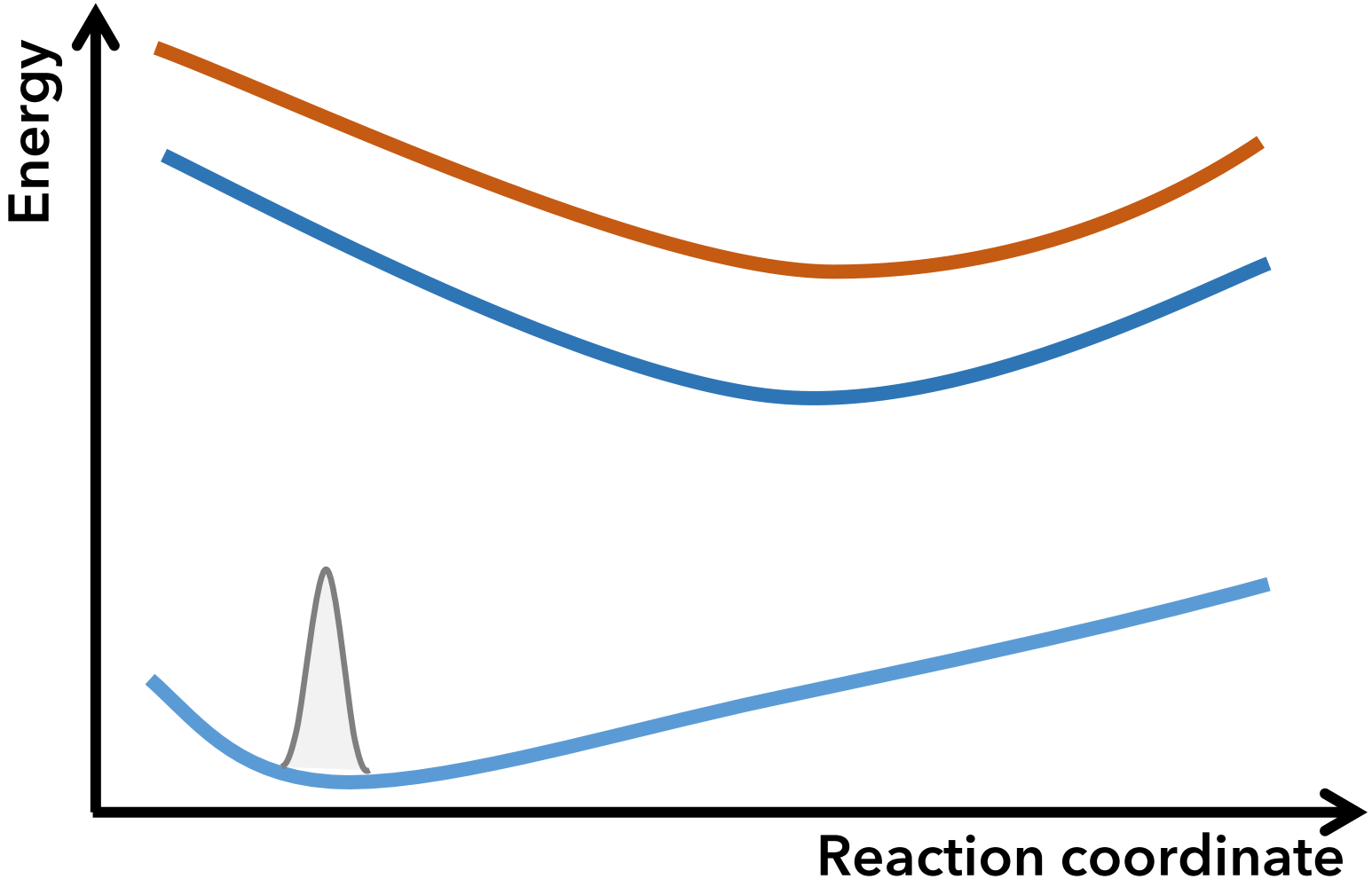
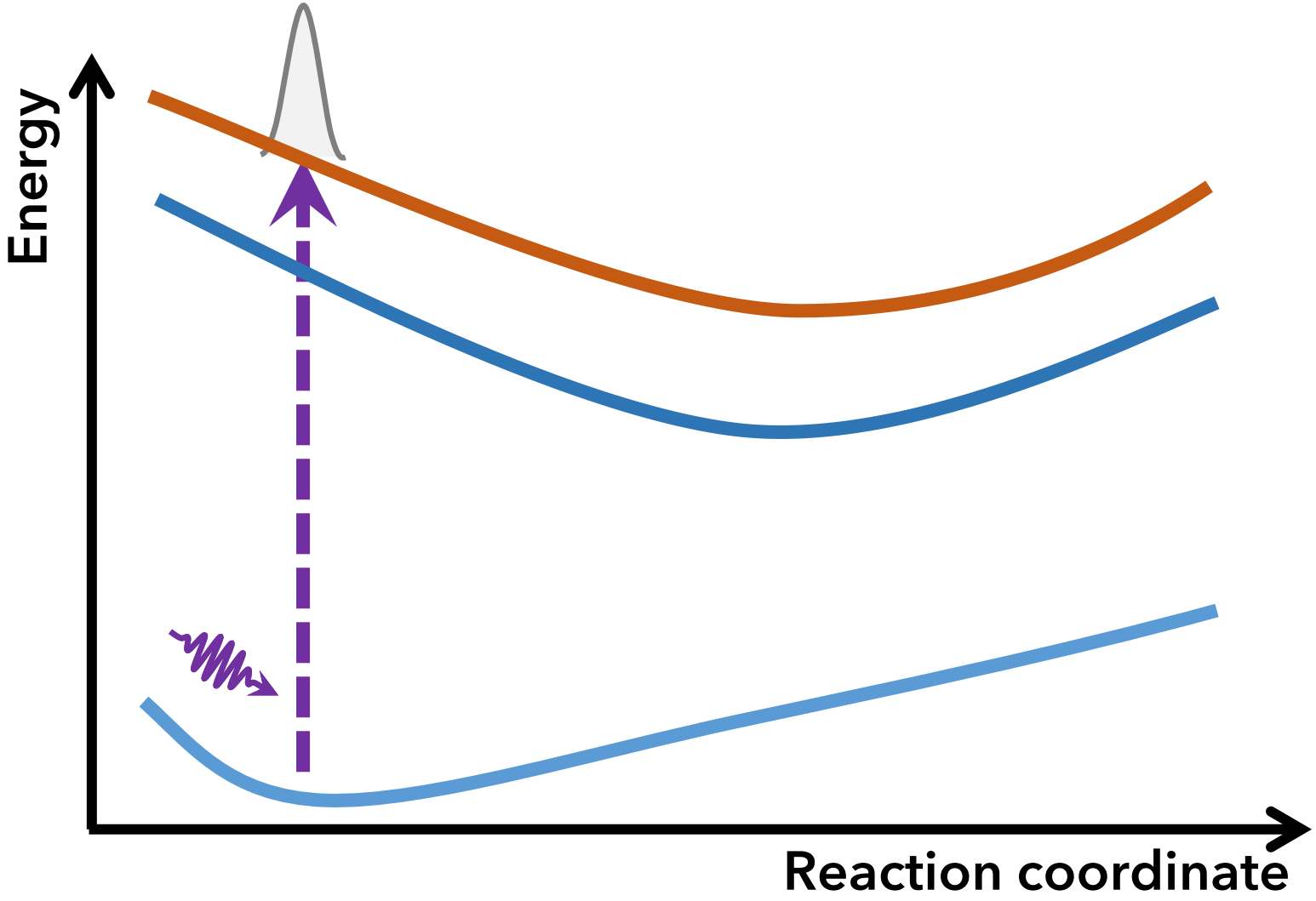
L11 – Statistical Mechanics 3

Monte Carlo spectrum simulations and statistical analysis

Practical use of Monte Carlo integration: Spectrum simulation


Energy
Reaction coordinate


Energy
Reaction coordinate

A simple expression for the photoabsorption cross-section σ from the ground state (0) to state *n* is (in atomic units)

$$\sigma(E) = \frac{2\pi^2}{c}\frac{1}{E}\int \left|\chi_0(\mathbf{R})\right|^2 \Delta E_{0n}(\mathbf{R})\, f_{0n}(\mathbf{R})\, g\left(E - \Delta E_{0n}(\mathbf{R}), \sigma\right) d\mathbf{R}$$

E : photon energy

$\mathbf{R}$: nuclear coordinates

$\chi_0(\mathbf{R})$: nuclear wave function of the ground state

$\Delta E_{0n}(\mathbf{R})$: potential energy gap $0 \rightarrow n$

$f_{0n}(\mathbf{R})$: oscillator strength $0 \rightarrow n$

$g(E - \Delta E, \sigma)$: Normalized Gaussian in E,

centered in ΔE and standard deviation σ

Crespo-Otero; Barbatti. *Theor Chem Acc* **2012**, *131*, 1237

A simple expression for the photoabsorption cross-section σ from the ground state (0) to state *n* is (in atomic units)

$$\sigma(E) = \frac{2\pi^2}{c}\frac{1}{E}\int \left|\chi_0(\mathbf{R})\right|^2 \Delta E_{0n}(\mathbf{R})\, f_{0n}(\mathbf{R})\, g\left(E - \Delta E_{0n}(\mathbf{R}), \sigma\right) d\mathbf{R}$$

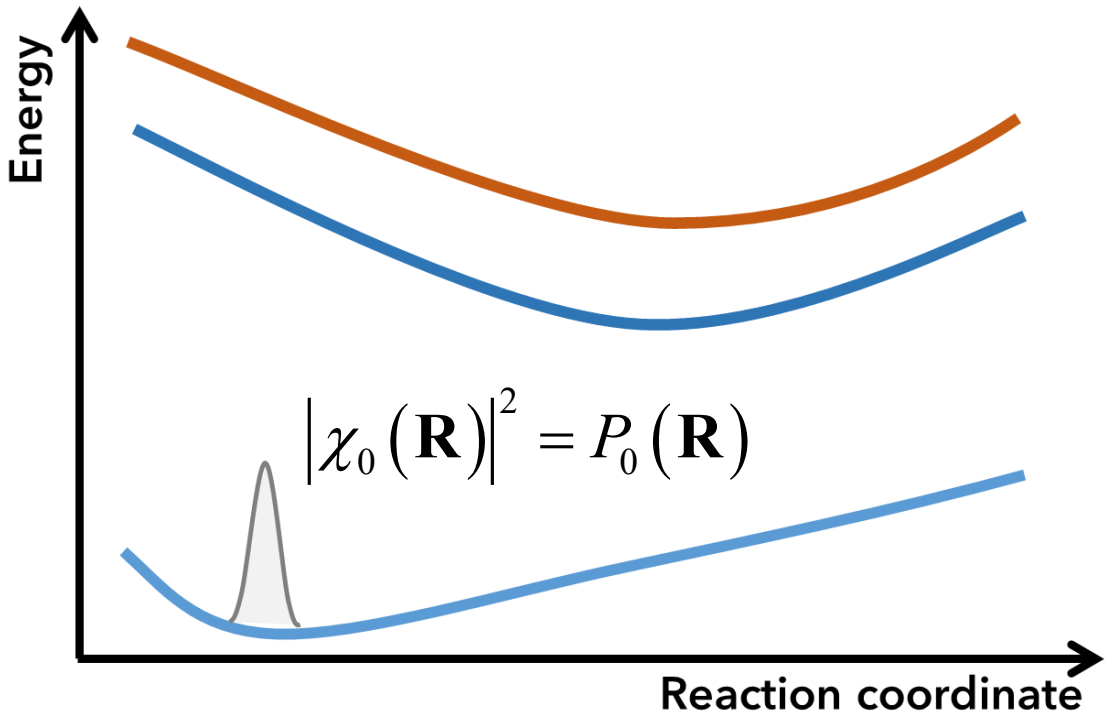
σ has unity of area and it is given in *bohr*^{2}

We can convert it to extinction coefficient ε (in L.mol^{-1}.cm^{-1}) via

$$\varepsilon(E) = 7321.2134\, \sigma(E)$$

Crespo-Otero; Barbatti. *Theor Chem Acc* **2012**, *131*, 1237

$$\sigma(E) = \frac{2\pi^2}{c}\frac{1}{E}\int \left|\chi_0(\mathbf{R})\right|^2 \Delta E_{0n}(\mathbf{R}) f_{0n}(\mathbf{R}) g\left(E - \Delta E_{0n}(\mathbf{R}), \sigma\right) d\mathbf{R}$$



$P_0(\mathbf{R})$ is the **probability density** of finding the molecule in the ground state with geometry **R**.

$P_0(\mathbf{R})$.d**R** is the **probability** of finding the molecule in the ground state with geometry **R** within the volume d**R**.

$$\sigma(E) = \int P_0(\mathbf{R}) \underbrace{\left[\frac{2\pi^2}{c} \frac{1}{E} \Delta E_{0n}(\mathbf{R}) f_{0n}(\mathbf{R}) g\left(E - \Delta E_{0n}(\mathbf{R}), \sigma\right) \right]}_{h(E,\mathbf{R})} d\mathbf{R}$$

$$\sigma(E) = \int P_0(\mathbf{R}) h(E, \mathbf{R}) d\mathbf{R}$$

We use Monte Carlo to integrate this expression in **R**

$$\sigma(E) = \int P_0(\mathbf{R}) h(E, \mathbf{R}) d\mathbf{R}$$

1. Sample N_p random geometries $\mathbf{R}_i$ following the distribution $P_0(\mathbf{R})$
2. Compute $h(\mathbf{R}_i)$ for each geometry
3. Estimate the integral as

$$\sigma(E) \approx \frac{1}{N_P} \sum_{i=1}^{N_p} h(E, \mathbf{R}_i)$$

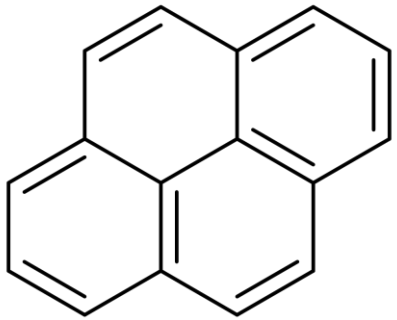
The photoabsorption cross section is

$$\sigma(E) = \frac{2\pi^2}{c} \frac{1}{N_p E} \sum_{i=1}^{N_p} \Delta E_{0n}(\mathbf{R}_i) f_{0n}(\mathbf{R}_i) g(E - \Delta E_{0n}(\mathbf{R}_i), \sigma)$$

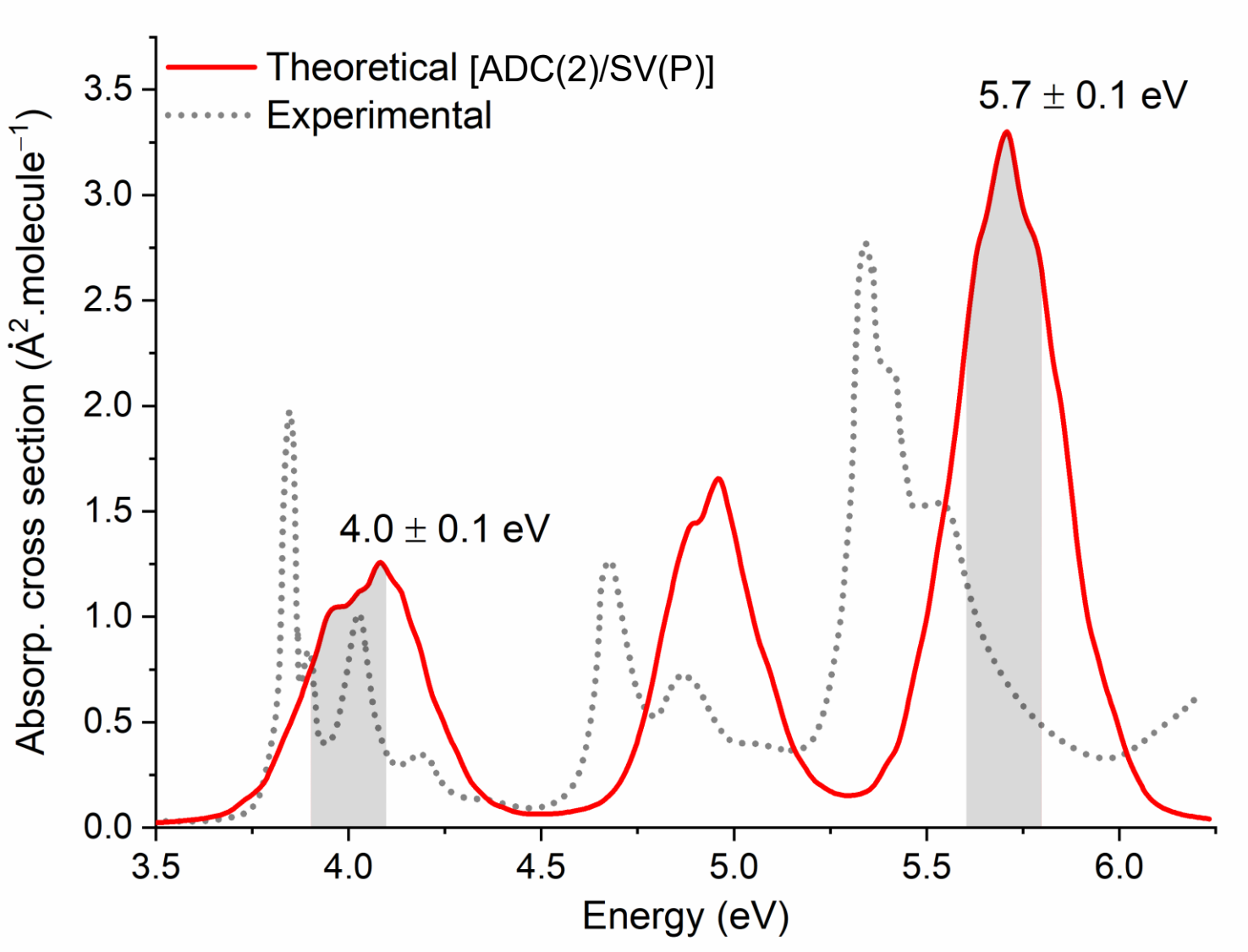
where $\mathbf{R}_i$ are nuclear geometries sampled from the probability density function $|\chi_0(\mathbf{R})|^2$.

- $\Delta E_{0n}(\mathbf{R}_i)$ and $f_{0n}(\mathbf{R}_i)$ are computed with quantum chemistry (TDDFT, for example)

- The distribution $|\chi_0(\mathbf{R})|^2$ can be taken from
  - ✓ a trajectory in the ground state
  - ✓ or from a Wigner sampling of the harmonic oscillator



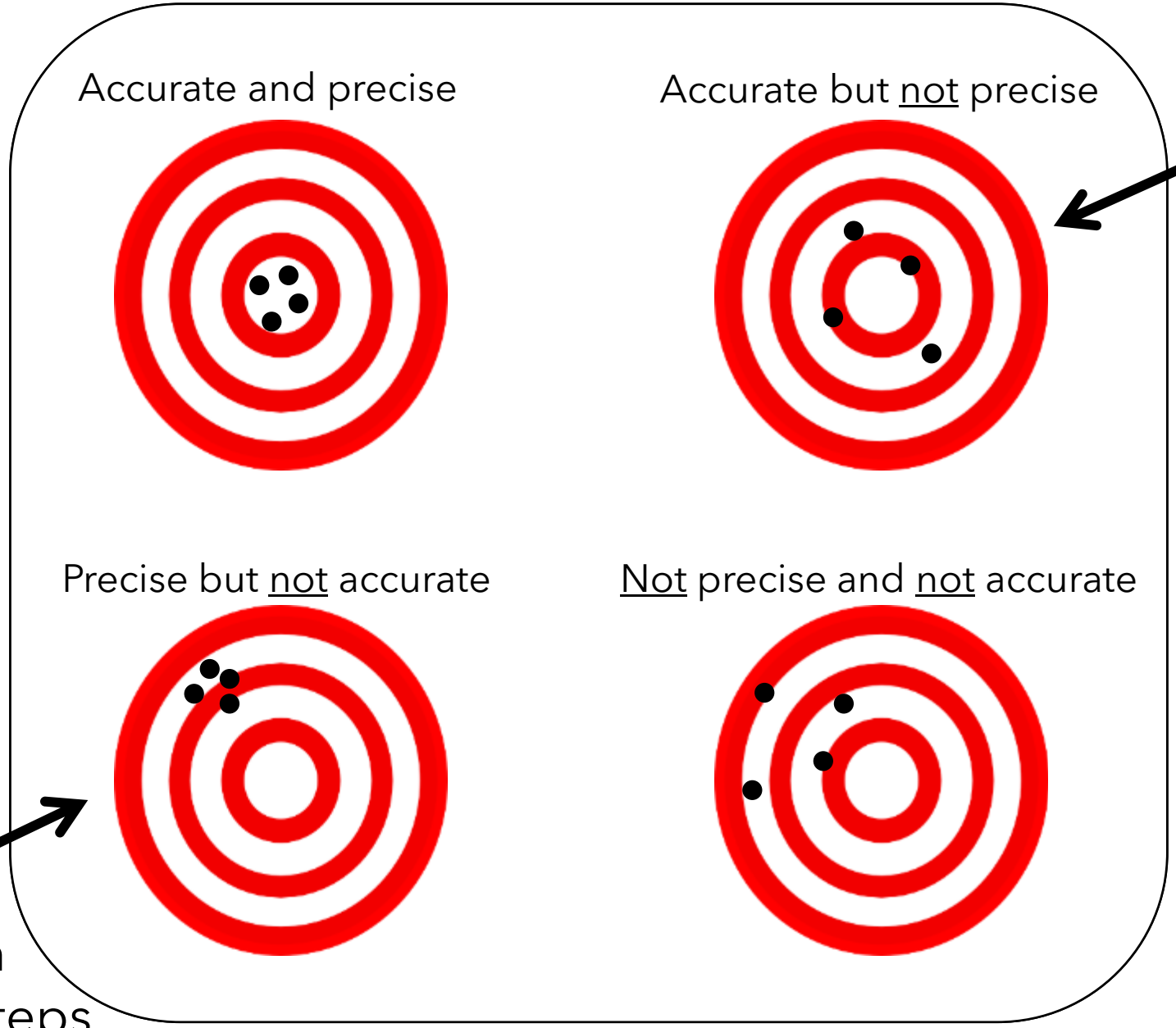
pyrene



Braun et al. *J Chem Phys* **2022**, *157*, 154305

N_p = 1000 geometries

Statistical errors in molecular dynamics

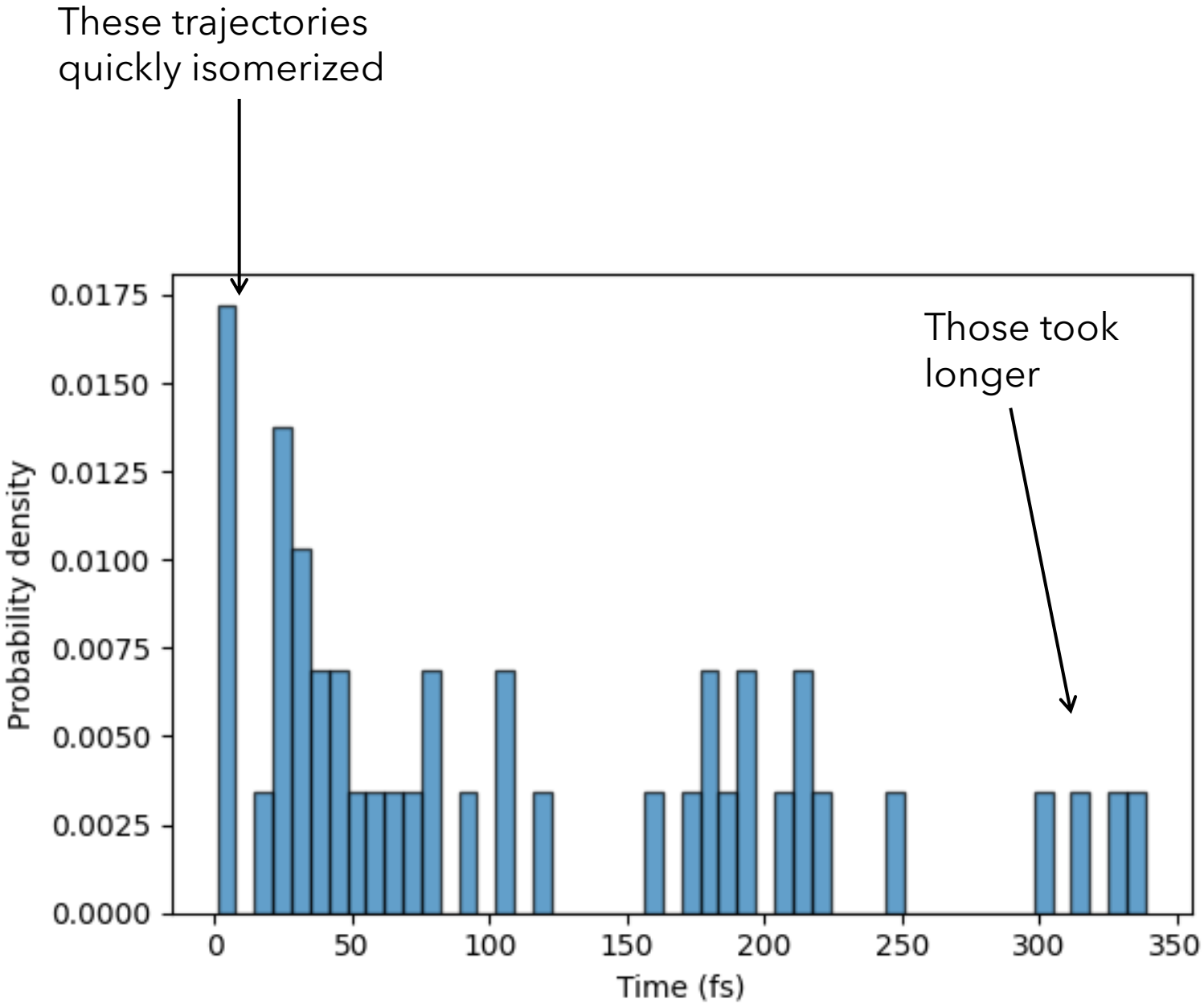

Accurate and precise
Accurate but not precise
Low statistics
Precise but not accurate
Not precise and not accurate
Bad Hamiltonian
Too large time steps

Consider the following example.
We ran 100 trajectories starting from an isomer C and find:

Channel	**Proportion of trajectories (%)**
No isomerization	5
Isomerization C→A	52
Isomerization C→B	43

The **mean value** of the isomerization time into isomer B was 150 fs with a **standard deviation** of also 150 fs.

What is the margin of error (precision) for these numbers?



Mean μ_{sim} and standard deviation σ_{sim} are the quantities of interest.

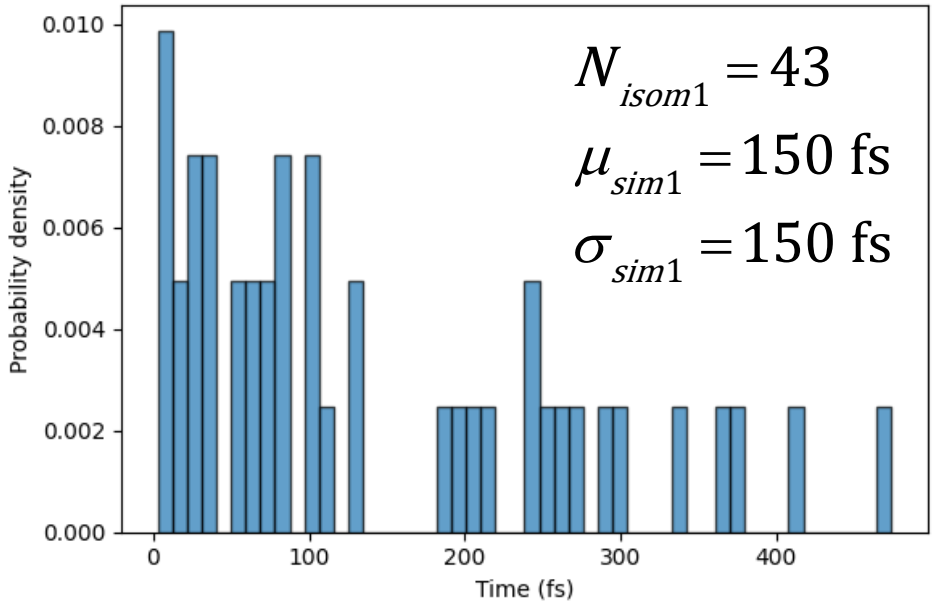
$$\mu_{sim} = \frac{1}{N_{isom}} \sum_{i=1}^{N} t_i$$

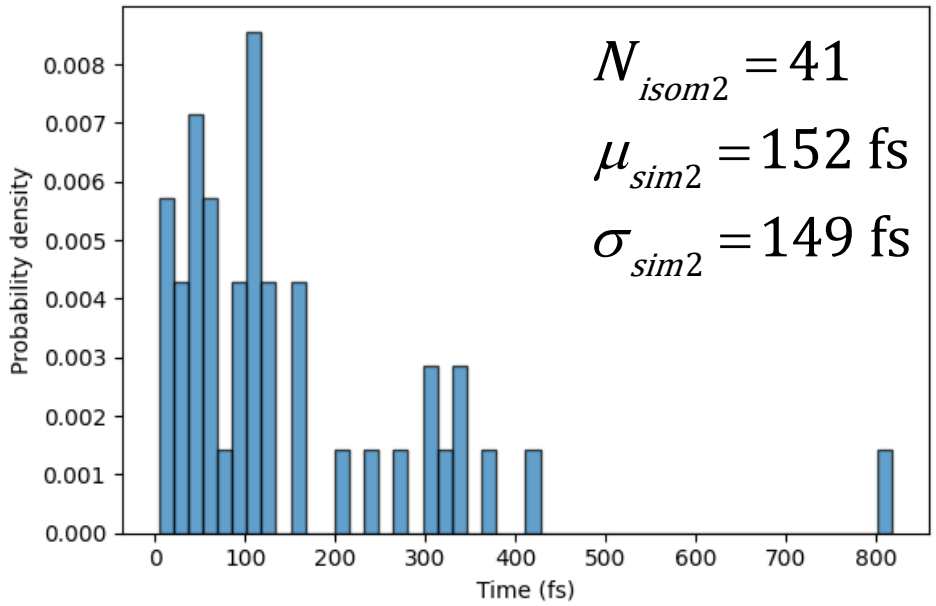
$$\sigma_{sim} = \sqrt{\frac{1}{N_{isom}} \sum_{i=1}^{N} \left(t_i - \mu_{sim} \right)^2}$$

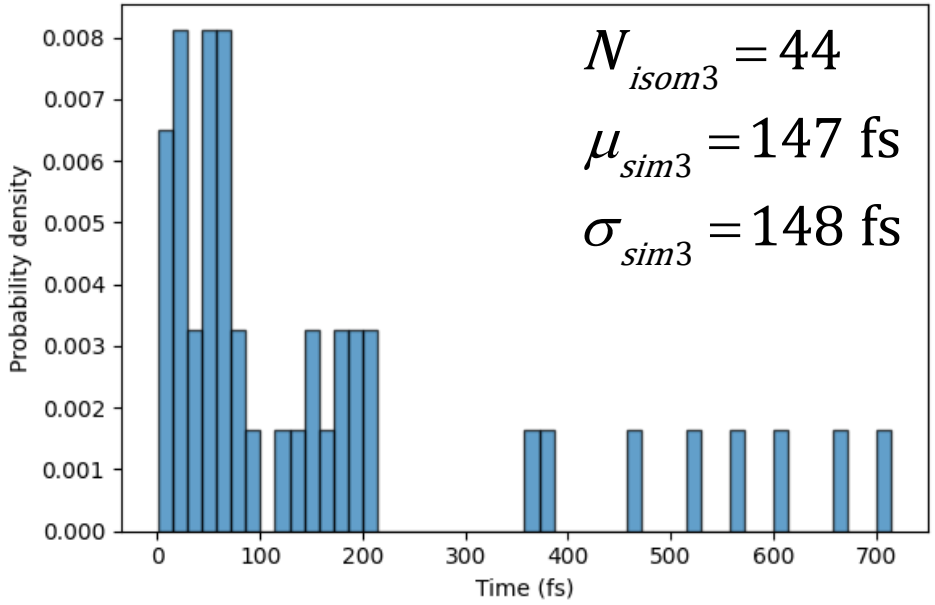
$$N_{isom} = 43$$

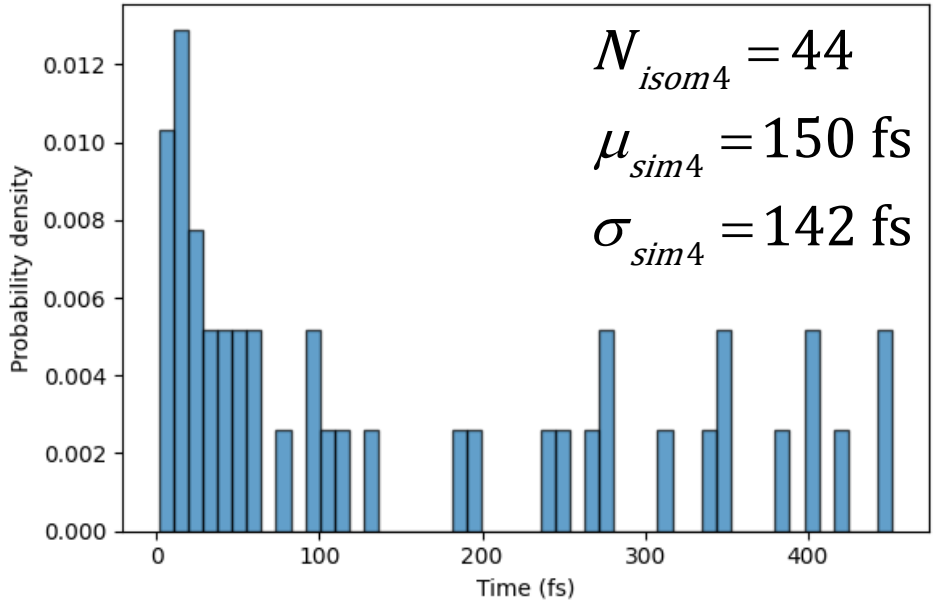
$$\mu_{sim} = 150 \text{ fs}$$

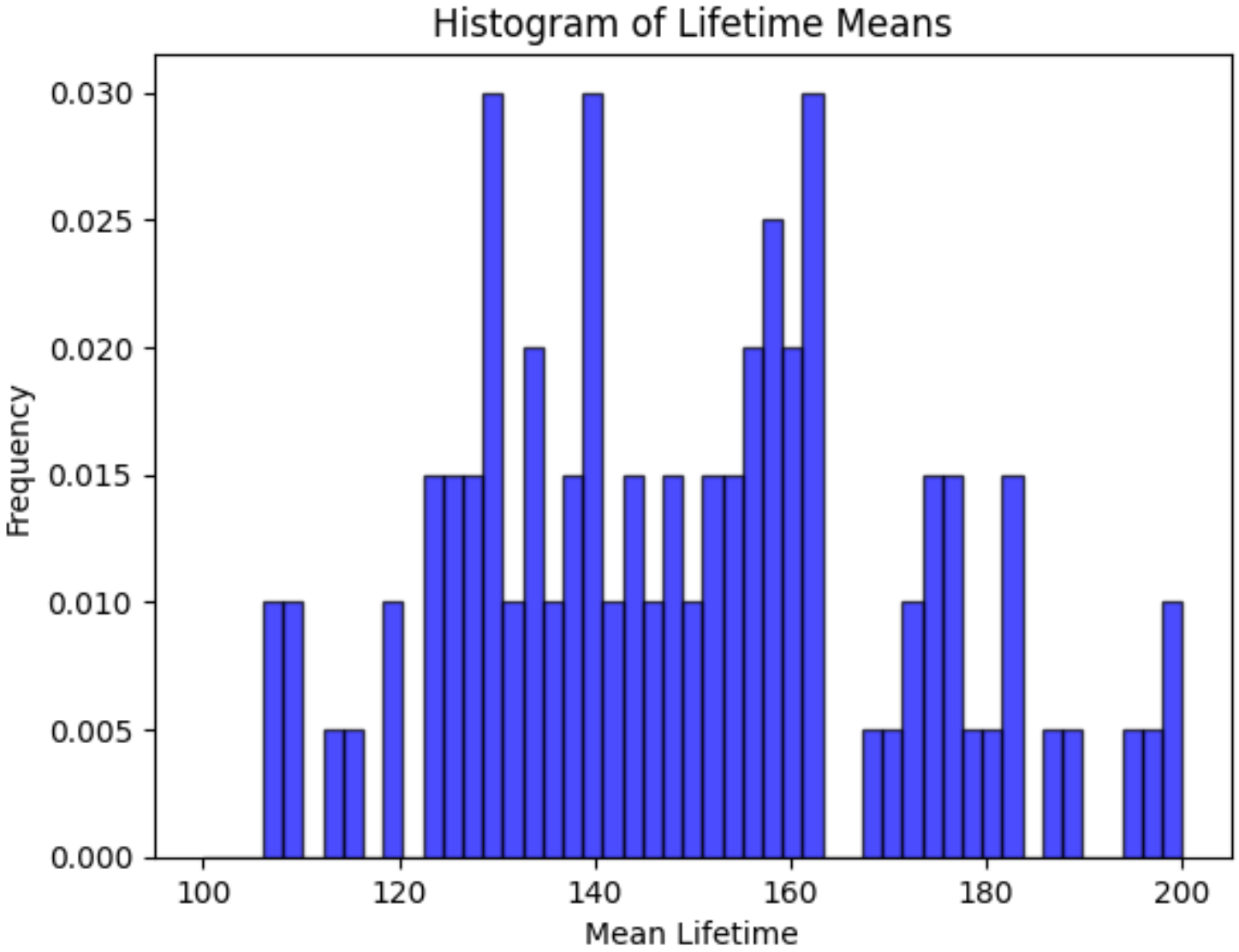
$$\sigma_{sim} = 150 \text{ fs}$$


$N_{isom1} = 43$
$\mu_{sim1} = 150$ fs
$\sigma_{sim1} = 150$ fs
Probability density
Time (fs)


$N_{isom2} = 41$
$\mu_{sim2} = 152$ fs
$\sigma_{sim2} = 149$ fs
Probability density
Time (fs)


$N_{isom3} = 44$
$\mu_{sim3} = 147$ fs
$\sigma_{sim3} = 148$ fs
Probability density
Time (fs)

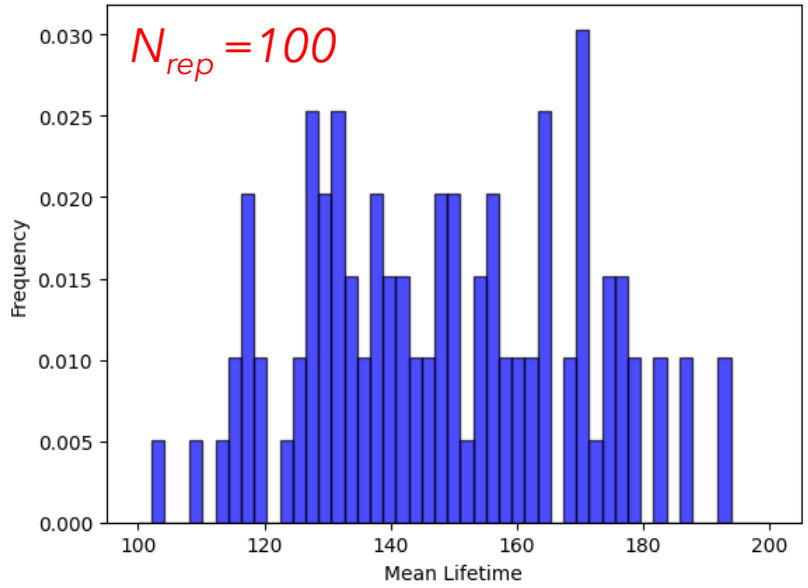

$N_{isom4} = 44$
$\mu_{sim4} = 150$ fs
$\sigma_{sim4} = 142$ fs
Probability density
Time (fs)

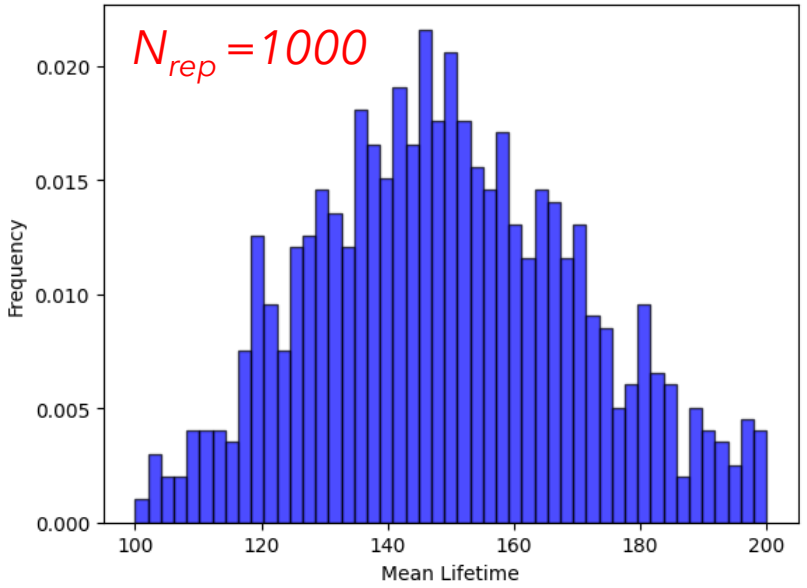


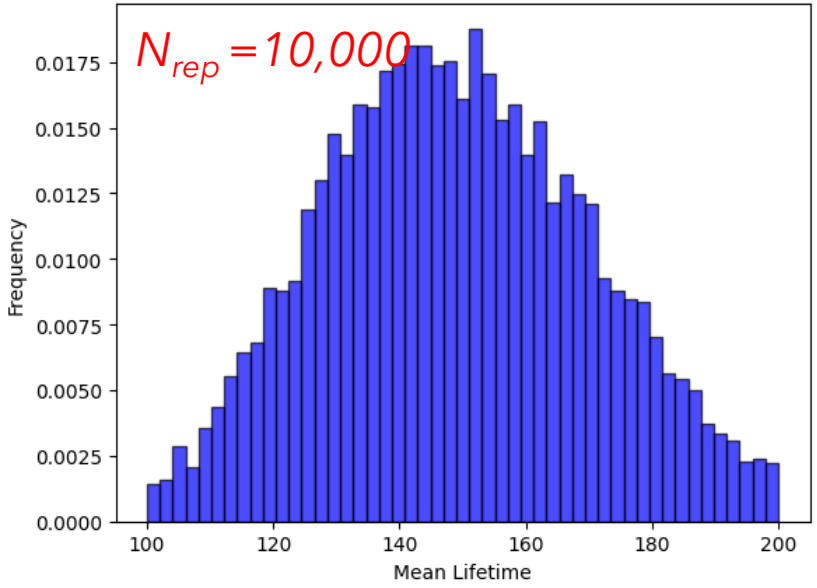
Suppose you repeat the simulation 100 times.

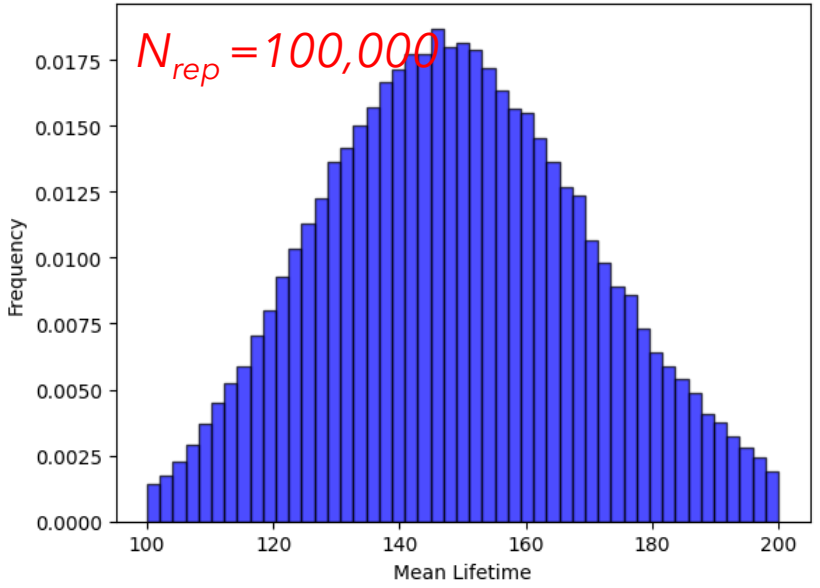
(Run N_{traj} = 100 trajectories, get the mean time; repeat N_{rep} = 100 times)

Each time, the mean will be slightly different.

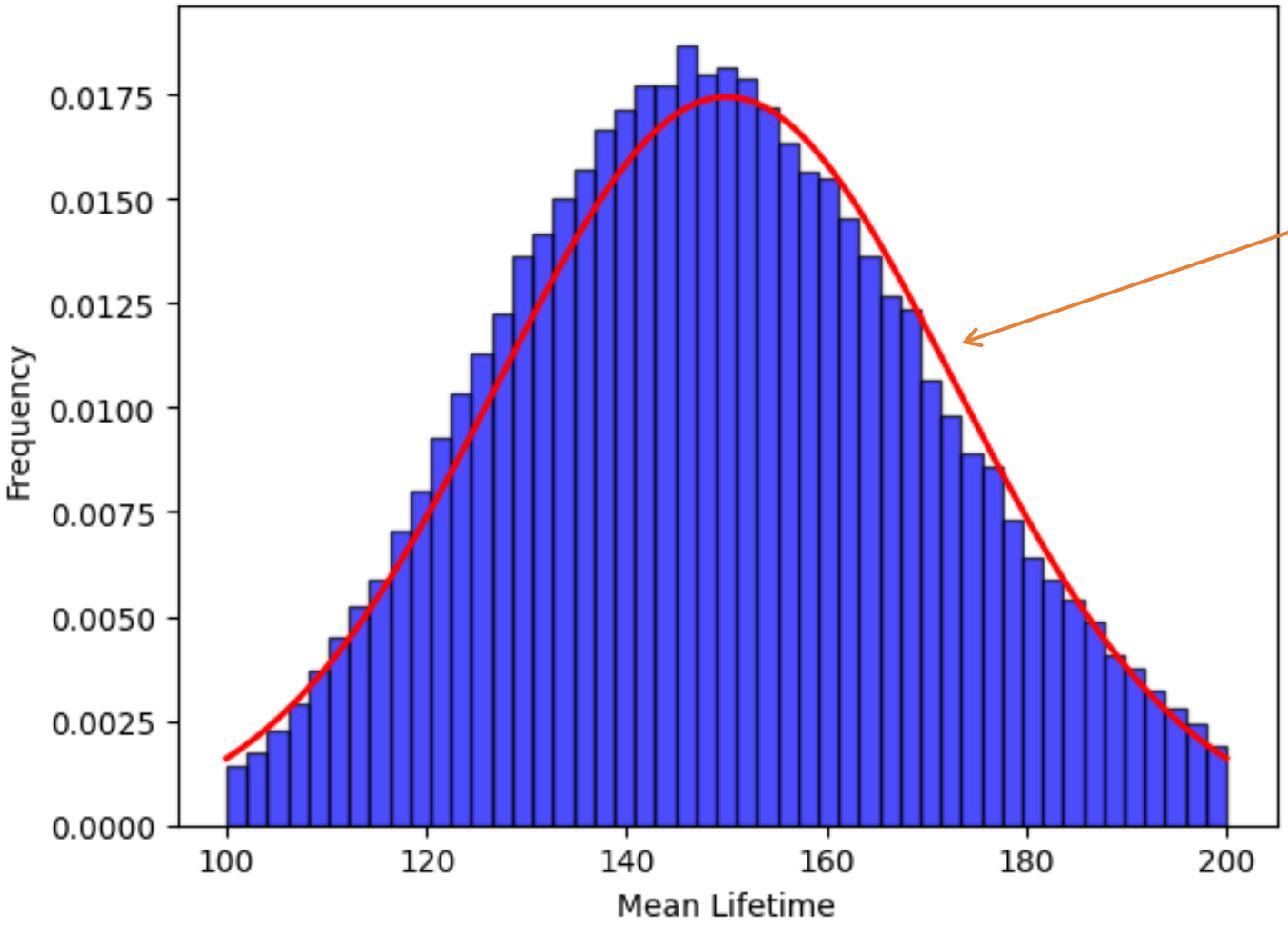








N_{rep} simulations of 100 trajectories.

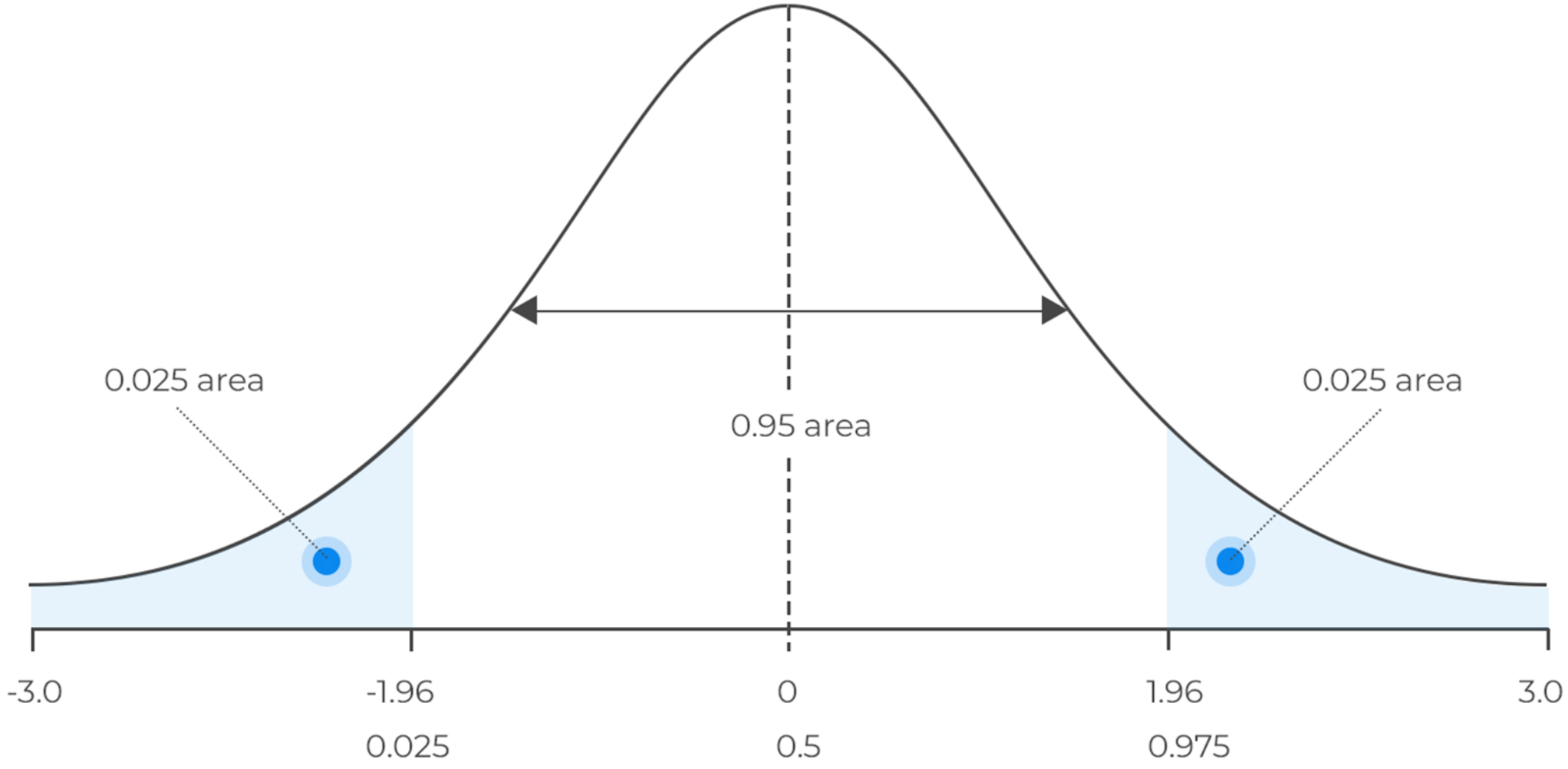


$$\rho \approx \frac{1}{\sigma_M \sqrt{2\pi}} e^{-\left(\frac{t-\mu_M}{2\sigma_M}\right)^2}$$

$$\sigma_M \approx \frac{\sigma}{\sqrt{N_{isom,M}}}$$

$N_{isom,M}$ – mean of N_{isom}
μ_M – mean of the means
σ_M – standard deviation of the means

Consider computing the isomerization time by simulating 100 trajectories. Find a range of values such that you're 95% sure the isomerization time will fall within this range.



Margin of error for a population mean

If μ_a and σ_a are the mean value and standard deviations of the simulation, we can assume that there is a 95% chance that a new simulation will give a mean value that is within $\mu_a - \varepsilon_a$ and $\mu_a + \varepsilon_a$, where

$$\varepsilon_a = 1.96 \frac{\sigma_a}{\sqrt{N_a}}$$

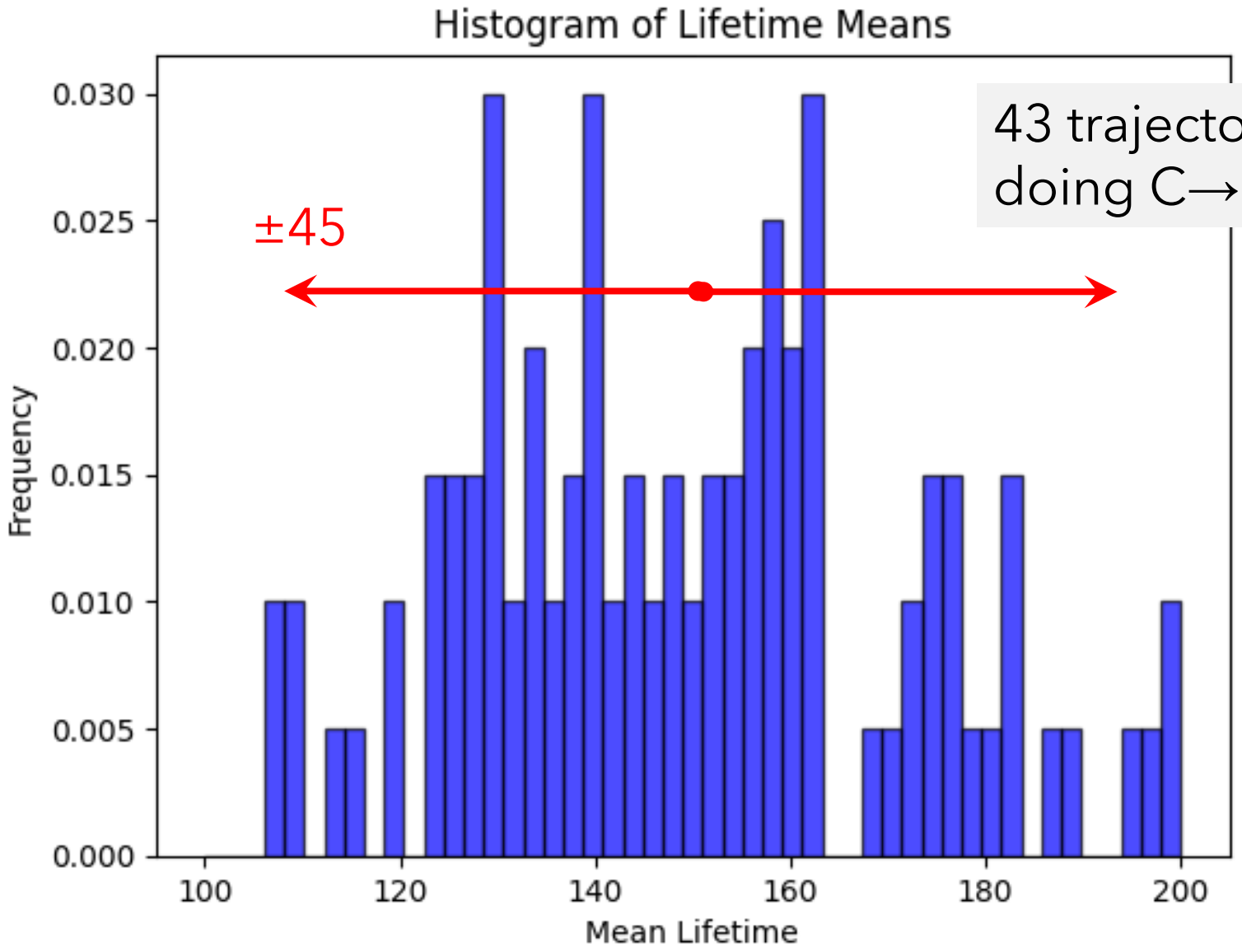
For the 43 trajectories isomerizing to B, the **mean value** of the isomerization time was 150 fs with a **standard deviation** of 150 fs

For 95% confidence, the margin of error of the mean C→B isomerization time is

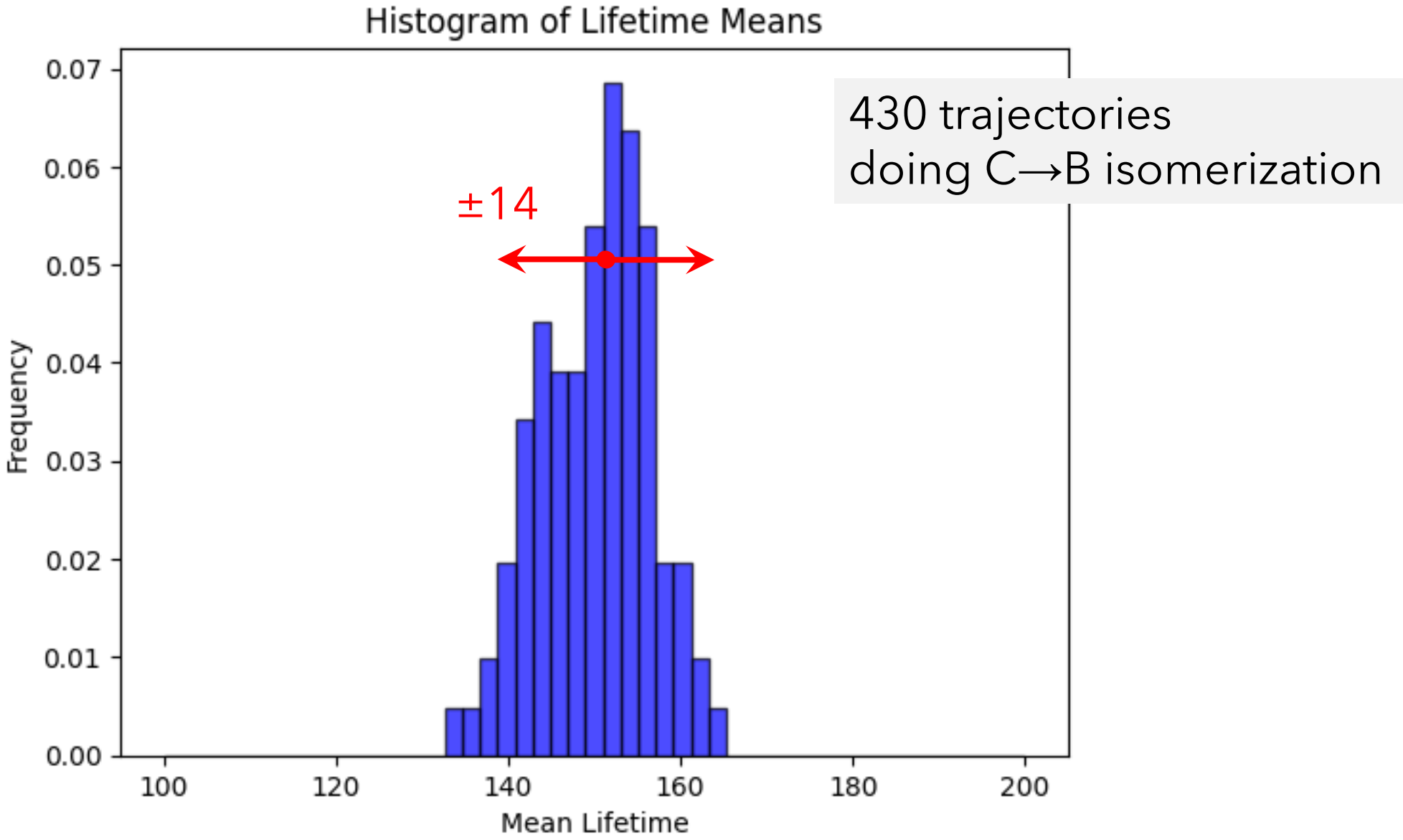
$$\varepsilon_{C\to B} = 1.96\frac{150}{\sqrt{43}} = 45$$

C→B Isomerization time:

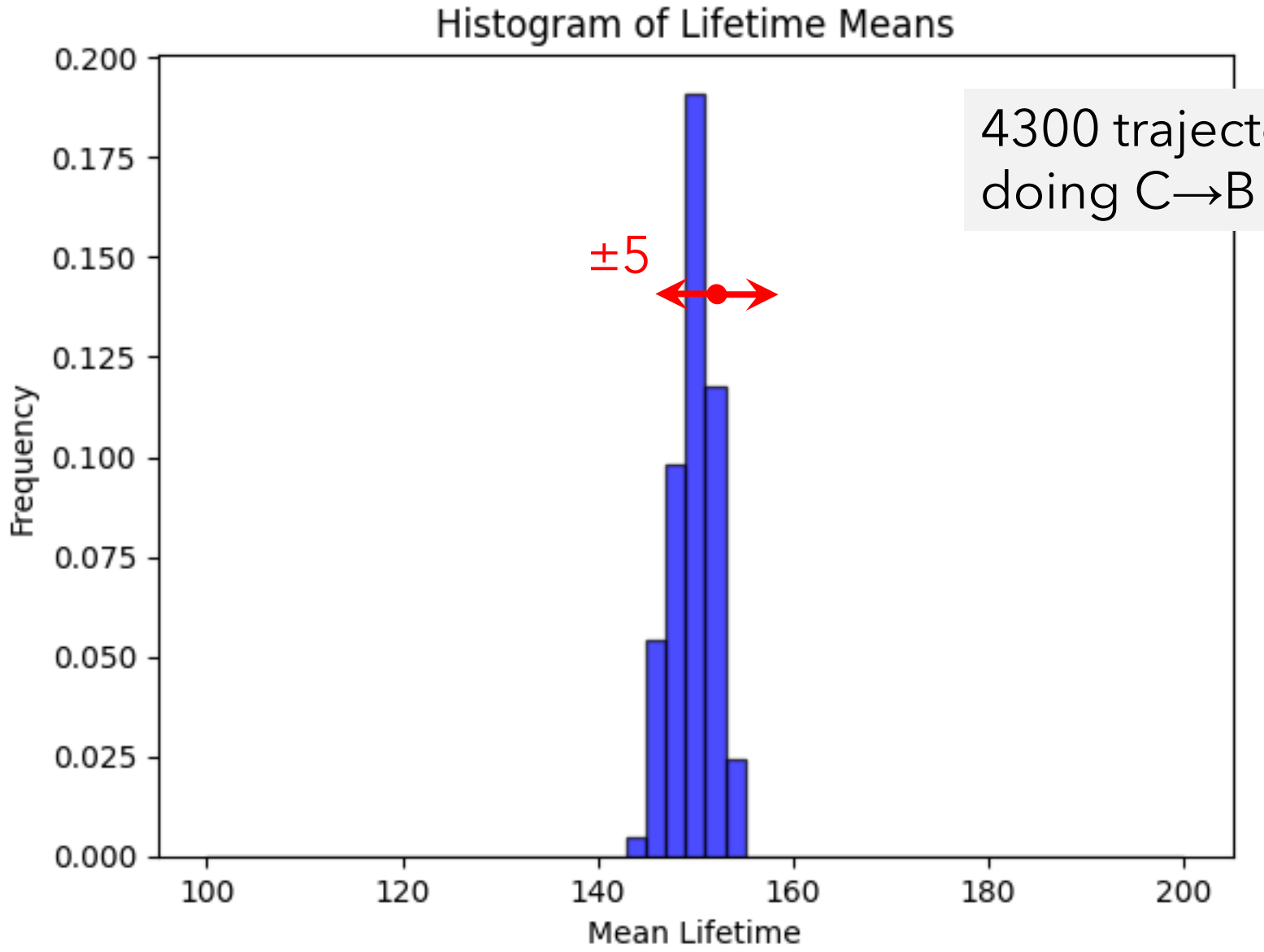
$$\tau_{C\to B} = 150 \pm 45 \text{ fs}$$



43 trajectories
doing C→B isomerization



430 trajectories
doing C→B isomerization



4300 trajectories
doing C→B isomerization

N = 100 trajectories

Channel	**Proportion of trajectories (%)**
No isomerization	5
Isomerization C→A	52
Isomerization C→B	43

Margin of error for a population proportion

$$\varepsilon_p = 1.96\sqrt{\frac{p(1-p)}{N}}$$

- p is the population proportion
- N is the population size

N = 100 trajectories

Channel	**Proportion of trajectories (%)**
No isomerization	5
Isomerization C→A	52
Isomerization C→B	43

For a confidence of 95%, the margin of error for "no isomerization" proportion is

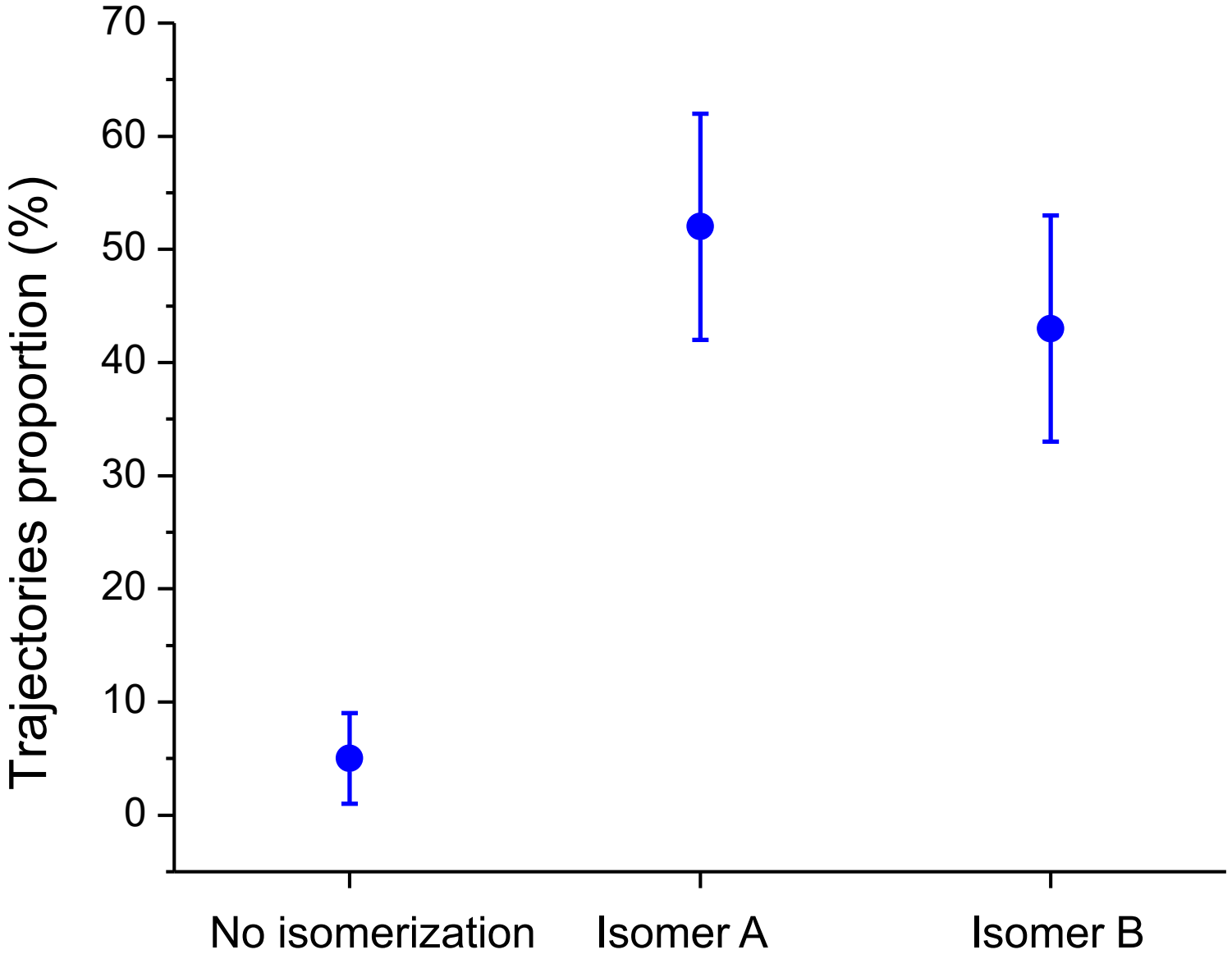
$$\varepsilon_p = 1.96\sqrt{\frac{p(1-p)}{N}} = 1.96\sqrt{\frac{0.05(1-0.05)}{100}} = 4\%$$

No isomerization is 5 ± 4 % for 95% confidence.

This means if we repeat the simulations 100 times, we are confident that the number of “no isomerization” should be between 1% and 9% in 95 of the simulations.

Channel	Proportion of trajectories (%)	Margin of error (%)
No isomerization	5	4
Isomer C→A	52	10
Isomer C→B	43	10

Which of the three channels is the most important?



We cannot tell that isomer A is more abundant than B.

Statistical errors in molecular dynamics: Bootstraping

The previous discussion assumed a normal distribution of populations.

Bootstrapping is an alternative to estimating statistical uncertainty if you don't want to make such an assumption

Biostatistics for Dummies, tinyurl.com/bstrapping

Bootstrapping:

Suppose you have *N* results and want to determine the margin of error.

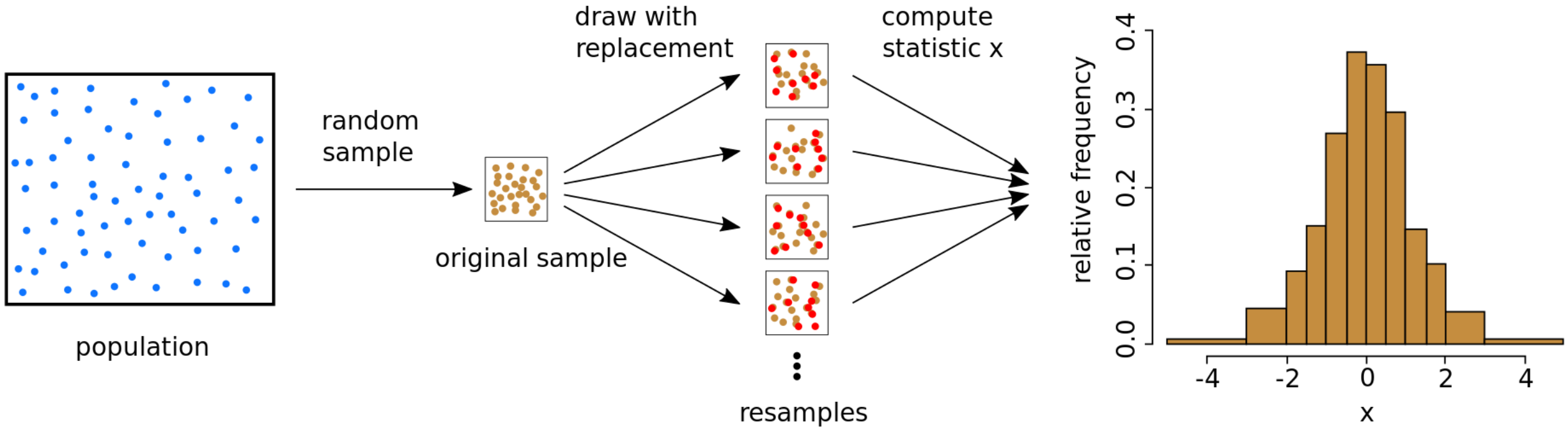
1. Compute the mean of the *N* results.

2. From these N results, random pick *N* values. You can pick the same value more than once.

3. Compute the mean of this new set of this resampling of *N* values.

4. Repeat the resampling many times, computing the mean for each one.

5. Compute the standard deviation and confidence interval for the mean values.



en.wikipedia.org/wiki/Bootstrapping_(statistics)

	Simulation
Traj 1	92
Traj 2	101
Traj 3	98
Traj 4	88
Traj 5	94
Traj 6	102
Traj 7	109
Traj 8	120
Traj 9	89
Traj 10	61
Traj 11	114
Traj 12	93
Traj 13	138
Traj 14	89
Traj 15	108
Traj 16	113
Traj 17	115
Traj 18	98
Traj 19	105
Traj 20	90
Mean	**101**
Std dev	16
95%SE	**7**

Result:

- Gaussian
  101 ± 7

	Simulation	Resamp 1
Traj 1	92	89
Traj 2	101	108
Traj 3	98	138
Traj 4	88	114
Traj 5	94	98
Traj 6	102	113
Traj 7	109	108
Traj 8	120	94
Traj 9	89	61
Traj 10	61	138
Traj 11	114	108
Traj 12	93	90
Traj 13	138	109
Traj 14	89	114
Traj 15	108	113
Traj 16	113	93
Traj 17	115	93
Traj 18	98	89
Traj 19	105	120
Traj 20	90	138
Mean	**101**	106
Std dev	16	
95%SE	**7**	

Result:

- Gaussian
  101 ± 7

	Simulation	Resamp 1	Resamp 2
Traj 1	92	89	109
Traj 2	101	108	90
Traj 3	98	138	109
Traj 4	88	114	115
Traj 5	94	98	98
Traj 6	102	113	108
Traj 7	109	108	108
Traj 8	120	94	89
Traj 9	89	61	98
Traj 10	61	138	138
Traj 11	114	108	92
Traj 12	93	90	108
Traj 13	138	109	92
Traj 14	89	114	93
Traj 15	108	113	89
Traj 16	113	93	105
Traj 17	115	93	113
Traj 18	98	89	98
Traj 19	105	120	88
Traj 20	90	138	109
Mean	**101**	106	102
Std dev	16		
95%SE	**7**		

Result:

- Gaussian
  101 ± 7

	Simulation	Resamp 1	Resamp 2	...	Resamp 9998	Resamp 9999	Resamp 10000
Traj 1	92	89	109		89	109	92
Traj 2	101	108	90		61	108	102
Traj 3	98	138	109		61	98	115
Traj 4	88	114	115		113	115	98
Traj 5	94	98	98		114	93	94
Traj 6	102	113	108		105	101	105
Traj 7	109	108	108		94	114	92
Traj 8	120	94	89		61	98	98
Traj 9	89	61	98		89	98	113
Traj 10	61	138	138		115	138	101
Traj 11	114	108	92		120	98	98
Traj 12	93	90	108		92	101	109
Traj 13	138	109	92		90	98	101
Traj 14	89	114	93		89	109	88
Traj 15	108	113	89		92	105	98
Traj 16	113	93	105		101	98	113
Traj 17	115	93	113		92	93	90
Traj 18	98	89	98		108	120	61
Traj 19	105	120	88		89	89	94
Traj 20	90	138	109		113	109	93
							95%SE
Mean	**101**	106	102	...	94	105	98 **6**
Std dev	16						
95%SE	**7**						

Result:

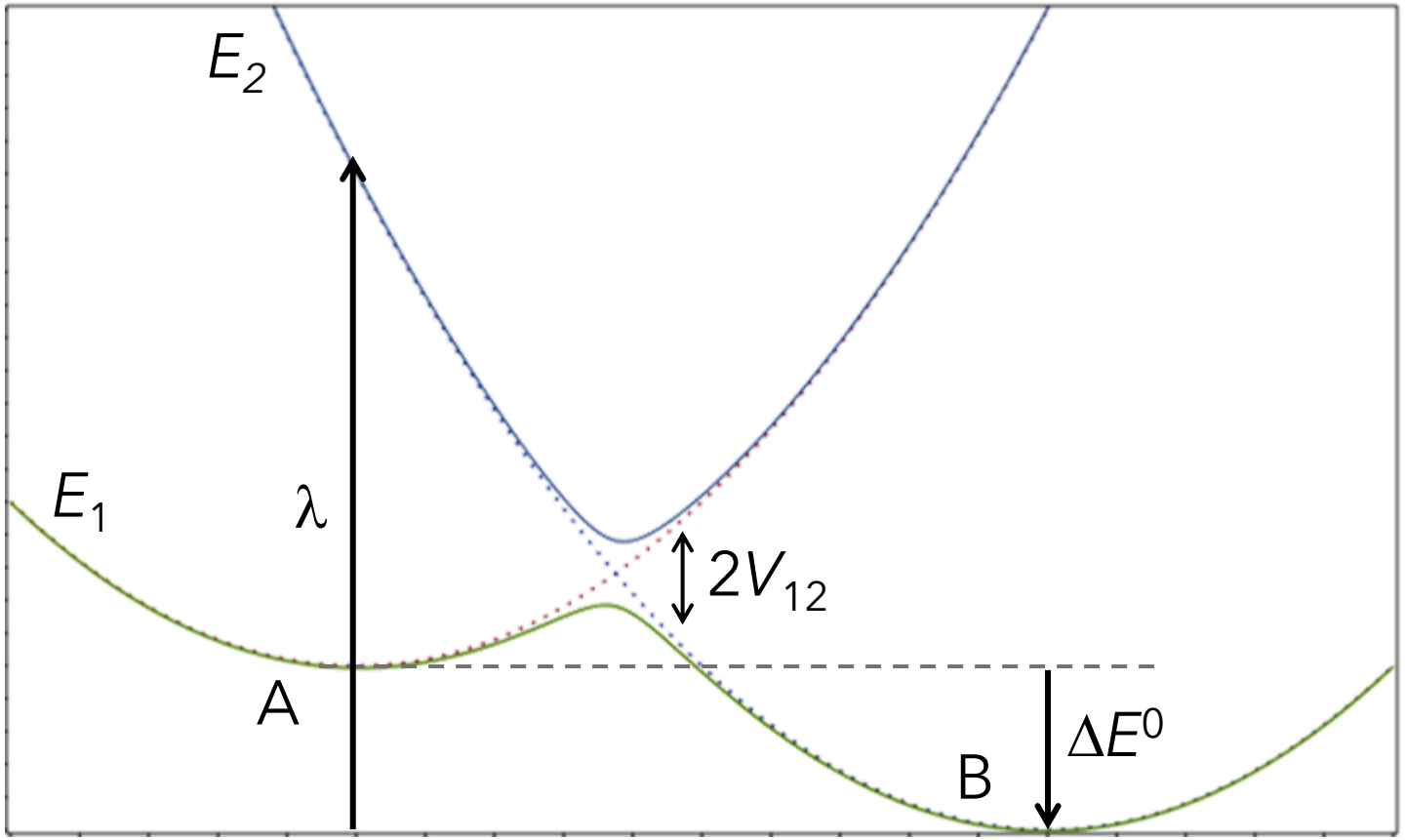
- Gaussian
  101 ± 7
- Bootstrapping
  101 ± 6

```
def bootstrap_stats(data, num_samples=1000, confidence_level=0.95):
    boot_samples = np.random.choice(data, size=(num_samples, len(data)), replace=True)

    # Calculate mean for each bootstrap sample
    means = np.mean(boot_samples, axis=1)

    # Calculate standard deviation of the mean
    std_dev_mean = np.std(means)

    # Calculate 95% confidence intervals for mean
    ci_mean = np.percentile(means, [(1-confidence_level)/2*100,(1+confidence_level)/2*100])

    # Calculate mean of the original data
    original_mean = np.mean(data)

    # Print results
    print("Original Mean:", original_mean)
    print("Standard Deviation of Mean:", std_dev_mean)
    print(f"95% ME of Bootstrapped Mean: {(ci_mean[1]-ci_mean[0])/2}")

# Example usage:
times = [61,88,89,89,90,92,93,94,98,98,101,102,105,108,109,113,114,115,120,138]
result = bootstrap_stats(times)
```

Monte Carlo Sampling of Errors

Suppose you want to compute Marcus rate for the reaction A → B

$$W_{AB} = \frac{2\pi}{\hbar}\left|V_{12}\right|^2 \frac{1}{\sqrt{4\pi\lambda k_B T}} \exp\left(-\frac{\left(\lambda + \Delta E^0\right)^2}{4\lambda k_B T}\right)$$



- Adiabatic energy gap
  ΔE^0 = -0.5 ± 0.1 eV
- Reorganization energy
  λ = 2.0 ± 0.1 eV
- Diabatic coupling
  V_{12} = 0.05 ± 0.01 eV
- Temperature
  T = 300 ± 1 K

The uncertainties are 95% CI

Suppose you want to compute Marcus rate for the reaction A → B

$$W_{AB} = \frac{2\pi}{\hbar}\left|V_{12}\right|^2 \frac{1}{\sqrt{4\pi\lambda k_B T}} \exp\left(-\frac{\left(\lambda + \Delta E^0\right)^2}{4\lambda k_B T}\right)$$

- Boltzmann constant
  $k_B = 8.617333262 \times 10^{-5}$ eV/K
- Reduced Planck constant
  $\hbar = 6.582119569 \times 10^{-16}$ eV.s

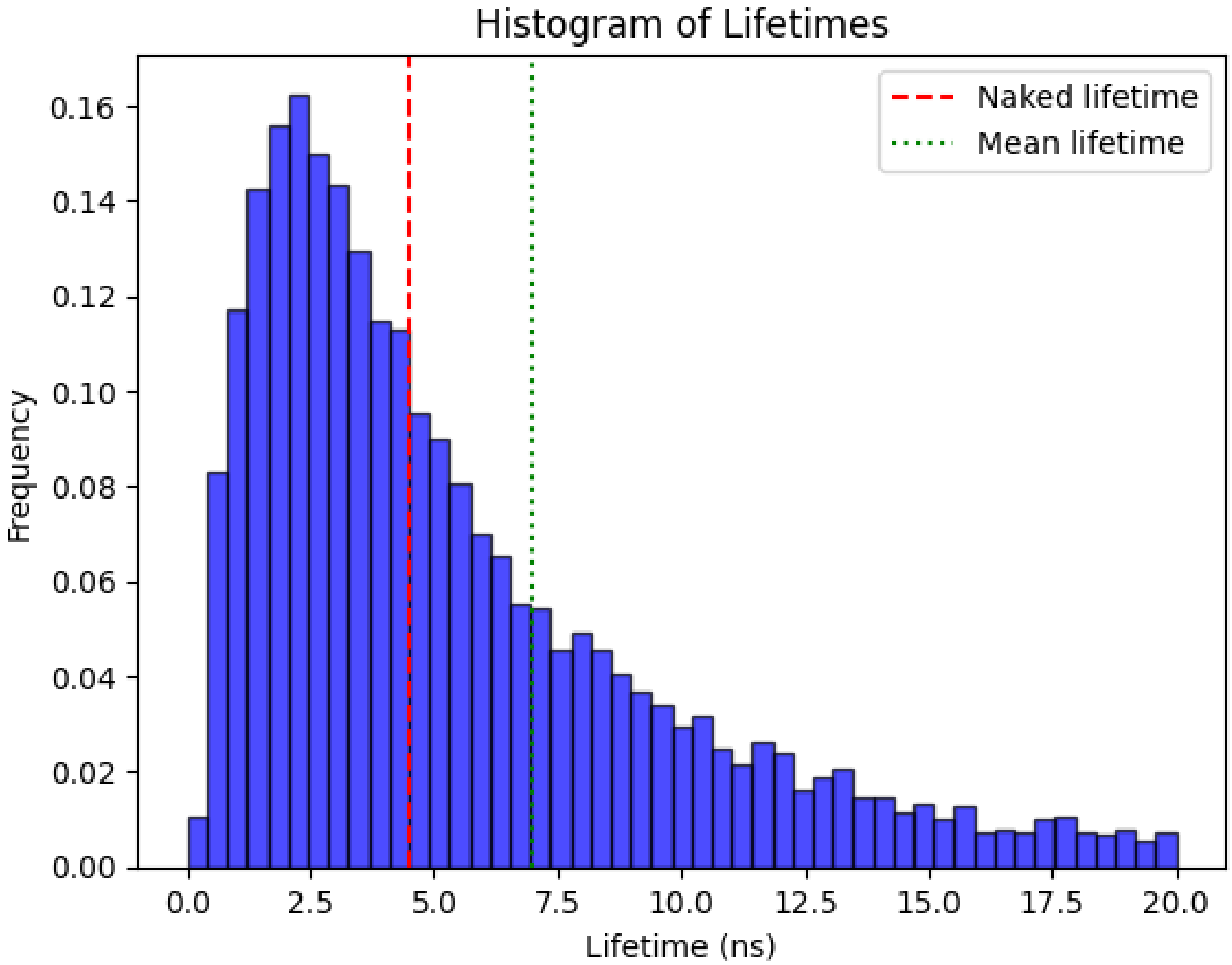
Naked results:
Marcus rate = 223 ns^{-1}
Marcus lifetime = 4.5 ps

- Adiabatic energy gap
  $\Delta E^0 = -0.5 \pm 0.1$ eV
- Reorganization energy
  $\lambda = 2.0 \pm 0.1$ eV
- Diabatic coupling
  $V_{12} = 0.05 \pm 0.01$ eV
- Temperature
  $T = 300 \pm 1$ K

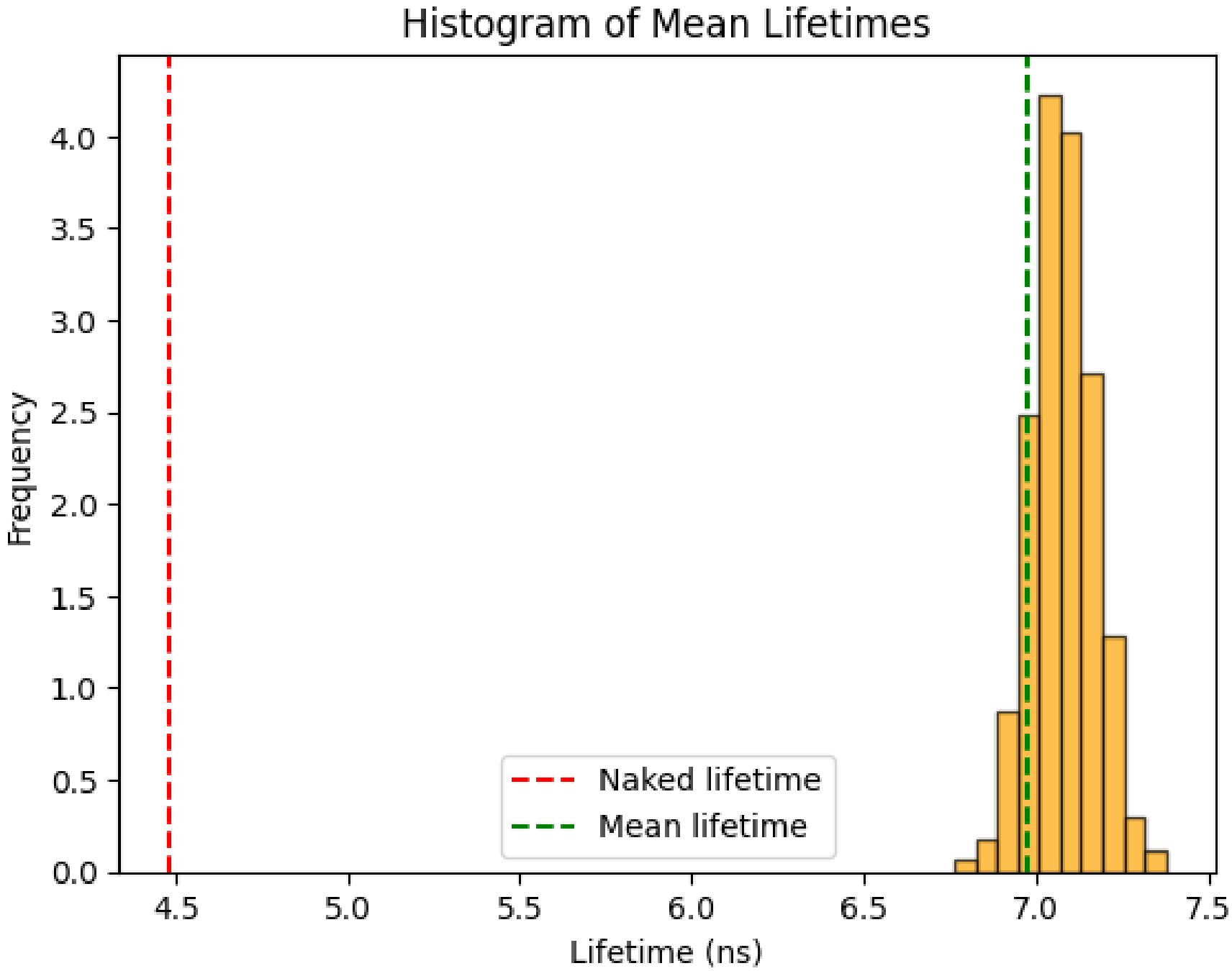
The uncertainties are 95% CI

What about the uncertainties?

```
def marcus_rate(DE,RE,V,TEMP):                          # Marcus rate
  W = 2*math.pi/hbar*
      1/math.sqrt(4*math.pi*RE*kB*TEMP)*
      math.exp(-(RE+DE)**2/(4*RE*kB*TEMP))

  return W

Ntot = 100000                                           # Number of points
Z    = 1.96                                             # 95% CI
all_values = []
for i in range (1, Ntot+1):                             # Loop over random values
  DE   = np.random.normal(loc=DE0,scale=DDE0/Z)
  RE   = np.random.normal(loc=lamb,scale=Dlamb/Z)
  V    = np.random.normal(loc=V12,scale=DV12/Z)
  TEMP = np.random.normal(loc=T,scale=DT/Z)

  time = 1/marcus_rate(DE,RE,V,TEMP)                    # Lifetime

  # store
  all_values.append(time)
```



Distribution of 10,000 random samples



Distribution of 1000 mean values.

To know more:

Margins of error

- tinyurl.com/blognormal

To truly understand Gaussian errors: central limit theorem

- www.youtube.com/watch?v=zeJD6dqJ5lo

Statistical errors in MD

- Schiferl; Wallace. *J Chem Phys* **1985**, *83*, 5203

The most important statistical ideas in the last 50 years

- Gelman; Vehtari. *J Am Stat Assoc* **2021**, *116*, 2087