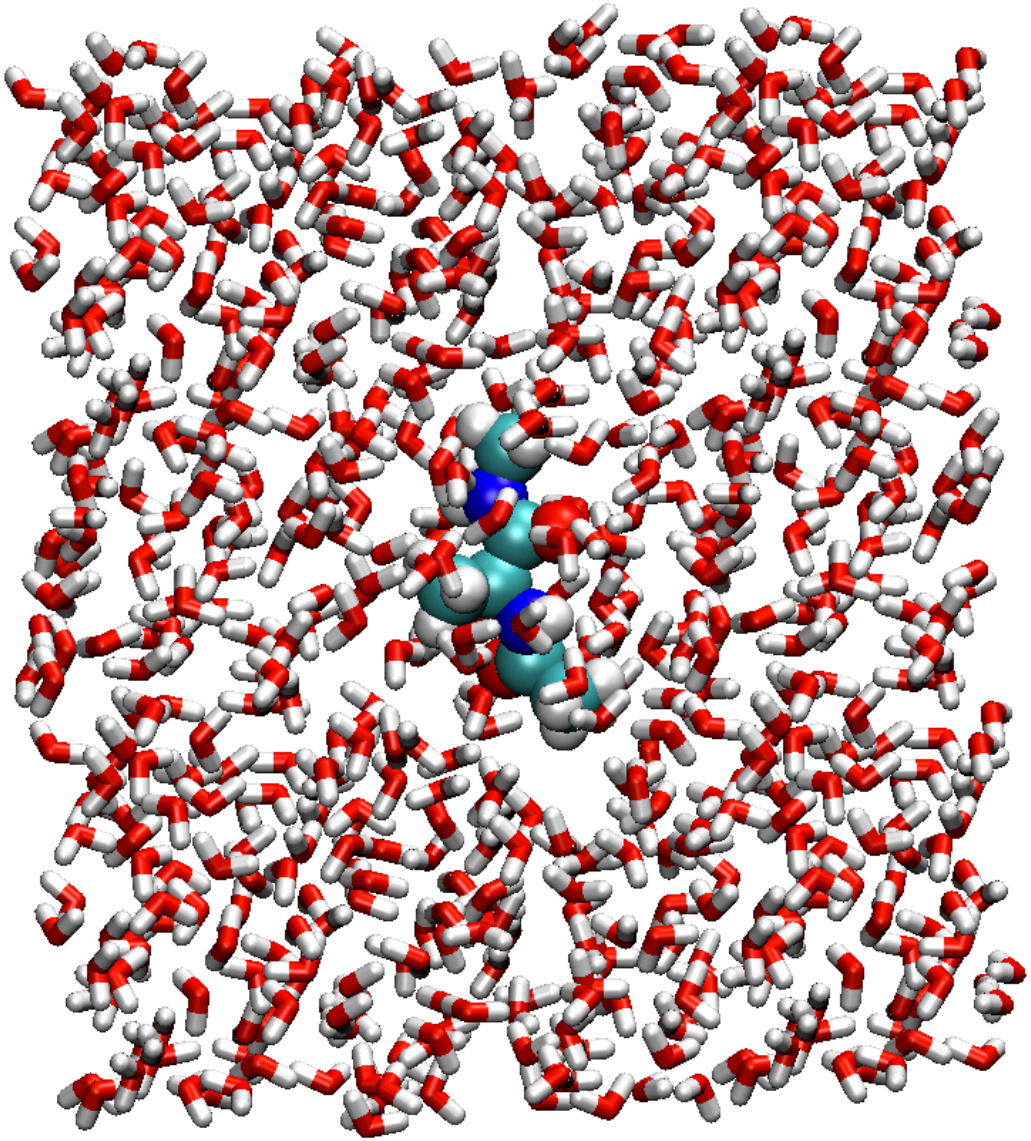
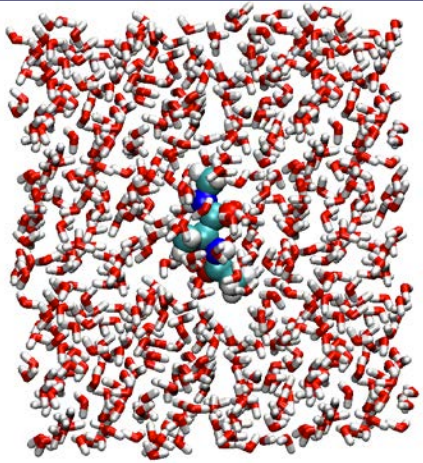
Exercise 1:

Let's ease into using OpenMM for MD simulations
The task: Simulate an alanine dipeptide in water



simbios

Remember: We can interface with OpenMM through python scripts



```
from simtk.openmm.app import *
from simtk.openmm import *
from simtk.unit import *
from sys import stdout

# Reading structure and force field files
pdb = PDBFile('input_exercise1.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')

# Creating System
system = forcefield.createSystem(pdb.topology, nonbondedMethod=PME,
     nonbondedCutoff=1.0*nanometer, constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)

# Creating simulation context
simulation = Simulation(pdb.topology, system, integrator)
simulation.context.setPositions(pdb.positions)

# Minimizing System
simulation.minimizeEnergy(maxIterations=25)

# Adding Reporters
simulation.reporters.append(PDBReporter('output_exercise1.pdb', 5))
simulation.reporters.append(StateDataReporter(stdout, 100, step=True,
potentialEnergy=True, temperature=True))

# Running simulation
simulation.step(1000)
```

Let's give it a go …

1. In your terminal window, go to the exercise directory.
2. Try running the exercise by typing: python exercise1.py

If you get "Import error" or "Library not found", try this:

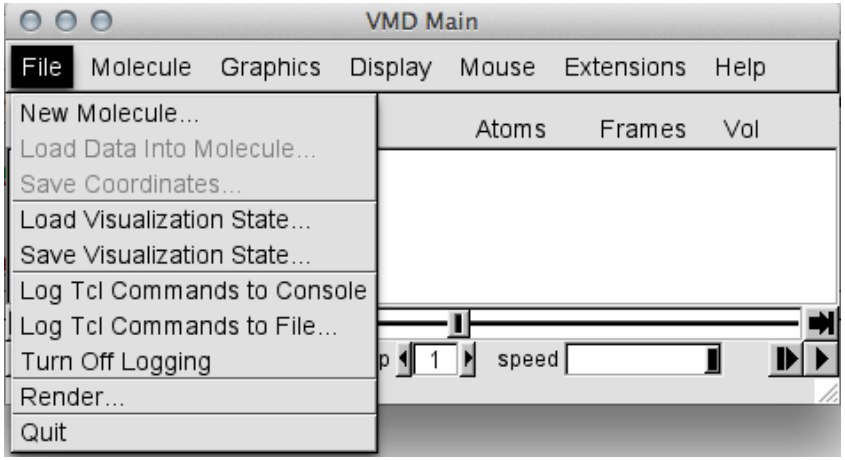
Mac OS X:
```
export DYLD_LIBRARY_PATH=/usr/local/openmm/lib:/usr/local/cuda/lib
```
Linux:
```
export LD_LIBRARY_PATH=/usr/local/openmm/lib:/usr/local/cuda/lib
```
Windows:
Refer to page 16 of the OpenMM Application Guide.

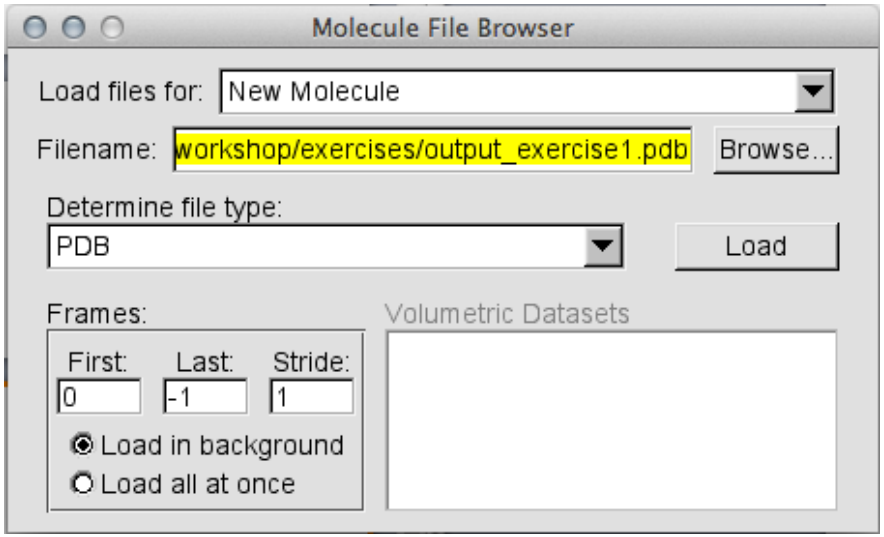
Your output will look something like this:

```
Creating System
Using Platform: OpenCL
Minimizing Energy
Adding Reporters to report Potential Energy and Temperature every 100 steps
Running Simulation for 1000 steps
Running step: 0
#"Step","Potential Energy (kJ/mole)","Temperature (K)"
100,-26643.6088239,175.727733927
Running step: 100
200,-26218.1442337,194.691558193
.
.
1000,-24957.7316677,283.859340959
Finished Simulation.
```

Let's now look at the simulation output

1. If your simulation ran on the Amazon cloud, download the output (psftp on Win, scp on Mac/Linux)
2. Open VMD





2. In 'VMD Main': Select File and New Molecule...
3. The 'Molecule File Browser' appears.
4. Click 'Browse' and select output_exercise1.pdb.
5. Click 'Load'.

Force fields available in OpenMM

Set with: `forcefield = ForceField('amber99sb.xml', 'tip3p.xml')`

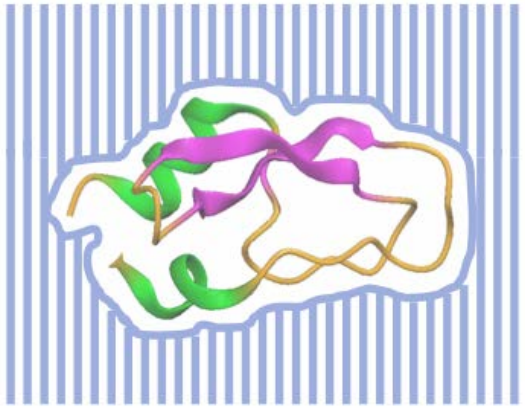
File	Force Field
amber96.xml	AMBER96[1]
amber99sb.xml	AMBER99[2] with modified backbone torsions[3]
amber99sbildn.xml	AMBER99SB plus improved side chain torsions[4]
amber99sbnmr.xml	AMBER99SB with modifications to fit NMR data[5]
amber03.xml	AMBER03[6]
amber10.xml	AMBER10
amoeba2009.xml	AMOEBA[7] (AMOEBA includes its own water model)

New/currently unavailable and custom force fields can be added.

Check out talk by Lee-Ping tomorrow, March 27, 9:30 am!

Solvent models

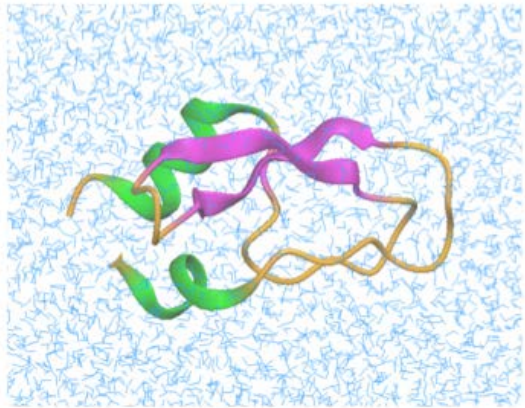
Set with: `forcefield = ForceField('amber99sb.xml', 'tip3p.xml')`



$$H = H_{prot} + \Delta G_{solv}$$

Implicit water model

-Fast (fewer atoms to track and less friction)
-Gets bulk properties 'right'



$$H = H_{prot} + H_{prot-wat} + H_{wat}$$

Explicit water model

-Slower (solvent atoms make up most of the system)
-Gets atomistic properties 'right'

Explicit solvent models …

… parameterized to compensate for their simplified description of reality.

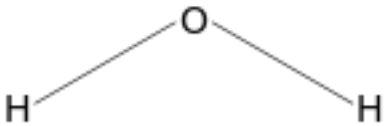
Most models have incomplete physics:

- Fixed point charges (no electronic polarization)
- Classical mechanics (no isotope effects)
- Fixed bond topology (no chemistry)

However, much can be recovered through parameterization:

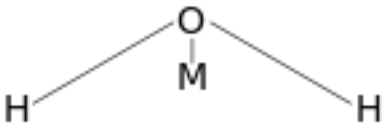
- Increase the partial charges, tune vdW parameters, etc.
- In many cases, force fields exceed the accuracy of quantum methods!

TIP3P



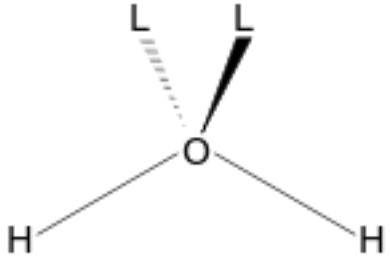
3-site

TIP4P EW

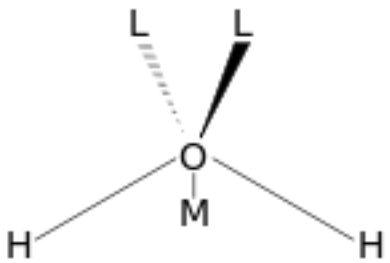


4-site

TIP5P



5-site



6-site

Adapted from Lee-Ping Wang

Implicit solvent models

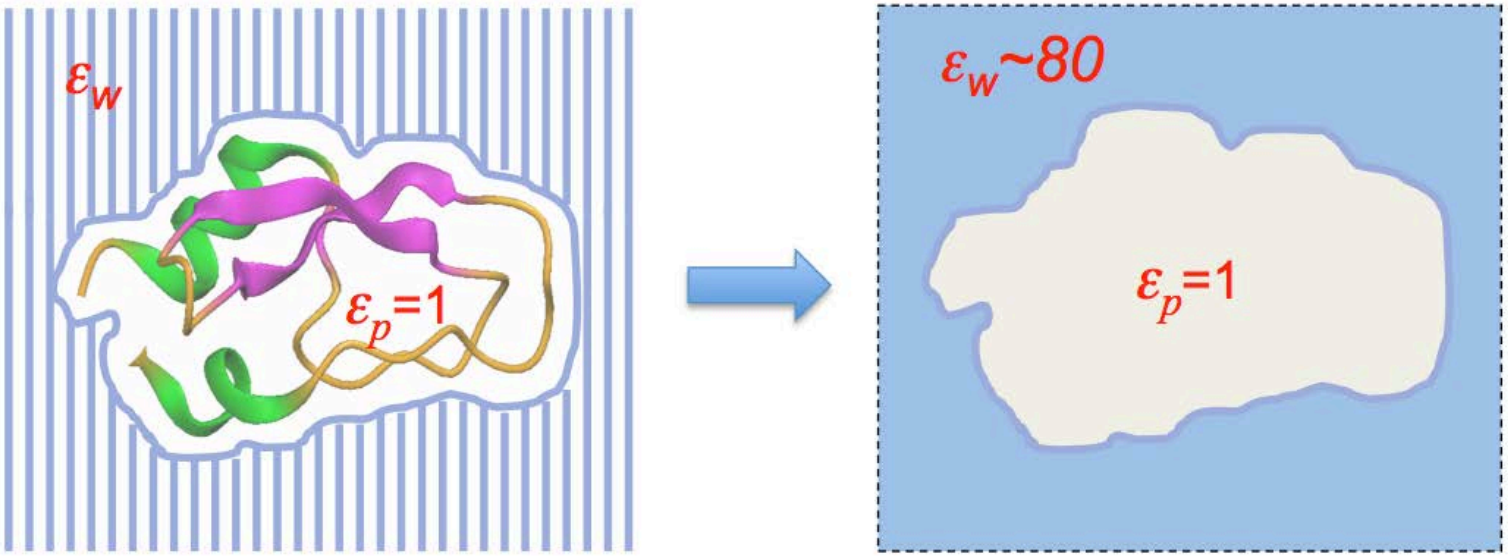
Water dynamics is typically much faster than that of protein conformational diffusion. Thus, water can be described as a continuum medium.

$$\Delta G_{\text{solv}} = \Delta G_{\text{elec}} + \Delta G_{\text{np}} \qquad \Delta G_{\text{np}} = \gamma \cdot S$$

The electrostatic component is commonly computed using the generalized Born (GB) approximation:

$$\Delta G_{\text{elec}} \approx \frac{1}{2}\left(\frac{1}{\varepsilon_{\text{w}}} - \frac{1}{\varepsilon_{\text{p}}}\right)\sum_{ij} \frac{q_i q_j}{\sqrt{r_{ij}^2 + R_i^{\text{GB}} R_j^{\text{GB}} \exp\left(-r_{ij}^2 / 4 R_i^{\text{GB}} R_j^{\text{GB}}\right)}}$$

This is then often augmented with a hydrophobic solvent accessible surface area (SA) term, giving the GBSA model.



Solvent models available in OpenMM

File	Water Model
tip3p.xml	TIP3P water model[8]
tip4pew.xml	TIP4P-Ew water model[9]
tip5p.xml	TIP5P water model[10]
spce.xml	SPC/E water model[11]

File	Implicit Solvation Model
amber96_obc.xml	GBSA-OBC solvation model[12] for use with AMBER96 force field
amber99_obc.xml	GBSA-OBC solvation model for use with AMBER99 force fields
amber03_obc.xml	GBSA-OBC solvation model for use with AMBER03 force field
amber10_obc.xml	GBSA-OBC solvation model for use with AMBER10 force field
amoeba2009_gk.xml	Generalized Kirkwood solvation model[13] for use with AMOEBA force field

Integrators in OpenMM

```
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)
```

Simulation Temperature — Friction coefficient — Timestep

Other Integrators:

Constant energy simulations: VerletIntegrator(0.002*picoseconds)

Brownian Dynamics: BrownianIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)

Variable time step: VariableLangevinIntegrator(300*kelvin, 1/picosecond, 0.001)
VariableVerletIntegrator(0.001)

Temperature coupling: system.addForce(AndersenThermostat(300*kelvin, 1/picosecond))

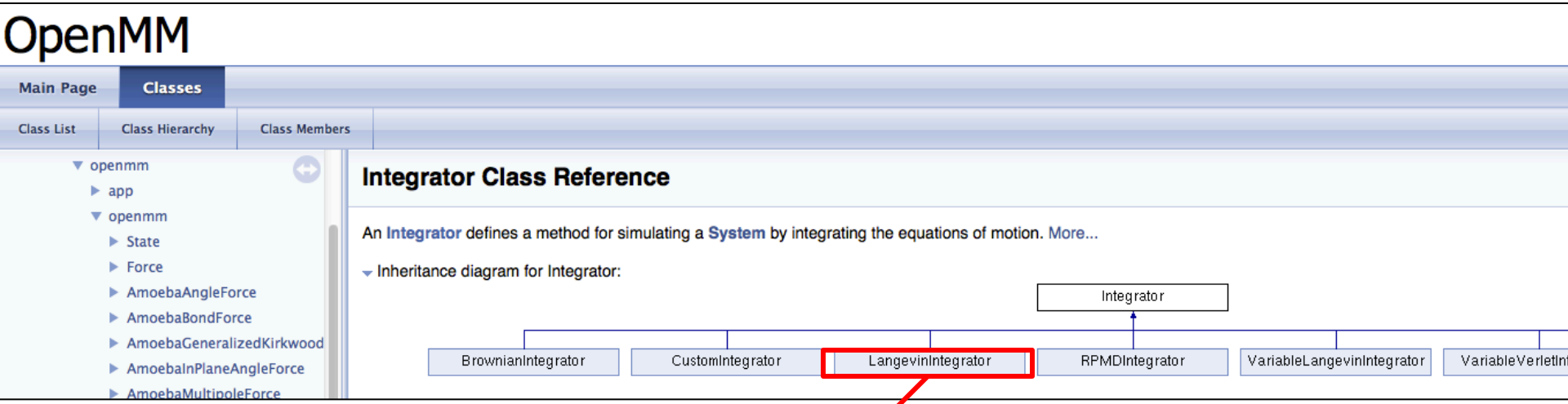
Pressure Coupling: system.addForce(MonteCarloBarostat(1*bar, 300*kelvin))

But why don't we find out for ourselves?

simbios

Learning about the Integrator Class from OpenMM

Documentation page: https://simtk.org/api_docs/openmm/api5_0/python



Constructor & Destructor Documentation

```
def __init__ (  self,
                args
              )
```

init(OpenMM::LangevinIntegrator self, double temperature, double frictionCoeff, double stepSize) -> **LangevinIntegrator** **init**(OpenMM::LangevinIntegrator self, **LangevinIntegrator** other) -> **LangevinIntegrator**

Create a **LangevinIntegrator**.

Parameters

temperature the temperature of the heat bath (in Kelvin)

frictionCoeff the friction coefficient which couples the system to the heat bath (in inverse picoseconds)

stepSize the step size with which to integrator the system (in picoseconds)

Reading values from an Integrator Object

OpenMM

Main Page | Classes

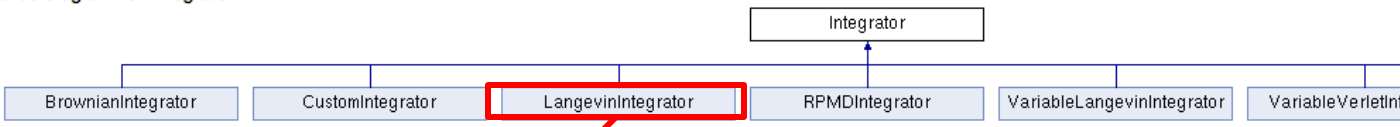
Class List | Class Hierarchy | Class Members

- openmm
  - app
  - openmm
    - State
    - Force
    - AmoebaAngleForce
    - AmoebaBondForce
    - AmoebaGeneralizedKirkwood
    - AmoebaInPlaneAngleForce
    - AmoebaMultipoleForce

Integrator Class Reference

An **Integrator** defines a method for simulating a **System** by integrating the equations of motion. More...

Inheritance diagram for Integrator:



Public Member Functions

def **getTemperature**
getTemperature(LangevinIntegrator self) -> double

def **setTemperature**
setTemperature(LangevinIntegrator self, double temp)

def **getFriction**
getFriction(LangevinIntegrator self) -> double

def **setFriction**
setFriction(LangevinIntegrator self, double coeff)

def **getRandomNumberSeed**
getRandomNumberSeed(LangevinIntegrator self) -> int

def **setRandomNumberSeed**
setRandomNumberSeed(LangevinIntegrator self, int seed)

def **step**
step(LangevinIntegrator self, int steps)

def **__init__**
init(OpenMM::LangevinIntegrator self, double temperature, double frictionCoeff, double stepSize) -> **LangevinIntegrator** **init**(OpenMM::LangevinIntegrator self, **LangevinIntegrator** other) -> **LangevinIntegrator**

def **__del__**
del(OpenMM::LangevinIntegrator self)

Setting non-bonded interactions in OpenMM

system = prmtop.createSystem(nonbondedMethod=NoCutoff, constraints=HBonds)

Value	Meaning
`NoCutoff`	No cutoff is applied.
`CutoffNonPeriodic`	The reaction field method is used to eliminate all interactions beyond a cutoff distance. Not valid for AMOEBA.
`CutoffPeriodic`	The reaction field method is used to eliminate all interactions beyond a cutoff distance. Periodic boundary conditions are applied, so each atom interacts only with the nearest periodic copy of every other atom. Not valid for AMOEBA.
`Ewald`	Periodic boundary conditions are applied. Ewald summation is used to compute long range interactions. (This option is rarely used, since PME is much faster for all but the smallest systems.) Not valid for AMOEBA.
`PME`	Periodic boundary conditions are applied. The Particle Mesh Ewald method is used to compute long range interactions.

Constraining certain bonds and angles in OpenMM

system = prmtop.createSystem(nonbondedMethod=NoCutoff, constraints=HBonds)

Value	Meaning	Time step
None	No constraints are applied. This is the default value.	<= 1 fs
HBonds	The lengths of all bonds that involve a hydrogen atom are constrained.	2 fs
AllBonds	The lengths of all bonds are constrained.	
HAngles	The lengths of all bonds are constrained. In addition, all angles of the form H-X-H or H-O-X (where X is an arbitrary atom) are constrained.	Up to 4 fs

Constraints? => Larger integration time steps => Greater speedup.

(Note: Be aware of the added level of approximation and apply constraints with care.)

Hint: It's good practice to choose 'None' while heating a system. Once equilibrated, one can safely choose 'Hbonds' for production runs.

By default, bonds and angles of water molecules are constrained (accessible through `rigidWater` parameter):

```
system = prmtop.createSystem(nonbondedMethod=NoCutoff,
constraints=None, rigidWater=False)
```

Reporters in OpenMM

Reporters for printing values of State parameters during simulation:

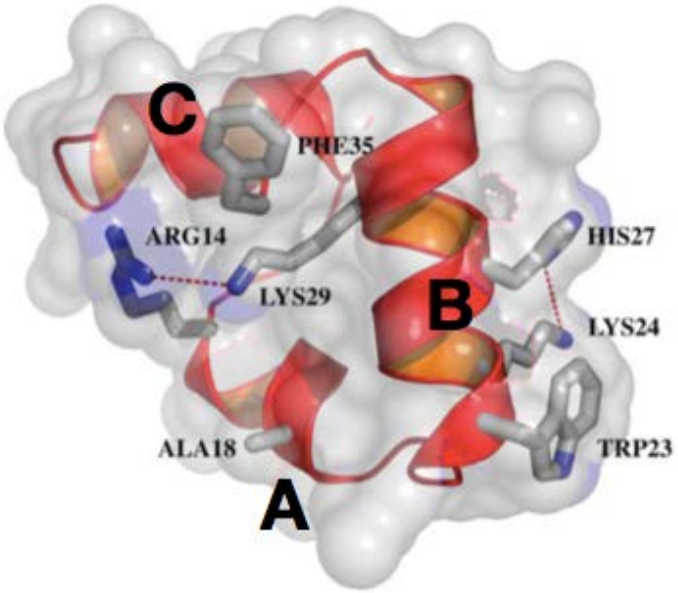
```
simulation.reporters.append(StateDataReporter('data.txt', 1000, time=True,
temperature=True, kineticEnergy=True, potentialEnergy=True, totalEnergy=True,
volume=True, density=True, separator=' '))
```

Check out talk by Lee-Ping tomorrow, March 27, 2:15 pm!

simbios

Exercise 2: Simulating the villin headpiece in implicit solvent

Task: Set up the protein in the provided .pdb file and perform an implicit water MD.



Copy exercise1.py to exercise2.py
Use input_exercise2.pdb as the starting structure.

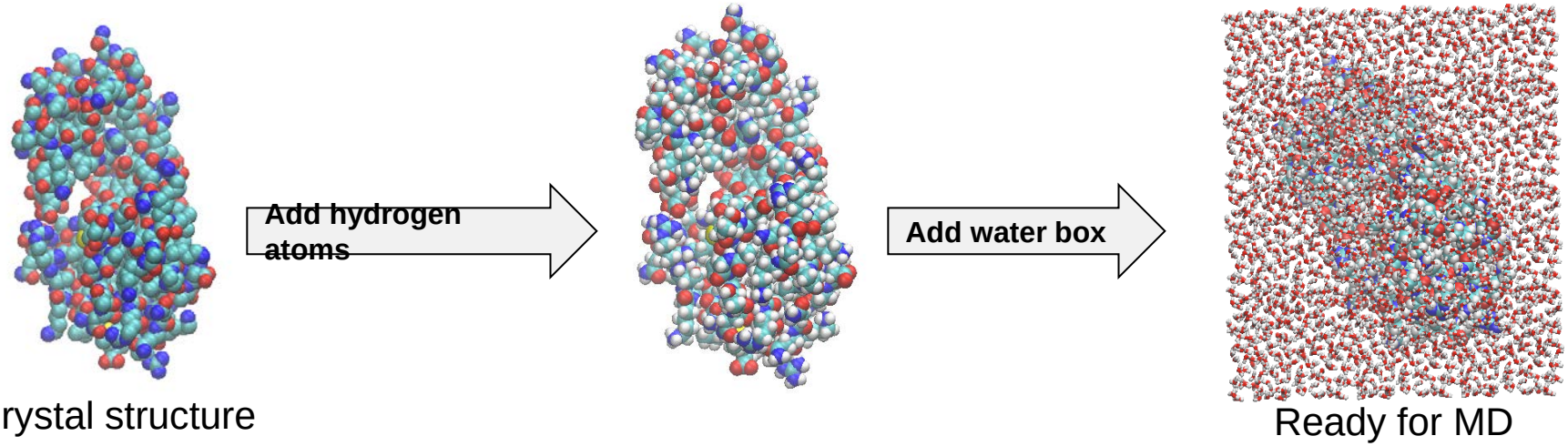
Exercise 2: Simulating the villin headpiece in implicit solvent

```
ValueError: No template found for residue 1 (LEU)

This message implies that there are missing atoms in the residues
of this Protein. We need to use modeller class to build missing atoms.
```

Model Building and Editing with OpenMM

(refer OpenMM Application Guide section 5)



OpenMM modeller class can fix these issues.

```
pdb = PDBFile('input.pdb')
modeller = Modeller(pdb.topology, pdb.positions)
# ... Call some modelling functions here ...
system = forcefield.createSystem(modeller.topology, nonbondedMethod=PME)
```

Available modeller functions:

Adding Hydrogen: modeller.addHydrogens(forcefield, pH=5.0)

Adding Solvent: modeller.addSolvent(forcefield, padding=1.0*nanometers, model='tip5p')
modeller.addSolvent(forcefield, boxSize=Vec3(5.0, 3.5, 3.5)*nanometers)

Adding Ions: modeller.addSolvent(forcefield, ionicStrength=0.1*molar, positiveIon='K+')

Exercise 2: Simulating the villin headpiece in implicit solvent

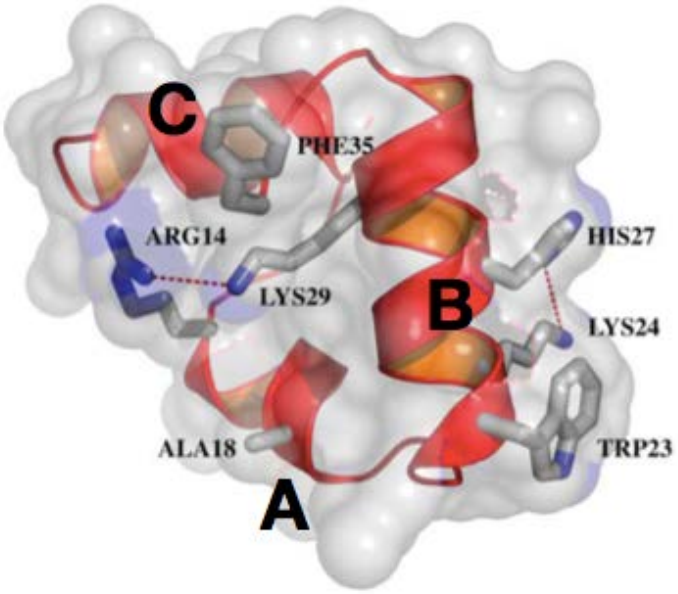
Task: Set up the protein in the provided .pdb file and perform an implicit water MD.

Instruction: Add modeller and replace the previous pdb.topology and pdb.positions with modeller.topology and modeller.positions.

```
pdb = PDBFile('input_exercise2.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')
print "Building Model.."
modeller = Modeller(pdb.topology, pdb.positions)
print('Adding hydrogens...')
modeller.addHydrogens(forcefield)
print "Creating System"
system = forcefield.createSystem(modeller.topology, constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)
simulation = Simulation(modeller.topology, system, integrator)
print "Using Platform:", simulation.context.getPlatform().getName()
simulation.context.setPositions(modeller.positions)
```

Exercise 2: Simulating the villin headpiece in implicit solvent

Task: Set up the protein in the provided .pdb file and perform an implicit water MD.



Now that we fixed the error:

- Use amber99_obc as the implicit solvent model.
- Change friction parameter to 91/ps for the implicit solvent model.
- Remove nonbonded parameters (method and cutoff) from the System object, which are necessary only for explicit solvent simulations.

Exercise 2: Simulating the villin headpiece in implicit solvent

Task: Set up the protein in the provided .pdb file and perform an implicit water MD.

Your input file should now look like this:

```
pdb = PDBFile('input_exercise2.pdb')
forcefield = ForceField('amber99sb.xml', 'amber99_obc.xml')
print "Building Model.."
modeller = Modeller(pdb.topology, pdb.positions)
print('Adding hydrogens...')
modeller.addHydrogens(forcefield)
print "Creating System"
system = forcefield.createSystem(modeller.topology, constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 91/picosecond, 0.002*picoseconds)
simulation = Simulation(modeller.topology, system, integrator)
print "Using Platform:", simulation.context.getPlatform().getName()
simulation.context.setPositions(modeller.positions)
```

Exercise 3: Simulating the villin headpiece in explicit solvent

Task: Build a water box with the modeller class and perform an explicit water MD.

Instructions:

- Copy exercise1.py to exercise3.py
- Use input_exercise2.pdb as the starting structure.
- Continue using amber99sb and tip3p as the explicit solvent model.
- Add a box of explicit water molecules using the modeller class with a 1nm padding.

Exercise 3: Simulating the villin headpiece in explicit solvent

Task: Build a water box with the modeller class and perform an explicit water MD.

So, …
… we copied exercise1.py to exercise3.py,
… used input_exercise2.pdb as the starting structure,
… used amber99sb and tip3p as the explicit solvent model,
… and added explicit water molecules using the modeller class.

```
pdb = PDBFile('input_exercise2.pdb')
forcefield = ForceField('amber99sb.xml', 'tip3p.xml')
print "Building Model..."
modeller = Modeller(pdb.topology, pdb.positions)
print('Adding hydrogens...')
modeller.addHydrogens(forcefield)
print "Adding Water.."
modeller.addSolvent(forcefield, model='tip3p', padding=1.0*nanometers)
print "Creating System..."
system = forcefield.createSystem(modeller.topology, nonbondedMethod=PME,
nonbondedCutoff=1.0*nanometer, constraints=HBonds)
integrator = LangevinIntegrator(300*kelvin, 1/picosecond, 0.002*picoseconds)
```