LIFE SCIENCE: USING HPC FOR INSIGHT INTO BIOMOLECULAR FUNCTION

Thomas E Cheatham III, University of Utah, Salt Lake City, UT
Erik Lindahl, Stockholm University, Stockholm, Sweden
tec3@utah.edu
erik.lindahl@scilifelab.se

Who are we? What type of careers have we had since we were students?

Why are we working with HPC? What are the challenges in our field?

How do large HPC codes evolve? Who develops them?

How do you write fast & parallel programs for real problems?

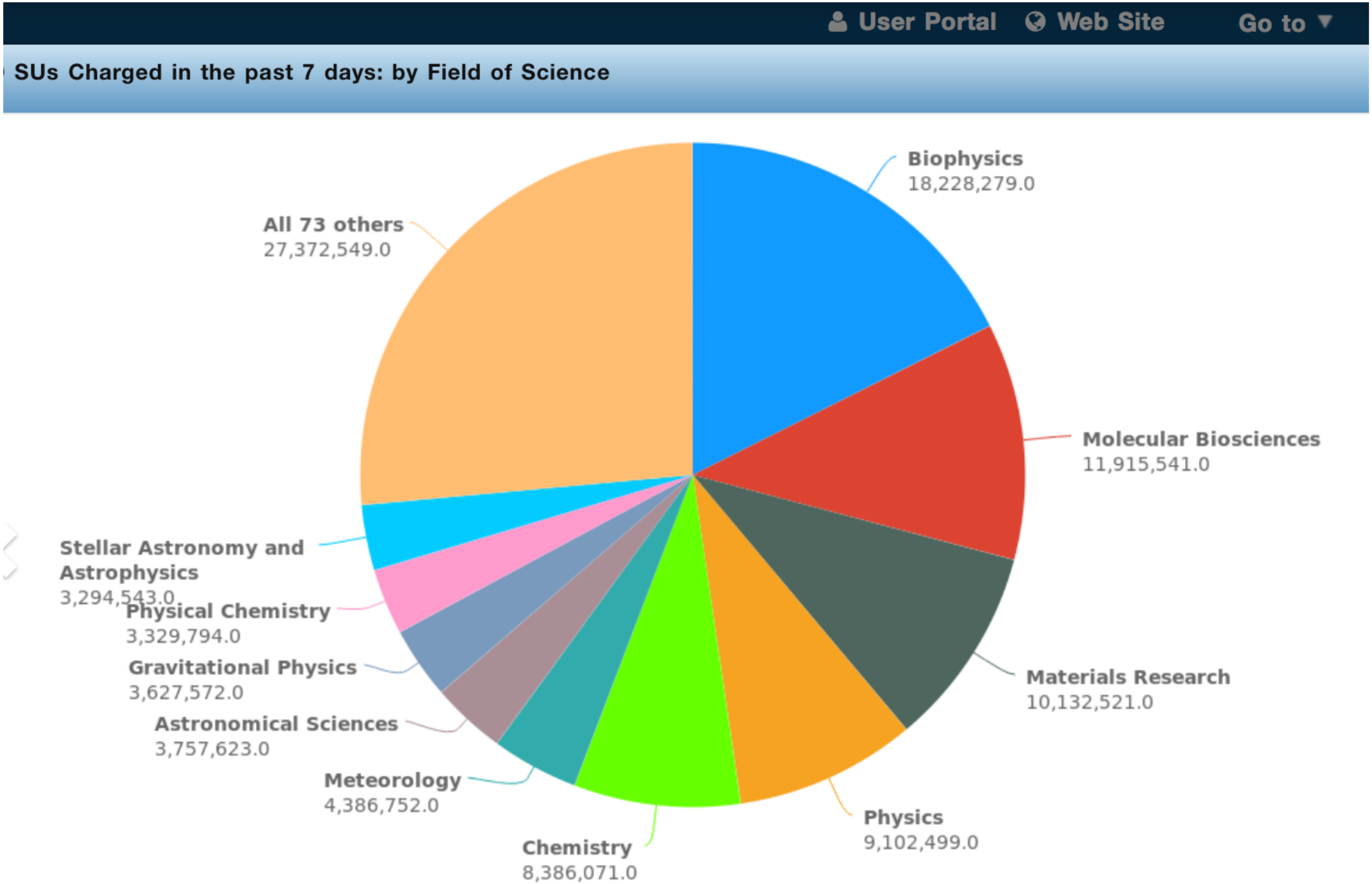
How do you extract more parallelism from an algorithm?

How do you evolve the code with new hardware?

How will we use next-generation extremely large machines?

Why multiple life sciences lectures?

XSEDE usage over the past 7 days (prior to 2015 iHPC-SS)



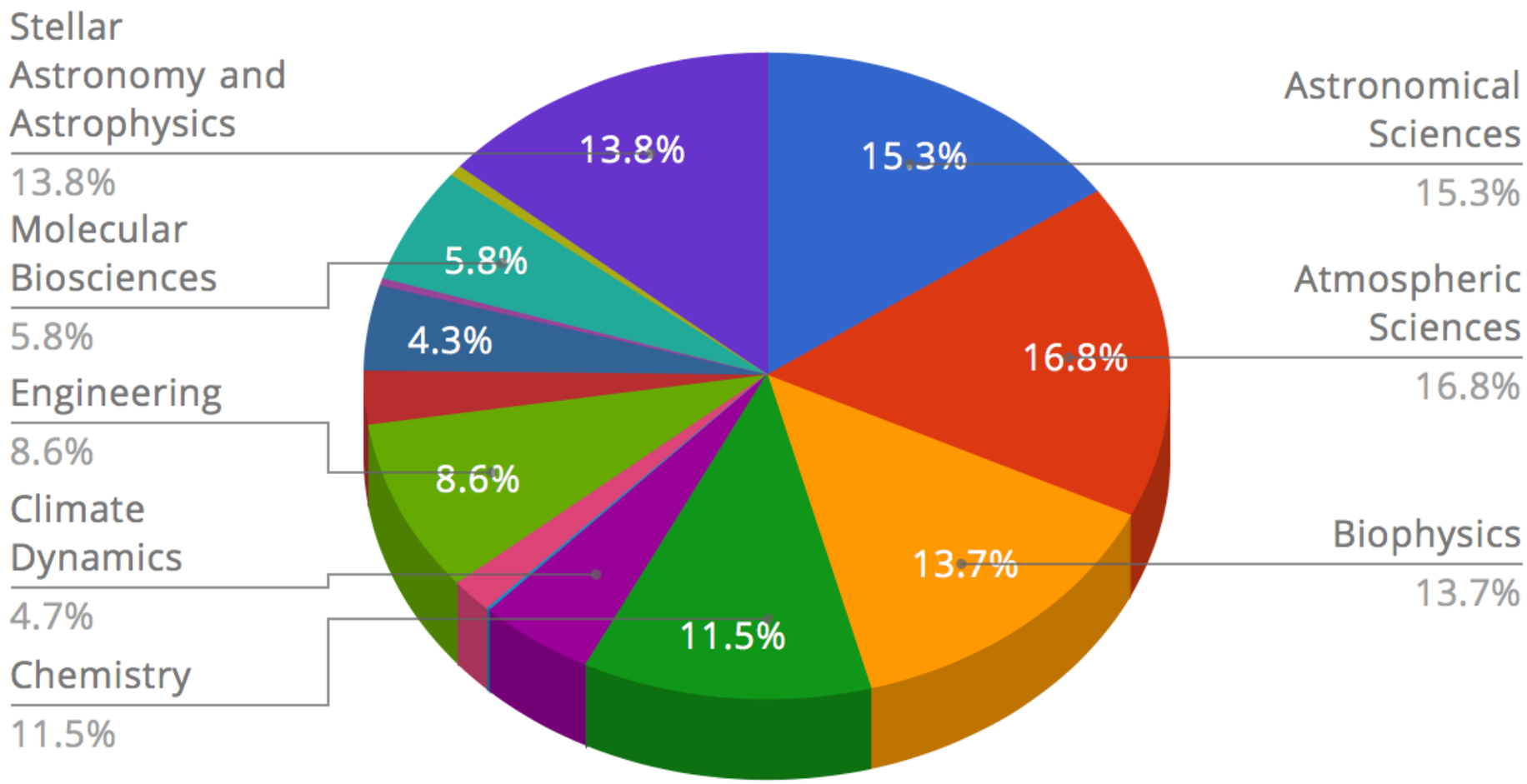
BLUE WATERS

SUSTAINED PETASCALE COMPUTING

GPU nodes

Ensembles of molecular dynamics engines for assessing force fields, conformational change, and free energies of proteins and nucleic acids (**Jobs:**38) **PI:** Thomas Cheatham, University of Utah	XK	28400	310,293.43
Predicting protein structures with physical petascale molecular simulations (**Jobs:**21) **PI:** Ken Dill, SUNY at Stony Brook	XK	10240	276,988.80

CURRENT RUNNING JOBS BY SCIENCE AREA



AMBER (Cheatham) & GROMACS (Lindahl)

ambermd.org

gromacs.org

history, code development, philosophy, approach, synergies, differences, challenges, futures, lessons learned

Tom:

Thomas Cheatham, III
Professor of Medicinal Chemistry, College of Pharmacy
Director, Center for High Performance Computing, University of Utah 7/1/14-

1988-1997	1990-1997	1997-2000	2000-present
programmer/analyst	graduate school	NIH postdoc	Res Asst Prof - Professor
Harvard U (DAS/ACS)	NSF centers	NIH only	CHPC+AAB→TG→XSEDE→BW
CM-2, CM-5, MasPar	T3D, T3E, Crays	beowulf, IBM SP-2	(many || + GPUs)

DOE Summer Institute @ Los Alamos (vector)
PSC Summer Institute in parallel computing
PSC Workshop on Hetereogeneous Computing
PSC AMBER Workshop (Teacher)

Allocations committee, chair
TeraGrid Science Advisory Board, chair
XSEDE User Advisory Committee, chair
XSEDE SMT, SAB
Blue Waters SETAC

Erik:

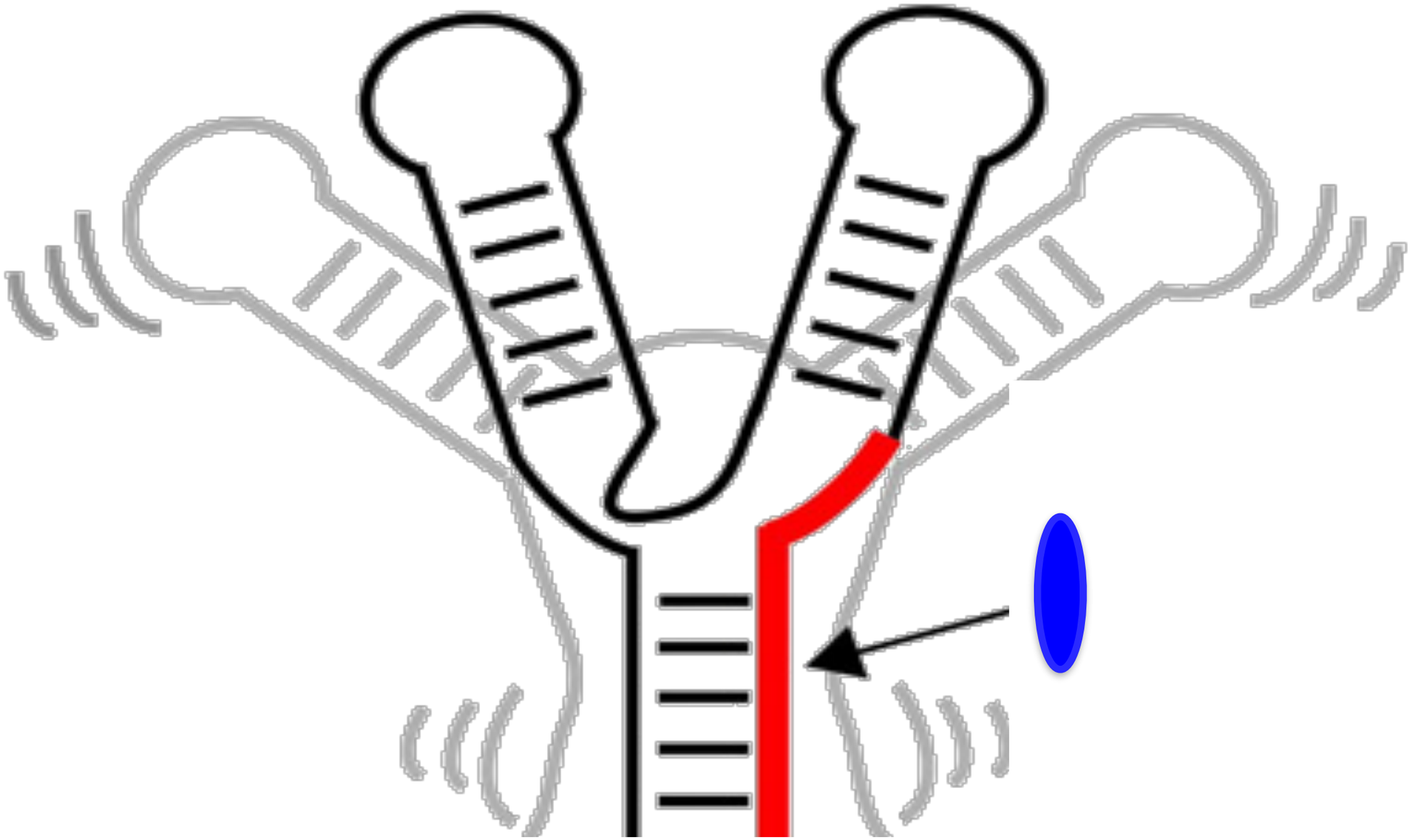
Erik Lindahl
Professor of Biophysics, Stockholm University
Professor of Theoretical Biophysics, KTH Royal Institute of Technology
Vice Director, Swedish e-Science Research Center

1996-2001	2001	2002-2003	2004	2004-present
grad. school	Groningen Univ.	Stanford Univ.	Inst. Pasteur	Faculty
KTH, Sweden	Local	Local (NIH)	Local+SNIC(SE)	SNIC+PRACE
IBM SP2, PPC	Beowulf	Linux/x86 clusters, PS3, F@H		Everything+CUDA+OCL

Board of Directors, Swedish National Infrastructure for Computing
Vice chair, PRACE Scientific Steering committee
Platform director, Bioinformatics, SciLifeLab
Lead scientist, BioExcel Center of Excellence for biomolecular computation
Vice director, Swedish e-Science Research Center
Swedish Research Council

What do we want to do? Accurately model the structure and dynamics of molecules in their native environment

(i.e. follow the motions of the atoms subject to an energetic potential or "force field" as a function of time…)



Accurate modeling of molecules requires:

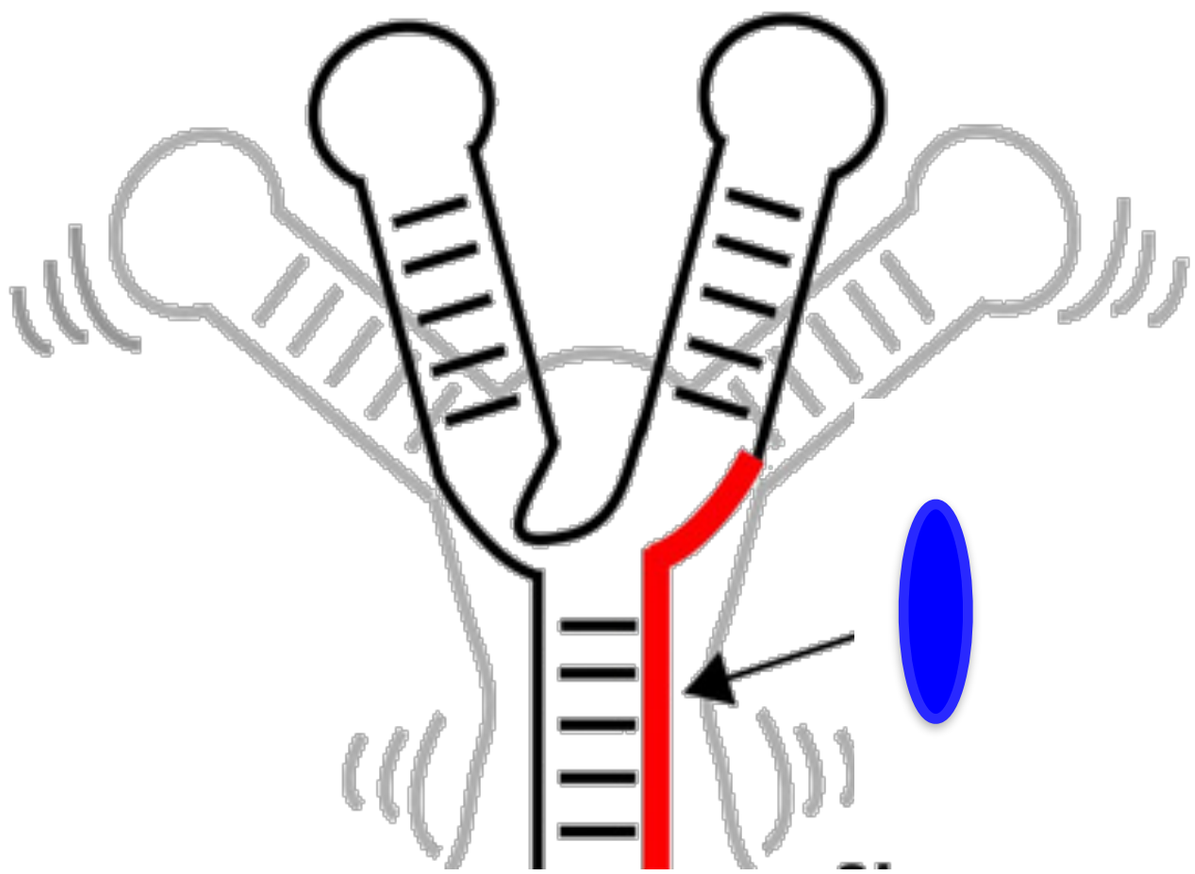
accurate and fast simulation methods

validated RNA, protein, water, ion, and ligand "force fields"

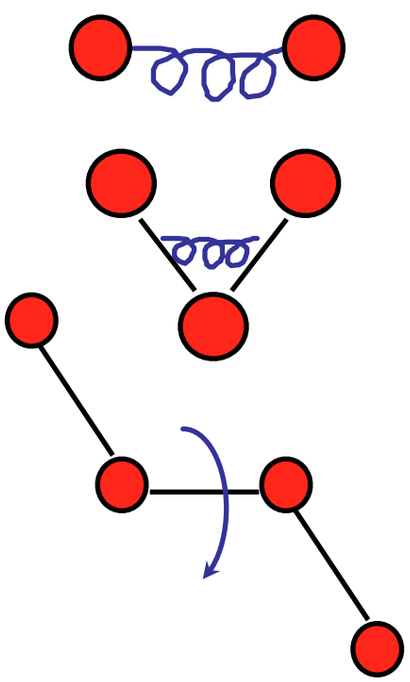
"good" experiments to assess results

dynamics and complete sampling: (convergence, reproducibility)

Question: Is the movement real or artifact?



What is a molecular mechanical "force field" ?



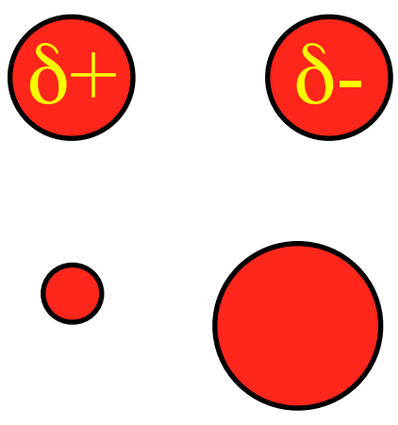
bonds

$$U = \sum_{bonds} k_r \left(r - r_{eq}\right)^2 + \sum_{angles} k_\theta \left(\theta - \theta_{eq}\right)^2$$

angles

dihedrals

$$+ \sum_{dihedrals} \sum_{\eta} \frac{V_\eta}{2} \left[1 + \cos(\eta\phi - \gamma)\right]$$



electrostatics

van der Waals

$$+ \sum_{i=1}^{atoms} \sum_{j>i}^{atoms} \left[\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^{6}} + \frac{q_i q_j}{\varepsilon r_{ij}} \right]$$

Key assumption: transferability of bonding

molecular simulation / molecular dynamics

…is not new,
has a rich history,
and is largely solved (?)

BBC

News | Sport | Weather | Capital | Culture | Autos | TV | Radio

NEWS SCIENCE & ENVIRONMENT

Home | US & Canada | Latin America | UK | Africa | Asia | Europe | Mid-East | Business | Health | Sci/Environment | Tech | Entertainment

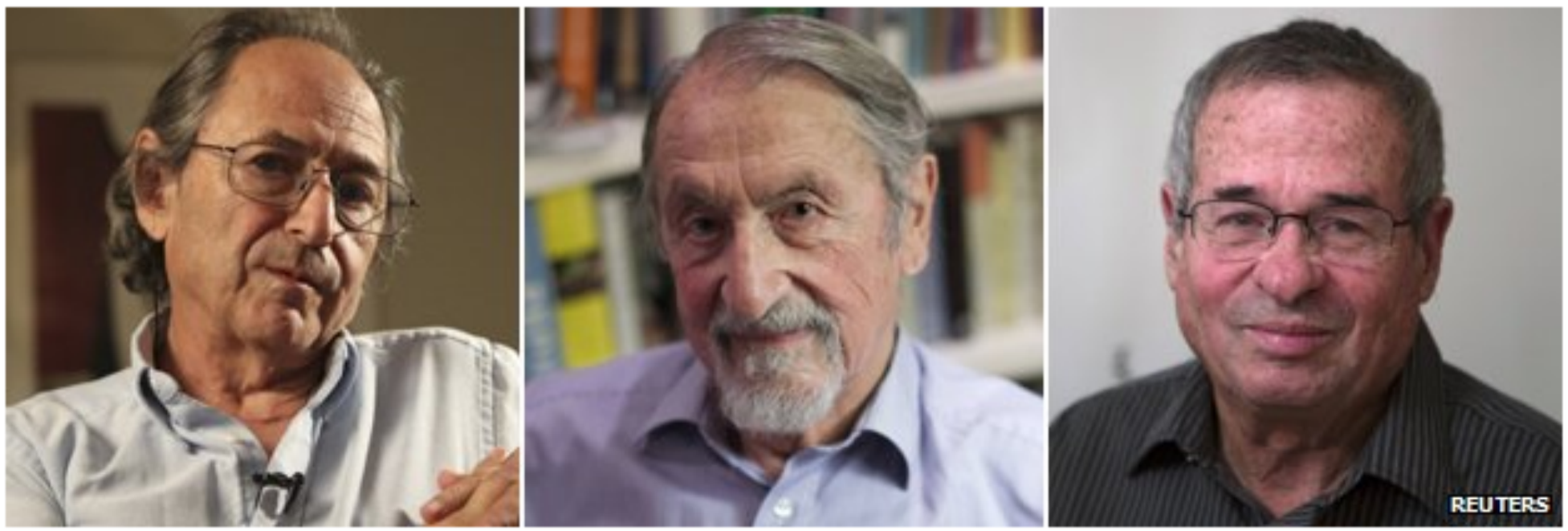
Computer chemists win Nobel prize

9 October 2013

By James Morgan and Jonathan Amos

Science reporters, BBC News

"for the development of multiscale models for complex chemical systems

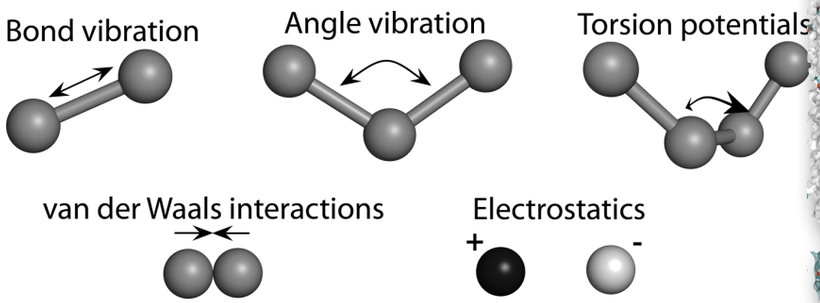


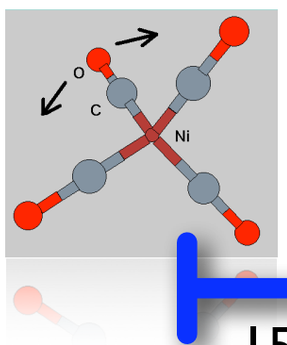
The work of Levitt, Karplus and Warshel has spawned a worldwide industry

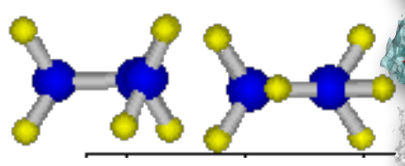
The Nobel Prize in chemistry has gone to three scientists who "took the chemical experiment into cyberspace".

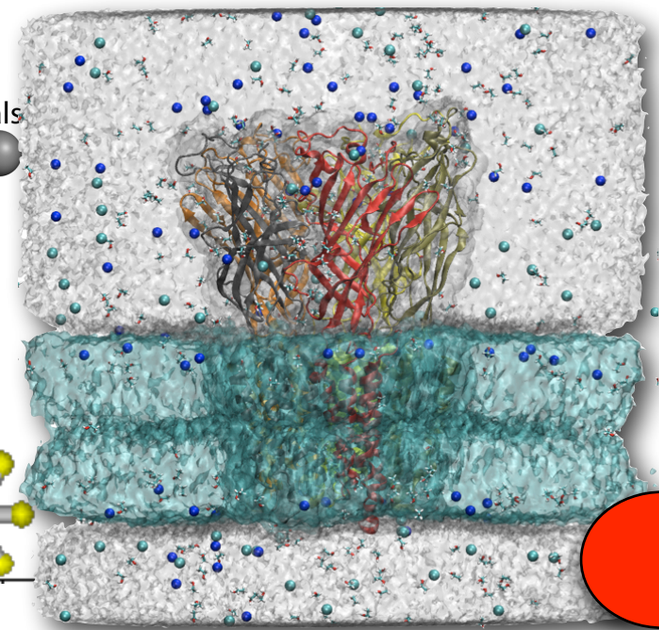
Michael Levitt: ""It's sort of nice in more general terms to see that computational science, computational biology is being recognized," he added. "It's become a very large field and it's always in some ways been the poor sister, or the ugly sister, to experimental biology."

Molecular simulations









Experiments

Efficient averaging

Less detail

Where we need to be

10^{-15} Physics 10^{-12} 10^{-9}s Chemistry 10^{-6}s 10^{-3}s 10^{0}s Biology 10^{3}s

Where we are

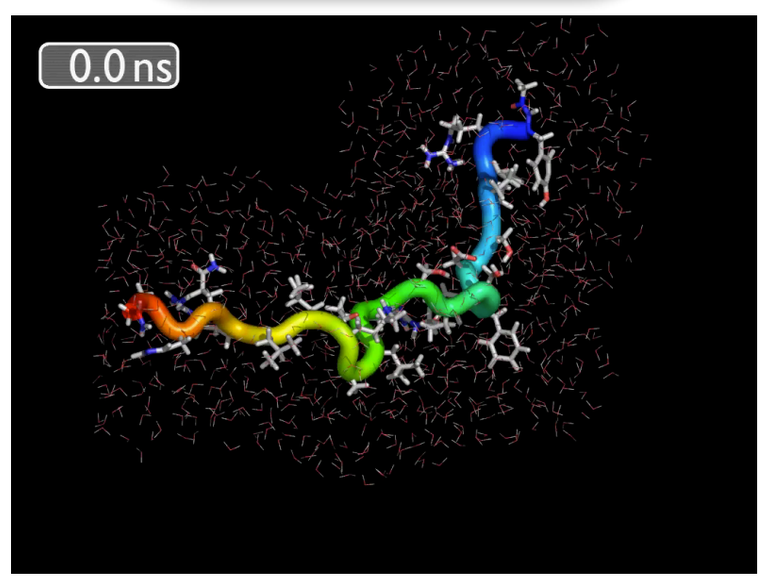
Where we want to be

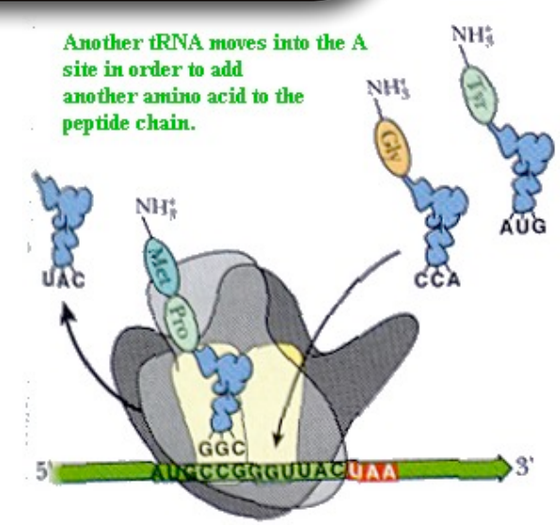
Simulations

Extreme detail

Sampling issues?

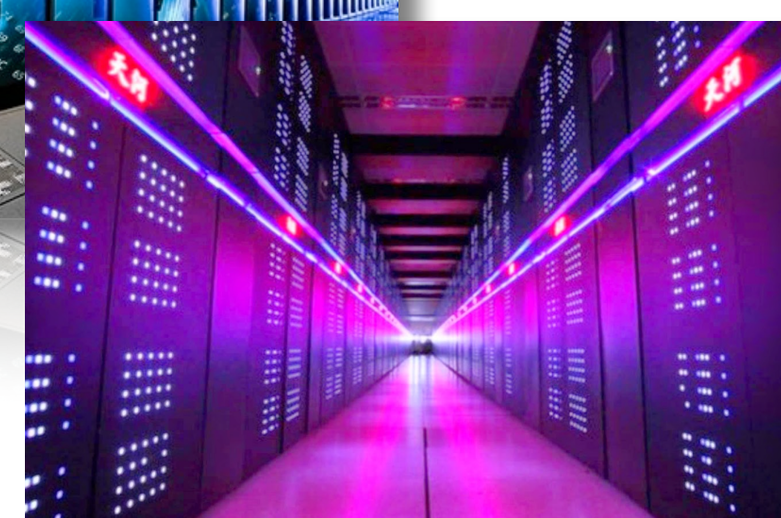
Parameter quality?





Challenge 1:

Larger machines have mostly enabled larger systems, not longer simulations



When we started, programs could use O(10) cores

How do we develop these codes and problems to use O(10,000+) cores?

An example of code evolution

~1978 - present

amber

Assisted Model Building with Energy Refinement

An example of code evolution

~1978 - present

amber

Assisted Model Building with Energy Refinement

code vs. force field

the setup and calculation engines

the parameters and potentials

An example of code evolution

~1978 - present

amber

Assisted Model Building with Energy Refinement

code vs. force field

the setup and calculation engines

the parameters and potentials

- not really a professional code (some experts, some beginners)
- not really software engineered (parts were, like GPU code, optimizations)
- it is continually evolving; one of the first “community codes”…
- development efforts are not directly funded (except maybe GPU)

An example of code evolution

~1978 - present

amber

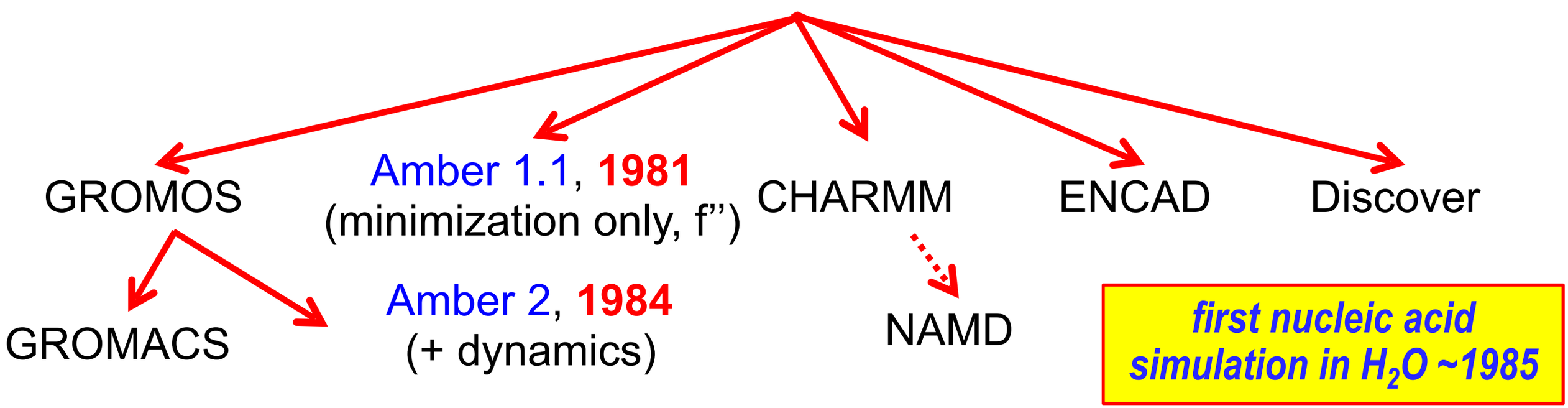
Assisted Model Building with Energy Refinement

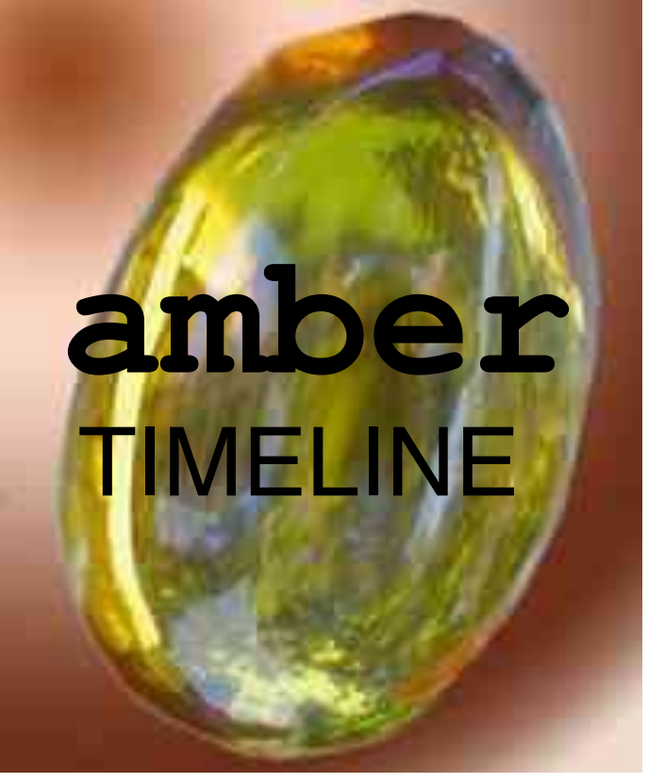
code vs. force field

late 60's: CFF (consistent force field) + early code
{Warshel, Levitt, Lifson}

first protein simulation ~1975

1978: Bruce Gelin thesis @ Harvard {Karplus}





1986: amber3

ΔG, QM/MM, non-additivity

1989: amber3a

code cleanup, bug fixes
increased performance, portability
vectorization, || on hypercube,
shared memory
Intel Paragon 1/3 speed of Y-MP

1990-1994: ~~SPASMS~~

?

(blue matter)
~2004

Special purpose?
MD-GRAPE
SFE, Tera, …

1991: amber4.0

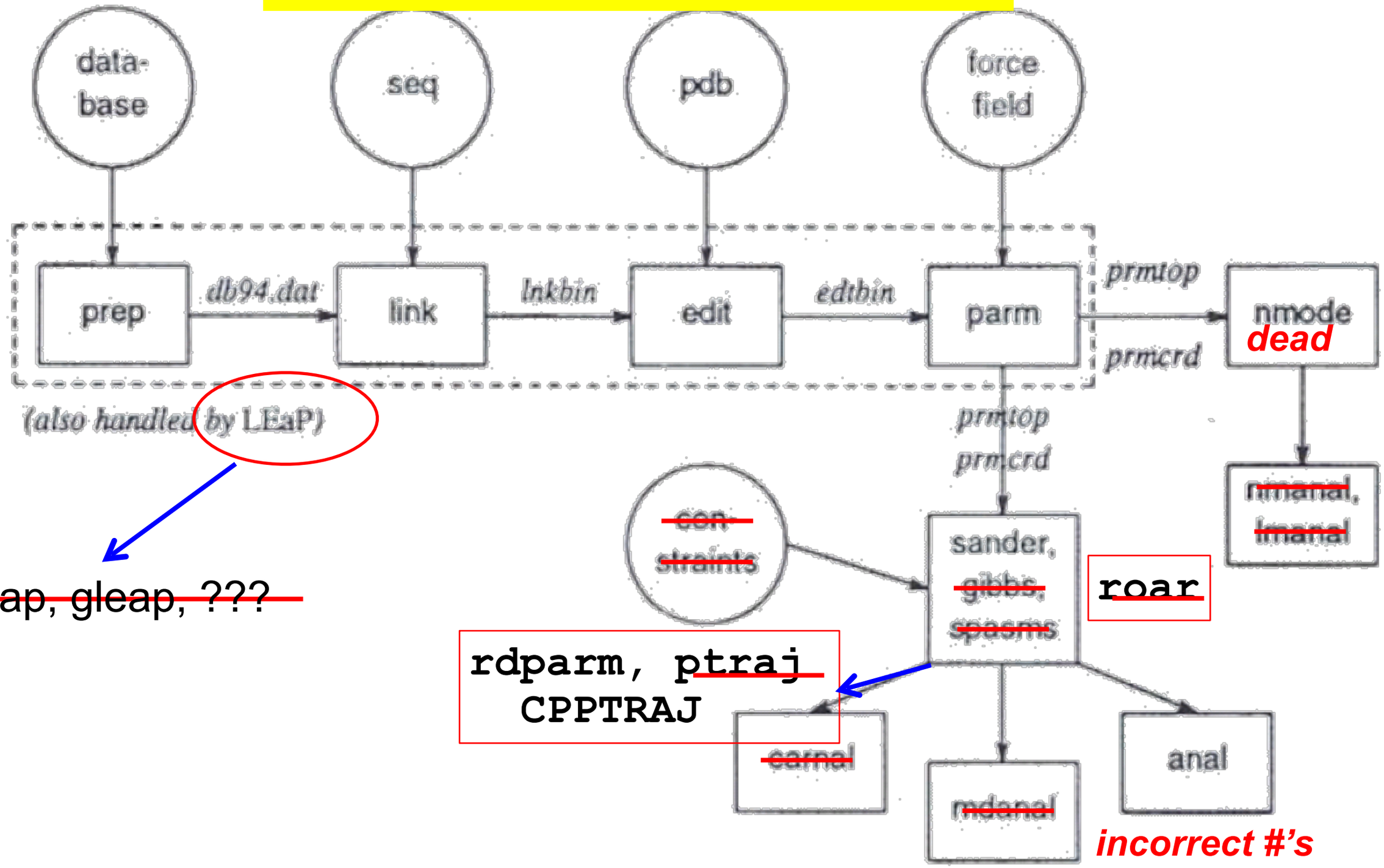
NMR refinement, normal modes, ΔG
serious code bifurcation
|| message passing
(TCGMSG, PVM, MPI, …)

1994: amber4.1

particle mesh Ewald ☺️
more shared memory, MPI only
`#ifdef MPI`

D.A. Pearlman et. al. / Computer Physics Communications 91 (1995) 1–41

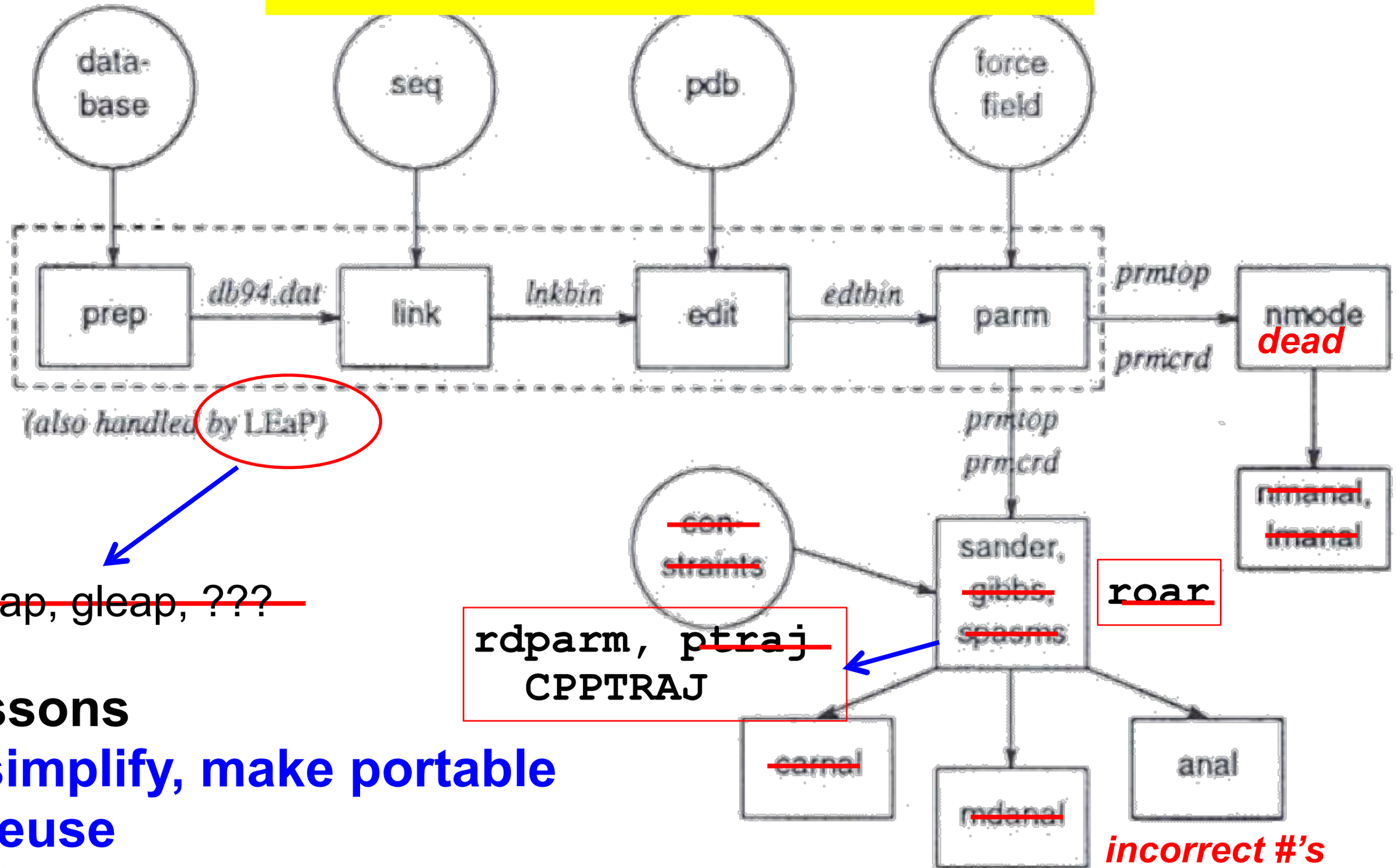
…evolution of AMBER 4.1 codes



sleap, gleap, ???

D.A. Pearlman et. al. / Computer Physics Communications 91 (1995) 1–41

…evolution of AMBER 4.1 codes



~~sleap, gleap, ???~~

rdparm, ~~ptraj~~
CPPTRAJ

~~roar~~

Lessons

- simplify, make portable
- reuse
- if development stops, code dies
- replace functioning code with new code most often fails

early days: **ftp repository, makefiles (many), MACHINEFILE**

4.1-7.0: **CVS, C memory allocation move to F90, makefiles**
compile script recognizing MACHINEFILE
(fight w/ compiler for giganet vs. myrinet vs. …)

simplify, unify (as machines are becoming homogeneous)
drop vectorization, drop shared memory, drop machine specific opts

8.0: (2004) **introduce fast engine pmemd, configure scripts**

focus on fewer compilers: gnu, intel, pgi, pathscale
minimize #ifdefs to infrequently used code paths

~1995-2005: homogeneous hardware, standard MPI ||

…to present day…

accelerators

optimize ||

special purpose

floating point precision: single vs. double vs. mixed/fixed

early days: **ftp repository, makefiles (many), MACHINEFILE**

4.1-7.0: **CVS, C memory allocation move to F90, makefiles**
compile script recognizing MACHINEFILE
(fight w/ compiler for giganet vs. myrinet vs. …)

simplify, unify (as machines are becoming homogeneous)
drop vectorization, drop shared memory, drop machine specific opts

8.0: (2004) **introduce fast engine pmemd, configure scripts**

focus on fewer compilers: gnu, intel, pgi, pathscale
minimize #ifdefs to infrequently used code paths

10.0: (2008) **AmberTools (open source), OpenMP**
separate configure for AmberTools, sander, pmemd

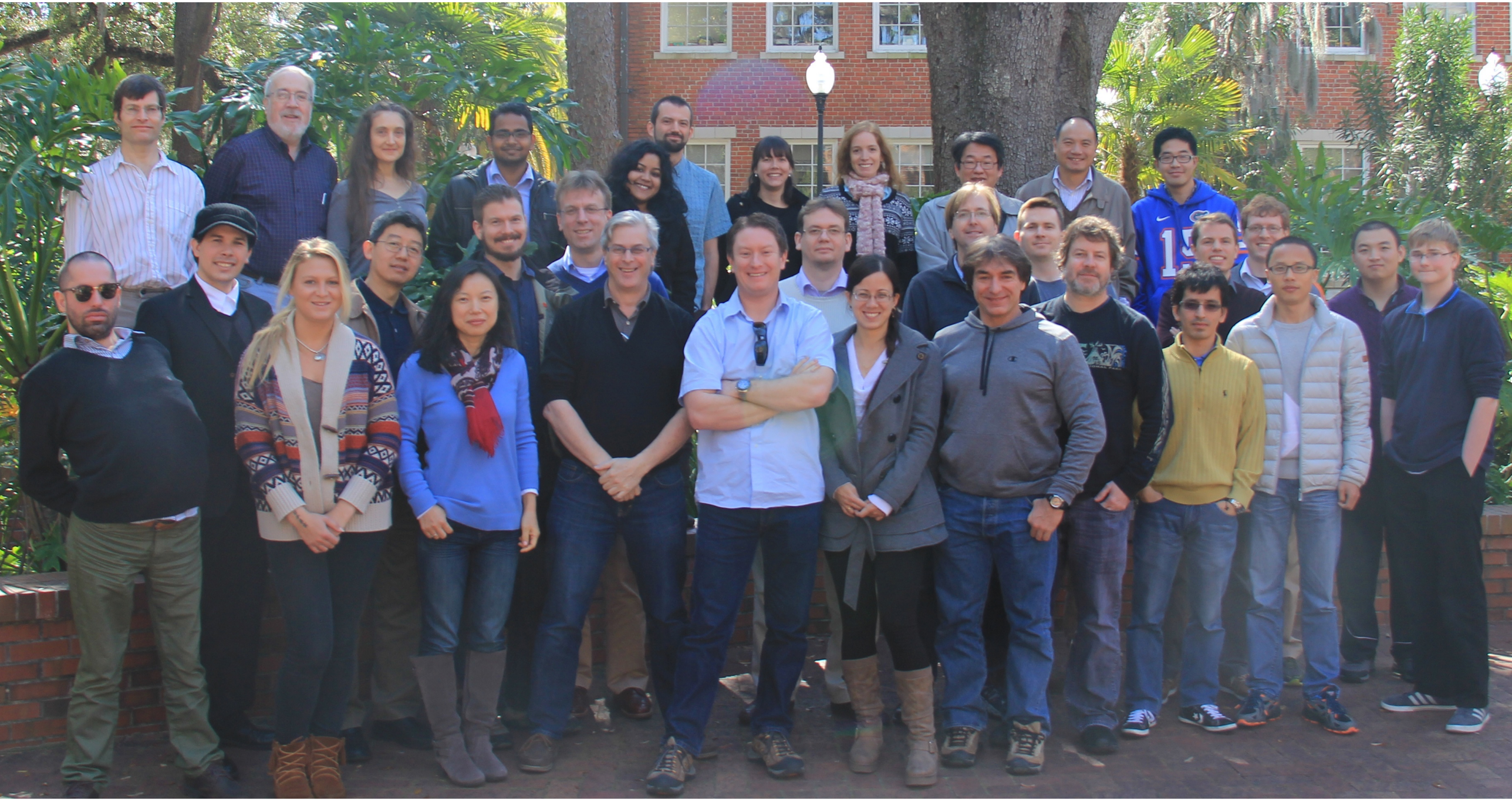
11:0: (2010) **git tree, full F90, make depend**

Challenge: building/patching the code!

12.0: (2012) **Unified "configure" script, easy compile, …**
!!! automatic bug patching !!!

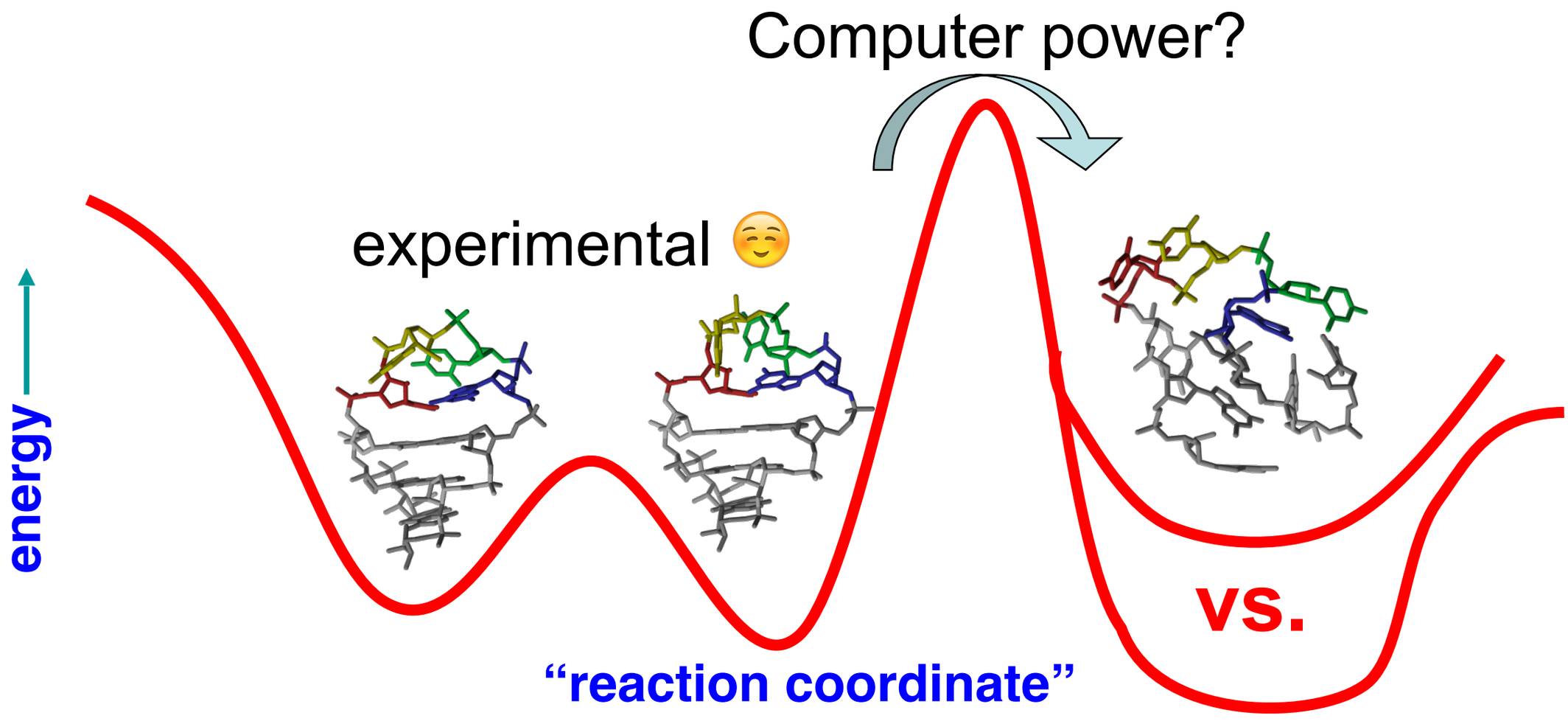
14.0: (2012) **GPU 1.23x, multi-GPU on node ||, …**

AMBER 16 (released ~May 2016)

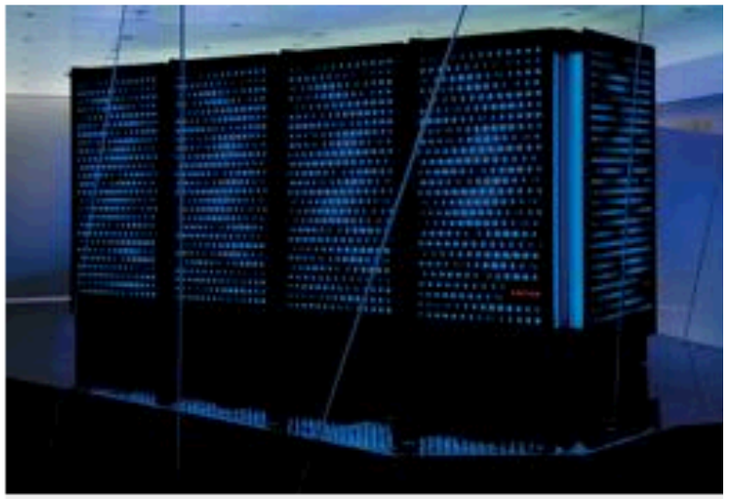


Challenge 2:

are the force fields reliable?
(structure, dynamics, free energies)
can we fully sample the conformational ensemble?
(convergence, reproducibility)



How to fully sample conformational ensemble?



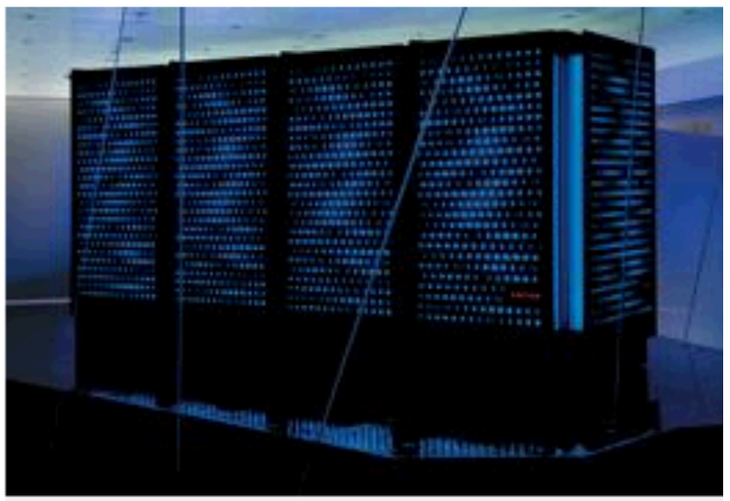
Simulating protein movements using Anton could aid drug design.

SCIENCE/AAAS

brute force – long contiguous in time MD
requires: special purpose / unique hardware

D.E. Shaw's Anton machine

How to fully sample conformational ensemble?

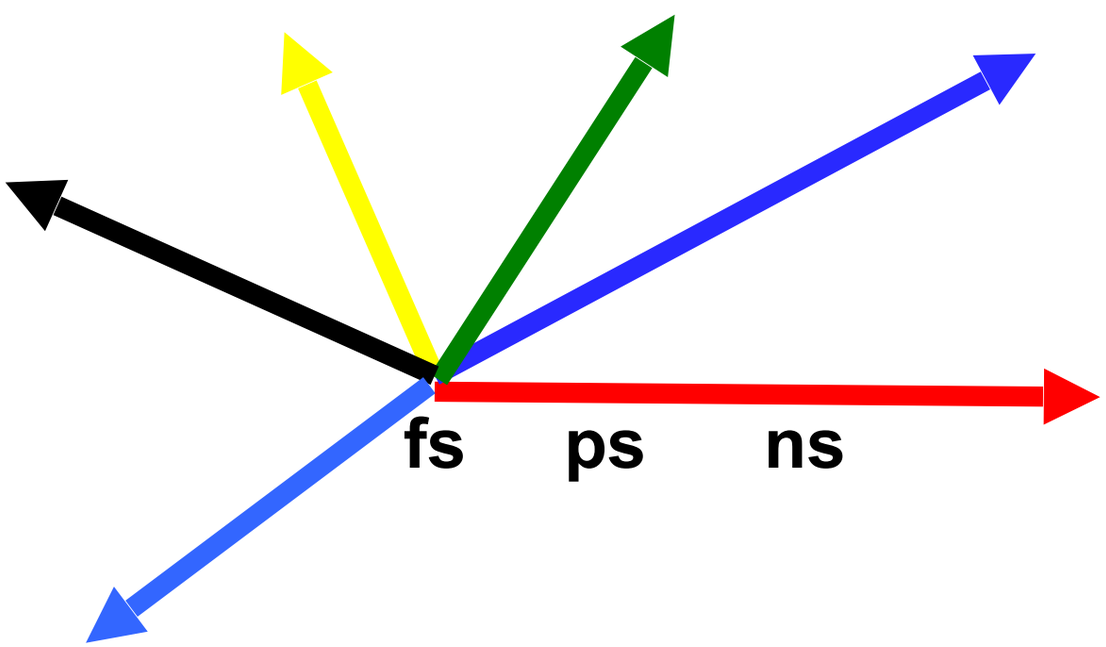


Simulating protein movements using Anton could aid drug design.

SCIENCE/AAAS

brute force – long contiguous in time MD
requires: special purpose / unique hardware

D.E. Shaw's Anton machine



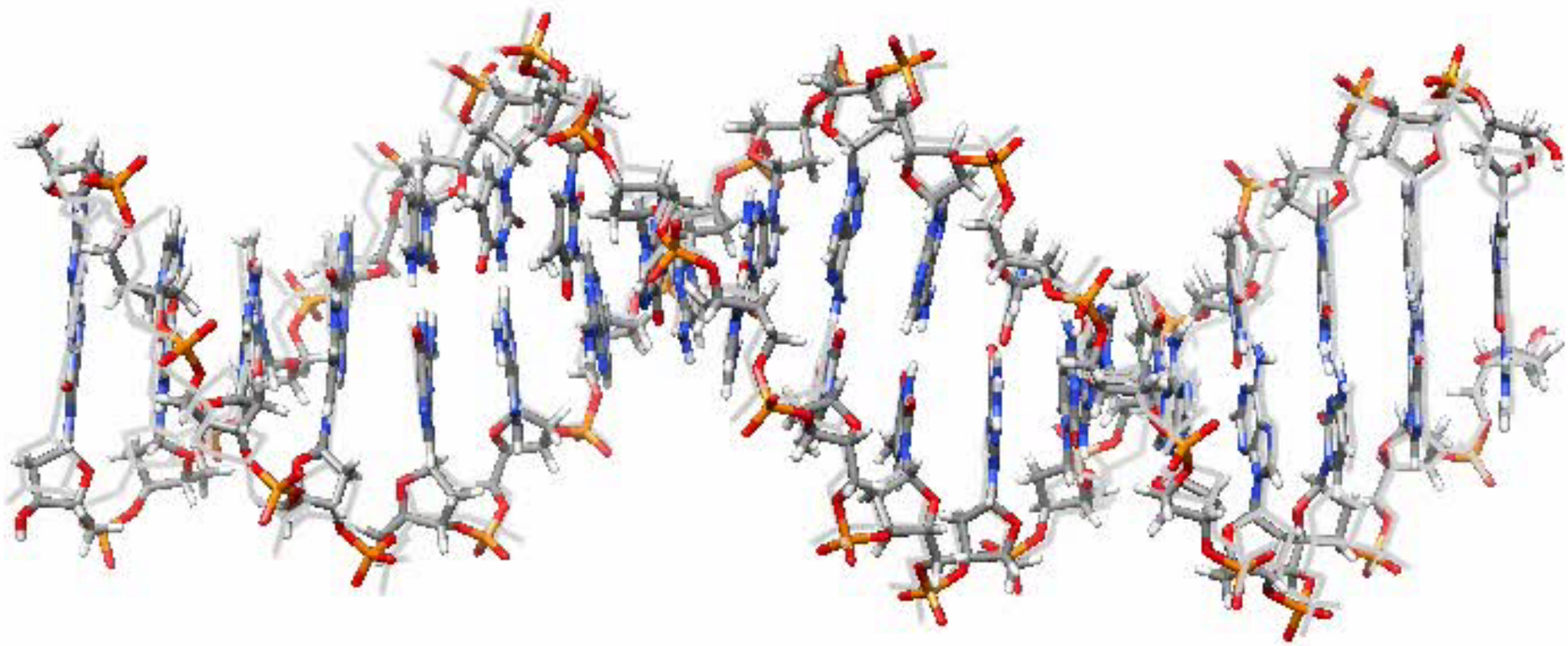
ensembles of independent simulations

AMBER on GPUs

220 ns/day!

Convergence, force field and salt dependence in simulations of nucleic acids

d(GCACGAACGAACGAACGC) – Anton vs. GPUs



2 ns intervals (10 ns running average), render every 5th frame: ~10 us total time

How to test for convergence between two simulations?

- Aggregate independent runs into a single trajectory
- Calculate principal components and/or clustering
- Project principal components independently on each separate run, compare cluster populations between individual runs
- Visualize results

2013

PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data

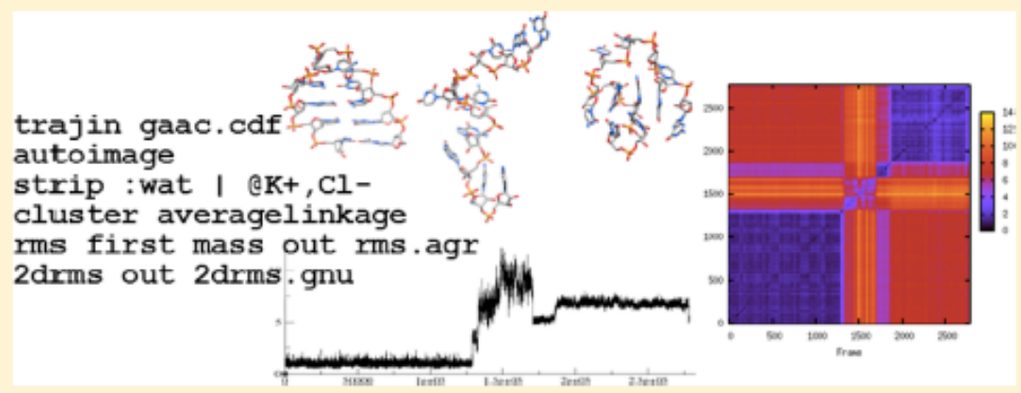
Daniel R. Roe* and Thomas E. Cheatham, III*

Department of Medicinal Chemistry, College of Pharmacy, 2000 South 30 East Room 105, University of Utah, Salt Lake City, Utah 84112, United States

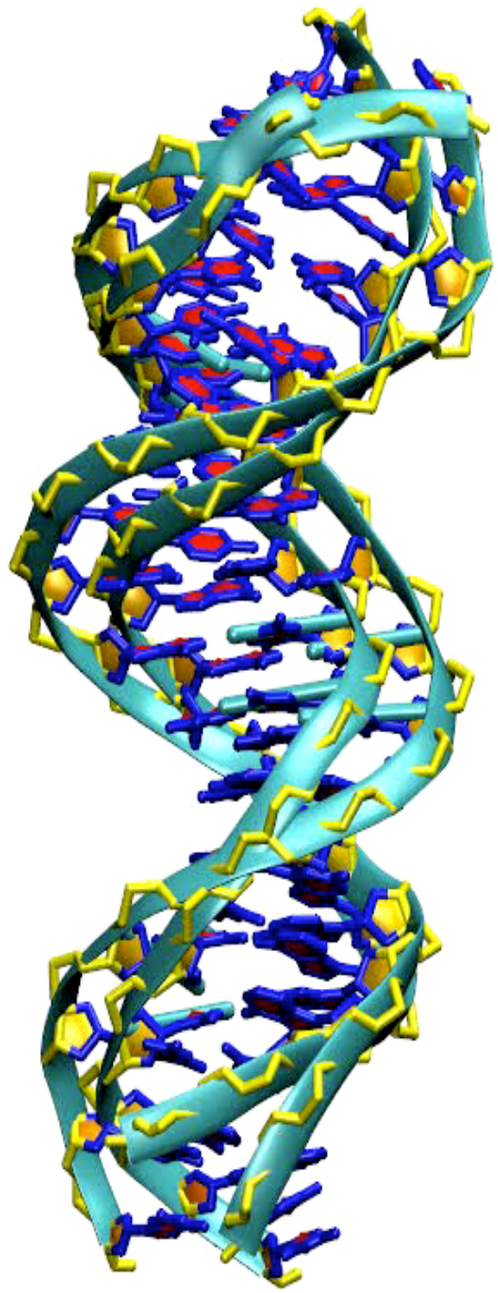
JCTC Journal of Chemical Theory and Computation

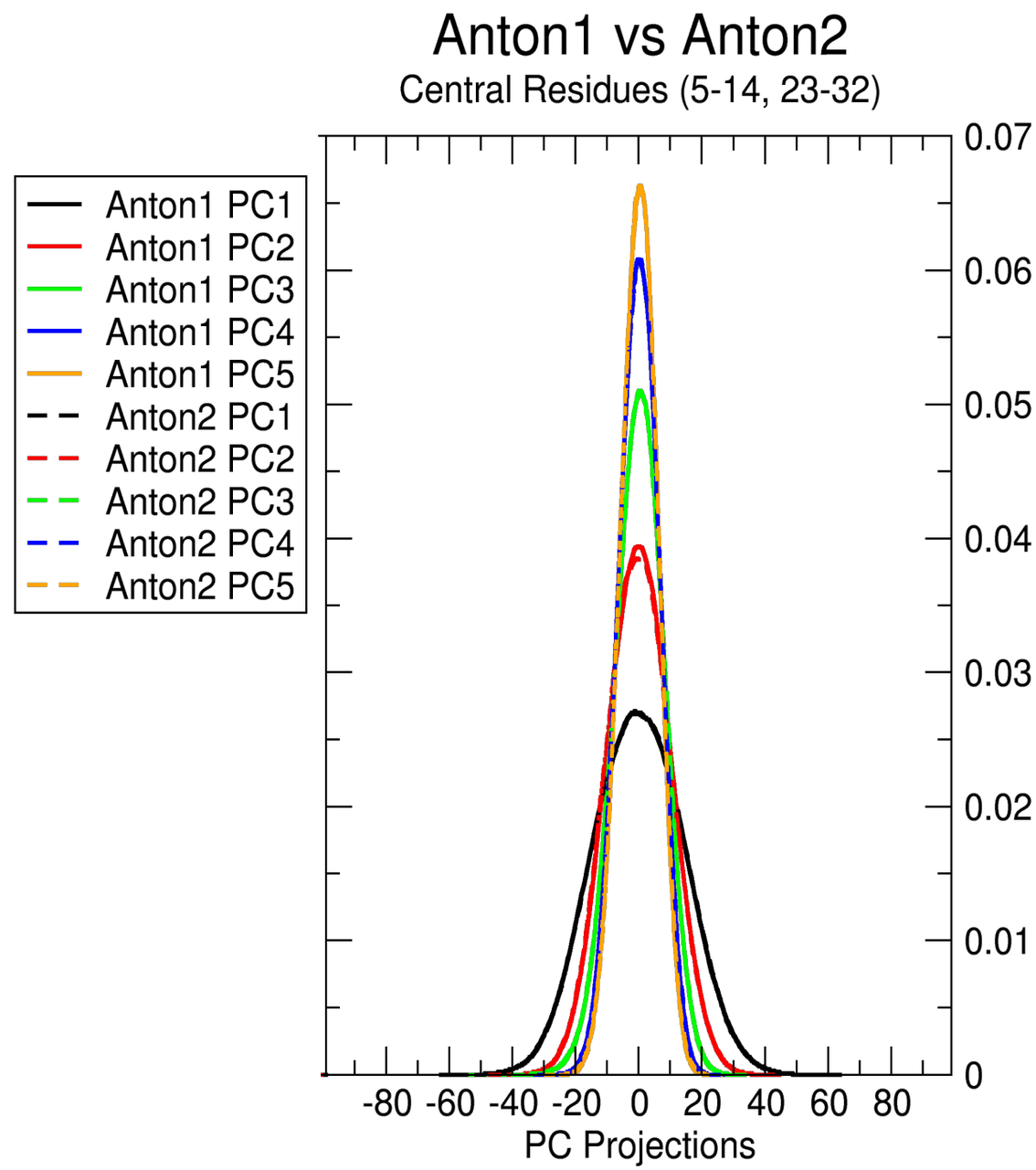
Supporting Information

ABSTRACT: We describe PTRAJ and its successor CPPTRAJ, two complementary, portable, and freely available computer programs for the analysis and processing of time series of three-dimensional atomic positions (i.e., coordinate trajectories) and the data therein derived. Common tools include the ability to manipulate the data to convert among trajectory formats, process groups of trajectories generated with ensemble methods (e.g., replica exchange molecular dynamics), image with periodic boundary conditions, create average structures, strip subsets of the system, and perform calculations such as RMS fitting, measuring distances, B-factors, radii of gyration, radial distribution functions, and time correlations, among other actions and analyses. Both the PTRAJ and CPPTRAJ programs and source code are freely available under the GNU General Public License version 3 and are currently distributed within the AmberTools 12 suite of support programs that make up part of the Amber package of computer programs (see http://ambermd.org). This overview describes the general design, features, and history of these two programs, as well as algorithmic improvements and new features available in CPPTRAJ.

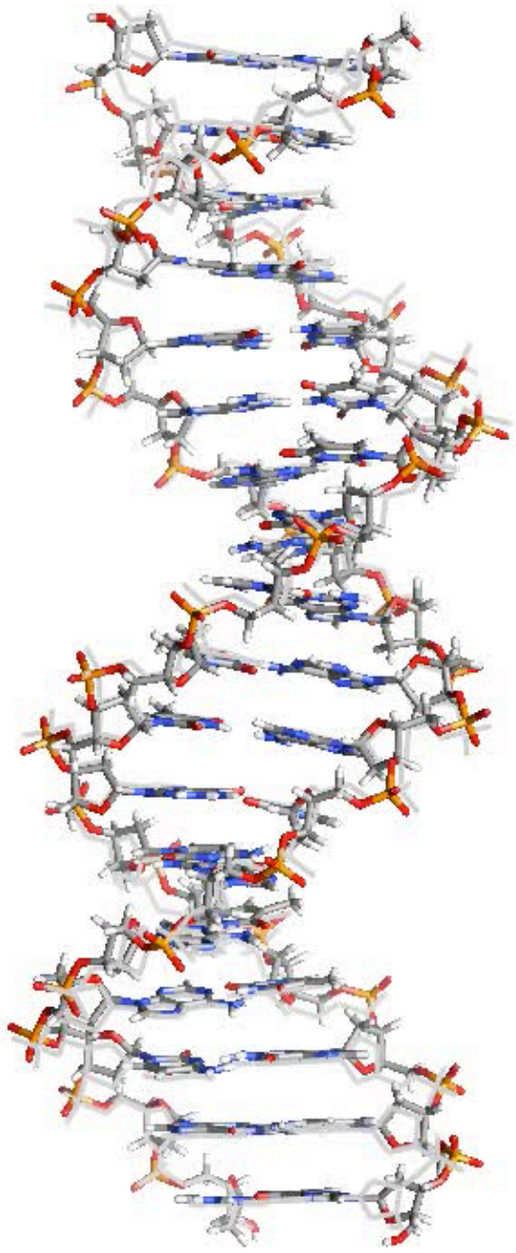


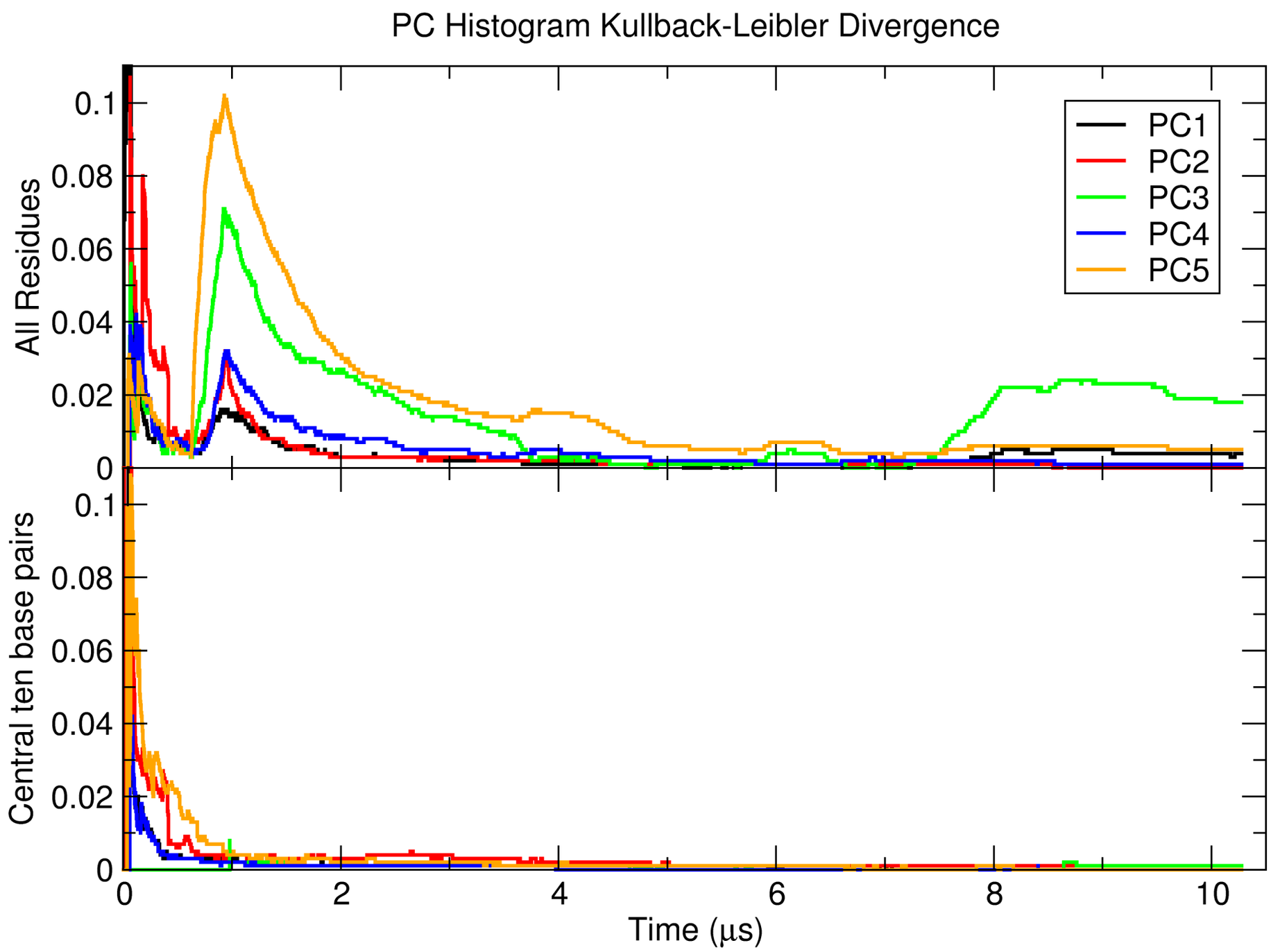
Test for convergence within and between simulations…



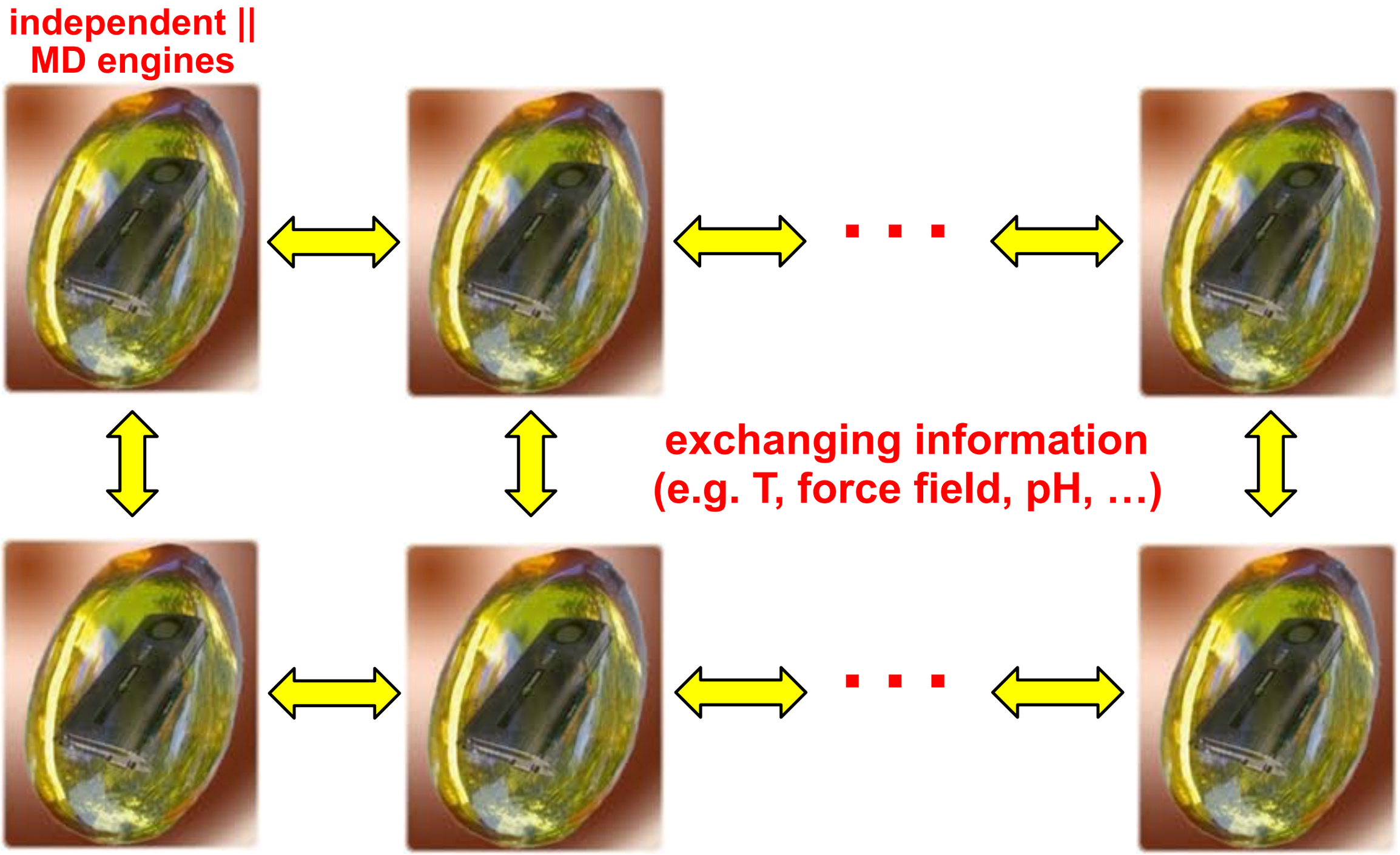


Test for convergence within and between simulations…



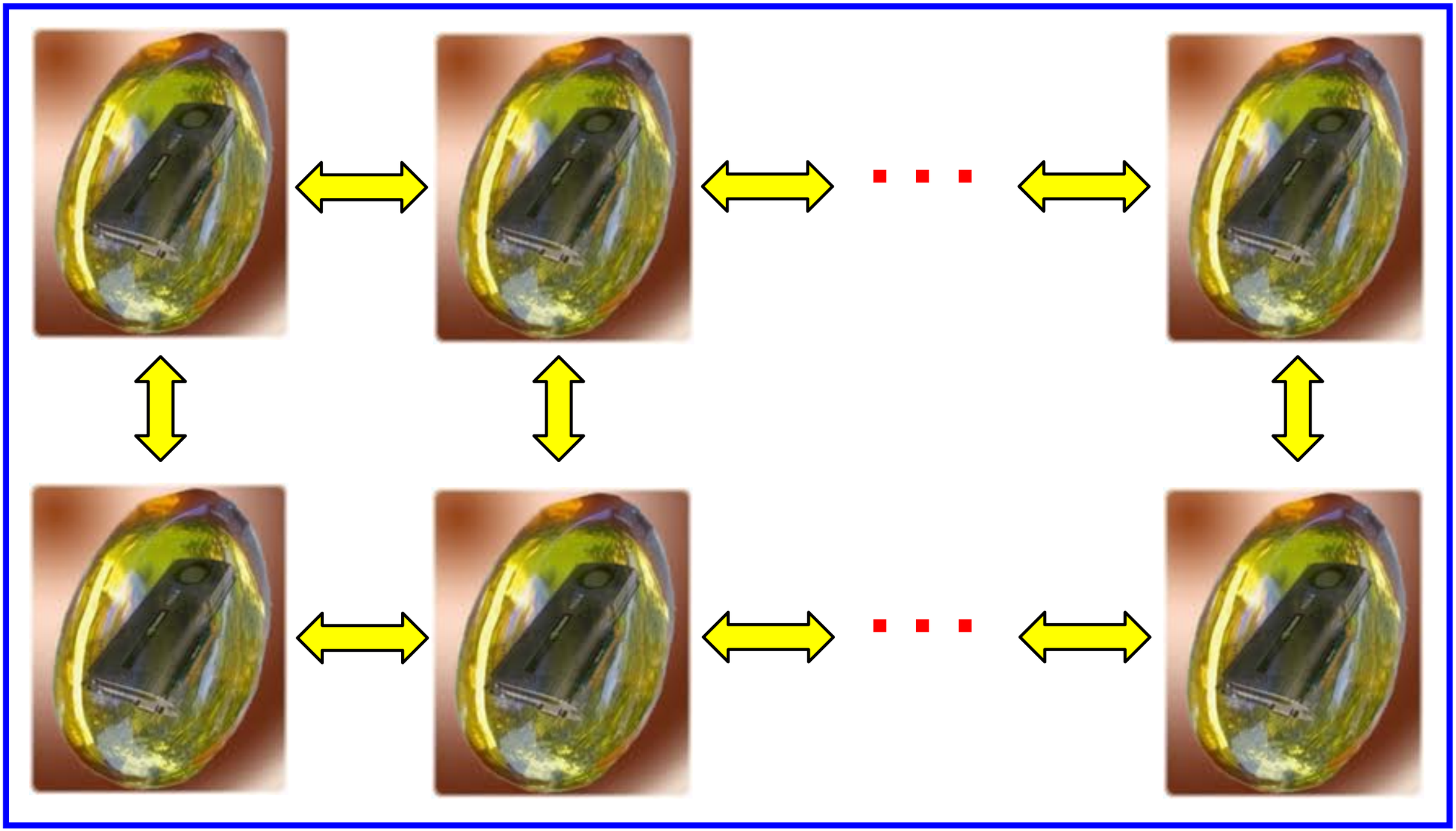


If we cannot scale to larger machine (more cores), couple independent MD simulations: i.e., use ensembles (replica exchange, || tempering, Markov State modeling, ...)

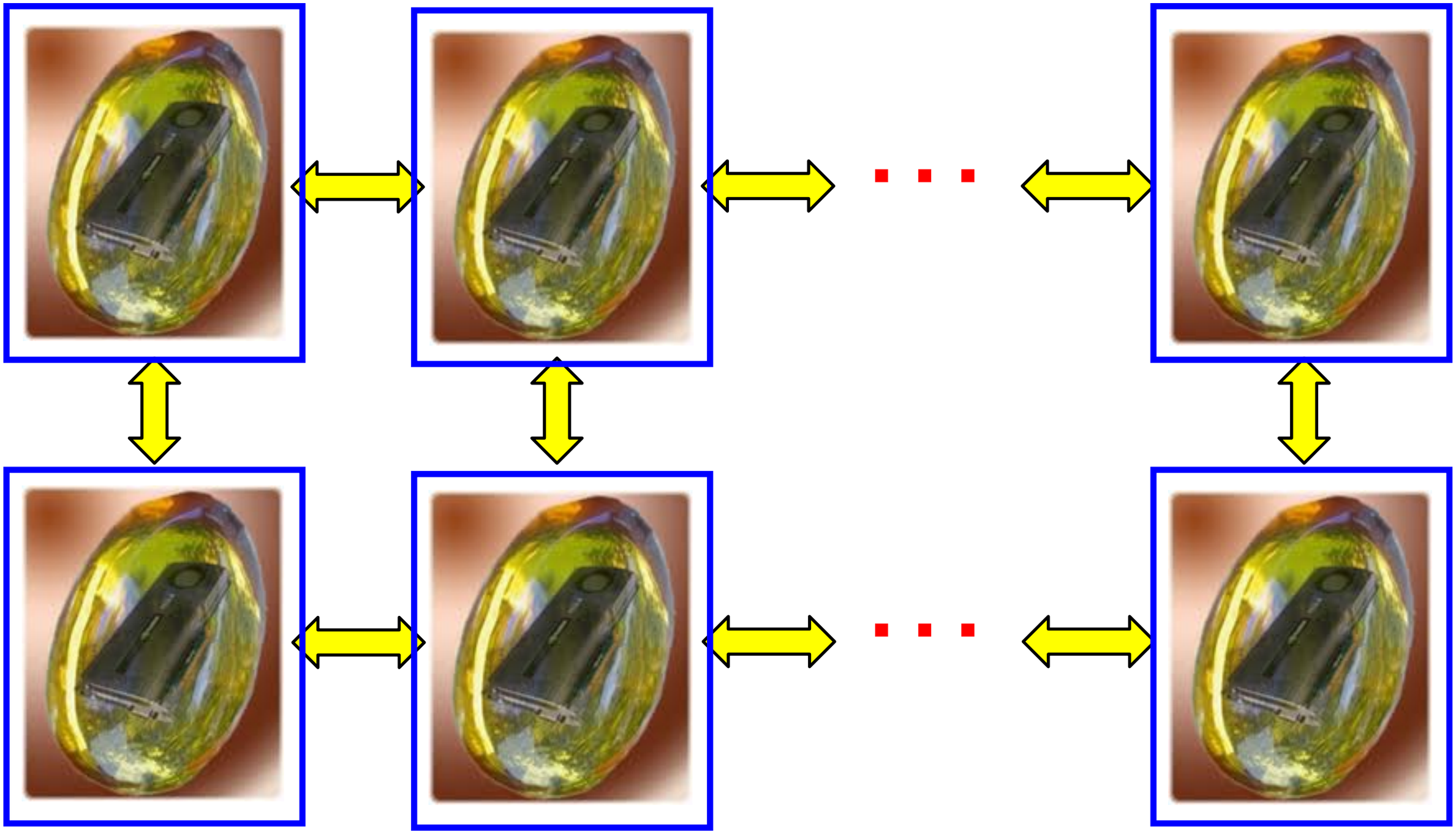


```
! o  All MPI communications should be done using the new
!    communicators rather than MPI_COMM_WORLD.  A number
!    of new communicators are defined:
!     CommSander  -- communications within a given sander job
!                    (replaces MPI_COMM_WORLD)
!     CommWorld   -- communications to ALL processors across
!                    multiple sander jobs
!     CommMaster  -- communications to the master node of each
!                    separate sander job each has corresponding
!                    size and rank,
!                    i.e. MasterRank, MasterSize

CommWorld = MPI_COMM_WORLD
call mpi_comm_rank( CommWorld, worldrank,  ierror )
call mpi_comm_size( CommWorld, worldsize,  ierror )
call mpi_barrier( CommWorld, ierror )
```

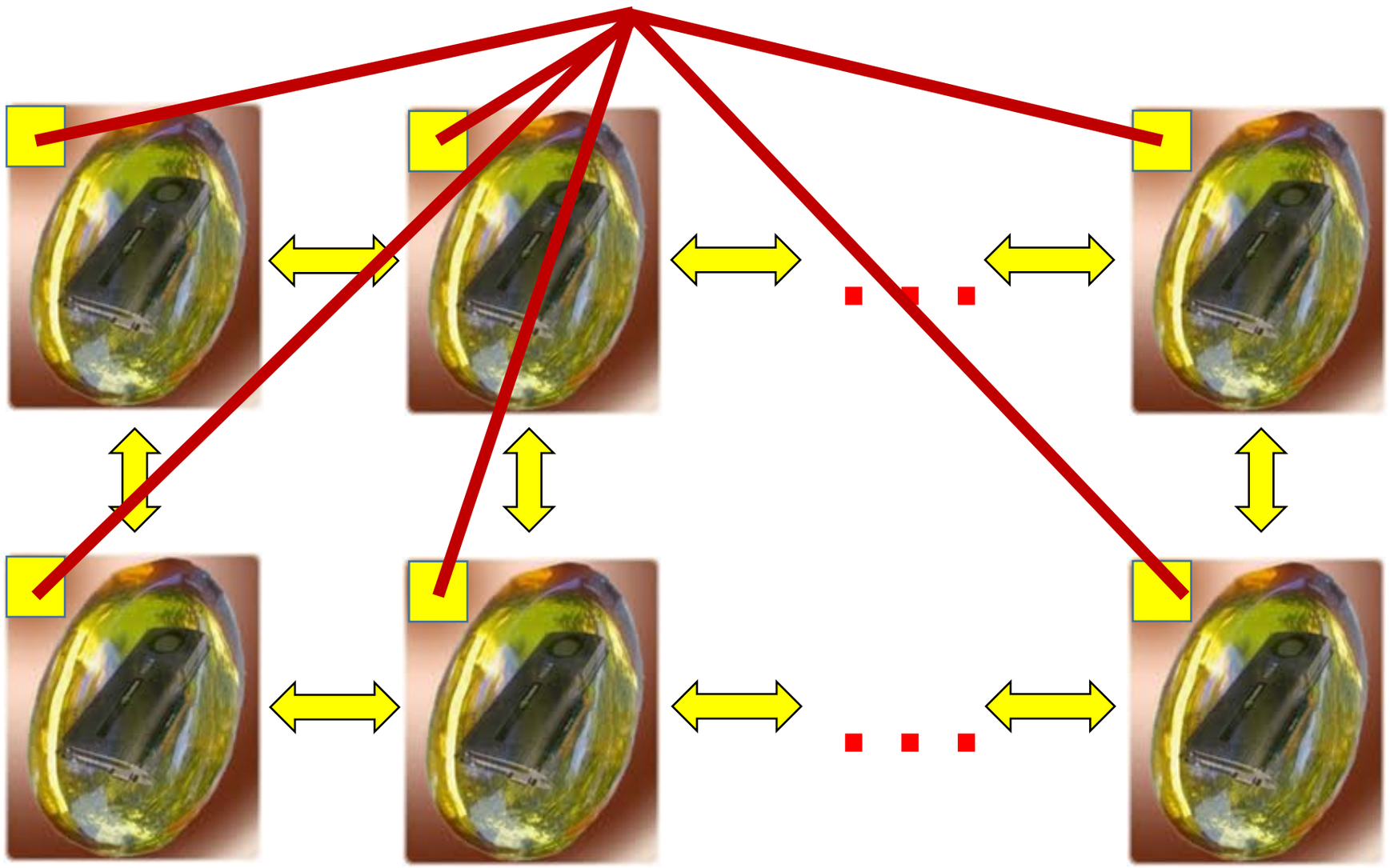
CommWorld



CommSander

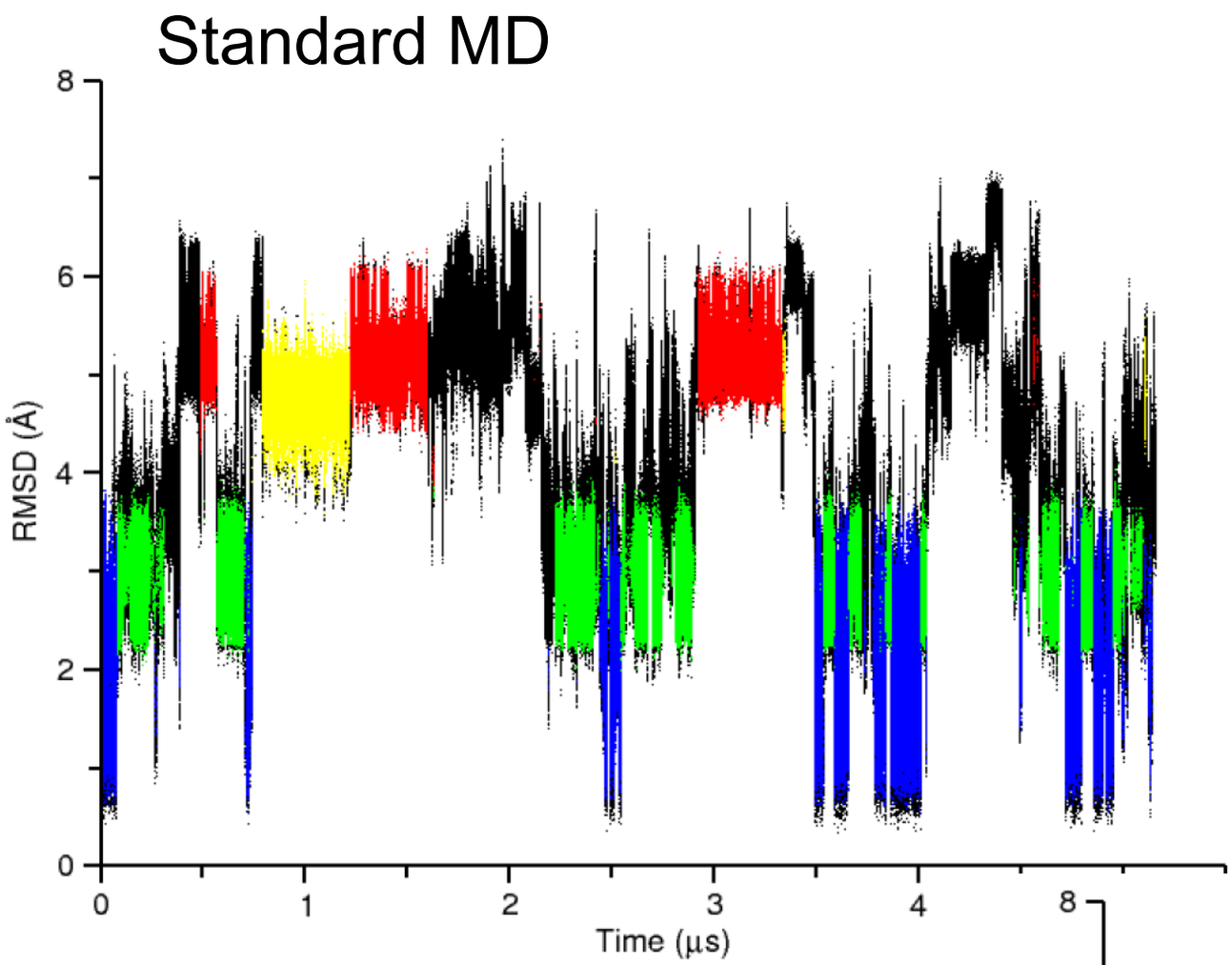


CommMaster



```
! Create a communicator for each group of -ng NumGroup processors

   commsander = mpi_comm_world
   sandersize = worldsize
   sanderrank = worldrank
   nodeid = mod(worldrank, numgroup)
   if (numgroup > 1) then
      commsander = mpi_comm_null
      call mpi_comm_split(commworld, nodeid, worldrank, &
            commsander, ierror)
      if (commsander == mpi_comm_null) then
         if (worldrank == 0) then
            write(6,'(a,i5,a,i5)') 'Error: NULL Communicator', &
                  ' on PE', worldrank, ' from group ', nodeid
         end if
         call mexit(6,1)
      end if
      call mpi_comm_size(commsander, sandersize, ierror)
      call mpi_comm_rank(commsander, sanderrank, ierror)
   end if
```

```
!  Define a communicator (CommMaster) that only talks between the local
!  "master" in each group.  This is equivalent to a SanderRank .eq. 0

  masterid = 0
  masterrank = MPI_UNDEFINED
  mastersize = 0
  if (numgroup > 1) then
     commmaster = mpi_comm_null
     if(sanderrank /= 0) then
        masterid = MPI_UNDEFINED
     end if

     call mpi_comm_split(commworld, masterid, worldrank, &
           commmaster, ierror)
     ! will this be emitted when using the default MPI error handler ?
     if (ierror /= MPI_SUCCESS) then
        write(6,*) 'Error: MPI_COMM_SPLIT error ', ierror, &
              ' on PE ', worldrank
     end if

     if(commmaster /= mpi_comm_null) then
        call mpi_comm_size(commmaster, mastersize, ierror)
        call mpi_comm_rank(commmaster, masterrank, ierror)
     end if
  end if
```

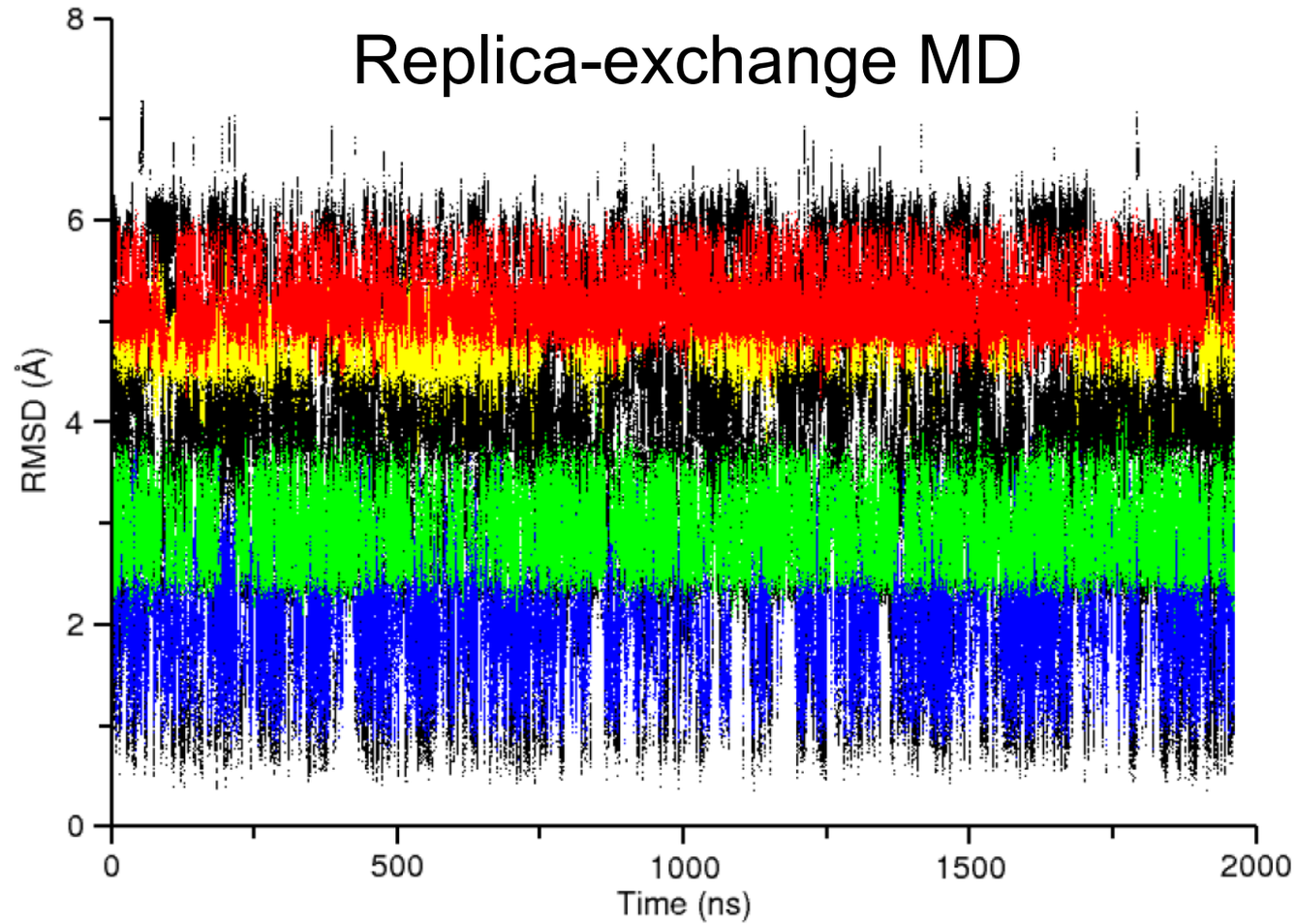


r(GACC) tetranucleotide

[Turner / Yildirim]

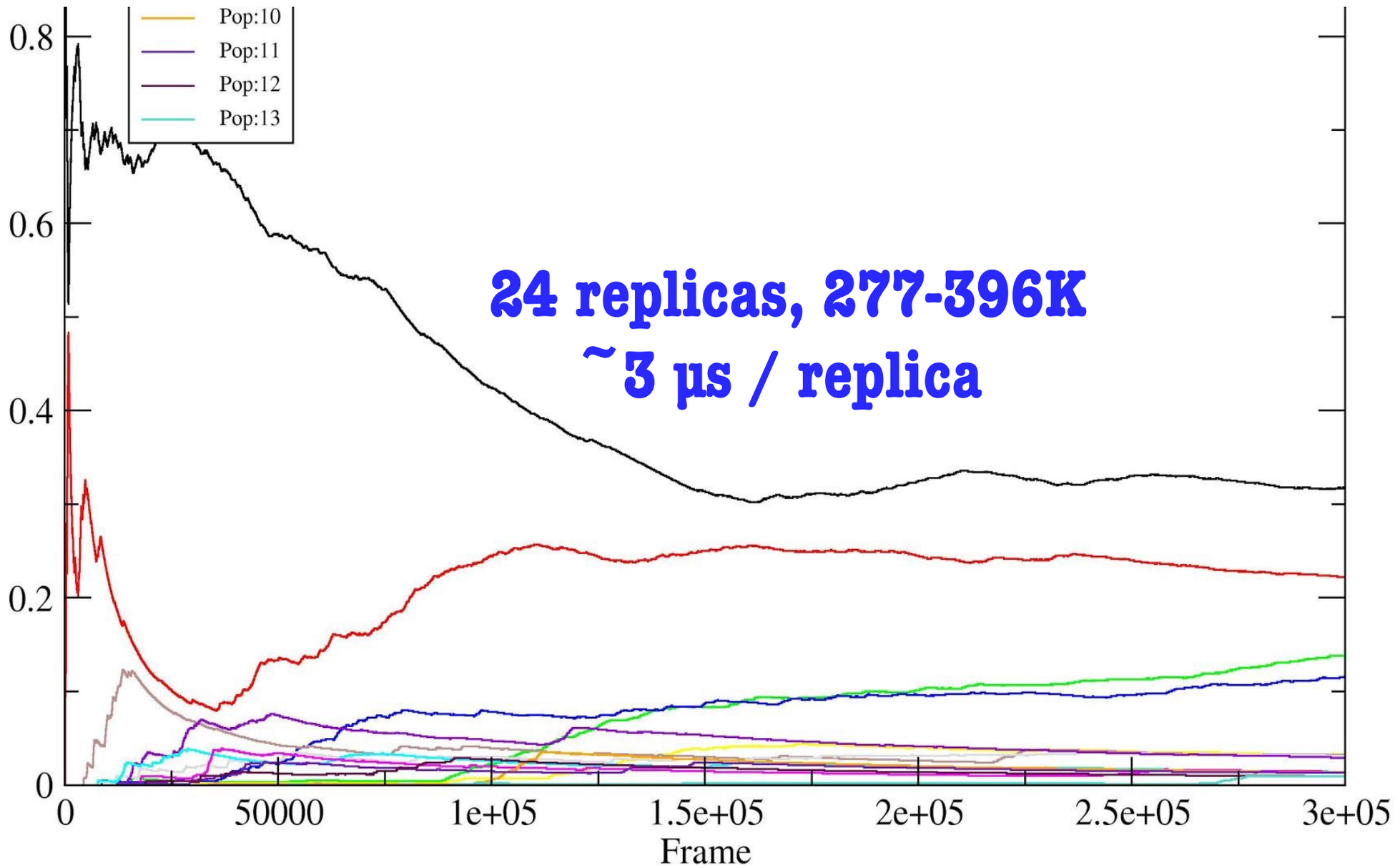
< explicit solvent >

…a system where we can get complete sampling

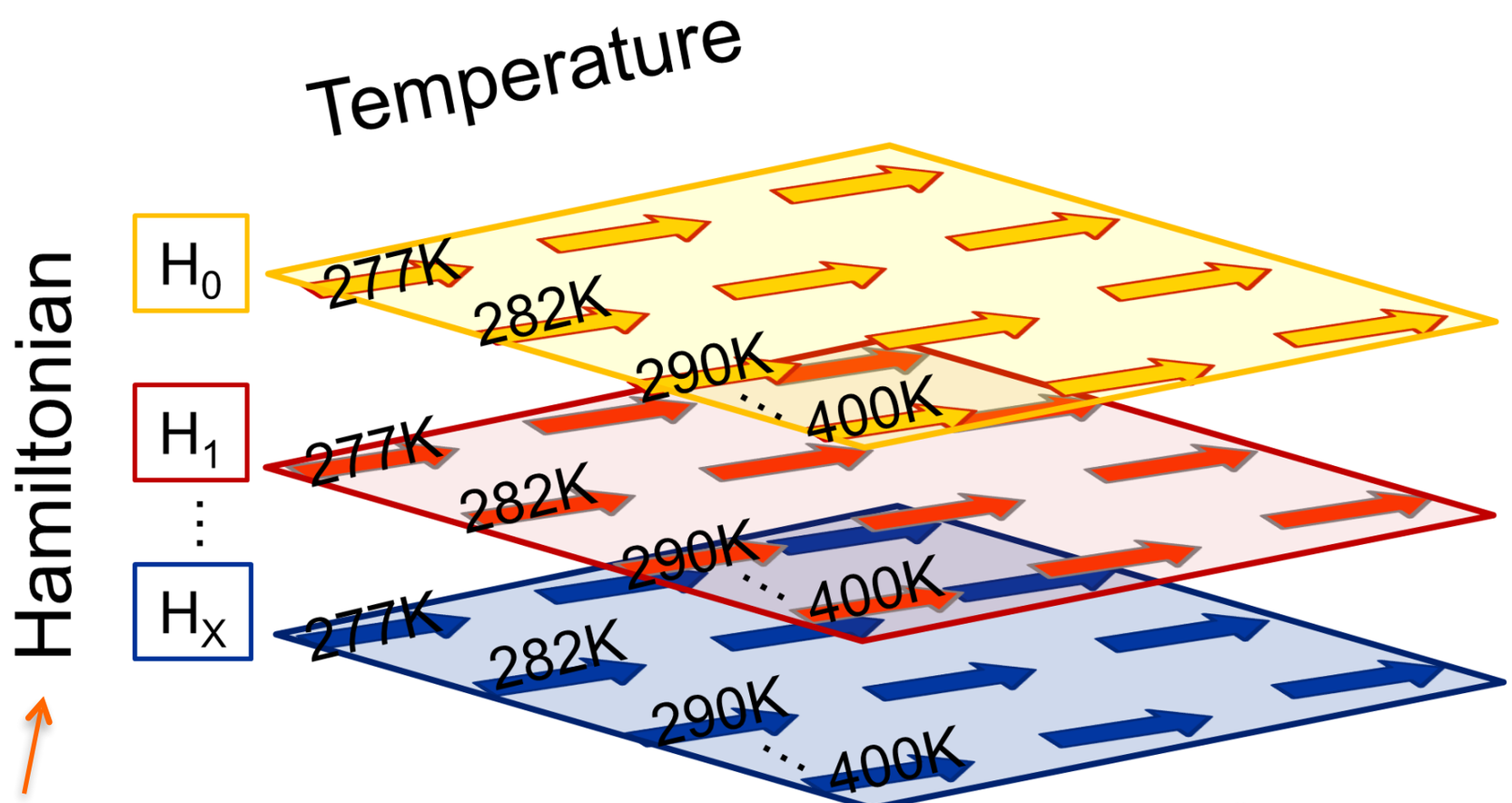


Other issues:

- **T-REMD still not "fully" converged (depending on def.)**



multi-D REMD

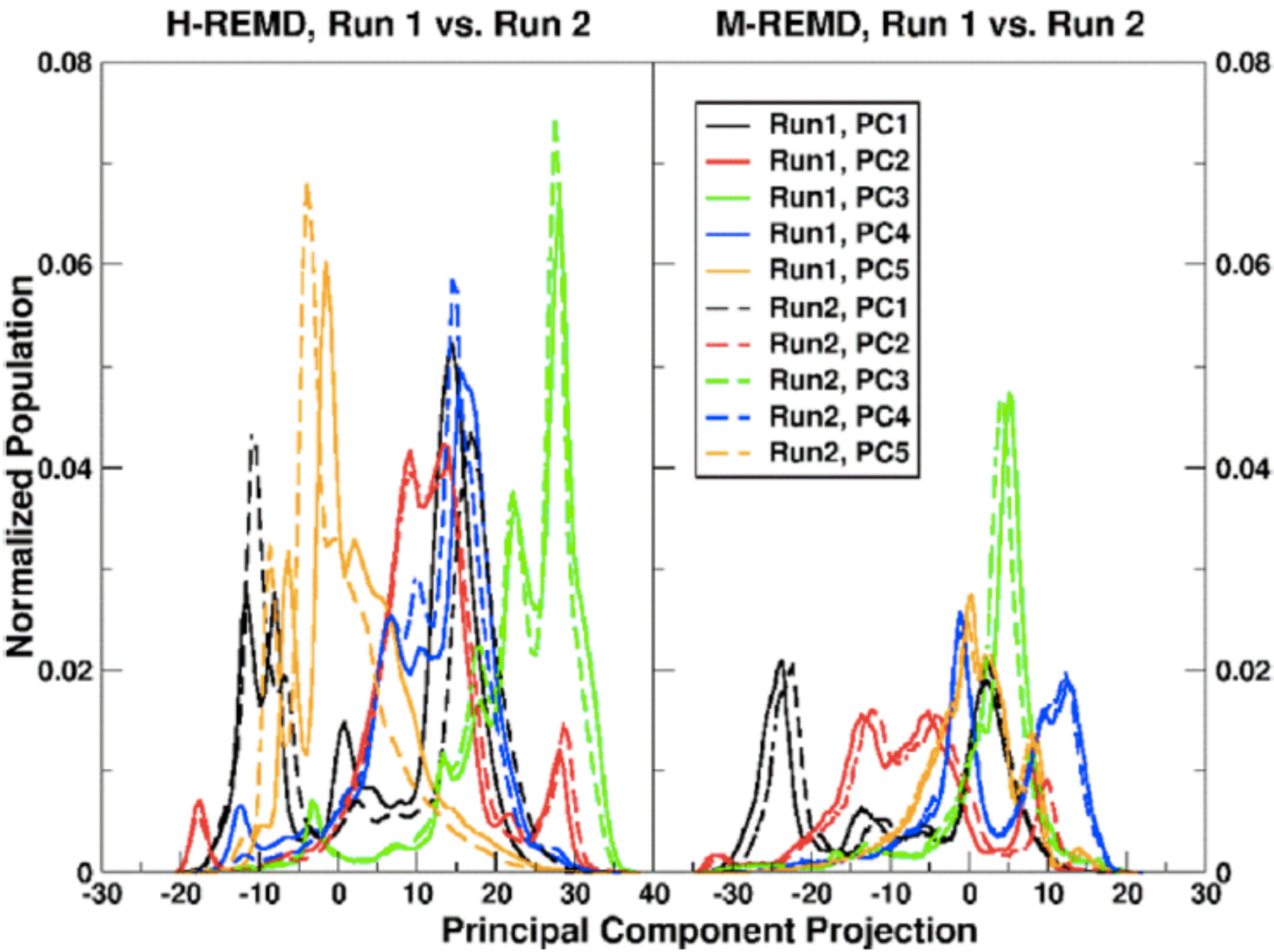


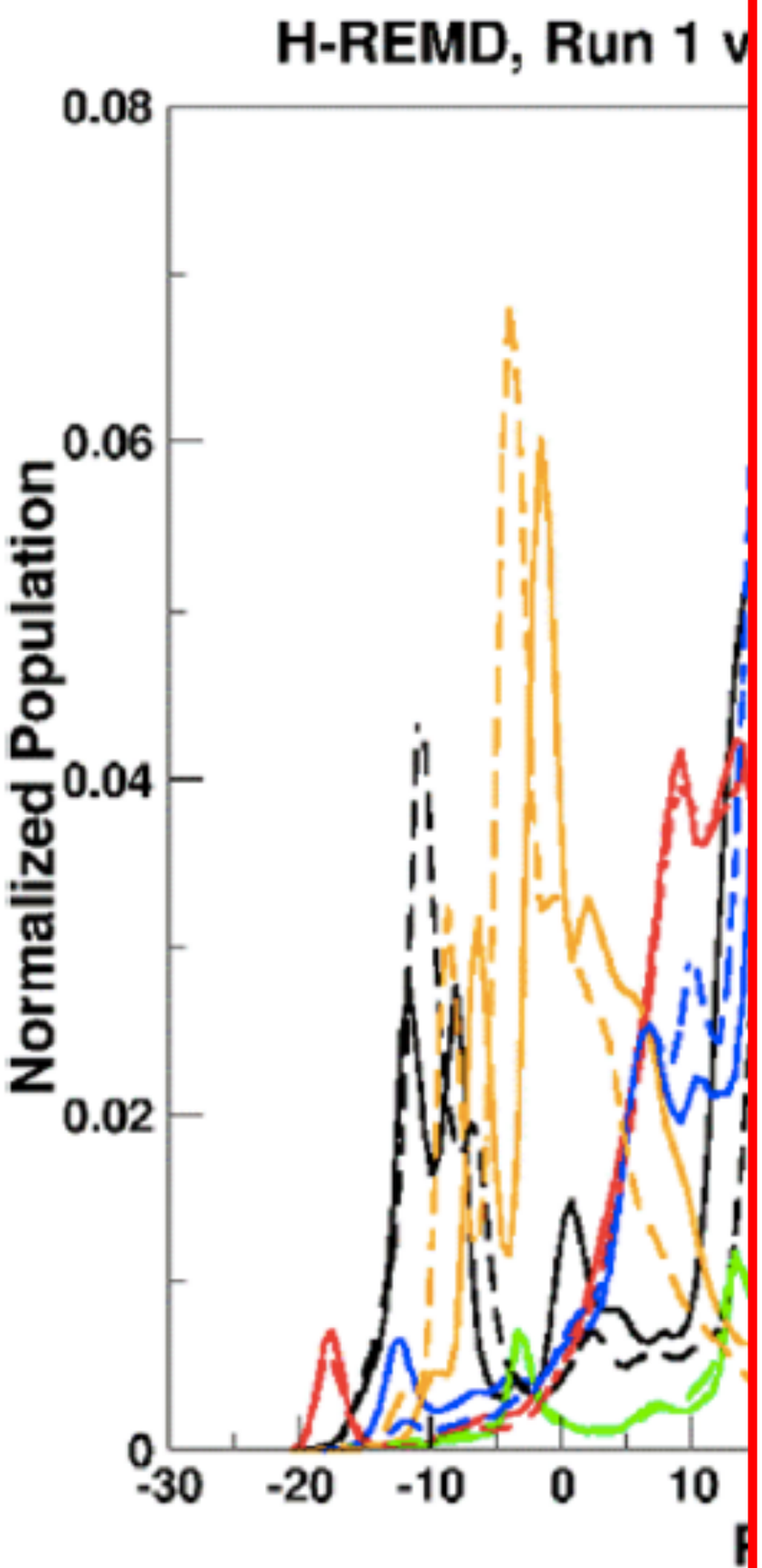
Change in "energy representation"

- **pH**
- **restraints, umbrella potentials, …**
- **force field / parameter sets**
- **biasing potentials (aMD)**

24 Temperatures
x 8 Hamiltonians
=192 replicas

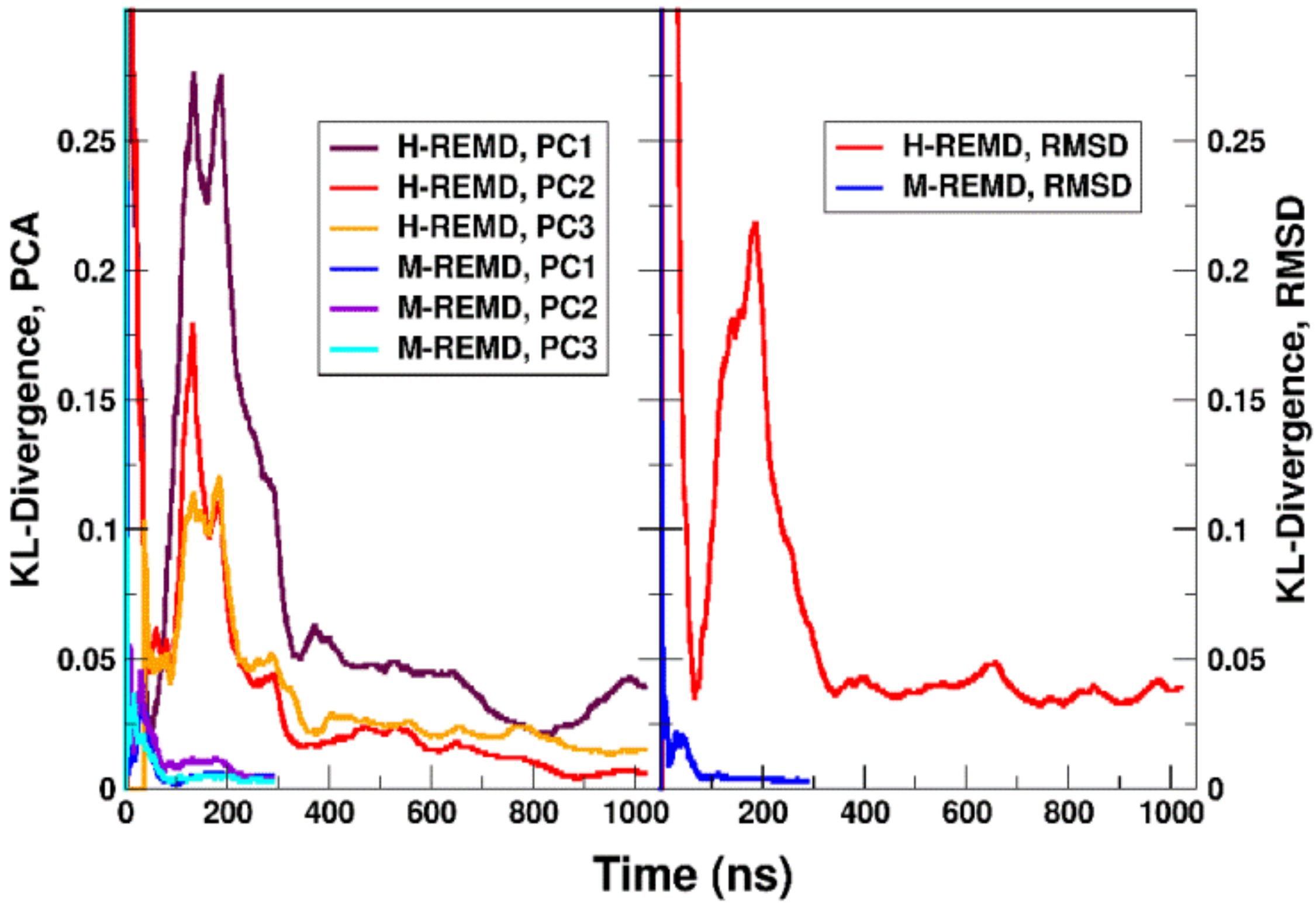
Fukunishi, H., Wanatabe, O., and Takada, S., J. Chem. Phys. 2002.
Sugita, Y., Kitao, A., and Y. Okamoto, J. Chem. Phys. 2000.


H-REMD, Run 1 vs. Run 2
M-REMD, Run 1 vs. Run 2
Normalized Population
Principal Component Projection
Run1, PC1
Run1, PC2
Run1, PC3
Run1, PC4
Run1, PC5
Run2, PC1
Run2, PC2
Run2, PC3
Run2, PC4
Run2, PC5
0.08
0.06
0.04
0.02
0
-30
-20
-10
0
10
20
30
40



CPPTRAJ in AmberTools

```
# Read in both trajectories
#
trajin traj.run1.nc
trajin traj.run2.nc
# RMS-fit to first frame
#
rms first :1-4&!@H=
# Create an average structure
#
average gaccAvg.rst7 ncrestart
# Save coordinates as 'crd1'
#
createcrd crd1
run
# Fit to average structure
#
reference gaccAvg.rst7.1 [avg]
# RMS-fit to average structure
#
crdaction crd1 rms ref [avg] :1-4&!@H=
# Calculate coordinate covariance matrix
#
crdaction crd1 matrix covar :1-4&!@H= name gaccCovar
# Diagonalize coordinate covariance matrix, first 15 E.vecs
#
runanalysis diagmatrix gaccCovar out evecs.dat vecs 15
# Now create separate projections for each trajectory
#
crdaction crd1 projection P1 modes evecs.dat \
    beg 1 end 15 :1-4&!@H= crdframes 1,$STOP1
crdaction crd1 projection P2 modes evecs.dat \
    beg 1 end 15 :1-4&!@H= crdframes $START2,last
# Now histogram first 5 projections for each
#
hist P1:1,*,*,*,100 out pca.hist.agr norm name P1-1
hist P1:2,*,*,*,100 out pca.hist.agr norm name P1-2
```


H-REMD, PC1
H-REMD, PC2
H-REMD, PC3
M-REMD, PC1
M-REMD, PC2
M-REMD, PC3
H-REMD, RMSD
M-REMD, RMSD
KL-Divergence, PCA
KL-Divergence, RMSD
Time (ns)
0
0.05
0.1
0.15
0.2
0.25
200
400
600
800
1000

Problems with our tightly coupled approach (and/or ensembles in general)

- How to analyze? Sort by replica? Temperature? H? (need parallel else kills file system or sequential)
- If one ensemble instance slows, they all slow…
- Start-up time is non-trivial as number of ensemble instances grows…

```
aprun –n 4 myprogram.exe &
aprun –n 4 myprogram.exe &
aprun –n 4 myprogram.exe &
…
wait
```

versus

```
aprun –n 40 myprogram.exe –np 40 &
```

Problems with our tightly coupled approach (and/or ensembles in general)

- How to analyze? Sort by replica? Temperature? H? (need parallel else kills file system or sequential)
- If one ensemble instance slows, they all slow…
- Start-up time is non-trivial as number of ensemble instances grows…

```
aprun –n 4 myprogram.exe &
aprun –n 4 myprogram.exe &
aprun –n 4 myprogram.exe &
…
wait
```

versus

```
aprun –n 40 myprogram.exe –np 40 &
```

Needs:
asynchronous
fault tolerance
heterogeneous
easy spec / DSL / API

Can converge r(GAAC) in 1 day, a tetraloop in ~1-2 weeks!!

Production MD is no longer rate limiting step in workflow!

Setup, analysis, data management, …

Needs:

- ensemble management tools
- workflow tools
- data management solutions
- means to compare and share research results and codes

use tiered resources to facilitate data analysis

Run jobs

Each resource has special strengths

Process data (deconvolute, cluster)

flash: moderate size, lifetime ~days, less trivial analysis (fast timescales)

QM calcs

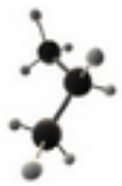
iBIOMES

Integrated Biomolecular Simulations

Home | Search | Documentation | About | Cart

jthibault | logout

[experiment] Small RNA - 1BIV (AMBER)



ibiomesZone > home > jthibault > 1BIV

Molecular dynamics of small RNA (1BIV)

Software package	AMBER
Method	Molecular dynamics
Molecule type	Protein / RNA

Author	jthibault
Uploaded on	8/2/12

Structure and methods | Files | References

Molecular structure and simulation method

STRUCTURAL INFORMATION

Residue chain: RG5 RG RC RU RC RG RU RG RU RA RG RC RU RC RA RU RU RA RG RC RU RC RC RG RA RG RC RC3 SER GLY PRO ARG PRO ARG GLY THR ARG GLY LYS GLY ARG ARG ILE ARG ARG

Normalized chains:

- Protein: SER GLY PRO ARG PRO ARG GLY THR ARG GLY LYS GLY ARG ARG ILE ARG ARG
- RNA: GGCUCGUGUAGCUCAUUAGCUCCGAGCC

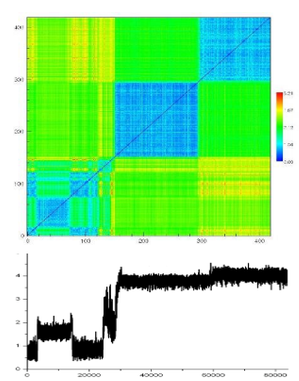
Number of atoms	20563
Number of water molecules	6449
Number of ions	31 [Cl-, Na+]

MOLECULAR DYNAMICS

Force-field(s)	AMBER 99
Solvent	In vacuo
Constraints	SHAKE
Electrostatics interactions	PME
Boundary conditions	Periodic

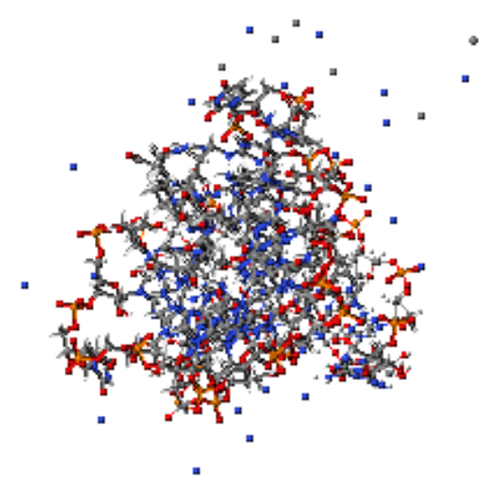
2D RMSd

RMSd vs. time



3D structure

Averaged structure based on clustering



Jmol_S

automate analysis
&
tools for deeper
“interactive”
analysis

Molecule structure and simulation parameters

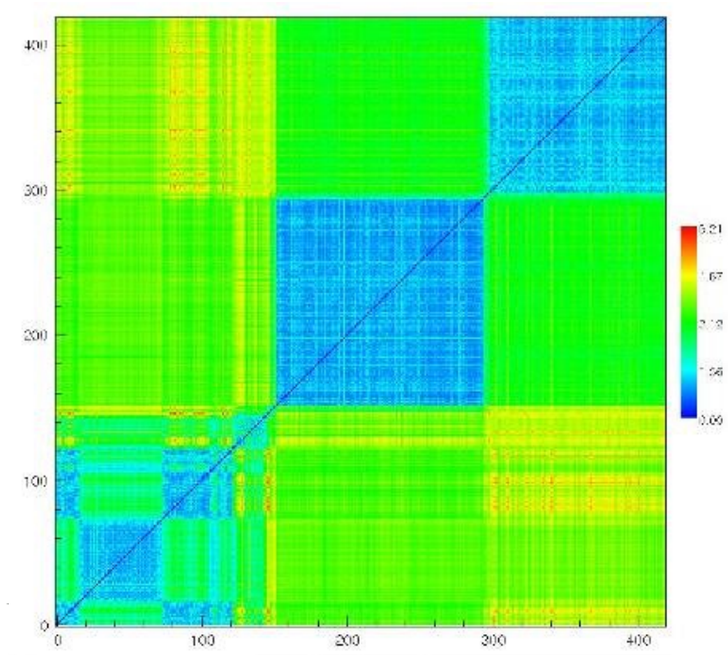
Residue chain: RG5 RC RC RC RG RG RA RU RA RG RC RU RC RA RG RU RC RG RG RU RA RG RA RG RC RA PSU RC RA RG RA RC RU U8U RU RU T6A RA PSU RC RU RG RA RG RG RG RU RC RC RA RG RG RG RU PSU RC RA RA RG RU RC RC RC RU RG RU RU RC RG RG RG RC RG RC RC RA3 RG5 RU RG RU RG RA RC RU RC RU RG RG RU RA RA RC RU RA RG RA RG RA RU RC RC RC RU RC RA RG RA RC RC RA RC RU RC RU RA RG RA RC RG RG RU RG RU RA RA RA RA RA RU RC RU RC RU RA RG RC RA RG RU RG RG RC RG RC RC RC RG RA RA RC RA RG RG RG RA RC RU RU RU RA RA RA RG RU RG RA RA RA RG RU RA RA RC RA RG RG RG RA3

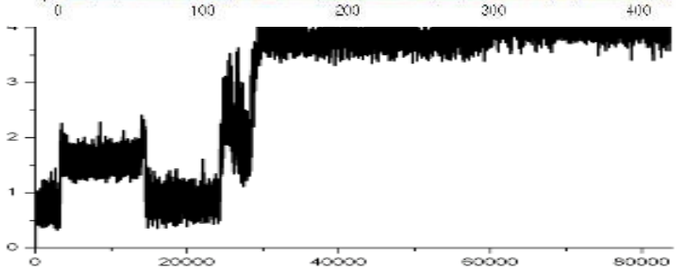
Non-standard residues:

- PSU: P O1P O2P O5' C5' H5'1 H5'2 C4' H4' O4' C1' H1' N1 C6 H6 C5 C4 O4 N3 H3 C2 O2 C3' H3' C2' H2'1 O2' HO'2 O3' H1
- T6A: P O1P O2P O5' C5' H5'1 H5'2 C4' H4' O4' C1' H1' N9 C8 H8 N7 C5 C6 N6 H6 N1 C2 H2 N3 C4 C3' H3' C2' H2'1 O2' HO'2 O3' C10 N11 O10 C12 H11 C13 O13A O13B C14 C15 O14 H14 HO4 H12 H151 H152 H153
- U8U: P O1P O2P O5' C5' H5'1 H5'2 C4' H4' O4' C1' H1' N1 C6 H6 C5 C C4 O4 N3 H3 C2 S2 C3' H3' C2' H2'1 O2' HO'2 O3' N CA HC1 HC2 HA1 HA2 HA3 HN HN2

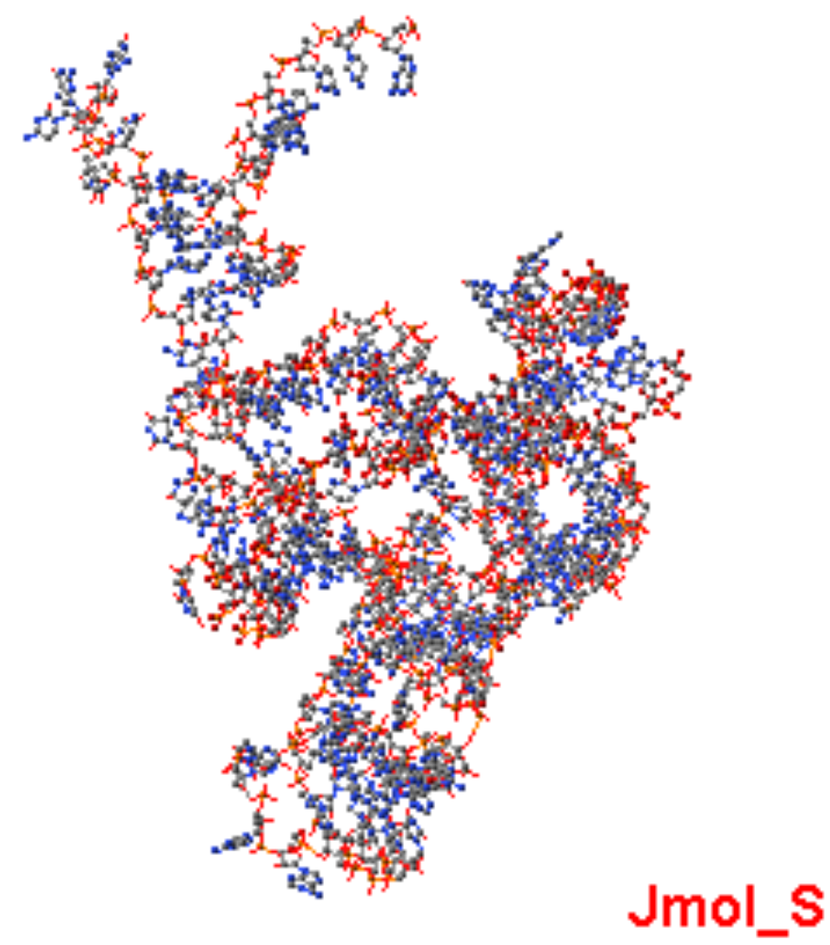
Number of atoms: 5746
Number of water molecules: 0
Number of ions: 0

Force-field: AMBER94

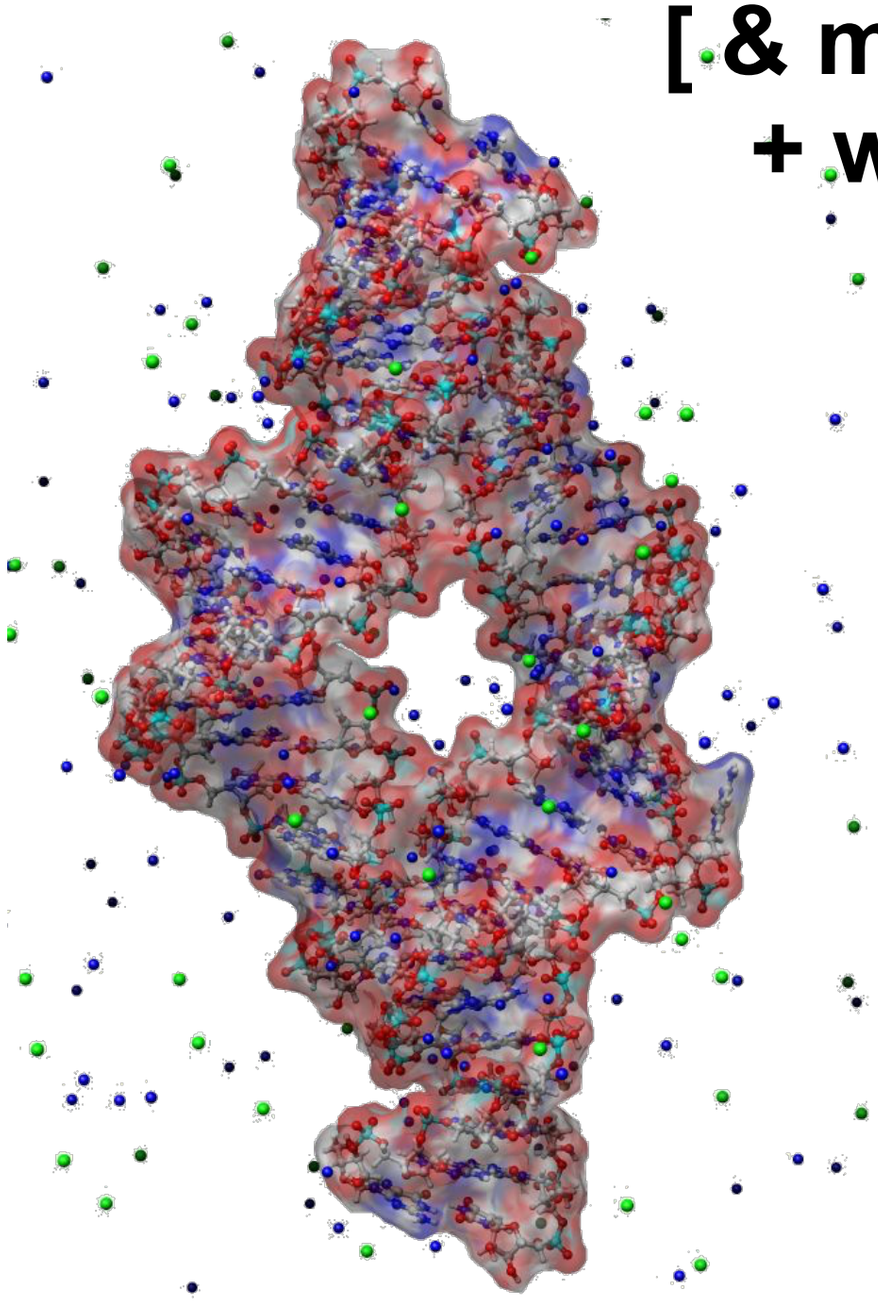




Sample structure (min_test.pdb)



Peta- or exa- scale science: the problem will only get worse!



Solutions? Analysis "on the fly…"
[& more coarse-grained sampling]
+ workflow tools for ensembles

- Do not move the data (?)
- Tiered resources
- Persistent storage
- Re-running the simulations

…what will we miss? Can we only get low hanging fruit?

Data challenges:

- no longer feasible to save all data (on local resources)
- insufficient local resources (back-up, HSM)
- data risk: can reboot compute, not disk…
- unclear cost models
- domain specific

Solutions?

- save / distribute only what you need
  - reduced data vs. raw vs. input decks
  - host data on national servers, remote analysis

Data challenges:

- no longer feasible to save all data (on local resources)
- insufficient local resources (back-up, HSM)
- data risk: can reboot compute, not disk…
- unclear cost models
- domain specific

Solutions?

- save / distribute only what you need
  - reduced data vs. raw vs. input decks
  - host data on national servers, remote analysis

Barriers to sharing data

- culture: “hidden gold” vs. exposing flaws
- cost models: how to pay?

Benefits:

- benchmarking results, assessment / validation

People: Hamed Hayatshahi, Dan Roe, Rodrigo Galindo, Christina Bergonzo, Sean Cornillie, James Robertson, Zahra Heidari [+ Henriksen, Thibault, Shahrokh]

$$$:

National Science Foundation
WHERE DISCOVERIES BEGIN

NIH R01-GM098102 "RNA-ligand interactions: simulation & experiment"
NSF CHE-1266307 "CDS&E: Tools to facilitate deeper data analysis, …"
NSF ACI-1521728 "RAPID: Optimizing … Ebola membrane fusion inhibitor … design"
NSF ACI-1443054 "CIF21 DIBBS: Middleware and high performance analytics…"
NSF ACI-1341034 "CC-NIE Integration: Science slices…" network DMZ
NSF "Blue Waters" PetaScale Resource Allocation for AMBER RNA

Computer time:

PITTSBURGH SUPERCOMPUTING CENTER

D E Shaw Research

"Anton"
(3 past awards)

XSEDE
Extreme Science and Engineering Discovery Environment

XRAC MCA01S027
~12M core hours

BLUE WATERS
SUSTAINED PETASCALE COMPUTING

~7-14M GPU hours

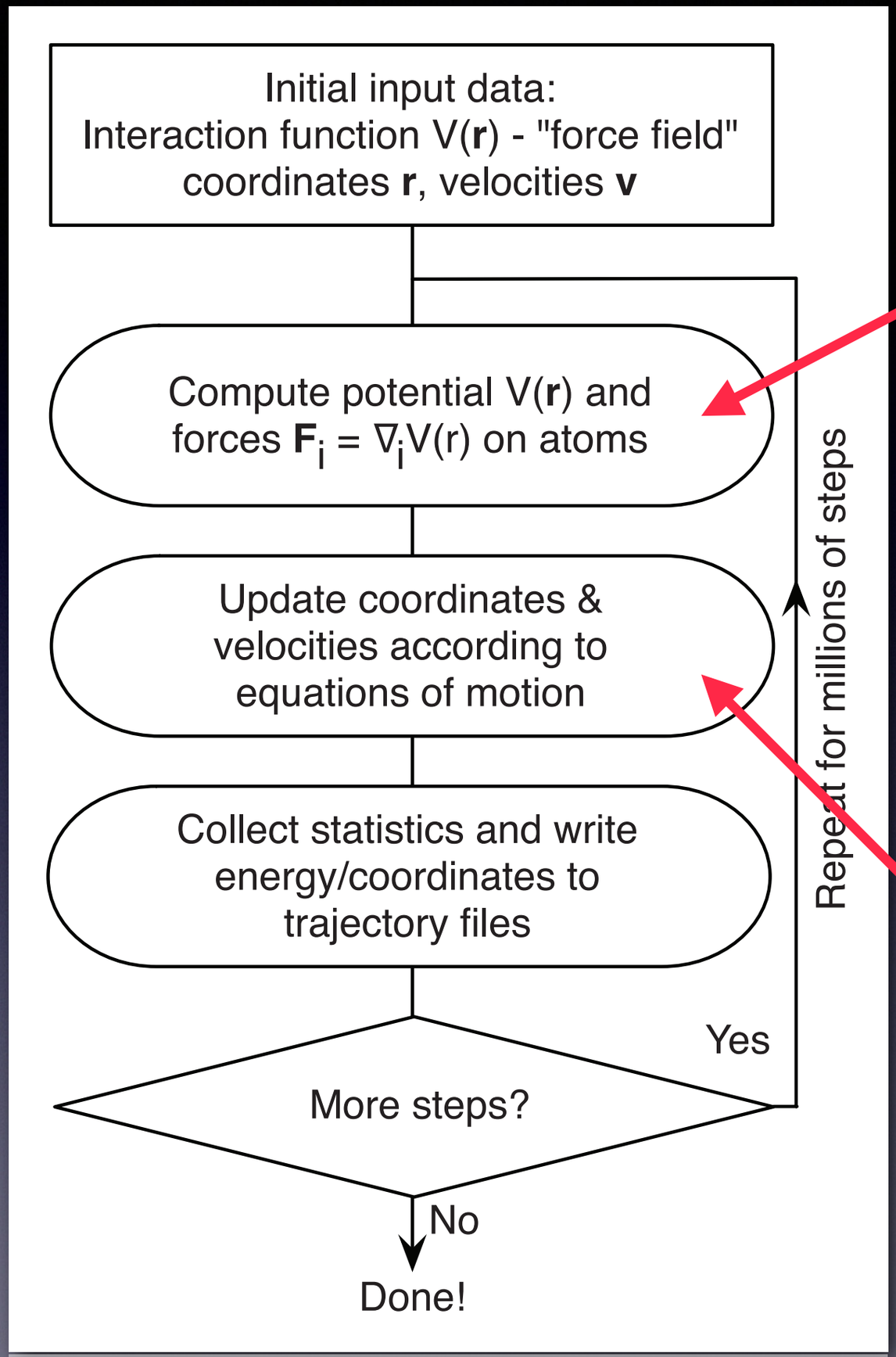
CHPC

~3M hours

!!!

LIFE SCIENCE: HOW DO WE SOLVE PROBLEMS FASTER IN PARALLEL AND ON ACCELERATORS?

Thomas E Cheatham III, University of Utah, Salt Lake City, UT
Erik Lindahl, Stockholm University, Stockholm
tec3@utah.edu
erik.lindahl@scilifelab.se

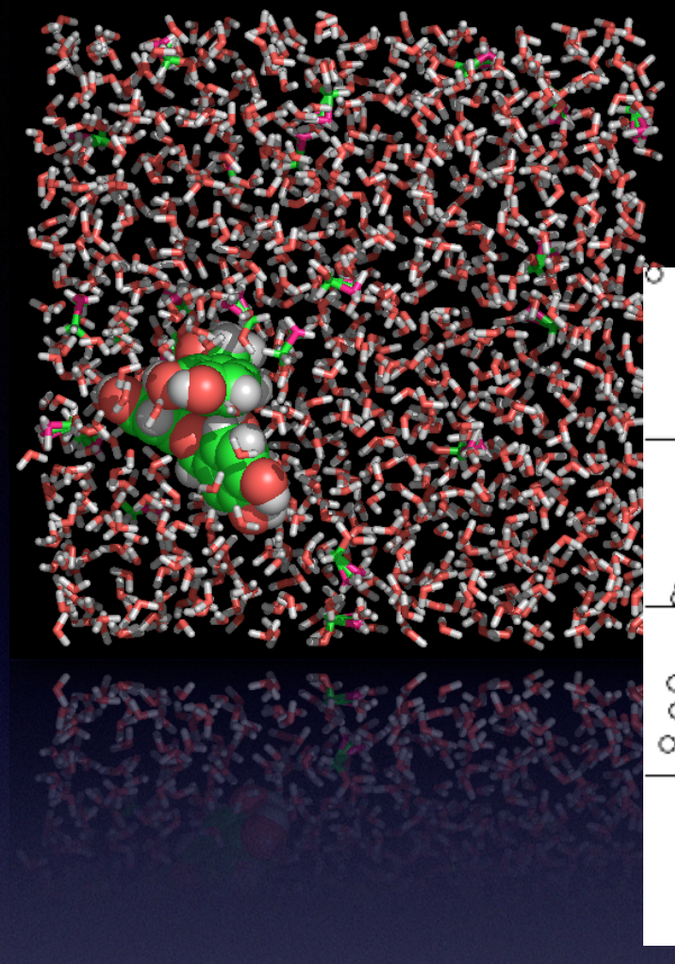


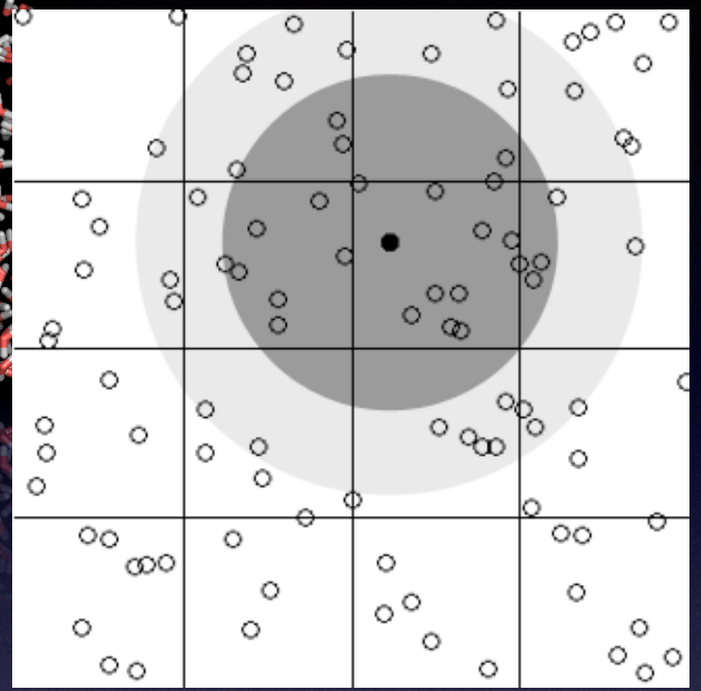
$$V(r) = \sum_{bonds} \frac{1}{2} k_{ij}^{b} \left(r_{ij} - r_{ij}^{0}\right)^2$$
$$+ \sum_{angles} \frac{1}{2} k_{ijk}^{\theta} \left(\theta_{ijk} - \theta_{ijk}^{0}\right)^2$$
$$+ \sum_{torsions} \left\{ \sum_{n} k_{\theta} \left[1 + \cos\left(n\phi - \phi_0\right)\right] \right\}$$
$$+ \sum_{impropers} k_{\xi} \left(\xi_{ijkl} - \xi_{ijkl}^{0}\right)$$
$$+ \sum_{i,j} \frac{q_i q_j}{4\pi\varepsilon_0 r_{ij}}$$
$$+ \sum_{i,j} \left[\frac{C_{12}}{r_{ij}^{12}} - \frac{C_6}{r_{ij}^{6}}\right]$$

Costly, because these terms involve all pairs

$$m_i \frac{\partial^2 r_i}{\partial t^2} = F_i \qquad i = 1..N$$

$$F_i = -\frac{\partial V(r)}{\partial r_i}$$





I interaction

```
for(k=nj0; (k<nj1); k++)
{

    /* Get j neighbor index, and coordinate index */
    jnr             = jjnr[k];
    j3              = 3*jnr;

    /* load j atom coordinates */
    jx1             = pos[j3+0];
    jy1             = pos[j3+1];
    jz1             = pos[j3+2];

    /* Calculate distance */
    dx11            = ix1 - jx1;
    dy11            = iy1 - jy1;
    dz11            = iz1 - jz1;
    rsq11           = dx11*dx11+dy11*dy11+dz11*dz11;

    /* Calculate 1/r and 1/r2 */
    rinv11          = 1.0/sqrt(rsq11);

    /* Load parameters for j atom */
    qq              = iq*charge[jnr];
    tj              = nti+2*type[jnr];
    c6              = vdwparam[tj];
    c12             = vdwparam[tj+1];
    rinvsq          = rinv11*rinv11;

    /* Coulomb interaction */
    vcoul           = qq*rinv11;
    vctot           = vctot+vcoul;

    /* Lennard-Jones interaction */
    rinvsix         = rinvsq*rinvsq*rinvsq;
    Vvdw6           = c6*rinvsix;
    Vvdw12          = c12*rinvsix*rinvsix;
    Vvdwtot         = Vvdwtot+Vvdw12-Vvdw6;
    fscal           = (vcoul+12.0*Vvdw12-6.0*Vvdw6)*rinvsq;

    /* Calculate temporary vectorial force */
    tx              = fscal*dx11;
    ty              = fscal*dy11;
    tz              = fscal*dz11;

    /* Increment i atom force */
    fix1            = fix1 + tx;
    fiy1            = fiy1 + ty;
    fiz1            = fiz1 + tz;

    /* Decrement j atom force */
    faction[j3+0]   = faction[j3+0] - tx;
    faction[j3+1]   = faction[j3+1] - ty;
    faction[j3+2]   = faction[j3+2] - tz;

    /* Inner loop uses 38 flops/iteration */
}
```

The challenge:

- ~100,000 atoms
- Each has ~500 neighbors
- ~50M interactions/step
- ~2B FLOPS per step
- ~1ms real time per step

Historical approaches to make our codes faster:

1970-1990: Reduce floating-point operations
(the fastest FLOP is the one we don't calculate)

1990-2000: Try to parallelize the existing algorithms
where we removed FLOPS

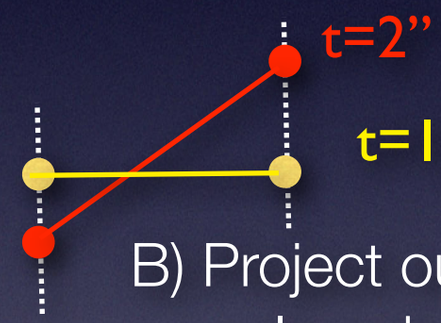
2000-: Change algorithms to extract parallelism

Example: Remove FLOPS by taking longer steps

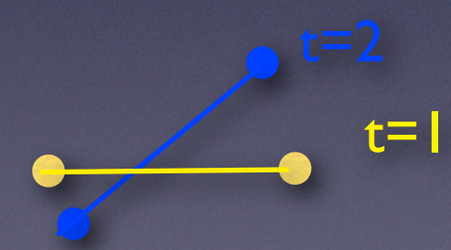
- Δt limited by fast motions - 1fs
  - Remove bond vibrations
- SHAKE - 2fs
  - Problematic in parallel (won't work)
  - Compromise: constrain h-bonds only - 1.4fs
- LINCS:
  - LINear Constraint Solver
  - Approximate matrix inversion expansion
  - Fast & stable - much better than SHAKE
  - Non-iterative
  - Enables 2-3 fs timesteps
  - Parallel: P-LINCS (from Gromacs 4.0)

LINCS:

A) Move w/o constraint



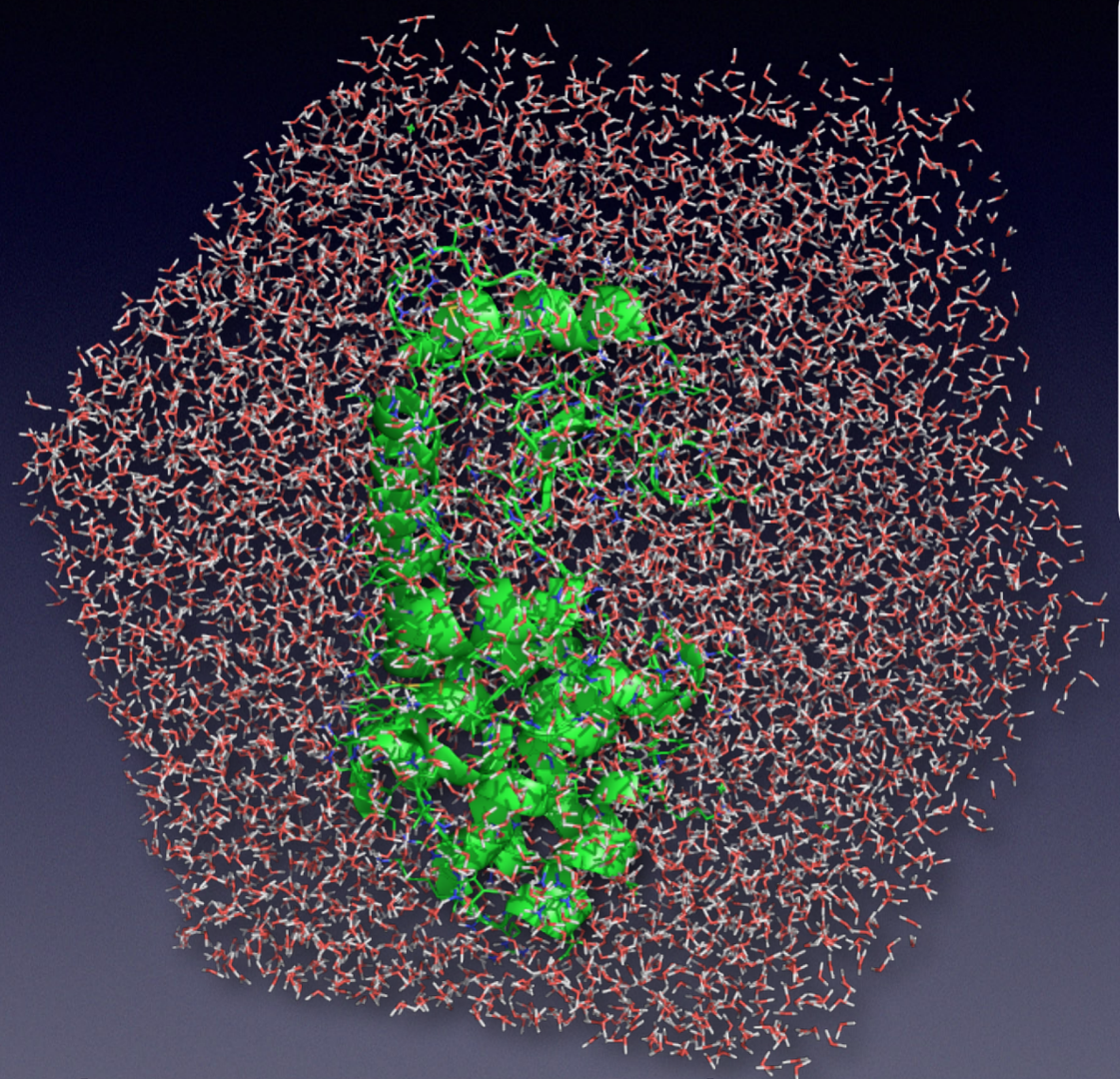
B) Project out motion along bonds

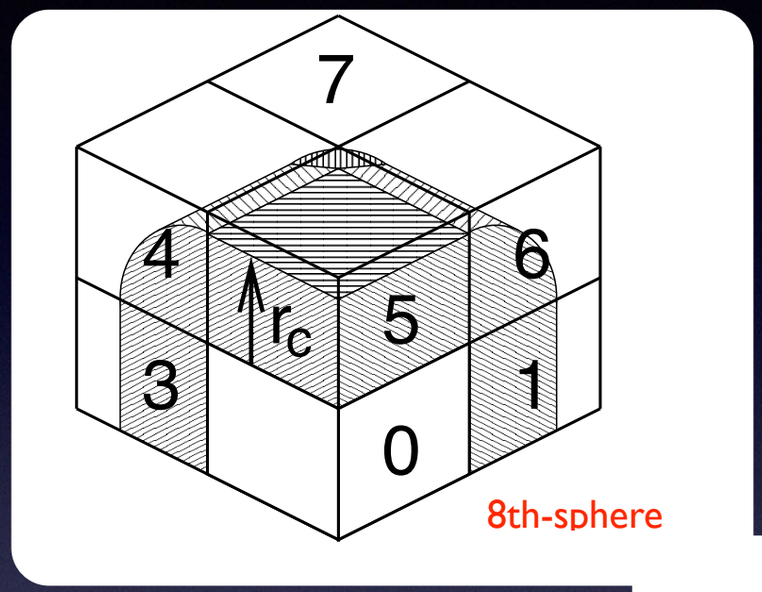


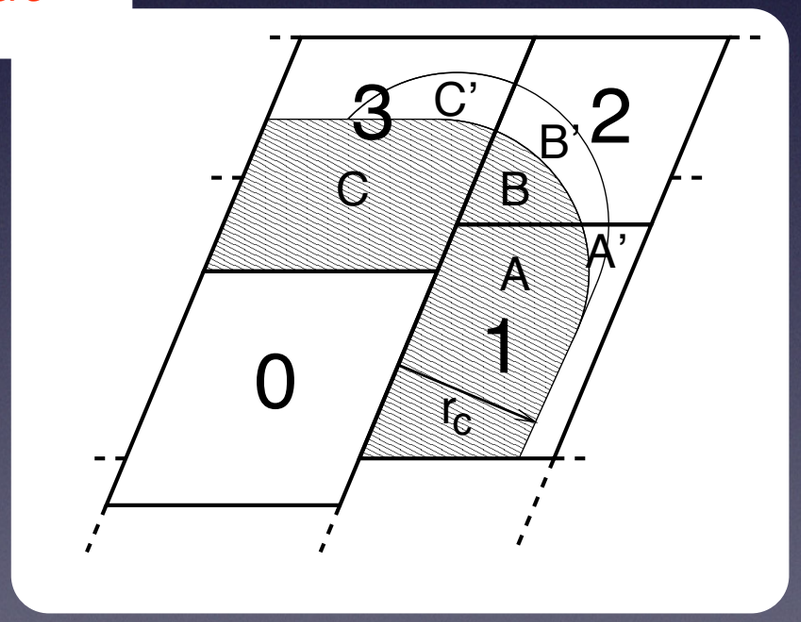
C) Correct for rotational extension of bond

These algorithms are complex to parallelize, but provide tremendous speedup
Performance is more important than relative scaling!

Example: Remove FLOPS by using smaller simulation boxes







Lysozyme, 25k atoms
Rhombic dodecahedron
(36k atoms in cubic cell)

Load balancing is tricky
for arbitrary triclinic cells

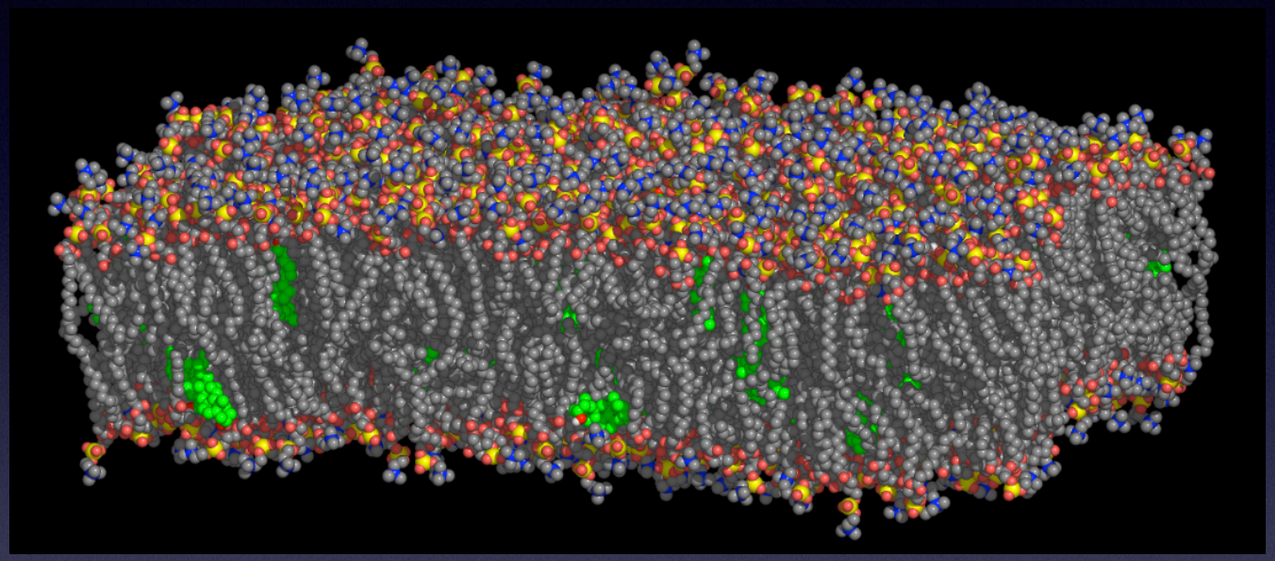
How do we find parallelism?

```
/* Get j neighbor index, and coordinate index */
jnr                 = jjnr[k];
j3                  = 3*jnr

/* load j atom coordinates */
jx1                 = pos[j3+0];
jy1                 = pos[j3+1];
jz1                 = pos[j3+2];

/* Calculate distance */
dx11                = ix1 - jx1;
dy11                = iy1 - jy1;
dz11                = iz1 - jz1;
rsq11               = dx11*dx11+dy11*dy11+dz11*dz11;

/* Calculate 1/r and 1/r2 */
rinv11              = 1.0/sqrt(rsq11);

/* Load parameters for j atom */
qq                  = iq*charge[jnr];
tj                  = nti+2*type[jnr];
c6                  = vdwparam[tj];
c12                 = vdwparam[tj+1];
rinvsq              = rinv11*rinv11;

/* Coulomb interaction */
vcoul               = qq*rinv11;
vctot               = vctot+vcoul;

/* Lennard-Jones interaction */
rinvsix             = rinvsq*rinvsq*rinvsq;
Vvdw6               = c6*rinvsix;
Vvdw12              = c12*rinvsix*rinvsix;
Vvdwtot             = Vvdwtot+Vvdw12-Vvdw6;
fscal               = (vcoul+12.0*Vvdw12-6.0*Vvdw6)*rinvsq;

/* Calculate temporary vectorial force */
tx                  = fscal*dx11;
ty                  = fscal*dy11;
tz                  = fscal*dz11;

/* Increment i atom force */
fix1                = fix1 + tx;
fiy1                = fiy1 + ty;
fiz1                = fiz1 + tz;

/* Decrement j atom force */
faction[j3+0]       = faction[j3+0] - tx;
faction[j3+1]       = faction[j3+1] - ty;
faction[j3+2]       = faction[j3+2] - tz;

/* Inner loop uses 38 flops/iteration */
}
```

i interaction

(Old scaling data from 2008)

DPPC & Cholesterol
130k atoms

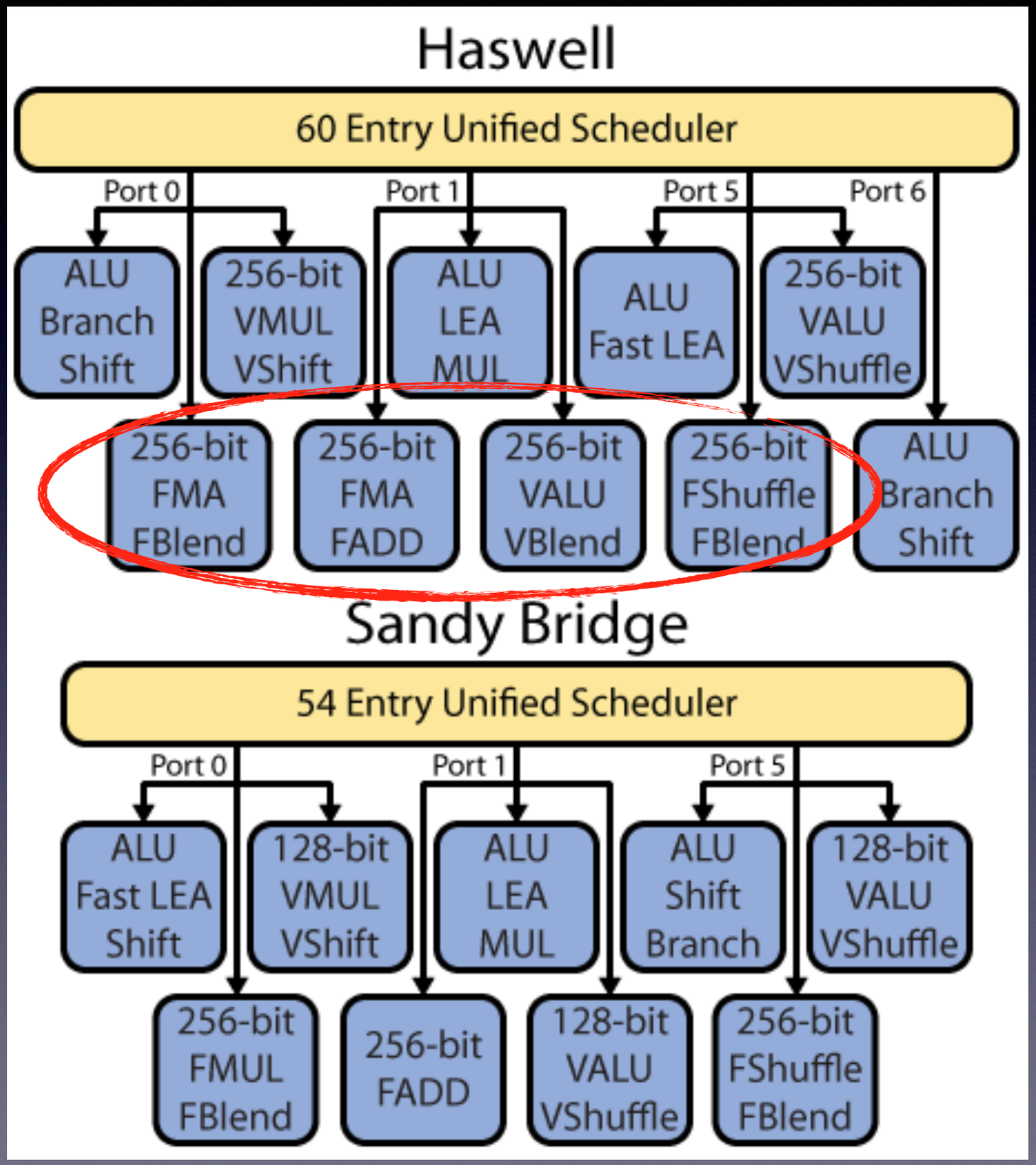


Blue Gene/L & Blue Matter:
scaled to 3 atoms/CPU
~10ns/day on 8192 CPUs

GROMACS 3: 2ns/day

...on a single dual dual-core Opteron!

What does a modern CPU look like?

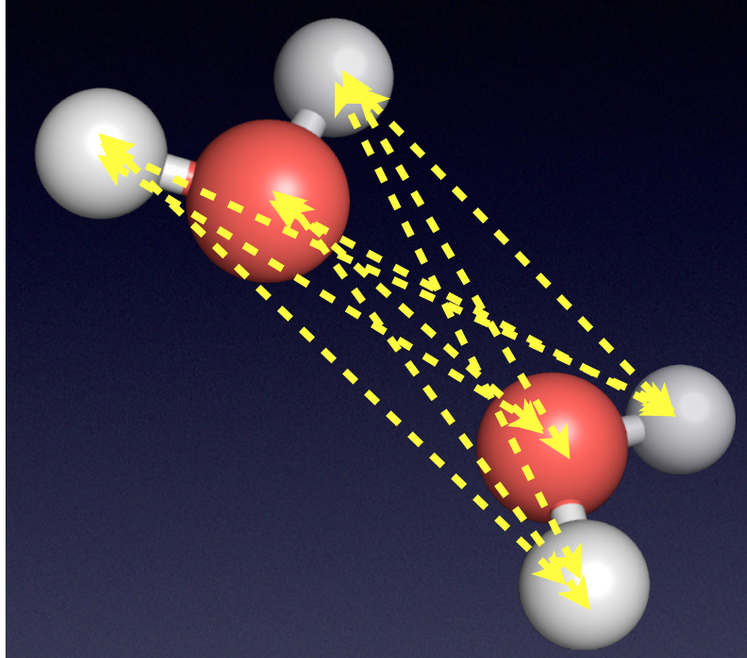


SIMD:

Single
Instruction
Multiple
Data

Execute 4 iterations of the innermost loop at once

```
for(k=nj0; (k<nj1); k++)
{

    /* Get j neighbor index, and coordinate index */
    jnr             = jjnr[k];
    j3              = 3*jnr;

    /* load j atom coordinates */
    jx1             = pos[j3+0];
    jy1             = pos[j3+1];
    jz1             = pos[j3+2];

    /* Calculate distance */
    dx11            = ix1 - jx1;
    dy11            = iy1 - jy1;
    dz11            = iz1 - jz1;
    rsq11           = dx11*dx11+dy11*dy11+dz11*dz11;

    /* Calculate 1/r and 1/r2 */
    rinv11          = 1.0/sqrt(rsq11);

    /* Load parameters for j atom */
    qq              = iq*charge[jnr];
    tj              = nti+2*type[jnr];
    c6              = vdwparam[tj];
    c12             = vdwparam[tj+1];
    rinvsq          = rinv11*rinv11;

    /* Coulomb interaction */
    vcoul           = qq*rinv11;
    vctot           = vctot+vcoul;

    /* Lennard-Jones interaction */
    rinvsix         = rinvsq*rinvsq*rinvsq;
    Vvdw6           = c6*rinvsix;
    Vvdw12          = c12*rinvsix*rinvsix;
    Vvdwtot         = Vvdwtot+Vvdw12-Vvdw6;
    fscal           = (vcoul+12.0*Vvdw12-6.0*Vvdw6)*rinvsq;

    /* Calculate temporary vectorial force */
    tx              = fscal*dx11;
    ty              = fscal*dy11;
    tz              = fscal*dz11;

    /* Increment i atom force */
    fix1            = fix1 + tx;
    fiy1            = fiy1 + ty;
    fiz1            = fiz1 + tz;

    /* Decrement j atom force */
    faction[j3+0]   = faction[j3+0] - tx;
    faction[j3+1]   = faction[j3+1] - ty;
    faction[j3+2]   = faction[j3+2] - tz;

    /* Inner loop uses 38 flops/iteration */
}
```



2 waters

1 neighborlist entry

9 interactions

O-O: Coulomb & L-J

All other: Coulomb

a_0	a_1	a_2	a_3

+

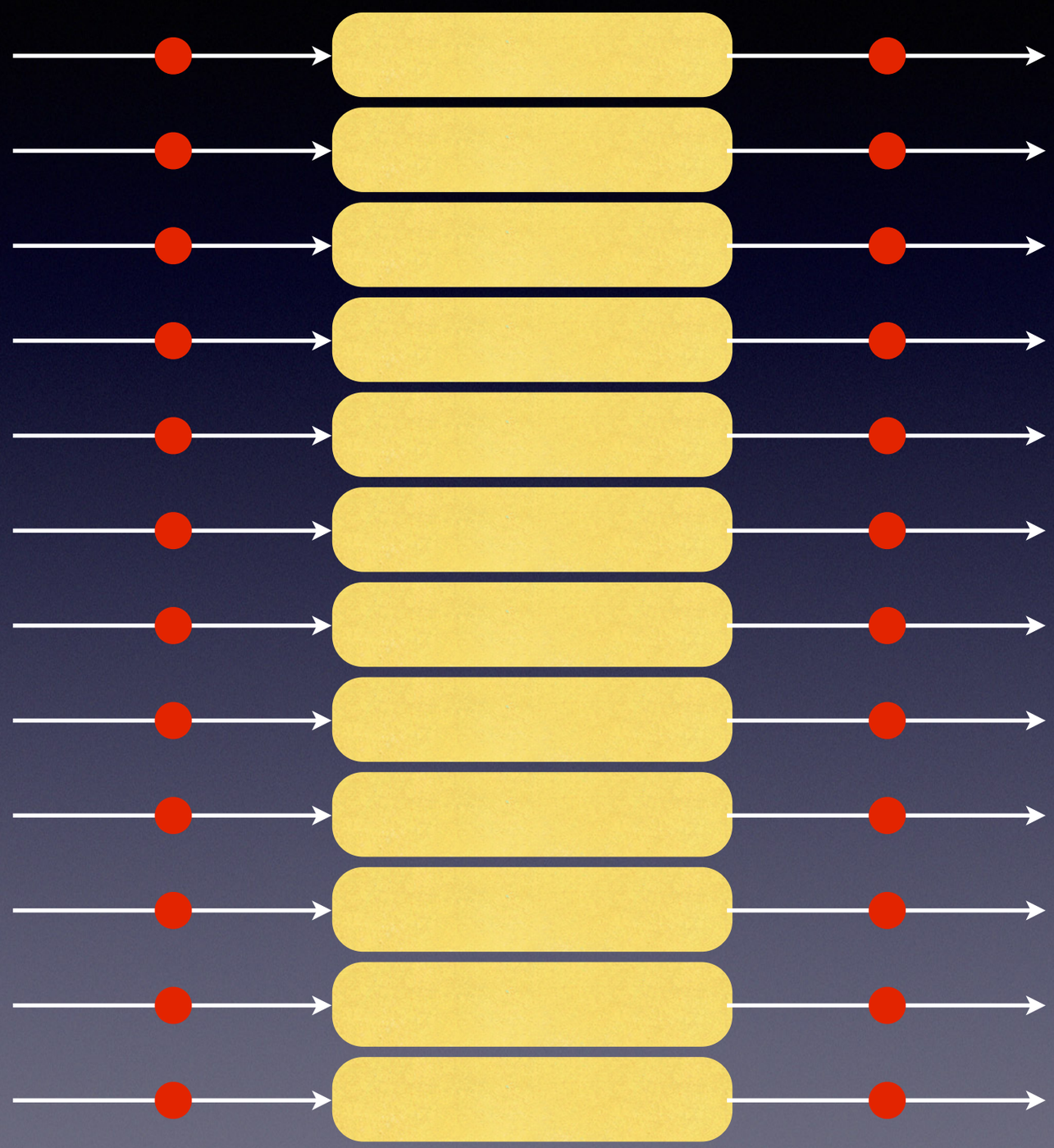
b_0	b_1	b_2	b_3

=

c_0	c_1	c_2	c_3

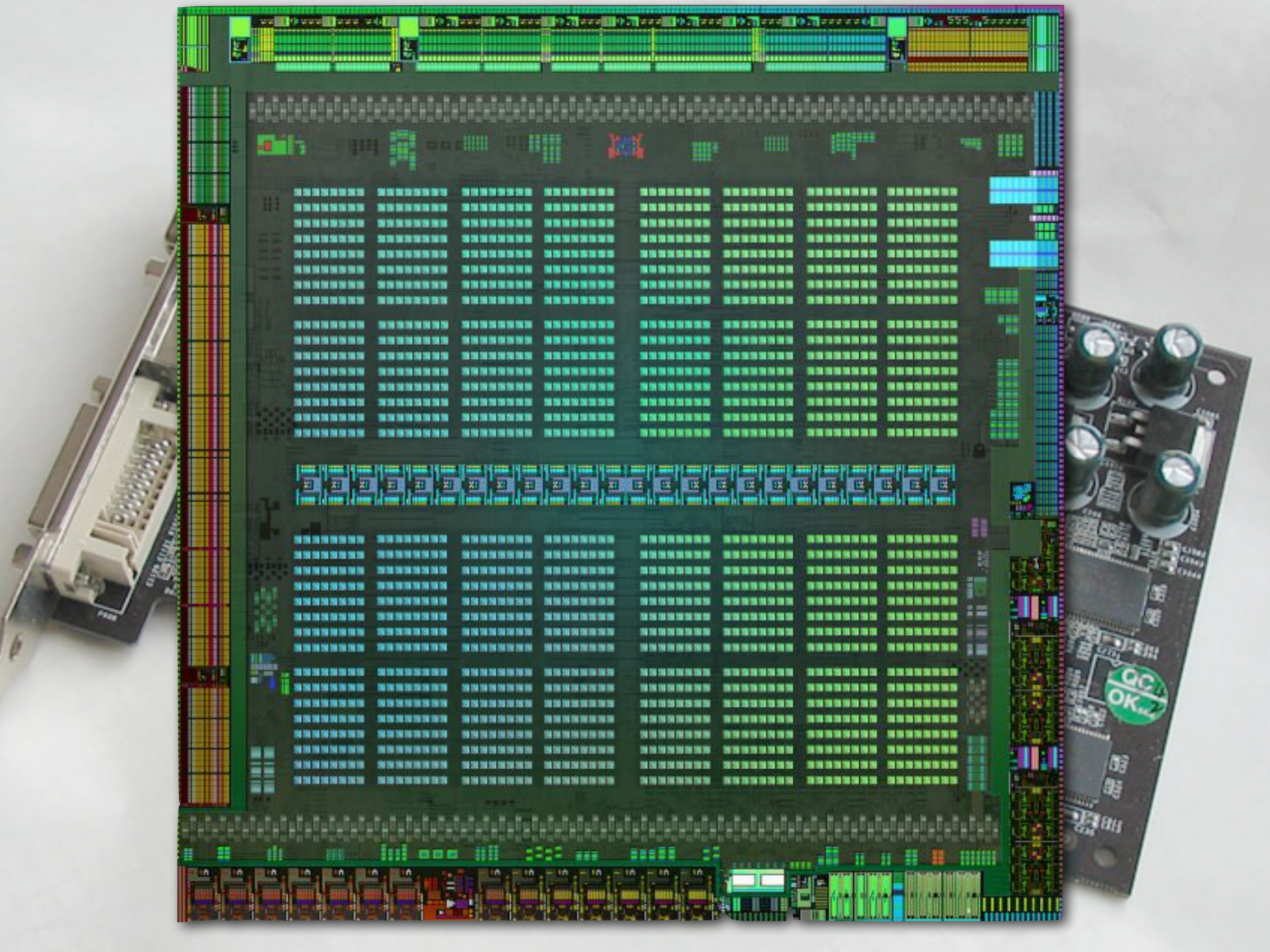
This has served us very well for 10+ years, but it's no longer good enough: We are spending way too much time shuffling data to fit 8-way SIMD registers

Explicit Data Parallelism



Stream=your data
Kernel=algoritm

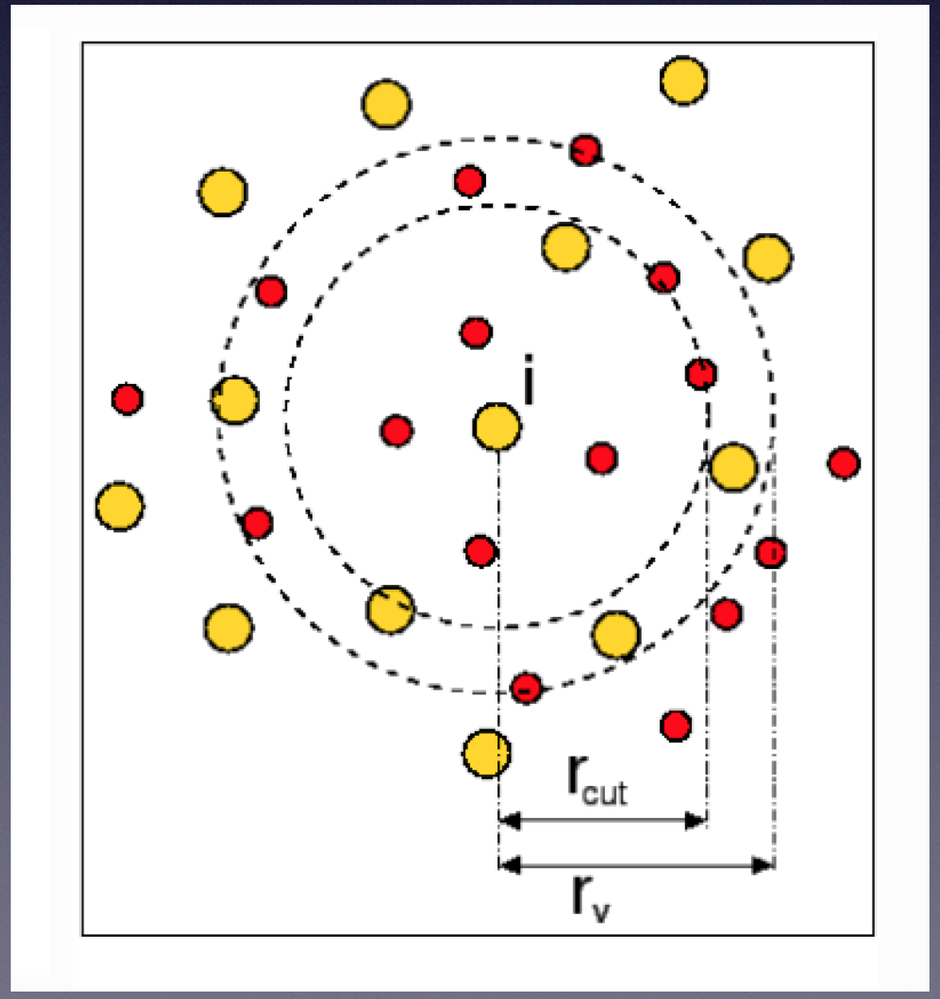
Without dependencies, all could be done in parallel if enough hardware was available!


QC
OK

It is much easier to port and scale a simple reference program

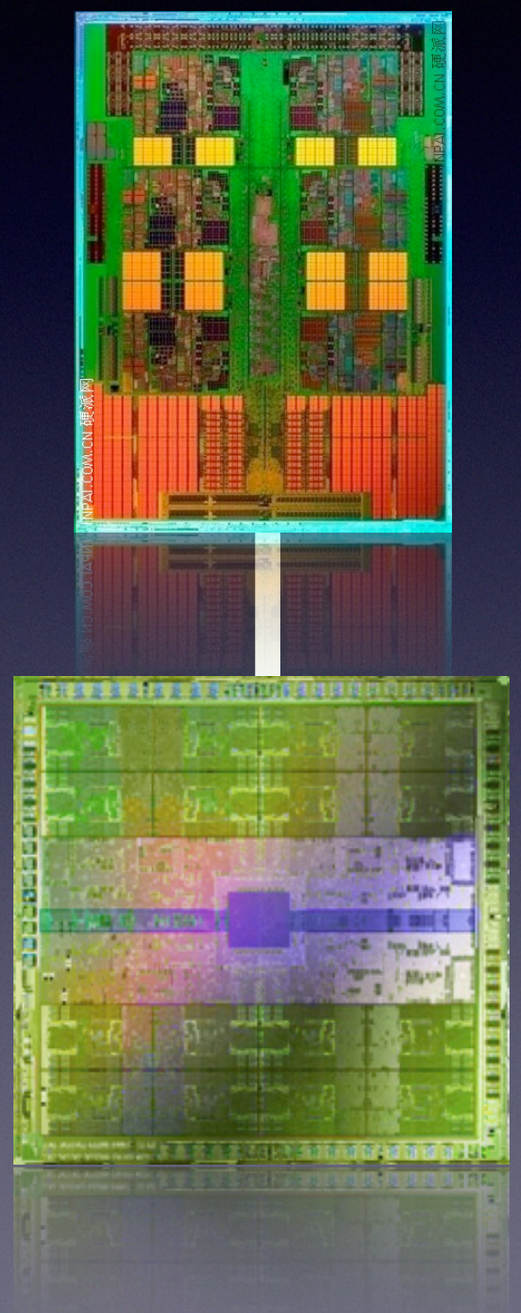
i.e., you see much better *relative* scaling before introducing any optimization

Our first GPU-try was 100x slower than running on CPUs...



A failed GPU attempt?

Gromacs running entirely on CPU as an interface

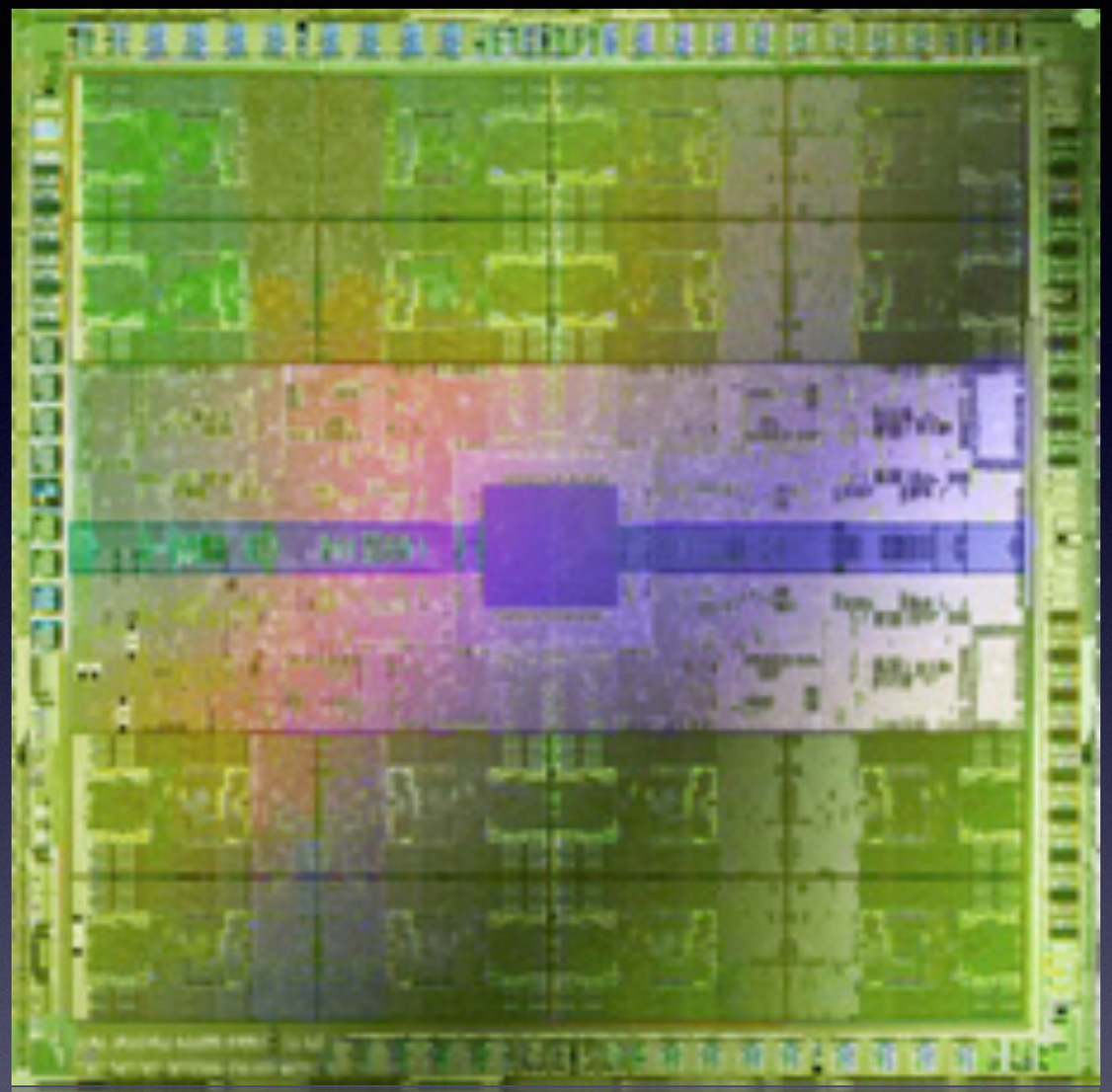


Actual simulation running entirely on GPU using OpenMM kernels

Only a few select algorithms worked

Multi-CPU usually beat GPU performance...

Option 1: Stay on the GPU

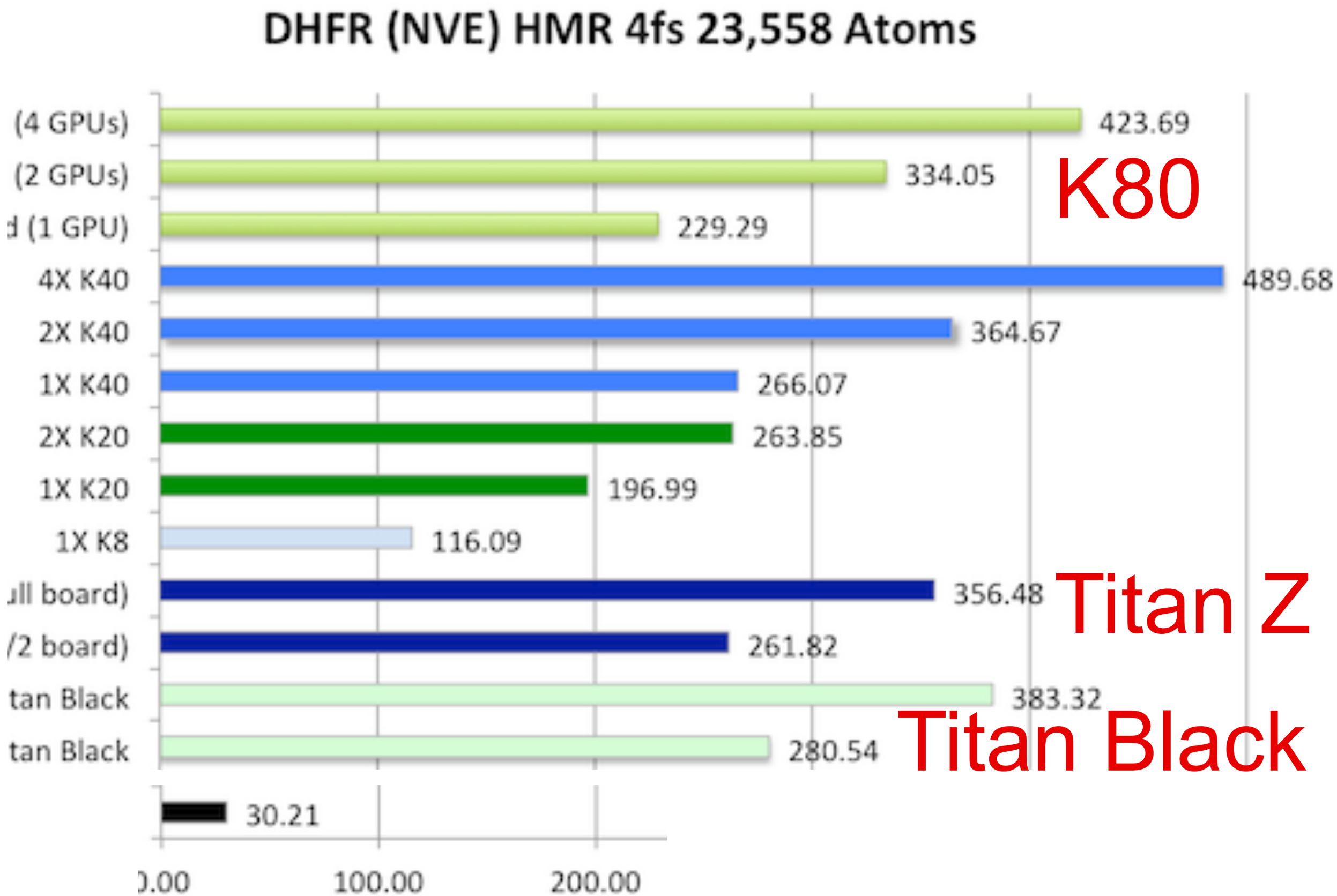


This avoids the CPU to GPU PCIe bottleneck completely

CPU irrelevant, any node will work

Awesome MD performance with AMBER

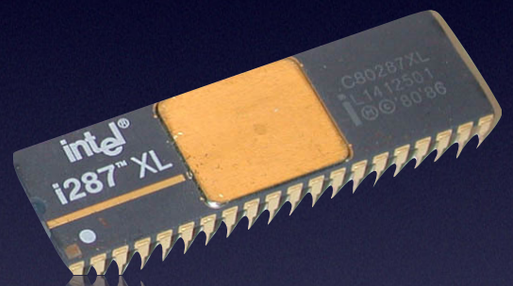
AMBER


DHFR (NVE) HMR 4fs 23,558 Atoms
(4 GPUs)
(2 GPUs)
d (1 GPU)
4X K40
2X K40
1X K40
2X K20
1X K20
1X K8
ull board)
/2 board)
tan Black
tan Black
423.69
334.05
229.29
489.68
364.67
266.07
263.85
196.99
116.09
356.48
261.82
383.32
280.54
30.21
0.00
100.00
200.00
K80
Titan Z
Titan Black

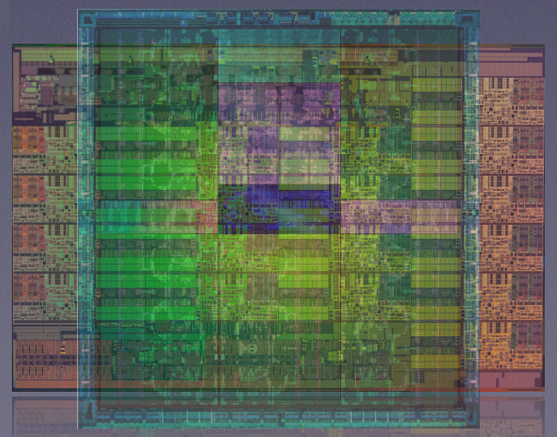
"I skate to where the puck is going to be, not where it has been."

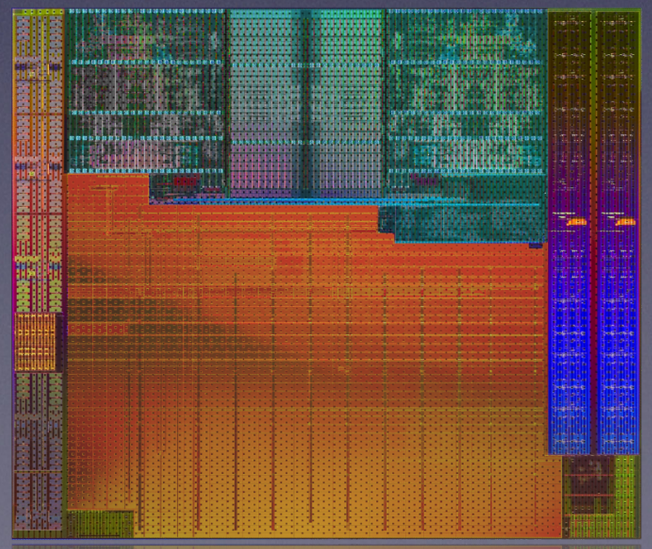
- I like my 1980s integer unit, thank you, so I'll emulate all floating-point there

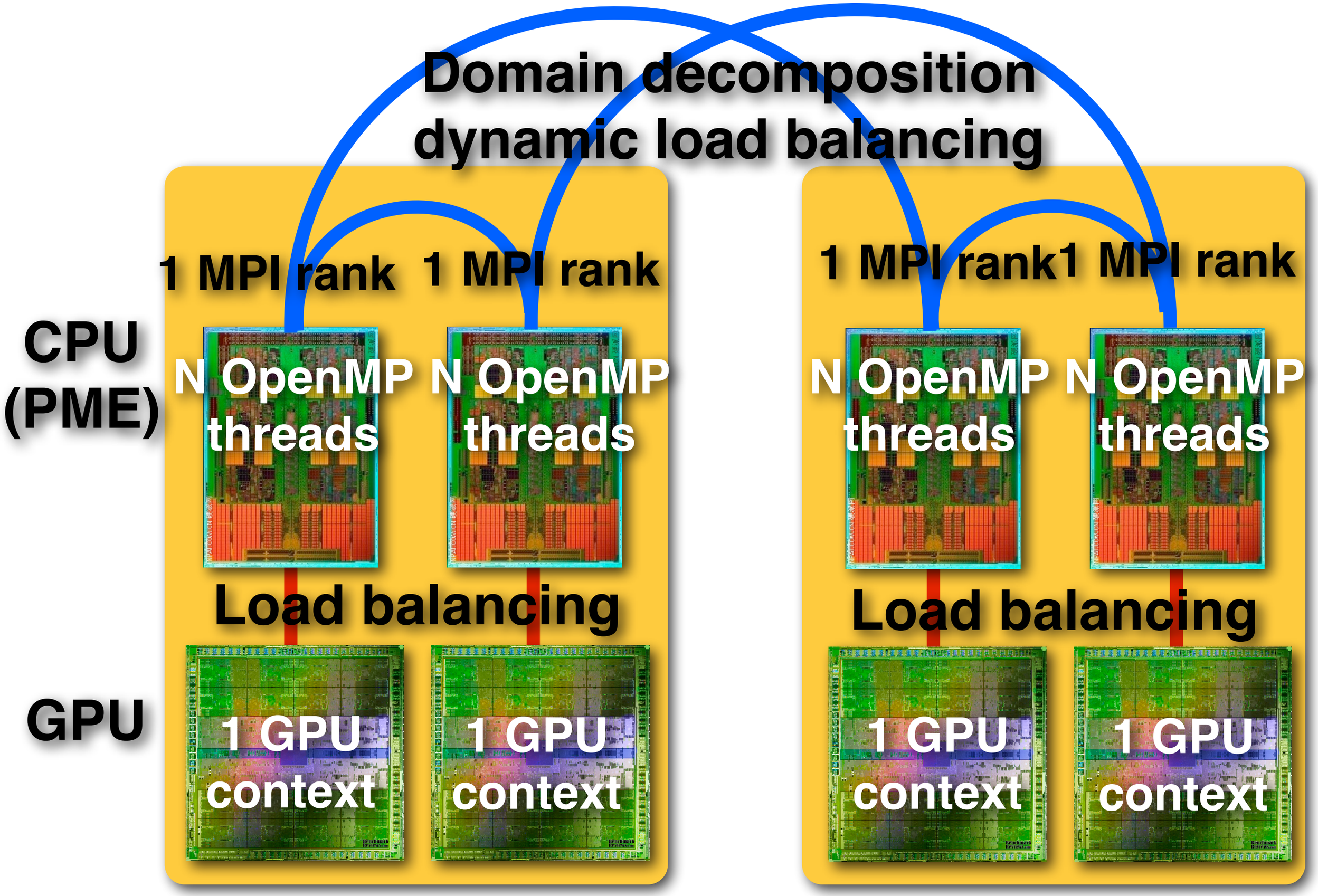
- I'm a floating-point person, so I always use floating-point variables as my for-loops counters



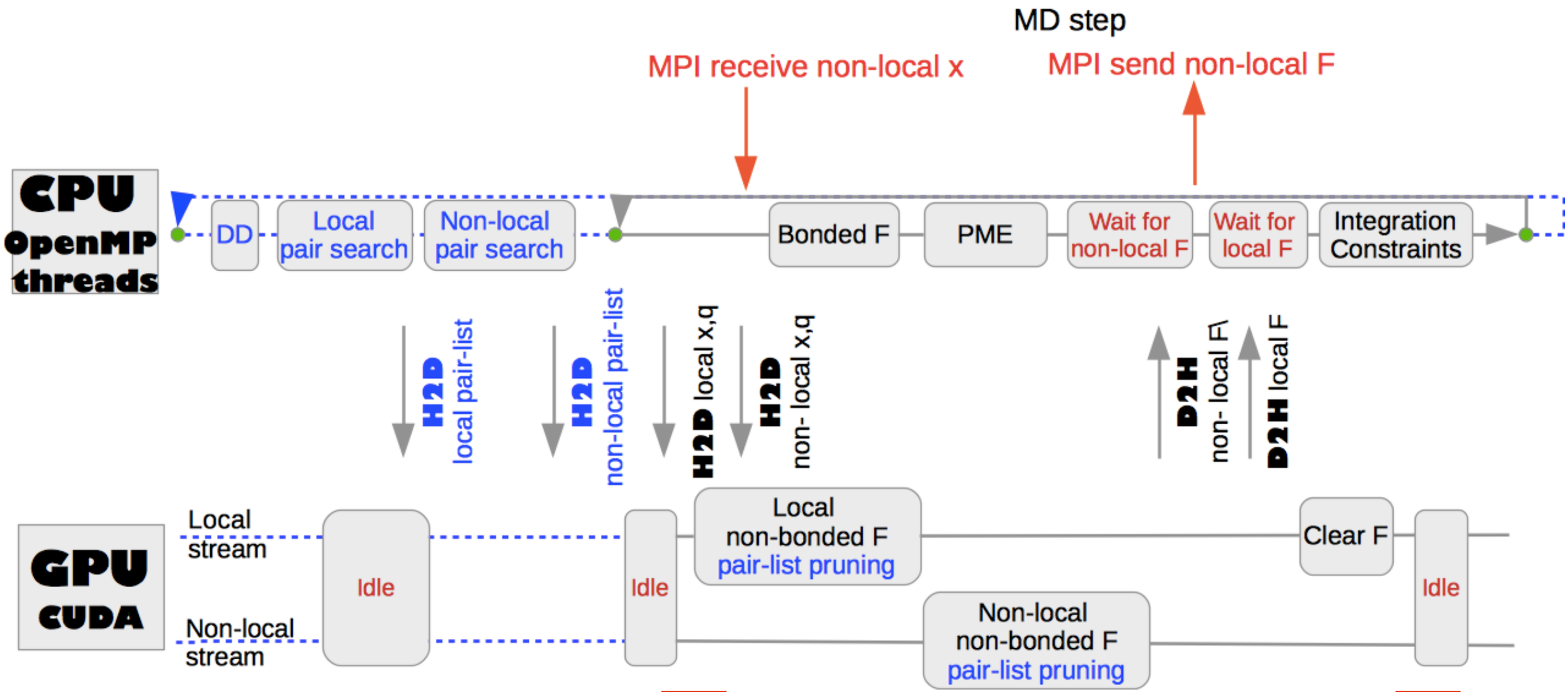
- Any other ideas?






Domain decomposition
dynamic load balancing
1 MPI rank
1 MPI rank
1 MPI rank
1 MPI rank
CPU
(PME)
N OpenMP threads
N OpenMP threads
N OpenMP threads
N OpenMP threads
Load balancing
Load balancing
GPU
1 GPU context
1 GPU context
1 GPU context
1 GPU context

Heterogeneous CPU-GPU acceleration in GROMACS



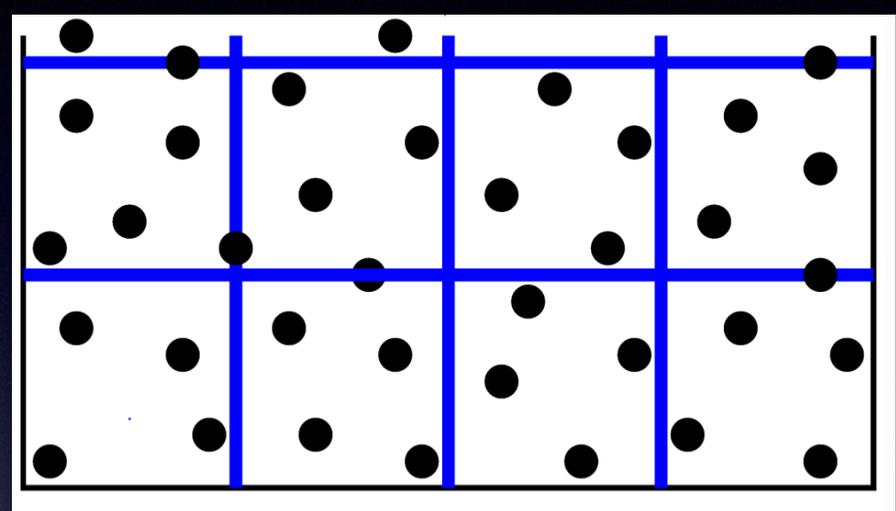
Wallclock time for a step:
~0.5 ms if we want to simulate 1μs/day
We cannot afford to lose all previous acceleration tricks!

100-500 μs

Problem:

You cannot use neighborlists…

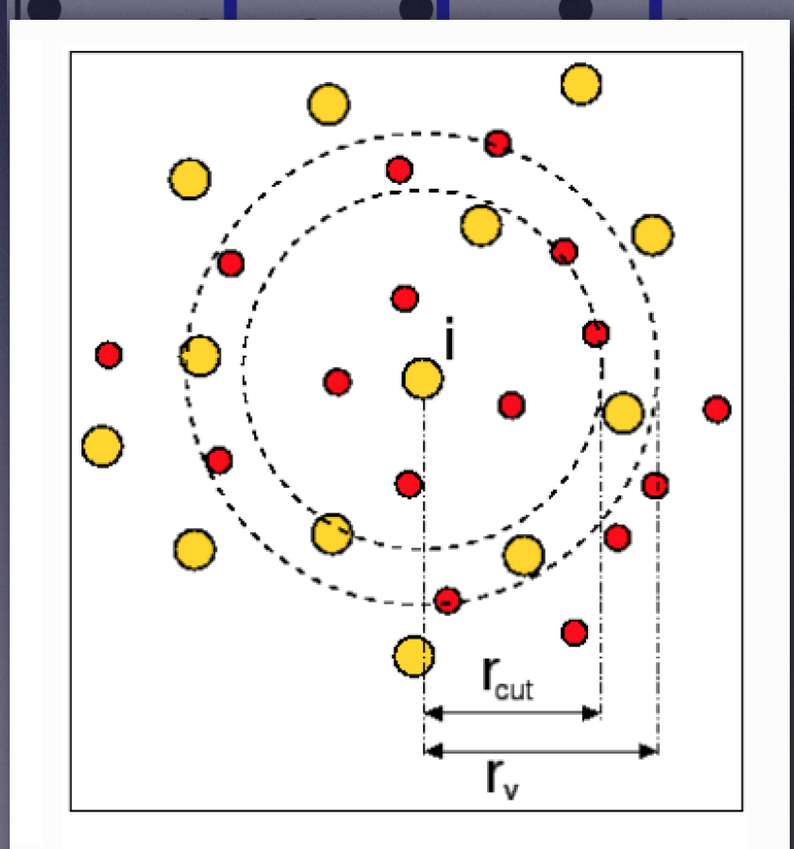
The Link-cell algorithm: **Verlet, Phys Rev 159, 98-103 (1967)]**



i=3:	5	6	9	12	15	17	18	25	32	…		
i=4:	7	8	9	11	12	15	17	25	32	43	54	…
…	8	9	10	11	12	13	19	20	…			

…

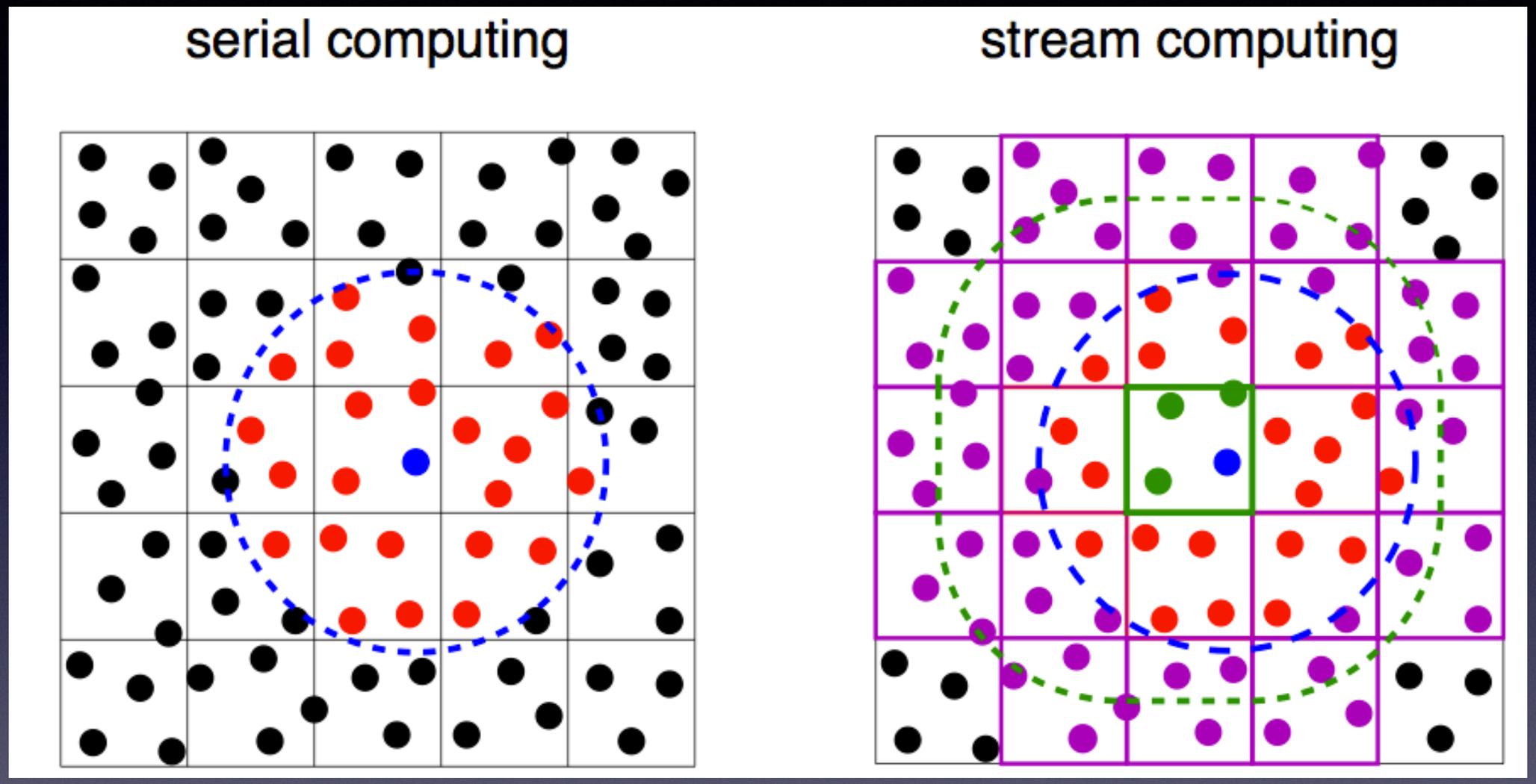
Load 1 atom, then compute 1 force



Too much data to send each step, each atom has different neighbors, memory bottleneck: Won't work well on GPUs!

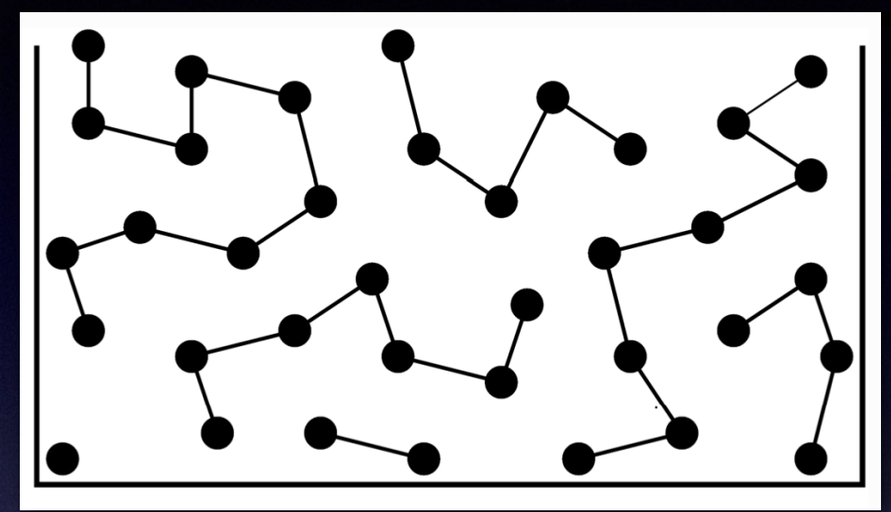
Traditional solution:
Group interactions into "tiles"
Load 4 atoms, then compute 16 forces

Tiling circles is difficult

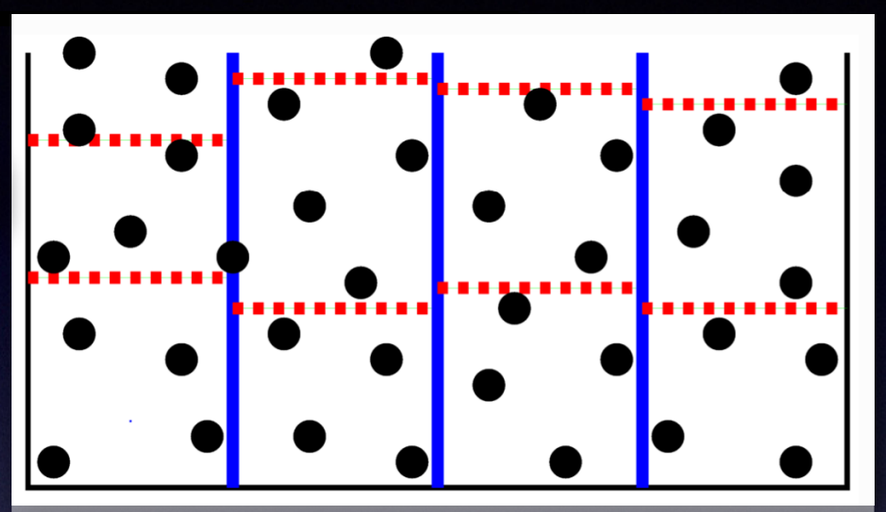


- You need a lot of cubes to cover a sphere
- All interactions beyond cutoff need to be zero

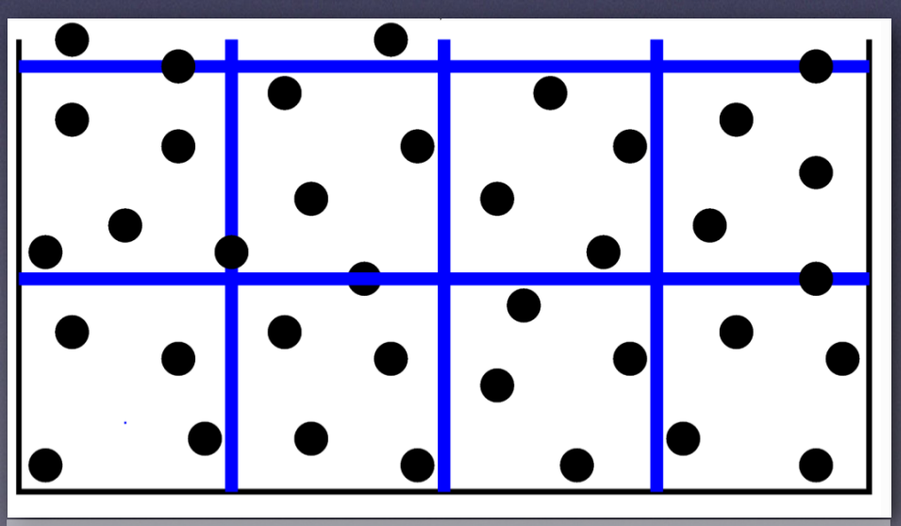
From neighborlists to cluster proximity lists



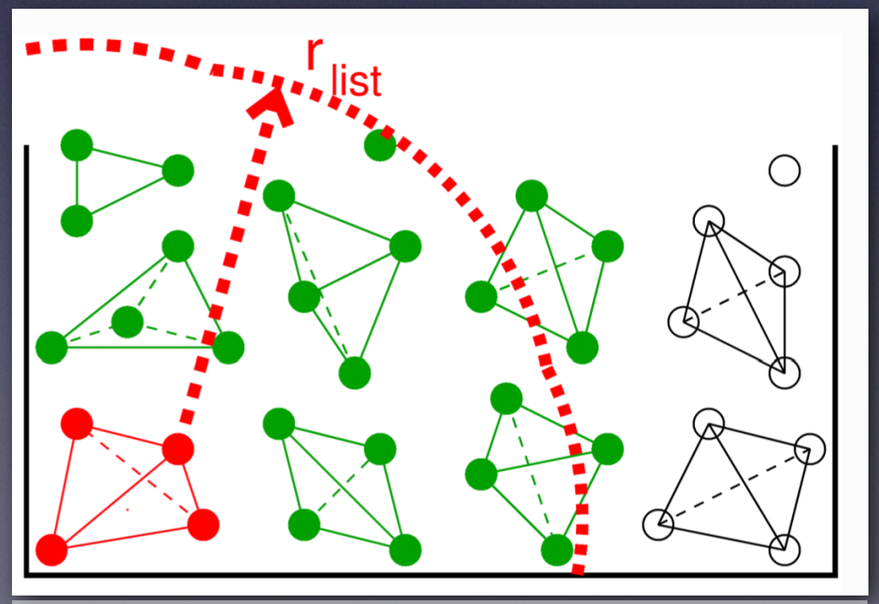
x,y grid
z sort
z bin



x,y,z
gridding

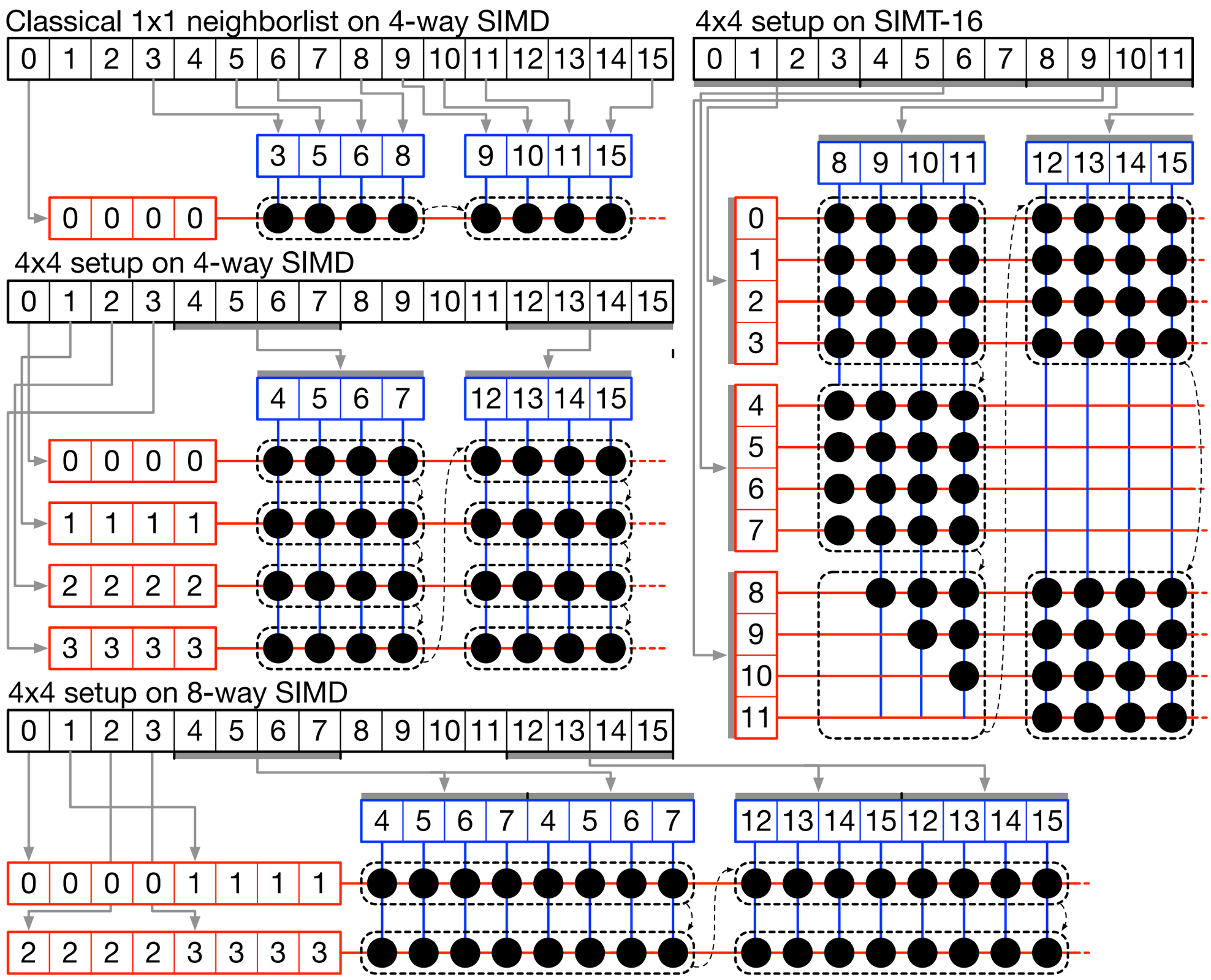


Cluster pairlist



Organize
as tiles with
all-vs-all
interactions:

Unified GPU/CPU architecture - completely portable



CUDA
OpenCL
Intel MIC
x86 SSE2
x86 SSE4.1
x86 AVX
x86 AVX2
x86 AVX-512
Arm Neon
Arm64 Asimd
IBM QPX
IBM VMX
IBM VSX
Fujitsu HPC-ACE

Wanted: Fujitsu HPC-ACE2

Surprisingly little CUDA code

```
n_cuda lindahl$ ls -ltar

lindahl  staff  13012 Apr 18 15:10 nbnxn_cuda_types.h
lindahl  staff   9155 Apr 18 15:10 nbnxn_cuda_kernels.cuh
lindahl  staff  21576 Apr 18 15:10 nbnxn_cuda_kernel_utils.cuh
lindahl  staff  20945 Apr 18 15:10 nbnxn_cuda_kernel.cuh
lindahl  staff   1965 Apr 18 15:10 CMakeLists.txt
lindahl  staff  39049 Apr 18 15:10 nbnxn_cuda_data_mgmt.cu
lindahl  staff   3667 Apr 18 15:10 nbnxn_cuda.h
lindahl  staff  30920 May 22 09:13 nbnxn_cuda.cu
lindahl  staff   2686 May 22 09:13 ..
lindahl  staff    340 May 22 09:13 .
```

A total of ~3500 lines of CUDA, compared to 3 million lines of C/C++

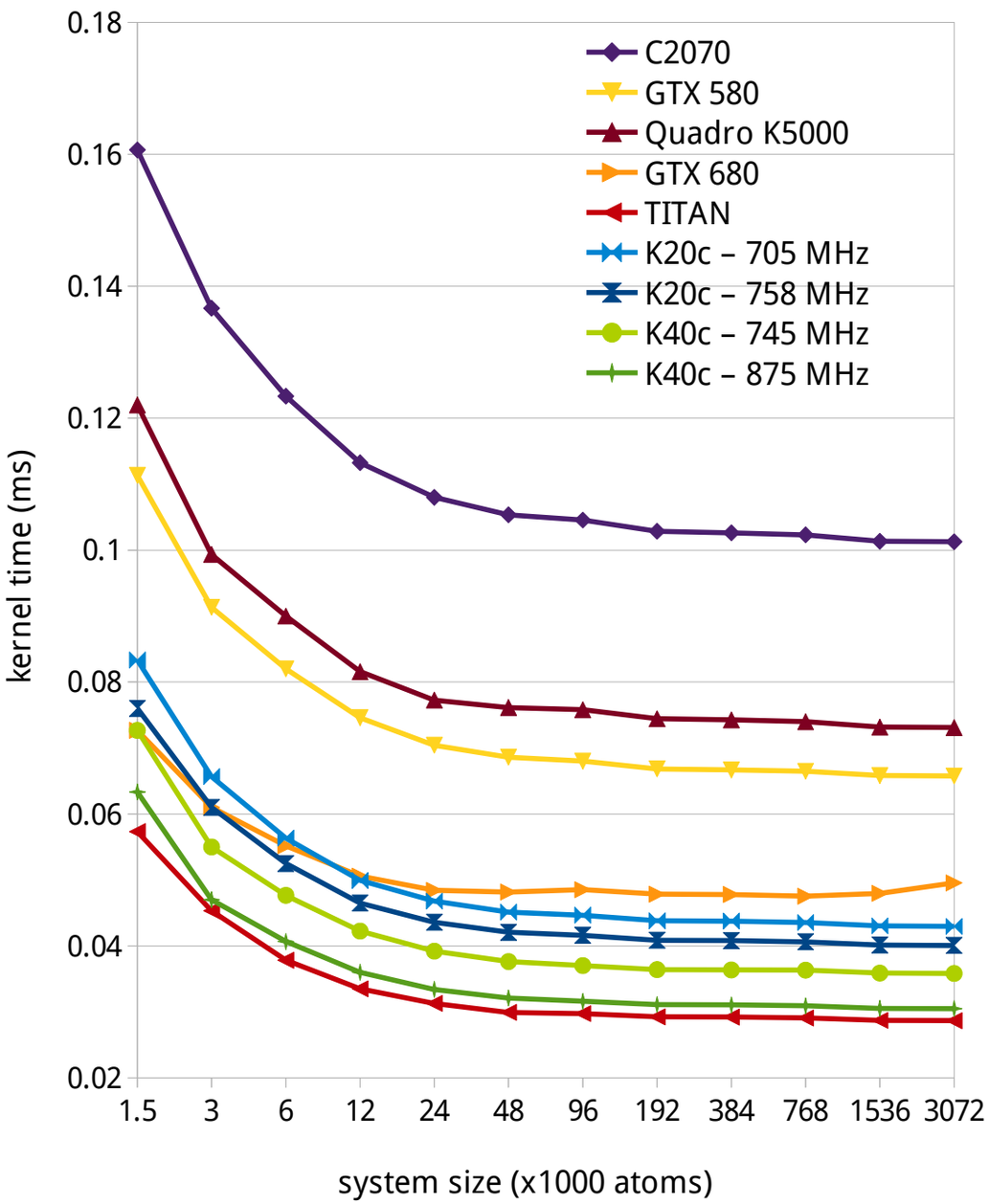
cuda_kernel.c

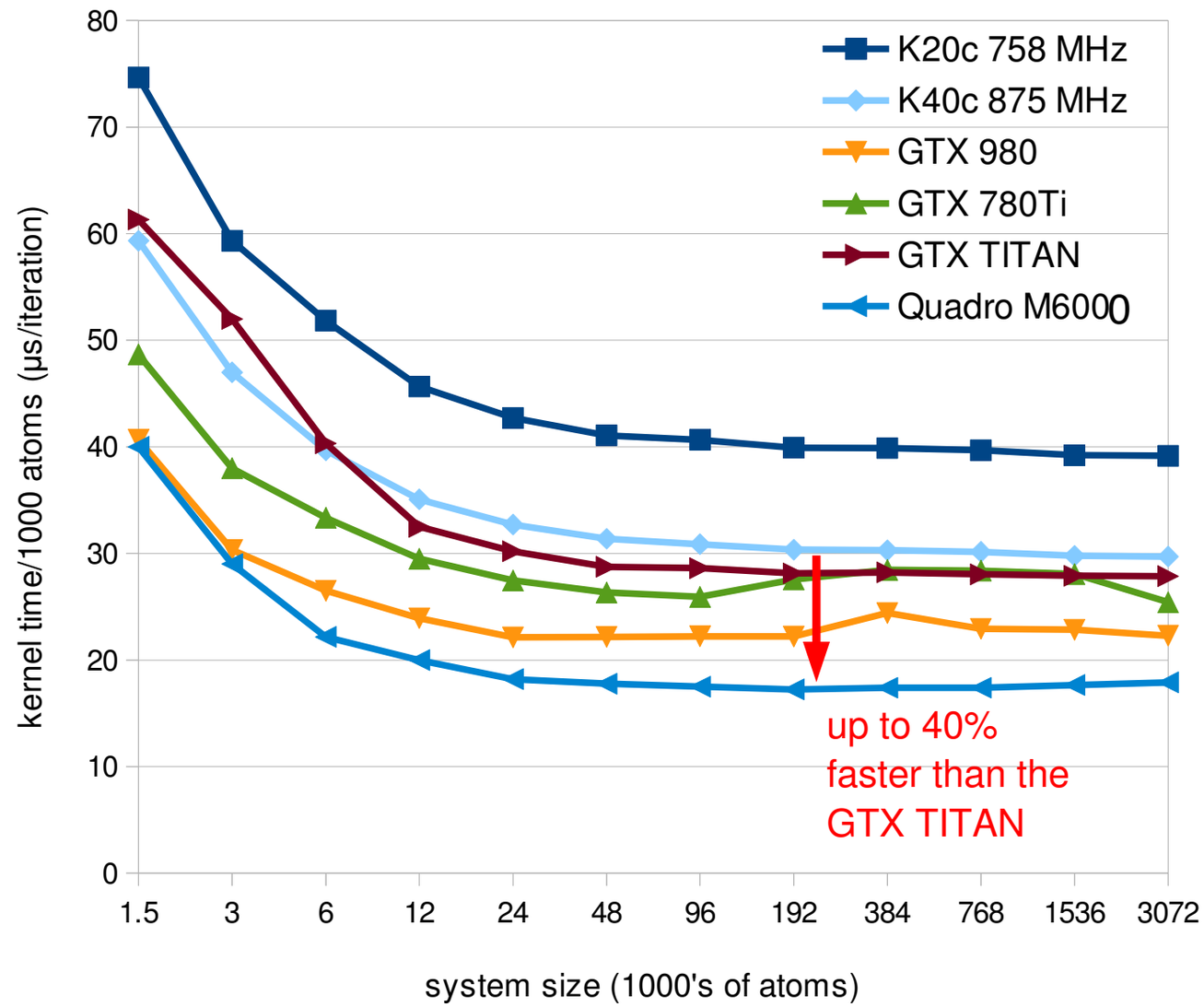
cuda_kernel.c 〉 No Selection

```
351                                 {
352                                     /* load the rest of the i-atom parameters */
353                                     qi      = xqbuf.w;
354 #ifdef IATYPE_SHMEM
355                                     typei   = atib[i * CL_SIZE + tidxi];
356 #else
357                                     typei   = atom_types[ai];
358 #endif
359
360                                     /* LJ 6*C6 and 12*C12 */
361 #ifdef USE_TEXOBJ
362                                     c6      = tex1Dfetch<float>(nbparam.nbfp_texobj, 2 * (ntypes * typei + typej));
363                                     c12     = tex1Dfetch<float>(nbparam.nbfp_texobj, 2 * (ntypes * typei + typej) + 1);
364 #else
365                                     c6      = tex1Dfetch(nbfp_texref, 2 * (ntypes * typei + typej));
366                                     c12     = tex1Dfetch(nbfp_texref, 2 * (ntypes * typei + typej) + 1);
367 #endif                              /* USE_TEXOBJ */
368
369
370                                     /* avoid NaN for excluded pairs at r=0 */
371                                     r2      += (1.0f - int_bit) * NBNXN_AVOID_SING_R2_INC;
372
373                                     inv_r   = rsqrt(r2);
374                                     inv_r2  = inv_r * inv_r;
375                                     inv_r6  = inv_r2 * inv_r2 * inv_r2;
376 #if defined EXCLUSION_FORCES
377                                     /* We could mask inv_r2, but with Ewald
378                                      * masking both inv_r6 and F_invr is faster */
379                                     inv_r6  *= int_bit;
380 #endif                              /* EXCLUSION_FORCES */
381
382                                     F_invr  = inv_r6 * (c12 * inv_r6 - c6) * inv_r2;
383 #if defined CALC_ENERGIES || defined LJ_POT_SWITCH
384                                     E_lj_p  = int_bit * (c12 * (inv_r6 * inv_r6 + nbparam.repulsion_shift.cpot)*ONE_TWELVETH_F -
385                                                          c6 * (inv_r6 + nbparam.dispersion_shift.cpot)*ONE_SIXTH_F);
386 #endif
387
388 #ifdef LJ_FORCE_SWITCH
389 #ifdef CALC_ENERGIES
390                                     calculate_force_switch_F_E(nbparam, c6, c12, inv_r, r2, &F_invr, &E_lj_p);
391 #else
392                                     calculate_force_switch_F(nbparam, c6, c12, inv_r, r2, &F_invr);
393 #endif /* CALC_ENERGIES */
394 #endif /* LJ_FORCE_SWITCH */
395
```

Kernel timing

Single exec per step for 1 GPU

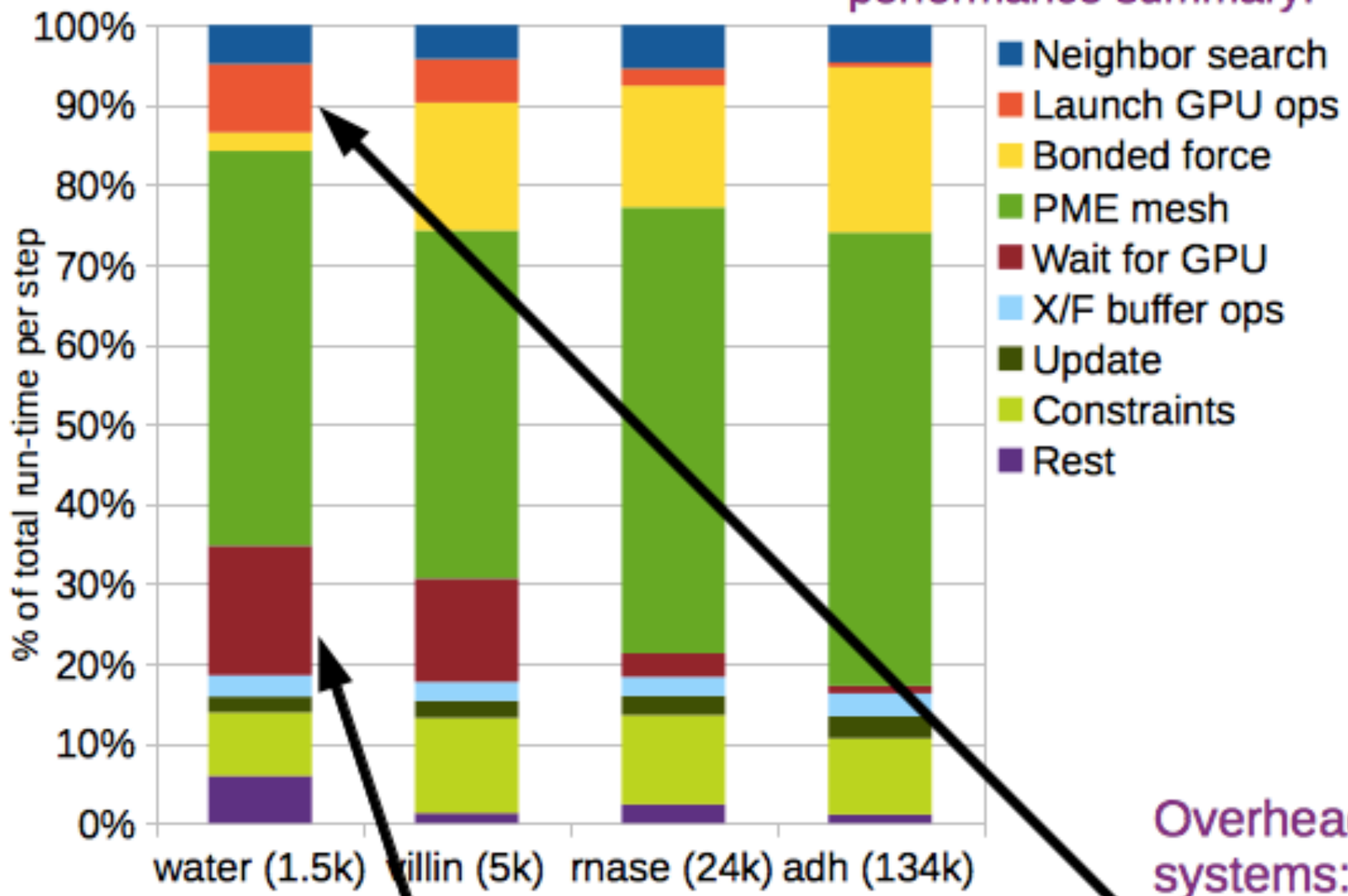




We are starting to use a lot of integer ops too for pruning & tweaking

CUDA overhead & scaling issues

Straight from the log file performance summary:



Runtime breakdown of GPU accelerated runs with:

- Hardware: Intel Core i7-3930 12T + GeForce GTX 680
- Settings: PME, rc>=0.9, nstlist=20

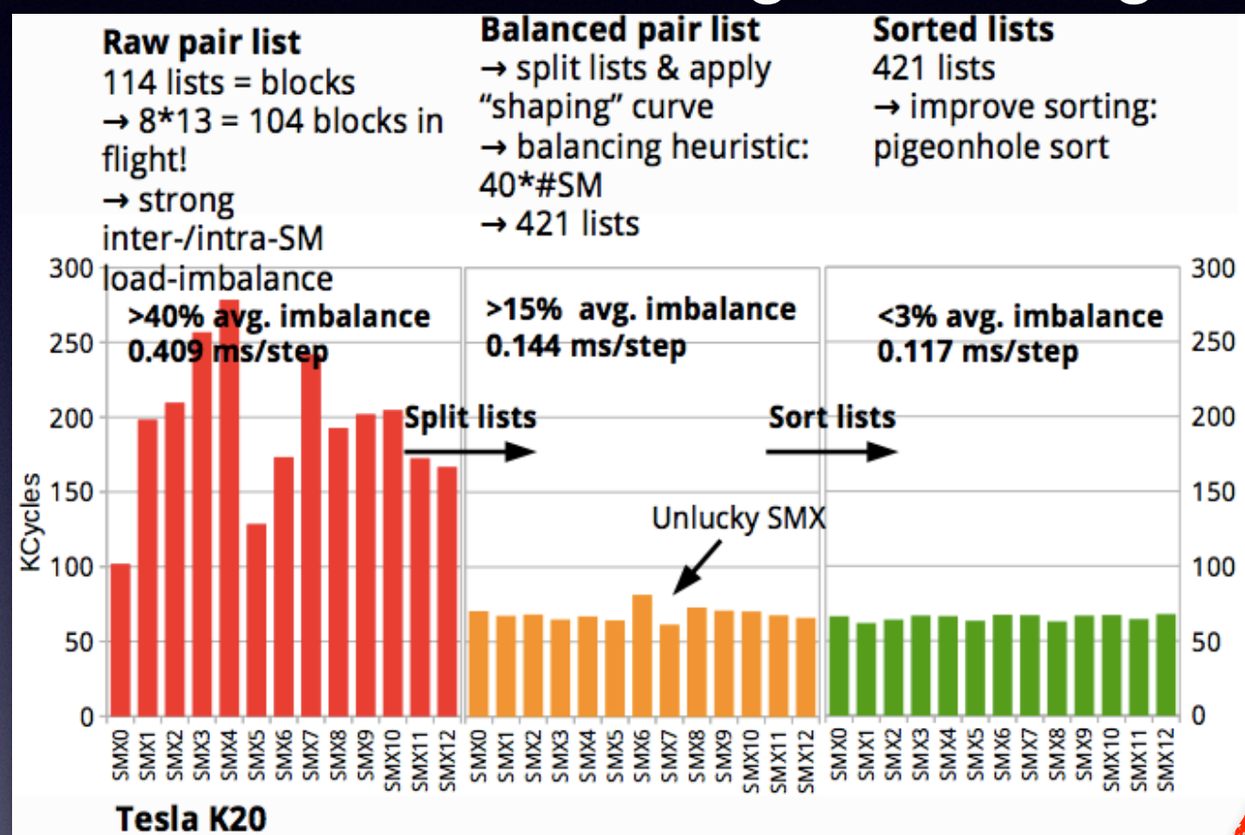
Overhead with small systems: launching the GPU operations takes up to 15%!

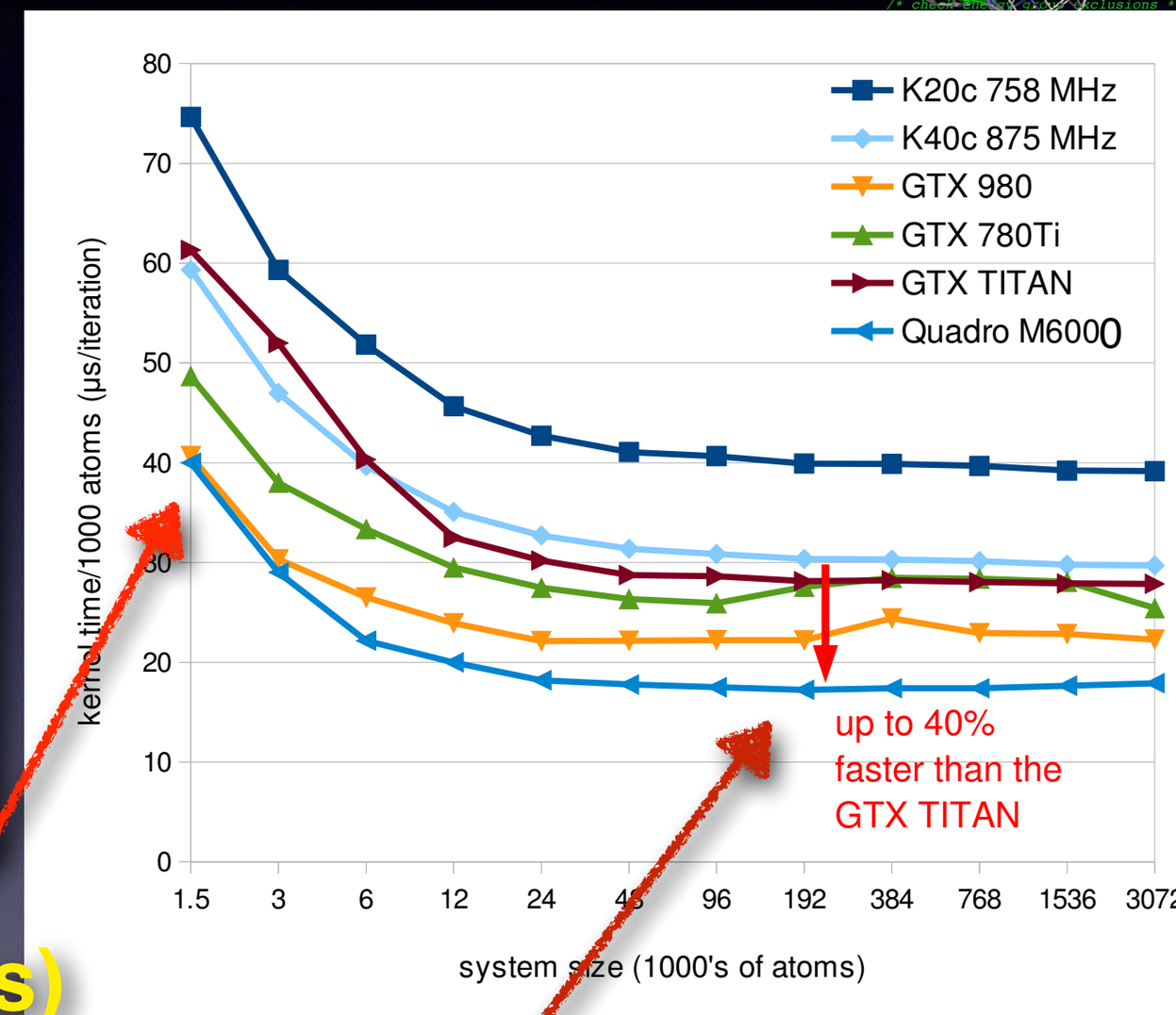
Kernel scaling deteriorating:
the GPU can't keep up with the CPU
=> CPU waiting

GPUs not designed for
~0.2 ms/step = 5000 FPS

A lot of low-level tuning

GPU SMX scheduling/balancing





60µs actual time (1500 atoms)

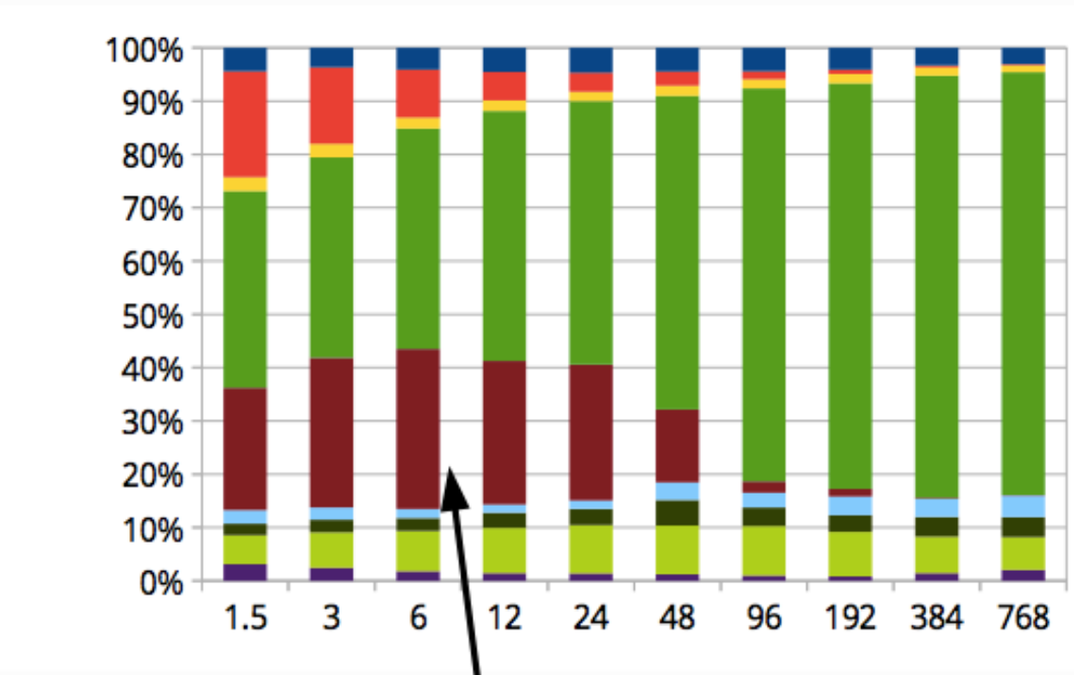
If we solve all latency bottlenecks, we would be below 20µs

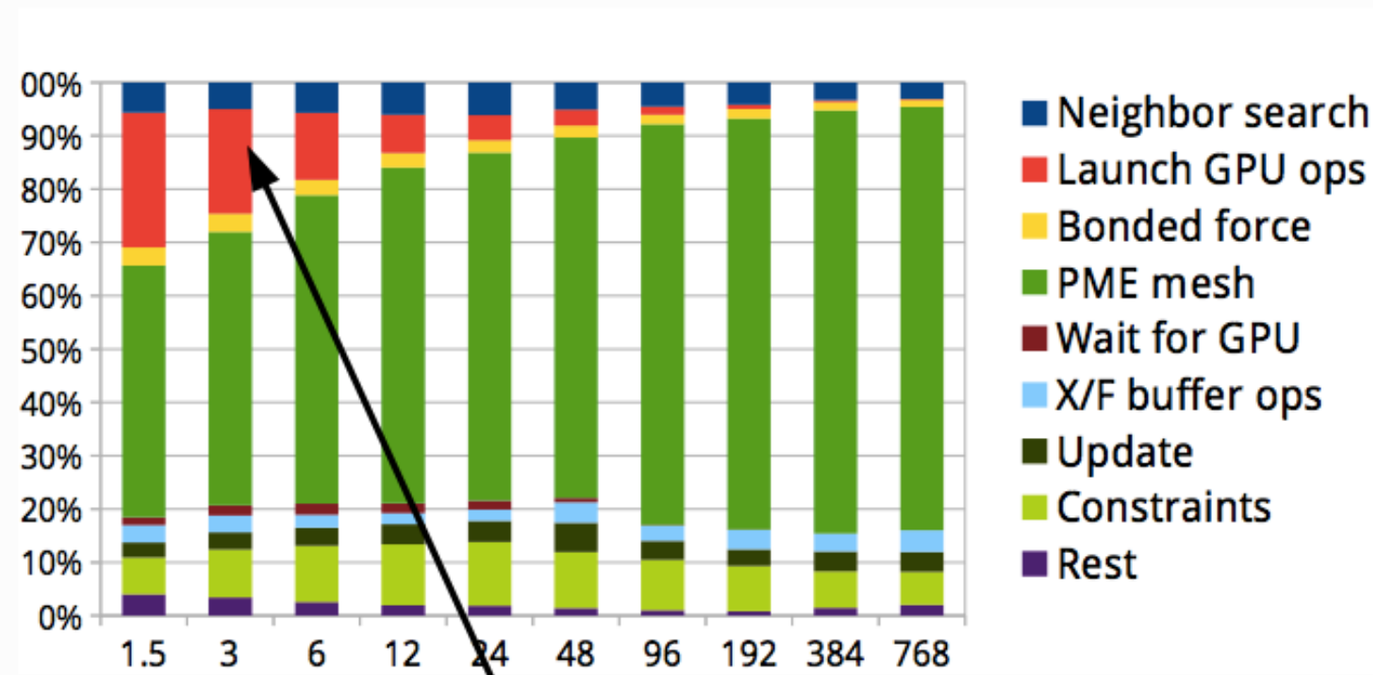
Integrating the GPU cont.

Run-time breakdown for varying system sizes:

- System: water 1.5k-768k
- PME, rc>=0.9, nstlist=20

Hardware: Intel Core i7-3930 12T + GTX680

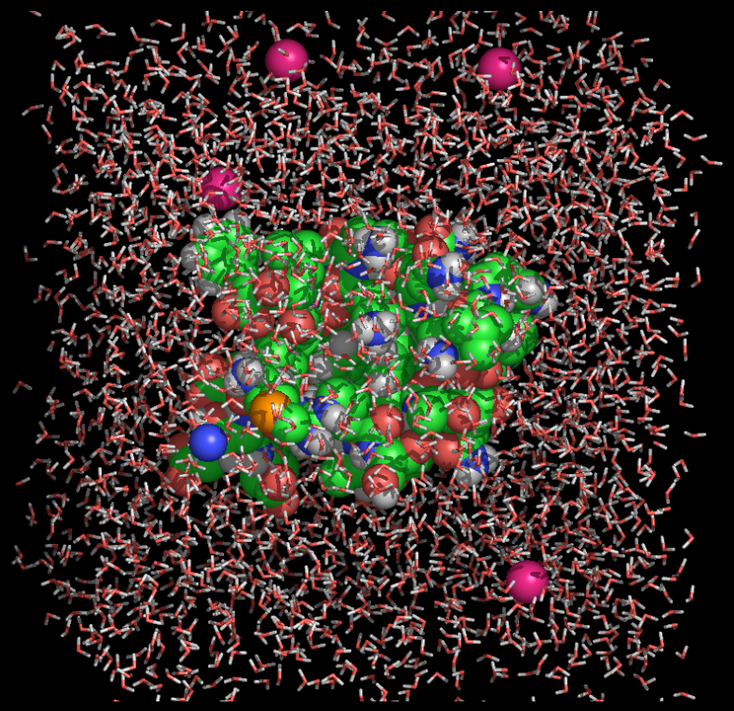




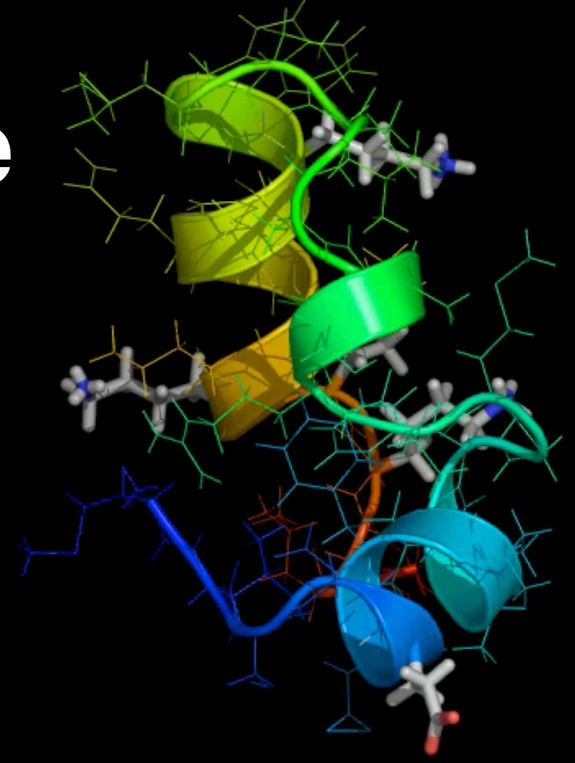
Kernel scaling deteriorating: CPU waiting

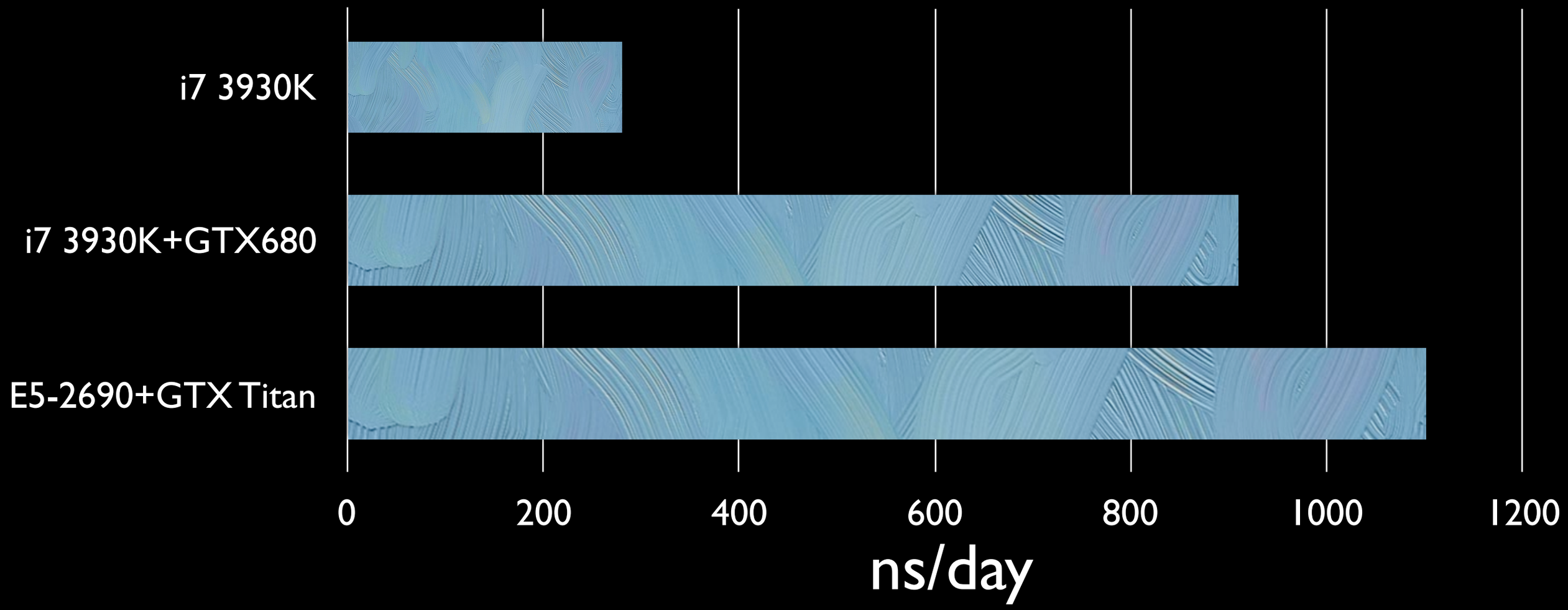
Let's assume we can solve the kernel scaling issue (or use faster GPU)

We are still left with: CUDA runtime overhead up to 25%!



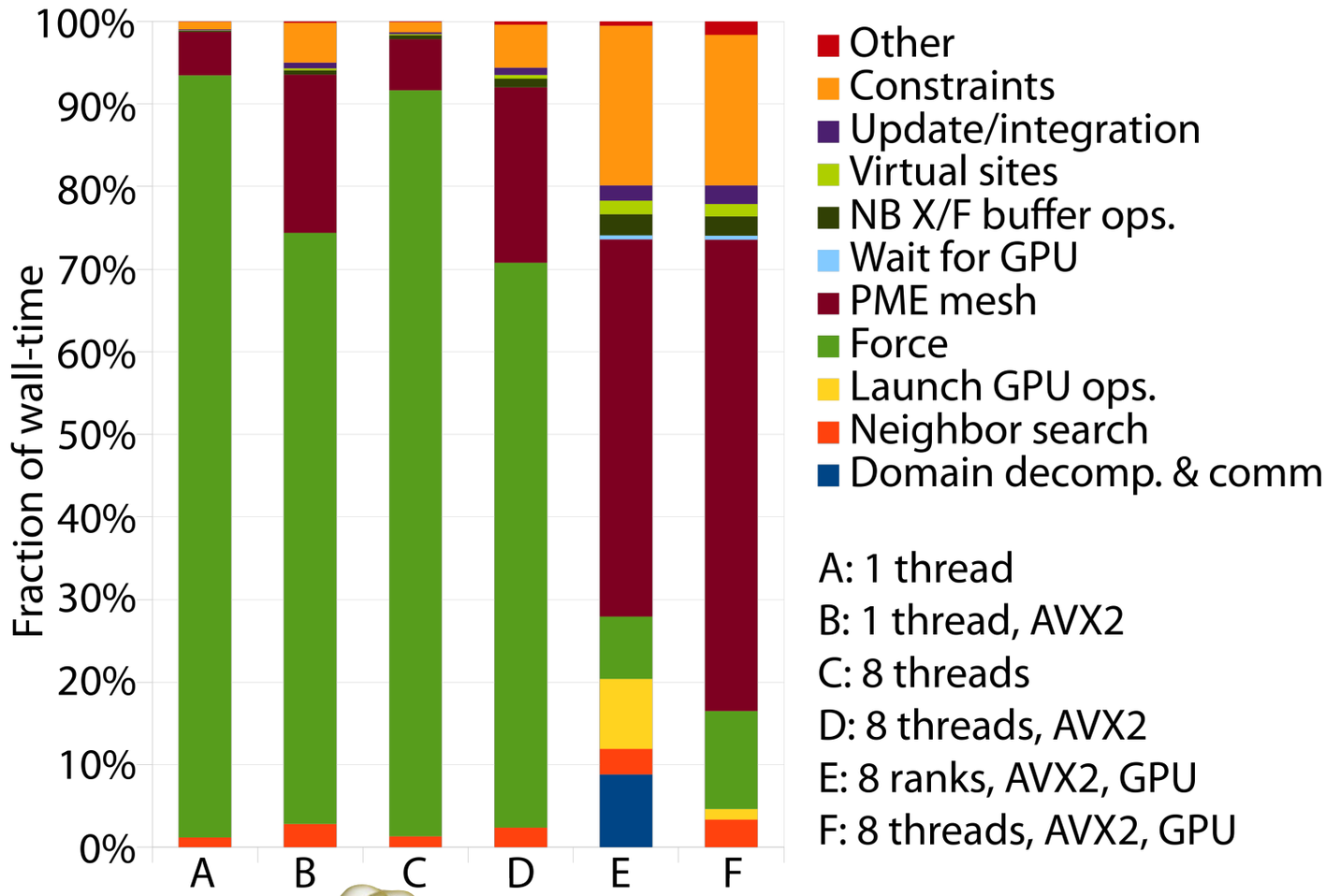
The Villin headpiece
~8,000 atoms
explicit solvent
triclinic box
PME electrostatics

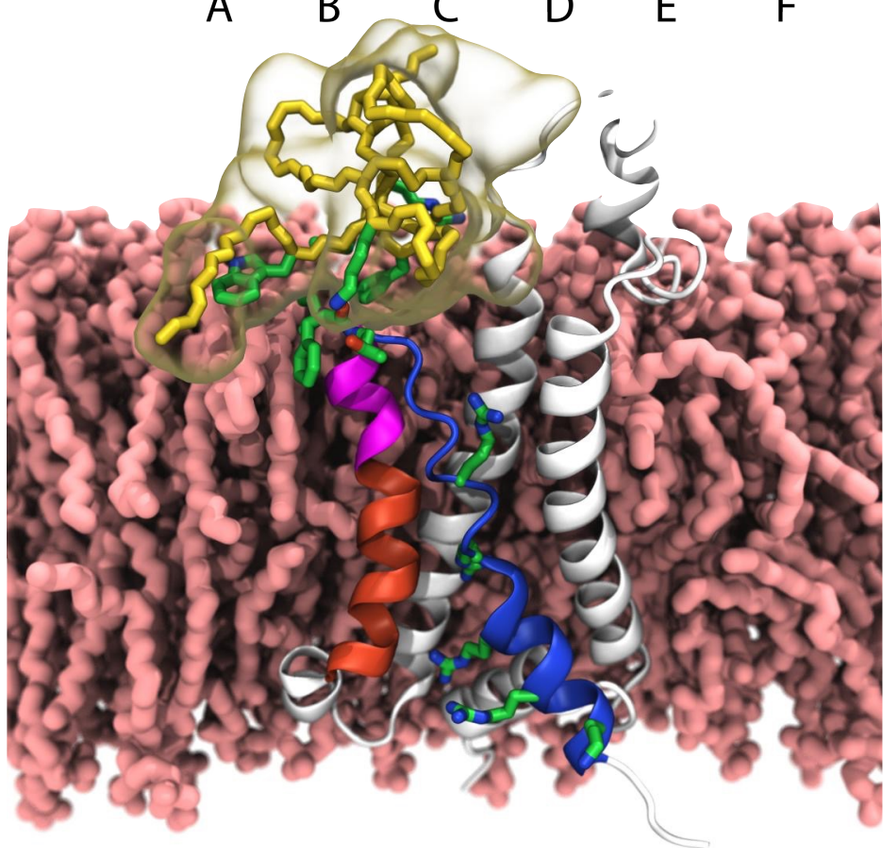




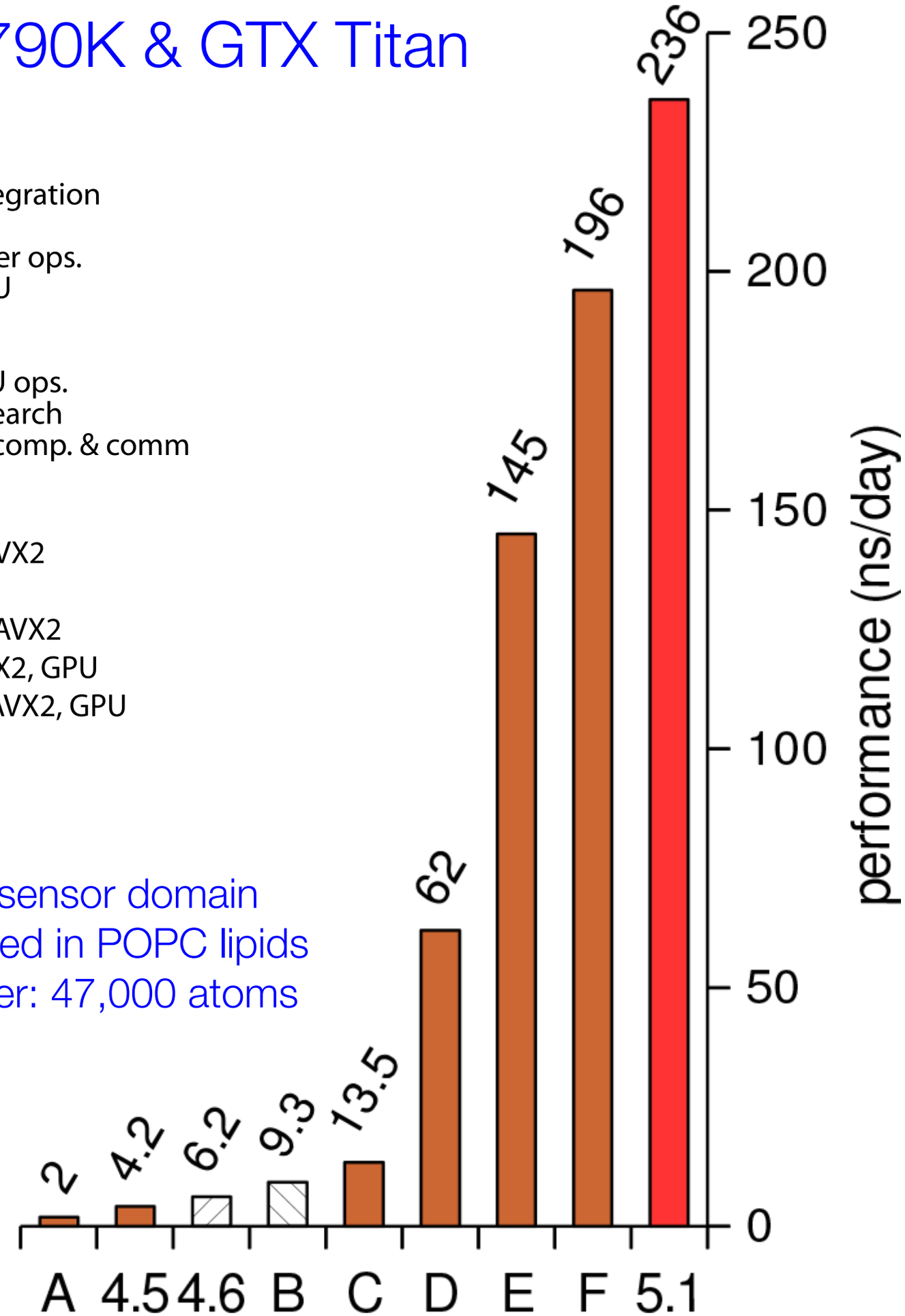
2,546 FPS (beat that, Battlefield)

Desktop example: Core i7 4790K & GTX Titan





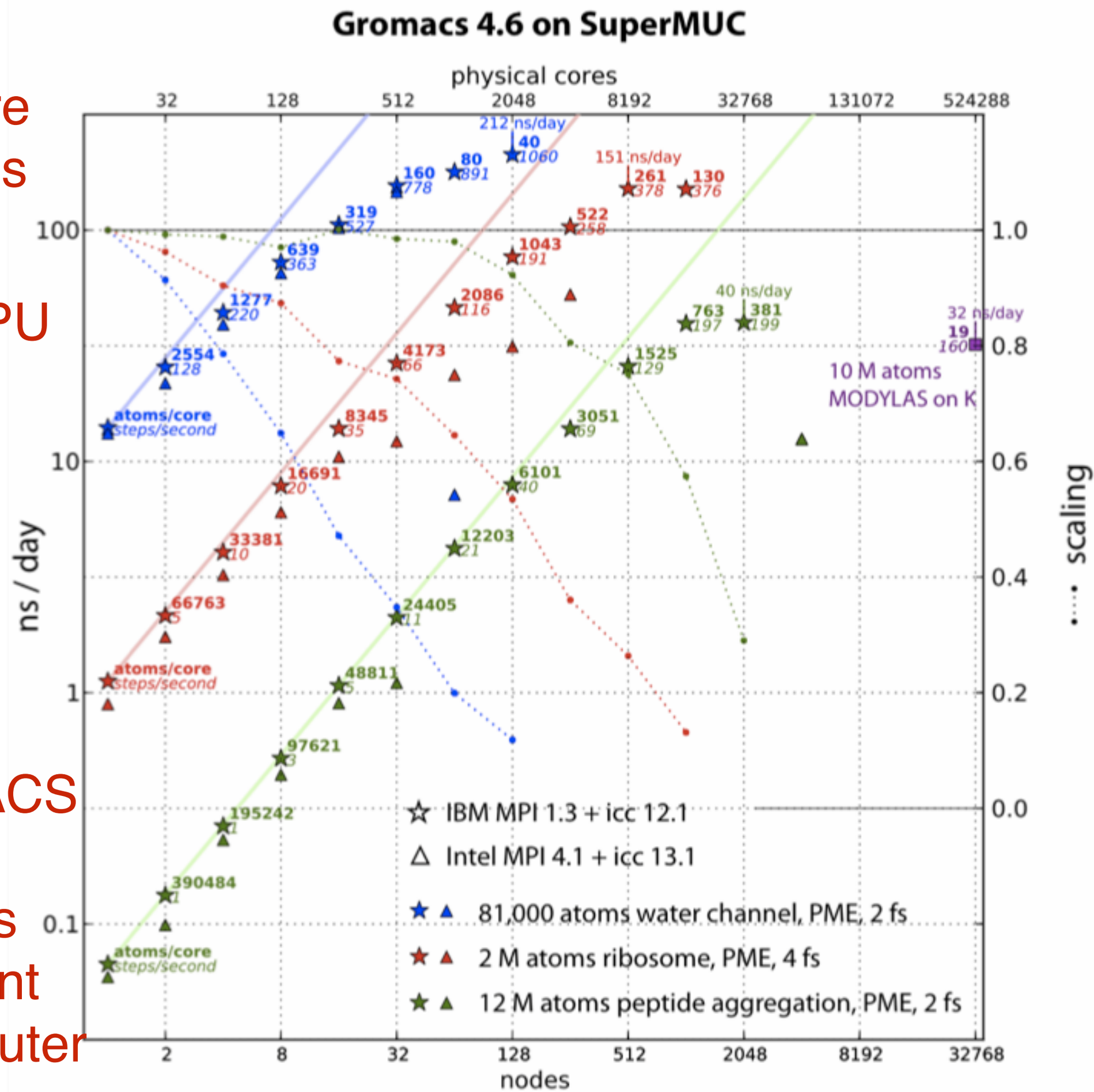
Voltage-sensor domain embedded in POPC lipids and water: 47,000 atoms



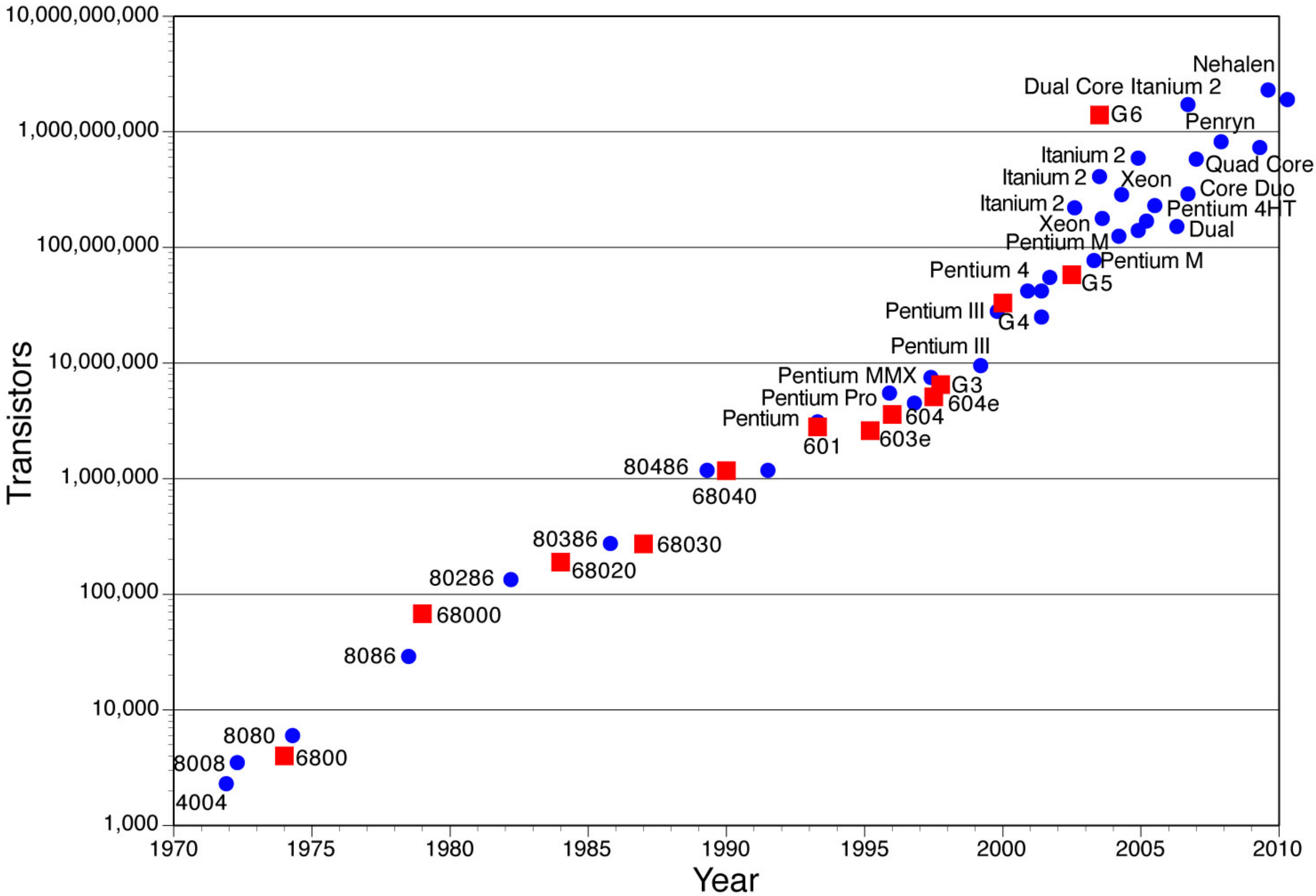
Strong scaling:
40-80 atoms/core
for small systems

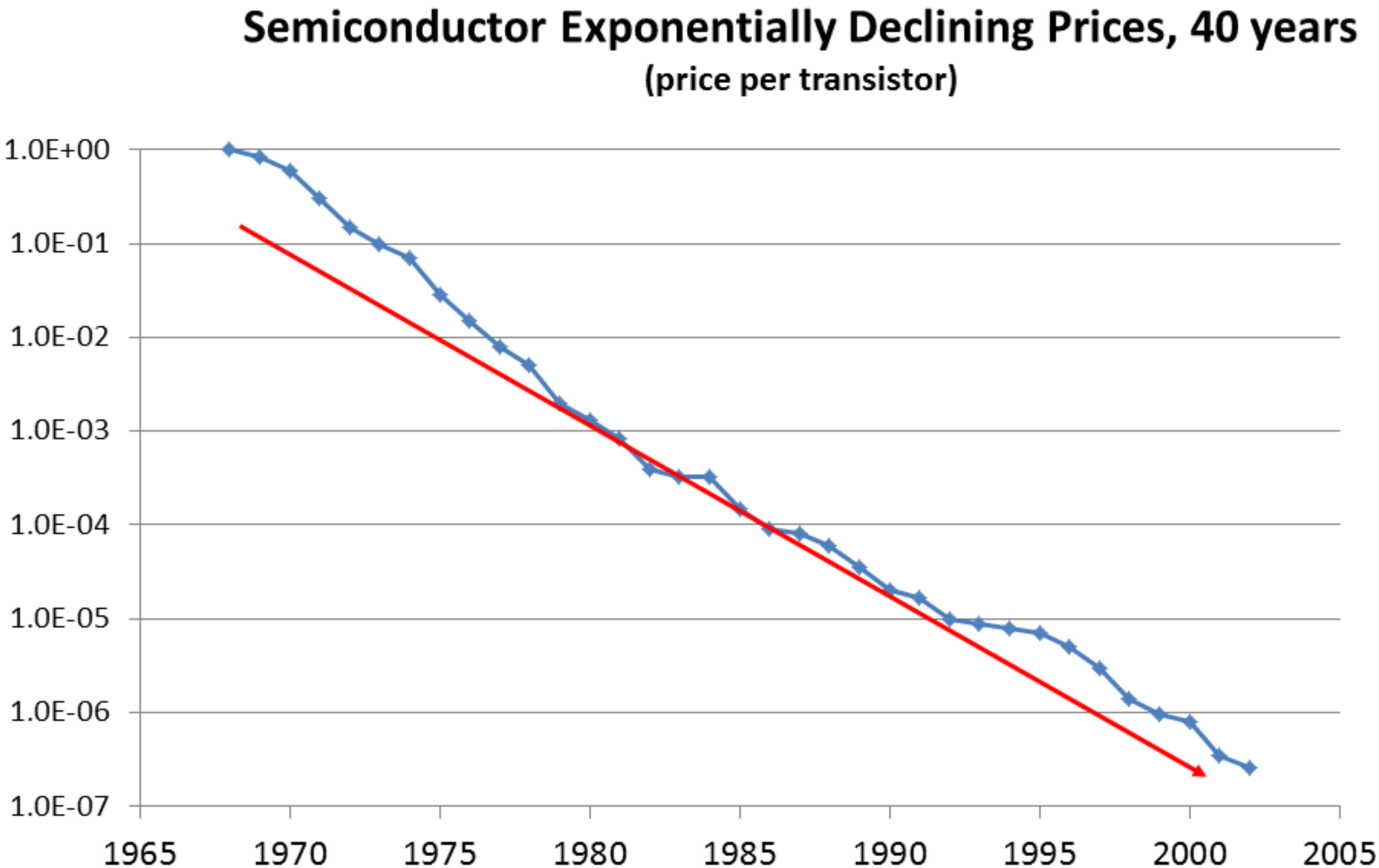
~1300 atoms/GPU

Performance:
12k Xeon cores
running GROMACS
on SuperMUC
beats 880k cores
running a different
code on K computer



You have already lost
the CPU game


10,000,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
1,000
Transistors
1970
1975
1980
1985
1990
1995
2000
2005
2010
Year
Nehalen
Dual Core Itanium 2
G6
Penryn
Itanium 2
Quad Core
Itanium 2
Xeon
Core Duo
Itanium 2
Pentium 4HT
Xeon
Dual
Pentium M
Pentium M
Pentium 4
G5
Pentium III
G4
Pentium III
Pentium MMX
G3
Pentium Pro
604e
Pentium
604
601
603e
80486
68040
80386
68030
68020
80286
68000
8086
8080
6800
8008
4004


Semiconductor Exponentially Declining Prices, 40 years
(price per transistor)
1.0E+00
1.0E-01
1.0E-02
1.0E-03
1.0E-04
1.0E-05
1.0E-06
1.0E-07
1965
1970
1975
1980
1985
1990
1995
2000
2005

No longer true: 14nm transition

- Technology limitations mean we get ~1.75x faster performance instead of ~2x
- Extremely difficult engineering:
  1.5x more expensive than 22nm
- We are suddenly looking at 16% improvement in bang-for-the-$ compared to 100% every 18M
- The question is not whether 5nm is technically feasible, but whether it is financially feasible
- There is a real risk it might stop at 10nm

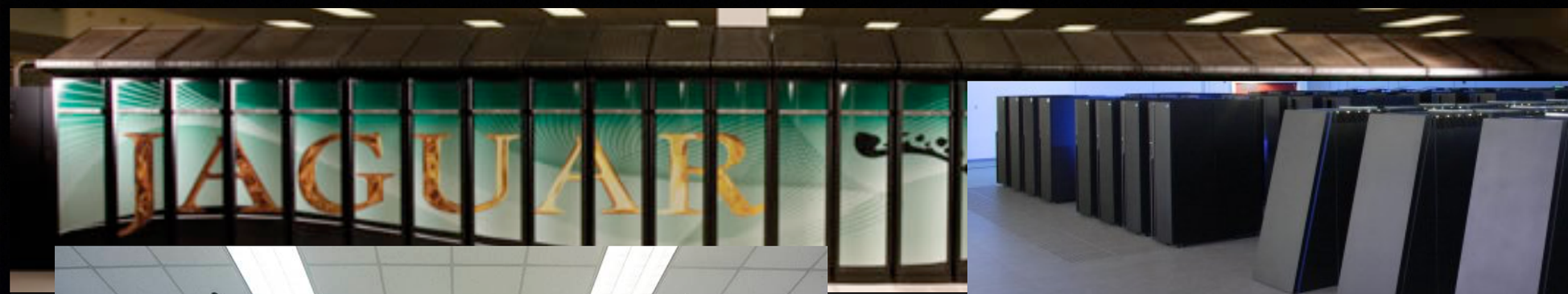
Memento mori.

The last vector computer (T90)
was shipped by Cray in 1999

Performance was 2-50 GFLOPS

Today, an iPhone 5s is 115.2 GFLOPS

If your code does not run on stream processors,
in 2025 it might be limited to the equivalent of
what less than a single iPhone core is today



~2024: 1B ‘cores’

2022: ~300M cores

2020: ~100M cores

2018: ~30M cores

2016: ~10M cores

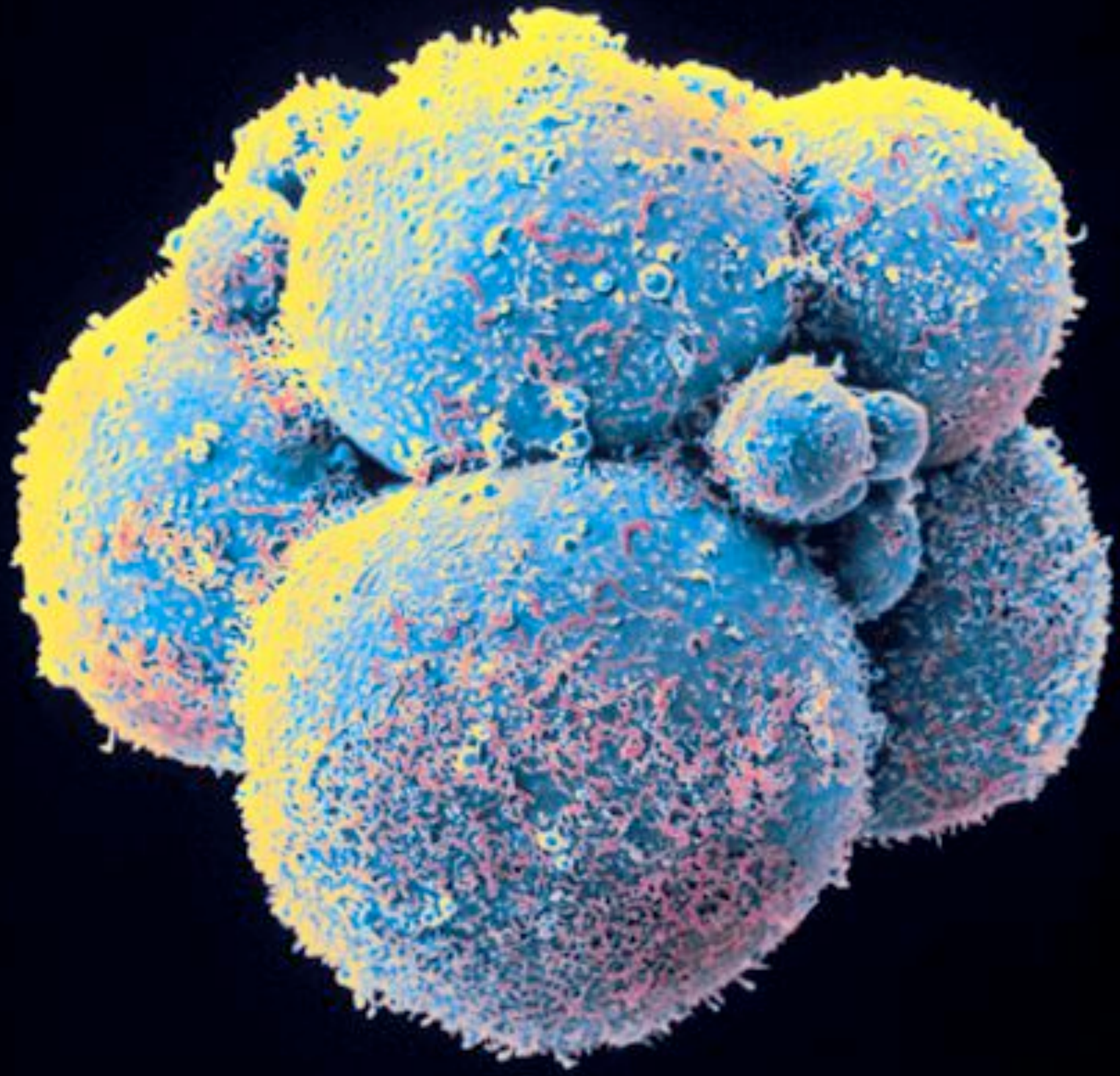
2014: ~3M cores

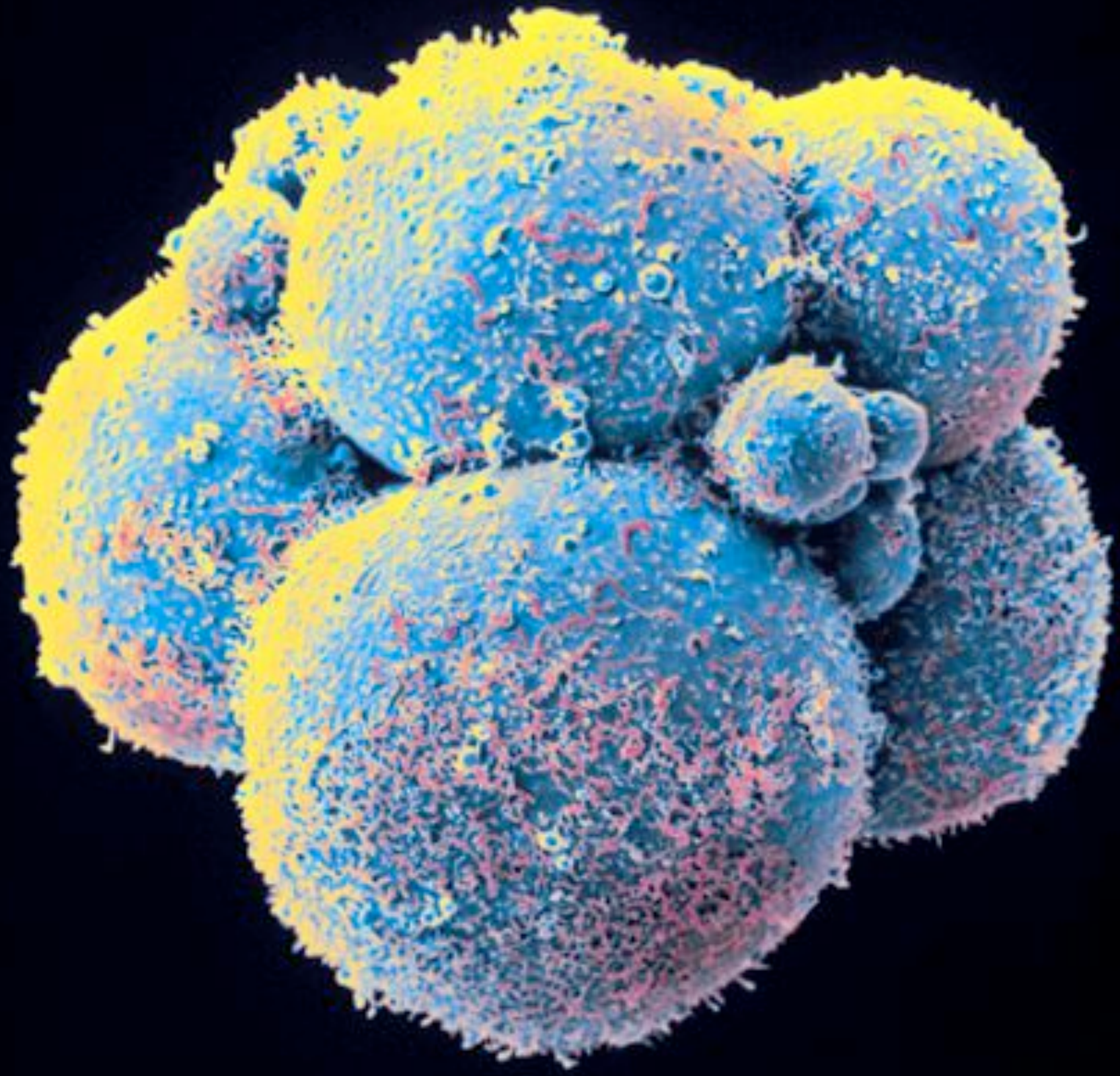
2012: ~1M cores

2010: ~300,000 cores

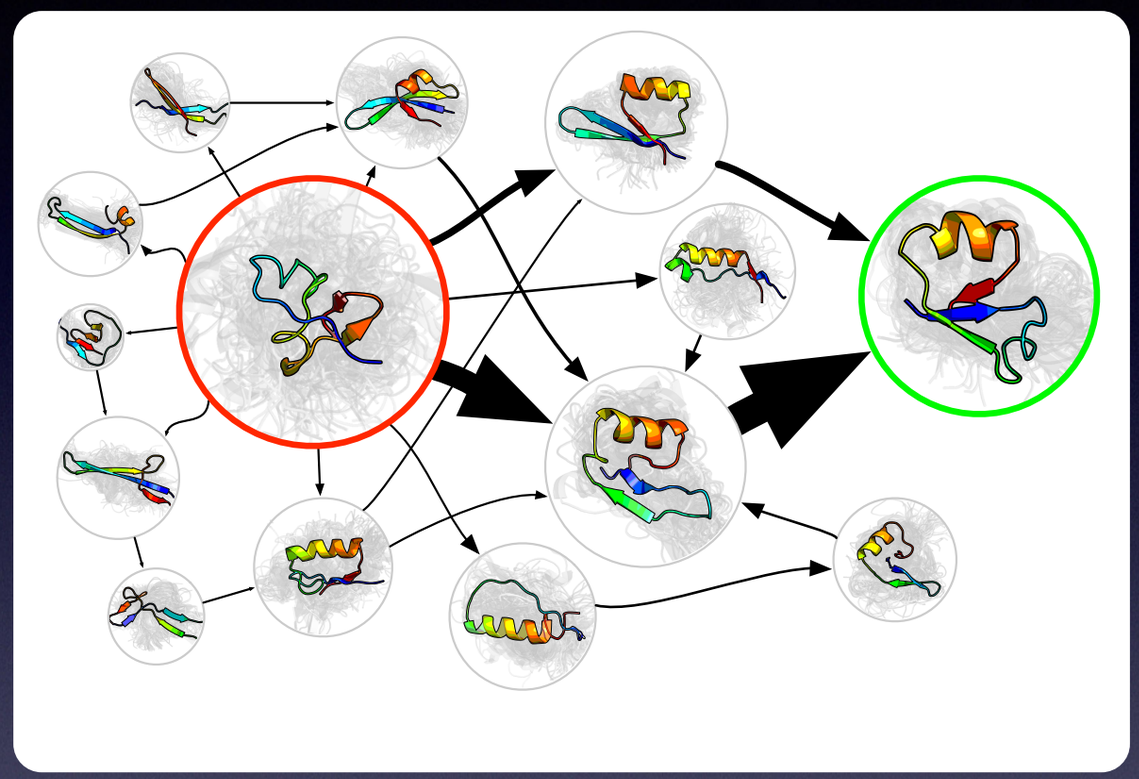
How will you use a billion cores?

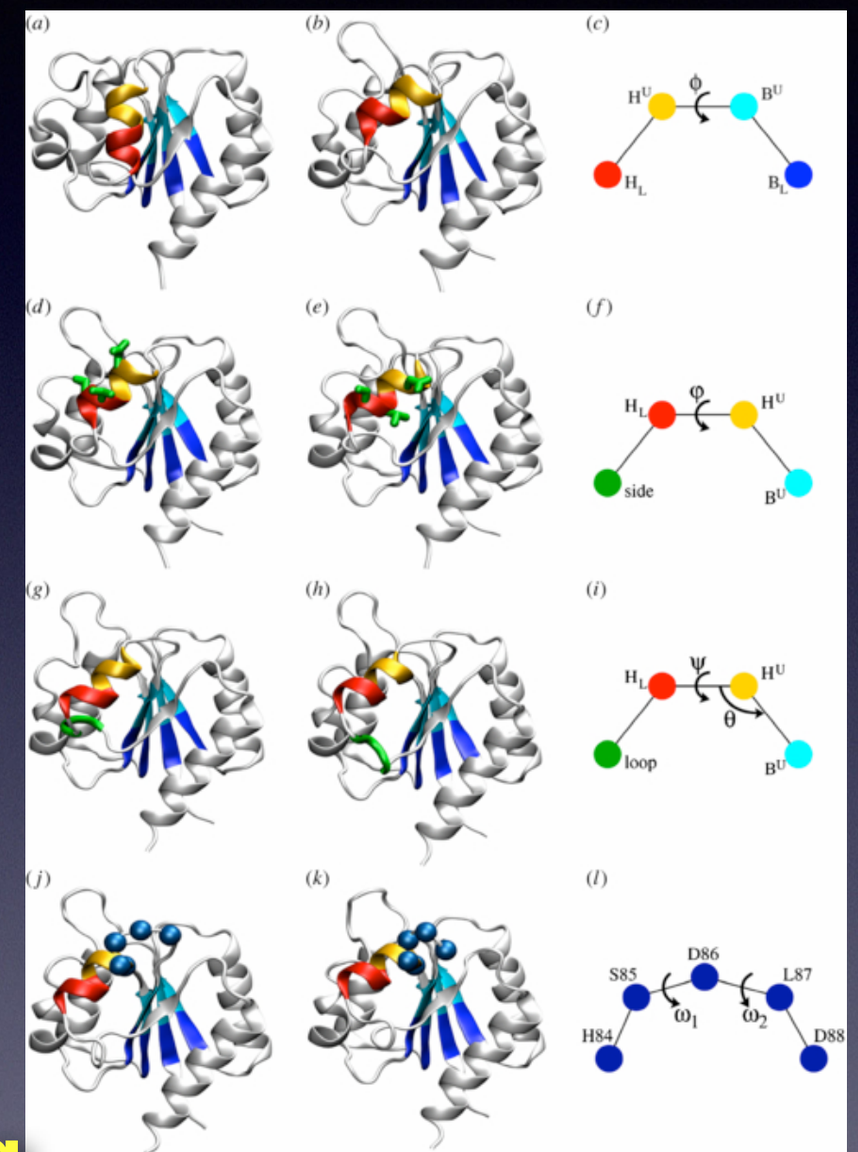
We keep scaling “up” (larger systems) where we should scale “down” (fine-grained parallelism, ensembles)!

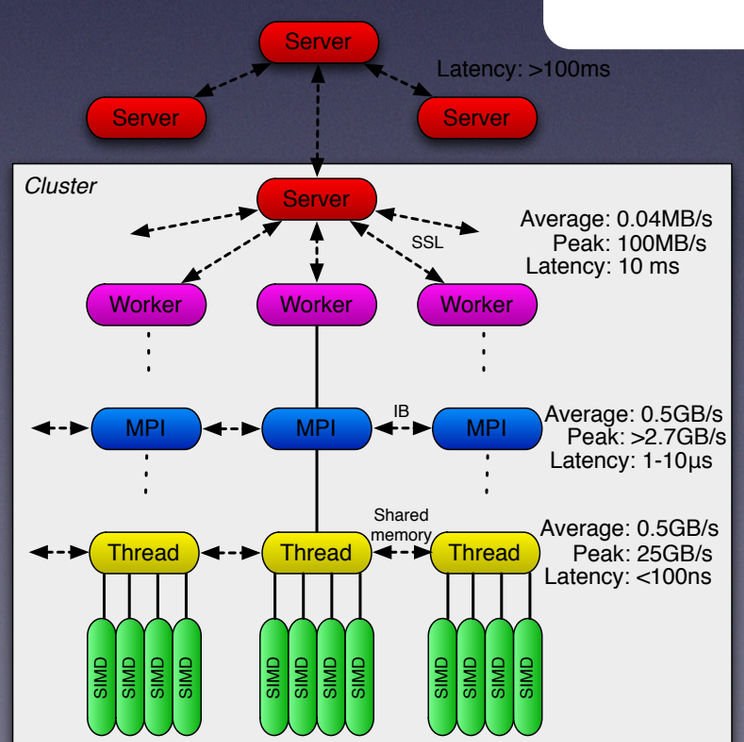




From ~100k cores to Exascale: Ensembles







Milestoning

Metadynamics

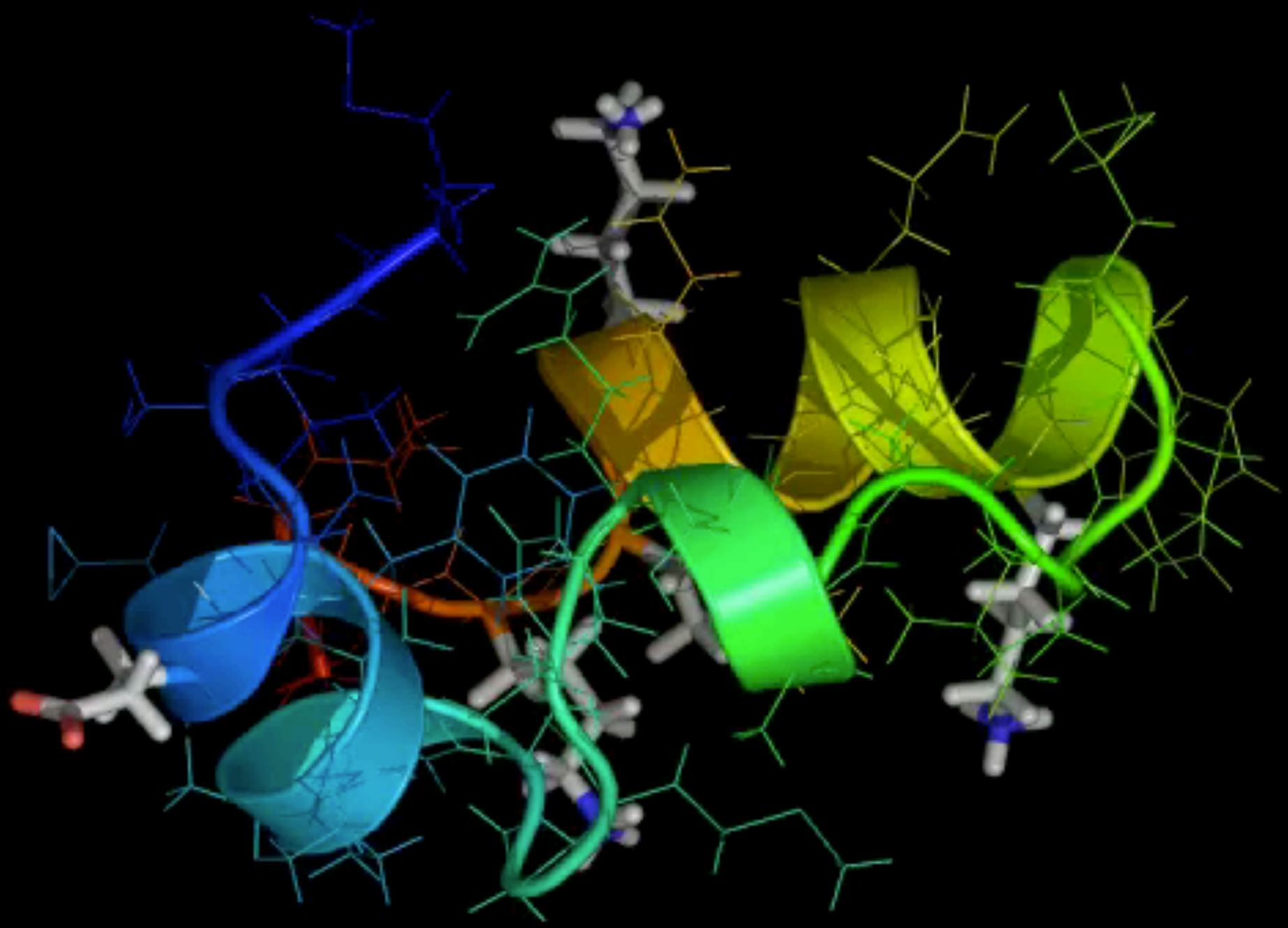
Markov State Models

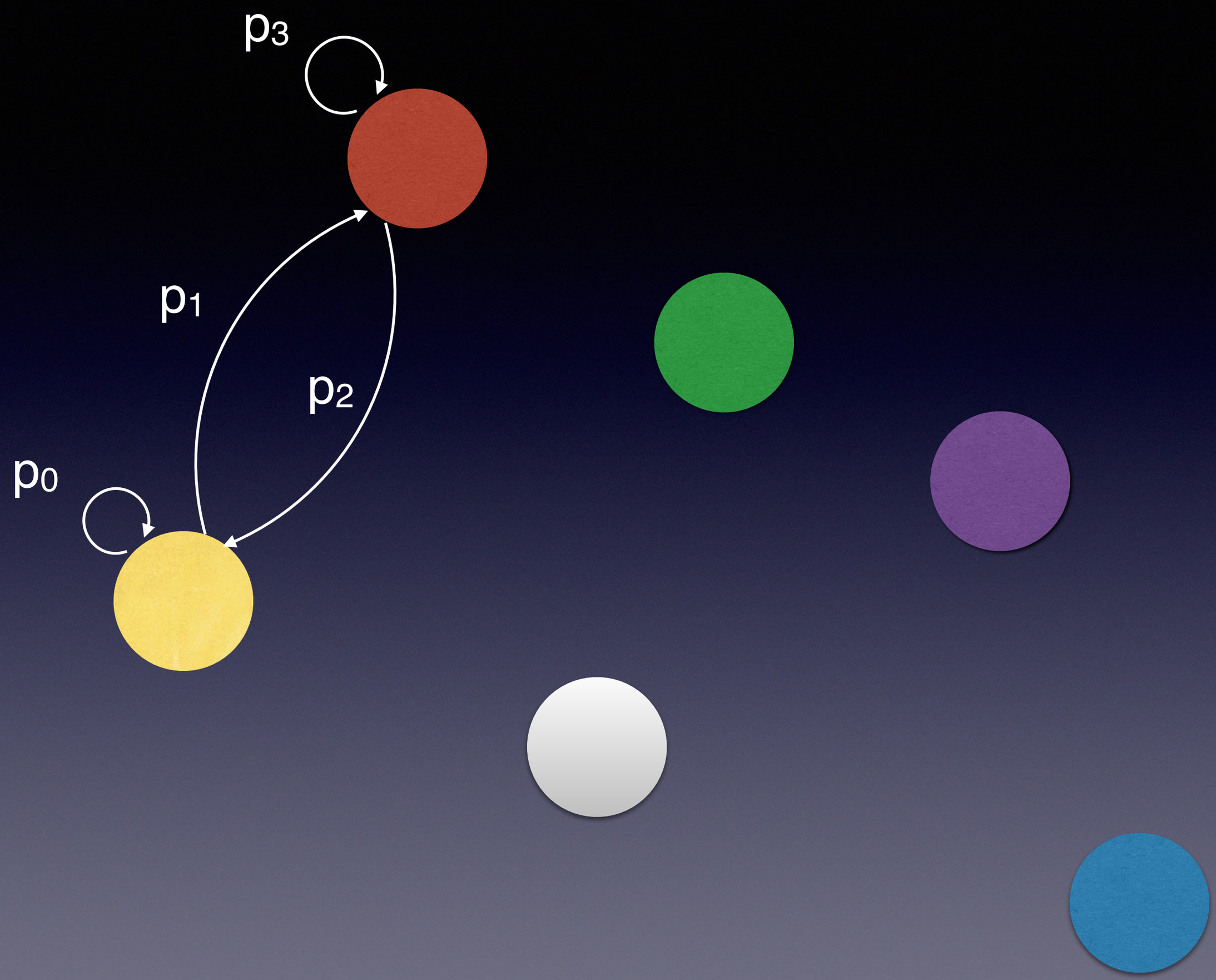
Monte Carlo Sampling

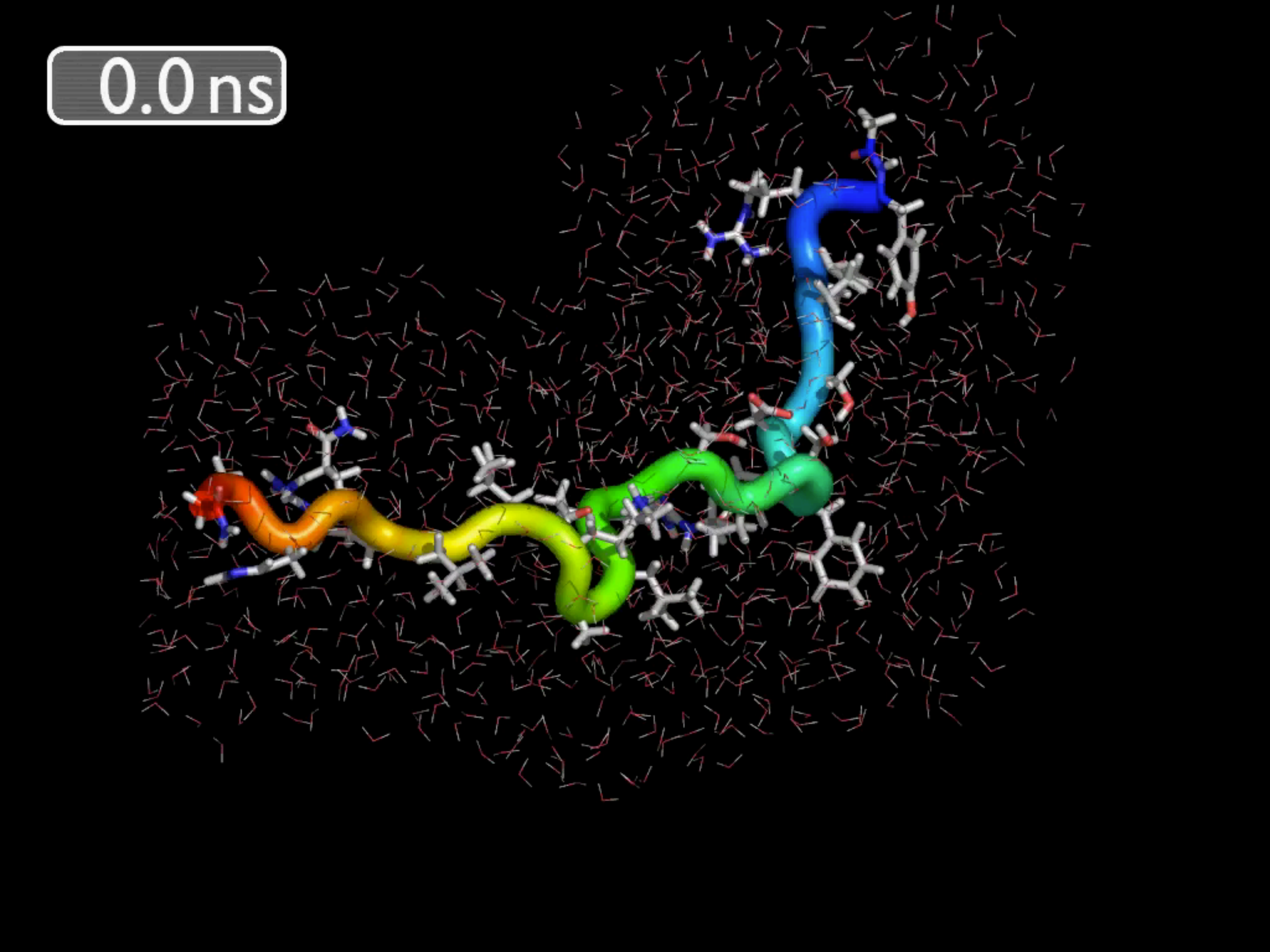
Free Energies

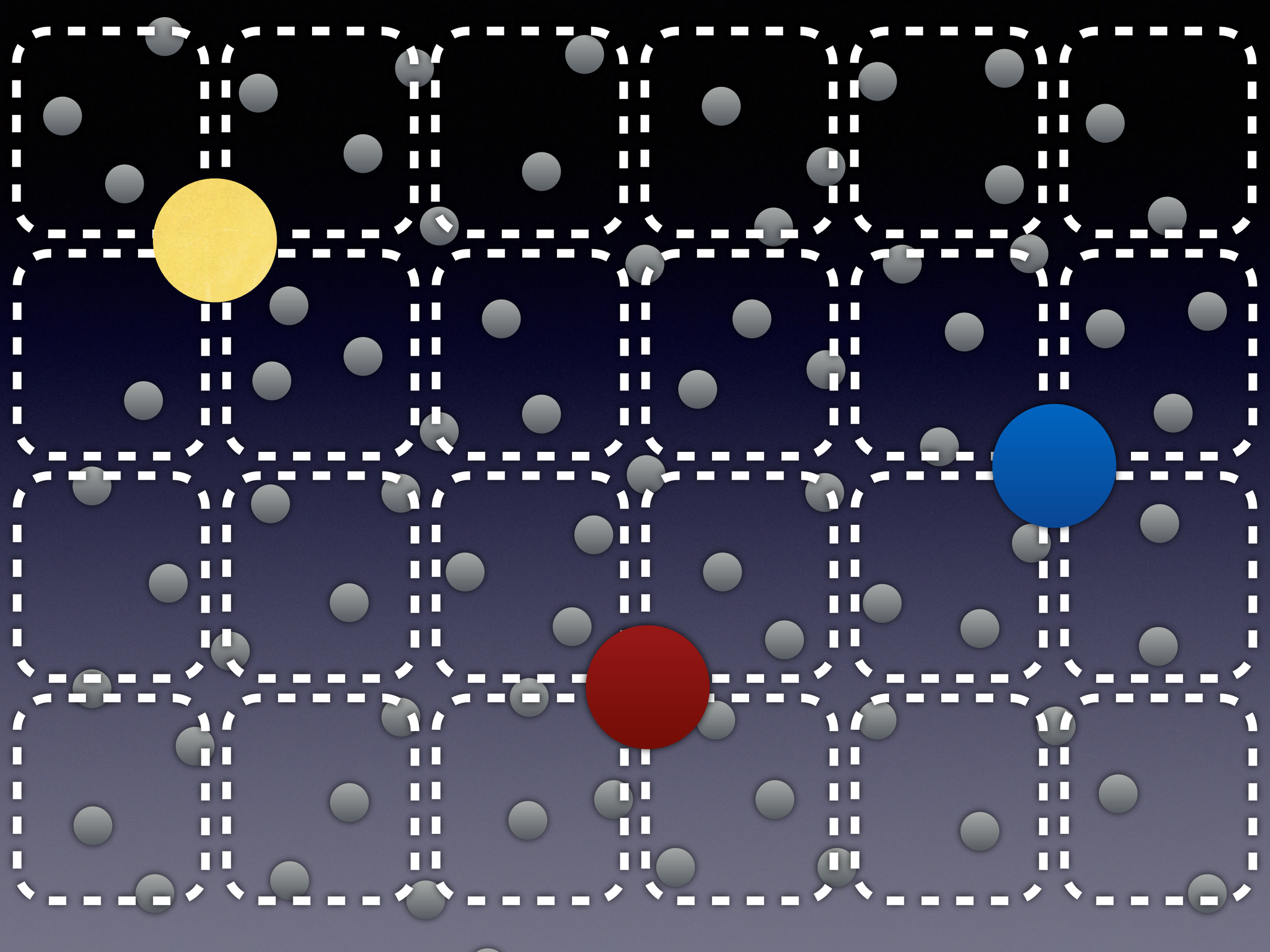
Swarms / Transition pathways

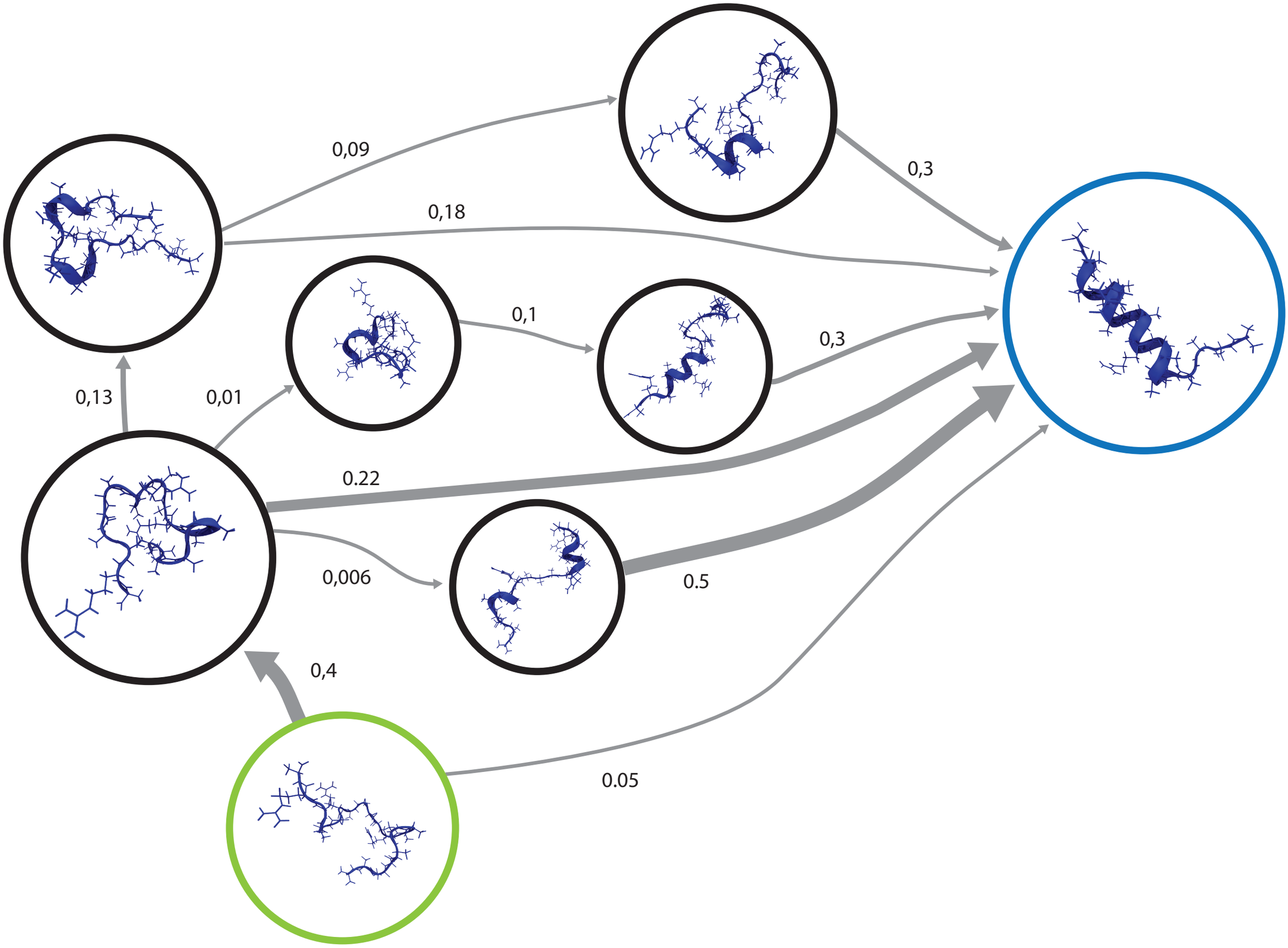
Markov State Models




p_3
p_1
p_2
p_0

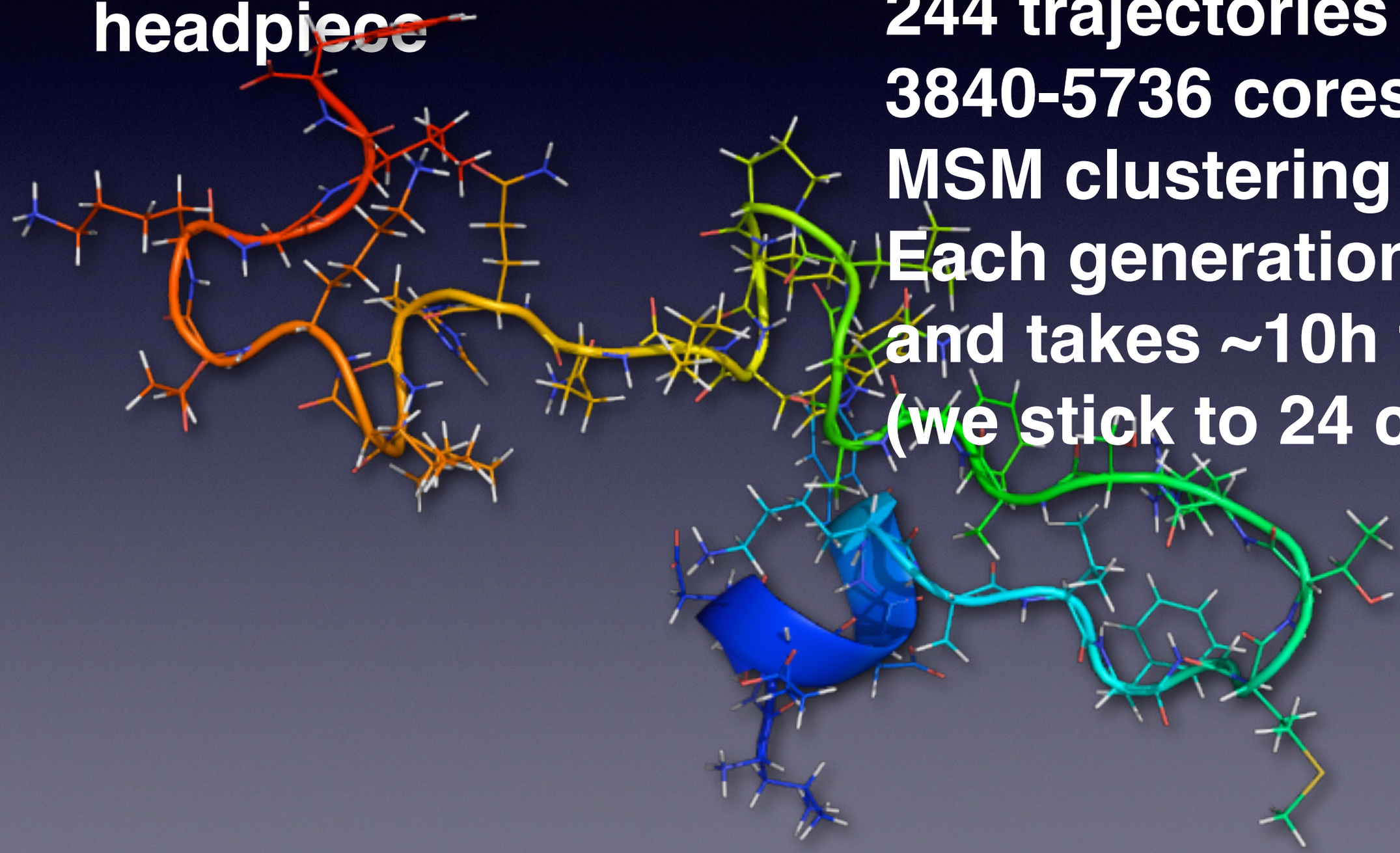

0.0 ns




0,09
0,3
0,18
0,1
0,3
0,13
0,01
0.22
0,006
0.5
0,4
0.05

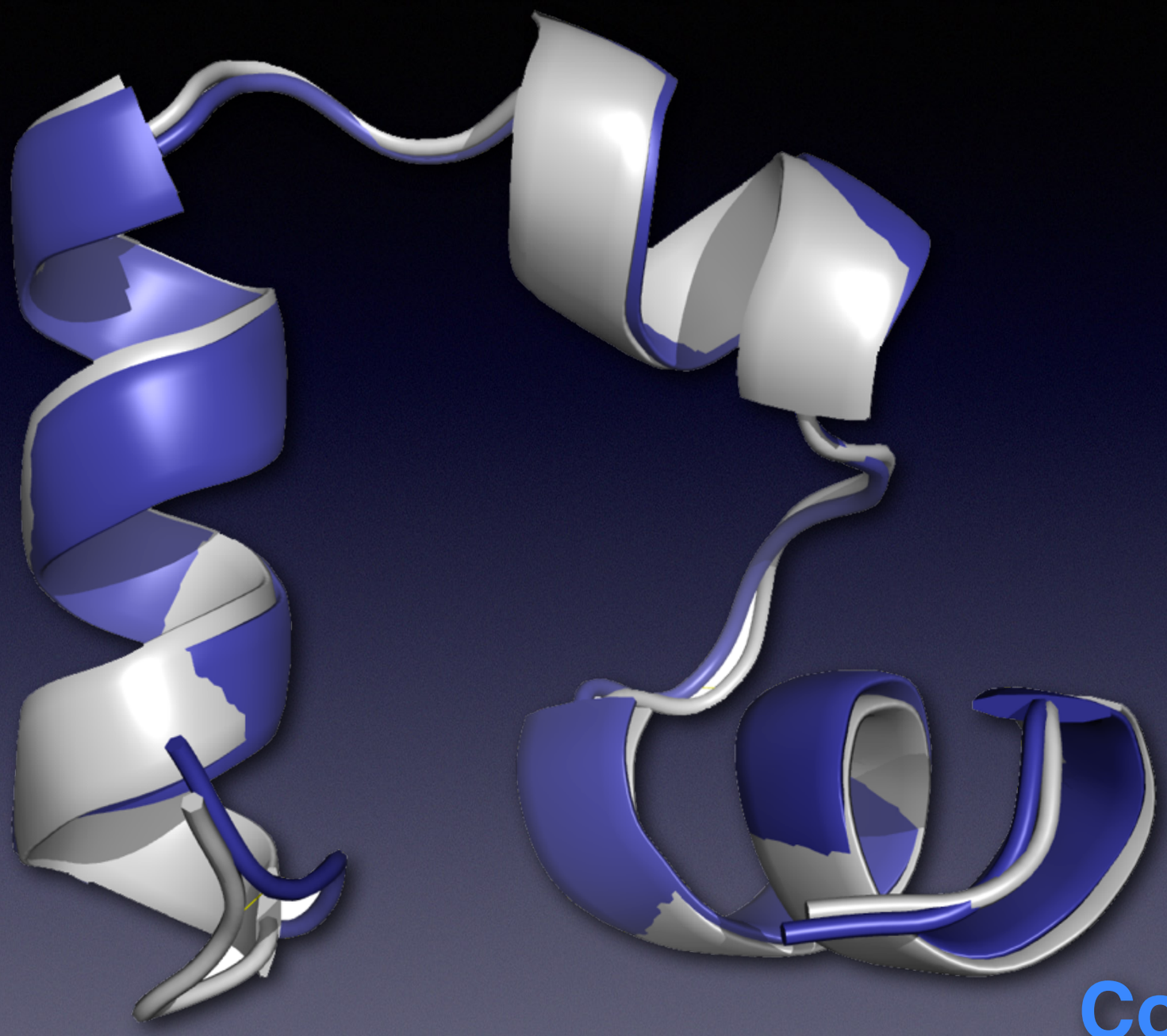
Ensembles in action

The villin headpiece



244 trajectories
3840-5736 cores used
MSM clustering
Each generation 50ns, and takes ~10h to run (we stick to 24 cores)

30 hours later



1.4 Å RMSD

Transition state matrix converges in 46h!

Compared to ASIC HW:
4-5X better throughput
2X more efficient sampling
10X total

Method Development:
Mark Abraham
Szilárd Páll
Berk Hess
Sander Pronk
Viveca Lindahl
Petter Johansson
Grant Rotskoff
Anders Gabrielsson
Christian Wennberg

Biophysics/ion channels:
Samuel Murail
Torben Brömstrup
Özge Yoluk
Iman Pouya
Jens Carlsson
Sophie Schwaiger
Göran Klement
Magnus Andersson

Stiftelsen för Strategisk Forskning
erc European Research Council
Sjunde Ramprogrammet
National Institutes of Health
PRACE
SeRC Swedish e-Science Research Centre
Vetenskapsrådet
bioexcel Center of Excellence for Computational Biomolecular Research