# HPy Present & Future

Antonio Cuni
@antocuni

Python Language Summit
May 11, 2021

## 1 minute recap: what is HPy?

- New/better C API for writing extensions: `#include "hpy.h"` instead of `"Python.h"`
- The current Python/C API is full of CPython implementation details
  - HPy enables **future evolution of CPython**
- HPy: more abstract
  - Easier/faster to support on **alternative implementations**
  - **GC friendly**
  - Handles instead of `PyObject*`
  - `HPy_Dup`/`HPy_Close` instead of `Py_INCREF`/`Py_DECREF`
- **0 overhead on CPython**
- **Incremental migration**
- Much faster on PyPy and GraalPython than emulating the existing C API

## Recent achievements

- Custom types
- Debug mode
- Windows support
- setuptools integration
- Very early attempt to port numpy
  - Positive feedback from the numpy team
- Cython backend (starting soon)

## Community & funding

- Launched https://hpyproject.org and https://hpyproject.org/blog/
- Lot of interest and positive feedback from various sources
  - E.g., someone started to port pillow: https://github.com/cklein/Pillow-hpy
- Got some funding and sponsors
  - But also a lot of non-funded open source development

ORACLE IBM Quansight Labs

## CPython ABI vs Universal ABI

foo.c

```
static inline HPy
HPyLong_FromLong(HPyContext *ctx, long x) {
    return _py2h(PyLong_FromLong(x));
}
```

$\downarrow$

```
foo.cpython-38-x86_64-linux-gnu.so
PyObject *PyInit_foo(void);
```

- Zero overhead
- Compile-time only dependency
- Indistinguishable from a normal extension

```
static inline HPy
HPyLong_FromLong(HPyContext *ctx, long x) {
    return ctx->ctx_Long_FromLong(ctx, x);
}
```

$\downarrow$

```
foo.hpy.so
HPy HPyInit_foo(HPyContext *ctx);
```

- 5-10% overhead on CPython
- One binary for CPython / PyPy / GraalPython
- Runtime dependency (`hpy.universal`)
- Debug mode
- (Currently) needs import hook or `.py` shim
- Wheels?

## Debug mode (1)

```
HPyDef_METH(inc, "inc", inc_impl, HPyFunc_O);
static HPy inc_impl(HPyContext *ctx, HPy self, HPy obj)
{
    HPy one = HPyLong_FromLong(ctx, 1);
    if (HPyErr_Occurred(ctx))
        return HPy_NULL;
    HPy result = HPy_Add(ctx, obj, one);
    if (HPy_IsNull(result))
        return HPy_NULL;
    //HPy_Close(ctx, one); // OOPS
    return result;
}
```

## Debug mode (2)

```
setup(
    name="foo",
    hpy_ext_modules=[
        Extension('foo',
            sources=['foo.c']),
    ],
    setup_requires=['hpy.devel'],
)
```

```
$ python setup.py --hpy-abi=universal build_ext --inplace
$ ls foo.hpy.so
```

## Debug mode (3)

```
>>> import hpy.universal, hpy.debug
>>> foo = hpy.universal.load('foo', './foo.hpy.so', debug=True)
>>> foo.inc(5)
6
>>> with hpy.debug.LeakDetector():
...     foo.inc(5)
...
6
Traceback (most recent call last):
[...]
hpy.debug.leakdetector.HPyLeakError: 1 unclosed handle:
    <DebugHandle 0x5621a4c683a0 for 1>
```

## Debug mode (3)

WIP, temporary hack

```
>>> import hpy.universal, hpy.debug
>>> foo = hpy.universal.load('foo', './foo.hpy.so', debug=True)
>>> foo.inc(5)
6
>>> with hpy.debug.LeakDetector():
...     foo.inc(5)
...
6
Traceback (most recent call last):
[...]
hpy.debug.leakdetector.HPyLeakError: 1 unclosed handle:
    <DebugHandle 0x5621a4c683a0 for 1>
```

Eventually we will also show the C stack trace where the leaked handle was opened

## Questions to the audience

- Can we make HPy a “semi-official” API in the future?
  - 1st class support for importing modules
  - 1st class support to distribute wheels
- What is the general feedback towards HPy?