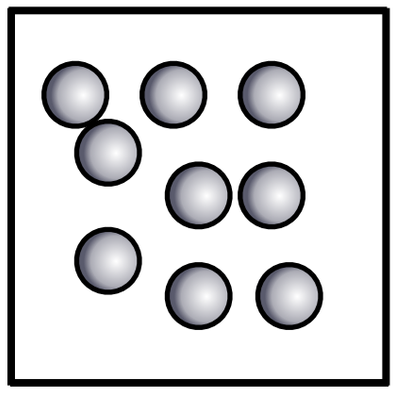
Recap

- MD integrators work based on taking small timesteps and solving for positions, velocities at each step, based on the forces
- Velocity Verlet and related integrators are good, stable choices that yield correct distributions

Often, we are interested in behavior of “large” systems or bulk phases; this presents challenges

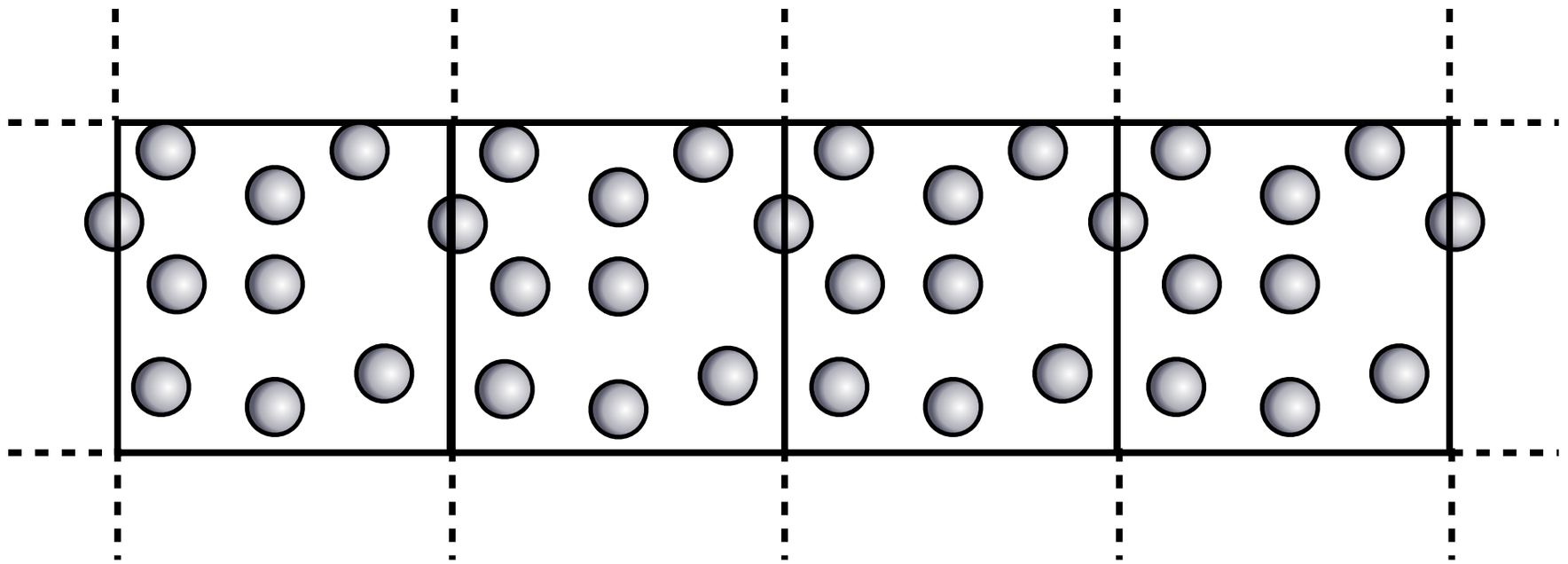
- Most experiments are done on at least ~10^{23} particles
- There is no way we can simulate more than 10^{3}-10^{6} particles for even very short times
- So, we are forced to consider a smaller system that will hopefully be representative of the “large” or bulk system of interest

But we have to be careful how we set up these smaller systems



- Small systems may have large finite size effects
  - Edge effects: Particles may avoid edges because they have no neighbors there
  - We're simulating a nanoscale system rather than bulk
- We would like to avoid simulating systems that behave "small"

Periodic Boundary Conditions (PBC) provide one way to minimize edge effects



- Tiling space with an infinite number of copies of a small system helps remove edge effects
- Copies are periodic
  - Contents can cross box edges, at which point they enter the original box from the opposite side

The only relevant coordinates are those of the central box; this results in the minimum image convention

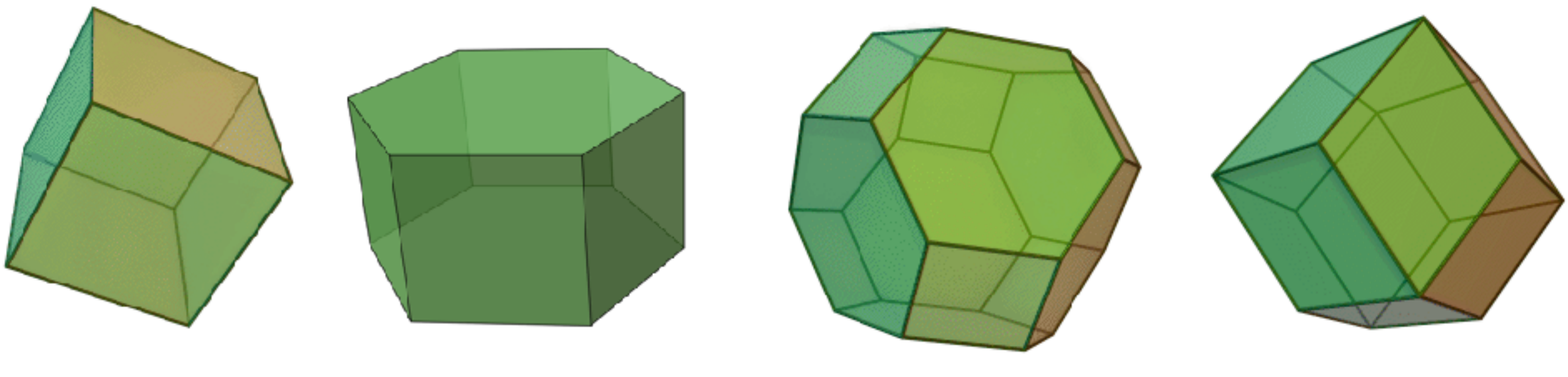
- Particles in the other copies of box are called image particles
- Every image can be mapped to a particle in the central box

$$\mathbf{r^c} = \mathbf{r} - L\ \mathrm{nint}(\mathbf{r}/L)$$ where nint is the nearest integer

- Often we need the distance between two specific particles; we use the shortest such distance (“minimum image convention”)

$$\mathbf{r}_{ij} = \mathbf{r}_j - \mathbf{r}_i \qquad \mathbf{r}^c_{ij} = \mathbf{r}_{ij} - L\ \mathrm{nint}(\mathbf{r}_{ij}/L)$$

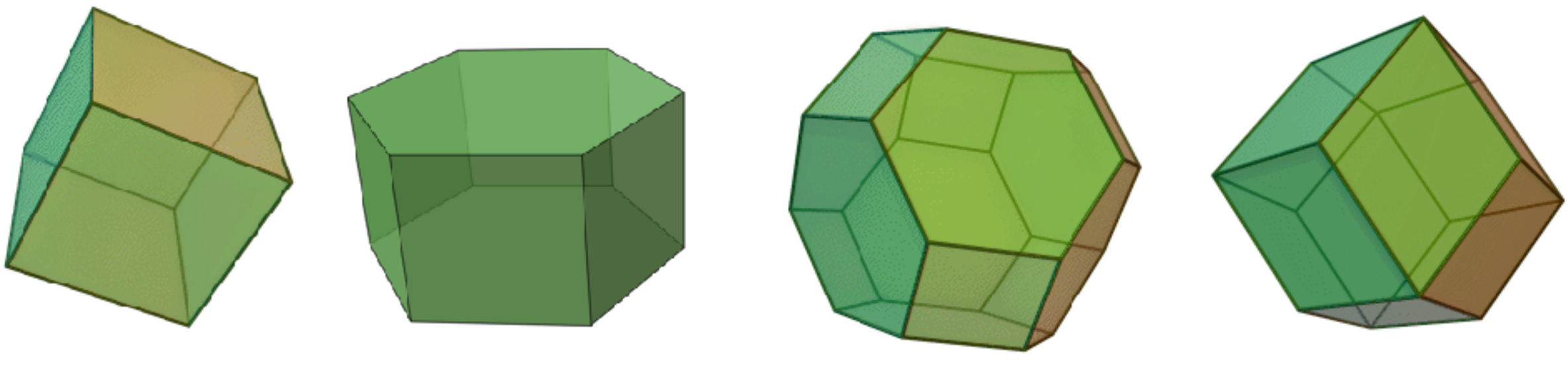
Various box geometries are used; any box geometry is OK as long as it is space-filling



Cubic | Hexagonal prism | Truncated octahedron | Rhombic dodecahedron

- Any geometry that will tile all of space works
- Which to pick?
  - Particular symmetry: i.e. hexagonal if expect hexagonal ordering
  - Save on number of atoms -- i.e. spherical protein in dodecahedral vs. cubic box

Various box geometries are used; any box geometry is OK as long as it is space-filling



Cubic | Hexagonal prism | Truncated octahedron | Rhombic dodecahedron

- Any geometry that will tile all of space works
- Which to pick?
  - Particular symmetry: i.e. hexagonal if expect hexagonal ordering
  - Save on number of atoms -- i.e. spherical protein in dodecahedral vs. cubic box

Parts of the potential are truncated to minimize the number of interactions that must be computed with minimal loss in accuracy

- We definitely shouldn't compute interactions of a particle with its images
  - Also, probably not with images of other particles that are far away
  - At least we should only consider each particle copy once
- Many interaction types decay very rapidly with distance
  - Save some interaction calculations by ignoring distant particles, i.e. for LJ:

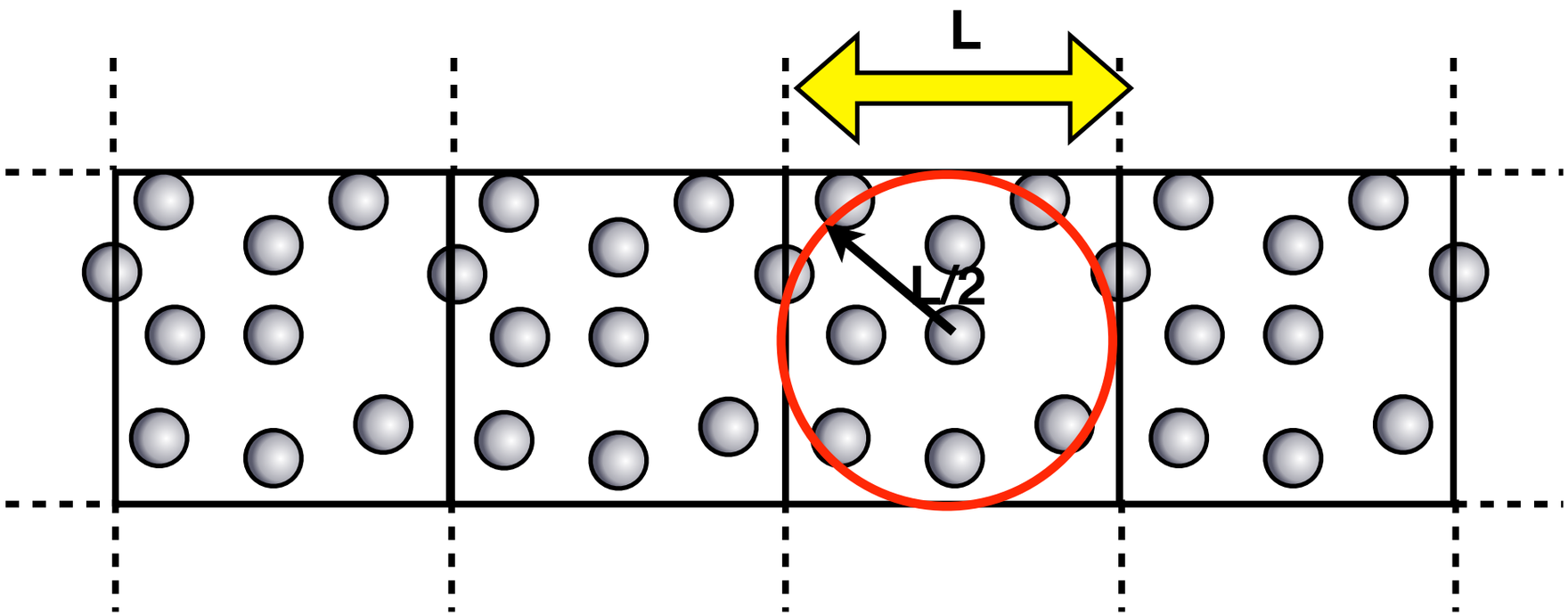
$$u(r) = 4\epsilon\left[\left(\frac{r}{\sigma}\right)^{-12} - \left(\frac{r}{\sigma}\right)^{-6}\right]$$

Instead, 'cut off' the potential:

$$u(r) = \begin{cases} 4\epsilon\left[\left(\frac{r}{\sigma}\right)^{-12} - \left(\frac{r}{\sigma}\right)^{-6}\right] & r \le r_c \\ 0 & r > r_c \end{cases}$$

if r_c is 2.5σ, energy is -0.016ϵ

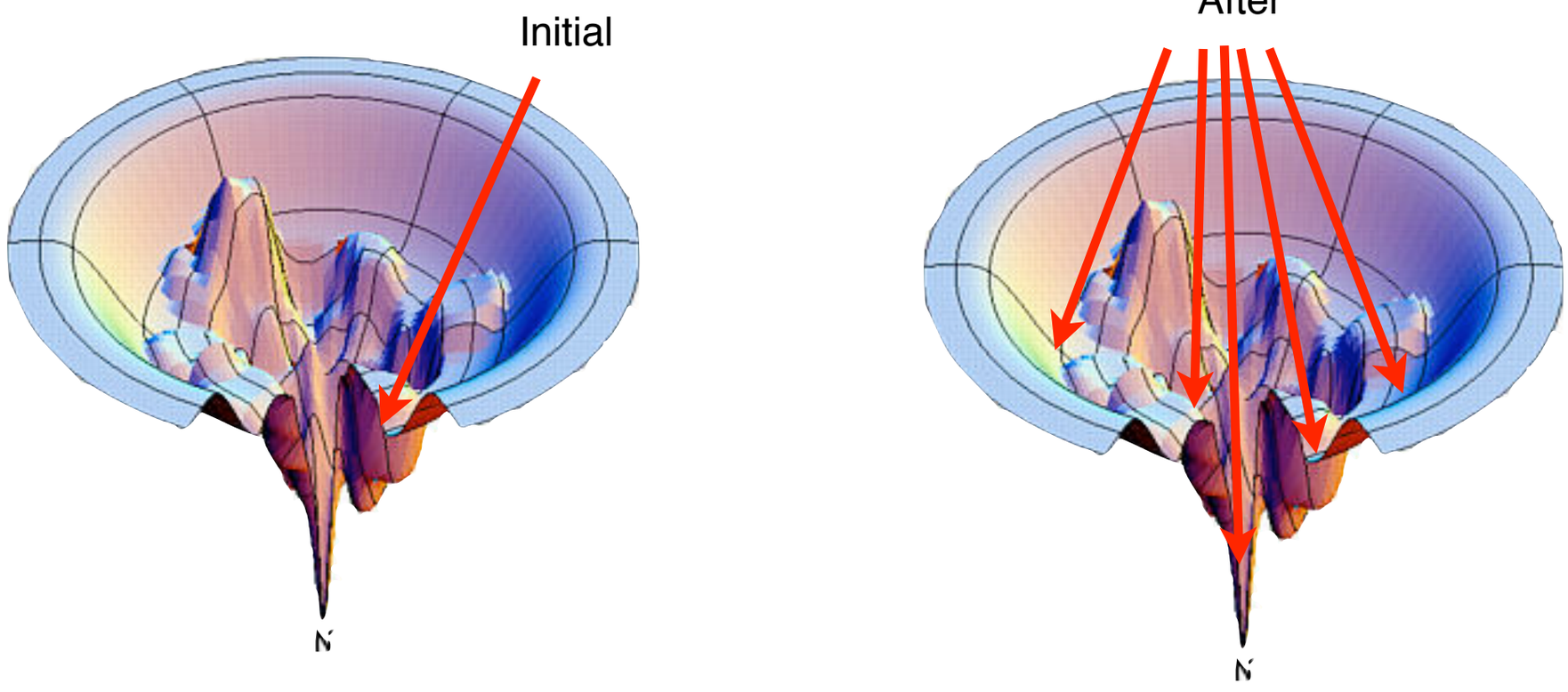
Avoiding interactions with multiple images of the same particle dictates box sizes



- If L/2 is smaller than r_c, particles will begin to interact with multiple images of neighboring particles
  - So, box sizes must be set at least twice as large as cutoff distance
  - Usually larger
- For a desired density, this sets minimum particle #

For simulation results to be trustworthy, we require equilibration and convergence

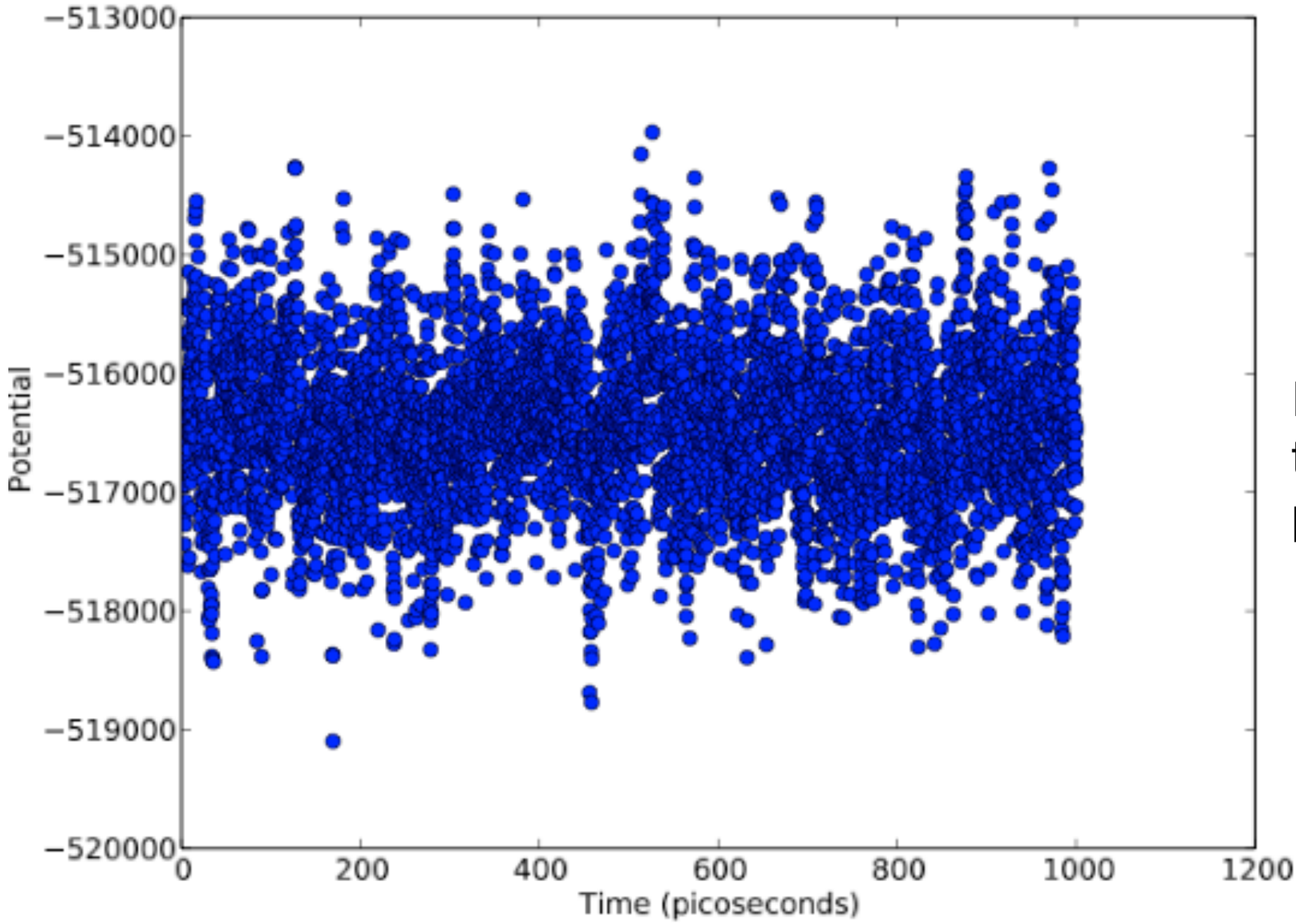
- Equilibration -- reaching thermodynamic equilibrium
  - “Forgetting about” initial conditions
  - Reaching correct distribution of populations across energy landscape
  - Reaching a “steady state” (in an average sense)



For simulation results to be trustworthy, we require equilibration and convergence

- Convergence: Obtaining the final value for the property we are computing
  - Collecting sufficient data to evaluate a good average
  - Averaging over a sufficient amount of simulation time
  - Adequately sampling enough motion of the system

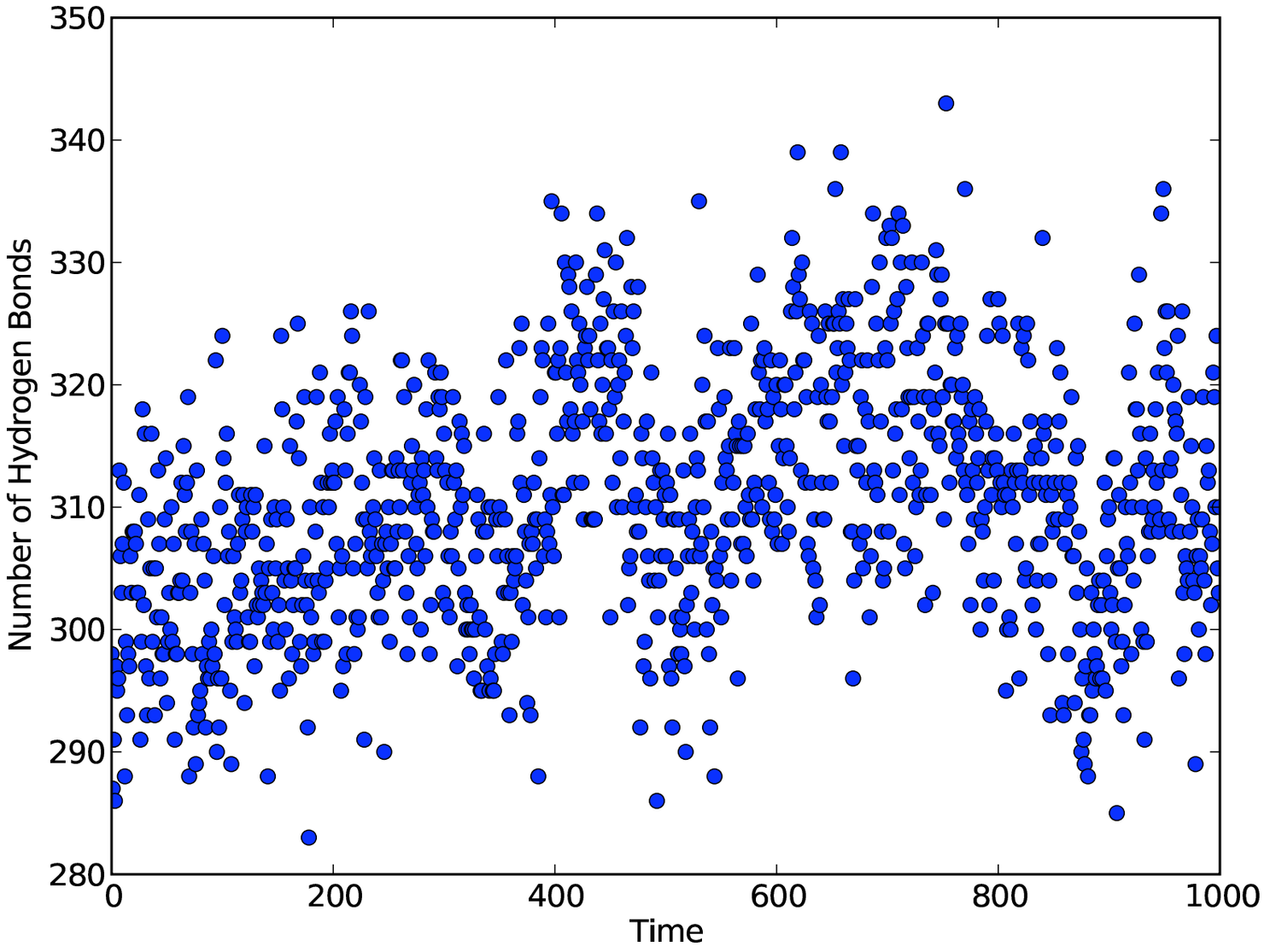
It is difficult to prove we have reached equilibrium, but it is often obvious we have not



Potential energy versus time for a system **possibly at equilibrium**

If a system is at equilibrium, we may see fluctuations in a particular observable (i.e. potential energy) but there should be no trend

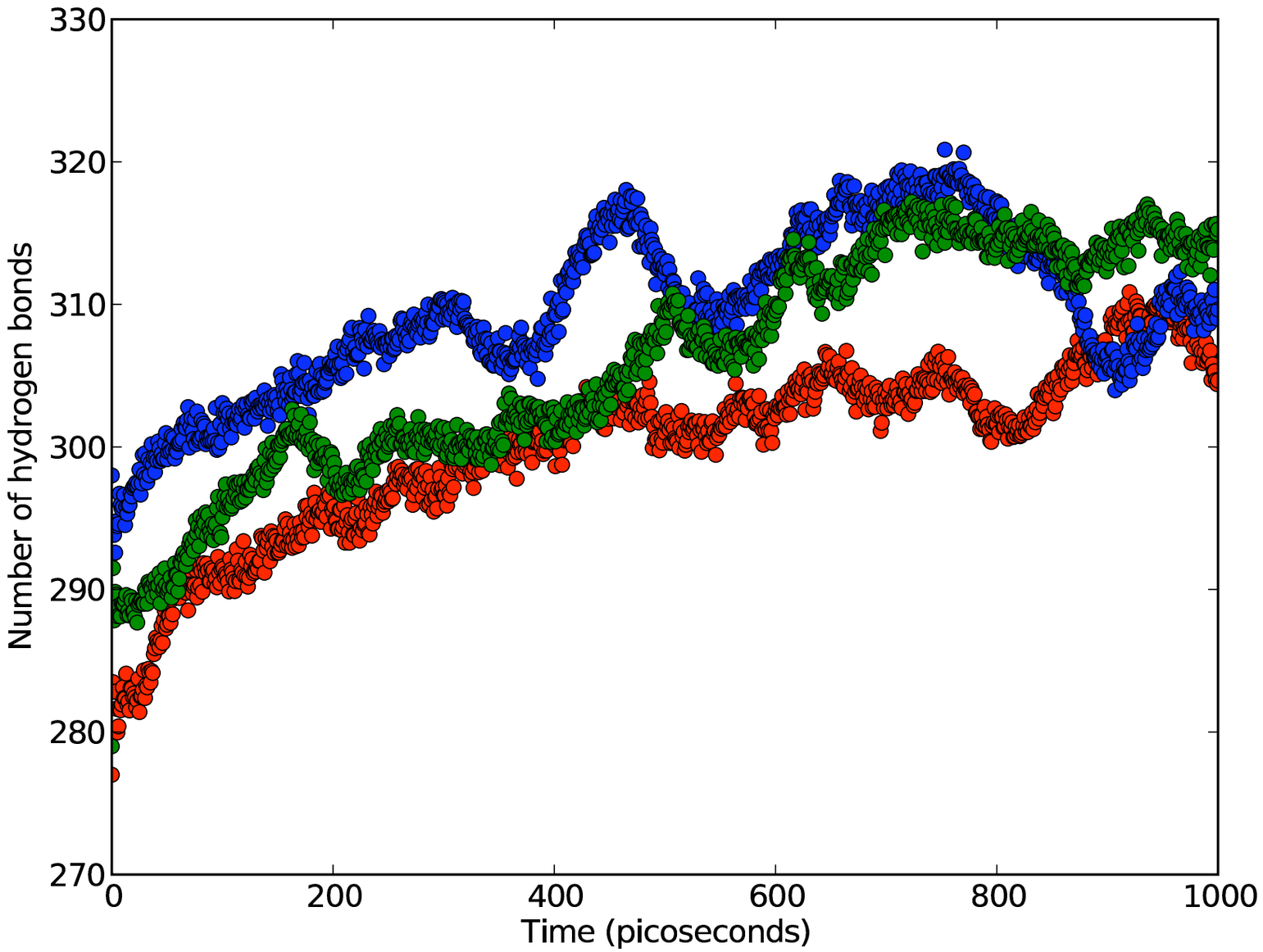
It is difficult to prove we have reached equilibrium, but it is often obvious we have not



Hydrogen bonds versus time for a system **probably not at equilibrium**

If a system is at equilibrium, we may see fluctuations in a particular observable (i.e. potential energy) but there should be no trend

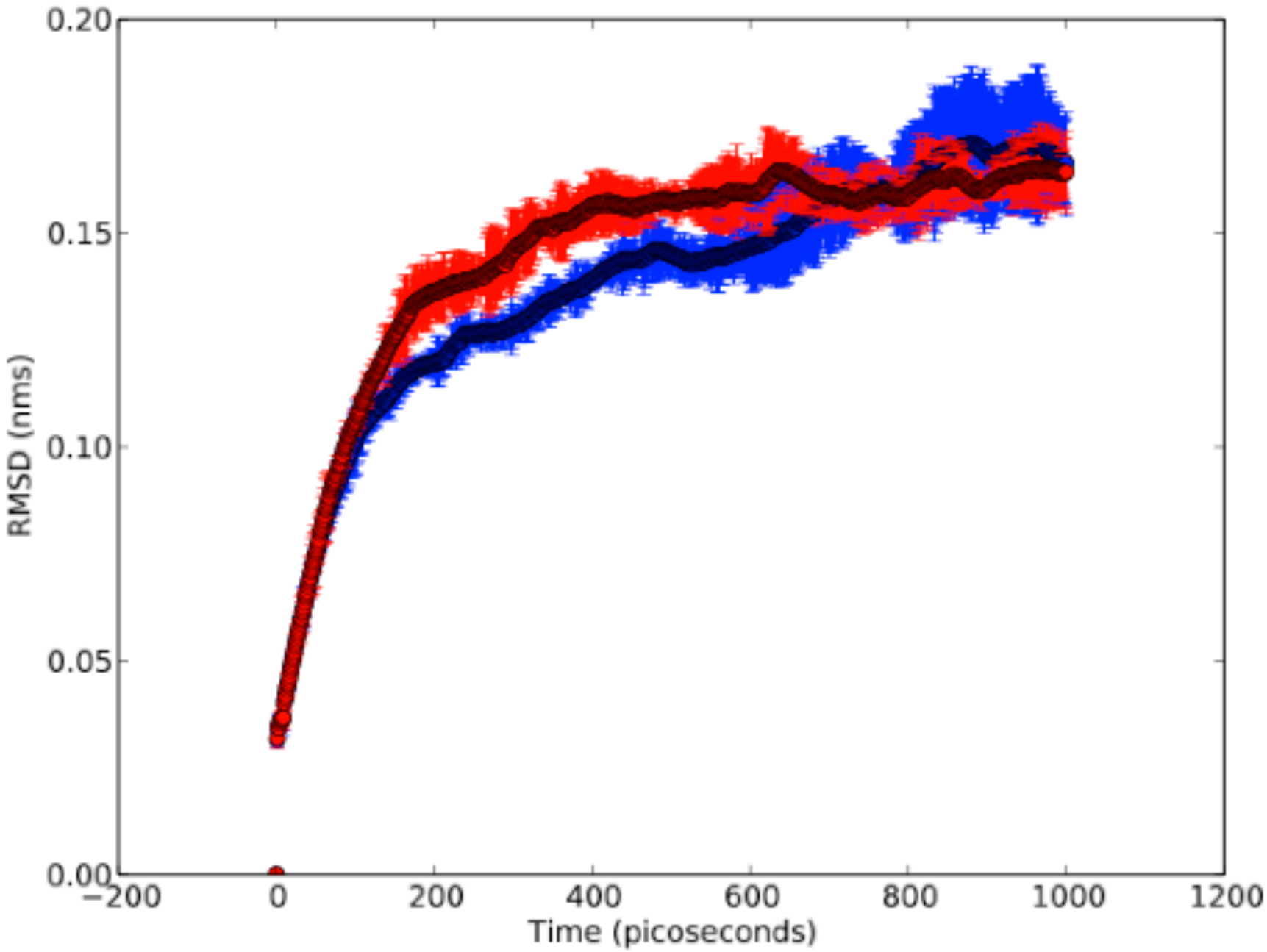
It is difficult to prove we have reached equilibrium, but it is often obvious we have not



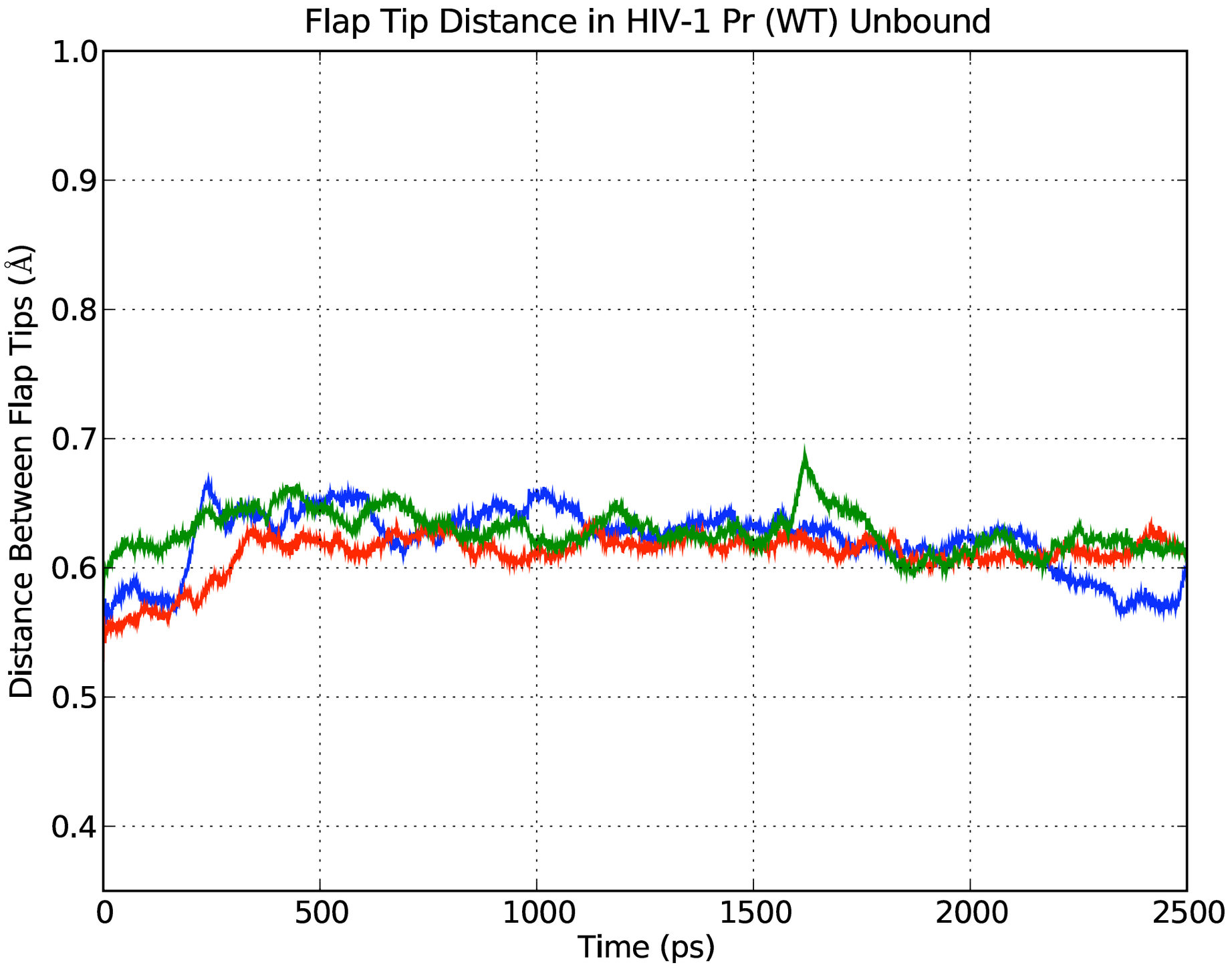
Hydrogen bonds versus time for a system **probably not at equilibrium**
-- running averages and multiple trials make this more clear

If a system is at equilibrium, we may see fluctuations in a particular observable (i.e. potential energy) but there should be no trend

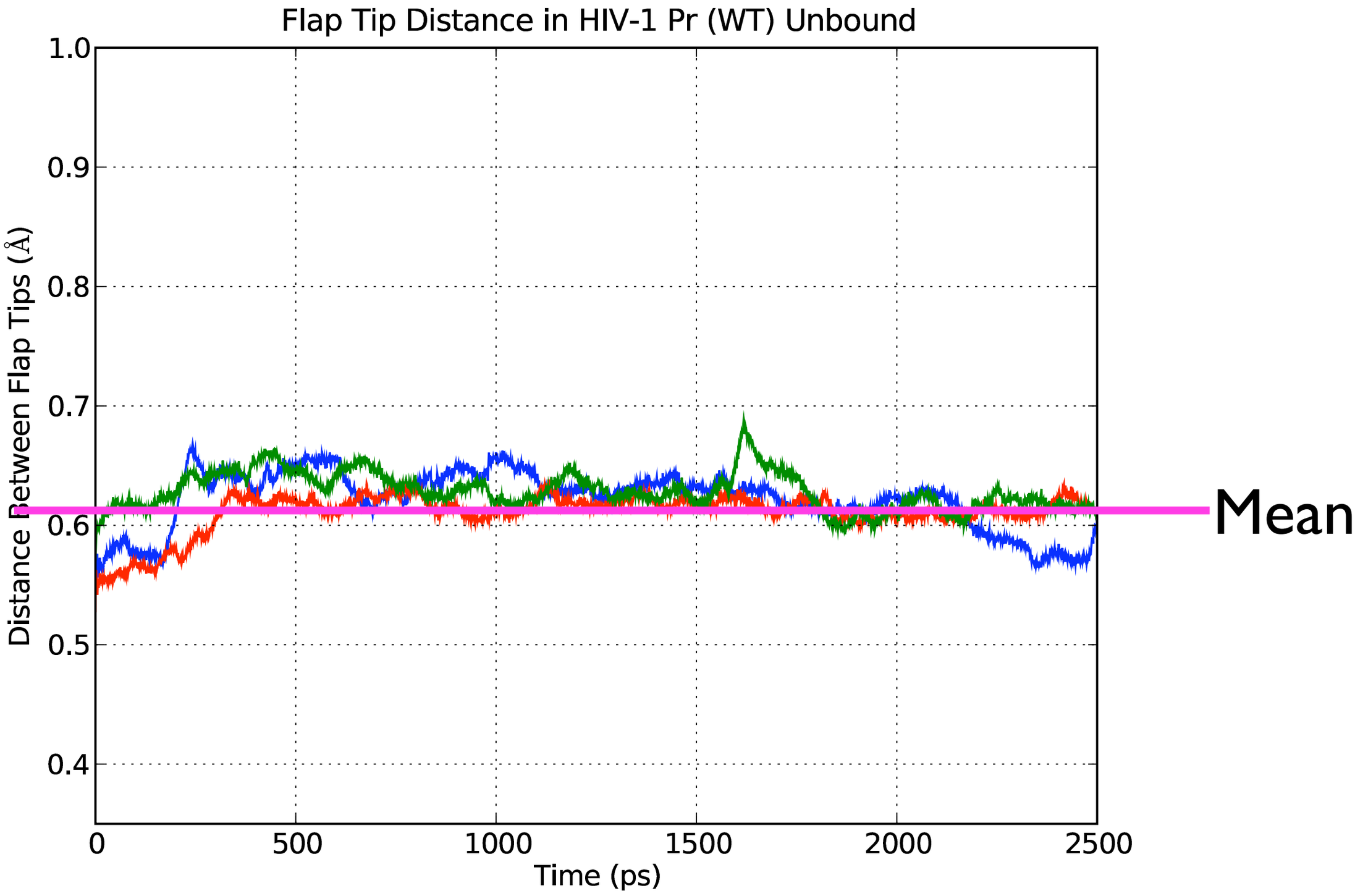
Another example: Maybe at equilibrium



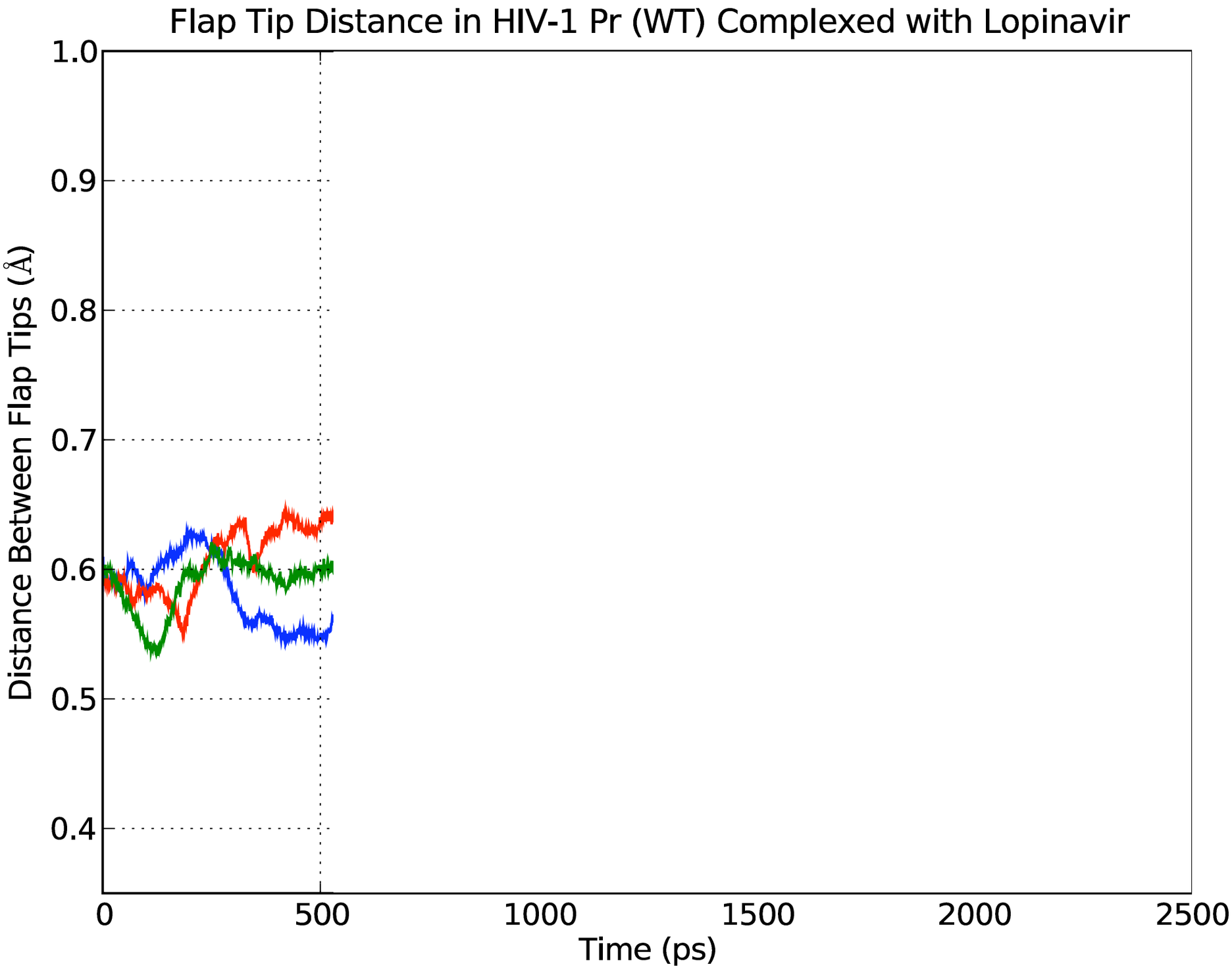
Convergence means obtaining correct final average for property we are computing. Example: HIV protease flap tip distance



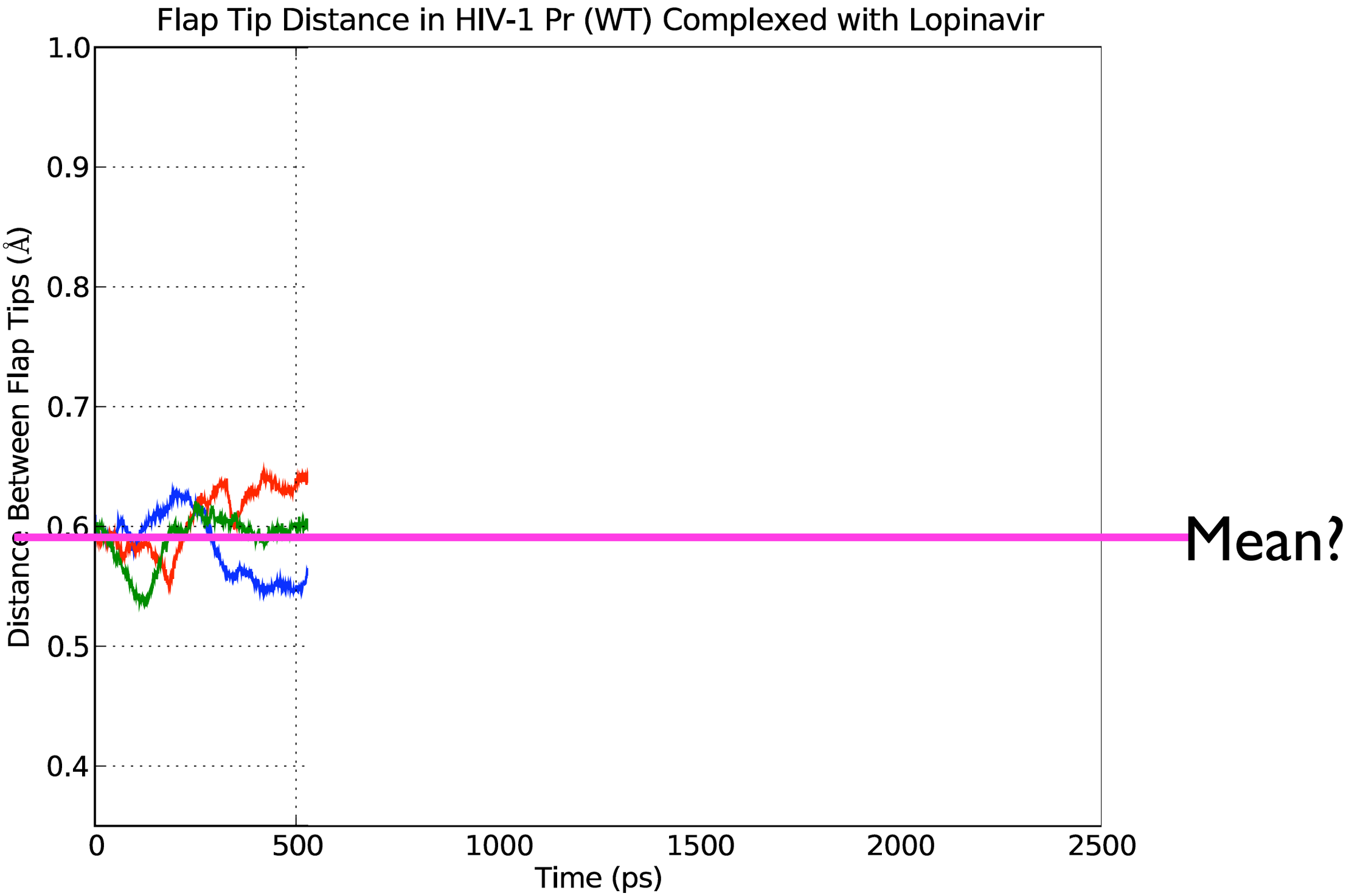
Convergence means obtaining correct final average for property we are computing. Example: HIV protease flap tip distance



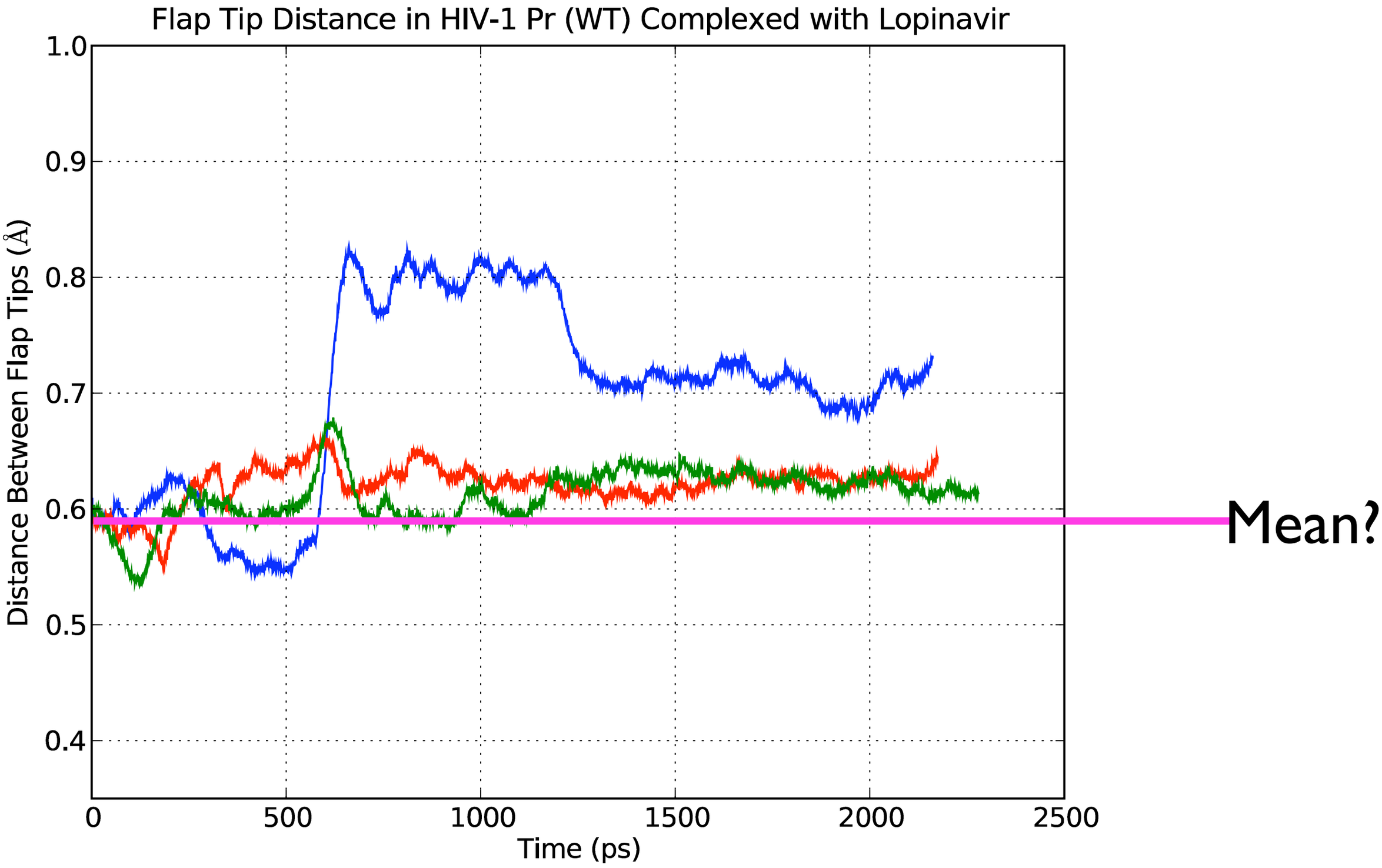
Convergence can be difficult



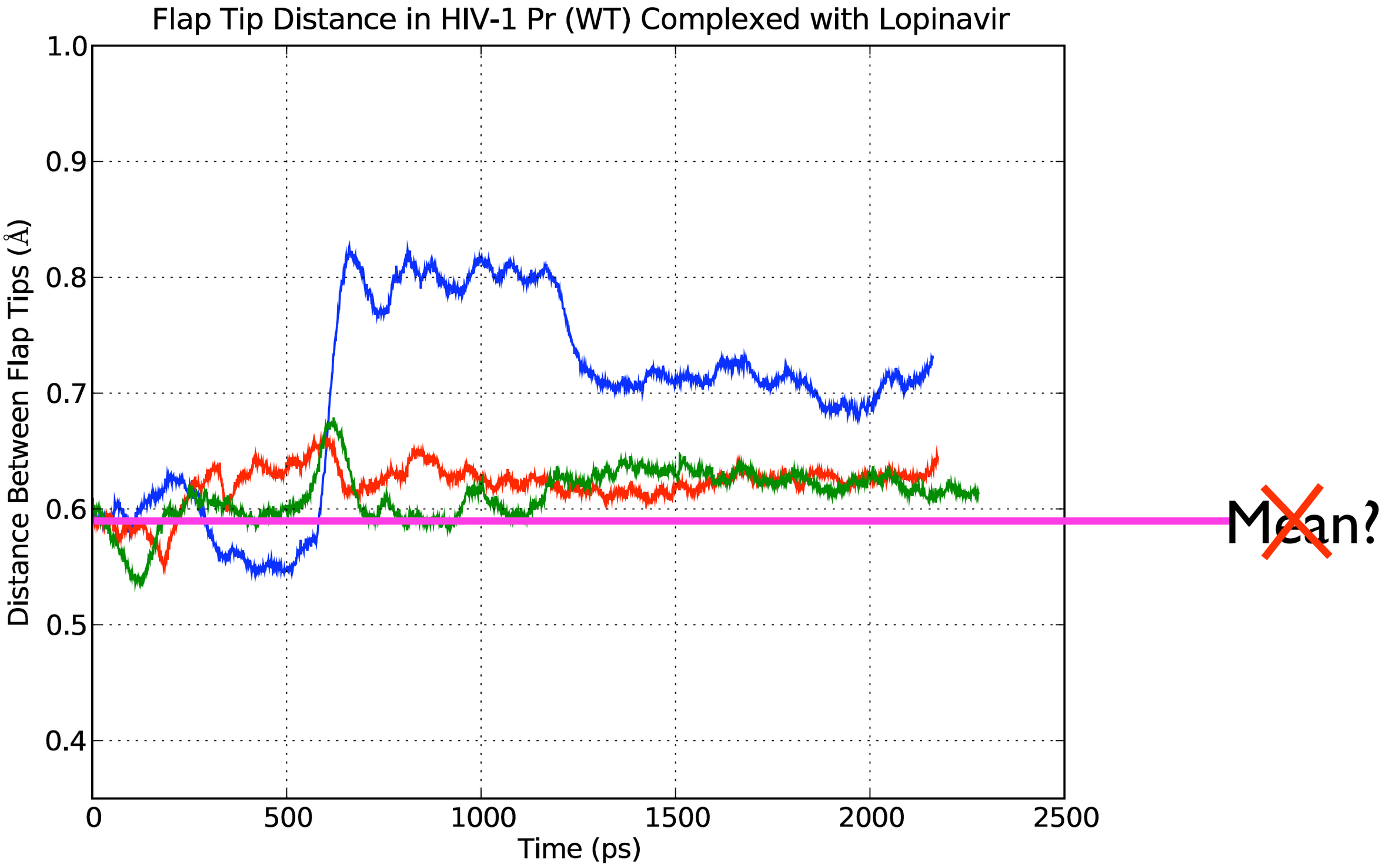
Convergence can be difficult



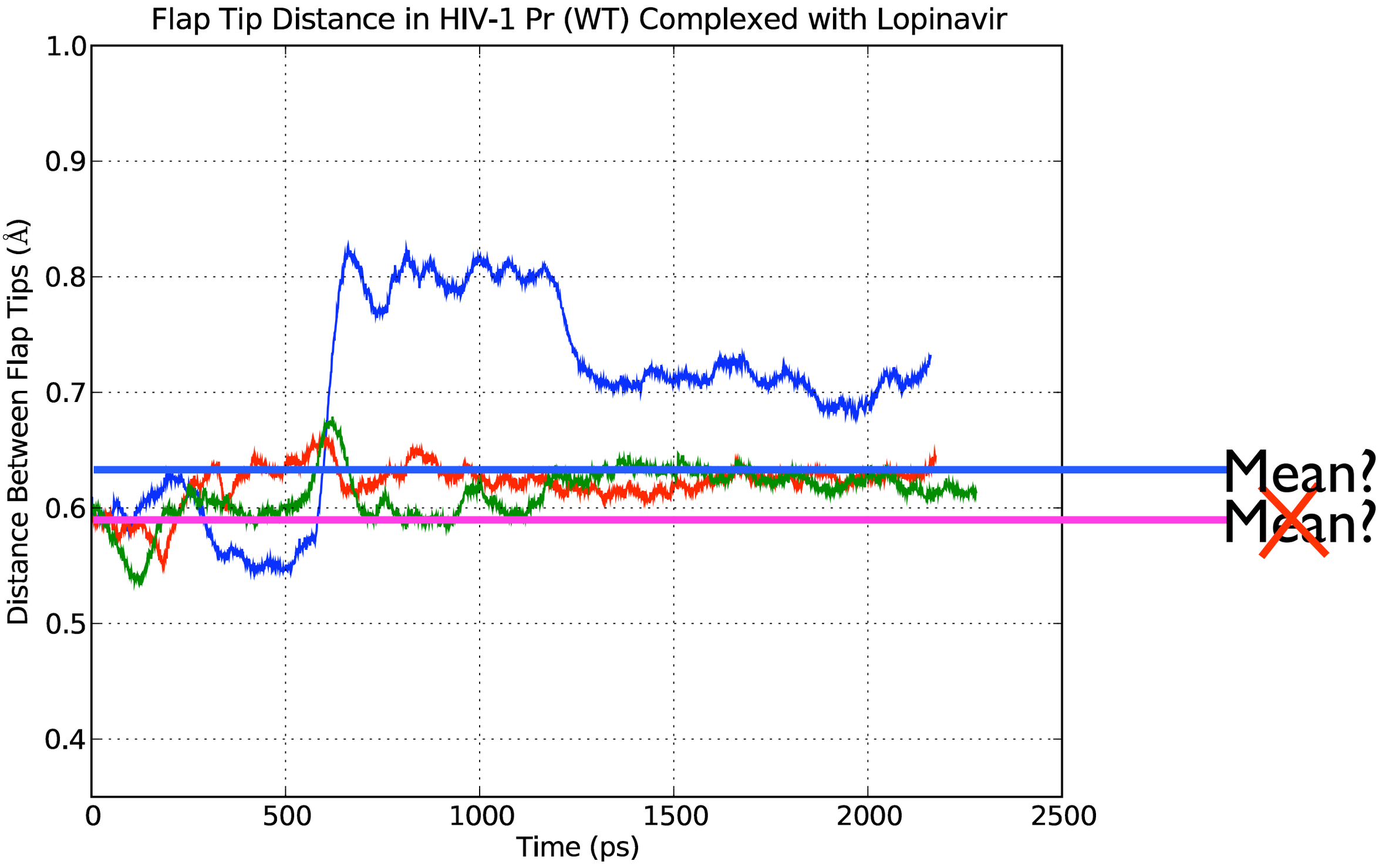
Convergence can be difficult



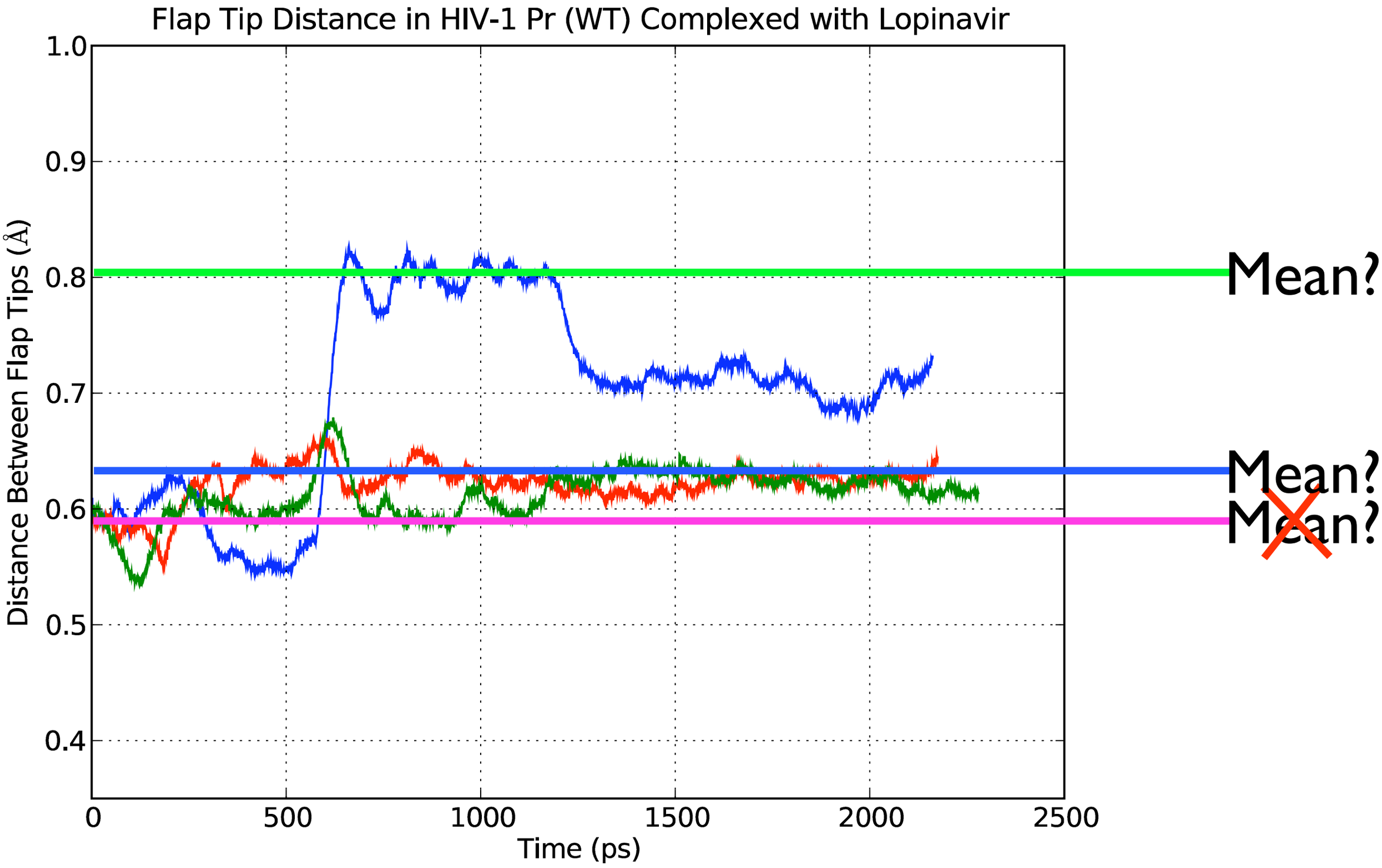
Convergence can be difficult



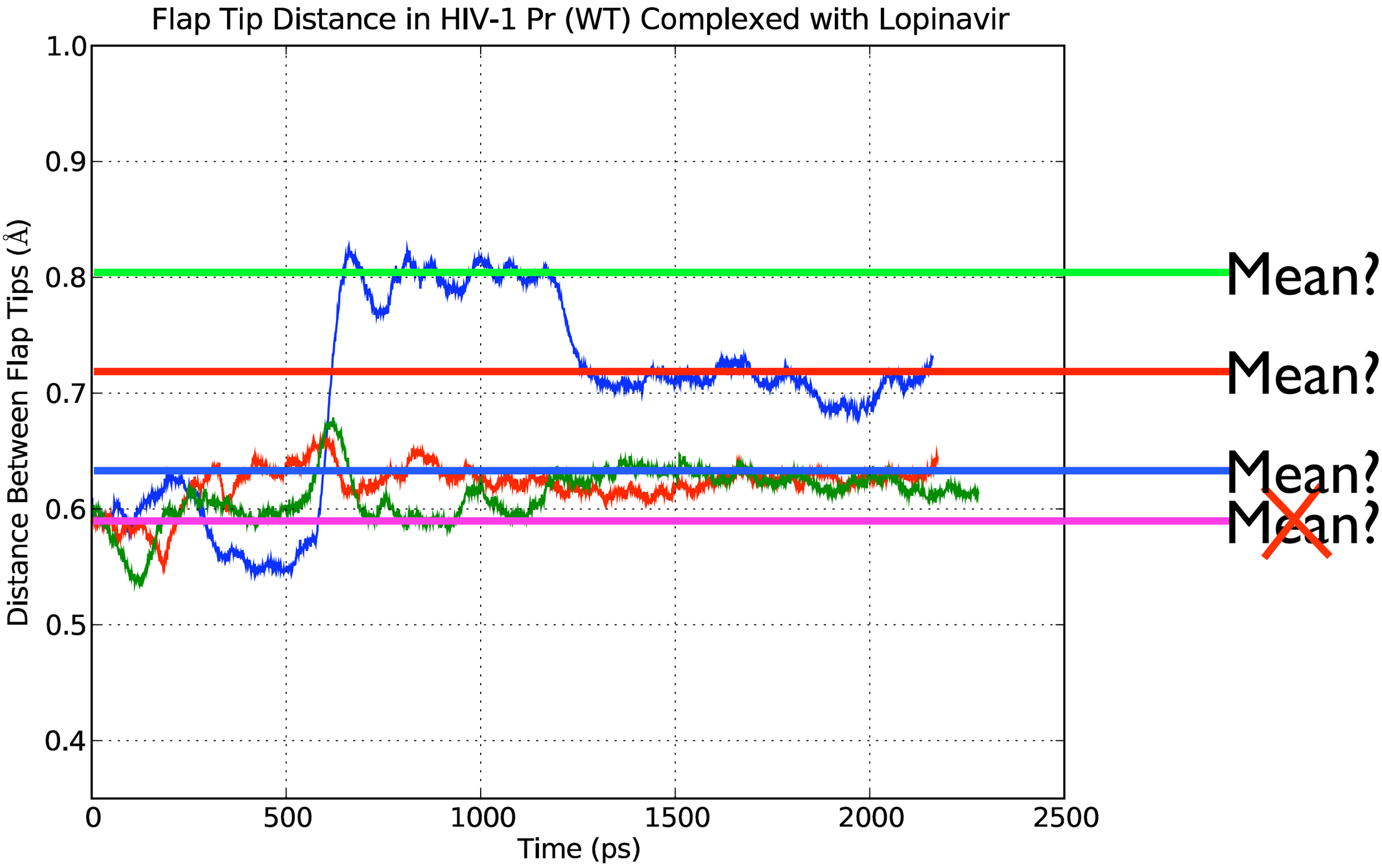
Convergence can be difficult



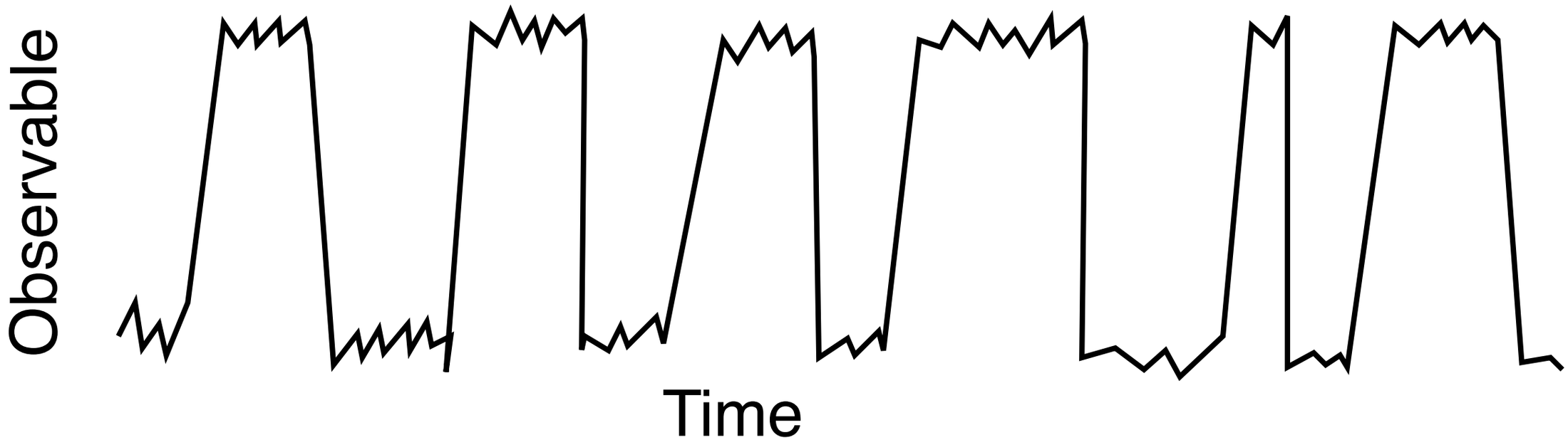
Convergence can be difficult



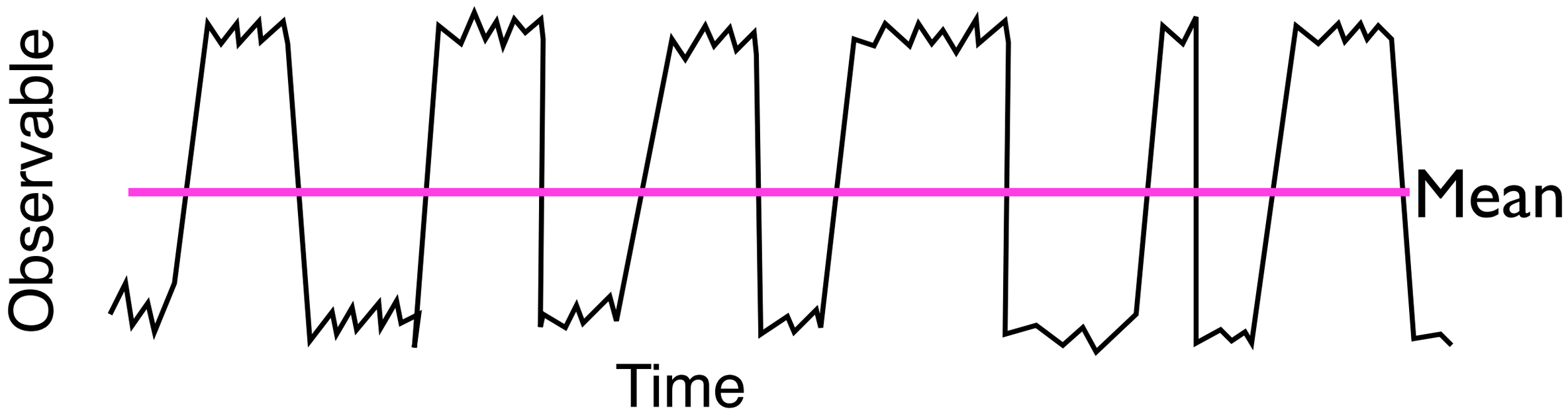
Convergence can be difficult



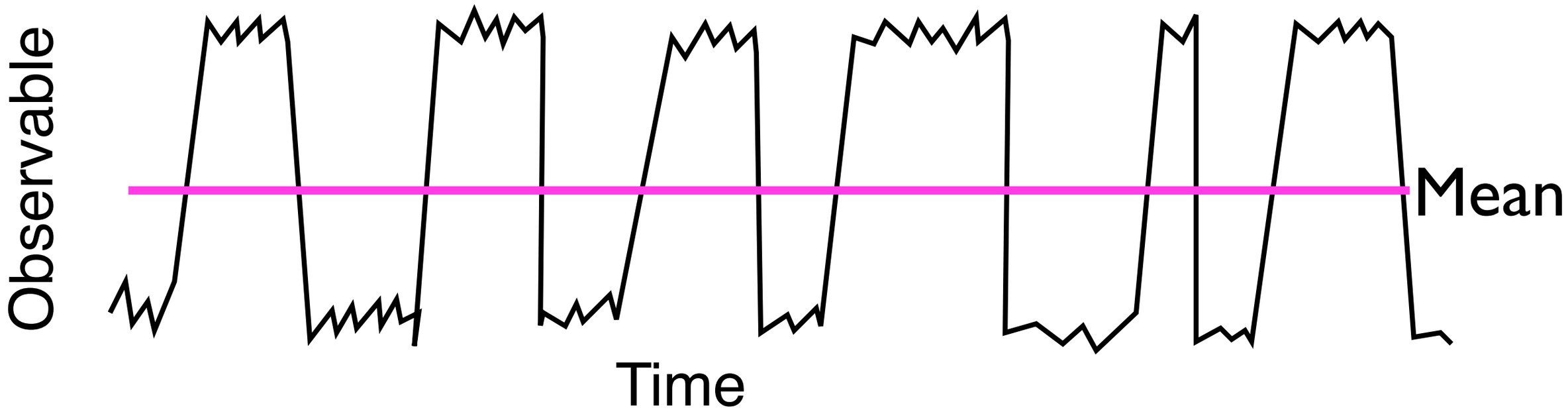
To get an accurate value of an observable with transitions, we need to see enough transitions to get the populations correct

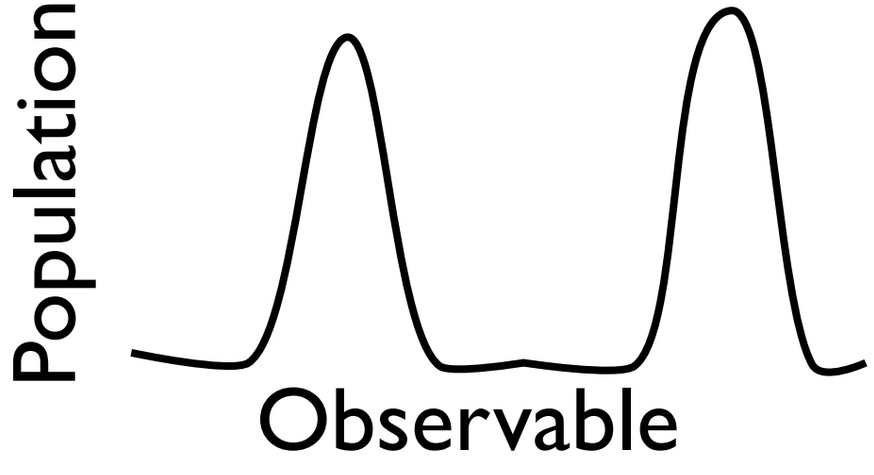


To get an accurate value of an observable with transitions, we need to see enough transitions to get the populations correct



To get an accurate value of an observable with transitions, we need to see enough transitions to get the populations correct





Remember,

$$\langle W \rangle = \int W(x)\mathcal{P}(x)dx$$

So we need correct P(x)

Even with many transitions, we aren't guaranteed convergence

- There may be other important transitions we have not yet seen that we would see if we simulated longer
- i.e. search for lowest valley -- the valley you haven't looked in may be lower than the lowest you have found so far

What are some ways to test for equilibration?

- Look at plots versus time and monitor trends for:
  - Total potential energy (fast)
  - Temperature/kinetic energy (fast)
  - RMSD (slower)
  - Other structural properties (?)
  - Number of hydrogen bonds, i.e. to water (slow)
  - Anything else you are interested in or which may be important in your system (i.e. order parameters if coming from solid, etc.)
- Assume the end is equilibrated and work backwards

The canonical ensemble specifies a certain equilibrium distribution

- A particular microstate is occupied according to

$$\wp(\mathbf{p}^N, \mathbf{r}^N) \propto \frac{e^{-\beta H(\mathbf{p}^N, \mathbf{r}^N)}}{Q(T, V, N)}$$

where $$Q(T, V, N) = \frac{1}{h^{3N} N!} \int e^{-\beta H(\mathbf{p}^N, \mathbf{r}^N)} d\mathbf{r}^N d\mathbf{p}^N$$

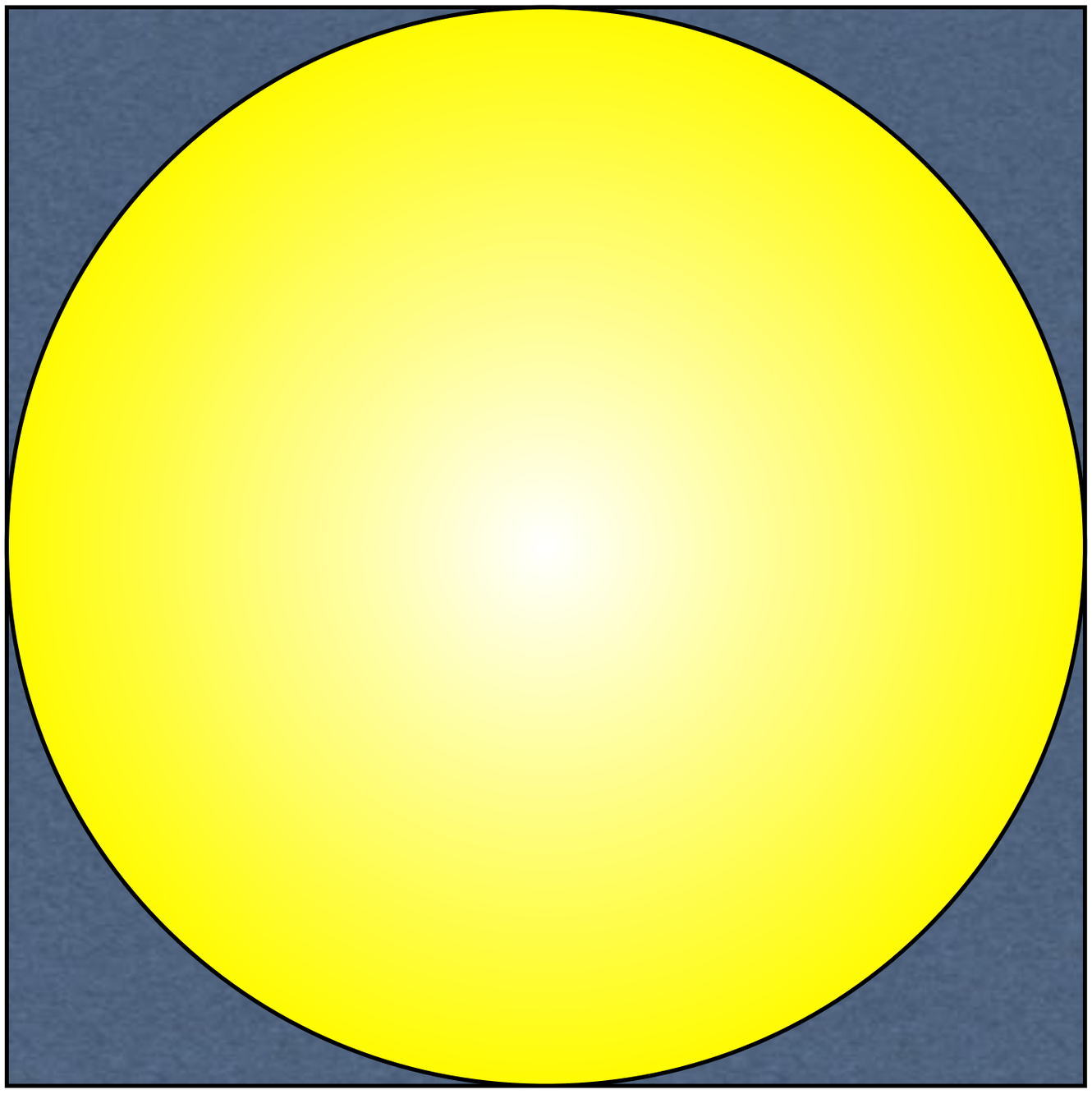
Molecular dynamics provides a way to get both thermodynamic and kinetic properties

- For kinetic properties, we need correct dynamics
- For thermodynamic properties, we just need to be in the correct ensemble
  - We don't necessarily need any particular dynamics
  - Just need to be in ensemble of interest, with correct distribution of states

If we are interested only in thermodynamics, there are other ways to sample states aside from Newtonian dynamics

- What about randomly sampling states?
  - Just randomly generating new states from scratch will not be very productive (too many high energy states relative to number of favorable states)
  - Also, if we randomly *sample* states, populations will not be correct
- Maybe there are ways we can bias "random" sampling of configurations to:
  - Avoid spending way too much effort simulating high energy states
  - Get correct ensemble
- Monte Carlo: A "random" way to do just this
  - Named after Monte Carlo Casino

Monte Carlo can be used to approximate pi



Monte Carlo methods are an alternative approach that has these properties

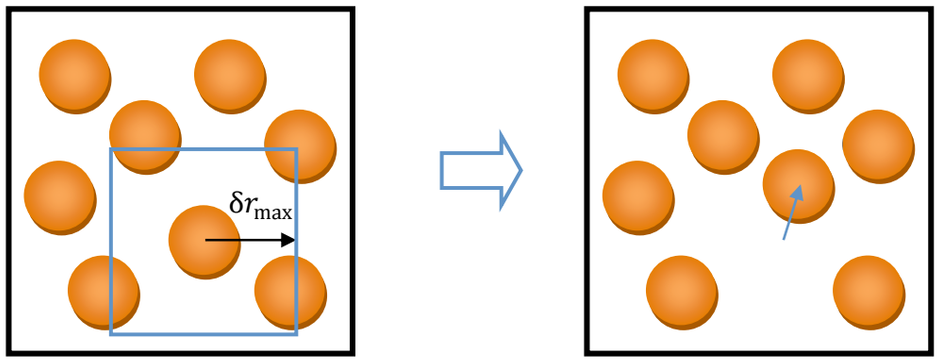
- Easily treat different thermodynamic ensembles
  - Simple to perform a rigorous constant temperature simulation (no special thermostat needed)
- Not subject to inaccuracies due to discrete-time approximations of the equations of motion
- Require only energies (not force calculations)
- Offer flexibility in choosing random moves which can speed certain types of sampling
  - But, efficiency is a function of the move set
- Downside: No dynamics

An example will help illustrate how the approach is going to work -- consider an LJ fluid

- Metropolis MC takes a series of steps at a temperature T. Each "Monte Carlo" step:
  - Randomly pick a particle
  - Change each of x, y, and z, by a small amount between $-\Delta r_{max}$ and Δr_{max}
  - Compute the energy change, $\Delta U = U_2 - U_1$
  - Apply the "Metropolis criterion" to decide whether to keep the move or reject it:
    - If $\Delta U<0$, accept
    - If $\Delta U>0$, accept with probability $P^{\text{acc}} = e^{-\Delta U/T}$
  - If accepted, keep the new configuration
  - In *either case (*accepted or rejected) update any running averages with state, energy

MC generates correct distribution; how the moves are made is an option

- Aggregated over enough steps, an MC simulation conducted this way produces configurations obeying the canonical distribution: $\wp(\mathbf{r}^N) \propto e^{-\frac{U(\mathbf{r}^N)}{T}}$
- We just discussed single-particle displacement *moves*



- For these moves, the maximum displacement is adjustable
  - Adjusted so that 30-50% of moves are accepted:
    - If acceptance is too low, we waste time never going anywhere
    - If too high, we could be making bigger moves to move further

Since conformations are sampled from the correct ensemble, thermodynamic averages are computed over randomly selected samples

- To compute the average of some property x, we just select random values of x and average over them
- Values already have the correct distribution: $\wp(\mathbf{r}^N) \propto e^{-\frac{U(\mathbf{r}^N)}{T}}$
- So, we just sum over M samples: $\langle x \rangle = \frac{1}{M} \sum_{i=1}^{M} x$
- Typically, like MD, we do not use all of the samples since the samples are correlated (sequentially, which no longer corresponds to time)
- This applies to any equilibrium property (not to transport properties)

Historically, MC just means some kind of random sampling, but now it usually means *importance sampling*

- Importance sampling:
  - Generating states that are low energy and are "important" for computing average properties (which are dominated by contributions of low energy states)
  - Typically, also interested in methods which generate the correct probability distribution, though not always
    - Some advanced techniques deliberately generate other distributions, but we will revisit these

Why Metropolis MC works: It generates a Markov chain of states

- A Markov chain describes a particular random process with these properties:
  - Outcome of each trial depends only on the current state, not upon any previous trials (history)
  - There are a finite number of total states (i.e. macrostates)

Importance sampling in MC generates a Markov chain as follows

- Start in some initial configuration A
- Make a move:
  - Randomly perturb, generating a new suggested configuration (MC move)
  - Accept or reject the proposed move, using a criterion so that configurations are generated according to $\wp(\mathbf{r}^N)$ at long times
- Repeat again and again
- Final trajectory, or set of states, includes *all* states sampled
  - Not just those where moves were accepted, but also the states retained when moves were rejected

MC and MD have similarities and differences

- Similarities:
  - Both approaches work with simulation steps
  - Both can be used to compute thermodynamic properties
- Differences
  - MC is stochastic, not deterministic (final state not determined by initial state)
  - MC does not consider velocities
  - Future state after one step depends only on current state and no previous state

In order for MC to work, we have to have a suitable way to accept or reject moves

- Not all ways of accepting or rejecting proposed moves will give the correct distribution of states
- There is an important criteria for this called *detailed balance* that we need to understand

- Some definitions:
  - $\wp_m(i)$ -- probability of being in state m at step i
  - π_{mn} -- transition probability: Given that current state is m, probability of going to state n

- Ultimately, we need correct distribution:

$$\lim_{i\to\infty} \wp_m(i) = \frac{e^{-\beta U_m}}{Z}$$

At this point, probability distribution should become "stationary" and no longer change

The requirement of reaching a steady state helps understand method's requirements

- If we start with an initial probability distribution, how does it change after a step?

$$\wp_m(i+1) = \wp_m(i) - \sum_n \pi_{mn}\wp_m(i) + \sum_n \pi_{nm}\wp_n(i)$$

The requirement of reaching a steady state helps understand method's requirements

- If we start with an initial probability distribution, how does it change after a step?

$$\wp_m(i+1) = \wp_m(i) - \sum_n \pi_{mn}\wp_m(i) + \sum_n \pi_{nm}\wp_n(i)$$

Probability of being in state m

The requirement of reaching a steady state helps understand method's requirements

- If we start with an initial probability distribution, how does it change after a step?

$$\wp_m(i+1) = \wp_m(i) - \sum_n \pi_{mn}\wp_m(i) + \sum_n \pi_{nm}\wp_n(i)$$

Probability of being in state m

Probability of leaving m for any other state n (summed over n)

The requirement of reaching a steady state helps understand method's requirements

- If we start with an initial probability distribution, how does it change after a step?

$$\wp_m(i+1) = \wp_m(i) - \sum_n \pi_{mn}\wp_m(i) + \sum_n \pi_{nm}\wp_n(i)$$

Probability of being in state m

Probability of leaving m for any other state n (summed over n)

Probability of entering m from any other state n (summed over n)

The requirement of reaching a steady state helps understand method's requirements

- If we start with an initial probability distribution, how does it change after a step?

$$\wp_m(i+1) = \wp_m(i) - \sum_n \pi_{mn}\wp_m(i) + \sum_n \pi_{nm}\wp_n(i)$$

Probability of being in state m

Probability of leaving m for any other state n (summed over n)

Probability of entering m from any other state n (summed over n)

- At equilibrium, the probabilities must not be changing, so we must have:

$$\sum_n \pi_{mn}\wp_m = \sum_n \pi_{nm}\wp_n \qquad \text{for all } m$$

This results in balance and detailed balance conditions

- Thus, for steady state, the *balance equation* must be satisfied:

$$\sum_n \pi_{mn} \wp_m = \sum_n \pi_{nm} \wp_n \qquad \text{for all } m$$

(The net probability of leaving state m is the same as the net probability of entering it)

- One simple way to satisfy this is *detailed balance:*

$$\pi_{mn} \wp_m = \pi_{nm} \wp_n \qquad \text{for all } m, n$$

(The net probability of leaving state m and going to state n is the same as the net probability of leaving state n and going to state m)

Detailed balance conditions provide way to get acceptance probabilities for moves

- Any acceptance probability satisfying this equation is OK; often the Metropolis criterion is the most efficient $P_{12}^{\text{acc}} = \min\left[1, e^{-\beta(U_2 - U_1)}\right]$

  In other words, take the move if it is downhill; take uphill moves with a reduced probability

A few practical issues

- It is unnecessary to compute the total potential energy when computing energy changes
  - We are proposing moving a single particle; we only need to recompute interactions with that particle (time savings!)
- Except, precision issues mean that we need to recompute the total energy sometimes
  - i.e. every 10-100 moves per particle
  - Do not update until AFTER a move is accepted
- Like MD, MC needs equilibration to reach a stationary state
- Remember, include every step (not just accepted steps) in running averages

Property averages are still susceptible to correlations in the data

- MC may still have “time” correlations
- Still need number of effective samples for error analysis

Random number generates are really *pseoudorandom*

- Computers can't really do anything random
  - Presents challenges when we want to do something "by chance"
- Pseudorandom numbers generate numbers in a deterministic sequence but have properties like true random variables
  - Typically take a seed value that gets algorithm started
  - If you write your own code, be aware of period and potential for subtle patterns and correlations
    - Period: How long it takes until sequence repeats. Long is good
    - Python uses Mersenne twister, period 2^{19937} (very good!)

Practicalities of accepting or rejecting moves

- Easiest way to accept or reject with a certain probability:

  If $\Delta U > 0$, compute $P^{\mathrm{acc}} = e^{-\Delta U/T}$. Draw a random number r on the interval $[0.0, 1.0)$ and accept the move if and only if $P^{\mathrm{acc}} > r$.