
# Z-SeT / ZéBuLoN

- *Fonctionnement de la communauté de développement*
- *Architecture du code*
- *Exemple de développements*
- *Discussion*

# Z–SeT : quelques points de repère (1)

2001 : 4 institutions dans un *groupe de développeurs*

Mines de Paris/CDM, ONERA, NWNumerics, Mines de St Etienne

- 1981 Premières versions de ZéBuLoN (Apple II)
- 1984 Version unique, 2D-3D, plasticité
- 1988 Version Unix/Sun, post-processeur graphique, mécanique de la rupture, thermique
- début des années 90 Déploiement tout matériel, post-traitement, mailleur interactif, lois de comportement

# Z–SeT : quelques points de repère (2)

- 1991-95 Première version C++
- 1996 Première version parallèle, ONERA
- 1996 NorthWest Numerics
- 1997 Optimiseur, diffusion
- 1999 Nouvelle interface utilisateur, coque
- 2000 Dynamique, explicite

# Z-SeT : le gang...

# Les objectifs

- Synergie de développement dans les recherches
- Développement d’une plate–forme généraliste
  - ⋆ Eléments finis
  - ⋆ Intégration de systèmes différentiels
  - ⋆ Optimisation
- Approches modulaires pour problèmes couplés

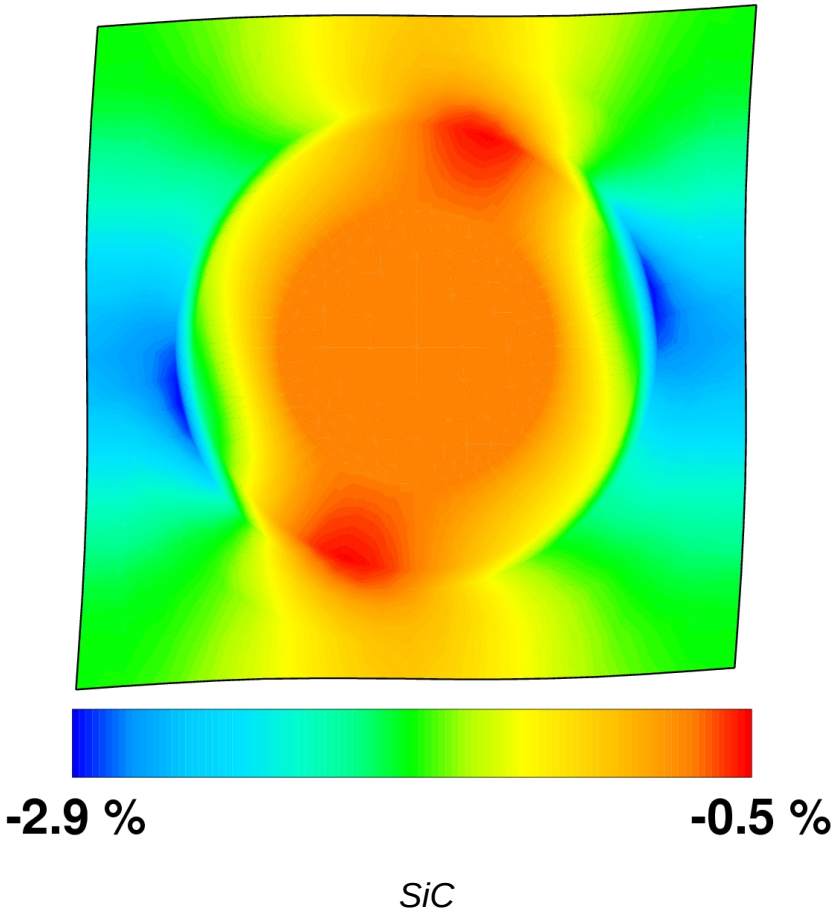
# Les domaines d'activité

- Problèmes à grand nombre de variables internes modélisation fines, multiéchelle
- Problèmes à grand nombre de degrés de liberté
  - ⋆ Géométrie des pièces industrielles
  - ⋆ Microstructures réelles calcul de microstructures
- Problèmes couplés pour approches multiphysiques

# Multi–échelle

- Modèles auto–cohérents, conduisant à quelques milliers de variables internes par point de Gauss

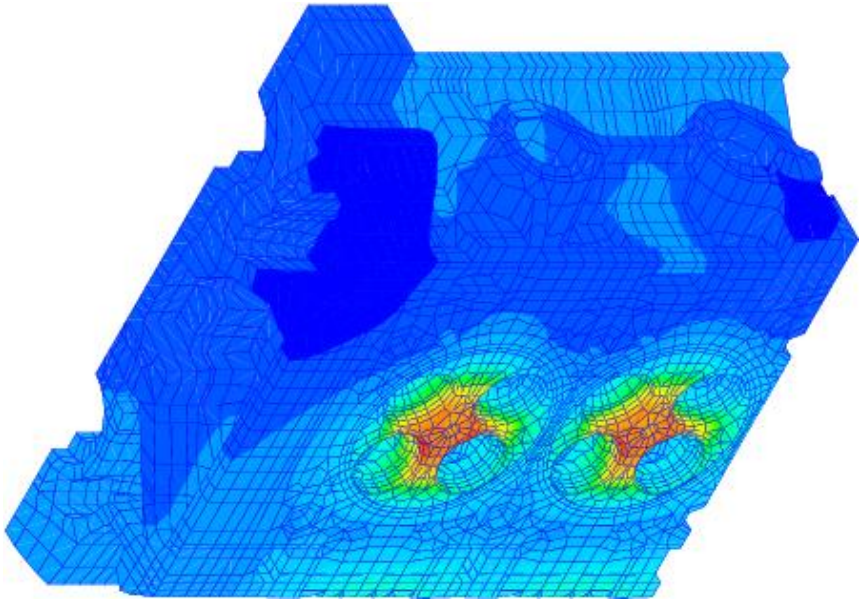
- $EF^2$, utilisant un maillage $EF$ en chaque point de Gauss




*objectifs*

# Pièces industrielles

- Calculs dans la zone $10^5$–$10^6$ degrés de liberté en non–linéaire
- Calculs cycliques, durée de vie. . .

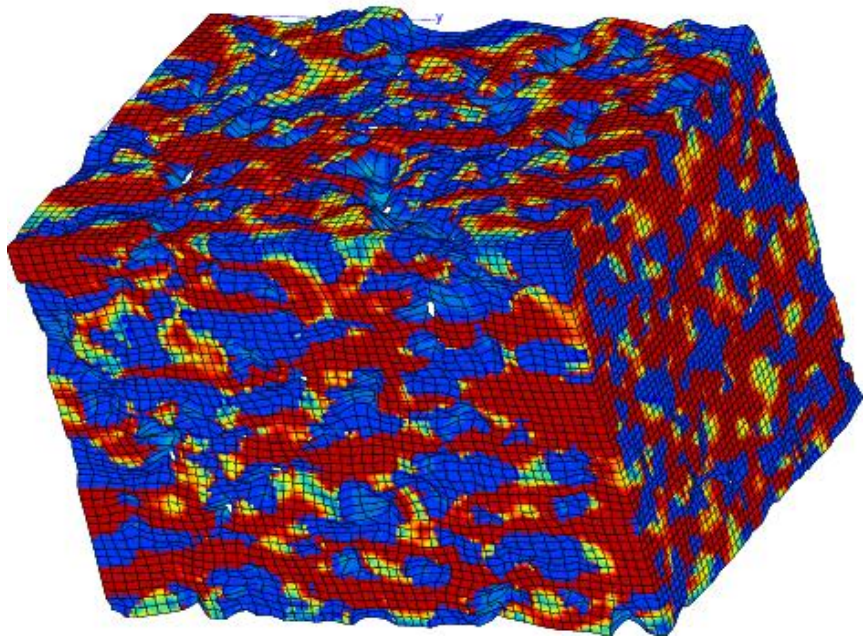


*Culasse diesel*

*objectifs*

# Calcul de microstructures

- Matériaux multiphasés, homogénéisation
- Grains, systèmes de glissement

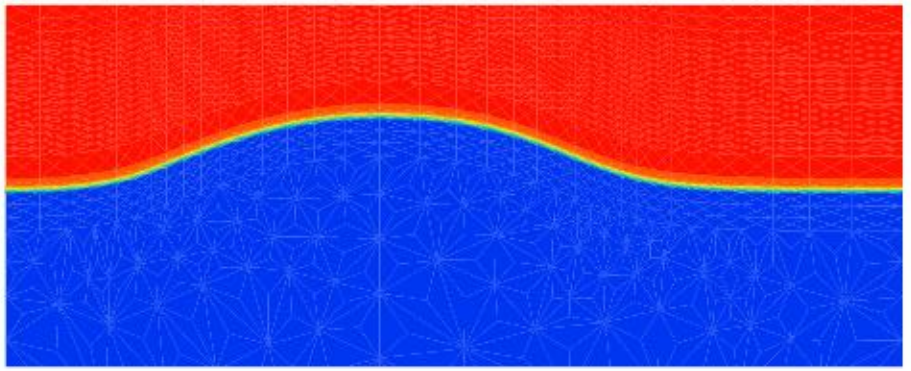


*Matériau biphasé de l'industrie alimentaire*

*objectifs*

# Problèmes couplés

- Thermique
- Mécanique
- Endommagement
- Diffusion



*Oxydation d'une barrière thermique*

*objectifs*

# Z-SeT : les modules

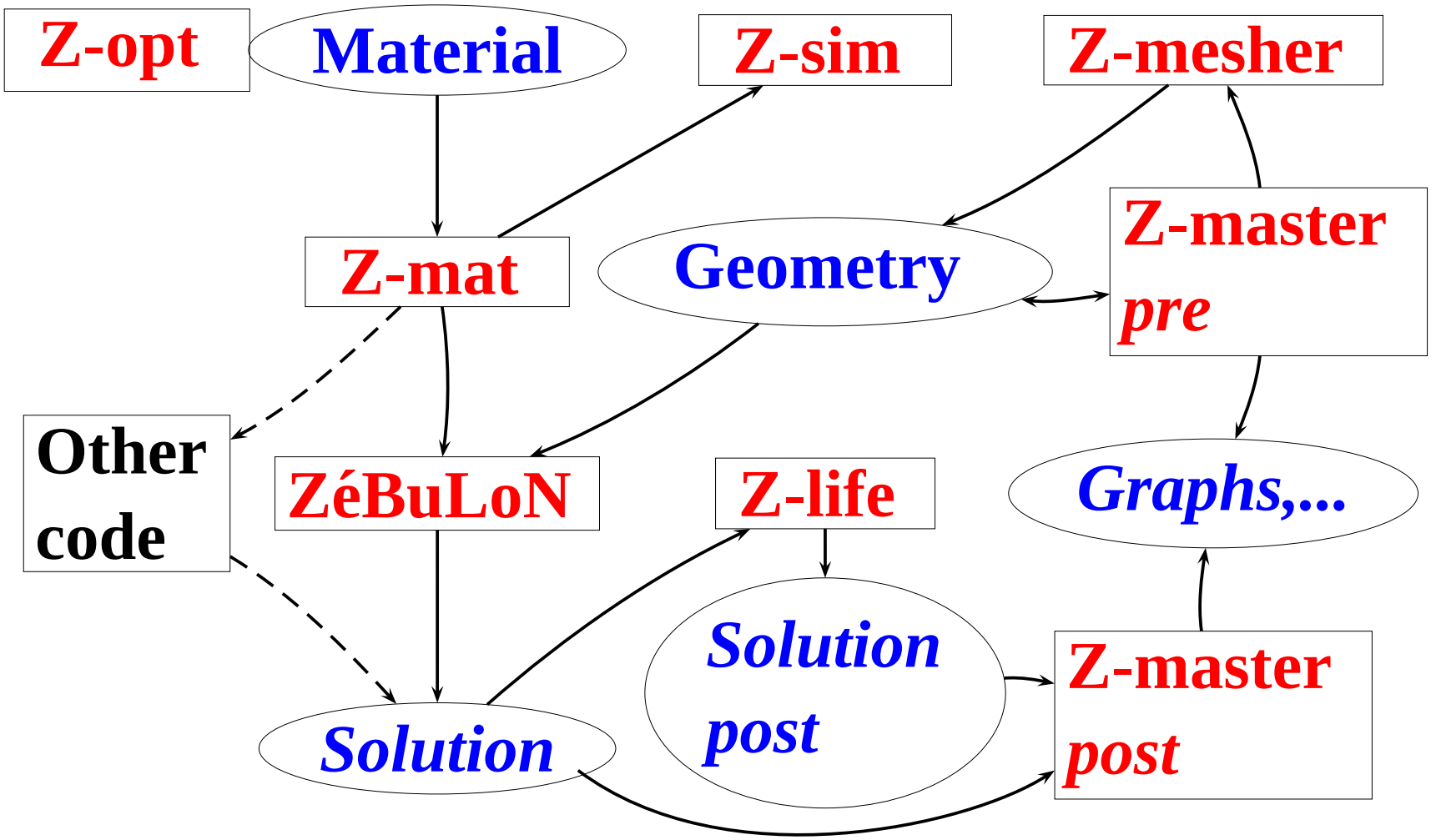
- *ZéBuLoN*, solveur séquentiel et parallèle
- *Z-master*, interface utilisateur, pré-post graphique
- *Z-mesher*, manipulation de maillages
- *Z-sim*, driver lois de comportement
- *Z-opt*, optimiseur généraliste
- *Z-life*, post-processeur d'endommagement, etc...

# Z-SeT : les modules (suite)

- *ZeBFRoNT*, générateur de code pour loi de comportement

- *Z-MAT*, librairie matériaux interfacée avec les grands codes commerciaux de mécanique des structures

# *Z-SET* modules



# Parallélisme

- ONERA : coopération avec l'équipe de calcul intensif (F.-X. Roux), développements depuis 95
- Mines : participation au Pôle parallélisme Ile-de-France Sud, développement et utilisation IBM SP2 (1997), cluster de 48 PC (2000)

# Parallélisme au CDM

- 48 PC, Pentium III-800 MHz, 768 Mo, 30 Go, + interface
- 3 à 4 thésards en "production" sur la machine

# ZéBuLoN ∗

- Type d'analyse
  - ⋆ Mécanique
    - ∗ Statique, dynamique
    - ∗ Non–linéaire matériau
    - ∗ Non–linéaire géométrique
    - ∗ Lagrangien total et réactualisé
    - ∗ Contact-Frottement
    - ∗ Analyse modale
  - ⋆ Thermique
  - ⋆ Diffusion

- Bibliothèque d'éléments
  - ⋆ Volumiques : 2D, 2.5D, 3D
  - ⋆ Poutres, coques
  - ⋆ Milieux de Cosserat

- Algorithmes
  - ⋆ Solveurs frontal, multi–frontal, itératif, parallèle
  - ⋆ Intégration par θ–méthode ou Runge–Kutta

# Parallèle

- Parallélisation SPMD
  - ⋆ Bibliothèque PVM ou MPI
  - ⋆ Mémoire partagée ou distribuée
  - ⋆ Parallélisme à forte granularité
  - ⋆ Scalabilité sur plusieurs dizaines de processeurs
- Méthode de sous-domaines
  - ⋆ Parallélisation de l'étape globale : FETI
  - ⋆ Parallélisation de l'étape locale
  - ⋆ Valide quel que soit le comportement

# Librairie matériaux * **

- Nombreux modèles
  - Elasticité, hyperélasticité
  - Plasticité, viscoplasticité
  - Ecrouissages isotrope linéaire et non-linéaire
  - Plasticité cristallographique
  - Modèles ductiles
  - Visco-élasticité
  - user

- Nombreux outils
  - Fabrication de modèles *multipotentiels* dans le fichier de données
  - Dépendance illimitée des coefficients des variables internes et paramètres extérieurs
  - *ZeBFRoNT*, générateur de code pour matériau utilisateur

# Z-mat

- Interface entre la librairie matériau et les grands codes commerciaux
  - ⋆ Abaqus
  - ⋆ Ansys
  - ⋆ Marc
  - ⋆ Cosmos

- Possibilités
  - ⋆ Même code pour identification et calcul EF
  - ⋆ Pas automatique
  - ⋆ Matrice tangente cohérente

# Zmaster : maillage 2D et post graphique

- Pré–processeur
  - Géométrie 2D
  - Maillage réglé et libre
  - triangles, quadrilatères
  - Interfaces avec Ideas, Abaqus

- Post–traitement
  - Visualisation en face cachées
  - Sélection de vues en coupe
  - Déformée
  - Isovaleurs sur la structure déformée
  - Tracé de courbes
  - Animation

# Zmesher

- Manipulation de maillage
  - ⋆ Extensions 3D
  - ⋆ extrusions, rotations
  - ⋆ Passage linéaire-quadratique
  - ⋆ Création de groupes
  - ⋆ éléments, faces, lignes,...
  - ⋆ Fusions, collages
  - ⋆ Renumérotation
- Découpage en sous–domaines
  - ⋆ Splitmesh
  - ⋆ Metis

# Z-sim ∗

- Objet
  - ⋆ Driver lois de comportement
  - ⋆ chargements 2D–3D en contrainte et déformation
  - ⋆ Utilisation pour tout système différentiel
- Possibilités
  - ⋆ Recherche de surfaces de charge
  - ⋆ Pilotage de la librairie matériaux
  - ⋆ Asservissements complexes

# Z-post/Z-life ∗

- Objet
  - ⋆ Zones de traitement (nœuds, pts de Gauss, incréments)
  - ⋆ Traitement par des fonctions
  - ⋆ Processus enchaînés
  - ⋆ Listes formattées
  - ⋆ Post–traitement graphique
  - ⋆ user

- Post- global
  - ⋆ Moyennes
  - ⋆ Weibull
  - ⋆ user

- Post- local
  - ⋆ fluage
  - ⋆ fatigue HCF, LCF
  - ⋆ rupture ductile
  - ⋆ user

# Z-opt

- Caractéristiques
  - ⋆ Interface élémentaire : ?coef
  - ⋆ Simulation extérieure
  - ⋆ Optimisation sous contrainte
  - ⋆ Méthodes de gradient :
  - ⋆ SQP, Levenberg–Marquardt
  - ⋆ Méthodes heuristiques :
  - ⋆ Simplexe, Algorithmes évolutionnaires

- Identification de ...
  - ⋆ Coefficients matériaux
  - ⋆ Coefficients d'échanges
  - ⋆ Coefficients de frottement
  - ⋆ Optimisation de forme

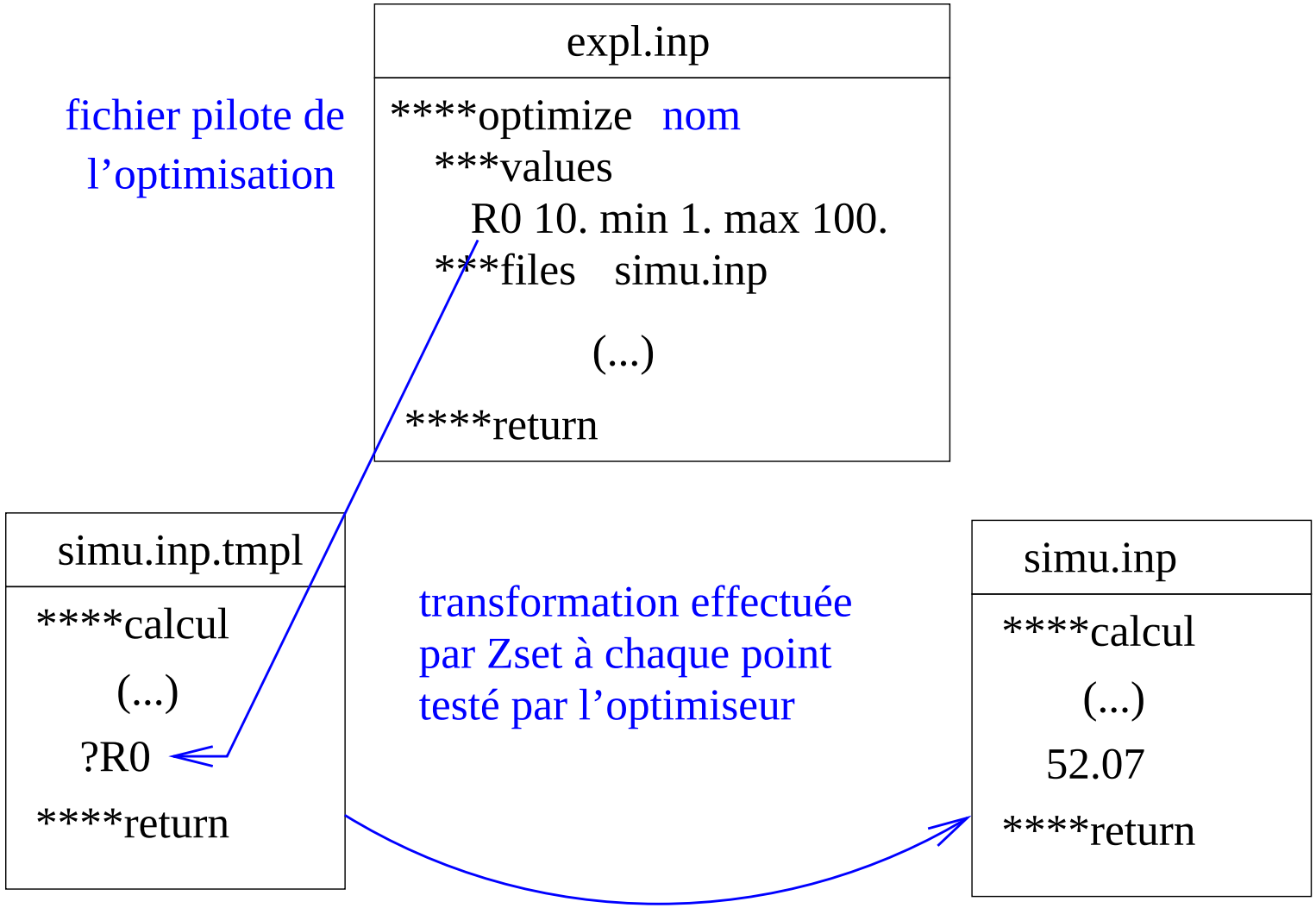
# Optimisation dans ZéBuLoN

→ Z-opt

# Z-opt

- Interface générale entre simulation et optimiseurs (template).
- Famille d'algorithmes couvrant beaucoup de problèmes d'optimisation.
- Possibilités de Z-opt :
  - ⋆ identification des coefficients,
  - ⋆ optimiser des fonctions analytiques,
  - ⋆ optimiser des géométries de pièces.
- Identification sur structure possible.

# Z-opt : interface par template

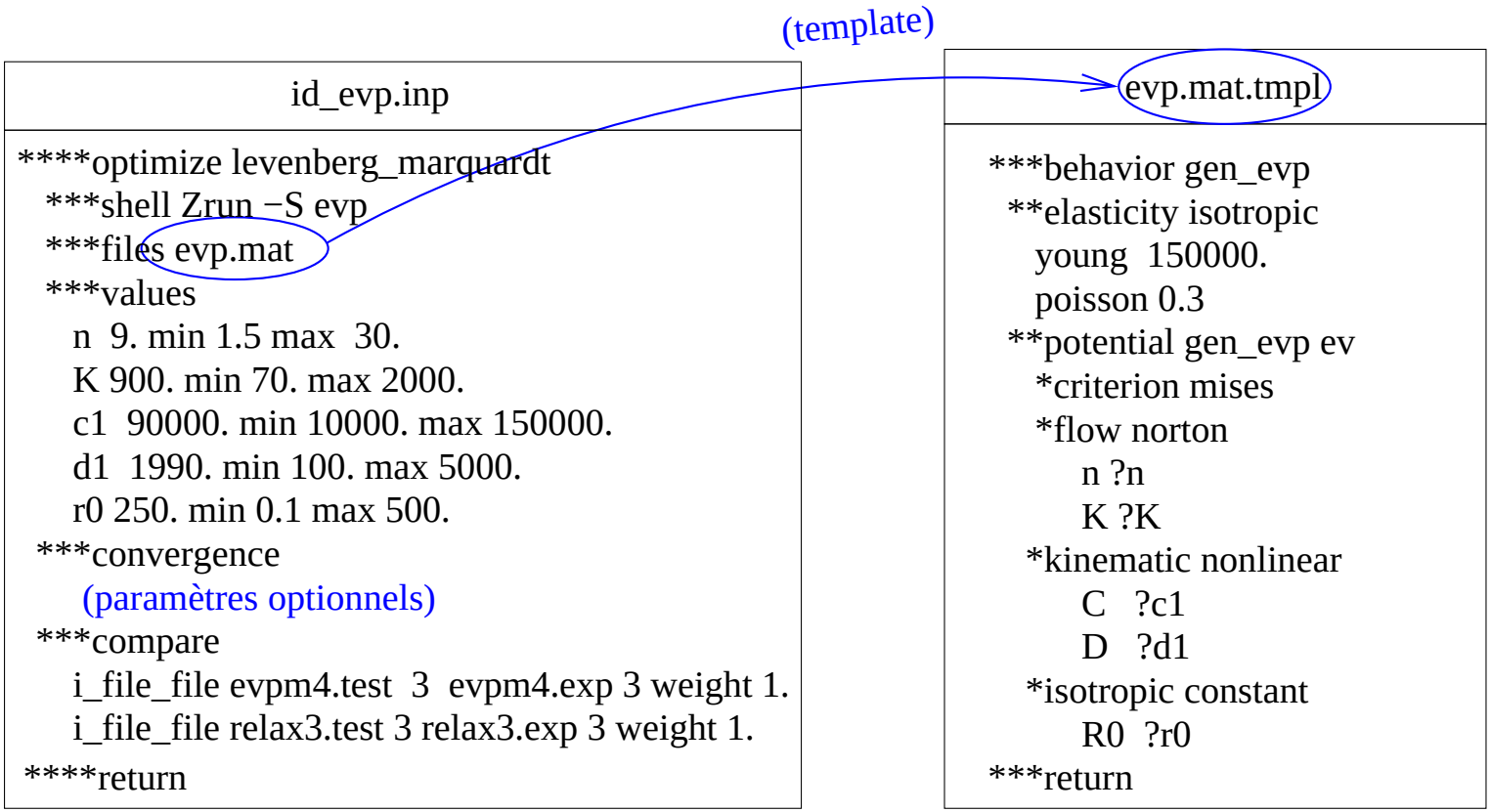


# Z-opt : méthodes

Z-Optim couvre les principales méthodes d'optimisation: avec ou sans contraintes d'optimisation, locale ou globale, avec ou sans gradient de la fonction coût.

- Nelder-Mead (simplex) : sans contraintes, locale, sans gradient.
- Levenberg-Marquardt : sans contraintes, locale, avec gradient, adaptée aux moindres carrés.
- SQP (Lagrangien projeté) : avec contraintes, locale, avec gradient.
- calcul évolutionnaire : avec contraintes, globale, sans gradient.

# Z-opt : exemple d'identification de loi EVP

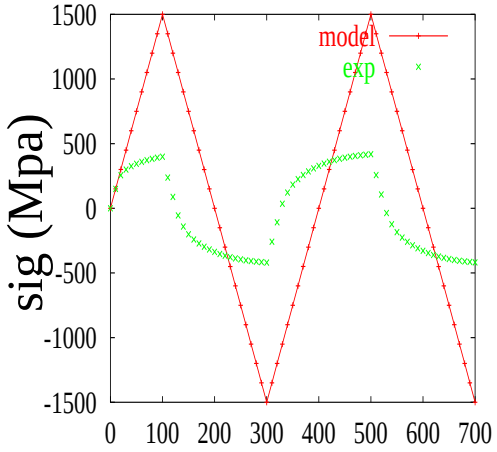


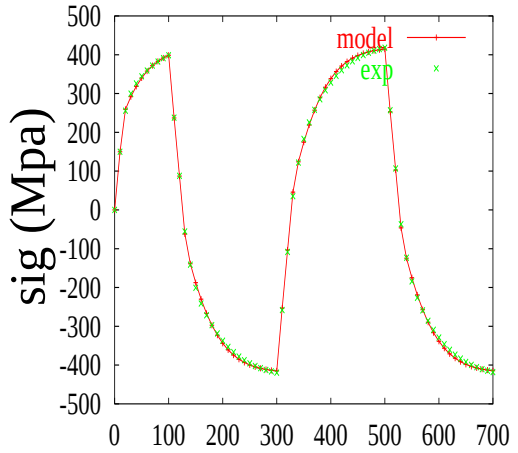
# Z-opt : exemple d'identification de loi EVP

départ

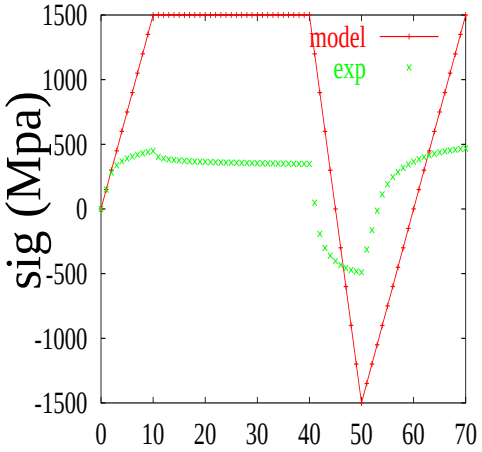
100 itérations LVM

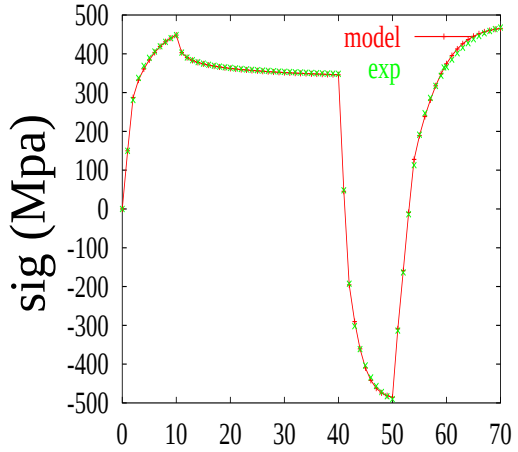
fatigue



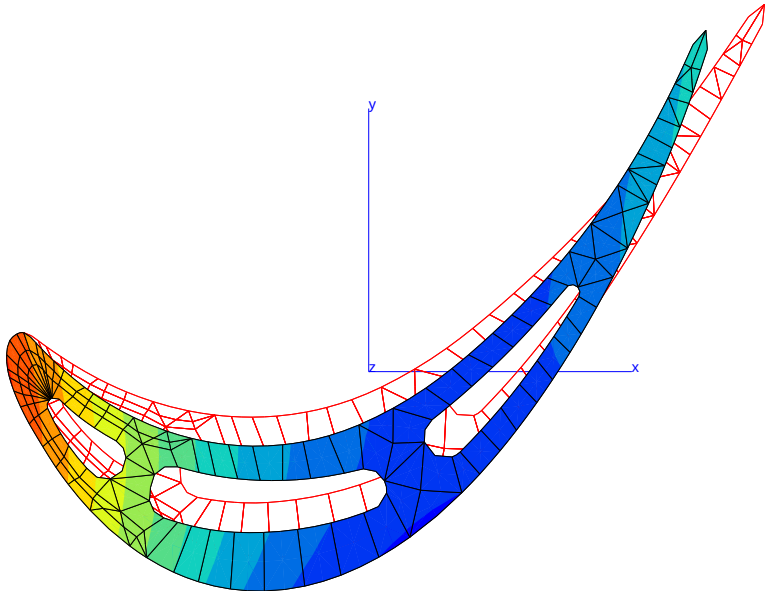

relaxation



# Z-opt : exemple d’identification sur structure

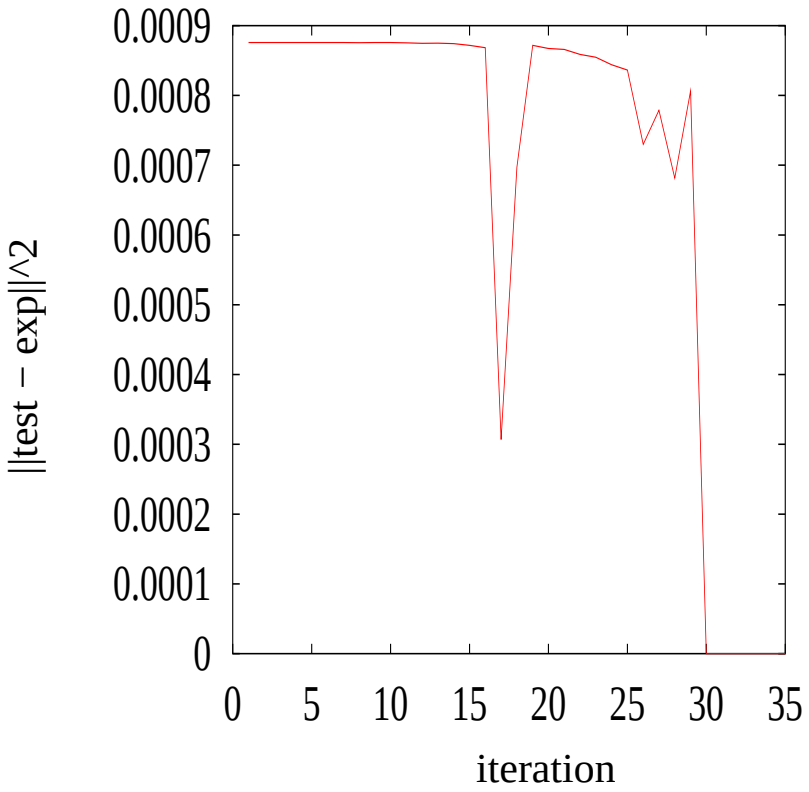
Identification de la loi de comportement d’un monocristal à partir de mesures de déplacements sur une aube.



# Z-opt : exemple d'identification sur structure

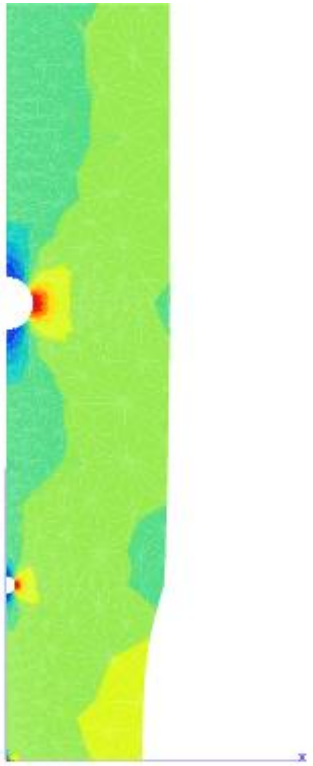
aube_auto.inp.tmpl

```
****calcul
   ***mesh 2_5D
   ***resolution newton
         ...
   ***bc
         ...
   ***output
     **test depl.test
       *node_var 316 U1 U2
         ...
****return

***behavior gen_evp auto_step
    **elasticity cubic
          y1111 ?y1111
          y1122  ?y1122
          y1212   ?y1212
          ...
***return
```



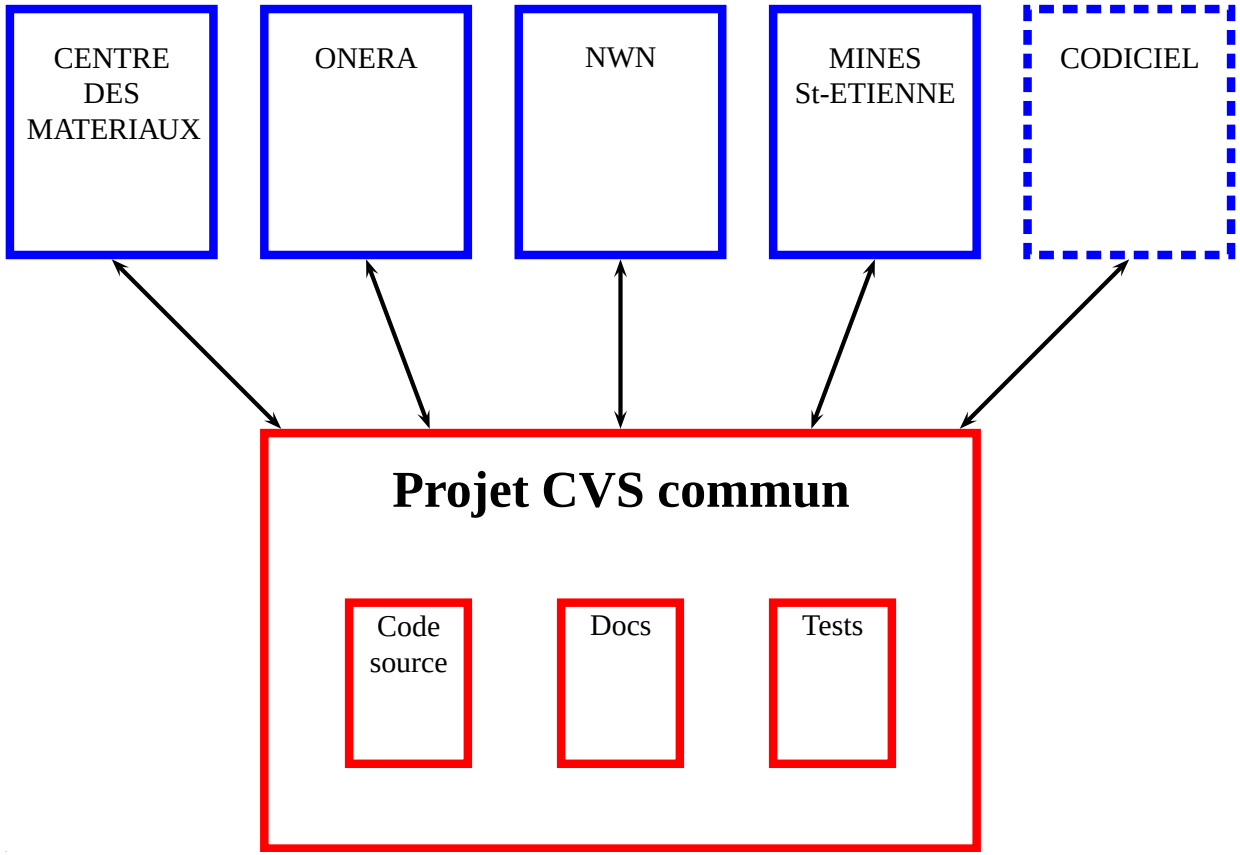
# Z-opt : exemple d'optimisation de forme

- Forme externe: splines d'ordre 2.
- Maillage : ZLanguage.
- EF non linéaire.
- Dépouillement: Z-post + ZLanguage.
- Optimisation: Levenberg-Marquardt.

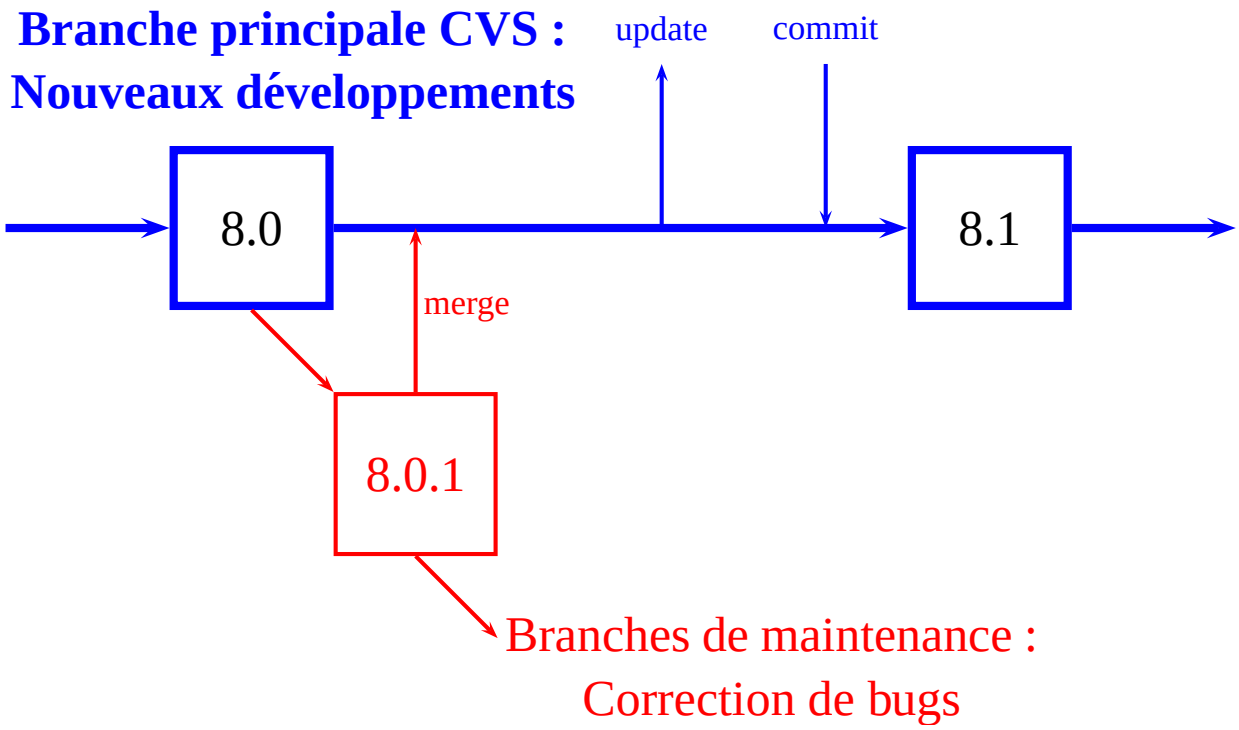


# Fonctionnement de la communauté de développement ZéBuLoN

# Plusieurs sites de développement, une seule version du code



# Gestion des mises à jour




Ajout d'une nouvelle fonctionnalité:

Mise à jour du code source, de la doc et des tests par CVS

# Base de tests

- 1061 tests
- Procédure automatique (Zrun -test)

```
$ ls Z8.2/TESTS
00_CPU_TESTS        Dynamic_test        Mesher_test         Program_test        Thermal_test
00_Changes          Eigen_test          Mooney_test         RK_test             Viscoelastic_test
00_INPUTS           Energy_test         Mooney_visco_test   Rainflow_test       Viscous_fluid_test
FE2_test            Multimat2_test      Reduced_test        Volume_test         CVS
Harmonic_test       Multimat_test       Rivlin_test         Contact_test        Input_output_test
Optimizer_test      Rivlin_visco_test   Z-mat               Coupled_test        Jint_test
Parallel_test       Damage_test         Kinematic_test      Parks_test          Shell_test
ZebFront_test       Debound_test        LCF_test            Polycrystal_test    Simulator_test
LS_test             Porous_test         Solver_test         Diffusion_test      Post_test
Static_test
```

# Contrôle qualité

Chaque nuit:

- Récupération par CVS de l'ensemble de la distribution
- Compilation complète
- Run automatique de la base de tests
- Mail avec résumé des résultats envoyé automatiquement aux développeurs

# La release Zébulon

```
$ ls Z8.2
00_README      HANDBOOK      Plugins       ZebFront      depend
CVS            Makefile      TESTS         bin           include
Extras         PUBLIC        User-project  calcul        lib
```

# Le code source

```
$ ls Z8.2/calcul
Zebulon_cpp_Linux    zCoupling          zMaster              zParallel          zUtilityMesh
library_files        zDiffusion         zMechanical          zPost              zViscousFluid
o_Linux              zFE_in_FE          zMechanicalBehavior  zProgram           zZeBaBa
og_Linux             zFemGeom           zMesher              zServer            zZebulon
zAnisoDamage         zFemProblem        zMotif               zShell             zZfrontBehavior
zBase                zGenEvp            zMultimat            zSimulator
zBehavior            zGraphics          zOpenGL              zThermal
zClient              zLicense           zOptimizer           zTools
```

# Documentation

## Manuels

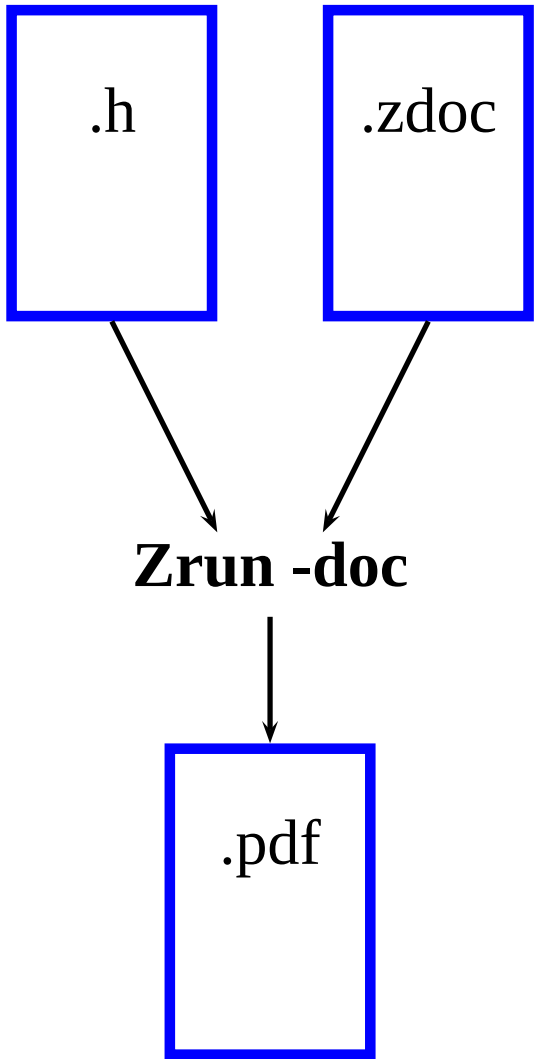
- User manual
- Zmat user manual
- Examples manual
- Developpers manual
- Auto devel manual

## Outils

- Ecriture en latex
- Macros (uniformisation)
- Compilation par "makefile"
- Fichier pdf avec liens hypertexte

# Documentation développeur automatique

- Génération automatique
- Exploitation des fichiers .h:
  - ⋆ Définition des classes
  - ⋆ Données et fonctions membres
  - ⋆ Liens hypertexte pour héritage
- Exploitation des fichiers .zdoc:
  - ⋆ Ecriture en latex
  - ⋆ Description de la classe



# Architecture du code

# Architecture

Aujourd'hui, ZéBuLoN c'est :

- 300 000 lignes de code C++
- plus de 1 000 classes

$\longrightarrow$ simplifications. . .

# Architecture

deux ”principes” de programmation : { Object Factory / Plugins ; objets composites }

+ une librairie "boite à outils" : `zTools`

# Object factory : but

Modularité

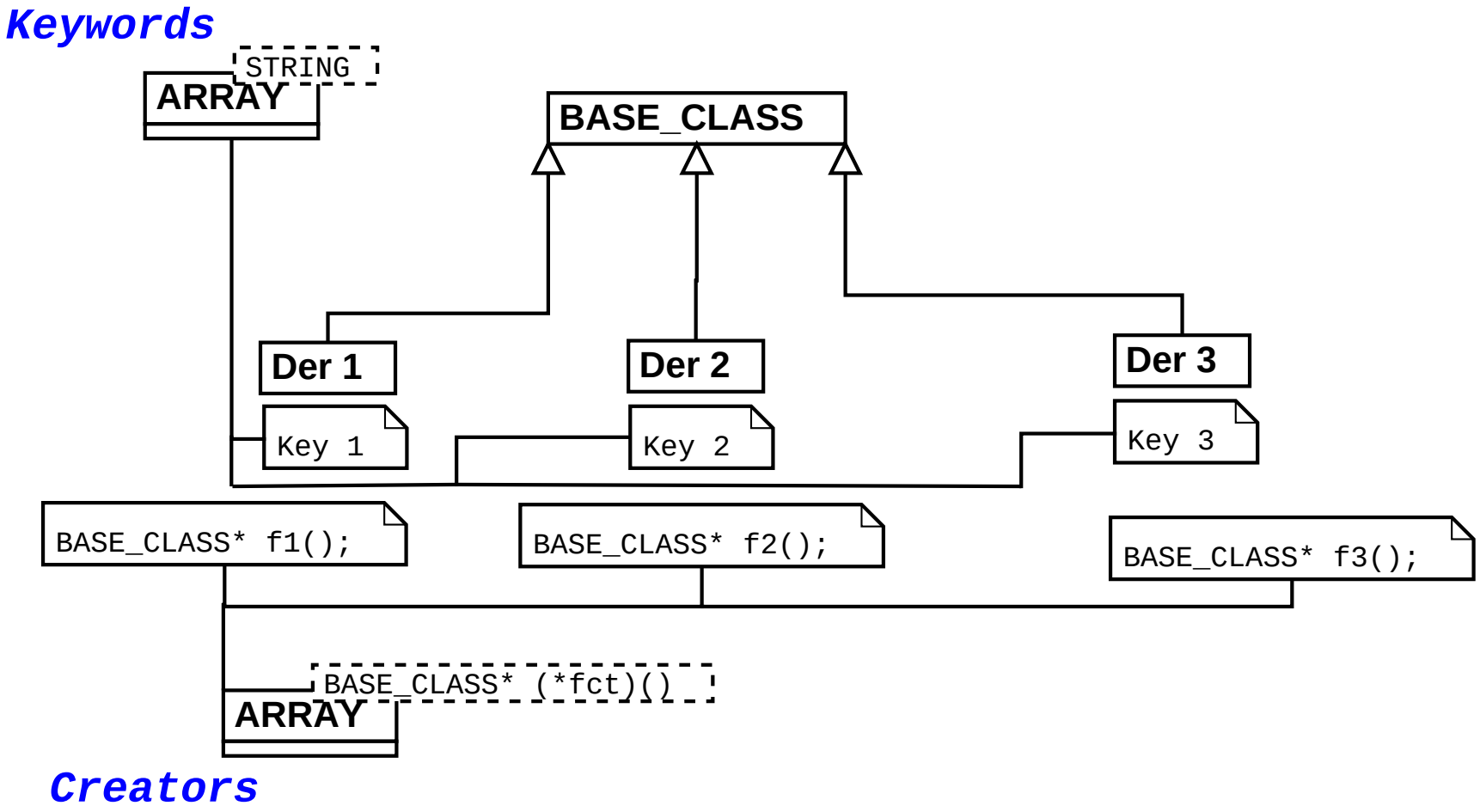
## A éviter. . .

```
BEHAVIOR *b;

if(ctl=="linear_elastic")
  b=new LINEAR_ELASTIC_BEHAVIOR();
else if(ctl=="plastic")
  b=new PLASTIC_BEHAVIOR();
else
  ERROR("Unknown behavior named : "+ctl);
```

## . . . Solution

```
BEHAVIOR *b;

b=Create_object(BEHAVIOR,ctl);
if(b=NULL)
  ERROR("Unknown behavior named : "+ctl);
```

# Object factory : structure




```
Create_object(BASE_CLASS,key)
```

# Object Factory : enregistrement et utilisation

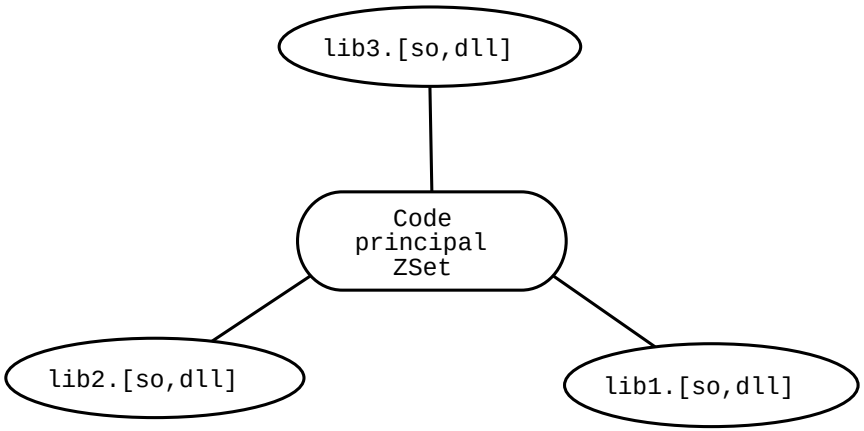
```
class A {
};

class B : public A {
};

class C : public A {
};
```

```
DECLARE_OBJECT(A,B,plastic);

DECLARE_OBJECT(A,C,elastic);

.........

A *a;

a=Create_object(A,"plastic");
a=Create_object(A,"elastic");
```

⟶ Massivement utilisé dans le code

# Plugins :

→ mécanisme d'enrichissement du code




- librairies dynamiques
- chargées à l'exécution
- activation automatique de l'object factory (*substitution*)

# Objets composites

## Dérivation :

```
#include <Mc.h>
#include <Axi.h>
#include <8r.h>
#include <ELastic.h>

class MC_AXI_8R_ELASTIC :
  public MC , public AXI,
  public 8R, public ELASTIC
{
...
}

...
  else if(ctl=="my_behavior")
    b=new MY_BEHAVIOR

...
```

→ enrichissement *intrusif*

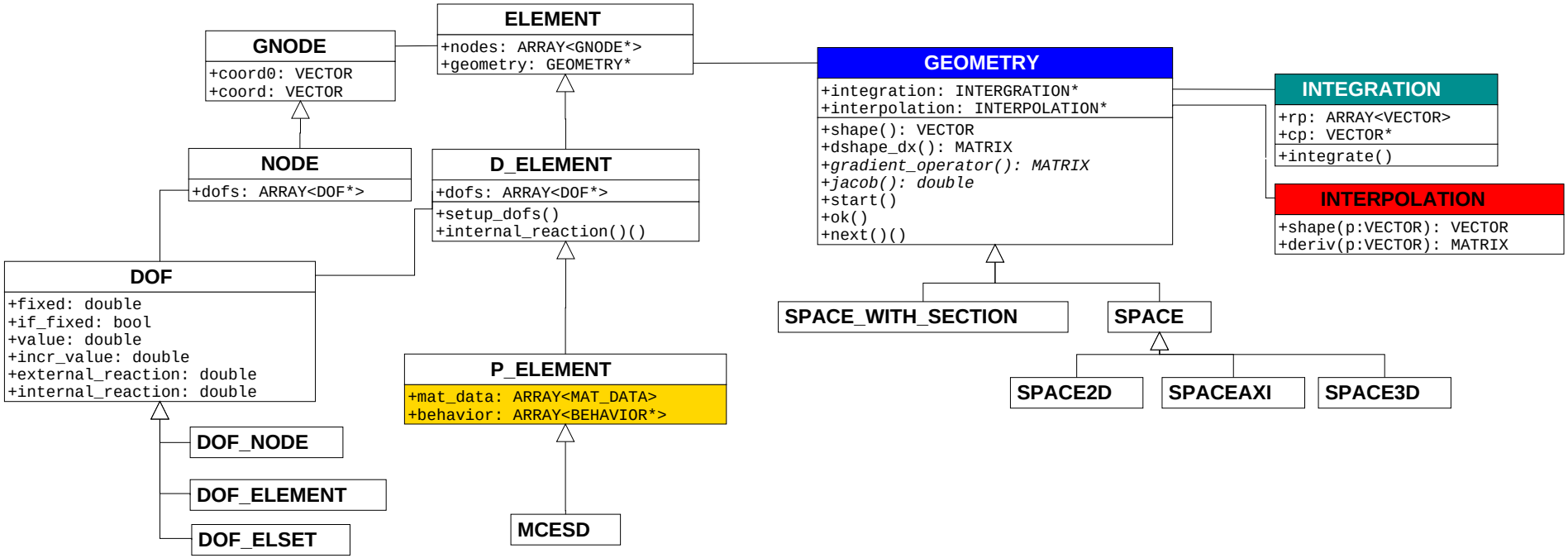
## Composition :

```
class ELEMENT
{
  INTERPOLATION *inter;
  INTEGRATION *integ;
  BEHAVIOR *beh;
}
```

# Objets composites

| | | |
|---|---|---|
| **D** | 50 classes de base | $\left.\begin{array}{c}\\ \\ \\ \end{array}\right\} \longrightarrow \infty$ de combinaisons |
| **d** | 1000 classes ”publiques” (`.h`) | |
| **ND** | ?X classes composites | |

# Classes : éléments finis

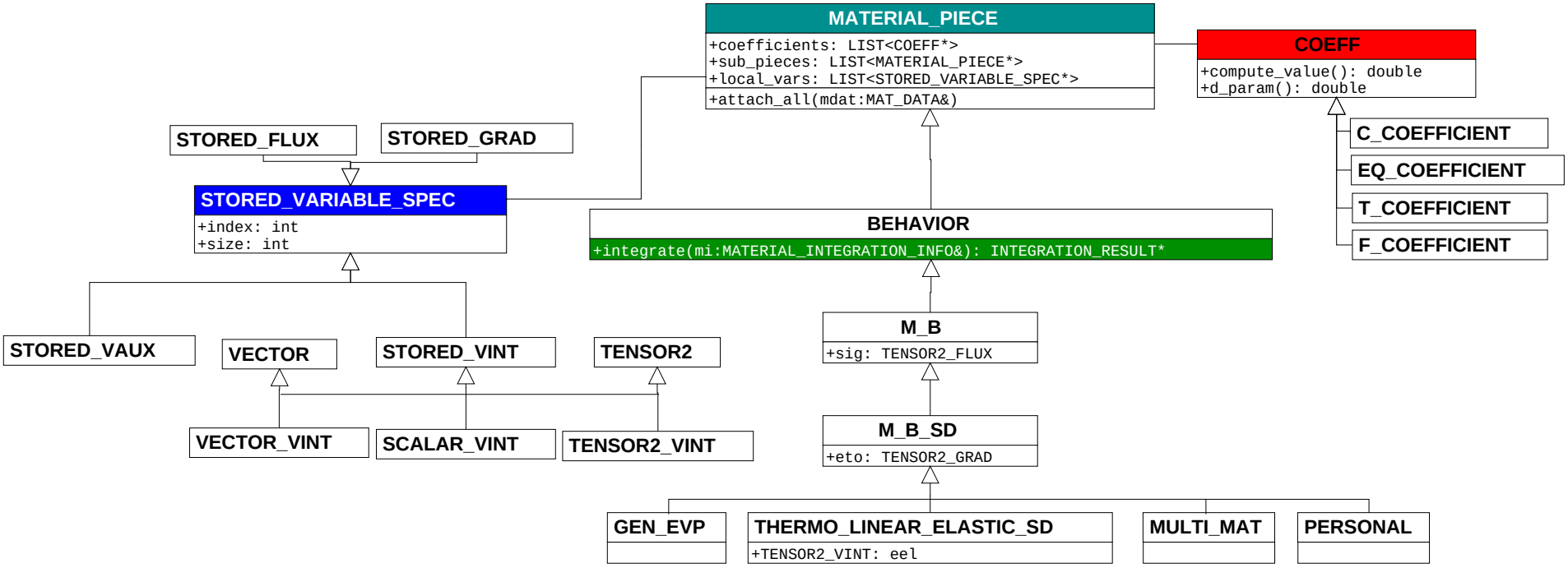


# Classes : comportements

Un comportement, c'est :

- des variables (internes, auxiliaires, in, out. . .)
- des coefficients
- des paramètres extérieurs (température, humidité)
- des méthodes d'intégration

# Classes : comportements



# zTools

- classes utilitaires (chaînes, tableaux, listes,. . . )
- classes mathématiques (vecteurs, tenseurs,. . . )
- des algorithmes généraux (intégration, solveurs itératifs,. . . )

$\longrightarrow$ choix délibéré de ne pas utiliser la *STL*

# Développements dans ZéBuLoN

→ ZLanguage

→ ZFront

→ C++

# ZLanguage : concept

- langage interprété
- sous-ensemble de C/C++
- fonctionnement en deux passes : «compilation» puis exécution
- peut être appelé n'importe où dans le code

- interpréteur / compilateur
- machine virtuelle
- ensemble de classes interface avec les classes réelles

# ZLanguage : atouts

- classes «wrapper» : exportabilité
- extensibilité : ajout de classes «wrapper» triviale
- processus pouvant être automatisé

- interpréteur / machine virtuelle : une classe C++
- utilisation très facile

# ZLanguage : applications

ZLanguage est aujourd'hui lié

- aux conditions aux limites ($\longrightarrow$ exemple)
- au post-processing ($\longrightarrow$ exemple)
- au pré-post graphique / mailleur
- aux éléments

utilisation possible en «découplé» : langage de script généraliste associé à la «boîte à outils ZéBuLoN».

# ZLanguage : exemple de conditions aux limites

```
void initialize()
{
  global POINTER set;
  set=mesh.find_nset("top-left");
}

void update()
{
  global POINTER set;
  int i;
  VECTOR pos;
  double topd;

  for(i=0;i<!(set.object);i=i+1) {
    pos=set.object[i].coord0();
    set.object[i].get_dof("U2").if_fixed=1;
    set.object[i].get_dof("U2").fixed=0.;
  }
}
```

```
****calcul
  ***resolution newton
    **sequence
      *time 1.
      *increment 3
  ***bc
    **impose_nodal_dof
      charge U2 1.0 tab1
    **z7p arm2_bc.z7p
  ***material
      *file mat0
  ***table
    **name tab1
      *time 0. 1.
      *value 0. 1.
****return
```

# ZLanguage : exemple de critère de post-processing

```
void initialize() {
  global double max_mises;
  max_mises=-MAX_DOUBLE;
}

void destroy() {
  global double max_mises;
  cout<<"Max mises = "<<max_mises<<endl;
  cflush();
}

void compute() {
  TENSOR2 sigma;
  global double max_mises;
  double mises;

  sigma.reassign(4,in,0); mises=sigma.mises();
  out.resize(1); out[0]=mises;
  if(out[0]>max_mises) max_mises=out[0];
}
```

```
****post_processing
  ***local_post_processing
    **output_number 1-3
    **elset ALL_ELEMENT
    **file integ
    **process z7p arm2_loc.z7p
****return
```

# ZebFront : concept

- Pré-processeur
  - ⋆ Réutilisation des classes de base
- Produit un fichier C++

`ZebFront fichier.z` $\longrightarrow$ `fichier.c`

- `fichier.c` est utilisé comme tout fichier du projet

# ZebFront : exemple

```
@Class NORTON_BEHAVIOR : BASIC_NL_BEHAVIOR
{
  @Name norton;
  @SubClass ELASTICITY elasticity;
  @Coefs K, n;
  @tVarInt eel;
  @sVarInt evcum;
};
@StrainPart {
  sig = *elasticity*eel;
  m_tg_matrix=*elasticity;
}
@Derivative {
  TENSOR2 sprime,norm;
  double J;
  sig=*elasticity*eel;
  sprime=deviator(sig);
  J=sqrt(1.5*(sprime|sprime));
  devcum=pow(J/K,n);
  norm=sprime*(1.5/J);
  deel=deto-devcum*norm;
}
```

Nom du comportement
Objet matrice d'élasticité
Coefficients de Norton
Variable interne tensorielle : $\underset{\sim}{\varepsilon}_e$
Variable interne scalaire : $p$

Calcul de la contrainte après intégration
$\underset{\sim}{\sigma} = \underset{\sim}{\underset{\sim}{\mathbf{E}}}\,\underset{\sim}{\varepsilon}_e$
Matrice tangente approchée (RK !)

Calcul du vecteur dérivé $\underline{\dot{Y}}$

Calcul du déviateur $\underset{\sim}{\sigma}'$
Calcul du deuxième invariant
Fluage de Norton : $\dot{p} = (\frac{J}{K})^n$
Direction de l'écoulement
Déformation élastique

```
@CalcGradF {
  ELASTICITY& E=*elasticity;
  sig = E*eel;
  f_vec_eel -= deto;
  TENSOR2 sigeff = deviator(sig);
  double J = sqrt(1.5*(sigeff|sigeff));
  if (J>(double)0.0) {
    TENSOR2 norm = sigeff*(1.5/J);
    f_vec_eel += norm*devcum;
    f_vec_evcum -= dt*pow(J/K,n);
    SMATRIX dn_ds = unit32;
    dn_ds -= norm ^ norm;
    dn_ds *= theta*devcum/J;
    deel_deel += dn_ds *E;
    deel_devcum += norm;
    double dv_df = tdt*n*pow(J/K,n-1)/K;
    TENSOR2 df_fs = dv_df*norm;
    devcum_deel -= df_fs*E;
  }
}
```

Intégration implicite

$\underset{\sim}{\sigma} = \underset{\sim}{\underset{\sim}{\mathbf{E}}}\underset{\sim}{\varepsilon}_e$

$\underset{\sim}{\mathbf{Re}} = \Delta\underset{\sim}{\varepsilon}_{\mathbf{e}} - \Delta\underset{\sim}{\varepsilon}$

Déviateur $\underset{\sim}{\sigma}'$

Deuxième invariant

Si on a plastifié

Direction de l'écoulement $\underset{\sim}{\mathbf{n}}$

$\underset{\sim}{\mathbf{Re}} = \Delta\underset{\sim}{\varepsilon}_{\mathbf{e}} - \Delta\underset{\sim}{\varepsilon} + \Delta p\underset{\sim}{\mathbf{n}}$

$\Delta p = (\frac{\tilde{J}}{K})^n \Delta t$

$\frac{\partial \underset{\sim}{\mathbf{Re}}}{\partial \Delta \underset{\sim}{\varepsilon}_{\mathbf{e}}}$

$\frac{\partial \underset{\sim}{\mathbf{Re}}}{\partial \Delta p}$

$\frac{\partial Rp}{\partial \Delta \underset{\sim}{\varepsilon}_{\mathbf{e}}}$

$\frac{\partial Rp}{\partial \Delta p} = 1$

# C++

Exemple de développement d’un nouvel élément :
élément piézo-électrique

# C++ : continuum piézo-électrique

Equations d'équilibre :

$$\begin{cases} \sigma_{ij,j} &= -f_i \\ D_{i,i} &= -\rho \end{cases}$$

Définition des grandeurs primales :

$$\begin{cases} \varepsilon_{ij} &= \frac{1}{2}(u_{i,j}+u_{j,i}) \\ E_i &= -V_{,i} \end{cases}$$

Loi de comportement :

$$\begin{cases} \sigma_{ij} &= c_{ijkl}\varepsilon_{kl} - e_{kij}E_k \\ D_i &= e_{ikl}\varepsilon_{kl} + A_{ij}E_j \end{cases}$$

# C++ : Définition d'un élément dans ZéBuLoN

La définition d'un nouvel élément dans ZéBuLoN passe par :

- l'énumération des divers degrés de liberté (nodaux, élémentaires, d'ensemble)
- le calcul des forces internes

On sépare

- la géométrie
- la cinématique
- l'intégration
- les lois de comportement

(association automatique)

Pour l'élément piézo-électrique : couplage fort mécanique / électrique.

Variables nodales : `Ui, V`

# C++ : élément piézo-électrique dans ZéBuLoN

```
class PIEZO : public ISOPARAMETRIC
{
  protected :
    int nb_mdof,nb_edof;
    int ezz_dof_start,ezz_dof_end;

    int mechanical_tsz();
    void compute_B(MATRIX&);
    void mechanical_symmetric_gradient_operator(const MATRIX&,MATRIX&) const;
    void electrical_gradient_operator(const MATRIX&,MATRIX&) const;

  public :
    PIEZO() : ISOPARAMETRIC() { ezz_dof_start=ezz_dof_end=-1; }
    virtual ~PIEZO() { }
    virtual void setup_dofs(const char*);
    virtual int dof_location(int) const;
    virtual INTEGRATION_RESULT* internal_reaction(bool,VECTOR&,SMATRIX&,
                                    bool only_get_tg_matrix=FALSE);
};
```

```
void PIEZO::setup_dofs(const char* key) {
  static DOF_TYPE dDofs[4]={
    DOF::translate("U1",DOF_NODE_ID),
    DOF::translate("U2",DOF_NODE_ID),
    DOF::translate("U3",DOF_NODE_ID),
    DOF::translate("EZZ",DOF_ELEMENT_ID) };
  static DOF_TYPE vDofs[1]={
    DOF::translate("V",DOF_NODE_ID) };

  int tot=0;

  nb_mdof = !node*space_dimension();
  if(space_dimension()==2) nb_mdof+=nb_gp();
  nb_edof = !node;

  size_dofs(nb_mdof+nb_edof);

  for(int inode=0;inode<nb_node();inode++) {
    for(int i=0;i<space_dimension();i++) {
      dof_type[tot]=dDofs[i];
      map_dof_to_node[tot]=get_node(inode);
      tot++;
    }
  }
  if(space_dimension()==2) {
    ezz_dof_start=tot;
    for(int igp=0;igp<nb_gp();igp++) {
      dof_type[tot]=dDofs[3];
      map_dof_to_node[tot]=NULL;
      tot++;
    }
    ezz_dof_end=tot;
  }

  for(int inode=0;inode<nb_node();inode++) {
    dof_type[tot]=vDofs[0];
    map_dof_to_node[tot]=get_node(inode);
    tot++;
  }
}
```

# Calcul *par bloc* des opérateurs gradient

$$B = \begin{pmatrix} B_{mech} & 0 \\ 0 & B_{elec} \end{pmatrix}$$

```
void PIEZO::compute_B(MATRIX& B)
{
  MATRIX Bmech (mechanical_tsz(),nb_mdof,B,0,0);
  MATRIX Belec (space_dimension(),nb_edof,B,
                mechanical_tsz(),nb_mdof);
  MATRIX dshape_dx(space_dimension(),nb_node());

  B=0.;
  compute_dshape_dx(dshape_dx);
  mechanical_symmetric_gradient_operator(dshape_dx,
                                         Bmech);
  electrical_gradient_operator(dshape_dx,Belec);
}
```

```
void PIEZO::electrical_gradient_operator(
       const MATRIX& dshape_dx, MATRIX& b) const
{
  b=0.;
  for(int inode=0;inode<nb_node();inode++)
    for(int idim=0;idim<space_dimension();idim++)
      b(idim,inode)=-dshape_dx(idim,inode);
}
```

# Calcul des forces internes :

```
INTEGRATION_RESULT*
PIEZO::internal_reaction(bool if_compute_stiffness, VECTOR& resi,
         SMATRIX& stiff, bool only_get_tg_matrix)
{
  VECTOR elem_UV(nb_dof()), elem_dUV(nb_dof());
  MATRIX B,elem_coord,*tg_matrix;
  SMATRIX dK;
  VECTOR delta_grad,vgrad,vflux,dF;
  INTEGRATION_RESULT *ret=NULL;
  int flags=0;

  get_elem_coord(elem_coord);
  get_elem_d_dof_tot(elem_UV);
  get_elem_d_dof_incr(elem_dUV);

  flags=BEHAVIOR::GIVE_FLUX;
  resi=0.;
  if(if_compute_stiffness) {
    stiff=(double)0.;
    flags=flags|BEHAVIOR::CALC_TG_MATRIX;
    dK.resize(nb_dof());
  }
```

```
for(start(elem_coord);ok();next(elem_coord)) {
  TENSOR2_GRAD& eto = *(TENSOR2_GRAD*)pub_behavior()->get_grad_var("eto");
  TENSOR2_FLUX& sig = *(TENSOR2_FLUX*)pub_behavior()->get_flux_var("sig");

  VECTOR_GRAD&  E   = *(VECTOR_GRAD*)pub_behavior()->get_grad_var("E");
  VECTOR_FLUX&  D   = *(VECTOR_FLUX*)pub_behavior()->get_flux_var("D");

  if(!vgrad==0) {
    vgrad.resize(!(TENSOR2)eto+!(VECTOR)E);
    vflux.resize(!vgrad);
    B.resize(!(VECTOR)E+!(TENSOR2)eto,nb_dof());
  }

  compute_B(B);

  delta_grad=B*elem_dUV;
  vgrad=B*elem_UV;
  VECTOR v_eto(!(TENSOR2)eto,vgrad,0),
         v_E(!(VECTOR)E,vgrad,!(TENSOR2)eto);
  eto=v_eto; E=v_E;
```

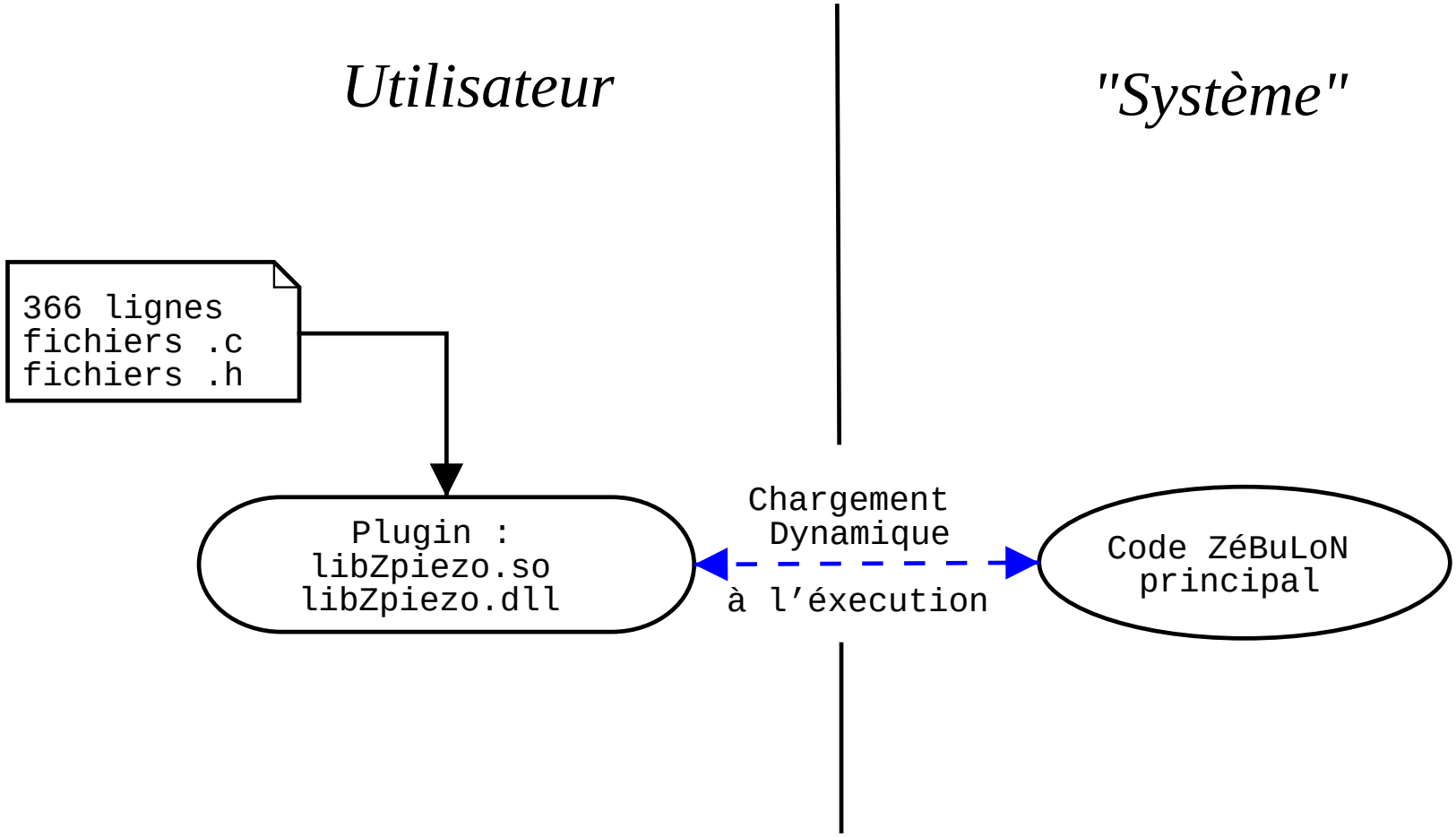
```
    if((ret=pub_behavior()->integrate(
              pub_mat_data(),delta_grad,tg_matrix,flags))) return ret;

    if (if_compute_stiffness) {
      dK=transpose(B)*( *((SMATRIX*)tg_matrix) )*B;
      integrate(dK,stiff);
    }

    VECTOR v_sig(!(TENSOR2)sig,vflux,0),v_D(!(VECTOR)E,vflux,!(TENSOR2)sig);
    v_sig=sig; v_D=D;

    dF=transpose(B)*vflux;
    integrate(dF,resi);

  }
  return(ret);
}
```

# Loi de comportement

```
class LINEAR_PIEZO : public BEHAVIOR
{
 protected :
  TENSOR2_GRAD eto;
  TENSOR2_FLUX sig;

  VECTOR_GRAD  E;
  VECTOR_FLUX  D;

  S_COEFFICIENT_TENSOR4* mechanical_elasticity;
  COEFFICIENT_MATRIX*    electrical_elasticity;
  COEFFICIENT_TENSOR3*   piezo_elasticity;

  SMATRIX tgm;
  SMATRIX tg_mm,tg_ee;
  MATRIX tg_em,tg_me;

  MATRIX matrix_tensor2_right(TENSOR3&);
  MATRIX matrix_vector_right(TENSOR3&);

  int mechanical_tsz() {return(_dim==2 ? 4 : 6);}
  int electrical_tsz() {return(_dim==2 ? 2 : 3);}

 public :
  LINEAR_PIEZO() : BEHAVIOR() { }
  virtual ~LINEAR_PIEZO() { }

  virtual void initialize(ASCII_FILE& file,
                          int dim,LOCAL_INTEGRATION*);
  virtual INTEGRATION_RESULT* integrate(MAT_DATA&,
                           const VECTOR&,MATRIX*&,int);
};
```

```
INTEGRATION_RESULT* LINEAR_PIEZO::integrate(
                      MAT_DATA&      mdat,
                      const VECTOR&  delta_grad,
                      MATRIX*&       tg_matrix,
                      int            flags_in   )
{
  if (!curr_ext_param)
     curr_ext_param = *mdat.param_set();
  calc_local_coefs();

  TENSOR2 s1(!(TENSOR2)eto),s2(!(TENSOR2)eto);
  VECTOR d1(!(VECTOR)E),d2(!(VECTOR)E);
  MATRIX mmm;
  int icol,iline;

  if(flags_in & GIVE_FLUX) {
    s1=(*mechanical_elasticity)*eto;
    s2=(*piezo_elasticity)*E;

    d1= (*electrical_elasticity)*E;
    d2= (*piezo_elasticity)*eto.mkmat();

    sig = s1 - s2;
    D   = d1 + d2;
  }

  if(flags_in & CALC_TG_MATRIX) {
    tg_matrix=&tgm;

    tg_mm=(MATRIX)*mechanical_elasticity;
    tg_ee=(MATRIX)*electrical_elasticity;

    tg_me=-1.*piezo_elasticity->matrix_tensor2_right()
  }

  return NULL;
}
```

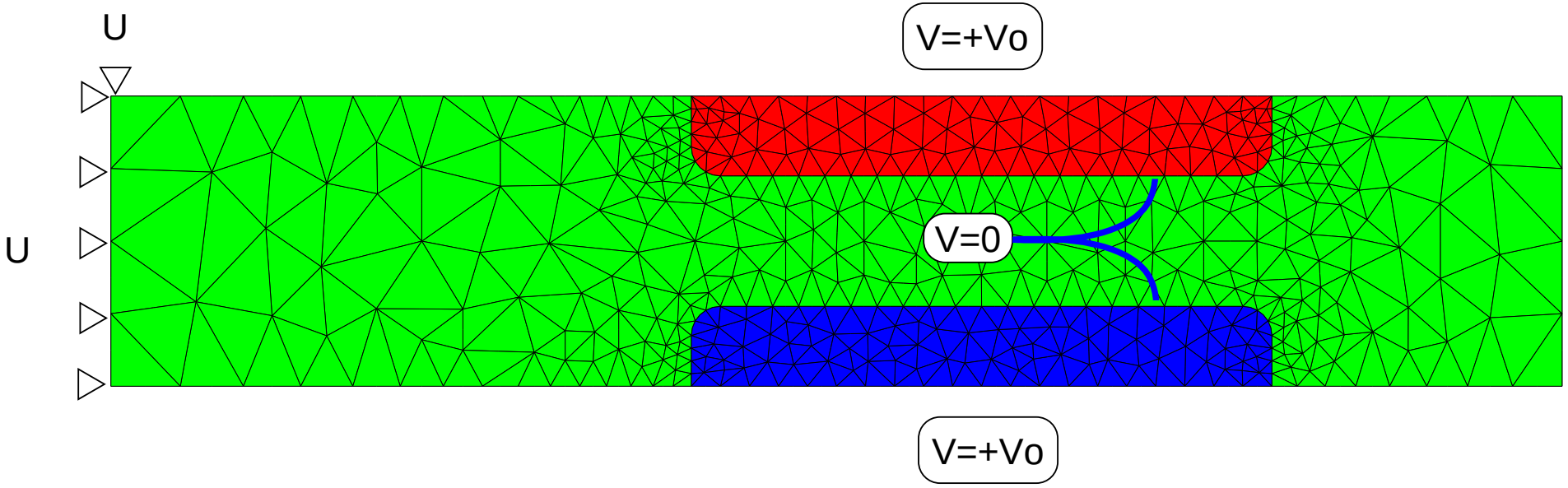
# C++ : Compilation & édition de liens



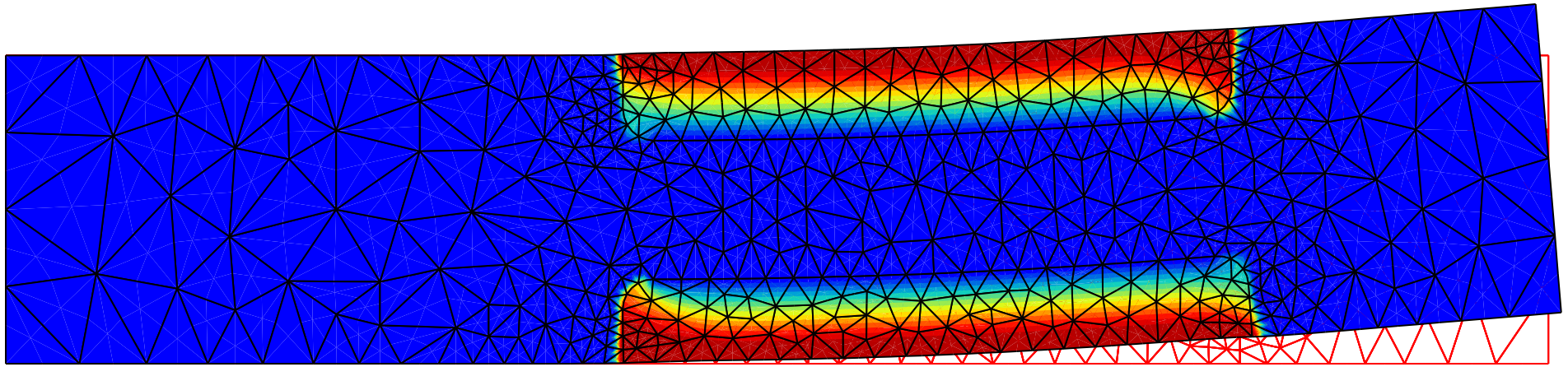
# Diverses applications

- → Elément piézo-électrique
- → ZéBuLoN
- → Essai Charpy
- → Microstructures
- → Aube de turbine
- → Culasse: performances parallèles

# Application de l’élément piézo-électrique à une poutre

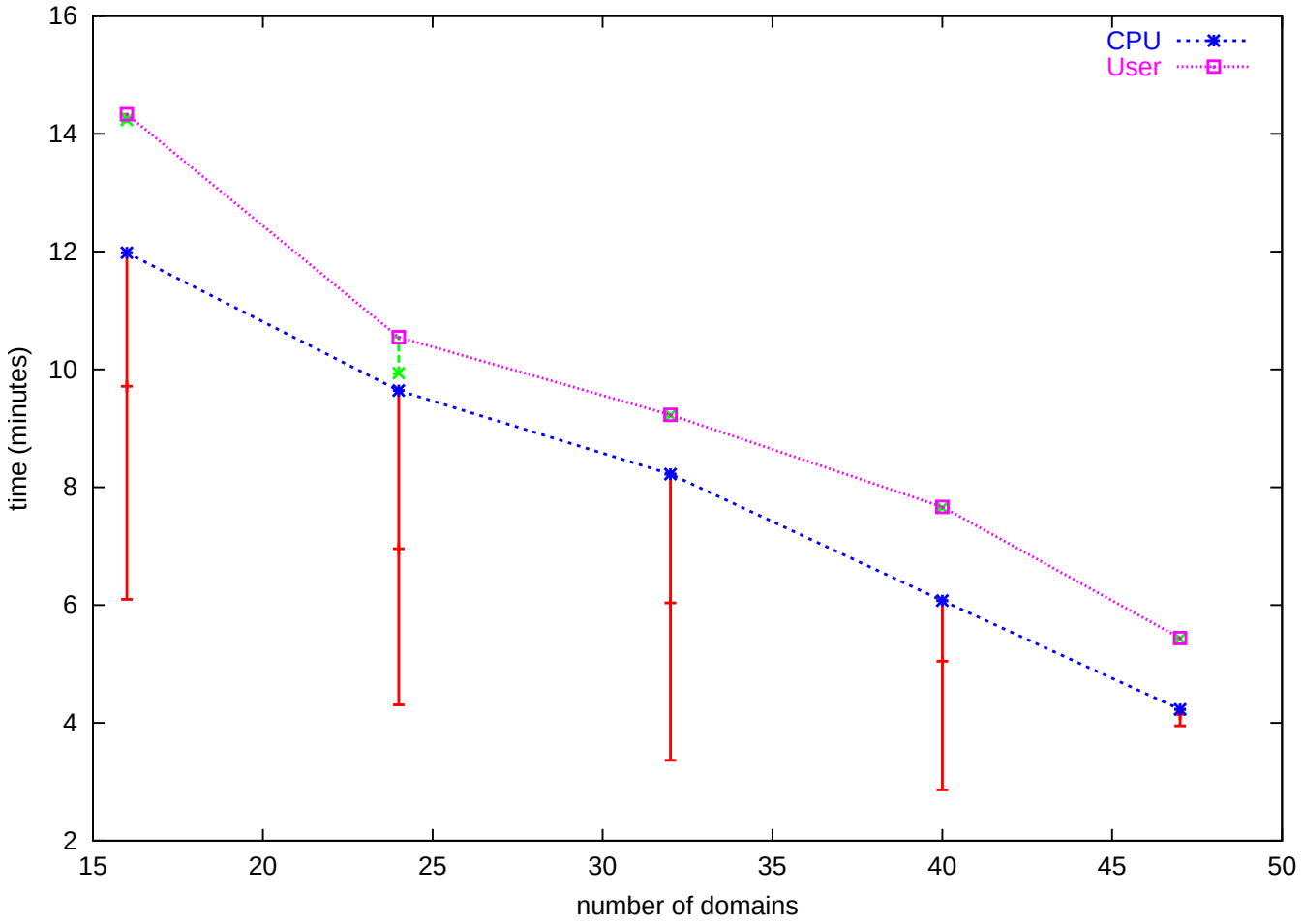


# Application de l’élément piézo-électrique à une poutre

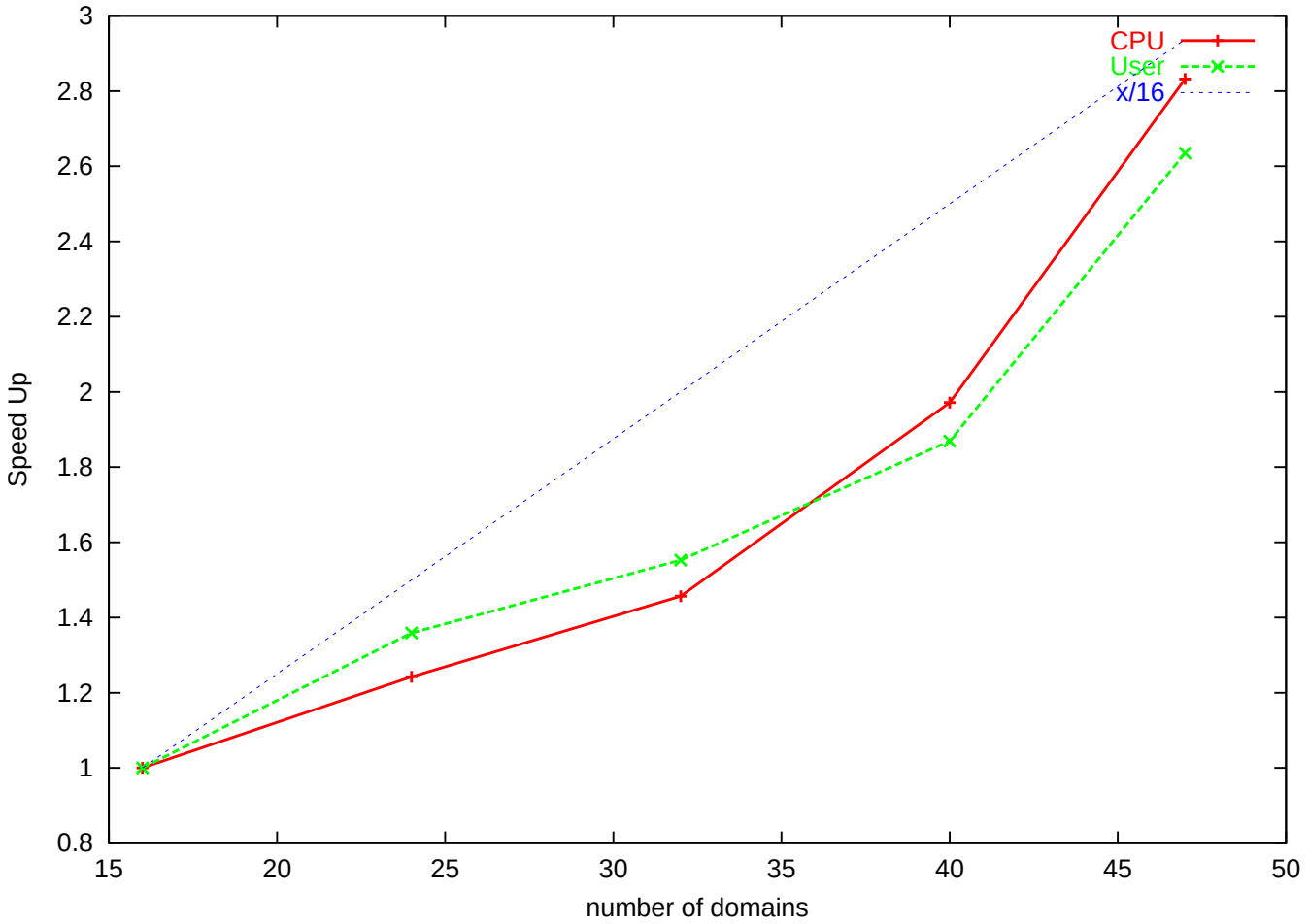


*exemples*

# Culasse : Temps de résolution du système linéaire



# Culasse : Speed-Up



# Diffusion du code : état actuel

- Installation et maintenance par VAL (Europe) et NWNumerics (USA)
- Location ou vente aux industriels (surtout Z-mat)
- Mise à disposition gratuite pour les universitaires
- Librairies, exécutables, headers
- 50 licences environ

# Diffusion du code : licences

- France : 21 I, 14 U
  - ⋆ Snecma, Renault, CEA, Paulstra,...
  - ⋆ ENSAM, ENSICA, Mines d'Albi,...
- Europe hors France : 7 I, 7 U
  - ⋆ BAM, DERA,...
  - ⋆ Imperial College, Bochum,...
- USA–Canada: 5 I, 2 U
  - ⋆ General Motor, 3M,...
  - ⋆ Washington Univ., Polytechnique Montréal

# Offre pour Codiciel

- Source pour Codiciel (branche CVS)
- Accord Codiciel–groupe de développeur Z-SeT
- Codiciel apparaît comme un acteur du groupe
- Codiciel peut redistribuer les executables et les headers
- Possibilité de diffuser le source si nécessaire
- Activité Codiciel ?

# *Exemple d'un carré en traction*

- Fichier carre.geof

```
***geometry
 **node
 1  0.   0.
 2  1.   0.
 3  1.   1.
 4  0.   1.
 **element
 1 c2d4 1 2 3 4
***group
 **nset bottom 1 2
 **nset top    3 4
 **nset left   4 1
***return
```

*zebulon*

- Fichier carre.inp

```
****calcul
 ***mesh plane_stress
 ***resolution
  **sequence
   *time 1.0
 ***bc
  **impose_nodal_dof
    bottom U1 0.0
    top    U1 0.01  time
    left   U2 0.0
 ***material
   *file carre.inp
****return

 ***behavior linear_elastic
  **elasticity
    young 200000. poisson 0.3
 ***return
```

- Fichier Elasticité

```
***behavior elastic lagrange_rotate
 **elasticity
   young 10000.  poisson 0.3
***return
```

- Fichier Viscoplasticité

```
***behavior gen_evp
 **elasticity isotropic
   young 200000.  poisson 0.25
 **potential gen_evp ev
  *flow norton
   n  4.0  K  500.
  *criterion hill
   a 1. b 2. c 3. d 1.2 e 1.7 f 0.8
  *kinematic nonlinear
   C 30000. D 500.
  *isotropic constant
   R0 function 100.-0.1*temperature
***return
```

- Fichier Matériau de Gurson

```
***behavior porous_plastic
 **elasticity
   young 2.e5  poisson 0.3
 **porous_potential
  *porous_criterion gurson
   q1 1.5 q2 1.0
   fs    f
   0.    0.
   5.    0.5
  10.    1.0
*flow norton K .01 n 5.
*isotropic_hardening power_law
   n .1 e0 1.e-3 K  399.052
***return
```

*bibmat*

# Multimat

- Utilisation de règles d'homogénéisation
  - ⋆ Règles de localisation
  - ⋆ Lois de comportement locales (éventuellement multimat)

- Echelle macroscopique (0)

```
***behavior mori_tanaka
   **material  0.65 matrice
     *file matrice.mat
   **material  0.35 fibre
     *file elas.mat
     *rotation  x1  0.2  0.3  0.4
                x2  0.7  0.1 -0.3
***return
```

- Matériaux de l'échelle (1) à définir

*bibmat*

- Echelle 1

matrice.mat

```
***behavior berveiller_zaoui
  **mu  75000.  **nu  0.3
  **material  0.50 austenite
    *file austenite.mat
  **material  0.50 ferrite
    *file ferrite.mat
***return
```

fibre.mat

```
***behavior linear_elastic
 **elasticity orthotropic
   y1111 100000. y2222 120000.
       ...       y3131  90000.
***return
```

- Echelle 2

austenite.mat

```
***behavior gen_evp
 **elasticity isotropic
   young 260000. poisson 0.3
 **potential gen_evp ep
  *flow plasticity
  *isotropic constant
      R0     130.
***return
```

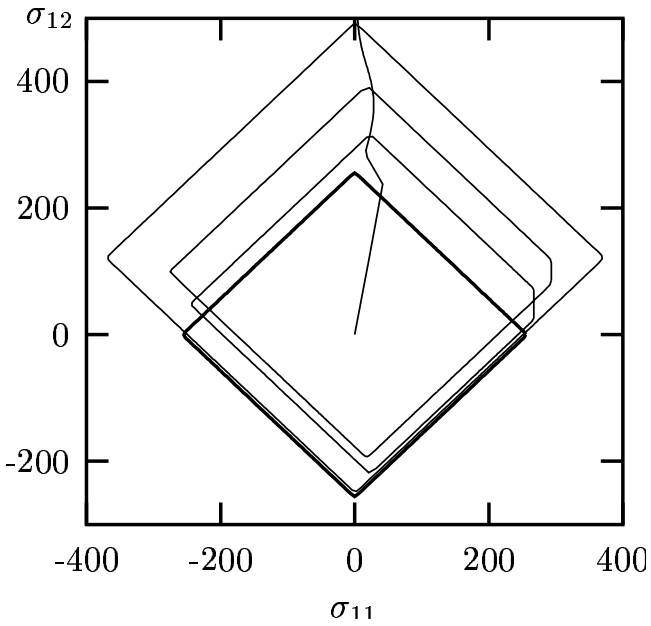
ferrite.mat

```
***behavior gen_evp
  ...
***return
```

*bibmat*

```
****simulate
  ***test yield-monox
   **load  *segment
     time  eto11 sig22 sig33 eto12
       0.   0.    0.    0.    0.
     100.   0.01  0.    0.    0.03
   **model
     *file yield.inp
   **yield_surface yield2.test
     *degrees    5.
     *component sig11 sig12
     *time      100.
****return

***behavior gen_evp
 **elasticity cubic
y1111 260000. Y1122 180000.
Y1212 120000.
 **potential octahedral ev
  *isotropic nonlinear
   R0 100. Q 100. b 12.
  *interaction slip
  h1 1. h2 1. h3 1.2 h4 1.5
***return
```

- Surfaces de charge sur un monocristal




Plan $\sigma_{11}$–$\sigma_{12}$

*zsim*

- Local

```
****post_processing
***local_post_processing
 **output_number 1
 **file node
 **nset MY_NODES
 **process function
  *output DEPLA
  *expression sqrt(U1*U1+U2*U2);
 **file integ
 **elset ALL_ELEMENT
 **process eigen2 *var sig
****return
```

- Global/local

```
****post_processing
***local_post_processing
 **file integ
 **output_number 1-70
 **process trace *var sig
***global_post_processing
 **process average_in_element
  *list_var sigii
***local_post_processing
 **process function
  *output sigm11
  *expression sig11+(ae_sigii-sigii)/3.
****return
```

*zlife*