# Extending Cython With GIL-free Types

| | Past | Present | Future |
|---|---|---|---|
| **ERP5 / SlapOS / Wendelin** | Python | Python | Cython+ |
| **Instagram** | Python | Python | Python |
| **Dropbox** | Python | Go | Go |
| **OpenSVC** | Python | Python | Go |
| **CleverCloud** | | Rust | Rust |
| **Kubernetes** | | Go | Go |
| **Scikit-learn** | Cython | Cython | Cython |

# What is Cython ?

```
a = 2
b = 3
c = a + b
```

```
PyObject * a, b, c;
a = PyInt_FromLong(2);
b = PyInt_FromLong(3);
c = PyNumber_Add(a, b);
```

```
cdef int a, b, c
```

```
cdef int a, b, c
a = 2
b = 3
c = a + b
```

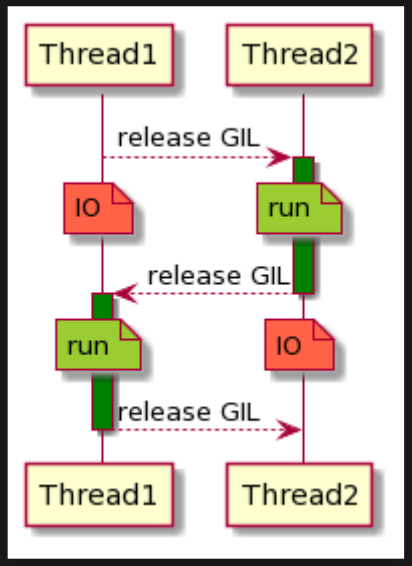
```
int a, b, c;
a = 2;
b = 3;
c = (a + b);
```

```
from libc.stdio cimport printf
printf("Hello World !\n")
```

```c
#include <stdio.h>
printf((char const *)"Hello World !\n");
```

```
>>> import helloworld
Hello Word !
>>>
```

# Bypassing the GIL with Cython


Thread1
Thread2
release GIL
IO
run
release GIL
run
IO
release GIL
Thread1
Thread2

```
typedef struct _object {
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;
    PyTypeObject *ob_type;
} PyObject;
```

```
with nogil:
    # the GIL is released here
```

```
with nogil:
    # ERROR!
    a = 2
    b = 3
    c = a + b
```

```
cdef int a, b, c
with nogil:
    # OK
    a = 2
    b = 3
    c = a + b
```

```
with nogil:
    # ERROR!
    l = [1, 2]
    l.append(3)
```

```
cdef int l[3]

with nogil:
    # OK
    l[0] = 1
    l[1] = 2
    l[2] = 3
```

```
with nogil:
    # ERROR!
    with open('somefile') as f:
        f.read()
```

```
from libc.stdio cimport fopen, fclose, fread, FILE

cdef unsigned char buf[1024]
cdef FILE * f
cdef int size

with nogil:
    # OK
    f = fopen('somefile.txt', 'r')
    size = fread(buf, 1, 1024, f)
    fclose(f)
```

| | Object Oriented | Memory Managed | Thread-Safe | Fast | Multi-Core |
|---|---|---|---|---|---|
| `PyObject` | yes | yes | yes | no | no |
| C | no | no | no | yes | yes |
| C++ | yes | no | no | yes | yes |

# Let’s hack a bit in the Cython compiler

```
cdef cypclass Adder:
    int value

    __init__(self, int value):
        self.value = value

    int add(self, int v):
        self.value += v
        return self.value
```

```
struct CyObject {
    std::atomic_int nogil_ob_refcnt;
    /* ... */
};

struct Adder : CyObject {
    int value;
    void __init__(int);
    int add(int);
    /* ... */
};
```

```
with nogil:
    a = Adder(1)
```

# Making `CyObject` Python-compatible

```
>>> from snakes import Adder
>>> a = Adder(2)
>>> repr(a)
'<snakes.Adder object at 0x2940c98>'
>>> a.value
2
>>> a.add(3)
5
>>>
```

```
typedef struct _object {
    Py_ssize_t ob_refcnt;
    /* ... */
} PyObject;

struct CyObject : PyObject {
    std::atomic_int nogil_ob_refcnt;
    /* ... */
};
```

```
void tp_dealloc_Adder(PyObject *o) {
    /* A normal PyObject would free the memory here */
    CyObject * c = static_cast<CyObject *>(o);
    Cy_DECREF(c);
}
```

# Making `CyObject` thread-safe

# Reference Capabilities

Castegren, E. (2018).
Capability-Based Type Systems for Concurrency Control.

| Object State | Type System Guarantee |
|---|---|
| *thread-local* | all references live in the same thread |
| *immutable* | all references can only read |
| *locked* | all references share a lock |
| *active* | all references share a message queue |
| *isolated* | only reachable through one reference |

```
a = Adder(1) # thread-local

# a cannot be shared
# with another thread
```

```
a = Adder(1) # thread-local

# ERROR!
cdef lock Adder b = a

# an object cannot be seen as
# thread-local and locked
# at the same time
```

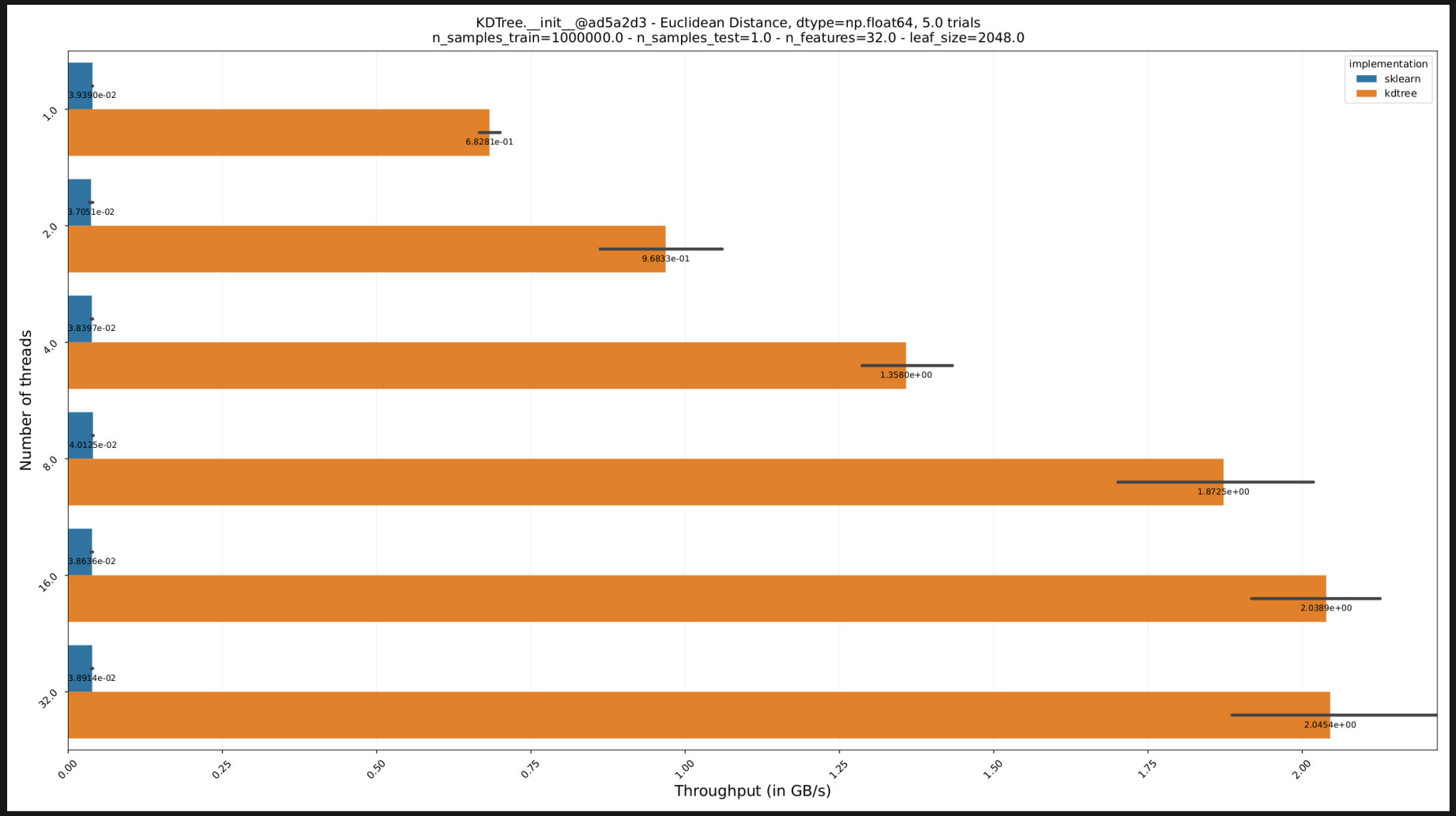
```
a = Adder(1) # thread-local

# OK if a is isolated
cdef lock Adder b = consume a

# a is now out of scope
# b can be safely shared with other threads
```

| | **Performance** | **Productivity** |
|---|---|---|
| Single-Thread | zero overhead | zero headache |
| Multi-Thread | minimal cost | no weird bugs |

| | Object Oriented | Memory Managed | Thread-Safe | Fast | Multi-Core |
|---|---|---|---|---|---|
| `PyObject` | yes | yes | yes | no | no |
| `C` | no | no | no | yes | yes |
| `C++` | yes | no | no | yes | yes |
| `CyObject` | **yes** | **yes** | **yes** | **yes** | **yes** |


KDTree.__init__@ad5a2d3 - Euclidean Distance, dtype=np.float64, 5.0 trials
n_samples_train=1000000.0 - n_samples_test=1.0 - n_features=32.0 - leaf_size=2048.0
implementation
sklearn
kdtree
Number of threads
1.0
2.0
4.0
8.0
16.0
32.0
3.9390e-02
6.8281e-01
3.7051e-02
9.6833e-01
3.8397e-02
1.3580e+00
4.0125e-02
1.8725e+00
3.8636e-02
2.0389e+00
3.8914e-02
2.0454e+00
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
Throughput (in GB/s)

# Thank You !

- Gwenaël Samain
- Boxiang Sun
- Bryton Lacquement
- Julien Muchembled
- Stéphane Fermigier
- Gaël Varoquaux
- Gilles Polart-Donat
- Julien Jerphanion
- Olivier Grisel
- François Gagnard
- Xiaowu Zhang
- Thomas Gambier

# Questions ?

More at https://cython.plus