Python
Pour le
calcul Scientifique

Konrad HINSEN

Centre de Biophysique Moléculaire (Orléans)

et

Synchrotron Soleil (St Aubin)

Une brève historique

1991: Python est publié

1994: premières applications scientifiques

1996: Numerical Python

.

.

2006: - un grand choix de bibliothèques

- colloque annuel SciPy

- premier livre épuisé au bout de deux mois

- enseigné à plusieurs universités

- des entreprises spécialisés

- une formation du CNRS attire la foule

Quelques applications

Astronomie

ASTROLIB
et PyFITS
(Space Telescope Science Institute)

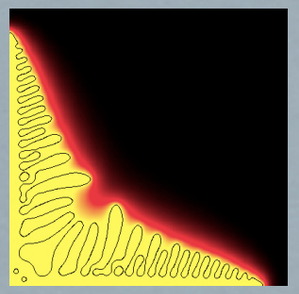
Neurologie

Vision Egg
(Collaboration internationale)

Bioinformatique

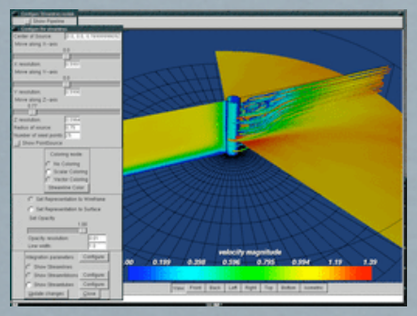
BioPython
(Collaboration internationale)

Eléments finis



FiPy
(NIST)

Visualisation



MayaVi
(Prabhu Ramachandran)

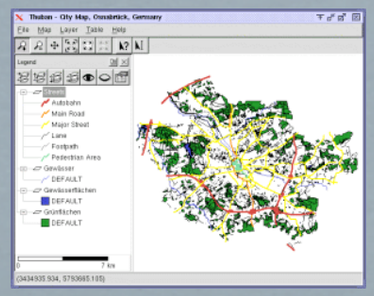
Statistique

Modular toolkit for Data Processing
(Humboldt-Universität)

Systèmes dynamiques

SimPy
(Collaboration internationale)

Géographie



Thuban
(Intevation GmbH)

Mathématiques

SAGE
(University of Washington)

Pourquoi Python ?

- Travail interactif
  - Développement rapide
  - Développement incrémentiel
  - Test, débogage
  - Analyse interactif de données

Pourquoi Python ?

- Travail interactif
- Simplicité du langage
  - Syntaxe claire et nette
  - Gestion automatique de la mémoire
  - Tout est dynamique,
    il n'y a rien à déclarer

Pourquoi Python ?

- Travail interactif
- Simplicité du langage
- Orientation objet
  - Structuration du programme en unités qui représentent un aspect du problème à résoudre
  - Facilite les modifications et les extensions

→ Démonstration

Pourquoi Python ?

- Travail interactif
- Simplicité du langage
- Orientation objet
- Ouverture au monde
  - Facile à interfacer avec le C/C++ et le Fortran
  - Facile à interfacer avec d'autres programmes
  - Excellente portabilité

Pourquoi Python ?

- Travail interactif
- Simplicité du langage
- Orientation objet
- Ouverture au monde
- Disponibilité de bibliothèques
  - Calcul scientifique
  - Lecture/écriture de données
  - Internet
  - Interfaces graphiques
  - ...

Pourquoi Python ?

- Travail interactif
- Simplicité du langage
- Orientation objet
- Ouverture au monde
- Disponibilité de bibliothèques
- Ça fait plaisir !

Scénarios d'usage

Langage de script

- Lire/écrire des fichiers
  ➥ perl, awk, grep, vi, emacs, ...
- Analyse de données, visualisation
  ➥ Matlab/Scilab/Octave, IDL, R
- Gestion de tâches de calcul
  ➥ sh/bash, csh
- Administration système
  ➥ sh/bash, csh, grep, awk, perl, ...

Avantages de Python:

☺ vrai langage de programmation
☺ bibliothèques utiles de qualité

Calcul exploratoire

- Analyse de données
- Visualisation

➥ Scripts simples et travail interactif

Outils pratiques:

→ IPython
→ Emacs + Python mode
→ matplotlib
→ VPython
→ Module pickle

➥ Démonstration

Calcul exploratoire parallèle

L'analyse de données contient souvent beaucoup de parallélisme évident...

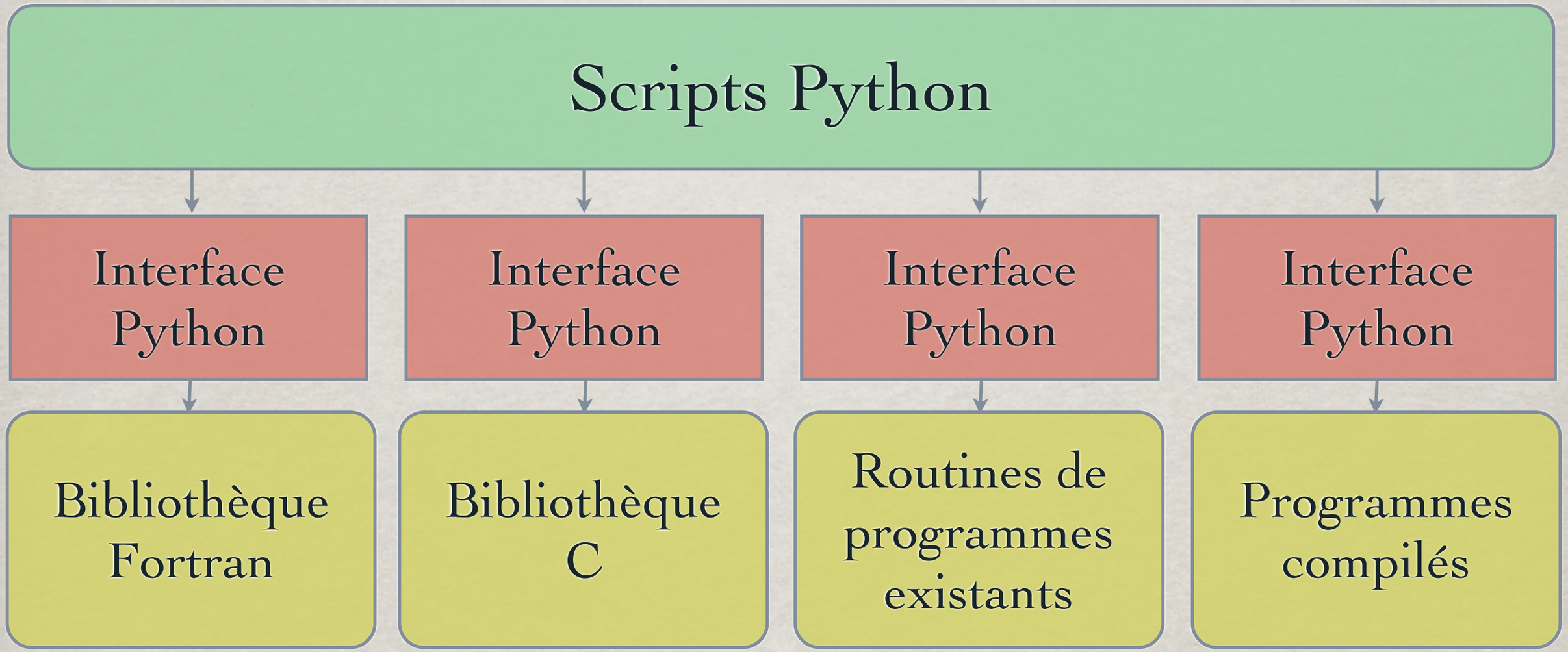
... mais pas facile à exploiter.

Python vous aide:

- Echange d'objets arbitraires entre processeurs
- Plusieurs bibliothèques pour gérer le parallélisme:
  - → Scientific.DistributedComputing
  - → Pypar/PyMPI
  - → Scientific.BSP

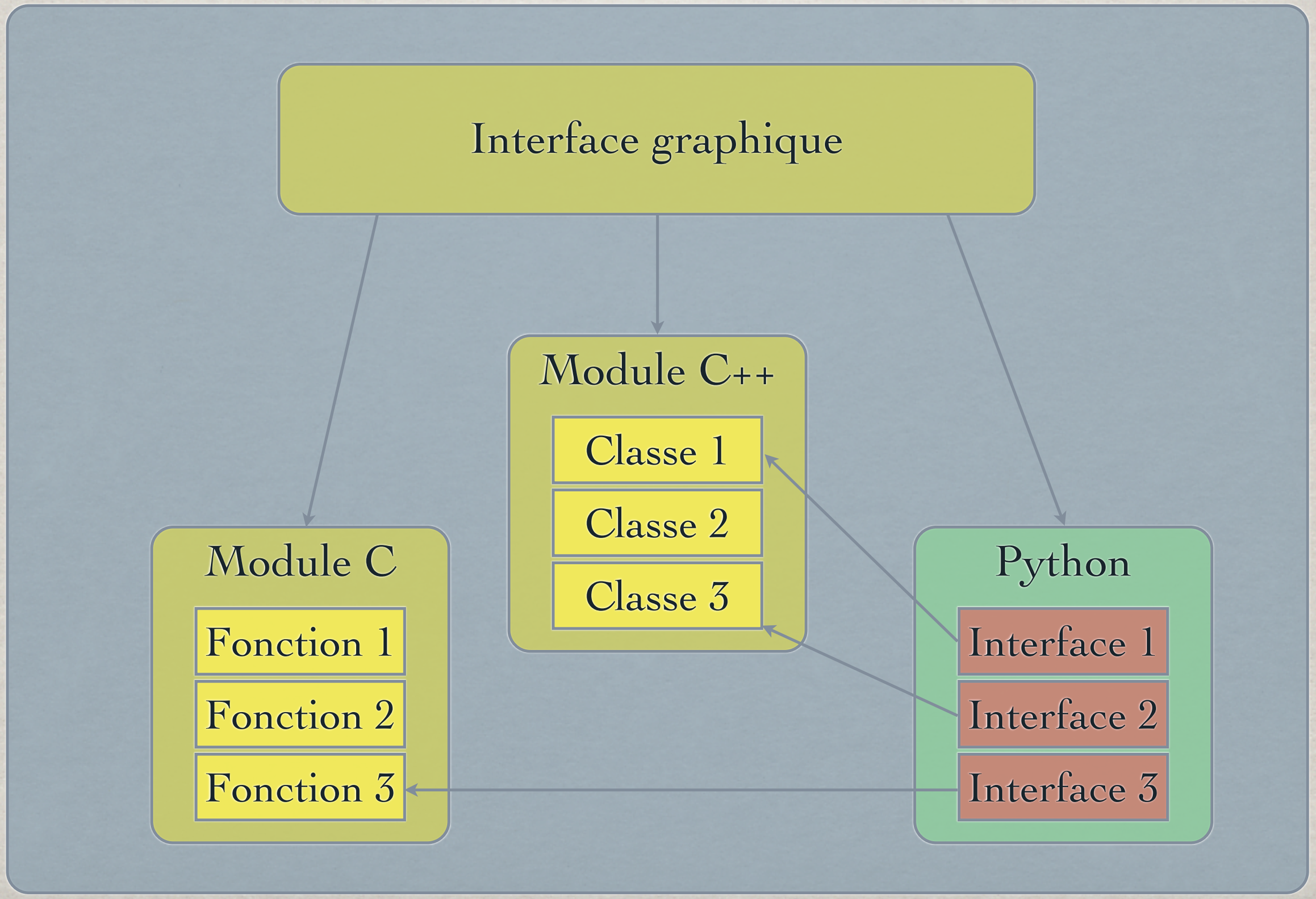
➥ Démonstration

Langage d'intégration

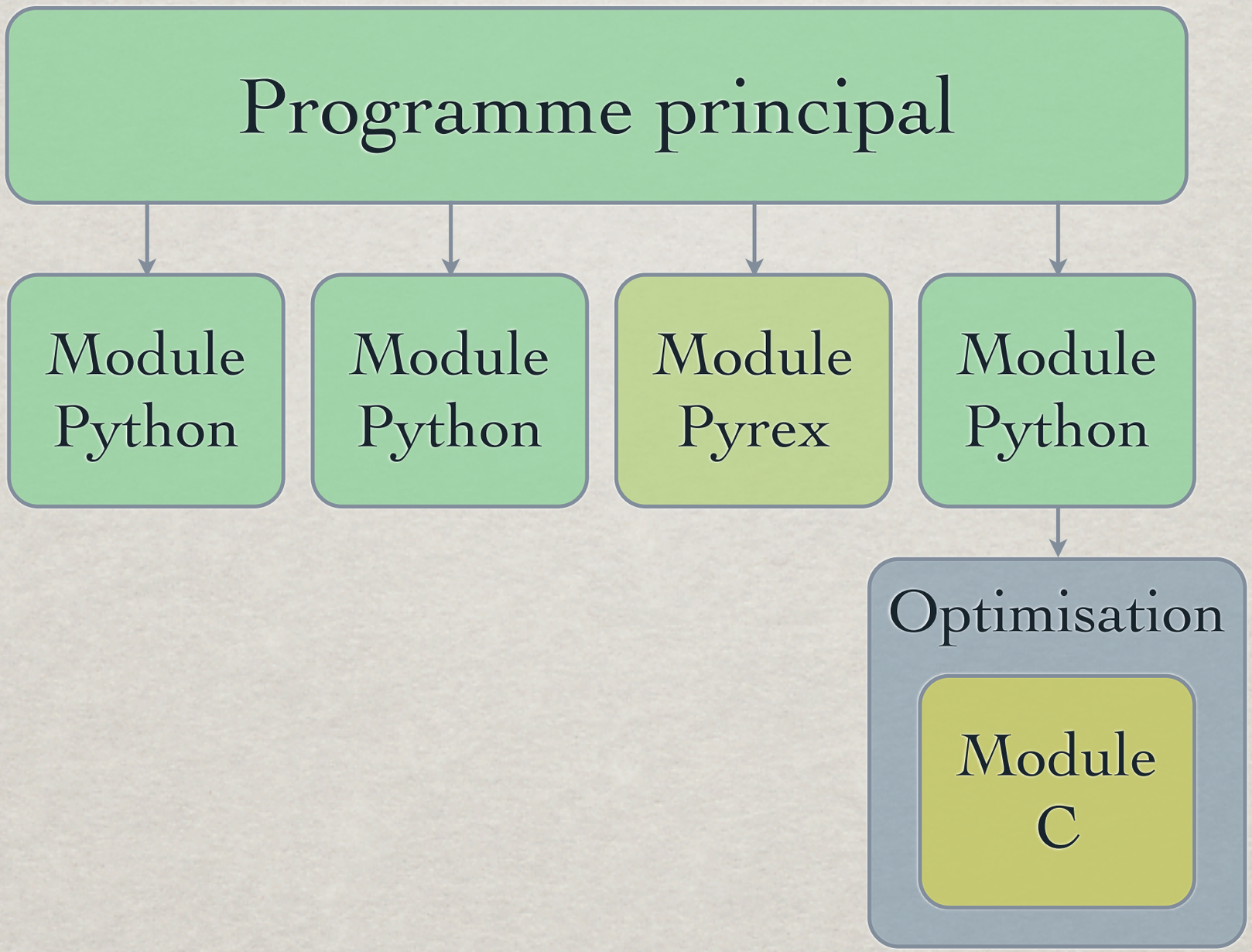


Outils: swig, boost, f2py, PyFort, Pyrex

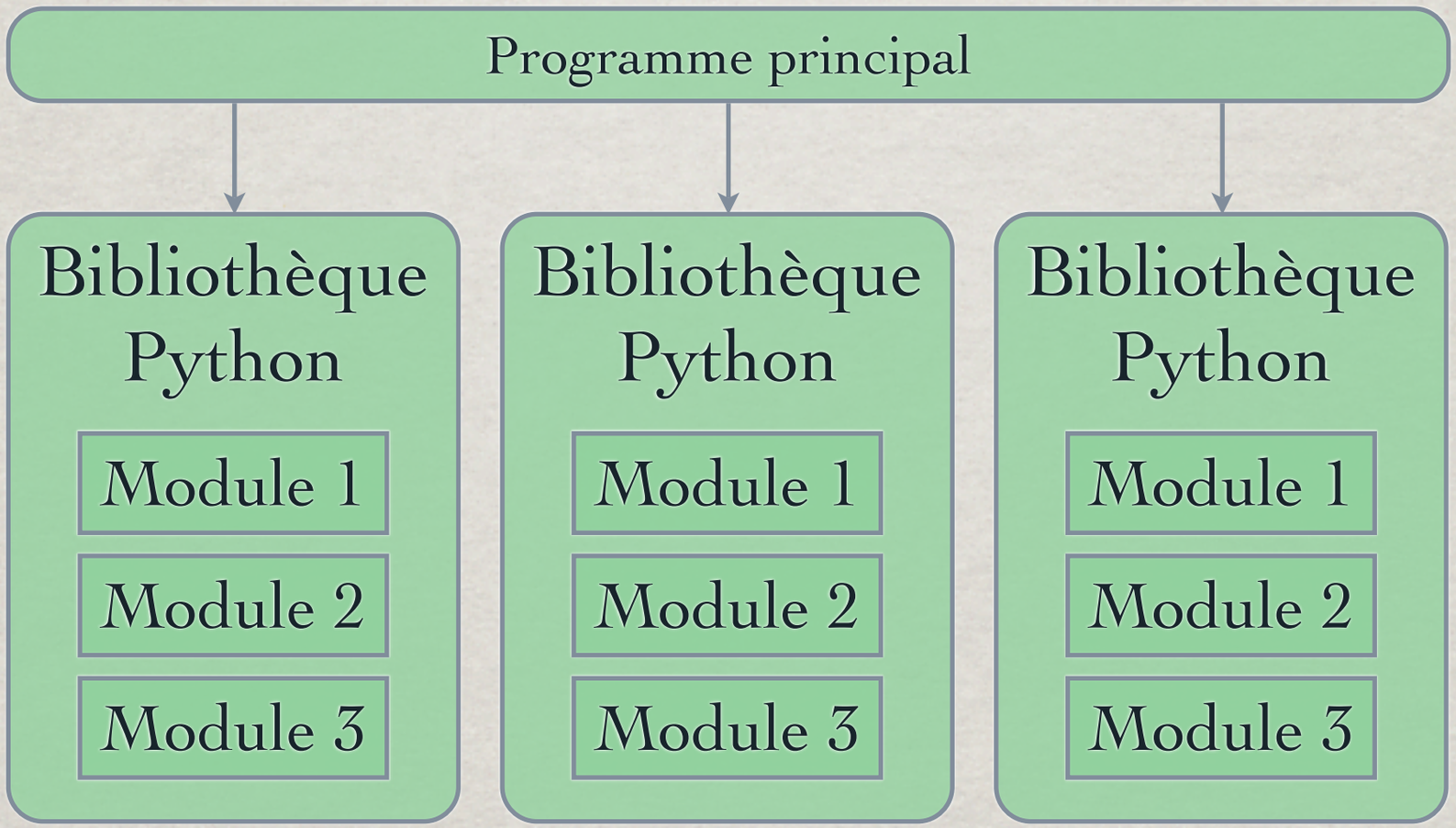
Langage de script intégré



Langage principal

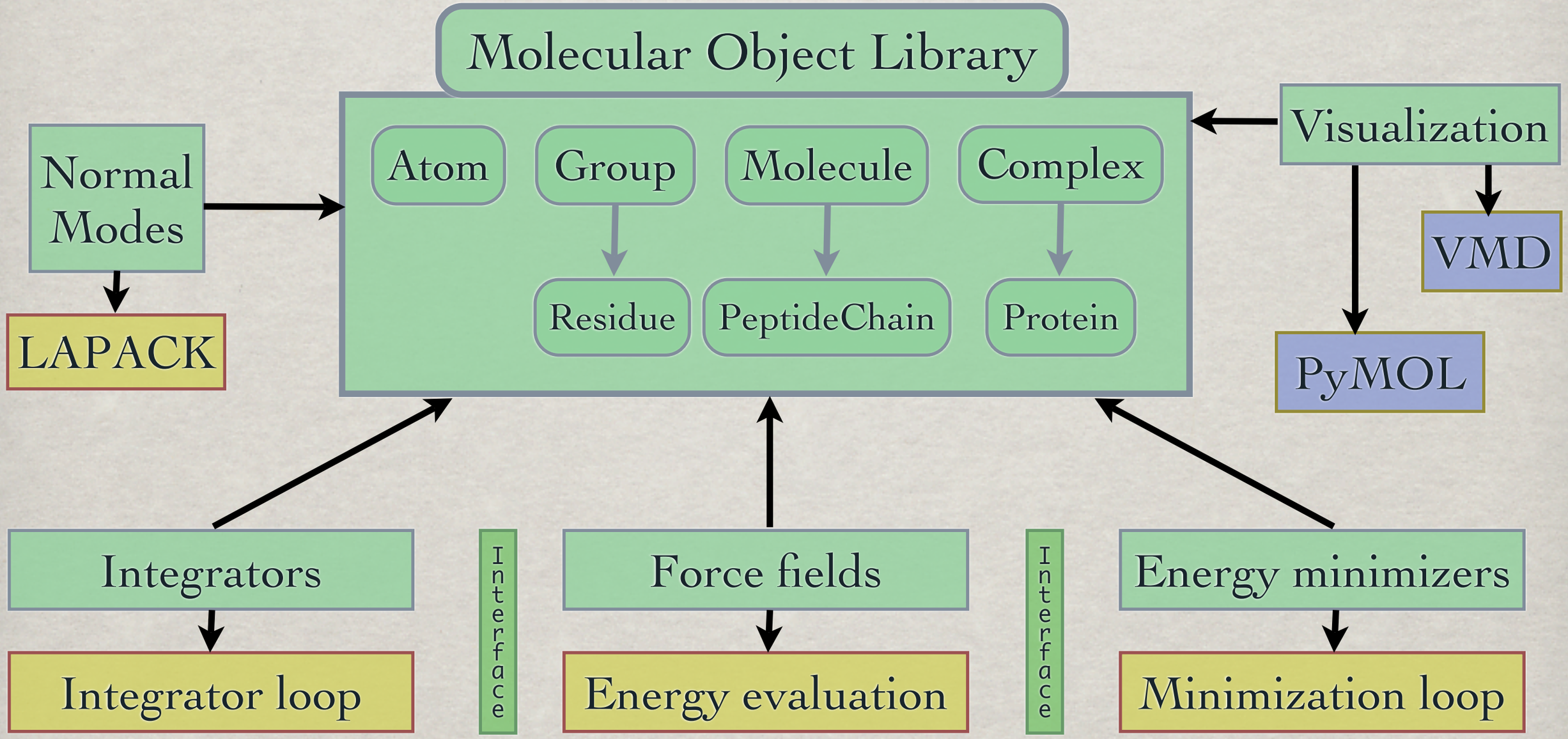


Pensez Bibliothèques !



Une bibliothèque est plus utile que des routines cachées dans un programme !

Molecular Modelling Toolkit

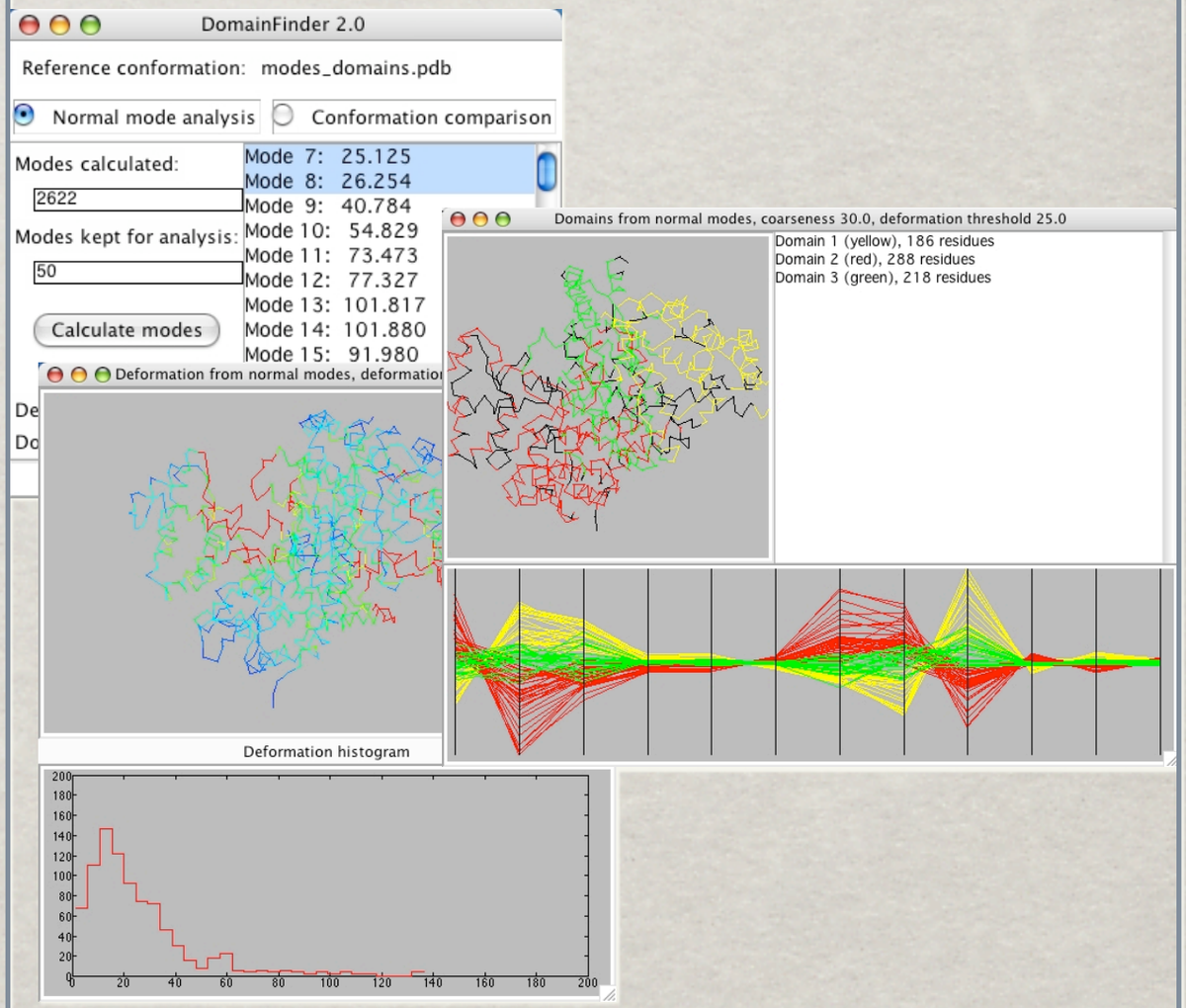


Utiliser MMTK...

Scripts

```
# Standard normal mode calculation.
#

from MMTK import *
from MMTK.Proteins import Protein
from MMTK.ForceFields import Amber99ForceField
from MMTK.NormalModes import VibrationalModes
from MMTK.Minimization import ConjugateGradientMinimizer
from MMTK.Trajectory import StandardLogOutput
from MMTK.Visualization import view

# Construct system
universe = InfiniteUniverse(Amber94ForceField())
universe.protein = Protein('bala1')

# Minimize
minimizer = ConjugateGradientMinimizer(universe,
                                       actions=[StandardLogOutput(50)])
minimizer(convergence = 1.e-3, steps = 10000)

# Calculate normal modes
modes = VibrationalModes(universe)

# Show animation of the first non-trivial mode
view(modes[6])
```

Interfaces graphiques



Performance

Python n'est pas rapide ... mais :

- Ce n'est pas toujours vrai.
  Certains aspects de Python ont été optimisés à fond.
- Il y a des modules performants en C/C++/Fortran.
  → Numeric
- C'est le programmeur qui devient plus efficace.
  Ce qu'il faut optimiser, c'est le temps jusqu'au bout du projet.

Conseils :

- Ecrivez votre programme en Python d'abord.
- Si c'est assez rapide, soyez contents.
- Sinon optimisez les parties critiques (et rien d'autre)

"Premature optimization is the root of all evil" C.A.R. Hoare/D. Knuth

Optimisation

1) Ne devinez pas quelles sont les parties critiques:
   `profile` (ou `cProfile` en Python 2.5) vous le dit.
   python -m profile -s time mon_script.py

2) Travaillez sur les algorithmes.
   Passer de $O(N^3)$ à $O(N)$ apporte plus que passer de Python à C.

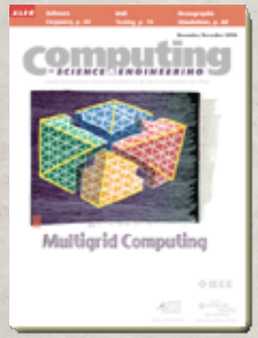
3) Cherchez des modules optimisés adéquates.

4) Cherchez des bibliothèques adaptées à interfacer.

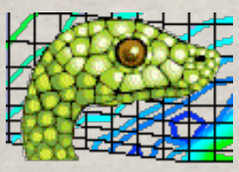
5) Tournez vers :
   - Pyrex
   - C / Swig
   - C++ / Swig
   - C++ / Boost
   - Fortran / f2py

Pour continuer...

Hans-Petter Langtangen
Python Scripting for Computational Science
Springer, 2005/2006



Computing in Science and Engineering
Special Issue “Python: Batteries included”
May/June 2007



Club des **U**tilisateurs de **P**ython **S**cientifique à Paris
https://www.logilab.net/cups