

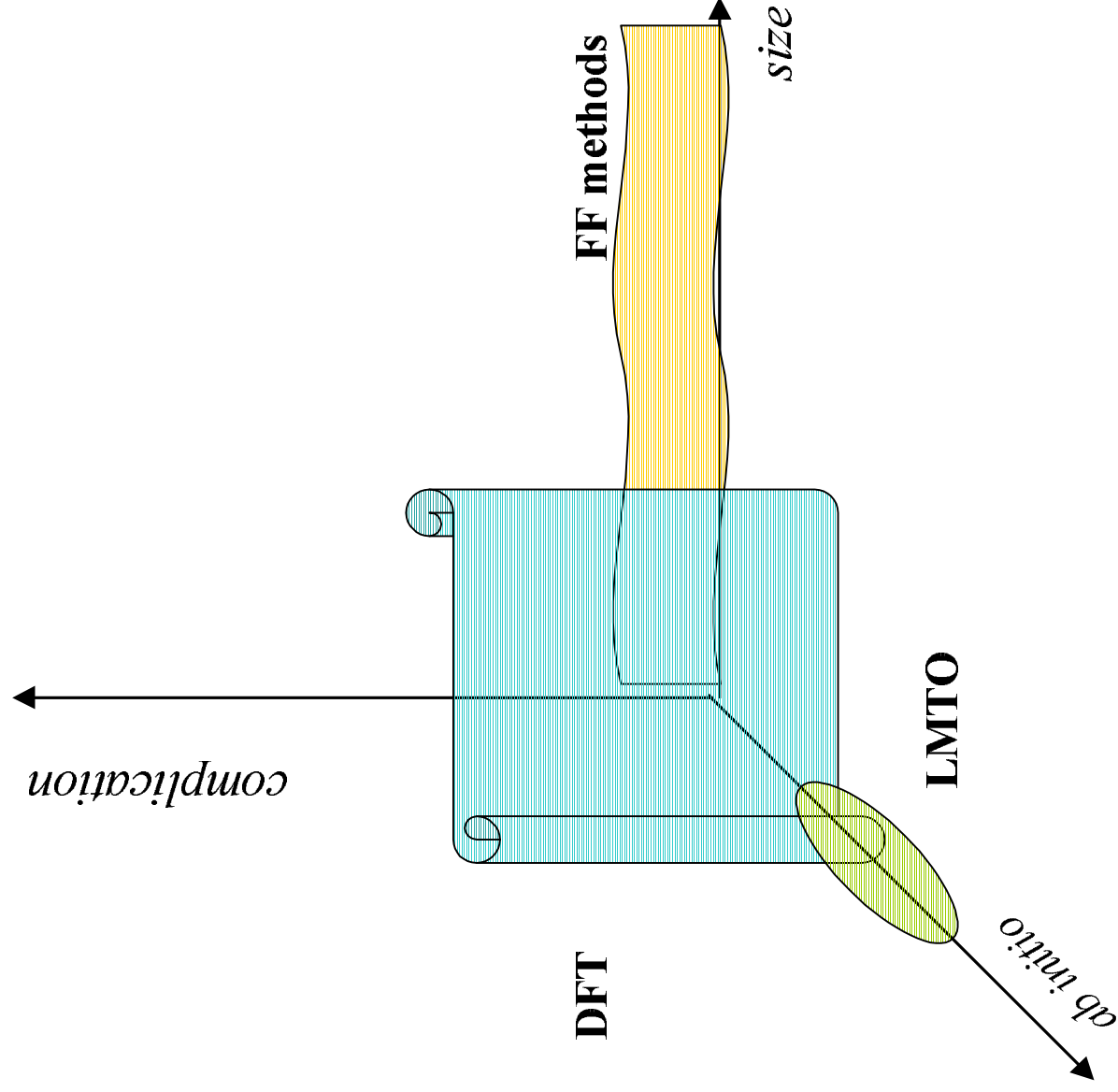
# CAMP Open Software

Zbigniew Łodziana  
CAMP, Dept. of Physics,  
Technical University of Denmark

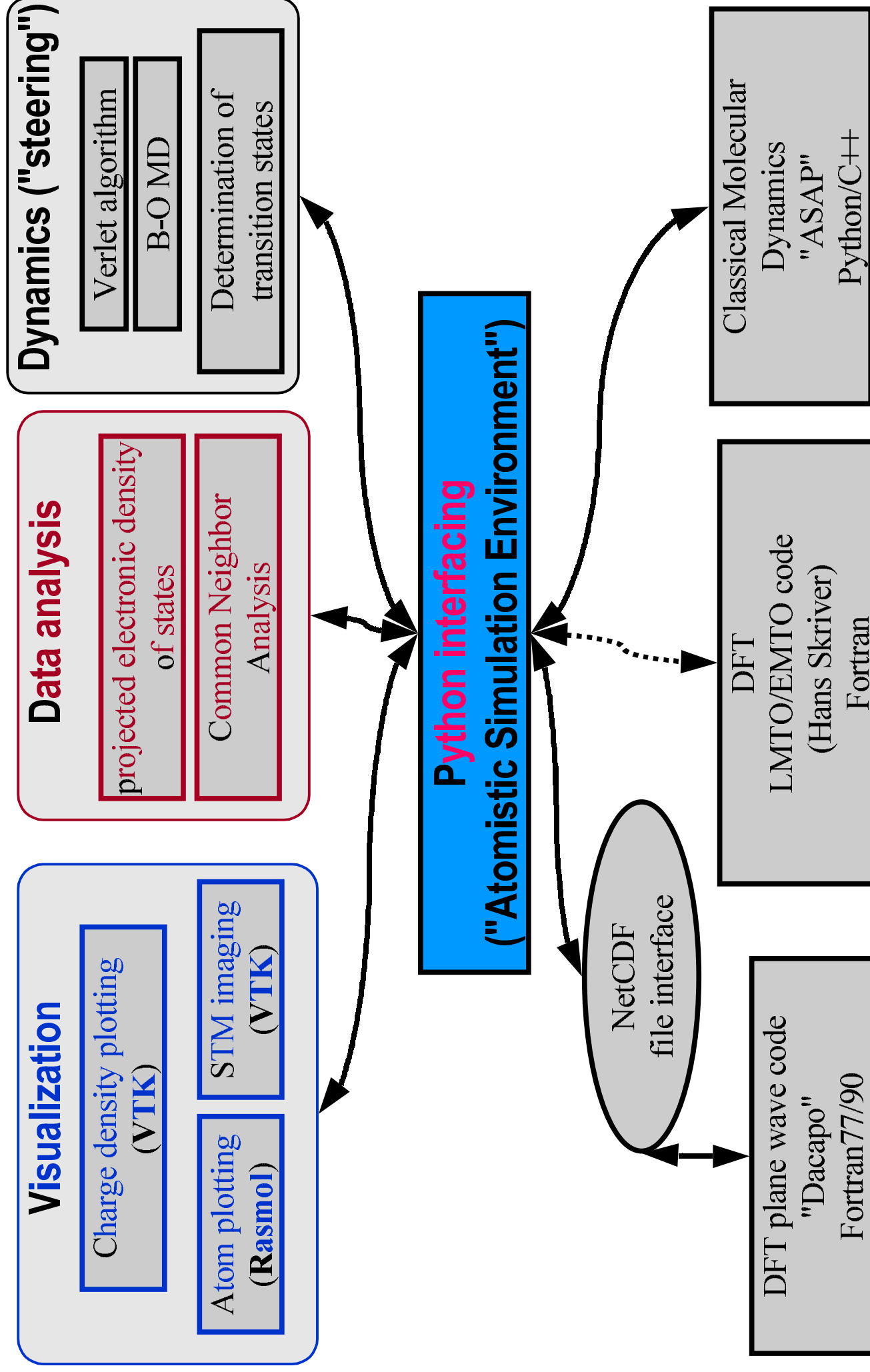
**PSSIT 2003**

Modeling and Simulation in Micro and Nano Technologies and Material Engineering

# Atomistic simulation space



# CAMPOS Project



# Legacy DFT Code "Dacapo"

- . Density Functional Theory (DFT) code
  - Fortran 77/90
  - Plane wave basis
  - Ultrasoft pseudopotentials (Vanderbilt)
  - Iterative diagonalizations
  - MPI parallelisation
    - . k-points, spin, plane waves/band (FFT on one node)
  - Double grid (density, plane waves)
  - Stress calculations
  - Suitable for surface calculations

# MD Code "ASAP"

- . Classical molecular dynamics code
  - Python/C++
  - Effective Medium Theory interatomic potentials
    - . Energy/forces/stresses
  - Neighborlists (short range int.)
  - MPI parallelized
  - Dynamics/constraints in common with DFT programs (except for parallel issues)
  - Quasicontinuum extension - *link to finite element method*

Jakob Schiotz

# LMTO/EMTO code

- . Density Functional Theory
  - Linearized Muffin Tin Orbitals basis (localized)
  - Exact Muffin Tin Orbitals basis (3<sup>rd</sup> generation)
  - Locally self-consistent Green function method (order N)
  - Periodic boundary conditions and interfaces
  - Alloys - coherent potential approximation
  - Generalized Perturbation Method potentials

Hans Skriver

# Python Programming/Interfacing

- Python
    - Object oriented
    - Script language (interpreted)
    - Dynamic types - little type checking
    - Fast prototyping
    - Flexible exception handling
- 

## Code/Interface Design Goals

- Fast; Reliable
- Easy and flexible to use
- Easy and flexible to maintain and extend
- ...

# Easy and Flexible to Use

- . First time users
  - Easy to do standard calculations
  - Help with setup/defaults
- . Power users
  - Advanced combinations of calculations
  - New unexpected use
  - "Layered" or modular code to dig into

# Easy to Maintain and Extend

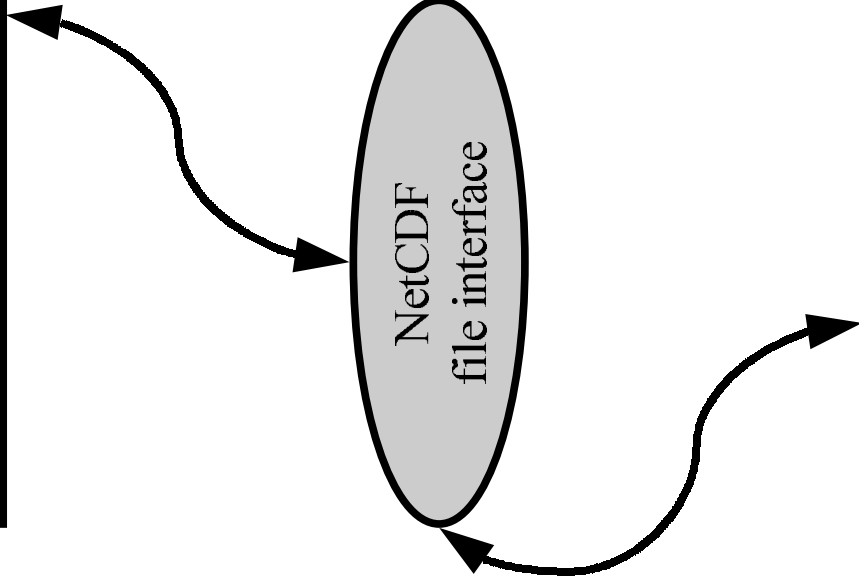
- Modularity → interfacing
  - Separation of DFT/MD number crunching, dynamics, visualization, data analysis, "steering"
  - Many developers
  - Many programming languages
  - Many different needs which vary with time in unpredictable ways

## Some Strategies

- . Reuse well-established code
- . Use available standards if possible
  - Python, NetCDF
- . Use available OS software
  - CVS, mailing list (listserv)
  - FFTW, BLAS, ...
  - Numeric python, Scientific Python, ...
  - Gnuplot, Visualization Tool Kit (VTK), ...
- . Cross platform?
- . GPL license

# The Dacapo Interface

Python interface  
("Atomistic Simulation Environment")



NetCDF file format:

- Binary
- Platform independent
- Fast, random access
- Data accessed by name
- IO routines and tools available for C, Fortran, Python, ...

DFT plane wave code  
"Dacapo"  
Fortran 77/90

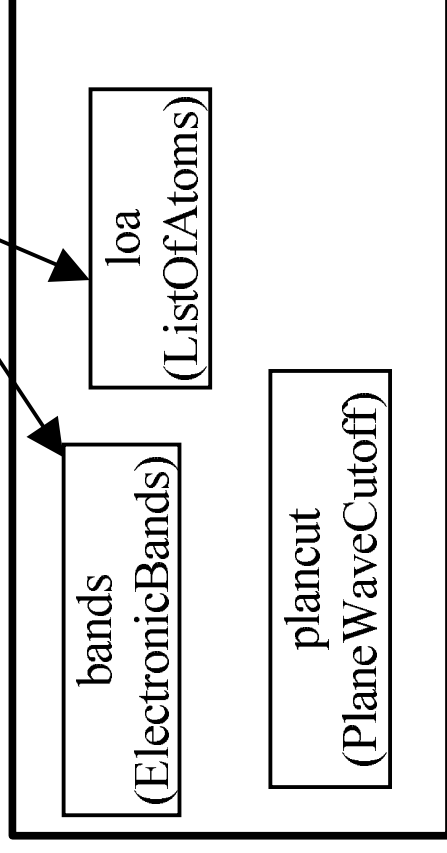
# Python Scripting Interface

A simple example:

```
from Simulations.Dacapo import *
mysim=Simulation()
mysim.bands=ElectronicBands(9)
mysim.loa=ListOfAtoms(
    atoms=[Atom(Mg_GGA, Vector([0,0,0])],
    unitcell=BravaisLattice([[ -1.425, 1.425, 1.425],
    [1.425, -1.425, 1.425],
    [1.425, 1.425, -1.425]])])

mysim.plancut=PlaneWaveCutoff(340)
mysim.Execute()
```

mysim Python "attributes"



Execute method:

Ask all attributes to write themselves to the NetCDF file and start "dacapo".

# Flexibility of scripting

Check of convergence with respect to plane wave cutoff:

```
from Simulations.Dacapo import *
mysim=Simulation()
mysim.bands=ElectronicBands(9)
mysim.loa=ListOfAtoms(
    atoms=[Atom(Mg_GGA, Vector([0,0,0]))],
    unitcell=BravaisLattice([[ -1.425, 1.425, 1.425],
                              [ 1.425, -1.425, 1.425],
                              [ 1.425, 1.425, -1.425]])])
for cutoff in [250,300,350,400]:
    mysim.plancut=PlaneWaveCutoff(cutoff)
    mysim.Execute()
    print mysim.loa.GetTotalEnergy()
```

The same with k-points, vacuum,  $T_{F_5}$ , ....

# Advanced Example

```
Al.SetProperty("PseudopotentialPath", "Al_SPECIAL.pseudo")

sim.UpdateFromNetCDFFile("POSITION.nc")

sim.atoms[32:44].SetConstraints(12*['123'])

sim.soft = PlaneWaveCutoff(340)
sim.hard = NetCDF.Entry("Density_WaveCutoff", 500)

sim.dipolecorr = NetCDF.Entry(name="DipoleCorrection")
sim.dipolecorr.MixingParameter=1.00

from IO.ListOfAtomsObserverNetCDF import ListOfAtomsObserverNetCDF
observer = ListOfAtomsObserverNetCDF(sim.atoms,netcdffilename=obsncfile)
# Invoke the python ionic relaxation
from Minimizations.LineWiseSearch.Minimizers import ConjugateGradientSearch

task = ConjugateGradientSearch.RecommendedParameters
param = task.GetParameters()
param['LineMinimization']['GradientAccuracyGoal'] = 0.050 # modify parameters
param['GlobalMinimization']['LogFile'] = "mylog"

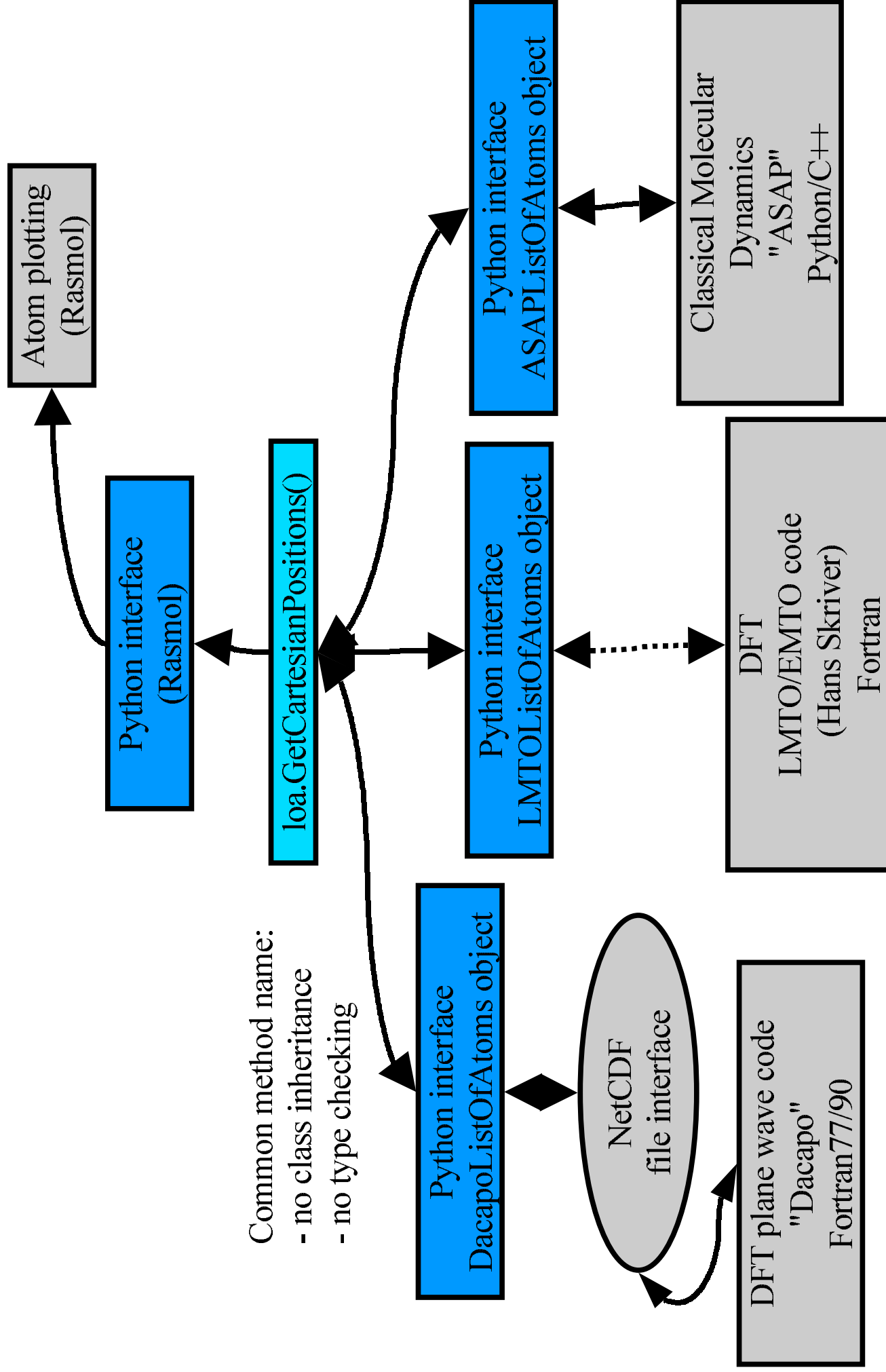
srch = ConjugateGradientSearch(sim.atoms, sim.GetExecuteMethod(dynamics='external'))
srch()

sim.dynamics.Type = "ExternalIonMotion"
sim.dynamics.ExternalIonMotion_script = ".bmd.py"
```

# Interface Development

- . Determination of objects
  - What is an atom? Which properties does it have?
- . .. and methods
  - What are basic operations involving atoms?
- . A key issue: Naming of methods
  - The methods ARE the interface
  - Crucial for further development

# The Methods ARE the Interface



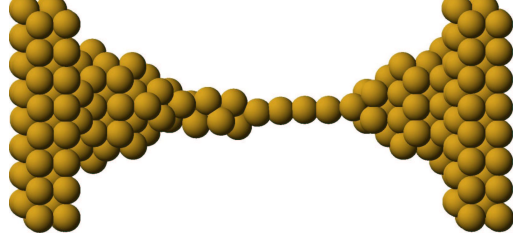
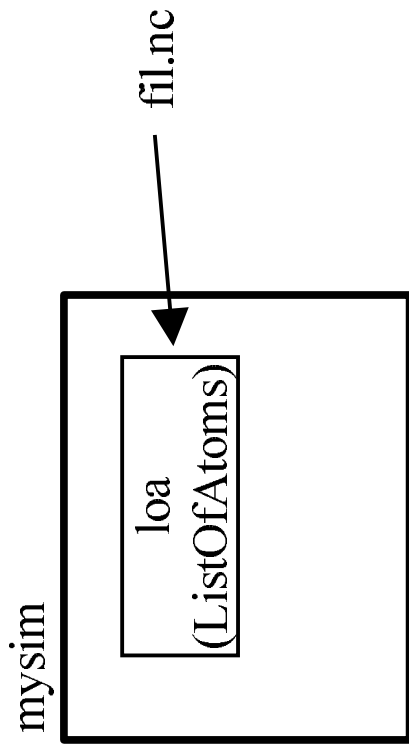
# How does it look to the user?

```
from Simulations.Dacapo import *
mysim=Simulation()
mysim.loa=ListOfAtoms()
mysim.UpdateFromNetCDFFile("fil.nc")
mysim.loa.GetPlot()
```

# or *mysim.loa.GetVTKAvatar()* to view in VTK

UpdateFromNetCDFFile method:

Ask all attributes to update themselves from the NetCDF file.



Nano chain formation in gold

# Naming of Methods

Several levels of abstraction useful:

`atom.GetPosition()` -> returns some position object  
`atom.GetCartesianPosition()` -> returns coordinates [0,2.5,1.7]

Concrete interface (`GetCartesianPosition`):

- Gives real information right away - gets the code up and running
- Can be quickly implemented
- Specific well-defined meaning which should (ideally) never change

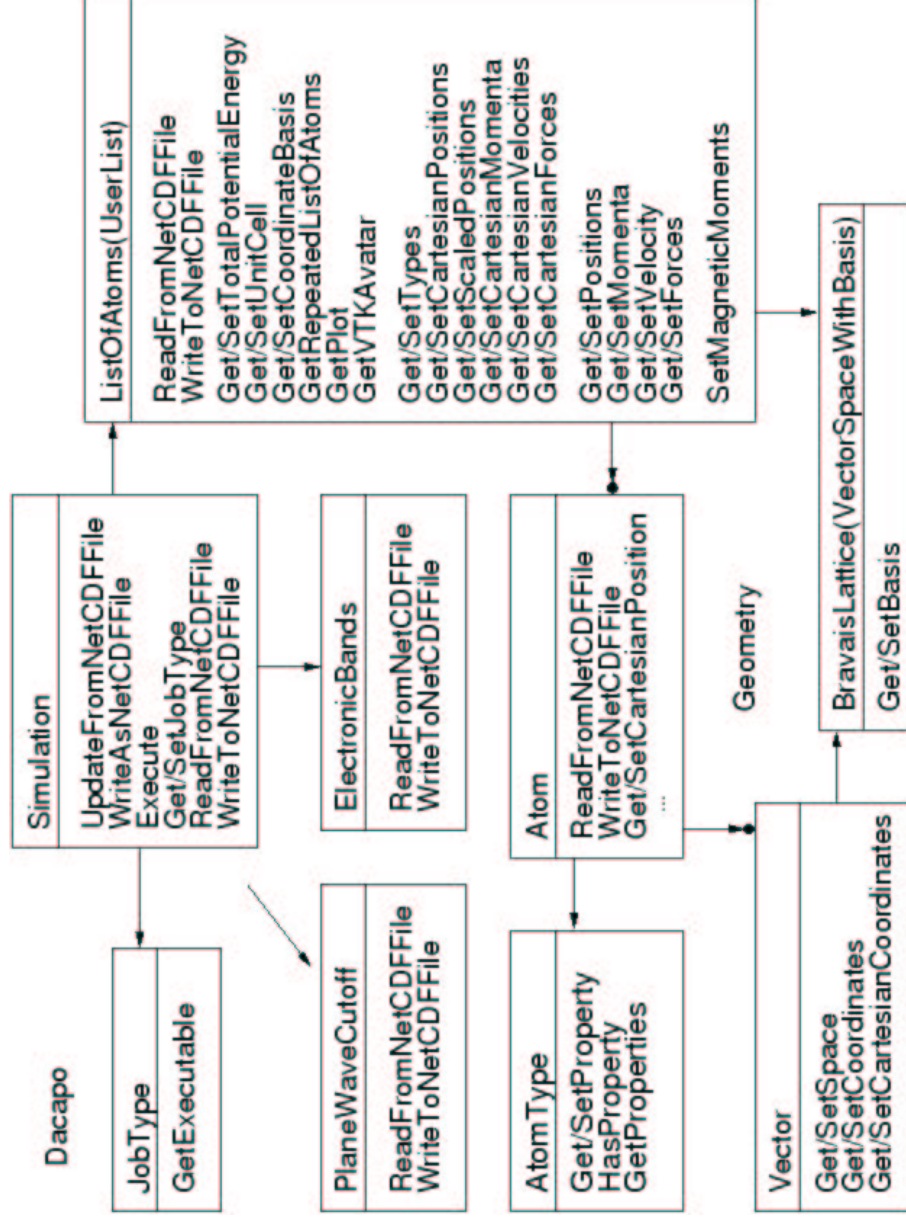
Abstract interface (`GetPosition`):

- Great for advanced manipulation (spherical coord., relative positions)
- Modules may send abstract objects around without the need for evaluation
- Can be extended later ...
- Can (eventually) form the basis for the concrete interface:

`GetCartesianPosition()=GetPosition().GetCartesianCoordinates()`

New modules should preferably ask for concrete interfaces  
Python "try" statement

# Object diagram

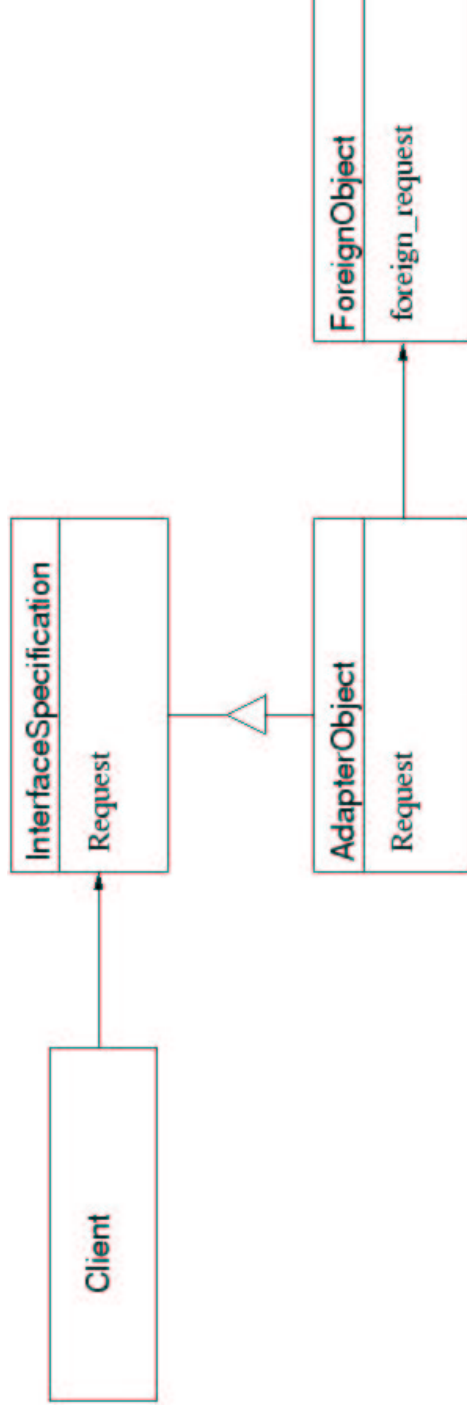


+ electron densities, wave functions, (projected) density-of-states,...

# Object adaptation

Can we use an ASAP ListOfAtoms with dacapo?

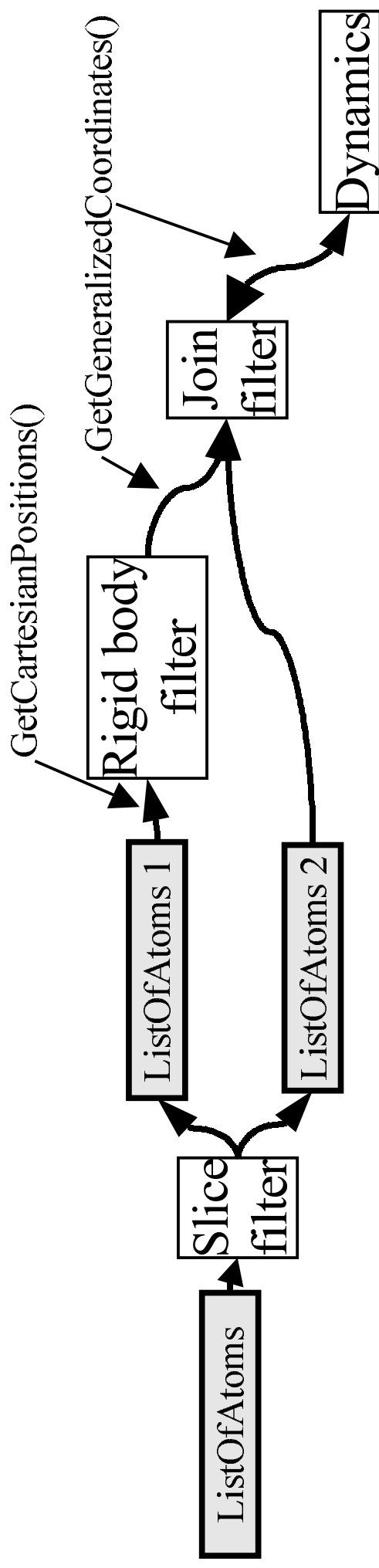
Use object-adapter design pattern:



R. Johnson et al.: "Design Patterns: Elements of Reusable Object-Oriented Software"  
(Addison Wesley Longman, Reading, Mass. 1995)

# Some concepts in the interface

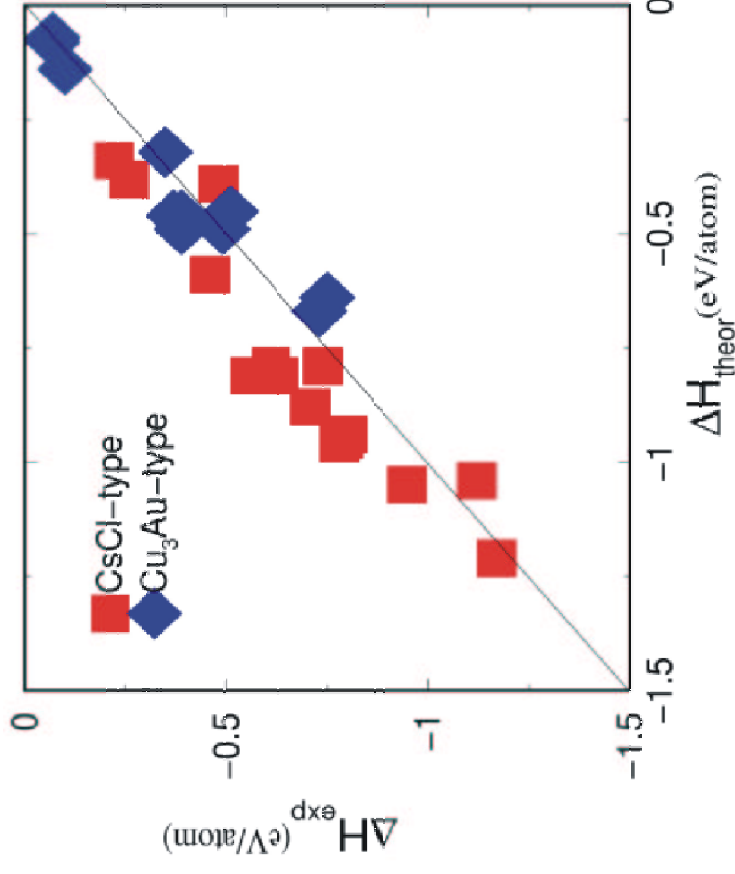
- Avatars - visual representation of objects
- Manipulators - graphical interactive representation of objects
  - GUI build from combining manipulators
  - GUI does not hide the structure but exposes it!
- Filters for constraints



## Electronic Structure Calculations

- Linear Muffin Tin Orbital (LMTO) Method
- KKR for those elements which cause trouble

## Formation Energy, $\Delta H_f$ (eV)



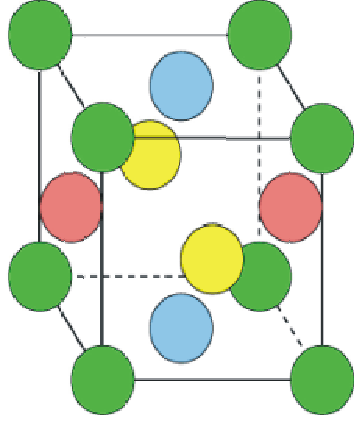
Experiments from "Cohesion in Metals", F.R. Boer et. al. (1988)



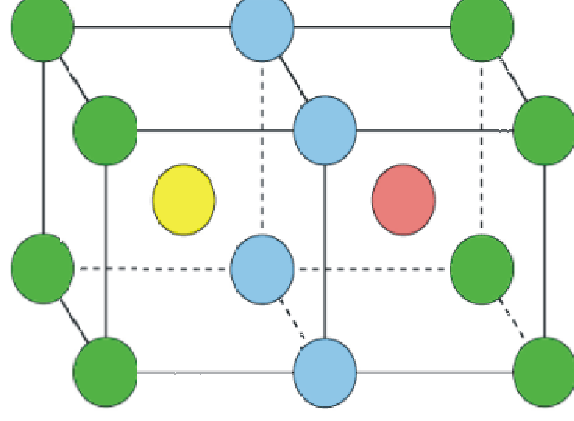
# Optimization with respect to structure

- . 4 atom unit cell
- . fcc and up to 3 different bcc's per alloy
- . pick structure with lowest formation energy

fcc unit cell



bcc unit cell





# Identified alloys

All	fcc	fcc excl. noble	fcc excl. Si			
Pt <sub>2</sub> Y <sub>2</sub>	Pt <sub>3</sub> Sc	-1.06	Si <sub>2</sub> Ti <sub>2</sub>	-0.57	AlNi <sub>3</sub>	-0.49
Pt <sub>2</sub> Sc <sub>2</sub>	HfPt <sub>3</sub>	-1.03	NiSiTi <sub>2</sub>	-0.55	Ni <sub>3</sub> Ti	-0.46
Lu <sub>2</sub> Pt <sub>2</sub>	Pt <sub>3</sub> Y	-1.02	Si <sub>2</sub> TaTi	-0.53	HfNi <sub>3</sub>	-0.44
Ir <sub>2</sub> Sc <sub>2</sub>	PdPt <sub>2</sub> Sc	-0.99	Ni <sub>3</sub> Si	-0.53	Al <sub>2</sub> Ti <sub>2</sub>	-0.43
HfIr <sub>2</sub> Sc	Pt <sub>3</sub> Zr	-0.98	AlSiTi <sub>2</sub>	-0.53	Al <sub>3</sub> Sc	-0.43
IrRhSc <sub>2</sub>	HfPt <sub>2</sub> Rh	-0.98	Ni <sub>2</sub> SiTa	-0.53	Al <sub>2</sub> Zr <sub>2</sub>	-0.42
PtRhY <sub>2</sub>	PdPt <sub>2</sub> Y	-0.97	CoNiSiTa	-0.52	Al <sub>2</sub> ZnZr	-0.42
PdPtY <sub>2</sub>	HfPdPt <sub>2</sub>	-0.96	NiSiTaTi	-0.51	Al <sub>2</sub> Sc <sub>2</sub>	-0.41
PdPtSc <sub>2</sub>	LuPt <sub>3</sub>	-0.95	AlSiTaTi	-0.50	Ni <sub>3</sub> Sc	-0.41
PdPtLu <sub>2</sub>	Pt <sub>2</sub> RhSc	-0.94	Sc <sub>2</sub> Si <sub>2</sub>	-0.50	Al <sub>3</sub> Zr	-0.40
HfIrRhSc	NiPt <sub>2</sub> Sc	-0.93	CoSiTaTi	-0.50	Al <sub>2</sub> TiZn	-0.39
Ir <sub>2</sub> ScZr	Pt <sub>2</sub> RhZr	-0.92	AlNi <sub>3</sub>	-0.49	Al <sub>2</sub> ScZn	-0.38
Hf <sub>2</sub> Ir <sub>2</sub>	HfNiPt <sub>2</sub>	-0.92	SiTi <sub>3</sub>	-0.49	Al <sub>3</sub> Ti	-0.38
Hf <sub>2</sub> Pt <sub>2</sub>	LuPdPt <sub>2</sub>	-0.92	Si <sub>2</sub> Zr <sub>2</sub>	-0.47	Co <sub>3</sub> Ti	-0.38
Rh <sub>2</sub> Sc <sub>2</sub>	PdPt <sub>2</sub> Zr	-0.92	AlSc <sub>2</sub> Si	-0.46	Ni <sub>3</sub> Zr	-0.36
IrPdSc <sub>2</sub>	Pd <sub>2</sub> PtSc	-0.91	Ni <sub>3</sub> Ti	-0.46	Al <sub>2</sub> NbTi	-0.36
PtRuSc <sub>2</sub>	HfIrPtRh	-0.90	NiSiTaZn	-0.46	Al <sub>2</sub> CuTi	-0.35
Al <sub>2</sub> Rh <sub>2</sub>	Pd <sub>2</sub> PtY	-0.90	AlSiZr <sub>2</sub>	-0.46	Al <sub>2</sub> HfZn	-0.34
OsPtSc <sub>2</sub>	HfPtRh <sub>2</sub>	-0.88	SiTi <sub>2</sub> Zn	-0.45	Al <sub>2</sub> CuZr	-0.34
IrRhScZr	HfIr <sub>3</sub>	-0.88	HfNi <sub>3</sub>	-	Al <sub>3</sub> Lu	-0.34
			0.44			

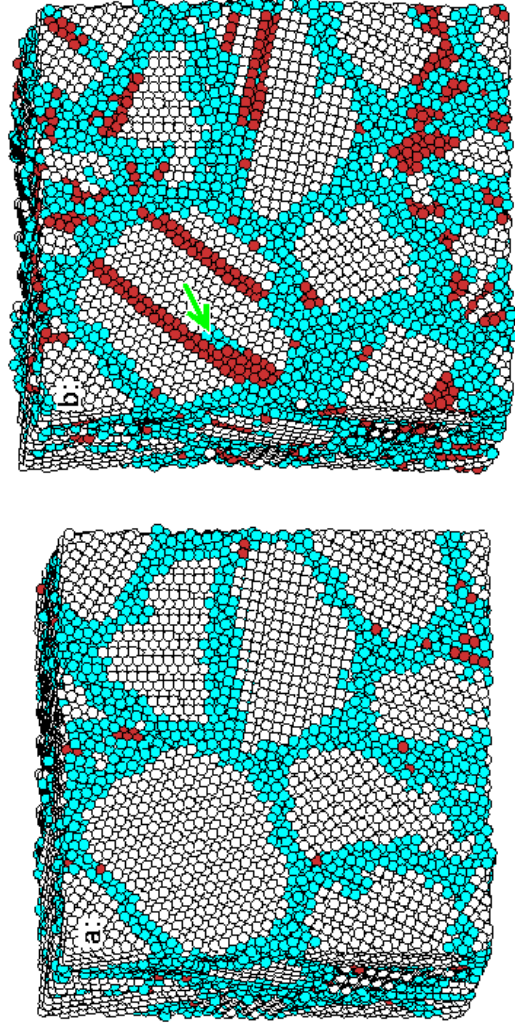
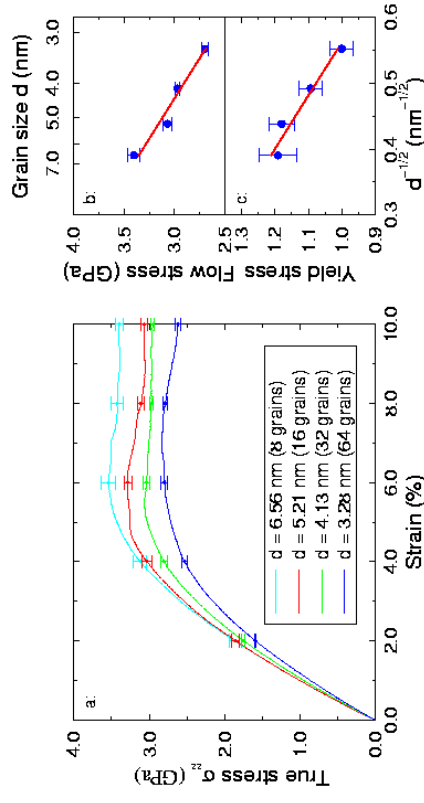
By now  
60000  
alloys  
have been  
calculated

# Conclusion

- . Python Interfacing
  - "Steering" (user/dacapo)
  - Visualization (dacapo/VTK-python)
  - Data analysis (dacapo/C)
  - Object transfer between codes (dacapo/ASAP)

# Plasticity

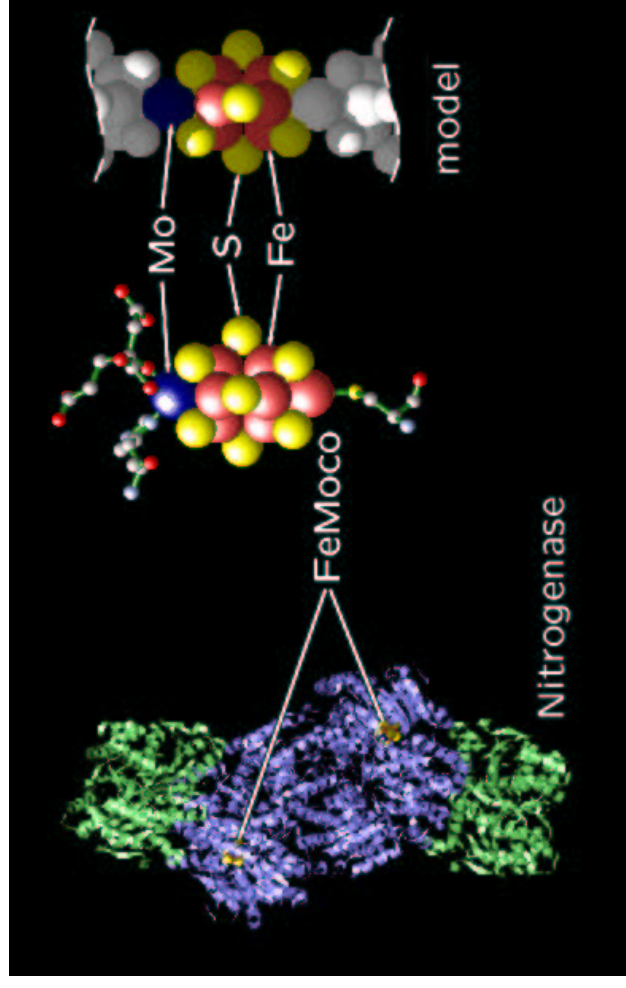
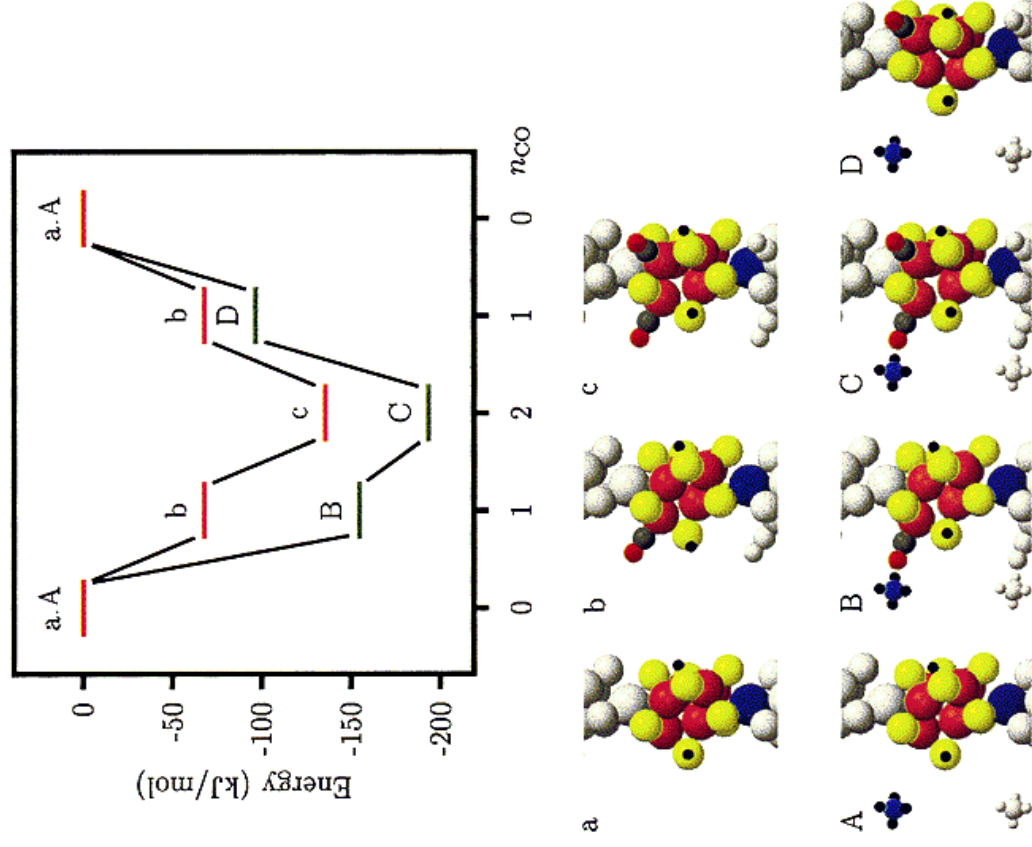
What is the interplay of crystallinity and elasticity ?



# Nitrogenase

What is the mechanism of nitrogenase in plants ?

Nitrogenase: conversion of  $H_2$ ,  $N_2$   
into  $NH_4$

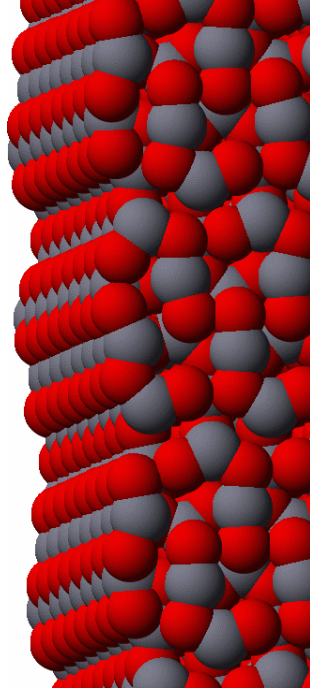
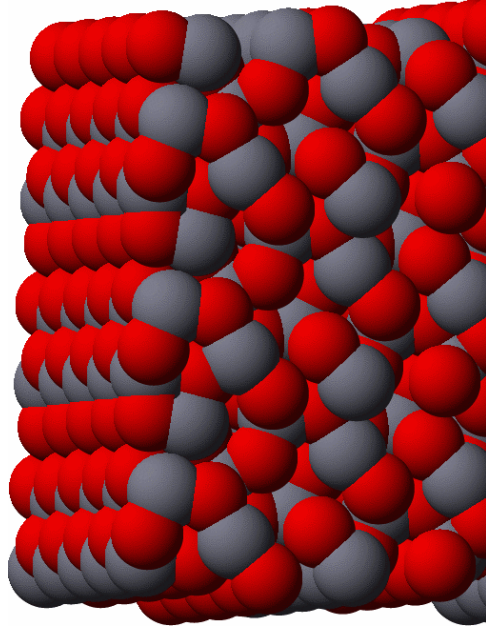


# oxides

Structure of the complicated oxide surfaces ( $\text{Al}_2\text{O}_3$ )

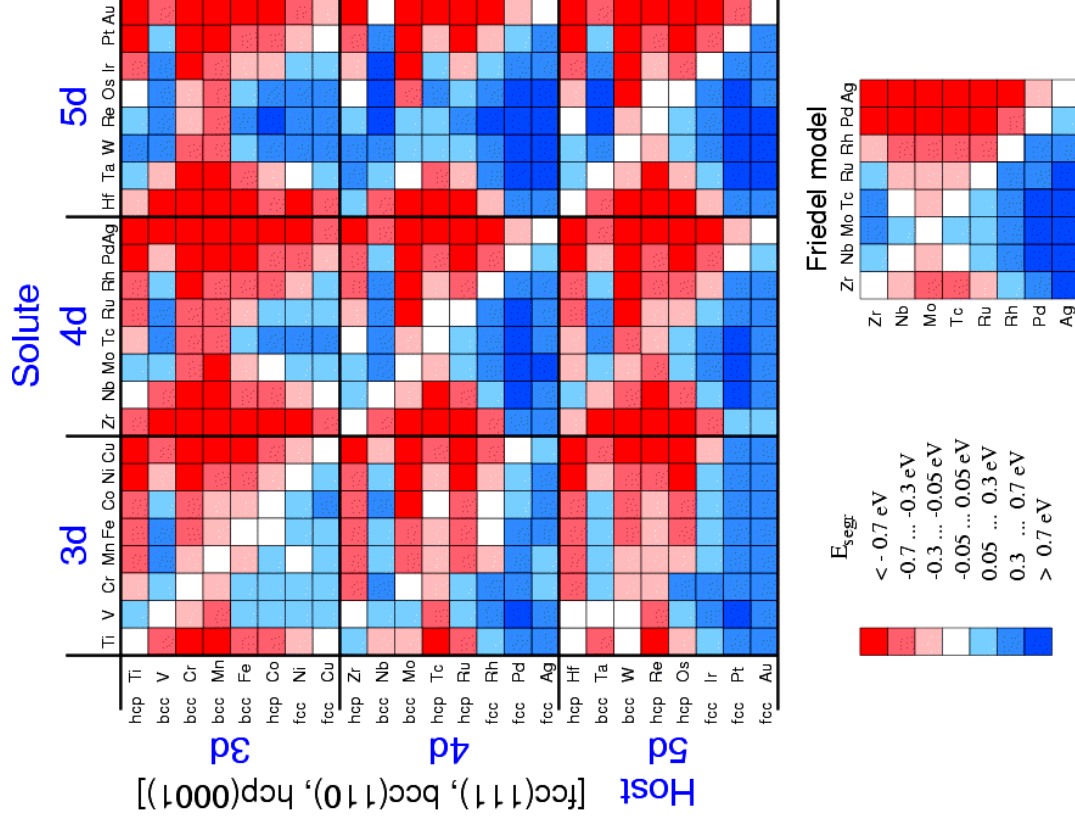
$\text{Al}_2\text{O}_3$  possesses surface area  $\sim 300\text{m}^2/\text{g}$   
what are the surface properties ?

	<i>Surface energy (<math>\text{J}/\text{m}^2</math>)</i>	<i>Surface cations</i>
(100)	0.73	$\text{Al}^{\text{O}-1}$
(110)	1.10	$\text{Al}^{\text{O}-1}; \text{Al}^{\text{T}}$
(111)	1.70	$\text{Al}^{\text{T}}$



# Alloys

## The segregation energy



LMTO approach