MDAnalysis 2.0 and beyond: Extensible and interoperable simulation analyses

Irfan Alibay[1], Jonathan Barnoud[2], Oliver Beckstein[3], Cédric Bouysset[4], Richard J. Gowers[5], Hugo MacDermott-Opeskin[6], Micaela Matta[7], Manuel N. Melo[8], Fiona B. Naughton[3], Tyler Reddy[9], Lily Wang[6], Yuxuan Zhuang[10]

1. Department of Biochemistry, University of Oxford, Oxfordshire, United Kingdom
2. School of Chemistry, University of Bristol, Bristol, United Kingdom
3. Department of Physics, Arizona State University, Tempe, AZ, United States
4. Institut de Chimie de Nice, Université Côte d'Azur, Nice, France
5. NextMove Software Ltd, Cambridge, Cambridgeshire, United Kingdom
6. Research School of Chemistry, Australian National University, Canberra, ACT, Australia
7. Department of Chemistry, University of Liverpool, Liverpool, Merseyside, United Kingdom
8. Instituto de Tecnologia Química e Biológica António Xavier, Universidade Nova de Lisboa, Lisboa, Portugal
9. CCS-7 Applied Computer Science, Los Alamos National Laboratory, Los Alamos, NM, United States
10. Department of Biochemistry and Biophysics, Stockholms Universitet, Stockholm, Stockholm, Sweden

GitHub
github.com/MDAnalysis

User Guide
userguide.mdanalysis.org

Introduction

Key to understanding molecular simulations is the ability to extract meaningful features through flexible analysis workflows.

Here we present MDAnalysis, a free open-source Python library for handling molecular simulation data.

The MDAnalysis library offers:

1. Ease of use
   - Simple Python API, using familiar NumPy components
2. Extensibility
   - Frameworks and library components to create your own analysis methods
3. Interoperability
   - Support for over 40 file formats
   - Easy data conversion to/from other Python packages
4. Ease of deployment
   - Cross-platform, now also supports ARM and Power9
   - Easily available through PyPi and conda

Acknowledgements

MDAnalysis exists due to the generous work of over 136 developers and countless more community members who have contributed over the last 16 years.

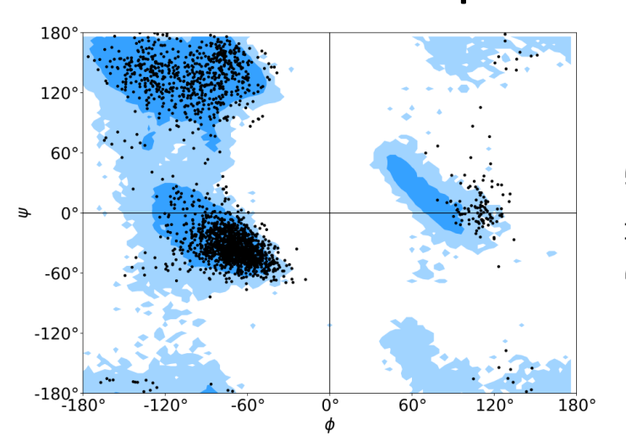
The MDAnalysis Ecosystem

Data I/O

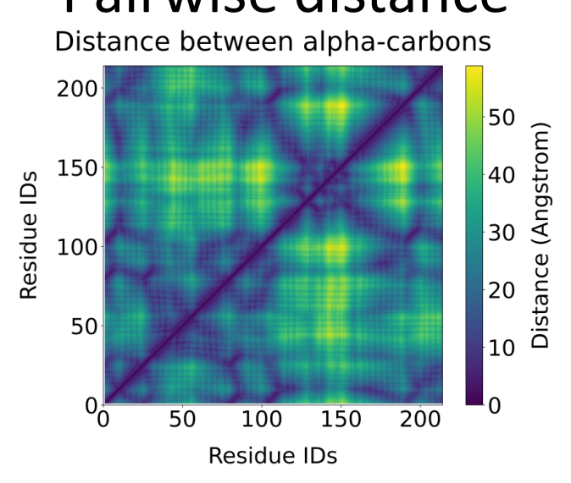
Software	File Type
AMBER	PRMTOP, RST7, TRJ, NETCDF
GROMACS	ITP, TPR, GRO, TRR, XTC
CHARMM	PSF, DCD, CRD
NAMD	DCD, COOR, NAMDBIN
LAMMPS	CONFIG, DATA, DUMP, DCD
DL_POLY	CONFIG, HISTORY
HOOMD	XML, GSD
GAMESS	GMS
DESRES	DMS
Others	XYZ, TXYZ, PDB, PDBQT, PQR, TRZ, MOL2, MMTF, FHIAIMS, H5MD, etc...

Fast math & distance routines

Ramachandran plot



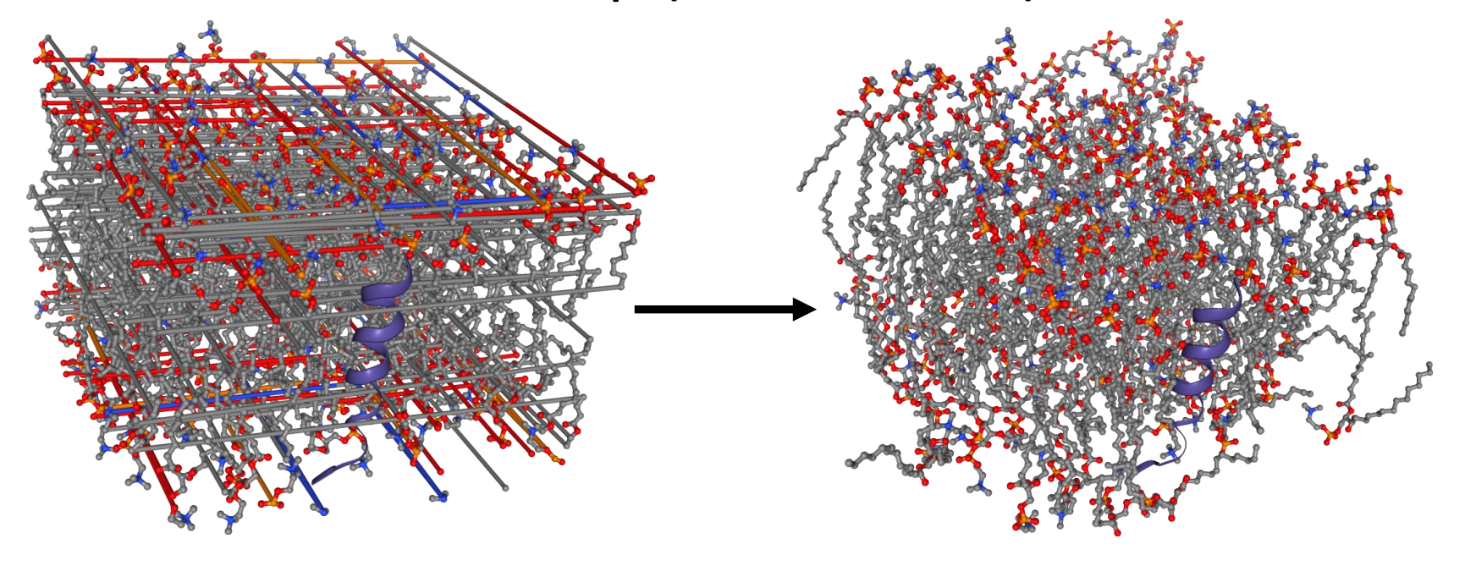
Pairwise distance



Transformations

On-the-fly trajectory manipulation

unwrap (make whole)

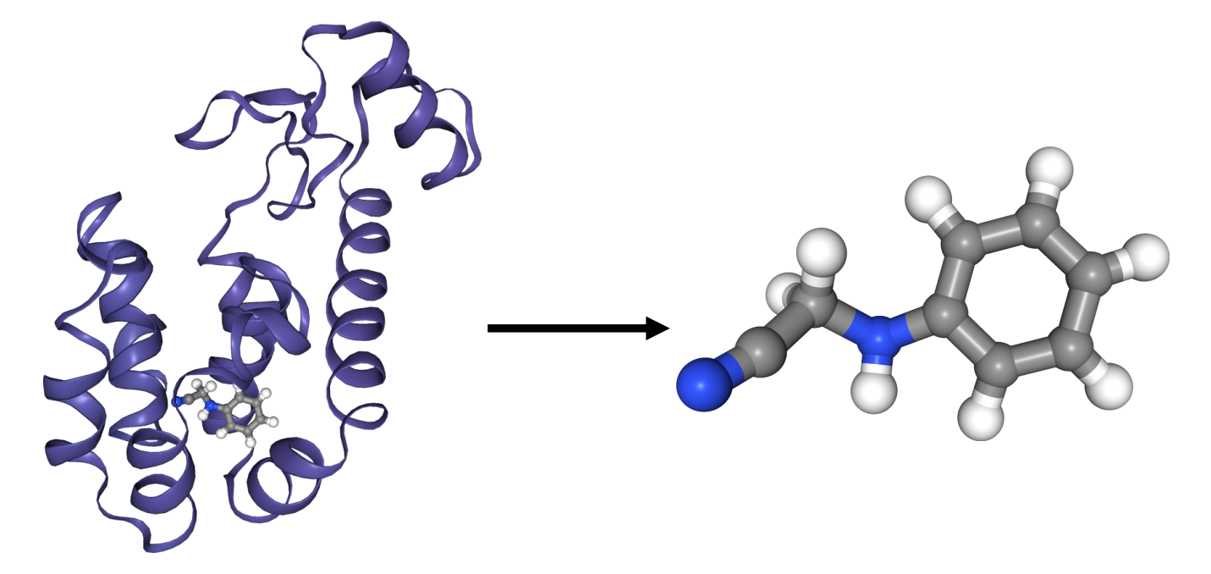


MDAnalysis Universe & AtomGroup

Selections

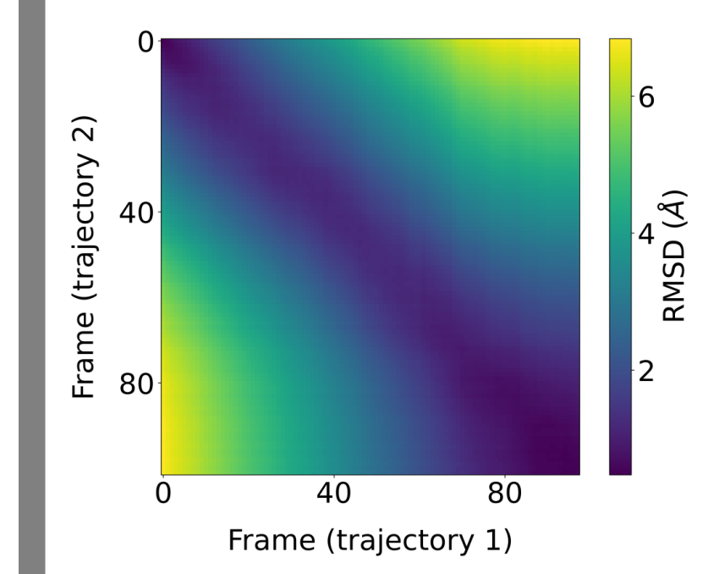
Intuitive atom selections

```
universe = mda.Universe(TOP, TRAJ)
ligand = universe.select_atoms('resname LIG')
```

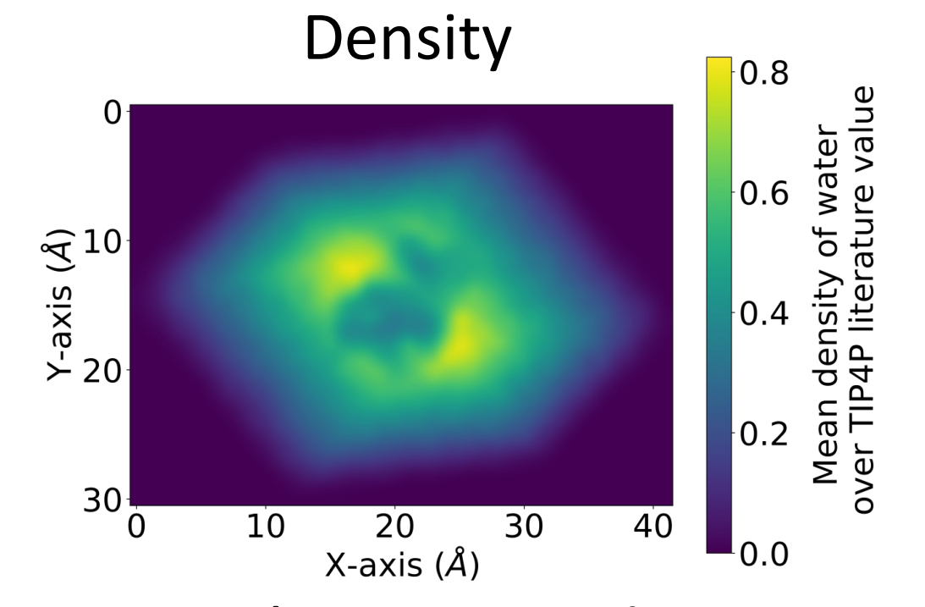


Analyses

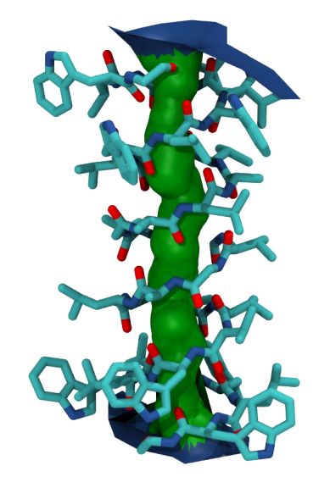
Pairwise RMSD



Density



Pore analysis (HOLE)



And many more!

Example Analysis: RMSF

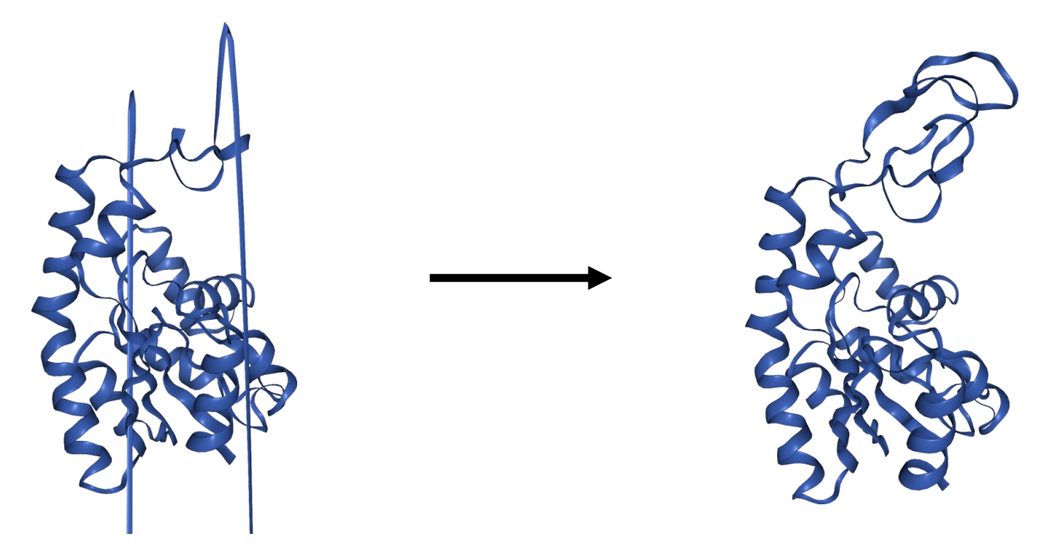
Imports and selections

```
import MDAnalysis as mda
import MDAnalysis.transformations as trans
from MDAnalysis.analysis import rms, align

u = mda.Universe(TOP, TRAJ)
protein = u.select_atoms('protein')
not_protein = u.select_atoms('not protein')
c_alphas = u.select_atoms('protein and name CA')
```

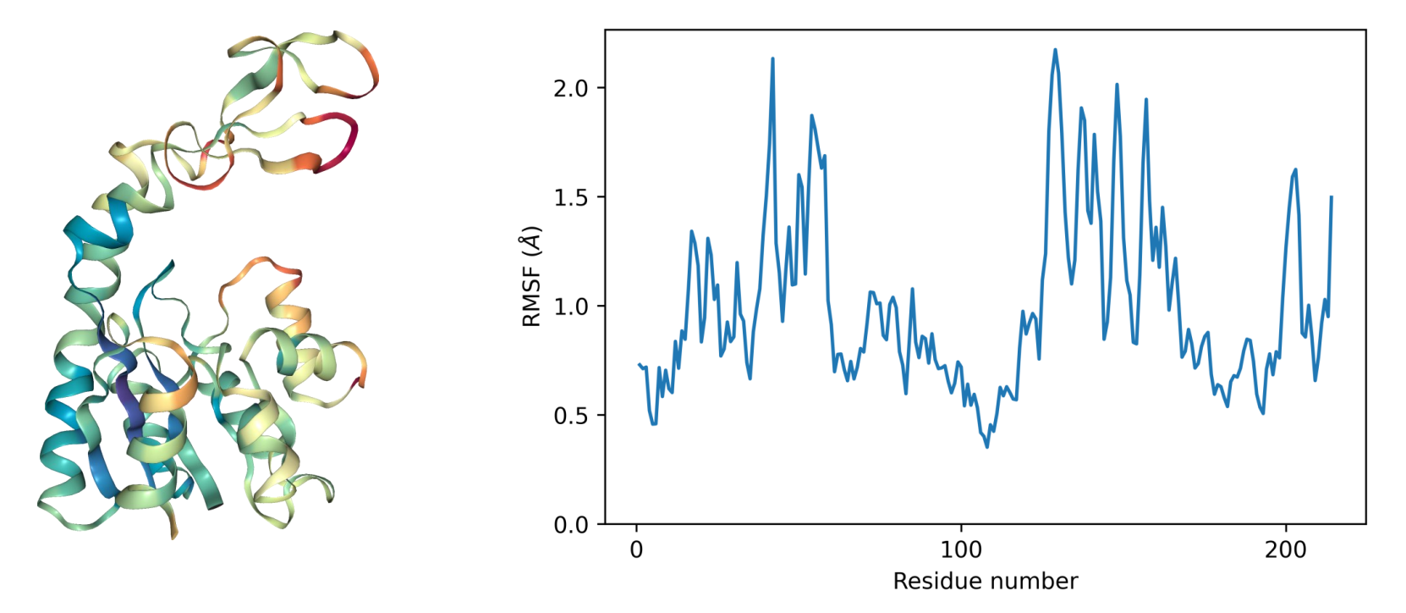
Fix PBC + align

```
transform = [trans.unwrap(protein),
             trans.center_in_box(protein, wrap=True),
             trans.wrap(not_protein),
             trans.fit_rot_trans(
                 c_alphas, c_alphas, weights='mass')]
u.trajectory.add_transformations(*transform)
```



RMSF

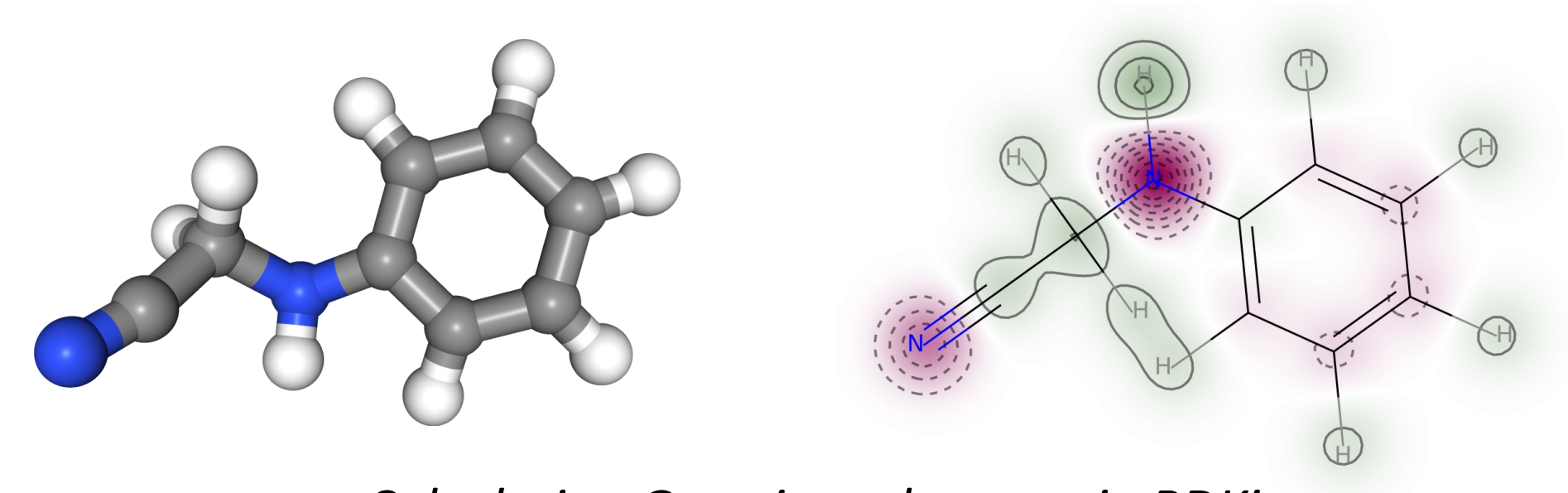
```
R = rms.RMSF(c_alphas).run()
```



Converters

- Seamless conversion layers of MDAnalysis structures to and from other Python packages
  - ♦ Chemfiles, ParmEd, RDKit, OpenMM

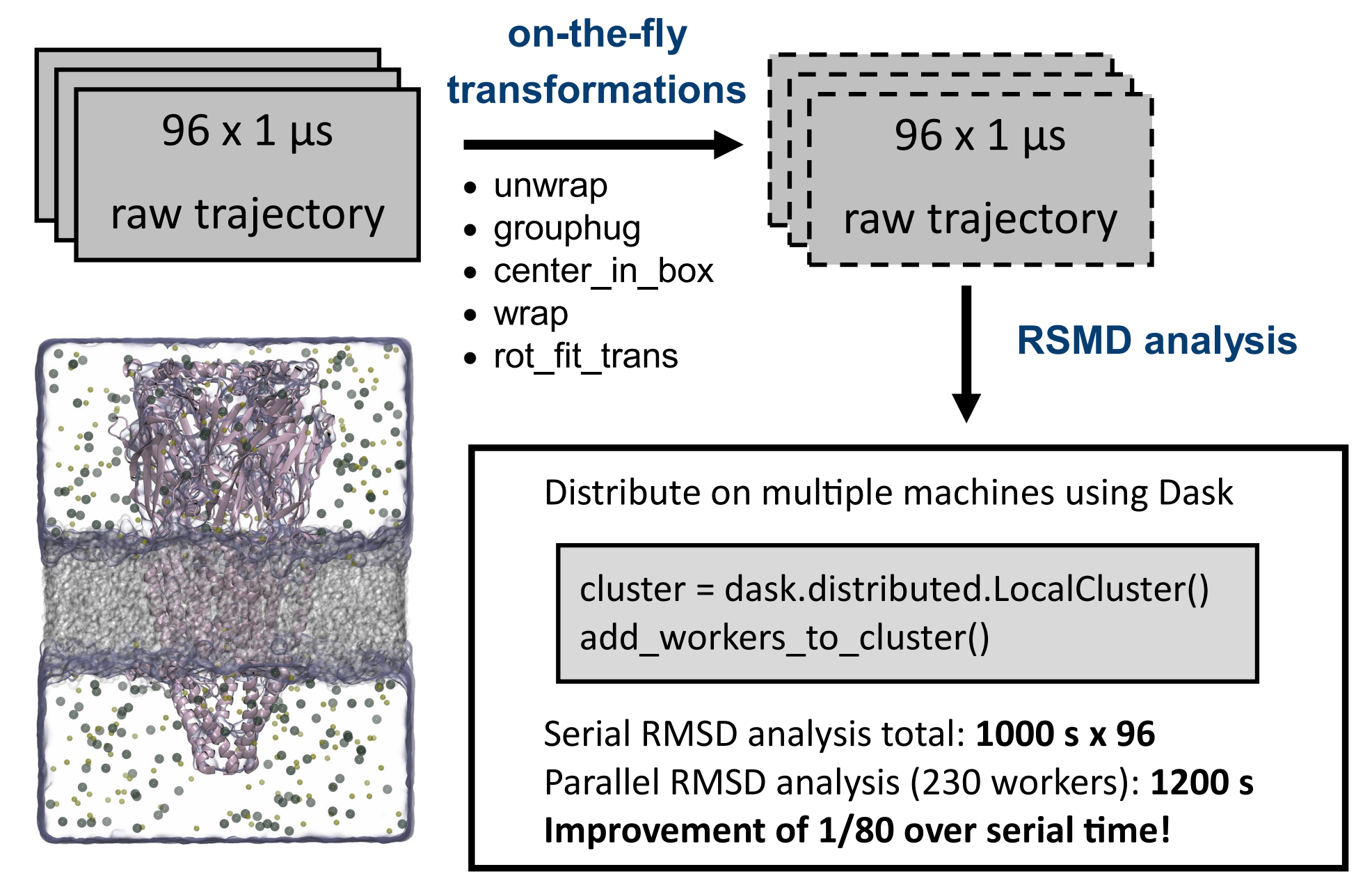
```
import MDAnalysis as mda
from rdkit.Chem import AllChem
u = mda.Universe(TOP, TRAJ)
ligand = u.select_atoms("resname LIG")
rdmol = ligand.convert_to("RDKIT")
AllChem.ComputeGasteigerCharges(rdmol)
```



Calculating Gasteiger charges via RDKit

Parallelism

- Can be easily combined with parallel frameworks to reduce time to solution
  - ♦ multiprocessing, dask, joblib, mpi4py



Distribute on multiple machines using Dask

```
cluster = dask.distributed.LocalCluster()
add_workers_to_cluster()
```

Serial RMSD analysis total: **1000 s x 96**
Parallel RMSD analysis (230 workers): **1200 s**
Improvement of 1/80 over serial time!

Speeding up a large scale RMSD analysis of a 250,000 atoms multimeric membrane protein

Future Directions

- Improved cythonization (C/C++) of core components
- Addition of new converters
  - ♦ ASE, OpenBabel, LOOS, PyTraj, MDTraj
- Increased focus on extensibility
  - ♦ Cookiecutter-based templates
- New file formats, analyses, components
  - ♦ TNG, solvation analysis, membrane curvature, command-line interface