Streamlining and sharing molecular simulation data flows with BioSimSpace

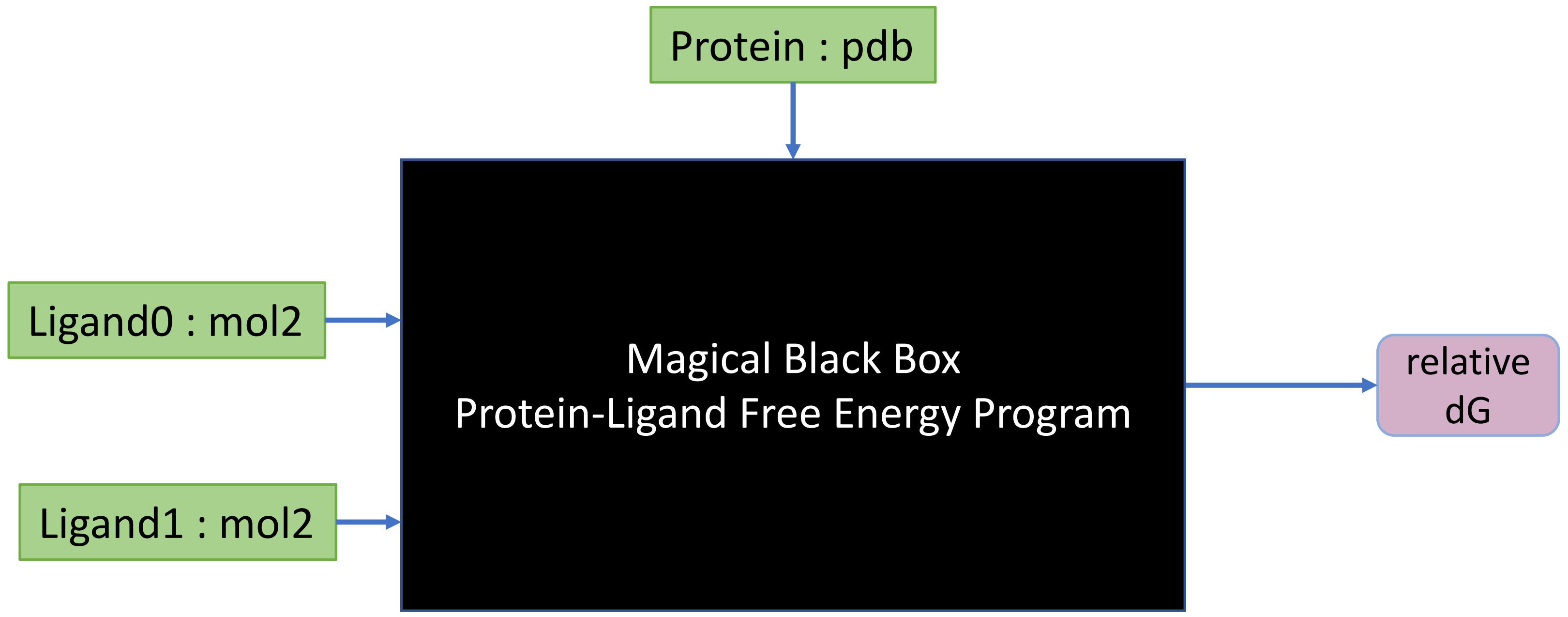
Christopher Woods

EPSRC Research Software Engineering (RSE) Fellow

RSE Group, Advanced Computing Research Centre

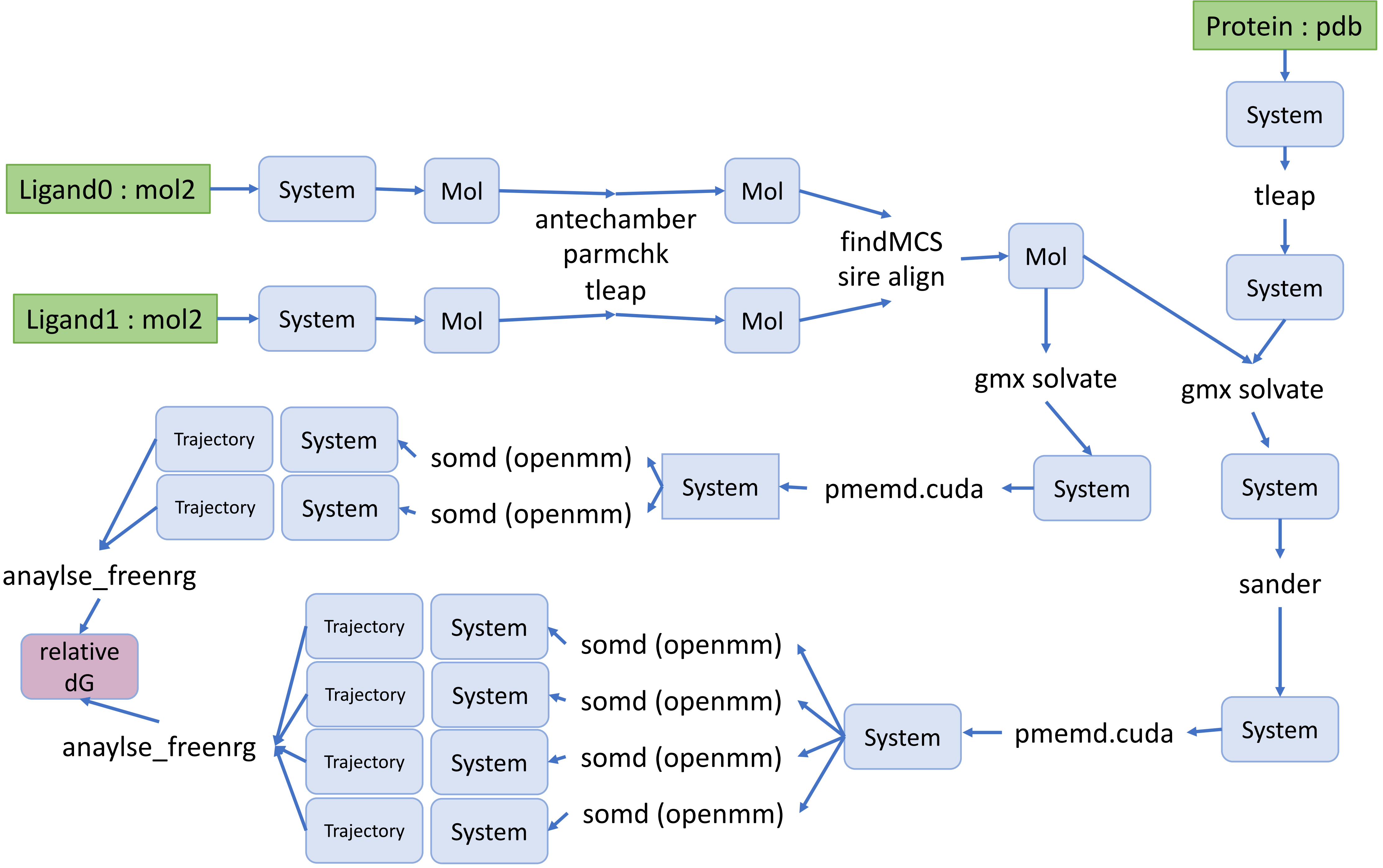
University of Bristol

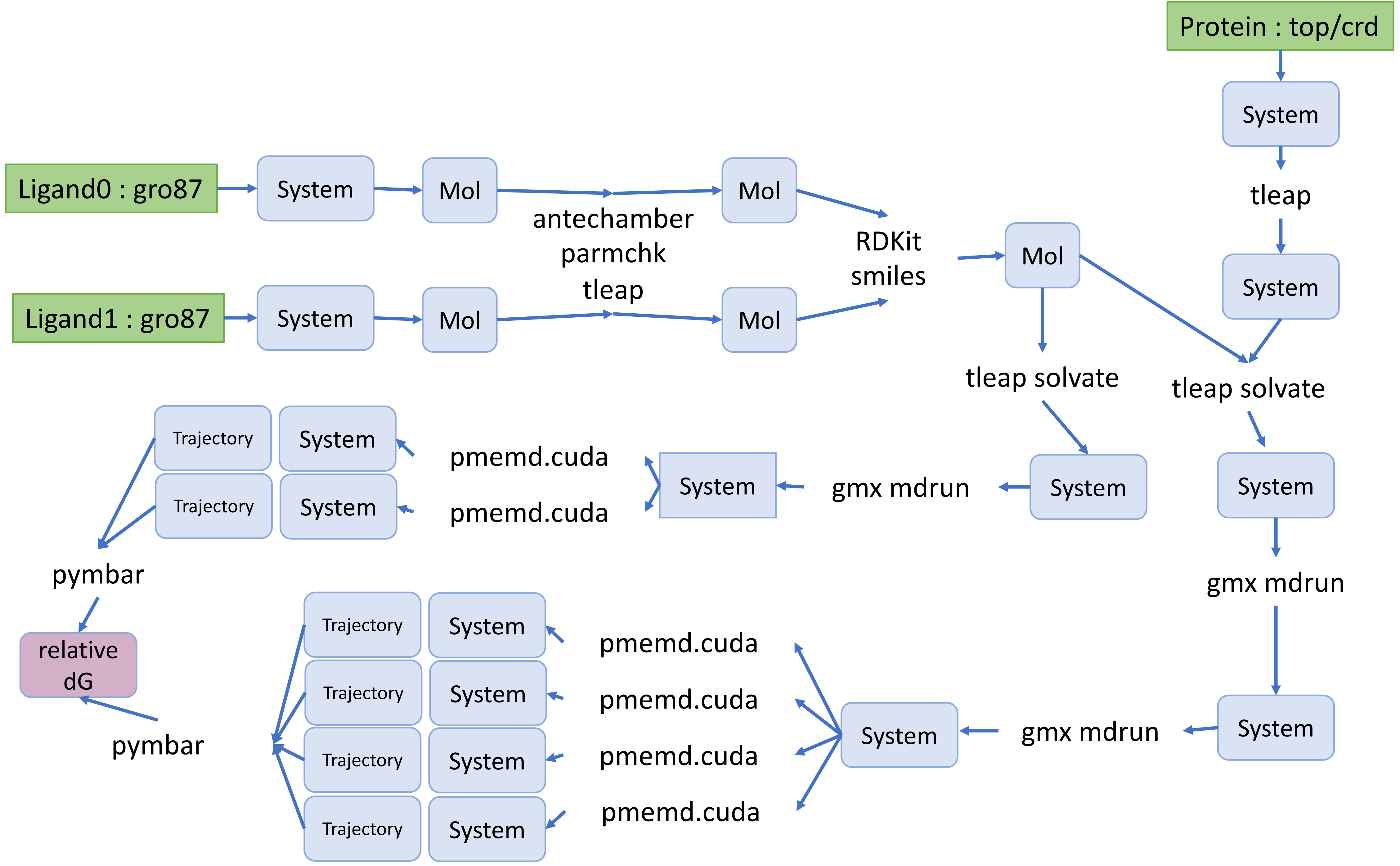
Calculating Relative Binding Free Energies



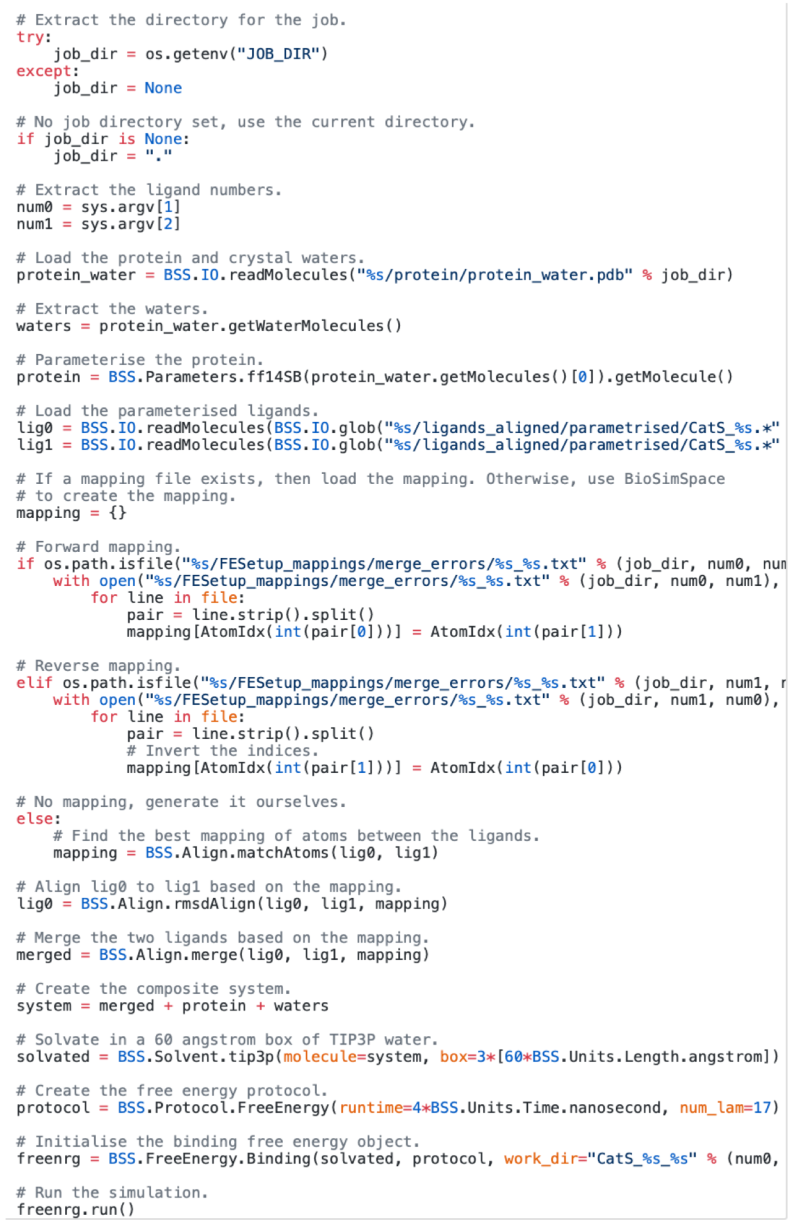
How can we share the protocols and data flows from the above calculation?

…just publish the scripts on GitHub?
…and the input files…
…and the input/output data…?


Protein : pdb
System
tleap
System
Ligand0 : mol2
System
Mol
antechamber
parmchk
tleap
Mol
Ligand1 : mol2
System
Mol
Mol
findMCS
sire align
Mol
gmx solvate
gmx solvate
System
System
pmemd.cuda
System
somd (openmm)
somd (openmm)
System
System
Trajectory
Trajectory
anaylse_freenrg
relative dG
sander
System
pmemd.cuda
System
somd (openmm)
somd (openmm)
somd (openmm)
somd (openmm)
System
System
System
System
Trajectory
Trajectory
Trajectory
Trajectory
anaylse_freenrg


Protein : top/crd
System
tleap
System
Ligand0 : gro87
System
Mol
antechamber
parmchk
tleap
Mol
Ligand1 : gro87
System
Mol
Mol
RDKit
smiles
Mol
tleap solvate
tleap solvate
System
System
gmx mdrun
System
Trajectory
System
Trajectory
System
pmemd.cuda
pmemd.cuda
pymbar
relative dG
pymbar
gmx mdrun
System
gmx mdrun
System
Trajectory
System
Trajectory
System
Trajectory
System
Trajectory
System
pmemd.cuda
pmemd.cuda
pmemd.cuda
pmemd.cuda

```python
import BioSimSpace as BSS

import re
import sys

# Read the list of ligands.
ligands = []
with open("ligands.txt", "r") as file:
    for line in file:
        ligands.append(line.rstrip())

# Get the ligand index.
idx = int(sys.argv[1])

# Extract the ligand name.
lig_name = re.search("(CatS_\d+).pdb", ligands[idx]).groups()[0]

# Create the prefix of the output files.
output = "parameterised/" + lig_name

# Load the ligand.
lig = BSS.IO.readMolecules(ligands[idx]).getMolecules()[0]

# Parameterise the ligand with GAFF2.
lig = BSS.Parameters.gaff2(lig).getMolecule()

# Save to AMBER format.
BSS.IO.saveMolecules(output, lig, ["rst7", "prm7"])
```

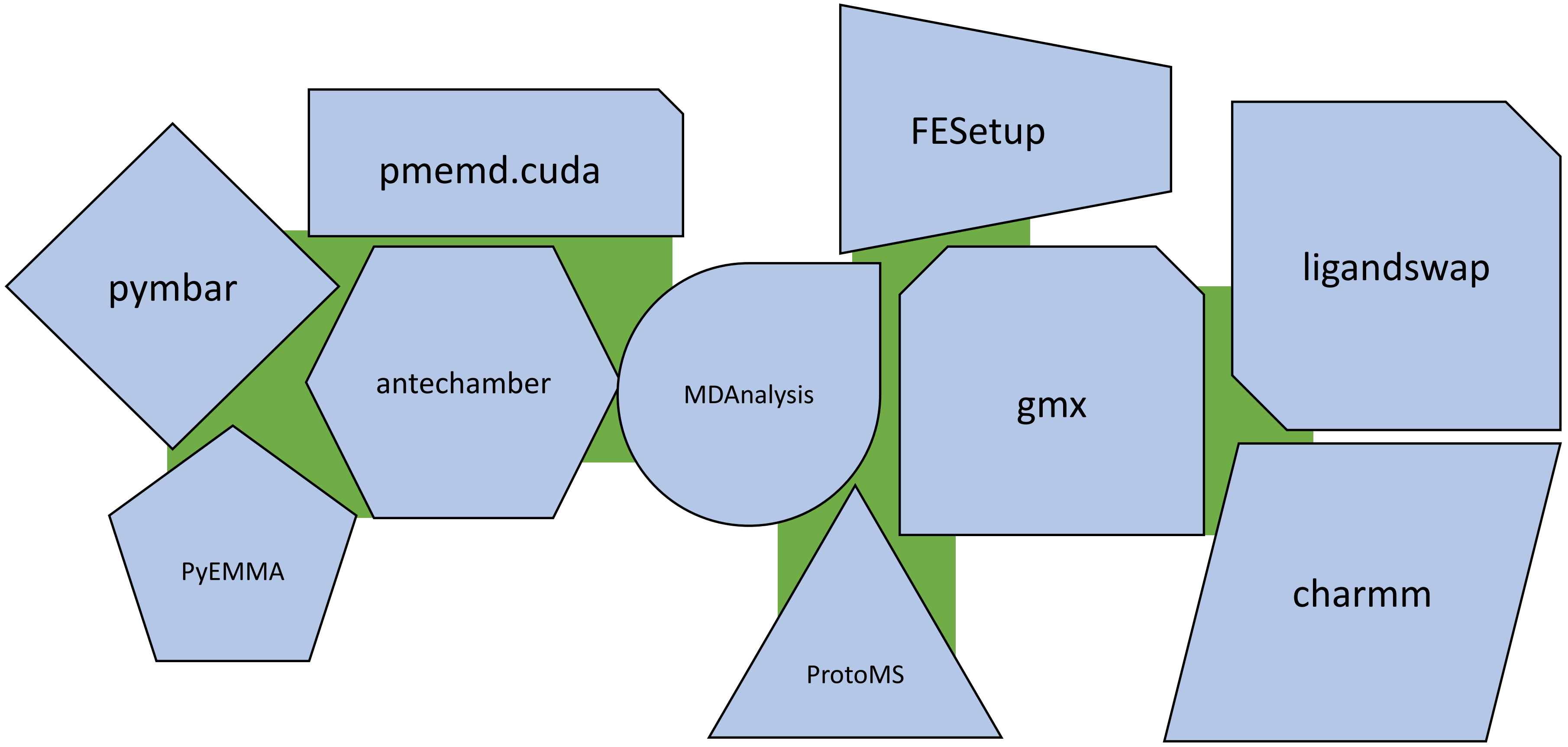


```python
# Extract the directory for the job.
try:
    job_dir = os.getenv("JOB_DIR")
except:
    job_dir = None

# No job directory set, use the current directory.
if job_dir is None:
    job_dir = "."

# Extract the ligand numbers.
num0 = sys.argv[1]
num1 = sys.argv[2]

# Load the protein and crystal waters.
protein_water = BSS.IO.readMolecules("%s/protein/protein_water.pdb" % job_dir)

# Extract the waters.
waters = protein_water.getWaterMolecules()

# Parameterise the protein.
protein = BSS.Parameters.ff14SB(protein_water.getMolecules()[0]).getMolecule()

# Load the parameterised ligands.
lig0 = BSS.IO.readMolecules(BSS.IO.glob("%s/ligands_aligned/parametrised/CatS_%s.*"
lig1 = BSS.IO.readMolecules(BSS.IO.glob("%s/ligands_aligned/parametrised/CatS_%s.*"

# If a mapping file exists, then load the mapping. Otherwise, use BioSimSpace
# to create the mapping.
mapping = {}

# Forward mapping.
if os.path.isfile("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num0, num
    with open("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num0, num1),
        for line in file:
            pair = line.strip().split()
            mapping[AtomIdx(int(pair[0]))] = AtomIdx(int(pair[1]))

# Reverse mapping.
elif os.path.isfile("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num1, r
    with open("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num1, num0),
        for line in file:
            pair = line.strip().split()
            # Invert the indices.
            mapping[AtomIdx(int(pair[1]))] = AtomIdx(int(pair[0]))

# No mapping, generate it ourselves.
else:
    # Find the best mapping of atoms between the ligands.
    mapping = BSS.Align.matchAtoms(lig0, lig1)

# Align lig0 to lig1 based on the mapping.
lig0 = BSS.Align.rmsdAlign(lig0, lig1, mapping)

# Merge the two ligands based on the mapping.
merged = BSS.Align.merge(lig0, lig1, mapping)

# Create the composite system.
system = merged + protein + waters

# Solvate in a 60 angstrom box of TIP3P water.
solvated = BSS.Solvent.tip3p(molecule=system, box=3*[60*BSS.Units.Length.angstrom])

# Create the free energy protocol.
protocol = BSS.Protocol.FreeEnergy(runtime=4*BSS.Units.Time.nanosecond, num_lam=17)

# Initialise the binding free energy object.
freenrg = BSS.FreeEnergy.Binding(solvated, protocol, work_dir="CatS_%s_%s" % (num0,

# Run the simulation.
freenrg.run()
```

We want to share the above scripts, together with the record of what was run

https://github.com/michellab/D3R2018/blob/master/CatS/BSS/binding_freenrg.py

BioSimSpace

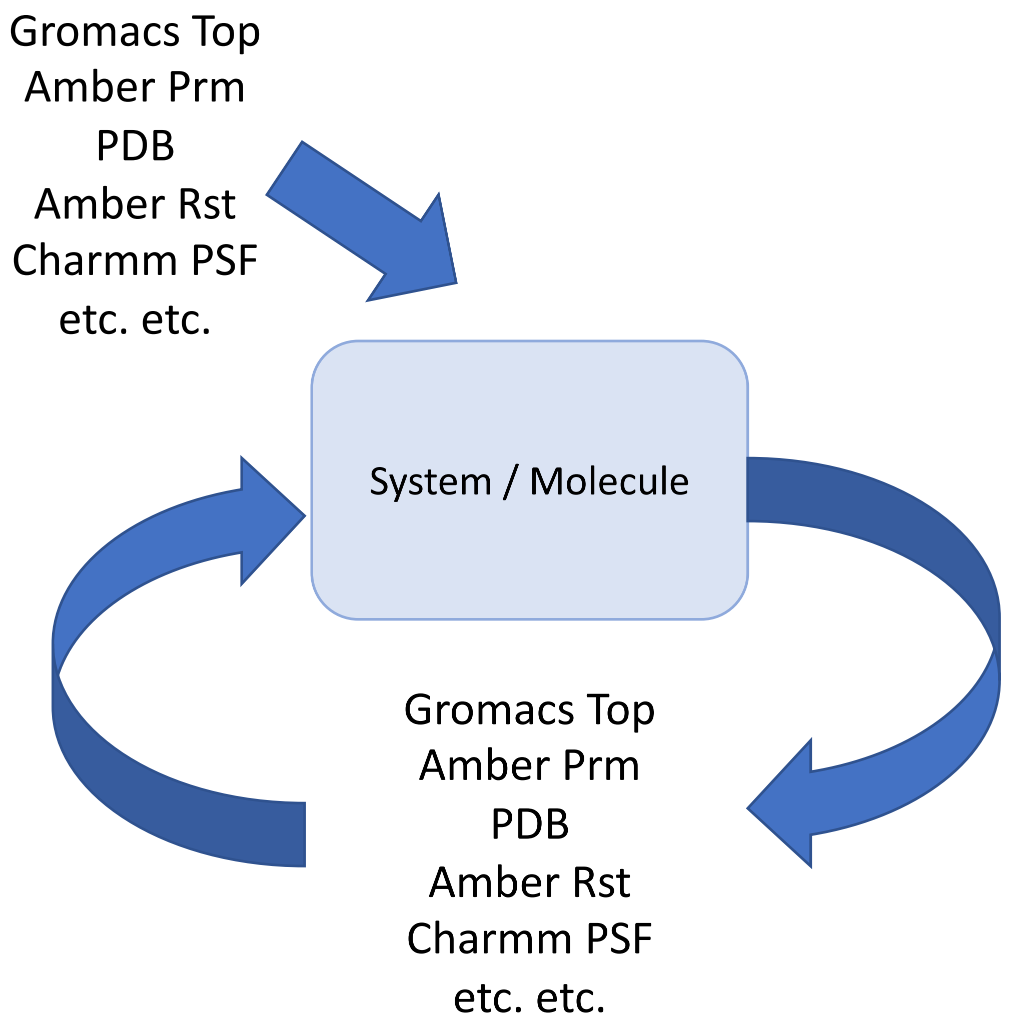


- Work with the existing formats and software we have
- Make it easier for this software to plug together
- Make it easier to translate one format into another

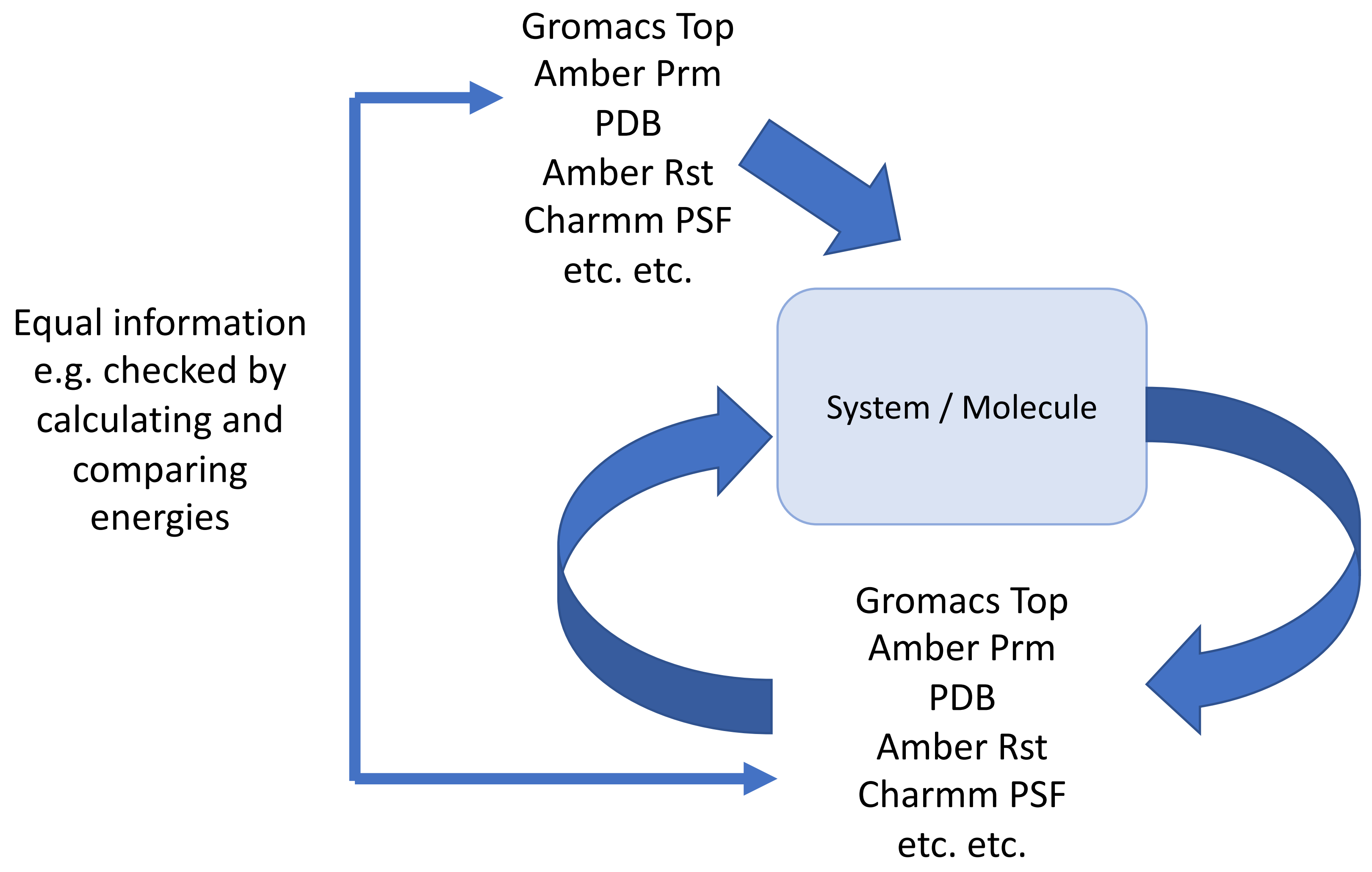
Make it easier to write the “shims”

7 Design Principles of BioSimSpace

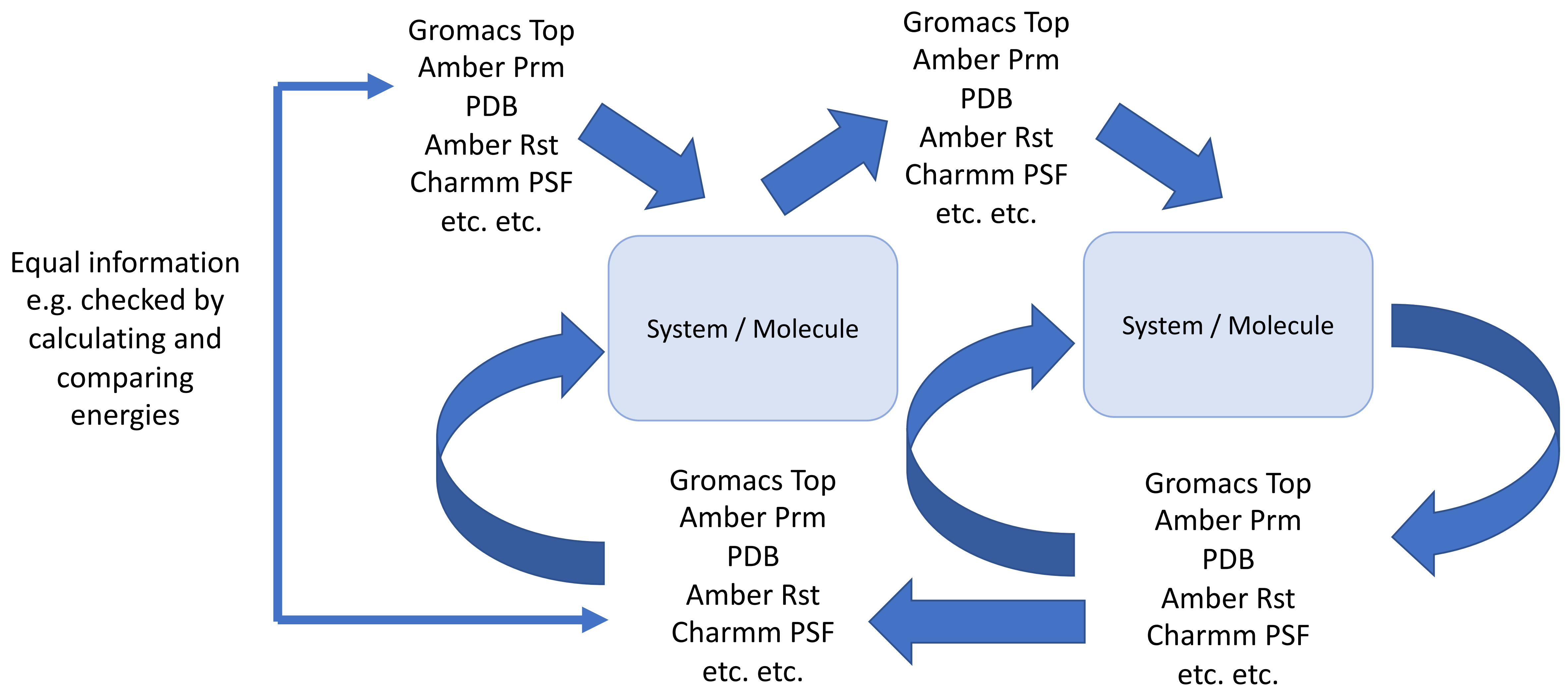
Design Principle 1: Read == Write



Design Principle 2: Information is preserved

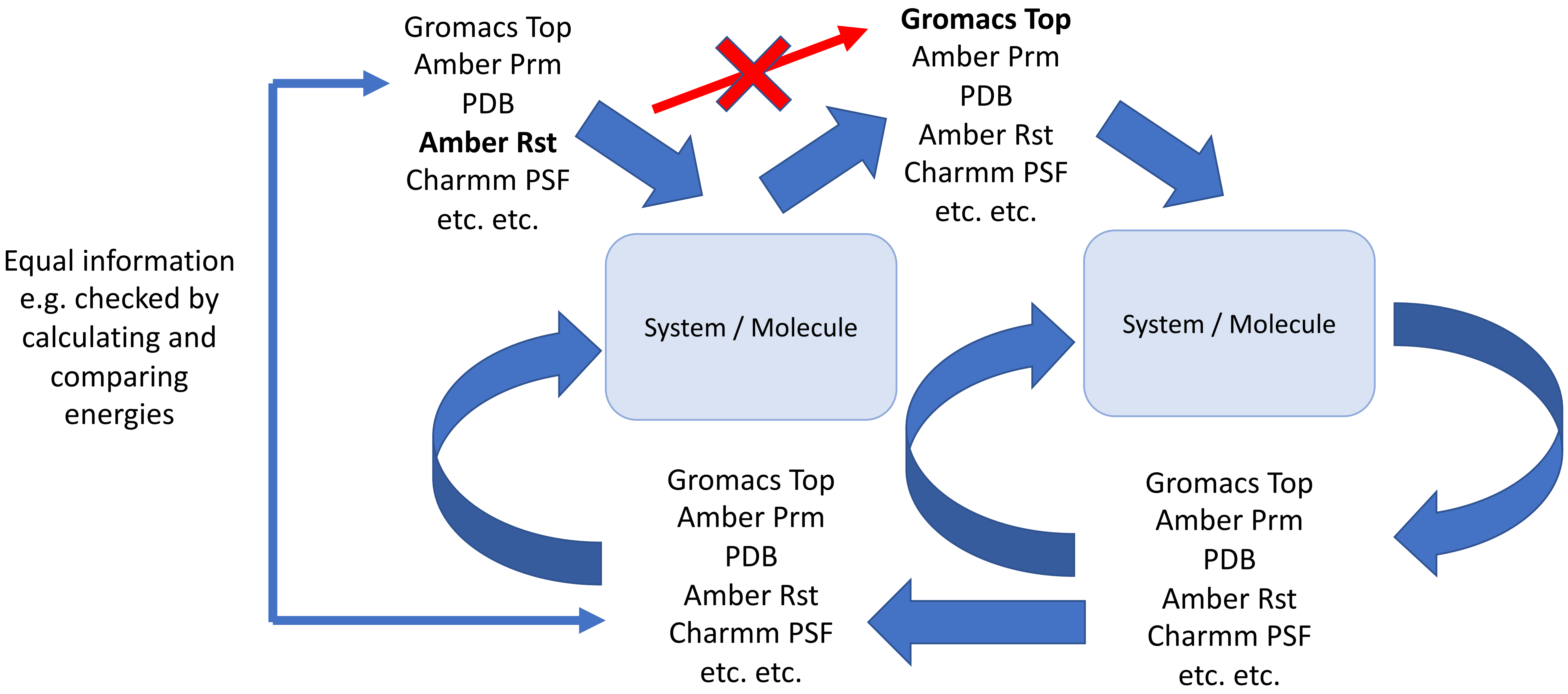


Design Principle 2: Information is preserved



Design Principle 3: Don’t be too clever!

(don’t guess missing info)



Hang on a minute - What is ”Molecule”?

Molecule

Collection of Property-derived objects organized into a key/value dictionary

“charge0” => AtomCharges
”LJ” => AtomLJs
“element” => AtomElements
“mass” => AtomMasses
“connectivity” => Connectivity
“bond”=> TwoAtomFunctions
“angle” => ThreeAtomFunctions
“dihedral” => FourAtomFunctions

Hang on a minute - What is ”Molecule”?

Arbitrary key names (and as many as you want and as many property types as you want!)

Molecule

Collection of Property-derived objects organized into a key/value dictionary

“charge0” => AtomCharges
“charge1” => AtomCharges
“fluffy_cat” => AtomLJs
“ELEMENT” => AtomElements
“silly name” => AtomMasses
“bonding” => Connectivity
“2”=> TwoAtomFunctions
“3” => ThreeAtomFunctions
“4” => FourAtomFunctions

Hang on a minute - What is ”Molecule”?

Arbitrary key names (and as many as you want and as many property types as you want!)

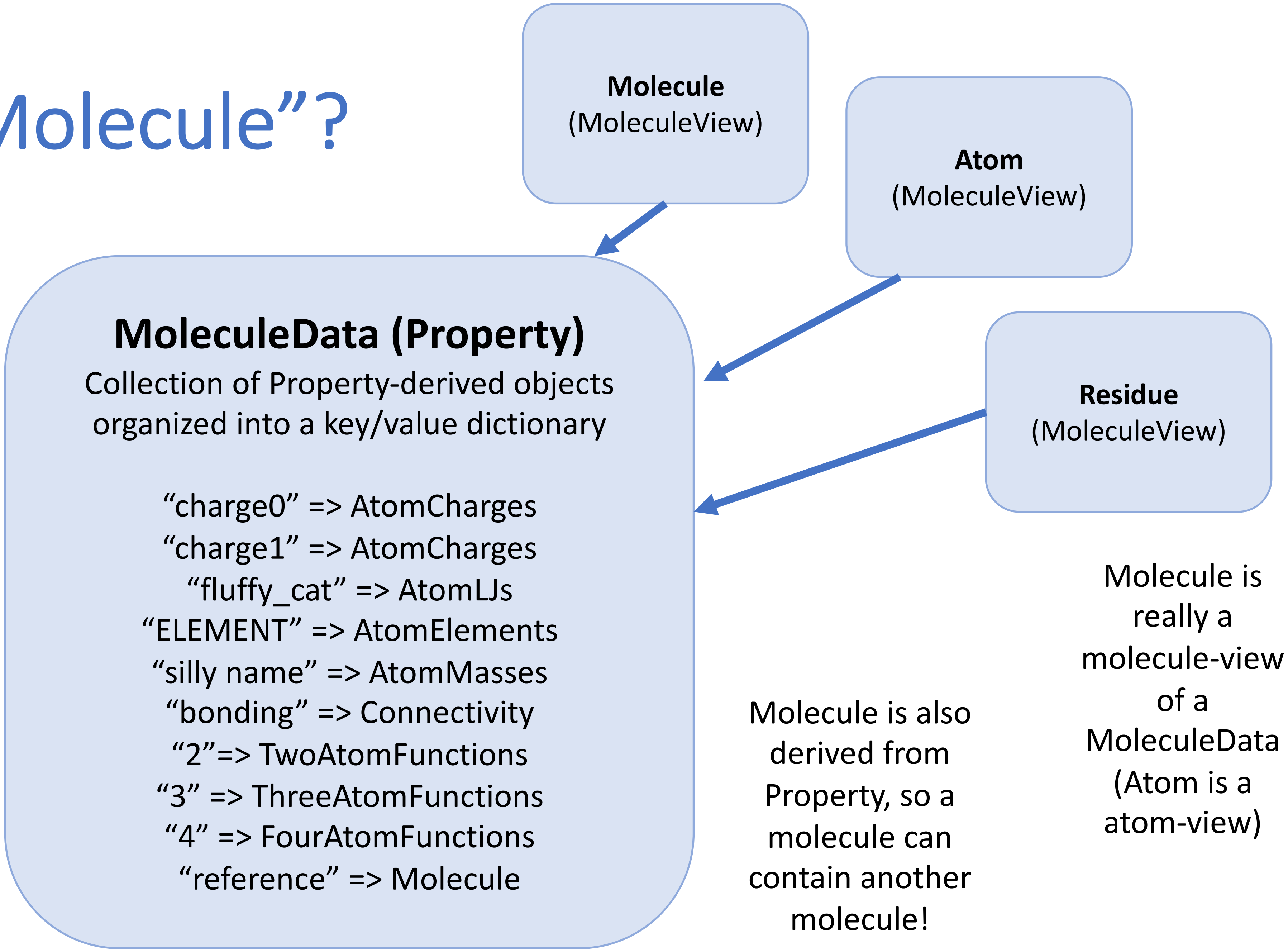
Molecule(Property)

Collection of Property-derived objects organized into a key/value dictionary

“charge0” => AtomCharges
“charge1” => AtomCharges
“fluffy_cat” => AtomLJs
“ELEMENT” => AtomElements
“silly name” => AtomMasses
“bonding” => Connectivity
“2”=> TwoAtomFunctions
“3” => ThreeAtomFunctions
“4” => FourAtomFunctions
“reference” => Molecule

Molecule is also derived from Property, so a molecule can contain another molecule!

What is ”Molecule”?



MoleculeData (Property)

Collection of Property-derived objects organized into a key/value dictionary

“charge0” => AtomCharges
“charge1” => AtomCharges
“fluffy_cat” => AtomLJs
“ELEMENT” => AtomElements
“silly name” => AtomMasses
“bonding” => Connectivity
“2”=> TwoAtomFunctions
“3” => ThreeAtomFunctions
“4” => FourAtomFunctions
“reference” => Molecule

Arbitrary key names (and as many as you want and as many property types as you want!)

Molecule is also derived from Property, so a molecule can contain another molecule!

Molecule is really a molecule-view of a MoleculeData (Atom is a atom-view)

And, What is ”System”?

Arbitrary key names (and as many as you want and as many property types as you want!)

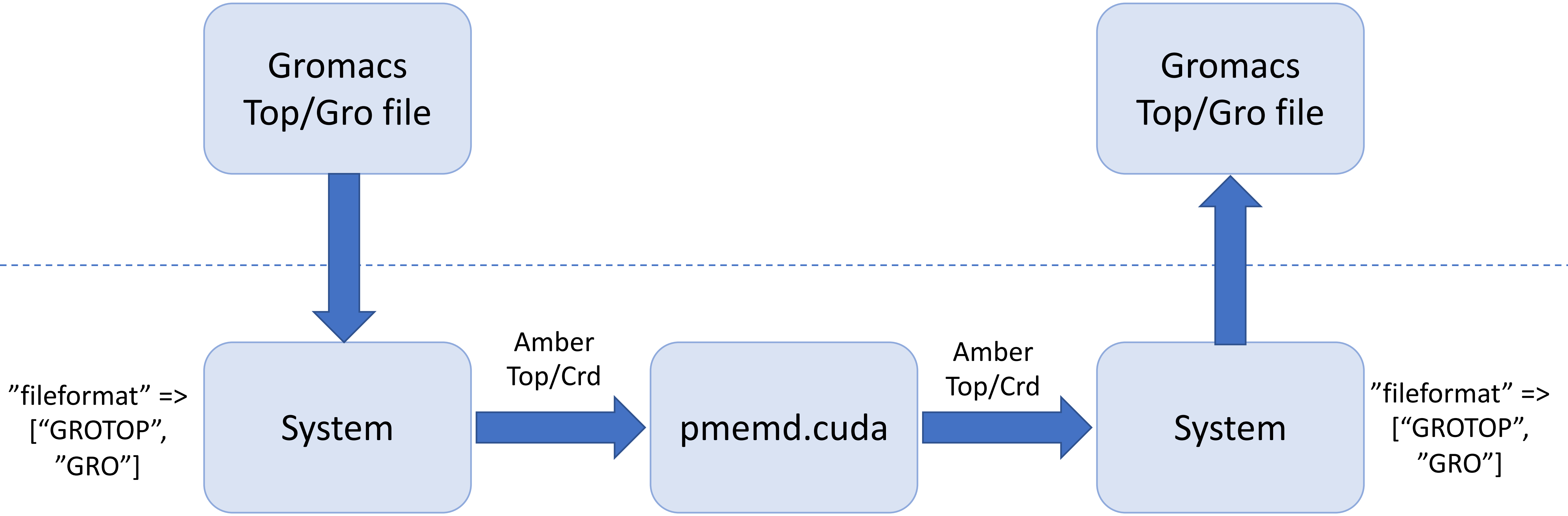
System(Property)
Collection of MoleculeGroups and Properties that describe the system

“space” => PeriodicBox
“time” => Time(5*nanosecond)
“all” => MoleculeGroup(molecules)
“protein” => MoleculeGroup(molecules)
“ligand” => MoleculeGroup(molecules)
“reference” => System
“free energy” => FreeEnergyMonitor

System groups together collections of molecules (really molecule views) into MoleculeGroups, and packages these with its own key/value dictionary of arbitrary properties.

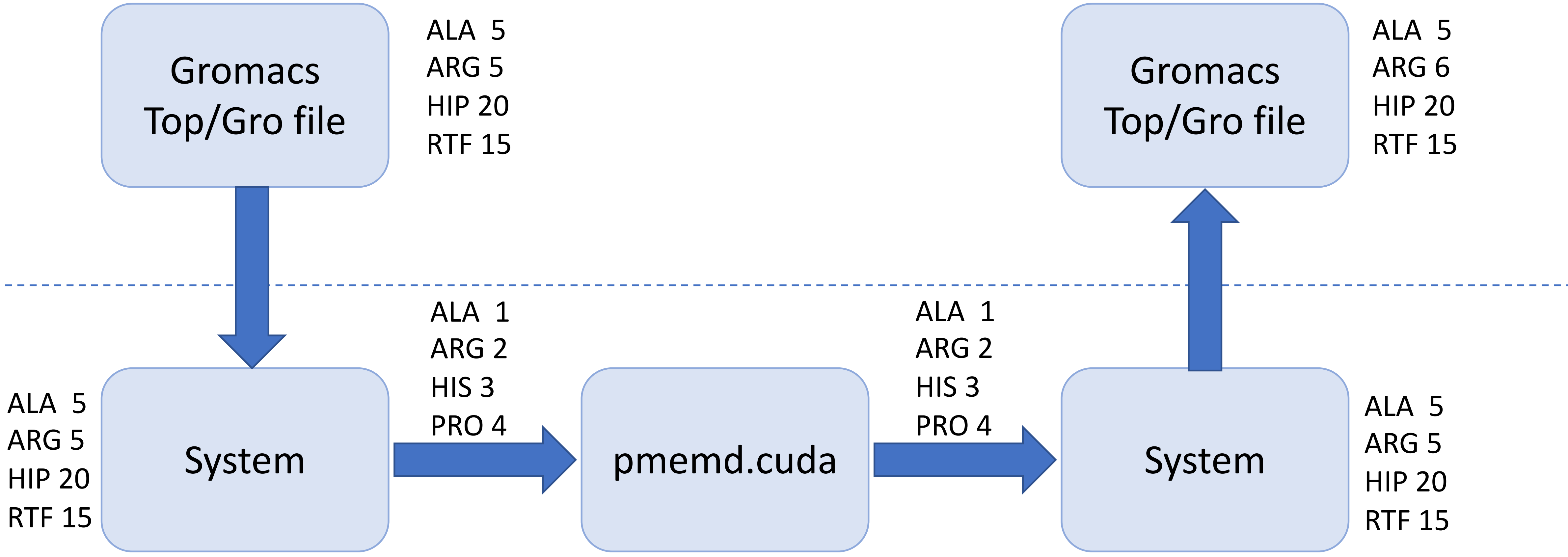
System is also a Property, so systems can contain other systems, molecules can contain systems – it can all be very inception!

Design Principle 4: Don’t Change Anything!



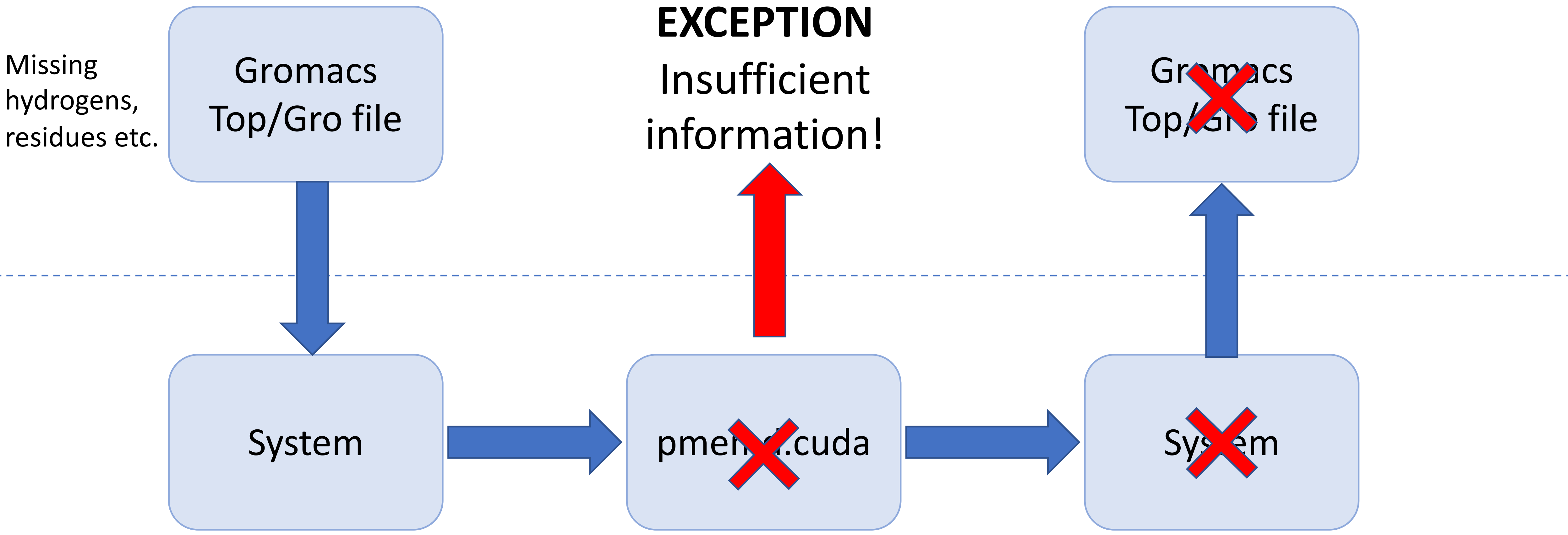
Write output in the same format as the user-supplied input

Design Principle 4: Don't Change Anything!



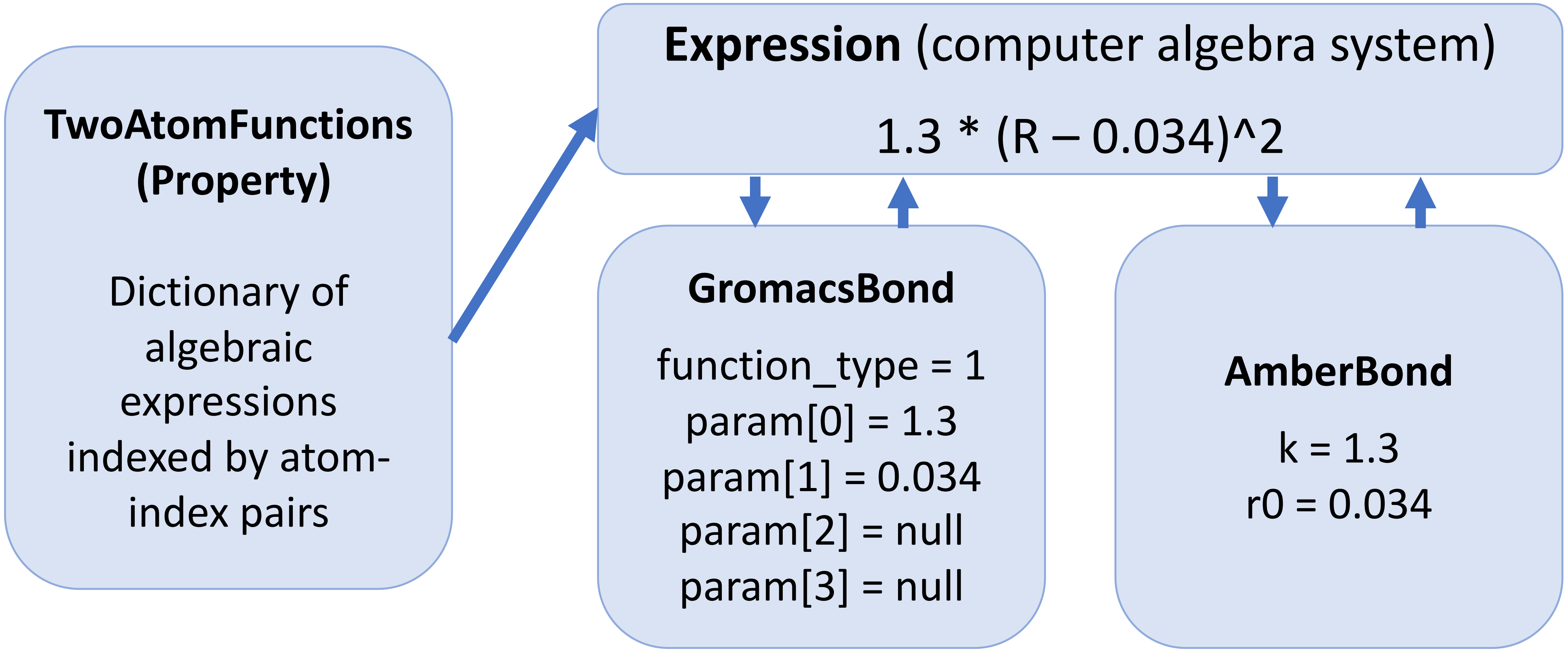
Preserve user-supplied identifiers and atom/residue ordering!

Design Principle 4: Don't Change Anything!



Don't add missing information unless it is unambiguous
(this is another example of DP3: Don't be too clever!)

Design Principle 5: Store General / Write Specific



Data is stored in a generic format and only converted to format-specific formats when writing. Exceptions raised if data cannot be converted

Design Principle 6: Units are important!

All data has attached units, using a complete units library, e.g.

```
temperature = 298 * kelvin
bond_k = 3.5 * kcal / (mol * angstrom * angstrom)
timestep = 2 * femtosecond
```

(needed to mix gromacs - SI – with amber and others – AKMA)

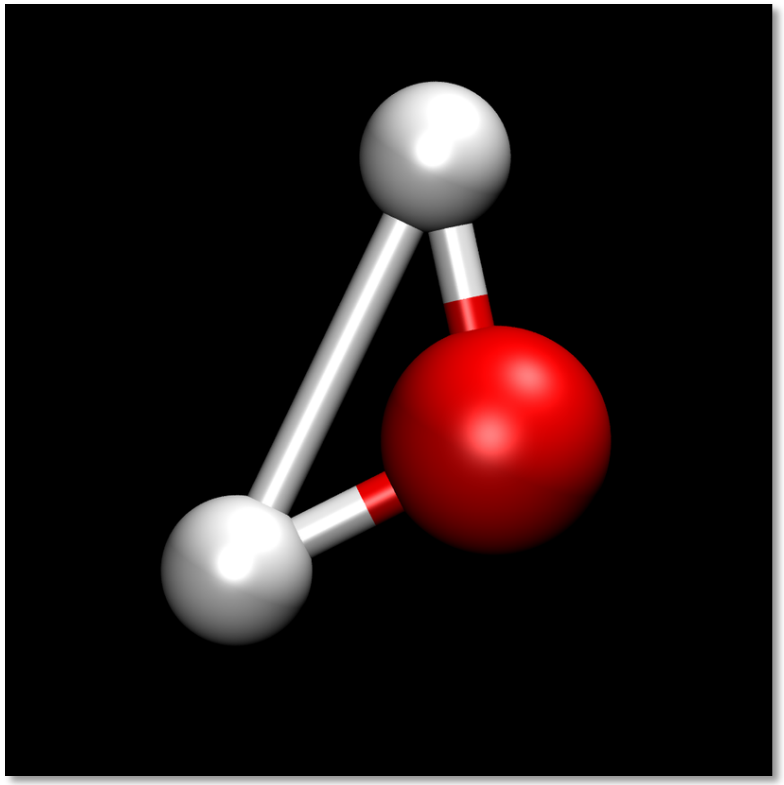
Design Principle 7: Don’t just assume – ask!

- Missing formal charge => ask the user
- Whatever is input is complete – don’t assume that residues or hydrogens are missing
- Don’t do things behind the user’s back because you assume they have given you the wrong thing.
- Raise an exception if you can’t deal with what you have got or there is insufficient information.
- This is related to DP3: “Don’t be too clever!” and DP4: “Don’t change anything!”

Challenges

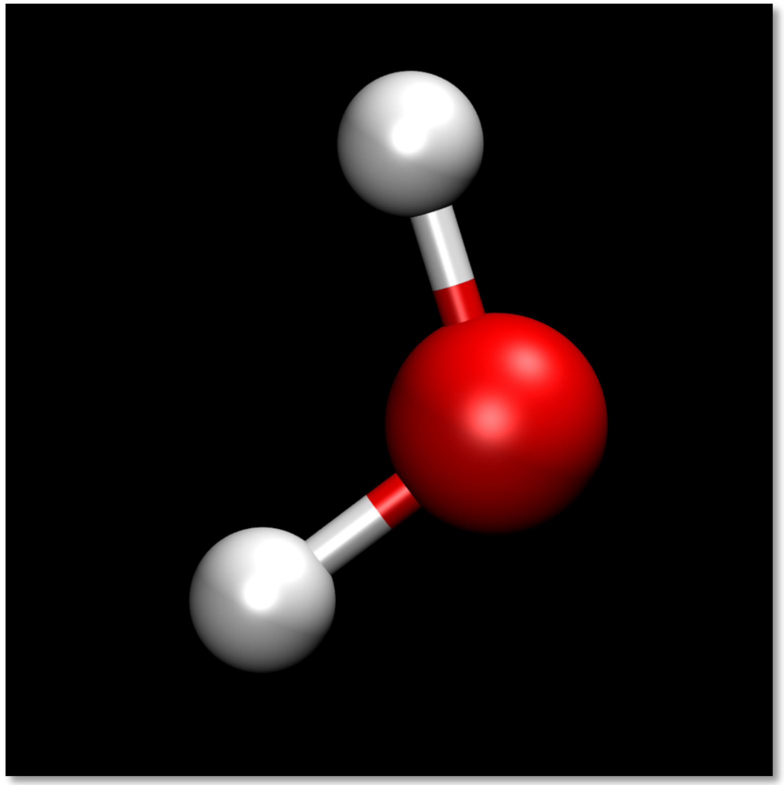
- **Different programs choose different places to store or represent information…**

Rigid AMBER water



Explicit H-H bond in topology file

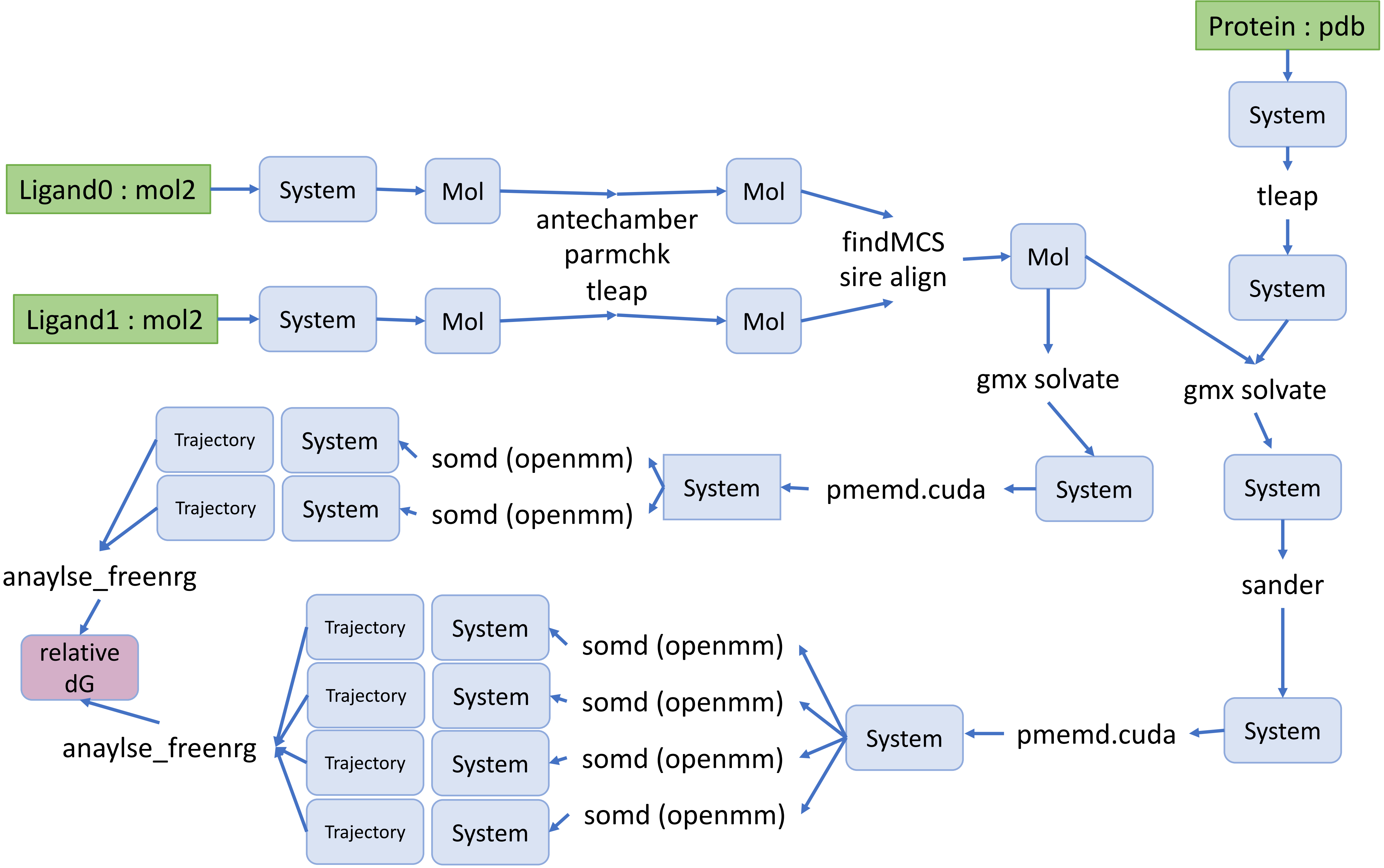
Rigid GROMACS water



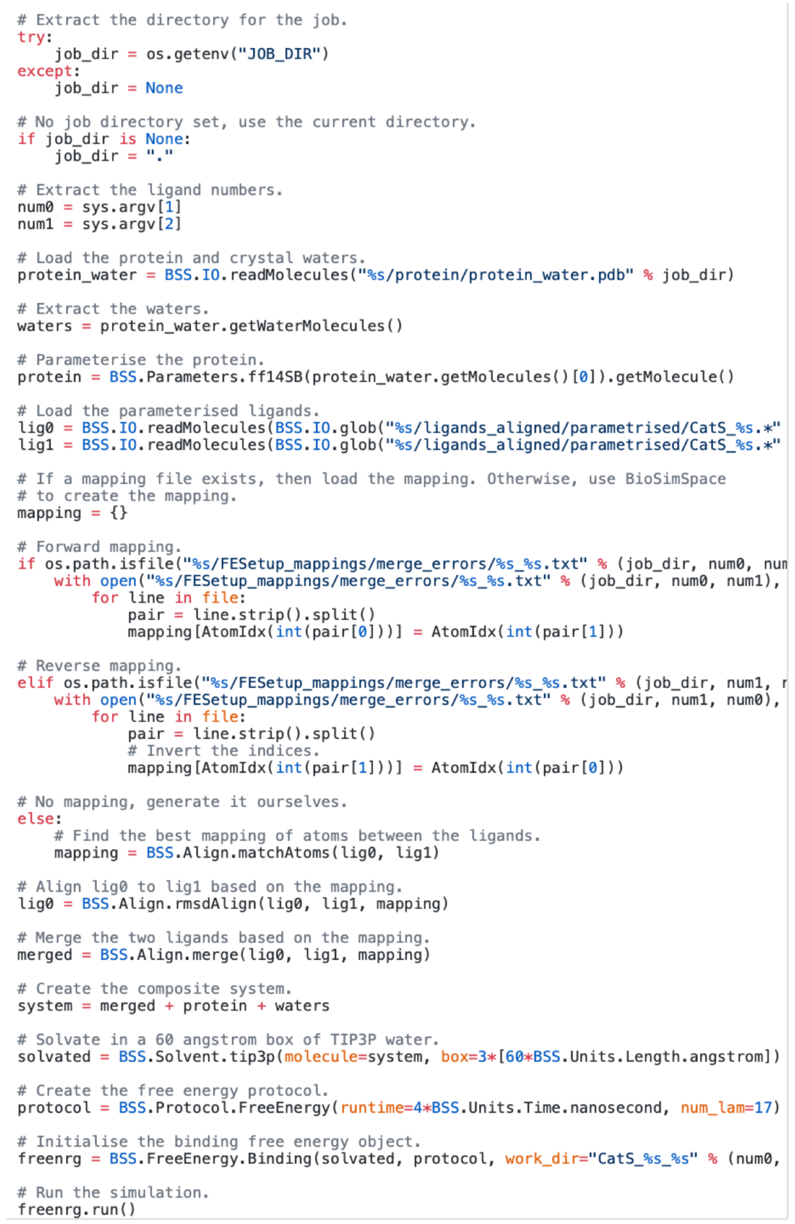
Controlled via configuration option (constraint-algorithm=shake), or #infdef FLEXIBLE block in topology file, i.e. “settles”

Challenges

- **Different programs choose different places to store or represent information…**
    - (where are rigid bonds or parameters for shake defined?)
- **Underlying tools are not sufficiently modular**
    - (tleap must parameterize + solvate, when I would like solvate only!)
- **Tools are not robust**
    - pmemd crashes when minimizing systems that work perfectly well with sander, somd or gromacs…
- **Not a perfect match of algorithms in packages**
    - different implementations of thermostats, shake algorithms, integrators etc.


Protein : pdb
System
tleap
System
Ligand0 : mol2
System
Mol
antechamber
parmchk
tleap
Mol
Ligand1 : mol2
System
Mol
Mol
findMCS
sire align
Mol
gmx solvate
gmx solvate
System
System
sander
System
pmemd.cuda
System
somd (openmm)
somd (openmm)
System
System
Trajectory
Trajectory
anaylse_freenrg
relative dG
pmemd.cuda
System
somd (openmm)
somd (openmm)
somd (openmm)
somd (openmm)
System
System
System
System
Trajectory
Trajectory
Trajectory
Trajectory
anaylse_freenrg

```python
import BioSimSpace as BSS

import re
import sys

# Read the list of ligands.
ligands = []
with open("ligands.txt", "r") as file:
    for line in file:
        ligands.append(line.rstrip())

# Get the ligand index.
idx = int(sys.argv[1])

# Extract the ligand name.
lig_name = re.search("(CatS_\d+).pdb", ligands[idx]).groups()[0]

# Create the prefix of the output files.
output = "parameterised/" + lig_name

# Load the ligand.
lig = BSS.IO.readMolecules(ligands[idx]).getMolecules()[0]

# Parameterise the ligand with GAFF2.
lig = BSS.Parameters.gaff2(lig).getMolecule()

# Save to AMBER format.
BSS.IO.saveMolecules(output, lig, ["rst7", "prm7"])
```



```python
# Extract the directory for the job.
try:
    job_dir = os.getenv("JOB_DIR")
except:
    job_dir = None

# No job directory set, use the current directory.
if job_dir is None:
    job_dir = "."

# Extract the ligand numbers.
num0 = sys.argv[1]
num1 = sys.argv[2]

# Load the protein and crystal waters.
protein_water = BSS.IO.readMolecules("%s/protein/protein_water.pdb" % job_dir)

# Extract the waters.
waters = protein_water.getWaterMolecules()

# Parameterise the protein.
protein = BSS.Parameters.ff14SB(protein_water.getMolecules()[0]).getMolecule()

# Load the parameterised ligands.
lig0 = BSS.IO.readMolecules(BSS.IO.glob("%s/ligands_aligned/parametrised/CatS_%s.*"
lig1 = BSS.IO.readMolecules(BSS.IO.glob("%s/ligands_aligned/parametrised/CatS_%s.*"

# If a mapping file exists, then load the mapping. Otherwise, use BioSimSpace
# to create the mapping.
mapping = {}

# Forward mapping.
if os.path.isfile("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num0, nun
    with open("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num0, num1),
        for line in file:
            pair = line.strip().split()
            mapping[AtomIdx(int(pair[0]))] = AtomIdx(int(pair[1]))

# Reverse mapping.
elif os.path.isfile("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num1, r
    with open("%s/FESetup_mappings/merge_errors/%s_%s.txt" % (job_dir, num1, num0),
        for line in file:
            pair = line.strip().split()
            # Invert the indices.
            mapping[AtomIdx(int(pair[1]))] = AtomIdx(int(pair[0]))

# No mapping, generate it ourselves.
else:
    # Find the best mapping of atoms between the ligands.
    mapping = BSS.Align.matchAtoms(lig0, lig1)

# Align lig0 to lig1 based on the mapping.
lig0 = BSS.Align.rmsdAlign(lig0, lig1, mapping)

# Merge the two ligands based on the mapping.
merged = BSS.Align.merge(lig0, lig1, mapping)

# Create the composite system.
system = merged + protein + waters

# Solvate in a 60 angstrom box of TIP3P water.
solvated = BSS.Solvent.tip3p(molecule=system, box=3*[60*BSS.Units.Length.angstrom])

# Create the free energy protocol.
protocol = BSS.Protocol.FreeEnergy(runtime=4*BSS.Units.Time.nanosecond, num_lam=17)

# Initialise the binding free energy object.
freenrg = BSS.FreeEnergy.Binding(solvated, protocol, work_dir="CatS_%s_%s" % (num0,

# Run the simulation.
freenrg.run()
```

We will be sharing the above scripts, together with the record of what was run

https://github.com/michellab/D3R2018/blob/master/CatS/BSS/binding_freenrg.py

Acknowledgements

BioSimSpace Research Team

Lester Hedges, Antonia Mey, Julien Michel, Adrian Mulholland, Charlie Laughton, Francesco Gervasio

EPSRC for funding (EP/N018591/1 and EP/P022138/1)

Download this talk from https://chryswoods.com/talks

Follow BioSimSpace development at https://github.com/michellab/BioSimSpace

Follow our D3R challenge simulations at https://github.com/michellab/D3R2018