Scripting and programming using

cctbx

(Computational Crystallography Toolbox)

Ralf Grosse-Kunstleve

Crystallographic Computing School, Oviedo, Spain, Aug 16-22, 2011

Luc's Observation

Collaboration is always a waste of time in the short term but we both learn how invaluable it is on the mid to long term.

Jun 3, 2008

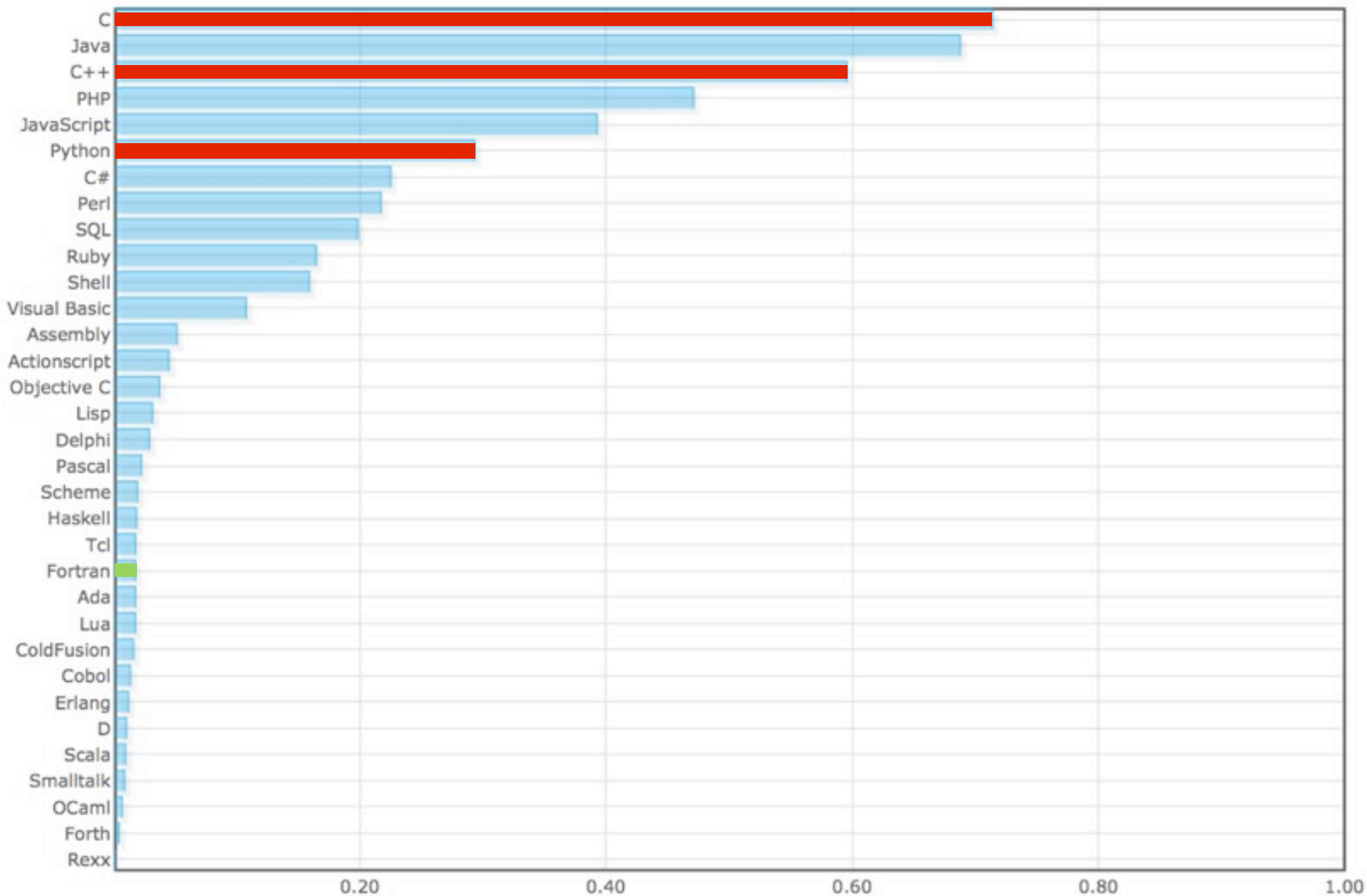
Aspects of a Library

- Functionality
- Environment
- Implementation languages
- Portability
- Developer community
- Reusability
- Stability
- Maintainability

Functionality - major cctbx modules

- Comprehensive symmetry algorithms (uctbx, sgtbx)
- Handling of reflection data (miller.array, iotbx.reflection_file_utils)
- Structure factor calculations (direct summation & FFT approximation)
- FFT library (fftpack)
- Map manipulation tools (maptbx)
- Direct methods (dmtbx)
- Charge flipping (smtbx)
- General purpose minimizers (lbfgs)
- Fully featured small-molecule refinement (smtbx)
- All major components for macromolecular refinement (mmtbx)
    - TLS constraints
    - Rigid-body refinement
    - Bulk-solvent correction
    - Twin refinement
    - NCS restraints (Cartesian space, torsion-angle space)
    - Secondary structure restraints
    - Simple molecular dynamics (Cartesian space, torsion-angle space)
    - Simulated annealing (Cartesian space, torsion-angle space)
    - Validation tools
- Data reduction tools: spot finding, indexing, integration (spotfinder, rstbx)
- Fast comprehensive PDB handling library (iotbx.pdb)
- Comprehensive CIF library (iotbx.cif)
- Comprehensive handling of SHELX ins/res/hkl files
- Family of array types and matrix algorithms (scitbx.array_family, scitbx.matrix)
- Parameter handling language (libtbx.phil)
- OpenGL support (crys3d, gltbx)
- OpenMP support (omptbx)
- Fortran to C++ converter (fable)
- Modular, non-intrusive build system (libtbx, SCons)

Environment



langpop.com

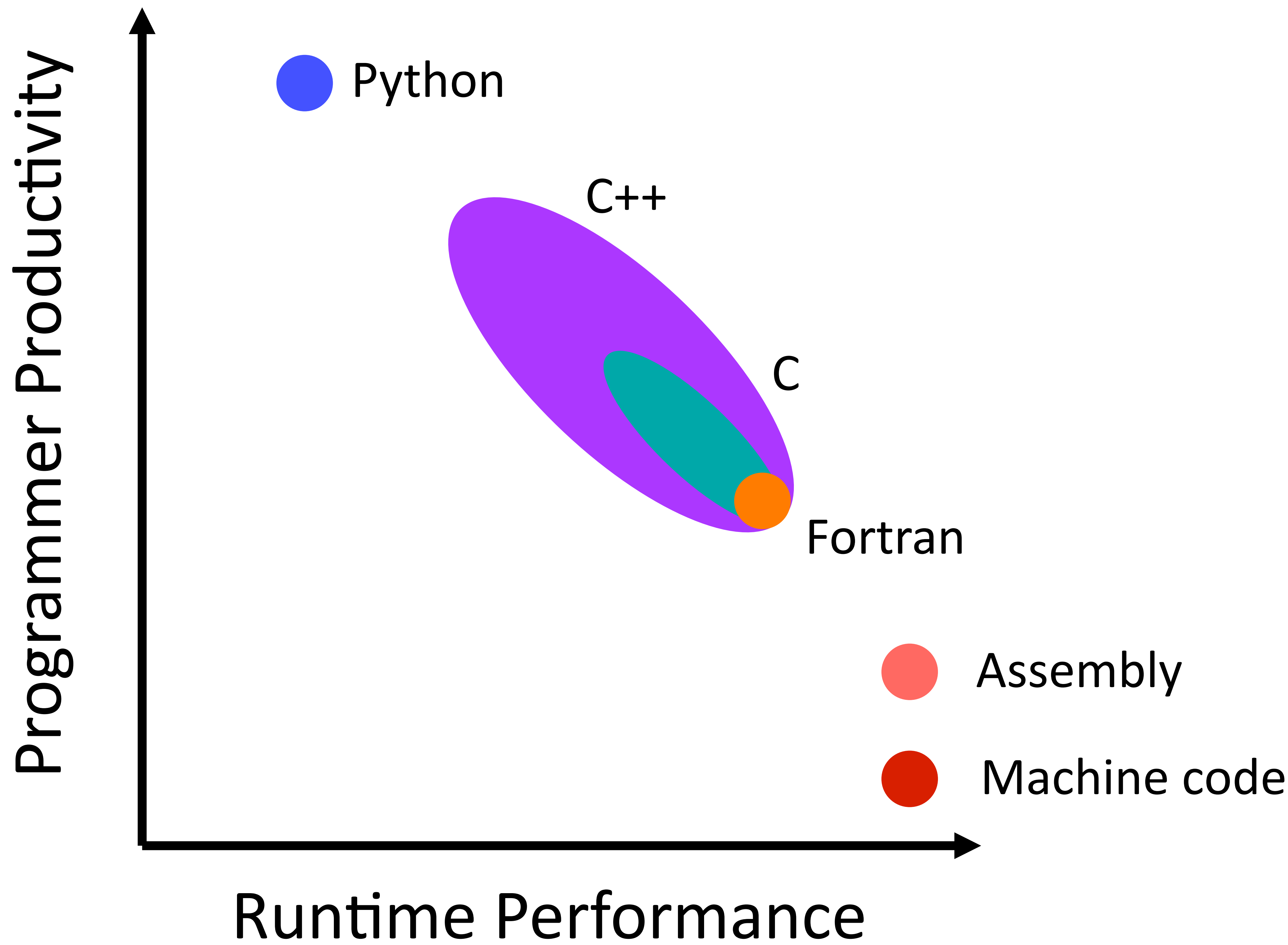
Environment

- Internet has fundamentally changed software development
- Confluence of technologies
- The World-Wide-Web in which we live
  - Revision control systems (e.g. Subversion)
  - Mailing lists for fast asynchronous exchange of ideas
  - Issue tracking systems (e.g. Bugzilla)
- Open-source tool chain
  - Linux ← GCC ← Boost, Python ← SCons ← cctbx

Implementation languages - Spectrum

- Python
    - Interpreted, Object Oriented, Exception handling
- C++
    - Compiled, Object Oriented, Exception handling
- C
    - Compiled, User defined data types, Dynamic memory management
- Fortran
    - Compiled, Some high-level data types (N-dim arrays, complex numbers)
- Assembler
    - Computer program is needed to translate to machine code
- Machine code
    - Directly executed by the CPU

Implementation languages - Efficiencies



Implementation languages - Matrix

	Dynamically typed ⇒ Programmer Productivity	Statically typed ⇒ Speed
Interpreted ⇒ Programmer Productivity	Python	Java
Compiled to machine code ⇒ Speed	PyPy	C++

Implementation languages - Pros & Cons

Python

- \+ Very high-level programming
- \+ Easy to use (dynamic typing)
- \+ Fast development cycle (no compilation required)
- − Too slow for certain tasks
- \+ Easy multiprocessing on multi-core machines (1800 × 64)
- \+ Abstraction of Operating System / Intersection with role of Operating System

C++

- \+ High-level or medium-level programming
- − Many arcane details (strong static typing, C legacy)
- \+ Largely automatic dynamic memory management (templates)
- \+ Much faster than Python

With enough attention, performance within 15% of FORTRAN

Portability

- "How easy is it to install cctbx on my machine?"
  - Reusing libraries
    - \+ Increased productivity ("don't re-invent the wheel")
    - − Dependencies
- End-users: distribute binaries
  - \+ Good approach in many situations
  - \+ Eliminates time-consuming compilation
  - − Requires access to many machines
  - − May lead to surprises ("strange crashes")
- Developers: need source by definition
  - Easy installation from sources is essential
  - Side-effect: easy installation from sources for end-user
  - Open-source is essential

Portability - cctbx approach

- Click to download **`cctbx_python_272_bundle.selfx`**
  `perl cctbx_python_272_bundle.selfx`

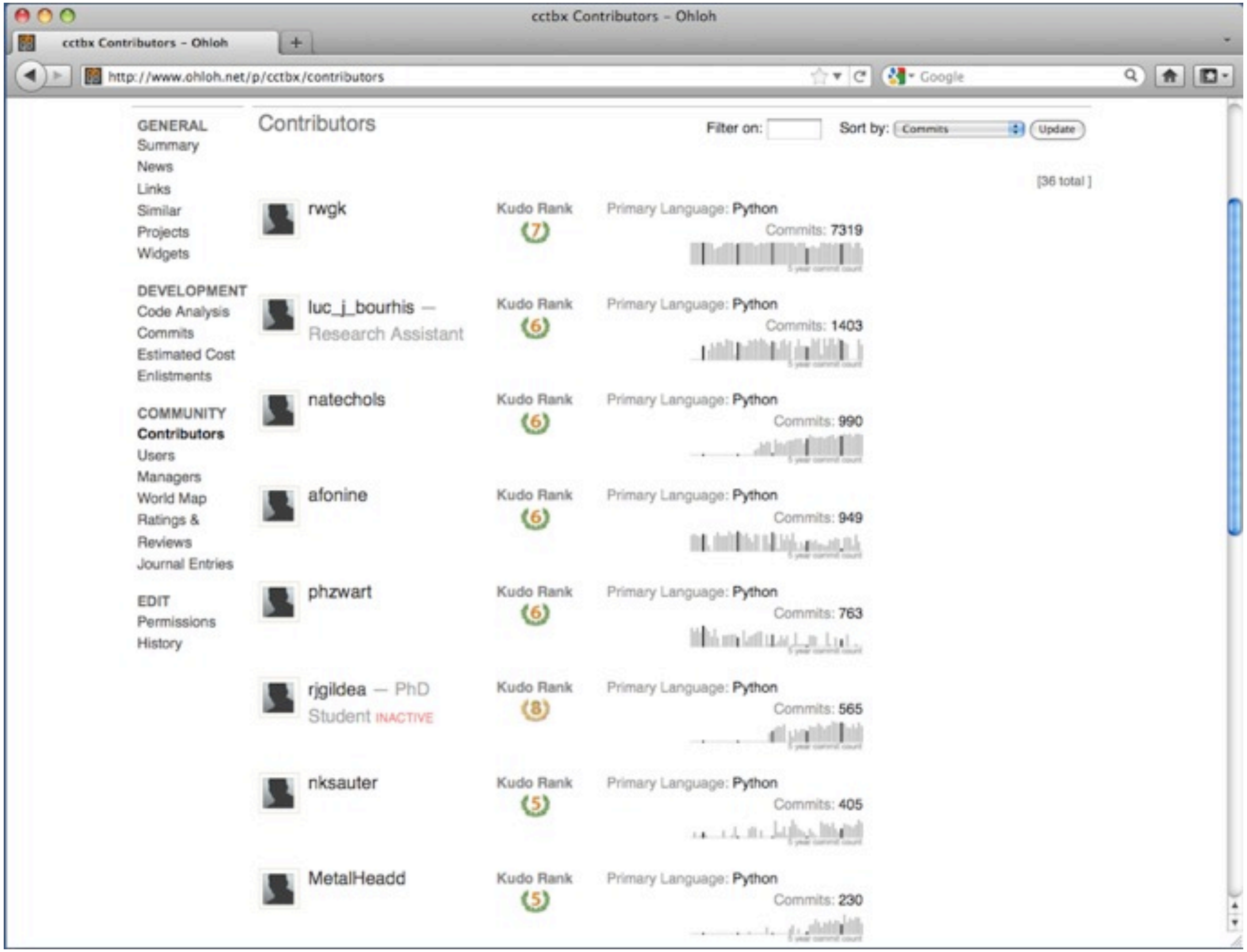
- Installs Python and cctbx including all dependencies from scratch
- There are also binary bundles (all major platforms including Windows)
- cctbx includes tools for building bundles
  - often easy to tie external sources into the cctbx build system

- Only dependencies: Operating System, C/C++ compiler

Developer community

- One-man project vs. group of developers
- Pre-internet era: mostly one-man projects or one-lab projects
- Post-internet era:
  - community geographically spread out
  - diverse communities, but with intersecting interests
  - communities are constantly improving infrastructure for working together most efficiently
  - self-organizing division of labor

cctbx contributors with >100 commits



registered developers: 42

Community resources

Reusability

- Object-oriented paradigm
  - Better name IMHO: context-oriented (namespaces)
  - Classes ≈ enhanced namespaces
  - Classes ≈ functions that preserve context (data & algorithms)
- Polymorphism
  - Runtime (dynamic typing, C++ virtual functions)
  - Compile-time (C++ templates)
- Exception handling
  - Bertrand Meyer (Eiffel creator) ca. mid 1990's:
    "It is impossible to write reusable code without exception handling."

Stability

- Automatic testing
  - Multiple developers: nobody knows all interactions
  - “No copy-and-paste” paradigm → generalization of existing code
  - Requires discipline: tests must be written together with the production code
- Interface changes
  - OK to change relatively new interfaces
  - Long-established interfaces should only be changed with great care (and ample warnings to anyone who could potentially be affected)

Typical development cycle

- Initial implementation in Python
  - Much faster than writing C++ (factor 3-5)
  - Tests are developed at the same time (ca. 1/3 of initial effort)
  - Often results in efficient code since optimized C++ libraries are reused
- Analysis of working code
  - Find performance bottlenecks (if any)
- Port rate-limiting parts to C++ (ca. 1/2 of total effort)
  - `cp algorithm.py algorith.hpp`
  - factor 10-30 speedup
- Bind C++ implementation to Python (ca. 1% of total effort)
- Adjust prototype to make use of C++ version
  - Remove original Python code
  - Or reuse in unit test, comparing the results of the two versions
- Integrate into application

Typical release cycle

- Run automatic multi-platform build & tests
- Manually check the results
- Tell co-workers about problems
- Wait for fixes
- Rerun until all problems are resolved
- Regenerate the online documentation
- Release (trivial operation)

Maintainability

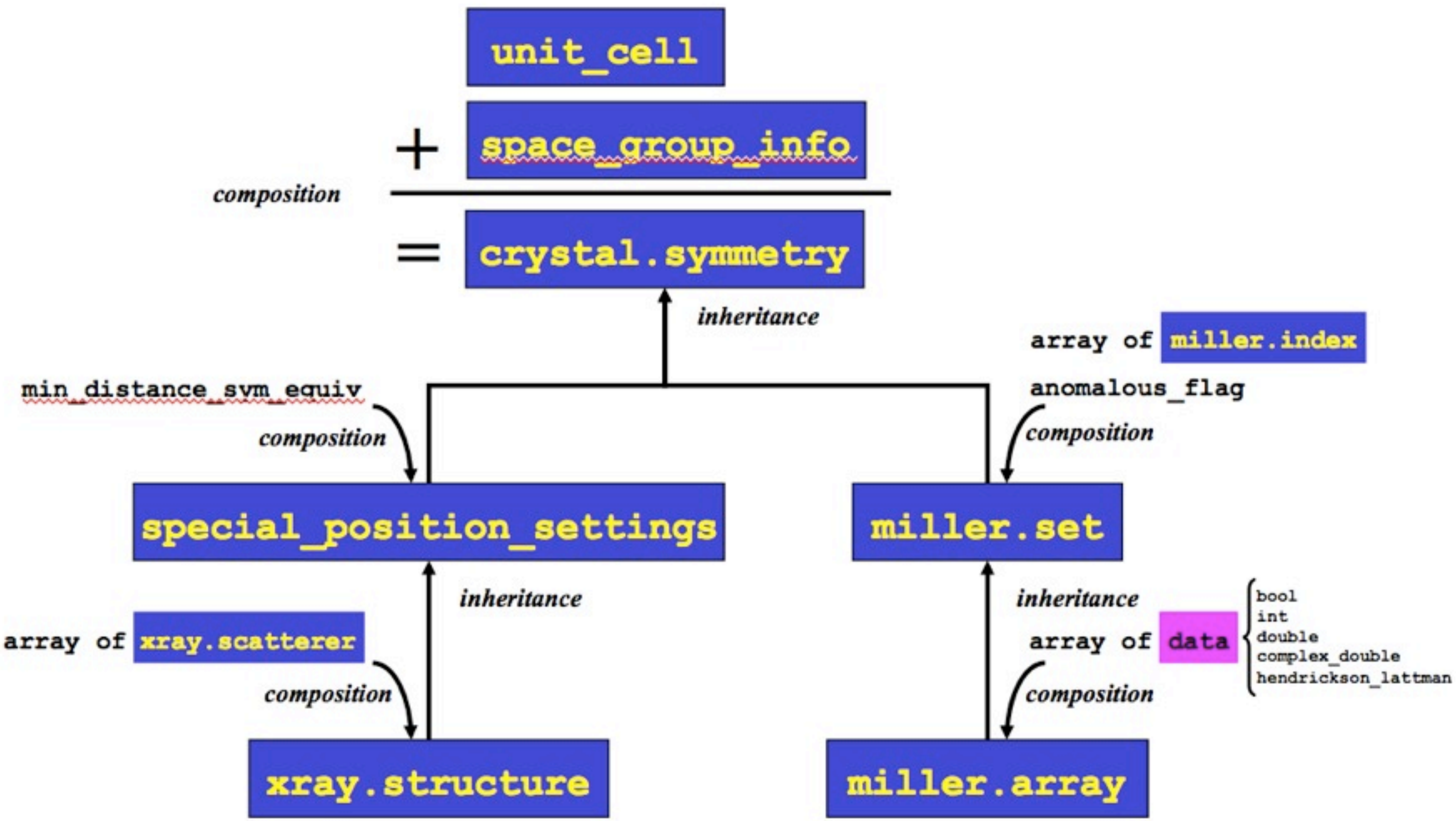
- "Redundancy is the worst enemy of long-term development."

- "Each time you copy-and-paste more than three lines without modifying at least two you are making a mistake."

- Redundancy leads to code inflation
  - Severe problem for large projects
- cctbx sizes after about ten years of development:
  - ca. 600k lines (20+ MB) source code
  - (ca. ⅓ unit test code)

Tutorials

Central cctbx types



Acknowledgments

- Luc Bourhis
- Phenix developers
  - Paul Adams, Pavel Afonine, Nathaniel Echols, Richard Gildea, Jeffrey Headd, Tom Ioerger, Airlie McCoy, Nigel Moriarty, Nicholas Sauter, Tom Terwilliger, Peter Zwart
- CCP4 (Martyn Win, Kevin Cowtan)
- David Abrahams (Boost.Python)
- NIH
- DOE
- Phenix industrial consortium members