Bioexcel Webinar

MDAnalysis

Interoperable analysis of biomolecular simulations in Python

Oliver Beckstein[1], Lily Wang[2], Irfan Alibay[3]

[1]Department of Physics, Arizona State University
[2]Research School of Chemistry, Australian National University
[3]Department of Biochemistry, The University of Oxford

1. Fundamentals
2. Extending MDAnalysis
3. Future directions

Acknowledgements

137 code contributors and countless community members

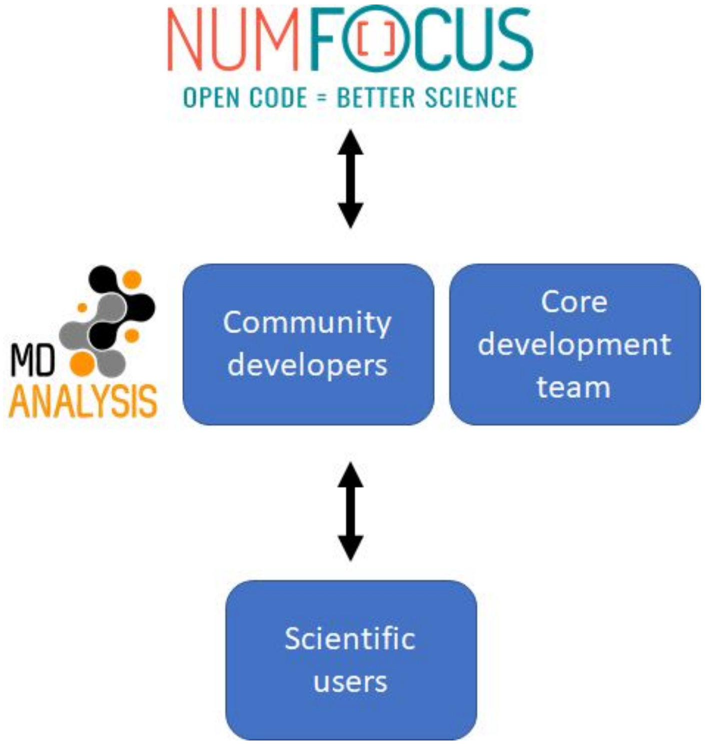
Naveen Michaud-Agrawal, Elizabeth J. Denning, **Oliver Beckstein**, Danny Parton, Philip Fowler, **Tyler Reddy**, Joseph Goose, **Jan Domanski**, Benjamin Hall, Paul Rigor, David Caplan, Christian Beckstein (logo), **Sébastien Buchoux**, Joshua L. Adelman, Lukas Grossar, Andy Somogyi, Lukas Stelzl, Jinju Lu, Joshua L. Phillips, Zhuyi Xue, Xavier Deupi, **Manuel Nuno Melo**, Robert McGibbon, **Richard J. Gowers**, Alejandro Bernardin, Lennard van der Feltz, Matthieu Chavent, Joe Jordan, Alex Nesterenko, Caio S. Souza, Sean L. Seyler, **David L. Dotson**, Carlos Yanez S., Kyle J. Huston, Isaac Virshup, **Max Linke**, Gorman Stock, **Jonathan Barnoud**, Hai Nguyen, Balasubramanian, Mattia F. Palermo, Utkarsh Saxena, Abhinav Gupta, John Detlefs, Eugen Hruska, Bart Bruininks, **Fiona B. Naughton**, Robert Delgado, Wouter Boomsma, **Matteo Tiberti**, Tone Bengtsen, Shantanu Srivastava, Pedro Reis, Ruggero Cortini, Zhiyi Wu, Kashish Punjani, Utkarsh Bansal, Shobhit Agarwal, Vedant Rathore, Akshay Gupta, Juan Eiros Zamora, Jon Kapla, Sang Young Noh, Andrew William King, Kathleen Clark, Dominik 'Rathann' Mierzejewski, Nestor Wendt, **Micaela Matta**, Jose Borreguero, Sören von Bülow, Nabarun Pal, Mateusz Bieniek, Paul Smith, Navya Khare, **Johannes Zeman**, Ayush Suhane, Davide Cruz, Shujie Fan, Andrew R. McCluskey, Henry Mull, **Irfan Alibay**, Philip Loche, Matthew W. Thompson, Ali Ehlen, Daniele Padula, Ninad Bhat, Fenil Suchak, Yibo Zhang, Luís Pedro Borges Araújo, Abhishek A. Kognole, Rocco Meli, **Lily Wang**, Matthijs Tadema, Joao Miguel Correia Teixeira, Charlie Cook, Yuanyu Chang, Guillaume Fraux, Ivan Hristov, Michael Quevillon, Hao Tian, Hugo MacDermott-Opeskin, Anshul Angaria, Shubham Sharma, Yuxuan Zhuang, Cédric Bouysset, Abhishek Shandilya, Morgan L. Nance, Faraaz Shah, Wiep van der Toorn, Siddharth Jain, Ameya Harmalkar, Shakul Pathak, Andrea Rizzi, William Glass, Marcello Sega, Edis Jakupovic, Nicholas Craven, Mieczyslaw Torchala, Ramon Crehuet, Haochuan Chen, Karthikeyan Singaravelan, Aditya Kamath, Leonardo Barneschi, Henrik Jäger, Jan Stevens, Orion Cohen, Dimitrios Papageorgiou, Hannah Pollak, Estefania Barreto-Ojeda, Paarth Thadani, Henry Kobin, Kosuke Kudo, Sulay Shah, Alexander Yang, Filip T. Szczypiński, Marcelo C. R. Melo, Mark D. Driver

NUMFOCUS
OPEN CODE = BETTER SCIENCE

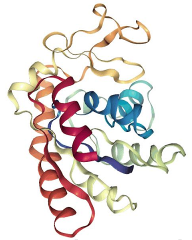
The **MDAnalysis** organization

https://mdanalysis.org
@mdanalysis

- Focus on developing tools to handle simulation data
  - MDAnalysis library
  - GridDataFormats, distopia, pmda, etc…
  - https://github.com/MDAnalysis/
- Community-led development
  - Majority non-funded work
  - CZI EOSS-4 grant (next 2 years)
- NumFOCUS fiscally-sponsored project



The **MDAnalysis** library



simulation trajectory

MD ANALYSIS

NumPy

python™

“accessible” structured data
`numpy.ndarray()`

analysis algorithm

processed data

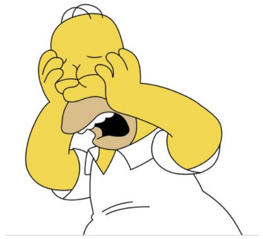
tables

images

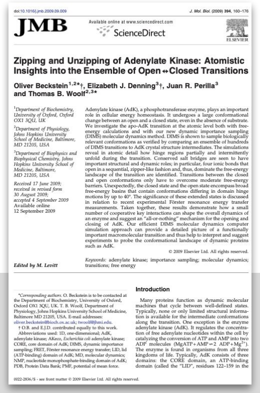
graphs

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!



doi:10.1016/j.jmb.2009.09.009 *J. Mol. Biol.* (2009) **394**, 160–176

JMB

Available online at www.sciencedirect.com
ScienceDirect

ELSEVIER

Zipping and Unzipping of Adenylate Kinase: Atomistic Insights into the Ensemble of Open↔Closed Transitions

Oliver Beckstein[1,2]*†, Elizabeth J. Denning[3]†, Juan R. Perilla[3] and Thomas B. Woolf[2,3]*

[1]*Department of Biochemistry, University of Oxford, Oxford OX1 3QU, UK*
[2]*Department of Physiology, Johns Hopkins University School of Medicine, Baltimore, MD 21205, USA*
[3]*Department of Biophysics and Biophysical Chemistry, Johns Hopkins University School of Medicine, Baltimore, MD 21205, USA*

Received 17 June 2009; received in revised form 30 August 2009; accepted 4 September 2009
Available online 12 September 2009

Adenylate kinase (AdK), a phosphotransferase enzyme, plays an important role in cellular energy homeostasis. It undergoes a large conformational change between an open and a closed state, even in the absence of substrate. We investigate the apo-AdK transition at the atomic level both with free-energy calculations and with our new dynamic importance sampling (DIMS) molecular dynamics method. DIMS is shown to sample biologically relevant conformations as verified by comparing an ensemble of hundreds of DIMS transitions to AdK crystal structure intermediates. The simulations reveal in atomic detail how hinge regions partially and intermittently unfold during the transition. Conserved salt bridges are seen to have important structural and dynamic roles; in particular, four ionic bonds that open in a sequential, zipper-like fashion and, thus, dominate the free-energy landscape of the transition are identified. Transitions between the closed and open conformations only have to overcome moderate free-energy barriers. Unexpectedly, the closed state and the open state encompass broad free-energy basins that contain conformations differing in domain hinge motions by up to 40°. The significance of these extended states is discussed in relation to recent experimental Förster resonance energy transfer measurements. Taken together, these results demonstrate how a small number of cooperative key interactions can shape the overall dynamics of an enzyme and suggest an "all-or-nothing" mechanism for the opening and closing of AdK. Our efficient DIMS molecular dynamics computer simulation approach can provide a detailed picture of a functionally important macromolecular transition and thus help to interpret and suggest experiments to probe the conformational landscape of dynamic proteins such as AdK.

© 2009 Elsevier Ltd. All rights reserved.

Edited by M. Levitt

Keywords: adenylate kinase; importance sampling; molecular dynamics; transitions; free energy

*Corresponding authors. O. Beckstein is to be contacted at the Department of Biochemistry, University of Oxford, Oxford OX1 3QU, UK. T. B. Woolf, Department of Physiology, Johns Hopkins University School of Medicine, Baltimore MD 21205, USA. E-mail addresses: oliver.beckstein@bioch.ox.ac.uk; twoolf@jhmi.edu.
† O.B. and E.J.D. contributed equally to this work.
Abbreviations used: 1D, one-dimensional; AdK, adenylate kinase; AKeco, *Escherichia coli* adenylate kinase; CORE, core domain of AdK; DIMS, dynamic importance sampling; FRET, Föster resonance energy transfer; LID, lid (ATP-binding) domain of AdK; MD, molecular dynamics; NMP, nucleotide monophosphate-binding domain of AdK; PDB, Protein Data Bank; PMF, potential of mean force.

Introduction

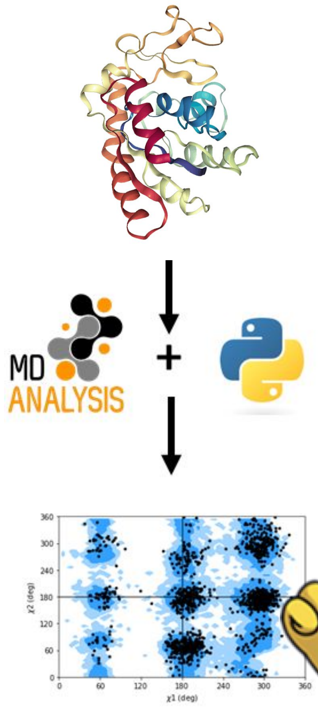
Many proteins function as dynamic molecular machines that cycle between well-defined states. Typically, none or only limited structural information is available for the intermediate conformations along the transition. One exception is the enzyme adenylate kinase (AdK). It regulates the concentration of free adenylate nucleotides within the cell by catalyzing the conversion of ATP and AMP into two ADP molecules ($MgATP + AMP \rightleftharpoons 2\ ADP + Mg^{2+}$). The enzyme is found in organisms from all three kingdoms of life. Typically, AdK consists of three domains: the CORE domain, an ATP-binding domain (called the "LID", residues 122–159 in the

0022-2836/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.

Insights & publication!

The MDAnalysis library

- Open source (GPLv2+) **Python** library for handling simulation data
  - Focus on analysing molecular dynamics data
  - … but really any *N*=const particle-based “trajectories”
- Components to build custom analyses and workflows
  - Low level: trajectory data, distance calculations (with PBC), …
  - High level: complete analysis classes (RMSD, RMSF, density, dihedrals/Ramachandran, ENCORE, HOLE, *g*(*r*), …)
- Platform agnostic
  - All major MD engine file formats
  - All major OS (Linux, macOS, Windows)
  - All* major CPU architectures



*missing: Apple M1

Core library components & functionality

File readers and writers

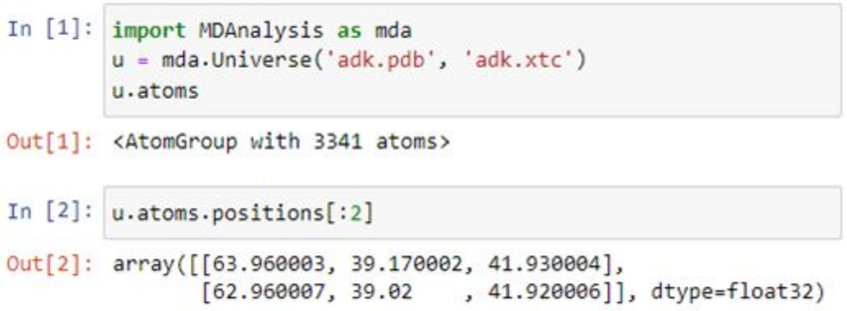
- Support for over 40 file formats
  - **Topologies** (read-only) & **coordinates** (single frame & trajectories)
  - Extensible via *Chemfiles* converter
  - Extensible via own classes (no source code modification necessary)
- MD package independence
  - own internal unit convention (Å, ps ,...)
  - consistent numbering
  - seamless conversion

```
import MDAnalysis as mda
u = mda.Universe("in.prmtop", "in.nc")

u.atoms.write("out.xtc", frames="all")
```

Software	File Type
AMBER	PRMTOP, RST7, TRJ, NETCDF
GROMACS	ITP, TPR, GRO, TRR, XTC
CHARMM	PSF, DCD, CRD
NAMD	DCD, COOR, NAMDBIN
LAMMPS	CONFIG, DATA, DUMP, DCD
DL_POLY	CONFIG, HISTORY
HOOMD	XML, GSD
GAMESS	GMS
DESRES	DMS
Others	XYZ, TXYZ, PDB, PDBQT, PQR, TRZ, MOL2, MMTF, FHIAIMS, H5MD, etc...

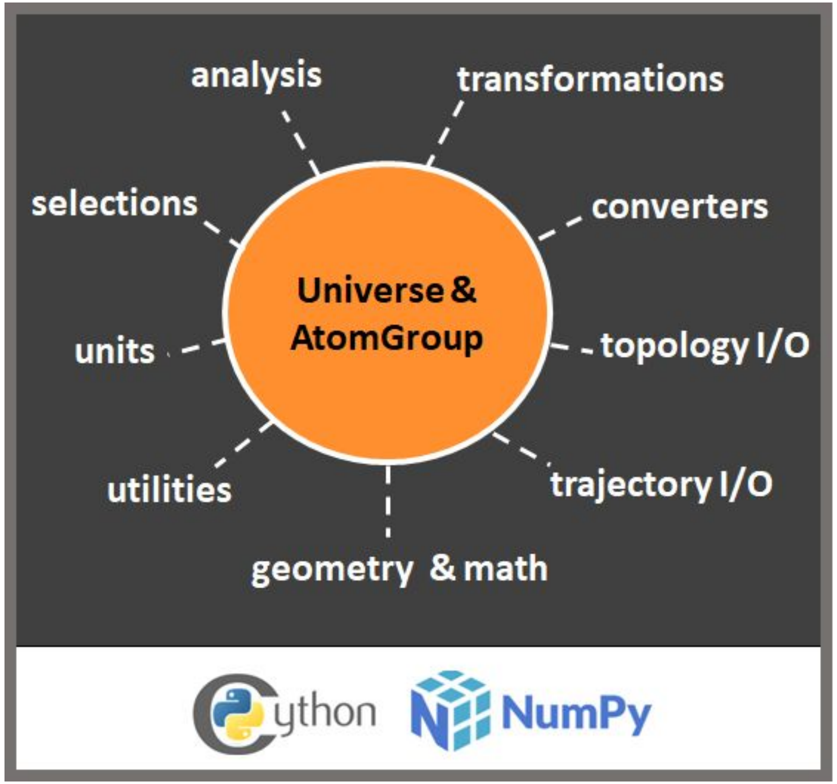
Core MDAnalysis data structures

- **Universe*** class
  - Ties **topology** and **trajectory** together
  - Holds all atom information



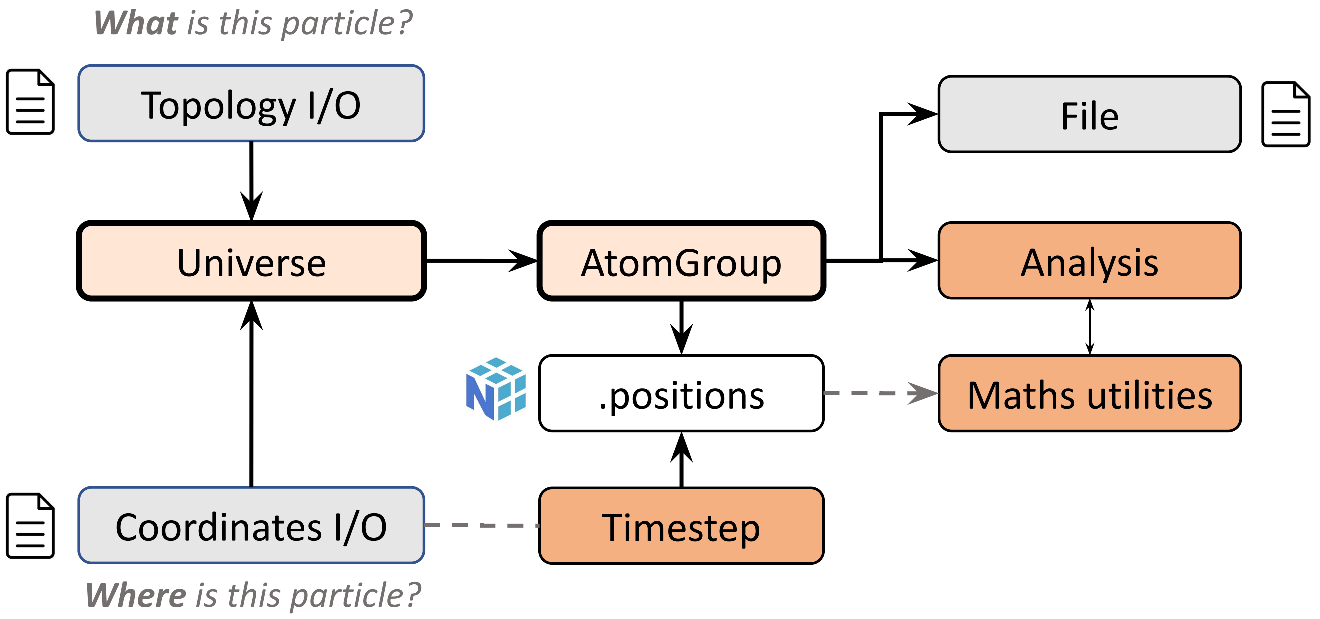
```
In [1]:  import MDAnalysis as mda
         u = mda.Universe('adk.pdb', 'adk.xtc')
         u.atoms
Out[1]:  <AtomGroup with 3341 atoms>

In [2]:  u.atoms.positions[:2]
Out[2]:  array([[63.960003, 39.170002, 41.930004],
                [62.960007, 39.02    , 41.920006]], dtype=float32)
```

- **AtomGroup** class
  - Access to **Atoms** = particles
  - NumPy based (array-like)



* inspired by Konrad Hinsen's MMTK https://github.com/khinsen/mmtk, *J. Comp. Chem.* 21, 79-85 (2000)

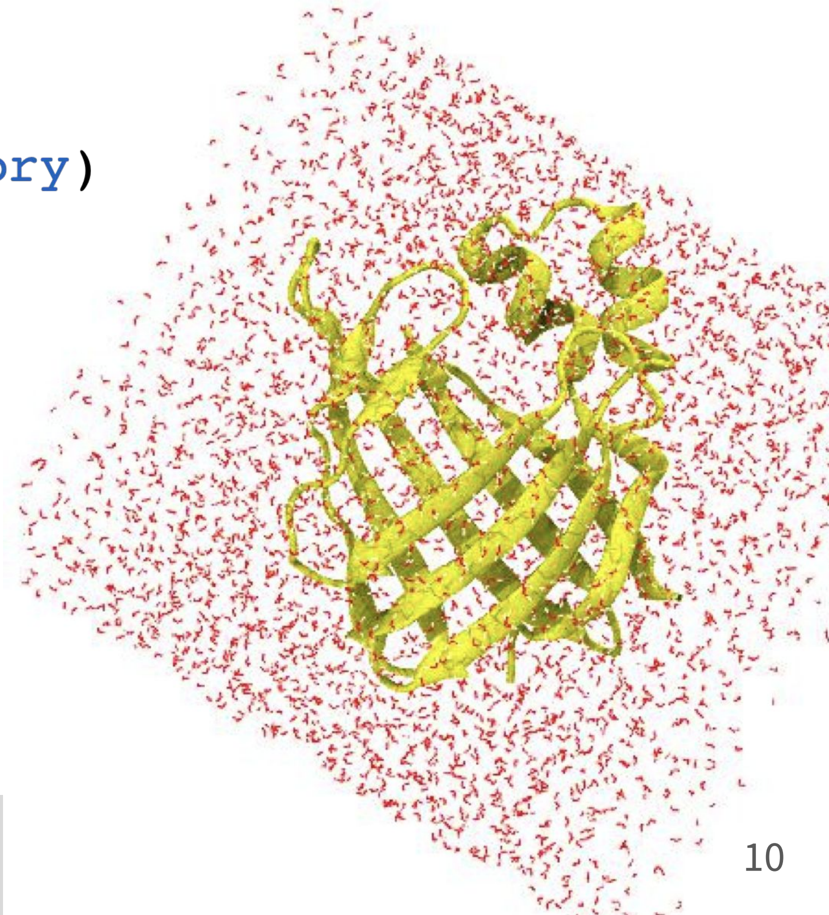
Core layers of MDAnalysis



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>
```



AtomGroup : array_like

```
protein = u.atoms[:2113]
```

slicing

```
protein
```

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

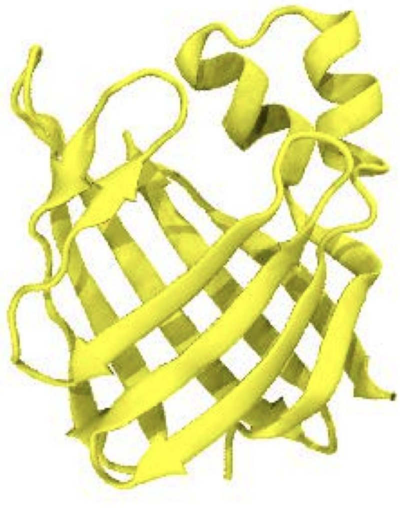
```
print(protein[10:15])
```

```
<AtomGroup [
<Atom 11: C of type 20 of resname ALA, resid 1 and segid IFAB>,
<Atom 12: O of type 70 of resname ALA, resid 1 and segid IFAB>,
<Atom 13: N of type 54 of resname PHE, resid 2 and segid IFAB>,
<Atom 14: HN of type 1 of resname PHE, resid 2 and segid IFAB>,
<Atom 15: CA of type 22 of resname PHE, resid 2 and segid IFAB>]>
```

```
protein[10]
```

indexing

```
<Atom 11: C of type 20 of resname ALA, resid 1 and segid IFAB>
```

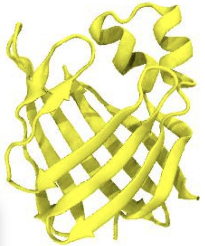


AtomGroup from *selection* and *set operations*

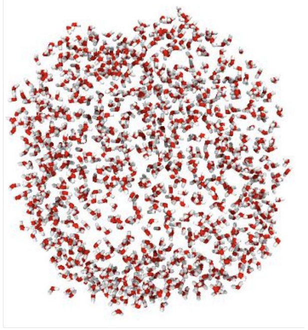
```
protein = u.select_atoms("protein")

protein
<AtomGroup with 2113 atoms>
```

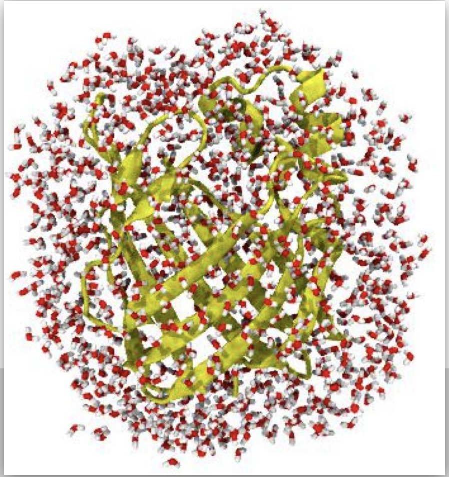
selection

```
solvshell =
  u.select_atoms("resname
    TIP3P and around 5.0
    protein")

solvshell
<AtomGroup with 3868 atoms>
```



+





```
ag = protein + solvshell

ag
<AtomGroup with 5981 atoms>
```

set operations

u.atoms.select_atoms(***selection***)

Basic selection keywords

- **protein** / **backbone** / **nucleic** / **nucleicbackbone**
- **index** 0-123
- **resid** 1-5
- **resname** LYS ARG GLU ASP
- **name** CA
- **type** 22, **type** CT
- **chainID** B
- **smarts** [#7;R]
- …

Geometric

- **around** 3 (resid 157 and name OD*)
- **point** 0 0 0 3.5
- **sphzone** / **sphlayer**
- **cyzone** / **cylayer**

+ dynamic selections:

```
u.atoms.select_atoms("name OW and around 3.0 name OD*", updating=True)
```

Connectivity

- **same** residue **as** (resname SOL and around 3 name NA)
- name H and **bonded** name O

Composition

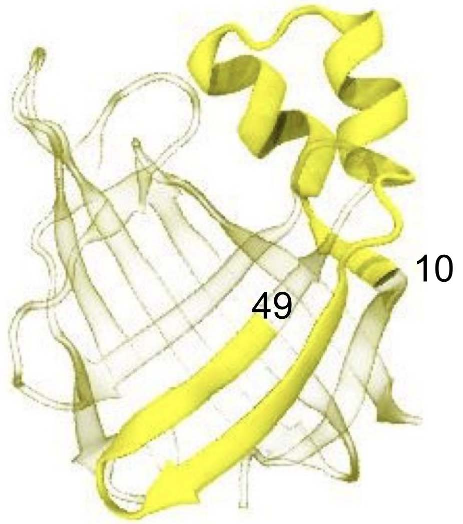
- **Boolean operators**: not, and, or
- **Grouping**: (...)
- **Globbing**: ?, *, [*sequence*], [!*sequence*]

Segment > Residue > Atom Container Hierarchy

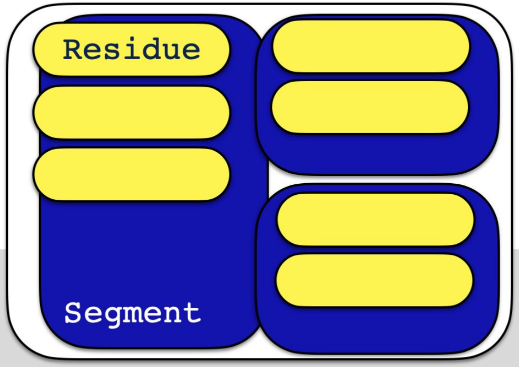
```
protein.residues[10:50]
```

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

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

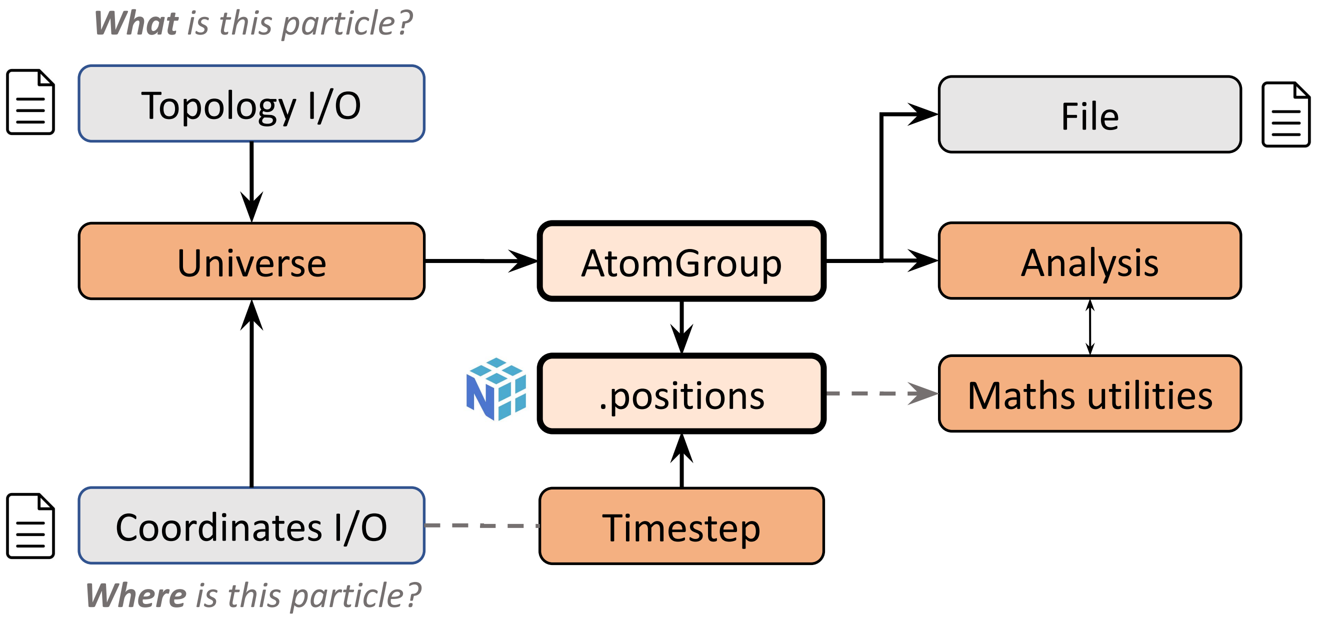


```
print(protein.segments)
<SegmentGroup [<Segment IFAB>]>
```



Universe

Core layers of MDAnalysis



Atom data as 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`

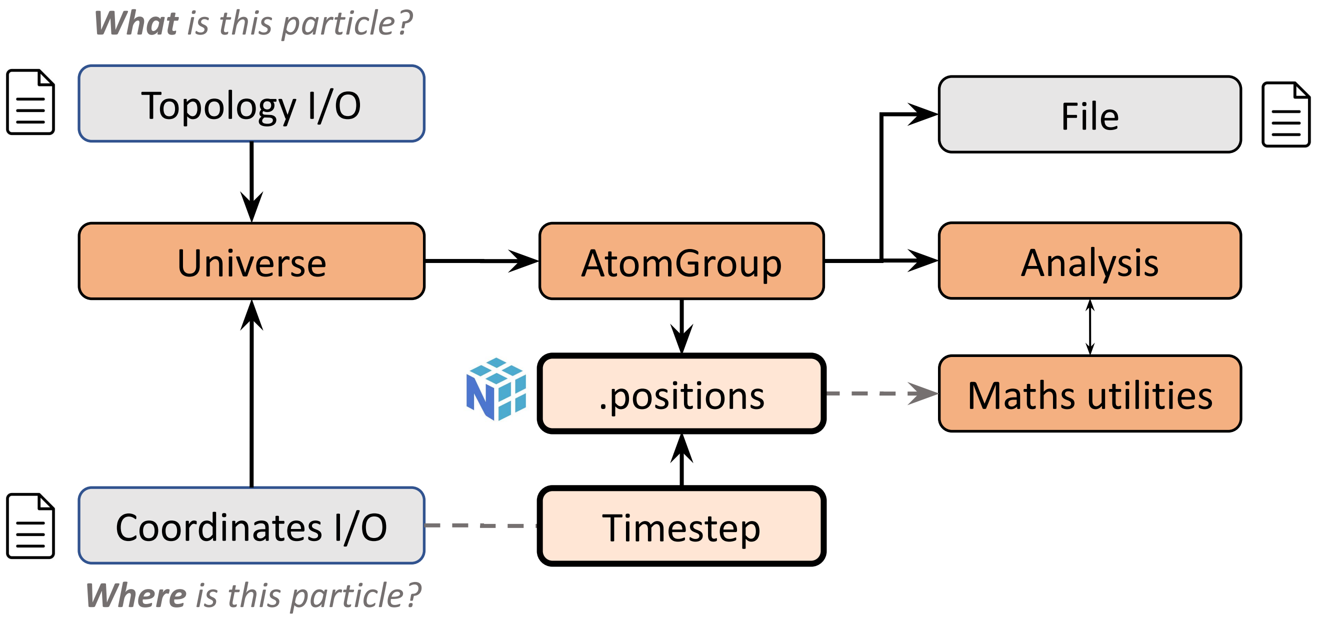
$(\mathbf{r}_1(t), \ldots, \mathbf{r}_N(t))$

```
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

Core layers of MDAnalysis



Basic trajectory analysis pattern

- Single trajectory frame (**r** [,**v** [,**f**]]) at *t* is loaded into memory.
- AtomGroup properties (`ag.positions`, `ag.velocities`, `ag.forces`) ***update***.
- `Universe.trajectory` is ***iterable***:

```
for ts in u.trajectory[start:stop:step]:
    print(ts.frame, ts.time, ts.dimensions)
    analyze(ag.positions)
```

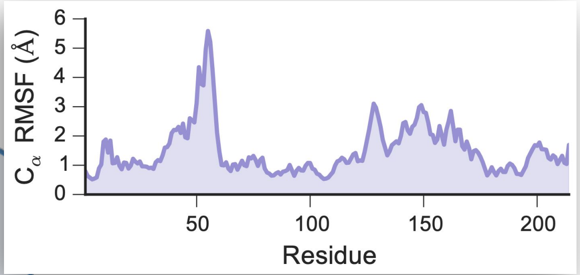
- `Timestep` (`ts`) holds all per-frame data.

- Random access: `u.trajectory[42]`
- Boolean indexing `u.trajectory[[False, True, False, True, …]]`
- Fancy indexing `u.trajectory[[0, 3, 5, 42, 77]]`

Example: Cα RMSF calculation

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

```python
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) * (ca.positions - means)**2
    means[:] = (k*means + ca.positions)/(k+1)
rmsf = np.sqrt(sumsq.sum(axis=1)/(k+1))

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



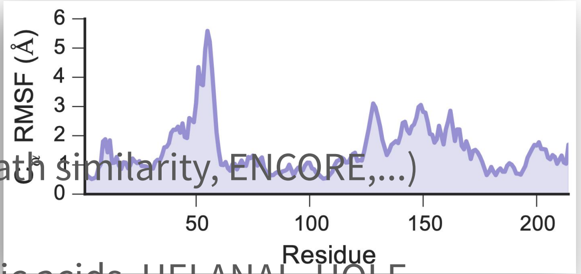
MDAnalysis.**analysis**

Library of commonly used analysis functionality (+ some specialized tools)

- classes
- common API* (based on `AnalysisBase`)
  a. Initialize with `AtomGroup` or `Universe` + parame
  b. Call **`run()`** method.
  c. Process collected data in **`.results`** attribute.

```
from MDAnalysis.analysis.rms import RMSF

ca = u.select_atoms("protein and name CA")
rmsfer = RMSF(ca).run(verbose=True, start=0, step=1)

matplotlib.pyplot.plot(ca.resnums, rmsfer.results.rmsf)
```

- Overview (see https://docs.mdanalysis.org/)
  - Distances and contacts (distances, align, RMSD|F, native contacts, path similarity, ENCORE,...)
  - Hydrogen bonding & water bridges
  - Structure of macromolecules, membranes, liquids (dihedrals, nucleic acids, HELANAL, HOLE, LeafletFinder, RDF, MSD, …)
  - Volumetric (1D and 3D density, water dynamics)
  - Dimensionality reduction (PCA, DiffusionMap)

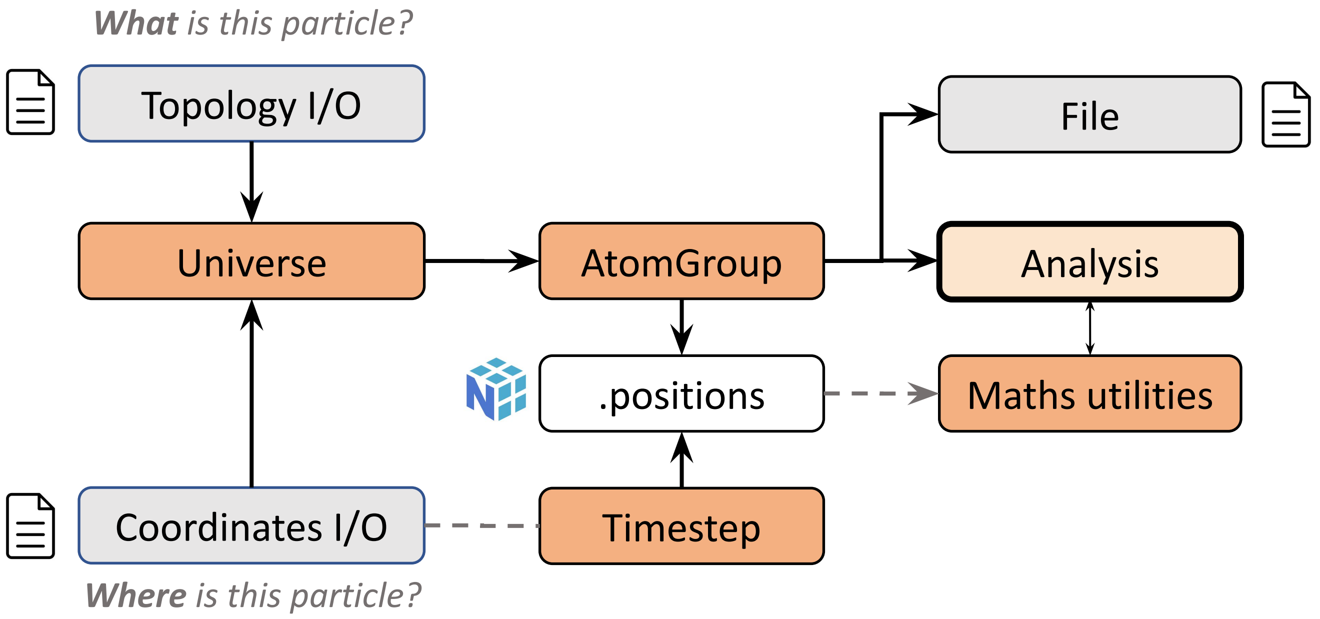


* except some legacy code

… and many more

Extending MDAnalysis

Core layers of MDAnalysis



Extending Analysis:

Ways to create new analyses:

1. Creating an analysis from a function with `AnalysisFromFunction`
2. Creating an analysis class from a function with `analysis_class`
3. Directly subclassing `AnalysisBase`

Extending Analysis

Calculating the radius of gyration

- Start with a function that can be applied per frame

$$R_g = \left(\frac{\sum_i \|\mathbf{r}_i\|^2 m_i}{\sum_i m_i} \right)^{\frac{1}{2}}$$

$$R_{g,x} = \left(\frac{\sum_i \left(r_{i,y}^2 + r_{i,z}^2\right) m_i}{\sum_i m_i} \right)^{\frac{1}{2}}$$

```
from MDAnalysis.tests.datafiles import PSF, DCD

u = mda.Universe(PSF, DCD)
protein = u.select_atoms("protein")
total_mass = protein.masses.sum()
```

```
def radgyr(atomgroup, masses, total_mass):
    # coordinates change for each frame
    coordinates = atomgroup.positions
    center_of_mass = atomgroup.center_of_mass()

    # get squared distance from center
    ri_sq = (coordinates-center_of_mass)**2
    # sum the unweighted positions
    sq = np.sum(ri_sq, axis=1)
    sq_x = np.sum(ri_sq[:,[1,2]], axis=1) # sum over y and z
    sq_y = np.sum(ri_sq[:,[0,2]], axis=1) # sum over x and z
    sq_z = np.sum(ri_sq[:,[0,1]], axis=1) # sum over x and y

    # make into array
    sq_rs = np.array([sq, sq_x, sq_y, sq_z])

    # weight positions
    rog_sq = np.sum(masses*sq_rs, axis=1)/total_mass
    # square root and return
    return np.sqrt(rog_sq)
```

Extending Analysis

1. Creating an analysis from a function with `AnalysisFromFunction`

```
In [1]:  from MDAnalysis.analysis.base import AnalysisFromFunction
         rog = AnalysisFromFunction(radgyr, u.trajectory,
                                    protein, protein.masses,
                                    total_mass)
         rog.run(start=1, stop=5)

Out[1]:  <MDAnalysis.analysis.base.AnalysisFromFunction at 0x7fc009e4fe50>
```

```
AnalysisFromFunction(function,
                     trajectory=None,
                     *args, **kwargs)
```

```
radgyr(atomgroup, masses, total_mass)
```

```
In [2]:  rog.results.timeseries

Out[2]:  array([[16.66901837, 12.6796255 , 13.74934255, 14.3490426 ],
                ...,
                [19.59157513, 13.44275041, 16.53792589, 17.7044938 ]])
```

Extending Analysis

2. Creating an analysis class from a function with `analysis_class`

```
In [1]:  from MDAnalysis.analysis.base import analysis_class
         RadiusOfGyration = analysis_class(radgyr)
         rog = RadiusOfGyration(u.trajectory, protein, protein.masses,
                                total_mass)
         rog.run(start=1, stop=5)

Out[1]:  <MDAnalysis.analysis.base.analysis_class.<locals>.WrapperClass at
         0x7fb6bec3b610>

In [2]:  rog.results.timeseries

Out[2]:  array([[16.66901837, 12.6796255 , 13.74934255, 14.3490426 ],
                ...,
                [19.59157513, 13.44275041, 16.53792589, 17.7044938 ]])
```

```
analysis_class(function)

radgyr(atomgroup, masses, total_mass)
```

Extending Analysis

3. Creating a new class by subclassing `AnalysisBase`

- How most analyses in MDAnalysis are created
- Includes a lot of nice things like progress bars

- ***`.__init__(self, trajectory, verbose, **kwargs):`***
  Class set-up
- ***`.run(self, start, stop, step, verbose):`***
  Calls the below functions and sets up frames to run on
  - ***`._prepare(self):`***
    Code to prepare for analysis. Usually sets up result containers
  - ***`._single_frame(self):`***
    Code that runs for each frame of analysis
  - ***`._conclude(self):`***
    Any code that finishes up the analysis, e.g. calculating means

Extending Analysis

3. Creating a new class by subclassing AnalysisBase

```python
class RadiusOfGyration(AnalysisBase):

    def __init__(self, atomgroup, verbose=True):
        """Set up the initial analysis parameters."""

        # must first run AnalysisBase.__init__
        trajectory = atomgroup.universe.trajectory
        super().__init__(trajectory, verbose=verbose)

        # set atomgroup as a property for access in other methods
        self.atomgroup = atomgroup
        self.masses = self.atomgroup.masses
        self.total_mass = np.sum(self.masses)
```

```
AnalysisBase
.__init__
.run
   ._prepare
   ._single_frame
   ._conclude
```

Extending Analysis

3. Creating a new class by subclassing AnalysisBase

```
class RadiusOfGyration(AnalysisBase):

    def __init__(self, atomgroup, verbose=True):
        ...

    def _prepare(self):
        """
        Create array of zeroes as a placeholder for results.
        Must go here instead of __init__ because it depends on
        the number of frames specified in .run()
        """
        self.results.radius = np.zeros((self.n_frames, 4))
```

AnalysisBase
.__init__
.run
  ._prepare
  ._single_frame
  ._conclude

Extending Analysis

3. Creating a new class by subclassing AnalysisBase

```
class RadiusOfGyration(AnalysisBase):

    def __init__(self, atomgroup, verbose=True):
        ...

    def _prepare(self):
        ...

    def _single_frame(self):
        """ This function is called for every frame chosen in run()."""
        rogs = radgyr(self.atomgroup, self.masses, self.total_mass)
        # save it into self.results
        self.results.radius[self._frame_index] = rogs
```

AnalysisBase
.__init__
.run
  ._prepare
  ._single_frame
  ._conclude

Extending Analysis

3. Creating a new class by subclassing AnalysisBase

```
class RadiusOfGyration(AnalysisBase):

    def __init__(self, atomgroup, verbose=True):
        ...

    def _prepare(self):
        ...

    def _single_frame(self):
        ...

    def _conclude(self):
        """Finish up by calculating an average"""
        self.average = np.mean(self.results.radius, axis=0)
```

```
AnalysisBase
.__init__
.run
   ._prepare
   ._single_frame
   ._conclude
```

Extending Analysis

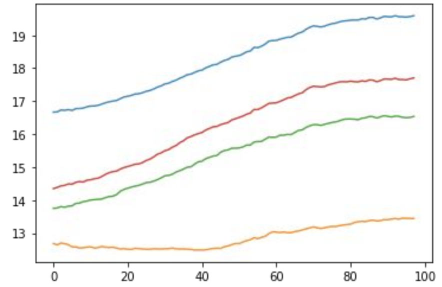
3. Creating a new class by subclassing AnalysisBase

AnalysisBase
.__init__
.run
  ._prepare
  ._single_frame
  ._conclude

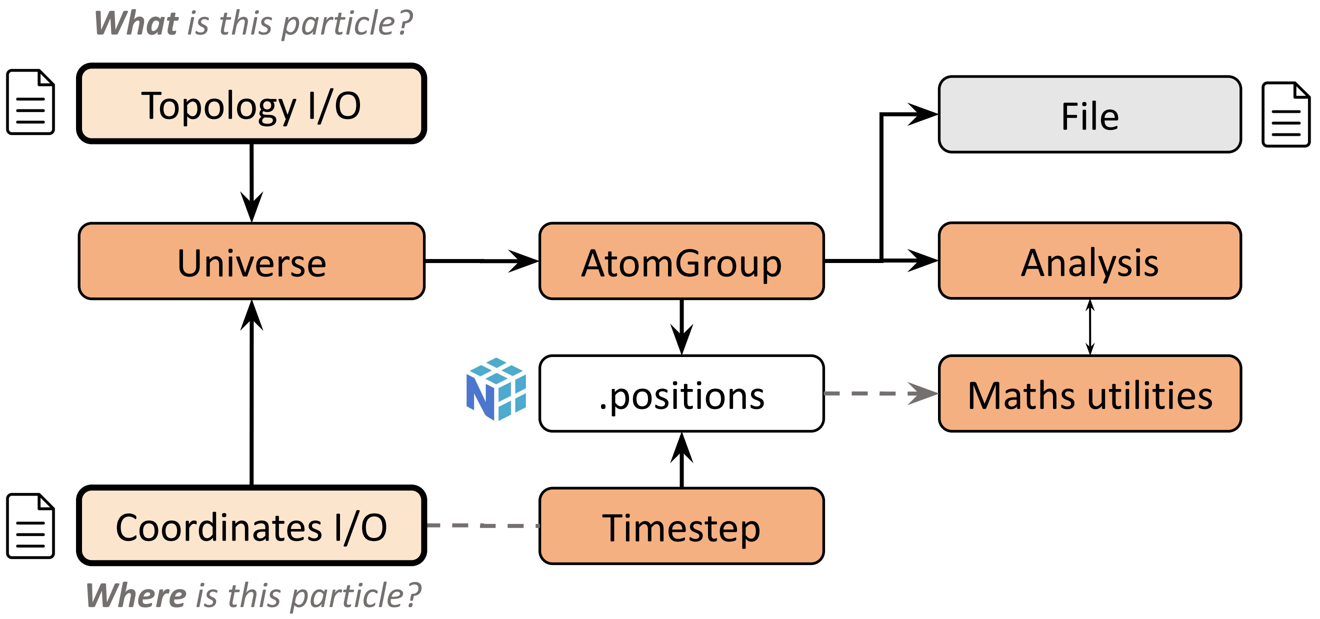
In [1]:
```
rog = RadiusOfGyration(protein).run()
rog.average
```

Out[1]:
```
array([18.26549552, 12.85342131, 15.37359575, 16.29185734])
```

In [2]:
```
import matplotlib.pyplot as plt
plt.plot(rog.results.radius);
```



Core layers of MDAnalysis



Extending I/O

Creating new Topology and Coordinate readers

- Subclass `TopologyReaderBase` for topology
- Subclass `ReaderBase` or `SingleFrameReaderBase` for coordinates
- Subclass `WriterBase` to write out files
- Metaclass magic makes them immediately available through the standard MDAnalysis interface

Extending I/O

Creating new Topology and Coordinate readers

In [1]:
```
from MDAnalysis.coordinates.base import ReaderBase

class NumpyArrayReader(ReaderBase):
    format = "NPY"

    def __init__(self, filename, **kwargs):
        super().__init__(filename, **kwargs)

        self._coords = np.load(filename)
        self.n_frames, self.n_atoms = self._coords.shape[:2]
        self.ts = self._Timestep(self.n_atoms, **self._ts_kwargs)
        self._read_next_timestep()

    def _read_next_timestep(self, ts=None):
        if ts is None:
            ts = self.ts

        ts.positions = self._all_coordinates[ts.frame + 1]
        ts.frame += 1
        return ts
```

Extending I/O

Creating new Topology and Coordinate readers

```
In [1]: from MDAnalysis.coordinates.base import ReaderBase

        class NumpyArrayReader(ReaderBase):

            def __init__(self, filename, **kwargs):
                super().__init__(filename, **kwargs)

                self._coords = np.load(filename)
                self.n_frames, self.n_atoms = self._coords.shape[:2]
                self.ts = self._Timestep(self.n_atoms, **self._ts_kwargs)
                self._read_next_timestep()

            def _read_next_timestep(self, ts=None):
                if ts is None:
                    ts = self.ts

                ts.positions = self._all_coordinates[ts.frame + 1]
                ts.frame += 1
                return ts
```

```
format = "NPY"
```

Extending I/O

Creating new Topology and Coordinate readers

```
In [1]: from MDAnalysis.coordinates.base import ReaderBase

        class NumpyArrayReader(Reade
```

```
format = "NPY"
```

```
            super().__init__(fil

            self._coords = np.lo
            self.n_frames, self.
            self.ts = self._Time
            self._read_next_time

        def _read_next_timestep(
            if ts is None:
                ts = self.ts

            ts.positions = self._
            ts.frame += 1
            return ts
```

```
In [2]: from MDAnalysis.tests.datafiles import PDB
        import numpy as np

        arr = np.random.rand(5, 47681, 3)
        np.save("my_coordinates.npy", arr)
        u = mda.Universe(PDB, "my_coordinates.npy")
        len(u.trajectory)

Out[2]: 5
```

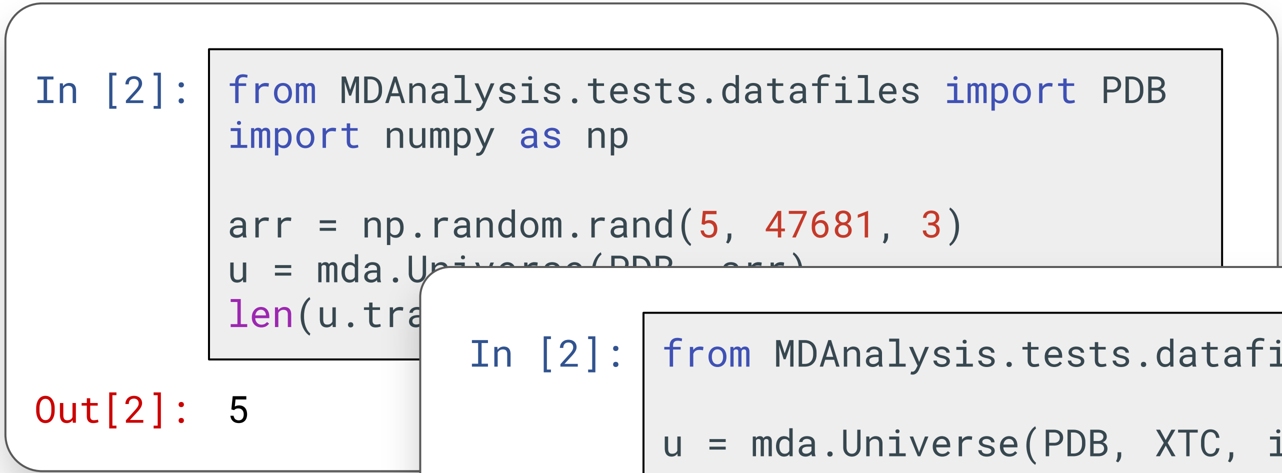
Extending I/O

Using the MemoryReader

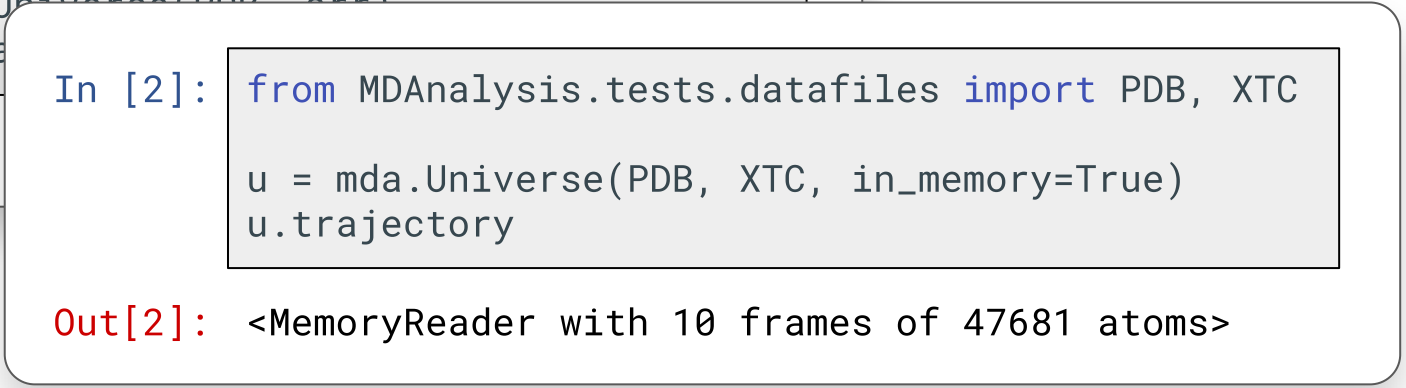
```
In [2]: from MDAnalysis.tests.datafiles import PDB
        import numpy as np

        arr = np.random.rand(5, 47681, 3)
        u = mda.Universe(PDB, arr)
        len(u.trajectory)

Out[2]: 5
```

Extending I/O

Using the MemoryReader



```
In [2]:  from MDAnalysis.tests.datafiles import PDB
         import numpy as np

         arr = np.random.rand(5, 47681, 3)
         u = mda.U
         len(u.tra

Out[2]:  5
```



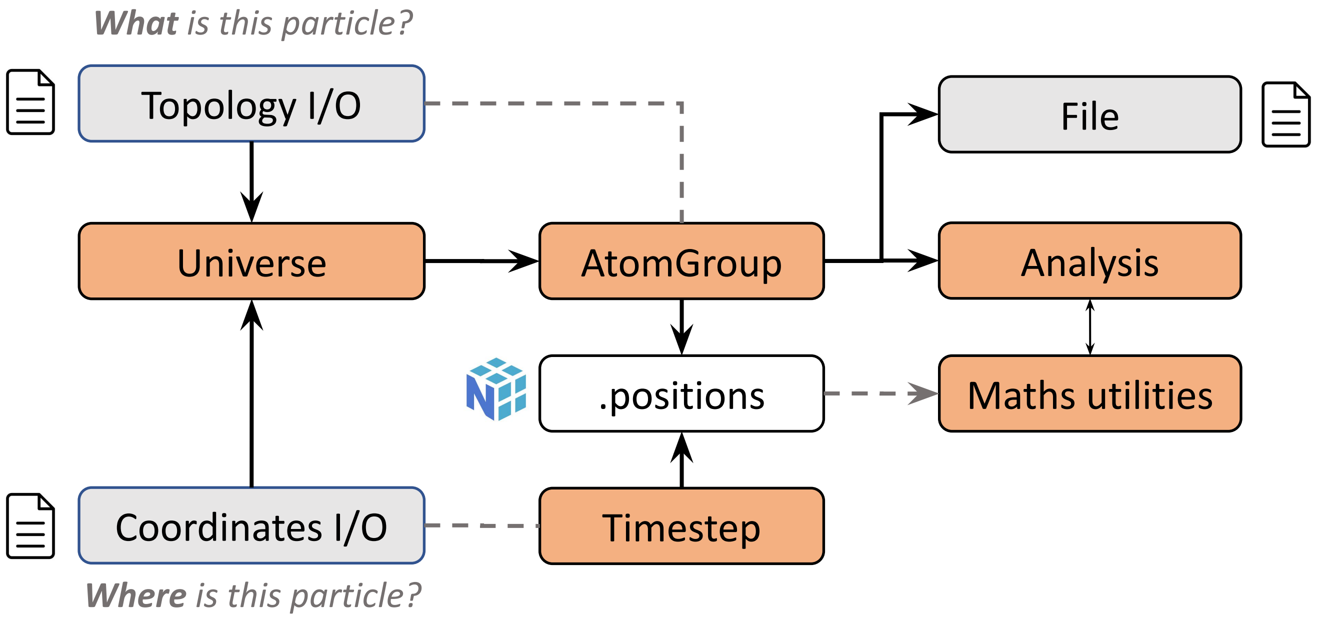
```
In [2]:  from MDAnalysis.tests.datafiles import PDB, XTC

         u = mda.Universe(PDB, XTC, in_memory=True)
         u.trajectory

Out[2]:  <MemoryReader with 10 frames of 47681 atoms>
```

Extending I/O

Using the MemoryReader

- Faster as data is in memory
- Very flexible
- Can be used to construct Universes from scratch (https://userguide.mdanalysis.org/stable/examples/constructing_universe)
- Can be used to work with all coordinates of all frames at once
    - `u.trajectory.coordinate_array` → 3D NumPy array

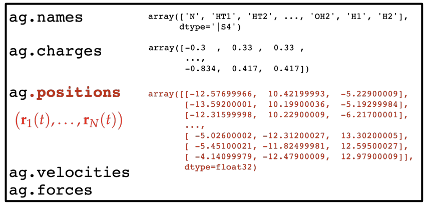
Core layers of MDAnalysis



Tagging atoms with topology attributes

The `TopologyAttr` system

- Label atoms, residues, segments
- Static tags that don’t change over a trajectory
- Often read from file but can also be added and set by user



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

Static topology attributes

```
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])
```

Dynamic trajectory data

```
ag.positions    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
```

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

Tagging atoms with topology attributes

```
In [1]:  from MDAnalysis.tests.datafiles import PDB
         u = mda.Universe(PDB)
         u.atoms.elements

Out[1]:  -------------------------------------------------------

         NoDataError: This Universe does not contain element information


In [2]:  u.add_TopologyAttr("elements")
         u.atoms.elements

Out[2]:  array(['', '', '', ..., '', '', ''], dtype=object)

In [3]:  u.atoms.elements = "C"
         u.atoms.elements

Out[3]:  array(['C', 'C', 'C', ..., 'C', 'C', 'C'], dtype=object)
```

Tagging atoms with topology attributes

```
In [4]: u.atoms.elements = u.atoms.resnames
        u.atoms.elements

Out[4]: array(['MET', 'MET', 'MET', ..., 'NA+', 'NA+', 'NA+'], dtype=object)
```

```
In [5]: u.residues[0].atoms.elements = "Z"
        u.atoms.elements

Out[5]: array(['Z', 'Z', 'Z', ..., 'NA+', 'NA+', 'NA+'], dtype=object)
```

```
In [6]: u.atoms[0].element = "First"
        u.atoms.elements

Out[6]: array(['First', 'Z', 'Z', ..., 'NA+', 'NA+', 'NA+'], dtype=object)
```

Tagging atoms with topology attributes

Canonical attributes (assigned by MDAnalysis and immutable)

`indices, resindices, segindices`

Common attributes (read or guessed from every format)

`ids, masses, resids, segids, types`

Format-specific attributes

```
altLocs, chainIDs, charges, elements, icodes, models, molnums,
moltypes, names, occupancies, radii, record_types, resnames,
tempfactors, type_indices
```

Connectivity attributes

`bonds, angles, dihedrals, impropers`

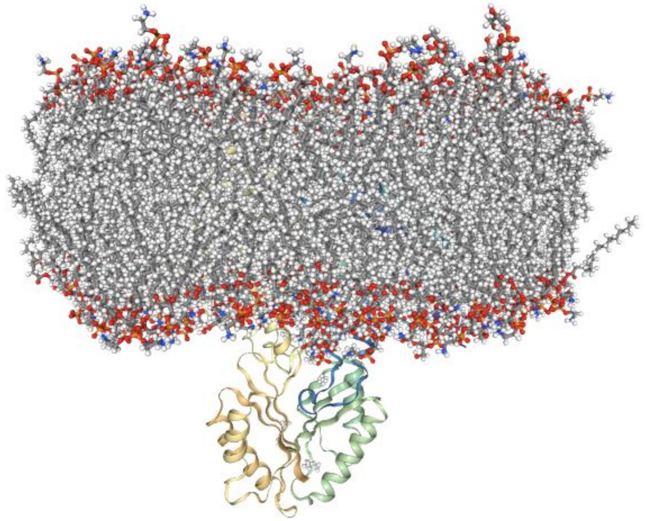
https://userguide.mdanalysis.org/stable/topology_system

Tagging atoms with topology attributes

Creating new TopologyAttrs

- Subclass `AtomAttr`, `ResidueAttr` or `SegmentAttr`
- e.g., labelling lipids in a membrane by leaflet

In [1]:
```
from MDAnalysis.tests.datafiles import GRO_MEMPROT

u = mda.Universe(GRO_MEMPROT)
```



Tagging atoms with topology attributes

Creating new TopologyAttrs

```
In [2]: from MDAnalysis.core.topologyattrs import _ResidueStringAttr

        class LipidClass(_ResidueStringAttr):
            attrname = "leaflets"
            singular = "leaflet"

            @staticmethod
            def _gen_initial_values(n_atoms, n_residues, n_segments):
                return np.array(["Other"] * n_residues, dtype=object)

        u.add_TopologyAttr("leaflets")
        u.residues.leaflets
```

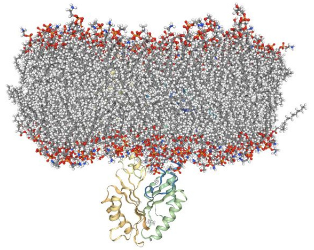
```
Out[2]: array(['Other', 'Other', 'Other', 'Other', 'Other', 'Other', ...,
               'Other', 'Other', 'Other', 'Other', 'Other', 'Other'], dtype=object)
```

Tagging atoms with topology attributes

Selection magic

- These labels can be used to select atoms with selection language
- New `TopologyAttr` classes are automatically picked up by the selection parser

```
In [3]: from MDAnalysis.analysis import leaflet

        finder = leaflet.LeafletFinder(u, select="name P", pbc=True)
        atomgroup_1 = finder.groups(0)
        atomgroup_1.residues.leaflets = "upper"
        atomgroup_2 = finder.groups(1)
        atomgroup_2.residues.leaflets = "lower"

        upper_ring = u.select_atoms("leaflet upper and around 5 protein")

        upper_ring

Out[3]: <AtomGroup with 1341 atoms>
```

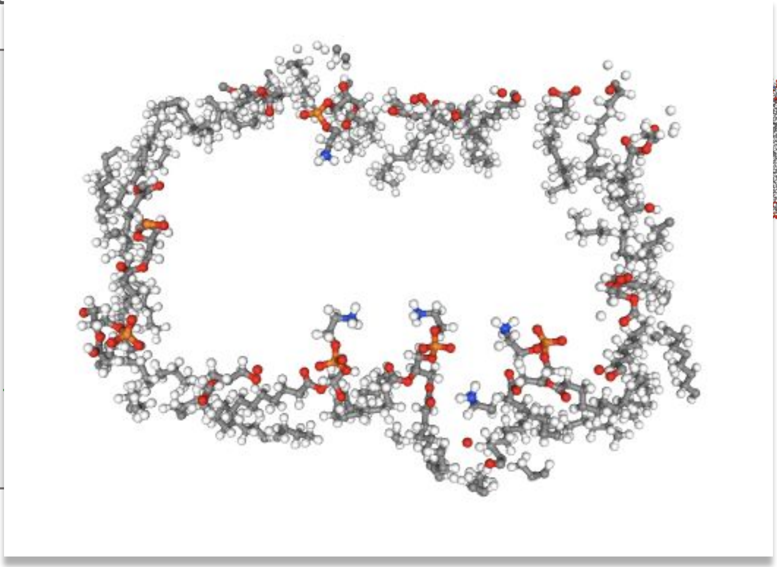


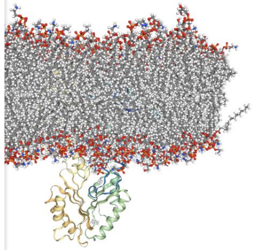
Tagging atoms with topology attributes

Selection magic

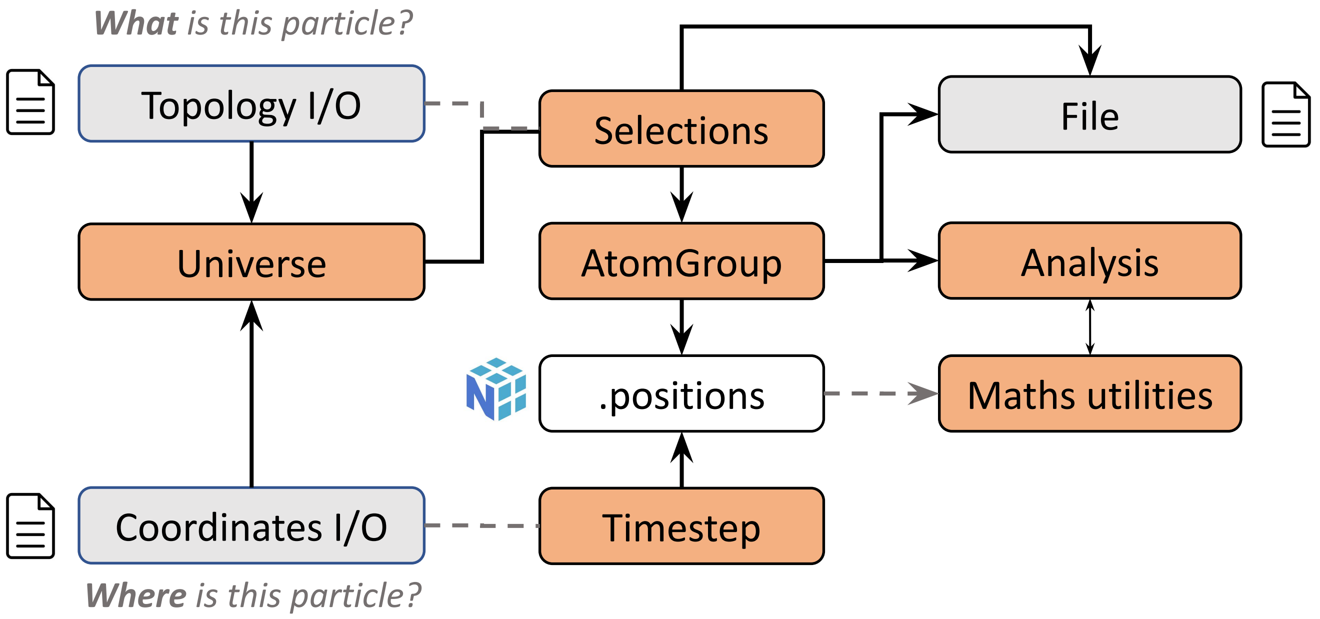
- These labels can be used to select atoms with selection language
- New `TopologyAttr` classes are automatically picked up by the selection parser

```
In [3]: from MDAnalysis.analysis import

        finder = leaflet.LeafletFinder(u
        atomgroup_1 = finder.groups(0)
        atomgroup_1.residues.leaflets =
        atomgroup_2 = finder.groups(1)
        atomgroup_2.residues.leaflets =

        upper_ring = u.select_atoms("lea

        upper_ring
Out[3]: <AtomGroup with 1341 atoms>
```





Hidden layers of MDAnalysis



Selection exporters

Writing selections out to file

- Supported formats:
    - CHARMM
    - GROMACS
    - VMD
    - PyMol
    - JMol

Python

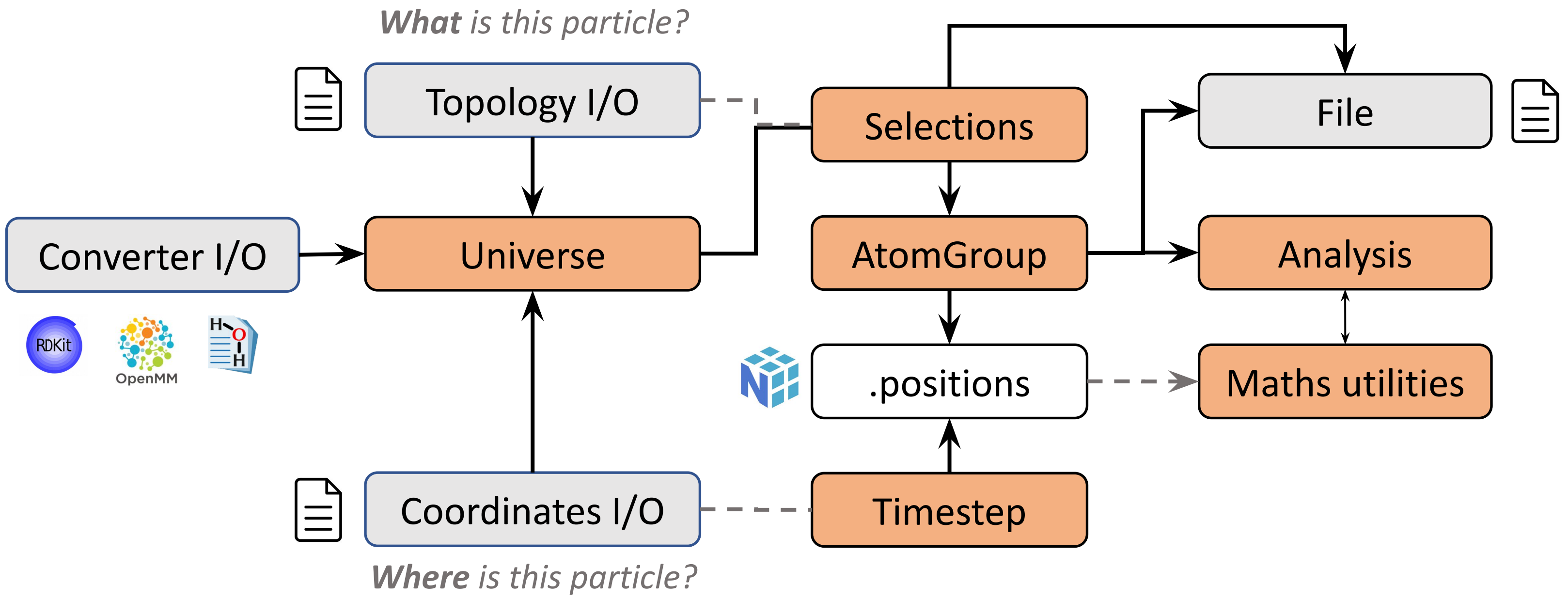
```
upper_ring.write("upper.vmd", name="upper_ring")
```

VMD

```
source upper.vmd
set sel [atomselect top upper_ring]
```

```
with mda.selections.gromacs.SelectionWriter('leaflets.ndx', mode='w') as ndx:
    ndx.write(atomgroup_1, name='upper')
    ndx.write(atomgroup_2, name='lower')
```

Hidden layers of MDAnalysis



Converters

Need for interoperability

- Lots of great tools
  - Limit to one toolset problematic
- MolSSI 2019 workshop *Molecular Dynamics Software Interoperability*
  - https://molssi.org/2019/07/29/molssi-workshop-molecular-dynamics-software-interoperability/
- Implement seamless conversion layers between MDAnalysis and other packages
  - RDKit, ParmEd, OpenMM, Chemfiles
  - More coming soon!

Converters

Chemfiles https://chemfiles.org/

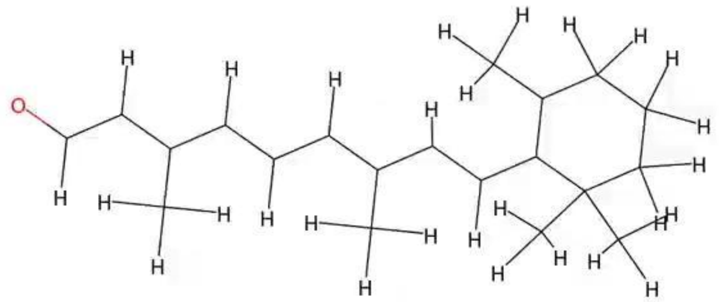
- Read and write many other formats
- Extension of topology and coordinate I/O system

```
In [1]: from MDAnalysis.tests.datafiles import TPR, TRR
        u = mda.Universe(TPR, TRR, format="CHEMFILES")
        u.trajectory

Out[1]: <ChemfilesReader mdanalysis/testsuite/MDAnalysisTests/
        data/adk_oplsaa.trr with 10 frames of 47681 atoms>
```

Converters

RDKit

- Convert to and from RDKit
- Guessing the correct chemistry (bond orders, valence, etc) is a non-trivial challenge!



+

RDKit

Cédric Bouysset

Converters

RDKit

```
In [1]: ua = mda.Universe("hydroxystearic_acid.pdb")
```

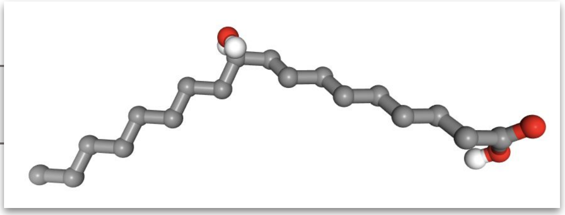
+

Cédric Bouysset

Converters

RDKit

```
In [1]: ua = mda.Universe("hydroxystearic_acid.pdb")
```



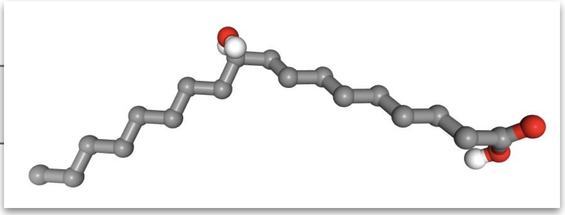
+

Cédric Bouysset

Converters

RDKit

```
In [1]: ua = mda.Universe("hydroxystearic_acid.pdb")
```



```
In [2]: ua.add_TopologyAttr("elements", u.atoms.types)
        rdmol = u.atoms.convert_to("RDKIT", NoImplicit=False)
```

+

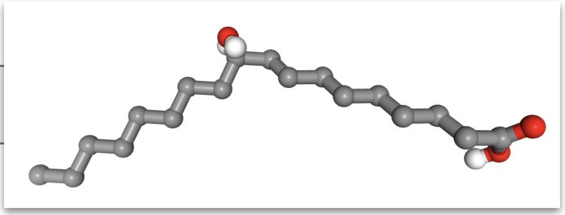
Cédric Bouysset

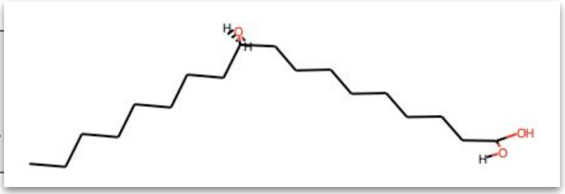
Converters

RDKit

```
In [1]: ua = mda.Universe("hydroxystearic_acid.pdb")
```

```
In [2]: ua.add_TopologyAttr("elements", u.atoms.types
        rdmol = u.atoms.convert_to("RDKIT", NoImplici
```





+

Cédric Bouysset

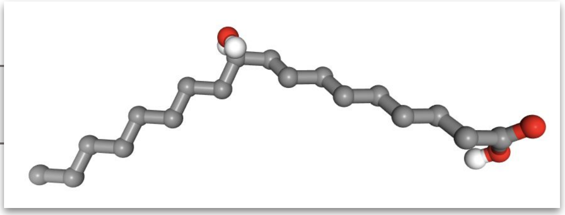
Converters

+

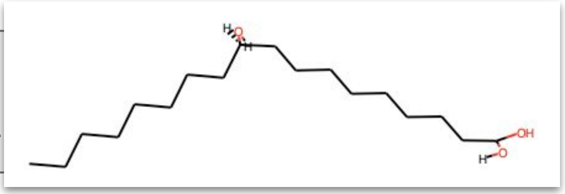
Cédric Bouysset

RDKit

In [1]:
```
ua = mda.Universe("hydroxystearic_acid.pdb")
```



In [2]:
```
ua.add_TopologyAttr("elements", u.atoms.types
rdmol = u.atoms.convert_to("RDKIT", NoImplici
```



In [3]:
```
rdmol = Chem.AddHs(rdmol, addCoords=True, addResidueInfo=True)
aa = mda.Universe(rdmol)
```

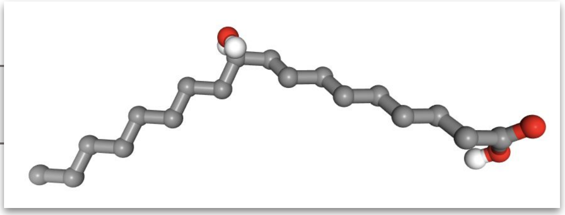
Converters

+

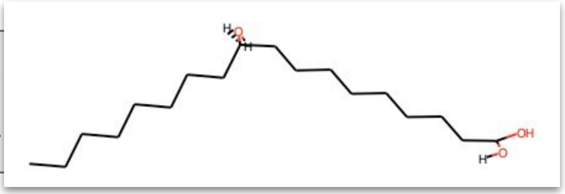
Cédric Bouysset

RDKit

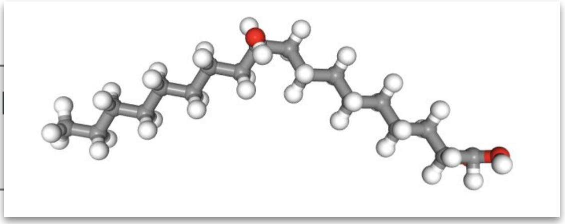
In [1]:
```
ua = mda.Universe("hydroxystearic_acid.pdb")
```



In [2]:
```
ua.add_TopologyAttr("elements", u.atoms.types
rdmol = u.atoms.convert_to("RDKIT", NoImplici
```



In [3]:
```
rdmol = Chem.AddHs(rdmol, addCoords=True, add
aa = mda.Universe(rdmol)
```



Converters

OpenMM

- Convert from OpenMM objects:
  - Topology
  - PDBFile
  - PDBxFile
  - Modeller
  - Simulation

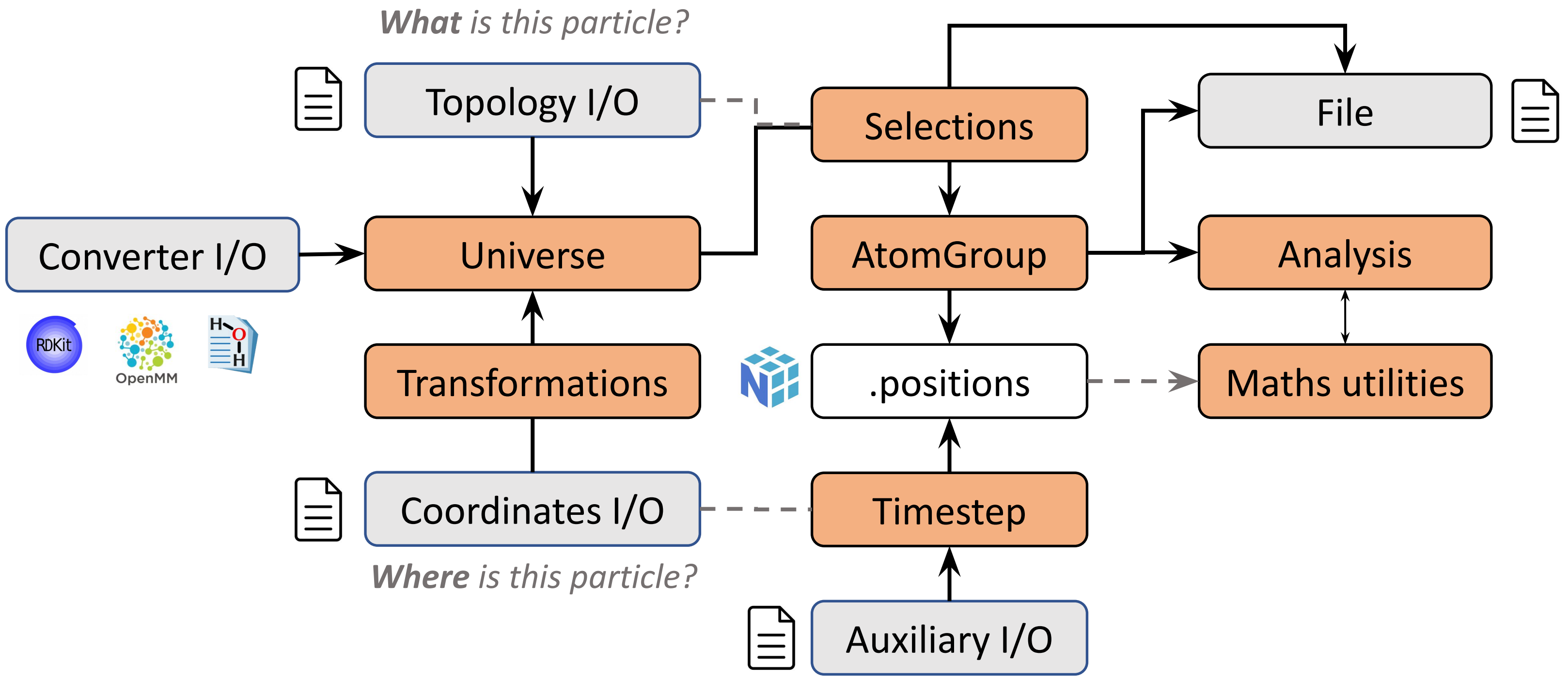
```
from simtk.openmm import app
structure = app.PDBxFile('4lzt.cif')
u = mda.Universe(structure)
```

ParmEd

- Convert to and from ParmEd Structures

```
import parmed
from MDAnalysis.tests.datafiles import PRM
pmd = parmed.load_file(PRM)
u = mda.Universe(pmd)
pmd2 = u.atoms[:10].convert_to("PARMED")
```

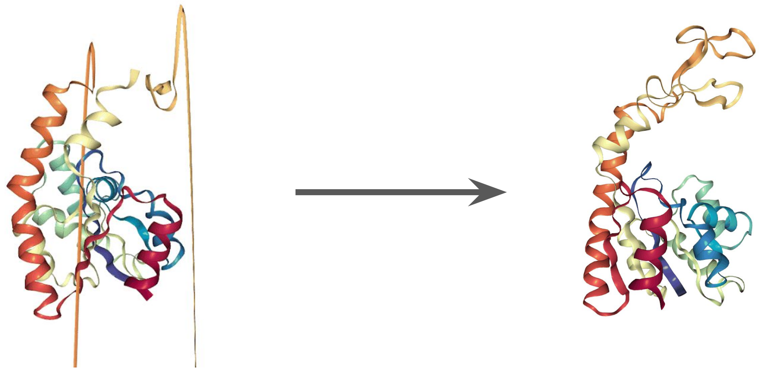
Hidden layers of MDAnalysis



Coordinate transformations

On-the-fly transformations

- Direct trajectory manipulations
  - PBC fixing, centering, trajectory alignment, etc…
  - No need for file duplication
  - Can be layered together
- Still improving
  - Substantial performance costs
  - Issues with multi-chain proteins



```python
from MDAnalysis import transformations as tform
transform = [tform.unwrap(protein),
             tform.center_in_box(protein, wrap=True),
             tform.wrap(not_protein),
             tform.fit_rot_trans(c_alphas, c_alphas, weights="mass")]
u.trajectory.add_transformations(*transform)
```

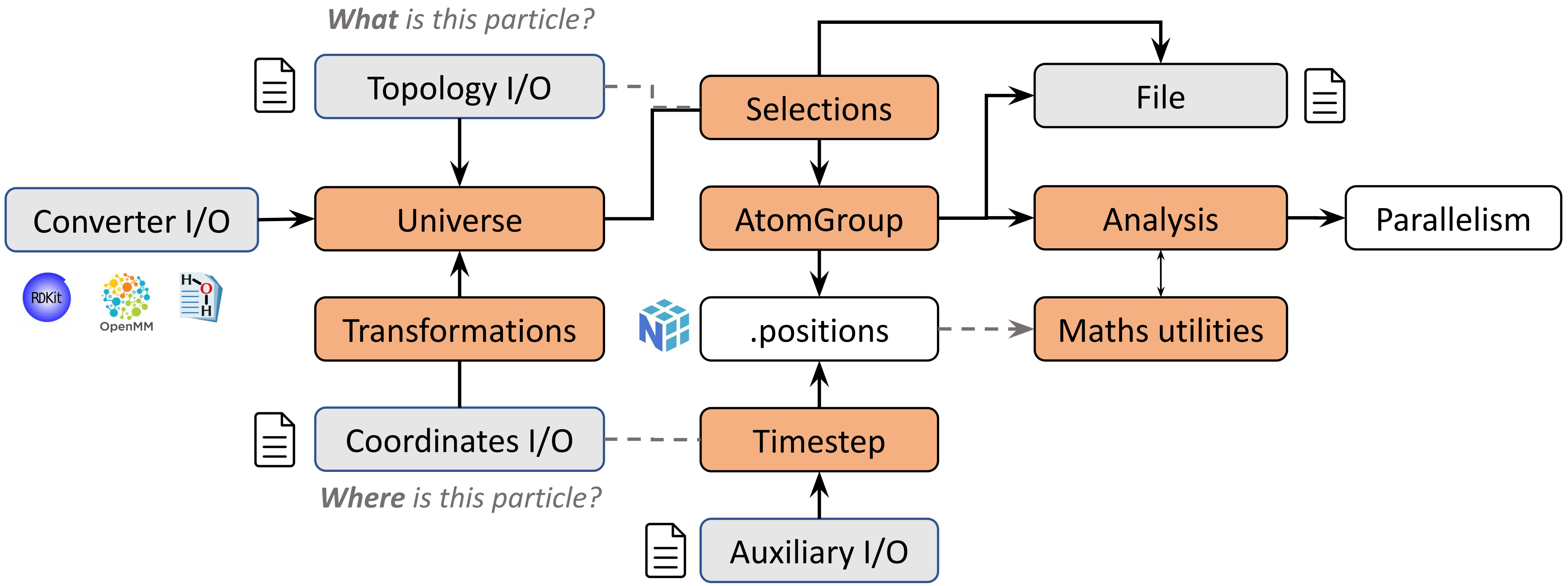
Auxiliary data

- Load in extra information per time-step
- Automatically iterate over frames where values are assigned
- Supported formats: XVG
- Subclass `MDAnalysis.auxiliary.base.AuxReader` for more

In [1]:

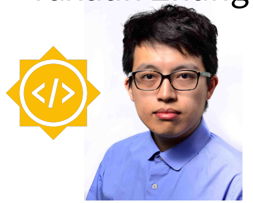
```
import MDAnalysis as mda
from MDAnalysisTests.datafiles import PDB_sub_sol, XTC_sub_sol, XVG_BZ2

u = mda.Universe(PDB_sub_sol, XTC_sub_sol)
u.trajectory.add_auxiliary('forces', XVG_BZ2)

# iterate over frames with force values assigned
for time_step in u.trajectory.iter_as_aux("forces"):
    print(time_step.aux.forces)
```

Hidden layers of MDAnalysis



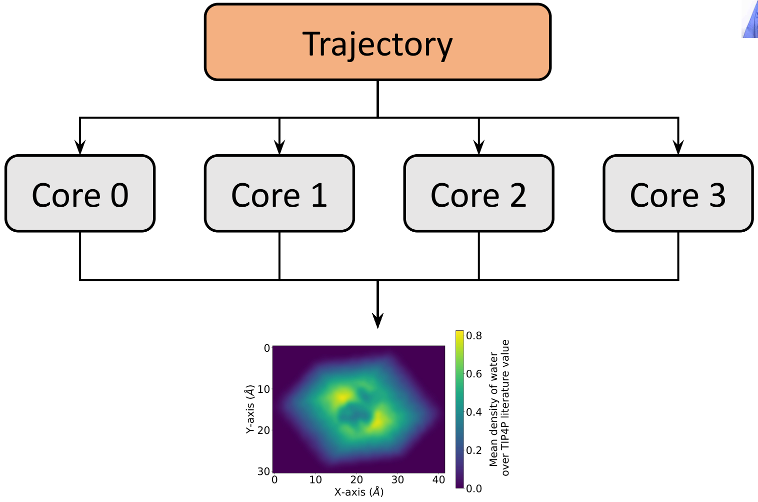
Parallelism

Yuxuan Zhang



Most analyses are embarrassingly parallelizable

- Datasets increasing in size:
  - Number of atoms
  - Number of frames
- Analyses increasing in complexity
- New: serialisation of Universes
  - Leverage common Python parallelism tools
  - Multiprocessing, Dask, etc...

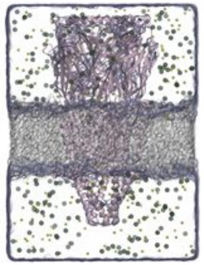


Parallelism

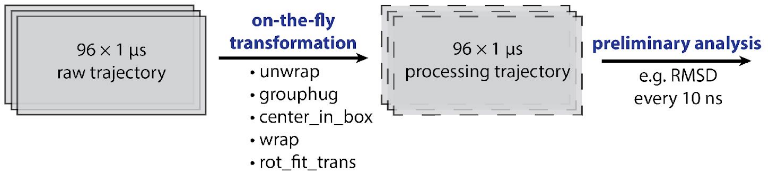
Yuxuan Zhang

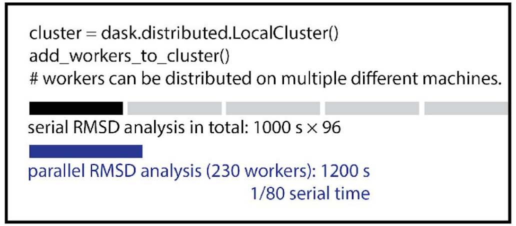
Most analyses are embarrassingly parallelizable

- Datasets increasing in size:
  - Number of atoms
  - Number of frames
- Analyses increasing in complexity
- New: serialisation of Universes
  - Leverage common Python parallelism tools
  - Multiprocessing, Dask, etc...



250,000 atoms



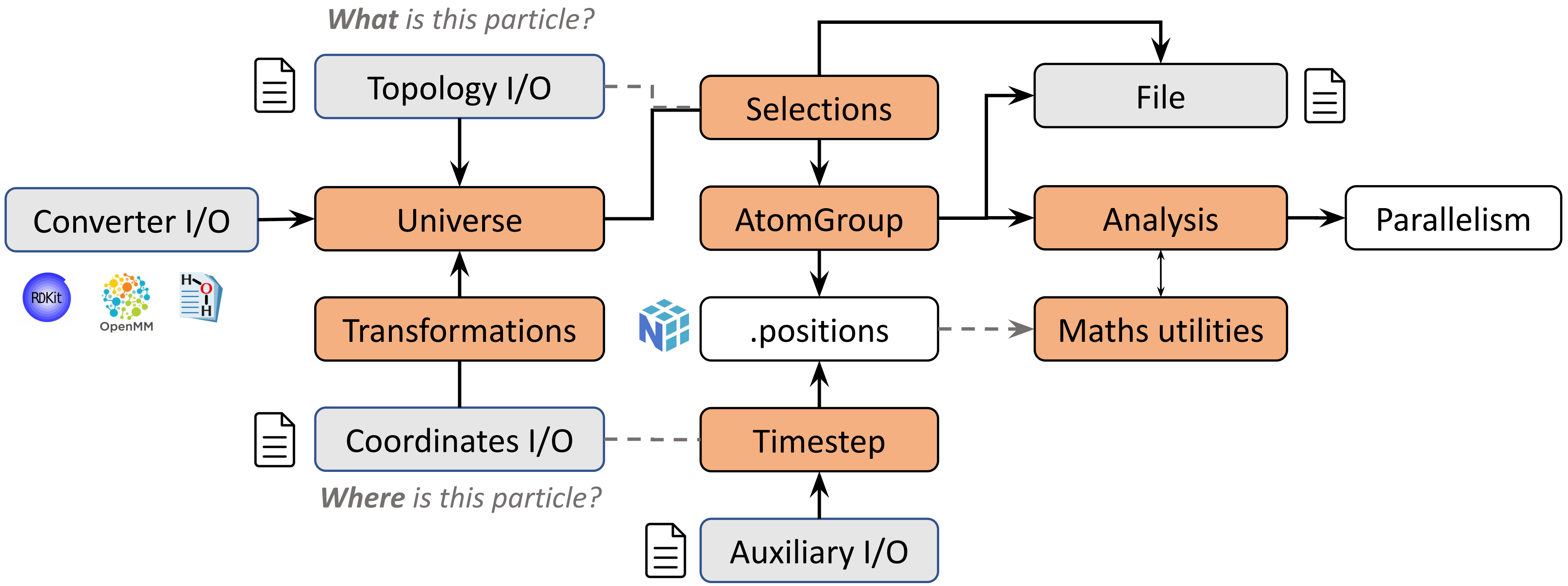


```
cluster = dask.distributed.LocalCluster()
add_workers_to_cluster()
# workers can be distributed on multiple different machines.
```

serial RMSD analysis in total: 1000 s × 96

parallel RMSD analysis (230 workers): 1200 s
1/80 serial time

MDAnalysis


What is this particle?
Topology I/O
Selections
File
Converter I/O
Universe
AtomGroup
Analysis
Parallelism
RDKit
OpenMM
Transformations
.positions
Maths utilities
Coordinates I/O
Timestep
Where is this particle?
Auxiliary I/O

Future directions

Future directions for MDAnalysis

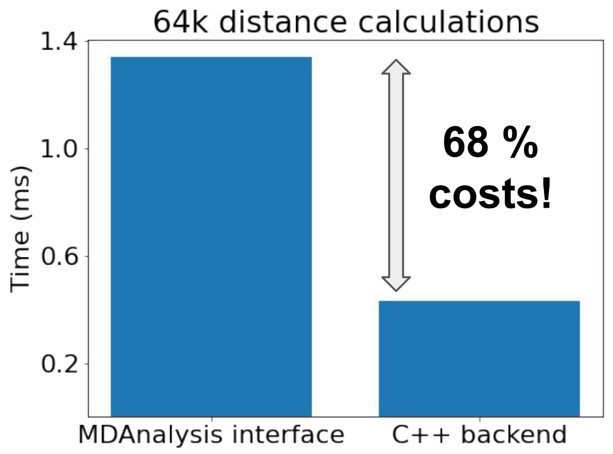
Focusing on performance and ecosystem building

- Chan Zuckerberg Initiative EOSS 4 funded project (2022 to 2023)
- Opportunity to focus on improving user experience
  - Enable faster and more extensible simulation data analyses
  - Mature API - minimal impact on user facing components
- Two major aims:
  1. Improving performance
  2. Enabling the development of reproducible MDAnalysis-using codes

Improving performance

Towards faster data handling routines

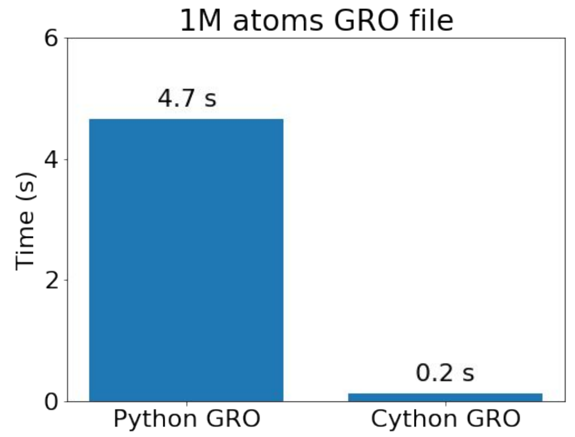
- Enable processing of increasingly large datasets
  - Cython centric strategy: from Python to C/C++
- Rewriting core data structures
  - Reduced memory access overheads
  - Better interoperability / usability with non-Python libraries



Improving performance

Towards faster data handling routines

- Enable processing of increasingly large datasets
  - Cython centric strategy: from Python to C/C++
- Rewriting core data structures
  - Reduced memory access overheads
  - Better interoperability / usability with non-Python libraries
- Cythonization of file parsers
  - Improved performance (especially for ASCII formats)
  - Streamed compressed reading of ASCII formats
  - Direct interface with C-level libraries for binary I/O



Improving reproducibility

Ensuring software reproducibility

- Reproducibility crisis
  - Code rarely provided in publications
  - Improved by recent community efforts
- Provided code often insufficient
  - Lacks tests, documentation, version control…
  - Quickly becomes non-reproducible
    - Python is a very dynamic language
  - Affects "packages" too!
- Outcomes
  - Time spent periodically re-implementing
  - Unknown changes can lead to erroneous results!

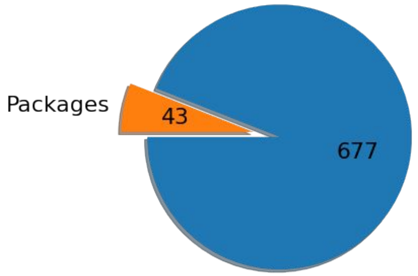


Fig 1. Number of published MDAnalysis-using "packages" since 2017*

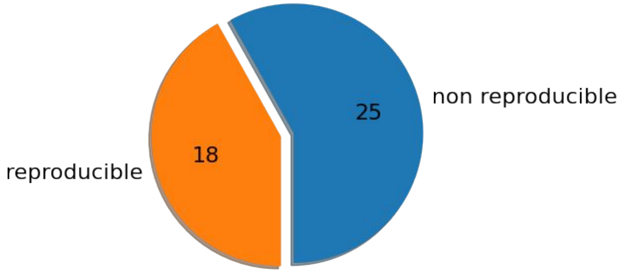


Fig 2. Breakdown of reproducible "packages"**

*Data gathered from years 2017+ Scopus & JOSS entries, "package" is defined as code advertised for re-use in a github, gitlab, or bitbucket repository.
N.B. Approximate, likely underestimated counts as Scopus has limited indexing of some journals and much validation was manual.

**Reproducible is counted as having unit tests, non-minimal documentation, and a means of installation (usually via setuptools)

Improving reproducibility

Ensuring software reproducibility

- How do we tackle this issue?
  - Ensure better code development and sharing practices
    - Unit tests, documentation, version control, ease of access, etc…
    - Existing efforts in this space; NumFOCUS, MOLSSI, BioExcel, OMSF, etc...
- Solutions for MDAnalysis-using packages
  - Increase adoption of user-developed codes in the MDAnalysis core library ❌
    - Not feasible; developer time intensive, long release cycles, dependency limits, etc…

Improving reproducibility

Ensuring software reproducibility

- How do we tackle this issue?
  - Ensure better code development and sharing practices
    - Unit tests, documentation, version control, ease of access, etc…
    - Existing efforts in this space; NumFOCUS, MOLSSI, BioExcel, OMSF, etc...
- Solutions for MDAnalysis-using packages
  - Increase adoption of user-developed codes in the MDAnalysis core library
    - Not feasible; developer time intensive, long release cycles, dependency limits, etc…
  - Help enable the development of downstream packages
    - Expose packages, encourage best practices, lower barrier to entry to package development, etc... ✅

Improving reproducibility

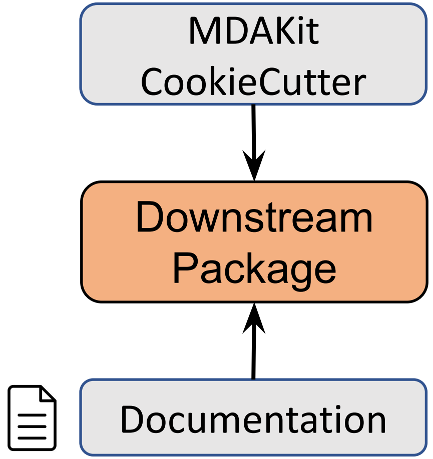
Enabling user-developed packages

- Develop an MDAKit ecosystem
    - Inspired by scipy's *scikit* system
    - Collection of packages that use MDAnalysis and meet standards of reproducibility
        - Testing
            - Unit tests + Continuous integration
        - Version control
        - Documentation
            - API + user docs
        - Community guidelines

- MDAnalysis support
    - Tools and documentation
    - Code review
    - Exposed via the MDAnalysis ecosystem
        - MDAKit registry

MDAKits

Proposed MDAKit workflow



COOKIECUTTER

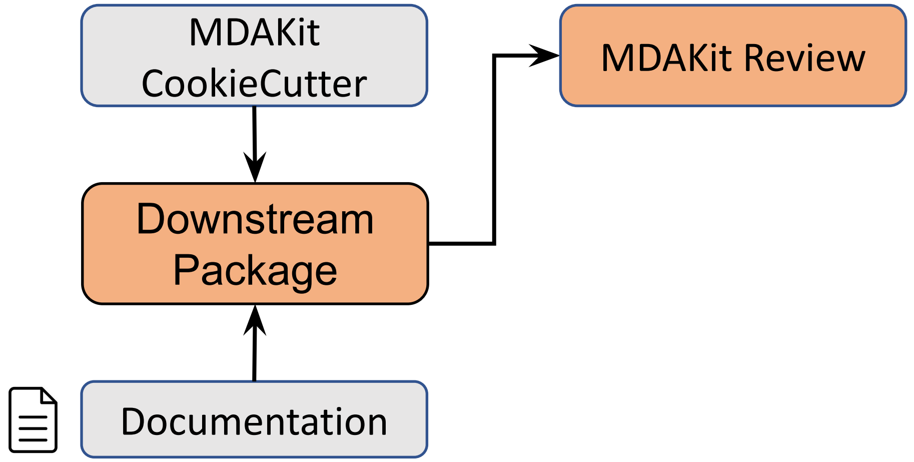
https://github.com/cookiecutter/cookiecutter

- CookieCutter MDAKits
  - Templates for key components
    - AnalysisBase
    - Readers/Writers
    - Library components
    - Continuous integration
    - Documentation
- Documentation
  - MDAKits requirements
  - Examples of MDAKit building
  - How to get the most out of MDAnalysis

Look out for our upcoming white paper!

MDAKits

Proposed MDAKit workflow

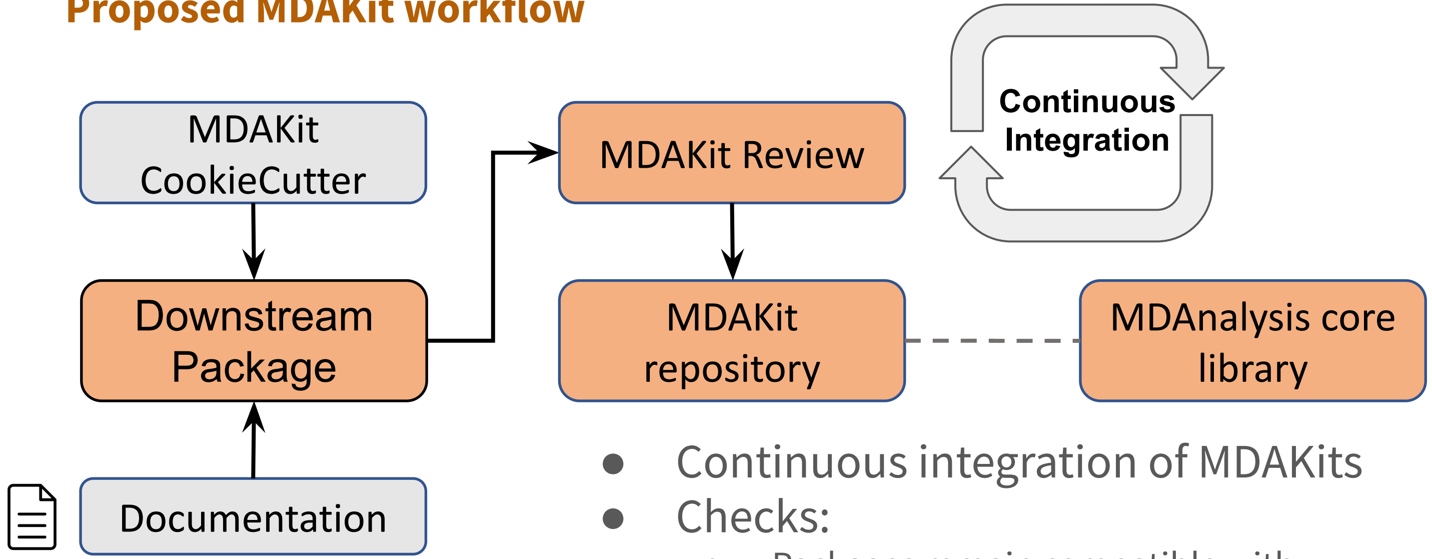


- Non-scientific review process
- Checks MDAKit adheres to reproducibility and integration requirements
  - Unit tests and continuous integration
  - Documentation
  - API compatibility
    - Use of AnalysisBase, etc...

Look out for our upcoming white paper!

MDAKits

Proposed MDAKit workflow

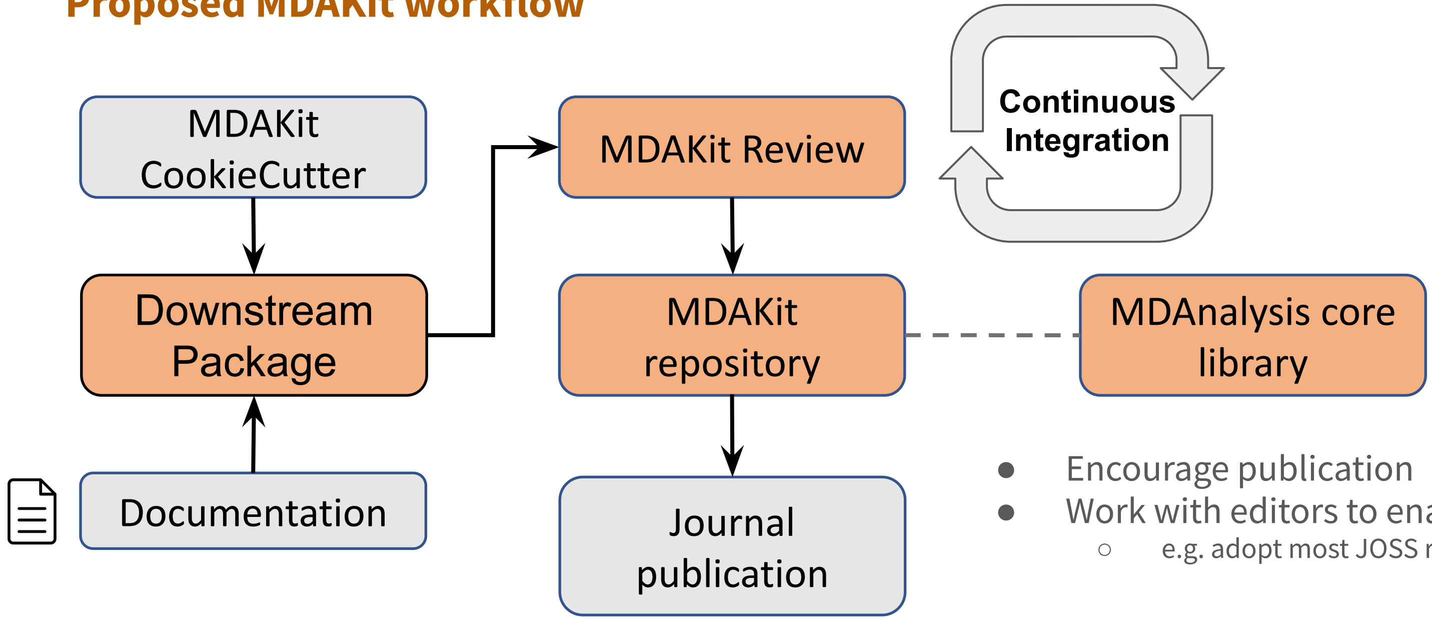


- Continuous integration of MDAKits
- Checks:
  - Packages remain compatible with upstream MDAnalysis
  - Conflicts between MDAKits
  - If MDAKits still work

Look out for our upcoming white paper!

MDAKits

Proposed MDAKit workflow



- Encourage publication
- Work with editors to enable process
  - e.g. adopt most JOSS review standards

Look out for our upcoming white paper!

Other future directions

Continued improvement of MDAnalysis components

- New converters
  - ASE, OpenBabel, LOOS, PyTraj, MDTraj, etc…
- New file formats
  - TNG reader/writer (https://github.com/MDAnalysis/pytng)
  - Multi-threaded read/write support (via HM5D, etc…)
- Command-line interface
  - https://github.com/MDAnalysis/mdacli
- Improved packaging / releases
  - Adoption of NEP29
  - Fortnightly development releases

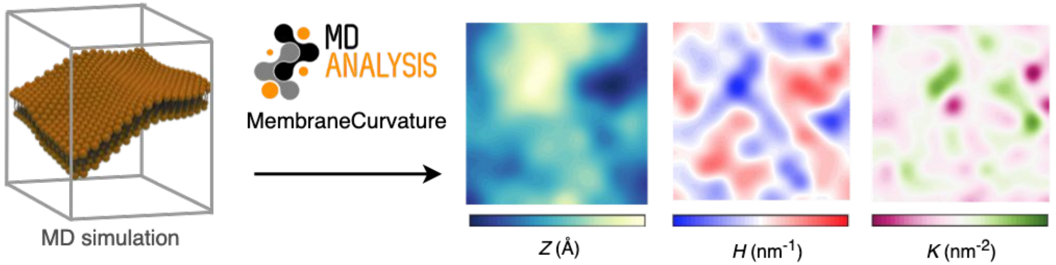
Hugo MacDermott-Opeskin
Modernising TNG code & python bindings

Help develop MDAnalysis

- All contributions appreciated!
- Participate
    - Email lists
    - Discord
- Let us know what we do wrong
    - Bug reports, feature requests, etc…
- Code contributions
    - 324 entries to our issue tracker
    - New ideas / changes always welcome
- Google Summer of Code
    - Funds for student developers
    - 10 week summer projects

Google Summer of Code

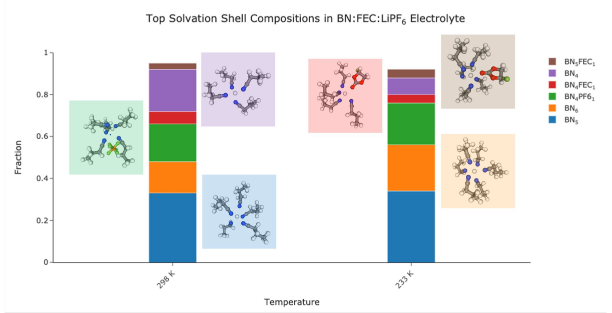
MembraneCurvature
A tool to calculate mean and Gaussian membrane curvature

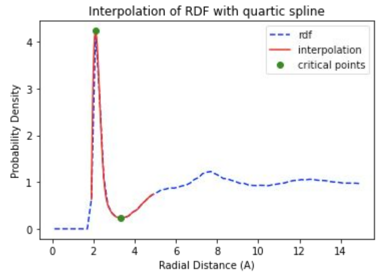


github.com/MDAnalysis/membrane-curvature

Estefania Barreto-Ojeda

SolvationAnalysis
A suite of tools for analyzing the solvation structure of a liquid





github.com/MDAnalysis/solvation-analysis

Orion Cohen

Thanks for listening :)

MDAnalysis 2.0 is now out!

GitHub

github.com/MDAnalysis

User Guide

userguide.mdanalysis.org

Join the conversation at

@mdanalysis

discord.gg/fXTSfDJyxE

Acknowledgements

All 137 MDAnalysis code contributors and the many more community members that use MDAnalysis, report bugs, and make feature requests.

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Q & A