



Sire: A powerful and flexible framework for molecular simulation

Christopher Woods

Centre for Computational Chemistry
University of Bristol, UK



Structure



- Part I: Background
- Part II: Design
- Part III: Current Status and Future

Feel free to ask questions at any time



Part I: Background



History



- Ph.D. co-funded by industry (UCB)
- Implementing new methods in existing academic software took significant time
- Technology transfer between academia and industry hindered by software licensing and overall poor-usability of academic code



ProtoMS



- ProtoMS - 20k line F77 program for protein-ligand Monte Carlo binding free energy calculations
- Order of magnitude faster than existing software
- Extendable, modifiable, clean code
- Now in use by other academics



ProtoMS Problems



- Fortran is difficult to extend cleanly
- Lots of different versions and authors
- Varying code quality
- Limit to what can be written - the original design is “set in stone”



The Vision



- Vision for new simulation program
- Extendable, modifiable, reusable, flexible
- Fast, easy to understand, documented
- A simulation framework, that could be used to build any simulation program



Sire



- February 2005 - started to write Sire
- C++ libraries, with objects exposed via Python
- Run simulations by writing Python scripts that combine robust simulation building blocks
- Easy for new Ph.D. students, but no black box



End of part I

Questions?



Part II: Design



Robust Objects



- Sire is built using robust C++ objects
- Objects are manipulated via well-defined and clean interfaces
- Many objects have different interfaces depending on the actions applied to them
- `mol = mol.move().translate([1,2,3])`



Molecule Class



Molecule



Properties

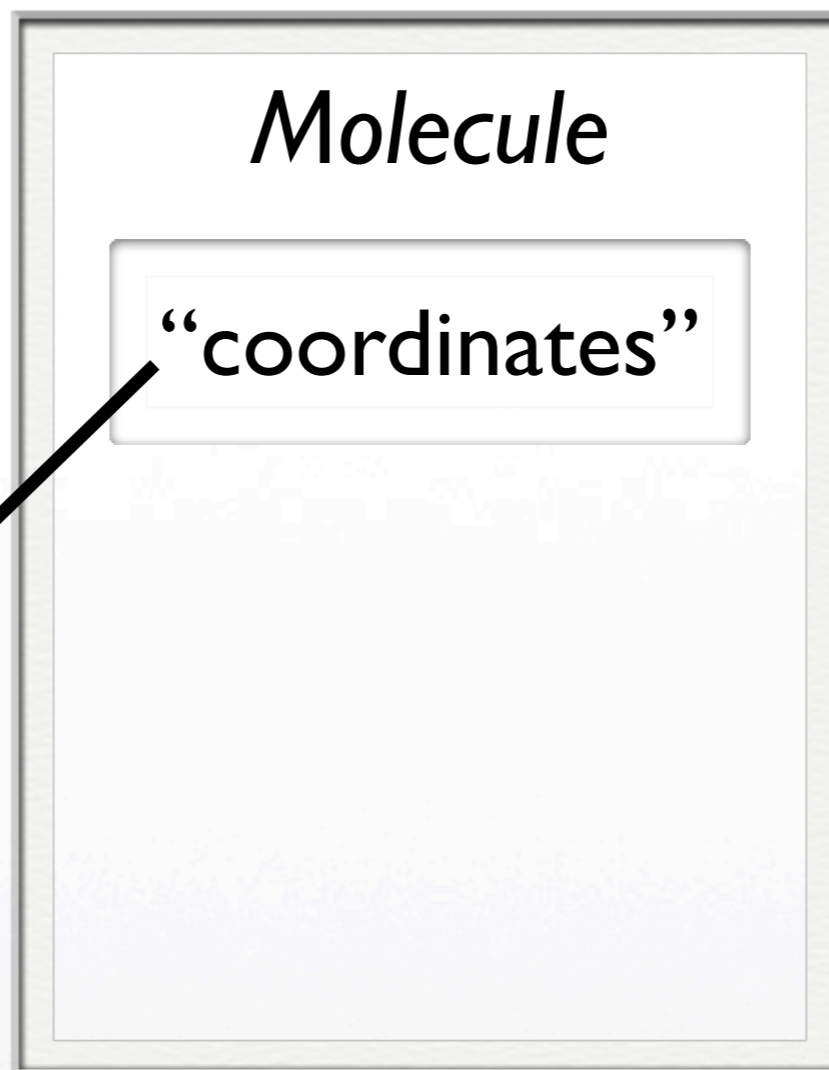


Molecule

0.0, 0.0, 0.0
1.0, 0.0, 2.0
2.0, 0.0, 1.0
4.3, 9.2, 1.6



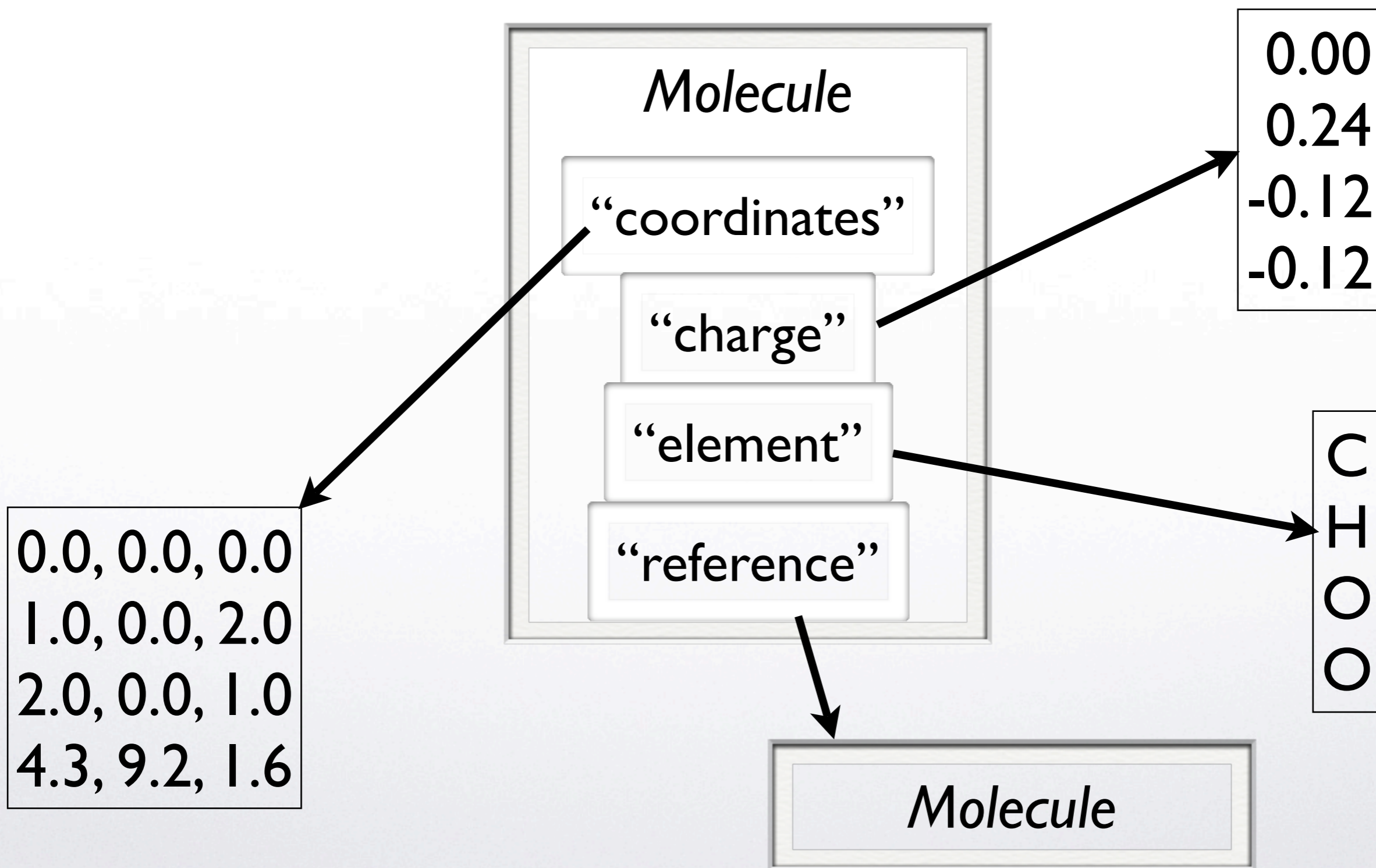
Properties



0.0, 0.0, 0.0
1.0, 0.0, 2.0
2.0, 0.0, 1.0
4.3, 9.2, 1.6



Properties





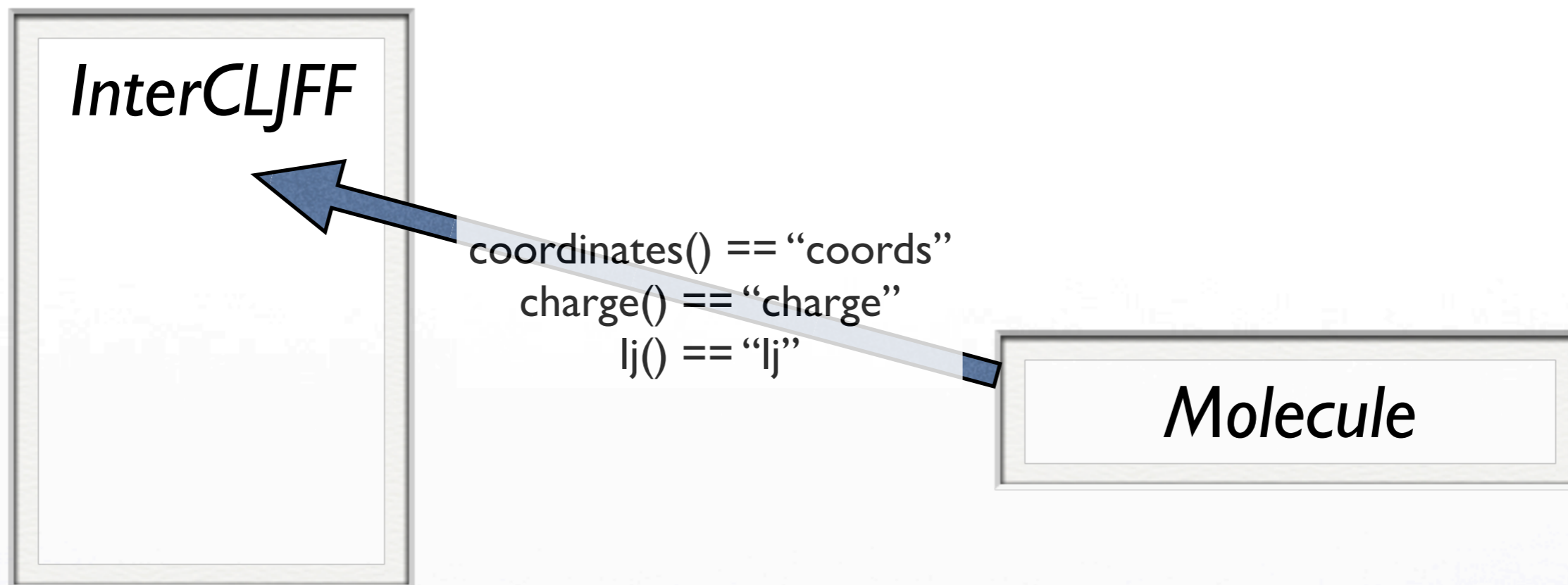
ForceField



- ForceField is the key class used to evaluate energies and forces between molecules
- InterCLJFF - this calculate the intermolecular coulomb and LJ energies and /or forces between its contents
- QMMMFF - this calculates the QM/MM energies and forces between QM and MM molecules



ForceField



```
ff.add( mol, {ff.coordinates() == "coords", \  
             ff.charge() == "charge", \  
             ff.lj() == "lj"} )
```



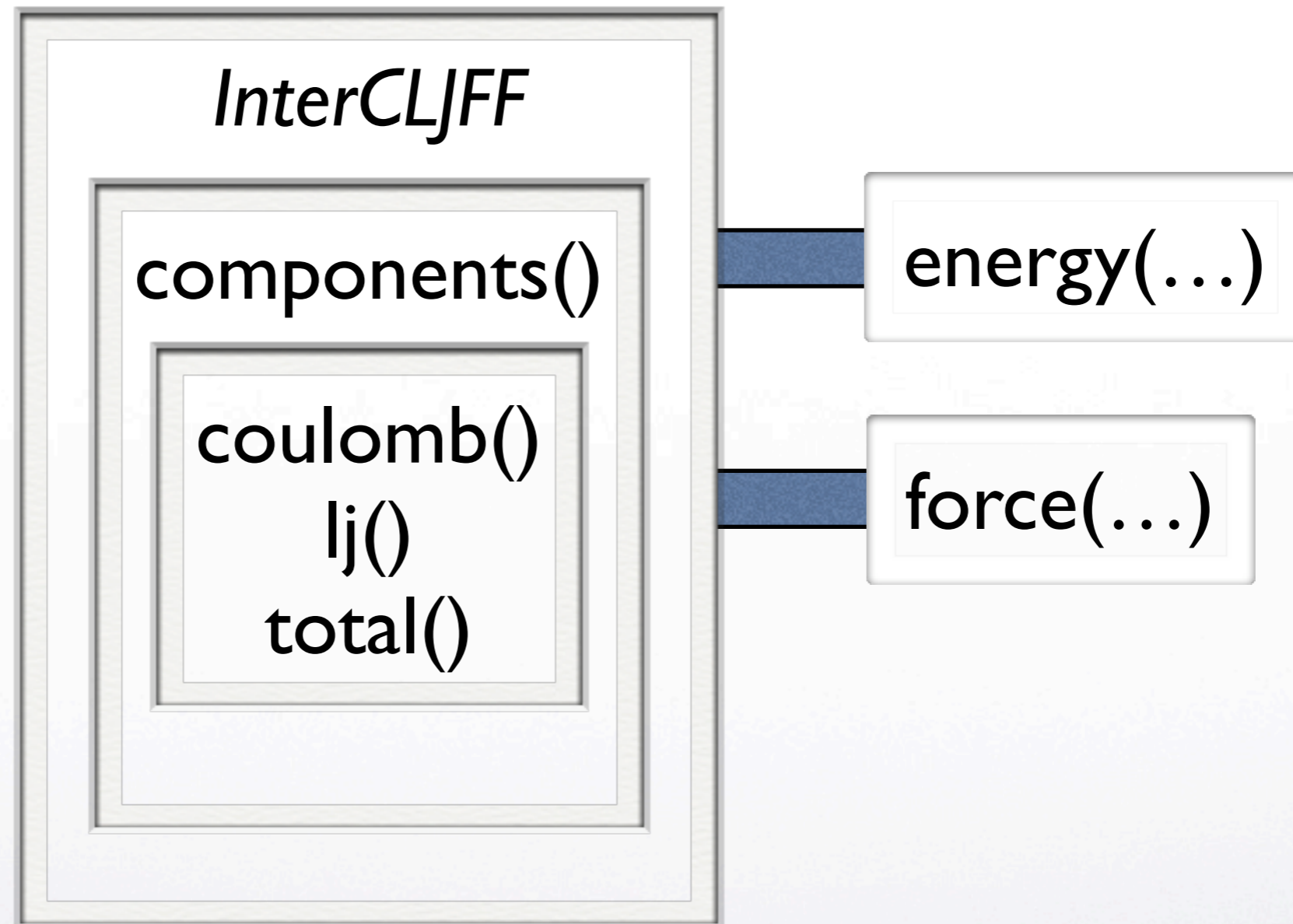
Flexibility



- Lots of different forcefield classes
- Lots of different property classes
 - (indeed, molecule can contain a forcefield or another molecule as a property)
- Easy to write new forcefield or property classes, so easy to extend Sire for new applications



Components



```
nrg = ff.energy( ff.components().coulomb() )  
ff.force( forces, ff.components().lj() )
```



Combinations



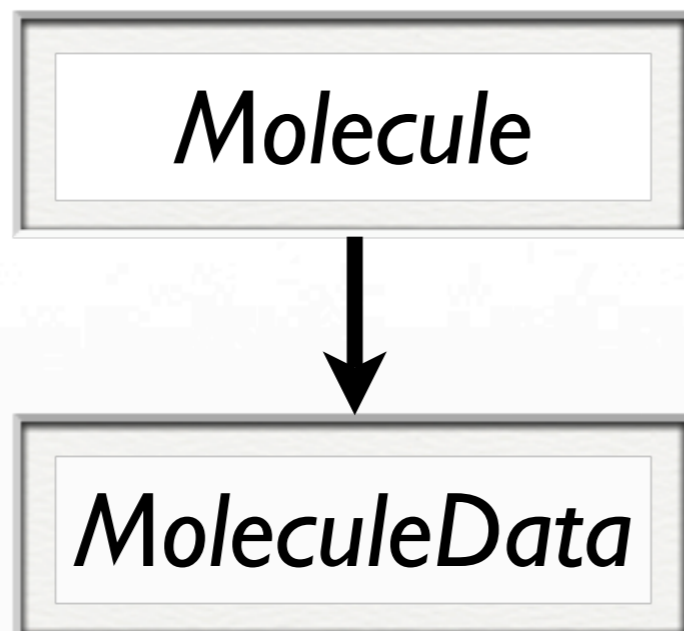
$$(1 - \lambda) \times \boxed{QMMMFF} + \lambda \times \boxed{InterCLJFF}$$

lam = Symbol("lambda")

total_nrg = (1 - lam) * qmff.components().total() + \ lam * mmff.components().total()

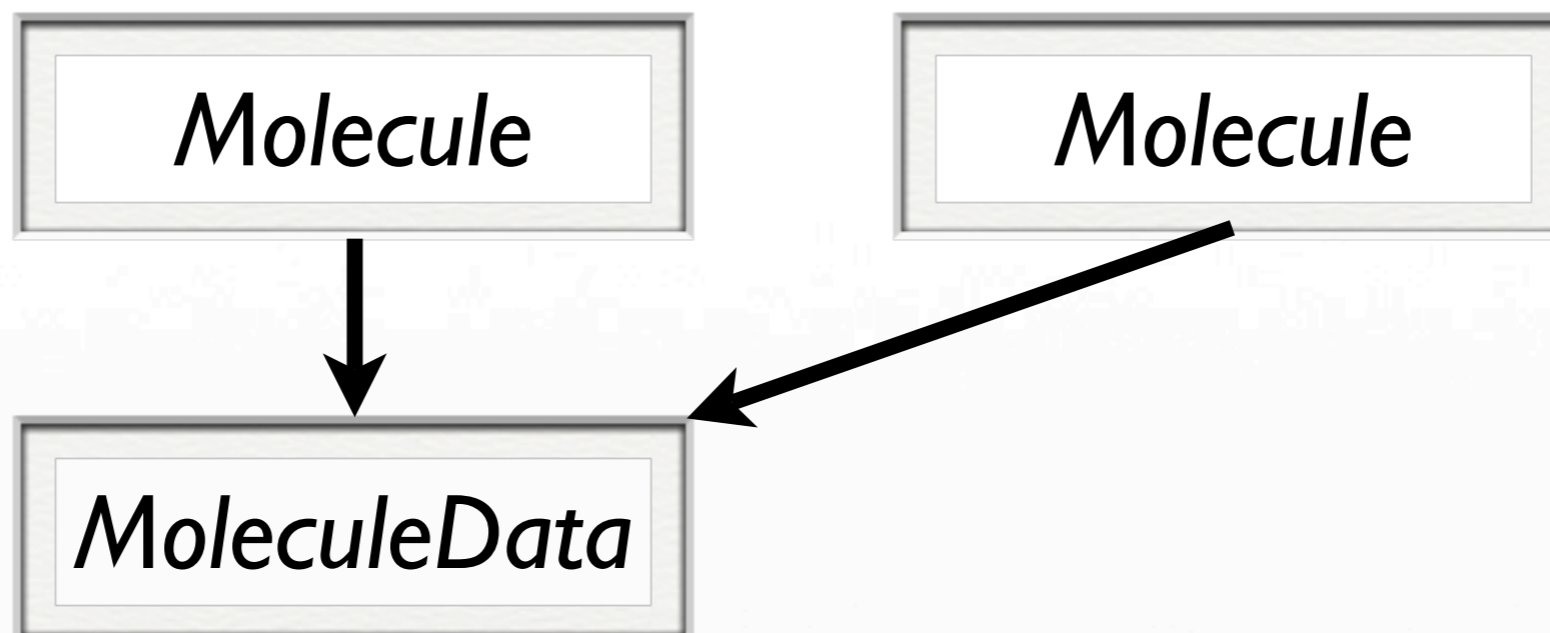


Implicit Sharing





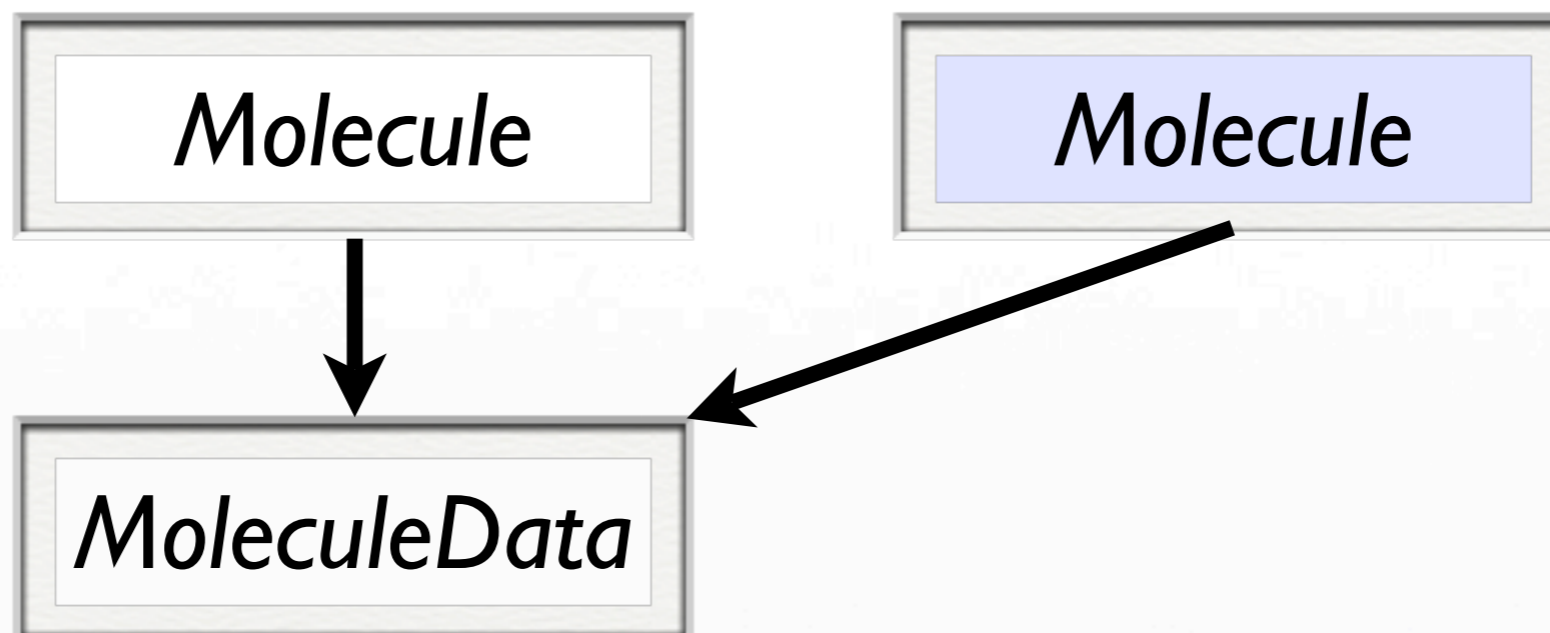
Implicit Sharing



Copying an object merely copies the pointer to the object's data



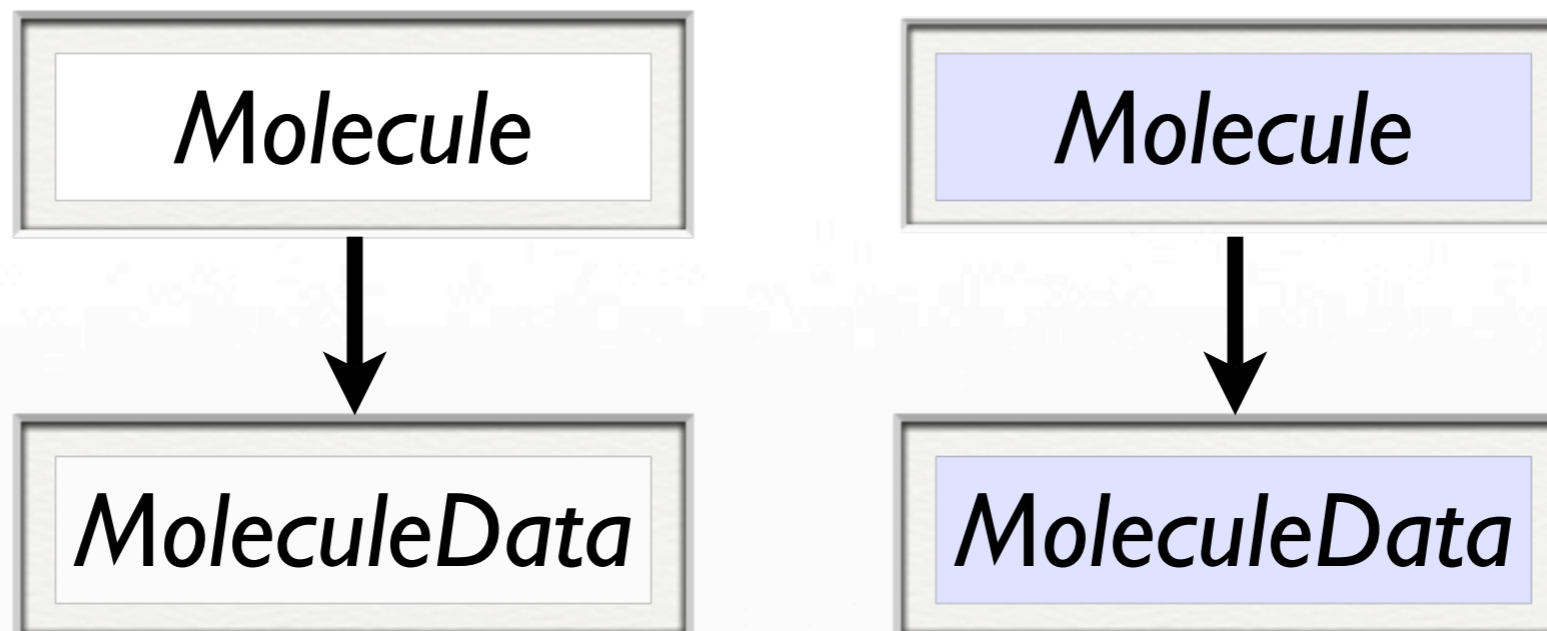
Implicit Sharing



Copying an object merely copies the pointer to the object's data



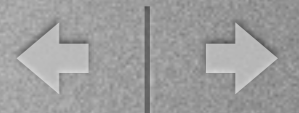
Implicit Sharing



When you edit (write) to either copy, then a true copy of the data is made, and the pointer updated
(Copy on write)



Copies Everywhere



- Implicit sharing is used everywhere
- Molecules hold implicitly shared copies of properties
- Forcefields hold implicitly shared copies of molecules
- No dangling pointers, highly thread-safe



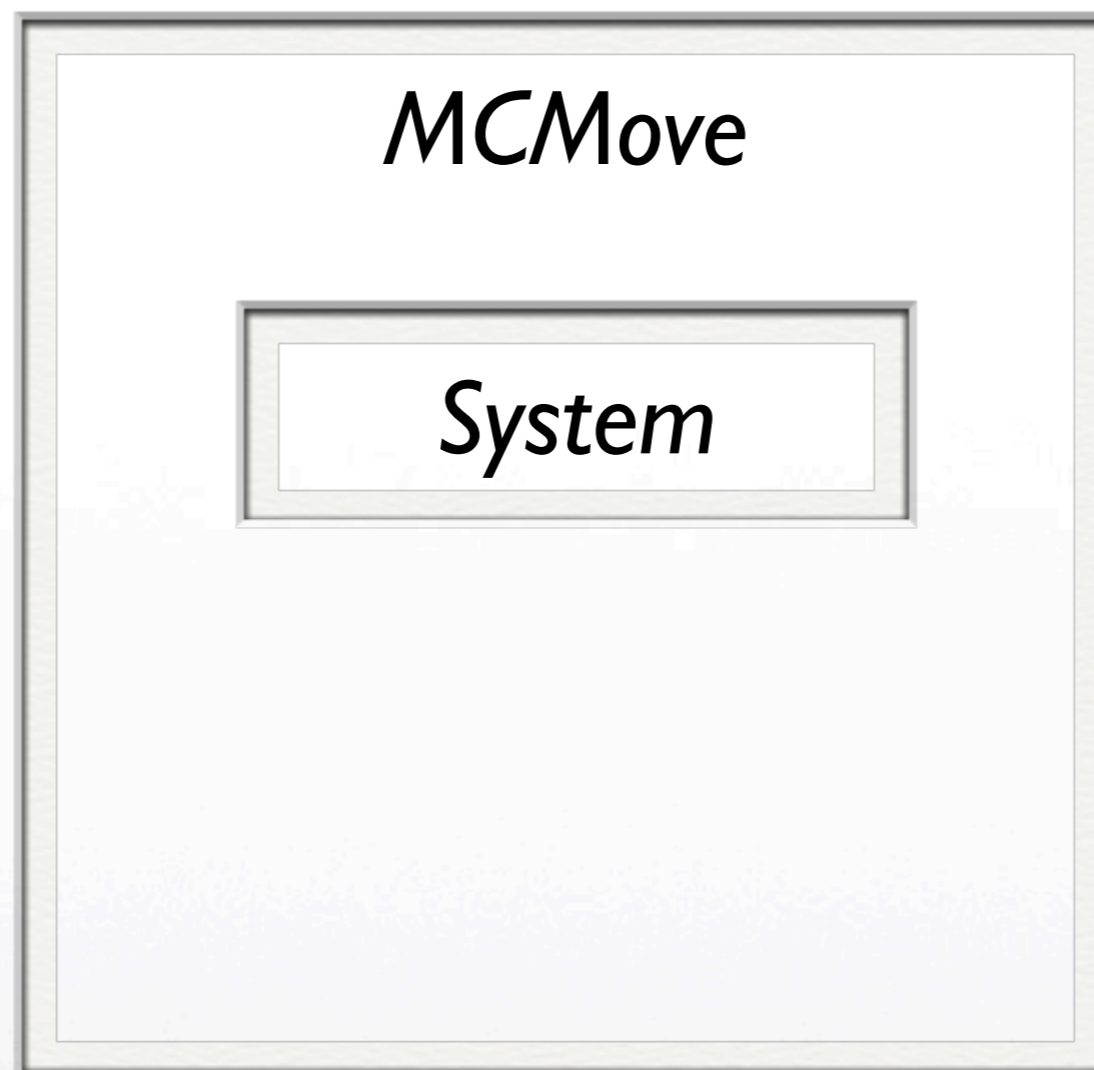
Monte Carlo



*MCM*Move



Monte Carlo





Monte Carlo



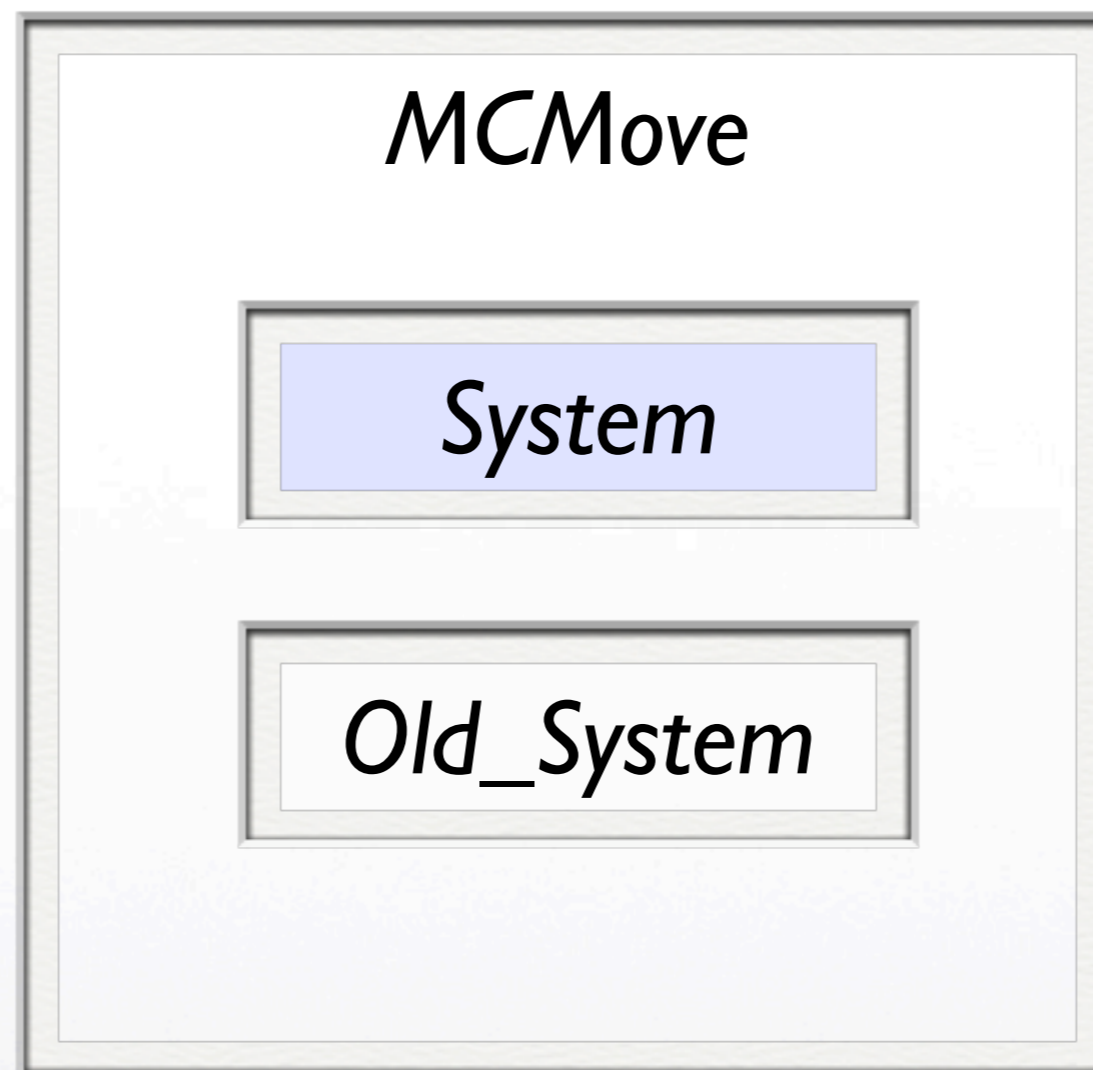
MCMove

System

Old_System



Monte Carlo





State



- Implicit sharing allows the object state to be stored in memory
- Objects can be serialised (marshalled) to a binary, version controlled format
- Allows the state of an object to be written to disk or database (persistent storage)
- Object can be transmitted over the network



Reusable Bricks





End of part II

Questions?



Part III:

Current Status and Future



Current Status



- Summer 2007 released alpha version
- Used to perform QM/MM Monte Carlo free energy calculations on water
- November 2007 released second alpha version
- Used to perform (basic) QM/MM Monte Carlo simulations on protein/ligand system



Rewrite



- December 2007 - present
- Significant rewrite of core objects to clean the design and remove cruft
- Expect to be finished soon
- Rewrite has enabled new features, like molecular dynamics, protein perturbation



Immediate Future



- Sire is being used by two students at Bristol
- I'll soon be re-running protein-ligand free energy calculations on neuraminidase
- We will port Sire to a national supercomputer during the summer for computational enzymology calculations



Future



- My next projects will involve implementing absolute binding free energy methodology and coarse grain forcefields in Sire
- Sire is, and will be used for all my research
- Over the next year, Sire will be used by more students and collaborators via limited roll-out



Open Source



- Sire is, and always will remain open source
- GPL - all contributions are shared equally
- All development online - <http://siremol.org>
- From there, you can download **any** past version of the code



Conclusion



- Sire is an efficient, highly flexible and promising framework for molecular simulation
- Still in development, but has been used by others, and for published science
- Aim is to provide a common platform for collaboration between academics, and between academics and industry



Acknowledgements



- Universities of Bristol and Southampton for allowing me to write and release this software
- UCB for providing initial support, and for allowing ProtoMS to be released under an open license
- Prof. J.W. Essex, Dr. J. Michel, Dr.A. Mulholland and Dr F. Manby, Dr. R. Taylor and Dr. M. King for useful discussions
- Get Sire at <http://siremol.org>