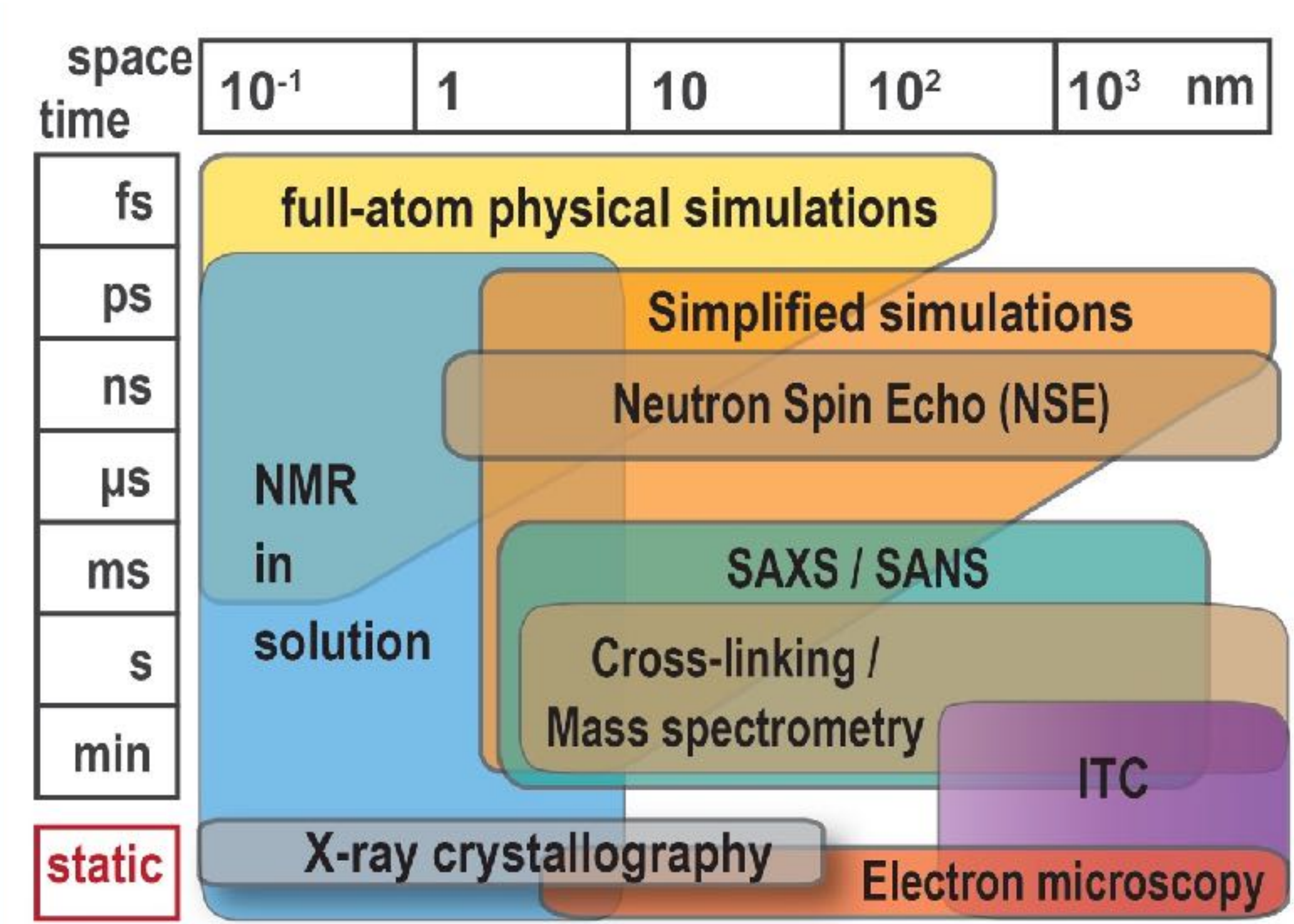
Using New Experimental Data in IMP

VMD/IMP Workshop

Seth Axen and Barak Raveh

Challenge: for a typical complex biological system, no single experimental or theoretical approach is generally *accurate*, *precise*, *complete* or *efficient* at all scales of interest



Given the complex questions, need quantitative models of that integrate multiple sources of data

A general approach for integrative modeling:

Integrate information from multiple source and scales to maximize:

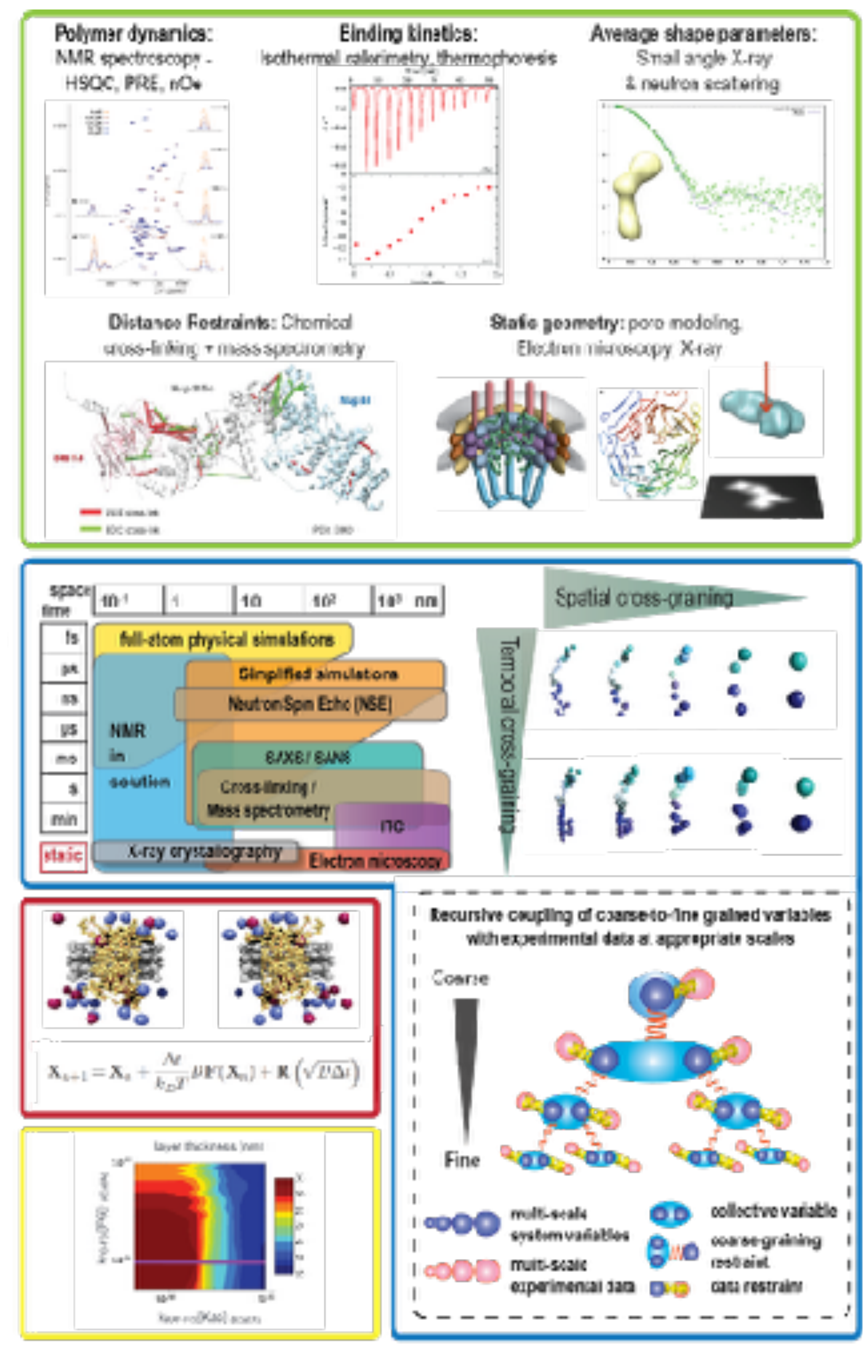
- accuracy - correctly describes all relevant aspects of the system
- precision - minimize the variance among different random solutions
- completeness - describe the entire system at every relevant spatial and temporal scale
- efficiency - derive the model rapidly and inexpensively

1. **Gather information**

2. Formally **represent** parametrized model and external information at appropriate scales

3. **Sample** favorable model instances

4. **Analyze and validate** parametrized models selected



Options for data integration

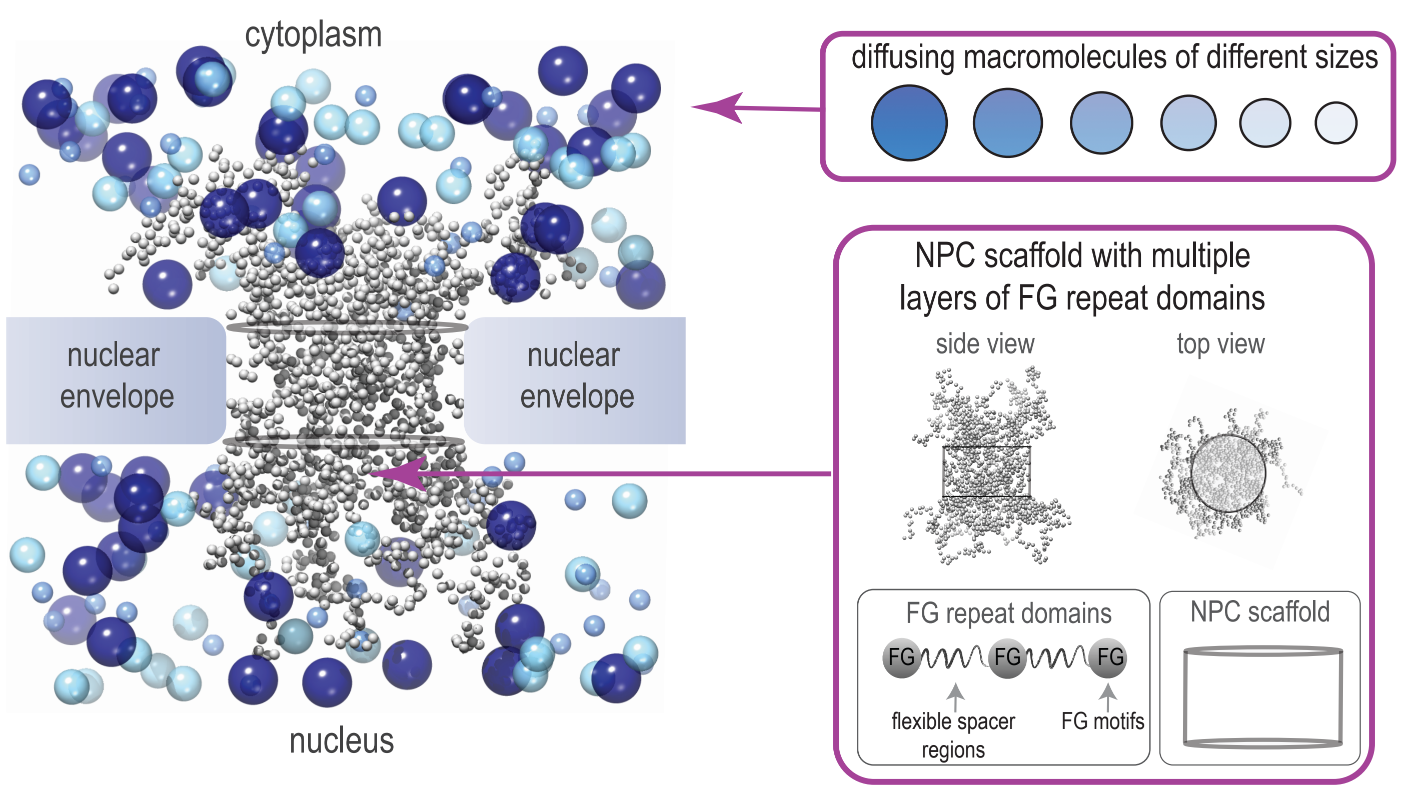
1. **Representation**
   Represent the system parts and their interactions to reflect our prior knowledge
2. **Scoring**
   Score different models based on data fit
3. **Sampling**
   Use sampling procedures that pick data-compatible models
4. **Filtering**
   Reject inconsistent models
5. **Validation and Testing**
   Contrast final models with data

Options for data integration

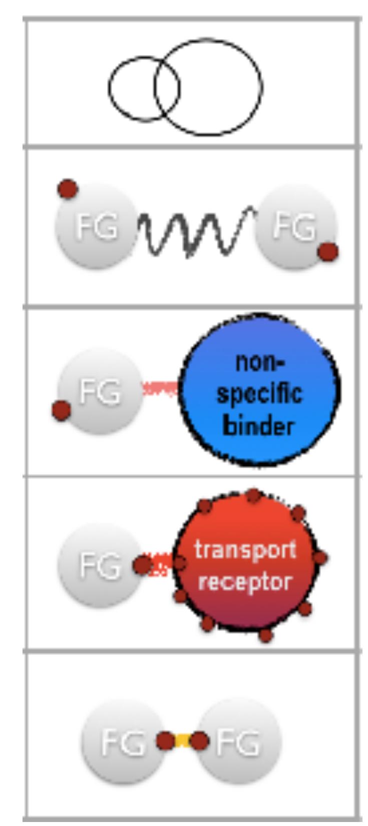
1. **Representation**
   Represent the system parts and their interactions to reflect prior information

Example: coarse-grained model of the nucleocytoplasmic transport
(*Timney et al., JCB 2016*)

The system parts



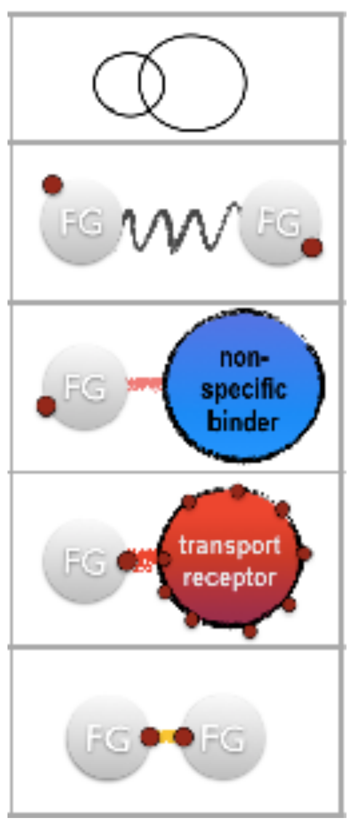
Interactions



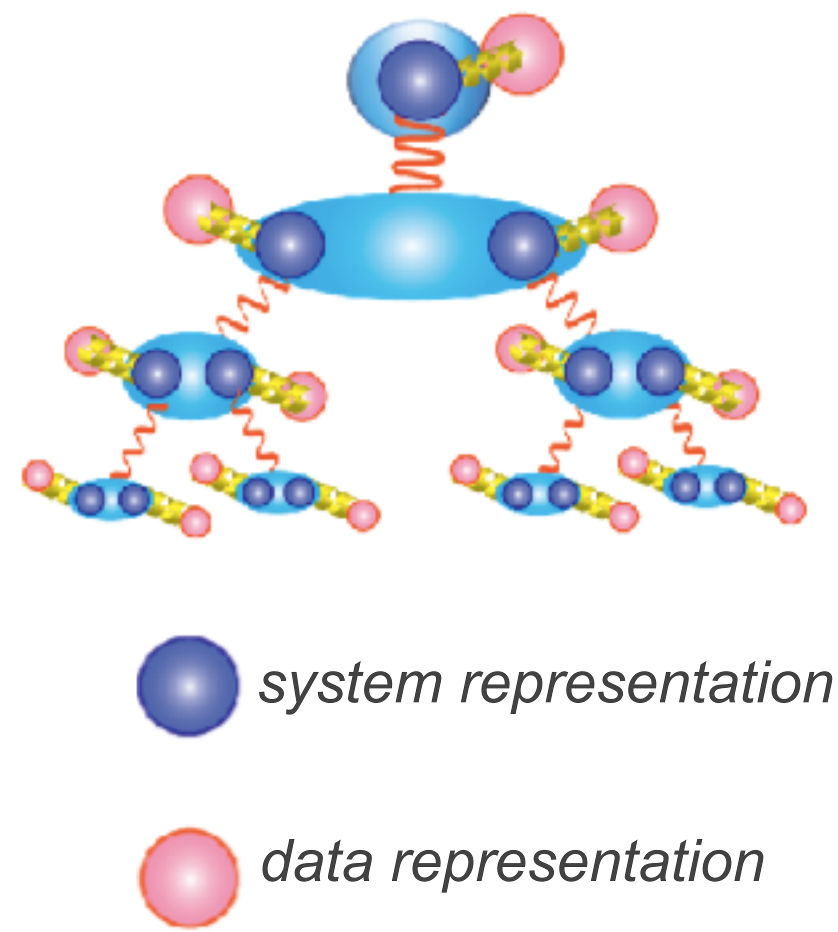
Options for data integration

2. **Scoring**
   Score different models based on their fit to the data

Parameterize interactions to reflect specific measurements



Data restraints



What is a Restraint?

- A restraint is any function that measures the degree of consistency between a given model and some expectation.
- Example: the hybrid energy function combines weighted data-based with physics-based restraints.

$$E_{hybrid} = E_{phys} + w_{data} E_{data}$$

- We often have more data-based restraints than physics-based restraints.

Types of Restraints

- Traditional (physical, spatial)

$$V = \sum_{\text{bonds}} k_b (b - b_0)^2 + \sum_{\text{angles}} k_\theta (\theta - \theta_0)^2 + \sum_{\text{dihedrals}} k_\phi [1 + \cos(n\phi - \delta)] + \sum_{\text{impropers}} k_\omega (\omega - \omega_0)^2 + \sum_{\text{Urey-Bradley}} k_u (u - u_0)^2 + \sum_{\text{nonbonded}} \epsilon \left[\left(\frac{R_{\min_{ij}}}{r_{ij}} \right)^{12} - \left(\frac{R_{\min_{ij}}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{\epsilon r_{ij}}$$

$$s(x, m, k) = \frac{1}{2} k (x - m)^2$$

MacKerell et. al. J. Phys. Chem. B, 102:3586-3616, 1998.

- Probabilistic (Bayesian)

Posterior Probability
"probability of a model, assuming data and prior information"

Likelihood
"Probability of observing the data, assuming the model and prior information"

Prior
"Probability of a model, assuming only prior information"

$$p(M \mid D, I) \propto p(D \mid M, I) p(M \mid I)$$

Forward Model
Error/Uncertainty Model

$$S(M, D, I) = -\ln \left[p(D \mid M, I) p(M \mid I) \right]$$

Advantages of Bayesian Restraints

- Handle uncertainty rigorously and infer it as a function of data or coordinates.
- Model unknown nuisance parameters.
- Monte Carlo integration enables calculation of posterior probability of any given model.
- Generate models that are maximally accurate and optimally precise for the given data and representation.
- Combine restraints from multiple data sources immediately with no parameterization.
- Force implicit assumptions to be explicit.

Bayesian Restraints Make Implicit Assumptions Explicit

- Harmonic Restraint

$$s(x, m, k) = \boxed{\frac{1}{2}k(x-m)^2}$$

- Gaussian (Normal) Restraint

$$s(x, \mu, \sigma) = -\ln\left[\frac{1}{\sqrt{2\pi}\sigma}e^{-(x-\mu)^2/2\sigma^2}\right] = \boxed{\frac{1}{2\sigma^2}(x-\mu)^2} + \boxed{\ln(\sigma) + \frac{1}{2}\ln(2\pi)}$$

- Harmonic restraint is (basically) a normal restraint and therefore assumes
  - uncertainty is a known constant
  - error is additive
  - assume equal uncertainty for all harmonics and independence of data points if multiple harmonics with same k used

Hypothetical Scenario

- A collaborator approaches us with NMR data from ubiquitin.
- They've computed 364 Nuclear Overhauser Effects (NOEs) and 98 J-couplings.
- They've used software to predict dihedral angles from J-coupling data.

$$J(\phi) = C\cos(2\phi) + B\cos(\phi) + A$$

- They want to know what states of ubiquitin are consistent with their data.

Data from Cornilescu, G et al. 1998. *J Am Chem Soc*. 120: 6836-7. PDB: 1D3Z

Nuclear Overhauser Effect (NOE)

- An effect seen when magnetic dipoles of two nearby protons interact
- Irradiating one to resonance and then relaxing causes state population of other to change, resulting in change in intensity
- Assume isolated spin-pair approximation (ISPA)

$$I_{NOE}(X_1, X_2, \gamma) = \frac{\gamma}{d(X_1, X_2)^6}, \gamma > 0$$

- Magnitude of NOE alone is meaningless. Only relative magnitude of NOE matters. Reference distances often used to parameterize.

Bayesian Restraint for NOEs

- NOE error is likely multiplicative, not additive
  - Absolute deviation of NOE is meaningless.
  - We therefore use a log-normal distribution

$$p(I_i \mid X_1, X_2, \gamma, \sigma, I) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2\sigma^2} \ln^2\left(I_i/\gamma d_i^{-6}(X_1, X_2)\right)\right\}$$

- We *a priori* don't know the scale of error or scale (gamma) parameter
  - Jeffreys Prior is like a uniform prior with maximal ignorance of scale magnitude.

$$p(\sigma) \propto \frac{1}{\sigma} \qquad p(\gamma) \propto \frac{1}{\gamma}$$

Rieping W, Habeck M, Nilges M. *Science*. 2005. 309(5732): 303-6.
Rieping W, Habeck M, Nilges M. *J Am Chem Soc*. 2005. 127(46): 16026-7.

Example IMP Restraint: Jeffreys Prior

$$p(\sigma) \propto \frac{1}{\sigma} \qquad s(\sigma) = -\ln\left[p(\sigma)\right] = \ln(\sigma) + \cancelto{0}{c}$$

$$\frac{\partial}{\partial \sigma} s(\sigma) = \frac{1}{\sigma}$$

```
class JeffreysRestraint(IMP.Restraint):
    """Jeffreys prior on the sigma parameter of a normal distribution."""
    def __init__(self, m, s):          Init with model and sigma (particle with IMP.isd.Scale)
        IMP.Restraint.__init__(self, m, "JeffreysRestraint%1%")
        self.s = s                                                        Store particle

    def do_add_score_and_derivatives(self, sa):         Take an IMP.ScoreAccumulator
        sig = IMP.isd.Scale(self.get_model(), self.s)
        score = math.log(sig.get_scale())                                 Compute score
        if sa.get_derivative_accumulator():
            deriv = 1. / sig.get_scale()           Compute derivative of score wrt sigma
            sig.add_to_scale_derivative(deriv, sa.get_derivative_accumulator())
        sa.add_score(score)                                   Add score to accumulator

    def do_get_inputs(self):                   Get particles used in score calculation
        return [self.get_model().get_particle(self.s)]
```

Activity: Write an IMP Restraint for NOE

- Open `imp_restraints.py`
- Take model, two **`IMP.core.XYZR`** particles (atoms) and two **`IMP.isd.Scale`** particles (sigma and gamma) as input.

$$p(I_i \mid X_1, X_2, \gamma, \sigma, I) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2\sigma^2} \ln^2\left(I_i / \gamma d_i^{-6}(X_1, X_2)\right)\right\}$$

```
class NOERestraint(IMP.Restraint):
    """Apply an NOE distance restraint between two particles."""
    def __init__(self, m, p0, p1, sigma, gamma, Iexp):
        IMP.Restraint.__init__(self, m, "NOERestraint%1%")
        pass                                                    Store variables

    def do_add_score_and_derivatives(self, sa):
        pass                                    Compute score and derivatives

    def do_get_inputs(self):
        pass                                                    Return all inputs
```

NOE Restraint Formulas

- Score

$$s(d, \sigma, \gamma, I_i) = \frac{1}{2\sigma^2} \ln^2 \left(\frac{I_i d^6}{\gamma} \right) + \ln(d\sigma) + \frac{1}{2} \ln(2\pi)$$

- Derivatives (if you have time)

$$\frac{\partial}{\partial \sigma} s(d, \sigma, \gamma, I_i) = \frac{1}{\sigma} + \frac{1}{\sigma^3} \ln^2 \left(\frac{I_i d^6}{\gamma} \right)$$

$$\frac{\partial}{\partial \gamma} s(d, \sigma, \gamma, I_i) = -\frac{6}{d\gamma\sigma^2}$$

$$\frac{\partial}{\partial d} s(d, \sigma, \gamma, I_i) = \frac{6}{d\sigma^2} \ln \left(\frac{I_i d^6}{\gamma} \right)$$

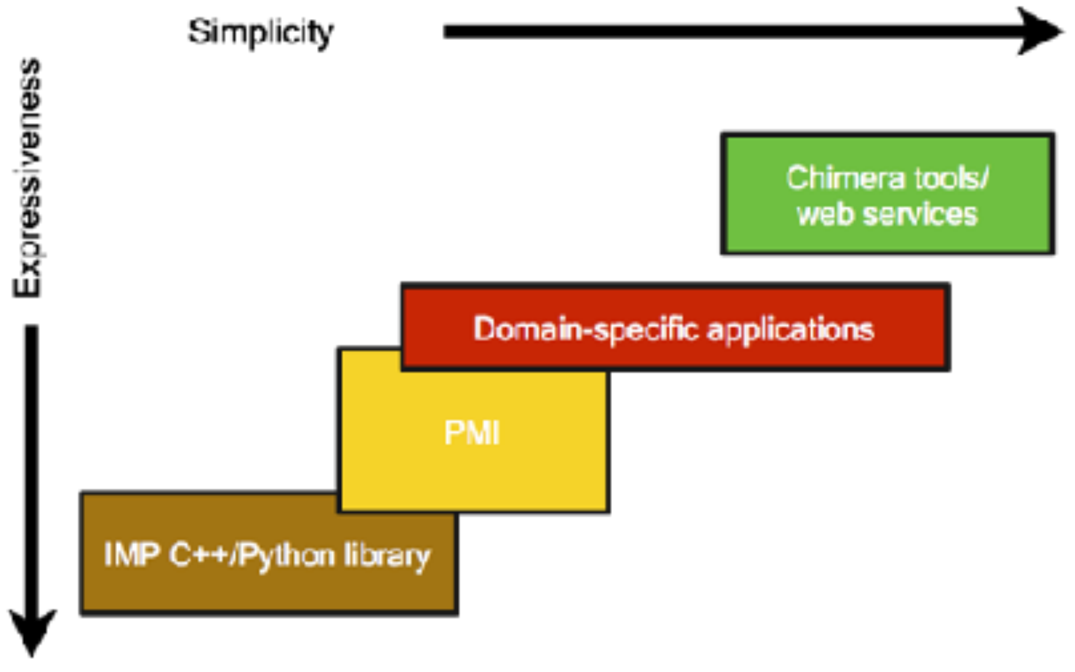
Tricky because you need to propagate to points

How `NOERestraint` is Implemented in IMP

- https://integrativemodeling.org/2.6.2/doc/ref/classIMP_1_1isd_1_1NOERestraint.html
- https://github.com/salilab/imp/blob/develop/modules/isd/src/NOERestraint.cpp

PMI vs IMP

- IMP is comprised of low-level, general components: Particles, geometries, restraints, optimizers, etc
- PMI is a collection of high-level wrappers:
  - 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



PMI vs IMP Restraints

- 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

Example: Wrapping Our Restraints in PMI

- Open `pmi_restraints.py`

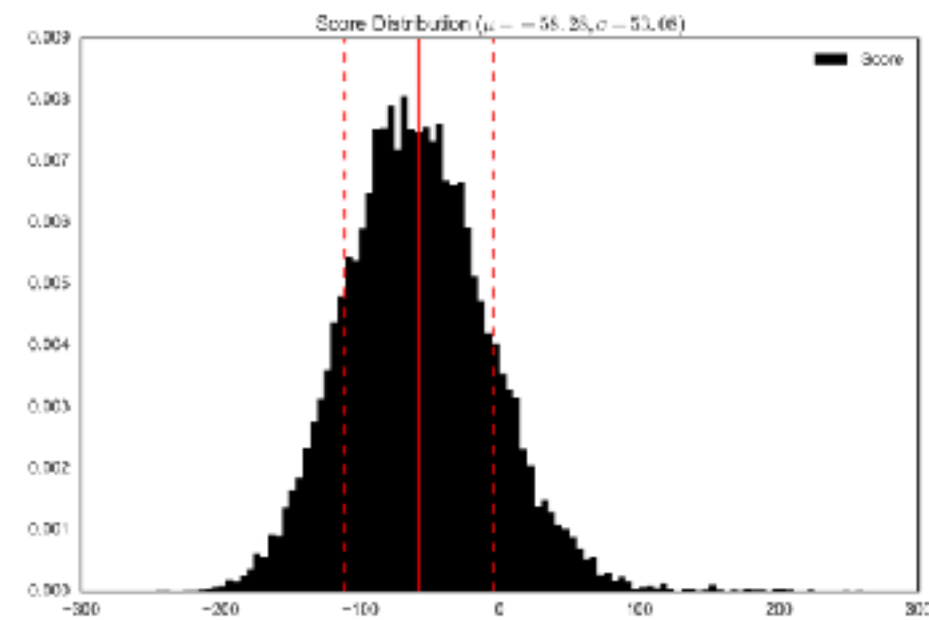
Coming Soon

- All restraints in **`IMP.pmi`** will soon share a common base class with add-ons.
- Most of the functionality needed to make a restraint PMI compatible will be automatically provided.
- Usually only **`__init__`** will need to be defined.

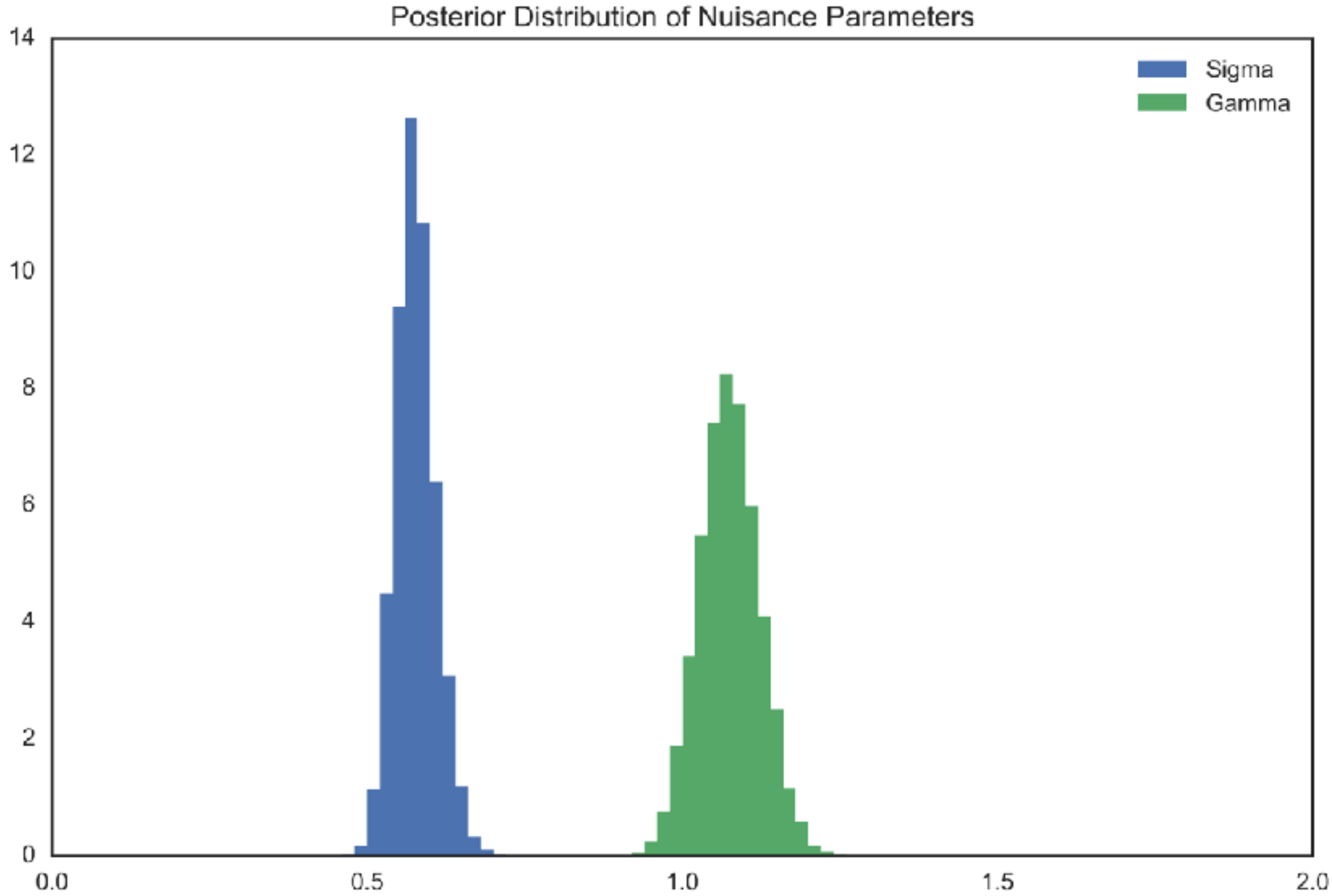
Running the Simulation

- Time to run the script.
  ```
  $ python run.py
  ```
- Because this uses replica exchange, for the full effect, run with an MPI-enabled IMP on a cluster.
  ```
  $ mpirun -np 8 python run.py
  ```
- When finished, run
  ```
  $ python cluster.py
  $ python plot_progress output/
  stat.0.out
  ```

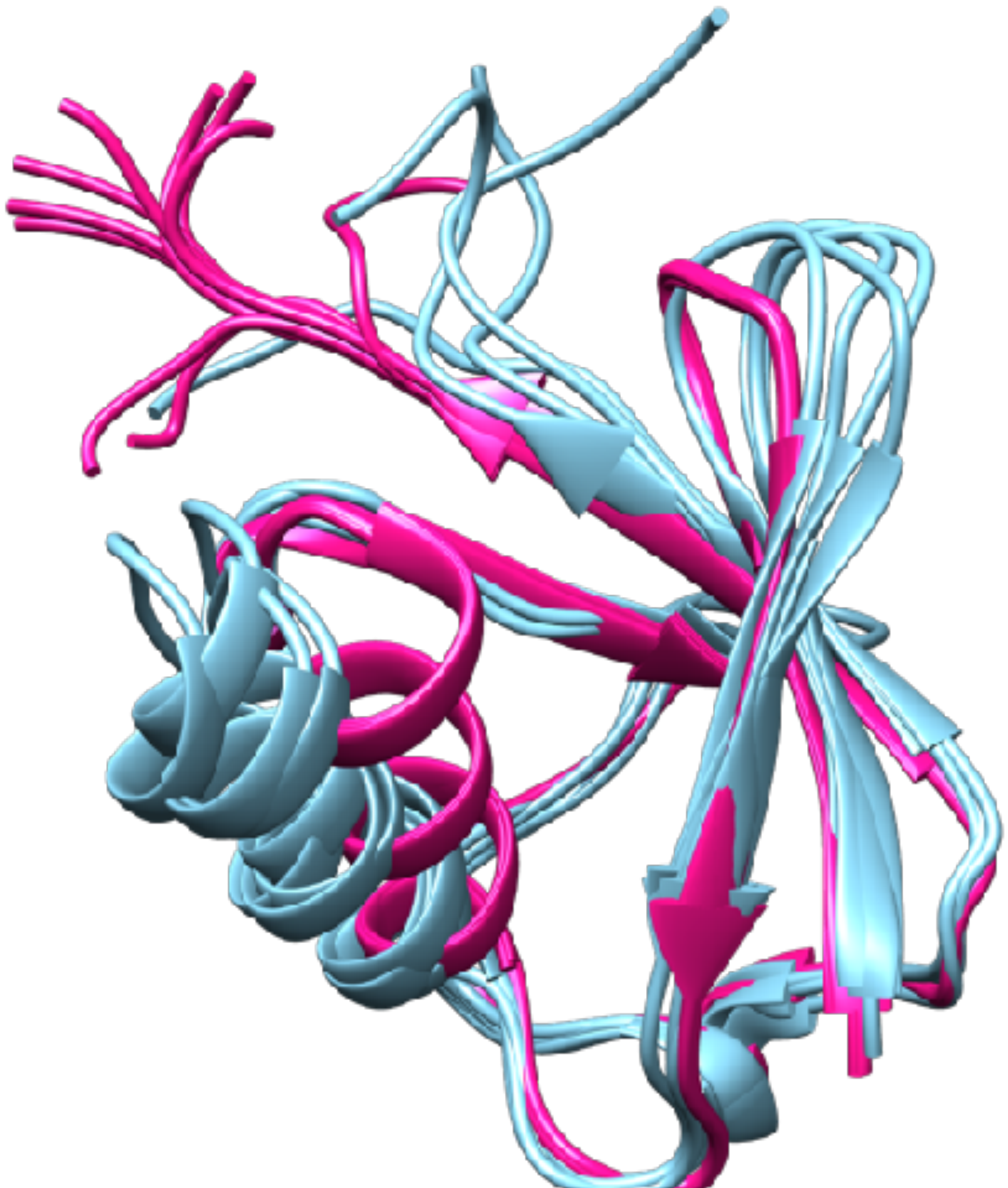
Scores Plateau Over Time



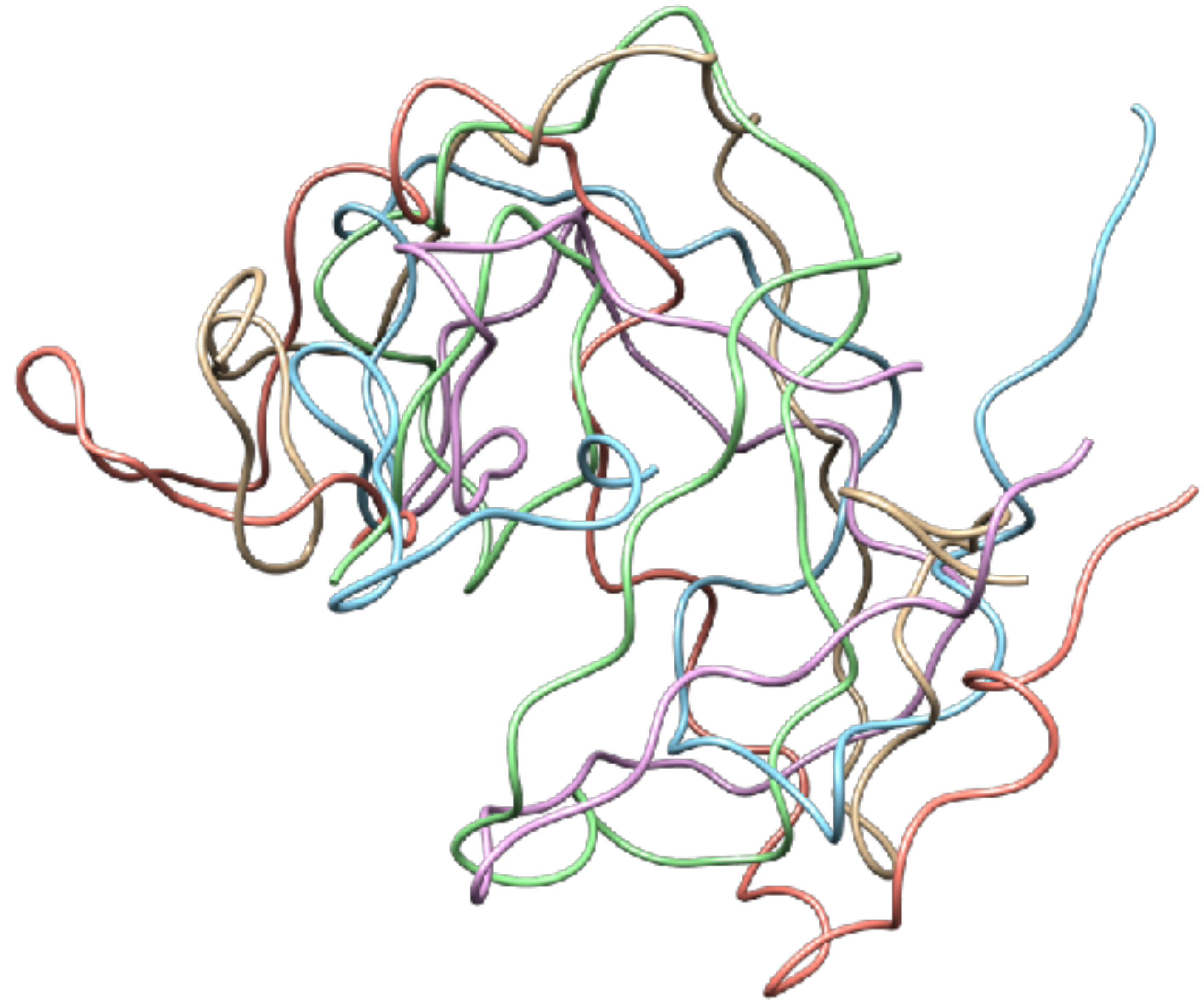
The Posterior Distribution of the Nuisance Parameters



Top Scoring Cluster Representatives



with all restraints
compared to original paper



with no data restraints

Data from Cornilescu, G et al. 1998. *J Am Chem Soc*. 120: 6836-7. PDB: 1D3Z

Options for data integration

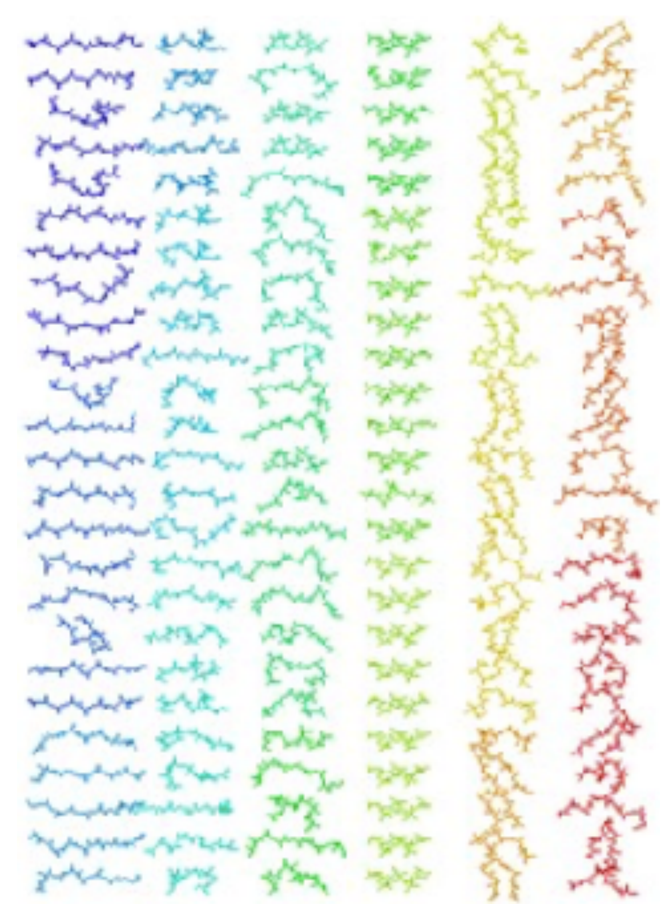
3. **Sampling**
   Use sampling procedures that pick data-compatible models

Example: fragment libraries for efficient backbone sampling in Rosetta

sequence

secondary structure prediction

NMR chemical shifts

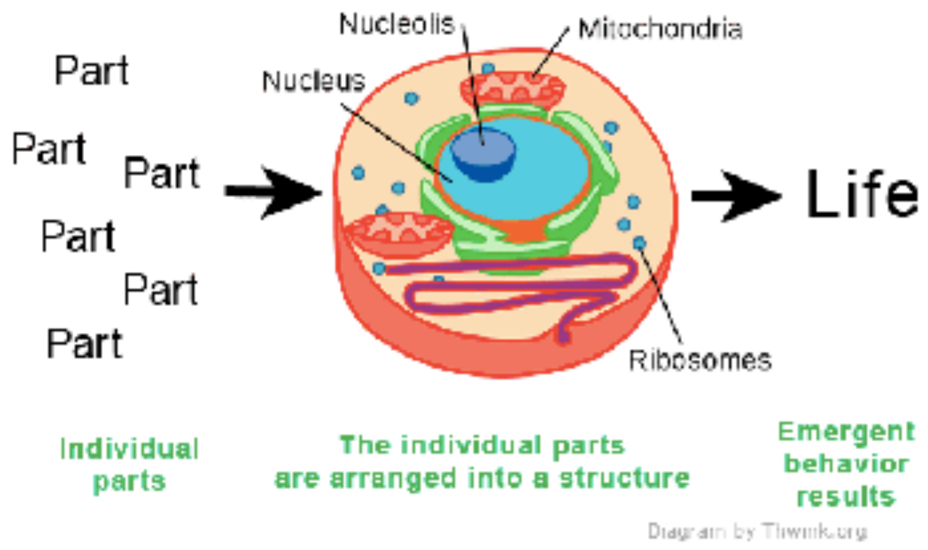


Options for data integration

4. **Filtering**
   Reject inconsistent models

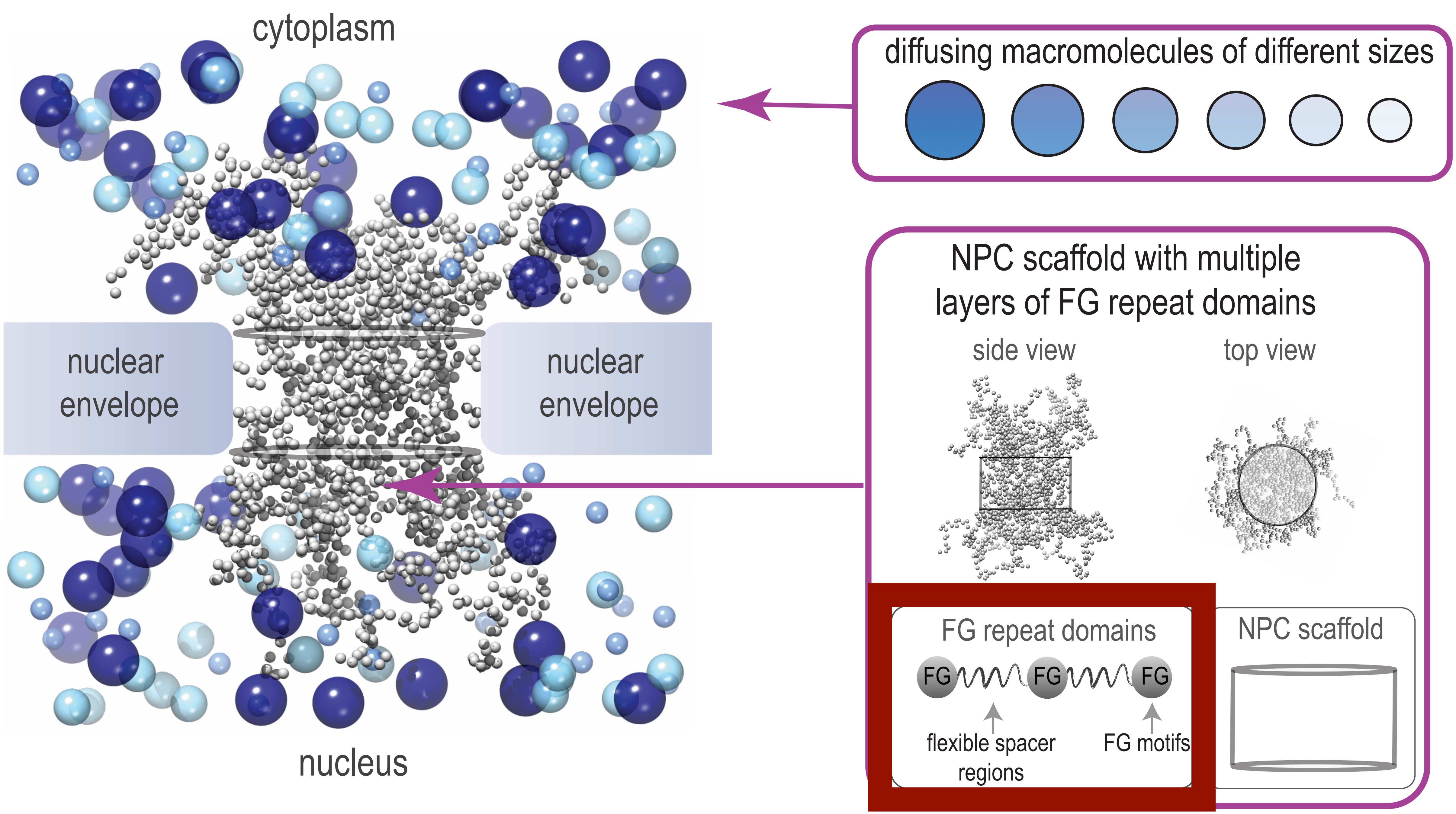
Useful for emergent properties, for which data restraints are less suitable
- when the whole is greater than the sum of its parts

Examples:
- radius of gyration
- # buried hydrogen bonds
- life



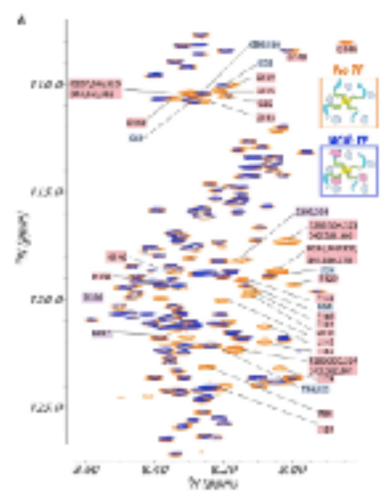
Integrating data about the radius-of-gyration (R_g) of disordered FG repeats

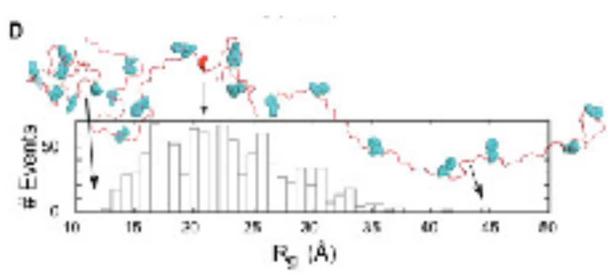
(representation, scoring, filtering)



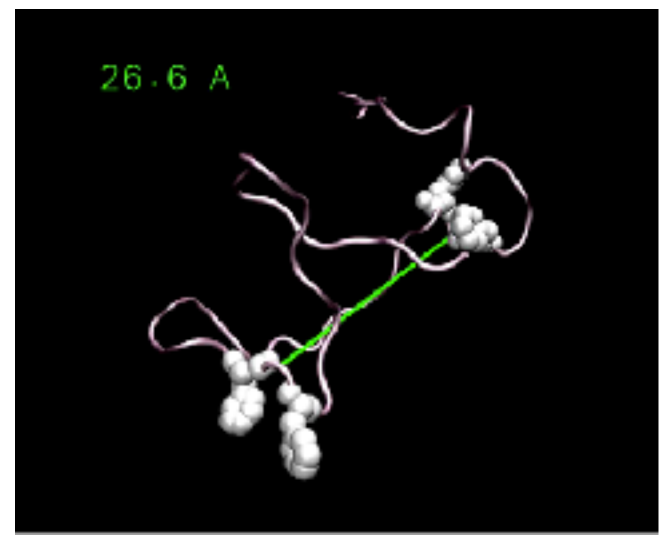
Multiple sources indicate that FG repeats are highly disordered; wide distribution of R_g values

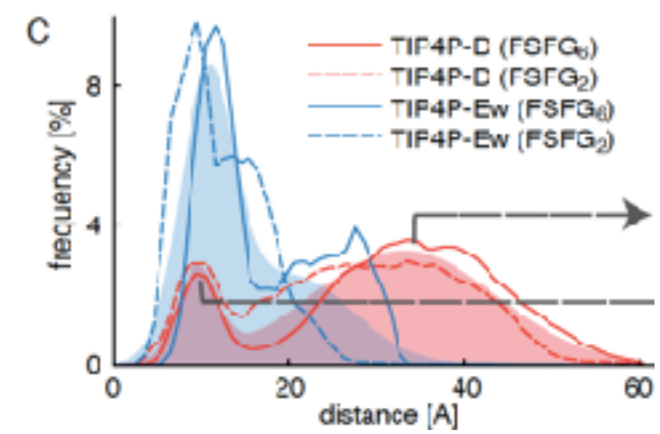
NMR
(Hough et al., 2015;
Milles et al., 2015;
Raveh, Karp, Sparks et al., 2016)



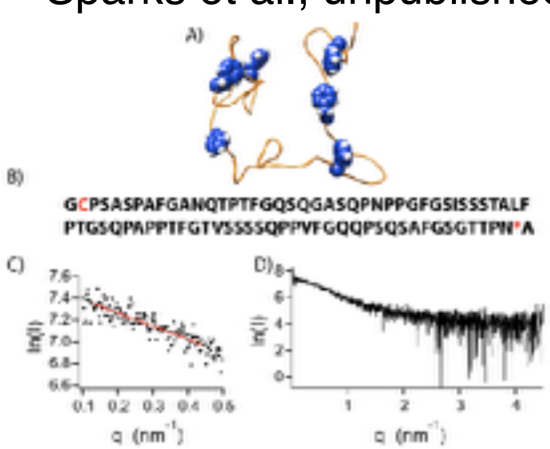


Full atom MD simulations (Raveh, Karp, Sparks et al, 2016; Mercadante et al. 2015)

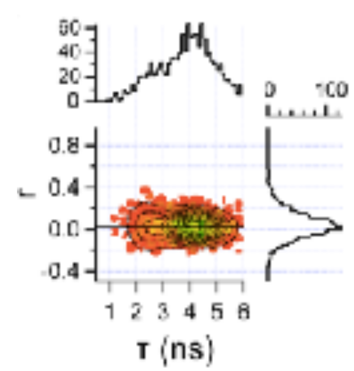




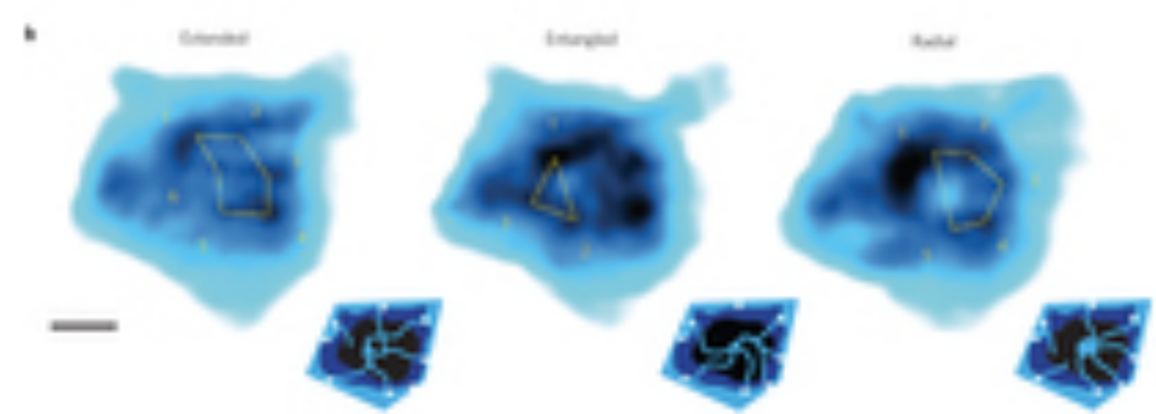
SAXS and SANS
(Mercadante et al., 2015;
Sparks et al., unpublished data)



smFRET
(Mercadante et al., 2015)



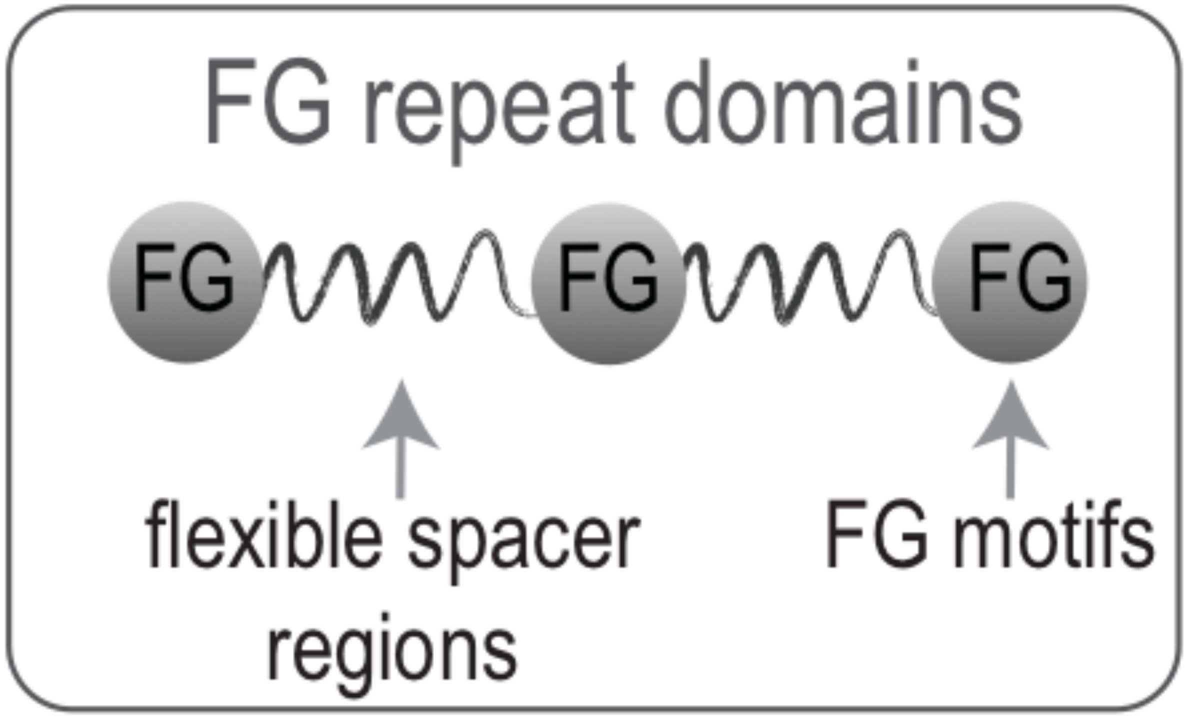
Atomic force microscopy (AFM)
(Sakiyama et al., 2016)



Objective: a coarse-grained model of FG repeats that reproduces observed $<R_g^2>$

I. Representation:

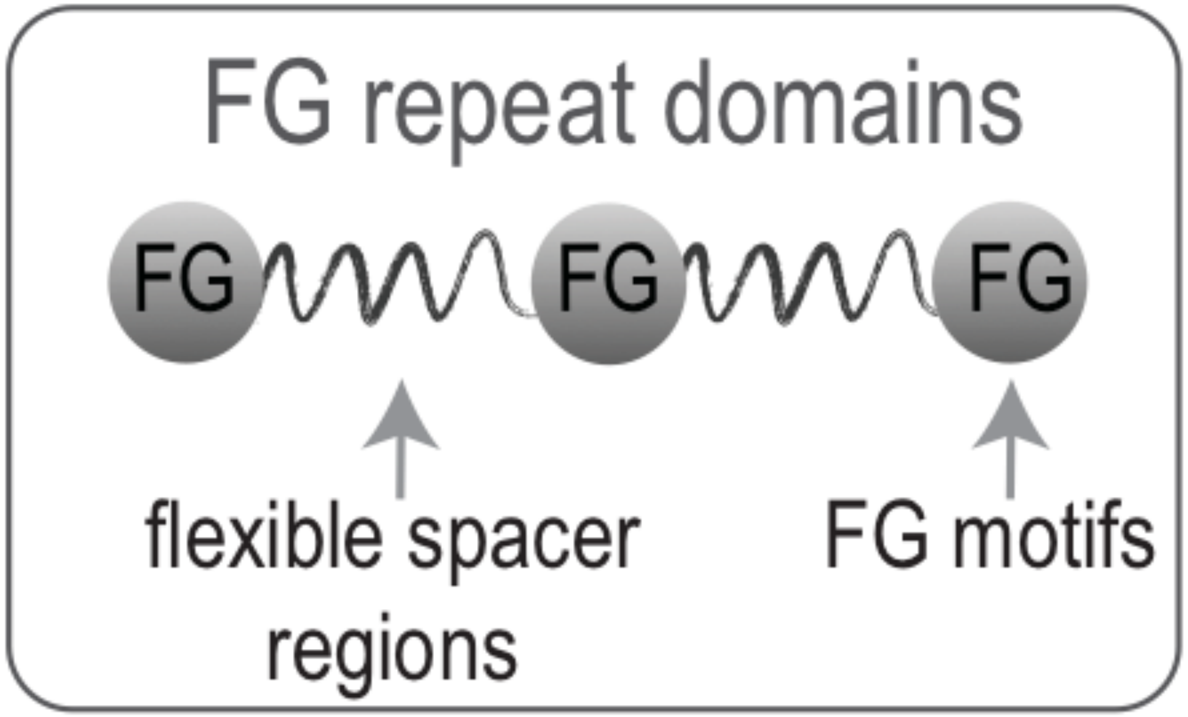
- Bead = ~20 amino acids
  (data: one repeat = ~20 amino acids)
- Springs with unknown spring parameter
  (data: polymer behaves as entropic-spring)



Objective: a coarse-grained model of FG repeats that reproduces observed $<R_g^2>$

II. Scoring:

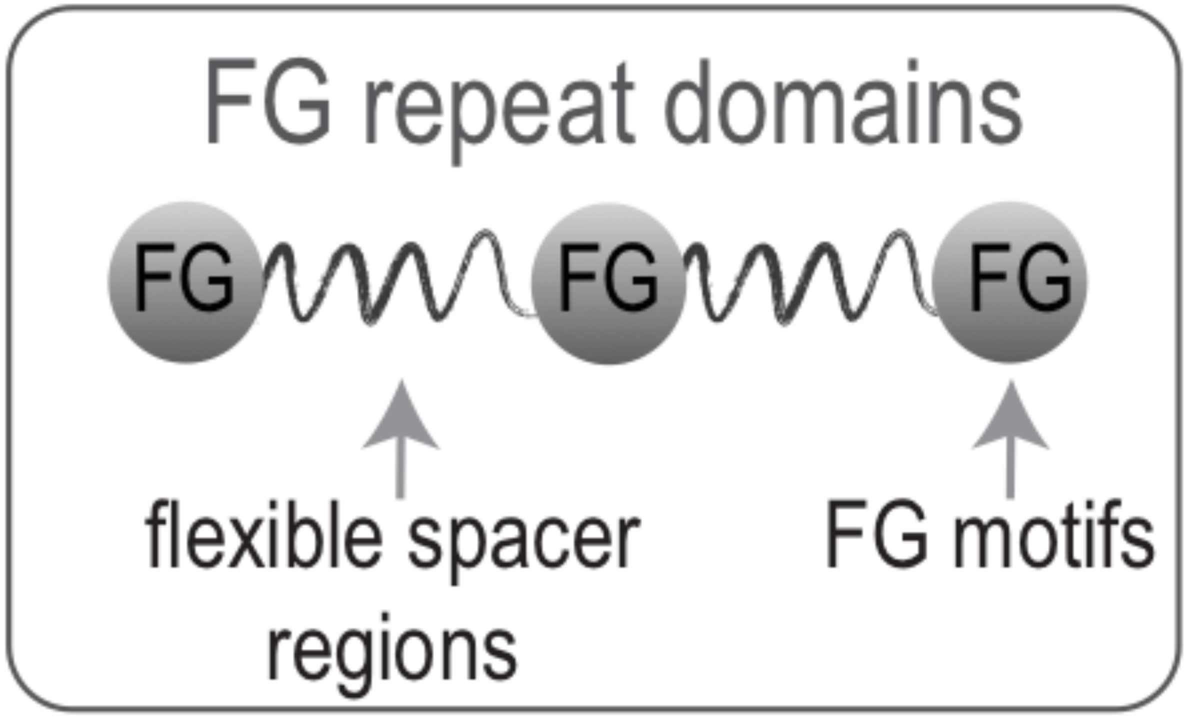
- Use data and polymer physics theory for initial guess if:
  - Spring resting length (bond length)
  - Spring force coefficient (in kcal/mol/A2)



Objective: a coarse-grained model of FG repeats that reproduces observed <R_g^2>

III. Filtering:

- Simulate model with different parameters
- Characterize subset of model parameters that reproduce observed <R_g^2>



Hands on:

- Represent FG repeats in IMP
- Simulate with different model parameters
- Filter out models that are inconsistent with the data

Hands on: Simulate Rg with different parameters

- `python simulate_radius_of_gyration <k> <rest_length_factor>`
  - try with k=0.01 vs. 10.0 kcal/mol/A^2
  - try with rest_length_factor=1.0 vs. 1.5 vs. 2.0
- What's the effect on mean Rg? Fluctuations?

Hands on:
Creating the custom scoring function

```
# ==
# II. Create scoring fucntion with chain bonds and excluded volume restraints
#     on chain particles
# ==
# bonded:
bond_score= IMP.core.HarmonicDistancePairScore(bond_rest_length,
                                               bond_k,
                                               "bond score")
bonded_pairs= \
    IMP.container.ExclusiveConsecutivePairContainer(m, P)
bond_restraint= \
    IMP.container.PairsRestraint(bond_score, bonded_pairs, "bond restraint")
# excluded volme:
excluded_score= IMP.core.SoftSpherePairScore(excluded_k)
close_pairs= IMP.container.ClosePairContainer(P, 0, slack_A) # container that
close_pairs.add_pair_filter( IMP.container.ExclusiveConsecutivePairFilter() )
excluded_restraint= IMP.container.PairsRestraint(excluded_score, close_pairs,
# final scoring function:
scoring_function= IMP.core.RestraintsScoringFunction \
    ([bond_restraint, excluded_restraint])
```

Summary: options for data integration

1. **Representation**
   Represent the system parts and their interactions to reflect our prior knowledge
2. **Scoring**
   Score different models based on data fit
3. **Sampling**
   Use sampling procedures that pick data-compatible models
4. **Filtering**
   Reject inconsistent models
5. **Validation and Testing**
   Contrast final models with data

Before Writing New IMP Code

- What exactly do you need?
  - New function, restraint, class(es)?
- Check IMP for existing code
  - Many abstractions, such as **`PairScore`** and **`UnaryFunction`** can be passed to existing restraints.
  - Can existing code be enhanced to provide the necessary functionality?
- Does the restraint belong in a general module (e.g. **`IMP.core`**) or does it need its own module?

Creating a Module for a New Experimental Restraint

- First, clone the IMP git repo and enter it.
- From the `imp` directory, run:

```
$ python tools/make-module.py
module_name
```

- Enter new module.
- Update `dependencies.py` to contain necessary dependencies.
- Update `README.md`.

IMP Module Directory Structure

..	
bin	Stand-alone programs
data	
examples	Examples for using the module code
include	C++ header files
pyext	Python code and C++/Python interfaces (SWIG)
src	C++ source files
test	Tests for Python and C++ code
utility	
.imp_info.py	
README.md	
dependencies.py	IMP module/library dependencies

Tests, Documentation, and Examples

- Rigorous testing ensures code functions correctly
  - Use **`IMP.test.TestCase`**, based on Python's **`unittest.TestCase`**.
  - Test all methods, initializations, use cases, including edge scenarios.
  - Test setup of larger systems/simulations ("medium" and "expensive" tests).
- Documentation enables others to use your code.
  - Thoroughly document class inputs (Doxygen format) and any unclear steps.
- Write interesting examples showing others how to use your code.
- See https://integrativemodeling.org/doc.html and https://github.com/salilab/imp for examples