BMI 206

Structure Prediction Lab

11/18/16

Benjamin Webb, Sali Lab
(ben@salilab.org)

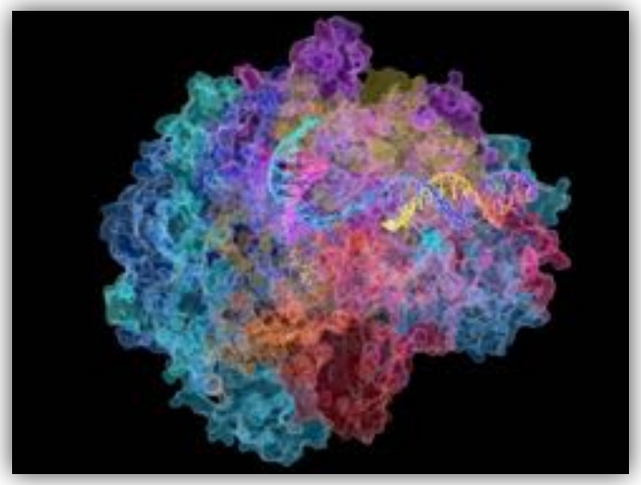
Why Protein Structure Prediction?

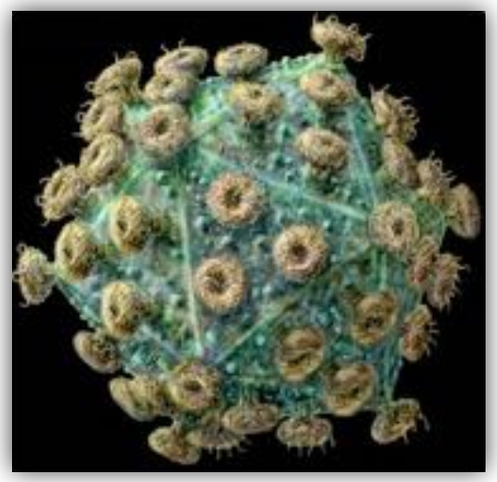
	Y 2016
Sequences	70,650,000
Structures	128,000

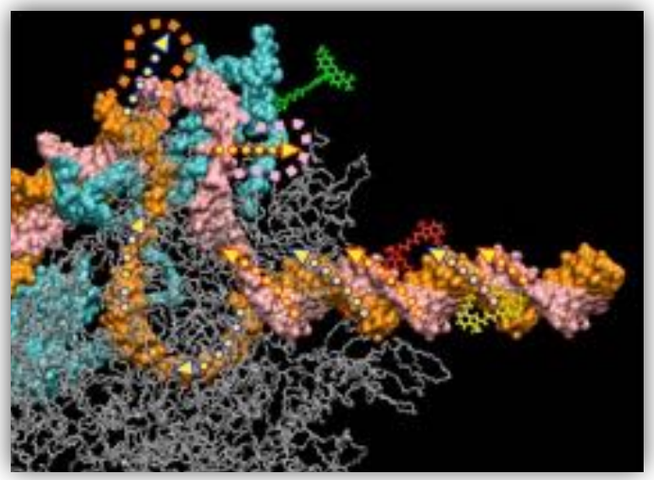
- We have an experimentally determined atomic structure for less than 1% of the known protein sequences (and this gets worse every year).
- For *assemblies* of multiple proteins, even less is known.

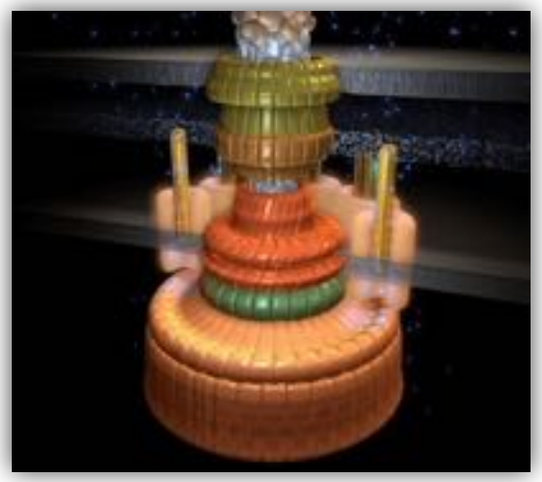
Structural biology:
Maximize accuracy, resolution, completeness, and efficiency of the structural coverage of macromolecular assemblies

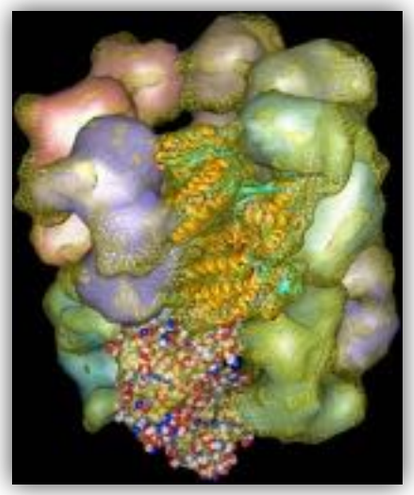
Motivation: Models will allow us to understand how machines work, how they evolved, how they can be controlled, modified, and perhaps even designed.

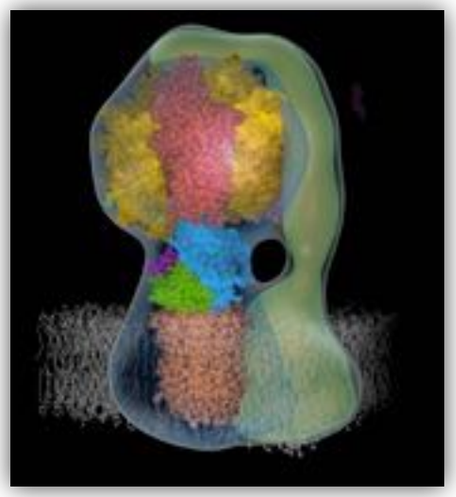

RNA polymerase II

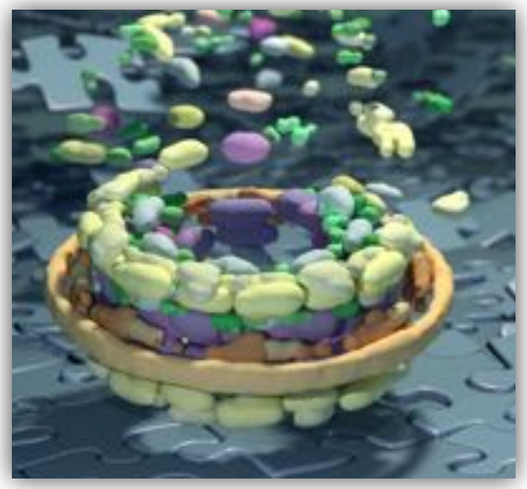

HIV virus

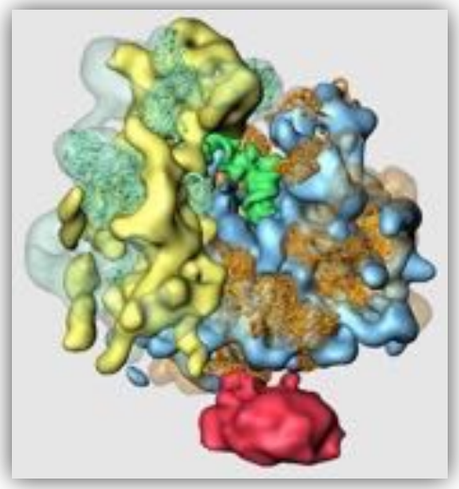

tRNA synthetase


flagellar motor


GroEL chaperonin


ATP synthase


nuclear pore complex


ribosome

There may be thousands of biologically relevant macromolecular complexes whose structures are yet to be characterized, involved in a few hundred core biological processes.

Integrative Structural Biology

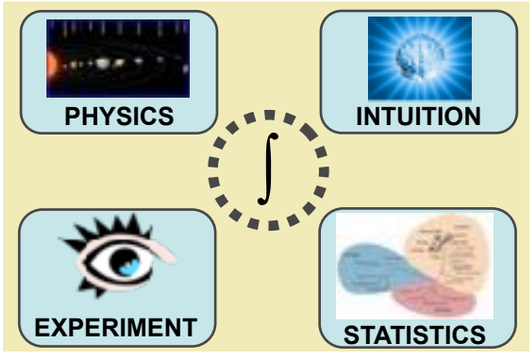
for maximizing accuracy, resolution, completeness, and efficiency of structure determination

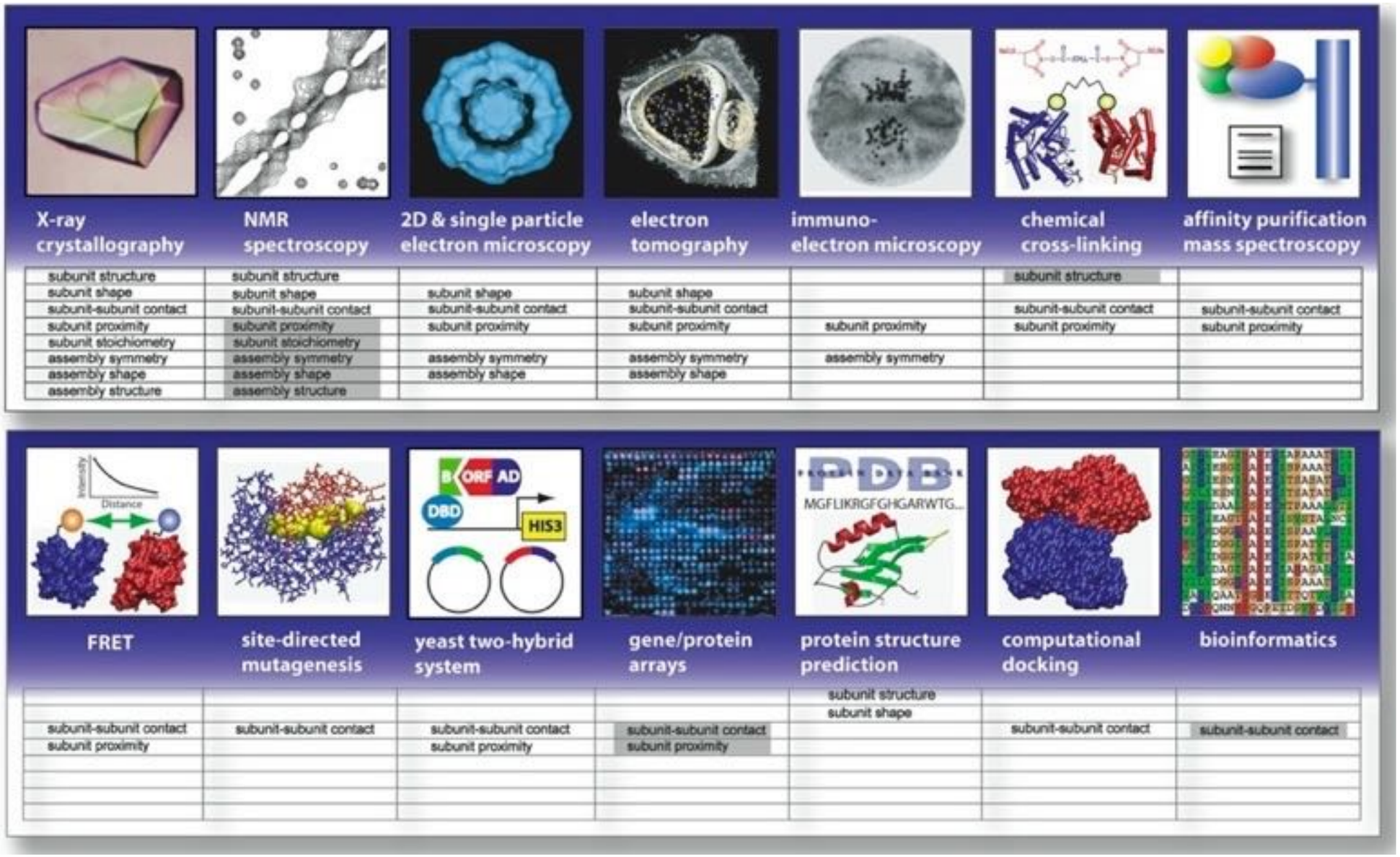
Use structural information from any

source: measurement, first principles, rules;

resolution: low or high resolution

to obtain the set of all models that are consistent with it.





Sali A, Earnest T, Glaeser R, Baumeister W. From words to literature in structural proteomics. *Nature* 422, 216-225, 2003.
Ward A, Sali A, Wilson I. Integrative structural biology. *Science* 339, 913-915, 2013.

Comparative modeling by satisfaction of spatial restraints: MODELLER

3D GKITFYERGFQGHCYESDC-NLQP…

SEQ GKITFYERG---RCYESDCPNLQP…

A. Šali & T. Blundell. *J. Mol. Biol.* **234**, 779, 1993.
J.P. Overington & A. Šali. *Prot. Sci*. **3**, 1582, 1994.
A. Fiser, R. Do & A. Šali, *Prot. Sci*., **9**, 1753, 2000.

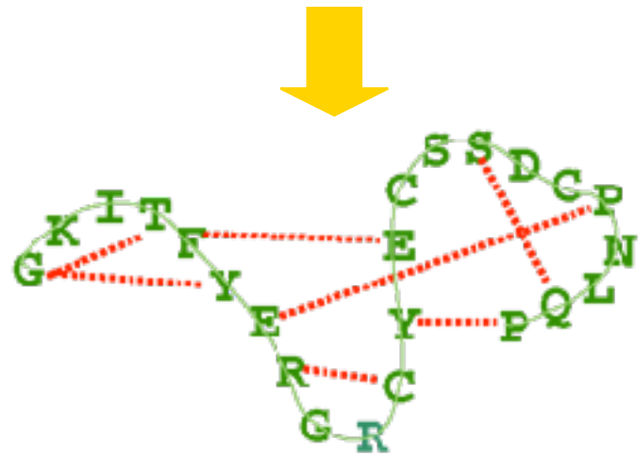
https://salilab.org/modeller/

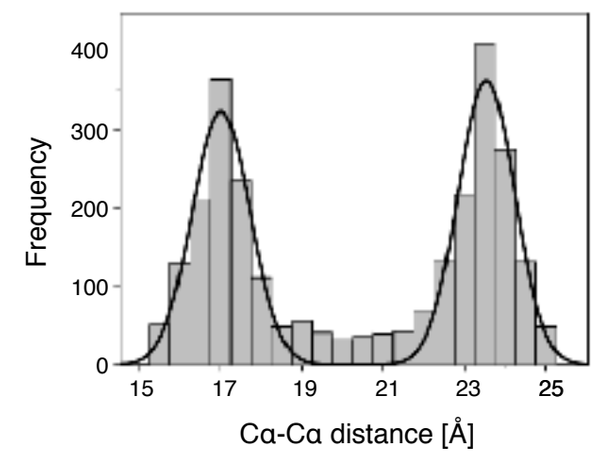
Comparative modeling by satisfaction of spatial restraints: MODELLER

3D GKITFYERGFQGHCYESDC-NLQP…

SEQ GKITFYERG---RCYESDCPNLQP…

1. Extract spatial restraints





A. Šali & T. Blundell. *J. Mol. Biol.* **234**, 779, 1993.
J.P. Overington & A. Šali. *Prot. Sci*. **3**, 1582, 1994.
A. Fiser, R. Do & A. Šali, *Prot. Sci*., **9**, 1753, 2000.

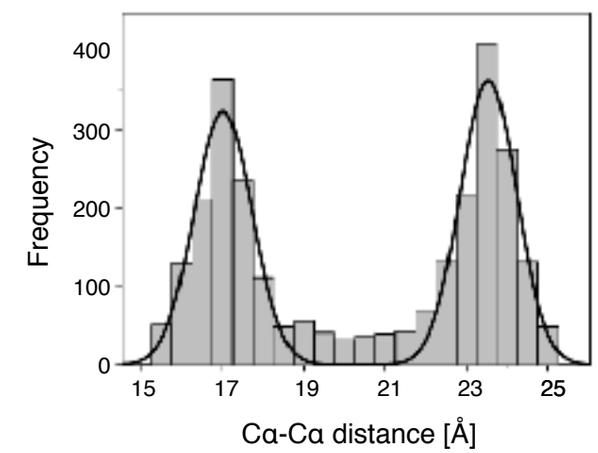
https://salilab.org/modeller/

Comparative modeling by satisfaction of spatial restraints: MODELLER

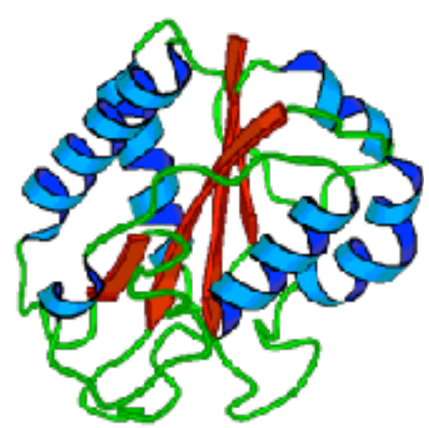
3D GKITFYERGFQGHCYESDC-NLQP…

SEQ GKITFYERG---RCYESDCPNLQP…

1. Extract spatial restraints



2. Satisfy spatial restraints



$$F(\mathbf{R}) = \prod_{i} p_i(f_i/l)$$

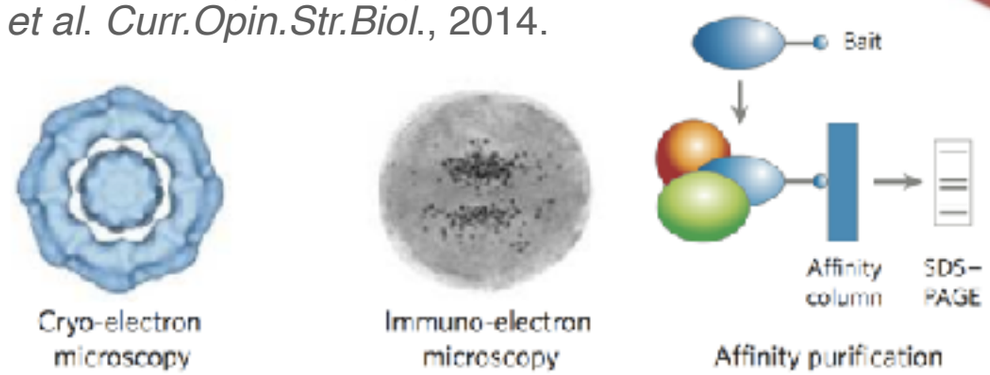
A. Šali & T. Blundell. *J. Mol. Biol.* **234**, 779, 1993.
J.P. Overington & A. Šali. *Prot. Sci*. **3**, 1582, 1994.
A. Fiser, R. Do & A. Šali, *Prot. Sci*., **9**, 1753, 2000.

https://salilab.org/modeller/

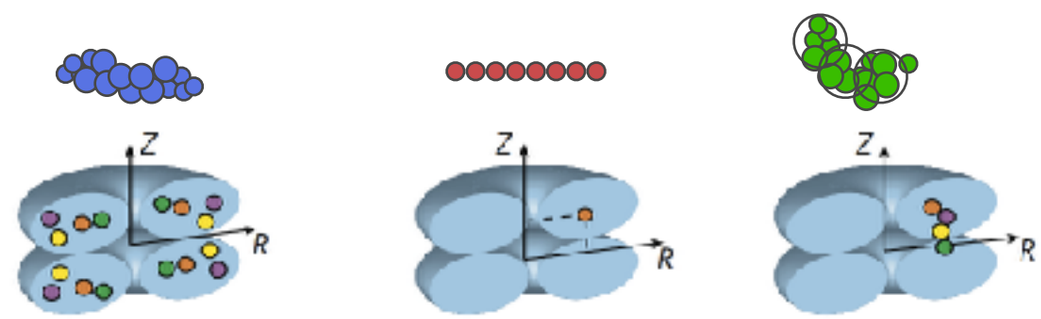
A description of integrative structure determination

Alber *et al*. *Nature* **450**, 683-694, 2007
Robinson *et al*. *Nature* **450**, 974-982, 2007
Alber *et al*. *Annual Reviews in Biochemistry* **77**, 11.1–11.35, 2008
Russel *et al*. *PLoS Biology* **10**, 2012
Ward *et al*. *Science* **339**, 913-915, 2013
Schneidman *et al*. *Curr.Opin.Str.Biol.*, 2014.

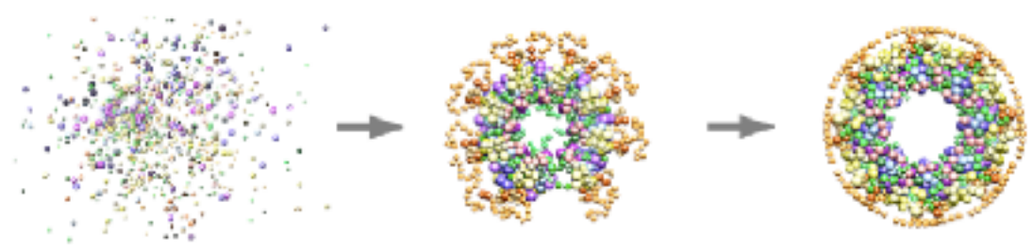
Gathering information



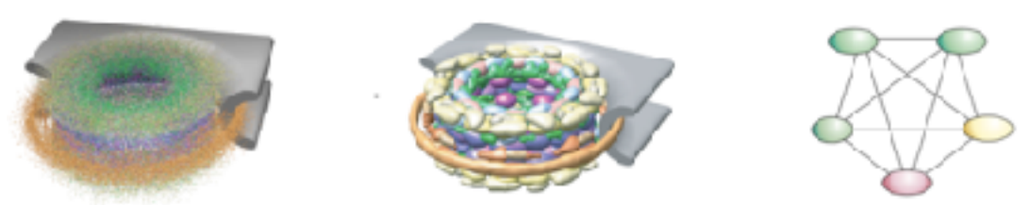
Designing model representation and evaluation



Sampling models



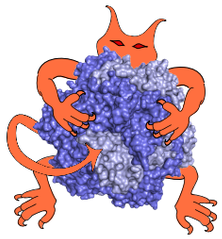
Analyzing models and information



While it may be hard to live with generalization, it is inconceivable to live without it. *Peter Gay*, Schnitzler's Century (2002).

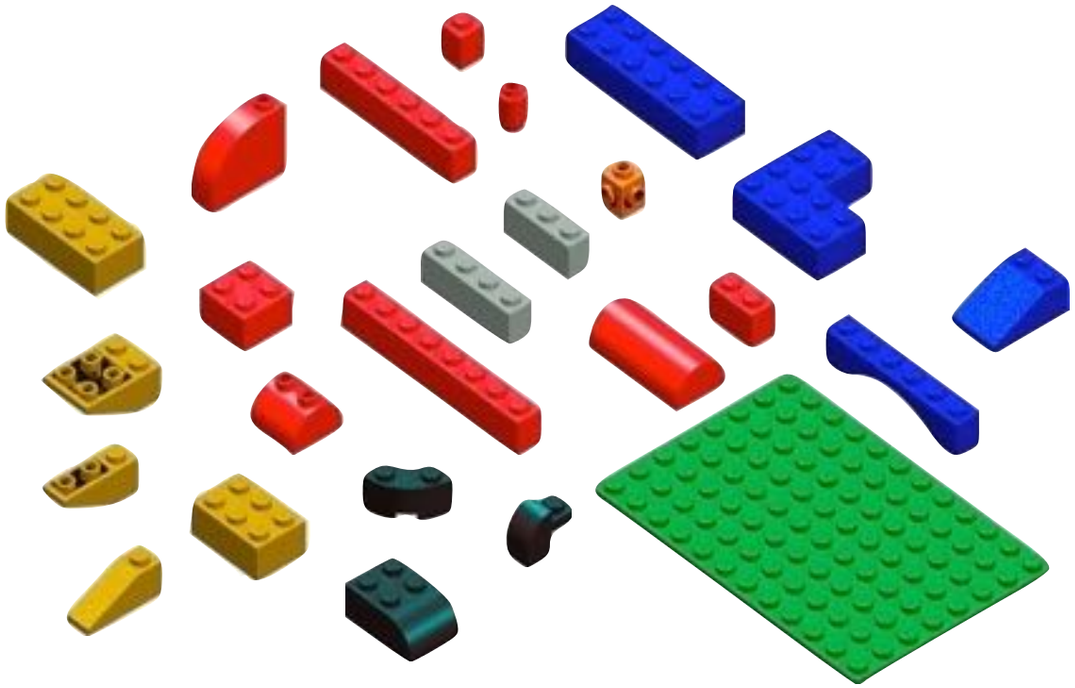
Integrative Modeling Platform (IMP)

https://integrativemodeling.org



D. Russel, K. Lasker, B. Webb, J. Velazquez-Muriel, E. Tjioe, D. Schneidman, F. Alber, B. Peterson, A. Sali, PLoS Biol, 2012.
R. Pellarin, M. Bonomi, B. Raveh, S. Calhoun, C. Greenberg, G.Dong.

- Diverse problems, so no one 'black box'
- "Mix and match" components for developing an integrative modeling protocol
- Open source (LGPL)
- Hosted on **GitHub**



Representation:

Atomic
Rigid bodies
Coarse-grained
Multi-scale
Symmetry / periodicity
Multi-state systems

Scoring:

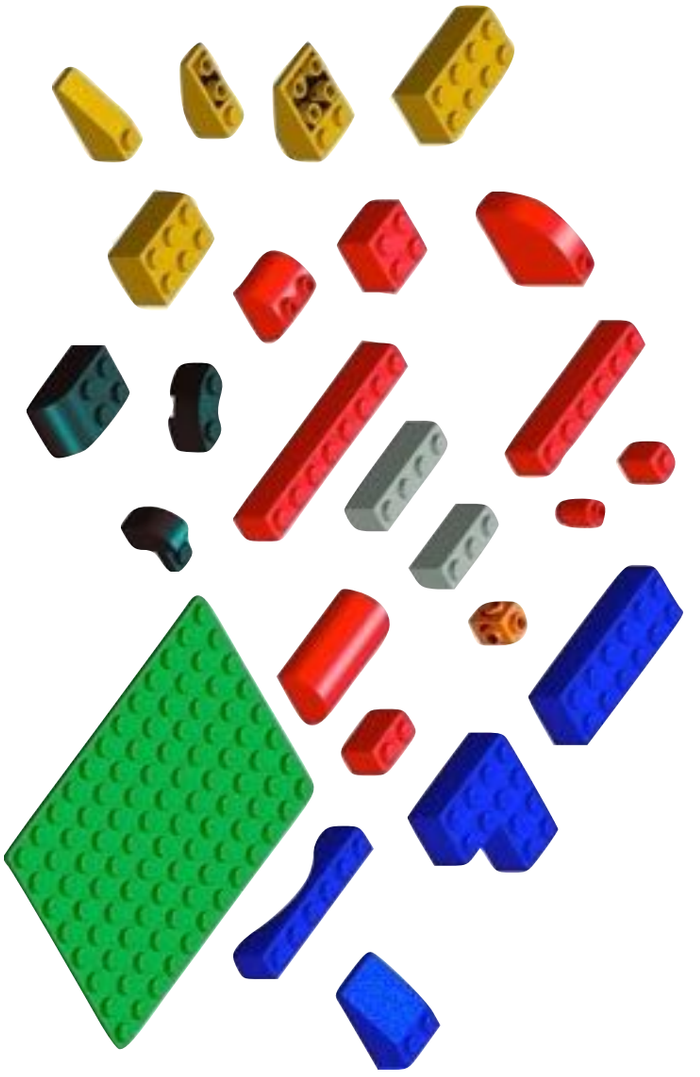
Density maps
EM images
Proteomics
FRET
Chemical and Cys cross-linking
Homology-derived restraints
SAXS
Native mass spectrometry
Statistical potentials
Molecular mechanics forcefields
Bayesian scoring
Library of functional forms (ambiguity, ...)

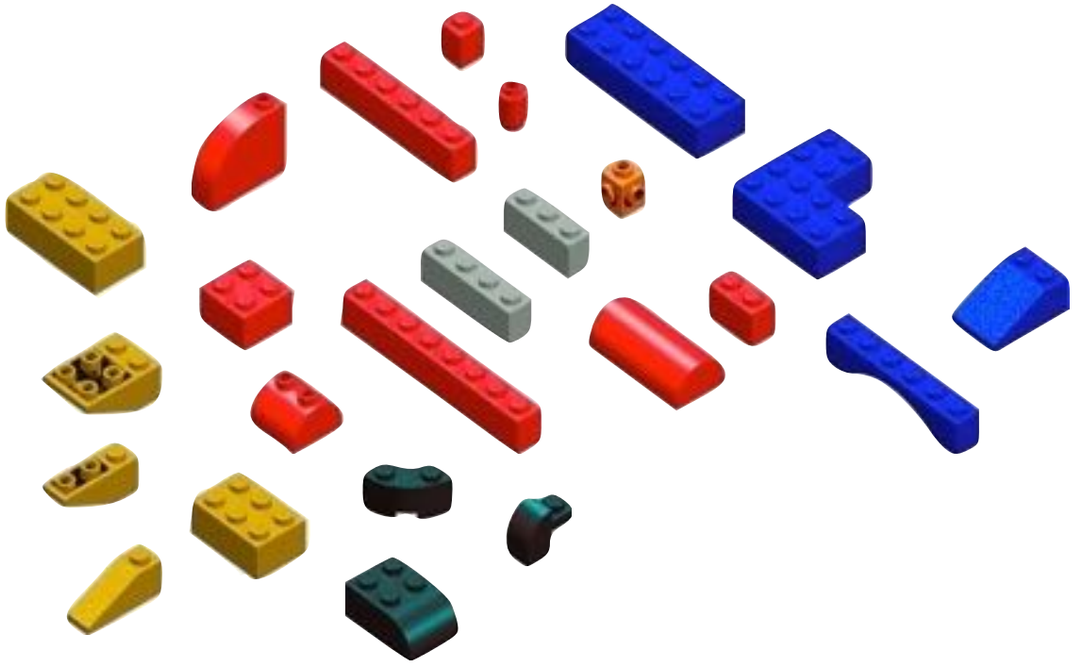
Sampling:

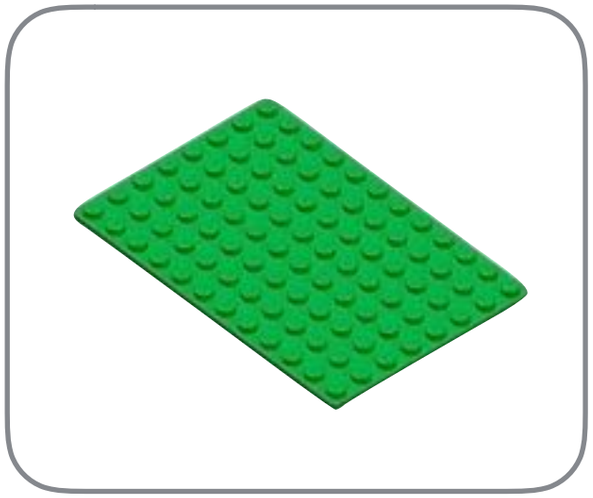
Simplex
Conjugate Gradients
Monte Carlo
Brownian Dynamics
Molecular Dynamics
Replica Exchange
Divide-and-conquer enumeration

Analysis:

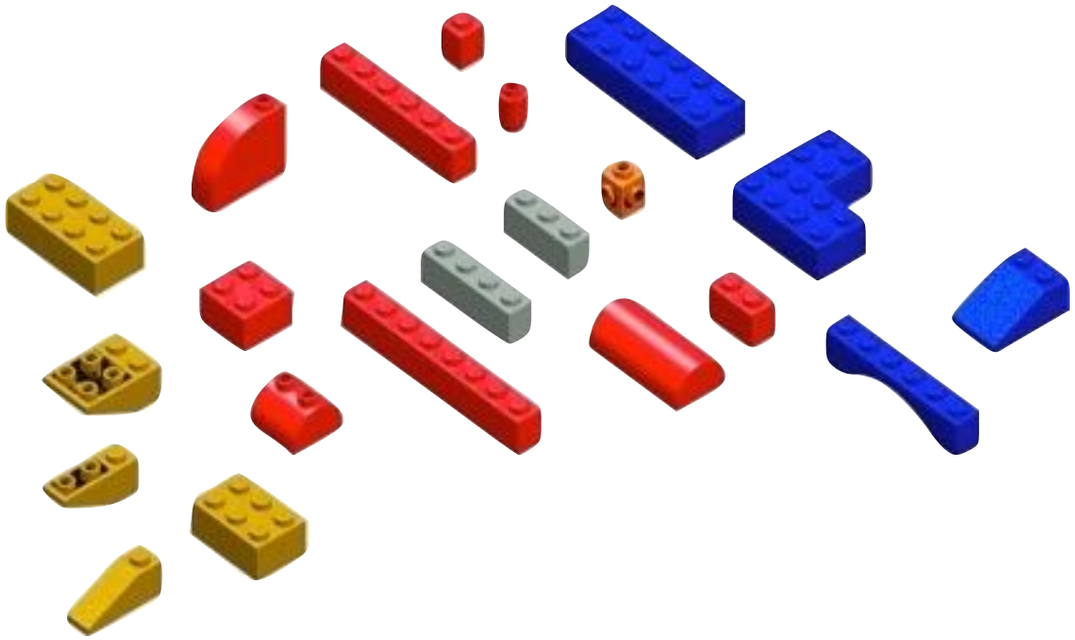
Clustering
Chimera
PyMOL
PDB files
Density maps





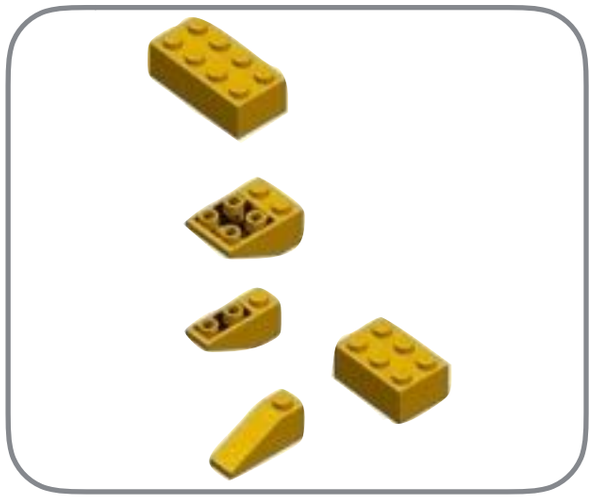


IMP kernel

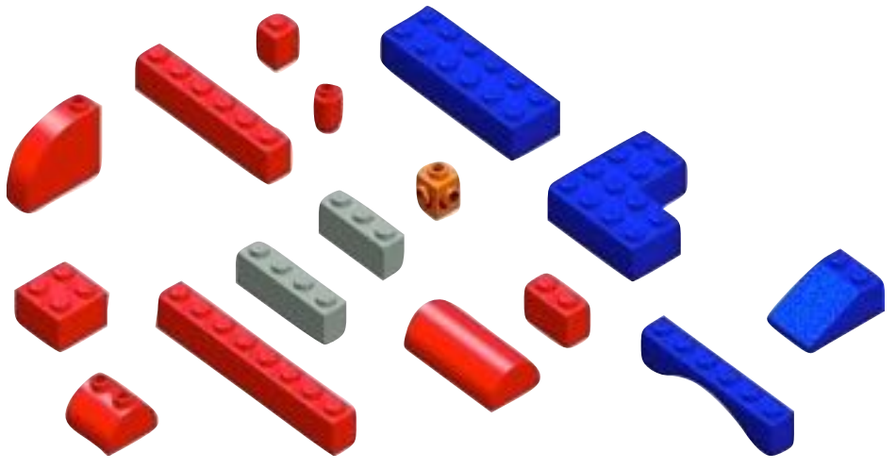


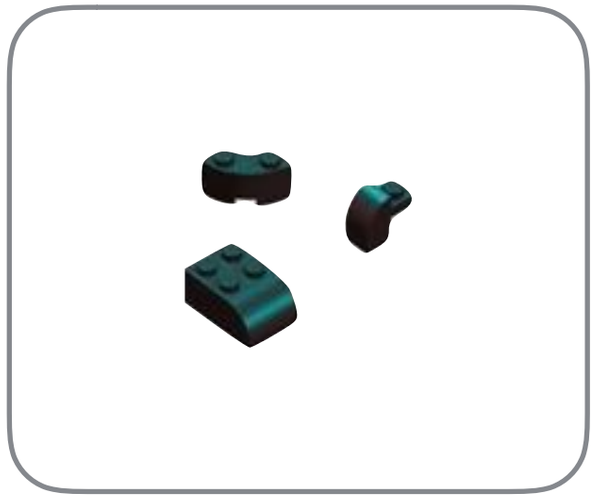
IMP.algebra

IMP kernel

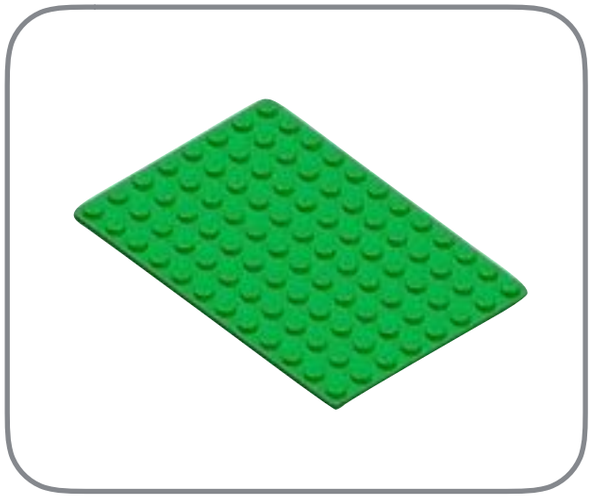


IMP.saxs

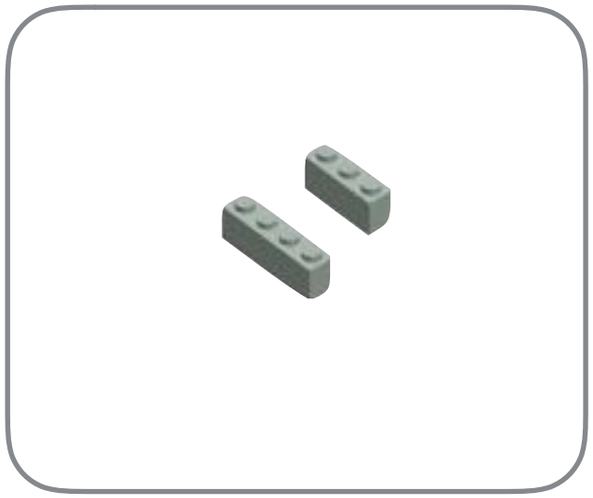




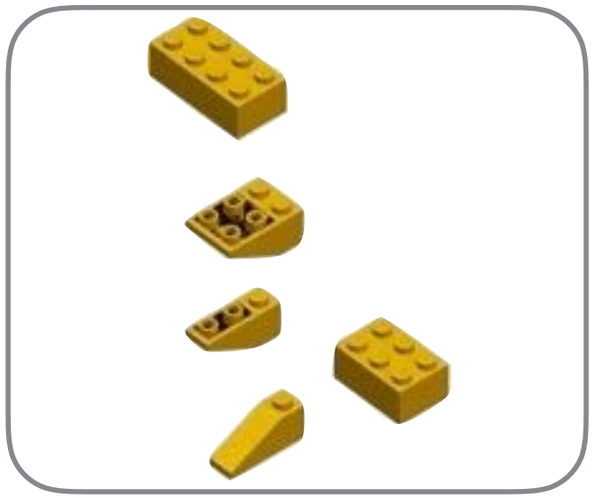
IMP.algebra



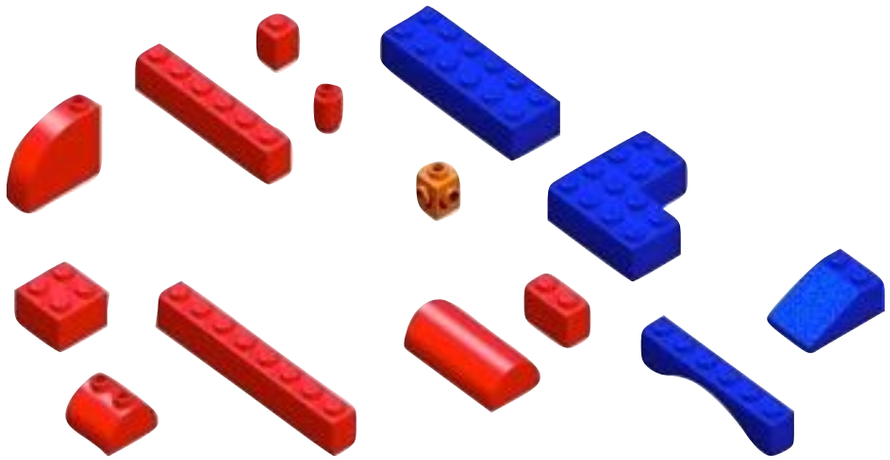
IMP kernel

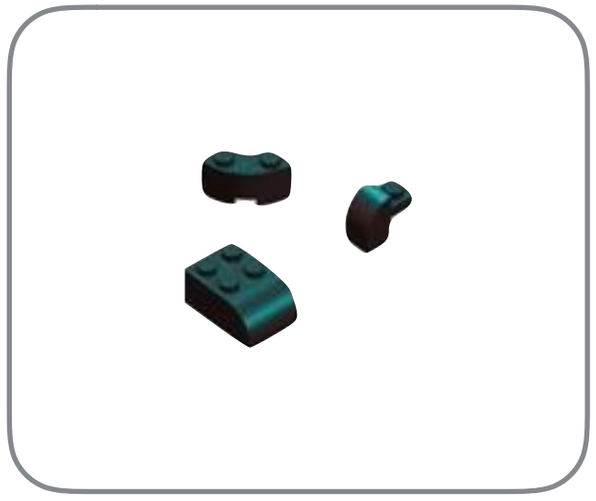


IMP.em

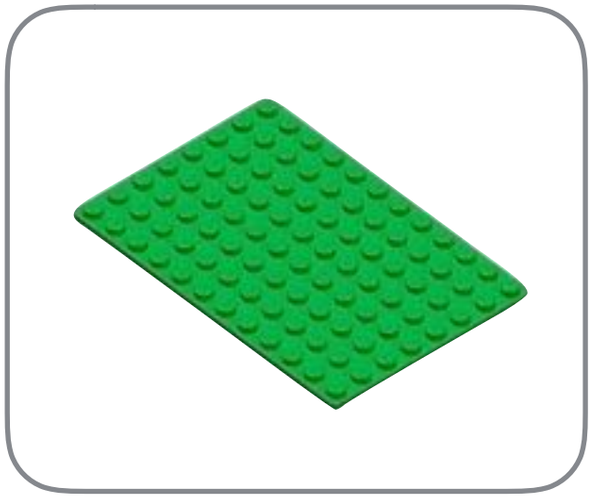


IMP.saxs

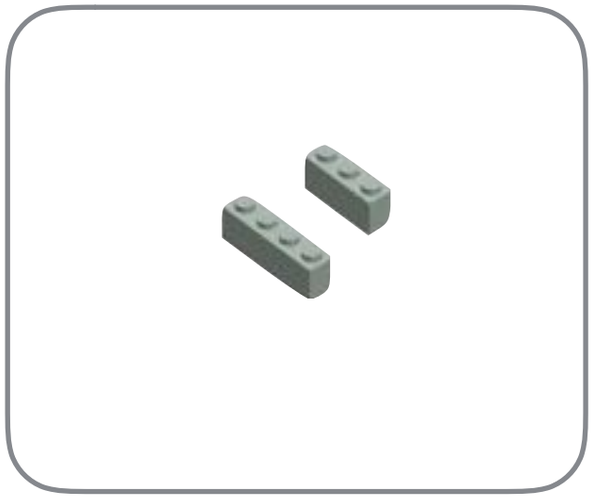




IMP.algebra

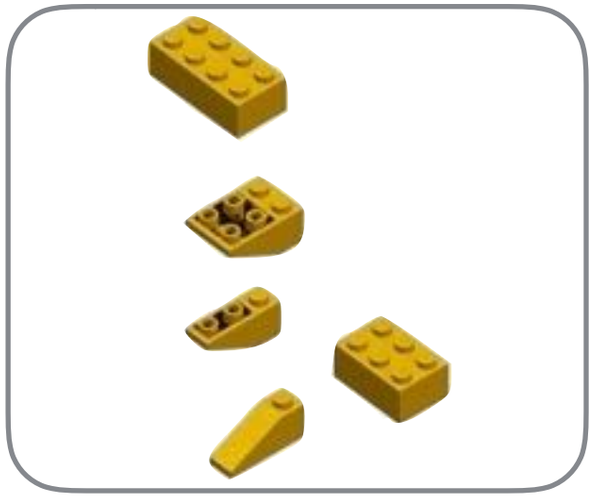


IMP kernel

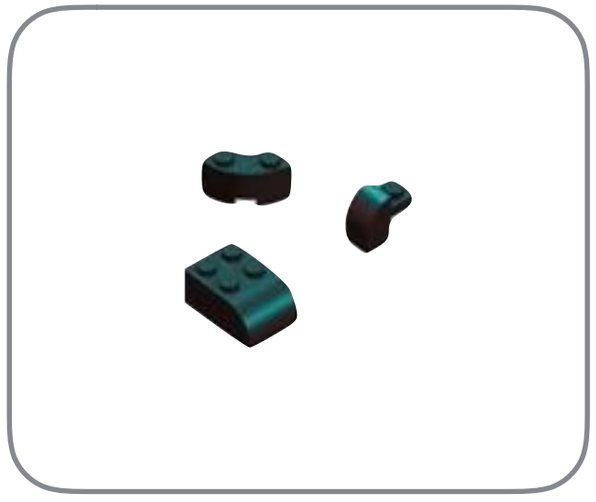


IMP.em

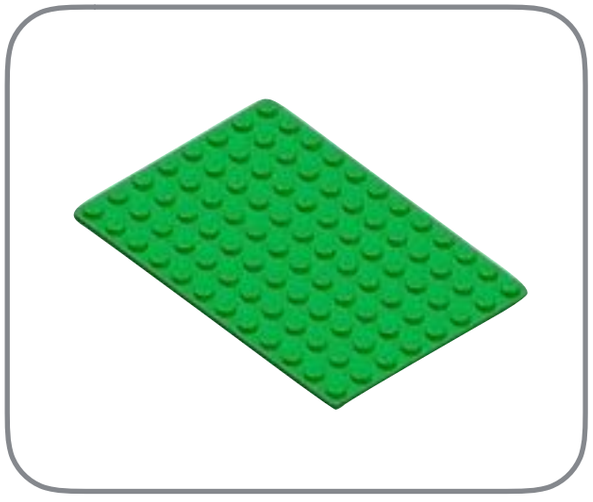
- Split into modules
- Distinct functionality
- Developed separately
- Licensed differently
- Stable interfaces



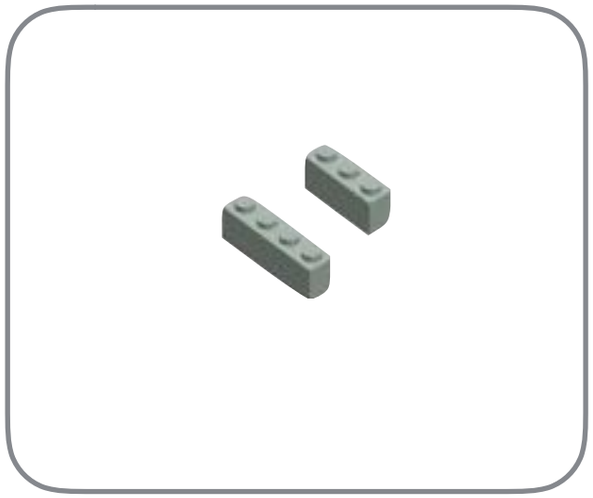
IMP.saxs



IMP.algebra

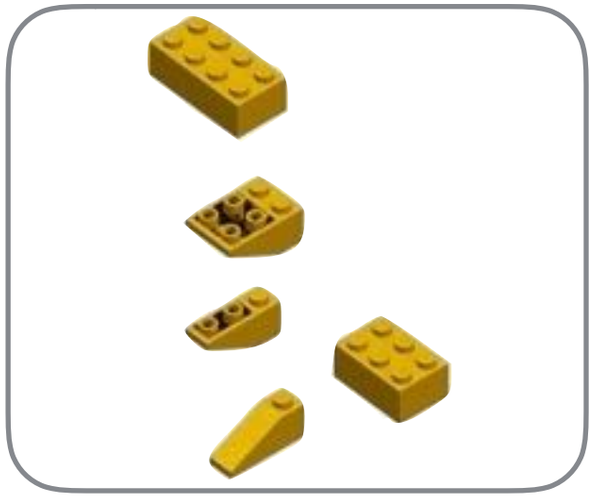


IMP kernel

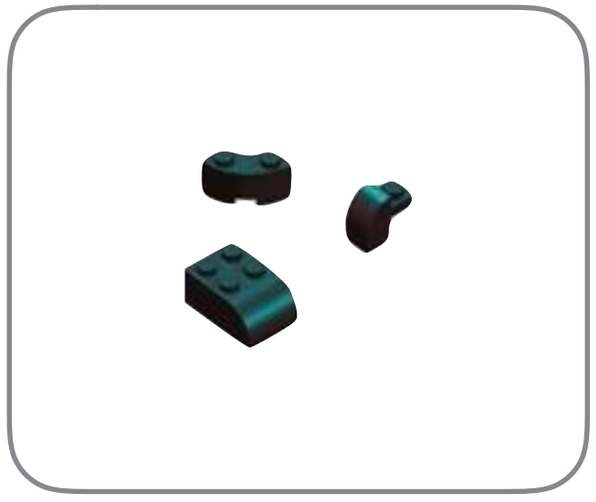


IMP.em

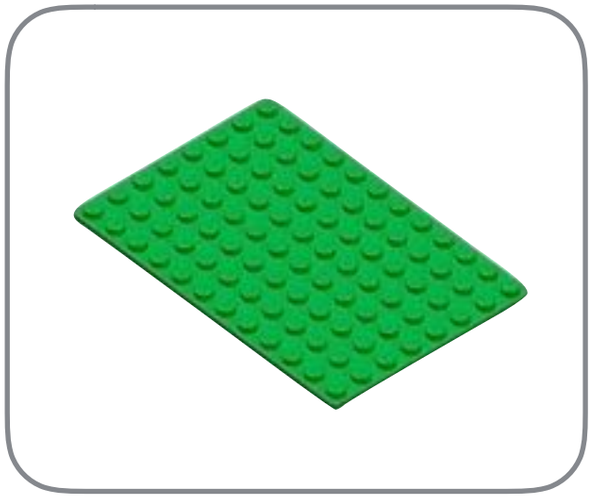
- Split into modules
- Distinct functionality
- Developed separately
- Licensed differently
- Stable interfaces



IMP.saxs

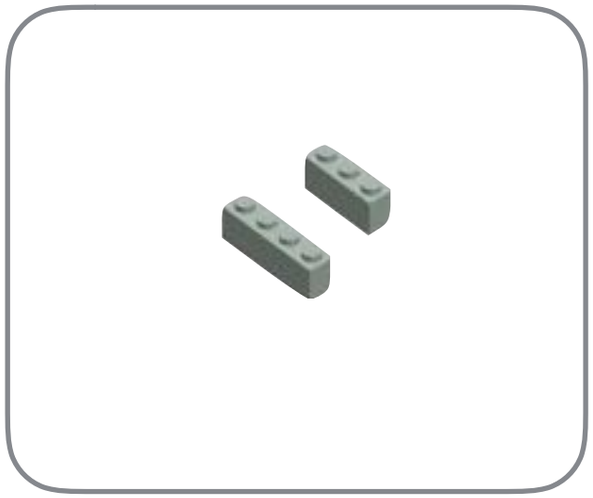


IMP.algebra



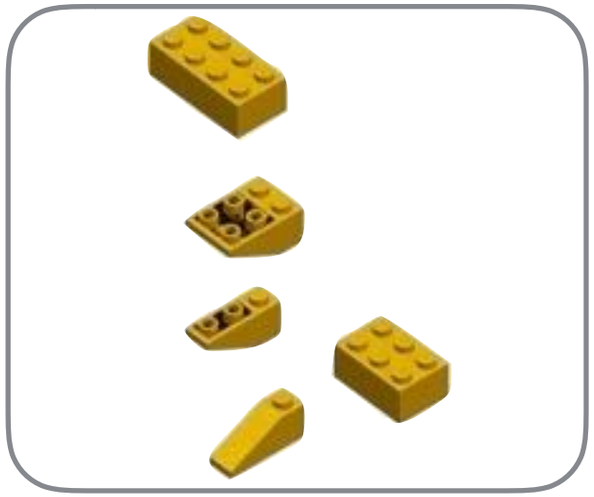
IMP kernel

Common functionality

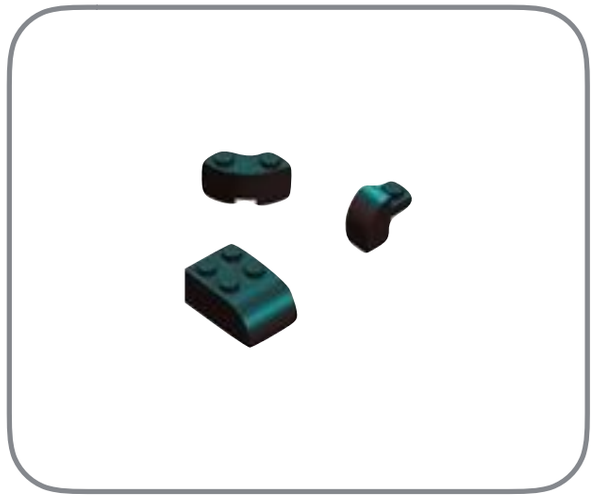


IMP.em

- Split into modules
- Distinct functionality
- Developed separately
- Licensed differently
- Stable interfaces

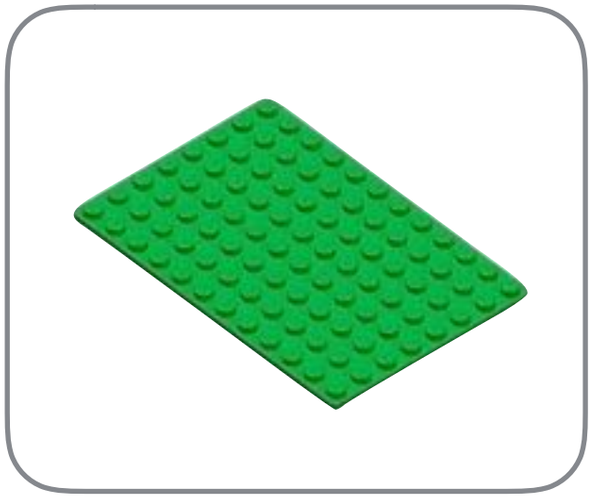


IMP.saxs



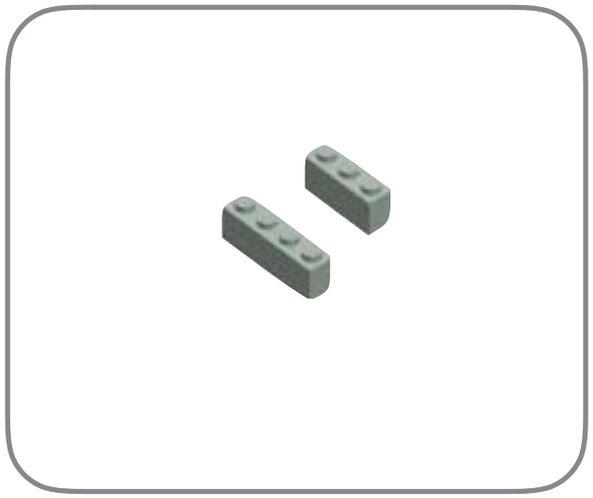
IMP.algebra

Geometry, primitive shapes



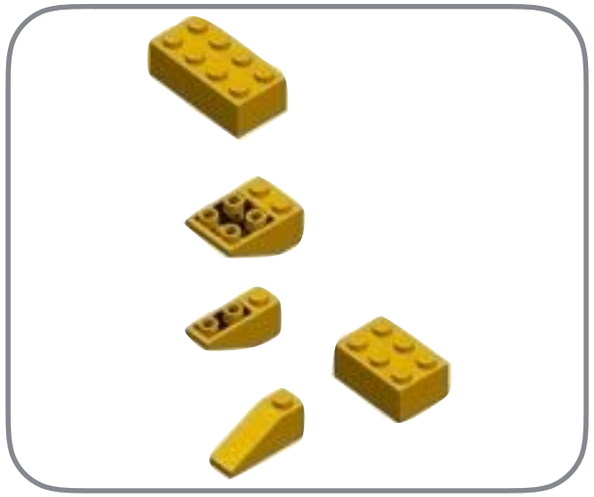
IMP kernel

Common functionality



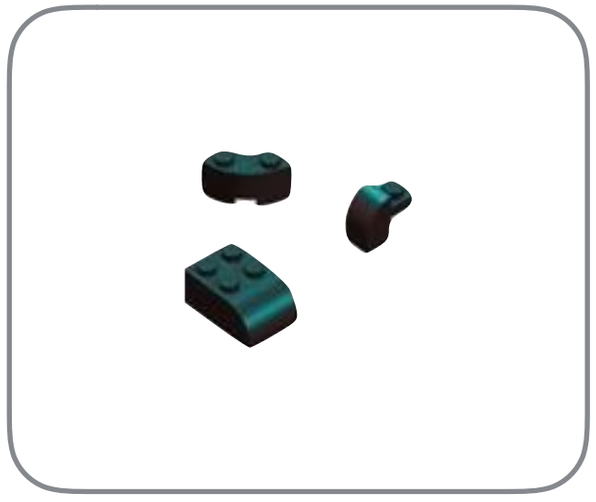
IMP.em

- Split into modules
- Distinct functionality
- Developed separately
- Licensed differently
- Stable interfaces



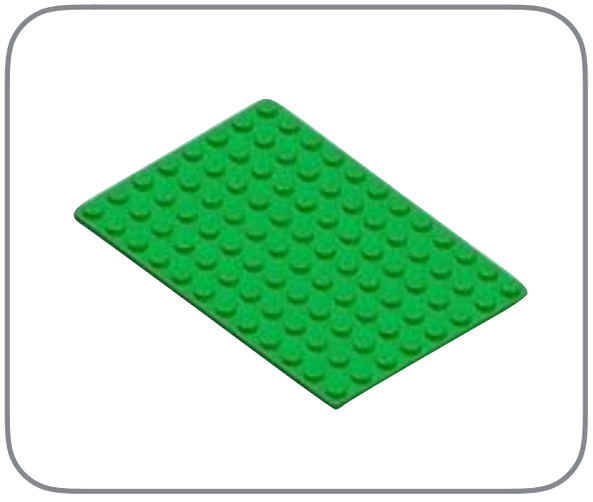
IMP.saxs

Handling of Small Angle X-ray (SAXS) data



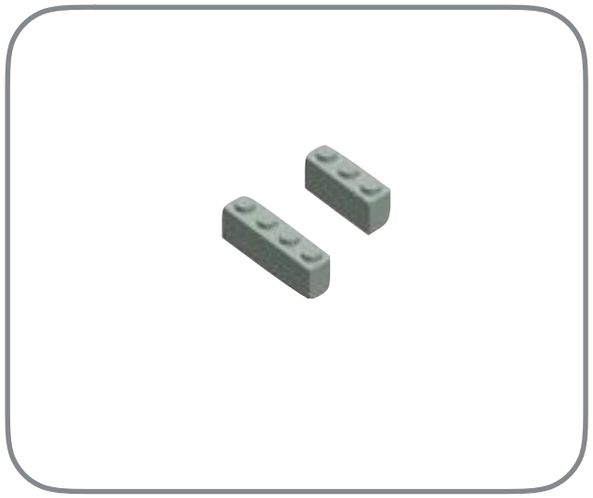
IMP.algebra

Geometry, primitive shapes



IMP kernel

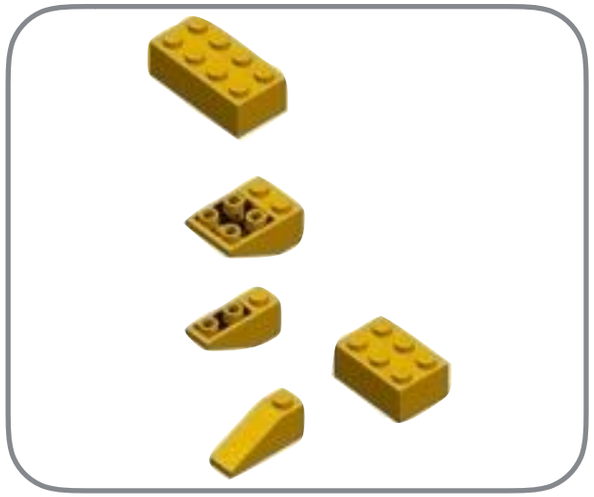
Common functionality



IMP.em

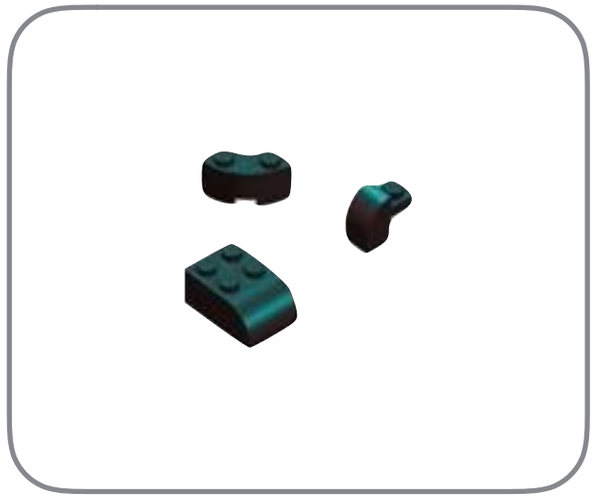
Handling of electron microscopy (EM) experimental data

- Split into modules
- Distinct functionality
- Developed separately
- Licensed differently
- Stable interfaces



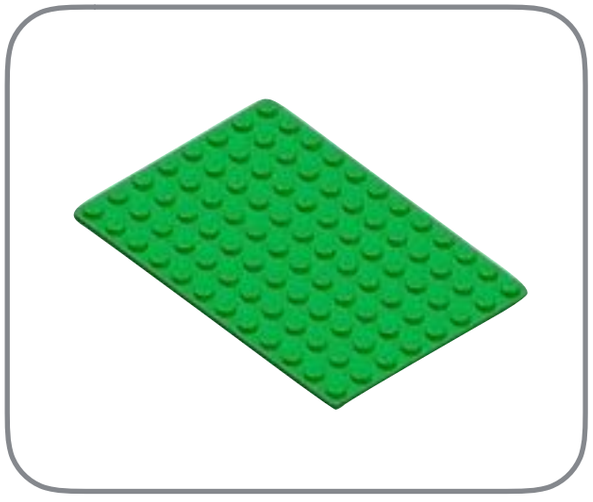
IMP.saxs

Handling of Small Angle X-ray (SAXS) data



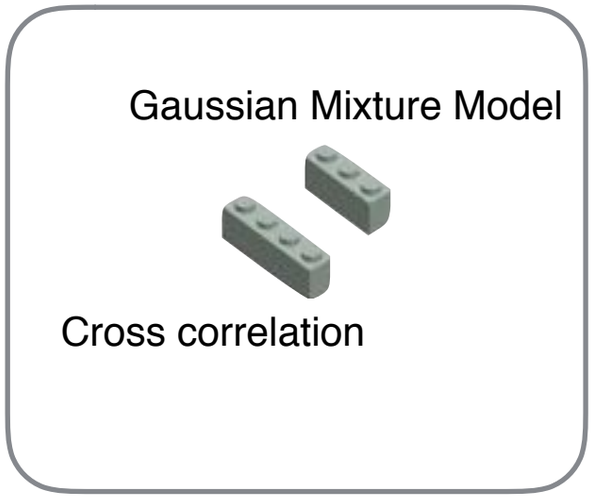
IMP.algebra

Geometry, primitive shapes



IMP kernel

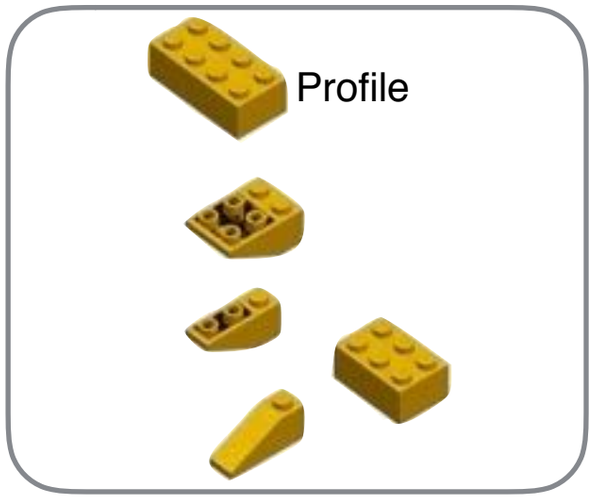
Common functionality



IMP.em

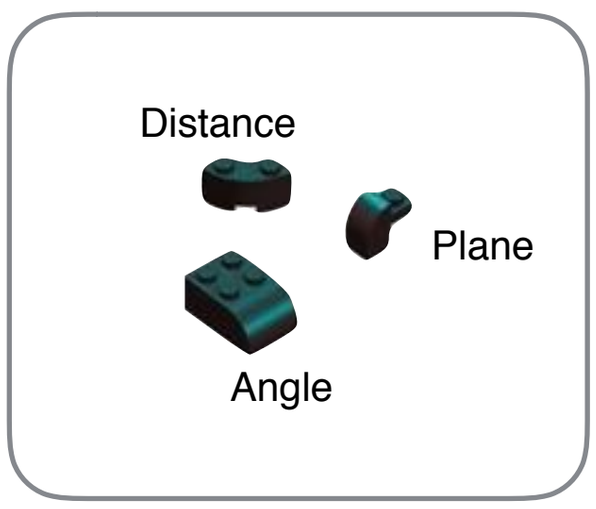
Handling of electron microscopy (EM) experimental data

- Split into modules
- Distinct functionality
- Developed separately
- Licensed differently
- Stable interfaces



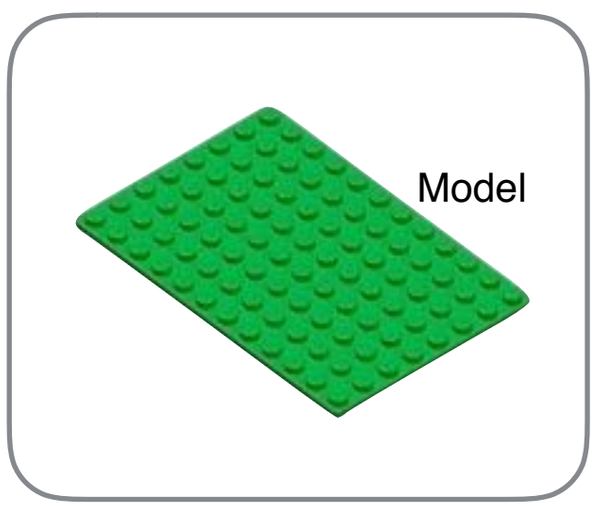
IMP.saxs

Handling of Small Angle X-ray (SAXS) data



IMP.algebra

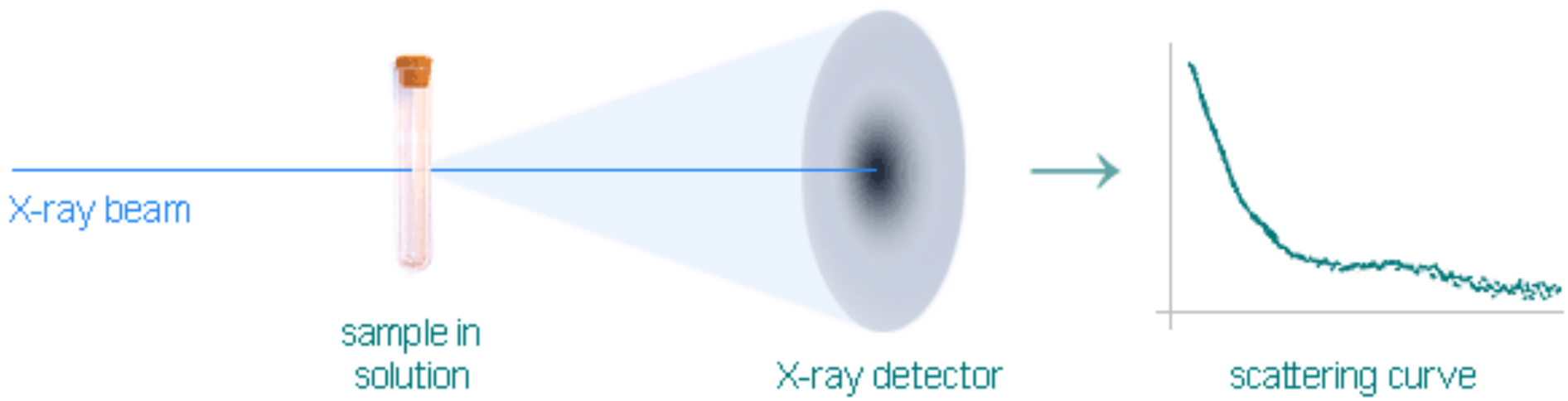
Geometry, primitive shapes



IMP kernel

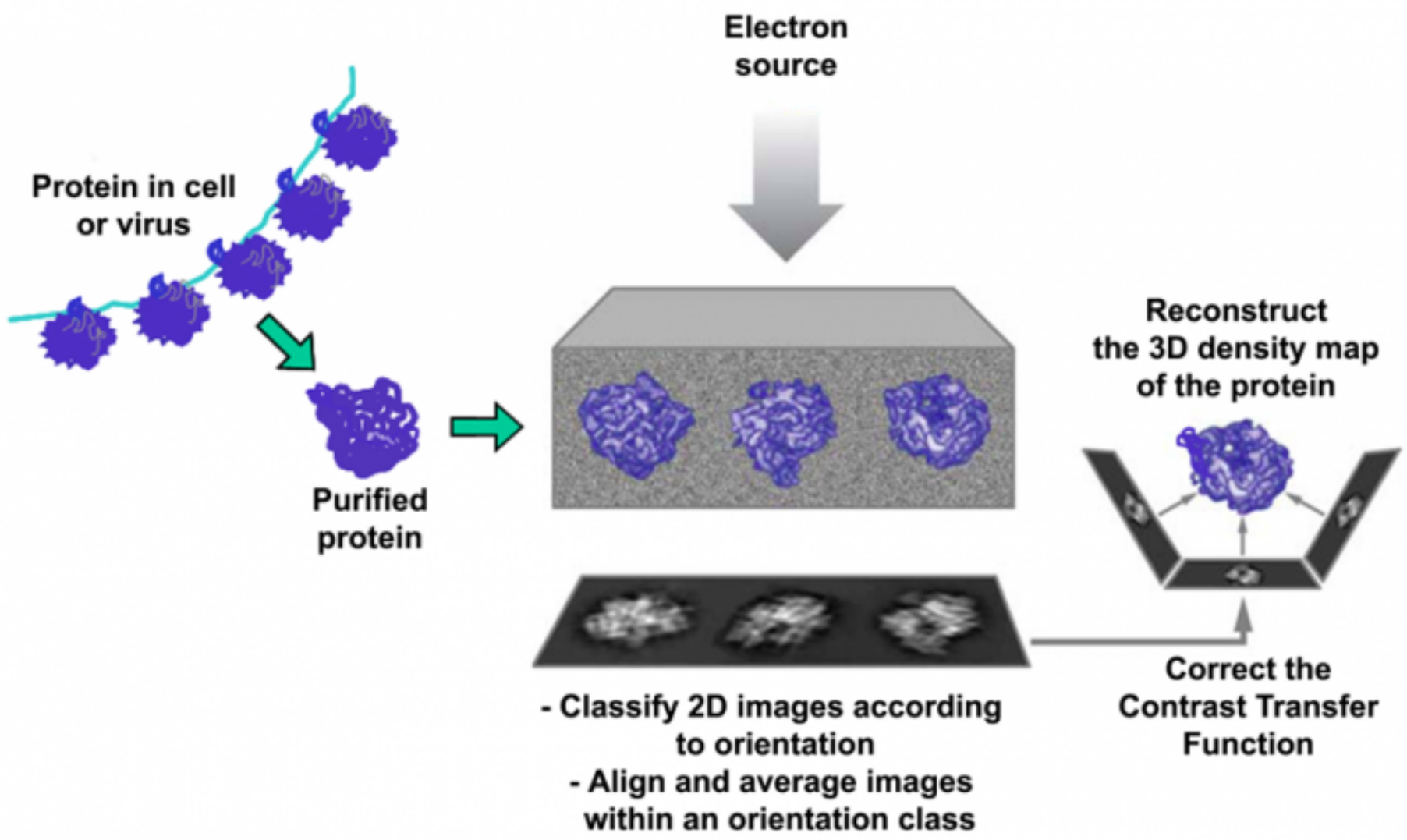
Common functionality

SAXS



- Sample is in solution
  - Pro: closer to its *in vivo* state
  - Con: rotationally averaged

EM



- Significant processing required to generate a 3D image

Link via Python to other packages (via standard interfaces) to avoid code duplication…

Link via Python to other packages (via standard interfaces) to avoid code duplication…

MODELLER

comparative modeling

Link via Python to other packages (via standard interfaces) to avoid code duplication…

MODELLER

comparative modeling

BioPython

handling of sequence data

Link via Python to other packages (via standard interfaces) to avoid code duplication…

MODELLER

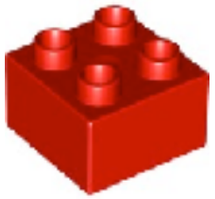
comparative modeling

BioPython

handling of sequence data

Chimera/VMD

visualization



Link via Python to other packages (via standard interfaces) to avoid code duplication…

MODELLER

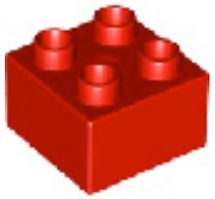
comparative modeling

BioPython

handling of sequence data

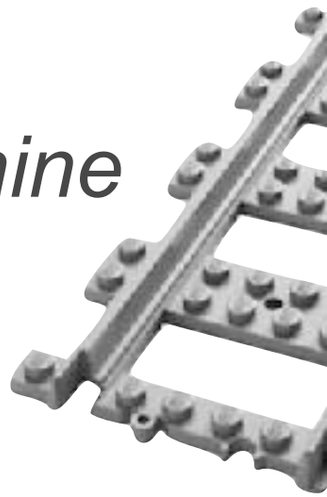
Chimera/VMD

visualization



scikit-learn

clustering, machine learning



Link via Python to other packages (via standard interfaces) to avoid code duplication…

MODELLER

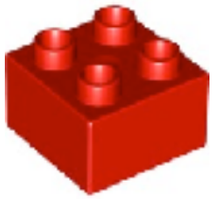
comparative modeling

BioPython

handling of sequence data

Chimera/VMD

visualization

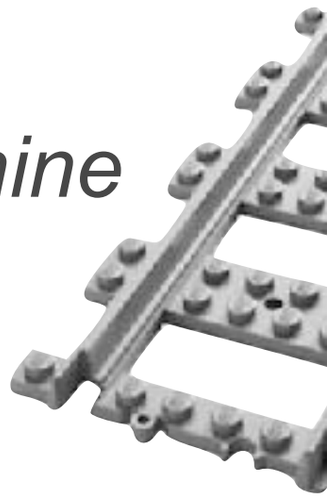


numpy/scipy

matrix/linear algebra

scikit-learn

clustering, machine learning



Link via Python to other packages (via standard interfaces) to avoid code duplication…

MODELLER

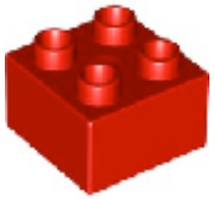
comparative modeling

BioPython

handling of sequence data

Chimera/VMD

visualization

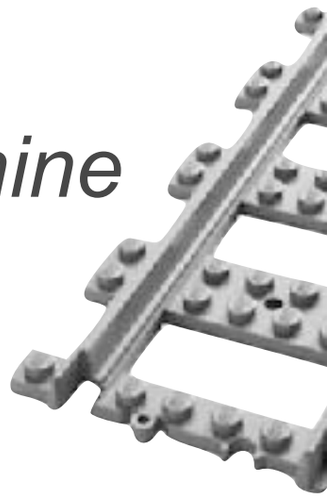


numpy/scipy

matrix/linear algebra

scikit-learn

clustering, machine learning



etc.

Integrative Modeling Platform (IMP)

https://integrativemodeling.org

- Each ‘piece’ is a Python class
- Most classes actually ‘wrap’ an underlying class in C++
  - C++ for speed, Python for flexibility
- Each module is a Python module, and C++ namespace
- IMP is usually used from Python, by writing a script
- A protocol is one or more Python scripts plus the input data

Example Python script

```
import IMP
import IMP.algebra
import IMP.core

m = IMP.Model()
# Create two "untyped" Particles
p1 = IMP.Particle(m)
p2 = IMP.Particle(m)

# "Decorate" the Particles with x,y,z attributes (point-like particles)
d1 = IMP.core.XYZ.setup_particle(p1)
d2 = IMP.core.XYZ.setup_particle(p2)

# Use some XYZ-specific functionality (set coordinates)
d1.set_coordinates(IMP.algebra.Vector3D(10.0, 10.0, 10.0))
d2.set_coordinates(IMP.algebra.Vector3D(-10.0, -10.0, -10.0))
print(d1, d2)

# Harmonically restrain p1 to be zero distance from the origin
f = IMP.core.Harmonic(0.0, 1.0)
s = IMP.core.DistanceToSingletonScore(f, IMP.algebra.Vector3D(0., 0., 0.))
r1 = IMP.core.SingletonRestraint(s, p1)

# Harmonically restrain p1 and p2 to be distance 5.0 apart
f = IMP.core.Harmonic(5.0, 1.0)
s = IMP.core.DistancePairScore(f)
r2 = IMP.core.PairRestraint(s, (p1, p2))

# Optimize the x,y,z coordinates of both particles with conjugate gradients
sf = IMP.core.RestraintsScoringFunction([r1, r2], "scoring function")
d1.set_coordinates_are_optimized(True)
d2.set_coordinates_are_optimized(True)
o = IMP.core.ConjugateGradients(m)
o.set_scoring_function(sf)
o.optimize(50)
print(d1, d2)
```

```
import IMP
import IMP.algebra
import IMP.core
```

Make IMP classes in the IMP kernel (‘IMP’) and IMP.algebra and IMP.core modules available

```
m = IMP.Model()
# Create two "untyped" Particles
p1 = IMP.Particle(m)
p2 = IMP.Particle(m)
```

- Create a new Model object (an *instance* of the Model *class*) and assign it to the variable ‘m’
  - An IMP Model is a container that holds knowledge of the entire system
- Create two Particles called ‘p1’ and ‘p2’
  - A Particle is an abstract data container and can hold any number of attribute:value pairs, e.g.
    - xyz coordinates
    - mass
    - radius
    - pointers to other Particles, to represent a bond (two other particles), or hierarchy (parents, children)
    - element, residue/atom name, etc.

```
# "Decorate" the Particles with x,y,z attributes (point-like
particles)
d1 = IMP.core.XYZ.setup_particle(p1)
d2 = IMP.core.XYZ.setup_particle(p2)

# Use some XYZ-specific functionality (set coordinates)
d1.set_coordinates(IMP.algebra.Vector3D(10.0, 10.0, 10.0))
d2.set_coordinates(IMP.algebra.Vector3D(-10.0, -10.0, -10.0))
print(d1, d2)
```

- A *decorator* lets us use a specific set of functionality on a Particle
  - ‘d1’ refers to the same underlying object as ‘p1’ but acts like a 3D point (IMP.core.XYZ class)
- set_coordinates() is a *method* of the XYZ class
  - IMP.algebra.Vector3D represents a 3D vector or coordinate

```
# Harmonically restrain p1 to be zero distance from the origin
f = IMP.core.Harmonic(0.0, 1.0)
s = IMP.core.DistanceToSingletonScore(f, IMP.algebra.Vector3D(0., 0., 0.))
r1 = IMP.core.SingletonRestraint(s, p1)
```

- A Restraint is a term in our scoring function
- IMP.core.SingletonRestraint applies a Score to a single particle (p1 in this case)
- In turn, DistanceToSingletonScore calculates the Cartesian distance between a fixed point and p1, then uses a unary function to weight that distance
- Harmonic is a unary function that applies a simple harmonic spring
- In this way, we can very flexibly build our scoring function from basic building blocks

```
# Harmonically restrain p1 and p2 to be distance 5.0 apart
f = IMP.core.Harmonic(5.0, 1.0)
s = IMP.core.DistancePairScore(f)
r2 = IMP.core.PairRestraint(s, (p1, p2))
```

- Similarly, we make another Restraint called ‘r2’ that restrains the distance between two particles
- Usually distances are considered to be angstroms but this isn’t required or enforced

```
# Optimize the x,y,z coordinates of both particles with conjugate gradients
sf = IMP.core.RestraintsScoringFunction([r1, r2], "scoring function")
d1.set_coordinates_are_optimized(True)
d2.set_coordinates_are_optimized(True)
o = IMP.core.ConjugateGradients(m)
o.set_scoring_function(sf)
o.optimize(50)
print(d1, d2)
```

- Finally, we make a simple scoring function ‘sf’ that’s just the sum of the two harmonic restraints
- We find the minimum of the function using up to 50 steps of conjugate gradients
    - At each step the algorithm will try to reduce the value of the scoring function by changing the coordinates of d1 and/or d2

Example Python script

```python
import IMP
import IMP.algebra
import IMP.core

m = IMP.Model()
# Create two "untyped" Particles
p1 = IMP.Particle(m)
p2 = IMP.Particle(m)

# "Decorate" the Particles with x,y,z attributes (point-like particles)
d1 = IMP.core.XYZ.setup_particle(p1)
d2 = IMP.core.XYZ.setup_particle(p2)

# Use some XYZ-specific functionality (set coordinates)
d1.set_coordinates(IMP.algebra.Vector3D(10.0, 10.0, 10.0))
d2.set_coordinates(IMP.algebra.Vector3D(-10.0, -10.0, -10.0))
print(d1, d2)

# Harmonically restrain p1 to be zero distance from the origin
f = IMP.core.Harmonic(0.0, 1.0)
s = IMP.core.DistanceToSingletonScore(f, IMP.algebra.Vector3D(0., 0., 0.))
r1 = IMP.core.SingletonRestraint(s, p1)

# Harmonically restrain p1 and p2 to be distance 5.0 apart
f = IMP.core.Harmonic(5.0, 1.0)
s = IMP.core.DistancePairScore(f)
r2 = IMP.core.PairRestraint(s, (p1, p2))

# Optimize the x,y,z coordinates of both particles with conjugate gradients
sf = IMP.core.RestraintsScoringFunction([r1, r2], "scoring function")
d1.set_coordinates_are_optimized(True)
d2.set_coordinates_are_optimized(True)
o = IMP.core.ConjugateGradients(m)
o.set_scoring_function(sf)
o.optimize(50)
print(d1, d2)
```

Example Python script

```
import IMP
import IMP.algebra
import IMP.core

m = IMP.Model()
# Create two "untyped" Particles
p1 = IMP.Particle(m)
p2 = IMP.Particle(m)

# "Decorate" the Particles with x,y,z attributes (point-like particles)
d1 = IMP.core.XYZ.setup_particle(p1)
d2 = IMP.core.XYZ.setup_particle(p2)

# Use some XYZ-specific functionality (set coordinates)
d1.set_coordinates(IMP.algebra.Vector3D(10.0, 10.0, 10.0))
d2.set_coordinates(IMP.algebra.Vector3D(-10.0, -10.0, -10.0))
print(d1, d2)

# Harmonically restrain p1 to be zero distance from the origin
f = IMP.core.Harmonic(0.0, 1.0)
s = IMP.core.DistanceToSingletonScore(f, IMP.algebra.Vector3D(0., 0., 0.))
r1 = IMP.core.SingletonRestraint(s, p1)

# Harmonically restrain p1 and p2 to be distance 5.0 apart
f = IMP.core.Harmonic(5.0, 1.0)
s = IMP.core.DistancePairScore(f)
r2 = IMP.core.PairRestraint(s, (p1, p2))

# Optimize the x,y,z coordinates of both particles with conjugate gradients
sf = IMP.core.RestraintsScoringFunction([r1, r2], "scoring function")
d1.set_coordinates_are_optimized(True)
d2.set_coordinates_are_optimized(True)
o = IMP.core.ConjugateGradients(m)
o.set_scoring_function(sf)
o.optimize(50)
print(d1, d2)
```

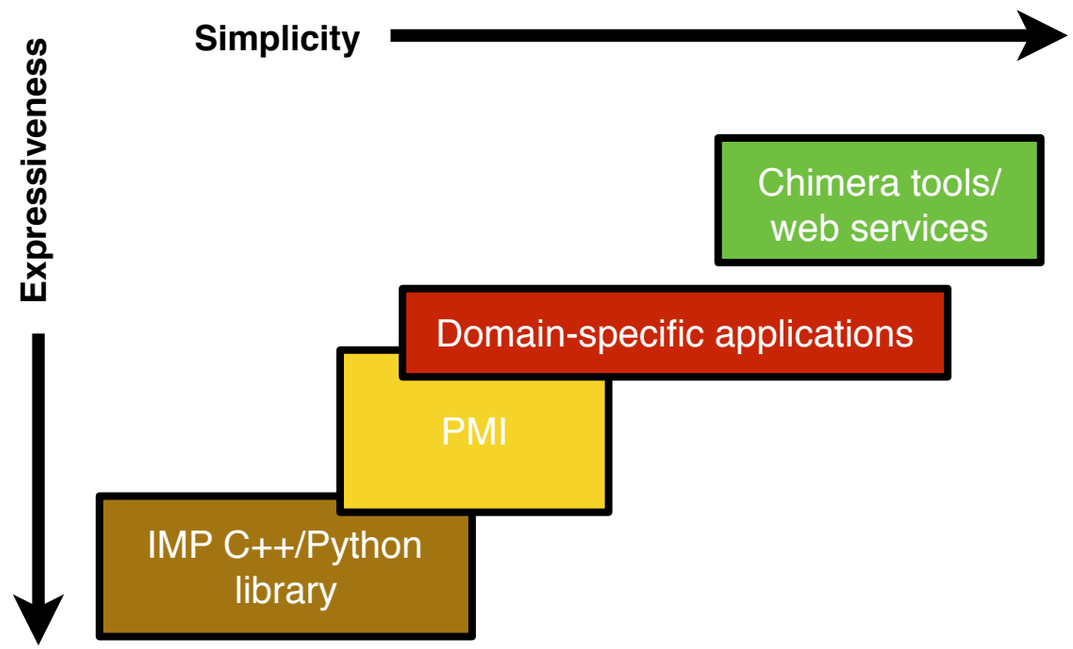
- So let’s run it…

Example Python script

```
import IMP
import IMP.algebra
import IMP.core

m = IMP.Model()
# Create two "untyped" Particles
p1 = IMP.Particle(m)
p2 = IMP.Particle(m)

# "Decorate" the Particles with x,y,z attributes (point-like particles)
d1 = IMP.core.XYZ.setup_particle(p1)
d2 = IMP.core.XYZ.setup_particle(p2)

# Use some XYZ-specific functionality (set coordinates)
d1.set_coordinates(IMP.algebra.Vector3D(10.0, 10.0, 10.0))
d2.set_coordinates(IMP.algebra.Vector3D(-10.0, -10.0, -10.0))
print(d1, d2)

# Harmonically restrain p1 to be zero distance from the origin
f = IMP.core.Harmonic(0.0, 1.0)
s = IMP.core.DistanceToSingletonScore(f, IMP.algebra.Vector3D(0., 0., 0.))
r1 = IMP.core.SingletonRestraint(s, p1)

# Harmonically restrain p1 and p2 to be distance 5.0 apart
f = IMP.core.Harmonic(5.0, 1.0)
s = IMP.core.DistancePairScore(f)
r2 = IMP.core.PairRestraint(s, (p1, p2))

# Optimize the x,y,z coordinates of both particles with conjugate gradients
sf = IMP.core.RestraintsScoringFunction([r1, r2], "scoring function")
d1.set_coordinates_are_optimized(True)
d2.set_coordinates_are_optimized(True)
o = IMP.core.ConjugateGradients(m)
o.set_scoring_function(sf)
o.optimize(50)
print(d1, d2)
```

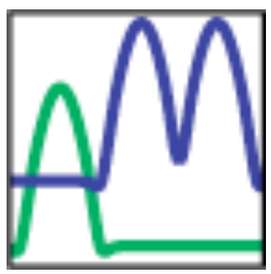
- So let's run it…
- Very flexible, but all we've done here is move two points!

Higher level interfaces

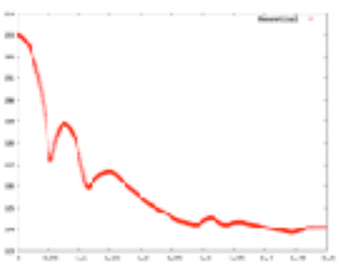


- In practice, scripts for “real” modeling problems would be too long and unwieldy to write this way
- Most usage of IMP is via simpler (but less ‘expressive’) interfaces

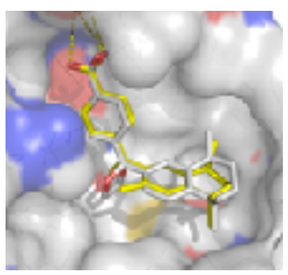
Chimera tools/
web services



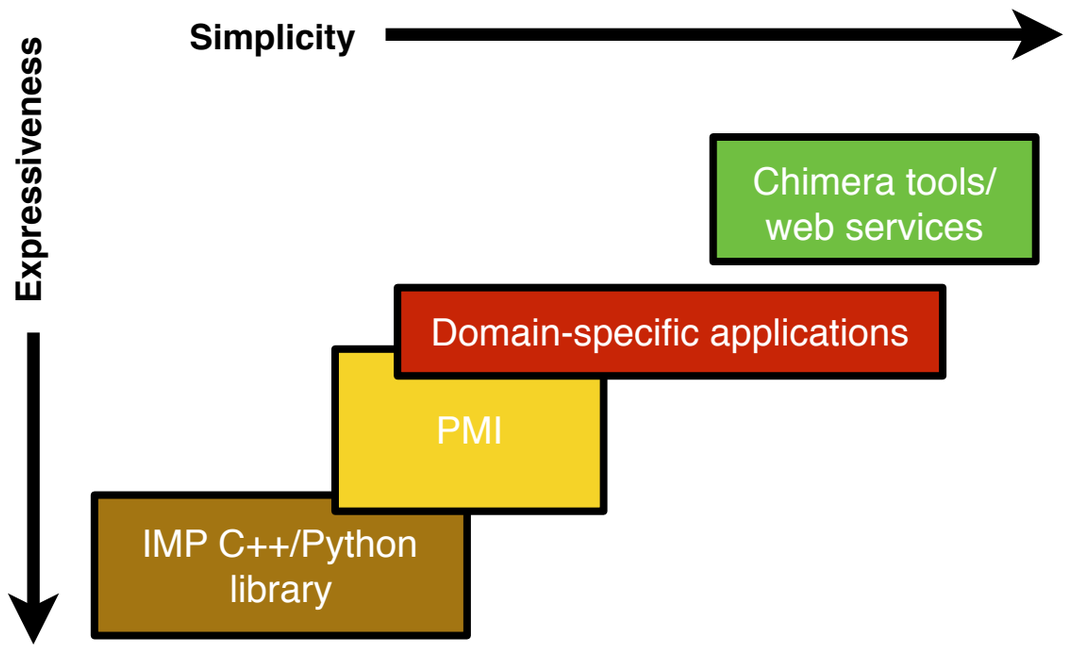
- AllosMod: modeling of ligand-induced protein dynamics, allostery



- FoXS: fast SAXS profile computation with Debye formula
- FoXSDock: macromolecular docking with SAXS Profile
- SAXSMerge: automated statistical method to merge SAXS profiles from different concentrations and exposure times

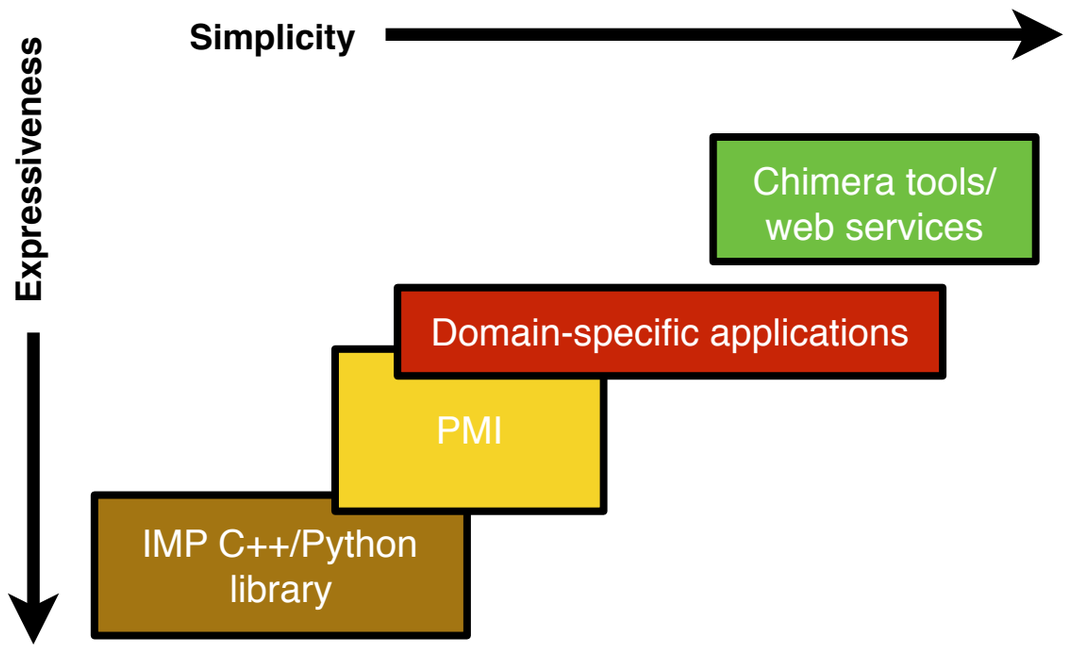


- Pose&Rank: scoring of protein-ligand complexes


Simplicity
Expressiveness
Chimera tools/
web services
Domain-specific applications
PMI
IMP C++/Python
library

Domain-specific applications

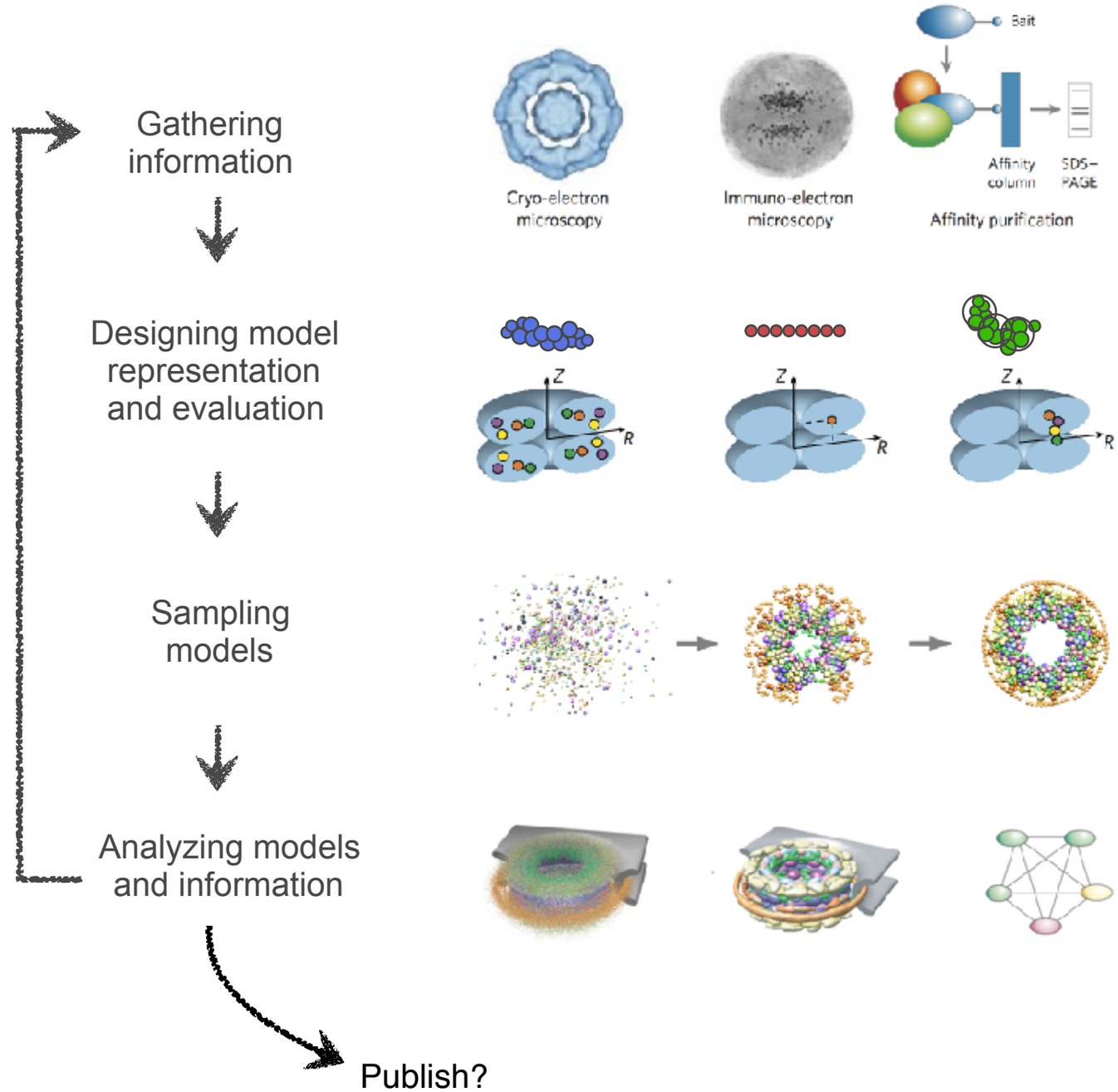
- Command line tools
- Generally, similar functionality to web services, but running locally


Simplicity
Expressiveness
Chimera tools/
web services
Domain-specific applications
PMI
IMP C++/Python
library

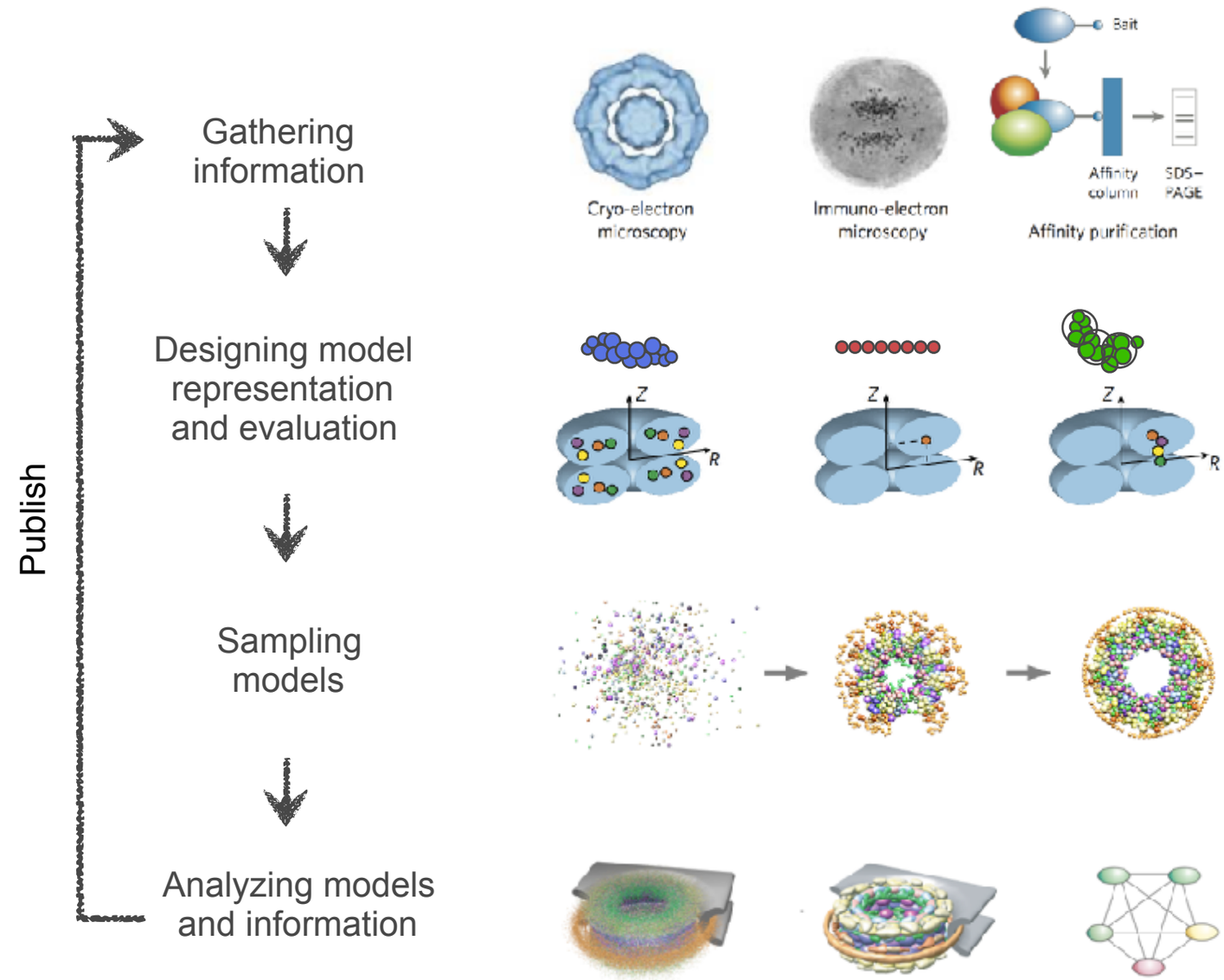
PMI

- Just another IMP module (IMP.pmi)
- A meta language for modeling
- We still write Python scripts, but…
  - Many protocols (e.g. replica exchange) already packaged up nicely for us
  - Refer to biological units rather than individual particles
  - Publication-ready plots are more or less automatic
- Regular IMP objects are constructed, so an advanced user can always customize things using the full collection of IMP classes if PMI is insufficient
- Today we will use PMI to model the stalk of the RNA Polymerase II complex: https://github.com/salilab/imp_tutorial/

Reproducibility/Deposition

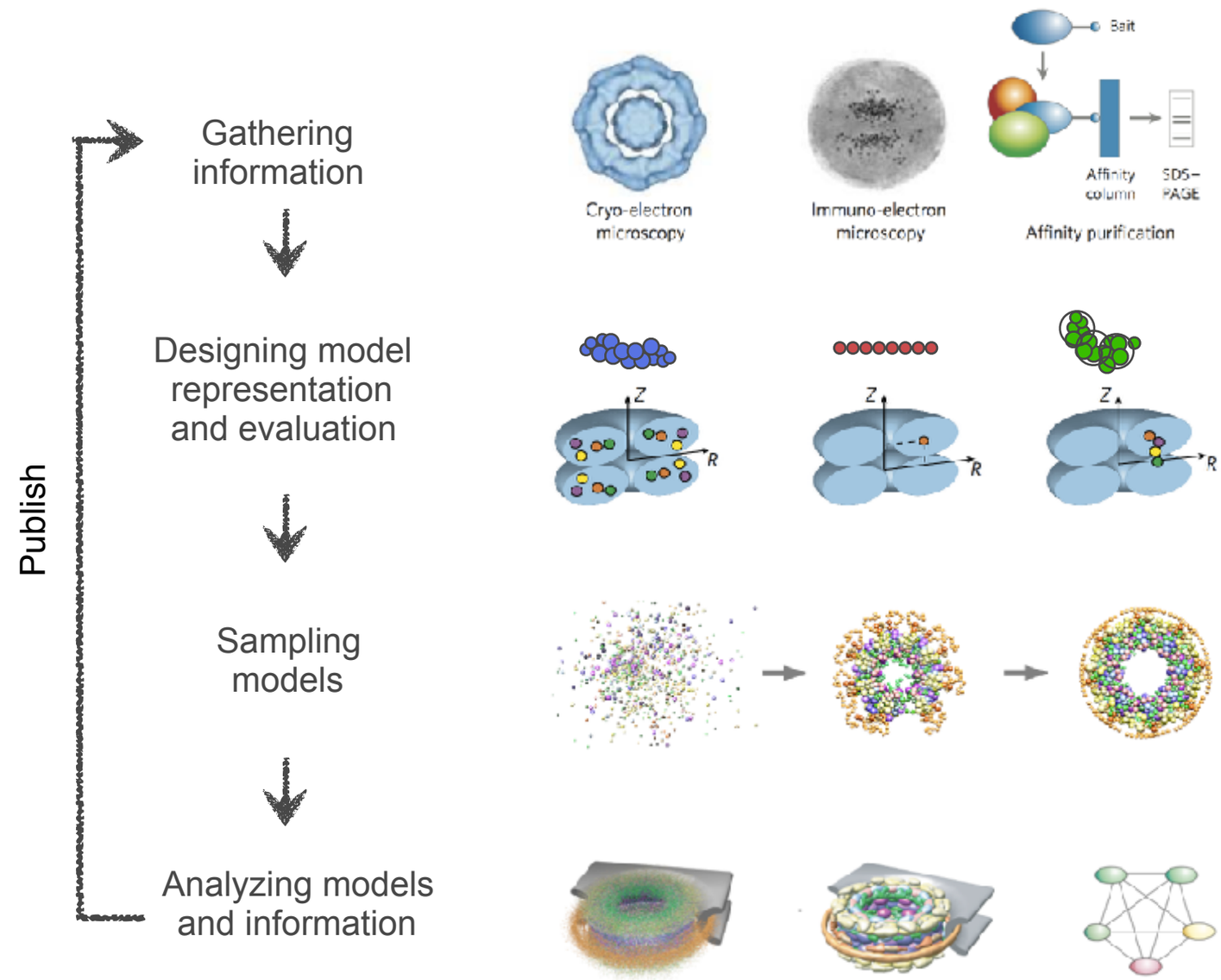


Reproducibility/Deposition



Reproducibility/Deposition

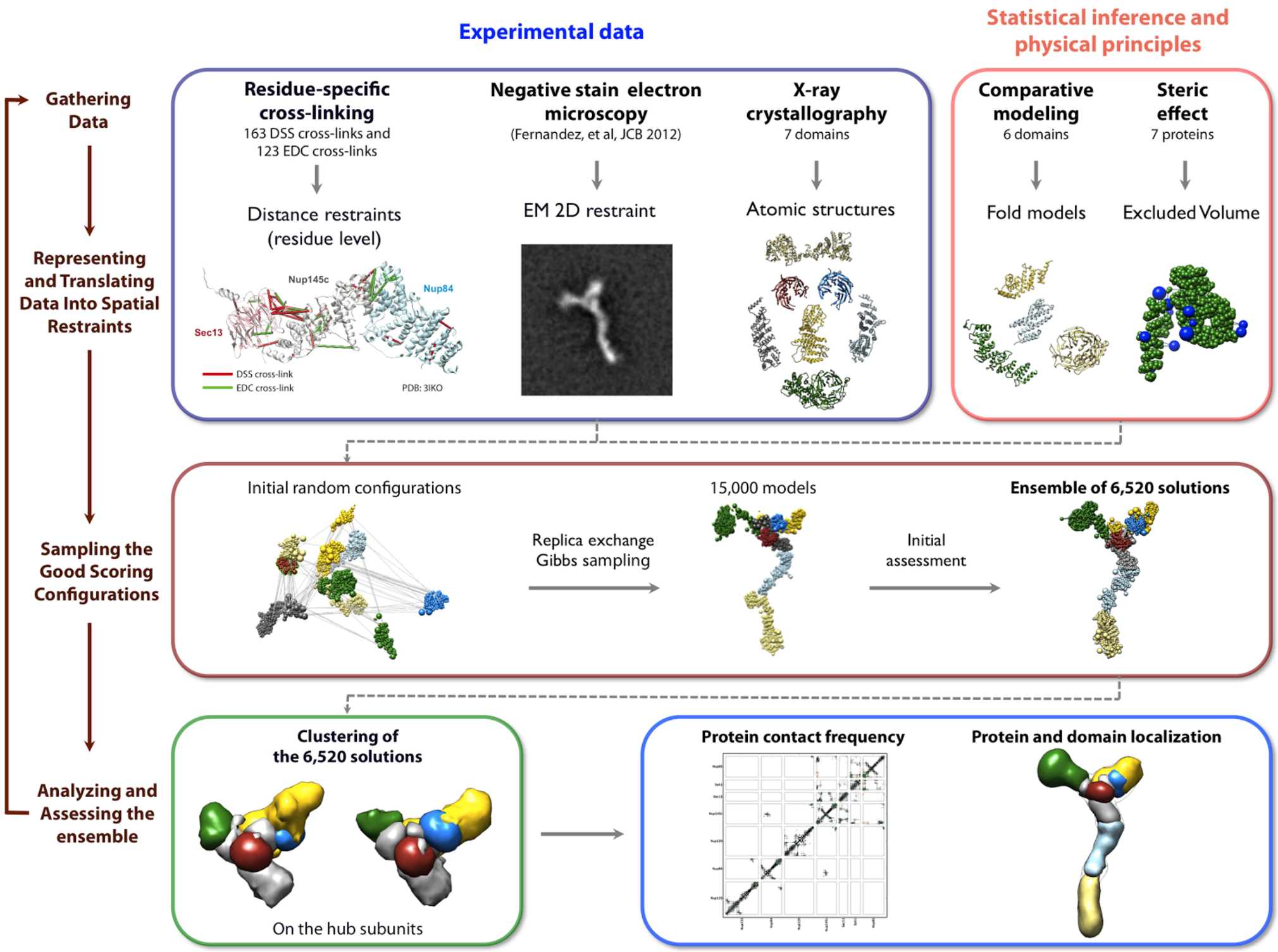
- Store protocol in GitHub so others can run it, improve it, modify it
- Document!
- Automated tests
- DOI (Zenodo, Figshare)



https://pdb-dev.rcsb.rutgers.edu/

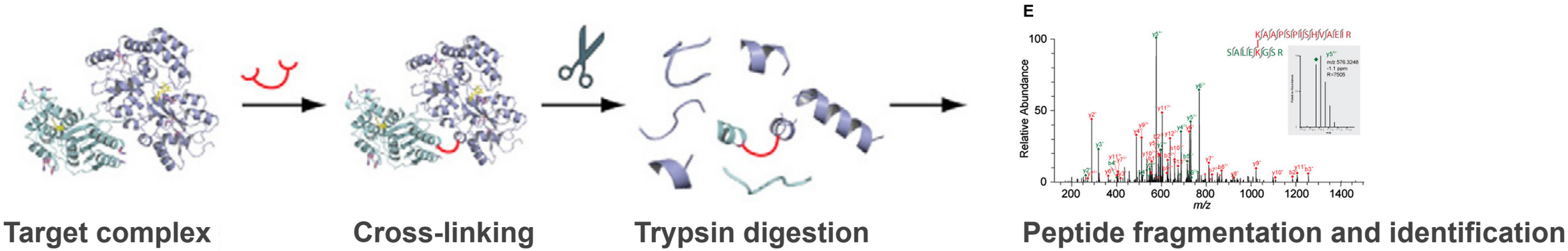
https://integrativemodeling.org/systems/

Example: Nup84



Mol Cell Proteomics 13, 2927-2943, 2014

Cross-linking coupled with mass spectrometry (CX-MS)



Output, essentially, is a list of proximal residue pairs (again, after processing)

Note: spectra can identify multiple cross-links (ambiguity)

Installation

- We need installed
  - **numpy** and **scipy** for matrix and linear algebra
  - **scikit-learn** for k-means clustering
  - **matplotlib** for plotting results
  - **Chimera** for visualization of results
  - **IMP** itself

- Easiest way is to install Anaconda Python, then run:

```
conda config --add channels salilab
conda install imp numpy scipy scikit-learn matplotlib
```

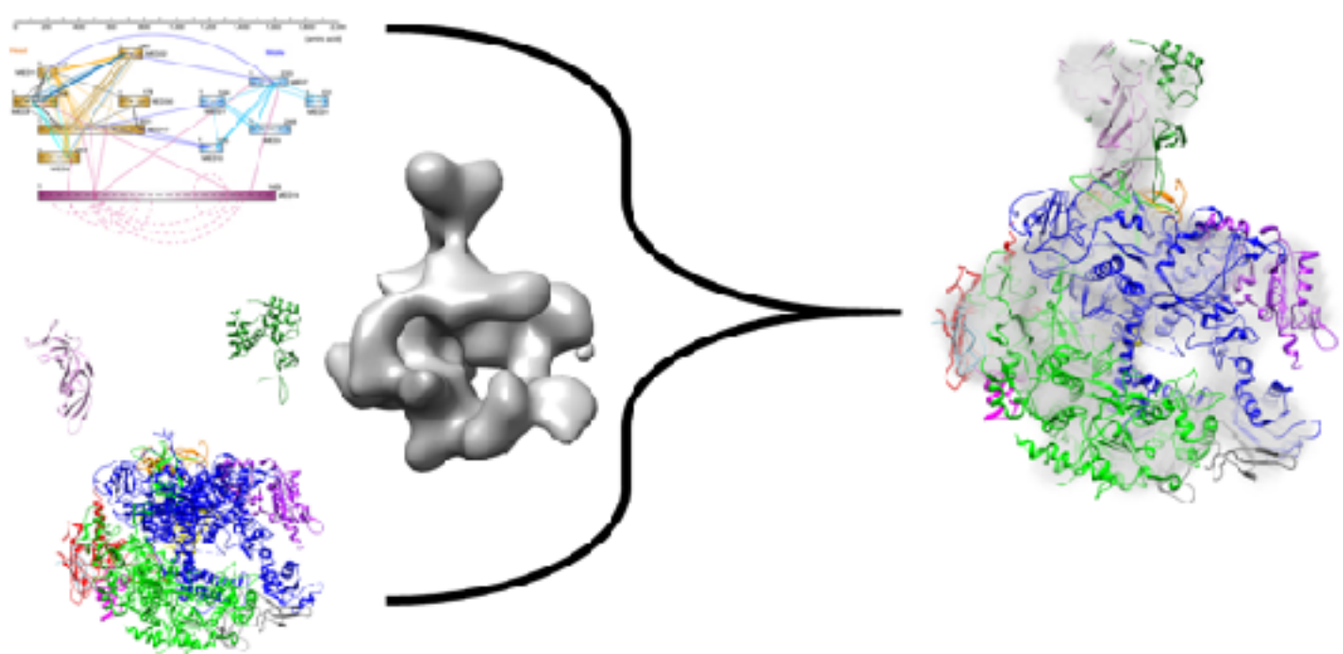
- Get the tutorial files from GitHub: https://github.com/salilab/imp_tutorial/

Integrative Structure Modeling of RNA Polymerase II stalk

- RNA Pol II is a eukaryotic complex that catalyzes DNA transcription to synthesize mRNA strands.
- Eukaryotic RNA polymerase II contains 12 subunits, Rpb1 to Rpb12.
- The yeast RNA Pol II dissociates into a 10-subunit core and a Rpb4/Rpb7 heterodimer.
- Rpb4 and Rpb7 are conserved from yeast to humans, and form a stalk-like protrusion extending from the main body of the RNA Pol II complex.

Integrative Structure Modeling of RNA Polymerase II stalk

- We want to determine the localization of two subunits of the yeast RNA Polymerase II, Rpb4 and Rpb7 (stalk), hypothesizing that we know already the structure of the remaining 10-subunit complex.
- This example utilizes:
  - chemical cross-linking coupled with mass spectrometry (CX-MS),
  - negative-stain electron microscopy (EM),
  - x-ray crystallography data



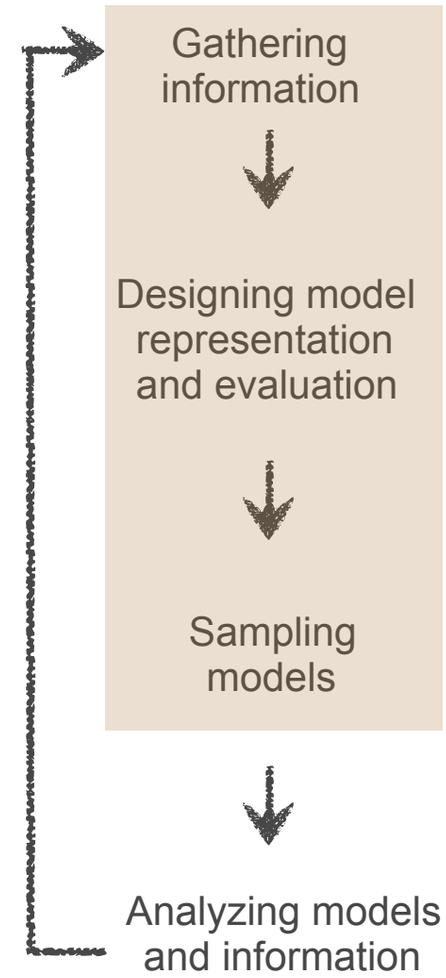
Get tutorial files from GitHub

- Let's get started by getting the main modeling script running while we look at what it's doing:

```
cd imp_tutorial/rnapolii/modeling
python modeling.py --test
```

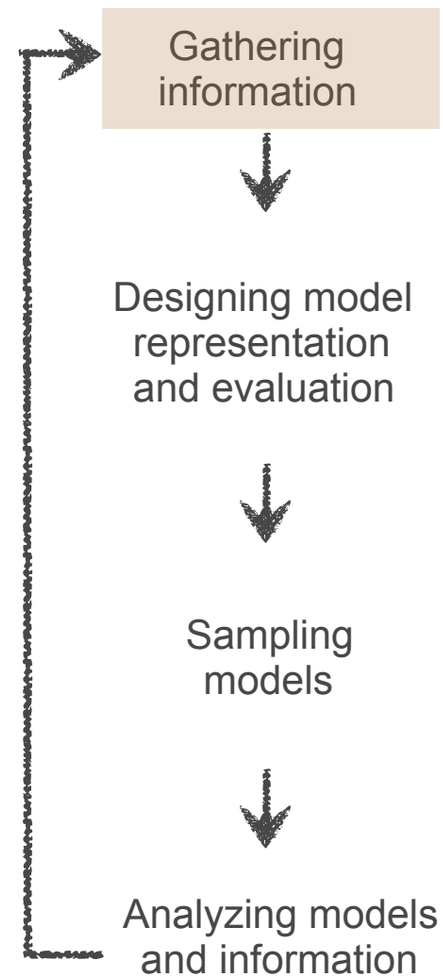
- "Real" modeling will take hours, so we're running in 'test' mode which generates only 50 frames (rather than 20,000)

- The script covers the first 3 steps of integrative modeling



Data for yeast RNA Polymerase II

- The `rnapolii/data` folder contains:
  - Sequence information (FASTA files for each subunit)
  - Electron density maps (.mrc, .txt files)
  - Structure from x-ray crystallography (PDB file)
  - Chemical crosslinking datasets (two data sets, one from Al Burlingame's lab, and another from Juri Rappsilber's lab)

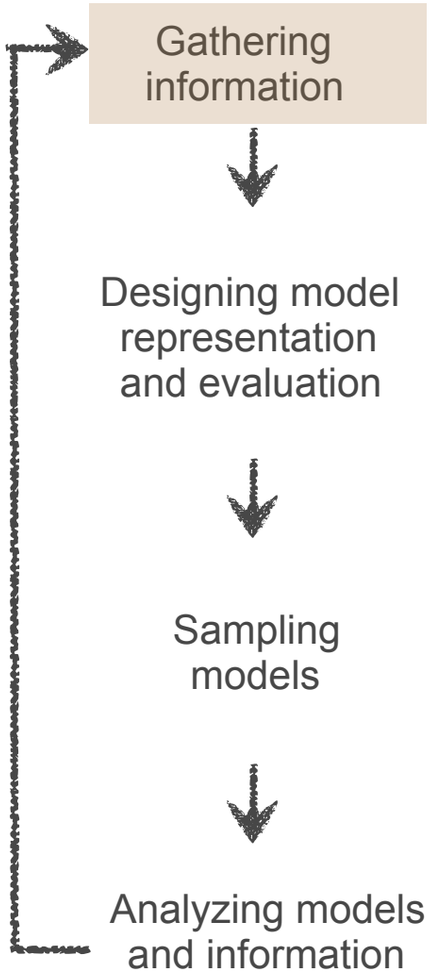


FASTA file

1WCM.fasta.txt:

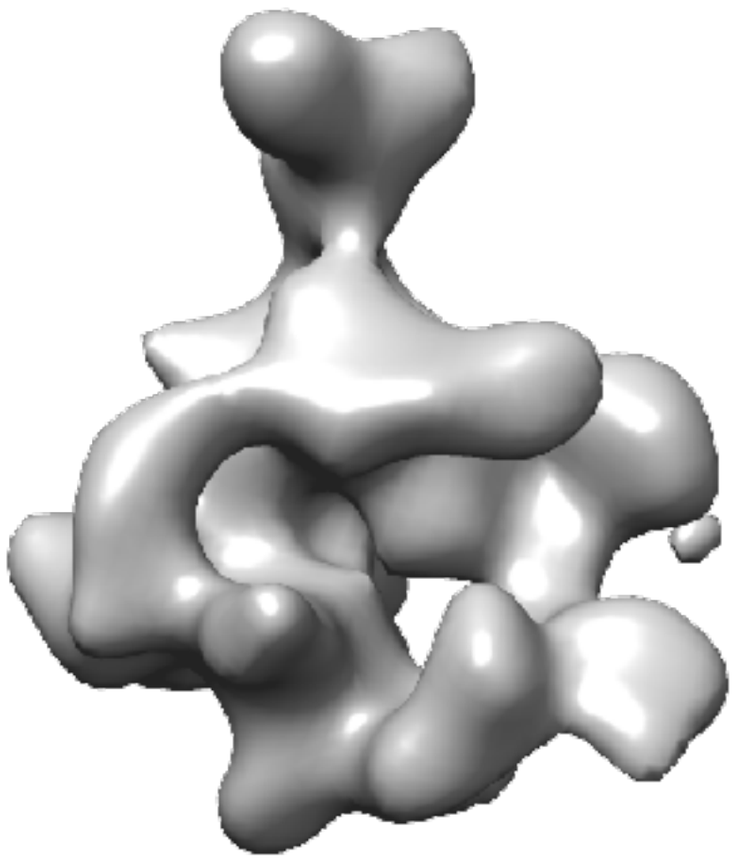
```
>1WCM:A
MVGQQYSSAPLRTVKEVQFGLFSPEEVRAISVAKIRFPETMDETQTRAKIGG
LNDPRLGSIDRNLKCQTCQEGMNECPGHFGHIDLAKPVFHVGFIAKIKKVCE
CVCMHCGKLLLDEHNELMRQALAIKDSKKRFAAIWTLCKTKMVCETDVPSED
...

>1WCM:B
MSDLANSEKYYDEDPYGFEDESAPITAEDSWAVISAFFREKGLVSQQLDSFN
QFVDYTLQDIICEDSTLILEQLAQHTTESDNISRKYEISFGKIYVTKPMVNE
SDGVTHALYPQEARLRNLTYSSGLFVDVKKRTYEAIDVPGRELKYELIAEES
...
```

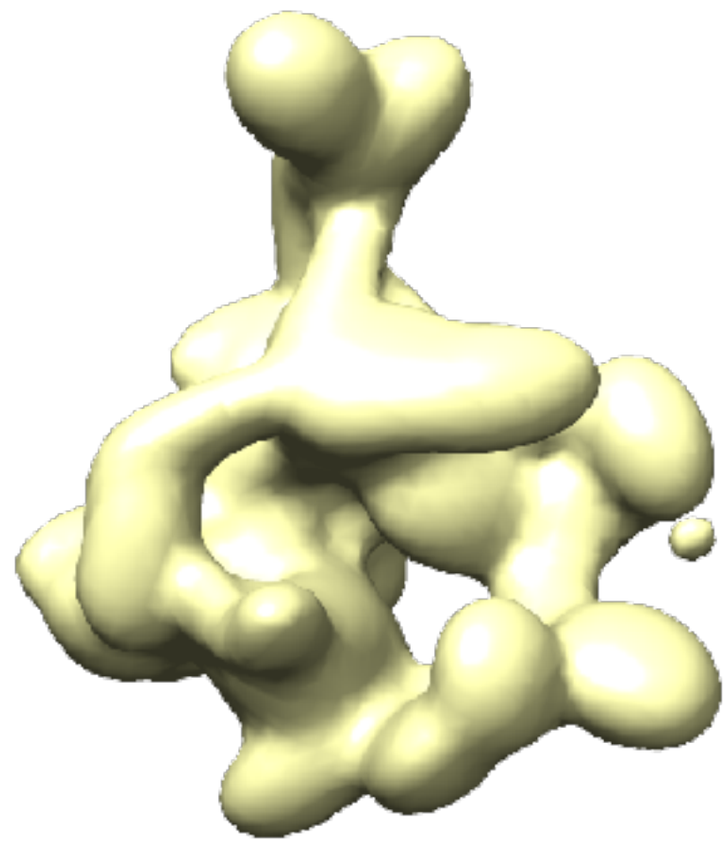
- defines two chains with unique IDs of 1WCM:A and 1WCM:B respectively
- 12 chains in total, A through L

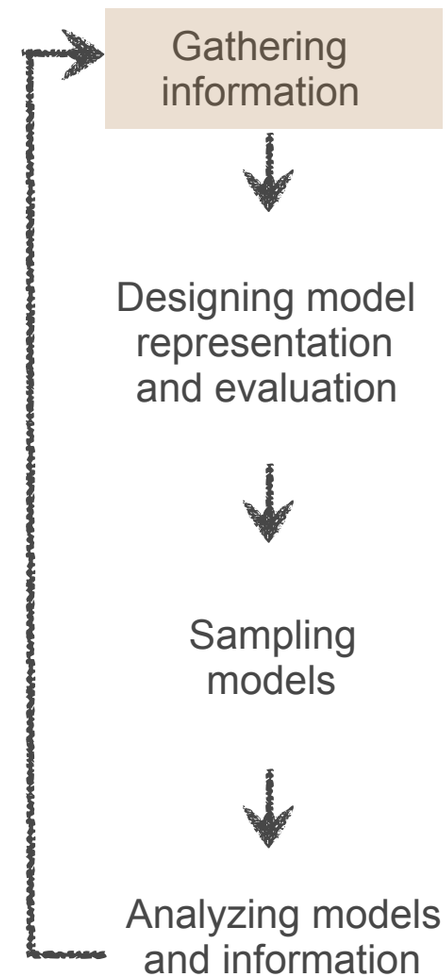


Electron density map

emd_1883.map.mrc experimental map of entire complex at 20.9Å resolution



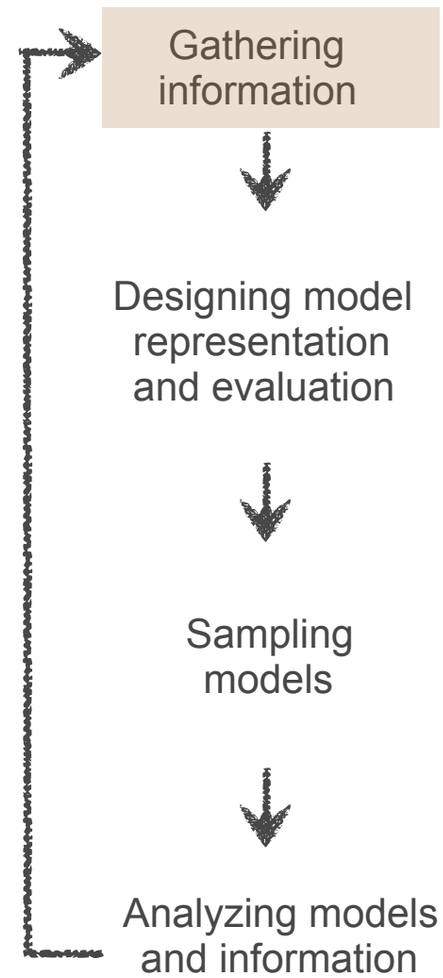




Gaussian mixture models (GMMs) are used to greatly speed up scoring by approximating the electron density of individual subunits and experimental EM maps as a sum of 3D Gaussians. The weight, center, and covariance matrix of each Gaussian used to approximate the original EM density can be seen in *emd_1883.map.mrc.gmm.50.txt*

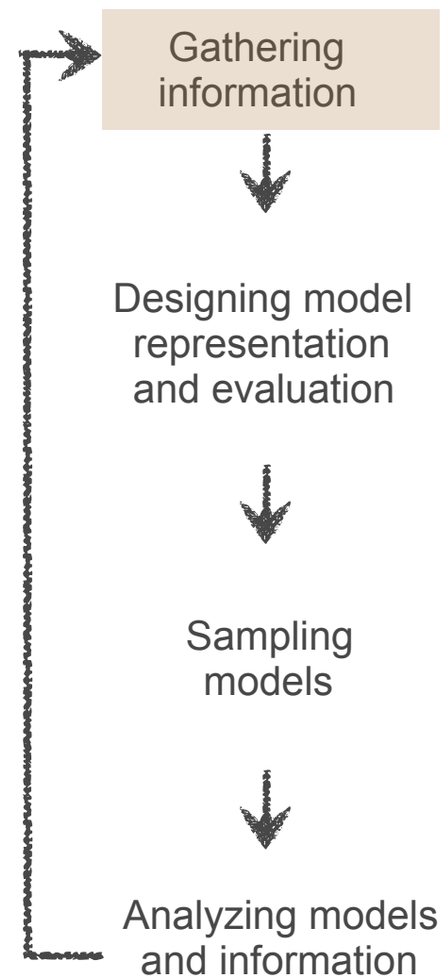
X-ray structures

1WCM.pdb high resolution coordinates for all 12 chains of RNA Pol II



Chemical cross-links

polii_xlinks.csv and *polii_juri.csv*: multiple comma-separated columns; four of these specify the protein and residue number for each of the two linker residues:




```
prot1,res1,prot2,res2
Rpb1,34,Rpb1,49
Rpb1,101,Rpb1,143
Rpb1,101,Rpb1,176
```

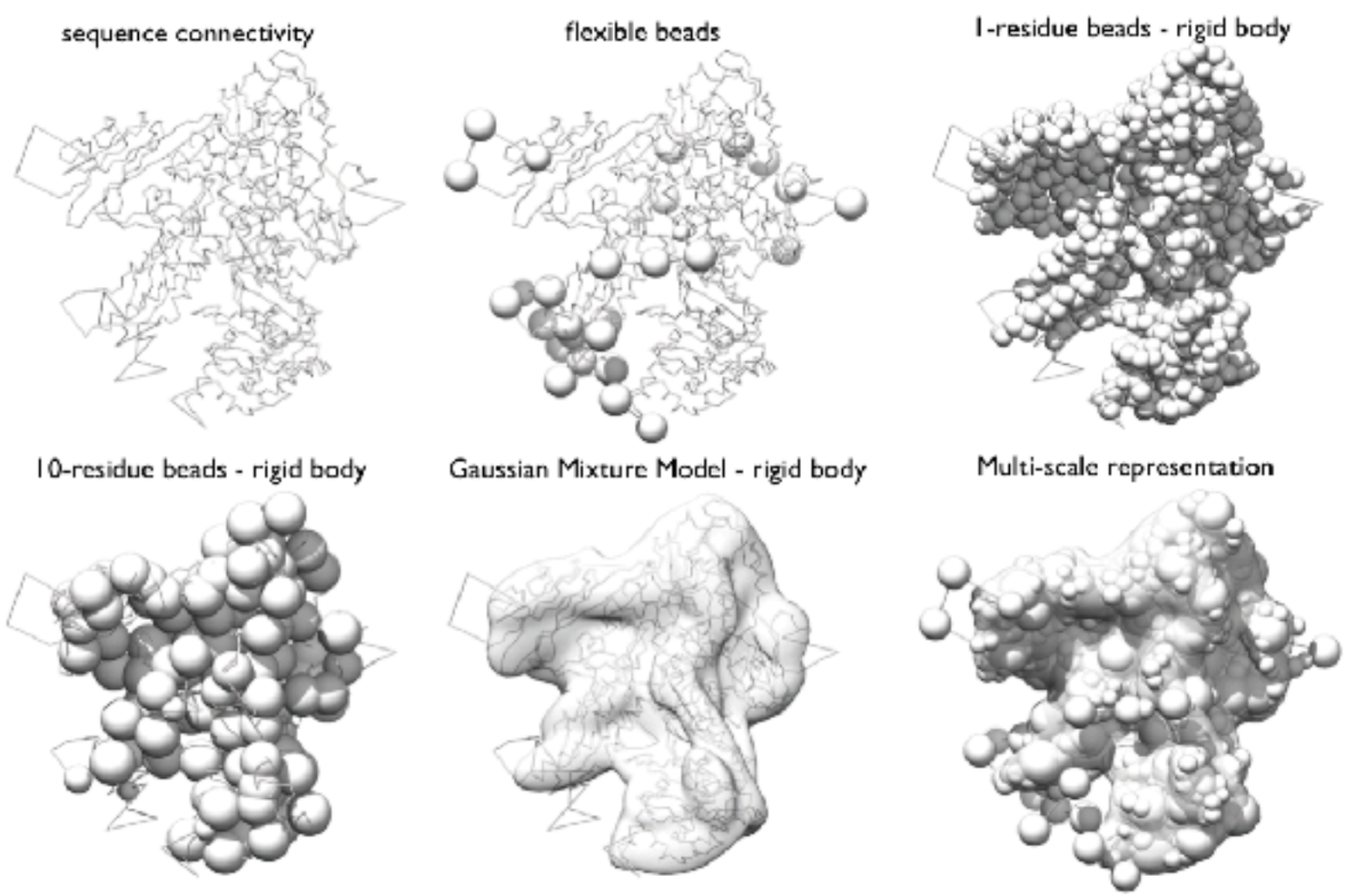
The length of the DSS/BS3 cross-linker reagent, 21Å, will be specified later in the modeling script.

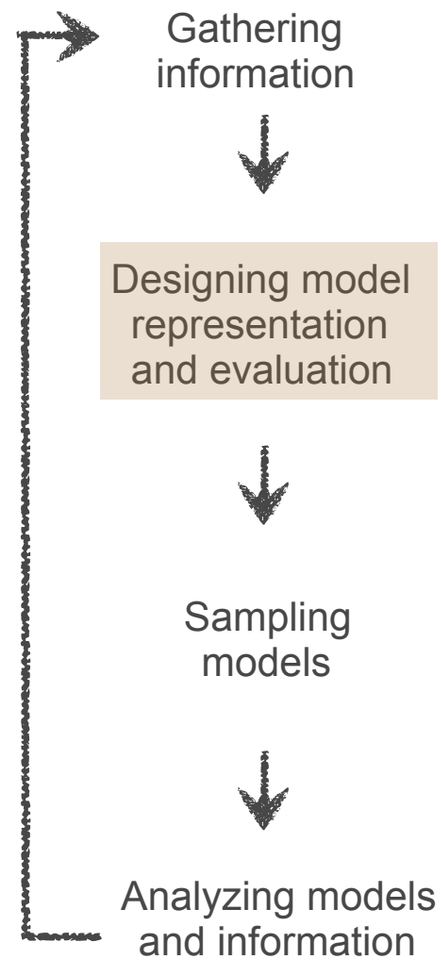
Model representation in IMP

Representation is defined by all the variables that need to be determined based on input information (e.g. points, spheres, ellipsoids, and 3D Gaussian density functions).

We use *spherical beads* and *3D Gaussians*. The *spatial restraints* will be applied to individual resolution scales as appropriate.

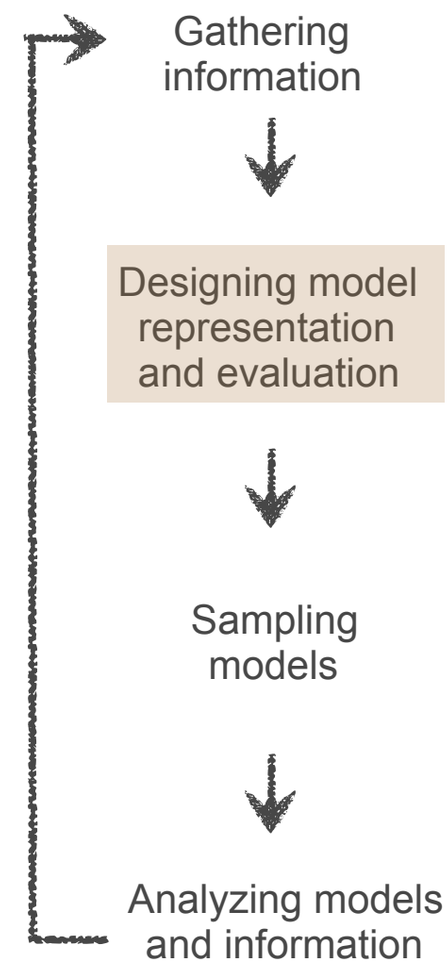
Beads and Gaussians of a given domain are arranged into either a rigid body or a flexible string.

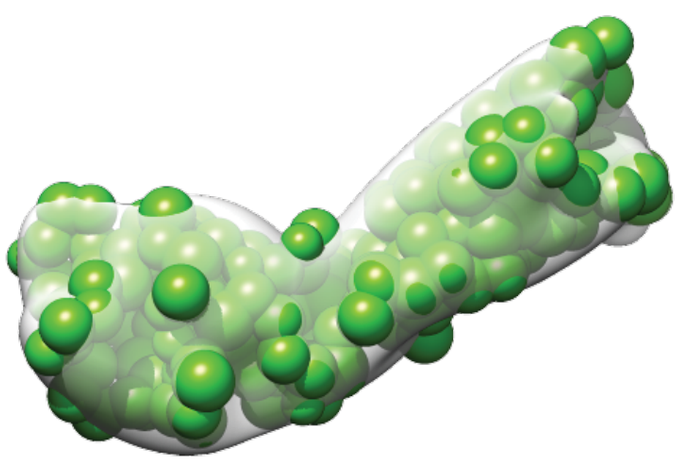


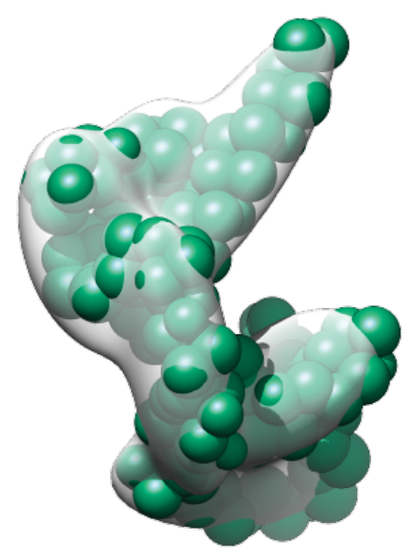


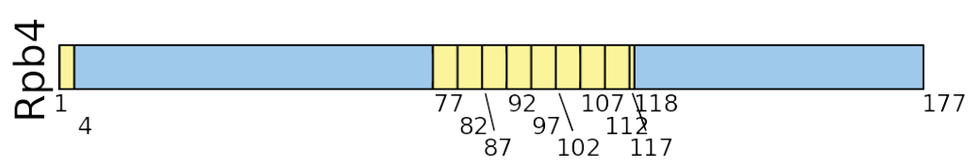
Handling of missing structure

- Even though we have X-ray structures, not all residues were resolved (yellow regions)
- Would be over-interpretation of the data to try to represent this at high resolution
- Use low resolution beads (20 residues per bead) instead here
- Treat high resolution regions as rigid bodies, allow low resolution regions to move (floppy bodies)



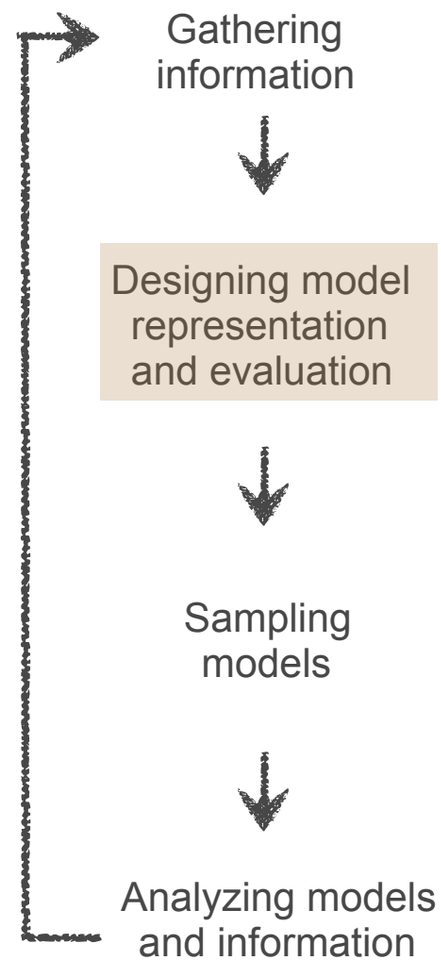






IMP topology file

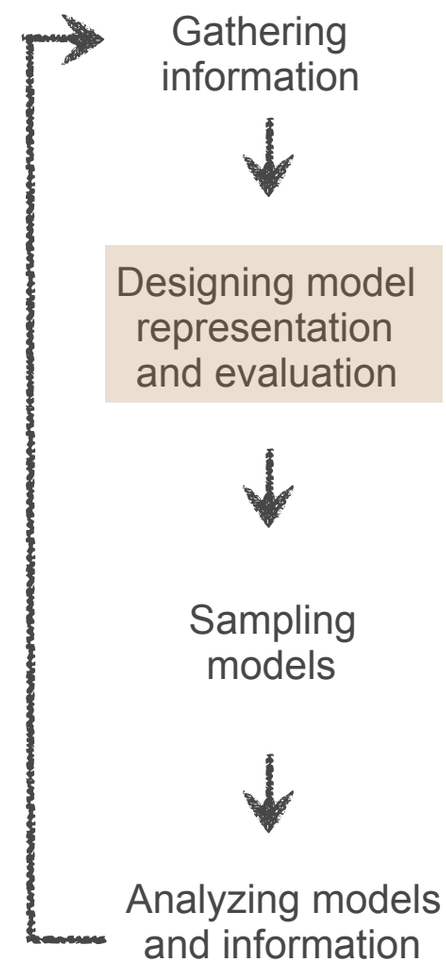
rnapolii/data/topology.txt The topology file stores the basic information needed to create a structural model in IMP.




```
|directories|
|pdb_dir|./|
|fasta_dir|./|
|gmm_dir|./|

|topology_dictionary|
|component_name|domain_name|fasta_fn|fasta_id|pdb_fn|chain|residue_range|pdb_offset|
bead_size|em_residues_per_gaussian|
|Rpb1 |Rpb1_1|1WCM_new.fasta.txt|1WCM:A|1WCM_map_fitted.pdb|A|1,1140   |0|20|0|
|Rpb1 |Rpb1_2|1WCM_new.fasta.txt|1WCM:A|1WCM_map_fitted.pdb|A|1141,1274|0|20|0|
|Rpb1 |Rpb1_3|1WCM_new.fasta.txt|1WCM:A|1WCM_map_fitted.pdb|A|1275,1455|0|20|0|
|Rpb2 |Rpb2_1|1WCM_new.fasta.txt|1WCM:B|1WCM_map_fitted.pdb|B|1,1102   |0|20|0|
|Rpb2 |Rpb2_2|1WCM_new.fasta.txt|1WCM:B|1WCM_map_fitted.pdb|B|1103,-1  |0|20|0|
|Rpb3 |Rpb3  |1WCM_new.fasta.txt|1WCM:C|1WCM_map_fitted.pdb|C|all      |0|20|0|
|Rpb4 |Rpb4  |1WCM_new.fasta.txt|1WCM:D|1WCM_map_fitted.pdb|D|all      |0|20|40|
|Rpb5 |Rpb5  |1WCM_new.fasta.txt|1WCM:E|1WCM_map_fitted.pdb|E|all      |0|20|0|
|Rpb6 |Rpb6  |1WCM_new.fasta.txt|1WCM:F|1WCM_map_fitted.pdb|F|all      |0|20|0|
|Rpb7 |Rpb7  |1WCM_new.fasta.txt|1WCM:G|1WCM_map_fitted.pdb|G|all      |0|20|40|
|Rpb8 |Rpb8  |1WCM_new.fasta.txt|1WCM:H|1WCM_map_fitted.pdb|H|all      |0|20|0|
|Rpb9 |Rpb9  |1WCM_new.fasta.txt|1WCM:I|1WCM_map_fitted.pdb|I|all      |0|20|0|
|Rpb10|Rpb10 |1WCM_new.fasta.txt|1WCM:J|1WCM_map_fitted.pdb|J|all      |0|20|0|
|Rpb11|Rpb11 |1WCM_new.fasta.txt|1WCM:K|1WCM_map_fitted.pdb|K|all      |0|20|0|
|Rpb12|Rpb12 |1WCM_new.fasta.txt|1WCM:L|1WCM_map_fitted.pdb|L|all      |0|20|0|
```

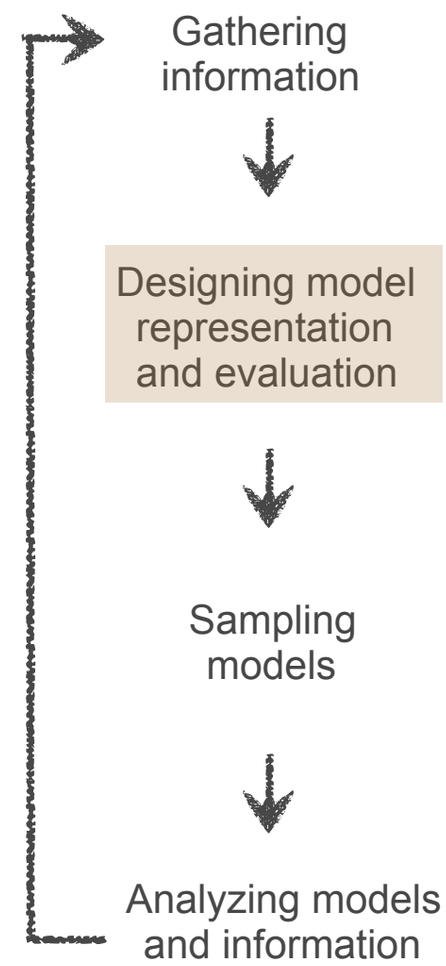
Evaluation

- At this point we need to create our scoring function, by which the individual structural models will be scored based on the input data
- A sum of individual restraints
- Each restraint maps to one of our input experiments or other physical/statistical information



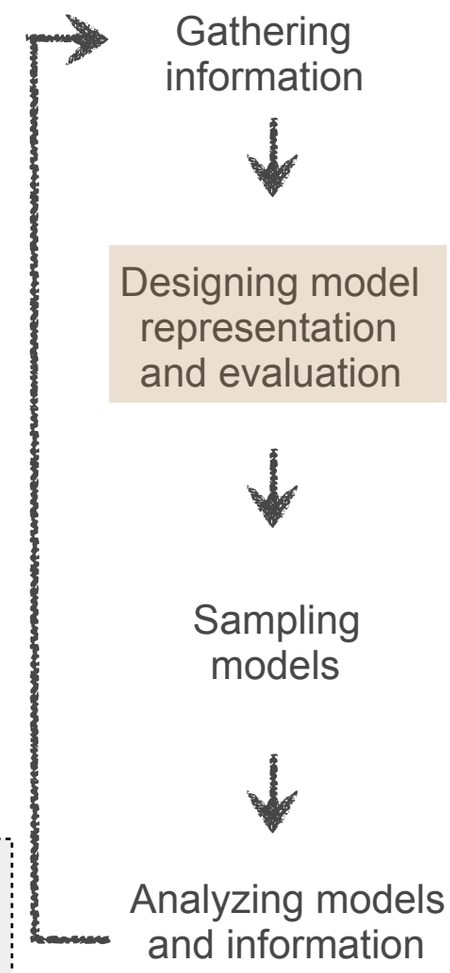
Sequence connectivity restraint

- We know that residues that are adjacent in *sequence* will also be close in *space*, due to the peptide bond
- We should enforce this in our modeling by adding simple harmonic restraints between beads
- PMI handles this automatically based on the FASTA file
  - nothing needed in our script



Excluded volume restraint

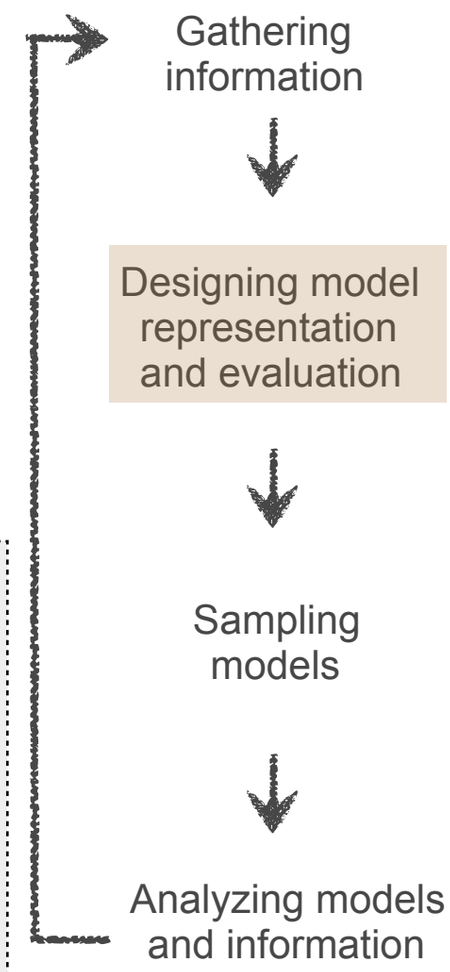
- We also know that one protein cannot occupy the same space as another
- The excluded volume restraint is calculated at resolution 20 (20 residues per bead)
  - Faster to evaluate, but more approximate
- We're maintaining a list of 'output objects', and this will be one of them
  - Statistics on such objects (e.g. whether the score is satisfied) will be collected during the modeling




```
ev = IMP.pmi.restraints.stereochemistry.ExcludedVolumeSphere(
                                        representation, resolution=20)
ev.add_to_model()
outputobjects.append(ev)
```

Crosslinking restraints

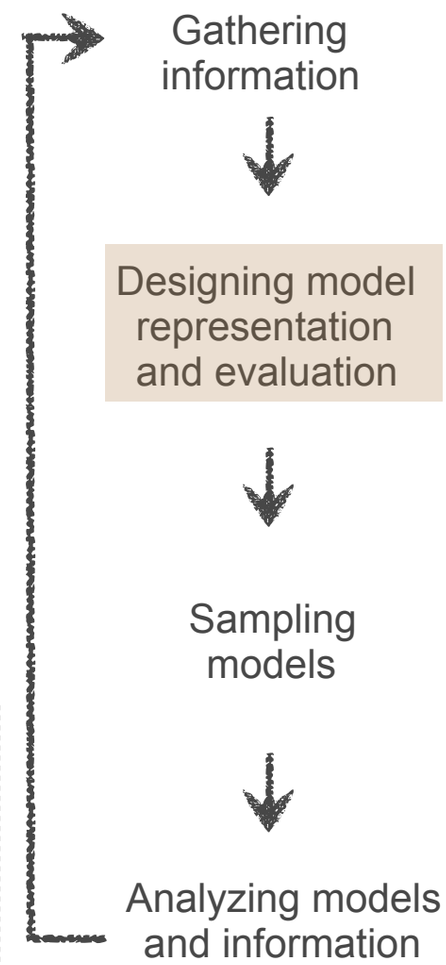
- Restrain residue pairs based on the crosslinks files
- Residue-level information, so apply at resolution 1
- Length of cross linker given here
- The restraint is Bayesian with ψ and σ noise parameters
    - We'll need to sample those parameters later at the same time as the xyz coordinates (`sampleobjects`)




```
xl1 = IMP.pmi.restraints.crosslinking.ISDCrossLinkMS(representation,
                                     datadirectory+'polii_xlinks.csv',
                                     length=21.0,
                                     slope=0.01,
                                     columnmapping=columnmap,
                                     resolution=1.0,
                                     label="Trnka",
                                     csvfile=True)

xl1.add_to_model()
sampleobjects.append(xl1)
outputobjects.append(xl1)
```

EM restraint

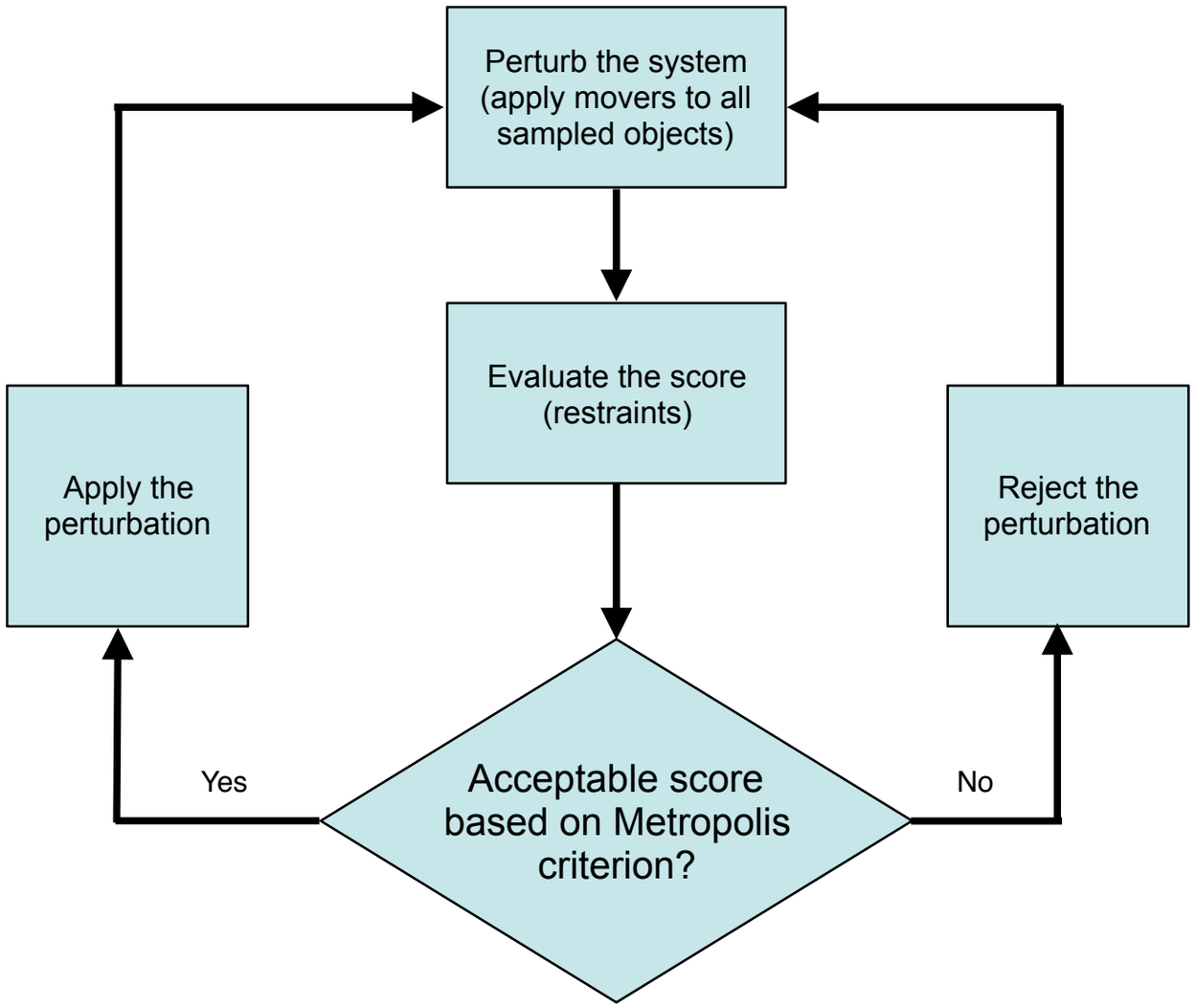
- We're using a density overlap function to compare the GMM approximation of our model (`em_components`) with that of the EM map itself (`target_gmm_file`)
  - `scale_to_target_mass` ensures the total masses of model and map are identical
  - `slope`: nudge model closer to map when far away
  - `weight`: heuristic, needed to calibrate the EM restraint with the other terms.

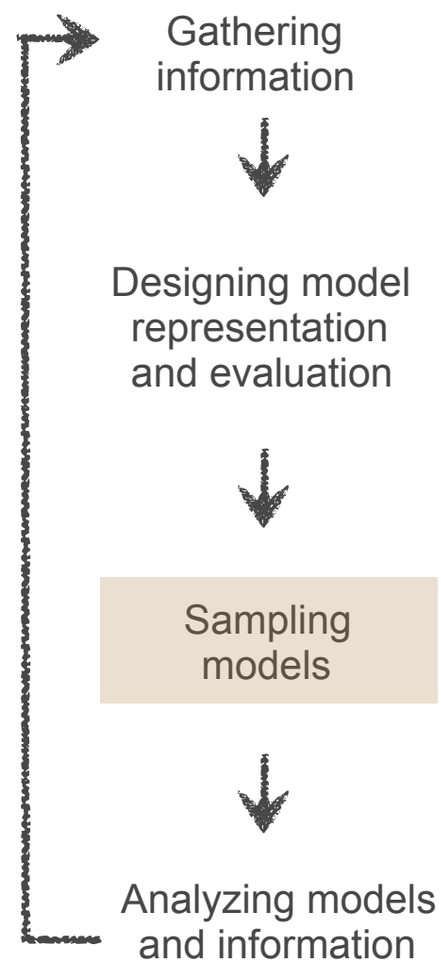



```
em_components = bm.get_density_hierarchies([t.domain_name for t in domains])
gemt = IMP.pmi.restraints.em.GaussianEMRestraint(em_components,
                                                 target_gmm_file,
                                                 scale_target_to_mass=True,
                                                 slope=0.000001,
                                                 weight=100.0)

gemt.add_to_model()
outputobjects.append(gemt)
```

Sampling

- We're going to use Monte Carlo to *sample* (not minimize) our system (generate many models that satisfy the data)



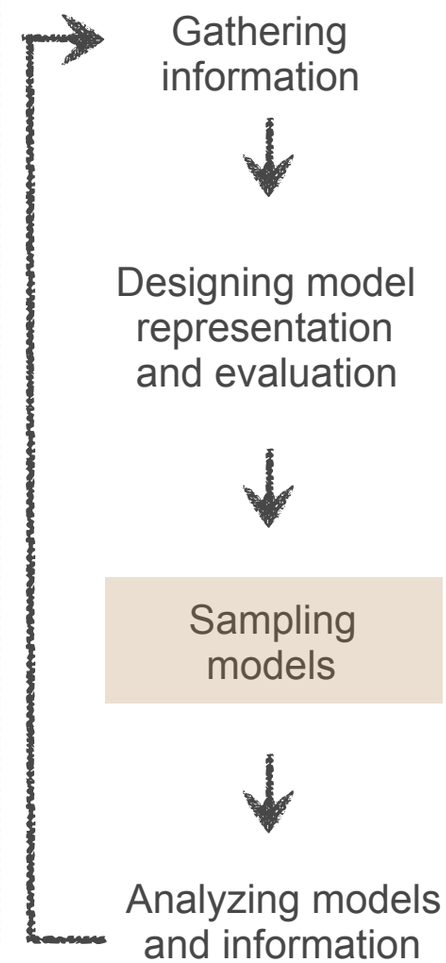


- Thus, need to define a set of movers

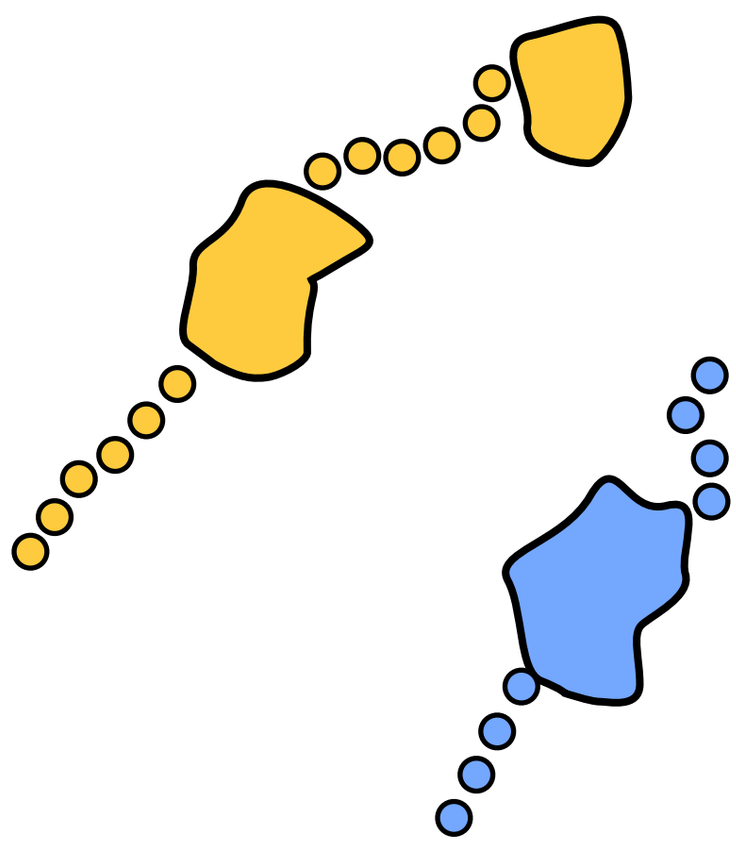
Monte Carlo setup

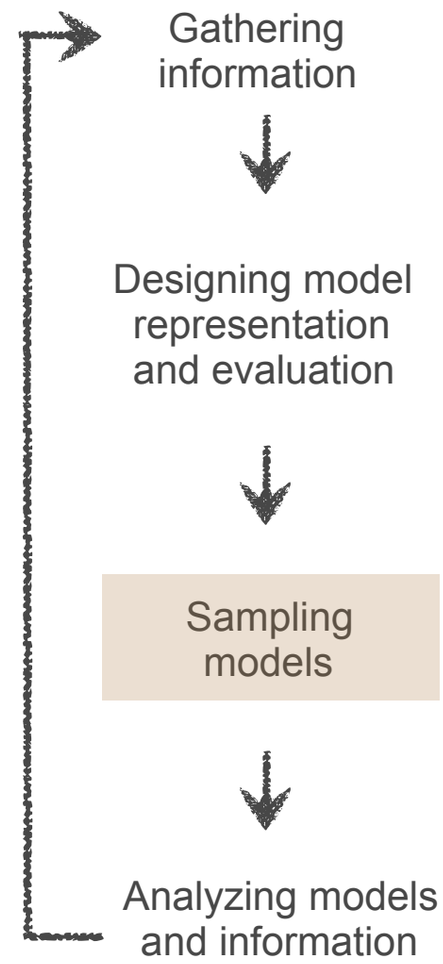
- Bead movers: simple 3D translation, sampled linearly up to a given max value
- Rigid body movers: 3D translation and rotation
- Also we define here how to move our rigid bodies
- (Remember that we also ‘move’ non-Cartesian parameters for our Bayesian restraints)

```
#-------------------------
# Set MC Sampling Parameters
#-------------------------
num_frames = 20000
num_mc_steps = 10

#-------------------------
# Create movers
#-------------------------

# rigid body movement params
rb_max_trans = 2.00
rb_max_rot = 0.04

# flexible bead movement
bead_max_trans = 3.00

rigid_bodies = [["Rpb4"],
                ["Rpb7"]]
super_rigid_bodies = [["Rpb4","Rpb7"]]
chain_of_super_rigid_bodies = [["Rpb4"],
                               ["Rpb7"]]
```



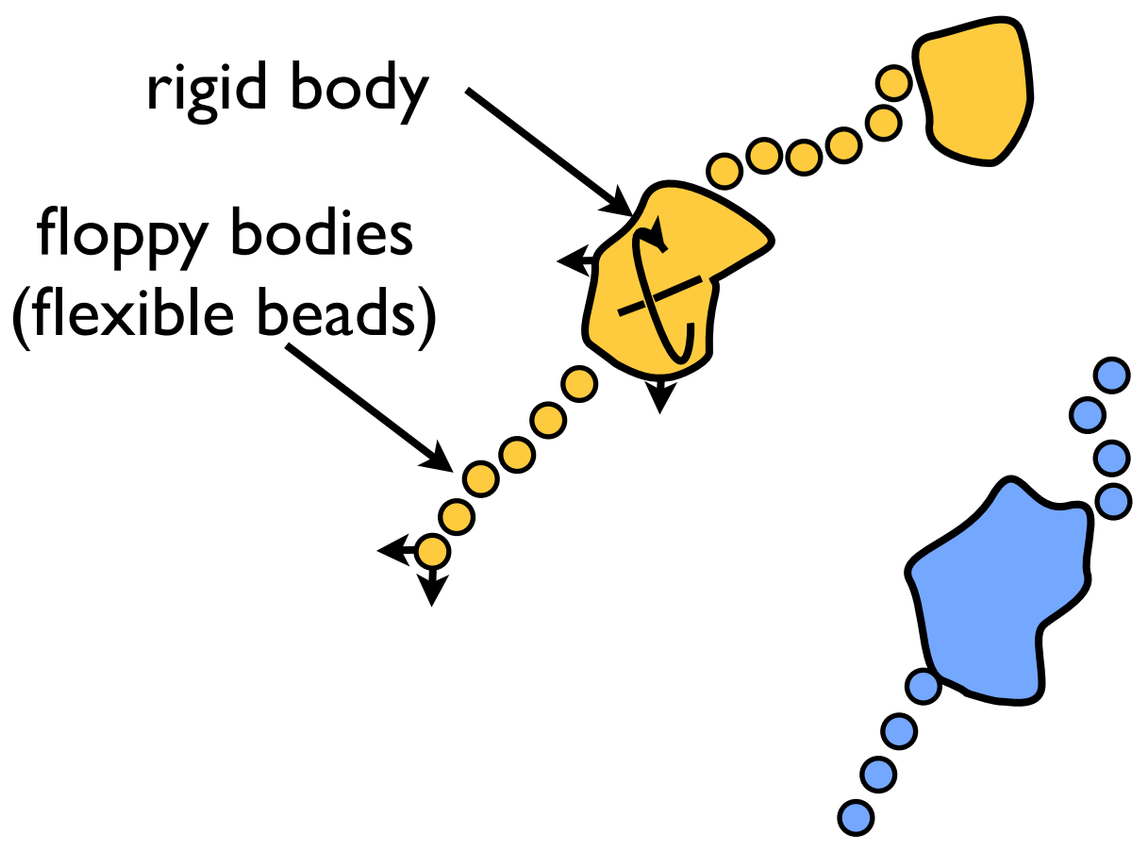
Rigid body movers

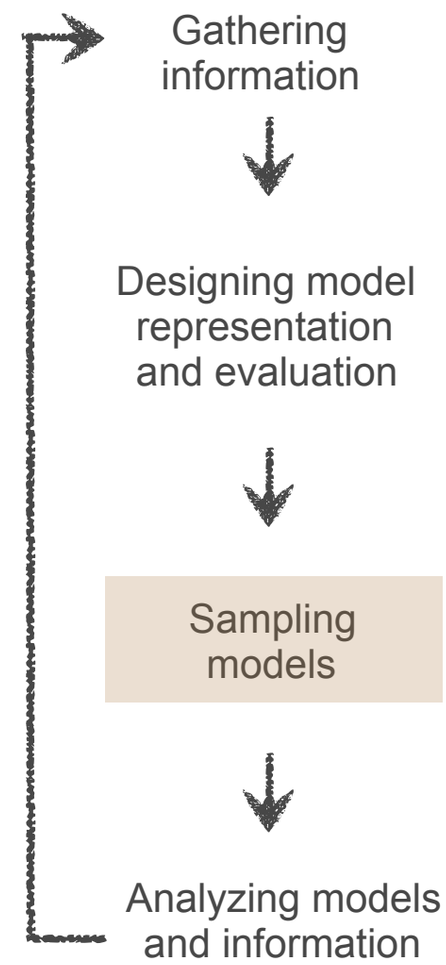




Rigid body movers

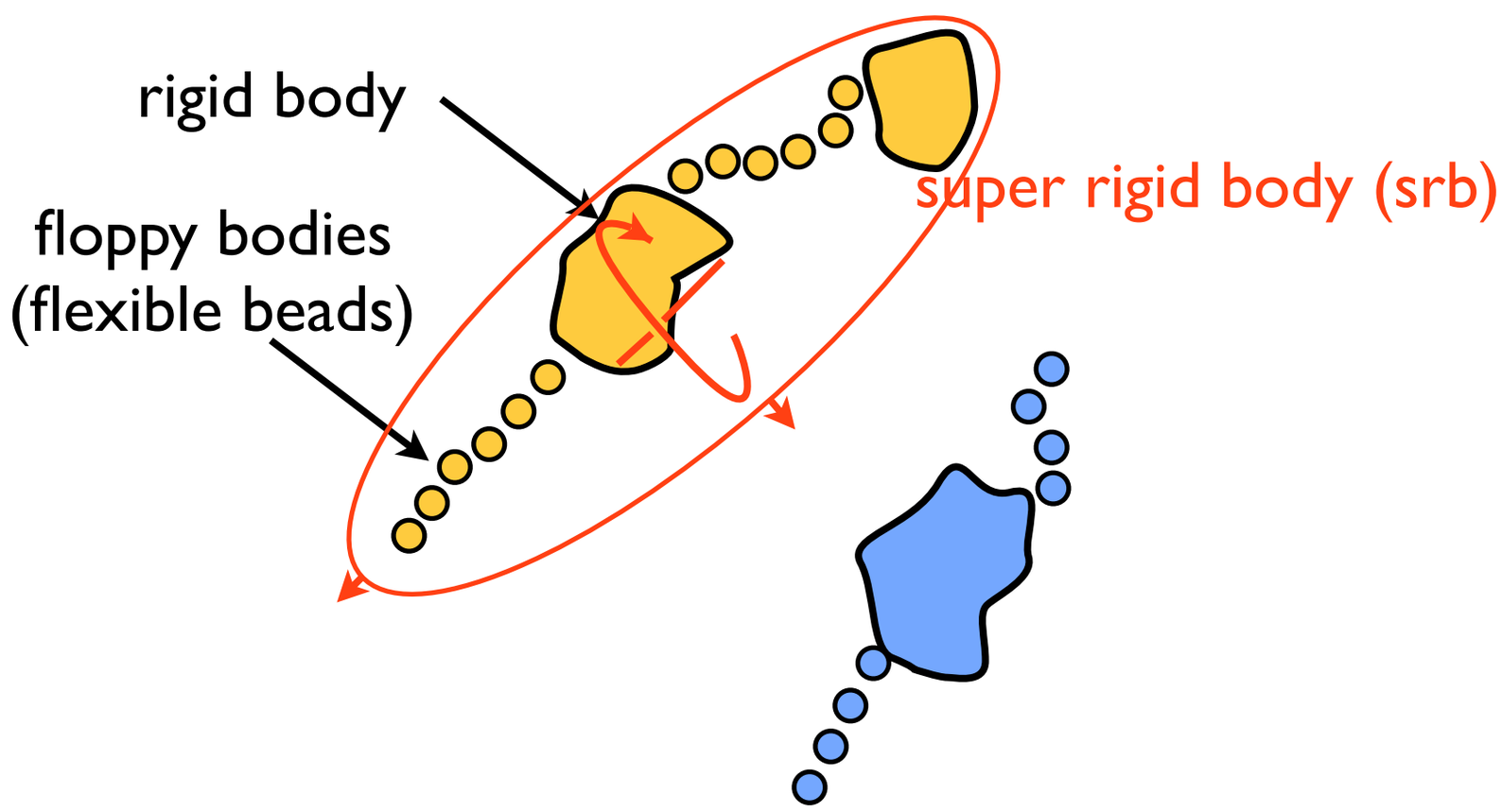
rigid_bodies defines the components that will be moved as rigid bodies (in this case, the parts of Rpb4 and Rpb7 for which we have X-ray structure). Unstructured regions will move as flexible beads.

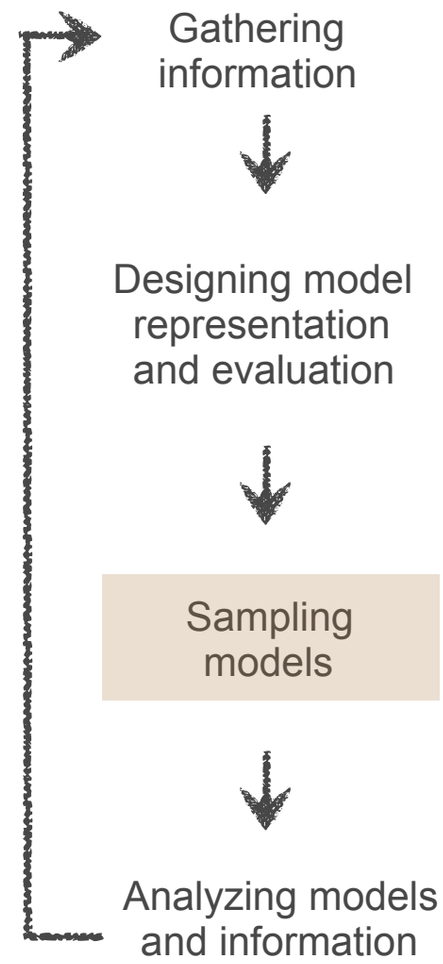




Rigid body movers

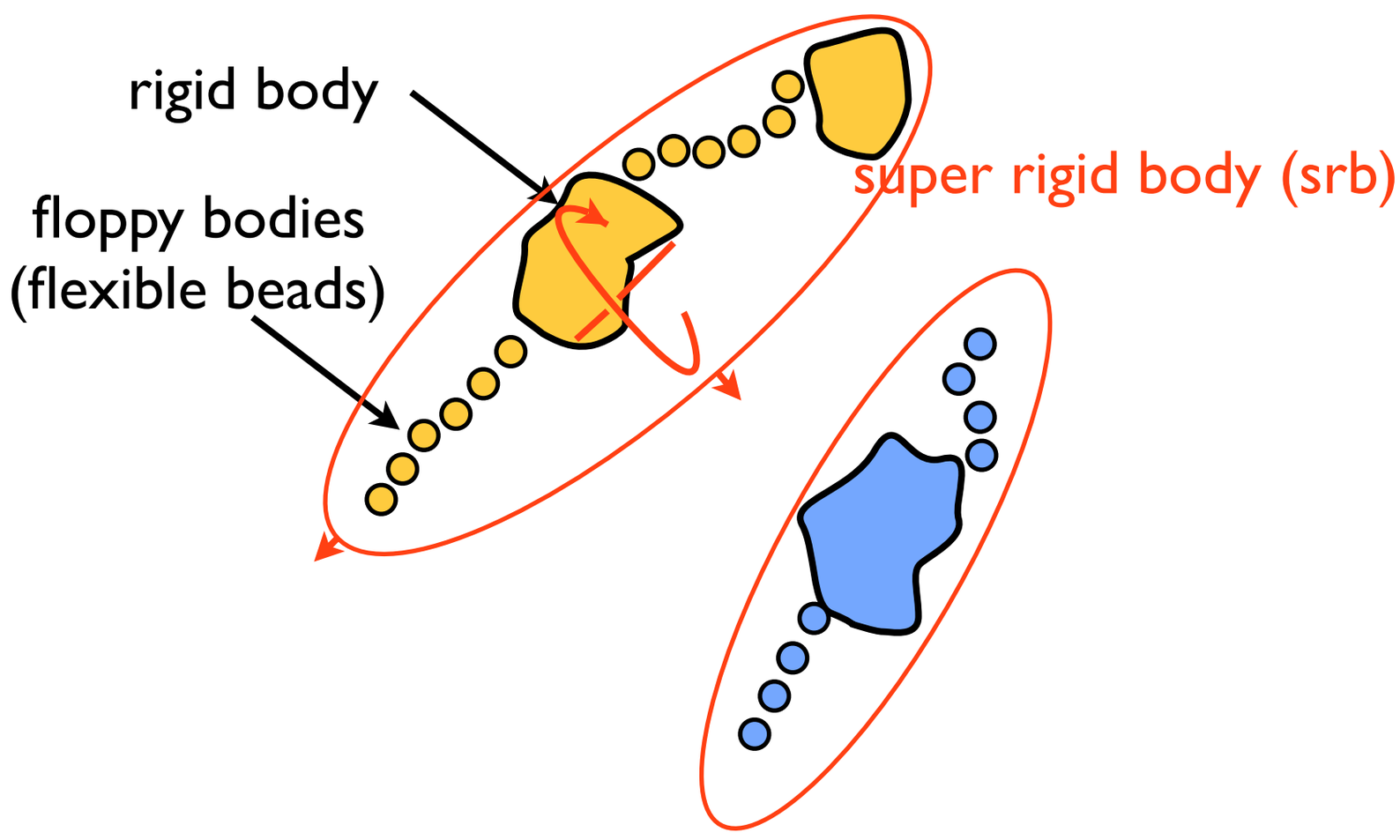
super_rigid_bodies defines sets of rigid bodies and beads that will move together in an additional Monte Carlo move.

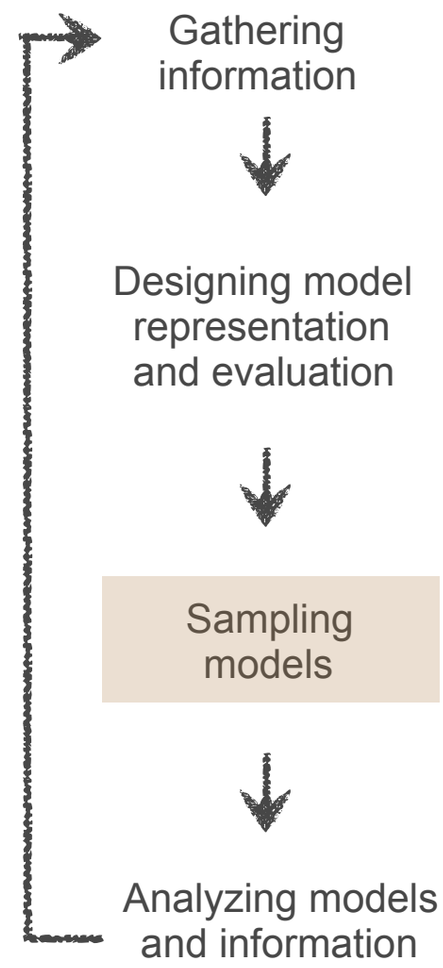




Rigid body movers

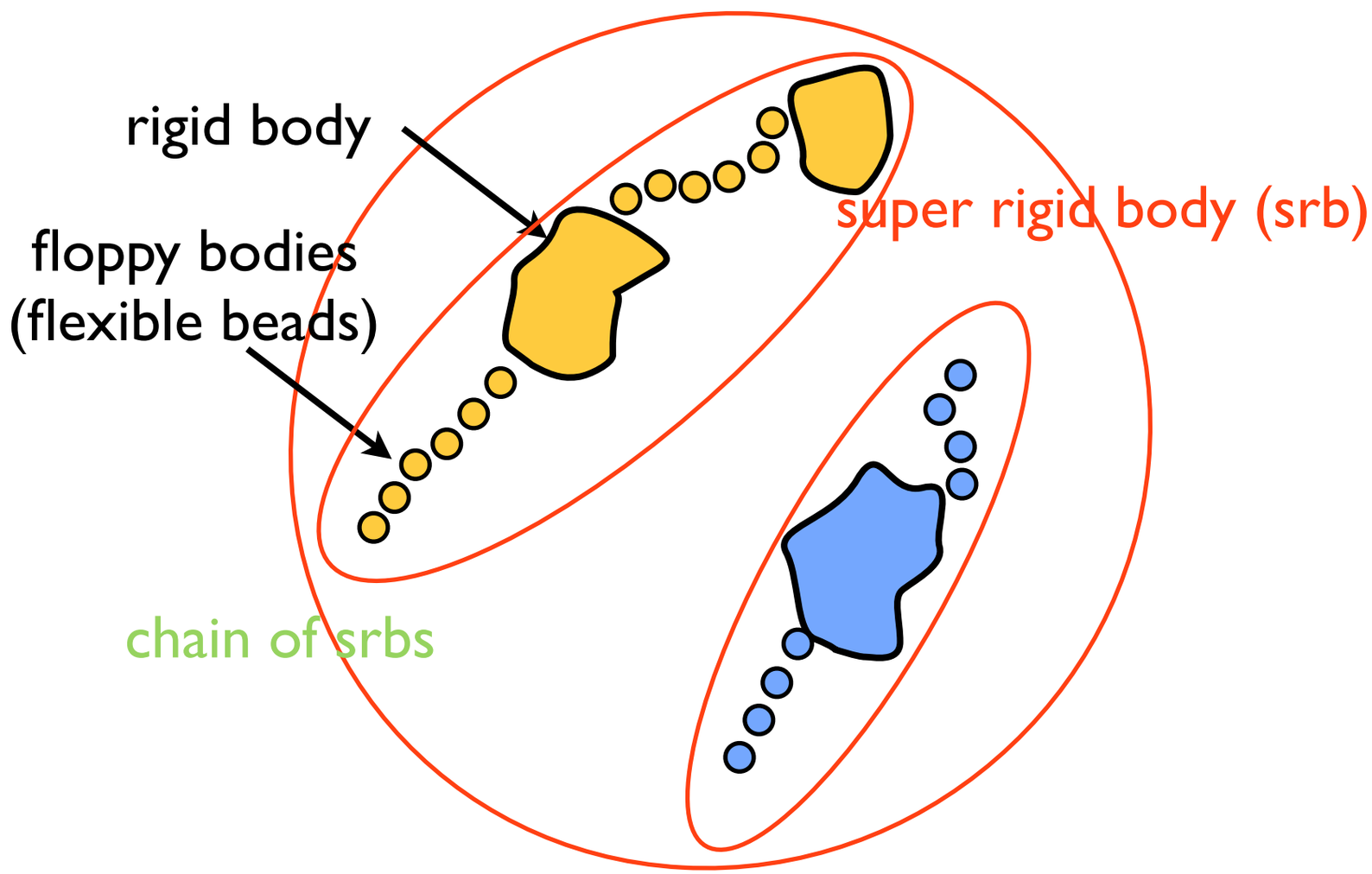
super_rigid_bodies defines sets of rigid bodies and beads that will move together in an additional Monte Carlo move.

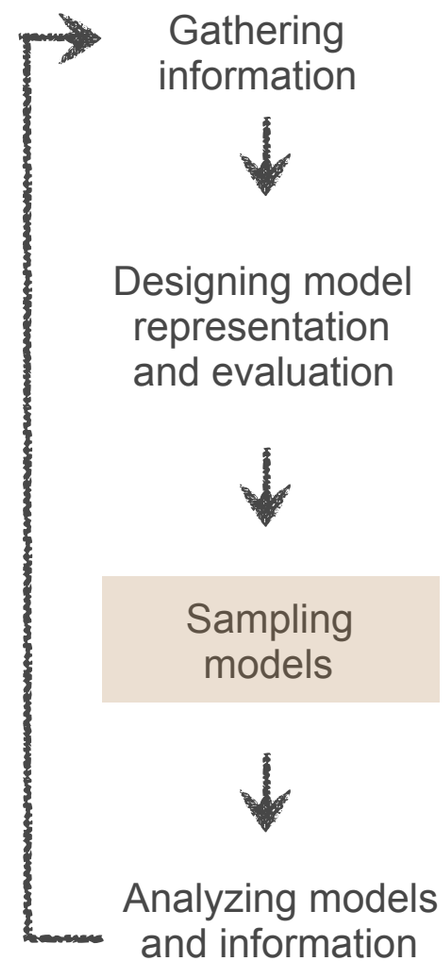




Rigid body movers

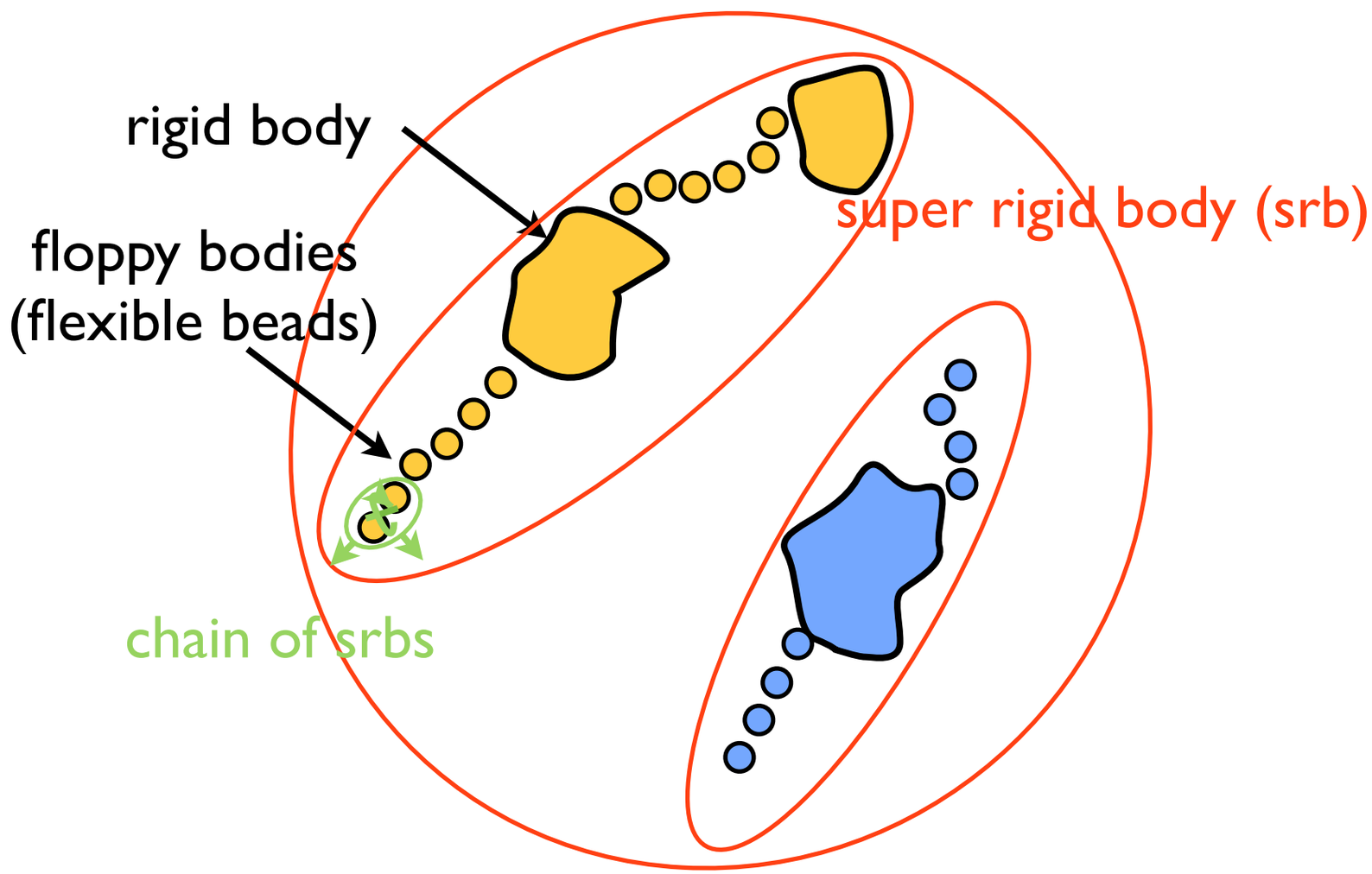
chain_of_super_rigid_bodies sets additional Monte Carlo movers along the connectivity chain of a subunit. It groups sequence-connected rigid domains and/or beads into overlapping pairs and triplets. Each of these groups will be moved rigidly. This mover helps to sample more efficiently complex topologies, made of several rigid bodies, connected by flexible linkers.

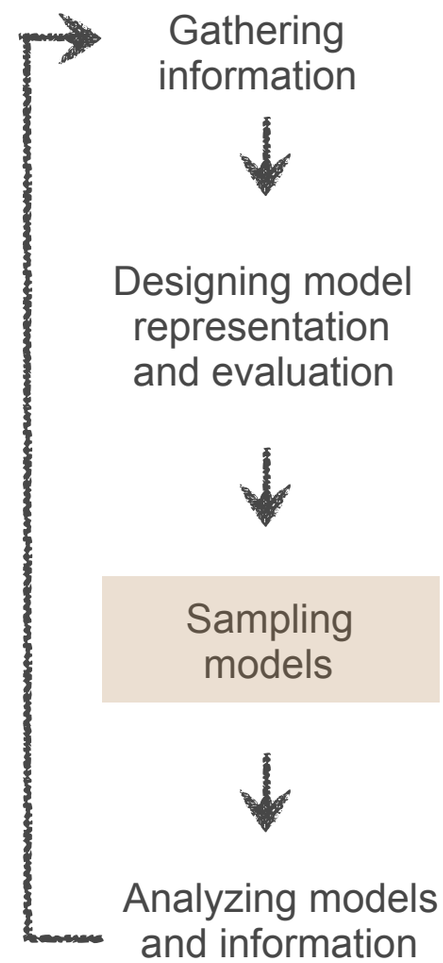




Rigid body movers

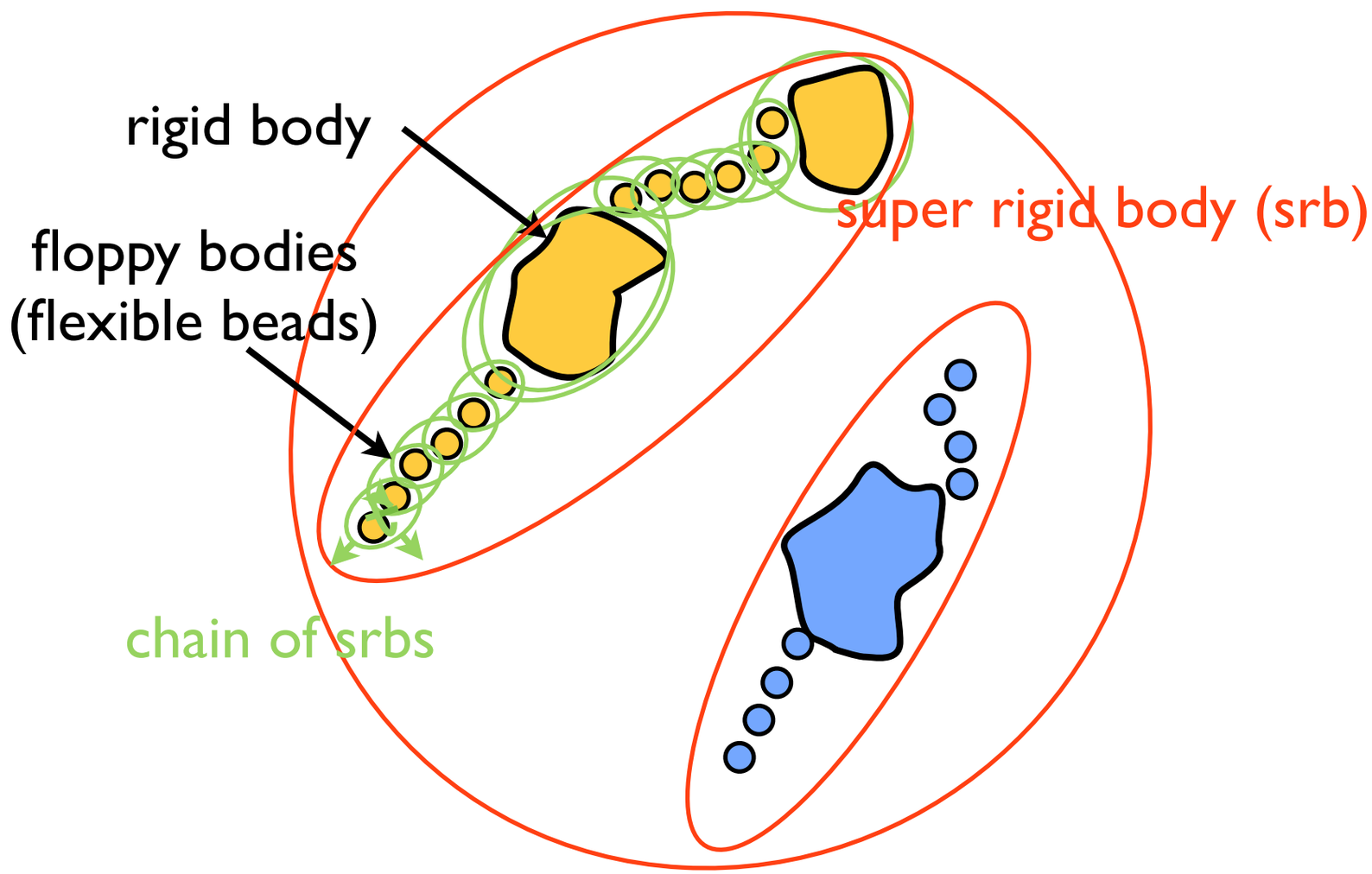
chain_of_super_rigid_bodies sets additional Monte Carlo movers along the connectivity chain of a subunit. It groups sequence-connected rigid domains and/or beads into overlapping pairs and triplets. Each of these groups will be moved rigidly. This mover helps to sample more efficiently complex topologies, made of several rigid bodies, connected by flexible linkers.

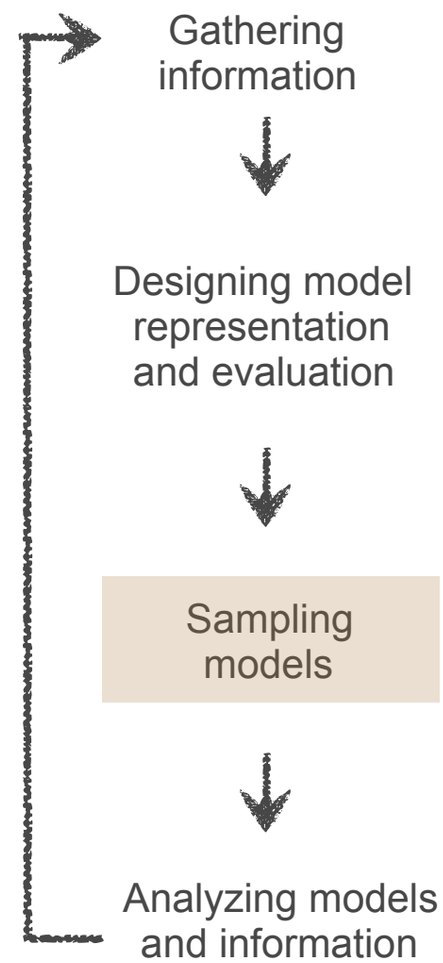




Rigid body movers

chain_of_super_rigid_bodies sets additional Monte Carlo movers along the connectivity chain of a subunit. It groups sequence-connected rigid domains and/or beads into overlapping pairs and triplets. Each of these groups will be moved rigidly. This mover helps to sample more efficiently complex topologies, made of several rigid bodies, connected by flexible linkers.

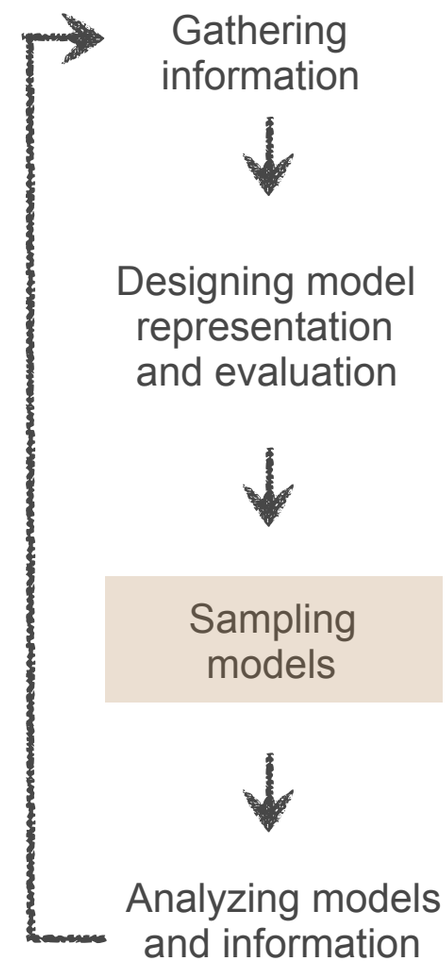




Sampling

- Finally, we run the Monte Carlo sampling itself
- Technically this is replica exchange but with only one replica (we're not running in parallel with MPI)

```
mc1=IMP.pmi.macros.ReplicaExchange0(m,
                                    representation,
                                    monte_carlo_sample_objects=sampleobjects,
                                    output_objects=outputobjects,
                                    crosslink_restraints=[xl1,xl2],
                                    monte_carlo_temperature=1.0,
                                    simulated_annealing=True,
                                    simulated_annealing_minimum_temperature=1.0,
                                    simulated_annealing_maximum_temperature=2.5,
                                    simulated_annealing_minimum_temperature_nframes=200,
                                    simulated_annealing_maximum_temperature_nframes=20,
                                    replica_exchange_minimum_temperature=1.0,
                                    replica_exchange_maximum_temperature=2.5,
                                    number_of_best_scoring_models=100,
                                    monte_carlo_steps=num_mc_steps,
                                    number_of_frames=num_frames,
                                    global_output_directory="output")
```



Running the script

```
python modeling.py
autobuild_model: constructing Rpb1 from pdb ../data/./1WCM_map_fitted.pdb and chain A
autobuild_model: constructing fragment (1, 1) as a bead
autobuild_model: constructing fragment (2, 186) from pdb
autobuild_model: constructing fragment (187, 194) as a bead
autobuild_model: constructing fragment (195, 1081) from pdb
autobuild_model: constructing fragment (1082, 1091) as a bead
autobuild_model: constructing fragment (1092, 1140) from pdb
autobuild_model: constructing Rpb1 from pdb ../data/./1WCM_map_fitted.pdb and chain A
autobuild_model: constructing fragment (1141, 1176) from pdb
autobuild_model: constructing fragment (1177, 1186) as a bead
autobuild_model: constructing fragment (1187, 1243) from pdb
autobuild_model: constructing fragment (1244, 1253) as a bead
```

•••

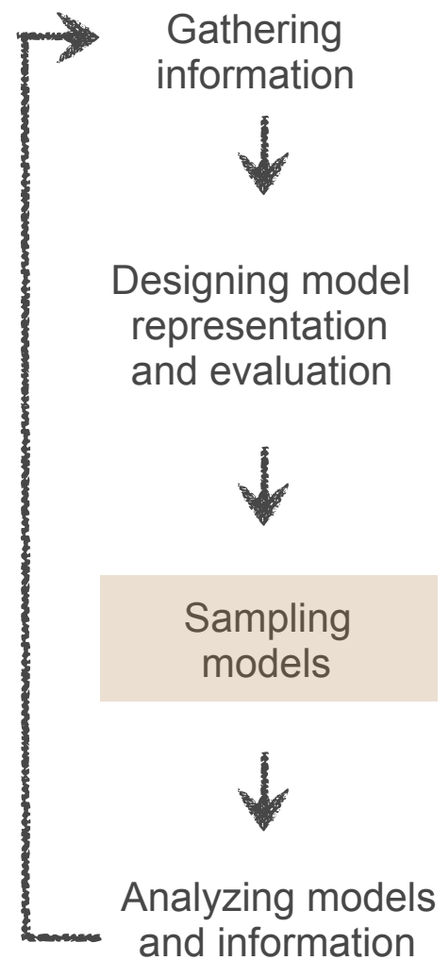
```
Adding sequence connectivity restraint between Rpb4_1-3_bead  and  Rpb4_4_13_pdb of distance 14.4
Adding sequence connectivity restraint between Rpb4_74_76_pdb  and  Rpb4_77-96_bead of distance 14.4
Adding sequence connectivity restraint between Rpb4_77-96_bead  and  Rpb4_97-116_bead of distance 14.4
Adding sequence connectivity restraint between Rpb4_97-116_bead  and  Rpb4_117_bead of distance 14.4
```

•••

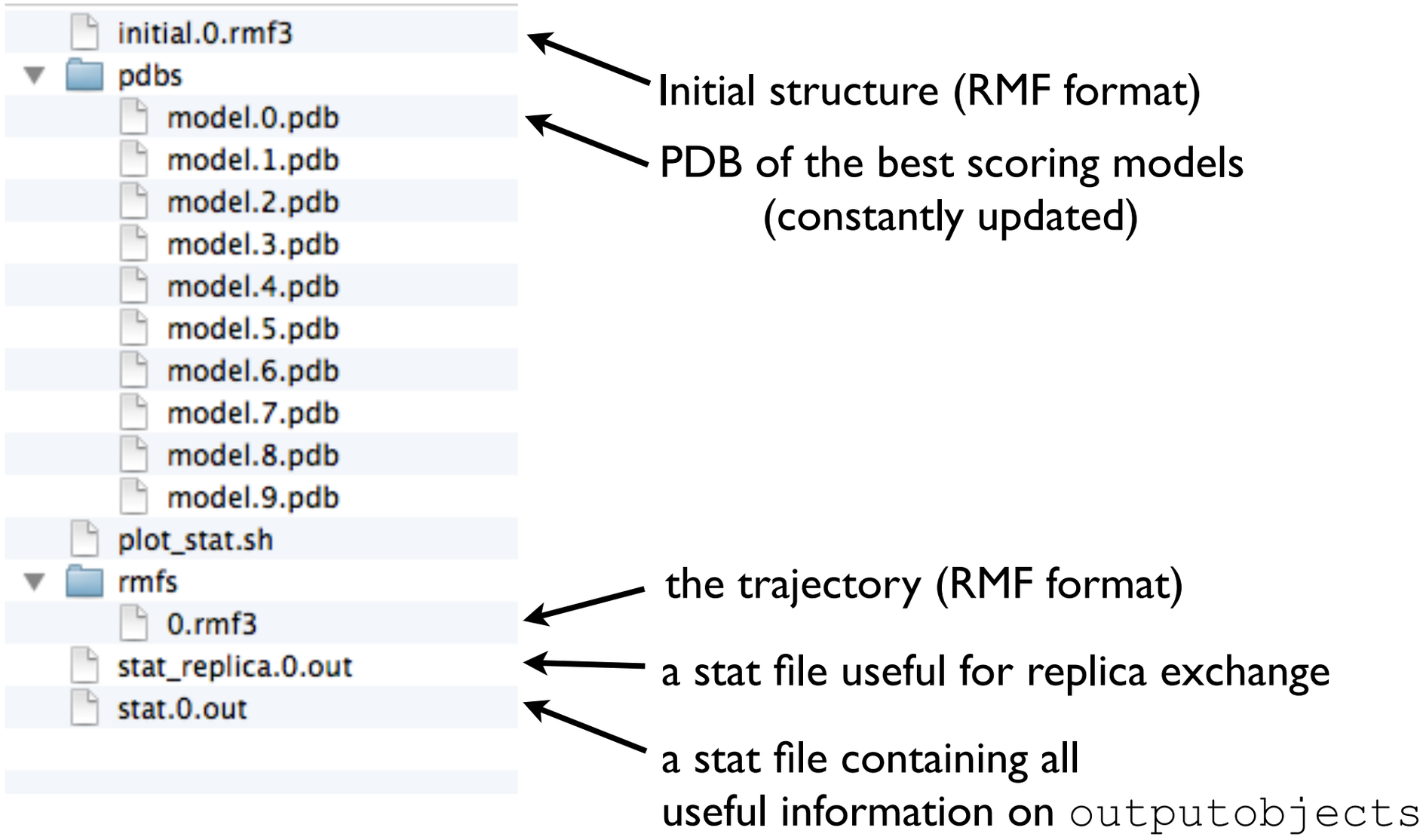
```
generating a new crosslink restraint
--------------
ISDCrossLinkMS: generating cross-link restraint between
ISDCrossLinkMS: residue 358 of chain Rpb2 and residue 246 of chain Rpb2
ISDCrossLinkMS: with sigma1 1.000000 sigma2 1.000000 psi 0.05
ISDCrossLinkMS: between particles Rpb2_358_pdb and Rpb2_246_pdb
==========================================
```

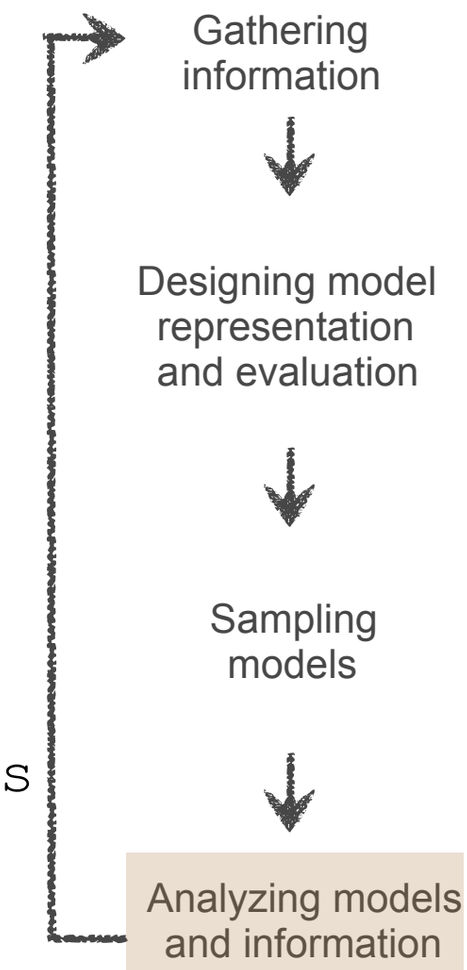
•••

```
--- frame 1 score 4814598.44759
--- writing coordinates
--- frame 2 score 3527090.92513
--- writing coordinates
--- frame 3 score 2662180.99705
--- writing coordinates
--- frame 4 score 2021182.74211
```



Output data

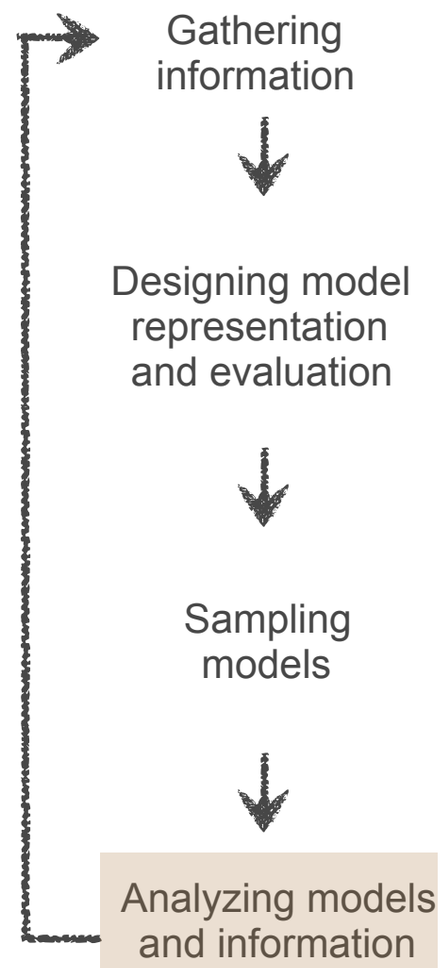




- Note that PDB is not well suited for non-atomic structures
- IMP uses its own format (RMF) for coarse-grained structures
- PDB's next generation file format (mmCIF) will natively support these structures

Analysis

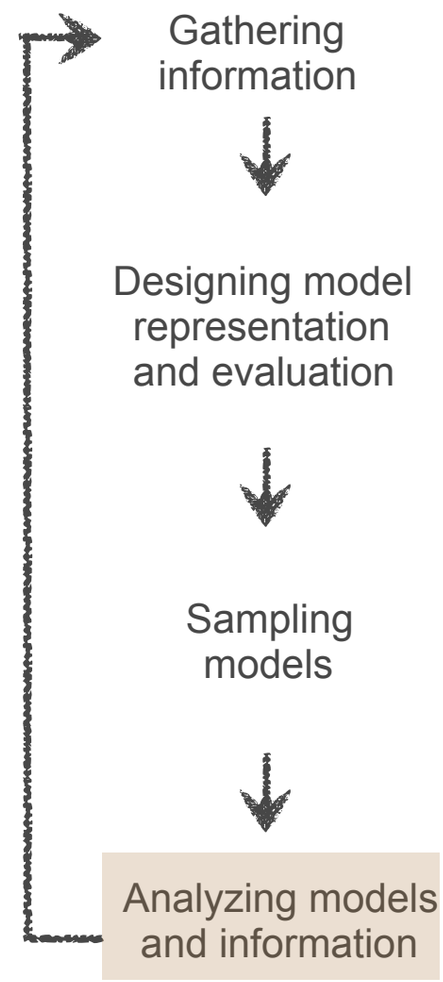
- In the analysis stage we cluster (group by similarity) the sampled models to determine high-probability configurations. Comparing clusters may indicate that there are multiple acceptable configurations given the data.
- Cluster Precision: Determining the within-group precision and between-group similarity via RMSD
- Cluster Accuracy: Fit of the calculated clusters to the true (known) solution
- Sampling Exhaustiveness: Qualitative and quantitative measurement of sampling completeness



Clustering

- A simple clustering protocol is shown in `rnapolii/analysis/clustering.py`
- Simply run with `python clustering.py --test`
- k-means clustering after discarding bad-scoring models using all-against-all comparisons of Rpb4 and Rpb7 positions

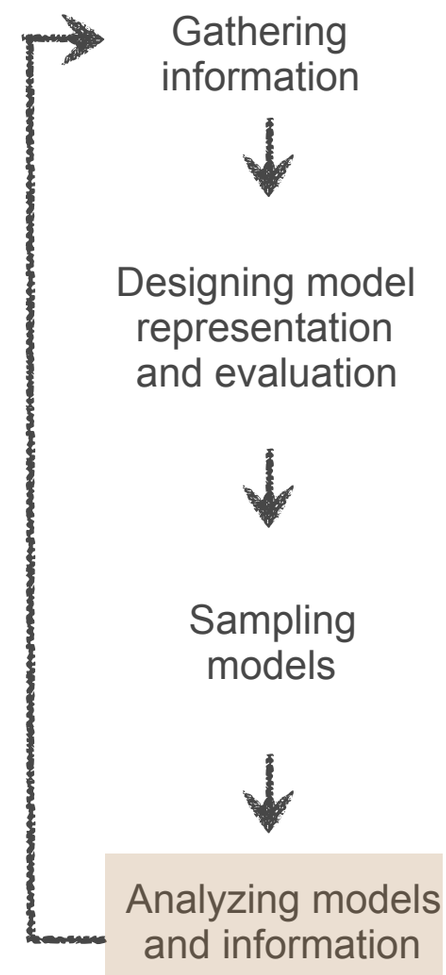
```
num_clusters = 1                        # how many clusters to create
num_top_models = 5                      # total number of best models to analyze
merge_directories = ["../modeling/"]    # directories to analyze
prefiltervalue = 2900.0                 # prefilter by score
```



- Also generates localization densities - maps showing the probability of finding each protein at each point in space - that give a good idea of the “spread” of all models in the cluster

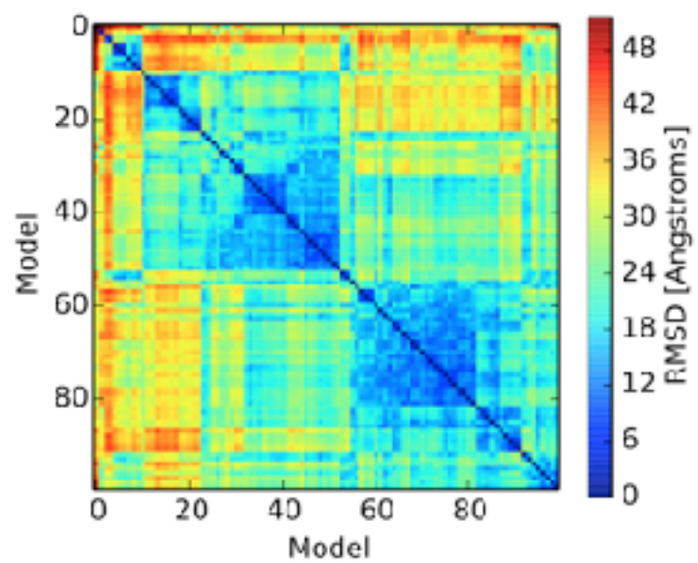
Clustering output

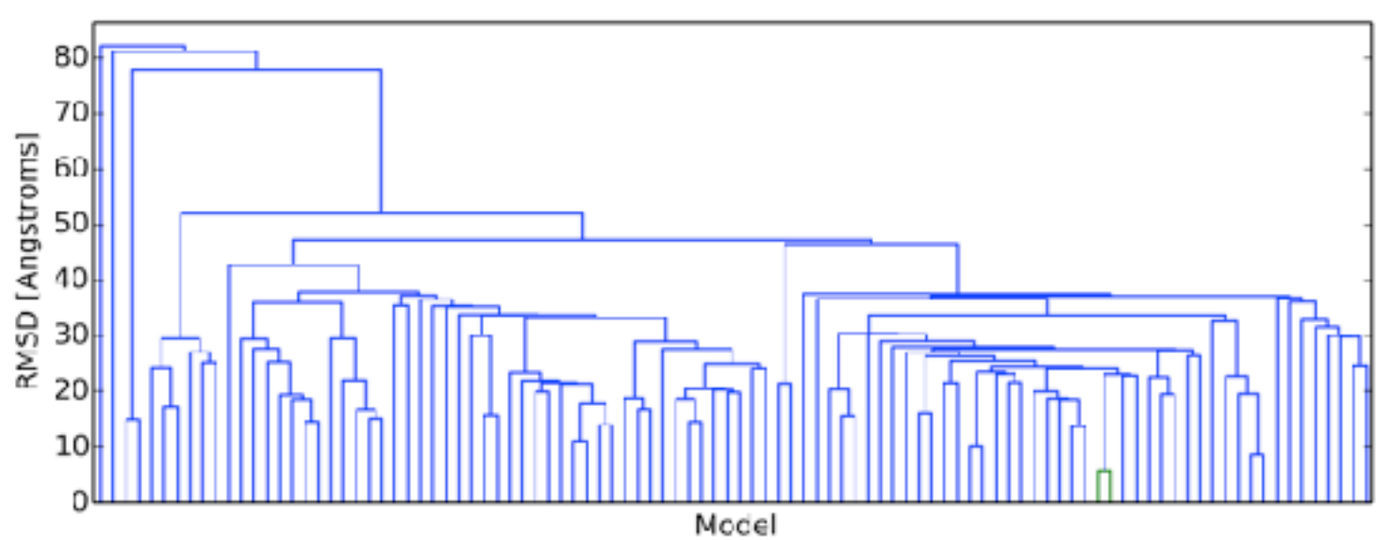
- Outputs won't look great, since we built only 50 models (many of which were discarded by prefiltering)
- Outputs shown here are from a much longer run (overnight) with 2 clusters requested
- Typically cluster representatives and localization densities are reported in publications

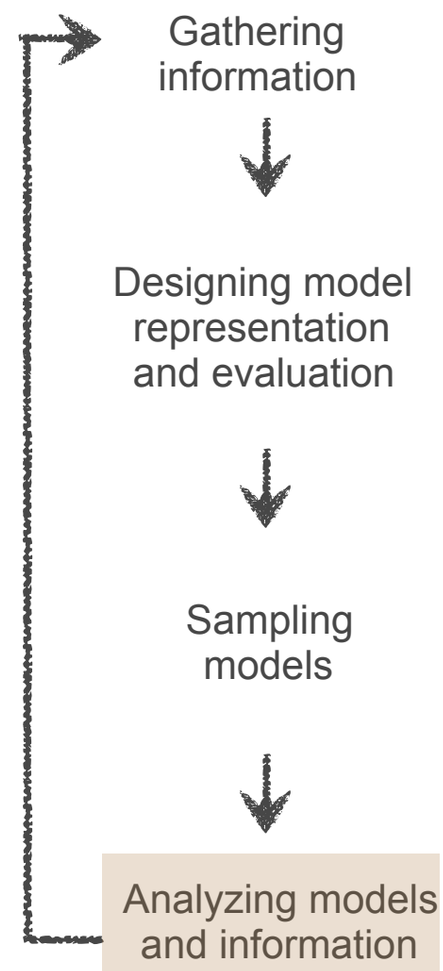


Clustering output

- Distance matrix (dist_matrix.pdf) and dendrogram of the models after being grouped into clusters. The matrix should show the requested number of clusters with much lower within-cluster than between-cluster distance. If this is not the case, then perhaps too many clusters were chosen.

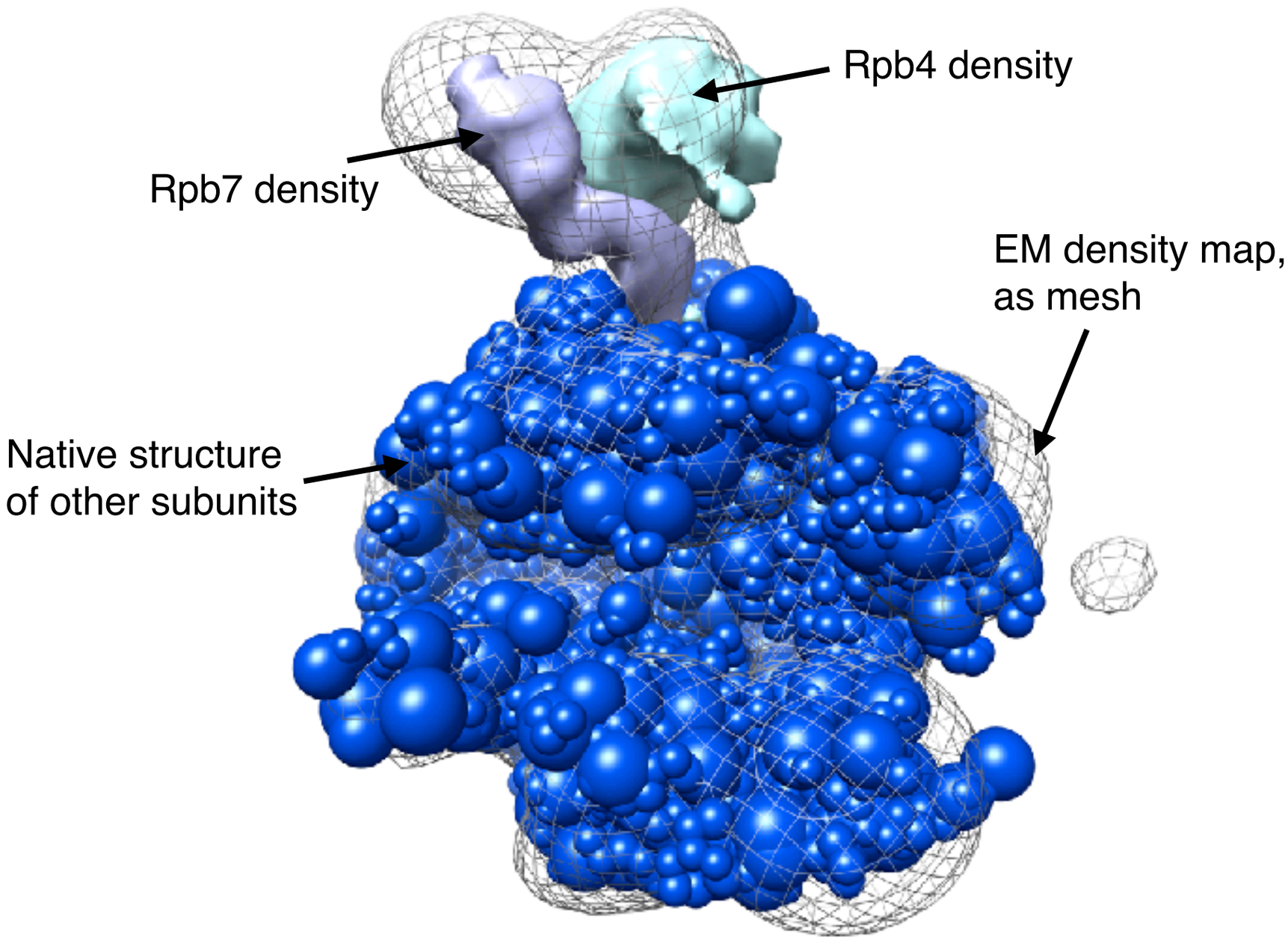


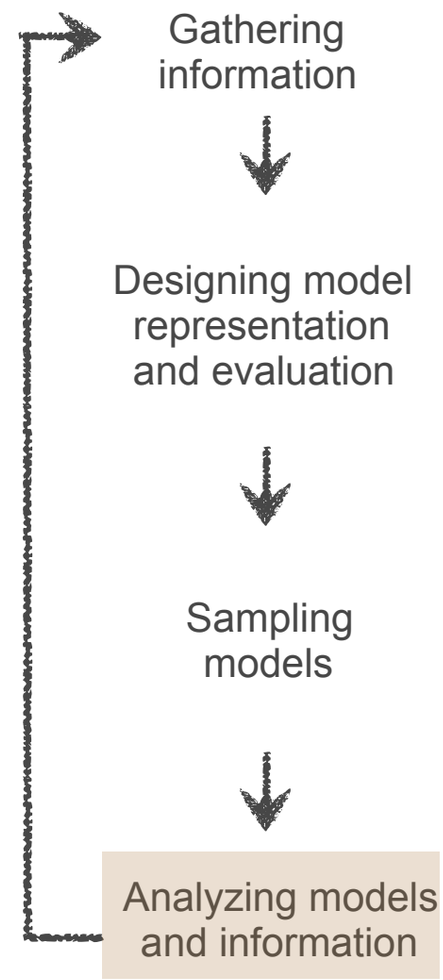




Clustering output

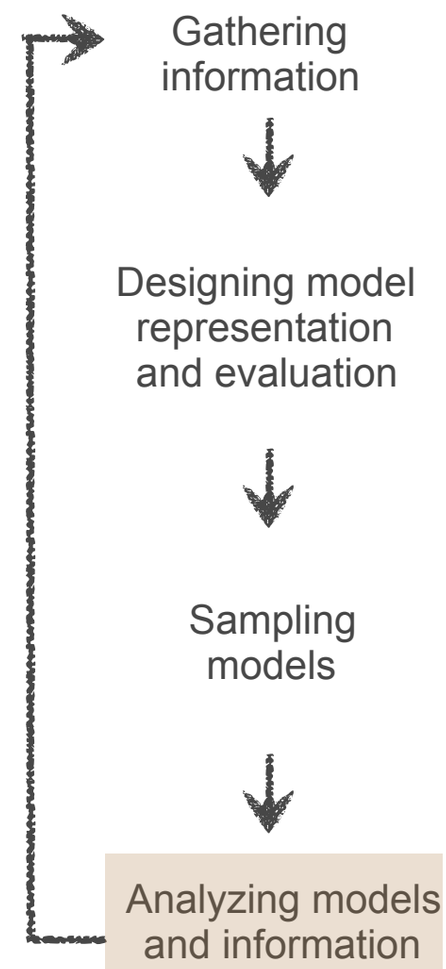
- Localization densities (*.mrc files)
- The localizations are quite narrow and close to native





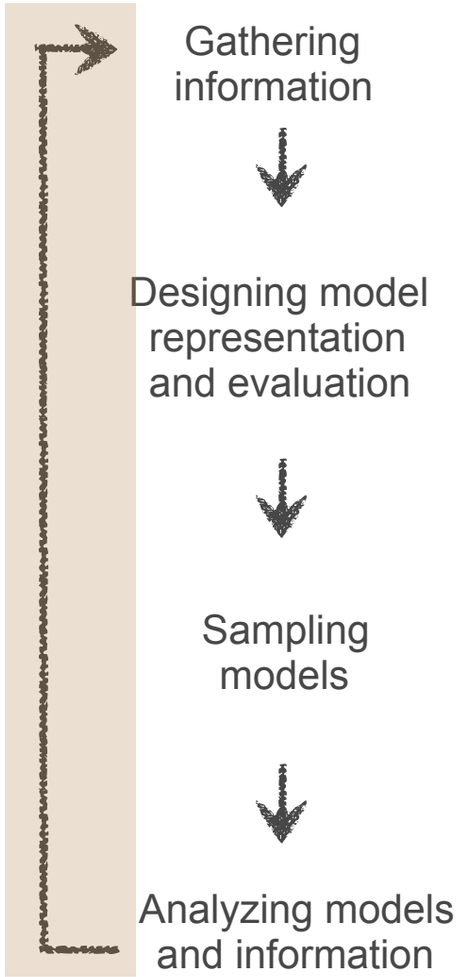
Other analysis

- Cluster precision (`precision_rmsf.py`)
  - shows spread of each cluster
- Accuracy evaluation (`accuracy.py`)
  - compare against known structure
- Sampling exhaustiveness: how can we be sure we've done enough sampling?
  - a variety of methods exist, not covered here today
  - for example, two independent runs should sample from the same distribution - can test statistically, or by comparing clusters
  - can also model leaving out some of the data (jackknife)
  - validate by comparison with data not used in the modeling
  - emergence of patterns not expected by chance



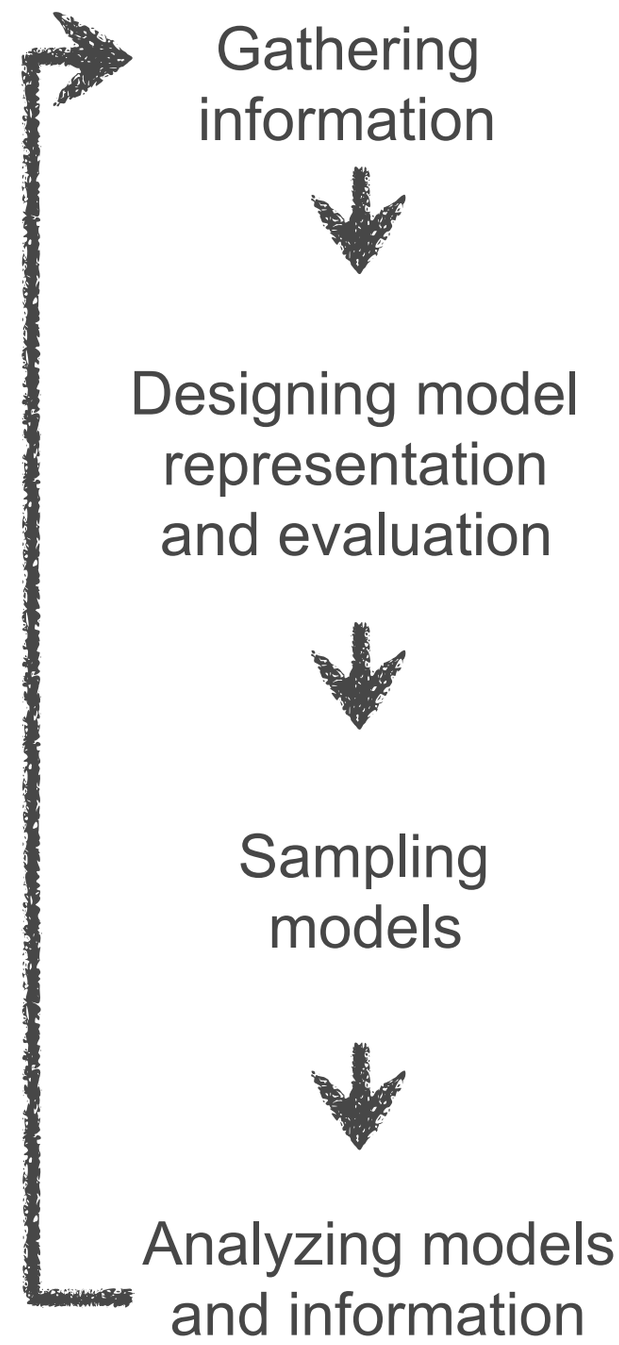
Iteration

- Once we're satisfied that our sampling is complete, we can use the output to suggest new experiments
- For example
  - a high value for a subunit precision suggests we need more intramolecular data (such as crosslinks)
  - clusters where the configuration of certain subunits is ambiguous suggests the need for more protein-protein interaction data involving those subunits
  - e.g. in this case crosslinks were sufficient to get Rpb4 and Rpb7 the 'right way round' in the stalk, but the EM map alone would likely not be



Conclusion

- Integrative modeling provides structural models where individual experimental methods fail
- The Integrative Modeling Platform (IMP) is a toolbox for solving integrative modeling problems
- Generate multi-scale (also multi-state, time ordered) ensembles of models consistent with multiple sources of information



https://integrativemodeling.org/

D. Russel, K. Lasker, B. Webb, J. Velazquez-Muriel, E. Tjioe, D. Schneidman, F. Alber, B. Peterson, A. Sali, PLoS Biol, 2012.
R. Pellarin, M. Bonomi, B. Raveh, S. Calhoun, C. Greenberg, G.Dong.