Practical Challenges of Python Core Team

Donghee Na
donghee.na@python.org

Speaker

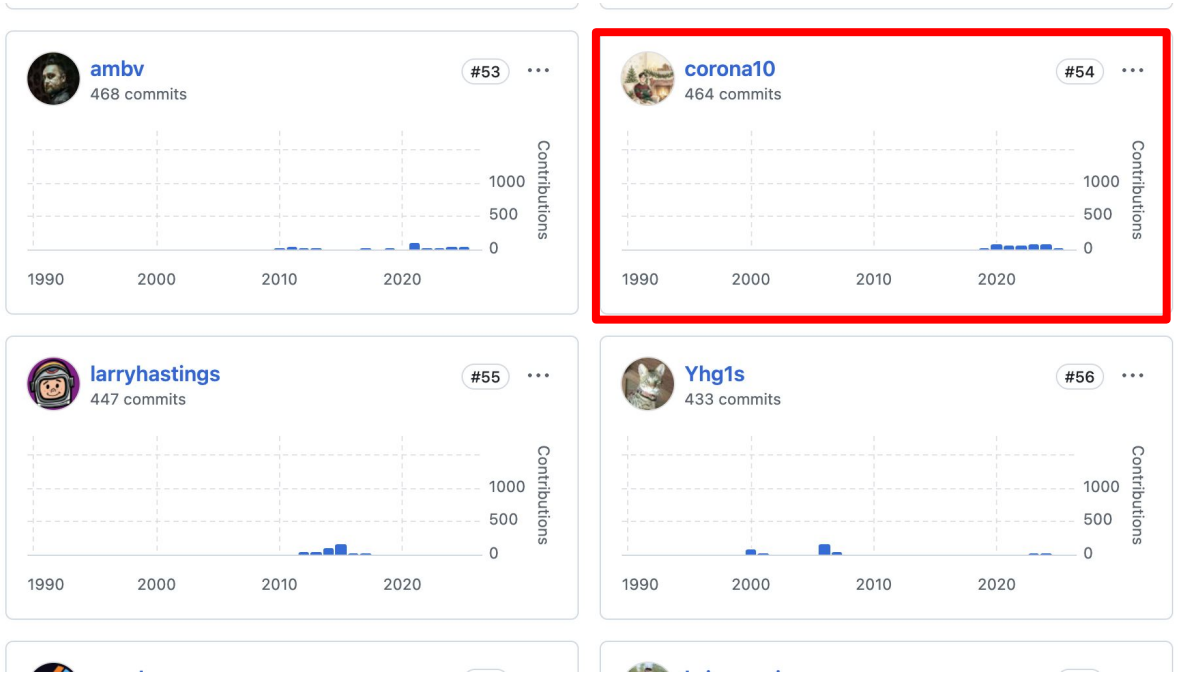
Software Development From 2018

- Karrot Lead SWE From October 2024
- LINE SWE: 2021 - 2024
- Kakao SWE: 2018 - 2021

CPython Core Development From 2017

- 2025/2026 Python Steering Council Member
- 2020 - Present: CPython Core Developer
- 2019 - 2020: CPython Triage Member
- donghee.na@python.org

Speaker

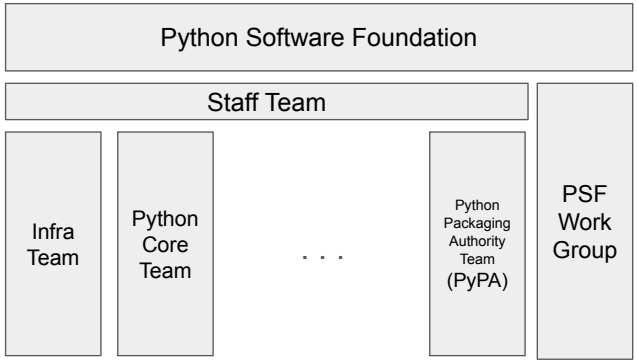

ambv
468 commits
#53
corona10
464 commits
#54
larryhastings
447 commits
#55
Yhg1s
433 commits
#56
Contributions
1000
500
0
1990
2000
2010
2020

Today Topics

- Governance and Structure of the Python Core Team
- Performance Improvement Challenges
- Concurrency Improvement Challenges
- Behind the Scenes

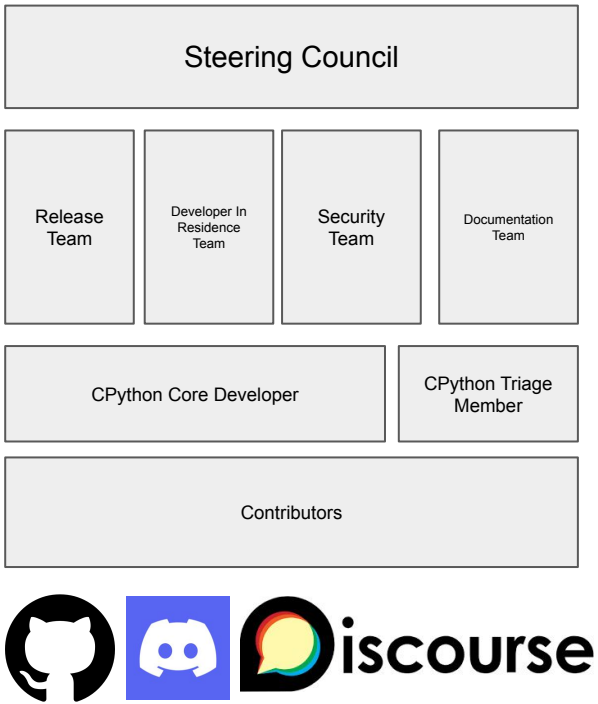
Introduction to the Python Core Team

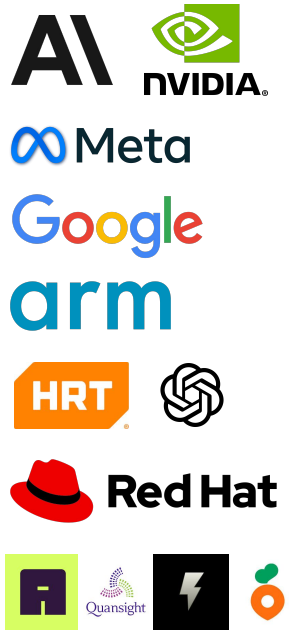
- The Python Core Team is responsible for developing and maintaining CPython, the reference implementation of the Python programming language.
- It operates under the Python Software Foundation (PSF) and is a separate group from the Python Packaging Authority (PyPA).



Introduction to the Python Core Team

- Triage Member
- Core Developer
- Release Manager
- Steering Council Member

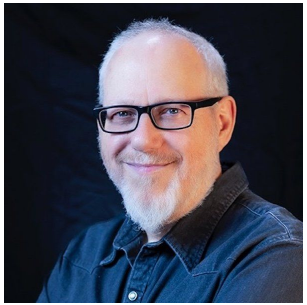




2026 Steering Council Member

- Pablo Galindo Salgado: Hudson River Trading
- Savannah Ostrowski: FastAPI Labs
- Barry Warsaw: Nvidia
- Donghee Na: Karrot
- Thomas Wouters: Meta





https://peps.python.org/pep-8107/


Steering Council Responsibilities

- Accept/reject PEPs (Python Enhancement Proposals)
- Maintain quality & stability of language & CPython
- Work with the PSF to manage project assets
- Seek consensus
- Delegate decisions whenever possible
- Veto (or not!) newly elected core members
- Office Hours

Steering Council Cadence

- Weekly Meetings - 90 minutes
  - Regular syncs w/
    - Deb Nicholson - PSF Executive Director
    - Łukasz Langa - Developer in Residence
- Office Hours (Thursdays at 1:00 PM PST)

Steering Council Responsibilities

- 2025 Meeting log
  - https://discuss.python.org/t/2025-psc-meeting-summaries/105626

CPython Core Team Technical Challenges

- Performance Improvements
  - Interpreter
  - Tier 2 Interpreter
  - JIT
- Concurrency Improvements
  - GIL: Global Interpreter Lock
  - Free-threading
  - Sub Interpreter

Topic: Performance Improvements

Many Attempts to Make CPython Faster

- Unladen Swallow (2011)
- Pyston V1 (2014)
- Cinder (2021-Current)
- Pyston V2 (2021)
- Faster CPython Project (2021 - Current)

CPython Core Team Technical Challenges

- Performance Improvements
  - Interpreter
  - Tier 2 Interpreter
  - JIT
- Concurrency Improvements
  - GIL: Global Interpreter Lock
  - Free-threading
  - Sub Interpreter

PEP 659: Specializing Adaptive Interpreter

```
def simulate_ducks(ducks):
    for duck in ducks:
        sound = duck.quack()
        if duck.echo:
            sount += sound
        print(sound)
```

```
L1: FOR_ITER
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR
    CALL
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR
    TO_BOOL
    POP_JUMP_IF_FALSE
    LOAD_FAST_BORROW
    LOAD_FAST_BORROW
    BINARY_OP
    STORE_FAST
L2: LOAD_GLOBAL

    LOAD_FAST_BORROW

    CALL
    POP_TOP
    JUMP_BACKWARD
```

```
L1: FOR_ITER_LIST
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR_METHOD_WITH_VALUES
    CALL_BY_EXACT_ARGS
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR_NONDESCRIPTOR_WITH_VALUES
    TO_BOOL_BOOL
    POP_JUMP_IF_FALSE
    LOAD_FAST_BORROW
    LOAD_FAST_BORROW
    BINARY_OP_INPLACE_ADD_UNICODE
    STORE_FAST
L2: LOAD_GLOBAL_BUILTIN

    LOAD_FAST_BORROW

    CALL_BUILTIN_FAST_WITH_KEYWORDS
    POP_TOP
    JUMP_BACKWARD
```

PEP 659: Specializing Adaptive Interpreter

Benchmark	3.10.4	3.11.0b3
deltablue	7.32 ms	**3.62 ms: 2.02x faster**
scimark_sor	196 ms	115 ms: 1.70x faster
raytrace	463 ms	292 ms: 1.58x faster
dulwich_log	75.2 ms	**63.1 ms: 1.19x faster**
python_startup	9.21 ms	**8.33 ms: 1.11x faster**
telco	6.60 ms	**6.49 ms: 1.02x faster**

PEP 744 – JIT Compilation

- C-like domain-specific language
- Tier 2 Interpreter
- Copy And Patch JIT

C-like DSL: Before CPython 3.12

```
case TARGET(BINARY_POWER): {
    PyObject *exp = POP();
    PyObject *base = TOP();
    PyObject *res = PyNumber_Power(base, exp, Py_None);
    Py_DECREF(base);
    Py_DECREF(exp);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}

case TARGET(BINARY_MULTIPLY): {
    PyObject *right = POP();
    PyObject *left = TOP();
    PyObject *res = PyNumber_Multiply(left, right);
    Py_DECREF(left);
    Py_DECREF(right);
    SET_TOP(res);
    if (res == NULL)
        goto error;
    DISPATCH();
}
```

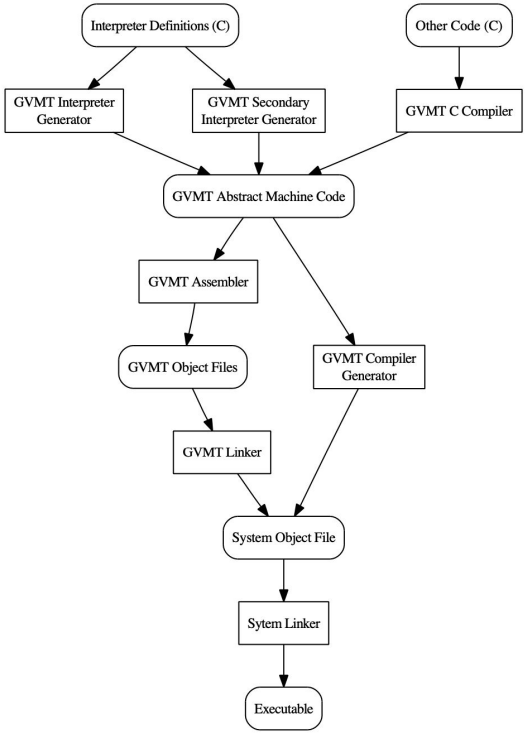
C-like DSL: After CPython 3.12

```
macro(BINARY_OP_MULTIPLY_FLOAT) =
    _GUARD_TOS_FLOAT + _GUARD_NOS_FLOAT + unused/5 + _BINARY_OP_MULTIPLY_FLOAT +
_POP_TOP_FLOAT + _POP_TOP_FLOAT;
macro(BINARY_OP_ADD_FLOAT) =
    _GUARD_TOS_FLOAT + _GUARD_NOS_FLOAT + unused/5 + _BINARY_OP_ADD_FLOAT + _POP_TOP_FLOAT +
_POP_TOP_FLOAT;
macro(BINARY_OP_SUBTRACT_FLOAT) =
    _GUARD_TOS_FLOAT + _GUARD_NOS_FLOAT + unused/5 + _BINARY_OP_SUBTRACT_FLOAT +
_POP_TOP_FLOAT + _POP_TOP_FLOAT;

pure op(_BINARY_OP_ADD_UNICODE, (left, right -- res, l, r)) {
    PyObject *left_o = PyStackRef_AsPyObjectBorrow(left);
    PyObject *right_o = PyStackRef_AsPyObjectBorrow(right);
    assert(PyUnicode_CheckExact(left_o));
    assert(PyUnicode_CheckExact(right_o));

    STAT_INC(BINARY_OP, hit);
    PyObject *res_o = PyUnicode_Concat(left_o, right_o);
    res = PyStackRef_FromPyObjectSteal(res_o);
    if (PyStackRef_IsNull(res)) {
        ERROR_NO_POP();
    }
    l = left;
    r = right;
    INPUTS_DEAD();
}
```

C-like DSL: After CPython 3.12



This design philosophy directly influenced CPython's Tier 2 interpreter and JIT.

"The Glasgow Virtual Machine Toolkit automatically generates a just-in-time compiler, integrates precise garbage collection into the virtual machine, and automatically manages the complex interdependencies between all the virtual machine components."

Source: Mark Shannon, The Construction of High-Performance Virtual Machines for Dynamic Languages, PhD thesis, University of Glasgow, 2011.

PEP 659: Specializing Adaptive Interpreter

```
def simulate_ducks(ducks):
    for duck in ducks:
        sound = duck.quack()
        if duck.echo:
            sount += sound
        print(sound)
```

```
L1: FOR_ITER
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR
    CALL
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR
    TO_BOOL
    POP_JUMP_IF_FALSE
    LOAD_FAST_BORROW
    LOAD_FAST_BORROW
    BINARY_OP
    STORE_FAST
L2: LOAD_GLOBAL

    LOAD_FAST_BORROW

    CALL
    POP_TOP
    JUMP_BACKWARD
```

```
L1: FOR_ITER_LIST
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR_METHOD_WITH_VALUES
    CALL_BY_EXACT_ARGS
    STORE_FAST
    LOAD_FAST_BORROW
    LOAD_ATTR_NONDESCRIPTOR_WITH_VALUES
    TO_BOOL_BOOL
    POP_JUMP_IF_FALSE
    LOAD_FAST_BORROW
    LOAD_FAST_BORROW
    BINARY_OP_INPLACE_ADD_UNICODE
    STORE_FAST
L2: LOAD_GLOBAL_BUILTIN

    LOAD_FAST_BORROW

    CALL_BUILTIN_FAST_WITH_KEYWORDS
    POP_TOP
    JUMP_BACKWARD
```

Tier 2 Interpreter (Traced Micro-Ops)

_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_LIST
_GUARD_NOT_EXHAUSTED_LIST
_ITER_NEXT_LIST_TIER_TWO
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST_BORROW
_CHECK_VALIDITY
_SET_IP
_GUARD_TYPE_VERSION
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_METHOD_WITH_VALUES
_CHECK_VALIDITY
_SET_IP
_CHECK_PEP_523
_CHECK_FUNCTIO_VERSION
_CHECK_FUNCTION_EXACT_ARGS
_CHECK_STACK_SPACE
_CHECK_RECURSION_REMAINING
_INIT_CALL_PY_EXACT_ARGS

_SAVE_RETURN_OFFSET
_PUSH_FRAME
_CHECK_VALIDITY
_SET_IP
_RESUME_CHECK
_CHECK_VALIDITY
_SET_IP
_LOAD_CONST_MORTAL
_CHECK_VALIDITY
_SET_IP
_RETURN_VALUE
_CHECK_VALIDITY
_SET_IP
_STORE_FAST
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST_BORROW
_CHECK_VALIDITY
_SET_IP
_GUARD_TYPE_VERSION
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_NONDECRIPTOR_WITH_VALUES
_CHECK_VALIDITY
_SET_IP

_TO_BOOL_BOOL
_CHECK_VALIDITY
_SET_IP
_GUARD_IS_FALSE_POP
_CHECK_VALIDITY
_SET_IP
_GUARD_GLOBALS_VERSION
_LOAD_GLOBAL_BUILTINS
_PUSH_NULL_CONDITIONAL
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST_BORROW
_CHECK_VALIDITY
_SET_IP
_CALL_BUILTIN_FAST_WITH_KEYWORDS
_CHECK_PERIODIC
_CHECK_VALIDITY
_SET_IP
_POP_TOP
_CHECK_VALIDITY
_SET_IP
_CHECK_PERIODIC
_JUMP_TO_TOP

Tier 2 Interpreter (Optimized Micro-Ops)

_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_LIST
_GUARD_NOT_EXHAUSTED_LIST
_ITER_NEXT_LIST_TIER_TWO
_CHECK_VALIDITY
_SET_IP
_STORE_FAST_1
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST_BORROW_1
_CHECK_VALIDITY
_SET_IP
_GUARD_TYPE_VERSION
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_METHOD_WITH_VALUES
_CHECK_VALIDITY
_SET_IP
_CHECK_PEP_523
_CHECK_FUNCTIO_VERSION
_CHECK_FUNCTION_EXACT_ARGS
_CHECK_STACK_SPACE_OPERAND
_CHECK_RECURSION_REMAINING
_INIT_CALL_PY_EXACT_ARGS_0

_SAVE_RETURN_OFFSET
_PUSH_FRAME
_CHECK_VALIDITY
_SET_IP
_RESUME_CHECK
_CHECK_VALIDITY
_SET_IP
_LOAD_CONST_INLINE
_CHECK_VALIDITY
_SET_IP
_RETURN_VALUE
_CHECK_VALIDITY
_SET_IP
_STORE_FAST_2
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST_BORROW_1
_CHECK_VALIDITY
_SET_IP
_GUARD_TYPE_VERSION
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_NONDECRIPTOR_WITH_VALUES
_CHECK_VALIDITY
_SET_IP

_TO_BOOL_BOOL
_CHECK_VALIDITY
_SET_IP
_GUARD_IS_FALSE_POP
_CHECK_VALIDITY
_SET_IP
_CHECK_FUNCTION
_LOAD_CONST_INLINE
_PUSH_NULL
_CHECK_VALIDITY
_SET_IP
_LOAD_FAST_BORROW_1
_CHECK_VALIDITY
_SET_IP
_CALL_BUILTIN_FAST_WITH_KEYWORDS
_CHECK_PERIODIC
_CHECK_VALIDITY
_SET_IP
_POP_TOP
_CHECK_VALIDITY
_SET_IP
_CHECK_PERIODIC
_JUMP_TO_TOP

Tier 2 Interpreter (Optimized Micro-Ops)

_CHECK_VALIDITY
_SET_IP
_ITER_CHECK_LIST
_GUARD_NOT_EXHAUSTED_LIST
_ITER_NEXT_LIST_TIER_TWO
_CHECK_VALIDITY
_SET_IP
_STORE_FAST_1
_CHECK_VALIDITY

_LOAD_FAST_BORROW_1

_GUARD_TYPE_VERSION
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_METHOD_WITH_VALUES

_SET_IP

_CHECK_FUNCTIO_VERSION
_CHECK_FUNCTION_EXACT_ARGS
_CHECK_STACK_SPACE_OPERAND
_CHECK_RECURSION_REMAINING
_INIT_CALL_PY_EXACT_ARGS_0

_SAVE_RETURN_OFFSET
_PUSH_FRAME

_RESUME_CHECK

_LOAD_CONST_INLINE
_CHECK_VALIDITY
_SET_IP
_RETURN_VALUE
_CHECK_VALIDITY
_SET_IP
_STORE_FAST_2
_CHECK_VALIDITY

_LOAD_FAST_BORROW_1

_SET_IP

_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_NONDECRIPTOR_WITH_VALUES
_CHECK_VALIDITY
_SET_IP

_TO_BOOL_BOOL

_GUARD_IS_FALSE_POP

_CHECK_FUNCTION
_LOAD_CONST_INLINE
_PUSH_NULL

_LOAD_FAST_BORROW_1

_SET_IP
_CALL_BUILTIN_FAST_WITH_KEYWORDS
_CHECK_PERIODIC
_CHECK_VALIDITY
_SET_IP
_POP_TOP

_SET_IP
_CHECK_PERIODIC
_JUMP_TO_TOP

Tier 2 Interpreter (Optimized Micro-Ops)

_ITER_CHECK_LIST
_GUARD_NOT_EXHAUSTED_LIST
_ITER_NEXT_LIST_TIER_TWO
_CHECK_VALIDITY
_SET_IP
_STORE_FAST_1

_LOAD_FAST_BORROW_1

_GUARD_TYPE_VERSION
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_METHOD_WITH_VALUES

_SET_IP

_CHECK_FUNCTIO_VERSION
_CHECK_FUNCTION_EXACT_ARGS
_CHECK_STACK_SPACE_OPERAND
_CHECK_RECURSION_REMAINING

_LOAD_CONST_INLINE
_CHECK_VALIDITY

_SET_IP
_STORE_FAST_2
_CHECK_VALIDITY

_LOAD_FAST_BORROW_1

_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION

_CHECK_FUNCTION
_LOAD_CONST_INLINE
_PUSH_NULL

_LOAD_FAST_BORROW_1

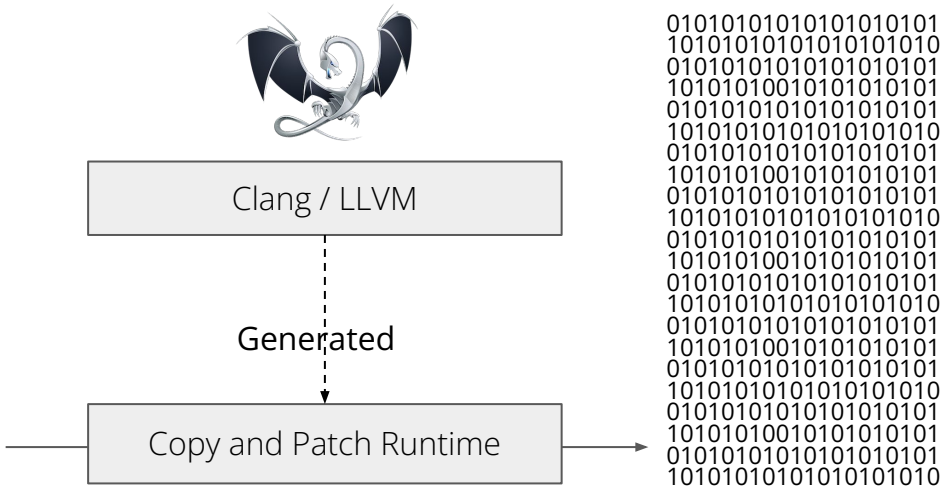
_SET_IP
_CALL_BUILTIN_FAST_WITH_KEYWORDS
_CHECK_PERIODIC
_CHECK_VALIDITY
_SET_IP
_POP_TOP

_CHECK_PERIODIC
_JUMP_TO_TOP

Copy and Patch JIT

Copy and Patch JIT

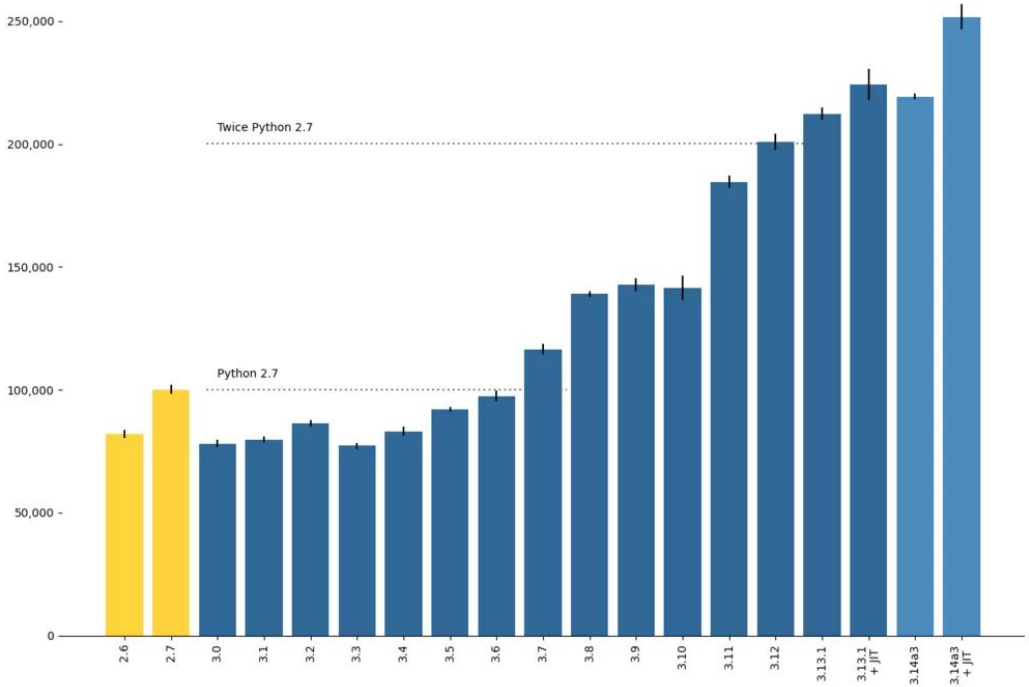
```
_ITER_CHECK_LIST
_GUARD_NOT_EXHAUSTED_LIST
_ITER_NEXT_LIST_TIER_TWO
_CHECK_VALIDITY
_SET_IP
_STORE_FAST_1
_LOAD_FAST_BORROW_1
_GUARD_TYPE_VERSION
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_LOAD_ATTR_METHOD_WITH_VALUES
_SET_IP
_CHECK_FUNCTION_VERSION
_CHECK_FUNCTION_EXACT_ARGS
_CHECK_STACK_SPACE_OPERAND
_CHECK_RECURSION_REMAINING
_LOAD_CONST_INLINE
_CHECK_VALIDITY
_SET_IP
_STORE_FAST_2
_CHECK_VALIDITY
_LOAD_FAST_BORROW_1
_GUARD_DORV_VALUES_INST_ATTR_FROM_DICT
_GUARD_KEYS_VERSION
_CHECK_FUNCTION
_LOAD_CONST_INLINE
_PUSH_NULL
_LOAD_FAST_BORROW_1
_SET_IP
_CALL_BUILTIN_FAST_WITH_KEYWORDS
_CHECK_PERIODIC
_CHECK_VALIDITY
_SET_IP
_POP_TOP
_CHECK_PERIODIC
_JUMP_TO_TOP
```



Machine Code

```
01010101010101010101
10101010101010101010
01010101010101010101
10101010010101010101
01010101010101010101
10101010101010101010
01010101010101010101
10101010010101010101
01010101010101010101
10101010101010101010
01010101010101010101
10101010010101010101
01010101010101010101
10101010101010101010
01010101010101010101
10101010010101010101
01010101010101010101
10101010101010101010
01010101010101010101
10101010010101010101
01010101010101010101
10101010101010101010
01010101010101010101
10101010010101010101
```

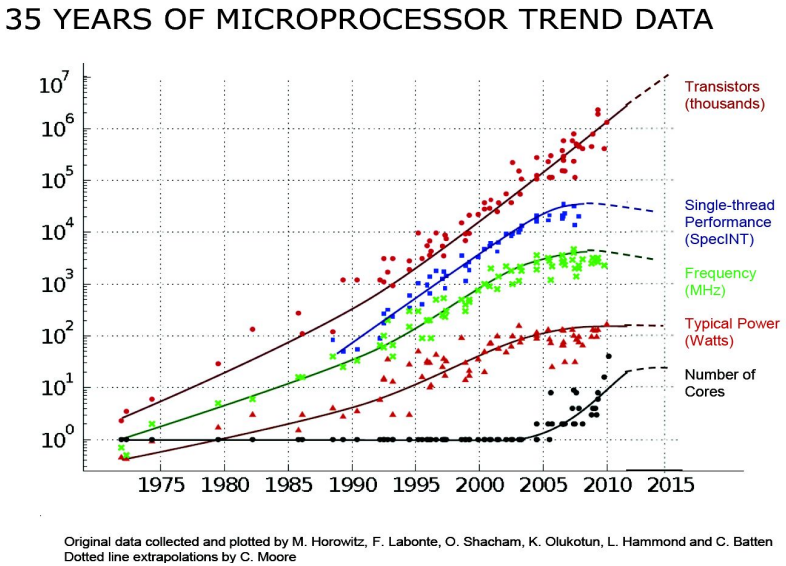
Result


250,000
200,000
150,000
100,000
50,000
0
Twice Python 2.7
Python 2.7
2.6
2.7
3.0
3.1
3.2
3.3
3.4
3.5
3.6
3.7
3.8
3.9
3.10
3.11
3.12
3.13.1
3.13.1 + JIT
3.14a3
3.14a3 + JIT

Topic: Concurrency Improvements

GIL

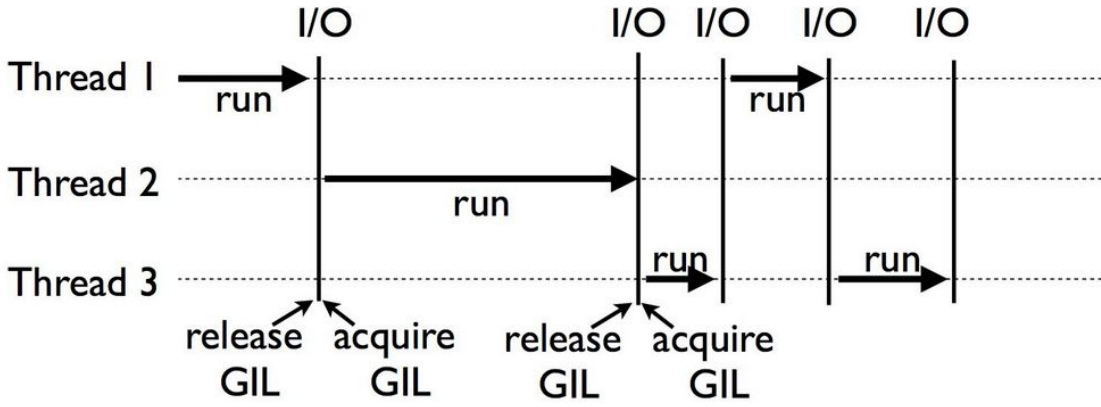
- GIL was first introduced at August 4th, 1992 by GvR.
  - https://github.com/python/cpython/commit/1984f1e1c6306d4e8073c28d2395638f80ea509b




```
+
+ #ifdef USE_THREAD
+ #include <errno.h>
+ #include "thread.h"
+ static type_lock interpreter_lock;
+
+ void
+ init_save_thread()
+ {
+ #ifdef USE_THREAD
+     if (interpreter_lock)
+         fatal("2nd call to init_save_thread");
+     interpreter_lock = allocate_lock();
+     acquire_lock(interpreter_lock, 1);
+ #endif
+ }
```

GIL

- Easy to implement
- No interpreter-level deadlock
- Low overhead at single thread



Many Attempts to Remove GIL

- Free-threading patch (1996)
- PyPy-STM (2012)
- Gilectomy Project (2016)
- Sub Interpreter (2017 - Current)
- NoGIL Free-threading (2021 - Current)

PEP 703: Making the Global Interpreter Lock Optional in CPython

- Proposed by Sam Gross at Meta FAIR.
- Based on *Choi, Shull & Torrellas, Biased reference counting: minimizing atomic operations in garbage collection, PACT 2018*

PEP 703: Object Header Changes

```
struct _object {
  _PyObject_HEAD_EXTRA
  uintptr_t ob_tid;         // owning thread id (4–8 bytes)
  uint16_t __padding;       // reserved for future use (2 bytes)
  PyMutex ob_mutex;         // per-object mutex (1 byte)
  uint8_t ob_gc_bits;       // GC fields (1 byte)
  uint32_t ob_ref_local;    // local reference count (4 bytes)
  Py_ssize_t ob_ref_shared; // shared reference count and state bits (4–8 bytes)
  PyTypeObject *ob_type;
};
```

the paper, local and shared refcounts are split in the object header to avoid overflow and reduce complexity.

PEP 703: Biased Reference Count

- Biased Reference Counting (Choi, 2018): Most objects are effectively single-thread owned; the owner updates a fast local refcount, while other threads update a shared refcount atomically.
- Layout choice: Instead of a single 64-bit packed field as in the paper, local and shared refcounts are split in the object header to avoid overflow and reduce complexity.

PEP 703: Deferred Reference Count

- Limit of BRC: Some objects are frequently accessed by multiple threads, which introduces overhead, but their lifetimes are too short to treat them as immortal.
- Interpreter level DRC: These objects use Deferred Reference Counting in the interpreter, skipping refcount updates on stack push and pop and reconstructing the skipped counts during GC.

PEP 703: Garbage Collections

- Explicit stop the world: Without the GIL, other threads can no longer be excluded during GC, so an explicit stop the world phase is required.
- ThreadState management: ThreadState must be explicitly tracked with three states: ATTACHED, DETACHED, and GC.
- Single generation GC: The plan is to replace generational GC with a single generation GC. Since most objects are reclaimed by refcounting, generational GC shows limited benefit, and frequent young collections are likely to hurt multithreaded performance.

PEP 703: Per-thread Specializer

- Each thread keeps its own specialized copy of bytecode
- Enables lock-free fast execution even without the GIL
- Main thread uses original bytecode (no extra copy → zero overhead)
- This design takes a different approach from the one Sam Gross introduced in his nogil implementation.

Result

Execution Overhead on pyperformance 1.0.6

	Intel Skylake	AMD Zen 3
One thread	6%	5%
Multiple threads	8%	7%

Topic: Behind the Scenes

Technical Challenges Behind the Scene

- C API Compatibility
- Various Hardware Supports
- Behavior Changes

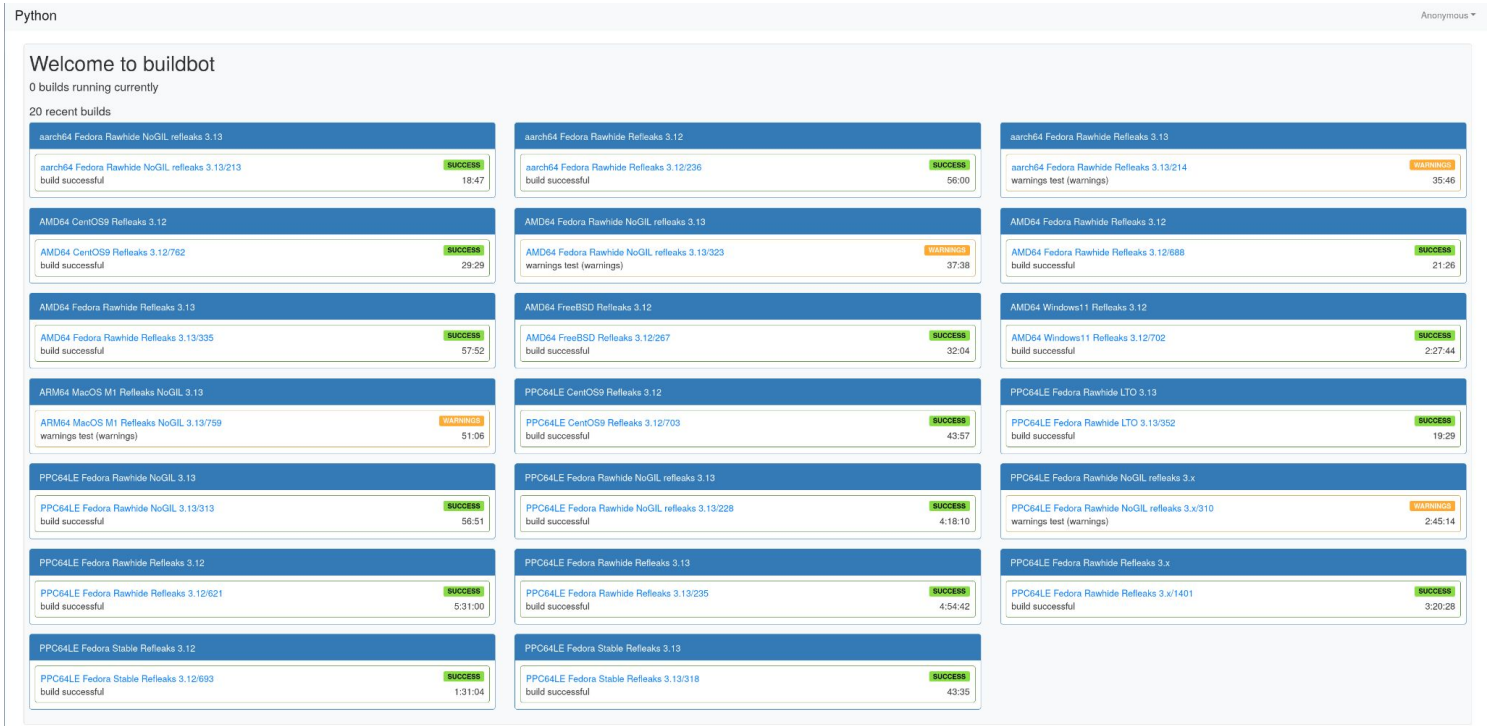
C API Compatibility

- Unstable API
  - may change in minor versions without a deprecation period. It is marked by the PyUnstable prefix in names
- Limited API
  - Extensions that only use the Limited API can be compiled once and be loaded on multiple versions of Python.
- Stable ABI
  - Set of symbols that will remain ABI-compatible across Python 3.x versions.

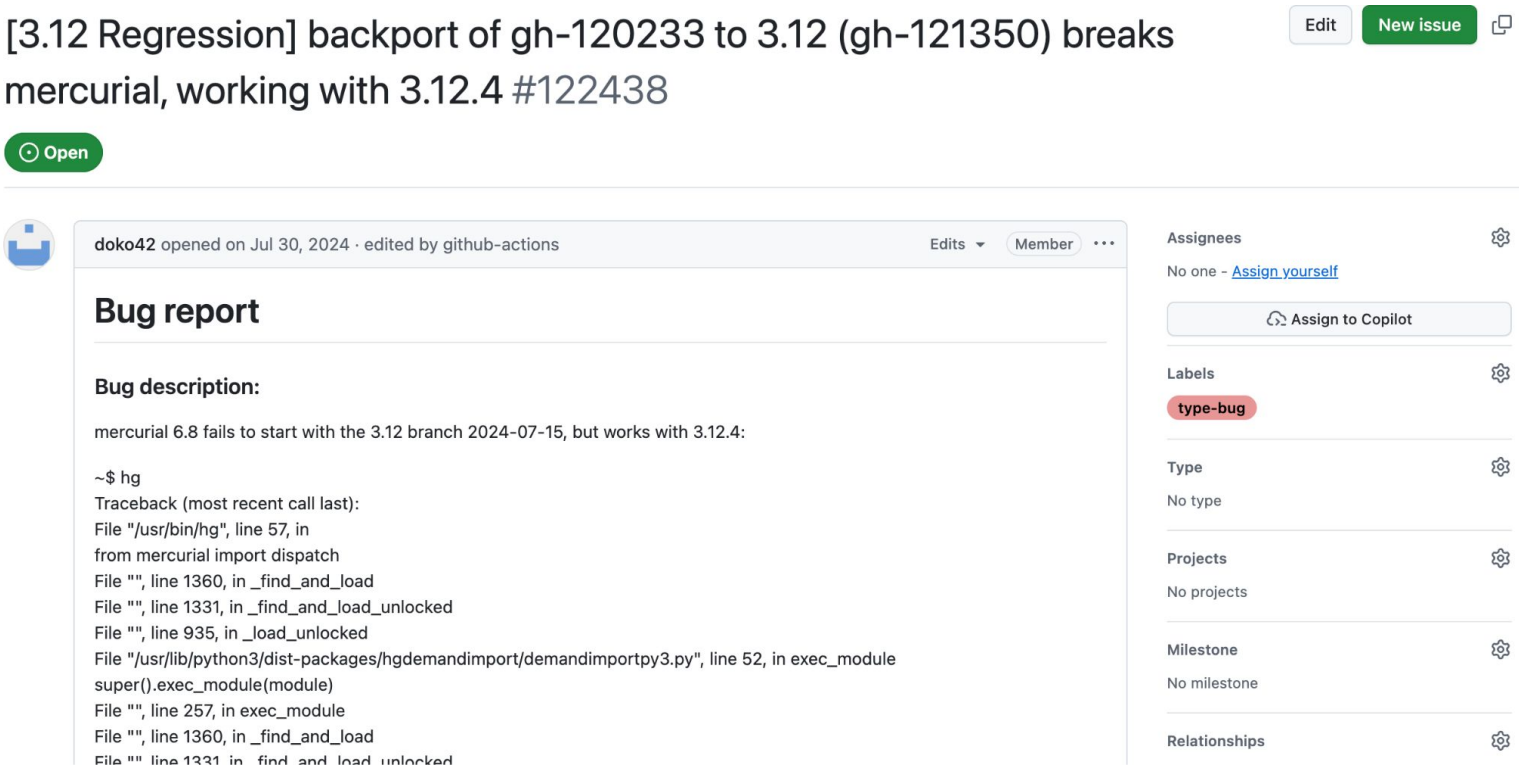
C API Compatibility

- Free-thrading
  - abi3 VS abi3t VS abi4
- Faster CPython
  - Compact Object Header

Various Hardware Supports



Behavior Changes



[3.12 Regression] backport of gh-120233 to 3.12 (gh-121350) breaks mercurial, working with 3.12.4 #122438

Edit | New issue

Open

doko42 opened on Jul 30, 2024 · edited by github-actions — Edits — Member

Bug report

Bug description:

mercurial 6.8 fails to start with the 3.12 branch 2024-07-15, but works with 3.12.4:

```
~$ hg
Traceback (most recent call last):
File "/usr/bin/hg", line 57, in
from mercurial import dispatch
File "", line 1360, in _find_and_load
File "", line 1331, in _find_and_load_unlocked
File "", line 935, in _load_unlocked
File "/usr/lib/python3/dist-packages/hgdemandimport/demandimportpy3.py", line 52, in exec_module
super().exec_module(module)
File "", line 257, in exec_module
File "", line 1360, in _find_and_load
File "", line 1331, in _find_and_load_unlocked
```

Assignees
No one - Assign yourself
Assign to Copilot

Labels
type-bug

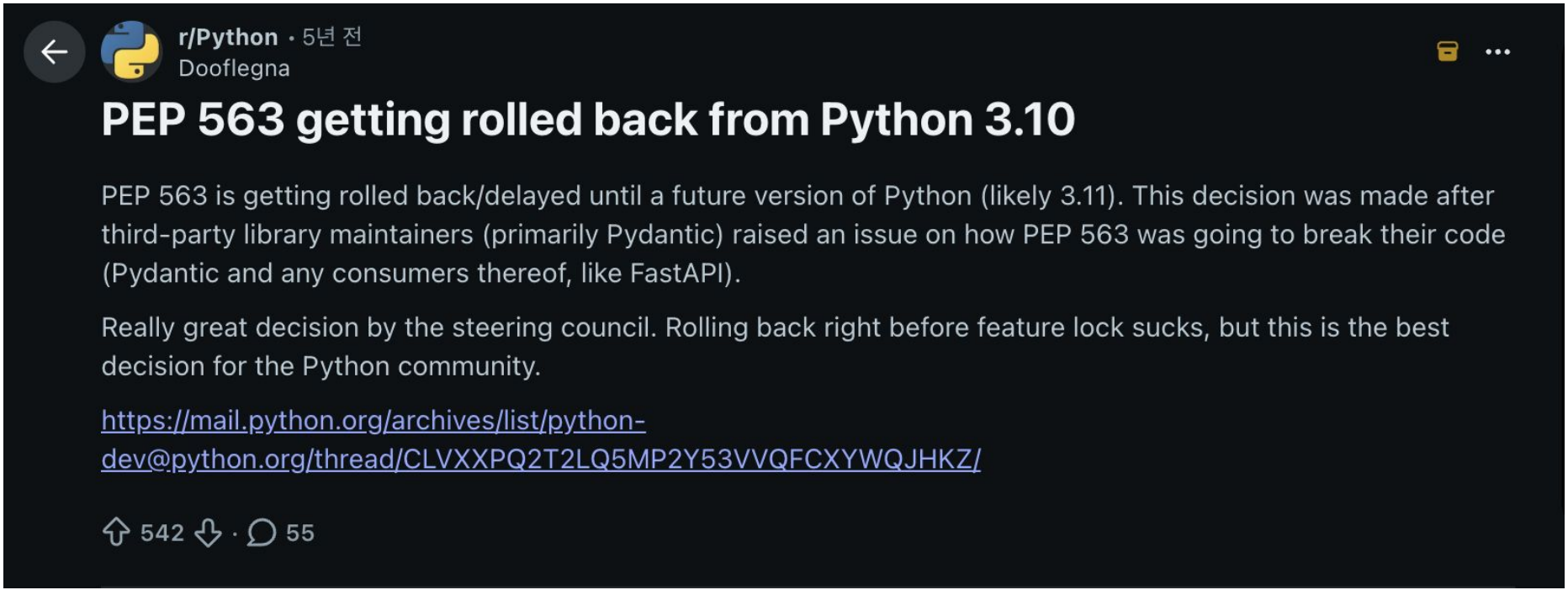
Type
No type

Projects
No projects

Milestone
No milestone

Relationships

Behavior Changes

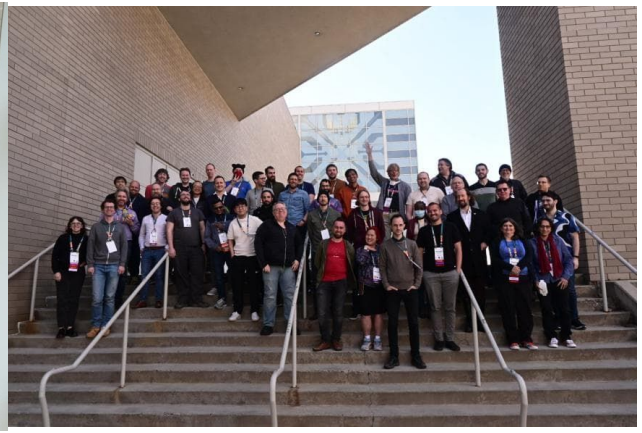


Topic: Core Team Annual Events

Python Language Summits

2025 (Pittsburgh, US)

2024 (Pittsburgh, US)



2023 (Salt Lake City, US)

Python Language Summits

- Python Language Summit 2025
  - Fearless Concurrency by Tobias Wrigstad, Matthew Parkinson, and Fridtjof Stoldt
  - An Uncontentious Talk about Contention: talk by Mark Shannon
  - What do Python core developers want from Rust?: talk by David Hewitt
- Python Language Summit 2024
  - Free-threading ecosystems by Daniele Parmeggiani
  - Native Interface and Limited C API by Petr Viktorin and Victor Stinner
- Python Language Summit 2023
  - Towards Native Profiling for Python by Joannah Nanjekye
  - Making the Global Interpreter Lock Optional by Sam Gross

Python Core Sprints

2025 (ARM.ltd)

2024 (Meta)

Python Core Sprints

2025 (ARM Ltd)

2024 (Meta)

This Year

- Python Language Summit at Euro Python (Porland)
- Python Core Sprint (TBA)

Future?

- Python Language Summit at Seoul?
- Python Core Sprint at Seoul?

Q&A