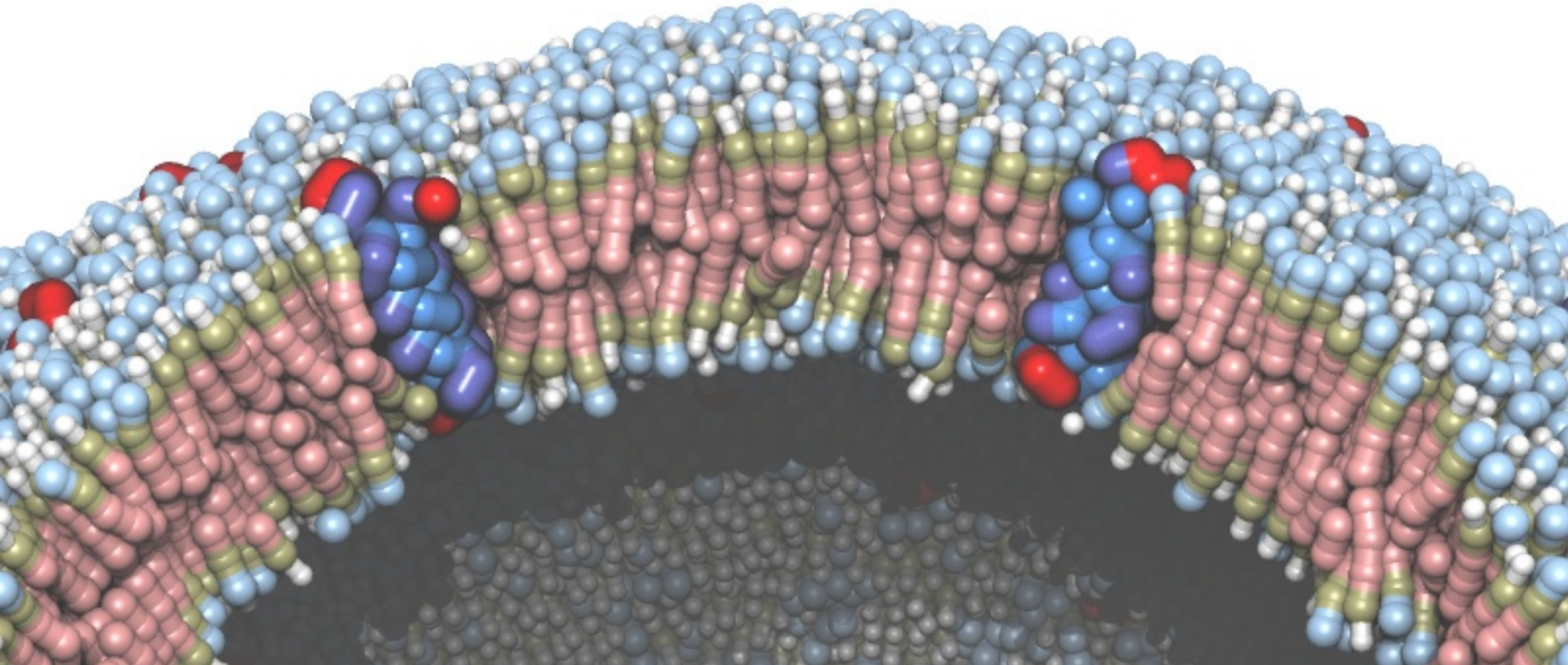
“Moltemplate Molecule Builder Tech Demo”

Andrew Jewett

LAMMPS workshop

August 7th, 2013

UCSB Physics



Moltemplate

Moltemplate is a general molecule builder and force-field database system for LAMMPS. (It creates LAMMPS "data files" and "input scripts".)

Goal:

Any system which LAMMPS can simulate can be prepared with moltemplate (almost any).

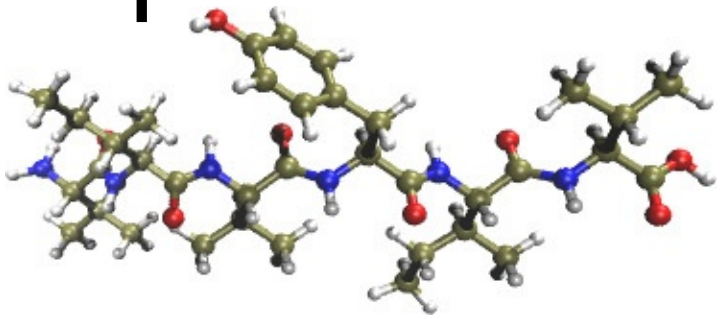
LAMMPS software:

- requires a **huge list** of atoms, bonds, coordinates, angles, etc...

```
Atoms

1 1 1 0.0 0.0 0.0 108.75
2 1 2 0.0 -1.0 0.0 101.25
3 2 3 0.0 3.75 6.49519 105.0
4 2 2 0.0 2.75 6.49519 97.5
  :

Bonds

1 1 1 3
2 2 1 4
3 3 2 5
  :

Angles

1 1 2 1 3
2 2 2 1 4
  :
```

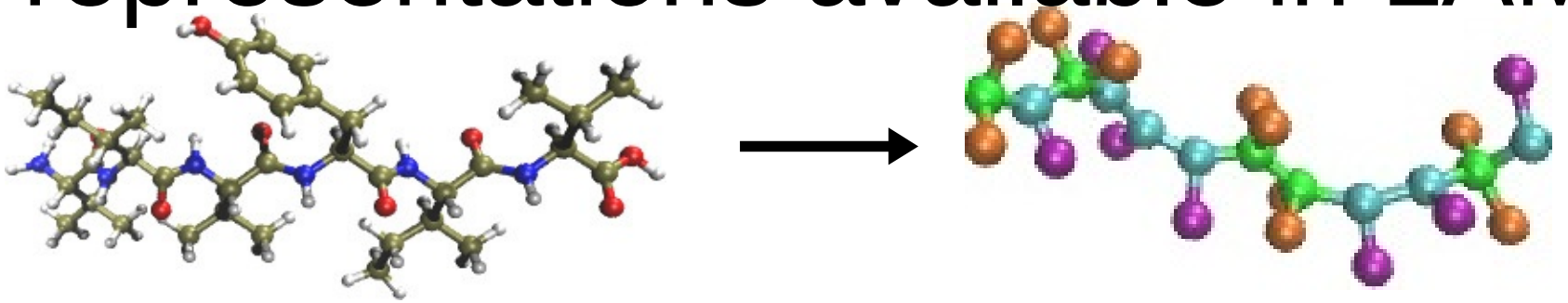
(LAMMPS data file format)

Diverse zoo of exotic molecular representations available in LAMMPS



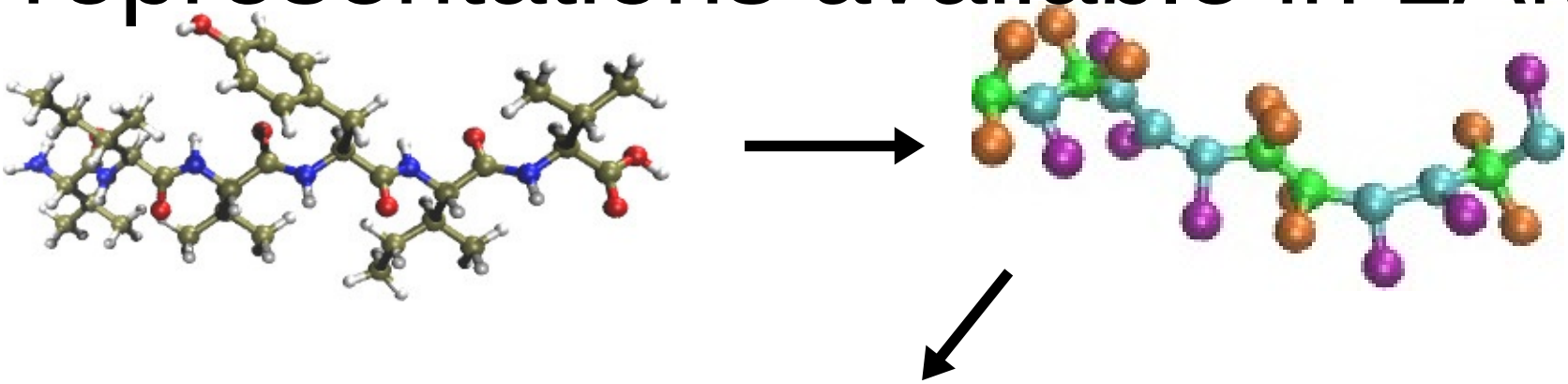
All atom models (explicit solvent)
(particles represent individual atoms)

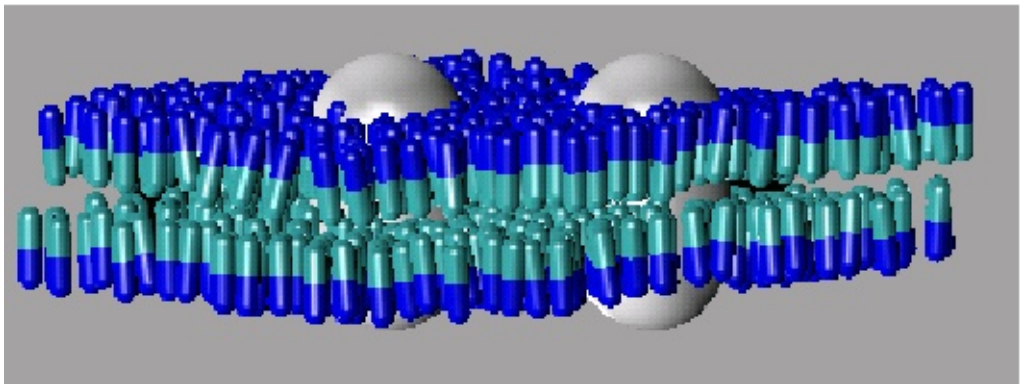
Diverse zoo of exotic molecular representations available in LAMMPS



Reduced atom models
(crude implicit solvent, typically)
particles represent chemical groups

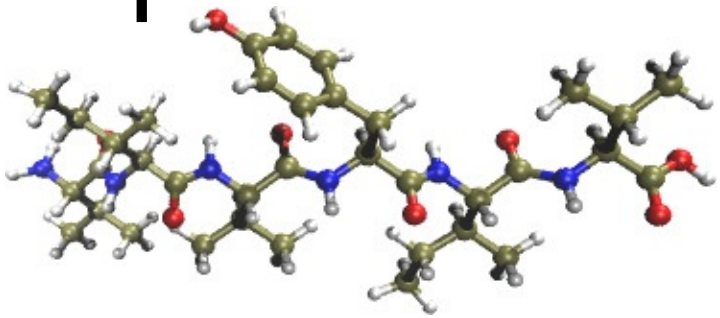
Diverse zoo of exotic molecular representations available in LAMMPS

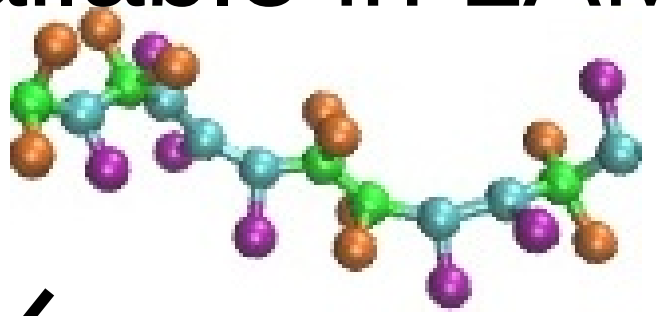


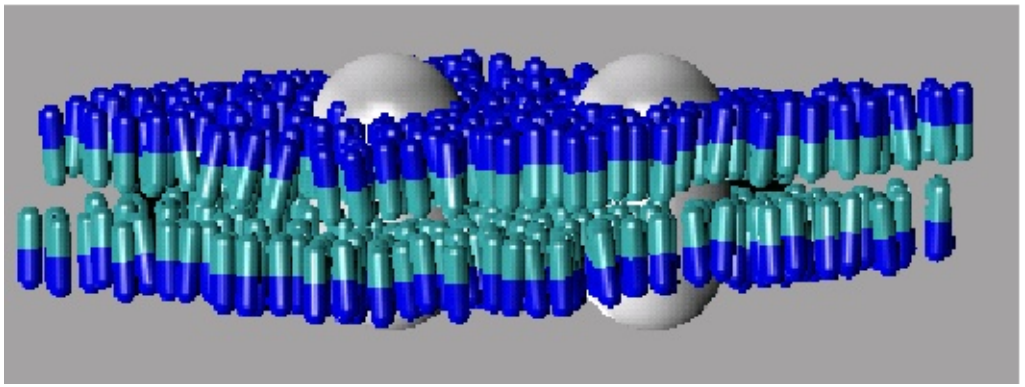


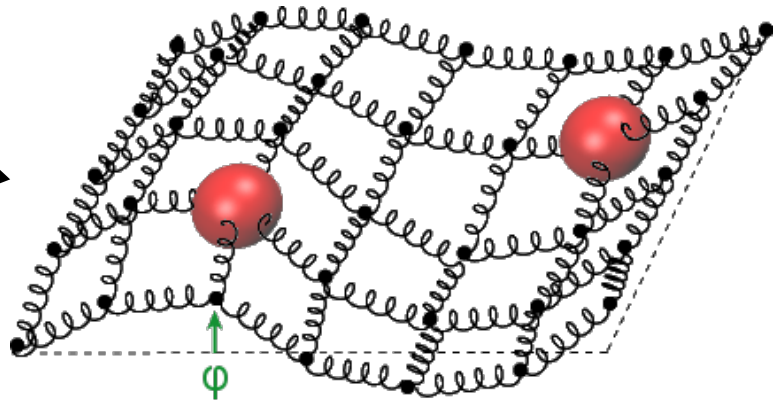
Non-point-like particles. (In this example by Grace Brannigan, ellipsoids and dipoles represent proteins and lipids.)

Diverse zoo of exotic molecular representations available in LAMMPS



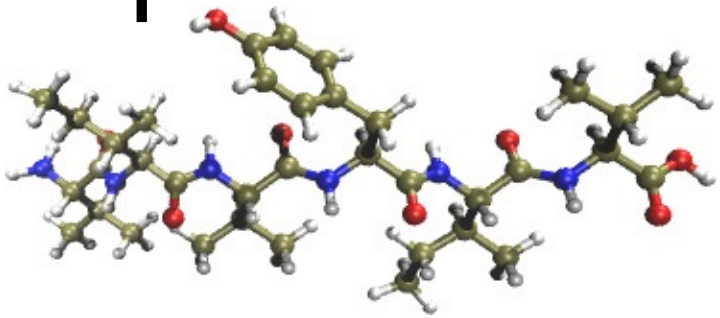


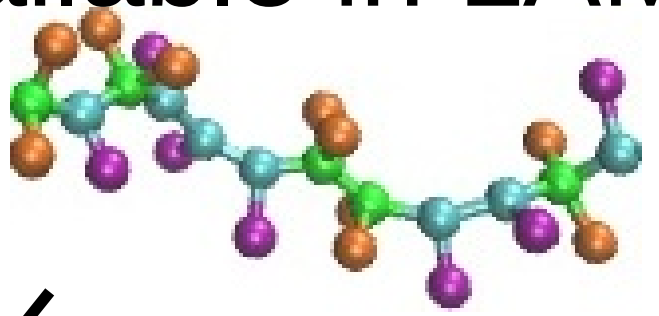


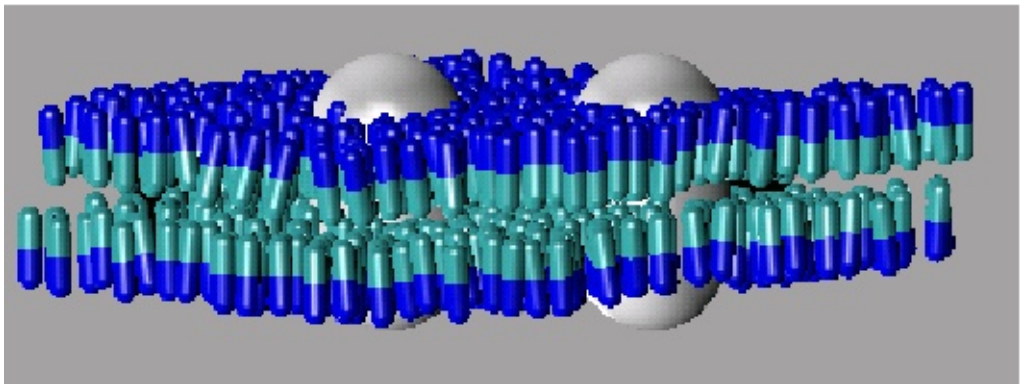


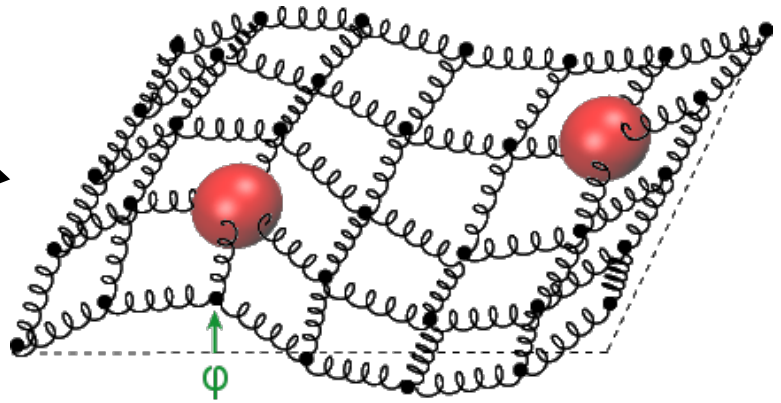
Coupled **continuum-field + discrete** particle hybrid systems

Diverse zoo of exotic molecular representations available in LAMMPS









Coupled **continuum-field + discrete** particle hybrid systems

New force-fields and atom styles are added by the community frequently...

LAMMPS

- **General**

  -Supports atoms, “*exotic” atoms*, and *continuum-field* hybrids

  -Force-field parameters can evolve over time

  -Particles and molecules can be created and destroyed

- **Modular**

  -Combine different molecule representations and force-fields

- **Customizable**

  -New force-fields and features are constantly submitted by users

→ **The file format is always changing**

LAMMPS

- **General**

  -Supports atoms, “*exotic*” *atoms*, and *continuum-field* hybrids

  -Force-field parameters can evolve over time

  -Particles and molecules can be created and destroyed

- **Modular**

  -Combine different molecule representations and force-fields

- **Customizable**

  -New force-fields and features are constantly submitted by users

  -**The file format is always changing**

→ **Writing a *general* molecule builder for LAMMPS is hard.**

LAMMPS

- **General**

  -Supports atoms, “*exotic” atoms*, and *continuum-field* hybrids

  -Force-field parameters can evolve over time

  -Particles and molecules can be created and destroyed

- **Modular**

  -Combine different molecule representations and force-fields

- **Customizable**

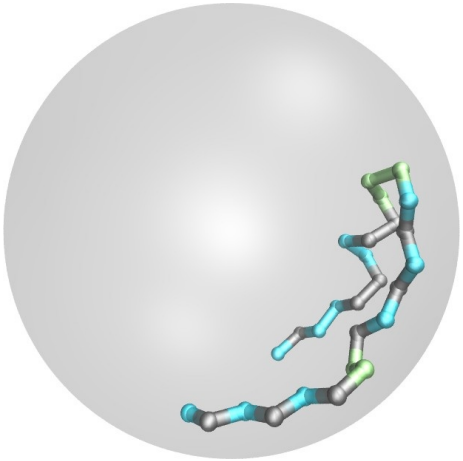
  -New force-fields and features are constantly submitted by users

  -**The file format is always changing**

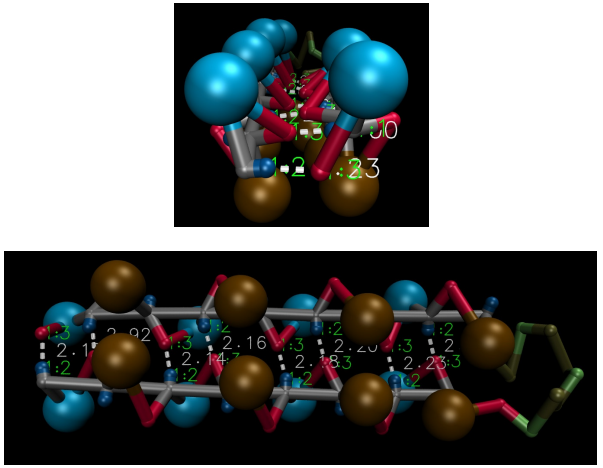
Writing a *general* molecule builder for LAMMPS is hard.

→ **templates**

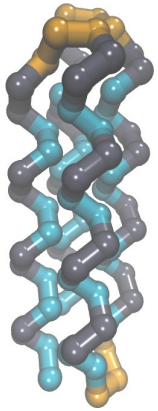
Exotic molecules built with moltemplate:



"1-bead/residue αβ-peptide + chaperonin",
A.I. Jewett, A. Baumketner, J-E. Shea, PNAS 2004



"4-bead/residue β-amyloid model",
A.I. Jewett, Z. Zhuang, J-E. Shea, manuscript in preparation



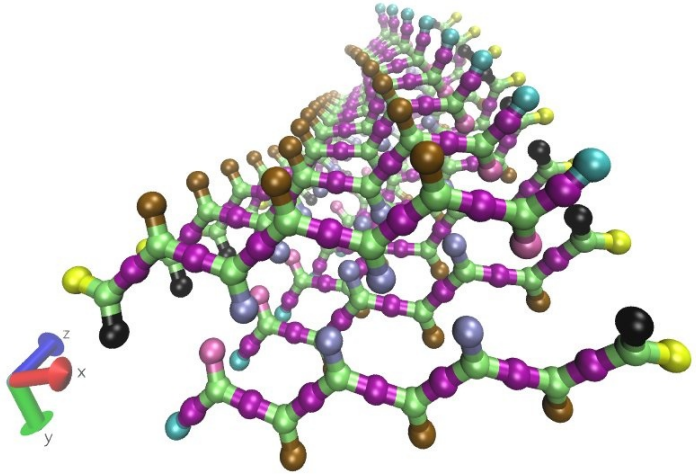
"4HelixBundle"

Zhuang, Z., Jewett, A. I., ... Shea, J.-E., Biophys. J., 100(5):1306-1315 (2011)

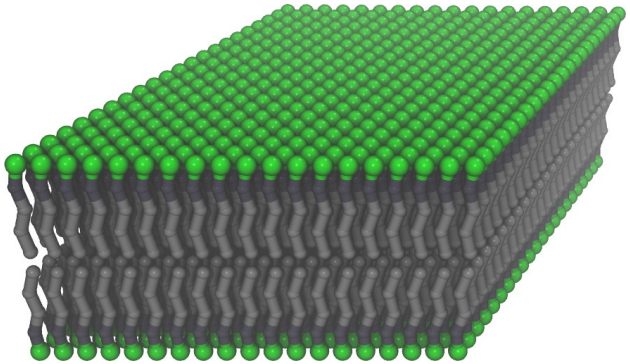
Bellesia, G., Jewett, A. I., and Shea. J.-E., Prot. Sci., 20(5):818-826 (2011)

Bellesia, G., Jewett, A. I., and Shea, J.-E., Prot. Sci., 19(1):141-154 (2010)

Exotic assemblies built with moltemplate:



"3-bead/residue β-amyloid model",
G. Bellesia, J-E Shea, J. Chem. Phys. 126:245104 (2007)

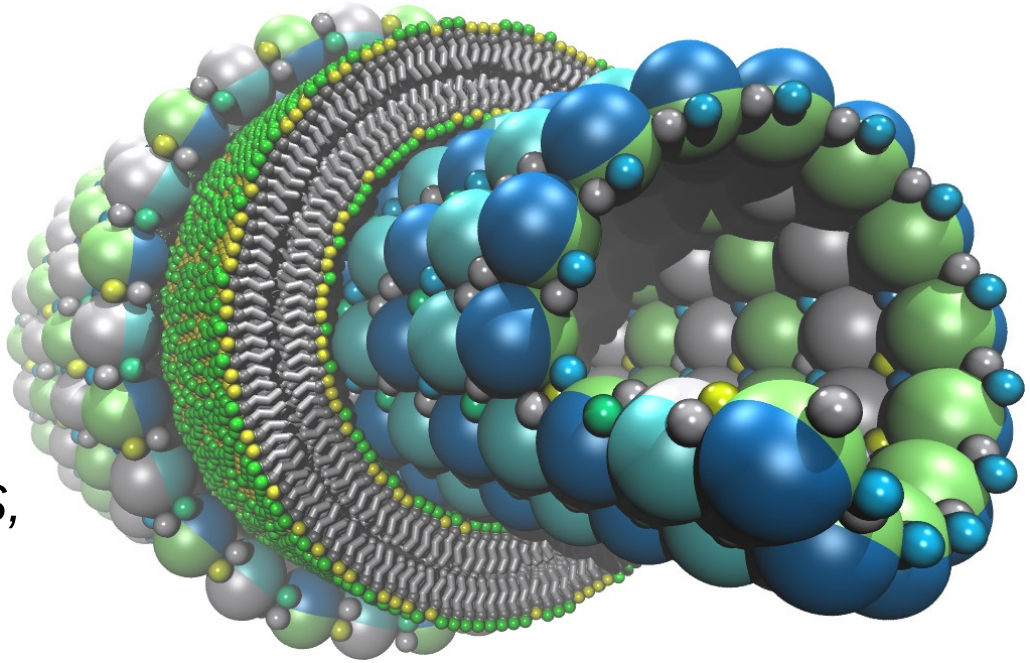


"lipid bilayer (DPPC)"
G. Brannigan, P.F. Philips, and F.L.H. Brown, Physical Review E, Vol 72, 011915 (2005)
(files located in examples/CG_biomolecules/membrane_BranniganPRE2005)

Microtubule + lipids

(lipid protein nantube, "LPN")

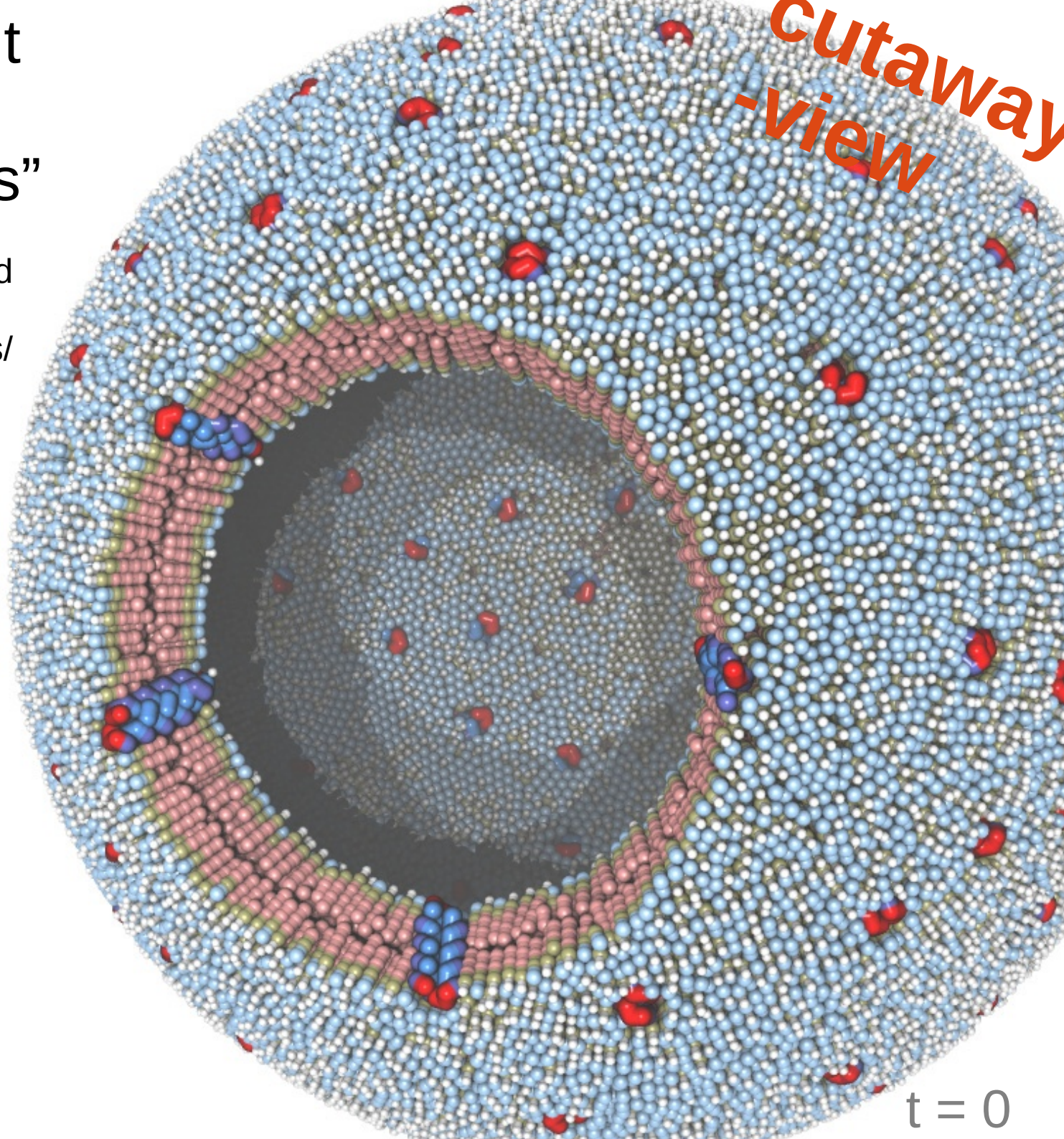
U. Raviv, D.J. Needleman, Y. Li, H.P. Miller, L. Wilson, and C.R. Safinya, PNAS, 102(32), 11167-11172, 2005



“multicomponent vesicle with protein inclusions”

(...built with moltemplate and packmol. Files located in: examples/CG_biomolecules/vesicle_PACKMOL)

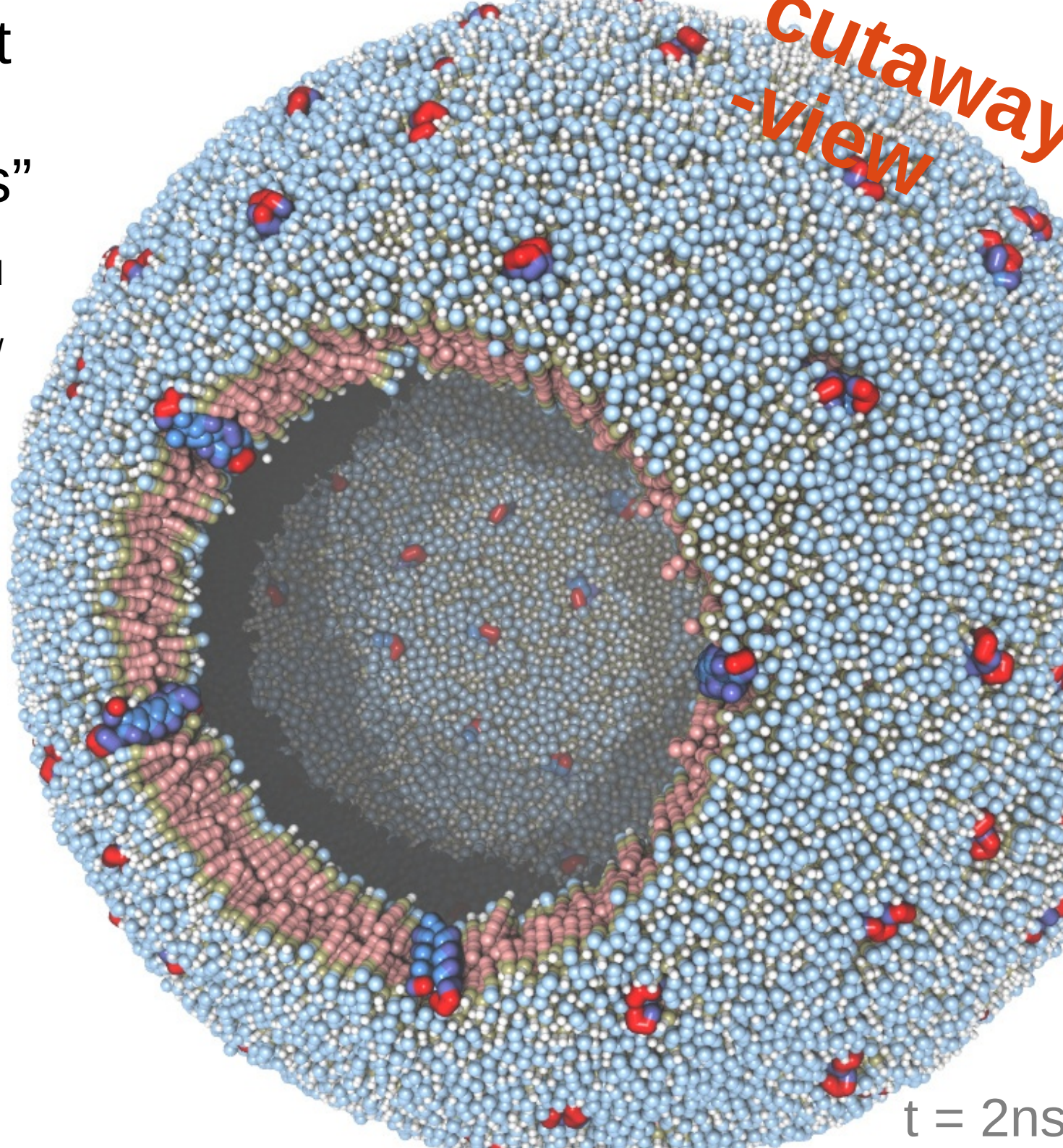
Coordinates generated by PACKMOL and loaded into moltemplate (using the “-xyz file” argument)



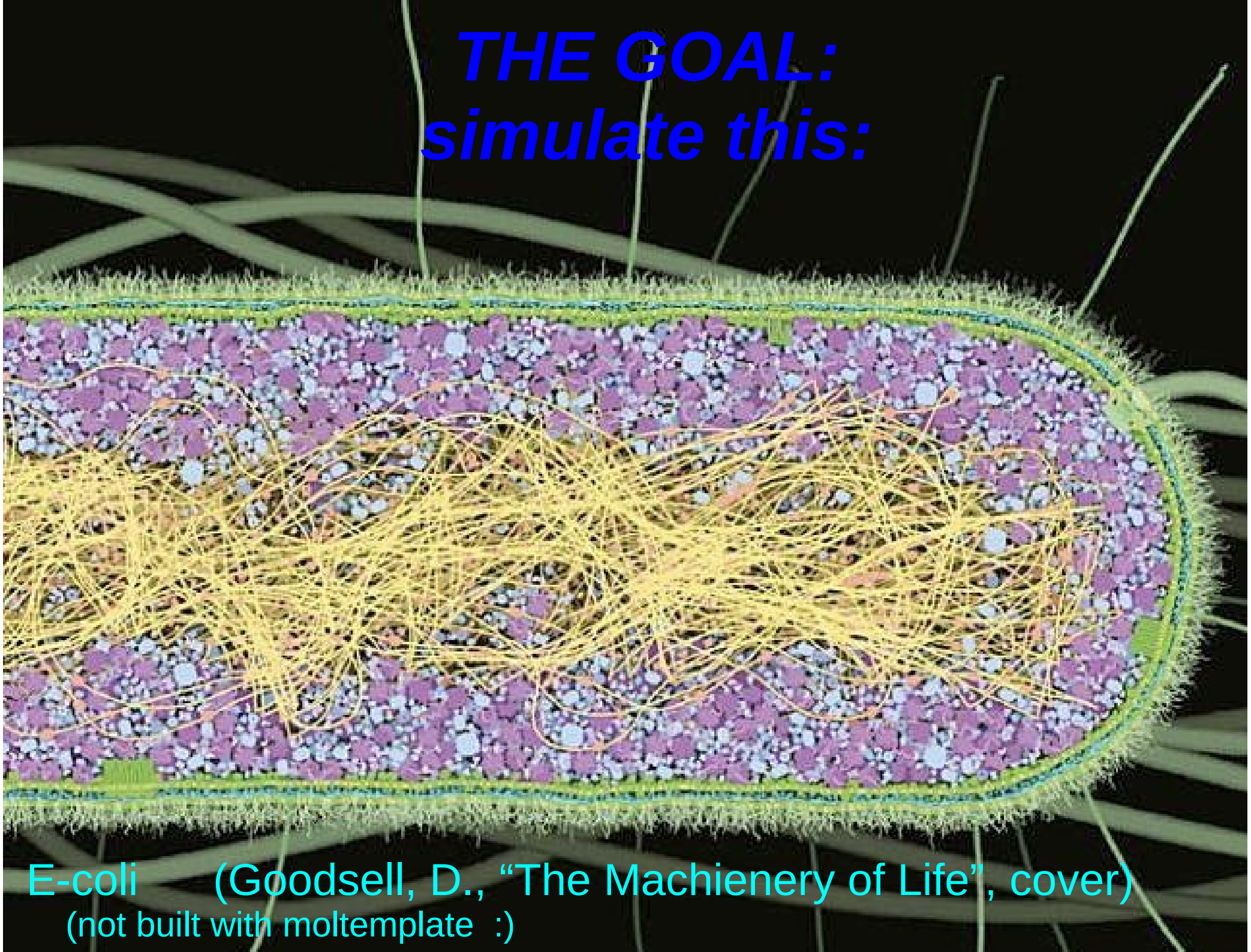
“multicomponent vesicle with protein inclusions”

(...built with moltemplate and packmol. Files located in: examples/CG_biomolecules/vesicle_PACKMOL)

Coordinates generated by PACKMOL and loaded into moltemplate (using the “-xyz file” argument)



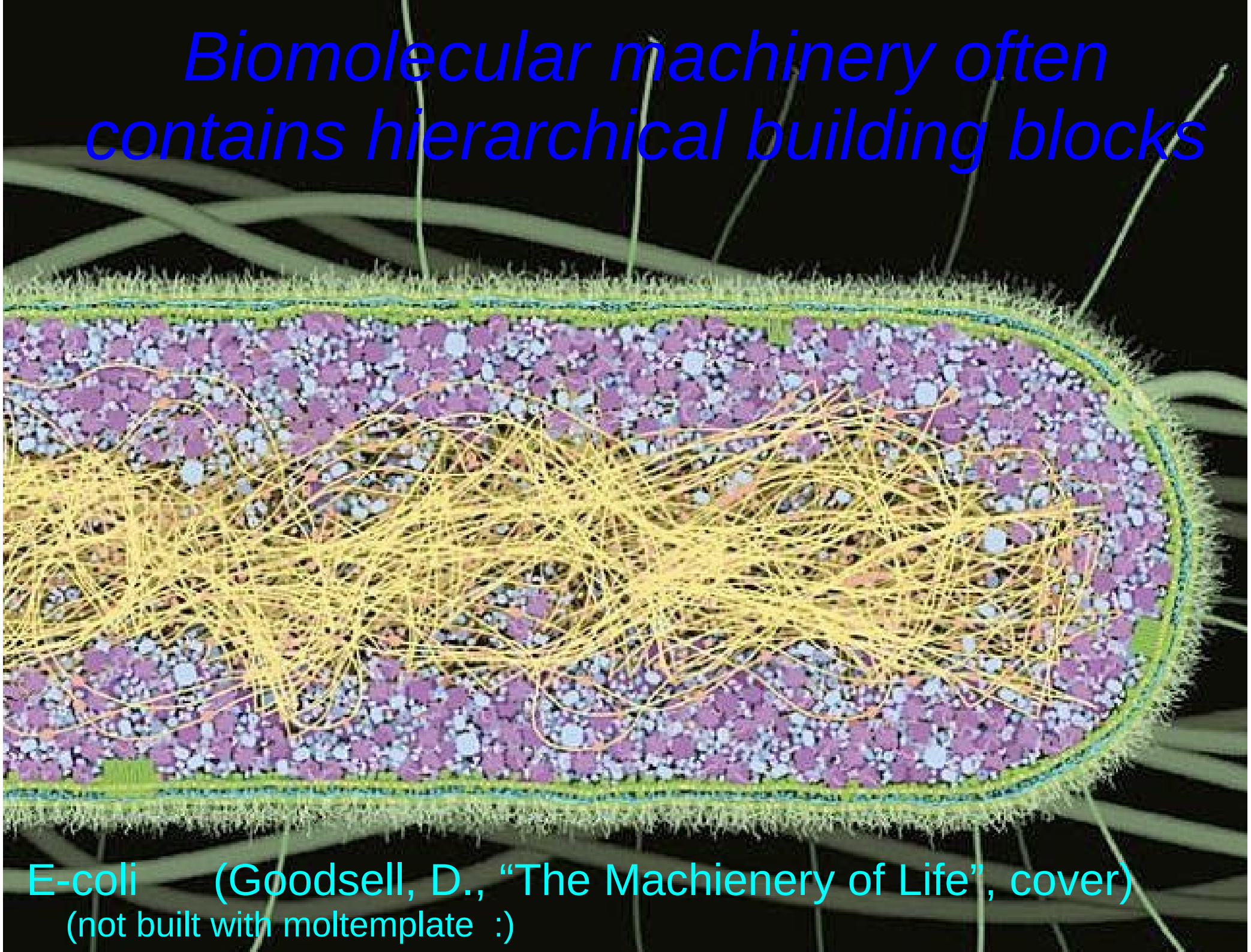
THE GOAL:
simulate this:



E-coli (Goodsell, D., "The Machienery of Life", cover)

(not built with moltemplate :)

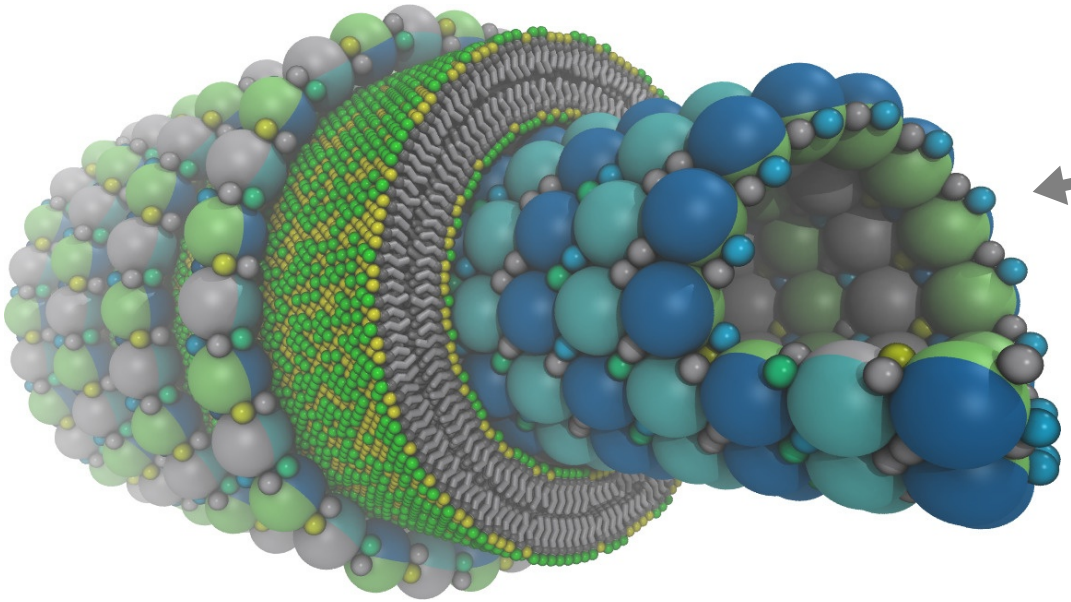
Biomolecular machinery often contains hierarchical building blocks



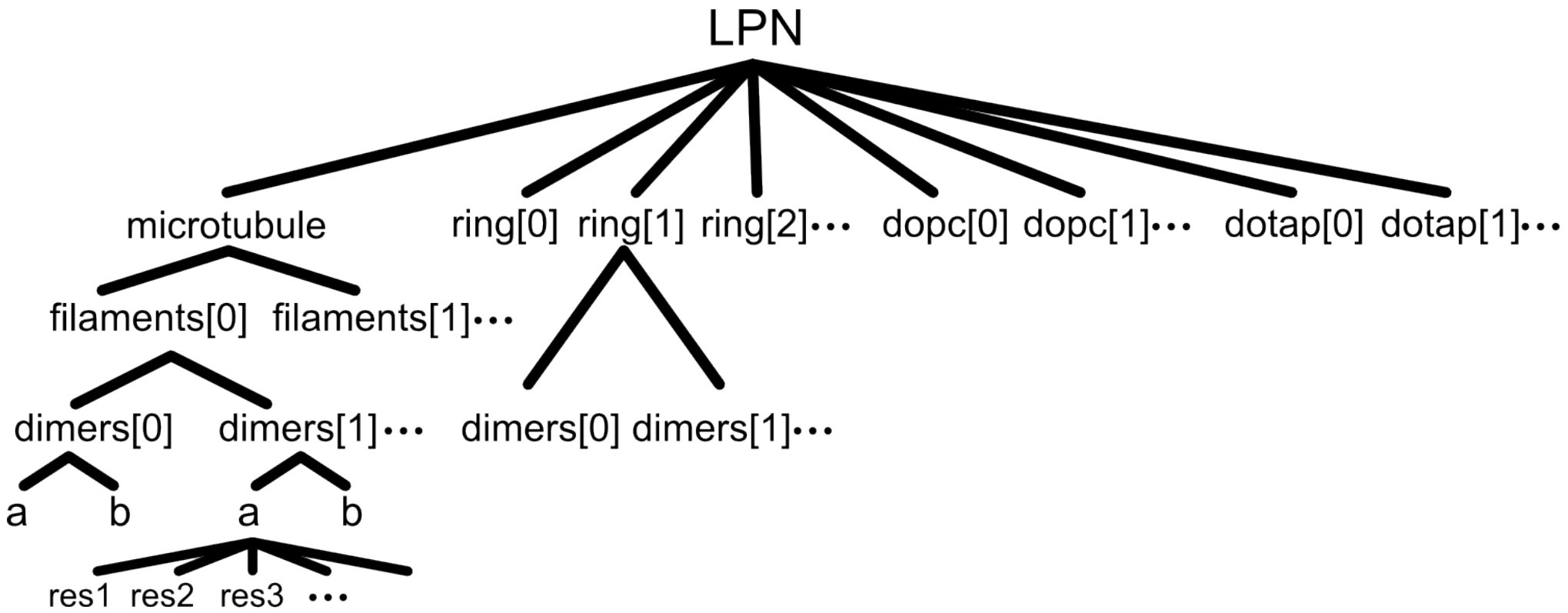
E-coli (Goodsell, D., “The Machienery of Life”, cover)

(not built with moltemplate :)

hierarchical example



LPN example (again)

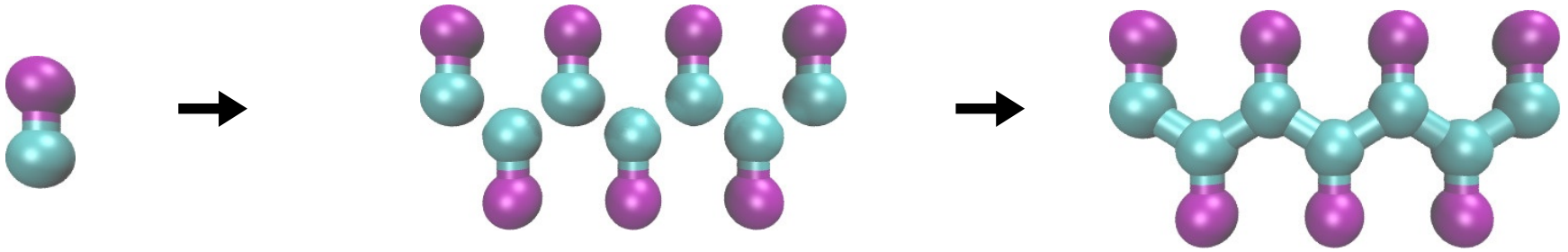


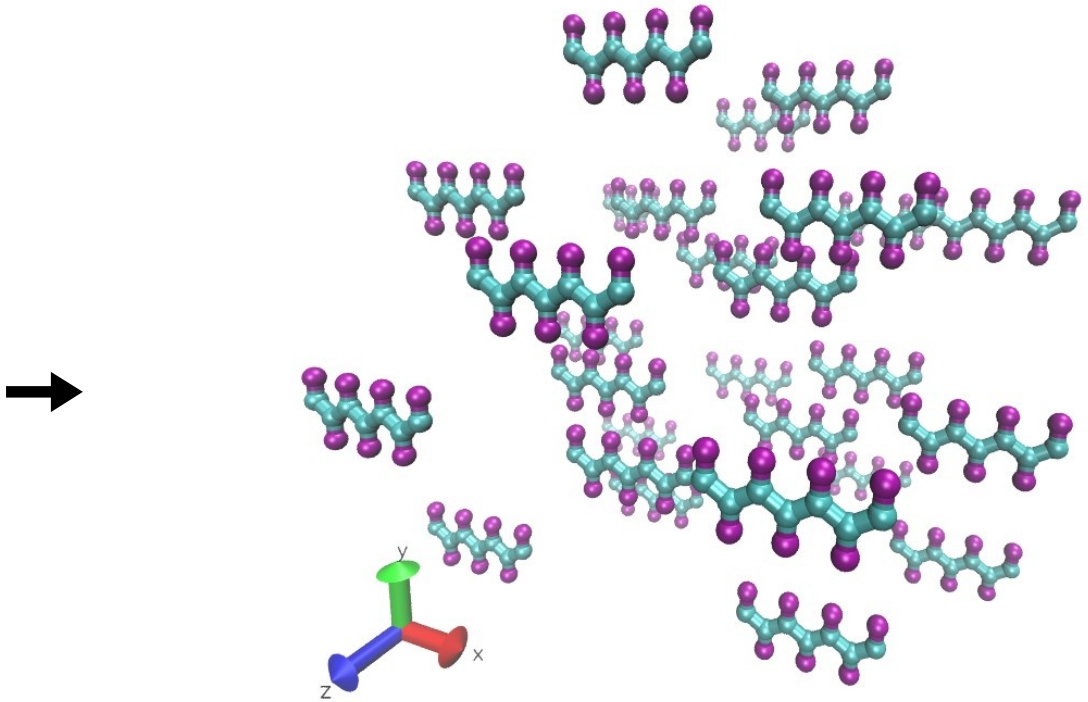
Moltemplate lets you build big things out of small things

(...and use them to build bigger things)

Object composition

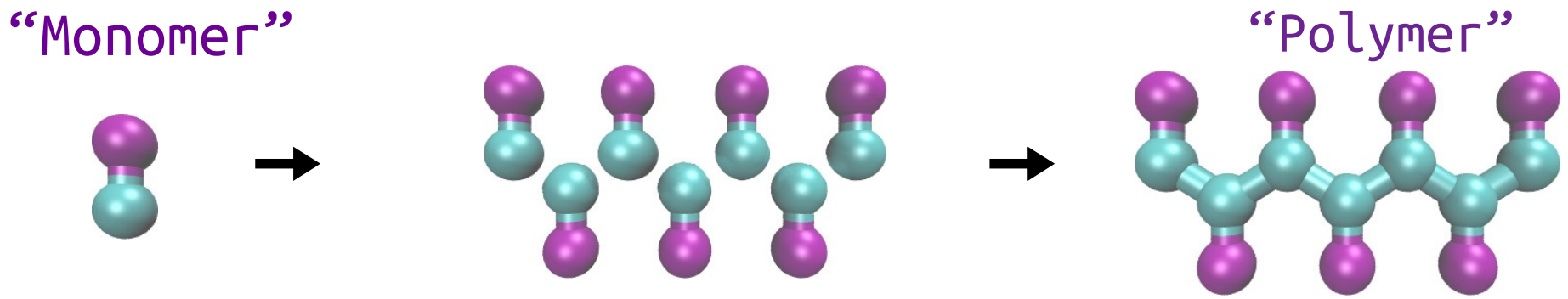
Objects can be built from smaller objects

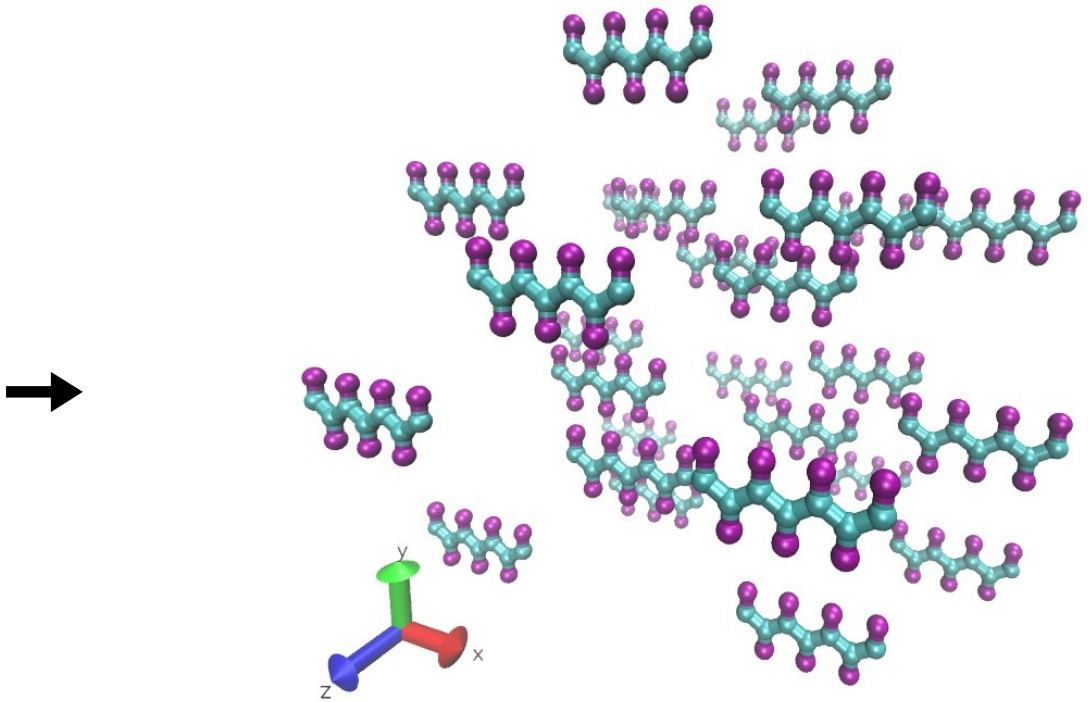




Object composition

Objects can be built from smaller objects





Object composition

Objects can be built from smaller objects

"Monomer"

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Moltemplate creates the files LAMMPS needs:

-A (LAMMPS) ***Data*** *file* (coords and topology)

-A (LAMMPS) ***In****put script* (force fields and constraints)

-(optional) simple auxiliary files needed to run LAMMPS

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

Define the “Monomer” molecule. (Use it later.)

“Monomer”

Writing to “**Data** ...” copies the enclosed text into the corresponding section of the LAMMPS data file (generated by moltemplate)

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

"Monomer"

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Writing to "**In** ..." copies the enclosed text into the corresponding section of a LAMMPS input script file (generated by moltemplate)

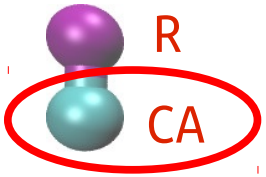
Example of a molecule object

"Monomer"


```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

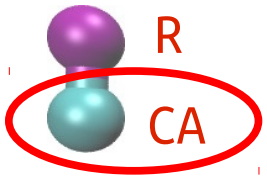
Example of a molecule object

"Monomer"




```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA   0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R    0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

"Monomer"

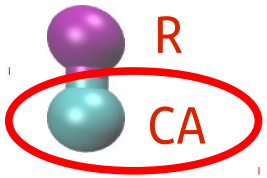


Atom Type (@ = type)

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA   0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R    0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

"Monomer"

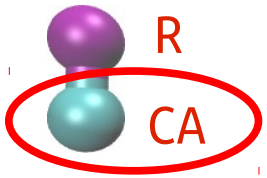


Unique
Atom ID
($ = name)

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

"Monomer"



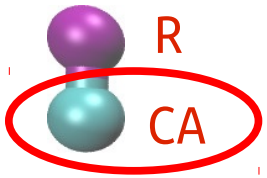
Unique
Atom ID
($ = name)

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

(this particular atom is
referred to later in a bond)

Example of a molecule object

"Monomer"



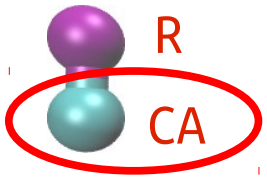
Atom Type (@ = type)

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA   0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R    0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Mass

Example of a molecule object

"Monomer"



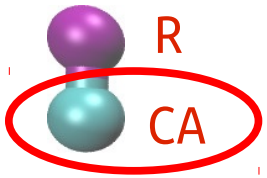
Atom Type (@ = type)

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA    0.10  2.0
    pair_coeff  @atom:R  @atom:R     0.50  3.6
    bond_coeff  @bond:sidechain      30.0  3.4
  }
}
```

List of non-bonded ("pair") interaction parameters (depends on "pair_style")

Example of a molecule object

"Monomer"

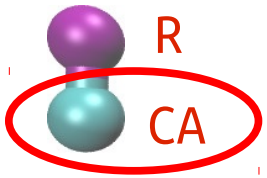


charge

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

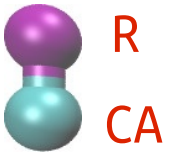
"Monomer"




```
Monomer {
  write("Data Atoms") {                                    X       Y       Z
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

"Monomer"



$mol: preceeds the name of a molecule id counter. Normally you can give it any name you like. (It is frequently named ".")
But "..." is used whenever this molecule might be part of a larger molecule.

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

"Monomer"

Note: You can create your own custom or shared counters (which do not need to have global scope). *This is useful in case new atom_styles are invented later.*

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

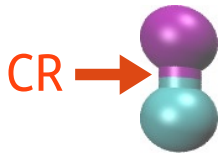
"Monomer"

Moltemplate format is identical to native LAMMPS formats.
Column format of the "Data Atoms" section depends on the LAMMPS "***atom_style***". Most atom styles are supported (even hybrid styles).

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

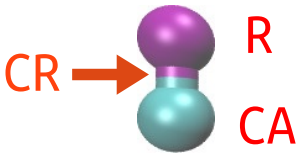
"Monomer"




```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

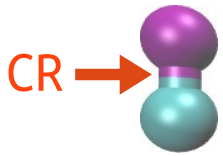
"Monomer"




```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Example of a molecule object

"Monomer"




```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Force-field parameters used for this type of bond (depends on the bond_style)

Example of a molecule object

"Monomer"

```
Monomer {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bonds") {
    $bond:CR  @bond:sidechain  $atom:ca  $atom:r
  }
  write_once("Data Masses") {
    @atom:CA  13.0
    @atom:R   50.0
  }
  write_once("In Settings") {
    pair_coeff  @atom:CA @atom:CA     0.10  2.0
    pair_coeff  @atom:R  @atom:R      0.50  3.6
    bond_coeff  @bond:sidechain       30.0  3.4
  }
}
```

Alternative example

...using a separate file to store the force field & mass

"Monomer"

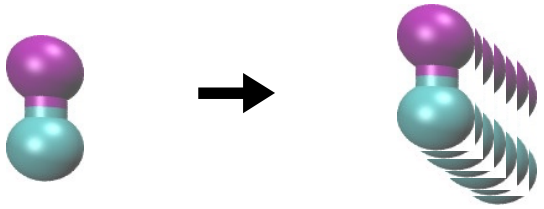
(This way, only the atomic charge, coordinates, and bonds are needed.)

```
import "force_field.lt"
Monomer inherits ForceField {
  write("Data Atoms") {
    $atom:ca  $mol:...  @atom:CA    0.0  0.000  1.0000   0.0000000
    $atom:r   $mol:...  @atom:R     0.0  0.000  4.4000   0.0000000
  }
  write("Data Bond List") {
    $bond:CR  $atom:ca  $atom:r
  }
}
```

An example showing the format of the "force_field.lt" file will be shown in later slides

Now create a larger molecule

"Monomer"

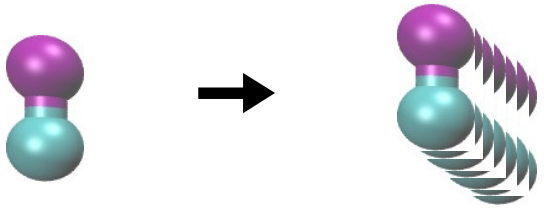


The "Polymer" molecule in this example is collection of 7 monomers

```
Polymer {
  monomers = new Monomer[7]
}
```

Now create a larger molecule

"Monomer"



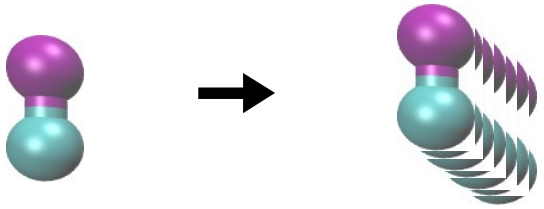
The "Polymer" molecule in this example is collection of 7 monomers

```
Polymer {
  monomers = new Monomer[7]
}
```

the "new" command creates one (or multiple) copies of the molecule

Now create a larger molecule

"Monomer"



The "Polymer" molecule in this example is collection of 7 monomers

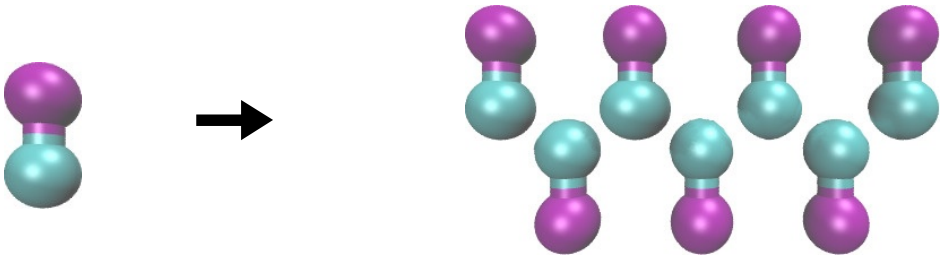
```
Polymer {
  monomers = new Monomer[7]
}
```

Problem: By default, when you create a new molecule, it has the same coordinates as the original (atoms overlap)

Object composition

Objects can be built from smaller objects

“Monomer” “Polymer”



```
Polymer {
  monomers = new Monomer[7].rot(180.0, 1,0,0).move(3.2,0,0)
}
```

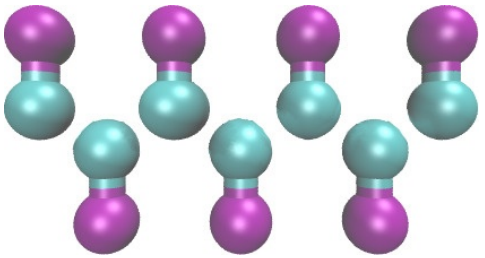
Solution: use coordinate transformations to specify the molecule position. *(tranformations following [] are stackable)*

Object composition

Objects can be built from smaller objects

"Monomer"

"Polymer"



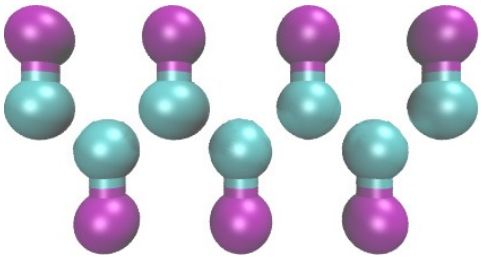
```
Polymer {
  mon1 = new Monomer
  mon2 = new Monomer.rot(180.0, 1,0,0).move(3.2,0,0)
  mon3 = new Monomer.rot(  0.0, 1,0,0).move(6.4,0,0)
  mon4 = new Monomer.rot(180.0, 1,0,0).move(9.6,0,0)
  mon5 = new Monomer.rot(  0.0, 1,0,0).move(12.8,0,0)
  mon6 = new Monomer.rot(180.0, 1,0,0).move(16.0,0,0)
  mon7 = new Monomer.rot(  0.0, 1,0,0).move(19.2,0,0)
}
```

Alternately, create molecules one at a time, and specify the position of each one manually. (This is a long-hand way of doing the same thing.)

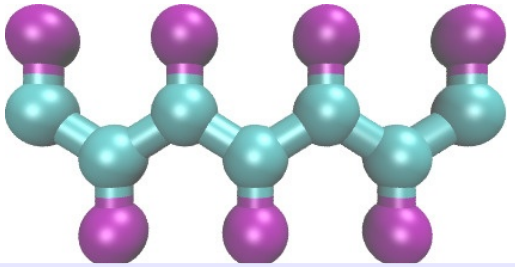
Object composition

Objects can be built from smaller objects

"Monomer"



"Polymer"



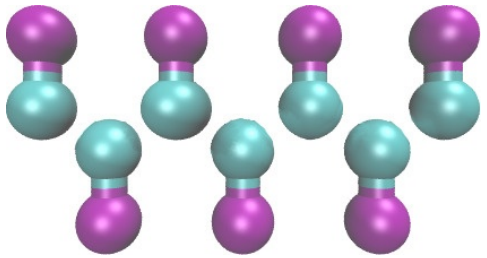
Now add bonds...

```
Polymer {
  mon1 = new Monomer
  mon2 = new Monomer.rot(180.0, 1,0,0).move(3.2,0,0)
  mon3 = new Monomer.rot(  0.0, 1,0,0).move(6.4,0,0)
  mon4 = new Monomer.rot(180.0, 1,0,0).move(9.6,0,0)
  mon5 = new Monomer.rot(  0.0, 1,0,0).move(12.8,0,0)
  mon6 = new Monomer.rot(180.0, 1,0,0).move(16.0,0,0)
  mon7 = new Monomer.rot(  0.0, 1,0,0).move(19.2,0,0)
  write("Data Bonds") {
    $bond:backbone1  @bond:2bead/backbone  $atom:mon1/ca  $atom:mon2/ca
    $bond:backbone2  @bond:2bead/backbone  $atom:mon2/ca  $atom:mon3/ca
    $bond:backbone3  @bond:2bead/backbone  $atom:mon3/ca  $atom:mon4/ca
    $bond:backbone4  @bond:2bead/backbone  $atom:mon4/ca  $atom:mon5/ca
    $bond:backbone5  @bond:2bead/backbone  $atom:mon5/ca  $atom:mon6/ca
    $bond:backbone6  @bond:2bead/backbone  $atom:mon6/ca  $atom:mon7/ca
  }
}
```

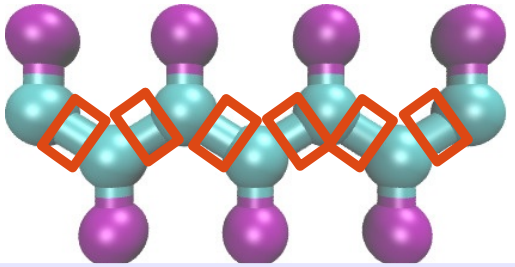
Object composition

Objects can be built from smaller objects

"Monomer"



"Polymer"

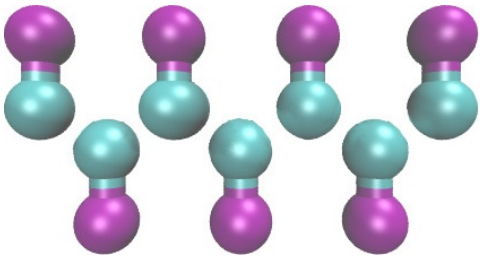


```
Polymer {
  mon1 = new Monomer
  mon2 = new Monomer.rot(180.0, 1,0,0).move(3.2,0,0)
  mon3 = new Monomer.rot(  0.0, 1,0,0).move(6.4,0,0)
  mon4 = new Monomer.rot(180.0, 1,0,0).move(9.6,0,0)
  mon5 = new Monomer.rot(  0.0, 1,0,0).move(12.8,0,0)
  mon6 = new Monomer.rot(180.0, 1,0,0).move(16.0,0,0)
  mon7 = new Monomer.rot(  0.0, 1,0,0).move(19.2,0,0)
  write("Data Bonds") {
    $bond:backbone1  @bond:2bead/backbone  $atom:mon1/ca  $atom:mon2/ca
    $bond:backbone2  @bond:2bead/backbone  $atom:mon2/ca  $atom:mon3/ca
    $bond:backbone3  @bond:2bead/backbone  $atom:mon3/ca  $atom:mon4/ca
    $bond:backbone4  @bond:2bead/backbone  $atom:mon4/ca  $atom:mon5/ca
    $bond:backbone5  @bond:2bead/backbone  $atom:mon5/ca  $atom:mon6/ca
    $bond:backbone6  @bond:2bead/backbone  $atom:mon6/ca  $atom:mon7/ca
  }
}
```

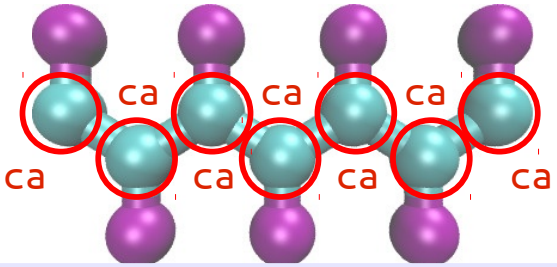
Object composition

Objects can be built from smaller objects

"Monomer"



"Polymer"

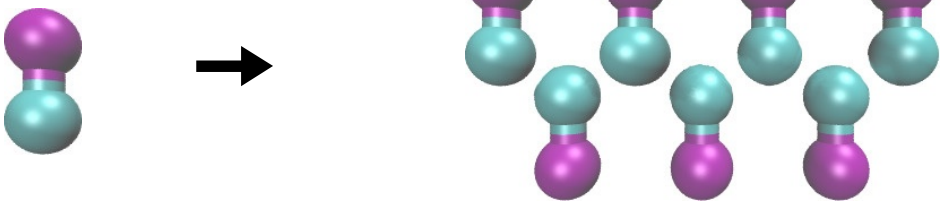



```
Polymer {
  mon1 = new Monomer
  mon2 = new Monomer.rot(180.0, 1,0,0).move(3.2,0,0)
  mon3 = new Monomer.rot(  0.0, 1,0,0).move(6.4,0,0)
  mon4 = new Monomer.rot(180.0, 1,0,0).move(9.6,0,0)
  mon5 = new Monomer.rot(  0.0, 1,0,0).move(12.8,0,0)
  mon6 = new Monomer.rot(180.0, 1,0,0).move(16.0,0,0)
  mon7 = new Monomer.rot(  0.0, 1,0,0).move(19.2,0,0)
  write("Data Bonds") {
    $bond:backbone1  @bond:2bead/backbone  $atom:mon1/ca  $atom:mon2/ca
    $bond:backbone2  @bond:2bead/backbone  $atom:mon2/ca  $atom:mon3/ca
    $bond:backbone3  @bond:2bead/backbone  $atom:mon3/ca  $atom:mon4/ca
    $bond:backbone4  @bond:2bead/backbone  $atom:mon4/ca  $atom:mon5/ca
    $bond:backbone5  @bond:2bead/backbone  $atom:mon5/ca  $atom:mon6/ca
    $bond:backbone6  @bond:2bead/backbone  $atom:mon6/ca  $atom:mon7/ca
  }
}
```

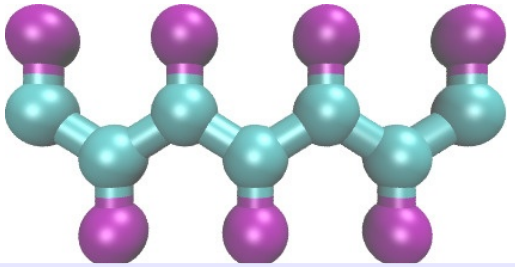
Object composition

Objects can be built from smaller objects

"Monomer"



"Polymer"

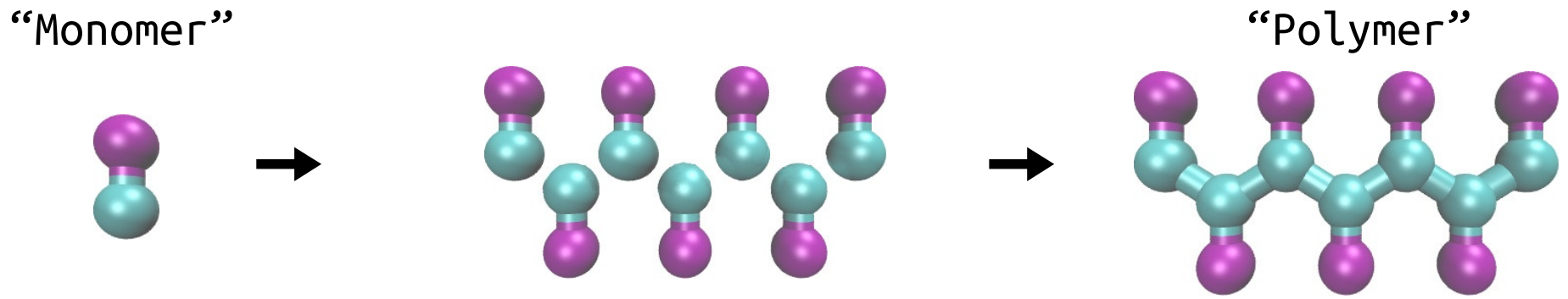


```
Polymer {
  mon1 = new Monomer
  mon2 = new Monomer.rot(180.0, 1,0,0).move(3.2,0,0)
  mon3 = new Monomer.rot(  0.0, 1,0,0).move(6.4,0,0)
  mon4 = new Monomer.rot(180.0, 1,0,0).move(9.6,0,0)
  mon5 = new Monomer.rot(  0.0, 1,0,0).move(12.8,0,0)
  mon6 = new Monomer.rot(180.0, 1,0,0).move(16.0,0,0)
  mon7 = new Monomer.rot(  0.0, 1,0,0).move(19.2,0,0)
  write("Data Bonds") {
    $bond:backbone1  @bond:2bead/backbone  $atom:mon1/ca  $atom:mon2/ca
    $bond:backbone2  @bond:2bead/backbone  $atom:mon2/ca  $atom:mon3/ca
    $bond:backbone3  @bond:2bead/backbone  $atom:mon3/ca  $atom:mon4/ca
    $bond:backbone4  @bond:2bead/backbone  $atom:mon4/ca  $atom:mon5/ca
    $bond:backbone5  @bond:2bead/backbone  $atom:mon5/ca  $atom:mon6/ca
    $bond:backbone6  @bond:2bead/backbone  $atom:mon6/ca  $atom:mon7/ca
  }
  write_once("In Settings") {
    bond_coeff  @bond:backbone    30.0  3.7
  }
}
```

Must specify the bond parameters for the new bond type

Angles, dihedrals, and impropers

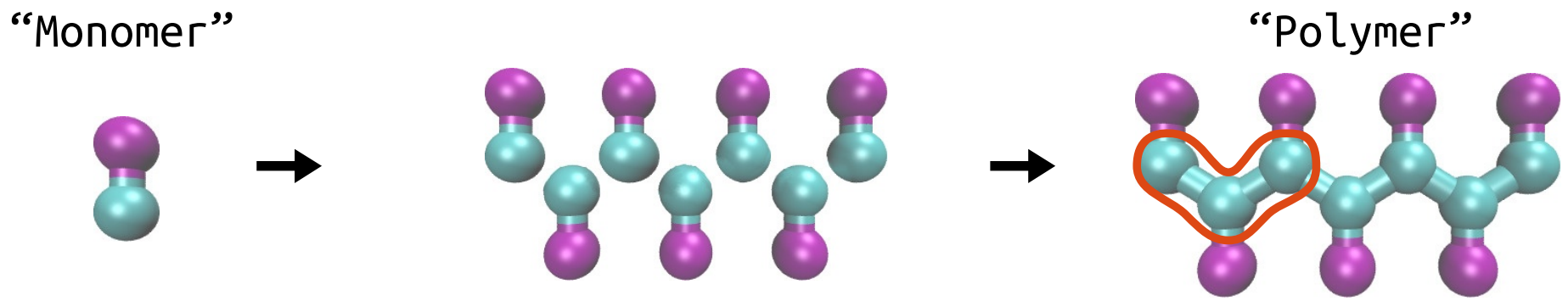
can be generated automatically by atom (& bond) type




```
#  Rules for generating 3 and 4-body bonded ANGULAR interactions by atom type
write_once("Data Angles By Type") {
  @angle:backbone  @atom:CA @atom:CA @atom:CA @bond:* @bond:*
  @angle:sidechain @atom:CA @atom:CA @atom:R  @bond:* @bond:*
}
write_once("Data Dihedrals By Type") {
  @dihedral:back @atom:CA @atom:CA @atom:CA @atom:CA @bond:* @bond:* @bond:*
  @dihedral:side @atom:R  @atom:CA @atom:CA @atom:R  @bond:* @bond:* @bond:*
}
write_once("In Settings") {
  angle_coeff  @angle:backbone   30.00  114
  angle_coeff  @angle:sidechain  30.00  123
}
write_once("In Settings") {
  dihedral_coeff  @dihedral:back   -0.5 1 -180 0.0
  dihedral_coeff  @dihedral:side   -1.5 1 -180 0.0
}
```

Angles, dihedrals, and impropers

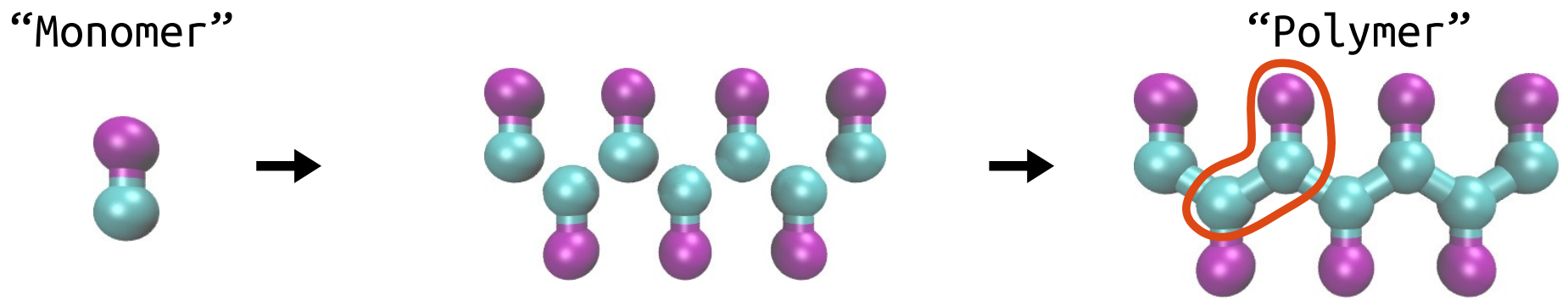
can be generated automatically by atom (& bond) type




```
#  Rules for generating 3 and 4-body bonded ANGULAR interactions by atom type
write_once("Data Angles By Type") {
  @angle:backbone  @atom:CA @atom:CA @atom:CA @bond:* @bond:*
  @angle:sidechain @atom:CA @atom:CA @atom:R  @bond:* @bond:*
}
write_once("Data Dihedrals By Type") {
  @dihedral:back @atom:CA @atom:CA @atom:CA @atom:CA @bond:* @bond:* @bond:*
  @dihedral:side @atom:R  @atom:CA @atom:CA @atom:R  @bond:* @bond:* @bond:*
}
write_once("In Settings") {
  angle_coeff  @angle:backbone   30.00  114
  angle_coeff  @angle:sidechain  30.00  123
}
write_once("In Settings") {
  dihedral_coeff  @dihedral:back   -0.5 1 -180 0.0
  dihedral_coeff  @dihedral:side   -1.5 1 -180 0.0
}
```

Angles, dihedrals, and impropers

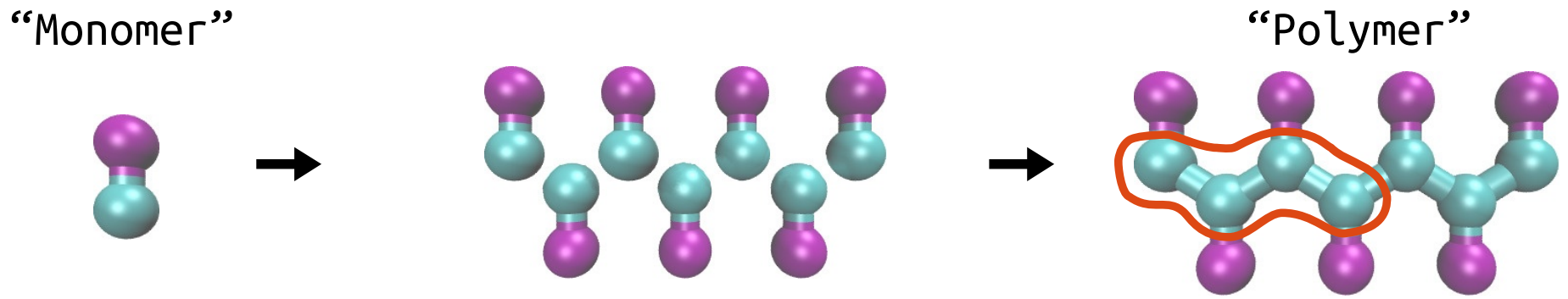
can be generated automatically by atom (& bond) type




```
#  Rules for generating 3 and 4-body bonded ANGULAR interactions by atom type
write_once("Data Angles By Type") {
  @angle:backbone  @atom:CA @atom:CA @atom:CA @bond:* @bond:*
  @angle:sidechain @atom:CA @atom:CA @atom:R  @bond:* @bond:*
}
write_once("Data Dihedrals By Type") {
  @dihedral:back @atom:CA @atom:CA @atom:CA @atom:CA @bond:* @bond:* @bond:*
  @dihedral:side @atom:R  @atom:CA @atom:CA @atom:R  @bond:* @bond:* @bond:*
}
write_once("In Settings") {
  angle_coeff  @angle:backbone   30.00  114
  angle_coeff  @angle:sidechain  30.00  123
}
write_once("In Settings") {
  dihedral_coeff  @dihedral:back   -0.5 1 -180 0.0
  dihedral_coeff  @dihedral:side   -1.5 1 -180 0.0
}
```

Angles, dihedrals, and impropers

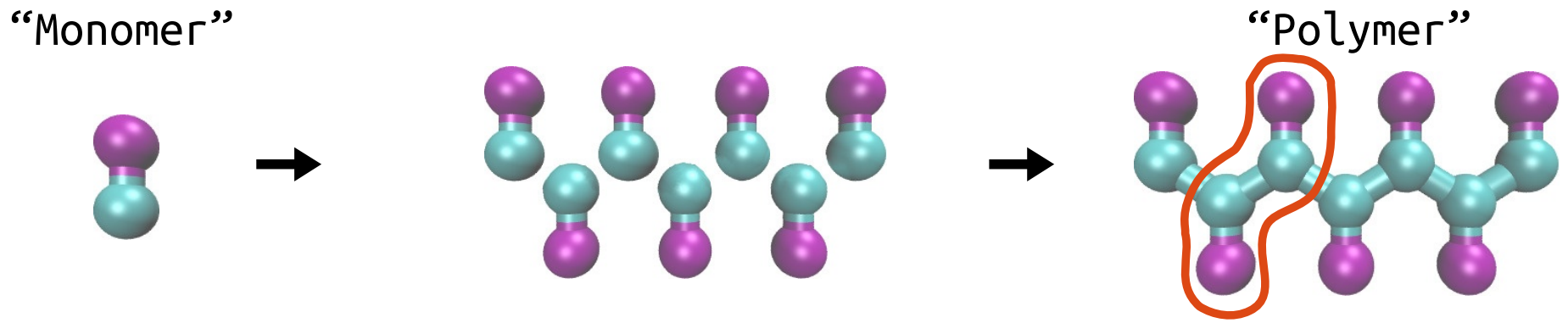
can be generated automatically by atom (& bond) type




```
#  Rules for generating 3 and 4-body bonded ANGULAR interactions by atom type
write_once("Data Angles By Type") {
  @angle:backbone  @atom:CA @atom:CA @atom:CA @bond:* @bond:*
  @angle:sidechain @atom:CA @atom:CA @atom:R  @bond:* @bond:*
}
write_once("Data Dihedrals By Type") {
  @dihedral:back @atom:CA @atom:CA @atom:CA @atom:CA @bond:* @bond:* @bond:*
  @dihedral:side @atom:R  @atom:CA @atom:CA @atom:R  @bond:* @bond:* @bond:*
}
write_once("In Settings") {
  angle_coeff  @angle:backbone   30.00  114
  angle_coeff  @angle:sidechain  30.00  123
}
write_once("In Settings") {
  dihedral_coeff  @dihedral:back   -0.5 1 -180 0.0
  dihedral_coeff  @dihedral:side   -1.5 1 -180 0.0
}
```

Angles, dihedrals, and impropers

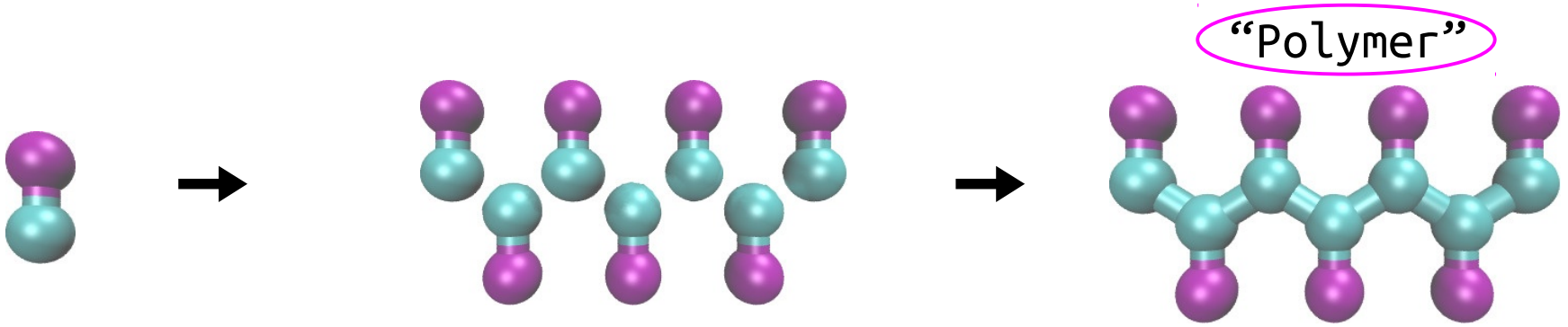
can be generated automatically by atom (& bond) type

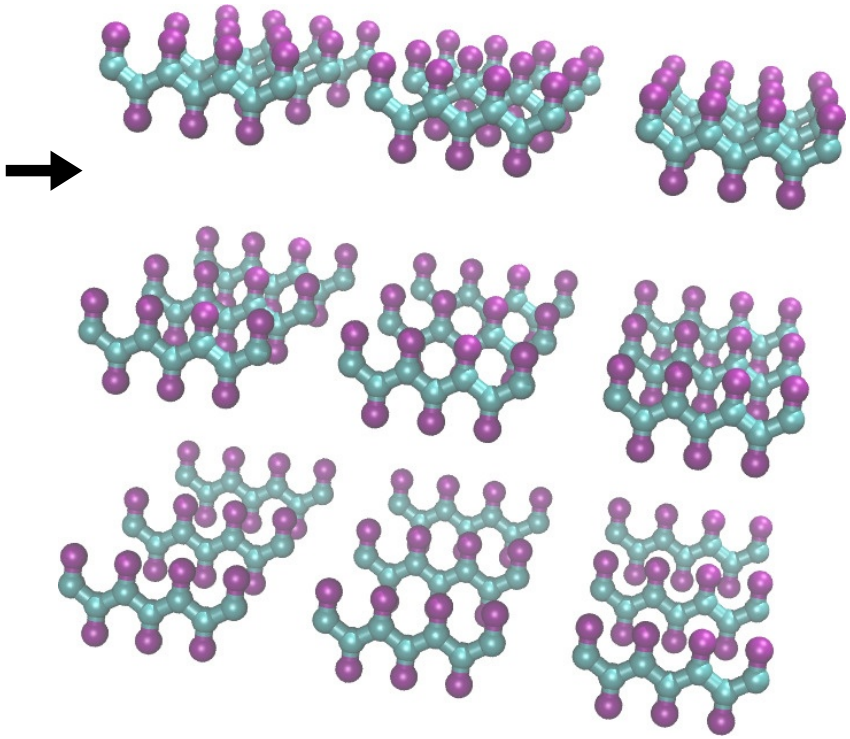



```
#  Rules for generating 3 and 4-body bonded ANGULAR interactions by atom type
write_once("Data Angles By Type") {
  @angle:backbone  @atom:CA @atom:CA @atom:CA @bond:* @bond:*
  @angle:sidechain @atom:CA @atom:CA @atom:R  @bond:* @bond:*
}
write_once("Data Dihedrals By Type") {
  @dihedral:back @atom:CA @atom:CA @atom:CA @atom:CA @bond:* @bond:* @bond:*
  @dihedral:side @atom:R  @atom:CA @atom:CA @atom:R  @bond:* @bond:* @bond:*
}
write_once("In Settings") {
  angle_coeff  @angle:backbone   30.00  114
  angle_coeff  @angle:sidechain  30.00  123
}
write_once("In Settings") {
  dihedral_coeff  @dihedral:back   -0.5 1 -180 0.0
  dihedral_coeff  @dihedral:side   -1.5 1 -180 0.0
}
```

For fun: make many copies

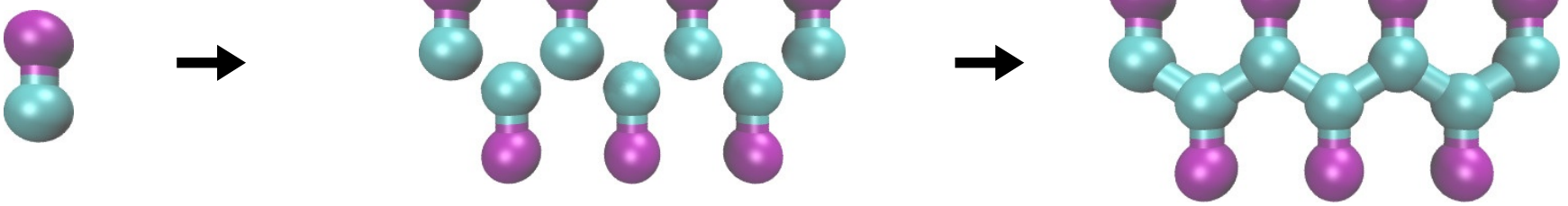
Create a 3x3x3 3D array of Polymers

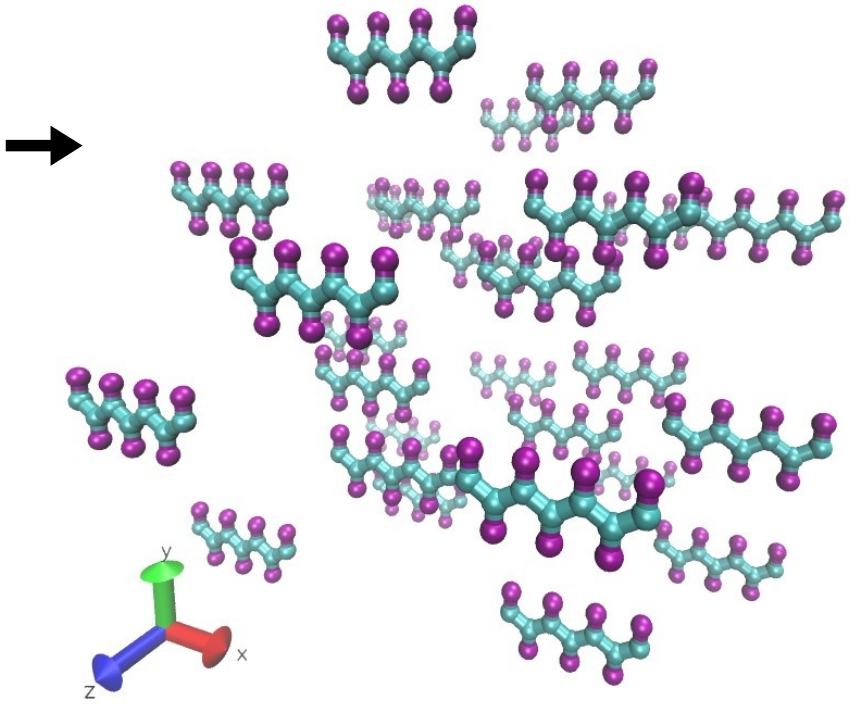




```
polymers = new Polymer [3].move(0, 0, 30.0)
                       [3].move(0, 30.0, 0)
                       [3].move(30.0, 0, 0)
```

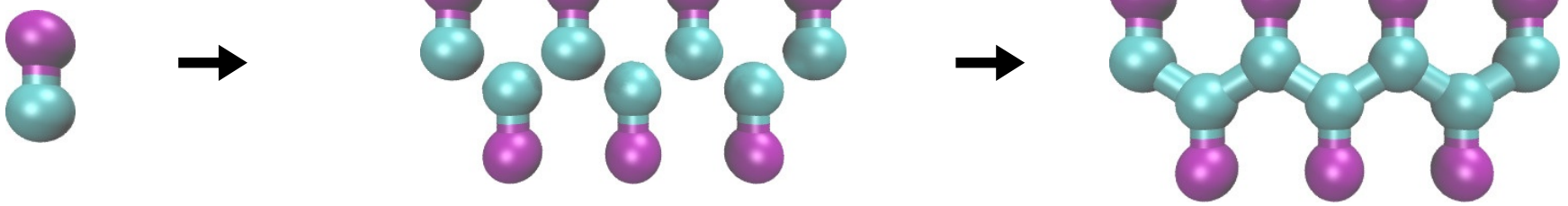
You can move (and modify) molecules after you create them:

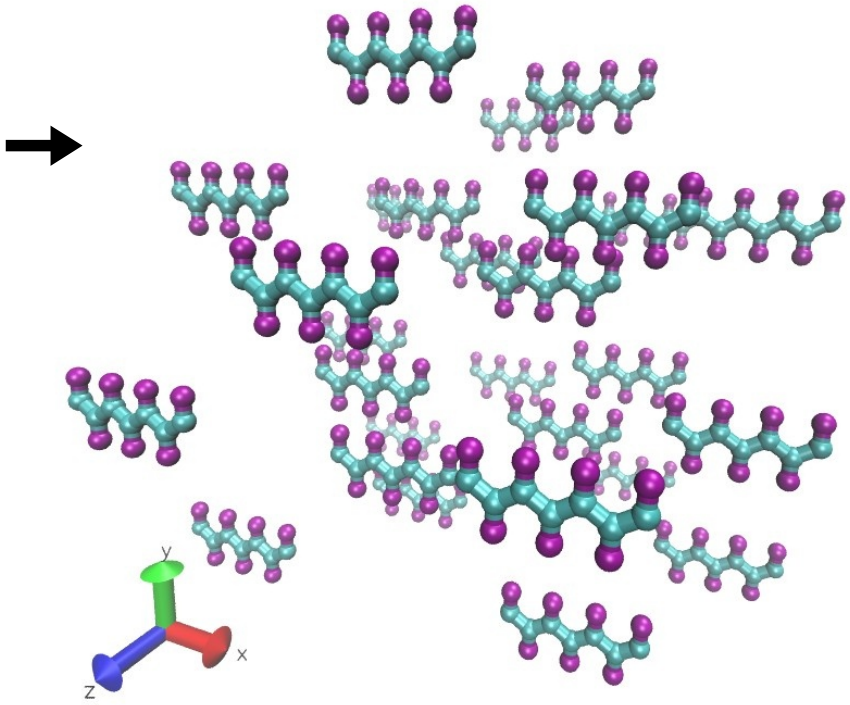





```
polymers = new Polymer [3].move(0, 0, 30.0)
                       [3].move(0, 30.0, 0)
                       [3].move(30.0, 0, 0)

polymers[1][*][*].move(20,0,0)
polymers[*][1][*].move(0,0,20)
polymers[*][*][1].move(0,20,0)
```

You can move (and modify) molecules after you create them:

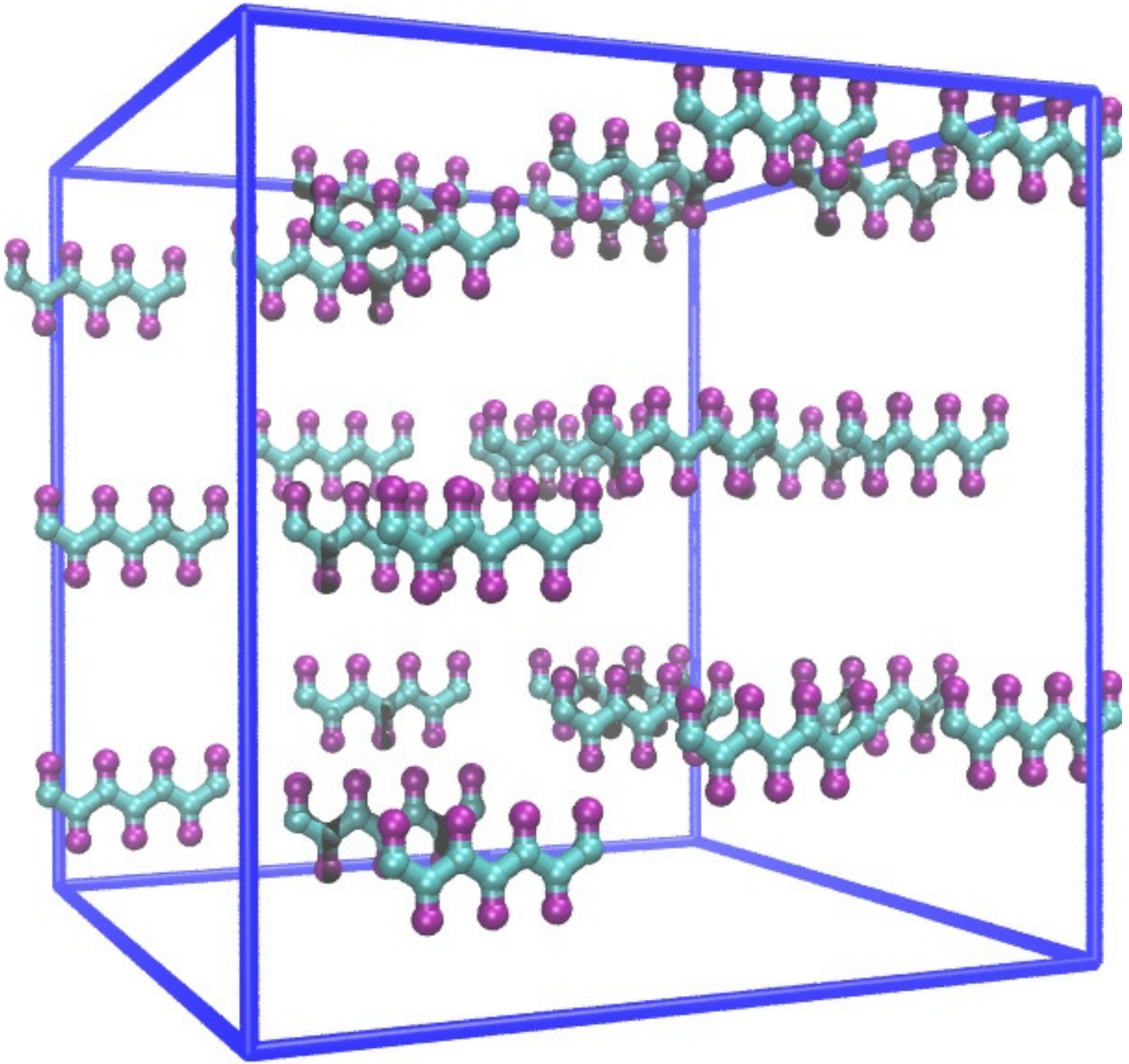




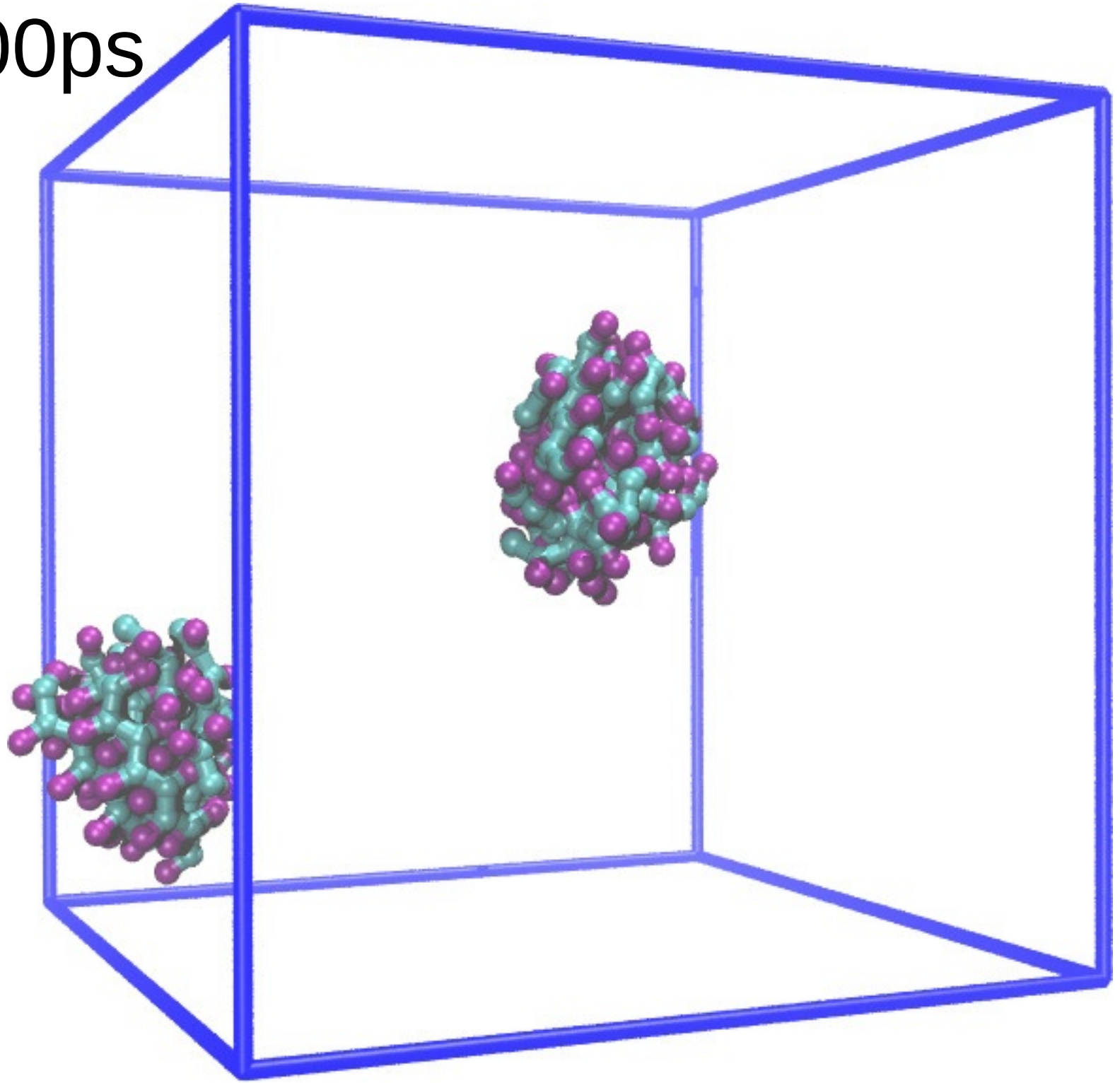

```
polymers = new Polymer [3].move(0, 0, 30.0)
                       [3].move(0, 30.0, 0)
                       [3].move(30.0, 0, 0)

polymers[1][*][*].move(20,0,0)
polymers[*][1][*].move(0,0,20)
polymers[*][*][1].move(0,20,0)
```

The [*] enables you to move multiple molecules at the same time

t = 0

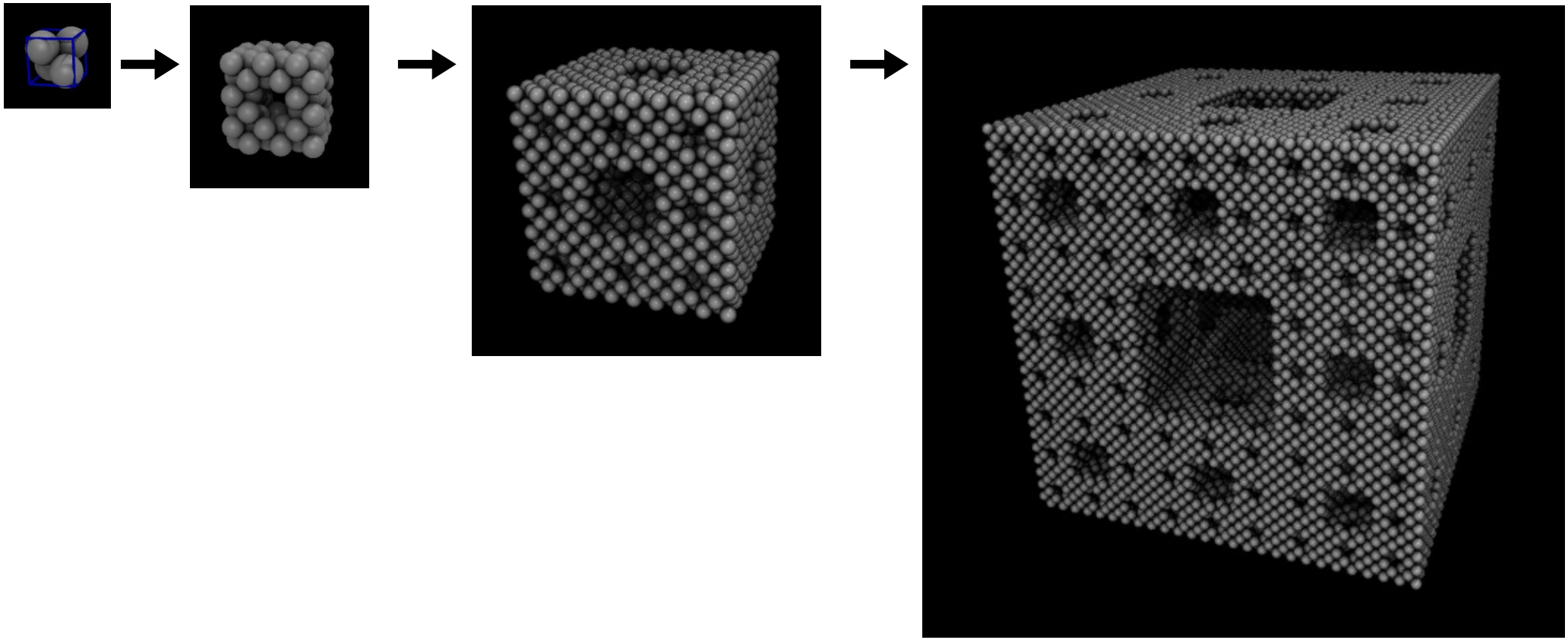


t = 200ps



Object composition

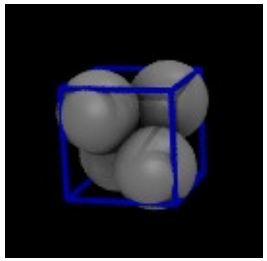
Objects can be built from smaller objects



Object composition

Objects can be built from smaller objects

AlCell

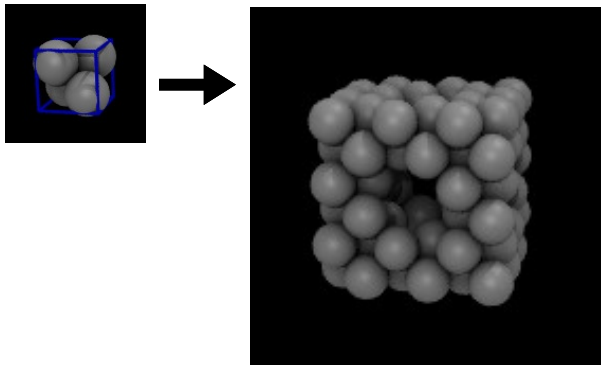


```
# "AlCell" defines the 4-atom FCC unit cell
# of Aluminum (with a 4.05 angstrom spacing)

AlCell
{
  # AtomID     AtomType  Charge    X     Y     Z

  write("Data Atoms") {
    $atom:AlC  @atom:Al    0.0   0.000 0.000 0.000
    $atom:AlX  @atom:Al    0.0   0.000 2.025 2.025
    $atom:AlY  @atom:Al    0.0   2.025 0.000 2.025
    $atom:AlZ  @atom:Al    0.0   2.025 2.025 0.000
  }

  write_once("In Settings") {
    pair_coeff  * *  eam/alloy  Al99.eam.alloy Al
  }

  write_once("Data Masses") {
    @atom:Al 27.0
  }

}  # AlCell
```

Object composition

Objects can be built from smaller objects

AlCell *SierpinskiCubeLvl1*

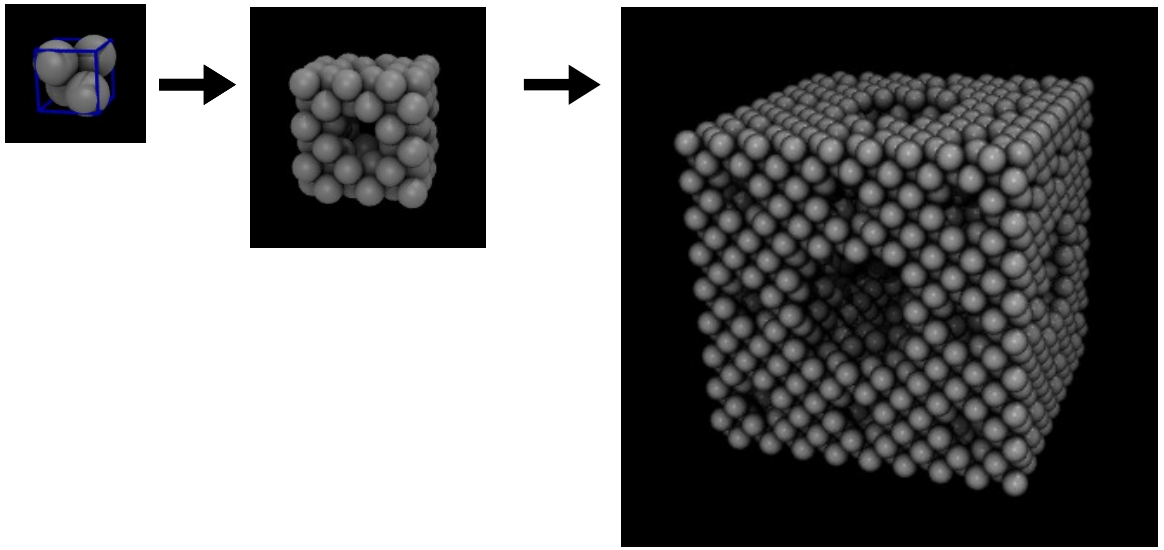


```
SierpinskiCubeLvl1
{
  cells = new AlCell [3].move(0.00, 0.00, 4.05)
                        [3].move(0.00, 4.05, 0.00)
                        [3].move(4.05, 0.00, 0.00)
  delete cells[*][1][1]
  delete cells[1][*][1]
  delete cells[1][1][*]
}
```

Object composition

Objects can be built from smaller objects

AlCell *SierpinskiCubeLvl1* *SierpinskiCubeLvl2*

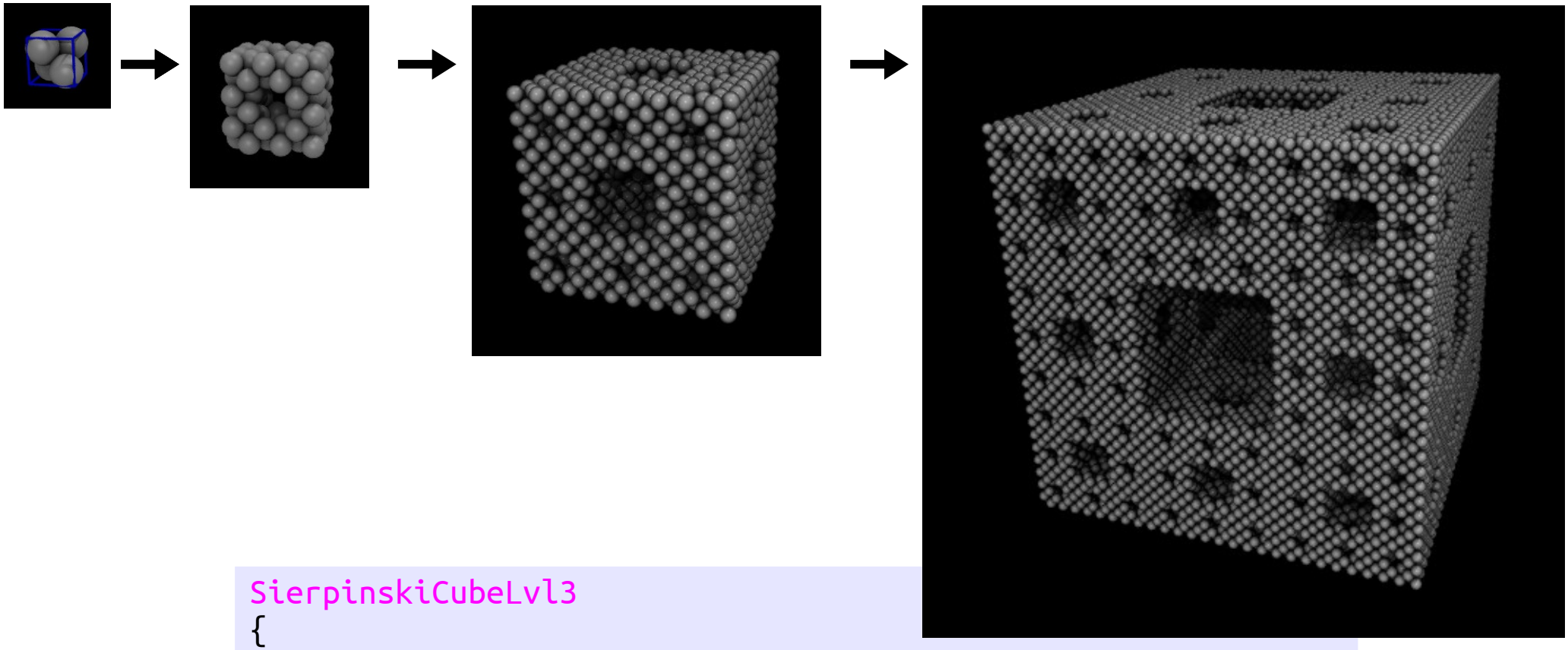


```
SierpinskiCubeLvl2
{
  cells = new SierpinskiCubeLvl1 [3].move(0.00, 0.00, 12.15)
                                 [3].move(0.00, 12.15, 0.00)
                                 [3].move(12.15, 0.00, 0.00)

  delete cells[*][1][1]
  delete cells[1][*][1]
  delete cells[1][1][*]
}
```

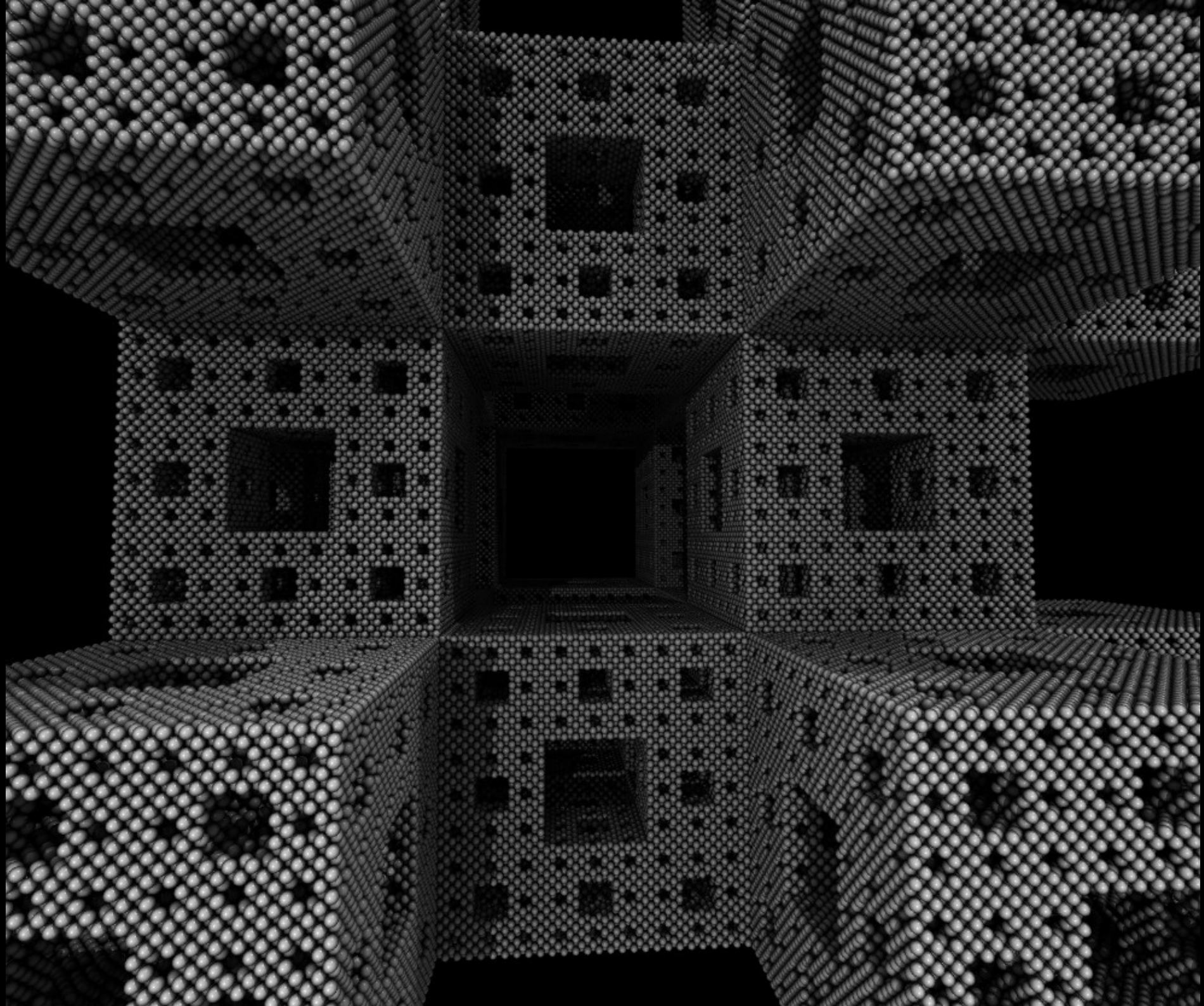
Object composition

Objects can be built from smaller objects

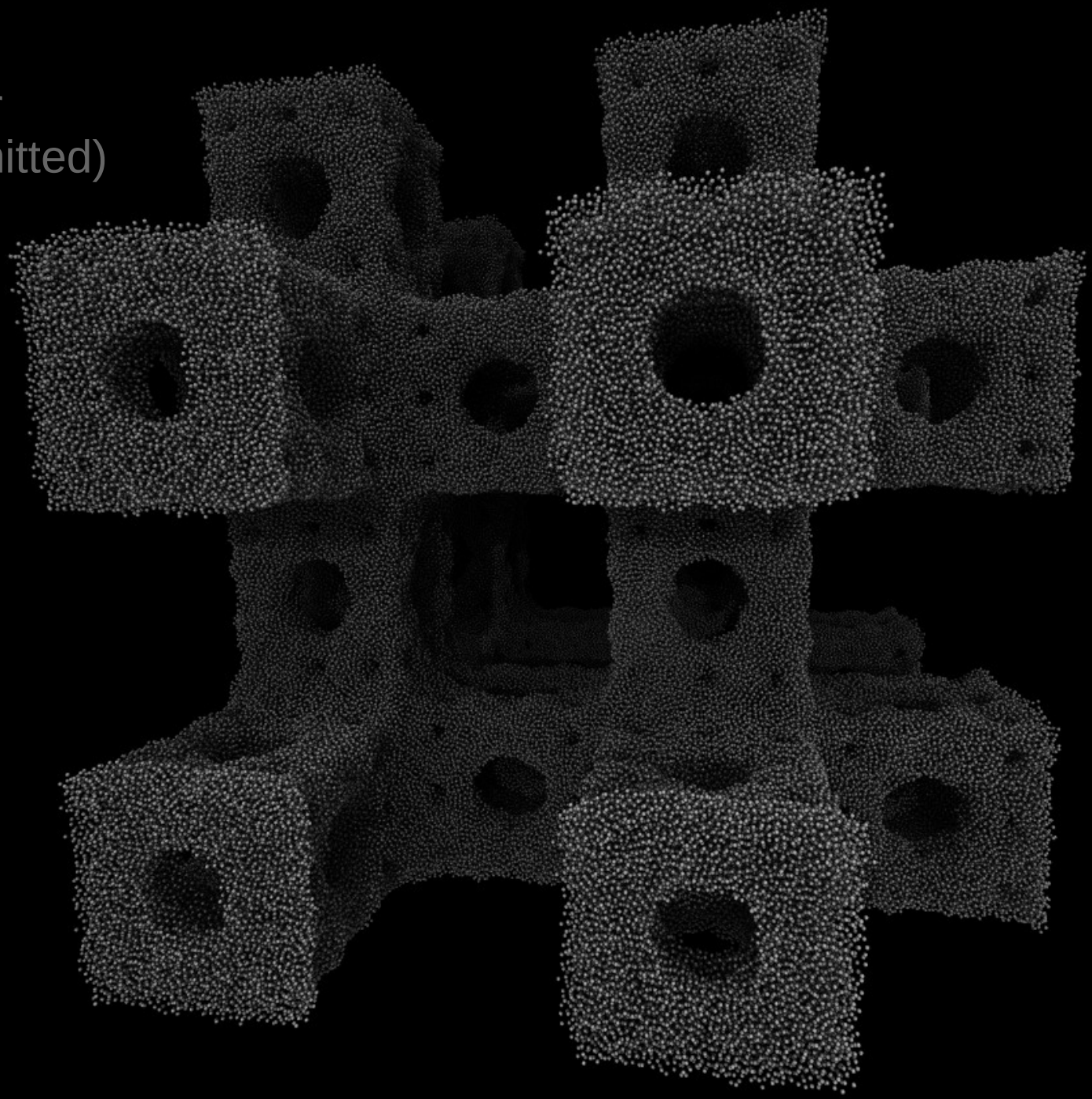
AlCell *SierpinskiCubeLvl1* *SierpinskiCubeLvl2* *SierpinskiCubeLvl3*



```
SierpinskiCubeLvl3
{
  cells = new SierpinskiCubeLvl2 [3].move(0.00, 0.00, 36.45)
                                 [3].move(0.00, 36.45, 0.00)
                                 [3].move(36.45, 0.00, 0.00)

  delete cells[*][1][1]
  delete cells[1][*][1]
  delete cells[1][1][*]
}
```



7400
steps later
(movie omitted)



Moltemplate is not only for coarse-grained models.

Moltemplate can be used to set up all-atom simulations as well.

Moltemplate is not only for coarse-grained models.

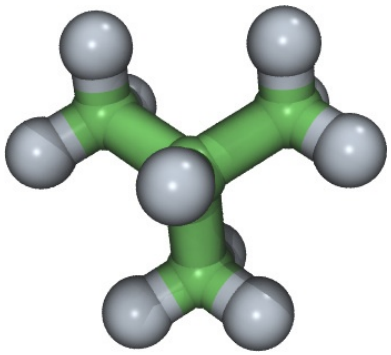
Moltemplate can be used to set up all-atom simulations as well.

Moltemplate objects can also be used to store force-fields (independent of the molecules associated with them)

Canned force fields
are available for LAMMPS

(experimental feature as of 2013-8)

Moltemplate encapsulates ***force-fields***



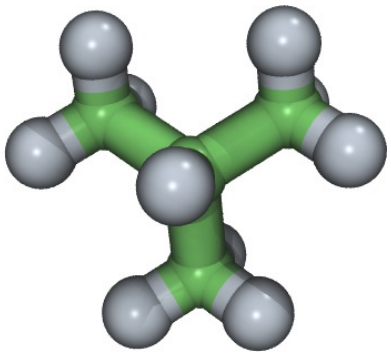
Isobutane

isubutane.lt

```
import "gaff.lt"
Isobutane inherits GAFF {
  write("Data Atoms") {
    $atom:C0  $mol  @atom:c3   0.0  -0.001  -0.001  -0.439
    $atom:C1  $mol  @atom:c3   0.0  -1.257  -0.726   0.078
    $atom:H0  $mol  @atom:h1   0.0  -0.003  -0.004  -1.439
    $atom:H11 $mol  @atom:h1   0.0  -2.075  -0.255  -0.254 ...}
  write("Data Bond List") {
    $bond:C01 $atom:C0 $atom:C1
    $bond:C0H $atom:C0 $atom:H0
    $bond:C1H1 $atom:C1 $atom:H11 ...}
}
```

A canned force-field allows you to omit all details except for the atom types, and bond topology, *charges*, and coordinates. In this example, moltemplate will generate everything else according to AMBER (GAFF) conventions. (**Note: moltemplate does not currently calculate atom charge.)**

Moltemplate encapsulates ***force-fields***



Isobutane

isubutane.lt

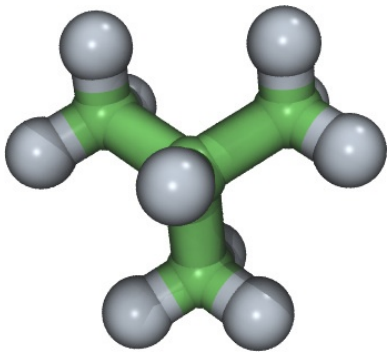
```
import "charmm.lt"
Isobutane inherits CHARMM {
  write("Data Atoms") {
    $atom:C0  $mol  @atom:c3   0.0  -0.001  -0.001  -0.439
    $atom:C1  $mol  @atom:c3   0.0  -1.257  -0.726   0.078
    $atom:H0  $mol  @atom:h1   0.0  -0.003  -0.004  -1.439
    $atom:H11 $mol  @atom:h1   0.0  -2.075  -0.255  -0.254 ...}
  write("Data Bond List") {
    $bond:C01 $atom:C0 $atom:C1
    $bond:C0H $atom:C0 $atom:H0
    $bond:C1H1 $atom:C1 $atom:H11 ...}
}
```

Swapping a different force field requires changing only two lines of your moltemplate LT file.

(However, as of 2013-8-07, only GAFF is supported. Hopefully I will add more eventually. Community help with testing is appreciated.)

Moltemplate encapsulates ***force-fields***

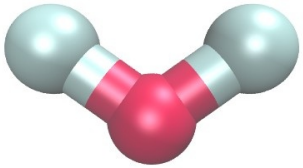
Test example: Hydrocarbon phase-separation in water



Isobutane

isubutane.lt

```
import "gaff.lt"
Isobutane inherits GAFF {
  write("Data Atoms") {
    $atom:C0  $mol  @atom:c3   0.0  -0.001  -0.001  -0.439
    $atom:C1  $mol  @atom:c3   0.0  -1.257  -0.726   0.078
    $atom:H0  $mol  @atom:h1   0.0  -0.003  -0.004  -1.439
    $atom:H11 $mol  @atom:h1   0.0  -2.075  -0.255  -0.254 ...}
  write("Data Bond List") {
    $bond:C01 $atom:C0 $atom:C1
    $bond:C0H $atom:C0 $atom:H0
    $bond:C1H1 $atom:C1 $atom:H11 ...}
}
```



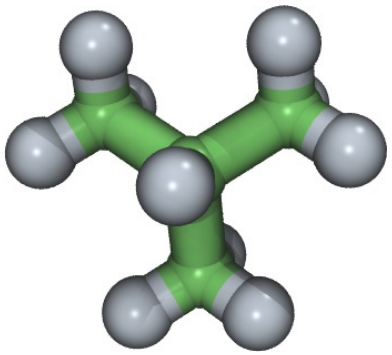
water (tip3p)

tip3p.lt

```
TIP3P {
  write("Data Atoms") {
    $atom:O $mol @atom:ow  -0.830   0.000 0.000 0.000
    $atom:H1 $mol @atom:hw  0.415   0.757 0.000 0.586
    $atom:H2 $mol @atom:hw  0.415  -0.757 0.000 0.586
  }
  Details omitted...
}
```

Moltemplate encapsulates ***force-fields***

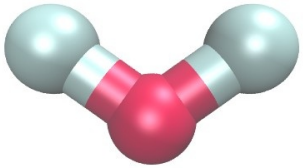
Example: Hydrocarbon phase-separation in water



Isobutane

isubutane.lt

```
import "gaff.lt"
Isobutane inherits GAFF {
  write("Data Atoms") {
    $atom:C0  $mol  @atom:c3   0.0  -0.001  -0.001  -0.439
    $atom:C1  $mol  @atom:c3   0.0  -1.257  -0.726   0.078
    $atom:H0  $mol  @atom:h1   0.0  -0.003  -0.004  -1.439
    $atom:H11 $mol  @atom:h1   0.0  -2.075  -0.255  -0.254 ...}
  write("Data Bond List") {
    $bond:C01 $atom:C0 $atom:C1
    $bond:C0H $atom:C0 $atom:H0
    $bond:C1H1 $atom:C1 $atom:H11 ...}
}
```



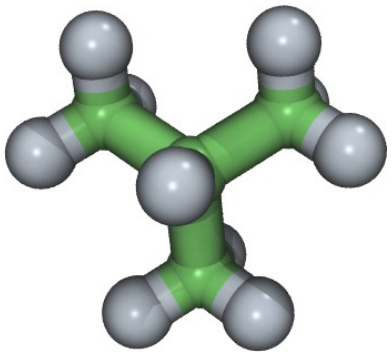
water (tip3p)

tip3p.lt

```
TIP3P {
  write("Data Atoms") {
    $atom:O $mol @atom:ow  -0.830   0.000 0.000 0.000
    $atom:H1 $mol @atom:hw  0.415   0.757 0.000 0.586
    $atom:H2 $mol @atom:hw  0.415  -0.757 0.000 0.586
  }
  Details omitted...
}
```

Moltemplate encapsulates ***force-fields***

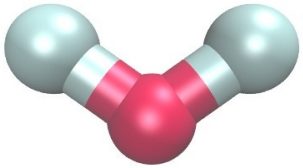
Example: Hydrocarbon phase-separation in water



Isobutane

isubutane.lt

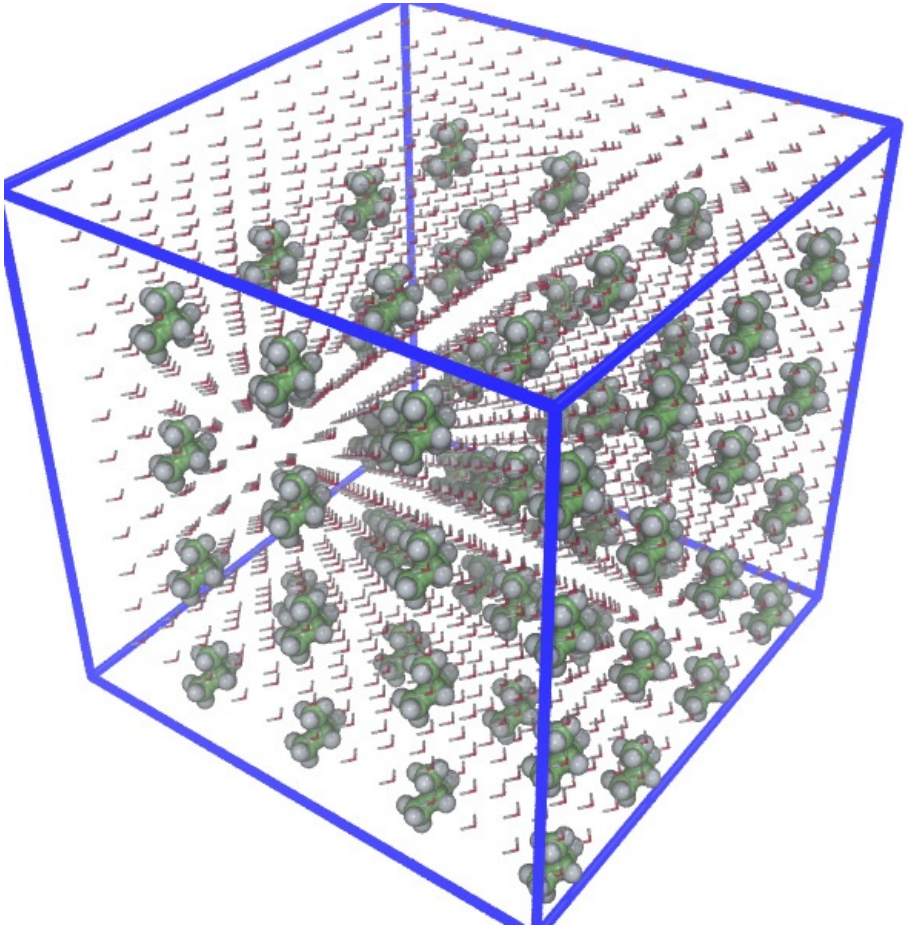
```
import "gaff.lt"
Isobutane inherits GAFF {
  write("Data Atoms") {
    $atom:C0  $mol  @atom:c3   0.0  -0.001  -0.001  -0.439
    $atom:C1  $mol  @atom:c3   0.0  -1.257  -0.726   0.078
    $atom:H0  $mol  @atom:h1   0.0  -0.003  -0.004  -1.439
    $atom:H11 $mol  @atom:h1   0.0  -2.075  -0.255  -0.254 ...}
  write("Data Bond List") {
    $bond:C01 $atom:C0 $atom:C1
    $bond:C0H $atom:C0 $atom:H0
    $bond:C1H1 $atom:C1 $atom:H11 ...}
}
```



water (tip3p)

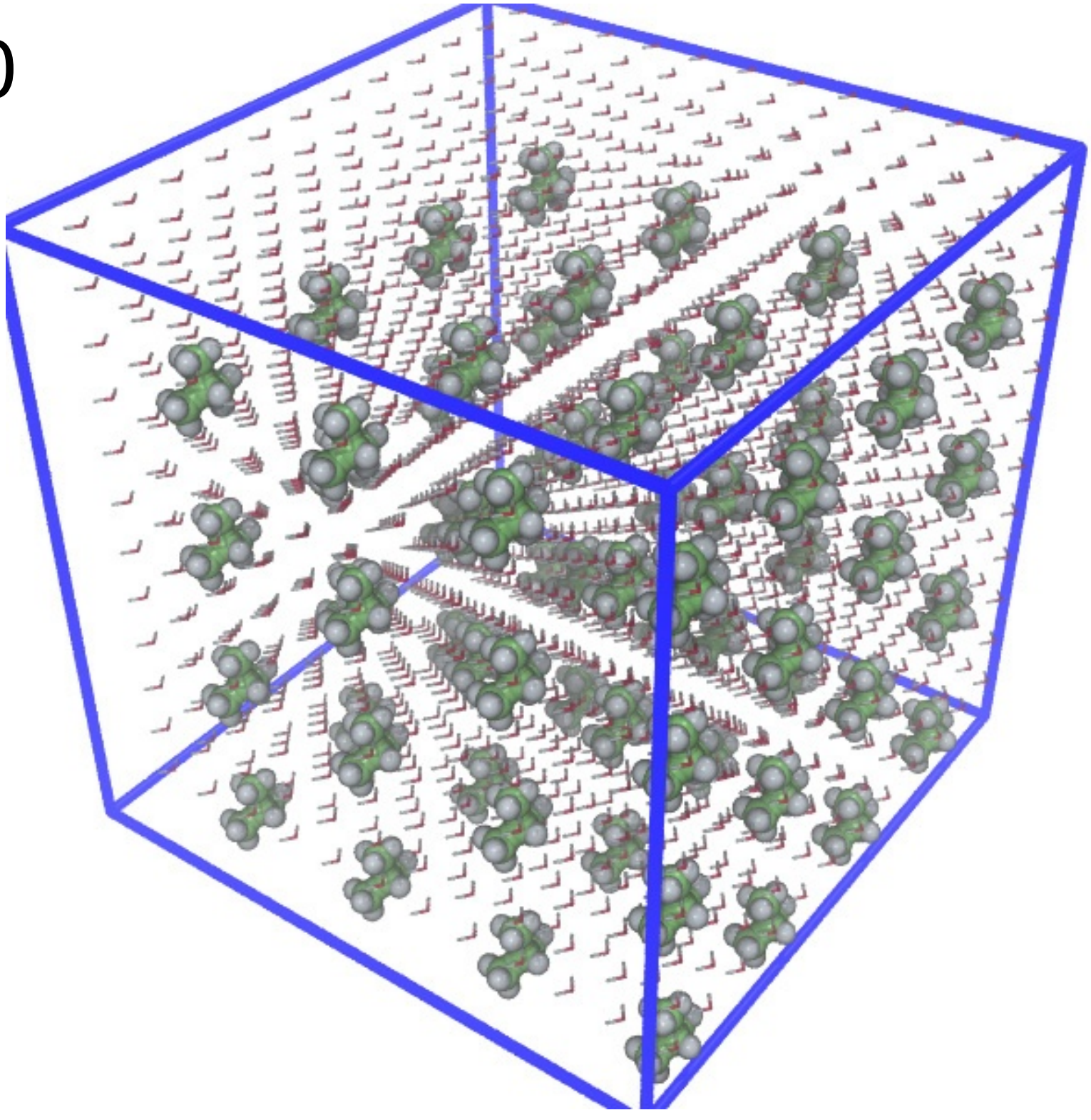
tip3p.lt

```
TIP3P {
  write("Data Atoms") {
    $atom:O $mol @atom:ow  -0.830   0.000 0.000 0.000
    $atom:H1 $mol @atom:hw  0.415   0.757 0.000 0.586
    $atom:H2 $mol @atom:hw  0.415  -0.757 0.000 0.586
  }
  Details omitted...
}
```



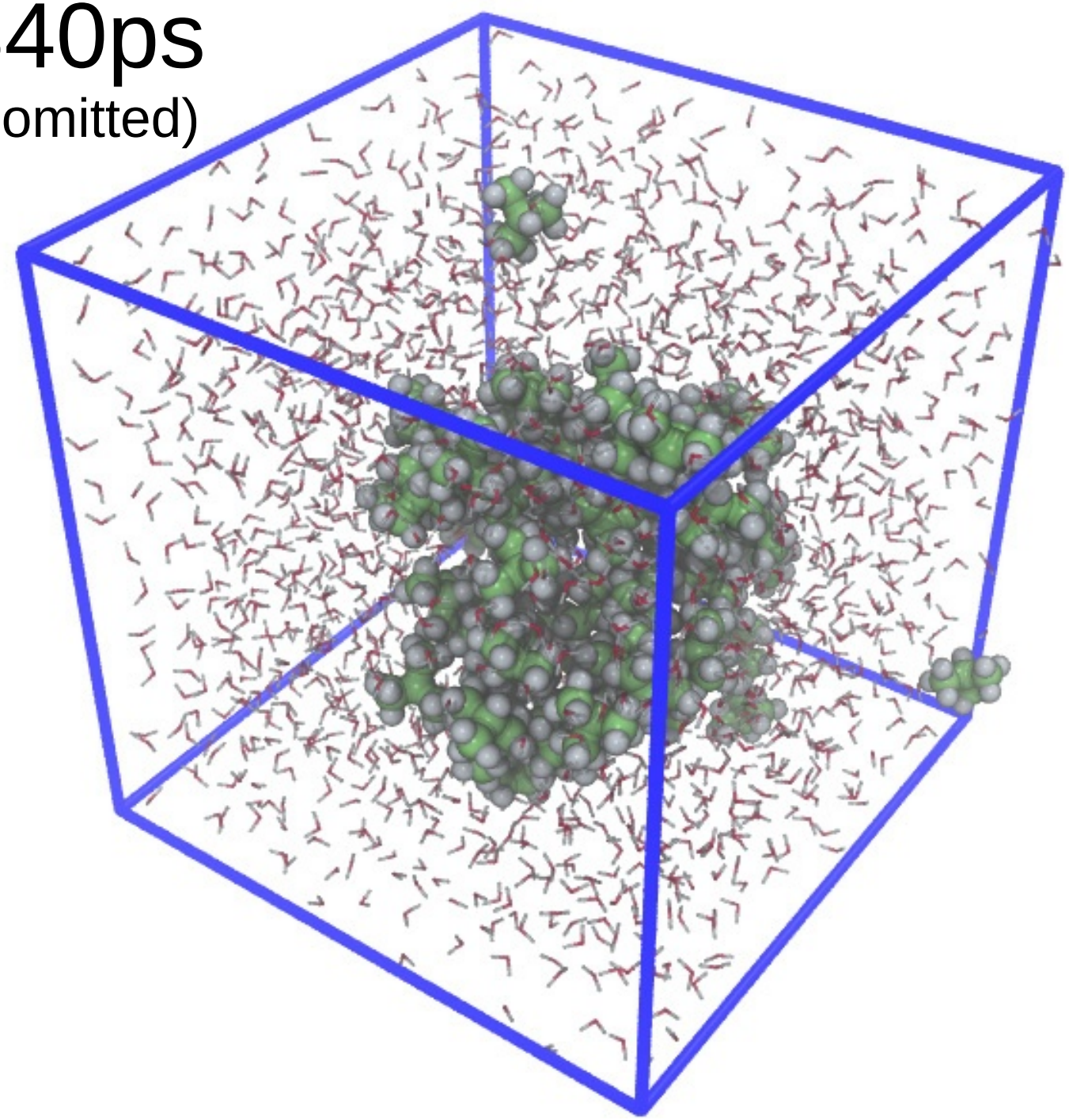
```
import "tip3p.lt"
import "isobutane.lt"

wat  = new TIP3P [12].move(0.00, 0.00, 3.45)
                 [12].move(0.00, 3.45, 0.00)
                 [12].move(3.45, 0.00, 0.00)

isobutanes = new Isobutane [4].move(0, 0, 10.35)
                           [4].move(0, 10.35, 0)
                           [4].move(10.35, 0, 0)

# move the isobutane molecules slightly to reduce overlap with the water
isobutanes[*][*][*].move(1.725, 1.725, 1.725)
```

t = 0

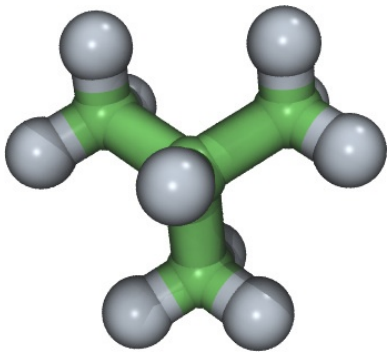


t = 840ps

(movie omitted)



The “gaff.lt” file contains force-field information and rules for generating angle, dihedral, and improper interactions



Isobutane

isubutane.lt

```
import “gaff.lt”
Isobutane inherits GAFF {
  write('Data Atoms') {
    $atom:C0  $mol  @atom:c3   0.0  -0.001  -0.001  -0.439
    $atom:C1  $mol  @atom:c3   0.0  -1.257  -0.726   0.078
    $atom:H0  $mol  @atom:h1    0.0  -0.003  -0.004  -1.439
    $atom:H11 $mol  @atom:h1   0.0  -2.075  -0.255  -0.254 ...}
  write('Data Bond List') {
    $bond:C01 $atom:C0 $atom:C1
    $bond:C0H $atom:C0 $atom:H0
    $bond:C1H1 $atom:C1 $atom:H11 ...}
}
```

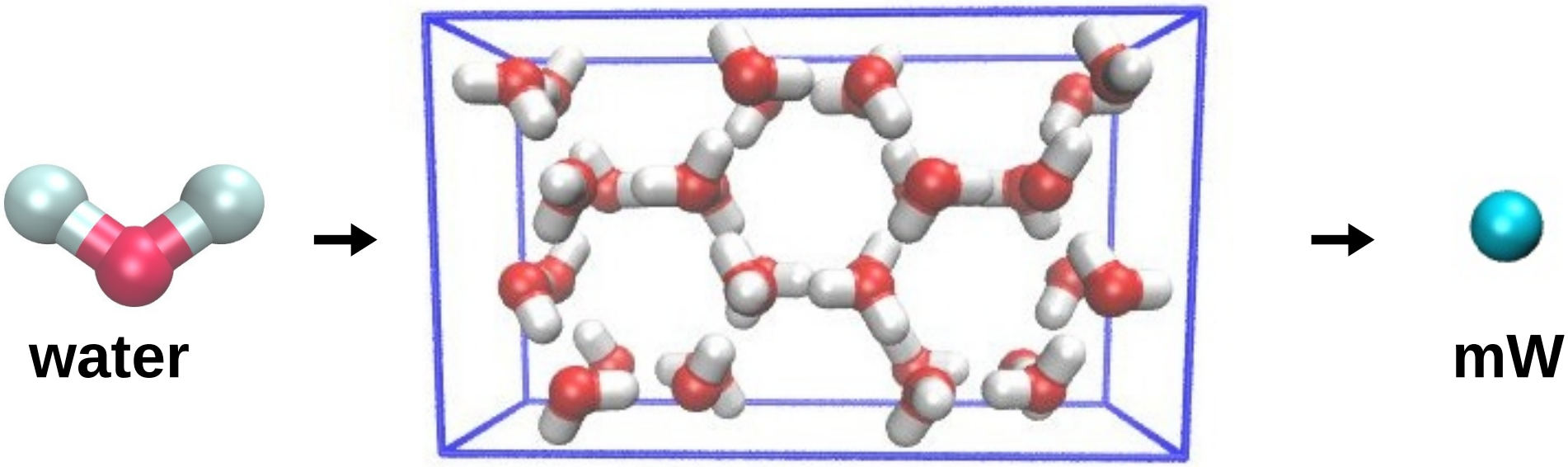
Contents of the “gaff.lt” file: (excerpts)

gaff.lt

```
GAFF {
  write_once("Data Masses") {
    @atom:c1 12.01   # Sp C
    @atom:h1 1.008   # H bonded to aliphatic carbon with 1 electrwd. Group ...
  }
  write_once("In Settings") {
    pair_coeff @atom:c1 @atom:c1 lj/charmm/coul/long 0.2100 1.9080
    pair_coeff @atom:h1 @atom:h1 lj/charmm/coul/long 0.0157 1.3870  # Veenstra JCC,8,963...
  }
  write_once("In Settings") {
    bond_coeff @bond:c2-o harmonic 623.6 1.2244   # SOURCE4 15 0.0036
    bond_coeff @bond:ch-nf harmonic 509.5 1.3262  # SOURCE4 17 same_as_cg-ne ...
  }
  write_once("Data Bonds By Type") {
    @bond:c2-o  @atom:c2 @atom:o
    @bond:ch-nf @atom:ch @atom:nf ...
  }
  write_once("In Settings") {
    angle_coeff @angle:cg-c1-ha harmonic 43.980 177.410   # SOURCE3 22 2.4947
    angle_coeff @angle:c2-c-s harmonic 81.850 119.160   # SOURCE3 2 0.0000 ...
  }
  write_once("Data Angles By Type") {
    @angle:cg-c1-ha @atom:cg @atom:c1 @atom:ha
    @angle:c2-c-s @atom:c2 @atom:c @atom:s  ...
  }
  # Dihedral and Improper similar to Angle (details omitted)
  write_once("In Settings") { dihedral_coeff ...}
  write_once("Data Dihedrals By Type") { ... }
  write_once("In Settings") { improper_coeff ...}
  write_once("Data Impropers By Type") { ... }
}
```

(charge-assignment not included)

New example: coarse-grained hydrocarbons in coarse-grained water



The “mW” water model has only 1 particle per water molecule and uses non-pairwise-additive forces to mimic hydrogen bonding.*(Molinero et al, J. Phys. Chem. B 2009, 113, 4008-4016)*

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

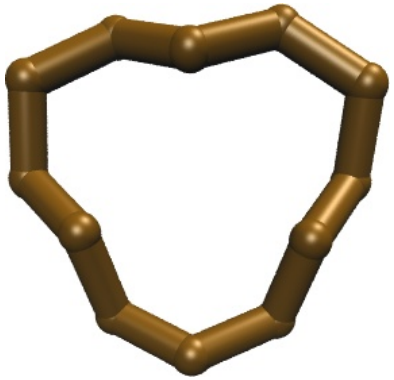
LAMMPS modularity enbables combining different force-field types

Example: Hydrocarbon phase-separation in water

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

V. Molinero, E.B. Moore, J. Phys. Chem. B 2009, 113, 4008-4016



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

Different force-fields or features use different file formats

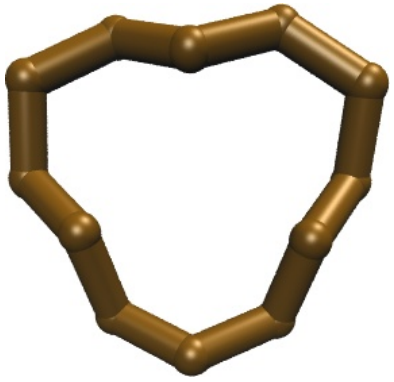
(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL
```

```
mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

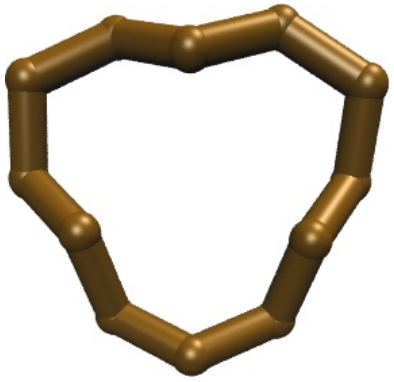
Different force-fields or features use different file formats

(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL

mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

Different force-fields or features use different file formats

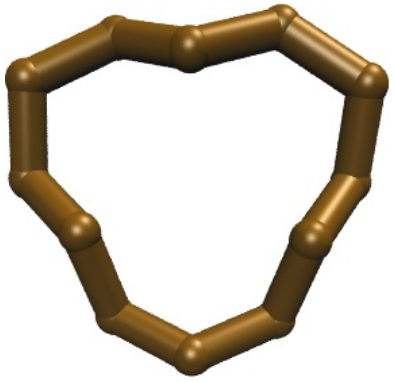
(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL
```

```
mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

Different force-field *parameters* are specified using different syntax

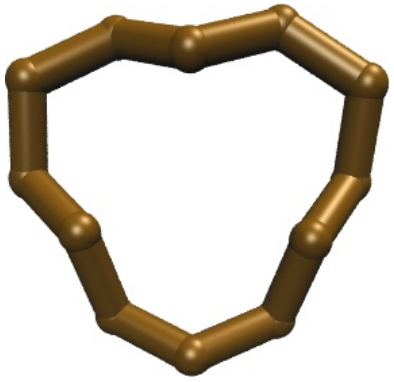
(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL
```

```
mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522  3.95
pair_coeff  3 3  lj/cut  0.194746286  3.75
```

Different force-field *parameters* are specified using different syntax

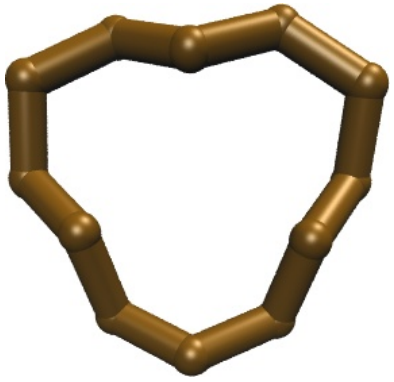
(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL
```

```
mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

Different force-field *parameters* are specified using different syntax

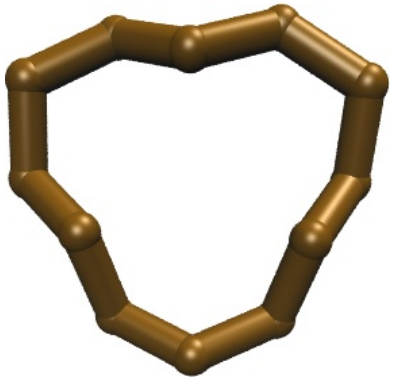
(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL
```

```
mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

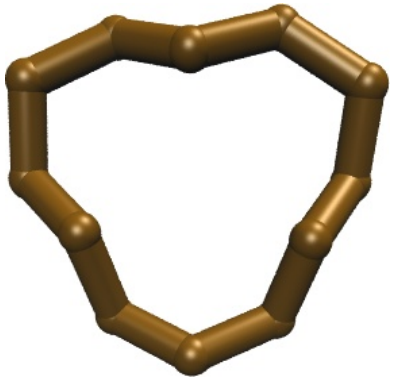
Different force-field *parameters* are specified using different syntax

(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL

mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

Atoms & atom-types are referenced using different syntax rules

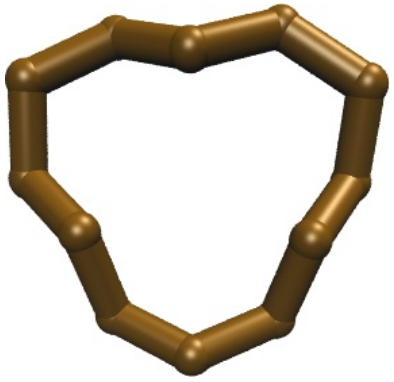
(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL
```

```
mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

Atoms & atom-types are referenced using different syntax rules

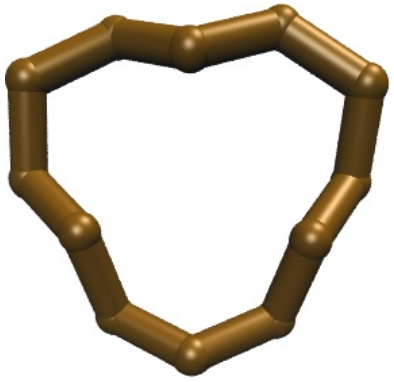
(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL
```

```
mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

```
pair_coeff  2 2  lj/cut  0.091411522 3.95
pair_coeff  3 3  lj/cut  0.194746286 3.75
```

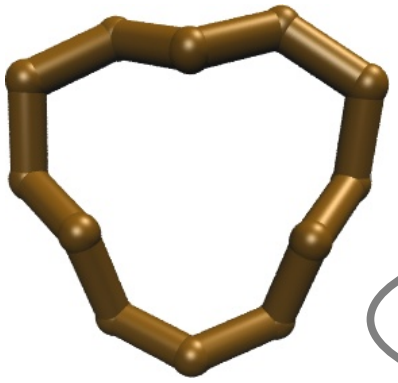
original LAMMPS format:

(LAMMPS format)

3-body

$$E_{nb} = \sum_{ijk} \varphi_{ijk}(\mathbf{r}_i, \mathbf{r}_j, \mathbf{r}_k)$$

```
pair_coeff  * * sw system.in.sw mW NULL NULL NULL

mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333333
7.049556277 0.602224558 4 0 0
```



2-body

$$E_{nb} = \sum_{ij} 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]$$

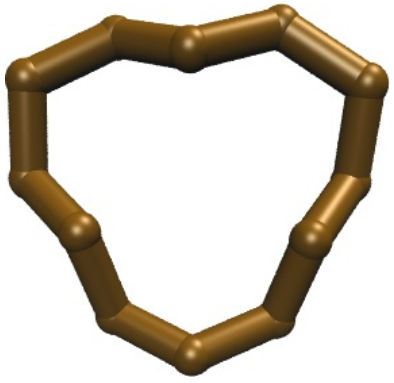
```
pair_coeff 2 2 lj/cut 0.091411522 3.95
pair_coeff 3 3 lj/cut 0.194746286 3.75
```

Moltemplate encapsulates molecules (moltemplate format)

3-body

watmw.lt

```
WatMW {
  write("Data Atoms") {$atom:mW $mol @atom:mW 0 0.0 0.0 0.0}

  write_once("system.in.sw") {
    mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333 7.0495562 \
0.602224558 4 0 0
  }
  write_once("In Settings") {
    pair_coeff    * * sw system.in.sw mW NULL NULL NULL
  }
}
```



2-body

cyclododecane.lt

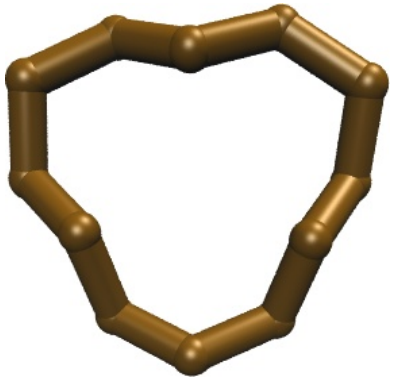
```
Cyclododecane {
  write('Data Atoms') {
    $atom:C1 $mol @atom:CH2  0.0 0.0 2.94118 0.0
    $atom:C2 $mol @atom:CH2  0.0 0.0 2.54714 1.47059 ...}
  write('Data Bonds') {
    $bond:bond1 @bond:TraPPE/saturated $atom:C1 $atom:C2
    $bond:bond2 @bond:TraPPE/saturated $atom:C2 $atom:C3 ...}
  write_once('In Settings') {
    pair_coeff @atom:CH2 @atom:CH2 lj/cut 0.091411522 3.95
    pair_coeff @atom:CH3 @atom:CH3 lj/cut 0.194746286 3.75
  }
}
```

Moltemplate encapsulates molecules (moltemplate format)

3-body

watmw.lt

```
WatMW {
  write("Data Atoms") {$atom:mW $mol @atom:mW 0 0.0 0.0 0.0}

  write_once("system.in.sw") {
    mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333 7.0495562 \
0.602224558 4 0 0
  }
  write_once("In Settings") {
    pair_coeff    * * sw system.in.sw mW NULL NULL NULL
  }
}
```



2-body

cyclododecane.lt

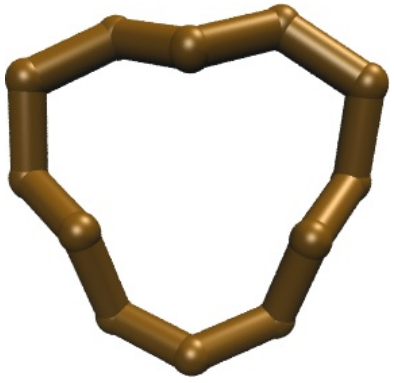
```
Cyclododecane {
  write('Data Atoms') {
    $atom:C1 $mol @atom:CH2  0.0 0.0 2.94118 0.0
    $atom:C2 $mol @atom:CH2  0.0 0.0 2.54714 1.47059 ...}
  write('Data Bonds') {
    $bond:bond1 @bond:TraPPE/saturated $atom:C1 $atom:C2
    $bond:bond2 @bond:TraPPE/saturated $atom:C2 $atom:C3 ...}
  write_once('In Settings') {
    pair_coeff @atom:CH2 @atom:CH2 lj/cut 0.091411522 3.95
    pair_coeff @atom:CH3 @atom:CH3 lj/cut 0.194746286 3.75
  }
}
```

Moltemplate encapsulates molecules (moltemplate format)

3-body

watmw.lt

```
WatMW {
  write("Data Atoms") {$atom:mW $mol @atom:mW 0 0.0 0.0 0.0}

  write_once("system.in.sw") {
    mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333 7.0495562 \
0.602224558 4 0 0
  }
  write_once("In Settings") {
    pair_coeff    * * sw system.in.sw mW NULL NULL NULL
  }
}
```

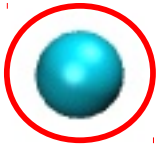


2-body

cyclododecane.lt

```
Cyclododecane {
  write('Data Atoms') {
    $atom:C1 $mol @atom:CH2  0.0 0.0 2.94118 0.0
    $atom:C2 $mol @atom:CH2  0.0 0.0 2.54714 1.47059 ...}
  write('Data Bonds') {
    $bond:bond1 @bond:TraPPE/saturated $atom:C1 $atom:C2
    $bond:bond2 @bond:TraPPE/saturated $atom:C2 $atom:C3 ...}
  write_once('In Settings') {
    pair_coeff @atom:CH2 @atom:CH2 lj/cut 0.091411522 3.95
    pair_coeff @atom:CH3 @atom:CH3 lj/cut 0.194746286 3.75
  }
}
```

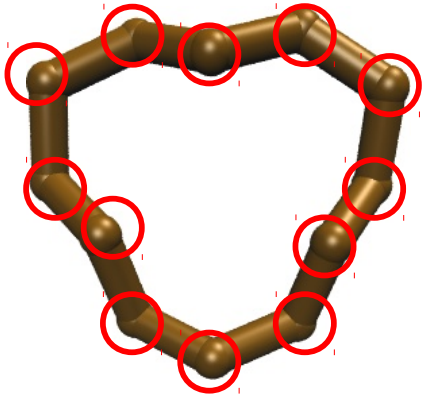
Moltemplate encapsulates molecules (moltemplate format)



3-body

watmw.lt

```
WatMW {
  write("Data Atoms") {$atom:mW $mol @atom:mW 0 0.0 0.0 0.0}

  write_once("system.in.sw") {
    mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333 7.0495562 \
0.602224558 4 0 0
  }
  write_once("In Settings") {
    pair_coeff    * * sw system.in.sw mW NULL NULL NULL
  }
}
```



2-body

cyclododecane.lt

```
Cyclododecane {
  write('Data Atoms') {
    $atom:C1 $mol @atom:CH2  0.0 0.0 2.94118 0.0
    $atom:C2 $mol @atom:CH2  0.0 0.0 2.54714 1.47059 ...}
  write('Data Bonds') {
    $bond:bond1 @bond:TraPPE/saturated $atom:C1 $atom:C2
    $bond:bond2 @bond:TraPPE/saturated $atom:C2 $atom:C3 ...}
  write_once('In Settings') {
    pair_coeff @atom:CH2 @atom:CH2 lj/cut 0.091411522 3.95
    pair_coeff @atom:CH3 @atom:CH3 lj/cut 0.194746286 3.75
  }
}
```

Moltemplate encapsulates molecules

(moltemplate format)

Example: Hydrocarbon phase-separation in water

3-body

watmw.lt

```
WatMW {
  write("Data Atoms") {$atom:mW $mol @atom:mW 0 0.0 0.0 0.0}

  write_once("system.in.sw") {
    mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333 7.0495562 \
0.602224558 4 0 0
  }
  write_once("In Settings") {
    pair_coeff    * * sw system.in.sw mW NULL NULL NULL
  }
}
```

2-body

cyclododecane.lt

```
Cyclododecane {
  write('Data Atoms') {
    $atom:C1 $mol @atom:CH2  0.0 0.0 2.94118 0.0
    $atom:C2 $mol @atom:CH2  0.0 0.0 2.54714 1.47059 ...}
  write('Data Bonds') {
    $bond:bond1 @bond:TraPPE/saturated $atom:C1 $atom:C2
    $bond:bond2 @bond:TraPPE/saturated $atom:C2 $atom:C3 ...}
  write_once('In Settings') {
    pair_coeff @atom:CH2 @atom:CH2 lj/cut 0.091411522 3.95
    pair_coeff @atom:CH3 @atom:CH3 lj/cut 0.194746286 3.75
  }
}
```

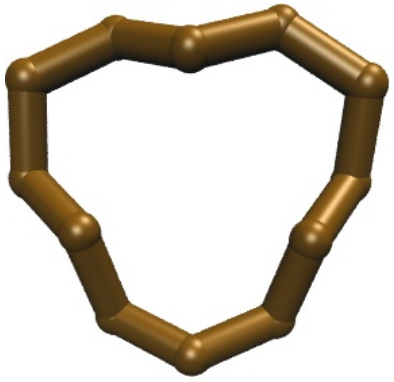
moltemplate format

(moltemplate format)

3-body

watmw.lt

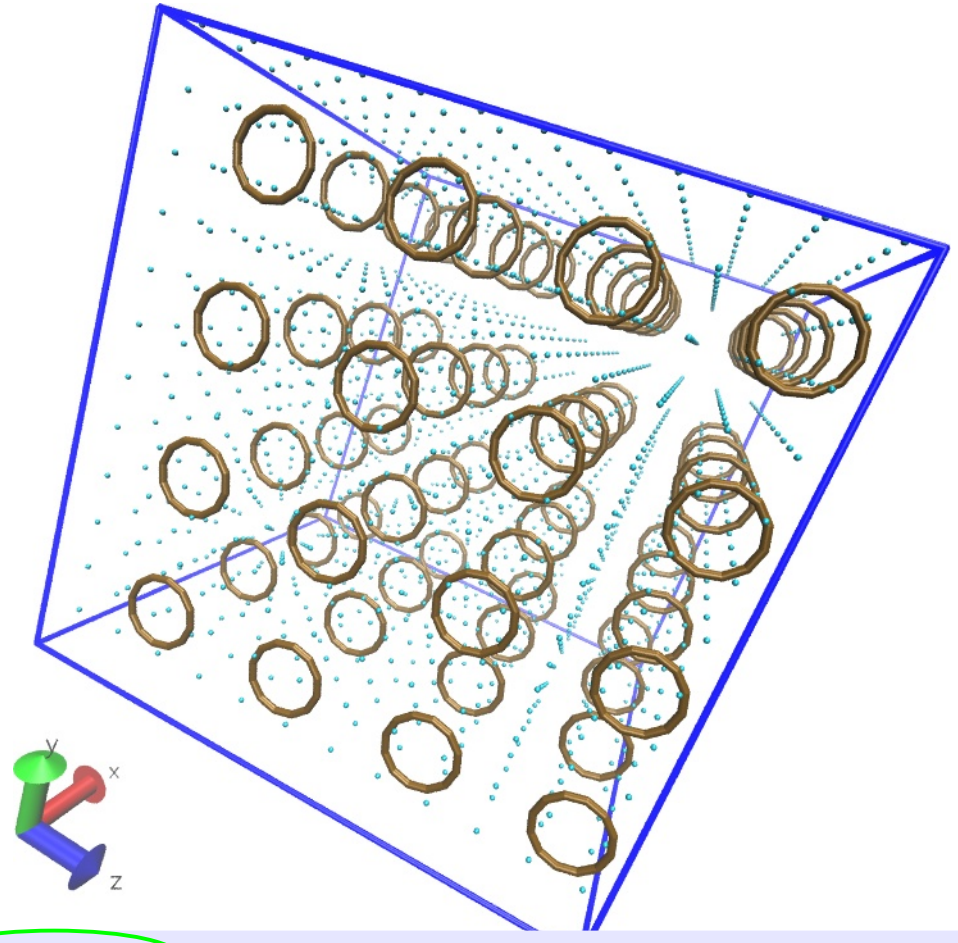
```
WatMW {
  write("Data Atoms") {$atom:mW $mol @atom:mW 0 0.0 0.0 0.0}

  write_once("system.in.sw") {
    mW mW mW 6.189 2.3925 1.8 23.15 1.2 -0.333333 7.0495562 \
0.602224558 4 0 0
  }
  write_once("In Settings") {
    pair_coeff    * * sw system.in.sw mW NULL NULL NULL
  }
}
```



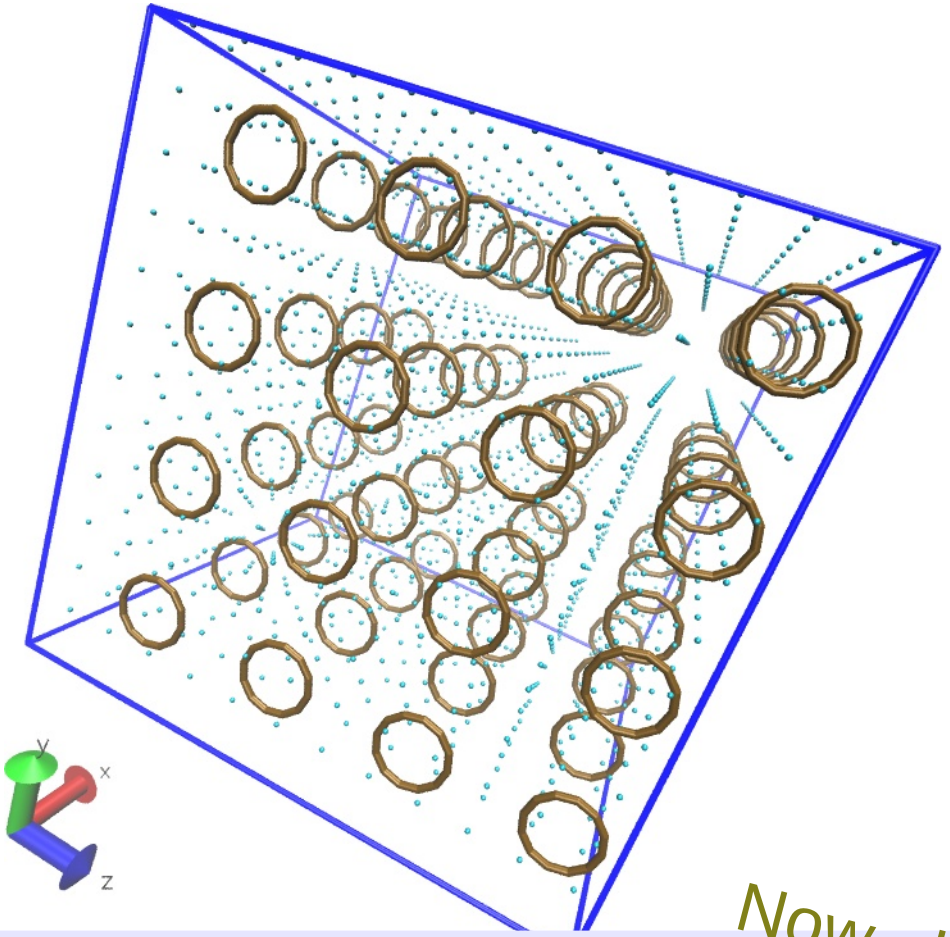
2-body

cyclododecane.lt

```
Cyclododecane {
  write('Data Atoms') {
    $atom:C1 $mol @atom:CH2  0.0 0.0 2.94118 0.0
    $atom:C2 $mol @atom:CH2  0.0 0.0 2.54714 1.47059 ...}
  write('Data Bonds') {
    $bond:bond1 @bond:TraPPE/saturated $atom:C1 $atom:C2
    $bond:bond2 @bond:TraPPE/saturated $atom:C2 $atom:C3 ...}
  write_once('In Settings') {
    pair_coeff @atom:CH2 @atom:CH2 lj/cut 0.091411522 3.95
    pair_coeff @atom:CH3 @atom:CH3 lj/cut 0.194746286 3.75
  }
}
```



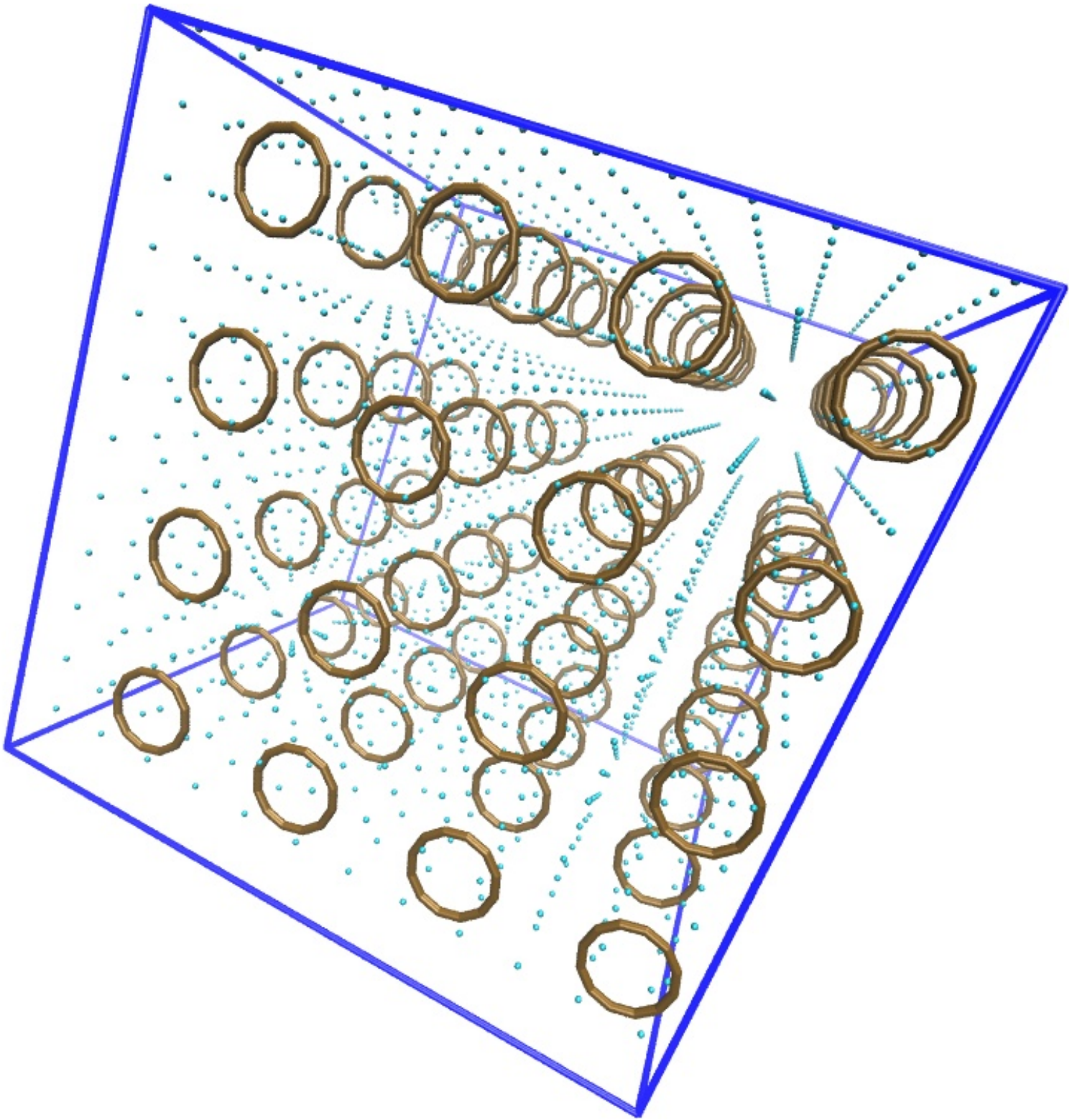

```
import "watmw.lt"
import "cyclododecane.lt"

wat = new WatMW [12].move(0, 0, 4.0)
                [12].move(0, 4.0, 0)
                [12].move(4.0, 0, 0)

cyclododecane = new Cyclododecane [4].move(0, 0, 12.0)
                                  [4].move(0, 12.0, 0)
                                  [4].move(12.0, 0, 0)
```




```
import "watmw.lt"
import "cyclododecane.lt"

wat = new WatMW [12].move(0, 0, 4.0)
                [12].move(0, 4.0, 0)
                [12].move(4.0, 0, 0)

cyclododecane = new Cyclododecane [4].move(0, 0, 12.0)
                                  [4].move(0, 12.0, 0)
                                  [4].move(12.0, 0, 0)
write_once("In Settings"){
  pair_coeff @atom:WatMW/mW @atom:Cyclododecane/CH2 lj/cut 0.1191 3.558
}
```

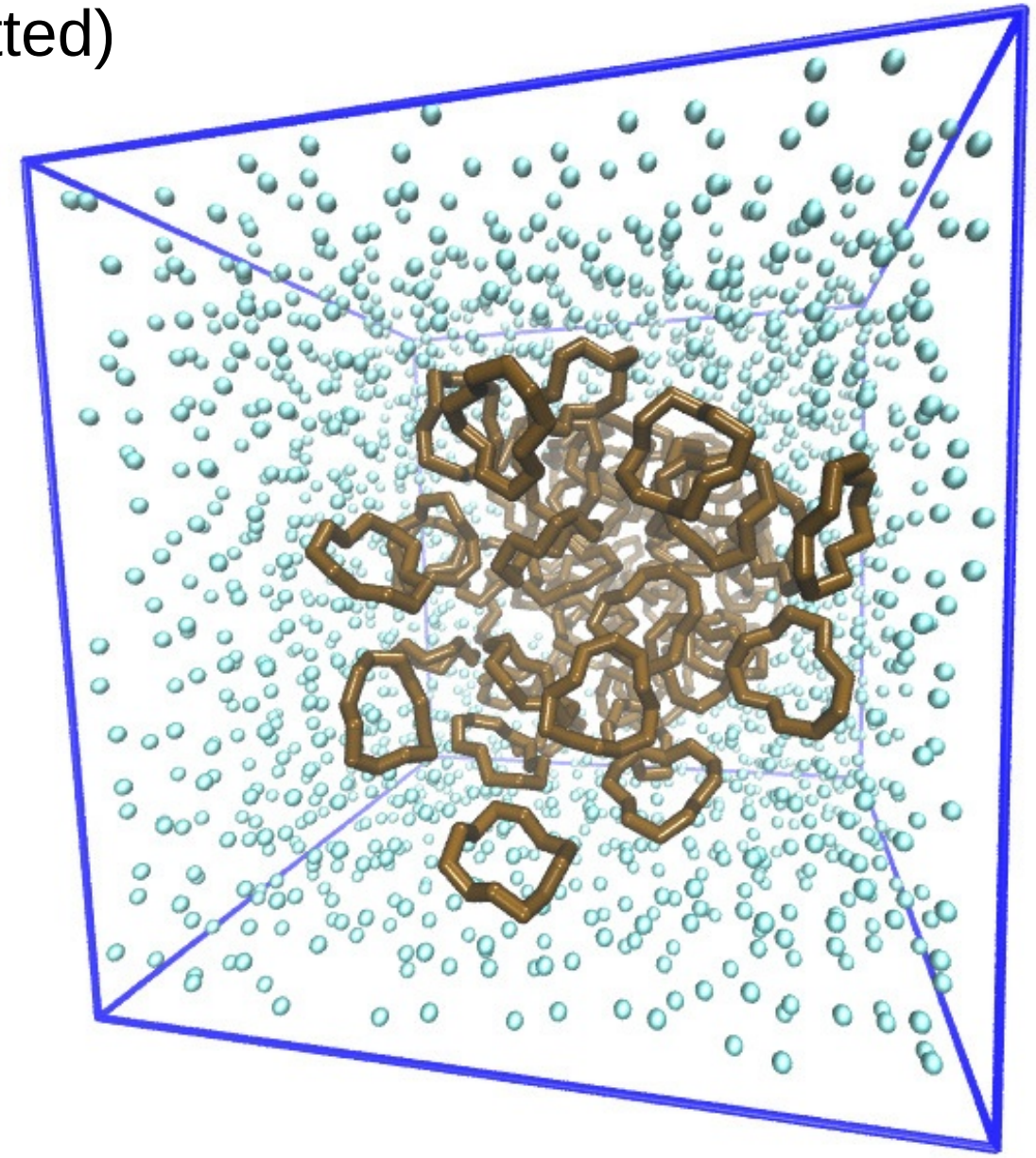
Now, define interactions between atoms from different types of molecules. (mixing rules disabled)

t = 0

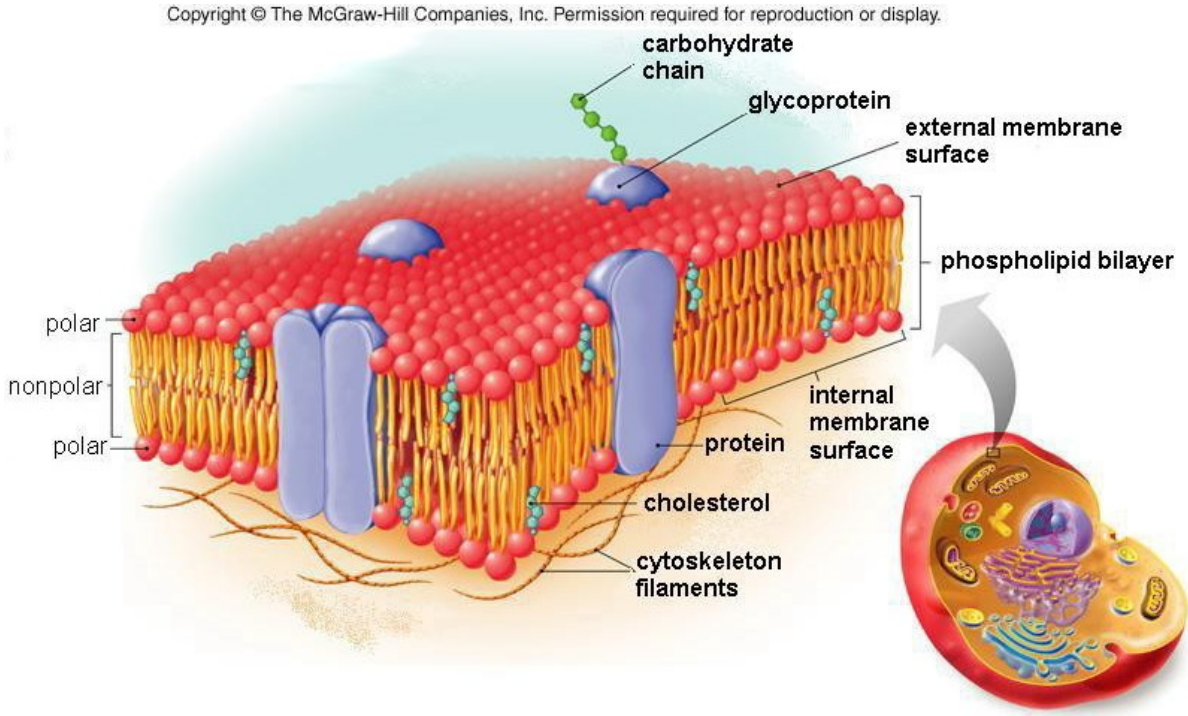


t = 400ps

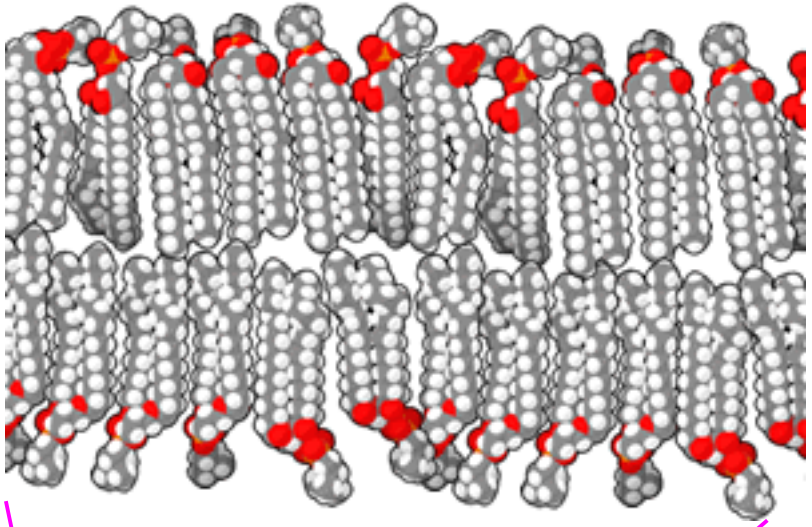
(movie omitted)

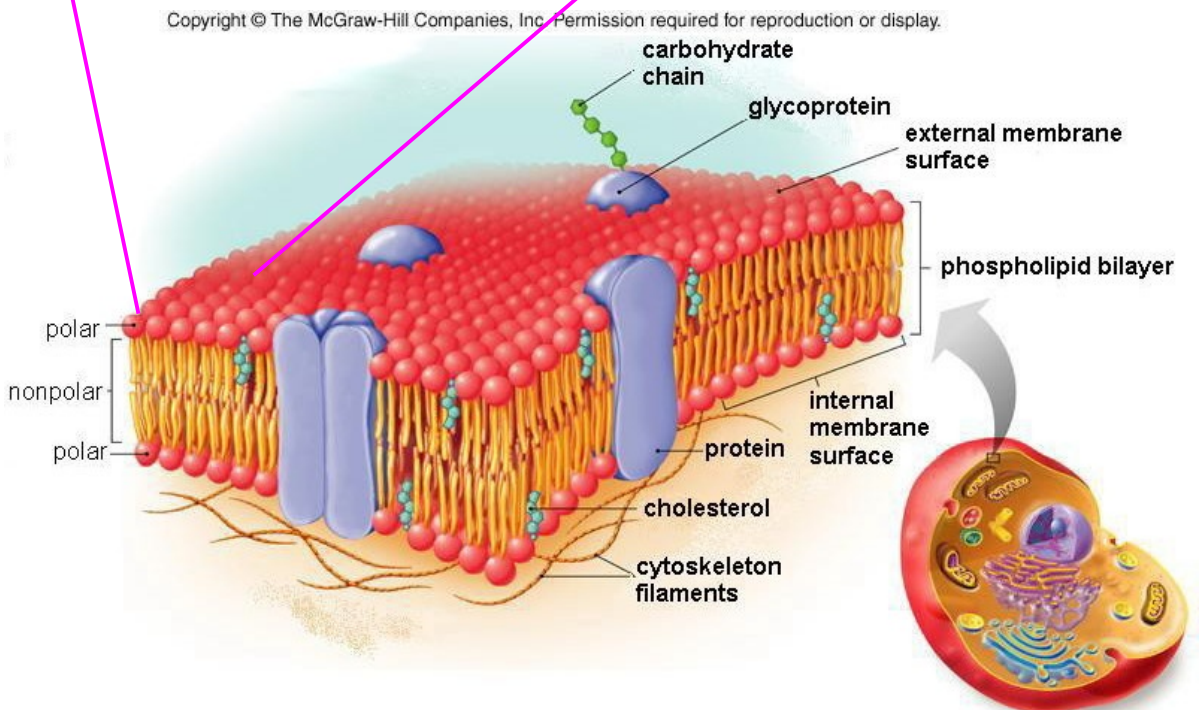


Building membrane structures

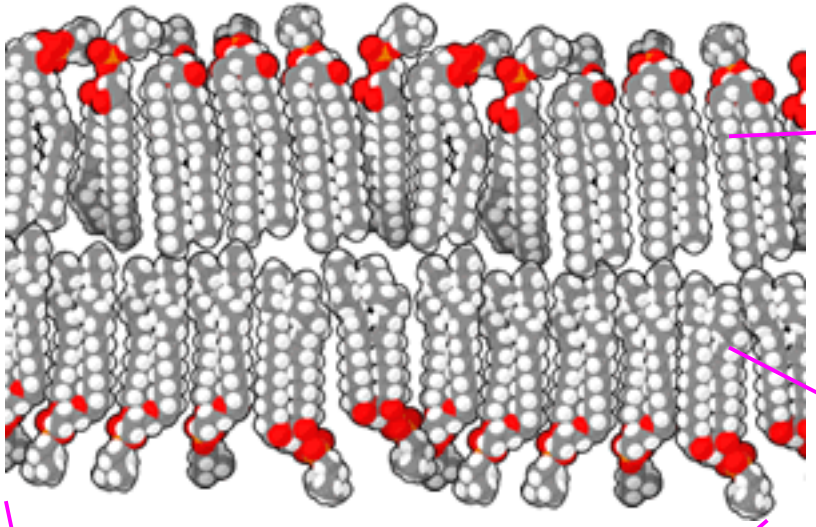


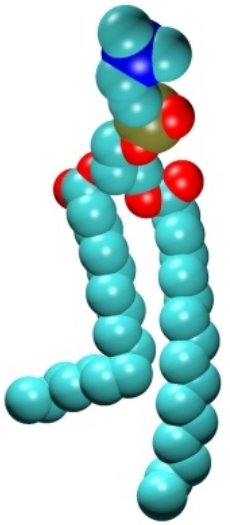
Building membrane structures

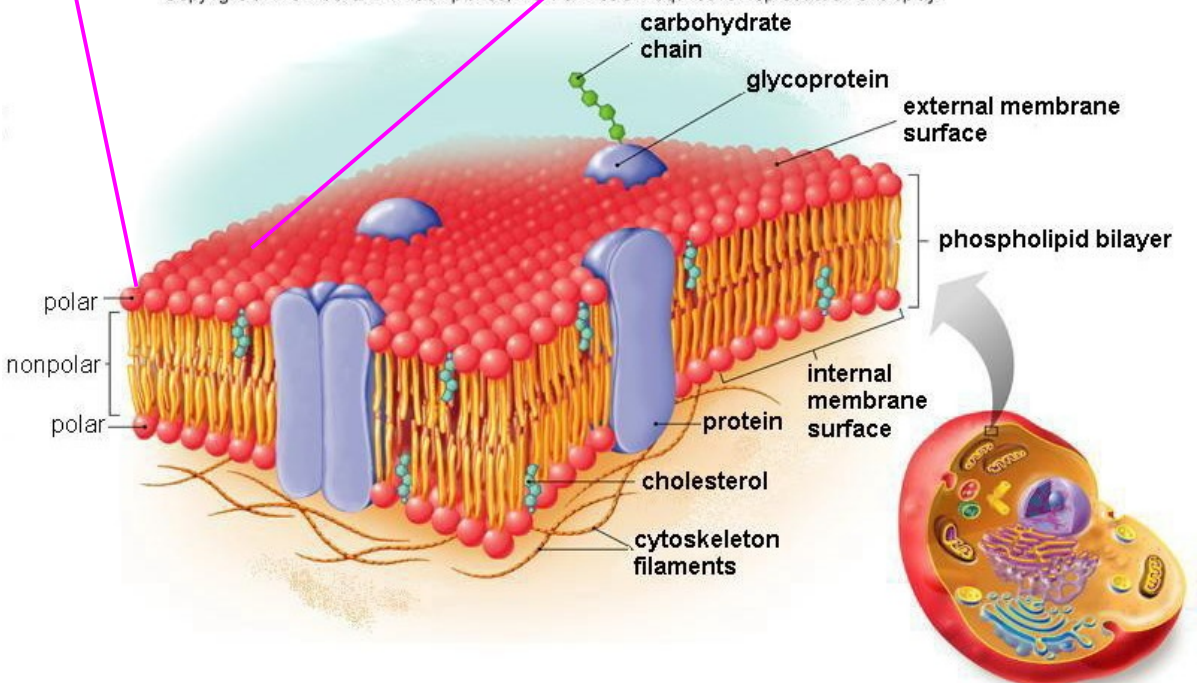




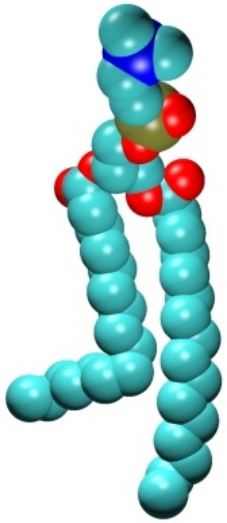
Building membrane structures







Building membrane structures



“DPPC”

?

Building CG membrane structures

"DPPC"

coarse-grained model by:

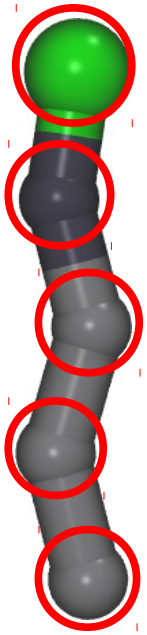
G. Brannigan, P.F. Philips, and F.L.H. Brown, Physical Review E, Vol 72, 011915 (2005)

Building CG membrane structures

"DPPC"

```
DPPC {
  write("Data Atoms") {
    $atom:h   $mol  @atom:head  0.0   0.00  0.00  33.75
    $atom:i   $mol  @atom:int   0.0  -1.00  0.00  26.25
    $atom:t1  $mol  @atom:tail  0.0   1.00  0.00  18.75
    $atom:t2  $mol  @atom:tail  0.0  -1.00  0.00  11.25
    $atom:t3  $mol  @atom:tail  0.0   1.00  0.00   3.75
  }
  write("Data Bonds") {
    $bond:b1    @bond:backbone   $atom:h  $atom:i
    $bond:b2    @bond:backbone   $atom:i  $atom:t1
    $bond:b3    @bond:backbone   $atom:t1 $atom:t2
    $bond:b4    @bond:backbone   $atom:t2 $atom:t3
  }
  write("Data Angles") {
    $angle:a1   @angle:backbone  $atom:h   $atom:i   $atom:t1
    $angle:a2   @angle:backbone  $atom:i   $atom:t1  $atom:t2
    $angle:a3   @angle:backbone  $atom:t1  $atom:t2  $atom:t3
  }
  write_once("In Settings") {
    pair_coeff @atom:head @atom:head lj/charmm/coul/charmm 0.1643 7.5
    :
  }
}
```

Building CG membrane structures



“DPPC”

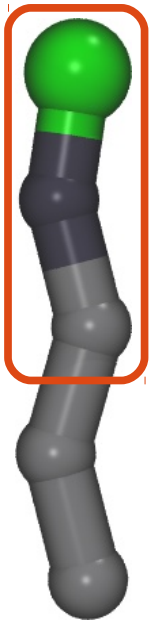
```
DPPC {
  write("Data Atoms") {
    $atom:h   $mol  @atom:head  0.0   0.00  0.00  33.75
    $atom:i   $mol  @atom:int   0.0  -1.00  0.00  26.25
    $atom:t1  $mol  @atom:tail  0.0   1.00  0.00  18.75
    $atom:t2  $mol  @atom:tail  0.0  -1.00  0.00  11.25
    $atom:t3  $mol  @atom:tail  0.0   1.00  0.00   3.75
  }
  write("Data Bonds") {
    $bond:b1    @bond:backbone   $atom:h  $atom:i
    $bond:b2    @bond:backbone   $atom:i  $atom:t1
    $bond:b3    @bond:backbone   $atom:t1 $atom:t2
    $bond:b4    @bond:backbone   $atom:t2 $atom:t3
  }
  write("Data Angles") {
    $angle:a1   @angle:backbone  $atom:h   $atom:i   $atom:t1
    $angle:a2   @angle:backbone  $atom:i   $atom:t1  $atom:t2
    $angle:a3   @angle:backbone  $atom:t1  $atom:t2  $atom:t3
  }
  write_once("In Settings") {
    pair_coeff @atom:head @atom:head lj/charmm/coul/charmm 0.1643 7.5
    :
  }
}
```

Building CG membrane structures

"DPPC"

```
DPPC {
  write("Data Atoms") {
    $atom:h   $mol  @atom:head  0.0   0.00  0.00  33.75
    $atom:i   $mol  @atom:int   0.0  -1.00  0.00  26.25
    $atom:t1  $mol  @atom:tail  0.0   1.00  0.00  18.75
    $atom:t2  $mol  @atom:tail  0.0  -1.00  0.00  11.25
    $atom:t3  $mol  @atom:tail  0.0   1.00  0.00   3.75
  }
  write("Data Bonds") {
    $bond:b1    @bond:backbone   $atom:h  $atom:i
    $bond:b2    @bond:backbone   $atom:i  $atom:t1
    $bond:b3    @bond:backbone   $atom:t1 $atom:t2
    $bond:b4    @bond:backbone   $atom:t2 $atom:t3
  }
  write("Data Angles") {
    $angle:a1   @angle:backbone  $atom:h   $atom:i   $atom:t1
    $angle:a2   @angle:backbone  $atom:i   $atom:t1  $atom:t2
    $angle:a3   @angle:backbone  $atom:t1  $atom:t2  $atom:t3
  }
  write_once("In Settings") {
    pair_coeff @atom:head @atom:head lj/charmm/coul/charmm 0.1643 7.5
    :
  }
}
```

Building CG membrane structures



"DPPC"

```
DPPC {
  write("Data Atoms") {
    $atom:h   $mol  @atom:head  0.0   0.00  0.00  33.75
    $atom:i   $mol  @atom:int   0.0  -1.00  0.00  26.25
    $atom:t1  $mol  @atom:tail  0.0   1.00  0.00  18.75
    $atom:t2  $mol  @atom:tail  0.0  -1.00  0.00  11.25
    $atom:t3  $mol  @atom:tail  0.0   1.00  0.00   3.75
  }
  write("Data Bonds") {
    $bond:b1    @bond:backbone   $atom:h  $atom:i
    $bond:b2    @bond:backbone   $atom:i  $atom:t1
    $bond:b3    @bond:backbone   $atom:t1 $atom:t2
    $bond:b4    @bond:backbone   $atom:t2 $atom:t3
  }
  write("Data Angles") {
    $angle:a1   @angle:backbone  $atom:h   $atom:i   $atom:t1
    $angle:a2   @angle:backbone  $atom:i   $atom:t1  $atom:t2
    $angle:a3   @angle:backbone  $atom:t1  $atom:t2  $atom:t3
  }
  write_once("In Settings") {
    pair_coeff @atom:head @atom:head lj/charmm/coul/charmm 0.1643 7.5
    :
  }
}
```

Building CG membrane structures

“DPPC”

```
DPPC {
  write("Data Atoms") {
    $atom:h   $mol  @atom:head  0.0   0.00  0.00  33.75
    $atom:i   $mol  @atom:int   0.0  -1.00  0.00  26.25
    $atom:t1  $mol  @atom:tail  0.0   1.00  0.00  18.75
    $atom:t2  $mol  @atom:tail  0.0  -1.00  0.00  11.25
    $atom:t3  $mol  @atom:tail  0.0   1.00  0.00   3.75
  }
  write("Data Bonds") {
    $bond:b1    @bond:backbone   $atom:h  $atom:i
    $bond:b2    @bond:backbone   $atom:i  $atom:t1
    $bond:b3    @bond:backbone   $atom:t1 $atom:t2
    $bond:b4    @bond:backbone   $atom:t2 $atom:t3
  }
  write("Data Angles") {
    $angle:a1   @angle:backbone  $atom:h   $atom:i   $atom:t1
    $angle:a2   @angle:backbone  $atom:i   $atom:t1  $atom:t2
    $angle:a3   @angle:backbone  $atom:t1  $atom:t2  $atom:t3
  }
  write_once("In Settings") {
    pair_coeff @atom:head @atom:head lj/charmm/coul/charmm 0.1643 7.5
    :
  }
}
```

Building CG membrane structures

“DPPC”

```
DPPC {
  write("Data Atoms") {
    $atom:h   $mol  @atom:head  0.0   0.00  0.00  33.75
    $atom:i   $mol  @atom:int   0.0  -1.00  0.00  26.25
    $atom:t1  $mol  @atom:tail  0.0   1.00  0.00  18.75
    $atom:t2  $mol  @atom:tail  0.0  -1.00  0.00  11.25
    $atom:t3  $mol  @atom:tail  0.0   1.00  0.00   3.75
  }
  write("Data Bonds") {
    $bond:b1    @bond:backbone   $atom:h  $atom:i
    $bond:b2    @bond:backbone   $atom:i  $atom:t1
    $bond:b3    @bond:backbone   $atom:t1 $atom:t2
    $bond:b4    @bond:backbone   $atom:t2 $atom:t3
  }
  write("Data Angles") {
    $angle:a1   @angle:backbone  $atom:h   $atom:i   $atom:t1
    $angle:a2   @angle:backbone  $atom:i   $atom:t1  $atom:t2
    $angle:a3   @angle:backbone  $atom:t1  $atom:t2  $atom:t3
  }
  write_once("In Settings") {
    pair_coeff @atom:head @atom:head lj/charmm/coul/charmm 0.1643 7.5
    :
  }
}
```

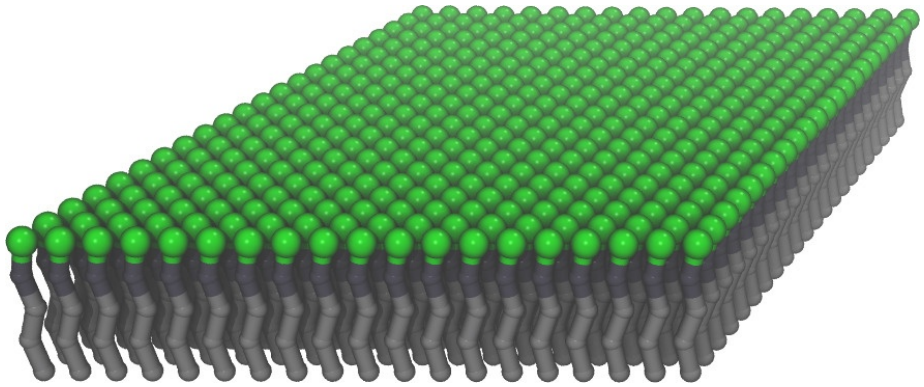
Building CG membrane structures


```
lipids = new DPPC
```

Building CG membrane structures

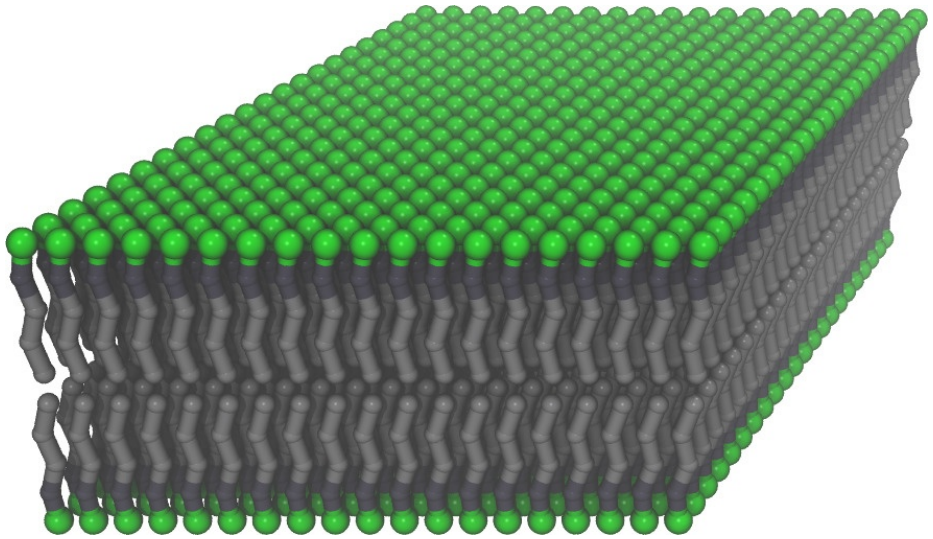

```
lipids = new DPPC [19][22]
```

Building CG membrane structures



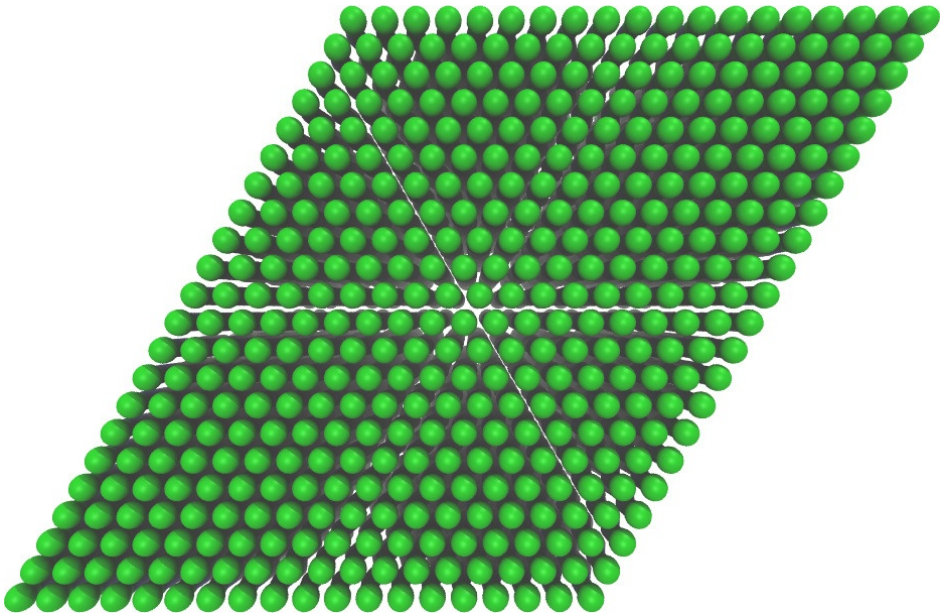
```
lipids = new DPPC [19].move(7.5,  0,  0)
                  [22].move(3.75, 6.49519, 0)
```

Building CG membrane structures



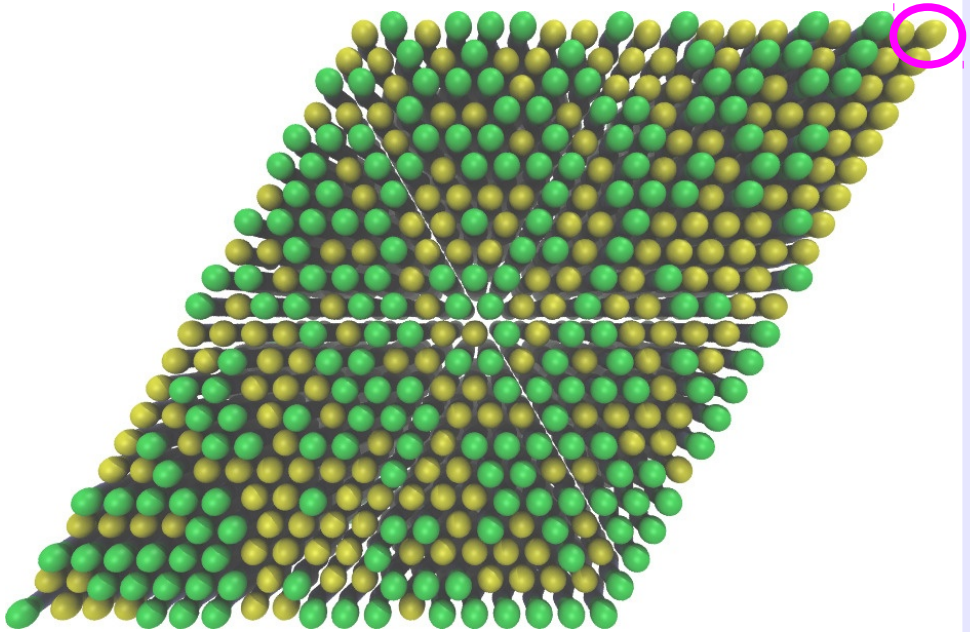
```
lipids = new DPPC [19].move(7.5,  0,  0)
                  [22].move(3.75, 6.49519, 0)
                  [2].rot(180, 1, 0, 0)
```

Building CG membrane structures



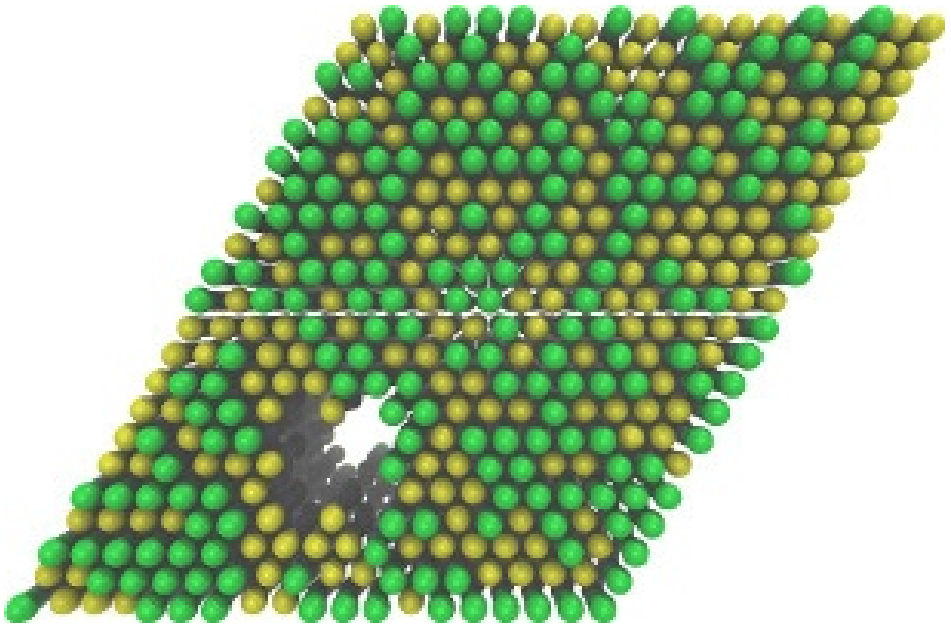
```
lipids = new DPPC [19].move(7.5,  0,  0)
                  [22].move(3.75, 6.49519, 0)
                  [2].rot(180, 1, 0, 0)
```

Building CG membrane structures



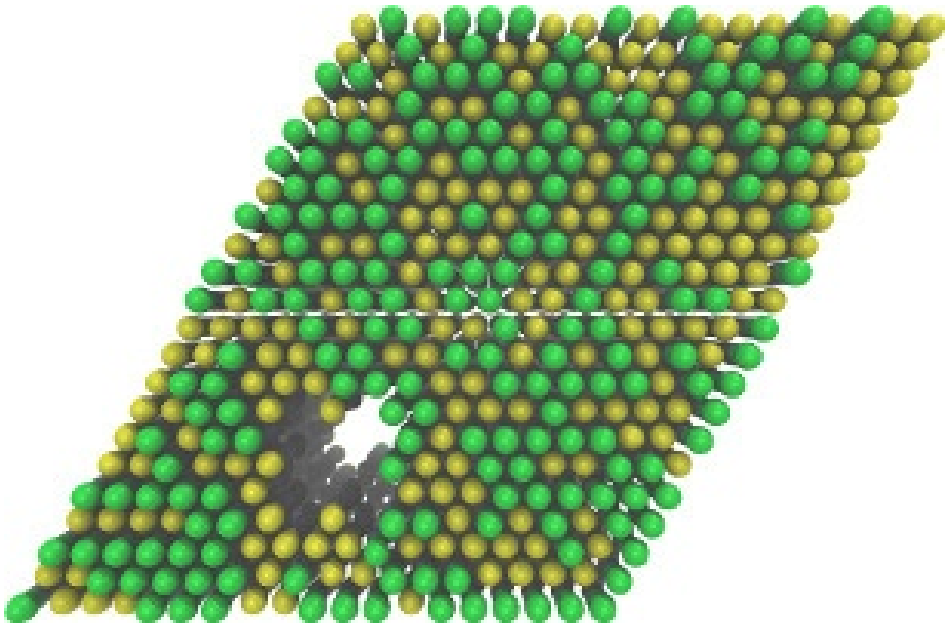
```
lipids = new random([DPPC,DLPC], [0.5,0.5])
                    [19].move(7.5,  0,  0)
                    [22].move(3.75, 6.49519, 0)
                    [2].rot(180, 1, 0, 0)
```

Building CG membrane structures

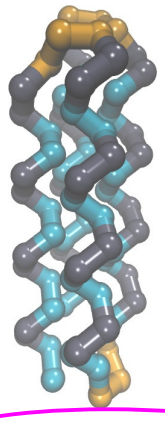


```
lipids = new random([DPPC,DLPC], [0.5,0.5])
                   [19].move(7.5,  0,  0)
                   [22].move(3.75, 6.49519, 0)
                   [2].rot(180, 1, 0, 0)
delete lipids[7][3][*]
delete lipids[9][3][*]
delete lipids[6-9][4][*]
delete lipids[6-8][5][*]
delete lipids[5-8][6][*]
delete lipids[5][7][*]
delete lipids[7][7][*]
```

Building CG membrane structures



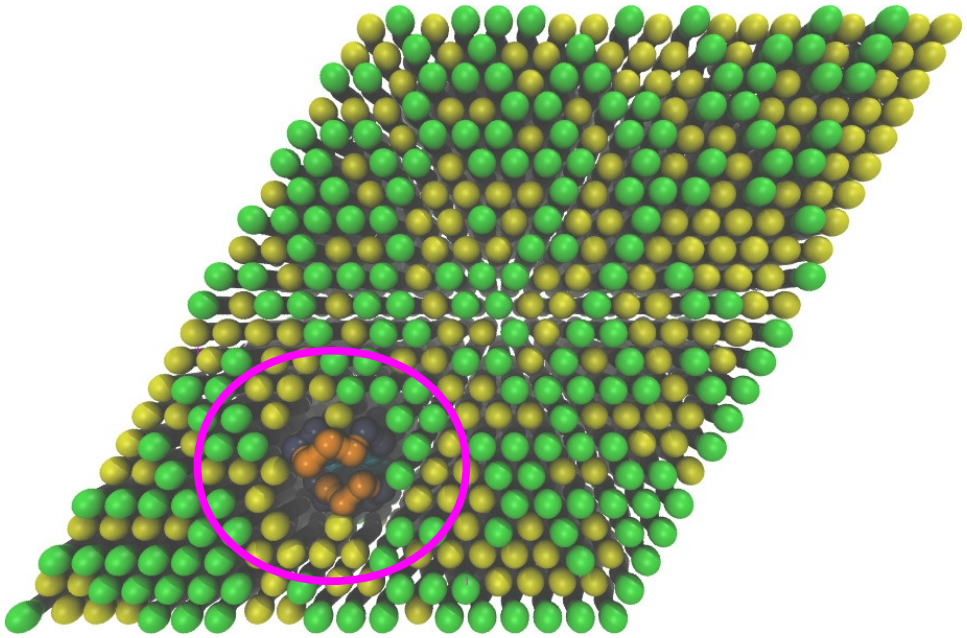
+



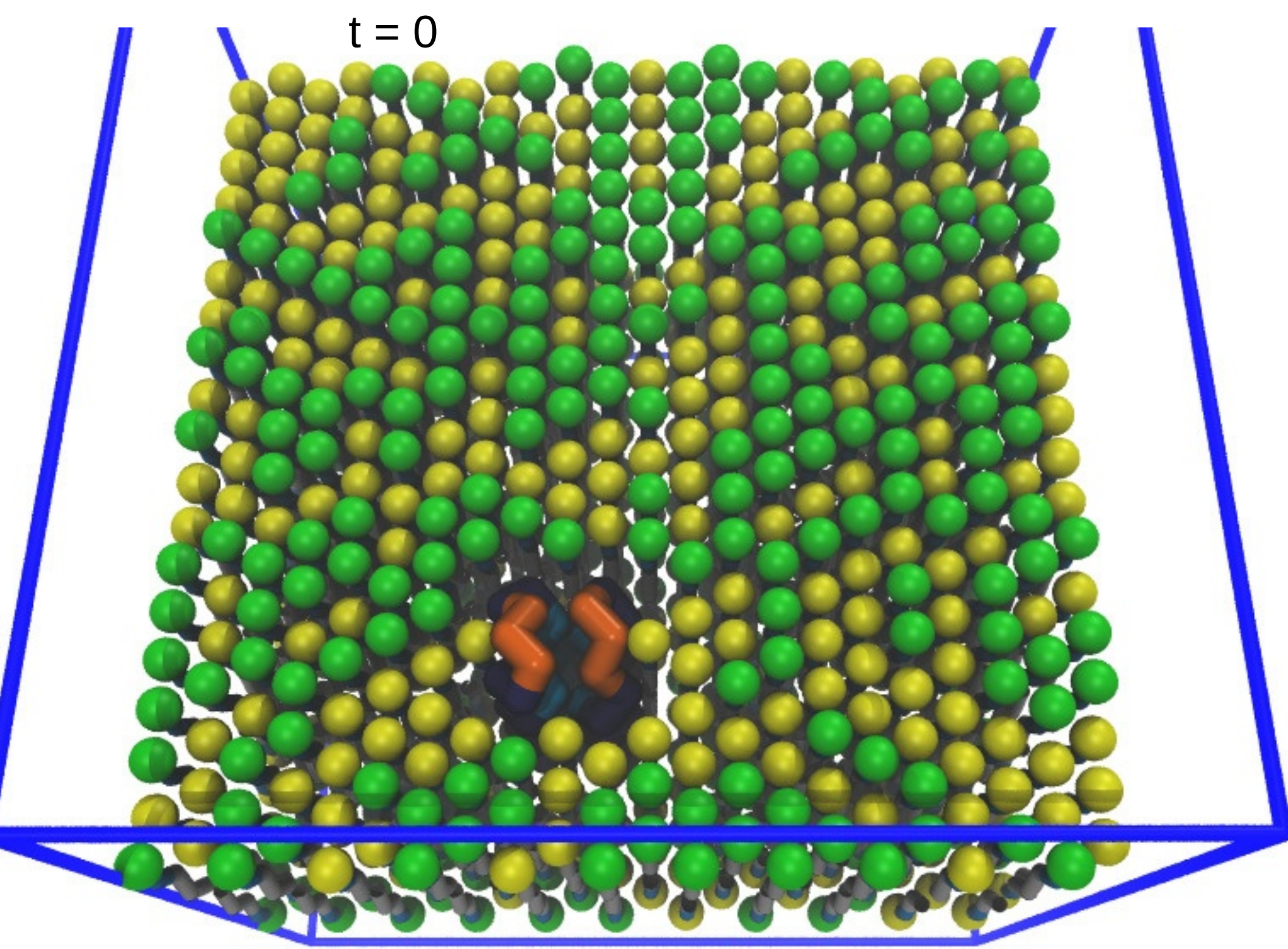
"4HelixBundle"

```
lipids = new random([DPPC,DLPC], [0.5,0.5])
                   [19].move(7.5,  0,  0)
                   [22].move(3.75, 6.49519, 0)
                   [2].rot(180, 1, 0, 0)
delete lipids[7][3][*]
delete lipids[9][3][*]
delete lipids[6-9][4][*]
delete lipids[6-8][5][*]
delete lipids[5-8][6][*]
delete lipids[5][7][*]
delete lipids[7][7][*]

protein = new 4HelixBundle
```

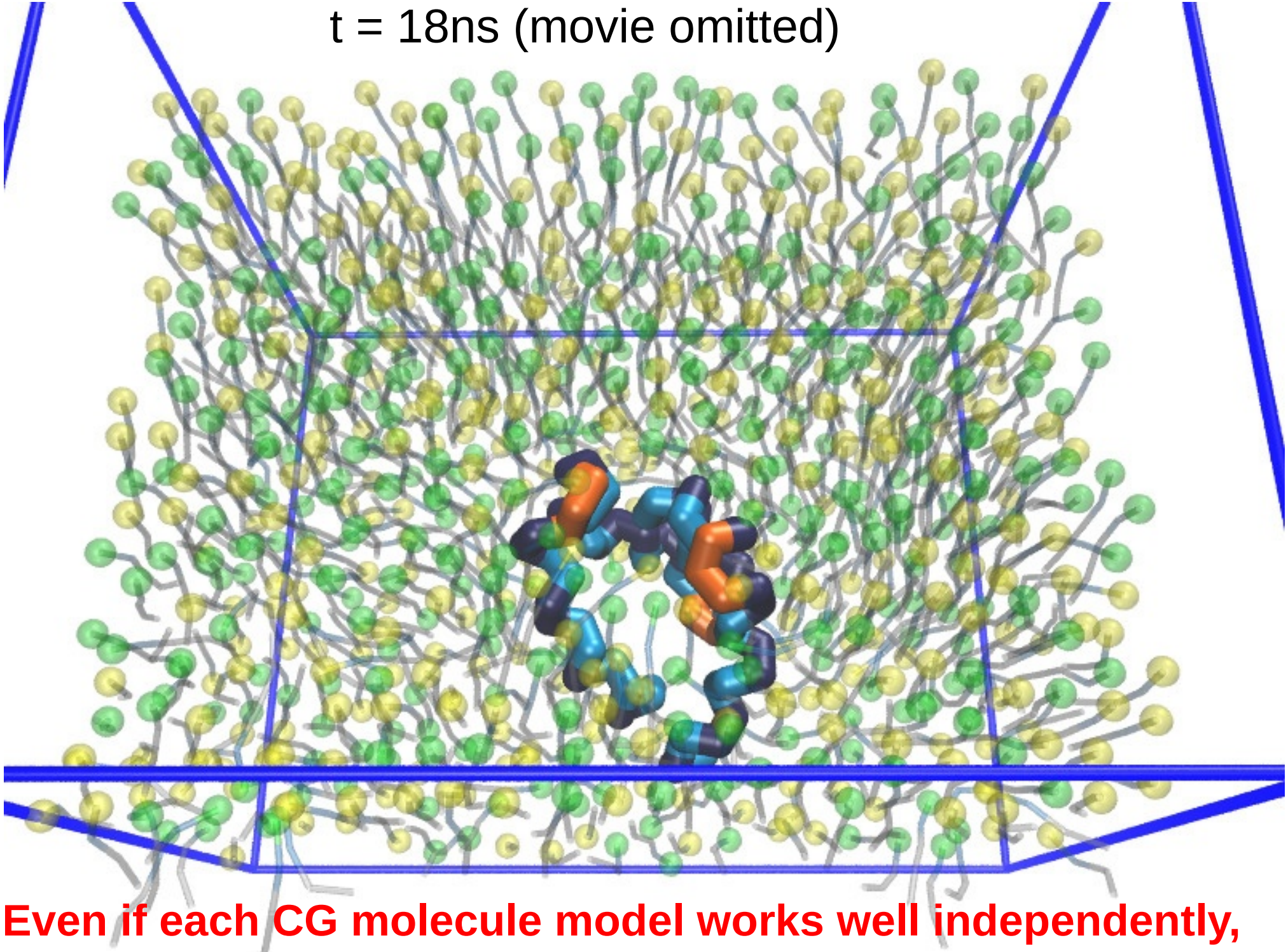
Building CG membrane structures



```
lipids = new random([DPPC,DLPC], [0.5,0.5])
                   [19].move(7.5,  0,  0)
                   [22].move(3.75, 6.49519, 0)
                   [2].rot(180, 1, 0, 0)
delete lipids[7][3][*]
delete lipids[9][3][*]
delete lipids[6-9][4][*]
delete lipids[6-8][5][*]
delete lipids[5-8][6][*]
delete lipids[5][7][*]
delete lipids[7][7][*]

protein = new 4HelixBundle
protein.move(45.0, 25.98076211, 0)
```


t = 0

t = 18ns (movie omitted)



Even if each CG molecule model works well independently, you must choose the interaction between them carefully.

Building CG membrane structures

“DPPC”

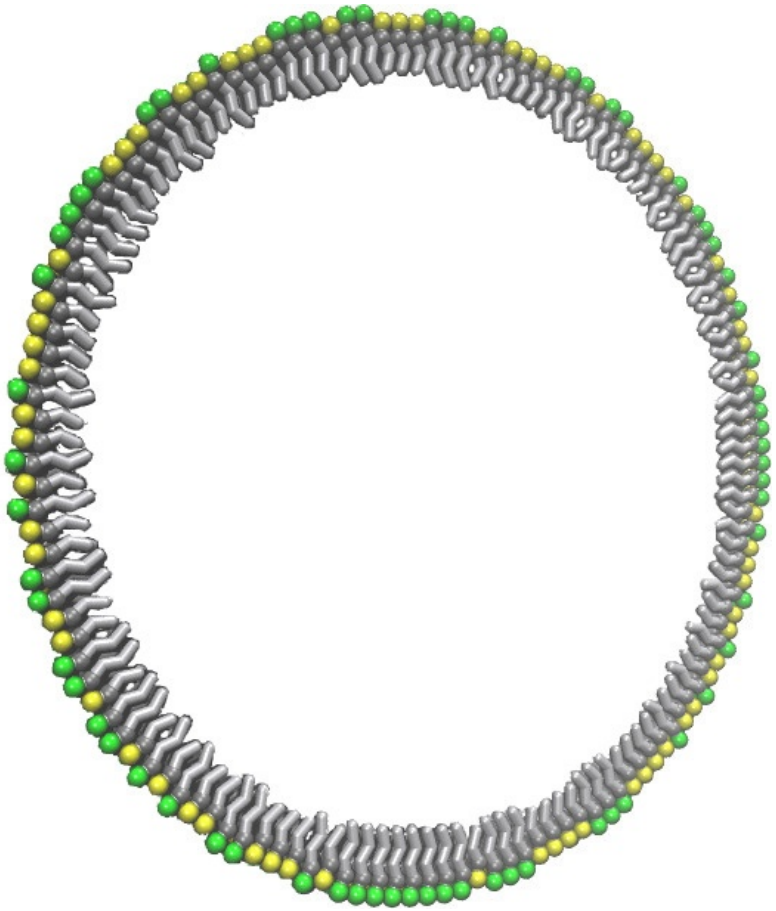
```
lipids = new DPPC
```

Building CG membrane structures

“DPPC”

```
lipids = new DPPC.move(0,0,177)
```

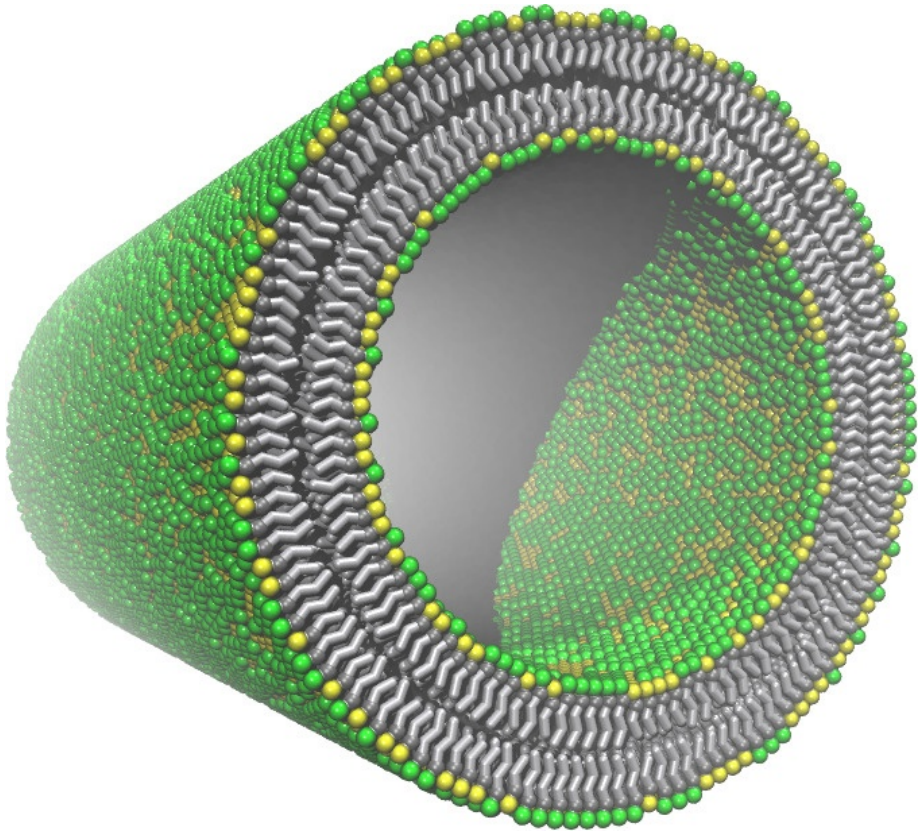
Building CG membrane structures



A ring of random DPPC or DLPC molecules

```
l_out=new random([DPPC.move(0,0,177),
                  DLPC.move(0,0,177)],
                 [0.5,0.5])
      [136].rot(2.6470588, 1,0,0)
```

Building CG membrane structures



```
l_out=new random([DPPC.move(0,0,177),
                  DLPC.move(0,0,177)],
                 [0.5,0.5])
      [136].rot(2.6470588, 1,0,0)
      [110].rot(1.3235294,1,0,0).move(6.49519, 0, 0)

l_in = new random([DPPC.rot(180,1,0,0).move(0,0,177),
                   DLPC.rot(180,1,0,0).move(0,0,177)],
                  [0.5,0.5])
      [112].rot(3.2142857, 1,0,0)
      [110].rot(1.60714285,1,0,0).move(6.49519, 0, 0)
```

Who uses moltemplate?

- Moltemplate has been downloaded 1801 times from 1021 unique IP addresses (in 10 months)

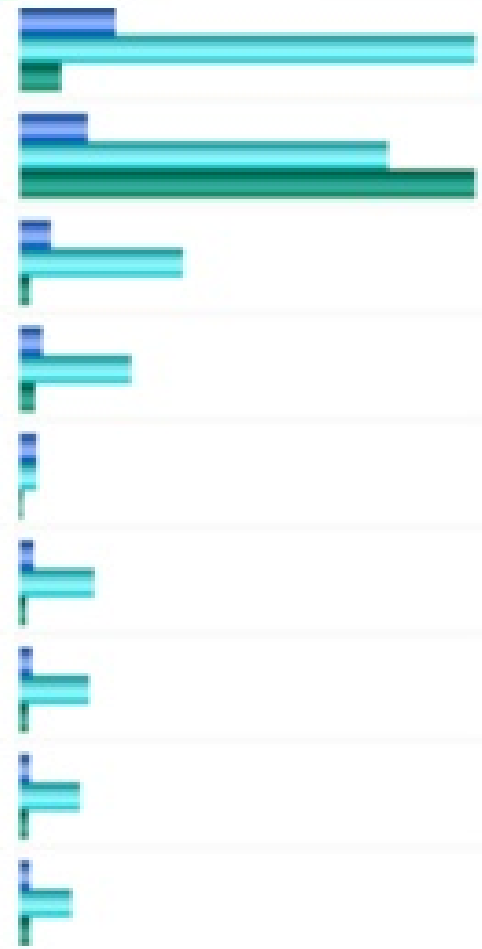
  (not including copies of moltemplate bundled with LAMMPS)

  At least half are from universities.

Who uses moltemplate?

Stats (July 2013, top 9 countries)

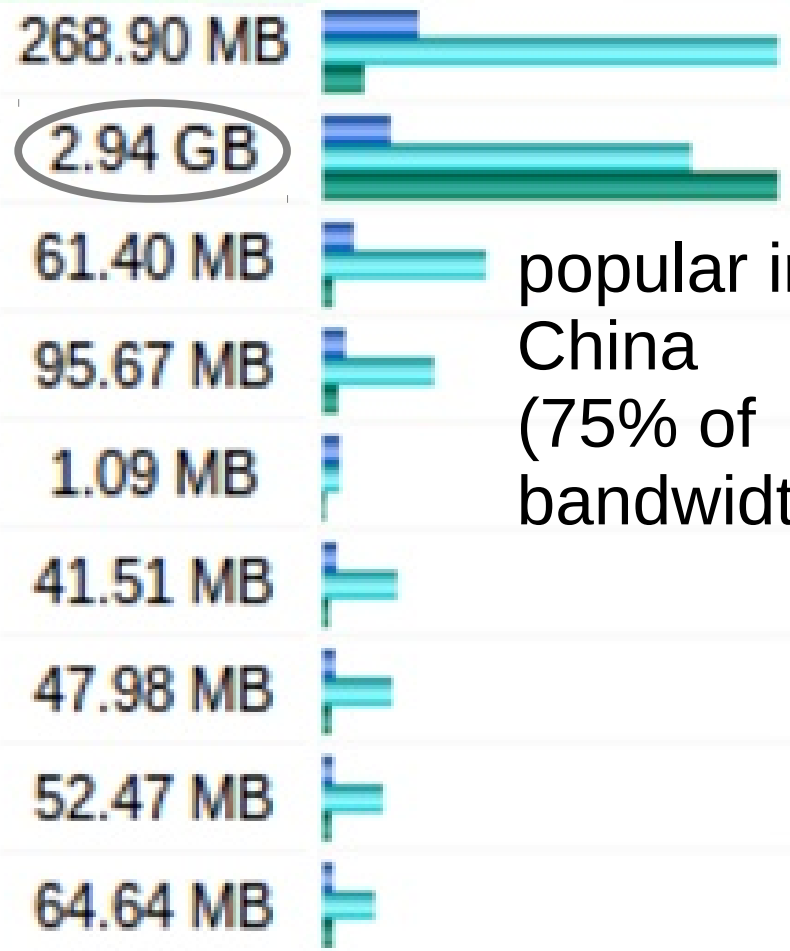
Countries	Pages	Hits	Bandwidth
United States	1,055	5,069	268.90 MB
China	754	4,096	2.94 GB
France	341	1,827	61.40 MB
Japan	234	1,231	95.67 MB
Ukraine	190	192	1.09 MB
Great Britain	141	826	41.51 MB
India	129	773	47.98 MB
Brazil	113	665	52.47 MB
Iran	112	584	64.64 MB
:	:	:	:	:



Who uses moltemplate?

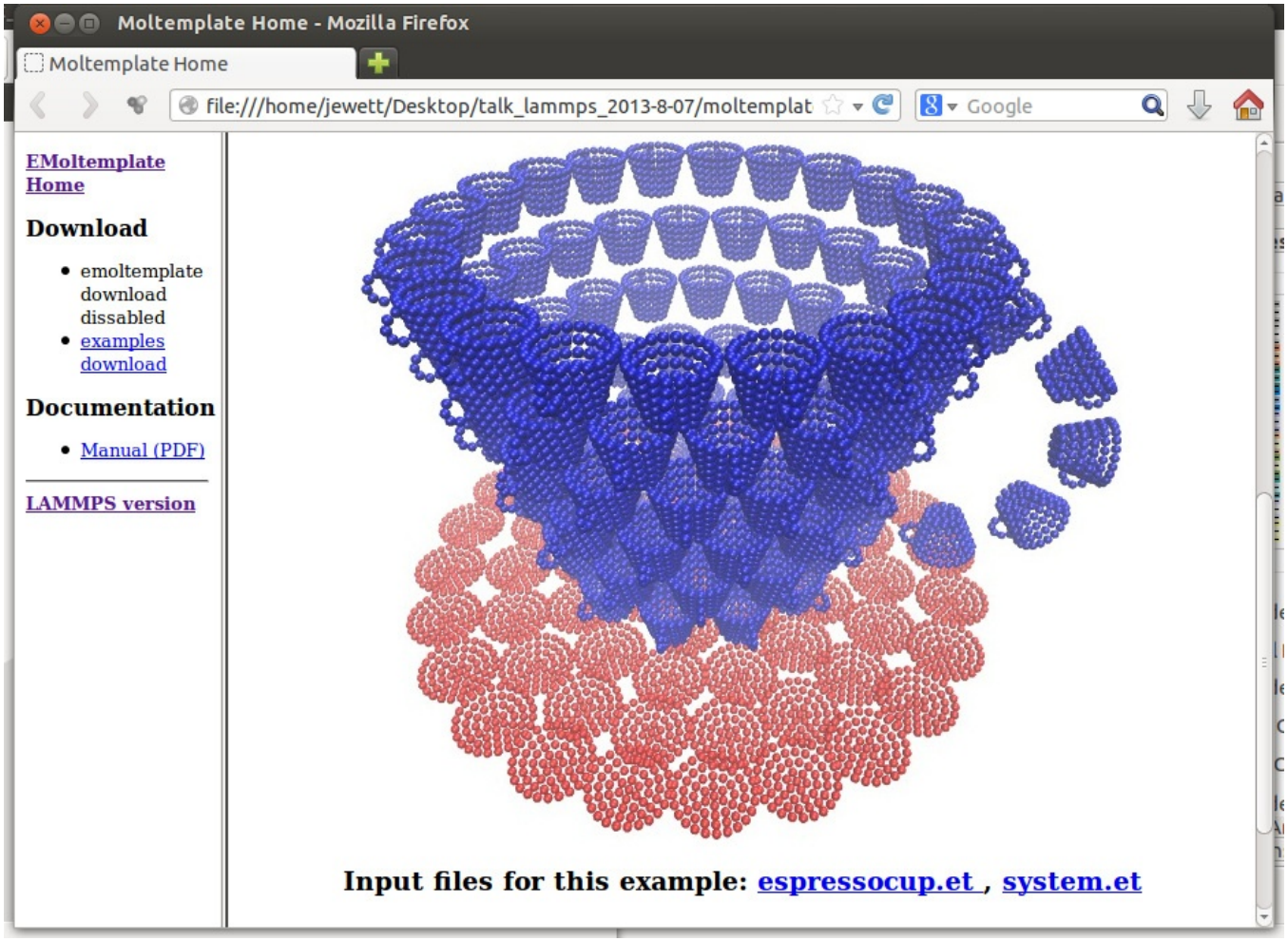
Stats (July 2013, top 9 countries)

Countries	Pages	Hits	Bandwidth
United States	1,055	5,069	268.90 MB
China	754	4,096	2.94 GB
France	341	1,827	61.40 MB
Japan	234	1,231	95.67 MB
Ukraine	190	192	1.09 MB
Great Britain	141	826	41.51 MB
India	129	773	47.98 MB
Brazil	113	665	52.47 MB
Iran	112	584	64.64 MB
:	:	:	:



popular in China (75% of bandwidth)

Moltemplate is a text manipulation program. It can be modified to work with other MD programs.



Moltemplate Summary

“Moltemplate” is just a text generator useful for creating input files for simulation programs.

Users can extract all of the text in the simulation input file(s) related to a particular molecule,

-convert it into a “*template-object*”,

-duplicate it (optional: customize each copy),

-use it as a building block for creating larger molecules (optional).

Moltemplate works with (nearly) any simulation file format. (E.g: AMBER, NAMD, CHARMM, etc.. Moltemplate has been customized for LAMMPS and ESPResSo.)

Acknowledgements (moltemplate)

Joan-Emma Shea

NSF-grant 1056587 (2012)

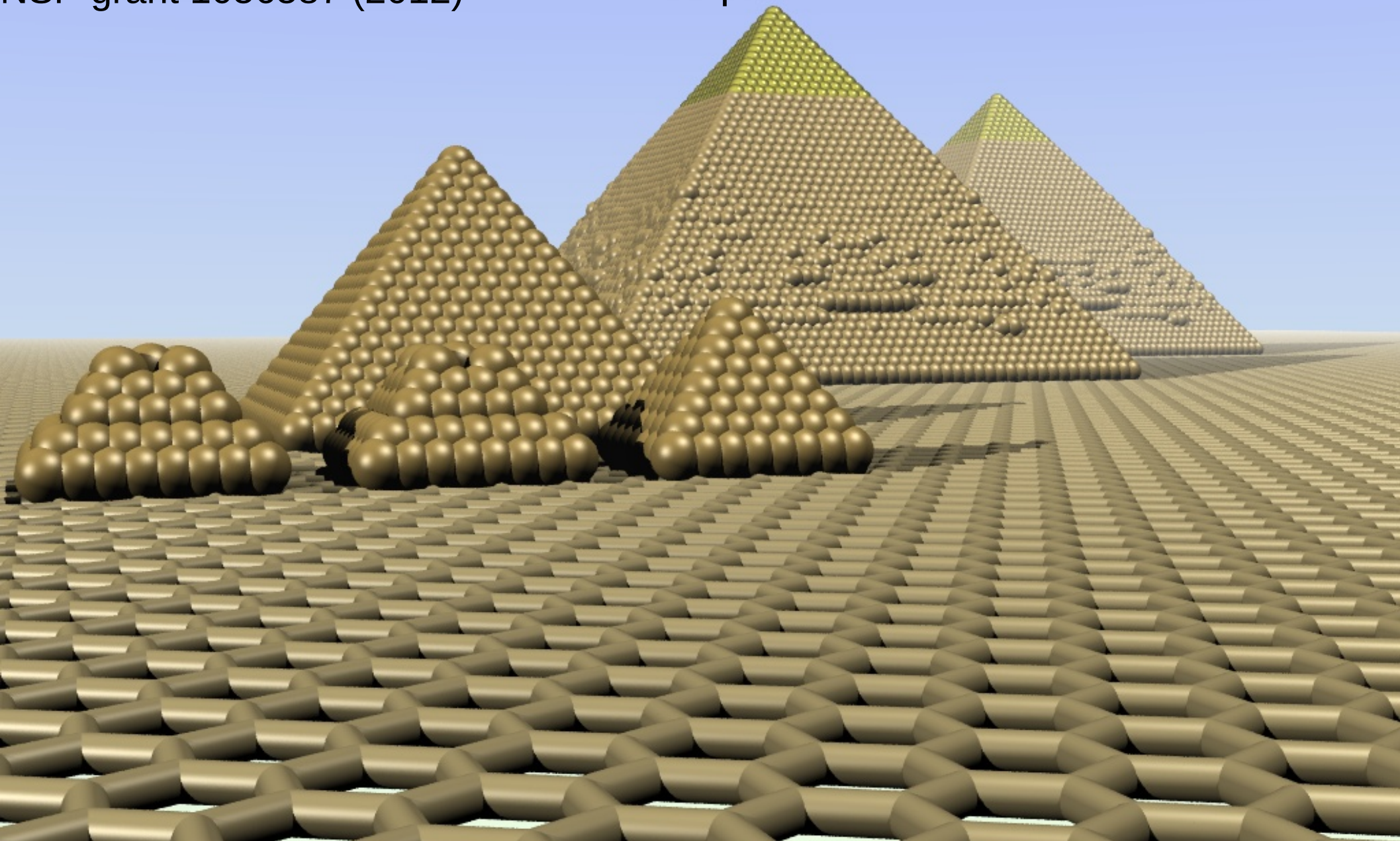
Steve Plimpton

Axel Kohlmeyer

Luca Larini

Sotiria Lampoudi

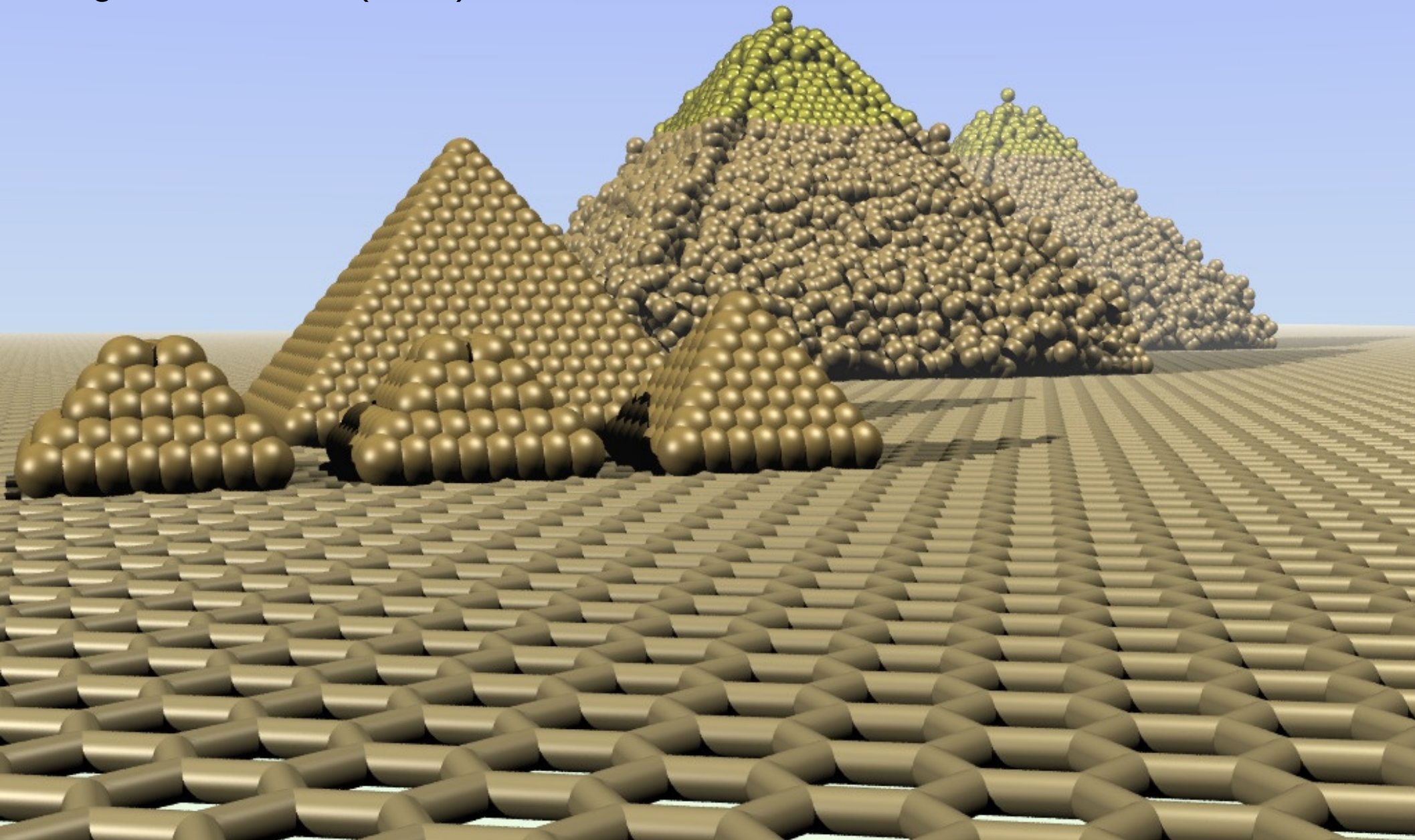
Erez Lieberman-Aiden



Acknowledgements (moltemplate)

Joan-Emma Shea Steve Plimpton Axel Kohlmeyer Luca Larini

NSF-grant 1056587 (2012) Sotiria Lampoudi Erez Lieberman-Aiden



Acknowledgements (moltemplate)

Joan-Emma Shea Steve Plimpton Axel Kohlmeyer Luca Larini

NSF-grant 1056587 (2012) Sotiria Lampoudi Erez Lieberman-Aiden