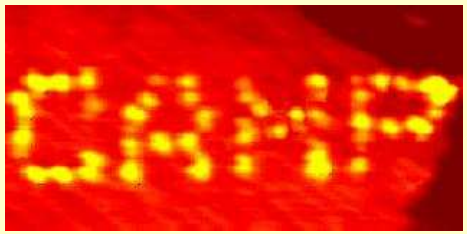
Development of Atomic Simulation Tools using Python and C++



CENTER FOR ATIMIC-SCALE MATERIALS PHYSICS
DEPARTMENT OF PHYSICS
TECHNICAL UNIVERSITY OF DENMARK

Jens Jørgen Mortensen,
Jakob Schiøtz,
Karsten W. Jacobsen

CORNELL THEORY CENTER
AND
LOBORATORY OF ATOMIC AND
SOLID STATE PHYSICS

C.-S. Chen, T. Cretegny, A. J. Dolgert,
C. R. Myers, J. P. Sethna

Outline of talk

- One language is not enough!
- Why use Python?
- Examples:
  - A Molecular Dynamics code
    - Parallelization
    - QuasiContinuum
  - A Density Functional Theory code
- Conclusions

From the python FAQ: (www.python.org/doc/FAQ.html)

1.1. What is Python?

Python is an interpreted, interactive, object-oriented programming language. It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes. Python combines remarkable power with very clear syntax. It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++. It is also usable as an extension language for applications that need a programmable interface. Finally, Python is portable: it runs on many brands of UNIX, on the Mac, and on PCs under MS-DOS, Windows, Windows NT, and OS/2.

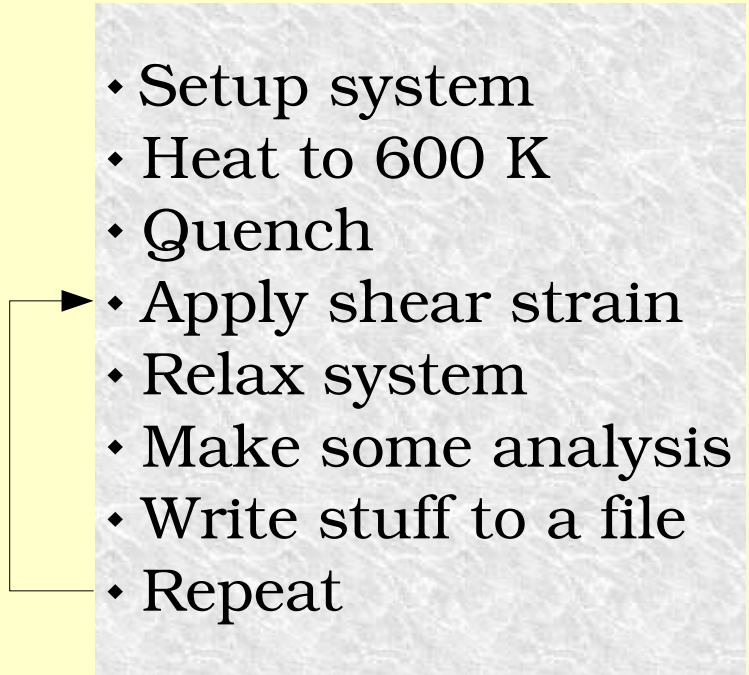
Two languages are needed

Use compiled languages for:

- time/memory critical parts
- simple, well defined tasks

Use interpreted languages for:

- Glue
- Formatted output
- File IO
- Analysis
- Visualization
- Interactive sessions
- Scripts



SWIG

(www.swig.org)

How fast is Python?

The number *e*:

$$e = \sum_{n=0}^{\infty} \frac{1}{n!}$$

```
e = 0.0
f = 1.0
for n in range(100):
    e += f
    f /= n + 1
print "e =", e
```

300 µs

```
#include <iostream>
int main()
{
    double e = 0.0;
    double f = 1.0;
    for (int n = 0; n < 100; n++)
      {
        e += f;
        f /= n + 1;
      }
    cout << "e = " << e << endl;
}
```

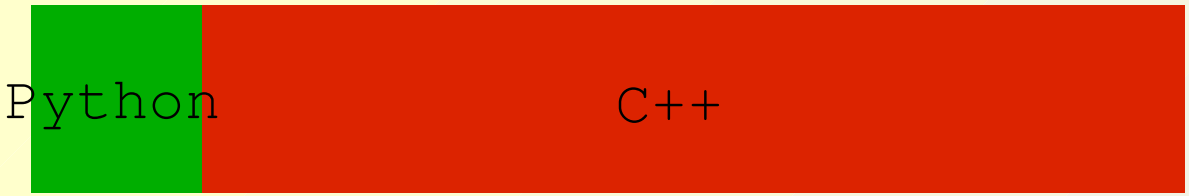
6 µs

```
print "e =", 1 + sum(divide.accumulate(arange(1, 100, typecode=Float)))
```

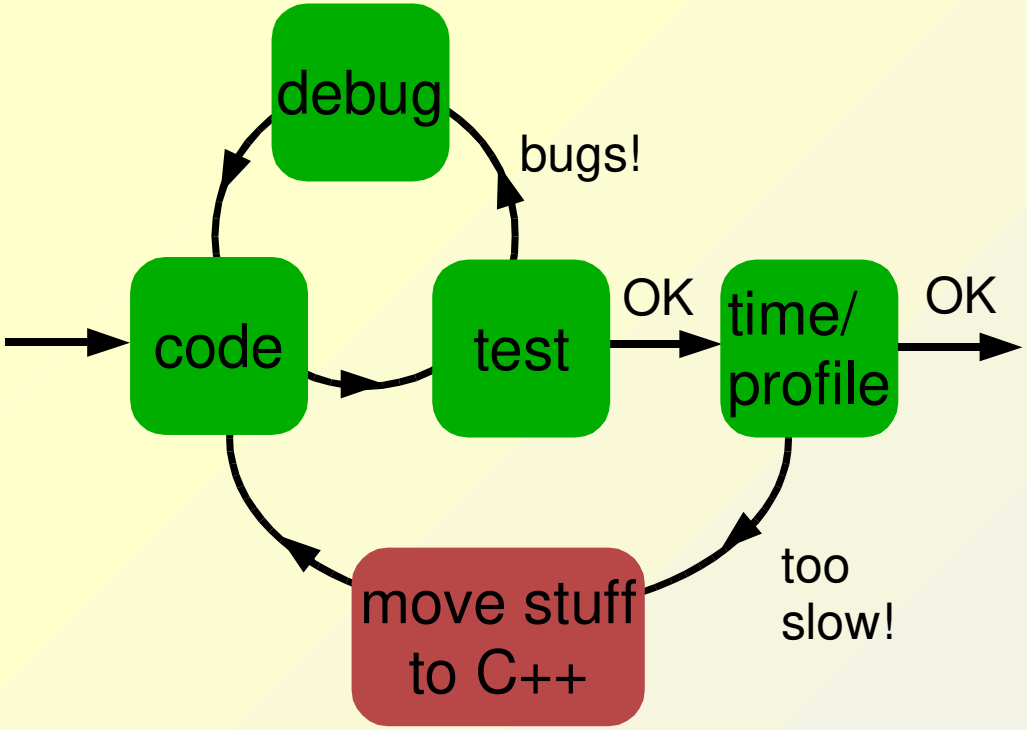
The ultimate goal:

Lines of code:

Execution time:



Python is Fast!



- No compilation and linking
- No declarations and .h files
- Easy interactive debugging
- No memory management
- Many bugs don't exist in Python
- Object Oriented
- "Self documenting" - hmm...

Molecular Dynamics

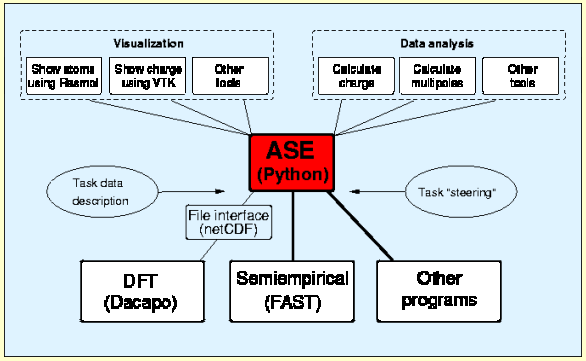
C++

- Energy/force calculations
- Neighbor list generation
- Neighbor list lookup
- Parallellization

Python

- Dynamics (Verlet, ...)
- Constraints
- Setup
- Analysis
- Visualization
- Scripting

Use Numeric Python for arrays of positions, velocities, ...



ASE: *Discussed in another talk ...*

Python

C++

Atoms

GetPositions
GetForces
GetPlot

velocities, ...

EMTPotential

CalculateForces

Atoms

GetPositions
GetForces

positions, ...

Potential

CalculateForces

SuperCell

basisVectors, ...

EMTPotential

CalculateForces

Xpotential

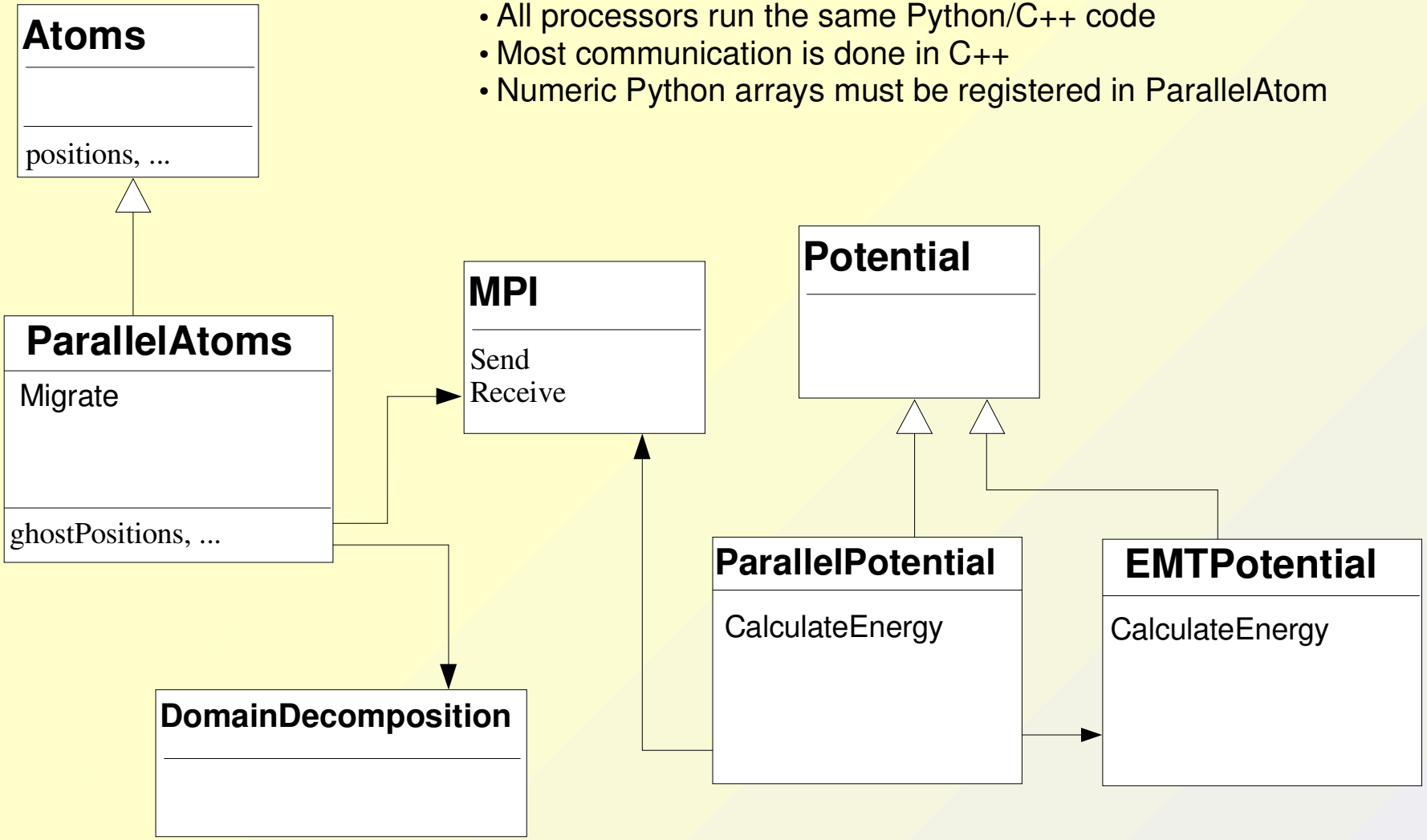
CalculateForces

NeighborList

MakeList

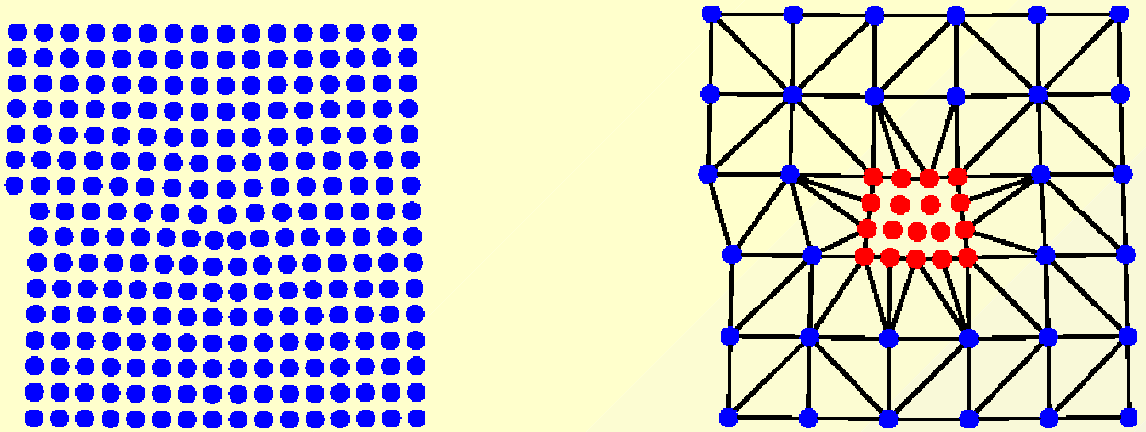
Parallellization

- All processors run the same Python/C++ code
- Most communication is done in C++
- Numeric Python arrays must be registered in ParallelAtom



QuasiContinuum

Edge dislocation



Atomistic

Atomistic + Finite Elements

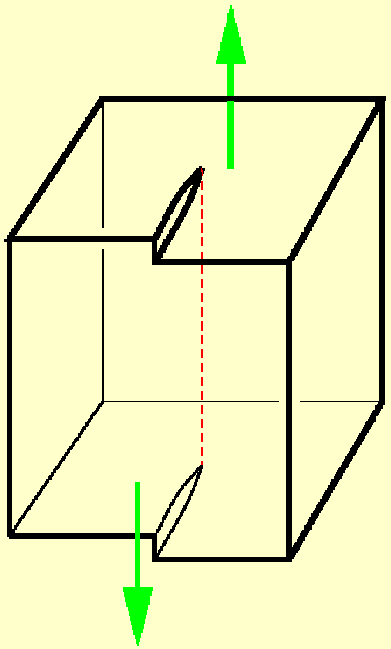
Generalize the concept of ghost atoms:

Ghost atoms can be:

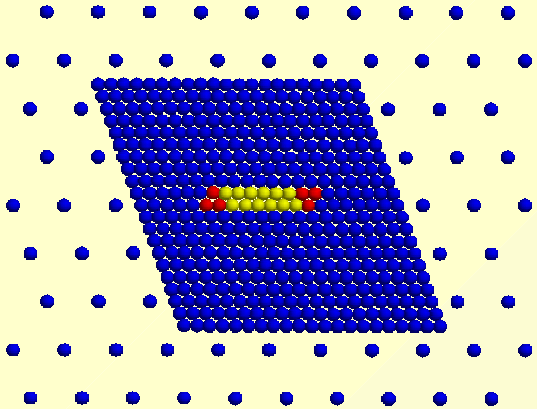
- atoms in a neighboring domain (parallel)
- atoms in a neighboring element (QC)

A QC example:

Cross-slip in Cu



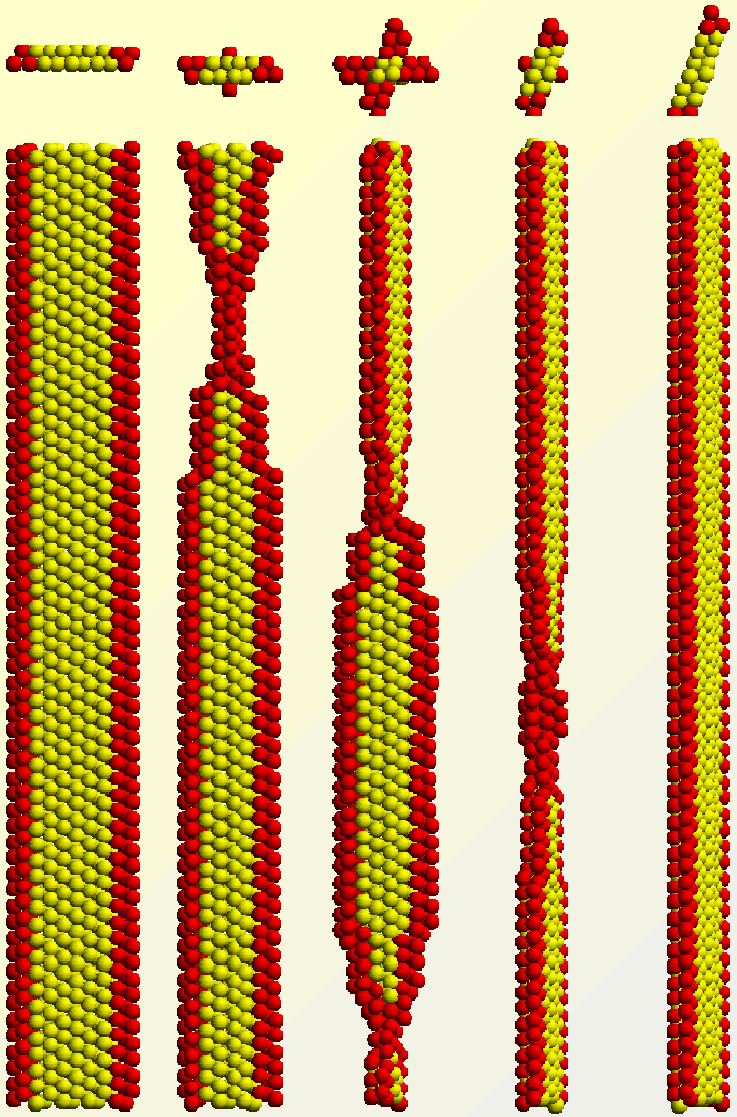
top view



Colors:
- FCC: blue/invisible
- HCP: yellow
- Defects: red

top view

side view



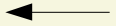
Primary plane

Transition state

Cross-slip plane

Outline of talk

- One language is not enough!
- Why use Python?
- Examples:
  - A Molecular Dynamics code
    - Parallelization
    - QuasiContinuum
  - A Density Functional Theory code
- Conclusions



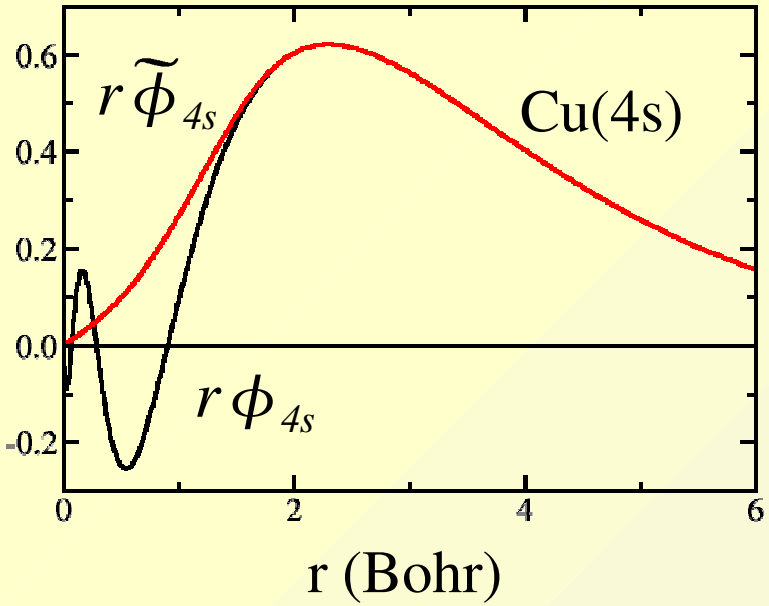
We are here!

Projector Augmented-Wave method

P. E. Blöchl, Phys. Rev. B **50**, 17953 (1994)

- Exact all-electron formalism
- Soft wavefunctions (like USPP)

Approximations:
- Finite number of projectors
- Truncated angular momentum expansions
- Frozen core



Projectors

$$|\Psi_n\rangle = \left(1 + \sum_i \left(|\phi_i\rangle - |\tilde{\phi}_i\rangle\right)\langle \tilde{p}_i|\right)|\tilde{\Psi}_n\rangle$$

All-electron

Soft

Real space grids

Advantages:

- Simple: Only one parameter (grid-spacing)
- Flexible boundary conditions:
  - Cluster, wire, surface, bulk
- First step towards O(*N*)
- No Fast Fourier Transforms

Parallelization:

- Domains in real space
- Low communication cost
- Good scaling of memory use

Techniques:

- Iterative multigrid solvers for Poisson and Kohn-Sham equations
- Short-ranged Mehrstellen finite differences operators

  (Phys. Rev. B, **54**, 14362 (1996))
- Double-grid technique

  (Phys. Rev. Lett., **82**, 5016 (1999))

Splitting the problem into small pieces:

C++

Operators: $\nabla^2 \tilde{\Psi}_n$ (Laplace)

Interpolation

Restriction

Projectors: $\langle \tilde{\Psi}_n | \tilde{p}_i \rangle$

Python

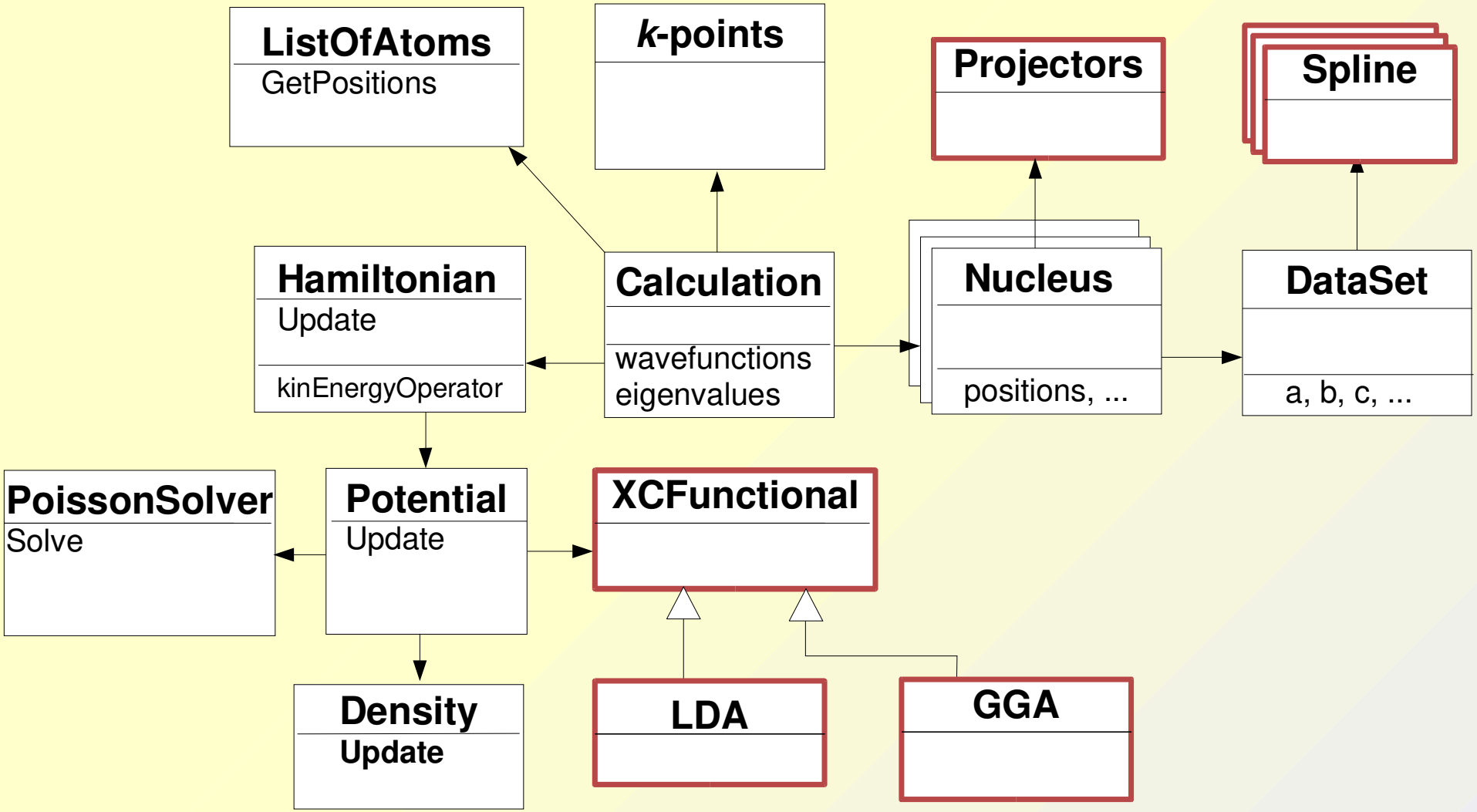
Matrix-vector products:

Numeric Python module (BLAS)

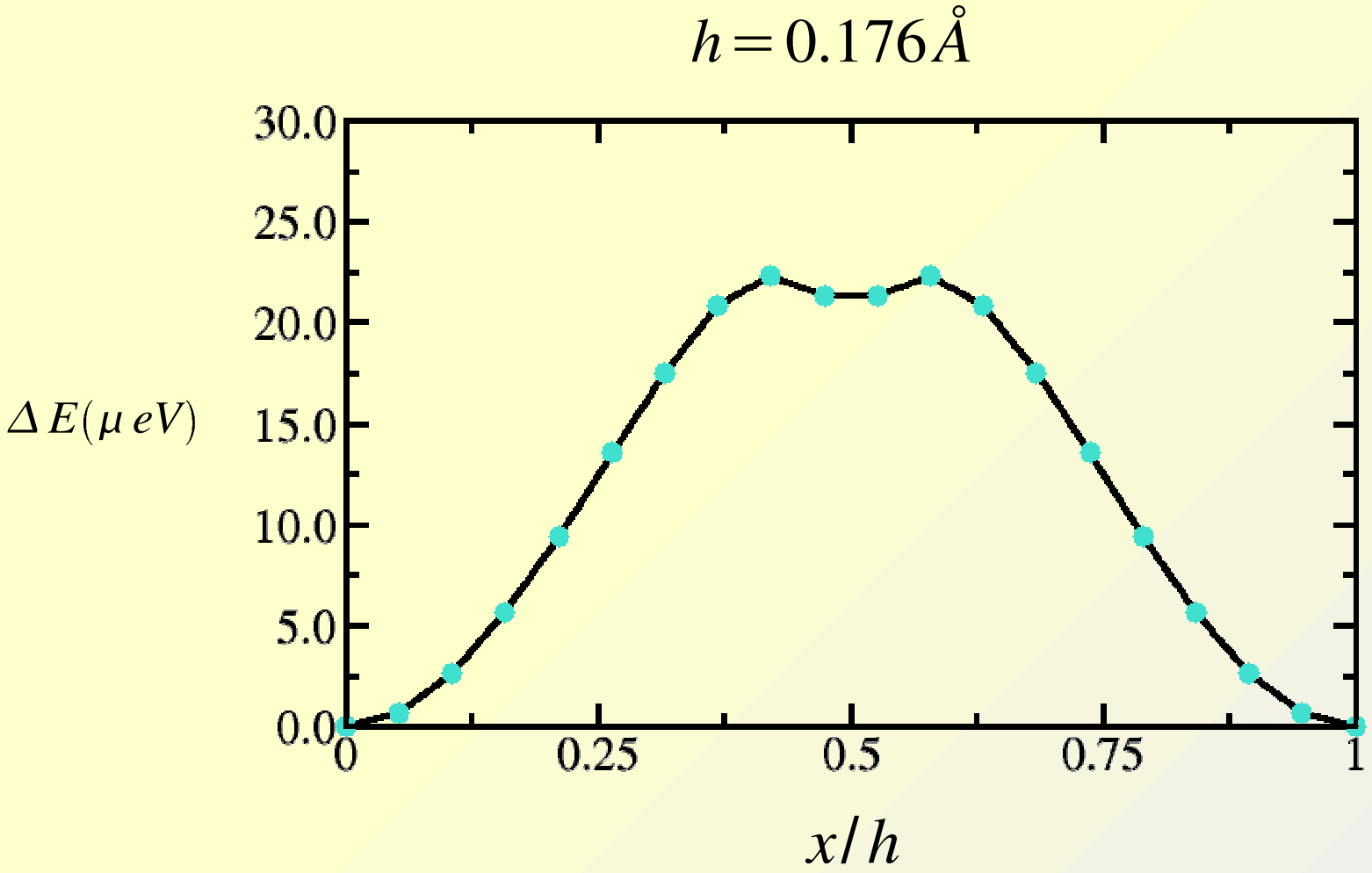
GLUE!

- Starting a calculation
- Setting parameters
- Controlling the C++ parts

Class/**object** diagram



Results



Conclusions

- Using the Python/C++ combination will
  - make development more enjoyable
  - speed up development
  - produce more readable and maintainable code
- The price to pay is worth it!
  - MD: 0%
  - PAW:?%