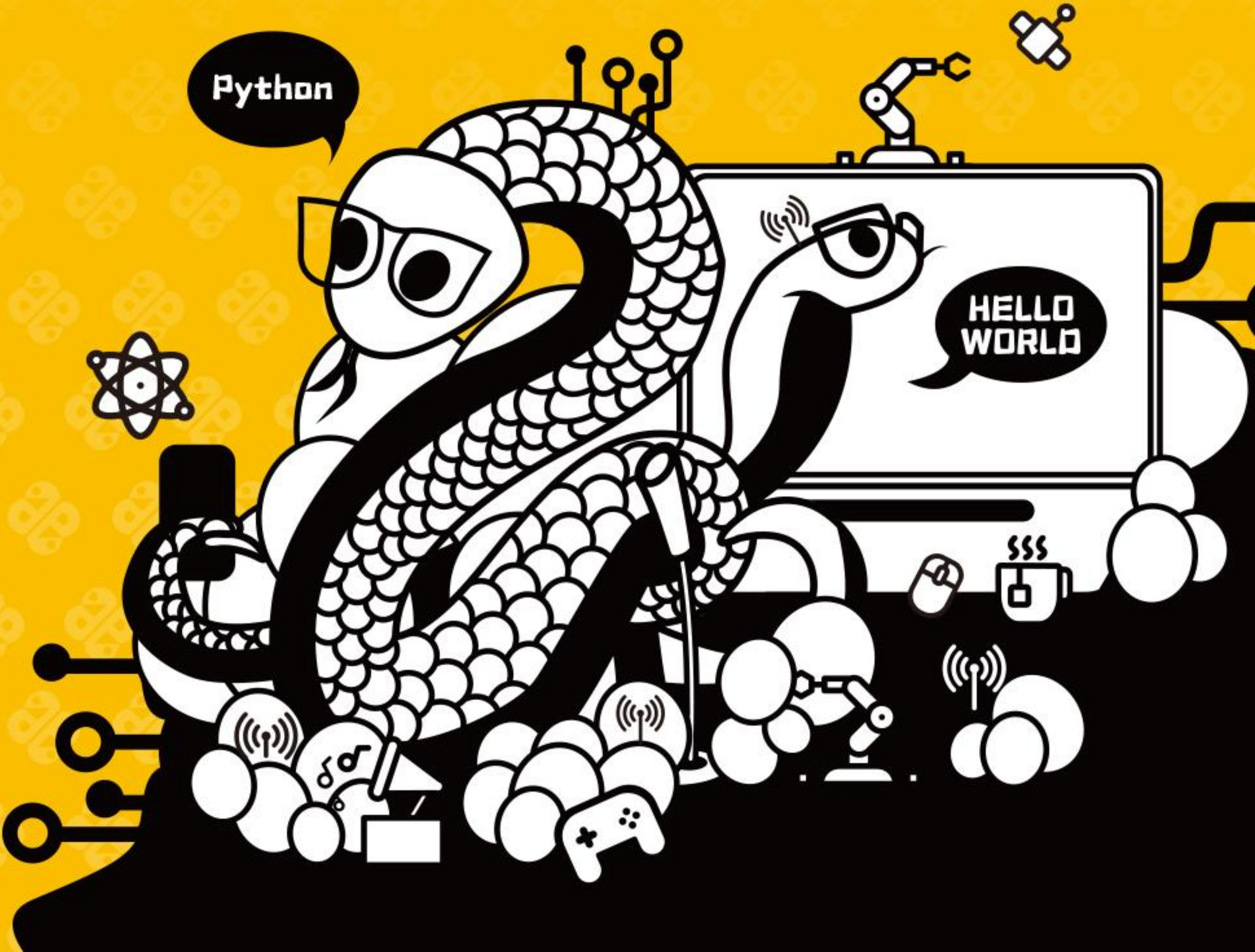
python × PYCHINA Python 中国社区

Python for Good

PyCon China 2022

大规模生产环境下的 Faster-CPython

主讲人： 王文洋



Why Faster?

老板思维

已知：公司有xx个计算集群
每个集群有xxxxx个core
Python进程占比xx%

如果：提升 10%
那么：可以节省 xx * xxxxx * xx% * 10%个core
降本 xx * xxxxx * xx% * 10% * n >> 我的工资

结论：。。。

Why CPython?

Cython

Numba

TorchDynamo

cinder

Pyston

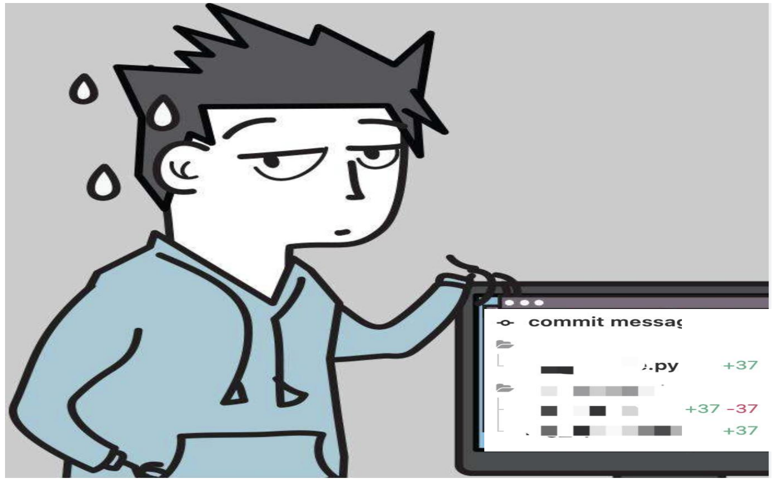
pypy

Why CPython?

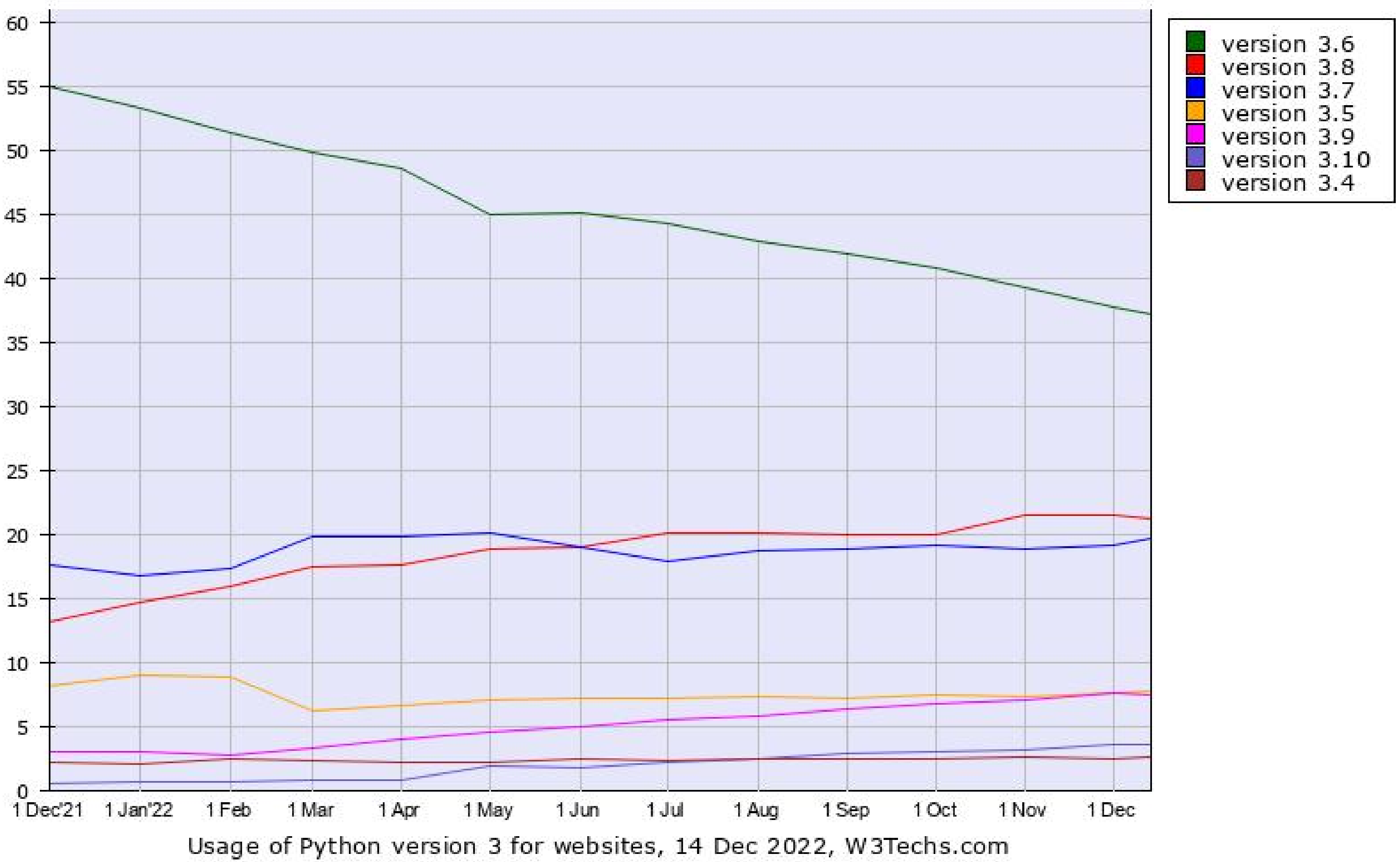
项目的存量代码

```
╭─ ~/code/ ──────────── ✔ | base Py | at 23:50:57
╰─ git clone https://xxxxxxxxxxxxxxxxxxx project
Cloning into 'project'...
remote: Enumerating objects: 5040402, done.
remote: Counting objects: 100% (2810/2810), done.
remote: Compressing objects: 100% (335/335), done.
Receiving objects:   1% (85157/5040402), 28.10 MiB | 1.88 MiB/s
```

老板每天看到的PR



Why CPython?


60
55
50
45
40
35
30
25
20
15
10
5
0
1 Dec'21
1 Jan'22
1 Feb
1 Mar
1 Apr
1 May
1 Jun
1 Jul
1 Aug
1 Sep
1 Oct
1 Nov
1 Dec
version 3.6
version 3.8
version 3.7
version 3.5
version 3.9
version 3.10
version 3.4
Usage of Python version 3 for websites, 14 Dec 2022, W3Techs.com

Why CPython?

居然要写代码 ✖

居然要动环境 ✖

Implementation plan for speeding up CPython

Overview

The overall aim is to speed up CPython by a factor of (approximately) five. We aim to do this in four distinct stages, each stage increasing the speed of CPython by (approximately) 50%.

1.5**4 ≈ 5

Each stage will be targetted at a separate release of CPython. A faster schedule is possible, but we believe that predictable and reliable performance improvements are more important than squeezing out the maximum performance for each release. Of course delays in software development are all too common, so a release might need to be skipped.

https://github.com/faster-cpython

Faster CPython

pyperformance:

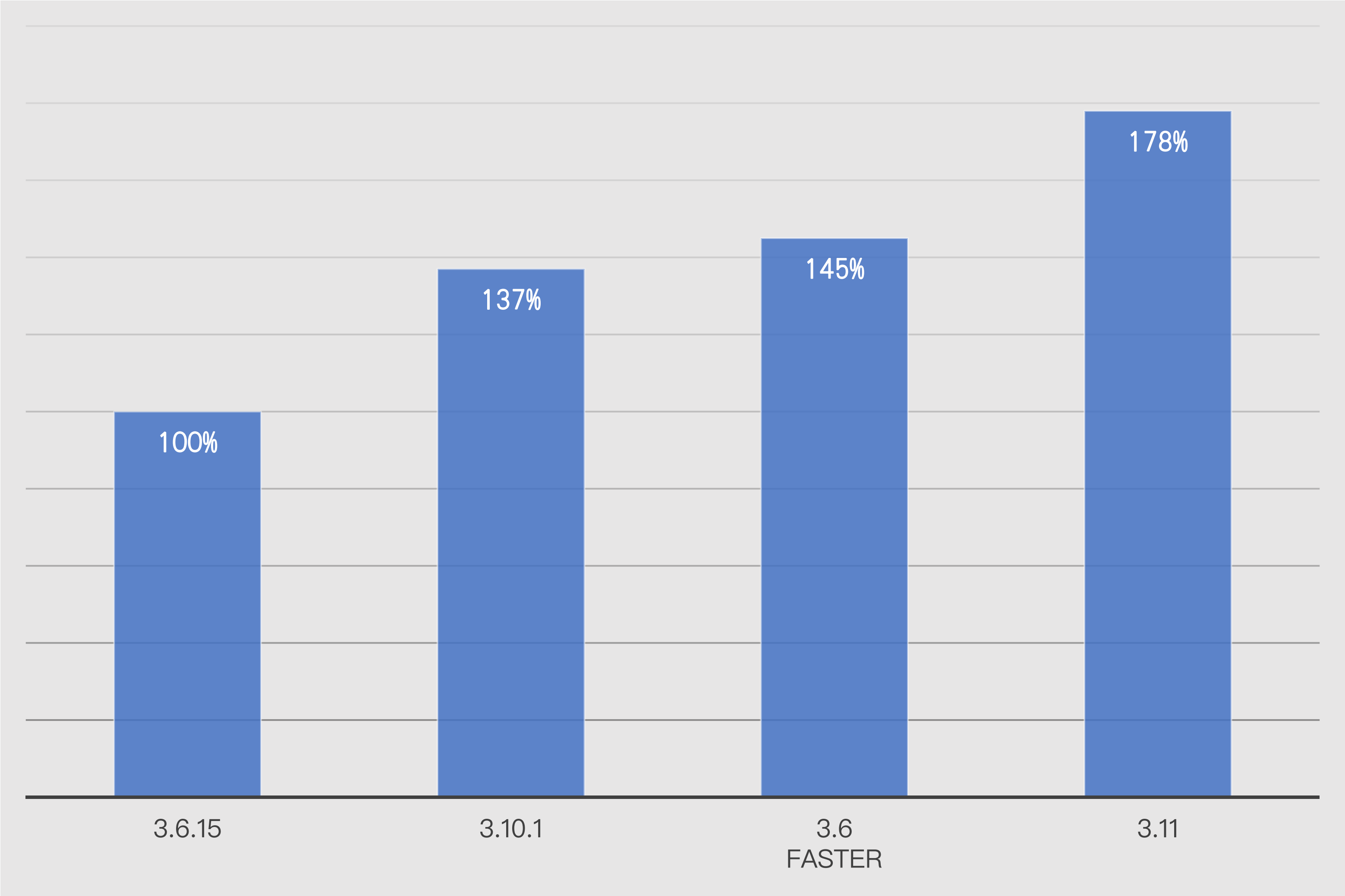
date	release	commit	host	mean
2022-06-07 (20:12 UTC)	cpython 3.10.4	9d38120e33	fc_linux	(ref)
2022-06-06 (22:23 UTC)	cpython 3.11.0b3	eb0004c271	fc_linux	1.28x faster
2022-11-20 (02:15 UTC)	cpython 3.12.0a0	b0e1f9c241	fc_linux	1.31x faster

https://github.com/faster-cpython

How To Faster In Older Versions

git cherry-pick ✖

Faster CPython In 3.6

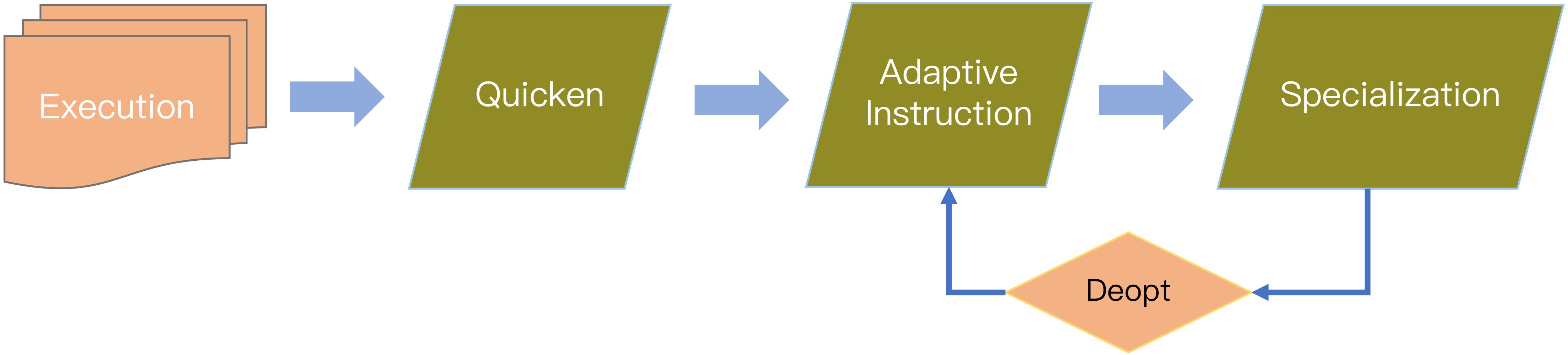


How To Faster

3.11 Faster Runtime

- Cheaper, lazy Python frames
- Inlined Python function calls
- **PEP 659: Specializing Adaptive Interpreter**

动态语言的虚拟机可以根据执行中出现过的类型和值对代码进行特化，以提高运行效率。这种特化通常与 “JIT ” 编译器联系在一起。但即使没有编译成机器代码，这种优化也是有益的。



PEP 659: Specializing Adaptive Interpreter

Python for Good
PyCon China 2022

```
def get_url(path):
    return "https://example.com" + path
print(dis.dis(get_url, adaptive=True))

for _ in range(8):
    get_url("/host") # warmup

get_url ("/host") # quicken + specialize
print(dis.dis(get_url, adaptive=True))
```

```
3        0 RESUME                 0

4        2 LOAD_CONST             1 ('https://example.com')
         4 LOAD_FAST              0 (path)
         6 BINARY_OP              0 (+)
        10 RETURN_VALUE
```

Quicken

```
3        0 RESUME_QUICK                 0

4        2 LOAD_CONST__LOAD_FAST        1 ('https://example.com')
         4 LOAD_FAST                    0 (path)
         6 BINARY_OP_ADAPTIVE           0 (+)
        10 RETURN_VALUE
```

Specialize: _Py_Specialize_BinaryOp

```
3        0 RESUME_QUICK                 0

4        2 LOAD_CONST__LOAD_FAST        1 ('https://example.com')
         4 LOAD_FAST                    0 (path)
         6 BINARY_OP_ADD_UNICODE        0 (+)
        10 RETURN_VALUE
```

Faster CPython

./configure --enable-pystats // 增加编译参数

Execution counts

▾ execution counts for all instructions

Name	Count	Self	Cumulative	Miss ratio
LOAD_FAST	14,357,152,597	14.7%	14.7%	
LOAD_FAST__LOAD_FAST	4,828,760,519	4.9%	19.7%	
LOAD_CONST	4,552,140,570	4.7%	24.3%	
RESUME	4,222,371,364	4.3%	28.7%	
STORE_FAST__LOAD_FAST	3,756,594,965	3.8%	32.5%	
POP_JUMP_IF_FALSE	3,667,430,041	3.8%	36.3%	
LOAD_GLOBAL_BUILTIN	3,494,419,938	3.6%	39.8%	0.0%
RETURN_VALUE	3,448,108,732	3.5%	43.4%	

BINARY_OP

▾ specialization stats for BINARY_OP family

Kind	Count	Ratio
specialization.deferred	849557914	19.8%
specialization.deopt	711020	0.0%
hit	3398293453	79.3%
miss	37685040	0.9%

Specialization attempts

	Count	Ratio
Success	714,799	39.2%
Failure	1,109,142	60.8%

Failure kind	Count	Ratio
subtract different types	579,285	52.2%
multiply different types	172,590	15.6%
add different types	152,450	13.7%

Overhead Analysis For CPython

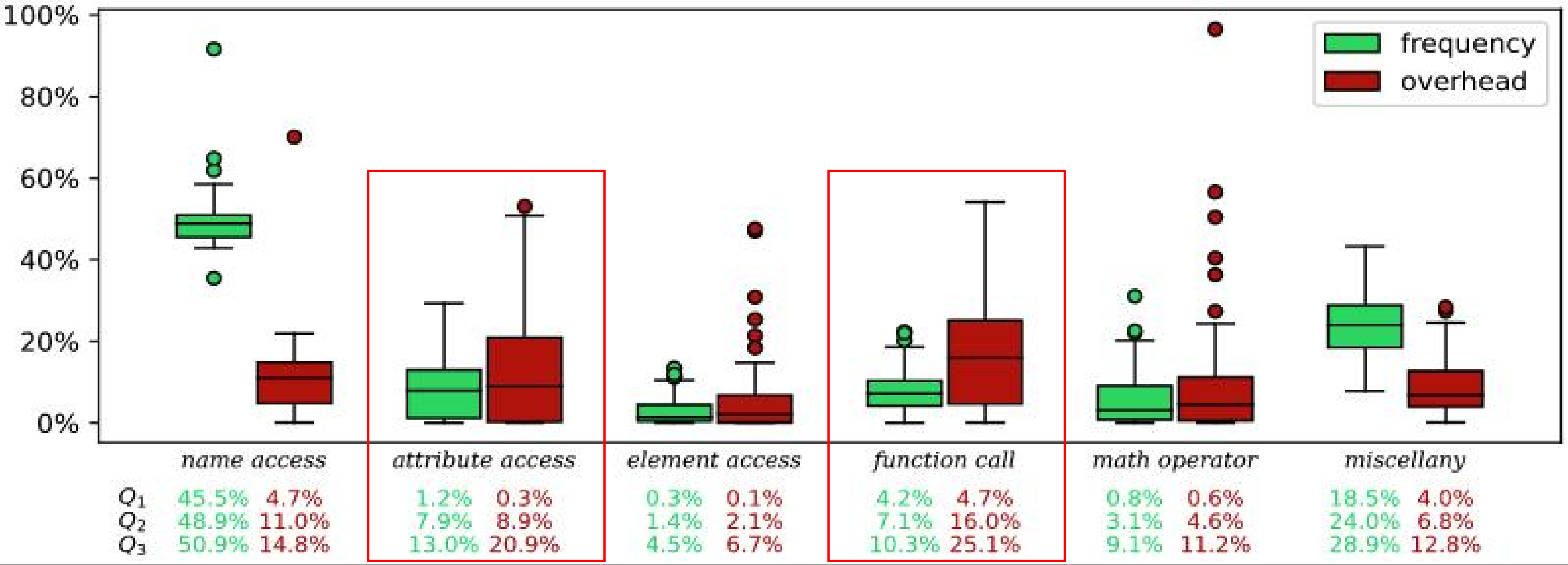
Which opcodes should be specialised?

Sources of performance overhead in CPython?

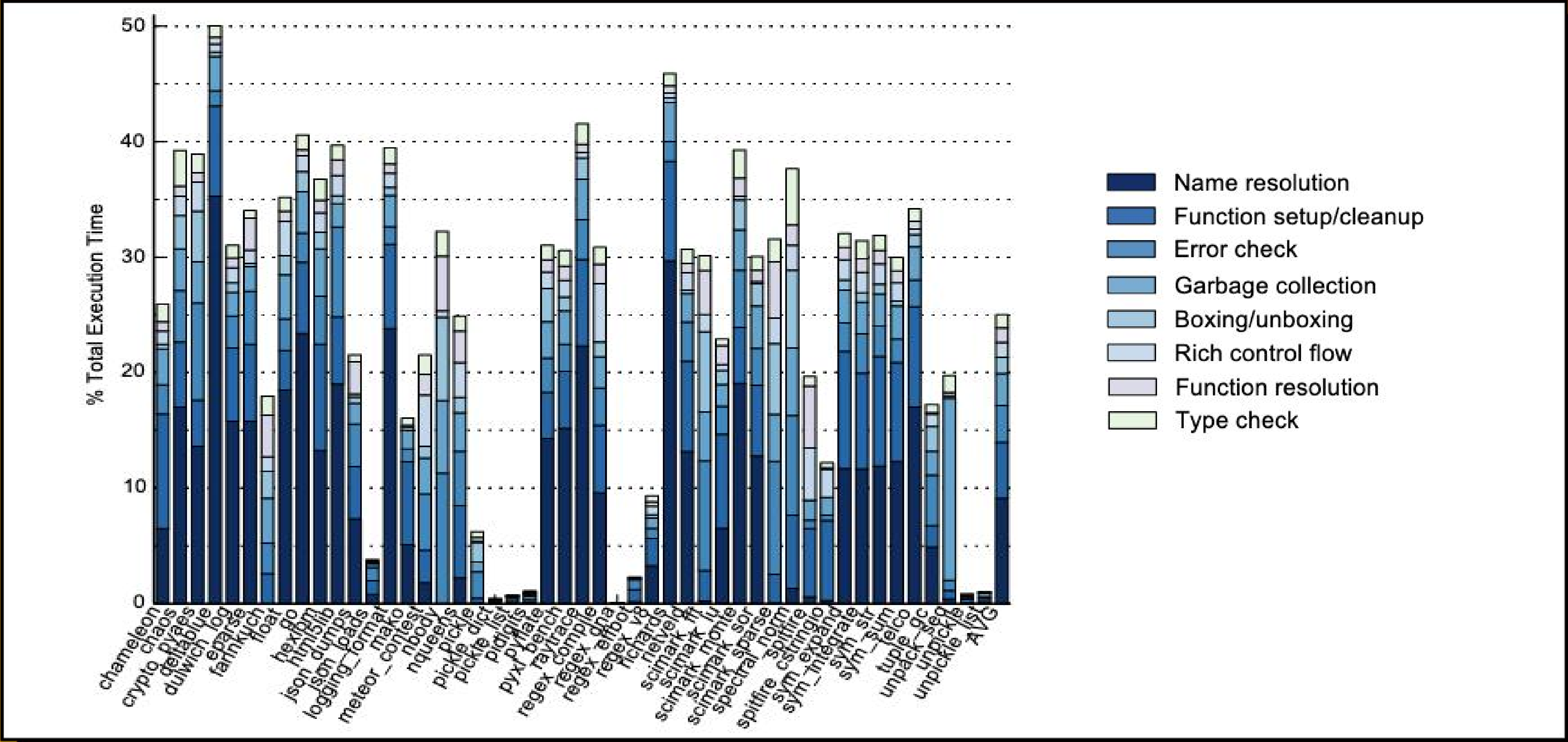
- Mohamed Ismail and G Edward Suh. 2018. ***Quantitative overhead analysis for Python***. In 2018 IEEE International Symposium on Workload Characterization (IISWC).
- Qiang Zhang, Lei Xu, Xiangyu Zhang, and Baowen Xu. 2022. ***Quantifying the interpretation overhead of Python***. Science of Computer Programming 215 (2022).

Overhead Analysis For CPython – opcodes execution

	Descriptions	Opcodes
name access	Access the values corresponding to the variable names	LOAD_FAST、LOAD_GLOBAL
attribute access	Access the object attributes or methods	LOAD_ATTR、STORE_ATTR
element access	Access the container elements, e.g dict['name']	BINARY_SUBSCR、STORE_SUBSCR
function call	Make the function calls	CALL_FUNCTION
math operator	Unary, comparison, binary, or inplace operations	BINARY_ADD、COMPARE_OP
miscellany	Others	POP_TOP、END_FINALLY

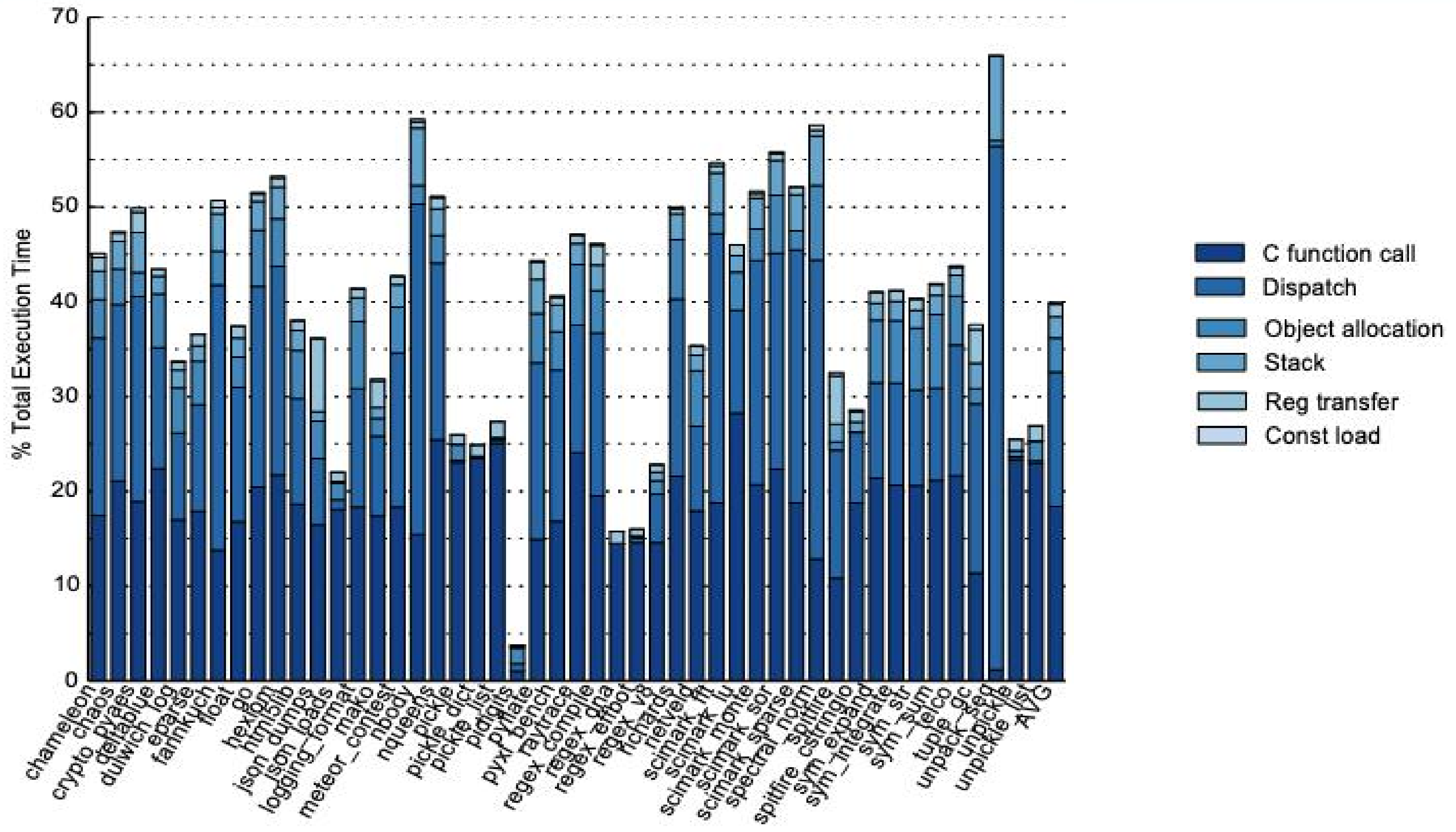


Overhead Analysis For CPython – language features



Overhead Analysis For CPython – interpreter operations

Python for Good
PyCon China 2022



How To Faster

- **Attributes(dictionary) caching**
- **Speeding up function calls**
- **Unboxing of numbers and static dispatch of arithmetic operations**
- ...
- Removes unnecessary reference count operations
- Zero-cost exception handling
- Better GC
- ...

Generic Get Attribute


```
PyAPI_FUNC(PyObject *)
_PyObject_GenericGetAttrWithDict(PyObject *obj, PyObject *name,...);
```

descriptor

```
for cls_ty in tp_mro:
```

```
dict = cls_ty->tp_dict;
descr = PyDict_GetItem(dict, name);
res = descr->ob_type->tp_descr_get(...)
```

instance value

```
dict_ptr = _PyObject_GetDictPtr(obj);
res = PyDict_GetItem(dict_ptr, name);
```

Generic Get Attribute


```
PyDictEntry *ma_table :
[
ep ➡    { me_hash, me_key, me_value },
        { null, null, null },
        { me_hash, me_key, me_value },
        { null, null, null},
        { ... },
]
```

```
PyDictEntry *lookdict(PyDictObject *mp, PyObject *key, Py_hash_t hash)
{
    ep0 = mp->ma_table;
    perturb = hash ;

    i = perturb & mask;

    ep = &ep0[i];

    ep->me_key == null

    perturb >>= PERTURB_SHIFT;

    i = ((i << 2) + i + perturb + 1) & mask;

    return ep
}
```

Attribute(Dictionary) Caching

```c
TARGET(LOAD_GLOBAL_MODULE) {
    ...
    PyDictObject *dict = (PyDictObject *)GLOBALS();
    _PyLoadGlobalCache *cache = (_PyLoadGlobalCache *)next_instr;
    uint32_t version = read_u32(cache->module_keys_version);
    DEOPT_IF(dict->ma_keys->dk_version != version, LOAD_GLOBAL);
    assert(DK_IS_UNICODE(dict->ma_keys));
    PyDictUnicodeEntry *entries = DK_UNICODE_ENTRIES(dict->ma_keys);
    PyObject *res = entries[cache->index].me_value;
    ...
    DISPATCH();
}
```

cache

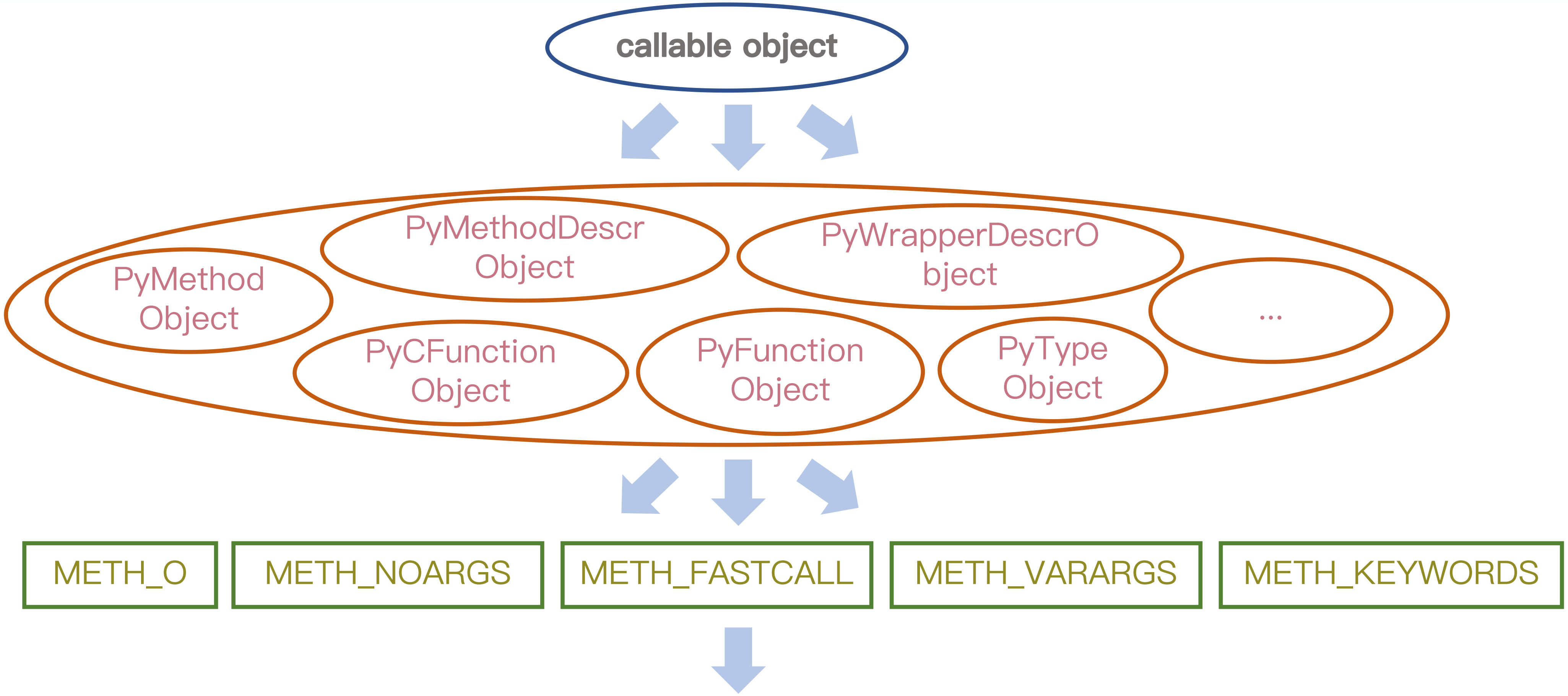
LOAD_GLOBAL_MODULE	0
counter	
index	
module_keys_version[2]	
...	
builtin_keys_version	
LOAD_METHOD	1
...	

Attribute(Dictionary) Caching


```
class Base:

    def get(self):
        pass


class Config(Base):
    ...


class Table(Base):
    ...


def load_method_loop(objs):
    for obj in objs:
        obj.get()
```

```
args = [Config()] * 20000000
start = time.perf_counter()
load_method_loop(args)
print(time.perf_counter() – start) # hit: 0.639s

args = [Config()] * 10000000 + [Table()] * 10000000
start = time.perf_counter()
load_method_loop(args)
print(time.perf_counter() – start) # deopt: 0.641s

args = [Config(), Table()] * 10000000
start = time.perf_counter()
load_method_loop(args)
print(time.perf_counter() – start) # miss: 0.821s
```

Speeding up function calls




```
res = meth(PyCFunction_GET_SELF(func), args, nargs);
or
retval = _PyEval_EvalFrame(tstate, f, 0);
```

Speeding up function calls

Python for Good
PyCon China 2022

```
getattr(sys.version_info, "major", None)
```

1 builtin_getattr bltinmodule.c:1085
2 cfunction_vectorcall_FASTCALL methodobject.c:430
3 _PyObject_VectorcallTstate abstract.h:114
4 PyObject_Vectorcall abstract.h:123
5 call_function ceval.c:5869
6 _PyEval_EvalFrameDefault ceval.c:4213

Speeding up function calls


```
TARGET(PRECALL_NO_KW_BUILTIN_FAST) {
    /* Builtin METH_FASTCALL functions, without keywords */
    ...
    PyObject *callable = PEEK(total_args + 1);
    DEOPT_IF(!PyCFunction_CheckExact(callable), PRECALL);
    DEOPT_IF(PyCFunction_GET_FLAGS(callable) !=
        METH_FASTCALL, PRECALL);
    ...
    PyCFunction cfunc = PyCFunction_GET_FUNCTION(callable);
    STACK_SHRINK(total_args);
    /* res = func(self, args, nargs) */
    PyObject *res = ((_PyCFunctionFast)(void(*)(void))cfunc)(
        PyCFunction_GET_SELF(callable),
        stack_pointer,
        total_args);
    ...
    DISPATCH();
}
```

1 builtin_getattr bltinmodule.c:1085
cfunction_vectorcall_FASTCALL methodobject.c:430
_PyObject_VectorcallTstate abstract.h:114
PyObject_Vectorcall abstract.h:123
call_function ceval.c:5869
2 _PyEval_EvalFrameDefault ceval.c:4213

Speeding up function calls

Python for Good
PyCon China 2022

```
def load_attr(version_info):
    start = time.perf_counter()
    for _ in range(100000000):
        version_info.major
    print(time.perf_counter() – start)
```

python3.10 : 3.22s

python3.11 : 2.96s

```
def call_c(version_info):
    start = time.perf_counter()
    for _ in range(100000000):
        getattr(version_info, "major", None)
    print(time.perf_counter() – start)
```

python3.10 : 5.51s

python3.11 : 3.89s

Unboxing of numbers and static dispatch of arithmetic operations

Python for Good
PyCon China 2022

```python
data = 100.0
...
data * 1.1
```

1 float_mul floatobject.c:589

2 binary_op1 abstract.c:891

3 PyNumber_Multiply abstract.c:1109

4 _PyEval_EvalFrameDefault ceval.c:2003

```c
    static PyObject *binary_op1(PyObject *v, PyObject *w, const int op_slot)
    {
863     binaryfunc slotv;
864     if (Py_TYPE(v)->tp_as_number != NULL) {
865         slotv = NB_BINOP(Py_TYPE(v)->tp_as_number, op_slot);
866     }
867     else {
868         slotv = NULL;
869     }
870
871     binaryfunc slotw;
872     if (!Py_IS_TYPE(w, Py_TYPE(v)) && Py_TYPE(w)->tp_as_number != NULL) {
873         slotw = NB_BINOP(Py_TYPE(w)->tp_as_number, op_slot);
            ...
        }
        else {
875         slotw = NULL;
        }

882     if (slotv) {
883         PyObject *x;
884         if (slotw && PyType_IsSubtype(Py_TYPE(w), Py_TYPE(v))) {
885             x = slotw(v, w);
                ...
890         }
891         x = slotv(v, w);  /* call float_mul */
            ...
        }
898     if (slotw) {
899         PyObject *x = slotw(v, w);
            ...
        }
906     Py_RETURN_NOTIMPLEMENTED;
    }
```

Unboxing of numbers and static dispatch of arithmetic operations

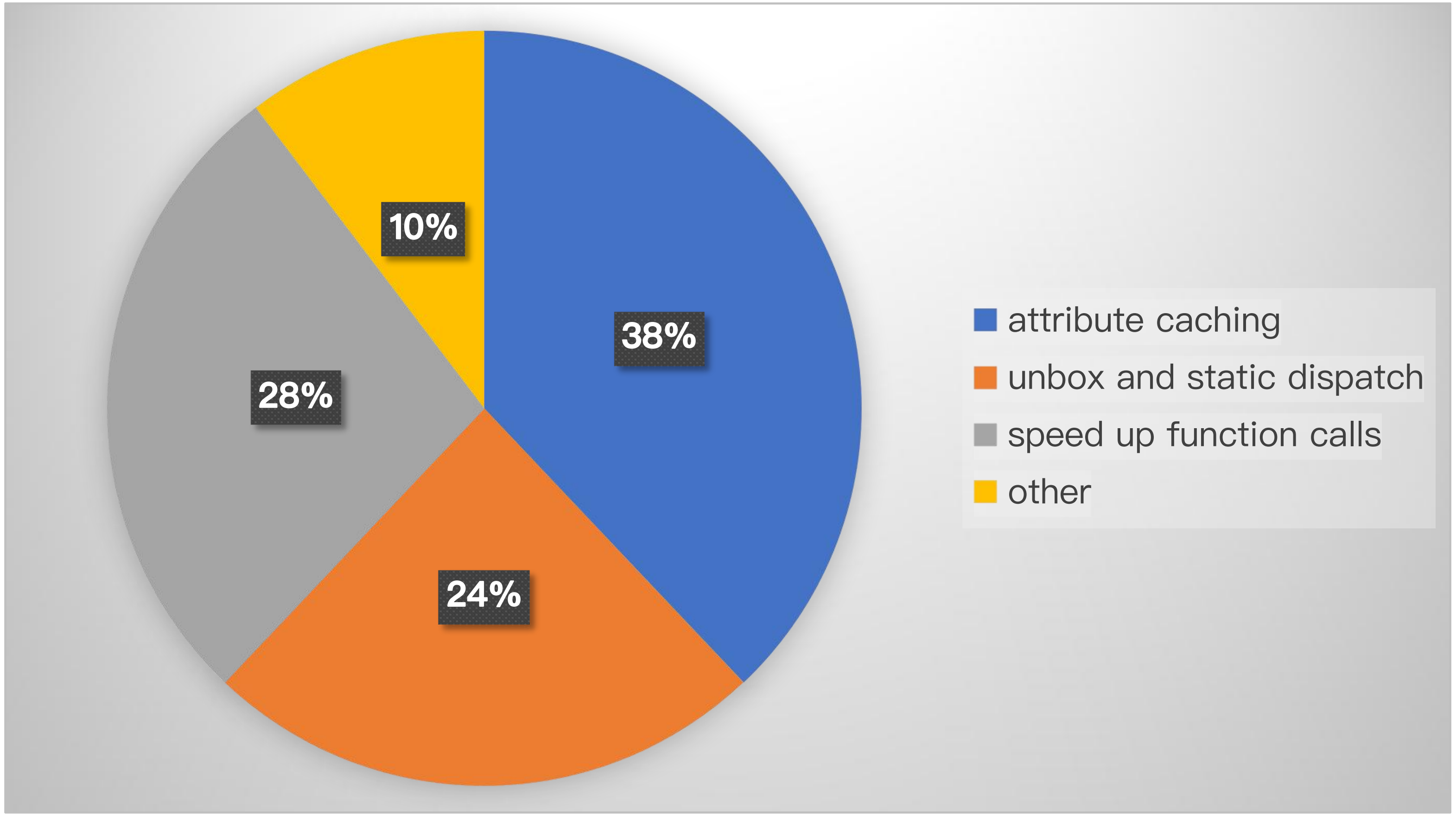

```c
TARGET(BINARY_OP_MULTIPLY_FLOAT) {
    ...
    DEOPT_IF(!PyFloat_CheckExact(left), BINARY_OP);
    DEOPT_IF(!PyFloat_CheckExact(right), BINARY_OP);
    STAT_INC(BINARY_OP, hit);
    double dprod = ((PyFloatObject *)left)->ob_fval *
        ((PyFloatObject *)right)->ob_fval;
    PyObject *prod = PyFloat_FromDouble(dprod);
    ...
    DISPATCH();
}
```

```python
def float_mul(left, right):
    start = time.perf_counter()
    for _ in range(100000000):
        res = left * right
    print(time.perf_counter() – start)
```

python3.10 : 2.86s

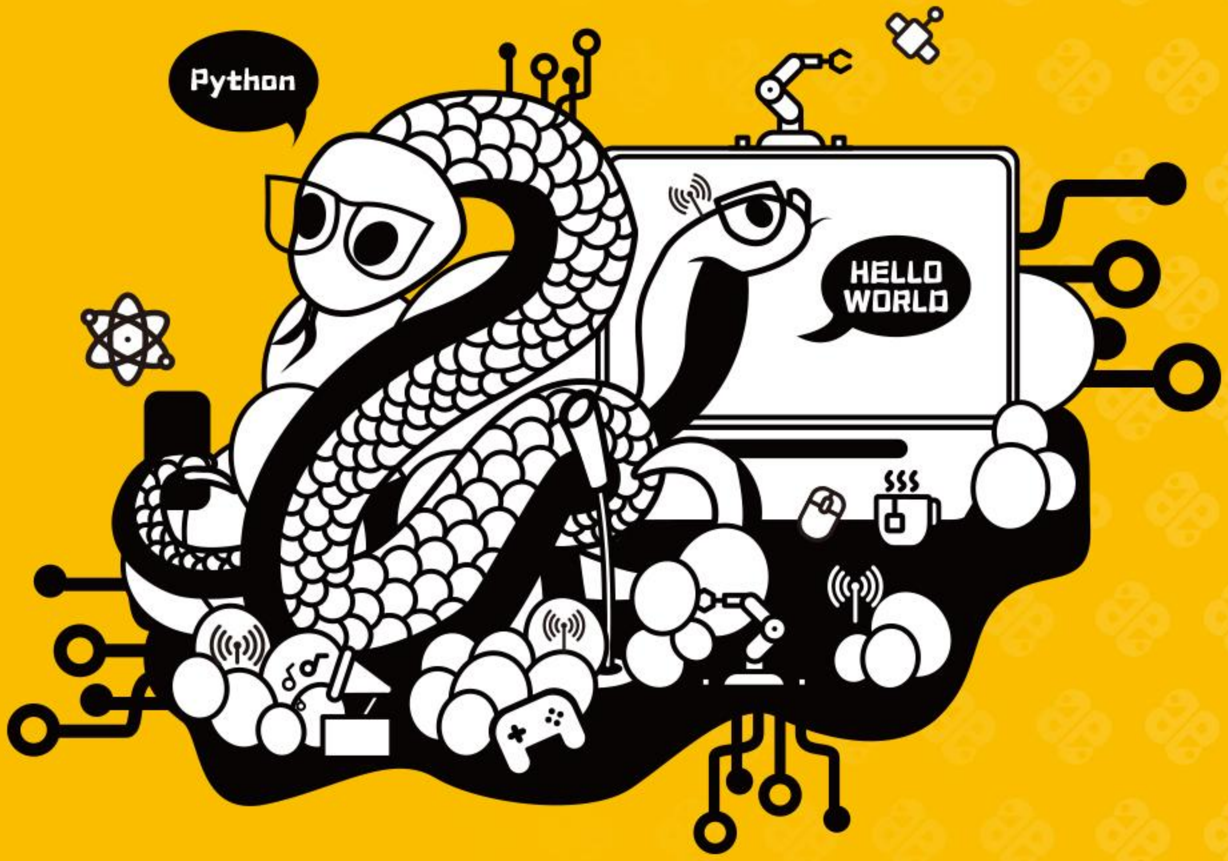
python3.11 : 2.27s

Faster CPython



How To Faster In Older Versions

Anything else?



Thanks!

感谢观看