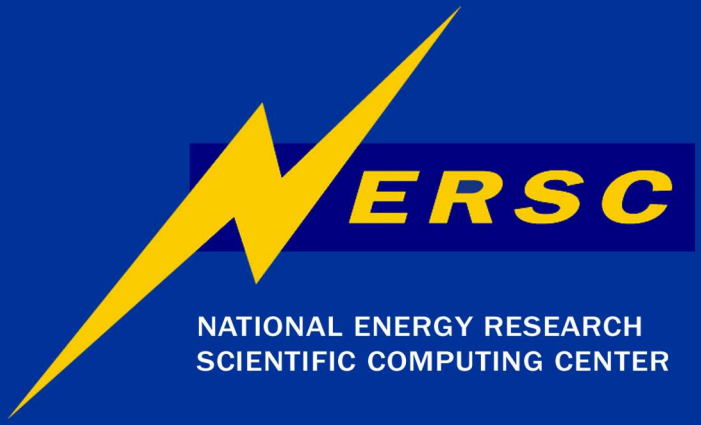


National Energy Research Scientific Computing Center (NERSC)

Reflection-Based Python-C++ Bindings

Wim T.L.P. Lavrijsen
NERSC HENPC, LBNL
CHEP 2004, Interlaken, Switzerland - 27/09/04

Outline

- **Motivation and Introduction**
  - Scripting languages
  - Python-C++ interoperation
- **Technology Overview**
  - Different game-plans
  - Available products
- **PyLCGDict, PyROOT**
  - Overview and status
  - Outlook

The Case for Scripting

- **Typically, scripting languages are:**
  - Simple, high-level, dynamically typed
  - Designed for “gluing” existing components
  - Interactive, interpreted
  - Missing steps in write/build/nap/run/debug
- **Improved productivity**
  - Reduced learning curve
  - More effective re-use of components
  - Shorter development cycle

The Case for Python

- **Simple, elegant, easy to learn**
  - Based on ABC, a teaching language
  - Tutorials available online (www.python.org)
- **Many standard and 3rd party modules**
  - 2nd most popular in use, most for bindings
- **Used for scientific programming**
  - Open source, freely available
  - Extensions for high performance and distributed parallel code (www.scipy.org)

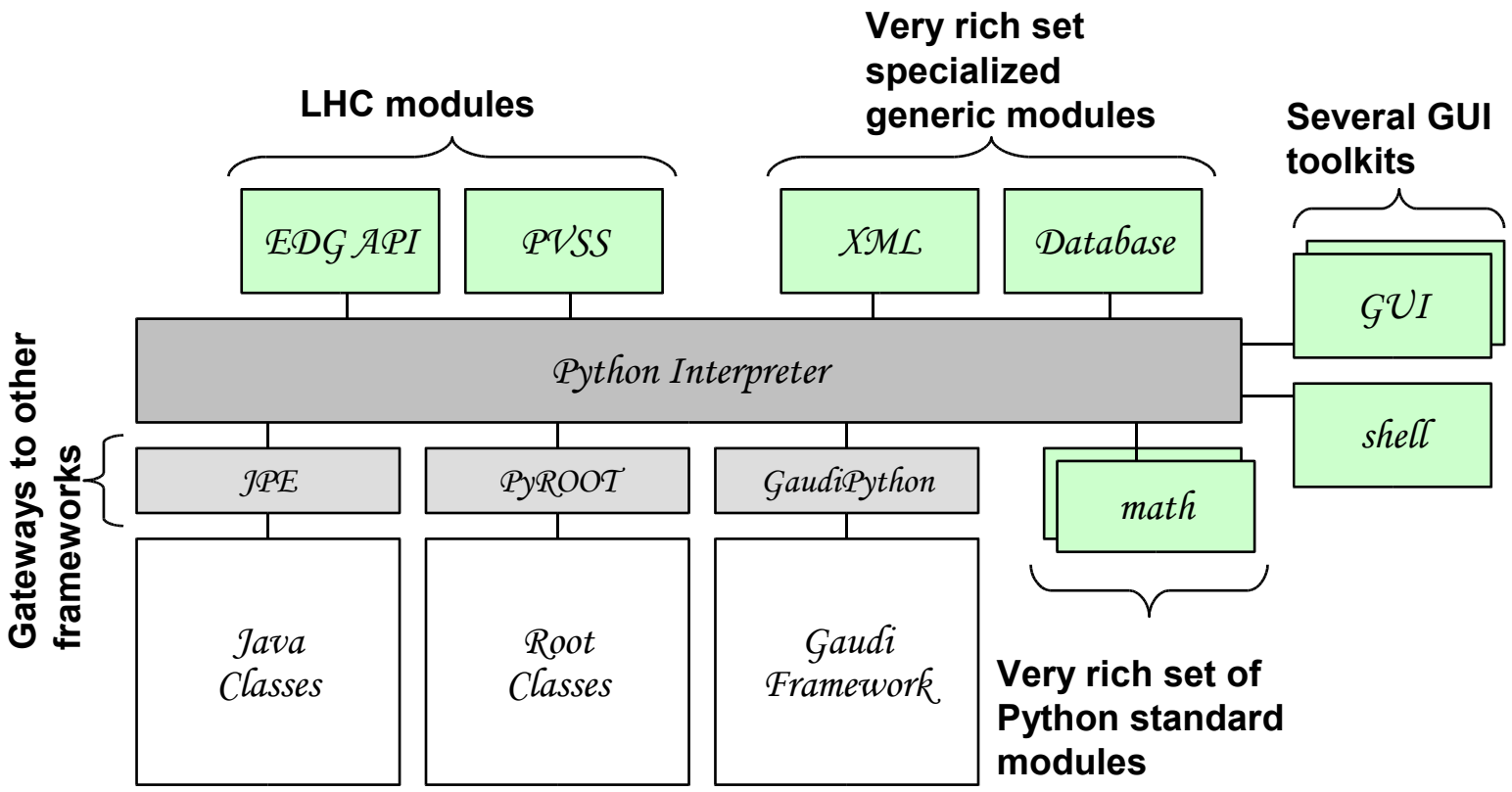
A Python Session

```
$~> python2.2
>>> import math
>>> math.sin( 0.5 * math.pi )
1.0
>>> from urllib import urlopen
>>> page = urlopen( "http://cern.ch" )
>>> for line in page.readlines():
...     print line,
...
<!DOCTYPE HTML PUBLIC "-//W3C//DTD H..
[ .. etc .. ]
>>> ^D
$~>
```

Python-C++ Interoperation

- **Access C++ code from Python**
  - Other languages from C++ through Python
- **Need Python *bindings* to C++ code**
  - Hand-written (C-API) or generated
  - Requires taking care of:
    - Object, parameter conversions
    - Memory management
    - C++ function overloading
    - C++ templates

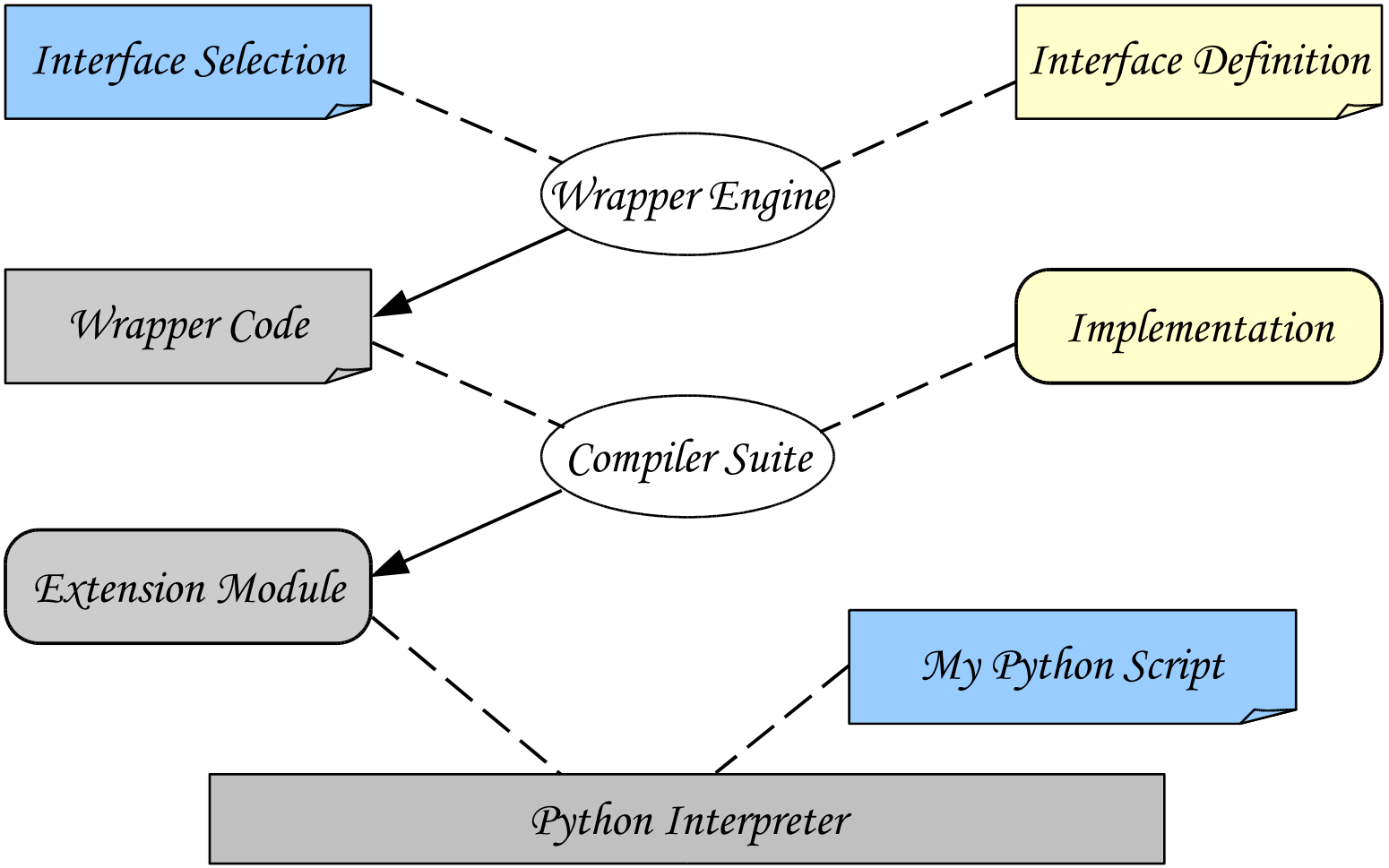
Python as Glue



Static Wrapping

- **Wrappers created from interface file**
  - Several steps required (can be automated)
  - Control creation with a selection file
- **Wrappers are written out as code**
  - Compiled into an extension module
    - Missing classes replaced by stubs
  - Internal bookkeeping for class sharing
    - Types are registered to allow conversions
  - Function returns follow bound signature
    - But allow explicit (dynamic) casts by the user

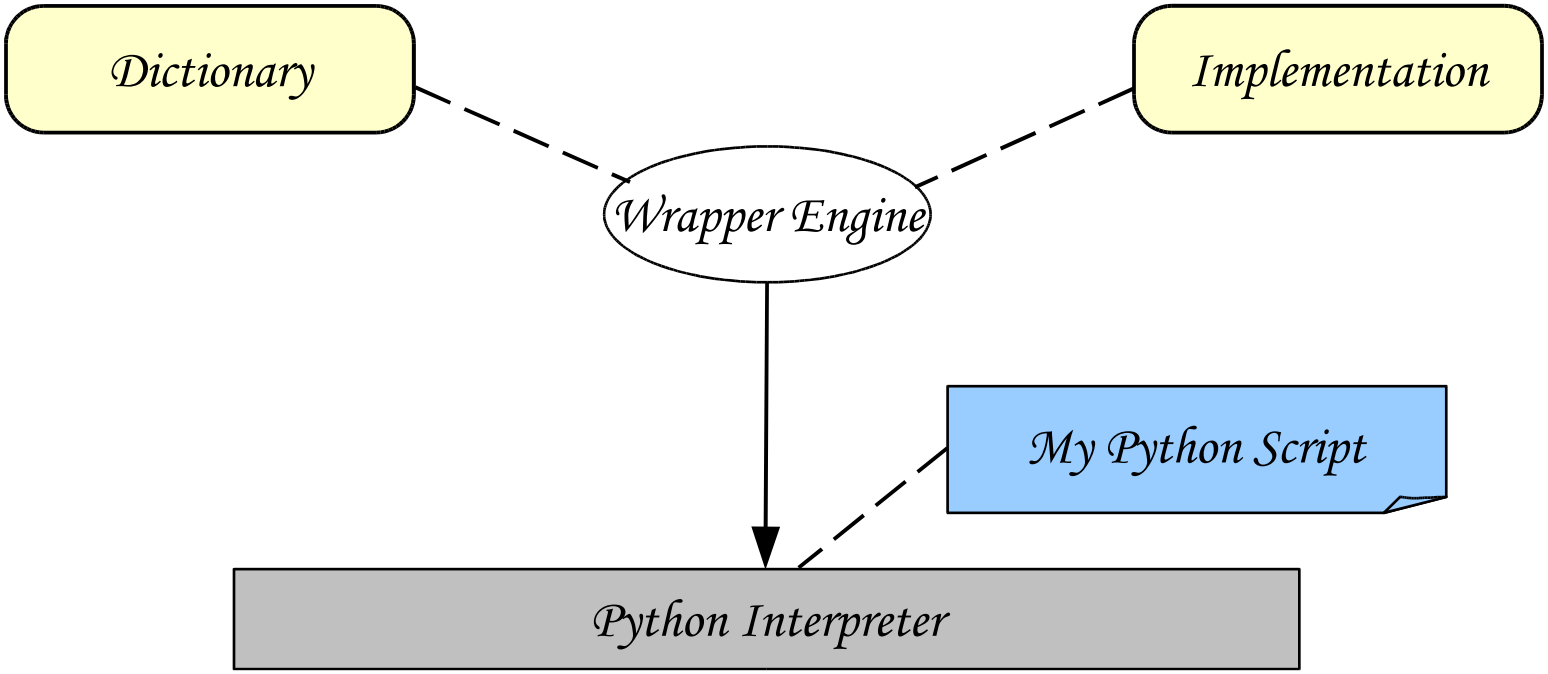
Static Wrapping



Dynamic Wrapping

- **Wrappers created from reflection info**
  - If dictionary available: minimal user effort
    - If not, very similar effort as static wrapping
  - No/Little control over creation mechanism
- **Python classes, built-up in memory**
  - Loaded and created on-demand
    - Missing classes can be automatically loaded
  - "Bookkeeping" is in Python classes itself
    - Conversions are derived from reflection info
  - Function returns follow actual type

Dynamic Wrapping



Available Products

- **SWIG**
  - Original tool, 13 target languages, static
- **Boost.Python**
  - Large feature set, C-API replacement, static
- **PyLCGDict**
  - LCGDict based, dynamic
- **PyROOT**
  - ROOT/CINT based, two-way, dynamic
- **CABLE, CXX, SIP, etc.**

Trade-offs

- **Prefer dynamic wrapping**
  - Lots of dictionaries already available
    - POOL persistency, ROOT analysis code
  - No casting, auto-loading
- **Boost.Python is rather slow**
  - C++ exception thrown on failed overload
  - Spurious object copying
- **SWIG is often cumbersome to use**
  - Generated code doesn't always compile

SEAL

- Shared Environment for Applications at LHC
- Provide LHC core and services libs
  - Foundation class libs (system, math, etc.)
  - Framework svcs (plugins, scripting)
  - Improve coherency of LCG applications
- Strategy of pluggable components

PyLCGDict

- **Based on LCGDict**
  - Dictionaries from POOL persistency svcs
  - Atlas/LHCb framework interactivity
    - Job configuration
    - Basis for Athena main program in python
    - Access to Transient Store and user classes
- **Part of SEAL**
  - Released with SEAL_1_4_0 and later
  - To be superseded by PyReflex
- **Goal: maximize physicist ease-of-use**

PyROOT

- ***Bridge* between Python and ROOT**
  - Dictionaries available from CINT
  - Access ROOT objects from Python *and VV.*
  - Interchange Python / CINT sessions
  - Works with ROOT memory handling
  - Class level “pythonization”
    - eg. for histogram fitting, ROOT arrays
- **Originally in SEAL, now part of ROOT**
  - Released with ROOT v4.00/04 and later

Resources

- **Online documentation**
  - www.python.org/doc
  - www.swig.org/doc.html
  - www.boost.org/python/doc
  - cern.ch/seal/snapshot/workbook/PyLCGDict2-howto.html
  - cern.ch/wlav/pyroot
- **Installations**
  - /afs/cern.ch/sw/lcg/app/releases/SEAL
  - /afs/cern.ch/sw/root

Outlook

- **Interactive access to Gaudi/Athena**
  - Work from Python interpreter or CINT
  - Useful for debugging user code
  - New ways of doing analysis:
    - Have Athena services available in analysis
- **LCGDict and CINT dict to integrate**
  - New API: Reflex, part of SEAL
  - Keep PyReflex general
  - Add specific ROOT features for PyROOT

Conclusions

- **Physicists' toolkit beefed up w/ Python**
  - Adds advantages of scripting languages
  - Easy connection to many existing libraries
- **Developed tools to provide ease of use**
  - PyLCGDict, PyROOT: generic, dynamic
  - Automatic binding for user classes
  - Provide binding for standard physics libs