PAUL SCHERRER INSTITUT

PSI

Channel Access from Cython
and
Other Cython Use Cases

J. Chrin
Paul Scherrer Institute

EPICS Team Meeting, Barcelona, Spain, 8 Oct. 2017

Cython

- Cython not to be confused with CPython

- CPython: standard Python implementation

  ↓

  C distinguishes from Python the specification and other language implementations

- CPython: C-level interface, *Python/C API* (used to wrap C code)
- *Python/C API* is involved and non-trivial
- Cython uses the Python/C API extensively

Cython

- Programming language based on Python,
  but with optional **C/C++ static type declarations**

- The `cython` compiler translates
  Cython source code into optimized
  C/C++ code (*Python/C API*), which
  in turn may be compiled to a Python
  extension module

- Cython has the ability to call
  directly into C/C++ libraries

Cython

- Programming language based on Python, but with optional **C/C++ static type declarations**

- The `cython` compiler translates *Cython* source code into optimized C/C++ code (*Python/C API*), which in turn may be compiled to a Py… extension module

- Cython has the ability to call directly into C/C++ libraries

Very fast program execution, with major speed improvements (≤ 10^3)

Close integration with external C/C++ libraries

Cython

- Fully-fledged language
  - super-set of Python
  - plus C-like constructs
- Keeps abreast with developments in Python
  - compile > 98% of Python code

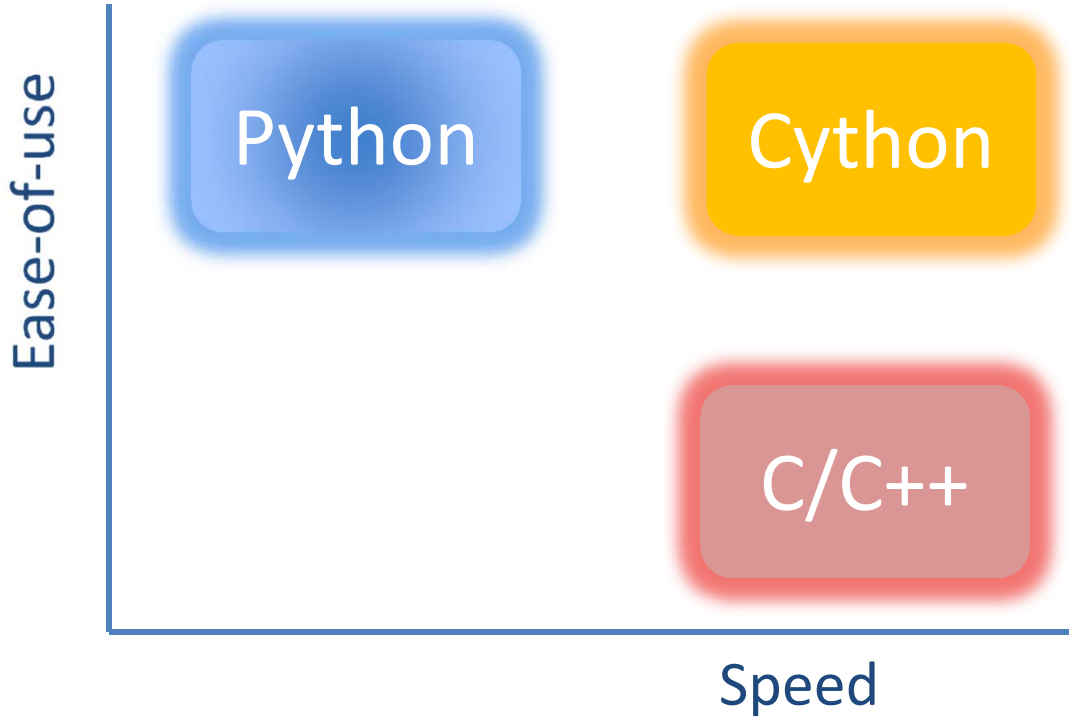
Cython

- Fully-fledged language
  - super-set of Python
  - plus C-like constructs
- Keeps abreast with developments in Python
  - compile > 98% of Python code

A major part of the Python ecosystem!

Cython

- A programming language
  with (extended) Python syntax
  but with the performance level of C!



PAUL SCHERRER INSTITUT

PSI

Use Case 1:
To Speed up Python Code

Python Example

A standard Python/Cython benchmark test

Python

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython Example

Cython can compile regular Python code to C code

Python

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython Example

Cython can compile regular Python code to C code

Interpreted

Python 3.5

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Compiled

Cython 0.23.4

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

15% Gain in Performance

Cython Example

Cython can compile regular Python code to C code, and gain major speed improvements from optional static type declarations

Python

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython

```
from math import sin


cdef double fn(double x):
  return sin(x**2)

def integrate_fn(double a,
                 double b, int N):
  cdef double s, dx
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython Example

Cython can compile regular Python code to C code, and gain major speed improvements from optional static type declarations

Python 3.5

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython 0.23.4

```
from math import sin


cdef double fn(double x):
  return sin(x**2)

def integrate_fn(double a,
                 double b, int N):
  cdef double s, dx
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

6x Gain in Performance

Cython Example

Cython can compile regular Python code to C code, and so gain major speed improvements from external declarations

Python

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython

```
cdef extern from "math.h"
  double sin(double)

cdef double fn(double x):
  return sin(x**2)

def integrate_fn(double a,
          double b, int N):
  cdef double s, dx
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython Example

Cython can compile regular Python code to C code, and so gain major speed improvements from external declarations

Python 3.5

```
from math import sin


def fn(x):
  return sin(x**2)

def integrate_fn(a,b,N):
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

Cython 0.23.4

```
cdef extern from "math.h"
  double sin(double)

cdef double fn(double x):
  return sin(x**2)

def integrate_fn(double a,
                 double b, int N):
  cdef double s, dx
  s=0
  dx=(b-a)/N
  for i in range(N):
    s+=fn(a+i*dx)
    return s*dx
```

12 x Gain in Performance

Cython Speed-up

Cython performance improvements due to combined use of external libraries and static typing.
What can be achieved will depend on use of…

- Nested loops
- Arithmetic operations
- Functions and their overhead
- Python Objects, which are are heap allocated versus Cython variables declared to be stack-allocated

Depending on use case, can lead to ~10^3 improvement in speed

PAUL SCHERRER INSTITUT

PSI

Use Case 2:
To Interface C/C++ Libraries

Channel Access from Cython

Cython wrapping around CA C library

PyCa.pyx

Cython

```
cdef extern from “cadef.h”
  int ca_pend_io(double)


def ca_pend_io(self, double timeOut):

   %interfacing ca_pend_io; not much to it

   with nogil:
        status=self.ca_pend_io(timeOut)

   return
```

Channel Access from Cython

Cython wrapping around CA C library

PyCa.pyx

Cython

```
cdef extern from “cadef.h”
  int ca_pend_io(double)


def ca_pend_io(self, double timeOut):

   %interfacing ca_pend_io; not much to it

   with nogil:
        status=self.ca_pend_io(timeOut)

   return
```

Releasing the GIL allows python threads to execute while a expensive operations runs concurrently

CyCafe

Cython wrapping around CA C++ library (CAFE)

PyCafe.pyx

Cython

PyCafe.pyd

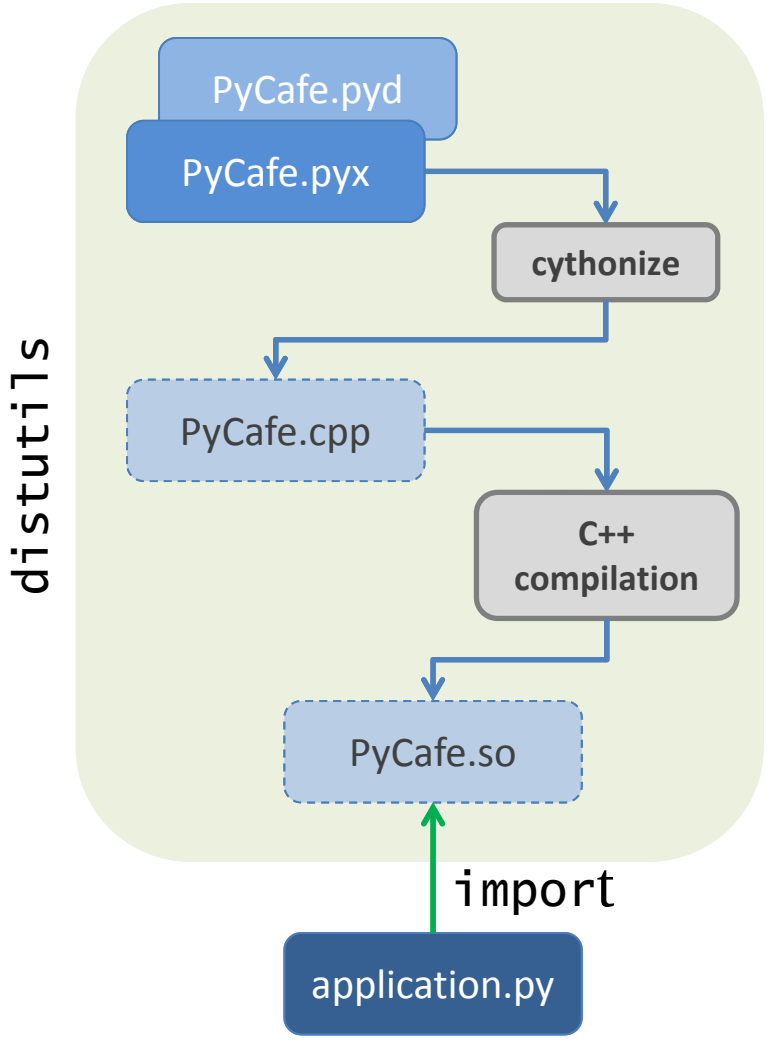
```
def getPV(self, handlePV, str dt='native'):

   %validate input, i.e., pvName or handle
   …
   cdef int status
   cdef PVDataHolder pvdata = PVDataHolder(1)

   with nogil:
        status=self._c_cafe.get(handle, pvdata)

   %throw exception if status !=ICAFE_NORMAL else
   …
   return PVDataHoldertoStruct(pvdata, dt)
```

PyCafe Extension Module

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
from numpy import get_include
setup(
  ext_modules=cythonize(
    [Extension('PyCafe',['PyCafe.pyx'],
      language='c++',
      include_dirs=[..., get_include()],
      library_dirs=[...],
      libraries=['ca','Com','dl','cafe'])
    ],
    annotate=True,
    compiler_directives={
      'embedsignature':False,
      'language_level':3,
      'c_string_type':'str',
      'c_string_encoding':'ascii',
      'py2_import':False,
      'warning_errors':False,
      'warn_unreachable':False,
      'noncheck':False,
      'boundscheck':False,
      'wraparound':False}
  )
)
```

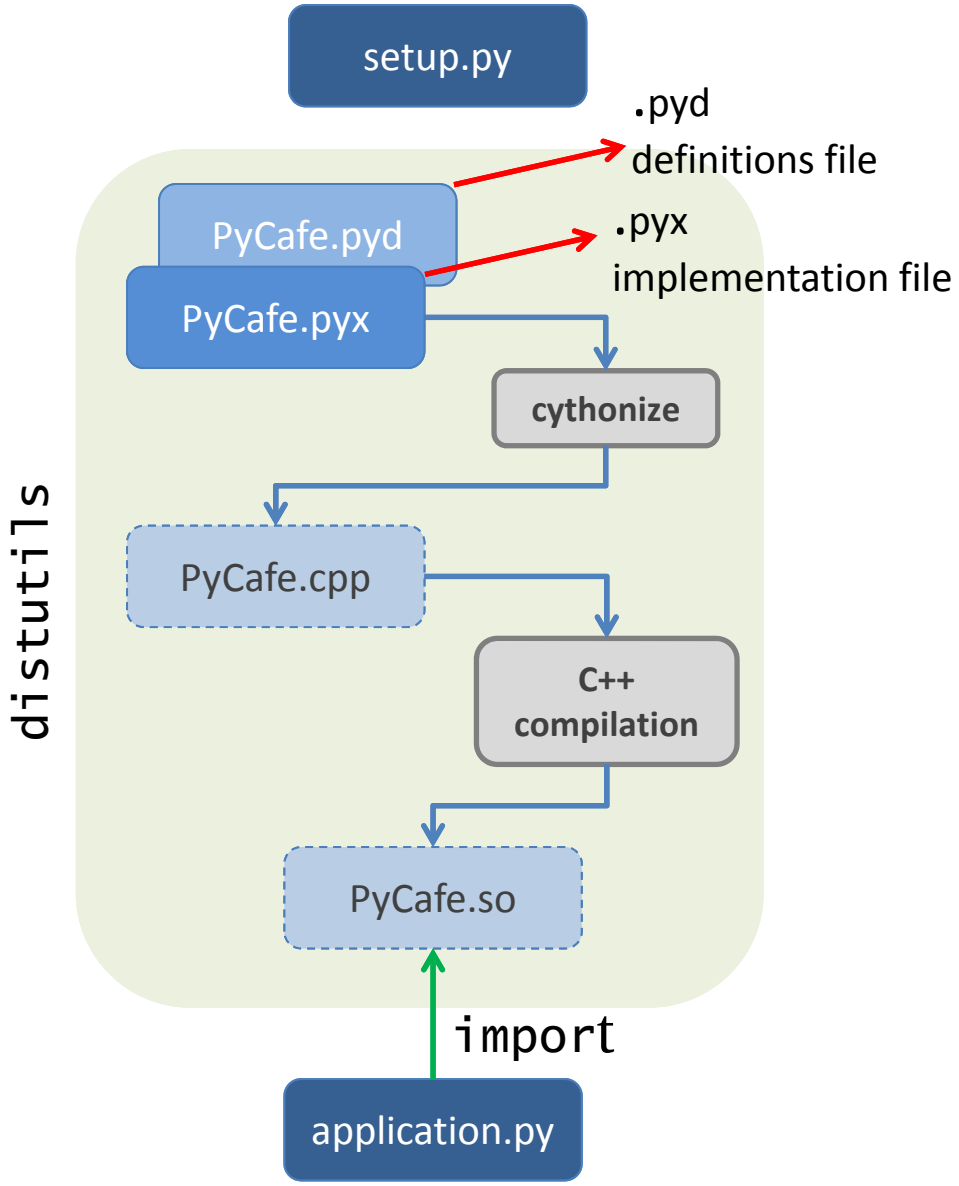
setup.py



PyCafe Extension Module

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
from numpy import get_include
setup(
  ext_modules=cythonize(
    [Extension('PyCafe',['PyCafe.pyx'],
      language='c++',
      include_dirs=[..., get_include()],
      library_dirs=[...],
      libraries=['ca','Com','dl','cafe'])
    ],
    annotate=True,
    compiler_directives={
      'embedsignature':False,
      'language_level':3,
      'c_string_type':'str',
      'c_string_encoding':'ascii',
      'py2_import':False,
      'warning_errors':False,
      'warn_unreachable':False,
      'noncheck':False,
      'boundscheck':False,
      'wraparound':False}
  )
)
```

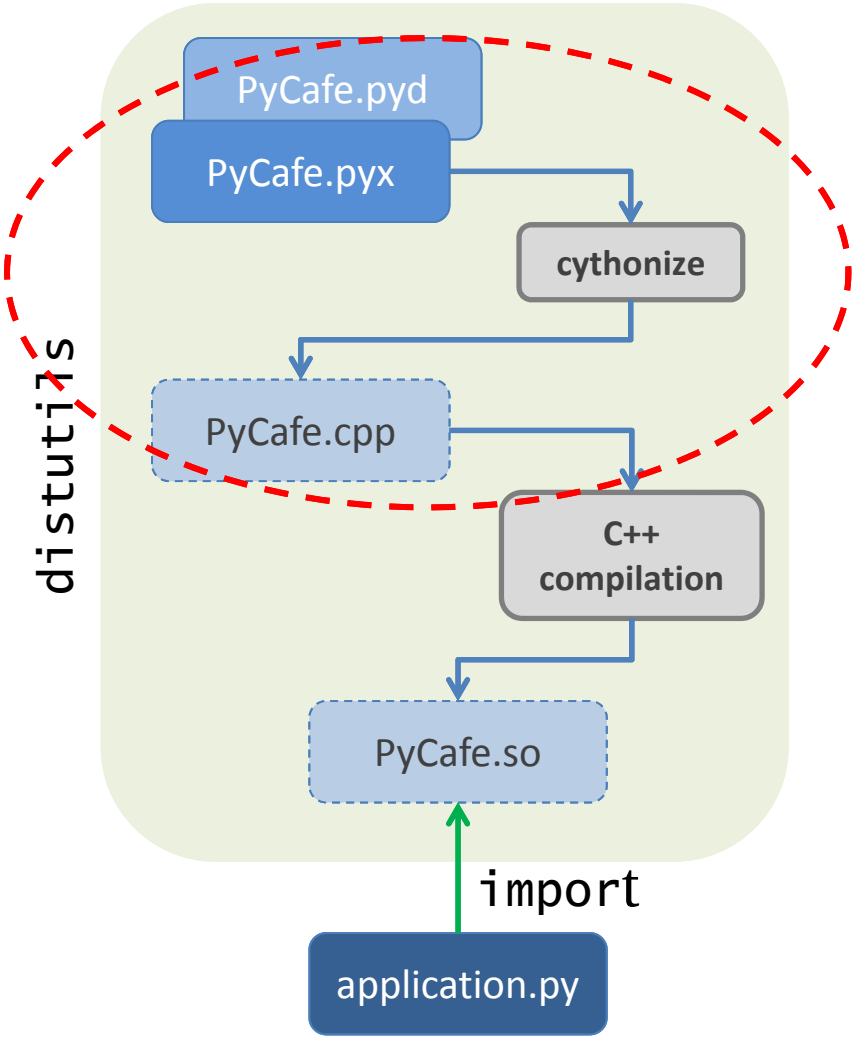


PyCafe Extension Module

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
from numpy import get_include
setup(
  ext_modules=cythonize(
    [Extension('PyCafe',['PyCafe.pyx'],
      language='c++',
      include_dirs=[..., get_include()],
      library_dirs=[...],
      libraries=['ca','Com','dl','cafe'])
    ],
    annotate=True,
    compiler_directives={
      'embedsignature':False,
      'language_level':3,
      'c_string_type':'str',
      'c_string_encoding':'ascii',
      'py2_import':False,
      'warning_errors':False,
      'warn_unreachable':False,
      'noncheck':False,
      'boundscheck':False,
      'wraparound':False}
  )
)
```

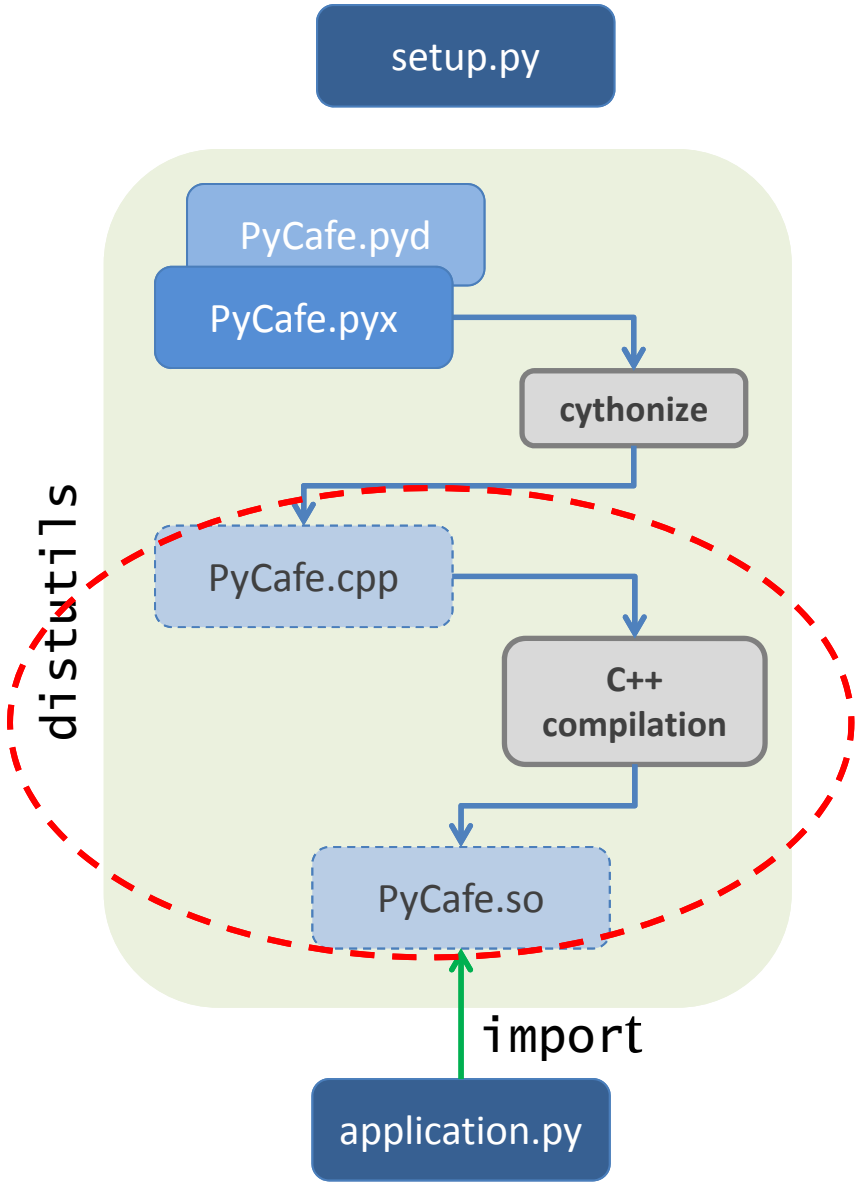
setup.py



PyCafe Extension Module

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
from numpy import get_include
setup(
  ext_modules=cythonize(
    [Extension('PyCafe',['PyCafe.pyx'],
      language='c++',
      include_dirs=[..., get_include()],
      library_dirs=[...],
      libraries=['ca','Com','dl','cafe'])
    ],
    annotate=True,
    compiler_directives={
      'embedsignature':False,
      'language_level':3,
      'c_string_type':'str',
      'c_string_encoding':'ascii',
      'py2_import':False,
      'warning_errors':False,
      'warn_unreachable':False,
      'noncheck':False,
      'boundscheck':False,
      'wraparound':False}
  )
)
```

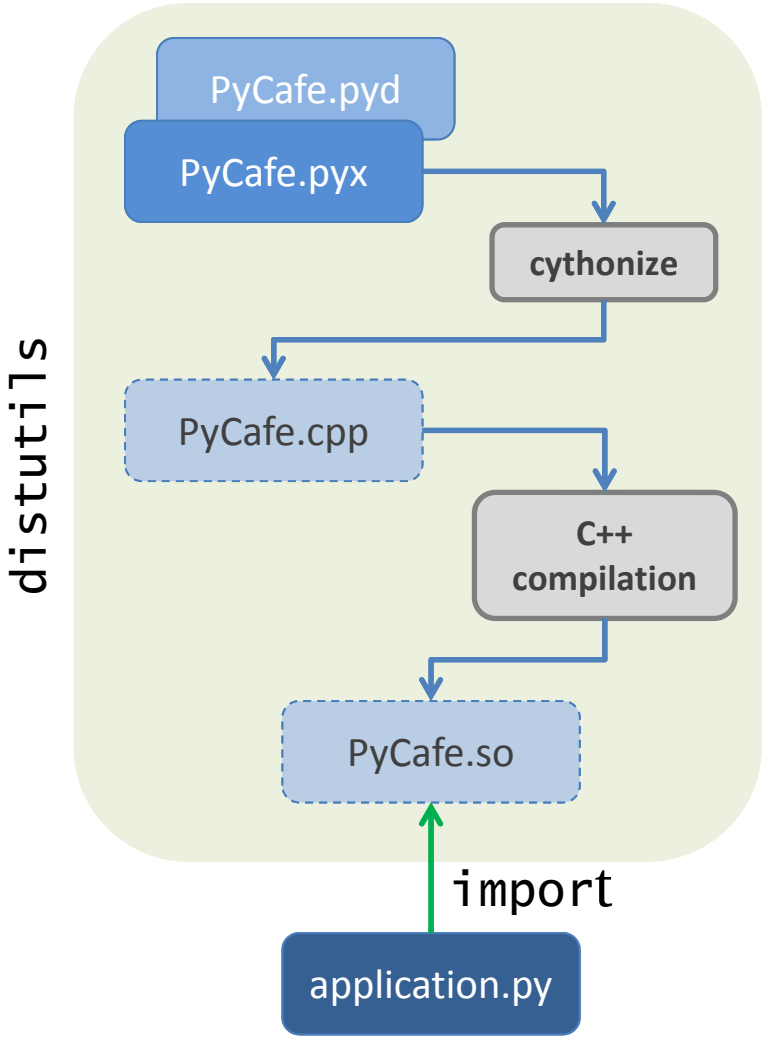


PyCafe Extension Module

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize
from numpy import get_include
setup(
  ext_modules=cythonize(
    [Extension('PyCafe',['PyCafe.pyx'],
      language='c++',
      include_dirs=[..., get_include()],
      library_dirs=[...],
      libraries=['ca','Com','dl','cafe'])
    ],
    annotate=True,
    compiler_directives={
      'embedsignature':False,
      'language_level':3,
      'c_string_type':'str',
      'c_string_encoding':'ascii',
      'py2_import':False,
      'warning_errors':False,
      'warn_unreachable':False,
      'noncheck':False,
      'boundscheck':False,
      'wraparound':False}
  )
)
```

setup.py



PyCafe

- Simple single channel operations
- Waveforms and arrays (`memoryview` data types)
- Multiple scalar operations, i.e., simultaneous operations on several PVs with scalar values
- Multiple compound operations, i.e., simultaneous operations on several PVs with either scalar values or arrays
- Synchronous group operations (user or externally defined)
- Asynchronous interactions and retrieving data from cache
- Monitors, either with or without user supplied callbacks
- Control system parameters, i.e., operating limits, engineering units
- Specialized methods, e.g., match, setAndMatch, wishlist …

Arrays and memoryview

Retrieving a waveform as a numpy.ndarray

```
value = cafe.getArray ( ⟨handlePV⟩,
                        art='numpy', [dt='native'])
```

Python objects that implement the new buffer protocol, e.g., `bytes`, `bytearray`, `numpy.ndarray`, `array.array`, *can share their data without copying.* The C level buffer interface may further be exposed as a `memoryview` object that, supports slicing and indexing

Retrieving a waveform as a memoryview

```
value = cafe.getArray ( ⟨handlePV⟩,
                        art='memoryview', [dt='native'])
```

Monitors and Callbacks

Initiating a monitor with a user supplied callback

```
monitorID = cafe.monitorStart (⟨handlePV⟩, cb=py_cb)
```

```
def py_cb(handle):

    if cafe.isConnected(handle):
        pvData=cafe.getPVCache(handle, [dt='native'])

        #  'set' operations, including those on other
        #  handles, are permitted.
        #  Invoke Qt signal here to update Qt Widget
        …
        return
```

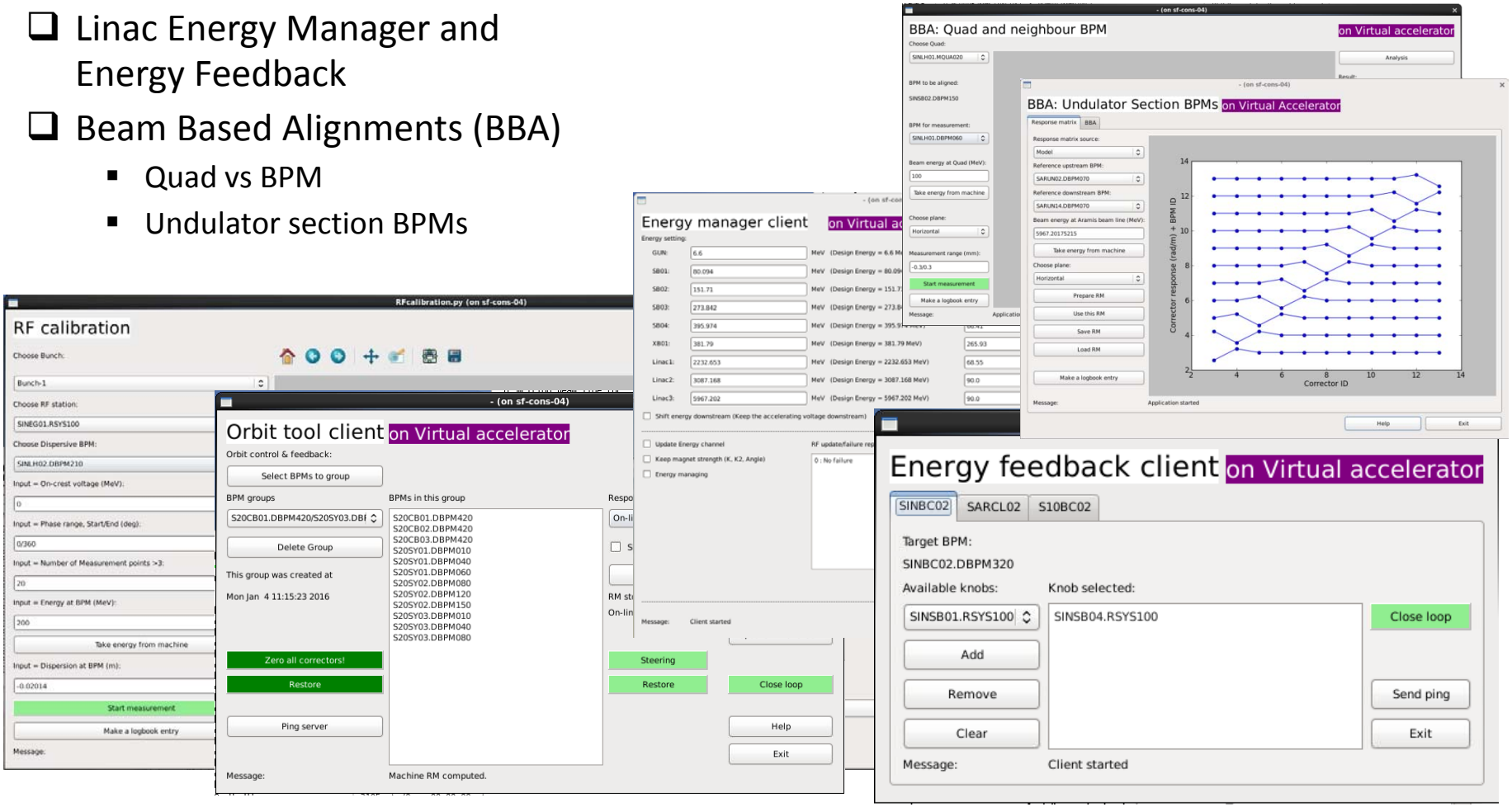
Only the handle (i.e., object reference) is, and need be, reported back to the callback function

SwissFEL HLA Development

Application developed and tested on **Virtual Accelerator** [Masamitsu Aiba]

- ❑ RF Calibration
- ❑ Orbit Tool and Orbit Feedback
- ❑ Linac Energy Manager and Energy Feedback
- ❑ Beam Based Alignments (BBA)
  - Quad vs BPM
  - Undulator section BPMs

~ 6.10^4 EPICS soft channels

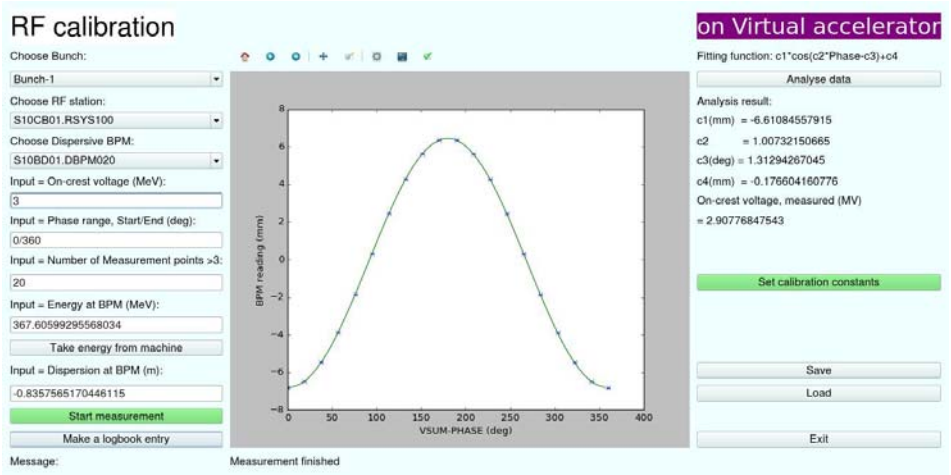


SwissFEL HLA Development

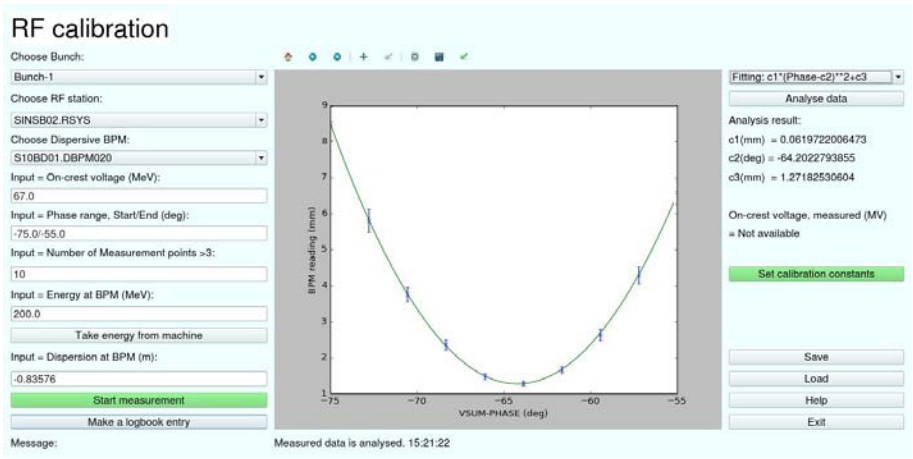
[Masamitsu Aiba]

Applications proven on Virtual Accelerator,
readily applied for SwissFEL commissioning

Virtual Accelerator



Real Accelerator



PyCafe Highlights

- Channel access requirements already met by the extensive interface provided by CAFE (hard work long done!)
- *PyCafe* exposes extensive CA interface to Python application developers; tested and verified through virtual (and now real) SwissFEL accelerator
- Performance improvement for single scalar read:

PyCafe is ~ 40% times faster than `ctypes`

Performance difference vanishes for large waveforms (10^6 elements)

PAUL SCHERRER INSTITUT

TRACY

TRACY: C/C++ Electron Beam Dynamics Simulation Code

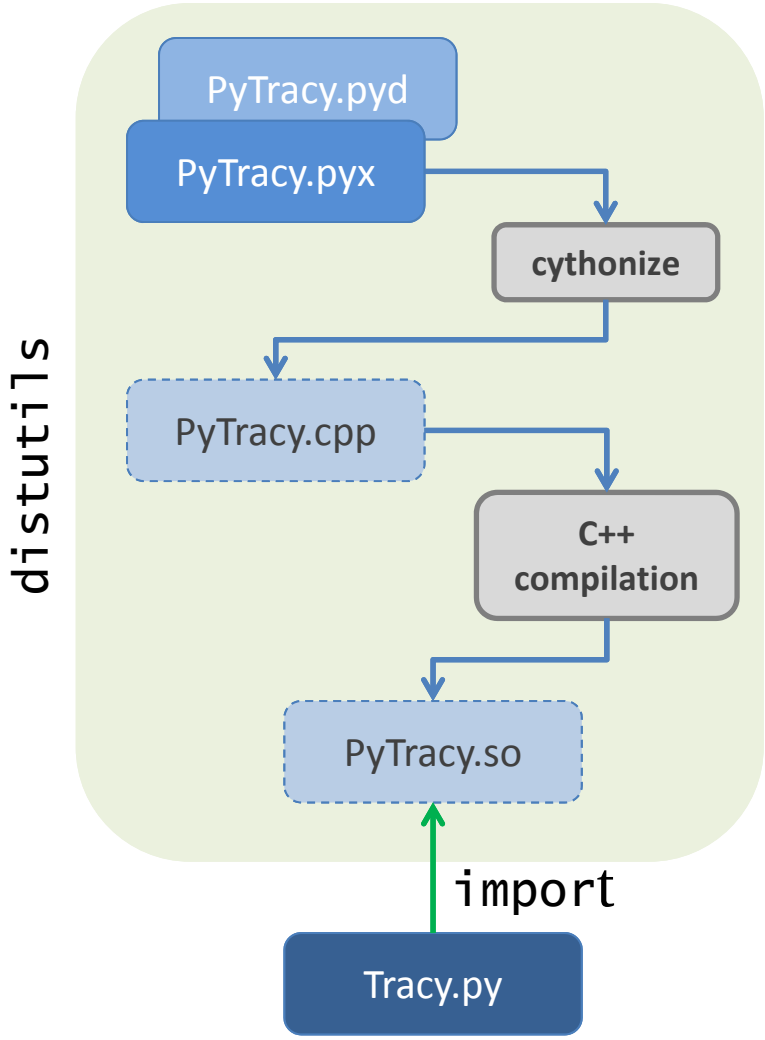
[J. Bengtsson, M. Böge]

PyTracy Extension Module

setup.py

```
from distutils.core import setup
from distutils.extension import Extension
from Cython.Build import cythonize

setup(
  ext_modules=cythonize(
    [Extension('PyTracy',['PyTracy.pyx'],
      language='c++',
      include_dirs=[...],
      library_dirs=[libtracy.a,
                    libnum_rec.a,…],
      libraries=[])
    ],
    annotate=False,
    compiler_directives={
      'embedsignature':False,
      'language_level':3,
      'c_string_type':'str',
      'c_string_encoding':'ascii',
      'py2_import':False,
      'warn_unreachable':False,
      'noncheck':False,
      'boundscheck':False,
      'wraparound':False}
  )
)
```

setup.py



TRACY

TRACY: C/C++ Electron Beam Dynamics Simulation Code
[J. Bengtsson, M. Böge]

PyTracy: exposes most common TRACY functions to Python

- Read Lattice File,
- Get Twiss Parameters,
- Beam Emittance ,
- Tune Measurement,
- and more….

PAUL SCHERRER INSTITUT PSI

Summary

Cython is an extended Python language and a tool for:

- Translating (all, or subset) Python Code to C/C++ Code, with spectacular improvements for CPU-bound operations
- Easily binding to external C/C++ libraries. Interfaces can be customized, simplified and pythonized as they are wrapped

PyCafe: extensive and efficient Python CA interface
PyTracy: accelerator modelling from Python

Cython can also be useful for:
Avoiding the gil, thread based parallelism
Interchanging between C and Python, without the Python C API

GRACIAS POR SU ATENCIÓN

Cython and EPICS ?
Hmmn…, I think I'll
give it a "shot"

It's available from:
gitlab.psi.ch/cafe/

https://anaconda.org/
paulscherrerinstitute/

Windows download
https://drive.switch.ch/index.php/
s/WaAVHCqYpGTc9N4

CAFE home:
ados.web.psi.ch/cafe/

PAUL SCHERRER INSTITUT

PSI

Supporting Material

PAUL SCHERRER INSTITUT PSI

CAFE, A C++ Channel Access Library

CAFE is a modern, C++ interface to the native EPICS Channel Access (CA) C client library that provides a concise, complete, and clean interface with minimal details of the low-level CA implementation propagating to the user. It places careful attention to:

- connection management and memory optimization, particularly as connections are broken and restored
- separation between data acquisition and its presentation
- strategies for converting between requested and native types
- caching of data related to the channel and its current state
- aggregation of requests to enhance performance
- capturing and reporting errors with integrity
- adaptive correction procedures, e.g., network timeouts
- *temporal decoupling from CA servers !!!!*

PAUL SCHERRER INSTITUT PSI

Advantages of the CAFE library approach

- Bindings to scripting and domain-specific languages simplified
- Inherent convenience of maintaining a single ca interface code
- New CA functionalities from future EPICS releases need only be integrated in a single repository
- A uniform response to error conditions will help identify problems and increase recovery time
- In-house CA expertise ensures a quick response to user needs and problem solving

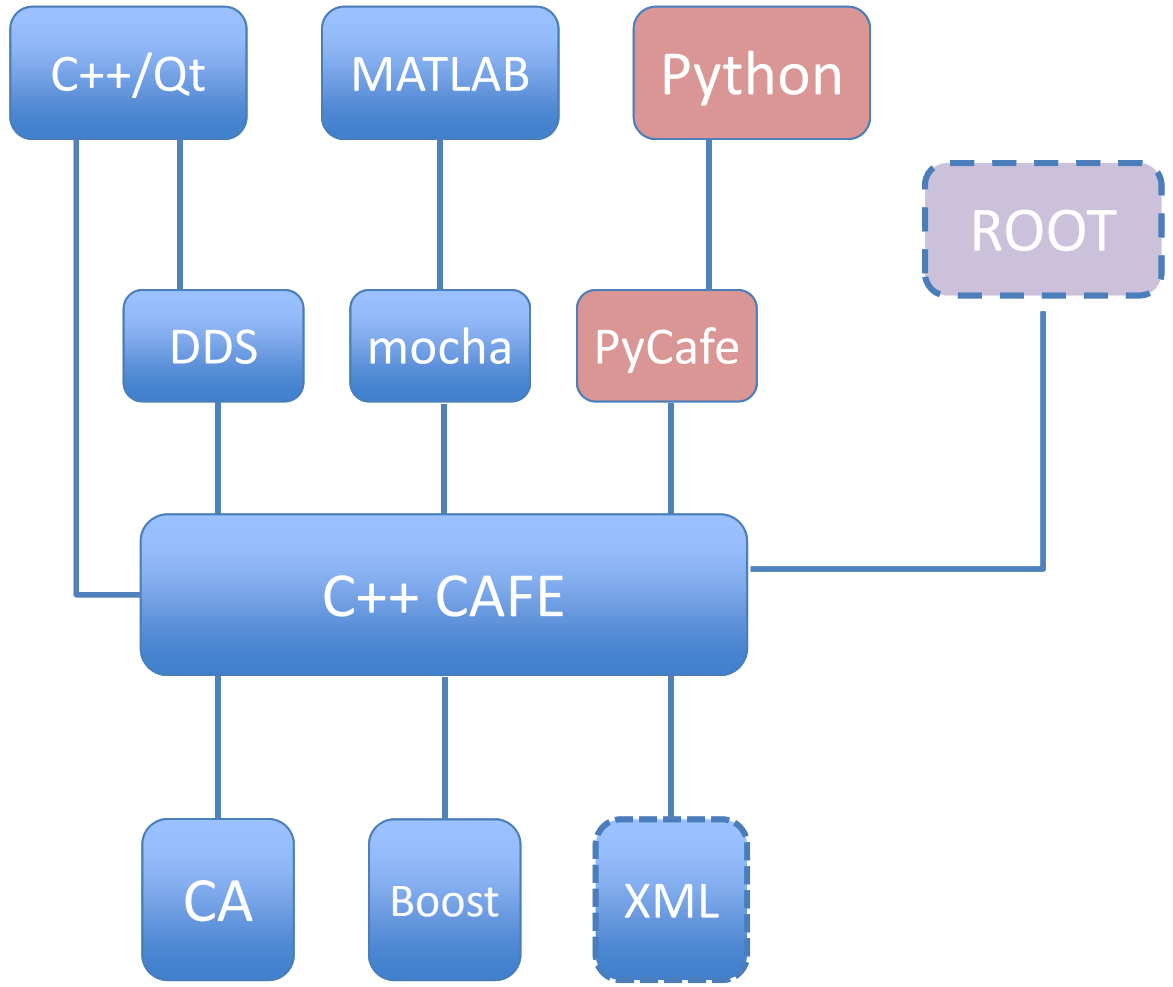
CAFE Extensions

Application Layer

Adapter Layer

CA Interface

Third Party Layer



CAFE Extensions

Application Layer

Adapter Layer

CA Interface

Third Party Layer