

IMP Software Introduction & Demonstration

**Benjamin Webb,
Sali Lab,
University of California San Francisco
(ben@salilab.org)**

Why Protein Structure **Prediction**?

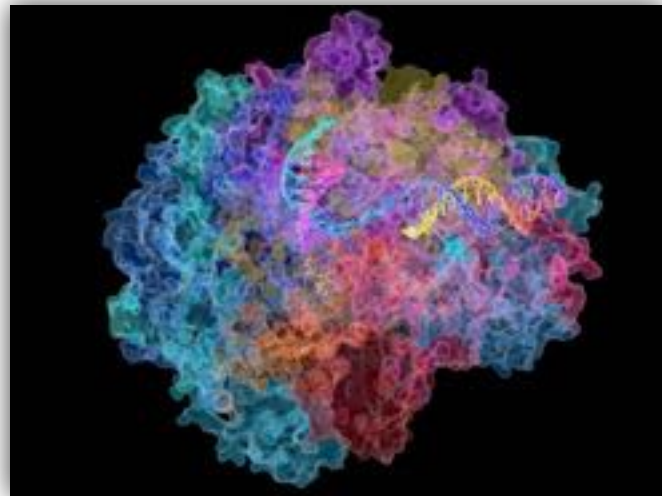
	Y 2016
Sequences	70,650,000
Structures	128,000

- 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.

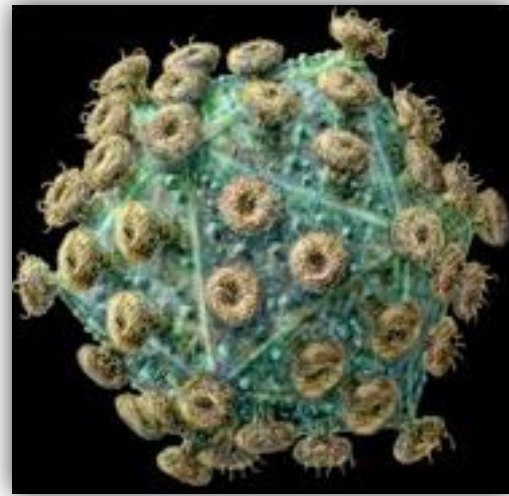
Structural biology:

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

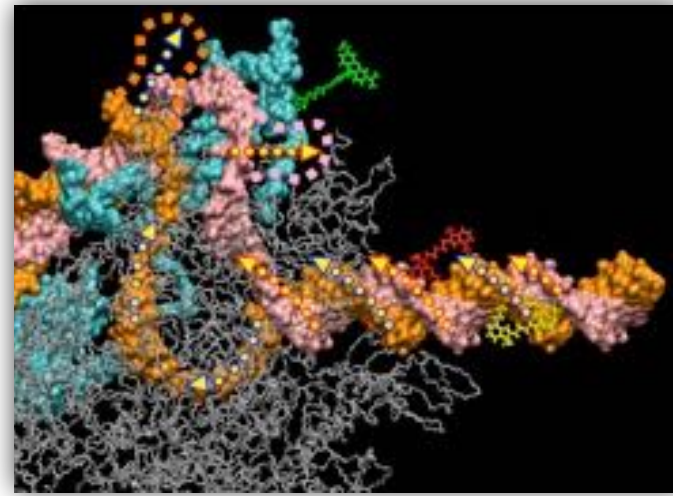
Motivation: Models will allow us to understand how machines work, how they evolved, how they can be controlled, modified, and perhaps even designed.



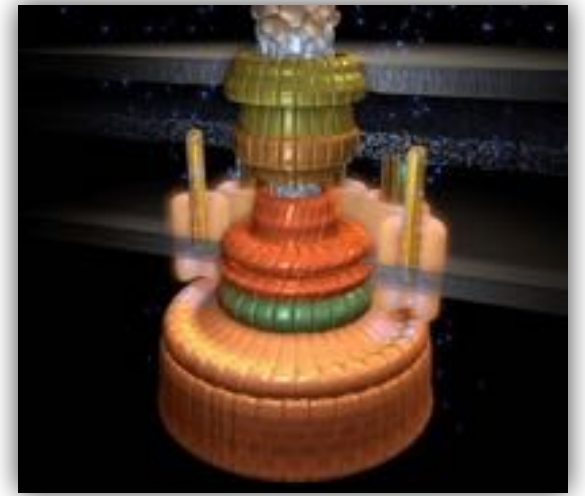
RNA polymerase II



HIV virus



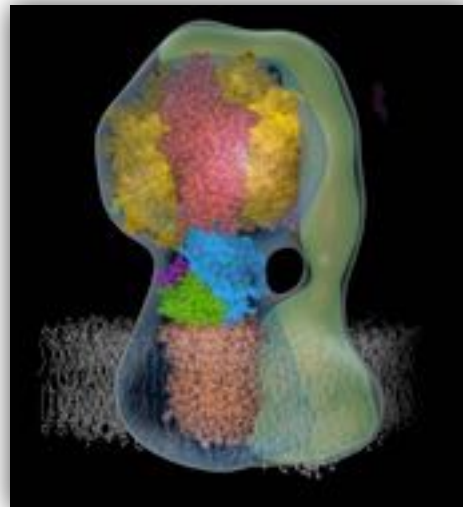
tRNA synthetase



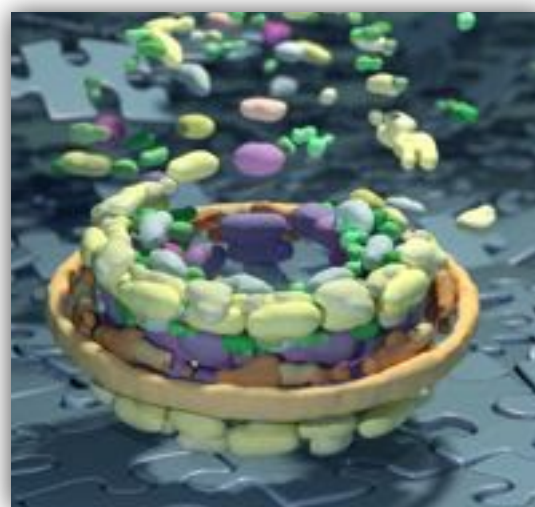
flagellar motor



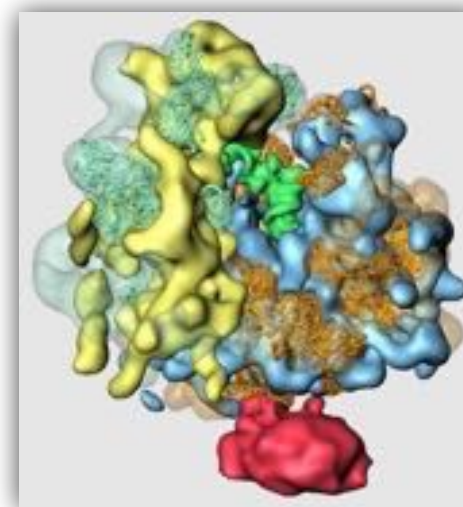
GroEL chaperonin



ATP synthase



nuclear pore complex



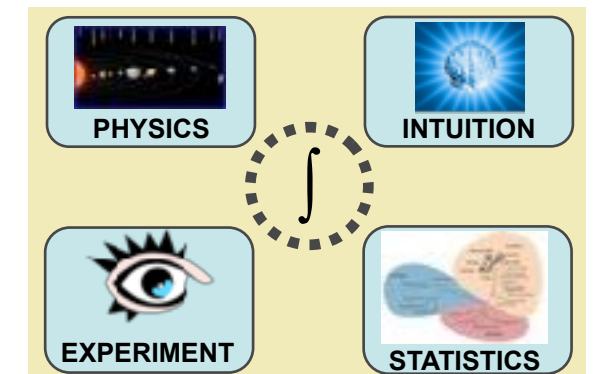
ribosome

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.



X-ray crystallography	NMR spectroscopy	2D & single particle electron microscopy	electron tomography	immuno-electron microscopy	chemical cross-linking	affinity purification mass spectroscopy
subunit structure	subunit structure				subunit structure	
subunit shape	subunit shape	subunit shape	subunit shape			
subunit-subunit contact	subunit-subunit contact	subunit-subunit contact	subunit-subunit contact		subunit-subunit contact	subunit-subunit contact
subunit proximity	subunit proximity	subunit proximity	subunit proximity	subunit proximity	subunit proximity	subunit proximity
subunit stoichiometry	subunit stoichiometry					
assembly symmetry	assembly symmetry	assembly symmetry	assembly symmetry	assembly symmetry		
assembly shape	assembly shape	assembly shape	assembly shape			
assembly structure	assembly structure					

FRET	site-directed mutagenesis	yeast two-hybrid system	gene/protein arrays	protein structure prediction	computational docking	bioinformatics
				subunit structure		
				subunit shape		
subunit-subunit contact	subunit-subunit contact	subunit-subunit contact	subunit-subunit contact		subunit-subunit contact	subunit-subunit contact
subunit proximity		subunit proximity	subunit proximity			

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.

A description of integrative structure determination

Alber et al. *Nature* **450**, 683-694, 2007

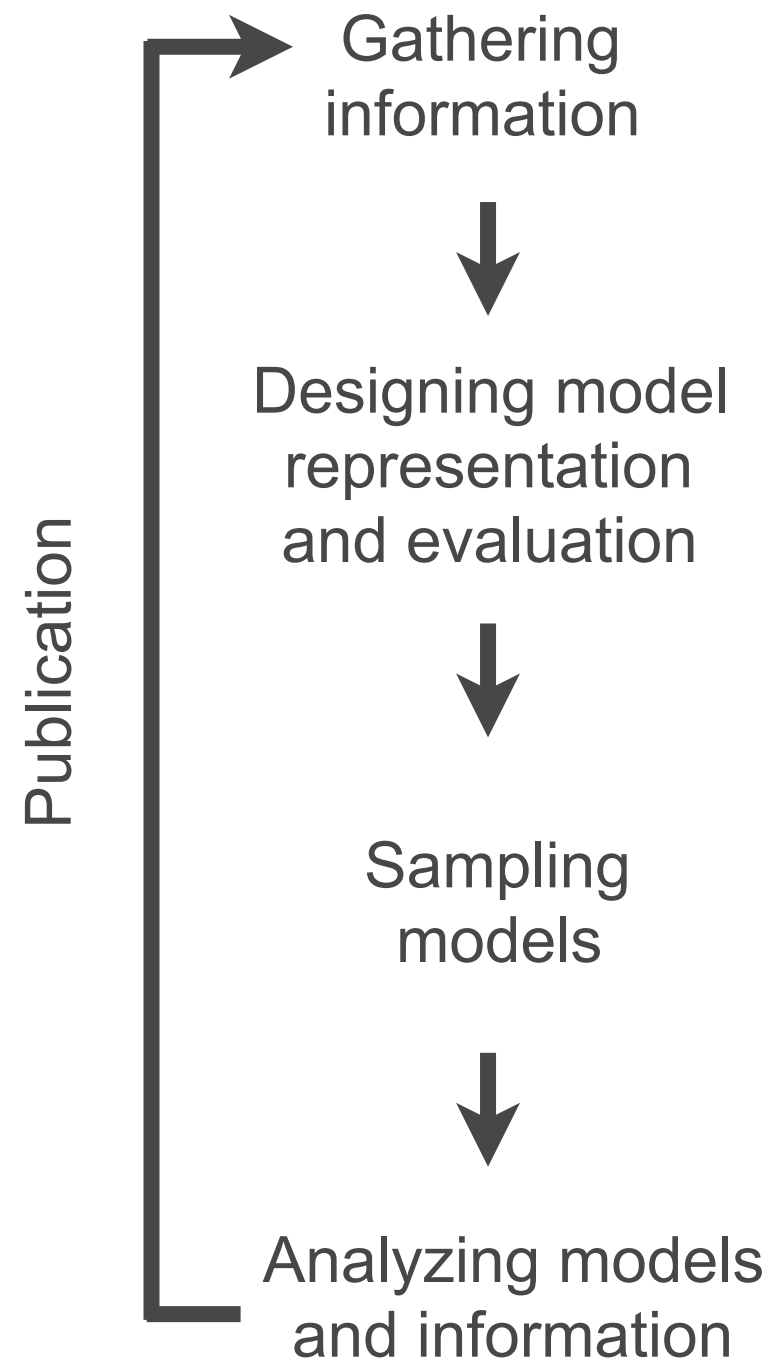
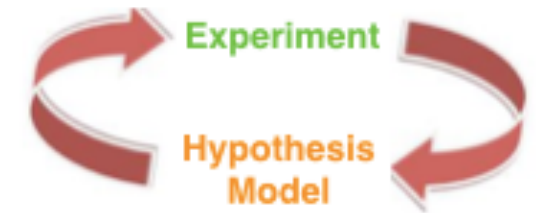
Robinson et al. *Nature* **450**, 974-982, 2007

Alber et al. *Annual Reviews in Biochemistry* **77**, 11.1–11.35, 2008

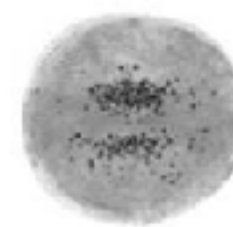
Russel et al. *PLoS Biology* **10**, 2012

Ward et al. *Science* **339**, 913-915, 2013

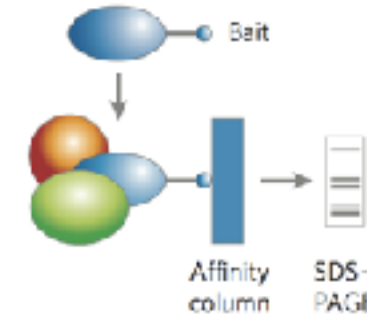
Schneidman et al. *Curr.Opin.Str.Biol.*, 2014.



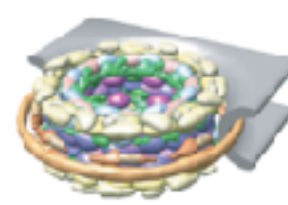
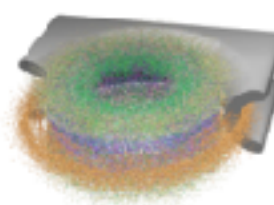
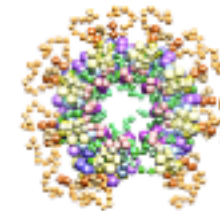
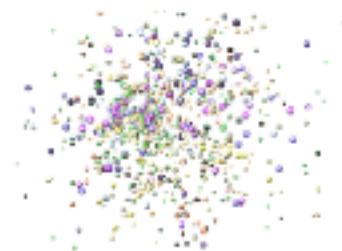
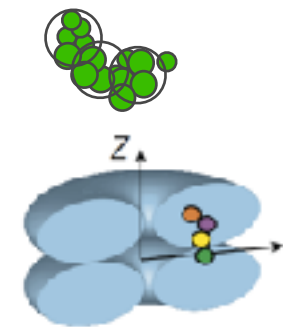
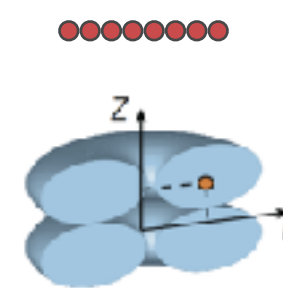
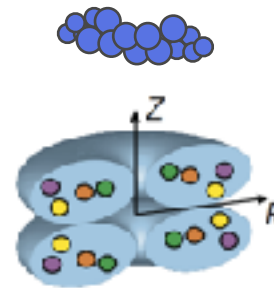
Cryo-electron microscopy



Immuno-electron microscopy



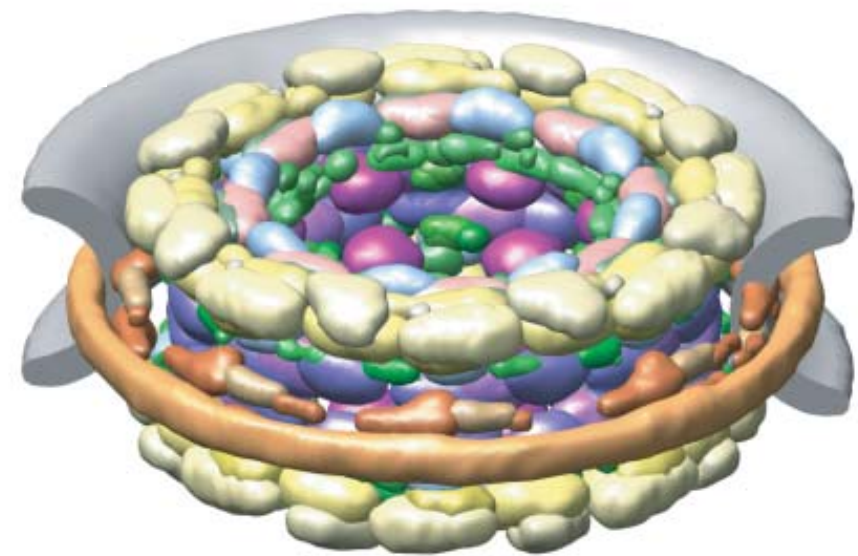
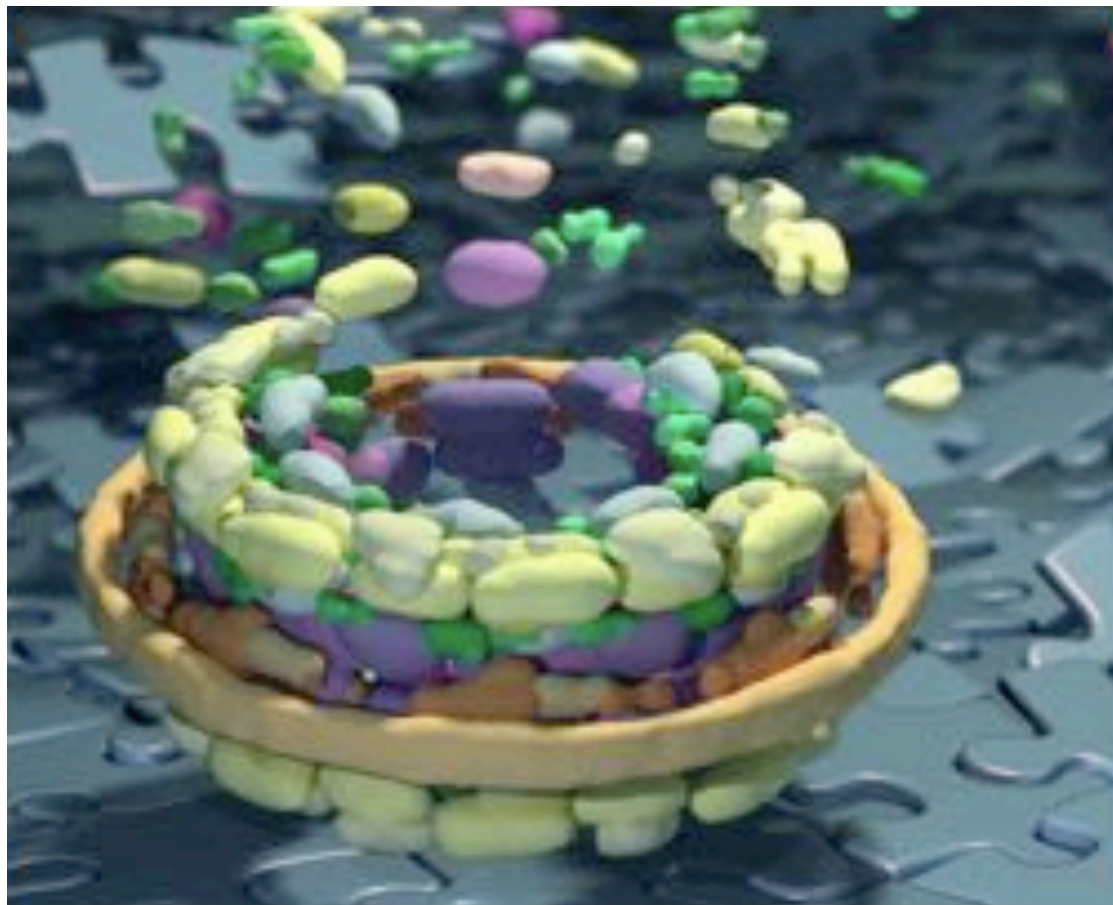
Affinity purification



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

Final 2007 NPC model

- Sufficient to place each *protein* within the entire complex
- Very coarse-grained model; no atomic information



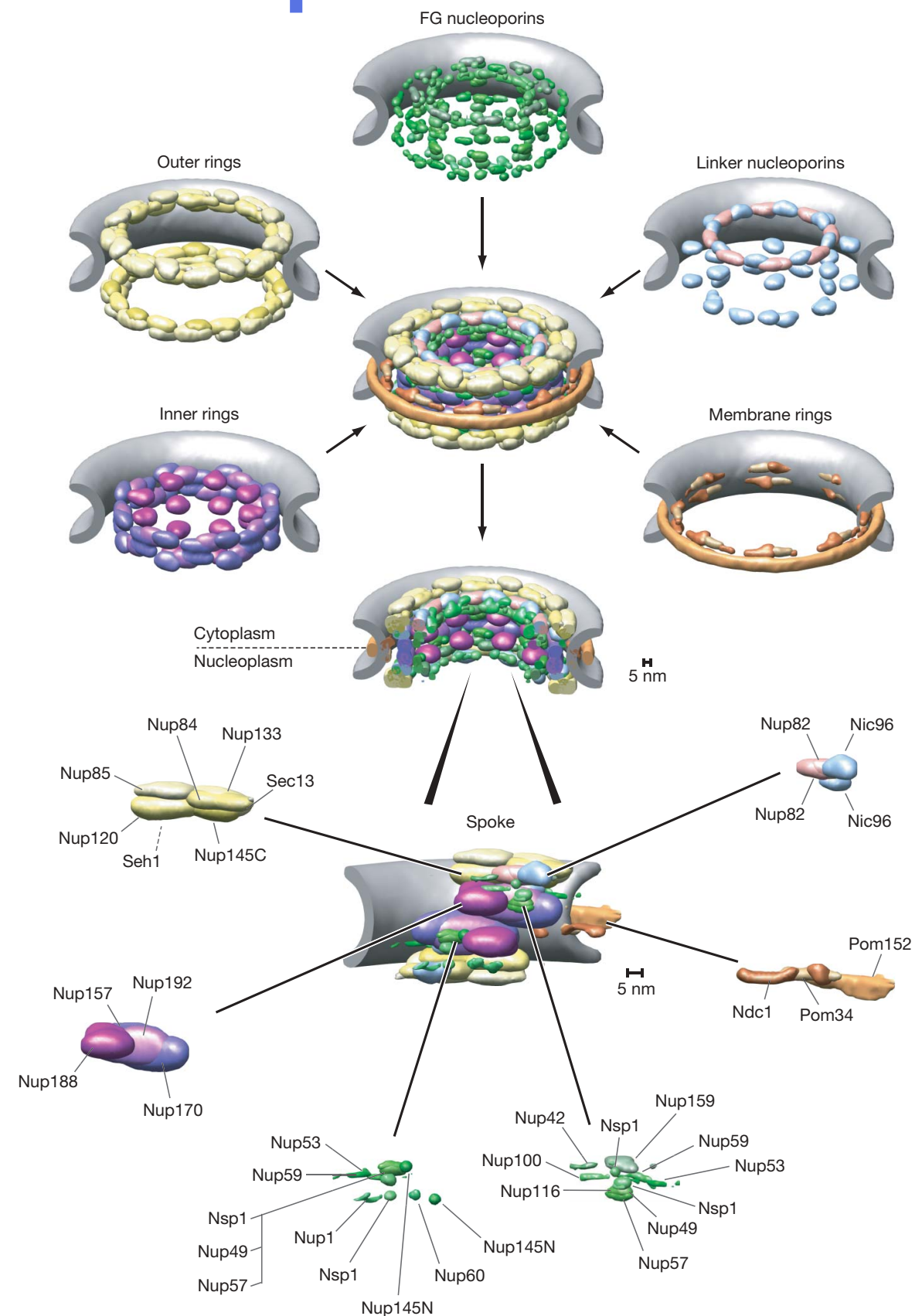
Alber *et al.* Nature 450, 684-694, 2007

Alber *et al.* Nature 450, 695-702, 2007

with M. Rout & B. Chait

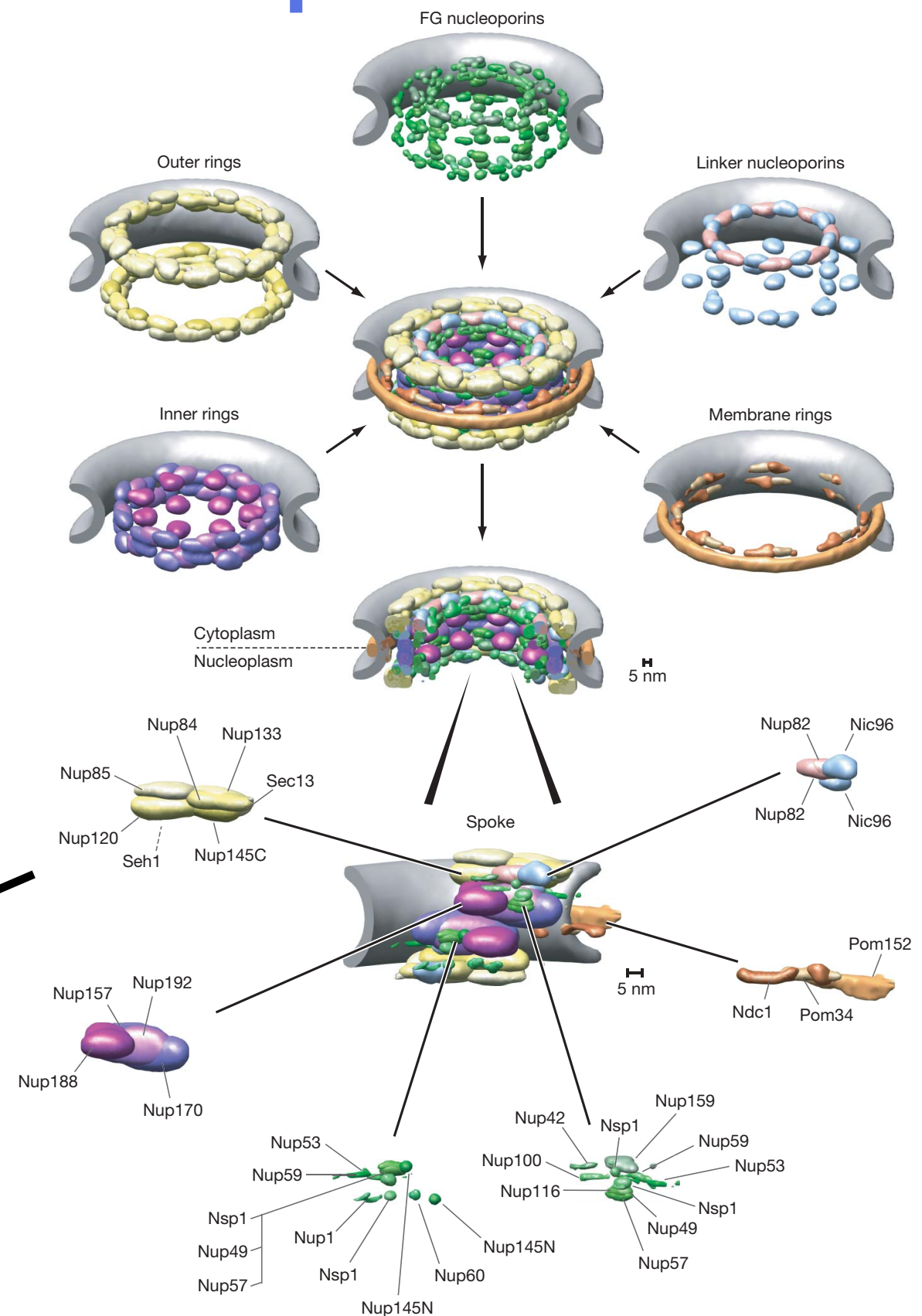
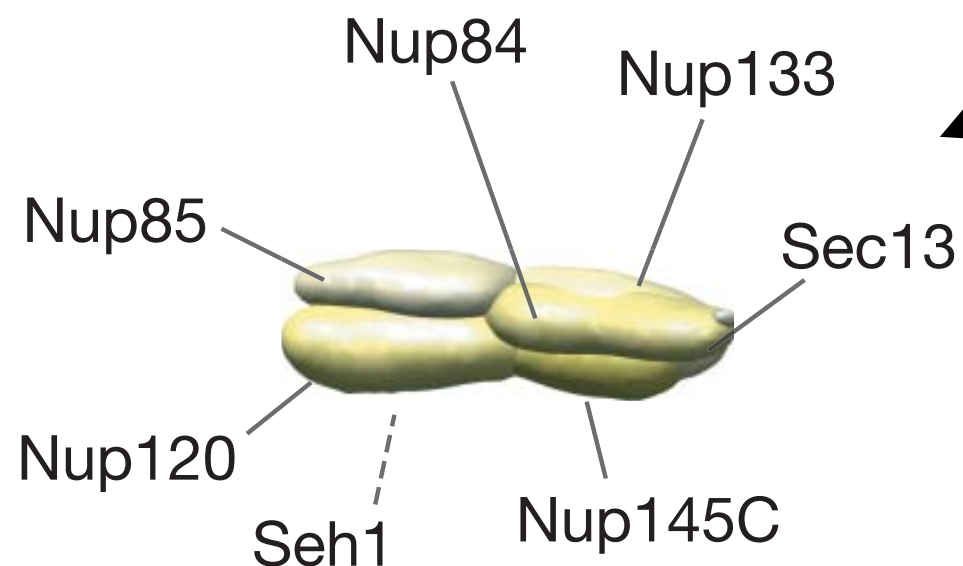
Nup84 subcomplex

- Look at subcomplexes, towards a higher resolution structure of the entire NPC
- Nup84 is one such subcomplex of 7 proteins, 16 copies of which form the outer ring

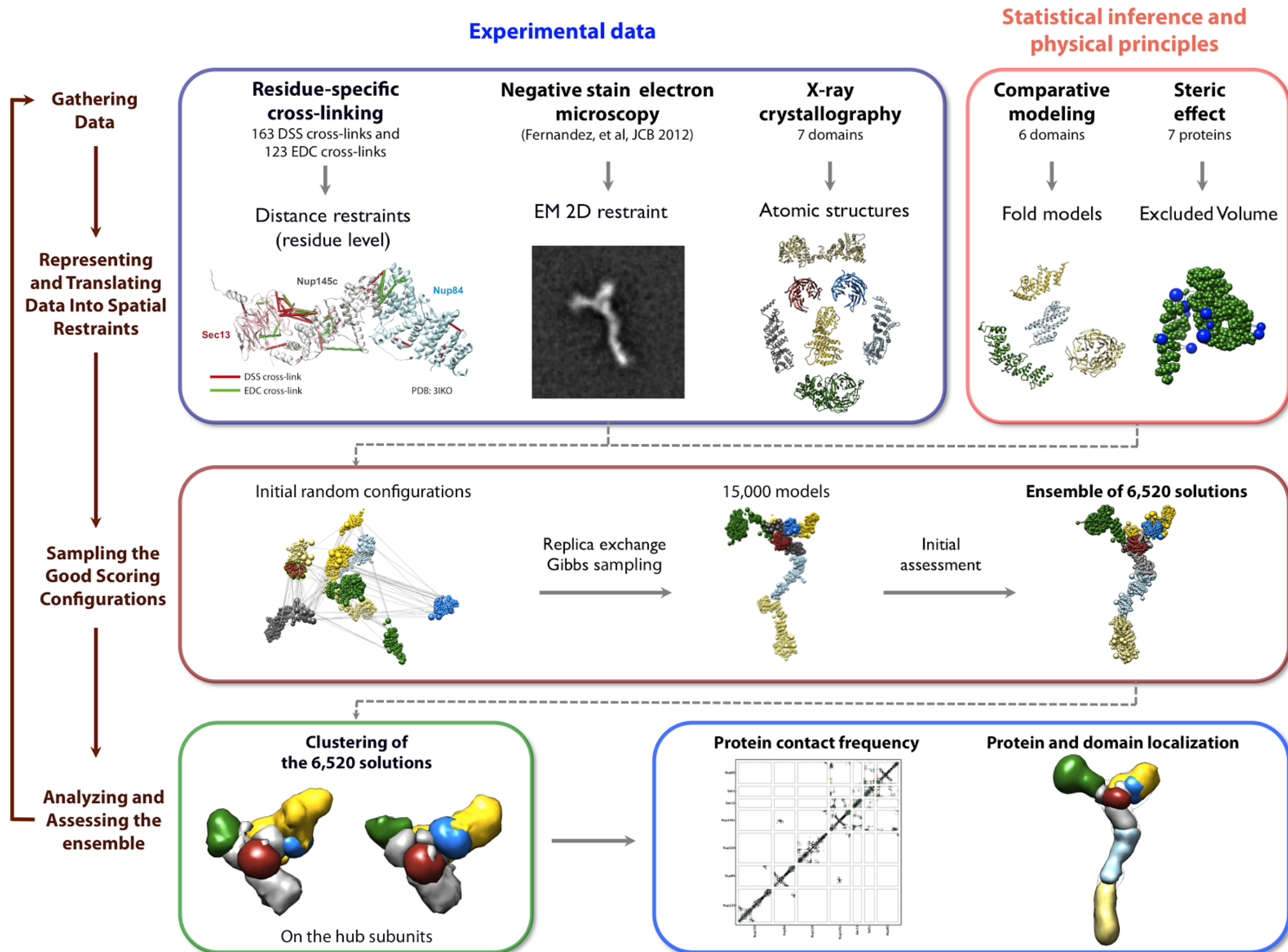


Nup84 subcomplex

- Look at subcomplexes, towards a higher resolution structure of the entire NPC
- Nup84 is one such subcomplex of 7 proteins, 16 copies of which form the outer ring



Modeling Nup84 with IMP (2014)



Deposition in PDB-Dev

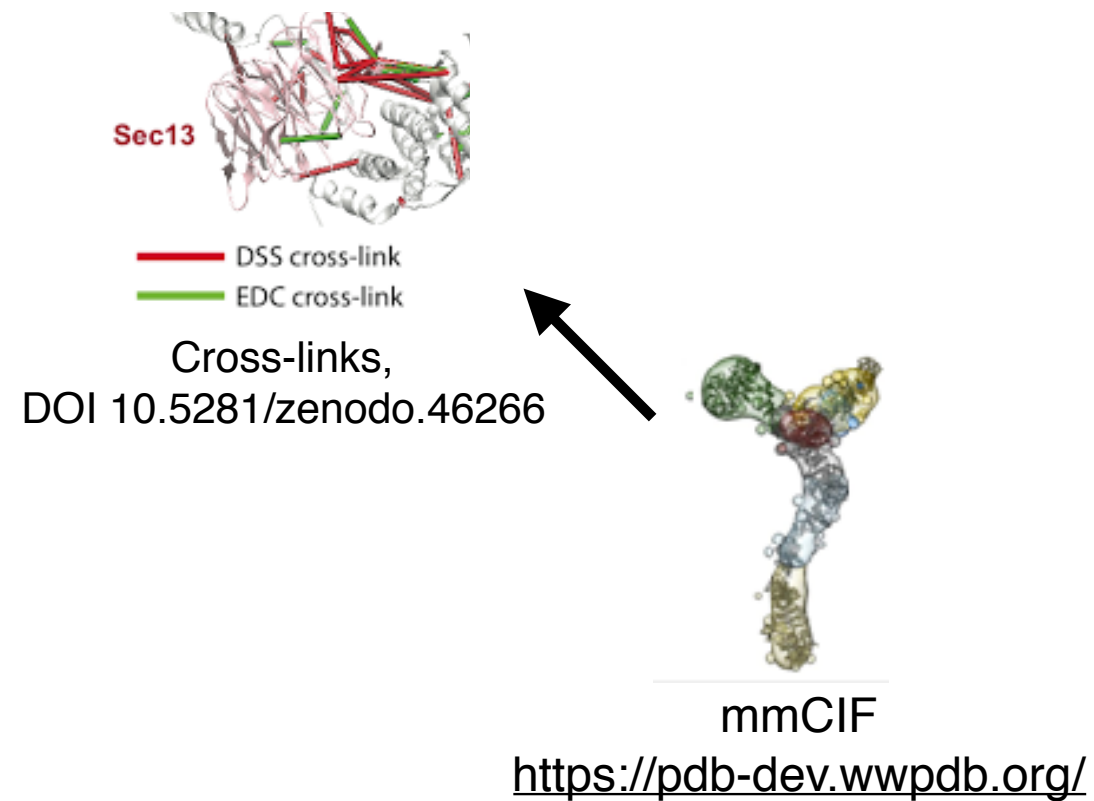


mmCIF

<https://pdb-dev.wwpdb.org/>

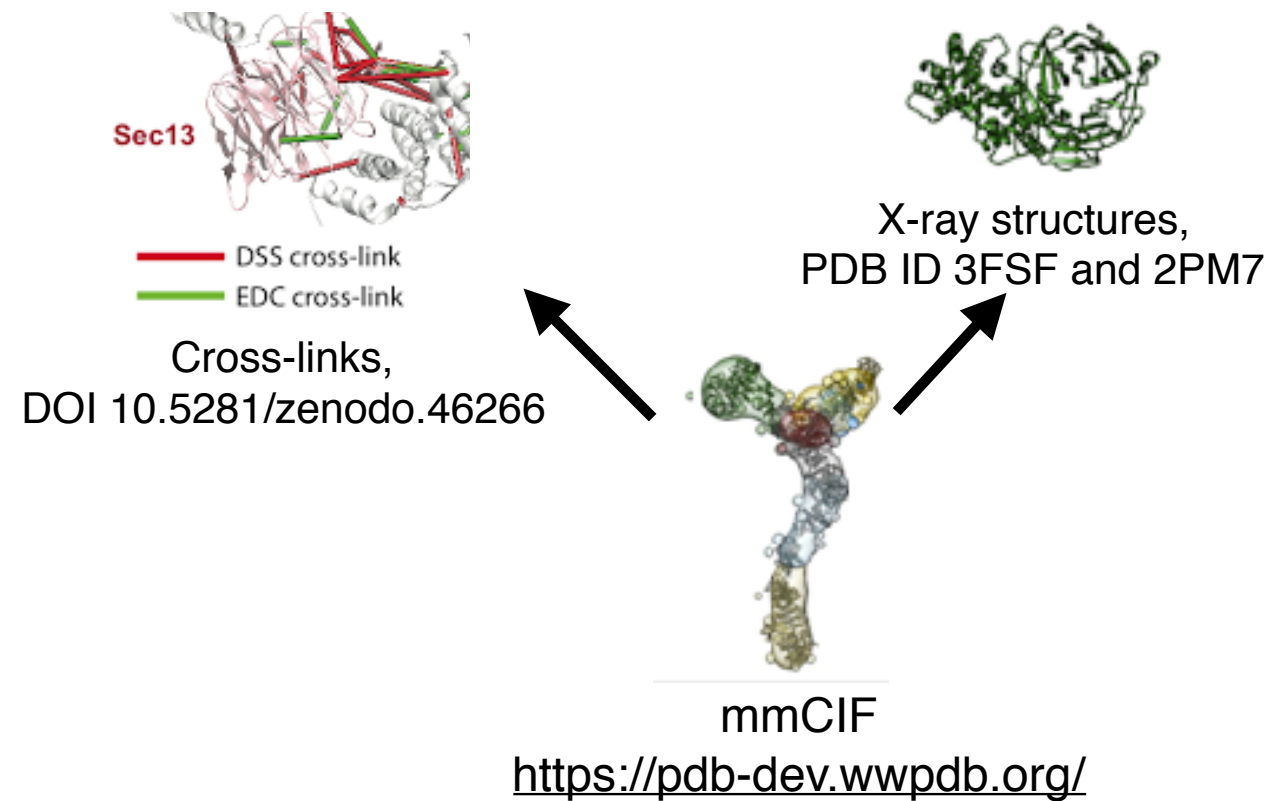
- 3D integrative models
- Multi-state, multi-scale, time-ordered
- Modeling protocol
- Input information
- Localization densities, ensembles
- Validation

Deposition in PDB-Dev



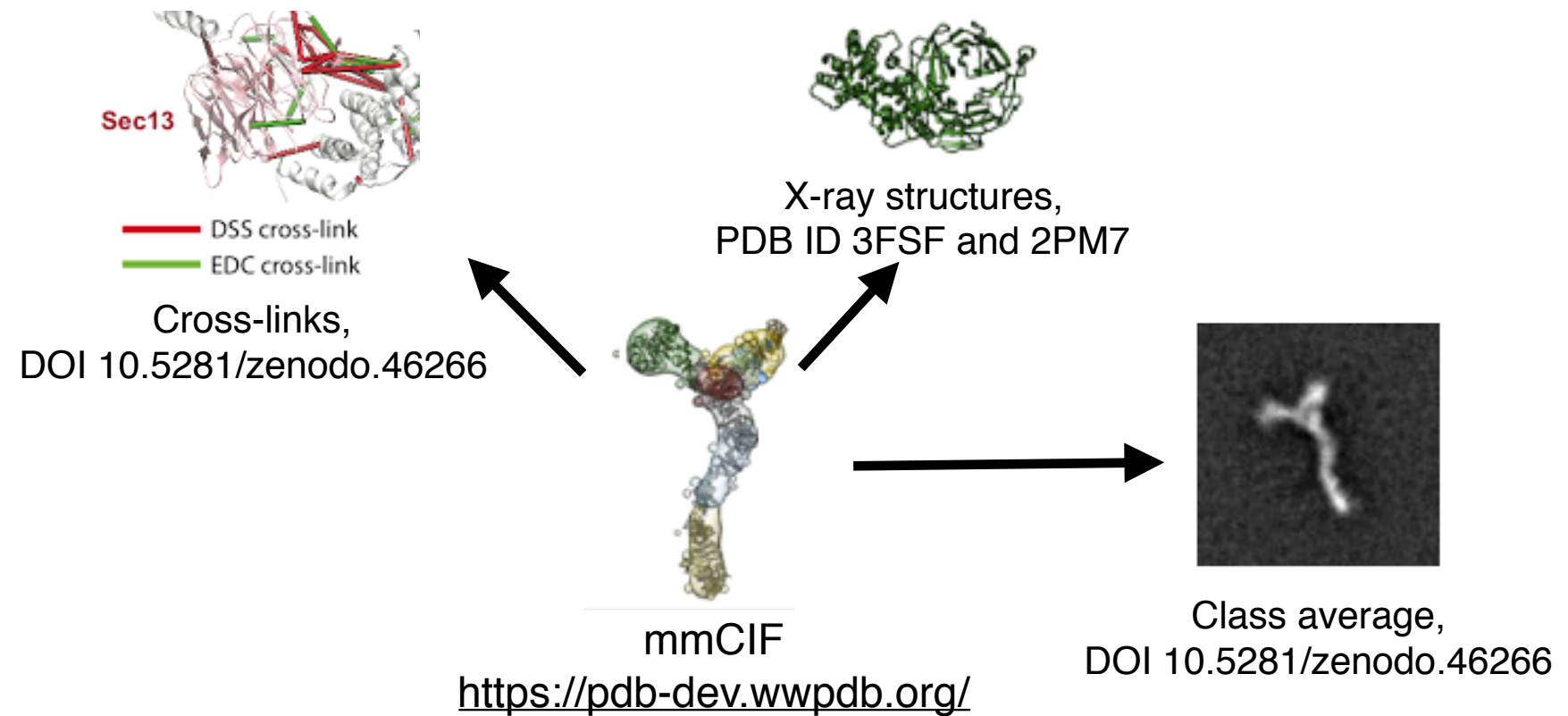
- 3D integrative models
- Multi-state, multi-scale, time-ordered
- Modeling protocol
- Input information
- Localization densities, ensembles
- Validation

Deposition in PDB-Dev



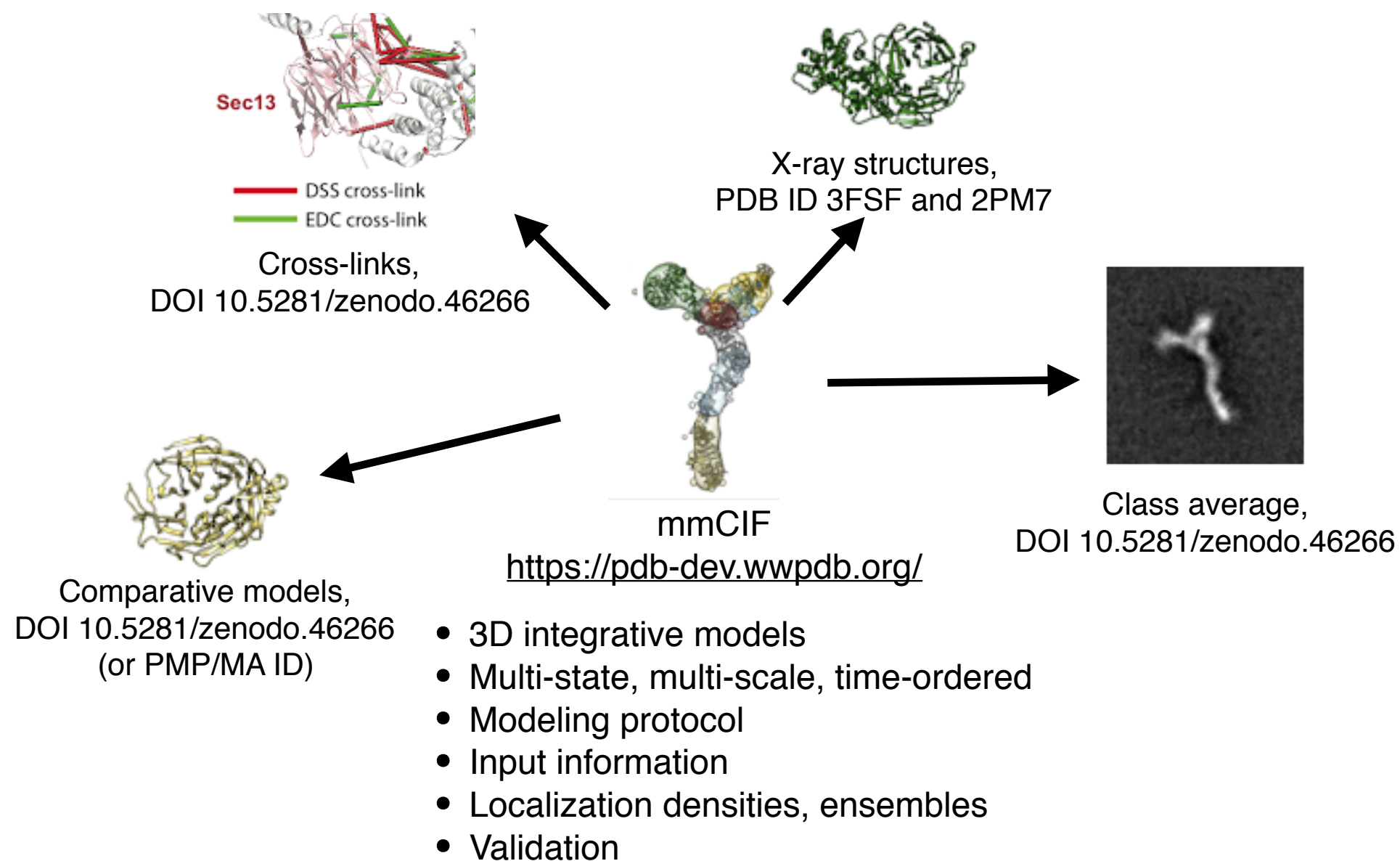
- 3D integrative models
- Multi-state, multi-scale, time-ordered
- Modeling protocol
- Input information
- Localization densities, ensembles
- Validation

Deposition in PDB-Dev

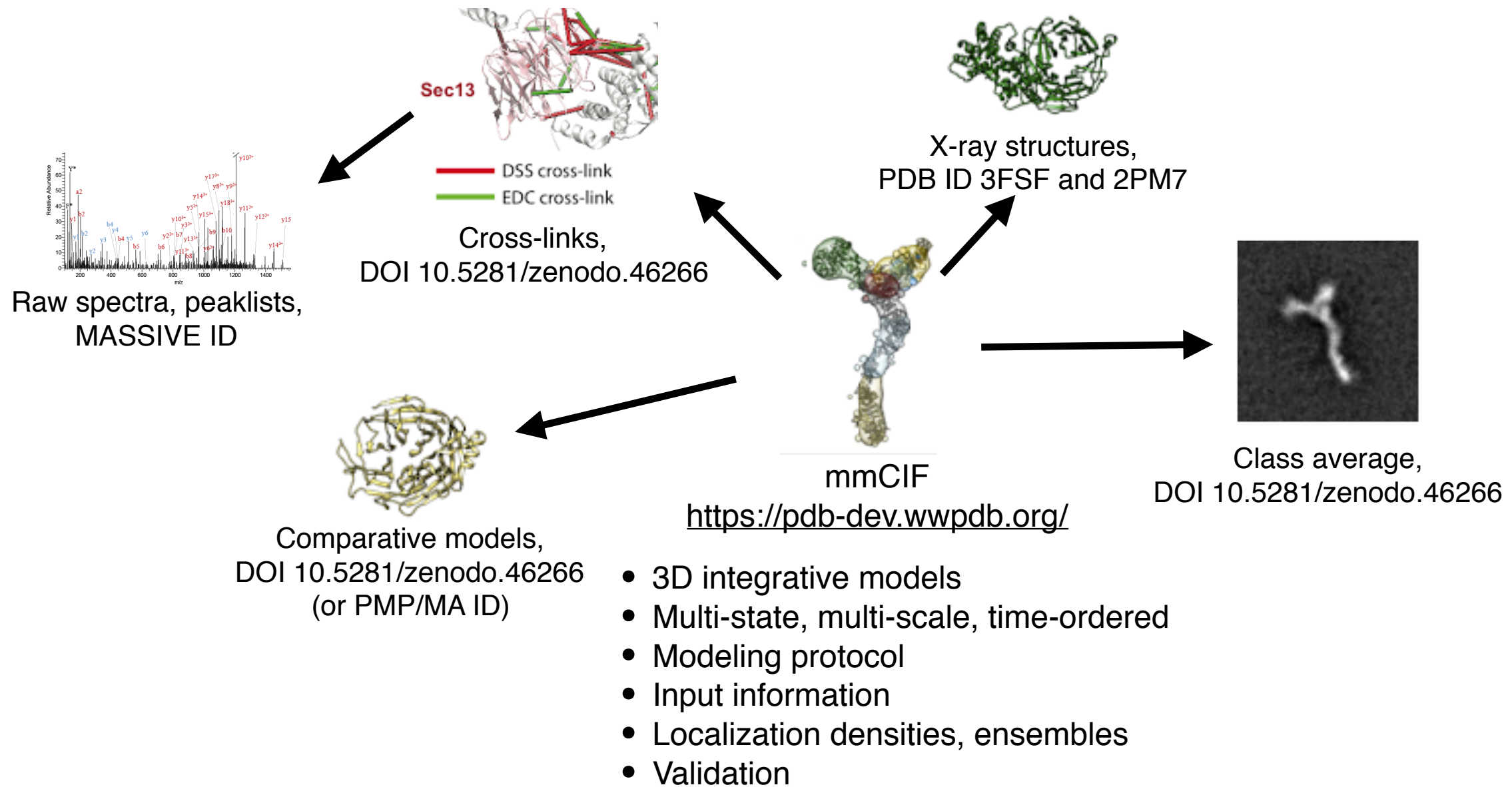


- 3D integrative models
- Multi-state, multi-scale, time-ordered
- Modeling protocol
- Input information
- Localization densities, ensembles
- Validation

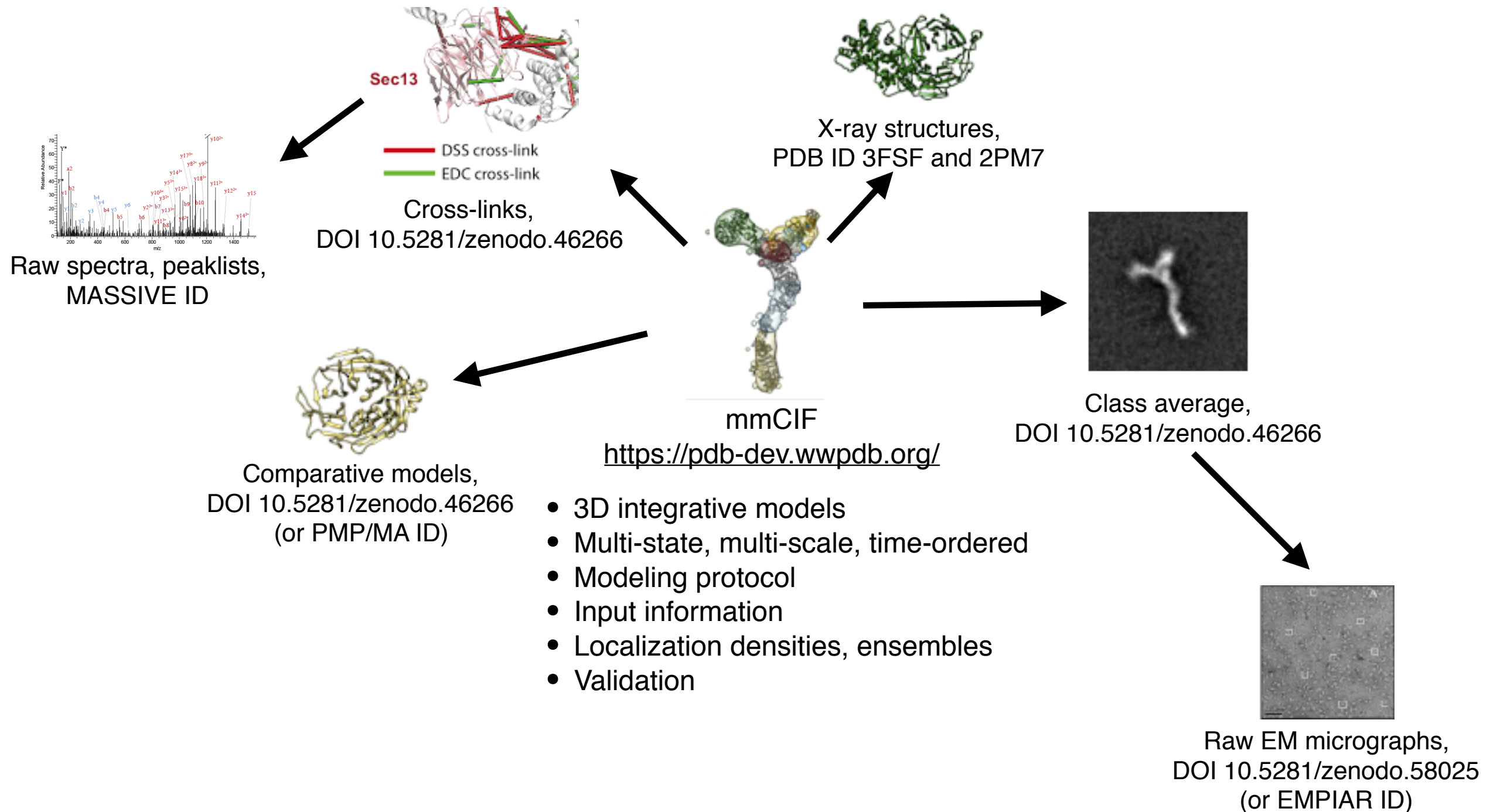
Deposition in PDB-Dev



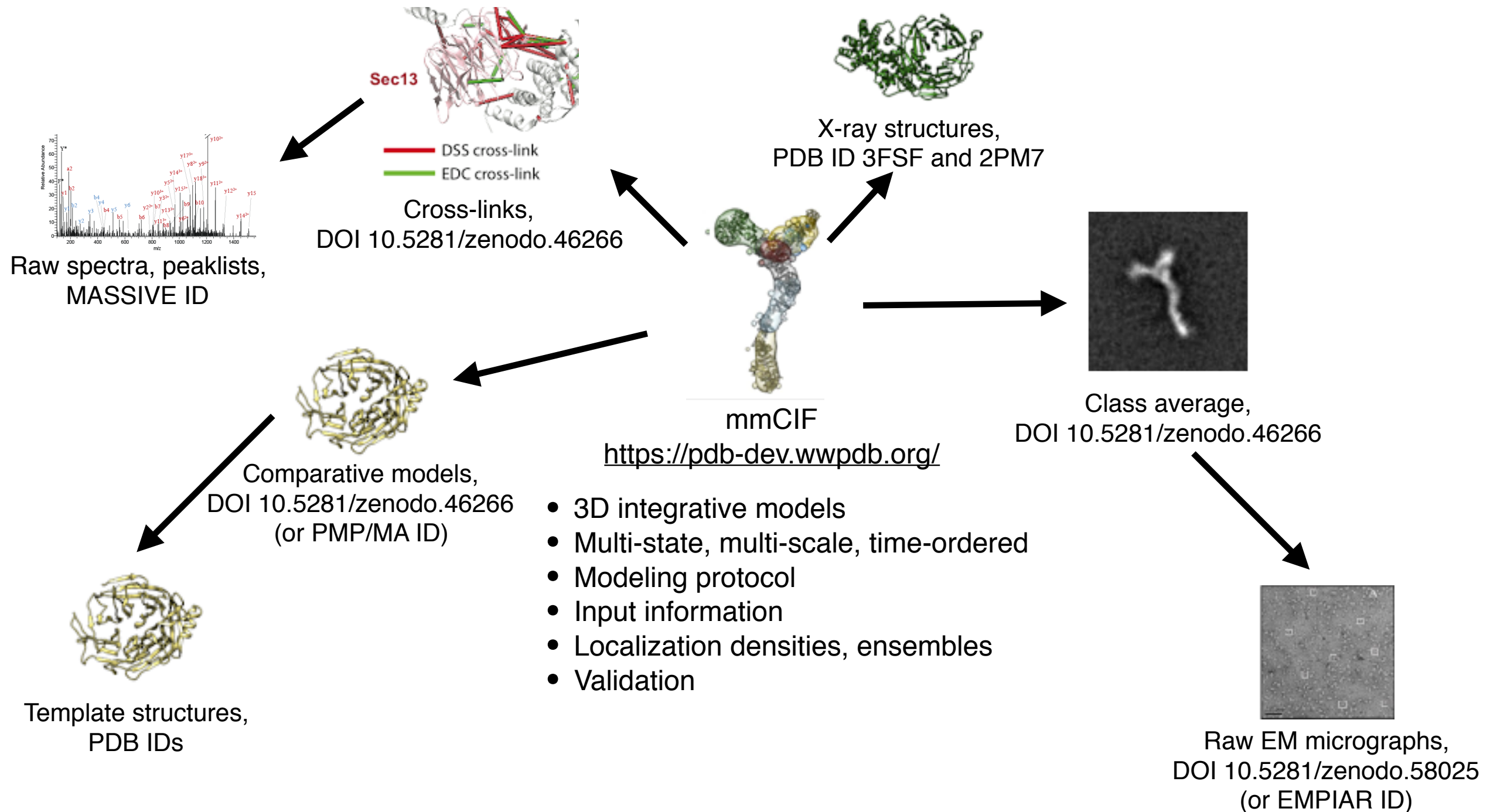
Deposition in PDB-Dev



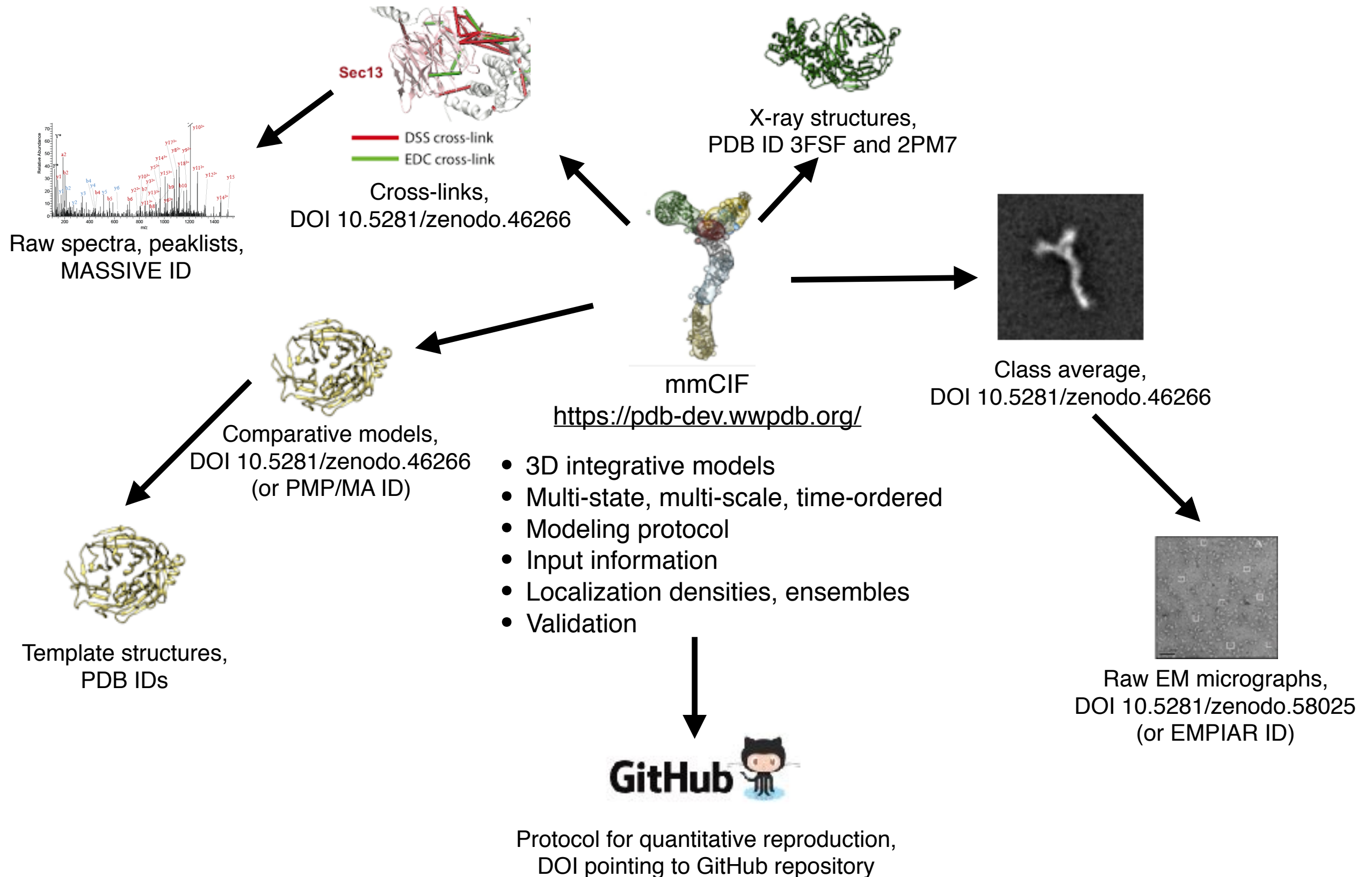
Deposition in PDB-Dev



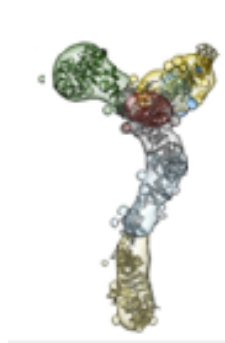
Deposition in PDB-Dev



Deposition in PDB-Dev



Our studies in PDB-Dev



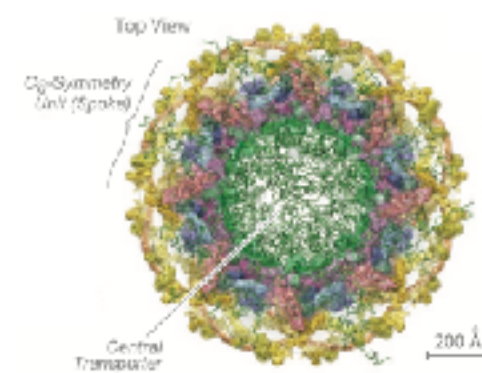
Nup84



Mediator



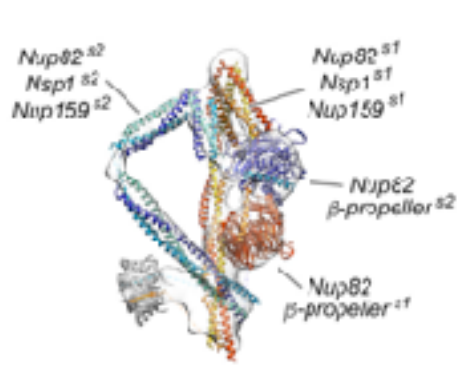
Exosome



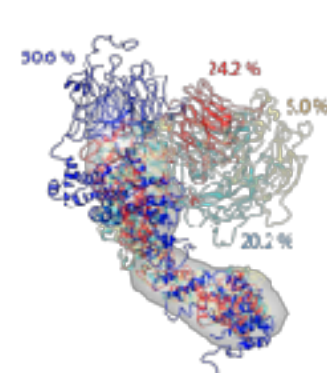
NPC



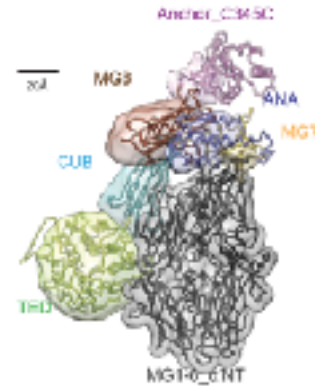
Pom152



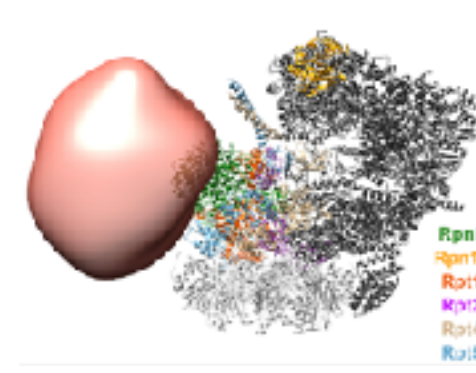
Nup82



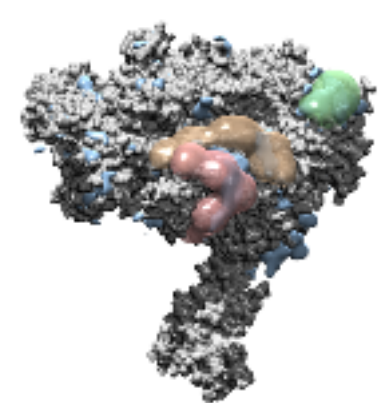
Nup133



Complement



Ecm29

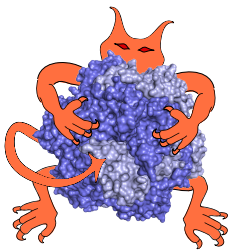


Pol II (G)

Going forward we expect to deposit **all** of our integrative modeling applications in PDB-Dev

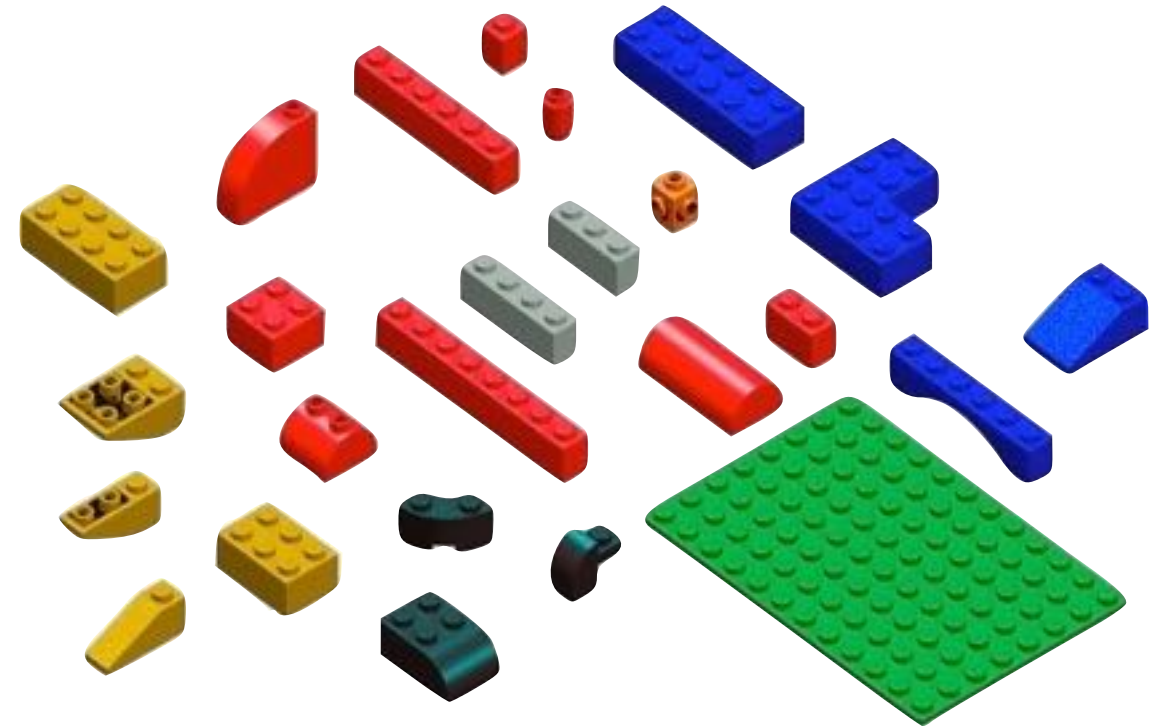
Integrative Modeling Platform (IMP)

<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, S.J. Kim, D. Saltzberg, I. Chemmama, S. Axen, S. Viswanath.

- 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
Time-ordered 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, ...)

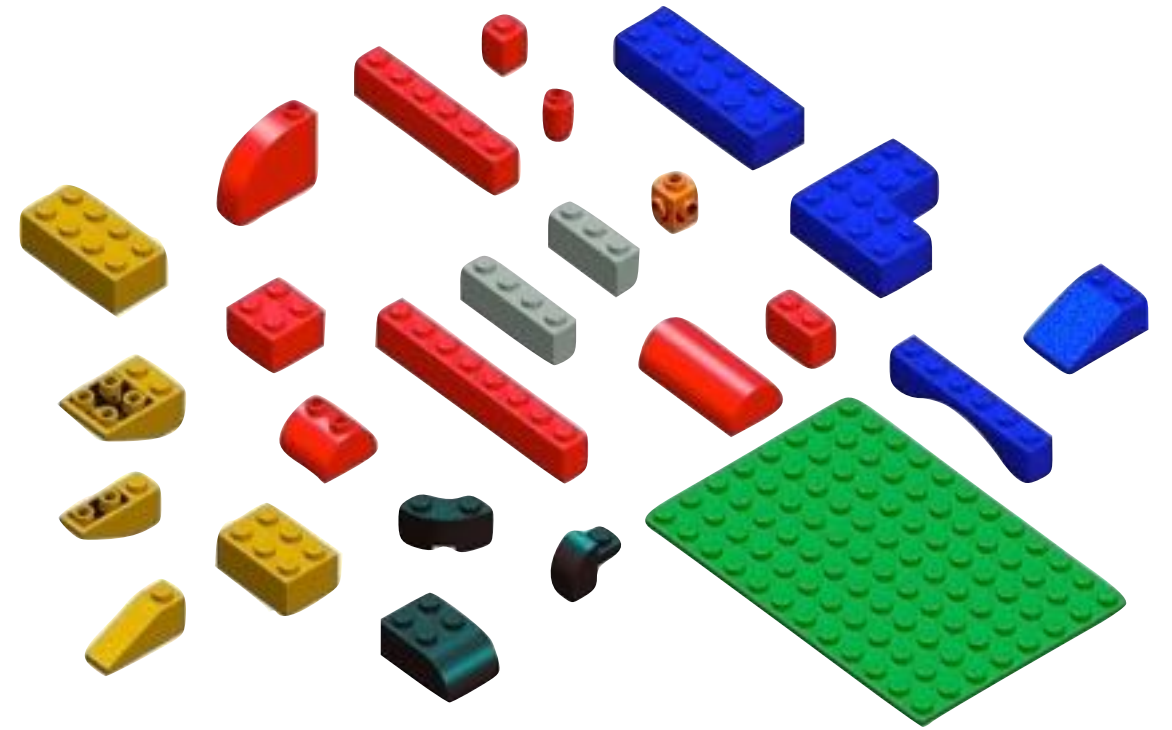
Sampling:

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

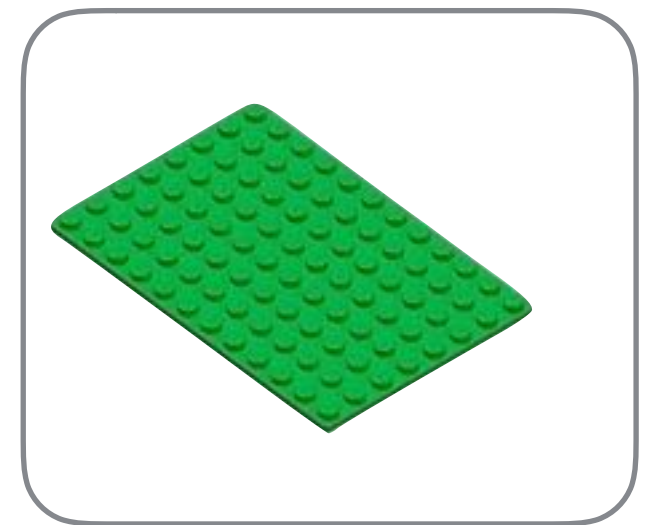
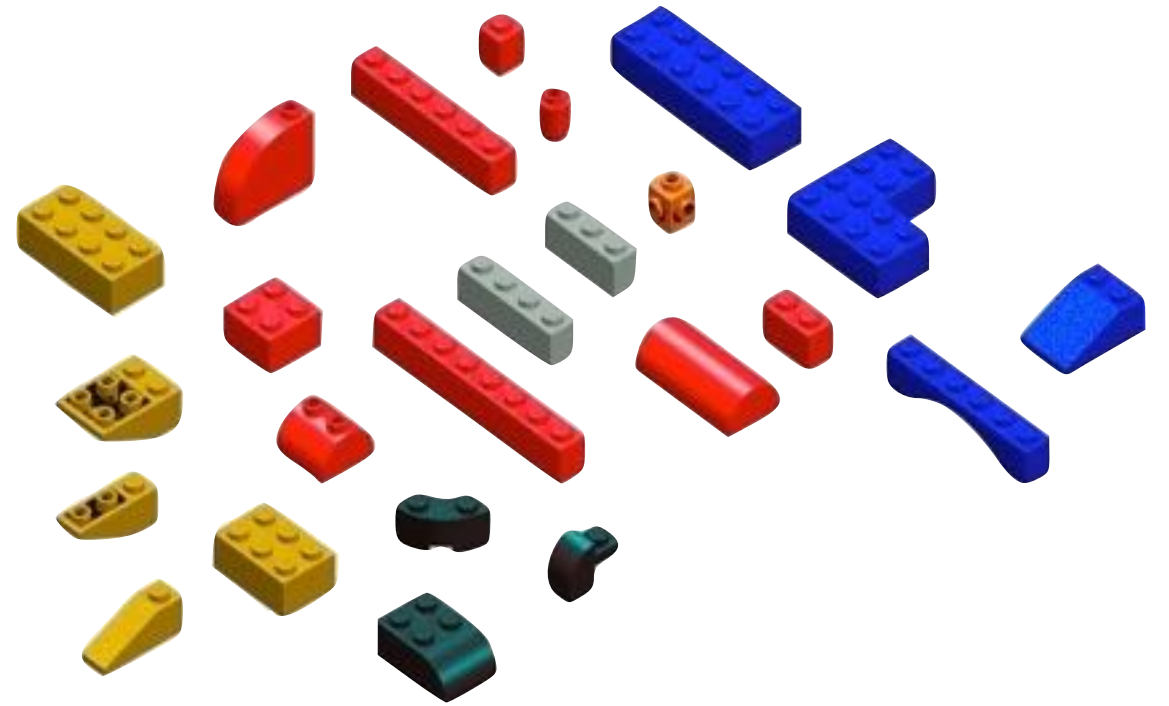
Analysis:

Clustering
Chimera
PyMOL
PDB files
Density maps

IMP is divided into *modules*

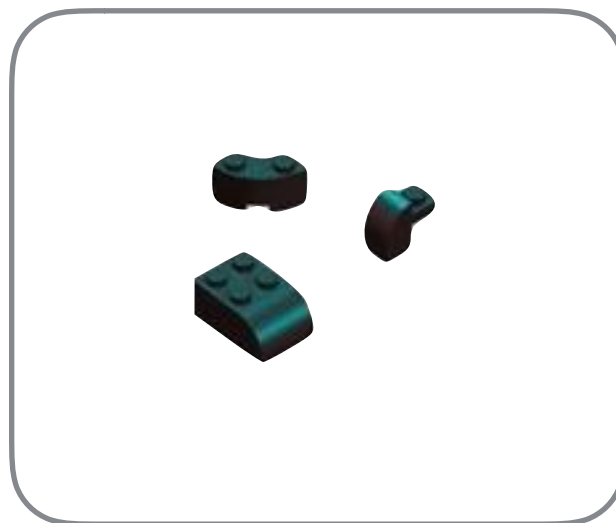
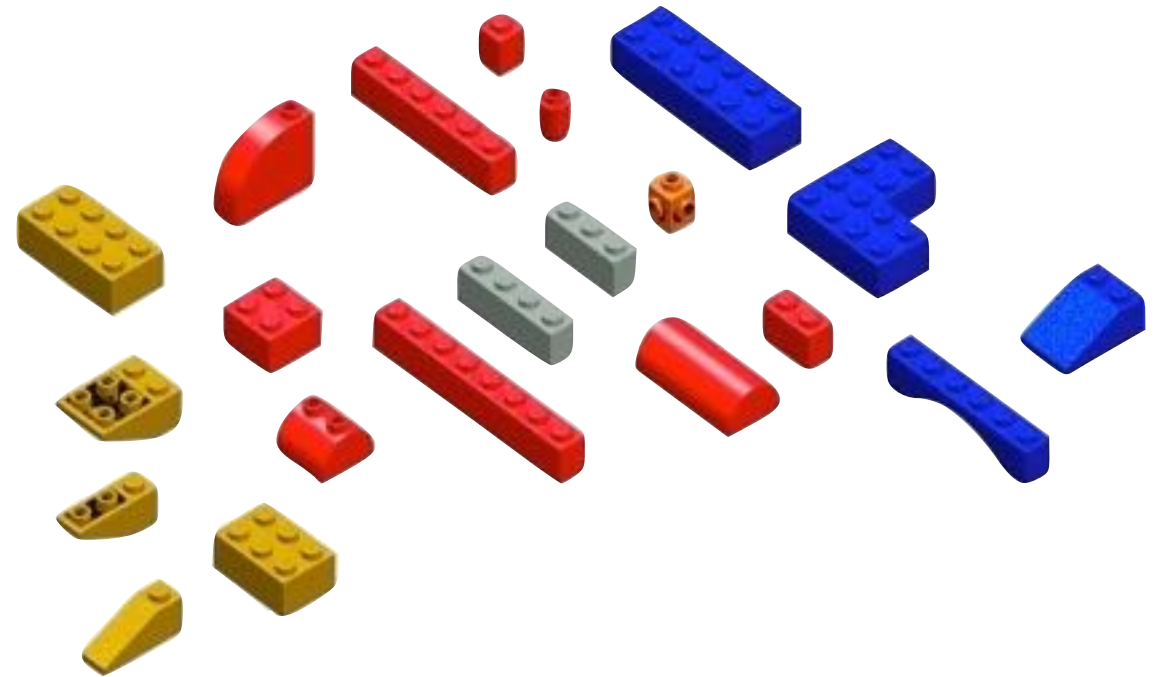


IMP is divided into *modules*

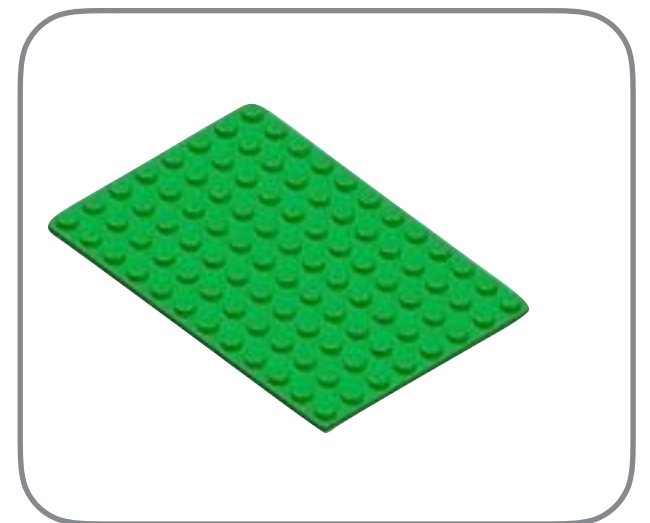


IMP kernel

IMP is divided into *modules*

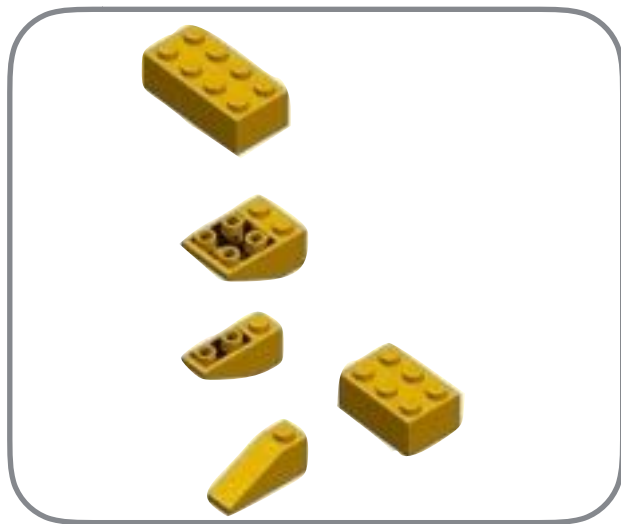
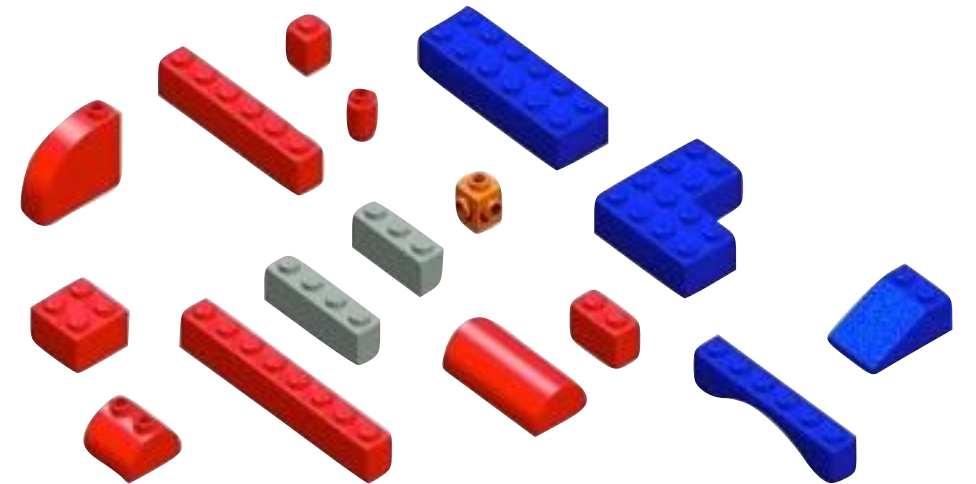


IMP.algebra

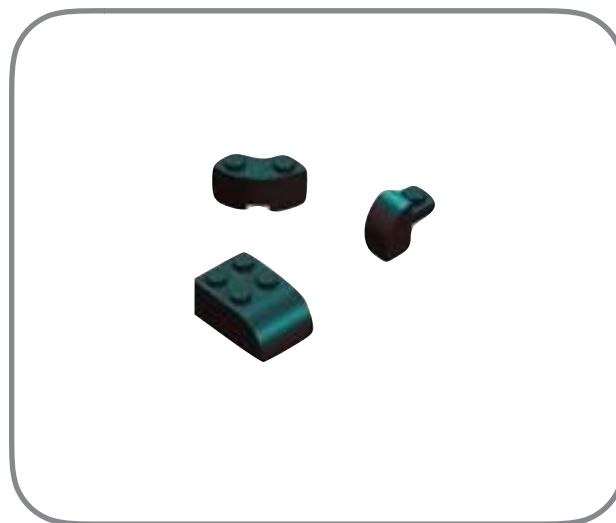


IMP kernel

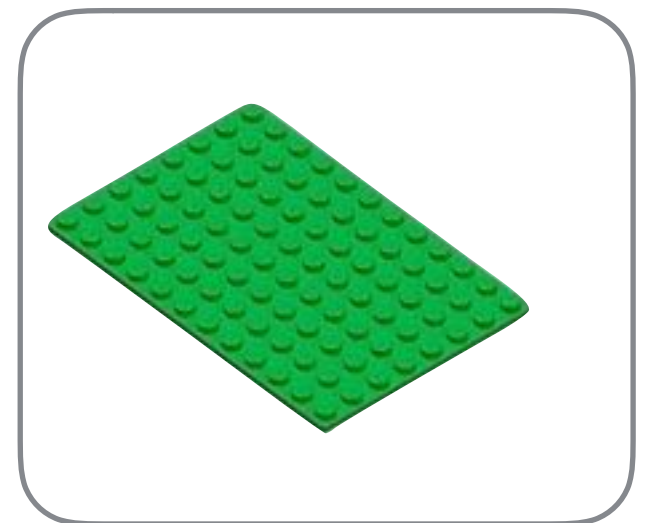
IMP is divided into *modules*



IMP.saxs

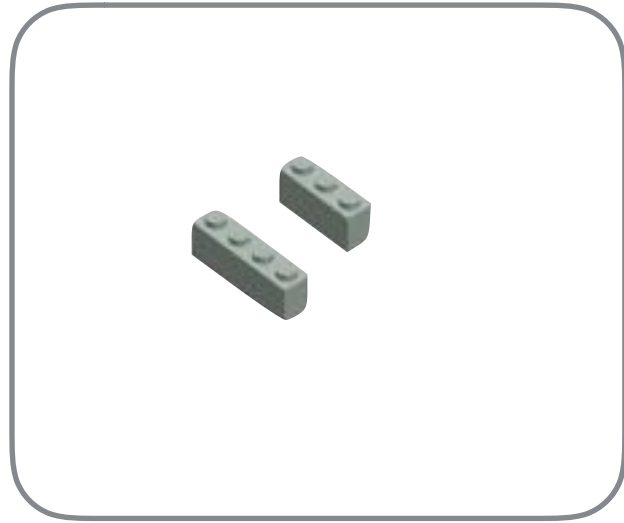


IMP.algebra

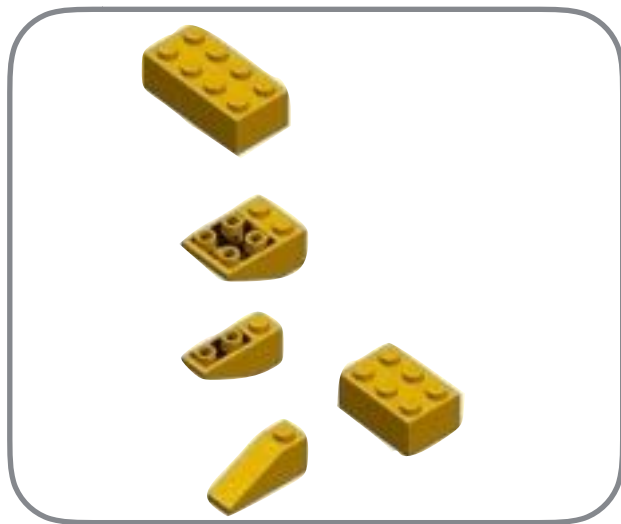
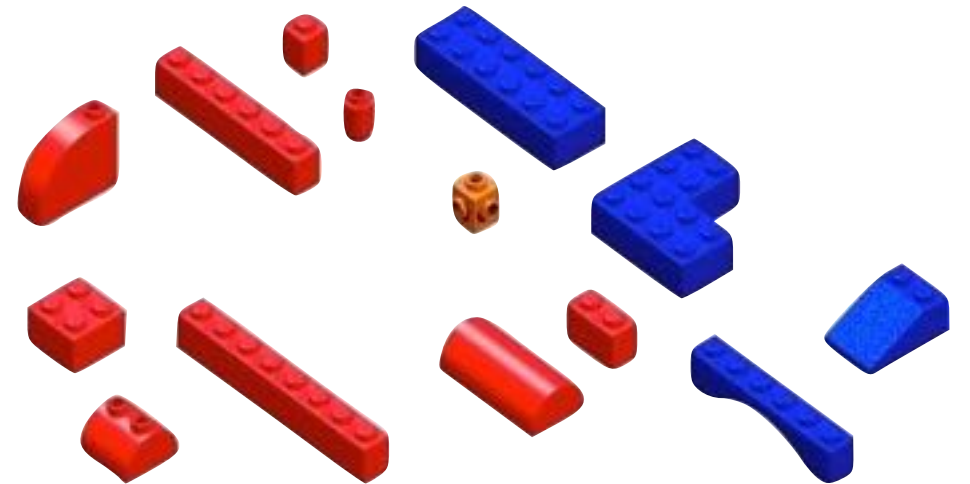


IMP kernel

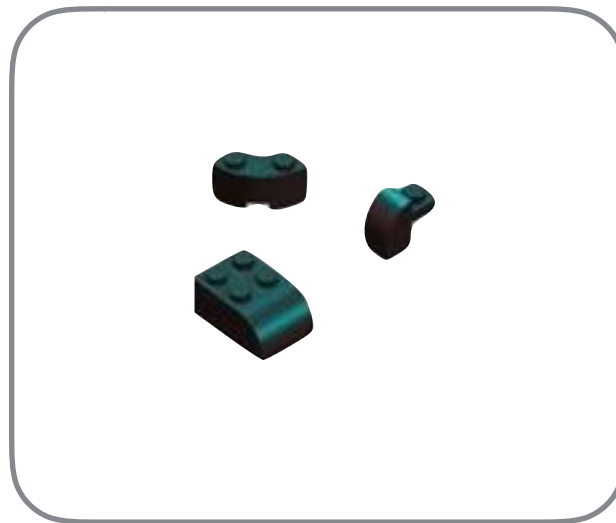
IMP is divided into *modules*



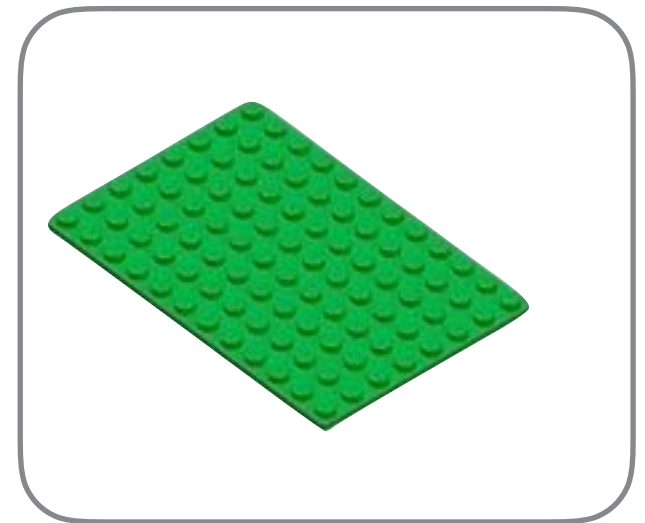
IMP.em



IMP.saxs

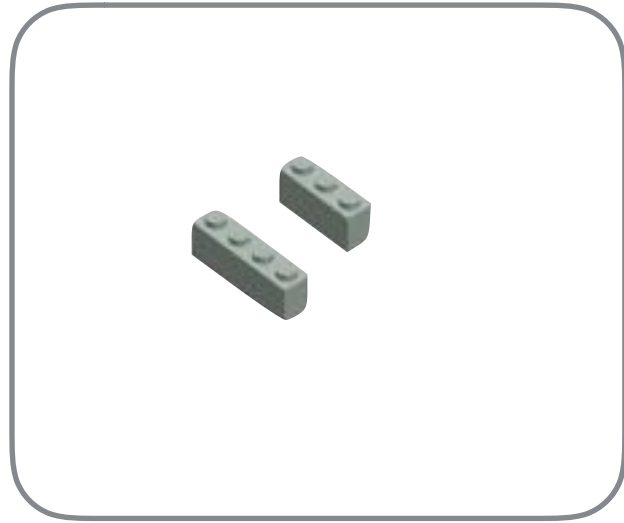


IMP.algebra



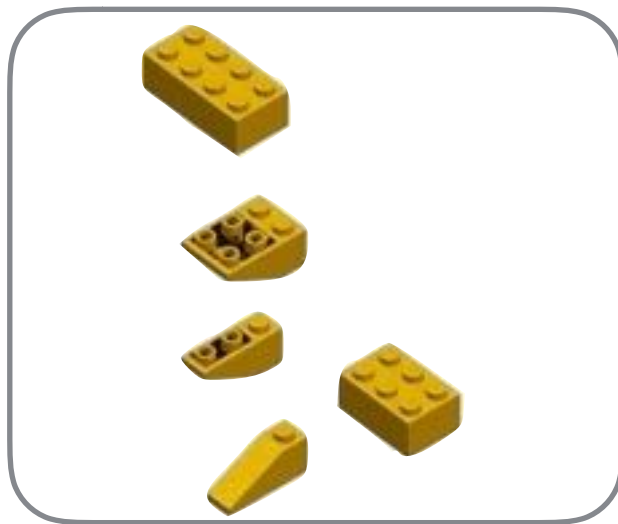
IMP kernel

IMP is divided into *modules*

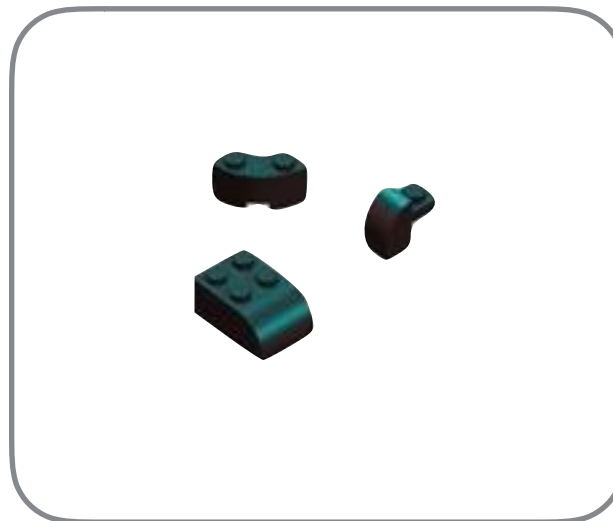


IMP.em

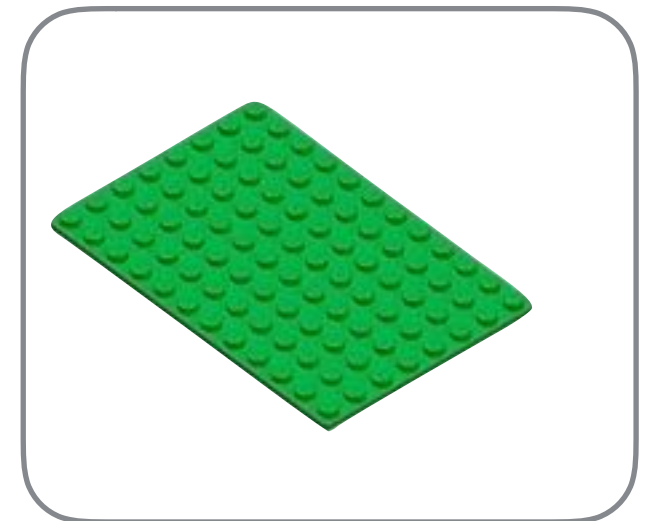
- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces



IMP.saxs

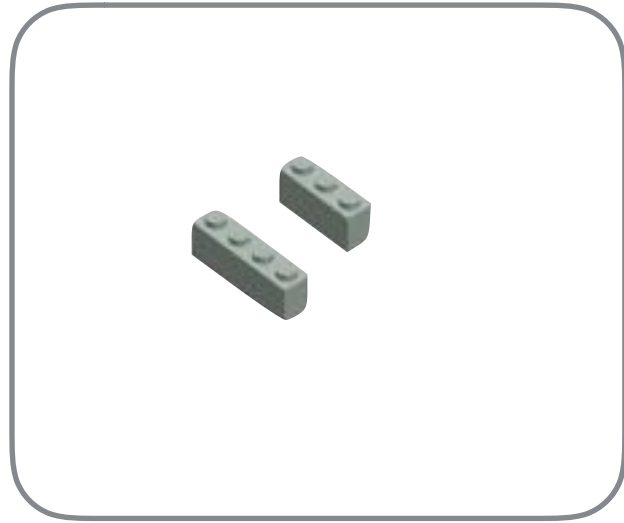


IMP.algebra



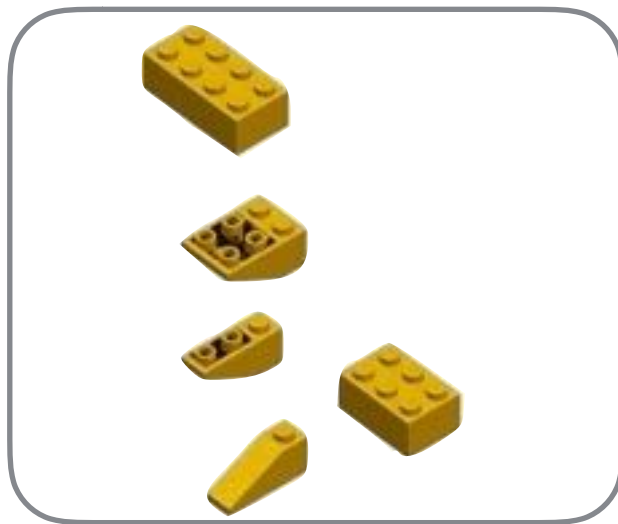
IMP kernel

IMP is divided into *modules*

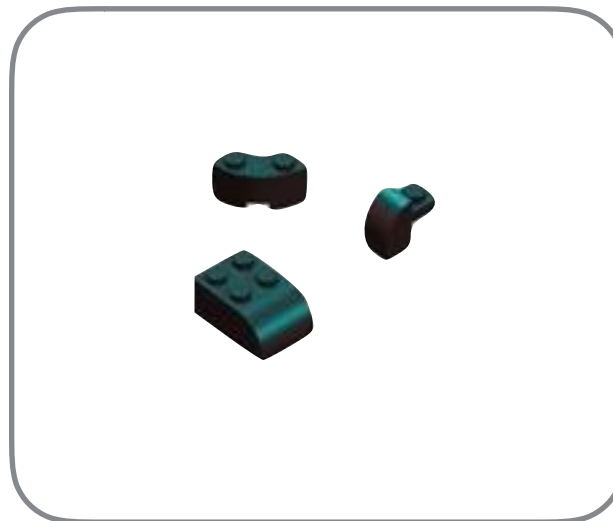


IMP.em

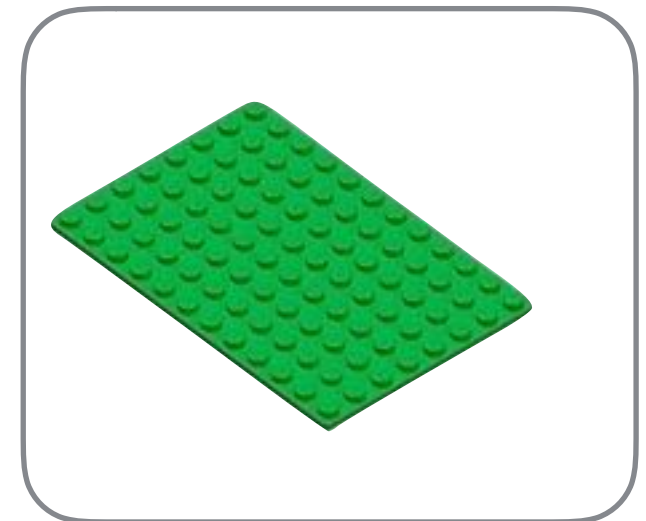
- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces



IMP.saxs



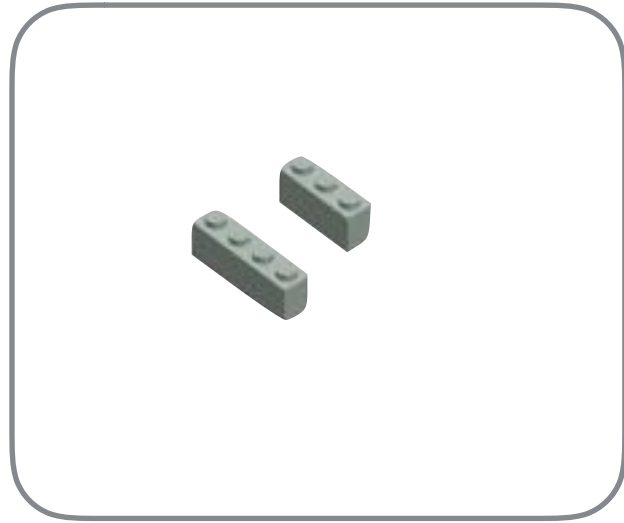
IMP.algebra



IMP kernel

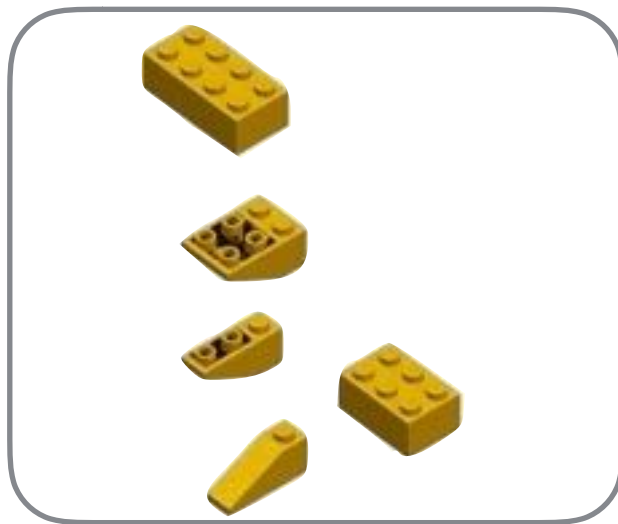
Common functionality

IMP is divided into *modules*

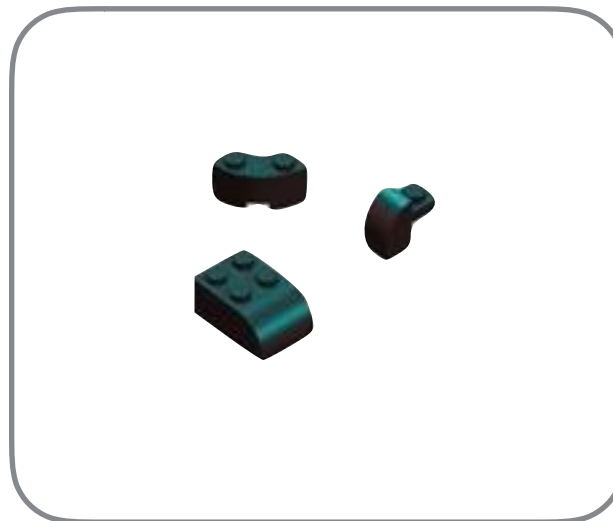


IMP.em

- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces

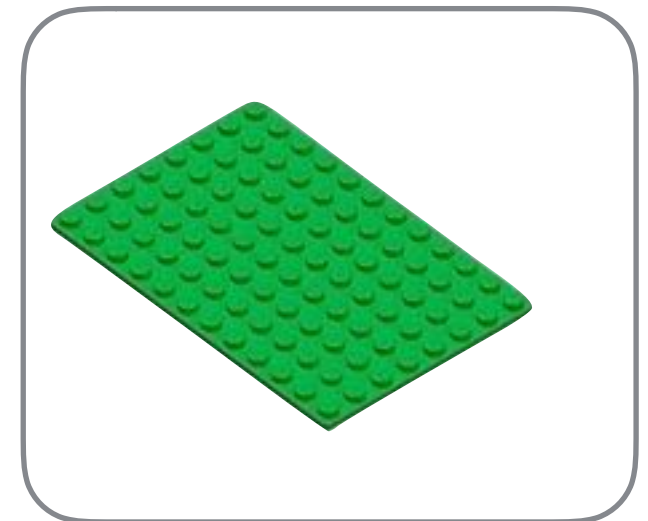


IMP.saxs



IMP.algebra

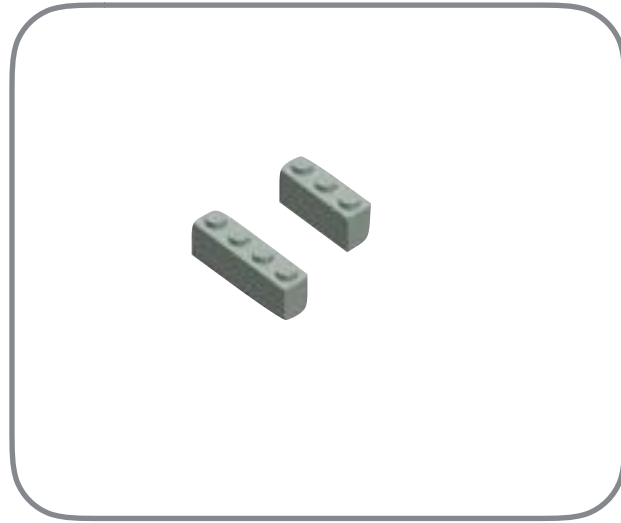
Geometry, primitive shapes



IMP kernel

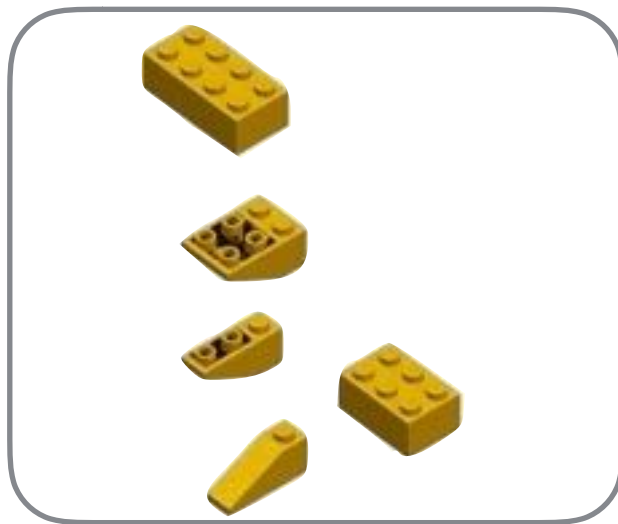
Common functionality

IMP is divided into *modules*



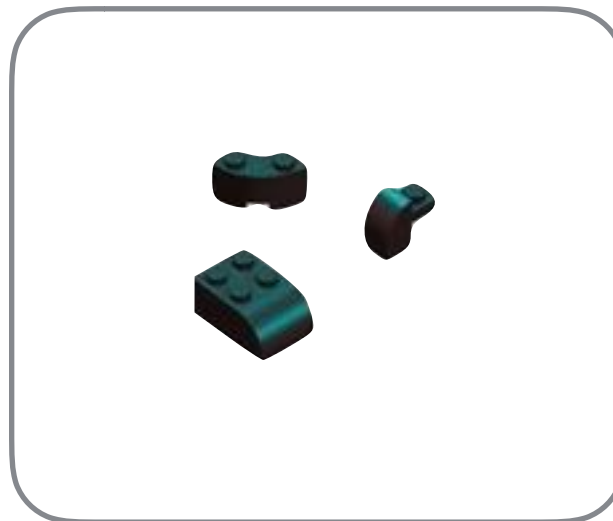
IMP.em

- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces



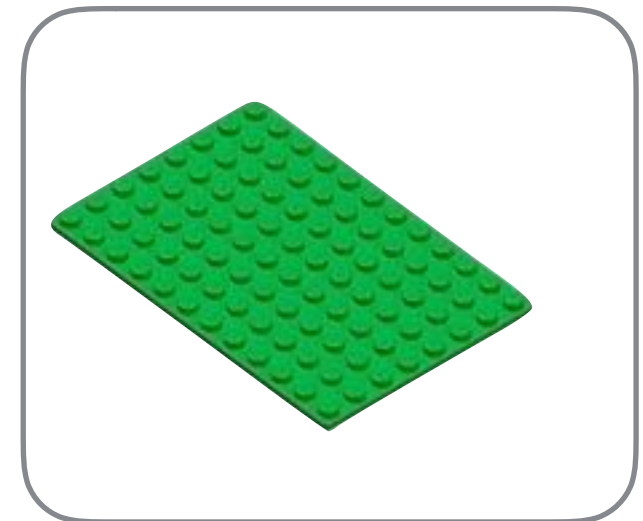
IMP.saxs

*Handling of Small Angle
X-ray (SAXS) data*



IMP.algebra

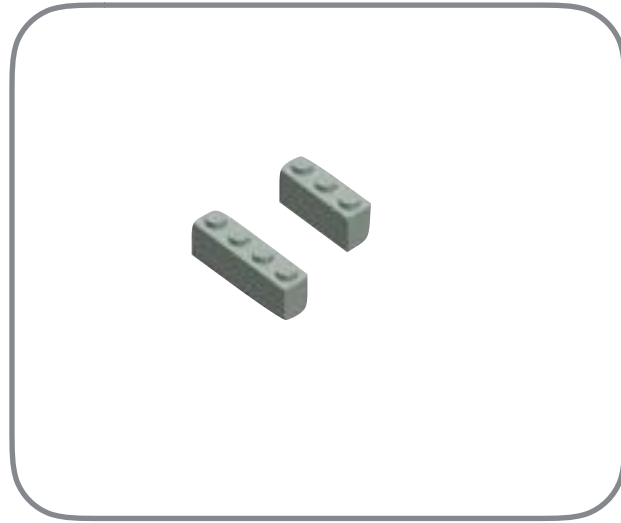
Geometry, primitive shapes



IMP kernel

Common functionality

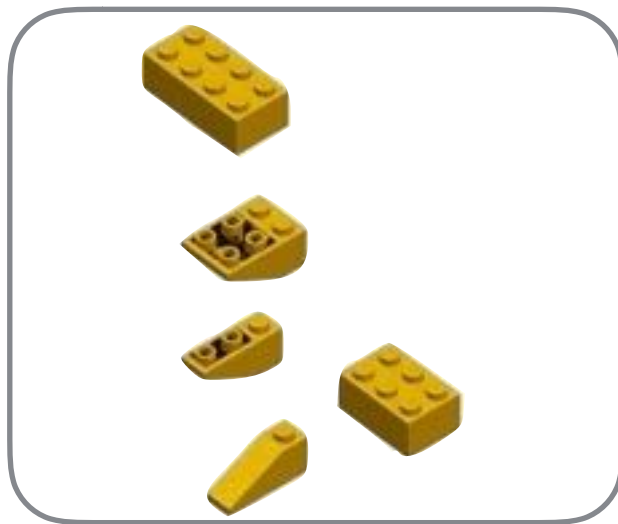
IMP is divided into *modules*



IMP.em

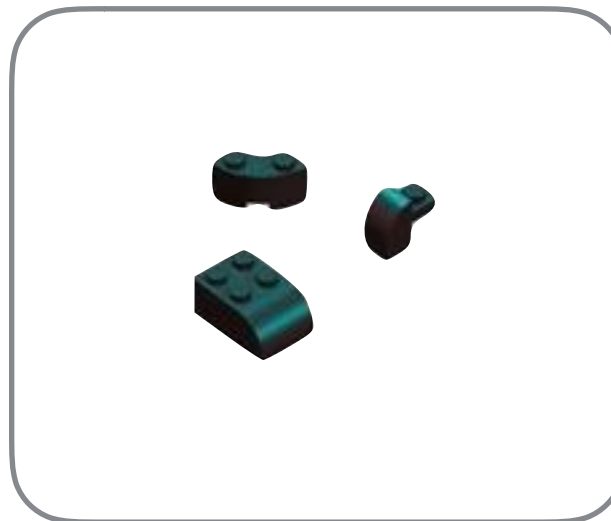
*Handling of electron microscopy
(EM) experimental data*

- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces



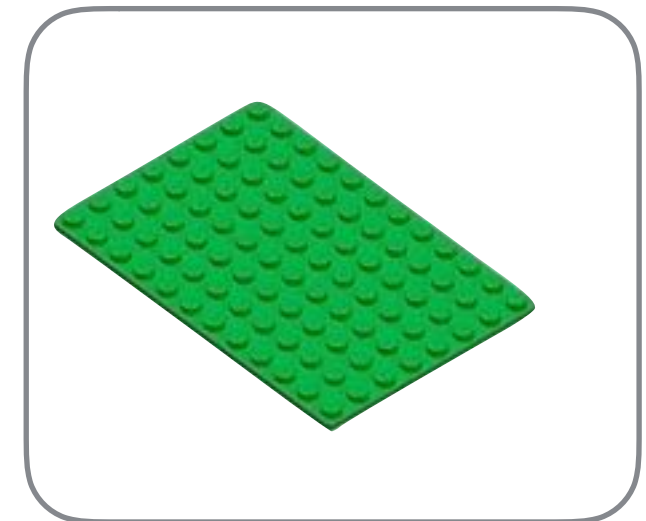
IMP.saxs

*Handling of Small Angle
X-ray (SAXS) data*



IMP.algebra

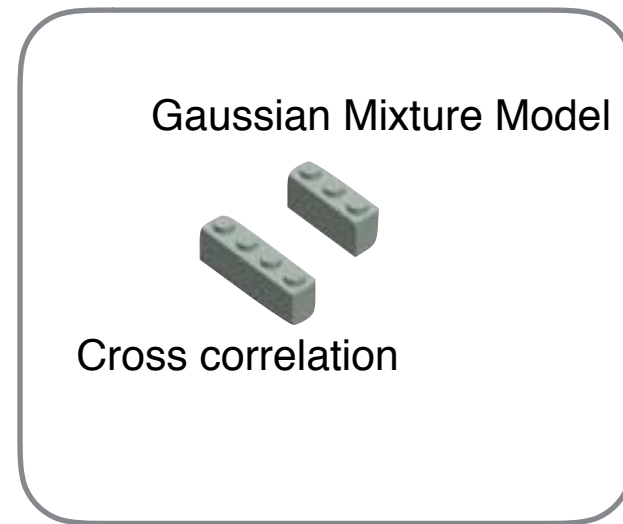
Geometry, primitive shapes



IMP kernel

Common functionality

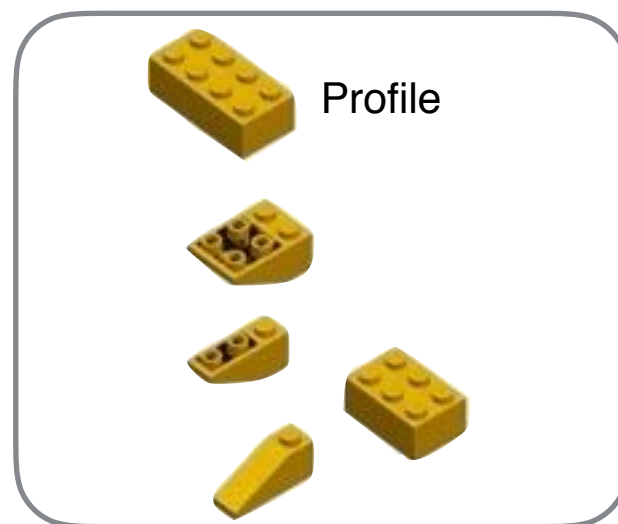
IMP is divided into *modules*



IMP.em

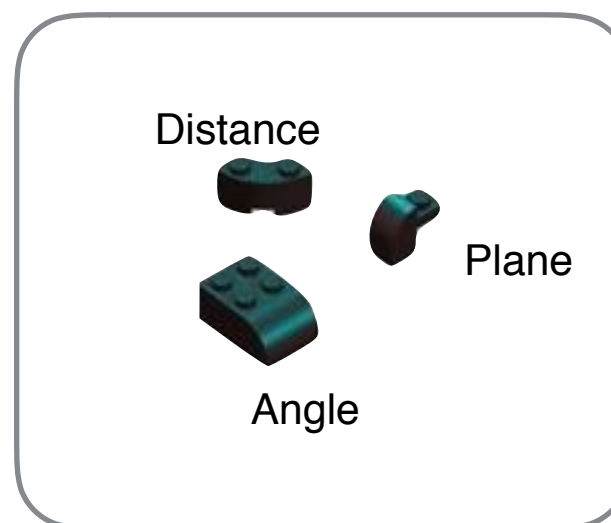
*Handling of electron microscopy
(EM) experimental data*

- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces



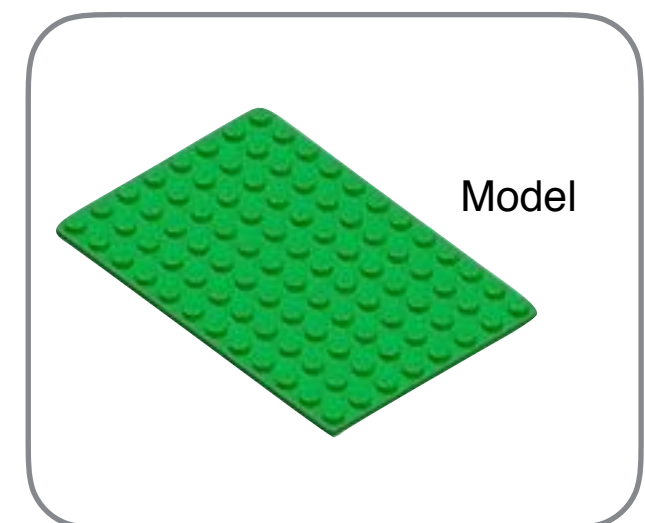
IMP.saxs

*Handling of Small Angle
X-ray (SAXS) data*



IMP.algebra

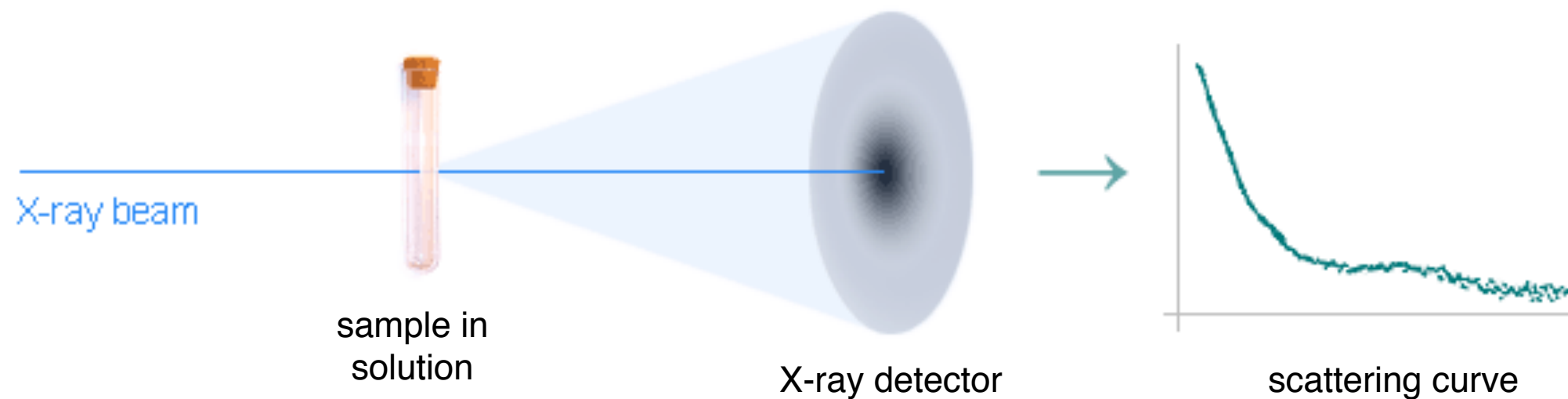
Geometry, primitive shapes



IMP kernel

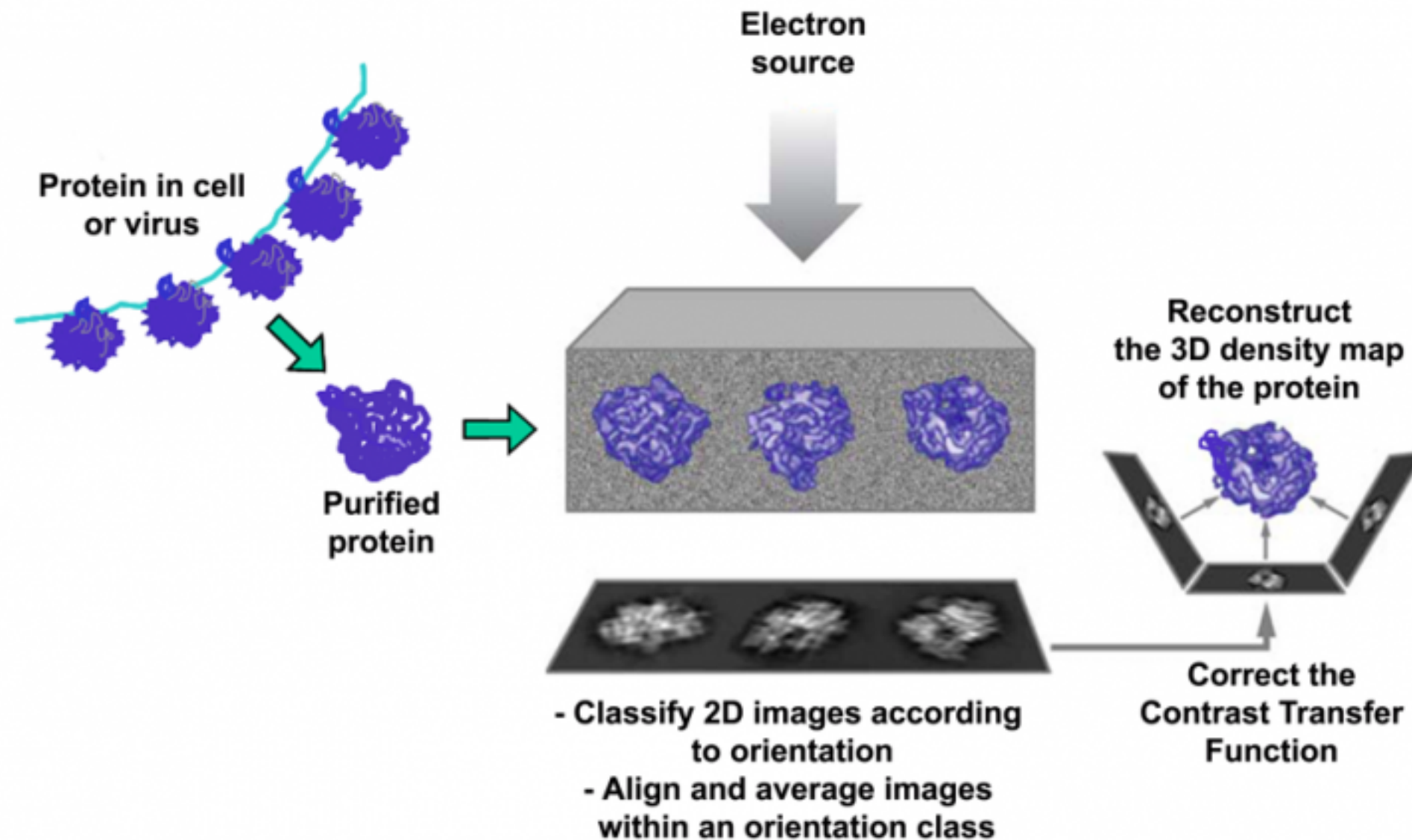
Common functionality

Small-angle X-ray scattering (SAXS)



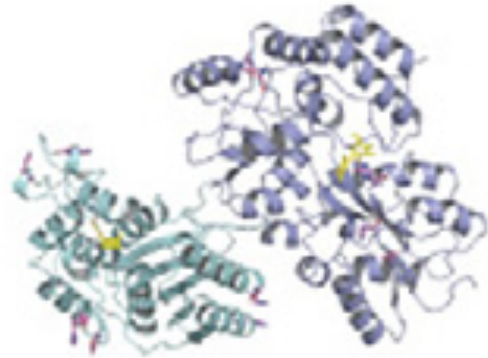
- Sample is in solution
 - Pro: easier to produce, closer to its *in vivo* state (compared to making crystals for X-ray crystallography)
 - Con: rotationally averaged

Electron microscopy (EM)



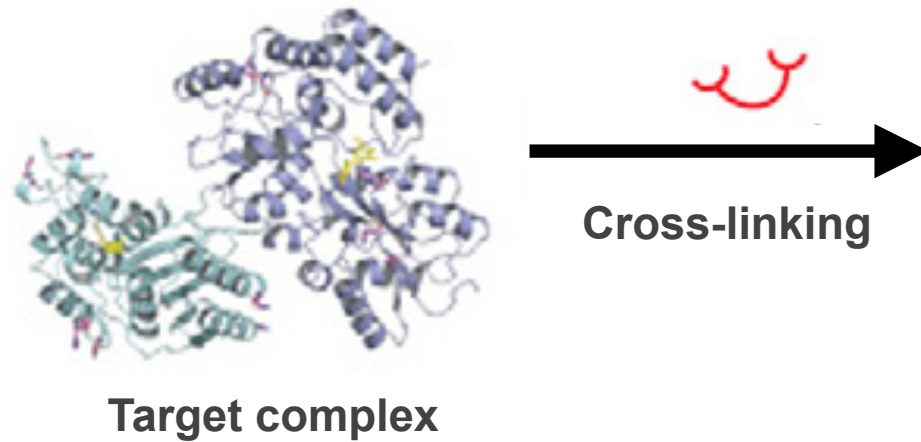
- Significant processing required to generate a 3D map

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

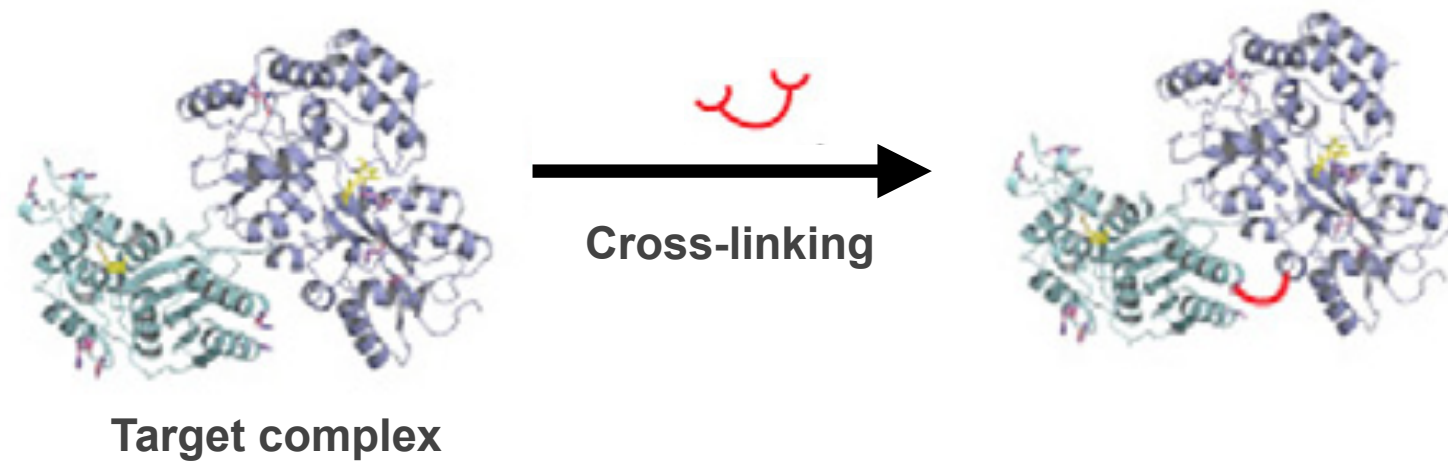


Target complex

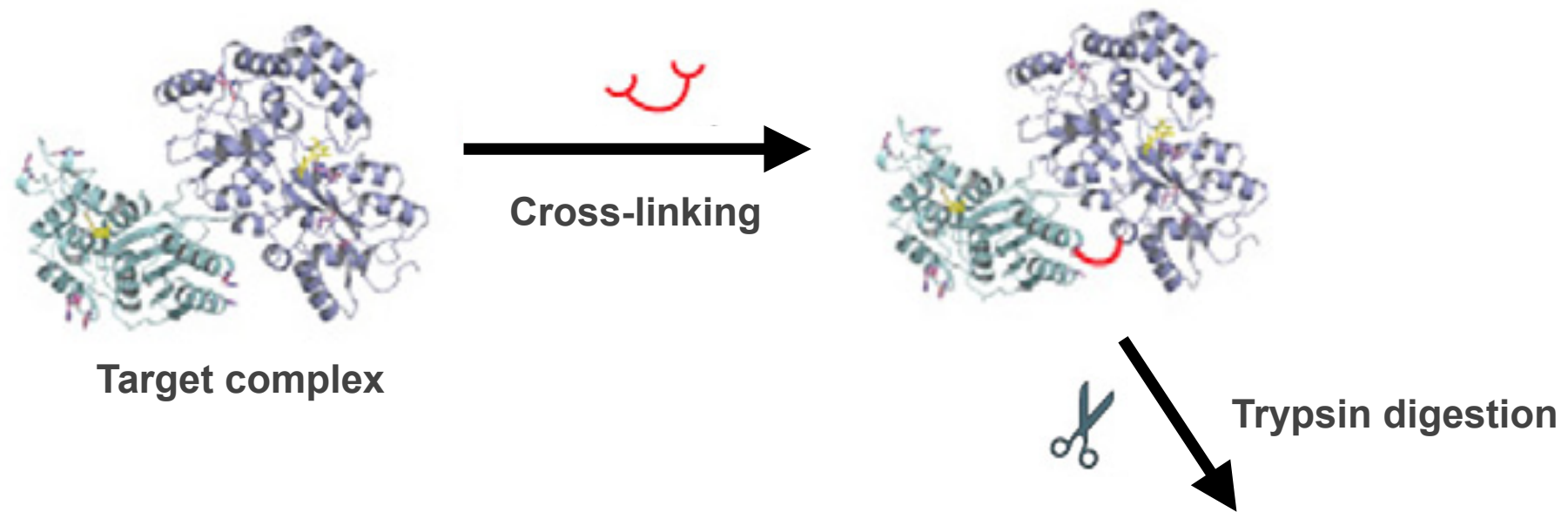
Cross-linking coupled with mass spectrometry (CX-MS)



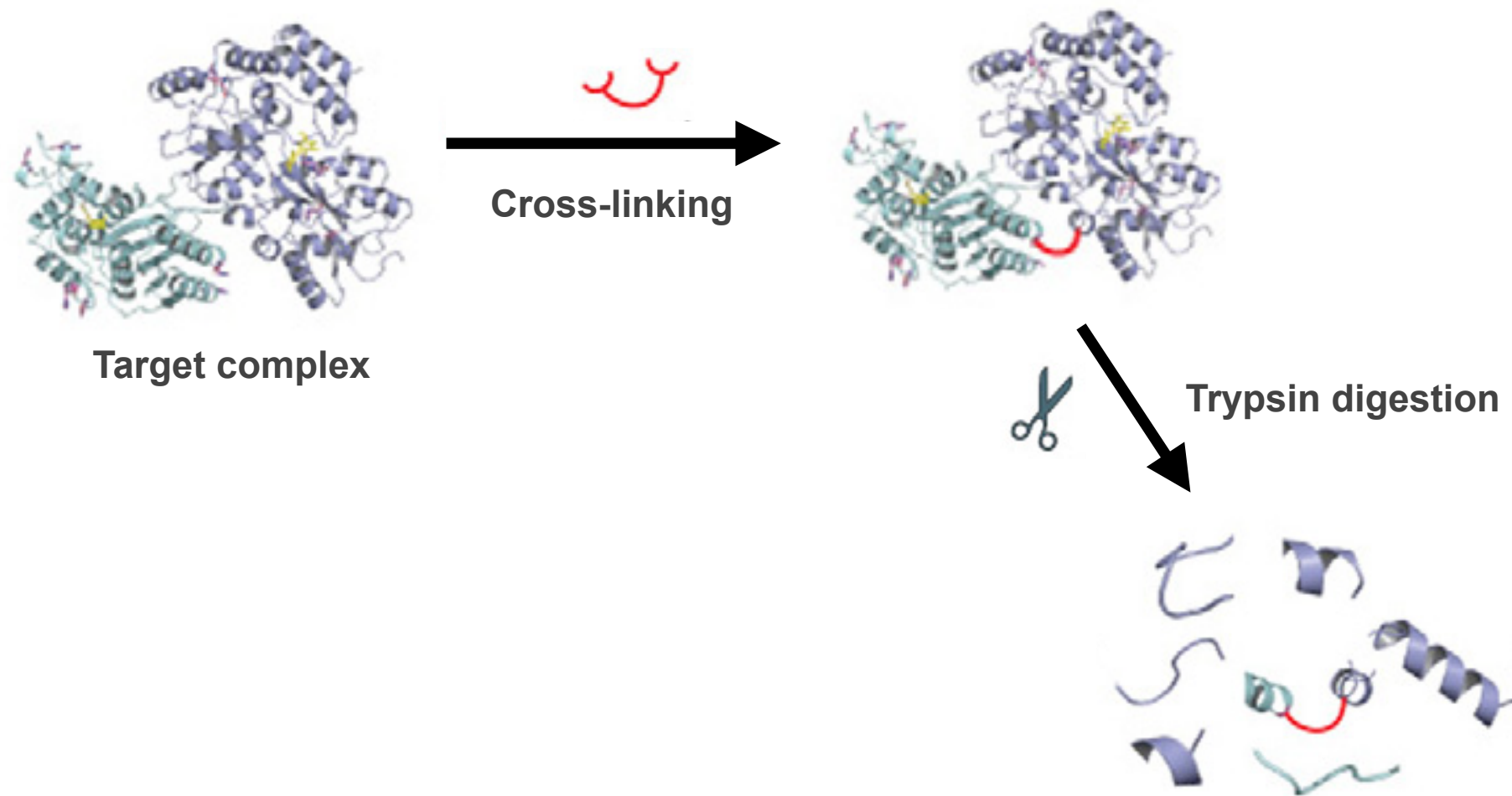
Cross-linking coupled with mass spectrometry (CX-MS)



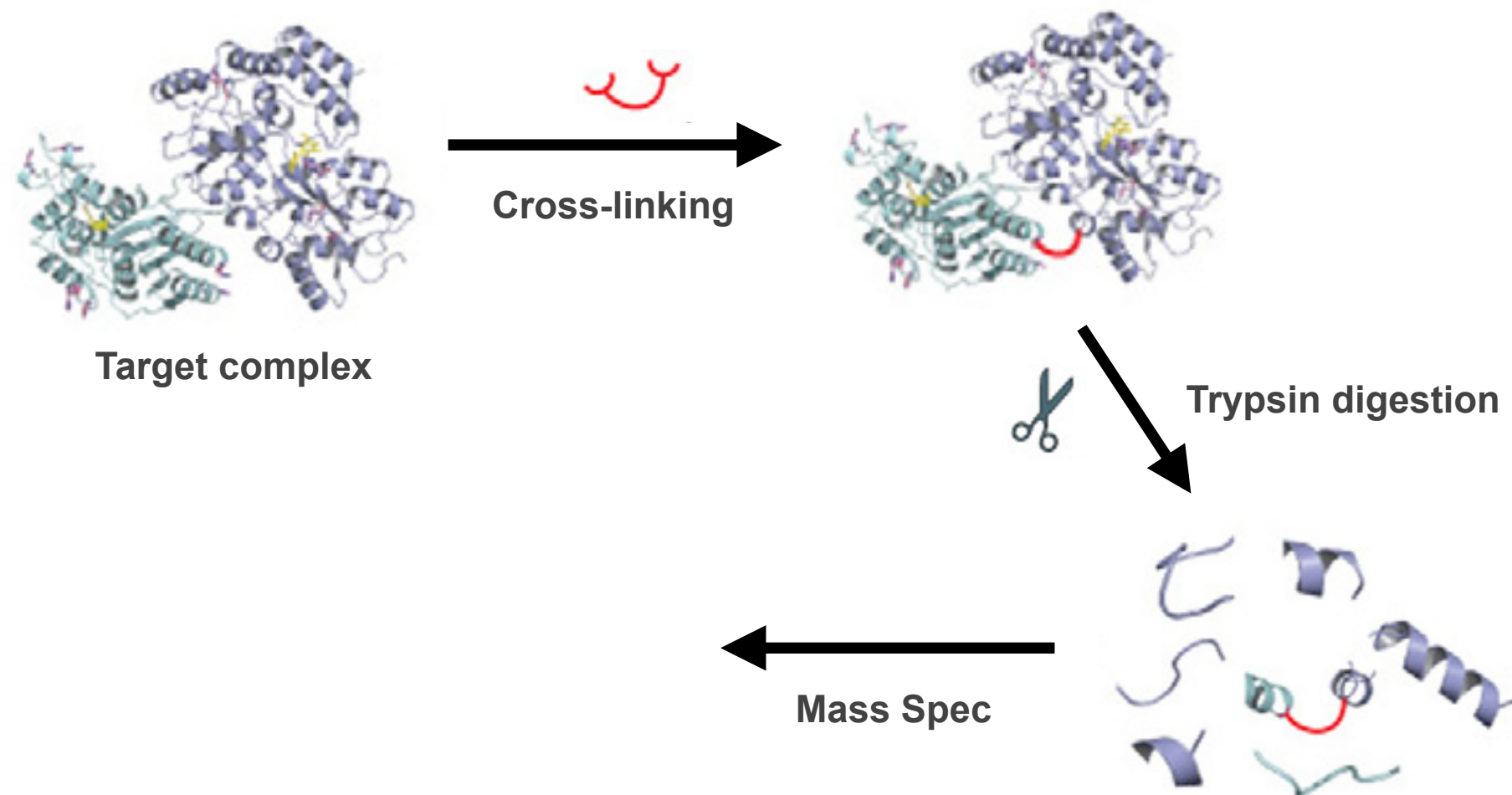
Cross-linking coupled with mass spectrometry (CX-MS)



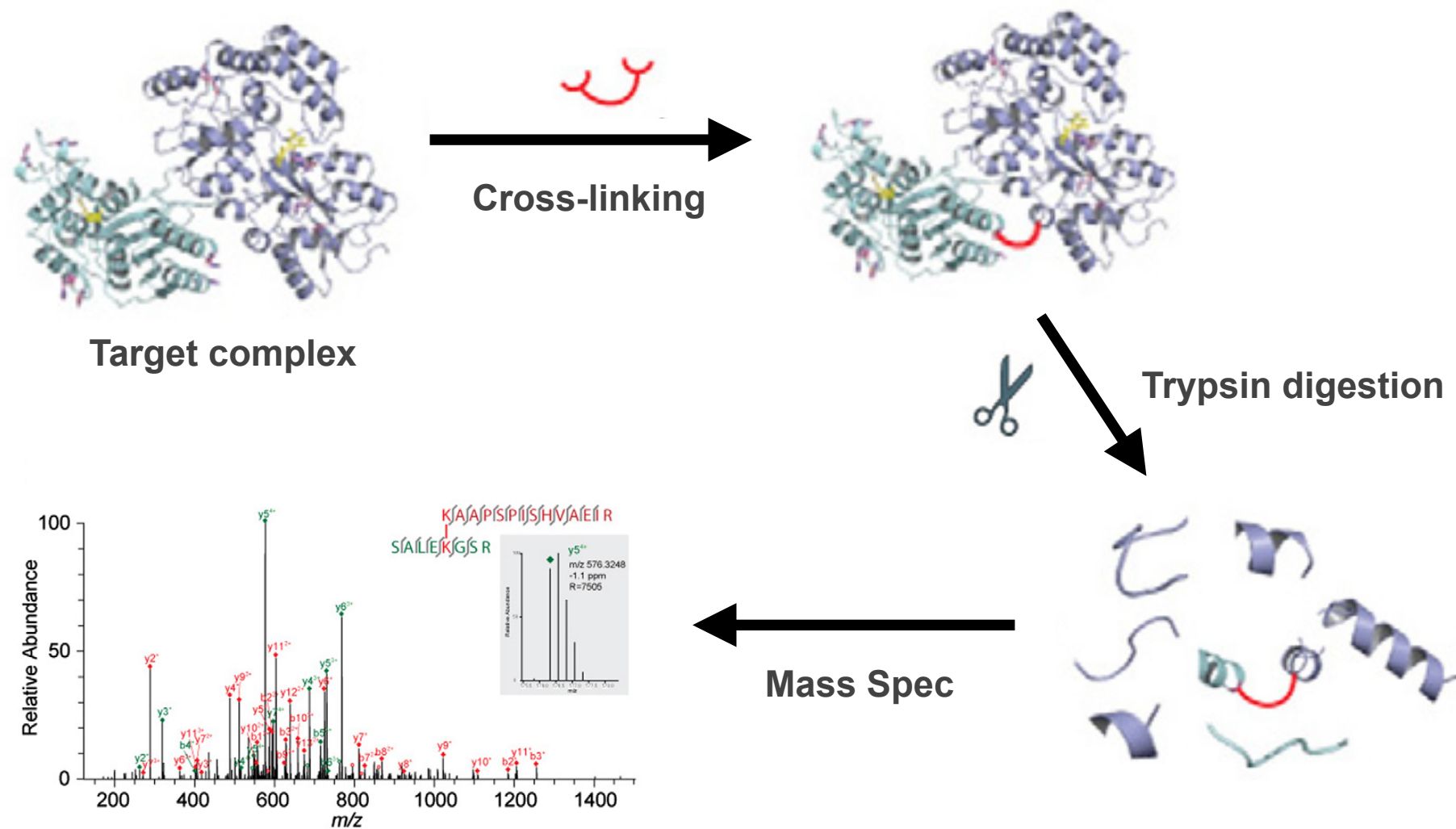
Cross-linking coupled with mass spectrometry (CX-MS)



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

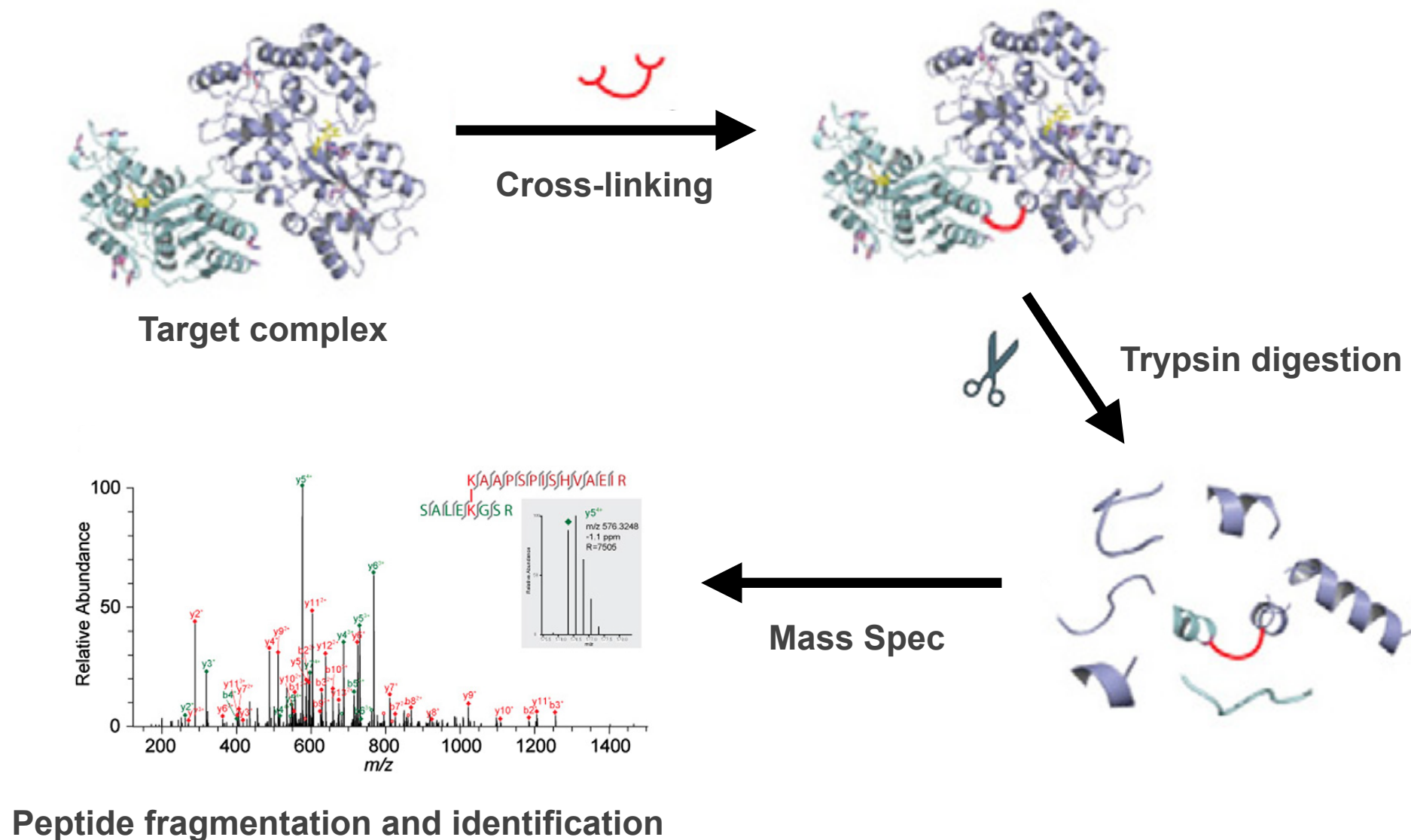


Cross-linking coupled with mass spectrometry (CX-MS)



Peptide fragmentation and identification

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



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

IMP software implementation

IMP software implementation

- Each 'piece' is a Python class

IMP software implementation

- Each 'piece' is a Python class

Distance



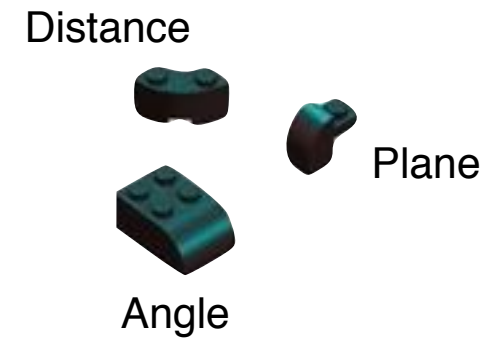
Plane



Angle

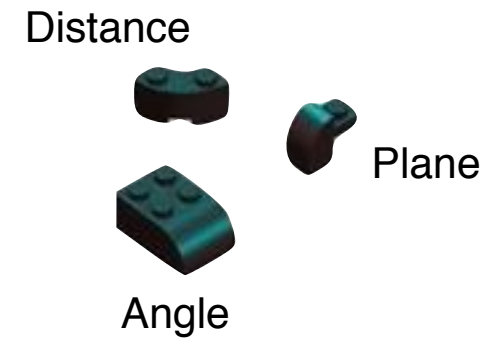
IMP software implementation

- Each 'piece' is a Python class
- Most classes actually 'wrap' an underlying class in C++



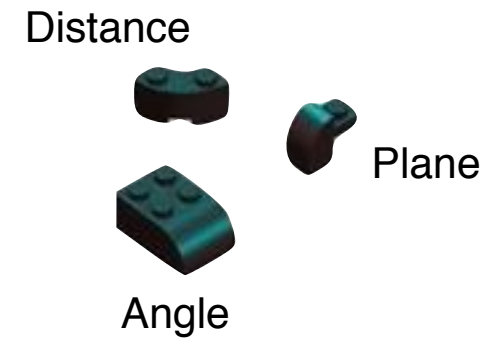
IMP software implementation

- Each 'piece' is a Python class
- Most classes actually 'wrap' an underlying class in C++
 - C++ for speed; Python for flexibility, interfacing



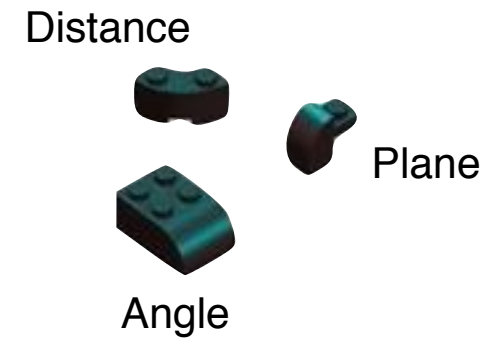
IMP software implementation

- Each 'piece' is a Python class
- Most classes actually 'wrap' an underlying class in C++
 - C++ for speed; Python for flexibility, interfacing
- Each module is a Python module, and C++ namespace



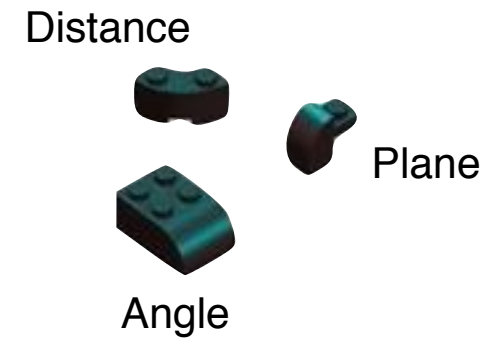
IMP software implementation

- Each 'piece' is a Python class
- Most classes actually 'wrap' an underlying class in C++
 - C++ for speed; Python for flexibility, interfacing
- Each module is a Python module, and C++ namespace
- IMP is usually used from Python, by writing a script (but certainly can use from C++)



IMP software implementation

- Each 'piece' is a Python class
- Most classes actually 'wrap' an underlying class in C++
 - C++ for speed; Python for flexibility, interfacing
- Each module is a Python module, and C++ namespace
- IMP is usually used from Python, by writing a script (but certainly can use from C++)
- A protocol is thus one or more Python scripts plus the input data



Link via Python to other packages

- Connect IMP components to other packages via standard Python interfaces
- Avoid code duplication



Link via Python to other packages

- Connect IMP components to other packages via standard Python interfaces
- Avoid code duplication



MODELLER

comparative modeling



Link via Python to other packages

- Connect IMP components to other packages via standard Python interfaces
- Avoid code duplication



MODELLER

comparative modeling



BioPython

*handling of
sequence data*



Link via Python to other packages

- Connect IMP components to other packages via standard Python interfaces
- Avoid code duplication



MODELLER

comparative modeling



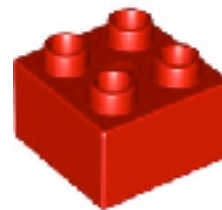
BioPython

*handling of
sequence data*



Chimera/VMD

visualization



Link via Python to other packages

- Connect IMP components to other packages via standard Python interfaces
- Avoid code duplication



MODELLER

comparative modeling



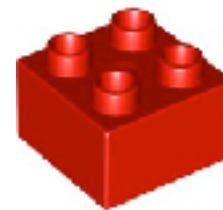
BioPython

*handling of
sequence data*



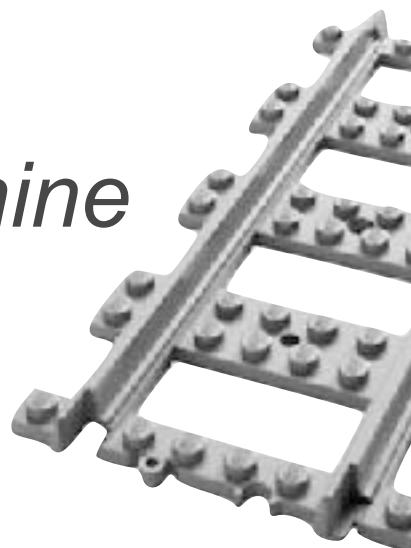
Chimera/VMD

visualization



scikit-learn

*clustering, machine
learning*



Link via Python to other packages

- Connect IMP components to other packages via standard Python interfaces
- Avoid code duplication



BioPython

*handling of
sequence data*



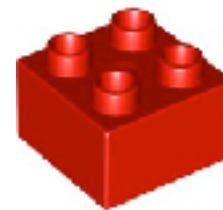
MODELLER

comparative modeling



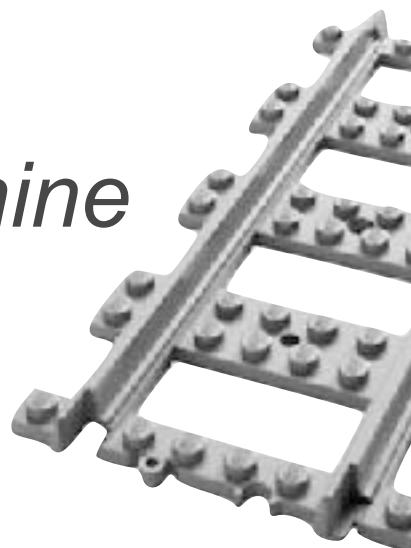
Chimera/VMD

visualization



scikit-learn

*clustering, machine
learning*



numpy/scipy

matrix/linear algebra



Link via Python to other packages

- Connect IMP components to other packages via standard Python interfaces
- Avoid code duplication



MODELLER

comparative modeling



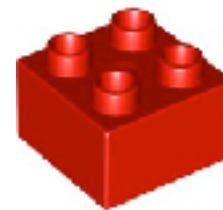
BioPython

*handling of
sequence data*



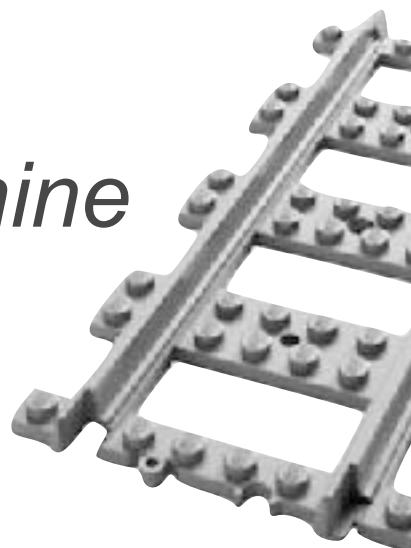
Chimera/VMD

visualization



scikit-learn

*clustering, machine
learning*



numpy/scipy

matrix/linear algebra



etc.

Documentation

- Can be found at <https://integrativemodeling.org/doc.html>
- Split into a manual (designed to be read sequentially, contains tutorials) and a reference guide (random access, documenting the IMP classes and modules)

Example Python script

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

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

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


Imports

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


Imports

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

- Make IMP classes in the IMP kernel (**IMP**) and **IMP.algebra** and **IMP.core** modules available

Imports

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

- Make IMP classes in the IMP kernel (**IMP**) and **IMP.algebra** and **IMP.core** modules available
- See the IMP Reference Guide ‘Modules’ tab for a comprehensive list of all modules:
<https://integrativemodeling.org/2.7.0/doc/ref/namespaces.html>

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```


Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- Create a new **Model** object (an *instance* of the *Model class*) and assign it to the variable 'm'

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- 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 (see the IMP Reference Guide)

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- 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 (see the **IMP Reference Guide**)
- Create two particles called 'p1' and 'p2'; each is an abstract data container inside the model (really **p1** and **p2** are just indices into a data structure inside **Model**) and can hold any number of attribute:value pairs, e.g.

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- 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 (see the **IMP Reference Guide**)
- Create two particles called 'p1' and 'p2'; each is an abstract data container inside the model (really **p1** and **p2** are just indices into a data structure inside **Model**) and can hold any number of attribute:value pairs, e.g.
 - xyz coordinates

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- 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 (see the **IMP Reference Guide**)
- Create two particles called 'p1' and 'p2'; each is an abstract data container inside the model (really **p1** and **p2** are just indices into a data structure inside **Model**) and can hold any number of attribute:value pairs, e.g.
 - xyz coordinates
 - mass

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- 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 (see the **IMP Reference Guide**)
- Create two particles called 'p1' and 'p2'; each is an abstract data container inside the model (really **p1** and **p2** are just indices into a data structure inside **Model**) and can hold any number of attribute:value pairs, e.g.
 - xyz coordinates
 - mass
 - radius

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- 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 (see the **IMP Reference Guide**)
- Create two particles called 'p1' and 'p2'; each is an abstract data container inside the model (really **p1** and **p2** are just indices into a data structure inside **Model**) 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)

Model and particles

```
m = IMP.Model()  
# Create two "untyped" particles  
p1 = m.add_particle('p1')  
p2 = m.add_particle('p2')
```

- 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 (see the **IMP Reference Guide**)
- Create two particles called 'p1' and 'p2'; each is an abstract data container inside the model (really **p1** and **p2** are just indices into a data structure inside **Model**) 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.

Decorators

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


Decorators

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

Decorators

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

Decorators

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

Decorators

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

Decorators

```
# "Decorate" the Particles with x,y,z attributes
# (point-like particles)
d1 = IMP.core.XYZ.setup_particle(m, p1)
d2 = IMP.core.XYZ.setup_particle(m, 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
- A single particle can be decorated multiple times (e.g. can be a 3D point and also have mass, be part of a bond, and have a parent, such as a residue)

Single-particle restraints

```
# 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(m, s, p1)
```


Single-particle restraints

```
# 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(m, s, p1)
```

- A **Restraint** is a term in our scoring function, just a function of one or more particles

Single-particle restraints

```
# 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(m, s, p1)
```

- A **Restraint** is a term in our scoring function, just a function of one or more particles
- **IMP.core.SingletonRestraint** applies a **SingletonScore** to a single particle (**p1** in this case)

Single-particle restraints

```
# 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(m, s, p1)
```

- A **Restraint** is a term in our scoring function, just a function of one or more particles
- **IMP.core.SingletonRestraint** applies a **SingletonScore** to a single particle (**p1** in this case)
- In turn, **DistanceToSingletonScore** calculates the Cartesian distance between a fixed point and **p1**, then uses a **UnaryFunction** to weight that distance

Single-particle restraints

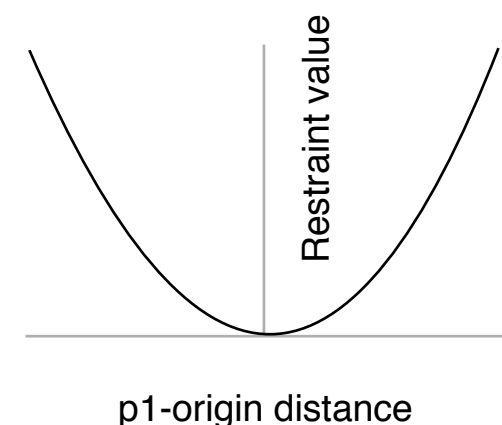
```
# Harmonically restrain p1 to be zero distance  
# from the origin  
f = IMP.core.Harmonic(0.0, 1.0)  
s = IMP.core.DistanceToSingletonScore(f,  
                                       IMP.algebra.Vector3D(0., 0., 0.))  
r1 = IMP.core.SingletonRestraint(m, s, p1)
```

- A **Restraint** is a term in our scoring function, just a function of one or more particles
- **IMP.core.SingletonRestraint** applies a **SingletonScore** to a single particle (**p1** in this case)
- In turn, **DistanceToSingletonScore** calculates the Cartesian distance between a fixed point and **p1**, then uses a **UnaryFunction** to weight that distance
- **Harmonic** is a unary function that applies a simple harmonic spring

Single-particle restraints

```
# 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(m, s, p1)
```

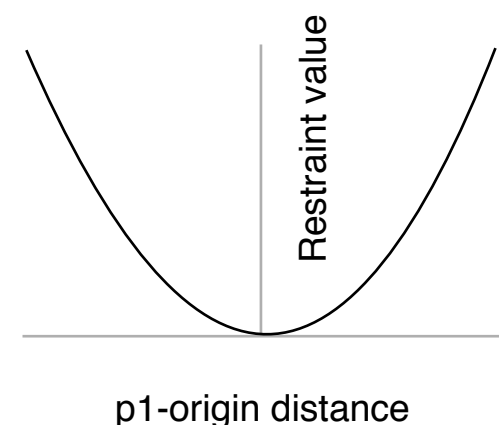
- A **Restraint** is a term in our scoring function, just a function of one or more particles
- **IMP.core.SingletonRestraint** applies a **SingletonScore** to a single particle (**p1** in this case)
- In turn, **DistanceToSingletonScore** calculates the Cartesian distance between a fixed point and **p1**, then uses a **UnaryFunction** to weight that distance
- **Harmonic** is a unary function that applies a simple harmonic spring



Single-particle restraints

```
# 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(m, s, p1)
```

- A **Restraint** is a term in our scoring function, just a function of one or more particles
- **IMP.core.SingletonRestraint** applies a **SingletonScore** to a single particle (**p1** in this case)
- In turn, **DistanceToSingletonScore** calculates the Cartesian distance between a fixed point and **p1**, then uses a **UnaryFunction** 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



Two-particle restraints

```
# 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(m, 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
- Note that the `core` module provides simple 'building block' restraints
- More complex restraints to handle specific types of input data are found in other modules (e.g. the `em` and `saxs` modules provide restraints to handle EM and SAXS data respectively)

Other restraints

```
# 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(m, s, (p1, p2))
```

- Other restraints can be set up by combining building blocks:

Other restraints

```
# 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(m, s, (p1, p2))
```

- Other restraints can be set up by combining building blocks:

Force field (bond terms)

- Given two **XYZ** and **Bonded** particles **p1** and **p2**,
- Look up the **Bond** particle that relates them
- Extract mean and stiffness parameters
- Enforce a simple harmonic between **p1** and **p2**

Other restraints

```
# 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(m, s, (p1, p2))
```

- Other restraints can be set up by combining building blocks:

Force field (bond terms)

- Given two **XYZ** and **Bonded** particles **p1** and **p2**,
- Look up the **Bond** particle that relates them
- Extract mean and stiffness parameters
- Enforce a simple harmonic between **p1** and **p2**

Statistical potential

- Given two **XYZ** and **Atom** particles **p1** and **p2**,
- Look up the atom type of each particle (e.g. CA, CB)
- Look up histogram as a function of the two types
- Enforce a cubic spline between **p1** and **p2** (-log of the histogram)

Sampling

```
# 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

Overall workflow

Overall workflow

IMP.Model, m



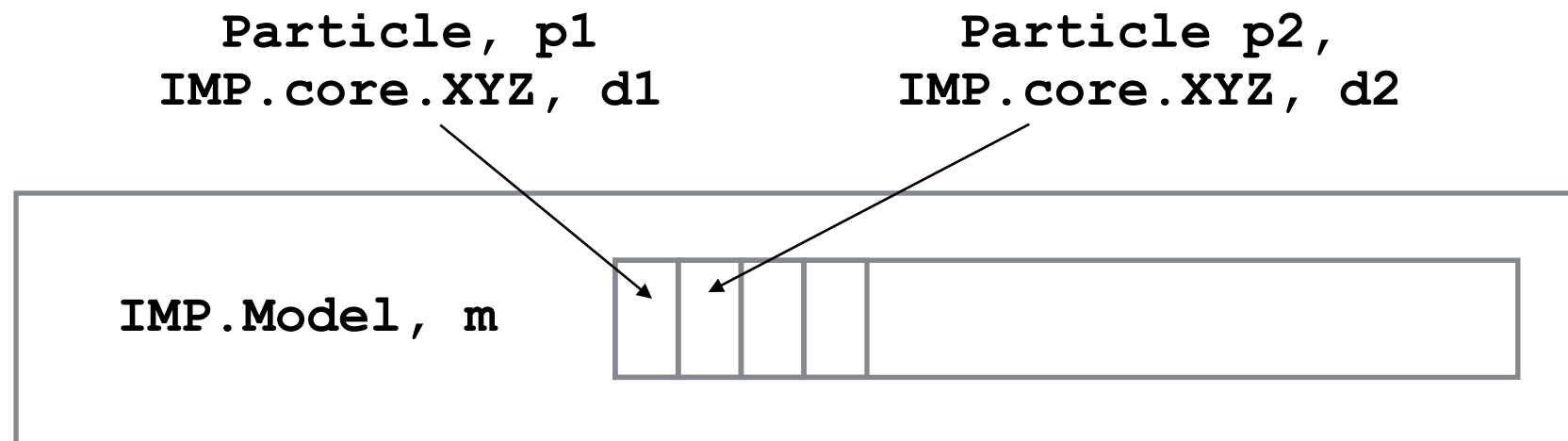
Overall workflow

Particle, p1
IMP.core.XYZ, d1

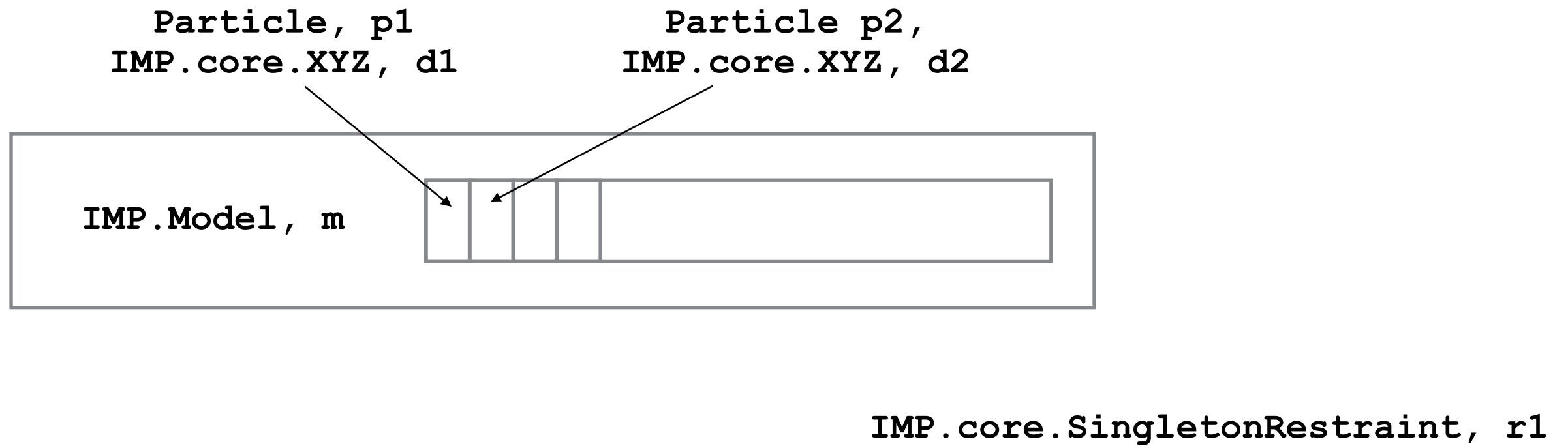
IMP.Model, m



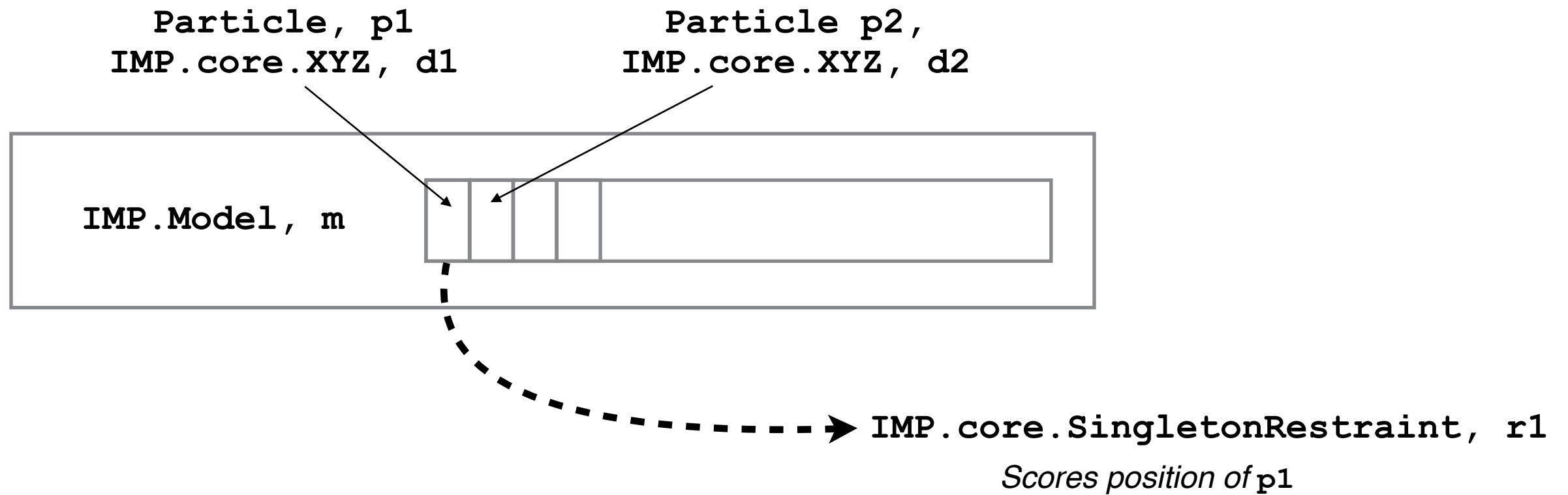
Overall workflow



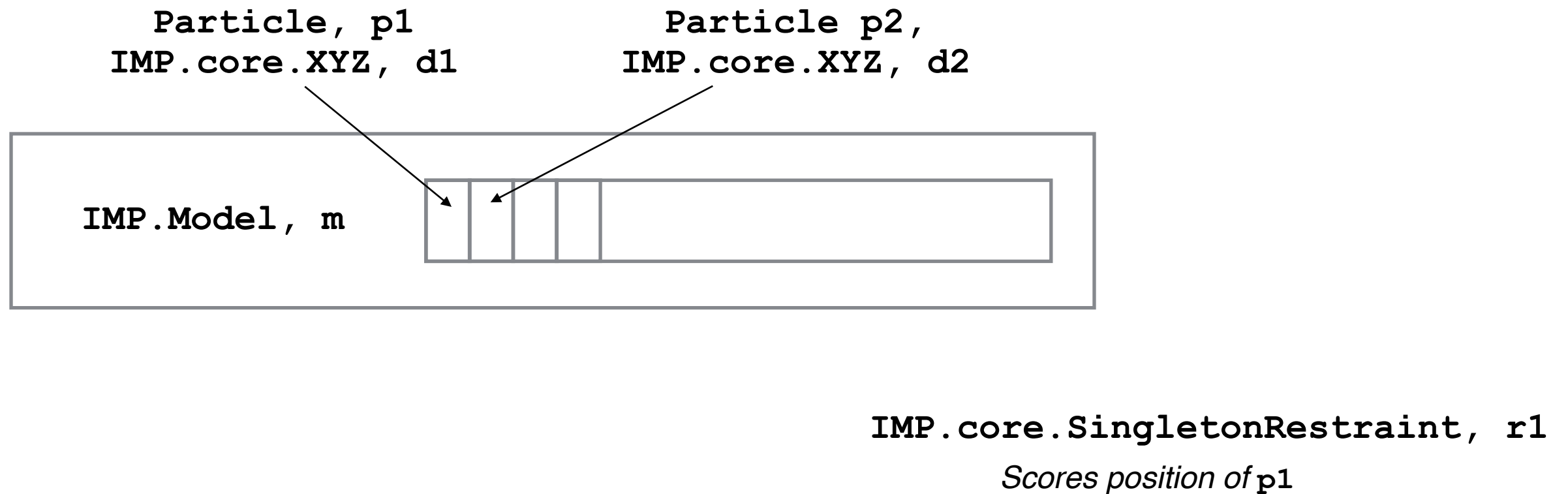
Overall workflow



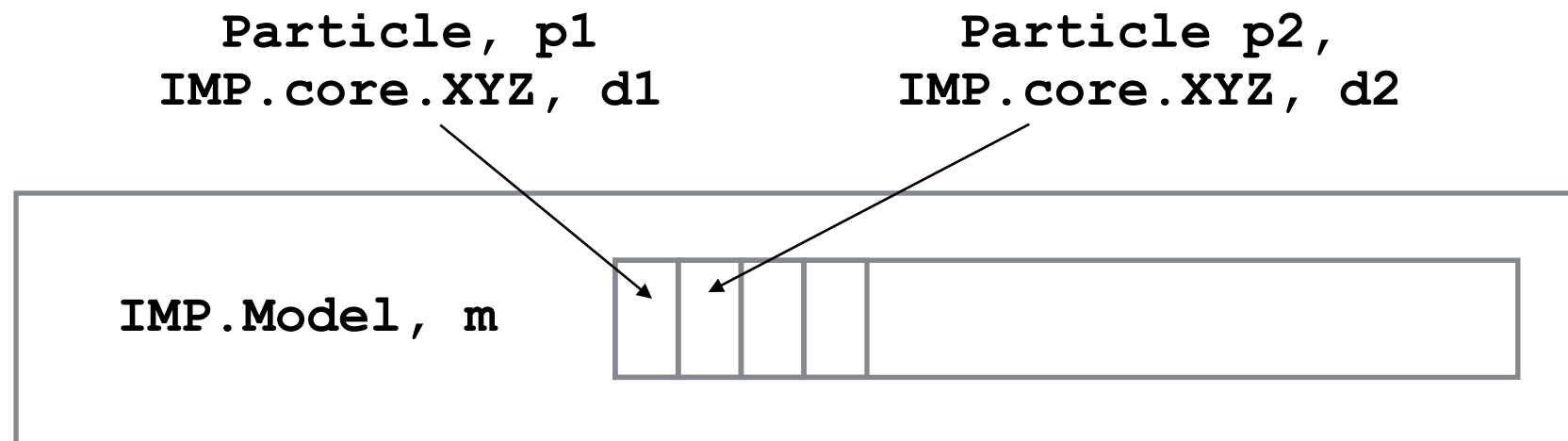
Overall workflow



Overall workflow



Overall workflow

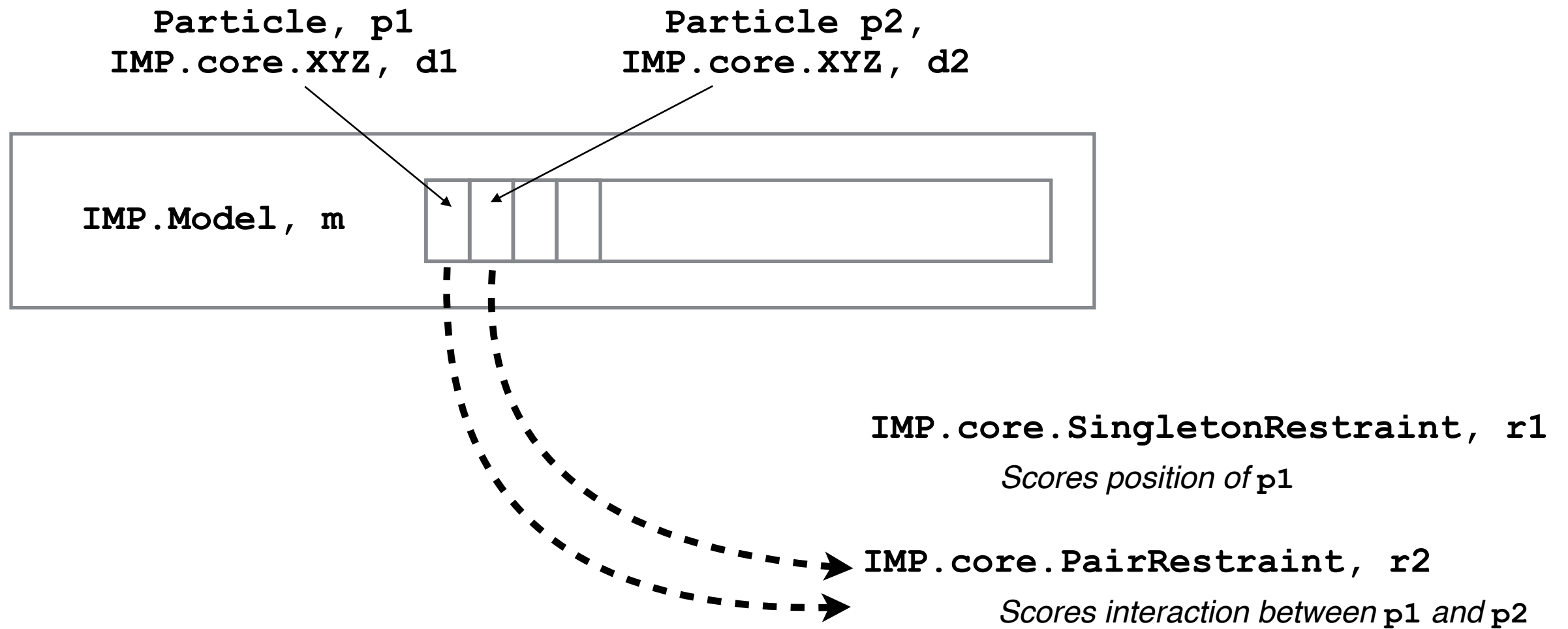


`IMP.core.SingletonRestraint, r1`

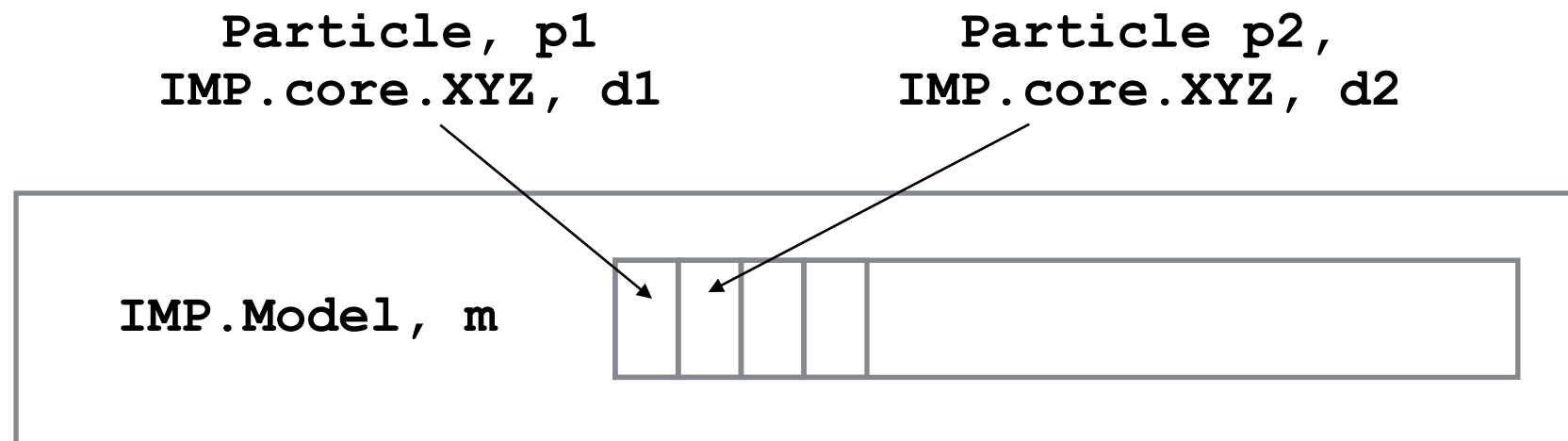
Scores position of p1

`IMP.core.PairRestraint, r2`

Overall workflow



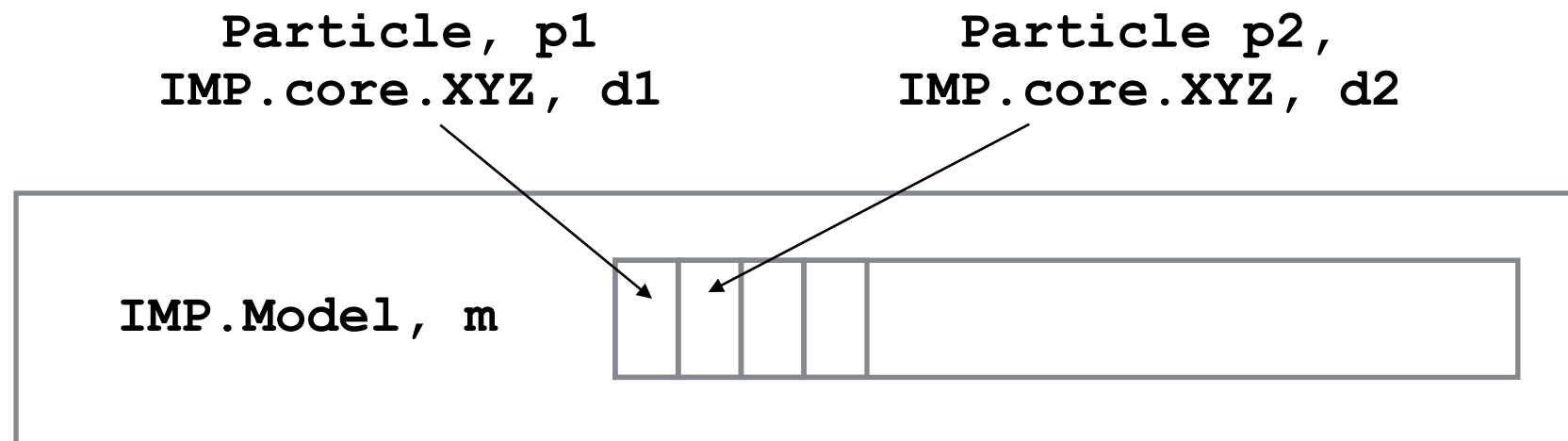
Overall workflow



`IMP.core.SingletonRestraint, r1`
Scores position of p1

`IMP.core.PairRestraint, r2`
Scores interaction between p1 and p2

Overall workflow

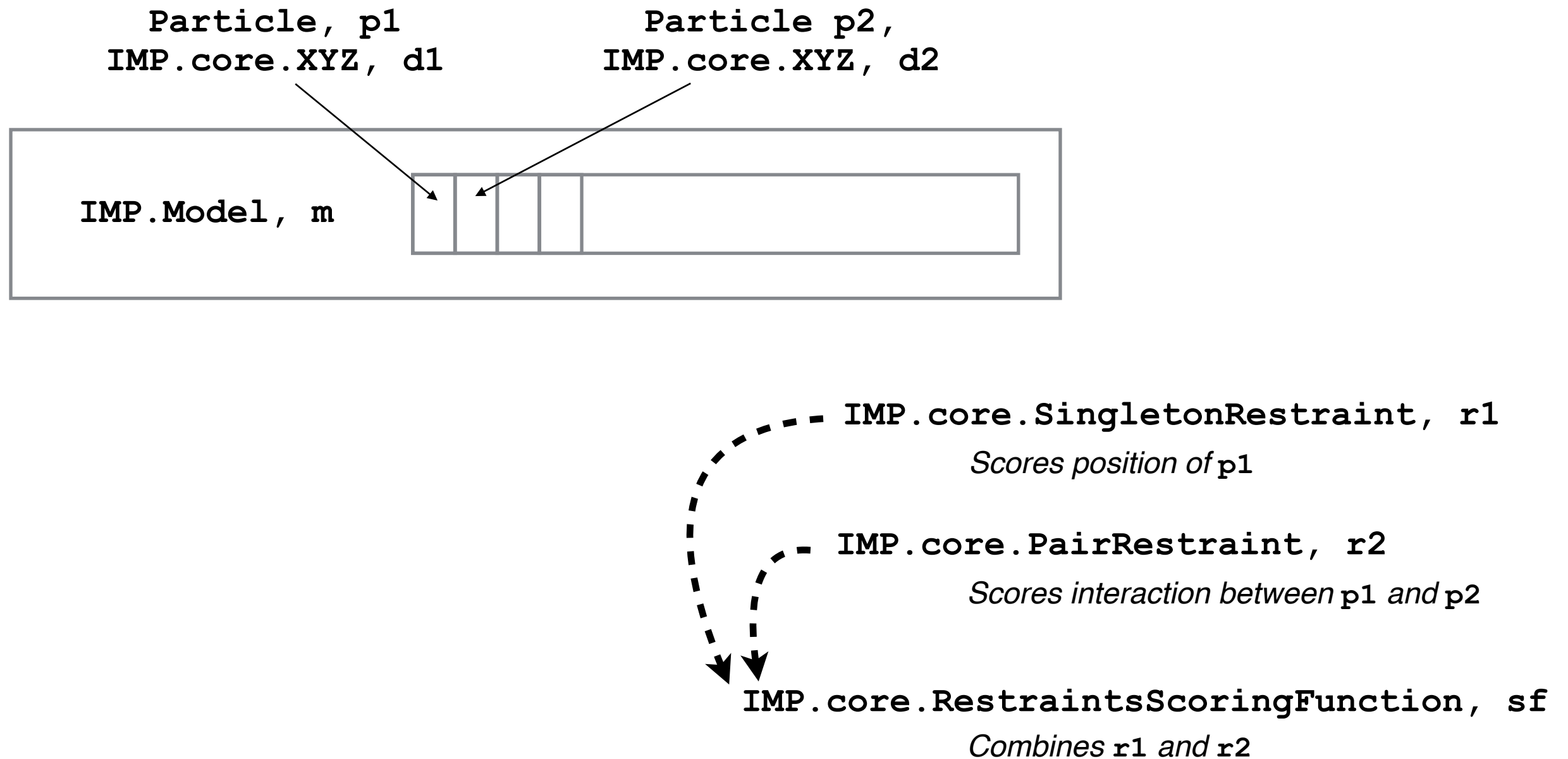


`IMP.core.SingletonRestraint, r1`
Scores position of p1

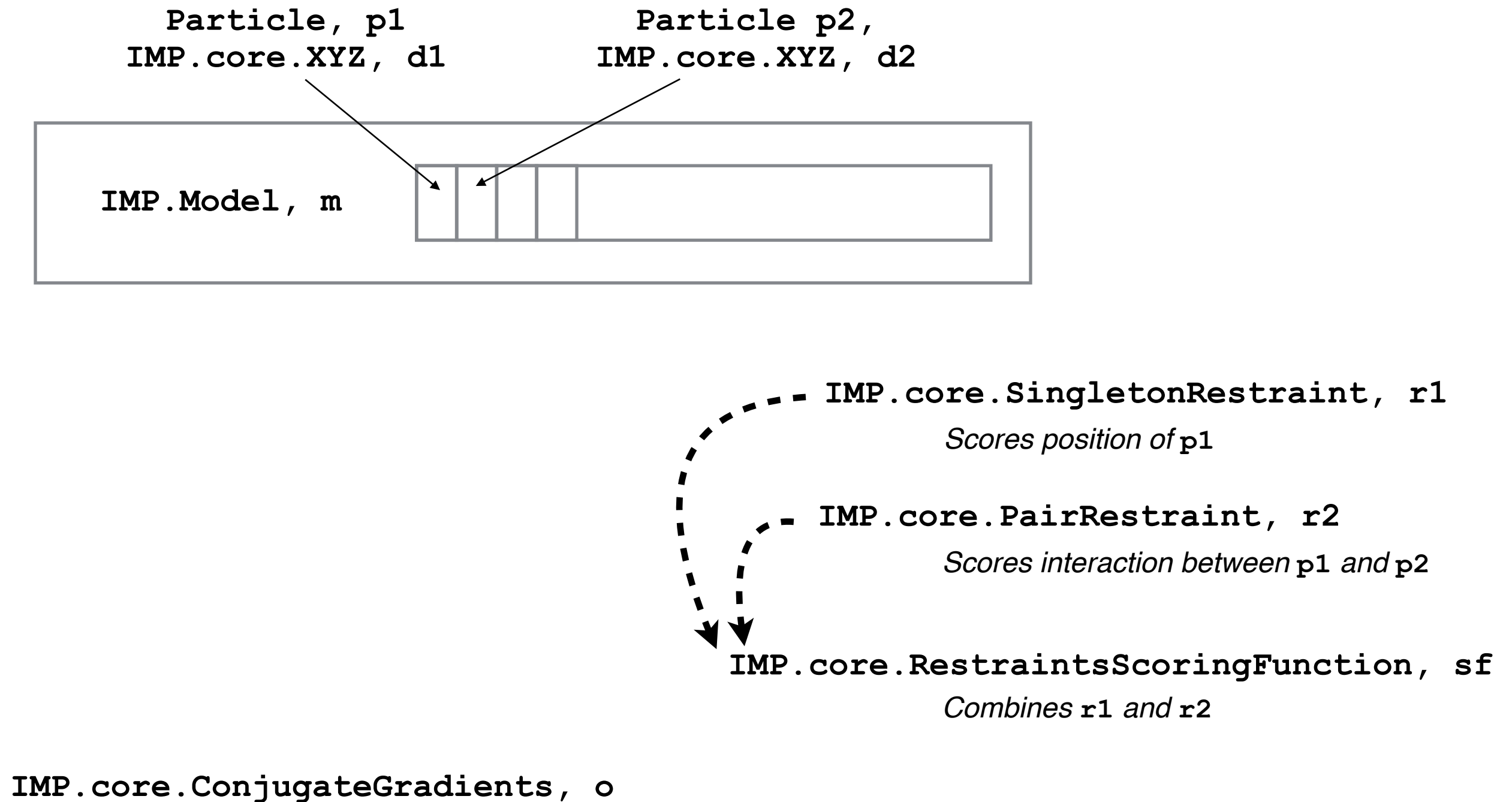
`IMP.core.PairRestraint, r2`
Scores interaction between p1 and p2

`IMP.core.RestraintsScoringFunction, sf`

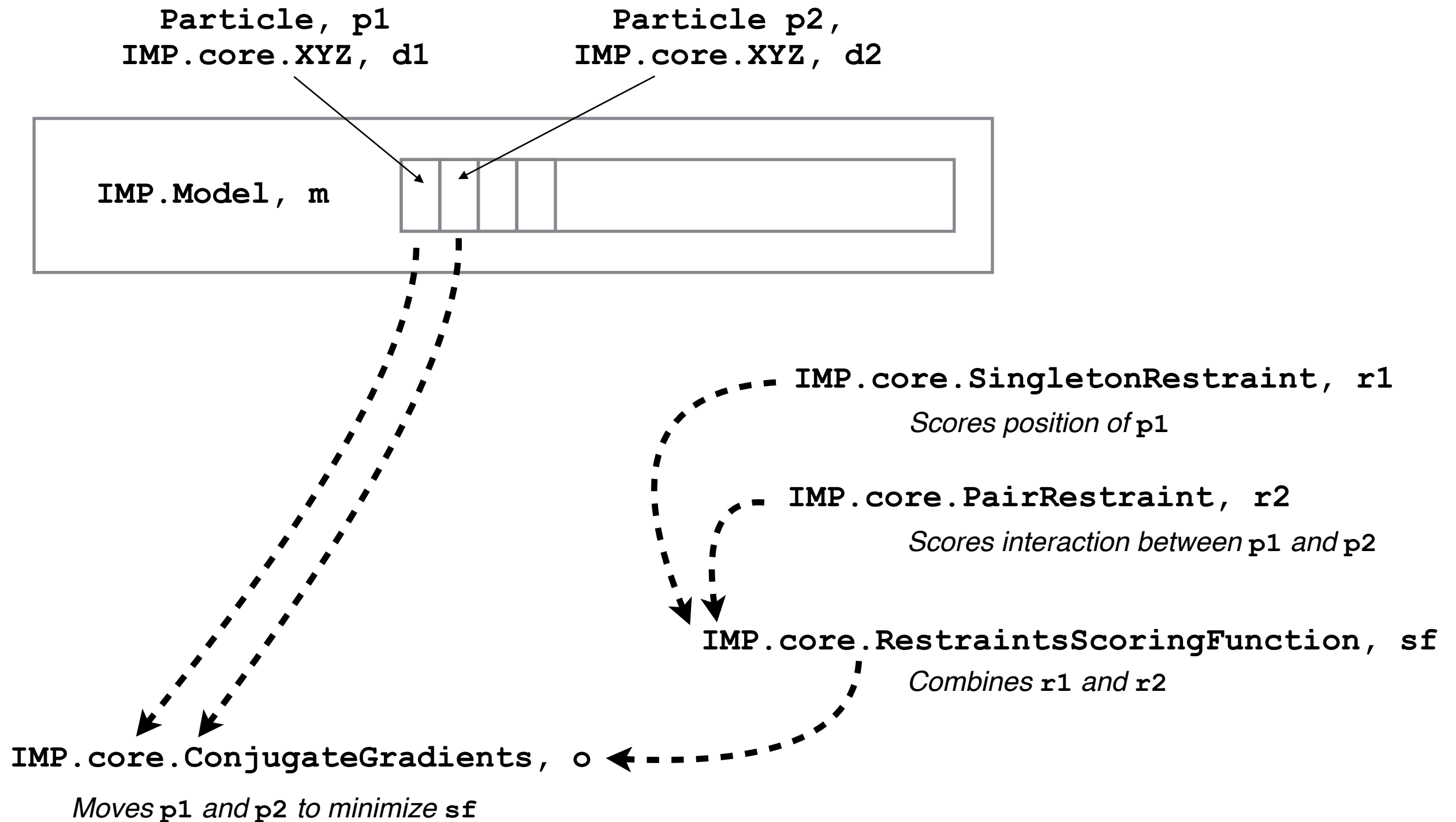
Overall workflow



Overall workflow



Overall workflow



Example Python script

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

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

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

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

IMP installation

IMP installation


- Easiest cross-platform (Windows, Mac, Linux) way is to install Anaconda Python (either Miniconda or the full Anaconda, 2 or 3), then run from a command prompt/terminal:

```
$ conda config --add channels salilab  
$ conda install imp
```


IMP installation

- Easiest cross-platform (Windows, Mac, Linux) way is to install Anaconda Python (either Miniconda or the full Anaconda, 2 or 3), then run from a command prompt/terminal:

```
$ conda config --add channels salilab  
$ conda install imp
```




The dollar sign (\$) here represents your command prompt (e.g. from Terminal on a Mac, Command Prompt on a Windows machine, or a Linux command line). Everything *following* the \$ should be typed at the command prompt.

IMP installation

- Easiest cross-platform (Windows, Mac, Linux) way is to install Anaconda Python (either Miniconda or the full Anaconda, 2 or 3), then run from a command prompt/terminal:

```
$ conda config --add channels salilab  
$ conda install imp
```



The dollar sign (\$) here represents your command prompt (e.g. from Terminal on a Mac, Command Prompt on a Windows machine, or a Linux command line). Everything *following* the \$ should be typed at the command prompt.

- Can also install IMP from source code, native package (.exe, .dmg, .rpm, .deb), or Homebrew (Mac) but you still need to figure out how to get other Python packages (e.g. via pip)

IMP installation

- Test that it installed correctly:

```
$ python
```

```
Python 3.7.0 (default, Oct 9 2018,  
10:31:47)
```

```
[GCC 7.3.0] :: Anaconda custom (64-  
bit) on linux
```

```
Type "help", "copyright", "credits" or  
"license" for more information.
```

```
>>> import IMP
```

```
>>> IMP.__version__  
'2.9.0'
```

```
>>> x = IMP.get_example_path('.')
```

```
>>> exit()
```

```
$
```


IMP installation

- Test that it installed correctly:

```
$ python
```

```
Python 3.7.0 (default, Oct 9 2018,  
10:31:47)
```

```
[GCC 7.3.0] :: Anaconda custom (64-  
bit) on linux
```

```
Type "help", "copyright", "credits" or  
"license" for more information.
```

```
>>> import IMP
```

```
>>> IMP.__version__  
'2.9.0'
```

```
>>> x = IMP.ge
```

```
>>> exit()
```

```
$
```

>>> is the Python prompt. Everything *following* the >>> should be typed into a Python interpreter (not the command prompt)

IMP installation

- Test that it installed correctly:

```
$ python
```

```
Python 3.7.0 (default, Oct 9 2018,  
10:31:47)
```

```
[GCC 7.3.0] :: Anaconda custom (64-  
bit) on linux
```

```
Type "help", "copyright", "credits" or  
"license" for more information.
```

```
>>> import IMP
```

```
>>> IMP.__version__  
'2.9.0'
```

```
>>> x = IMP.g
```

```
>>> exit()
```

```
$
```

These are double-underscores. Variables starting and ending with double-underscores have special meaning in Python (this one is the version of the module)

IMP installation

- Test that it installed correctly:

```
$ python
```

```
Python 3.7.0 (default, Oct 9 2018,  
10:31:47)
```

```
[GCC 7.3.0] :: Anaconda custom (64-  
bit) on linux
```

```
Type "help", "copyright", "credits" or  
"license" for more information.
```

```
>>> import IMP
```

```
>>> IMP.__version__  
'2.9.0'
```

```
>>> x = IMP.get_example_path('.')  
      ^
```

```
>>> exit()  
$
```



Parentheses () usually denote a function call.
This function should print nothing if all is OK.

IMP installation

- Test that it installed correctly:

```
$ python
```

```
Python 3.7.0 (default, Oct 9 2018,  
10:31:47)
```

```
[GCC 7.3.0] :: Anaconda custom (64-  
bit) on linux
```

```
Type "help", "copyright", "credits" or  
"license" for more information.
```


```
>>> import IMP
```

```
>>> IMP.__version__  
'2.9.0'
```

```
>>> x = IMP.get_example_path('.')
```

```
>>> exit()
```

```
$
```



The exit() function leaves the Python interpreter and drops us back at the command prompt

IMP installation

- Test that it installed correctly:

```
$ python
```

```
Python 3.7.0 (default, Oct 9 2018,  
10:31:47)
```

```
[GCC 7.3.0] :: Anaconda custom (64-  
bit) on linux
```

```
Type "help", "copyright", "credits" or  
"license" for more information.
```

```
>>> import IMP
```

```
>>> IMP.__version__  
'2.9.0'
```

```
>>> x = IMP.get_example_path('.')
```

```
>>> exit()
```

```
$
```


Running the script

Running the script

- First, determine where it is (it is included with IMP, as an example for the 'core' module):
\$ python
>>> import IMP.core
>>> IMP.core.get_example_path('simple.py')

Running the script

- First, determine where it is (it is included with IMP, as an example for the 'core' module):
\$ python
>>> import IMP.core
>>> IMP.core.get_example_path('simple.py')

Should just print a full path to 'simple.py'; if not, ask for help

Running the script

- First, determine where it is (it is included with IMP, as an example for the 'core' module):

```
$ python
```

```
>>> import IMP.core
```

```
>>> IMP.core.get_example_path('simple.py')
```

Should just print a full path to 'simple.py'; if not, ask for help

- Then, copy it to your working directory/folder:

```
$ mkdir simple_script
```

```
$ cd simple_script
```

```
$ cp <path_to_simple.py> .
```


Running the script

- First, determine where it is (it is included with IMP, as an example for the 'core' module):

```
$ python
```

```
>>> import IMP.core
```

```
>>> IMP.core.get_example_path('simple.py')
```


Should just print a full path to 'simple.py'; if not, ask for help

- Then, copy it to your working directory/folder:

```
$ mkdir simple_script
```

```
$ cd simple_script
```

```
$ cp <path_to_simple.py> .
```



Windows users, use 'copy' rather than 'cp' and \ rather than \\ or / in filenames/paths.

Running the script

- First, determine where it is (it is included with IMP, as an example for the 'core' module):

```
$ python
```

```
>>> import IMP.core
```

```
>>> IMP.core.get_example_path('simple.py')
```

Should just print a full path to 'simple.py'; if not, ask for help

- Then, copy it to your working directory/folder:

```
$ mkdir simple_script
```

```
$ cd simple_script
```

```
$ cp <path_to_simple.py> .
```

Windows users, use 'copy' rather than 'cp' and \ rather than \\ or / in filenames/paths.

- Finally, run it:

```
$ python simple.py
```


Python vs. C++

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

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

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

```
#include <IMP.h>
#include <IMP/algebra.h>
#include <IMP/core.h>

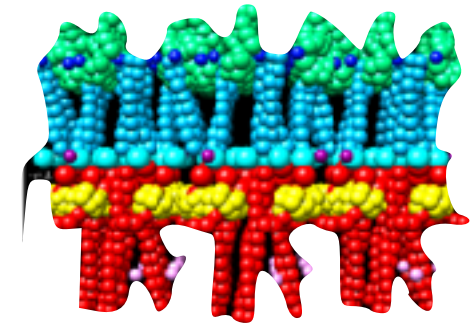
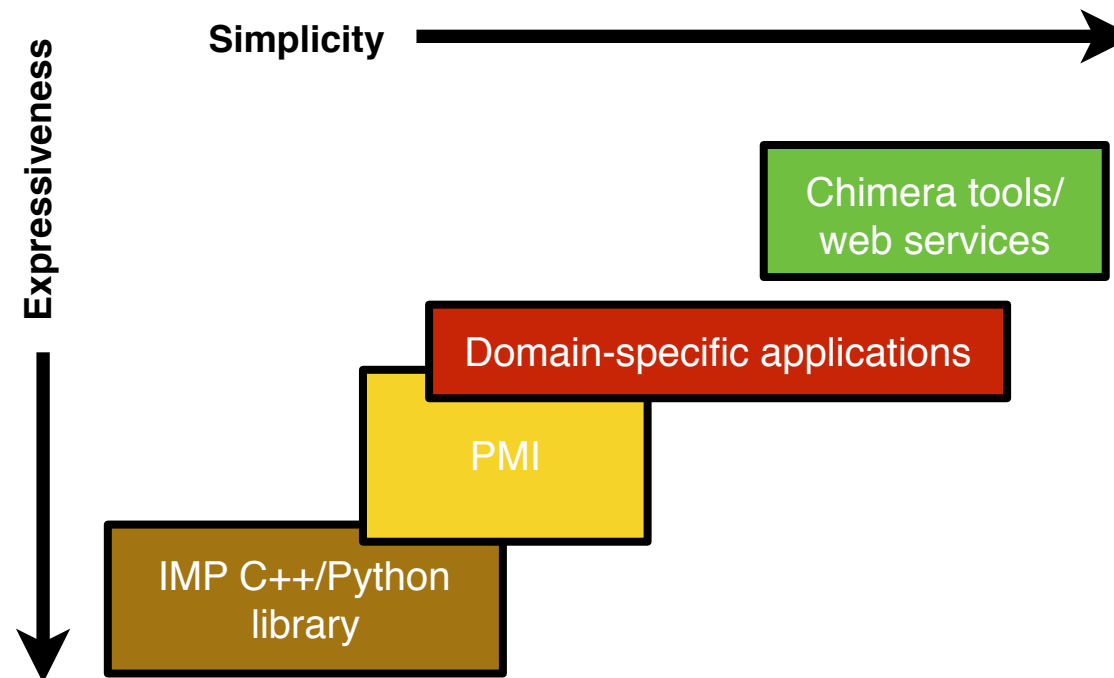
int main() {
    IMP_NEW(IMP::Model, m, ());
    // Create two "untyped" particles
    IMP::ParticleIndex p1 = m->add_particle("p1");
    IMP::ParticleIndex p2 = m->add_particle("p2");

    // "Decorate" the particles with x,y,z attributes
    // (point-like particles)
    IMP::core::XYZ d1 = IMP::core::XYZ::setup_particle(m, p1);
    IMP::core::XYZ d2 = IMP::core::XYZ::setup_particle(m, 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));
    std::cout << d1 << " " << d2 << std::endl;
}
```

- Note that usage from C++ is very similar (main differences are in language syntax, typing, and memory management)

Higher level interfaces



- Very flexible, for specialist modeling, e.g. <https://salilab.org/spb/>
- In practice, scripts for most modeling problems would be too long and unwieldy to write this way
- Most usage of IMP is via simpler (but less flexible or expressive) interfaces, which we'll look at next

Chimera tools/
web services

Chimera tools/
web services

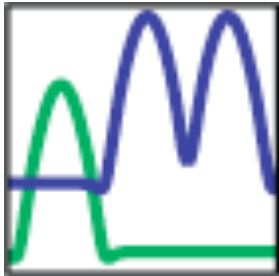
- Several plugins to UCSF Chimera that use IMP

Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:

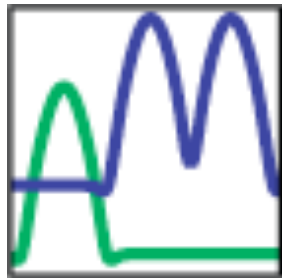
Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:



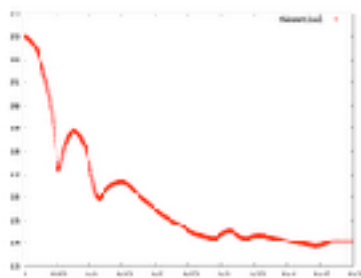
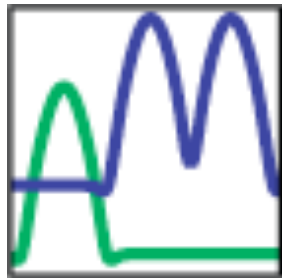
Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:
- AllosMod: modeling of ligand-induced protein dynamics, allostery



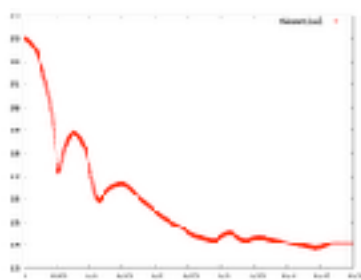
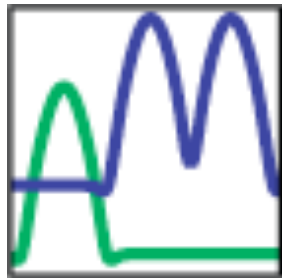
Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:
- AllosMod: modeling of ligand-induced protein dynamics, allostery



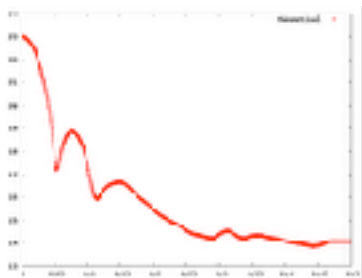
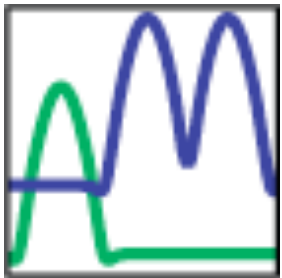
Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:
- AllosMod: modeling of ligand-induced protein dynamics, allostery
- FoXS: fast SAXS profile computation with Debye formula



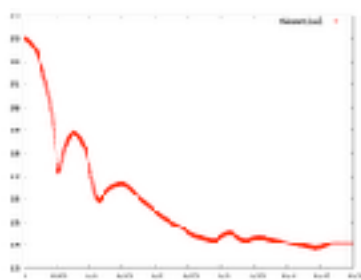
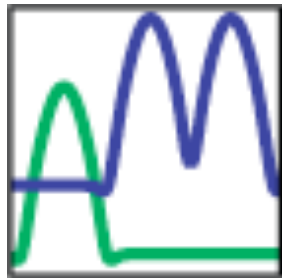
Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:
- AllosMod: modeling of ligand-induced protein dynamics, allostery
- FoXS: fast SAXS profile computation with Debye formula
- FoXSDock: macromolecular docking with SAXS Profile



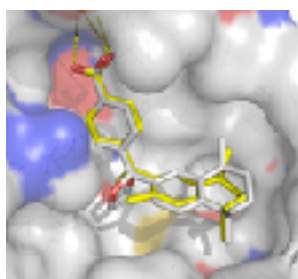
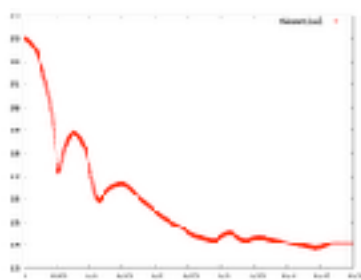
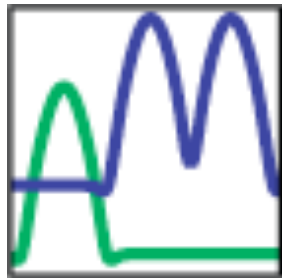
Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:
- 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



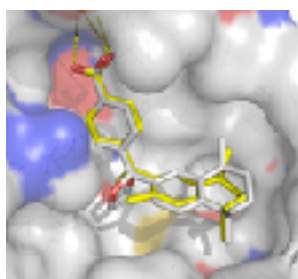
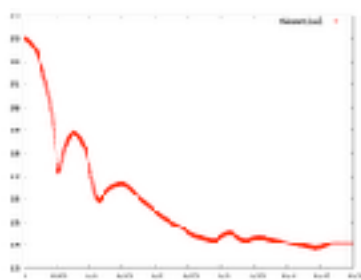
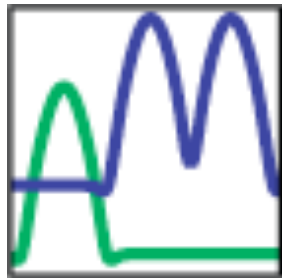
Chimera tools/ web services

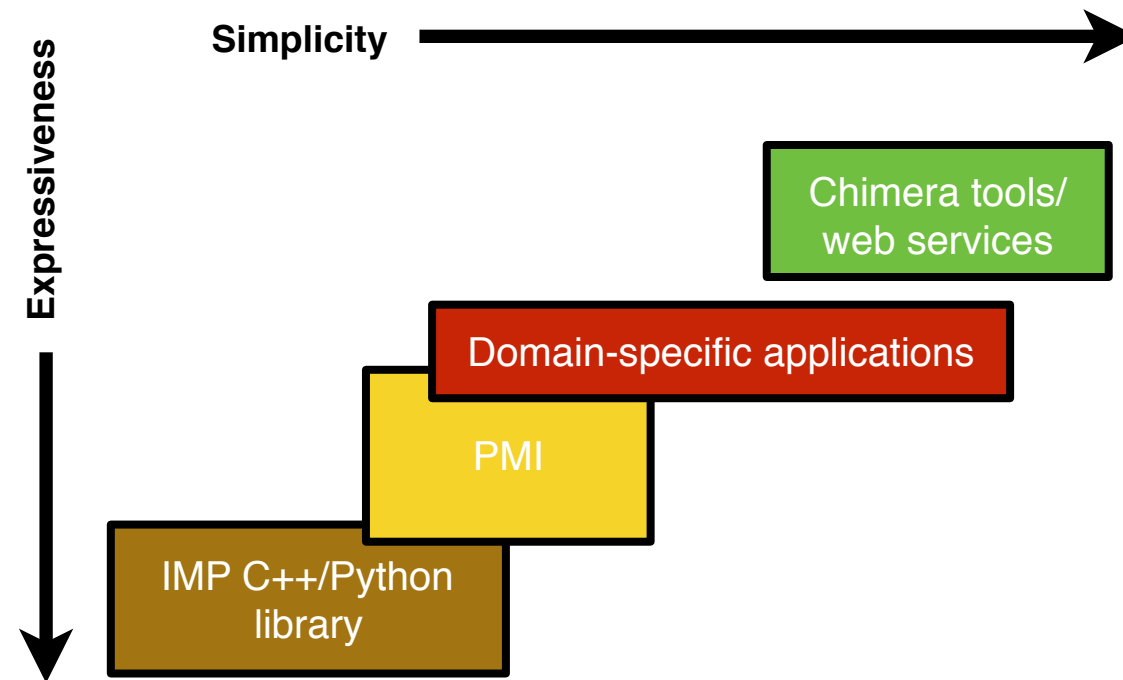
- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:
- AllosMod: modeling of ligand-induced protein dynamics, allostery
- FoXS: fast SAXS profile computation with Debye formula
- FoXSDock: macromolecular docking with SAXS Profile
- SAXSMerge: automated statistical method to merge SAXS profiles from different concentrations and exposure times



Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP
- Web services at <https://salilab.org/> including:
- 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





Domain-specific applications

Domain-specific applications

- Command line tools

Domain-specific applications

- Command line tools
- Perform a very specific task, a subset of IMP functionality

Domain-specific applications

- Command line tools
- Perform a very specific task, a subset of IMP functionality
- Generally, similar functionality to web services, but

Domain-specific applications

- Command line tools
- Perform a very specific task, a subset of IMP functionality
- Generally, similar functionality to web services, but
 - running locally

Domain-specific applications

- Command line tools
- Perform a very specific task, a subset of IMP functionality
- Generally, similar functionality to web services, but
 - running locally
 - more adjustable parameters, flexibility

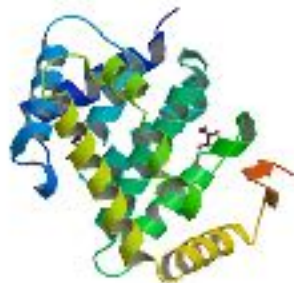
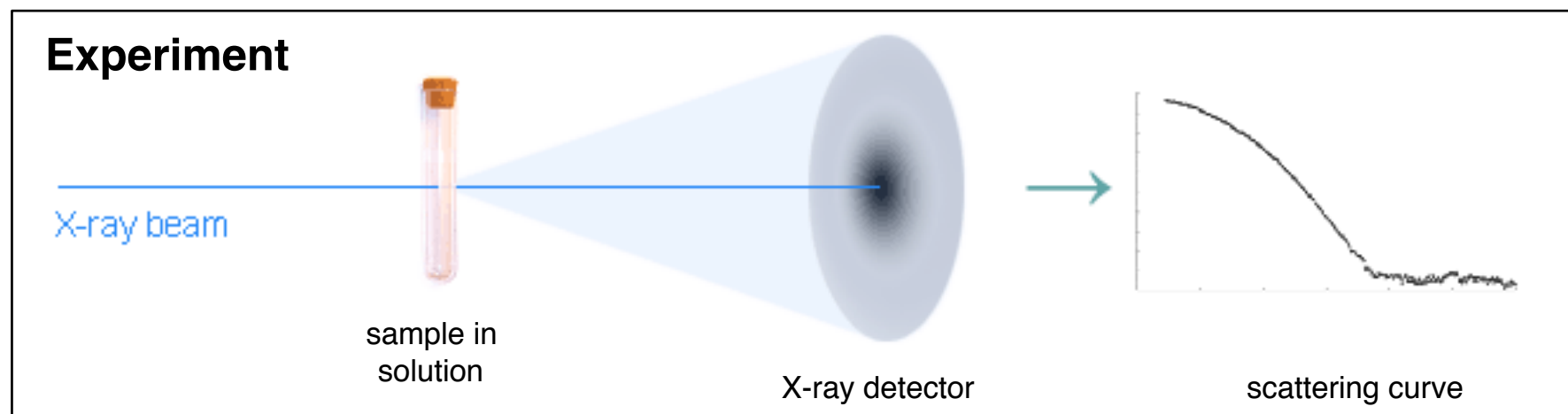
Domain-specific applications

- Command line tools
- Perform a very specific task, a subset of IMP functionality
- Generally, similar functionality to web services, but
 - running locally
 - more adjustable parameters, flexibility
- Let's look briefly at using the **foxs** command line tool to leverage SAXS data

FoXS

FoXS

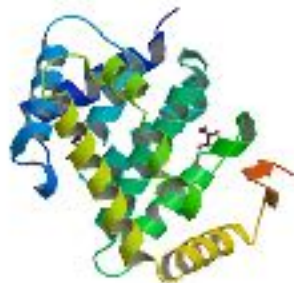
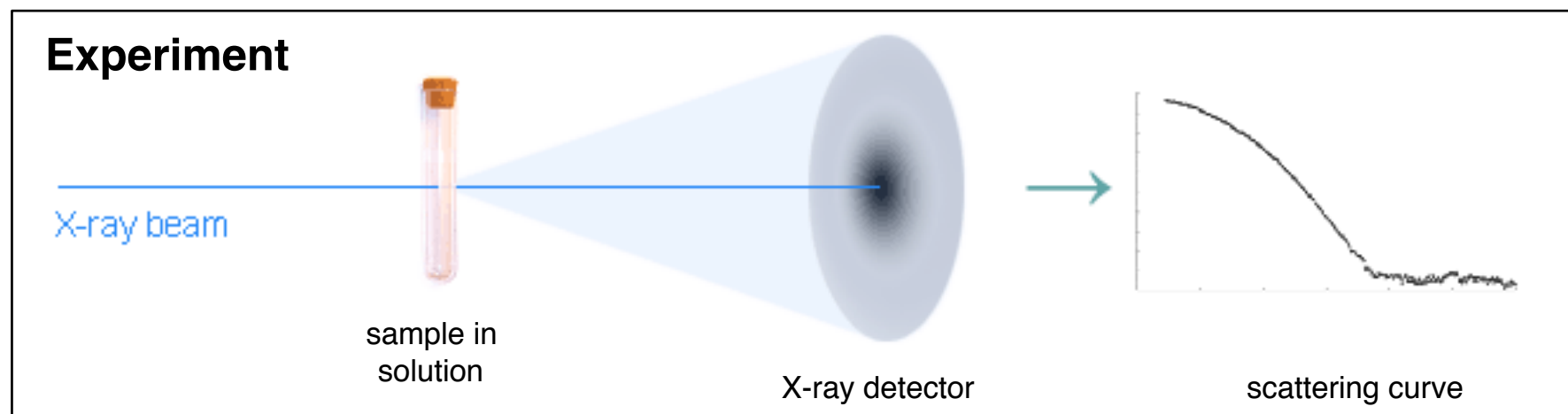
- Given an experimental SAXS profile and a 3D model, FoXS:



3D model

FoXS

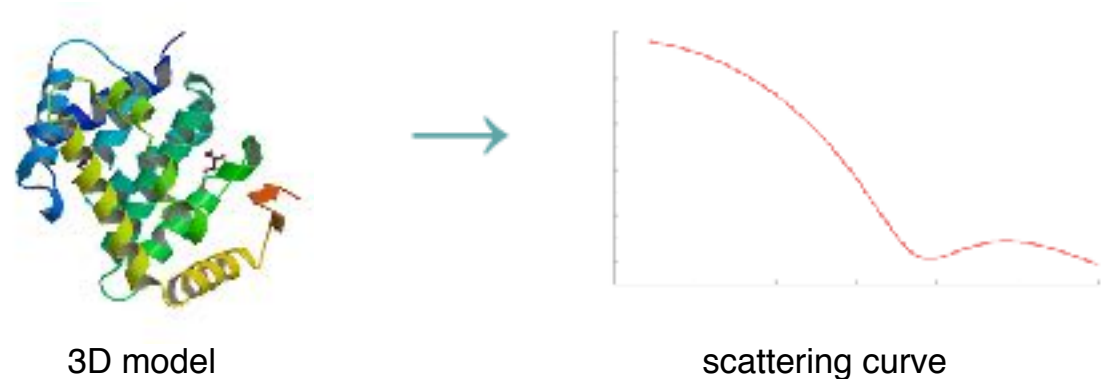
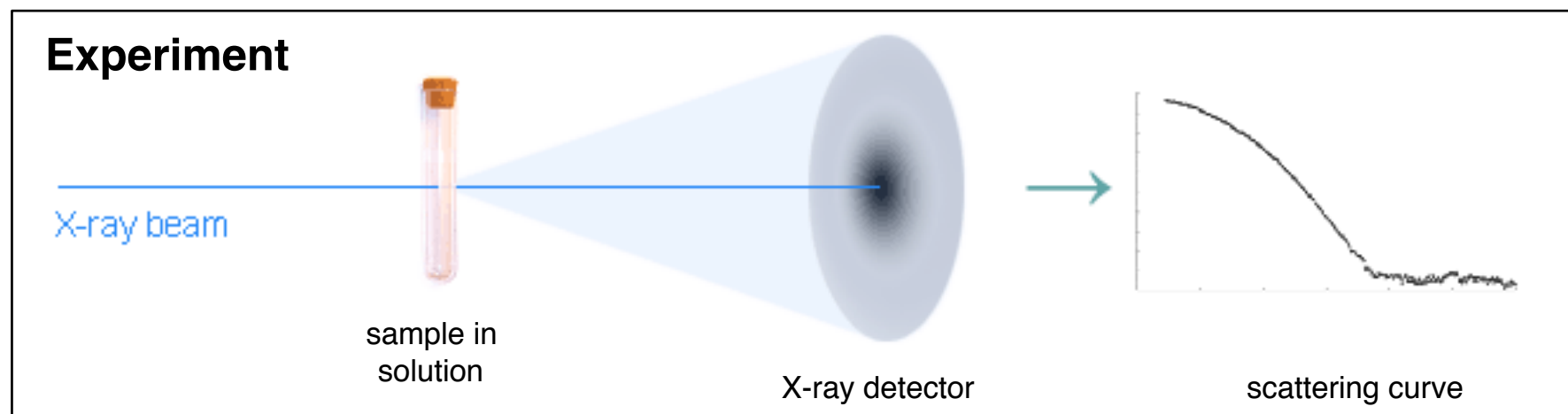
- Given an experimental SAXS profile and a 3D model, FoXS:
 - Calculates the theoretical profile of the model



3D model

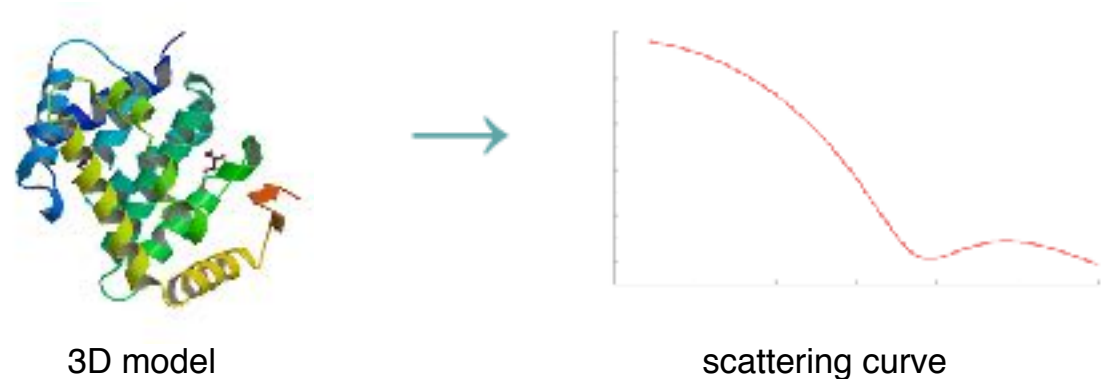
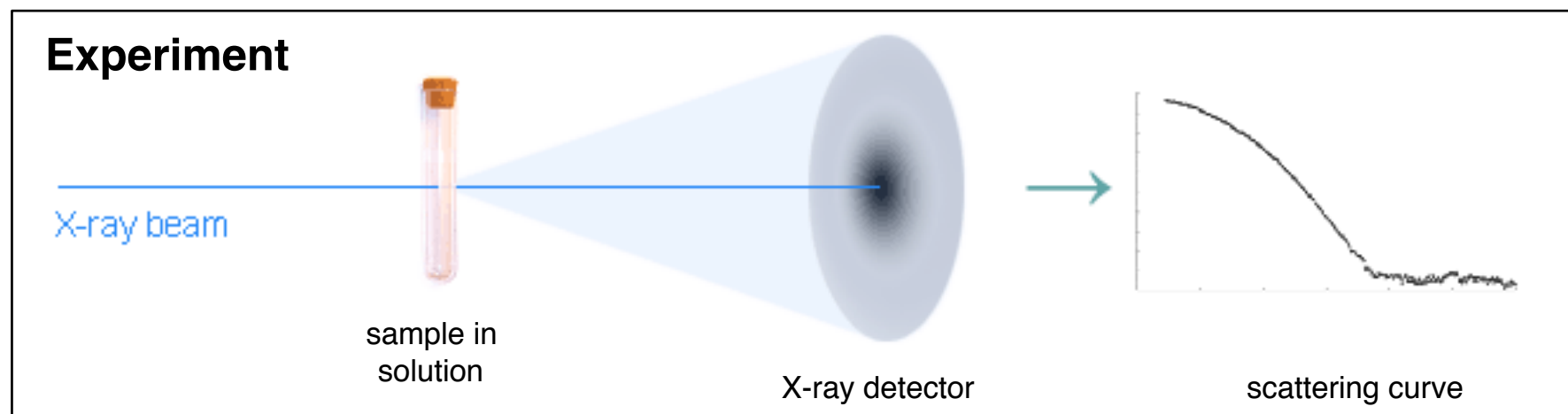
FoXS

- Given an experimental SAXS profile and a 3D model, FoXS:
 - Calculates the theoretical profile of the model



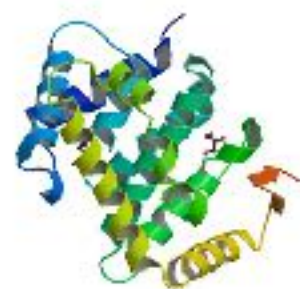
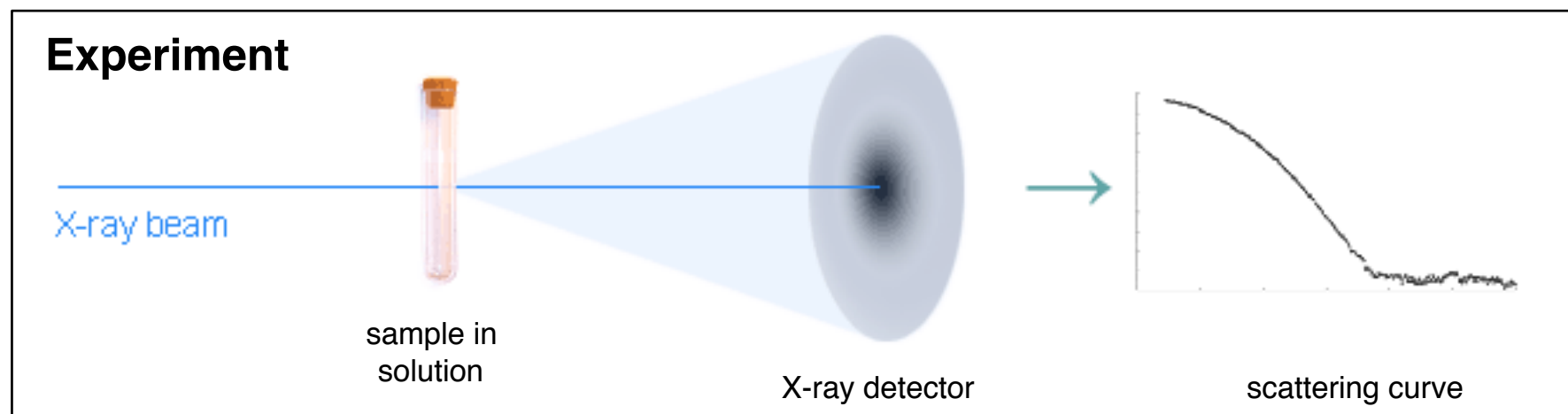
FoXS

- Given an experimental SAXS profile and a 3D model, FoXS:
 - Calculates the theoretical profile of the model
 - Fits the two profiles together and reports a fit value, χ

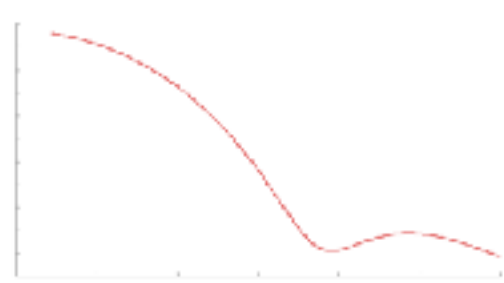


FoXS

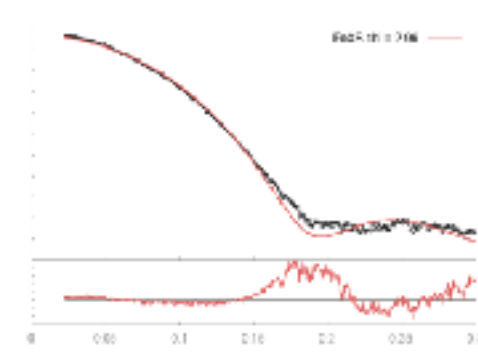
- Given an experimental SAXS profile and a 3D model, FoXS:
 - Calculates the theoretical profile of the model
 - Fits the two profiles together and reports a fit value, χ



3D model

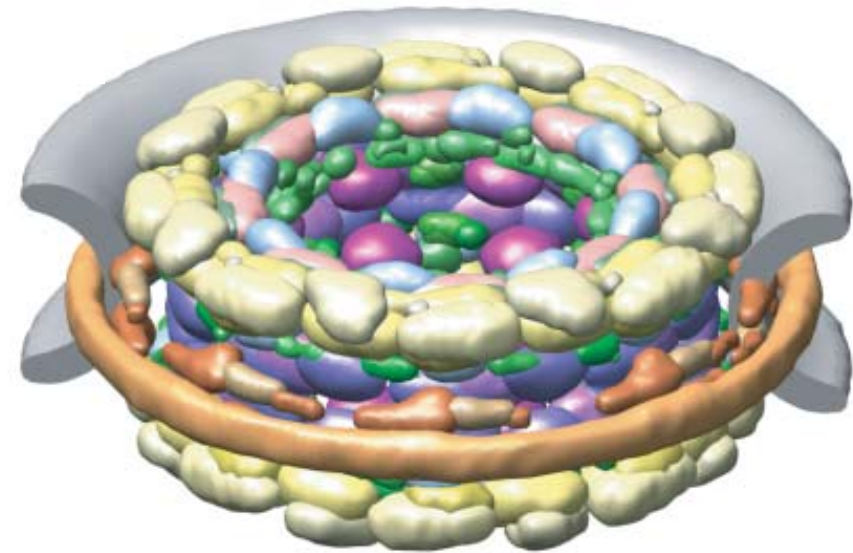


scattering curve



FoXS usage

- Here we'll use FoXS to improve the structure of the C terminal domain of Nup133, one of the subunits of the Nup84 subcomplex of the NPC

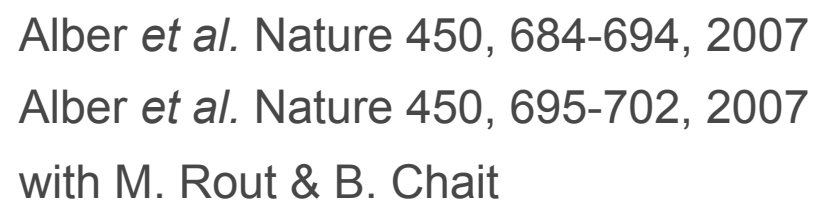


Alber *et al.* Nature 450, 684-694, 2007

Alber *et al.* Nature 450, 695-702, 2007

with M. Rout & B. Chait

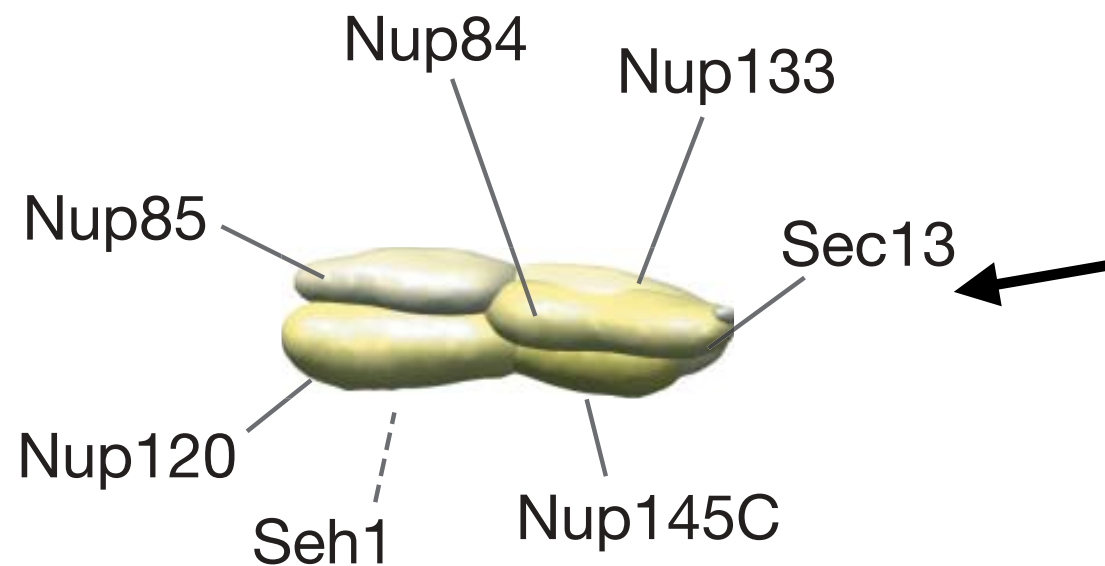
- Here we'll use FoXS to improve the structure of the C terminal domain of Nup133, one of the subunits of the Nup84 subcomplex of the NPC



Alber *et al.* Nature 450, 684-694, 2007
Alber *et al.* Nature 450, 695-702, 2007
with M. Rout & B. Chait

FoXS usage

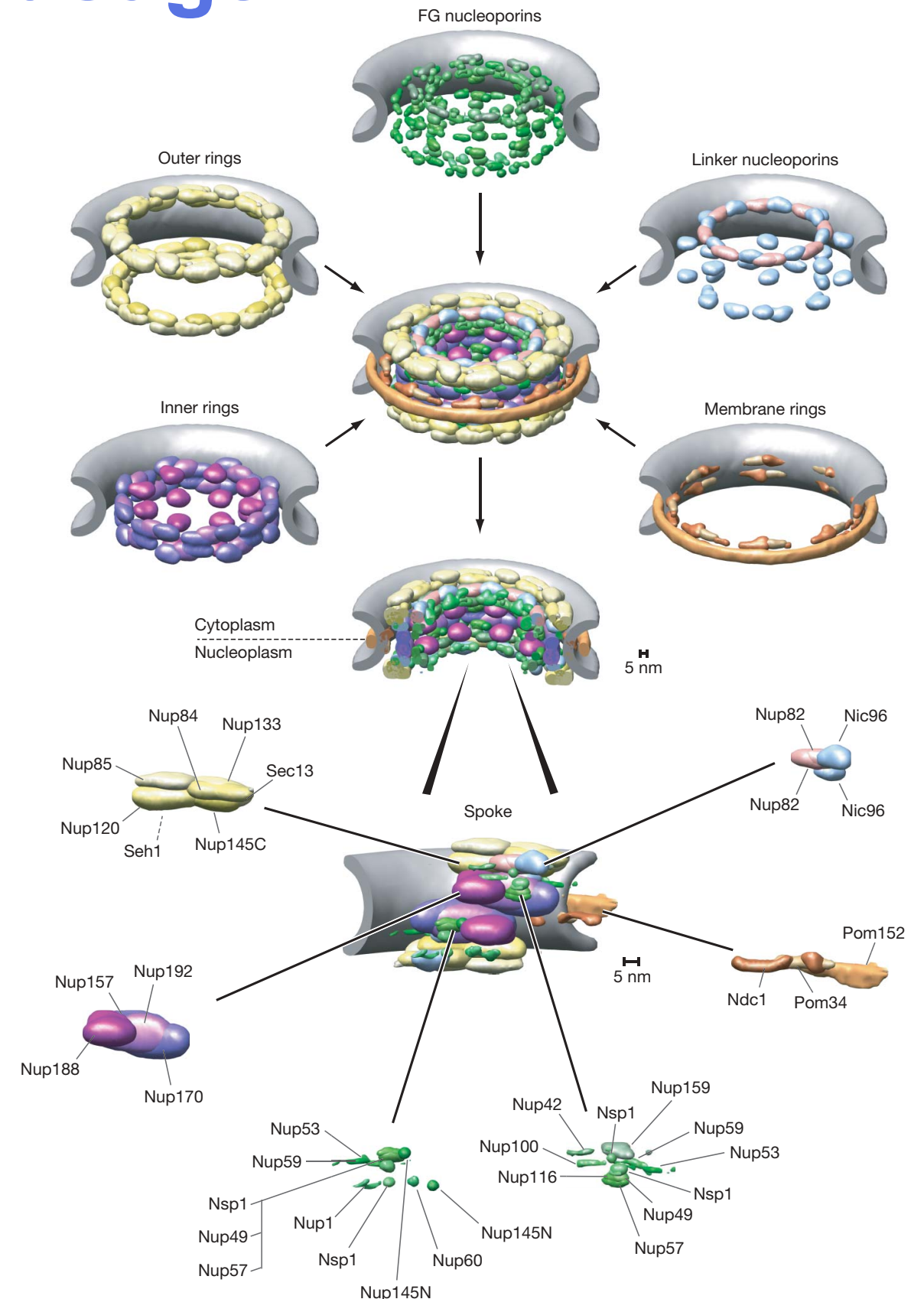
- Here we'll use FoXS to improve the structure of the C terminal domain of Nup133, one of the subunits of the Nup84 subcomplex of the NPC



Alber *et al.* Nature 450, 684-694, 2007

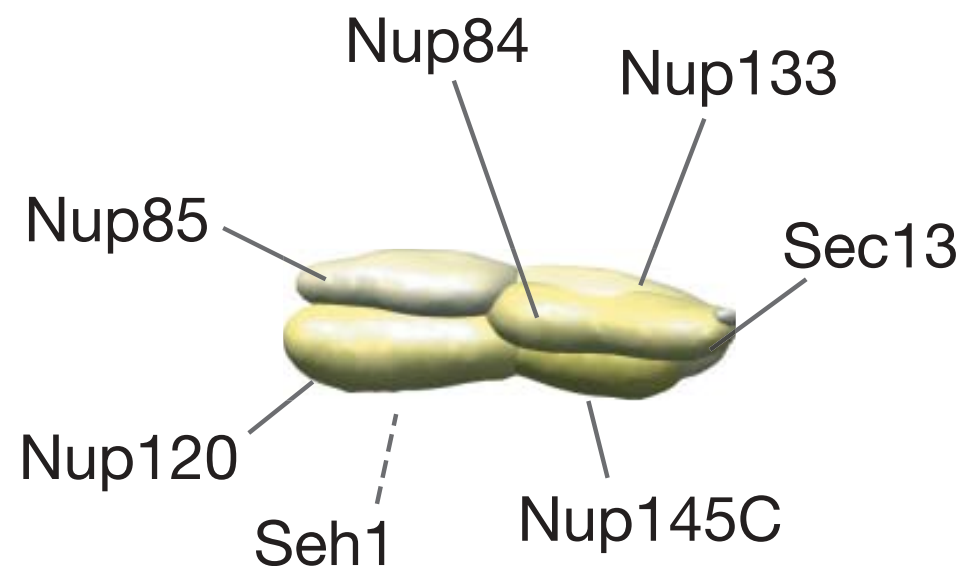
Alber *et al.* Nature 450, 695-702, 2007

with M. Rout & B. Chait



FoXS usage

- SAXS is rotationally averaged so we can't *predict* an X-ray-like structure, but we can *check consistency* with an existing structure
- We **can** generate *ab initio* SAXS shapes, but they are (typically) degenerate



Get FoXS inputs

- First, determine where they are (again, included as IMP examples, in the 'foxs' module):

```
$ python
>>> import IMP.foxs
>>> IMP.foxs.get_example_path('nup133')
```

- Then, copy them to your working directory/folder:

```
$ mkdir fogs_example
$ cd fogs_example
$ cp <path_to_nup133>/* .
```



Get FoXS inputs

- First, determine where they are (again, included as IMP examples, in the 'foxs' module):

```
$ python
>>> import IMP.foxs
>>> IMP.foxs.get_example_path('nup133')
```

- Then, copy them to your working directory/folder:

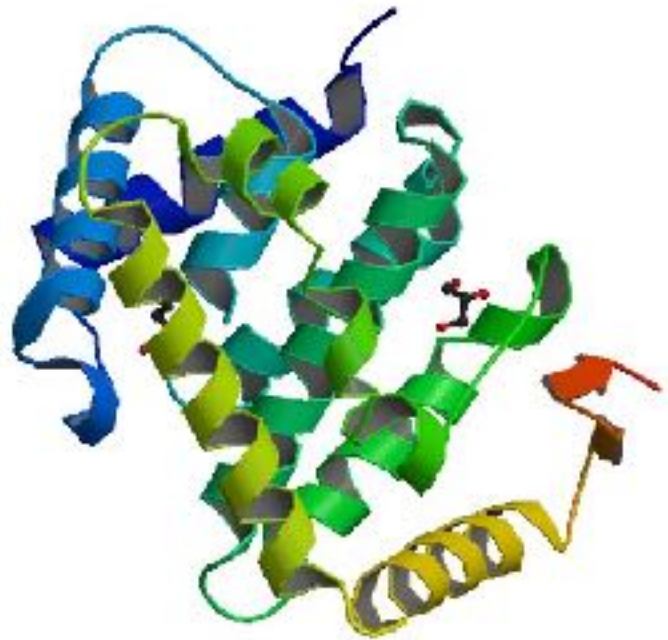
```
$ mkdir fogs_example
$ cd fogs_example
$ cp <path_to_nup133>/* .
```



Windows users, use 'copy' rather than 'cp' and \ rather than /.

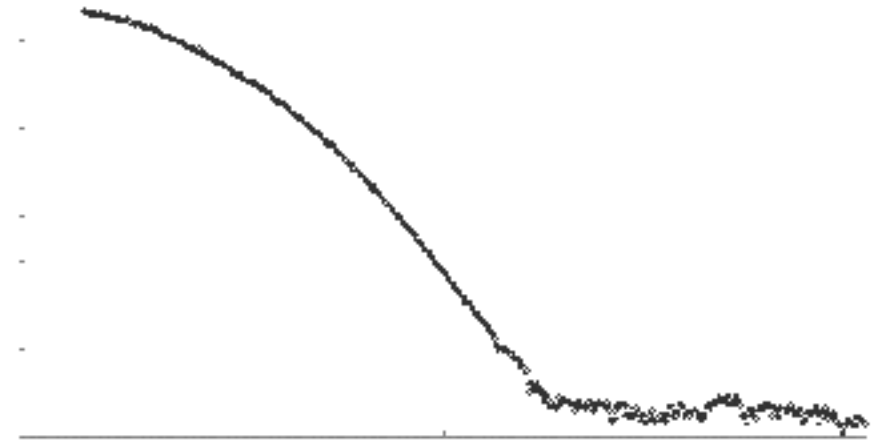
Input files

3KFO.pdb



X-ray crystal structure of the C terminal domain of Nup133, in PDB format

23922_merge.dat



Experimental SAXS profile of the same structure (simple table of intensity vs. angle, plotted here for clarity)

Run FoXS

Run FoXS

- Running FoXS is simple; we just give it the PDB file and the profile:
`$ foxs 3KFO.pdb 23922_merge.dat`

Run FoXS

- Running FoXS is simple; we just give it the PDB file and the profile:
`$ foxs 3KFO.pdb 23922_merge.dat`
- Output will end with something like
`3KFO.pdb 23922_merge.dat Chi = 2.95998 c1 = 1.02509 c2 = 3.3952 default chi = 9.87946`

Run FoXS

- Running FoXS is simple; we just give it the PDB file and the profile:
`$ foxs 3KFO.pdb 23922_merge.dat`
- Output will end with something like
`3KFO.pdb 23922_merge.dat Chi = 2.95998 c1 = 1.02509 c2 = 3.3952 default chi = 9.87946`
- i.e. quality of fit (χ) is 2.96 (smaller is better, so this is not great)

Why such a poor fit?

Why such a poor fit?

- Both the X-ray structure and the SAXS profile were collected for the same structure, so shouldn't they match?

Why such a poor fit?

- Both the X-ray structure and the SAXS profile were collected for the same structure, so shouldn't they match?
- Let's look at the `3KFO.pdb` file in a text editor (not a molecular viewer), specifically REMARK 465, 470 and 999 lines

Why such a poor fit?

- Both the X-ray structure and the SAXS profile were collected for the same structure, so shouldn't they match?
- Let's look at the `3KFO.pdb` file in a text editor (not a molecular viewer), specifically REMARK 465, 470 and 999 lines
 - proteolysis removed residues 881-943 (REMARK 999) so these weren't seen in either experiment

Why such a poor fit?

- Both the X-ray structure and the SAXS profile were collected for the same structure, so shouldn't they match?
- Let's look at the `3KFO.pdb` file in a text editor (not a molecular viewer), specifically REMARK 465, 470 and 999 lines
 - proteolysis removed residues 881-943 (REMARK 999) so these weren't seen in either experiment
 - the X-ray experiment was unable to resolve residues 944, 945, and 1159-1165 (REMARK 465)

Why such a poor fit?

- Both the X-ray structure and the SAXS profile were collected for the same structure, so shouldn't they match?
- Let's look at the `3KFO.pdb` file in a text editor (not a molecular viewer), specifically REMARK 465, 470 and 999 lines
 - proteolysis removed residues 881-943 (REMARK 999) so these weren't seen in either experiment
 - the X-ray experiment was unable to resolve residues 944, 945, and 1159-1165 (REMARK 465)
 - 16 other residues in the X-ray experiment had unresolved side chains (REMARK 470)

Why such a poor fit?

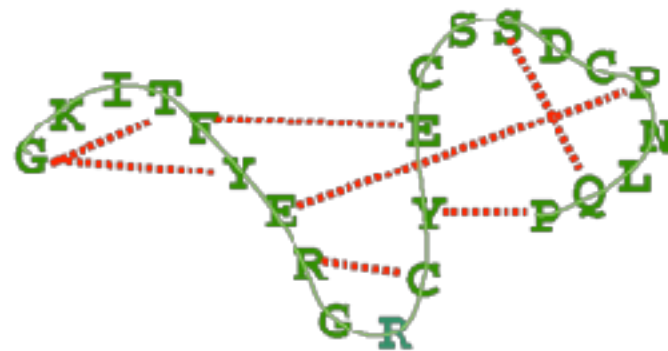
- Both the X-ray structure and the SAXS profile were collected for the same structure, so shouldn't they match?
- Let's look at the `3KFO.pdb` file in a text editor (not a molecular viewer), specifically REMARK 465, 470 and 999 lines
 - proteolysis removed residues 881-943 (REMARK 999) so these weren't seen in either experiment
 - the X-ray experiment was unable to resolve residues 944, 945, and 1159-1165 (REMARK 465)
 - 16 other residues in the X-ray experiment had unresolved side chains (REMARK 470)
- We can resolve these issues by filling in the missing residues with MODELLER

Comparative modeling by satisfaction of spatial restraints: MODELLER

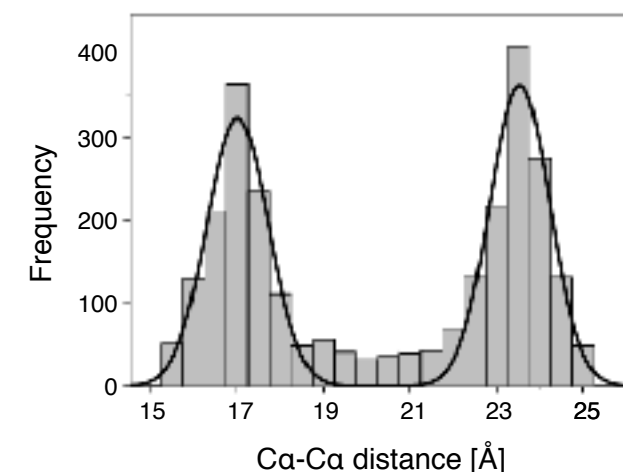
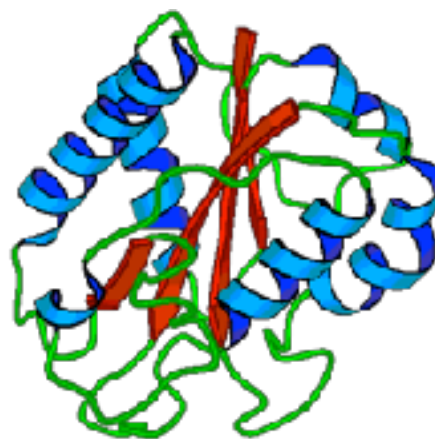
3D GKITYERGFQGHYESDC-NLQP...

SEQ GKITYERG---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/>

Run FoXS on the model

Run FoXS on the model

- Precalculated MODELLER model is available for those that don't have MODELLER

Run FoXS on the model

- Precalculated MODELLER model is available for those that don't have MODELLER
- FoXS is run in the same way as before, just on the model:
`$ foxs 3KFO-fill.B99990005.pdb 23922_merge.dat`

Run FoXS on the model

- Precalculated MODELLER model is available for those that don't have MODELLER
- FoXS is run in the same way as before, just on the model:

```
$ foxs 3KFO-fill.B99990005.pdb 23922_merge.dat
```
- Output will end with something like

```
3KFO-fill.B99990005.pdb 23922_merge.dat Chi  
= 1.14507 c1 = 1.02835 c2 = 0.93184 default  
chi = 6.36924
```


Run FoXS on the model

- Precalculated MODELLER model is available for those that don't have MODELLER
- FoXS is run in the same way as before, just on the model:

```
$ foxs 3KFO-fill.B99990005.pdb 23922_merge.dat
```
- Output will end with something like

```
3KFO-fill.B99990005.pdb 23922_merge.dat Chi  
= 1.14507 c1 = 1.02835 c2 = 0.93184 default  
chi = 6.36924
```
- i.e. quality of fit (χ) is 1.1, much improved

Visualize outputs

Visualize outputs

- Also generates *.dat files for plotting

Visualize outputs

- Also generates *.dat files for plotting
- If you have gnuplot, add -g option to get gnuplot input files (*.plt) too:

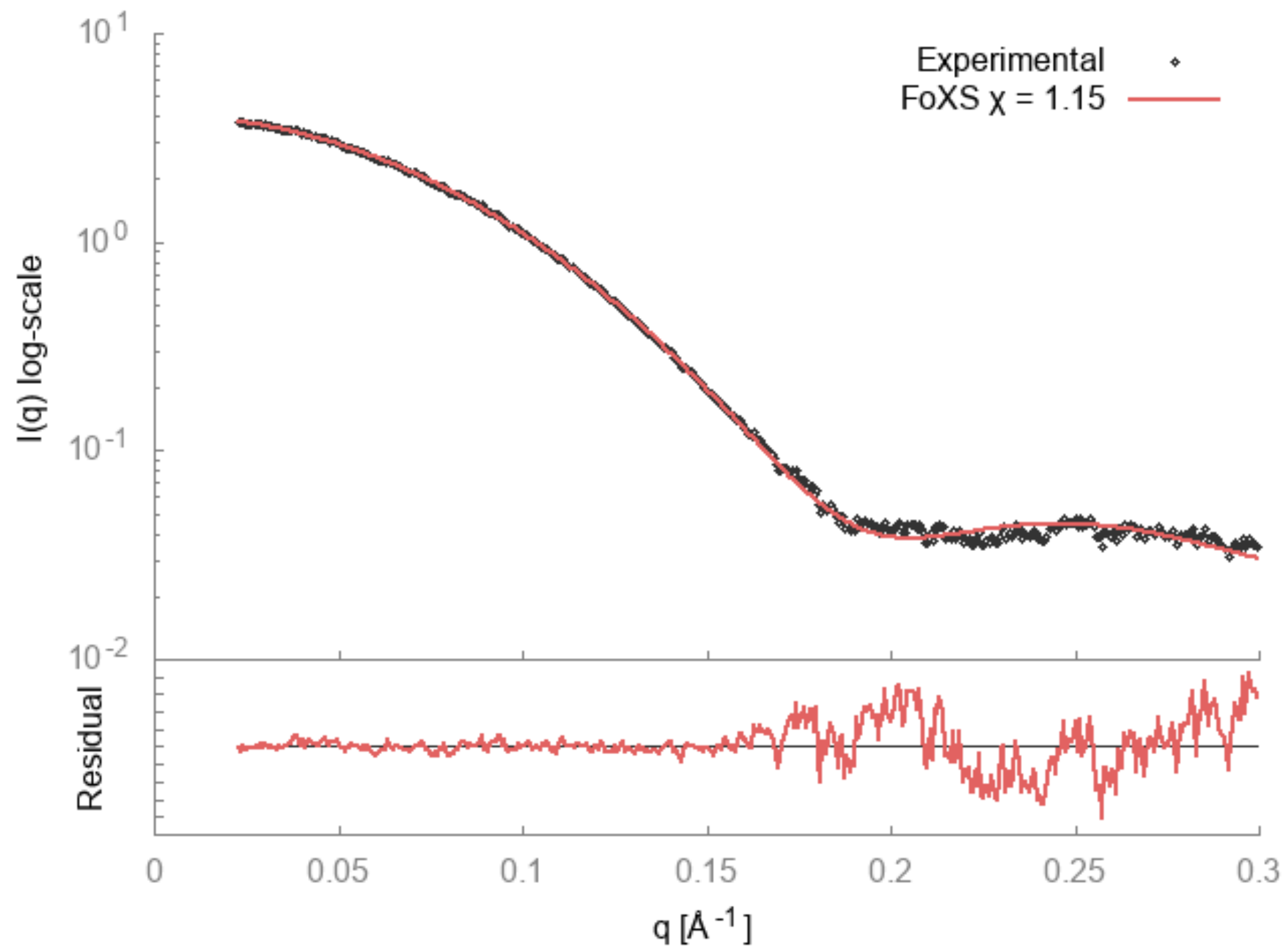
```
$ foxs -g 3KFO-fill.B999900005.pdb 23922_merge.dat  
$ gnuplot 3KFO-fill.B999900005_23922_merge.plt
```


Visualize outputs

- Also generates *.dat files for plotting
- If you have gnuplot, add -g option to get gnuplot input files (*.plt) too:



```
$ foxs -g 3KFO-fill.B999900005.pdb 23922_merge.dat  
$ gnuplot 3KFO-fill.B999900005_23922_merge.plt
```
- Look at
3KFO-fill.B999900005_23922_merge.png in an image viewer

gnuplot output



FoXS web service

- Alternatively, use the FoXS web service: <https://salilab.org/foxs/>
- Takes same inputs, makes plots etc.



Fast SAXS Profile Computation with Debye Formula

[About FOXS](#) • [Web Server](#) • [Help](#) • [FAQ](#) • [Download](#) • [Sali Lab](#) • [IMP](#) • [Links](#)

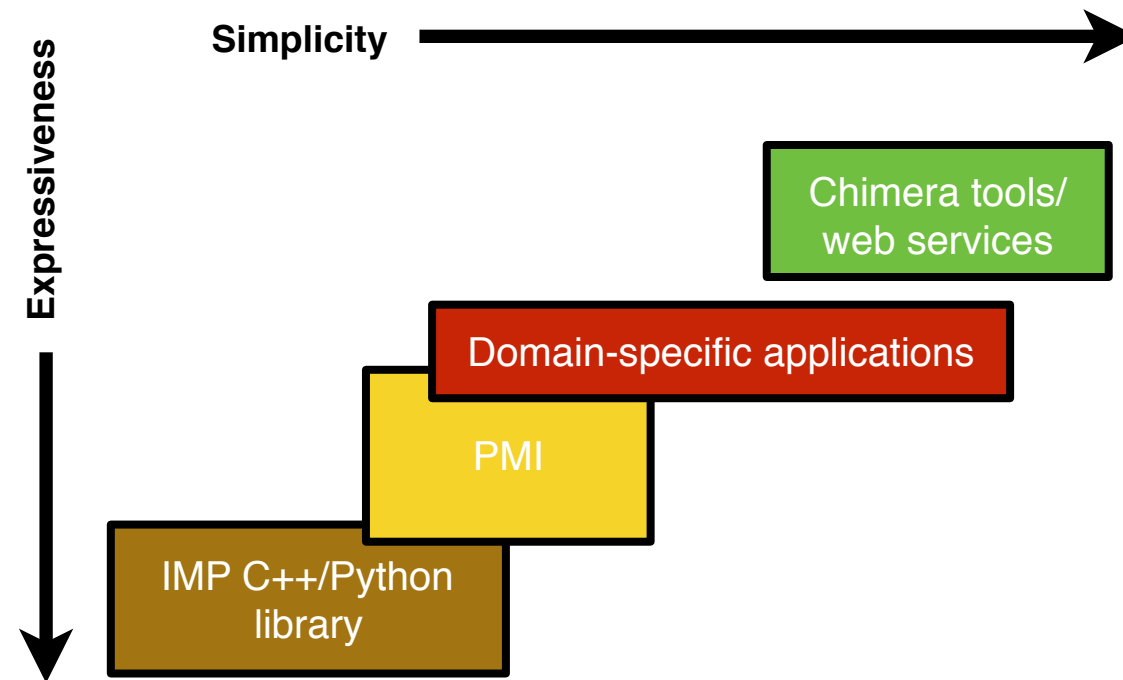
Type PDB code of input molecule or upload files in PDB format (zip file with several PDBs can be uploaded):

Input molecule:	<input type="text"/>	(PDB:chainId e.g. 6lyz:A) or upload file:	<input type="button" value="Browse..."/> No file selected.
Experimental profile:	<input type="button" value="Browse..."/> No file selected.	(optional) sample input	
e-mail address:	<input type="text"/>	(optional, the results are sent to this address)	

[Advanced Options](#)

NEW! [AntiFoXS](#) [Now with conformational sampling and multi-state modeling, try here](#)

If you use FoXS, please cite:
Schneidman-Duhovny D, Hammel M, Tainer JA, and Sali A. Accurate SAXS profile computation and its assessment by contrast variation experiments. *Biophysical Journal* 2013. 105 (4), 962-974
Schneidman-Duhovny D, Hammel M, Tainer JA, and Sali A. FoXS, FoXSDock and MultiFoXS: Single-state and multi-state structural modeling of proteins and their complexes based on SAXS profiles *NAR* 2016 [[FREE Full Text](#)]
Contact: dina@salilab.org





PMI

- Just another IMP module (`IMP.pmi`)
- A meta language for modeling
- We still write Python scripts, but...
 - Refer to biological units rather than individual particles
 - Many protocols (e.g. replica exchange) already packaged up nicely for us
 - 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
- One of the IMP tutorials uses PMI to model the stalk of the RNA Polymerase II complex

Software installation for PMI

- We need installed
 - **numpy** and **scipy** for matrix and linear algebra
 - **scikit-learn** for k-means clustering
 - **matplotlib** for plotting results
 - **UCSF Chimera** for visualization of results
 - **IMP** itself
 - **git** is very useful for tracking our work (but not essential)
- Again, easiest way (for everything except Chimera) is to install Anaconda Python, then run from a command prompt/terminal:

```
$ conda config --add channels salilab
```

```
$ conda install imp git numpy scipy scikit-learn matplotlib
```


PMI tutorial data

PMI tutorial data

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

PMI tutorial data

- Get the tutorial files from GitHub:
https://github.com/salilab/imp_tutorial/
- Best way is to clone with git:

```
$ git clone https://github.com/salilab/imp_tutorial.git
```


PMI tutorial data

- Get the tutorial files from GitHub:
https://github.com/salilab/imp_tutorial/
- Best way is to clone with git:

```
$ git clone https://github.com/salilab/imp_tutorial.git
```
- If you don't have a git client, get the zip file instead from the “clone or download” link



PMI I vs. PMI2



PMI I vs. PMI2

- The PMI folks replaced the PMI Representation class (“PMI1”) with a new (“PMI2”) System class

PMI I vs. PMI2

- The PMI folks replaced the PMI Representation class (“PMI1”) with a new (“PMI2”) System class
- Much cleaner and faster

PMII vs. PMI2

- The PMI folks replaced the PMI Representation class (“PMII”) with a new (“PMI2”) System class
- Much cleaner and faster
- But most IMP docs still refer to the PMII style approach

PMII vs. PMI2

- The PMI folks replaced the PMI Representation class (“PMII”) with a new (“PMI2”) System class
 - Much cleaner and faster
 - But most IMP docs still refer to the PMII style approach
- PMI2 tutorials:

PMII vs. PMI2

- The PMI folks replaced the PMI Representation class (“PMII”) with a new (“PMI2”) System class
 - Much cleaner and faster
 - But most IMP docs still refer to the PMII style approach
- PMI2 tutorials:
 - https://github.com/salilab/actin_tutorial/

PMII vs. PMI2

- The PMI folks replaced the PMI Representation class (“PMII”) with a new (“PMI2”) System class
 - Much cleaner and faster
 - But most IMP docs still refer to the PMII style approach
- PMI2 tutorials:
 - https://github.com/salilab/actin_tutorial/
 - https://github.com/salilab/imp_tutorial/tree/develop

Why git?

<https://git-scm.com/book>

Why git?

<https://git-scm.com/book>

- git tracks who changed what and when (like an electronic lab notebook)

Why git?

<https://git-scm.com/book>

- git tracks who changed what and when (like an electronic lab notebook)
 - often hard to keep modeling protocols organized

Why git?

<https://git-scm.com/book>

- git tracks who changed what and when (like an electronic lab notebook)
 - often hard to keep modeling protocols organized
 - don't have to use git for this, but *vitaly important* to use some kind of change tracking software (e.g. SVN, CVS, hg, etc.)

Why git?

<https://git-scm.com/book>

- git tracks who changed what and when (like an electronic lab notebook)
 - often hard to keep modeling protocols organized
 - don't have to use git for this, but *vitaly important* to use some kind of change tracking software (e.g. SVN, CVS, hg, etc.)
- git integrates nicely with GitHub which provides a web front end

Why git?

<https://git-scm.com/book>

- git tracks who changed what and when (like an electronic lab notebook)
 - often hard to keep modeling protocols organized
 - don't have to use git for this, but *vitally important* to use some kind of change tracking software (e.g. SVN, CVS, hg, etc.)
- git integrates nicely with GitHub which provides a web front end
 - Simplifies collaboration on software and protocols

Why git?

<https://git-scm.com/book>

- git tracks who changed what and when (like an electronic lab notebook)
 - often hard to keep modeling protocols organized
 - don't have to use git for this, but *vitaly important* to use some kind of change tracking software (e.g. SVN, CVS, hg, etc.)
- git integrates nicely with GitHub which provides a web front end
 - Simplifies collaboration on software and protocols
- Our ultimate goal with any modeling is to make it public, reproducible (see other published systems, also managed by git: <https://integrativemodeling.org/systems/>)

Why git?

<https://git-scm.com/book>

- git tracks who changed what and when (like an electronic lab notebook)
 - often hard to keep modeling protocols organized
 - don't have to use git for this, but *vitaly important* to use some kind of change tracking software (e.g. SVN, CVS, hg, etc.)
- git integrates nicely with GitHub which provides a web front end
 - Simplifies collaboration on software and protocols
- Our ultimate goal with any modeling is to make it public, reproducible (see other published systems, also managed by git: <https://integrativemodeling.org/systems/>)
- Helpful git commands: `git log`, `git show`, `git pull`, `git status`, `git diff`, `git commit`, `git push`

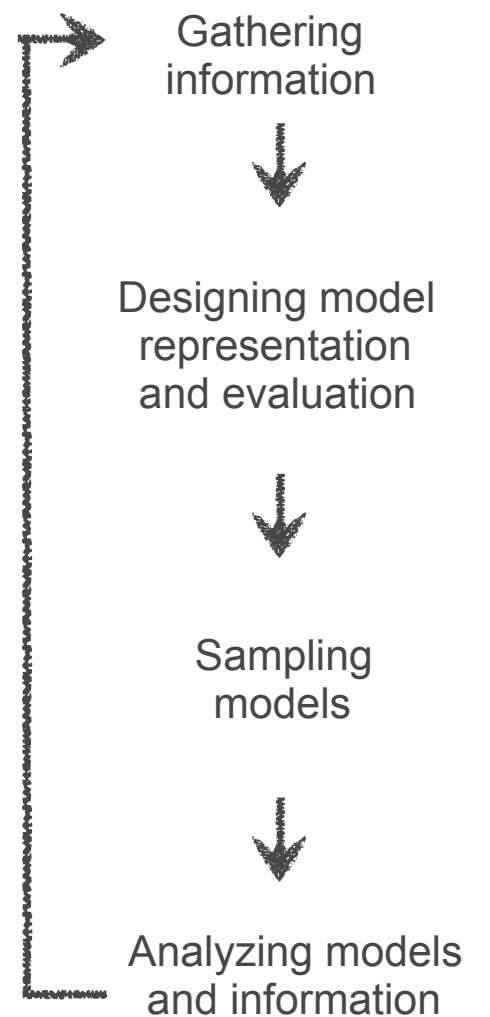
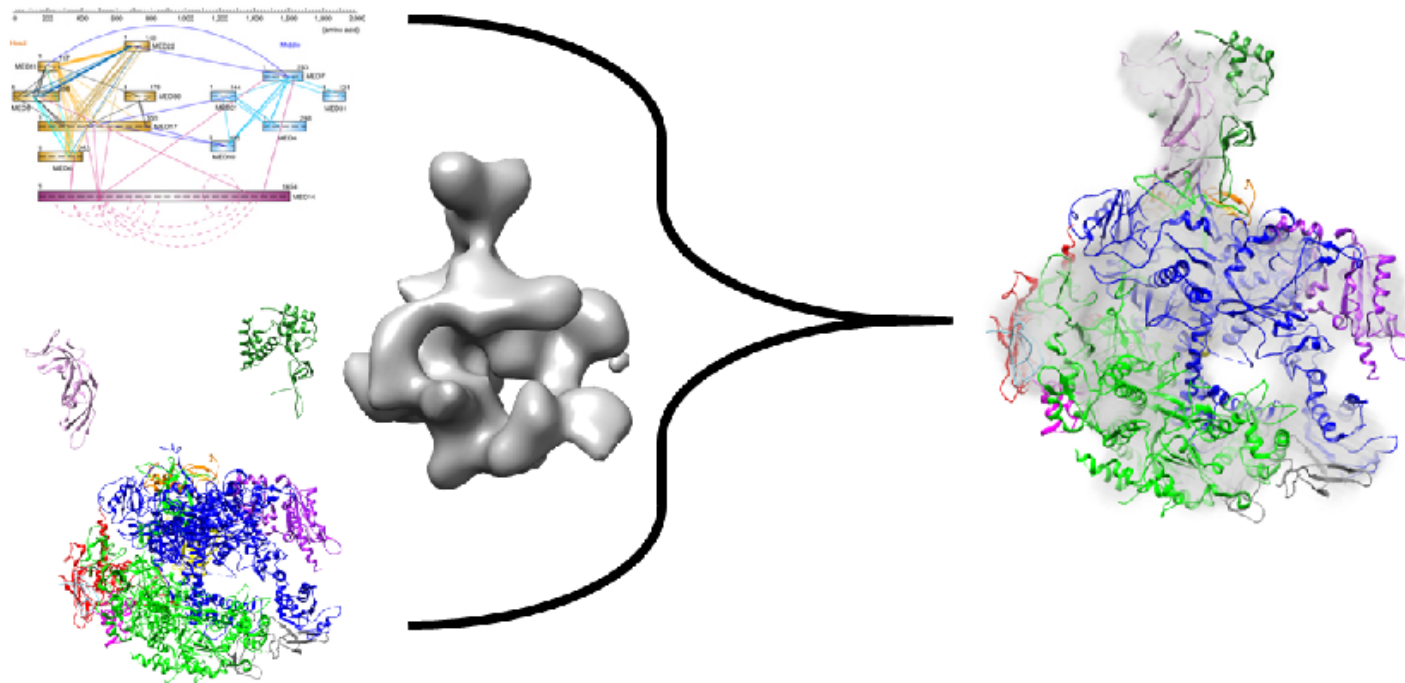
Integrative structure modeling of RNA Polymerase II stalk

- 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 already know the structure of the remaining 10-subunit complex
- This example utilizes:
 - chemical cross-linking with mass spectrometry (CX-MS),
 - negative-stain electron microscopy (EM),
 - X-ray crystallography data

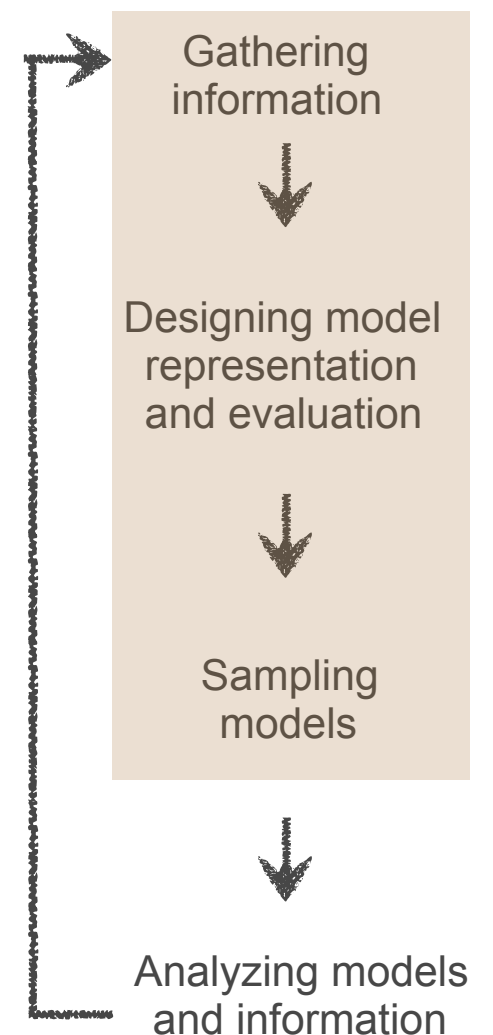


Main modeling script

- Let's get started by getting the main modeling script running while we look at what it's doing
- Do this by running in a terminal/command prompt:

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

- “Real” modeling will take hours, so we're running in 'test' mode which generates only 100 frames (rather than 20,000)
- The script covers the first 3 steps of integrative modeling



Expected script output

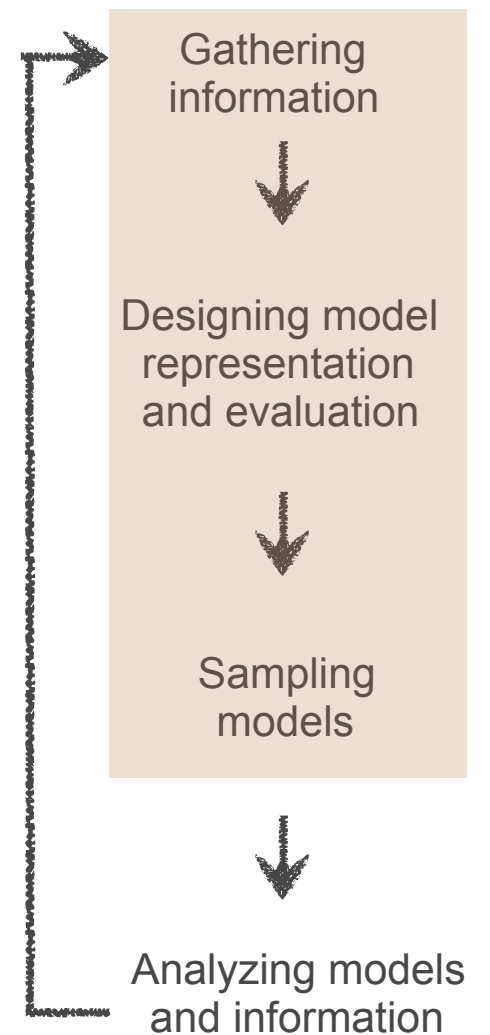
```
$ python modeling.py --test
autobuild_model: constructing Rpb1 from pdb ../data/./1WCM_map_fitted.pdb and chain A
autobuild_model: constructing fragment (1, 1) as a bead
autobuild_model: constructing fragment (2, 186) from pdb
autobuild_model: constructing fragment (187, 194) as a bead
autobuild_model: 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

...

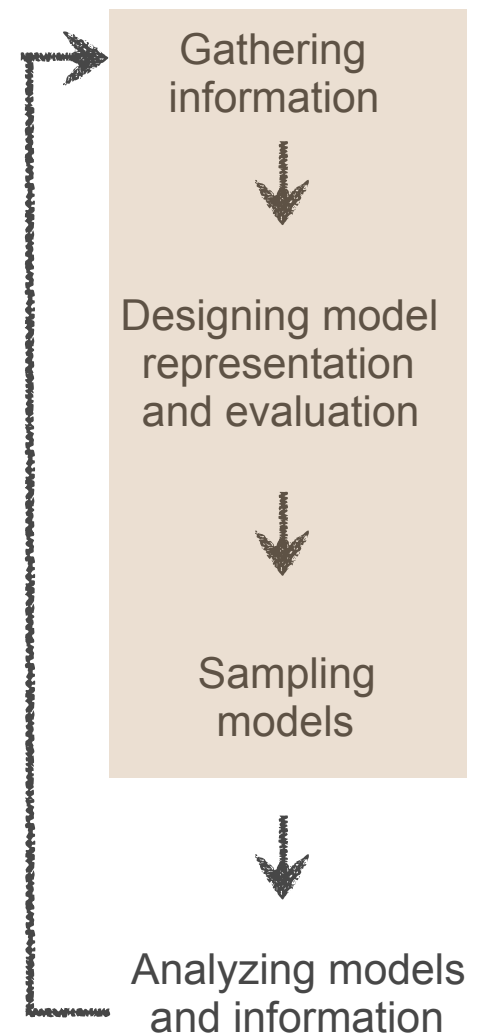
--- frame 1 score 4814598.44759
--- writing coordinates
--- frame 2 score 3527090.92513
--- writing coordinates
--- frame 3 score 2662180.99705
--- writing coordinates
--- frame 4 score 2021182.74211
--- writing coordinates
--- frame 5 score 1459614.23926
```



Common errors

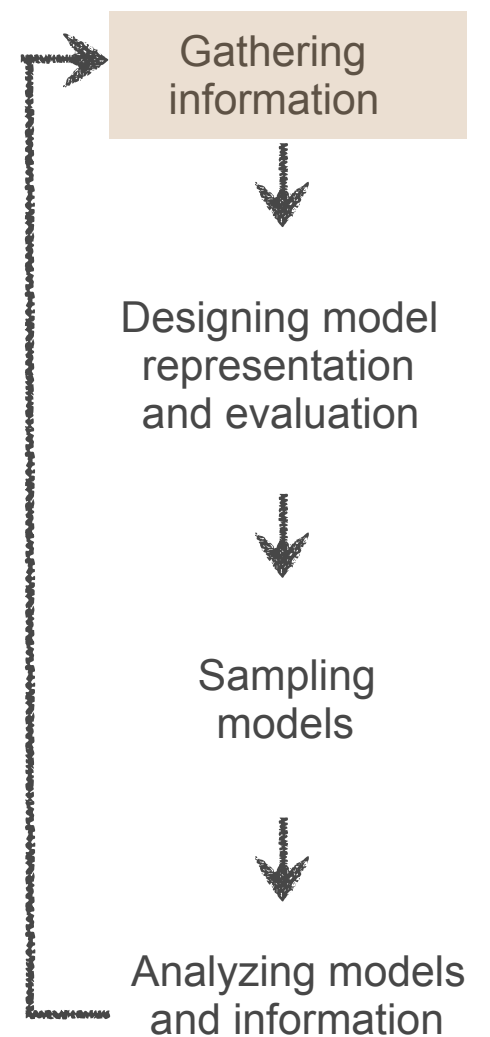
- If you see

 NameError: name 'inf' is not defined
- ... try running the script again
(sometimes IMP's initial random model results
in a very bad fit to the EM map, and the
system cannot recover)



Data for yeast RNA Polymerase II

- The `rnapolii/data` folder (within the `imp_tutorial` folder) contains, amongst other data:
 - Sequence information (FASTA files for each subunit)
 - Electron density maps (`.mrc`, `.txt` files)
 - Structure from X-ray crystallography (PDB file)
 - Chemical cross-linking datasets (two data sets, one from Al Burlingame's lab, and another from Juri Rappsilber's lab)
- Most IMP files, including these, can be viewed in a text editor, Chimera/VMD/other viewer, or from the GitHub web interface
- We'll look at each data source in turn



FASTA file

1WCM.fasta.txt is a simple text file containing sequences in FASTA format:

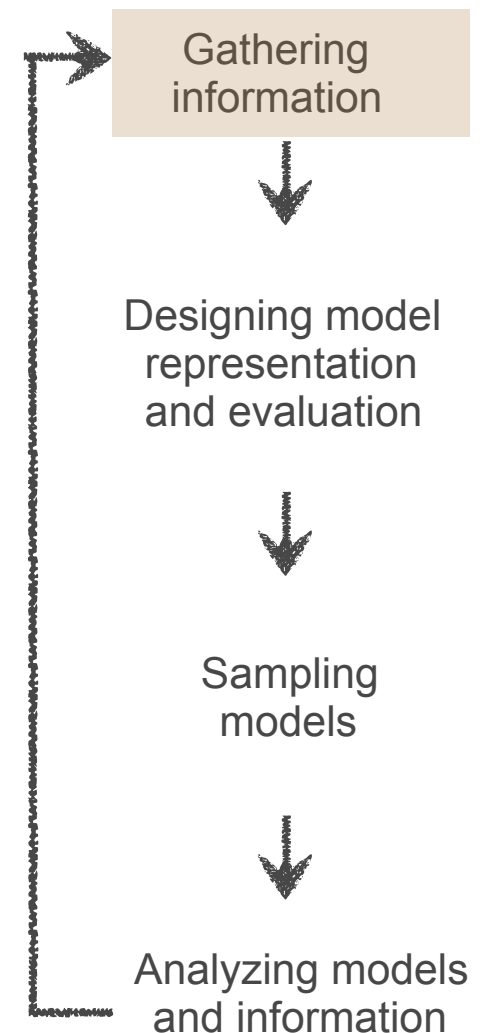
>1WCM:A

MVGQQYSSAPLRTVKEVQFGLFSPEEVRAISVAKIRFPETMDETQTRAKIGG
LNDPRLGSIDRNLKCQTCQEGMNECPGHFGHIDLAKPVFHVGFIAKIKKVCE
CVCMHCGKLLLDEHNELMRQALAIKDSKKRFAAIWTLCKTKMVCETDVPSED
...

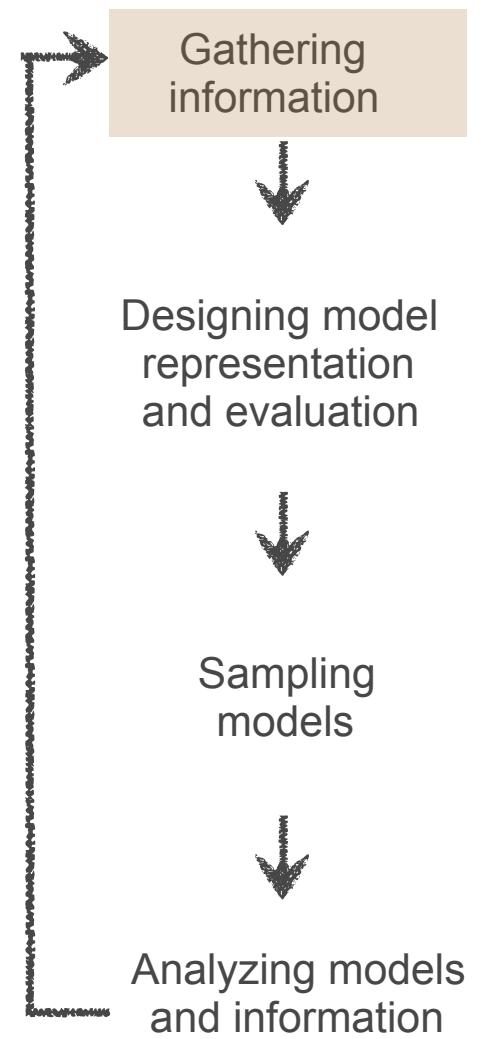
>1WCM:B

MSDLANSEKYYDEDPYGFEDESAPITAEDSWAVISAFFREKGLVSQQLDSEFN
QFVDYTLQDIICEDSTLILEQLAQHTTESDNISRKYEISFGKIYVTKPMVNE
SDGVTHALYPQEARLRNLTYSSGLFVDVKKRTYEAIDVPGRELKYELIAEES
...

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

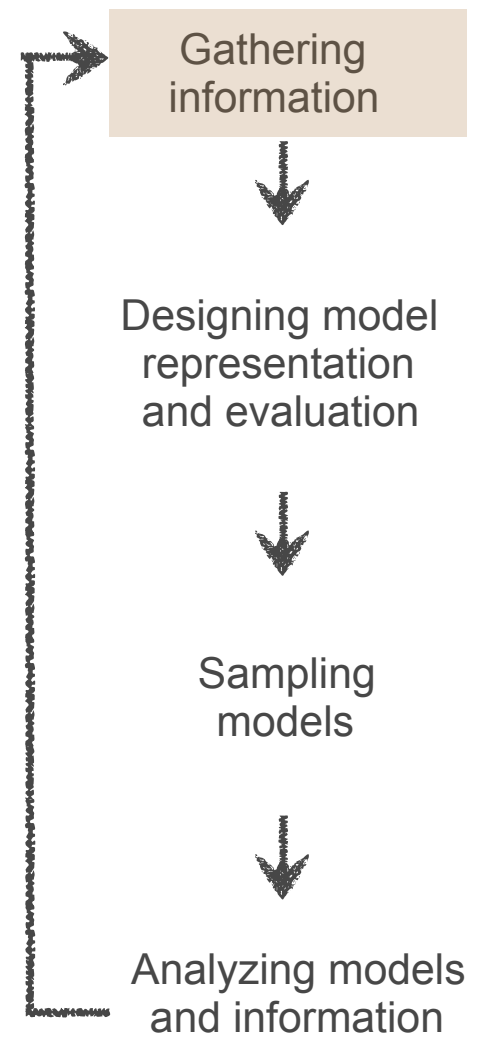
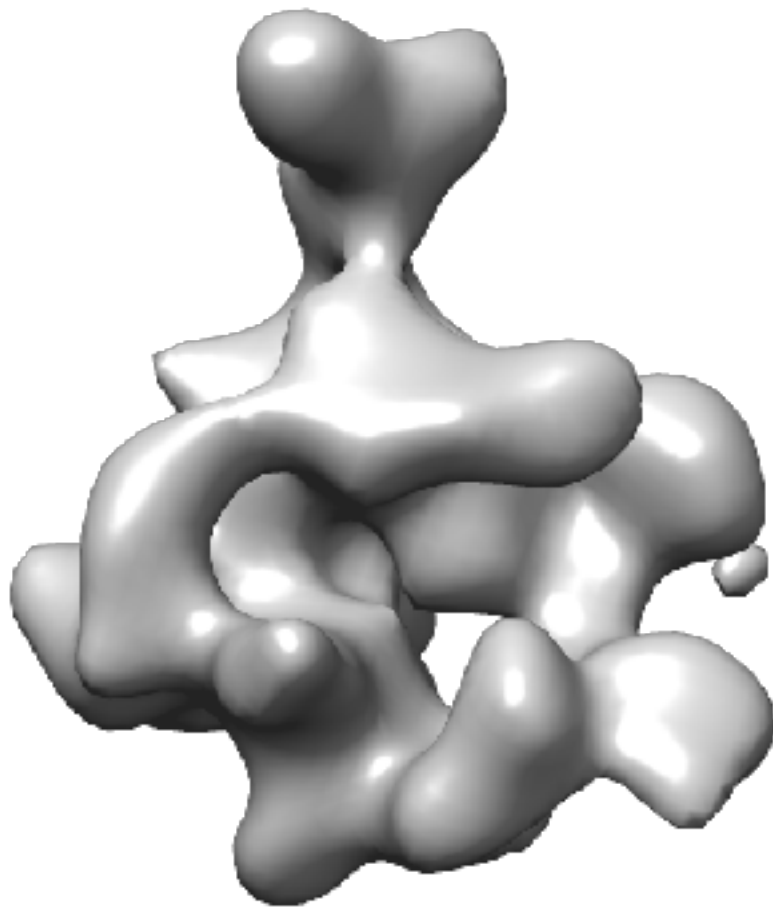


Electron density map



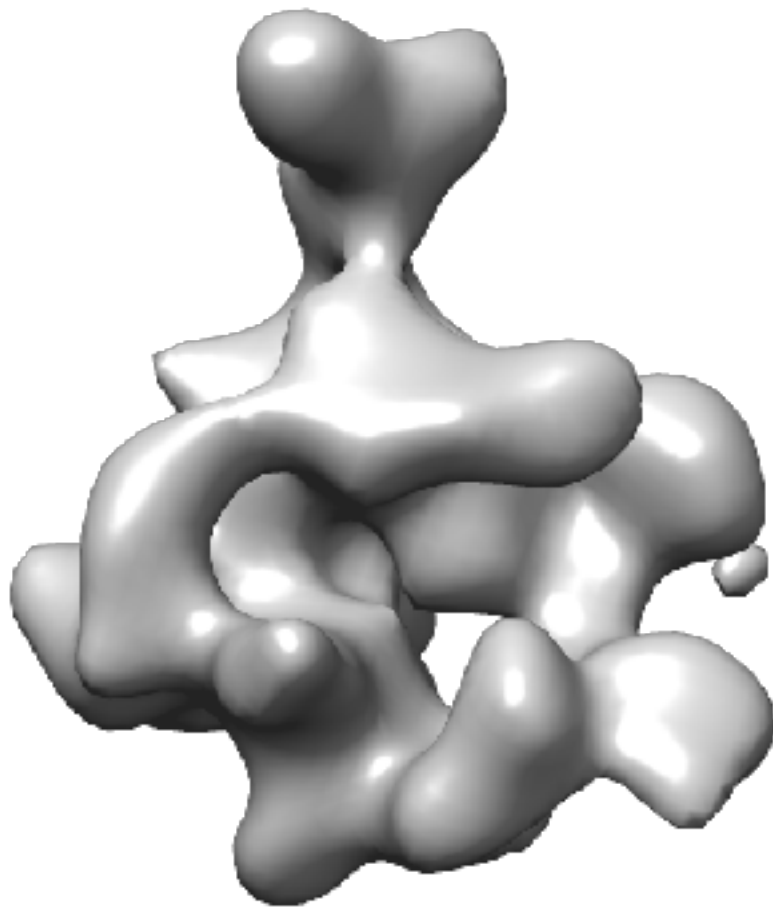
Electron density map

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

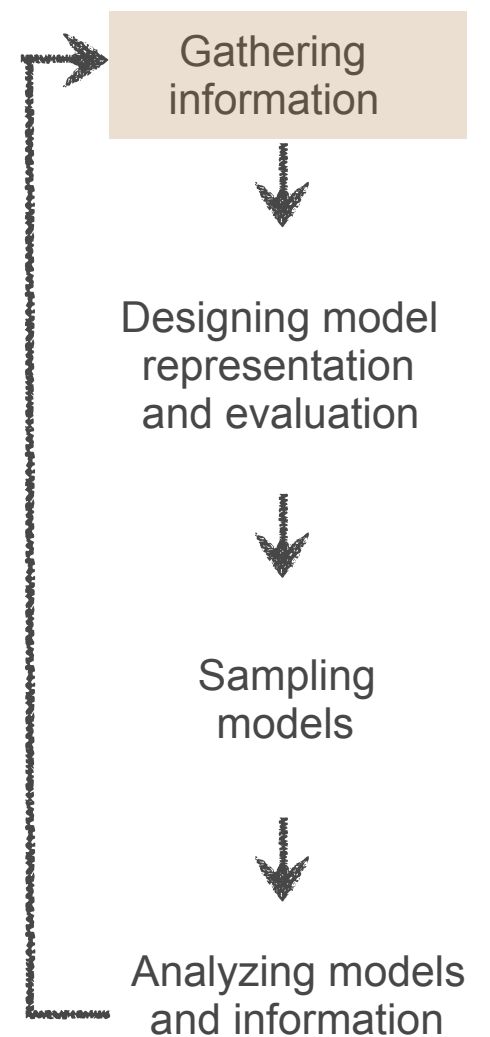


Electron density map

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

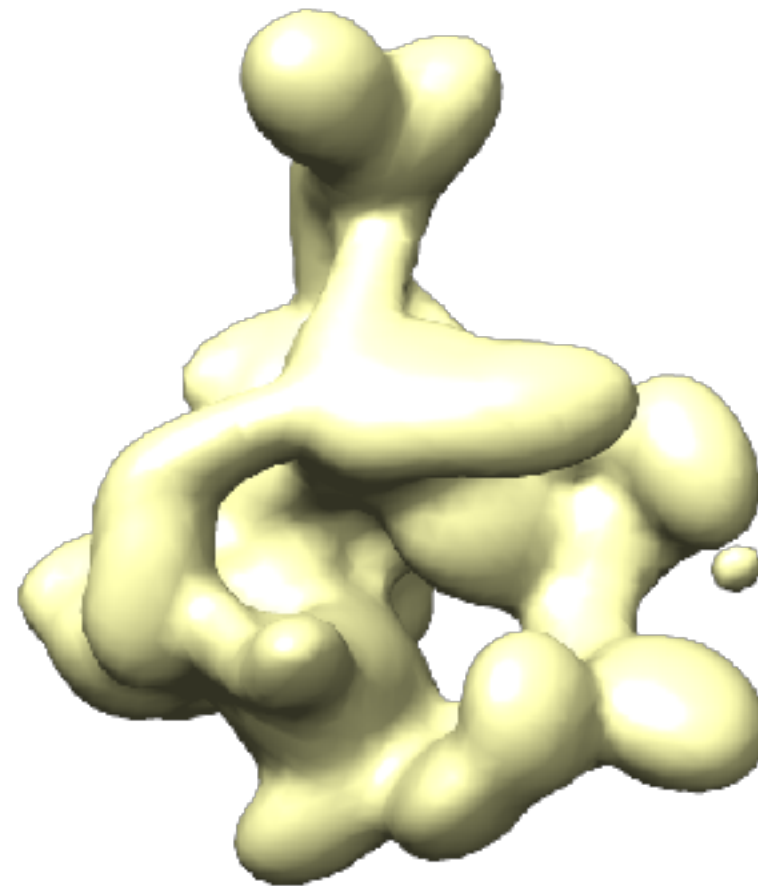
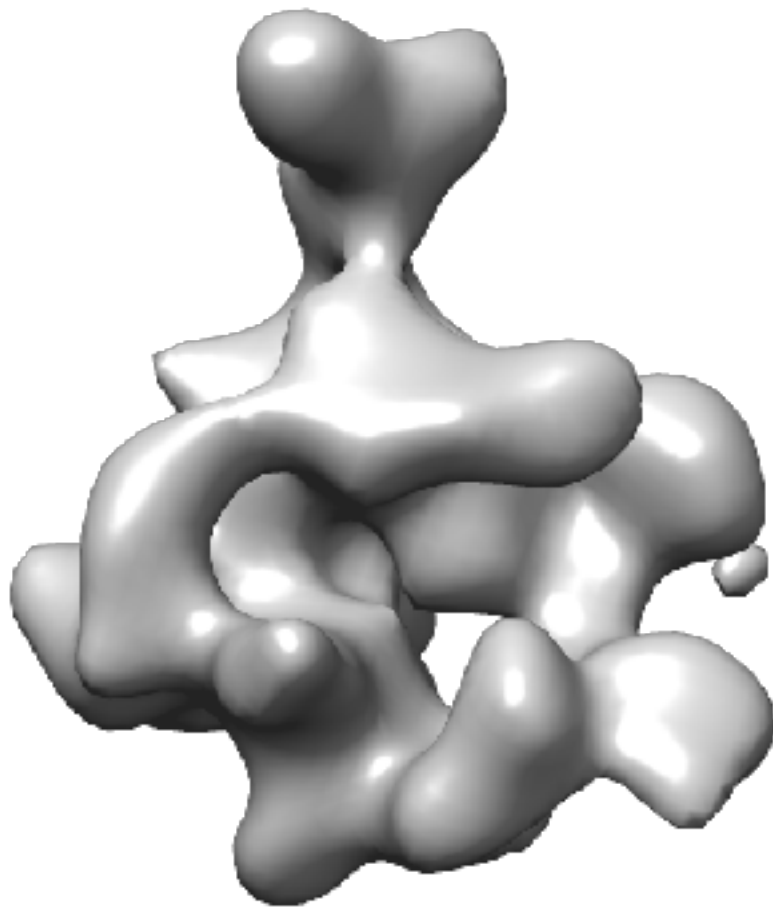


Gaussian mixture models (GMMs) are used to greatly speed up scoring by approximating the electron density of individual subunits and experimental EM maps as a sum of 3D Gaussians. The weight, center, and covariance matrix of each Gaussian used to approximate the original EM density can be seen in *emd_1883.map.mrc.gmm.50.txt*

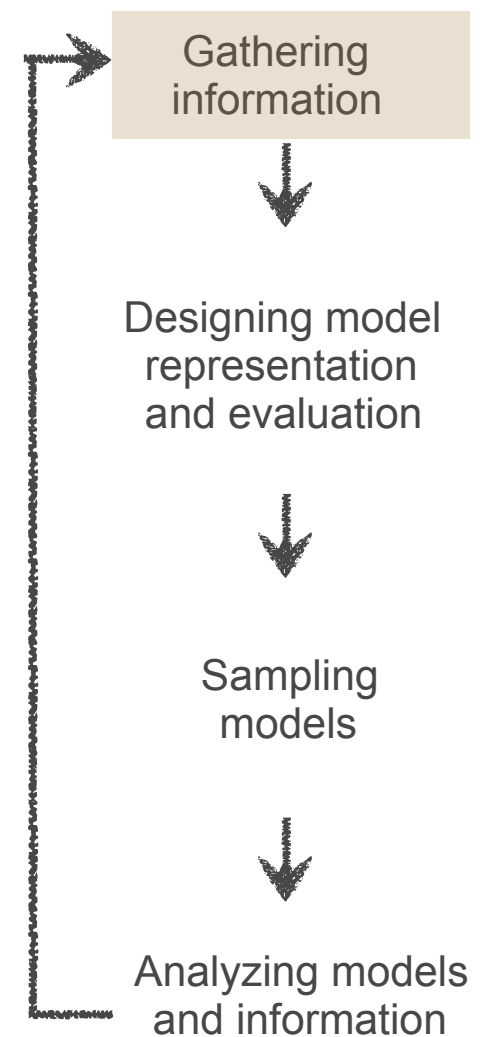


Electron density map

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*

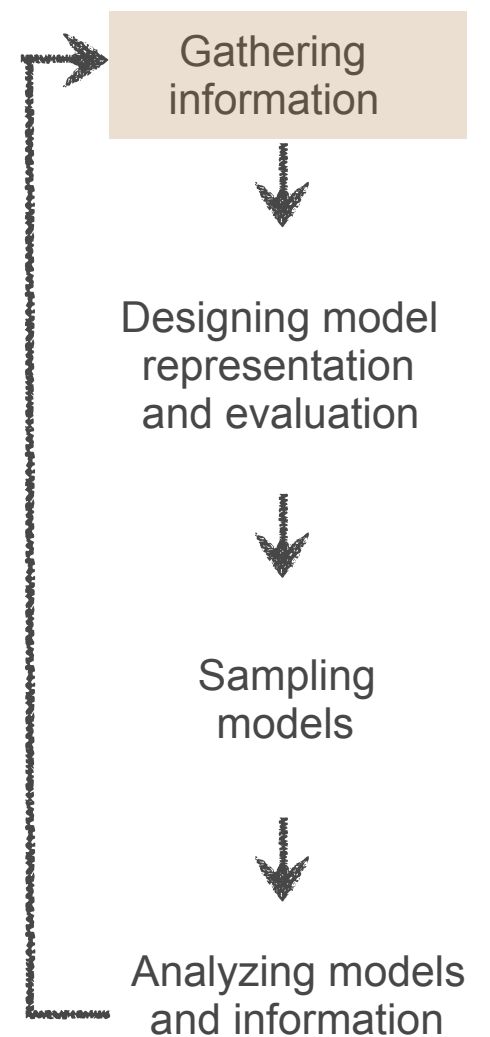


Chemical cross-links

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 linked residues:

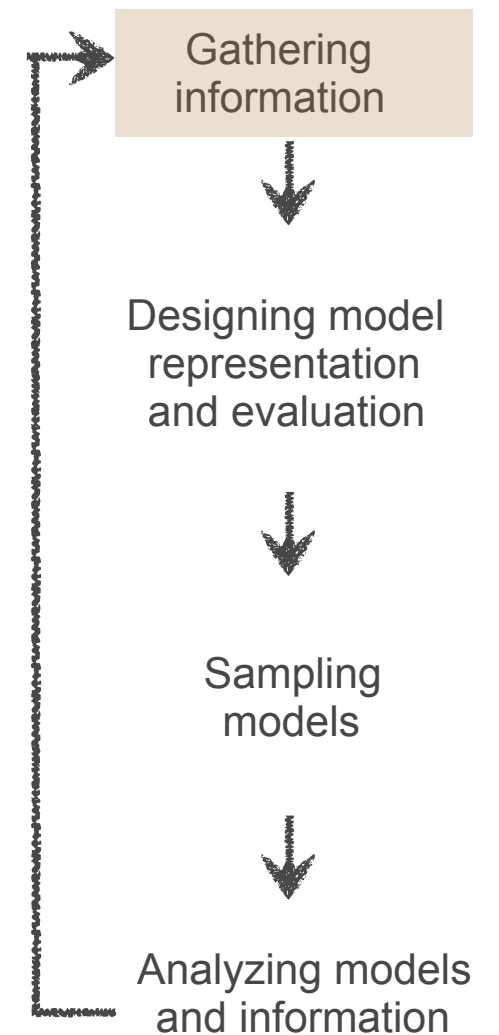
```
prot1,res1,prot2,res2  
Rpb1,34,Rpb1,49  
Rpb1,101,Rpb1,143  
Rpb1,101,Rpb1,176
```

(The length of the DSS/BS3 cross-linker reagent, 21Å, is not in this file; it'll be specified in the modeling script.)



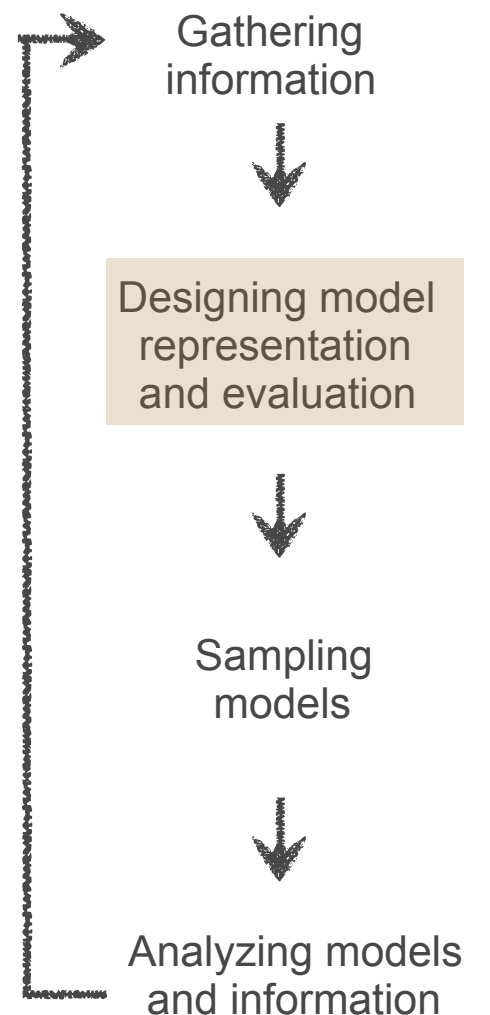
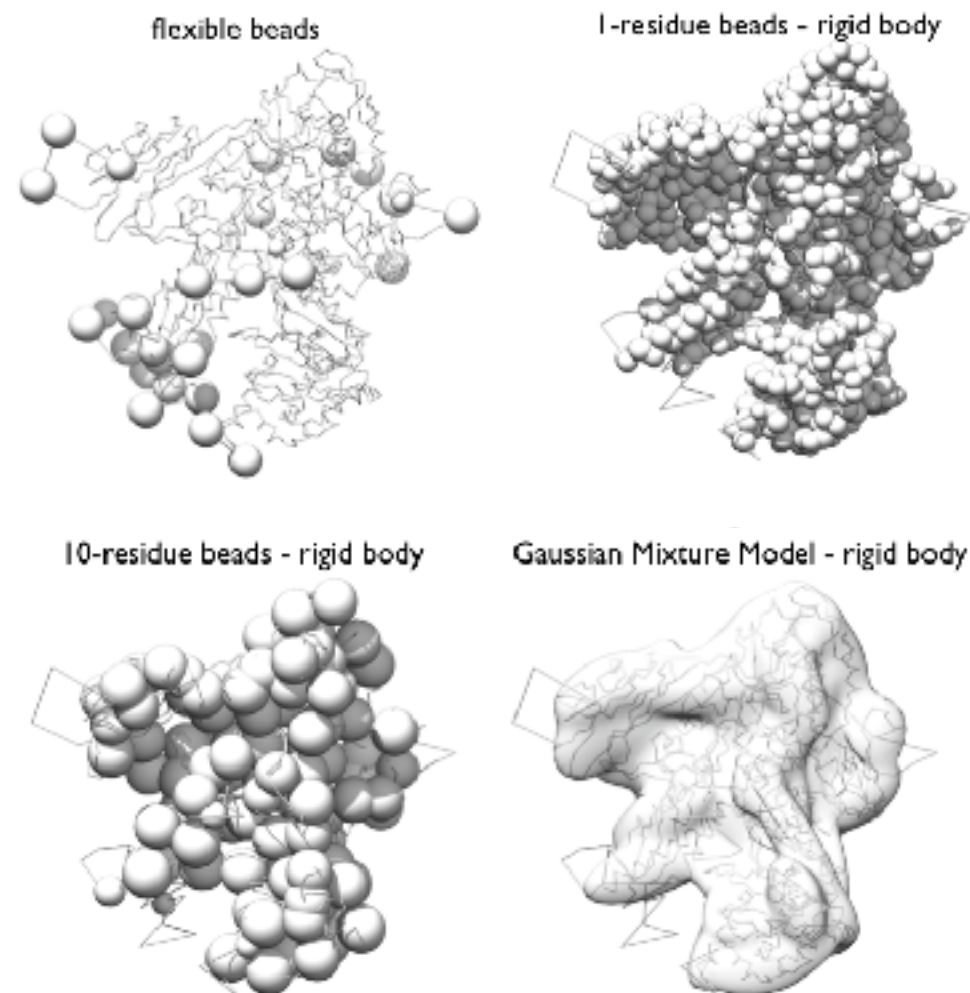
X-ray structures

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



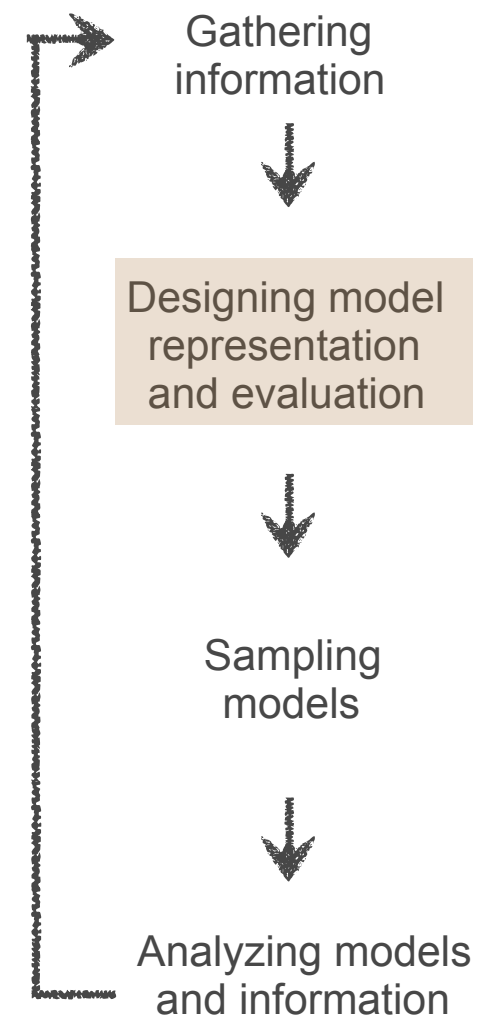
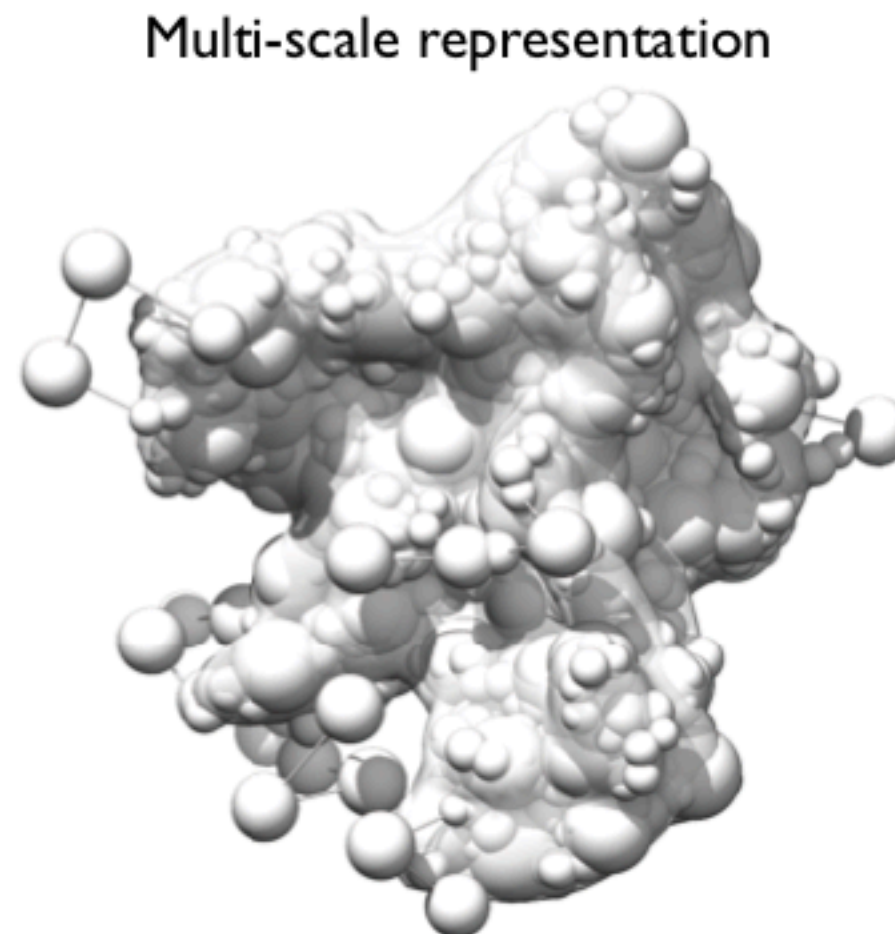
Model representation in IMP

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

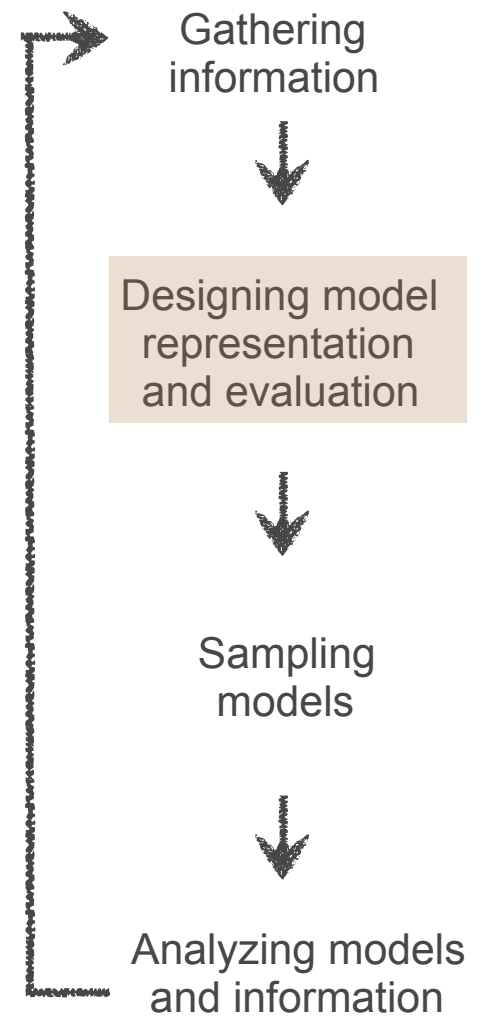


Model representation in IMP

- Note that our representation is **multi-scale**
- i.e. we use both low resolution **and** high resolution bead and Gaussian representations of the model **simultaneously** (“resolution 1”: 1 residue per spherical bead, and “resolution 20”: 20 residues per bead)
- Restraints are applied to the most appropriate representation



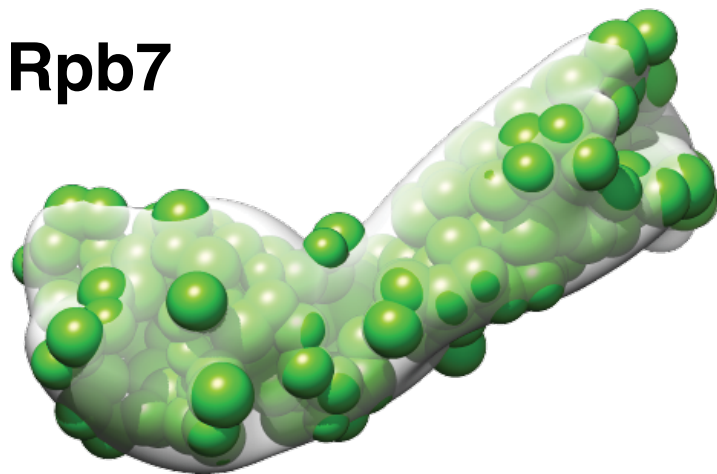
Handling of missing structure



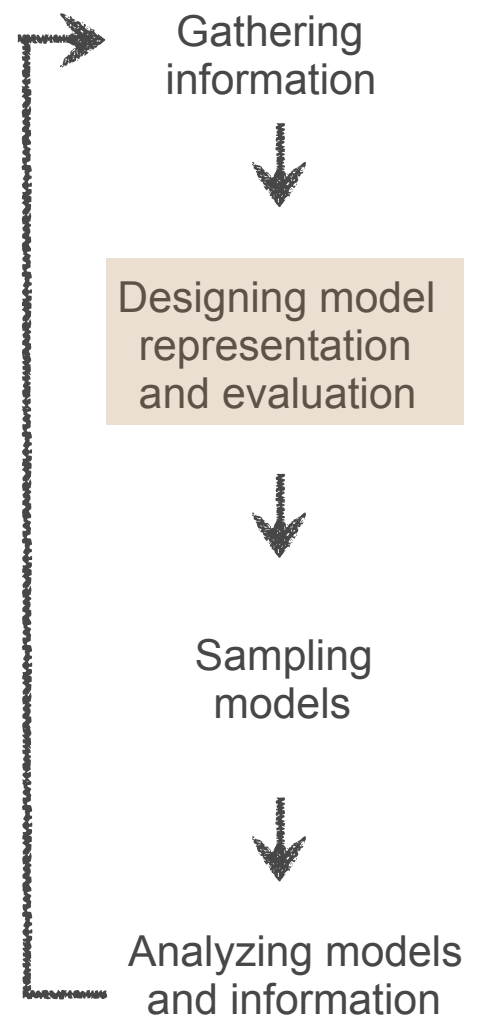
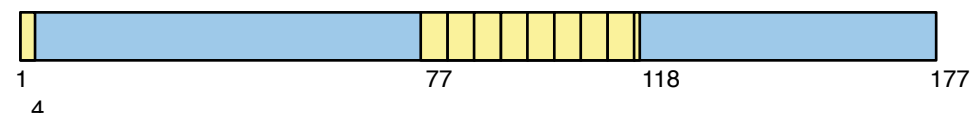
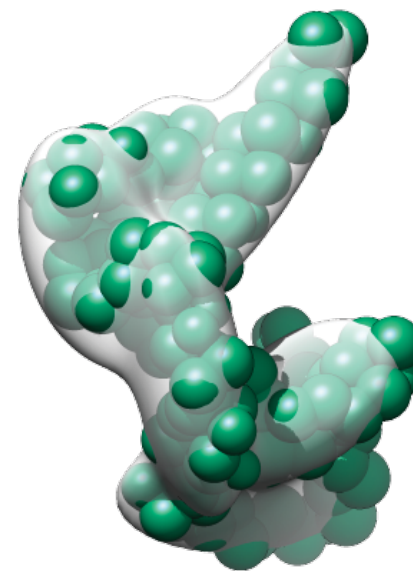
Handling of missing structure

- Even though we have X-ray structures, not all residues were resolved (yellow regions)

Rpb7



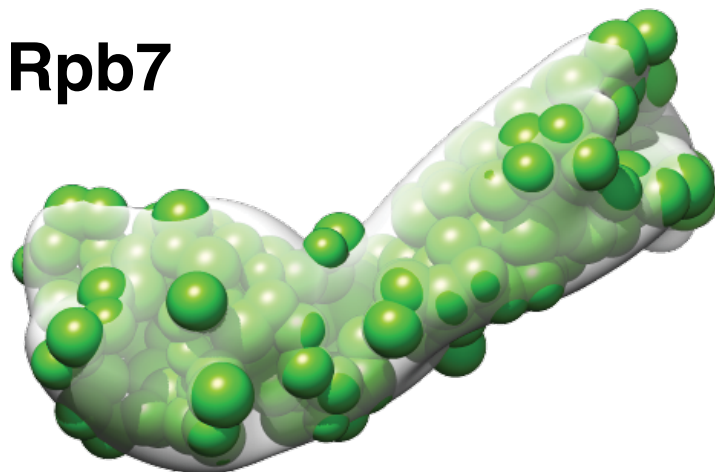
Rpb4



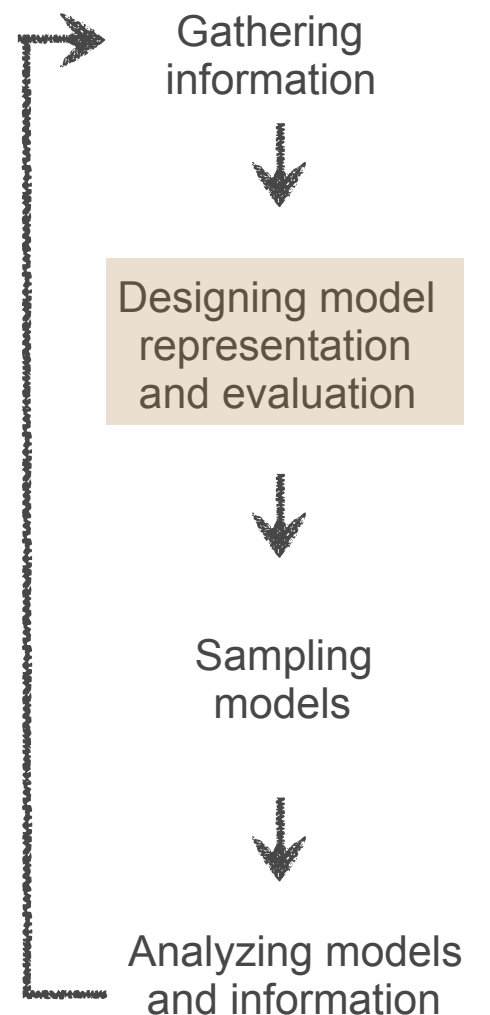
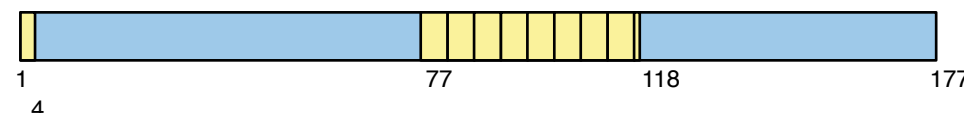
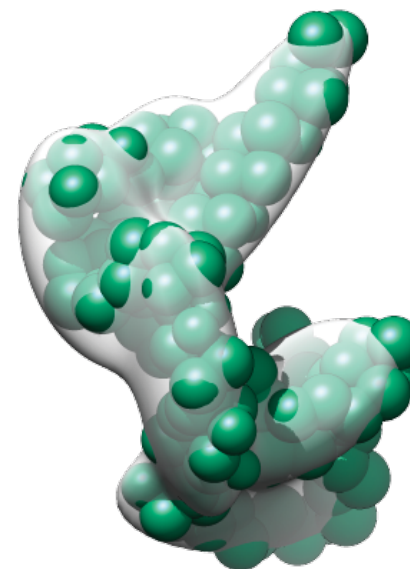
Handling of missing structure

- 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

Rpb7



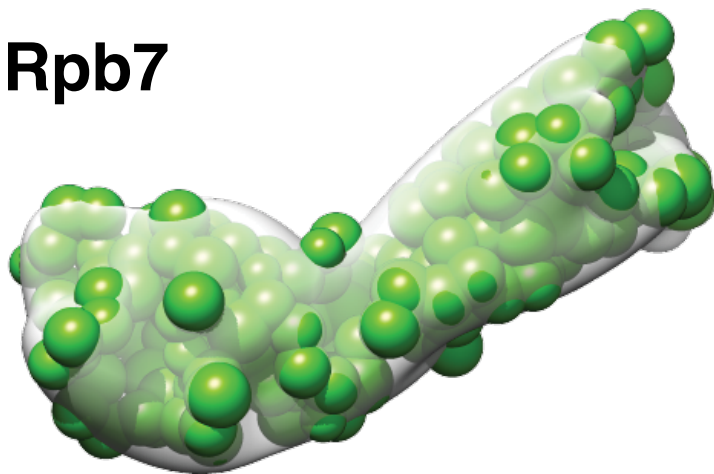
Rpb4



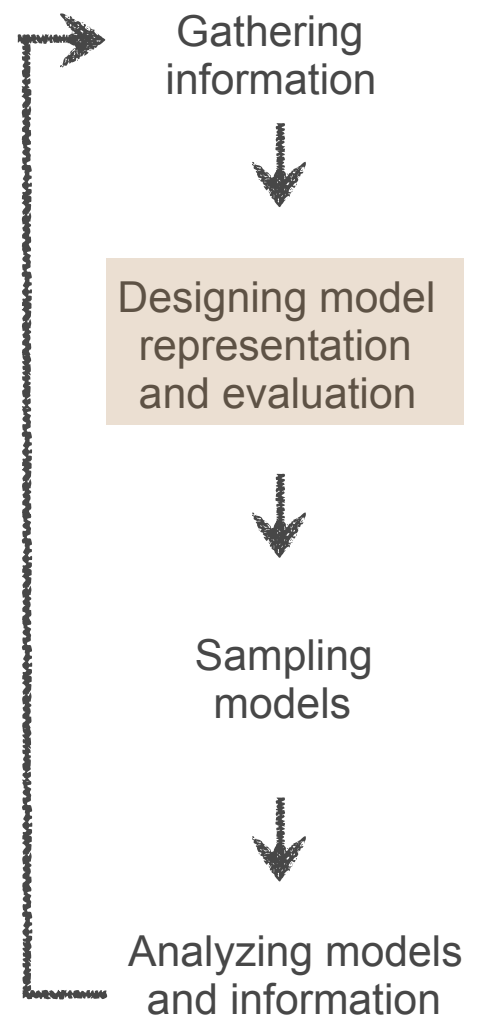
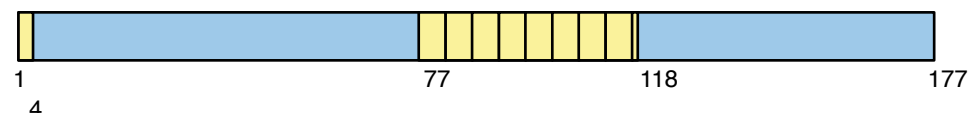
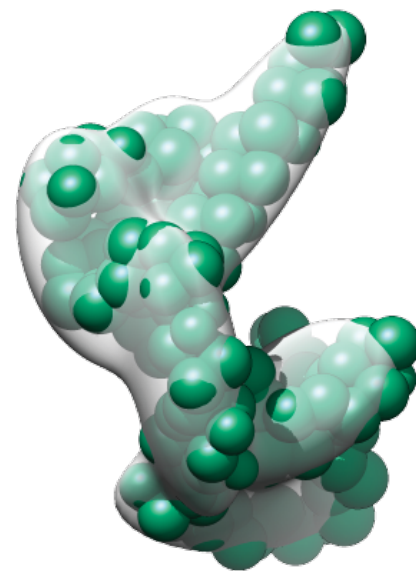
Handling of missing structure

- 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 (up to 20 residues per bead) instead here

Rpb7

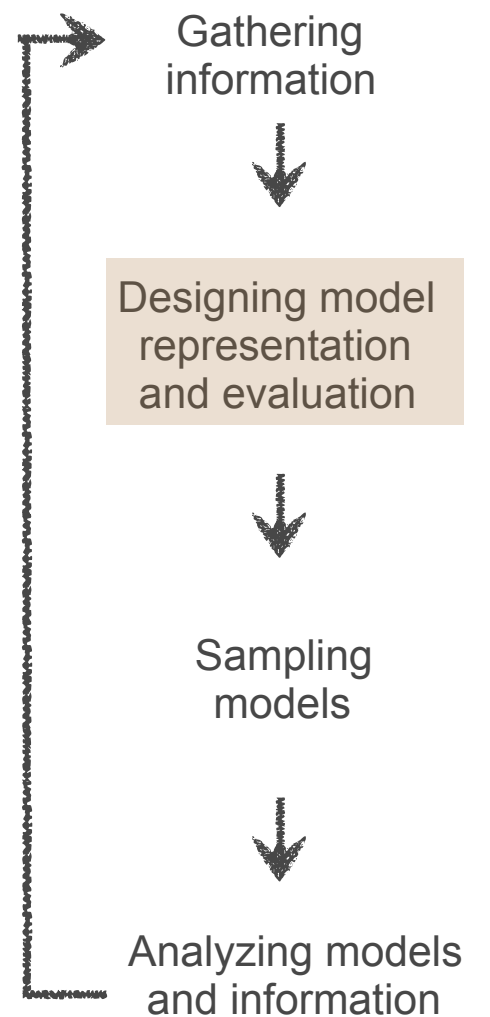


Rpb4

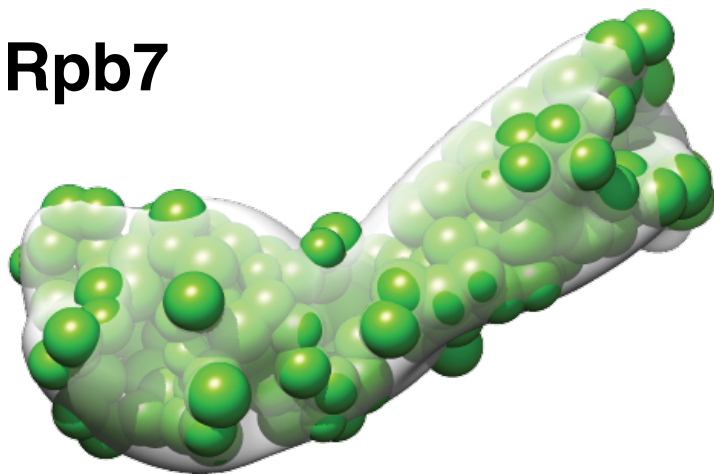


Handling of missing structure

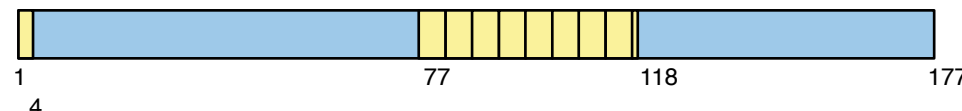
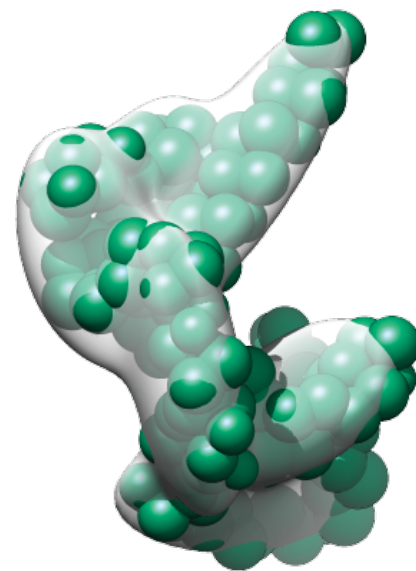
- Even though we have X-ray structures, not all residues were resolved (yellow regions)
- Would be over-interpretation of the data to try to represent this at high resolution
- Use low resolution beads (up to 20 residues per bead) instead here
- Treat resolved regions as rigid bodies, allow unresolved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



Rpb7



Rpb4

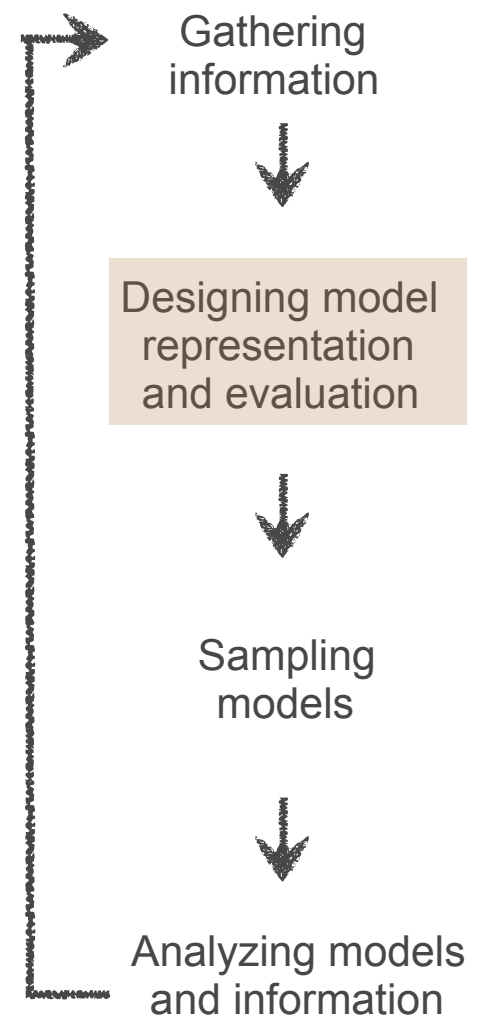


IMP topology file

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

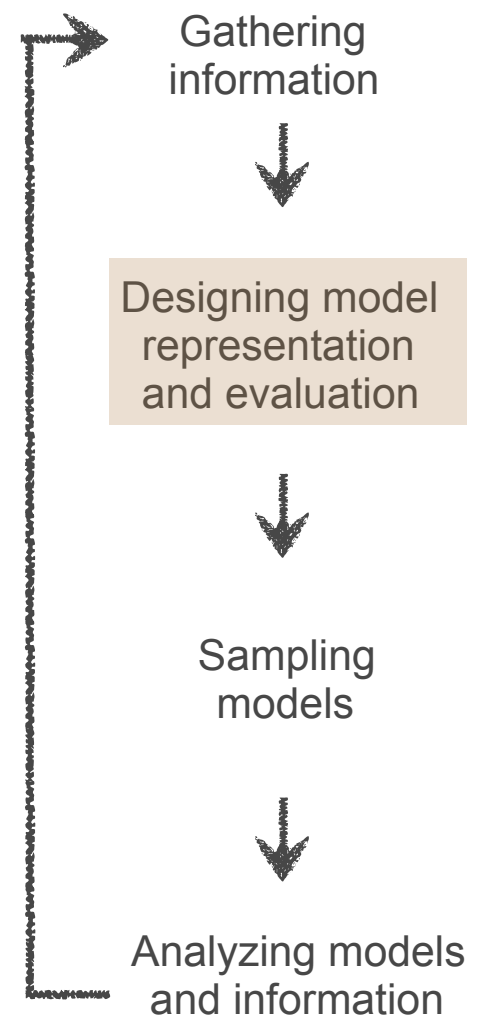
```
|directories|
|pdb_dir|. / |
|fasta_dir|. / |
|gmm_dir|. / |

|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|
```



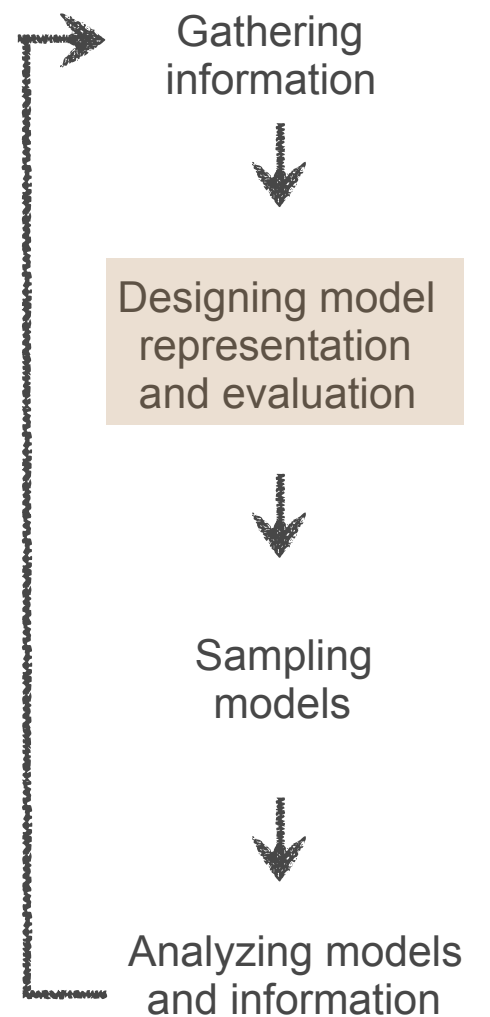
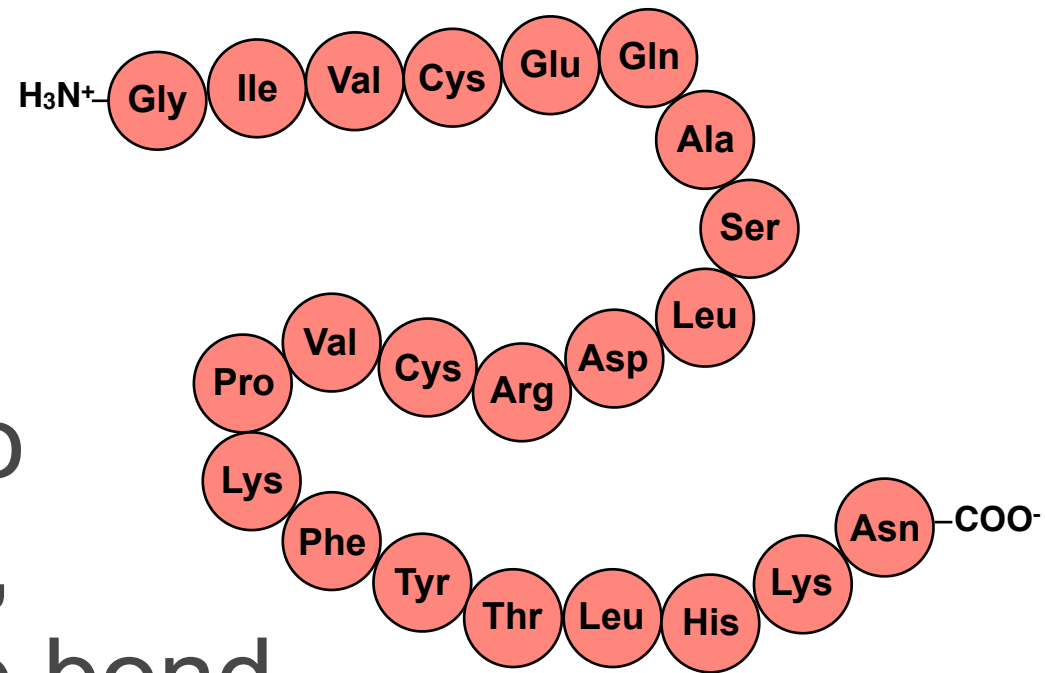
Evaluation

- 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 simple sum of individual restraints
- Each restraint maps to one of our input experiments or other physical/statistical information
- We'll look at each restraint in turn



Sequence connectivity restraint

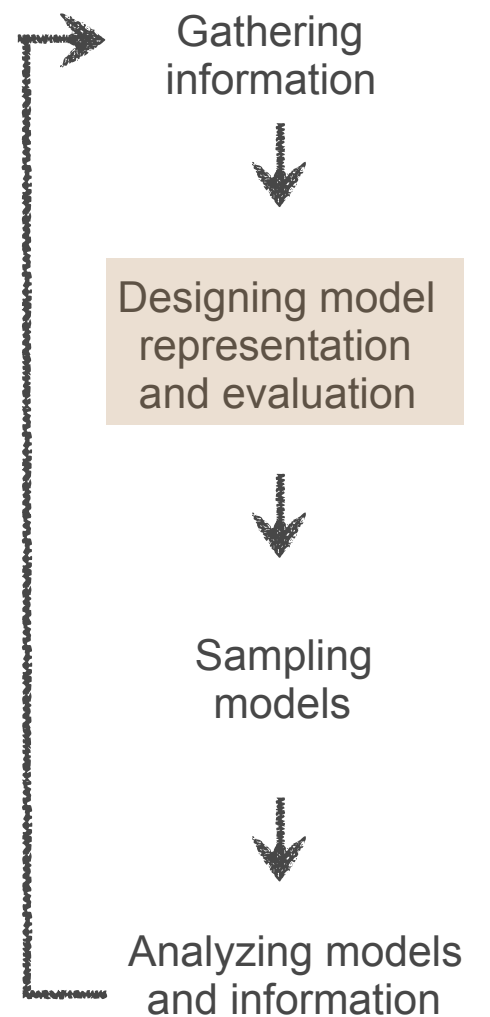
- We know that residues that are adjacent in *sequence* will also be close in *space*, due to the peptide bond
- We should enforce this in our modeling by adding simple harmonic restraints between beads (flexible string)
- PMI handles this automatically based on the FASTA file
 - nothing further needed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



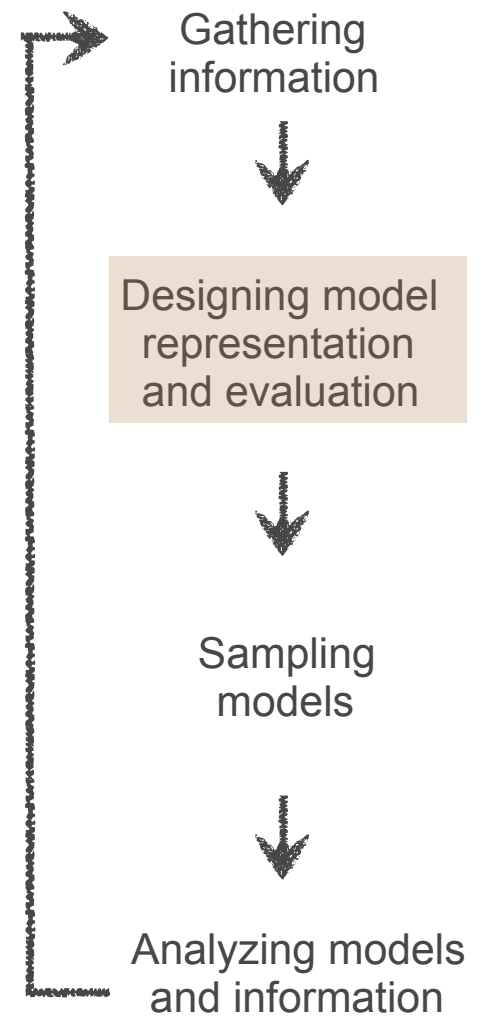
Excluded volume restraint

- 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)
```

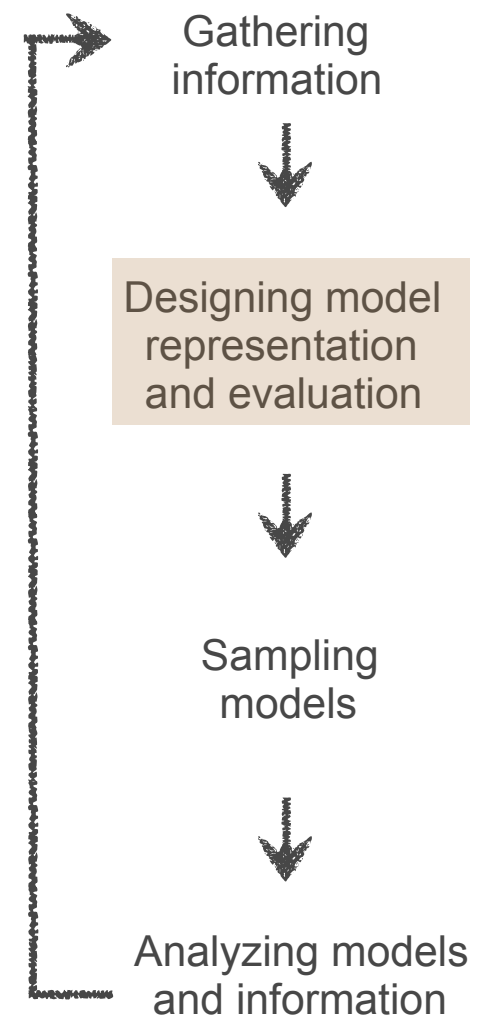


PMI vs. core IMP restraints



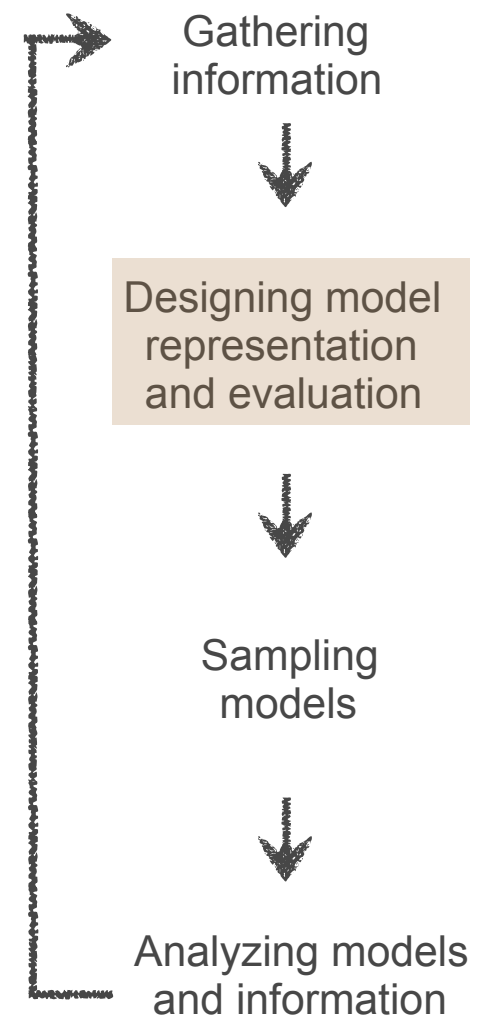
PMI vs. core IMP restraints

- Compare IMP.pmi's `ExcludedVolumeSphere` restraint with the IMP.core `SingletonRestraint` seen earlier in the simple example Python script



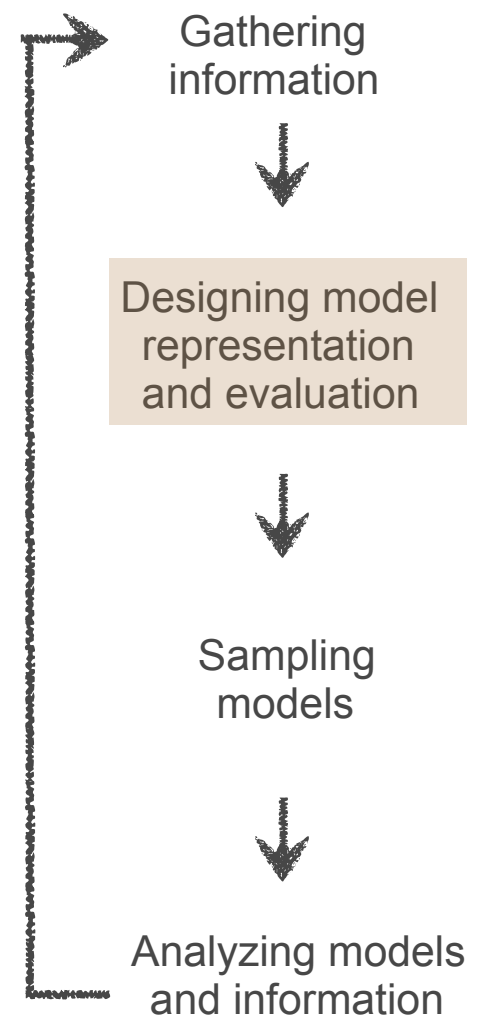
PMI vs. core IMP restraints

- Compare IMP.pmi's `ExcludedVolumeSphere` restraint with the IMP.core `SingletonRestraint` seen earlier in the simple example Python script
- Core IMP restraints act on explicitly defined particles (bottom up)



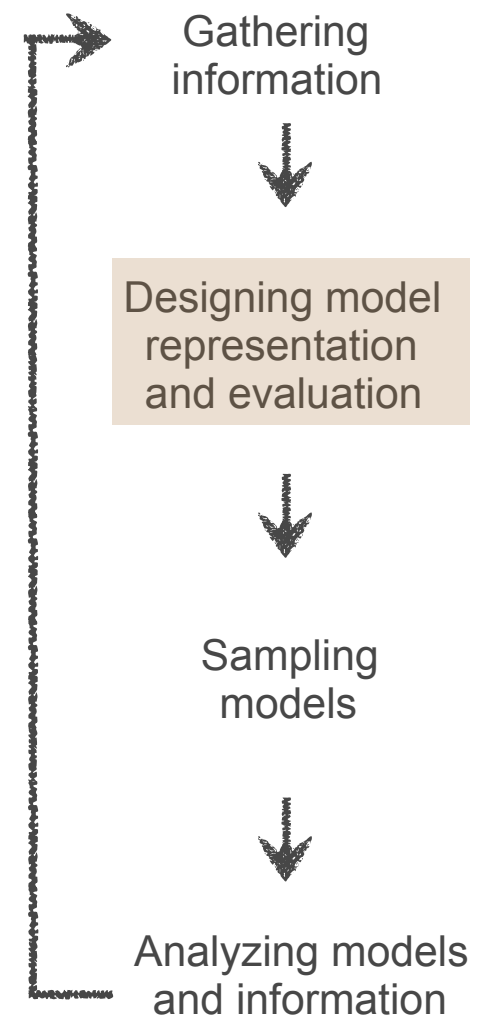
PMI vs. core IMP restraints

- Compare IMP.pmi's `ExcludedVolumeSphere` restraint with the IMP.core `SingletonRestraint` seen earlier in the simple example Python script
- Core IMP restraints act on explicitly defined particles (bottom up)
- PMI restraints act on named biological units (or the entire system, as in this case; top down)



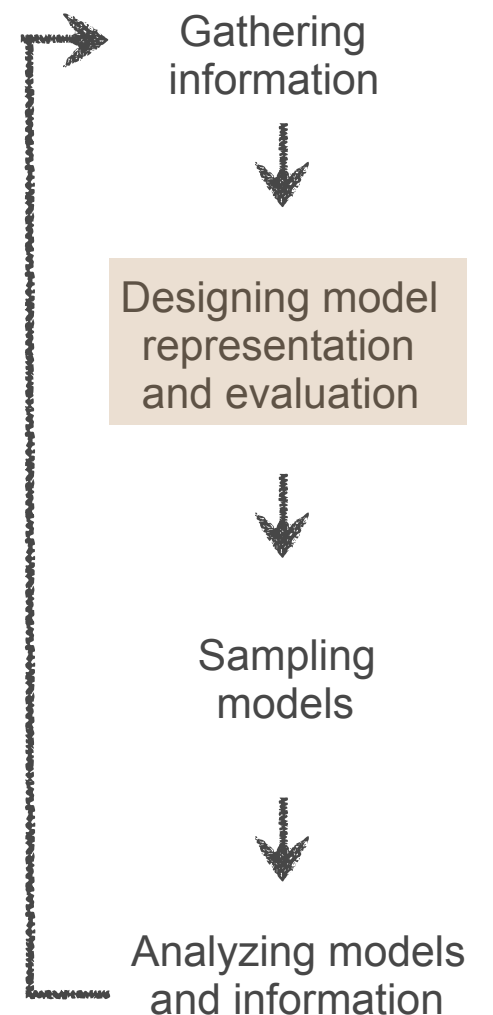
PMI vs. core IMP restraints

- Compare IMP.pmi's `ExcludedVolumeSphere` restraint with the IMP.core `SingletonRestraint` seen earlier in the simple example Python script
- Core IMP restraints act on explicitly defined particles (bottom up)
- PMI restraints act on named biological units (or the entire system, as in this case; top down)
- PMI restraints are automatically multi-scale (unlike core restraints)



PMI vs. core IMP restraints

- Compare IMP.pmi's `ExcludedVolumeSphere` restraint with the IMP.core `SingletonRestraint` seen earlier in the simple example Python script
- Core IMP restraints act on explicitly defined particles (bottom up)
- PMI restraints act on named biological units (or the entire system, as in this case; top down)
- PMI restraints are automatically multi-scale (unlike core restraints)
- Most PMI restraints simply 'wrap' one or more underlying core IMP restraints

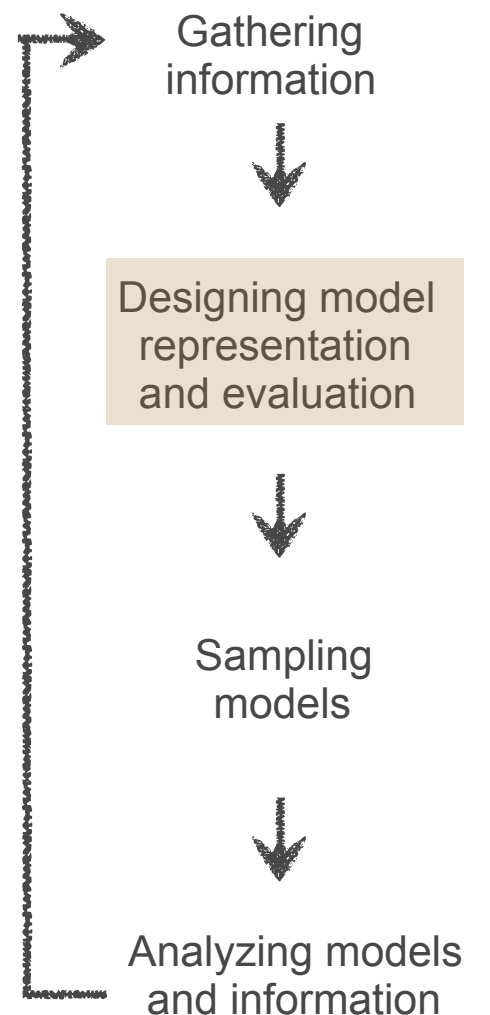


EM restraint

- 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 (i.e. zero GMM overlap)
 - `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=80.0)

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

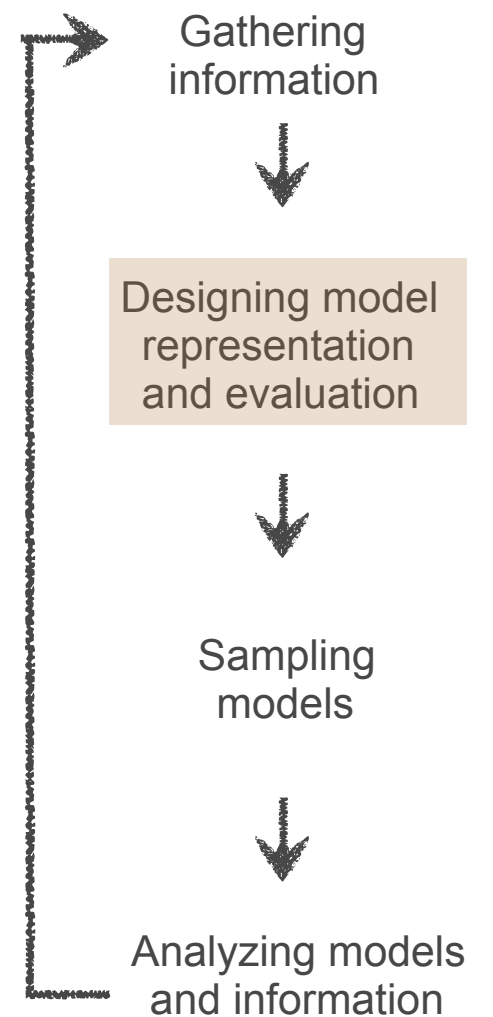


Cross-linking restraints

- Restrain residue pairs based on the cross-links 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**)

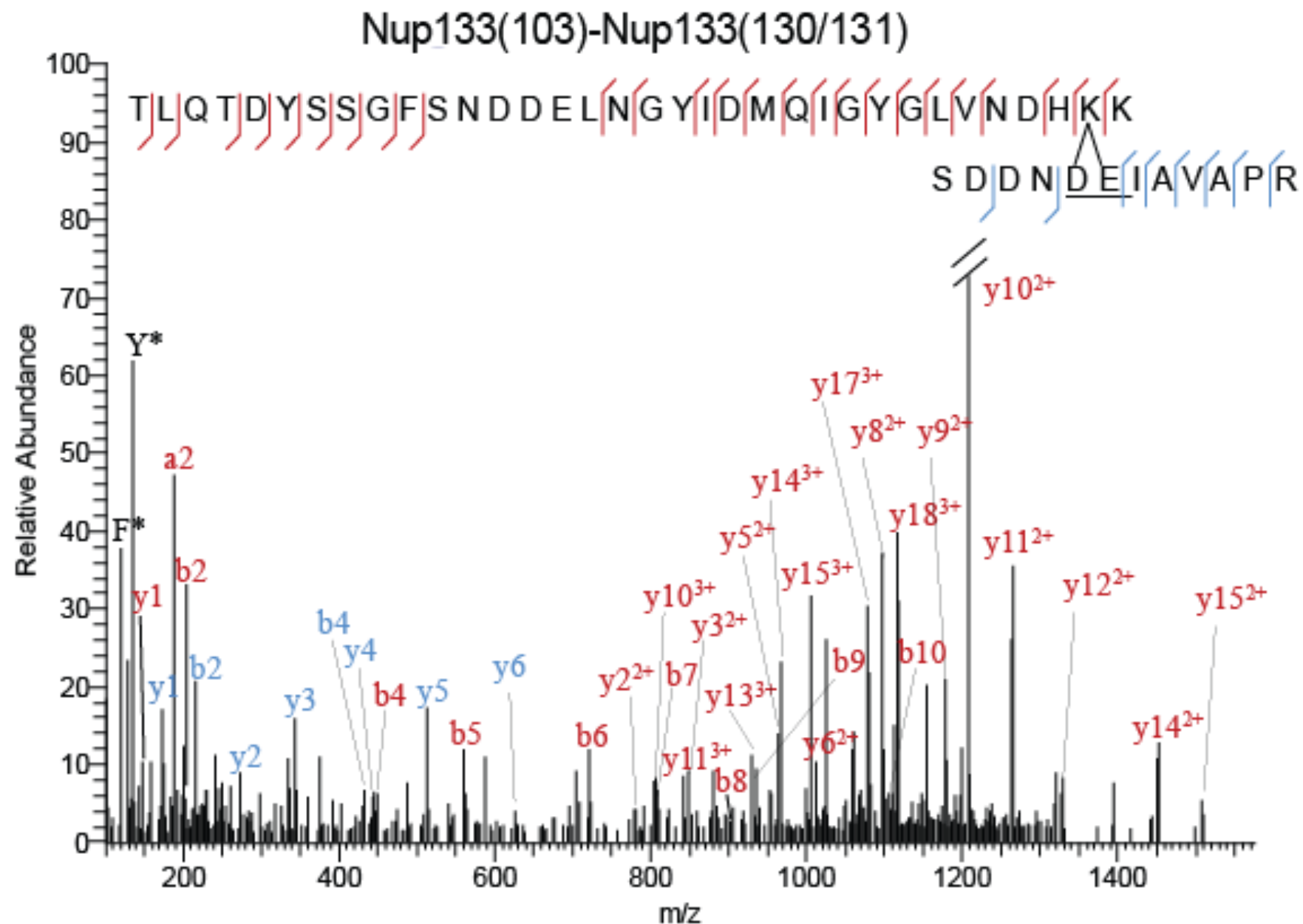
```
x11 = IMP.pmi.restraints.crosslinking.ISDCrossLinkMS(representation,
                                                    datadirectory+'polii_xlinks.csv',
                                                    length=21.0,
                                                    slope=0.01,
                                                    columnmapping=columnmap,
                                                    resolution=1.0,
                                                    label="Trnka",
                                                    csvfile=True)
```

```
x11.add_to_model()
sampleobjects.append(x11)
outputobjects.append(x11)
```



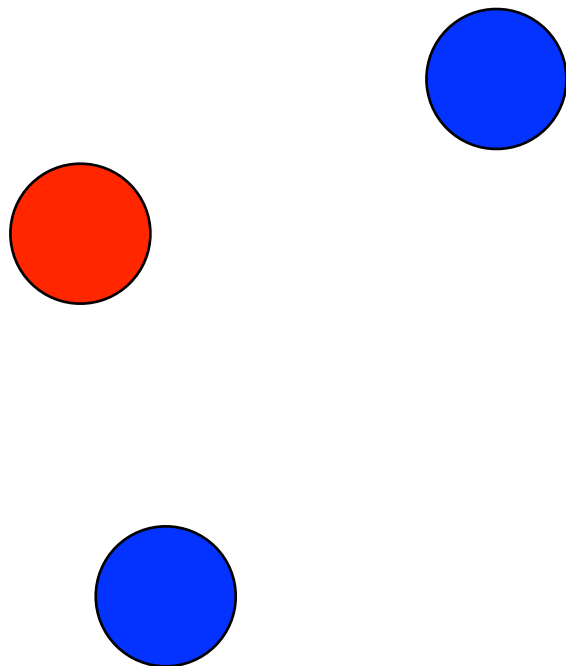
XL-MS Ambiguity (1/3)

- Not always possible to uniquely *identify* a cross-link from the spectra:



XL-MS Ambiguity (2/3)

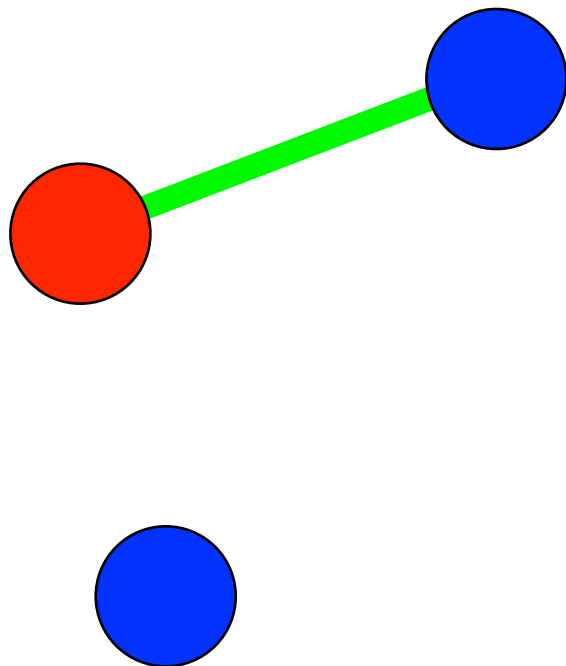
- *Compositional* ambiguity can also occur if there are multiple copies of a protein available (although not the case here):



A cross-link observed between the red and blue proteins does not identify *which* blue protein is interacting with red

XL-MS Ambiguity (2/3)

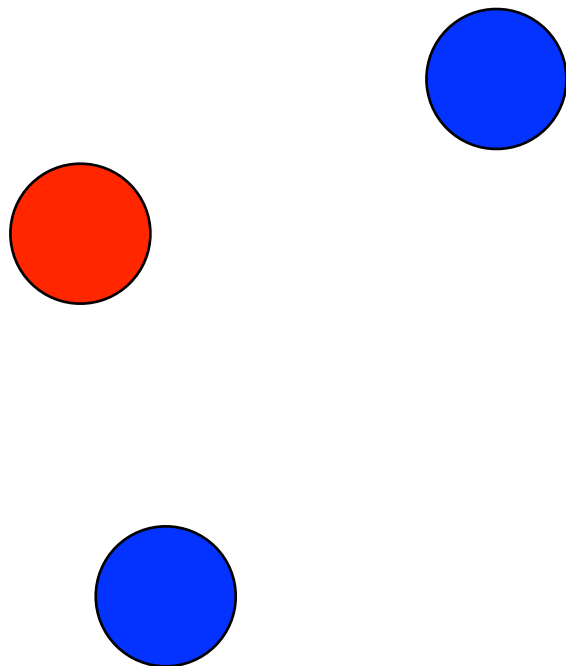
- *Compositional* ambiguity can also occur if there are multiple copies of a protein available (although not the case here):



A cross-link observed between the red and blue proteins does not identify *which* blue protein is interacting with red

XL-MS Ambiguity (2/3)

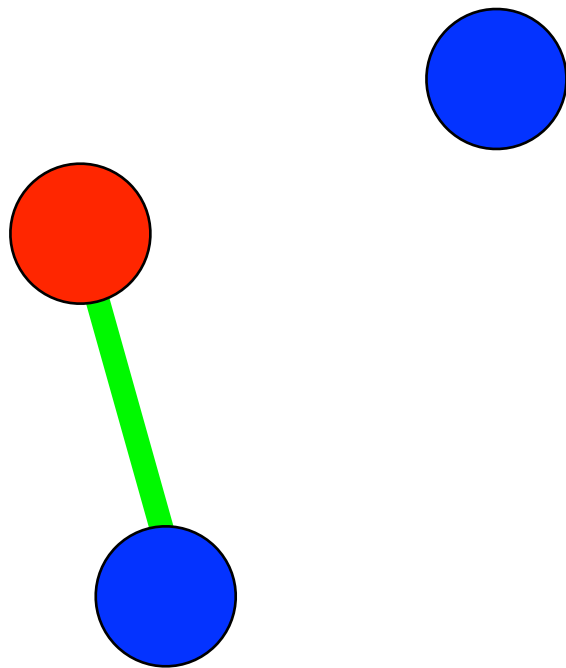
- *Compositional* ambiguity can also occur if there are multiple copies of a protein available (although not the case here):



A cross-link observed between the red and blue proteins does not identify *which* blue protein is interacting with red

XL-MS Ambiguity (2/3)

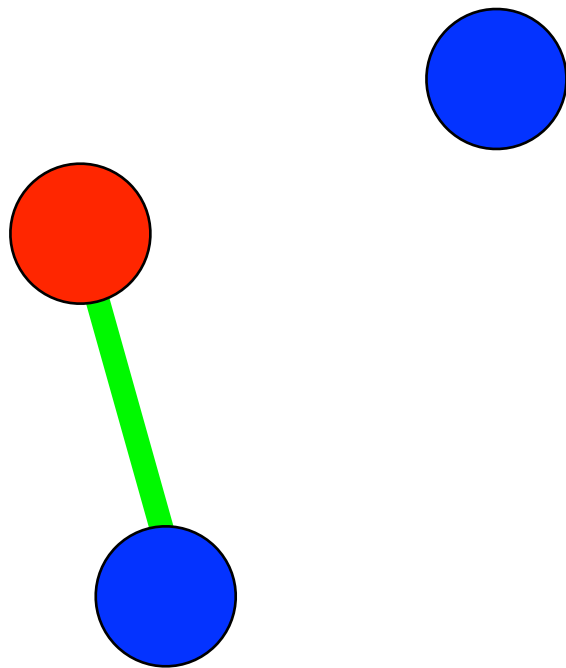
- *Compositional* ambiguity can also occur if there are multiple copies of a protein available (although not the case here):



A cross-link observed between the red and blue proteins does not identify *which* blue protein is interacting with red

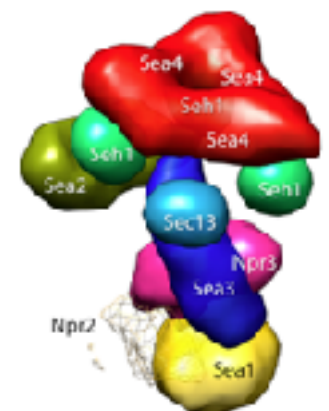
XL-MS Ambiguity (2/3)

- *Compositional* ambiguity can also occur if there are multiple copies of a protein available (although not the case here):



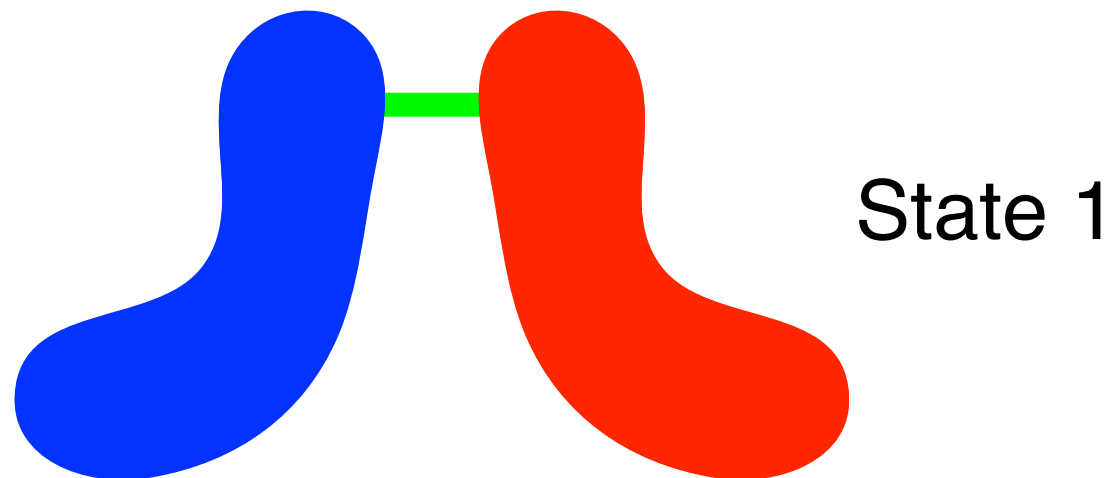
A cross-link observed between the red and blue proteins does not identify *which* blue protein is interacting with red

<https://salilab.org/sea>



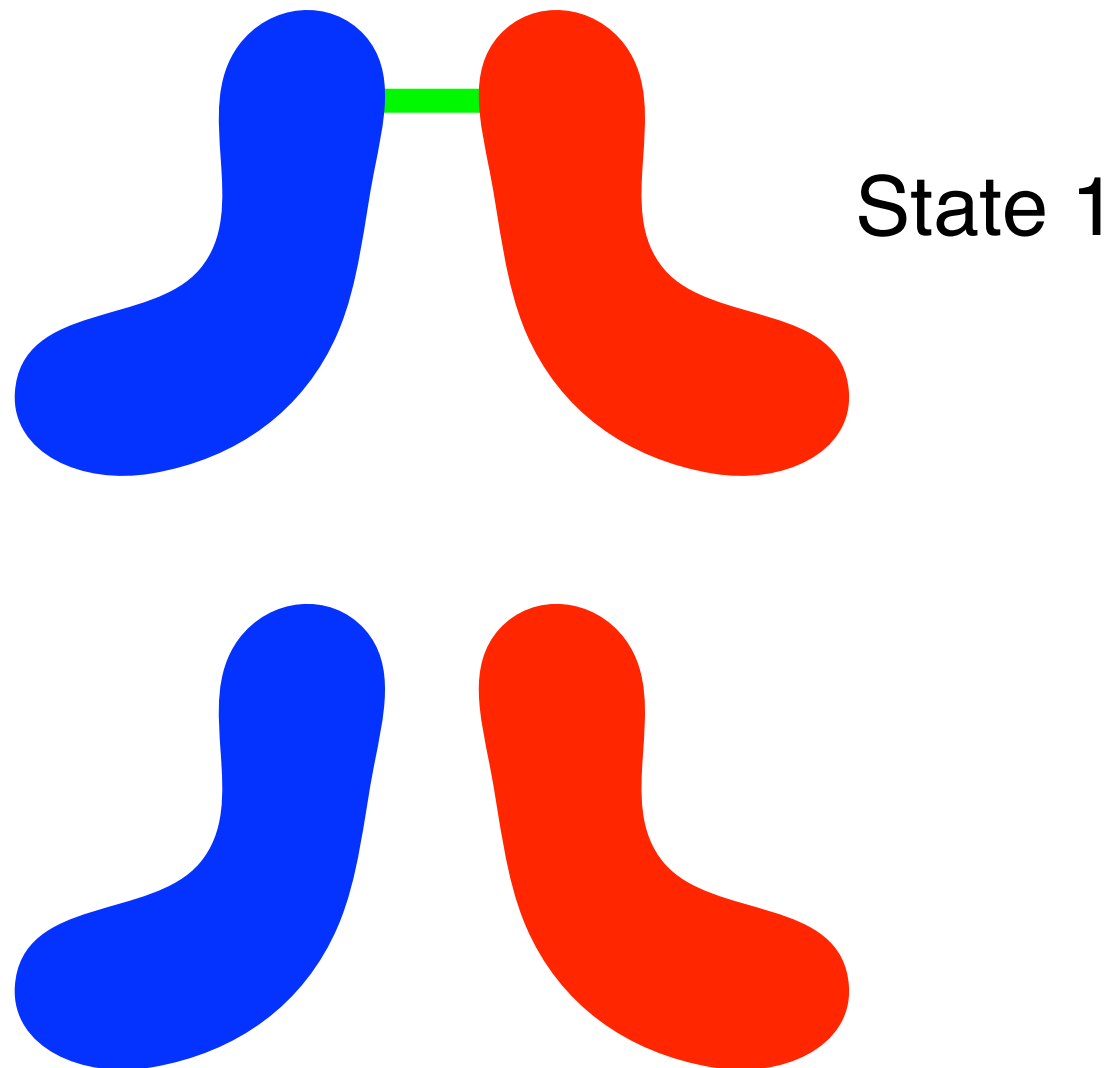
XL-MS Ambiguity (3/3)

- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):



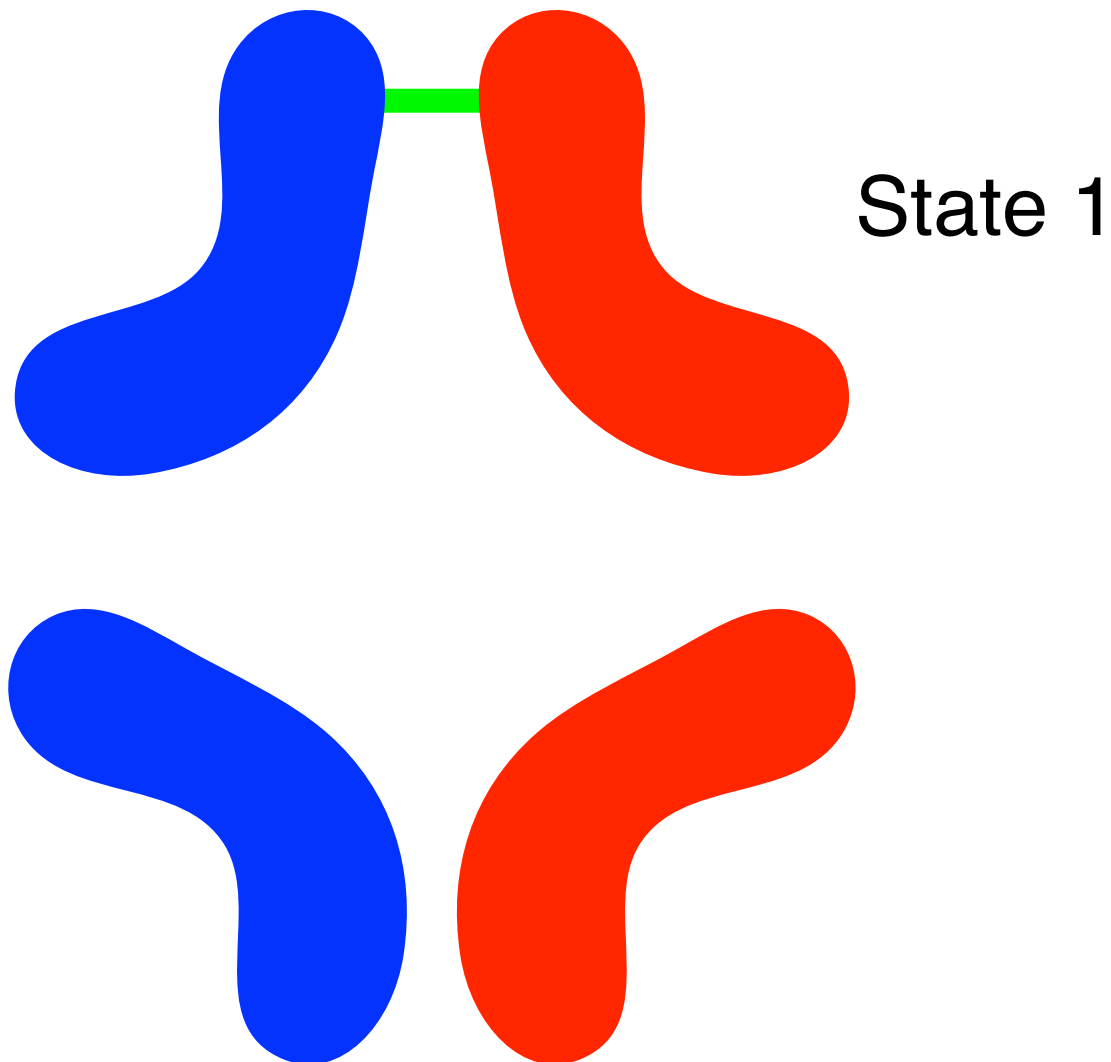
XL-MS Ambiguity (3/3)

- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):



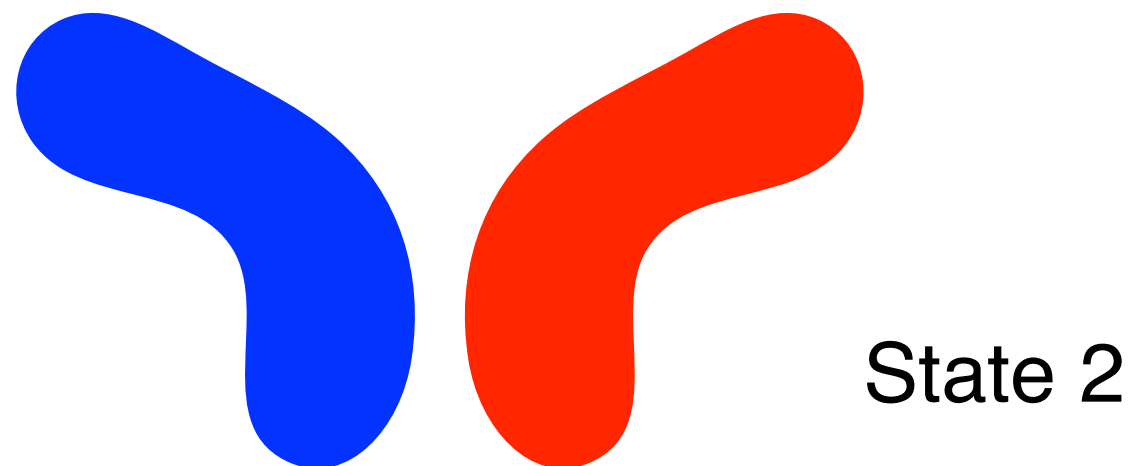
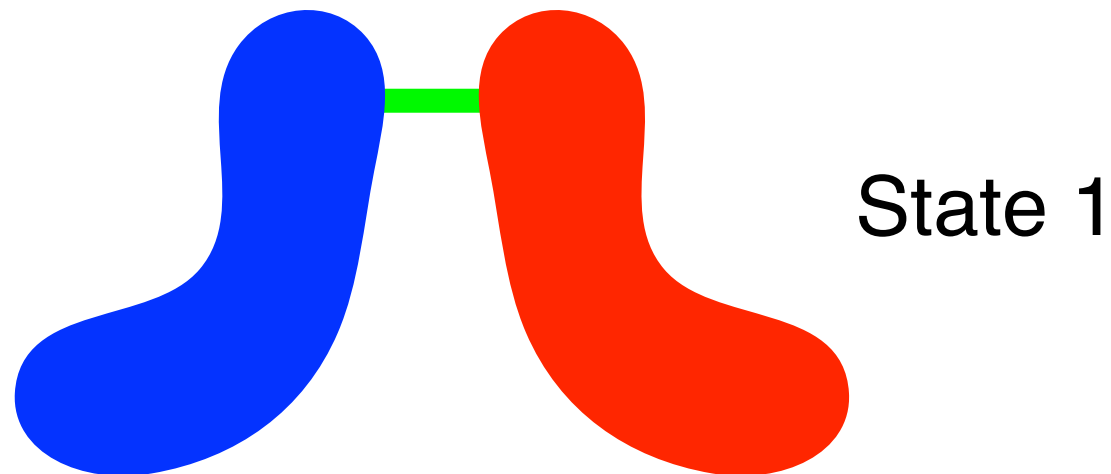
XL-MS Ambiguity (3/3)

- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):



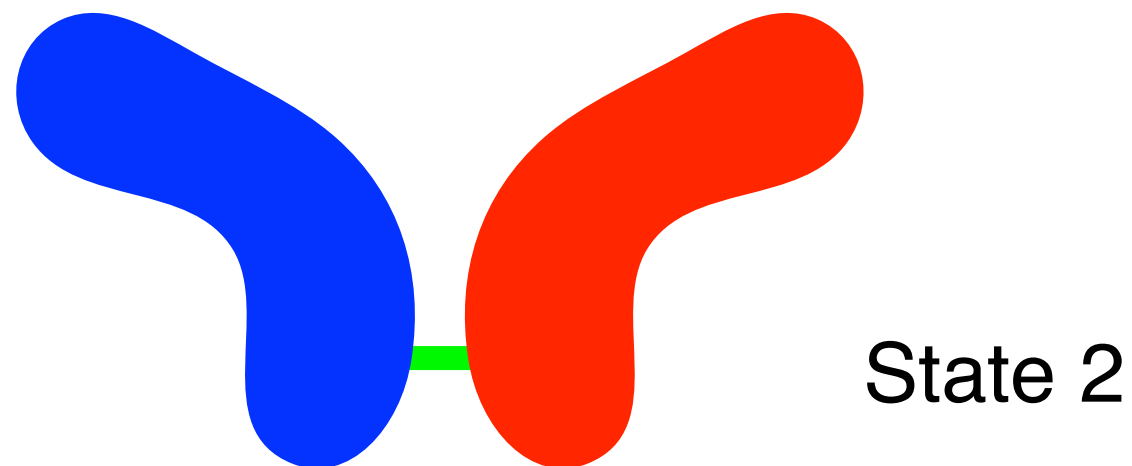
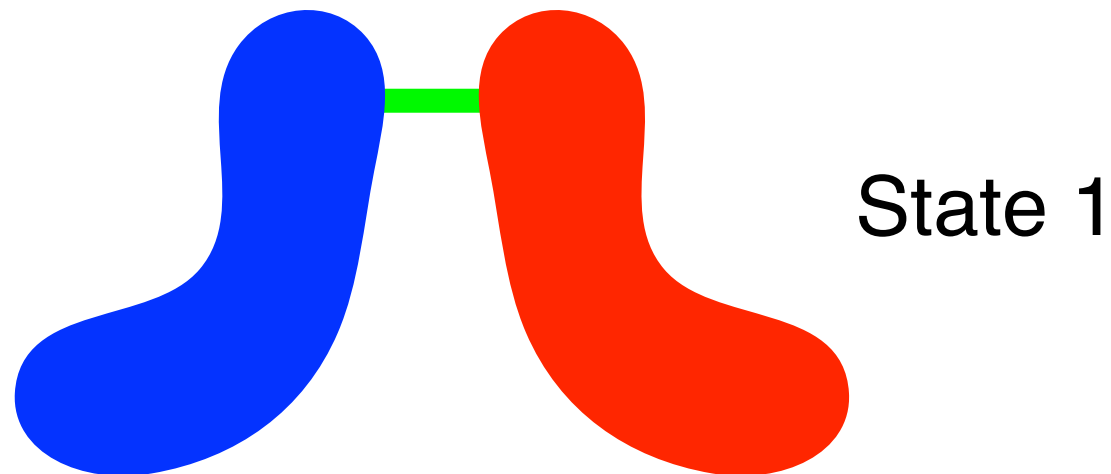
XL-MS Ambiguity (3/3)

- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):



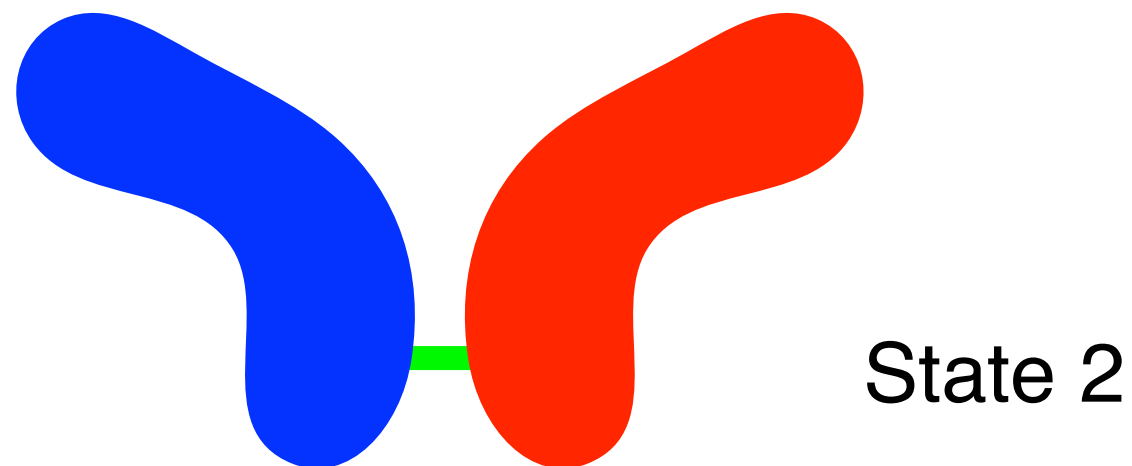
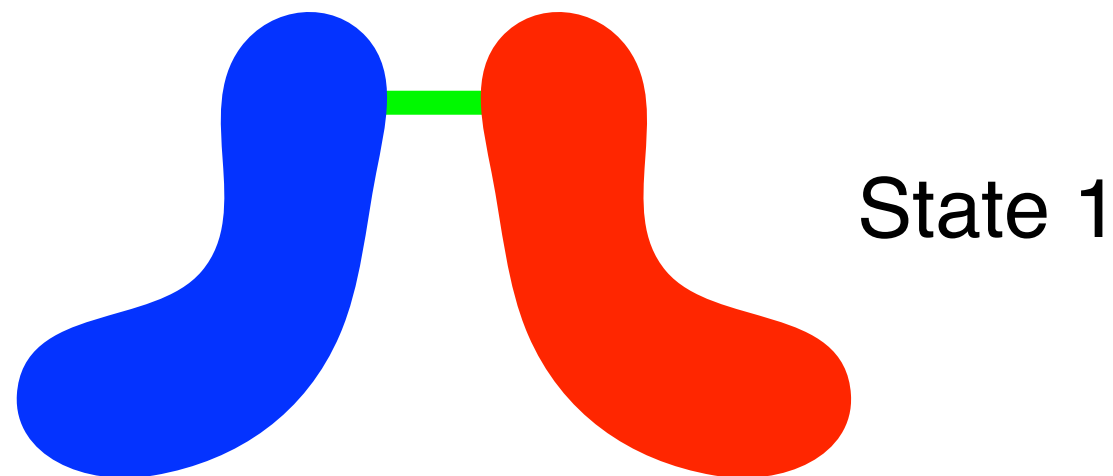
XL-MS Ambiguity (3/3)

- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):



XL-MS Ambiguity (3/3)

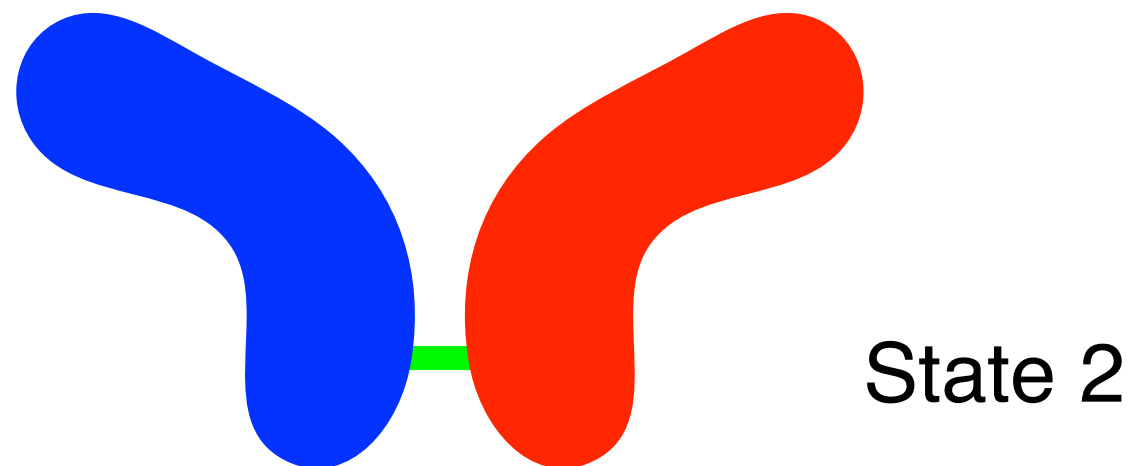
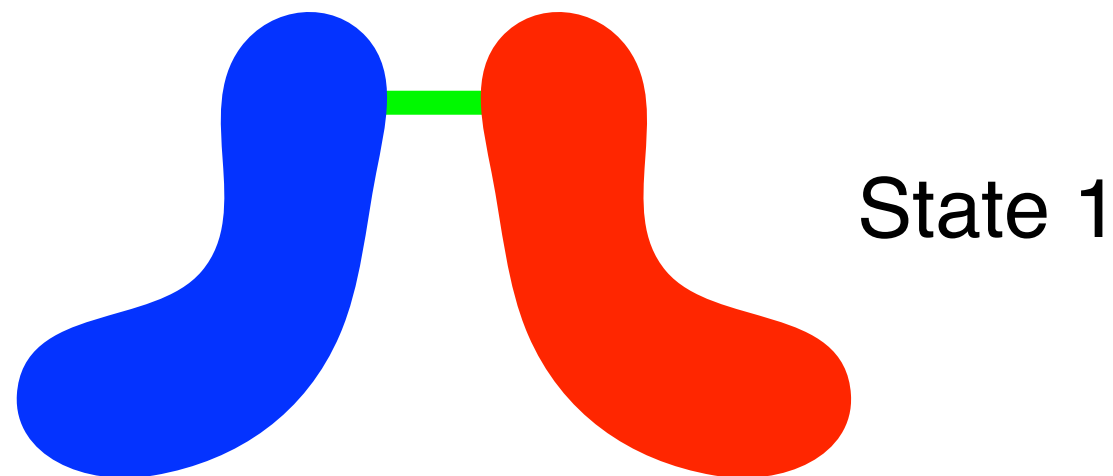
- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):



- The cross-linking experiment will yield cross-links representative of both states

XL-MS Ambiguity (3/3)

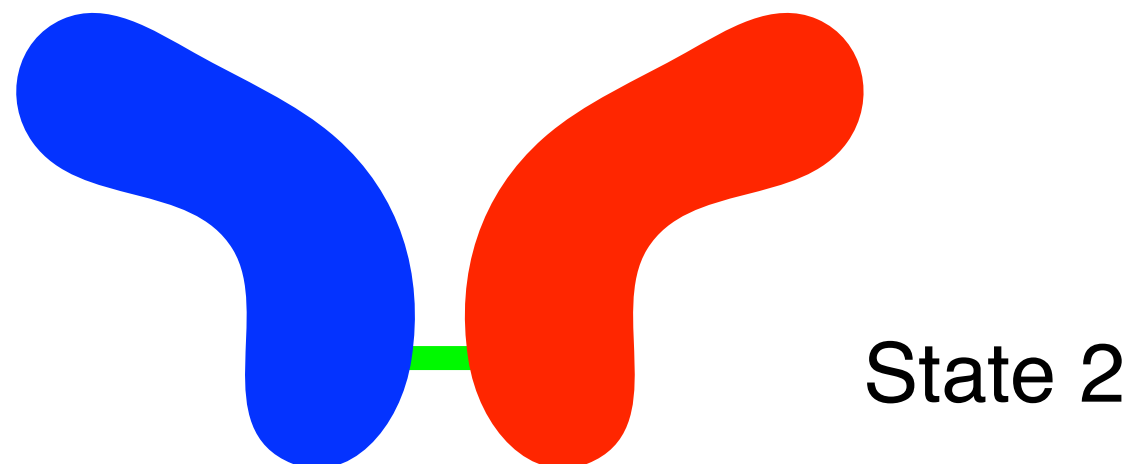
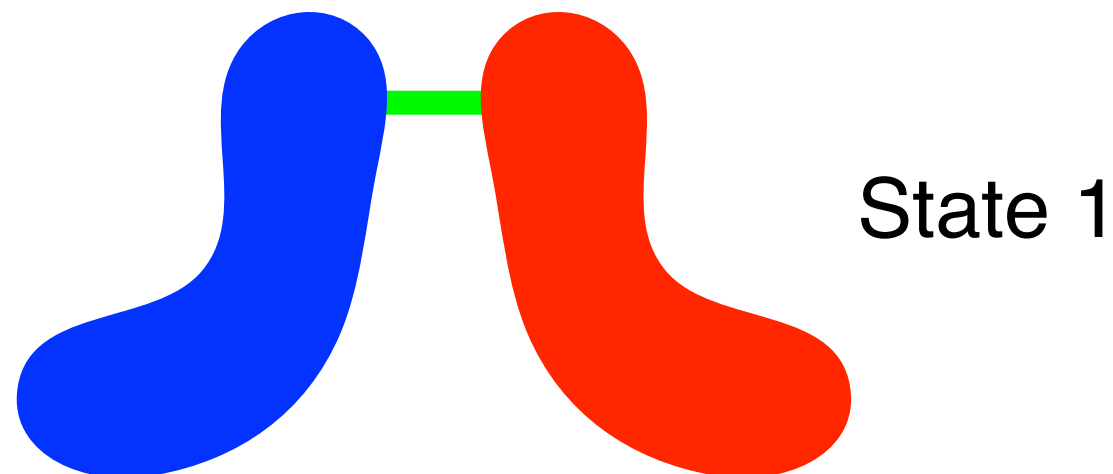
- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):



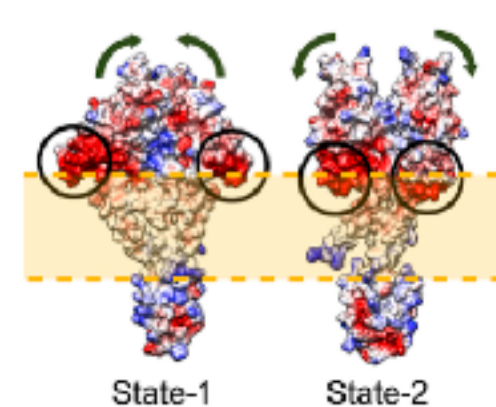
- The cross-linking experiment will yield cross-links representative of both states
- A single model cannot satisfy both cross-links simultaneously

XL-MS Ambiguity (3/3)

- *State* ambiguity can also occur if there are multiple states of the complex present (heterogeneity):

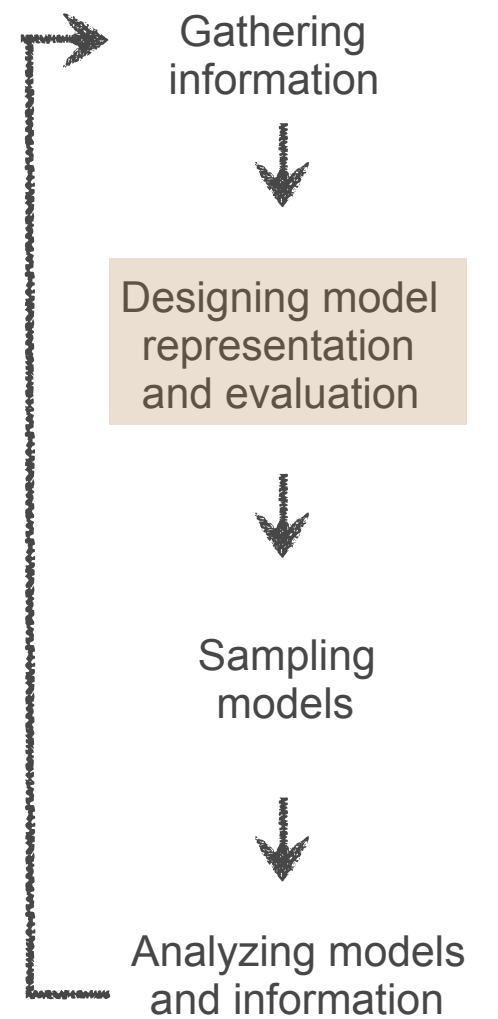


- The cross-linking experiment will yield cross-links representative of both states
- A single model cannot satisfy both cross-links simultaneously



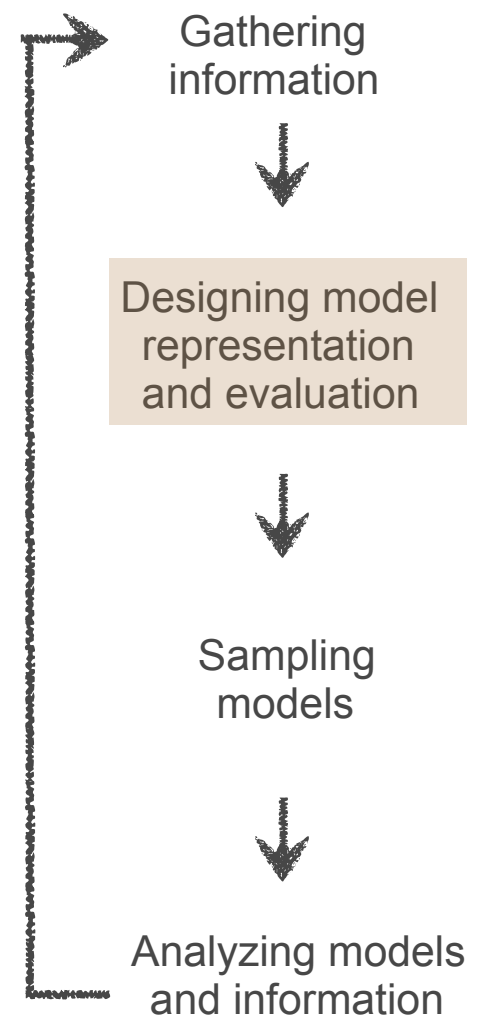
<https://salilab.org/phoq>

Bayesian scoring function for XL-MS



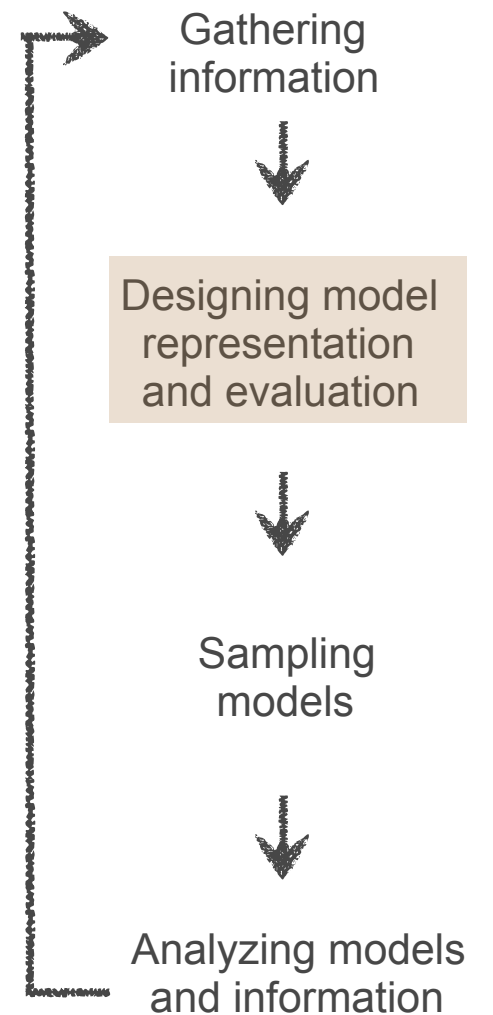
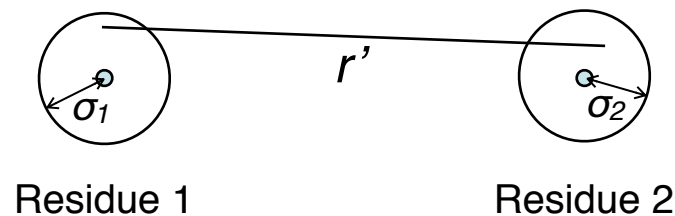
Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues



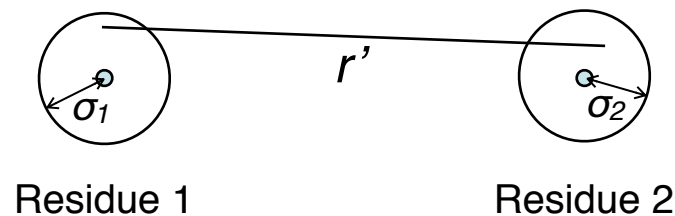
Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

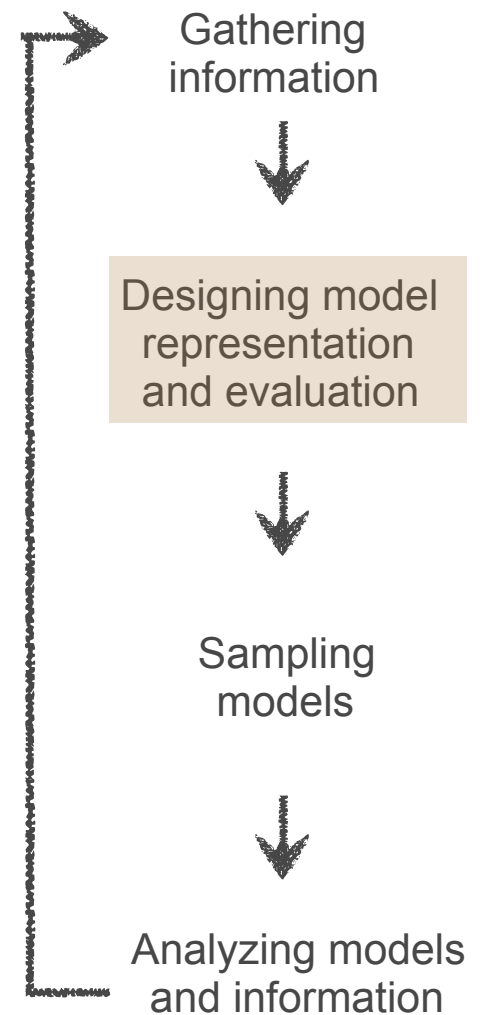


Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

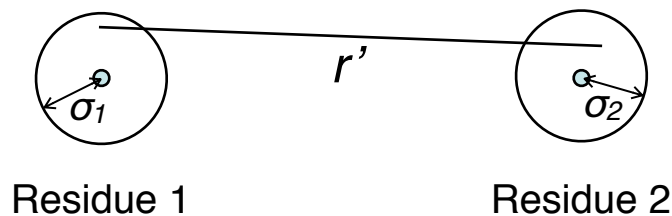


- Confidence in the cross-links themselves is measured with another parameter, ψ

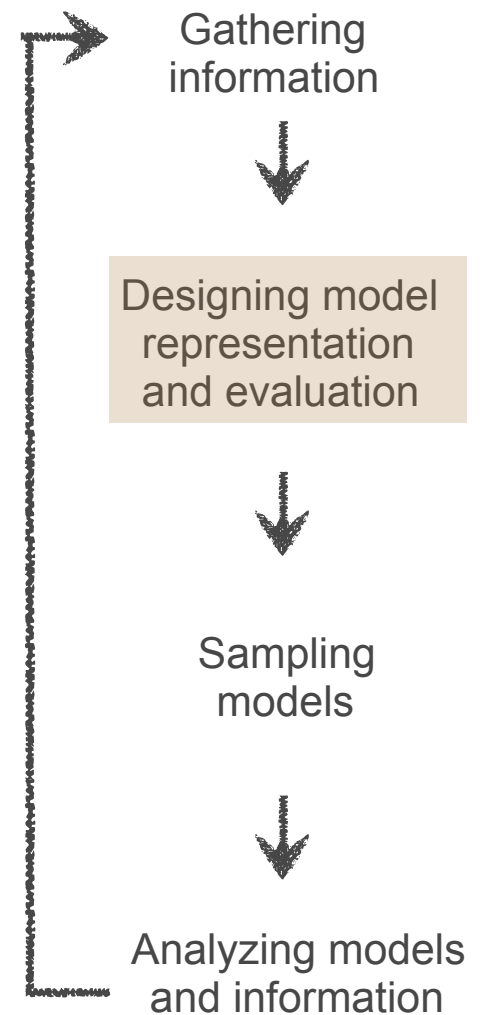


Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

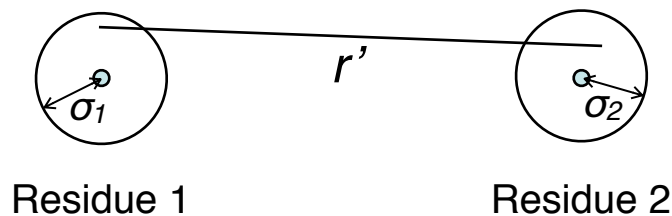


- Confidence in the cross-links themselves is measured with another parameter, ψ
- In principle, could optimize σ and ψ for *every* cross-link

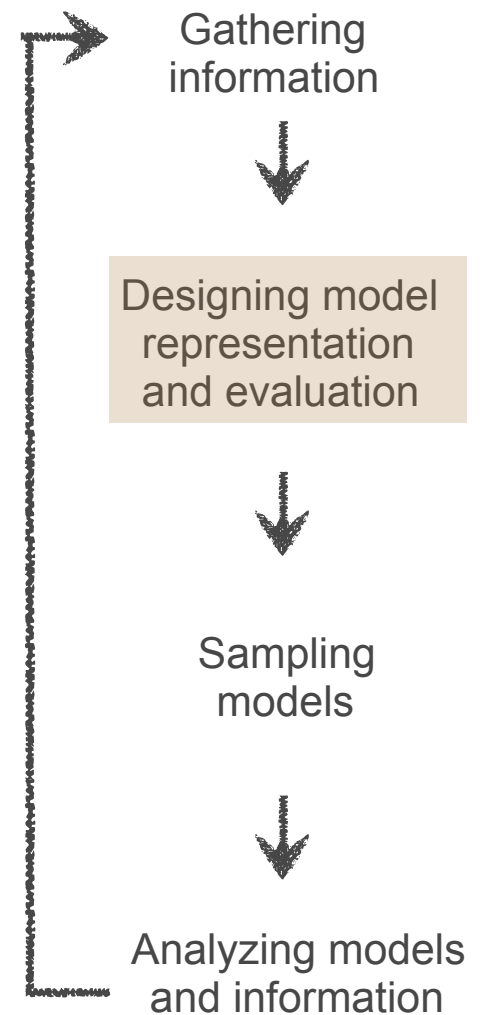


Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

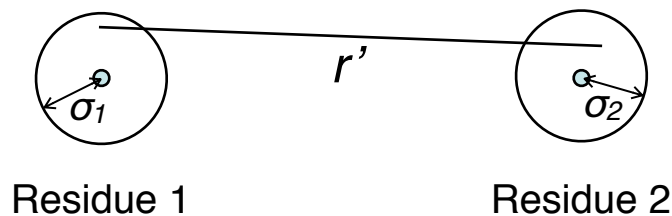


- Confidence in the cross-links themselves is measured with another parameter, ψ
- In principle, could optimize σ and ψ for *every* cross-link
- In practice, too many parameters (overfitting)

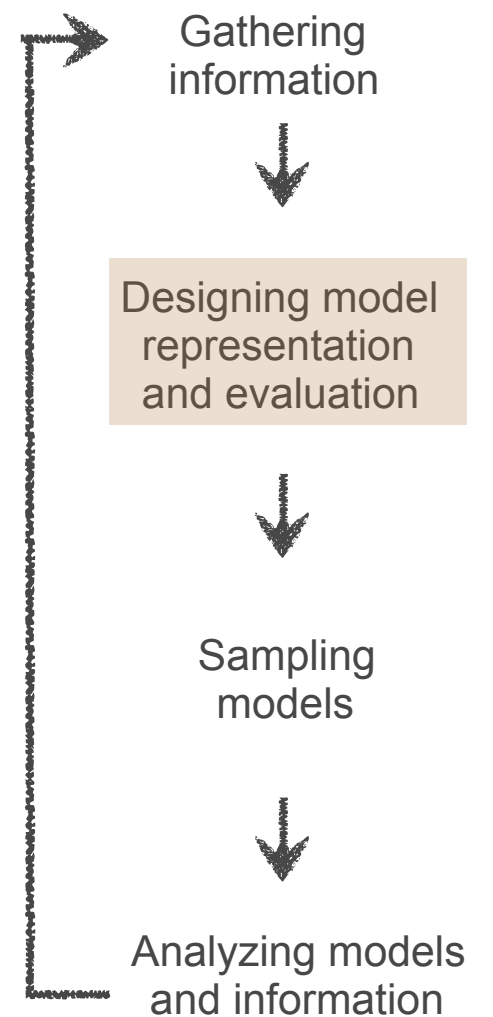


Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

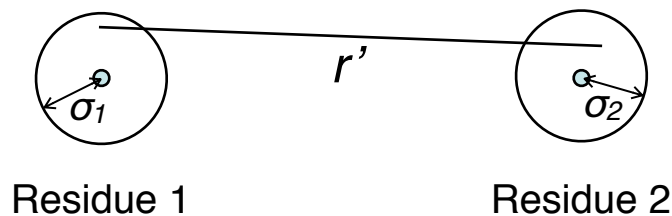


- Confidence in the cross-links themselves is measured with another parameter, ψ
- In principle, could optimize σ and ψ for *every* cross-link
- In practice, too many parameters (overfitting)
- In this case, we assume

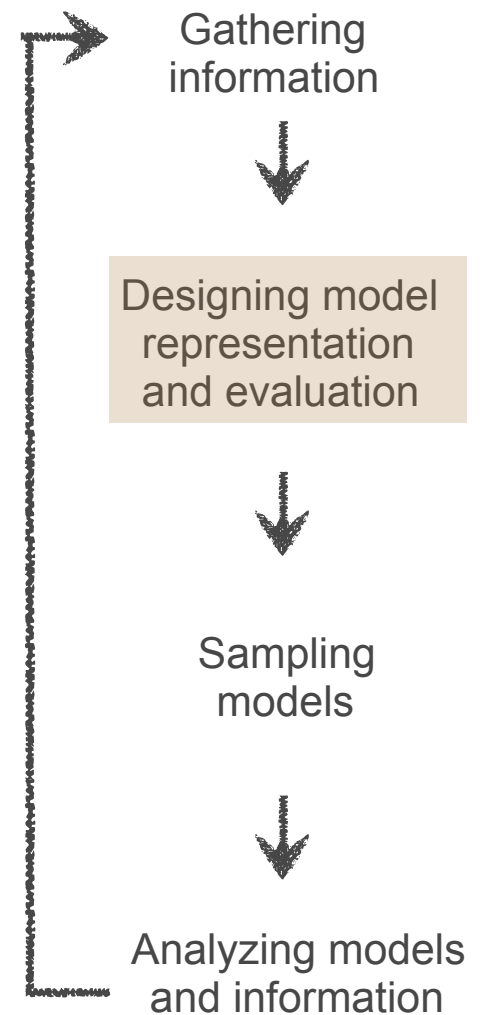


Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

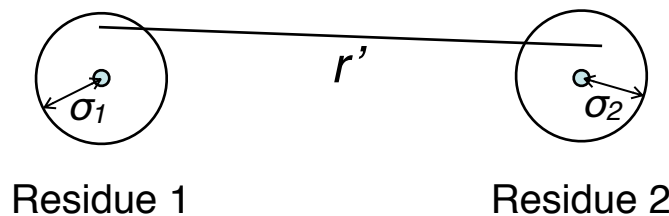


- Confidence in the cross-links themselves is measured with another parameter, ψ
- In principle, could optimize σ and ψ for *every* cross-link
- In practice, too many parameters (overfitting)
- In this case, we assume
 - Each cross-link dataset has a single ψ

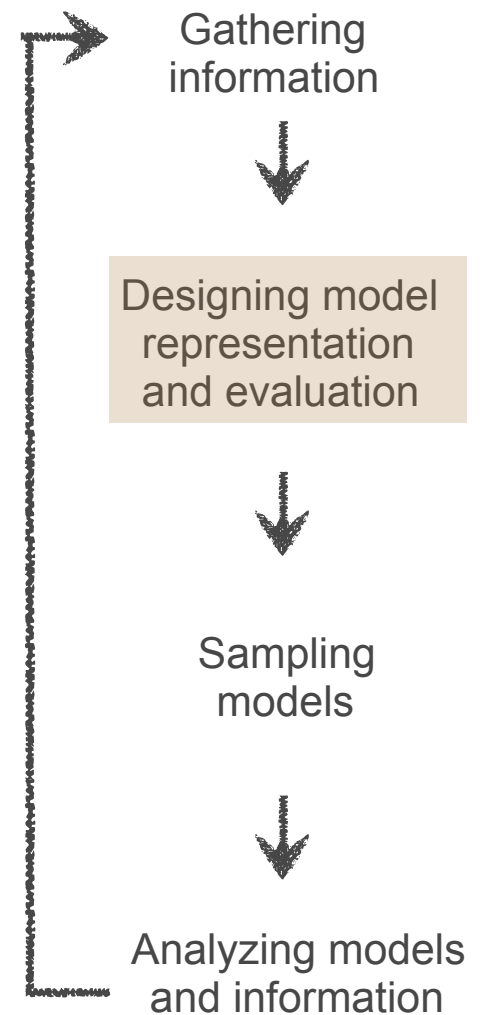


Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

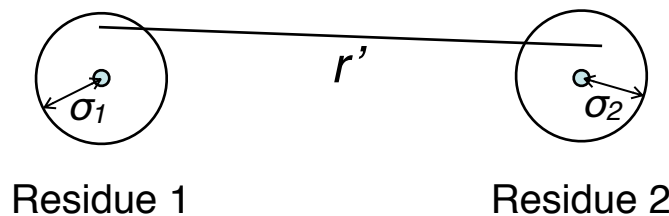


- Confidence in the cross-links themselves is measured with another parameter, ψ
- In principle, could optimize σ and ψ for *every* cross-link
- In practice, too many parameters (overfitting)
- In this case, we assume
 - Each cross-link dataset has a single ψ
 - All residues have the same σ for a dataset

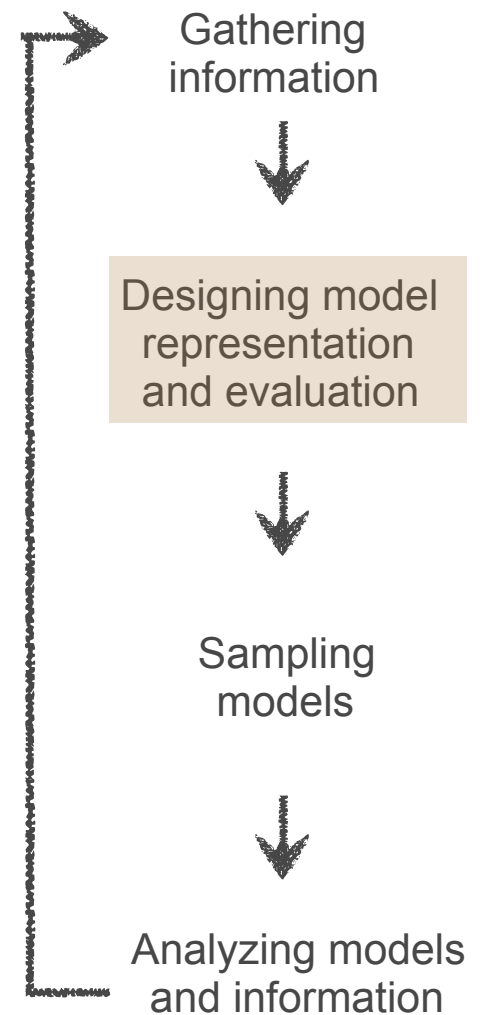


Bayesian scoring function for XL-MS

- The Bayesian restraint accounts for uncertainty in position, σ , by restraining intersphere distance between residues

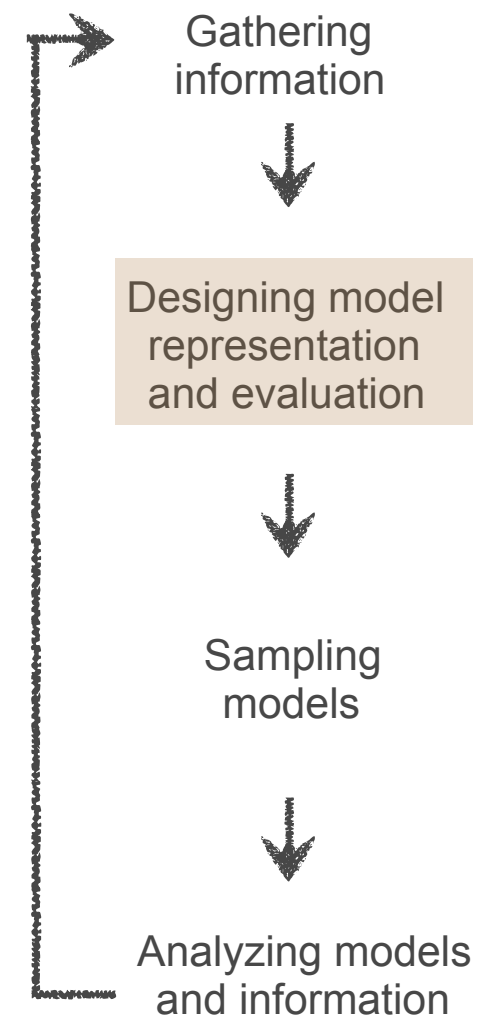


- Confidence in the cross-links themselves is measured with another parameter, ψ
- In principle, could optimize σ and ψ for *every* cross-link
- In practice, too many parameters (overfitting)
- In this case, we assume
 - Each cross-link dataset has a single ψ
 - All residues have the same σ for a dataset
- So, we will sample σ_1 , σ_2 , ψ_1 , ψ_2 during our modeling



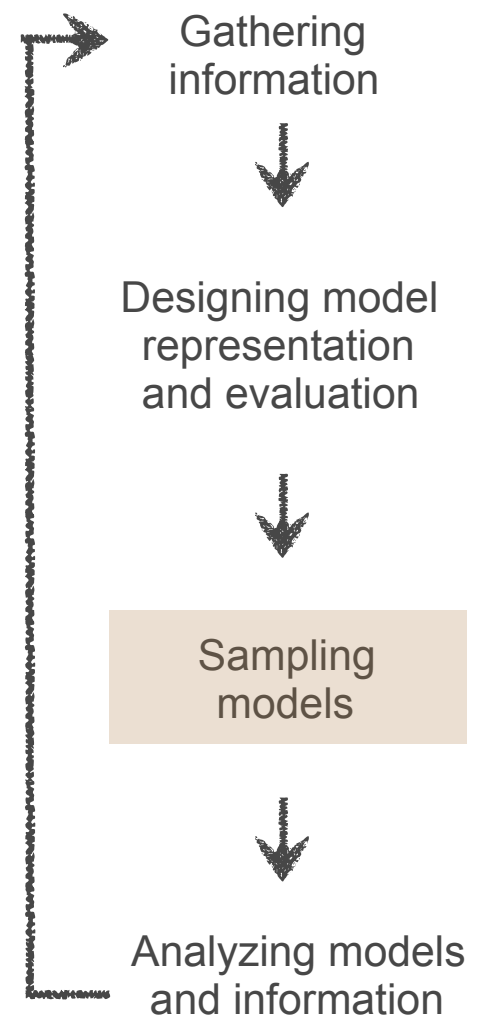
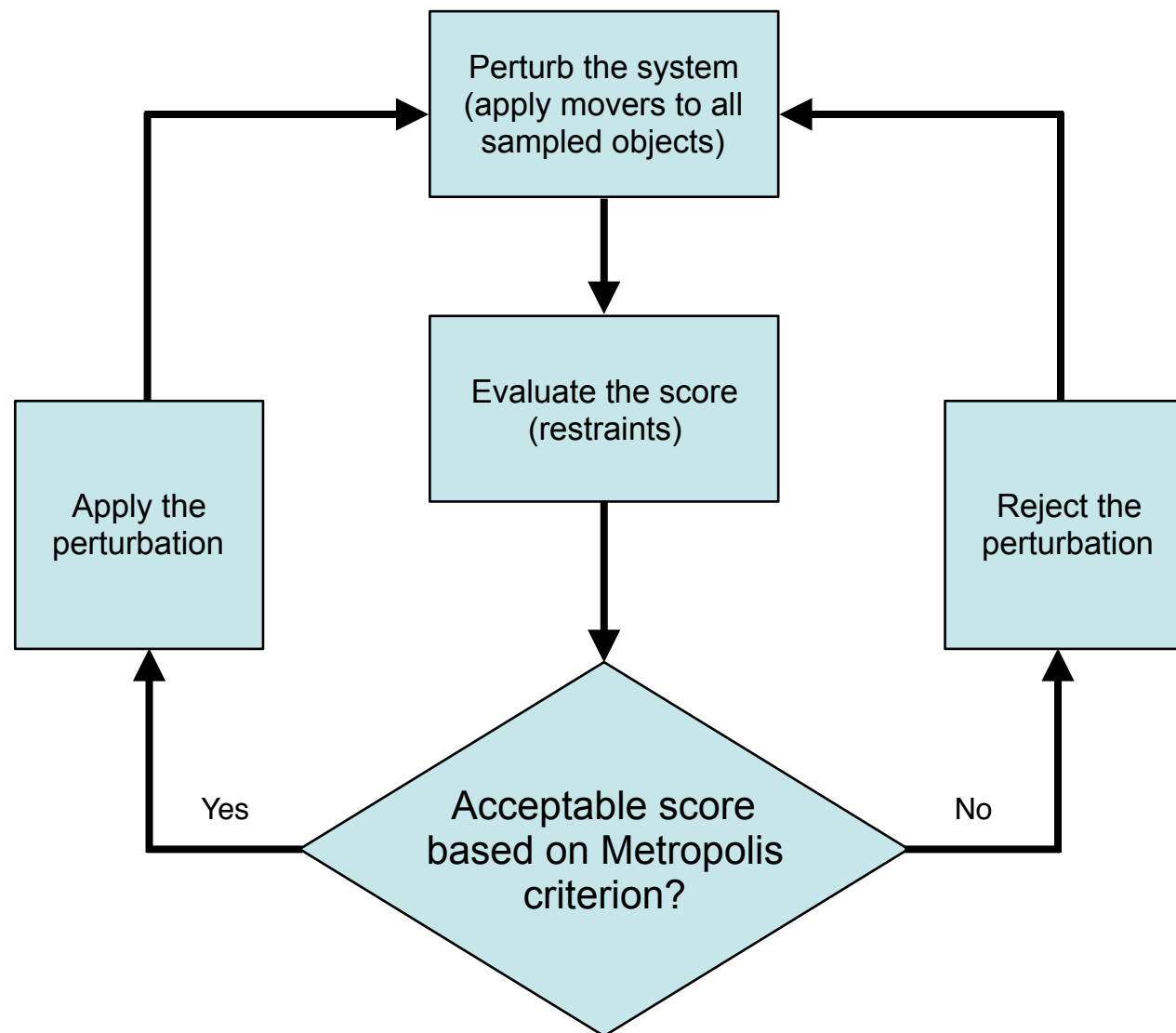
Other restraints

- Note that we're not using electrostatics or stereochemistry; very different to a typical molecular mechanics simulation
 - Electrostatics usually not relevant on this scale
 - Where it is, it is considered implicitly (from the input structures)
 - No atomic data in this case, so no stereochemistry
 - Can use CHARMM forcefield if we do have atoms



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

- Rigid body movers: simple 3D translation and rotation, sampled linearly up to given maximum values
- Bead movers: 3D translation
- Also we define here *how* to move our rigid bodies

```
#-----  
# 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"]]
```

Gathering
information



Designing model
representation
and evaluation

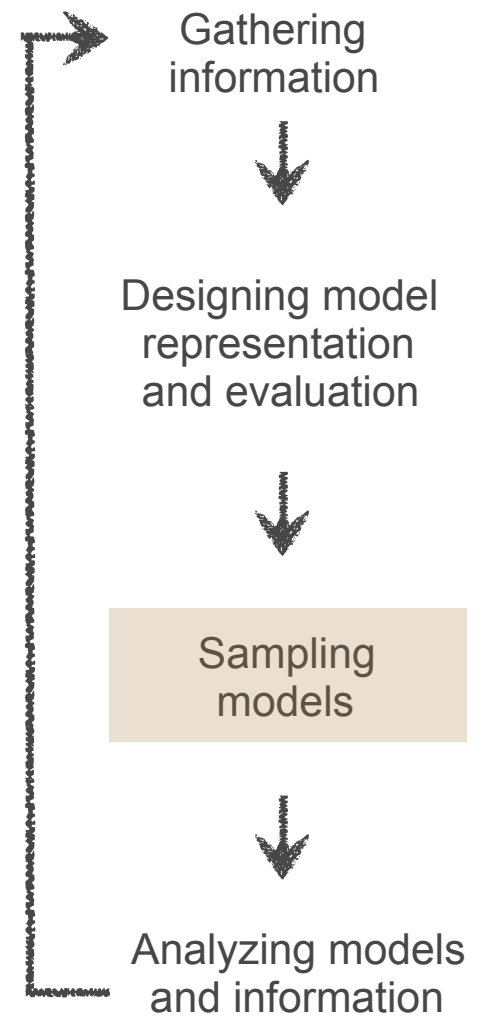
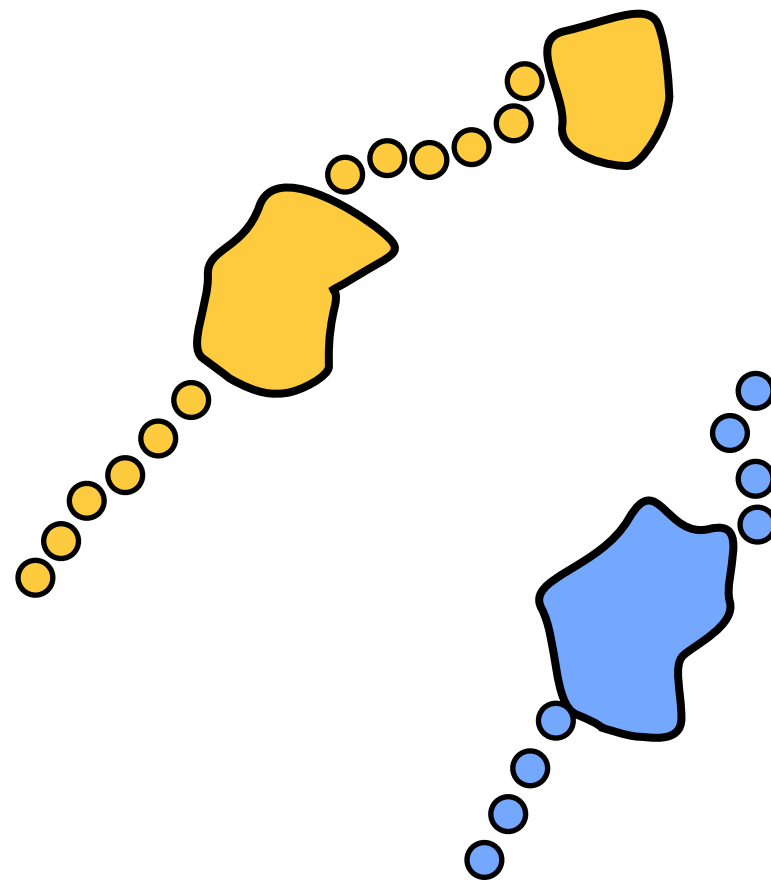


Sampling
models



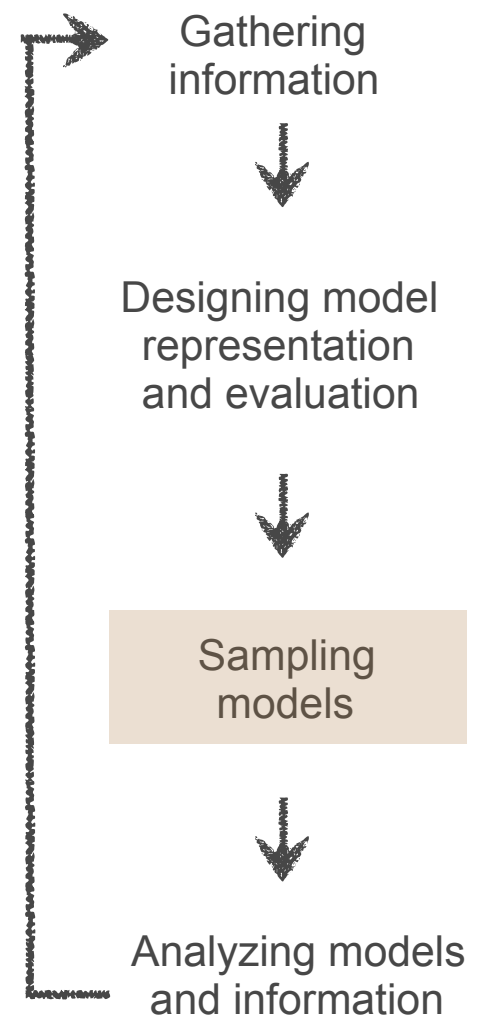
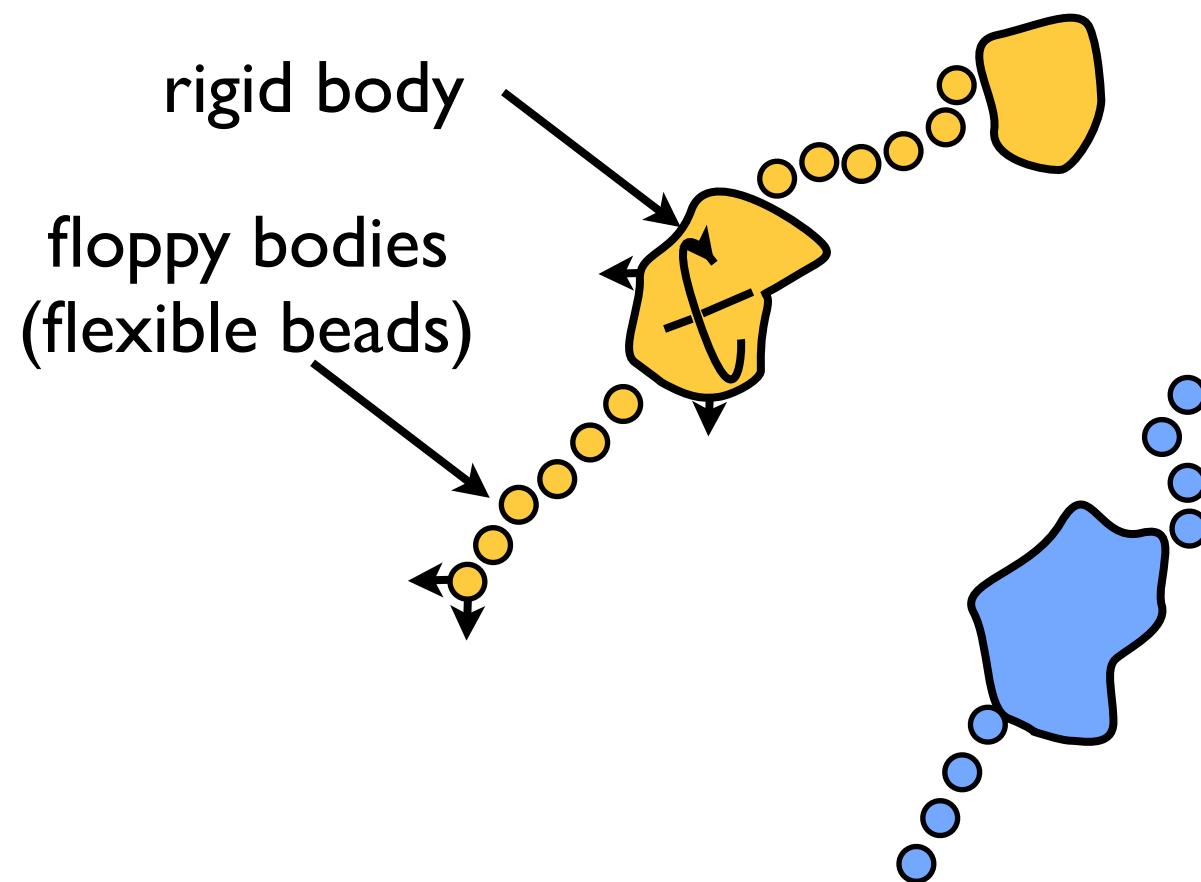
Analyzing models
and information

Rigid body movers



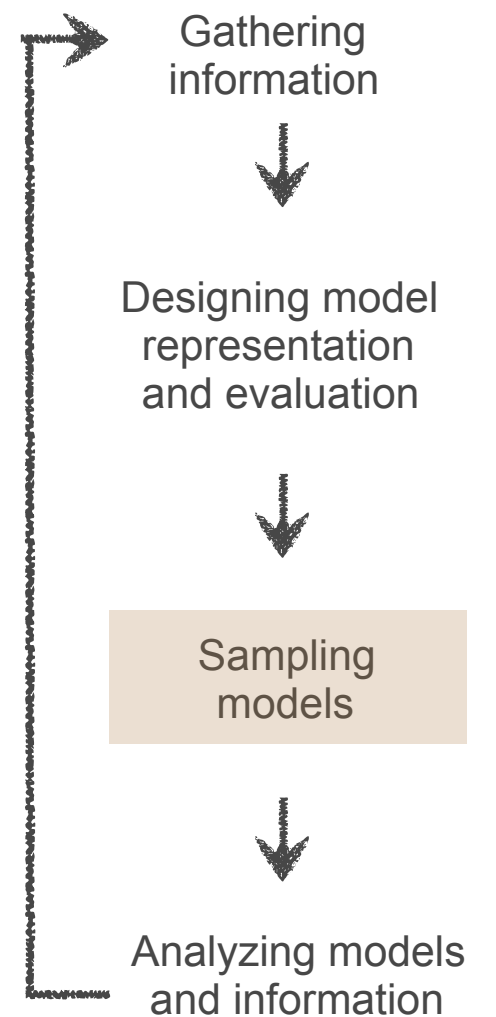
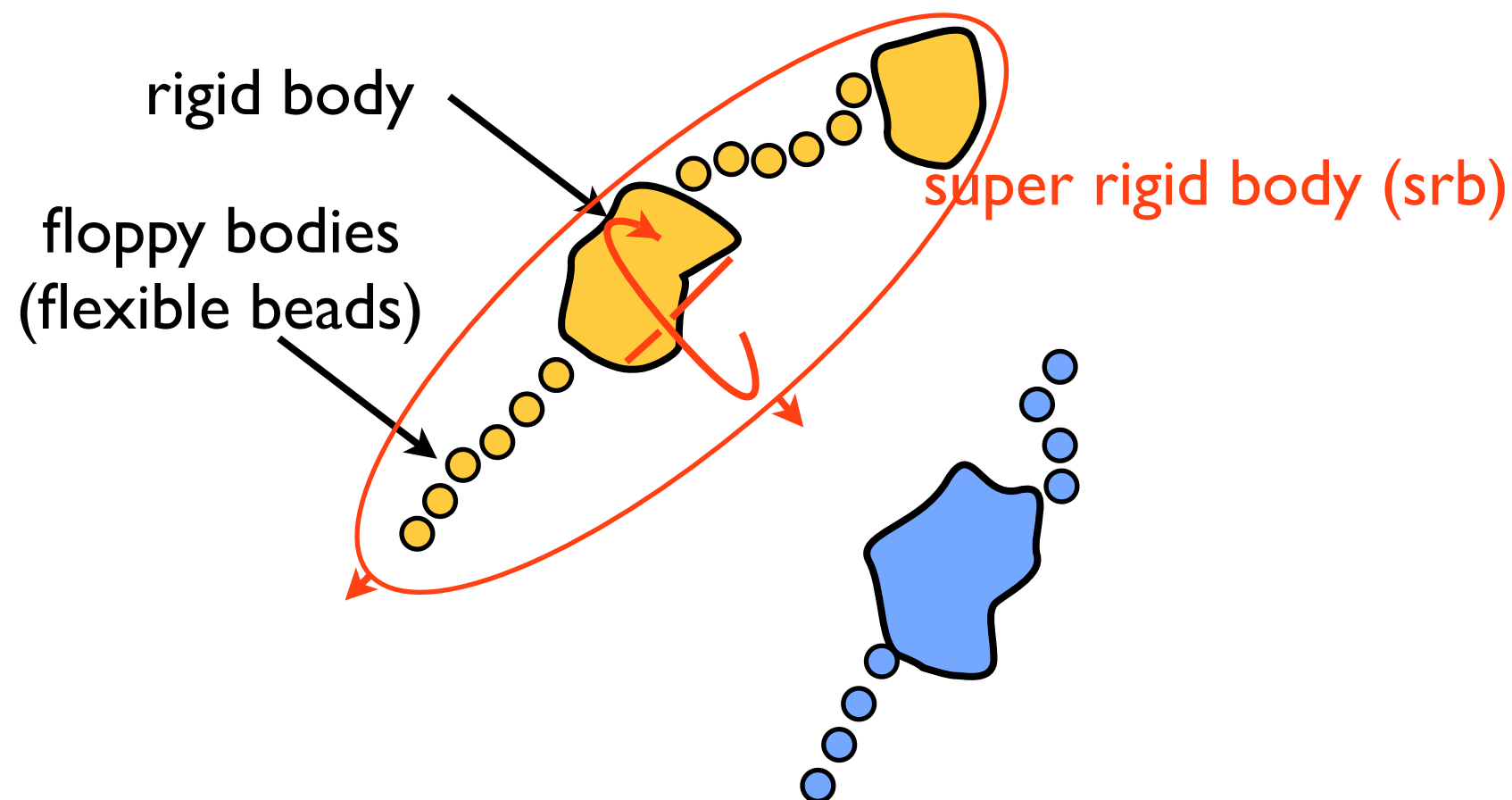
Rigid body movers

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.



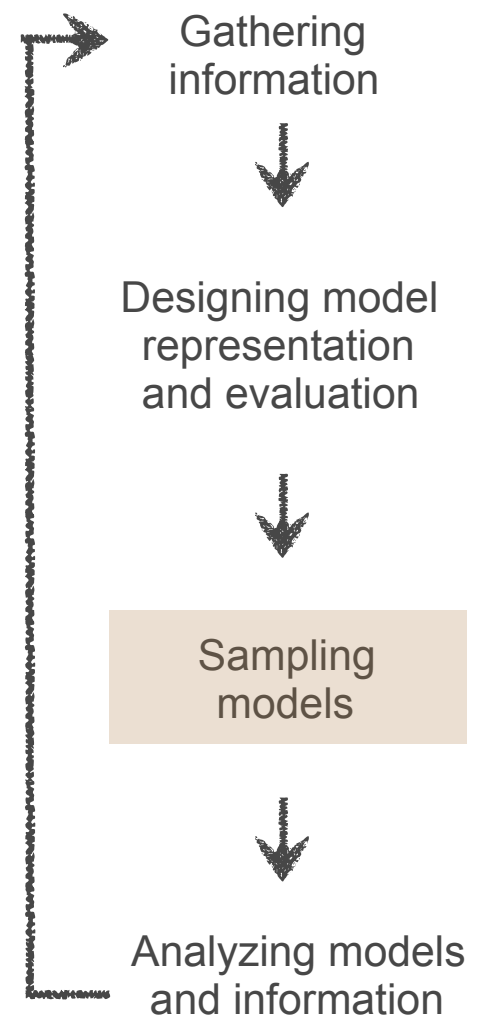
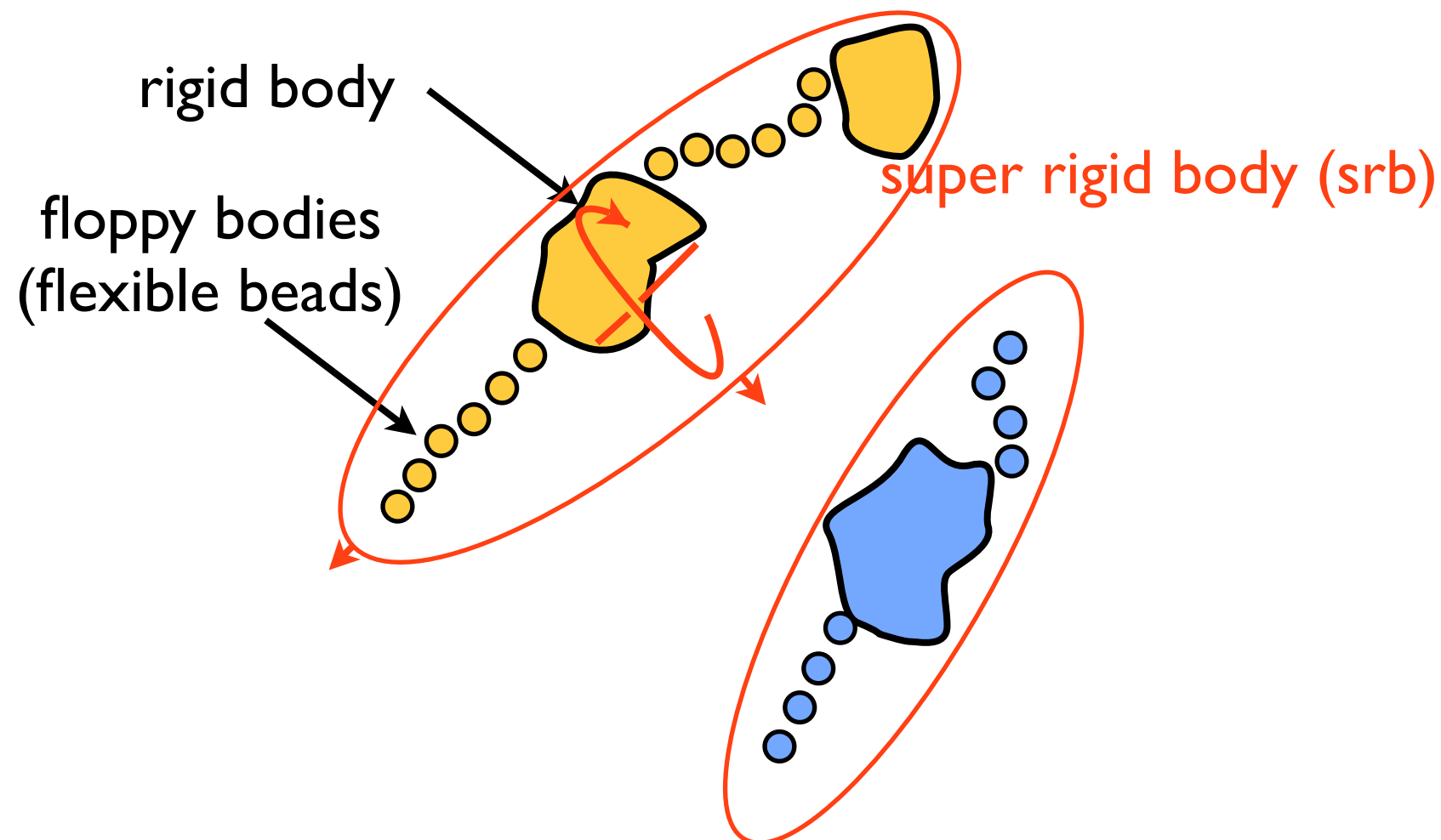
Rigid body movers

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



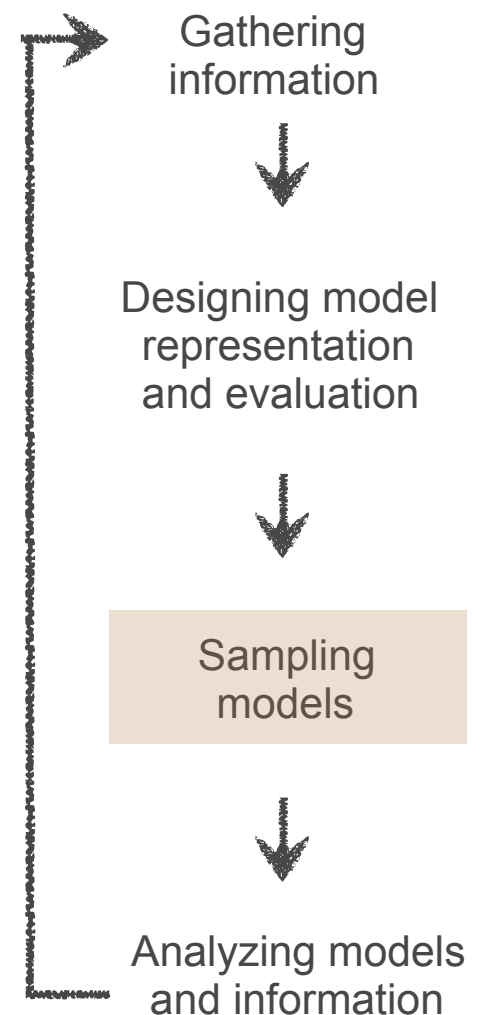
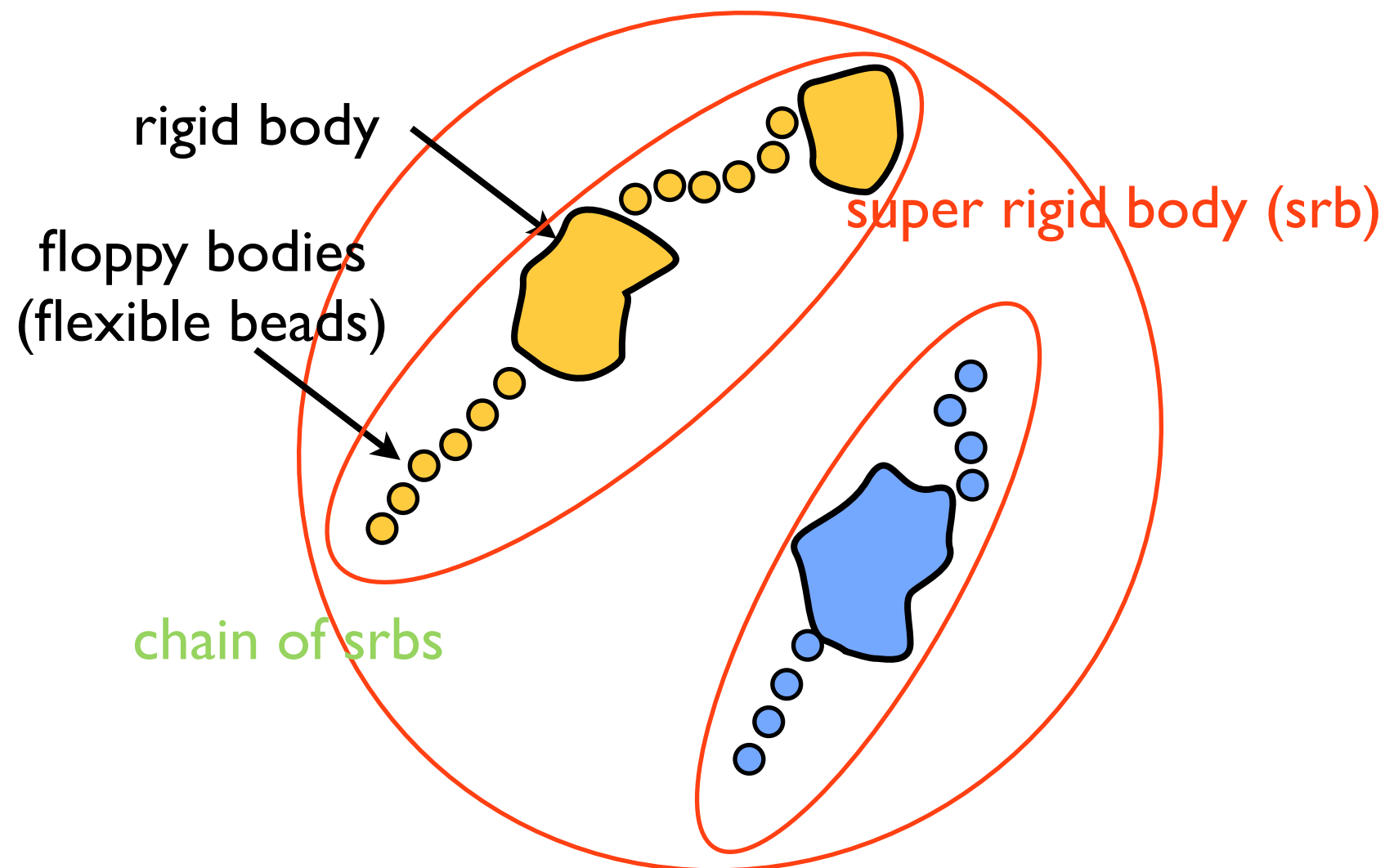
Rigid body movers

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



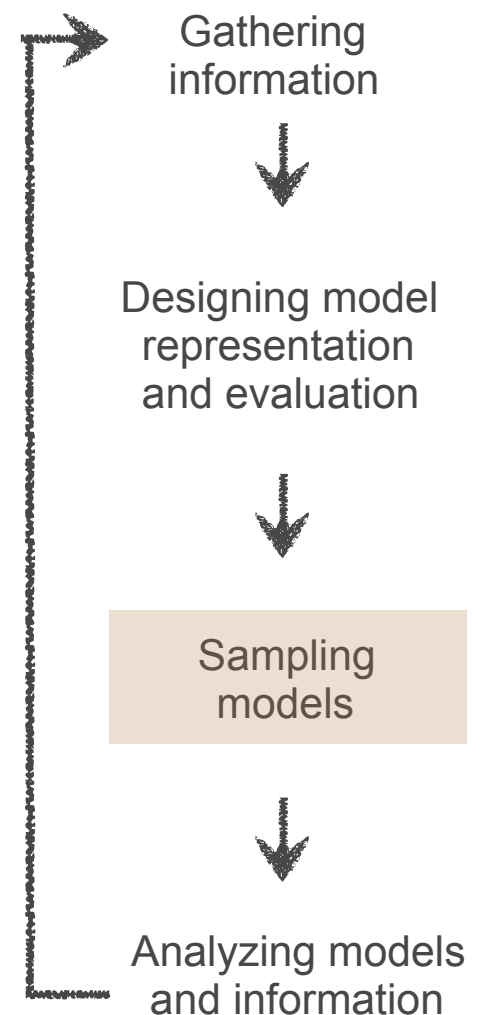
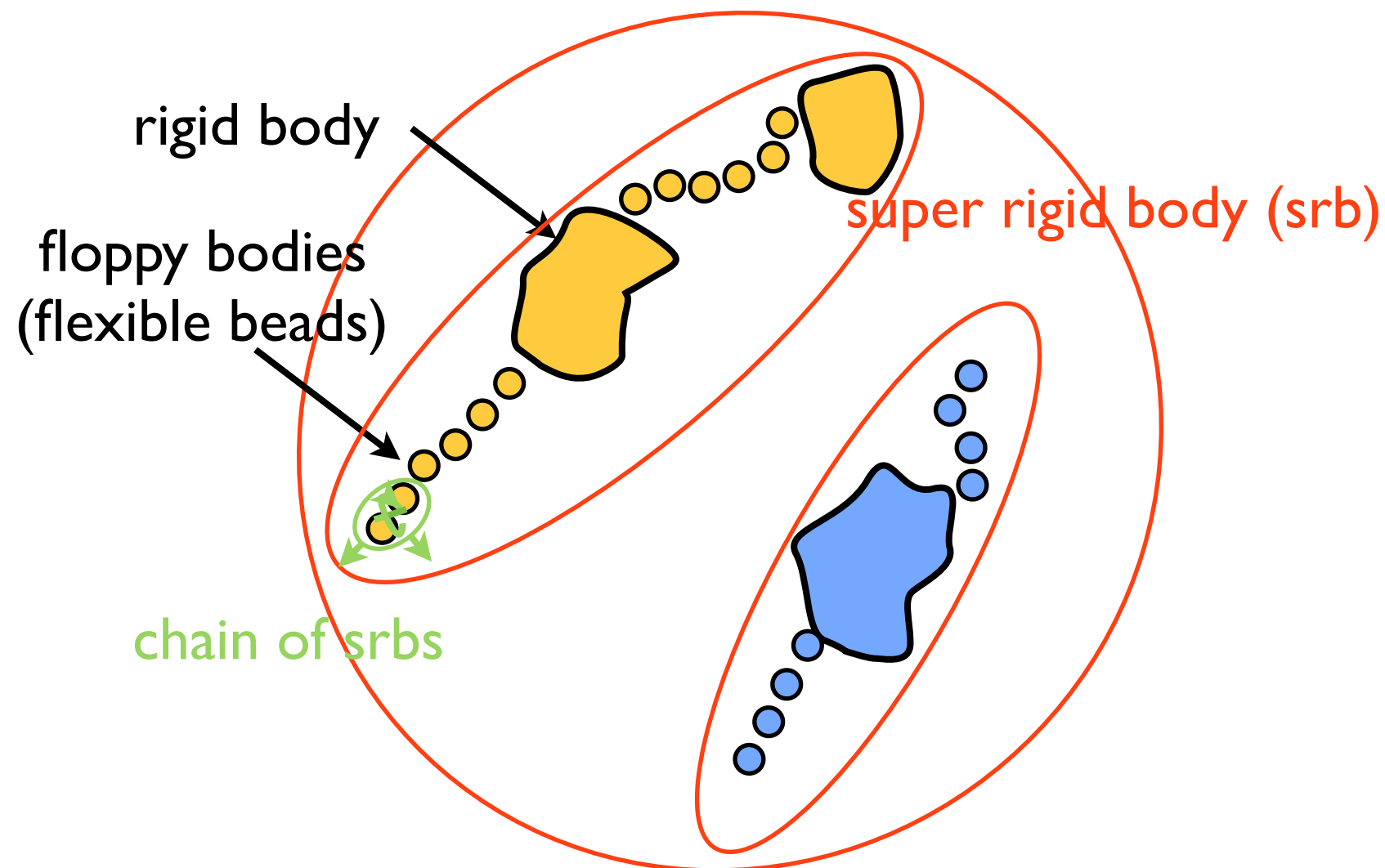
Rigid body movers

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.



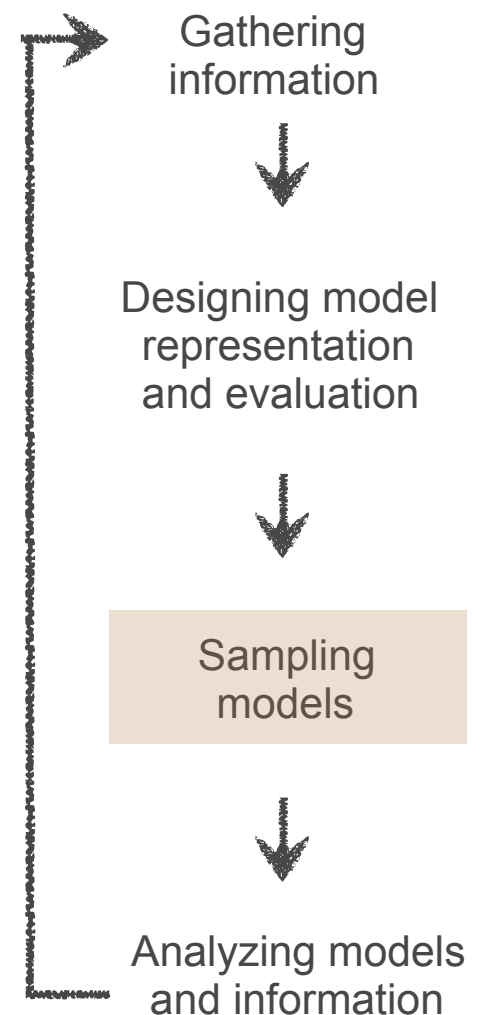
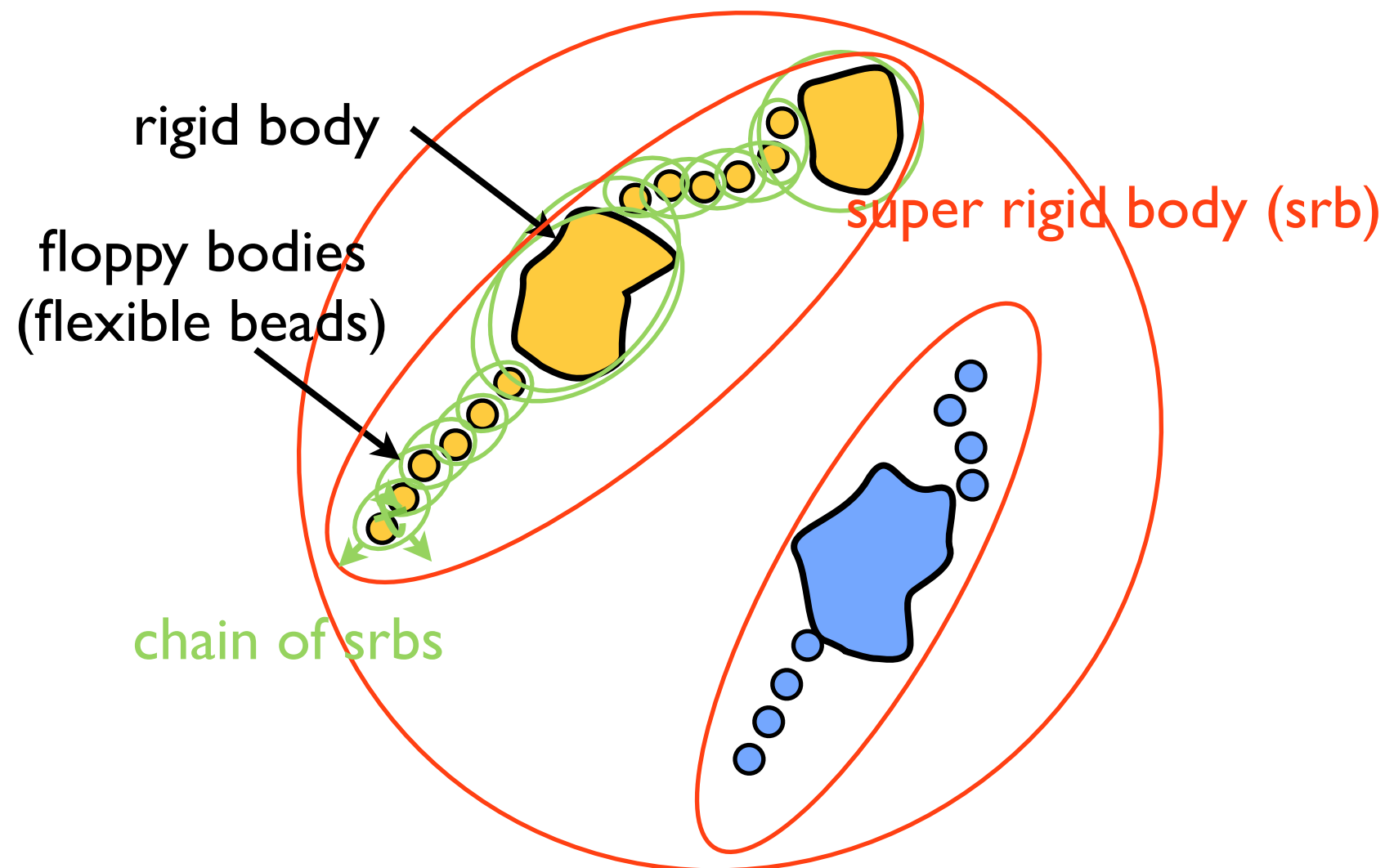
Rigid body movers

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

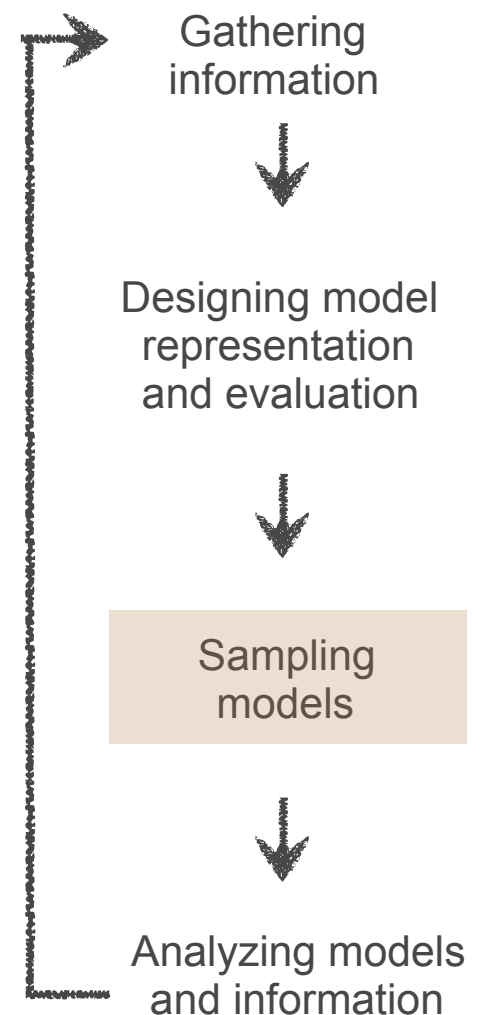
chain_of_super_rigid_bodies sets additional Monte Carlo movers along the connectivity chain of a subunit. It groups sequence-connected rigid domains and/or beads into overlapping pairs and triplets. Each of these groups will be moved rigidly. This mover helps to sample more efficiently complex topologies,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



Sampling

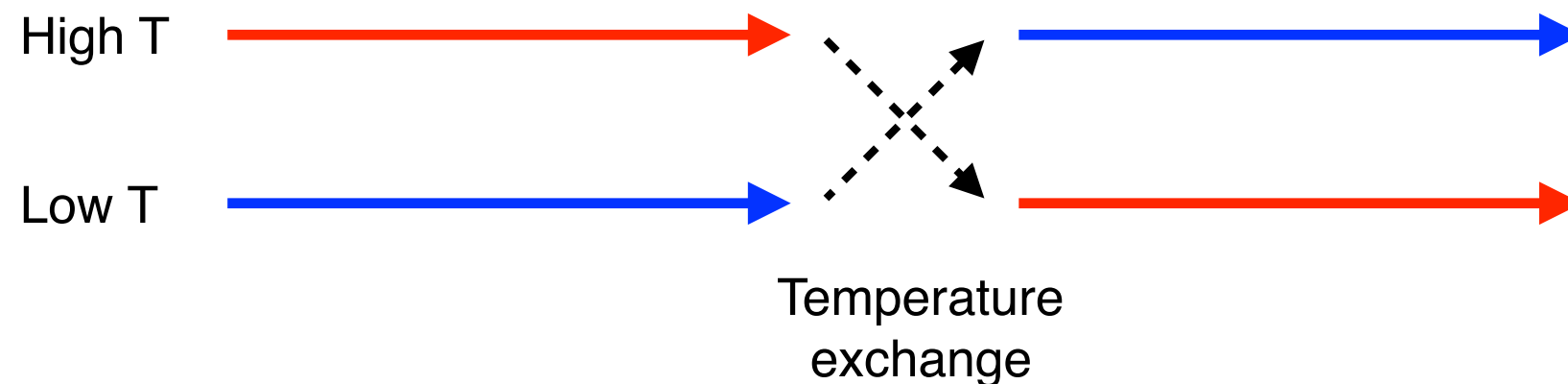
- Finally, we run the Monte Carlo sampling itself
- Technically this is replica exchange (as the class name suggests)
- **crosslink_restraints** ensures that our cross-links are added to output models (for visualization)
- Note that we also move non-Cartesian parameters for our Bayesian restraints, as per **sampleobjects**

```
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")
```

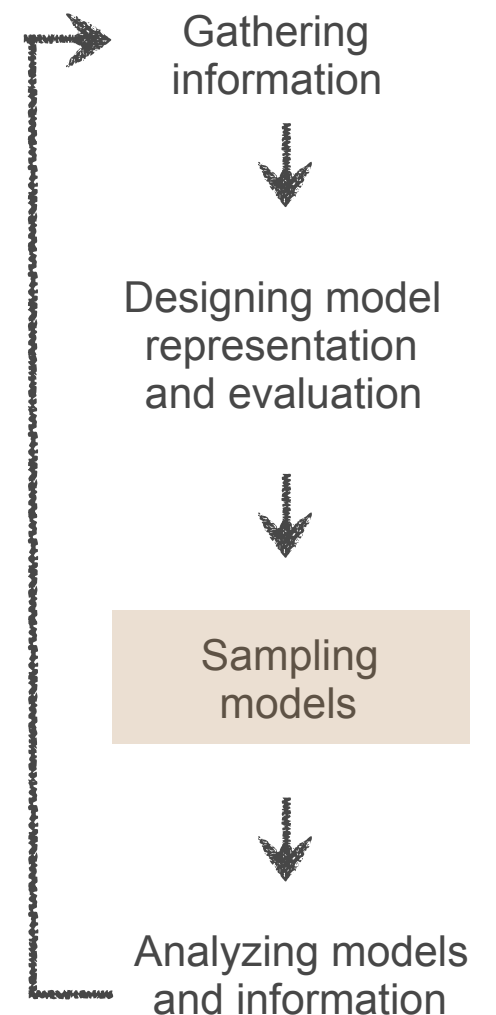


Replica exchange

- Multiple simulations run in parallel, at different temperatures
- Periodically, coordinates/temperatures may be swapped
- Helps to overcome local minima with little communication overhead



- If IMP is built with MPI support and run with mpirun, will do replica exchange (1 replica per process)
- In this case, we are running only a single process so no exchange occurs (thus, equivalent to regular Monte Carlo)



Script output

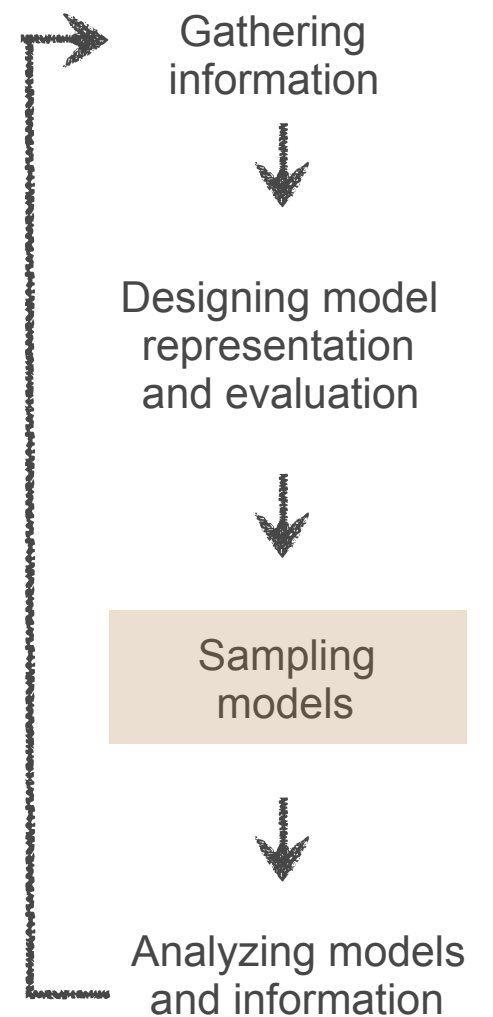
```
$ python modeling.py --test
autobuild_model: constructing Rpb1 from pdb ../data/./1WCM_map_fitted.pdb and chain A
autobuild_model: constructing fragment (1, 1) as a bead
autobuild_model: constructing fragment (2, 186) from pdb
autobuild_model: constructing fragment (187, 194) as a bead
autobuild_model: 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

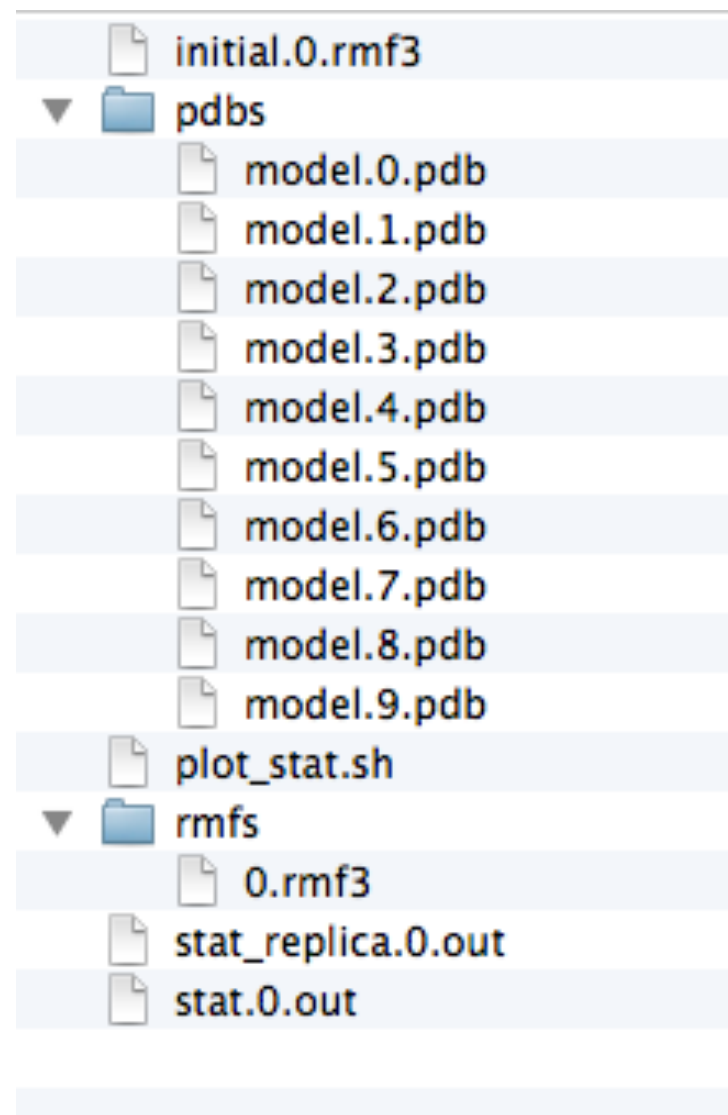
...

--- 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
--- writing coordinates
--- frame 5 score 1459614.23926
```



Output data

- A directory `output` is created, looking like:



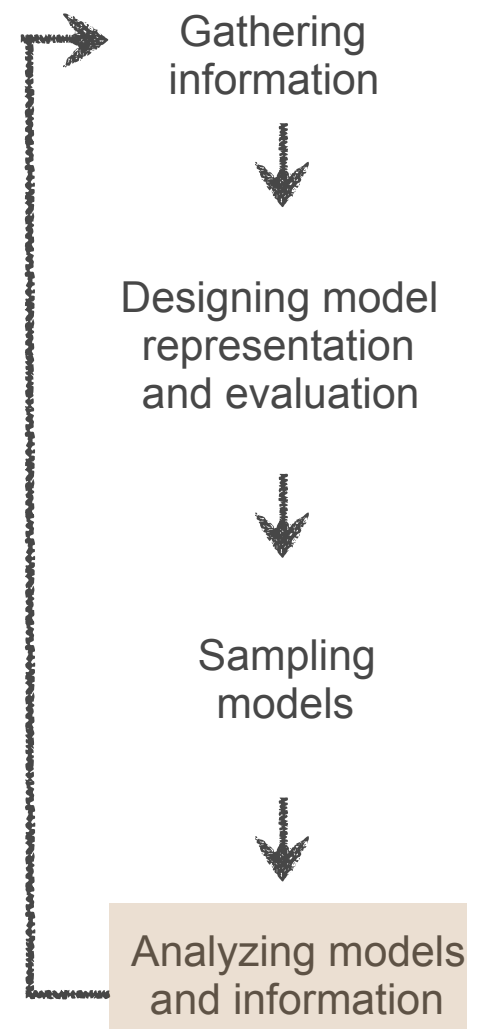
Initial structure (RMF format)

PDBs of the best scoring models (updated throughout the simulation)

the trajectory (RMF format)

replica exchange statistics

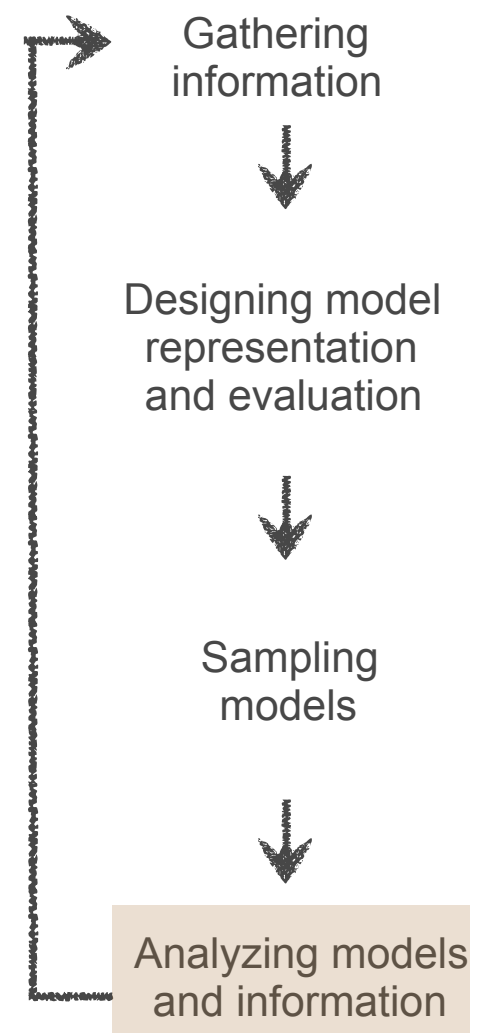
a stat file containing all useful information on `outputobjects`



RMF file format

<https://integrativemodeling.org/rmf/>

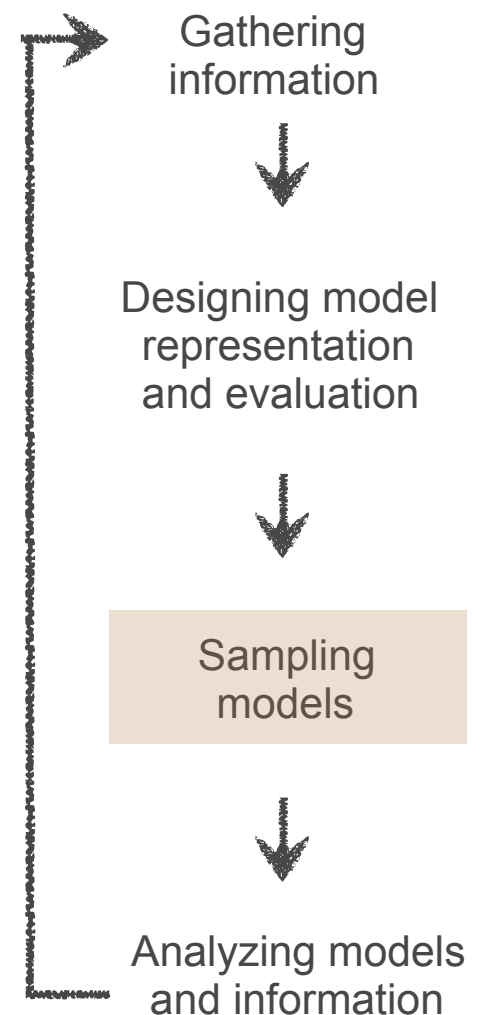
- Clear to see that PDB is not well suited for non-atomic structures
- So, IMP uses RMF format files for coarse-grained structures
 - File format designed to store hierarchical molecular data
 - Binary, so efficient for storage of trajectories
 - Not limited to traditional atom-residue-chain relationships; can store arbitrary hierarchies, multiple states, and coarse-grained models
 - Can also store non-Cartesian data, such as individual restraint scores, cross-link Bayesian parameters
- Drawback: limited viewer support (basically just Chimera)
- PDB's next generation file format (mmCIF) will natively support this class of structure (including but not limited to IMP structures)



Modeling statistics

- `output/stat.0.out` is a simple text format file containing modeling statistics
- `output/pmi_plot_stat.py` can make simple plots, or it's easy to parse yourself
- e.g. can plot the EM score (GaussianEMRestraint_None) as a function of time:

```
$ python pmi_plot_stat.py  
  -y GaussianEMRestraint_None stat.0.out
```

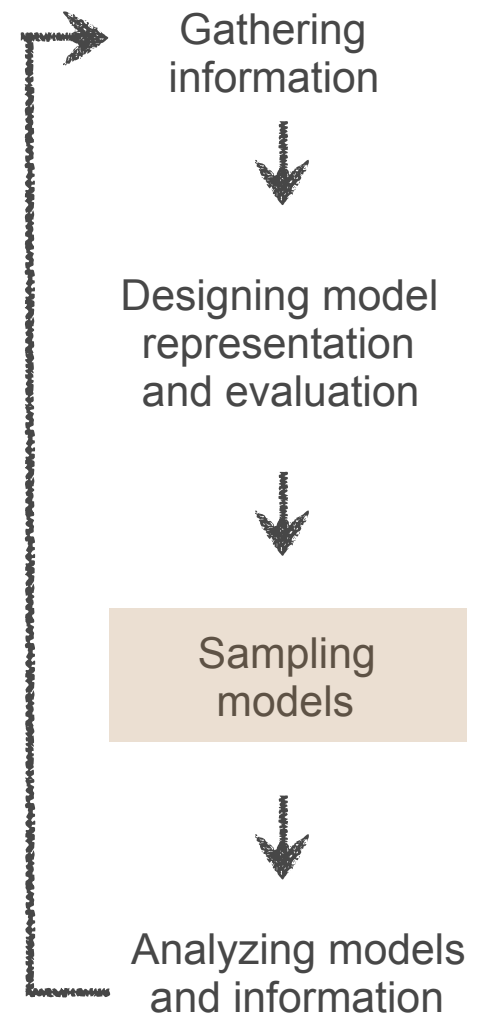
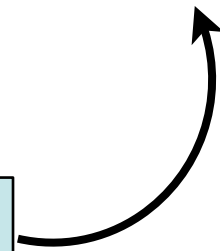


Modeling statistics

- `output/stat.0.out` is a simple text format file containing modeling statistics
- `output/pmi_plot_stat.py` can make simple plots, or it's easy to parse yourself
- e.g. can plot the EM score (GaussianEMRestraint_None) as a function of time:

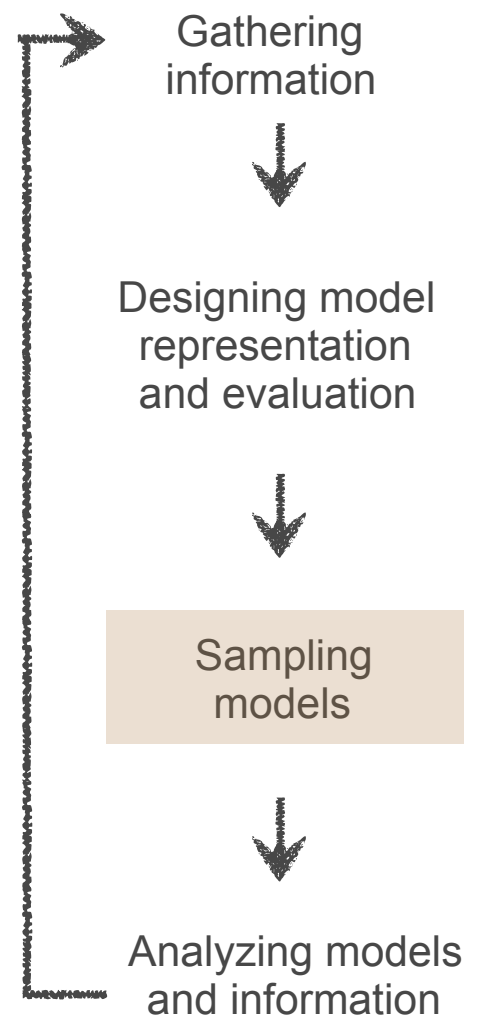
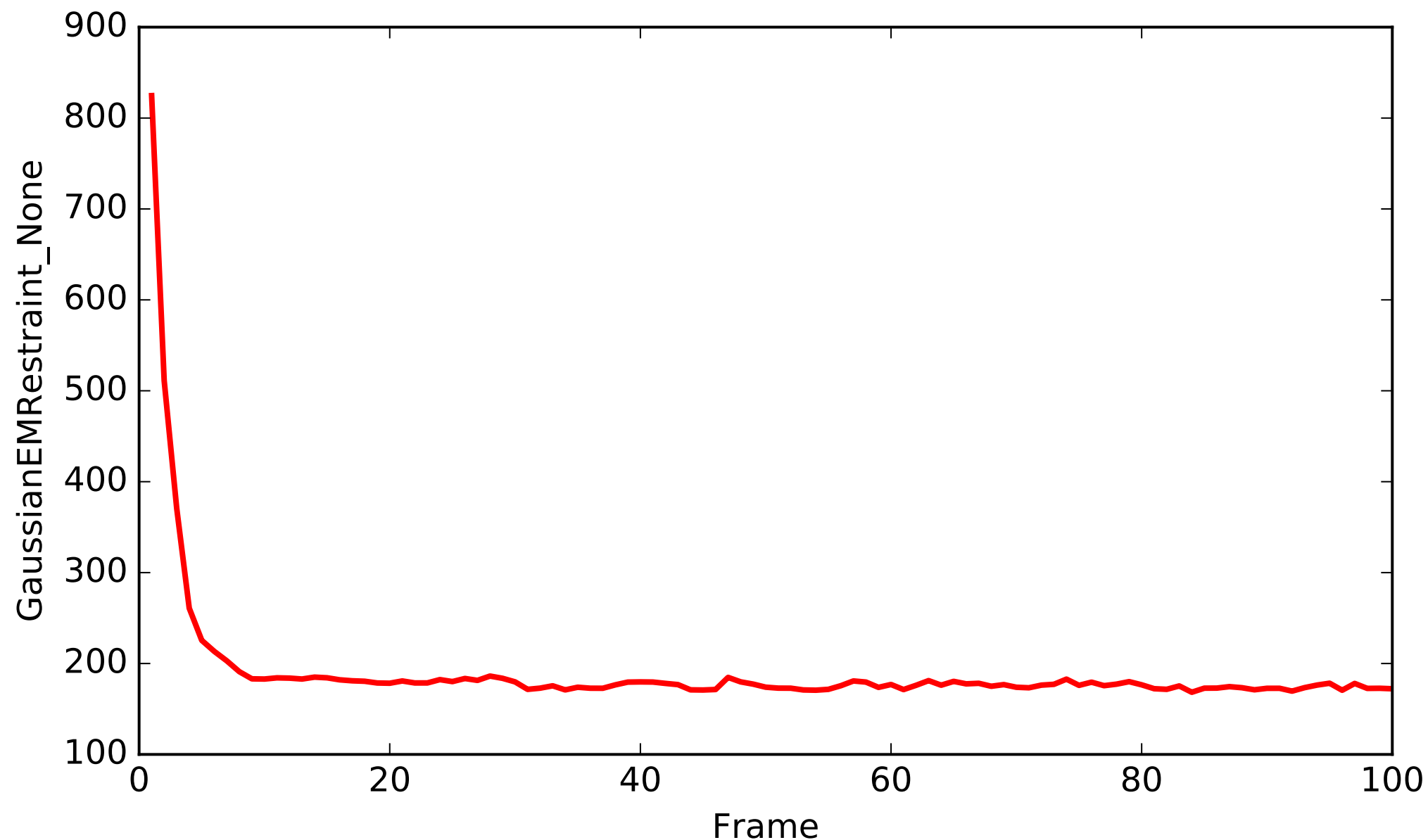
```
$ python pmi_plot_stat.py  
  -y GaussianEMRestraint_None stat.0.out
```

This should all be on one line



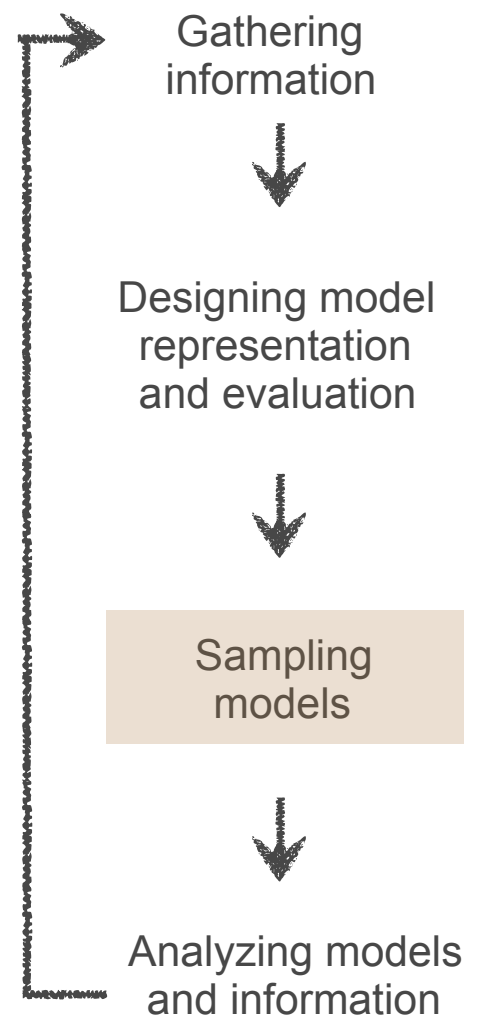
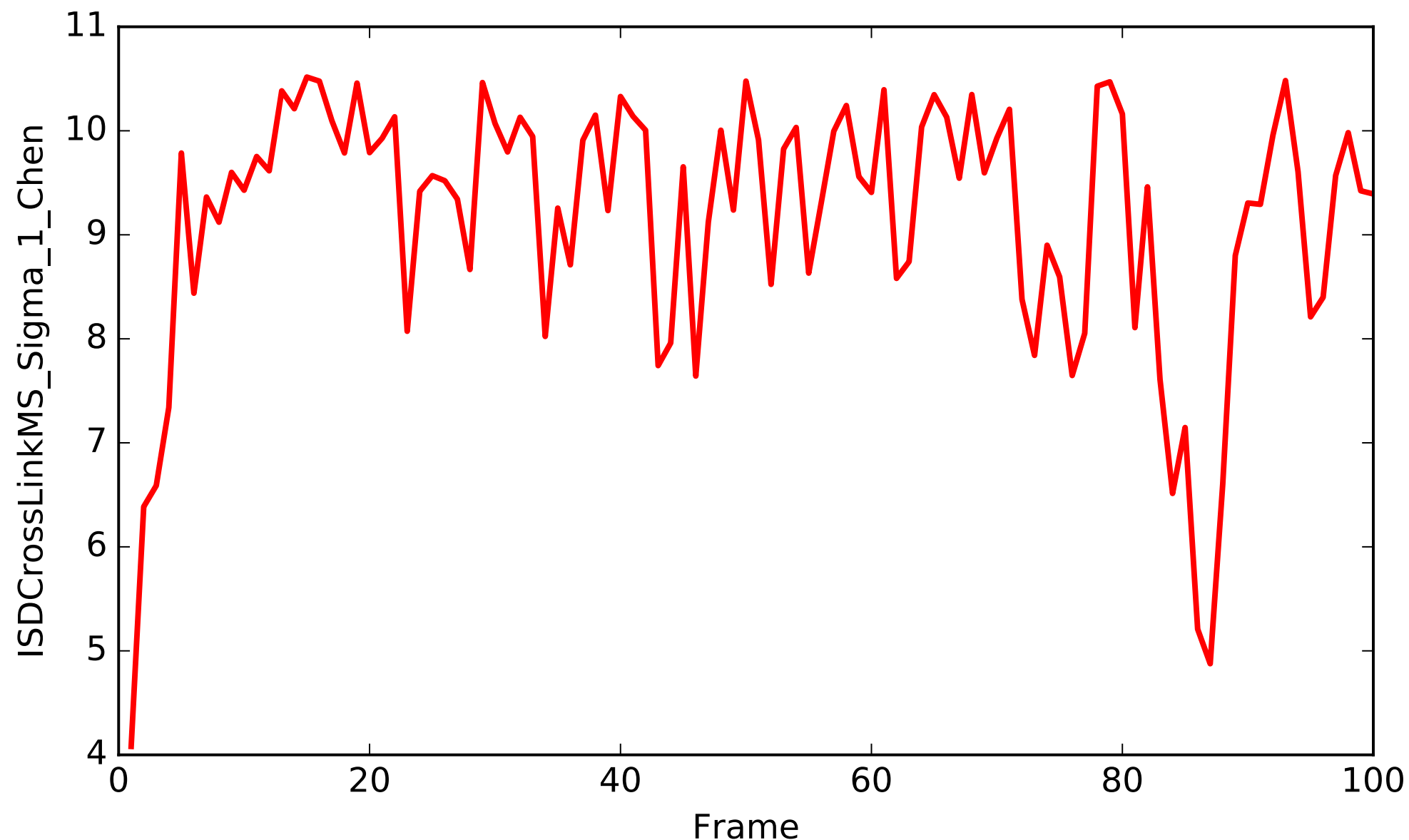
Example plots

- As expected, the EM score drops as the simulation proceeds:



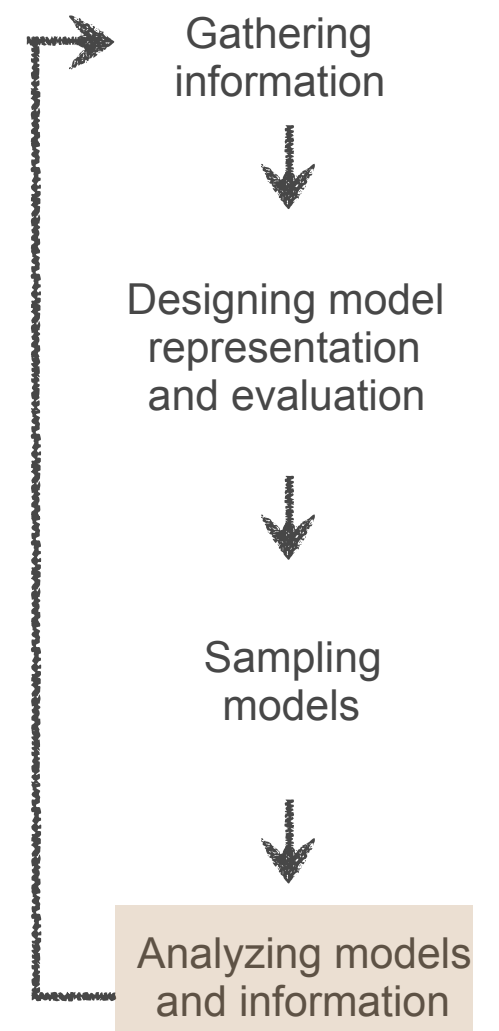
Example plots

- Bayes σ parameter ends up around 10Å (makes sense given the model resolution)



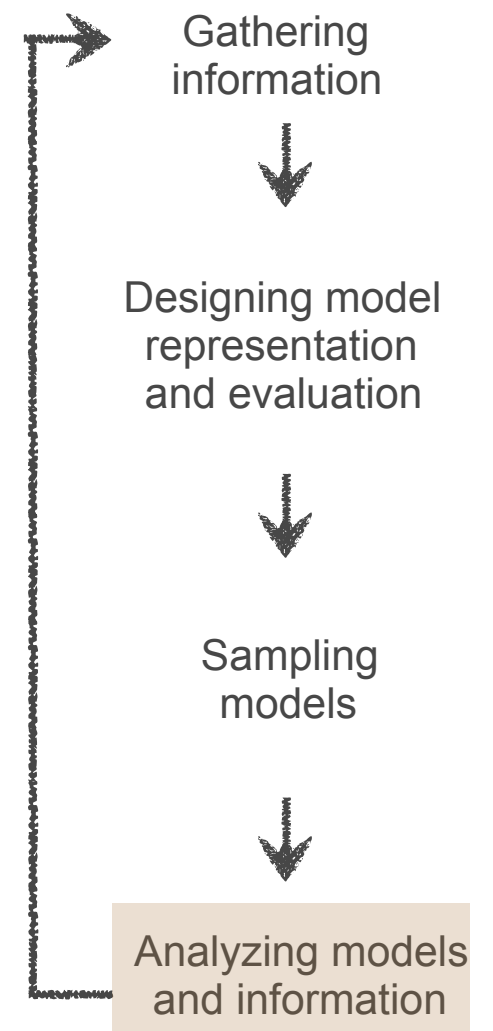
Analysis

- 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

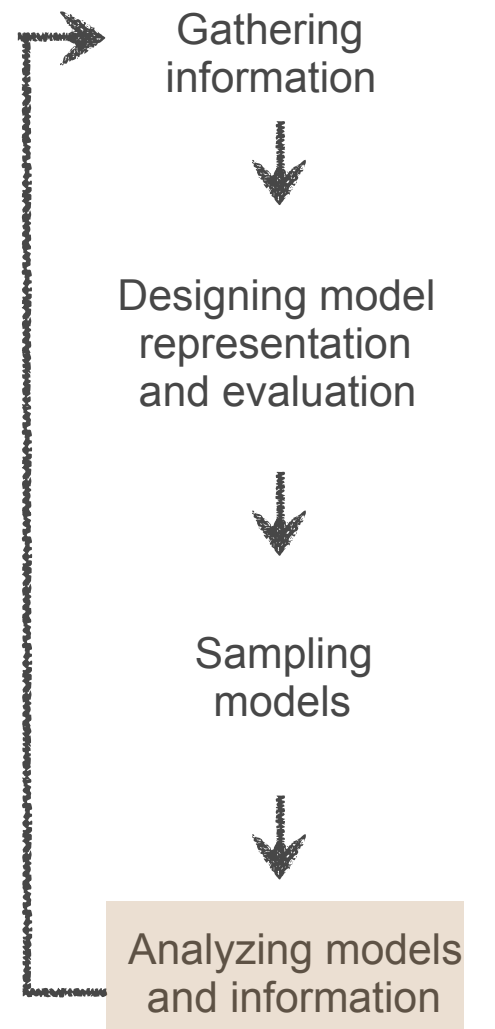
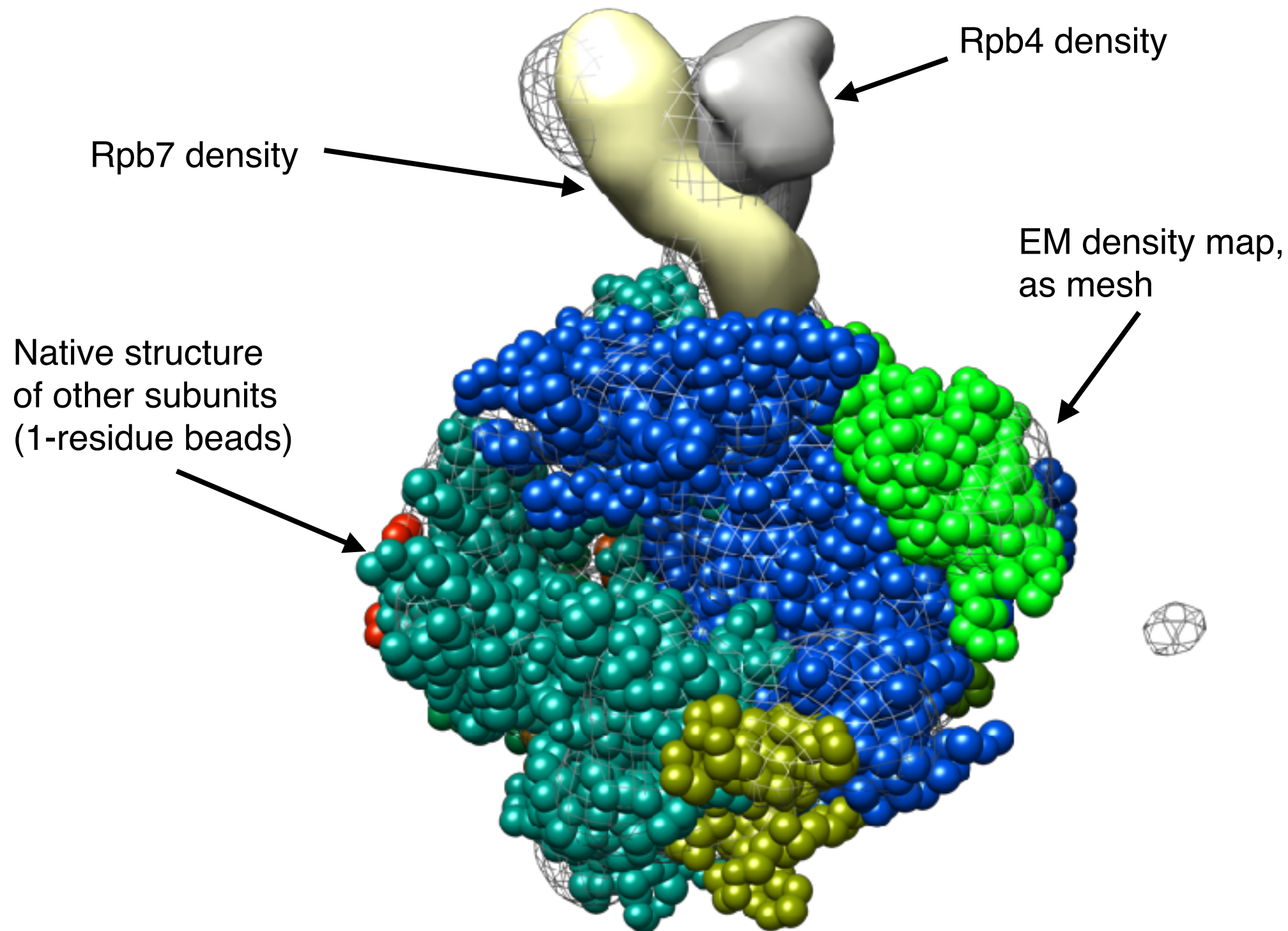


! Analysis

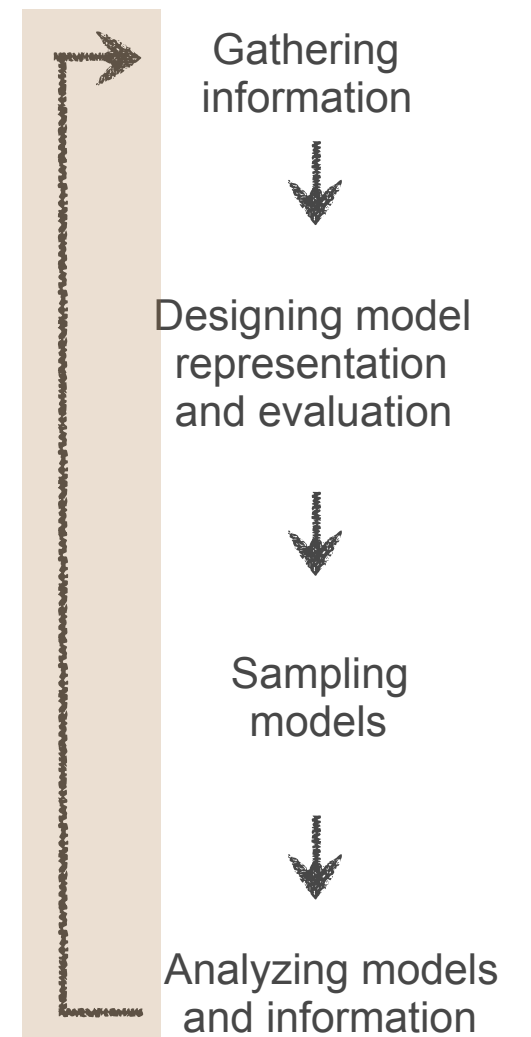
- Won't cover this in detail today
- New analysis from Shruthi not yet in public IMP, analysis in current PMI is outdated
- For more info
 - See S. Viswanath, I.E.Chemmama, P. Cimmermancic and A. Sali, Validating Exhaustiveness of Stochastic Sampling for Integrative Modeling of Macromolecular Structures, *Biophysical Journal* 113, 2344-2353, 2017
 - Bug Shruthi or Ilan directly



View in Chimera

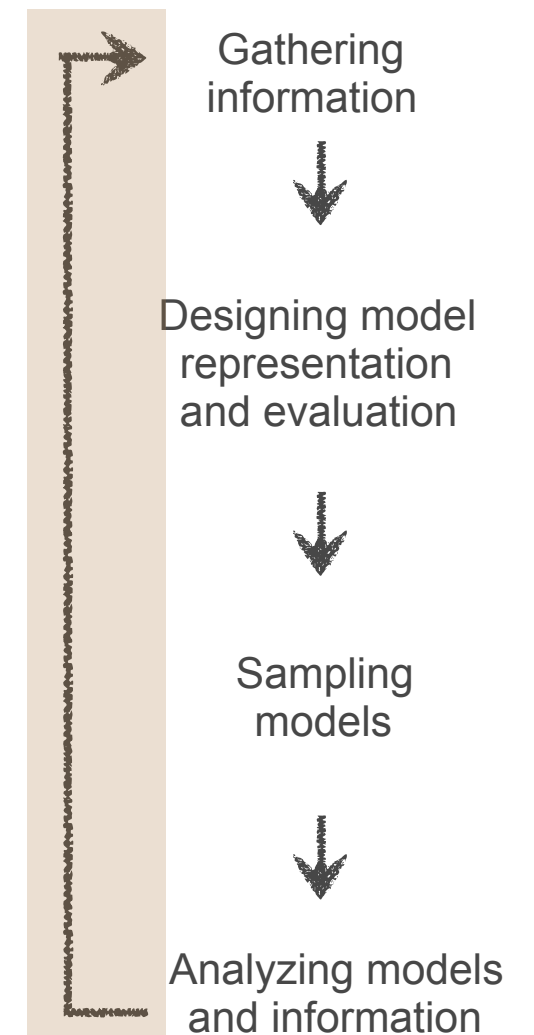


Iteration



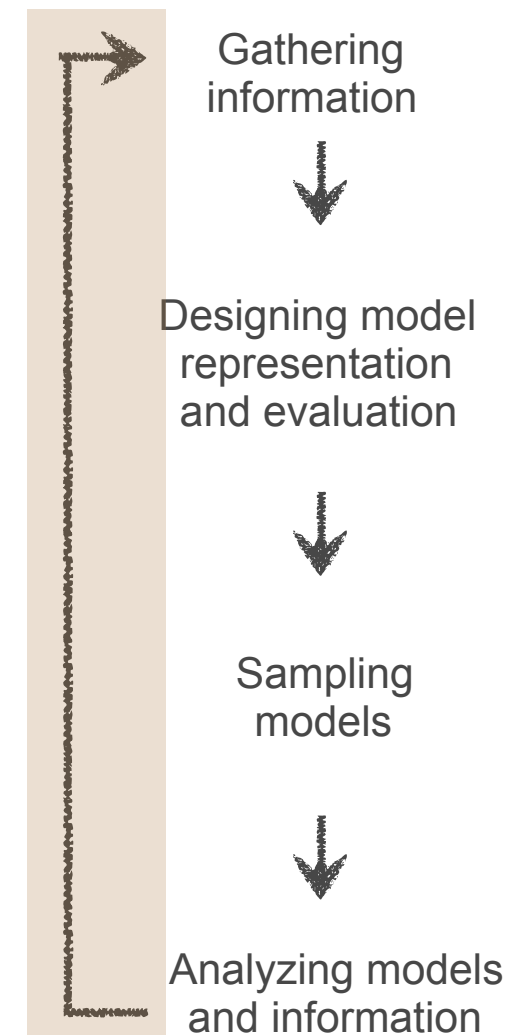
Iteration

- If necessary (or if new data become available) iteration can continue



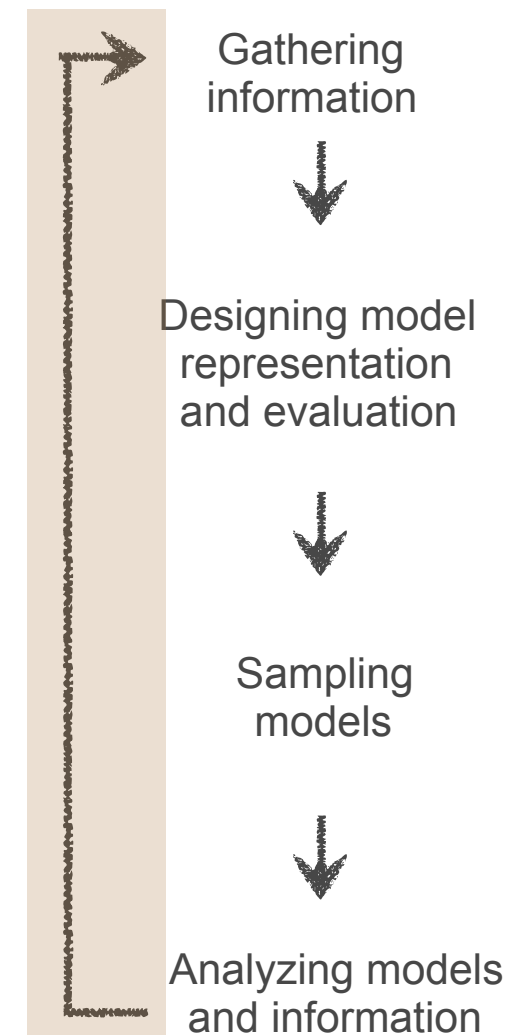
Iteration

- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)



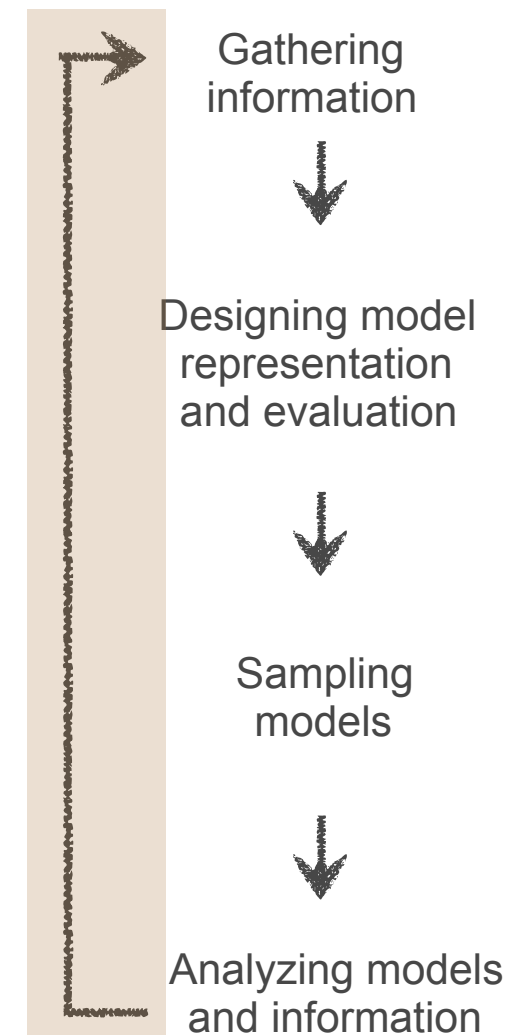
Iteration

- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)
 - by other labs (after publication)



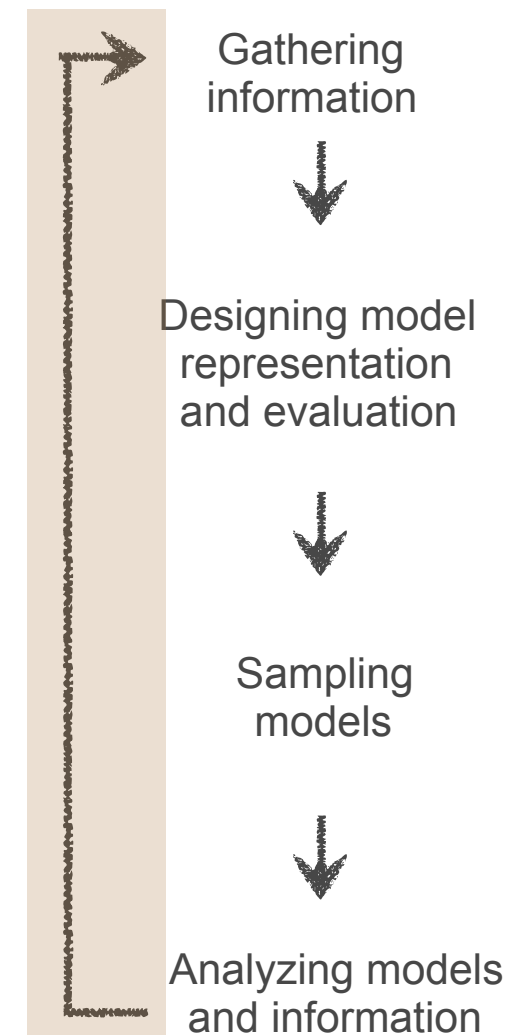
Iteration

- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)
 - by other labs (after publication)
- This can't happen unless the data are deposited somewhere in a public location



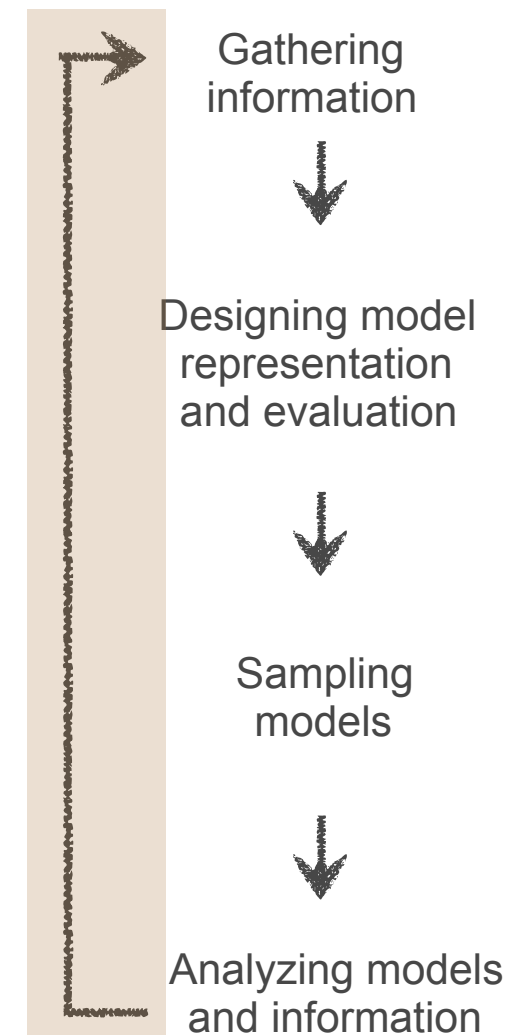
Iteration

- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)
 - by other labs (after publication)
- This can't happen unless the data are deposited somewhere in a public location
 - “Data” include inputs, the protocol, and the outputs, not just a final set of coordinates



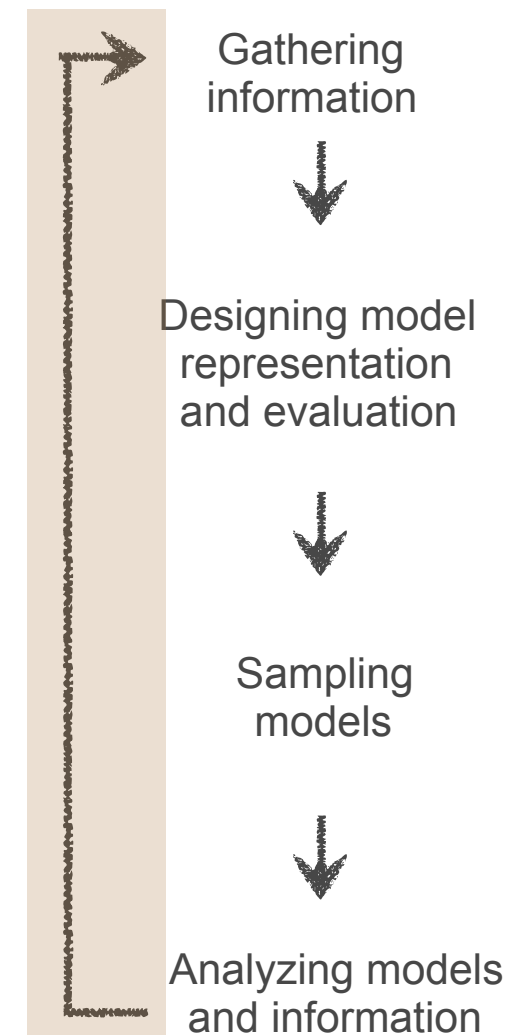
Iteration

- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)
 - by other labs (after publication)
- This can't happen unless the data are deposited somewhere in a public location
 - “Data” include inputs, the protocol, and the outputs, not just a final set of coordinates
- Thus,



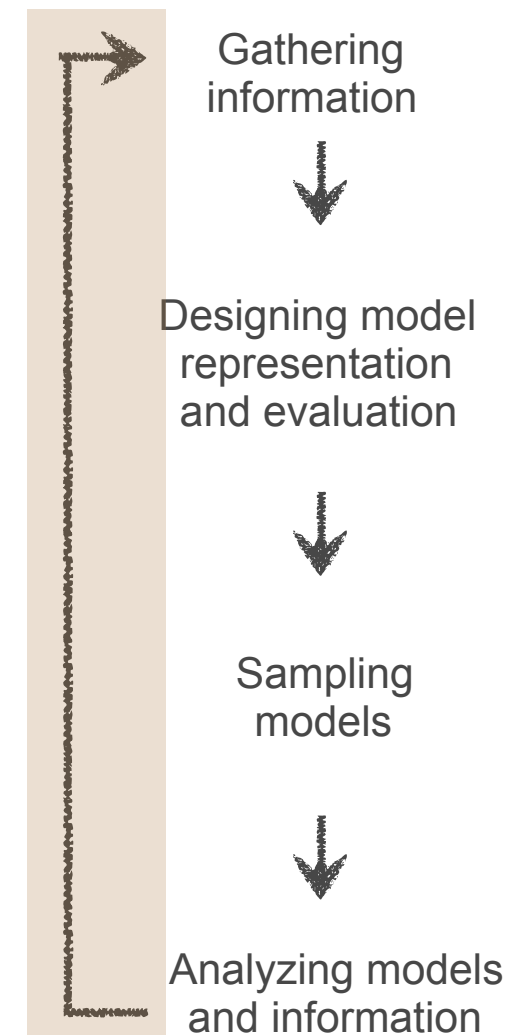
Iteration

- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)
 - by other labs (after publication)
- This can't happen unless the data are deposited somewhere in a public location
 - “Data” include inputs, the protocol, and the outputs, not just a final set of coordinates
- Thus,
 - store entire modeling protocol in a GitHub repository



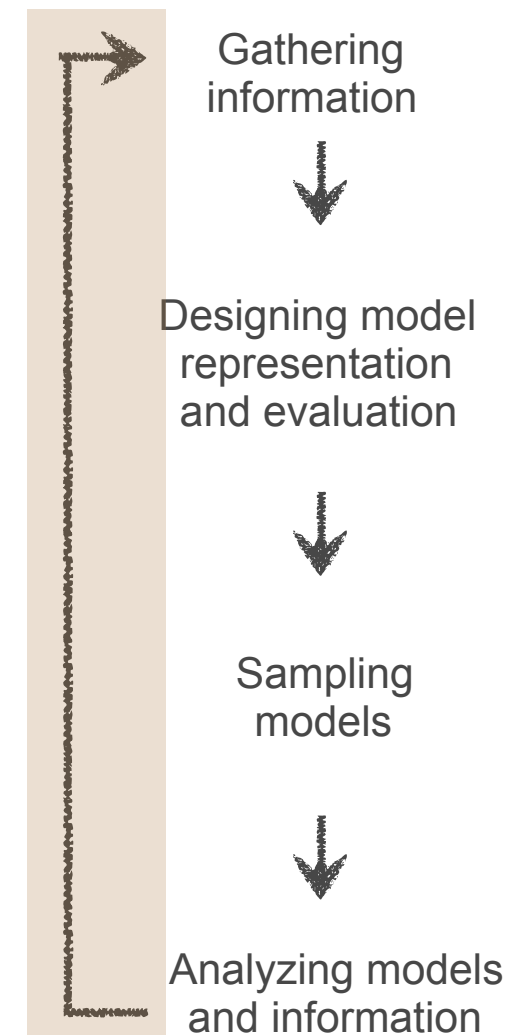
Iteration

- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)
 - by other labs (after publication)
- This can't happen unless the data are deposited somewhere in a public location
 - “Data” include inputs, the protocol, and the outputs, not just a final set of coordinates
- Thus,
 - store entire modeling protocol in a GitHub repository
 - deposit in the new PDB-Dev archive:
<https://pdb-dev.wwpdb.org/>



Iteration

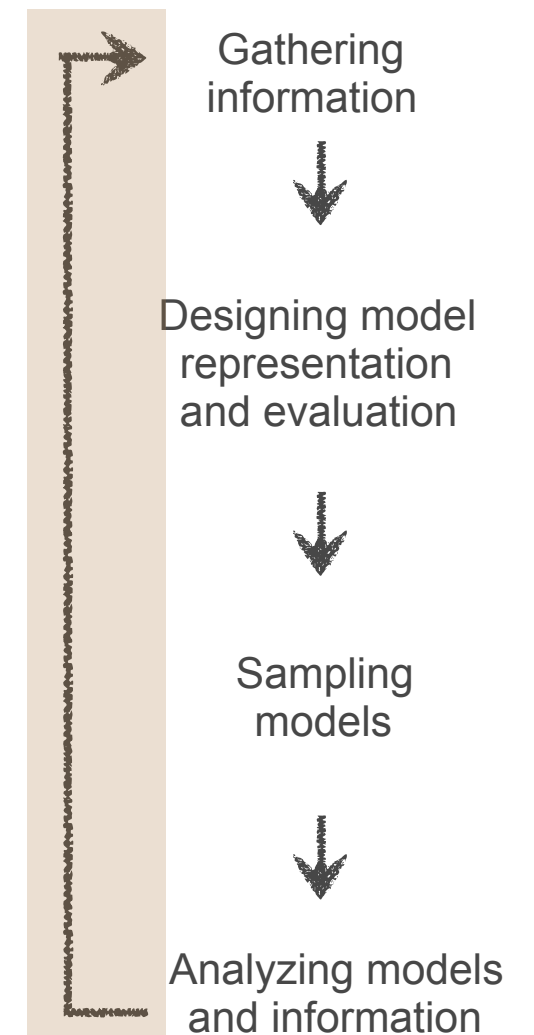
- If necessary (or if new data become available) iteration can continue
 - by us (prior to or after publication)
 - by other labs (after publication)
- This can't happen unless the data are deposited somewhere in a public location
 - “Data” include inputs, the protocol, and the outputs, not just a final set of coordinates
- Thus,
 - store entire modeling protocol in a GitHub repository
 - deposit in the new PDB-Dev archive:
<https://pdb-dev.wwpdb.org/>



<https://integrativemodeling.org/systems/>

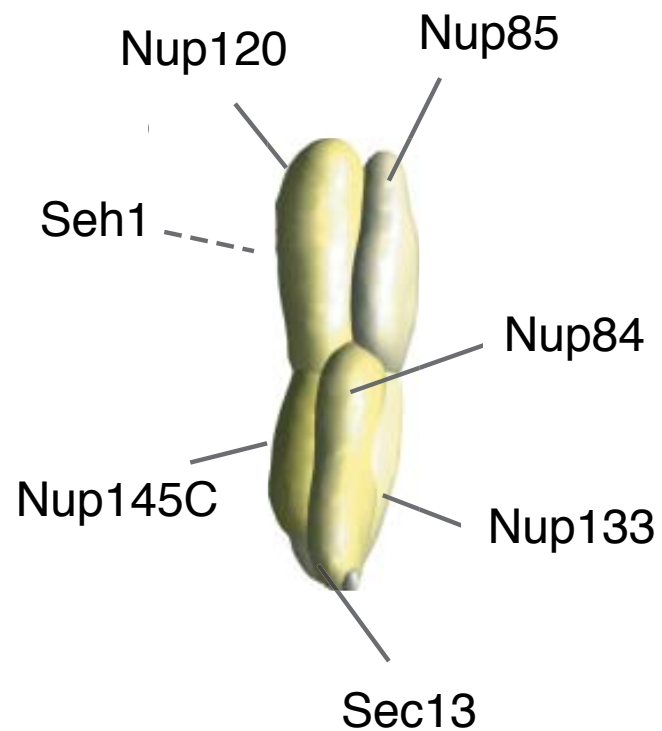
Iteration

- Models of the Nup84 sub complex of the NPC improved *via* iteration over time



Iteration

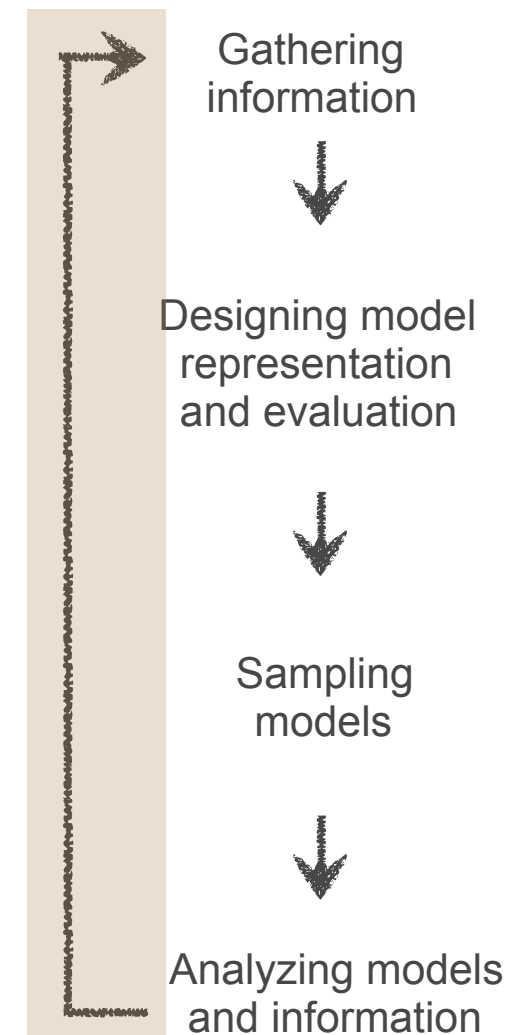
- Models of the Nup84 sub complex of the NPC improved *via* iteration over time



2007

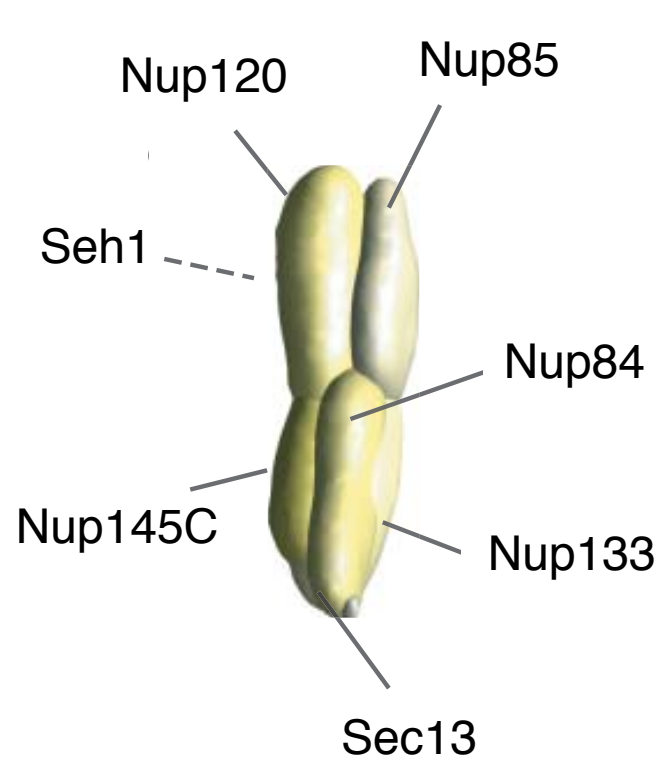
Nature 450,
684-694, 2007

Nature 450,
695-702, 2007



Iteration

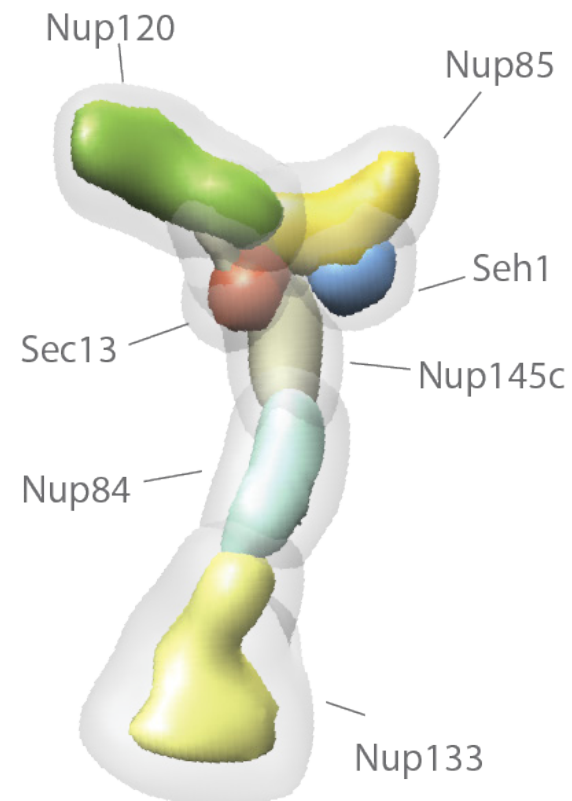
- Models of the Nup84 sub complex of the NPC improved *via* iteration over time



2007

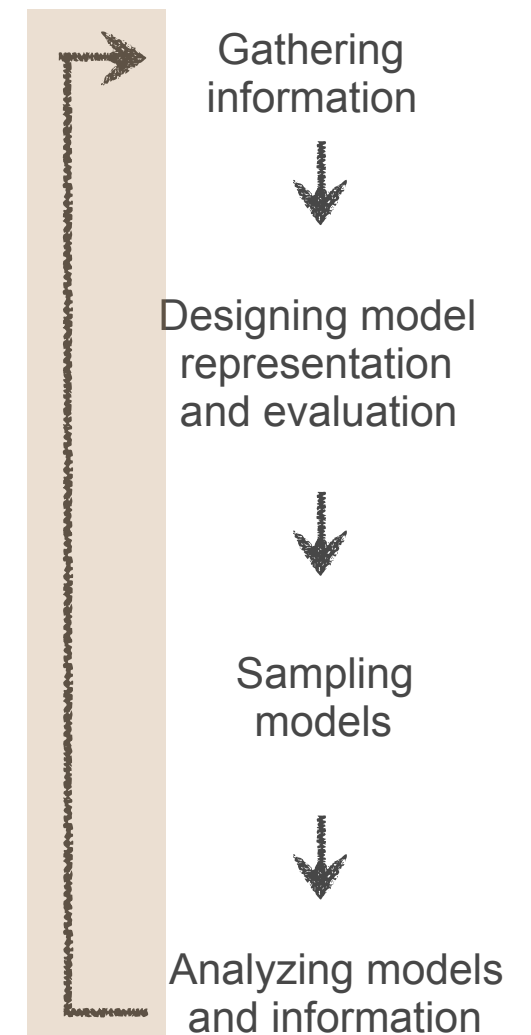
Nature 450,
684-694, 2007

Nature 450,
695-702, 2007



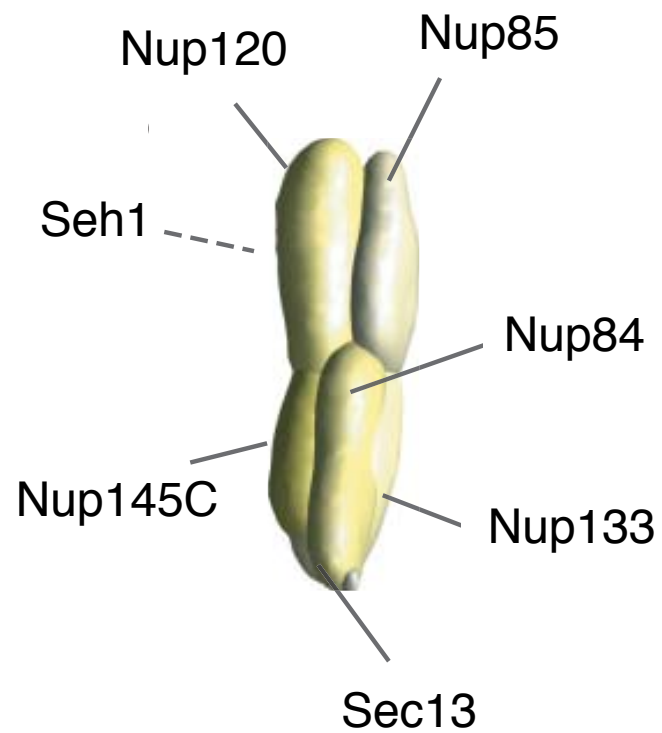
2012

J Cell Biol 196,
419-434, 2012



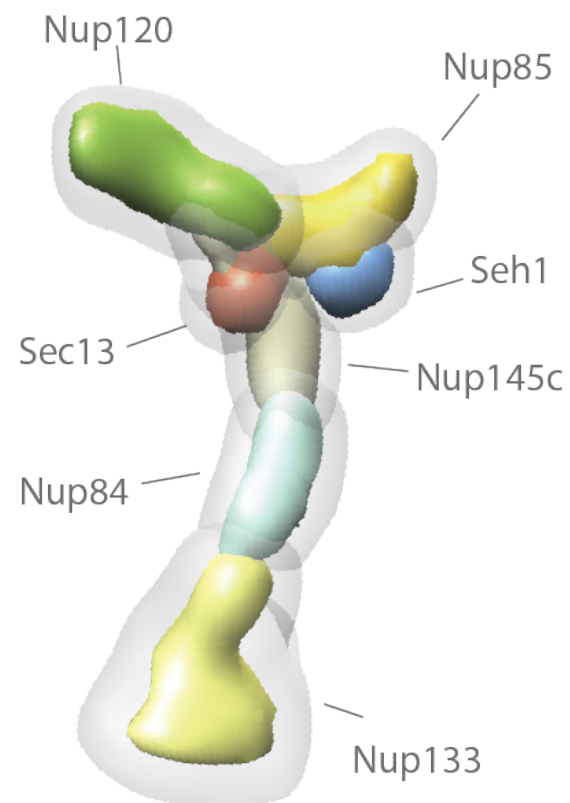
Iteration

- Models of the Nup84 sub complex of the NPC improved *via* iteration over time



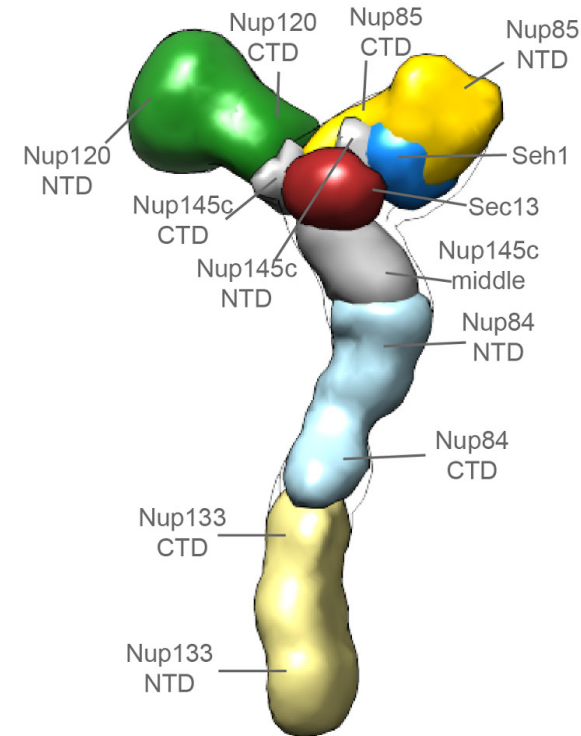
2007

Nature 450,
684-694, 2007
Nature 450,
695-702, 2007



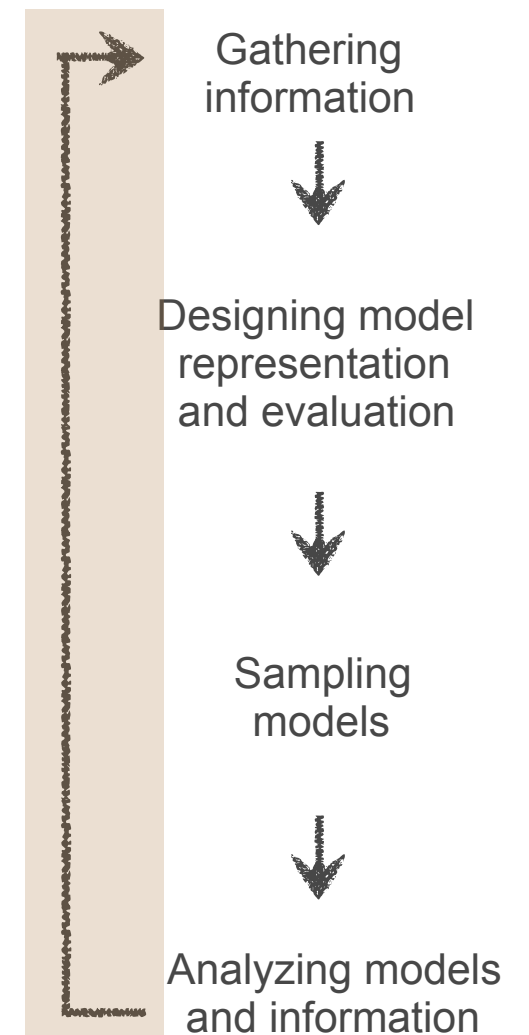
2012

J Cell Biol 196,
419-434, 2012



2014

Mol Cell
Proteomics 13,
2927-2943, 2014



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, S.J. Kim, D. Saltzberg, I. Chemmama, S. Axen, S. Viswanath.

