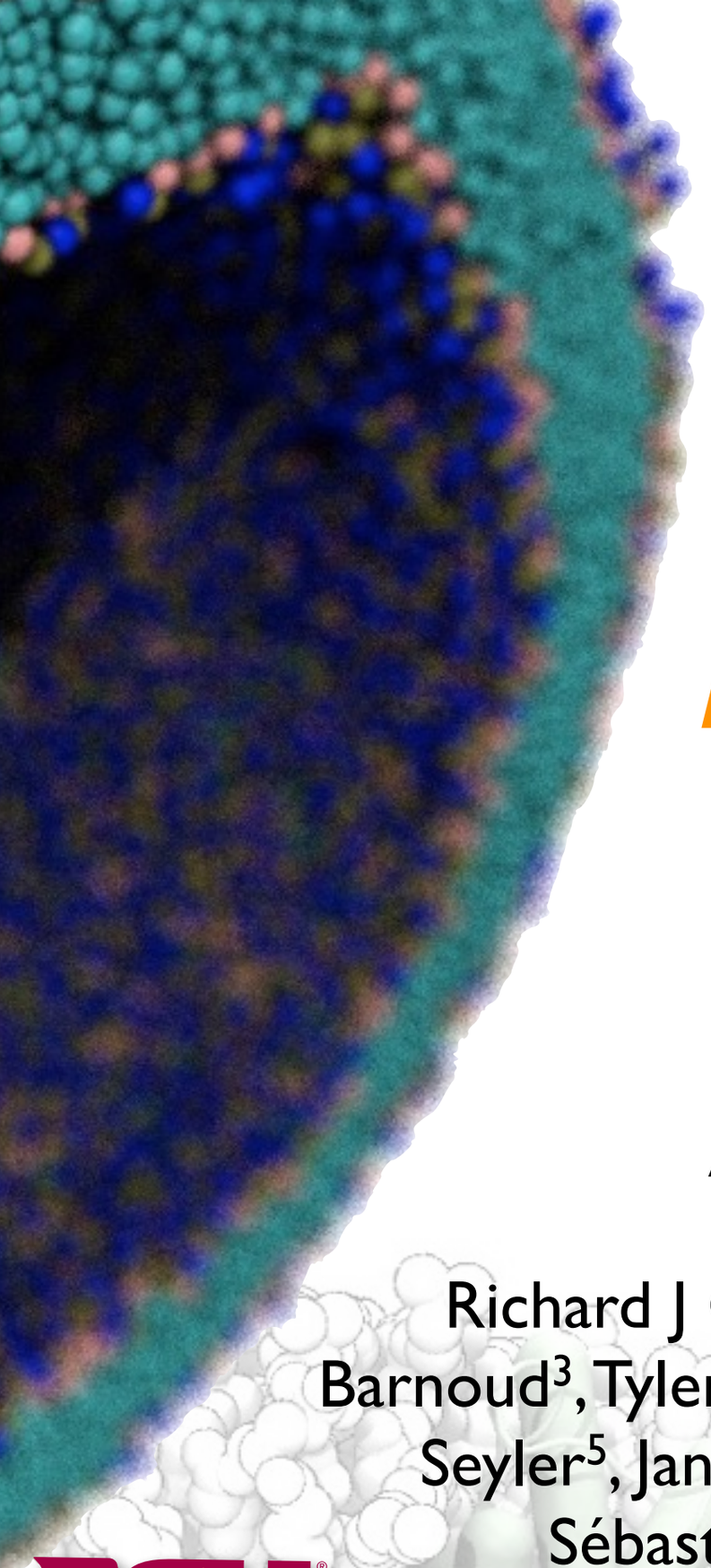


MD ANALYSIS

A Python package for the rapid analysis of molecular dynamics simulations

SciPy 2016, Austin, TX
July 13, 2016

Oliver Beckstein
Arizona State University

Richard J Gowers[1], Max Linke[2], Jonathan Barnoud[3], Tyler J. E. Reddy[4], Manuel Melo[3], Sean. S. Seyler[5], Jan Domanski[4,6], David L. Dotson[5], Sébastien Buchoux[7], Ian Kenney[5]

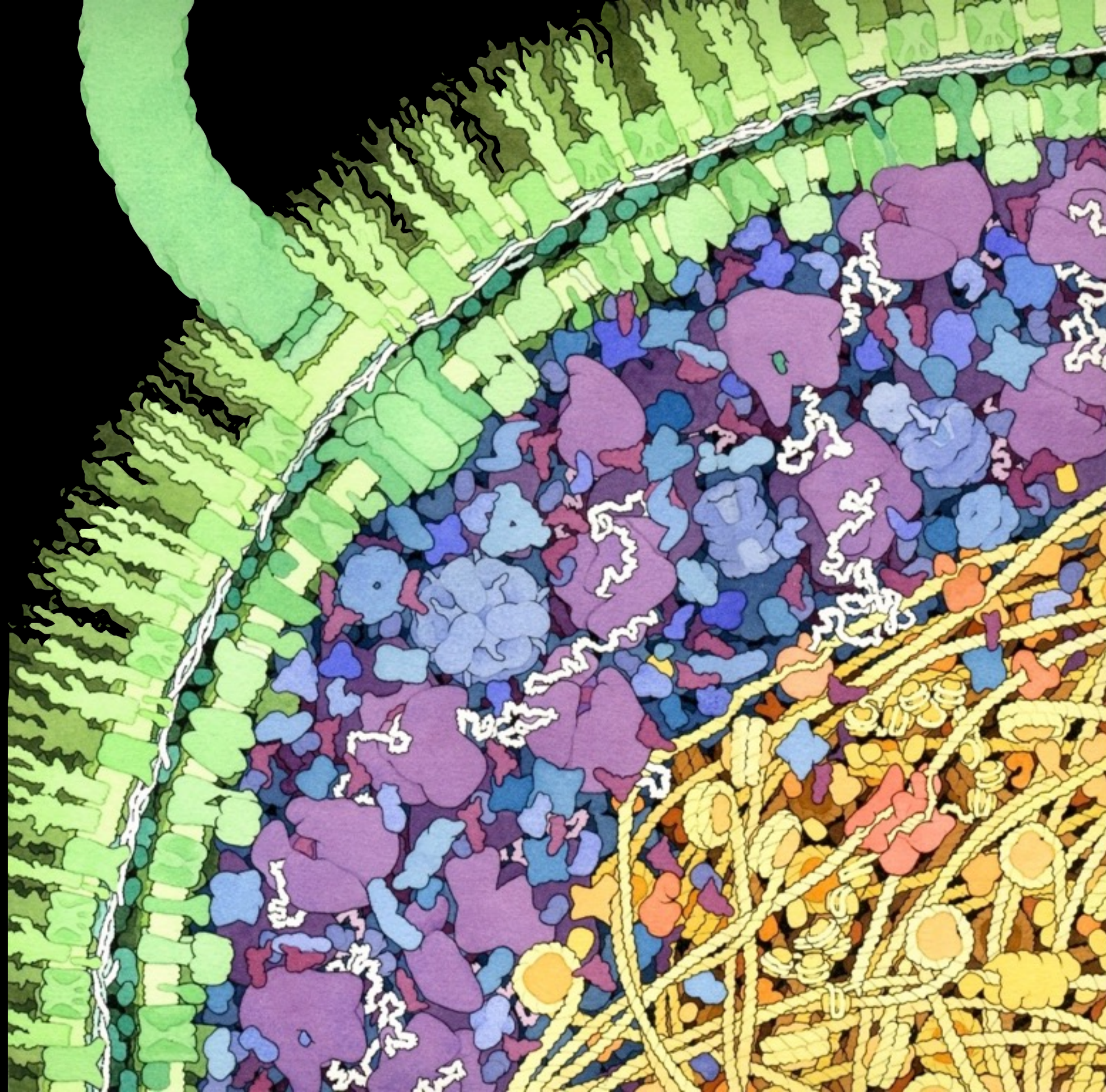
[1]University of Edinburgh, UK, [2]Max Planck Institut für Biophysik, Frankfurt, Germany, [3]University of Groningen, The Netherlands, [4]University of Oxford, UK, [5]Arizona State University, [6]NIH NIDDK, Bethesda, MD, [7]Université de Picardie Jules Verne, France

LIFE

cells

molecules

atoms



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

Biomolecules

nucleic acids
(DNA, RNA)

proteins

lipids

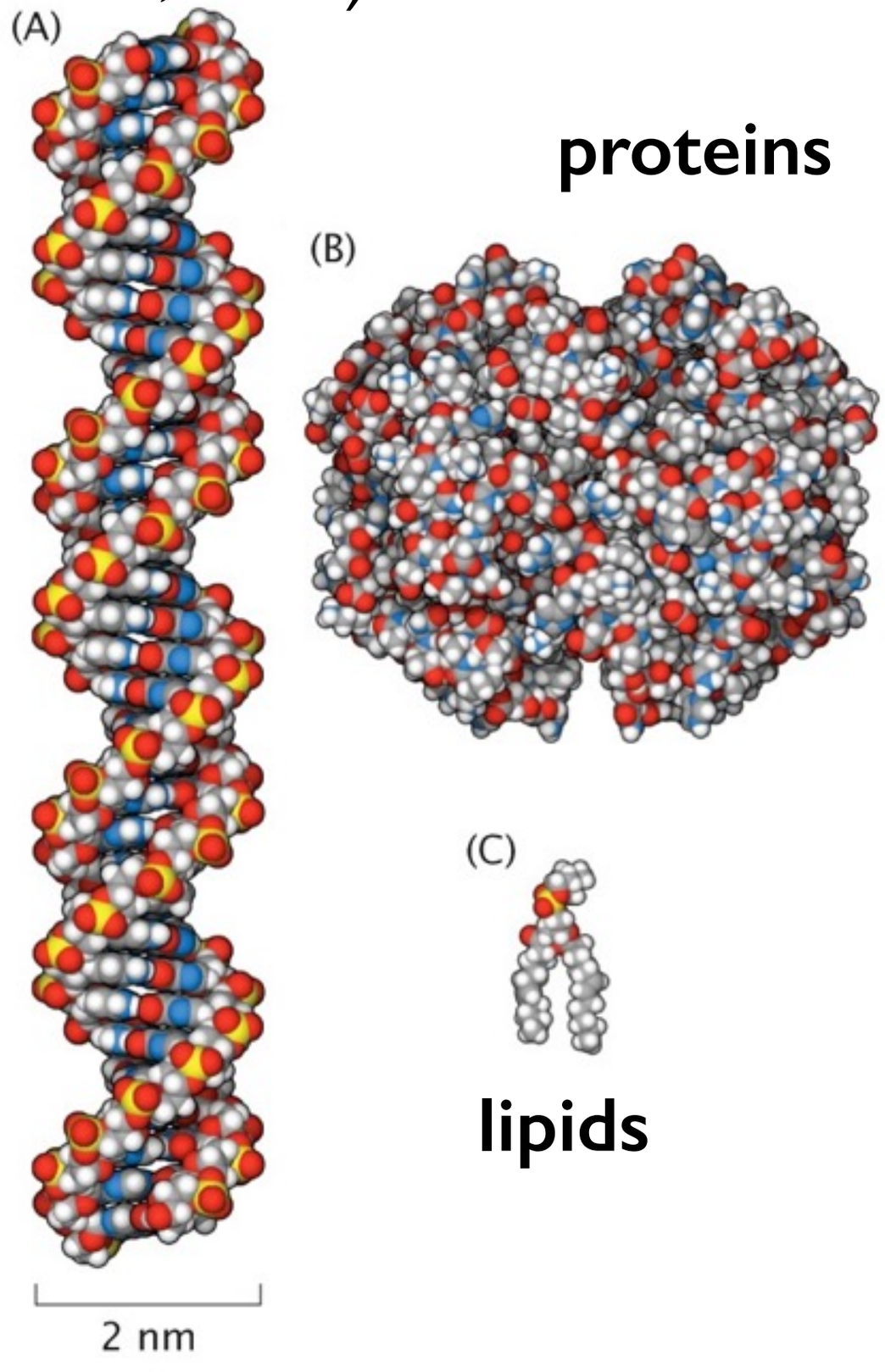


Figure 1.1 Physical Biology of the Cell, 2ed. (© Garland Science 2013)

Biomolecules

nucleic acids
(DNA, RNA)

proteins

lipids

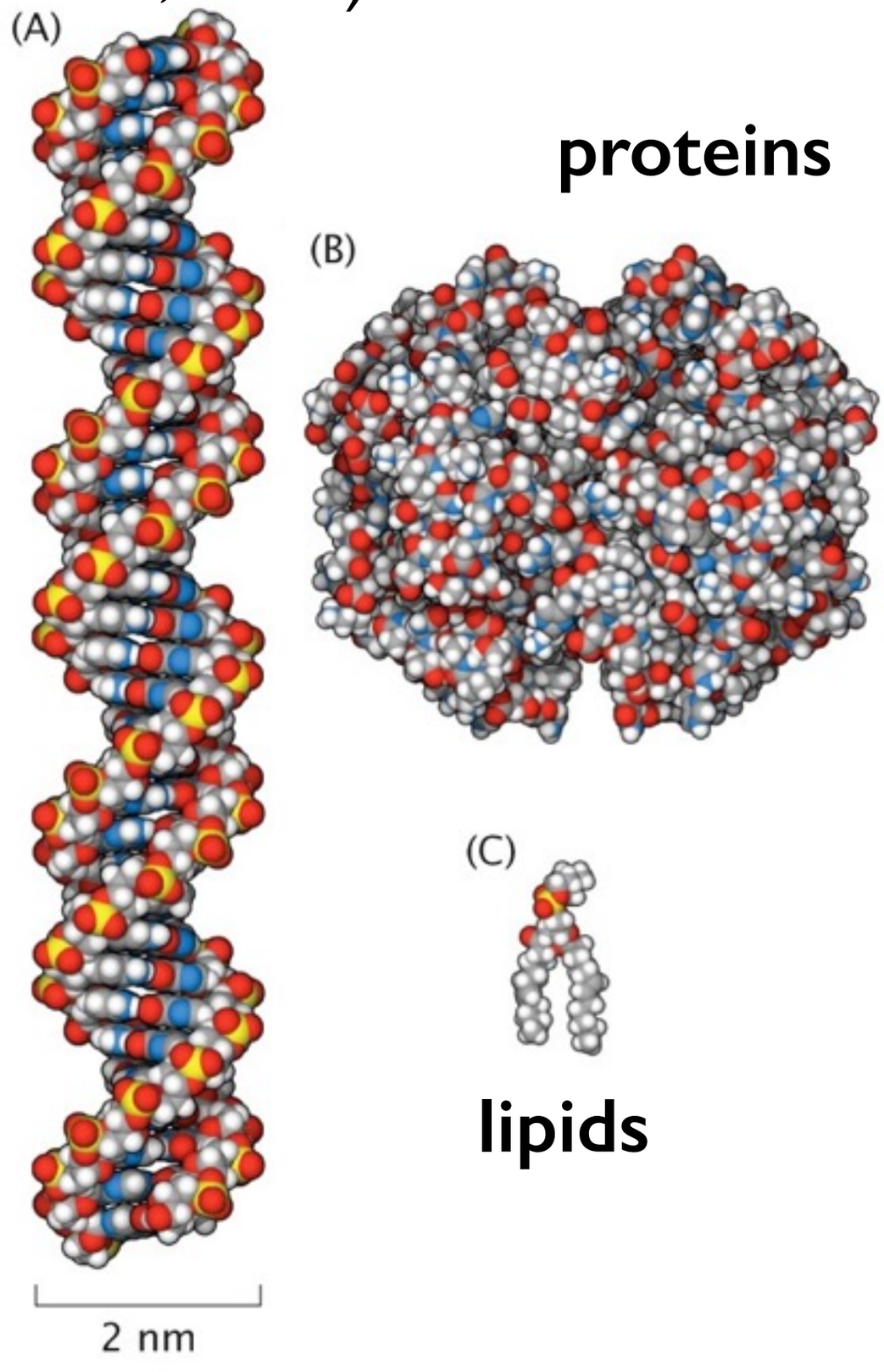


Figure 1.1 Physical Biology of the Cell, 2ed. (© Garland Science 2013)

protein
representations

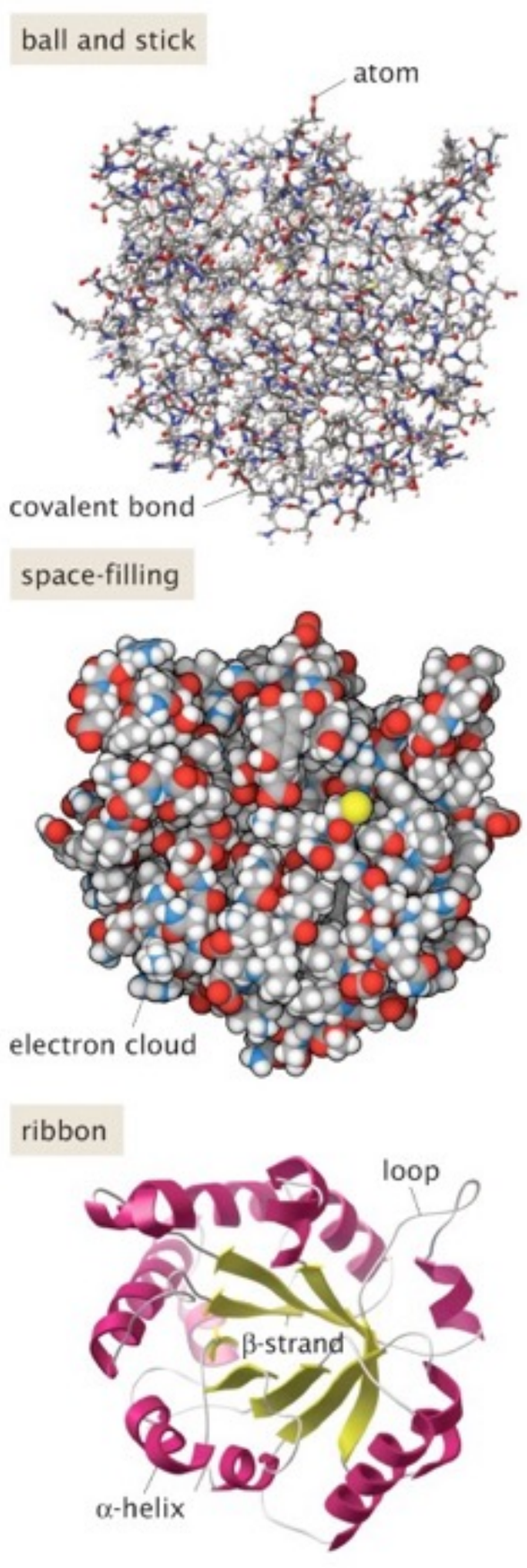
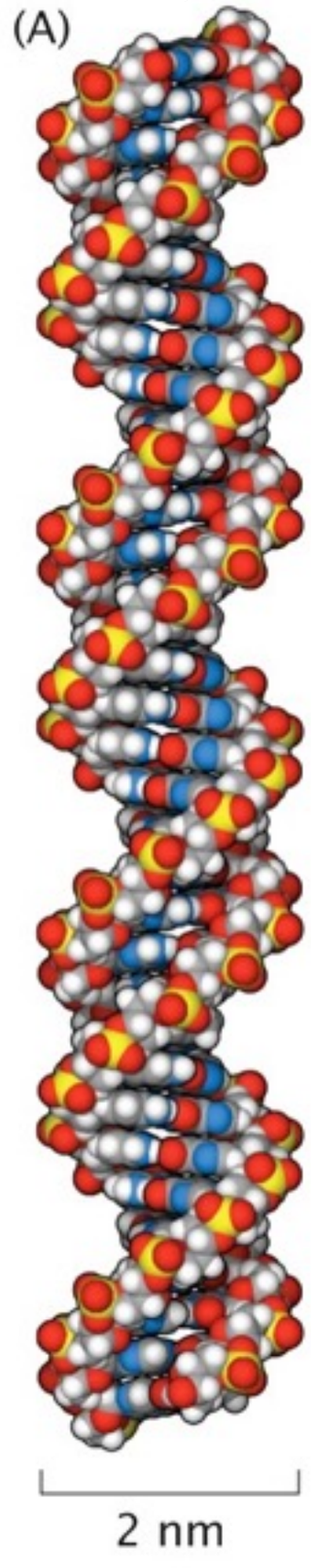


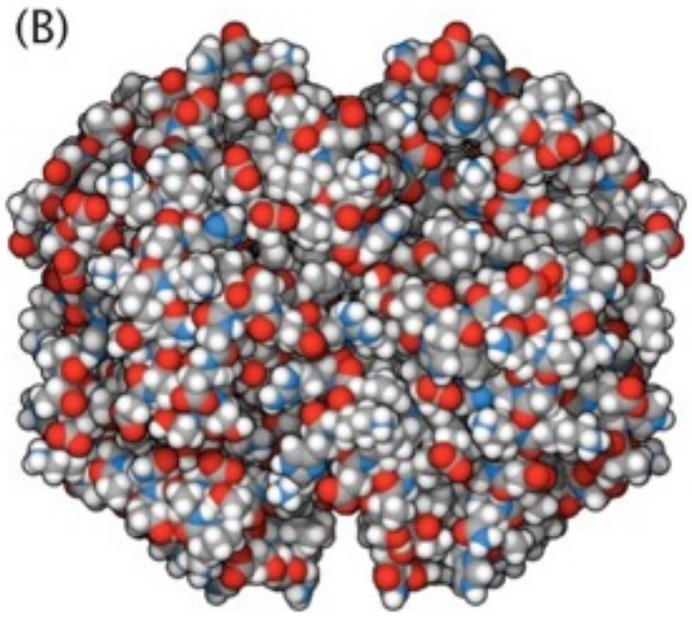
Figure 2.32 Physical Biology of the Cell, 2ed.

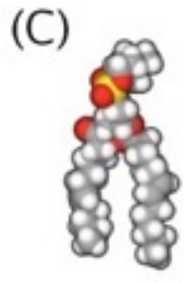
Biomolecules

nucleic acids
(DNA, RNA)



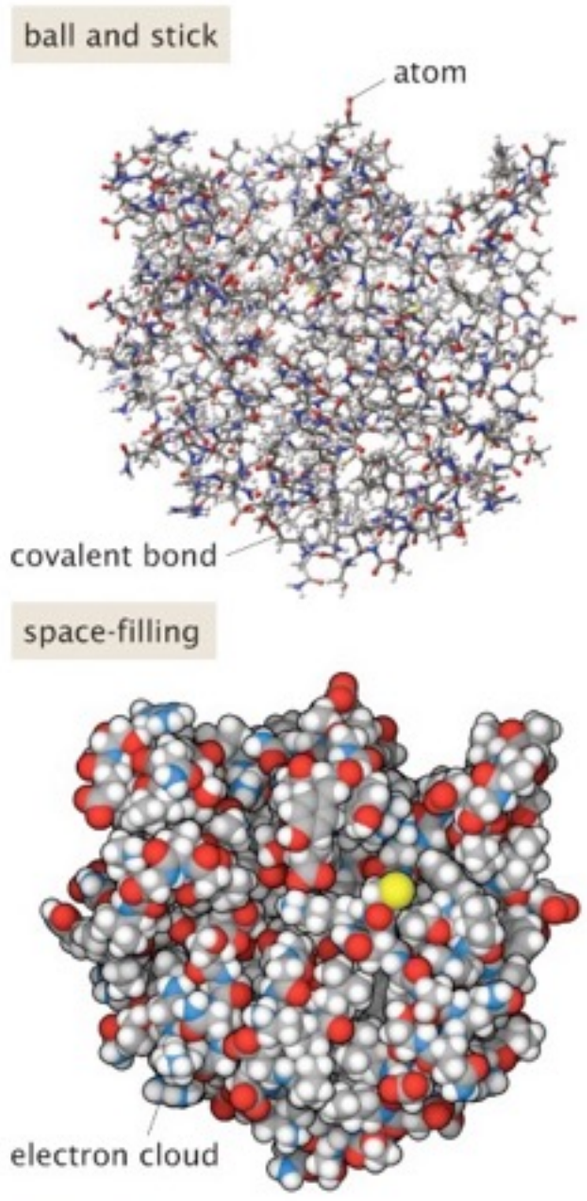
proteins





lipids

water



protein representations

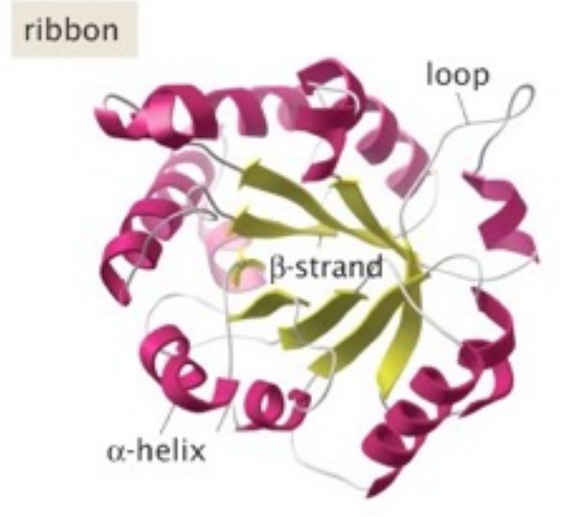
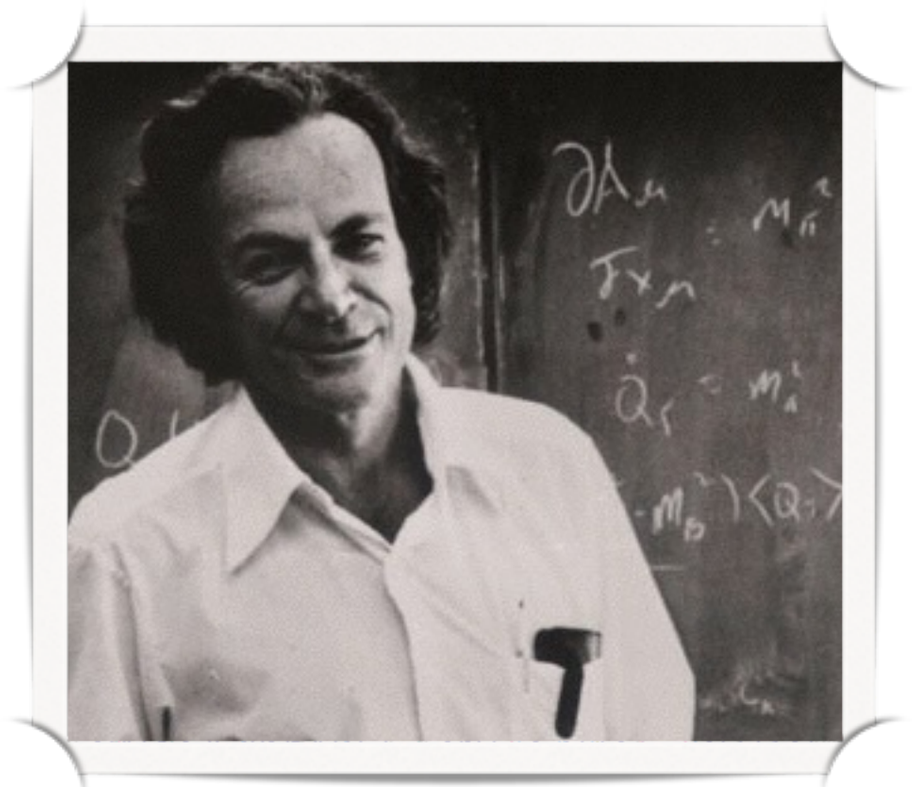


Figure 1.1 Physical Biology of the Cell, 2ed. (© Garland Science 2013)

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*



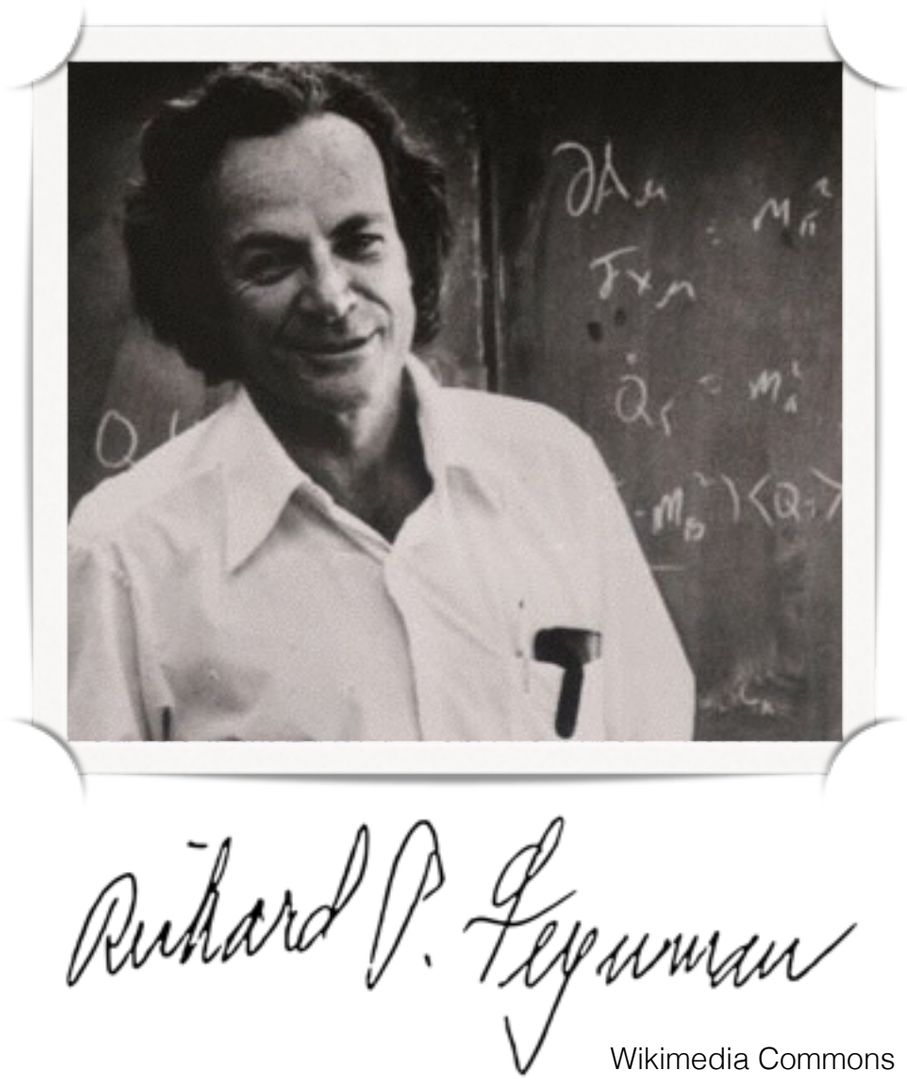
Wikimedia Commons

* 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*

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

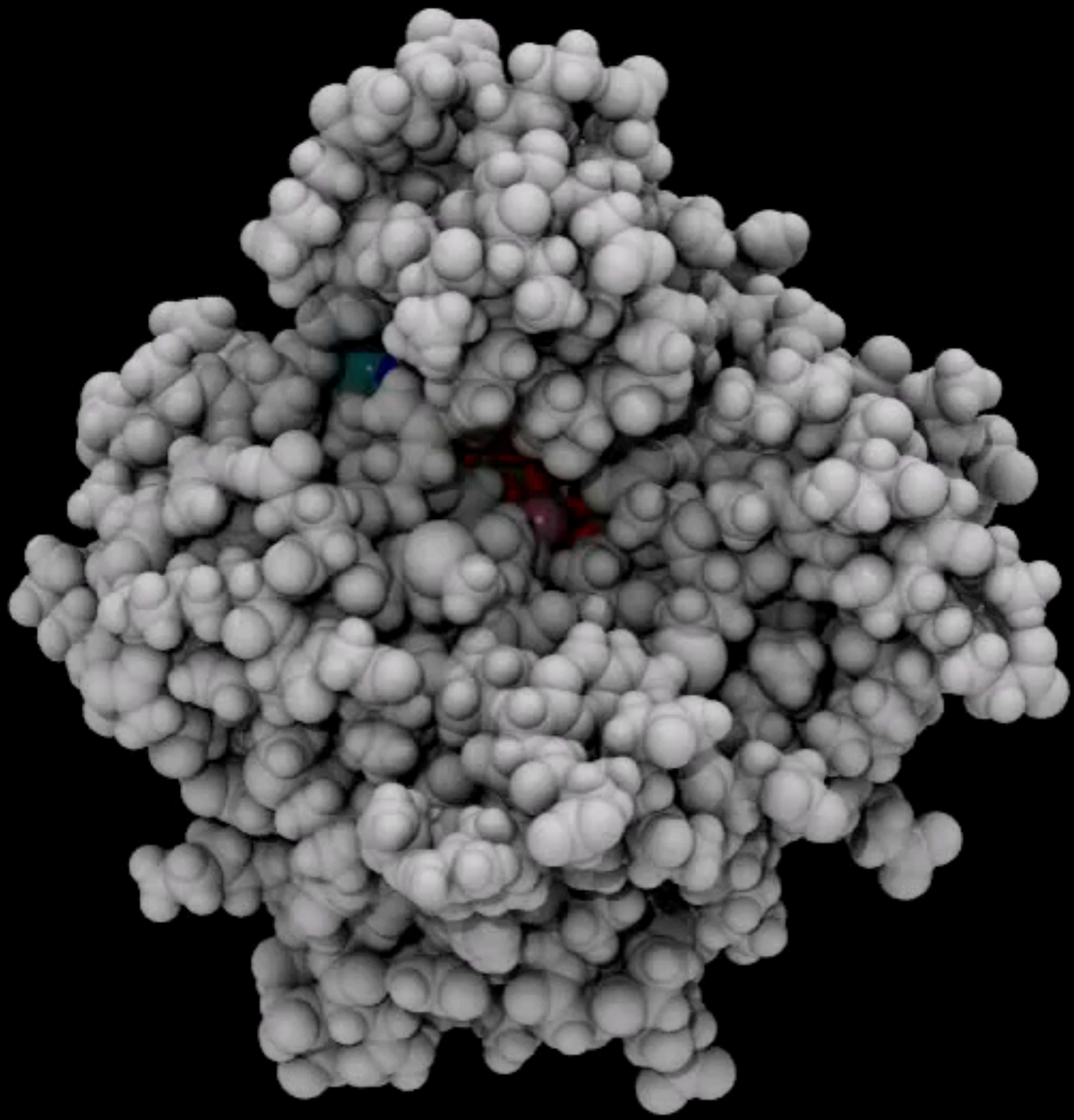
positions of all *N* atoms over time



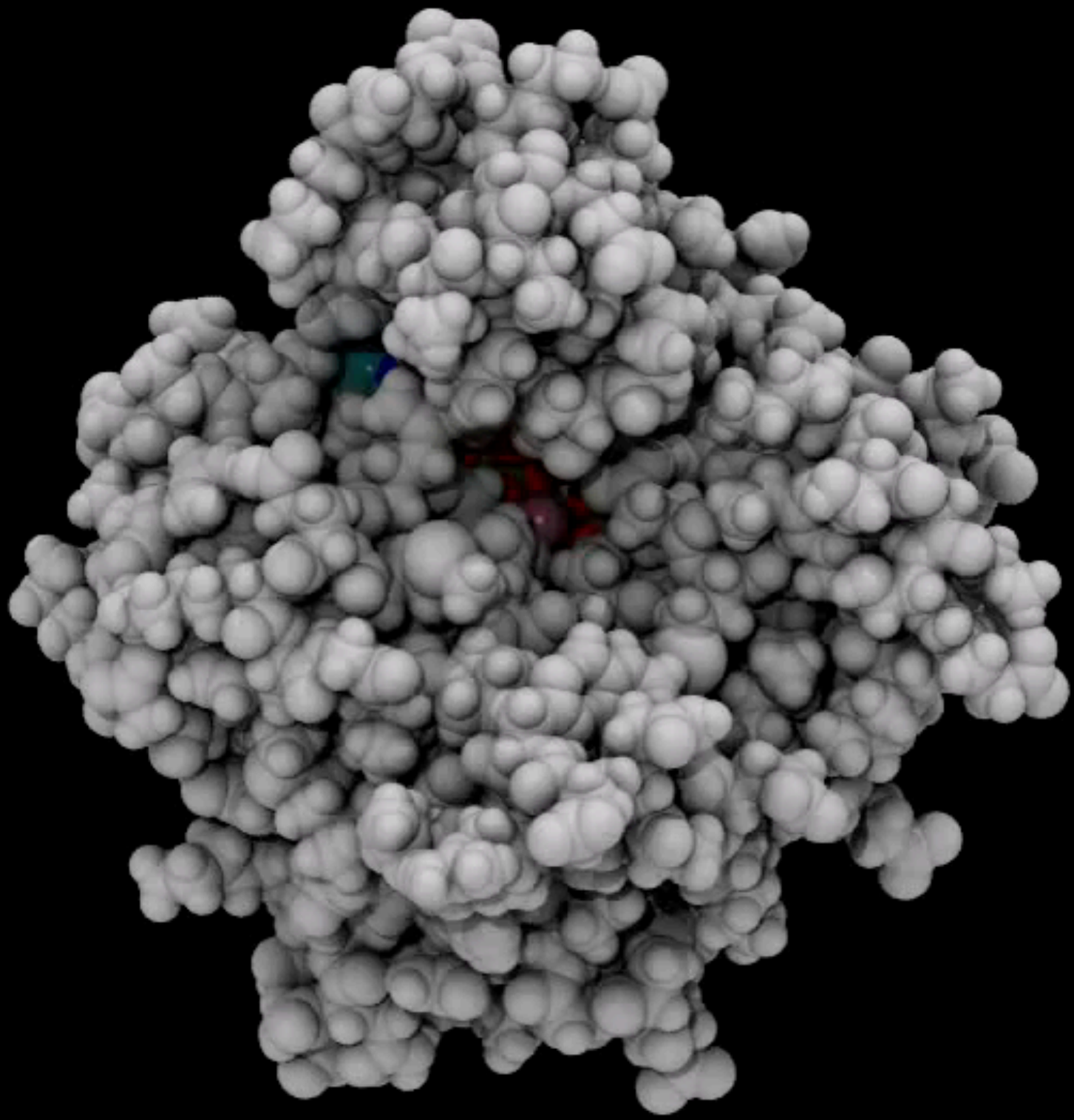
Wikimedia Commons

* 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"

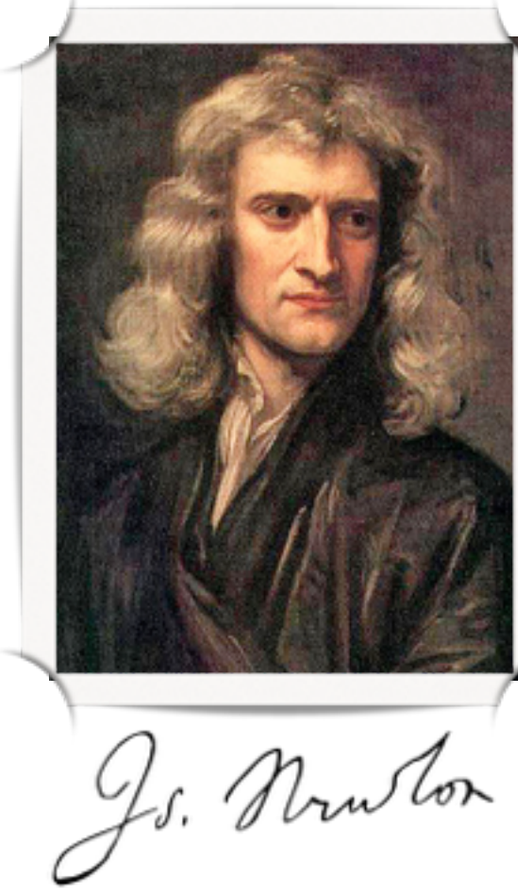
$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

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"

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



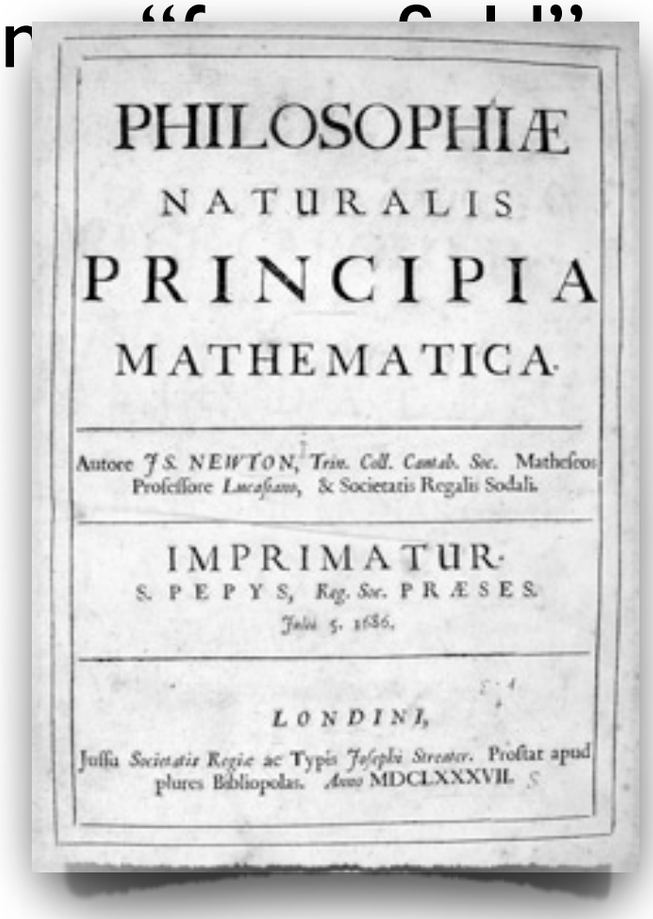
source: Wikimedia Commons

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

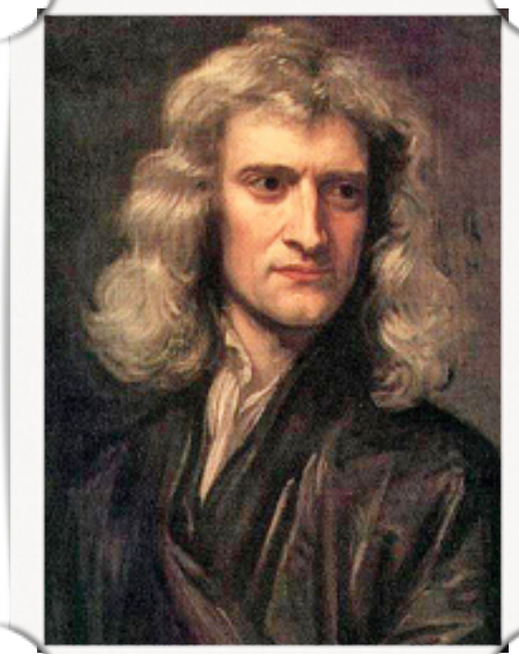
$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$


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

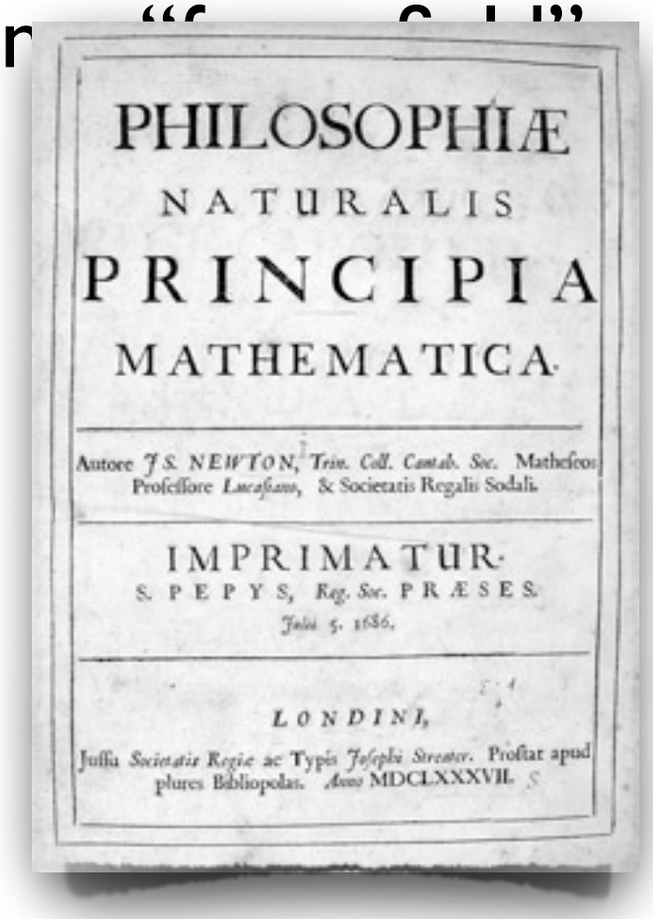
Molecular Dynamics (MD) Simulations (classical)

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

← energy functio[illegible]

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

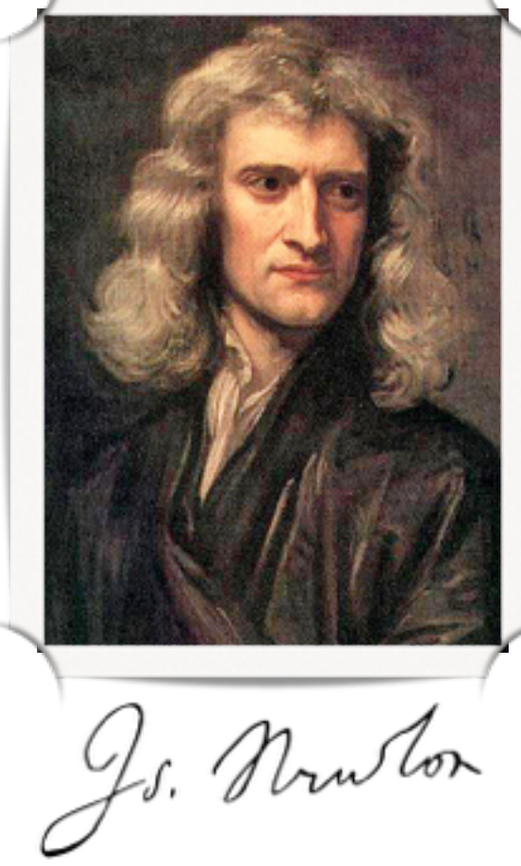
$$\mathbf{F}_i = m_i \mathbf{a}_i$$ ← Newton's 2nd law


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)

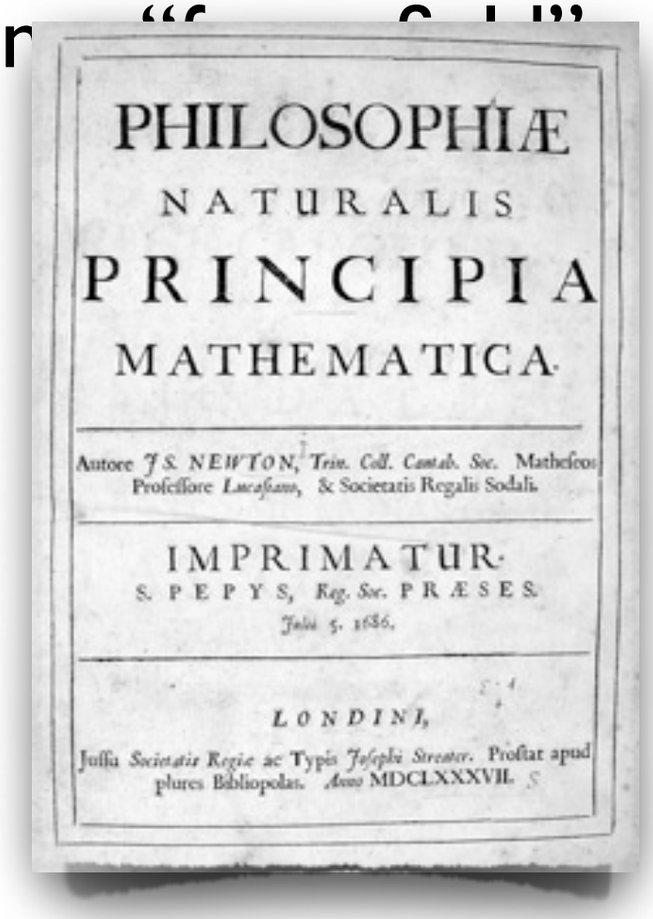
$$U(\mathbf{r}_1, \ldots, \mathbf{r}_N) = U_{\text{bonded}}(\mathbf{r}_1, \ldots) + U_{\text{non-bonded}}(\mathbf{r}_1, \ldots)$$

← 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

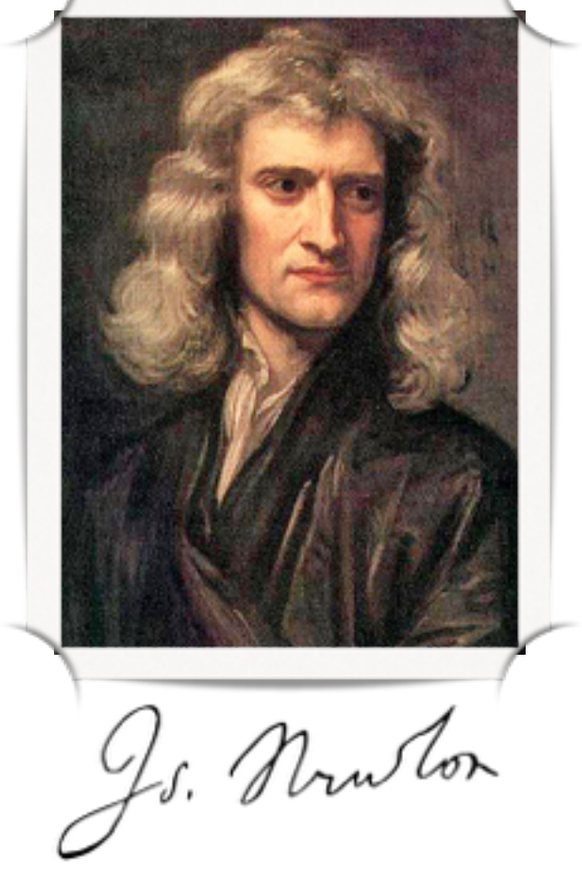
$$\frac{d^2 \mathbf{r}_i}{dt^2} = \frac{\mathbf{F}_i}{m_i}$$


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 functio[illegible]

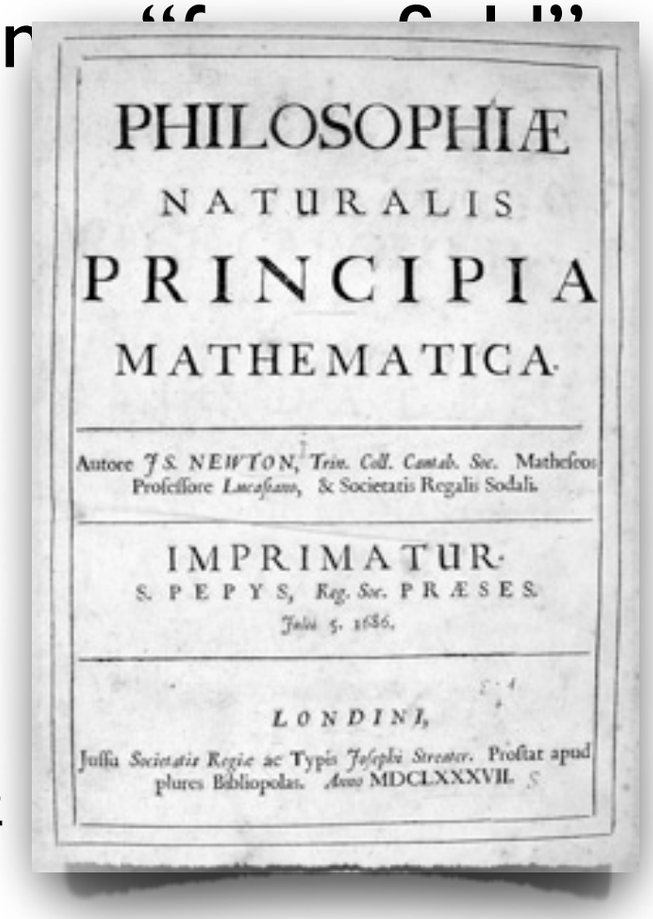
$$\mathbf{F}_i = -\frac{\partial}{\partial \mathbf{r}_i} U(\mathbf{r}_1, \ldots, \mathbf{r}_i, \ldots, \mathbf{r}_N)$$

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

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

integrator →

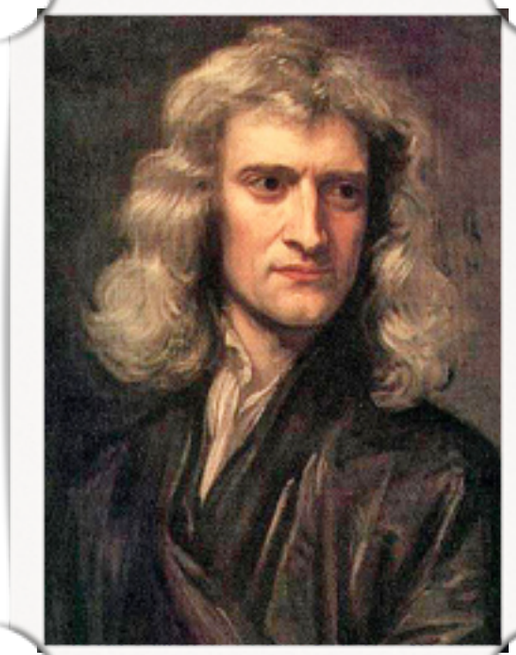
$$\mathbf{r}_i(t + \Delta t) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t - \Delta t) + \frac{\mathbf{F}_i}{m_i}\Delta t^2$$


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



source: Wikimedia Commons

Molecular Dynamics (MD) Simulations (classical)

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

energy func

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

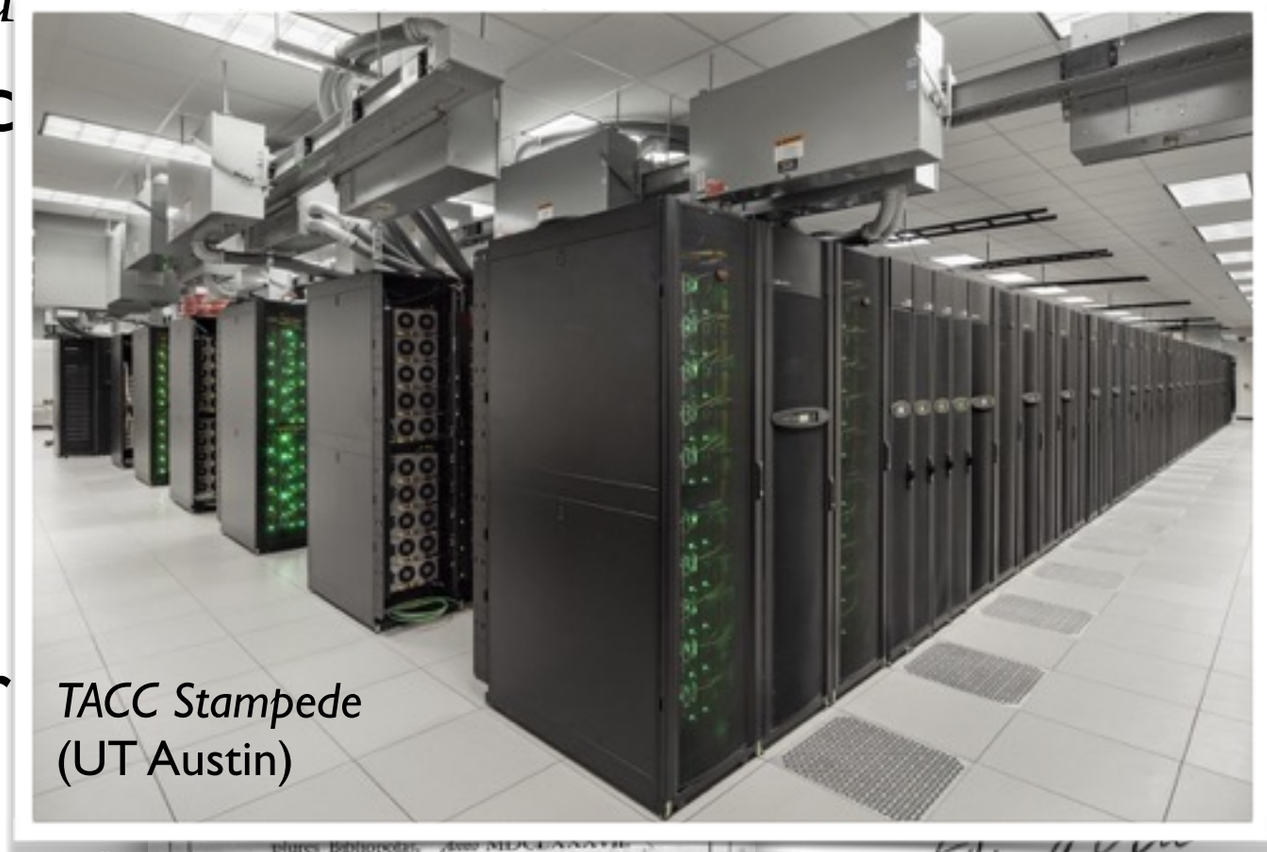
$$\mathbf{F}_i = m_i \mathbf{a}_i$$

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

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

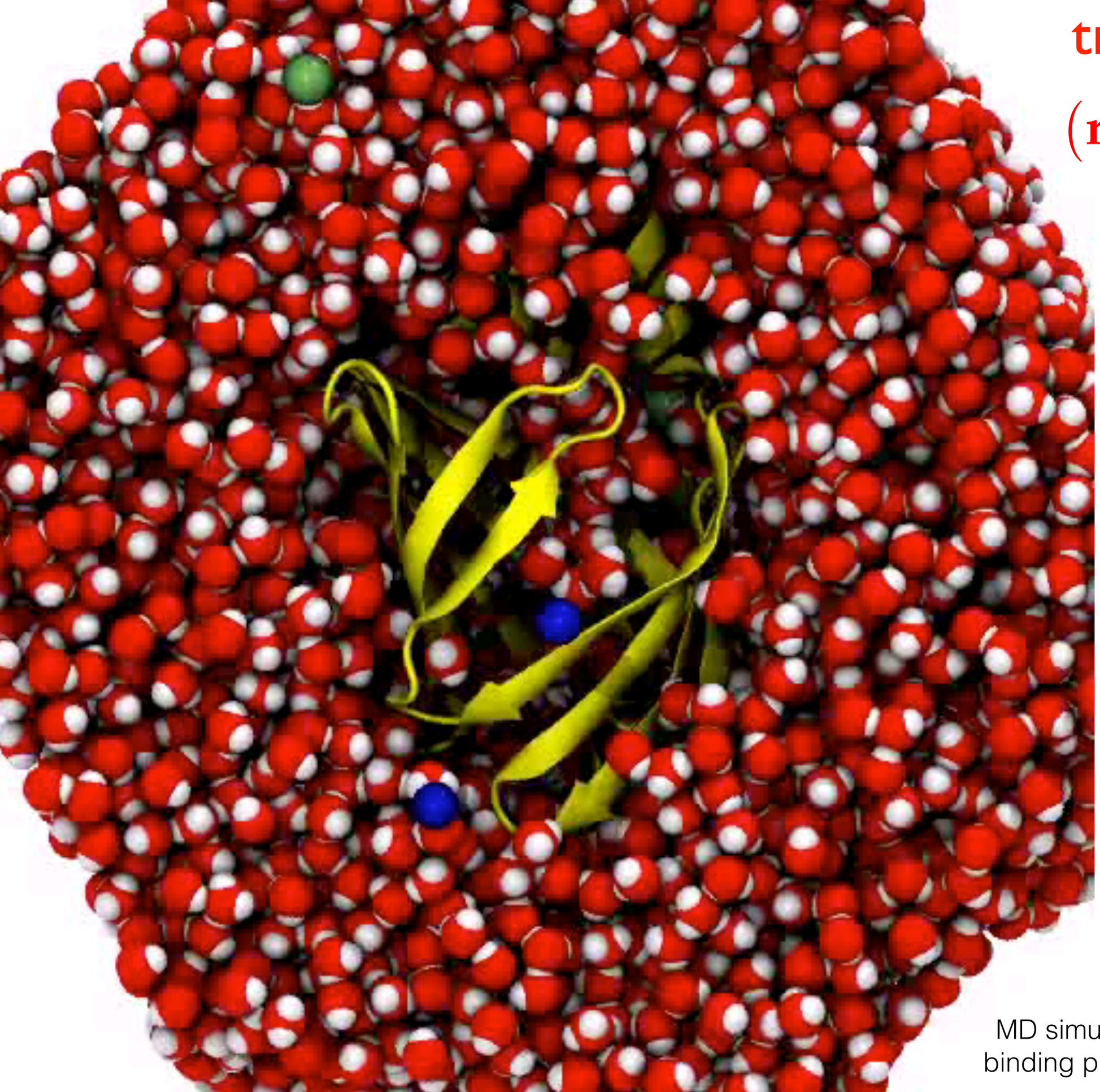
TACC Stampede
(UT Austin)

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

trajectory

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

source: Wikimedia Commons



trajectory

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

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

1 protein

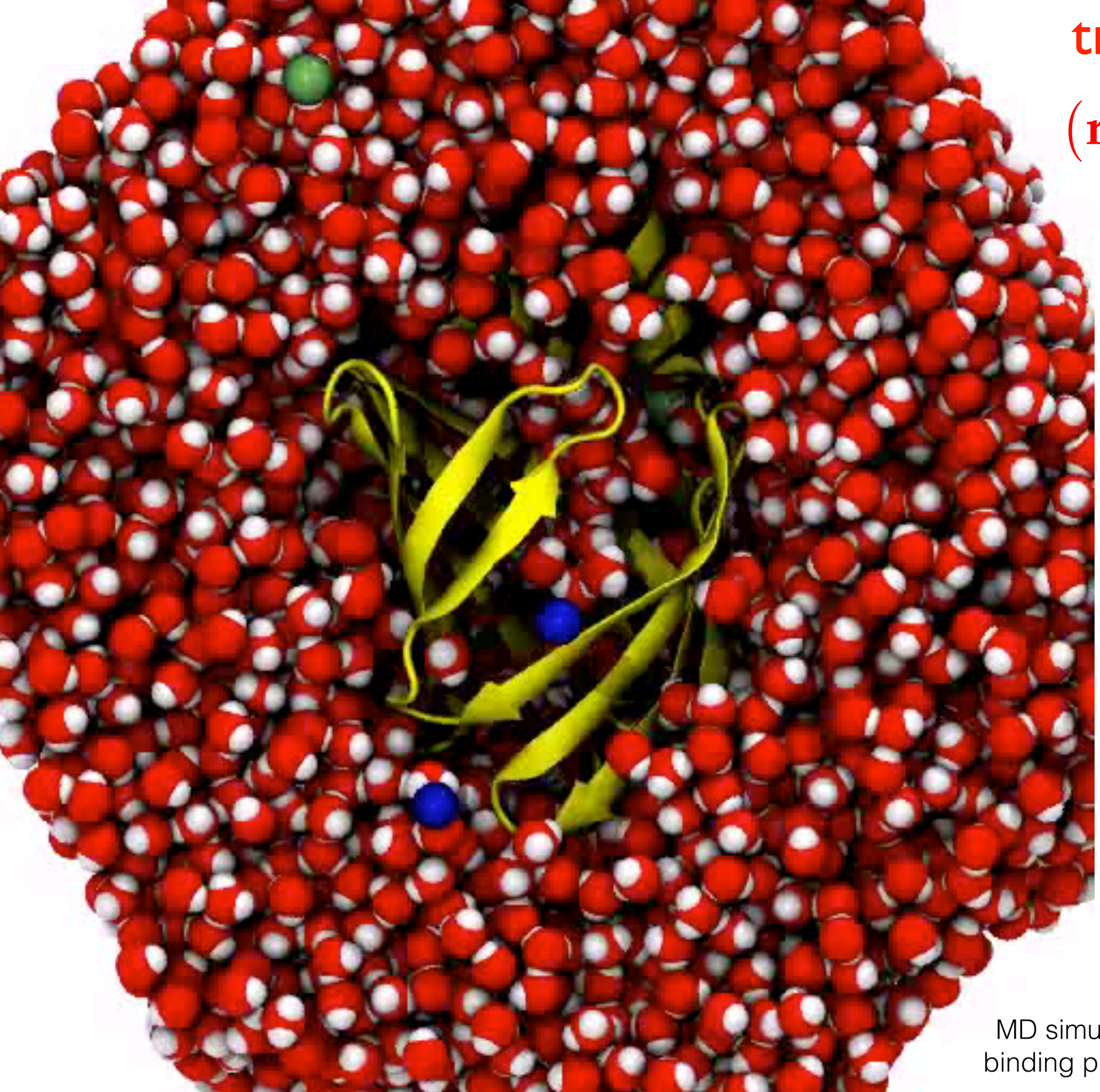
12 ions

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

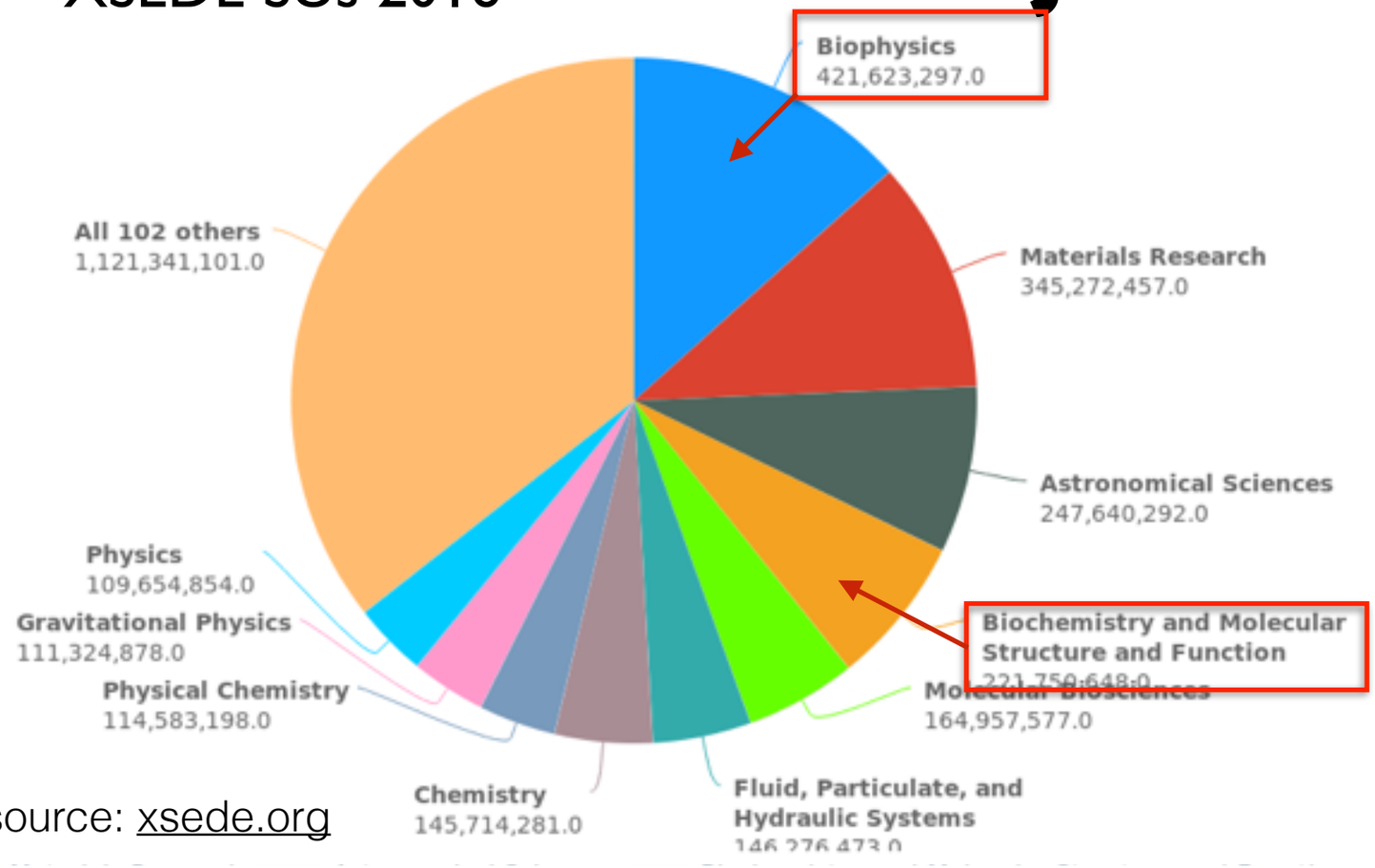
2113 atoms
12 atoms
10296 atoms

12421 atoms

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

XSEDE SUs 2016

MD trajectories can be big-ish



source: xsede.org

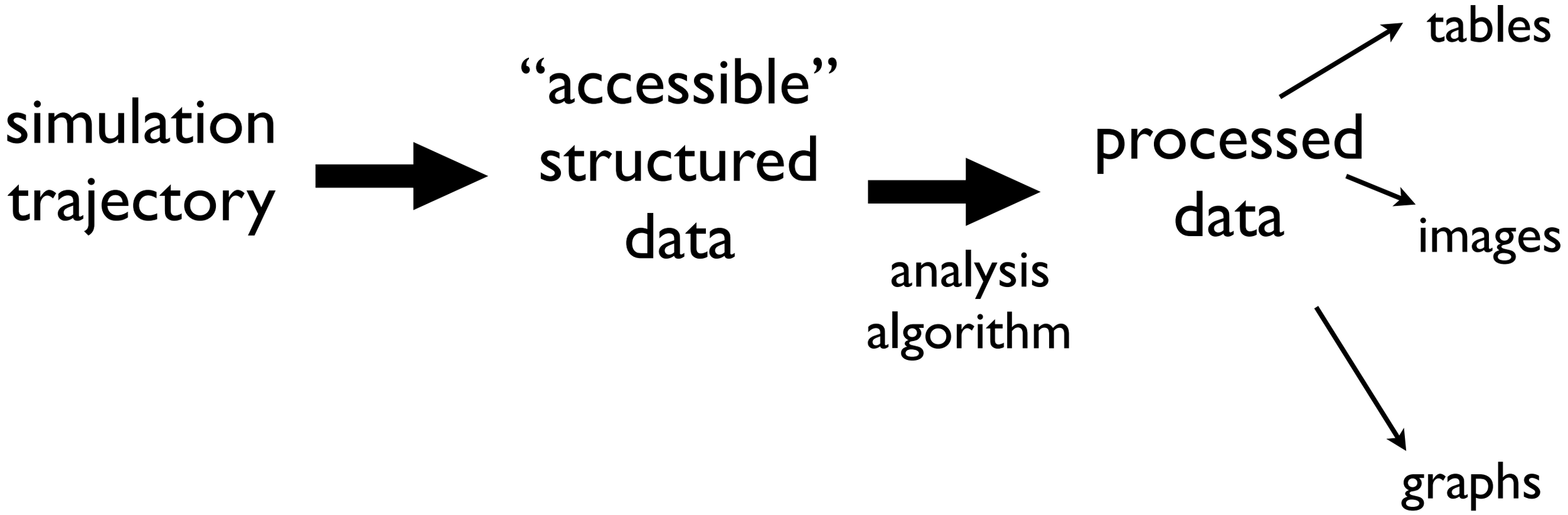
trajectory

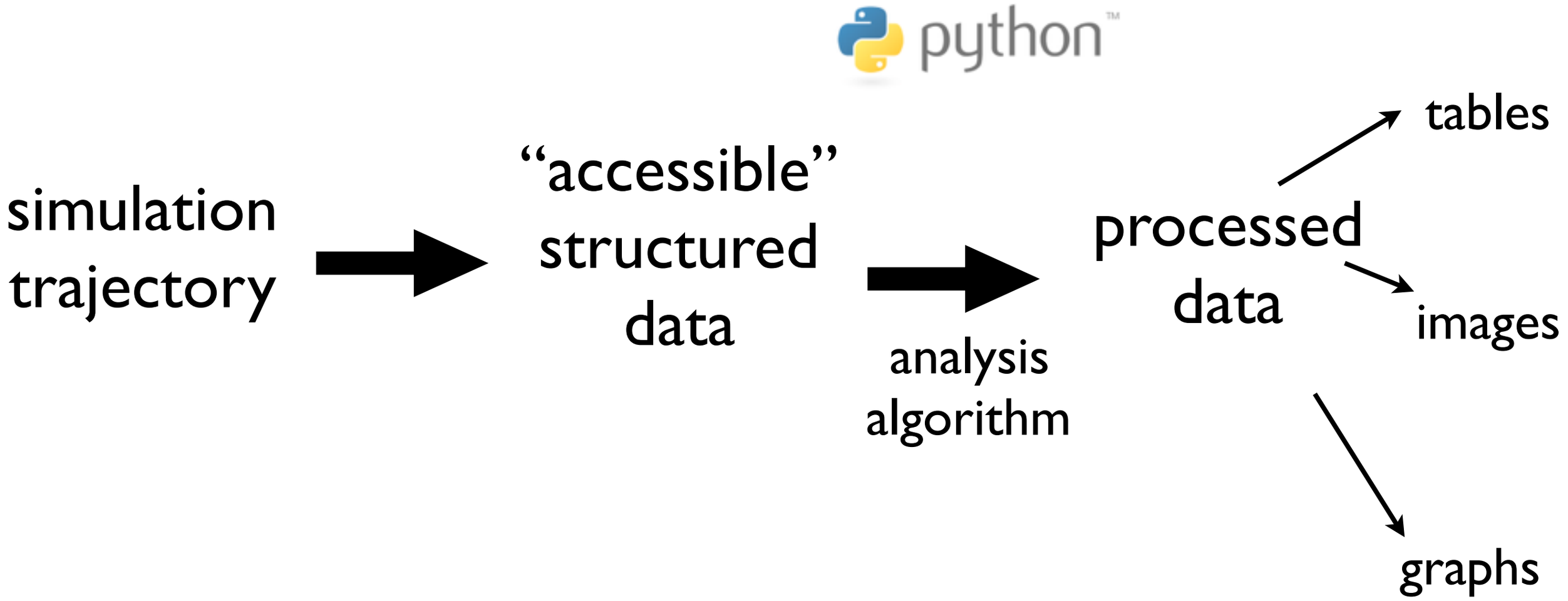
$$\left(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t)\right)$$

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

	max (2016)	typical (2016)
atoms *N*	~10^7	~10^5
simulated time τ	~10 μs	0.1–1 μs
trajectory frames	~10^9	~10^5
trajectory size	< 10 TiB	150 GiB

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


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 → analysis algorithm → processed data → tables, images, graphs

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, ...

"accessible" structured data

analysis algorithm

processed data

tables

images

graphs

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

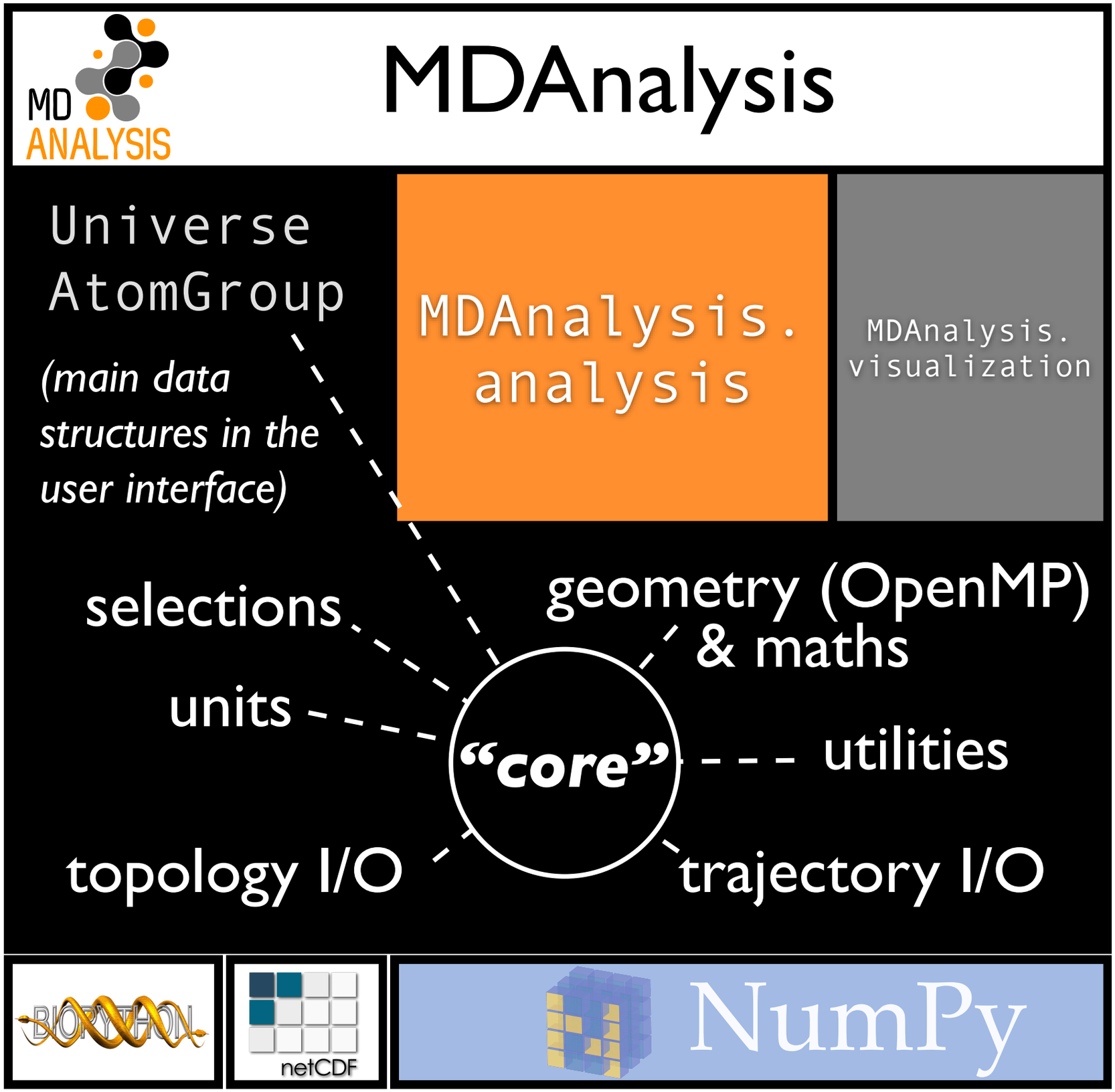
analysis algorithm

processed data

tables

images

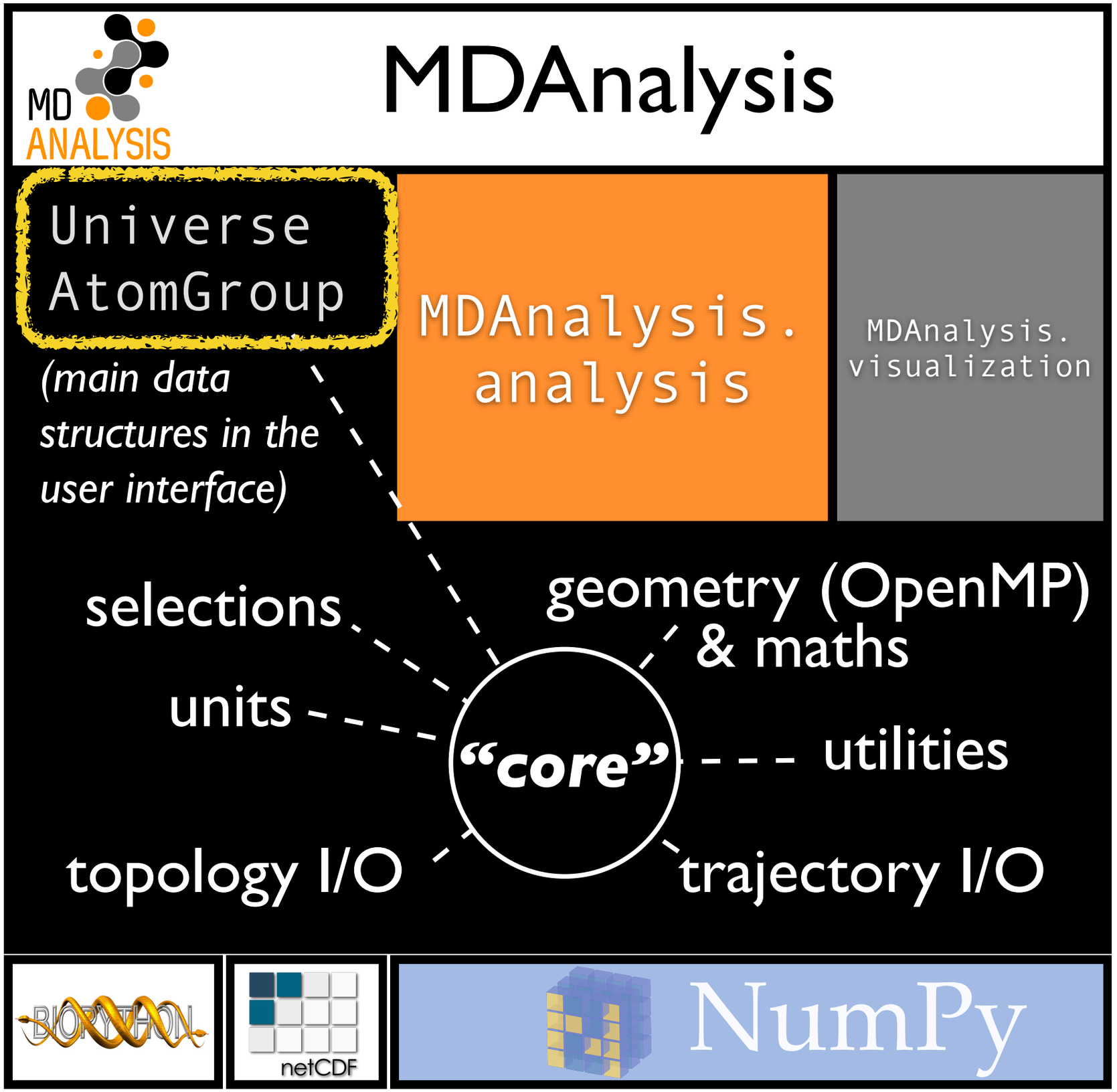
graphs



http://mdanalysis.org

Code base:

- python 2.7
- py 3 (~80%)
- cython
- C
- ~42k LOC
- ~24k lines comments



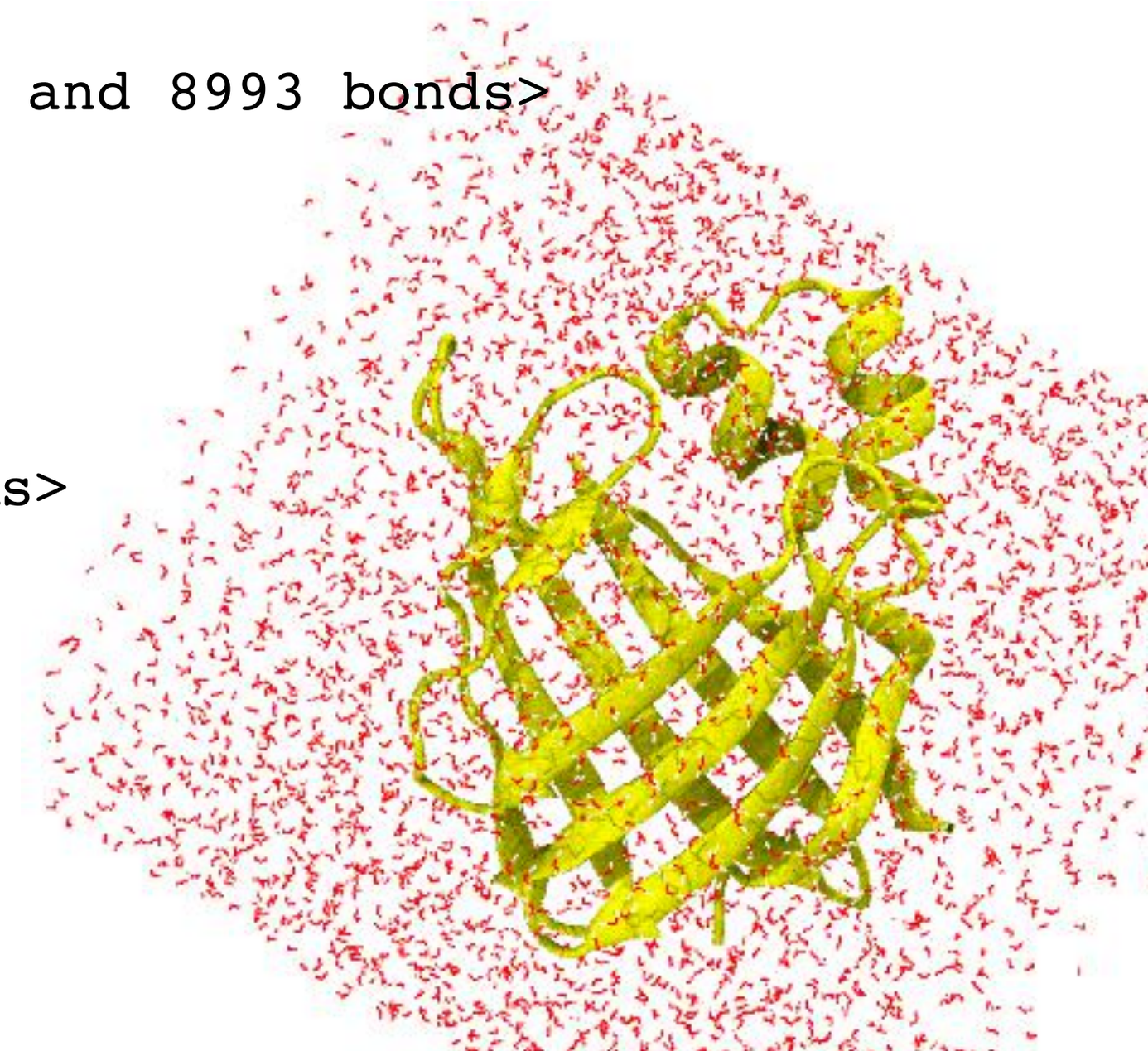
http://mdanalysis.org

Code base:

- python 2.7
- py 3 (~80%)
- cython
- C
- ~42k LOC
- ~24k lines comments

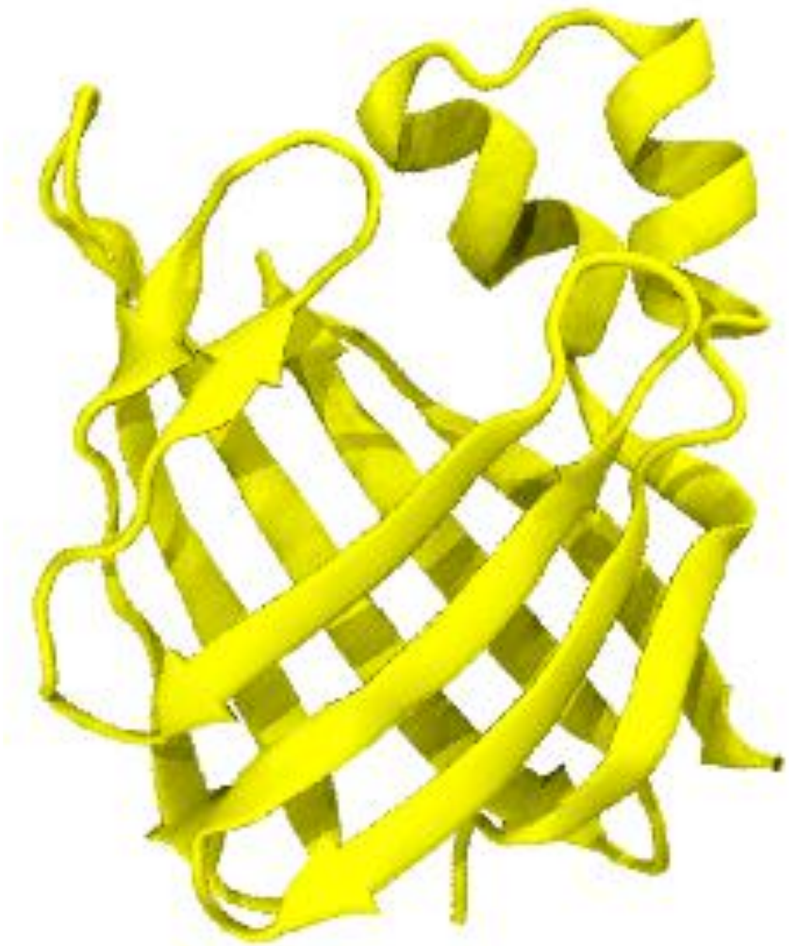
Fundamental data structures: *Universe*

```
import MDAnalysis as mda
u = mda.Universe('topol.tpr', 'traj.trr')

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

print(u.atoms)
<AtomGroup with 12421 atoms>
```



Fundamental data structures: *AtomGroup*

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



```
print(protein)
```

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

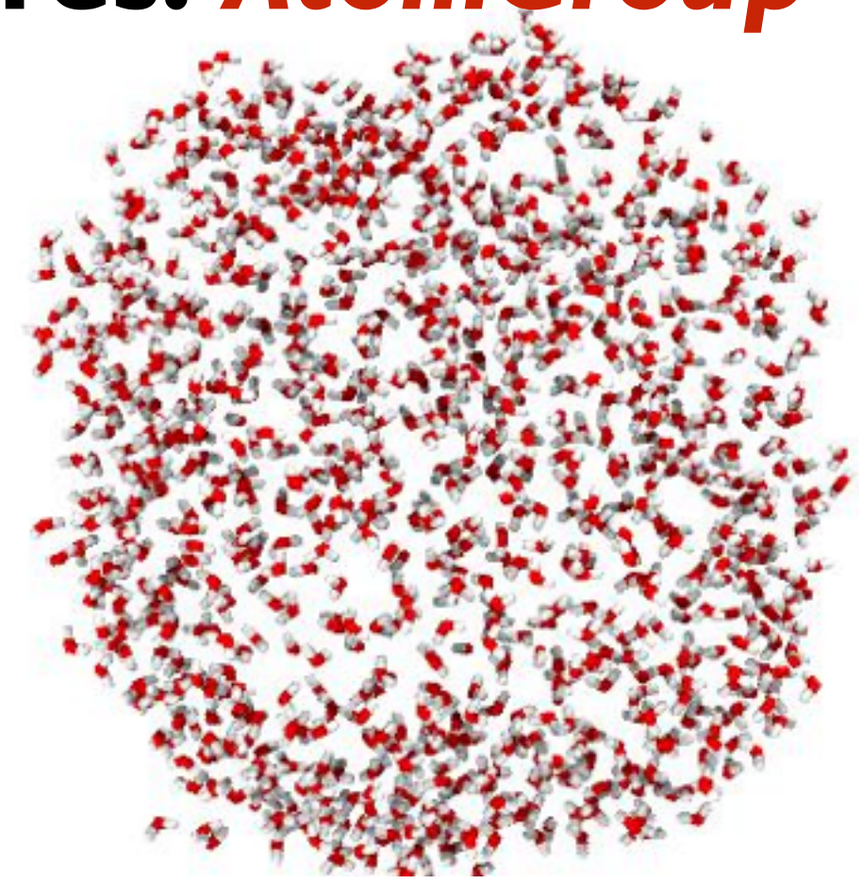
```
print(list(protein[:5]))
```

```
[<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*

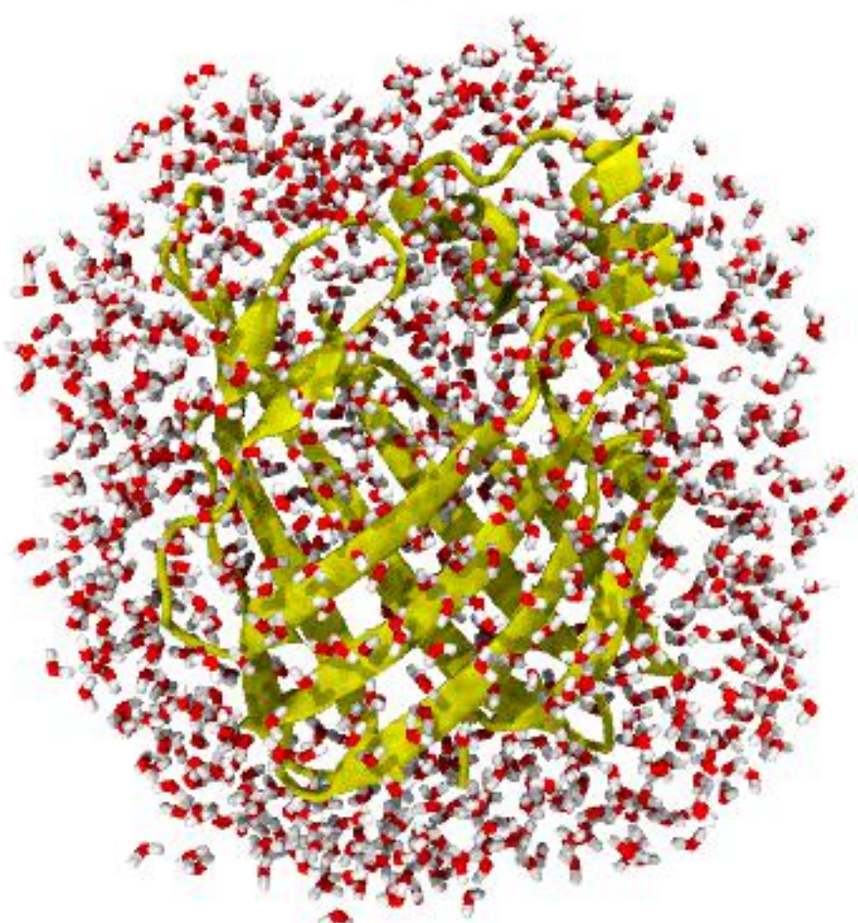
```
solvshell =
   u.atoms.select_atoms(
      “resname SOL and
       around 5.0 protein”)

print(solvshell)
```

```
<AtomGroup with 2792 atoms>
```



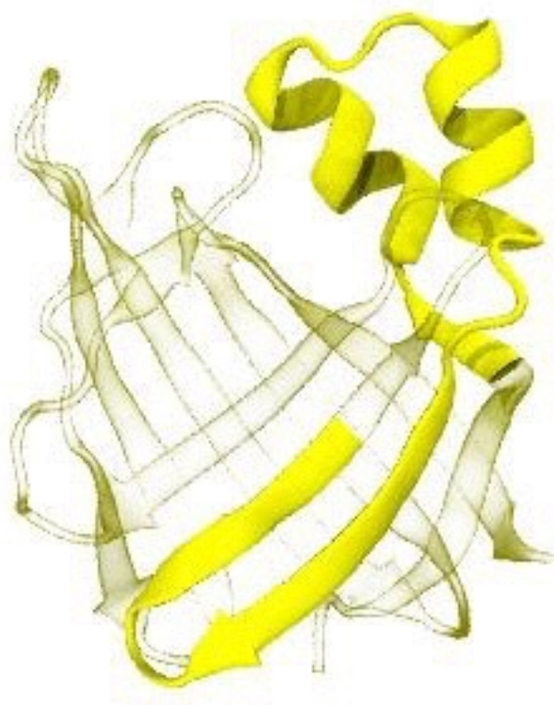
```
ag = protein + solvshell
print(ag)
```

```
<AtomGroup with 4905 atoms>
```



AtomGroups: Residues and Segments

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

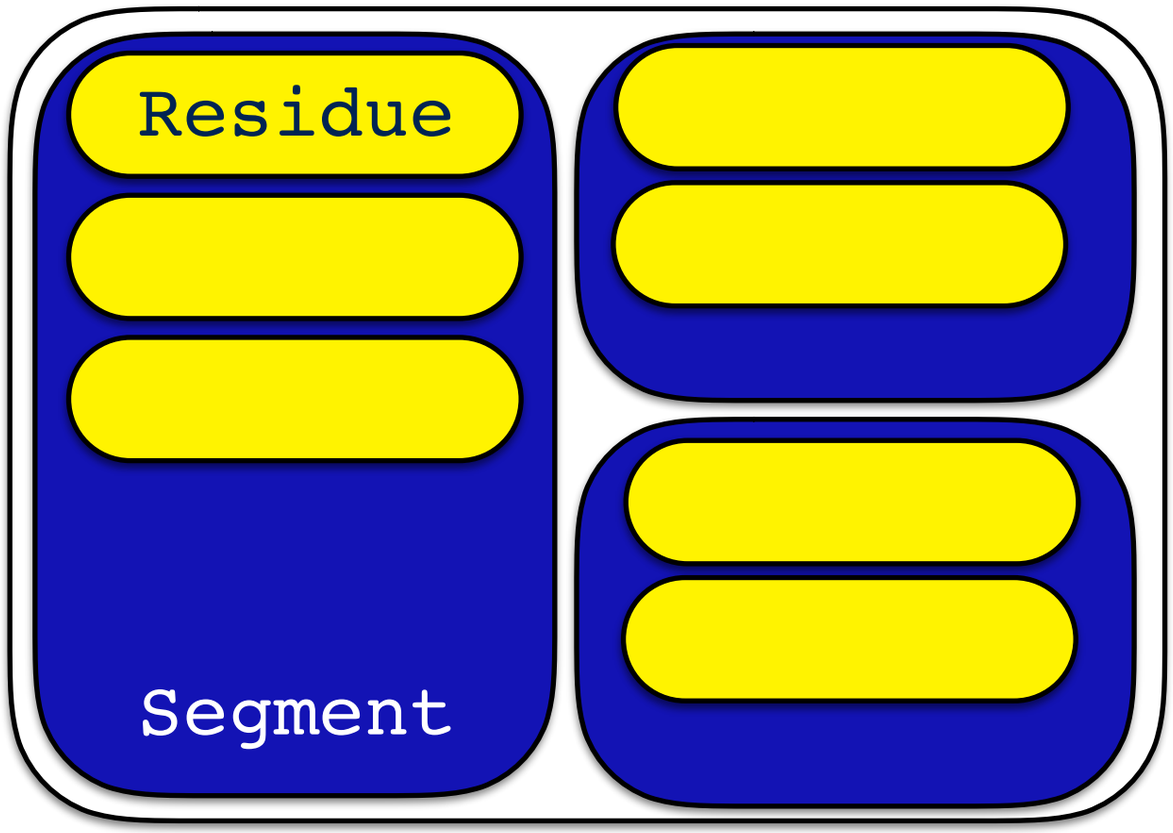


```
list(protein.residues[10:50])
```

```
[<Residue ASN, 11>, <Residue GLU, 12>,
<Residue ASN, 13>, <Residue TYR, 14>,
<Residue GLU, 15>, …
```

```
list(protein.segments)
```

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



Universe

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`

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

```
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)
```

Cα 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)
```

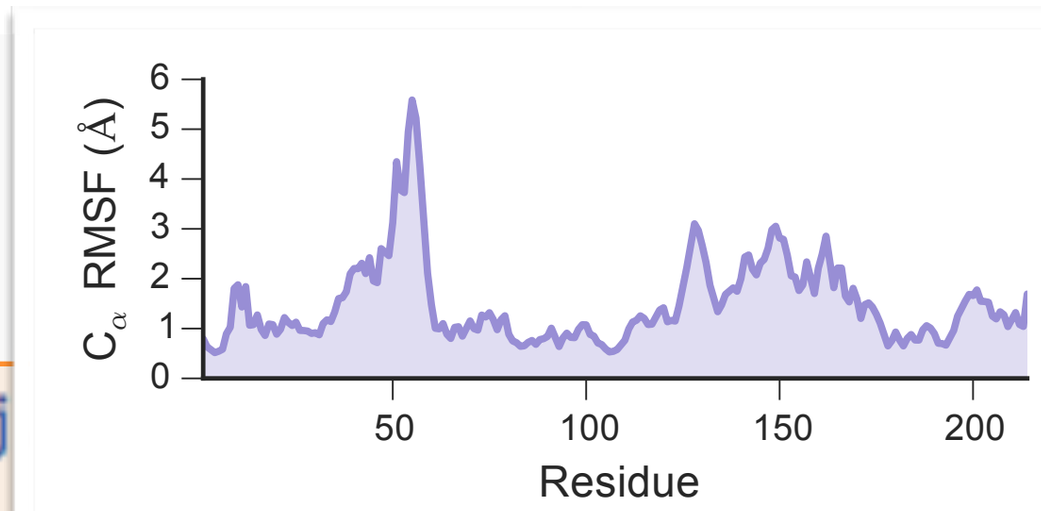
C_α RMSF

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

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)
```



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, ...

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)
```

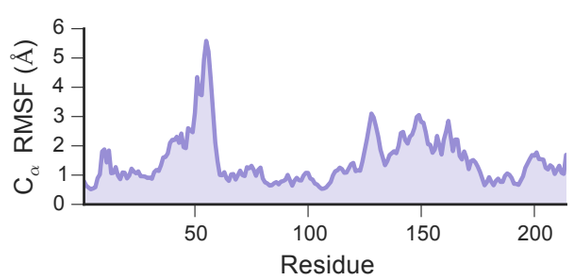
$$\rho_i = \sqrt{\left\langle (\mathbf{x}_i(t) - \langle \mathbf{x}_i \rangle)^2 \right\rangle}$$

C_α RMSF

processed data

- tables
- images
- graphs

RESULTS!



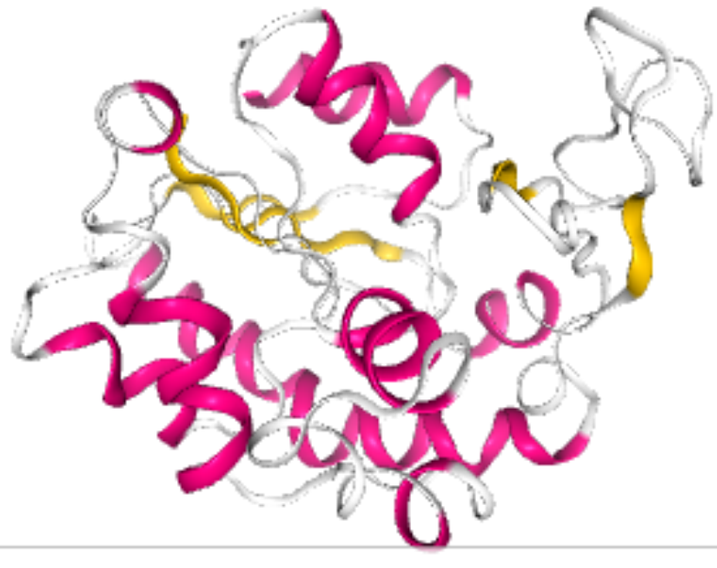
Interactive use

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

In [1]:
```
import nglview as nv
import MDAnalysis as mda
from MDAnalysisTests.datafiles import PSF, DCD
```

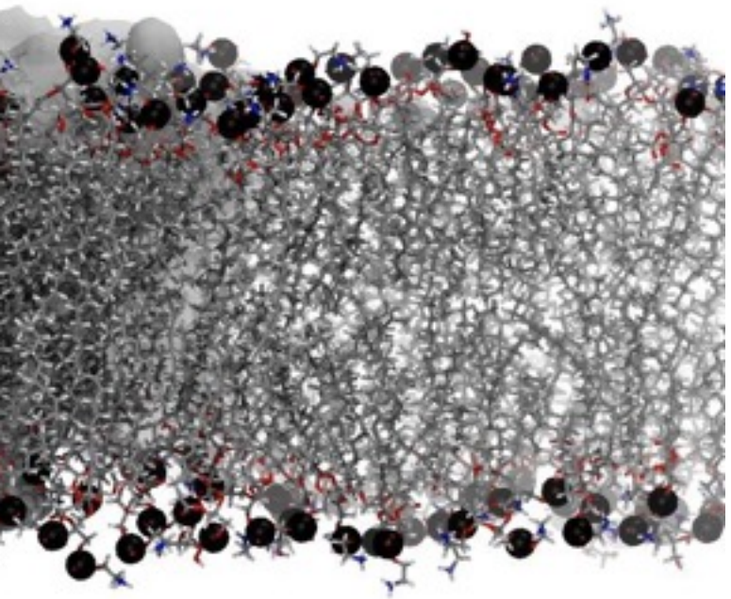
In [2]:
```
u = mda.Universe(PSF, DCD)
protein = u.select_atoms('protein')
```

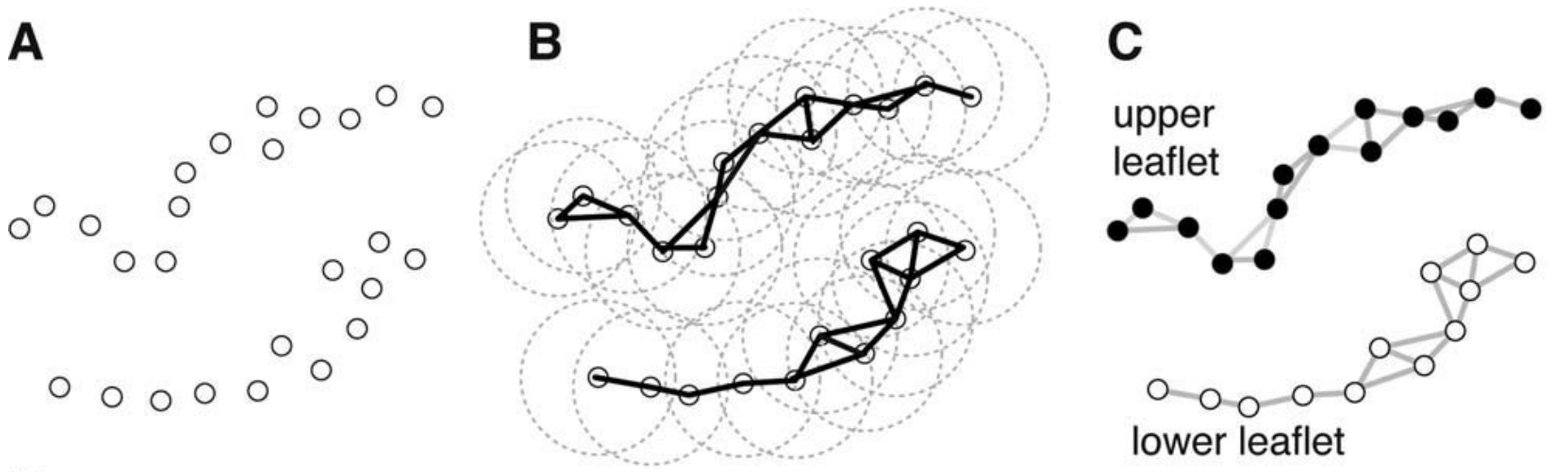
In [4]:
```
w = nv.show_mdanalysis(protein)
w
```



play

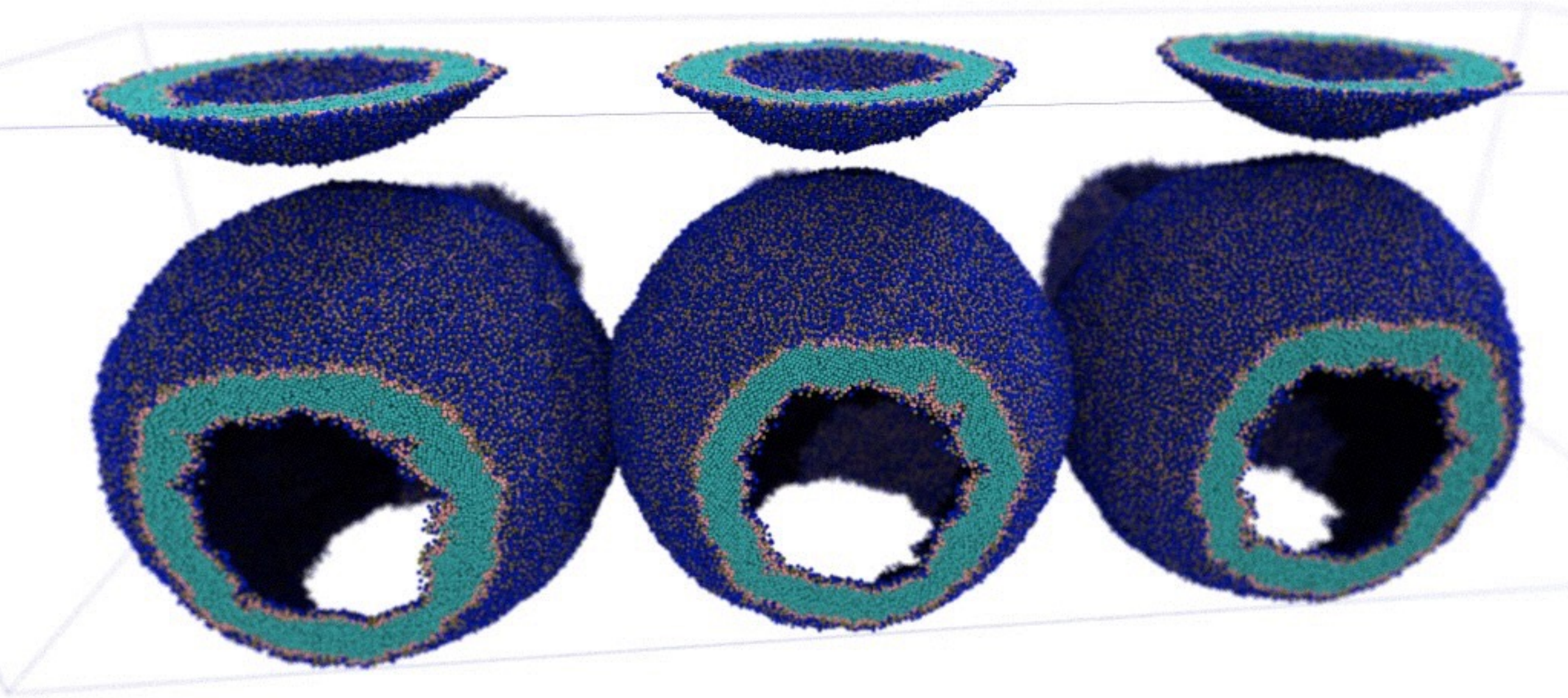
Interoperability with the Python Ecosystem

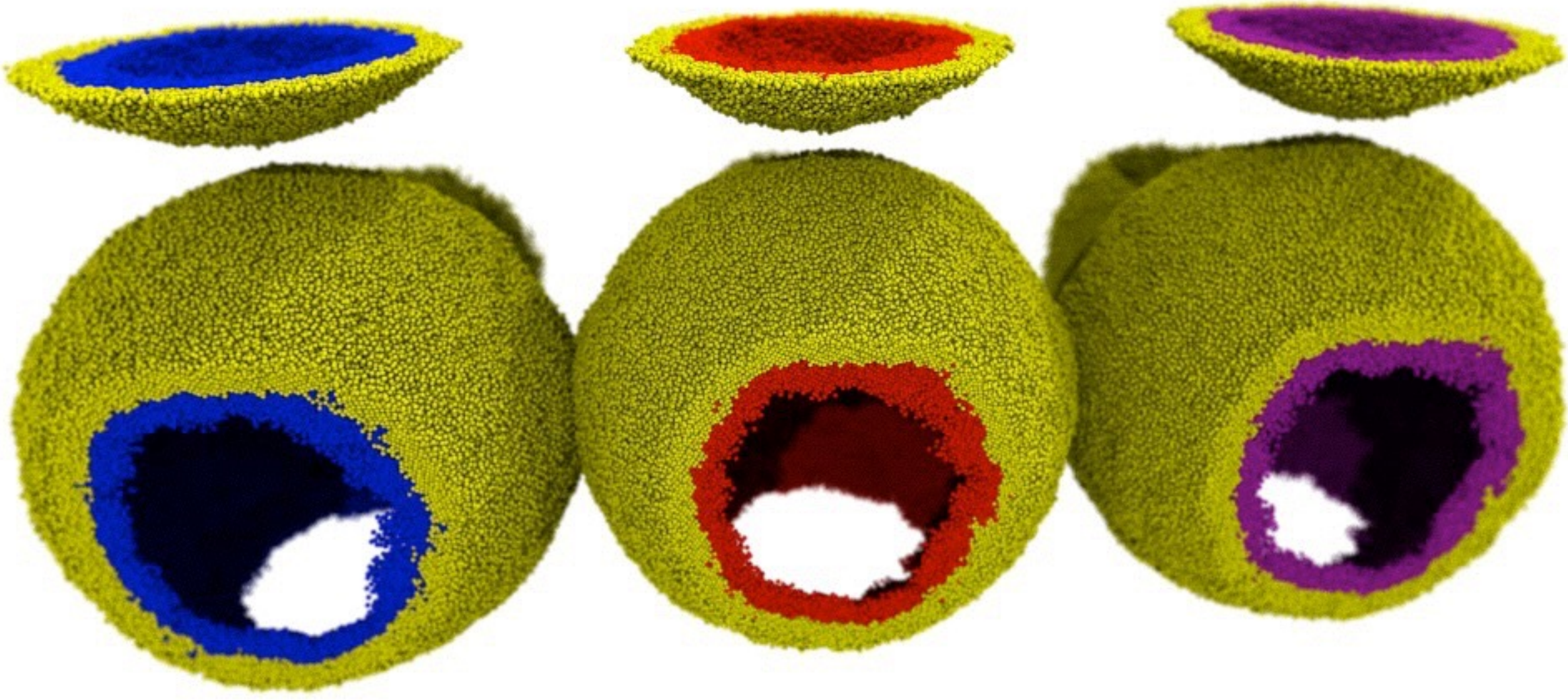




LeafletFinder algorithm

```
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()
A.plot()
print(A.rmsf)
```

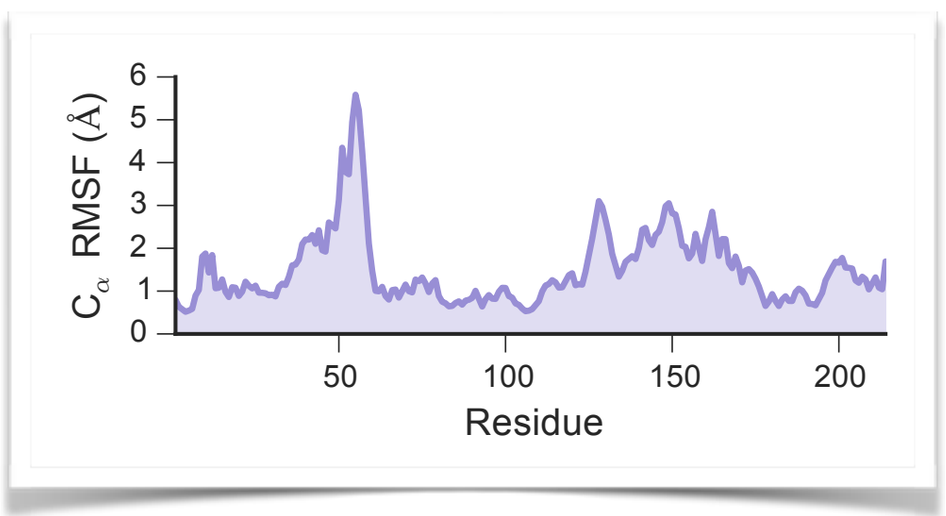
```
array([ 2.55843466,  3.07868589,
        2.37143333, ...,  3.60916556,
        4.05652126,  3.88028668])
```

Analysis module: *MDAnalysis.analysis*

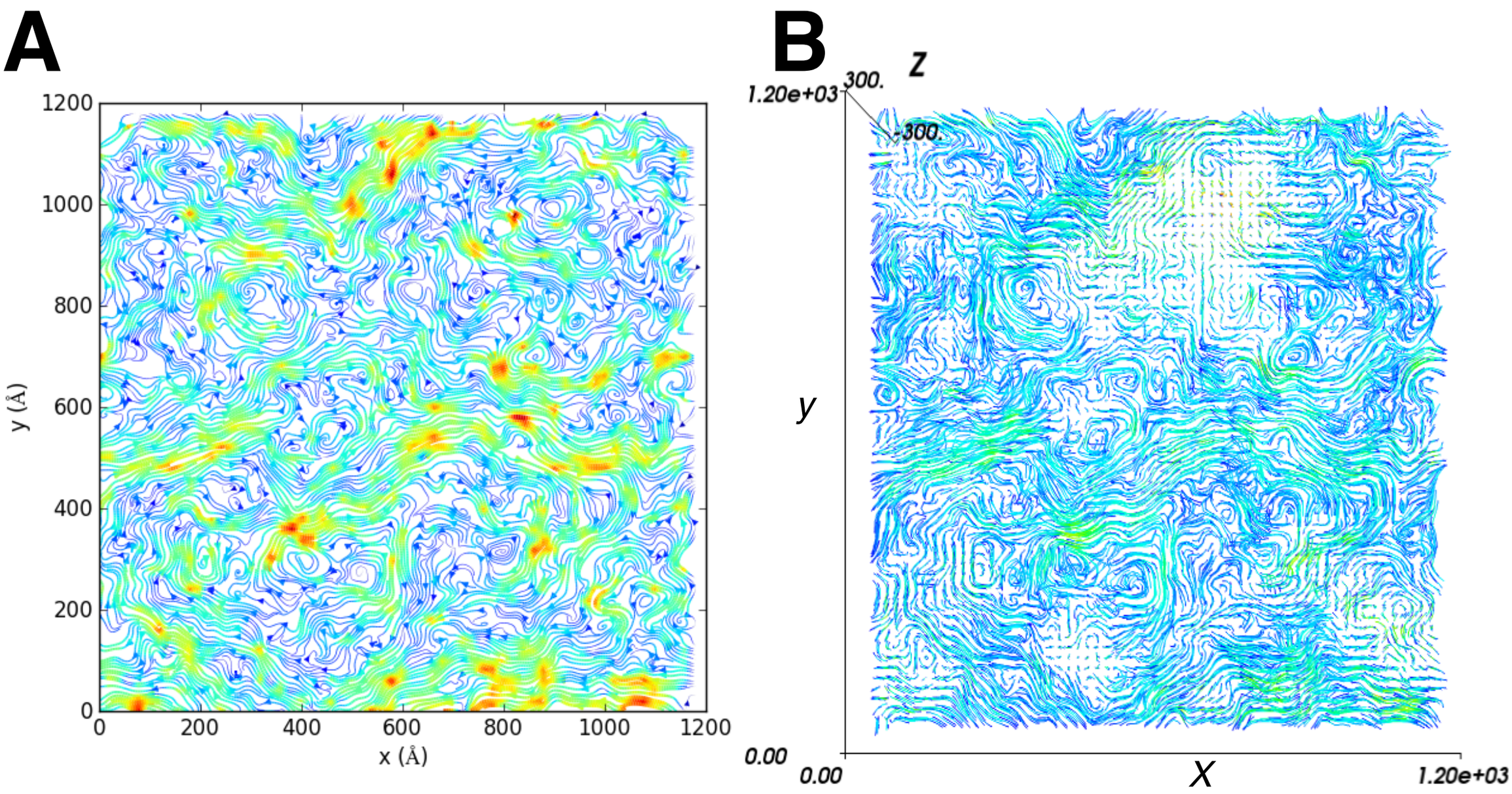
- standard analysis functionality (RMSD, RMSF, distances, density, hydrogen bonds, native contacts…)
- unique capabilities: Path Similarity Analysis, LeafletFinder, water dynamics, HOLE, diffusion map
- moving towards a standard API with an *AnalysisBase* class

```
from MDAnalysis.analysis.rms import RMSF
A = RMSF(protein).run()
A.plot()
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.

Recent performance improvements

- originally: *AtomGroup* as a Python list of *Atom* instances
- now:
  - NumPy array of integer atom indices
  - additional arrays for Atom properties
    - *Atom* only manages access to arrays (but is seldom used)
    - *Topology* instance manages all arrays

Array of Structs → **Struct of Arrays**

Recent performance improvements

subselecting AtomGroup from existing one

# atoms	v0.15.0	v0.16.0	speed up
1.75 M	19 ms	0.45 ms	42
3.50 M	18 ms	0.54 ms	33
10.1 M	17 ms	0.45 ms	38

```
ag = u.atoms[np.arange(3, 574393, 7)]
```

accessing attributes

# atoms	v0.15.0	v0.16.0	speed up
1.75 M	250 ms	35 ms	7.1
3.50 M	490 ms	72 ms	6.8
10.1 M	1500 ms	300 ms	5.0

```
resnames = u.atoms.resnames
```

Recent performance improvements

loading topology file

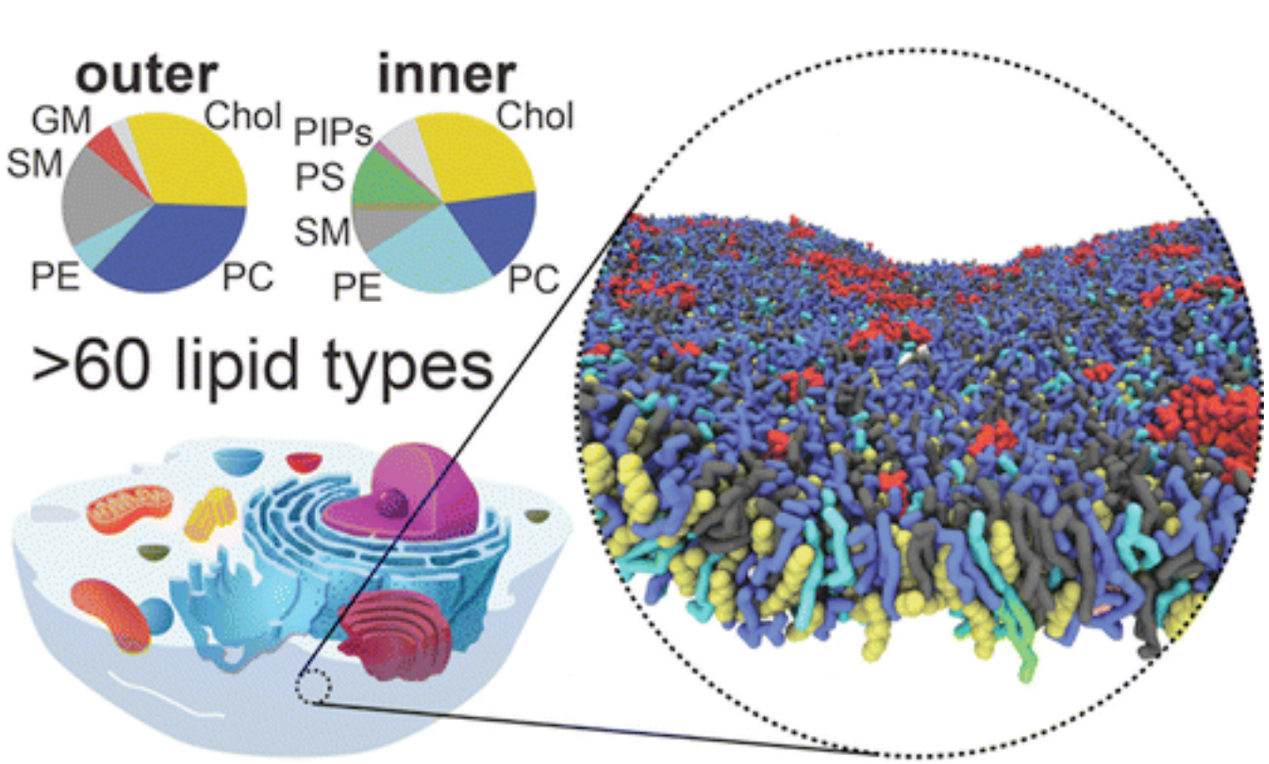
# atoms	v0.15.0	v0.16.0	speed up
1.75 M	18 s	5 s	3.6
3.50 M	36 s	11 s	3.3
10.1 M	105 s	31 s	3.4

```
u = Universe(“system.gro”)

MDAnalysis.topology.GROParser.GROParser(“system.gro”).parse()
```

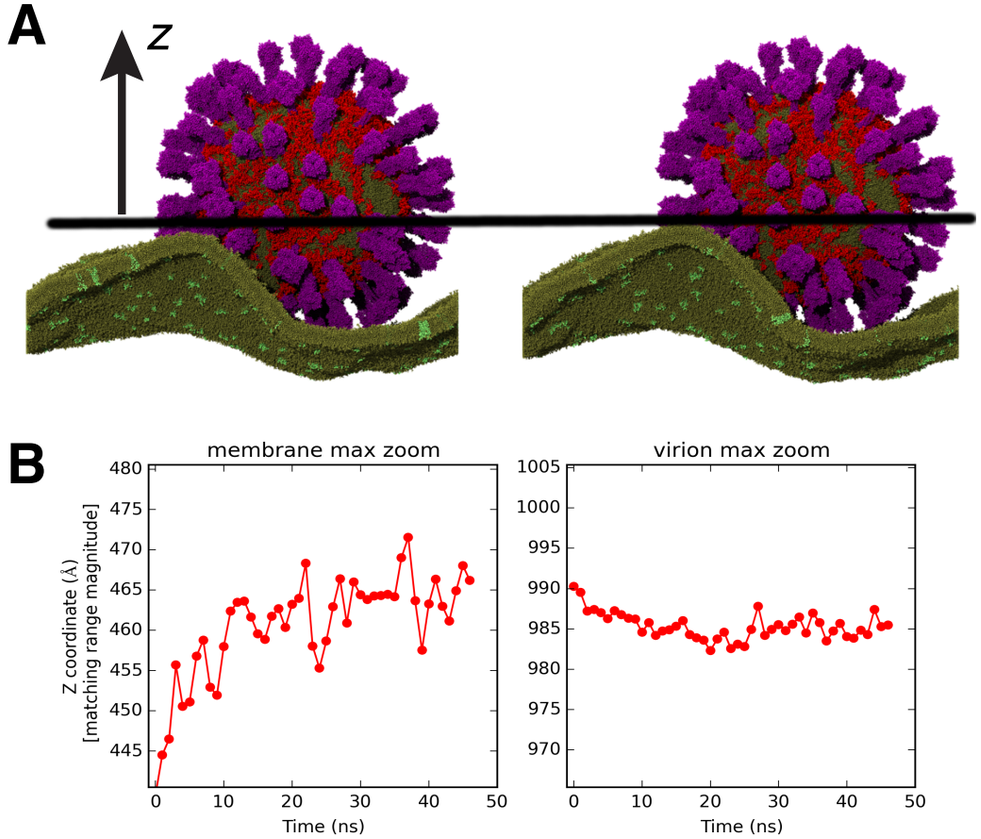
… + reduced memory footprint down to 1/3

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

- Python Package Index

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

- Conda

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

- from source (develop or master == release)

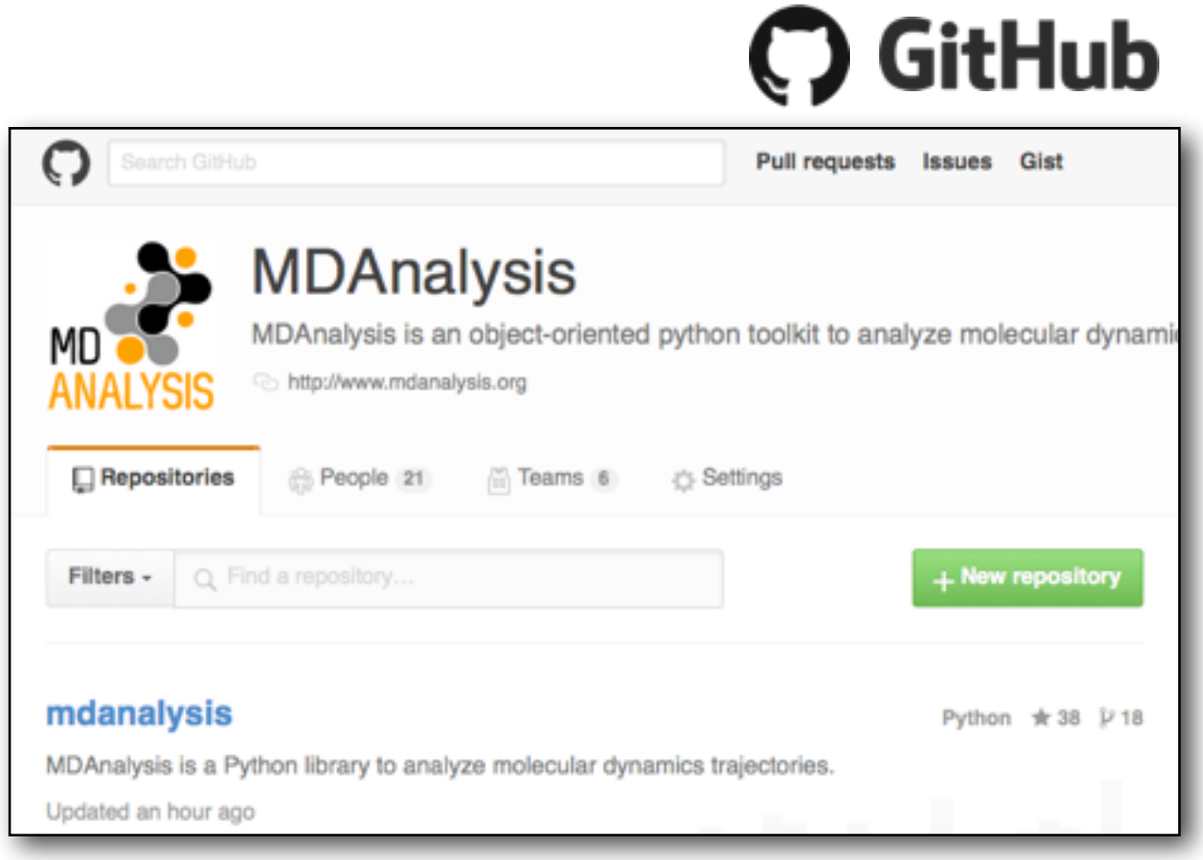
```
git clone https://github.com/MDAnalysis/mdanalysis.git
cd mdanalysis/package
python setup.py install
```

Runs on

- Linux
- Mac OS X

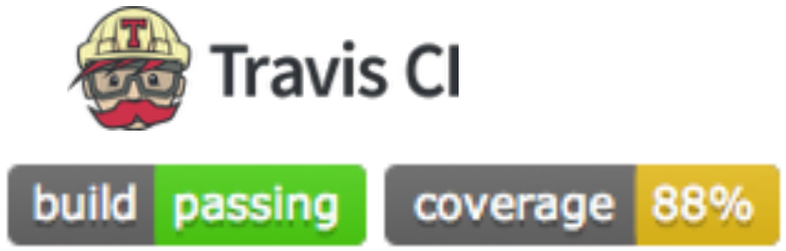
Open source

- GPL v2
- github.com/MDAnalysis

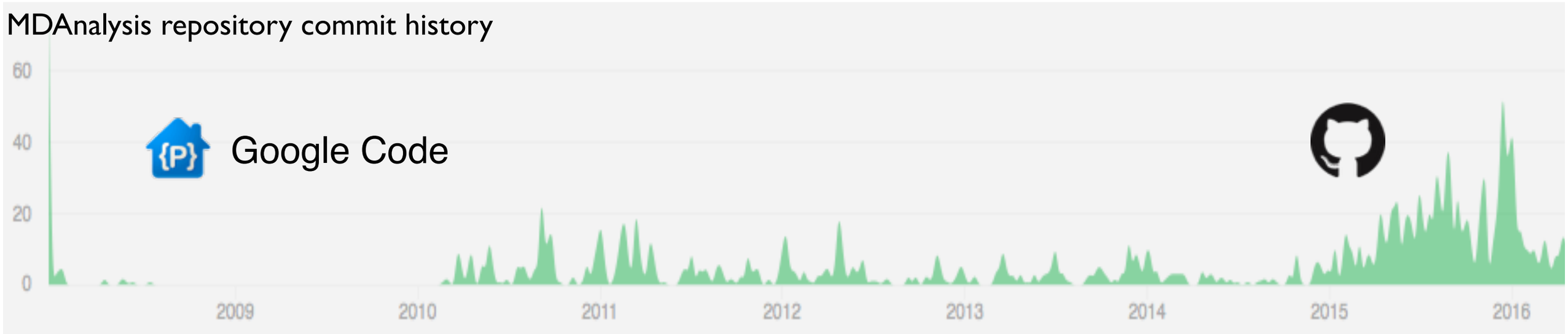


Development process

- pull request / review / merge
- continuous integration with > 3,500 unit tests



Impact of “Social Coding”?




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

initial import
```

GitHub

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

(>200 citations on GoogleScholar (July 2016))

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

- 44 contributing authors (July 2016)
- 7 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, Joshua Adelman, **Jonathan Barnoud**, Balasubramanian, Christian Beckstein (logo), Alejandro Bernardin, Bart Bruininks, Sébastien Buchoux, David Caplan, Matthieu Chavent, John Detlefs, **David L. Dotson**, Xavier Deupi, Jan Domański, Lennard van der Feltz, Philip Fowler, Joseph Goose, **Richard J. Gowers**, Lukas Grossar, Benjamin Hall, Kyle J. Huston, Joe Jordan, **Max Linke**, Jinju Lu, Robert McGibbon, Fiona Naughton, Alex Nesterenko, **Manuel Nuno Melo**, Caio S. Souza, Hai Nguyen, Mattia F. Palermo, Danny Parton, Joshua L. Phillips, **Tyler Reddy**, Paul Rigor, Utkarsh Saxena, Sean L. Seyler, Andy Somogyi, Lukas Stelzl, Gorman Stock, Isaac Virshup, Zhuyi Xue, Carlos Yáñez S, and Oliver Beckstein

Join us at

mdanalysis.org

github.com/MDAnalysis

ACI-1443054

GSoC 2016