



ATLAS Software Workshop - 24 Sep 2003

Physics Analysis Toolkits Introduction

David R. Quarrie

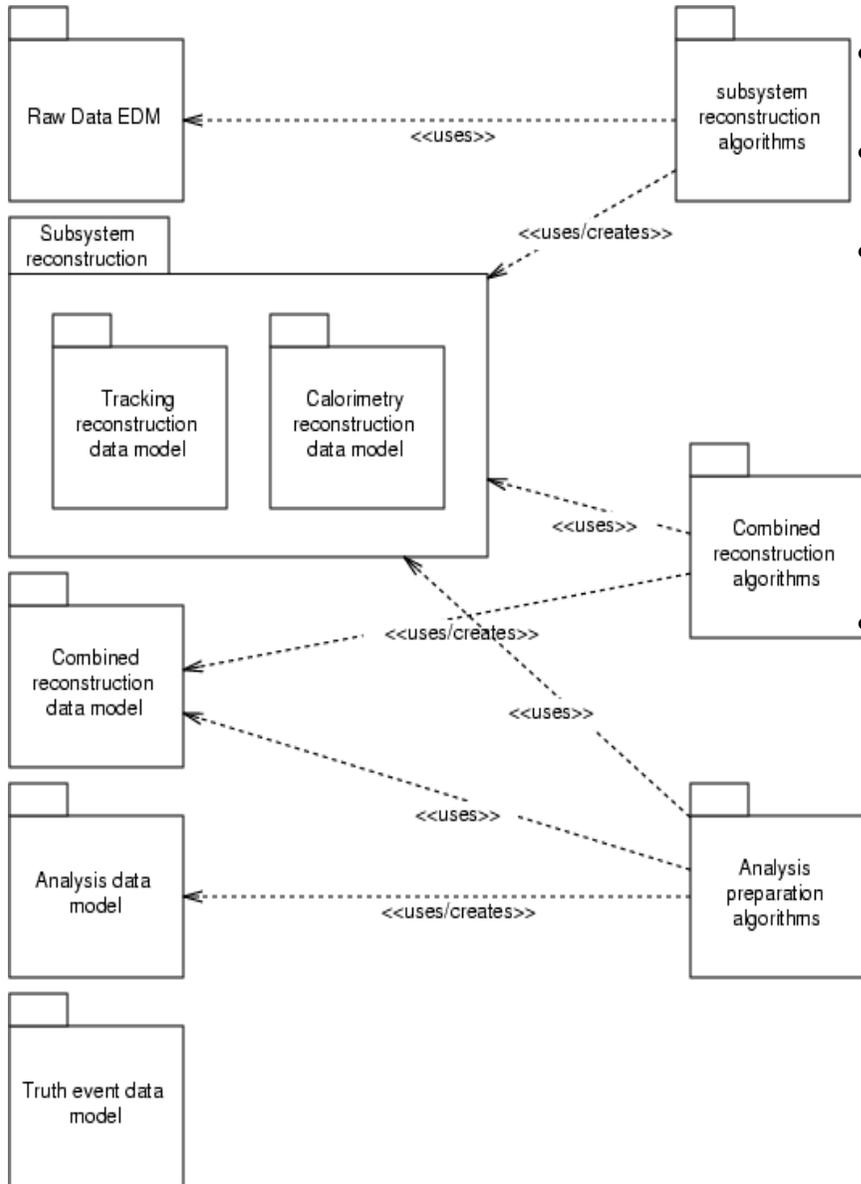
CERN/LBNL

David.Quarrie@cern.ch

Scope

- Span the gap from reconstruction to n-tuple analysis
- RTF mandate stopped at the "Analysis Preparation" Domain with the production of Analysis Data Model
- Goal is to provide an environment where reconstruction tools are available (e.g. refit tracks, vertices), but where advantage can be taken of "n-tuple" analysis tools (ROOT/PAW/JAS)
 - Operate in context of Athena framework
- Support both batch and interactive operation
- This session isn't about ROOT/PAW/JAS as stand-alone tools
- Neither isn't about distributed physics analysis (DIAL, AliEn...)

RTF recommendations



- Very brief overview... please read the report and see following talks
- Modularity, granularity, baseline reconstruction
- Reconstruction top down design (dataflow)
 - Domains: sub-systems, combined reconstruction and analysis preparation
 - Analysis of algorithmic components, identified common tools
 - Integration of fast simulation
 - Steering
- EDM
 - Common interfaces between algorithms
 - e.g. common classes for tracking subsystems
 - Design patterns to give uniformity to data classes in combined reconstruction domain
 - Approach to units and transformations
 - Separation of event and non-event data
 - Navigation



History/Organization

- We had a similar session in Nov 2002 software week
 - David Rousseau will summarize that as well as adding his own thoughts/ideas
- My intention is to form a sub-group within the Event Selection, Reconstruction and Analysis Tools (ESRAT) group to work in this area
 - This was meant to be a "kick-off" session for this sub-group
 - But as I indicated on Monday, the organization of ESRAT has not yet converged



What are other experiments doing?

- I spent some time trying to get feedback from other experiments
 - CMS, LHCb, CDF, D0, BaBar, CLEO
- Gloria Corti will describe DaVinci, the LHCb "toolkit"
- When talking to Vincenzo I discovered that they have just held a workshop on this area and are pretty much in the same situation as we are
 - Working group formed
 - Technical approach similar to ours
 - <http://agenda.cern.ch/fullAgenda.php?ida=a031708>
 - ◆ Includes talks from BaBar, CDF, D0, Phobos



Other experiments

- CDF

- Mix of physics-specific Modules (Algorithms in Athena terminology) and ROOT n-tuple based analyses
- Some use of general-purpose toolkits (Physics Analysis Expert - PAX)
- Believe should allow multiple approaches - one might be more appropriate than the other for particular analyses

- BaBar

- Generic analysis toolkit (Beta) & ROOT based analyses
 - ◆ <http://www.slac.stanford.edu/BFROOT/www/Computing/Offline/Beta/Beta.html>

- CLEO

- DChain combinatorics package for decay chains
 - ◆ www.hep.net/chep98/PDF/61.pdf



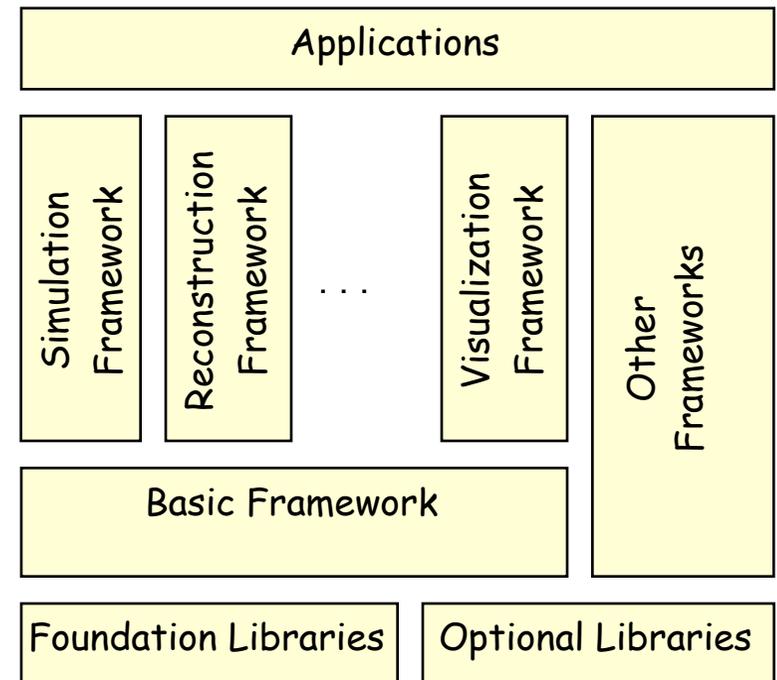
Relationship to LCG Blueprint

- Athena essentially conforms to the component-based architecture proposed by the LCG Blueprint RTAG
- Clearly predates it, but is evolving to conform
 - Integration of POOL, SEAL and PI components
- Use of Component Bus
- Central role of Data Dictionary
- Support for batch & interactive use
 - Scripting

Architectural Elements (Pere Mato)

- ◆ Interface model
 - Abstract interfaces, versioning, guidelines,...
- ◆ Component model
 - Communication via public interfaces (no hidden channels)
 - Plug-ins (run-time loading)
 - Life-time management (reference counting)
 - Configuration
- ◆ Design guidelines
 - Dependencies
 - Exception handling
 - [Interface to external components](#)

Software Structure



Architectural Elements (2)

◆ Object Dictionary

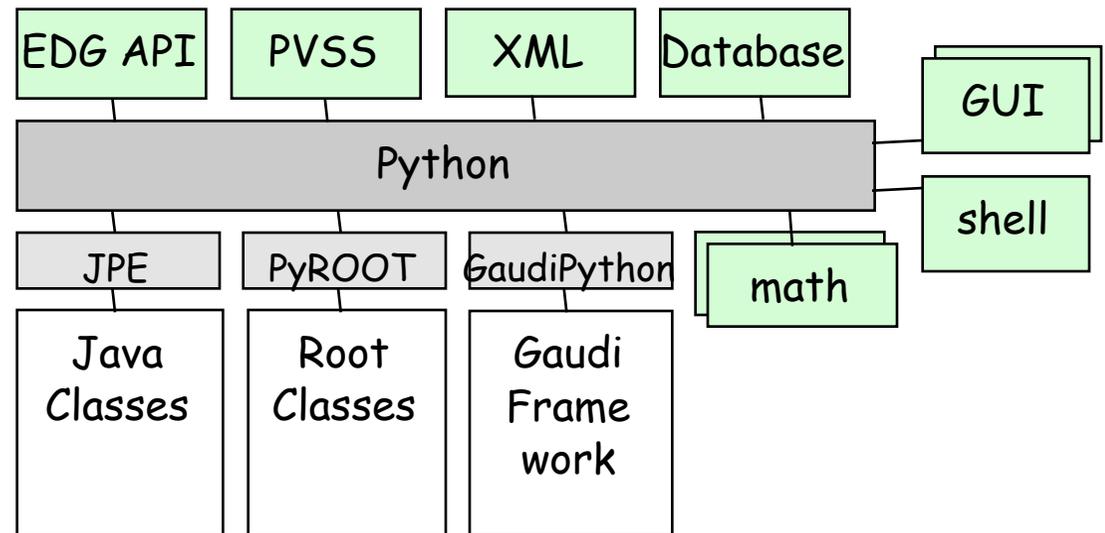
- The ability to query a class about its internal structure (Introspection)
- Essential for data browsing, rapid prototyping, persistency, etc.

◆ Object Whiteboard

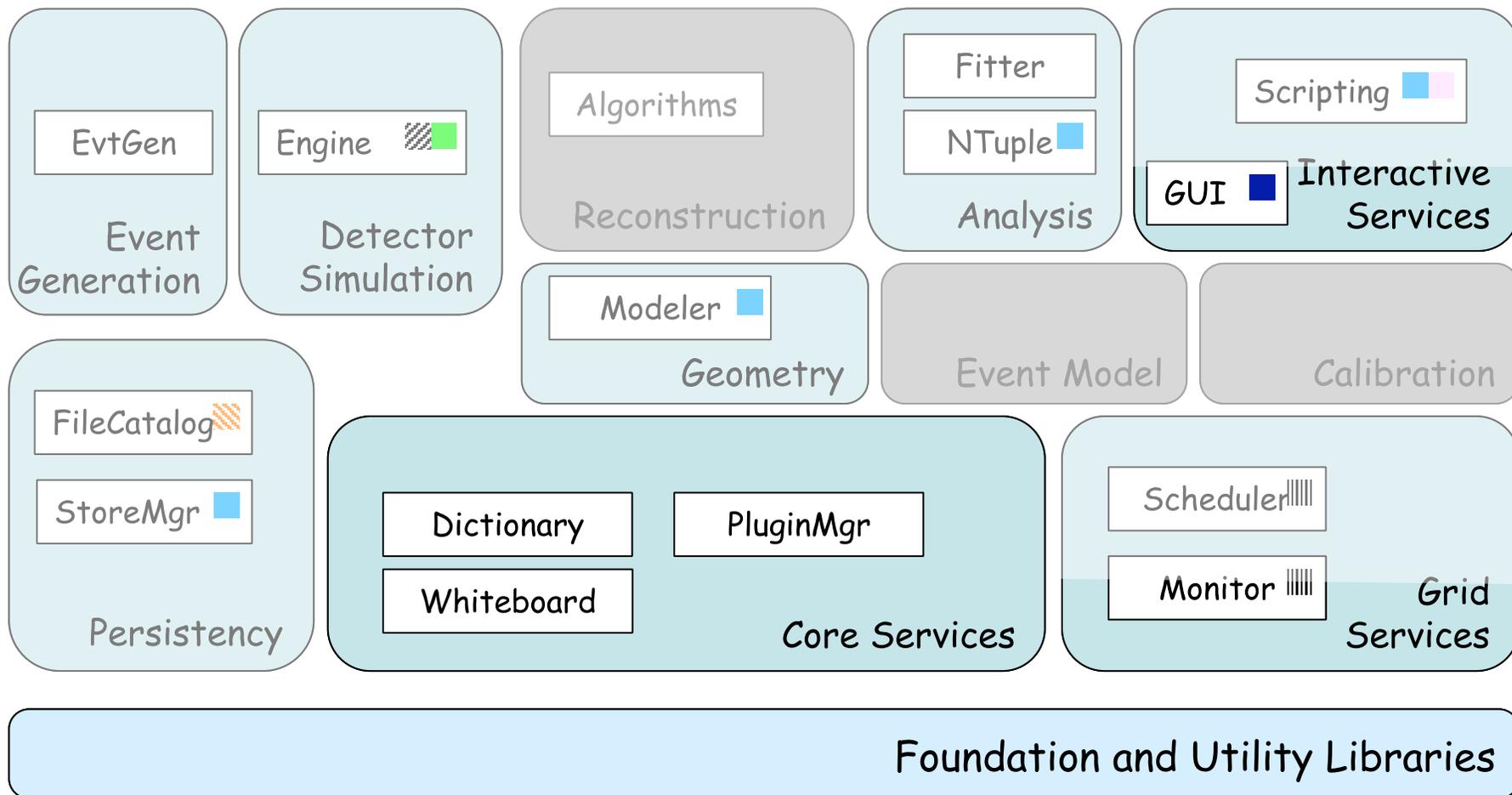
- Uniform access to application-defined objects (equivalent to the Gaudi transient stores)

◆ Component Bus

- Integration of components providing a wide variety of functionality



Domain Coverage



■ ROOT
 ■ GEANT4
 ▨ FLUKA
 ▨ MySQL
 ||||| DataGrid
 ■ Python
 ■ Qt
 ...



Scripting Services

- ◆ Scripting goals, among others, are:
 - Interactive use and development
 - Run-time configuration
 - Rapid prototyping
 - Integration
- ◆ Two commonly used tools in HEP: Python, CINT
 - Interoperability considered useful: end-user choice
- ◆ Python bindings
 - Possible to any package that can interface to C
 - Evaluation: SWIG, Boost.Python functionally equivalent



Scripting Services (2)

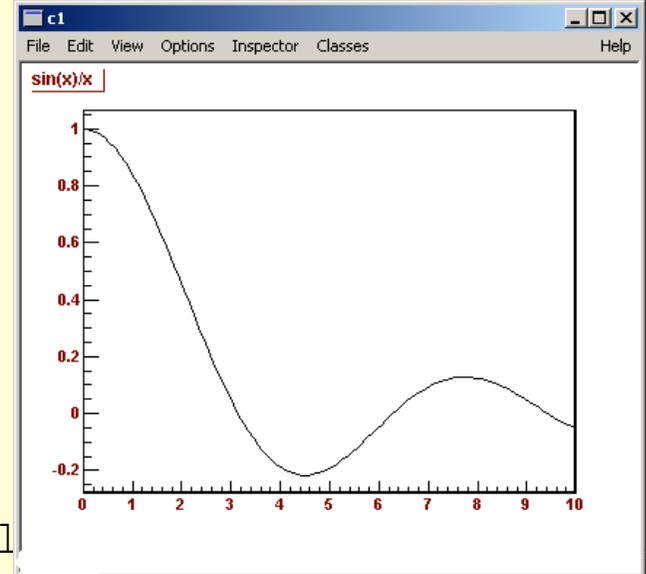
- ◆ PyROOT (former RootPython): bindings to ROOT
 - Python extension module (Boost.Python style)
 - Uses ROOT/CINT dictionary.
 - » Python-side classes created at run-time, no code generation
 - Tool symmetry (close, but not 100%)
 - » Call into CINT from Python and vice versa
- ◆ PyLCGDict: bindings to user (Atlas) classes
 - Started to work on the LCG Dictionary bindings
 - Repeats PyROOT work, but with LCG dictionary

PyROOT Motivation (Pere Mato)

- ◆ Use ROOT classes from Python
 - Without the need of wrapping each class
 - Using the ROOT object dictionary information
- ◆ Provide access to ROOT files and other facilities for non-ROOT applications
- ◆ Proof-of-concept: "Python as Software Bus"
 - Analogous to a "hardware bus," where you can plug a variety of modules (and interface adaptors to other buses) to access new functionality.

Example

```
C:\> python
...
>>> from ROOT import TF1
>>> f1 = TF1('func1', 'sin(x)/x', 0, 10)
>>> f1.Eval(3)
0.047040002686622402
>>> f1.Derivative(3)
-0.34567505667199266
>>> f1.Integral(0, 3)
1.8486525279994681
>>> f1.Draw()
<TCanvas::MakeDefCanvas>: created default
```



- ◆ Note: CINT users can be comfortable with this syntax

Why Python?

- ◆ Interpreted (uses bytecode for speed)
 - No compilation cycle, has run-time module reload
- ◆ Dynamically typed
 - Enables rapid development
- ◆ Simple, consistent syntax
- ◆ Variety of available shells
- ◆ Explorative, interactive interpreter
 - Docstrings, access to dictionaries etc.
- ◆ Popular and well maintained: many users, a plethora of books (50+), tons of available (extension-)modules, etc.

Python Language

- ◆ Expressive, consistent, clear and small
 - Very easy to learn, easy to read
- ◆ Portable, with a version transition strategy
- ◆ Supports multiple paradigms:
 - Object-Oriented
 - Structured
 - Functional
- ◆ Enforces modular design
- ◆ Extensive standard library (>300 modules)

Another example

ntuple1.py

```
import sys, string
infile = open( 'aptuple.txt', 'r' )
lines  = infile.readlines()
title  = lines[0]
labels = string.split( lines[1] )

from ROOT import TFile, TNtuple
outfile = TFile('aptuple.root','RECREATE','NTuple file')
ntuple  = TNtuple('ntuple', title, string.join(labels,':'))

for line in lines[2:]:
    words = string.split( line )
    row = map( float, words )
    apply( ntuple.Fill, row )

outfile.Write()
```

CERN Personnel

Category Division Flag Age Service ...

202 9 15 58 28 0 10 13 4 40 11975

530 5 15 63 33 0 9 13 3 40 10228

316 9 15 56 31 2 9 13 7 40 10730

...

For each line in the file, split the line into a list of words. Convert the list of words into a list of floating-point numbers. And finally, call the Fill() method of the ntuple with it.



Python access to Athena Components

- Python scripting available in 7.0.0
 - Supports access to Algorithms, Services, AlgTools, Properties, etc.
- Initially targetting configuration (converter tool available)
 - Replacement for text JobOptions files
- But also allows run-time access to components
- In conjunction with PyROOT and PyLCGDict provides a flexible and powerful interactive environment
 - PyROOT, PyLCGDict, and GaudiPython will be integrated through the software bus
- Now we need to understand how best to utilize it
 - Balance between Python & C++ Code



ROOT Access to Athena Components?

- Blueprint architecture supports concept of scripting symmetry
 - ROOT/CINT should have symmetric access to Athena components and data model as from Python
 - Currently only available through Python
- The schedule for this is unclear
 - Doesn't appear in SEAL WBS
- Perhaps Rene's talk might provide some insights on this



Multiple Programming Languages

- Central role of Data Dictionary and Python bus provide support for multiple programming languages
 - Python Algorithms already prototyped
 - Support for Java being prototyped
 - Coupling to JAS?
- Is this a confusing proliferation of possibilities?
- Or an example of using the best available tool for a particular job and planning for change over the lifetime of ATLAS?



Summary

- Look at filling the gap between reconstruction and n-tuple analysis in both batch and interactive environments
- Prompt discussion
- Are class libraries (e.g. BaBar/Beta, CLEO/DChain) useful?
- Or is it better to have physics-specific Algorithms/Tools?
- What is the appropriate role of scripting and programming languages?
 - Configuration in Python
 - Rapid prototyping in Python?
- Hope to get people actively working in this area soon

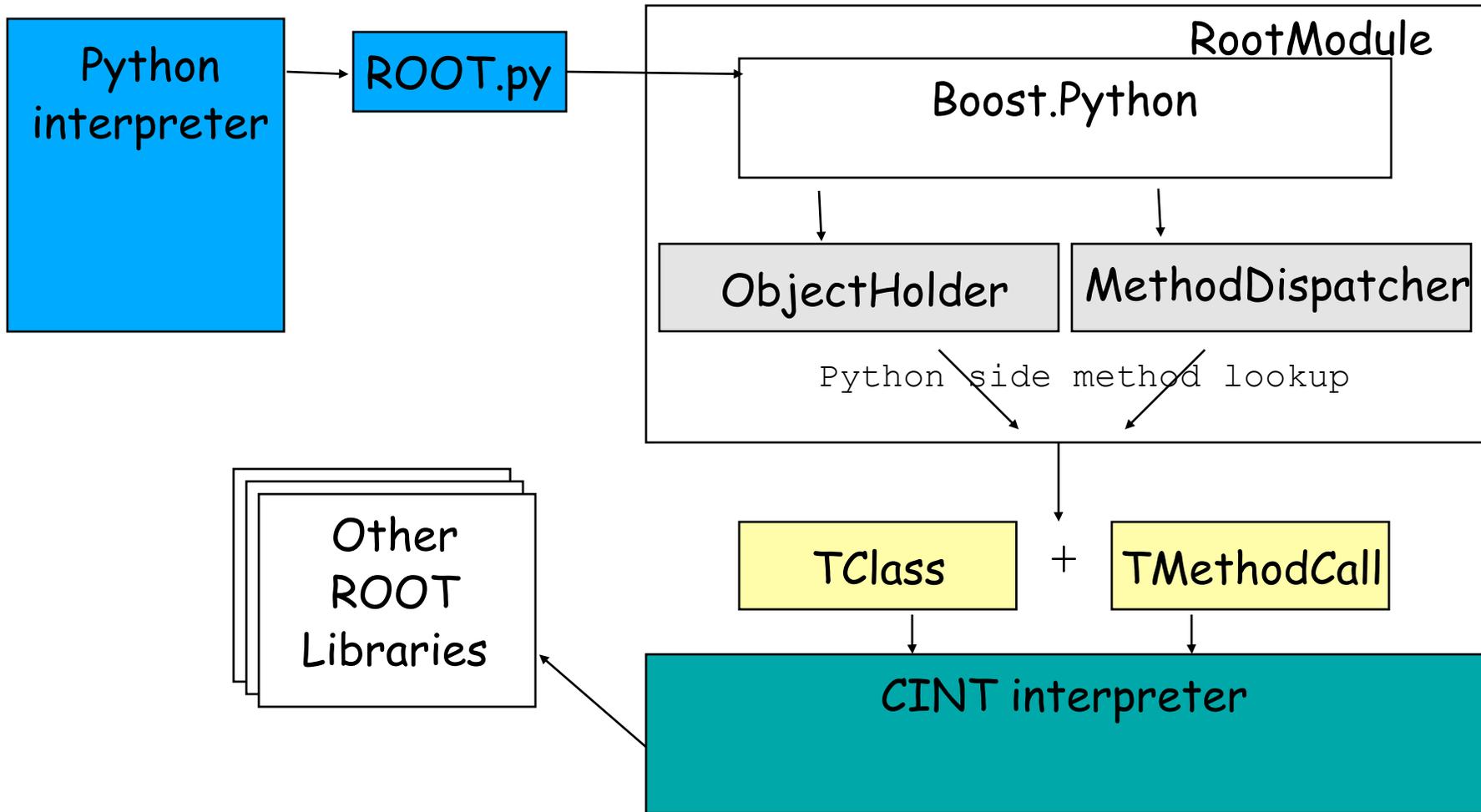


Support Slides

Software requirements

- ◆ Generic, on-demand interface
 - Fill Python class dictionaries at run-time from ROOT/CINT dictionary information
- ◆ Emulate ROOT look and semantics, but close to Python
 - Same ROOT class names, global variables, etc.
 - Python object instantiation, docstrings, namespaces, garbage collection, etc.
 - Support overloading, subclassing etc.
- ◆ The extension should be self-contained
 - Currently: a PyROOTmodule extension module and a ROOT.py usability module (the latter may disappear)
- ◆ The extension should be symmetric in its use

Sketch of the design





More Python/PyROOT Stuff

- Python has proven to be “ideal” at binding to other languages and toolkits
 - Jython, Boost.Python, Qt, LDAP, MySQL, XML, OpenInventor, etc., etc.
- PyROOT is a small package (<1 KLOC)
- Significant Boost.Python expertise at LBNL (not our group)
- Widespread adoption of Python bindings in GRID area
- Other HEP labs (FNAL,SLAC) have shown interest, not just LHC