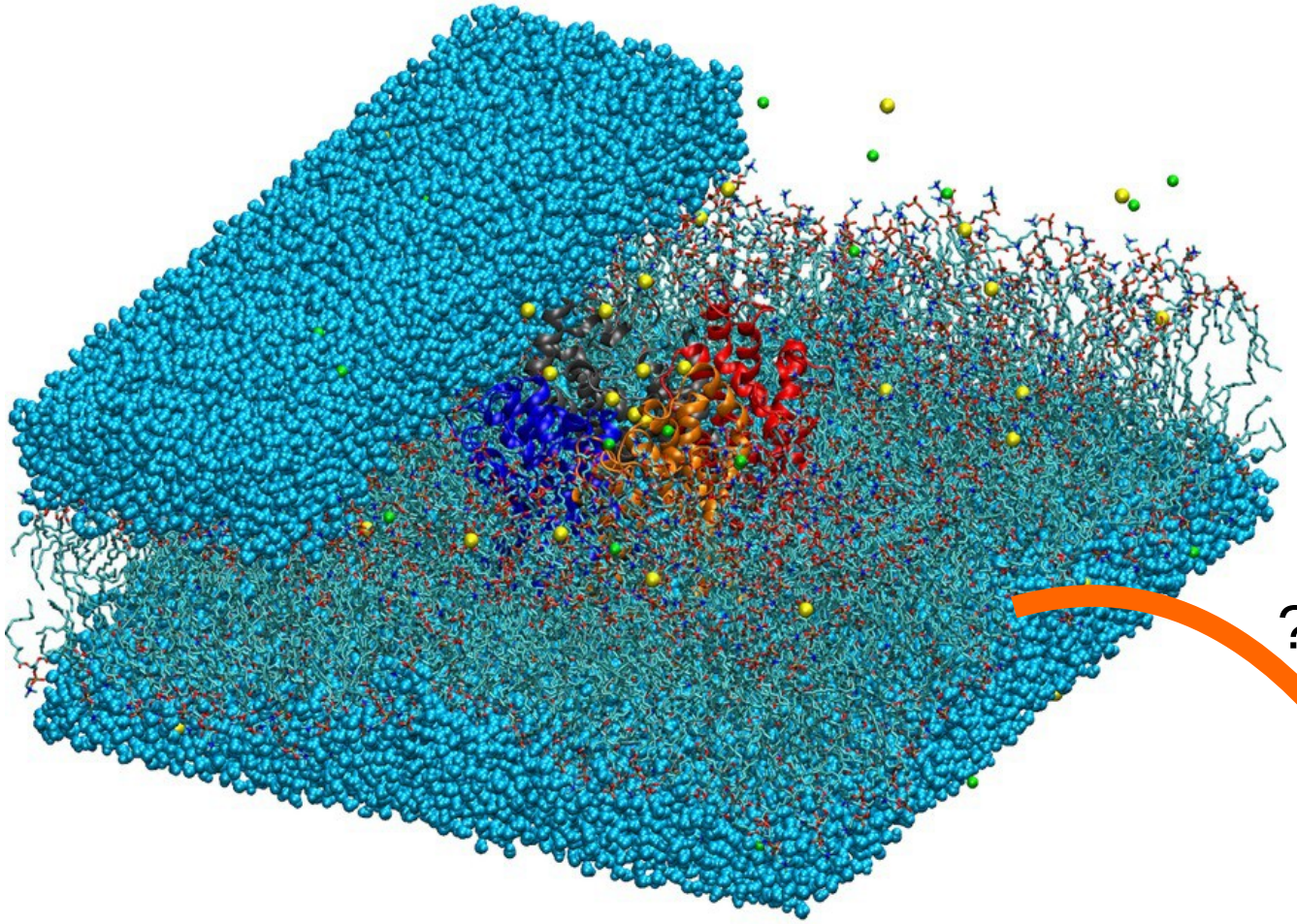
Designing, Implementing and Optimizing Collective Variables in VMD and NAMD

Jérôme Hénin

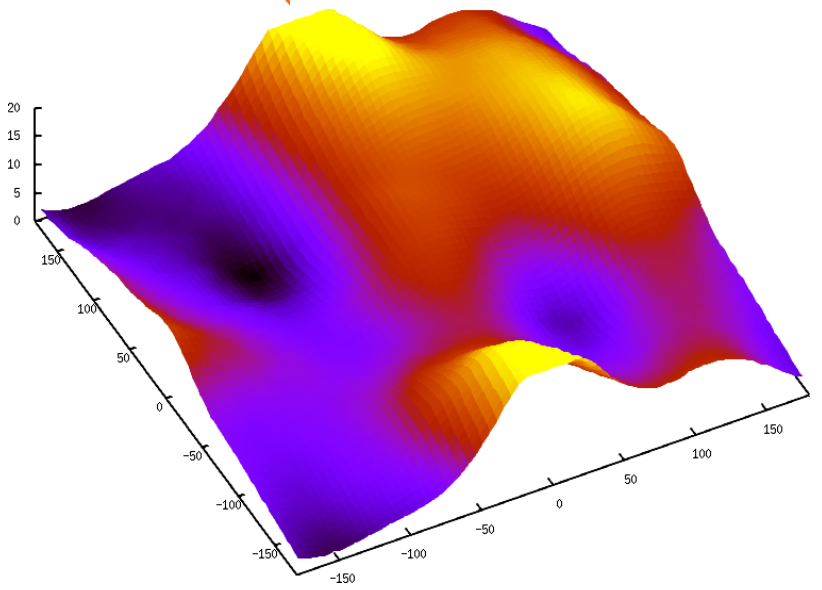
Enhanced Sampling
and Free-Energy Calculations
Urbana, 12 September 2018



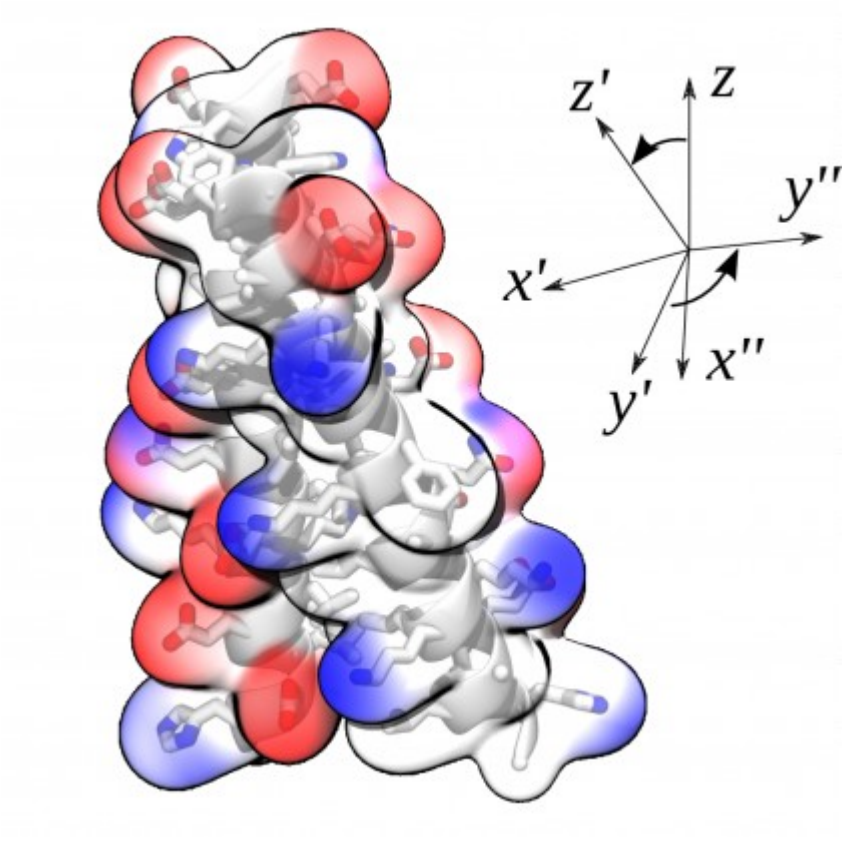
How to:

- define colvars?
- analyze colvars?
- choose colvars?
- bias colvars?

??



Collective Variables Module



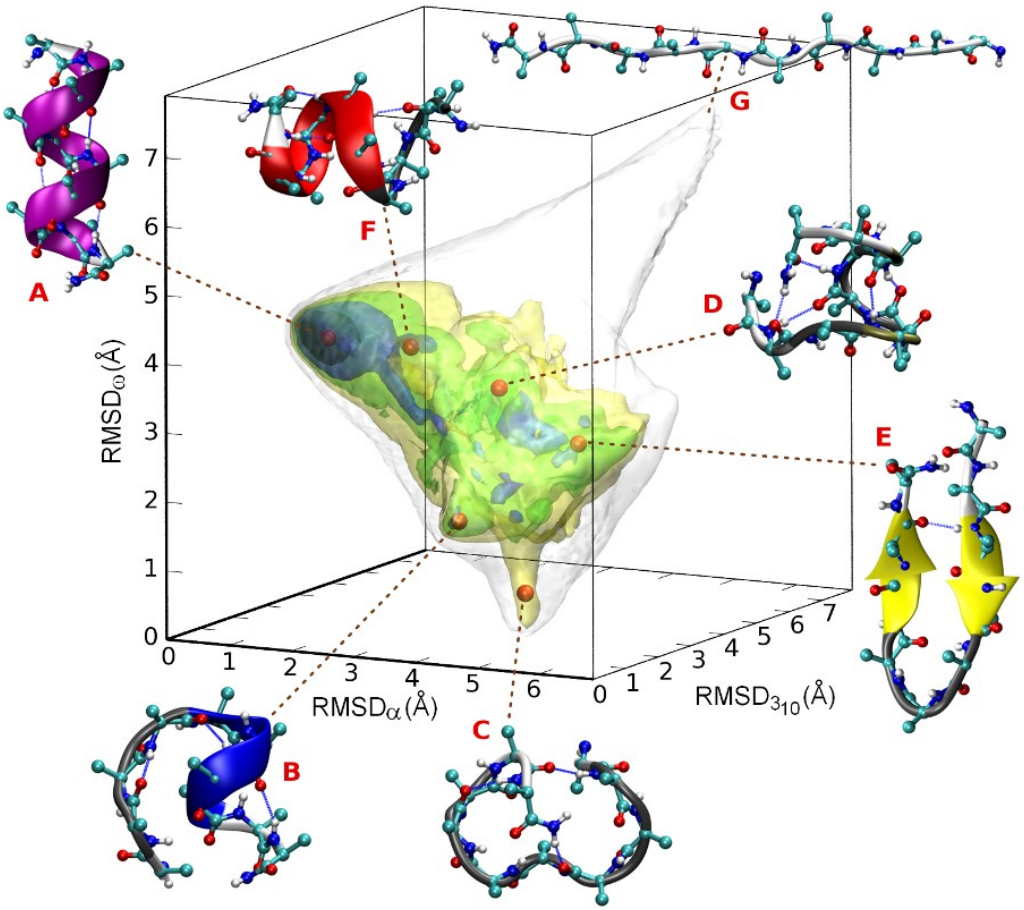
Giacomo Fiorin
Temple University

Versatile biases in generalized coordinates

- arbitrary dimension
- run-time combination of variables
- many variables available
- time-dependent biases
- → adaptive free energy methods (metadynamics, ABF)

- included in **NAMD**, **LAMMPS**, **VMD**

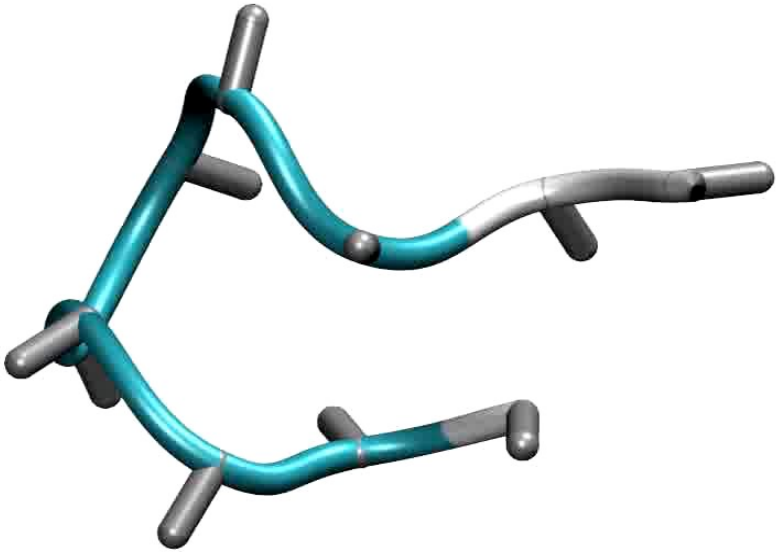
- extensible through **C++** or **Tcl**



Example Targeted MD

colvar simulation = collective variable + bias

```
colvar {
  name RMSD

  rmsd {
    atoms {
      atomsFile  beta.pdb
      atomsCol   O
    }
    refPositionsFile beta.pdb
  }
}


harmonic {
  colvars RMSD

  centers 5.3
  targetCenters 0.0
  targetNumSteps 200000
  forceConstant 100.
}
```



Getting Colvars 1: with precompiled binaries

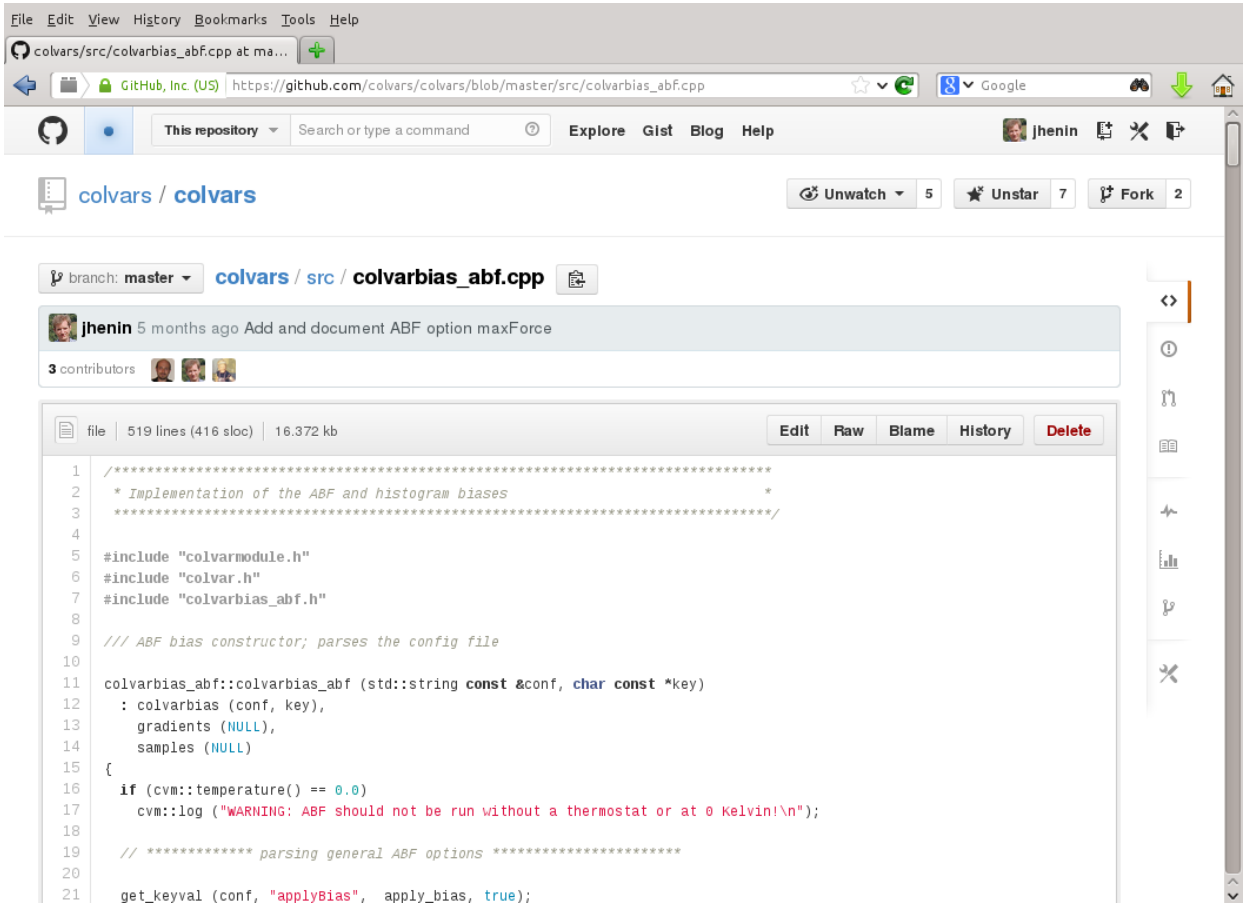
- recent versions of NAMD and VMD include Colvars

1) Download NAMD or VMD binaries
2) use Colvars
3) …
4) Profit!

Getting Colvars 2: source repository

- public repository on GitHub http://github.com/Colvars/colvars
- always up-to-date code
- documentation
- automated regression tests

- issue tracker
- we rely on user feedback



Where can I find documentation?

Information 1: reference manuals

- Colvars chapter in the NAMD and VMD user guides
- available as online HTML at http://colvars.github.io
- details in reference publication (Open Access)

Molecular Physics, 2013
Vol. 111, Nos. 22–23, 3345–3362, http://dx.doi.org/10.1080/00268976.2013.813594

INVITED ARTICLE

Using collective variables to drive molecular dynamics simulations

Giacomo Fiorin[a,*], Michael L. Klein[a] and Jérôme Hénin[b]

[a]*Department of Chemistry and Institute for Computational Molecular Science, Temple University, Philadelphia, PA, USA;* [b]*Laboratoire de Biochimie Théorique, Institut de Biologie Physico-Chimique, CNRS, Paris, France*

(*Received 24 April 2013; final version received 4 June 2013*)

A software framework is introduced that facilitates the application of biasing algorithms to collective variables of the type commonly employed to drive massively parallel molecular dynamics (MD) simulations. The modular framework that is presented enables one to combine existing collective variables into new ones, and combine any chosen collective variable with available biasing methods. The latter include the classic time-dependent biases referred to as steered MD and targeted MD, the temperature-accelerated MD algorithm, as well as the adaptive free-energy biases called metadynamics and adaptive biasing force. The present modular software is extensible, and portable between commonly used MD simulation engines.

Keywords: molecular dynamics simulation; collective variable; free-energy calculation; adaptive bias; sampling

Information 2: online examples

- Simple, runnable examples:

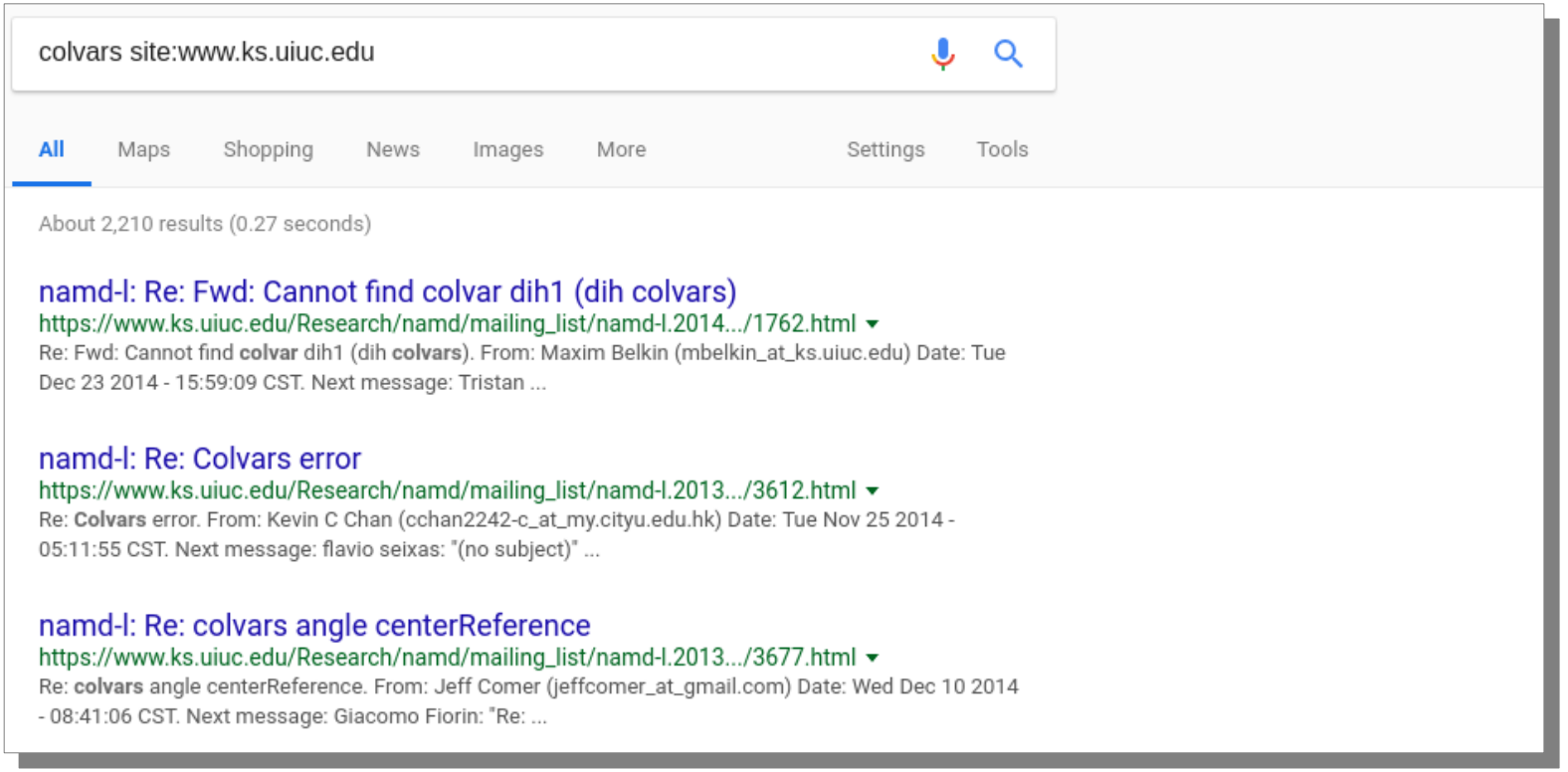
  http://github.com/Colvars/examples

- Various advanced examples used in Colvars publication

  http://github.com/Colvars/colvars/tree/master/examples

Information 3: mailing lists

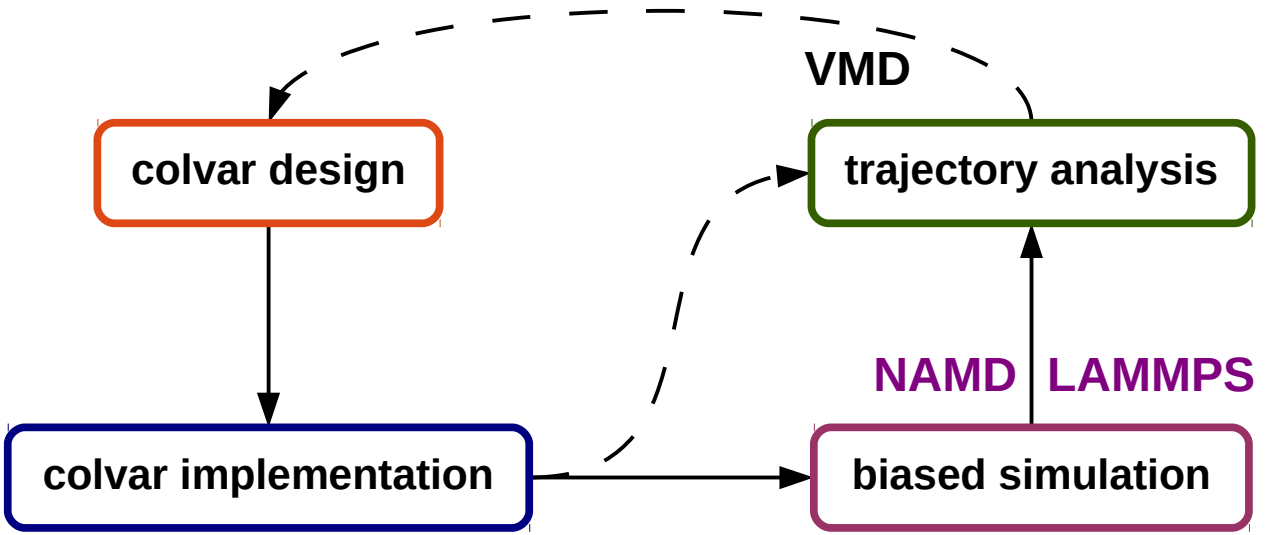
- NAMD and VMD mailing lists offer **searchable archives**



- if the answer is not there, send a precise question to the **most relevant mailing list** (namd-l or vmd-l)

A tour of Colvars features

Collective variable modeling workflow



basis functions
(aka *components)*

$$\xi(x) = \zeta(x)$$

polynomials

$$\xi(x) = \sum_{i=1}^{n} c_i \zeta_i(x)^{p_i}$$

custom functions

$$\xi(x) = f_{\text{user}}(\zeta_1(x), \ldots, \zeta_n(x))$$

- classic algorithms
- adaptive sampling
- multiple-walker sampling
- scripted biases

Components (basis functions)

distances	distance	center-of-mass distance between two groups
	distanceZ	projection of a distance vector on an axis
	distanceXY	modulus of the projection of a distance vector on a plane
	distanceVec	distance vector between two groups
	distanceDir	distance unit vector between two groups
	distanceInv	mean distance between two groups of atoms
raw data	distancePairs	set of pairwise distances between two groups
	cartesian	vector of atomic Cartesian coordinates
angles	angle	angle between three groups
	dipoleAngle	angle between two groups and dipole of a third group
	dihedral	torsional angle between four groups
	polarTheta	polar angle in spherical coordinates
	polarPhi	azimuthal angle in spherical coordinates
coordination	coordNum	coordination number between two groups
	selfCoordNum	coordination number between atoms within a group
	hBond	hydrogen bond between two atoms
collective	rmsd	root mean square displacement (RMSD) from reference positions
	rmsd-based	path collective variables
	eigenvector	projection of the atomic coordinates on a vector
	gyration	radius of gyration of a group of atoms
	inertia	total moment of inertia of a group of atoms
	inertiaZ	total moment of inertia of a group of atoms around a chosen axis
orientation	orientation	orientation from reference coordinates
	orientationAngle	angle of rotation from reference coordinates
	orientationProj	cosine of the angle of rotation from reference coordinates
	spinAngle	angle of rotation around a given axis
	tilt	cosine of the rotation orthogonal to a given axis
protein structure	alpha	α-helix content of a protein segment
	dihedralPC	protein dihedral principal component

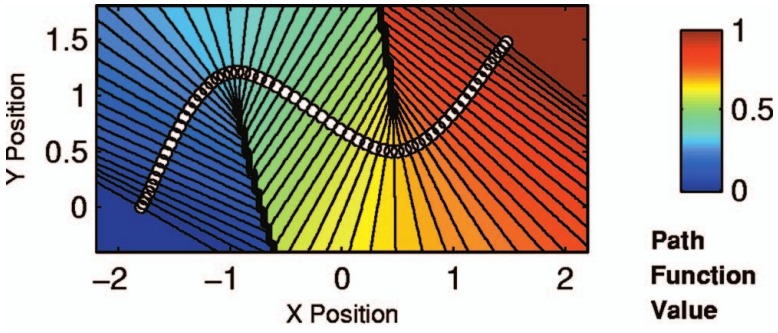
Write your own colvar (WYOC):
Custom functions with the Lepton library

Combine existing basis functions with custom expressions

```
colvar {
  name myVariable

  # This is a 2-vector function of a 4-vector
  customFunction x - r1
  customFunction cos(x) + r1 + r2 + r3

  distance {
    name x
    group1 { atomNumbers 4 }
    group2 { atomNumbers 99 }
  }
  distanceVec {
    name r
    group1 { atomNumbers 50 }
    group2 { atomNumbers 60 }
  }
}

harmonic {
  colvars myVariable
  centers (20, -5)
  forceConstant 100
}
```

Scripted function: path collective variables

```
colvar {
   name   s
   rmsd {
       atoms { atomNumbers { 10 20 30 } }
       refpositionsfile string-1.pdb ;# coordinates of the first bead
       componentExp     1            ;# index of the first bead
   }
   rmsd {
       atoms { atomNumbers { 10 20 30 } }
       refpositionsfile string-2.pdb ;# coordinates of the second bead
       componentExp     2            ;# index of the second bead
   }
   scriptedFunction pathCV
}
```

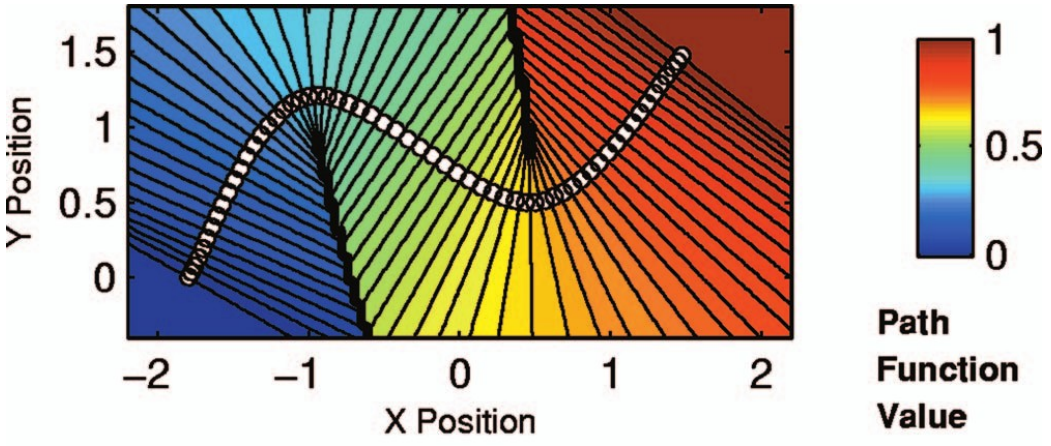


$$s(\mathbf{R}) = \frac{1}{P-1}\frac{\sum_{i=1}^{P}(i-1)e^{-\lambda(\mathbf{R}-\mathbf{R}(i))^2}}{\sum_{i=1}^{P}e^{-\lambda(\mathbf{R}-\mathbf{R}(i))^2}}$$

Branduardi et al. JCP 2007

```
proc calc_pathCV { args } {
    global pathCVlambda; global pathCVu; global pathCVv
    set N [llength $args]
    set i 0; set u 0.0; set v 0.0
    foreach x $args {
        set u [expr {$u + $i * exp(-$lambda * $x * $x)}]
        set v [expr {$v + exp(-$lambda * $x * $x)}]
        incr i
    }
    return [expr {1.0 / ($N - 1.0) * $u / $v}]
}

proc calc_pathCV_gradient { args } {
    global pathCVlambda; global pathCVu; global pathCVv
    set N [llength $args]
    set grad {} ; set i 0
    foreach x $args {
        set uprime [expr {-2.0* $i * $lambda * $x * exp(-$lambda*$x*$x)}]
        set vprime [expr {-2.0 * $lambda * $x * exp(-$lambda*$x*$x)}]
        incr i
        lappend grad [expr {1.0/($N-1.0)*($uprime*$v - $vprime*$u)/($v*$v)}]
    }
    return $grad
}
```

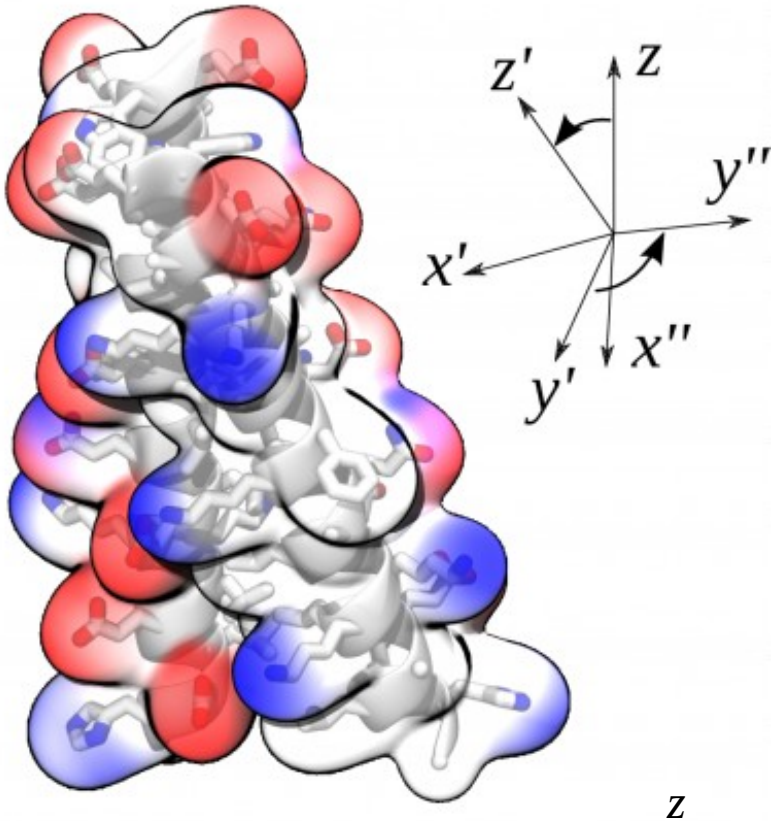
Optimization: dynamic colvar components



$$s(\mathbf{R}) = \frac{1}{P-1} \frac{\sum_{i=1}^{P} (i-1) e^{-\lambda(\mathbf{R}-\mathbf{R}(i))^2}}{\sum_{i=1}^{P} e^{-\lambda(\mathbf{R}-\mathbf{R}(i))^2}}$$

- path collective variables
  - depends on RMSD from all images on a discrete path – **expensive**
  - dominated by a few terms nearby images
- Colvars implementation is a Tcl-scripted coordinate
  - each RMSD is a colvar component (cvc)
- scripting command `cvcflags` is used to limit calculation to relevant RMSDs

Describing “soft-body” rotations



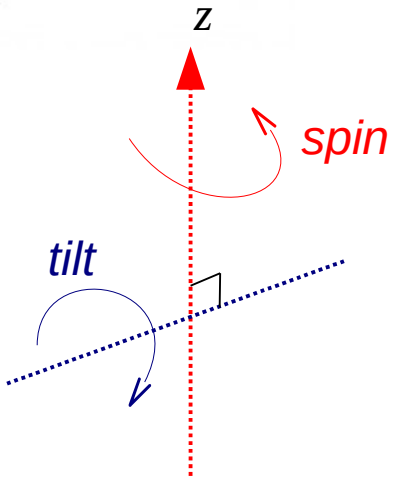
Problem

describe collective rotation of *flexible* objects

- least-squares fit, minimizing

$$d^2 = \left(R(X) - X^0\right)^2$$

- solved as eigenproblem with quaternion representation of rotations
- use **optimal rotation as coordinate**

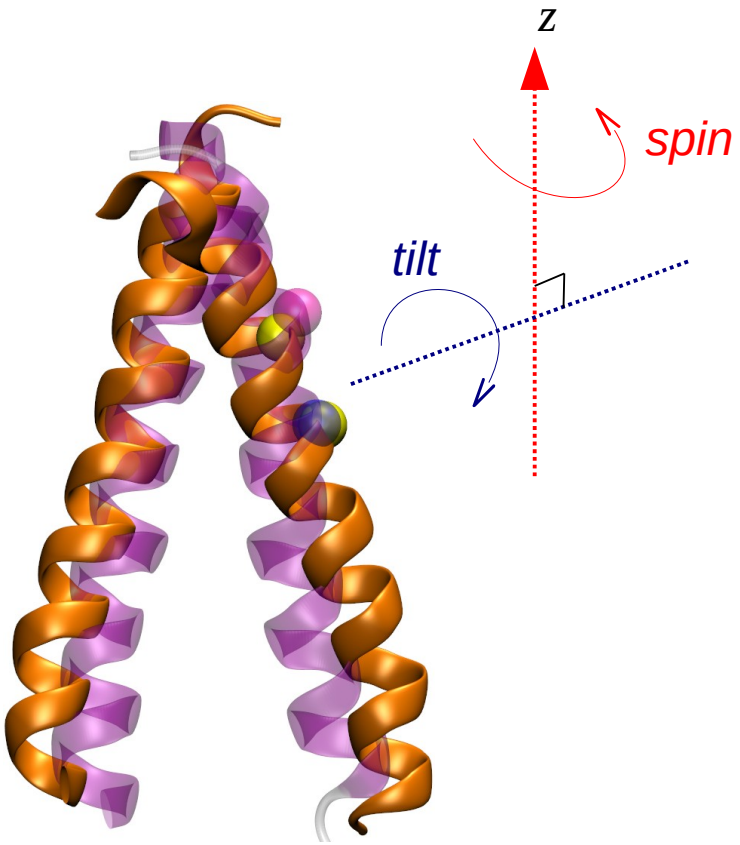


- preferred axis *z*
- rotation decomposed into *spin* and *tilt*
- gives two rotation angles

Local frames of reference

- all coordinates based on atom groups
- atom groups can center and rotate themselves transparently to fit reference positions, working in a separate frame of reference
  - **centerReference** translation
  - **rotateReference** best-fit rotation
  - contribution of rotation to the gradients is calculated

Application 1: internal rotations in a dimer



To describe *relative* rotation of one helix

- fit pair of objects, minimizing

$$d_{AB}^2 = \left(R_{AB}(X_{AB}) - X_{AB}^0\right)^2$$

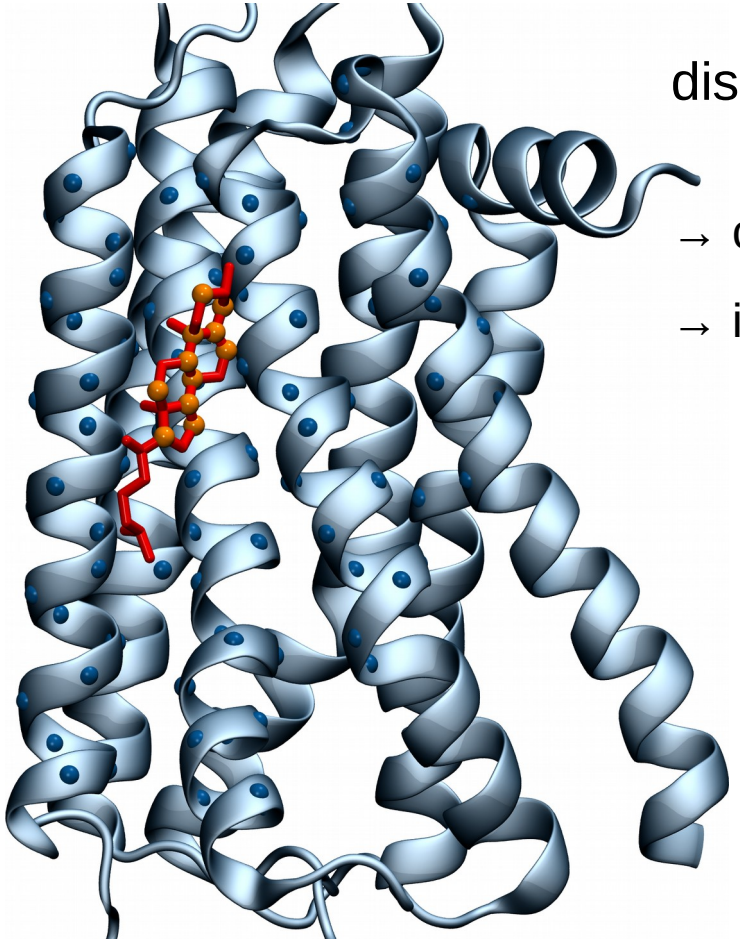
- fit of one object A, relative to pair

$$d_A^2 = \left(R_A \circ \tilde{R}_{AB}(X_A) - X_A^0\right)^2$$

- rotation R_A split into *spin* and *tilt*
- gives two internal rotation angles

- defined at run-time, no coding needed

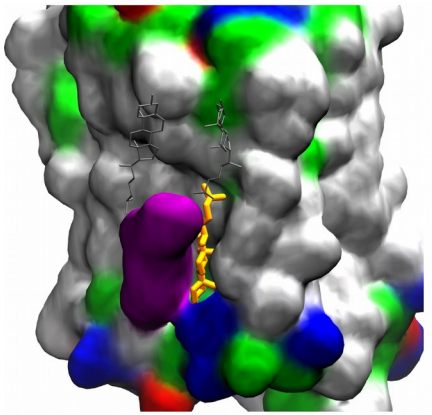
Application 2: ligand binding coordinate



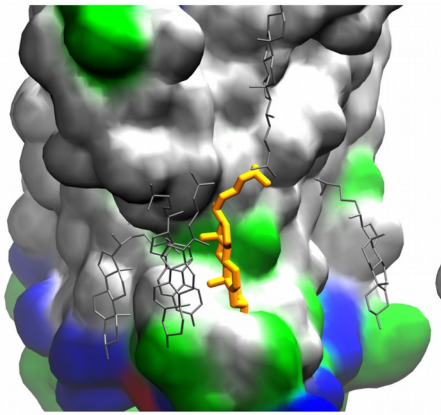
distance to bound configuration (DBC)
= ligand RMSD in *receptor's frame of reference*

→ captures **ligand** position, orientation and conformation

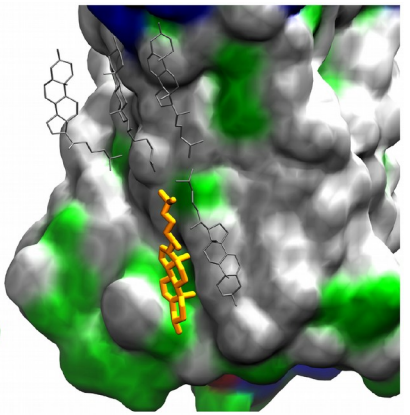
→ independent of **receptor** position, orientation, and conformation



β2-Adrenergic
3D4S
$x_{50} = 10^{-9}$



5-HT2B
4NC3
$x_{50} = 10^{-2}$

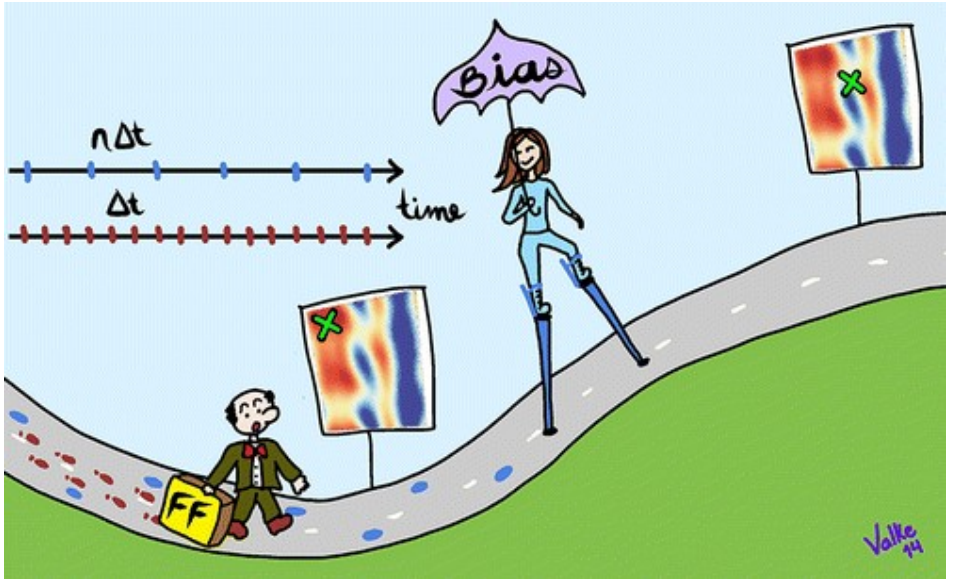


μ-Opioid
5C1M
$x_{50} = 0.7$

GPCR-cholesterol binding affinities

Brannigan, Hénin & coworkers, in press

Multiple time-step colvars



- Bussi and coworkers (Ferrarotti et al. 2014)
- colvar forces are *slow forces*
- colvars can be coarse-grained in time
- biasing forces on colvars integrated at coarse time using impulses as in r-RESPA
- extended-Lagrangian case *explicit* coarse-time dynamics

Write your own bias (WYOB)

Adiabatic Bias MD (Marchi et al. 1999) pushes a variable with a “ratchet potential” that follows the variables high-water mark (highest level reached)

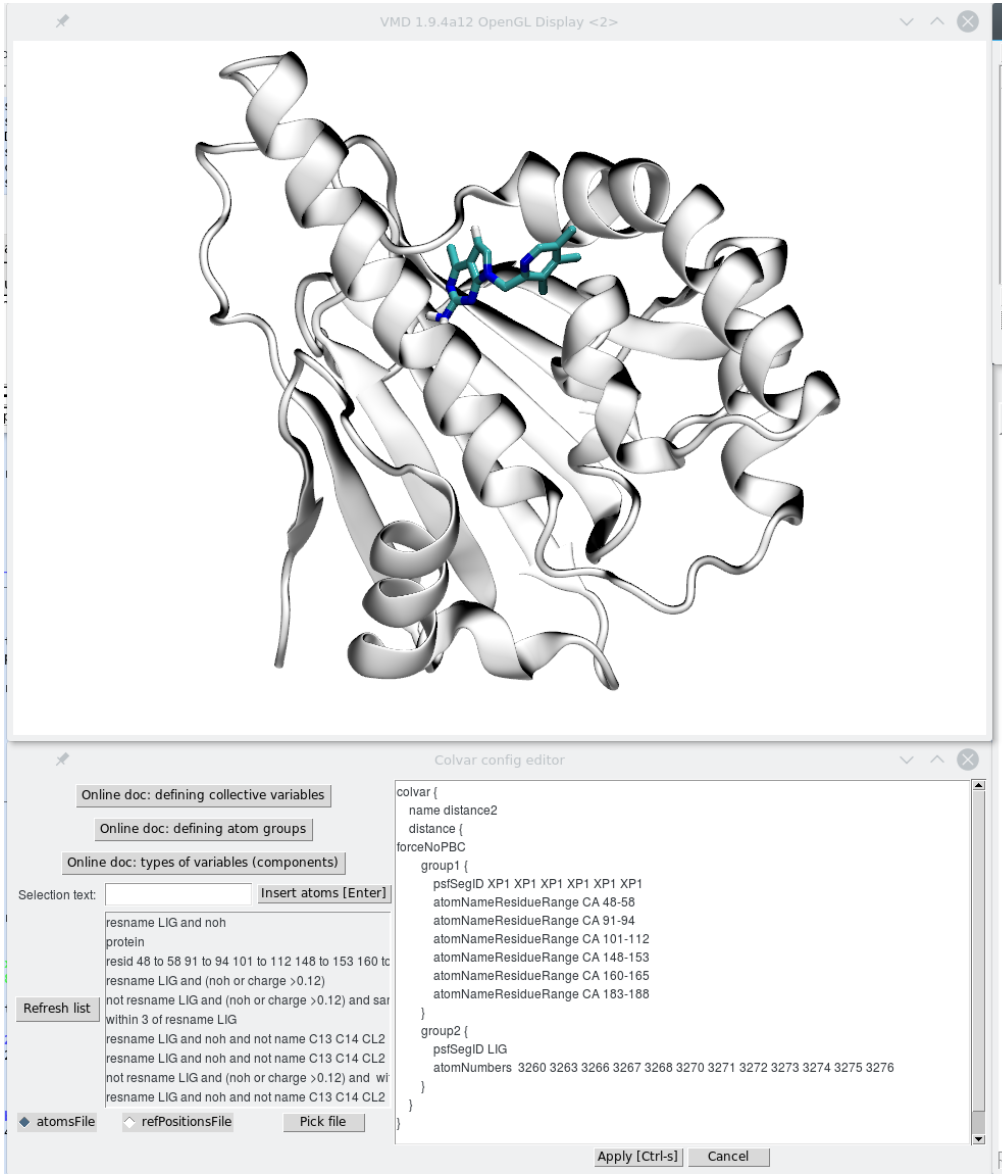
In a few lines of Tcl/Colvars:

```
proc calc_colvar_forces { ts } {
  if { $ts == 0 } {
    set max [cv colvar $cvname value]
  }

  set x [cv colvar $cvname value]
  if { $x > $max } {                                  ;# above high-water mark?
    if { $x <= $xmax } { set max $x }                 ;# then raise it
  } else {
    cv colvar $cvname addforce [expr { $k * ($max - $x) } ]  ;# else apply bias
  }
}
```

(https://github.com/Colvars/colvars/blob/master/colvartools/abmd.tcl)

The Colvars Dashboard in VMD

https://raw.githubusercontent.com/Colvars/colvars/dashboard/vmd/scripts/cv_dashboard.tcl



Practical tricks

- the Colvars module can read multiple configuration files / strings
  - colvarsConfig <file> *(NAMD only)*
  - cv configfile <file>
  - cv config “<config string>”
- → you can split your input files to reuse common parts
- e.g. one file for variables only, one for biases
- config for variables can be written by Colvars Dashboard in VMD (does not handle biases)

- depending on workflow, most **convenient definition of atom groups**:
  - index file (see `tma-aco/Common/write_index_file.tcl`)
  - PDB files with flags
  - atom ID lists from VMD selections (Colvars Dashboard)

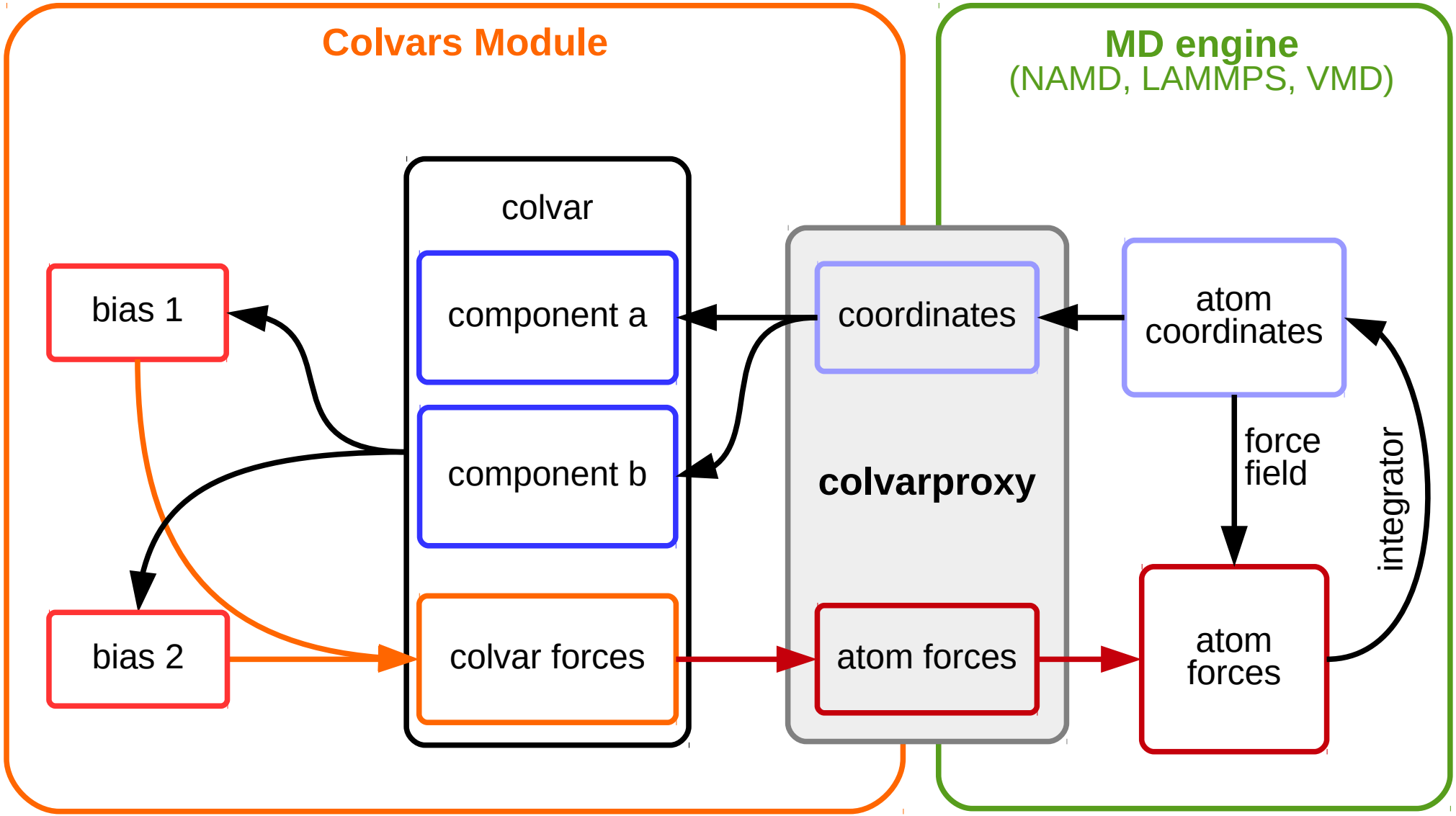
Performance-tuning tricks

- NAMD is highly parallelized
- Colvars is only partially parallelized → can be a bottleneck

- benchmark your own system and colvars on production hardware
- optimizations:
  - use no more atoms than necessary (eg. RMSD on alpha carbons)
  - variables that depend on centers of mass scale better
  - have multiple colvars? Make sure SMP feature is enabled
  - use multiple-timestep colvars if possible
    (first, test carefully for physical consistency; fullElectFreq is often safe)
  - if not all variables are needed at all times, write script setting cvcflags
    (see pathCV example), or even creating or deleting colvars on-the-fly

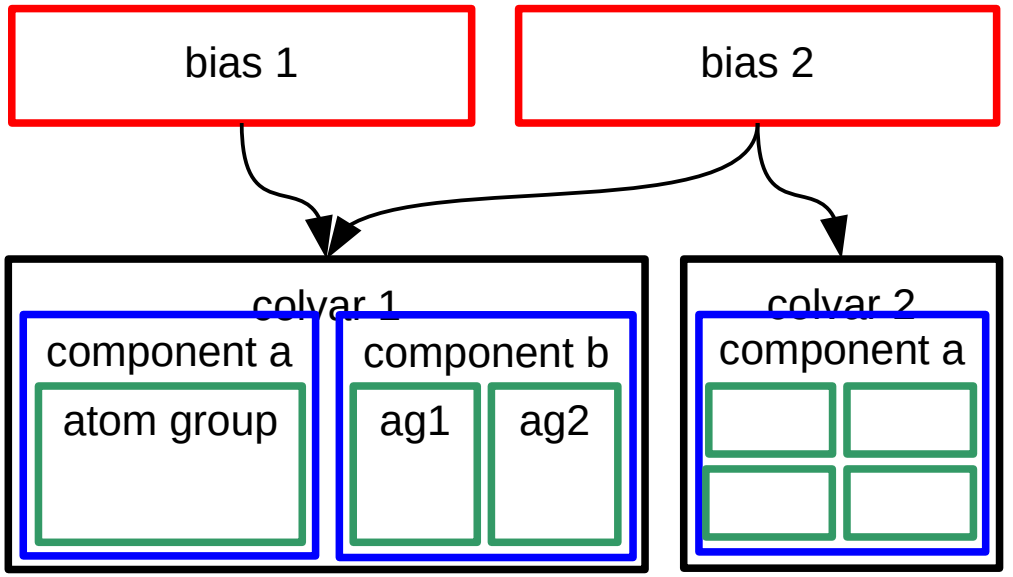
under the hood: a developer's view

Interface with MD engines: the *colvarproxy* class



The dependency problem in a modular code

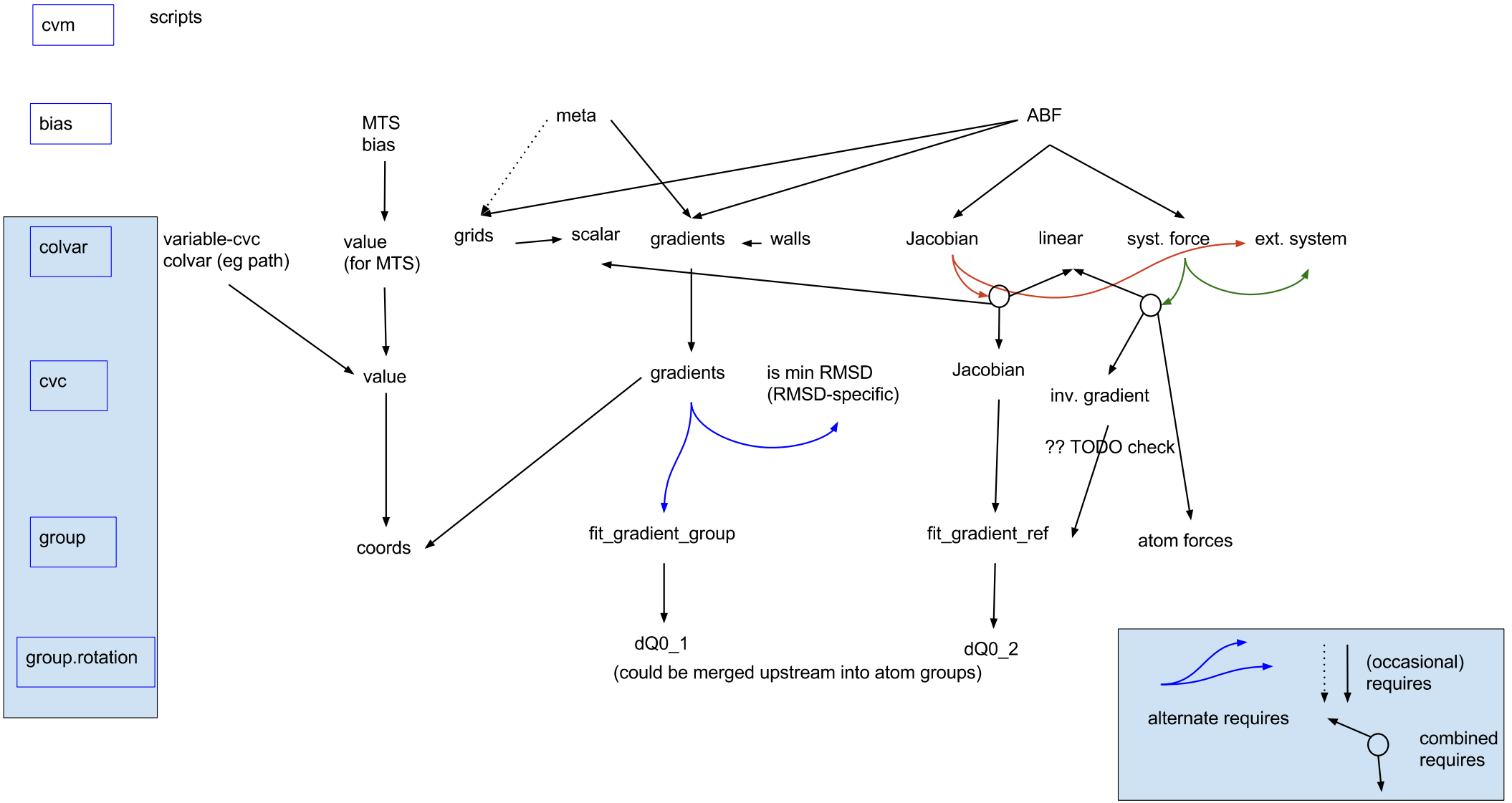
- hierarchy of objects



- objects have many features that can be combined
- modular combinations are key to functionality

- originally dependencies implemented as control structures in the code
- very hard to maintain there are more use cases than we can think of

Draft dependency tree



Current dependency tree

(generated from the code)