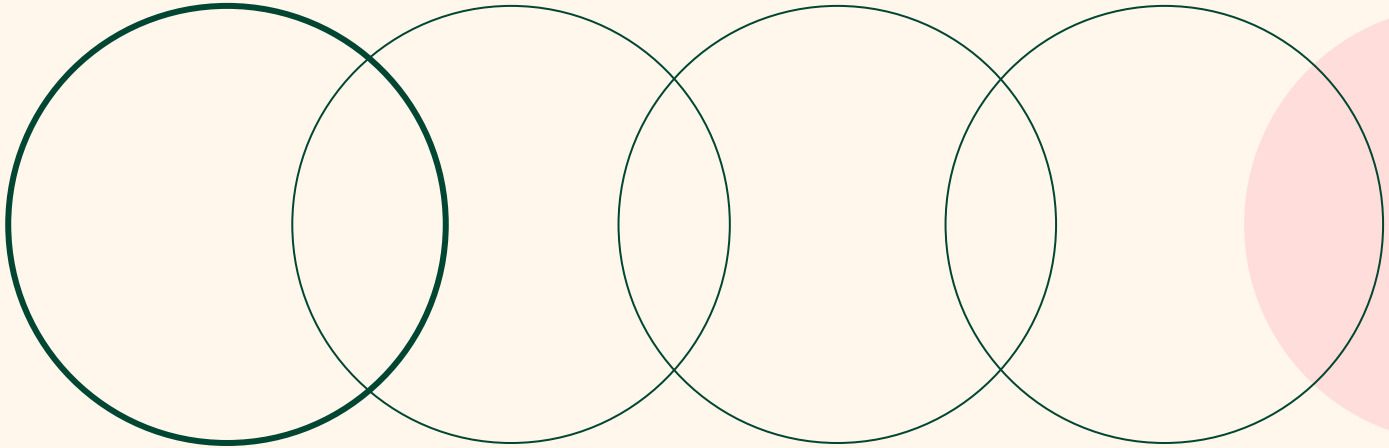
What is GIL and which problems does it solves

# NON-CPYTHON IMPLEMENTATIONS

- Jython have no GIL and can fully exploit multiprocessor system, but it doesn't support Python 3
- Pypy-stm, version of PyPy to support Software transactional Memory, similar to database transactions
- Stackless python: A Python Implementation That Does Not Use The C Stack


stackless-dev / stackless Public

forked from python/cpython

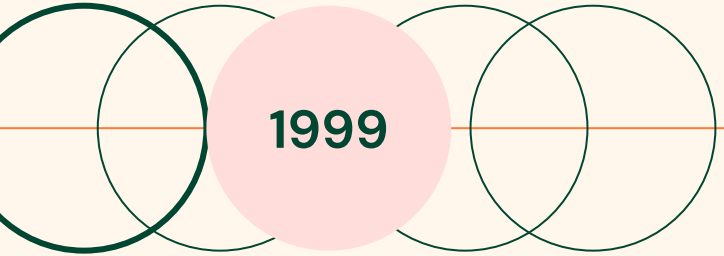


04

# REMOVING GIL

Why it isn't that easy?

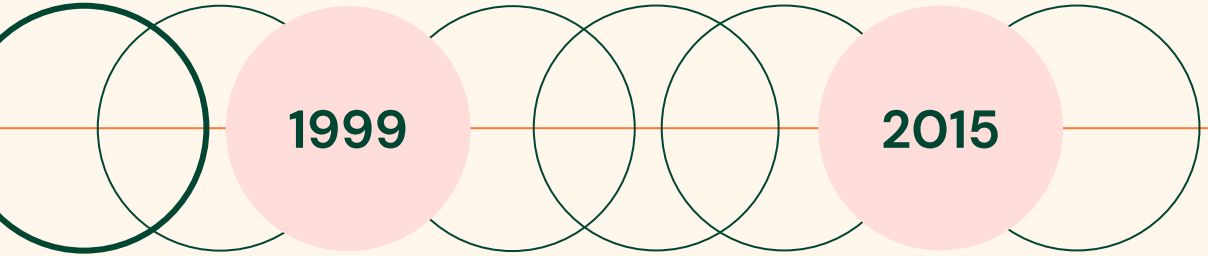
# REMOVING GIL




## FreeThreading

Greg Stein’s “free threading” work was one of the first (successful!) attempts to remove the GIL

# REMOVING GIL




**FreeThreading**

The first (successful!) attempts to remove the GIL

**Gilectomy**

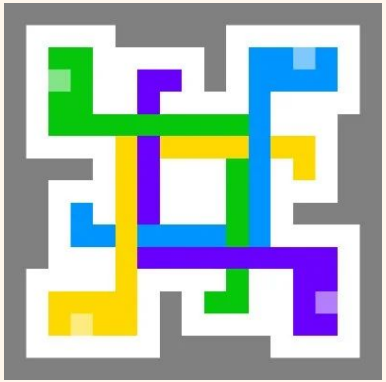
Larry Hasting's Gilectomy

# GILECTOMY

Larry Hasting’s Gilectomy presented on the PyCon 2016, introduced a bit more about how to remove the GIL, technical and "political" considerations.

First, four technical considerations that must be addressed when removing the GIL:

1. Reference Counting
2. Global and Statics
3. C Extensions
4. Atomicity



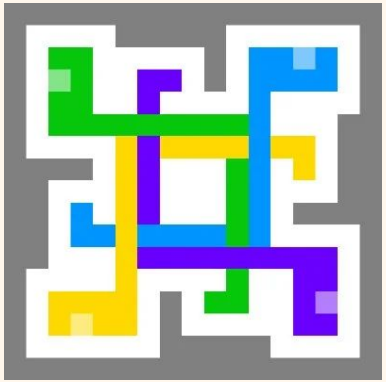
# GILECTOMY

Considerations to keep the good relationship between Python developers and Python users

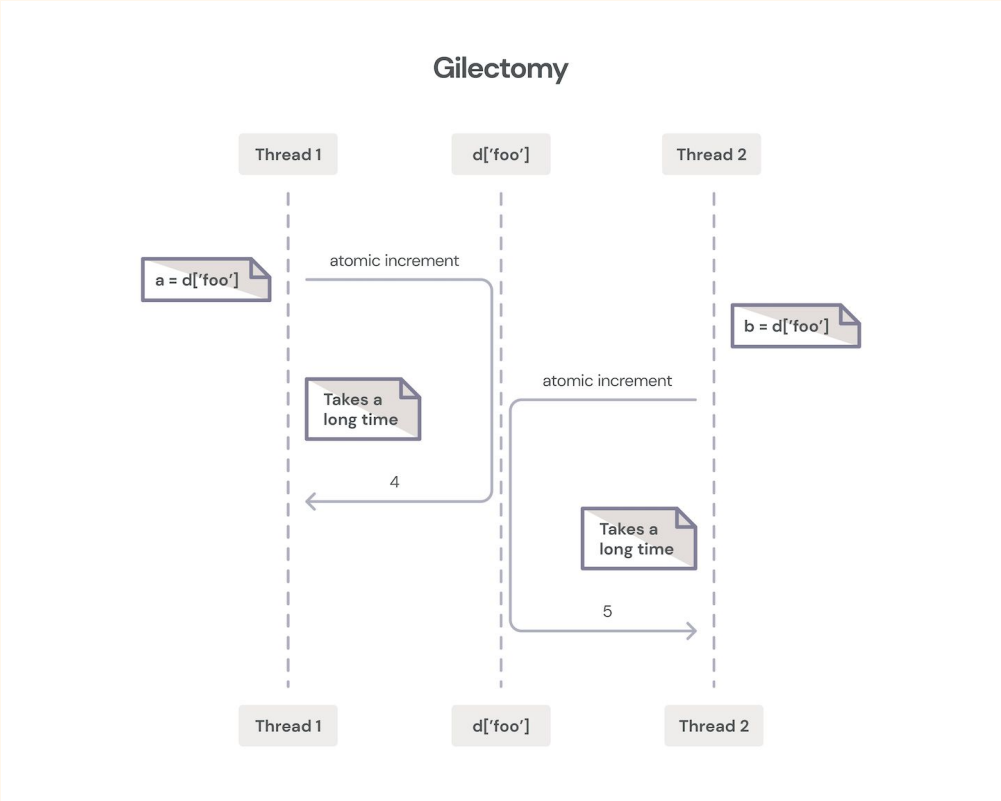
1. Removing the GIL should not hurt performance for single-threaded or I/O-bound multithreaded code.
2. Keep compatibility with C extensions
3. Don’t let GIL removal make the CPython interpreter too complicated or difficult to understand.

# GILECTOMY

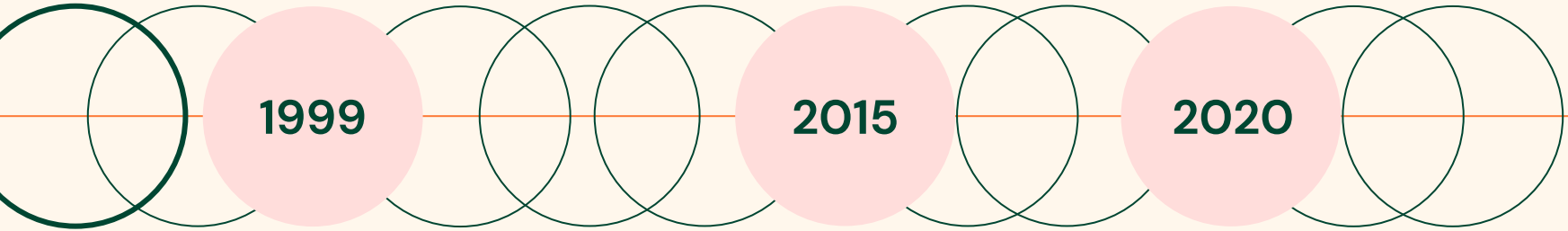
- Removing GIL isn't the hard part
- The hard part is doing keeping the mentioned technical and social constraints
- Retaining Python’s single-threaded performance
- Building a mechanism that scales with the number of cores, so it doesn't get in a plateau with 4 or 8 core



# GILECTOMY


Gilectomy
Thread 1
d['foo']
Thread 2
a = d['foo']
atomic increment
b = d['foo']
Takes a long time
atomic increment
4
Takes a long time
5
Thread 1
d['foo']
Thread 2

# REMOVING GIL




**FreeThreading**

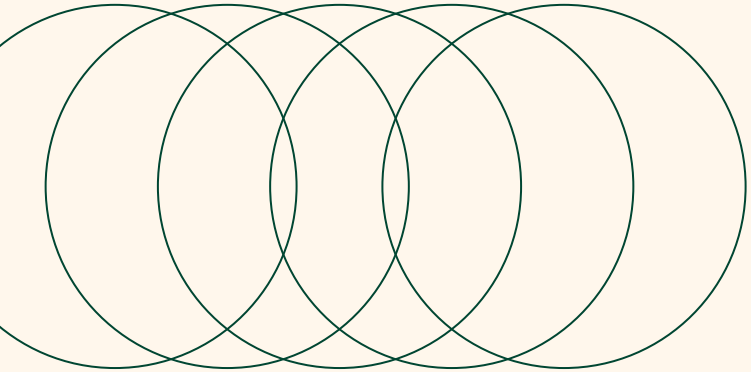
The first (successful!) attempts to remove the GIL

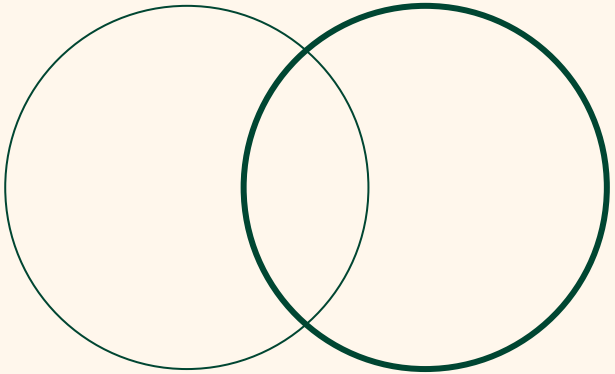
**Gilectomy**

Larry Hasting's Gilectomy

**nogil**

Promising solution



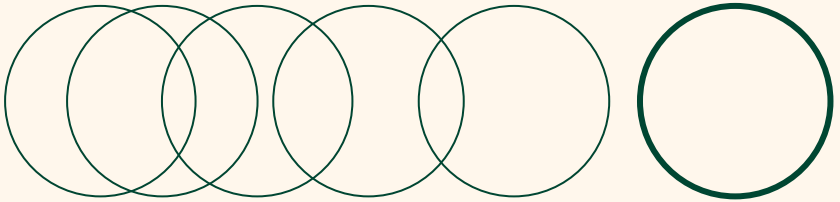


# 05

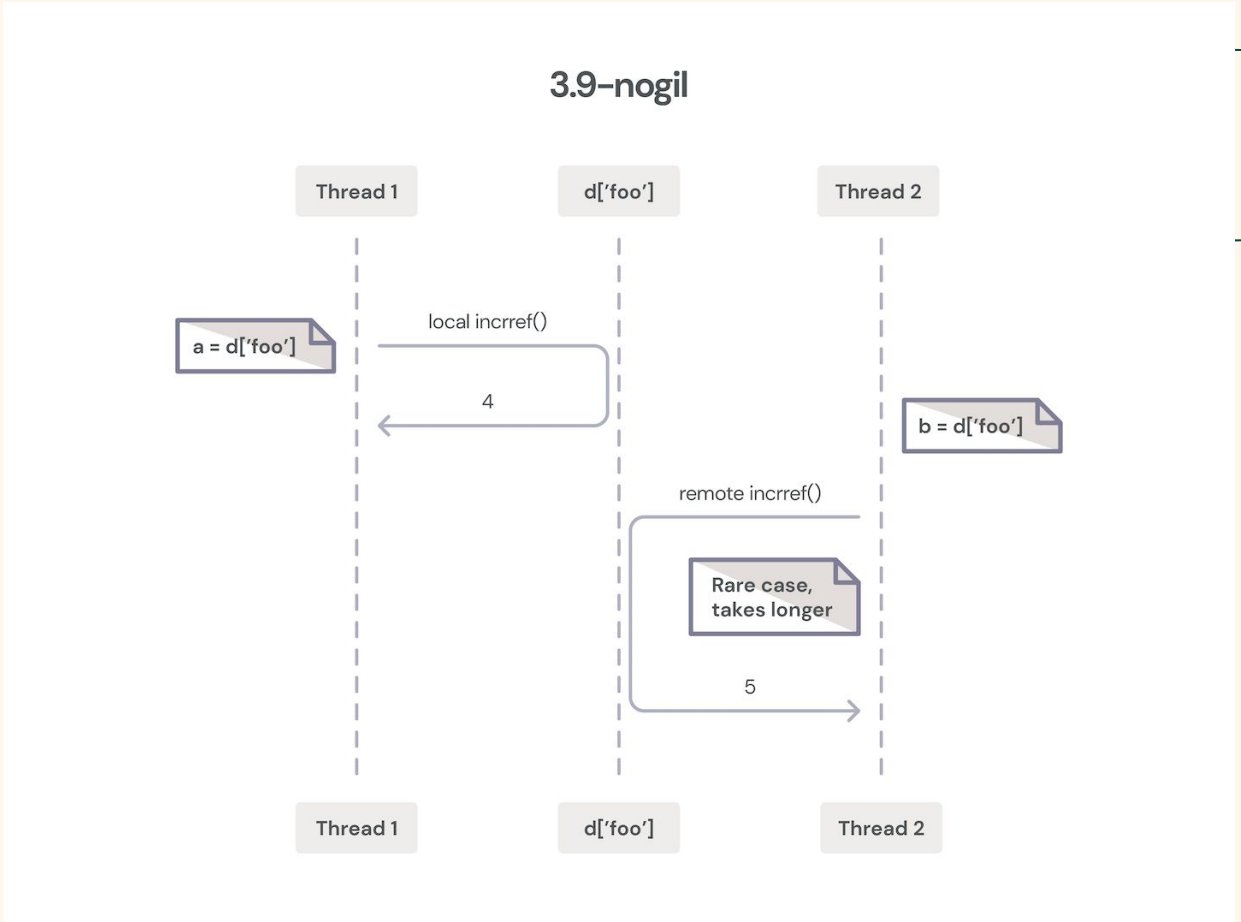
# FUTURE OF GIL

More GILs or NOGIL that is the question

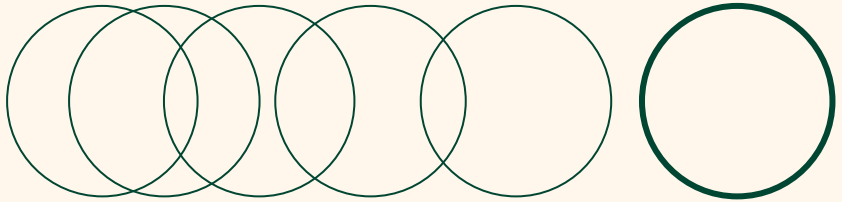
# NOGIL



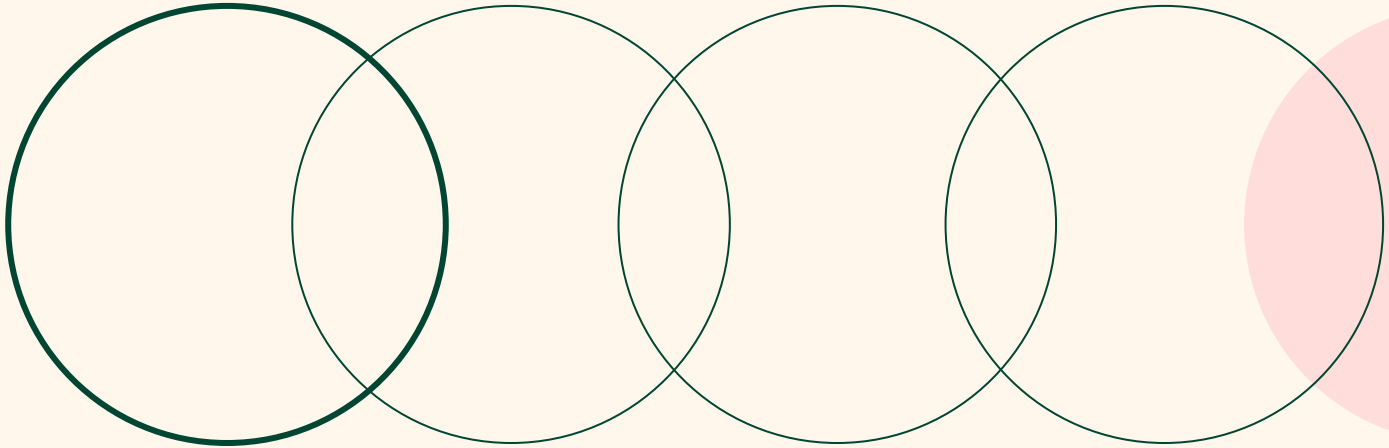
- Sam Gross' "nogil" work, which holds the promise of a performant, GIL-less CPython with minimal backward incompatibilities at both the Python and C layers
- The no-GIL proof-of-concept interpreter is about 10% faster than CPython 3.9 (and 3.10) on the pyperformance benchmark suite. It gets about the same average performance as the "main" branch of CPython 3.11 as of early September 2021.


3.9-nogil
Thread 1
d['foo']
Thread 2
a = d['foo']
local incrref()
4
b = d['foo']
remote incrref()
Rare case,
takes longer
5
Thread 1
d['foo']
Thread 2

# PEP 554: MULTI INTERPRETERS



- It is an on-going work proposing a module `interpreters` to add multiple interpreters to python
- Along with exposing the existing (in CPython) subinterpreter support, the module will also provide a mechanism for sharing data between interpreters.
- This mechanism centers around “channels”, which are similar to queues and pipes.
- if the GIL could be moved from global state to per-interpreter state, each interpreter instance could theoretically run concurrently with the others



# 06

# CONCLUSIONS

# GIL: IMPORTANCE AND PROBLEMS

We saw a bit about the Global Interpreter Lock:

- How it implements a solution make Python compatible with threads
- GIL importance in popularizing python, compatibility with C extensions
- Still maintains a good performance with single-threads processing
- However, it doesn't allow Python to get the full performance of threads and multicore
- It is hard to remove keeping it simple with good single-thread performance but seems to be more possible now

# THANK YOU!

## DOES ANYONE HAVE ANY QUESTIONS?

linkedin.com/in/avaladares/

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik**