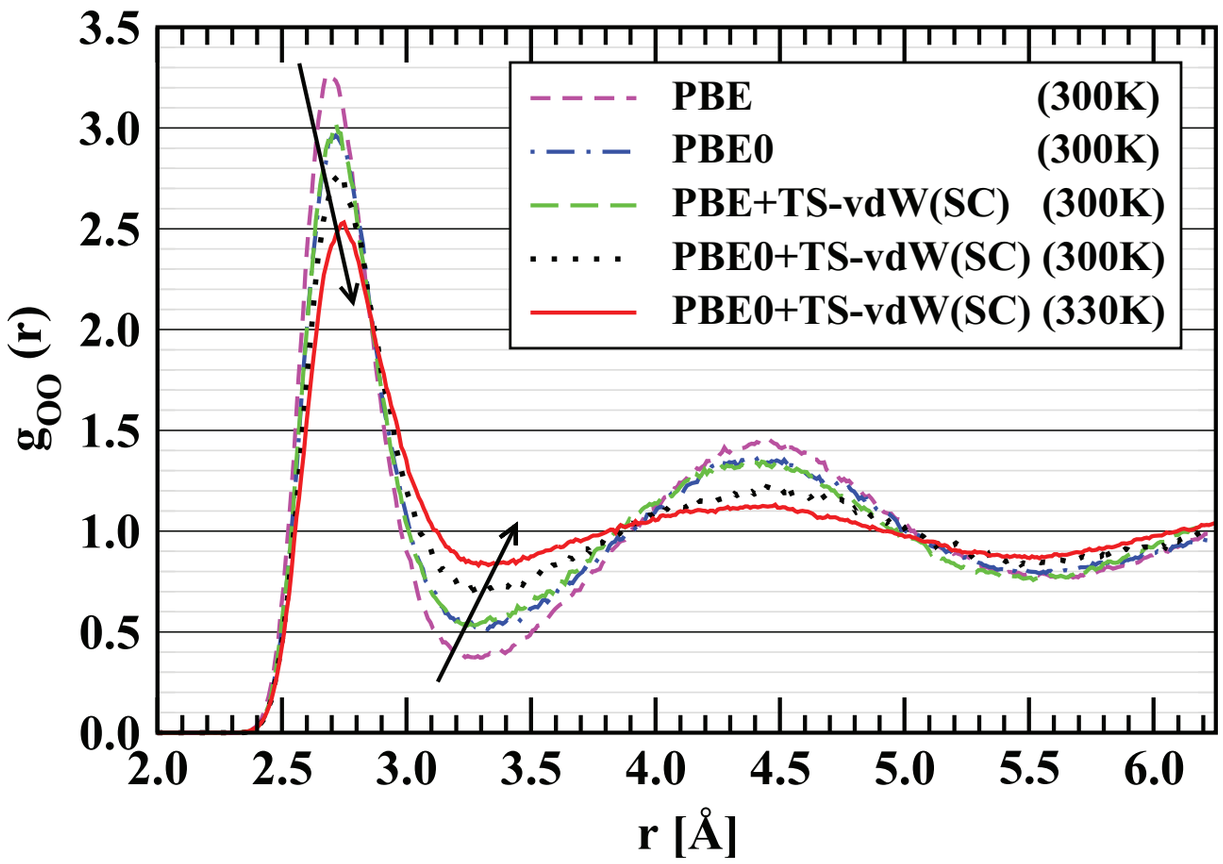
TCCM Winter School LTTC 2025
Exploration of Potential and Free Energy Surfaces

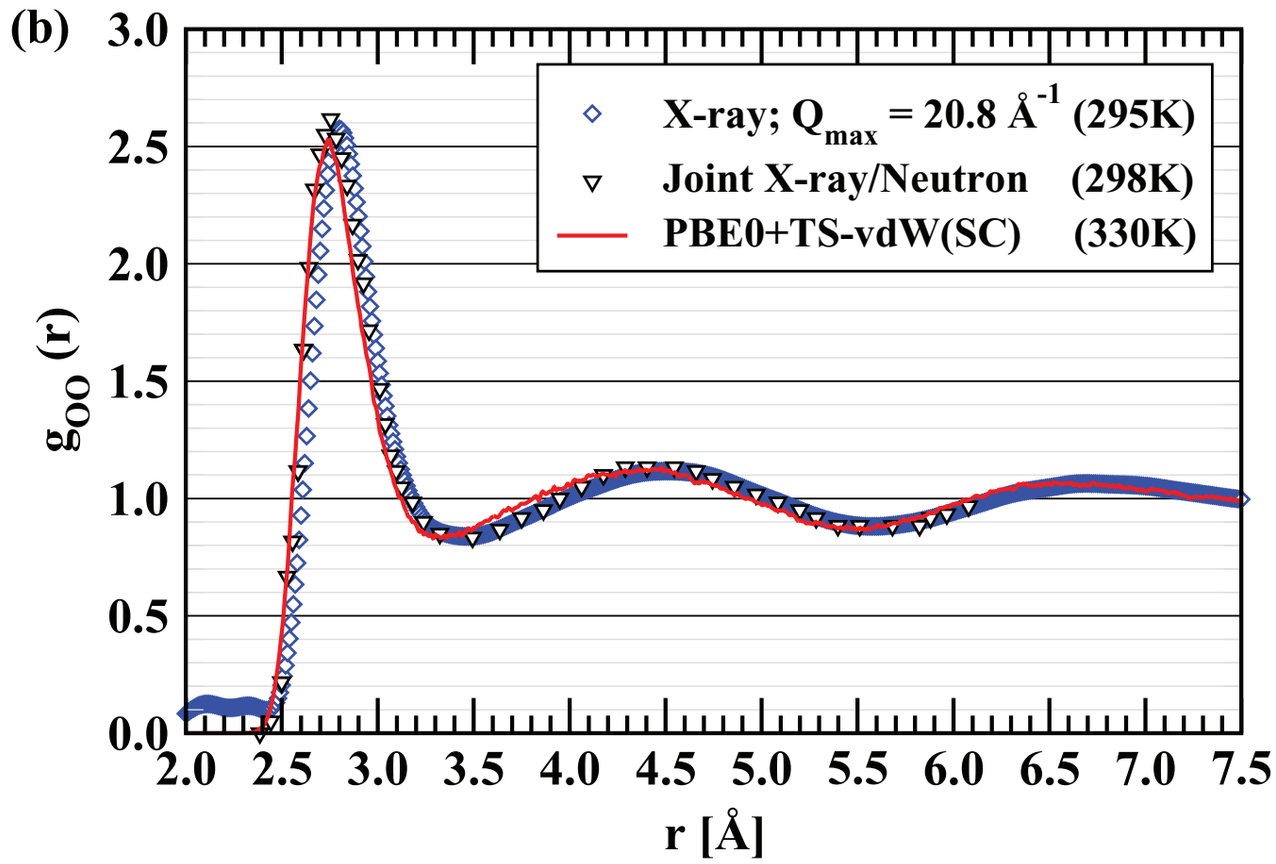
Jérôme Cuny

Laboratoire de Chimie et Physique Quantiques
*UPS, **Bureau 210, Bât. 3R1b4**, 118 route de Narbonne*
31062 Toulouse Cedex 09, France
Tel : +33 (0) 5 61 55 68 36
jerome.cuny@irsamc.ups-tlse.fr

Molecular Dynamics Simulation of Liquid Water



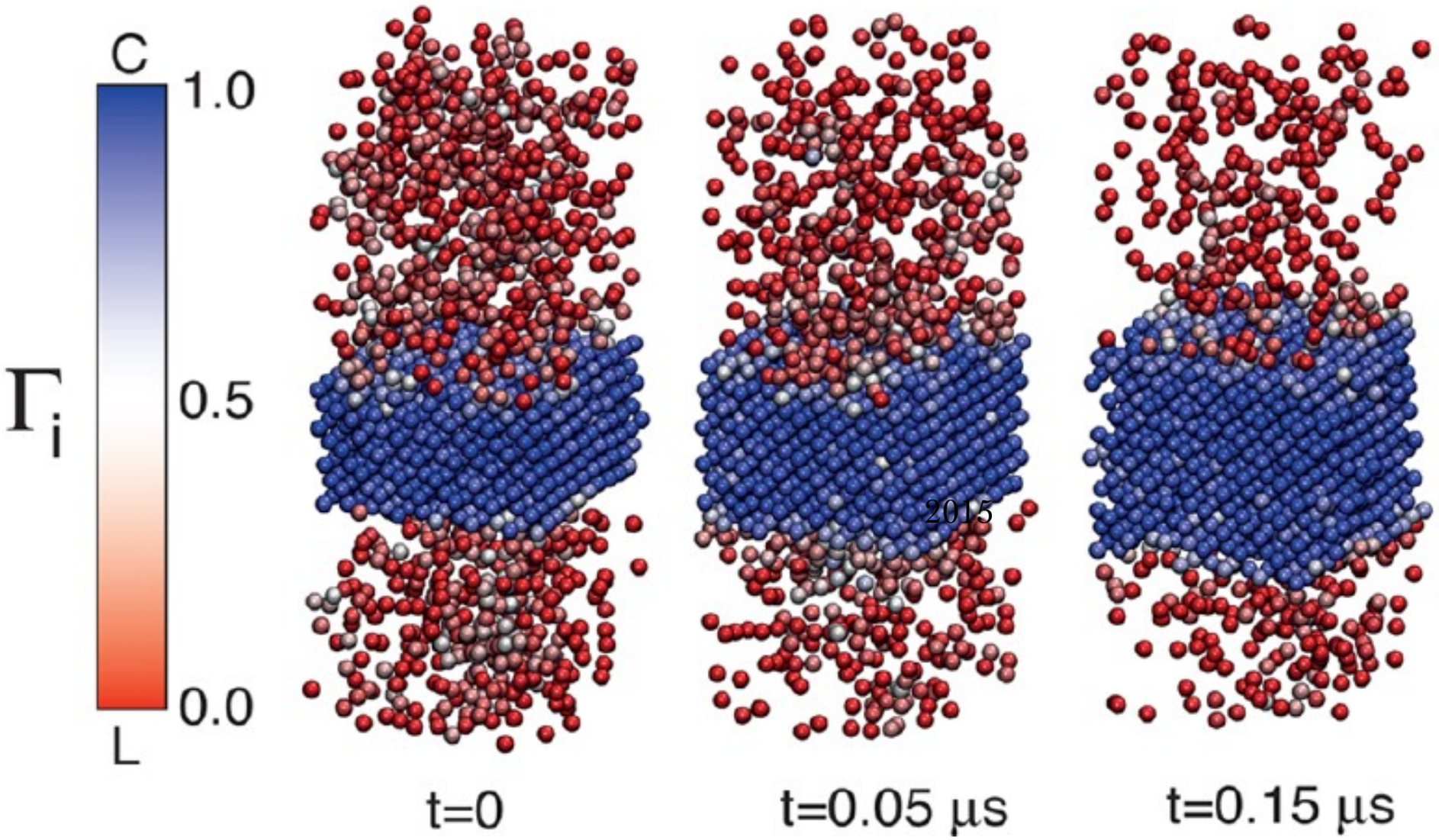
The oxygen-oxygen radial distribution functions $g_{OO}(r)$ of liquid water obtained from theory (via the DFT-based AIMD simulations performed in this work). The arrows indicate systematic shifts in the main peak positions and intensities of the computed distributions with improvement of the under- lying exchange-correlation functional.



The oxygen-oxygen radial distribution functions, $g_{OO}(r)$, of liquid water obtained from theory (via the DFT-based AIMD simulations performed in this work) and various scattering experiments.

R. A. DiStasio Jr., B. Santra, Z. Li, X. Wu, R. Car *J. Chem. Phys.* **141**, 084502 (2014)

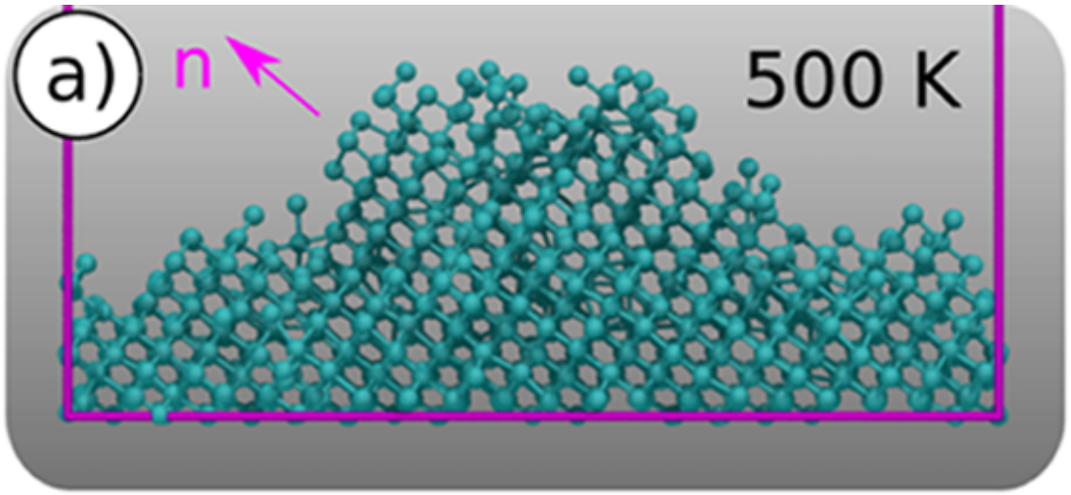
Crystal Growth of Urea at Interfacres

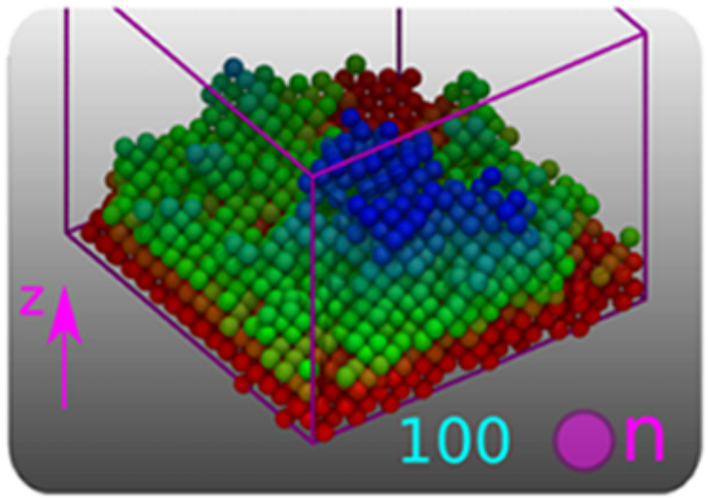


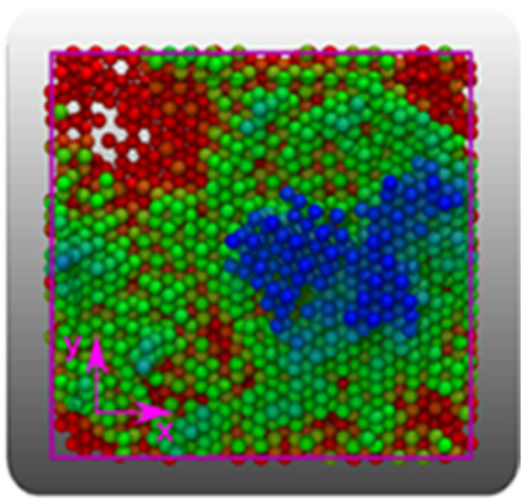
Snapshots from the simulation B at different timesteps. For the sake of clarity water molecules are omitted, while urea molecules are depicted as spheres centered on the carbon atom and colored in function of their degree of crystallinity Γ_i.

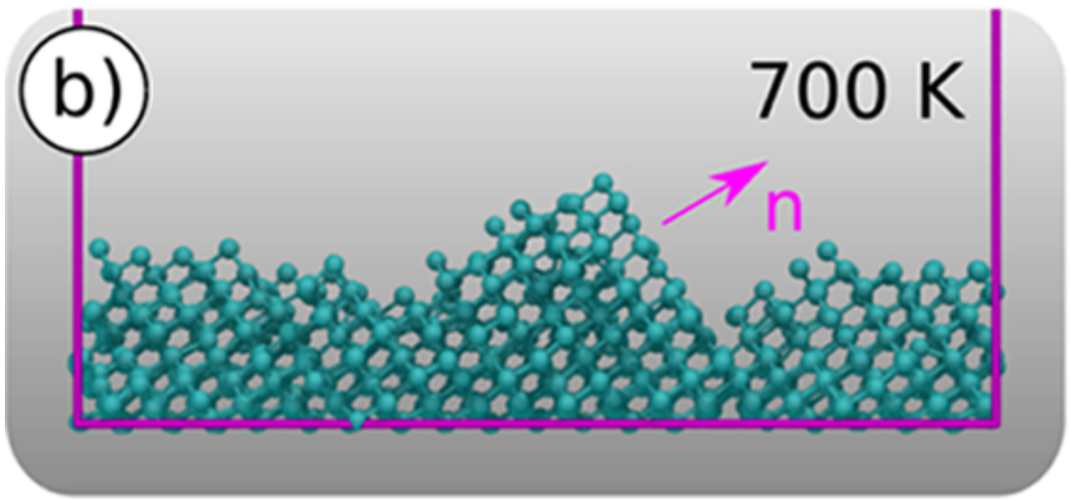
M. Salvalaglio, T. Vetter, F. Giberti, M. Mazzotti, M. Parrinello *J. Am. Chem. Soc.* **134**, 17221–17233 (2012)

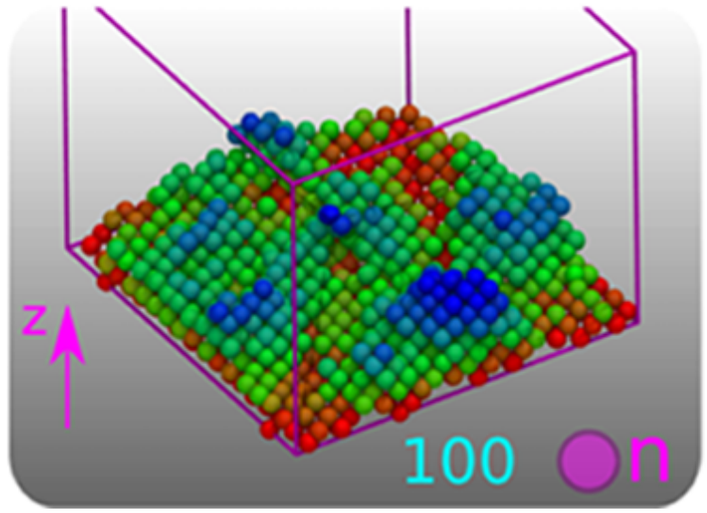
Heterogeneous Crystallization of the Phase Change Material Ge Te

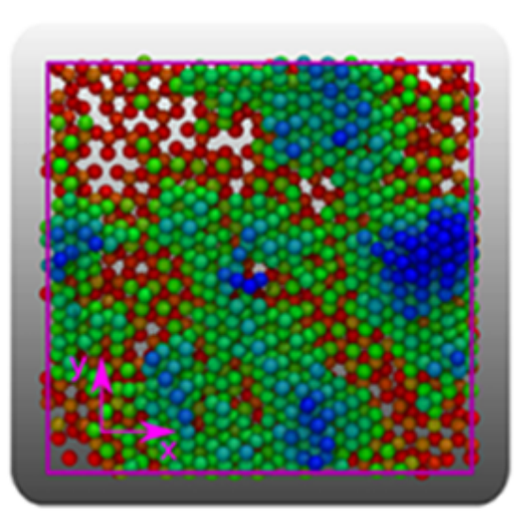






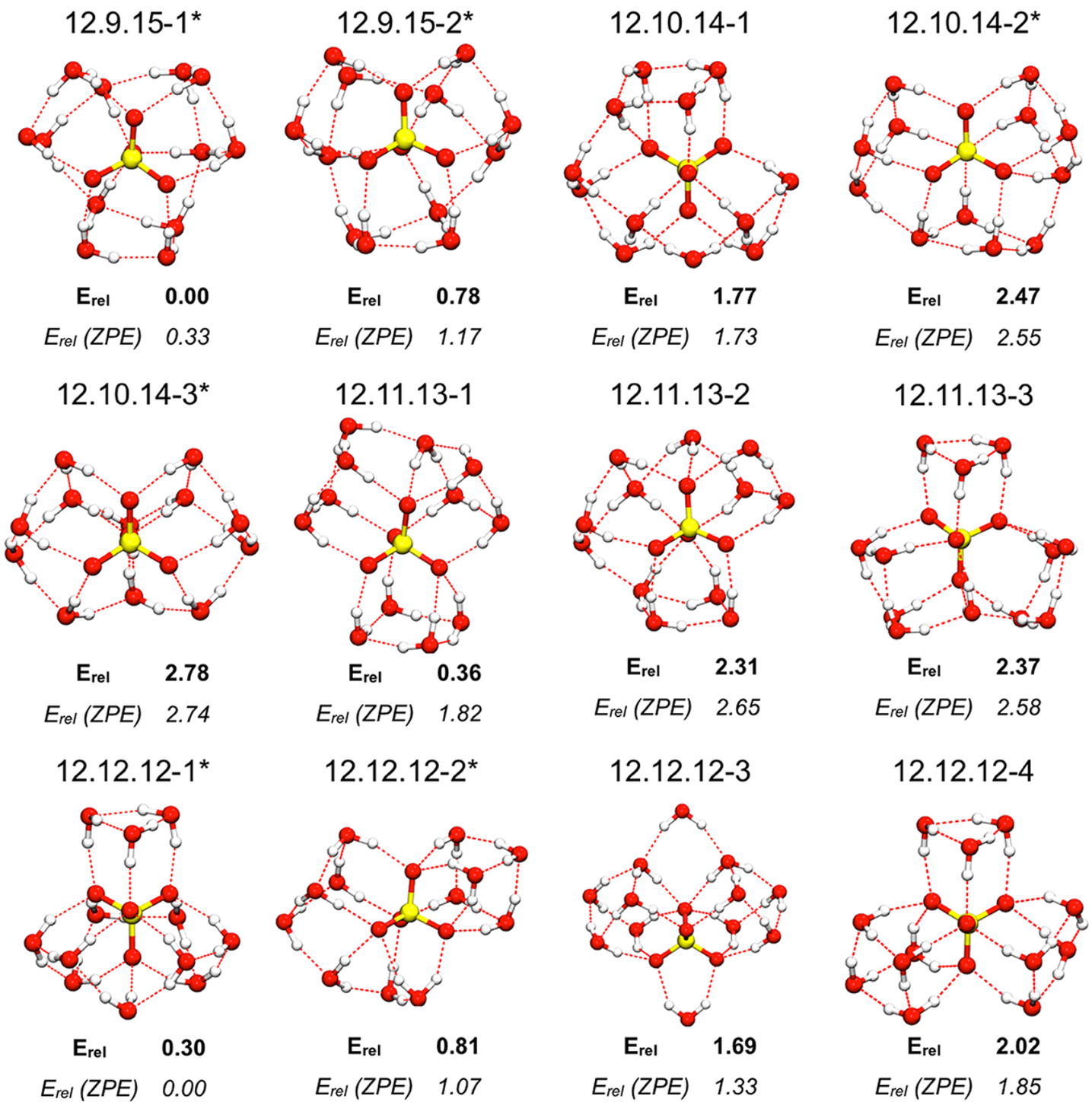






(a) Snapshot of the (111) surface morphology during crystal growth at 500 K. Only crystalline atoms belonging to the growing (bottom) surface are depicted. The same configuration is shown projected onto the xz plane (left), along the normal n at one of the {100} planes formed (middle), and onto the xy plane (right). Atoms are colored according to their z coordinate in middle and right panels. (b) Same as above at 700 K.

G. C. Sosso, M. Salvalaglio, J. Behler, M. Bernasconi, M. Parrinello *J. Phys. Chem. C* **119**, 6428–6434 (2015)



Selected low-energy isomers of $(H_2O)_{12}SO_4^{2-}$ and associated relative energies (in kcal·mol^{-1}) without (bold) and with (italic) ZPE corrections.

K. A. Korchagina, A. Simon, M. Rapacioli, F. Spiegelman, J. Cuny *J. Phys. Chem. A* **120**, 9089–9100 (2016)

Summary

Integrating the Newton's Equations of Motion

- In ***classical molecular dynamics***, knowing atomic positions at a given time t, one wants to know what are the positions at time $t+\partial t$.
  - ⇒ Here ∂t is a small amount of time
- By repeating this one step *n times*, one gets a dynamical description of the system of particles over a *time* $n\times\partial t$.

Integrating the Newton's Equations of Motion

- ✓ In ***classical molecular dynamics***, knowing atomic positions at a given time t, one wants to know what are the positions at time $t+\partial t$.
  - ✤ ⇒ Here ∂t is a small amount of time
- ✓ By repeating this one step *n times*, one gets a dynamical description of the system of particles over a *time* $n \times \partial t$.
- ✓ To obtain $r(t+\partial t)$ from $r(t)$, one needs an ***integrator***, *i.e.* a numerical method to integrate/ propagate the Newton's Equations of motions.
- ✓ Various integrator exist:
  - ✤ Gear predictor-corrector at various orders
  - ✤ Runge-Kutta (RK1, RK2, RK4)
  - ✤ Verlet algorithms: simple Verlet, leap-frog Verlet, velocity Verlet

Integrating the Newton's Equations of Motion

- ✓ In ***classical molecular dynamics***, knowing atomic positions at a given time t, one wants to know what are the positions at time $t+\partial t$.
  - ✤ ⇒ Here ∂t is a small amount of time
- ✓ By repeating this one step *n times*, one gets a dynamical description of the system of particles over a *time* $n\times\partial t$.
- ✓ To obtain $r(t+\partial t)$ from $r(t)$, one needs an ***integrator***, *i.e.* a numerical method to integrate/ propagate the Newton's Equations of motions.
- ✓ Various integrator exist:
  - ✤ Gear predictor-corrector at various orders
  - ✤ Runge-Kutta (RK1, RK2, RK4)
  - ✤ Verlet algorithms: simple Verlet, leap-frog Verlet, velocity Verlet
- ✓ We use ***Velocity-Verlet*** as it is simple, not time consuming and it provides good numerical stability and time-reversal properties.

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t=0), velocities v(t=0) and accelerations a(t=0)

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t=0), velocities v(t=0) and accelerations a(t=0)

$$\downarrow$$

$$v(0+1/2dt) = v(0) + 1/2\ a(0)\ dt$$

$$r(0+dt) = r(0) + v(0+1/2dt)\ dt$$

get a(0+dt)

$$v(0+dt) = v(0+1/2dt) + 1/2\ a(0+dt)\ dt$$

Velocity-Verlet
integrator

dt = timestep

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t=0), velocities v(t=0) and accelerations a(t=0)

↓

v(0+1/2dt) = v(0) + 1/2 a(0) dt

r(0+dt) = r(0) + v(0+1/2dt) dt

get a(0+dt)

v(0+dt) = v(0+1/2dt) + 1/2 a(0+dt) dt

Velocity-Verlet
integrator

dt = timestep

↓

You get positions r(t=dt), velocities v(t=dt) and accelerations a(t=dt)

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t=dt), velocities v(t=dt) and accelerations a(t=dt)

v(0+1/2dt) = v(0) + 1/2 a(0) dt

r(0+dt) = r(0) + v(0+1/2dt) dt

get a(0+dt)

v(0+dt) = v(0+1/2dt) + 1/2 a(0+dt) dt

You get positions r(t=dt), velocities v(t=dt) and accelerations a(t=dt)

Velocity-Verlet
integrator

dt = timestep

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t=dt), velocities v(t=dt) and accelerations a(t=dt)

v(dt+1/2dt) = v(dt) + 1/2 a(dt) dt

r(2dt) = r(dt) + v(dt+1/2dt) dt

get a(2dt)

v(2dt) = v(dt+1/2dt) + 1/2 a(2dt) dt

You get positions r(t=dt), velocities v(t=dt) and accelerations a(t=dt)

Velocity-Verlet
integrator

dt = timestep

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t=dt), velocities v(t=dt) and accelerations a(t=dt)

$$\mathbf{v(dt+1/2dt) = v(dt) + 1/2\ a(dt)\ dt}$$

$$\mathbf{r(2dt) = r(dt) + v(dt+1/2dt)\ dt}$$

get a(2dt)

$$\mathbf{v(2dt) = v(dt+1/2dt) + 1/2\ a(2dt)\ dt}$$

You get positions r(t=2dt), velocities v(t=2dt) and accelerations a(t=2dt)

Velocity-Verlet
integrator

dt = timestep

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

$$\mathbf{v(t+1/2dt) = v(t) + 1/2\ a(t)\ dt}$$

$$\mathbf{r(t+dt) = r(t) + v(t+1/2dt)\ dt}$$

get a(t+dt)

$$\mathbf{v(t+dt) = v(t+1/2dt) + 1/2\ a(t+dt)\ dt}$$

You get positions r(t+dt), velocities v(t+dt) and accelerations a(t+dt)

Velocity-Verlet
integrator

dt = timestep

Integrating the Newton's Equations of Motion

Newton equations of motion for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

$$v(t+1/2dt) = v(t) + 1/2\ a(t)\ dt$$

$$r(t+dt) = r(t) + v(t+1/2dt)\ dt$$

get a(t+dt)

$$v(t+dt) = v(t+1/2dt) + 1/2\ a(t+dt)\ dt$$

You get positions r(t+dt), velocities v(t+dt) and accelerations a(t+dt)

Velocity-Verlet integrator

dt = timestep

WHERE IS QUANTUM MECHANICS HERE ??

Integrating the Newton's Equations of Motion

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

$$v(t+1/2dt) = v(t) + 1/2\ a(t)\ dt$$

$$r(t+dt) = r(t) + v(t+1/2dt)\ dt$$

get a(t+dt)

$$v(t+dt) = v(t+1/2dt) + 1/2\ a(t+dt)\ dt$$

You get positions r(t+dt), velocities v(t+dt) and accelerations a(t+dt)

Velocity-Verlet
integrator

dt = timestep

Forces in Classical Molecular Dynamics

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

↓

$$\mathbf{v(t+1/2dt) = v(t) + 1/2\ a(t)\ dt}$$

$$\mathbf{r(t+dt) = r(t) + v(t+1/2dt)\ dt}$$

$$\mathbf{get\ a(t+dt)}$$

$$\mathbf{v(t+dt) = v(t+1/2dt) + 1/2\ a(t+dt)\ dt}$$

Velocity-Verlet
integrator

dt = timestep

In **Molecular Dynamics**, one needs to calculate the forces. You can use:

Forces in Classical Molecular Dynamics

Newton equations of motion
for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

$$\downarrow$$

v(t+1/2dt) = v(t) + 1/2 a(t) dt

r(t+dt) = r(t) + v(t+1/2dt) dt

get a(t+dt)

v(t+dt) = v(t+1/2dt) + 1/2 a(t+dt) dt

Velocity-Verlet
integrator

dt = timestep

In **Molecular Dynamics**, one needs to calculate the forces. You can use:

- ✓ Force fields (= molecular mechanics)
- ✓ Quantum chemical methods (DFT, DFTB or others) (= AIMD means ab initio molecular dynamics)

Forces in Classical Molecular Dynamics

- In ***classical molecular dynamics*** one look at the forces acting on a nucleus (or a particle) originating from all the other particles of the system.

- The force acting on the i^{th} particle along the direction α (= x, y or z) is defined as :

$$F_{i,\alpha} = -\frac{\partial E}{\partial R_{i,\alpha}}$$

Forces in Classical Molecular Dynamics

- In ***classical molecular dynamics*** one look at the forces acting on a nucleus (or a particle) originating from all the other particles of the system.

- The force acting on the i^{th} particle along the direction α (= x, y or z) is defined as :

$$F_{i,\alpha} = -\frac{\partial E}{\partial R_{i,\alpha}}$$

- Today, we will use a simple Lennard-Jones potential:
  - It is a pair potential.
  - The total energy can be written as the sum over all atomic pairs of the interaction energy defined as:

$$E_{i,j} = 4 * eps * \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right]$$

Summary

The Ingredients of a Simple MD Code

Inputing data otherwise than with read(*,*)

- ✓ The simplest of MD code requires the following ingredients:
  - ✤ The **timestep** ⇒ **taken as an input data**
  - ✤ Maximum number of MD steps in the simulation ⇒ **taken as an input data**
  - ✤ **Initial coordinates** (⇒ control initial forces) ⇒ **read from an input file (molden format)**

The Ingredients of a Simple MD Code

Inputing data otherwise than with read(*,*)

- ✓ The simplest of MD code requires the following ingredients:
  - ✤ The **timestep** ⇒ **taken as an input data**
  - ✤ Maximum number of MD steps in the simulation ⇒ **taken as an input data**
  - ✤ **Initial coordinates** (⇒ control initial forces) ⇒ **read from an input file (molden format)**
- ✓ Simple **Fortran Procedure** for inputing data uses the COMMAND_ARGUMENT_COUNT fortran command:
  - ✤ **COMMAND_ARGUMENT_COUNT** returns the number of arguments passed on the command line when the containing program is invoked.

The Ingredients of a Simple MD Code

Inputing data otherwise than with read(*,*)

- ✓ The simplest of MD code requires the following ingredients:
  - ✤ The **timestep** ⇒ **taken as an input data**
  - ✤ Maximum number of MD steps in the simulation ⇒ **taken as an input data**
  - ✤ **Initial coordinates** (⇒ control initial forces) ⇒ **read from an input file (molden format)**
- ✓ Simple **Fortran Procedure** for inputing data uses the COMMAND_ARGUMENT_COUNT fortran command:
  - ✤ **COMMAND_ARGUMENT_COUNT** returns the number of arguments passed on the command line when the containing program is invoked.
  - ✤ **Simple example**:

```
      program test_command_argument_count
integer :: count
count = command_argument_count()
print *, count
end program test_command_argument_count
```

The Ingredients of a Simple MD Code

Inputing data otherwise than with read(*,*)

- ✓ The simplest of MD code requires the following ingredients:
  - ✤ The **timestep** ⇒ **taken as an input data**
  - ✤ Maximum number of MD steps in the simulation ⇒ **taken as an input data**
  - ✤ **Initial coordinates** (⇒ control initial forces) ⇒ **read from an input file (molden format)**
- ✓ Simple **Fortran Procedure** for inputing data:

```
if (COMMAND_ARGUMENT_COUNT().eq.0) then
        write(*,*) "This a kind of Manual for your program!"
        write(*,*) "OPTIONS: "
        write(*,*) " -FILE            Filename (default is input.xyz)"
        write(*,*) " -TIMESTEP        MD Timestep"
        write(*,*) " -MAX_MD_STEPS    Maximum number of MD steps (default is 100)"
        stop
endif

do i = 1, COMMAND_ARGUMENT_COUNT()

    call getarg(i,string_lec)
    if (index(string_lec,'-FILE').NE.0)then
      call getarg(i+1,filename)
    endif
    if (index(string_lec,'-TIMESTEP').NE.0)then
      call getarg(i+1,string_lec)
      read(string_lec,*) timestep
    endif
    if (index(string_lec,'-MAX_MD_STEPS').NE.0)then
      call getarg(i+1,string_lec)
      read(string_lec,*) str_md_out
    endif
enddo
```

Structure of the Code, an Overview of the Initialization

```
program my_first_MD_Code

use utilities ! Module allowing to perform different tasks

implicit none

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Variable Declaration                                       !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Inputing data with COMMAND_ARGUMENT_COUNT()                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Opening of output files : energies and structure           !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Reading of Initial positions in a file
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Generate Initial Velocities (can be all 0)                 !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Generate Initial Forces (Have to be none 0)                !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!Here you can start a MD trajectory !!!!!!!!!!!!!!!!!!!!!!
```

Structure of the Code, an Overview of the MD Loop

```
!!!!!!!!!!!!!!!! Master MD Loop !!!!!!!!!!!!!!!!
do m=1, max_md_Step

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Velocity Verlet
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

! Velocities to t+0.5dt
do i=1,nbr_atoms
  do j=1,3
    velocities(i,j)=velocities(i,j)+0.5*timestep*forces(i,j)/mass
  enddo
enddo

do i=1,nbr_atoms
  do j=1,3
    positions(i,j)=positions(i,j)+timestep*velocities(i,j)
  enddo
enddo

!!!!! get the new forces here from the new positions !!!!!!!

do i=1,nbr_atoms
  do j=1,3
    velocities(i,j)=velocities(i,j)+0.5*timestep*forces(i,j)/mass
  enddo
enddo

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!End of Velocity Verlet                                  !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!Output energies and structures at each MD step          !
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

!!!!!!!!!!!!!!!! Master MD Loop !!!!!!!!!!!!!!!!
enddo

end program my_first_MD_Code
```

Structure of the Code, more details

- ✓ The two previous slides present the minimum ingredients of an MD code. Other details can be added:
  - ✤ A ***random distribution of the velocities*** at the beginning of the program ⇒ has to correspond to the classical distribution of velocities at the desired temperature for classical particles.
  - ✤ A ***thermostat*** to control the temperature of your system ⇒ in that case, you work in the NVT (or something close) ensemble and not anymore in the NVE ensemble.
  - ✤ A spherical potential that prevents your system to evaporate.
  - ✤ …

Summary

Velocity Rescaling

- The simplest thermostat.

- At each MD step or every N MD steps, all the velocities of the system are rescaled:

$$v_{new}^{N} = \lambda v_{old}^{N}$$

Velocity Rescaling

- ✓ The simplest thermostat.

- ✓ At each MD step or every N MD steps, all the velocities of the system are rescaled:

$$v_{new}^{N} = \lambda v_{old}^{N}$$

- ✓ In the Velocity Rescaling, λ is defined as:

$$\lambda = \sqrt{\frac{T}{T_{instantaneous}}} \qquad T_{instantaneous} = \frac{2E_{kinetic}}{k_B N_{DOF}}$$

Velocity Rescaling

- ✓ The simplest thermostat.

- ✓ At each MD step or every N MD steps, all the velocities of the system are rescaled:

$$v_{new}^{N} = \lambda v_{old}^{N}$$

- ✓ In the Velocity Rescaling, λ is defined as:

$$\lambda = \sqrt{\frac{T}{T_{instantaneous}}} \qquad T_{instantaneous} = \frac{2E_{kinetic}}{k_B N_{DOF}}$$

- ✓ At each application of the thermostat, the kinetic energy of the system is equal to the target kinetic energy T.

Velocity Rescaling

- ✓ The simplest thermostat.
- ✓ At each MD step or every N MD steps, all the velocities of the system are rescaled:
- ✓ In the Velocity Rescaling, λ is defined as:
- ✓ At each application of the thermostat, the kinetic energy of the system is equal to the target kinetic energy T.
- ✓ **Advantages:**
  - ✤ Very easy to implement in any MD code

NVE and NVT Ensembles: Thermostats

Velocity Rescaling

- ✓ The simplest thermostat.
- ✓ At each MD step or every N MD steps, all the velocities of the system are rescaled:
- ✓ In the Velocity Rescaling, λ is defined as:
- ✓ At each application of the thermostat, the kinetic energy of the system is equal to the target kinetic energy T.
- ✓ **Advantages:**
  - ✤ Very easy to implement in any MD code
- ✓ **Disadvantages:** if one uses the thermostat at each MD step, there is no fluctuation of the kinetic energy of the system ⇒ the NVT ensemble is not described properly.

$$\sigma^2_{E_{kinetic}} = \frac{N_{DOF}}{2} k_B^2 T^2$$

Variance of the kinetic energy in the NVT ensemble.

Velocity Rescaling

- ✓ The simplest thermostat.
- ✓ At each MD step or every N MD steps, all the velocities of the system are rescaled:
- ✓ In the Velocity Rescaling, λ is defined as:
- ✓ At each application of the thermostat, the kinetic energy of the system is equal to the target kinetic energy T.
- ✓ **Advantages:**
  - ✤ Very easy to implement in any MD code
- ✓ **Disadvantages:** if one uses the thermostat at each MD step, there is no fluctuation of the kinetic energy of the system ⇒ the NVT ensemble is not described properly.
  - ✤ Poor thermalization.
  - ✤ Dynamics strongly biased along the trajectory.

Berendsen Thermostat

- Works on the same idea than the Velocity Rescaling but uses a different λ factor:

$$\lambda^2 = 1 + \frac{\delta t}{\tau}\left(\frac{T}{T_{instantaneous}} - 1\right)$$

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

Berendsen Thermostat

- Works on the same idea than the Velocity Rescaling but uses a different λ factor:

$$\lambda^2 = 1 + \frac{\delta t}{\tau}\left(\frac{T}{T_{instantaneous}} - 1\right)$$

- Expression proposed by Berendsen *et al.* in 1984. It supposes that the system is weakly coupled to a thermal bath.

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

Berendsen Thermostat

- Works on the same idea than the Velocity Rescaling but uses a different λ factor:

$$\lambda^2 = 1 + \frac{\delta t}{\tau}\left(\frac{T}{T_{instantaneous}} - 1\right)$$

- Expression proposed by Berendsen *et al.* in 1984. It supposes that the system is weakly coupled to a thermal bath.
  - The coupling strength is determined by τ: coupling constant with the bath or characteristic time of the thermal transfer.

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

Berendsen Thermostat

- Works on the same idea than the Velocity Rescaling but uses a different λ factor:

$$\lambda^2 = 1 + \frac{\delta t}{\tau}\left(\frac{T}{T_{instantaneous}} - 1\right)$$

- Expression proposed by Berendsen *et al.* in 1984. It supposes that the system is weakly coupled to a thermal bath.
  - The coupling strength is determined by τ: coupling constant with the bath or characteristic time of the thermal transfer.
  - If τ is large compared to δt (simulation timestep) ⇒ no thermostat, the NVE ensemble is explored.

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

Berendsen Thermostat

- Works on the same idea than the Velocity Rescaling but uses a different λ factor:

$$\lambda^2 = 1 + \frac{\delta t}{\tau}\left(\frac{T}{T_{instantaneous}} - 1\right)$$

- Expression proposed by Berendsen *et al.* in 1984. It supposes that the system is weakly coupled to a thermal bath.
  - The coupling strength is determined by τ: coupling constant with the bath or characteristic time of the thermal transfer.
  - If τ is large compared to δt (simulation timestep) ⇒ no thermostat, the NVE ensemble is explored.
  - If τ is equal to δt ⇒ Velocity Rescaling

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

Berendsen Thermostat

✓ Works on the same idea than the Velocity Rescaling but uses a different λ factor:

$$\lambda^2 = 1 + \frac{\delta t}{\tau}\left(\frac{T}{T_{instantaneous}} - 1\right)$$

✓ Expression proposed by Berendsen *et al.* in 1984. It supposes that the system is weakly coupled to a thermal bath.

- ✤ The coupling strength is determined by τ: coupling constant with the bath or characteristic time of the thermal transfer.
- ✤ If τ is large compared to δt (simulation timestep) ⇒ no thermostat, the NVE ensemble is explored.
- ✤ If τ is equal to δt ⇒ Velocity Rescaling

✓ Exponential decreasing of the system energy.

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

NVE and NVT Ensembles: Thermostats

Berendsen Thermostat

- ✓ Works on the same idea than the Velocity Rescaling but uses a different λ factor:
- ✓ Expression proposed by Berendsen *et al.* in 1984. It supposes that the system is weakly coupled to a thermal bath.
- ✓ Exponential decreasing of the system energy.

- ✓ **Advantages:**
  - ✤ Works well for large systems (several 100 of atoms).
  - ✤ Allows a quick equilibration of a system initially far from equilibrium.

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

NVE and NVT Ensembles: Thermostats

Berendsen Thermostat

- ✓ Works on the same idea than the Velocity Rescaling but uses a different λ factor:
- ✓ Expression proposed by Berendsen *et al.* in 1984. It supposes that the system is weakly coupled to a thermal bath.
- ✓ Exponential decreasing of the system energy.

- ✓ **Advantages:**
  - ✤ Works well for large systems (several 100 of atoms).
  - ✤ Allows a quick equilibration of a system initially far from equilibrium.

- ✓ **Disadvantages:**
  - ✤ As for the Velocity Rescaling, the Berendsen thermostat suppresses the kinetic energy fluctuations ⇒ do not allow to correctly explore the NVT ensemble.

H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, J. R. Haak J. Chem. Phys., *81*, 3684, **1984**.

Newton equations of motion for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

$$v(t+1/2dt) = v(t) + 1/2\ a(t)\ dt$$

$$r(t+dt) = r(t) + v(t+1/2dt)\ dt$$

get a(t+dt)

$$v(t+dt) = v(t+1/2dt) + 1/2\ a(t+dt)\ dt$$

You get positions r(t+dt), velocities v(t+dt) and accelerations a(t+dt)

Velocity-Verlet integrator

dt = timestep

WHERE IS THE THERMOSTAT ??

Integrating the Newton's Equations of Motion

Newton equations of motion for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

$$\mathbf{v(t+1/2dt) = v(t) + 1/2\ a(t)\ dt}$$

$$\mathbf{r(t+dt) = r(t) + v(t+1/2dt)\ dt}$$

get a(t+dt)

$$\mathbf{v(t+dt) = v(t+1/2dt) + 1/2\ a(t+dt)\ dt}$$

You get positions r(t+dt), velocities v(t+dt) and accelerations a(t+dt)

The thermostat will be applied here with ∂t

Velocity-Verlet integrator

dt = timestep

Integrating the Newton's Equations of Motion

Newton equations of motion for the nuclei

$$m\frac{d^2\mathbf{x}}{dt^2} = \mathbf{F}$$

Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)

$$\mathbf{v(t+1/2dt) = v(t) + 1/2\ a(t)\ dt}$$

$$\mathbf{r(t+dt) = r(t) + v(t+1/2dt)\ dt}$$

get a(t+dt)

$$\mathbf{v(t+dt) = v(t+1/2dt) + 1/2\ a(t+dt)\ dt}$$

The thermostat will be applied here with $\partial t/2$

Velocity-Verlet integrator

dt = timestep

You get positions r(t+dt), velocities v(t+dt) and accelerations a(t+dt)

First Exercise of Day 6

Write your MD Code

- ✓ **The goal is to write a first MD code that works properly**

First Exercise of Day 6

Write your MD Code

- ✓ **The goal is to write a first MD code that works properly**
- ✓ To help you, I provide several files in the LTTC network (all in the TP_MD folder):
  - ✤ *my_first_MD_code.f90* : this file contains the structure of an MD code. Everything works fine in this code except that **it calls a number of subroutines** that are in the *utilities.f90* module.

First Exercise of Day 6

Write your MD Code

- ✓ **The goal is to write a first MD code that works properly**
- ✓ To help you, I provide several files in the LTTC network (all in the TP_MD folder):
  - ✤ *my_first_MD_code.f90* : this file contains the structure of an MD code. Everything works fine in this code except that **it calls a number of subroutines** that are in the *utilities.f90* module.
  - ✤ I provide also an *utilities.f90* file, but this one has only its general structure. **Nothing is written in the different subroutines**.
  - ✤ **⇒ This is your work ! Complete the various subroutines in *utilities.f90***

First Exercise of Day 6

Write your MD Code

- ✓ **The goal is to write a first MD code that works properly**
- ✓ To help you, I provide several files in the LTTC network (all in the TP_MD folder):
  - ✤ *my_first_MD_code.f90* : this file contains the structure of an MD code. Everything works fine in this code except that **it calls a number of subroutines** that are in the *utilities.f90* module.
  - ✤ I provide also an *utilities.f90* file, but this one has only its general structure. **Nothing is written in the different subroutines**.
  - ✤ ⇒ **This is your work ! Complete the various subroutines in *utilities.f90***
  - ✤ To compile the whole code you do:

    ⇒ gfortan -c utilities.f90 (this generates utilities.o which is a kind of library)

    ⇒ gfortan utilities.o my_first_MD_code.f90 -o my_first_MD_code

First Exercise of Day 6

Write your MD Code

- ✓ **The goal is to write a first MD code that works properly**
- ✓ To help you, I provide several files in the LTTC network (all in the TP_MD folder):
  - ✤ *my_first_MD_code.f90* : this file contains the structure of an MD code. Everything works fine in this code except that **it calls a number of subroutines** that are in the *utilities.f90* module.
  - ✤ I provide also an *utilities.f90* file, but this one has only its general structure. **Nothing is written in the different subroutines**.
  - ✤ ⇒ **This is your work ! Complete the various subroutines in *utilities.f90***
  - ✤ To compile the whole code you do:

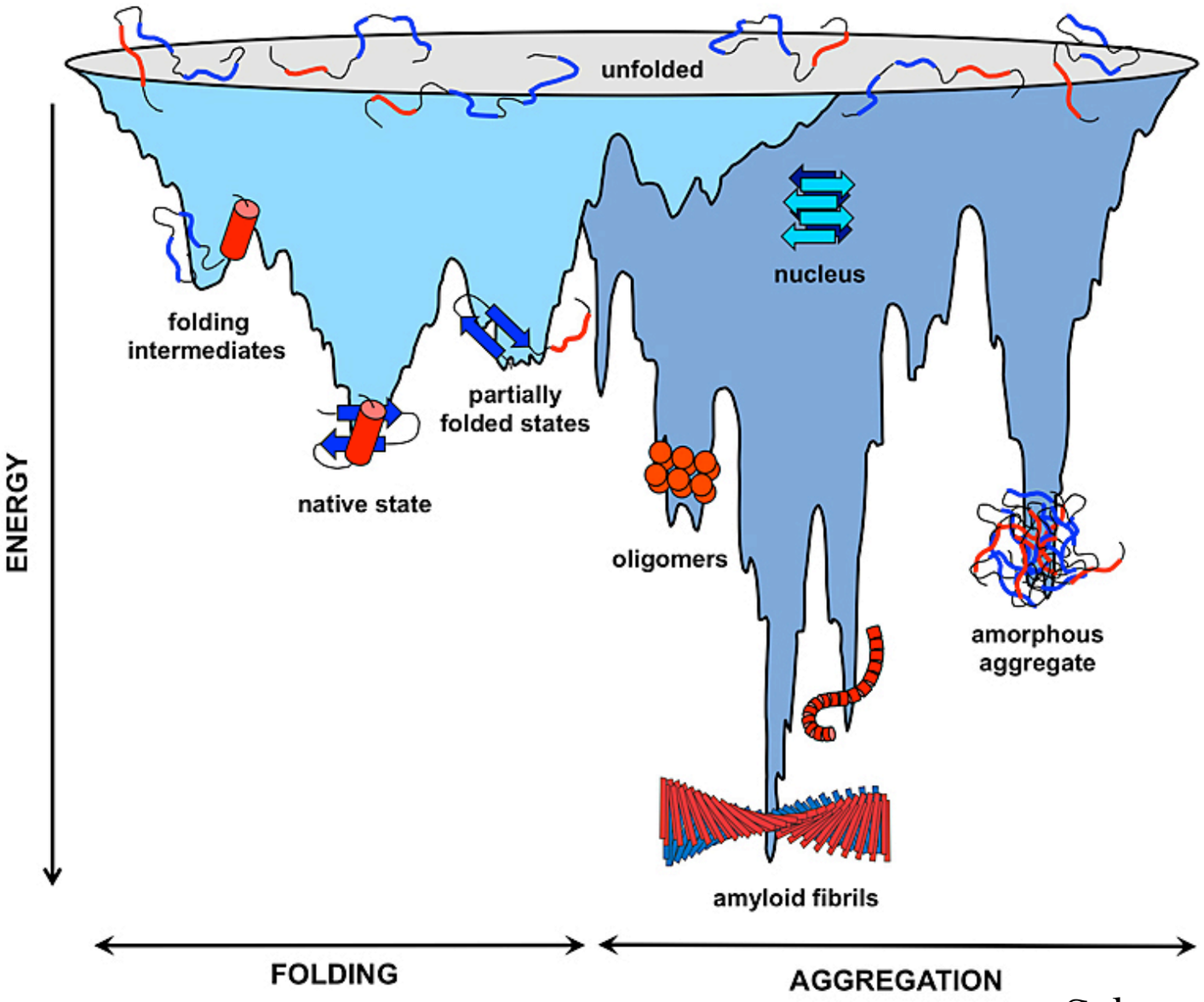
    ⇒ gfortan -c utilities.f90 (this generates utilities.o which is a kind of library)

    ⇒ gfortan utilities.o my_first_MD_code.f90 -o my_first_MD_code
  - ✤ At the end of the morning, the solution is given in *Utilities_solution.f90*

Summary

Enhanced Sampling: Parallel-Tempering

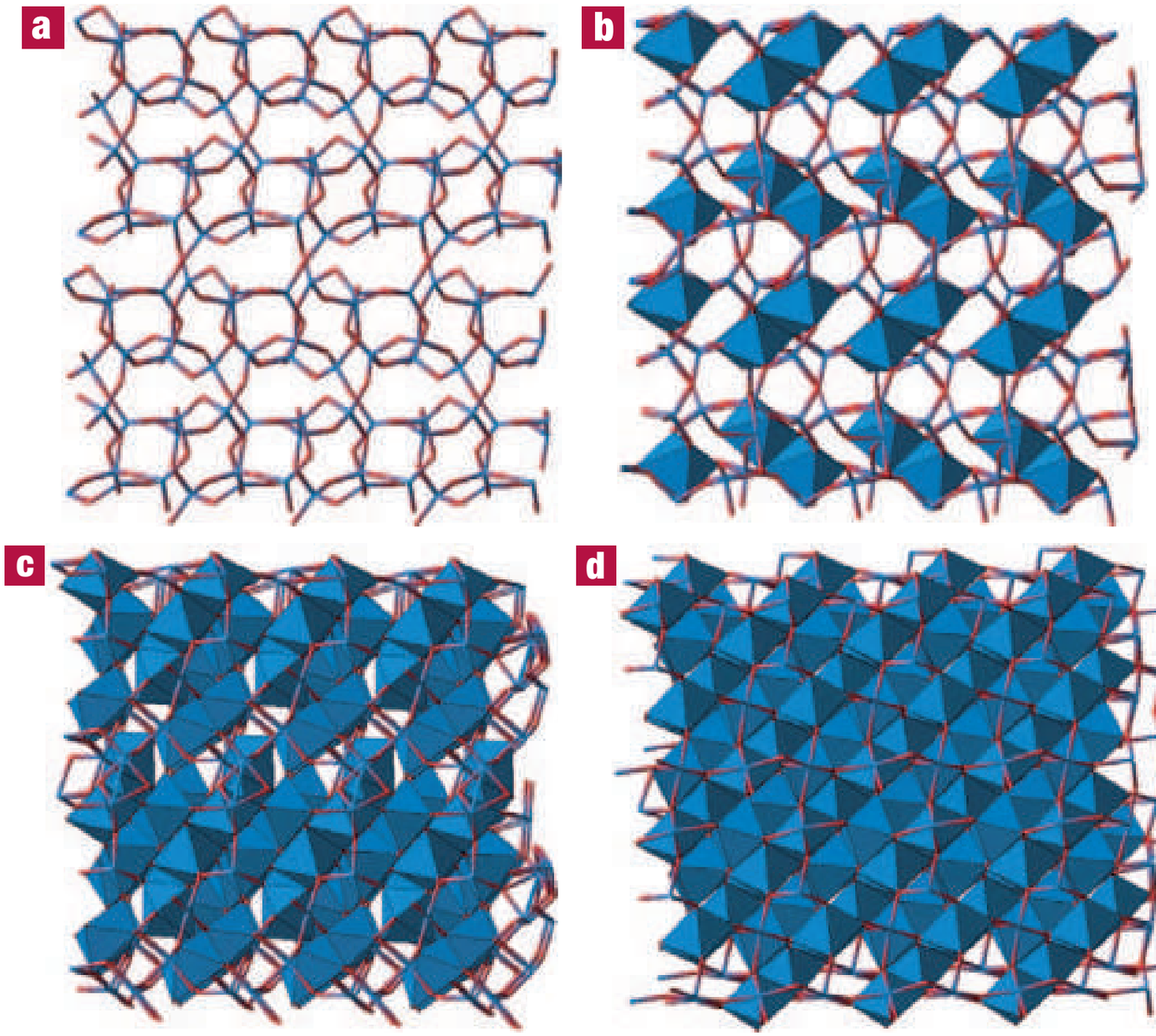
The Sampling Problem in Molecular Dynamics



Schematic **energy landscape for protein folding** and aggregation (see Jahn & Radford FEBS J. 2005). The competition between unimolecular folding and aggregate formation is intricately balanced by the cellular proteostasis network.

Enhanced Sampling: Parallel-Tempering

The Sampling Problem in Molecular Dynamics



Phase transition between coesite and α-PbO_2 phase of the SiO_2 system.

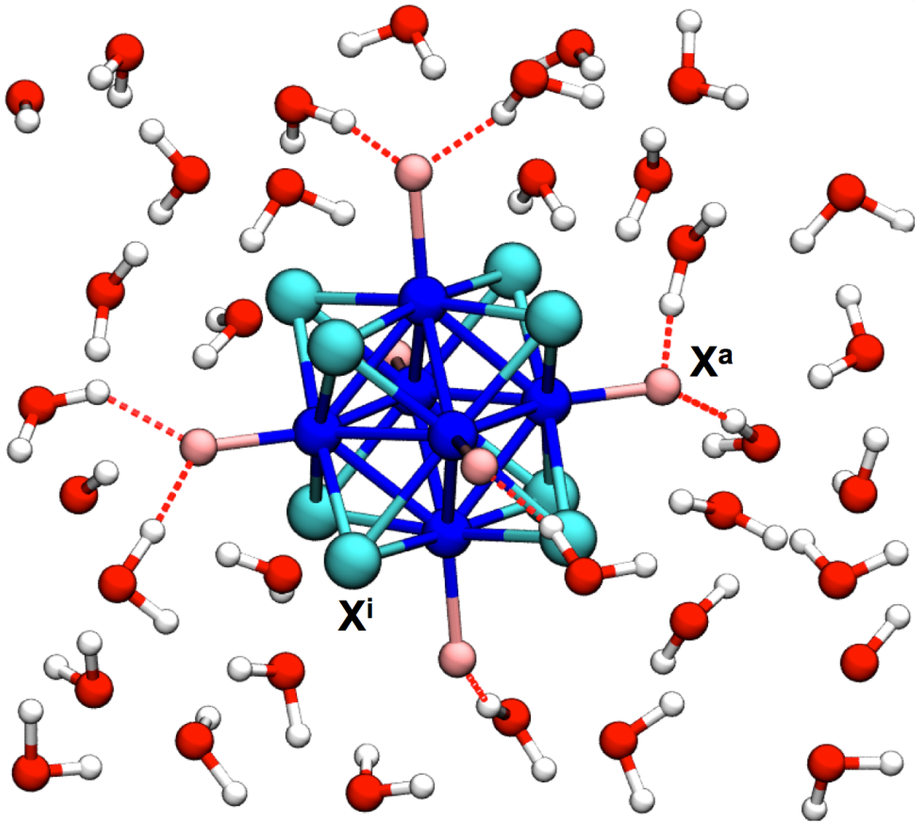
R. Martoňàk, D. Donadio, A. Oganov, M. Parrinello

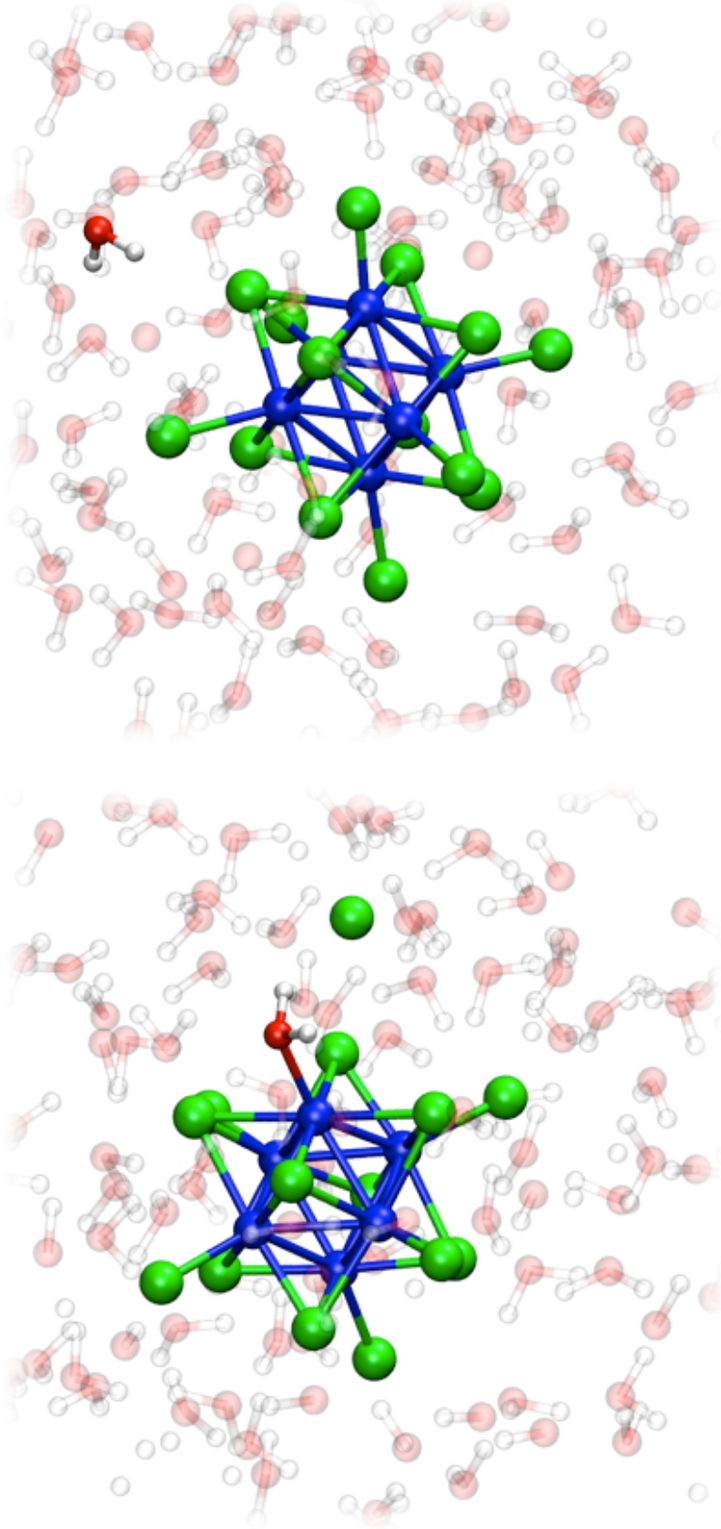
Nat. Mater., **2006**, *5*, 623-626

Enhanced Sampling: Parallel-Tempering

UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

The Sampling Problem in Molecular Dynamics



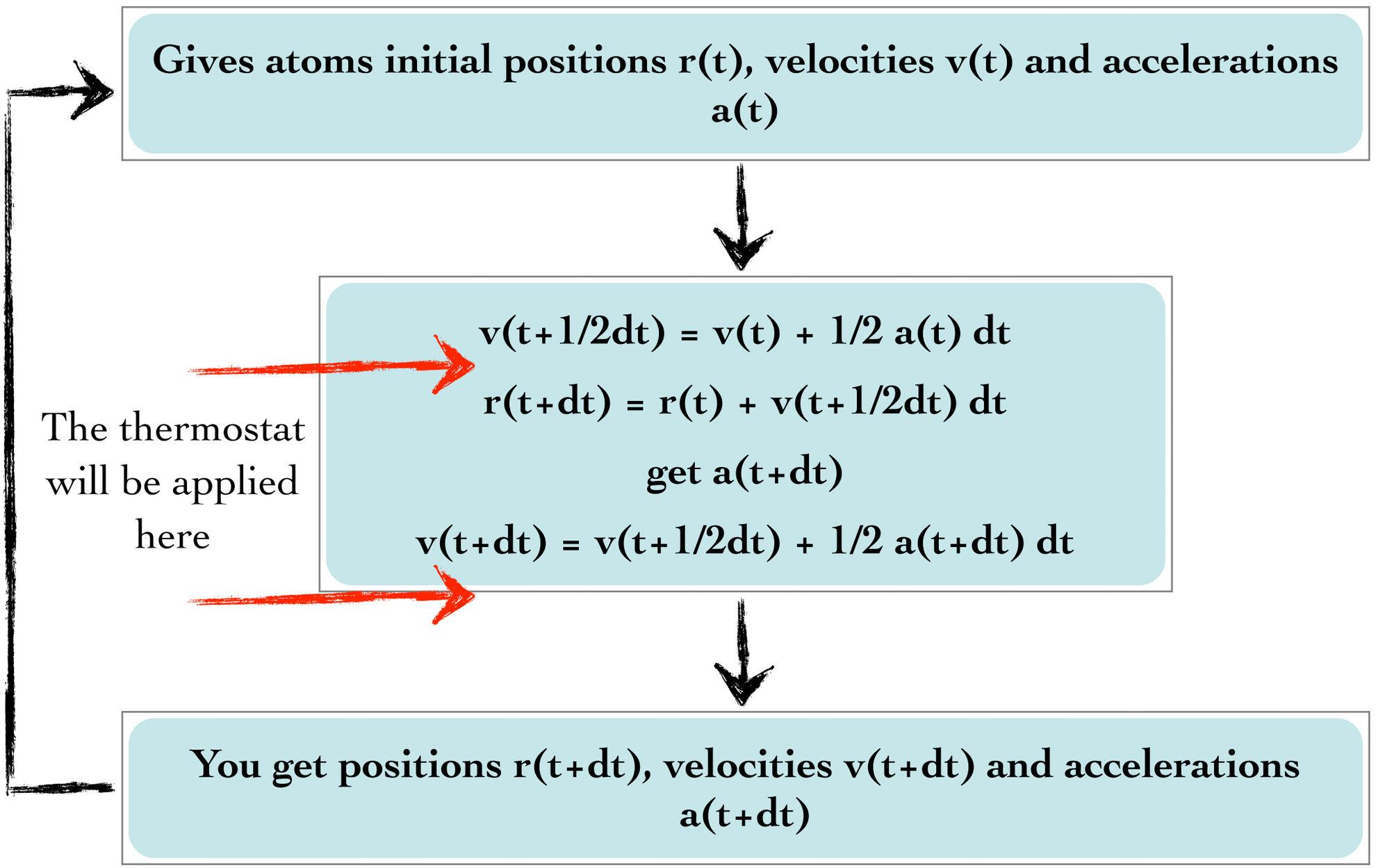


The Sampling Problem in Molecular Dynamics

Molecular dynamics simulations will **fail to model chemical** (or physical) **processes with energetic barriers higher than ~k_bT**.

This makes difficult (if not impossible) the simulation of:

- *Chemical Reactions*
- *Isomerisation*
- *Phase Transition within the solid-state*
- *Bio-molecular behavior such as protein folding, ligand binding and else*
- *and probably a lot of other interesting things ...*


Gives atoms initial positions r(t), velocities v(t) and accelerations a(t)
v(t+1/2dt) = v(t) + 1/2 a(t) dt
r(t+dt) = r(t) + v(t+1/2dt) dt
get a(t+dt)
v(t+dt) = v(t+1/2dt) + 1/2 a(t+dt) dt
The thermostat will be applied here
You get positions r(t+dt), velocities v(t+dt) and accelerations a(t+dt)

- ✓ **Simulated Annealing**
  - ✤ S. Kirkpatrick, C. D. Gelatt Jr, M. P. Vecchi *"Optimization by Simulated Annealing"* Science **1983**, *220*, 671–680
  - ✤ V. Černý "*Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*" Journal of Optimization Theory and Applications **1985**, *45*, 41–51.

- ✓ **Simulated Annealing**
  - ✤ S. Kirkpatrick, C. D. Gelatt Jr, M. P. Vecchi *"Optimization by Simulated Annealing"* Science **1983**, *220*, 671–680
  - ✤ V. Černý "*Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*" Journal of Optimization Theory and Applications **1985**, *45*, 41–51.

- ✓ **Umbrella Sampling**
  - ✤ G.M. Torrie, J. P. Valleau Journal of Computational Physics **1977**, *23*, 187-199.

Enhanced Sampling: Parallel-Tempering

The Zoo of Enhanced Sampling Methods

- ✓ **Simulated Annealing**
  - ✤ S. Kirkpatrick, C. D. Gelatt Jr, M. P. Vecchi "*Optimization by Simulated Annealing*" Science **1983**, *220*, 671–680
  - ✤ V. Černý "*Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*" Journal of Optimization Theory and Applications **1985**, *45*, 41–51.

- ✓ **Umbrella Sampling**
  - ✤ G.M. Torrie, J. P. Valleau Journal of Computational Physics **1977**, *23*, 187-199.

- ✓ **Parallel-Tempering MD** (= Replica-Exchange MD)
  - ✤ Y. Sugita and Y. Okamoto Chem. Phys. Lett., **1999**, *314*, 141– 151
  - ✤ Y. Sugita and Y. Okamoto Chem. Phys. Lett., **2000**, *329*, 261– 270

Enhanced Sampling: Parallel-Tempering

The Zoo of Enhanced Sampling Methods

- ✓ **Simulated Annealing**
  - ✤ S. Kirkpatrick, C. D. Gelatt Jr, M. P. Vecchi "*Optimization by Simulated Annealing*" Science **1983**, *220*, 671–680
  - ✤ V. Černý "*Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm*" Journal of Optimization Theory and Applications **1985**, *45*, 41–51.

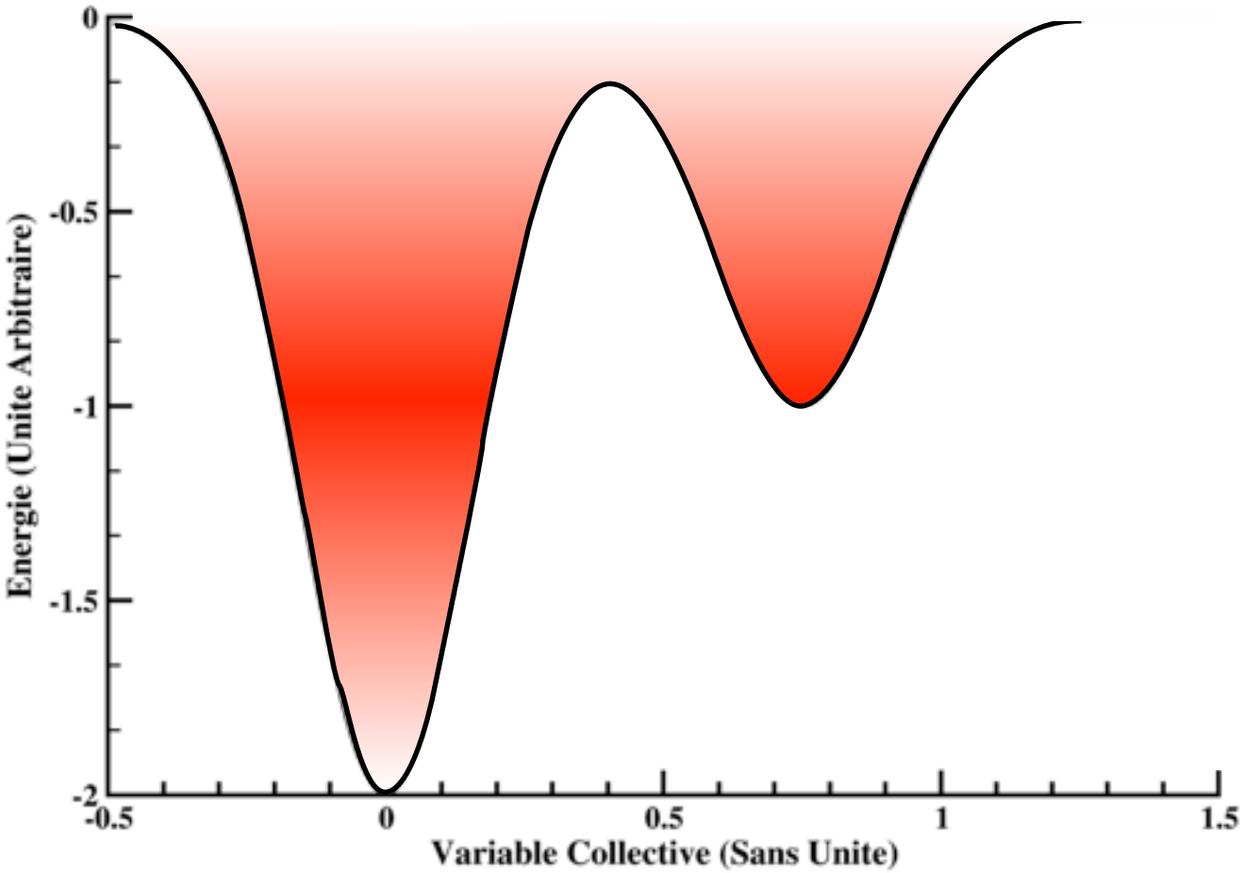
- ✓ **Umbrella Sampling**
  - ✤ G.M. Torrie, J. P. Valleau Journal of Computational Physics **1977**, *23*, 187-199.

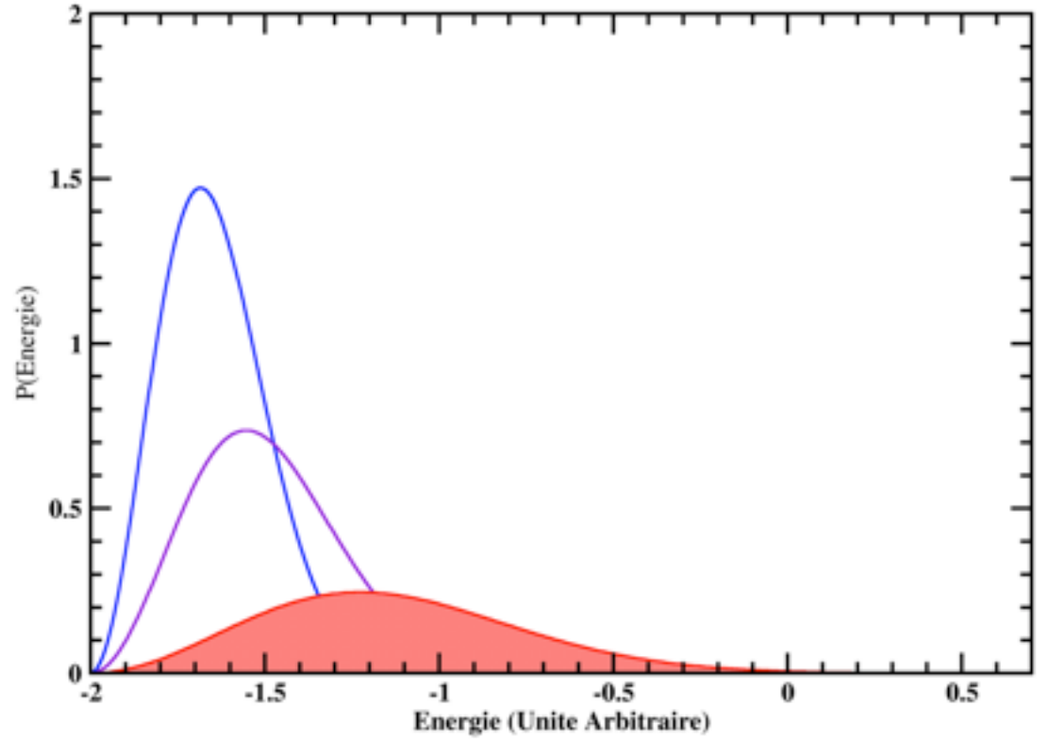
- ✓ **Parallel-Tempering MD** (= Replica-Exchange MD)
  - ✤ Y. Sugita and Y. Okamoto Chem. Phys. Lett., **1999**, *314*, 141– 151
  - ✤ Y. Sugita and Y. Okamoto Chem. Phys. Lett., **2000**, *329*, 261– 270

- ✓ **Metadynamics**
  - ✤ A. Laio, M. Parrinello "*Escaping free-energy minima*" Proc. Natl Acad. Sci. USA **2002**, *99*, 562–566
  - ✤ A. Laio, A. Rodriguez-Fortea, F. L. Gervasio, M. Ceccarelli, M. Parrinello "*Assessing the accuracy of metadynamics*" J. Phys. Chem. B **2005**, *109*, 6714–6721

The Parallel-Tempering Algorithm

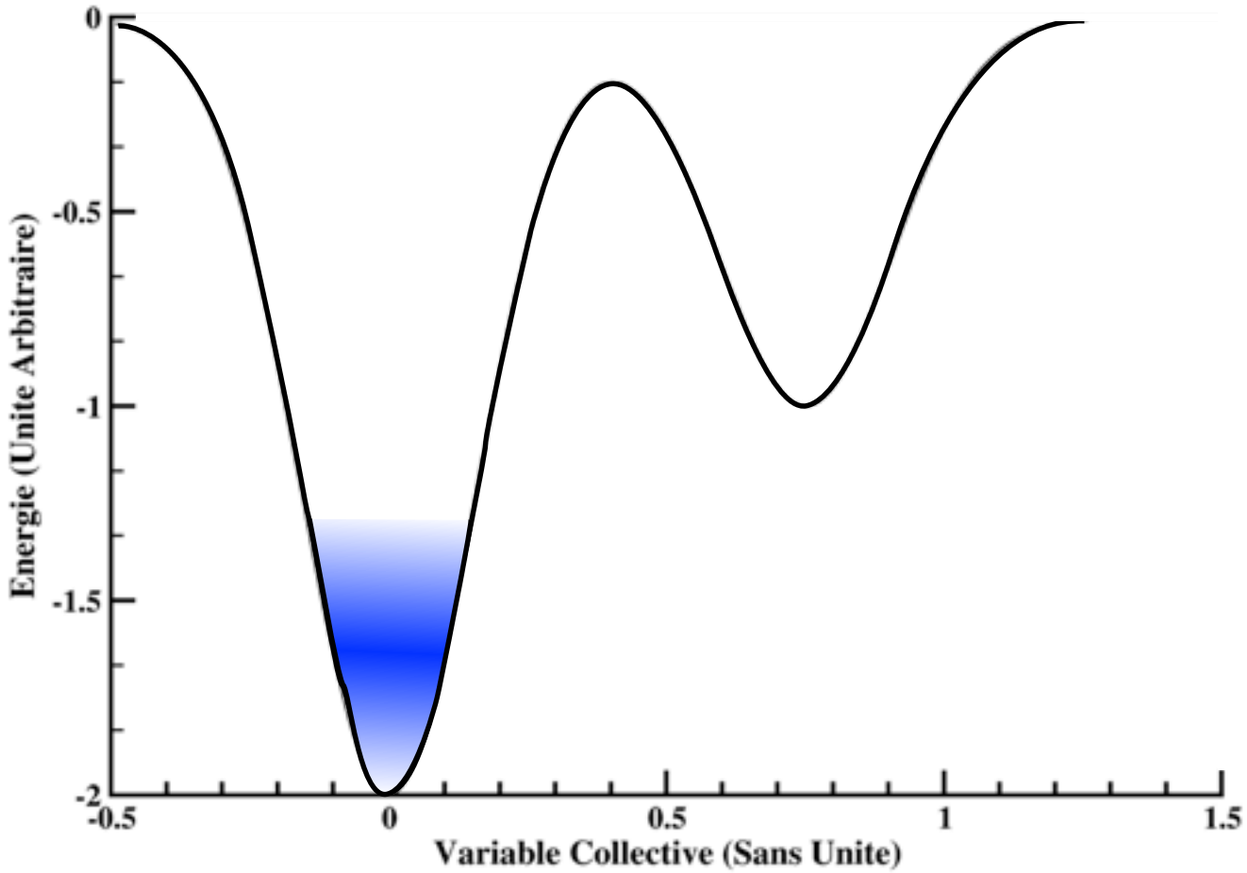
- A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).

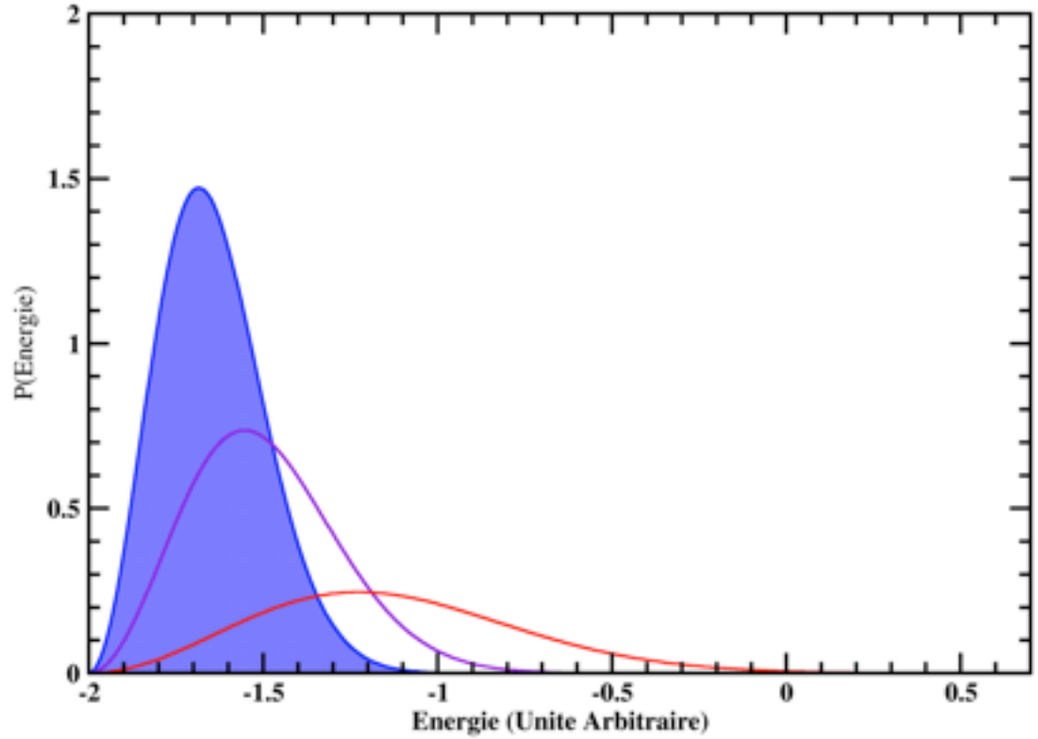




The Parallel-Tempering Algorithm

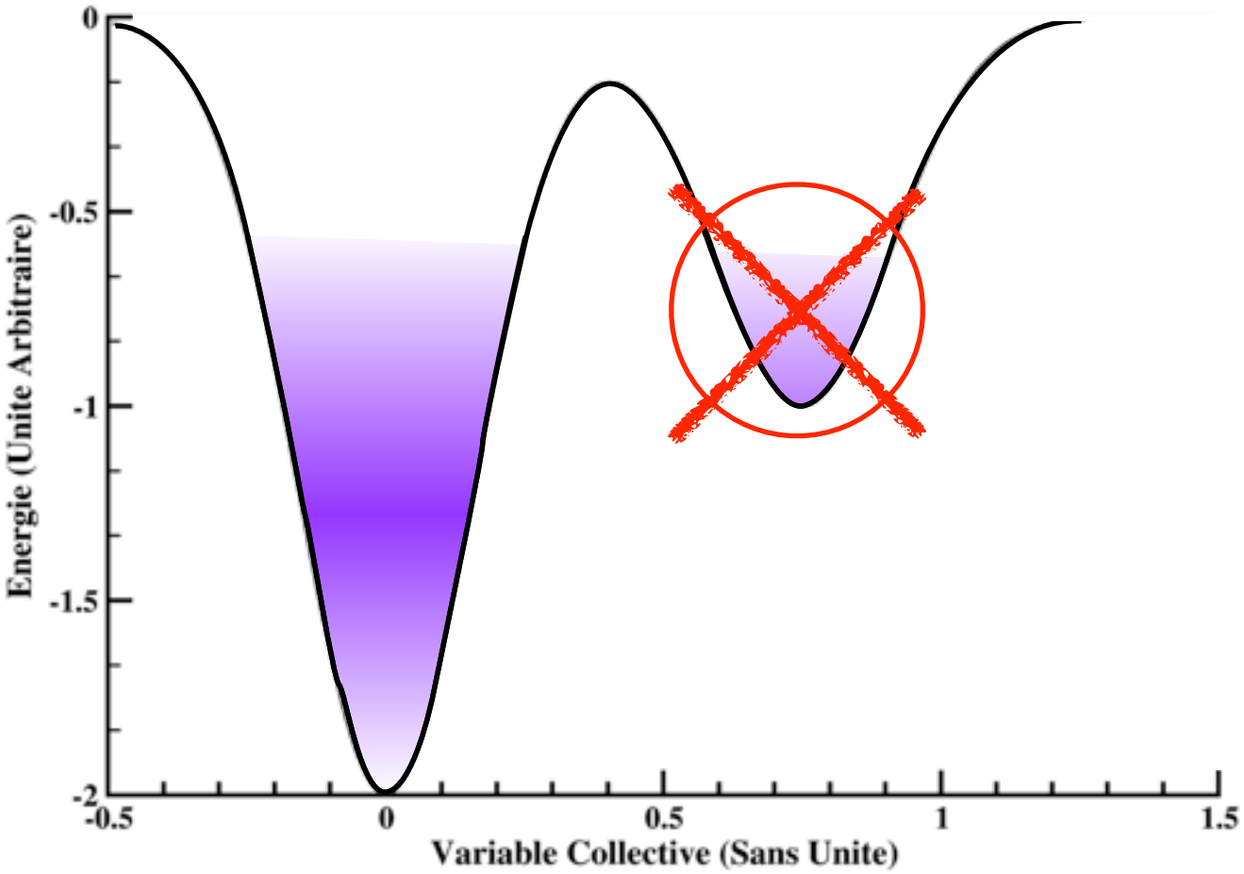
- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.

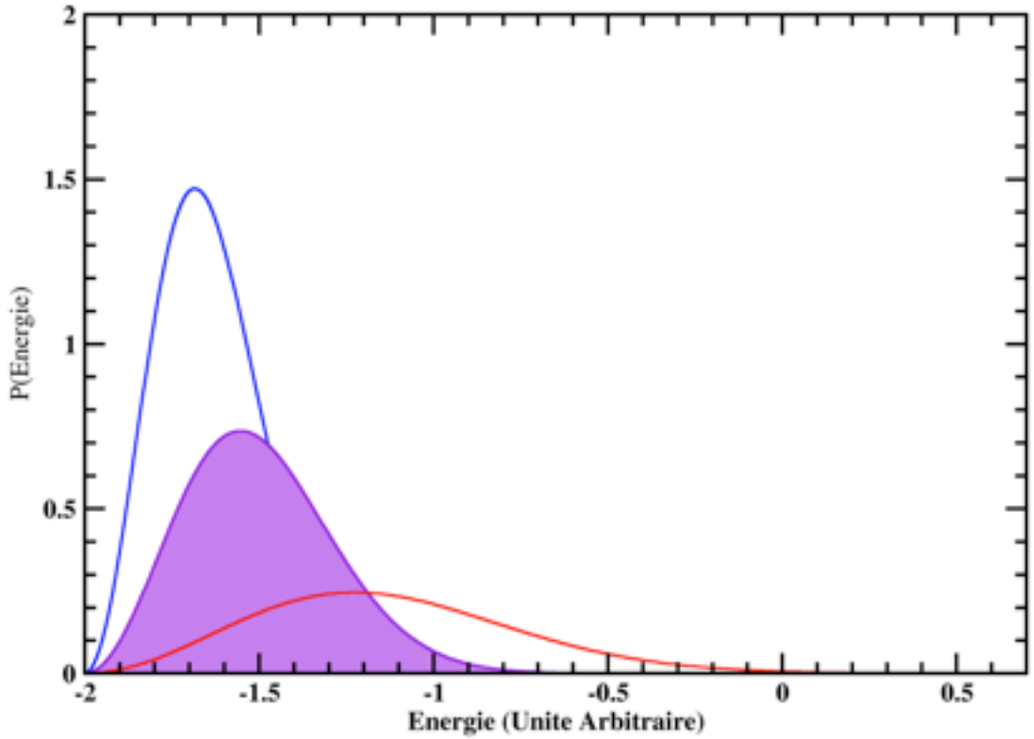




The Parallel-Tempering Algorithm

- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.

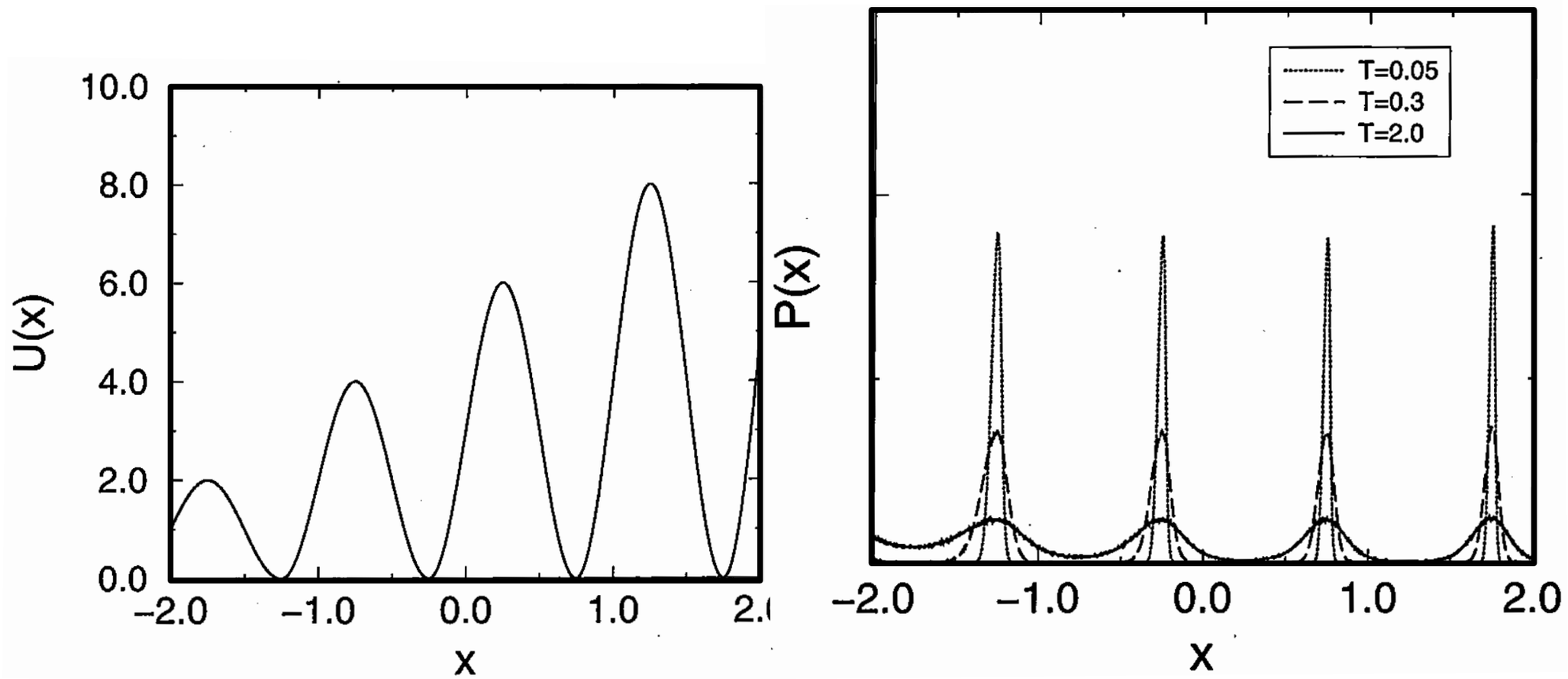




Enhanced Sampling: Parallel-Tempering

The Parallel-Tempering Algorithm

- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.

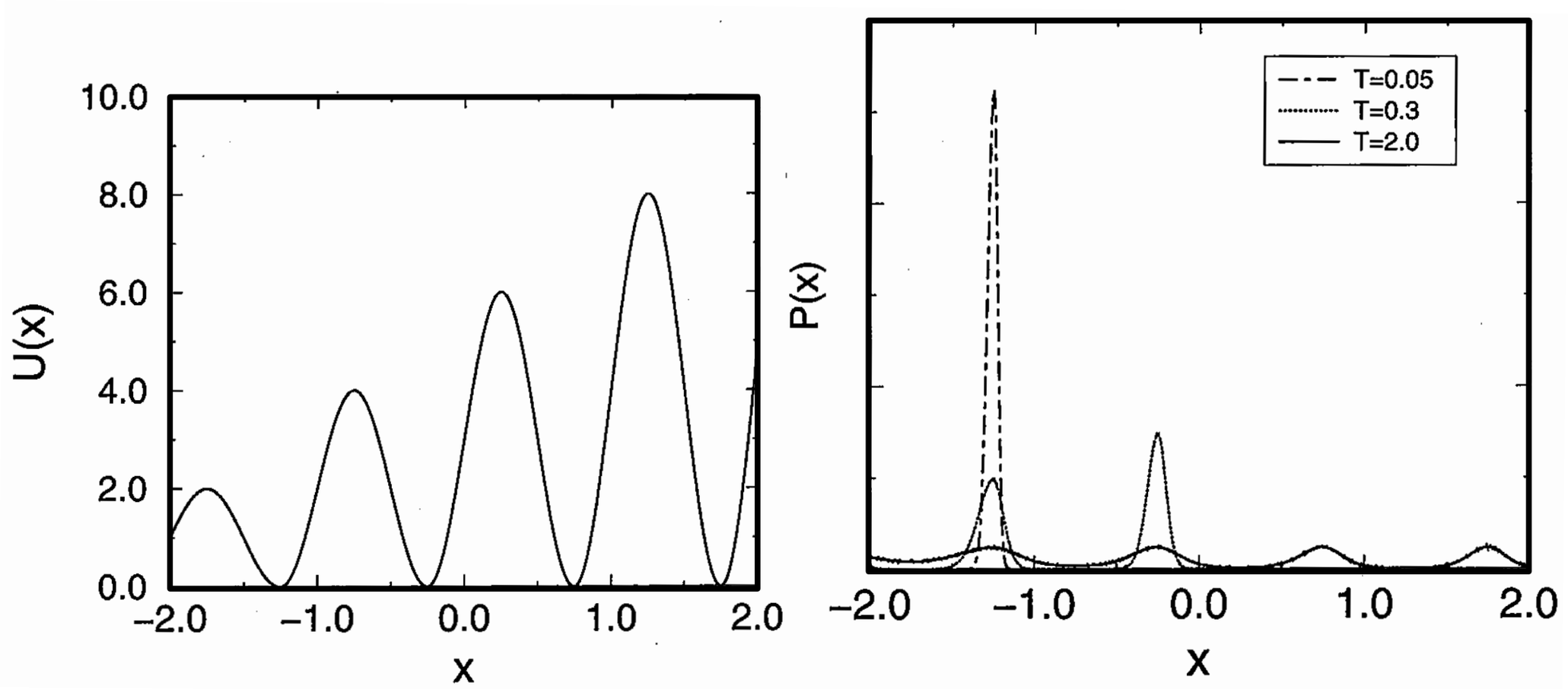


D. Frenkel and B. Smit *Understanding Molecular Simulations, From Algorithms to Applications,* ***Academy Press,*** **2002**, ch. 14, p. 392

Enhanced Sampling: Parallel-Tempering

The Parallel-Tempering Algorithm

- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.

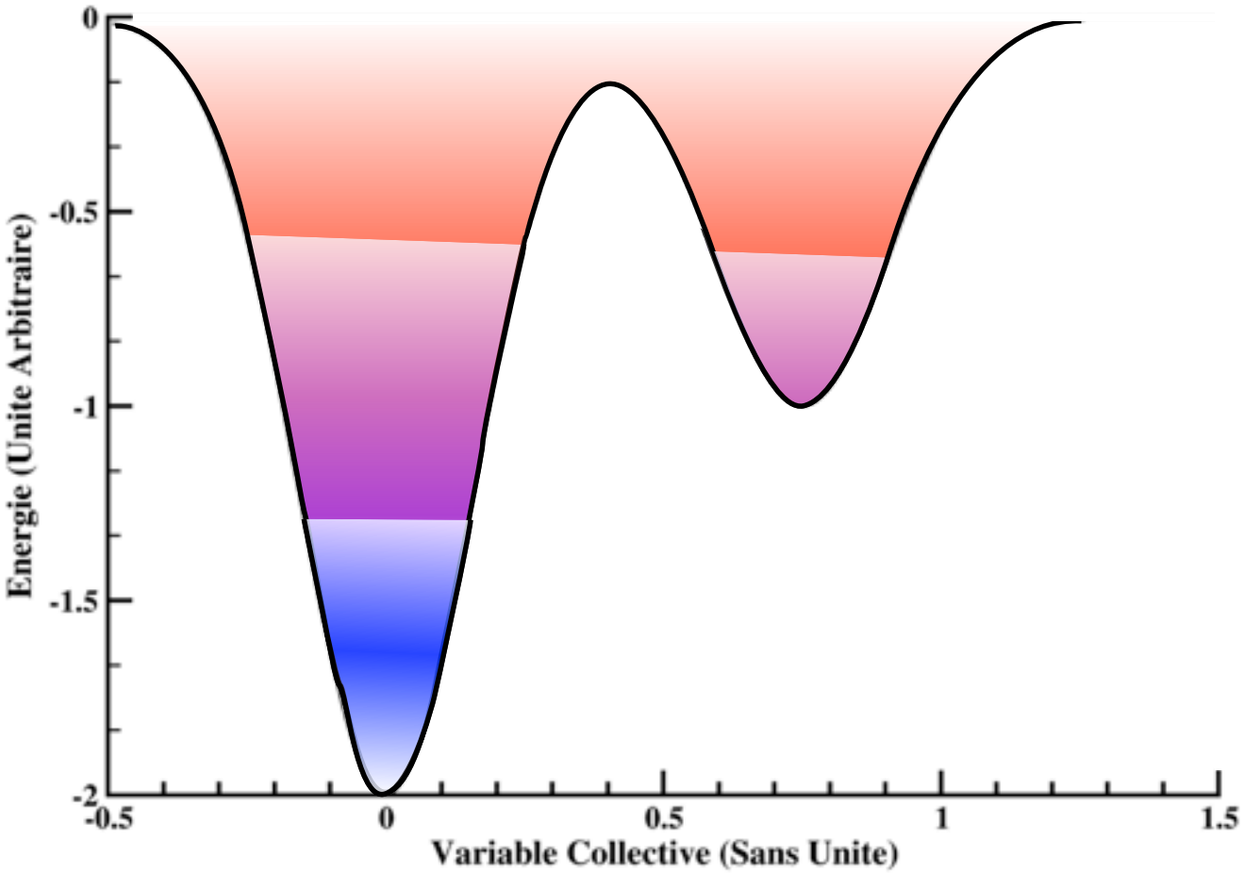


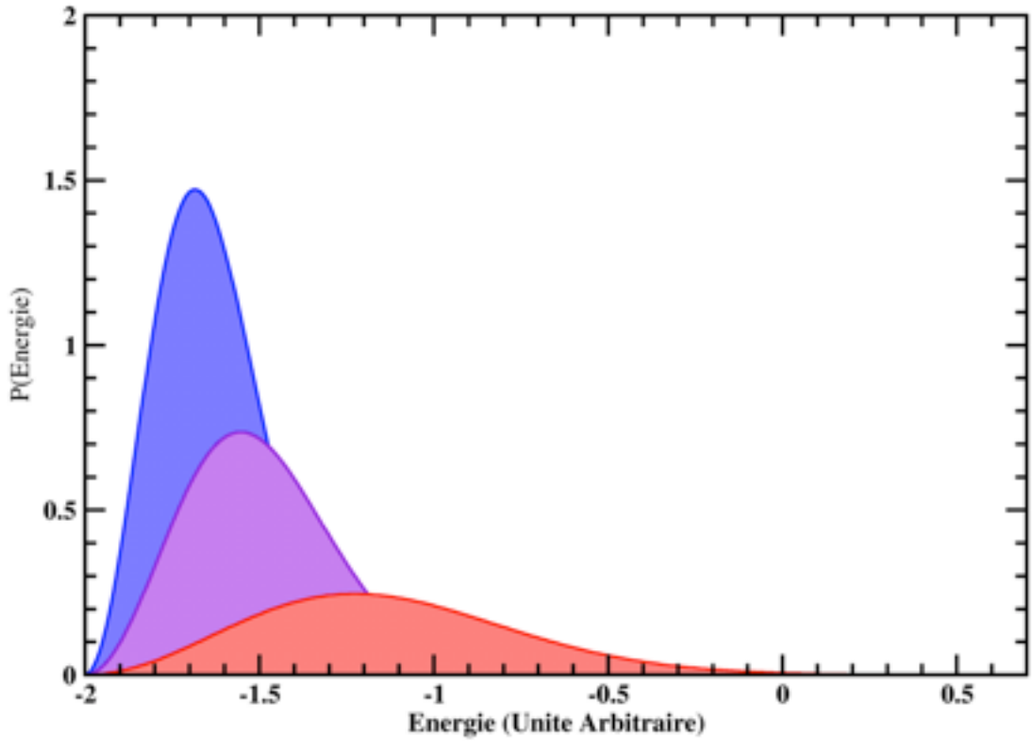
D. Frenkel and B. Smit *Understanding Molecular Simulations, From Algorithms to Applications,* ***Academy Press,*** **2002**, ch. 14, p. 392

Enhanced Sampling: Parallel-Tempering

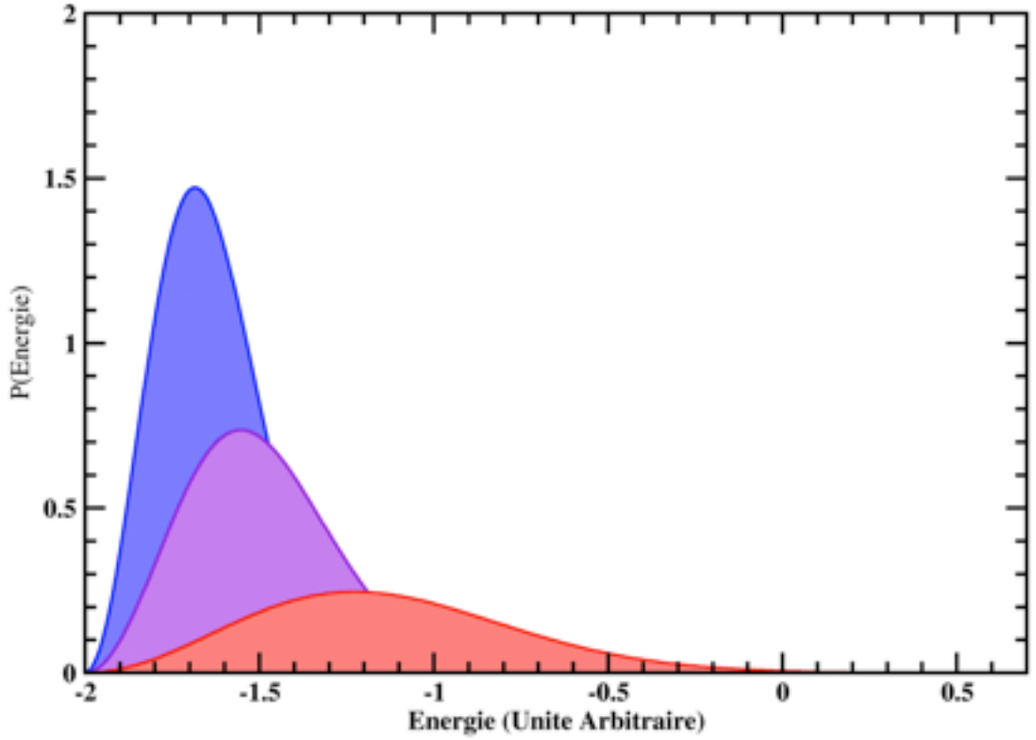
The Parallel-Tempering Algorithm

- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.



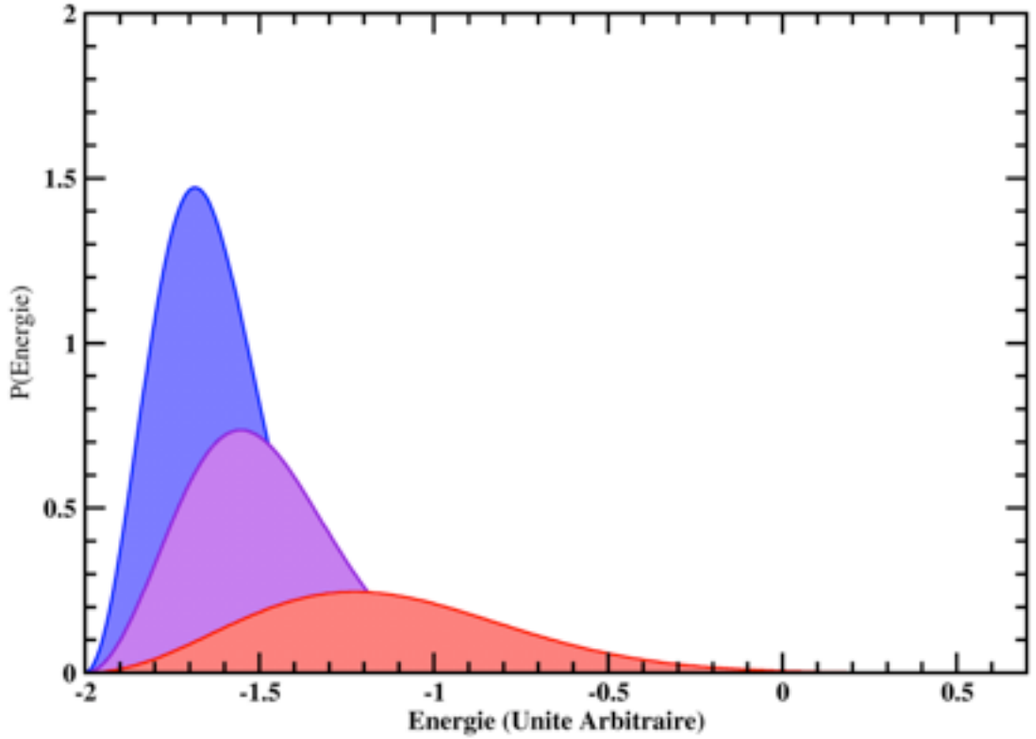


- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.
- ✓ There is overlap between the high and low temperature populations.



- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.

- ✓ There is overlap between the high and low temperature populations.
- ✓ Some configurations **belong to two (or several) populations** ⇒ one can thus exchange them without modifying the visited thermodynamics ensemble we are visiting.

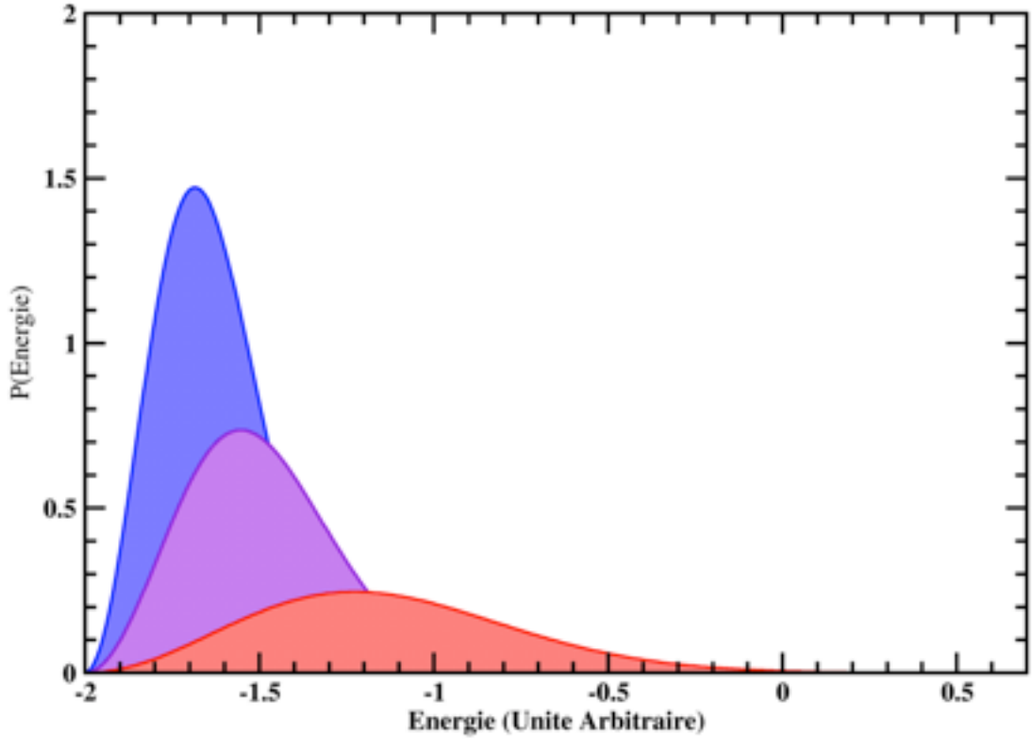


Enhanced Sampling: Parallel-Tempering

The Parallel-Tempering Algorithm

- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.

- ✓ There is overlap between the high and low temperature populations.
- ✓ Some configurations **belong to two (or several) populations** ⇒ one can thus exchange them without modifying the visited thermodynamics ensemble we are visiting.
- ✓ The probability that two configurations belong to two distinct populations is larger **if ΔT is smaller**.



Enhanced Sampling: Parallel-Tempering

The Parallel-Tempering Algorithm

- ✓ A high temperature MD simulation allow to cross the **energy barriers** characteristic of the free energy surface (FES).
- ✓ The interesting physics **is at low temperature**: **energy barriers** are not **crossable** within the length time of MD simulations.
- ✓ At **intermediate temperatures**, some barriers are crossed, others not.

  - ✤ In **Parallel-tempering**, one simulate in parallel **N replicas** (noted C_1, C_2, ..., C_N) of the same system, each replica having its own **temperature** (noted T_1, T_2, ..., T_N).

  - ✤ During the MD simulation, **the replicas are compared and a test is performed between pairs of replicas**.

  - ✤ Consider two replicas C_i and C_j with $T_i < T_j$, one uses a Métropolis criterion for the comparison:

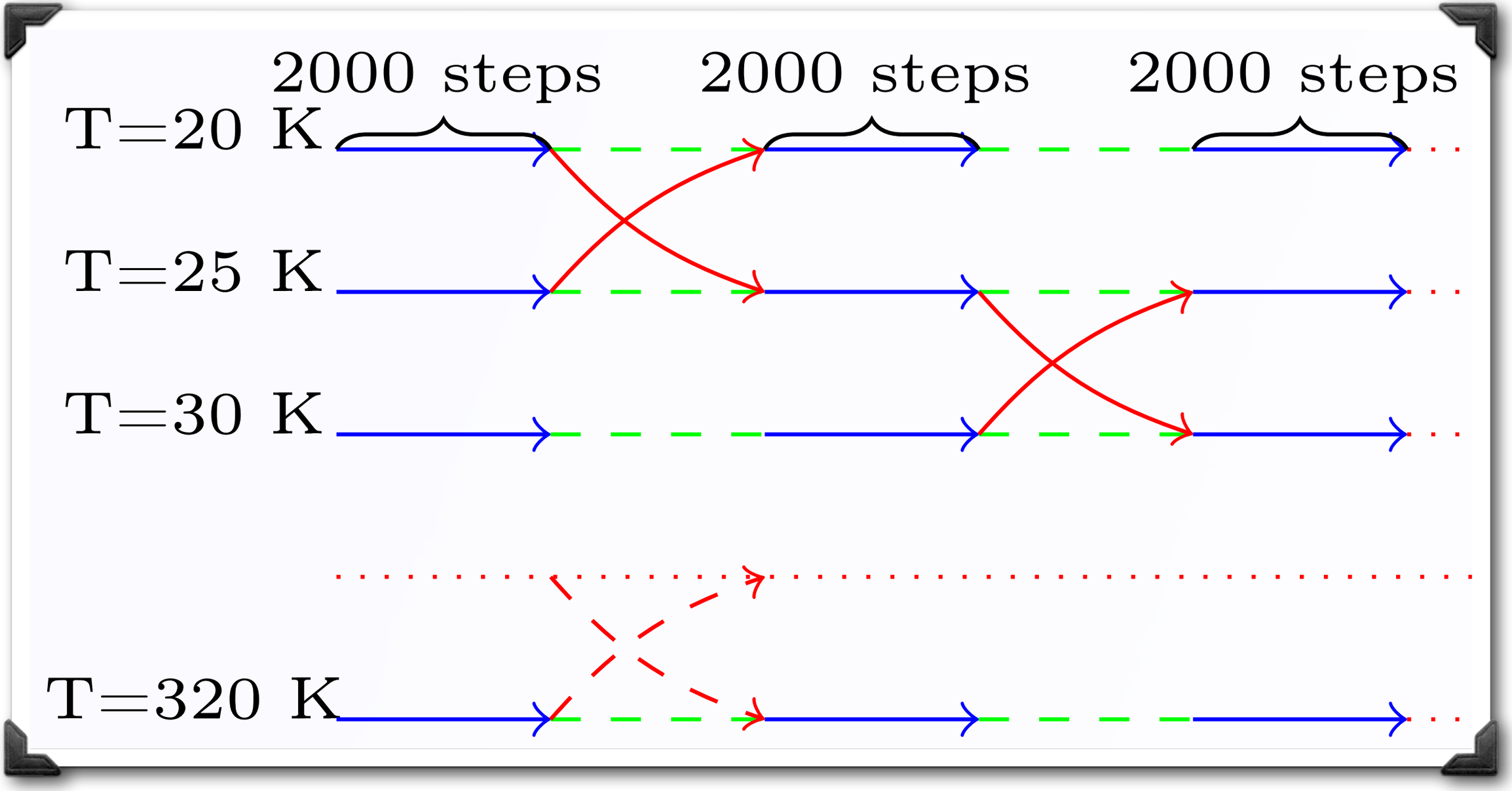
$$p\{C_{Old} \longrightarrow C_{New}\} = min\{1, e^{\Delta}\}$$

where Δ is:

$$\Delta = \Delta\beta\Delta E = (\beta_i - \beta_j)(E_i - E_j) = (\frac{1}{k_B T_i} - \frac{1}{k_B T_j})(E_i - E_j)$$

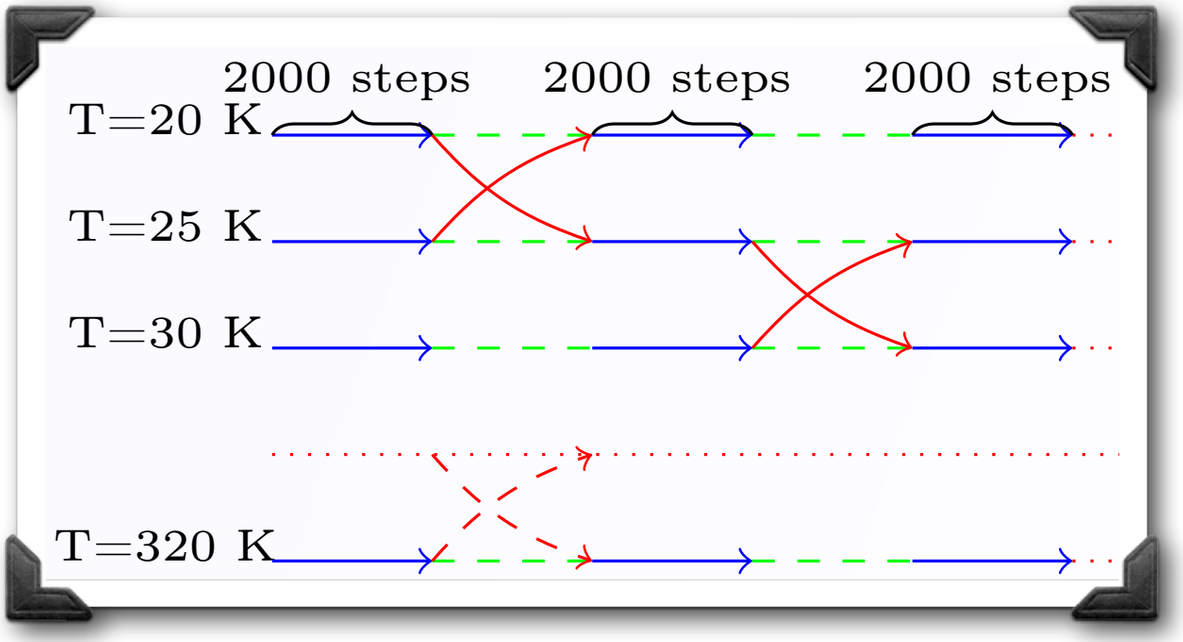
Enhanced Sampling: Parallel-Tempering

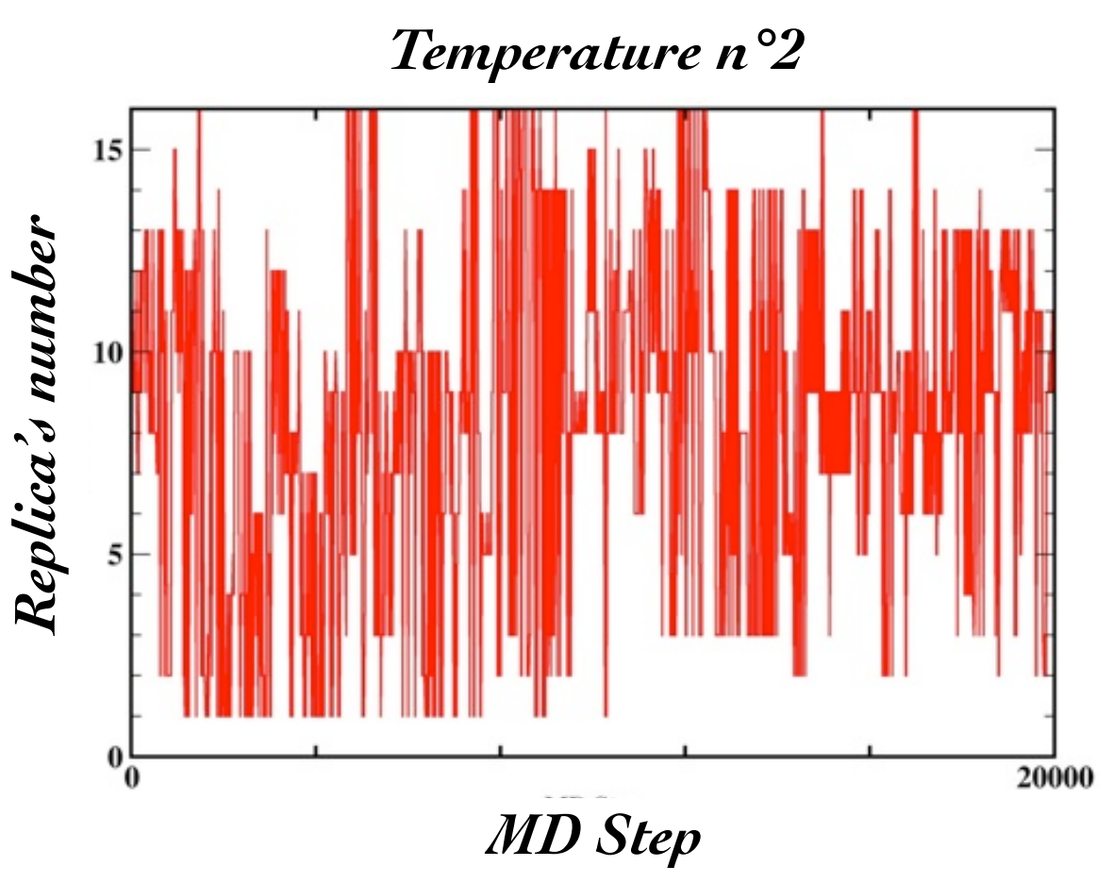
The Parallel-Tempering Algorithm

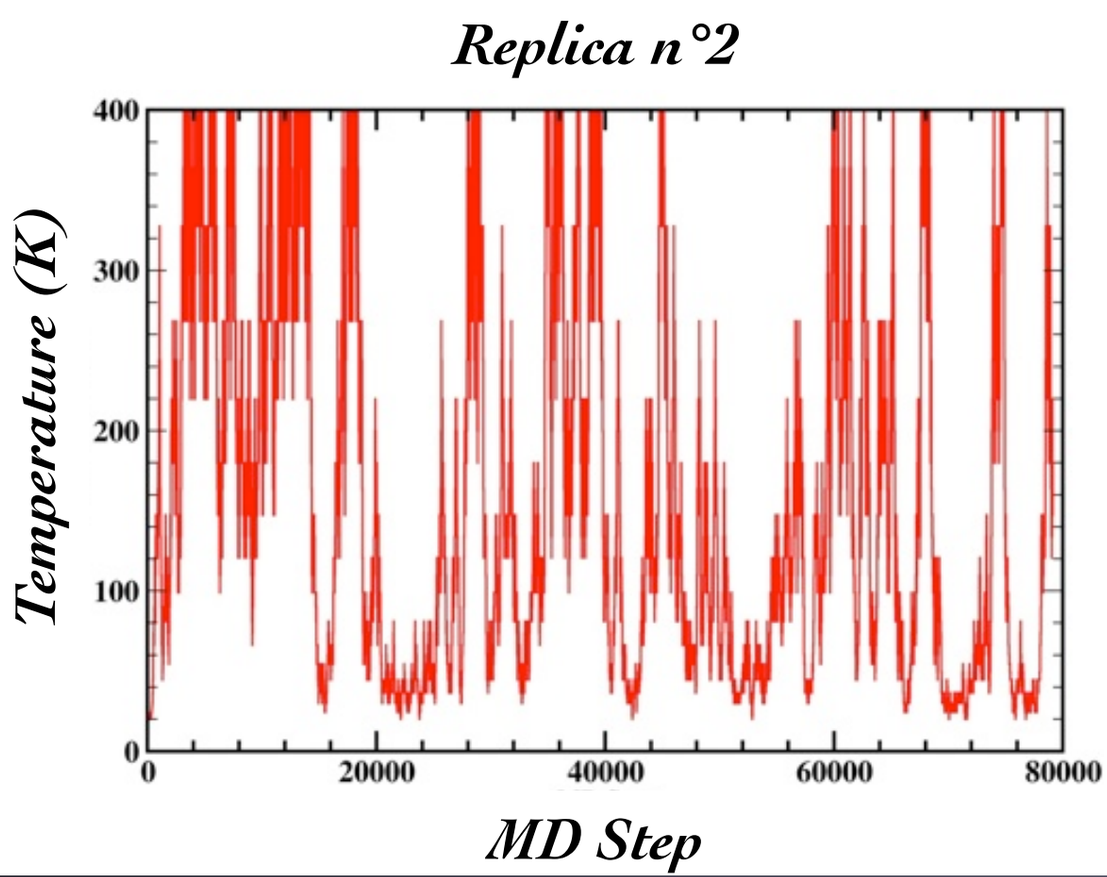


Enhanced Sampling: Parallel-Tempering

The Parallel-Tempering Algorithm

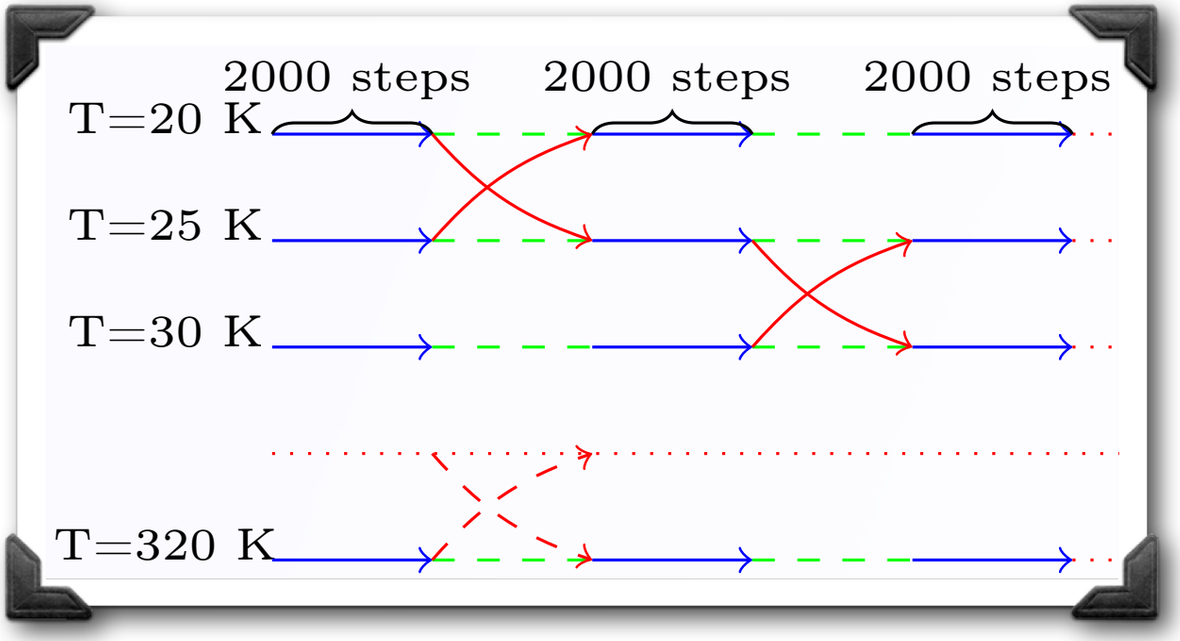


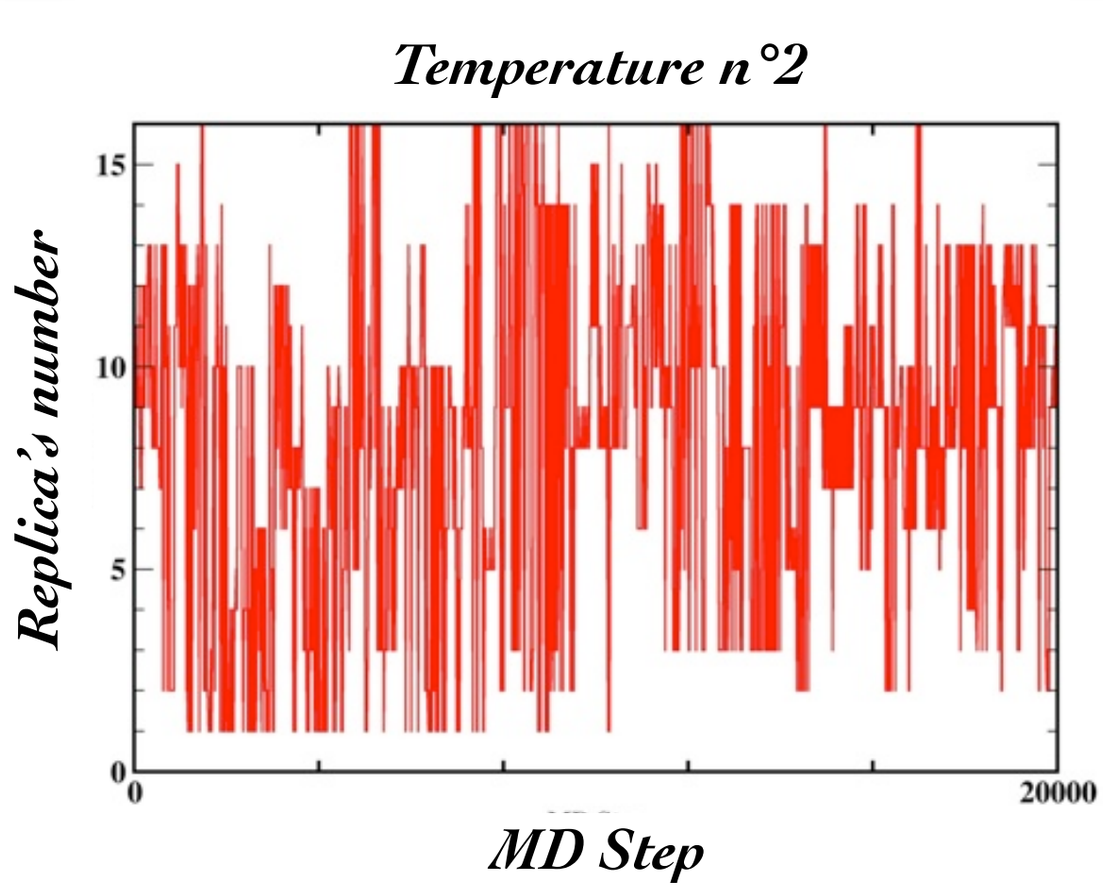


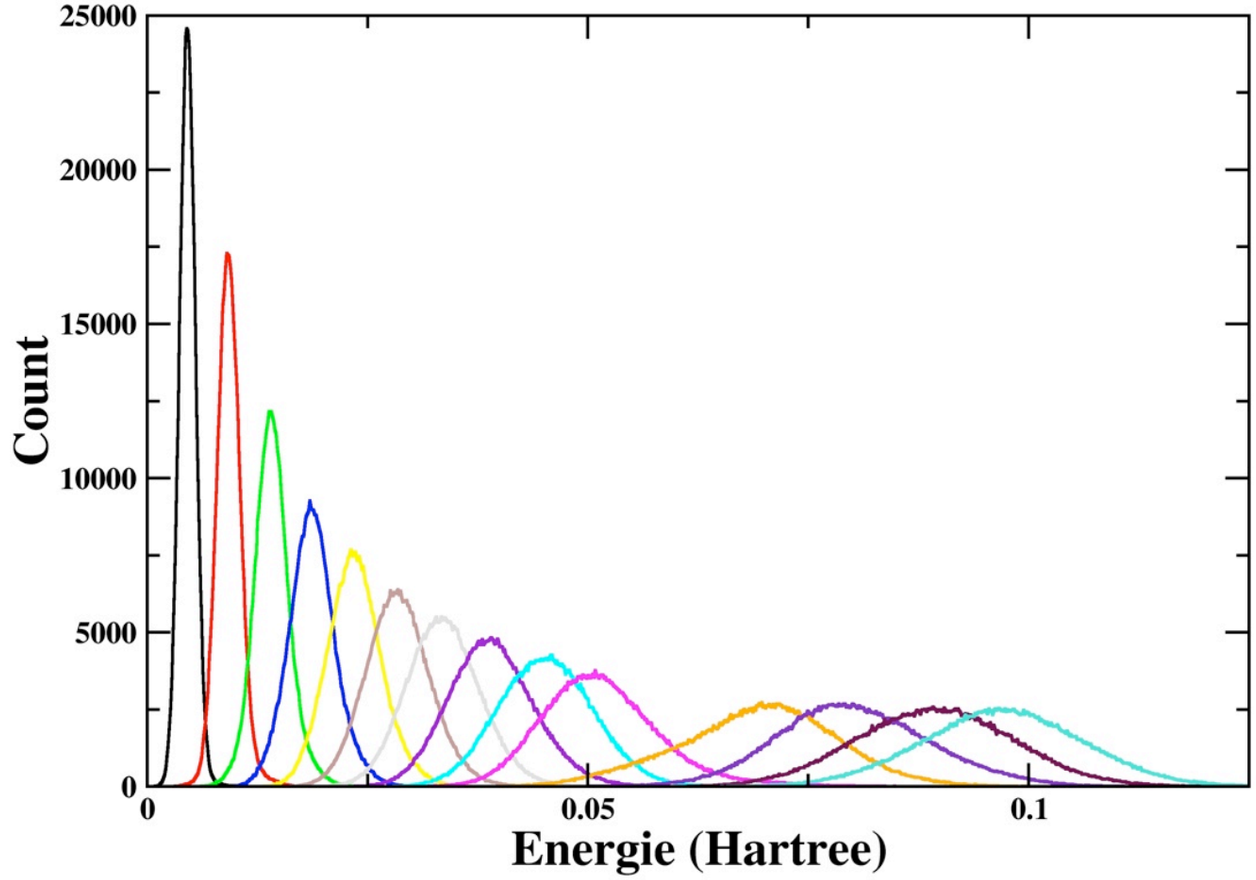


Enhanced Sampling: Parallel-Tempering

The Parallel-Tempering Algorithm

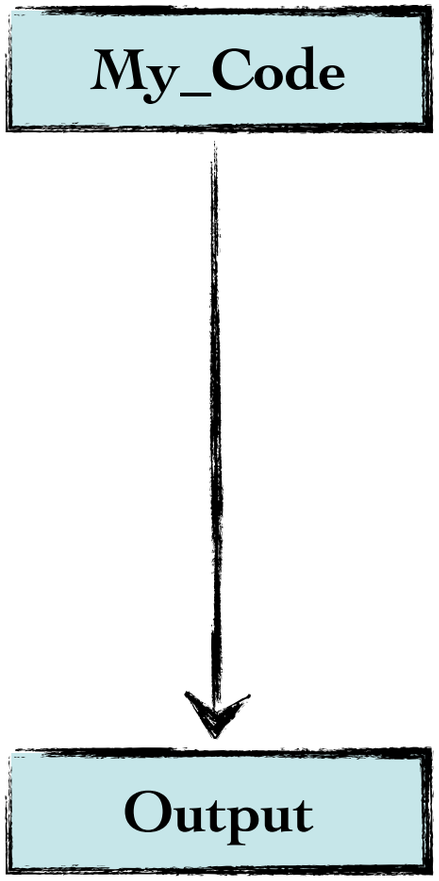






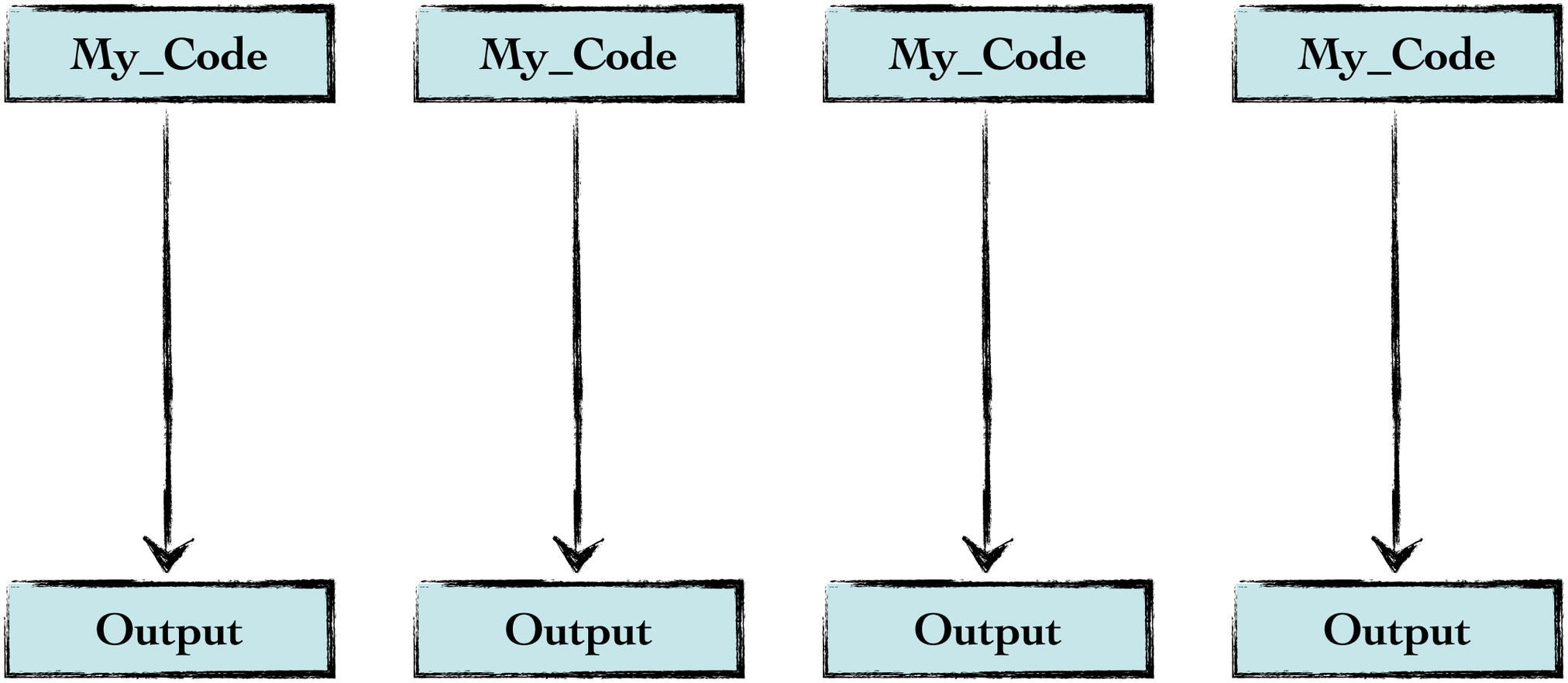
Summary

✓ I can run My_Code once as a serial code:



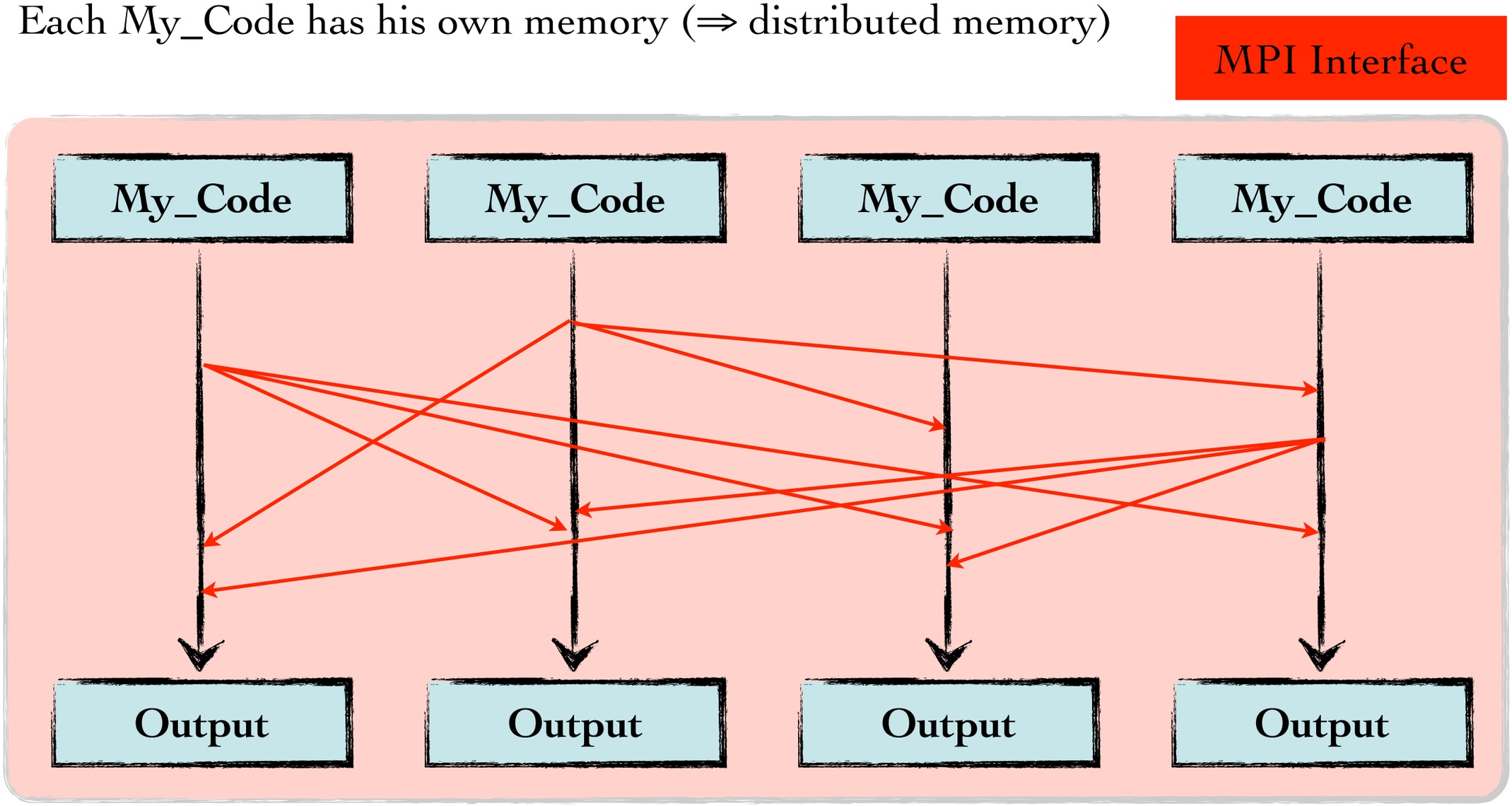
✓ I can run My_Code 4 times each as a serial code, each My_Code (= process) is independent from the other ones.

✤ Each My_Code has his own memory (⇒ distributed memory)



✓ I can run My_Code 4 times each as a serial code, but:

- ✤ I can define a global environment that allows the sharing of informations between the processes
  ⇒ This is MPI
- ✤ Each My_Code has his own memory (⇒ distributed memory)



- ✓ I can run My_Code 4 times each as a serial code, but:
  - ✤ I can define a global environment that allows the sharing of informations between the processes ⇒ This is MPI
  - ✤ Each My_Code has his own memory (⇒ distributed memory)
  - ✤ Serial sections of the program are implemented by identical computation on all nodes rather than computing the result on one node and sending it to the others

- ✓ I can run My_Code 4 times each as a serial code, but:
  - ✤ I can define a global environment that allows the sharing of informations between the processes ⇒ This is MPI
  - ✤ Each My_Code has his own memory (⇒ distributed memory)

- ✓ MPI adds a number of instructions to the Fortran, C, C++ languages that allows for communications between processes.

- ✓ Nothing changes as compared to a serial Fortran code, there are just new instructions to learn.

- ✓ I can run My_Code 4 times each as a serial code, but:
  - ✤ I can define a global environment that allows the sharing of informations between the processes ⇒ This is MPI
  - ✤ Each My_Code has his own memory (⇒ distributed memory)

- ✓ MPI adds a number of instructions to the Fortran, C, C++ languages that allows for communications between processes.

- ✓ Nothing changes as compared to a serial Fortran code, there are just new instructions to learn.
- ✓ A MPI code cannot be compiled with a gfortran (or ifort, ...) compiler, it needs another compiler that knows the MPI instructions: mpif90, mpif77, mpicxx.

- ✓ I can run My_Code 4 times each as a serial code, but:
  - ✤ I can define a global environment that allows the sharing of informations between the processes ⇒ This is MPI
  - ✤ Each My_Code has his own memory (⇒ distributed memory)

- ✓ MPI adds a number of instructions to the Fortran, C, C++ languages that allows for communications between processes.

- ✓ Nothing changes as compared to a serial Fortran code, there are just new instructions to learn.
- ✓ A MPI code cannot be compiled with a gfortran (or ifort, ...) compiler, it needs another compiler that knows the MPI instructions: mpif90, mpif77, mpicxx.
- ✓ Similarly, a MPI code cannot be executed as it is (./a.exe), it needs the mpirun command:

```
mpirun -np 4 a.exe
```

- ✓ Here -np 4 designs the number of processes used for the execution, the code is run on 4 processes

First MPI code to see how it works (In TP_MD)

```
program first_prog_mpi

implicit none

include 'mpif.h'

integer           :: ierr
integer           :: nprocs          ! total number of processes
integer           :: ME              ! process index
integer,parameter :: source=0
character(len=1)  :: numero

call MPI_Init(ierr)

call MPI_Comm_size(MPI_COMM_WORLD,nprocs,ierr) ! Number of available processes

call MPI_Comm_rank(MPI_COMM_WORLD,ME,ierr)     ! Index of each process

print*, "nprocs", nprocs, "ME", ME

write(numero,'(I1)') ME

open(6,file='TEST'//numero)

write(6,'(A,I2)'), "Hello from processe number ", ME

close(6)

call MPI_Finalize(ierr)

end program
```

MPI: Basis Commands

- They are four fundamental MPI commands:
  - *call MPI_Init(ierr)*: initialize the MPI environment
  - *call MPI_Comm_size(MPI_COMM_WORLD,nprocs,ierr)*: provides the number of processes available
  - *call MPI_Comm_rank(MPI_COMM_WORLD,ME,ierr):* gives to each process an index
  - *call MPI_Finalize(ierr)*: close the MPI environment

MPI: Basis Commands

- They are four fundamental MPI commands:
  - *call MPI_Init(ierr)*: initialize the MPI environment
  - *call MPI_Comm_size(MPI_COMM_WORLD,nprocs,ierr)*: provides the number of processes available
  - *call MPI_Comm_rank(MPI_COMM_WORLD,ME,ierr):* gives to each process an index
  - *call MPI_Finalize(ierr)*: close the MPI environment

- MPI_COMM_WORLD is the default name for the communicator encompassing all the processes. **Processes belonging to the same communicator can communicate.**

- By default, all processes can communicate in MPI_COMM_WORLD.

✓ Try to modify *first_mpi_program.f90* to execute simple different operations on each process

```
program first_prog_mpi

implicit none

include 'mpif.h'

integer            :: ierr
integer            :: nprocs         ! total number of processes
integer            :: ME             ! process index
integer,parameter  :: source=0
character(len=1)   :: numero

call MPI_Init(ierr)

call MPI_Comm_size(MPI_COMM_WORLD,nprocs,ierr) ! Number of available processes

call MPI_Comm_rank(MPI_COMM_WORLD,ME,ierr)     ! Index of each process

print*, "nprocs", nprocs, "ME", ME

write(numero,'(I1)') ME

open(6,file='TEST'//numero)

write(6,'(A,I2)'), "Hello from processe number ", ME

close(6)

call MPI_Finalize(ierr)

end program
```

MPI: Basis Commands to share data

- ✓ Of course, it exists **commands to communicate between the processes**:

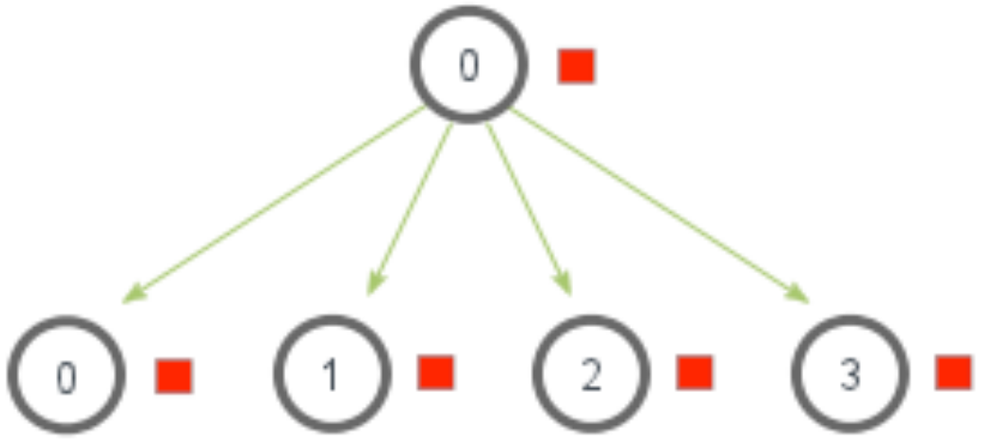
- ✓ See **http://mpitutorial.com/** for a complete description of all the MPI commands.

http://mpitutorial.com/

MPI: Basis Commands to share data

- ✓ Of course, it exists **commands to communicate between the processes**:
  - ✤ **mpi_bcast: send a data** from one process to all the other one

    ⇒call mpi_bcast(integer_to_send,1,mpi_integer,0, mpi_comm_world,ierr)

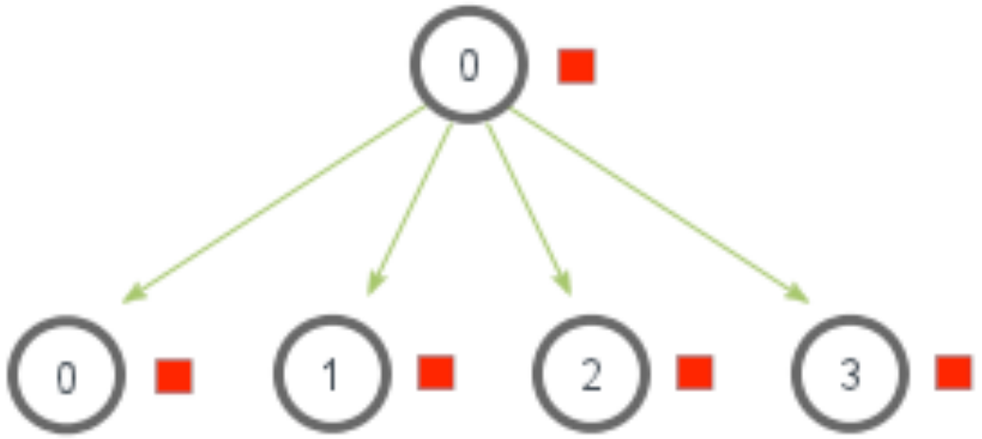


http://mpitutorial.com/

MPI: Basis Commands to share data

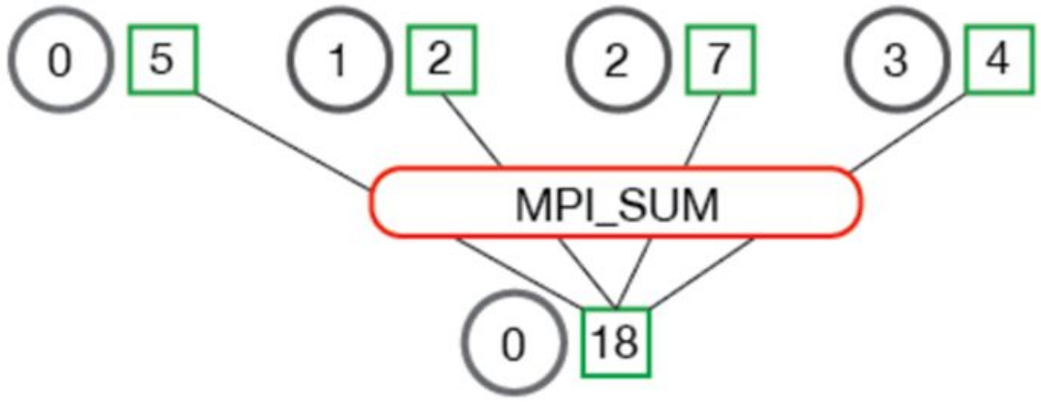
- ✓ Of course, it exists **commands to communicate between the processes**:
  - ✤ **mpi_bcast: send a data** from one process to all the other one

    ⇒call mpi_bcast(integer_to_send,1,mpi_integer,0, mpi_comm_world,ierr)



  - ✤ **mpi_reduce: take the same data** on all processes and apply an operation on them to obtain a unique value on the calling process

    ⇒call mpi_reduce(pi,pi_final,1, mpi_double_precision,mpi_sum,0,mpi_comm_world,ierr)

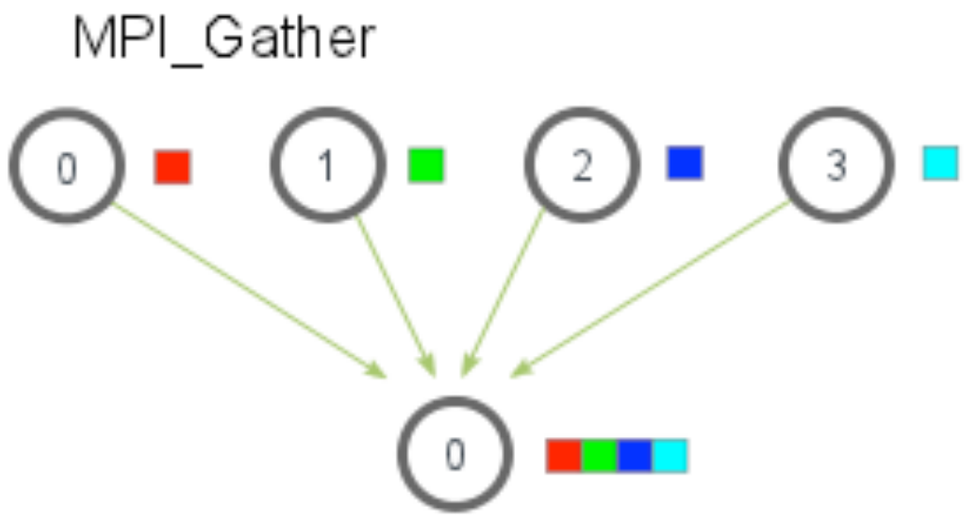


http://mpitutorial.com/

MPI: Basis Commands to share data

- ✓ Of course, it exists **commands to communicate between the processes**:
  - ✤ **mpi_gather**: **take the same data** on all processes and gather them in an array of size NPROCS on the calling process

    ⇒mpi_gather(pi,1,mpi_double_precision, valeurs_pi, 1, mpi_double_precision, 0, mpi_comm_world, ierr)

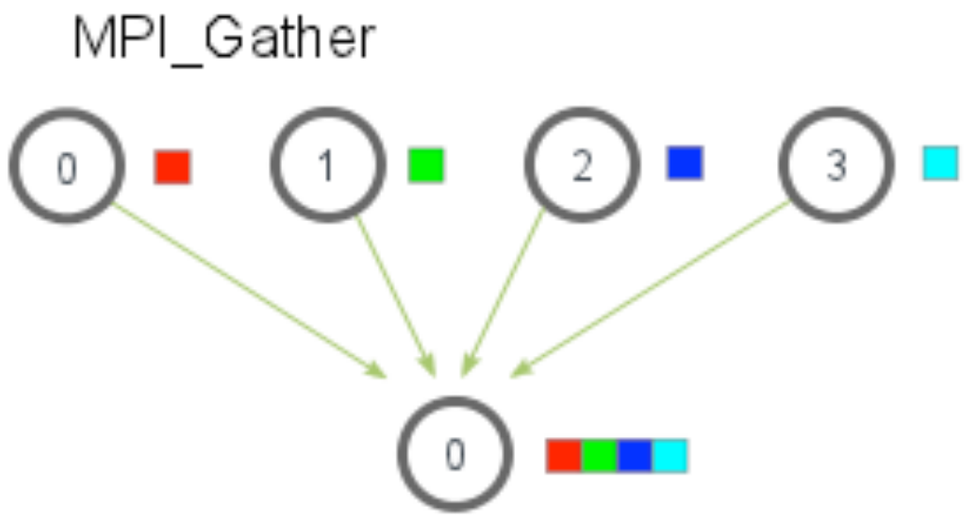


http://mpitutorial.com/

MPI: Basis Commands to share data

- Of course, it exists **commands to communicate between the processes**:
  - **mpi_gather**: **take the same data** on all processes and gather them in an array of size NPROCS on the calling process

    ⇒mpi_gather(pi,1,mpi_double_precision, valeurs_pi, 1, mpi_double_precision, 0, mpi_comm_world, ierr)



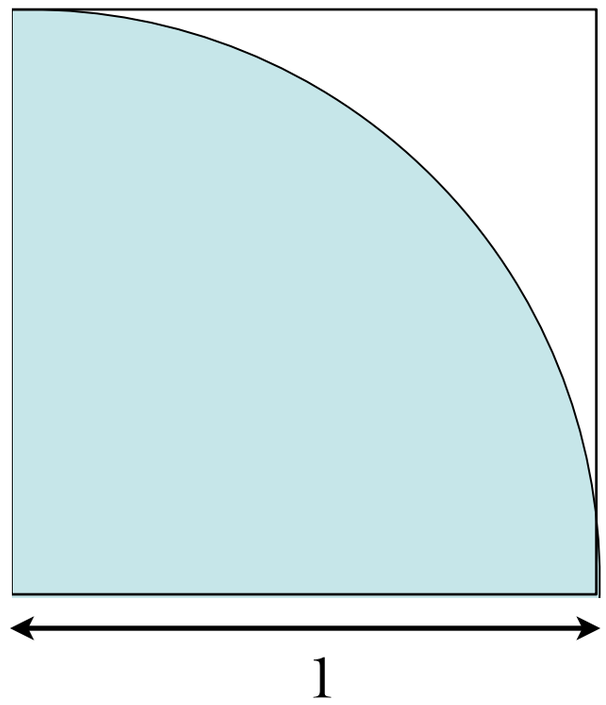
  - **mpi_barrier**: put a barrier in the code, the processes will only be able to continue the code when all the processes get to the barrier

http://mpitutorial.com/

✓ **Take the code of Anthony Scemama to calculate π and make it parallel.**

✓ How it works:

- ✤ take 2 random number x and y between 0 and 1
- ✤ calculate number=sqrt(x^2+y^2)
- ✤ if number is lower or equal to 1 do count=count+1
- ✤ π=4×count/count_total

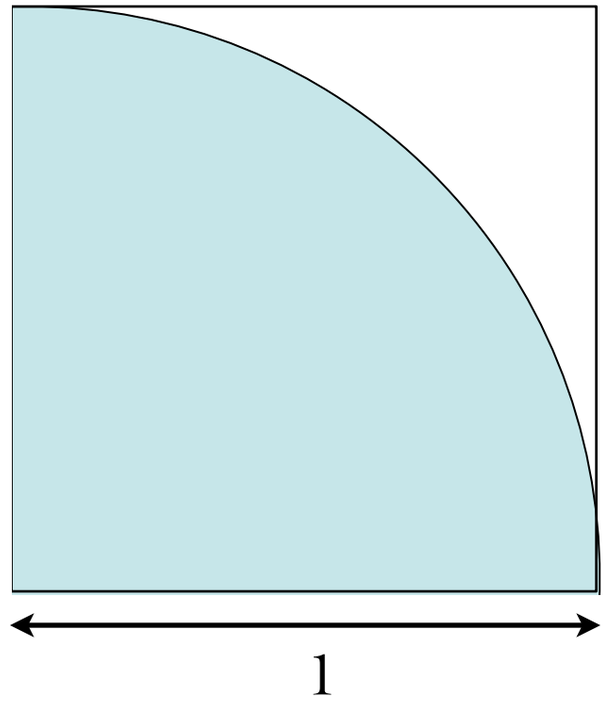



```
do i=1,nbr_pts

  call random_number(x)
  call random_number(y)

  somme=sqrt(x**2+y**2)

  if (somme.le.1) n_inside=n_inside+1

  n_total=n_total+1

  if (modulo(i,10)==0) then
    write(*,*) 4.0d0* dble(n_inside)/dble(n_total)
  endif
enddo

write(*,*) "Final pi"
write(*,*) 4.0d0* dble(n_inside)/dble(n_total)
```

- ✓ **Take the code of Anthony Scemama to calculate π and make it parallel.**

- ✓ How it works:
  - ✤ take 2 random number x and y between 0 and 1
  - ✤ calculate number=sqrt(x^2+y^2)
  - ✤ if number is lower or equal to 1 do count=count+1
  - ✤ π=4×count/count_total

- ✓ The serial solution is in *Calculate_pi.f90*



Fourth Exercise of Day 7

- ✓ **Take your MD code and make it a parallel-tempering code**

- ✓ I give you a clue: first, do not think about exchanges between processes. Just try to run in parallel *N* simulations, each one at a different temperature.

Thanks for your attention