Convert-XY: Type-Safe Interchange of C++ and Python Containers for NumPy Extensions

Damian Eads[1,3] and Edward Rosten[2]
1. Department of Computer Science, University of California, Santa Cruz
2. Department of Engineering, University of Cambridge
3. Los Alamos National Laboratory

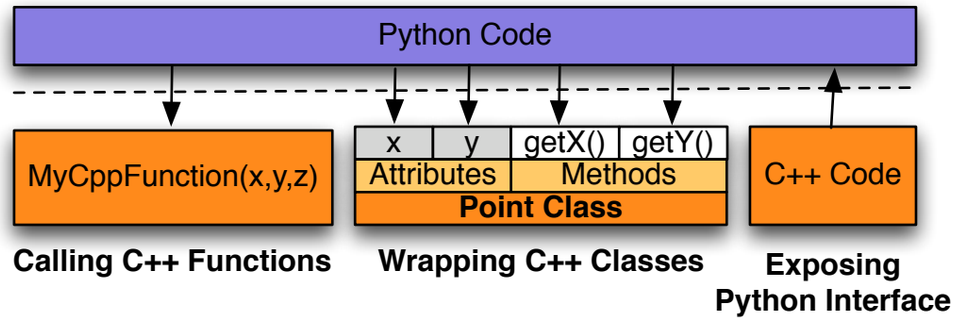
Why *extensions*? Why not do it ***all*** in Python?

- Prefer **pure python**
  - easy prototyping
  - vectorized: very fast
  - succinct: readable, maintainable
    - NumPy array broadcasting
  - **dynamic typing**: check for type errors at run-time

- Why C/C++?
  - can't always vectorize (e.g. DP)
  - can't express and need **speed**
  - build new tools
  - extensions
- Run-time type errors more painful
  - **static typing**: check for type errors at compile-time

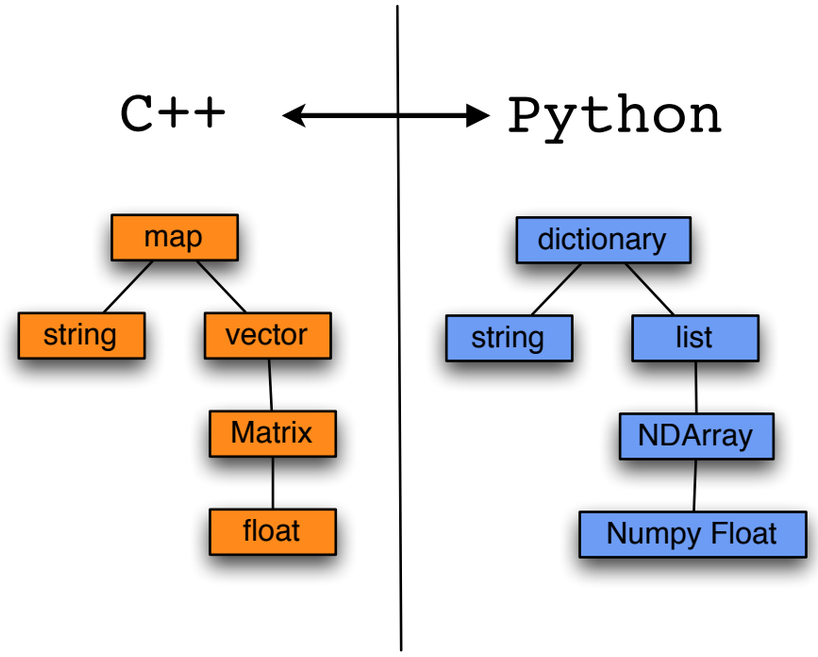
What this talk is ***not*** about?

- **Not talking about**
  - **calling** C++ functions **from Python**
  - **wrapping** C++ classes
  - **exposing** Python from C++
- **Not a comparison**
  - Boost, PyCXX, Py++, SWIG, Cython, Weave, Python C API, ctypes



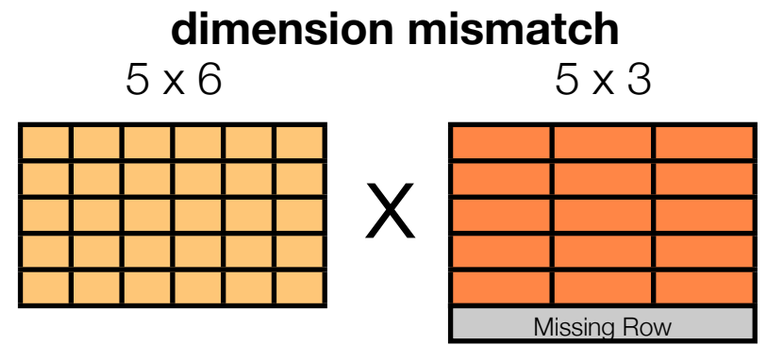
Interchanging Python and C++ containers in C++

- Assume we're already in C++-space
- Convert arbitrary Python containers to C++ containers and vice versa (e.g. big list of images).



Examples of Containers

- Standard Template Library containers
- Canonical C arrays
- LIBCVD: Cambridge Video Dynamics library (C++)
  - frame-rate real-time
  - hardware-exploitative
  - large numbers of images
- TooN Linear Algebra Library (C++):
  - large numbers of small matrices
  - compile-time error checking



example.cpp

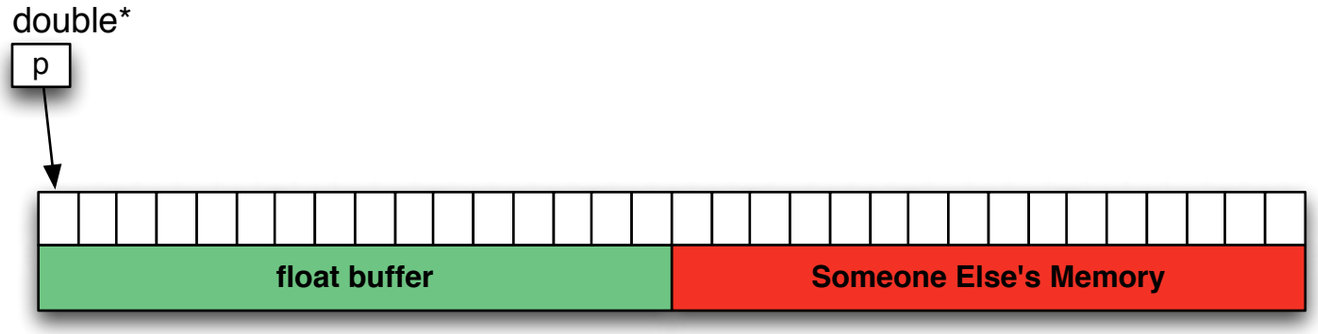
```
Matrix<5,6> A;
Matrix<5,3> B;
cout << B*A;
```

compiler error:

```
example.cpp:9
instantiated from
here: ‘size_mismatch‘
has incomplete type
```

Preventing Run Time Errors with Static Typing

- **Run-time Type Errors**
  - painful in C and C++ (**memory access**): buffer overruns, dangling pointers



  - hard to find: programs don't *immediately* ***crash***

- ***Static Typing***:
  - avoid *painful, hard to find* memory bugs
  - use type system for other things?


UNIVERSITY OF CAMBRIDGE
800 YEARS
1209–2009

C++ Template Facility: A Language In Itself

- **Features of Templates**:
  - Turing-complete: arbitrary computation
  - functional: immutable variables (store types and integers)
  - pattern matching language
- **That's great but is it useful?**
- Exploit structure of container types
  - figure out how to build conversion functions
  - check for logic errors

Container Interchange Between Python and C++

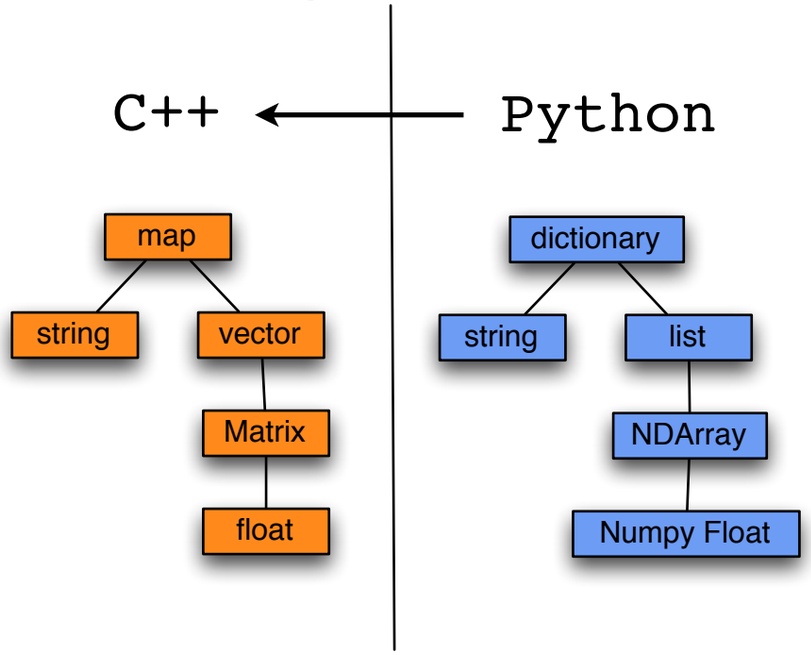
- Convert Python containers (`PyObject *x`) into C++ containers, examples:
  - `vector<int> y;`
  - `map<string, vector<Matrix<-1, -1, float> > > y;`
  - `double ***y;`
  - `vector<CVD::Image<byte> > y;`

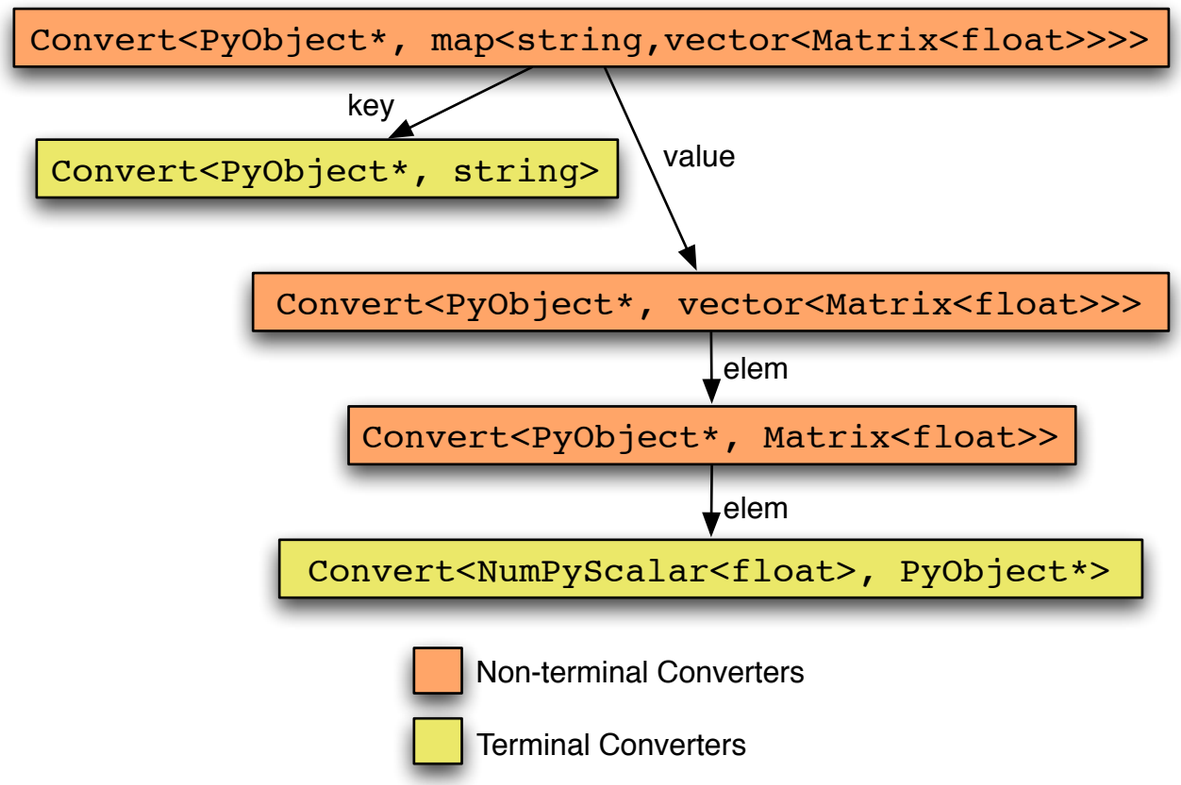
Los Alamos NATIONAL LABORATORY EST. 1943

UNIVERSITY OF CAMBRIDGE 800 YEARS 1209–2009

Compile-time Conversion Functions

- ```
  void example(PyObject *x) {
    map<string, vector<Matrix<-1, -1, float> > > y;
    convert(x, y);
  }
  ```
- Recursively built, one template instantiation builds another



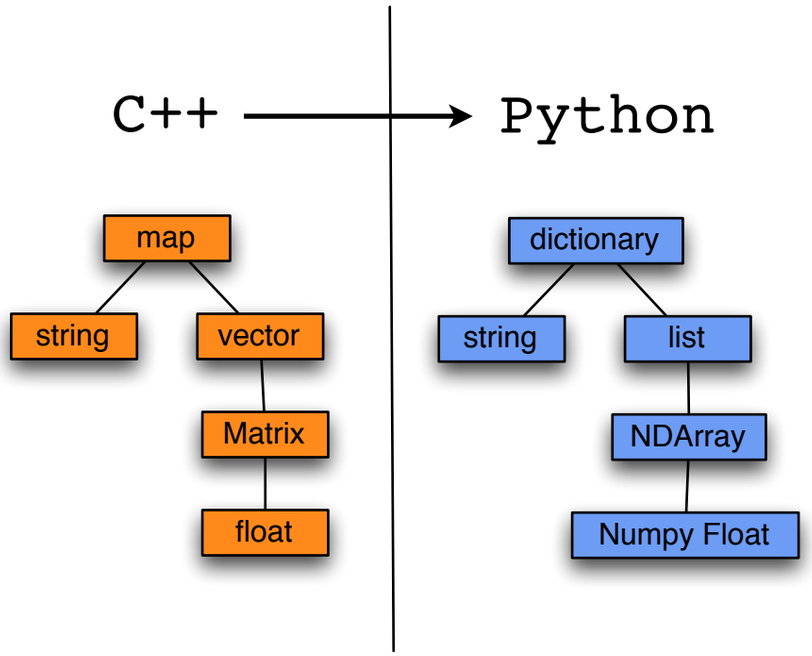


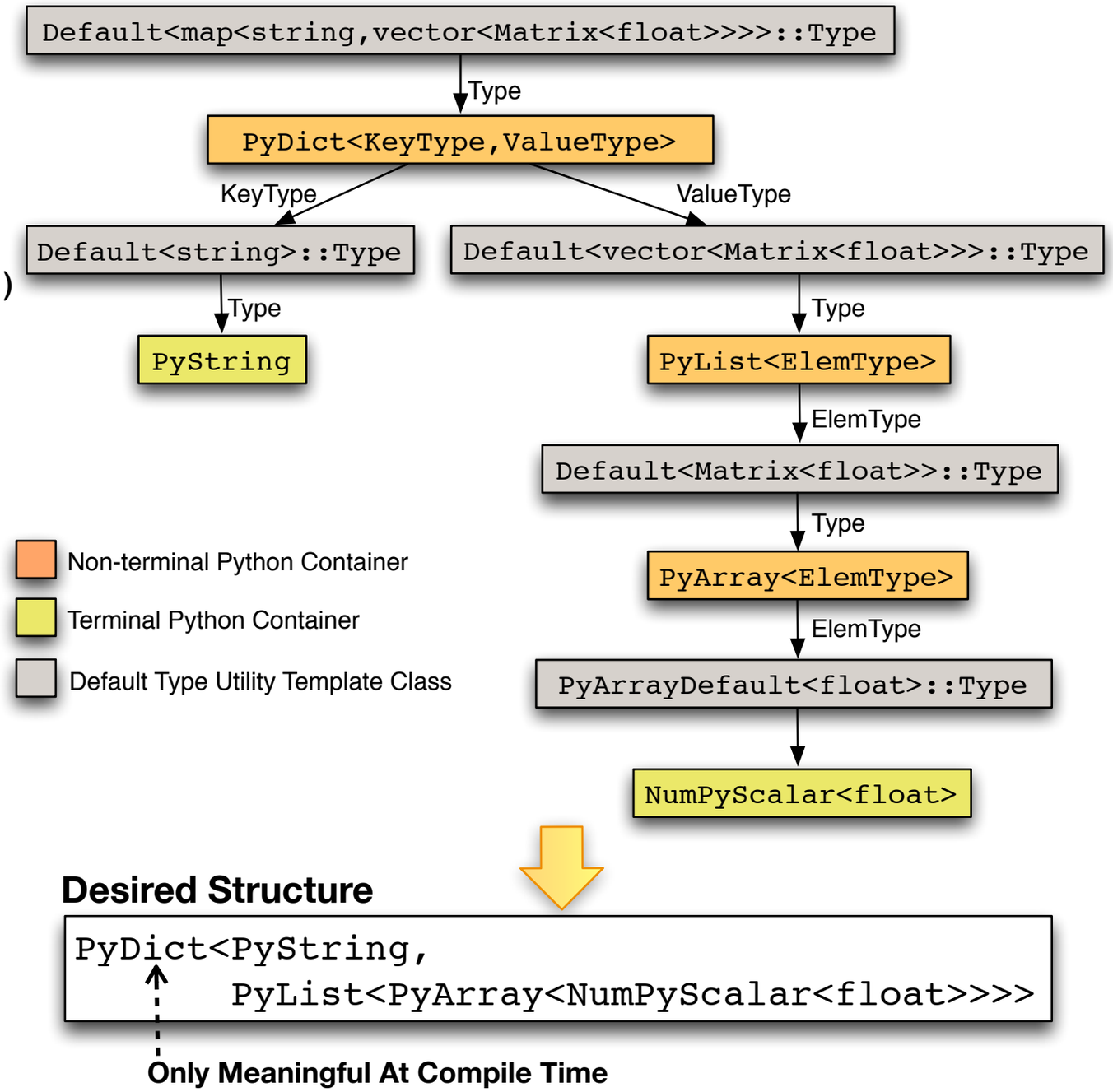
Note: for brevity, spaces before > are omitted.
  ```

Converting from C++ containers to Python

- Example (C++ to Python):

```
PyObject*
example(map<string,
        vector<Matrix<float> > > &x)
{
  PyObject *y;
  convert(x,y);
  return y;
}
```



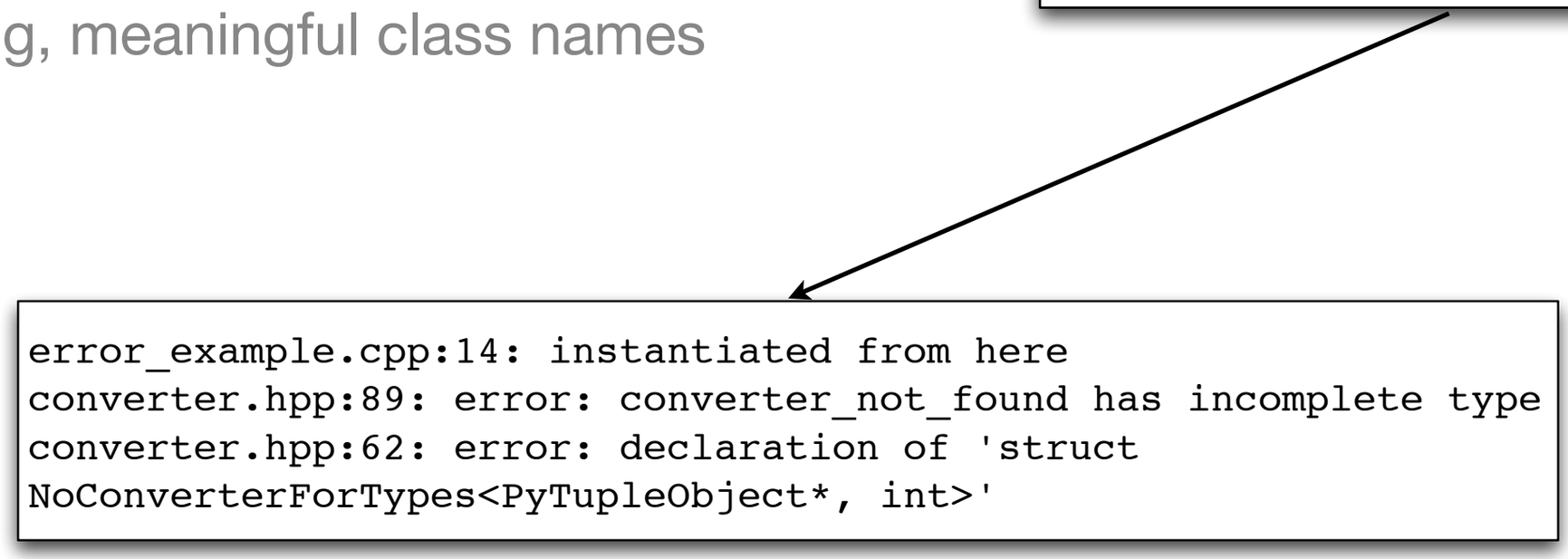


How bad are the template errors?

- Catch mistakes with compile-time-only classes
  - only serve a purpose at compile time
  - long, meaningful class names

Example of a bug:

```
PyTupleObject *x;
int y;
convert(x,y);
```



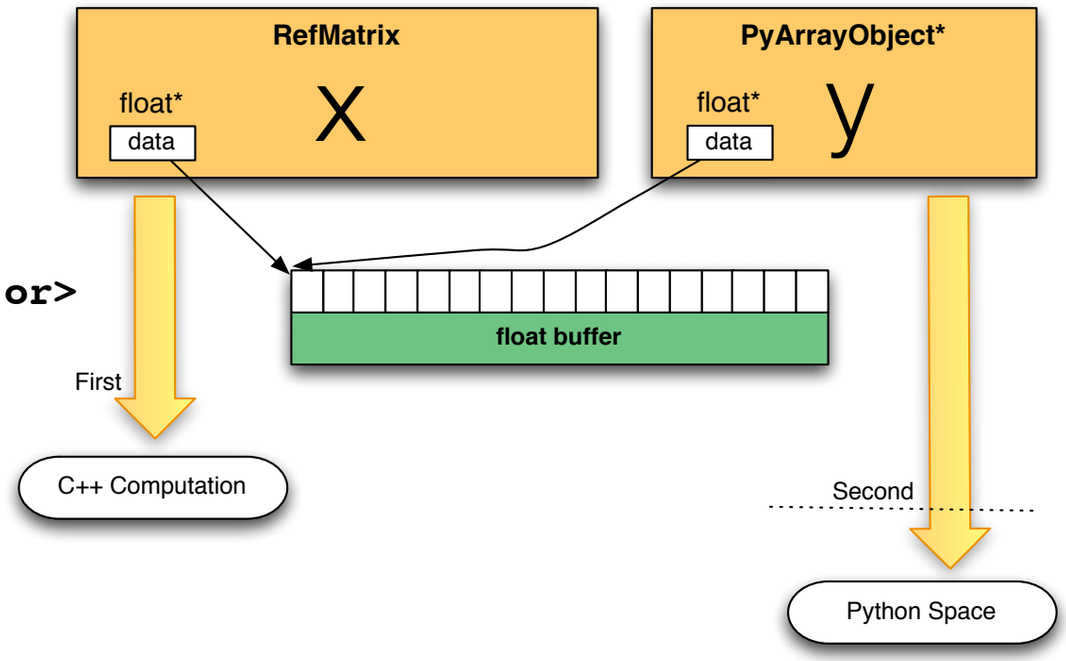
Los Alamos NATIONAL LABORATORY EST. 1943

UNIVERSITY OF CAMBRIDGE 800 YEARS 1209–2009

Allocating Result Arrays in C++ for Python

- Want to avoid copying
- Use TooN *Reference* arrays or CVD BasicImage class.
- NumPy array owns memory buffer.
- Example allocates a 10x10 array:

```
typedef
  Matrix<Dynamic,Dynamic,float,Reference::RowMajor>
  RefMatrix;
BiAllocate<RefMatrix, PyArrayObject*> a(10,10);
RefMatrix x(a->first());
PyArrayObject *y(a->second());
// Do C++ computation
...
// Return the result back to Python
return y;
```



Other neat tricks not covered

- Invoke (not wrap!) template algorithms using *type lists* and *selector functions*.
- Conversion of canonical multi-dimensional C arrays (e.g. `double***`)
- Interfacing Python containers and arrays with
  - **TooN**: many fast linear algebra algorithms for real-time applications.
  - **CVD**: frame-rate image processing and computer vision.
- If you have any questions about how all this works, ask me *in person*!

Source Code

- Convert-XY: http://code.google.com/p/convert-xy/
- TooN: http://mi.eng.cam.ac.uk/~er258/cvd/toon.html
- LIBCVD: http://mi.eng.cam.ac.uk/~er258/cvd/index.html

- Contact: eads@soe.ucsc.edu or er258@cam.ac.uk
- Questions?