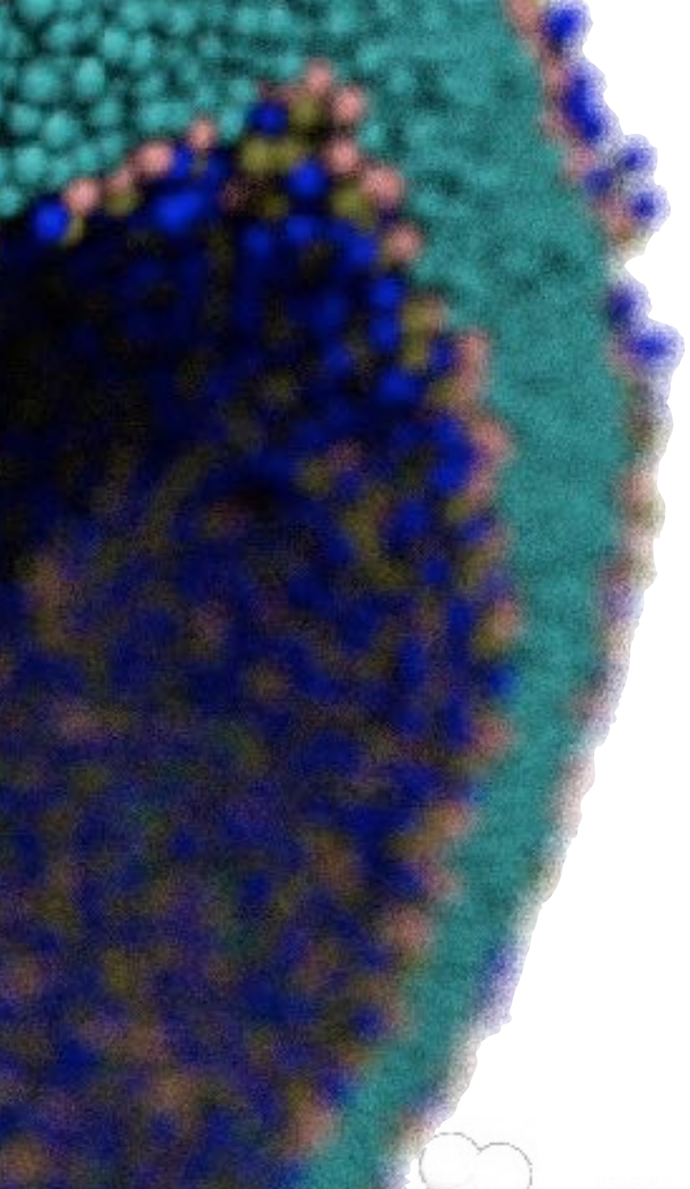


NUMFOCUS

OPEN CODE = BETTER SCIENCE

Northwestern

WEINBERG COLLEGE
OF ARTS & SCIENCES

Northwestern

McCORMICK SCHOOL
OF ENGINEERING

NSF

MD ANALYSIS

Introduction

MDAnalysis Workshop and Hackathon,
Northwestern University, Evanston, IL
Nov 12, 2018

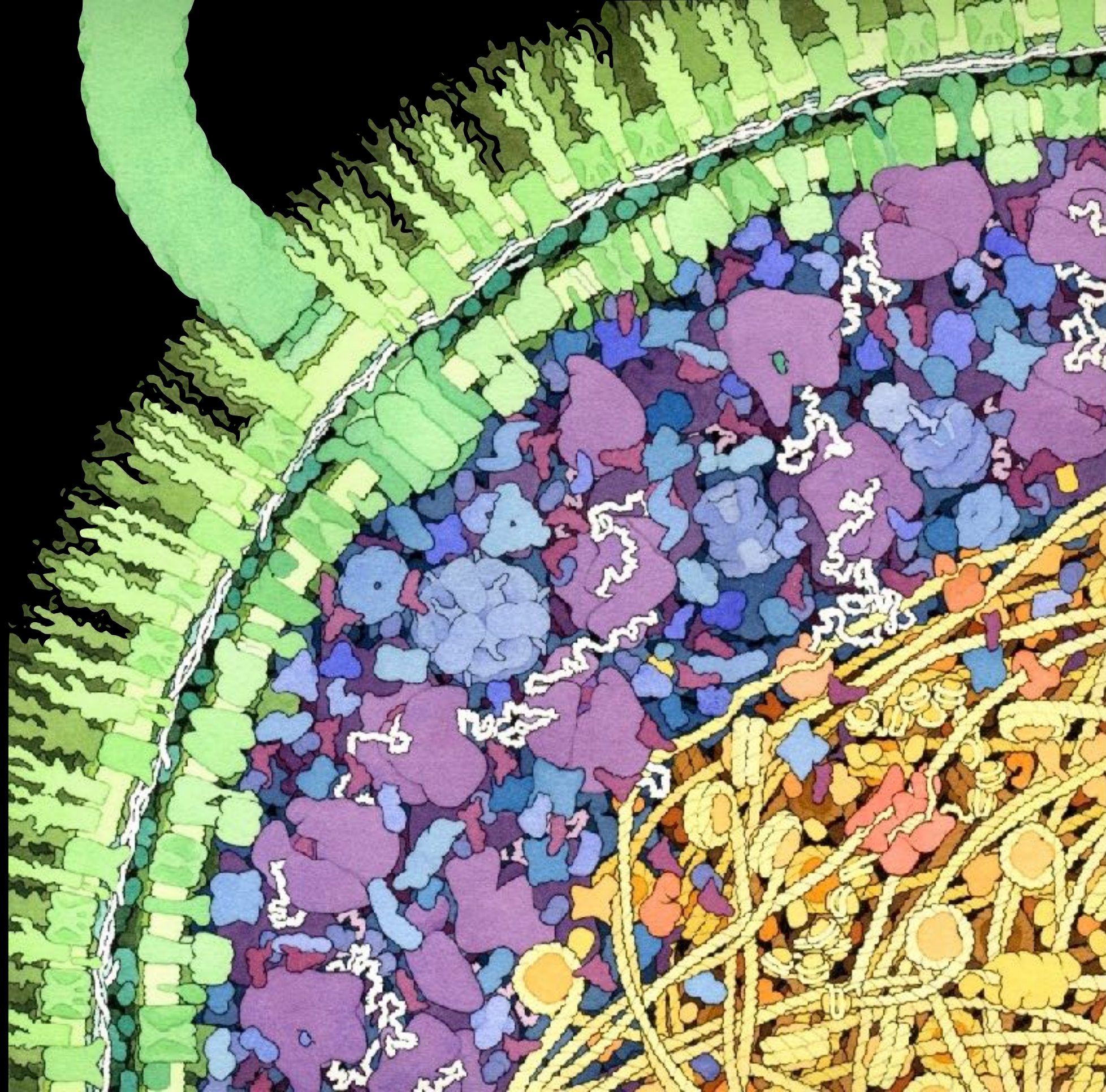
Oliver Beckstein
Arizona State University
ASU
https://becksteinlab.physics.asu.edu

LIFE

cells

molecules

atoms



Escherichia coli (© 1999 David S. Goodsell, the Scripps Research Institute)

Biomolecules

nucleic acids (DNA, RNA)

proteins

lipids

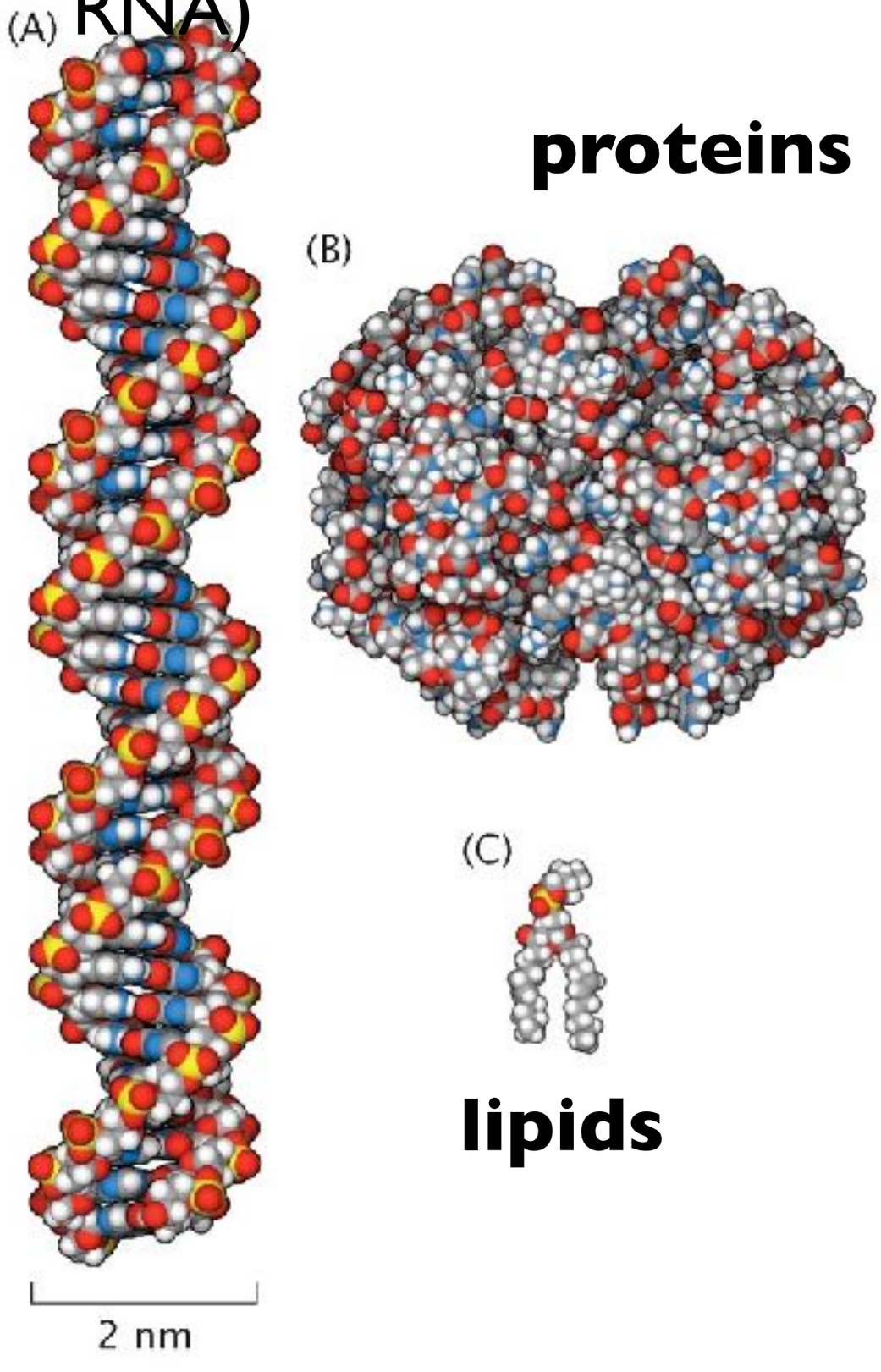


Figure 1.1 Physical Biology of the Cell, 2ed. (© Garland Science 2013)

Biomolecules

nucleic acids (DNA, RNA)

proteins

lipids

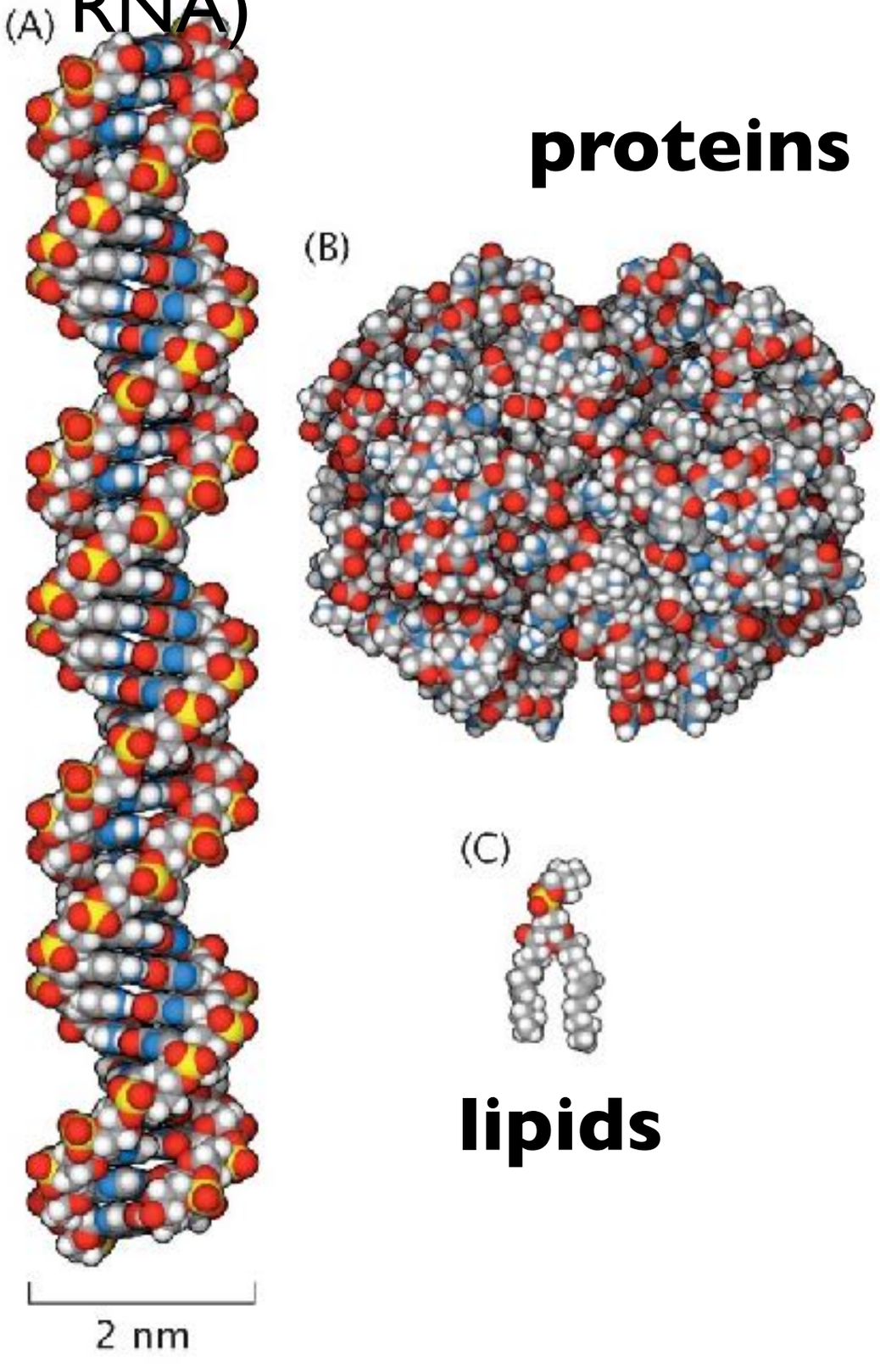
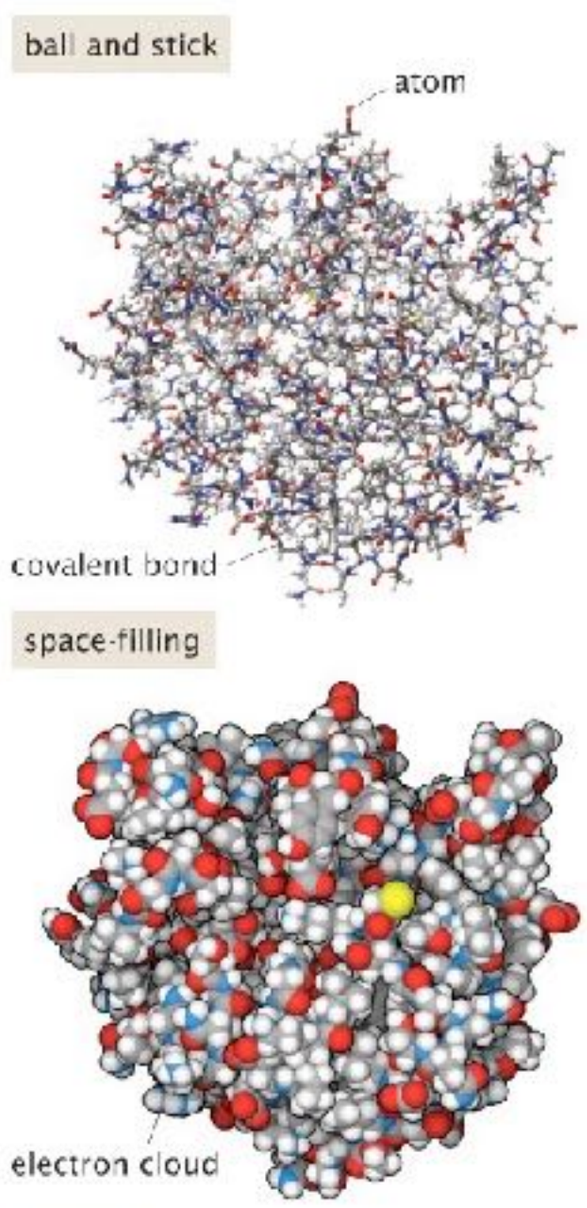


Figure 1.1 Physical Biology of the Cell, 2ed. (© Garland Science 2013)

protein representations



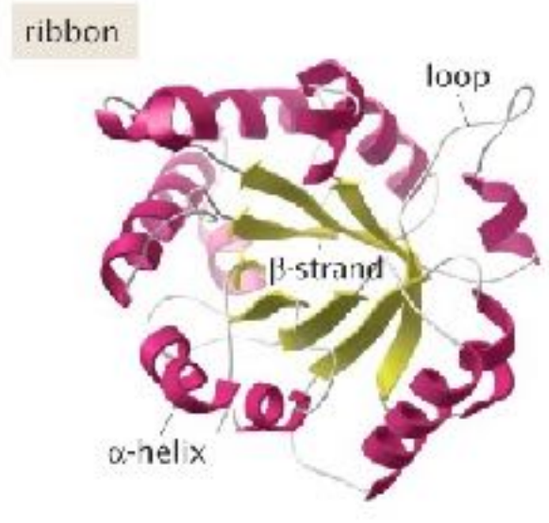


Figure 2.32 Physical Biology of the Cell, 2ed.

Biomolecules

nucleic acids (DNA, RNA)

proteins

lipids

water

protein representations

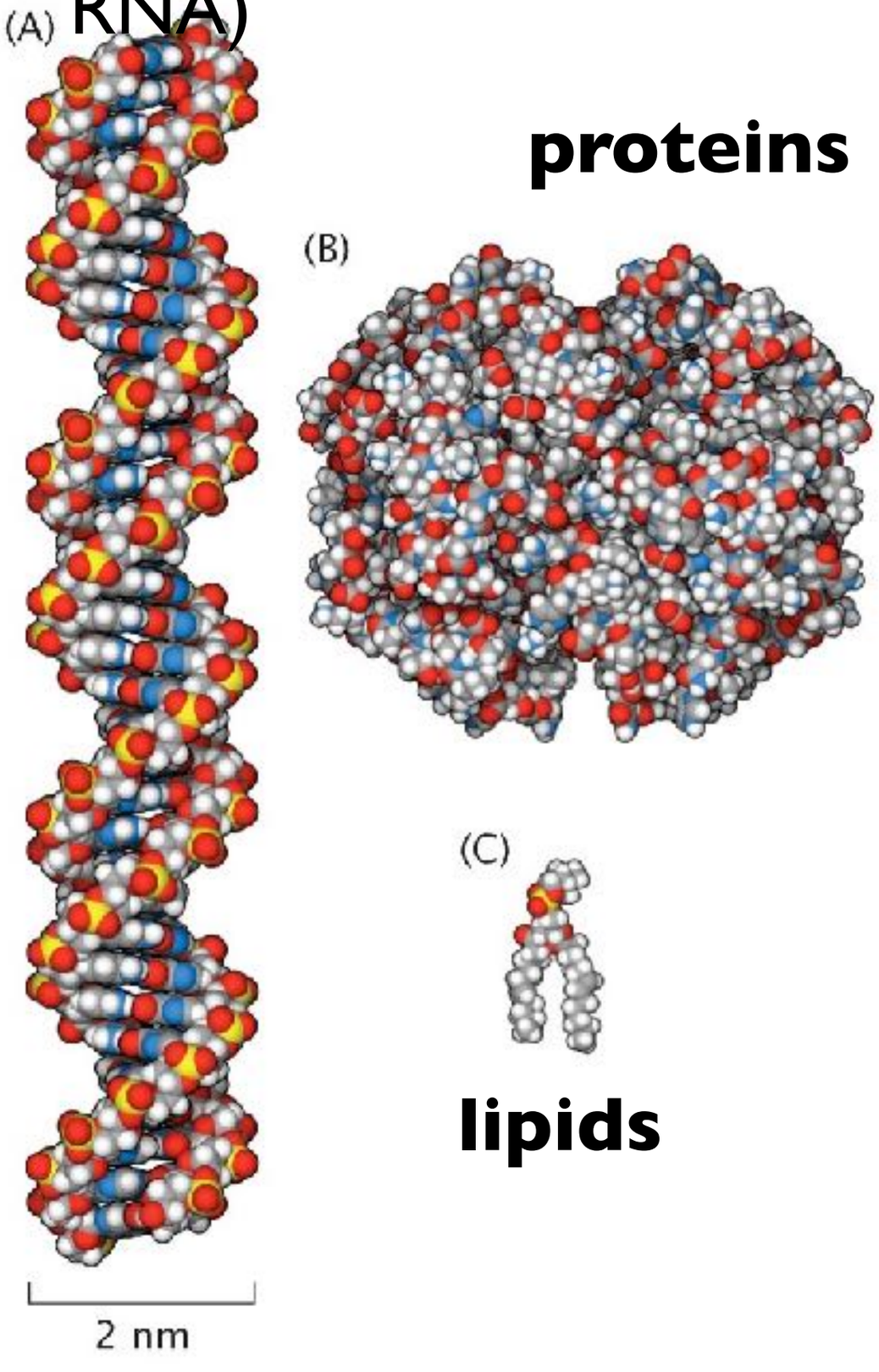


Figure 1.1 Physical Biology of the Cell, 2ed. (© Garland Science 2013)

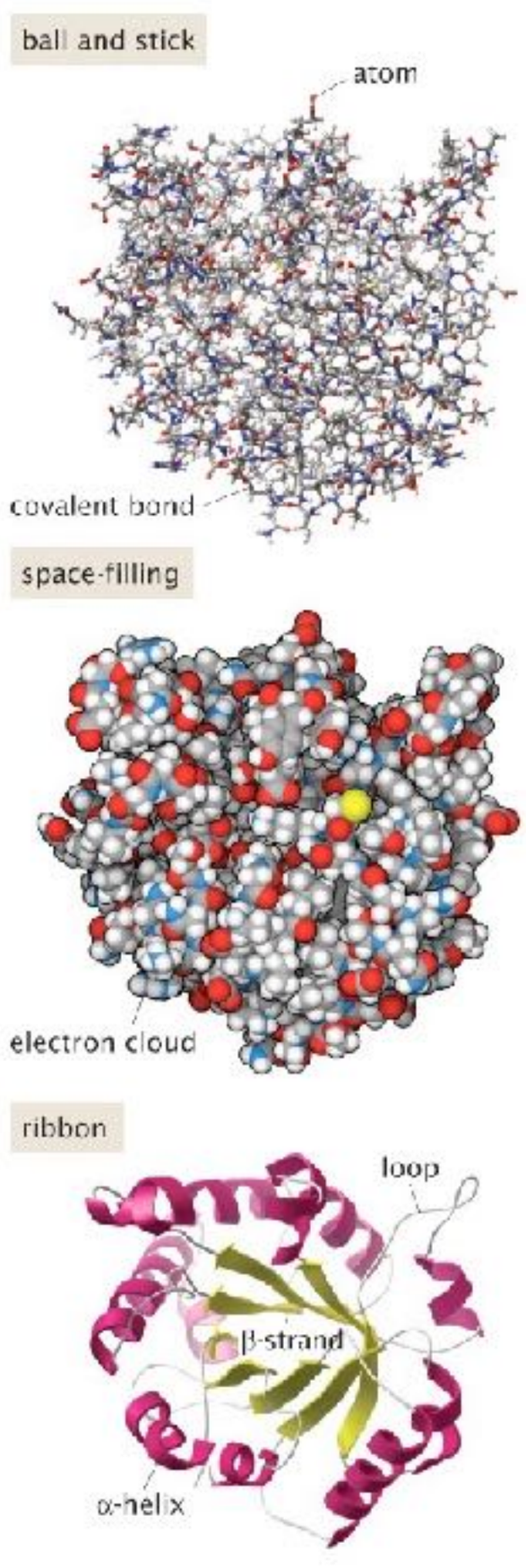
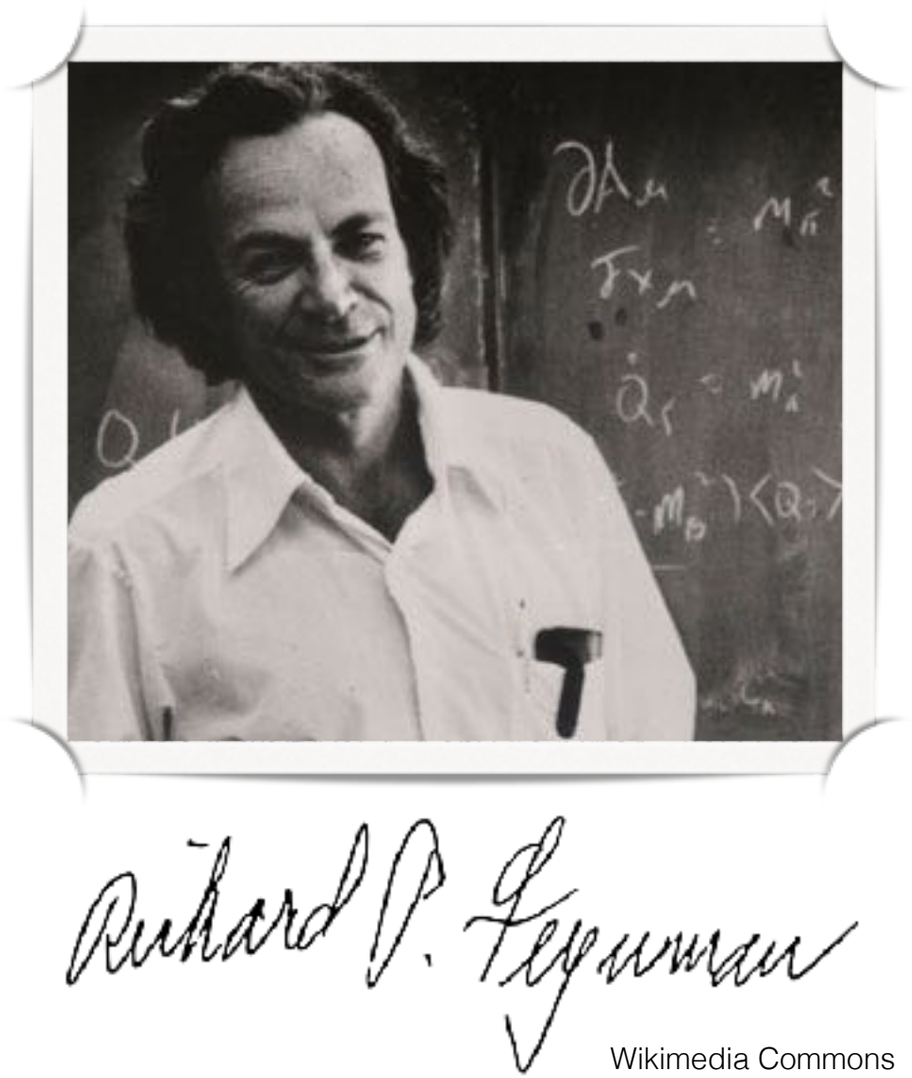


Figure 2.32 Physical Biology of the Cell, 2ed.

"Everything that living things do can be understood in terms of the jiggling and wiggling of atoms." —
Richard Feynman*



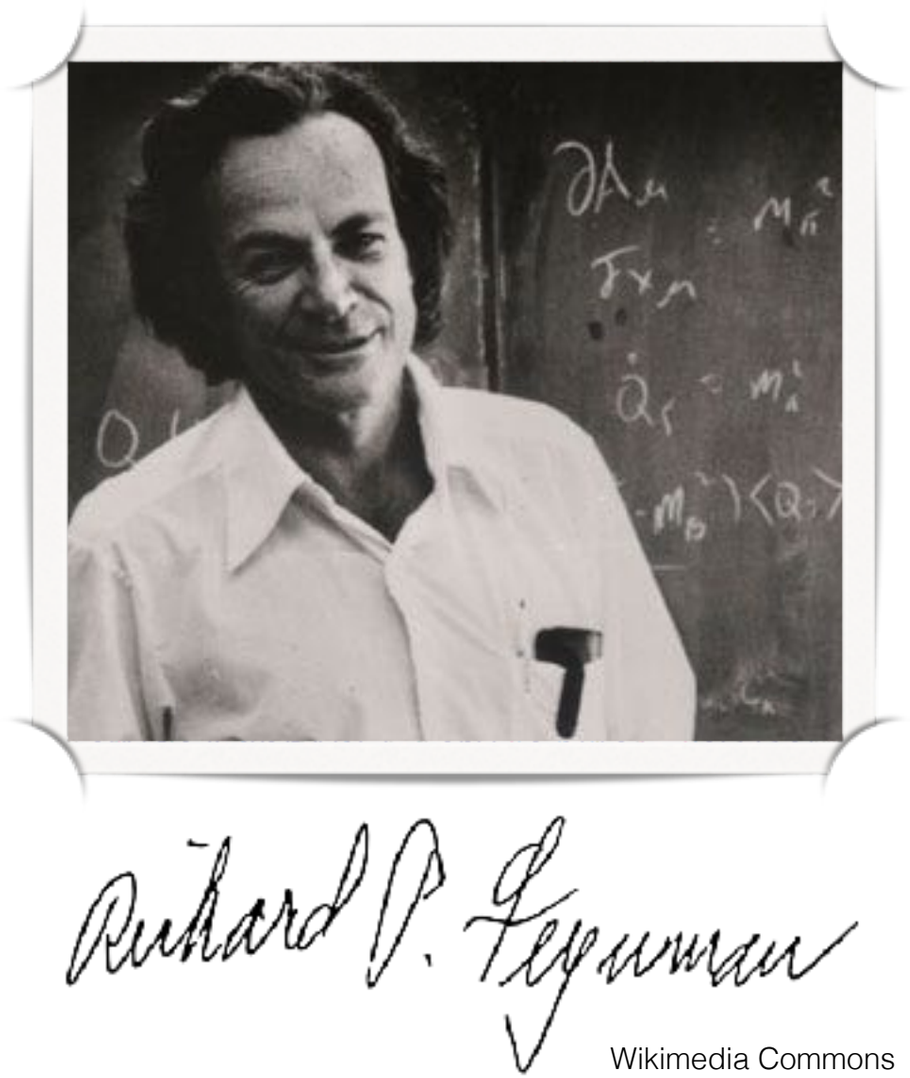
Wikimedia Commons

* R.P. Feynman. *The Feynman Lectures on Physics*. 1963

"Everything that living things do can be understood in terms of the jiggling and wiggling of atoms." —
Richard Feynman*

$$(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t))$$

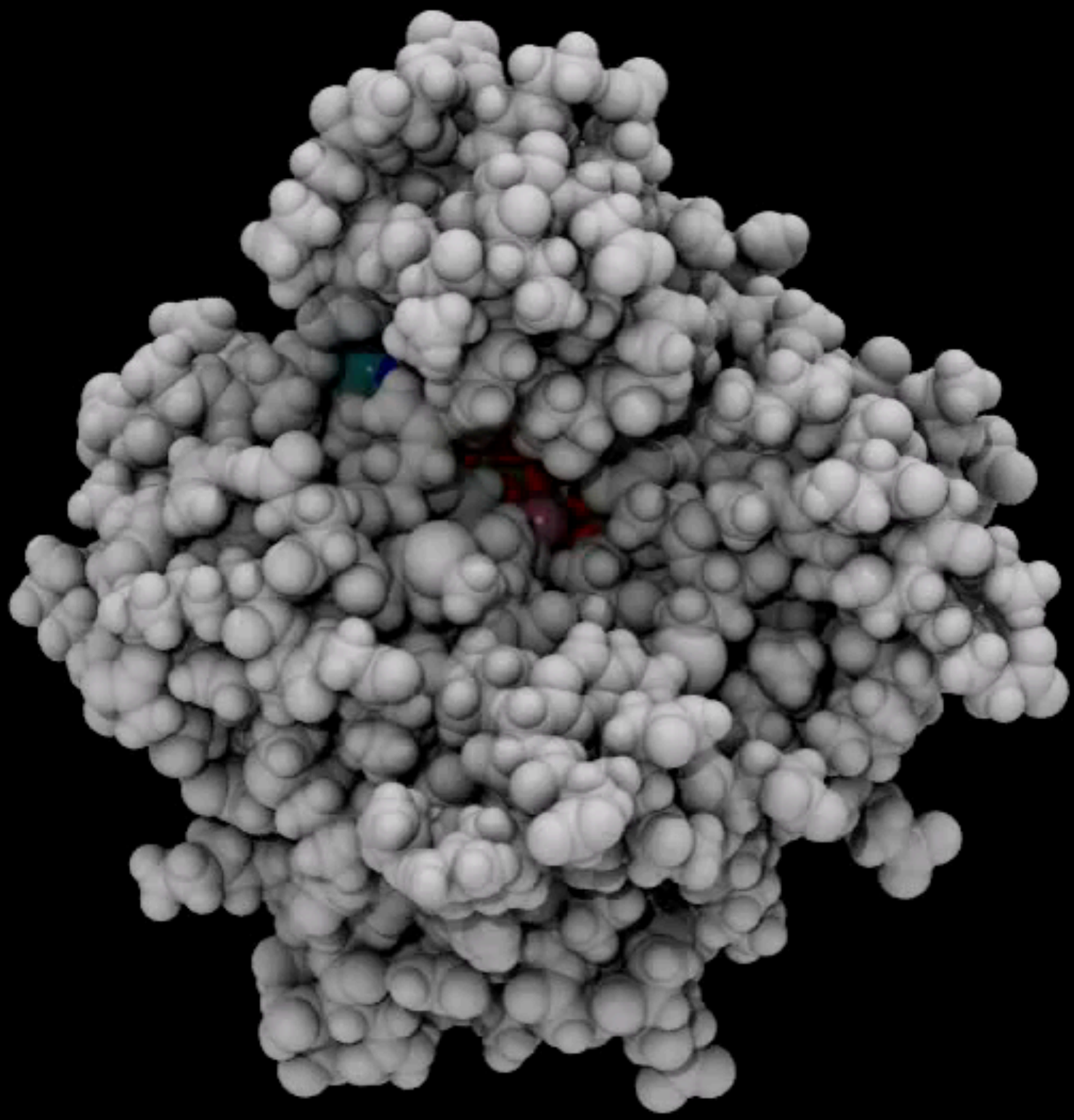
positions of all *N* atoms over time



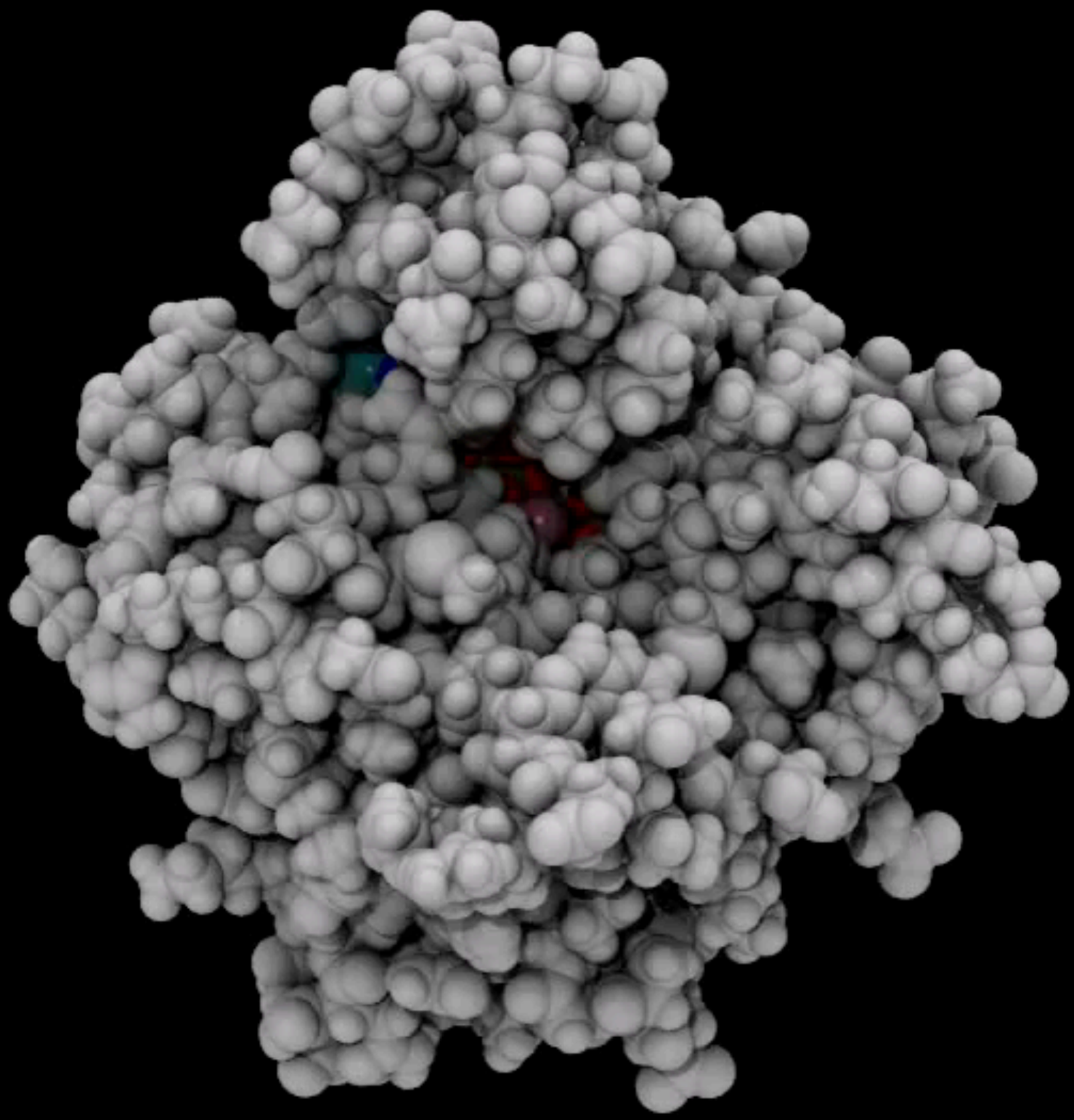
Wikimedia Commons

* R.P. Feynman. *The Feynman Lectures on Physics*. 1963

Molecular Dynamics (MD) Simulations



Molecular Dynamics (MD) Simulations



Molecular Dynamics (MD) Simulations
(classical)

Molecular Dynamics (MD) Simulations

(classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

energy function = "force field"

Molecular Dynamics (MD) Simulations

(classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

energy function = "force field"

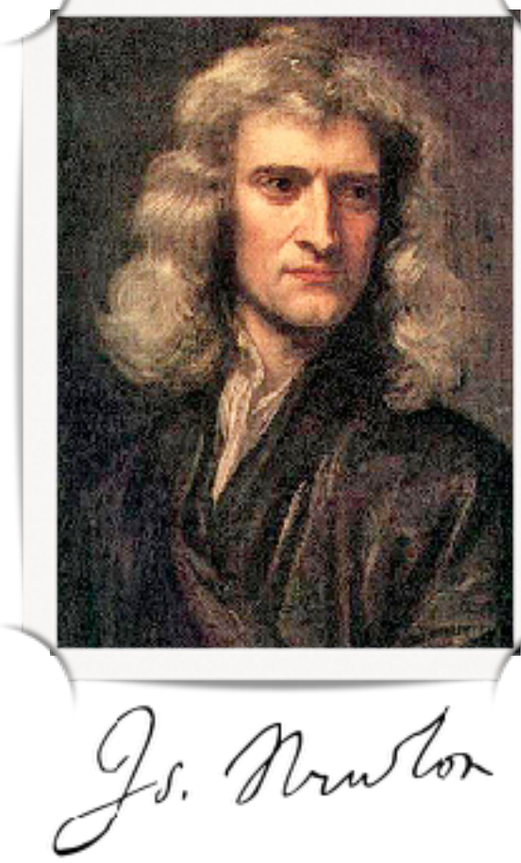
$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

Molecular Dynamics (MD) Simulations (classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

energy function = "force field"

$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$



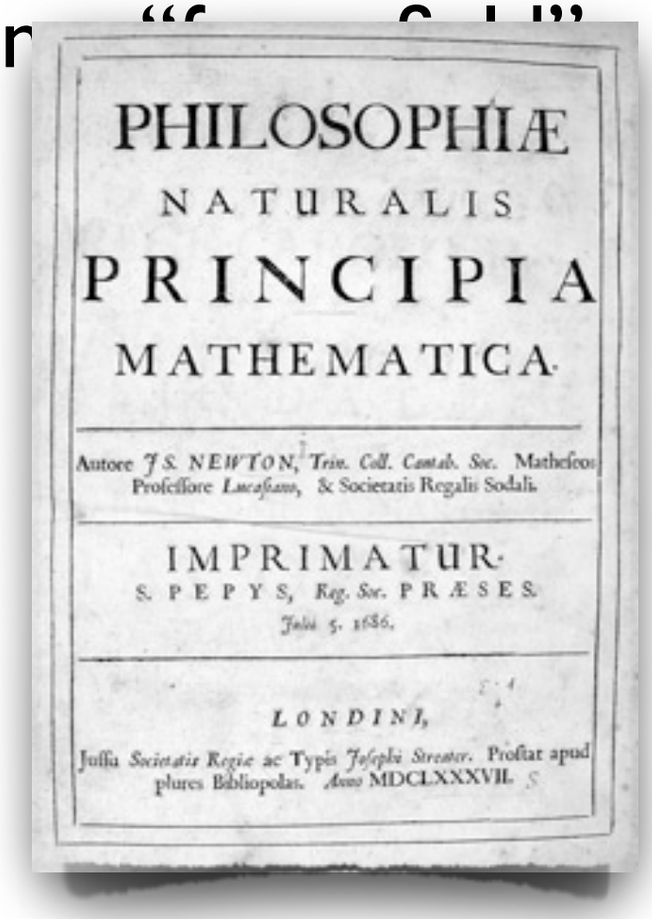
source: Wikimedia Commons

Molecular Dynamics (MD) Simulations (classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

energy functio[illegible]

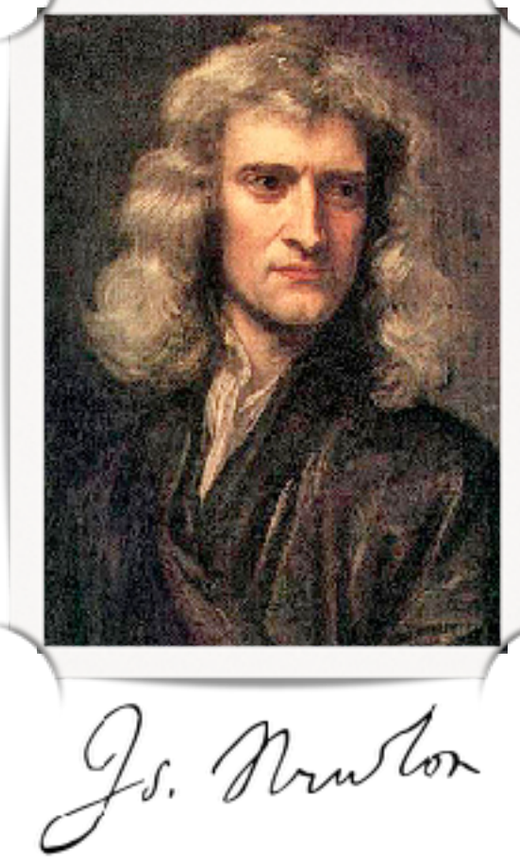
$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore J S. NEWTON, Trin. Coll. Cantab. Soc. Matheseos Professore Lucasiano, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, Reg. Soc. PRÆSES.
Julii 5. 1686.

LONDINI,
Justu Societatis Regiæ ac Typis Josephi Streater. Prostat apud plures Bibliopolas. Anno MDCLXXXVII.



source: Wikimedia Commons

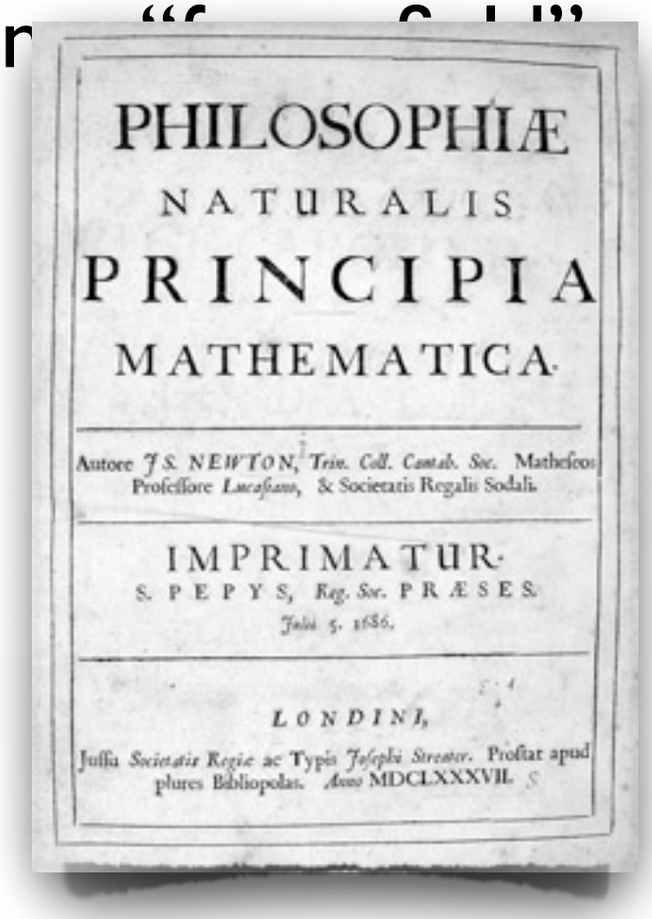
Molecular Dynamics (MD) Simulations (classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

← energy functio[illegible]

$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

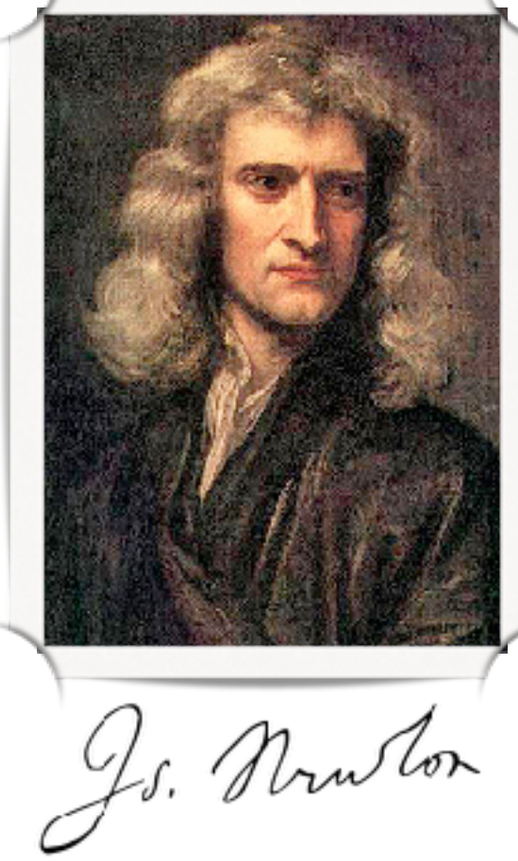
$$\mathbf{F}_i = m_i \mathbf{a}_i$$ ← Newton's 2nd law


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore *J.S. NEWTON*, *Trin. Coll. Cantab. Soc.* Matheseos Professore *Lucasiano*, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, *Reg. Soc.* PRÆSES.
Julii 5. 1686.

LONDINI,
Justu *Societatis Regiæ* ac Typis *Josephi Streater*. Prostat apud plures Bibliopolas. *Anno* MDCLXXXVII.



source: Wikimedia Commons

Molecular Dynamics (MD) Simulations (classical)

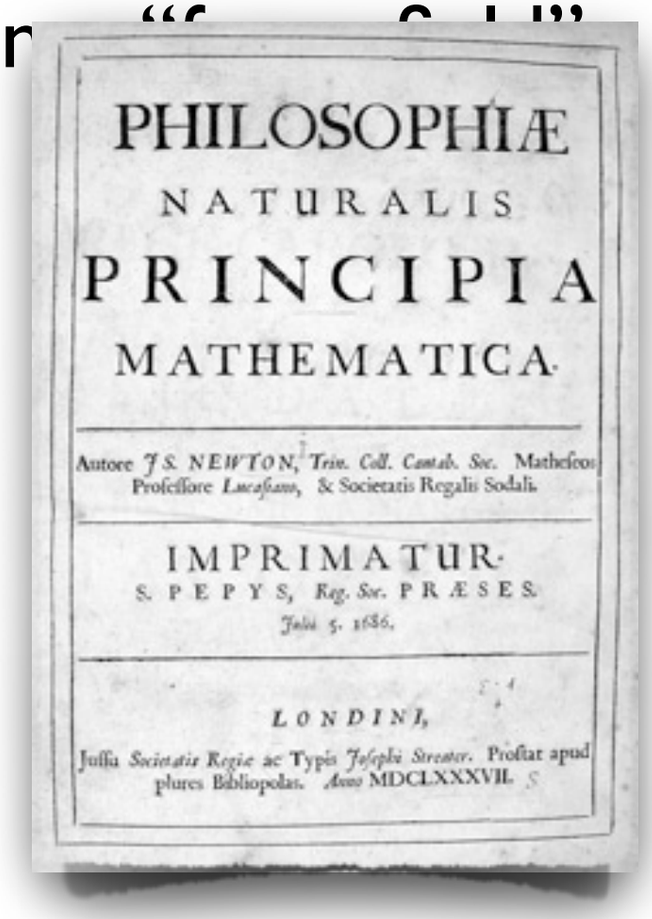
$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

← energy functio[illegible]

$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

$$\mathbf{F}_i = m_i \mathbf{a}_i$$ ← Newton's 2nd law

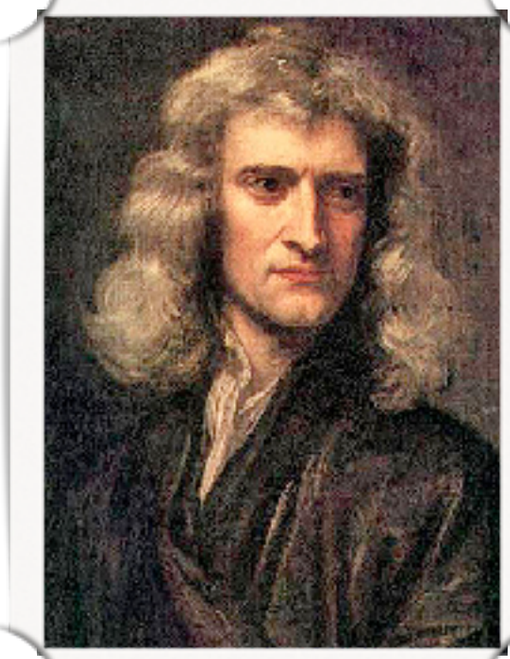
$$\frac{d^2 \mathbf{r}_i}{dt^2} = \frac{\mathbf{F}_i}{m_i}$$


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore *J.S. NEWTON*, *Trin. Coll. Cantab. Soc.* Matheseos Professore *Lucasiano*, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, *Reg. Soc.* PRÆSES.
Julii 5. 1686.

LONDINI,
Justu *Societatis Regiæ* ac Typis *Josephi Streater*. Prostat apud plures Bibliopolas. *Anno* MDCLXXXVII.



source: Wikimedia Commons

Molecular Dynamics (MD) Simulations (classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

← energy function

$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

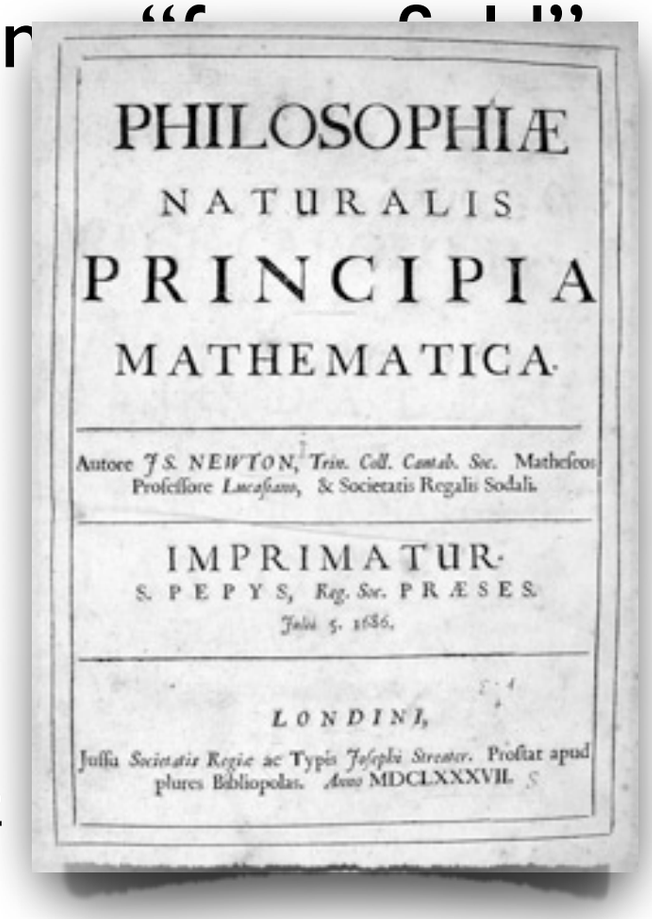
$$\mathbf{F}_i = m_i \mathbf{a}_i$$

← Newton's 2nd law

$$\frac{d^2\mathbf{r}_i}{dt^2} = \frac{\mathbf{F}_i}{m_i}$$

→ integrator

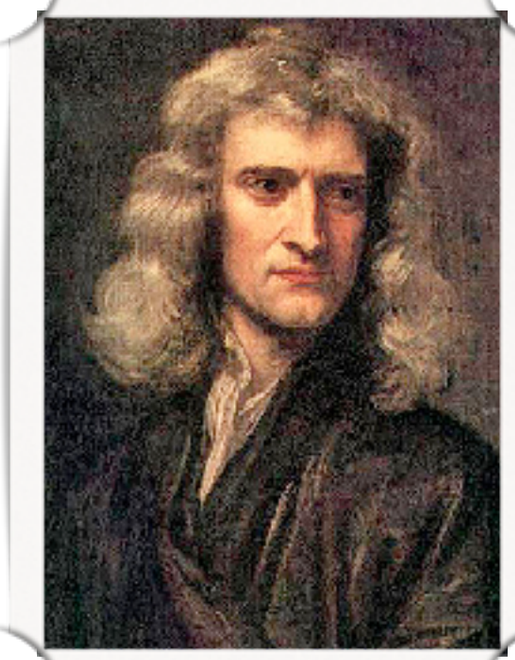
$$\mathbf{r}_i(t + \Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t - \Delta t) + \frac{\mathbf{F}_i}{m_i}\Delta t^2$$


PHILOSOPHIÆ
NATURALIS
PRINCIPIA
MATHEMATICA.

Autore *J.S. NEWTON*, *Trin. Coll. Cantab. Soc.* Matheseos Professore *Lucasiano*, & Societatis Regalis Sodali.

IMPRIMATUR.
S. PEPYS, *Reg. Soc.* PRÆSES.
Julii 5. 1686.

LONDINI,
Juſſu *Societatis Regiæ* ac Typis *Joſephi Streater*. Proſtat apud plures Bibliopolas. *Anno* MDCLXXXVII.



source: Wikimedia Commons

Molecular Dynamics (MD) Simulations (classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U$$

energy func

$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

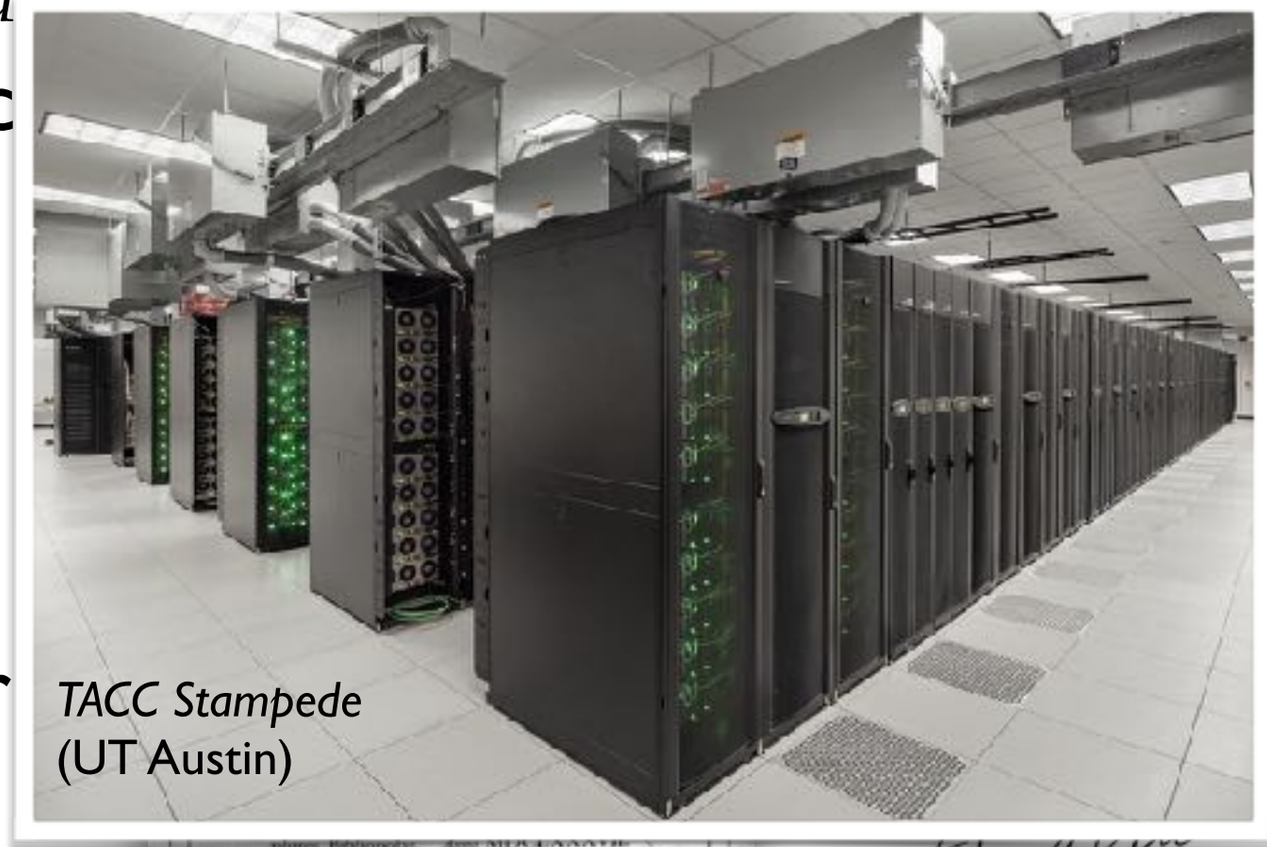
$$\mathbf{F}_i = m_i \mathbf{a}_i$$

Newton's 2nd law

$$\frac{d^2 \mathbf{r}_i}{dt^2} = \frac{\mathbf{F}_i}{m_i}$$

integrator

$$\mathbf{r}_i(t + \Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t - \Delta t) + \frac{\mathbf{F}_i}{m_i} \Delta t^2$$



TACC Stampede
(UT Austin)

source: Wikimedia Commons

Molecular Dynamics (MD) Simulations

(classical)

$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U$$

energy func

$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

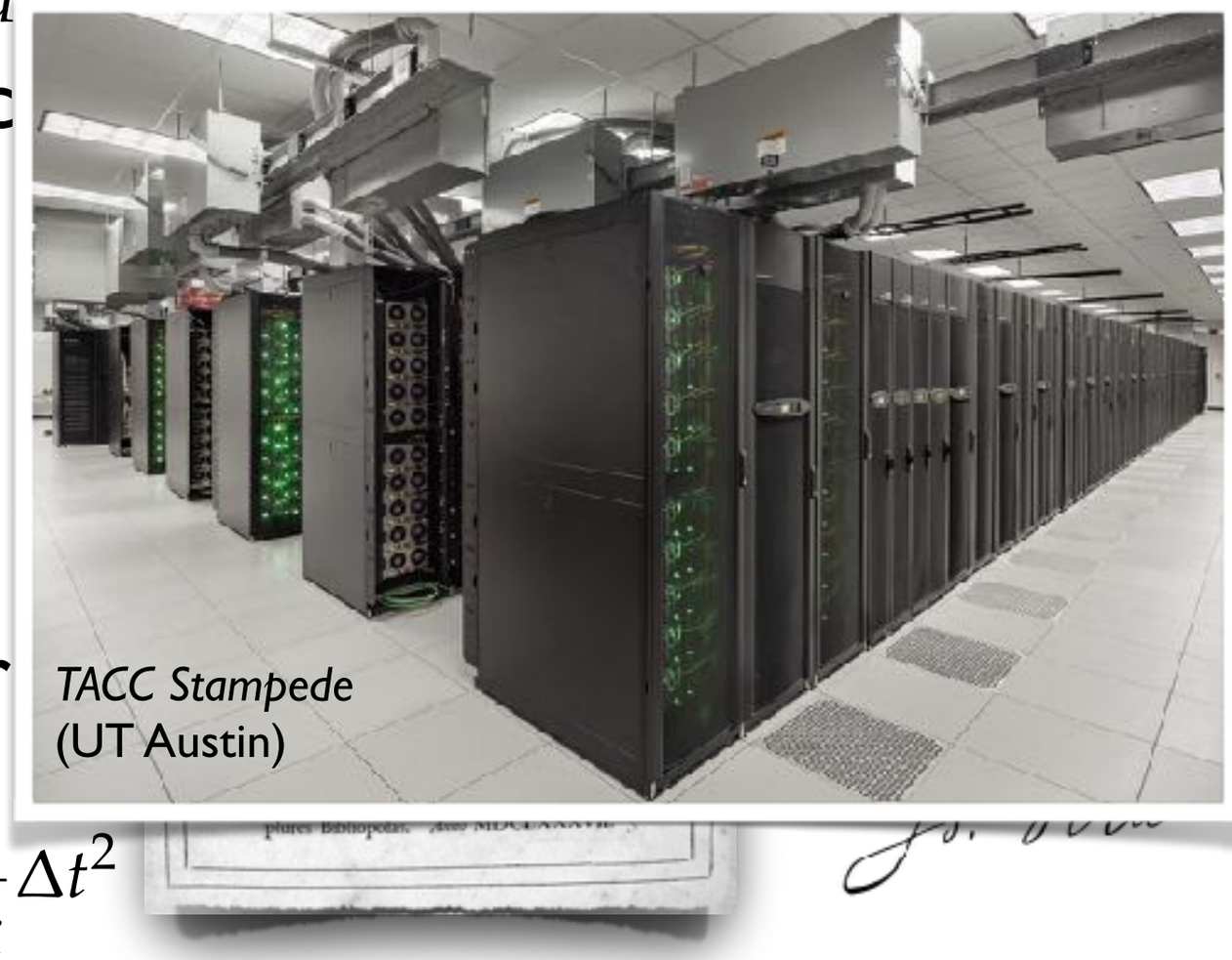
$$\mathbf{F}_i = m_i \mathbf{a}_i$$

Newton's 2nd law

$$\frac{d^2 \mathbf{r}_i}{dt^2} = \frac{\mathbf{F}_i}{m_i}$$

integrator

$$\mathbf{r}_i(t + \Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t - \Delta t) + \frac{\mathbf{F}_i}{m_i} \Delta t^2$$



TACC Stampede
(UT Austin)

$$\mathbf{r}_1(0), \ldots, \mathbf{r}_N(0)$$
$$\mathbf{r}_1(\Delta t), \ldots, \mathbf{r}_N(\Delta t)$$
$$\mathbf{r}_1(2\Delta t), \ldots, \mathbf{r}_N(2\Delta t)$$
$$\mathbf{r}_1(3\Delta t), \ldots, \mathbf{r}_N(3\Delta t)$$
$$\vdots$$

trajectory

$$\big(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t)\big)$$
$$0 \le t \le \tau$$

source: Wikimedia Commons

Computing observables

- average over *estimator* function to get observable (phase space/"ensemble" average)

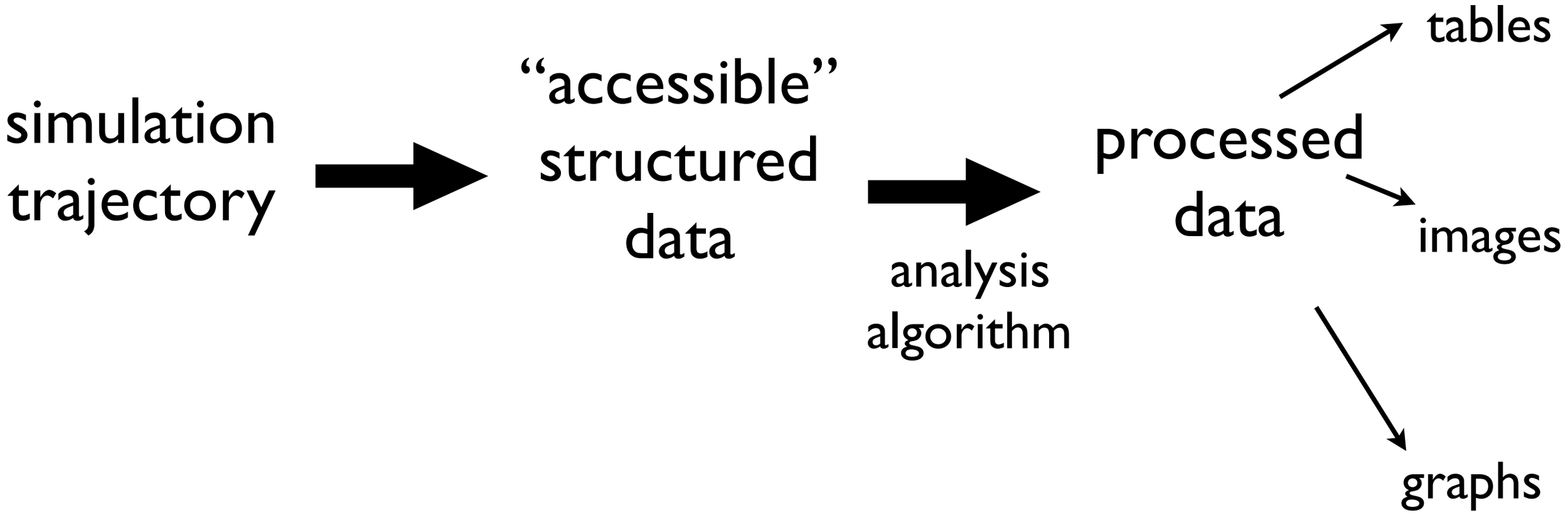
$$A = \langle \mathcal{A} \rangle = \int d\mathbf{x}_1 d\mathbf{x}_2 \dots d\mathbf{x}_N \, p(\mathbf{x}_1, \dots, \mathbf{x}_N) \mathcal{A}(\mathbf{x}_1, \dots, \mathbf{x}_N)$$

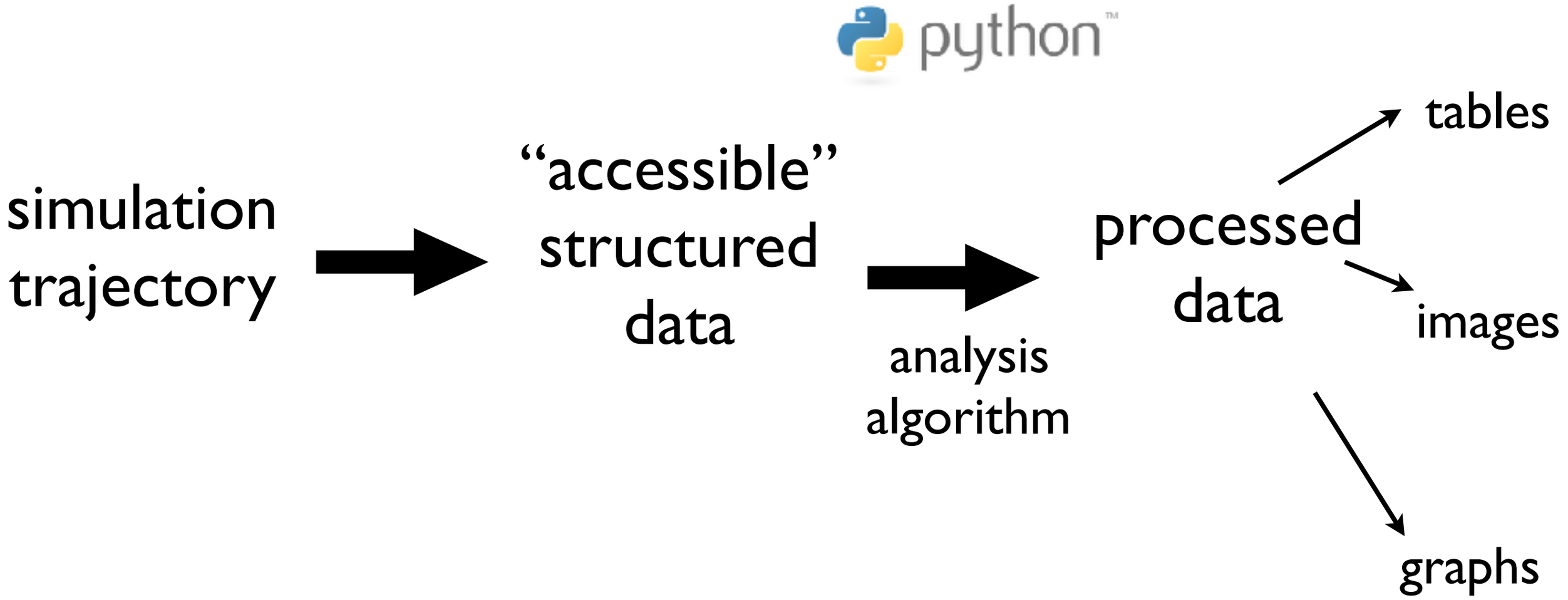
$$= \int d\mathbf{x}_1 d\mathbf{x}_2 \dots d\mathbf{x}_N \, e^{-U(\mathbf{x}_1,\dots,\mathbf{x}_N)/kT} \mathcal{A}(\mathbf{x}_1, \dots, \mathbf{x}_N)$$

- ergodic hypothesis: time average = ensemble average

$$\overline{\mathcal{A}(x_t)} = \lim_{\tau \to \infty} \frac{1}{\tau} \int_0^\tau dt \mathcal{A}(x_t) \approx \frac{1}{N} \sum_{t=0}^{N} \mathcal{A}(x_t)$$

$$\langle \mathcal{A}(x) \rangle = \overline{\mathcal{A}(x_t)}$$


simulation
trajectory
"accessible"
structured
data
analysis
algorithm
processed
data
tables
images
graphs


python
simulation trajectory
"accessible" structured data
analysis algorithm
processed data
tables
images
graphs

Basic Idea: Use NumPy

NumPy python™

simulation trajectory → "accessible" structured data → processed data → tables, images, graphs

analysis algorithm

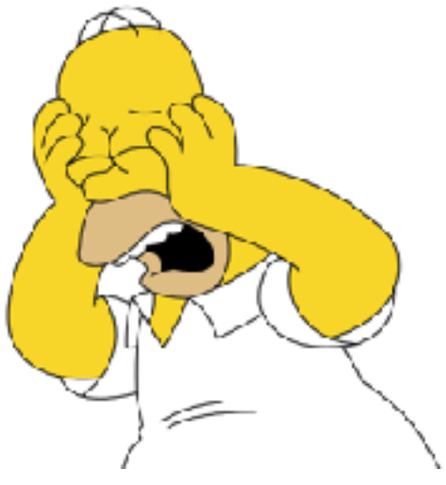
Basic Idea: Use NumPy

NumPy python™

simulation trajectory

dcd, xtc, trr, ncdf, trj, pdb, pqr, gro, crd, dms, trz, mol2, xyz, config, history, gms, ...

psf, tpr, prmtop, dms, mol2, hoomd xml, ...



Oh nooooo!

"accessible" structured data

analysis algorithm

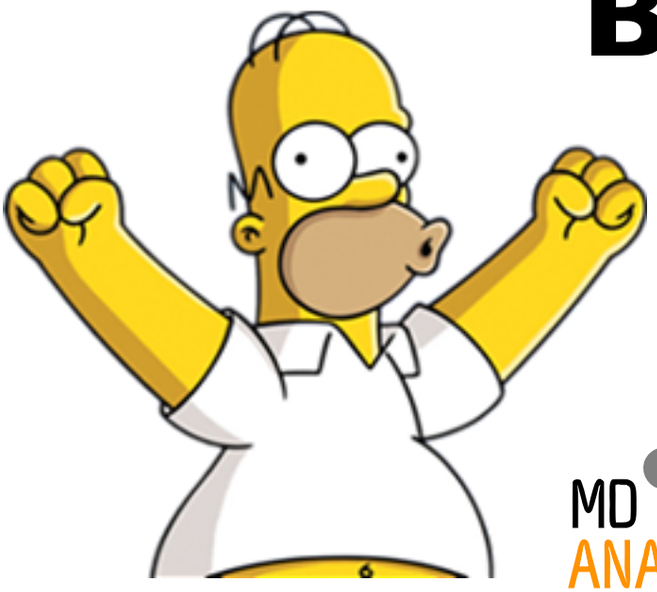
processed data

tables

images

graphs

Woo-hoo!



Basic Idea: Use NumPy

MD ANALYSIS

NumPy

python™

simulation trajectory

dcd, xtc, trr, ncdf, trj, pdb, pqr, gro, crd, dms, trz, mol2, xyz, config, history, gms, ...

psf, tpr, prmtop, dms, mol2, hoomd xml, ...

"accessible" structured data

analysis algorithm

processed data

tables

images

graphs

Oh nooooo!

Table of supported coordinate formats

Name	extension	IO	remarks
CHARMM, NAMD	dcd	r/w	standard CHARMM binary trajectory; endianness is autodetected. Fixed atoms may not be handled correctly (requires testing). Module `MDAnalysis.coord`
LAMMPS	dcd	r/w	CHARMM-style bina autodetected. Units Module `MDAnalysi`
LAMMPS [1]	data	r	Single frame of coo
LAMMPS [1]	lammpsdump	r	Ascii trajectory in a
Gromacs	xtc	r/w	Compressed (lossy) `MDAnalysis.coord`
Gromacs	trr	r/w	Full precision trr tra velocities are proce `MDAnalysis.coord`
XYZ [1]	xyz	r/w	Generic white-spac compressed (gzip o `MDAnalysis.coord`
TXYZ [1]	txyz, arc	r	Tinker XYZ format

Table of Supported Topology Formats

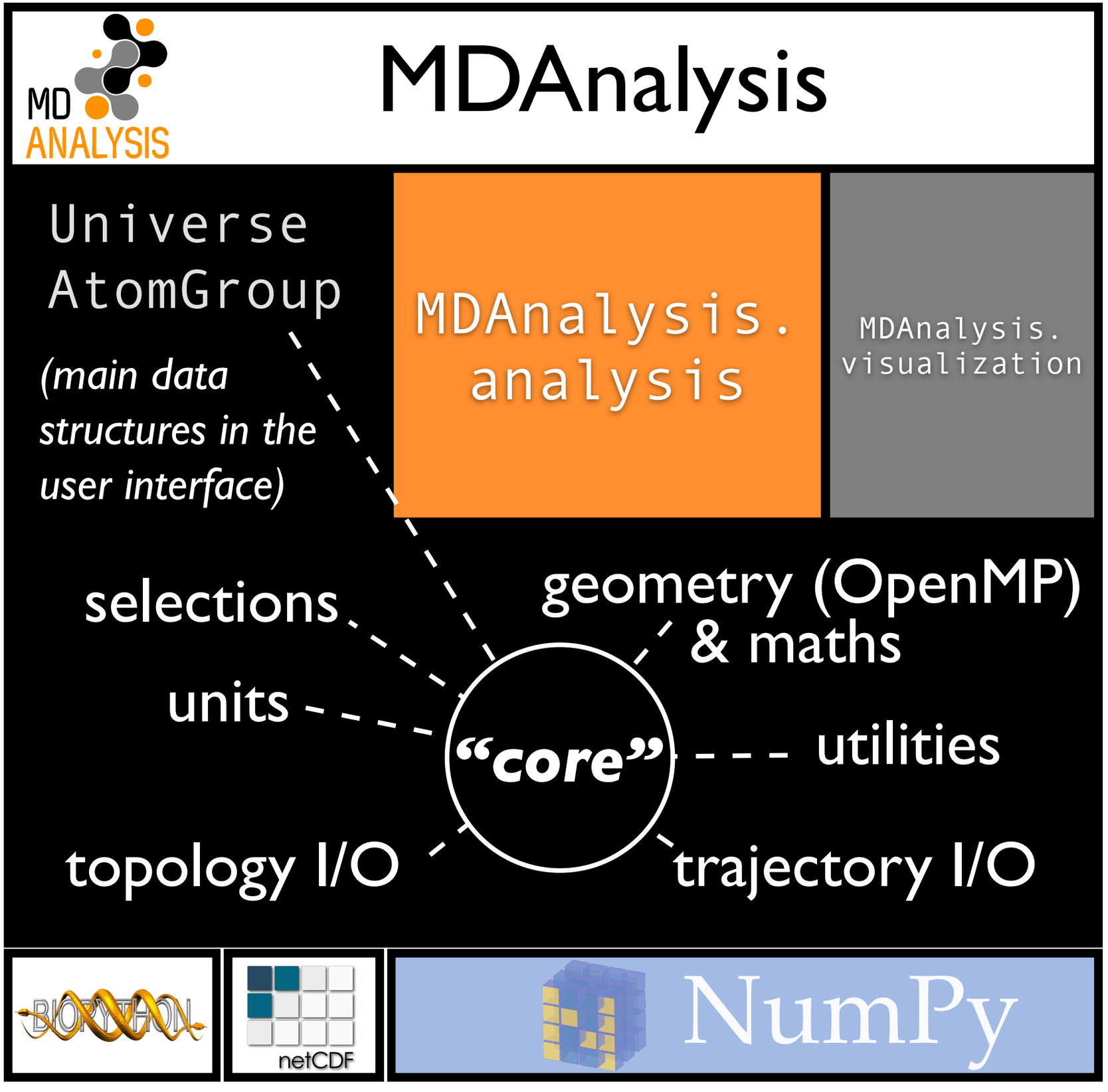
Name	extension	attributes	remarks
CHARMM/XPLOR PSF	psf	resnames, names, types, charges, bonds, angles, dihedrals, impropers	`MDAnalysis.topology.PSFParser`
CHARMM CARD [1]	crd	names, tempfactors, resnames,	"CARD" coordinate output from CHARMM; deals with either standard EXTended format; `MDAnalysis.topology.CRDParser`
Brookhaven [1]	pdb/ent	names, bonds, resids, resnums, types, chainids, occupancies, bfactors, resids,	a simplified PDB format (as used in N simulations) is read by default

25 coordinate formats

20 topology formats

(v0.19.2)

MDAnalysis



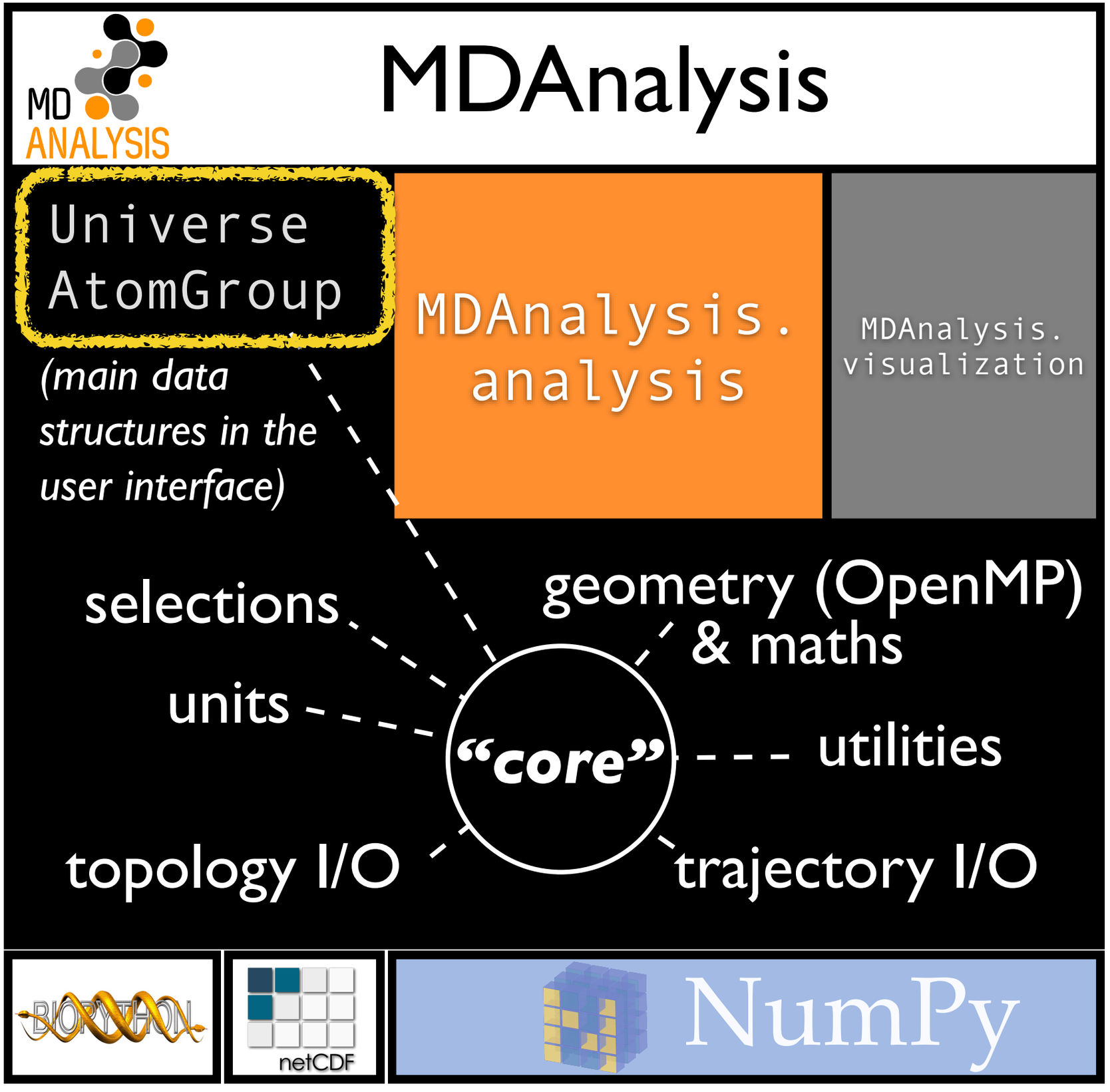
https://mdanalysis.org

Code base:

- python 3 & 2.7
- cython
- C
- ~63k LOC
- ~39k lines comments

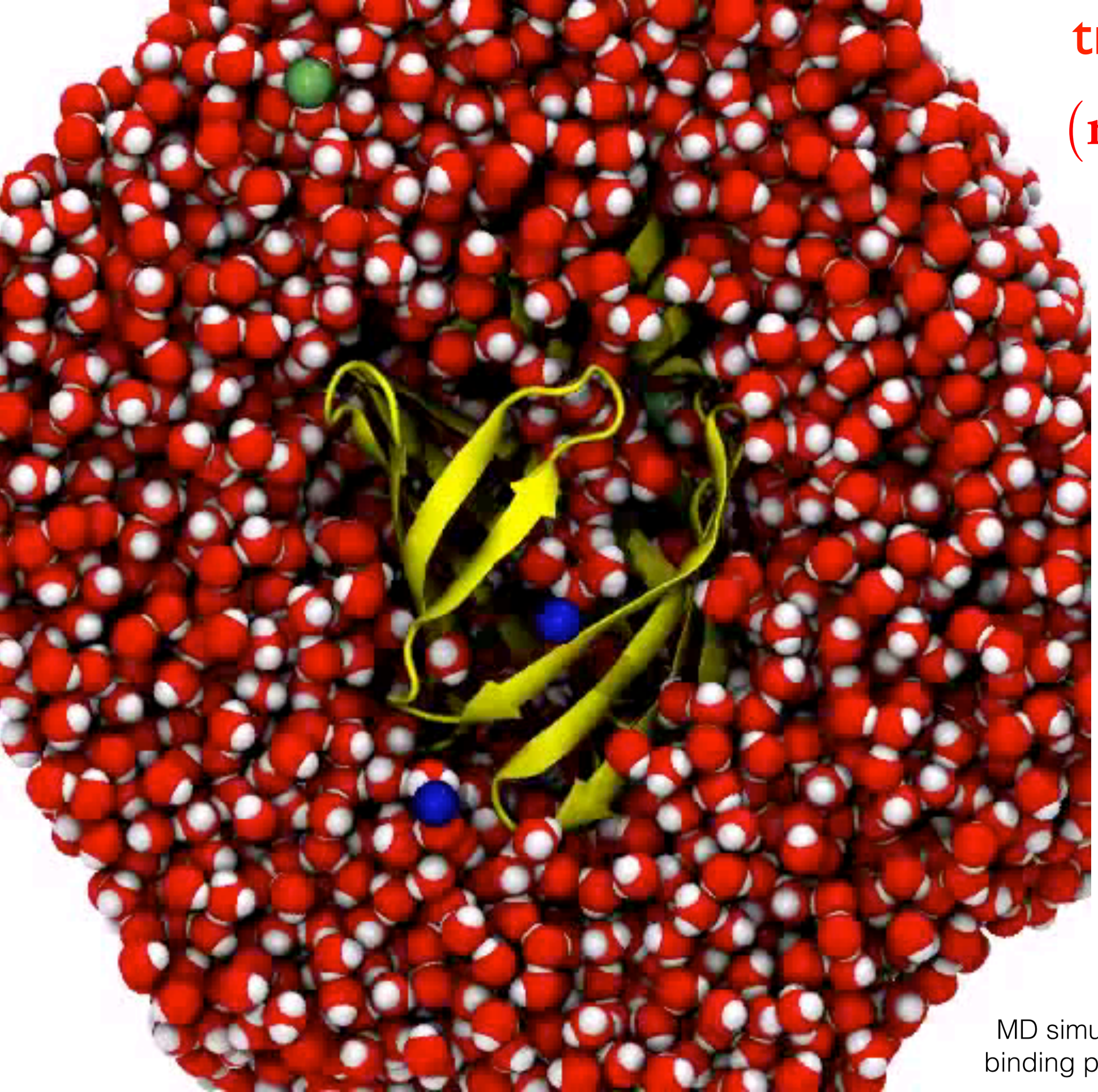
MDAnalysis

https://mdanalysis.org



Code base:

- python 3 & 2.7
- cython
- C
- ~63k LOC
- ~39k lines comments



trajectory

$$\left(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t)\right)$$

$$0 \le t \le \tau$$

1 protein

12 ions

3432 waters

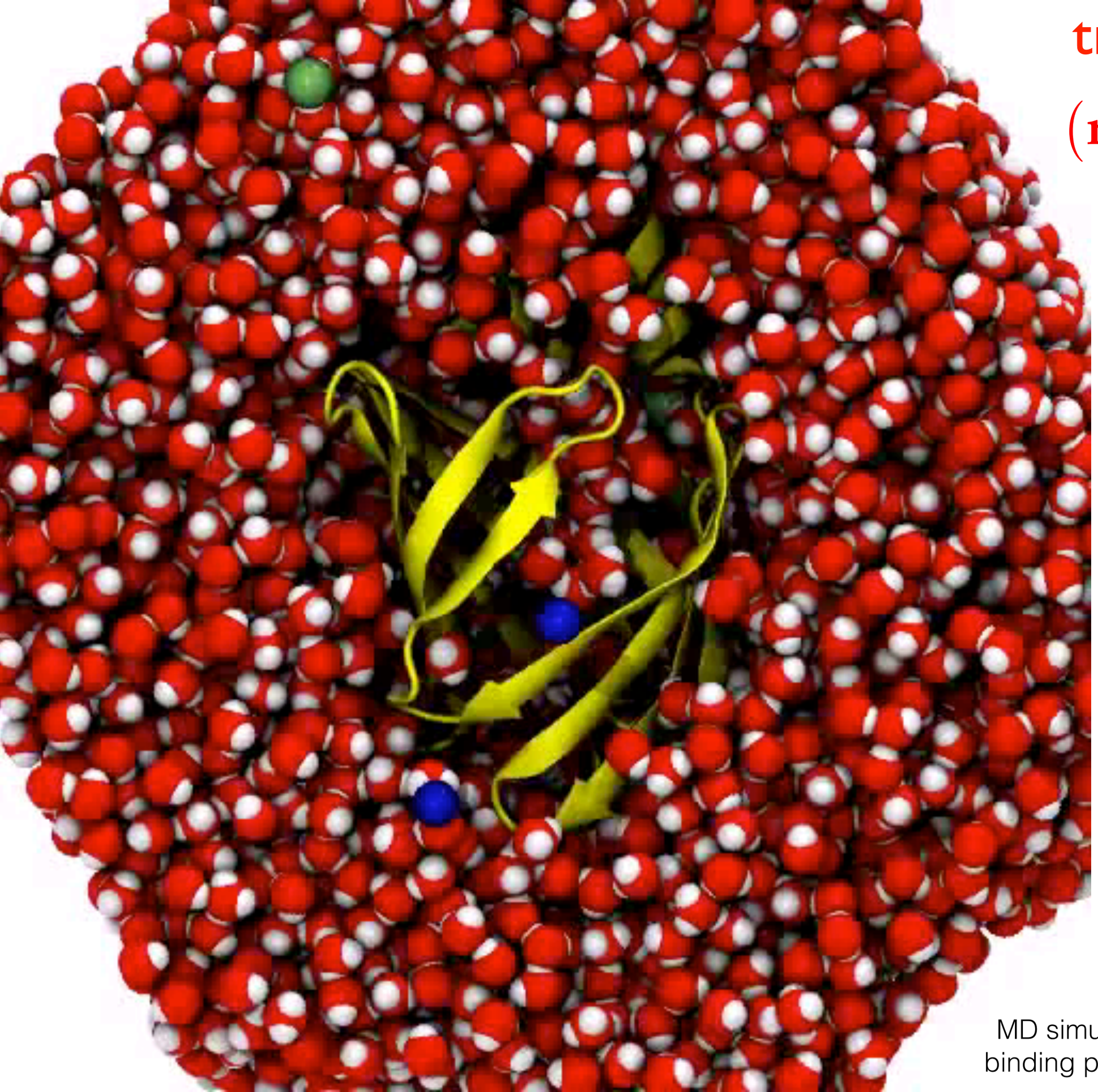
2113 atoms

12 atoms

10296 atoms

12421 atoms

MD simulation of intestinal fatty acid binding protein. Rendered with VMD.



trajectory

$$\left(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t)\right)$$

$$0 \le t \le \tau$$

1 protein

12 ions

3432 waters

2113 atoms

12 atoms

10296 atoms

12421 atoms

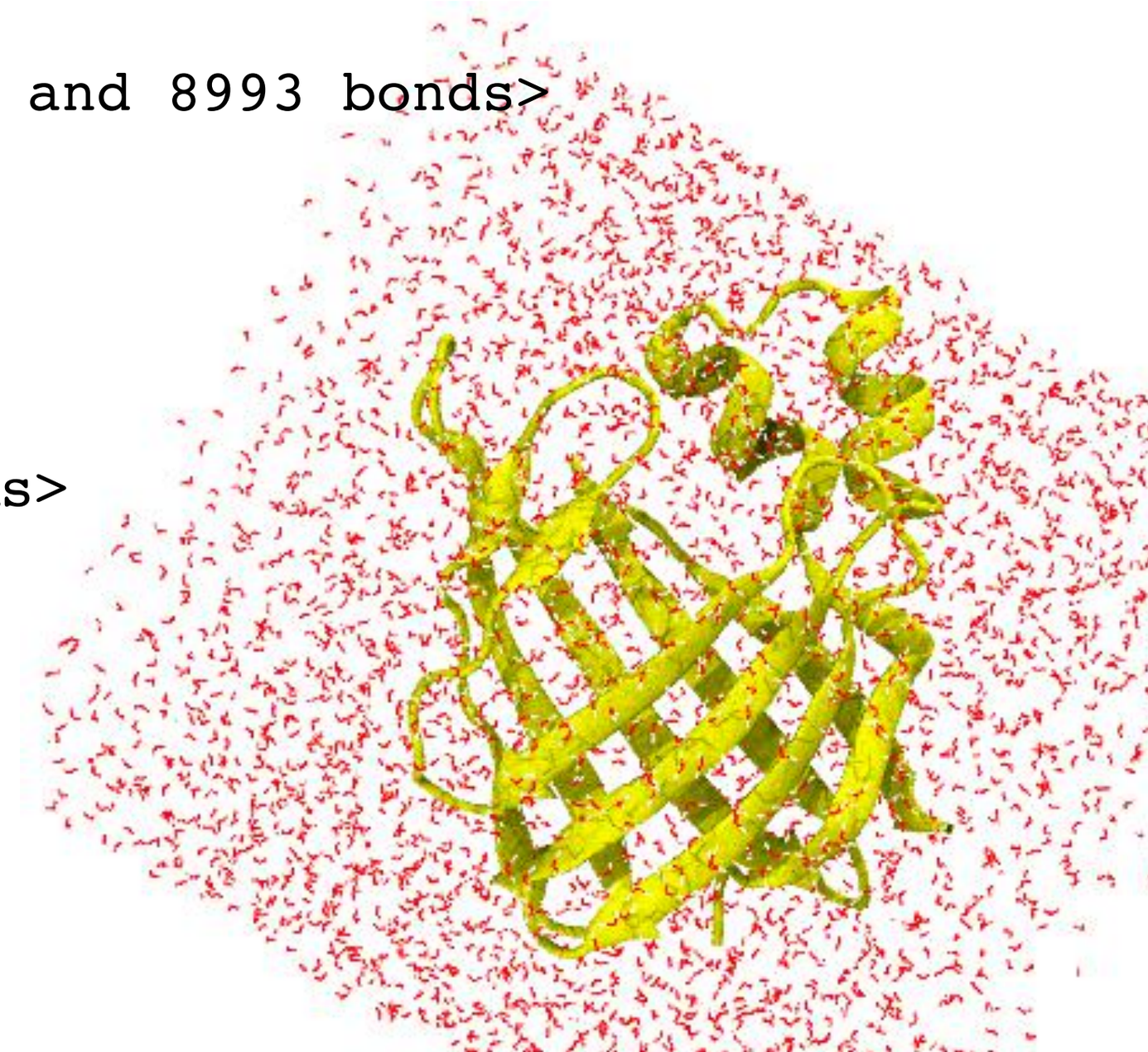
MD simulation of intestinal fatty acid binding protein. Rendered with VMD.

Fundamental data structures: *Universe*

```
import MDAnalysis as mda
u = mda.Universe(topology, trajectory)
```

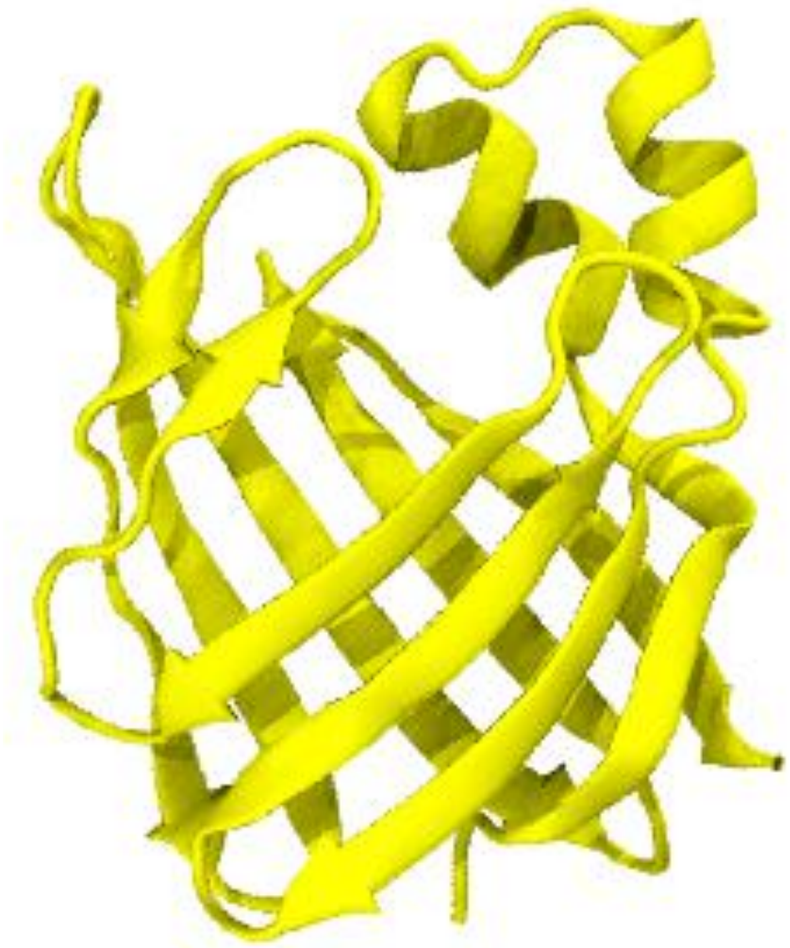
```
print(u)
<Universe with 12421 atoms and 8993 bonds>
```

```
u.atoms
<AtomGroup with 12421 atoms>
```



Fundamental data structures: *AtomGroup*

```
protein =
   u.atoms.select_atoms(
      “protein”)
```



```
protein
```

```
<AtomGroup with 2113 atoms>
```

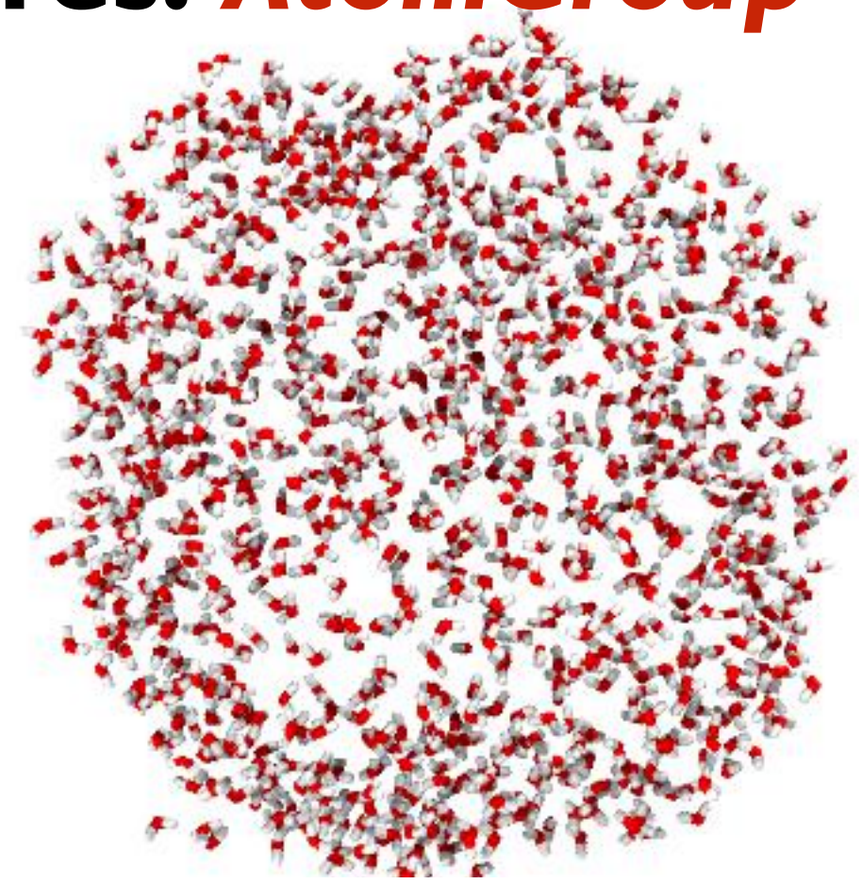
```
print(protein[:5])
```

```
<AtomGroup
[<Atom 1: N of type NH3 of resname ALA, resid 1 and segid IFAB>,
 <Atom 2: HT1 of type HC of resname ALA, resid 1 and segid IFAB>,
 <Atom 3: HT2 of type HC of resname ALA, resid 1 and segid IFAB>,
 <Atom 4: HT3 of type HC of resname ALA, resid 1 and segid IFAB>,
 <Atom 5: CA of type CT1 of resname ALA, resid 1 and segid IFAB>]>
```

Fundamental data structures: *AtomGroup*

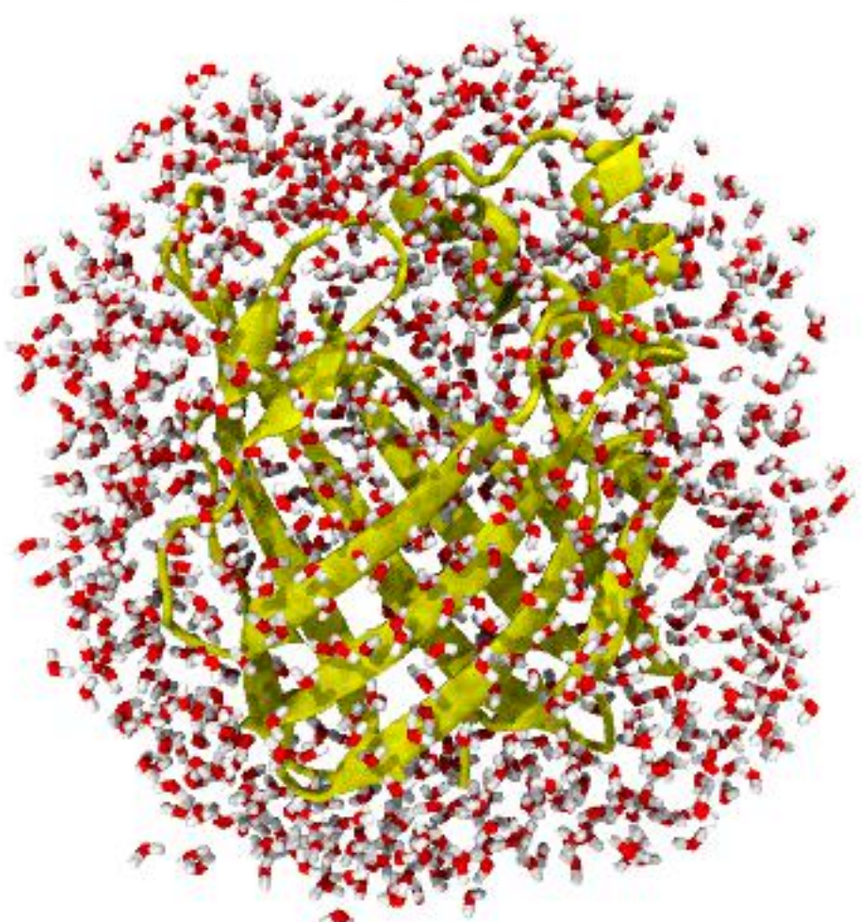
```
solvshell =
   u.atoms.select_atoms(
      “resname TIP3 and
       around 5.0 protein”)

solvshell
```

```
<AtomGroup with 3868 atoms>
```



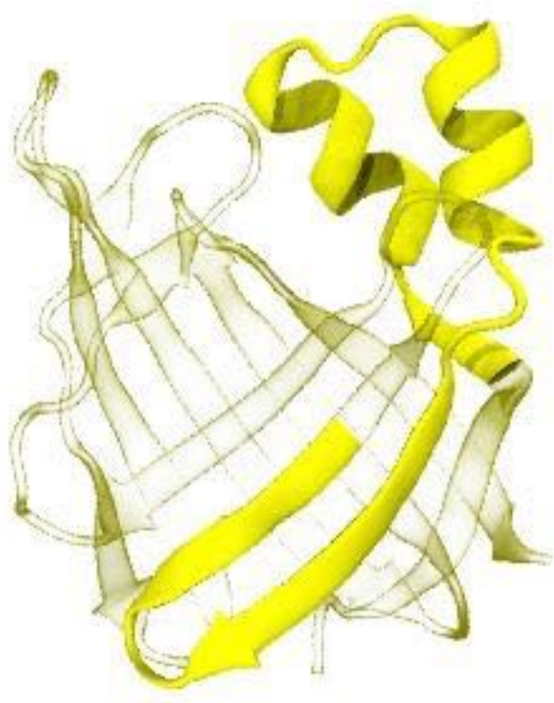
```
ag = protein + solvshell
ag
```

```
<AtomGroup with 5981 atoms>
```



AtomGroups: Residues and Segments

```
protein.residues[10:50]
```

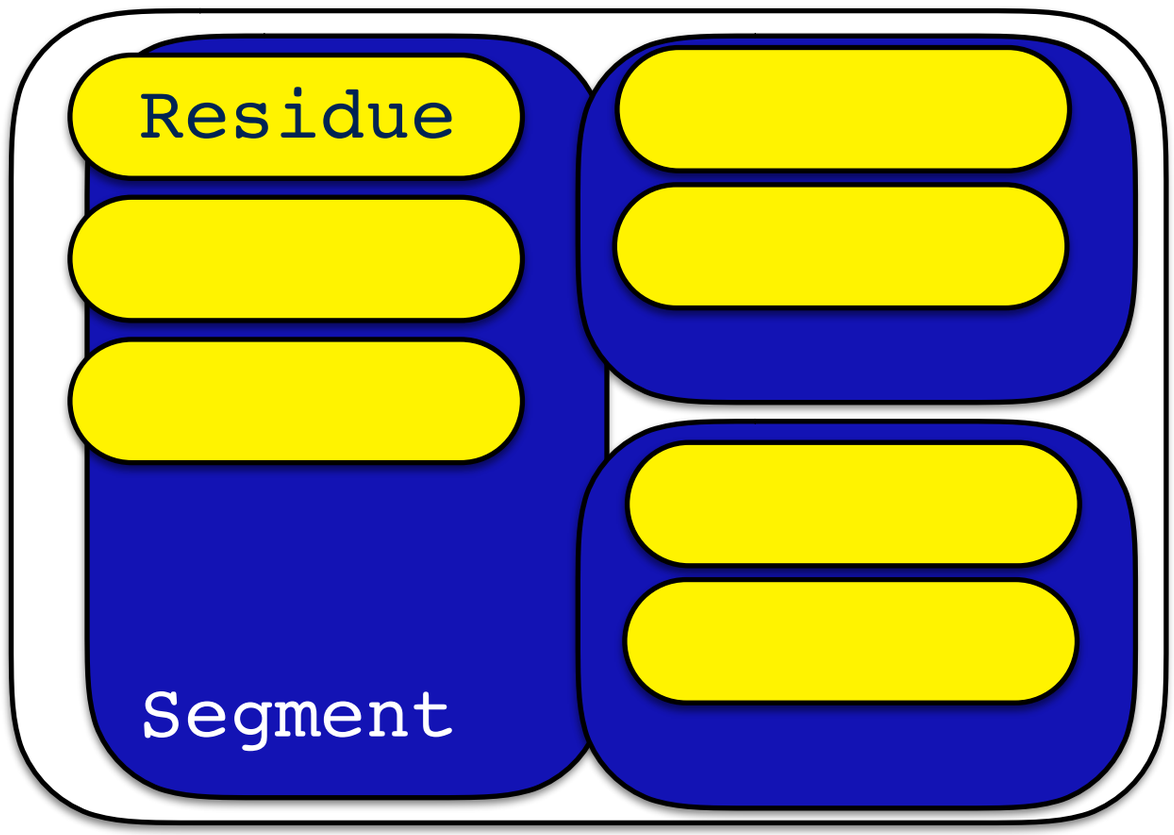


```
print(protein.residues[10:50])
```

```
<ResidueGroup [<Residue ASN, 11>,
<Residue GLU, 12>, <Residue ASN, 13>,
<Residue TYR, 14>, <Residue GLU, 15>,
…, <Residue LYS, 50>]>
```

```
print(protein.segments)
```

```
<SegmentGroup [<Segment IFAB>]>
```



Atom data as NumPy arrays

AtomGroups contain particles (“atoms”).

Properties of all particles are NumPy arrays:

`ag.names`

```
array(['N', 'HT1', 'HT2', ..., 'OH2', 'H1', 'H2'],
      dtype='|S4')
```

`ag.charges`

```
array([-0.3  ,  0.33 ,  0.33 ,
       ...,
       -0.834,  0.417,  0.417])
```

`ag.positions`

$(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t))$

```
array([[-12.57699966,  10.42199993,  -5.22900009],
       [-13.59200001,  10.19900036,  -5.19299984],
       [-12.31599998,  10.22900009,  -6.21700001],
       ...,
       [ -5.02600002, -12.31200027,  13.30200005],
       [ -5.45100021, -11.82499981,  12.59500027],
       [ -4.14099979, -12.47900009,  12.97900009]],
      dtype=float32)
```

`ag.velocities`
`ag.forces`

… and many more

Basic analysis pattern: Iterate over frames

- trajectories contain *frames*: one snapshot of all particles at a specific time (*positions*[, *velocities*[, *forces*]])
- `Universe.trajectory` is iterable

```
for ts in u.trajectory[::10]:
    analyze(ag.positions)
```

updates every step

- random access to a specific frame

  `u.trajectory[72]`

- number of frames

  `len(u.trajectory)`

Example analysis: Per-residue RMSF

- root mean square fluctuation ρ_i (RMSF) measures local flexibility of amino acid *i*
- standard quantity to compute for protein simulations

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$

- use the "C-alpha" atom in each residue to characterise the motion of the whole residue: $\boldsymbol{x}_i$ (position of $C_{\alpha,i}$)

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$


```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

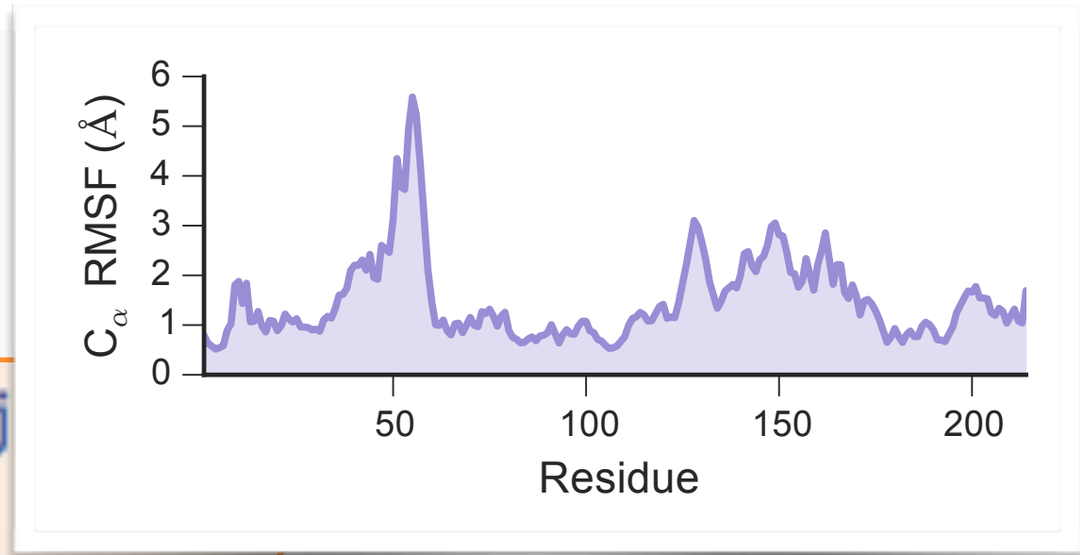
C_α RMSF

$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$

MD ANALYSIS

NumPy

```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```



simulation trajectory

dcd, xtc, trr, ncdf, trj, pdb, pqr, gro, crd, dms, trz, mol2, xyz, config, history, gms, ...

psf, tpr, prmtop, dms, mol2, hoomd xml, ...

25 different formats — one analysis script

"accessible" structured data

analysis algorithm

```
import numpy as np
import MDAnalysis as mda

u = mda.Universe("topol.tpr", "trj.xtc")
ca = u.select_atoms("name CA")
means = np.zeros((len(ca), 3))
sumsq = np.zeros_like(means)
for k, ts in enumerate(u.trajectory):
    sumsq += (k/(k+1.0)) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1.0)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1.0))

matplotlib.pyplot.plot(ca.residues.resids, rmsf)
```

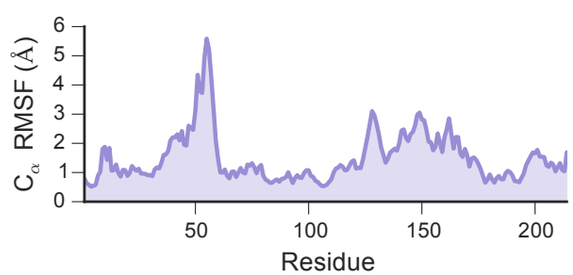
$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$

C_α RMSF

processed data

- tables
- images
- graphs

RESULTS!



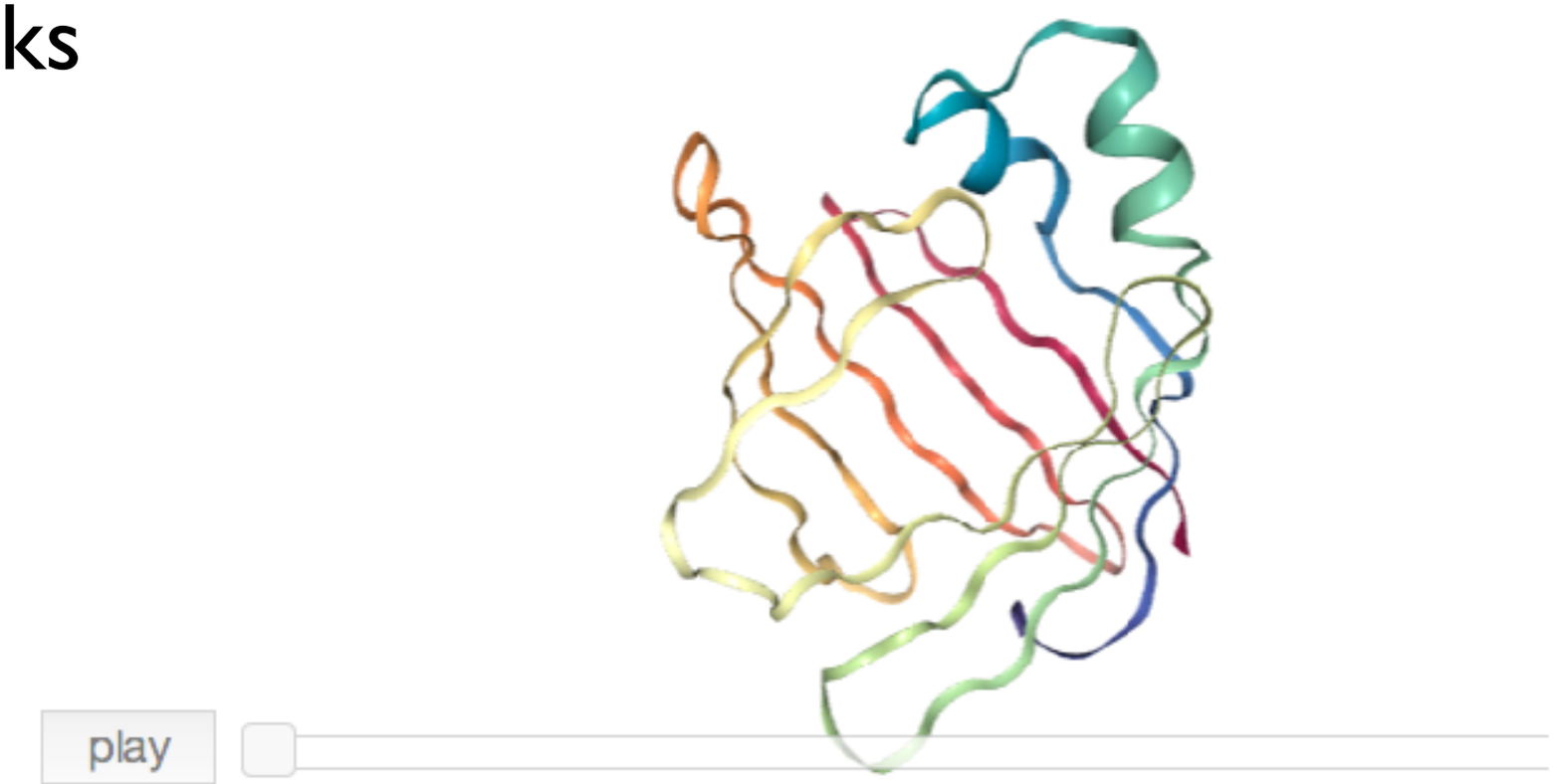
Interactive use

```
In [1]:
1 import MDAnalysis as mda
2 from MDAnalysisData import datasets as data
3 import nglview as nv
```

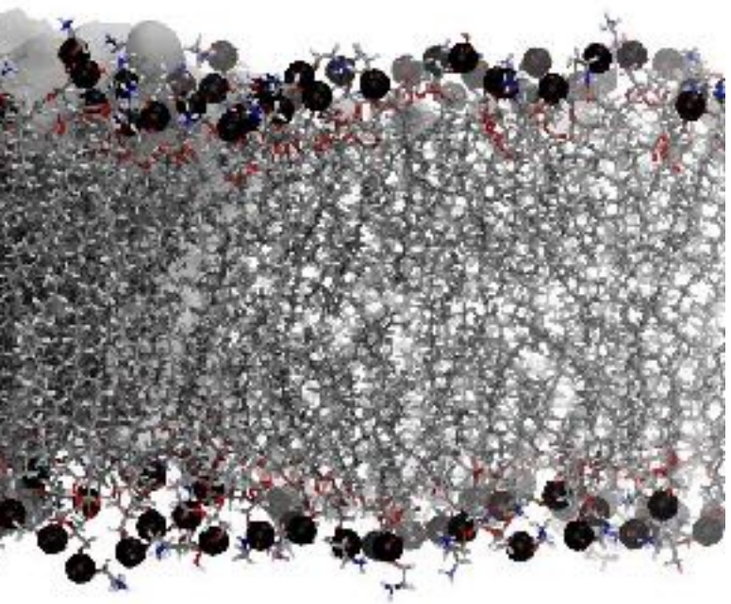
```
In [2]:
1 ifabp = data.fetch_ifabp_water()
2 u = mda.Universe(ifabp.structure, ifabp.trajectory)
3 protein = u.select_atoms("protein")
```

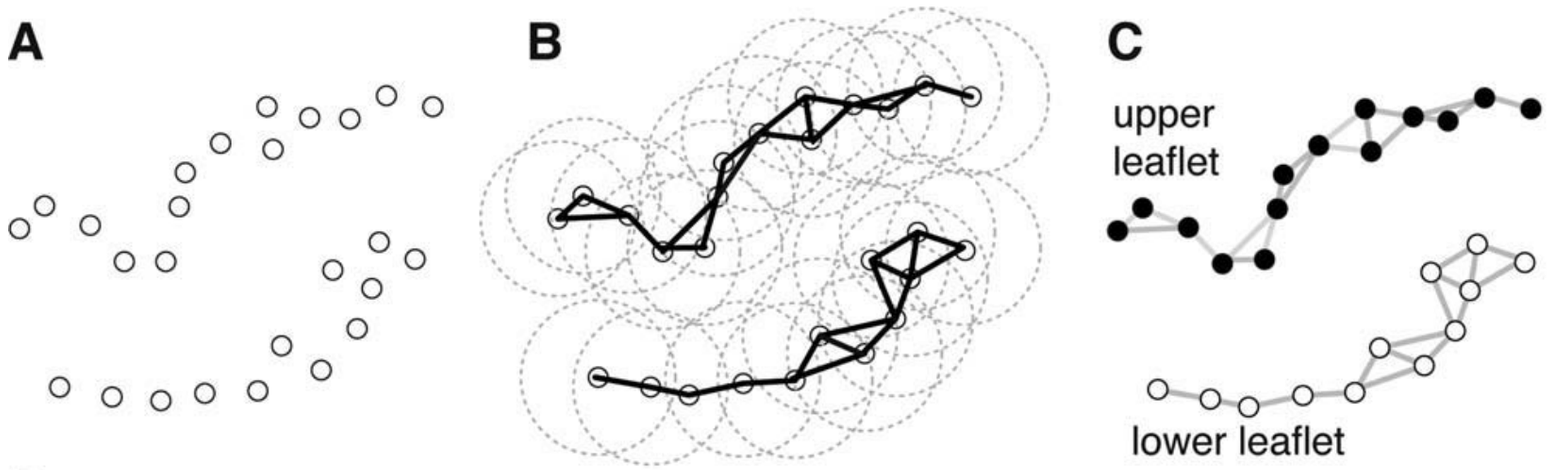
```
In [3]:
1 w = nv.show_mdanalysis(protein)
2 w
```

- **Jupyter** notebooks (+ **pandas**, …)
- visualisation with **nglviewer**



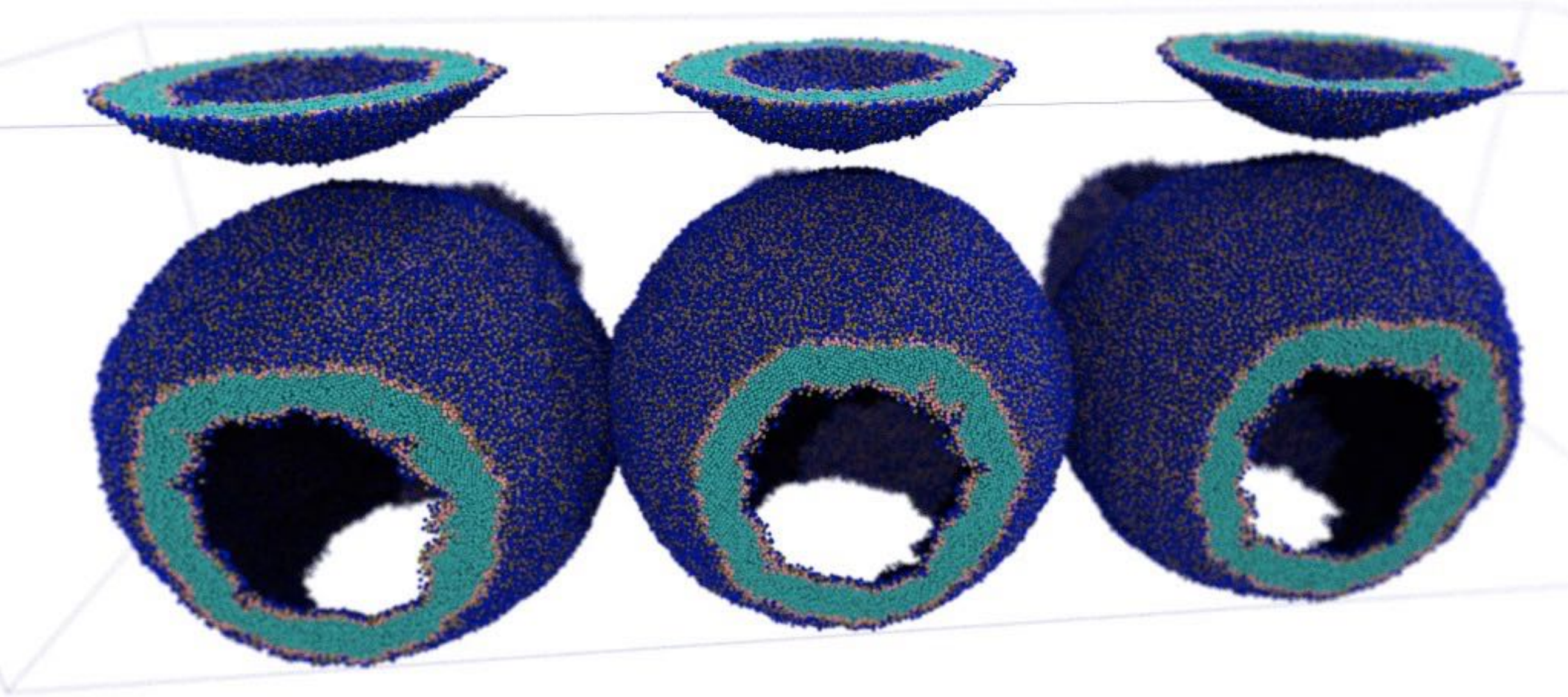
Interoperability with the Python Ecosystem

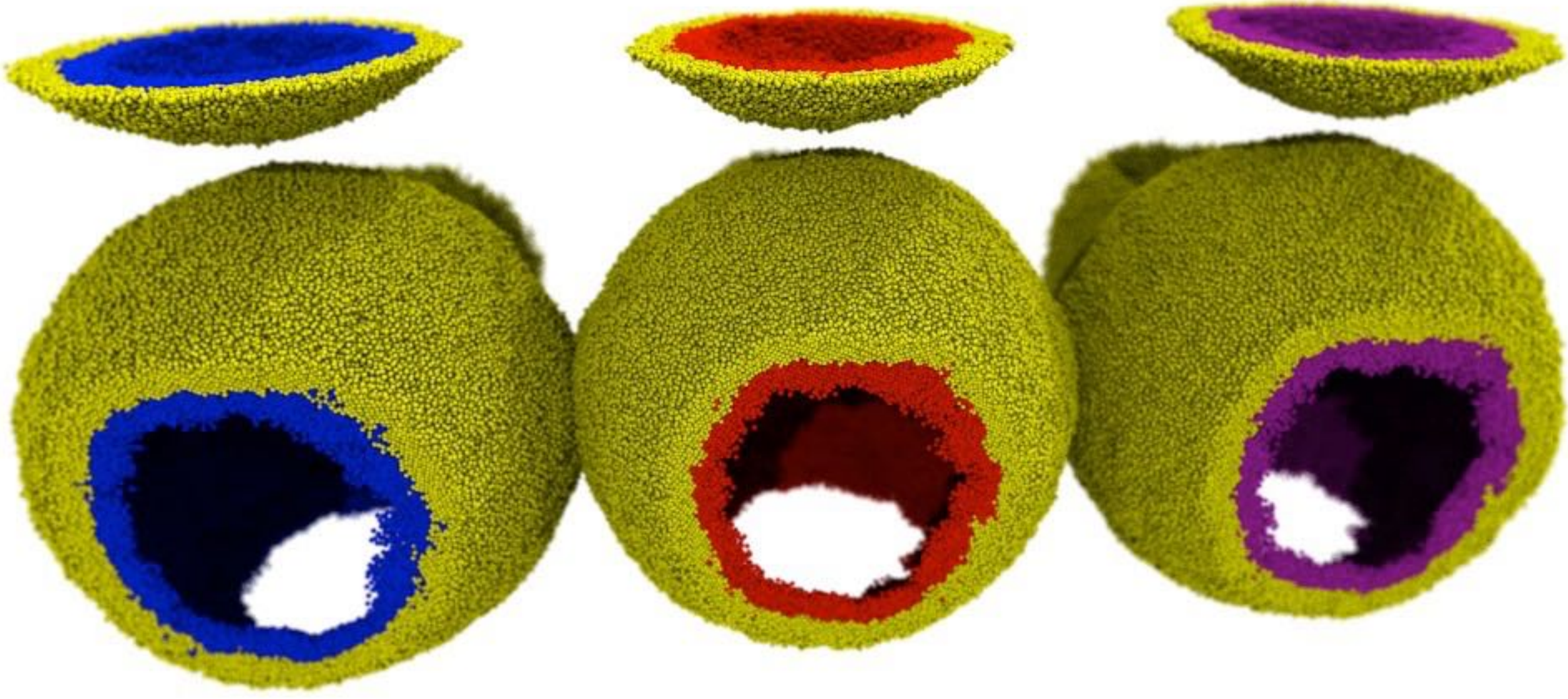


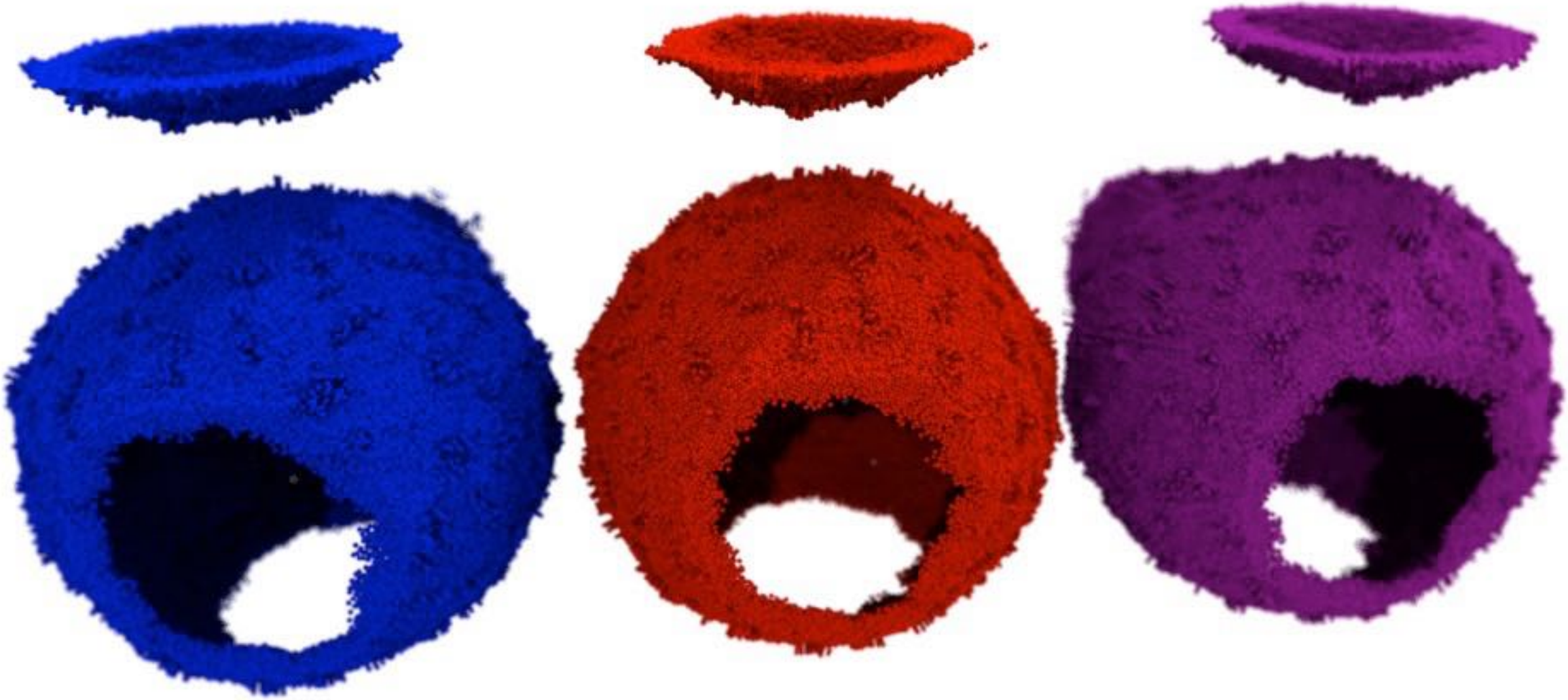


LeafletFinder algorithm

```
import MDAnalysis as mda
import networkx as nx
from MDAnalysis.lib.distances import distance_array

u = mda.Universe(pdb, xtc)
headgroup_atoms = u.select_atoms("name P*")
A x = headgroup_atoms.positions

B adj = (distance_array(x, x) < 12)
C leaflets = sorted(nx.connected_components(nx.Graph(adj)), key=len, reverse=True)

A_lipids = headgroup_atoms[leaflets[0]].residues
B_lipids = headgroup_atoms[leaflets[1]].residues
```





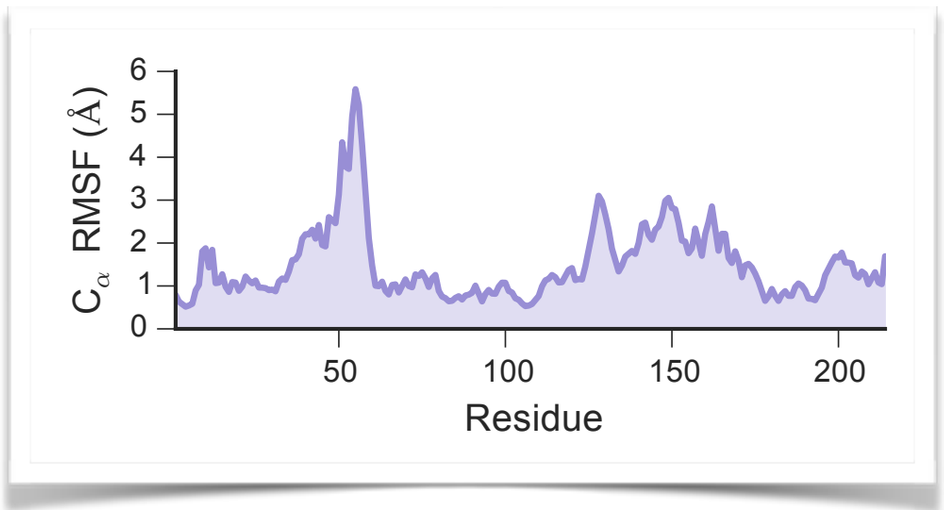


Analysis module: *MDAnalysis.analysis*

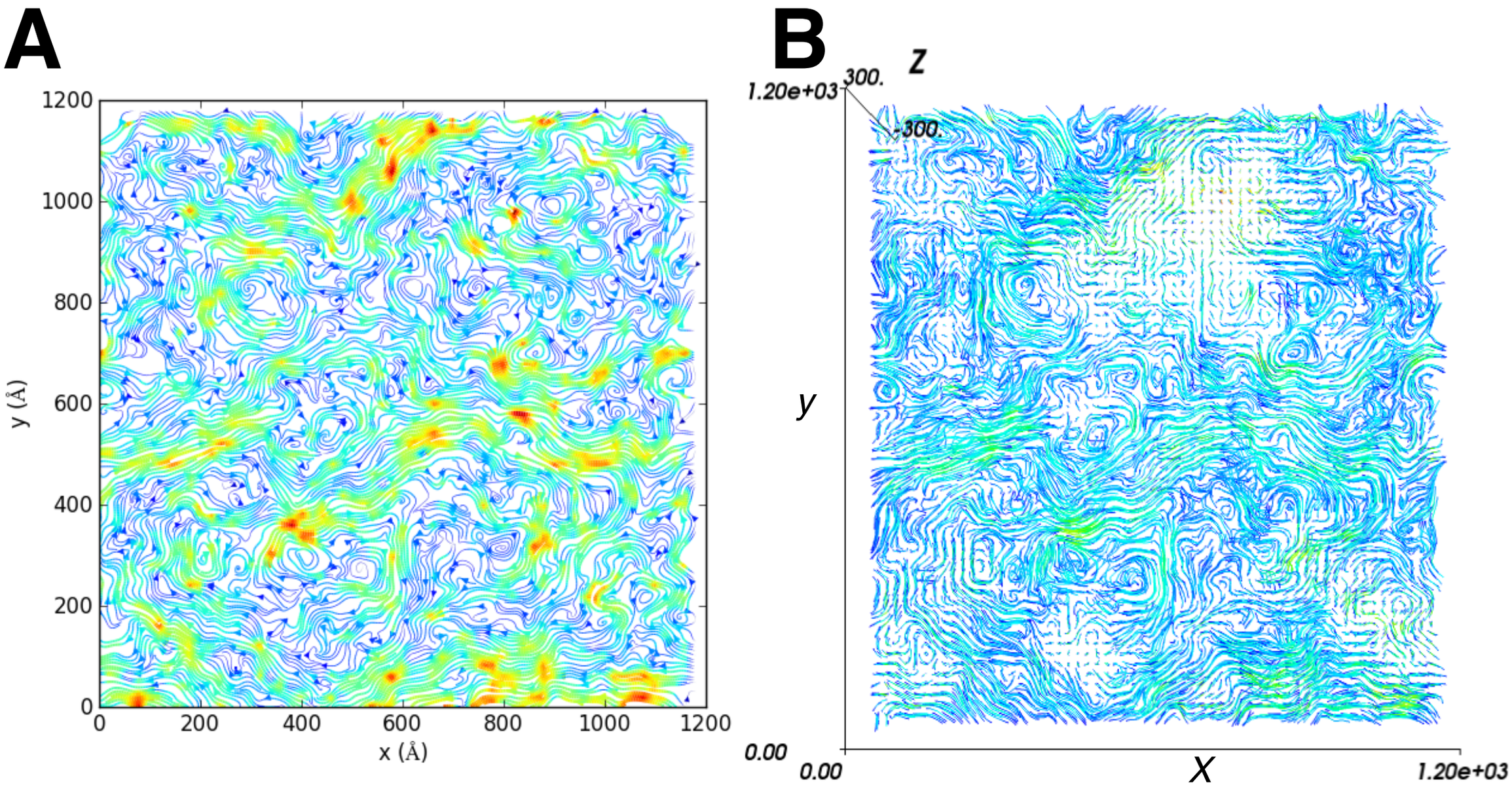
- standard analysis functionality (RMSD, RMSF, distances, density, hydrogen bonds, native contacts…)
- unique capabilities: Path Similarity Analysis, LeafletFinder, water dynamics, HOLE, diffusion map, …
- moving towards a standard API with an *AnalysisBase* class

```
from MDAnalysis.analysis.rms import RMSF
A = RMSF(protein).run()
plt.plot(protein.resids, A.rmsf)
print(A.rmsf)
```

```
array([ 2.55843466,  3.07868589,
        2.37143333, ...,  3.60916556,
        4.05652126,  3.88028668])
```



Visualisation module: *MDAnalysis.visualization*

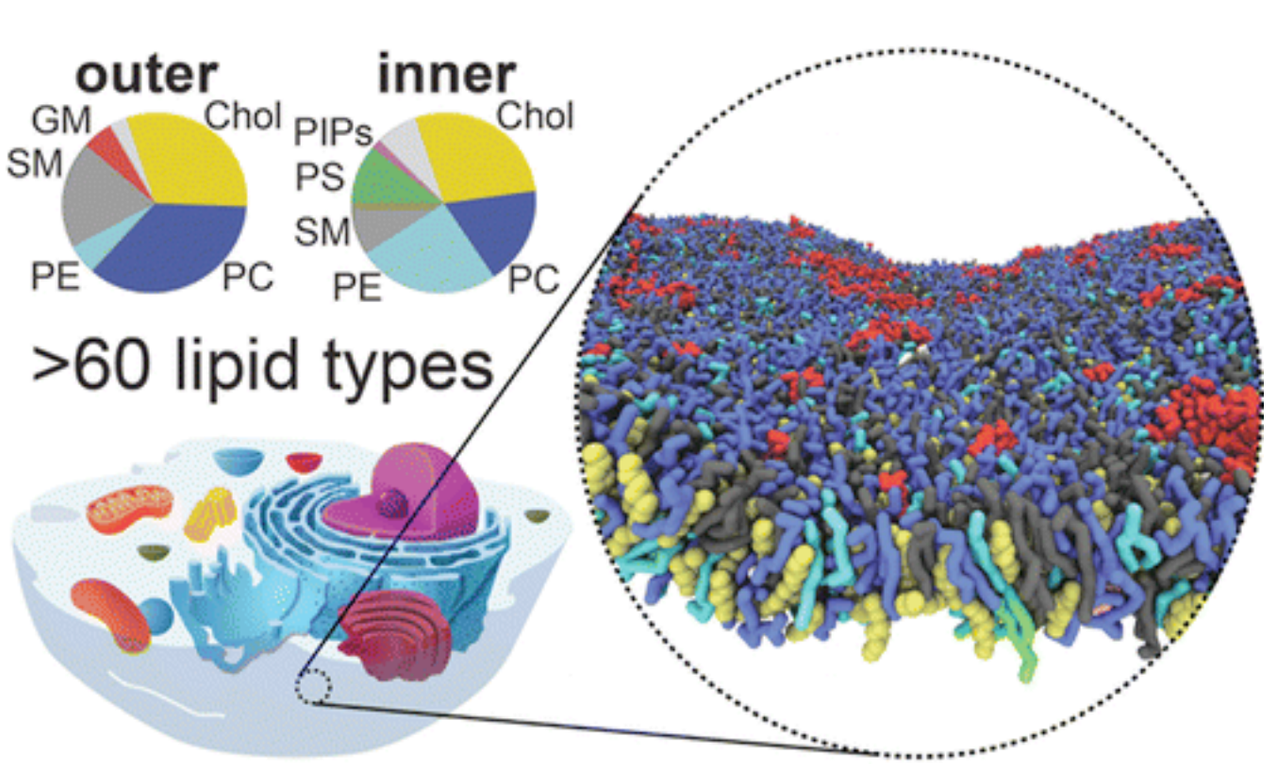


- streamlines: lipid flow

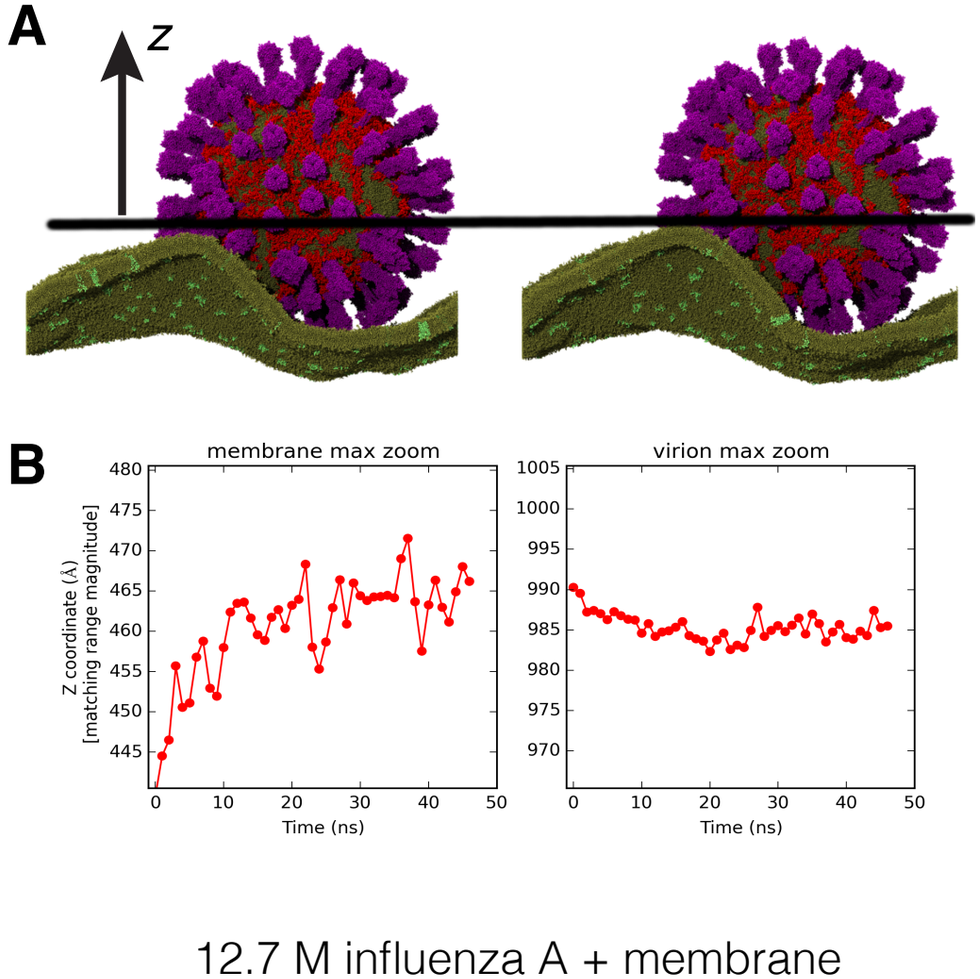
M. Chavent, T. Reddy, J. Goose, A. C. E. Dahl, J. E. Stone, B. Jobard, and M. S. P. Sansom. Methodologies for the analysis of instantaneous lipid diffusion in MD simulations of large membrane systems. *Faraday Discuss.*, 169:455–475, 2014. doi: 10.1039/C3FD00145H.

Very large systems

- MDAnalysis is capable to analyse very large simulation systems (millions of particles, many μs of data)
- E.g., lipid membranes, virions



H. I. Ingólfsson, M. N. Melo, F. J. V. Eerden, C. Arnarez, C. A. López, T. A. Wassenaar, X. Periole, A. H. D. Vries, D. P. Tieleman, and S. J. Marrink. Lipid Organization of the Plasma Membrane Lipid Organization of the Plasma Membrane. *J Am Chem Soc*, 136(41):14554–14559, 2014.



12.7 M influenza A + membrane
(TJE Reddy, unpublished)

Installation

Linux
macOS
Windows

- Conda

```
conda config --add channels conda-forge
conda install mdanalysis
```

- Python Package Index

```
pip install --upgrade MDAnalysis
```

- from source (develop or master == release)

```
git clone https://github.com/MDAnalysis/mdanalysis.git
cd mdanalysis/package
python setup.py install
```

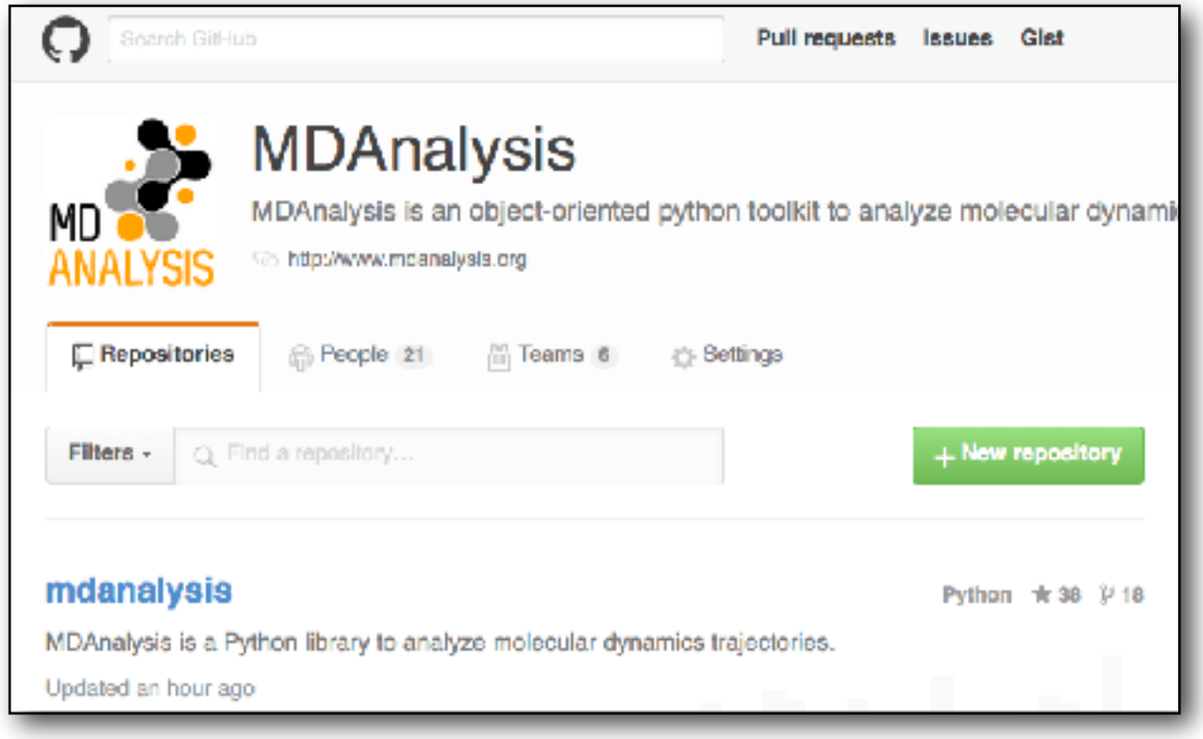
Runs on

- Linux
- macOS
- Windows

Open source

- GPL v2
- github.com/MDAnalysis

GitHub



Development process

- pull request / review / merge
- continuous integration with > 6,500 unit tests

Travis CI

AppVeyor

A history of MDAnalysis

MDAnalysis repository commit history


```
Author: Naveen Michaud-Agrawal
Date:   Thu Jan 31 11:42:33 2008 +0000

initial import
```

Software News and Updates
MDAnalysis: A Toolkit for the Analysis of Molecular Dynamics Simulations

NAVEEN MICHAUD-AGRAWAL,[1] ELIZABETH J. DENNING,[1,2] THOMAS B. WOOLF,[1,3] OLIVER BECKSTEIN[3,4]

Received 23 October 2010; Revised 6 February 2011; Accepted 12 February 2011
DOI 10.1002/jcc.21787
Published online 15 April 2011 in Wiley Online Library (wileyonlinelibrary.com).

J Comput Chem 32: 2319–2327, 2011

MDAnalysis: A Python Package for the Rapid Analysis of Molecular Dynamics Simulations

Richard J. Gowers[‖**†], Max Linke[‡‡†], Jonathan Barnoud[§†], Tyler J. E. Reddy[‡], Manuel N. Melo[§], Sean L. Seyler[¶], Jan Domański[‡], David L. Dotson[¶], Sébastien Buchoux[††], Ian M. Kenney[¶], Oliver Beckstein[¶*]

In S. Benthall and S. Rostrup, editors, Proceedings of the 15th Python in Science Conference, pages 98–105, Austin, TX, 2016. SciPy

(>530 citations on GoogleScholar (Nov 2018))

MDA 

USERS

(… and developers, of course!)

MDAnalysis is not just code, it's also a community

MDA ❤ USERS

Open and inclusive:

- questions are answered (mailing list)
- extensive **docs**
- **pull requests** very welcome!

Reduce upgrade pain for users:

- semantic versioning
- unavoidable API breaks
  - deprecation cycles
  - **fixer** script based on standard **lib2to3**

users ⟶ contributors ⟶ developers

- 83 contributing authors (Nov 2018)
- ~9 core developers

MD ANALYSIS

- forms a bridge between most of the commonly encountered MD trajectory formats and NumPy arrays
- provides an object-oriented interface to data on particles
- comes with many analysis (*MDAnalysis.analysis*) and visualisation (*MDAnalysis.vizualiation*) classes/functions
- has a welcoming and active community

MDA ❤ USERS

Naveen Michaud-Agrawal, Elizabeth J. Denning, Christian Beckstein (logo), Joshua L. Adelman, Shobhit Agarwal, Irfan Alibay, Balasubramanian, Utkarsh Bansal, Jonathan Barnoud, Tone Bengtsen, Alejandro Bernardin, Mateusz Bieniek, Wouter Boomsma, Jose Borreguero, Bart Bruininks, Sébastien Buchoux, Sören von Bülow, David Caplan, Matthieu Chavent, Kathleen Clark, Ruggero Cortini, Davide Cruz, Robert Delgado, John Detlefs, Xavier Deupi, Jan Domanski, **David L. Dotson**, **Shujie Fan**, Lennard van der Feltz, Philip Fowler, Joseph Goose, **Richard J. Gowers**, Lukas Grossar, Abhinav Gupta, Akshay Gupta, Benjamin Hall, Eugen Hruska, Kyle J. Huston, Joe Jordan, Jon Kapla, Navya Khare, Andrew William King, Max Linke, Philip Loche, Jinju Lu, **Micaela Matta**, Andrew R. McCluskey, Robert McGibbon, Manuel Nuno Melo, Dominik 'Rathann' Mierzejewski, Henry Mull, Fiona B. Naughton, Alex Nesterenko, Hai Nguyen, Sang Young Noh, Nabarun Pal, Mattia F. Palermo, Danny Parton, Joshua L. Phillips, Kashish Punjani, Vedant Rathore, Tyler Reddy, Pedro Reis, Paul Rigor, Carlos Yanez S., Utkarsh Saxena, Sean L. Seyler, Paul Smith, Andy Somogyi, Caio S. Souza, Shantanu Srivastava, Lukas Stelzl, Gorman Stock, Ayush Suhane, Xiki Tempula, Matteo Tiberti, Isaac Virshup, Nestor Wendt, Zhiyi Wu, Zhuyi Xue, Juan Eiros Zamora, Johannes Zeman, and **Oliver Beckstein**.

9 core developers
5 GSoC students
4 REU students
83 contributors

Join us at
mdanalysis.org
github.com/MDAnalysis

NUMFOCUS
OPEN CODE = BETTER SCIENCE

ASU

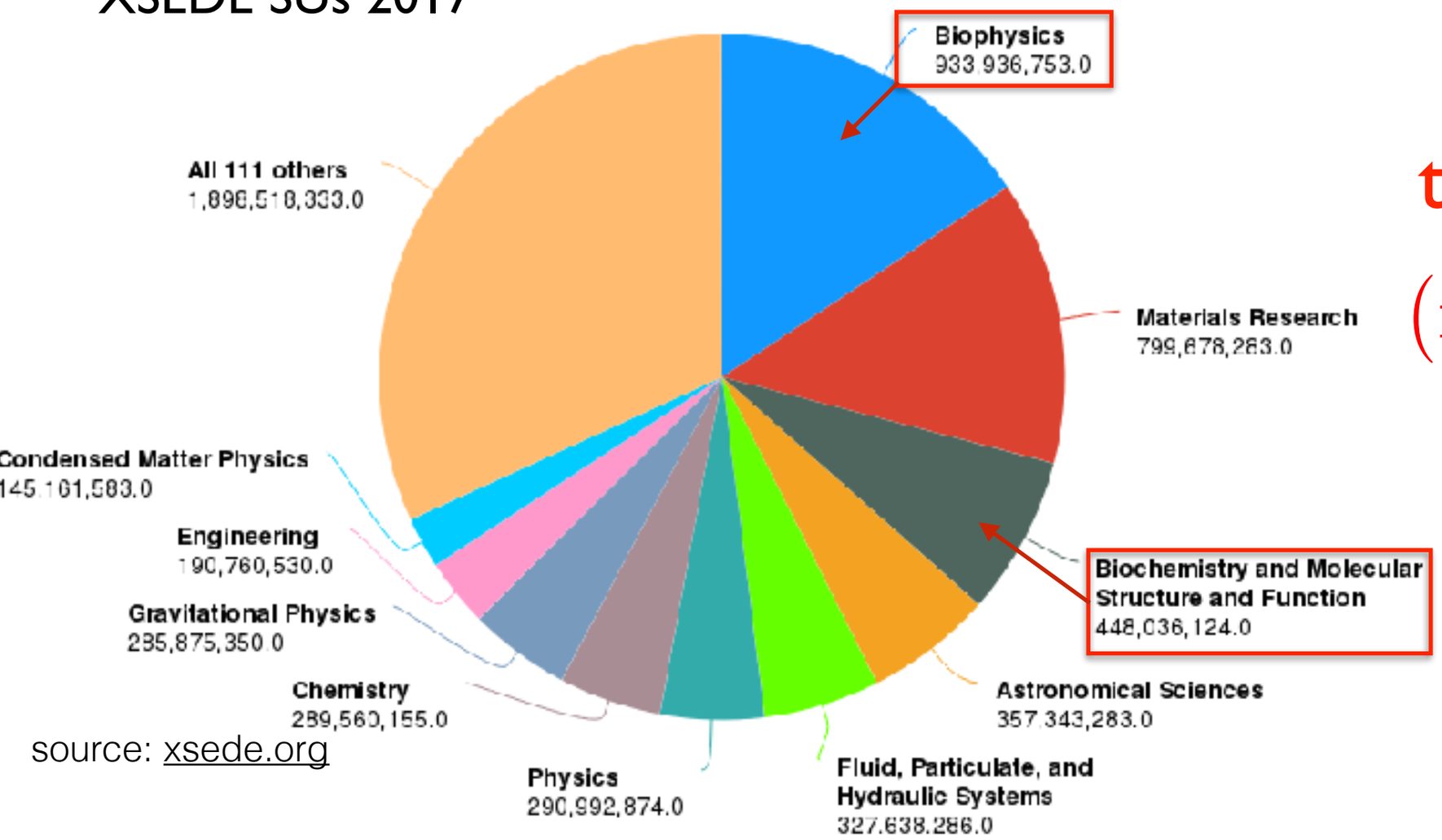
NSF
ACI-1443054

python SOFTWARE FOUNDATION

Google
GSoC 2016–2018

Appendix

XSEDE SUs 2017



source: xsede.org

trajectory

$$\left(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t)\right)$$

$$0 \le t \le \tau$$

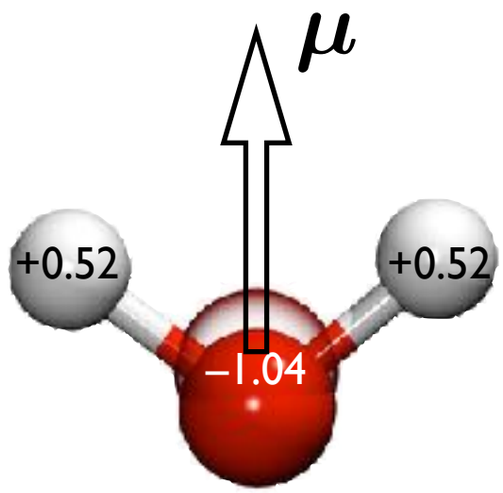
	max (2018)	typical (2018)
atoms *N*	~10^8	~10^5
simulated time τ	~1 ms	1 μs
trajectory frames	~10^{12}	~10^6
trajectory size	< 100 TiB	0.5 TiB

see also: T. Cheatham and D. Roe. Computing in Science Engineering, 17:30–39, 2015.

Molecular Dynamics (MD) Simulations

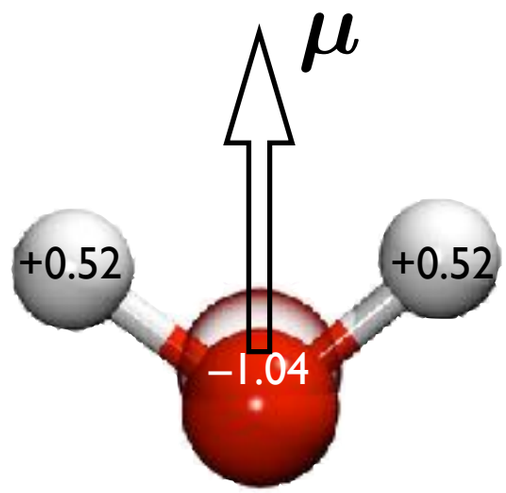
Molecular Dynamics (MD) Simulations

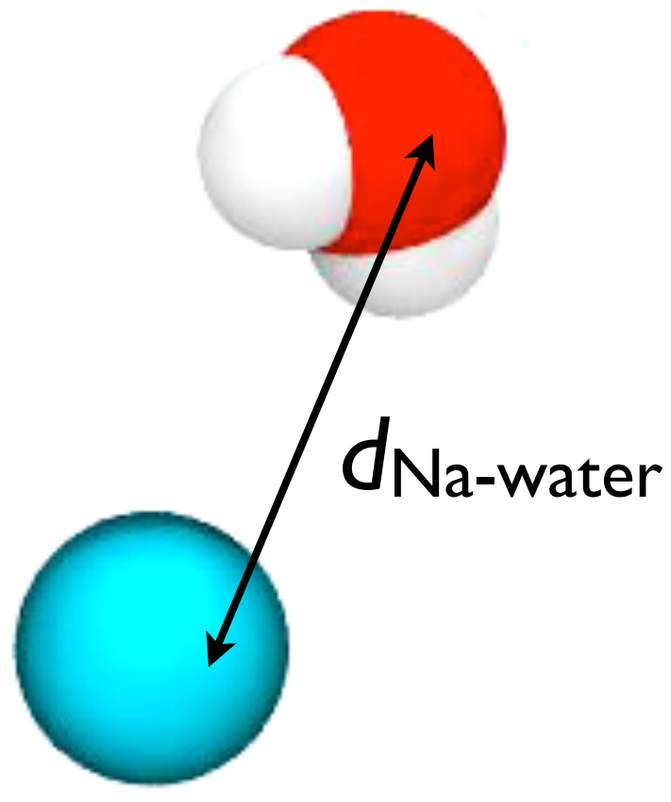
$$\boldsymbol{\mu} = q_H(\mathbf{r}_{\mathrm{HW}_1} + \mathbf{r}_{\mathrm{HW}_2} - 2\mathbf{r}_{\mathrm{MW}})$$



$$\boldsymbol{\mu} = q_H(\mathbf{r}_{\mathrm{HW}_1} + \mathbf{r}_{\mathrm{HW}_2} - 2\mathbf{r}_{\mathrm{MW}})$$

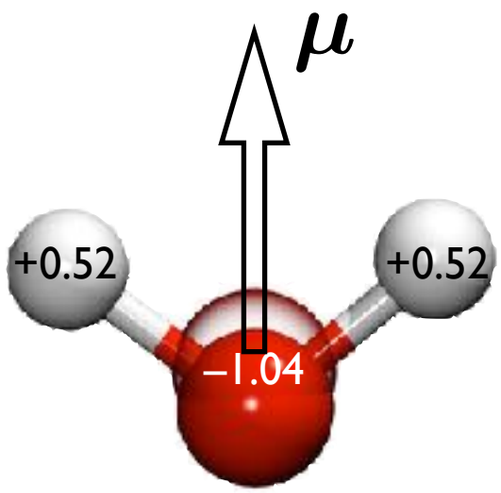
$$d_{\text{Na-water}} = |\mathbf{r}_{\mathrm{Na}} - \mathbf{r}_{\text{water}}|$$

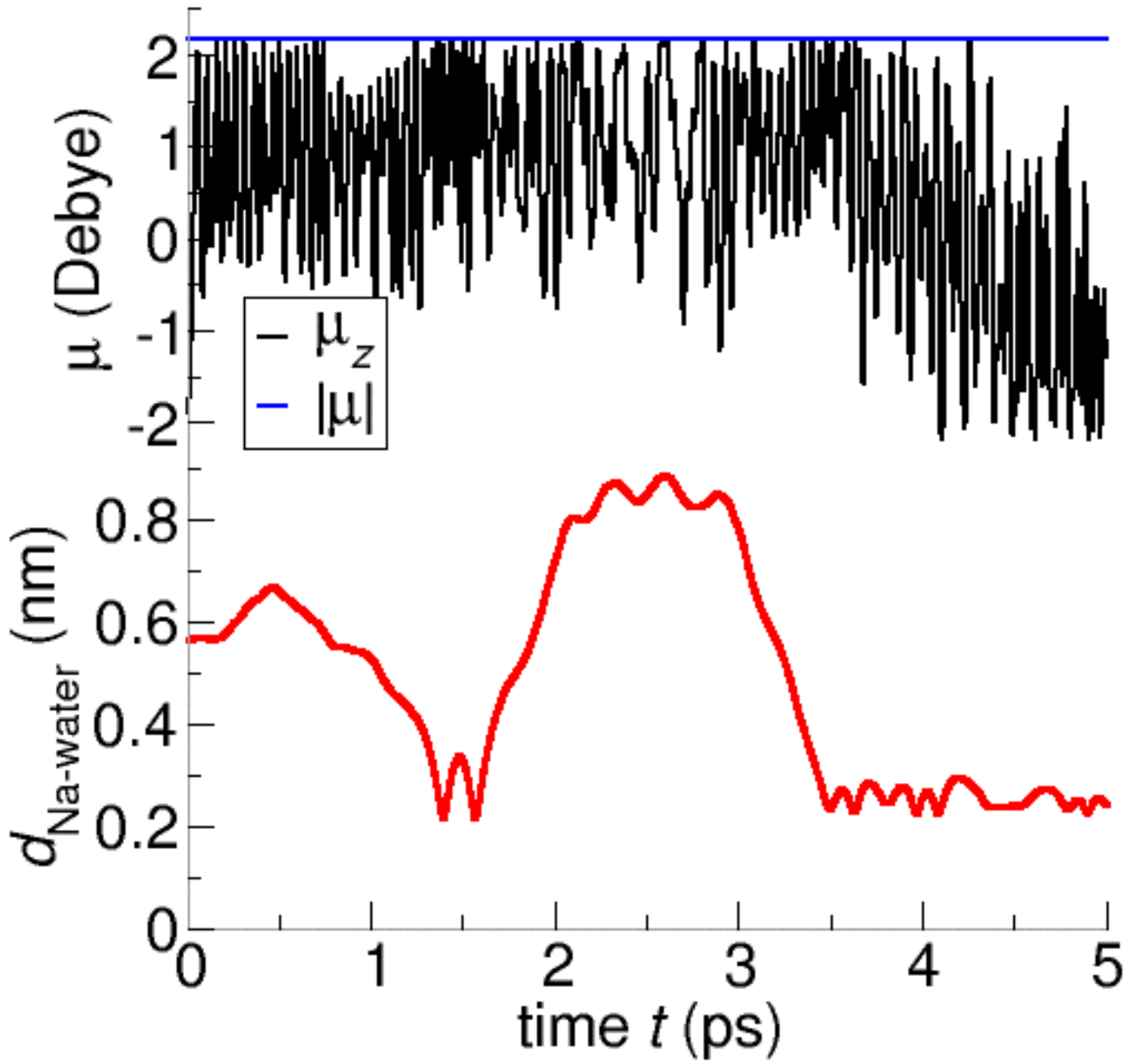




$$\boldsymbol{\mu} = q_H(\mathbf{r}_{\mathrm{HW}_1} + \mathbf{r}_{\mathrm{HW}_2} - 2\mathbf{r}_{\mathrm{MW}})$$

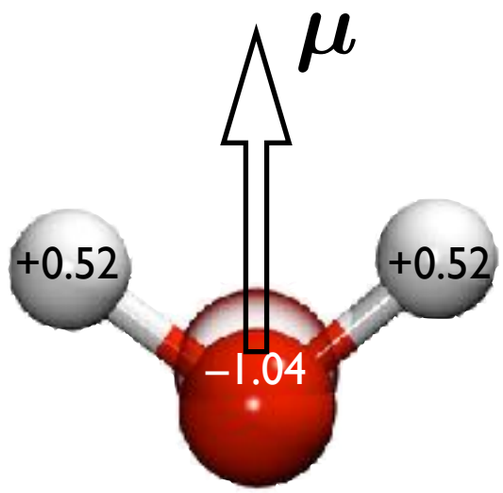
$$d_{\text{Na-water}} = |\mathbf{r}_{\mathrm{Na}} - \mathbf{r}_{\text{water}}|$$

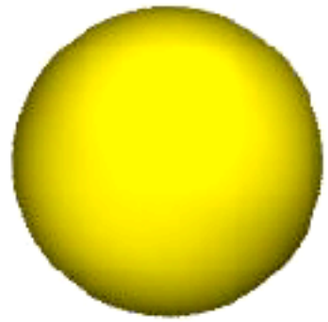




$$\boldsymbol{\mu} = q_H(\mathbf{r}_{\mathrm{HW}_1} + \mathbf{r}_{\mathrm{HW}_2} - 2\mathbf{r}_{\mathrm{MW}})$$

$$d_{\text{Na-water}} = |\mathbf{r}_{\mathrm{Na}} - \mathbf{r}_{\text{water}}|$$





$$\boldsymbol{\mu} = q_H(\mathbf{r}_{\mathrm{HW}_1} + \mathbf{r}_{\mathrm{HW}_2} - 2\mathbf{r}_{\mathrm{MW}})$$

$$d_{\text{Na-water}} = |\mathbf{r}_{\mathrm{Na}} - \mathbf{r}_{\text{water}}|$$