SCREAM—A Protein Sidechain Placement Module

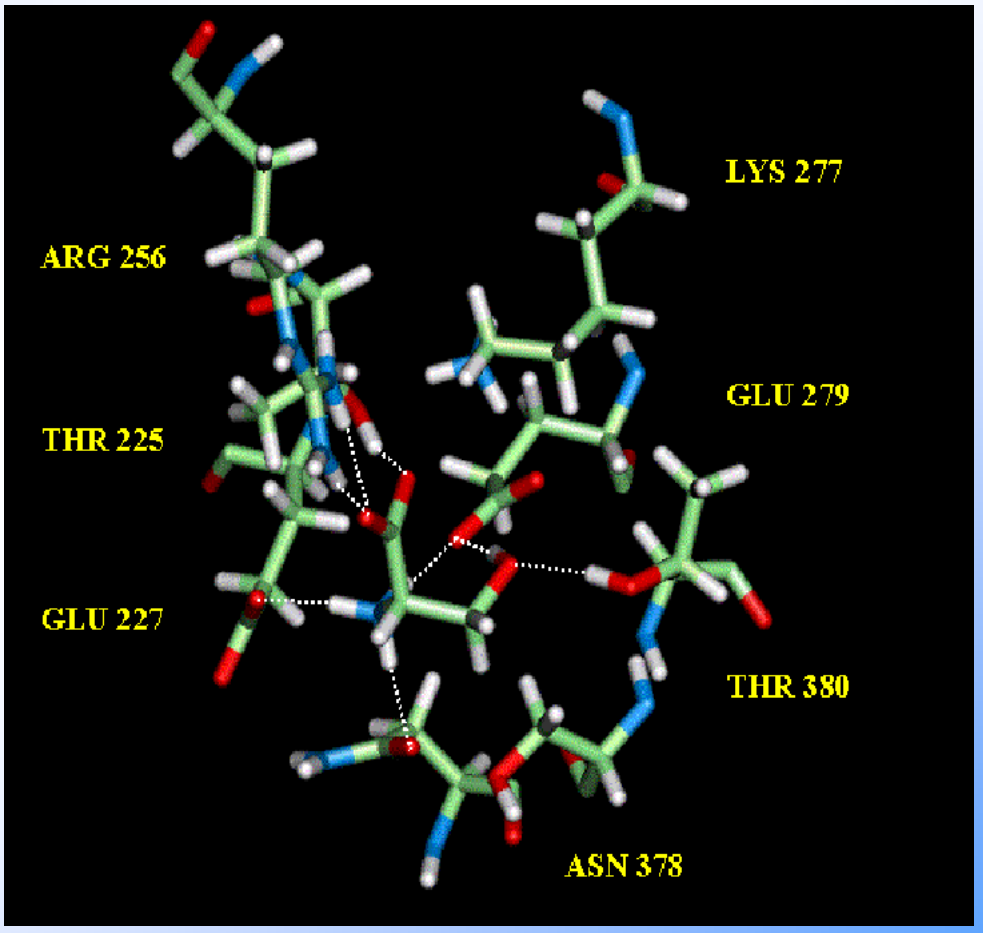
Victor Wai Tak Kam, Nagarajan Vaidehi, William A. Goddard III

Overview

- The Sidechain Placement Problem
- Components of SCREAM (*Side*Chain *R*otamer *E*nergy *A*nalysis *M*ethod)
  - Rotamer library
  - Scoring function
  - Placement Algorithm
- SCREAM and CMDF

Placement of SideChains in Proteins

- Placement of sidechains in proteins has wide-ranging applications, from ligand docking to mutant design
- The Sidechain placement problem (or the *Protein Design* problem) is *NP-hard:* no efficient exact algorithm is believed to exist



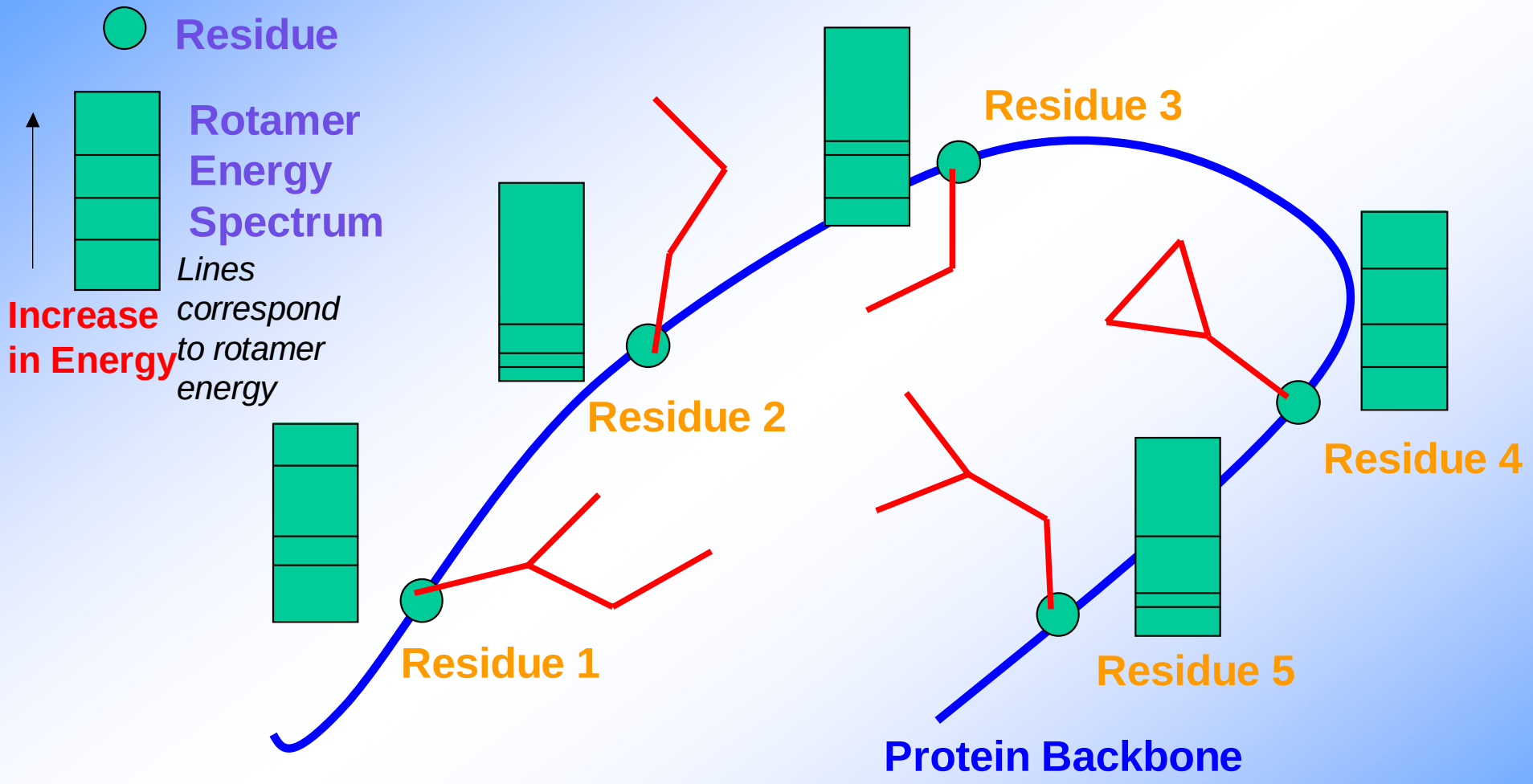
Raison d'etre for SCREAM

- popular programs such as SCWRL exist, why SCREAM?
- SCREAM is designed for the designing of active sites in mind
  - Full-atom forcefield important for accurate placement of critical sidechain in an active site (e.g. H-bond)
  - Capable of handling user-defined rotamers (such as non-natural amino acids) and rotamer libraries

Components to a Sidechain Placement Program

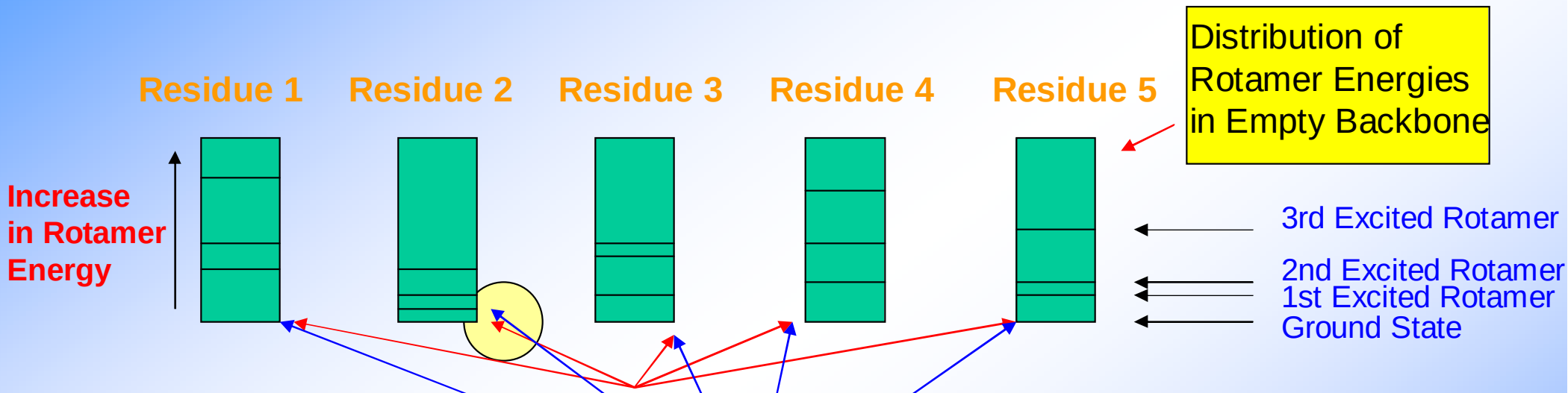
- Rotamer Library
  - sidechains are represented by a set of discrete conformations
- Scoring Function
  - a function which determines which configuration of rotamers are more favorable
  - typically, this is an energy function
- Placement Algorithm
  - the NP-hard part of the problem: how do you go through the rotamers to pick up the best set?
  - brute force: say 10 residues, each has 10 rotamers, ➔10^{10} possible combinations
  - existing methods: Dead-end elimination, divide-and-conquer methods, etc.

The SCREAM Placement Algorithm—Energy Excitation (I)



The Rotamer energy spectrum is calculated in the absence of other sidechains, i.e. each rotamer interacts only with the protein backbone.

The SCREAM Placement Algorithm—Energy Excitation (II)



Starting point: pick ground state for all residues, evaluate energy

Next: pick the set of rotamers with lowest Linear Sum energy, evaluate energy

In this case, it would be taking the 1st Excited rotamer from Residue 2

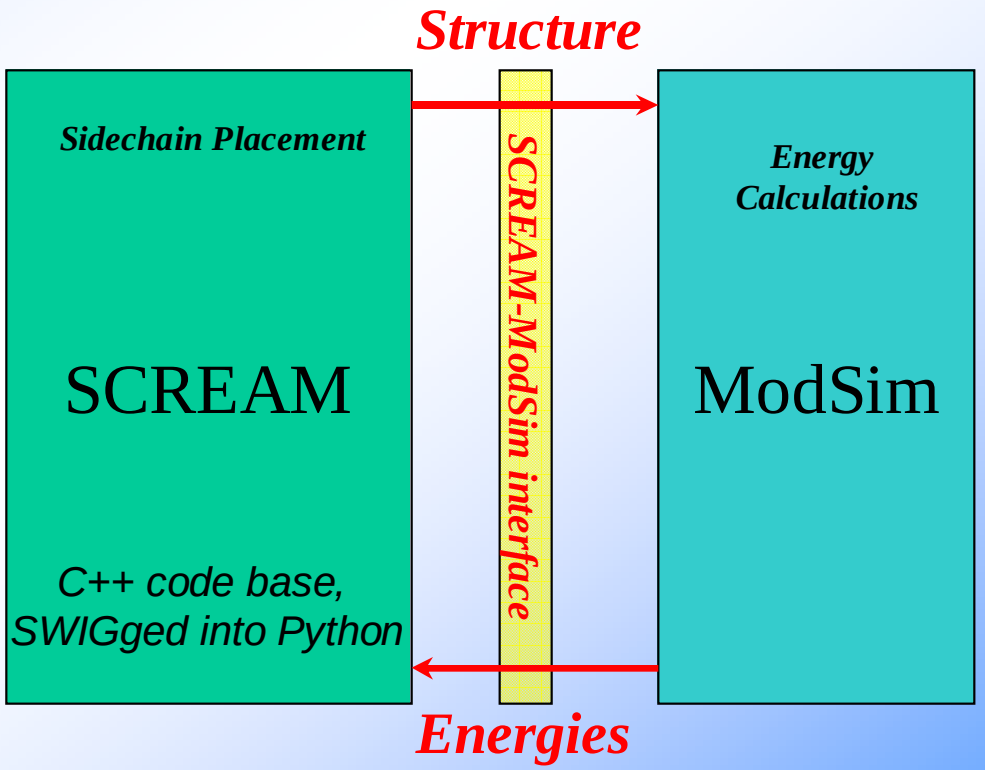
Keep picking the next lowest Linear Sum set of rotamers, and evaluate energy

Prototyping Using CMDF

- Python, a scripting language, makes prototyping easy
- When proving concept—no need to write routines that calculate energies for sidechains, use an existing module instead
- All that is needed is an interface

Using ModSim to Calculate Sidechain Energies

- Logic:
  - Place rotamers in SCREAM module
  - Copy coordinates over to ModSim via the interface
  - Calculate energies using ModSim
  - Return energies back to SCREAM
  - Iterate



Example: Placing Rotamers and Calculating Energies

```
import ModSim, SCREAM
from SCREAM_algorithms import *

[… initialization of ModSim model and Protein]

Rotlib = SCREAM.Rotlib(filename)
Rotamer = Rotlib.get_next_rot()
while (Rotamer != None):
  Protein.placeRotamer(Rotamer, chain_name, pstn)          # Sidechain placement
  remapProteinToModSimModel(Protein, modsim_model)         # Passing structure info
  modsim_model.compute_thermo()                            # Energy calculation
  Rotamer.set_empty_lattice_E_abs(modsim_model.thermo_properties.te)      # Store it!
  Rotamer = Rotlib.get_next_rot()                          # Iterate
```

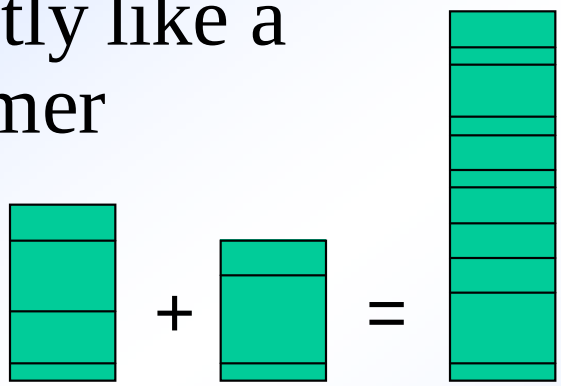
Placement for Large Systems—Rotamer Clustering Scheme

- Can combine any number of rotamers to form a rotamer cluster that behaves exactly like a rotamer

$$E_{tot}(A,B) = E_{self}(A) + E_{self}(B) + E_{Int.}(A,B) \equiv E_{self}(AB)$$

$$E_{tot}(A,B,C) = E_{self}(A) + E_{self}(B) + E_{self}(C) +$$
$$+ E_{Int}(A,B) + E_{Int}(A,C) + E_{Int}(B,C)$$
$$= E_{self}(AB) + E_{self}(C) + E_{Int}(A,C) + E_{Int}(B,C)$$
$$\equiv E_{self}(AB) + E_{self}(C) + E_{Int}(AB,C)$$

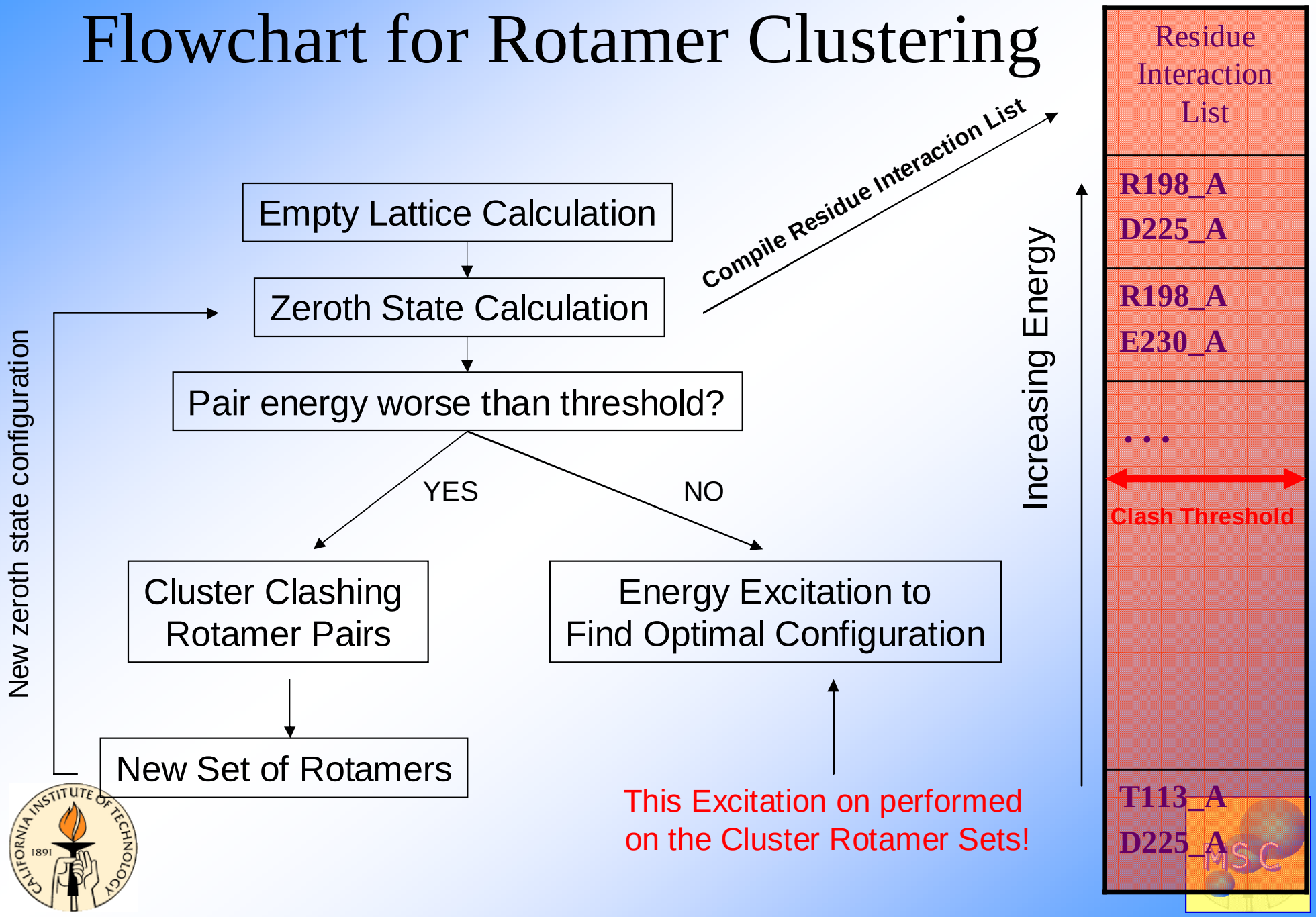
Which establishes the recursive relation. This only works if E_{self} is defined as the empty lattice energy.



Residue A spectrum | Residue B Spectrum | Residue AB spectrum

(includes interaction energies Between residue A and B)

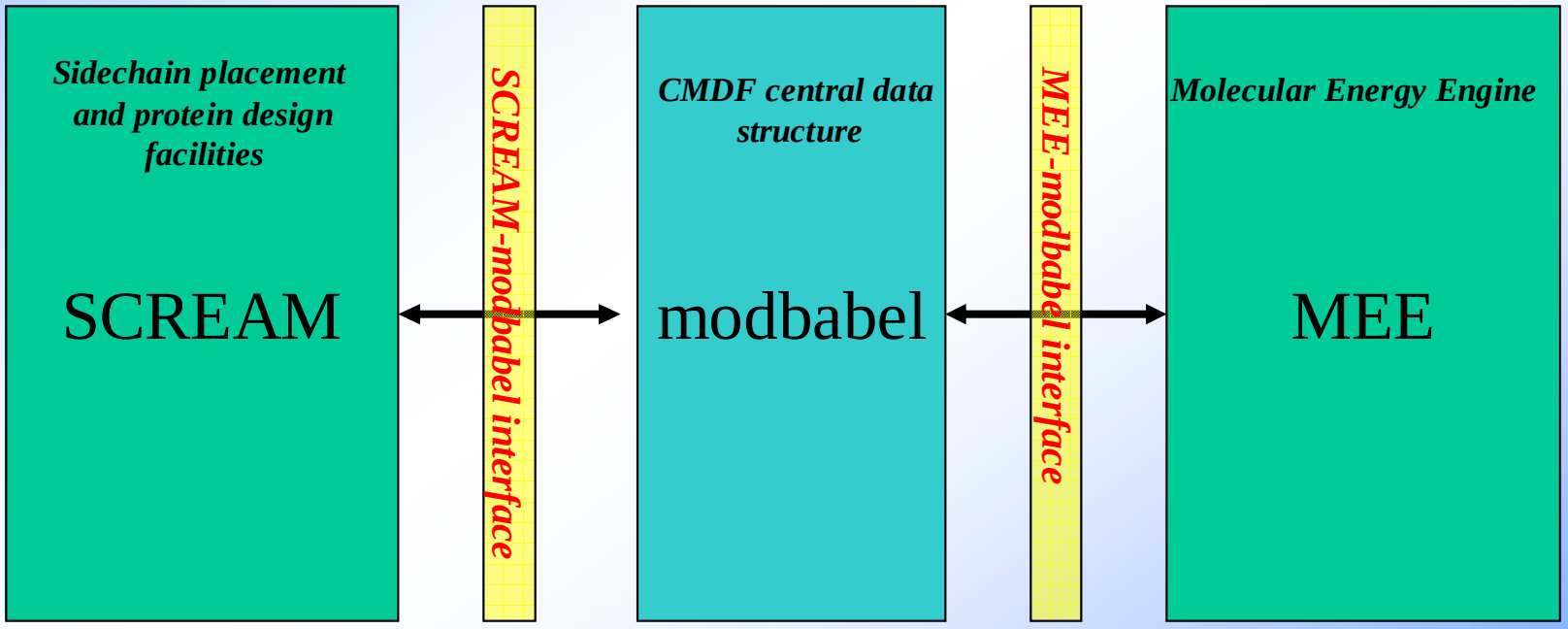
Flowchart for Rotamer Clustering



SCREAM in CMDF

- SCREAM is a fully functional module in CMDF, i.e. interface, wrappers, etc all incorporated
- Can use energy functions from CMDF modules instead of SCREAM energy functions
- Other options to the CMDF end-user: user-defined rotamer libraries, mutations, protein design facilities
- Functionalities such as minimization and dynamics can be easily imported from pertinent modules (MEE etc.)

Example: Using minimization from an External Module



Python Script

```
import Py_SCREAM, MEE # engines
import modbabel              # CMDF central data structure
import SCREAM_OBMOL, MOLTOOLS       # interfaces

scream_model = Py_Scream_EE.ScreamModel("scream.par")  # SCREAM init.
obmol = modbabel.OBMol()
SCREAM_OBMOL.InitOBMolFromScream(scream_model, obmol)       # Mapping
MEE.cmdf_mee_init(5000)
MOLTOOLS.OBToMOLSCAPE(obmol, MEE)          # MEE init

steps = 10
conv = 0.1
MOLTOOLS.Minimize(steps, conv, MEE)                    # Do minimization
```