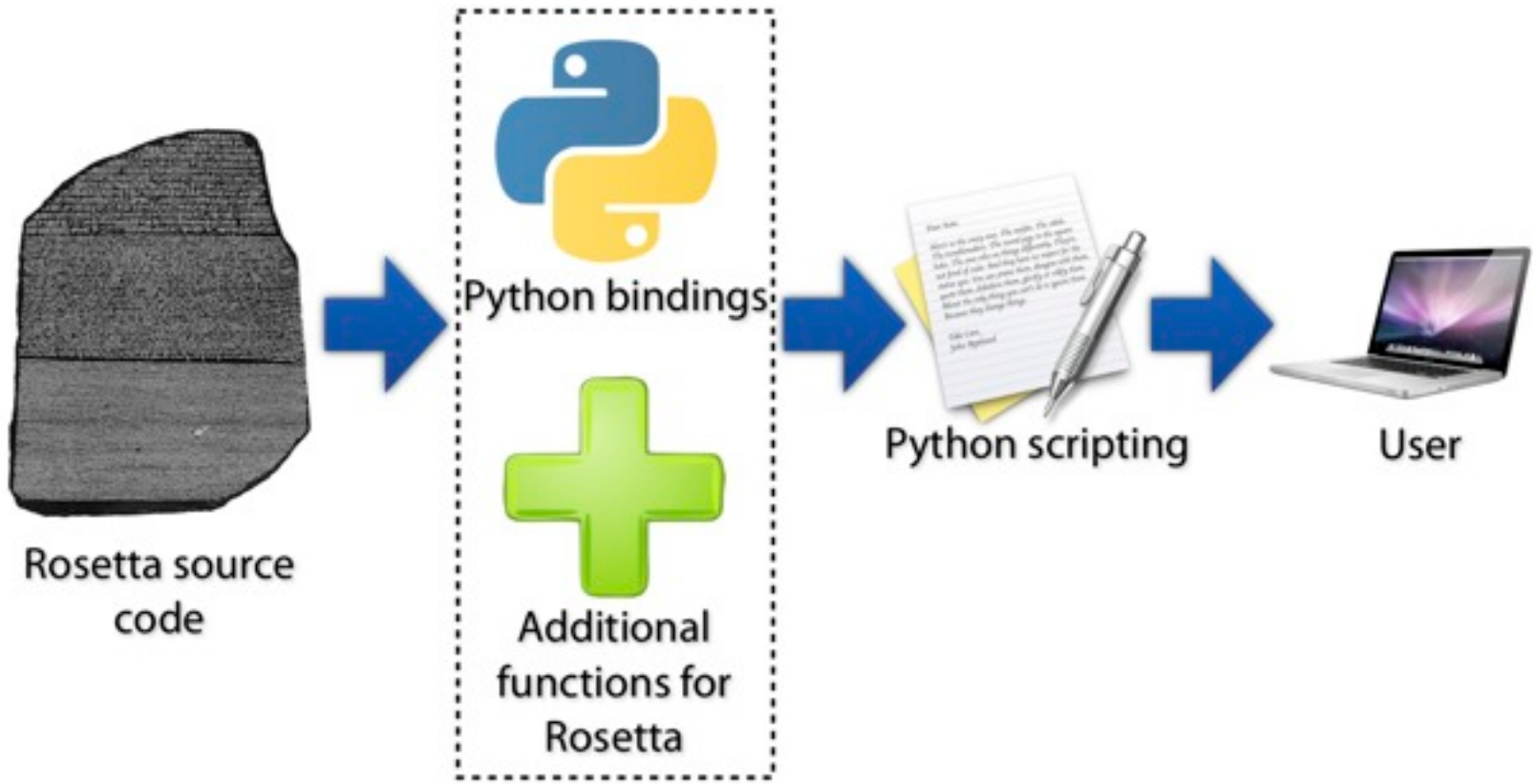
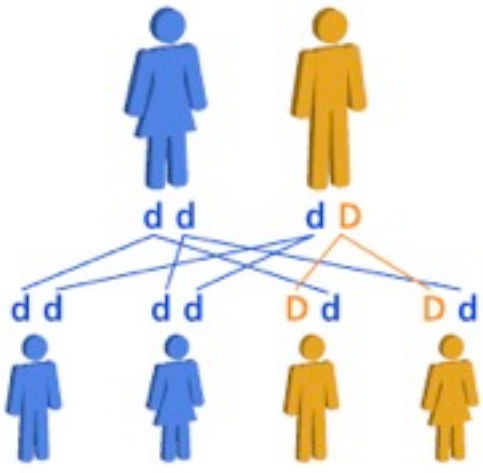
PyRosetta 2.011



Sergey Lyskov & Brian Weitzner
GrayLab @ JHU
RosettaCon 2011

Build System

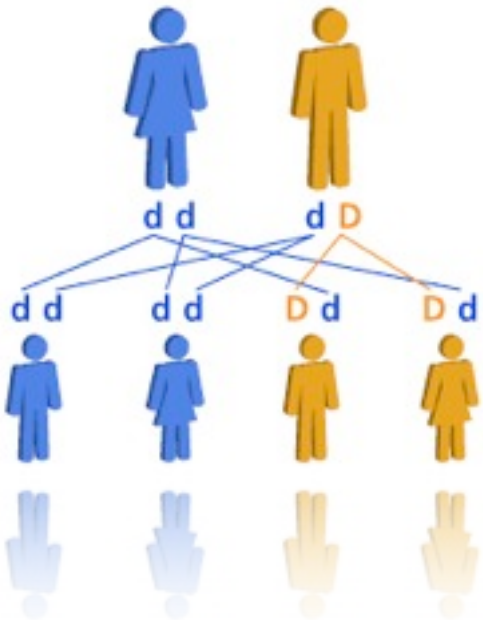
Build System



Inheritance

Build System

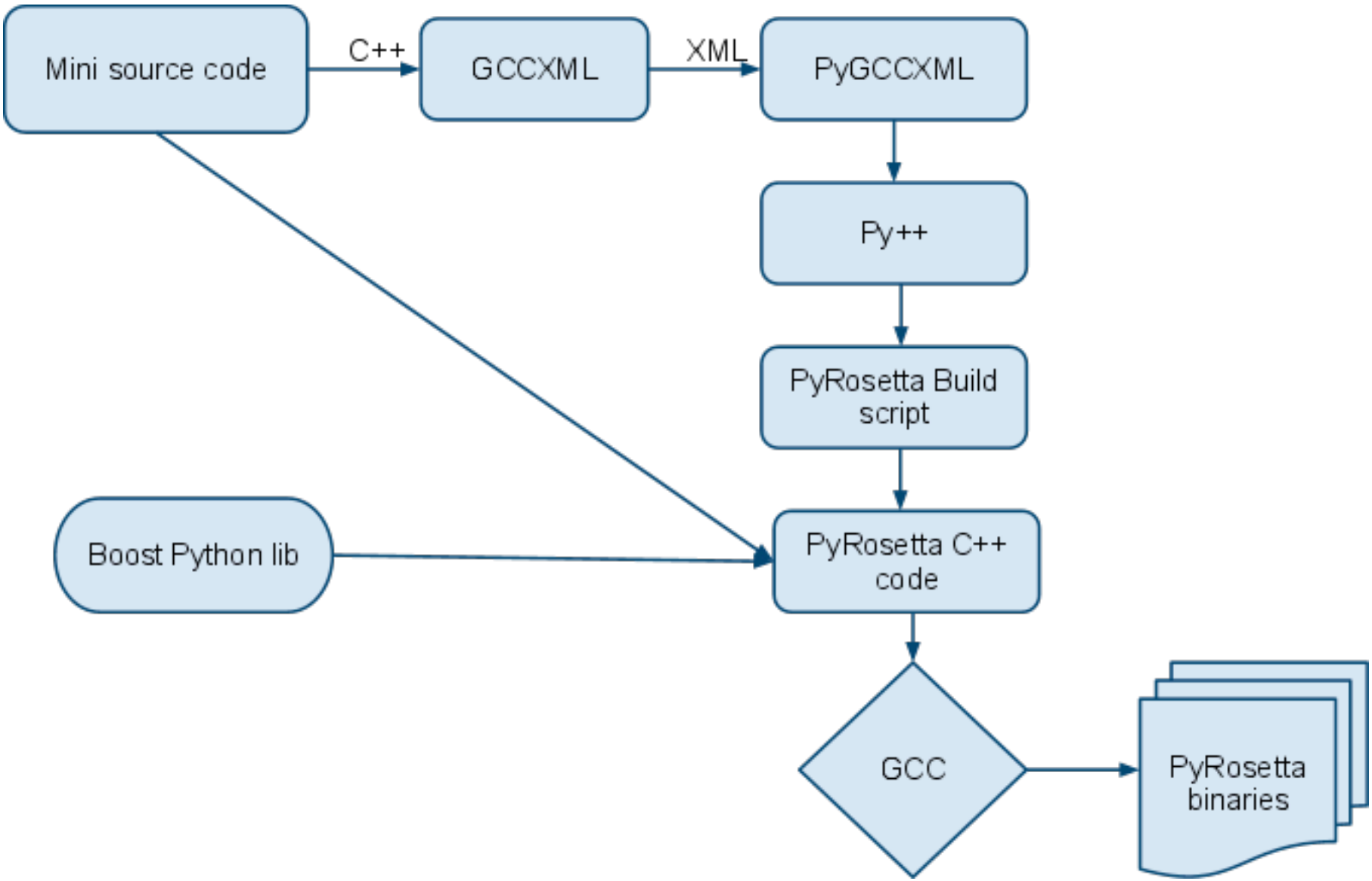
Community



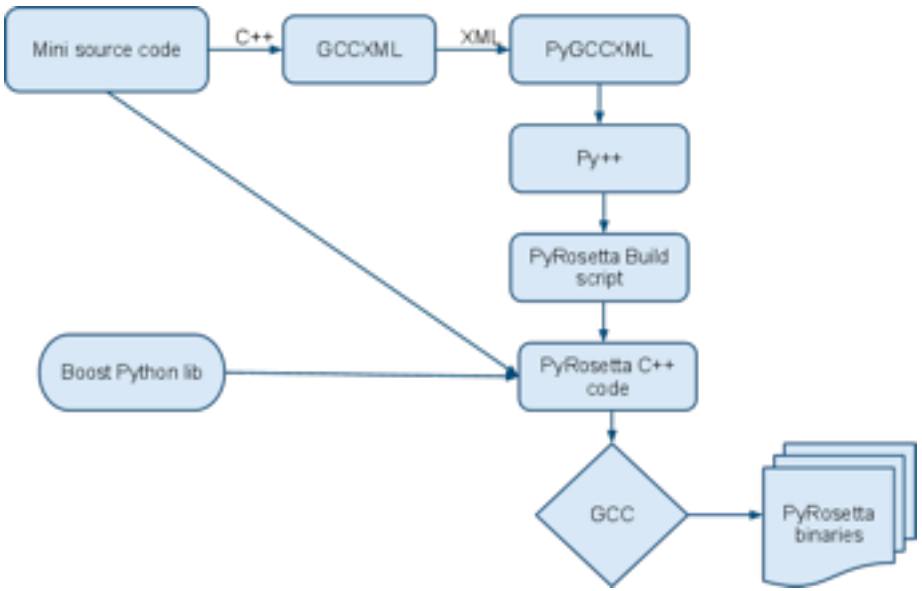
Inheritance

Build System

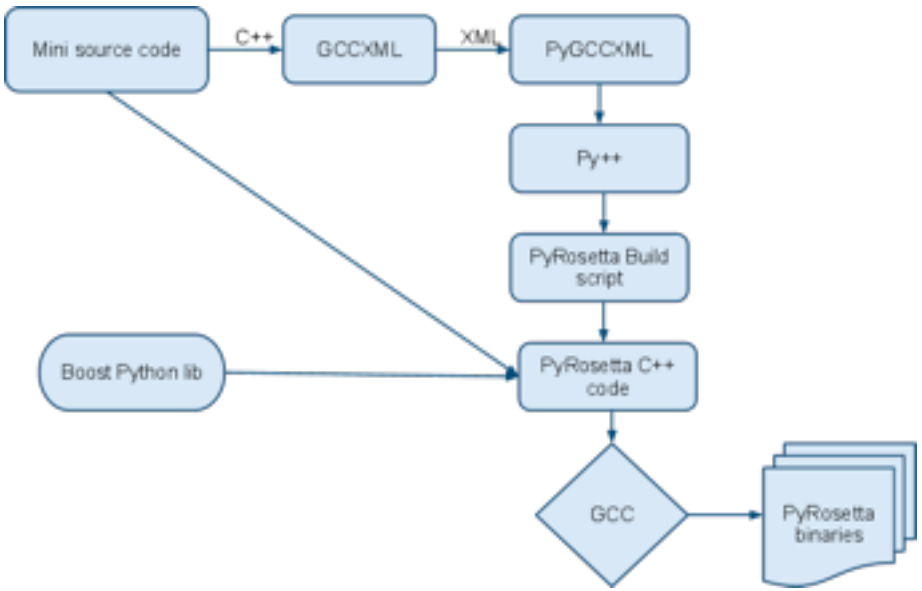
Old Build System



Old Build System

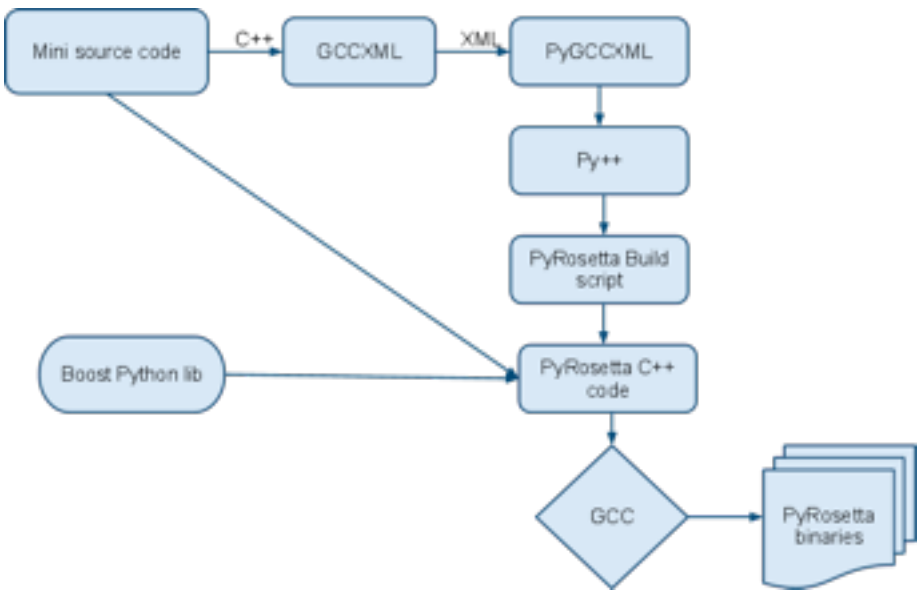


Old Build System



Problems with this approach:

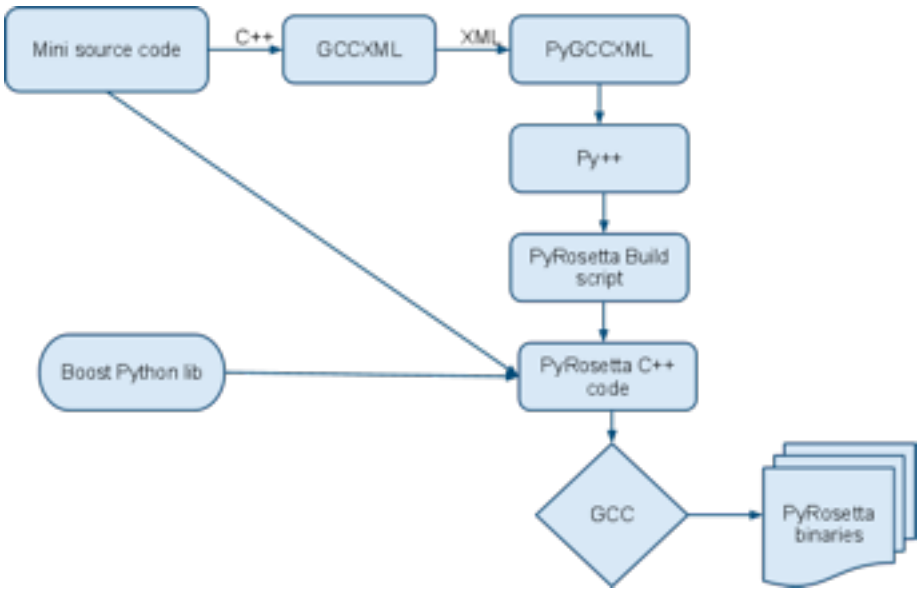
Old Build System



Problems with this approach:

- PyGCCXML and Py++ have a different philosophy than PyRosetta and we depend on them to fix bugs

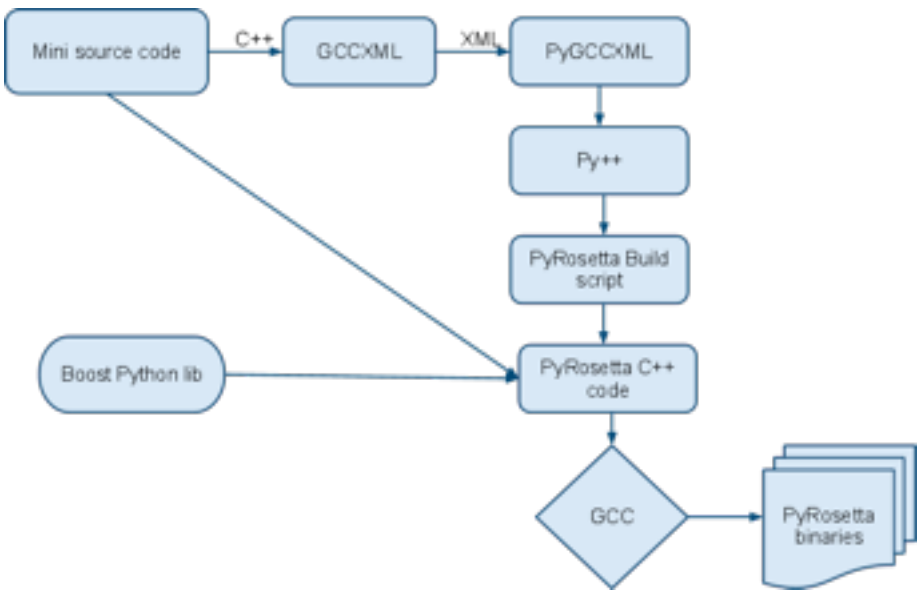
Old Build System



Problems with this approach:

- PyGCCXML and Py++ have a different philosophy than PyRosetta and we depend on them to fix bugs
- No automatic subclassing. User can not inherit and re-use arbitrary Rosetta class.

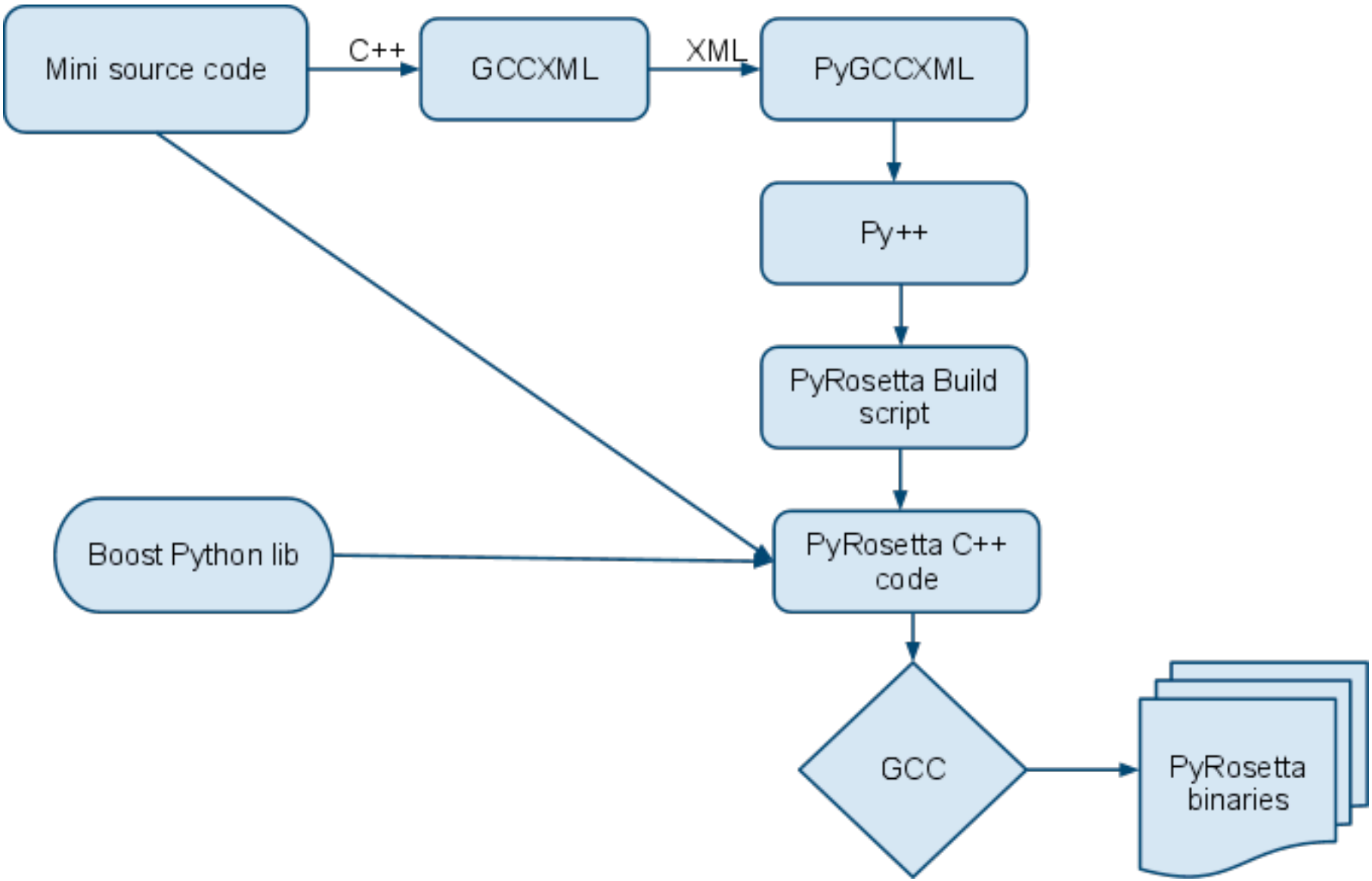
Old Build System



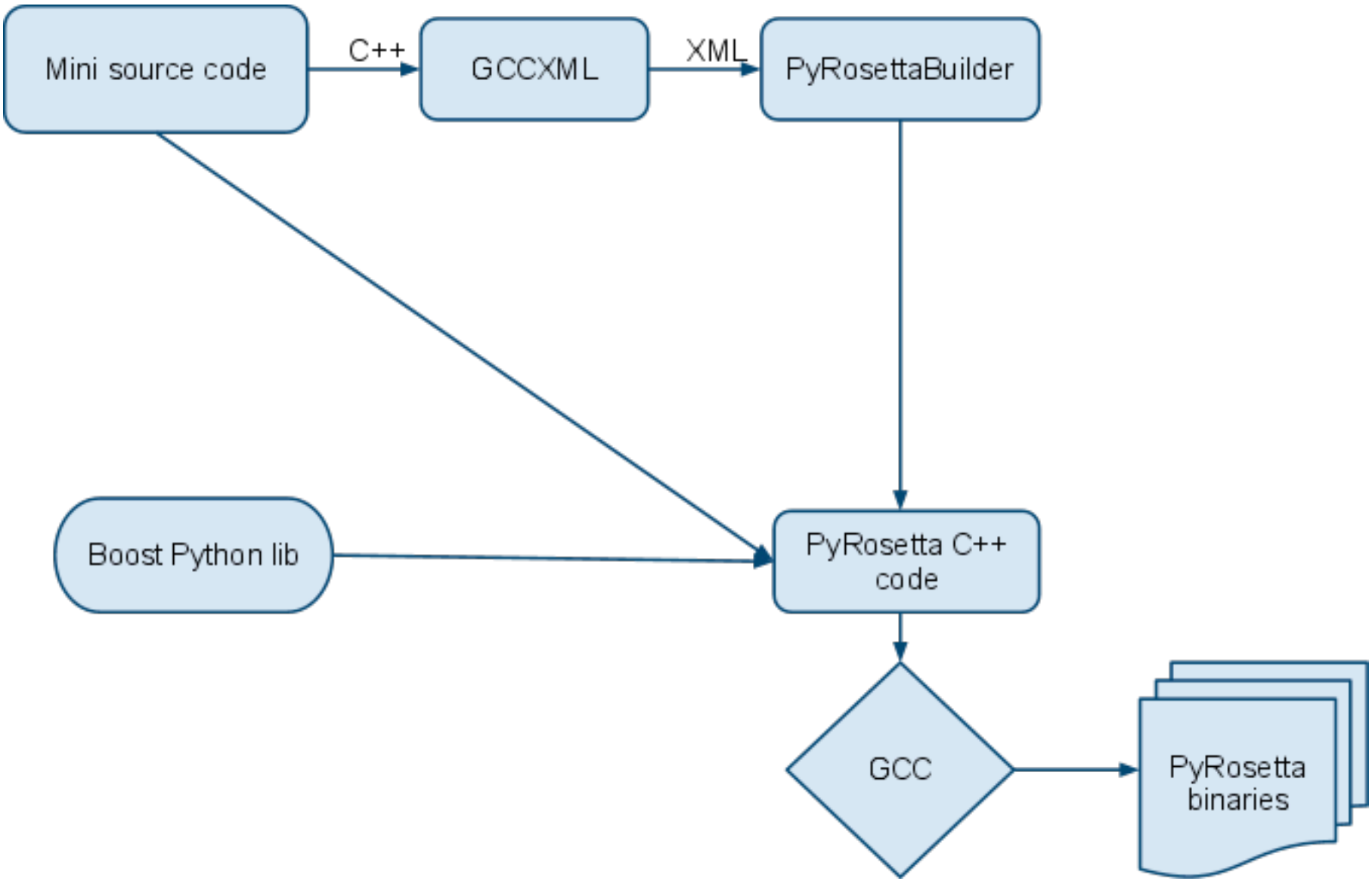
Problems with this approach:

- PyGCCXML and Py++ have a different philosophy than PyRosetta and we depend on them to fix bugs
- No automatic subclassing. User can not inherit and re-use arbitrary Rosetta class.
- Can not use newer compilers, have to use GCCXML therefore limiting our options.

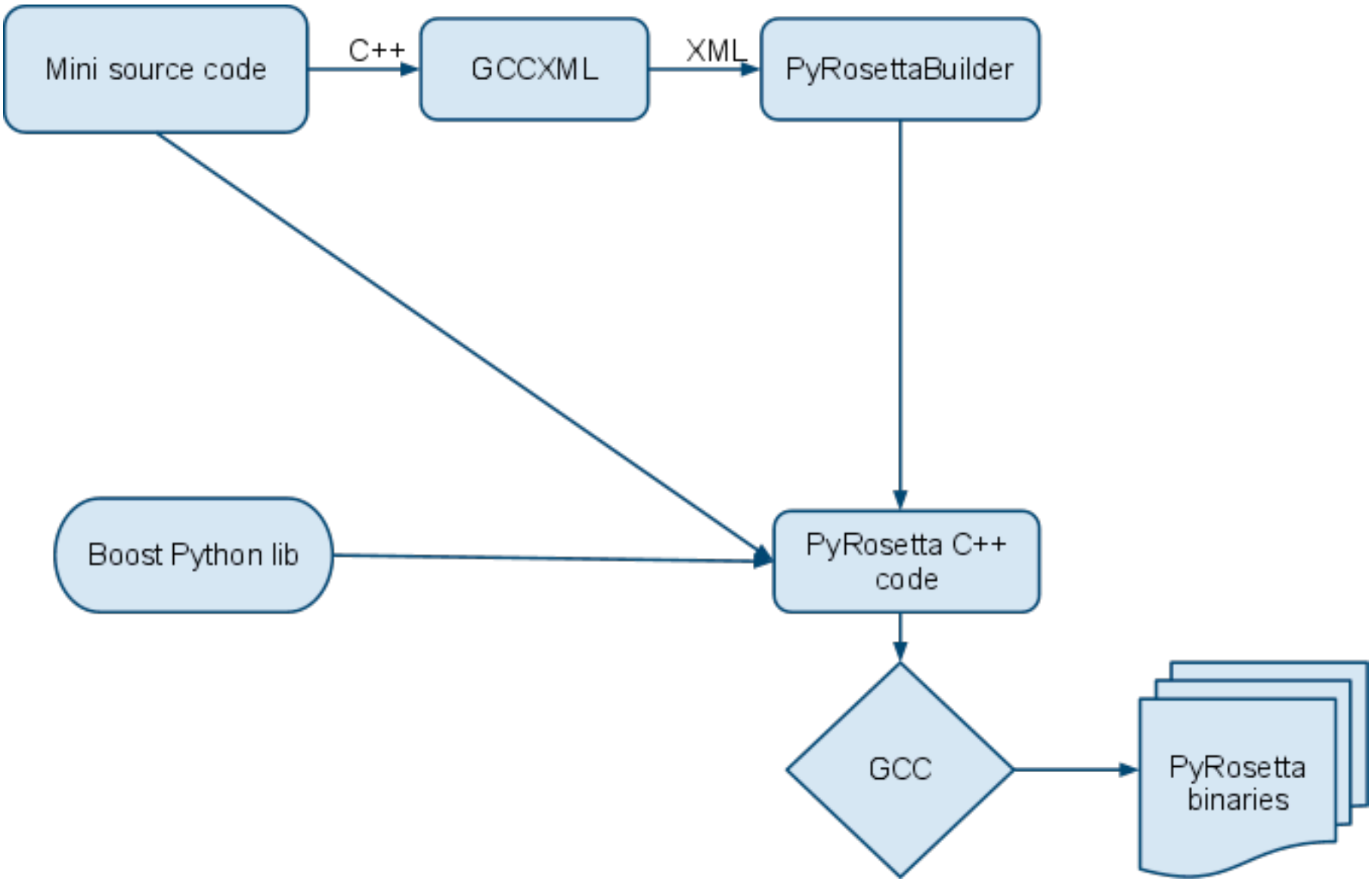
Old Build System



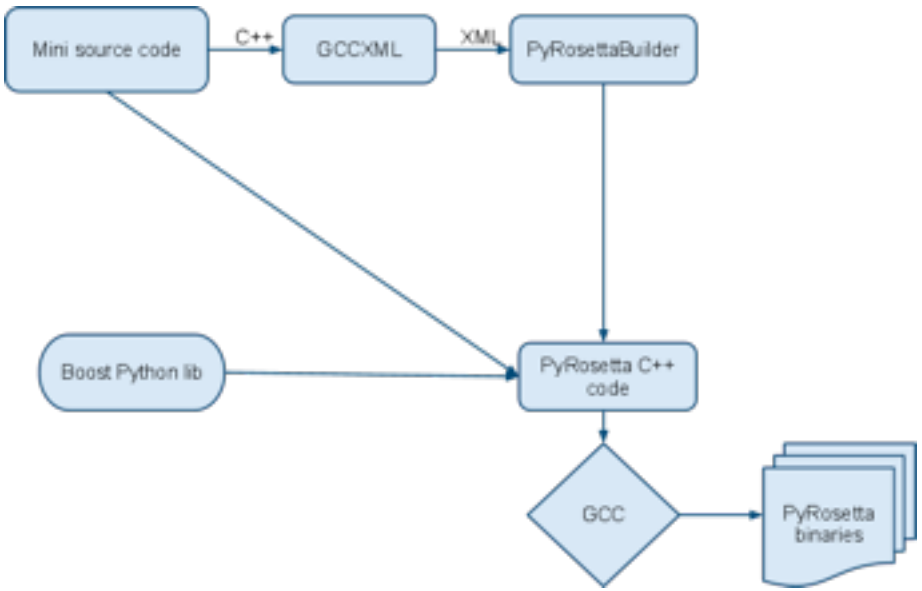
New Build System



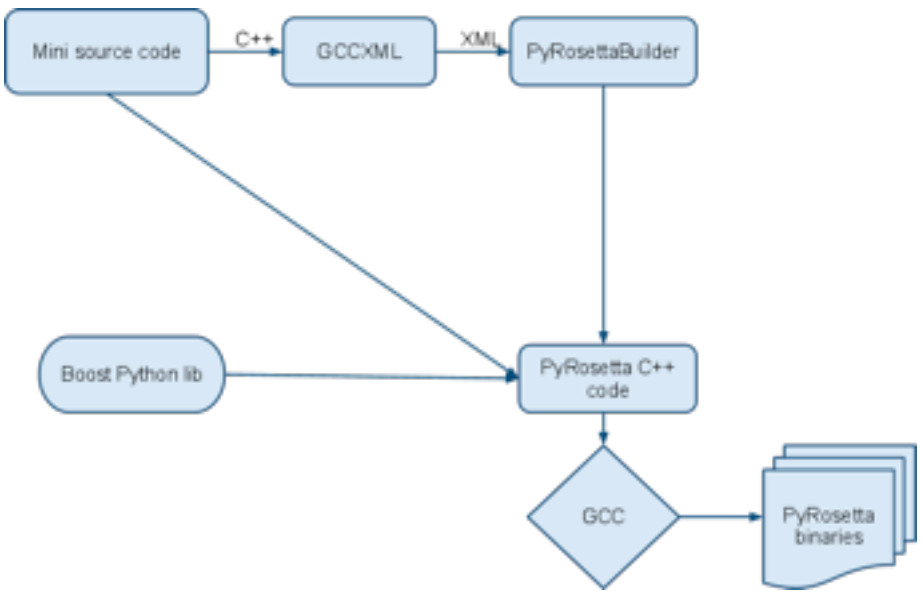
New Build System



New Build System

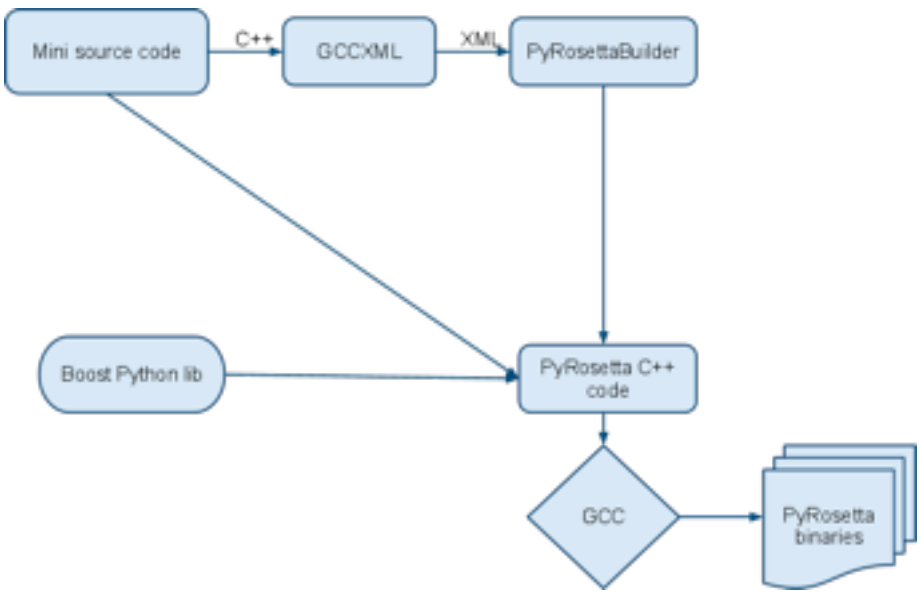


New Build System



Removing PyGCCXML and PyPlusPlus allows us to:

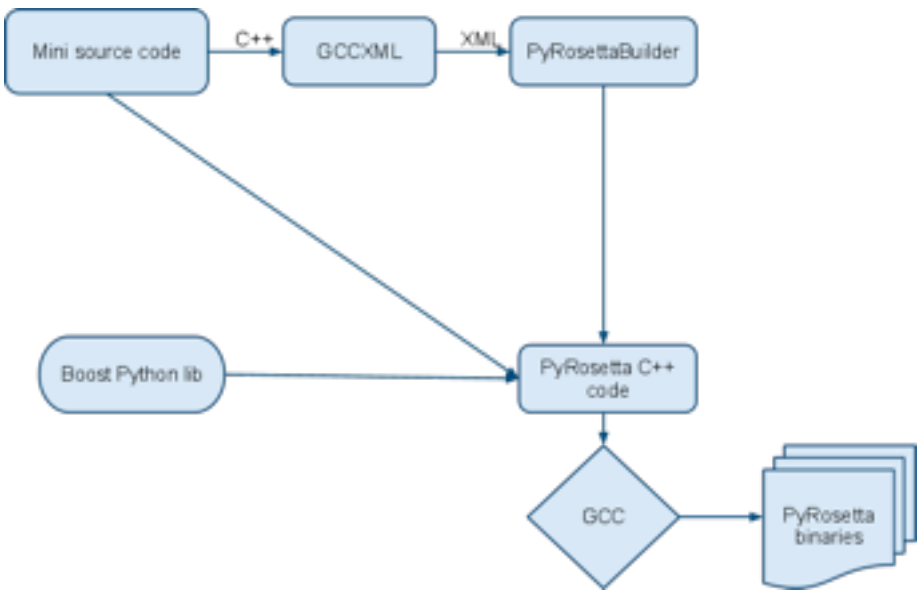
New Build System



Removing PyGCCXML and PyPlusPlus allows us to:

- Parallel builds. Full rebuild on modern workstation takes only ~20-30min. (was 6hrs before!)

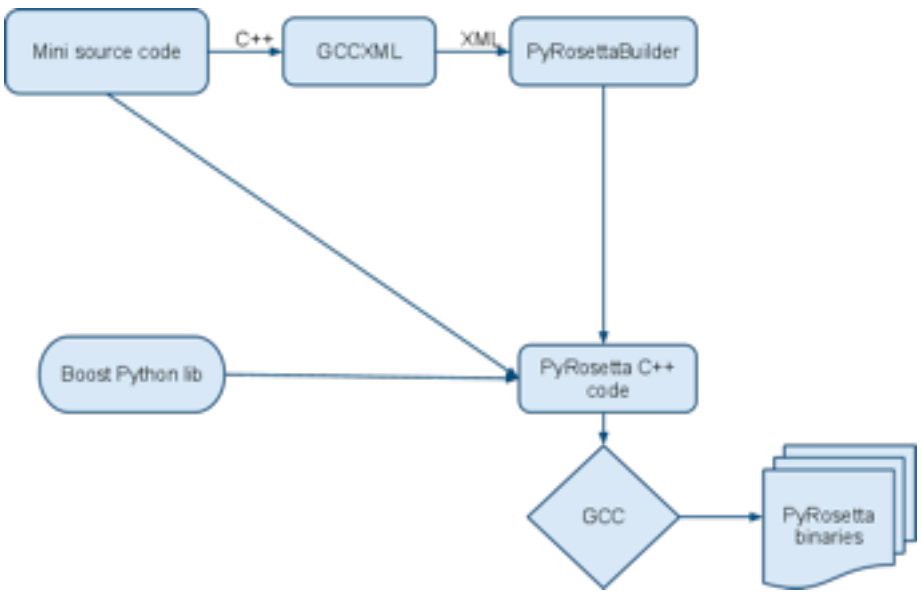
New Build System



Removing PyGCCXML and PyPlusPlus allows us to:

- Parallel builds. Full rebuild on modern workstation takes only ~20-30min. (was 6hrs before!)
- PyRosetta classes are first class citizens

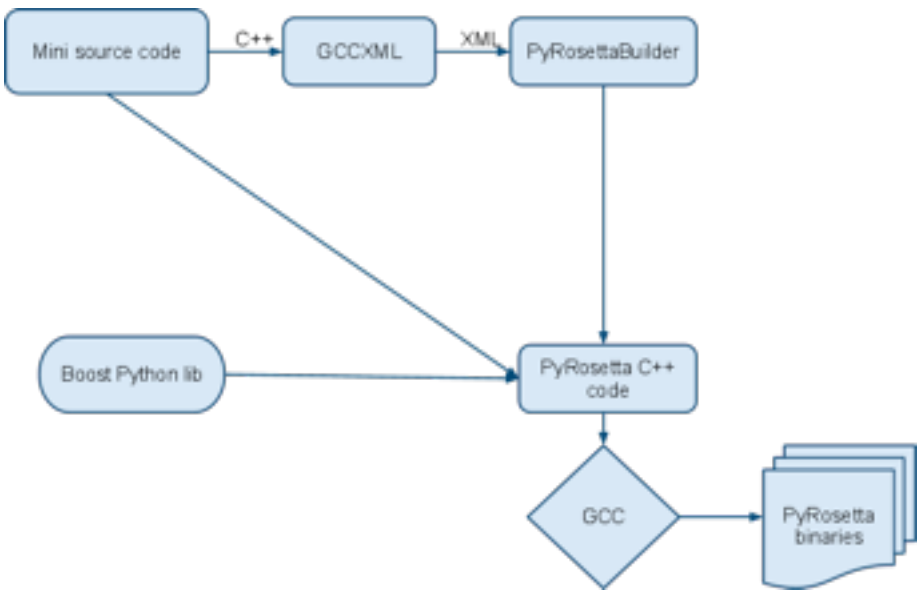
New Build System



Removing PyGCCXML and PyPlusPlus allows us to:

- Parallel builds. Full rebuild on modern workstation takes only ~20-30min. (was 6hrs before!)
- PyRosetta classes are first class citizens
- Clang support

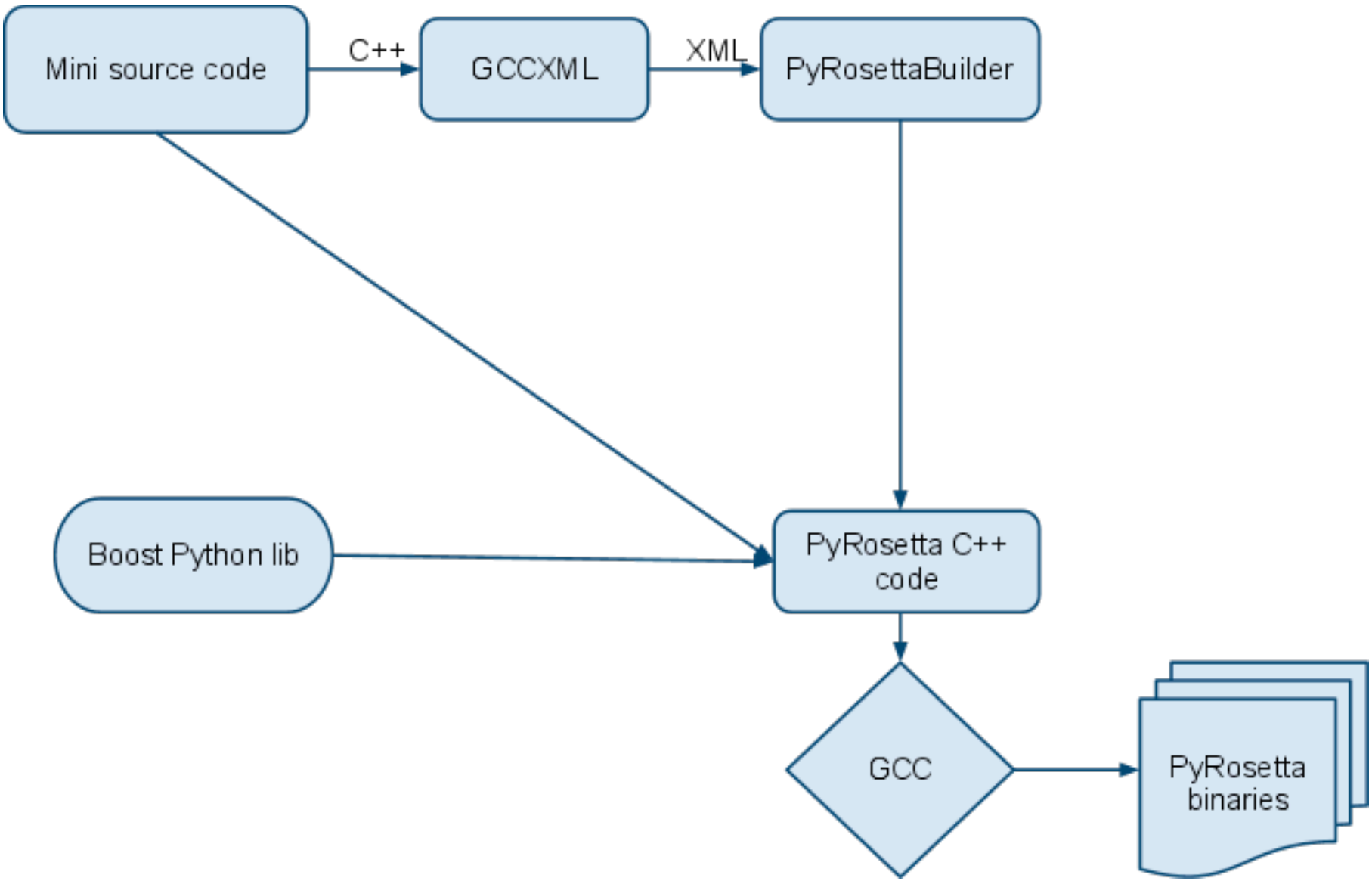
New Build System



Removing PyGCCXML and PyPlusPlus allows us to:

- Parallel builds. Full rebuild on modern workstation takes only ~20-30min. (was 6hrs before!)
- PyRosetta classes are first class citizens
- Clang support
- Future: drop GCCXML and move to Clang++ when it is mature.

New Build System



Setting up a build environment for PyRosetta is **SUPER** easy

Setting up a build environment for PyRosetta is **SUPER** easy

```
$ svn co https://svn.rosettacommons.org/source/trunk/PyRosetta.develop
$ ./DeployPyRosetta.py
```

This will:

Setting up a build environment for PyRosetta is **SUPER** easy

```
$ svn co https://svn.rosettacommons.org/source/trunk/PyRosetta.develop
$ ./DeployPyRosetta.py
```

This will:

Setting up a build environment for PyRosetta is **SUPER** easy

```
$ svn co https://svn.rosettacommons.org/source/trunk/PyRosetta.develop
$ ./DeployPyRosetta.py
```

This will:

- Build and setup (locally, no root access is required) CMake, GCCXML, Boost, Py++, PyGCCXML [and currently in the works Clang]

Setting up a build environment for PyRosetta is **SUPER** easy

```
$ svn co https://svn.rosettacommons.org/source/trunk/PyRosetta.develop
$ ./DeployPyRosetta.py
```

This will:

- Build and setup (locally, no root access is required) CMake, GCCXML, Boost, Py++, PyGCCXML [and currently in the works Clang]
- Check out copy of mini, create ‘BuildPyRosetta.sh’ script inside it (which has reference to build env. and later can be used with any copy of mini)

Setting up a build environment for PyRosetta is **SUPER** easy

```
$ svn co https://svn.rosettacommons.org/source/trunk/PyRosetta.develop
$ ./DeployPyRosetta.py
```

This will:

- Build and setup (locally, no root access is required) CMake, GCCXML, Boost, Py++, PyGCCXML [and currently in the works Clang]
- Check out copy of mini, create ‘BuildPyRosetta.sh’ script inside it (which has reference to build env. and later can be used with any copy of mini)

Setting up a build environment for PyRosetta is **SUPER** easy

```
$ svn co https://svn.rosettacommons.org/source/trunk/PyRosetta.develop
$ ./DeployPyRosetta.py
```

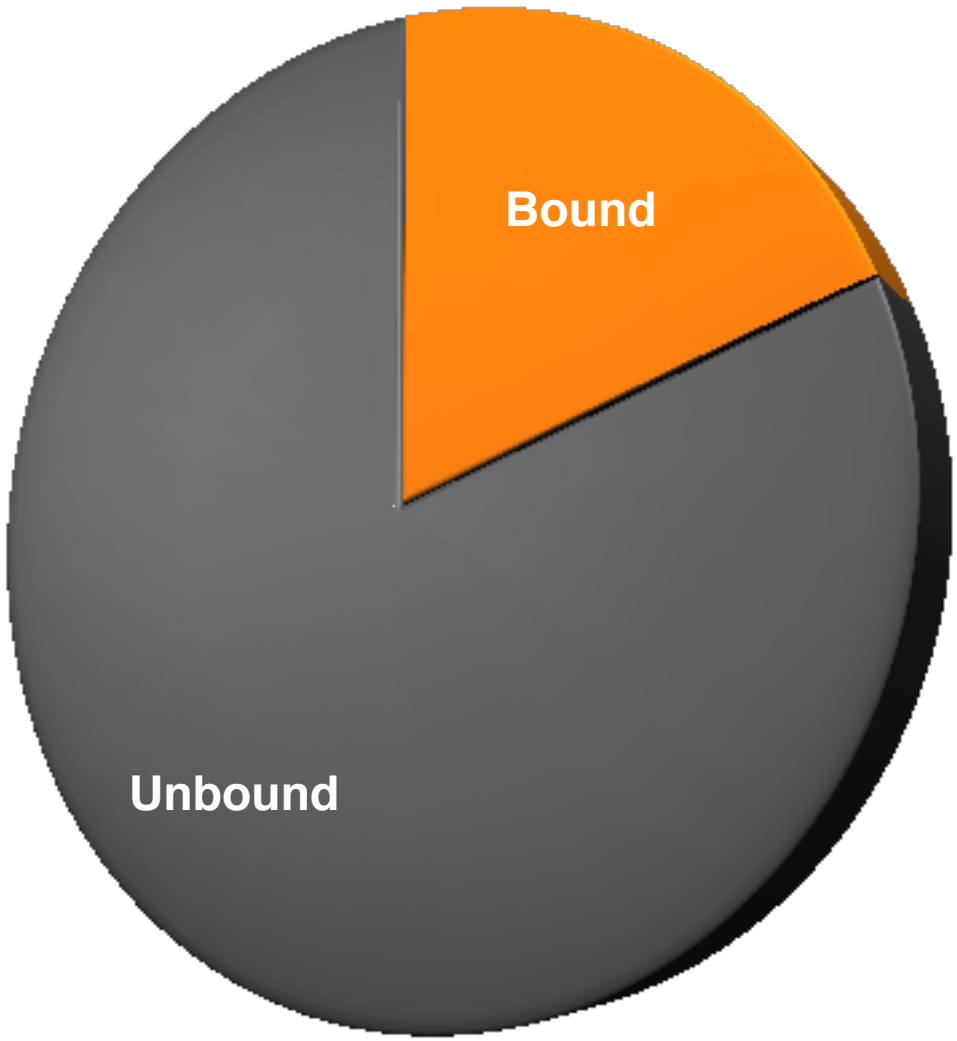
This will:

- Build and setup (locally, no root access is required) CMake, GCCXML, Boost, Py++, PyGCCXML [and currently in the works Clang]
- Check out copy of mini, create ‘BuildPyRosetta.sh’ script inside it (which has reference to build env. and later can be used with any copy of mini)
- Build PyRosetta.

Classes available in PyRosetta

Classes available in PyRosetta

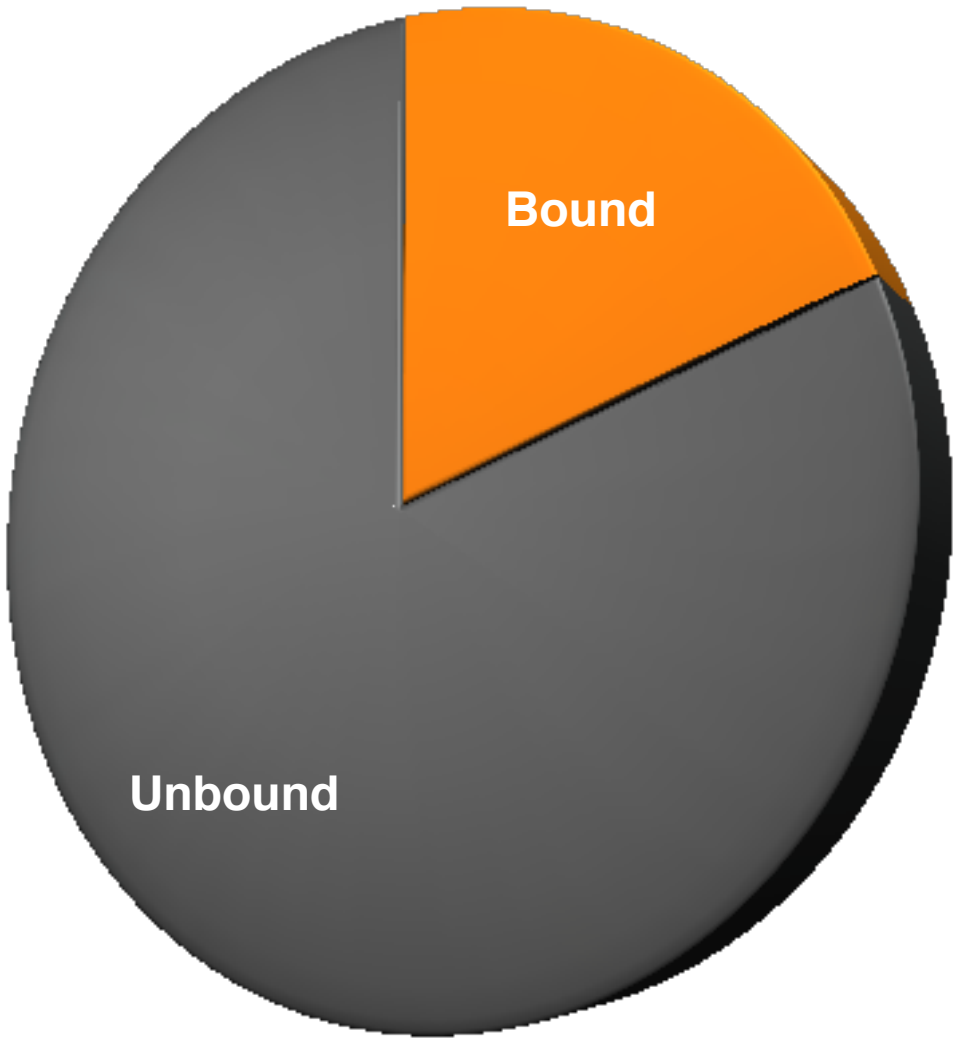
PyRosetta 1.1 provided access to 580 Rosetta classes



Classes available in PyRosetta

PyRosetta 1.1 provided access to 580 Rosetta classes

PyRosetta 2.0's new build system has allowed us to bind many more classes

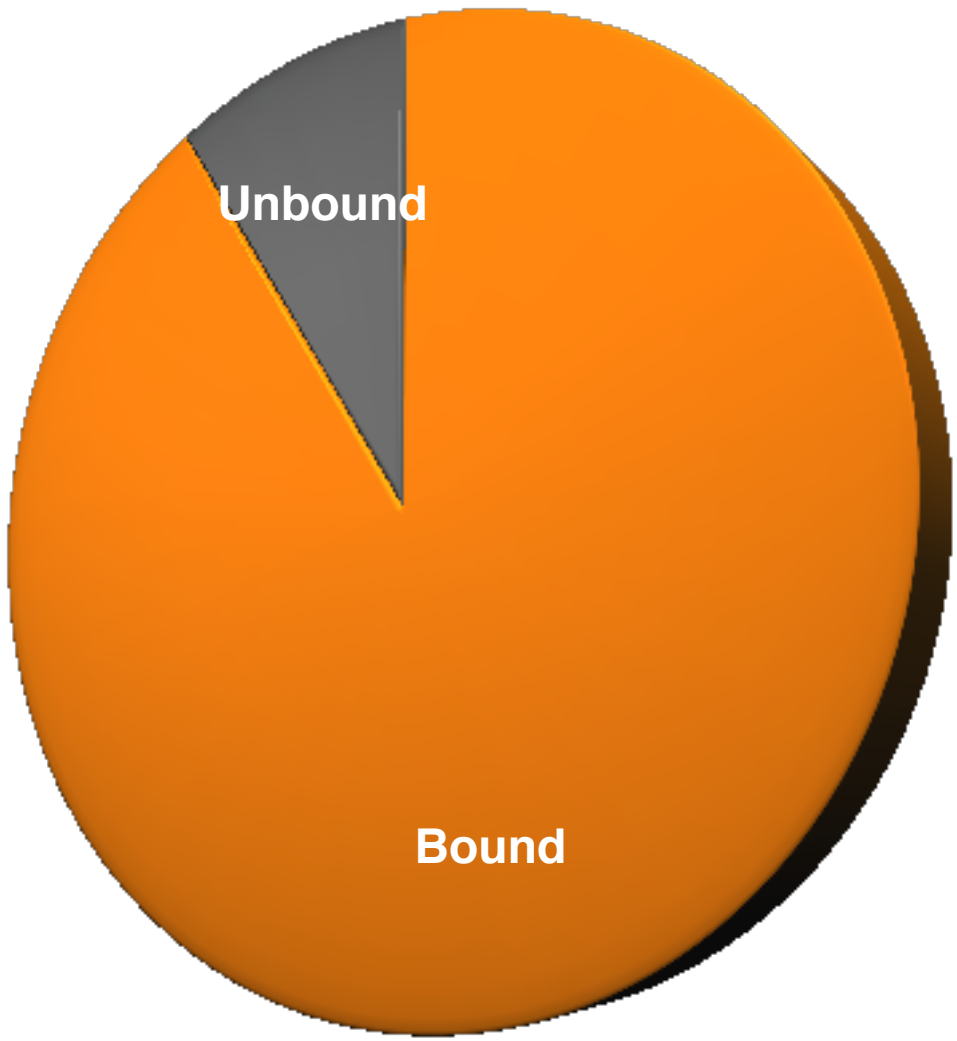


Classes available in PyRosetta

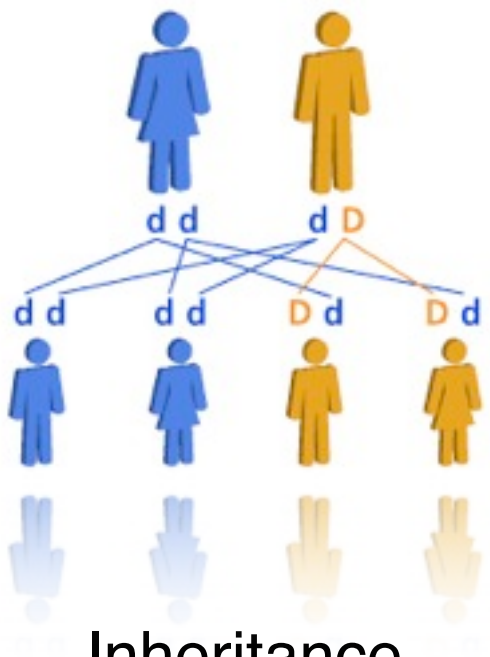
PyRosetta 1.1 provided access to 580 Rosetta classes

PyRosetta 2.0's new build system has allowed us to bind many more classes

Over 90% of Rosetta's 3,000 classes are available in PyRosetta

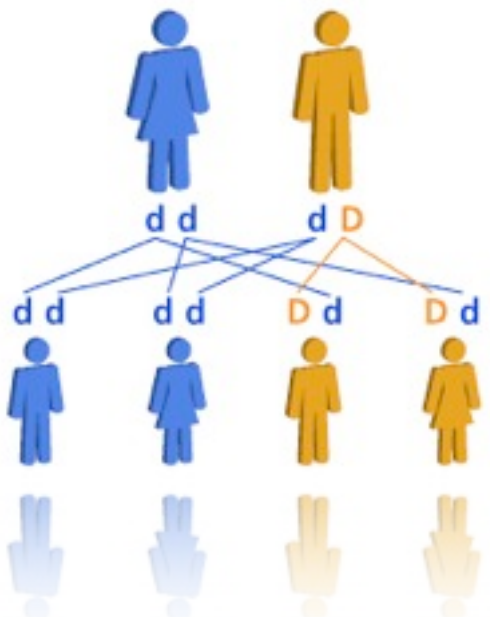


Build System



Inheritance

Demo



Inheritance

Community

We test and provide binaries for:

We test and provide binaries for:

ubuntu

32/64 Bit

Mac OS X Snow Leopard

Scientific Linux 64

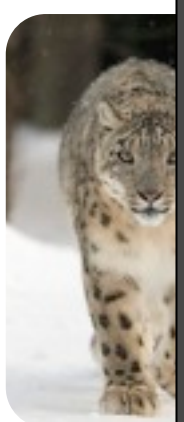
Mac OS X Lion

We test and provide binaries for:

ubur

32/64

ux 64



Mac OS X Snow Leopard

Mac OS X Lion

We test and provide binaries for:

ubuntu

32/64 Bit

Scientific Linux 64

Mac OS X Snow Leopard

Mac OS X Lion

Do people use PyRosetta?

Do people use PyRosetta?

- Each day, we get approximately 45 unique visitors to our web site (www.PyRosetta.org)

Do people use PyRosetta?

- Each day, we get approximately 45 unique visitors to our web site (www.PyRosetta.org)
- Since January, over 400 new academic licenses were issued. This means there are 2-3 new PyRosetta users every day!

Do people use PyRosetta?

- Each day, we get approximately 45 unique visitors to our web site (www.PyRosetta.org)
- Since January, over 400 new academic licenses were issued. This means there are 2-3 new PyRosetta users every day!
- Each week we getting several posts at RosettaCommons forums as well as personal emails.

Welcoming New Users

- We now can use mini trunk version which allows us to help users much faster.
- PyAsserts
- Help!

Summary, things that improved since last year

Summary, things that improved since last year

- New build system makes almost all C++ objects accessible in Python

Summary, things that improved since last year

- New build system makes almost all C++ objects accessible in Python
- Support for proper base classes and subclassing in Python: **It is now possible to extend Rosetta via Python**

Summary, things that improved since last year

- New build system makes almost all C++ objects accessible in Python
- Support for proper base classes and subclassing in Python: **It is now possible to extend Rosetta via Python**
- Steadily increasing community of PyRosetta users

Future goals

Future goals

- Separate PyRosetta source code generation and compiling phases. This should allow us to port PyRosetta to platforms that are unfriendly to GNU tool-chain.

Future goals

- Separate PyRosetta source code generation and compiling phases. This should allow us to port PyRosetta to platforms that are unfriendly to GNU tool-chain.
- Extend Library of scripts aka ‘Protocol demos’. (Would be great if PyRosetta demo could be added to publishing requirements)

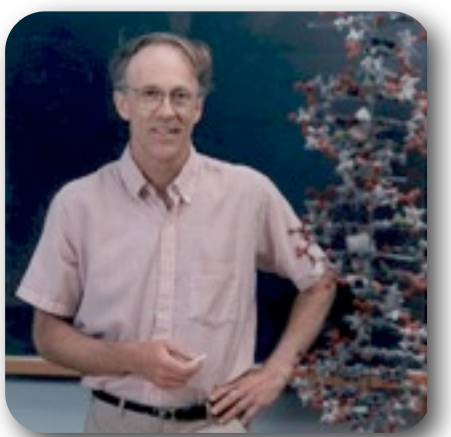
Future goals

- Separate PyRosetta source code generation and compiling phases. This should allow us to port PyRosetta to platforms that are unfriendly to GNU tool-chain.
- Extend Library of scripts aka ‘Protocol demos’. (Would be great if PyRosetta demo could be added to publishing requirements)
- Build-time test for detecting commits that break PyRosetta (undefined functions, unconventional namespaces, etc.)

Acknowledgments

Evan Baugh

Brian Weitzner



Robert Schleif

Jeffrey Gray

Huge Thanks to **Steven Lewis**
who constantly answers questions on RosettaCommons forums and redirects PyRosetta specific questions to me!

Without his help we would be able to provide only a fraction of support available now.

Thank you!

Buckle up your seatbelts, technical slides ahead!

How did we solve subclassing problem?

How did we solve subclassing problem?

The problem Function argument in C++ is allocated on stack; to appear in PyRosetta it needs to be copied to heap-allocated-memory. This leads to:

How did we solve subclassing problem?

The problem Function argument in C++ is allocated on stack; to appear in PyRosetta it needs to be copied to heap-allocated-memory. This leads to:

A. **Slooooowness** (Some objects are large so copying takes time!)

How did we solve subclassing problem?

The problem Function argument in C++ is allocated on stack; to appear in PyRosetta it needs to be copied to heap-allocated-memory. This leads to:

A. **Slooooowness** (Some objects are large so copying takes time!)

B. In C++ if you pass object by reference - you can modify it, but if you work with a copy of an object - you can't do it! This makes **impossible to write** functions like **mover.apply** and many others!

How did we solve subclassing problem?

The problem Function argument in C++ is allocated on stack; to appear in PyRosetta it needs to be copied to heap-allocated-memory. This leads to:

A. **Slooooowness** (Some objects are large so copying takes time!)

B. In C++ if you pass object by reference - you can modify it, but if you work with a copy of an object - you can't do it! This makes **impossible to write** functions like **mover.apply** and many others!

Perfect Solution: all function arguments are OP. Requires to change every single function in mini! - this would be Perfect in a Perfect world but in Practical world it's Practically impossible!

How did we solve subclassing problem?

The problem Function argument in C++ is allocated on stack; to appear in PyRosetta it needs to be copied to heap-allocated-memory. This leads to:

A. **Slooooowness** (Some objects are large so copying takes time!)

B. In C++ if you pass object by reference - you can modify it, but if you work with a copy of an object - you can't do it! This makes **impossible to write** functions like **mover.apply** and many others!

Perfect Solution: all function arguments are OP. Requires to change every single function in mini! - this would be Perfect in a Perfect world but in Practical world it's Practically impossible!

Interim Solution: Subclass original class, overload virtual functions, convert arguments to AP, and write code for each class that would allow to subclass it in PyRosetta. Problem with this approach: even if we write this for ~3000 classes it would be impossible to keep up with mainstream changes!

How did we solve subclassing problem?

The problem Function argument in C++ is allocated on stack; to appear in PyRosetta it needs to be copied to heap-allocated-memory. This leads to:

A. **Slooooowness** (Some objects are large so copying takes time!)

B. In C++ if you pass object by reference - you can modify it, but if you work with a copy of an object - you can't do it! This makes **impossible to write** functions like **mover.apply** and many others!

Perfect Solution: all function arguments are OP. Requires to change every single function in mini! - this would be Perfect in a Perfect world but in Practical world it's Practically impossible!

Interim Solution: Subclass original class, overload virtual functions, convert arguments to AP, and write code for each class that would allow to subclass it in PyRosetta. Problem with this approach: even if we write this for ~3000 classes it would be impossible to keep up with mainstream changes!

The Solution

The Solution

- For each class in mini automatically generate subclass and overload all public virtual functions.

The Solution

- For each class in mini automatically generate subclass and overload all public virtual functions.

- For each overloaded function examine the types of arguments that it receives and generate code to convert it to AP. Then generate code for new virtual functions that have AP for all their arguments. The resulting function will be overload-able in PyRosetta.

The Solution

- For each class in mini automatically generate subclass and overload all public virtual functions.

- For each overloaded function examine the types of arguments that it receives and generate code to convert it to AP. Then generate code for new virtual functions that have AP for all their arguments. The resulting function will be overload-able in PyRosetta.

- Our final code can do the following automatically:

The Solution

- For each class in mini automatically generate subclass and overload all public virtual functions.

- For each overloaded function examine the types of arguments that it receives and generate code to convert it to AP. Then generate code for new virtual functions that have AP for all their arguments. The resulting function will be overload-able in PyRosetta.

- Our final code can do the following automatically:

    - Subclass ~2700 mini classes (replicate public constructors and call original constructors).

The Solution

- For each class in mini automatically generate subclass and overload all public virtual functions.

- For each overloaded function examine the types of arguments that it receives and generate code to convert it to AP. Then generate code for new virtual functions that have AP for all their arguments. The resulting function will be overload-able in PyRosetta.

- Our final code can do the following automatically:

    - Subclass ~2700 mini classes (replicate public constructors and call original constructors).

    - Overload all public virtual functions, examine arguments and convert them to AP so original virtual functions now act as 'proxy' between mini and Python.

The Solution

- For each class in mini automatically generate subclass and overload all public virtual functions.

- For each overloaded function examine the types of arguments that it receives and generate code to convert it to AP. Then generate code for new virtual functions that have AP for all their arguments. The resulting function will be overload-able in PyRosetta.

- Our final code can do the following automatically:

    - Subclass ~2700 mini classes (replicate public constructors and call original constructors).

    - Overload all public virtual functions, examine arguments and convert them to AP so original virtual functions now act as 'proxy' between mini and Python.

    - Adapt all these subclasses so they can be used as ‘call-back’ structure to allow subclassing in Python.

Explain How Subclassing work

Subclasses in PyRosetta

Subclasses in PyRosetta

- Before in PyRosetta 1.1 to allow subclassing in PyRosetta hand written was needed. For average class it took ~1hr of work plus it would require some effort to keep it in-sync with the main trunk. Now take in to account number of classes in mini and its easy to see how this approach became a problem.

Subclasses in PyRosetta

- Before in PyRosetta 1.1 to allow subclassing in PyRosetta hand written was needed. For average class it took ~1hr of work plus it would require some effort to keep it in-sync with the main trunk. Now take in to account number of classes in mini and its easy to see how this approach became a problem.
- Also, due to constraints of build system it was hard-impossible to set correct base which lead to situation when inherited classes in PyRosetta would lack some virtual functions of a base classes.

Subclasses in PyRosetta

- Before in PyRosetta 1.1 to allow subclassing in PyRosetta hand written was needed. For average class it took ~1hr of work plus it would require some effort to keep it in-sync with the main trunk. Now take in to account number of classes in mini and its easy to see how this approach became a problem.
- Also, due to constraints of build system it was hard-impossible to set correct base which lead to situation when inherited classes in PyRosetta would lack some virtual functions of a base classes.

- Now all this done automatically for all classes in mini. So all classes could be inherited in PyRosetta.