ADVENTURES IN COMPATIBILITY:

EMULATING CPYTHON'S C API IN PYPY

Ronan Lamy

ABOUT ME

- PyPy core dev
- Python consultant and freelance developer
- Contact:
    - Ronan.Lamy@gmail.com
    - @ronanlamy

PLAN

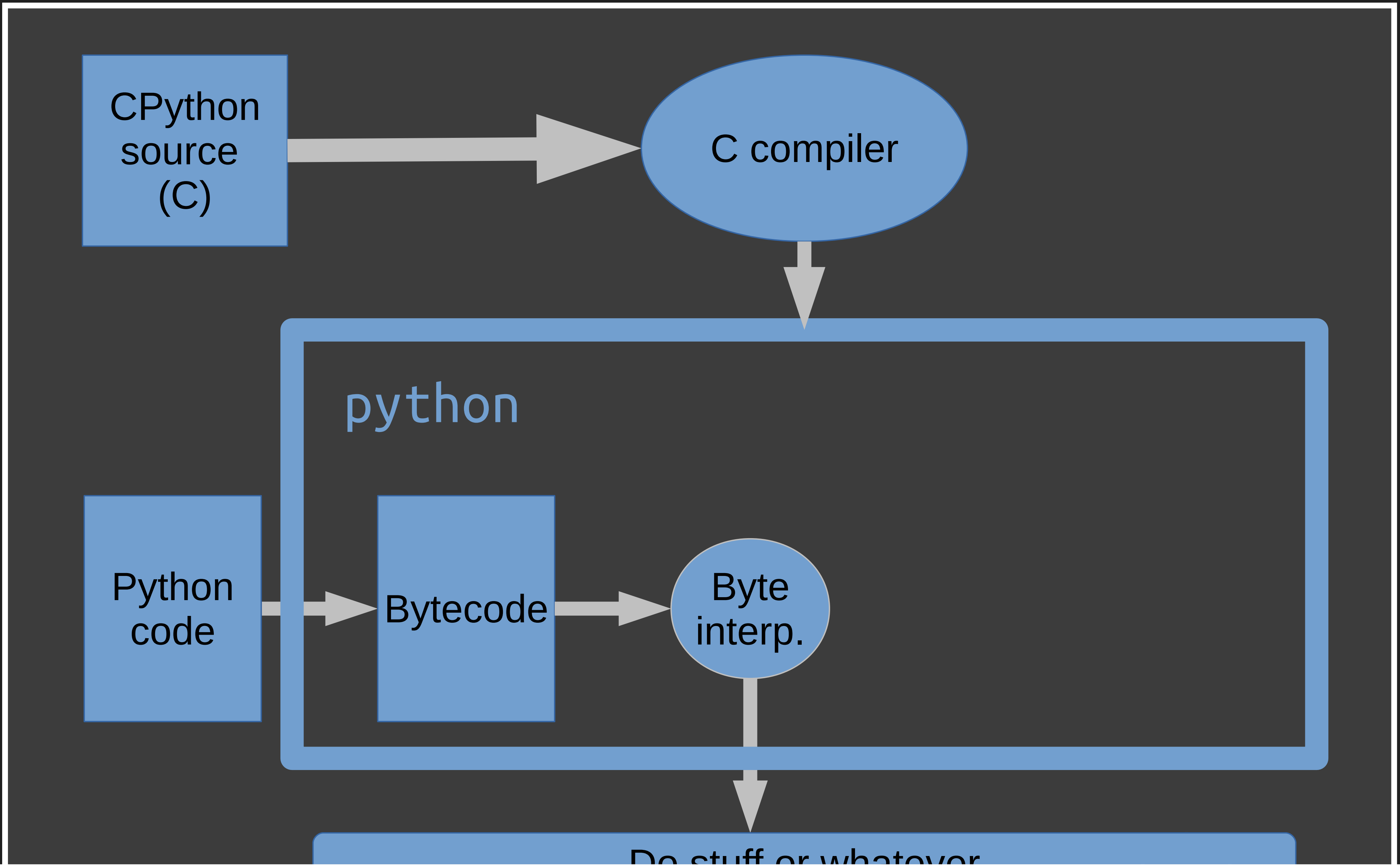
- PyPy introduction
  - Current status
- cpyext

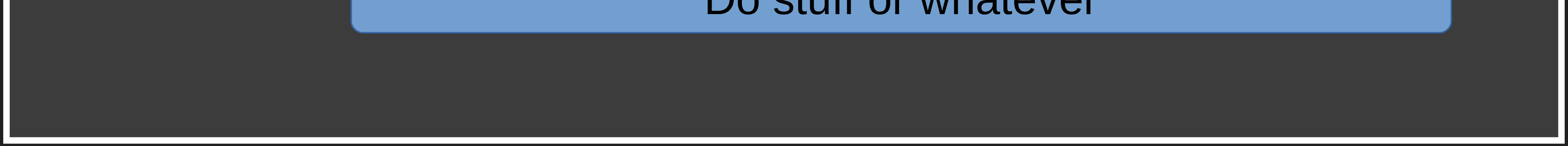
ABOUT PYPY

"PyPy is a fast, compliant alternative implementation of the Python language"

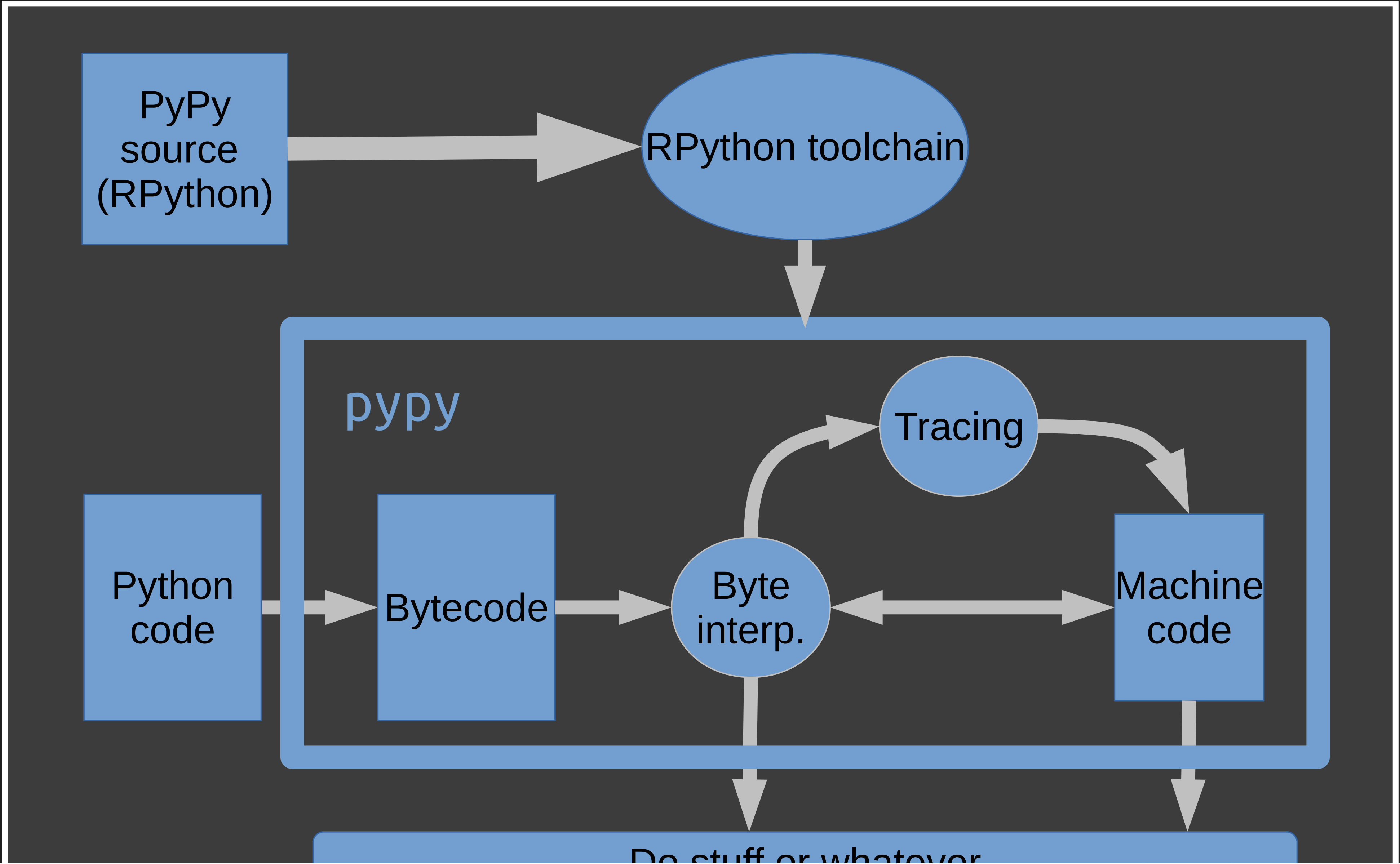
http://pypy.org

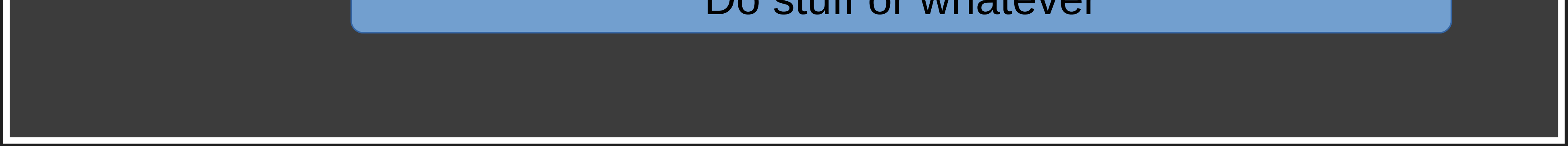
CPYTHON ARCHITECTURE




Do stuff or whatever

PYPY ARCHITECTURE




Do stuff or whatever

MEMORY MANAGEMENT

- CPython (< 3.6)
  - malloc
  - Deallocation is deterministic (except when it isn't)
  - Needs refcounting

- PyPy
  - incminimark
  - Deallocation happens eventually

OPTIMISING FOR PYPY

- Benchmark
- Profile (use vmprof)

OPTIMISING FOR PYPY

- Benchmark
- Profile (use vmprof)
- Performance tips:
  - Aim for mostly-static types
  - Function calls are ~free
  - Attribute access faster than dict indexing
  - Homogeneous lists

PYPY STATUS

- Python 3
  - PyPy3.5 v6.0 released 26 April 2018
  - still beta-quality on Windows
  - 3.6 being worked on
  - Needs more optimisations

- PyPy2.7 v6.0 released 26 April 2018
- cffi: still the best way to interface with C
- Improved C extension compatibility

  -
    ```
    pip install numpy scipy pandas
    ```
  - Wheels available at https://github.com/antocuni/pypy-wheels

CPYEXT

OVERVIEW

- Python.h for PyPy
- Generated C headers
- Some C code (copied from CPython!)
- Translated RPython code
- **Must** compile against PyPy headers

IMPLEMENTATION IN RPYTHON

```python
@cpython_api([PyObject, Py_ssize_t, PyObject], rffi.INT_real, error=-1)
def PyList_SetItem(space, w_list, index, py_item):
    """Set the item at index index in list to item.  Return 0 on success
    or -1 on failure.

    This function "steals" a reference to item and discards a reference to
    an item already in the list at the affected position.
    """
    if not isinstance(w_list, W_ListObject):
        decref(space, py_item)
        PyErr_BadInternalCall(space)
    if index < 0 or index >= w_list.length():
        decref(space, py_item)
        raise oefmt(space.w_IndexError, "list assignment index out of range")
    storage = get_list_storage(space, w_list)
    py_old = storage._elems[index]
    storage._elems[index] = py_item
    decref(w_list.space, py_old)
    return 0
```

```
def wrapper_second_level(callable, pname, *args):
    from pypy.module.cpyext.pyobject import make_ref, from_ref, is_pyobj
    from pypy.module.cpyext.pyobject import as_pyobj
    from pypy.module.cpyext import pystate
    # we hope that malloc removal removes the newtuple() that is
    # inserted exactly here by the varargs specializer

    # see "Handling of the GIL" above (careful, we don't have the GIL here)
    tid = rthread.get_or_make_ident()
    _gil_auto = False
    if gil_auto_workaround and cpyext_glob_tid_ptr[0] != tid:
        # replace '-1' with the real tid, now that we have the tid
        if cpyext_glob_tid_ptr[0] == -1:
            cpyext_glob_tid_ptr[0] = tid
        else:
            _gil_auto = True
    if _gil_auto or gil_acquire:
        if cpyext_glob_tid_ptr[0] == tid:
            deadlock_error(pname)
        rgil.acquire()
        assert cpyext_glob_tid_ptr[0] == 0
    elif pygilstate_ensure:
        if cpyext_glob_tid_ptr[0] == tid:
            cpyext_glob_tid_ptr[0] = 0
            args += (pystate.PyGILState_LOCKED,)
        else:
            rgil.acquire()
```

ISSUES

	PyPy	C extension
Language	RPython	C
Objects	W_Root	PyObject
Memory	Managed, moving	Pointers
Exceptions	Yes	Error indicator
Refcounts	No	Yes

SOLUTIONS

- Link PyObjects to W_Root

```
#define PyObject_HEAD  \
    Py_ssize_t ob_refcnt;        \
    Py_ssize_t ob_pypy_link;     \
    struct _typeobject *ob_type;
```

SOLUTIONS

- Link PyObjects to W_Root

```
#define PyObject_HEAD  \
    Py_ssize_t ob_refcnt;          \
    Py_ssize_t ob_pypy_link;       \
    struct _typeobject *ob_type;
```

SOLUTIONS

- Link PyObjects to W_Root

```
#define PyObject_HEAD  \
    Py_ssize_t ob_refcnt;         \
    Py_ssize_t ob_pypy_link;      \
    struct _typeobject *ob_type;
```

- Use GC to manage the link

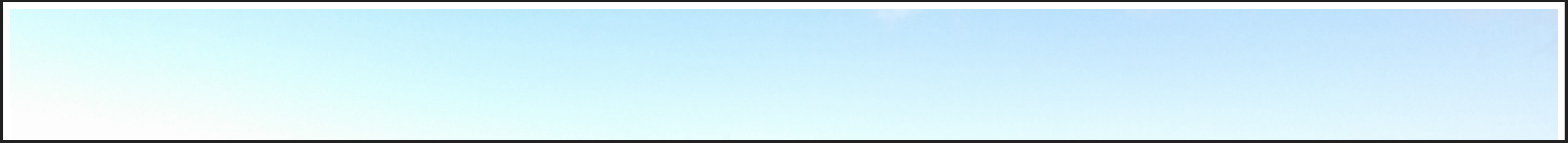
SOLUTIONS

- Link PyObjects to W_Root

```
#define PyObject_HEAD  \
    Py_ssize_t ob_refcnt;         \
    Py_ssize_t ob_pypy_link;      \
    struct _typeobject *ob_type;
```

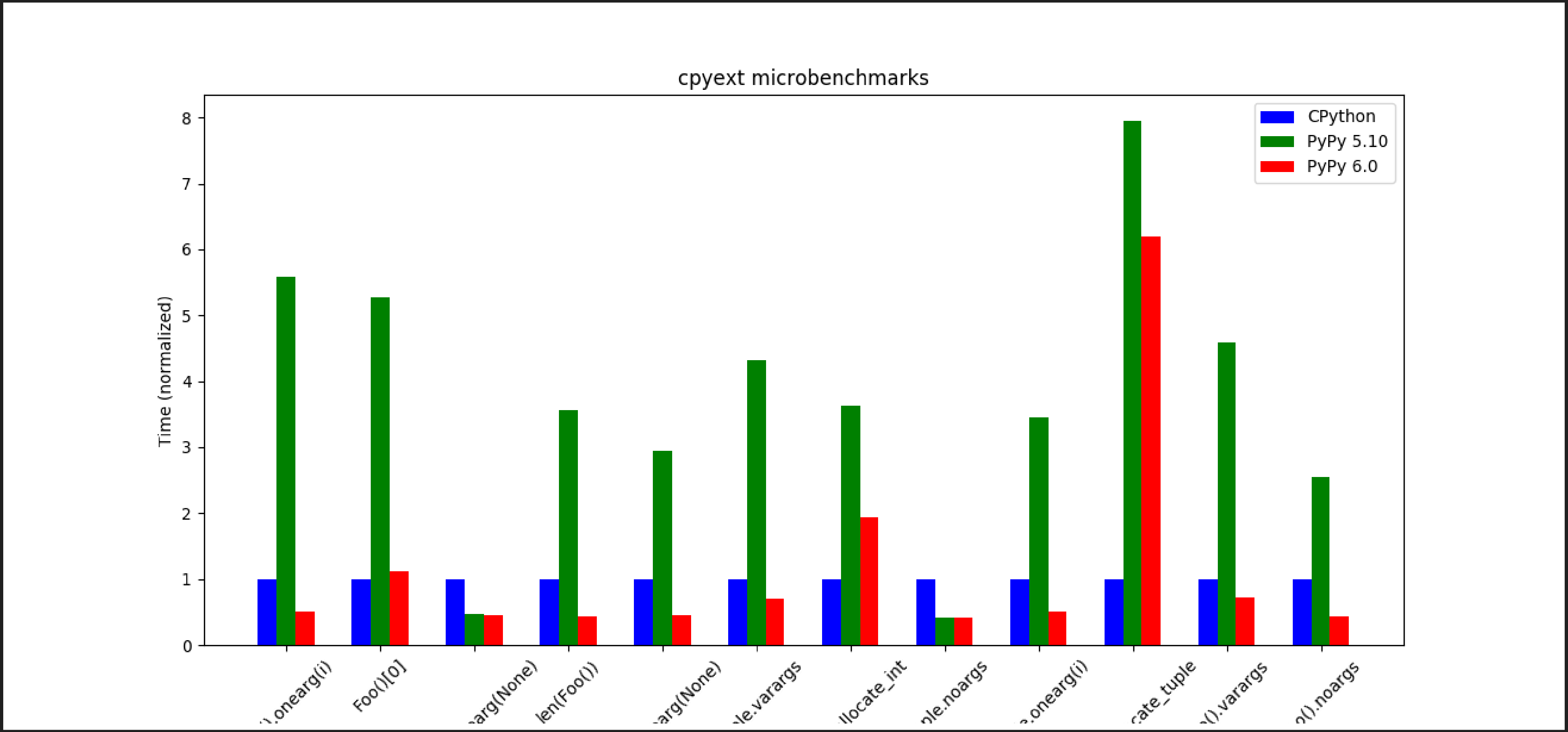
- Use GC to manage the link
- "All problems in computer science can be solved by another level of indirection"

CAPE TOWN OPTIMISATIONS



- Crossing the boundary is expensive
- Magic decorators make it easy
- Be more careful!
- Rewrite in C

Results


cpyext microbenchmarks
Time (normalized)
0
1
2
3
4
5
6
7
8
CPython
PyPy 5.10
PyPy 6.0
).oneargs(i)
Foo()[0]
arg(None)
len(Foo())
arg(None)
le.varargs
llocate_int
ple.noargs
e.onearg(i)
cate_tuple
o().varargs
o().noargs

WHAT NEXT?

- More optimisations
- PyPy open space this afternoon 14:00
- Questions?

THE END