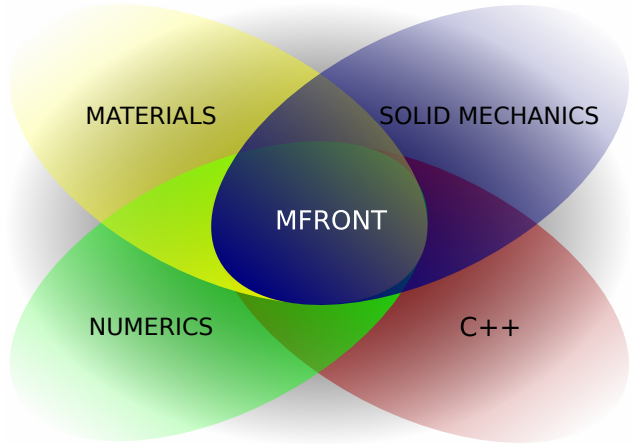
DE LA RECHERCHE À L'INDUSTRIE

cea

www.cea.fr

Material knowledge management with the MFront code generator. Description of the ZMAT interface



Club des utilisateurs du code Z-set | THOMAS HELFER, OLIVIER FANDEUR, THOMAS DE SOZA, DOMINIQUE DELOISON, CHARLES TOULEMONDE

28 FEBRUAR 2017

Context

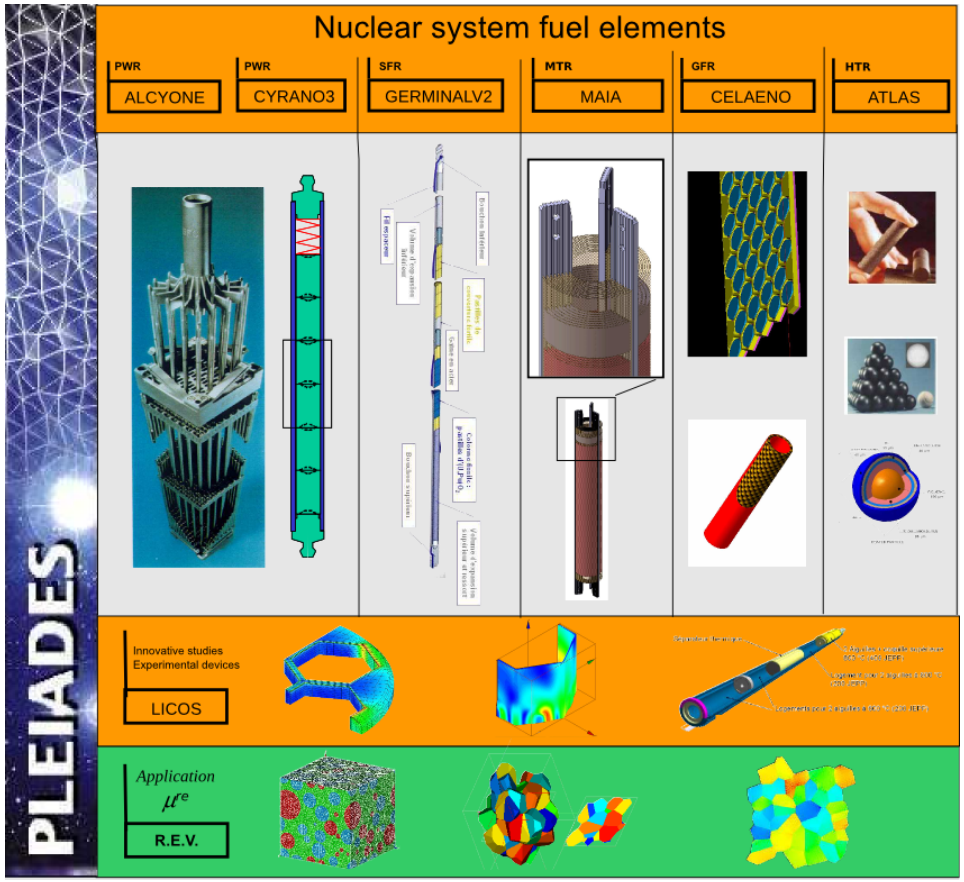
DE LA RECHERCHE À L'INDUSTRIE
cea

Outline

- Context
- Material knowledge management in nuclear simulations
- The `TFEL` project and the `MFront` code generator
- Mechanical behaviours
- The `ZMAT` interface
- Conclusions

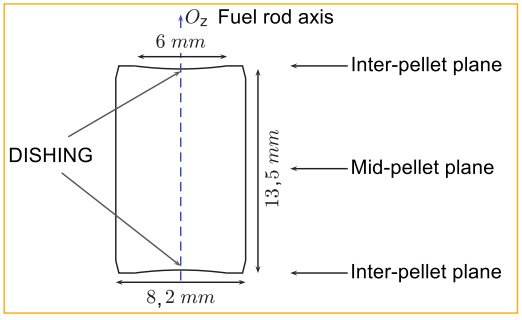
Material knowledge management in nuclear simulations

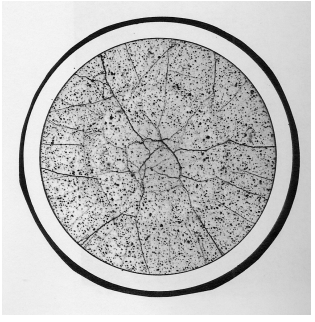
The Pleiades platform

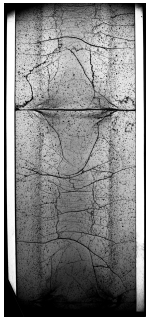


- A wide range of materials (ceramics, metals, composites).
- A wide range of mechanical phenomena and behaviours.
  - Creep, swelling, irradiation effects, phase transitions, etc..
- A wide range of mechanical loadings.

The PWR fuel element







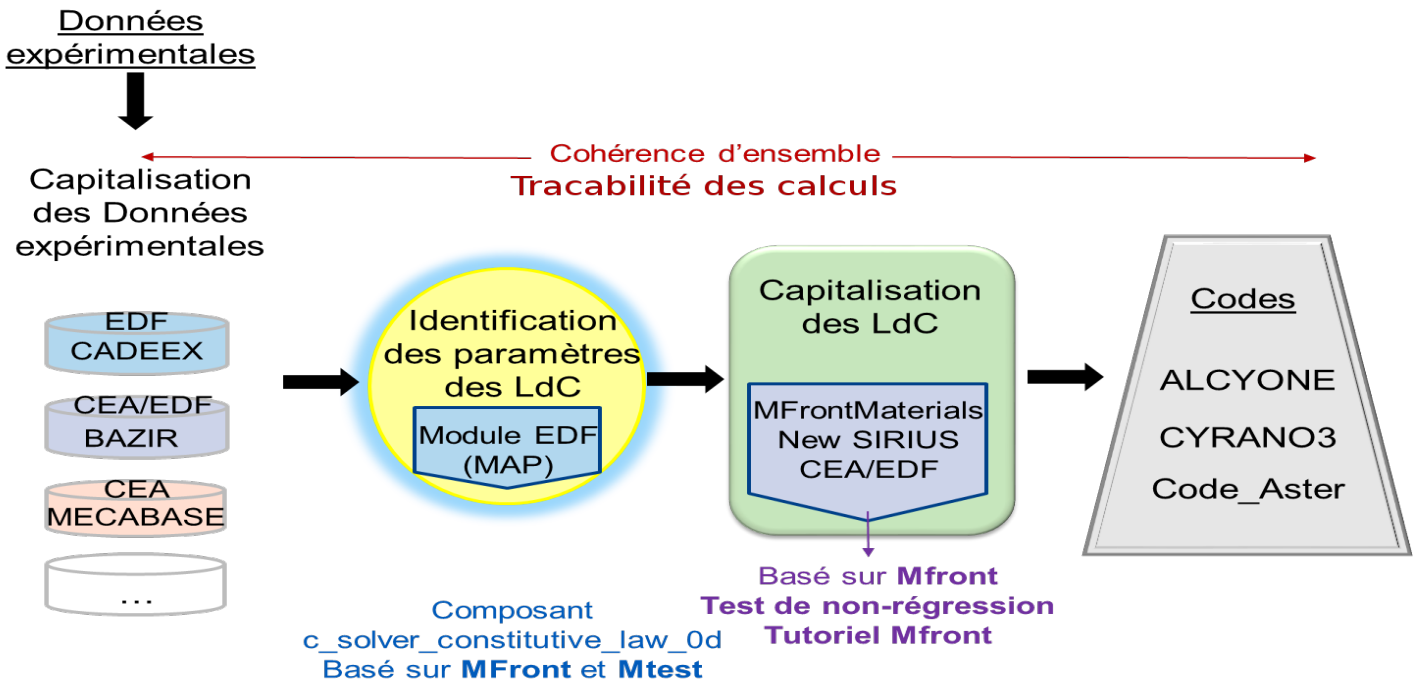
- Brittle fracture.
- Porosity growth.
- Viscoplatic behaviour.

Material knowledge management

- The need to guarantee the quality of engineering studies has never been so high and is constantly growing.
- Every part of a study must be covered by strict AQ procedures :
  - The finite element solver on the one hand (see the `Code_Aster` documentation and unit tests).
  - The material knowledge (material properties, **mechanical behaviours**) and experimental data on the other hand.
- **One must guarantee a complete consistency from experimental data to engineering studies**
- Many mechanical behaviours used in nuclear simulations are identified during Phd's under the supervision of the Centre des Matériaux.

$\Rightarrow$ **Strong interest to interoperate with `ZSet` tools.**

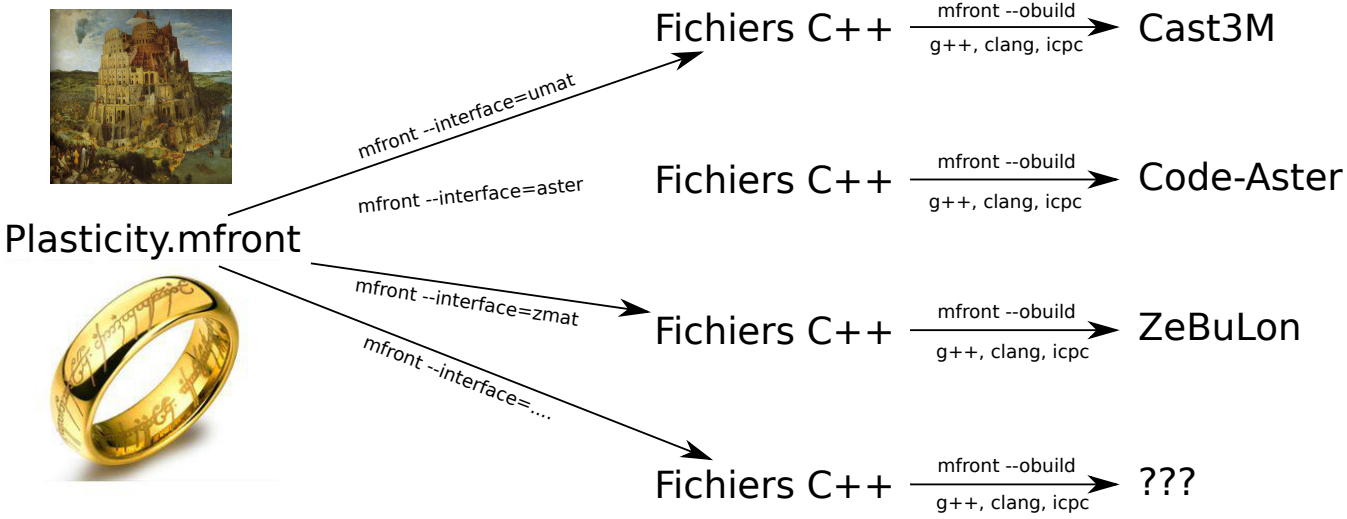
Current situation and perspectives



- **Rationalisation and Standardisation** is **required**, in the Nuclear industry (EDF, AREVA, CEA) :
  - Each partner has built its own strategy.
  - **A working group** has recently been set up to tackle this issue.
- However, we are willing to share this effort with the whole (french) community of the non-linear mechanics.

The TFEL project and the MFront code generator

MFront goals



- `TFEL` is a project various libraries/softwares, including `MFront`.
- `MFront` is a code generation tool dedicated to material knowledge (material properties, mechanical behaviours, point-wise models) with focus on :
  - Numerical efficiency (see various benchmarks).
  - Portability (`Cast3M`, `Cyrano`, `Code_Aster`, `Europlexus`, `TMFTT`, `AMITEX_FFTP`, `Abaqus`, `CalculiX`, `MTest`).
  - **Ease of use** : *Longum iter est per praecepta, breve et efficax per exempla* (It's a long way by the rules, but short and efficient with examples).

Timeline : more than 10 years of development

- 2006 : first line of codes :
  - Mainly focused on the `Cast3M` finite solver.
- 2009 : officially part of the PLEIADES project :
  - Development of the `Cyrano` interface.
- 2013 : first contact with the `Code_Aster` team.
- 2014 : TFEL-2.0 is released as an open-source project :
  - Officially co-developped by CEA and ÉDF.
  - Implementing an interface to `ZMAT`.
- 2015 : Officially part of the `Code_Aster/Salomé-Méca` distribution.
- 2016 : TFEL-3.0 is released :
  - Support of `Europlexus`, `Abaqus/Standard`, `Abaqus/Explicit`
  - $\approx$ 250 000 lines of codes

Licences : Open-source (again) !

- To meet CEA and EDF needs, TFEL 2.0 is released under a multi-licensing scheme :
  - Open-source licences :
    - GNU Public License : This licence is used by the Code_Aster finite element solver.
    - CECILL-A : License developped by CEA, EDF and INRIA, compatible with the GNU Public License and designed for conformity with the French law.
  - CEA and EDF are free to distribute TFEL under custom licences : Mandatory for the PLEIADES plateform.

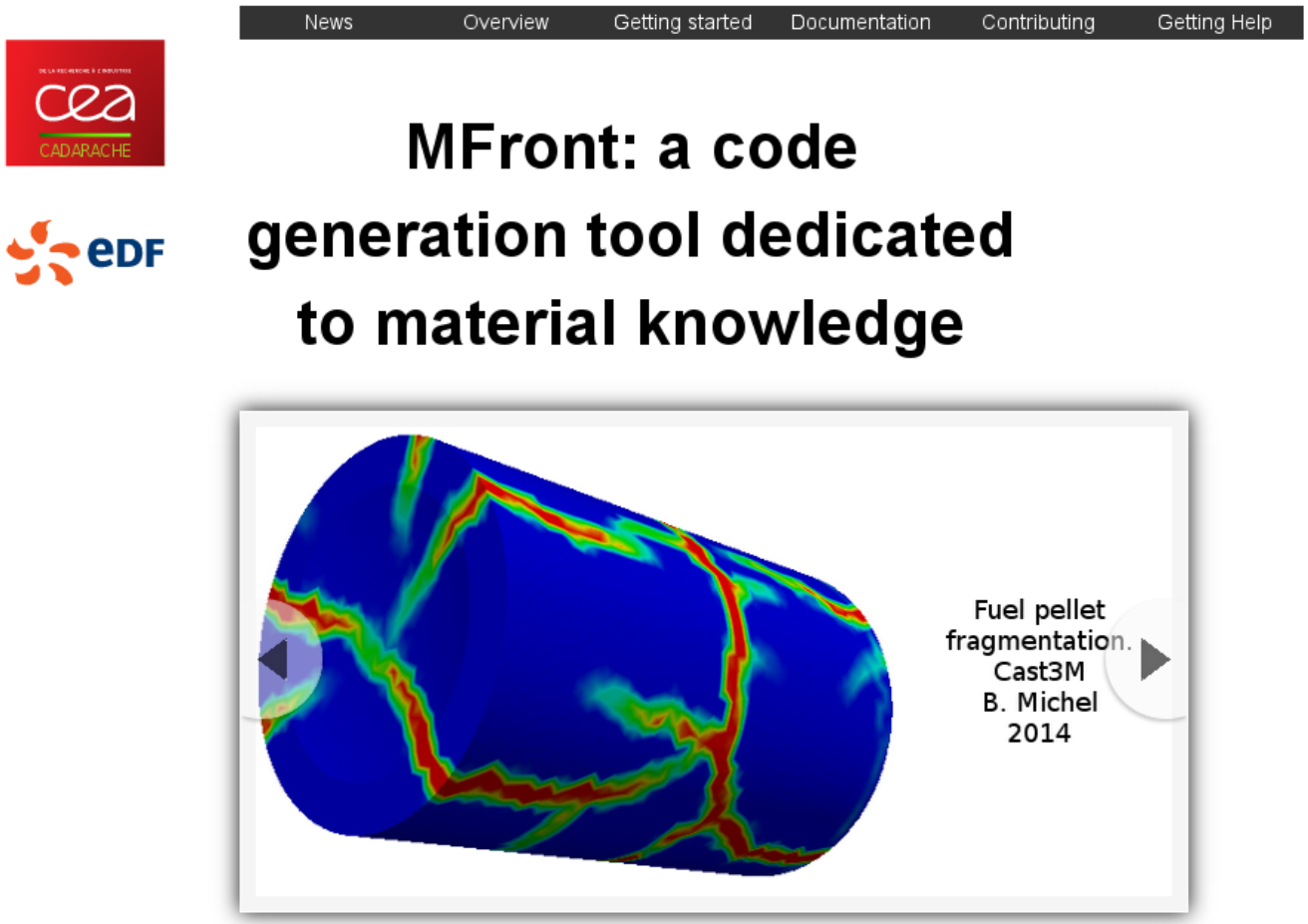
Software Portability

- TFEL 3.*x* is based on the C++ 11 standard.
- TFEL 3.*x* can be compiled with various C++ compilers :
  - gcc, from version 4.7 to version 6.3
  - clang, from version 3.5 to version 3.9
  - icc, version 2016.
  - Visual Studio, version 2016.
- TFEL is mainly developed on LiNuX.
- ports have been made to various POSIX systems, Mac Os, FreeBSD, OpenSolaris, etc... and Windows !

Software quality : an industrial strength software

- Very stringent compilers warnings :
  - `g++ -Wall -W -Wextra -pedantic -Wdisabled-optimization -Wlong-long -Winline -Wswitch -Wsequence-point -Wignored-qualifiers -Wzero-as-null-pointer-constant -Wvector-operation-performance -Wtrampolines -Wstrict-null-sentinel -Wsign-promo -Wsign-conversion -Wold-style-cast -Wnoexcept -Wmissing-include-dirs -Wmissing-declarations -Wlogical-op -Winit-self ...`
- Documentation.
- Continuous integration based on `Jenkins`
- More than 3 500 tests :
  - Most of them are based on `mtest`
- More tests inside `PLEIADES` applications.

The website



http://tfel.sourceforge.net

http://tfel.sourceforge.net/about.html

http://tfel.sourceforge.net/gallery.html

Mechanical behaviours

Resolution

- Mechanical equilibrium : find$\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

Resolution

- Mechanical equilibrium : find $\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- element contribution to inner forces :

$$\begin{aligned}\vec{\mathbb{F}}_i^e &= \int_{V^e} \underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}, \Delta t\right) : \underline{\mathbf{B}}\, \mathrm{d}V \\ &= \sum_{i=1}^{N^G} \left(\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta t\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right)\right) w_i\end{aligned}$$

where $\underline{\mathbf{B}}$ gives the relationship between $\Delta\underline{\epsilon}^{to}$ and $\Delta\vec{\mathbb{U}}$

Resolution

- Mechanical equilibrium : find$\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- element contribution to inner forces :

$$\vec{\mathbb{F}}_i^e = \sum_{i=1}^{N^G} \left(\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta t\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right)\right) w_i$$

- Resolution using the Newton-Raphson algorithm :

$$\Delta\vec{\mathbb{U}}^{n+1} = \Delta\vec{\mathbb{U}}^n - \left(\left.\frac{\partial\vec{\mathbb{R}}}{\partial\Delta\vec{\mathbb{U}}}\right|_{\Delta\vec{\mathbb{U}}^n}\right)^{-1} .\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right) = \Delta\vec{\mathbb{U}}^n - \underline{\underline{\mathbb{K}}}^{-1} .\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right)$$

Resolution

- Mechanical equilibrium : find $\Delta\vec{\mathbb{U}}$ such as :

$$\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{O}} \quad \text{avec} \quad \vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}\right) = \vec{\mathbb{F}}_i\left(\Delta\vec{\mathbb{U}}\right) - \vec{\mathbb{F}}_e$$

- element contribution to inner forces :

$$\vec{\mathbb{F}}_i^e = \sum_{i=1}^{N^G} \left(\underline{\sigma}_{t+\Delta t}\left(\Delta\underline{\epsilon}^{to}\left(\vec{\eta}_i\right), \Delta t\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right)\right) w_i$$

- Resolution using the Newton-Raphson algorithm :

$$\Delta\vec{\mathbb{U}}^{n+1} = \Delta\vec{\mathbb{U}}^n - \underline{\underline{\mathbb{K}}}^{-1}.\vec{\mathbb{R}}\left(\Delta\vec{\mathbb{U}}^n\right)$$

- element contribution to the stiffness :

$$\underline{\underline{\mathbb{K}}}^e = \sum_{i=1}^{N^G} {}^t\underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) : \frac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}\left(\vec{\eta}_i\right) : \underline{\underline{\mathbf{B}}}\left(\vec{\eta}_i\right) w_i$$

$\frac{\partial\Delta\underline{\sigma}}{\partial\Delta\underline{\epsilon}^{to}}$ is the **consistent tangent operator**

Main functions of the mechanical behaviour

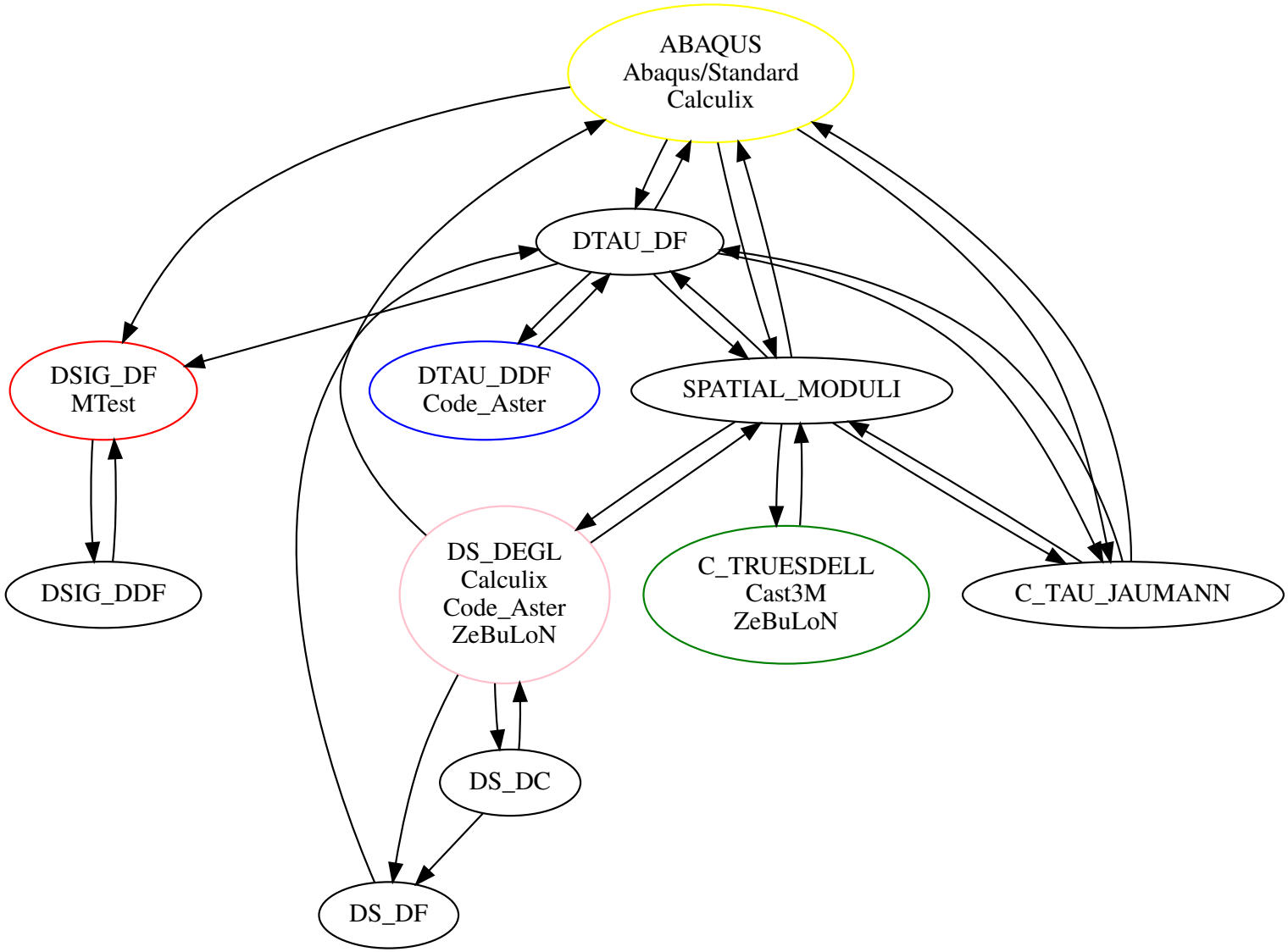
$$\left(\underline{\epsilon}^{to}\big|_t \, , \, \vec{Y}\Big|_t \, , \Delta \underline{\epsilon}^{to} \, , \Delta t \right) \underbrace{\Longrightarrow}_{behaviour} \left(\underline{\sigma}|_{t+\Delta t} \, , \, \vec{Y}\Big|_{t+\Delta t} \, , \frac{\partial \Delta \underline{\epsilon}^{to}}{\partial \Delta \underline{\epsilon}^{to}} \right)$$

- Given a strain increment $\Delta \underline{\epsilon}^{to}$ over a time step Δt, the mechanical behaviour must compute :
  - The value of the stress $\underline{\sigma}|_{t+\Delta t}$ at the end of the time step.
  - The value of internal state variables, noted $\vec{Y}\Big|_{t+\Delta t}$ at the end of the time step.
  - The consistent tangent operator : $\dfrac{\partial \Delta \underline{\sigma}}{\partial \Delta \underline{\epsilon}^{to}}$
- For specific cases, the mechanical behaviour shall also provide :
  - a prediction operator
  - the elastic operator (Abaqus-Explicit, Europlexus)
  - estimation of the stored and dissipated energies (Abaqus-Explicit)

Other functions of the mechanical behaviour

- Provide a estimation of the next time step for time step automatic adaptation
- Check bounds :
  - Physical bounds
  - Standard bounds
- Clear error messages
- Parameters
  - It is all about AQ !
  - Parametric studies, identification, etc. . .
- Generate mtest files on integration failures
- Generated example of usage :
  - Generation of MODELISER/MATERIAU instructions (Cast3M)
  - Input file for Abaqus
- Provide information for dynamic resolution of inputs (MTest/Aster/Europlexus) :
  - Numbers Types (scalar, tensors, symmetric tensors)
  - Entry names /Glossary names. . .

Example of portability issue : tangent operator for finite strain behaviours



Some optimisation techniques in `TFEL/Math`

- Expression templates :
  - `const auto a=b+c;`
  - `a` is the operation of adding `b` and `c`.
  - the operation is only performed when `a` is assigned to a concrete object (lazy evaluation).
  - `a` **shall be eliminated** by the optimizer.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.

Some optimisation techniques in `TFEL/Math`

- Expression templates.
- Loop unrolling :
  - Consider `const stensor<1u> a=b+c+2*d;`.
  - This operation is equivalent to :

```
a[0]=b[0]+c[0]+2*d[0];
a[1]=b[1]+c[1]+2*d[1];
a[2]=b[2]+c[2]+2*d[2];
```

  - This requires to compile a specialisation of the behaviour several times for every supported modelling hypotheses.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.

Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations :

```
template<typename T,typename T2>
typename std::enable_if<
   (( tfel ::meta::Implements<T,StensorConcept>::cond)  && (StensorTraits<T>::dime==3u)&&
    ( tfel ::meta::Implements<T2,StensorConcept>::cond) && (StensorTraits<T2>::dime==3u)&&
    ( tfel :: typetraits ::IsFundamentalNumericType<StensorNumType<T2>>::cond)),
  stensor<3u,StensorNumType<T>>
 >::type
convertCorotationnalCauchyStressToSecondPiolaKirchhoffStress(const T& s, const T2& U){
  using real = tfel :: typetraits ::base_type<StensorNumType<T>>;
  constexpr real cste = Cste<real>::sqrt2;
  const auto J  = det(U);
  const auto iU = invert(U);
  return {J*(s[2]*iU[4]*iU[4]+(cste*s[5]*iU[3]+2*s[4]*iU[0])*iU[4]+s[1]*iU[3]*iU[3]+2*s[3]*iU[0]*iU[3]+2*s[0]*iU[0]*iU[0])/2,
          J*(s[2]*iU[5]*iU[5]+(cste*s[4]*iU[3]+2*s[5]*iU[1])*iU[5]+s[0]*iU[3]*iU[3]+2*s[3]*iU[1]*iU[3]+2*s[1]*iU[1]*iU[1])/2,
          J*(s[1]*iU[5]*iU[5]+(cste*s[3]*iU[4]+2*s[5]*iU[2])*iU[5]+s[0]*iU[4]*iU[4]+2*s[4]*iU[2]*iU[4]+2*s[2]*iU[2]*iU[2])/2,
          J*((cste*s[2]*iU[4]+s[5]*iU[3]+cste*s[4]*iU[0])*iU[5]+(s[4]*iU[3]+cste*s[5]*iU[1])*iU[4]+s[3]*iU[3]*iU[3]+(2*s[1]*iU[1]+2*s[0]*iU[0])*iU[3]+2*s[3]*iU[0]*iU[1])/2,
          J*((s[5]*iU[4]+cste*s[1]*iU[3]+cste*s[3]*iU[0])*iU[5]+s[4]*iU[4]*iU[4]+(s[3]*iU[3]+2*s[2]*iU[2]+2*s[0]*iU[0])*iU[4]+cste*s[5]*iU[2]*iU[3]+2*s[4]*iU[0]*iU[2])/2,
          J*(s[5]*iU[5]*iU[5]+(s[4]*iU[4]+s[3]*iU[3]+2*s[2]*iU[2]+2*s[1]*iU[1])*iU[5]+(cste*s[0]*iU[3]+cste*s[3]*iU[1])*iU[4]+cste*s[4]*iU[2]*iU[3]+2*s[5]*iU[1]*iU[2])/2};
}
```

- Views.
- Compile-time dimensional analysis.

Some optimisation techniques in `TFEL/Math`

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views :
  - $J = \begin{pmatrix} \frac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta \underline{\epsilon}^{el}} & \frac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta p} \\ \frac{\partial f_p}{\partial \Delta \underline{\epsilon}^{el}} & \frac{\partial f_p}{\partial \Delta p} \end{pmatrix}$
  - `dfeel_ddeel` is view of J :
    - Implementing the `ST2toST2Concept` concept.
    - Direct modification of J.
    - Compile-time computation of the offset.
- Compile-time dimensional analysis.

Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis :
  - Adding a strain and a stress tensor leads to a **compile-time** error.
  - Available in **TFEL** for years.
  - Not yet available in MFront.
  - Feature planned for TFEL 3.1.

Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.
- **Those techniques far exceed the programming skills of most C++ professional programmers.**
  - We have anticipated the C++ language evolutions.
  - Some parts of TFEL are deprecated and removed when an equivalent features are introduced in the standard.
  - Concepts will have build-in language support in C++ -20.
  - It explains the great gap between TFEL-2.0 and TFEL-3.0, but « old »MFront files are unaffected ⇒ transparent for the end users.

Some optimisation techniques in TFEL/Math

- Expression templates.
- Loop unrolling.
- Concepts and partial specialisations.
- Views.
- Compile-time dimensional analysis.
- **Those techniques far exceed the programming skills of most C++ professional programmers.**
  - We have anticipated the C++ language evolutions.
  - Some parts of TFEL are deprecated and removed when an equivalent features are introduced in the standard.
  - Concepts will have build-in language support in C++ -20.
  - It explains the great gap between TFEL-2.0 and TFEL-3.0, but « old »MFront files are unaffected ⇒ transparent for the end users.
- **A code generator is required for the library to be used by "standard" engineers** ⇒ MFront**.**

An very simple example

```
@DSL       Implicit ;
@Behaviour Norton;
@Brick StandardElasticity;

@MaterialProperty stress young;
@MaterialProperty real nu,A,E;
young.setGlossaryName("YoungModulus");
nu.setGlossaryName("PoissonRatio");
A.setEntryName("NortonCoefficient");
E.setEntryName("NortonExponent");

@StateVariable strain p;
p.setGlossaryName("EquivalentViscoplasticStrain");

@Integrator{
  const real  eps = 1.e-12;
  const auto mu  = computeMu(young,nu);
  const auto seq = sigmaeq(sig);
  const auto tmp = A*pow(seq,E-1.);
  const auto df_dseq = E*tmp;
  const auto iseq  = 1/max(seq,eps*young);
  const Stensor n = 3*deviator(sig)*iseq/2;
  //  implicit  system
  feel  += dp*n;
  fp    -= tmp*seq*dt;
  //  jacobian
  dfeel_ddeel += 2*mu*theta*dp*iseq*(Stensor4::M()-(n^n));
  dfeel_ddp    = n;
  dfp_ddeel    = -2*mu*theta*df_dseq*dt*n;
} //  end of @Integrator
```

- Implicit integration.
- Implicit system :

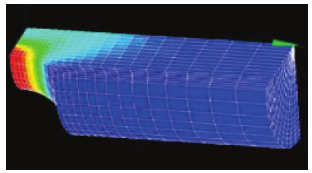
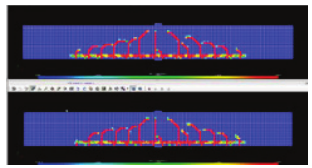
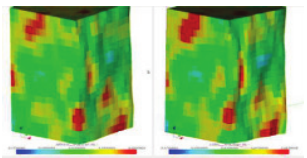
$$\begin{cases} f_{\underline{\epsilon}^{el}} = \Delta\,\underline{\epsilon}^{el} - \Delta\,\underline{\epsilon}^{to} + \Delta\,p\,\underline{\mathbf{n}} \\ f_p = \Delta\,p - A\,\sigma_{eq}^{n} \end{cases}$$

- Jacobian :

$$\begin{cases} \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,\underline{\epsilon}^{el}} = \underline{\underline{\mathbf{I}}} + \dfrac{2\,\mu\,\theta\,\Delta\,p}{\sigma_{eq}}\left(\underline{\underline{\mathbf{M}}} - \underline{\mathbf{n}} \otimes \underline{\mathbf{n}}\right) \\ \dfrac{\partial f_{\underline{\epsilon}^{el}}}{\partial \Delta\,p} = \underline{\mathbf{n}} \\ \dfrac{\partial f_p}{\partial \Delta\,\underline{\epsilon}^{el}} = -2\,\mu\,\theta\,A\,n\,\sigma_{eq}^{n-1}\,\Delta\,t\,\underline{\mathbf{n}} \end{cases}$$

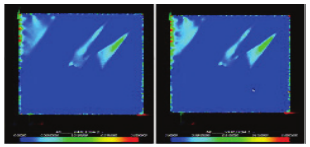
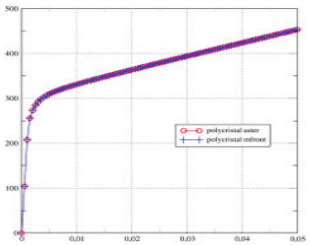
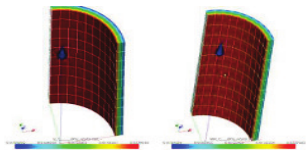
- **All programming and numerical details are hidden (by default).**

Benchmarks with Code_Aster

Description	Algorithme	Temps CPU total (Aster vs MFront)	Illustration
Visco-plastic and damaging for steel (Mustata and Hayhurst 2005; EDF 2012)	Implicit	17*mn*43*s* vs 7*mn*58*s*	
Damaging for concrete (Mazars and Hamon; EDF 2013a)	Default	45*mn* vs 63*mn*	
Generic Single crystal viscoplasticity (Méric and Cailletaud 1991; EDF 2013b)	Implicit	28*mn* vs 24*mn*	

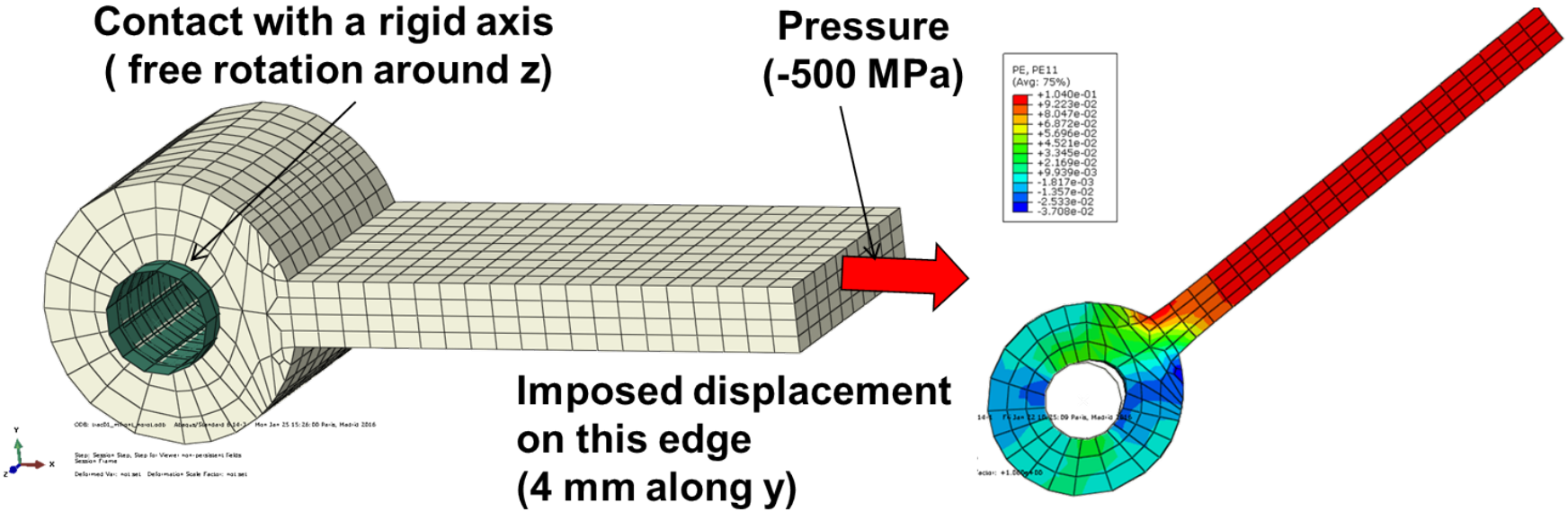
- Benchmarks made in 2013 by the Code_Aster developpers when evaluating MFront.
- Calling external behaviours in Code_Aster has improved...

Benchmarks with Code_Aster

Description	Algorithme	Temps CPU	Illustration
FCC single crystal viscoplasticity (Monnet, Naamane, and Devincre 2011 EDF (2013b))	Implicit	33*m*54*s* vs 29*m*30*s*	
FCC homogeneized polycrystals 30 grains (Berveiller and Zaoui 1978; EDF 2013b)	Runge-Kutta 4/5	9*s*67 vs 8*s*22	
Anisotropic creep with phase transformation (EDF 2013c)	Implicit	180*s* vs 171*s*	

- Benchmarks made in 2013 by the Code_Aster developpers when evaluating MFront.
- Calling external behaviours in Code_Aster has improved...

Benchmarks with Abaqus/Standard



	CPU Time (s)	Difference	Number of increments	Difference
ABAQUS	192	-	52	-
ABAQUS/UMAT(*)	1200	525%	295	467%
ABAQUS/MFRONT	201	5%	52	0%
(*) Numerical Jacobian				

MFRONT seems to be much more efficient
than a badly implemented UMAT!!!

The ZMAT interface

Overview

Un bon code n'est jamais documenté : il a été difficile à écrire, il doit être difficile à comprendre.

F.F.

- To date, the `ZMAT` interface was **the easiest** to build :
  - Thanks to S. Quilicci and J. Besson's help.
  - The interface boils down to about `1500` lines of `C++` code, plus some additional header files for object conversions.
  - No special treatment for plane stress, no handling of finite strain strategies, no local sub-stepping required, orthotropy, etc...
  - Compatibility with `Zebulon` finite strain strategies.
  - We tried to mimic as much as possible the keywords used by "native" `ZMAT` behaviours.
  - Support for small and finite strain behaviours :
    - No support for cohesive zone model.
    - Finite strain are only available for the updated lagrangian formulation.

Generating shared libraries

- `mfront` usage :

```
$ mfront --obuild --interface=zmat norton.mfront
Treating target : all
The following library has been built :
- libZMATBehaviour.so : Norton
```

- The previous command generates :
  - A set of C++ source files.
  - A shared library `libZMATBehaviour.so` containing one `ZMAT` class named `Norton` located in the `src` directory.

Using the generated library

- Use like any other `ZMAT` behaviour :

```
@MaterialProperty stress young;
@MaterialProperty real nu,A,E;
young.setGlossaryName("YoungModulus");
nu.setGlossaryName("PoissonRatio");
A.setEntryName("NortonCoefficient");
E.setEntryName("NortonExponent");
```

```
***behavior Norton
  **model_coef
        YoungModulus 150.e9
        PoissonRatio 0.3
        NortonCoefficient 8.e-67
        NortonExponent 8.2
***return
```

- Standard postprocessing :

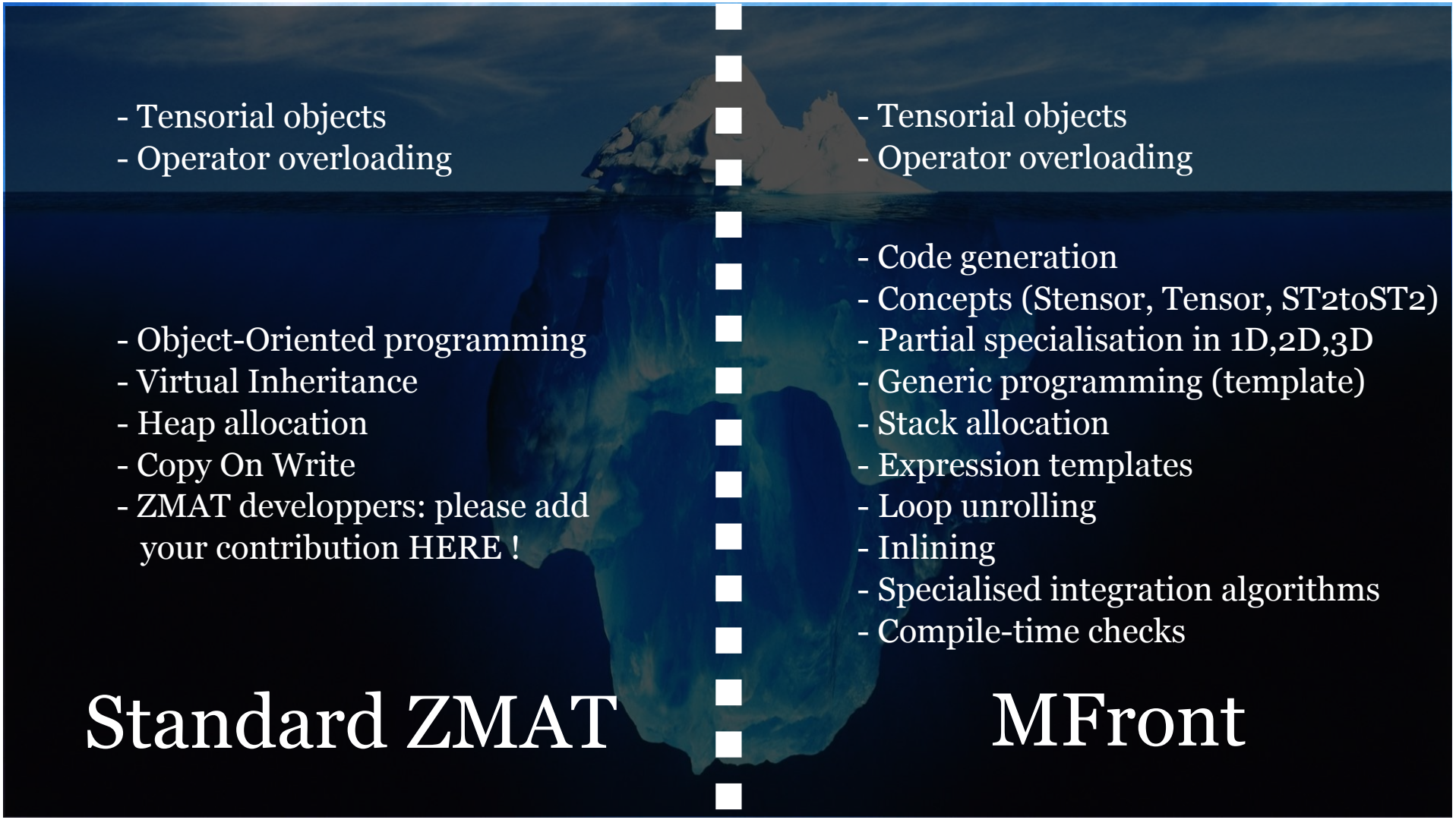
```
@StateVariable strain p;
p.setGlossaryName("EquivalentViscoplasticStrain");
```

```
***output
   **value_at_integration
   **curve
      *gauss_var
         1,1  sig22 eto22 EquivalentViscoplasticStrain
```

- **The temperature must be defined** :

```
***parameter temperature
   0.  uniform 293.15
   2.  uniform 293.15
```

Comparing standard ZMAT behaviours and MFront behaviours



In pratice : the initialisation stage, dimension dispatch

```
void ZMATNorton::initialize(ASCII_FILE& file,int dim,LOCAL_INTEGRATION* integ){
  BEHAVIOR::initialize(file , dim,integ);
  using namespace std;
  int keep_verbose = ZSET::Verbose;
  this->coord.resize(dim);
  this->sig.initialize(this,"sig",this->tsz(),1);
  this->eto.initialize(this,"eto",this->tsz(),1);
  this->tg_mat.resize(this->tsz(), this->tsz());
  //   initialisation   dispatch
  if(this->tsz()==6){
    this->mprops.resize(4);
    this->eel.initialize(this,"ElasticStrain",this->tsz(),1);
    this->p.initialize(this,"EquivalentViscoplasticStrain",1,1);
    for(;;) {
      STRING str=file.getSTRING();
      if(this->base_read(str,file)){
      } else if((str=="**model_coef")||(str=="**material_properties")){
        this->initializeMaterialProperties3D(file);
      } else if(str=="**parameters"){
        this->initializeParameters3D(file);
      } else if(str=="**out_of_bounds_policy"){
        STRING policy=file.getSTRING();
        if(policy=="None"){
          this->obp=tfel::material::None;
        } else if(policy=="Strict"){
```

- Dispatch according to the space dimension.
- Reading the input file is devoted to the behaviour.

In pratice : the integration stage, dimension dispatch

```
INTEGRATION_RESULT* ZMATNorton::integrate(MAT_DATA& mdat,const VECTOR& delta_grad,
                                          MATRIX*& tg_matrix,int flags){
  int  keep_verbose  = ZSET::Verbose;
  CLOCK* keep_clock = ZSET::stored_thread_zbase_globals->ptr()->active_clock;
  tg_matrix = &(this->tg_mat);
  this->set_var_aux_to_var_aux_ini();
  this->set_var_int_to_var_int_ini();
  LIST<EXTERNAL_PARAM*>* ep_save = &EXTERNAL_PARAM::Get_EP_list();
  EXTERNAL_PARAM::set_EP_list(this->zbb_keep_ep);
  if (! this->curr_ext_param){
    this->curr_ext_param = *mdat.param_set();
  }
  this->calc_local_coefs();
  INTEGRATION_RESULT* r = NULL;
  try {
    if ( this->tsz()==6){
      r=this->callMFrontBehaviour3D(mdat,delta_grad,tg_matrix,flags);
    } else  if ( this->tsz()==4){
```

- Dispatch according to the space dimension.
- Error handling through exceptions.

In pratice : the integration stage, in 3D

```
INTEGRATION_RESULT* ZMATNorton::callMFrontBehaviour3D(MAT_DATA& mdat,const VECTOR& delta_grad,
                                                      MATRIX*& tg_matrix,int flags){
  typedef tfel::material::Norton<tfel::material::ModellingHypothesis::TRIDIMENSIONAL,double,false> Norton;

   ....
  auto smflag = Norton::STANDARDTANGENTOPERATOR;
  auto smtype = Norton::NOSTIFFNESSREQUESTED;
  if (flags&CALC_TG_MATRIX){
    smtype = Norton::CONSISTENTTANGENTOPERATOR;
  }
  Norton b(this->sig,stran,dstran,this->mprops,mdat,this->temperature_position,
           this->evs_positions,ZSET::stored_thread_zbase_globals->ptr()->active_clock->get_dtime());
  b.initialize();
  if (b.integrate(smflag,smtype)!=Norton::SUCCESS){
    static INTEGRATION_RESULT bad_result;
    bad_result.set_error(INTEGRATION_RESULT::UNDEFINED_BEHAVIOR);
    return &bad_result;
  }
  b.ZMATexportStateData(this->sig,mdat);
  if (smtype!=Norton::NOSTIFFNESSREQUESTED){
    zmat::ZMATInterface::convert(*tg_matrix,b.getTangentOperator());
  }
  return nullptr;
} // end of ZMATNorton::callMFrontBehaviour3D
```

- Call a class templated by the modelling hypothesis.
- Conversion `ZMAT` $\Leftrightarrow$ `MFront` on input/output.

Conclusions

Current status of the ZMAT interface

- The ZMAT interface is functional :
  - Thanks to S. Quilici and J. Besson.
  - The easiest interface to set up, up to now.
  - Support of small strain and finite strain behaviours.

Current status of the ZMAT interface

- The ZMAT interface is functional :
  - Thanks to S. Quilici and J. Besson.
  - The easiest interface to set up, up to now.
  - Support of small strain and finite strain behaviours.
- **but :**
  - No extensive and systematic testing yet.
  - No benchmarks against native behaviours.
  - No review by ZSet core developers.
  - Some pitfalls :
    - Finite strain behaviours are restricted to updated lagrangian.
    - Orthotropic axes management in $2D$ is not possible, which lead to inconsistent definition of the stiffness tensor, Hill tensor...
    - Some functionalities are missing (e.g. removal of broken elements).

On going efforts

- `MFront` is a building block for the material knowledge management strategy of `EDF` and the `PLEIADES` plateform :
  - Focused of software quality and numerical performances.
- There are ongoing efforts on defining open tools to ensure a fully-consistent strategy encompassing all the steps up to engineering studies :
  - Experimental data.
  - Identification procedures.
  - Behaviour implementations.
  - Unit testing.
  - Documentation.
- The `ZMAT` interface has been made to integrate naturally Phd's work in this workflow.

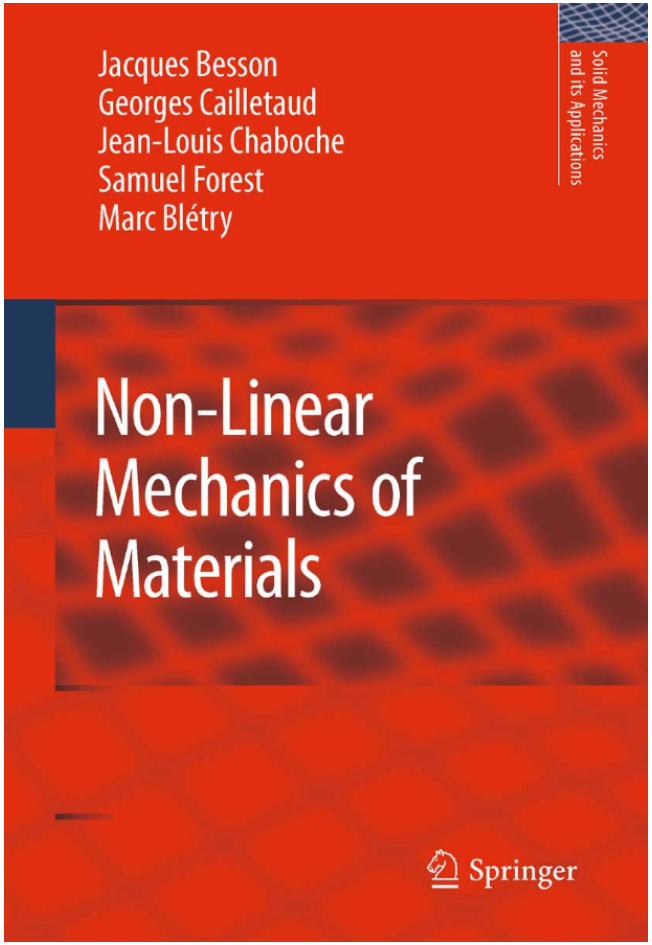
On going efforts

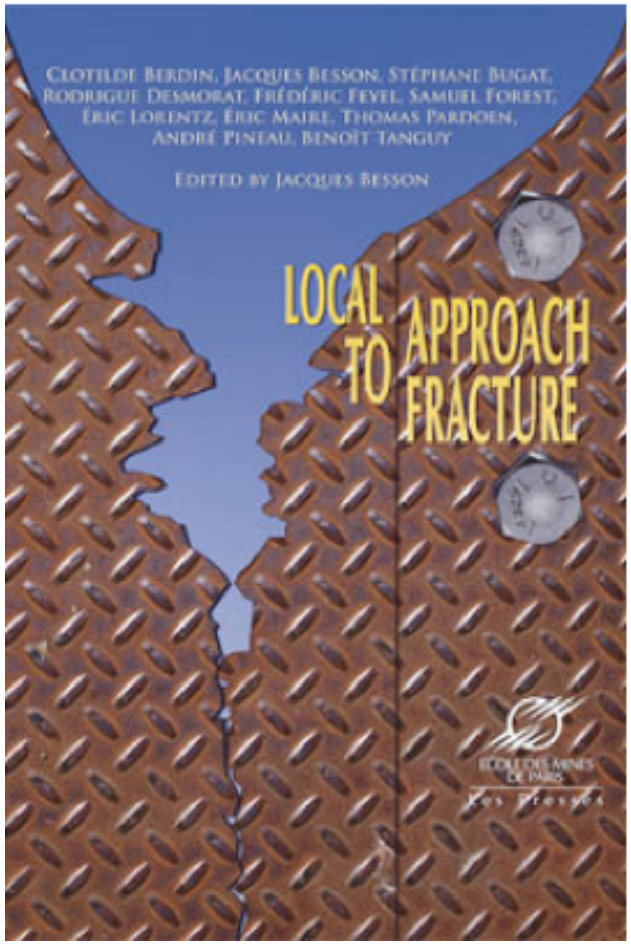
- `MFront` is a building block for the material knowledge management strategy of `EDF` and the `PLEIADES` plateform :
  - Focused of software quality and numerical performances.
- There are ongoing efforts on defining open tools to ensure a fully-consistent strategy encompassing all the steps up to engineering studies :
  - Experimental data.
  - Identification procedures.
  - Behaviour implementations.
  - Unit testing.
  - Documentation.
- The `ZMAT` interface has been made to integrate naturally Phd's work in this workflow.

Future works

- Documentation :
  - Better code documentation.
  - New entries in the gallery.
- Numerical stability tests based on the `CADNA` library :
  - `www.lip6.fr/cadna`
  - `https://github.com/thelfer/cadna`
- Support of units :
  - Compile-time checks without any runtime overhead.
- More work on implicit algorithms :
  - Trust-region algorithms.
- New bricks :
  - Finite strain single crystal (visco-)plasticity.
- Better support of multi-surfaces plasticity.
- Generalised behaviours.
- New interfaces (ANSYS,...).
- An official Debian/Ubuntu package.
- **The third MFront user days !**

A huge technical depth





Thank you for your attention.
Time for discussion !

Commissariat à l'énergie atomique et aux énergies alternatives
Centre de Cadarache | DEN/DEC/SESC b151 - 13108 Saint-Paul-Lez-Durance
T. +33 (0)4.42.25.23.66 | F. +33 (0)4.42.25.47.47
Établissement public à caractère industriel et commercial | RCS Paris B 775 685 019

Direction de l'Énergie Nucléaire
Département d'Études des Combustibles
Service d'Études et de Simulation du comportement des Combustibles