Broken `__slots__` are a silent performance killer—let's fix them!

July 18, 2025

Arie Bovenberg

Performance

Free-threading, asyncio, JIT, subinterpreters, Rust, ...

`__slots__`

About me

- Software Engineer @ KLM Royal Dutch Airlines
- Open source:
  - `whenever` (https://github.com/ariebovenberg/whenever)
  - `__slots__` nerd

What you didn’t know about `__slots__`

`__slots__` affect behavior

Normal

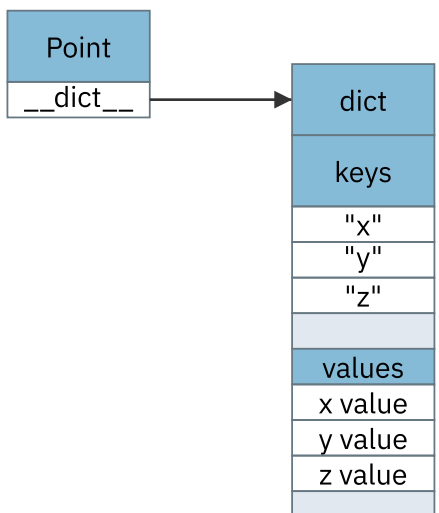
```
class Point:
    pass

p = Point()
p.x = 10
p.a = 9
p.__dict__  # {"x": 10, "a": 9}
```

With `__slots__`

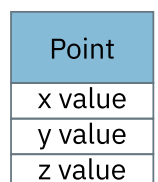
```
class Point:
    __slots__ = ("x", "y", "z")

p = Point()
p.x = 10
p.a = 9     # Error!
p.__dict__  # Error!
```

`__slots__` affect memory layout

Normal



With `__slots__`



Sizing things up

Normal

```
sizeof(p) + sizeof(p.__dict__)
# 344
```

```
tracemalloc.start()
_ = [Point(1, 2, 3)
     for _ in range(1_000_000)]
get_traced_memory()  # 104 MB
```

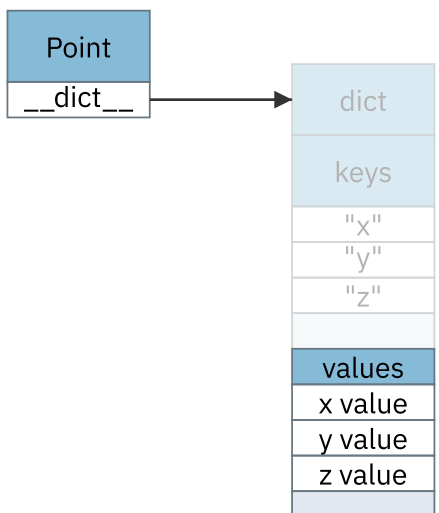
With `__slots__`

```
import sys.getsizeof as sizeof
sizeof(p)  # 56
```

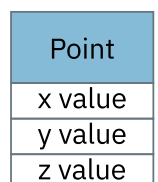
```
tracemalloc.start()
_ = [Point(1, 2, 3)
     for _ in range(1_000_000)]
get_traced_memory()  # 64 MB
```

Optimized memory footprint

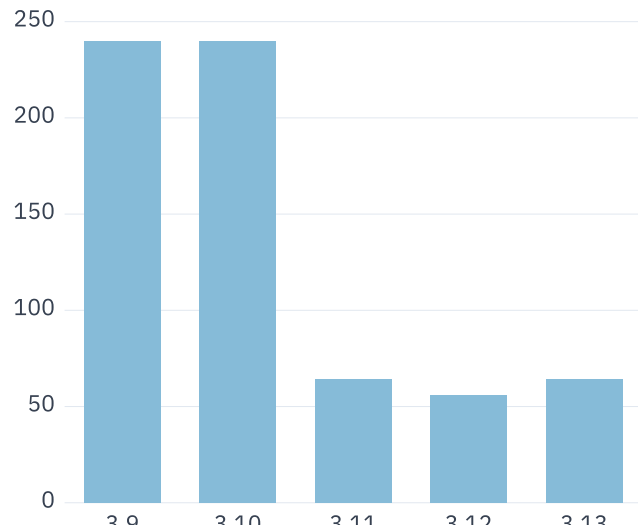
Normal



With `__slots__`



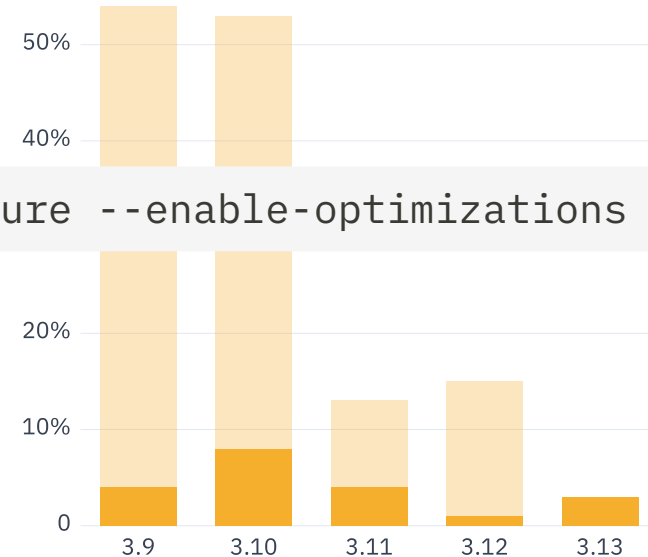
`__slots__` memory savings



Max. bytes per object saved

What about lookup speed?

`__slots__` lookup speed boost?




```
./configure --enable-optimizations --with-lto
```

Speed improvement of attribute lookup with `__slots__` over `__dict__`.

On modern Python...

Normal

±40-64 bytes overhead

dynamic attributes

With `__slots__`

0 bytes overhead

static attributes

0-10% faster attribute lookup

Using `__slots__` wrong

up to **400** bytes overhead

unreliably static attributes

up to **2x slower** attribute access

`__slots__` pitfalls

Unused slots

```
class Point:
    __slots__ = ("x", "y", "z")

p = Point()
p.x, p.y = 1, 2

getsizeof(p)  # 56
```

Duplicate slots

```
class Point:
    __slots__ = ("x", "y", "z", "x")

p = Point()
p.x, p.y, p.z = 1, 2, 3

getsizeof(p)  # 64
```

Overlapping slots

```
class Point2D:
    __slots__ = ("x", "y")

class Point(Point2D):
    __slots__ = ("x", "y", "z")

p = Point()
p.x, p.y, p.z = 1, 2, 3
getsizeof(p)  # 72
```

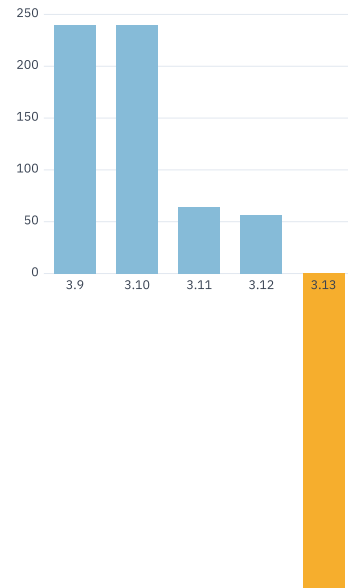
```
class Point2D:
    __slots__ = ("x", "y")

class Point(Point2D):
    __slots__ = ("z", )

p = Point()
p.x, p.y, p.z = 1, 2, 3
getsizeof(p)  # 56
```

Broken slots

```
class Base:
    pass

class Point(Base):
    __slots__ = ("x", "y")

p = Point()
p.x, p.y = 1, 2
p.z = 3      # whoops
p.__dict__   # {"z": 3}
```

```
tracemalloc.start()
_ = [Point(1, 2, 3)
    for _ in range(1_000_000)]
get_traced_memory()  # 416 MB
```

`__slots__` memory savings **broken**



Preventing broken slots

```
class Base:
    __slots__ = ()

class Point(Base):
    __slots__ = ("x", "y", "z")
```

Keeping your `__slots__` in check

Slots checking—simple cases

```
class Point:
    __slots__ = ("x", "y")

class Point3D(Point):
    __slots__ = ("z",)
```

Slots checking—inheritance

```
from foo import *

def __getattr__(name):
    ...

try:
    from place import Base
except ImportError:
    from other_place import Base
```

Slots checking—dynamic slots

```
class Point(metaclass=B):
    ...

@dataclass(slots=True)
class Point:
    ...

class Point(pydantic.BaseModel):
    ...
```

Slots checking—just import!

```
from mymod import Point

Point.__slots__
Point.__mro__

Point.__static_attributes__

Point.__dict__["__slots__"]
```

Slotscheck

```
$ pip install slotscheck
$ slotscheck -m sanic
ERR: 'app:Sanic' has overlapping slots.
     - name (sanic.base.root:BaseSanic)
ERR: 'sanic.response:HTTPResponse's base needs slots
     - sanic.response:BaseHTTPResponse
Oh no, found some problems!
Scanned 72 module(s), 111 class(es).
```

Looking ahead

- Closing the `__dict__` gap
- `dataclass(slots=True)`
- Advanced static analysis
- LLMs

When to use `__slots__`?

1. Untyped code
2. Python 3.10 or earlier
3. Low-hanging fruit

More information

- By me
  - **slotscheck: https://github.com/ariebovenberg/slotscheck**
  - benchmarks: https://github.com/ariebovenberg/slots-bench
  - slides: https://github.com/ariebovenberg/europython2025-slots-talk
  - original blogpost: https://dev.arie.bovenberg.net/blog/finding-broken-slots-in-popular-python-libraries
- Other resources
  - official docs: https://docs.python.org/3/reference/datamodel.html#slots
  - optimizations in 3.11+: https://github.com/python/cpython/blob/main/Objects/object_layout.md
  - are `__slots__` still worth it? https://github.com/python/cpython/issues/136016
  - addressing broken `__slots__` : https://github.com/python/cpython/issues/135385
  - key-sharing instance dicts: https://peps.python.org/pep-0412