Developing the Next Generation of Molecular Modelling Software

Christopher Woods

Why code a new program?

- Academic software is difficult to extend
  - Spaghetti code, need for whole-code knowledge
- Sharing modifications is hard
  - License restrictions, incompatible branches
- Sharing methodology between academia and industry is slow
  - License restrictions, academic code is hard to use!
- Constantly re-inventing the wheel!
  - How many people have written a PDB reader?

Software Requirements

- Be easily developed by academics
  - Can have new methodology implemented, in a compatible manner
  - Modular and self-contained design
  - Modifications should not require whole-code knowledge
  - Code must encourage good programming practices

Three Core Requirements

- Speed – must be very fast
- Easy to use, easy to develop, easy to maintain
- Incredibly flexible – few software assumptions

Choosing the Right Language

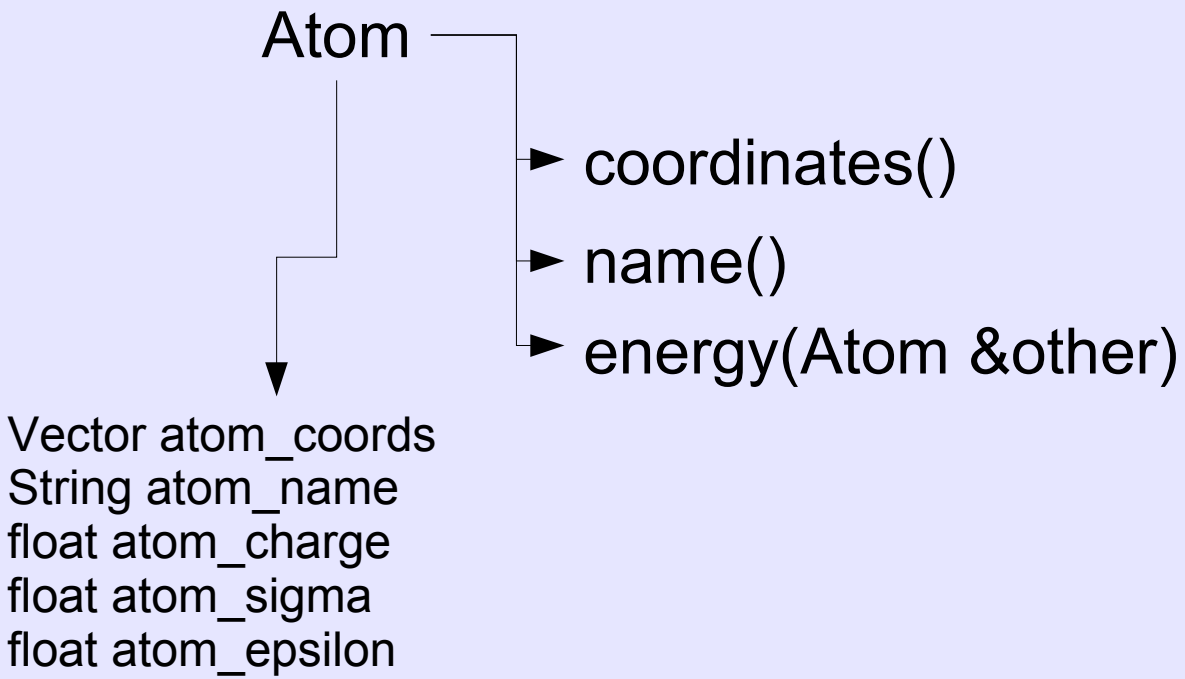
- Choose a programming language;
  - Fortran. Very fast, easy to understand
- ProtoMS is a modelling program I wrote at Southampton, in Fortran
- Very fast, now used by several researchers and groups
- Fortran code is not extended easily
- Linear control flow and static data structures make for a rigid design

Object Orientated Programming

- Flexible design mandates an object orientated programming language
- OO programming uses “objects”
  - e.g. “Atom” object would represent an atom
  - “AtomSet” object represents a set of atoms
  - “Molecule” object represents a molecule
- Speed requirement mandates use of a compiled OO language – C++

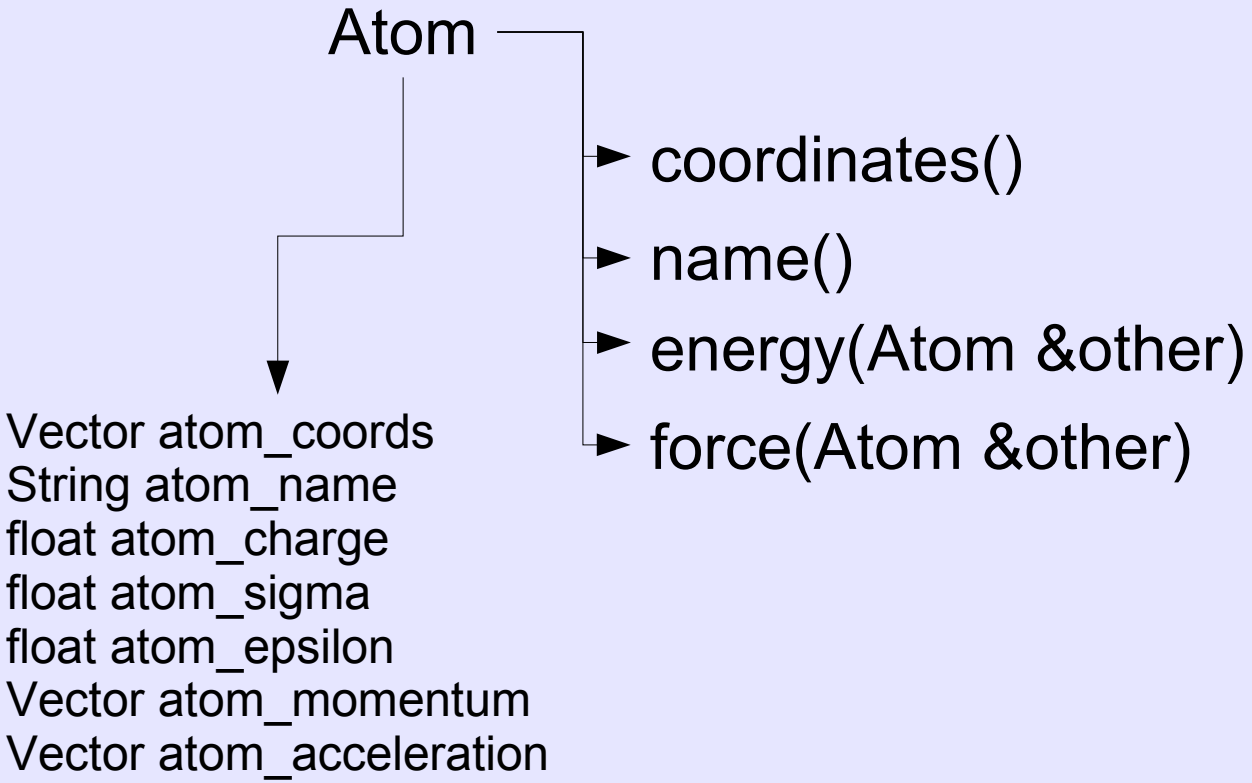
Designing the Core Objects

- Success of an OO code depends on the design of the core objects (the “data structure”)
- Lets design an Atom...



Inflexible design

- This design has already reduced the program's flexibility
- What about Molecular Dynamics? Atom would need momentum and acceleration

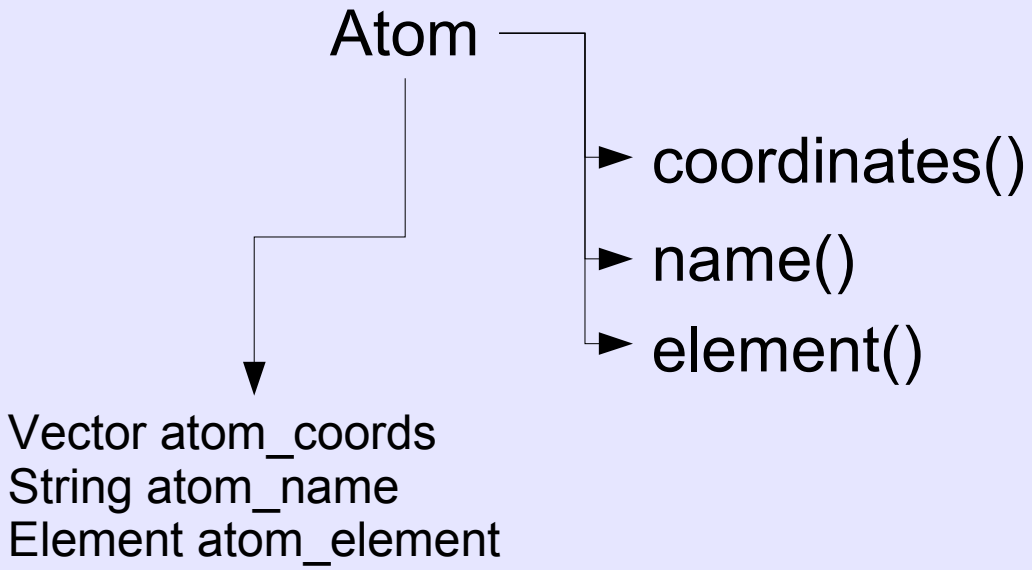


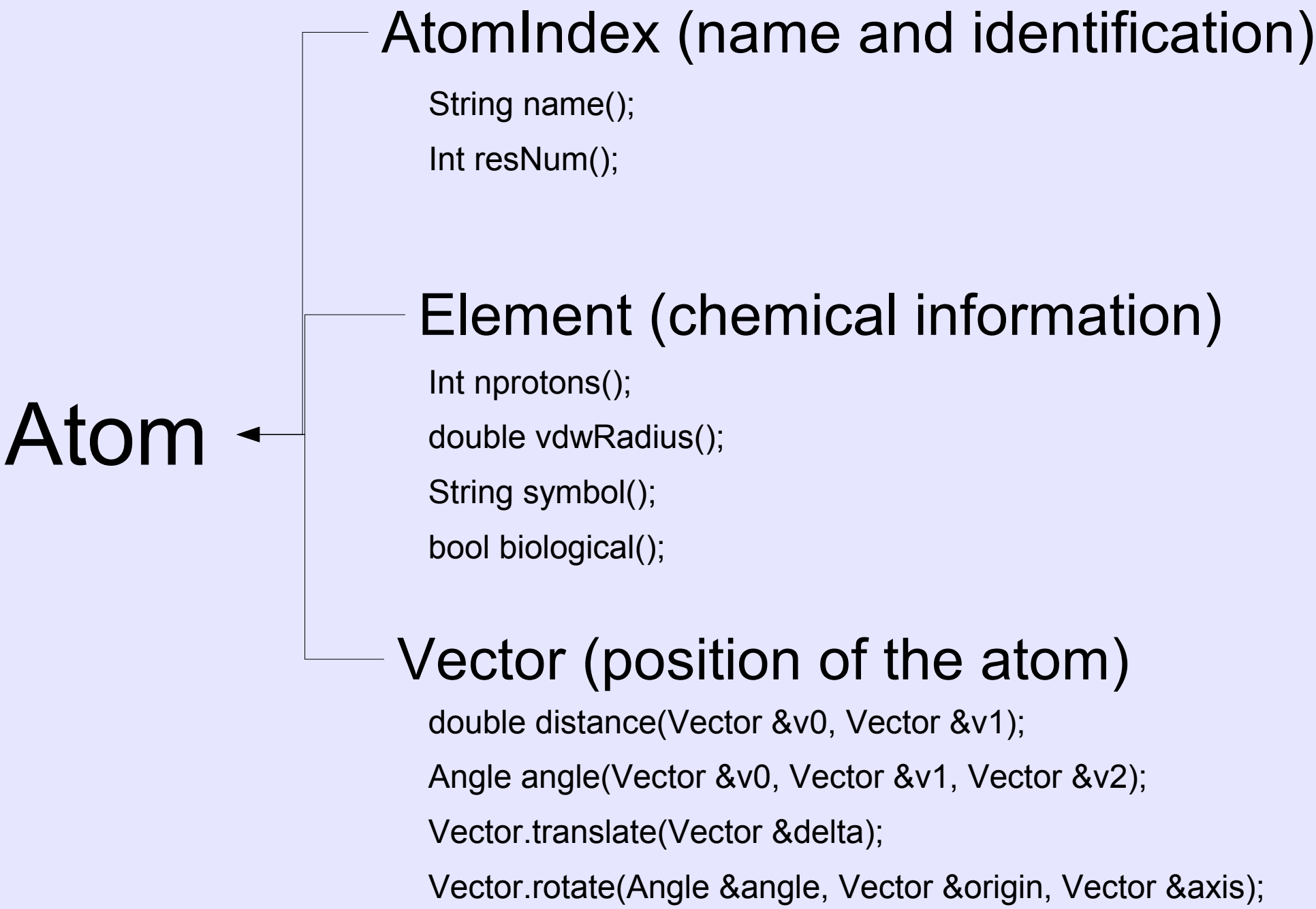
Atom-Centric Design

- What about QM? Or multipole forcefield?
- We could extend Atom to deal with all of these cases
- Extended design leads to bloat and irrelevant code – why should an atom have a velocity during an MC simulation?
- Most software uses such an atom-centred design
- Functionality is added to the Atom to cope with the demands of the software
- This was the approach used by Woodman

What does an atom really need?

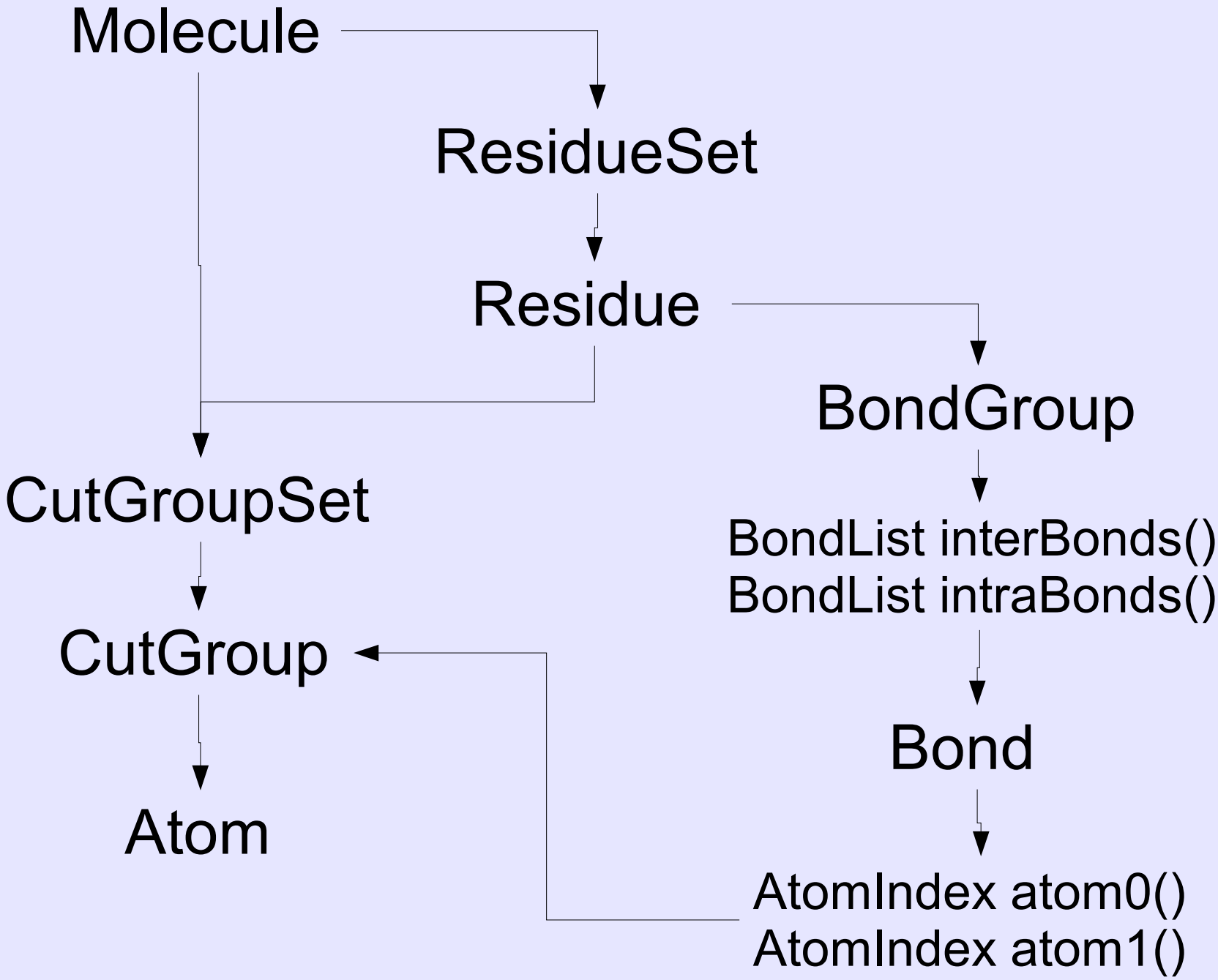
- Lets take a different approach. Just define a class based on the minimum information necessary




Atom
AtomIndex (name and identification)
String name();
Int resNum();
Element (chemical information)
Int nprotons();
double vdwRadius();
String symbol();
bool biological();
Vector (position of the atom)
double distance(Vector &v0, Vector &v1);
Angle angle(Vector &v0, Vector &v1, Vector &v2);
Vector.translate(Vector &delta);
Vector.rotate(Angle &angle, Vector &origin, Vector &axis);

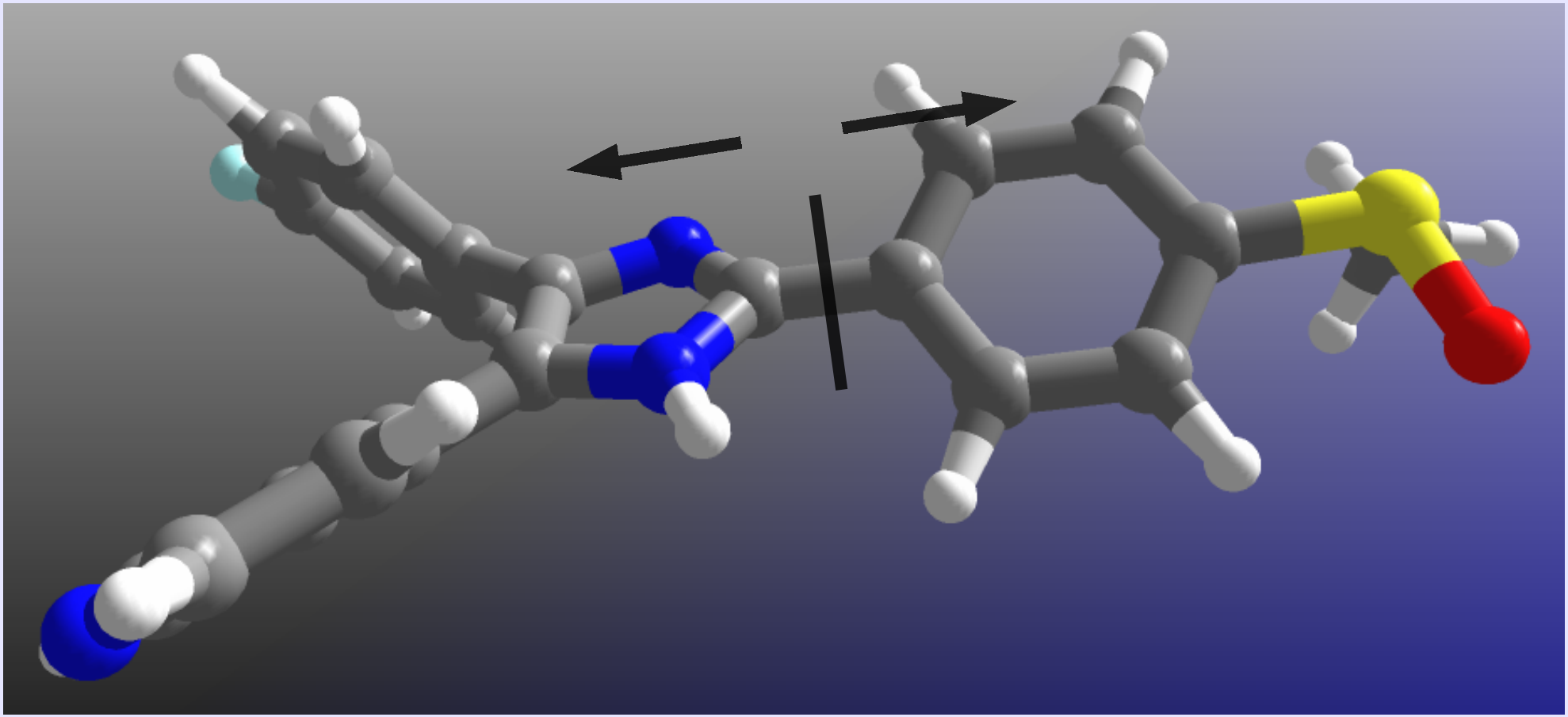
Grouping Atoms Together

- How should Atoms be grouped together? In residues or molecules?
- Neither! Most important sub-unit is the one that is used for non-bonded cutting, and periodic boundaries.
- Group of atoms that is treated as a single-unit for cutting or wrapping
- Call this unit a CutGroup. Organise Atoms into CutGroups

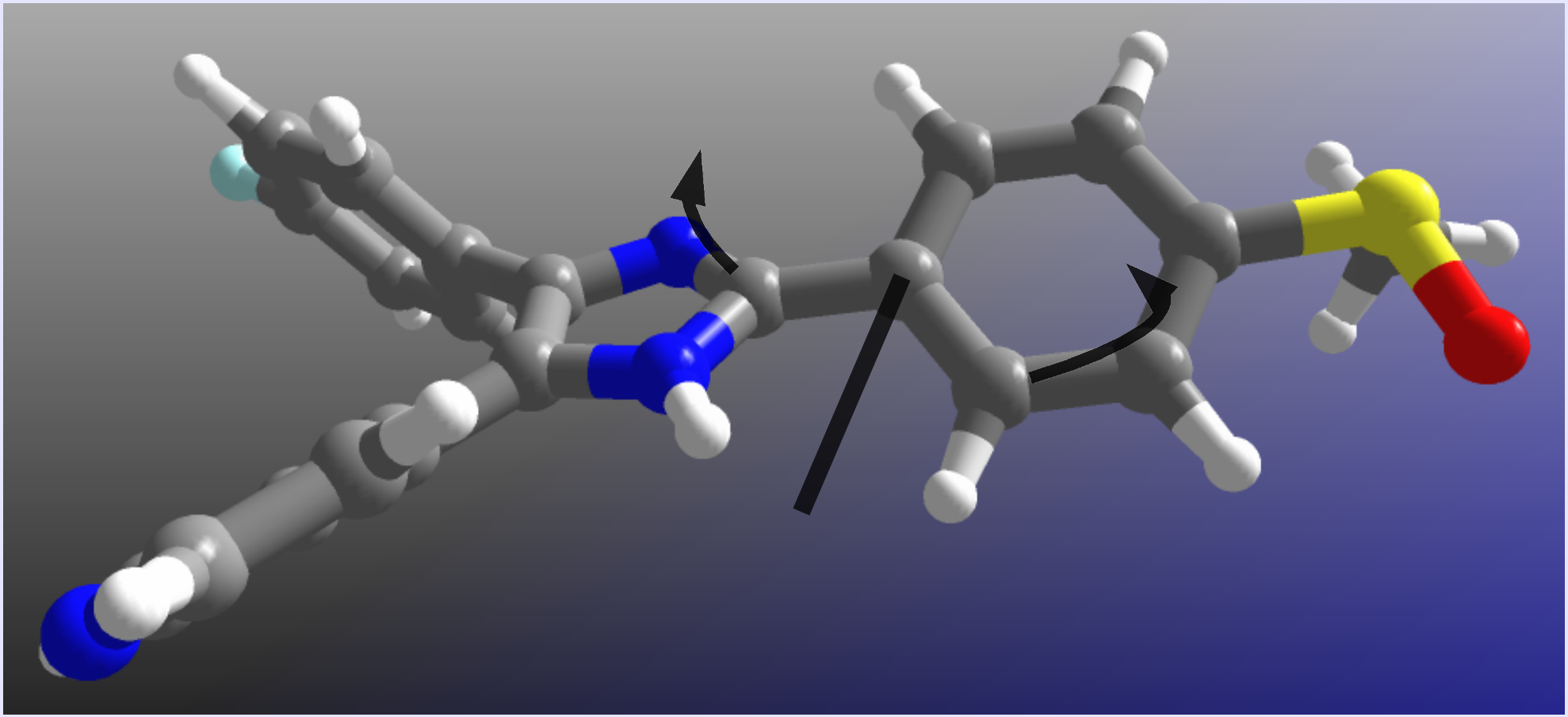

Molecule
ResidueSet
Residue
BondGroup
CutGroupSet
BondList interBonds()
BondList intraBonds()
CutGroup
Bond
Atom
AtomIndex atom0()
AtomIndex atom1()

Minimal Design

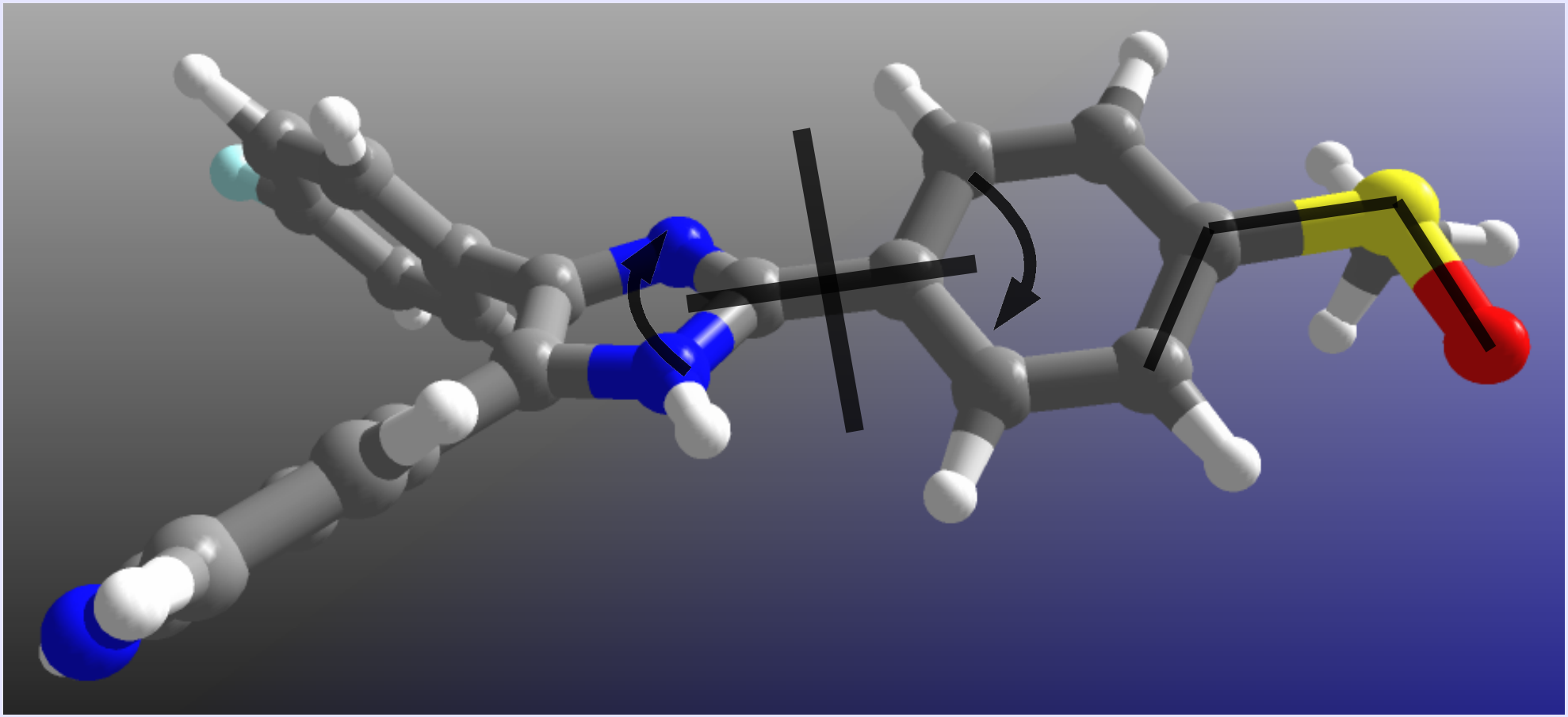
- We now have a minimalist design – Atom's are basically geometric, arranged into CutGroups, then into Molecules, which contain bonding information
- What can we do now with this design?
- Translate and rotate the molecule
- Perform internal geometry moves



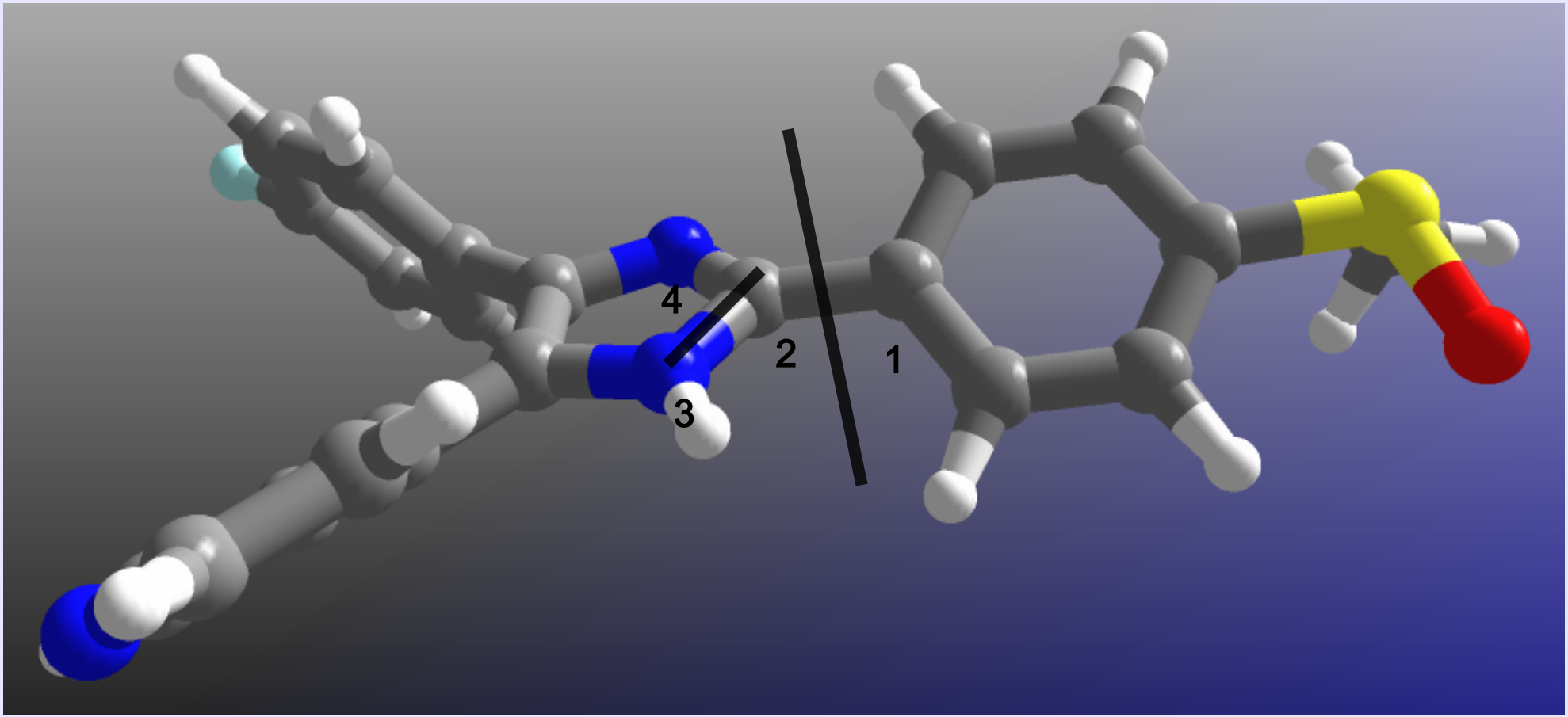
- Use connectivity to split molecule into two
- Translate both halves along vector of the bond
- Can weight the translation of each side
- Can anchor one side so it doesn't move



- Use connectivity to split molecule into two
- Rotate both sides about perpendicular to angle
- Can weight the rotation, or anchor one side

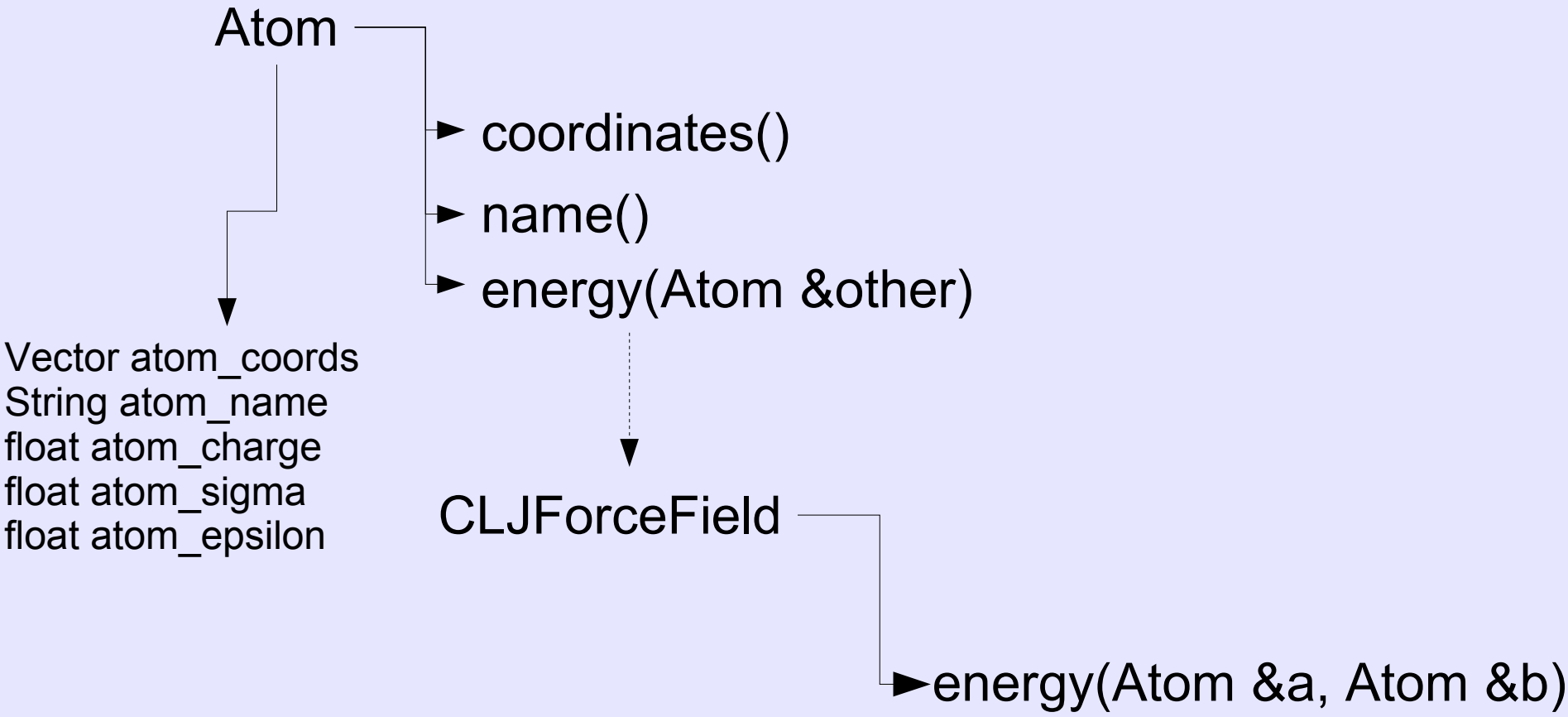


- Use connectivity to split molecule into two
- Rotate both sides about dihedral vector
- Can rotate all parts of the dihedral, or just selected parts



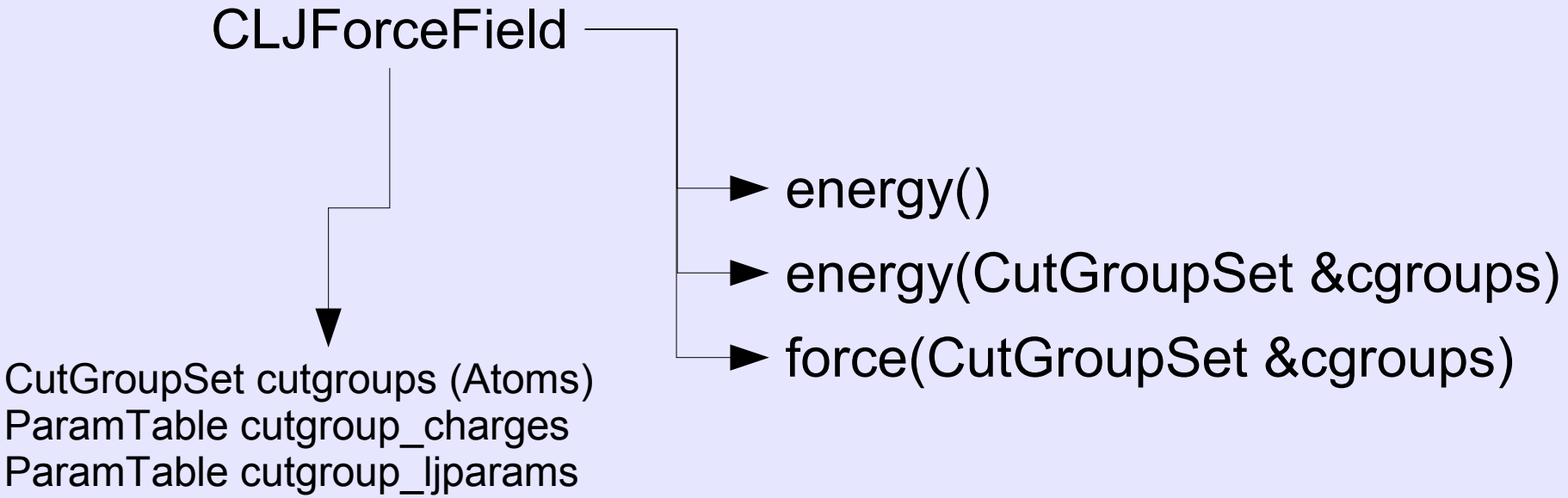
- Use connectivity to split molecule into two
- Rotate both parts about vector between atoms 2-3 of the improper (this keeps atoms 2-3 stationary, and will move atoms 1 and 4, and anything they are connected to)

Calculating Energies and Forces



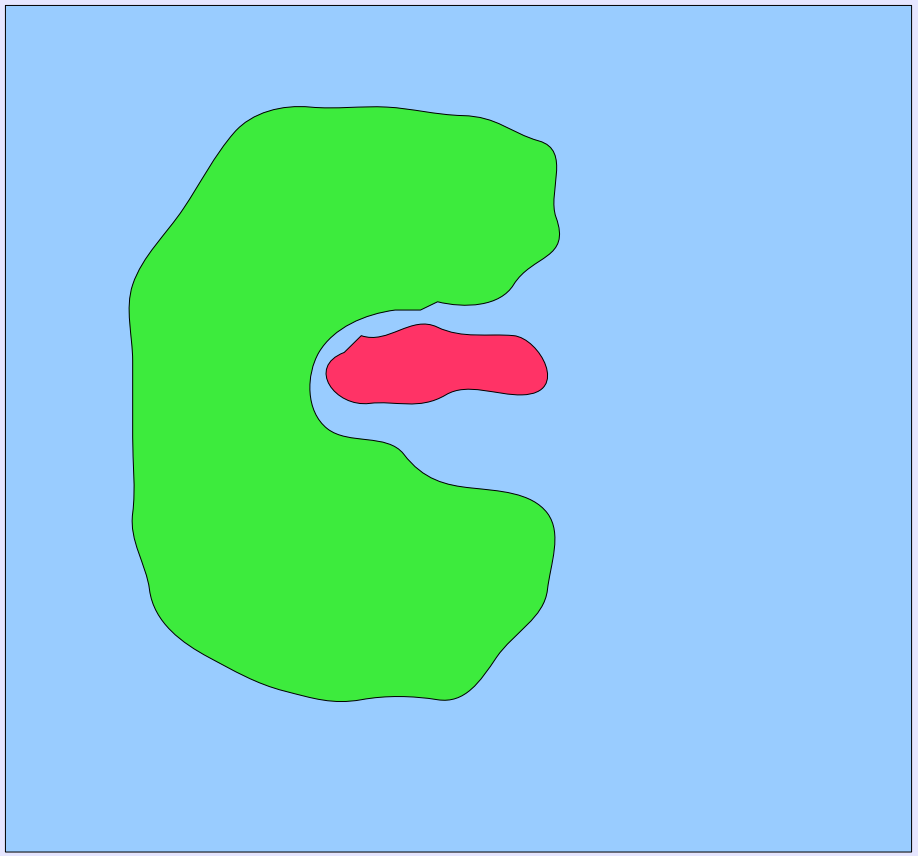
- Calculating energy uses information stored in the atom
- Atom must be compatible with the forcefield
- Calculating energy is slow (function call per Atom pair)

Calculating Energies and Forces



- CutGroups are added to a ForceField (e.g. CLJFF)
- ForceField contains all additional information necessary to calculate the energy (CLJ parameters)
- Atom class needs no additional information
- Calculating energy / force is fast (one function call)

QM/MM example



QM Ligand (red)

MM protein (green)

MM water solvent (blue)

3 ForceFields

(1) QM ForceField

Contains ligand CutGroups

(2) MM ForceField

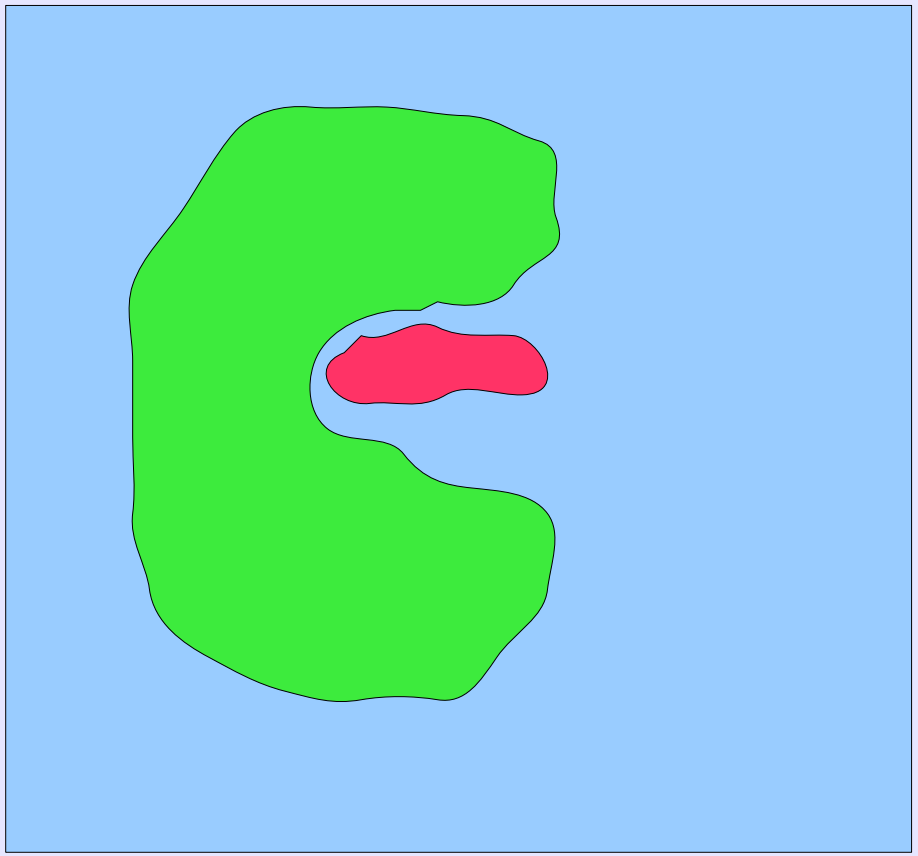
Contains protein and water CutGroups

(3) QM/MM ForceField

Contains all CutGroups

Total energy is sum of the total energies of each of these forcefields

Energy Components



MM Ligand (red)

MM protein (green)

MM water solvent (blue)

(1) MM ForceField
(ligand energy)

(2) MM ForceField
(protein+water energy)

(3) MM Intermolecular FF
(ligand-protein energy)

(4) MM Intermolecular FF
(ligand-water energy)

Use of forcefields has allowed the deconstruction of the total energy into arbitrary components

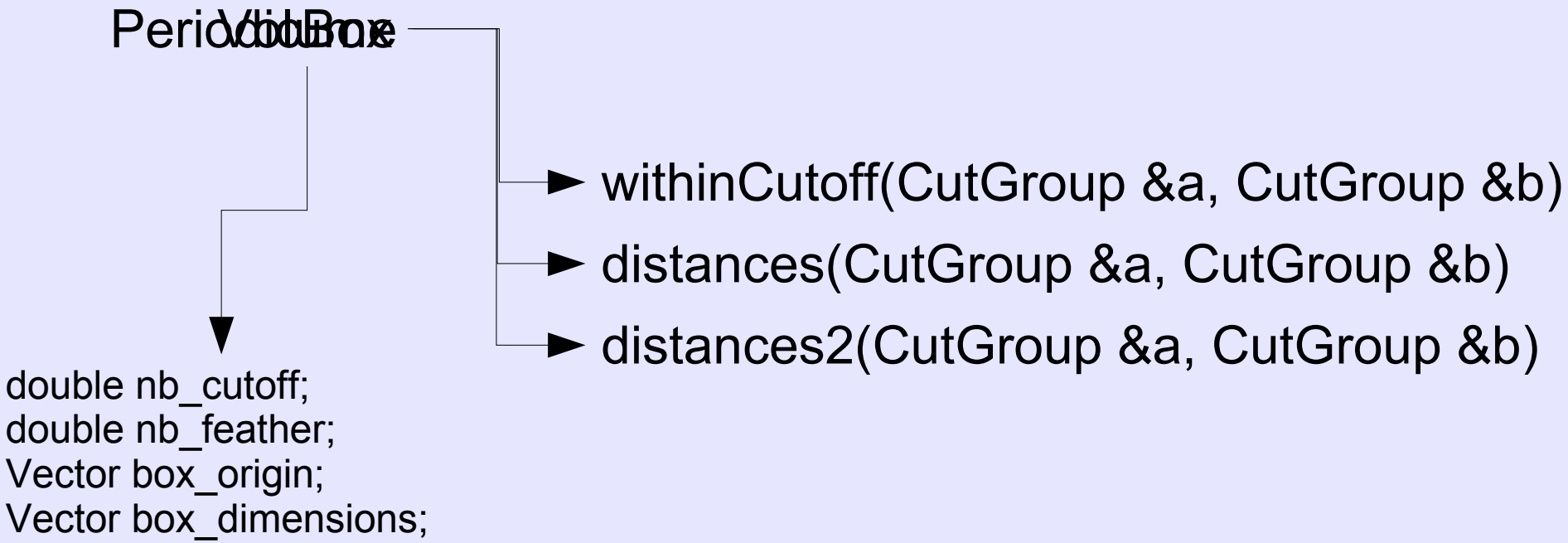
ForceField Flexibility

- ForceField centred design gives incredible flexibility
- ForceField could treat collection of atoms as a guide to which it fits a reduced-representation energy function, e.g. Lipids
- ForceField could be used to call external programs, e.g. Gaussian
- ForceField is completely self-contained – you can write a new one without breaking others

Design Philosophy

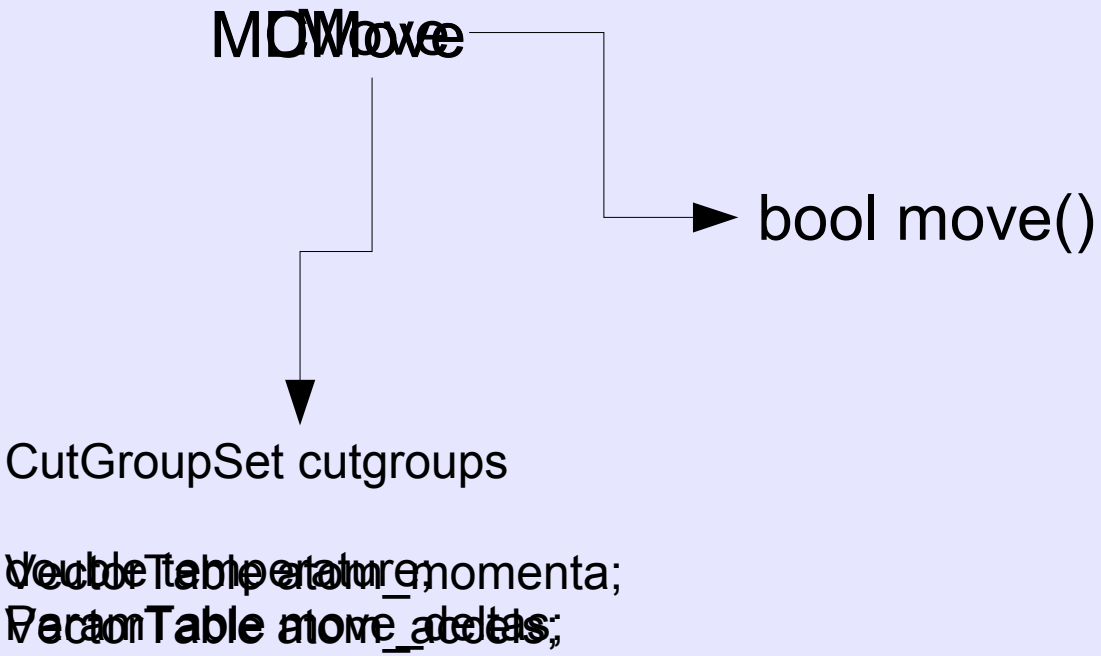
- Use the minimum necessary information to describe the object
- Allow that information to be accessed in a fast and flexible manner
- Place additional information necessary for a calculation in the class that performs that calculation
- Maximise the functionality available from that minimum information

Simulation Volumes



- ForceField class uses the volume class to calculate inter-molecular distances
- Different ForceFields can use different volumes
- Allows multiple cutoff distances in one simulation

Simulation Moves



- Use Move classes to change the system
- MCMove, MDMove, GAMove, MinimisationMove, LambdaMove

Complex Moves

- Complex moves made from existing moves
  - Hybrid Monte Carlo (HMC) combines MD with MC
  - Can derive HMCMove from MCMove and MDMove
  - No need to write new MD or MC code
- Moves can make use of CheckPoint objects
  - Checkpoints can be set whenever you wish
  - You can always roll back to a previous checkpoint
  - Could checkpoint, make some moves, perform a test, then clear the checkpoint if accept, or restore the checkpoint if reject
  - Checkpoints can be nested

Non-linear program flow

- C++ used for all core code
- C++ is useless for user interface
- It is a linear program, and would lead to a linear interface
- Python is a OO scripting language. Beautiful, scripted, non-linear interface
- I have wrapped all of my classes in Python. Interface to Sire is by writing Python scripts
- As code is C++, it has the speed of C++

EditMol

- The Molecule class is not editable
- Instead, I have created a different, editable molecule (EditMol) class
- Fast functions to convert to and from an EditMol
- EditMol provides the easy, user-view of a molecule
- Different classes to read different molecule files, creating lists of EditMols
  - EditMolList mols = PDB().read(“waterbox.pdb”)

```
tip4p = EditMol.create("TIP4P")

residue = tip4p.addResidue(1, "WTR")

residue.add("H01")
residue.add("H02")
residue.add("O00")
residue.add("M03")

residue.addBond("H01","O00")
residue.addBond("H02","O00")
residue.addBond("M03","O00")

residue.setBond("H01","O00", 0.96)
residue.setBond("H02","O00", 0.96)
residue.setBond("M03","O00", 0.3)

hoh = Angle.degrees(109.5)
residue.setAngle("H01","O00","H02", hoh)

residue.setAngle("M03","O00","H01", 0.5*hoh, “H01”)
residue.setImproper("M03","O00","H01","H02", Angle.degrees(0.0))

waters = PDB().load(“spcbox.pdb”)

for water in waters:
        water.applyTemplate(tip4p)

PDB().write(waters, “tip4pbox.pdb”)
```

Create a new EditMol

Add a residue

And the atoms to this residue

Add interatomic bonds

Set the bond lengths

Set the angle

Now the improper angle

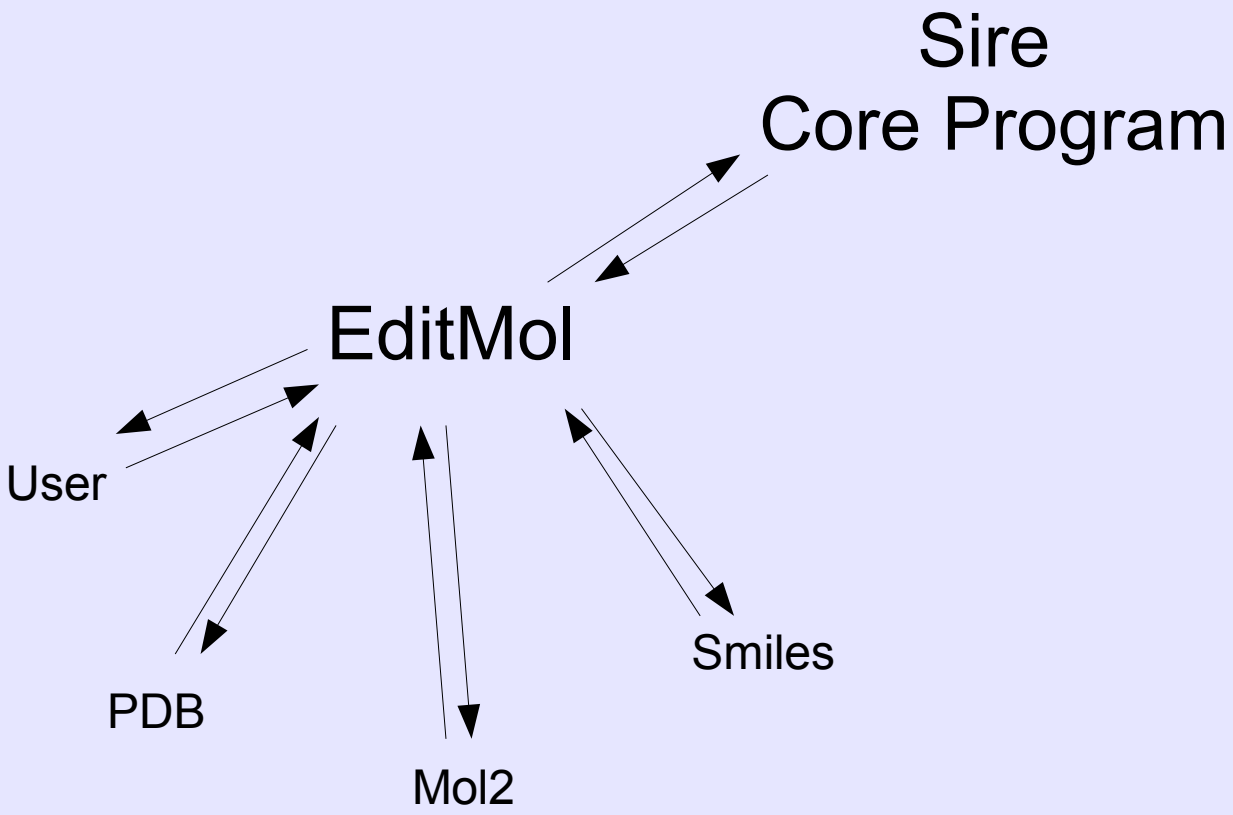
Load an SPC water box

Use tip4p as a template to convert all the SPC

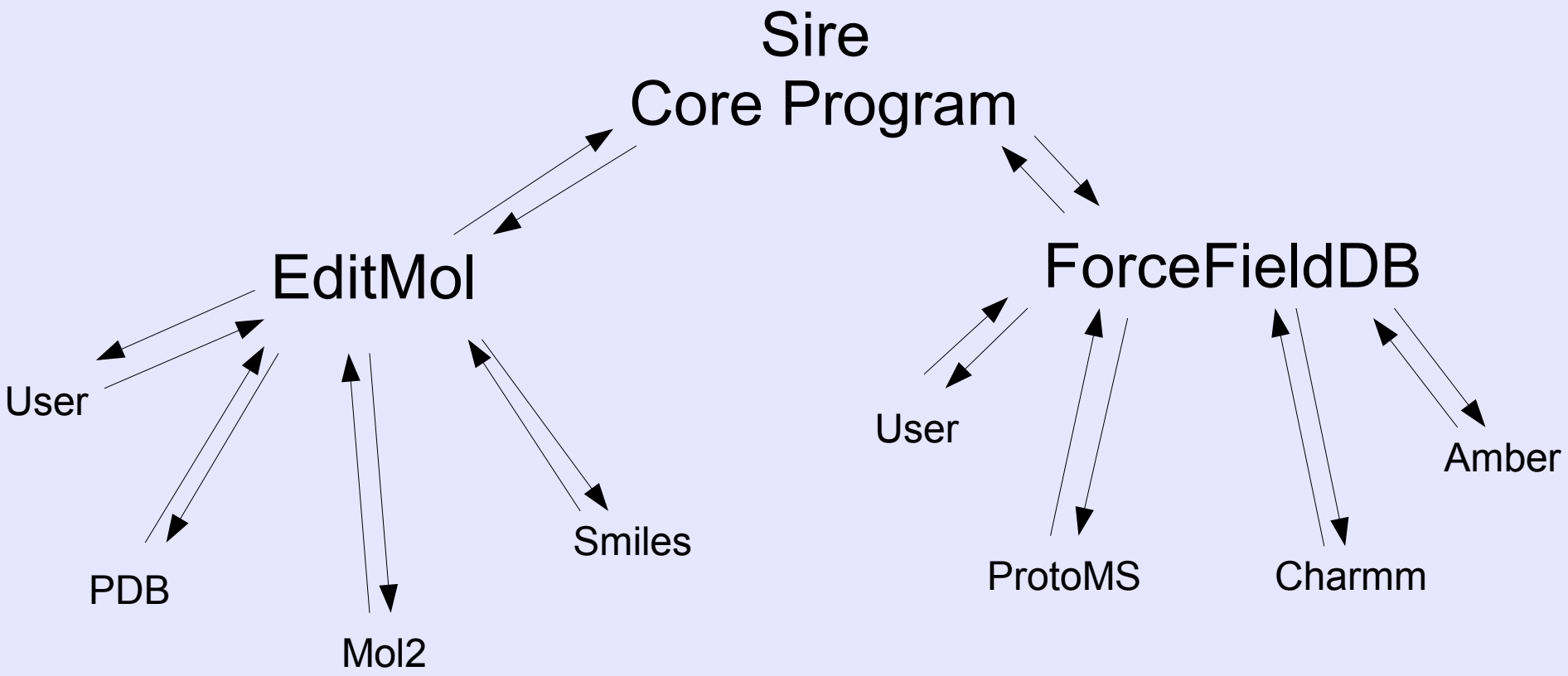
Write out the new TIP4P box

Buffering

- EditMol acts as a buffer between the program and the user
- Users create EditMols, either via reading files or manual creation, and then pass these to the core program
- The core program needs only work with EditMols, and does not need code to deal with PDB, Mol2, XYZ etc.
- Core program outputs an EditMol, which can then be saved via the same IO classes/actions



- Easy to add support for a new file format
- File code can be written in C++ or Python
- Could use code on its own for file format conversion, or file editing



- Easy to add support for a new file format
- File code can be written in C++ or Python
- Could use code on its own for file format conversion, or file editing

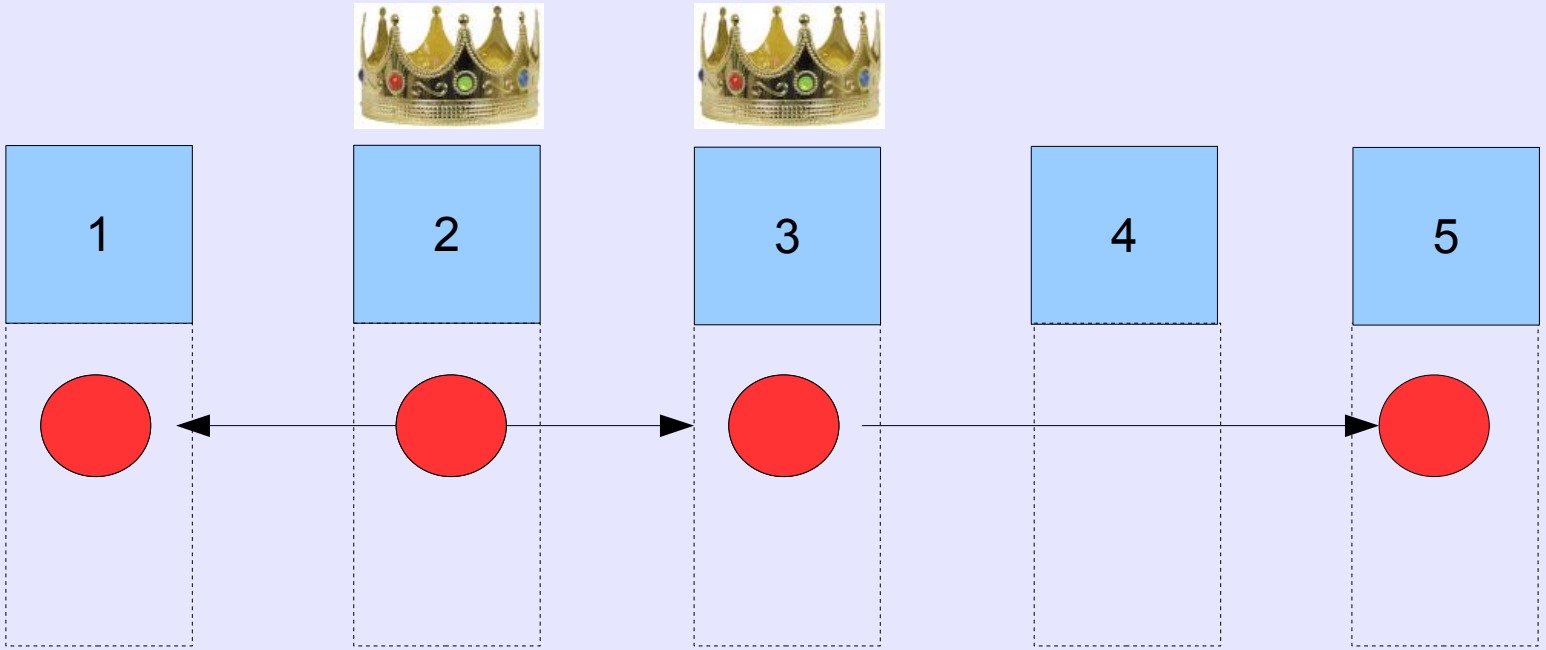
SQL Parameter Database

- ForceFieldDB holds forcefield parameters
- An in-process, in-memory SQL database class is used as a buffer between the user and the core program for forcefield parameters
- User loads parameters into database (manually, or from a file), and the core code then extracts parameters from the database using database searches

Networking

- Handling parallel jobs manually is very frustrating! (ssh problems, nfs problems, reliability, custom scripts etc.)
- Sire handles all networking itself!
- Sire uses MPI to manage nodes
- Created two subsystems to handle networking;
  - NetObjects: Responsible for sending objects over the network and maintaining consistency
  - IPI: Responsible for controlling remote objects in an application transparent manner

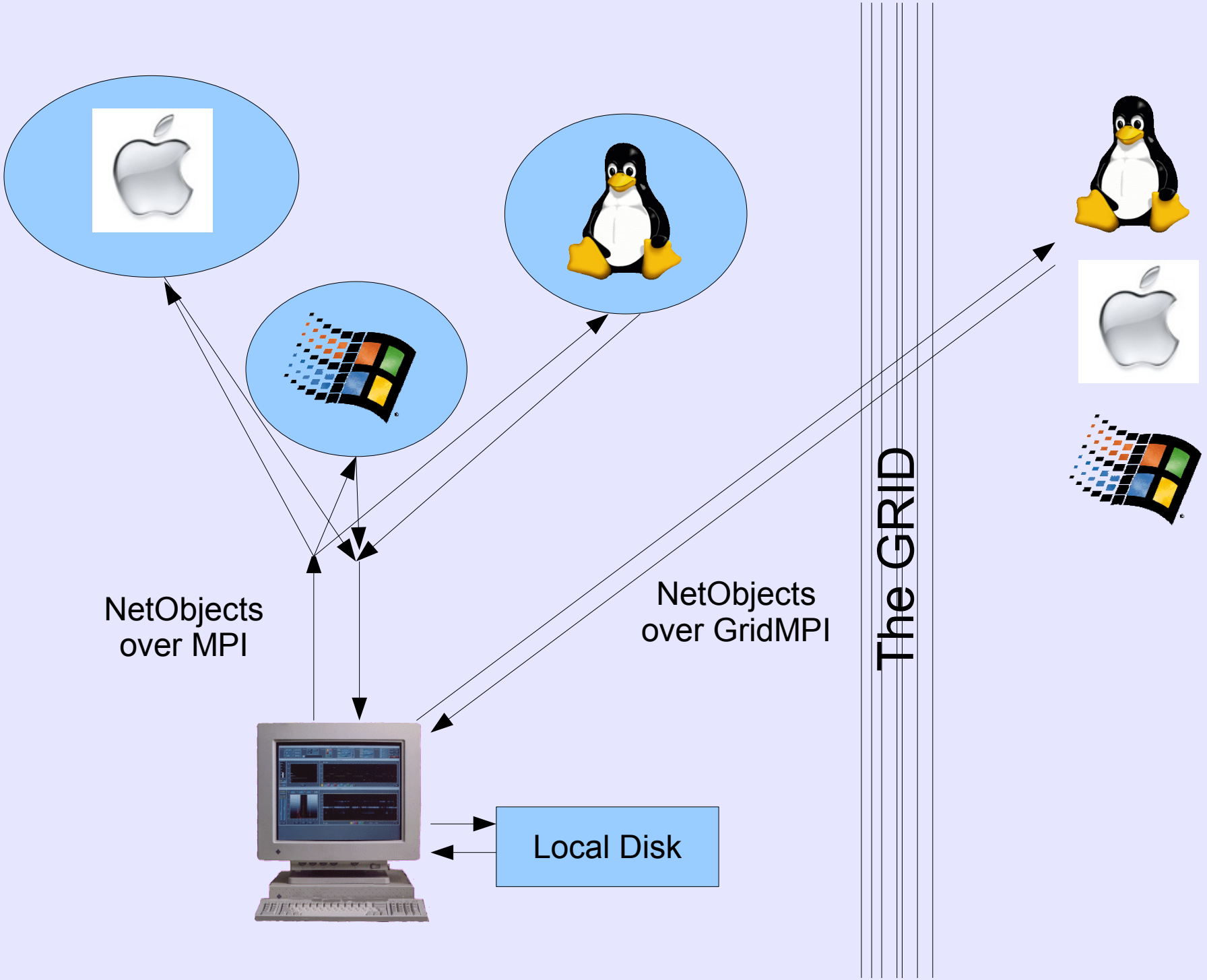
NetObjects



- Sending objects between processors
- Optimised for read often, change little (norm for molecular simulation)

NetObjects

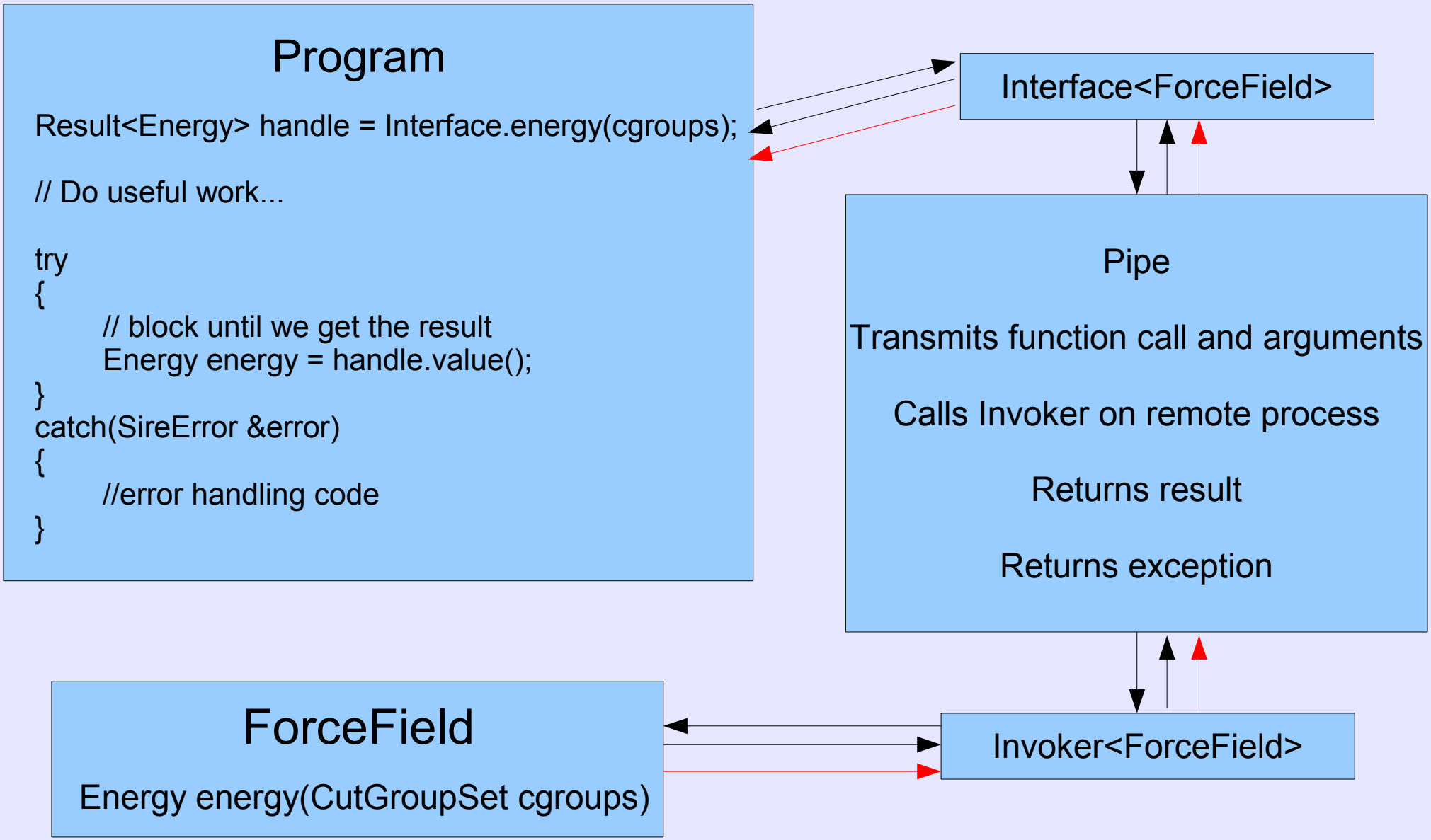
- Advantage of NetObjects is that only one process needs access to the disk
  - e.g. Only your desktop processor needs to be able to read the input files
  - It can create the Molecule/ForceField objects, then use NetObjects to send them to the other processors in the cluster
  - The other processors can return their results via NetObjects as well
- No need for NFS or other shared disk resource
- Ideally suited to the GRID (via GridMPI)


NetObjects
over MPI
NetObjects
over GridMPI
The GRID
Local Disk

Interface-Pipe-Invoker

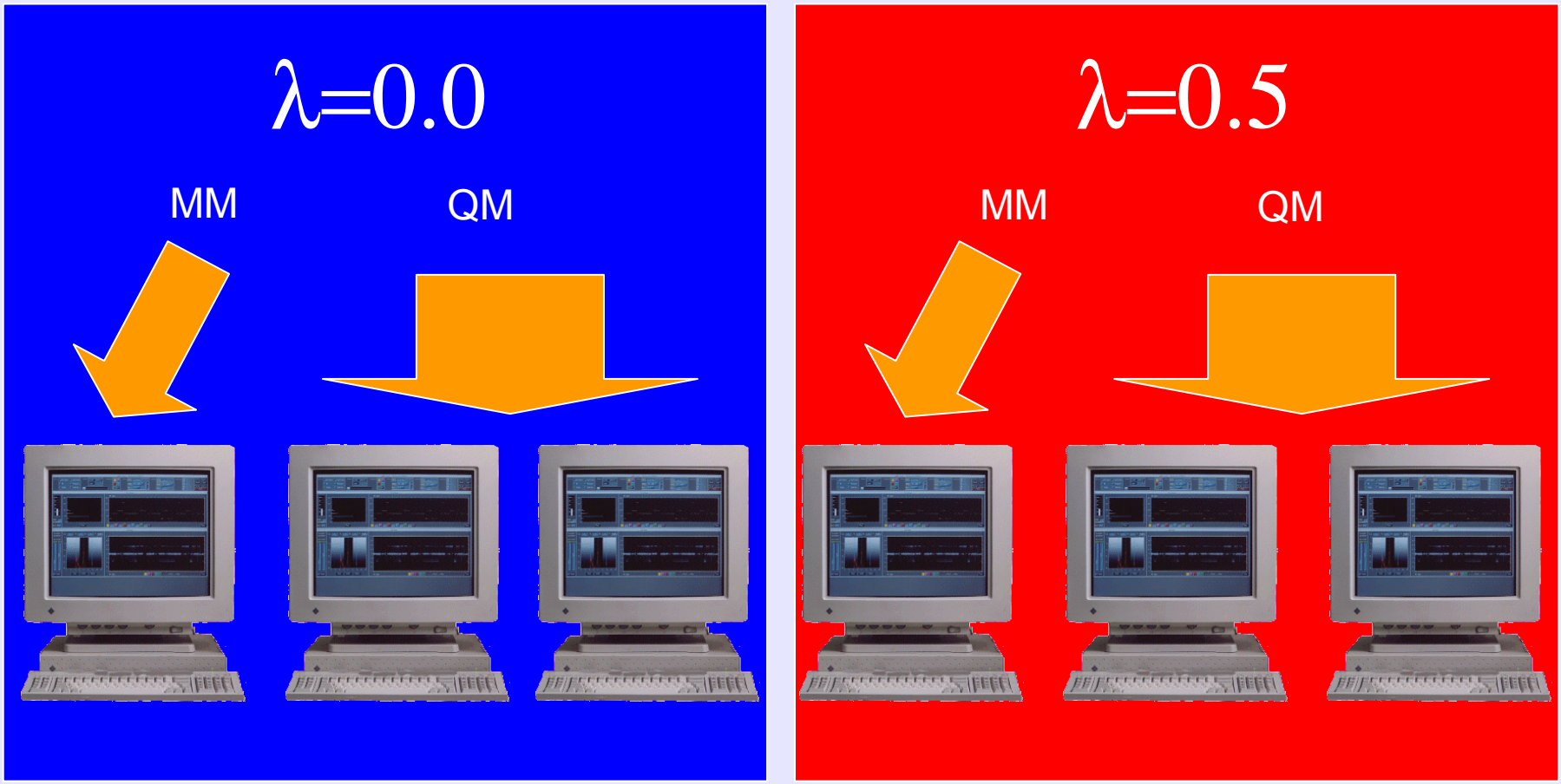
- Its not sufficient to just send objects to remote processors
- You also need to be able to control them!
- Developed Interface-Pipe-Invoker (IPI) system to remote-control objects
- Class to be controlled is wrapped up by its own Interface and Invoker class

Interface-Pipe-Invoker



Parallel Calculations

- IPI allows energy calculations to be split over multiple processors

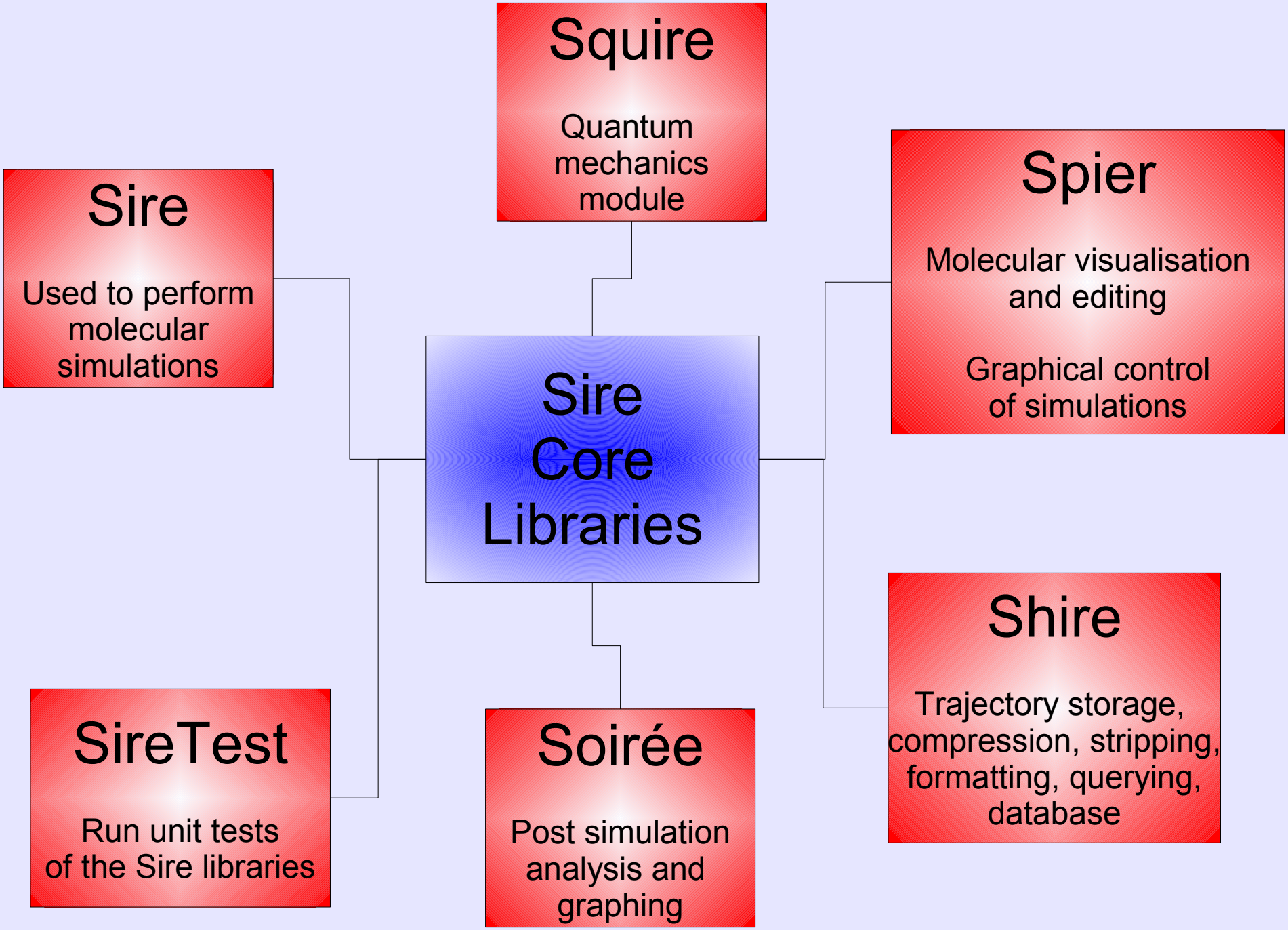


Current Status

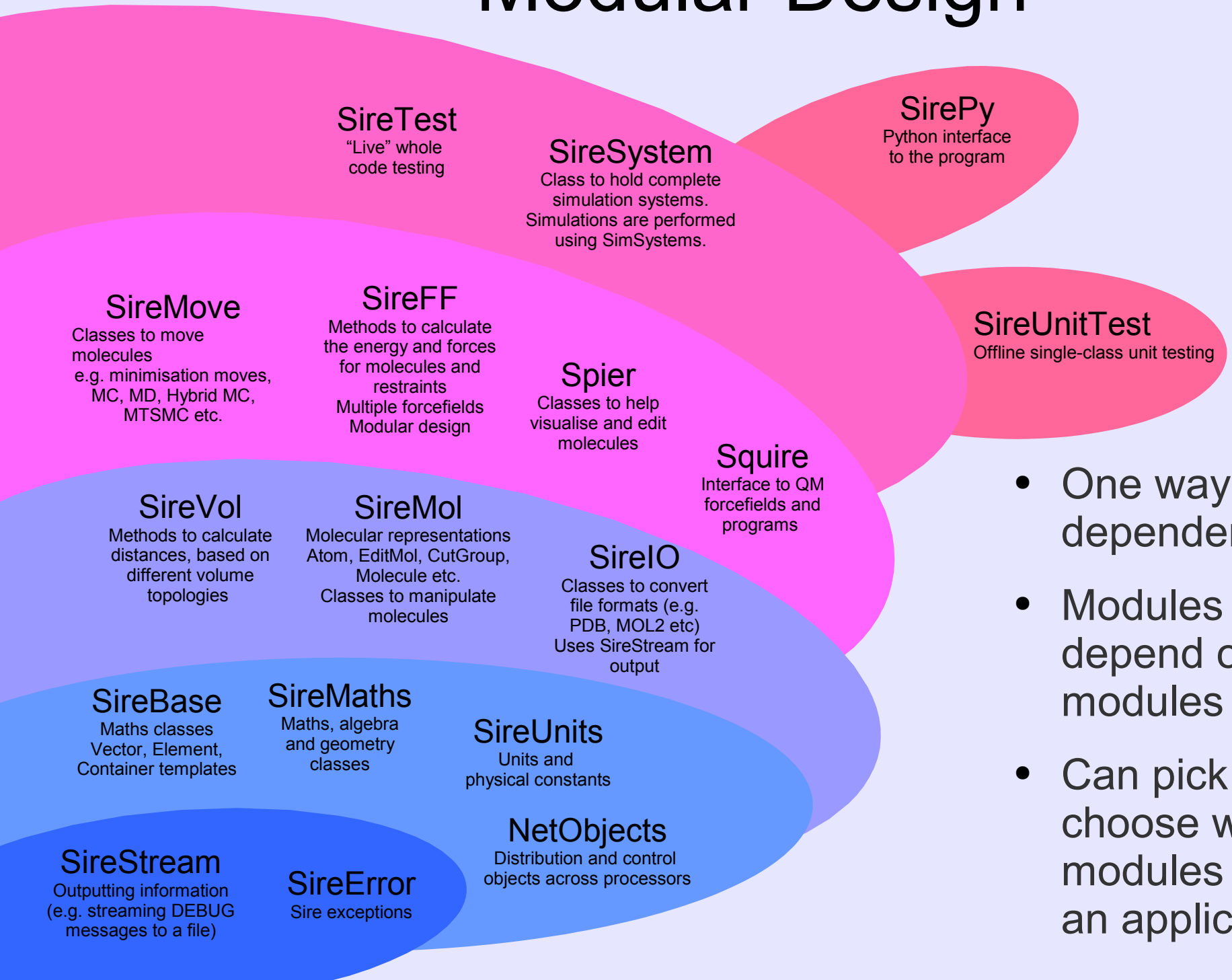
- Most of what I have presented is implemented!
- Energy calculations are faster than ProtoMS!
- Nearly 50k lines of code
- I hope to be running simulations by the end of February
- I plan to release an alpha version in the summer, a beta version around Christmas
- My plans are, however, even more ambitious...

Modular Design

- Sire is a collection of modular libraries (SireMaths, SireMol, NetObjects etc.)
- Can be combined together to make different applications
- Open source, so can be combined with existing open source libraries
- Rapid application development!
- I plan a range of applications based on the core Sire libraries...
- Spier – molecular visualisation and editing


Squire
Quantum mechanics module
Sire
Used to perform molecular simulations
Spier
Molecular visualisation and editing
Graphical control of simulations
Sire Core Libraries
Shire
Trajectory storage, compression, stripping, formatting, querying, database
SireTest
Run unit tests of the Sire libraries
Soirée
Post simulation analysis and graphing

Modular Design



- One way dependencies
- Modules only depend on modules below
- Can pick and choose which modules to use in an application