IMP Software Introduction & Tutorial

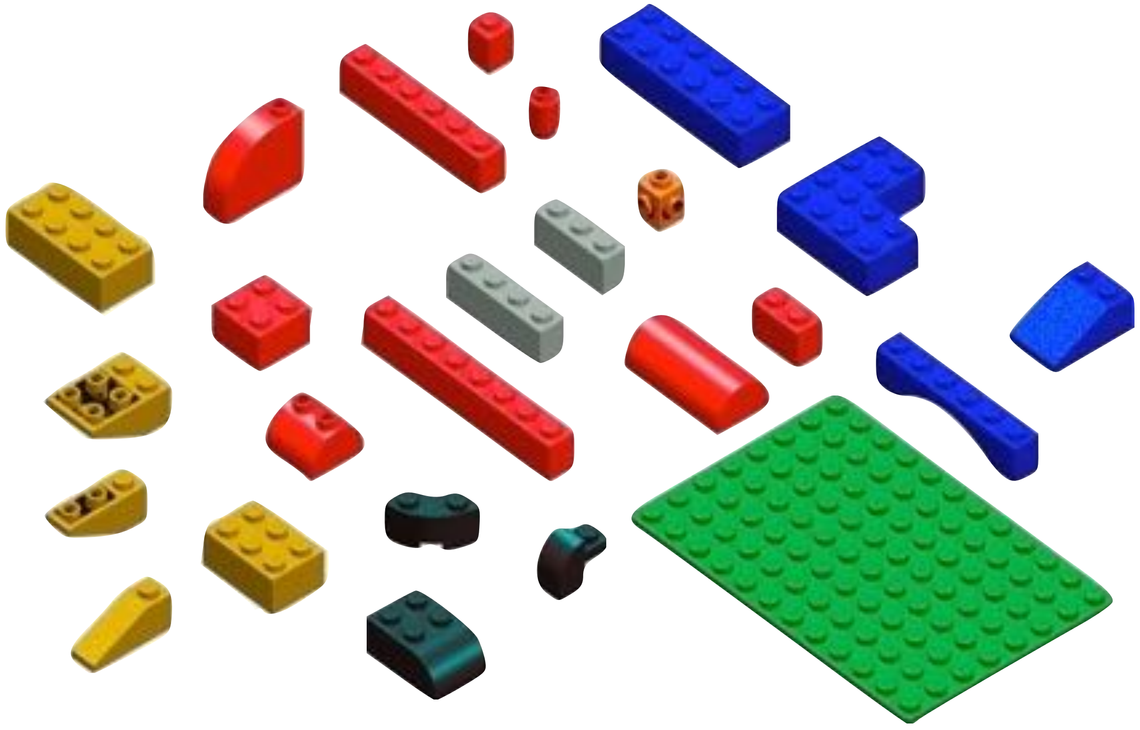
Benjamin Webb, Sali Lab
(ben@salilab.org)

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, ...)

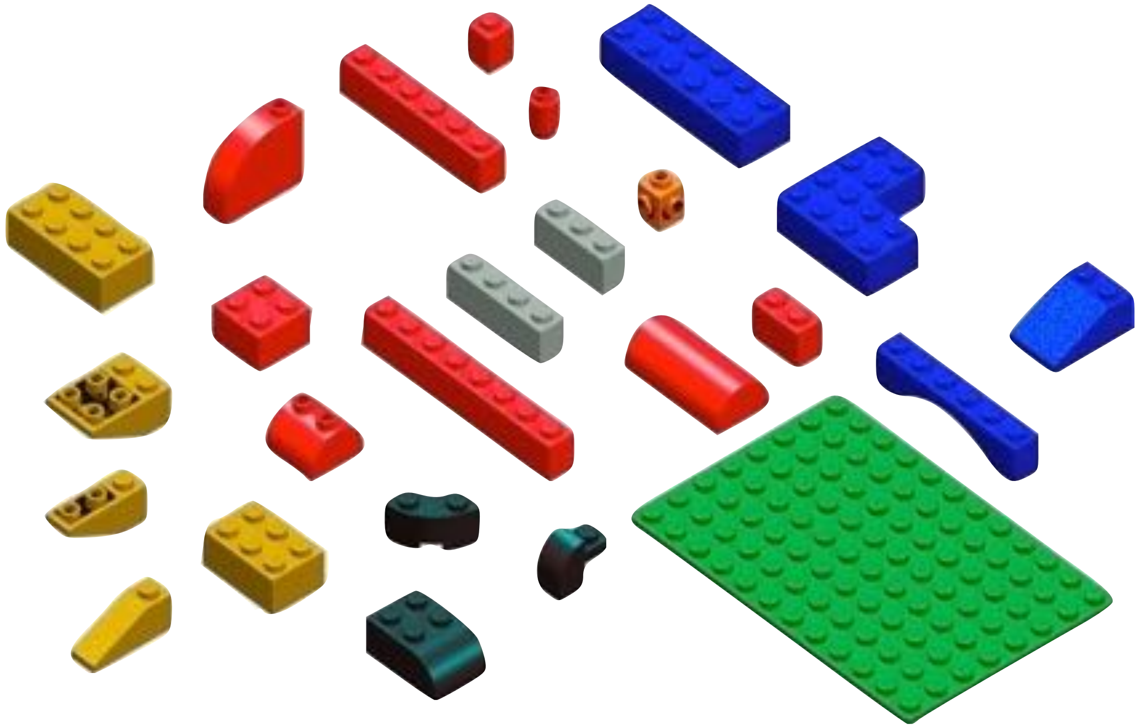
Sampling:

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

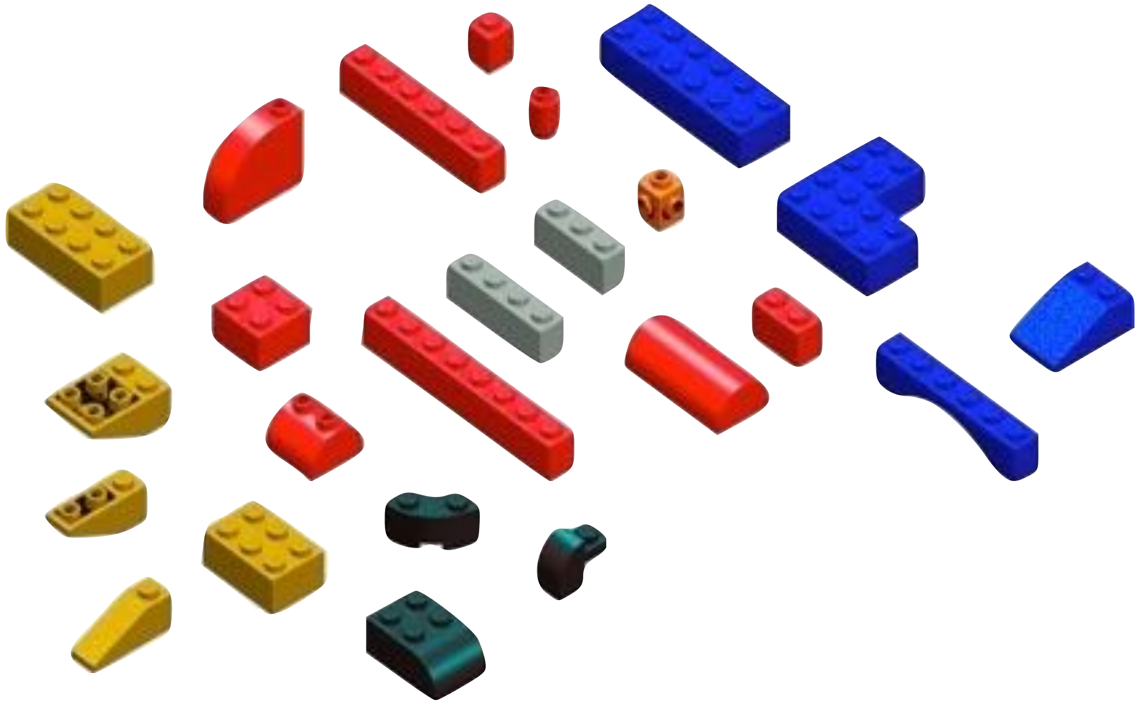
Analysis:

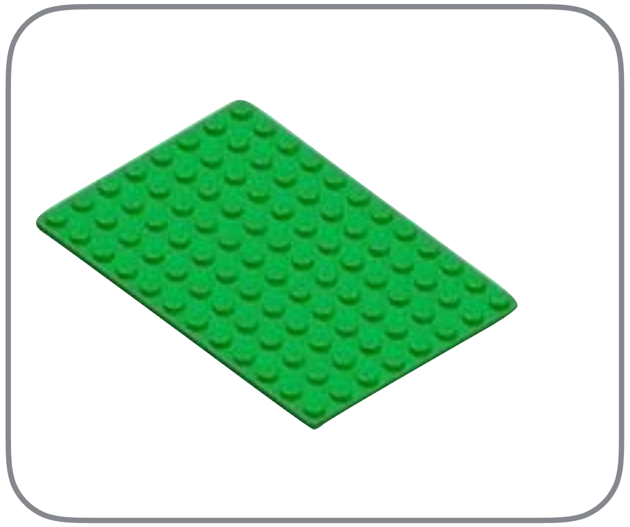
Clustering
Chimera
PyMOL
PDB files
Density maps

IMP is divided into *modules*



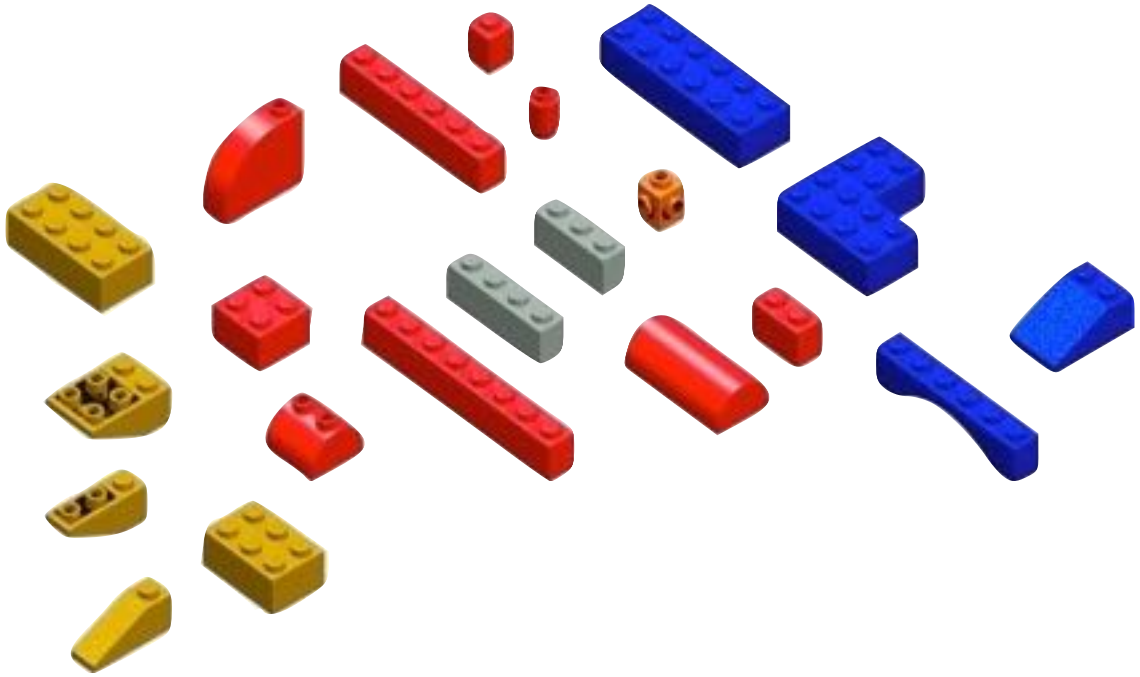
IMP is divided into *modules*

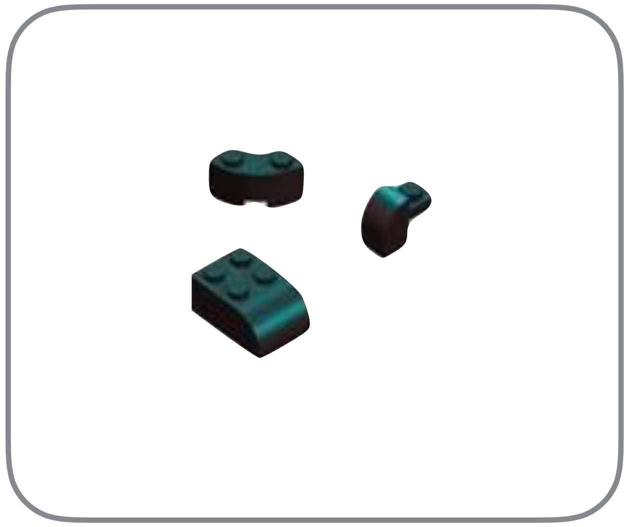




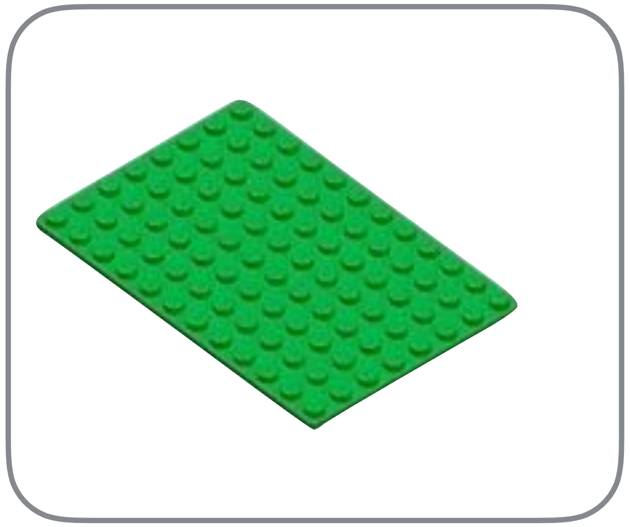
IMP kernel

IMP is divided into *modules*



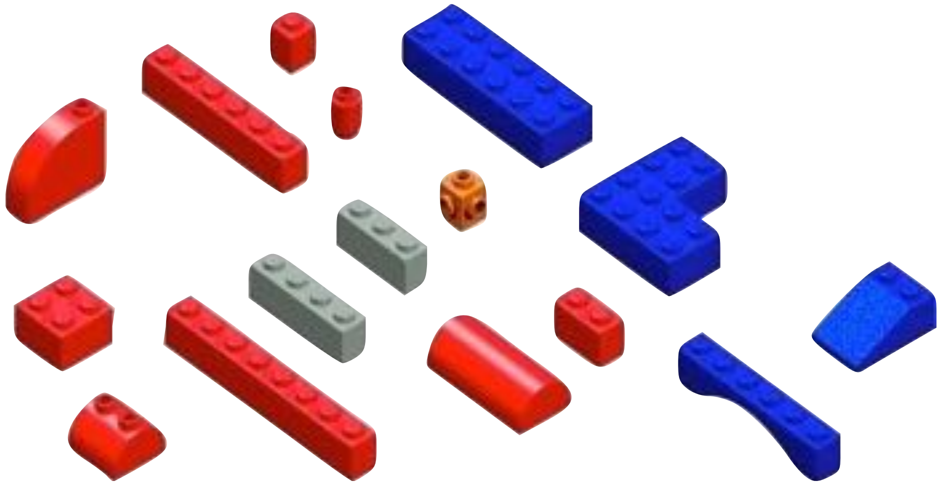


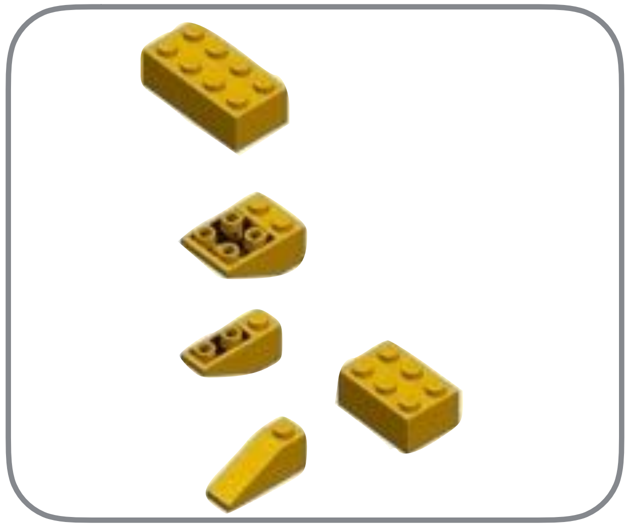
IMP.algebra



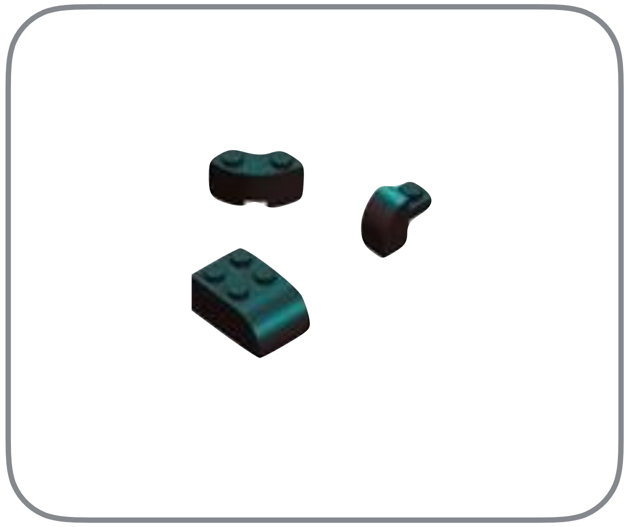
IMP kernel

IMP is divided into *modules*

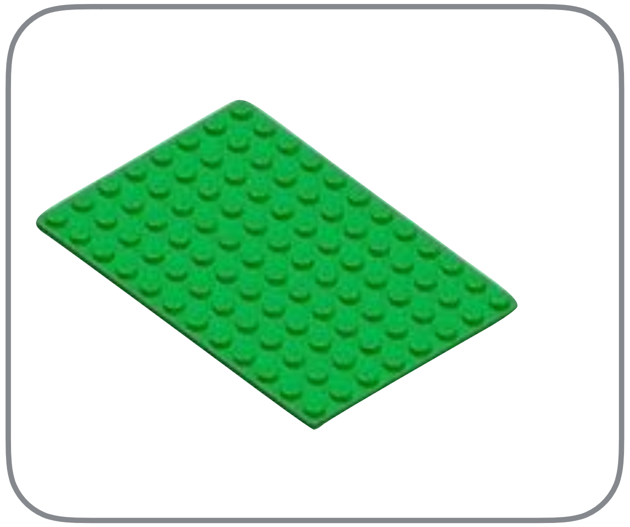




IMP.saxs

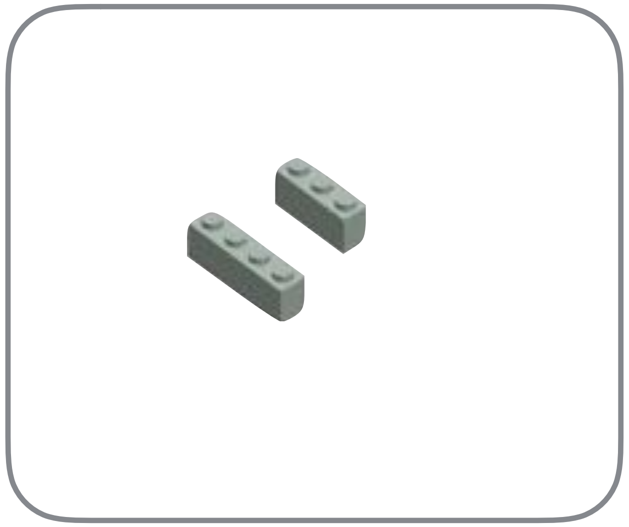


IMP.algebra

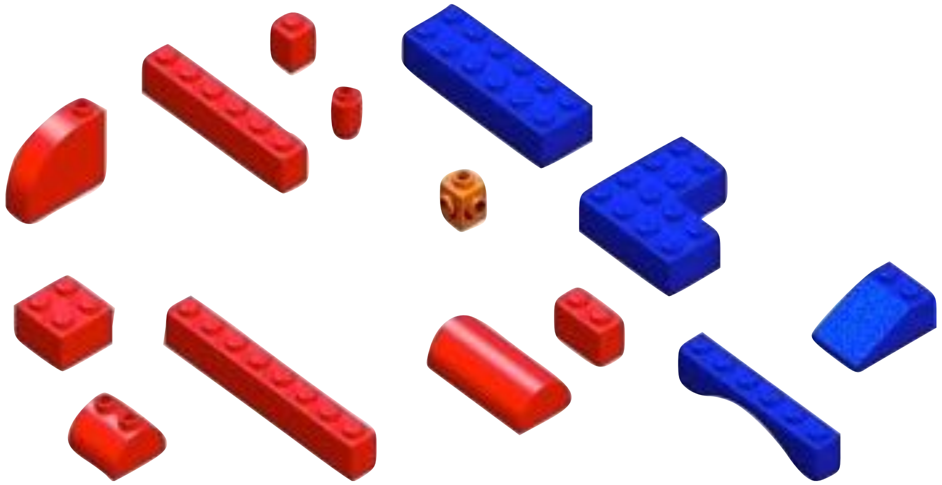


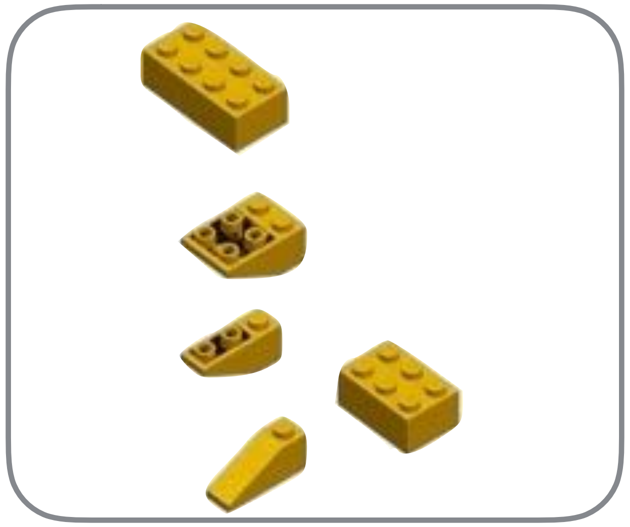
IMP kernel

IMP is divided into *modules*

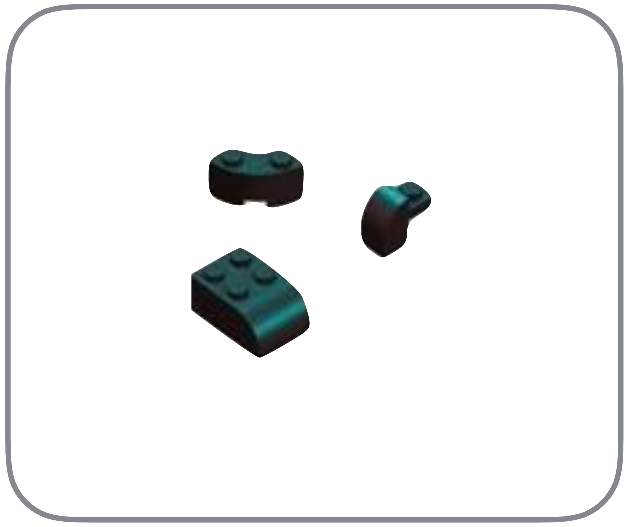


IMP.em

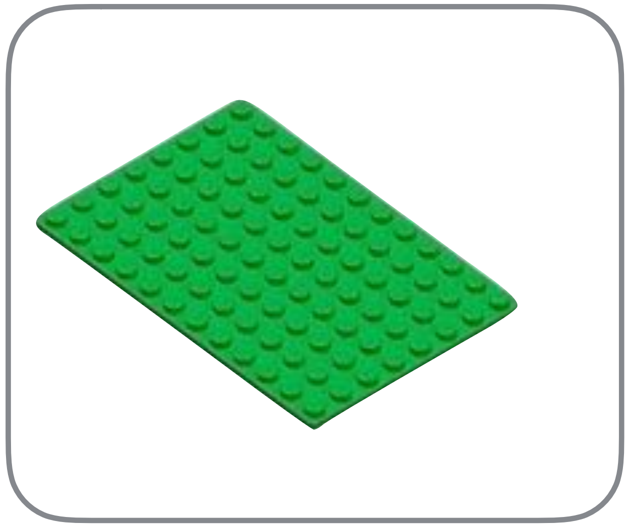




IMP.saxs

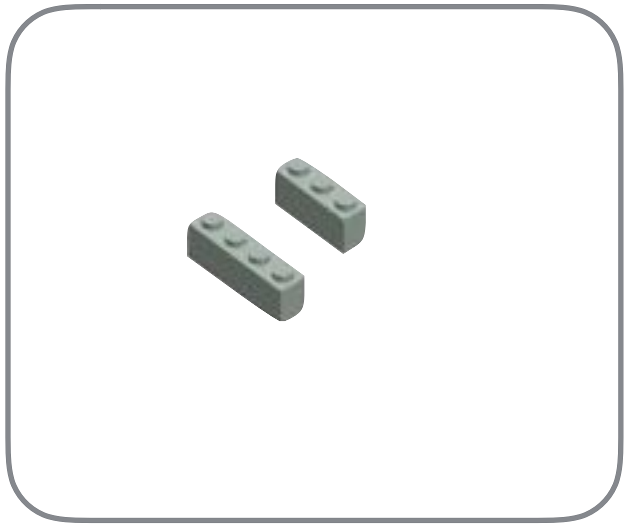


IMP.algebra



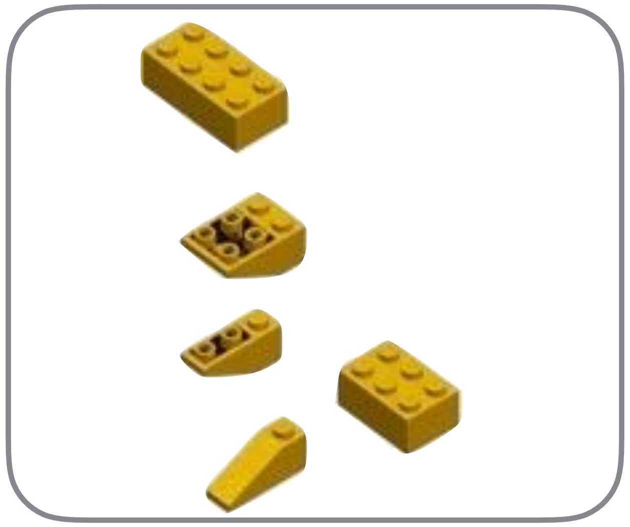
IMP kernel

IMP is divided into *modules*

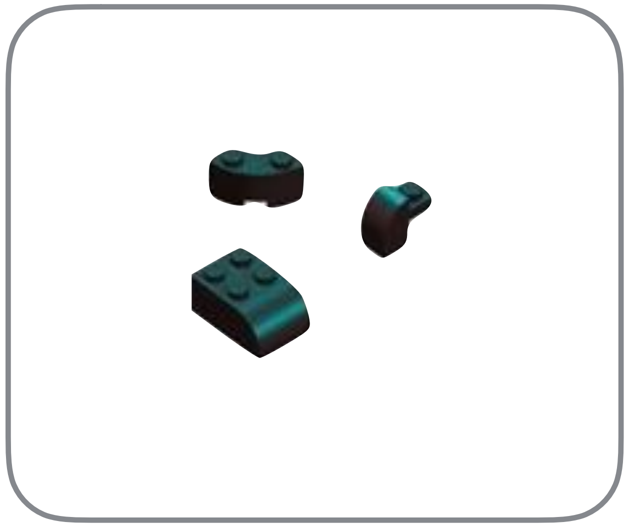


IMP.em

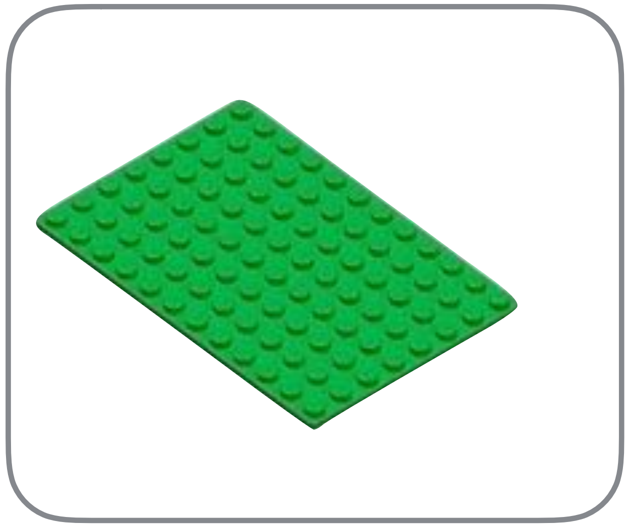
- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces



IMP.saxs

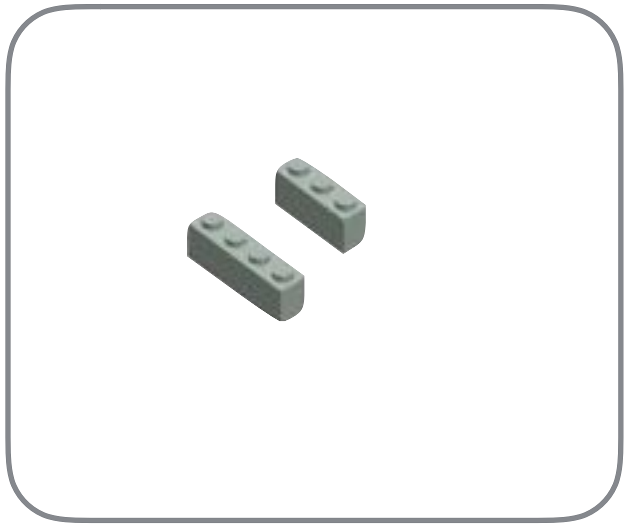


IMP.algebra



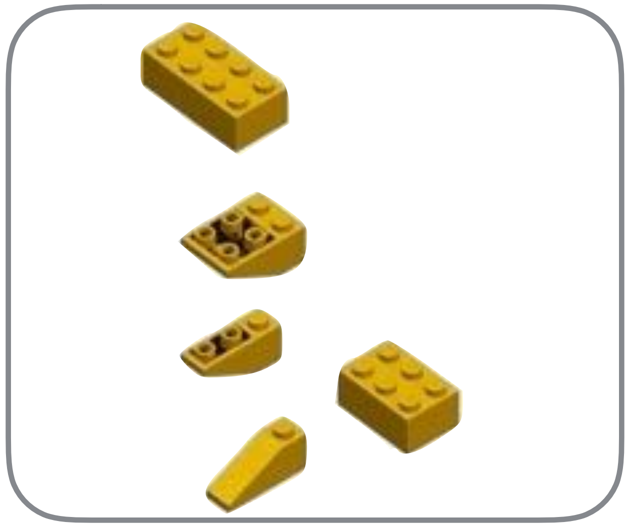
IMP kernel

IMP is divided into *modules*

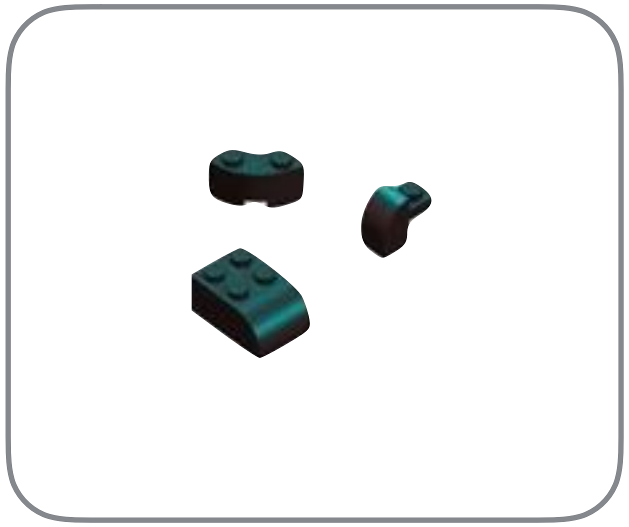


IMP.em

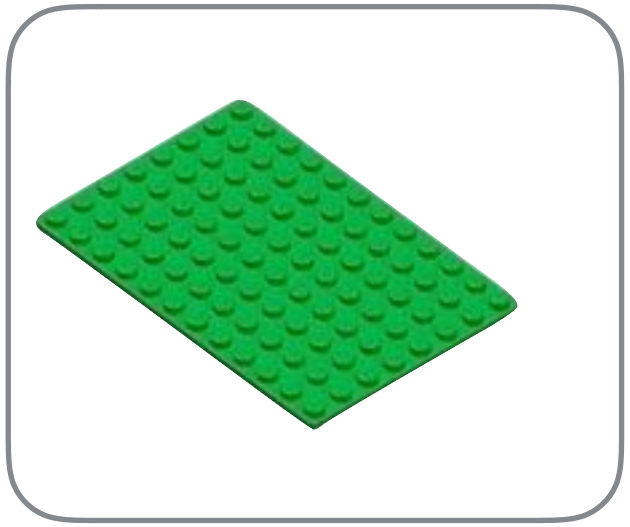
- Related functionality
- Can be developed separately
- Can be licensed differently
- Stable interfaces



IMP.saxs



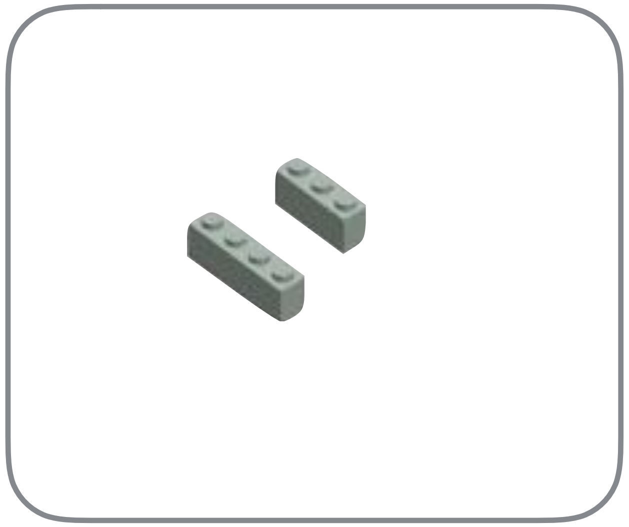
IMP.algebra



IMP kernel

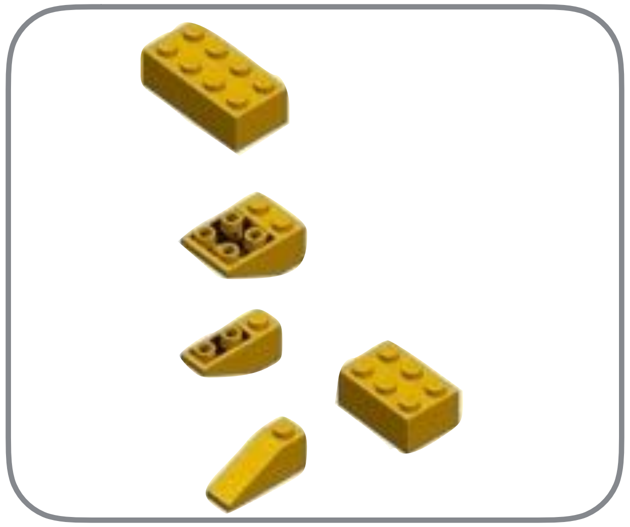
Common functionality

IMP is divided into *modules*

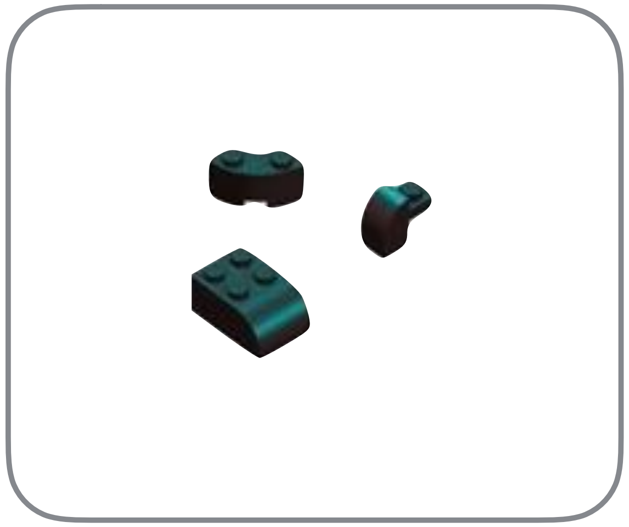


IMP.em

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

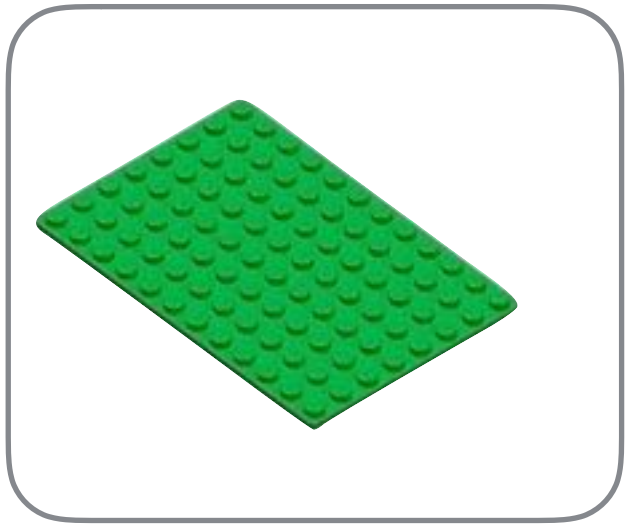


IMP.saxs



IMP.algebra

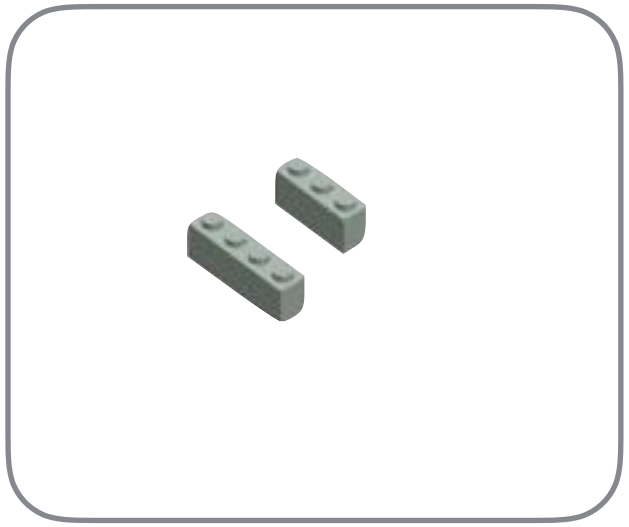
Geometry, primitive shapes



IMP kernel

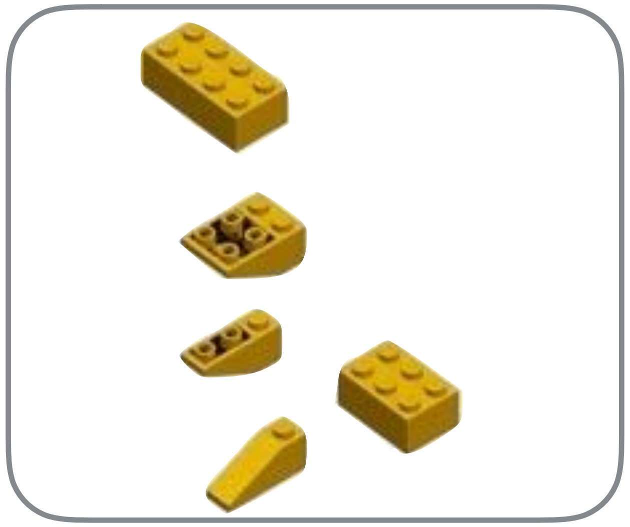
Common functionality

IMP is divided into *modules*



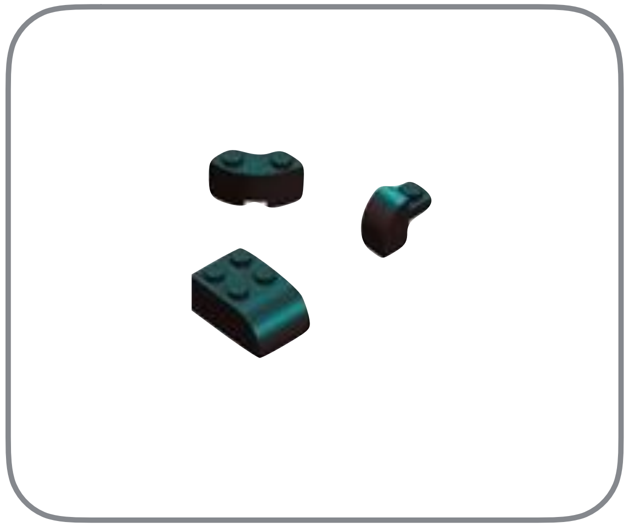
IMP.em

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



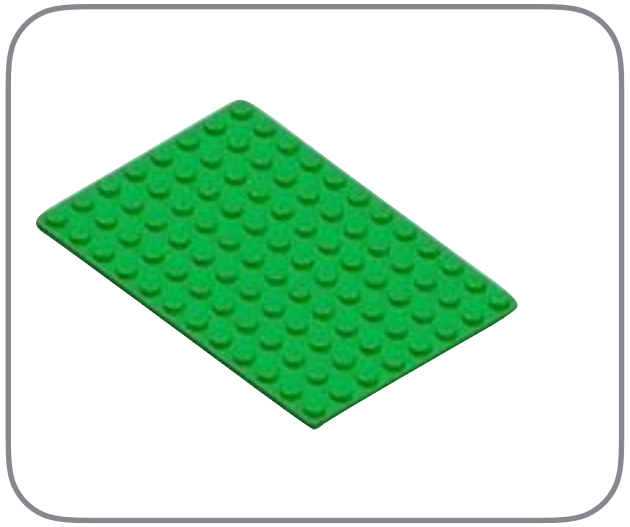
IMP.saxs

Handling of Small Angle X-ray (SAXS) data



IMP.algebra

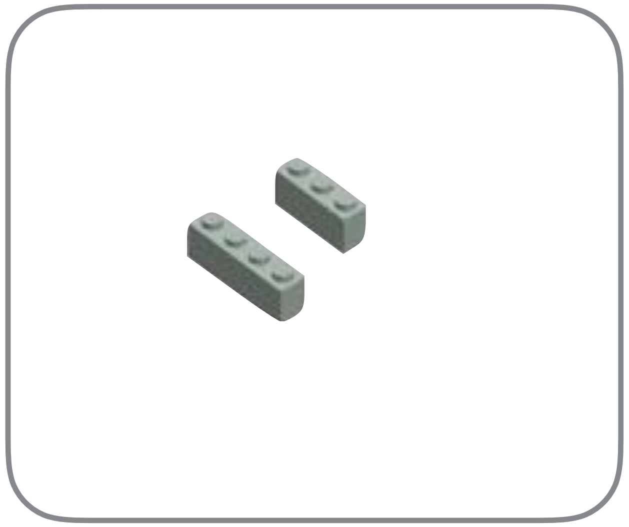
Geometry, primitive shapes



IMP kernel

Common functionality

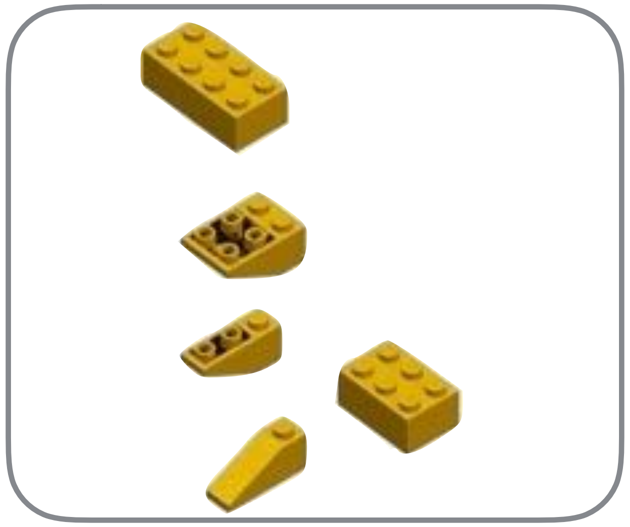
IMP is divided into *modules*



IMP.em

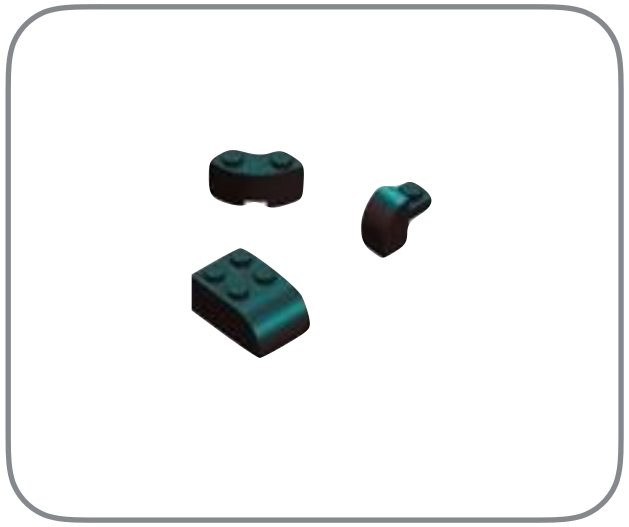
Handling of electron microscopy (EM) experimental data

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



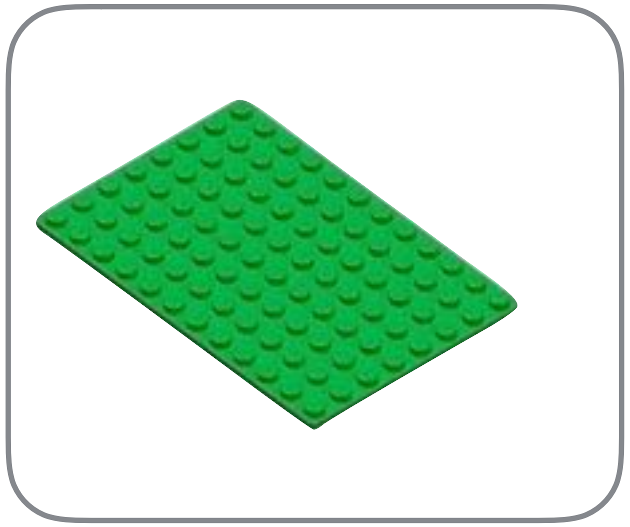
IMP.saxs

Handling of Small Angle X-ray (SAXS) data



IMP.algebra

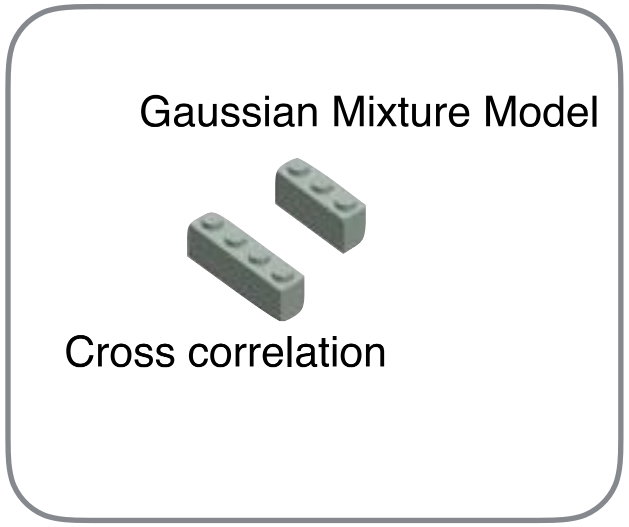
Geometry, primitive shapes



IMP kernel

Common functionality

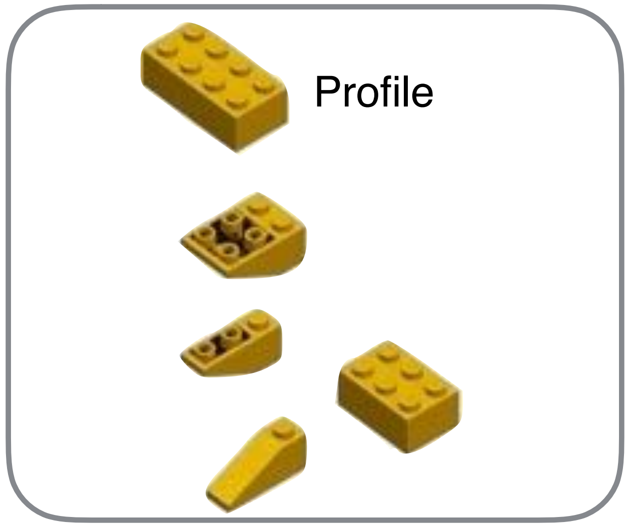
IMP is divided into *modules*



IMP.em

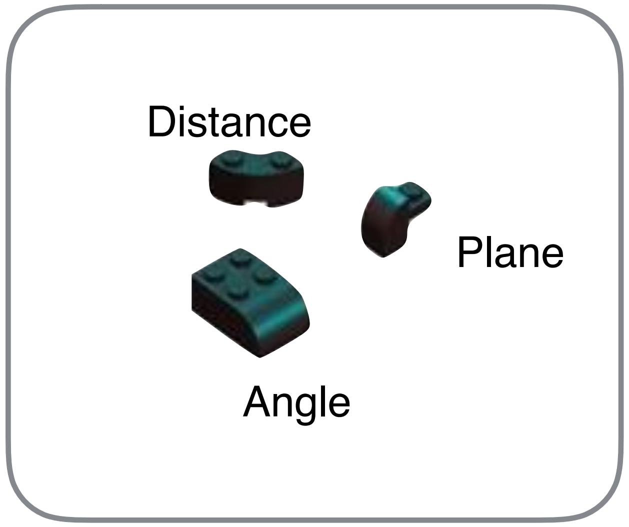
Handling of electron microscopy (EM) experimental data

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



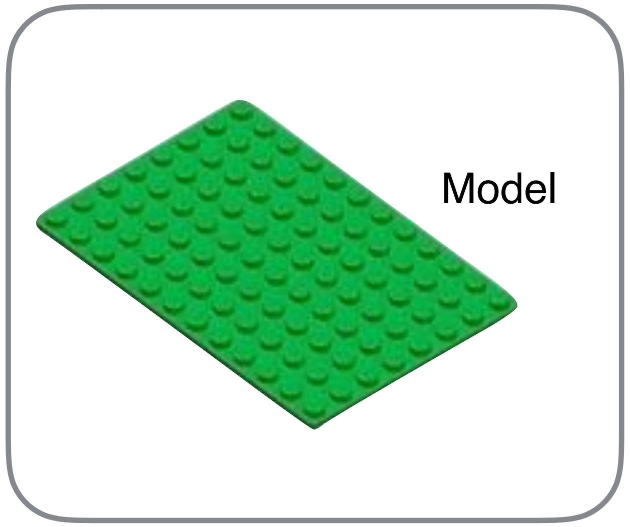
IMP.saxs

Handling of Small Angle X-ray (SAXS) data



IMP.algebra

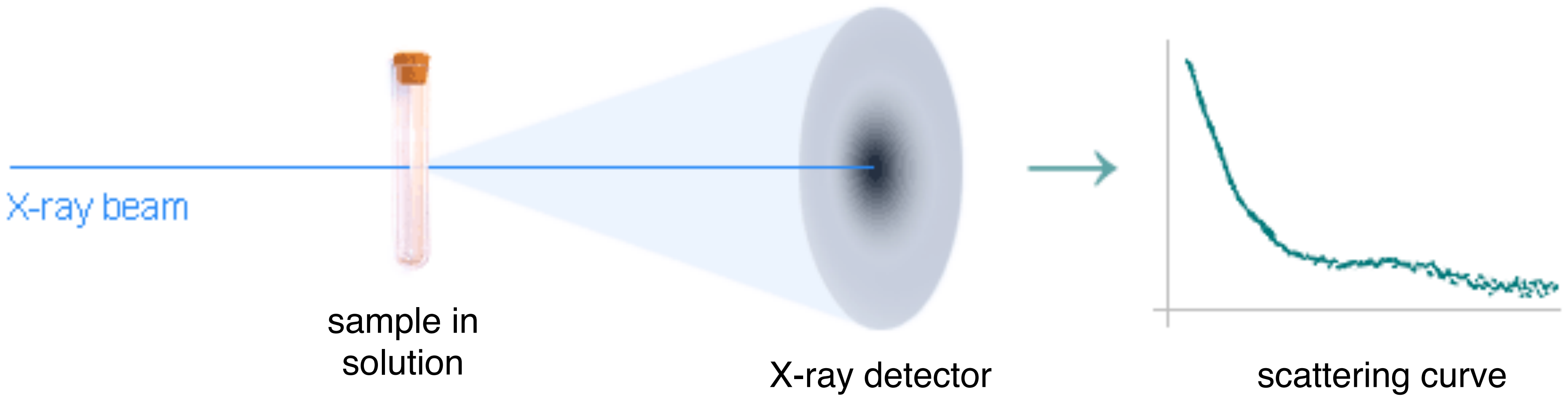
Geometry, primitive shapes



IMP kernel

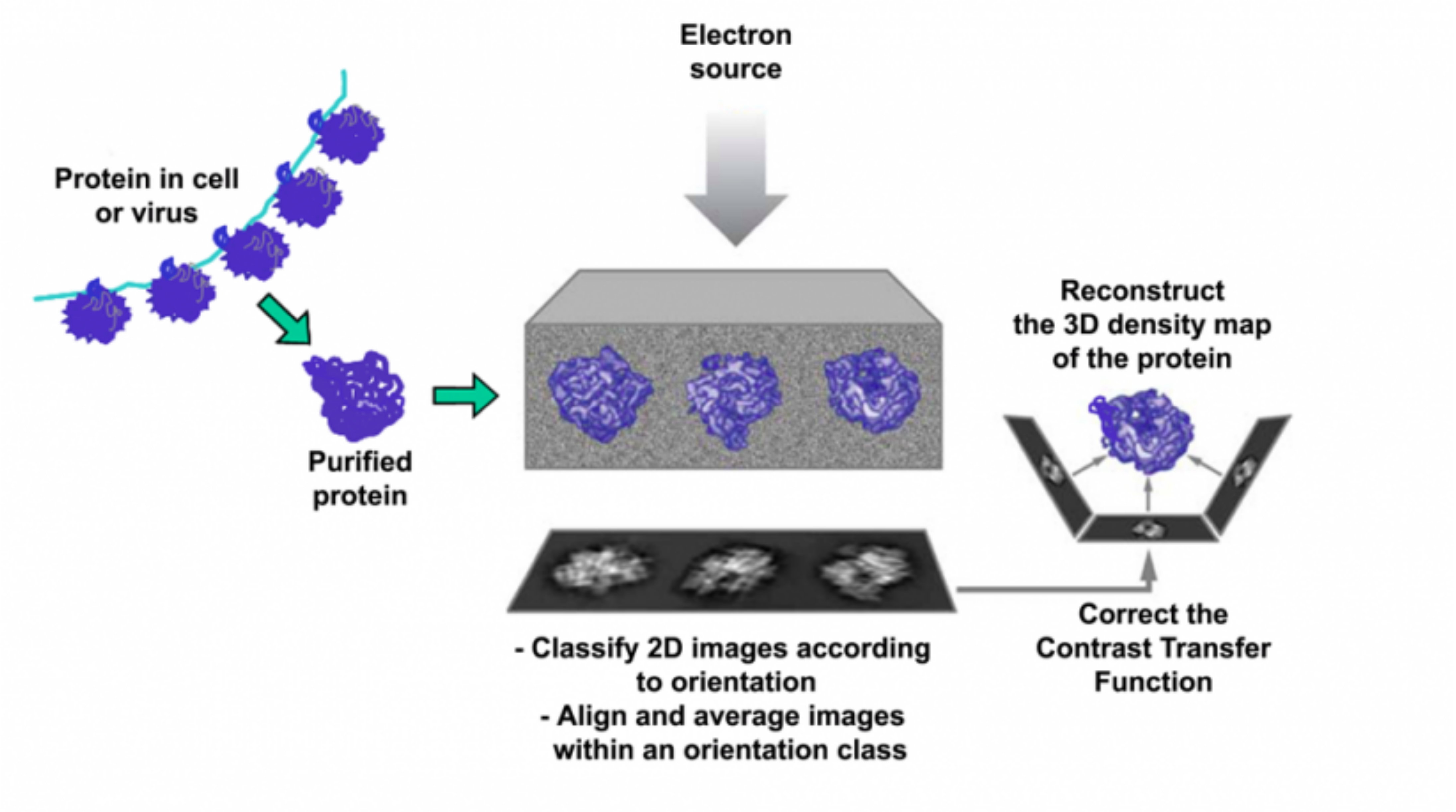
Common functionality

SAXS



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

EM


Electron source
Protein in cell or virus
Purified protein
Reconstruct the 3D density map of the protein
- Classify 2D images according to orientation
- Align and average images within an orientation class
Correct the Contrast Transfer Function

- Significant processing required to generate a 3D map

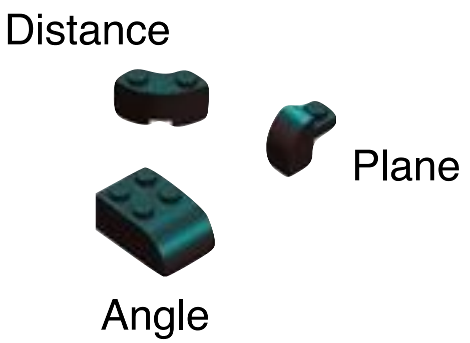
IMP software implementation

IMP software implementation

- Each ‘piece’ is a Python class

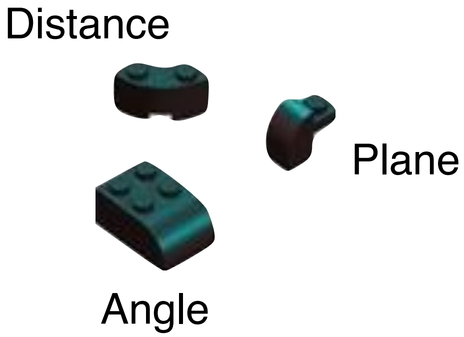
IMP software implementation

- Each ‘piece’ is a Python class



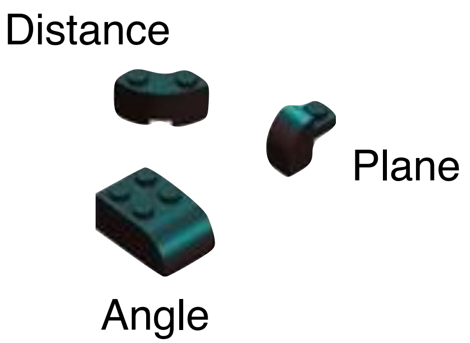
IMP software implementation

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



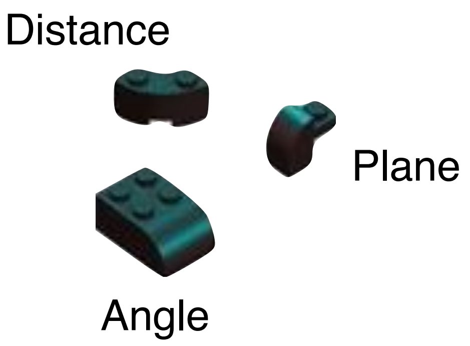
IMP software implementation

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

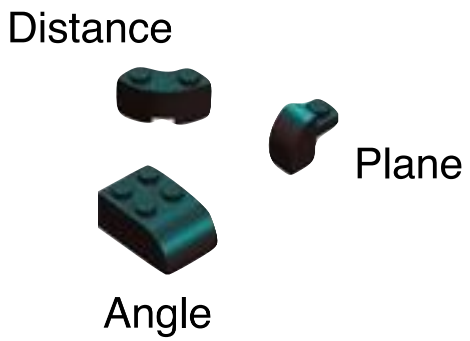


IMP software implementation

- 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



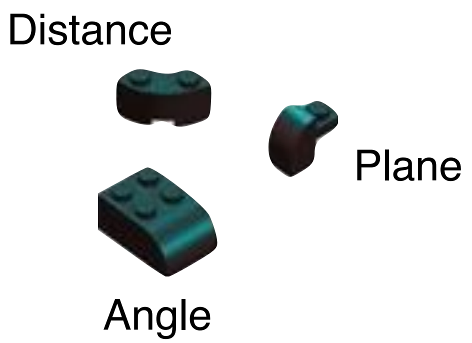
IMP software implementation



- 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

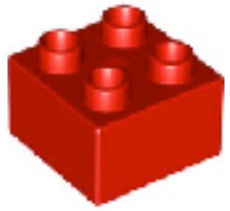
BioPython

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

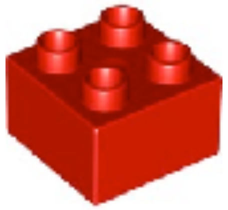
BioPython

handling of

sequence data

Chimera/VMD

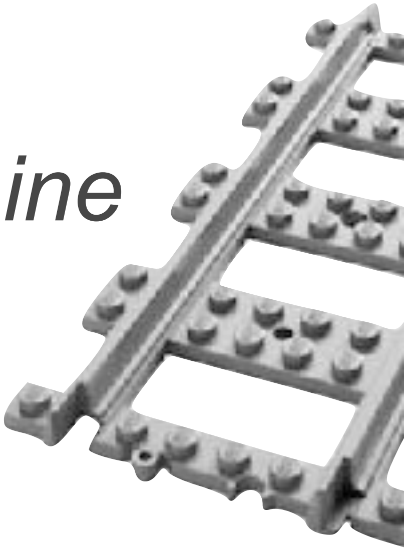
visualization



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

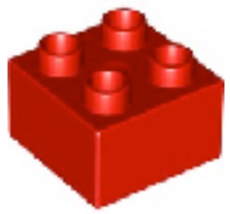
sequence data

MODELLER

comparative modeling

Chimera/VMD

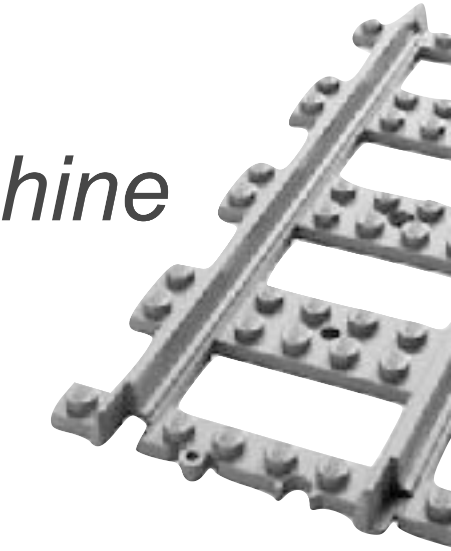
visualization



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

BioPython

handling of

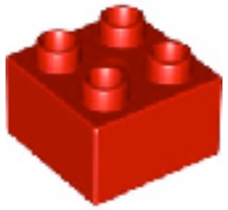
sequence data

MODELLER

comparative modeling

Chimera/VMD

visualization



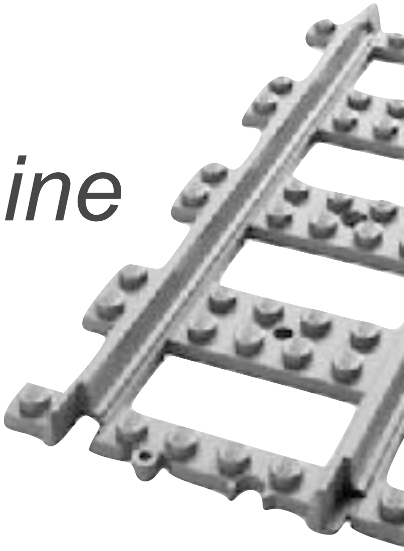
numpy/scipy

matrix/linear algebra

scikit-learn

clustering, machine

learning



etc.

Documentation

- Can be found at https://integrativemodeling.org/doc.html

- Split into a manual (designed to be read sequentially, contains tutorials similar to this one) and a reference guide (random access, documenting the IMP classes and modules)

Example Python script

```python
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/2.6.2/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

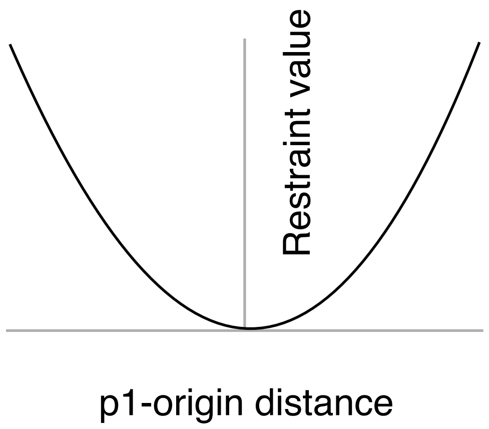
```
# Harmonically restrain p1 to be zero distance
# from the origin
f = IMP.core.Harmonic(0.0, 1.0)
s = IMP.core.DistanceToSingletonScore(f,
                                 IMP.algebra.Vector3D(0., 0., 0.))
r1 = IMP.core.SingletonRestraint(m, s, p1)
```

- 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)
```

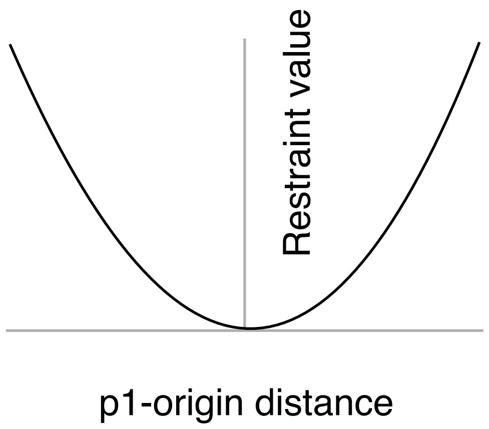
- A **`Restraint`** is a term in our scoring function, just a function of one or more particles
- **`IMP.core.SingletonRestraint`** applies a **`SingletonScore`** to a single particle (**`p1`** in this case)
- In turn, **`DistanceToSingletonScore`** calculates the Cartesian distance between a fixed point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

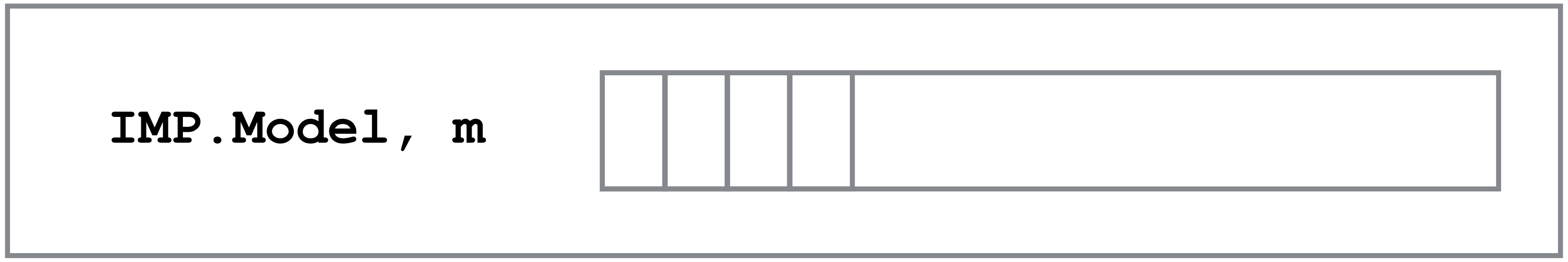
Sampling

```
# 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`

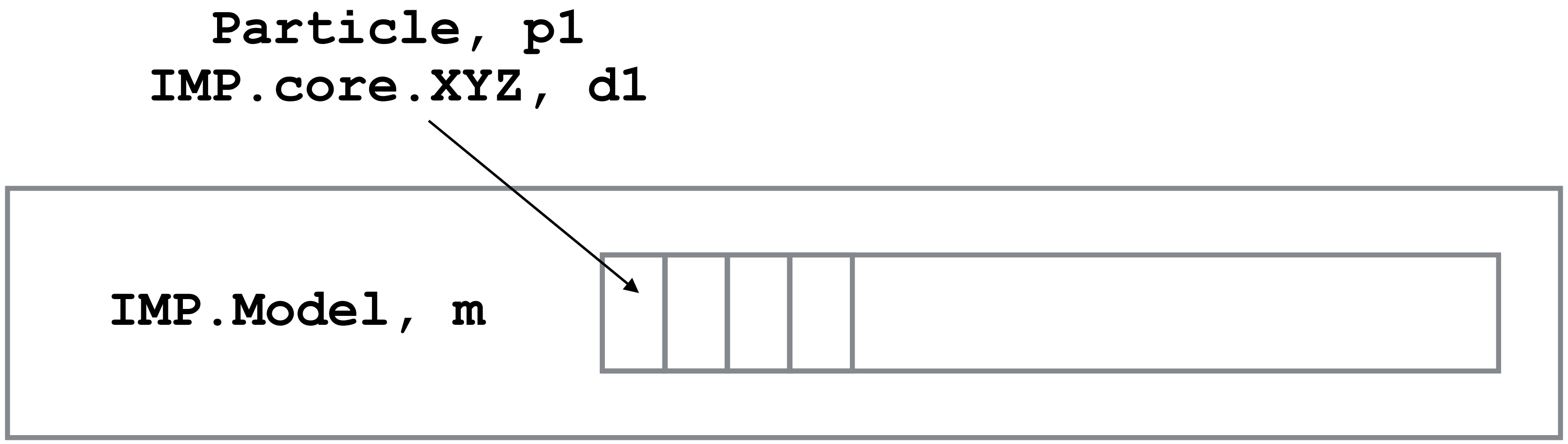
Overall workflow

Overall workflow

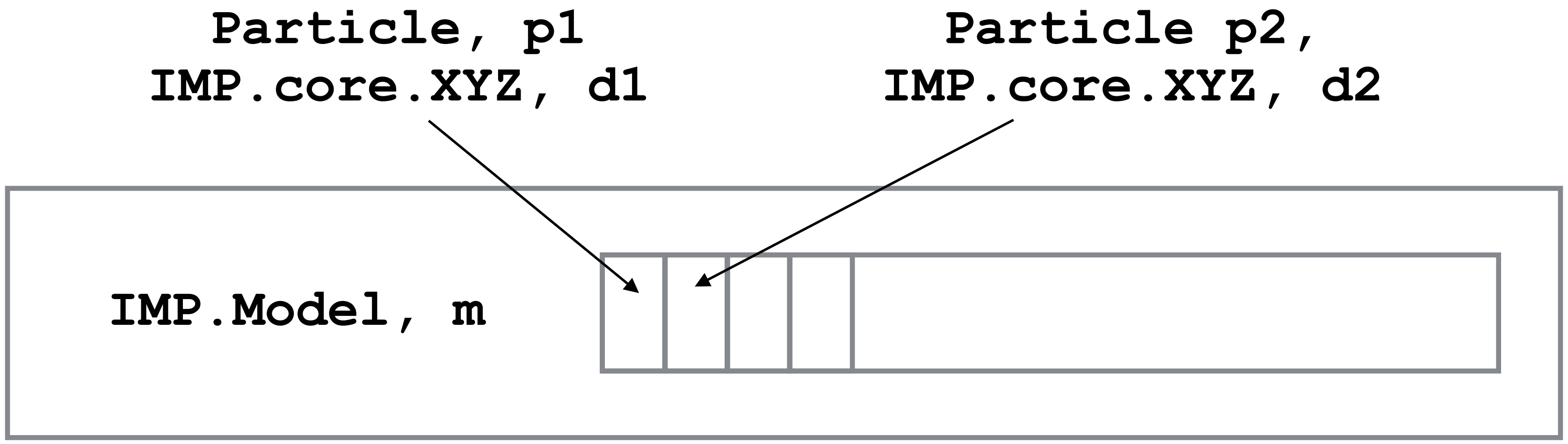


IMP.Model, m

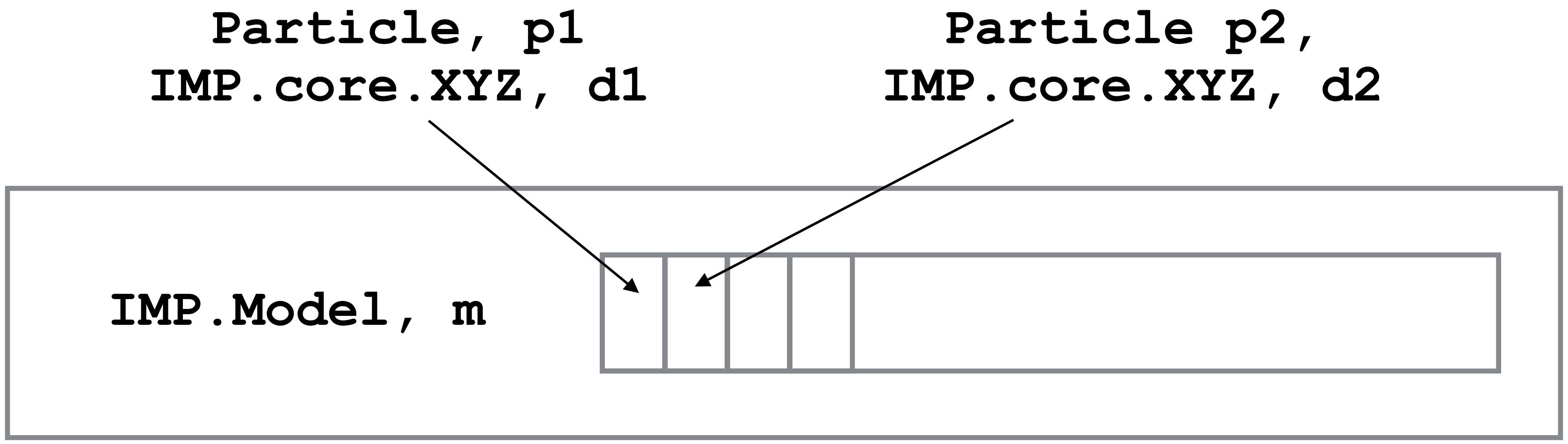
Overall workflow



Overall workflow



Overall workflow



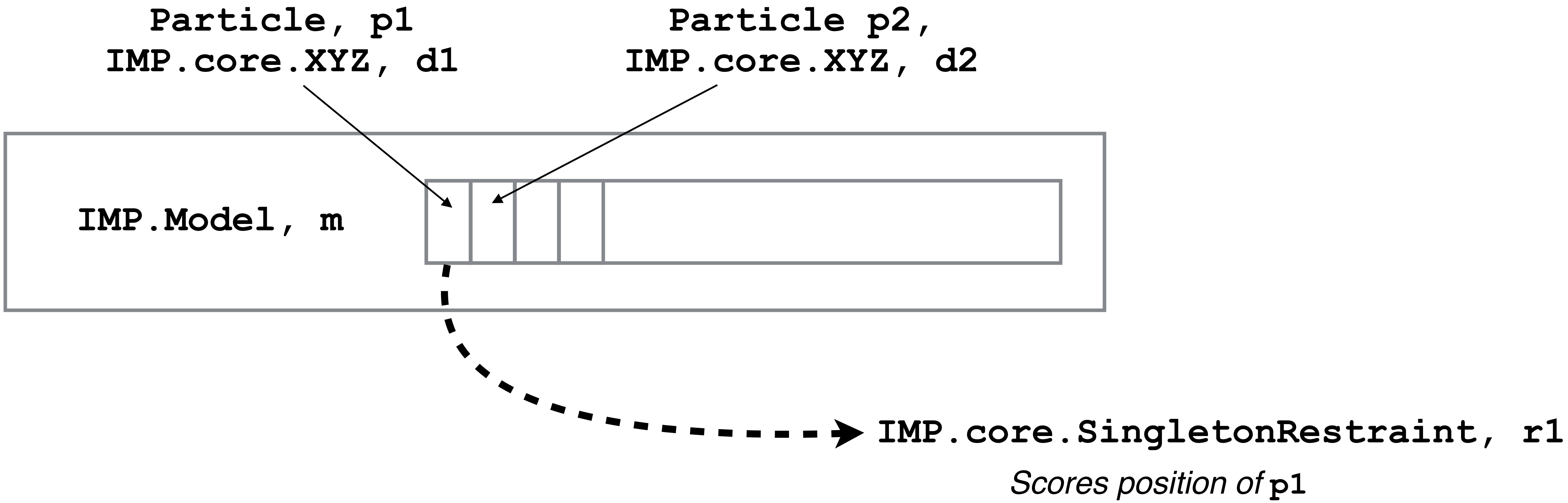
Particle, p1
IMP.core.XYZ, d1

Particle p2,
IMP.core.XYZ, d2

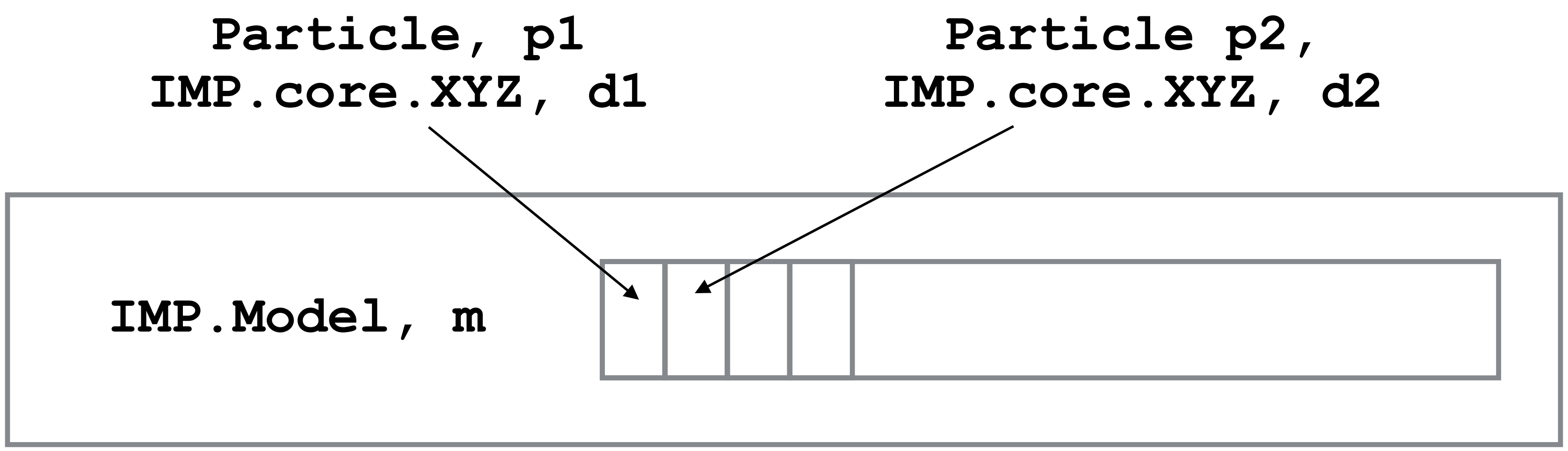
IMP.Model, m

IMP.core.SingletonRestraint, r1

Overall workflow



Overall workflow



Particle, p1
IMP.core.XYZ, d1

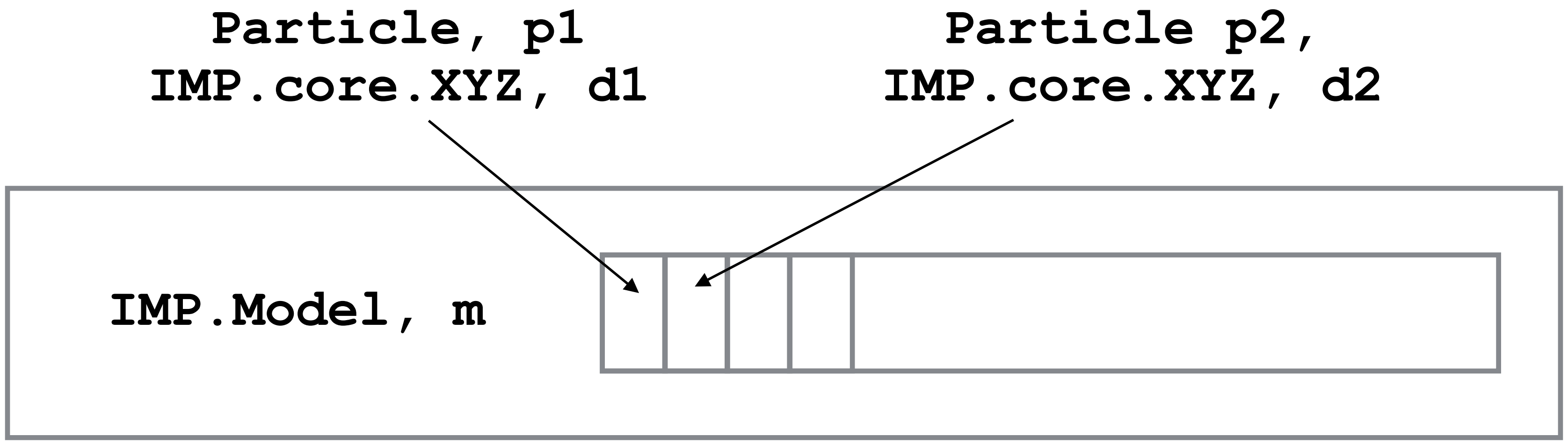
Particle p2,
IMP.core.XYZ, d2

IMP.Model, m

IMP.core.SingletonRestraint, r1

Scores position of **p1**

Overall workflow



`Particle, p1`
`IMP.core.XYZ, d1`

`Particle p2,`
`IMP.core.XYZ, d2`

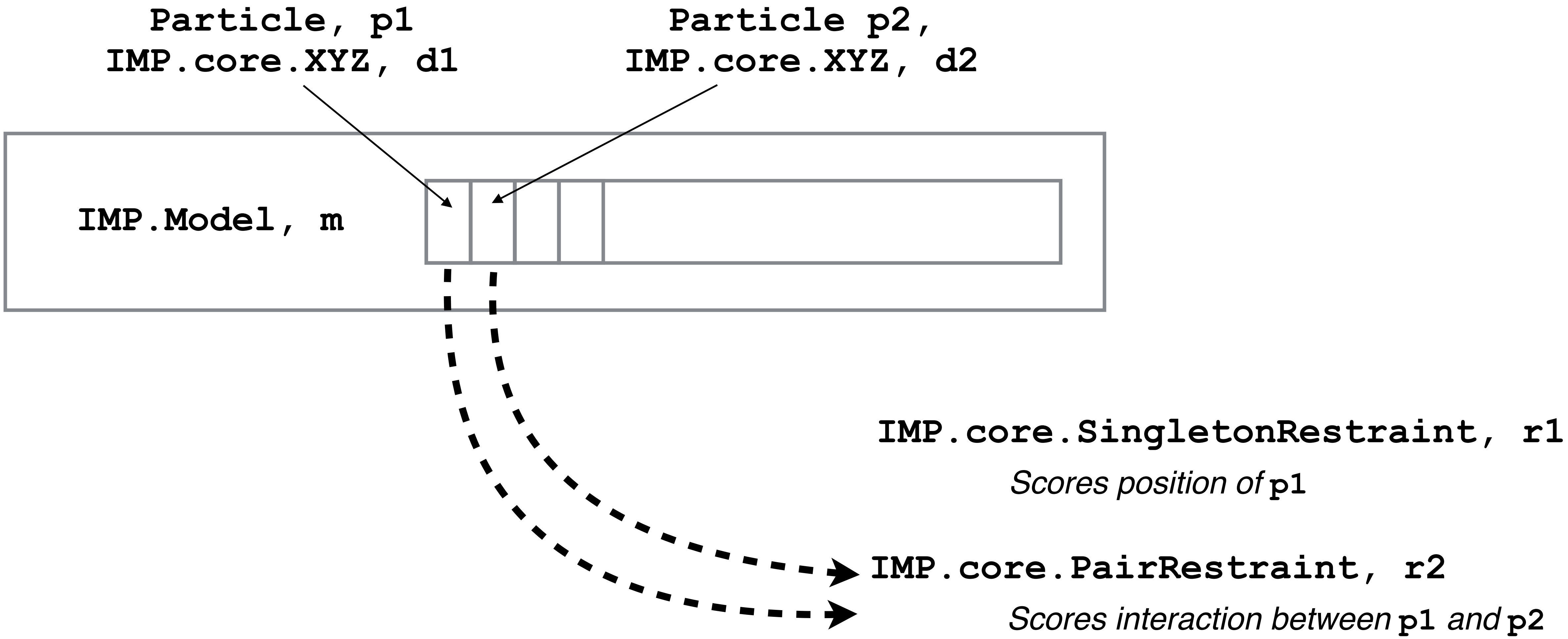
`IMP.Model, m`

`IMP.core.SingletonRestraint, r1`

Scores position of `p1`

`IMP.core.PairRestraint, r2`

Overall workflow



Particle, p1
IMP.core.XYZ, d1

Particle p2,
IMP.core.XYZ, d2

IMP.Model, m

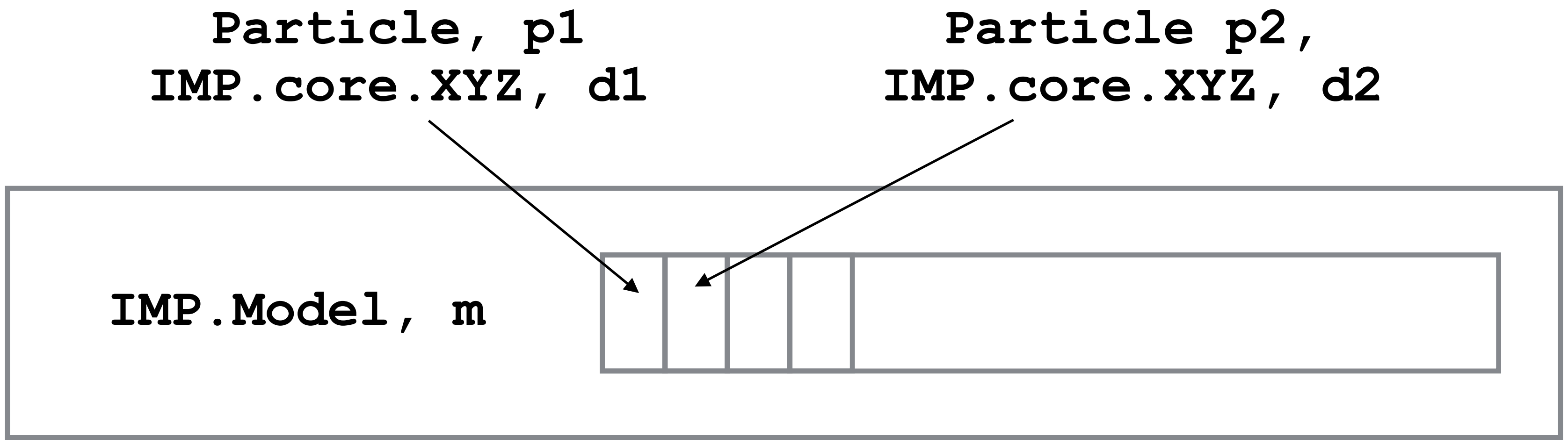
IMP.core.SingletonRestraint, r1

Scores position of **p1**

IMP.core.PairRestraint, r2

Scores interaction between **p1** *and* **p2**

Overall workflow



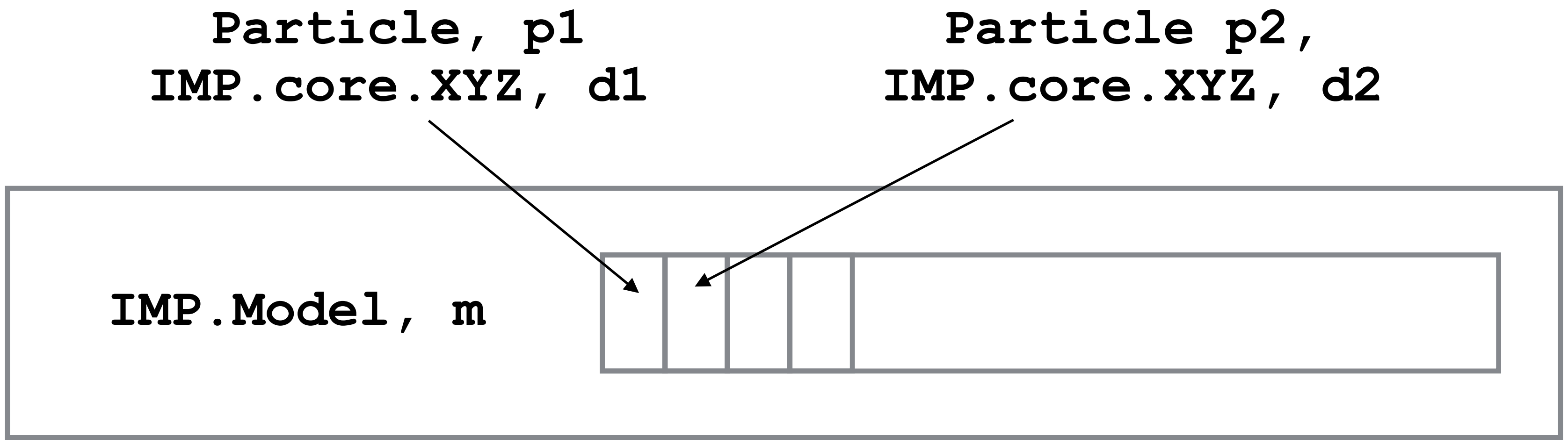
`IMP.core.SingletonRestraint, r1`

Scores position of **`p1`**

`IMP.core.PairRestraint, r2`

Scores interaction between **`p1`** *and* **`p2`**

Overall workflow



`Particle, p1`
`IMP.core.XYZ, d1`

`Particle p2,`
`IMP.core.XYZ, d2`

`IMP.Model, m`

`IMP.core.SingletonRestraint, r1`

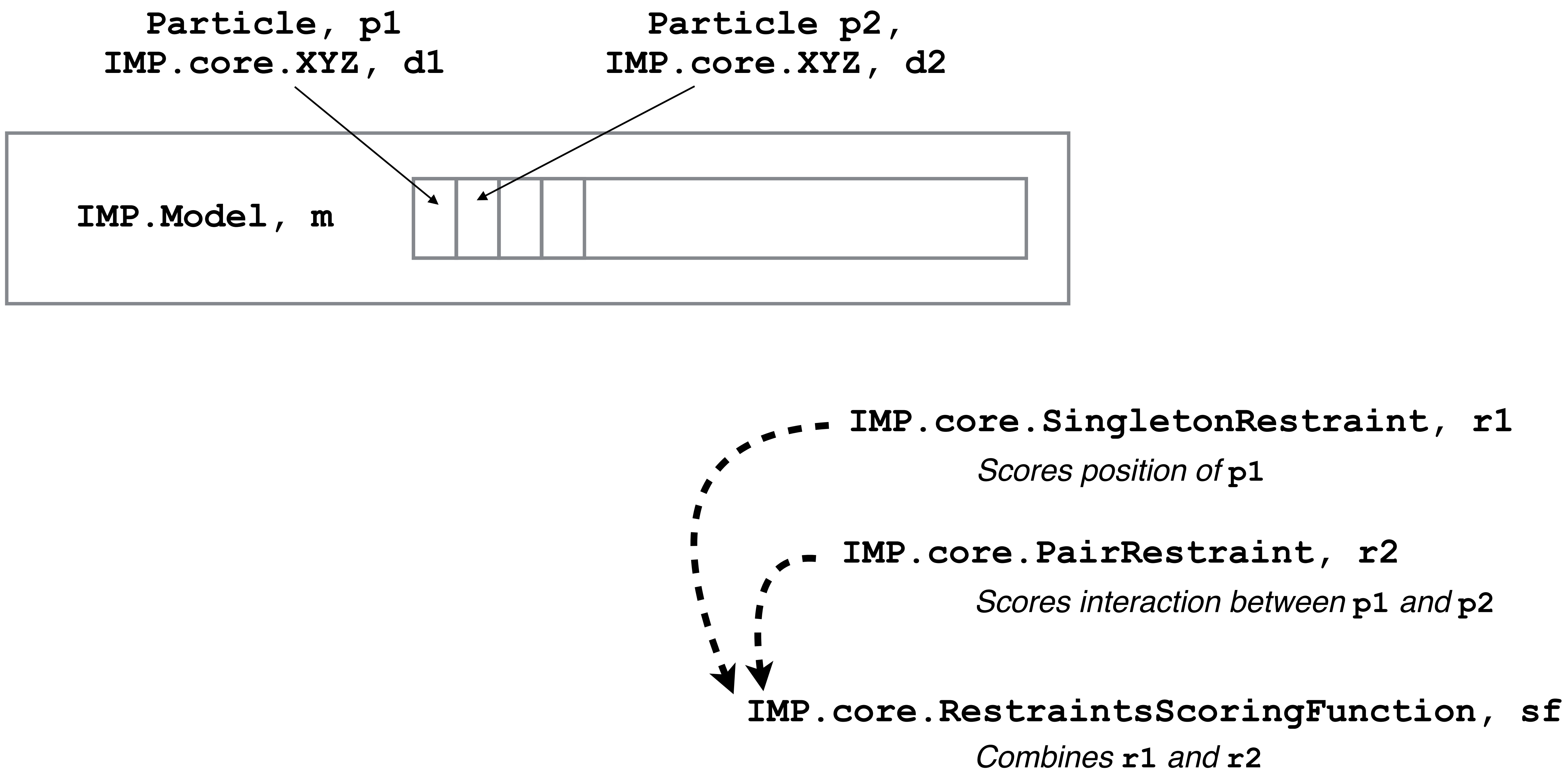
Scores position of `p1`

`IMP.core.PairRestraint, r2`

Scores interaction between `p1` *and* `p2`

`IMP.core.RestraintsScoringFunction, sf`

Overall workflow



`Particle, p1`
`IMP.core.XYZ, d1`

`Particle p2,`
`IMP.core.XYZ, d2`

`IMP.Model, m`

`IMP.core.SingletonRestraint, r1`

Scores position of `p1`

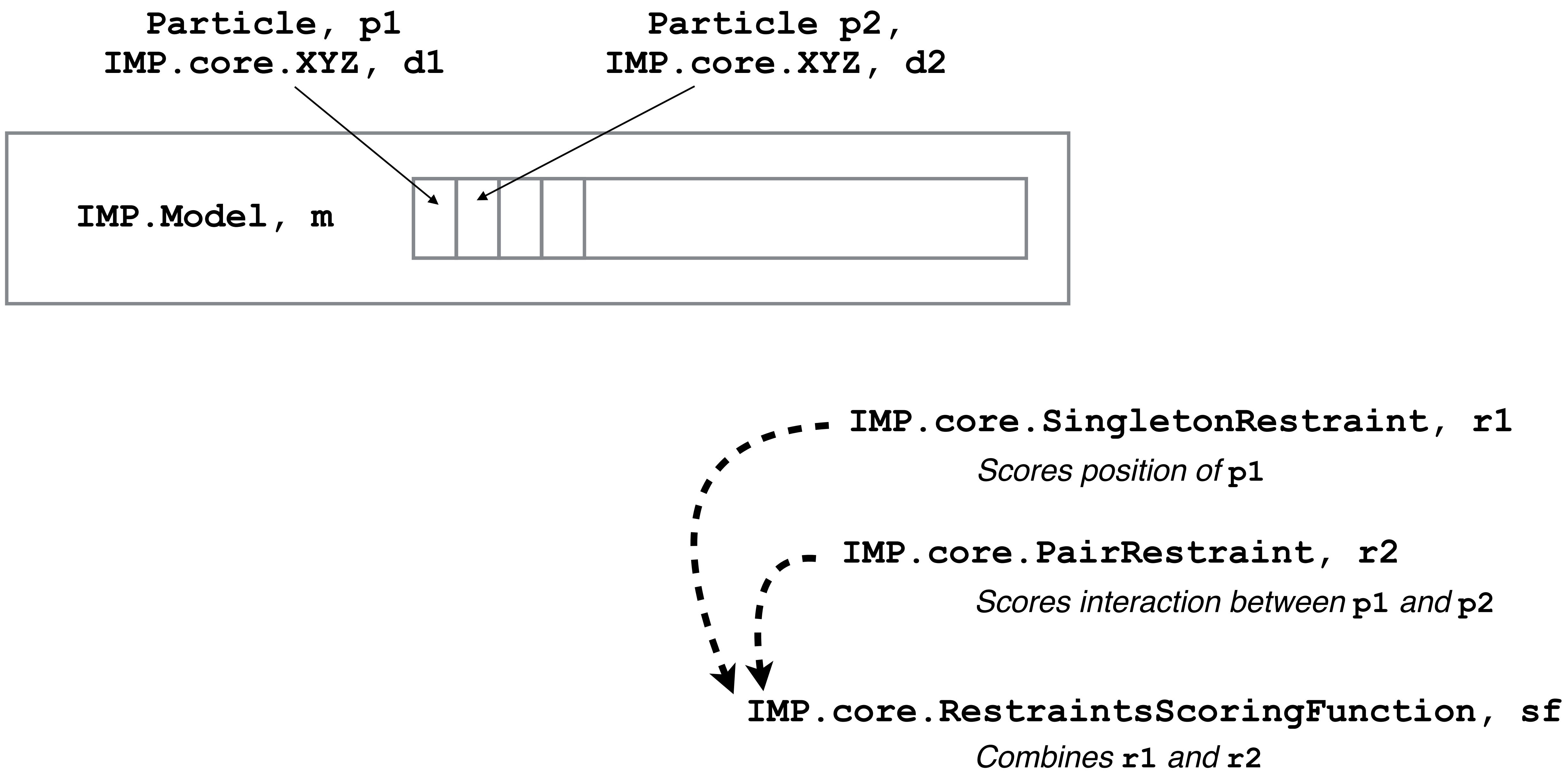
`IMP.core.PairRestraint, r2`

Scores interaction between `p1` *and* `p2`

`IMP.core.RestraintsScoringFunction, sf`

Combines `r1` *and* `r2`

Overall workflow



Particle, p1
IMP.core.XYZ, d1

Particle p2,
IMP.core.XYZ, d2

IMP.Model, m

IMP.core.SingletonRestraint, r1

Scores position of p1

IMP.core.PairRestraint, r2

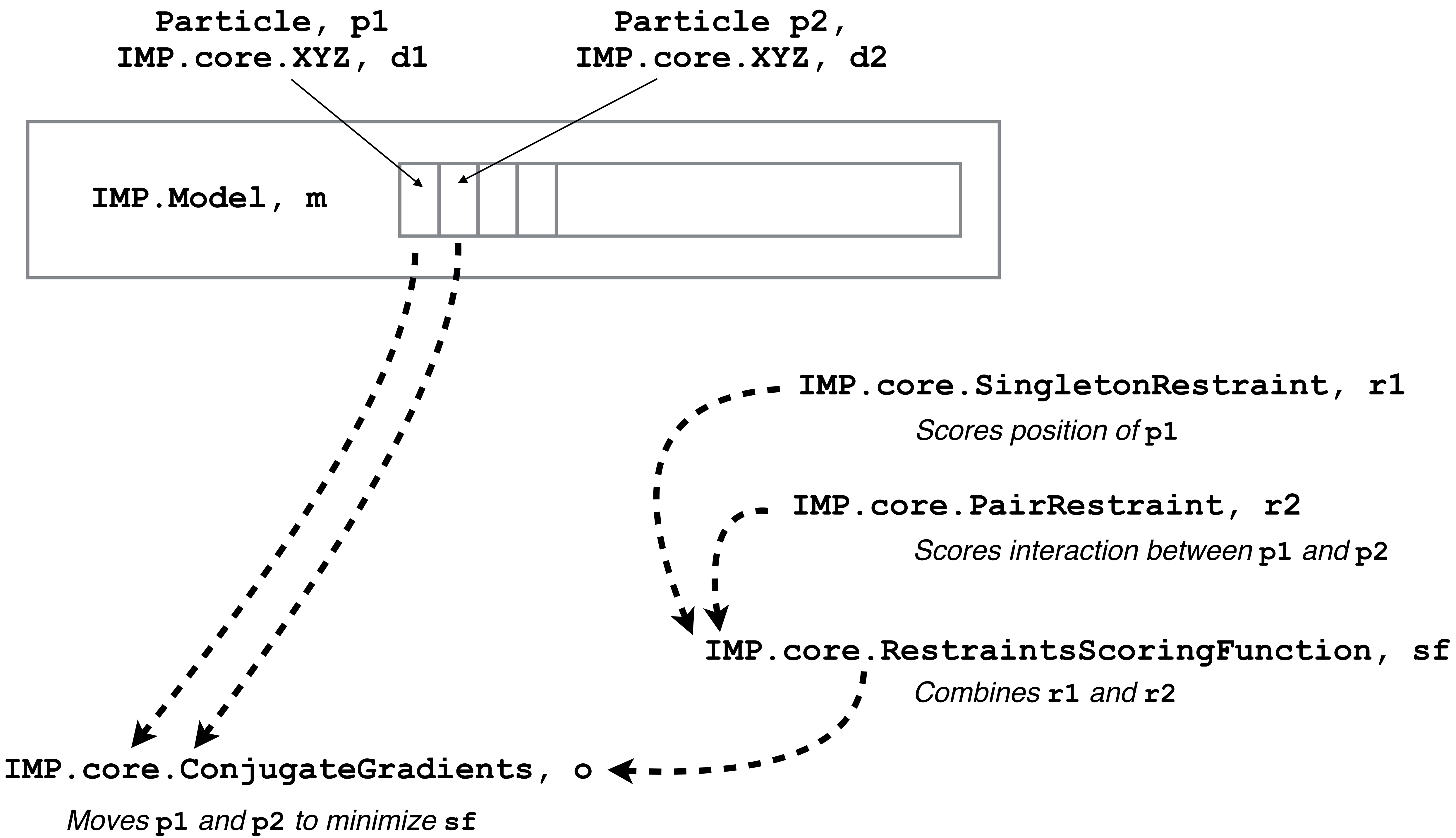
Scores interaction between p1 *and* p2

IMP.core.RestraintsScoringFunction, sf

Combines r1 *and* r2

IMP.core.ConjugateGradients, o

Overall workflow



Particle, p1
IMP.core.XYZ, d1

Particle p2,
IMP.core.XYZ, d2

IMP.Model, m

IMP.core.SingletonRestraint, r1

Scores position of p1

IMP.core.PairRestraint, r2

Scores interaction between p1 *and* p2

IMP.core.RestraintsScoringFunction, sf

Combines r1 *and* r2

IMP.core.ConjugateGradients, o

Moves p1 *and* p2 *to minimize* sf

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

```python
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 3.5.2 |Anaconda custom (64-
bit)| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import IMP
>>> IMP.__version__
'2.6.2'
>>> x = IMP.get_example_path('.')
>>> exit()
$
```

IMP installation

- Test that it installed correctly:

```
$ python
Python 3.5.2 |Anaconda custom (64-
bit)| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import IMP
>>> IMP.__version__
'2.6.2'
>>> 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 3.5.2 |Anaconda custom (64-
bit)| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import IMP
>>> IMP.__version__
'2.6.2'
>>> 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 3.5.2 |Anaconda custom (64-
bit)| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import IMP
>>> IMP.__version__
'2.6.2'
>>> 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 3.5.2 |Anaconda custom (64-
bit)| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import IMP
>>> IMP.__version__
'2.6.2'
>>> 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 3.5.2 |Anaconda custom (64-
bit)| (default, Jul  2 2016, 17:53:06)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-1)]
on linux
Type "help", "copyright", "credits" or
"license" for more information.
>>> import IMP
>>> IMP.__version__
'2.6.2'
>>> x = IMP.get_example_path('.')
>>> exit()
$
```

Windows errors

- If on Windows you see an error ending in "**`IMP is not installed or set up correctly.`**" and the path it mentions contains lots of "**`placehold_placehold`**" then you may have run into a Windows Anaconda bug. Workaround:

```
$ conda uninstall imp
$ conda install conda=4.2.9
$ conda install imp conda=4.2.9
```

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, raise a hand

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, raise a hand

- 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, raise a hand

- 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, raise a hand

- 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
```

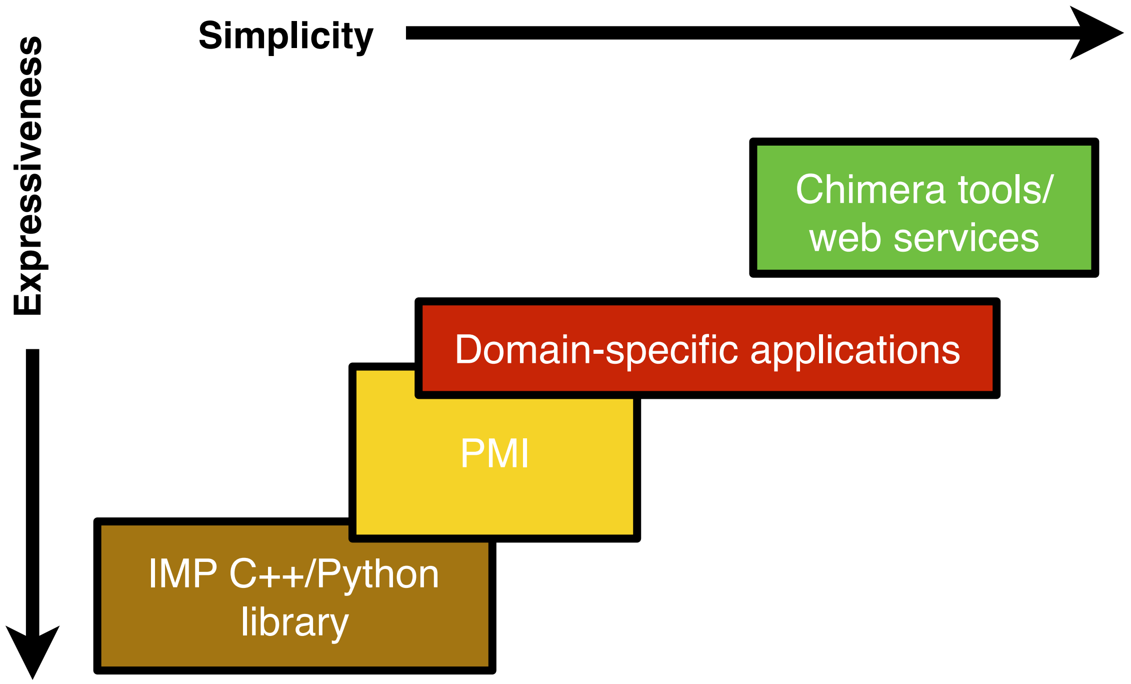
Python vs. C++

```python
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)
```

```cpp
#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



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

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

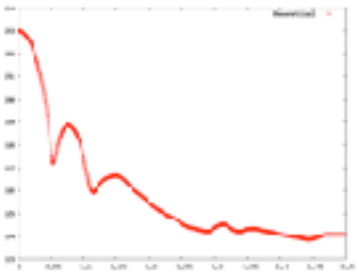
- Web services at https://salilab.org/ including:

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

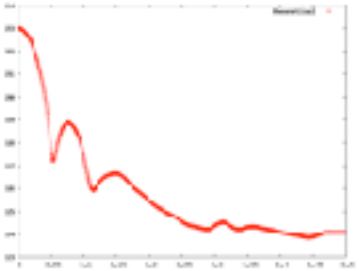
Chimera tools/ web services

- 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

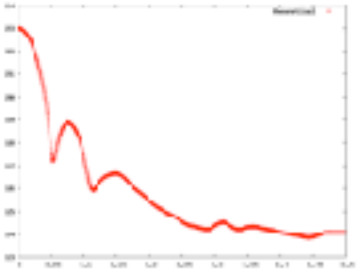
- Several plugins to UCSF Chimera that use IMP
- Web services at https://salilab.org/ including:
- AllosMod: modeling of ligand-induced protein dynamics, allostery



- 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



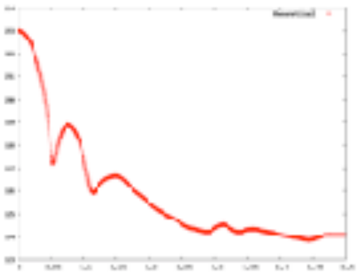
- FoXS: fast SAXS profile computation with Debye formula
- FoXSDock: macromolecular docking with SAXS Profile

Chimera tools/ web services

- Several plugins to UCSF Chimera that use IMP

- Web services at https://salilab.org/ including:

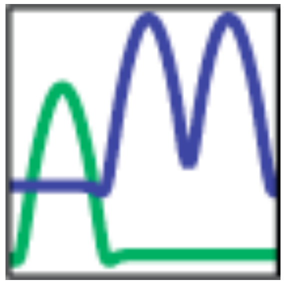
- AllosMod: modeling of ligand-induced protein dynamics, allostery



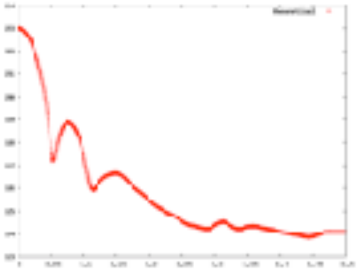
- FoXS: fast SAXS profile computation with Debye formula
- FoXSDock: macromolecular docking with SAXS Profile
- SAXSMerge: automated statistical method to merge SAXS profiles from different concentrations and exposure times

Chimera tools/ web services

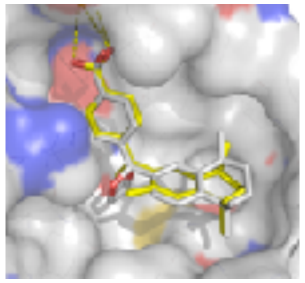
- Several plugins to UCSF Chimera that use IMP
- Web services at https://salilab.org/ including:



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

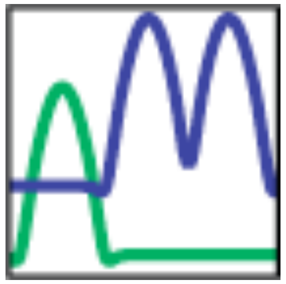


- FoXS: fast SAXS profile computation with Debye formula
- FoXSDock: macromolecular docking with SAXS Profile
- SAXSMerge: automated statistical method to merge SAXS profiles from different concentrations and exposure times

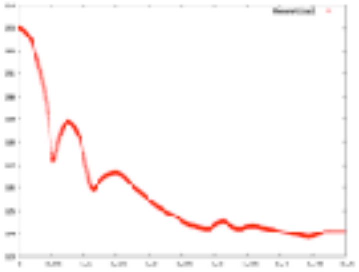


Chimera tools/ web services

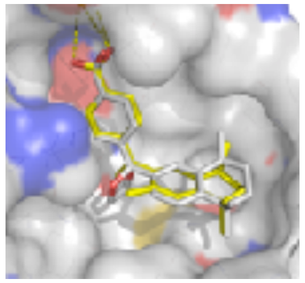
- Several plugins to UCSF Chimera that use IMP
- Web services at https://salilab.org/ including:



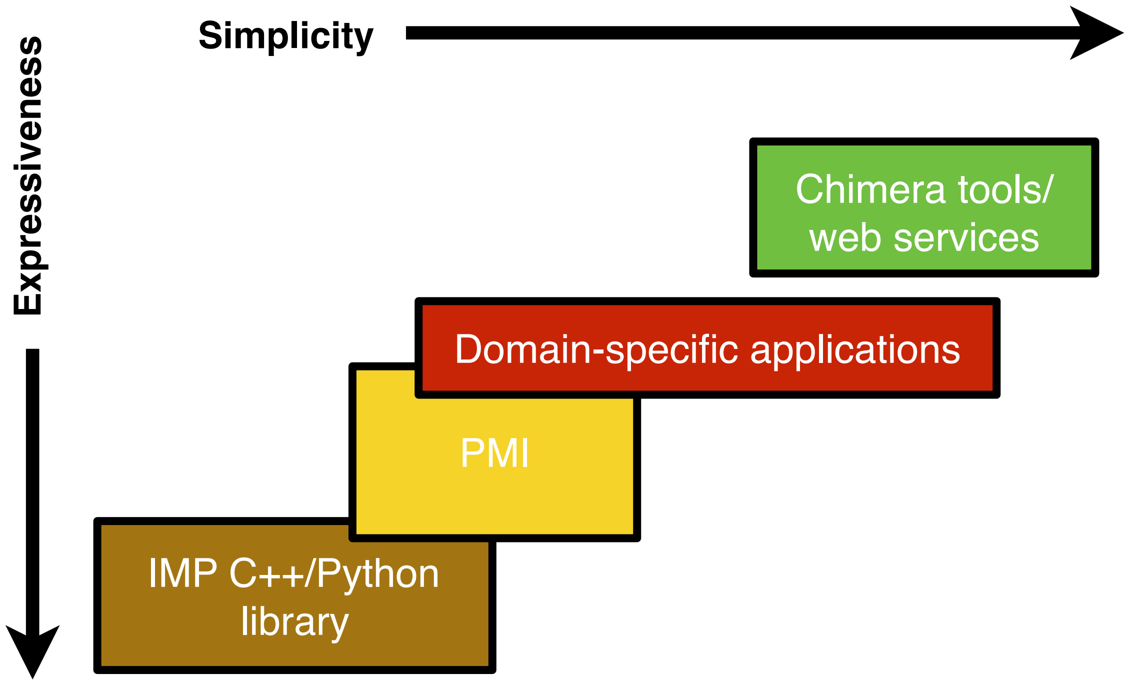
- AllosMod: modeling of ligand-induced protein dynamics, allostery



- 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


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

Domain-specific applications

Domain-specific applications

- Command line tools

Domain-specific applications

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

Domain-specific applications

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

Domain-specific applications

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

Domain-specific applications

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

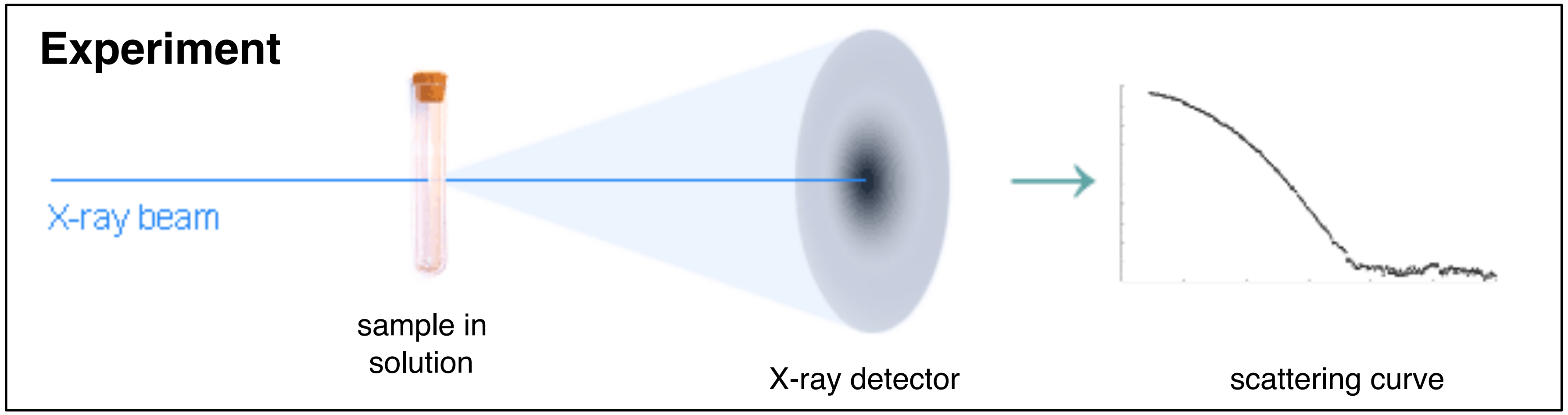
Domain-specific applications

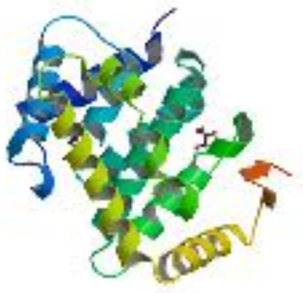
- Command line tools
- Do a very specific task, a subset of IMP functionality
- Generally, similar functionality to web services, but
  - running locally
  - more adjustable parameters, flexibility
- Today, we'll look briefly at using the `foxs` command line tool to leverage SAXS data

FoXS

FoXS

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

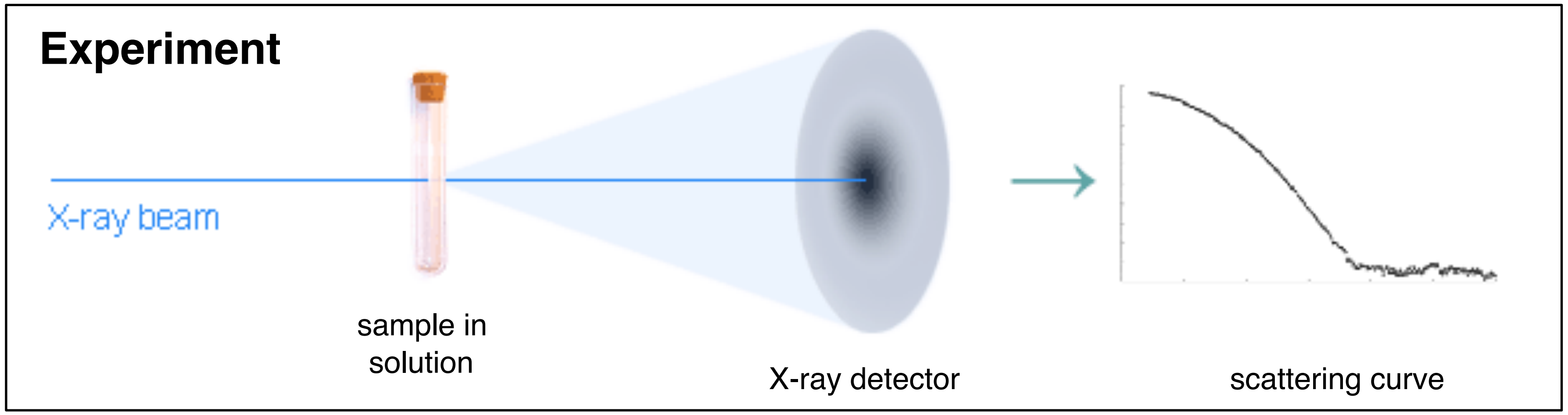


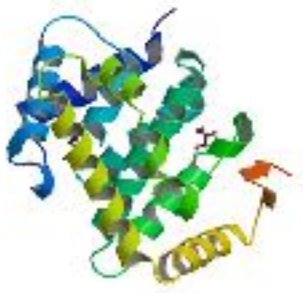


3D model

FoXS

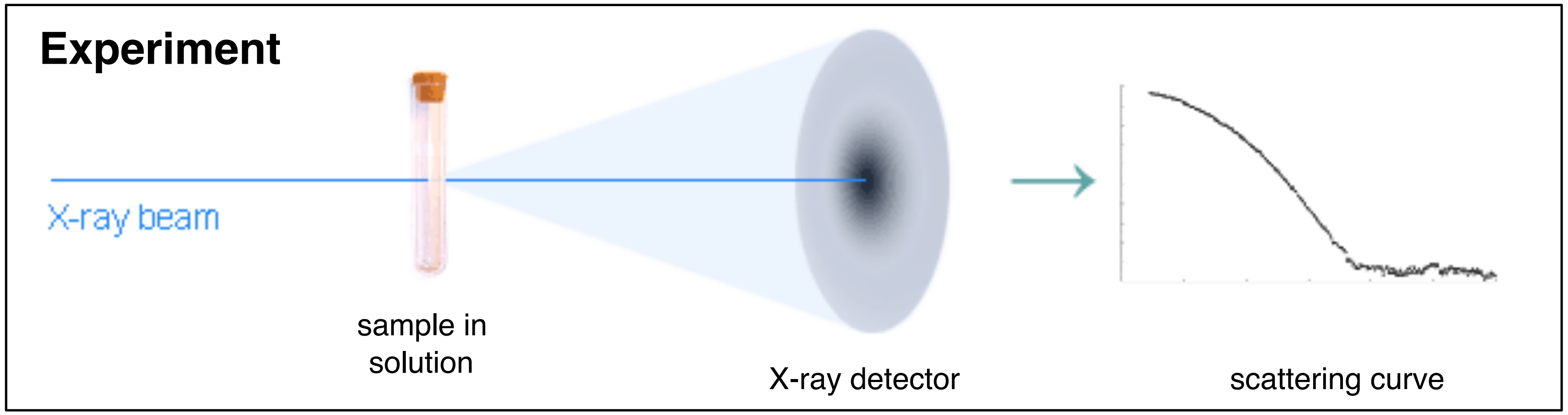
- Given an experimental SAXS profile and a 3D model, FoXS:
  - Calculates the theoretical profile of the model

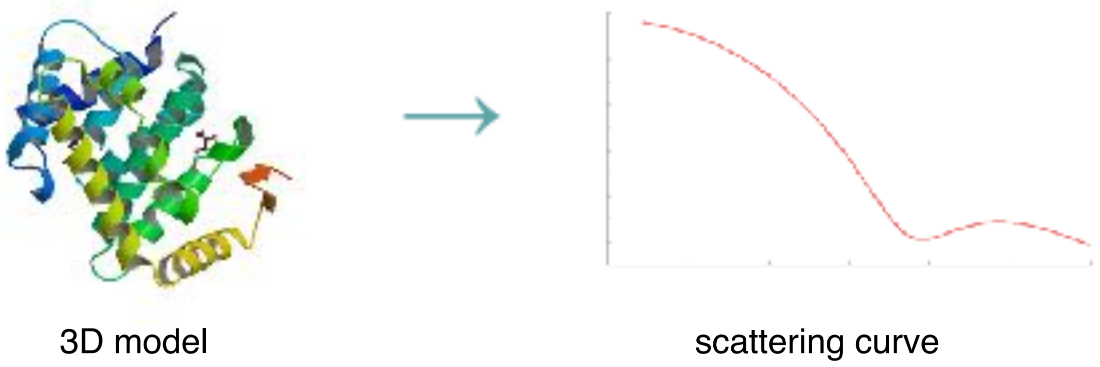




FoXS

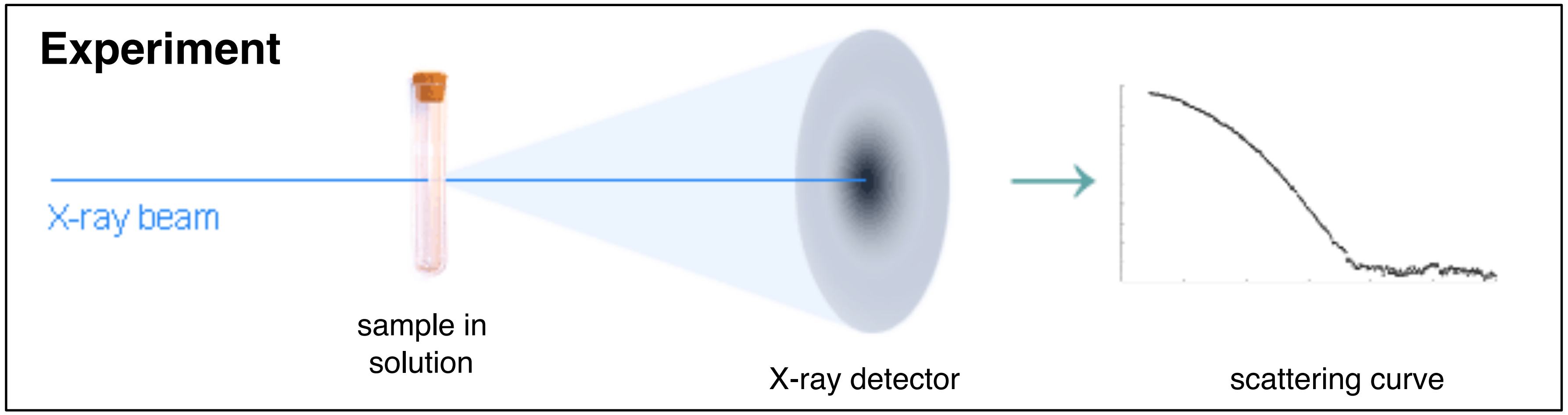
- Given an experimental SAXS profile and a 3D model, FoXS:
  - Calculates the theoretical profile of the model

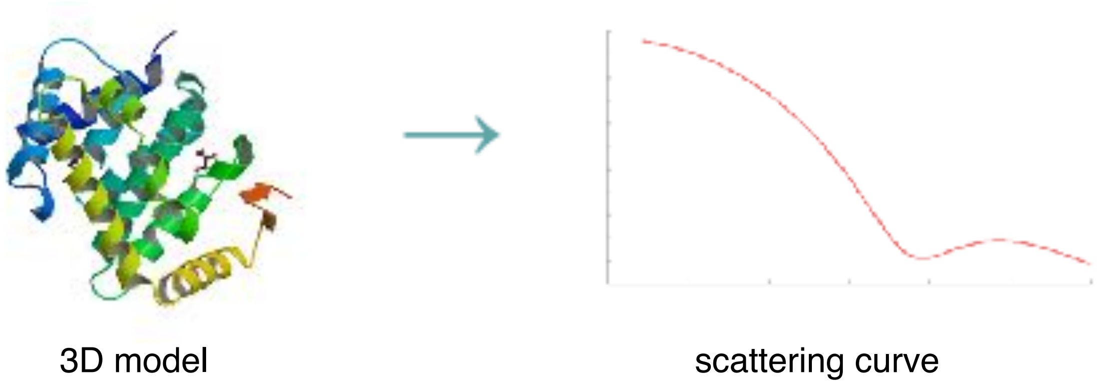




FoXS

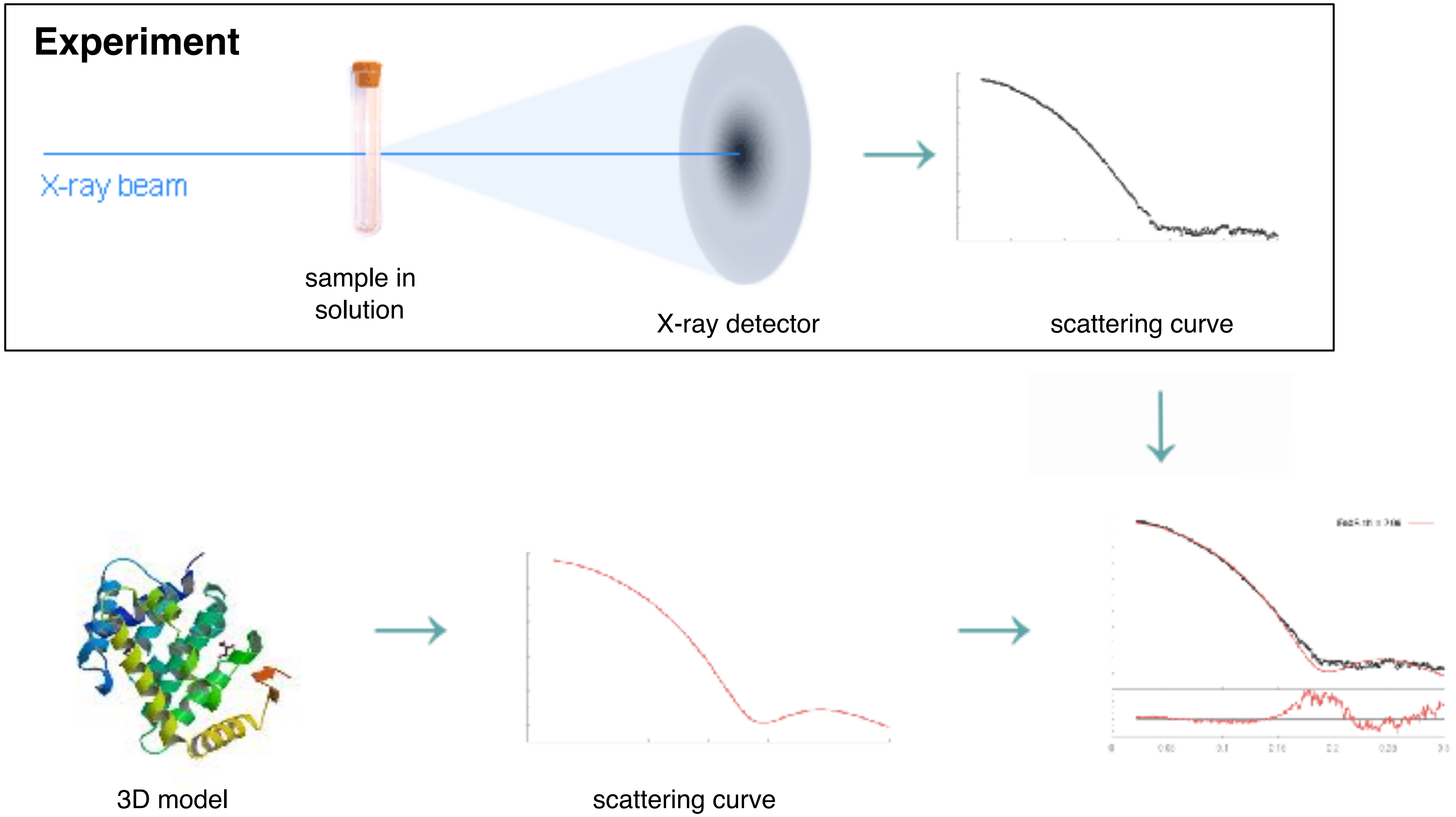
- Given an experimental SAXS profile and a 3D model, FoXS:
  - Calculates the theoretical profile of the model
  - Fits the two profiles together and reports a fit value, χ





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



Experiment

X-ray beam

sample in solution

X-ray detector

scattering curve

3D model

scattering curve

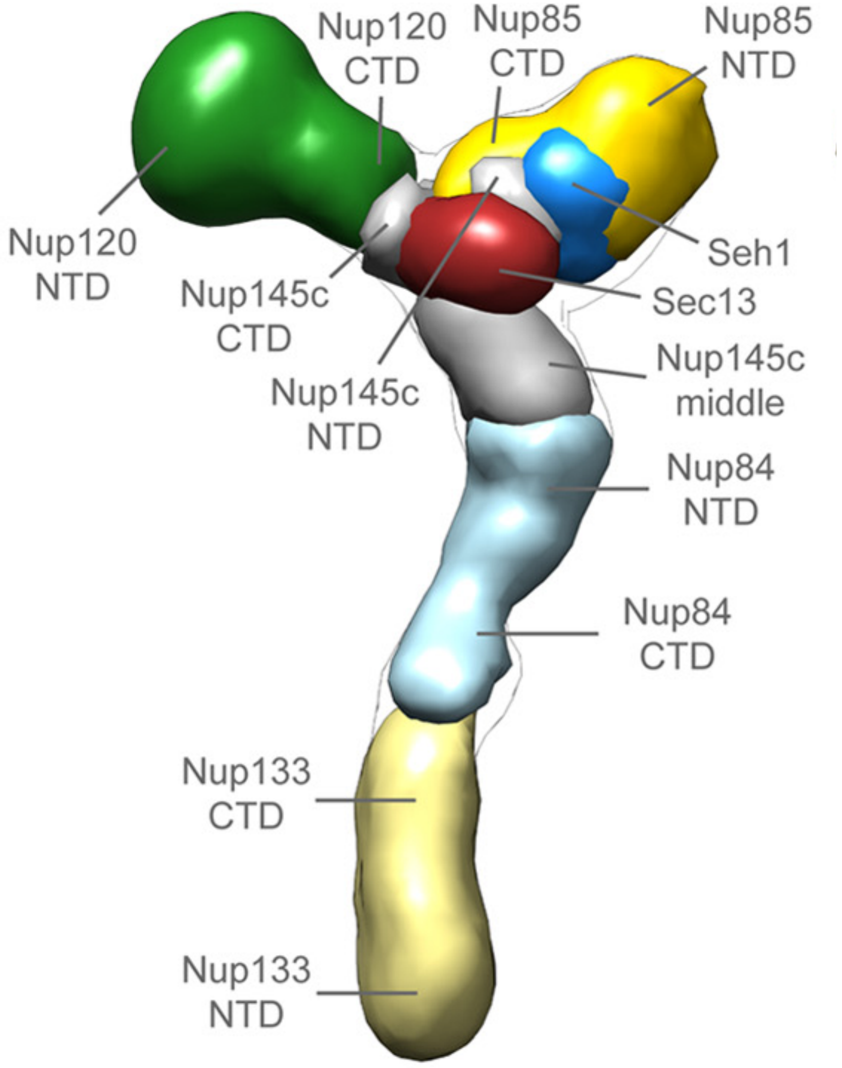
FoXS usage

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 complex

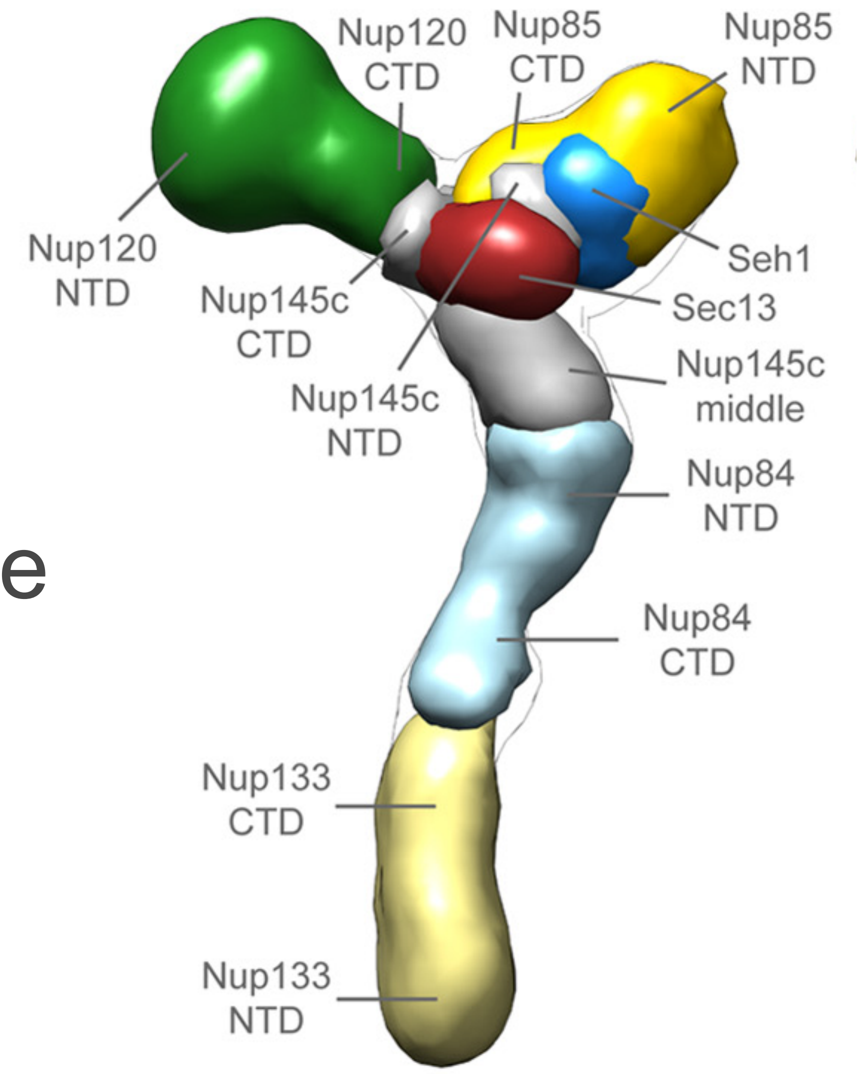
FoXS usage

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



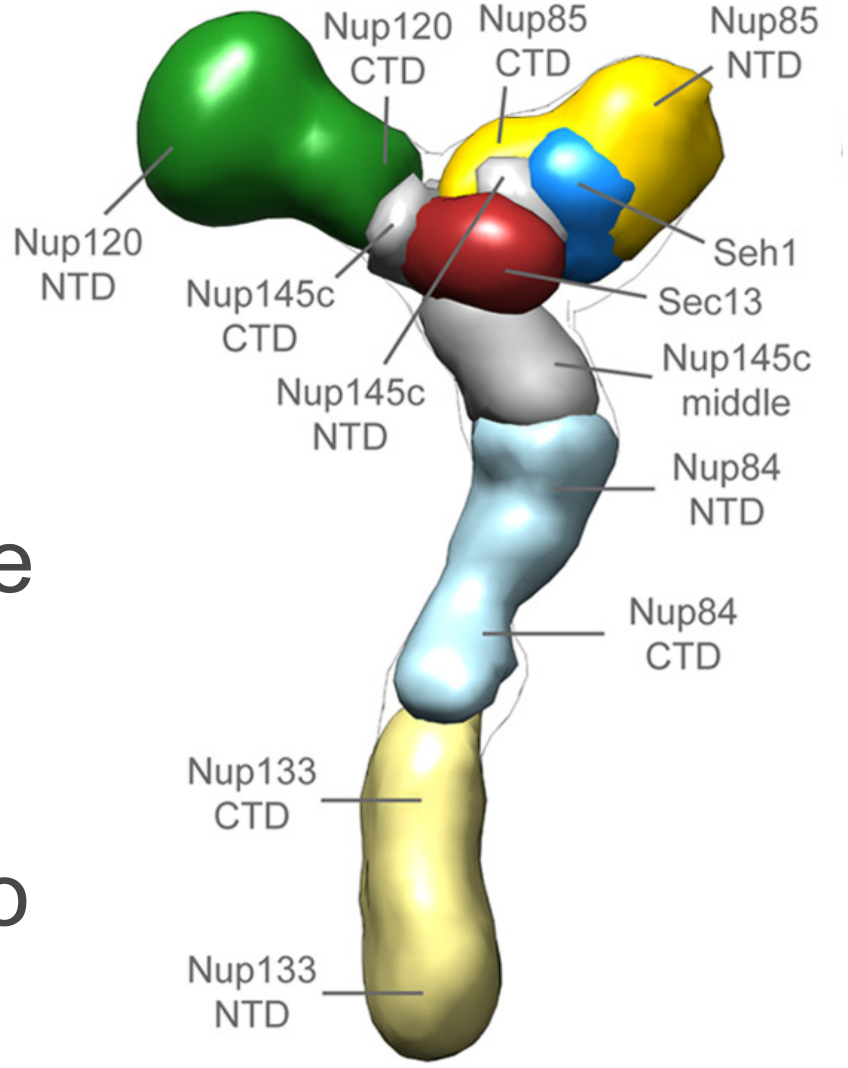
FoXS usage

- Here we'll use FoXS to improve the structure of the C terminal domain of Nup133, one of the subunits of the Nup84 complex
- SAXS is rotationally averaged so we can't *predict* an X-ray-like structure, but we can *check consistency* with an existing structure



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 complex
- SAXS is rotationally averaged so we can't *predict* an X-ray-like structure, but we can *check consistency* with an existing structure
- For the Nup84 study we built structures of the complete Nup133 using comparative modeling since no X-ray structures were available



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 foxs_example
$ cd foxs_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')
```

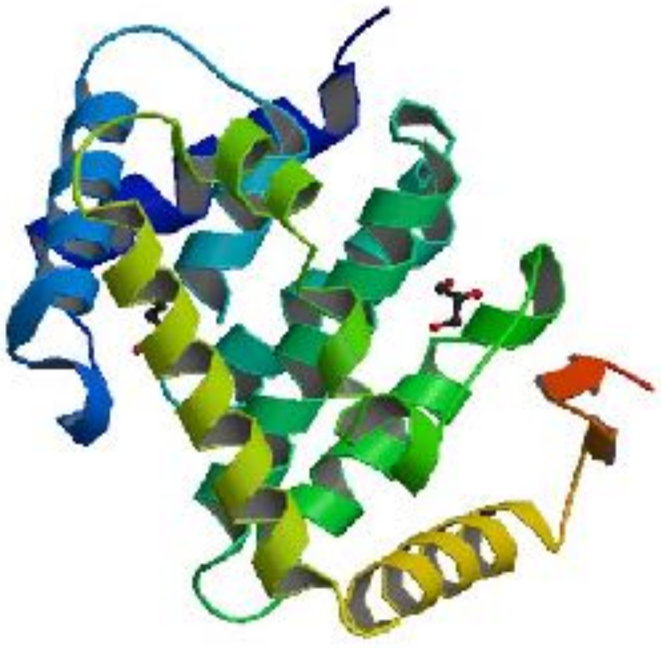
- Then, copy them to your working directory/ folder:

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

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

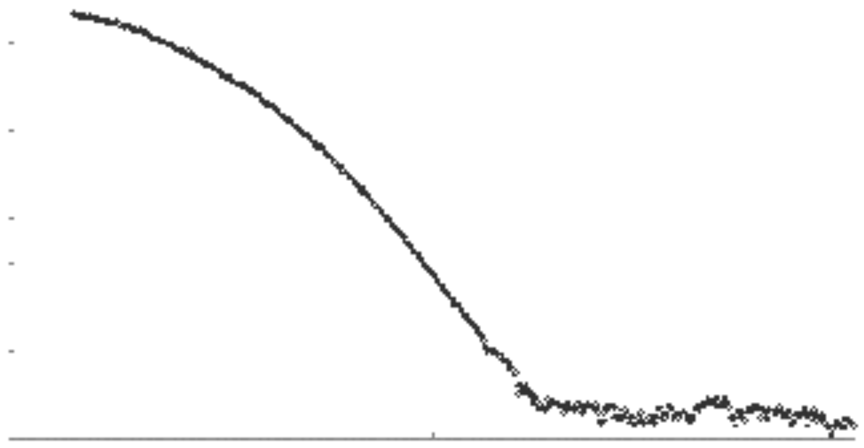
Input files

`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)

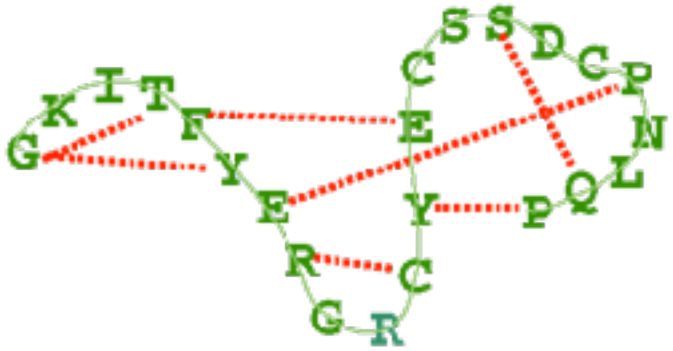
Why such a poor fit?

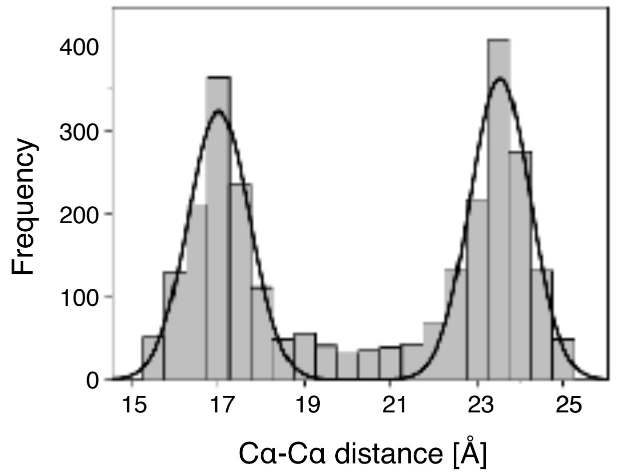
- Both the X-ray structure and the SAXS profile were collected for the same structure, so shouldn’t they match?
- Let’s look at the `3KFO.pdb` file in a text editor (not a molecular viewer), specifically REMARK 465, 470 and 999 lines
  - proteolysis removed residues 881-943 (REMARK 999) so these weren’t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

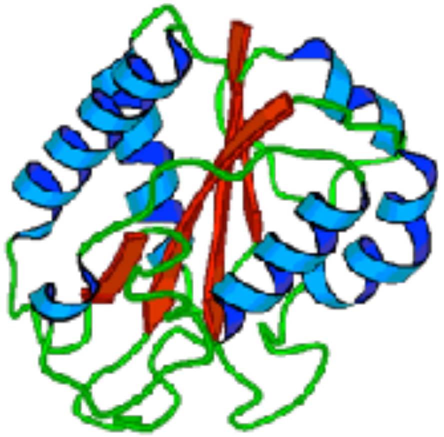
```
3D  GKITFYERGFQGHCYESDC-NLQP…
SEQ GKITFYERG---RCYESDCPNLQP…
```

1. **Extract spatial restraints**





2. **Satisfy spatial restraints**



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

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.B99990005.pdb 23922_merge.dat
$ gnuplot 3KFO-fill.B99990005_23922_merge.plt
```

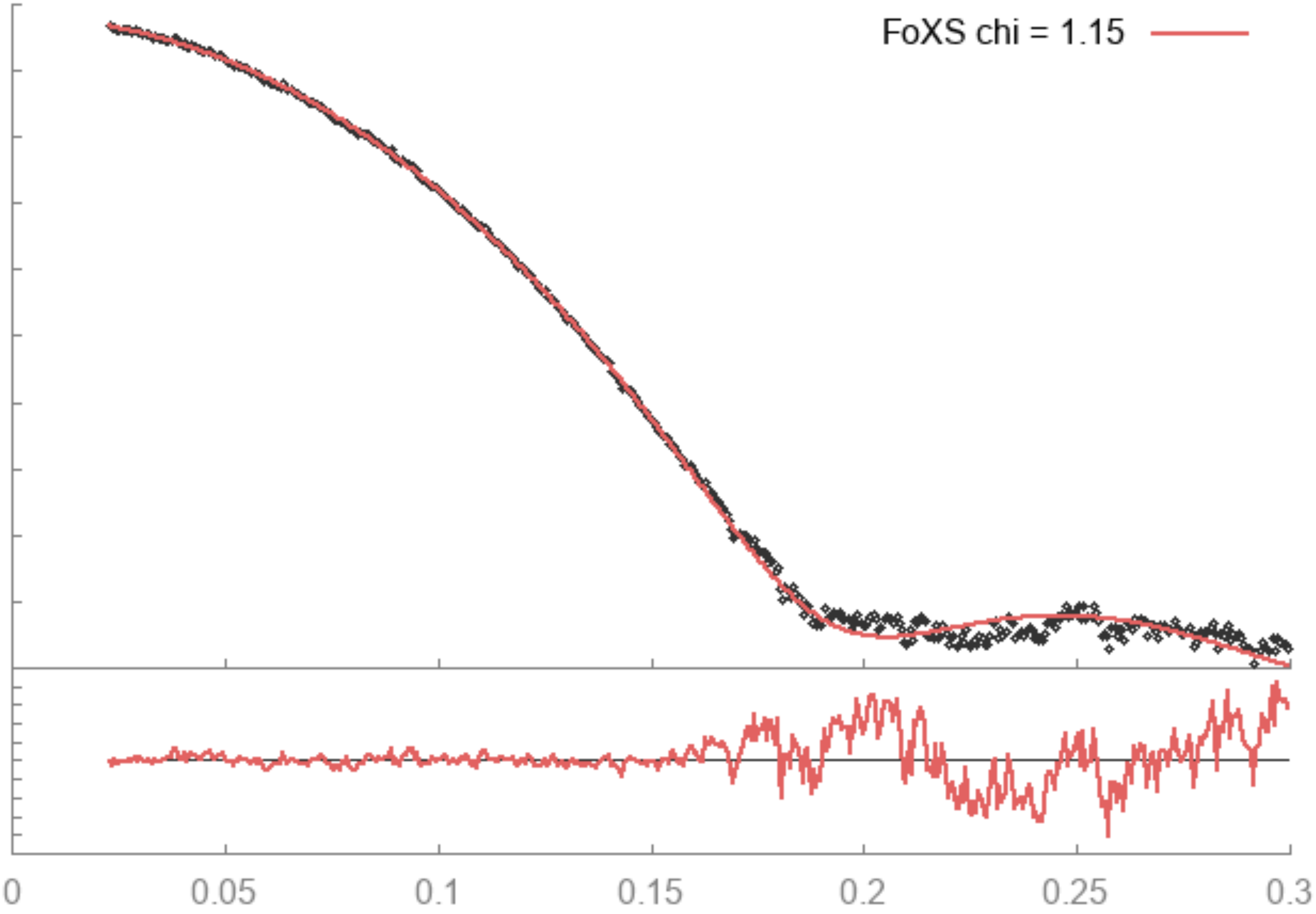
Visualize outputs

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

```
$ foxs -g 3KFO-fill.B99990005.pdb 23922_merge.dat
$ gnuplot 3KFO-fill.B99990005_23922_merge.plt
```

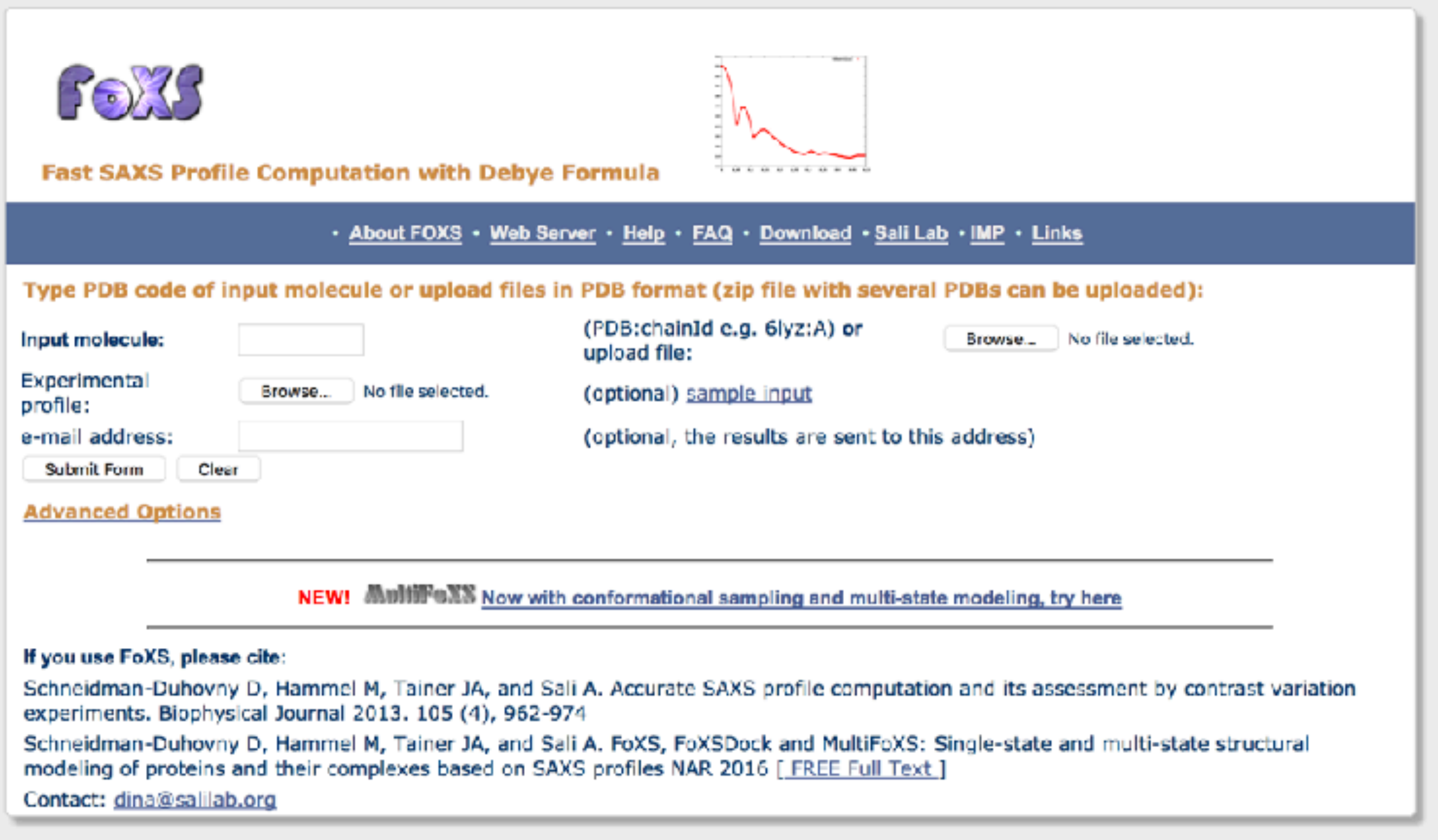
- Look at `3KFO-fill.B99990005_23922_merge.png` in an image viewer

gnuplot output



FoXS web service

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


FoXS
• About FOXS • Web Server • Help • FAQ • Download • Sali Lab • IMP • Links
Browse...
Submit Form
Clear
Advanced Options

Fast SAXS Profile Computation with Debye Formula

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

Input molecule: (PDB:chainId e.g. 6lyz:A) or upload file: No file selected.

Experimental profile: No file selected. (optional) sample input

e-mail address: (optional, the results are sent to this address)

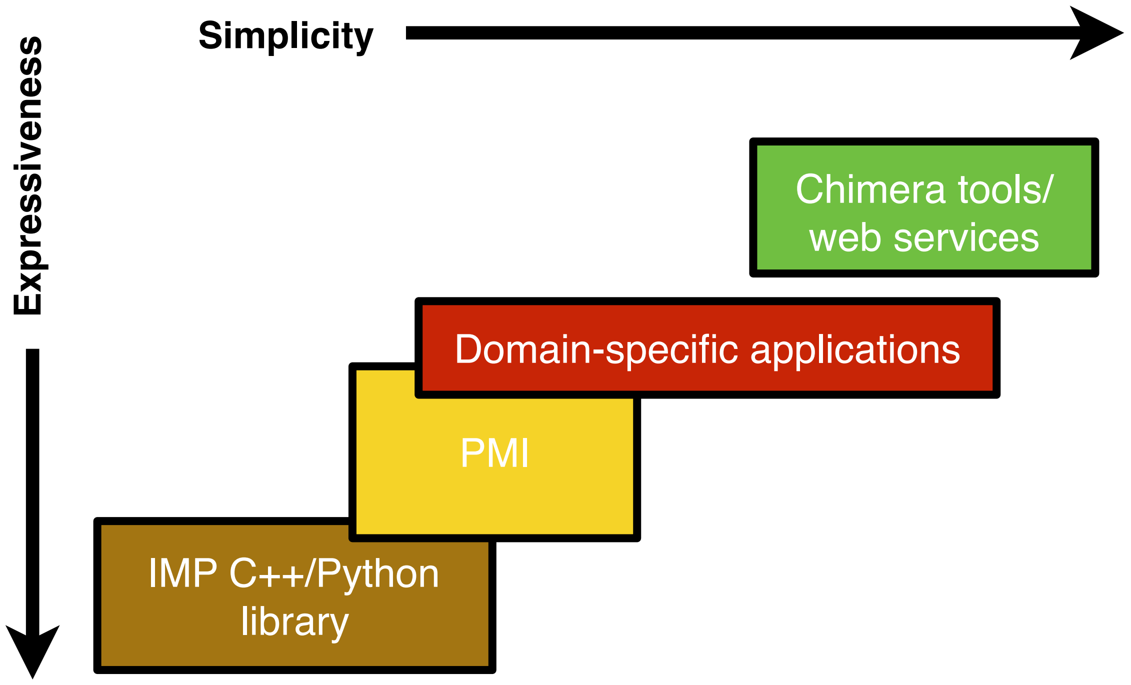
NEW! MultiFoXS Now with conformational sampling and multi-state modeling, try here

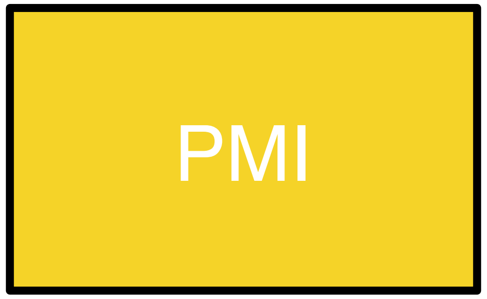
If you use FoXS, please cite:

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


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



PMI

- Just another IMP module (`IMP.pmi`)
- A meta language for modeling
- We still write Python scripts, but…
  - 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
- Today we will use 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

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 *vitally 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 *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

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 *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
- 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

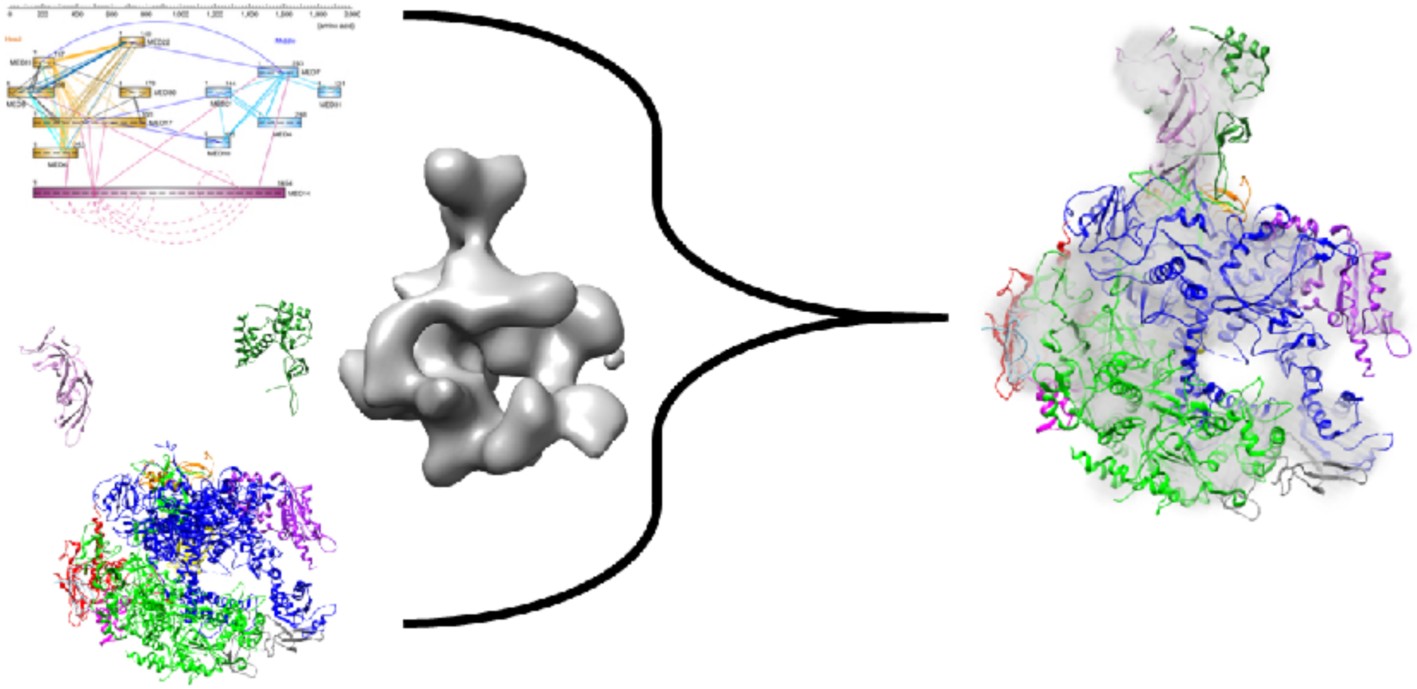
- git tracks who changed what and when (like an electronic lab notebook)
  - often hard to keep modeling protocols organized
  - don't have to use git for this, but *vitally important* to use some kind of change tracking software (e.g. SVN, CVS, hg, etc.)
- git integrates nicely with GitHub which provides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

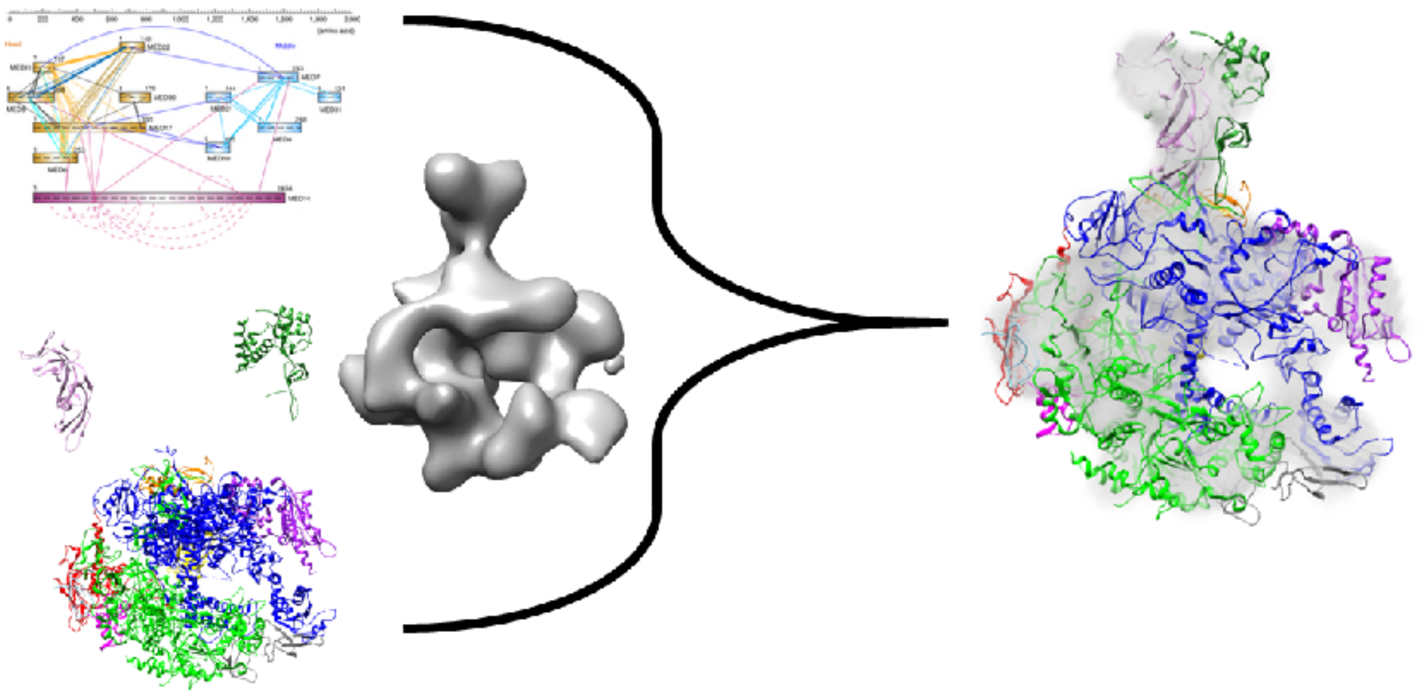
Integrative structure modeling of RNA Polymerase II stalk

- 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 coupled with mass spectrometry (CX-MS),
  - negative-stain electron microscopy (EM),
  - X-ray crystallography data



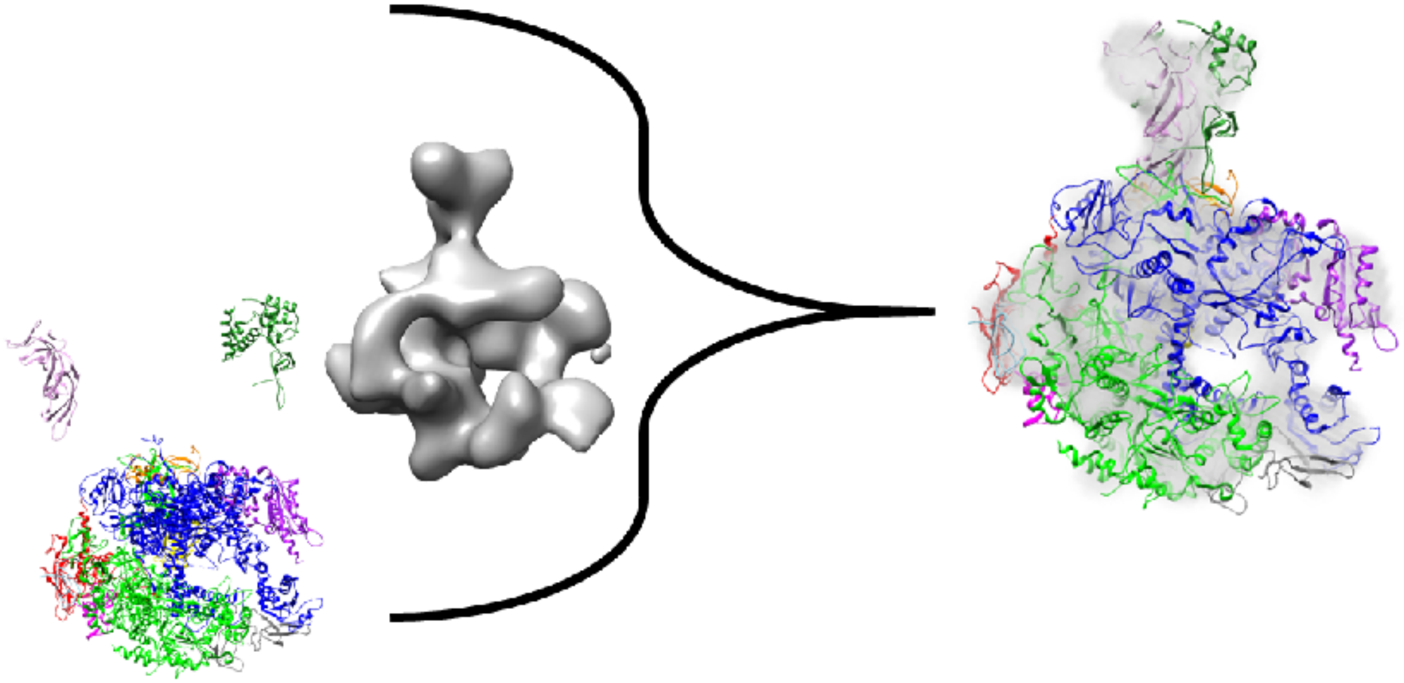
First round: modeling with EM/X-ray only

- For the purposes of demonstration, we’ll first model the complex using *only* the EM and X-ray data



First round: modeling with EM/X-ray only

- For the purposes of demonstration, we'll first model the complex using *only* the EM and X-ray data



Main modeling script

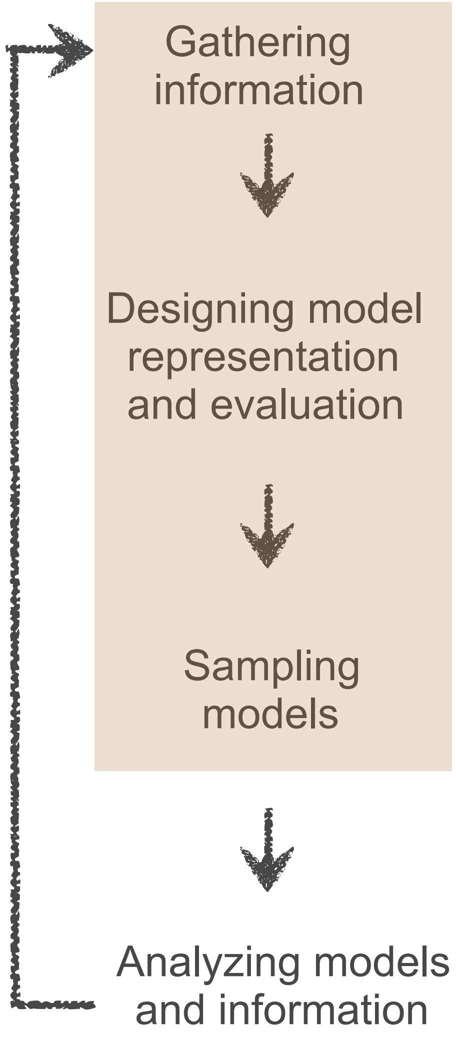
- Let's get started by getting the main modeling script running while we look at what it's doing

- Do this by running in a terminal/command prompt:

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

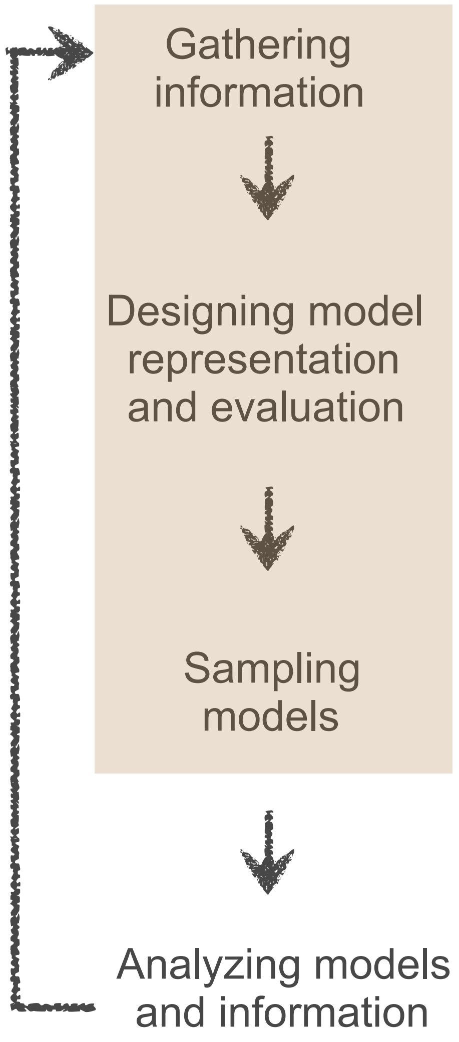
- "Real" modeling will take hours, so we're running in 'test' mode which generates only 100 frames (rather than 20,000)

- The script covers the first 3 steps of integrative modeling


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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
```

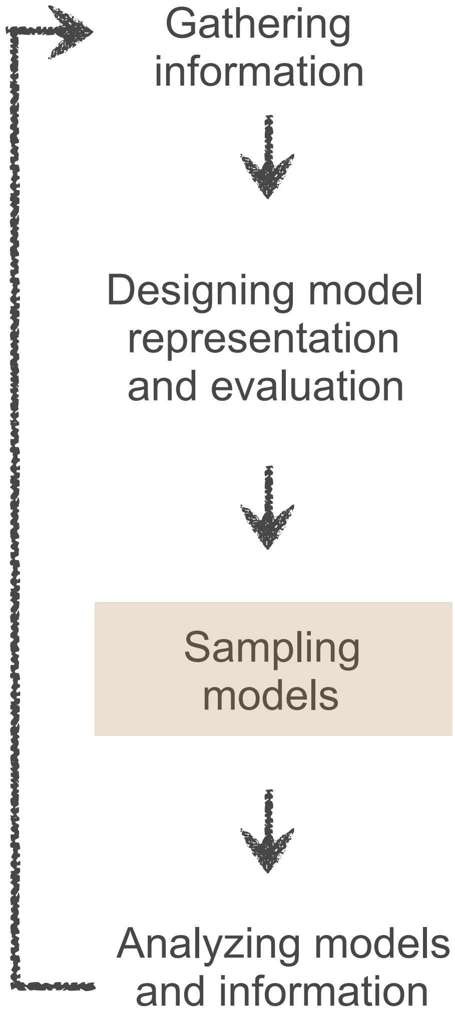

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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)



Gathering information

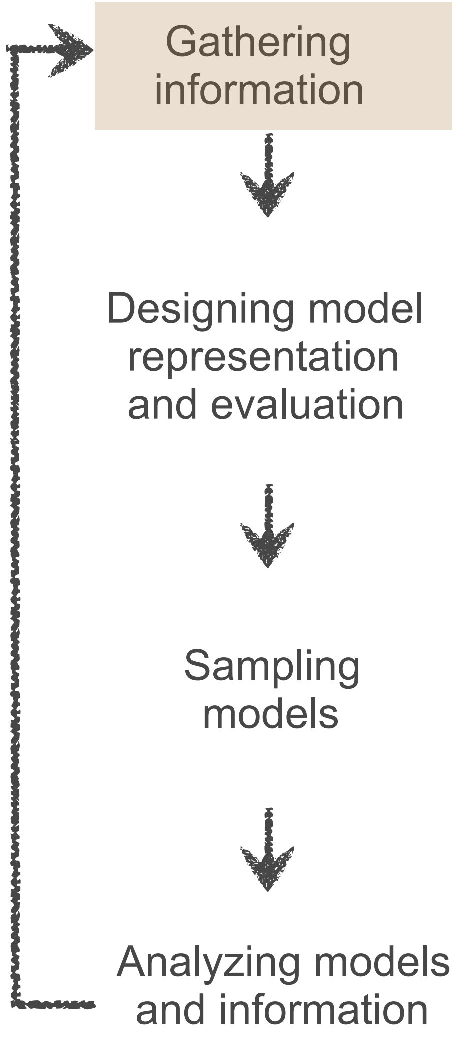
Designing model representation and evaluation

Sampling models

Analyzing models and information

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)
- 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



Gathering information

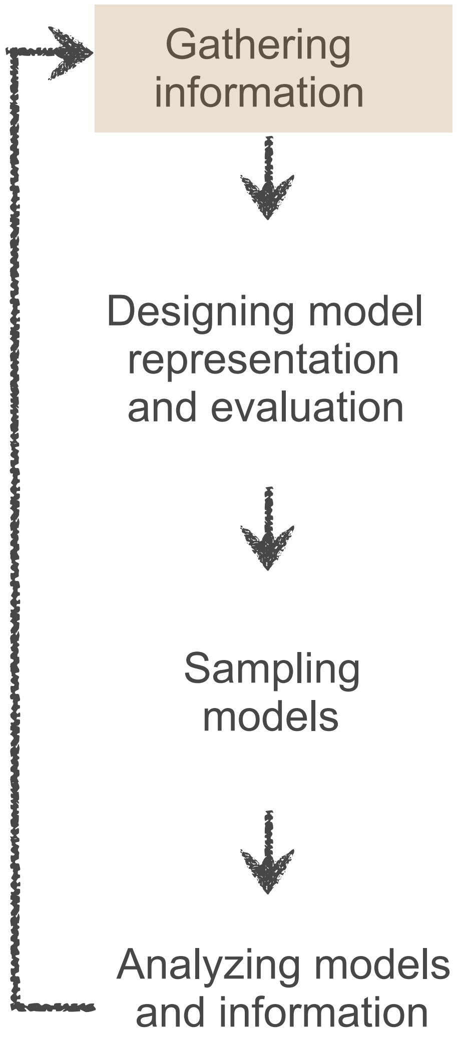
Designing model representation and evaluation

Sampling models

Analyzing models and information

UCSF Chimera

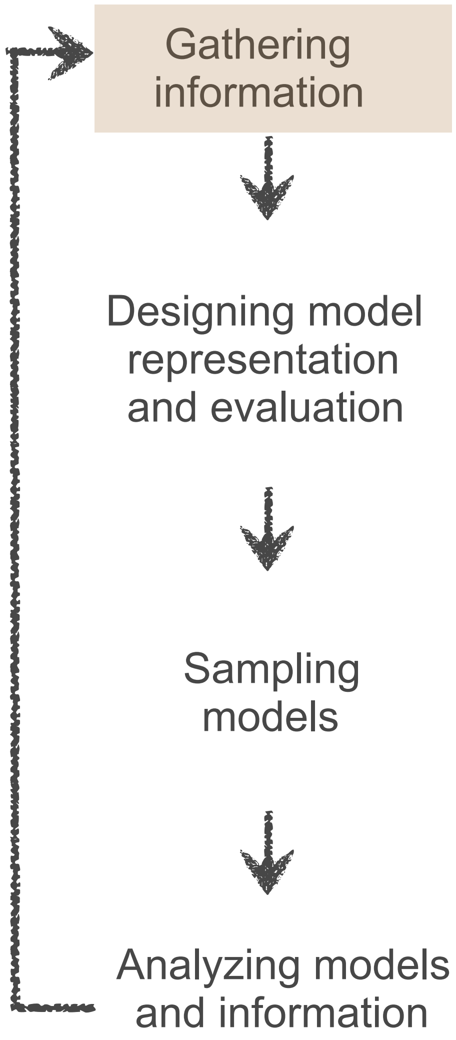
- We use both VMD and UCSF Chimera in our work, but we're using Chimera in this tutorial because
  - some of the file formats we generate are understood only by Chimera (for now)
  - new IMP features generally work with Chimera first (since the Chimera guys are just down the hall from us)
- Feel free to visualize standard formats (such as PDB) in your favorite viewer!


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

FASTA file

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

```
>1WCM:A
MVGQQYSSAPLRTVKEVQFGLFSPEEVRAISVAKIRFPETMDETQTRAKIGG
LNDPRLGSIDRNLKCQTCQEGMNECPGHFGHIDLAKPVFHVGFIAKIKKVCE
CVCMHCGKLLLDEHNELMRQALAIKDSKKRFAAIWTLCKTKMVCETDVPSED
...
>1WCM:B
MSDLANSEKYYDEDPYGFEDESAPITAEDSWAVISAFFREKGLVSQQLDSFN
QFVDYTLQDIICEDSTLILEQLAQHTTESDNISRKYEISFGKIYVTKPMVNE
SDGVTHALYPQEARLRNLTYSSGLFVDVKKRTYEAIDVPGRELKYELIAEES
...
```



Gathering information

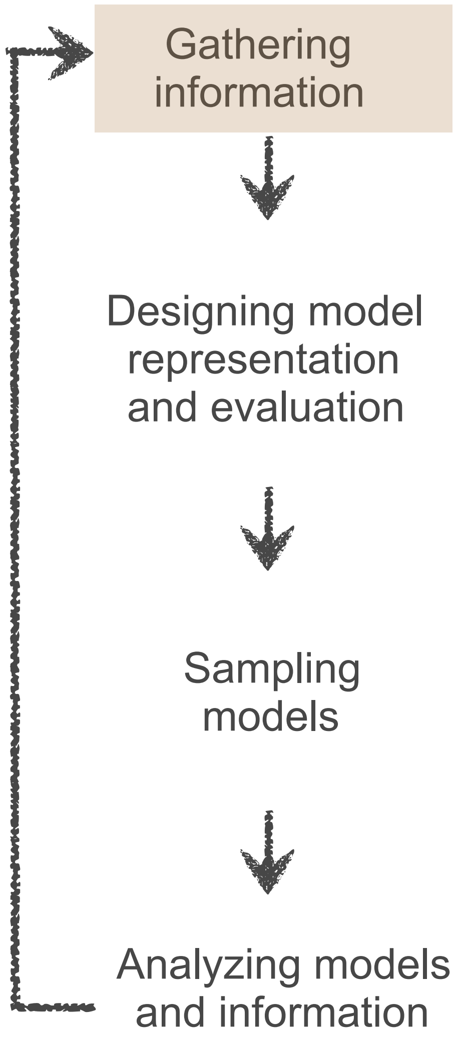
Designing model representation and evaluation

Sampling models

Analyzing models and information

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

Electron density map



Gathering information

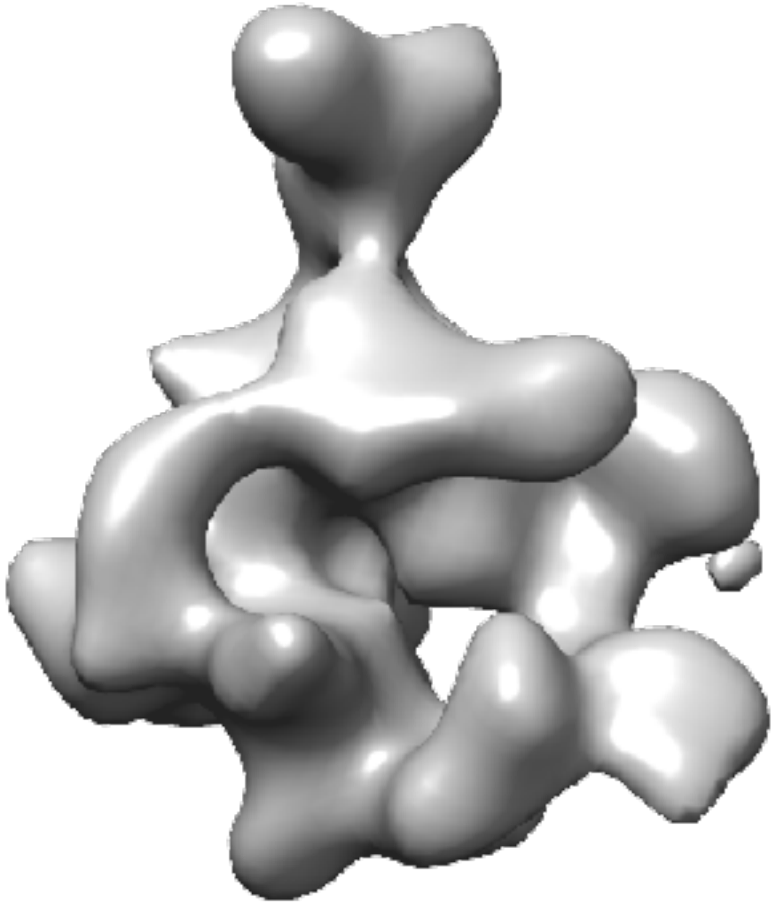
Designing model representation and evaluation

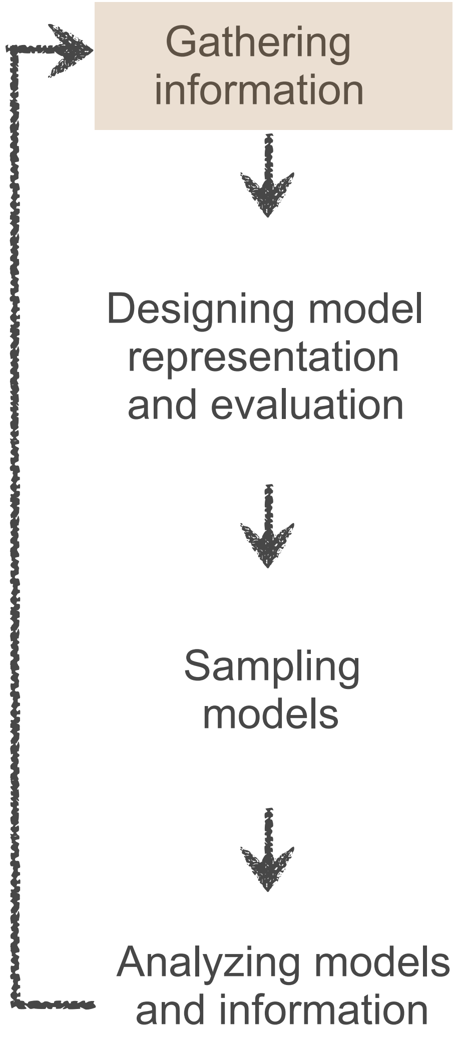
Sampling models

Analyzing models and information

Electron density map

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





Gathering information

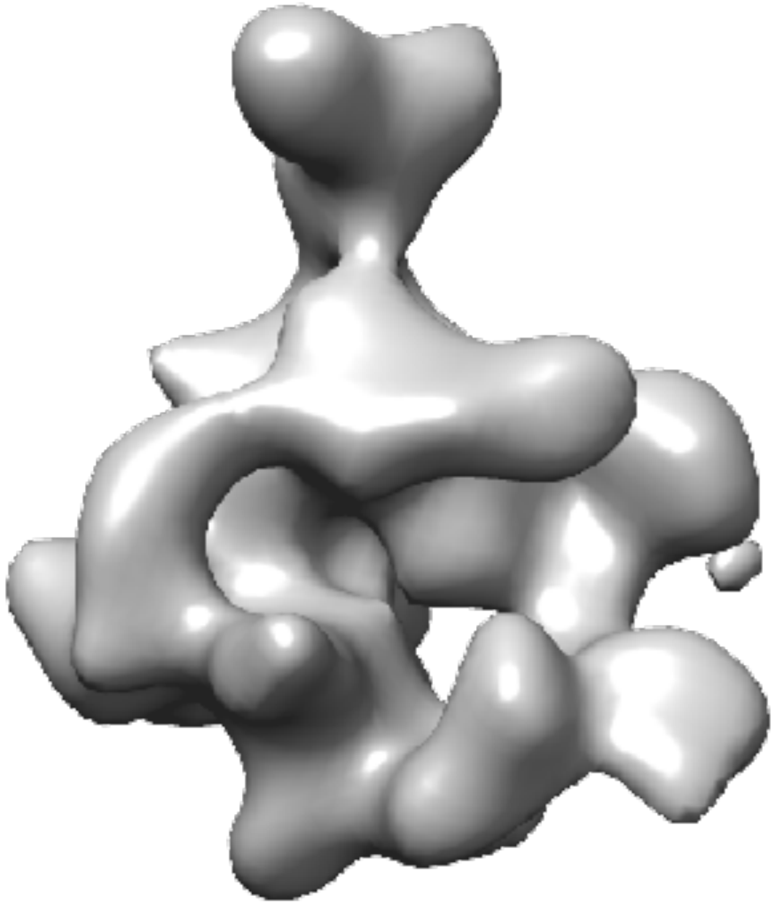
Designing model representation and evaluation

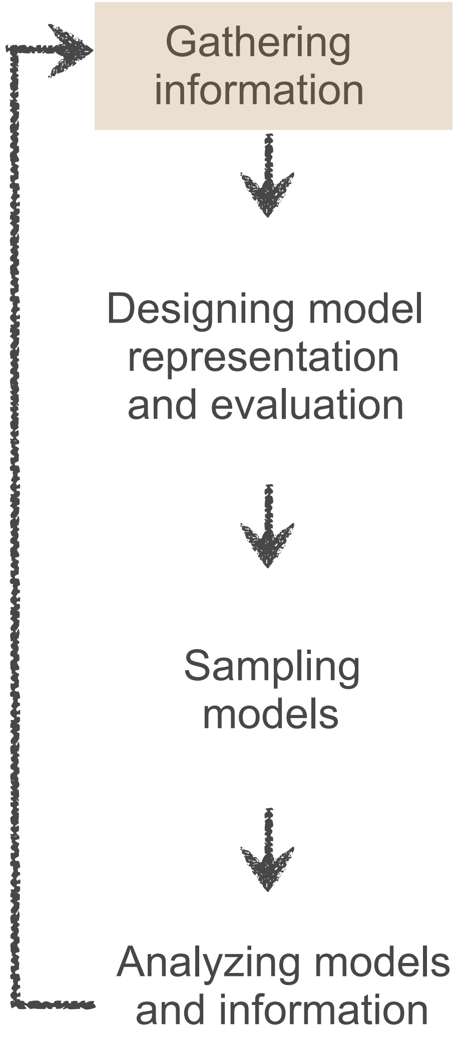
Sampling models

Analyzing models and information

Electron density map

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





Gathering information

Designing model representation and evaluation

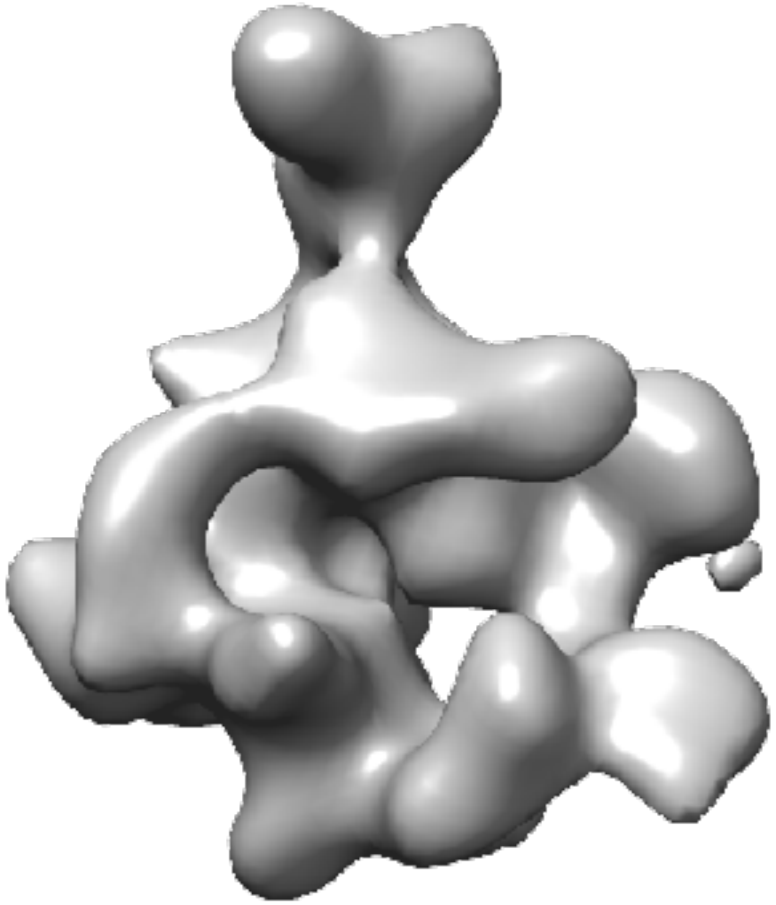
Sampling models

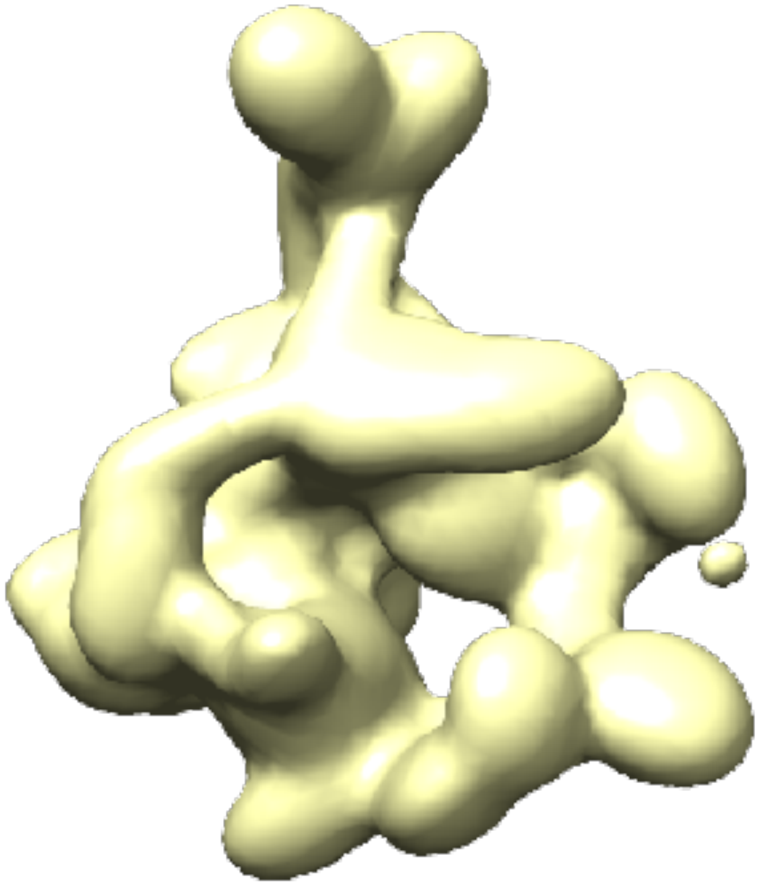
Analyzing models and information

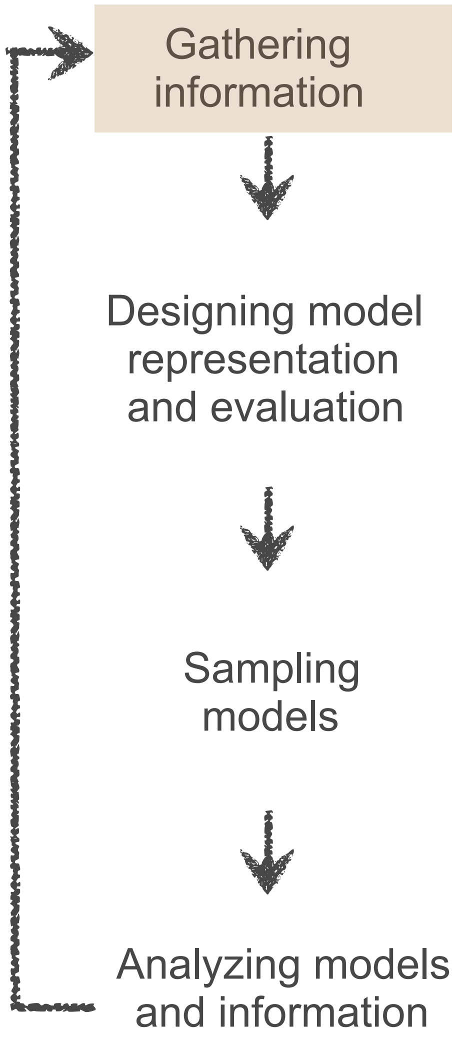
Gaussian mixture models (GMMs) are used to greatly speed up scoring by approximating the electron density of individual subunits and experimental EM maps as a sum of 3D Gaussians. The weight, center, and covariance matrix of each Gaussian used to approximate the original EM density can be seen in *emd_1883.map.mrc.gmm.50.txt*

Electron density map

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







Gathering information

Designing model representation and evaluation

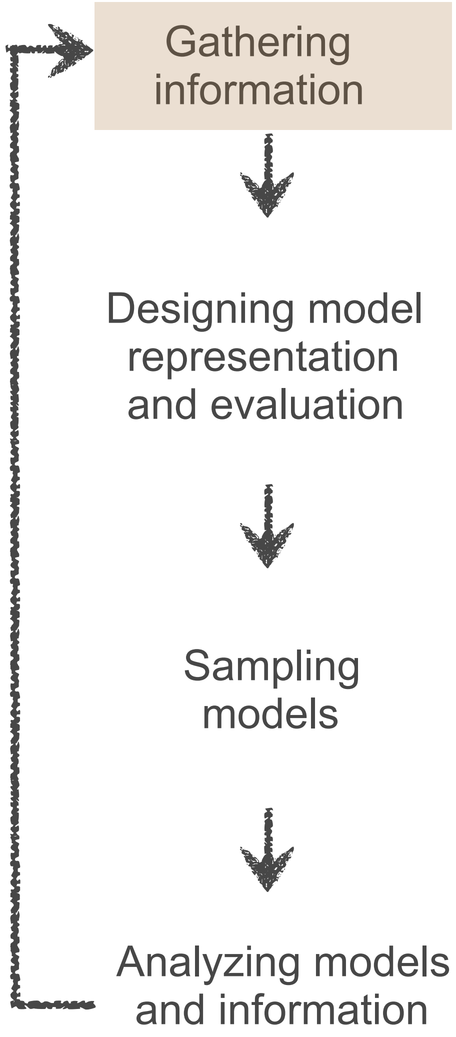
Sampling models

Analyzing models and information

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

X-ray structures

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



Gathering information

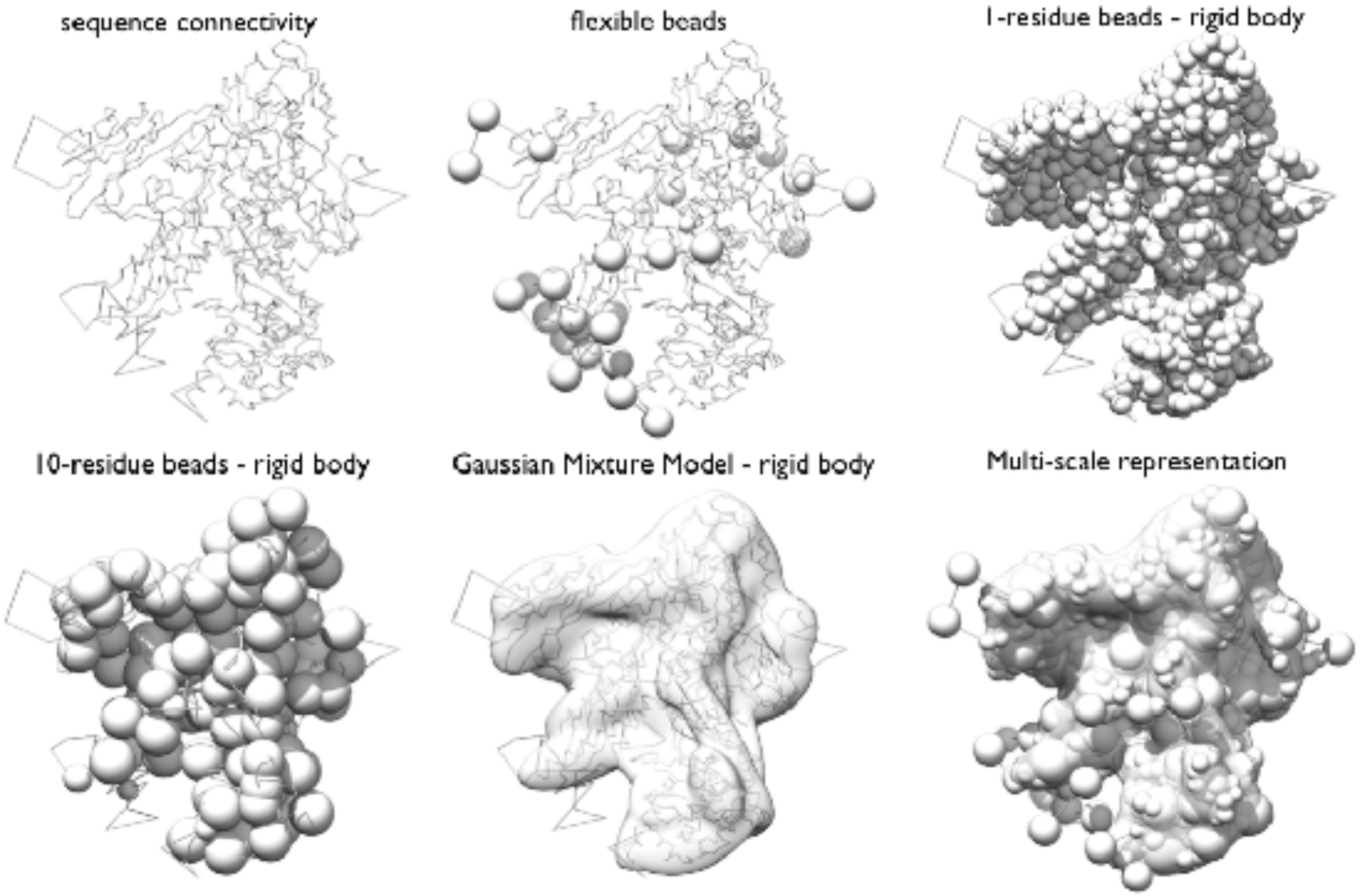
Designing model representation and evaluation

Sampling models

Analyzing models and information

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*



sequence connectivity

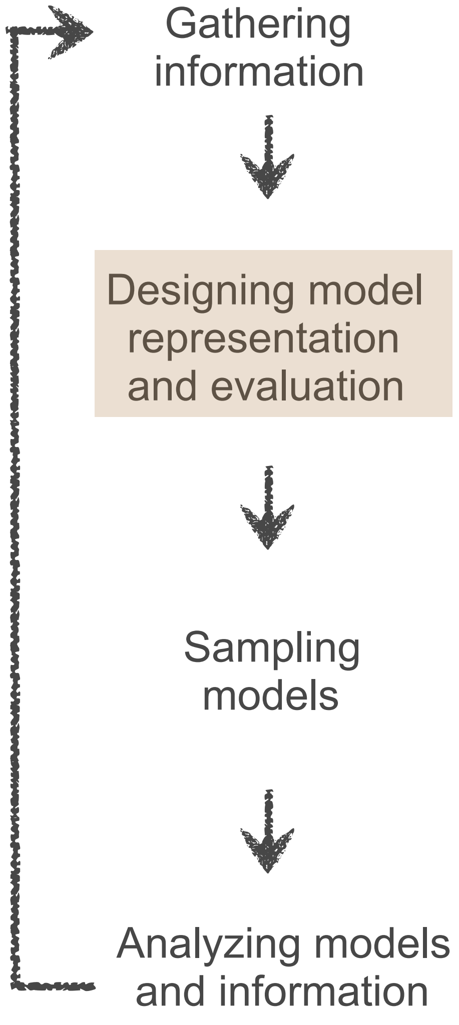
flexible beads

1-residue beads - rigid body

10-residue beads - rigid body

Gaussian Mixture Model - rigid body

Multi-scale representation



Gathering information

Designing model representation and evaluation

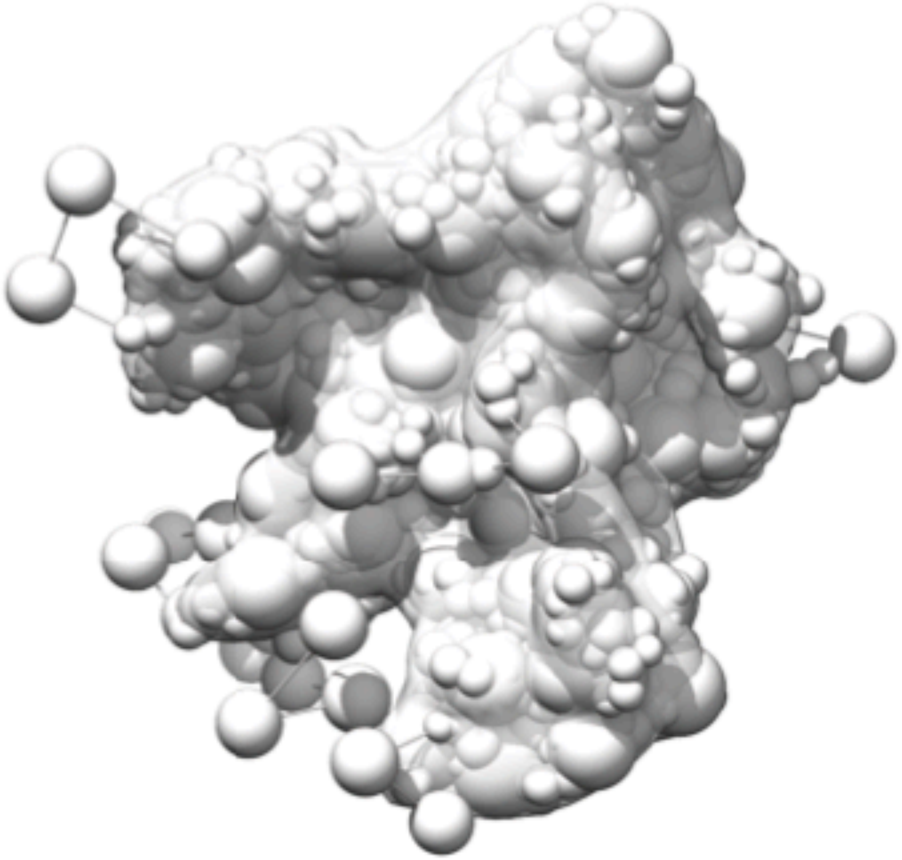
Sampling models

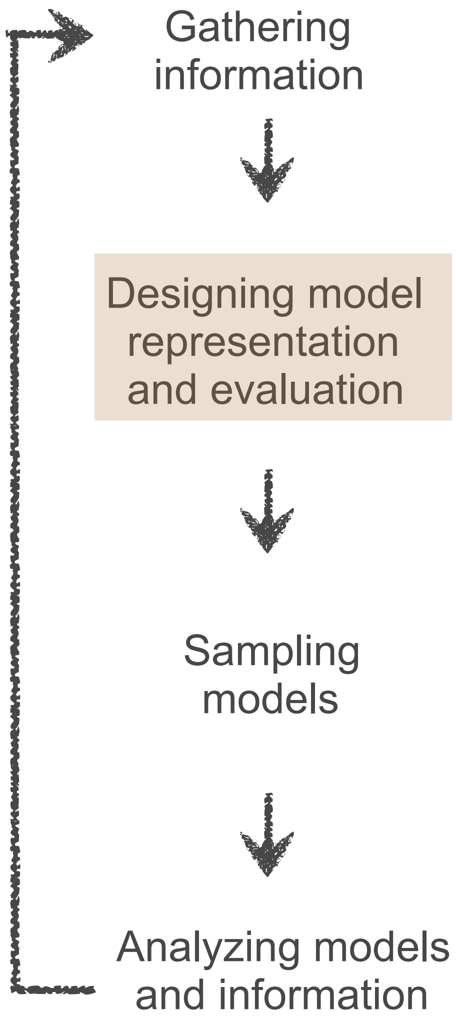
Analyzing models and information

Model representation in IMP

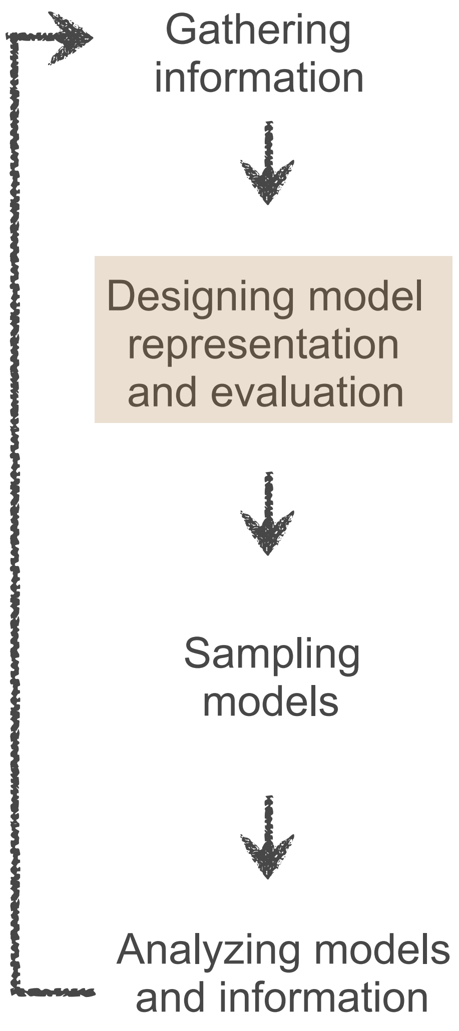
- Note that our representation is **multi-scale**
- i.e. we use both low resolution **and** high resolution bead and Gaussian representations of the model **simultaneously** (“resolution 1”; 1 residue per spherical bead, and “resolution 20”: 20 residues per bead)
- Restraints are applied to the most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

Multi-scale representation




Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Handling of missing structure



Gathering information

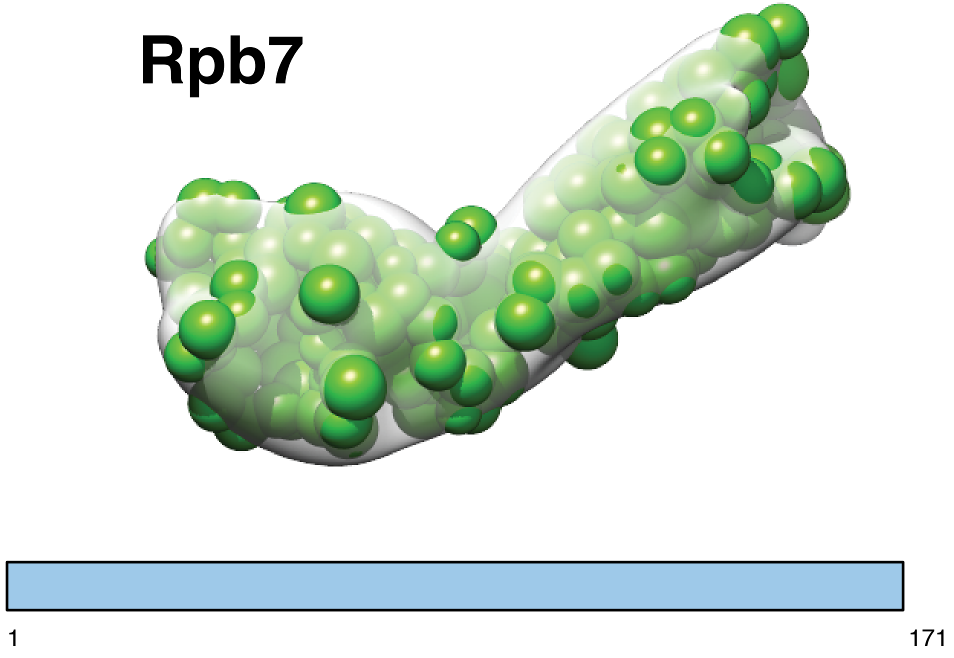
Designing model representation and evaluation

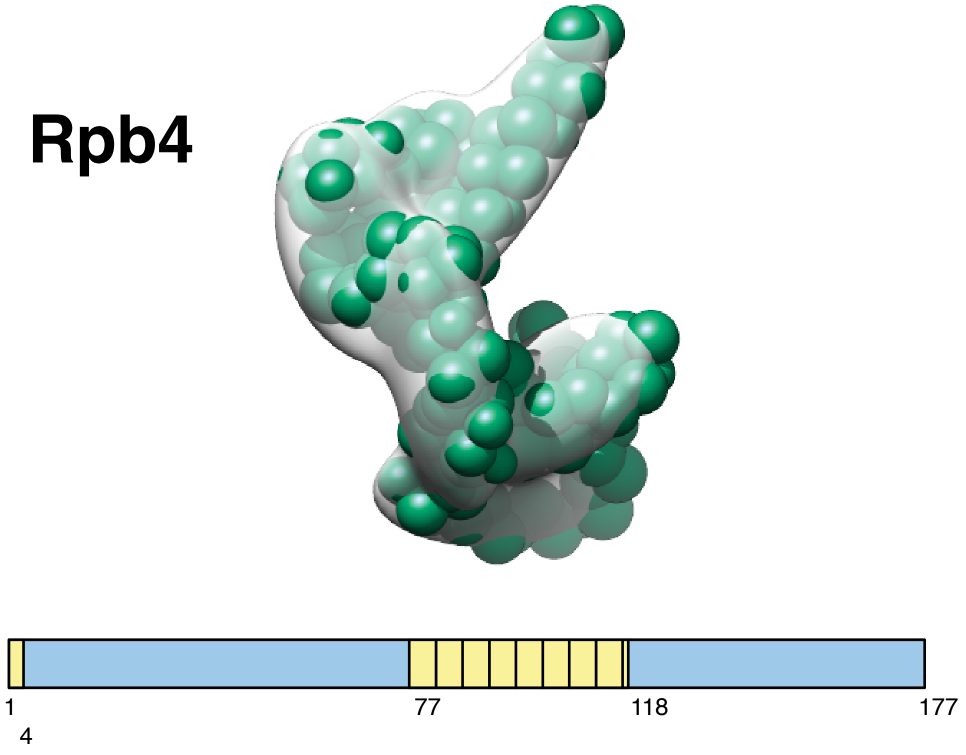
Sampling models

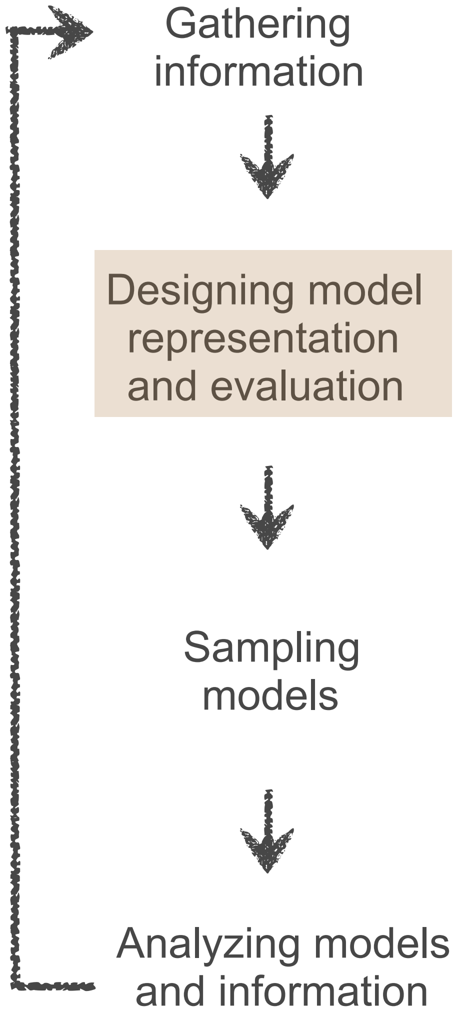
Analyzing models and information

Handling of missing structure

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






Gathering information

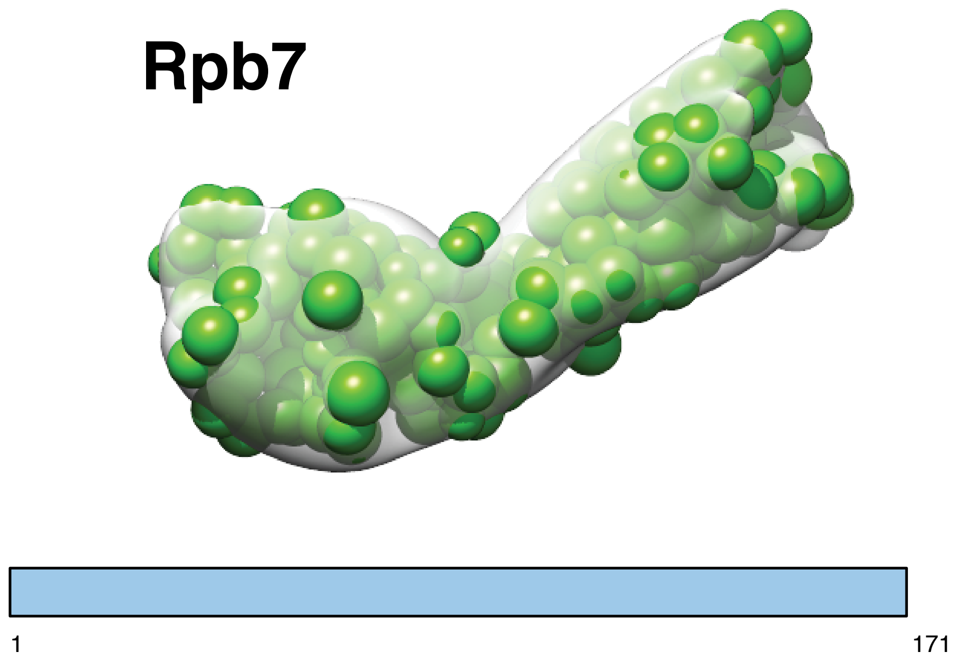
Designing model representation and evaluation

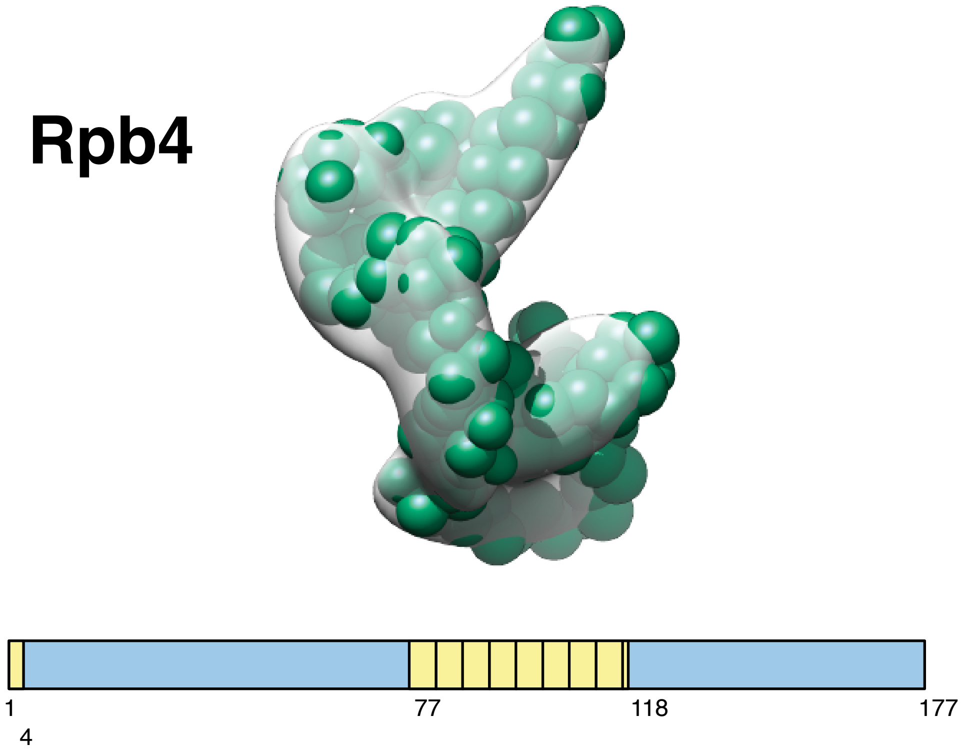
Sampling models

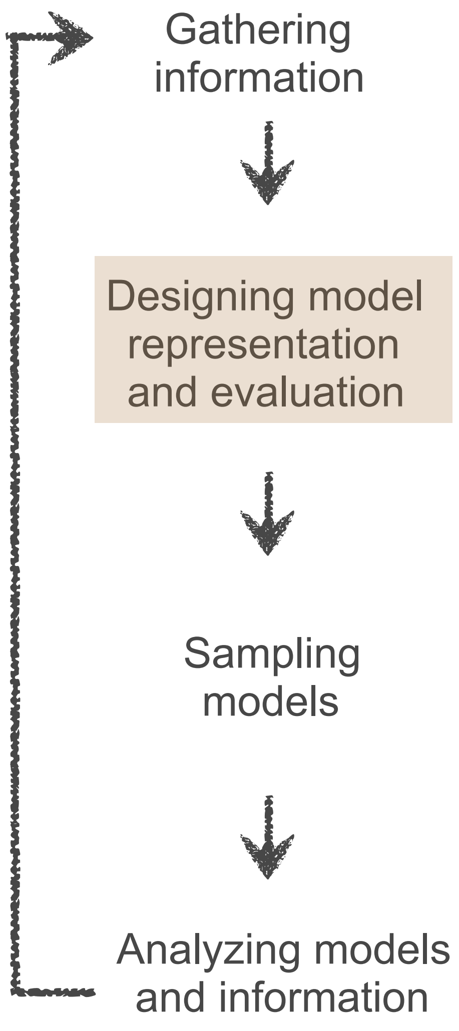
Analyzing models and information

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







Gathering information

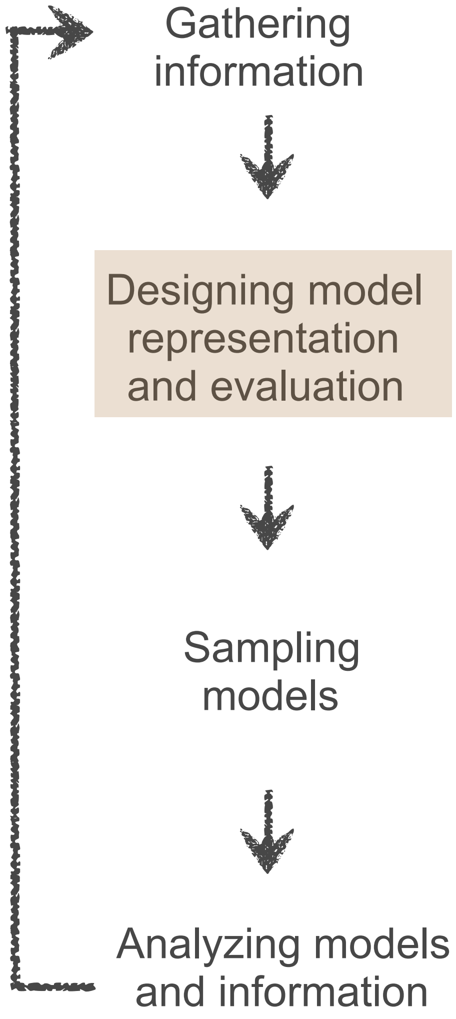
Designing model representation and evaluation

Sampling models

Analyzing models and information

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

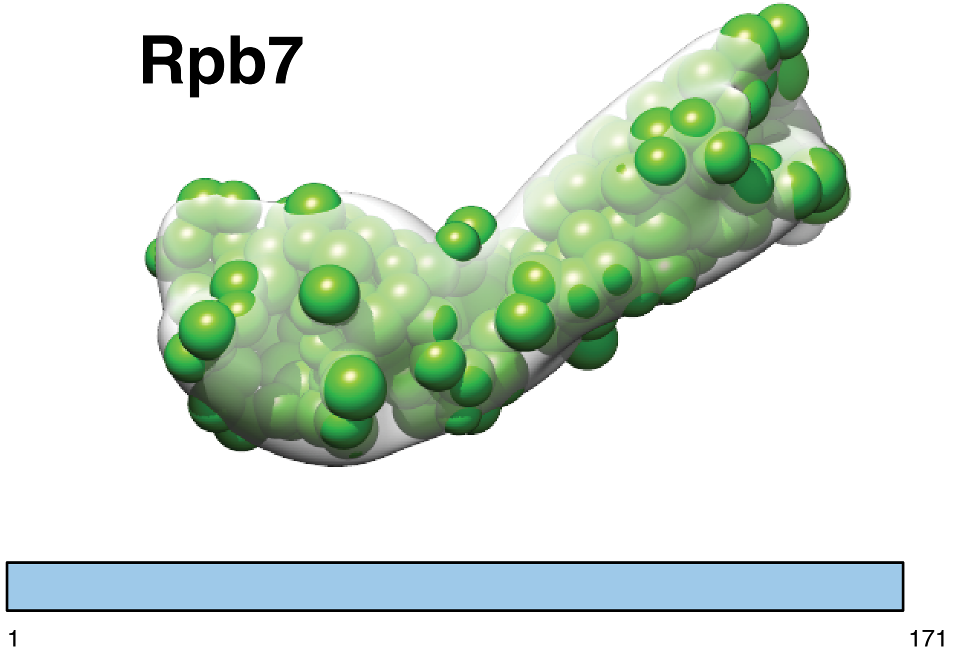

Gathering information

Designing model representation and evaluation

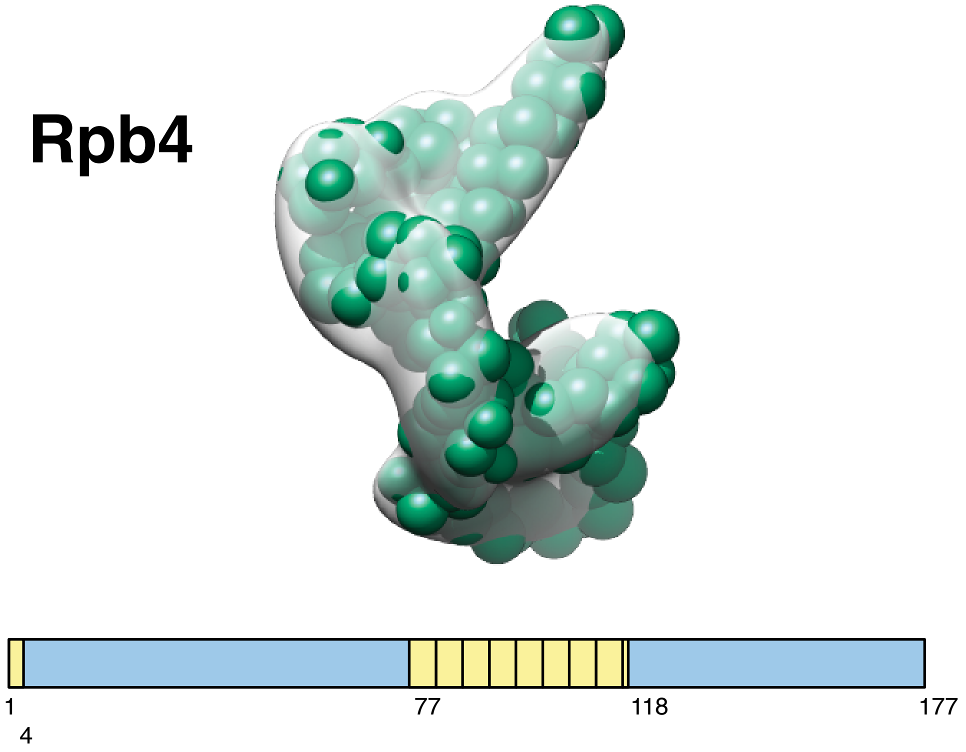
Sampling models

Analyzing models and information

Rpb7

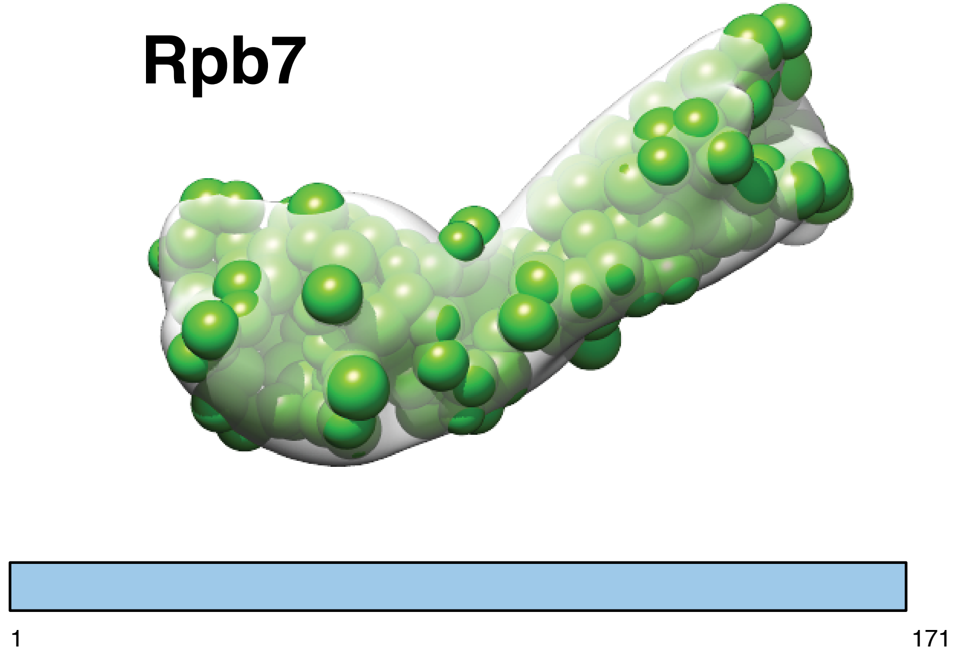


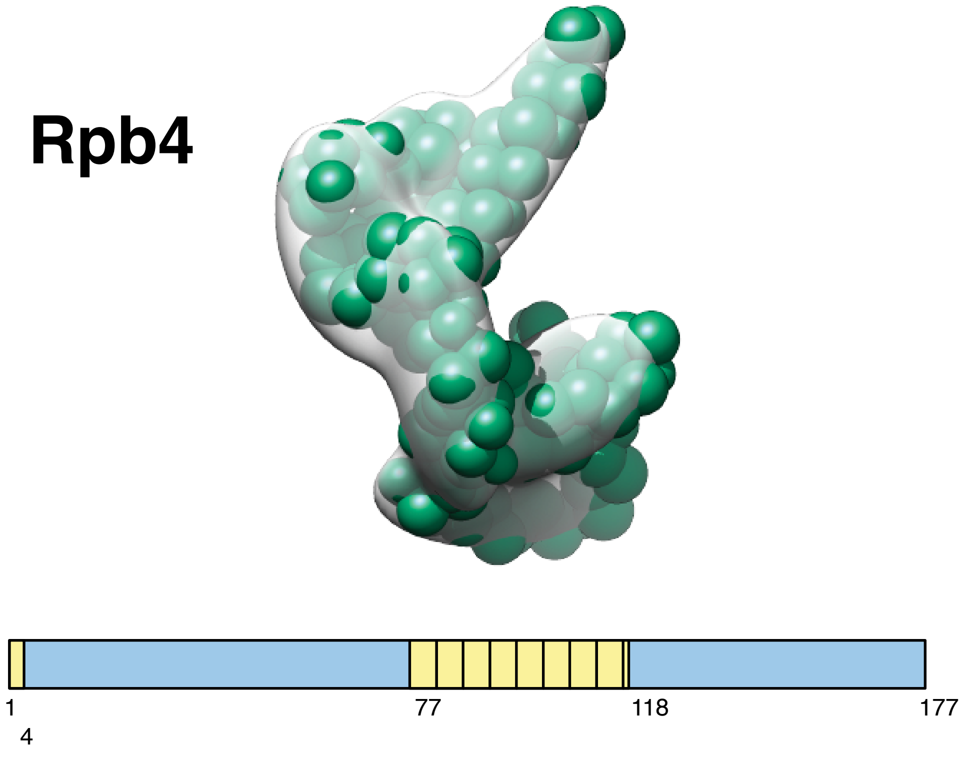
Rpb4

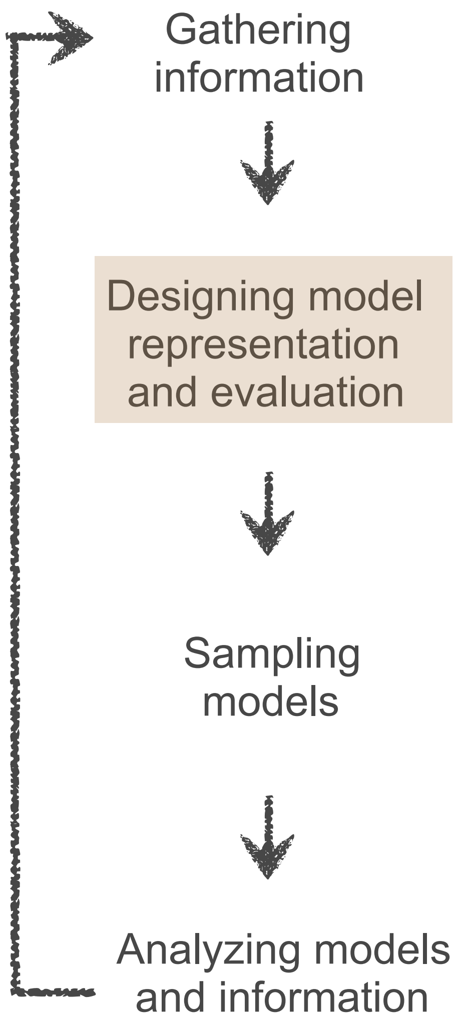


Handling of missing structure

- 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)



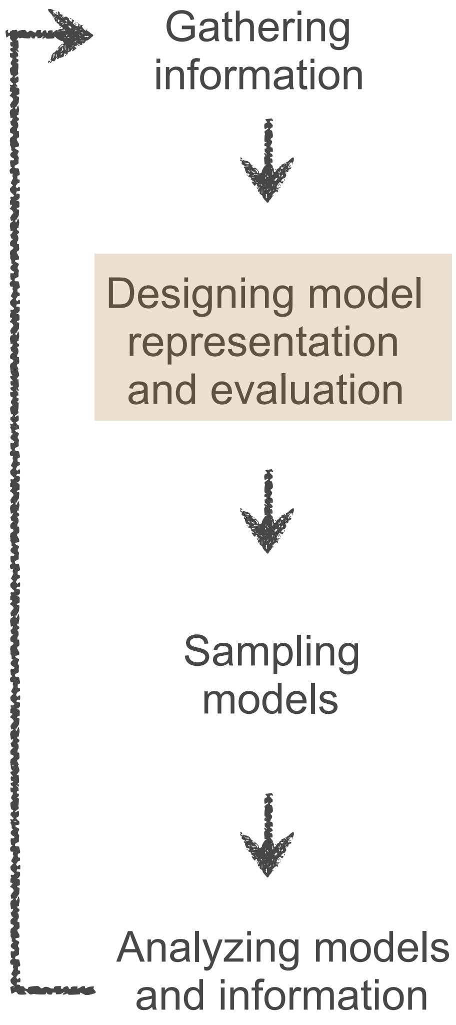



Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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|
```


Gathering information

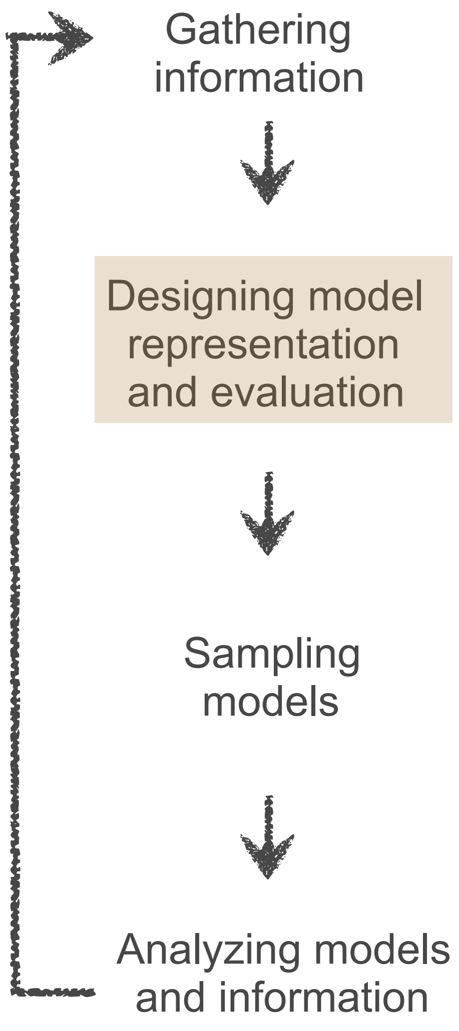
Designing model representation and evaluation

Sampling models

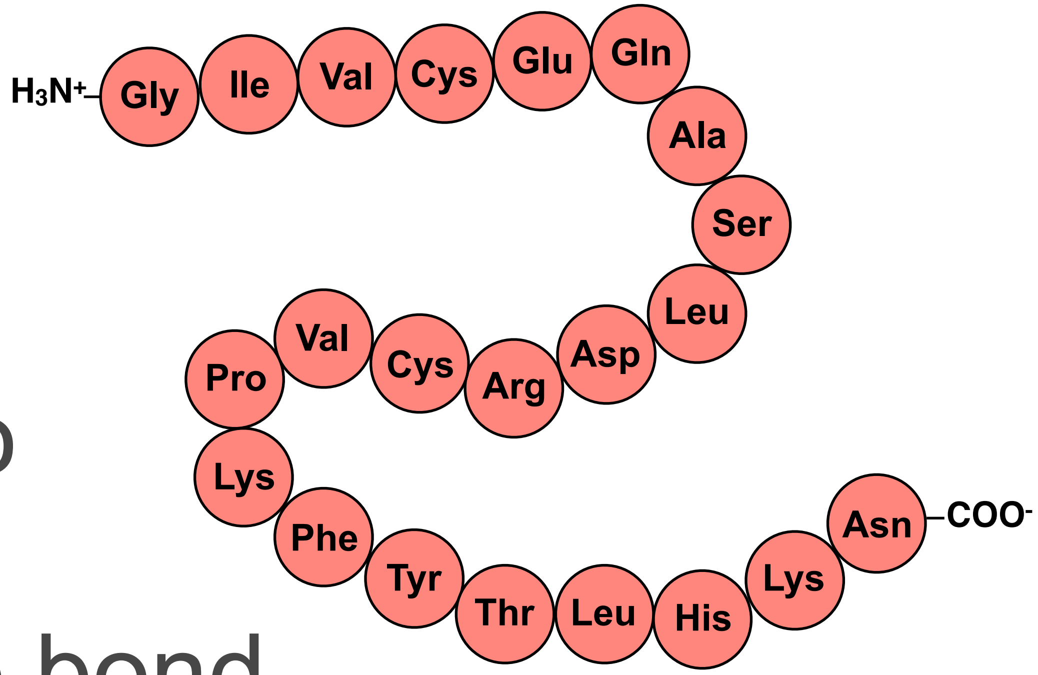
Analyzing models and information

Evaluation

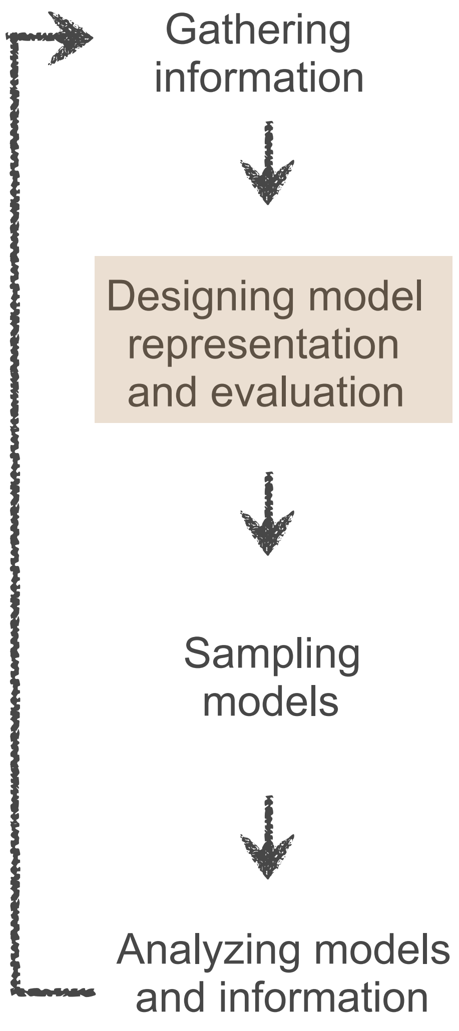
- At this point we need to create our scoring function, by which the individual structural models will be scored based on the input data
- A simple sum of individual restraints
- Each restraint maps to one of our input experiments or other physical/statistical information
- We'll look at each restraint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


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Sequence connectivity restraint



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


Gathering information

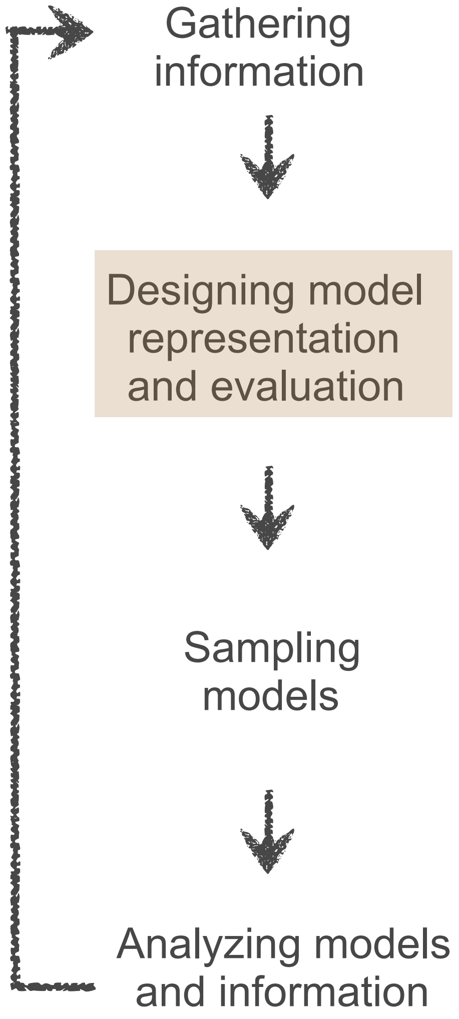
Designing model representation and evaluation

Sampling models

Analyzing models and information

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



Gathering information

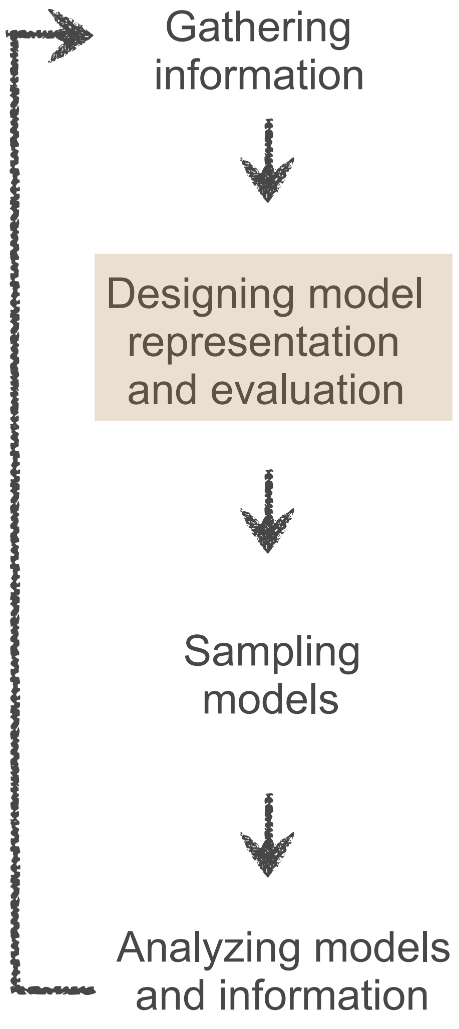
Designing model representation and evaluation

Sampling models

Analyzing models and information

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

PMI vs. core IMP restraints



Gathering information

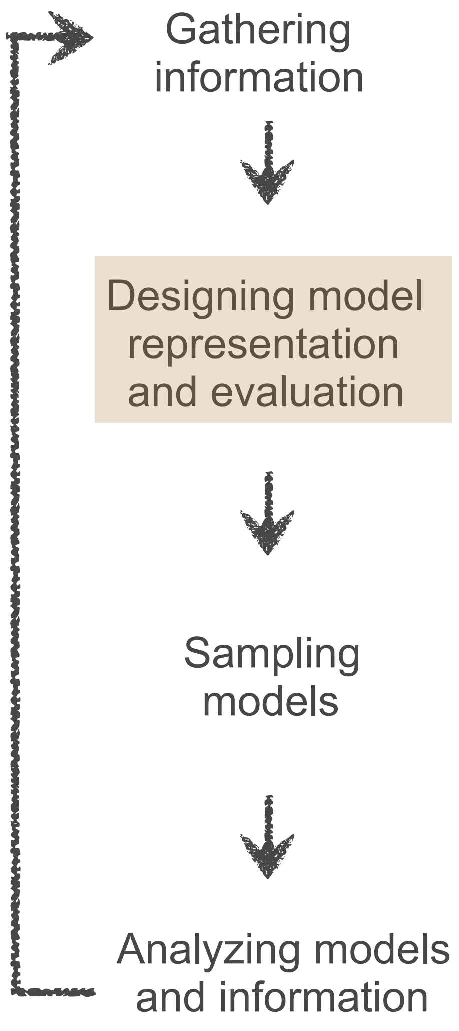
Designing model representation and evaluation

Sampling models

Analyzing models and information

PMI vs. core IMP restraints

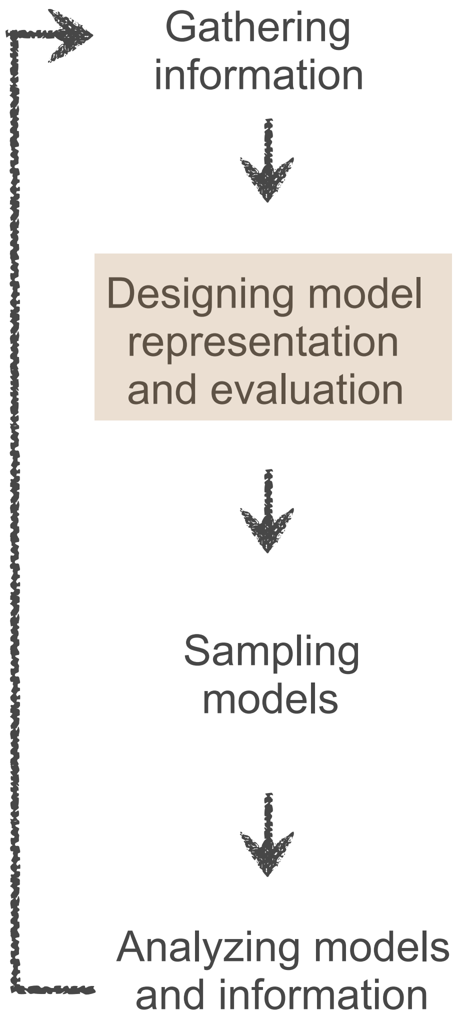
- Compare IMP.pmi's **`ExcludedVolumeSphere`** restraint with the IMP.core **`SingletonRestraint`** seen earlier in the example Python script



Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

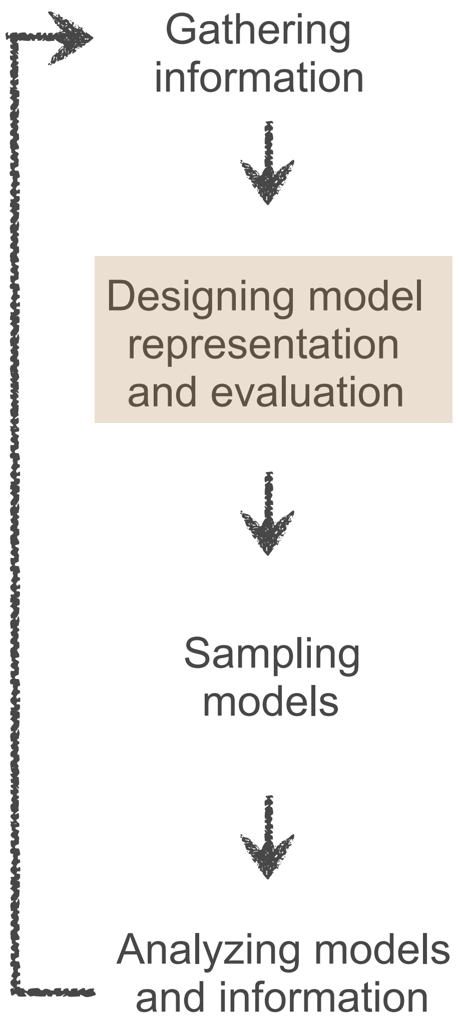
PMI vs. core IMP restraints

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


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

PMI vs. core IMP restraints

- Compare IMP.pmi's **`ExcludedVolumeSphere`** restraint with the IMP.core **`SingletonRestraint`** seen earlier in the 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)



Gathering information

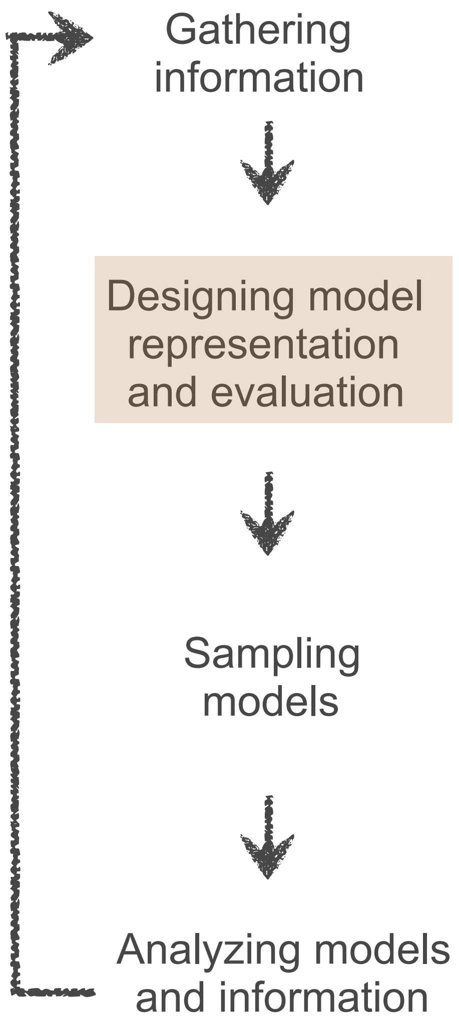
Designing model representation and evaluation

Sampling models

Analyzing models and information

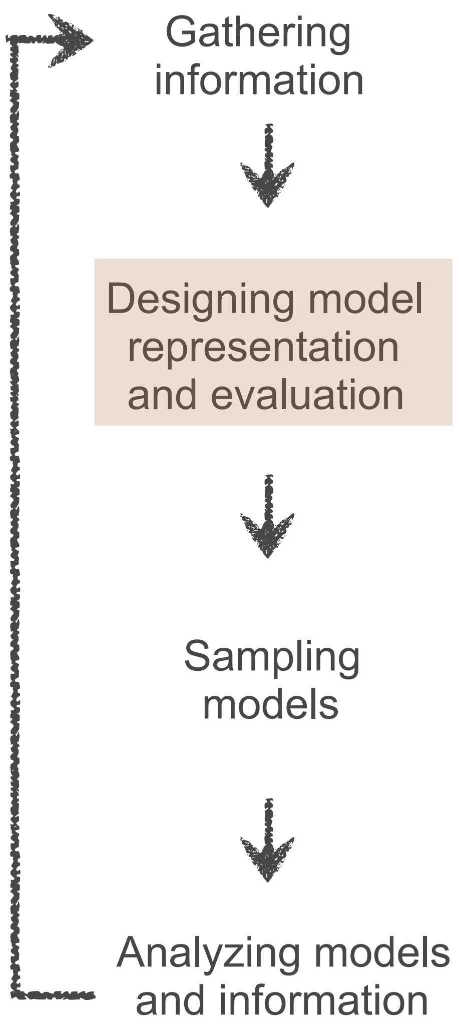
PMI vs. core IMP restraints

- Compare IMP.pmi's **`ExcludedVolumeSphere`** restraint with the IMP.core **`SingletonRestraint`** seen earlier in the 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)


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

PMI vs. core IMP restraints

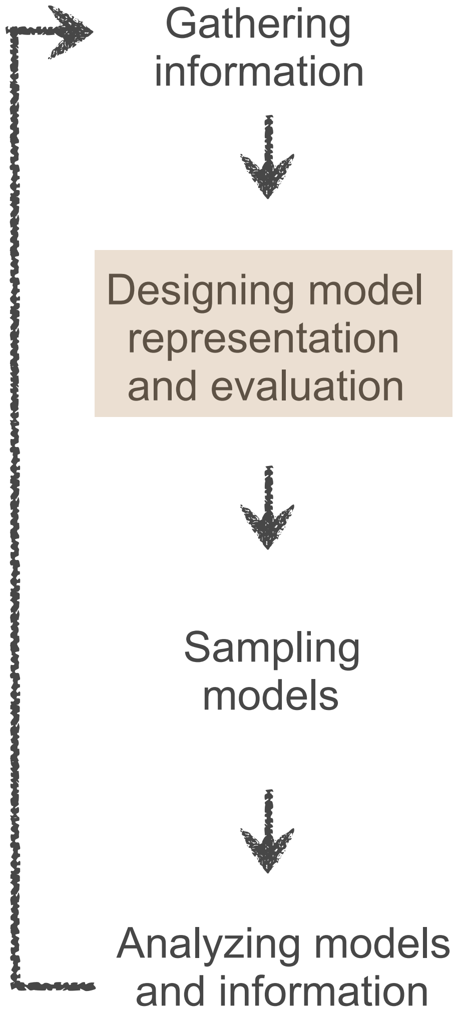
- Compare IMP.pmi's **`ExcludedVolumeSphere`** restraint with the IMP.core **`SingletonRestraint`** seen earlier in the example Python script
- Core IMP restraints act on explicitly defined particles (bottom up)
- PMI restraints act on named biological units (or the entire system, as in this case; top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


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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)
```



Gathering information

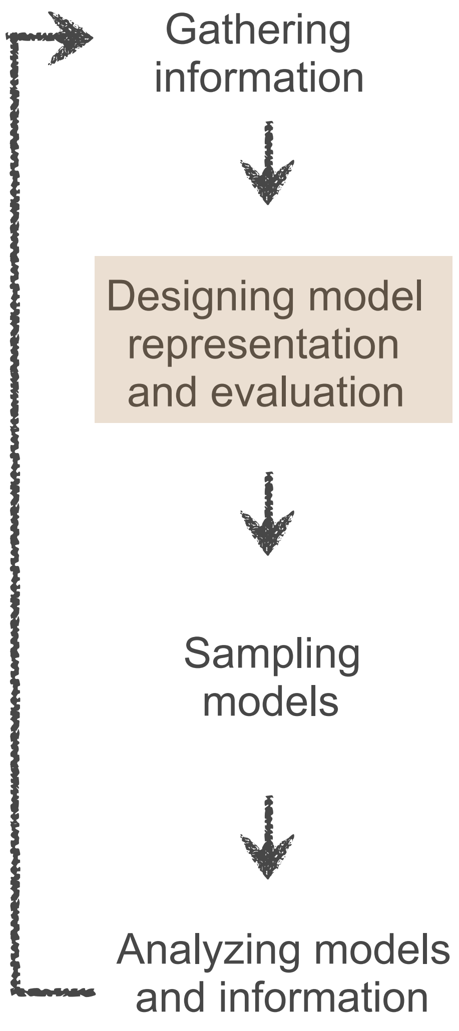
Designing model representation and evaluation

Sampling models

Analyzing models and information

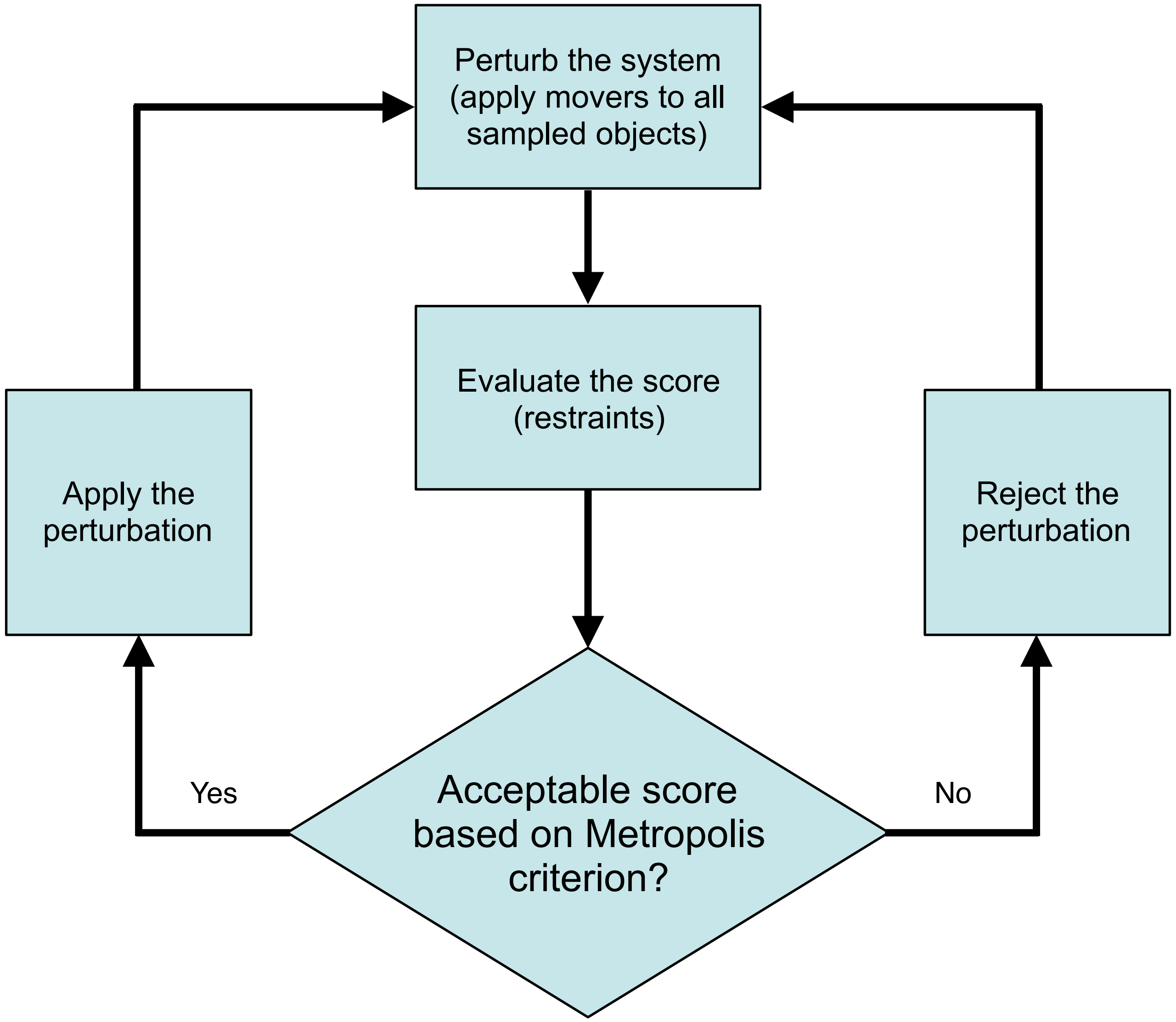
Other restraints

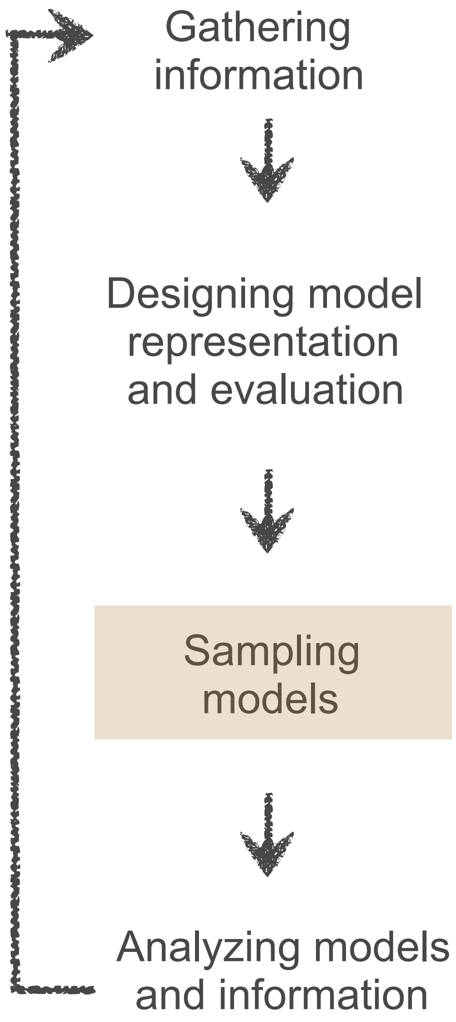
- Note that we're not using electrostatics or stereochemistry; very different to a typical molecular mechanics simulation
  - Electrostatics usually not relevant on this scale
  - Where it is, it is considered implicitly (from the input structures)
  - No atomic data in this case, so no stereochemistry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


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Sampling

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


Perturb the system (apply movers to all sampled objects)
Evaluate the score (restraints)
Apply the perturbation
Reject the perturbation
Acceptable score based on Metropolis criterion?
Yes
No

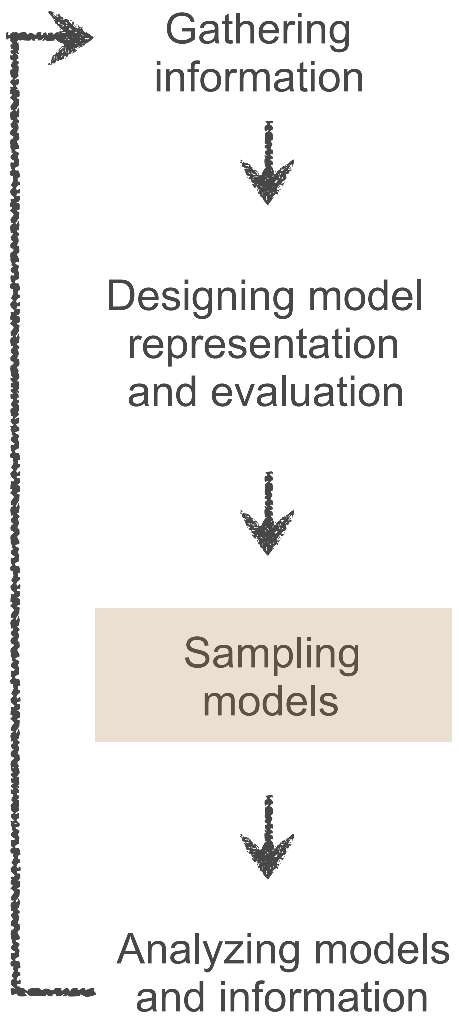

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

- Thus, need to define a set of movers

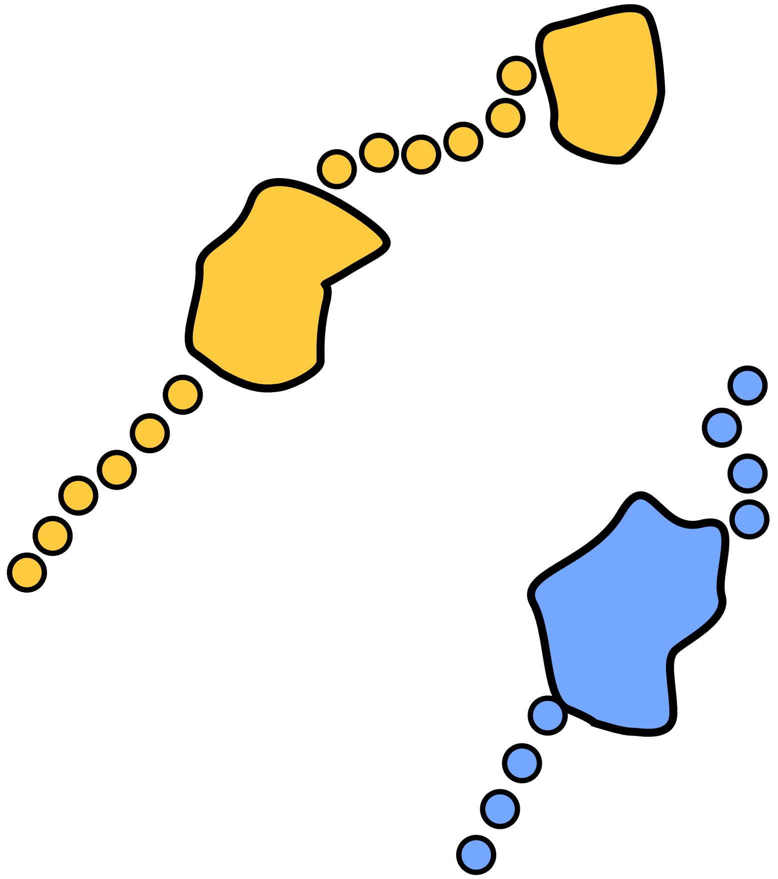
Monte Carlo setup

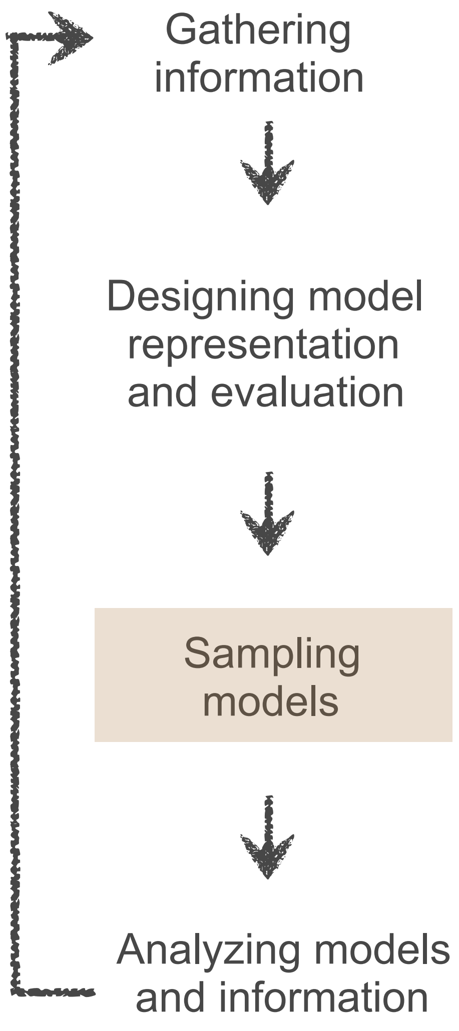
- Rigid body movers: simple 3D translation and rotation, sampled linearly up to given maximum values
- Bead movers: 3D translation
- Also we define here *how* to move our rigid bodies

```
#--------------------------
# 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





Gathering information

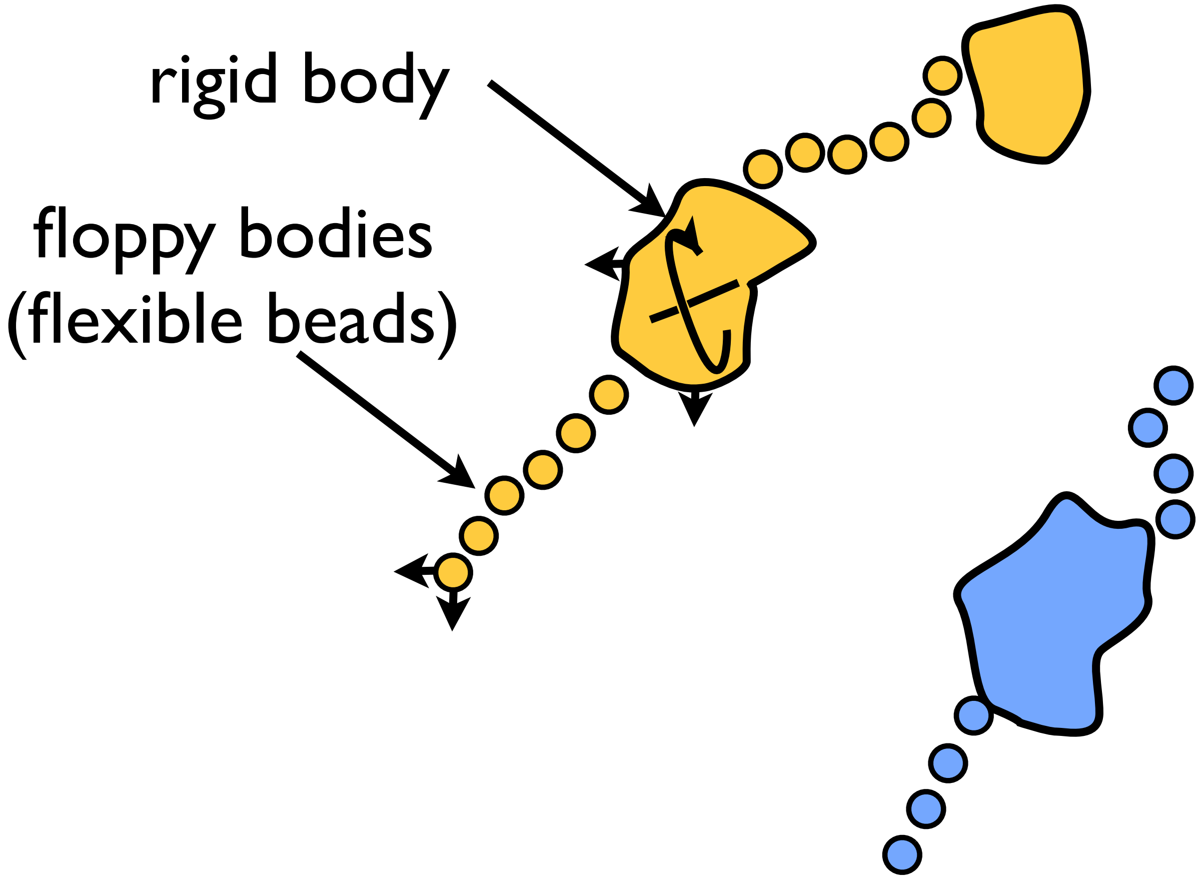
Designing model representation and evaluation

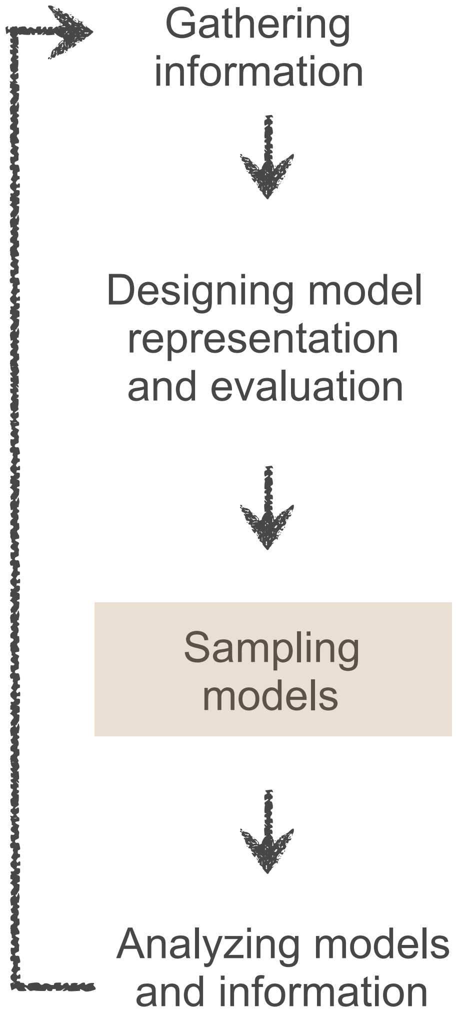
Sampling models

Analyzing models and information

Rigid body movers

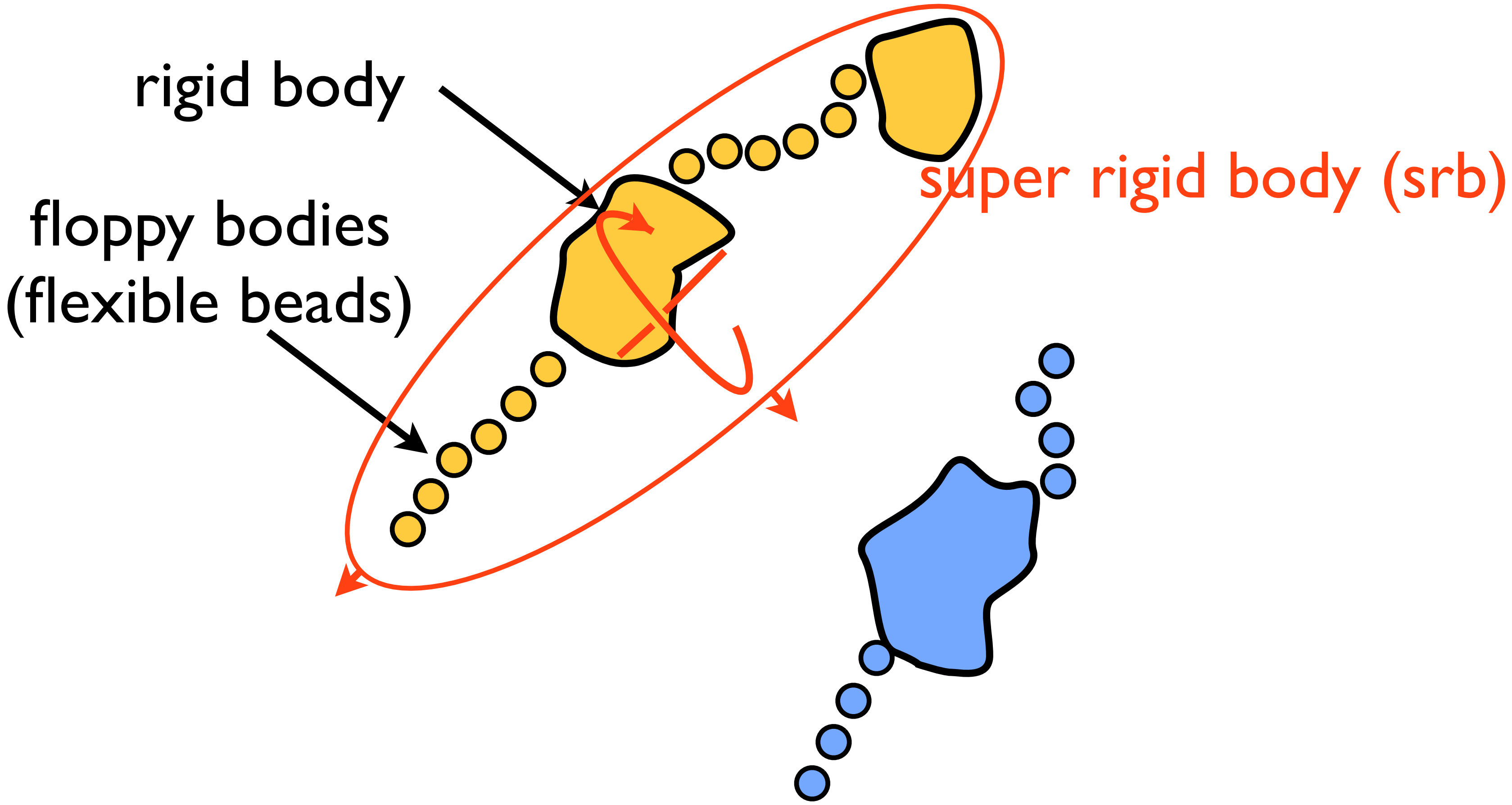
rigid_bodies defines the components that will be moved as rigid bodies (in this case, the parts of Rpb4 and Rpb7 for which we have X-ray structure). Unstructured regions will move as flexible beads.

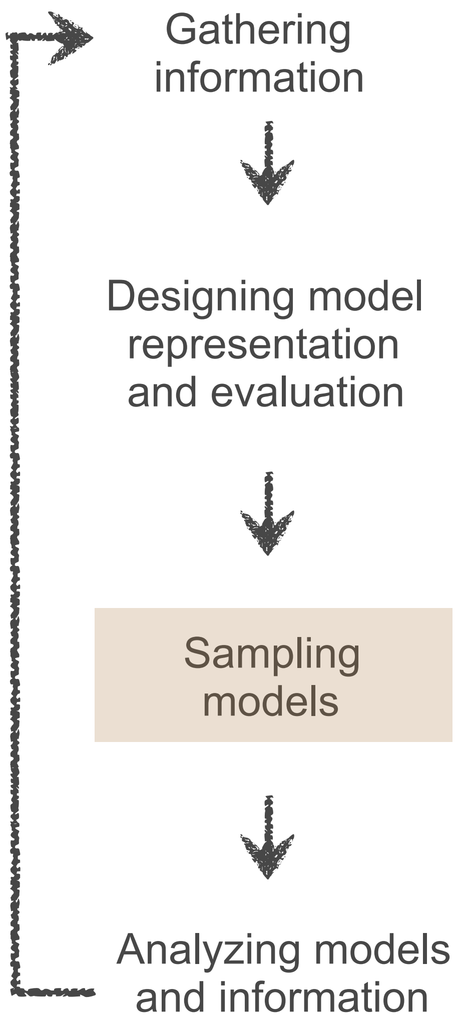



Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Rigid body movers

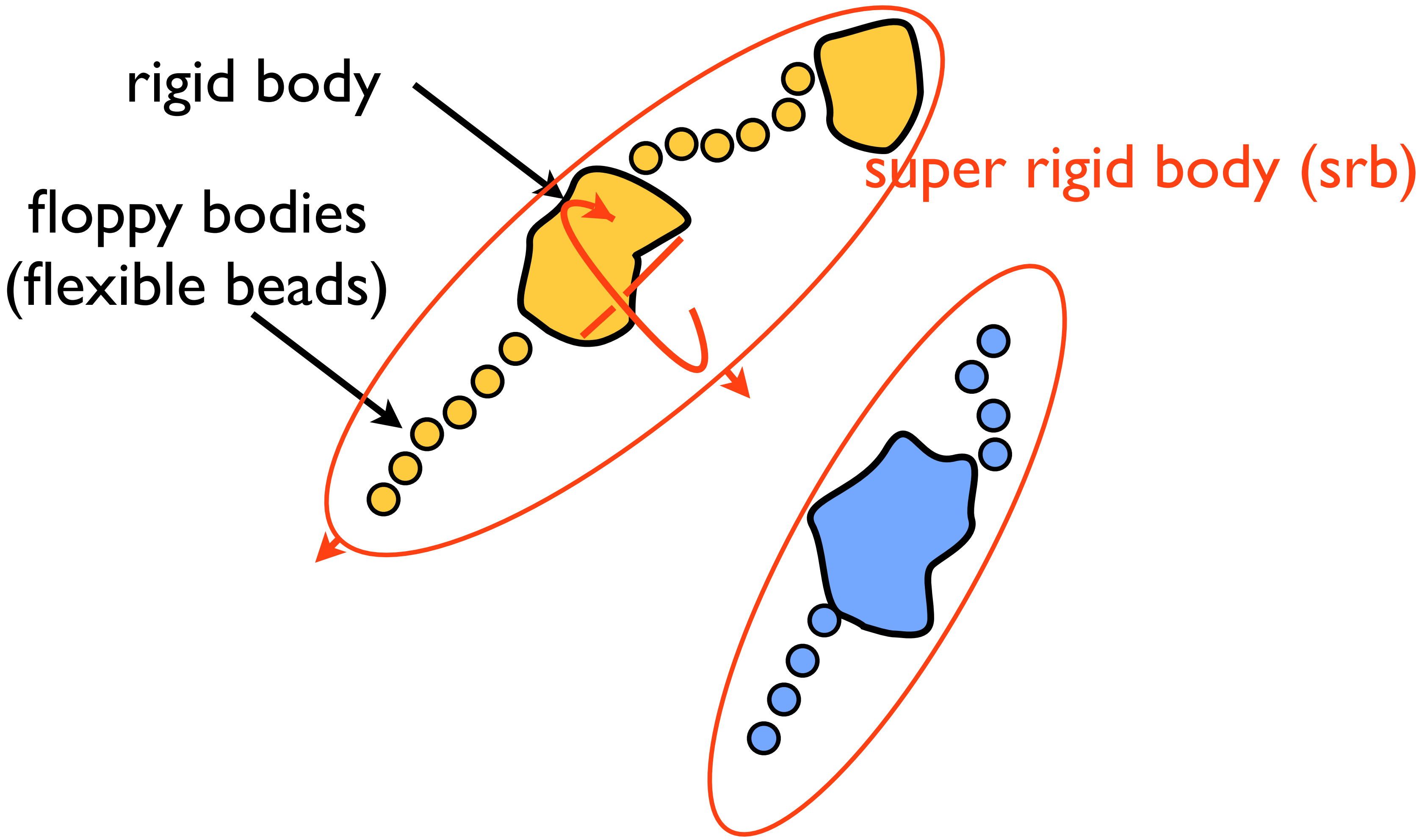
super_rigid_bodies defines sets of rigid bodies and beads that will move together in an additional Monte Carlo move.




Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Rigid body movers

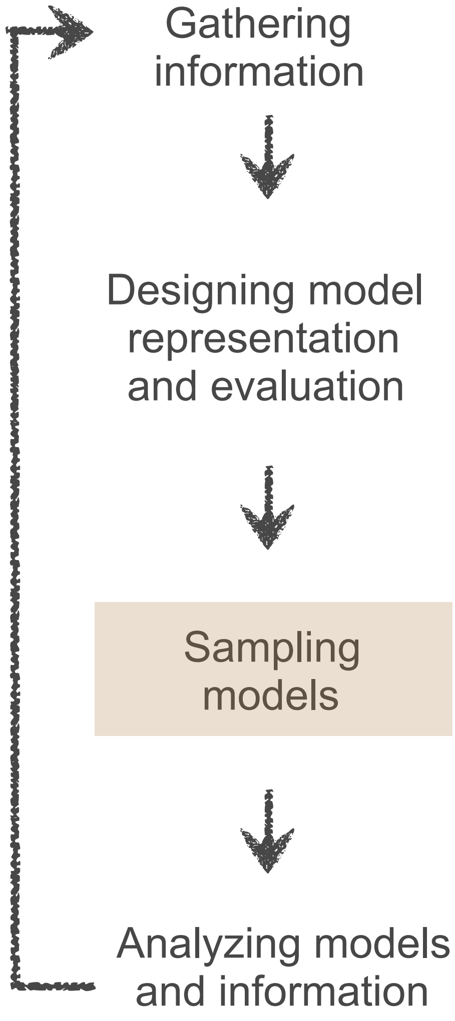
super_rigid_bodies defines sets of rigid bodies and beads that will move together in an additional Monte Carlo move.



rigid body

floppy bodies
(flexible beads)

super rigid body (srb)



Gathering information

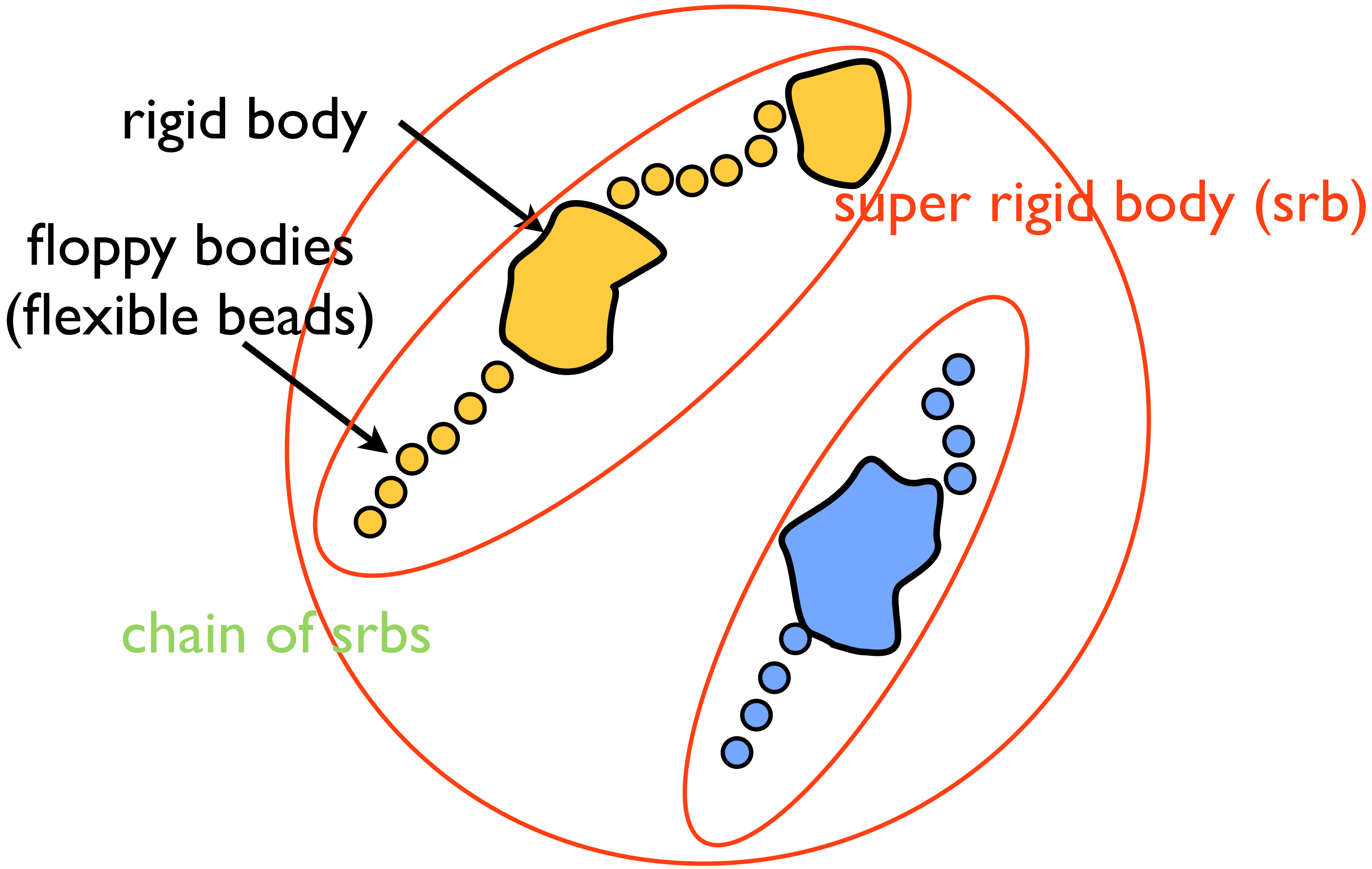
Designing model representation and evaluation

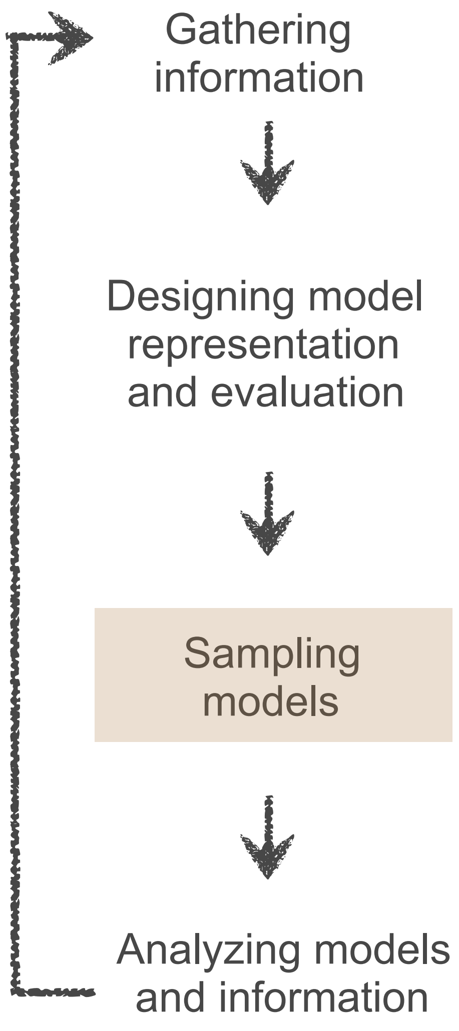
Sampling models

Analyzing models and information

Rigid body movers

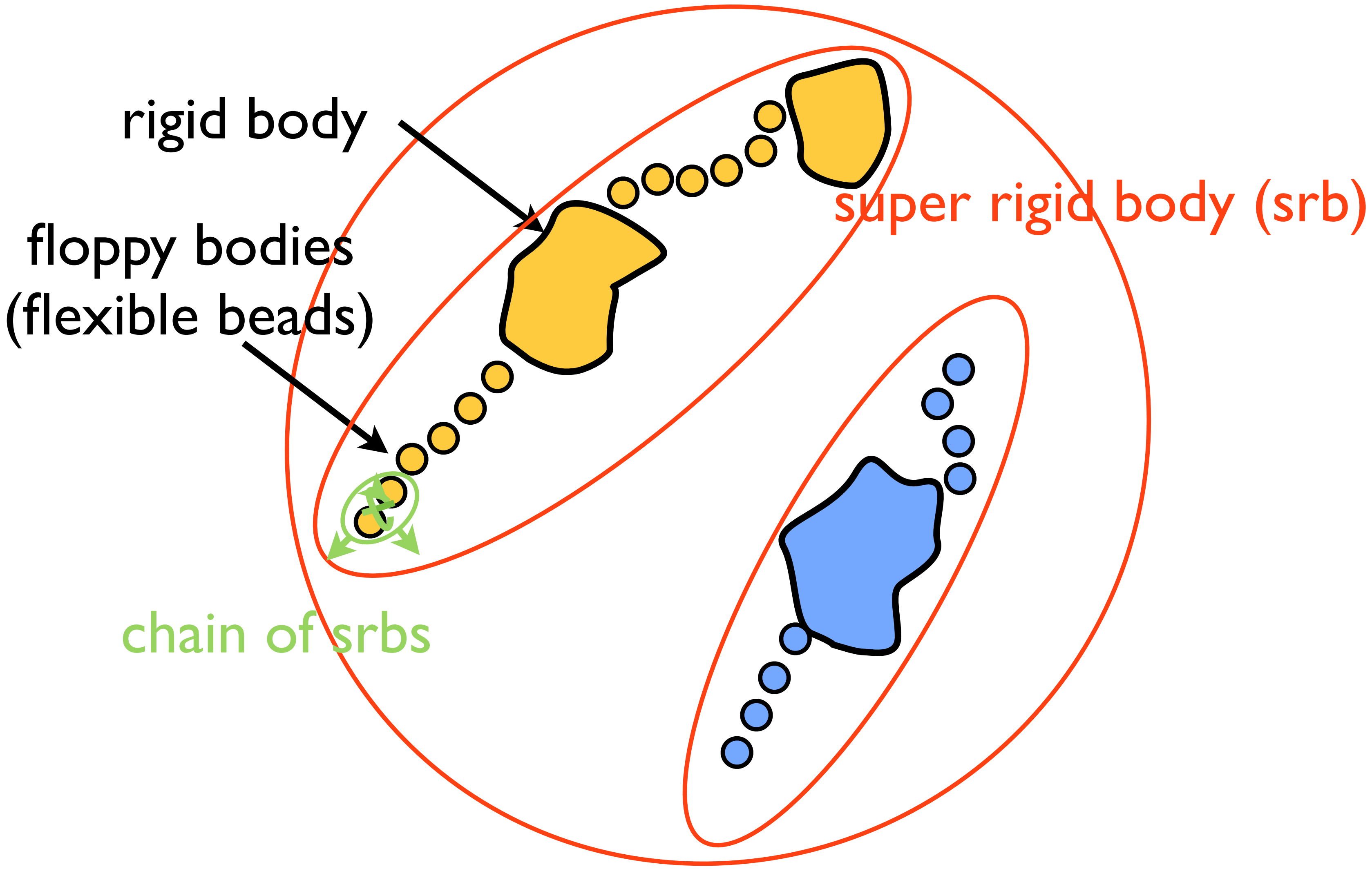
chain_of_super_rigid_bodies sets additional Monte Carlo movers along the connectivity chain of a subunit. It groups sequence-connected rigid domains and/or beads into overlapping pairs and triplets. Each of these groups will be moved rigidly. This mover helps to sample more efficiently complex topologies,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

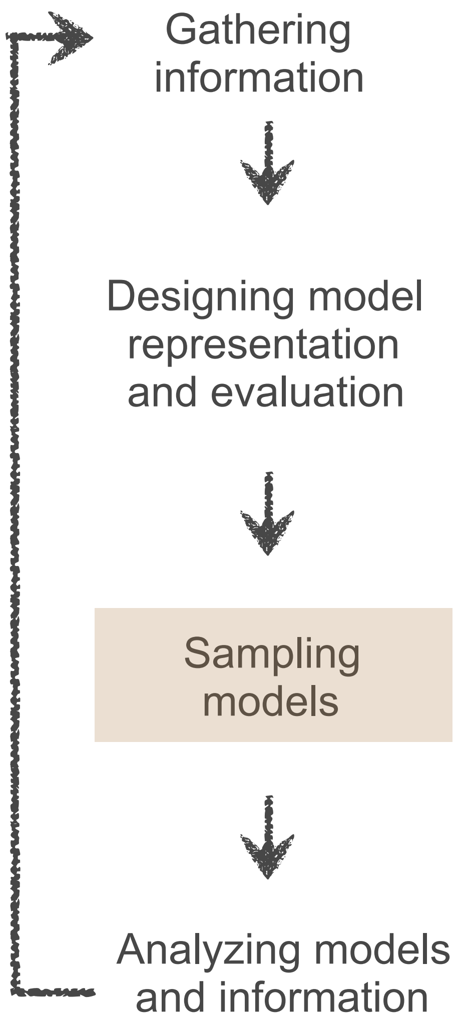



Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Rigid body movers

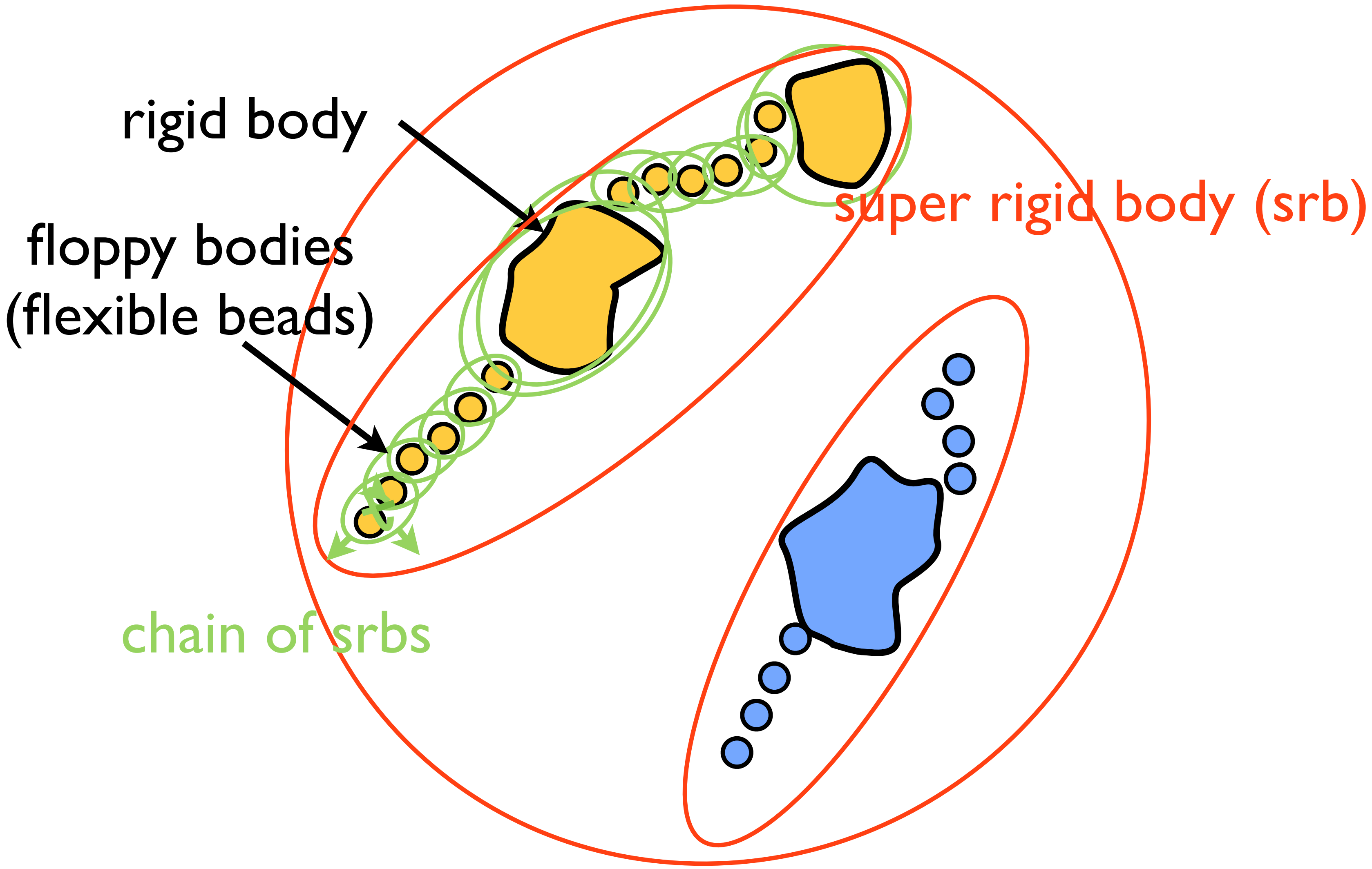
chain_of_super_rigid_bodies sets additional Monte Carlo movers along the connectivity chain of a subunit. It groups sequence-connected rigid domains and/or beads into overlapping pairs and triplets. Each of these groups will be moved rigidly. This mover helps to sample more efficiently complex topologies,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

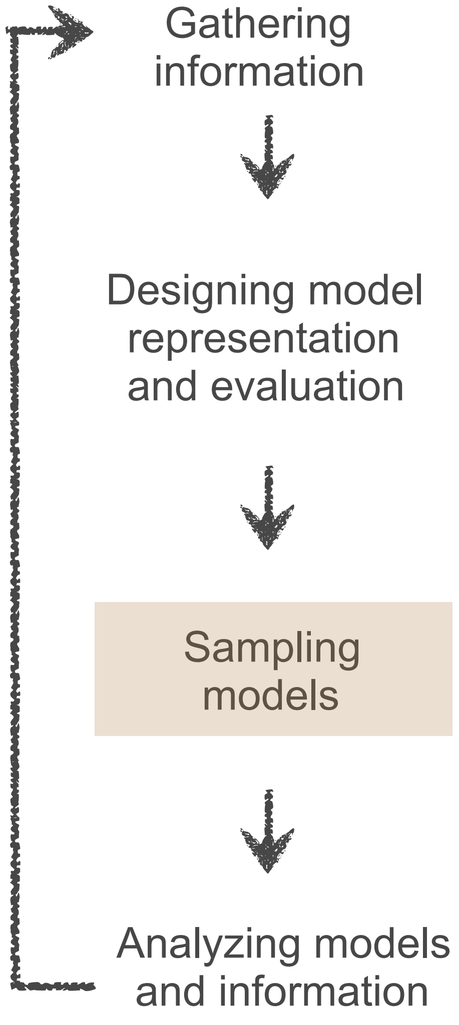



Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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.

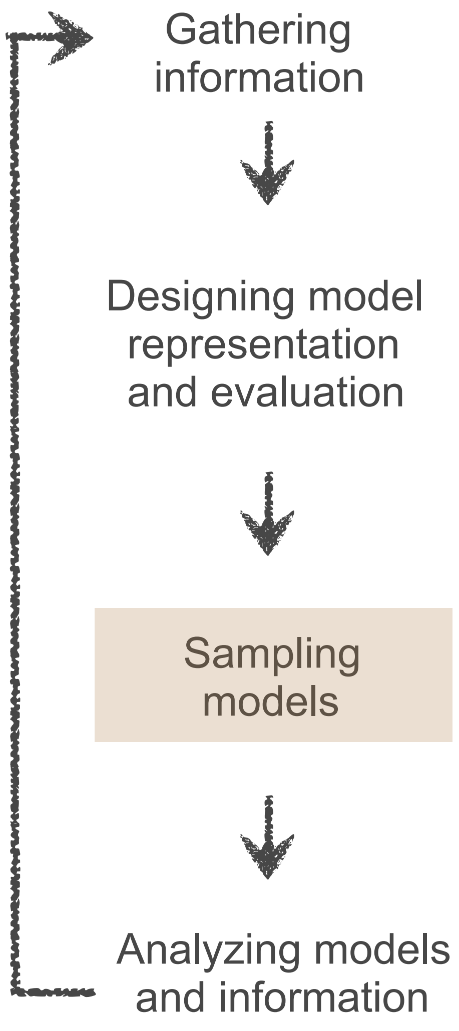



Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Sampling

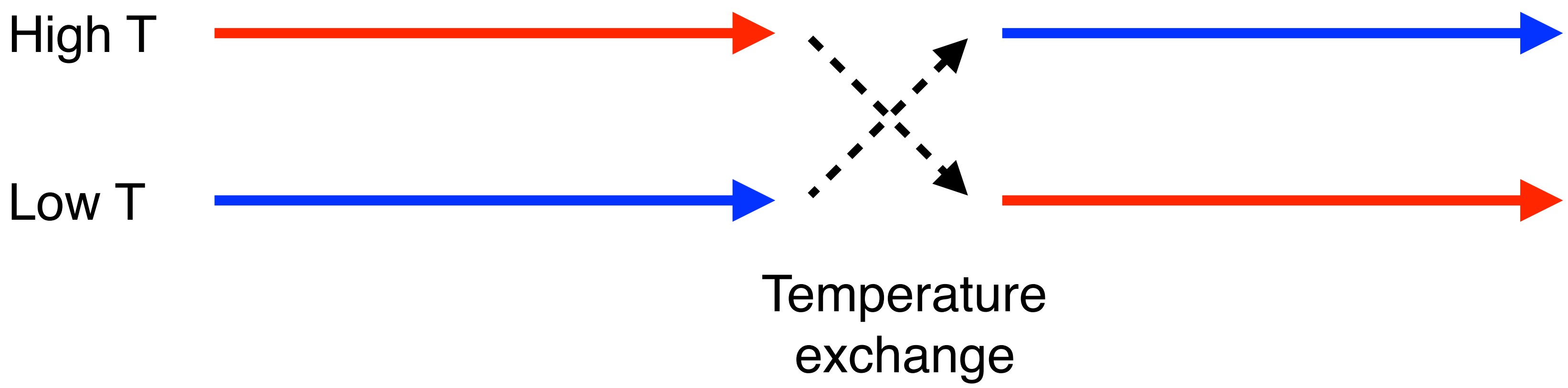
- Finally, we run the Monte Carlo sampling itself
- Technically this is replica exchange (as the class name suggests)

```
mc1=IMP.pmi.macros.ReplicaExchange0(m,
                               representation,
                               monte_carlo_sample_objects=sampleobjects,
                               output_objects=outputobjects,
                               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")
```


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Replica exchange

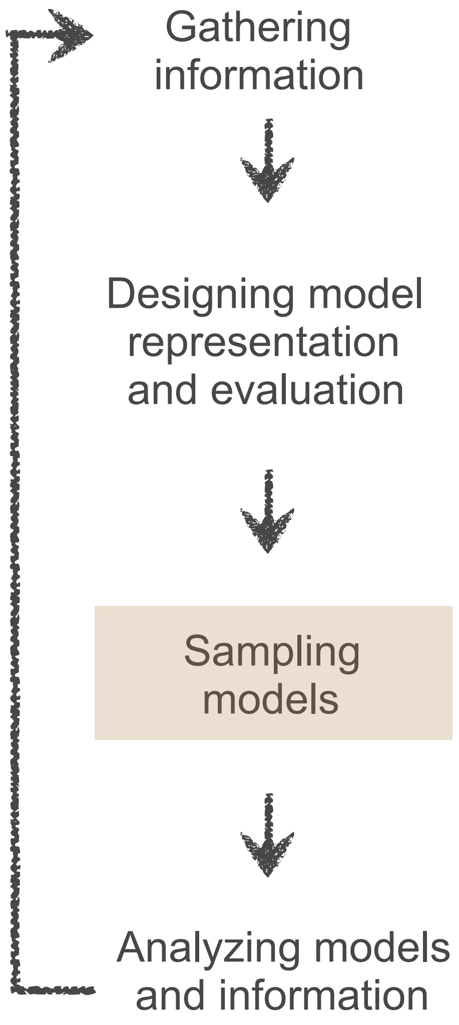
- Multiple simulations run in parallel, at different temperatures
- Periodically, coordinates/temperatures may be swapped
- Helps to overcome local minima with little communication overhead



- 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)

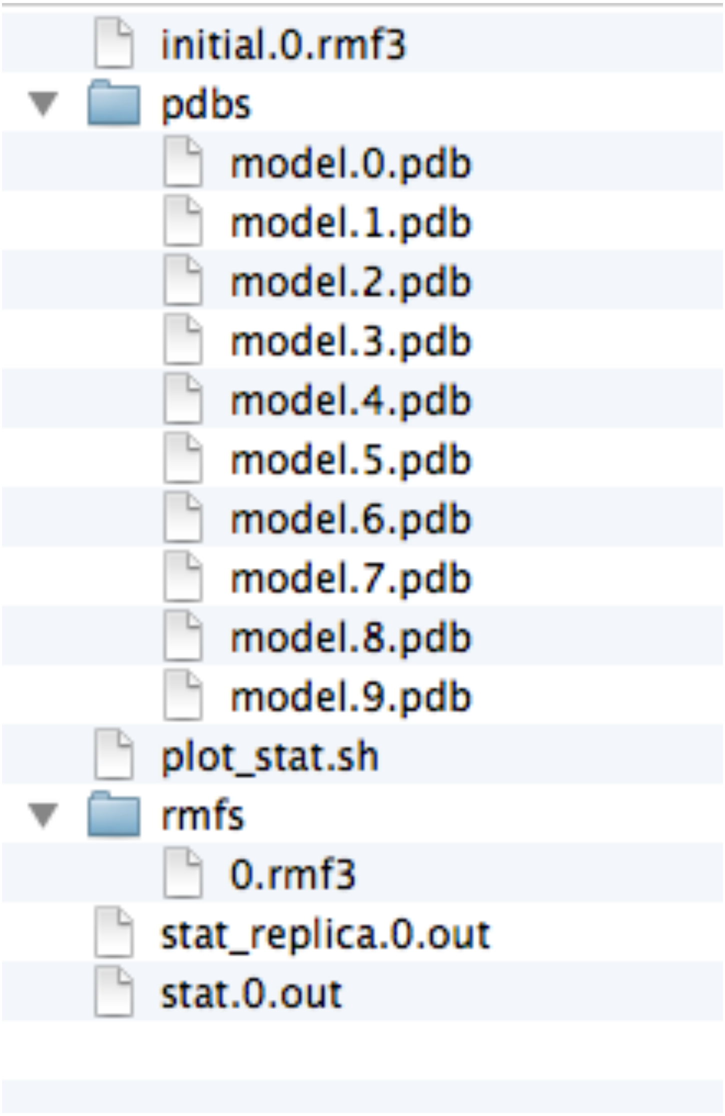
Script output

```
$ 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
```


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Output data

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



```
initial.0.rmf3
▼ pdbs
    model.0.pdb
    model.1.pdb
    model.2.pdb
    model.3.pdb
    model.4.pdb
    model.5.pdb
    model.6.pdb
    model.7.pdb
    model.8.pdb
    model.9.pdb
plot_stat.sh
▼ rmfs
    0.rmf3
stat_replica.0.out
stat.0.out
```

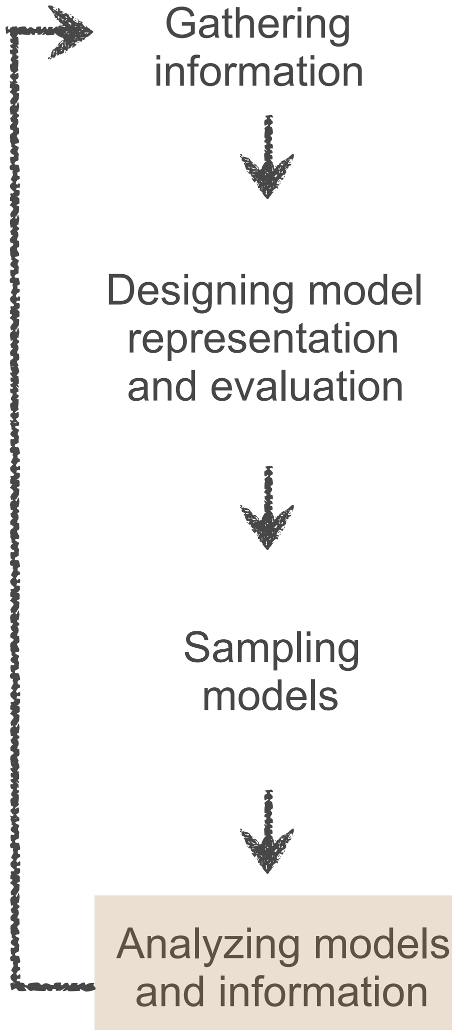
Initial structure (RMF format)

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

the trajectory (RMF format)

replica exchange statistics

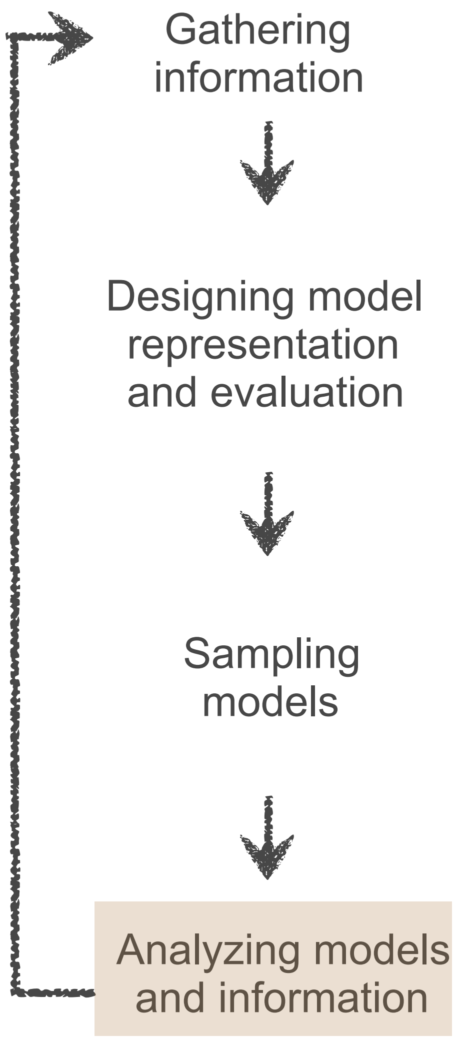
a stat file containing all useful information on **`outputobjects`**


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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
- 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)



Gathering information

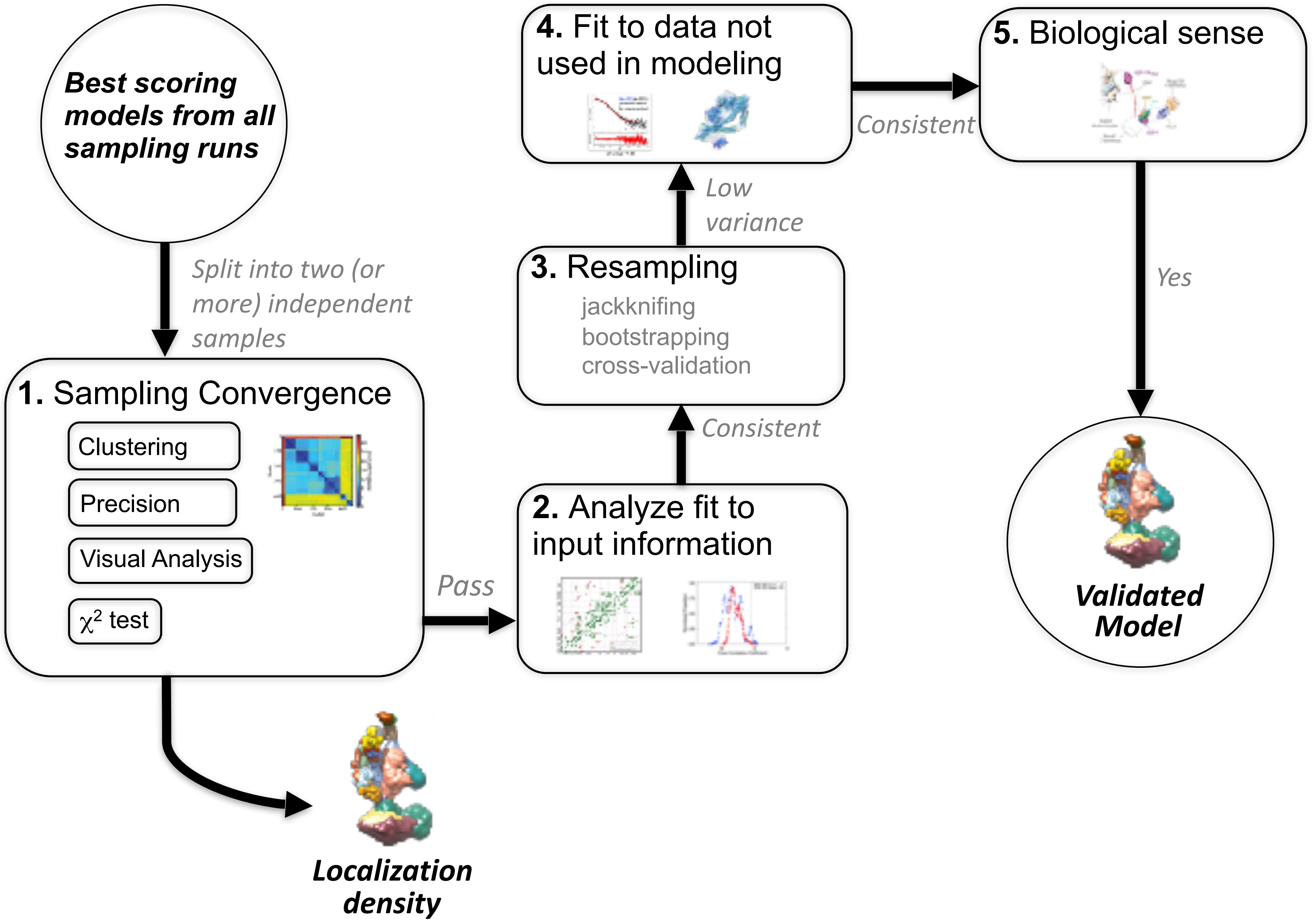
Designing model representation and evaluation

Sampling models

Analyzing models and information

Analysis

- Many steps involved in analysis; only a subset covered here and in the earlier Nup84 talk
- Daniel will talk in more detail about this tomorrow



Best scoring models from all sampling runs

Split into two (or more) independent samples

1. Sampling Convergence

Clustering

Precision

Visual Analysis

χ^2 test

Pass

2. Analyze fit to input information

Consistent

3. Resampling

jackknifing

bootstrapping

cross-validation

Low variance

4. Fit to data not used in modeling

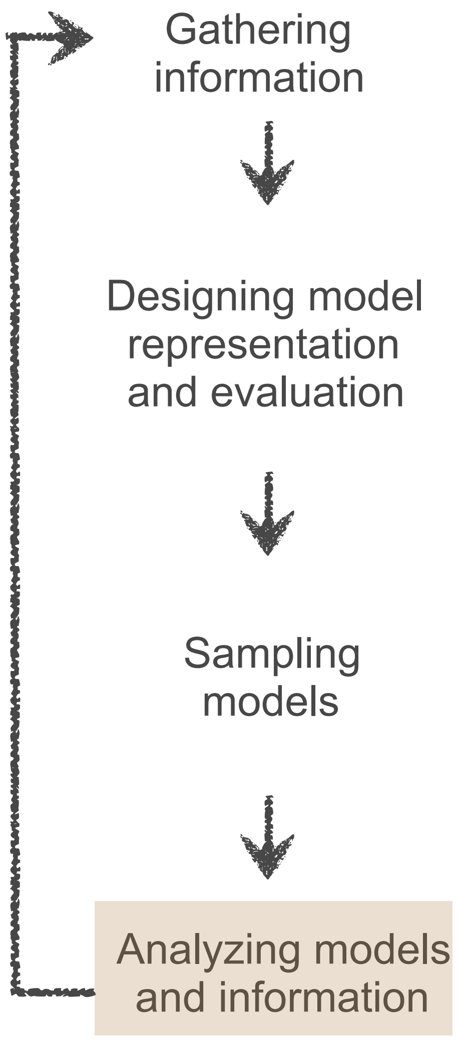
Consistent

5. Biological sense

Yes

Validated Model

Localization density



Gathering information

Designing model representation and evaluation

Sampling models

Analyzing models and information

Clustering

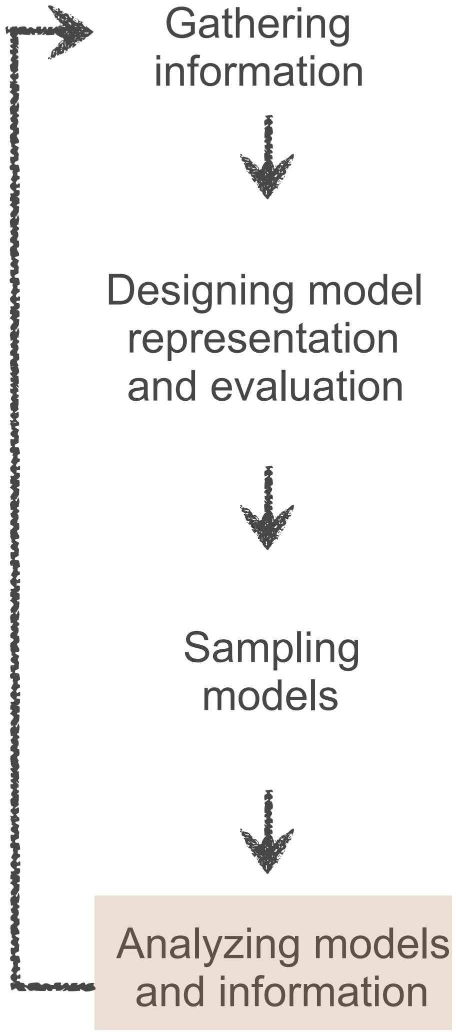
- A simple clustering protocol is shown in `rnapolii/analysis/clustering_em.py`
- Simply run with

```
$ cd ../analysis
$ python clustering_em.py
```

- k-means clustering after discarding bad-scoring models, using comparisons of Rpb4 and Rpb7 positions (RMSD)
- Can be used to merge multiple independent runs

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

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



Gathering information

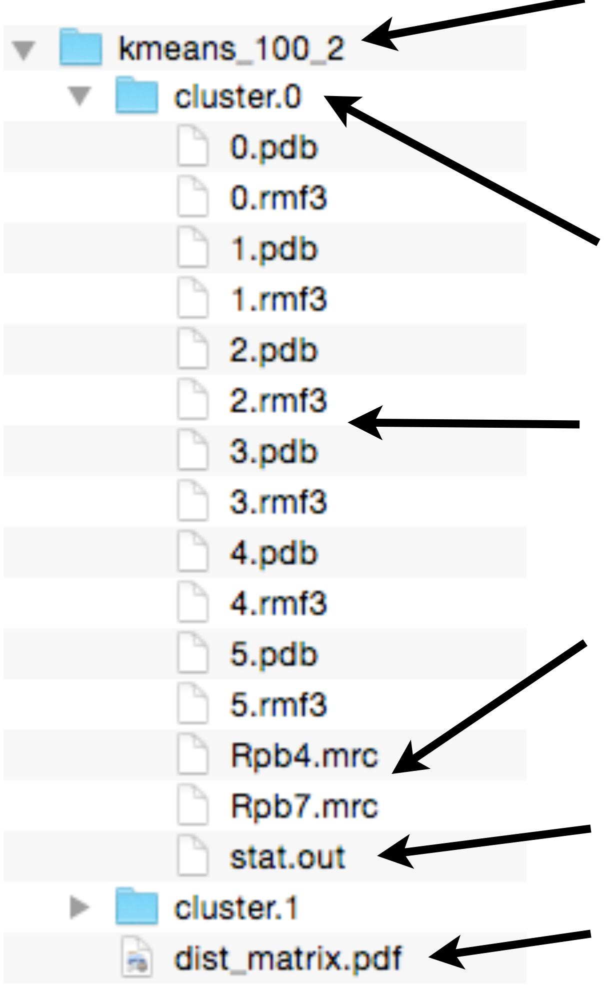
Designing model representation and evaluation

Sampling models

Analyzing models and information

Clustering output

- For demonstration a very small single cluster (5 top models) is generated, from a very short sampling run
- Outputs shown here are from the complete run (without `--test`), 100 top models put into 2 clusters
  - Provided in em_full_kmeans_100_2.zip



```
▼ kmeans_100_2
    ▼ cluster.0
        0.pdb
        0.rmf3
        1.pdb
        1.rmf3
        2.pdb
        2.rmf3
        3.pdb
        3.rmf3
        4.pdb
        4.rmf3
        5.pdb
        5.rmf3
        Rpb4.mrc
        Rpb7.mrc
        stat.out
    ▶ cluster.1
      dist_matrix.pdf
```

Directory name includes number of top models plus number of clusters

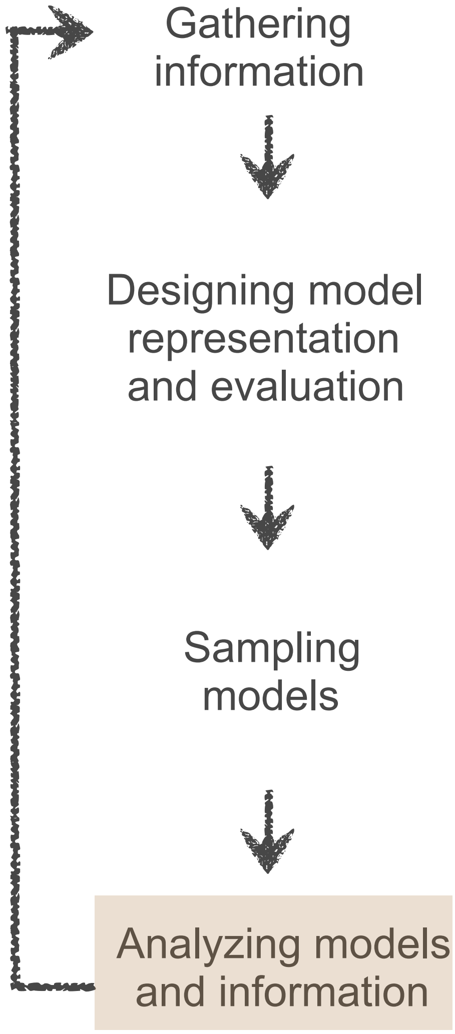
Subdirectory for each cluster, largest first

Cluster models in PDB and RMF format

Localization densities in MRC format

Statistics on the cluster

Distance matrix and dendrogram



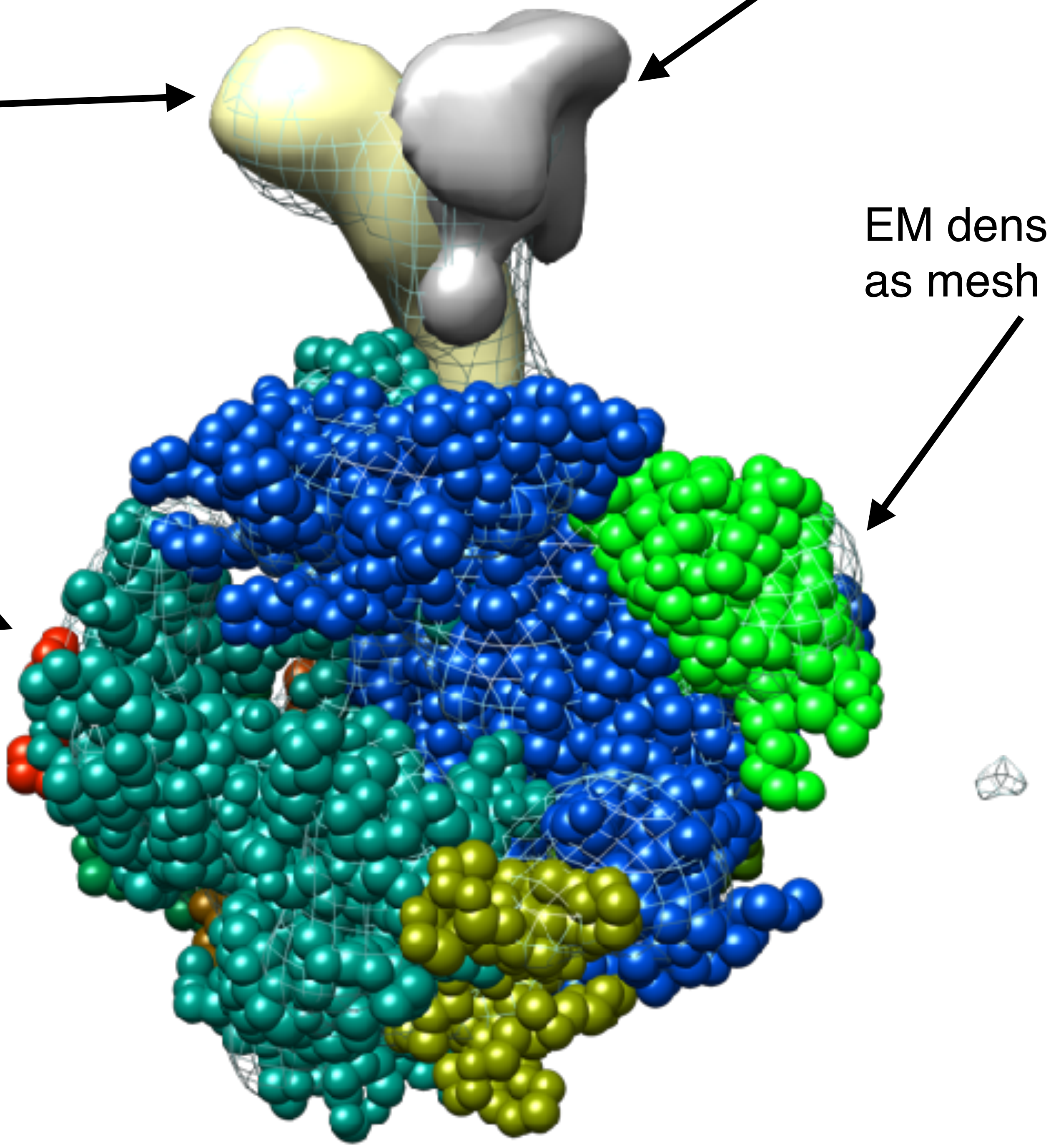
Gathering information

Designing model representation and evaluation

Sampling models

Analyzing models and information

View in Chimera

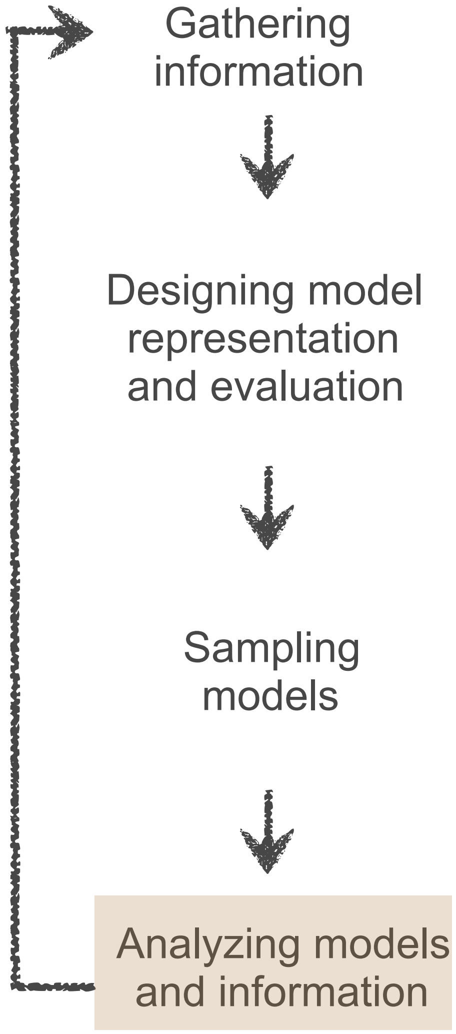


Rpb7 density

Rpb4 density

EM density map, as mesh

Native structure of other subunits (1-residue beads)



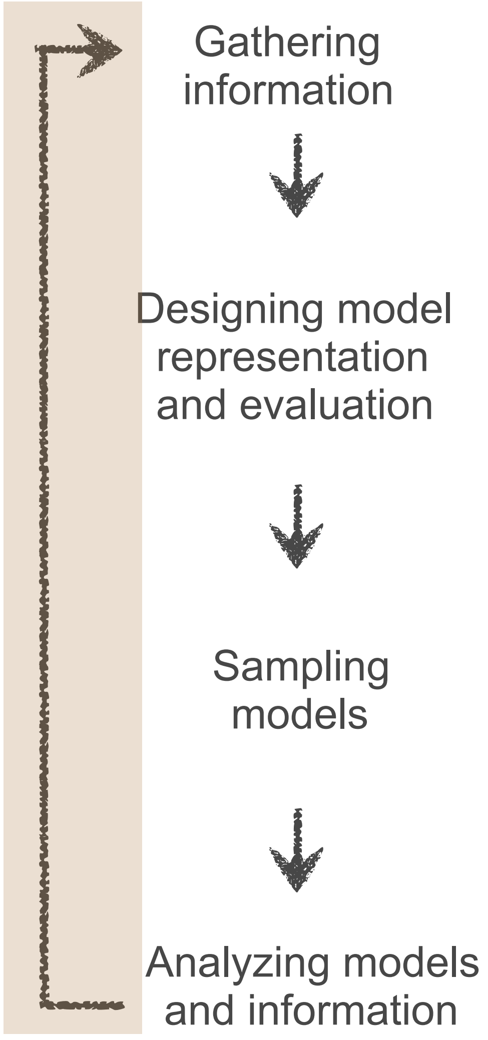
Gathering information

Designing model representation and evaluation

Sampling models

Analyzing models and information

Iteration



Gathering information

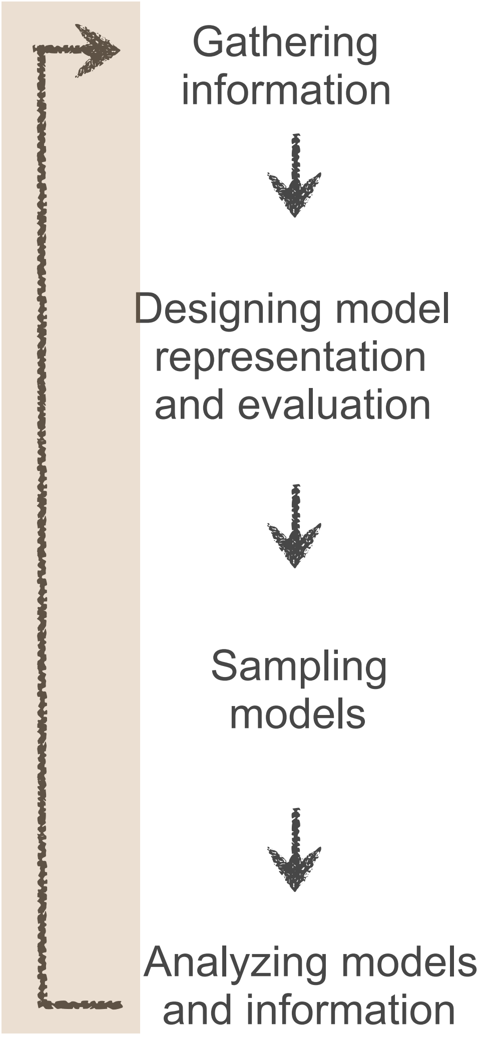
Designing model representation and evaluation

Sampling models

Analyzing models and information

Iteration

- Modeling output can suggest new experiments



Gathering information

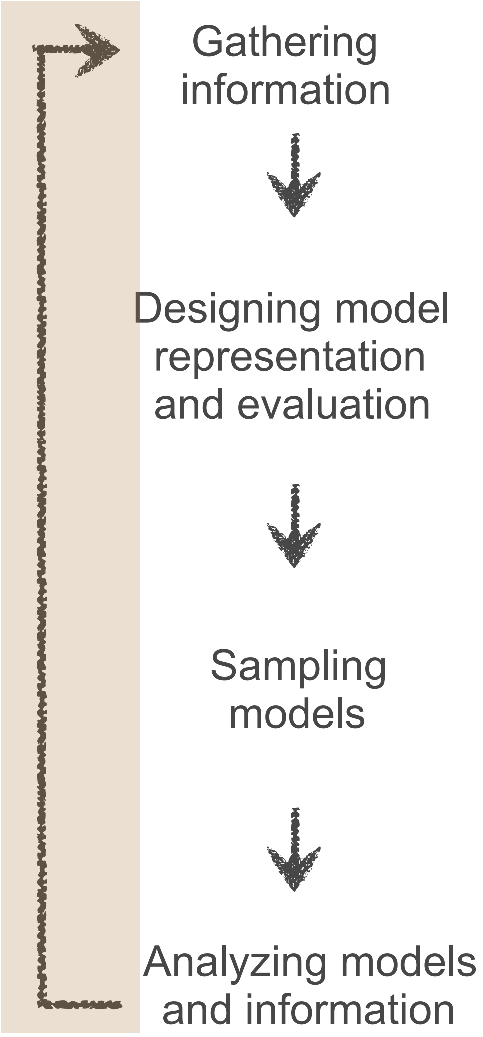
Designing model representation and evaluation

Sampling models

Analyzing models and information

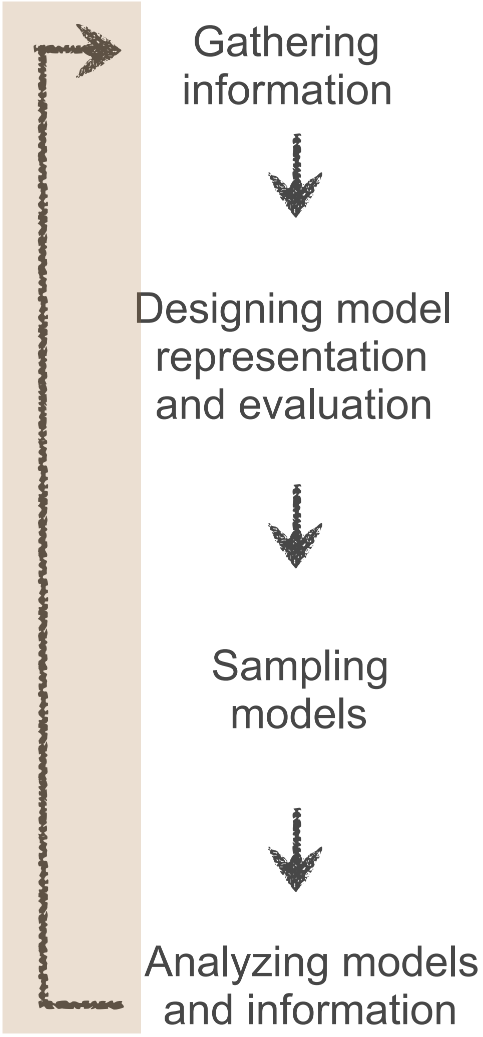
Iteration

- Modeling output can suggest new experiments
- In this case it's clear from looking at the localization densities that while Rpb4 and Rpb7 are placed in the EM map, most likely the protein-protein interfaces are not correct (no orientation dependence)


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Iteration

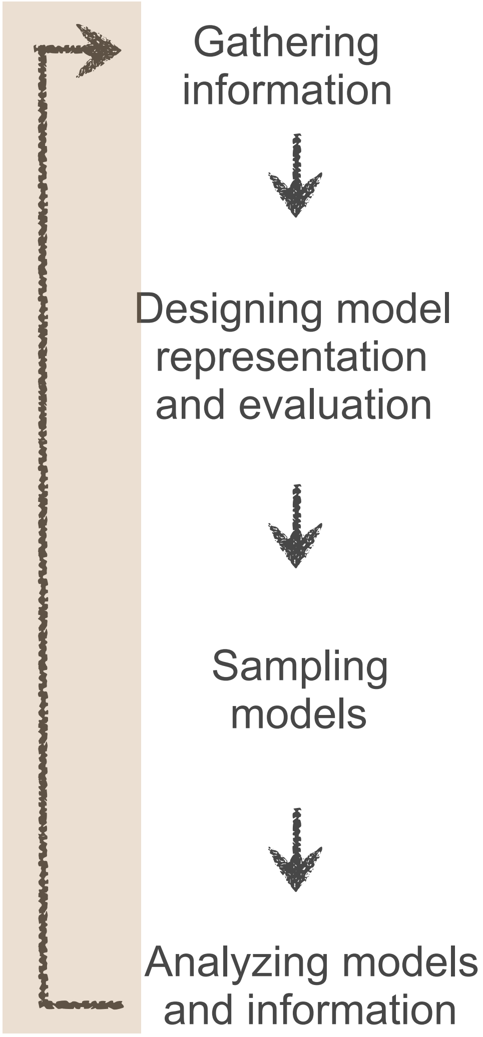
- Modeling output can suggest new experiments
- In this case it's clear from looking at the localization densities that while Rpb4 and Rpb7 are placed in the EM map, most likely the protein-protein interfaces are not correct (no orientation dependence)
- In more complex modeling with more subunits, their arrangement may not be pinned down



Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

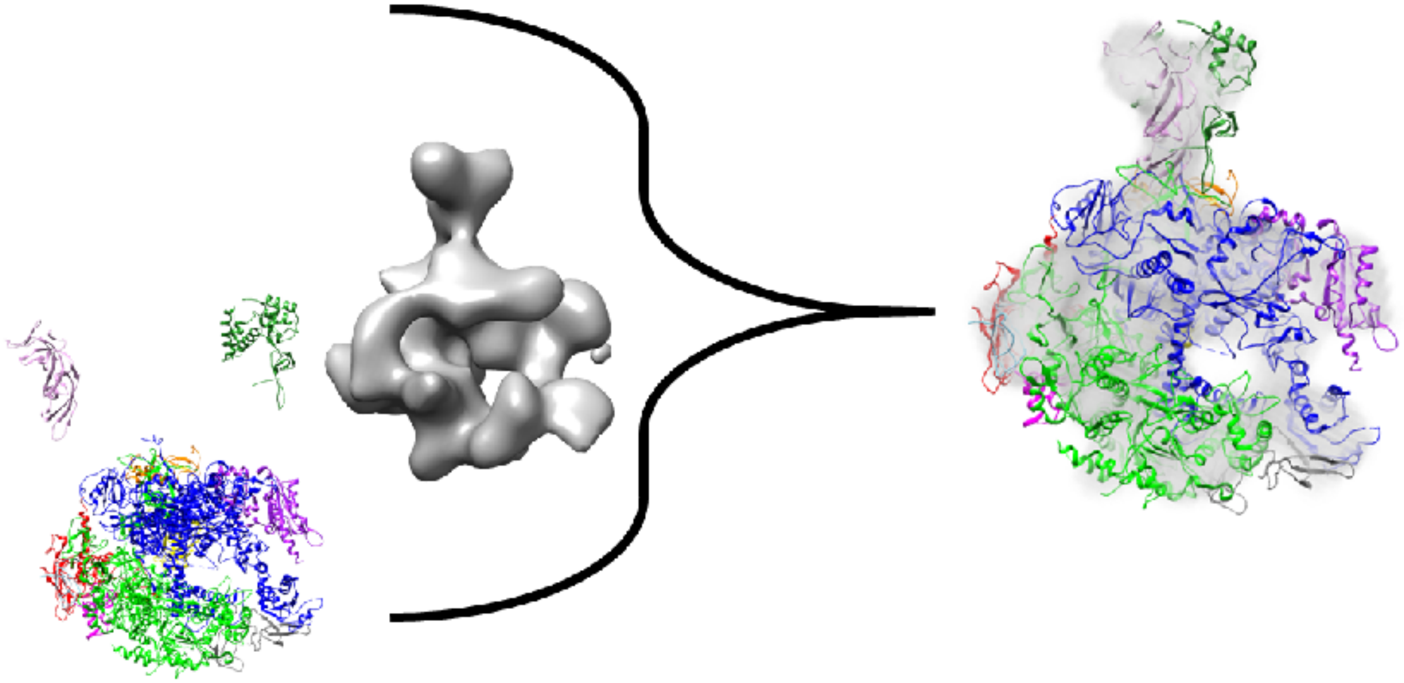
Iteration

- Modeling output can suggest new experiments
- In this case it's clear from looking at the localization densities that while Rpb4 and Rpb7 are placed in the EM map, most likely the protein-protein interfaces are not correct (no orientation dependence)
- In more complex modeling with more subunits, their arrangement may not be pinned down
- Pairwise residue interaction data should improve these interfaces


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

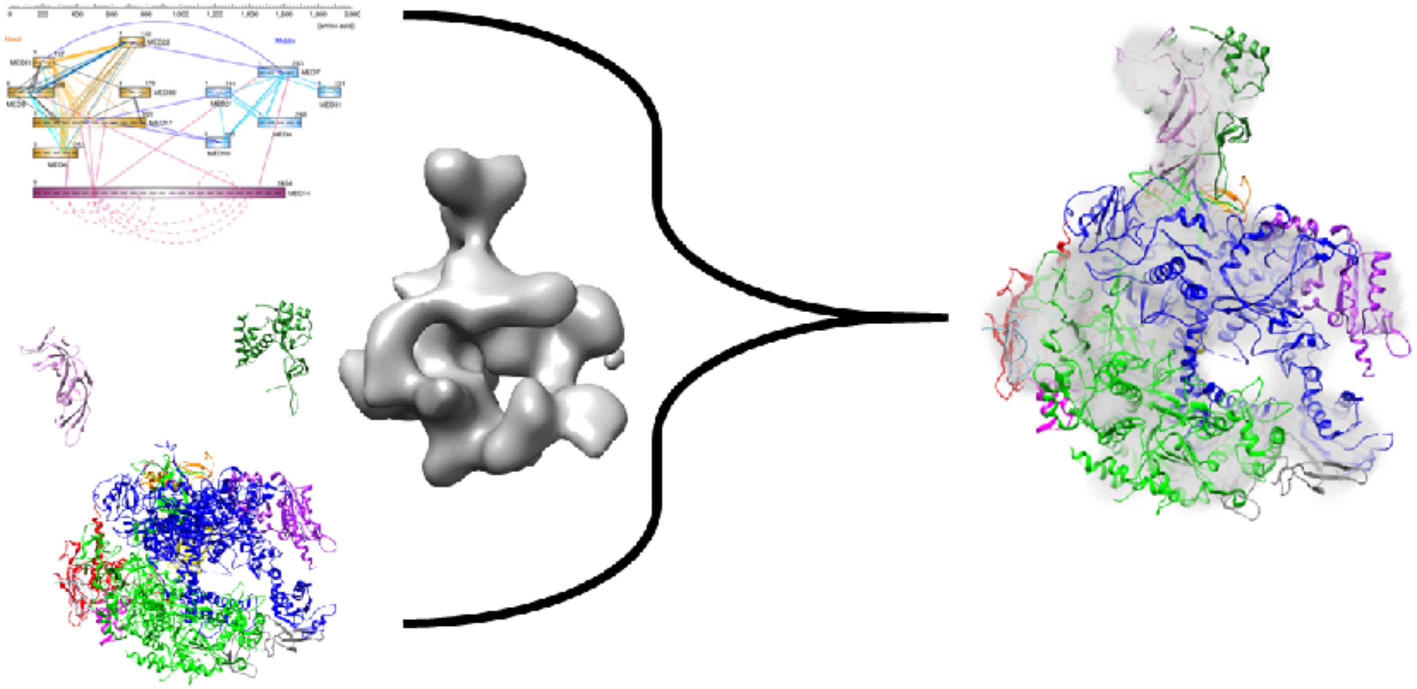
Addition of CX-MS data

- We'll add data from chemical cross-linking coupled with mass spectrometry (CX-MS) to the existing EM and X-ray



Addition of CX-MS data

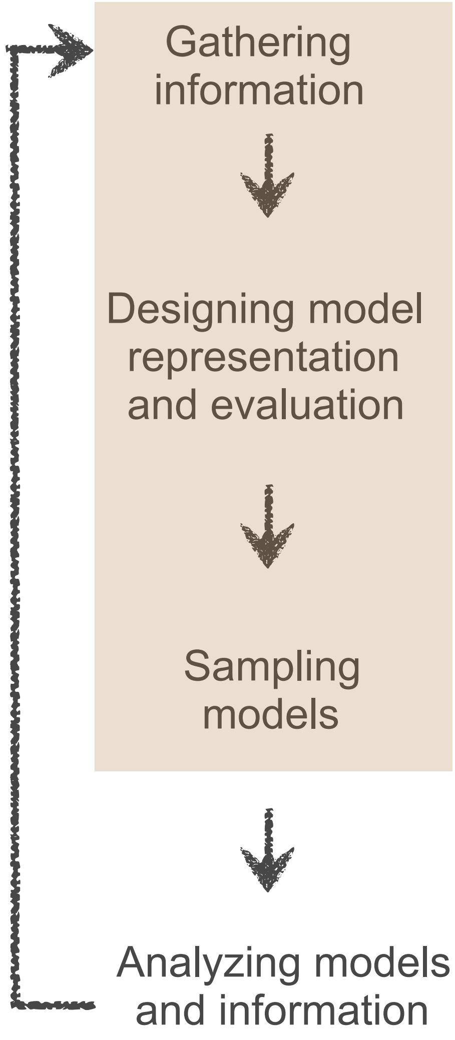
- We’ll add data from chemical cross-linking coupled with mass spectrometry (CX-MS) to the existing EM and X-ray



Running the new script

- As before, let’s get the new main modeling script running while we look at what it’s doing:

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

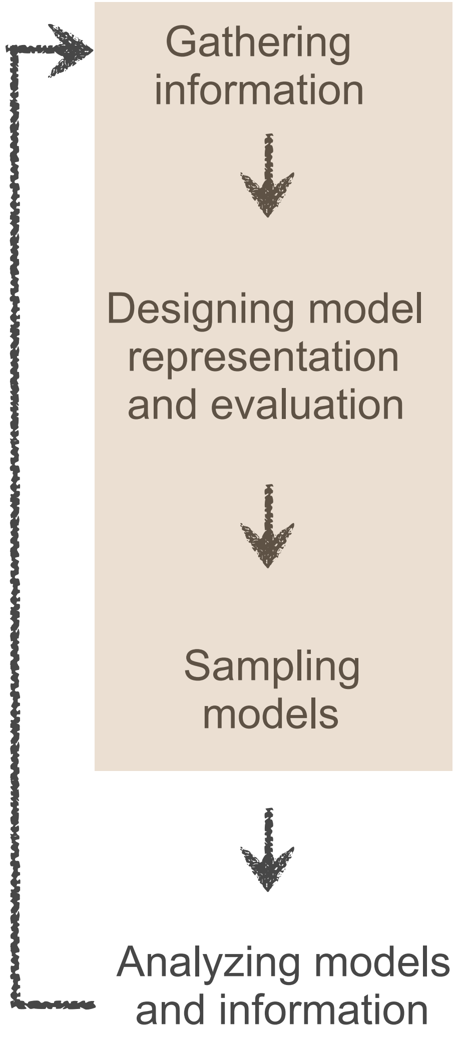

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Running the new script

- As before, let's get the new main modeling script running while we look at what it's doing:

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

Note that we're running a script in the **`modeling`** directory (not **`modeling_em`**) this time



Gathering information

Designing model representation and evaluation

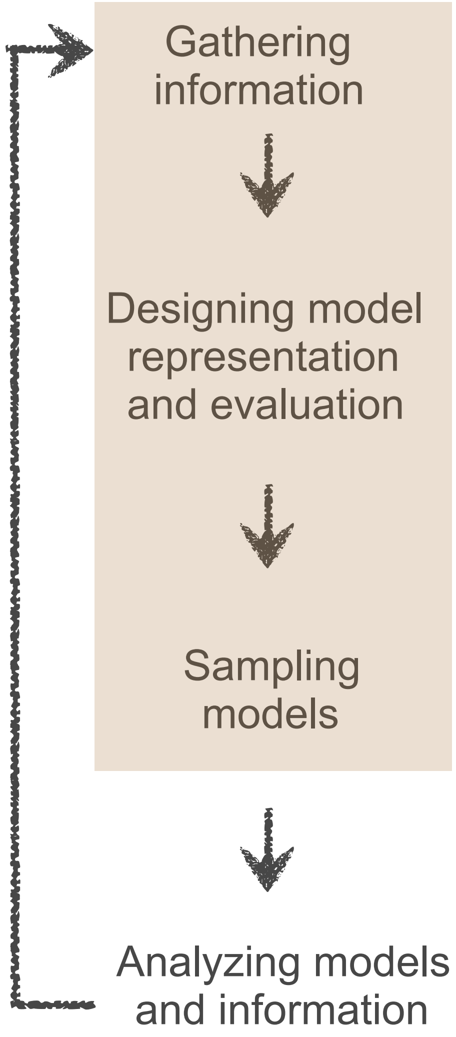
Sampling models

Analyzing models and information

Running the new script

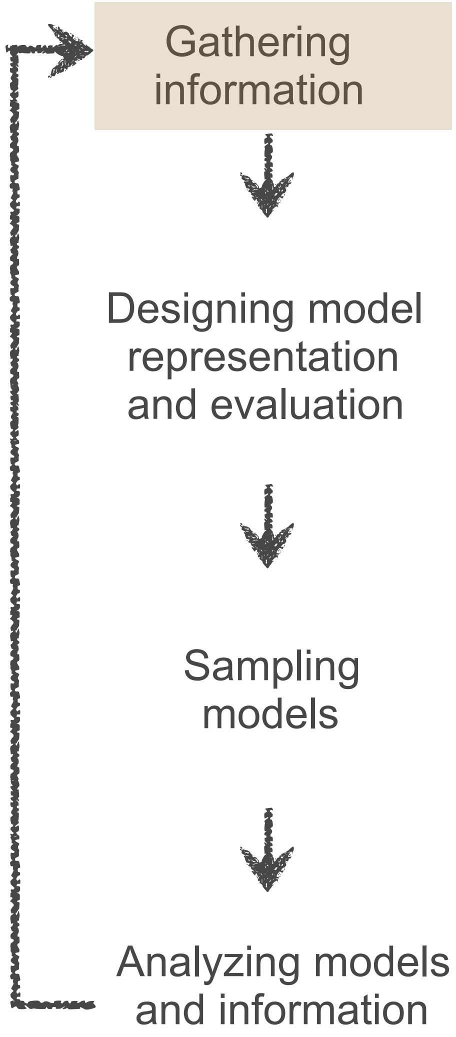
- As before, let’s get the new main modeling script running while we look at what it’s doing:

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


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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)
- Only the CX-MS data is new here

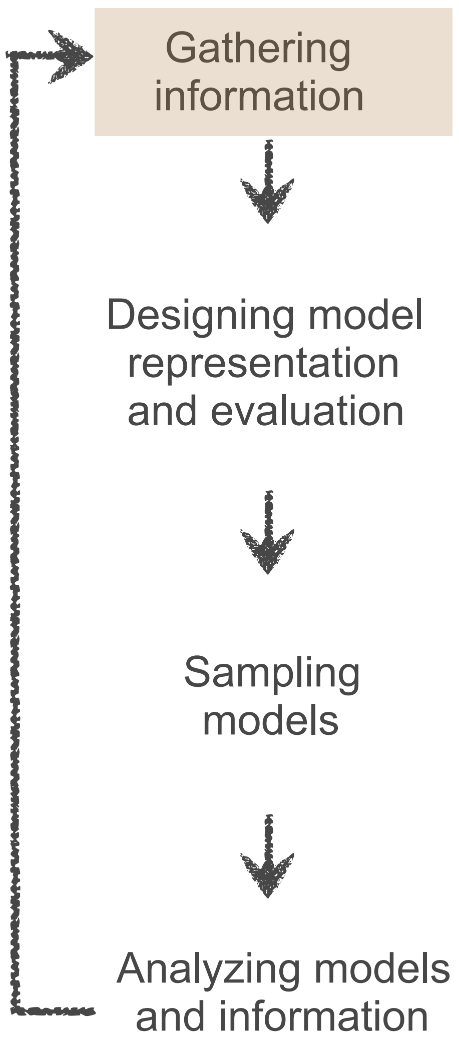

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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
```

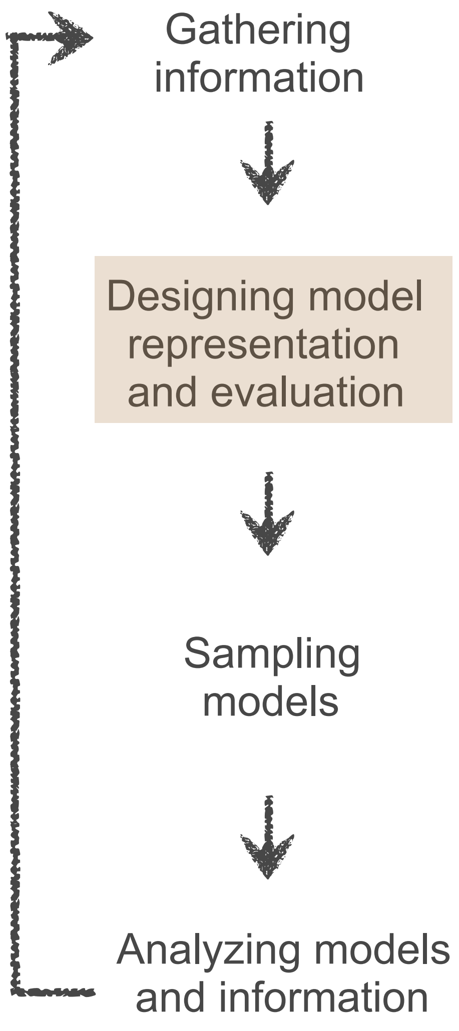
(The length of the DSS/BS3 cross-linker reagent, 21Å, is not in this file; it'll be specified in the modeling script.)


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

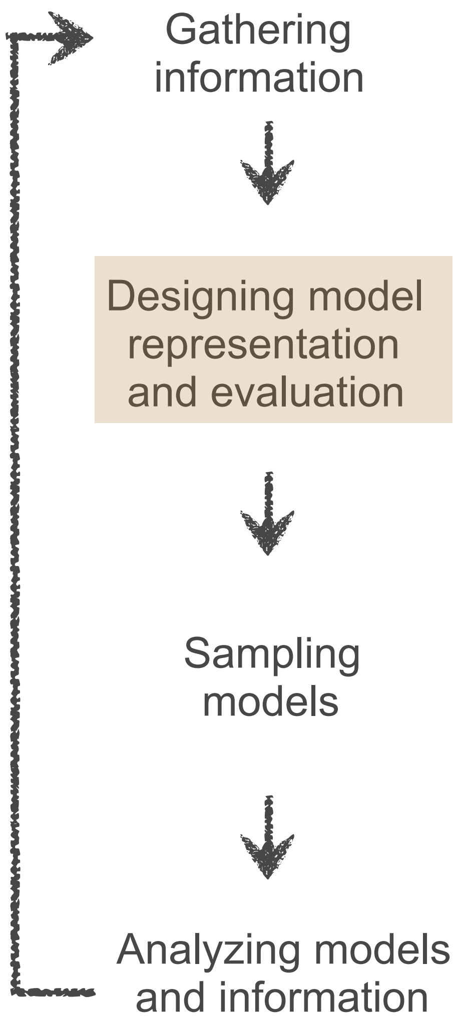
Cross-linking restraints

- 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**)

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

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


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Bayesian scoring function for XL-MS



Gathering information

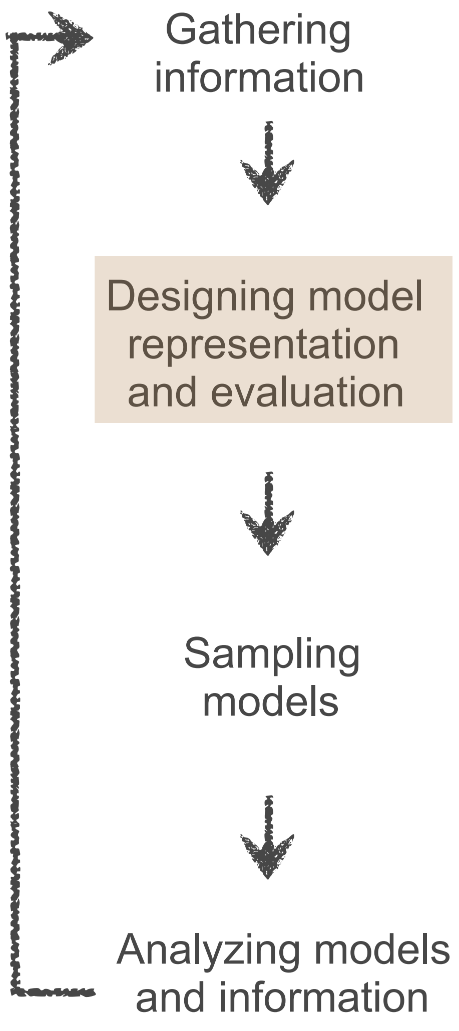
Designing model representation and evaluation

Sampling models

Analyzing models and information

Bayesian scoring function for XL-MS

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



Gathering information

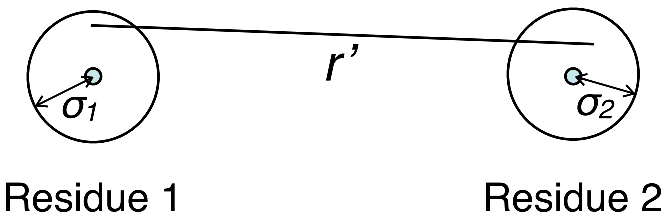
Designing model representation and evaluation

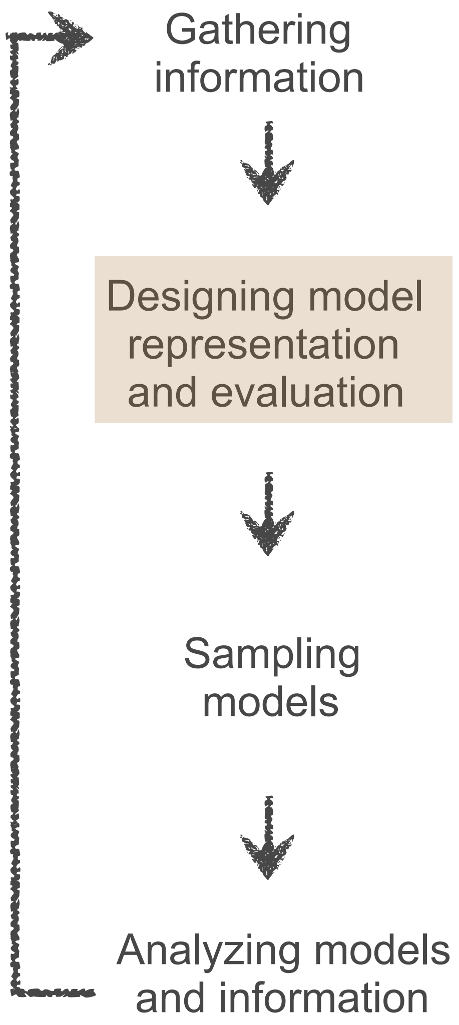
Sampling models

Analyzing models and information

Bayesian scoring function for XL-MS

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




Gathering information

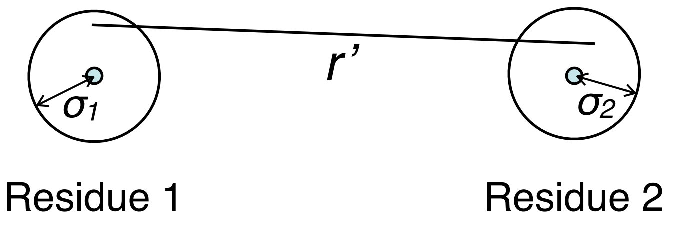
Designing model representation and evaluation

Sampling models

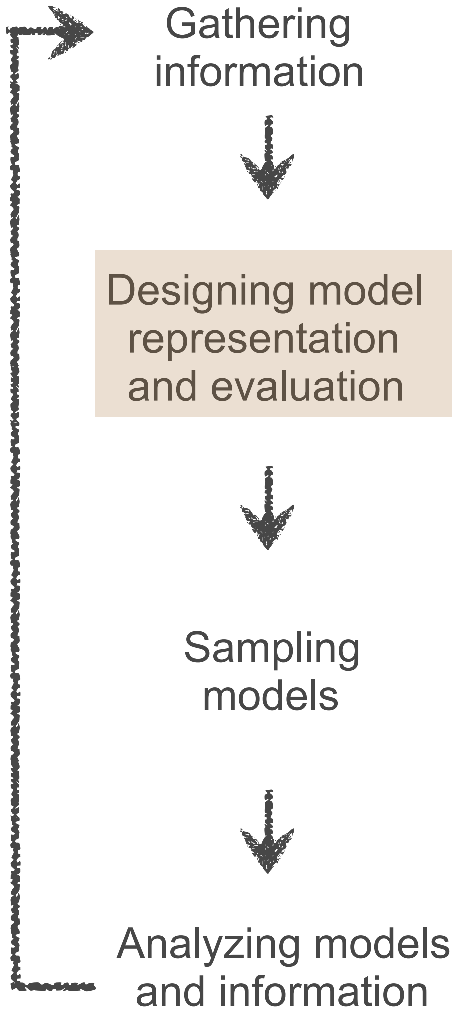
Analyzing models and information

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



Gathering information

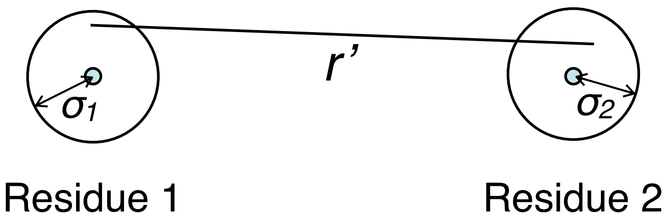
Designing model representation and evaluation

Sampling models

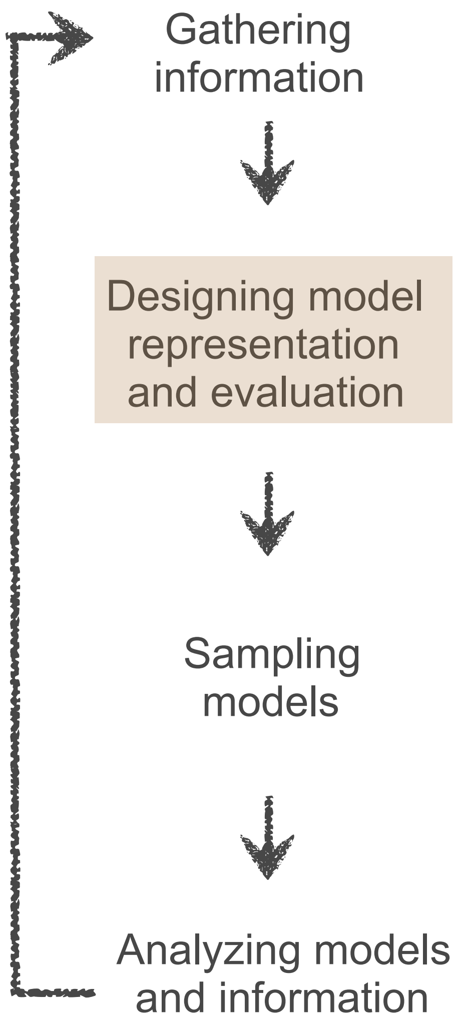
Analyzing models and information

Bayesian scoring function for XL-MS

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

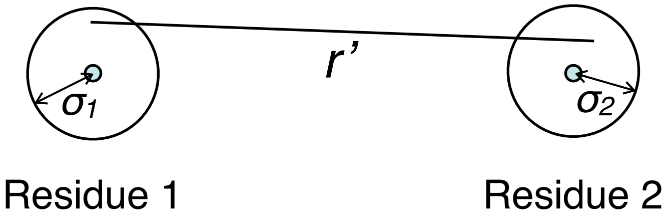


- Confidence in the cross-links themselves is measured with another parameter, ψ
- In principle, could optimize σ and ψ for *every* cross-link

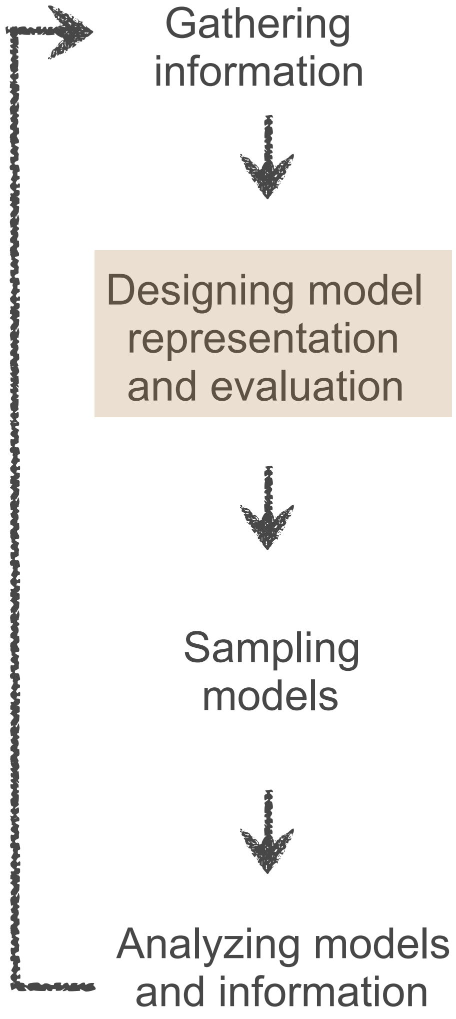

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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)



Gathering information

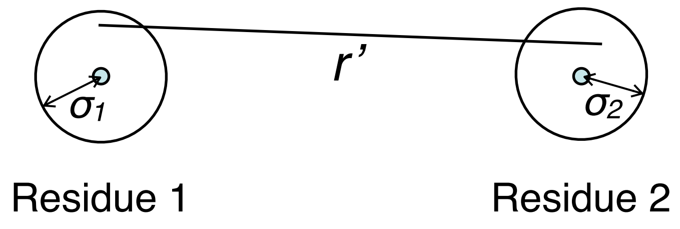
Designing model representation and evaluation

Sampling models

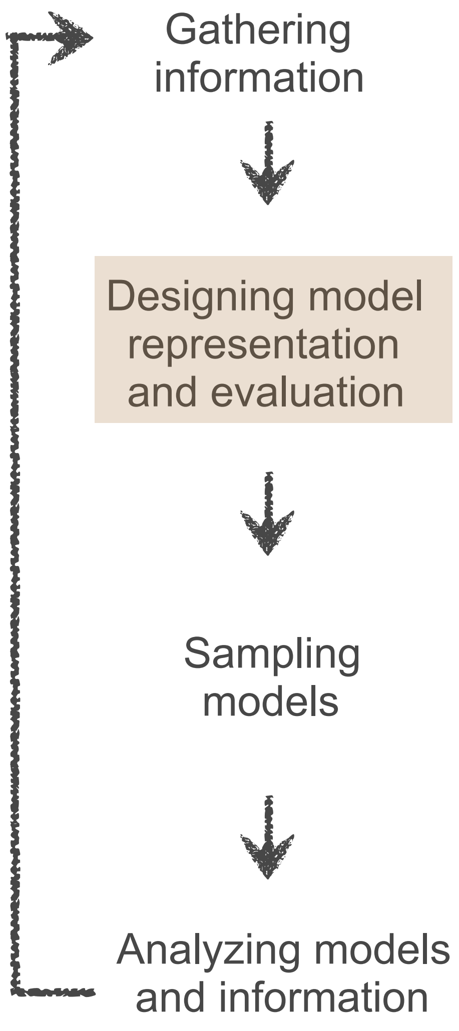
Analyzing models and information

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



Gathering information

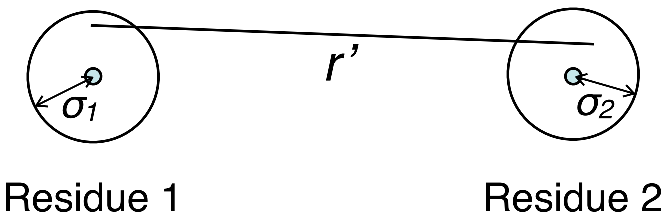
Designing model representation and evaluation

Sampling models

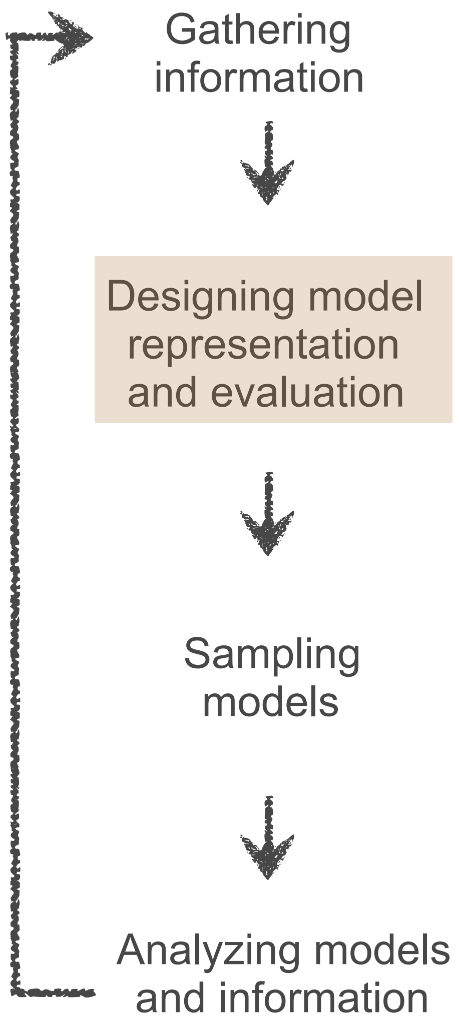
Analyzing models and information

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


Gathering information

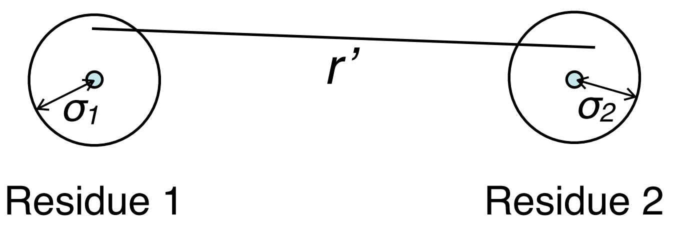
Designing model representation and evaluation

Sampling models

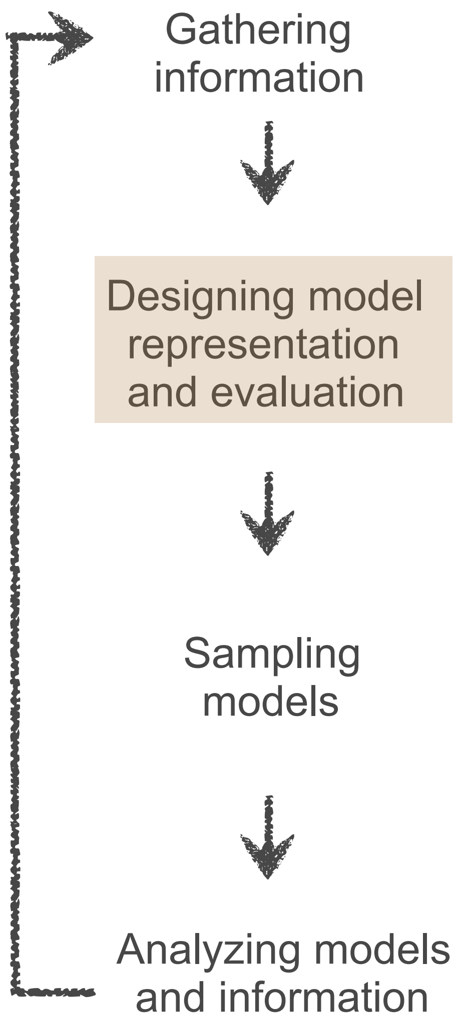
Analyzing models and information

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



Gathering information

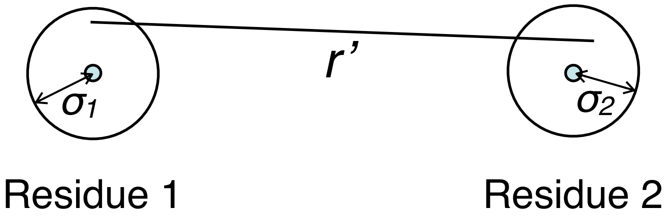
Designing model representation and evaluation

Sampling models

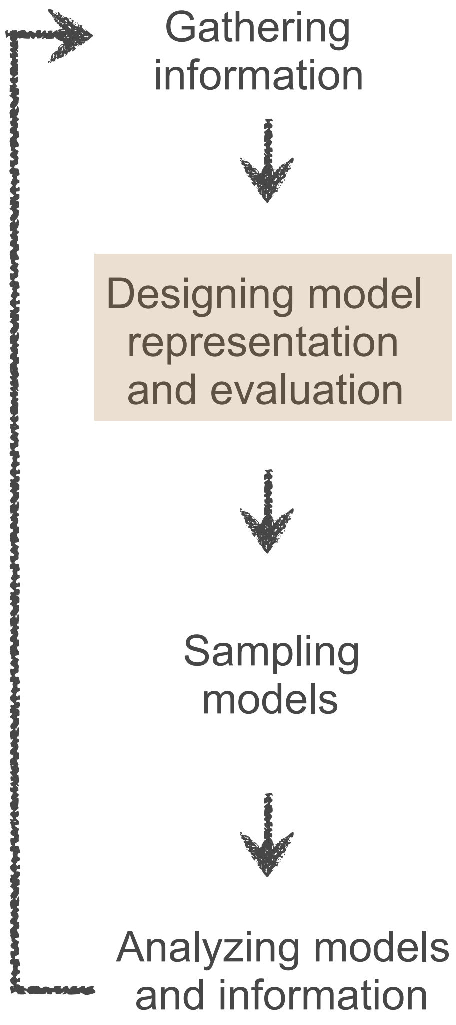
Analyzing models and information

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



Gathering information

Designing model representation and evaluation

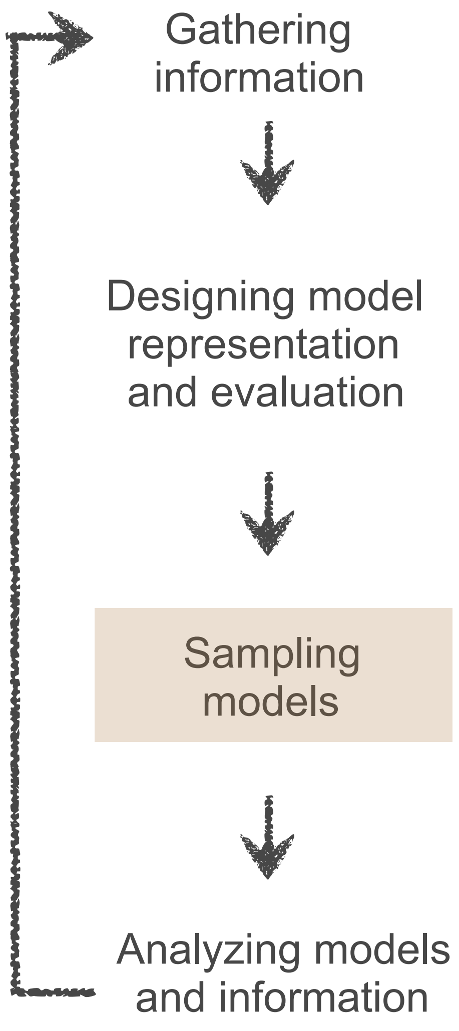
Sampling models

Analyzing models and information

Sampling

- Essentially, the same as before
- **`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`**
- As before, this generates an **`output`** directory

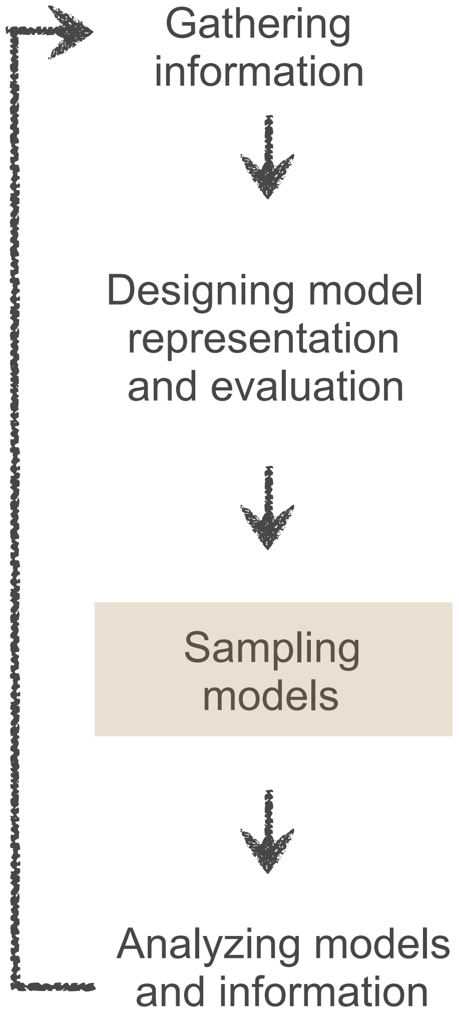
```
mc1=IMP.pmi.macros.ReplicaExchange0(m,
                               representation,
                               monte_carlo_sample_objects=sampleobjects,
                               output_objects=outputobjects,
                               crosslink_restraints=[xl1,xl2],
                               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")
```


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Modeling output

- Similar output to before, with the addition of crosslink setup info:

```
Generating a NEW crosslink restraint with a uniqueID 100
--------------
ISDCrossLinkMS: generating cross-link restraint between
ISDCrossLinkMS: residue 1 of chain Rpb4 and residue 72 of
chain Rpb6
ISDCrossLinkMS: with sigma1 1.000000 sigma2 1.000000 psi 0.05
ISDCrossLinkMS: between particles
Rpb4_1-3_bead_floppy_body_rigid_body_member and Rpb6_72_pdb
==========================================
```

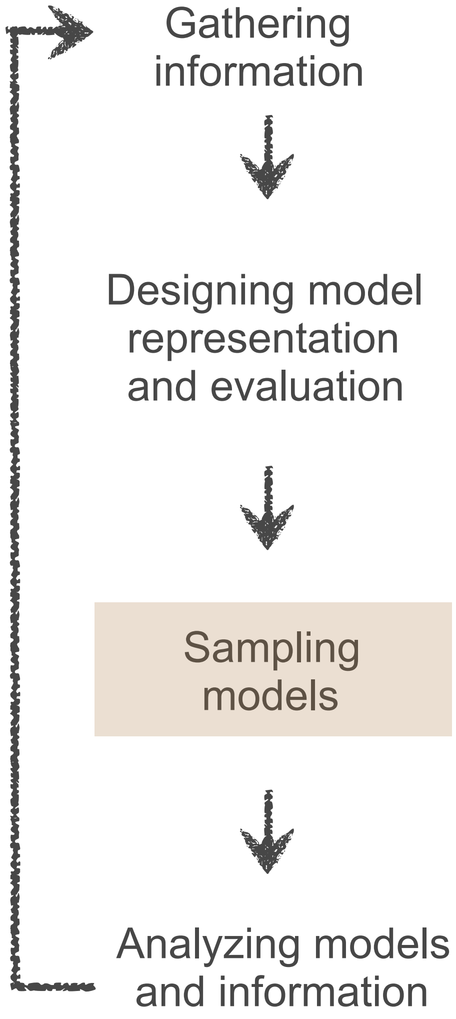

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

- Output RMF files now contain cross-link info

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
```

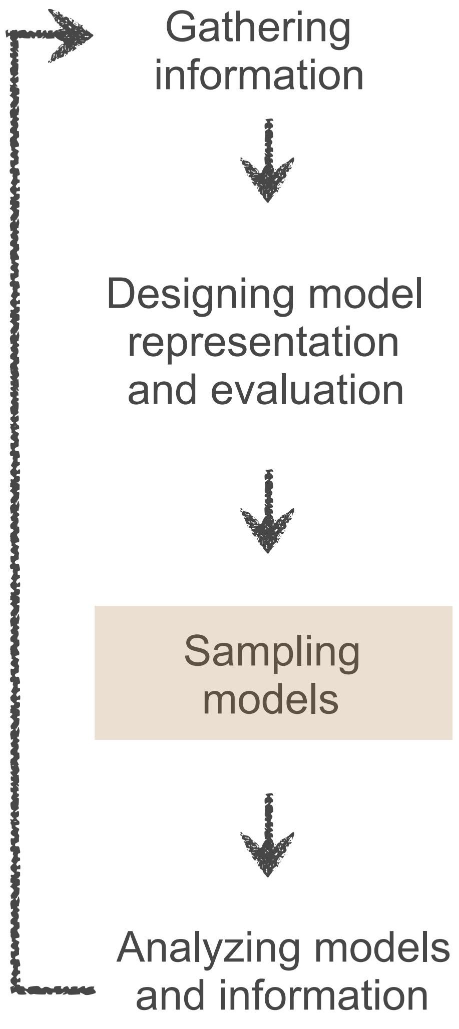

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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:

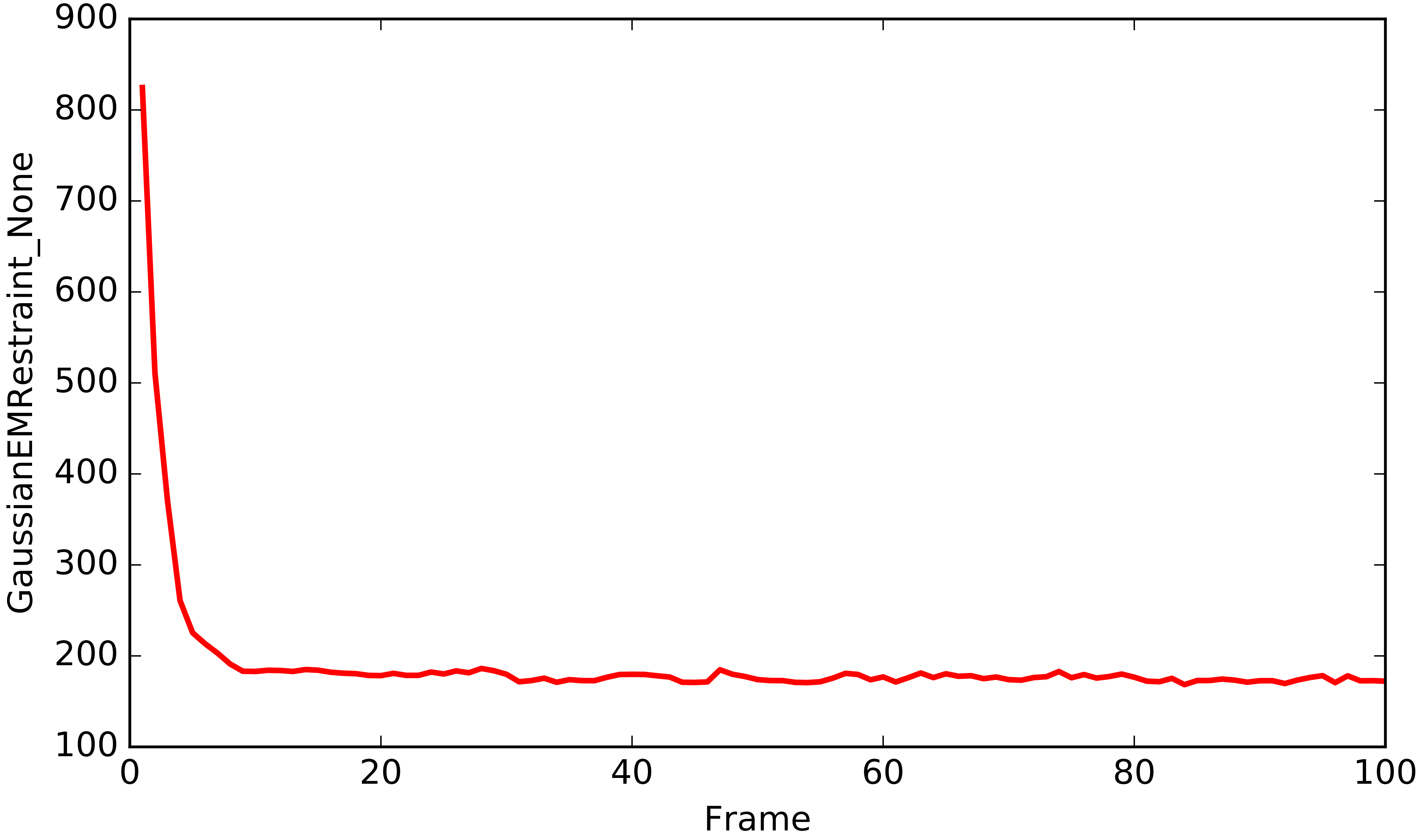
```
$ python pmi_plot_stat.py
    -y GaussianEMRestraint_None stat.0.out
```

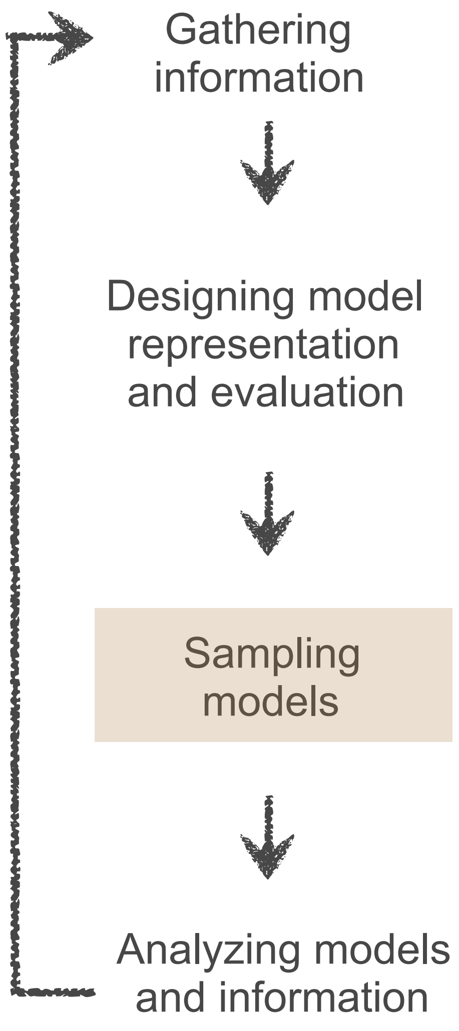
This should all be on one line


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Example plots

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




Gathering information

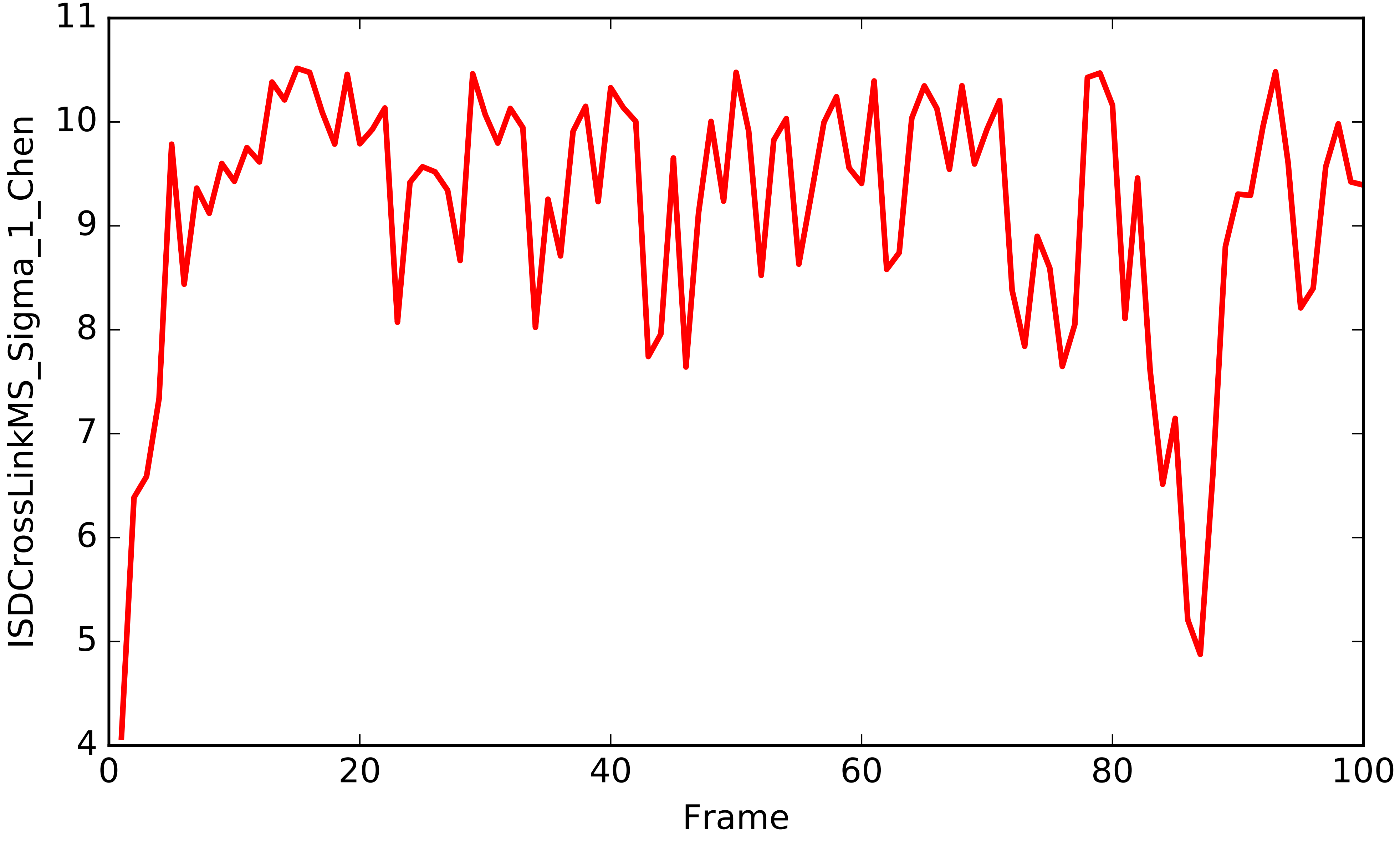
Designing model representation and evaluation

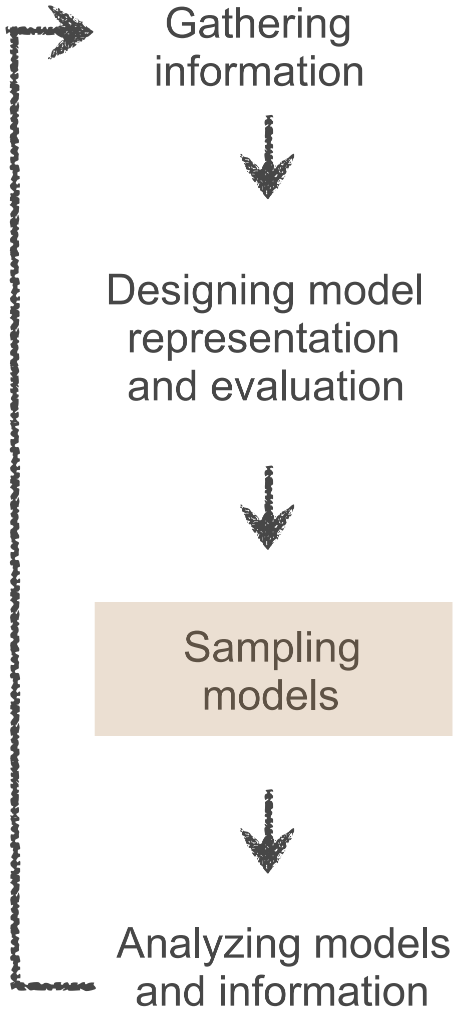
Sampling models

Analyzing models and information

Example plots

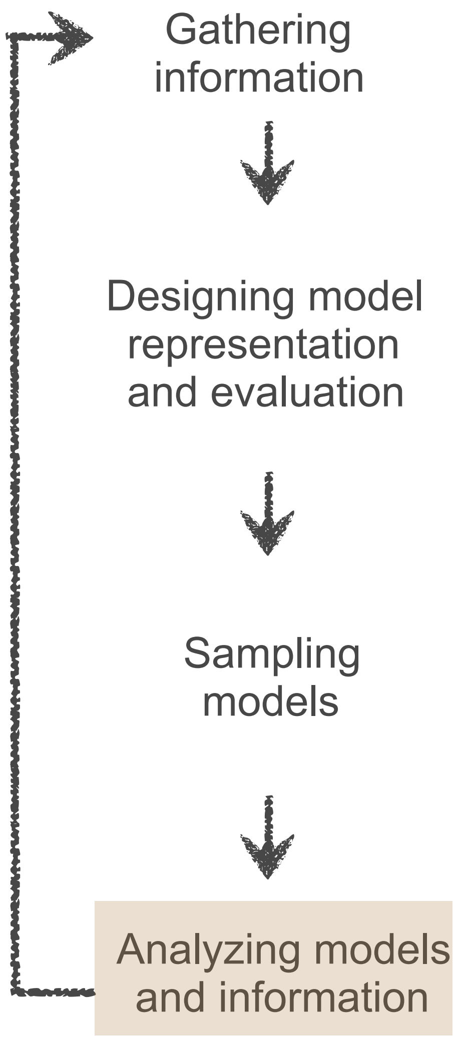
- Bayes σ parameter ends up around 10Å (makes sense given the model resolution)




Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Analysis

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


Gathering information

Designing model representation and evaluation

Sampling models

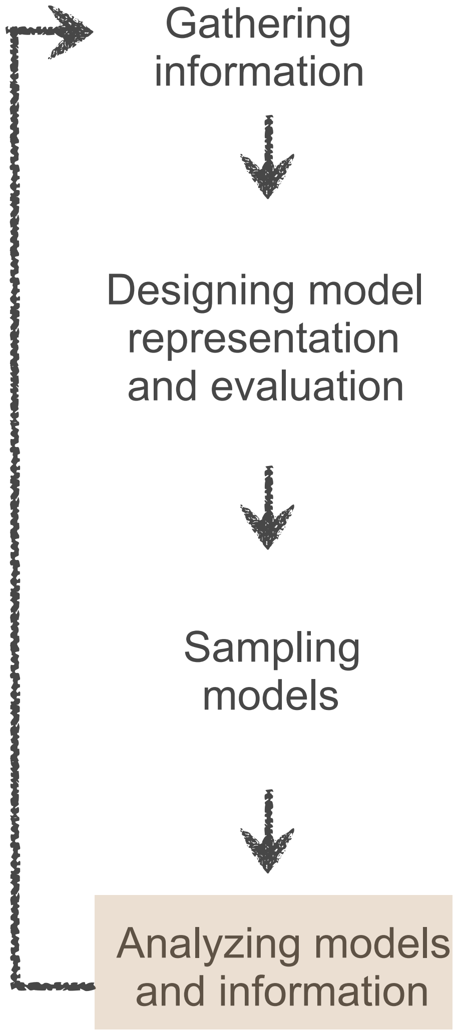
Analyzing models and information

Clustering

- We'll cluster in the same way as before, using **`rnapolii/analysis/clustering.py`**
- Simply run with
  `$ cd ../analysis`
  `$ python clustering.py`
- Almost identical to the EM-only clustering script; only **`merge_directories`** is changed

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

- As before, pre-generated analysis for the full run is also provided, in **`full_kmeans_100_2.zip`**



Gathering information

Designing model representation and evaluation

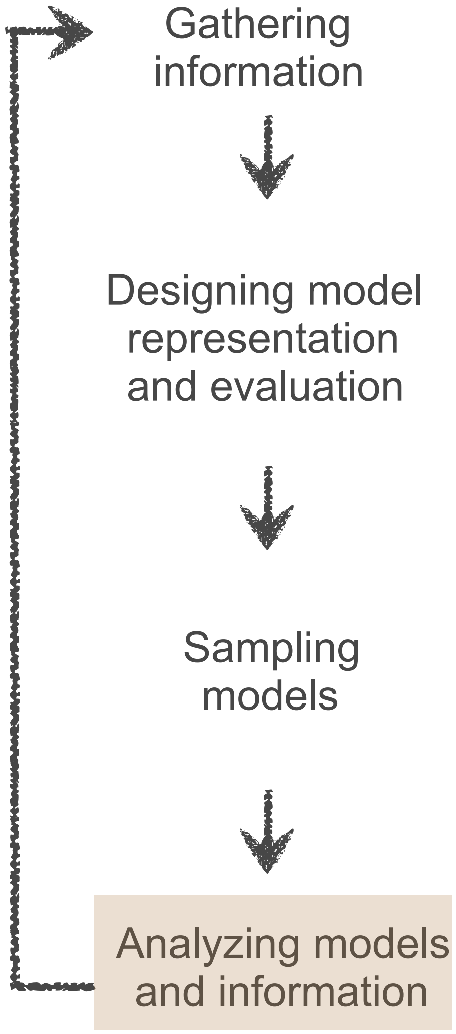
Sampling models

Analyzing models and information

Clustering: step 1, alignment

- First step in clustering is to put all structures into the same reference frame
- This is done by setting **`align_names`**, listing the subunit(s) to use as a reference
- Not needed in this case (set to special Python value **`None`**) because all structures are already aligned - to the EM map

```
align_names = None # (None because EM provides reference frame)
```



Gathering information

Designing model representation and evaluation

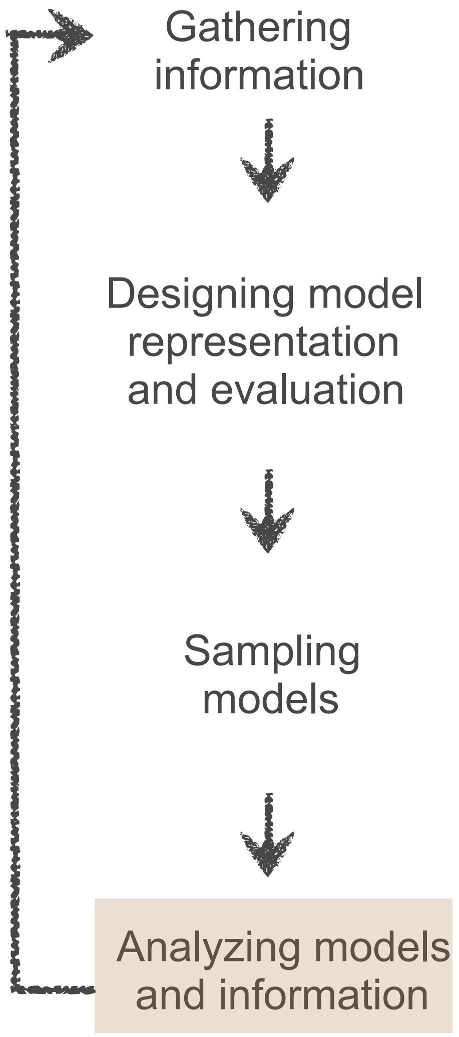
Sampling models

Analyzing models and information

Clustering: step 2, distance calculation

- Next step: calculate distances between structures (RMSD)
- This is done by setting **`rmsd_names`**
- Distances will be calculated between the subunits listed here
- k-means algorithm then proceeds using this distance metric

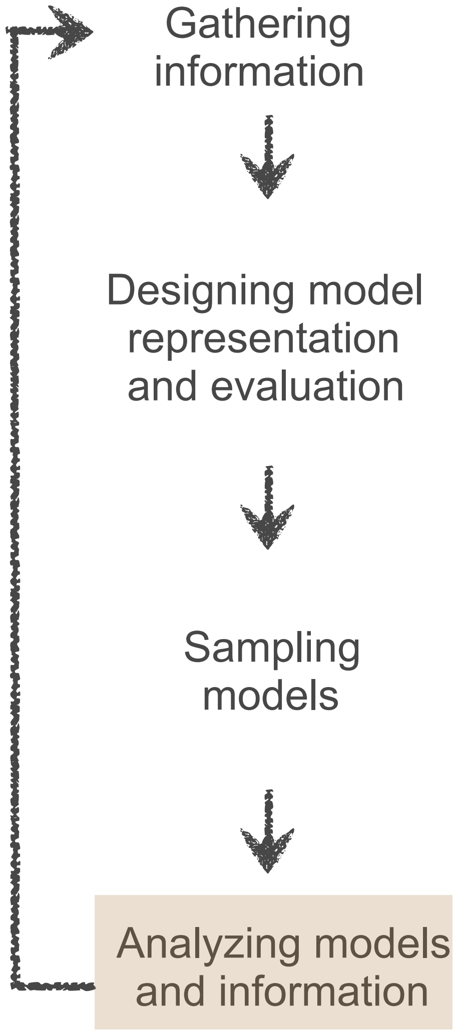
```
rmsd_names = {"Rpb4":"Rpb4",
              "Rpb7":"Rpb7"}
```


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Clustering: step 3, localization densities

- Calculate localization densities for selected subunits
- This is done by setting **`density_names`**
  - Key: output file names
  - Value: list of subunits to calculate localization density of

```
density_names = {"Rpb4":["Rpb4"],
                 "Rpb7":["Rpb7"]}
```


Gathering information

Designing model representation and evaluation

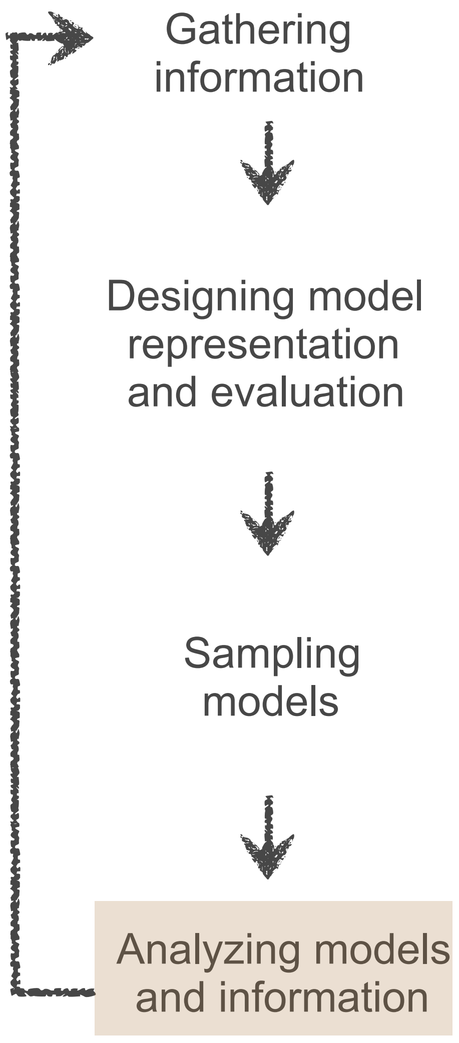
Sampling models

Analyzing models and information

Clustering: step 4, statistics

- A cluster-specific stats file is also generated
- We can also ask for features to be copied in from the original stats file by setting **`feature_list`**

```
feature_list=["ISDCrossLinkMS_Distance_intrarb",
              "ISDCrossLinkMS_Distance_interrb",
              "ISDCrossLinkMS_Data_Score",
              "GaussianEMRestraint_None",
              "SimplifiedModel_Linker_Score_None",
              "ISDCrossLinkMS_Psi",
              "ISDCrossLinkMS_Sigma"]
```



Gathering information

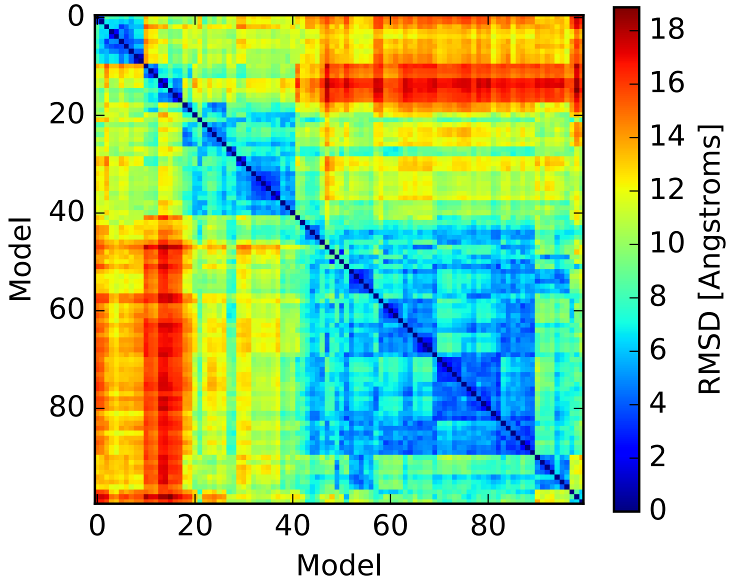
Designing model representation and evaluation

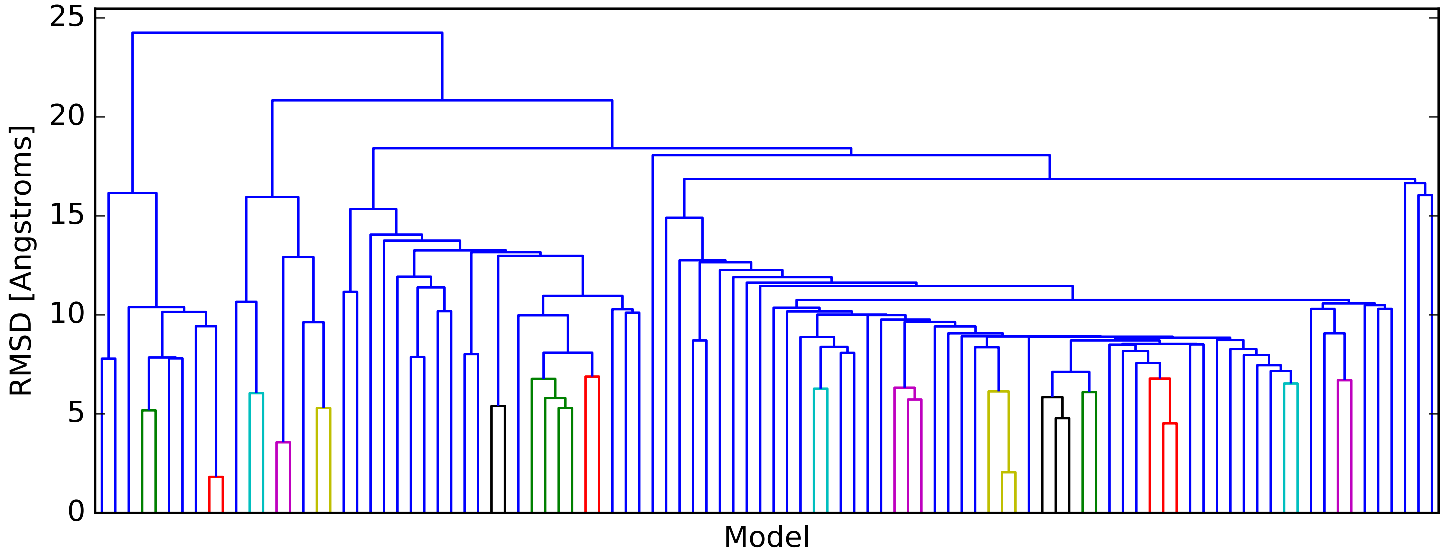
Sampling models

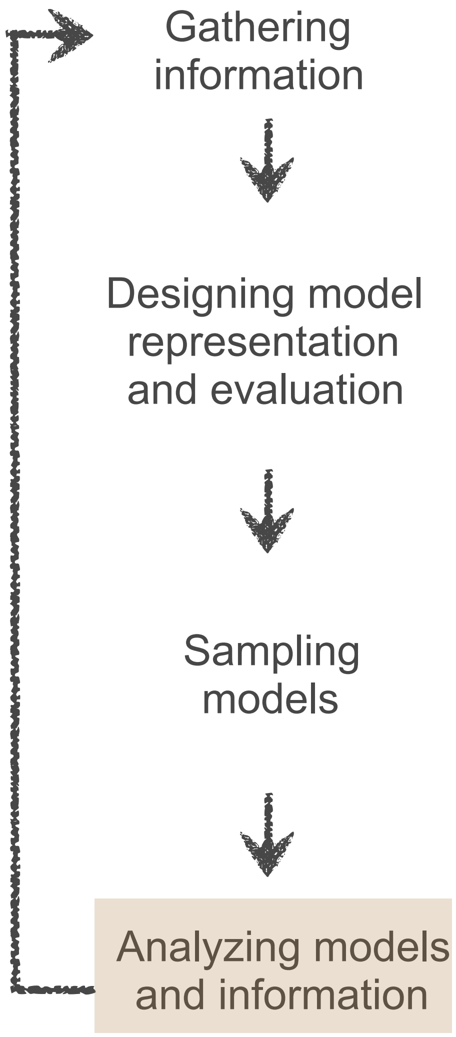
Analyzing models and information

Clustering output

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







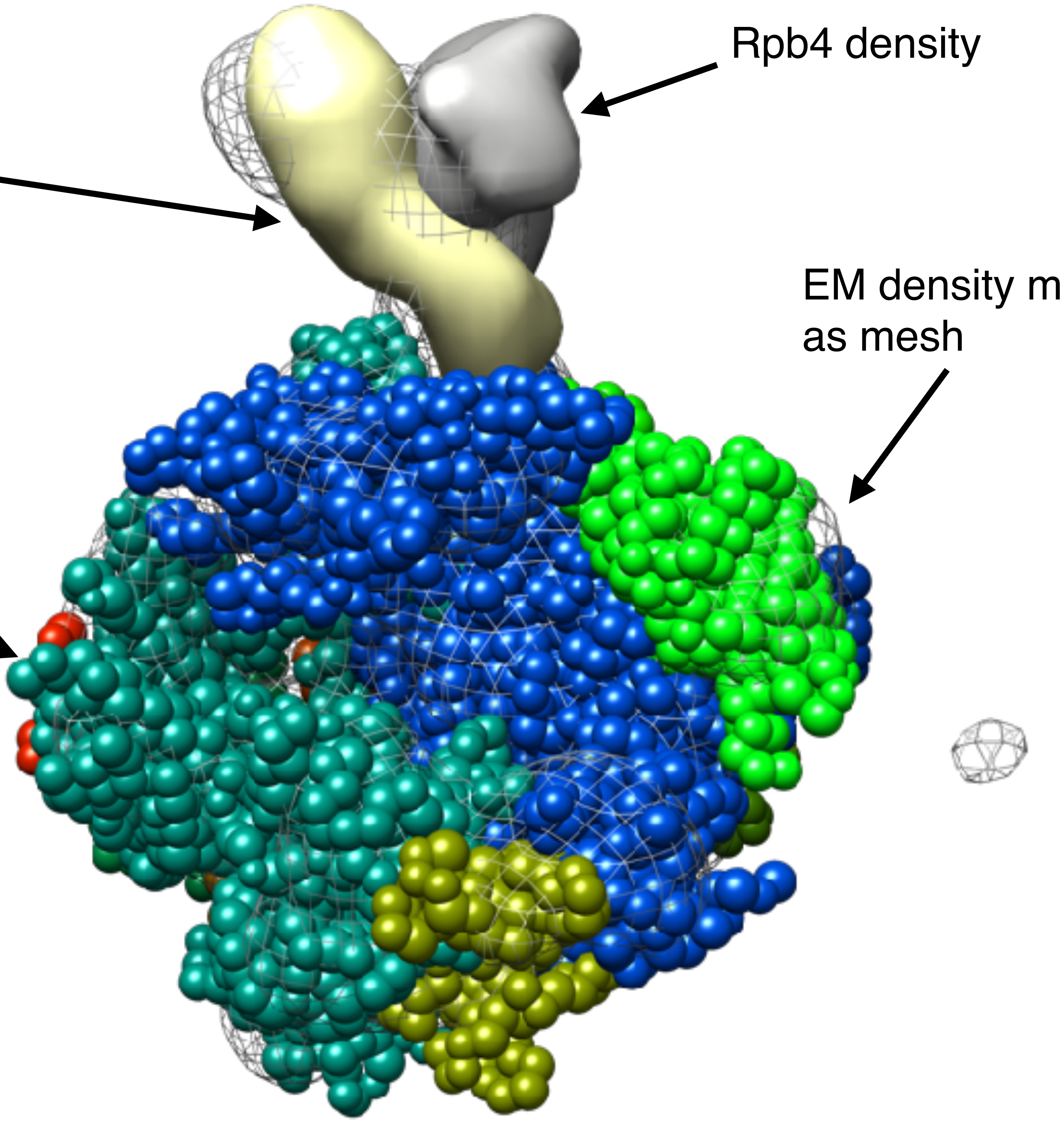
Gathering information

Designing model representation and evaluation

Sampling models

Analyzing models and information

View in Chimera

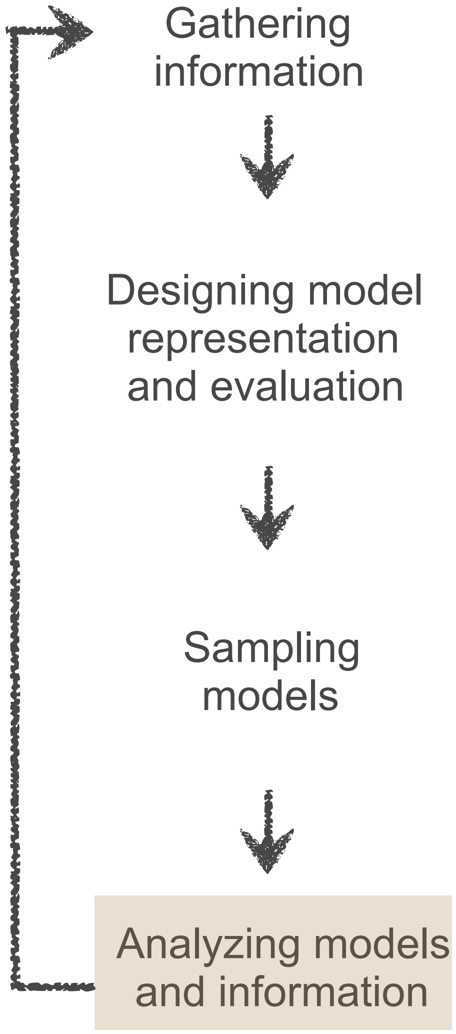


Rpb4 density

Rpb7 density

EM density map, as mesh

Native structure of other subunits (1-residue beads)



Gathering information

Designing model representation and evaluation

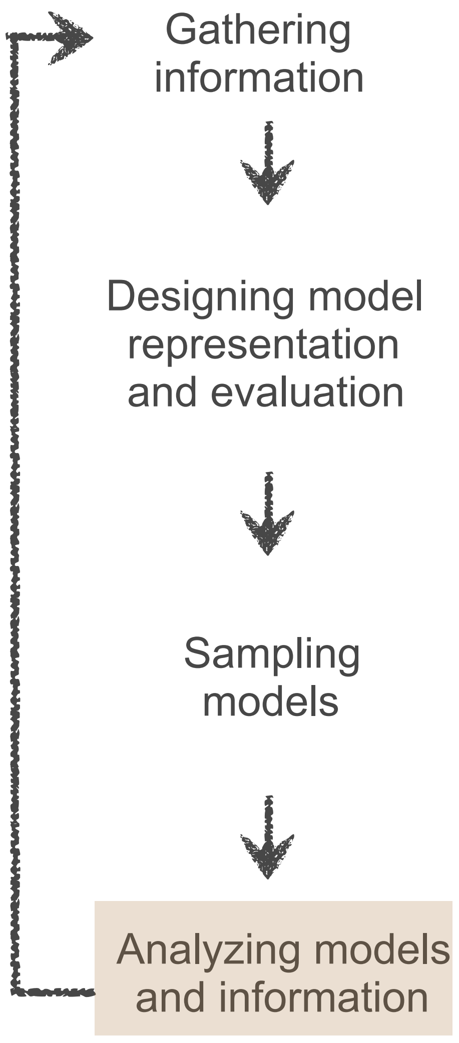
Sampling models

Analyzing models and information

Cluster precision

- Now that we’ve clustered, we can determine the within- and between-cluster RMSD, i.e. precision
- Do this with the `precision_rmsf.py` script:
  `$ python precision_rmsf.py`
- As before, we need to specify in this script the subunits we want the precision of (here, each of Rpb4 and Rpb7, plus both of them together):

```
selections={"Rpb4":["Rpb4"],
            "Rpb7":["Rpb7"],
            "Rpb4_Rpb7":["Rpb4","Rpb7"]}
```


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Precision output

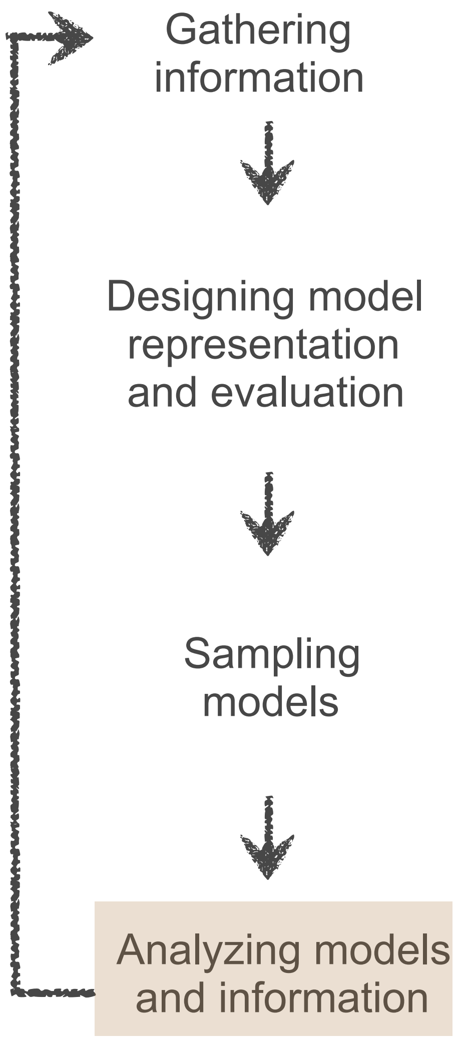
- Generates, inside the cluster output directories:

  `precision.0.0.out` ← Precision of cluster.0

  `precision.1.1.out` ← Precision of cluster.1

  `precision.0.1.out` ← Between-cluster precision

- Each shows the precision of the requested subunits

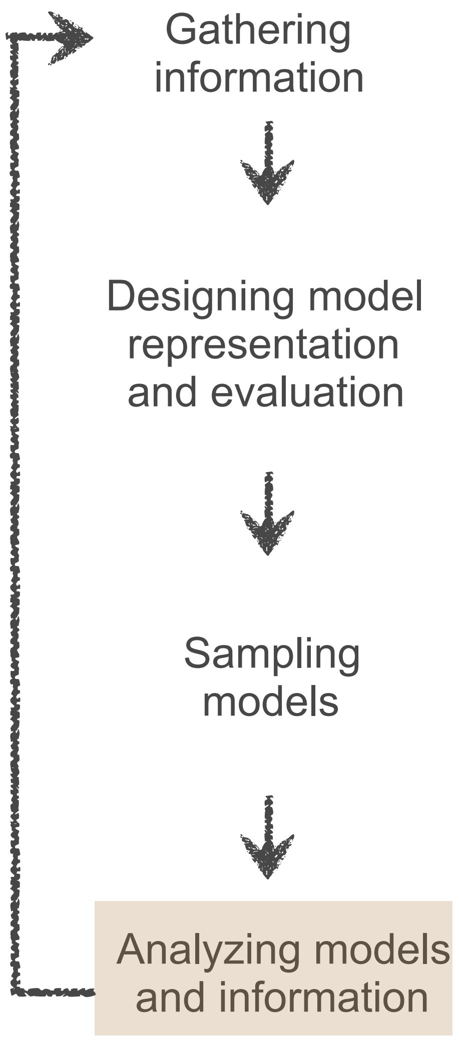

Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

RMSF output

- For each cluster, generates
  `rmsf.Rpb4.dat`
  `rmsf.Rpb4.pdf`

Raw RMSF data (fluctuation of each residue’s position over all models in the cluster) for Rpb4

Plot of this data



Gathering information

Designing model representation and evaluation

Sampling models

Analyzing models and information

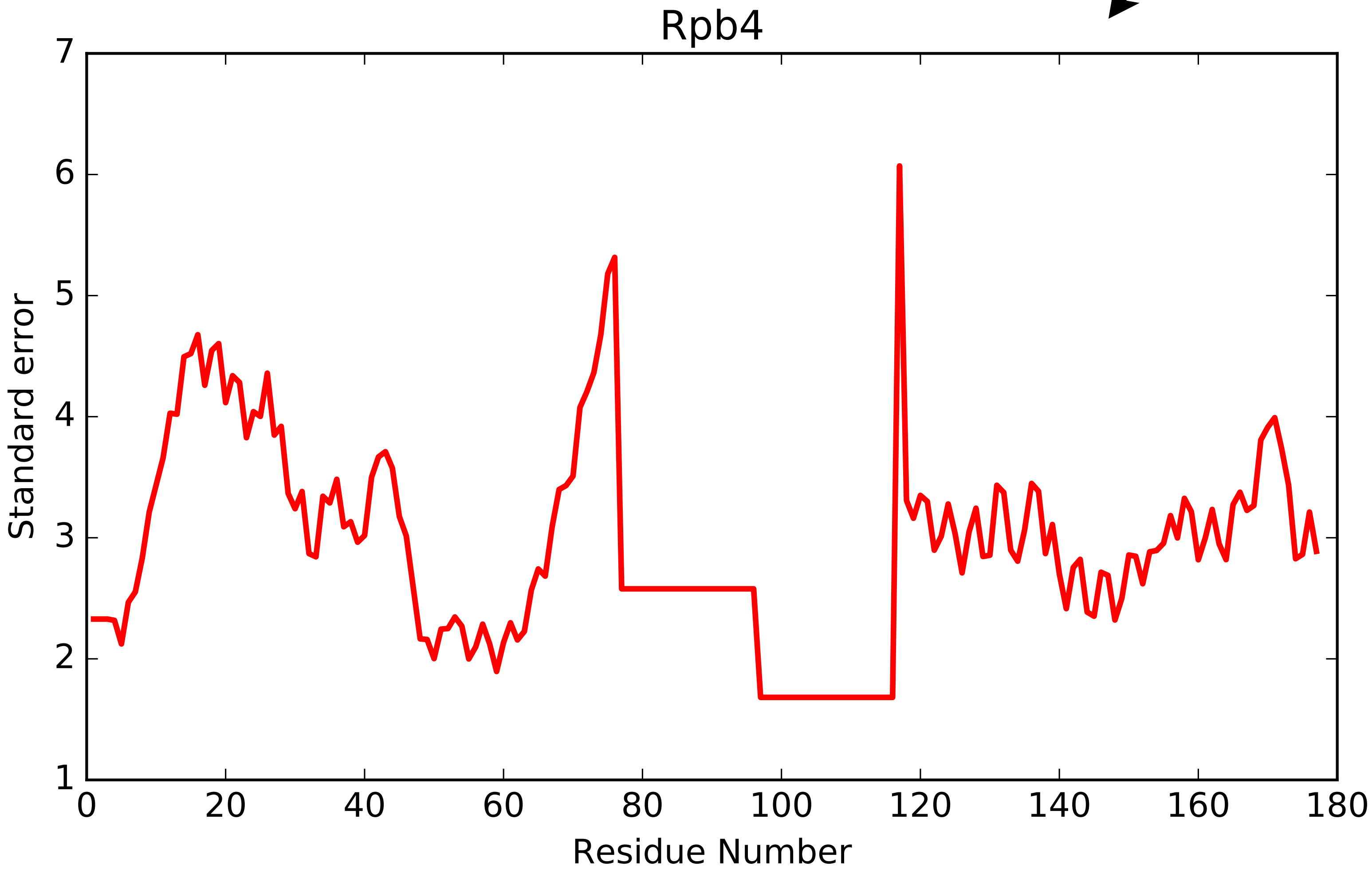
RMSF output

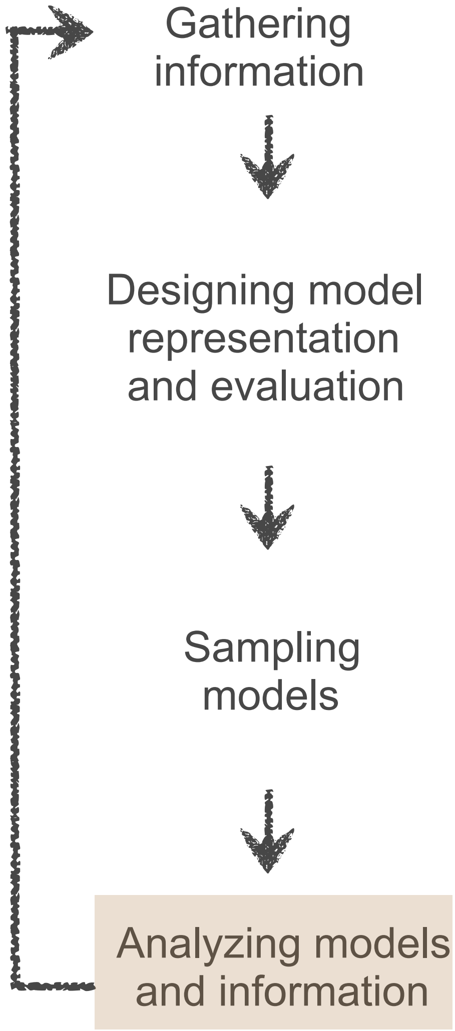
- For each cluster, generates
  `rmsf.Rpb4.dat`

  `rmsf.Rpb4.pdf`

Raw RMSF data (fluctuation of each residue's position over all models in the cluster) for Rpb4

Plot of this data



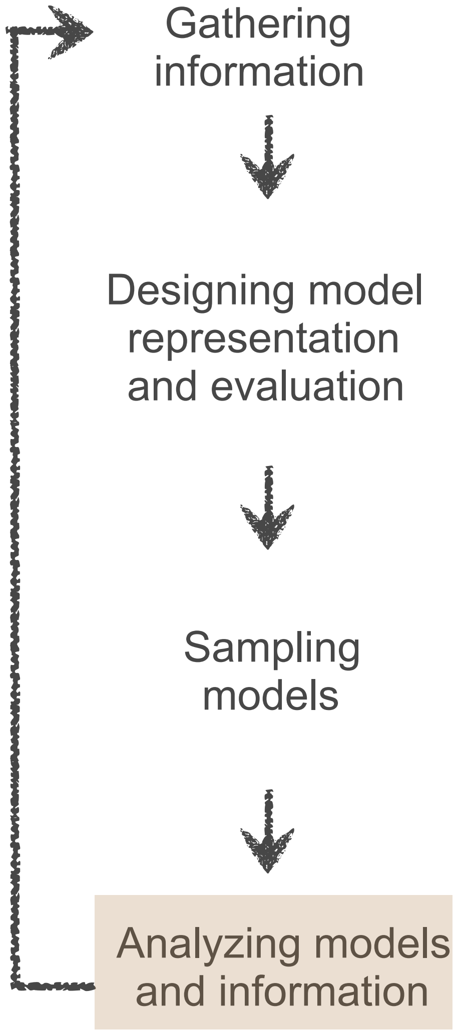


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Cluster accuracy

- If we know the native structure, we can compare each of the cluster models against it to quantify the accuracy
- Do this with the `accuracy.py` script:
  `$ python accuracy.py`
- We select our subunits exactly as for precision, and also need to provide the reference structure and the set of models to compare:

```
reference_rmf = "../data/native.rmf3"
rmfs = glob.glob('kmeans_*_*/cluster.0/*.rmf3')
```



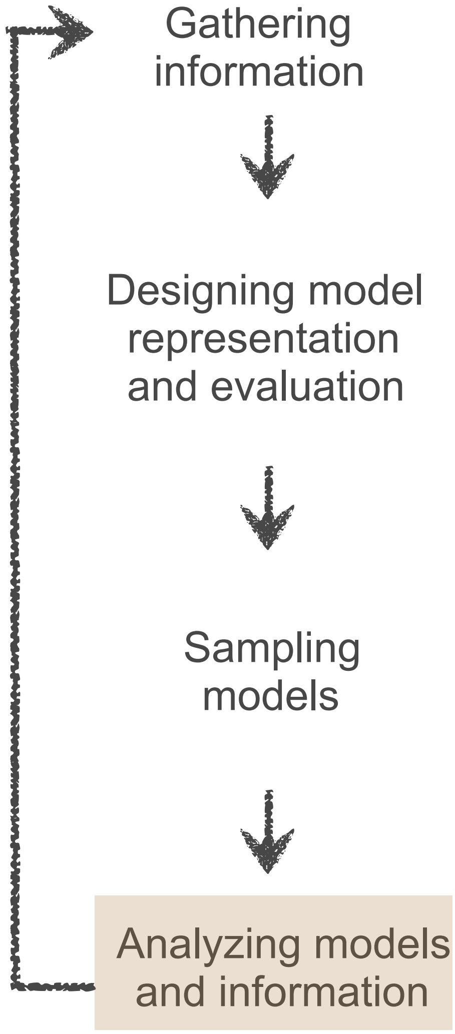
Gathering information

Designing model representation and evaluation

Sampling models

Analyzing models and information

Sampling exhaustiveness



Gathering information

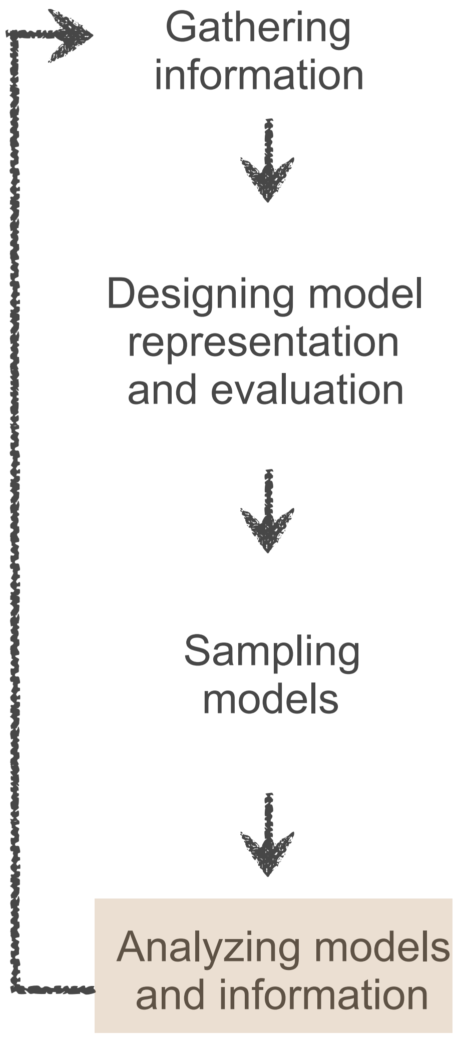
Designing model representation and evaluation

Sampling models

Analyzing models and information

Sampling exhaustiveness

- How confident can we be that we’ve done enough sampling?



Gathering information

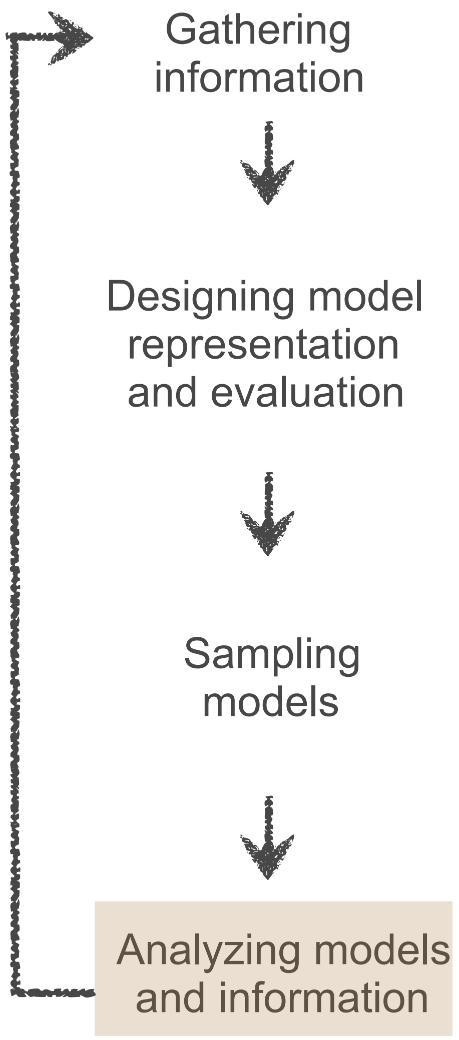
Designing model representation and evaluation

Sampling models

Analyzing models and information

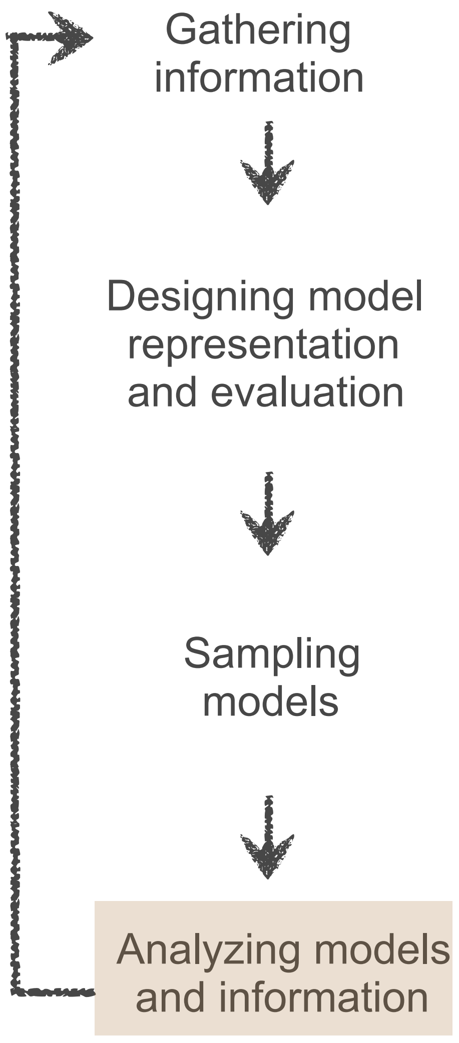
Sampling exhaustiveness

- How confident can we be that we’ve done enough sampling?
  - a variety of methods exist, not covered in this tutorial


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

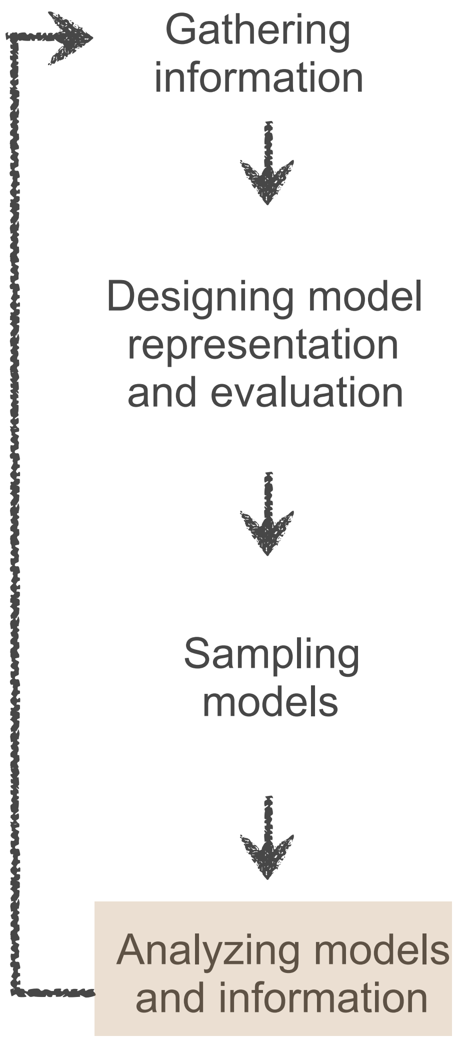
Sampling exhaustiveness

- How confident can we be that we've done enough sampling?
  - a variety of methods exist, not covered in this tutorial
  - for example, two independent runs should sample from the same distribution - can test statistically (χ^2 test), or by comparing clusters (as in the Nup84 study)


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

Sampling exhaustiveness

- How confident can we be that we've done enough sampling?
  - a variety of methods exist, not covered in this tutorial
  - for example, two independent runs should sample from the same distribution - can test statistically (χ^2 test), or by comparing clusters (as in the Nup84 study)
  - model leaving out some of the data (jackknife, compare with R free calculation)



Gathering information

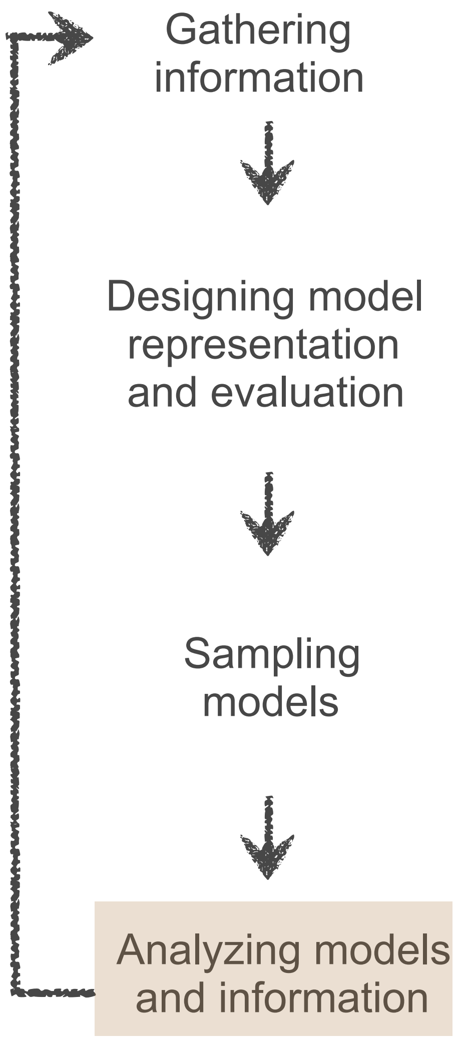
Designing model representation and evaluation

Sampling models

Analyzing models and information

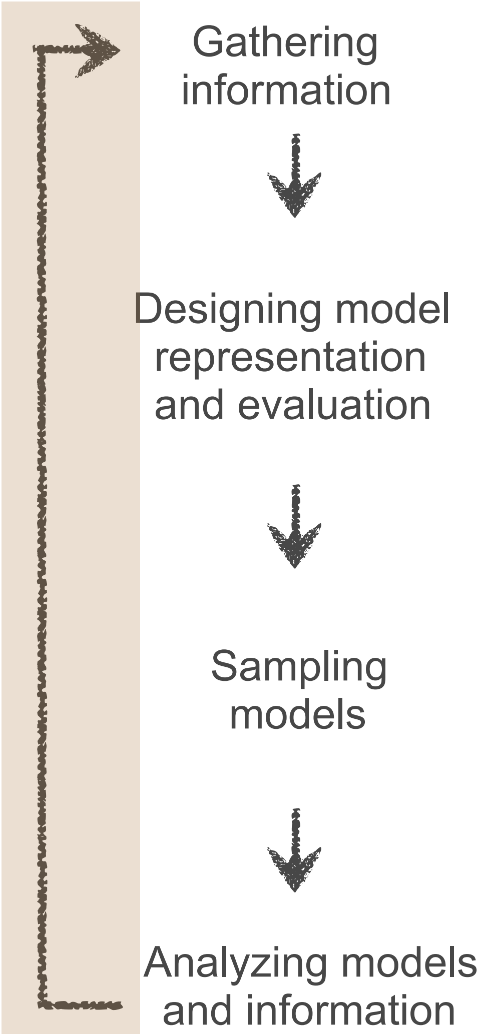
Sampling exhaustiveness

- How confident can we be that we’ve done enough sampling?
  - a variety of methods exist, not covered in this tutorial
  - for example, two independent runs should sample from the same distribution - can test statistically (χ^2 test), or by comparing clusters (as in the Nup84 study)
  - model leaving out some of the data (jackknife, compare with R free calculation)
  - Daniel will talk more about this tomorrow


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

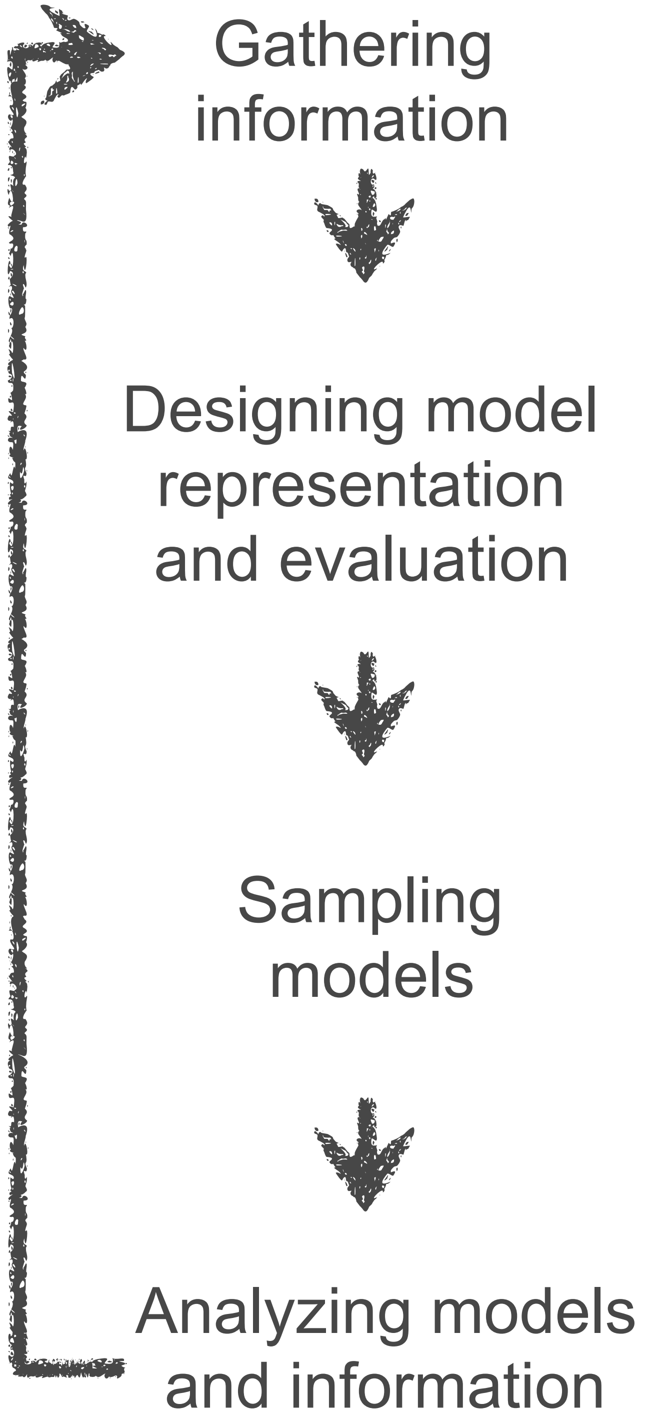
Iteration

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


Gathering information
Designing model representation and evaluation
Sampling models
Analyzing models and information

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



Gathering information

Designing model representation and evaluation

Sampling models

Analyzing models and 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.