

```
#pragma omp parallel for num_threads(nbt)  
for (int i=0; i<n; i++)
```

Effective development of a finite-element solver at the University

Romain BOMAN,

Luc PAPELEUX,

Jean-Philippe PONTHOT

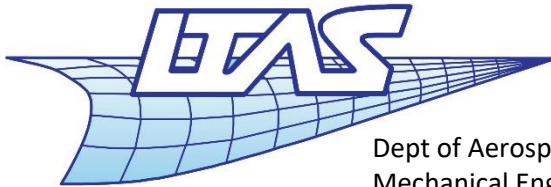
```
int idx2=0;  
for(int nbt=trange.getMin(); nbt<=trange.getMax(); nbt+=trange.getStep())  
{  
    idx2++;  
    double tstart = omp_get_wtime();  
    test.execute(nbt);  
    double tstop = omp_get_wtime();
```

ICSAAM

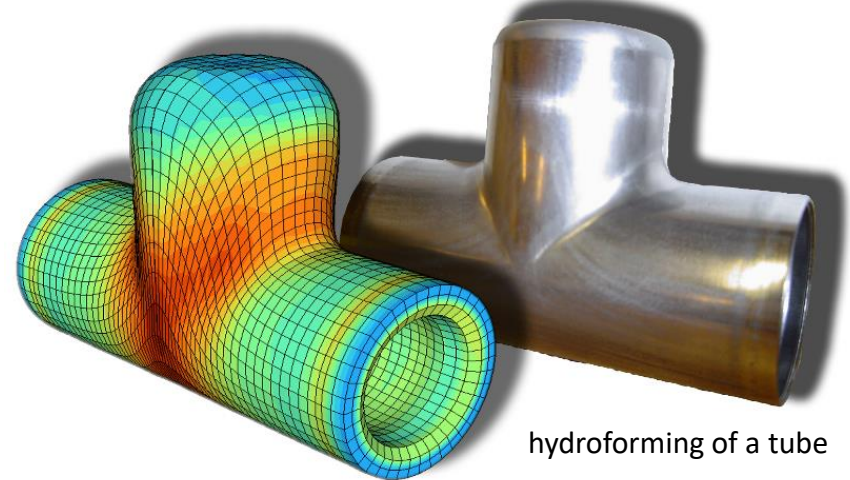
August 28-31, 2018 – Tarbes, France



Our lab within the university



Dept of Aerospace and
Mechanical Engineering



hydroforming of a tube

Computational Mechanics

- Numerical simulation
- Solid mechanics
- Finite element method
- Software development

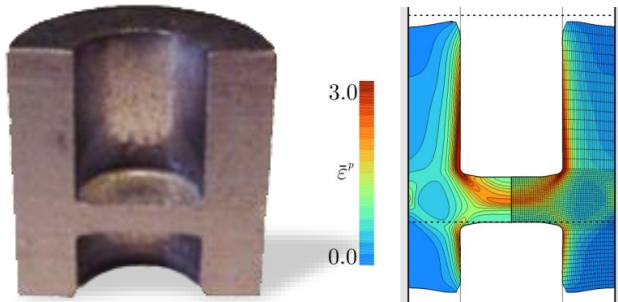


J.-P. Ponthot

Our main simulation code: Metafor

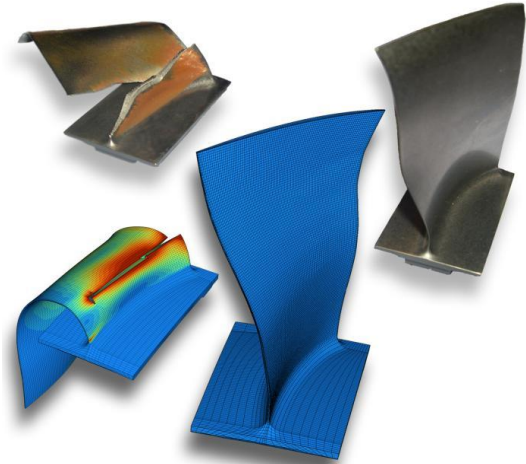
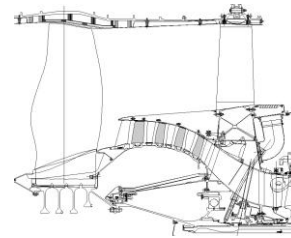
*Implicit Finite-Element solver
for the numerical simulation of large deformations of solids*

Metal Forming applications



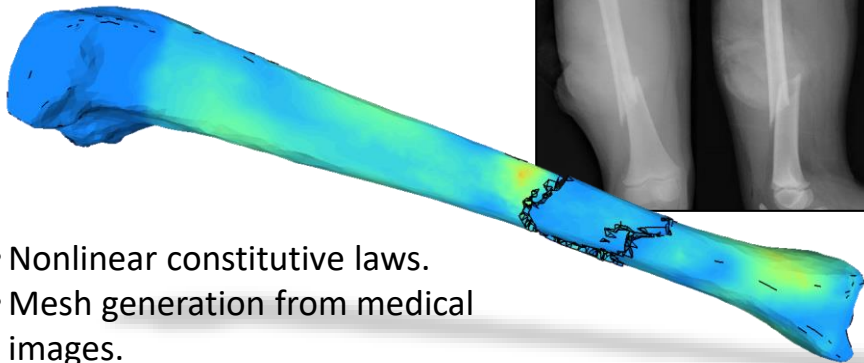
- ALE Formalism, remeshing.
- Thermomechanical time-integration schemes.

Crash / Impact



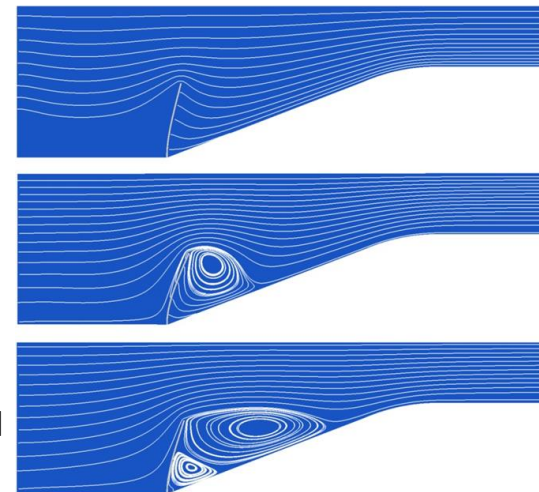
- Modelling of cracks, fracture.
- Contact algorithms.

Biomechanics



- Nonlinear constitutive laws.
- Mesh generation from medical images.

Fluid/structure interaction



- Fluid finite elements.
- Monolithic schemes.
- Coupling with external solvers.

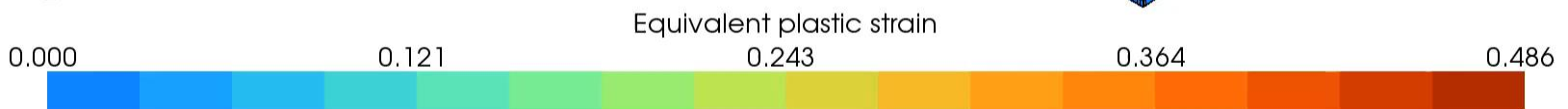
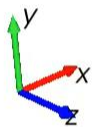
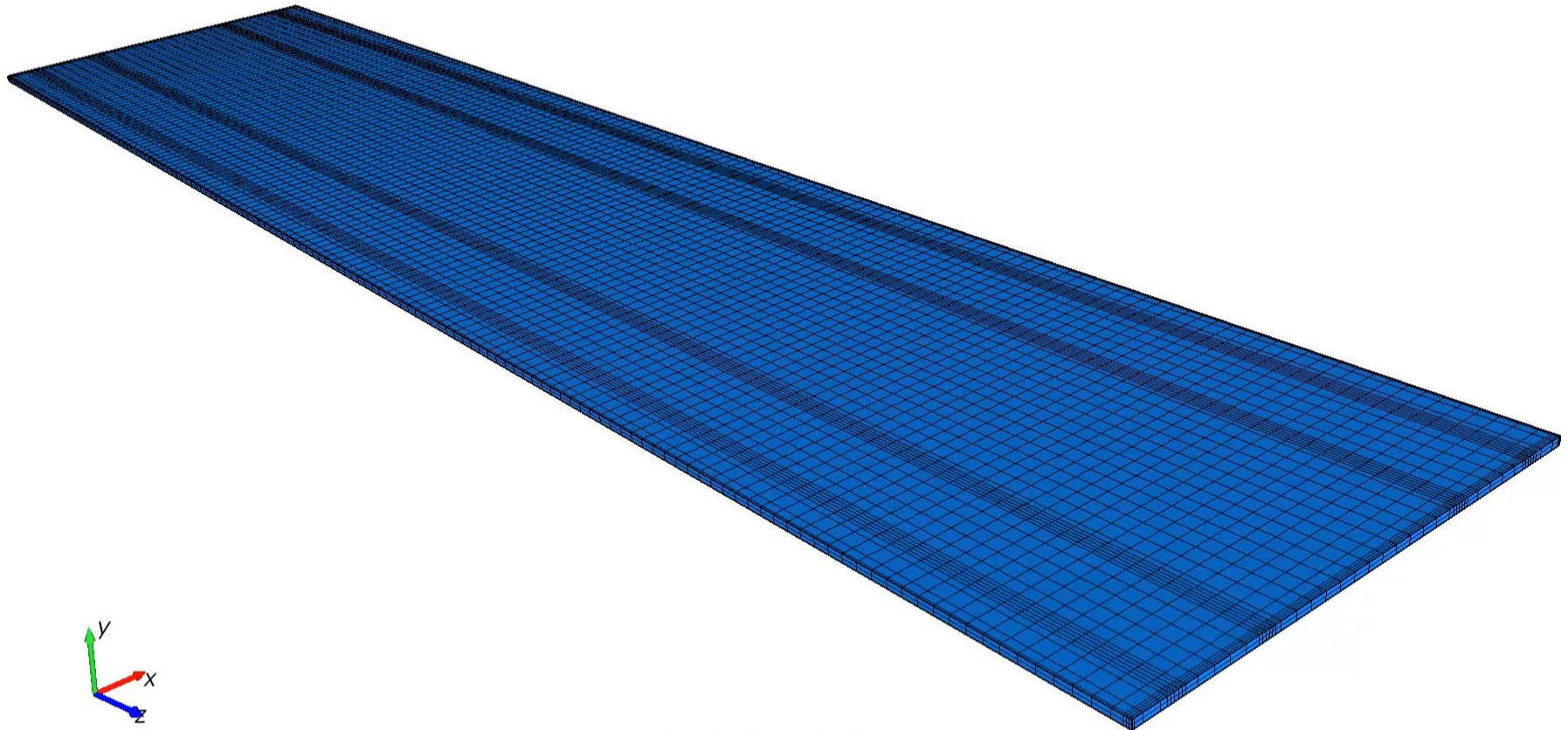
Typical simulation

Modelling of a roll forming mill (forming of beams and tubes from steel sheets)



ArcelorMittal

step 0 t=0/64.872 dt=0.12



Current team



*D. Boemer
(PhD student)*



*R. Boman
(Senior Researcher)*



*M.L. Cerquaglia
(PhD student)*



*L. Papeleux
(Senior Researcher)*



*C. Laruelle
(PhD student)*



*P. Flores
(visiting academic)*

3 PhD students – 2 senior researchers – 1 visiting academic

Former developers



C. Canales



G. Wautelet



Y. Crutzen



G. Deliège



C. Hennuyer



P. Joris



Y. Carretta



A. Stephany



E. Biotteau



C. Laurent



J. Xhardez



L. Vigneron



L. Adam



L. Noels



L. Ziane



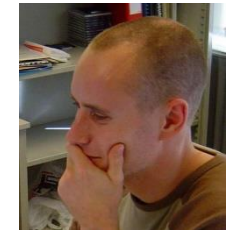
M. Mengoni



O. Karaseva



P. Bussetta



P.-P. Jeunechamps



R. Koeune



S. Hannay



S. Trichon



V.Q. Bui



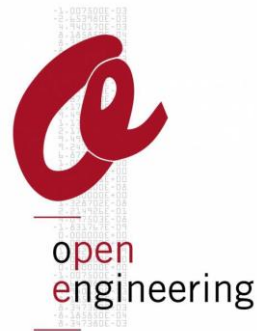
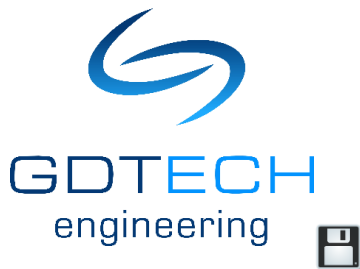
V. d'Otreppe



W. Guo

25 contributors to the present code – from 2000

Industrial partners



 = owns a Metafor license

Scope

1. Practical Management of Simulation Codes

- Metafor... from 1992 to 2018

2. Numerical Applications

- Introduction to ALE formalism
- Thixoforming
- Continuous Roll forming
- Friction Stir Welding
- Additive manufacturing

3. Conclusions

Scope

1. Practical Management of Simulation Codes

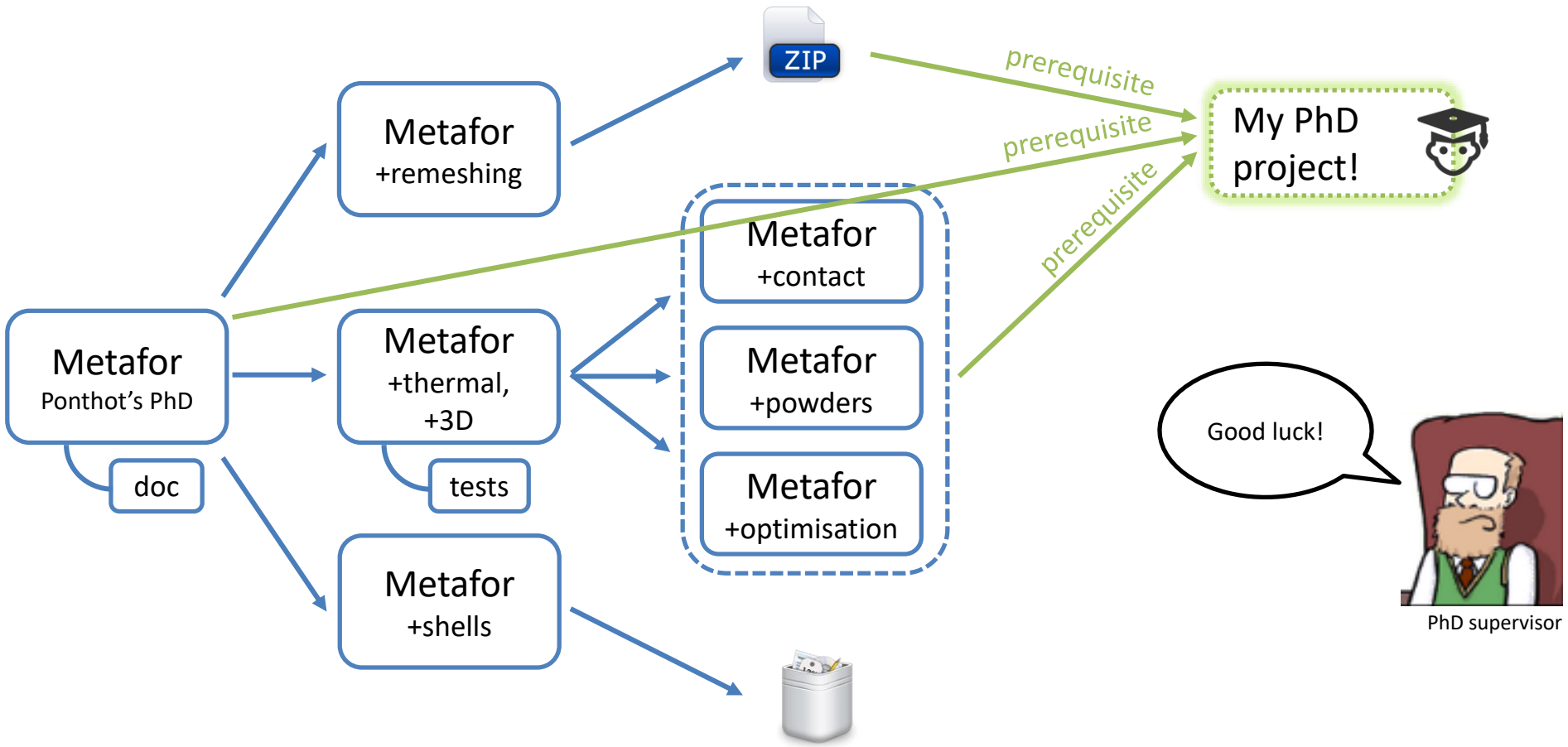
- Metafor... from 1992 to 2018

2. Numerical Applications

- Introduction to ALE formalism
- Thixoforming
- Continuous Roll forming
- Friction Stir Welding
- Additive manufacturing

3. Conclusions

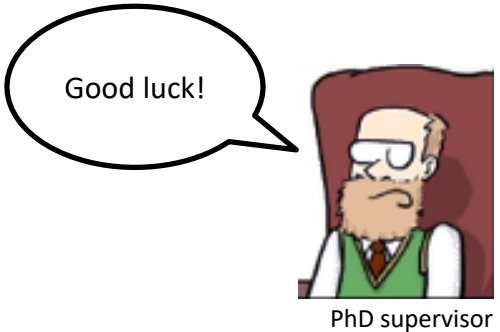
My PhD project in 1997



1992

1996

1997



State of Metafor in 1997

Source Code?

- “Spaghetti” code (Fortran 77)
- Several versions of the code

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Non-automatic and incomplete procedure.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- An outdated technical report from 1992 printed on paper.

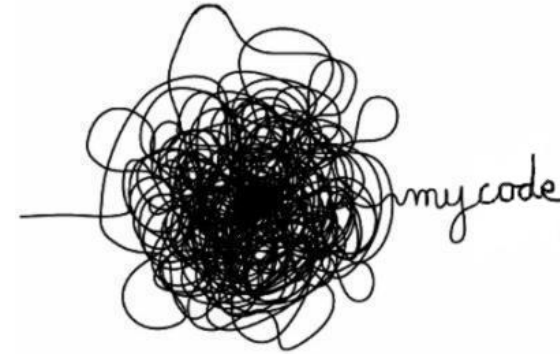
Extensibility?

- None.

...and then?

During 3 years I did **almost no research at all!**

In 2000, I seriously considered stopping my PhD research...



FORTUNATELY,

my supervisor gave me the opportunity to **solve all these problems** and build a **framework** that would allow us to avoid the errors from the past.



One **unique** version
since 2001.

Almost **no loss** of
development since
then!

How?

State of Metafor in 1997

Source Code?

- “Spaghetti” code (Fortran 77)
- Several versions of the code

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Non-automatic and incomplete procedure.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

State of Metafor in 1997

All these problems have **very elegant solutions**
in the “computer science” world

BUT

...we are neither “computer scientists” nor “IT specialists”!

Even worse:

- Some of my colleagues do not want to learn how it works.
- Moreover these things are not taught in class (in ULiège).

I'm here to
“do science”,
not coding!



State of Metafor in 1997

Portability?

For each problem, we have tried to find a **compromise** between both worlds

Welcome to the academic world!

Welcome to the computer science world!



let's publish!



let's code!

characters from <http://geek-and-poke.com/>

printed on paper.

State of Metafor

Source Code?

- “Spaghetti” code (Fortran 77)
- Several versions of the code

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Non-automatic and incomplete procedure.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

State of Metafor

Source Code?

- “Spaghetti” code (Fortran 77)
- Several versions of the code

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Non-automatic and incomplete procedure.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

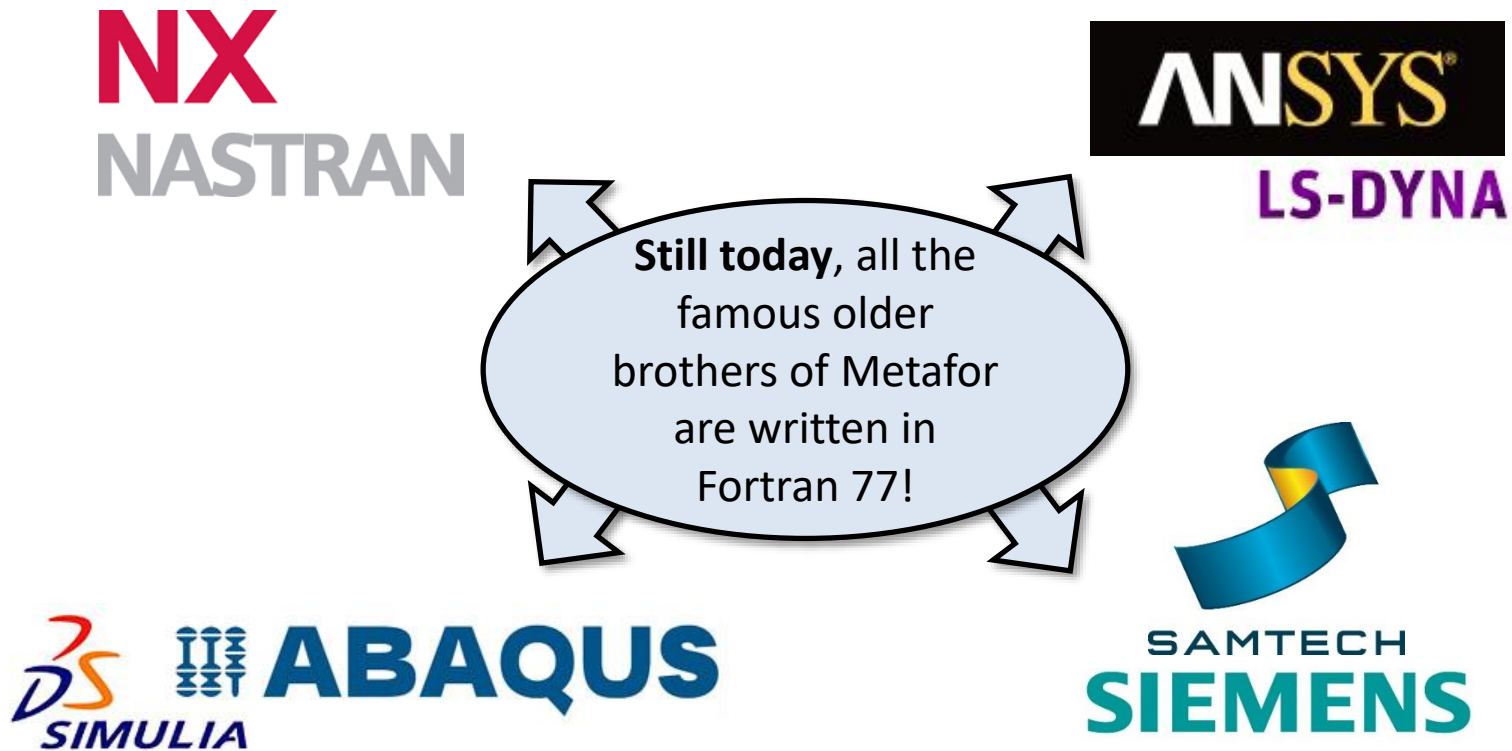
- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

Source code

Very long tradition of **Fortran 77** for implementing the finite element method...



... and a lot of other computation programs.

(cfr BLAS: <http://www.netlib.org/blas/>
MUMPS: <http://mumps.enseeiht.fr/>, etc.)

... also in the industry:



Source code

Main problems of Fortran 77

Yes, it is obviously **possible** to write a good finite-element solver in Fortran...

BUT

- the language does not help you produce clean/maintainable code
- the **maintenance effort** increases with time!

```

CALL META
(S, IREST, LS, INOE, NPRDOM, NTOPDO, P, Q,
X00, X0, X1, X2, LOCSIG, LOCEL, LOCEL2, LOCNOD,
FINT, FEXT, IDENT, ACR, NREMA, T, DEPL, NREAC,
DEPKHI, NREKHI, FIMP, NFI, LOCS, LOCSIT, S(I2),
S(I3), S(I4), DX2, VIT, ACC, DIAMAS, SMS, DJ,
DETJ0V, DETJ1V, SIG0, SIG1, EPL0, EPL1, BETA1,
SVM1, NEPG1, S(I5), AMAT, LOCP, LOCP2, ALOCP,
NGR, ANGR, LOCC, LOCC2, VLOCO, ICONT, CODIR,
RCONT0, RCONT1, LMATER, PMATER, NAEL,
RAY0, RAY1, DRM, MDE, MAINOE, IMPTIME, IMPDDL,
EPS0, EPS1, NOTIN, VALTIN, NFIXT, VALFIX,
LOCLT, LOCLT2, ALOCLT, DETFV0, DETFV1,
F0, F1,
SIGV0, SIGV1, SIGH0, SIGH1,
INTMET, LISNOE, LISMAI, HGQ0, HGQ1,
ALP0, ALP1, ICTM, TT, STPG, HHT, HHTA,
CONMMAS, LIA, LIA2, END0, END1, COEHYP,
DCHYP, NAELCL, DXT, RHO0, RHO1, FTCAP,
AMULA0, AMULA1, FPENAL, XFON, MATMOR,
IDON_INV(1+IPOPT), IDON_INV(1+IPOPT+NXOPT),
RDON_INV(1+IPOPT),
RES_INV(2), RES_INV(2+NPOIOPT), RES_INV(1), TRAV1JP, X1SSB,
X2SSB, SIG1SSB, DETJ1VSSB, EPL1SSB, RAY1SSB, EPS1SSB, END1SSB,
HGQ1SSB, ALP1SSB, DETFV1SSB, F1SSB, RHO1SSB, RCONT1SSB, UEFPLUS,
UEFMOINS, SIG0SSB, DETJ0VSSB, EPL0SSB, RAY0SSB, EPS0SSB,
END0SSB, HGQ0SSB, ALP0SSB, DETFV0SSB, F0SSB, RHO0SSB,
SIGV0SSB, SIGV1SSB, SIGH0SSB, SIGH1SSB,
RCONT0SSB, X0SSB, RESPLUS, X00SSB, PMATERSSB, VLOC0SSB,
IDIFFSSB, DPSSB, VITSSB, ACCSSB, HHTSSB, HHTASSB, STPGSSB,
PSSB, QSSB, X000, XM00, XM0, XM1, XM00SSB,
XM0SSB, XM1SSB, S(I12), S(I8), S(I9), S(I13),
S(I10), S(I11), S(I14), S(I15), S(I6),
S(I7), IROB0, NNZ, DJSSB, UPWTMP,
S(I16), S(I17), S(I18), S(I19),
LOCGAU, LOCELM, LOCNG, MAIADJ, LOCSITG, FCL,
MAIADJ2, LOCGAP, LOCSITP,
S(I20), S(I21), S(I22), S(I23),
NBRNOETFX, LNOETFX, NBRPTSFCT, FIXT_TIME, VALFCT, ALNPSY,
DIAMT, ALNKPSY, ACCLNM1, ACCLN, FINERLN, FINERLNM1, MQLN,
DX2LN, ALPN0, ALPN1, EVPL0, EVPL1, SMALLR0, SMALLR1, V_FCT)

```

Example of a subroutine call in Metafor written in Fortran
(2000)

Source code

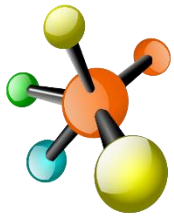
Main problems of Fortran 77



- **Memory management?**

No dynamic memory allocation:

- any modification is very intrusive



- **Modularity?**

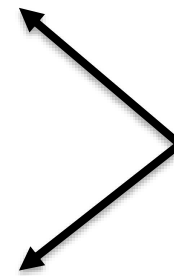
No easy way to gather similar variables/ subroutines together

- structures?



- **Features?**

Necessity to learn another programming language: (graphics, GUI, use of libraries, network, etc.)



partially solved with later versions of Fortran

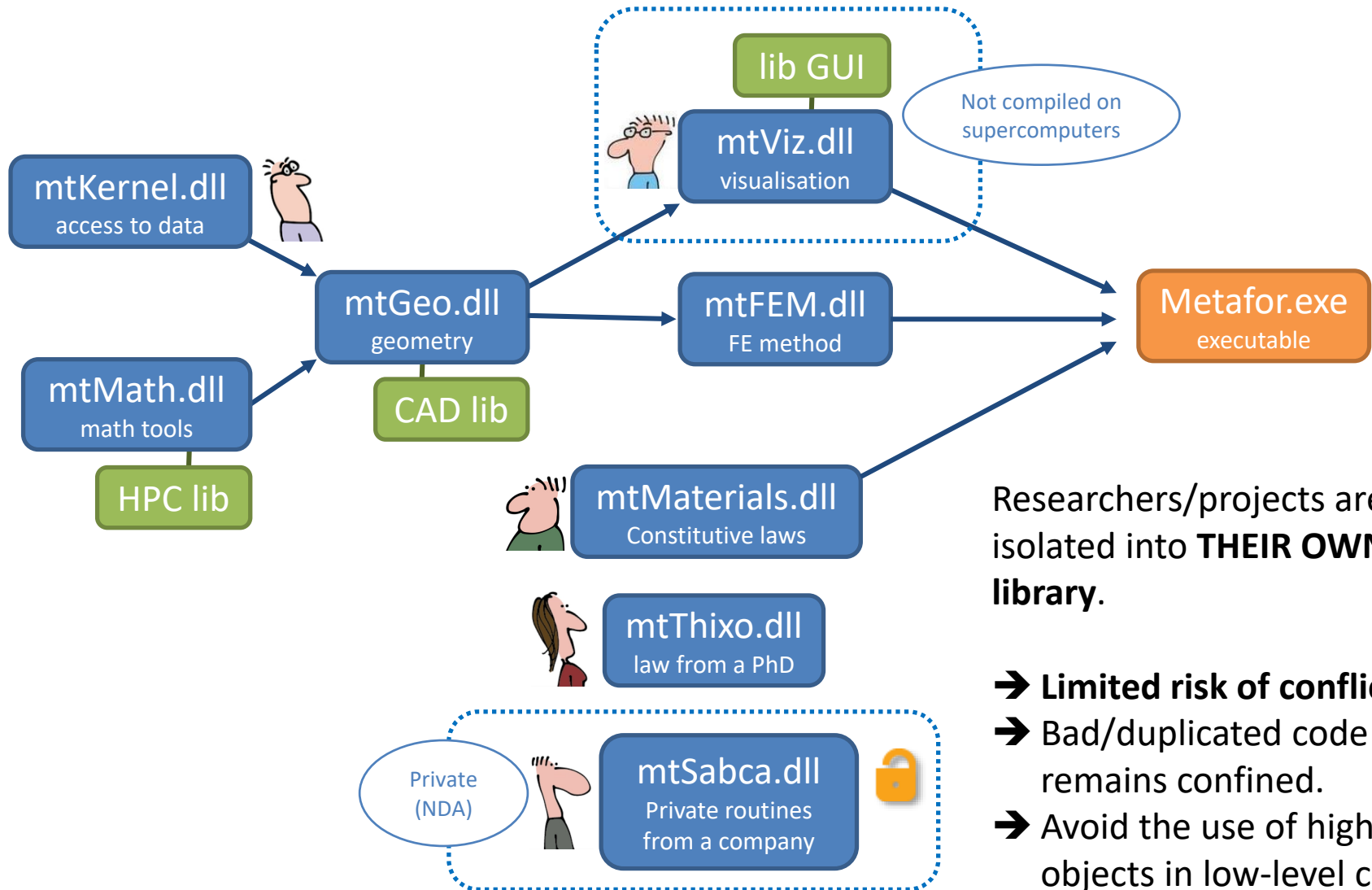


remaining problem

We finally decided to rewrite Metafor in C++ and take advantage of Object Oriented Programming

Source code

The resulting source code (300k lines) can be split into **33 dynamic libraries**

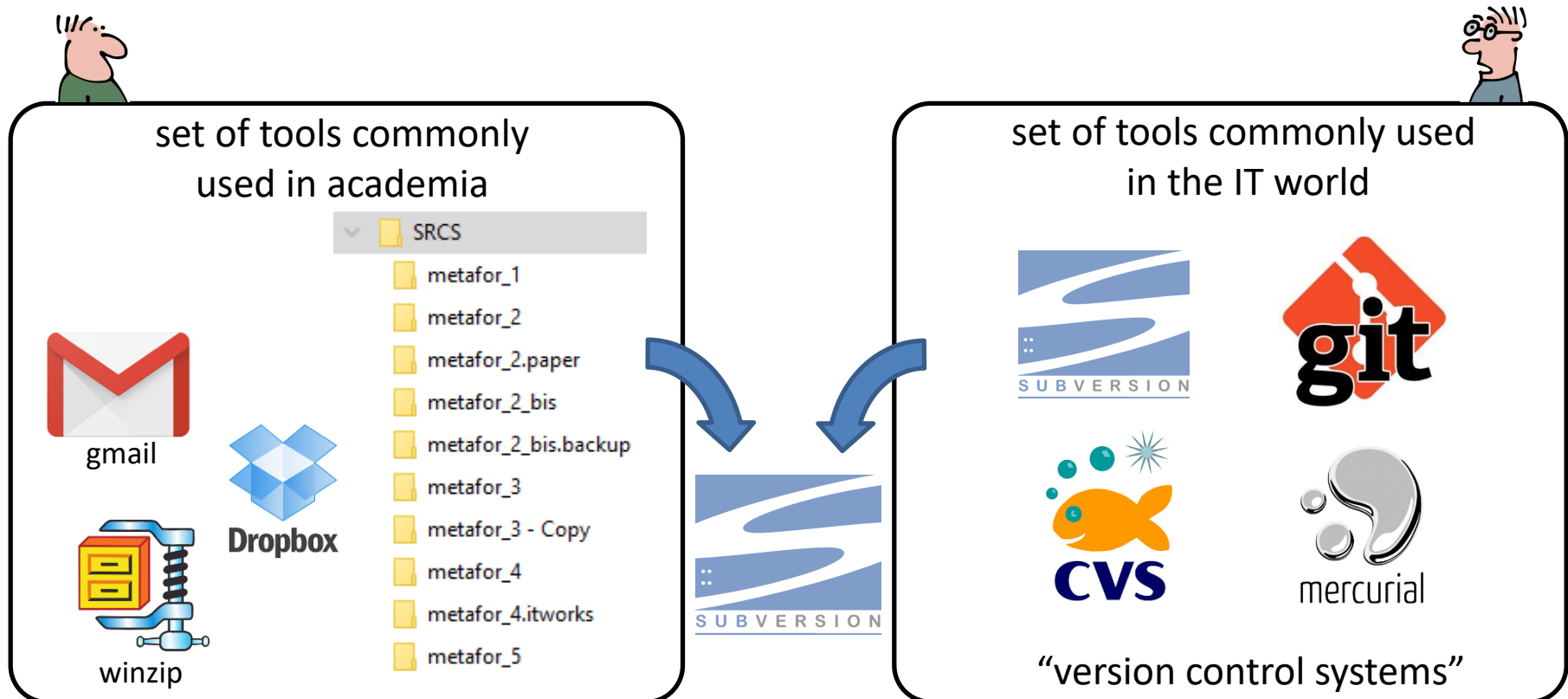


Source Code

How to have one single set of source files?

Our goals

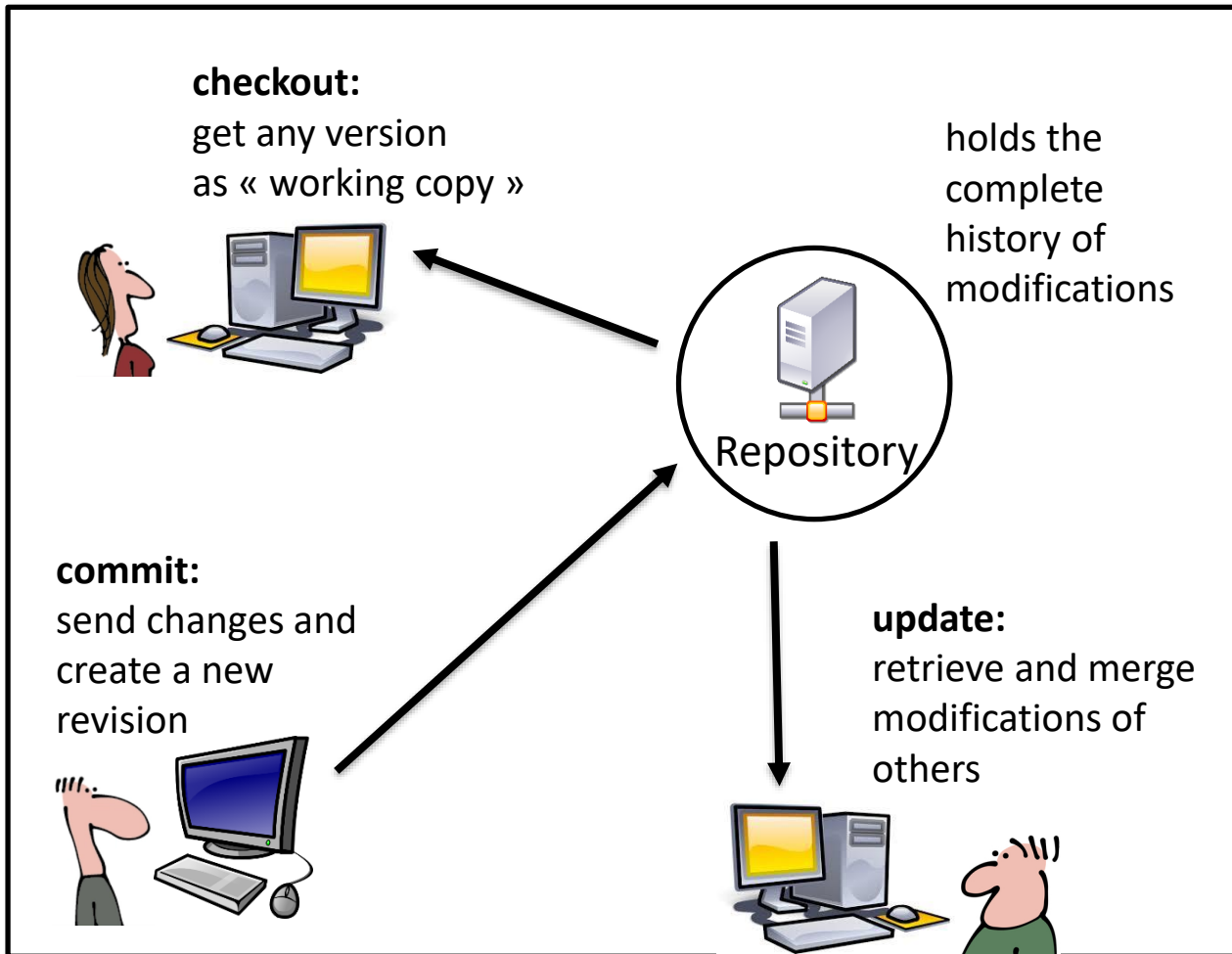
- One single reference version for everybody (4-10 simultaneous researchers).
- Try to gather all the results / code / models from past projects.
- Keep track of the whole history of modifications in the source code.



Source Code

Version control systems - How does it work?

Most basic workflow



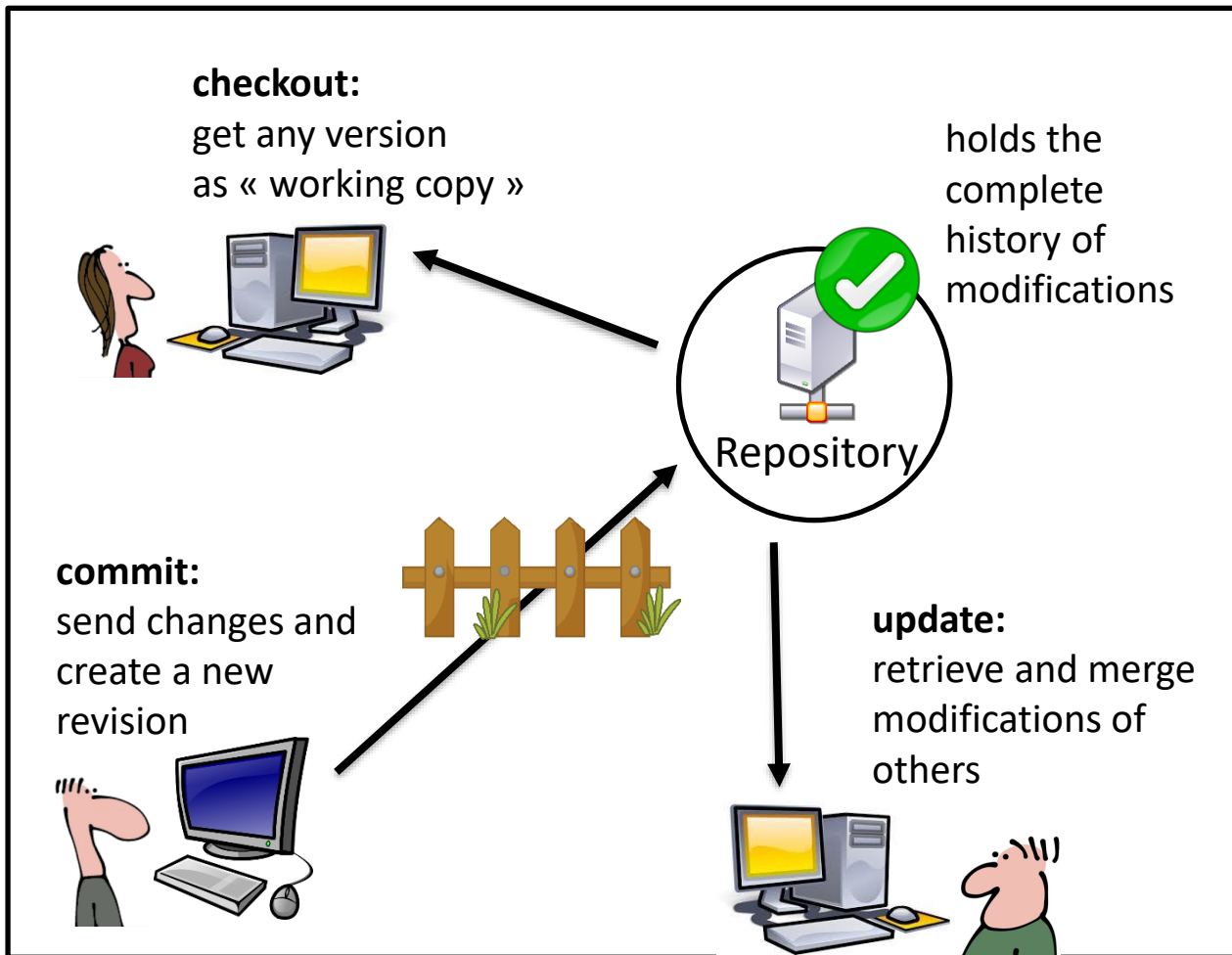
Daily usage should be **very easy** and intuitive because my colleagues **will not read the manual...**

➔ No "branches"

Source Code

Version control systems - How does it work?

Most basic workflow



Additional safeguard:

➔ A series of tests (“**test suite**”) should succeed before any commit.

➔ The latest version is **always stable**.

State of Metafor

Source Code?

- “Spaghetti” code (Fortran 77)
- Several versions of the code

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Non-automatic and incomplete procedure.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Non-automatic and incomplete procedure.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Regression testing?

- Non-automatic and incomplete procedure.

Documentation?

- An outdated technical report from 1992 printed on paper.



Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Input files?

- Home-made input files.
- Error-prone obscure syntax.

Extensibility?

- None.

Regression testing

Aims

- Non regression of the algorithms and the numerical models.
- Avoid conflicts between people.



set of tools commonly
used in academia



manual testing
(when something is already broken)

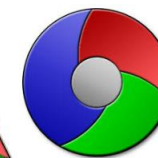


set of tools commonly used
in the IT world

`#!/bin/bash`
shell scripts



CTest



CDash



Travis CI

“continuous integration”

Regression testing

Principles of our “test suite”

- In-house python script.
- Easy procedure – “just run the script!”
- Runs a **series of about 3100 FE models**, in parallel if the machine allows it.
- Lasts less than 1 night on a classical desktop PC (~75000s CPU = ~5h30 on a 4-core PC).
- 4 configurations (various OS/compiler):
- Each test extracts “well chosen” results that are compared to the results of the previous version of the code (these results are committed with the source code.)



2 simple rules



1. It is forbidden to commit without a full success of the test suite.
2. Any untested development can be destroyed (adding tests becomes mandatory).

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Regression testing?

- Non-automatic and incomplete procedure.

Documentation?

- An outdated technical report from 1992 printed on paper.



Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Input files?

- Home-made input files.
- Error-prone obscure syntax.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

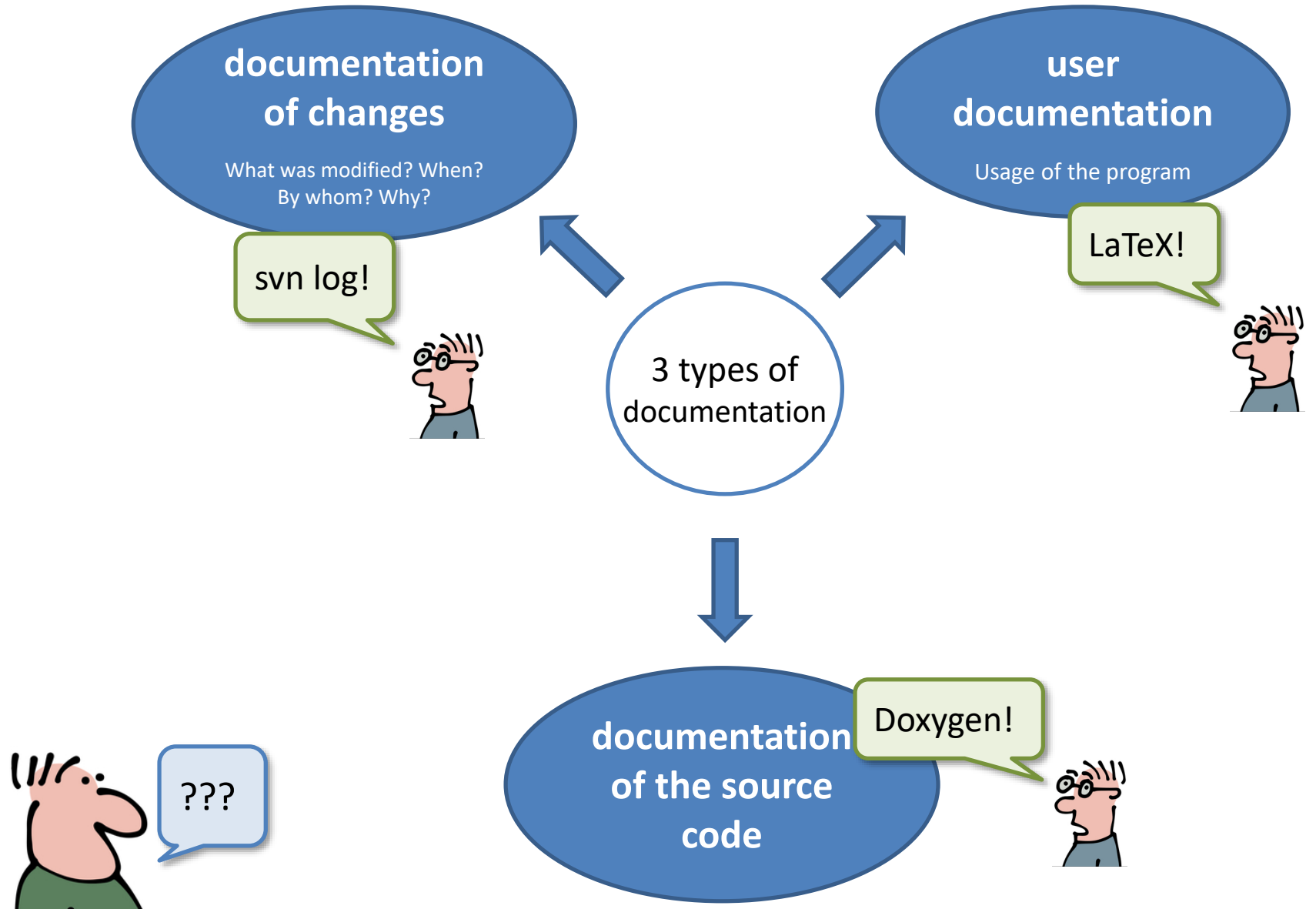
Documentation?

- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

Documentation



Documentation



Wiki system: *Dokuwiki*

<http://metafor.ltas.ulg.ac.be/>

<https://www.dokuwiki.org/>

Strong points

- Created 14 years ago.
- Very easy installation (text-based DB).
- Numerous plugins.
- Regular / automatic updates.
- ... and no need to read a long manual to edit a page.

The screenshot shows a web browser displaying the documentation page for Metafor. The page title is "metafor.ltas.ulg.ac.be". The main content area features a large image of a fan-like structure with the word "Metafor" written across it. Below the image, there is a list of features included in the code:

- 2D/3D elements for large strains analysis (SRI, EAS).
- Implicit/explicit time integration (HHT, Chung Hulbert, ...).
- Thermomechanical coupling (staggered or fully coupled schemes).
- Frictional contact between deformable bodies or analytical surfaces.
- Arbitrary Lagrangian Eulerian formalism.
- Meshing and remeshing procedures.
- Large set of constitutive laws (thermo-elasto-visco-plastic, damage, ...).
- Crack propagation (erosion method).

The page also includes a navigation menu on the left with links for Home, Documentation, Research team, Applications, Publications, Research Projects, and Developer Zone. At the bottom, there are social media links for Facebook and Google+, and a footer with "start.txt · Last modified: 2016/03/30 15:23 (external edit)".

Documentation

As any wiki:

- Online editing.
- Keep track of all changes.
- Importation of images, videos, etc.

Online view

Online editor

Edit the page and hit Save. See [Formatting Syntax](#) for Wiki syntax. Please edit the page or test some things, learn to make your first steps on the [playground](#).

Rich text editor interface showing the following content:

=== Description ===

The damage tensor is denoted D

$$\dot{D} = \left(\frac{\tilde{\sigma}_{eq}^2 R_\nu}{2ES} \right)^* |D^p| \sqrt{\varepsilon^p}$$

where $|D^p|$ is a tensor with the same eigenvectors as D^p , and eigenvalues equal to the absolute value of D^p eigenvalues. The triaxiality function is defined as :

$$R_\nu = \frac{2}{3}(1 + \nu) + 3(1 - 2\nu) \left(\frac{p}{\sigma_{eq}} \right)^2$$

where p is the pressure and σ_{eq} Von Mises stress

Buttons: Save, Preview, Cancel, Edit summary, Minor Changes

doc/user/elements/Volumes/continuousanisodamage.txt · Last modified: 2016/03/30 15:23 (edit)

Online view of the documentation page for 'Continuous orthotropic damage'. The page includes a navigation sidebar, a table of contents, and the main content area with the following text:

Continuous orthotropic damage

The ContinuousAnisoDamage class manages the continuous orthotropic damage evolution laws. When defining a new law, the evolution of the damage variable δH must be defined, and so must be its derivatives with respect to pressure, plastic strain and damage.

Laws implemented in Metafor

AnisoDamageDummy

A dummy testing all possible variations of the damage variable.

LemaitreChabocheContinuousAnisoDamage

Anisotropic extension of Lemaitre isotropic damage law

Description

The damage tensor is denoted D

$$\dot{D} = \left(\frac{\sigma_{eq}^2 R_\nu}{2ES} \right)^* |D^p| \text{ if } \varepsilon^p > \varepsilon_D^p$$

where $|D^p|$ is a tensor with the same eigenvectors as D^p , and eigenvalues equal to the absolute value of D^p eigenvalues. The triaxiality function is defined as :

$$R_\nu = \frac{2}{3}(1 + \nu) + 3(1 - 2\nu) \left(\frac{p}{\sigma_{eq}} \right)^2$$


Print



Continuous orthotropic damage

The ContinuousAnisoDamage class manages the continuous orthotropic damage evolution laws. When defining a new law, the evolution of the damage variable δH must be defined, and so must be its derivatives with respect to pressure, plastic strain and damage.

Laws implemented in Metafor

AnisoDamageDummy

A dummy testing all possible variations of the damage variable.

LemaitreChabocheContinuousAnisoDamage

Anisotropic extension of Lemaitre isotropic damage law

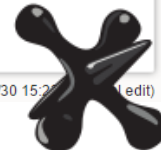
Description

The damage tensor is denoted D

$$\dot{D} = \left(\frac{\sigma_{eq}^2 R_\nu}{2ES} \right)^* |D^p| \text{ if } \varepsilon^p > \varepsilon_D^p$$

where $|D^p|$ is a tensor with the same eigenvectors as D^p , and eigenvalues equal to the absolute value of D^p eigenvalues. The triaxiality function is defined as :

$$R_\nu = \frac{2}{3}(1 + \nu) + 3(1 - 2\nu) \left(\frac{p}{\sigma_{eq}} \right)^2$$



javascript rendering engine for equations: **MathJax**

Documentation

Documentation of source code changes (“commits”)

Commits [Metafor] x

Non sécurisé | metafor.ltas.ulg.ac.be/dokuwiki/commit/start

Applications Bkm

2000-2001

2018

Date	Version	Auteur	Résumé
7 Août 2018	3224	DB	Traction at trans ContactTraction
1 Août 2018	3220	RoBo	Surroundedness
27 Juillet 2018	3216	Luc	Bug Fix Thermic
23 Juillet 2018	3211	Marco	Center of gravity
19 Juillet 2018	3206	Luc	refactoring Ther
17 Juillet 2018	3202	DB	Out-of-plane thi area computatio
13 Juillet 2018	3198	DB	New ContactTr
09 Juillet 2018	3194	Luc	bug fix YieldGp
05 Juillet 2018	3190	Luc	YieldGpState
03 Juillet 2018	3186	DB	New coupling pr
25 Juin 2018	3182	Luc	VeloHyperDv2



Each commit is fully documented by its author on the wiki.

commit:2018:08_17

Table of Contents

Commit August 17th, 2018

Previously, in commit 3198, the `ContactTractionElement`, which sets the resultant traction forces equal to zero at the nodes that are in contact, was introduced. Setting the traction forces equal to zero might, however, be incoherent at a node belonging to an element at the transition from “no contact” to “contact”, since the majority of the element is still stressed by the traction as can be seen in the following figure (red triangle). For this reason, an additional property was added to the `ContactTractionElement`, i.e. `TRACTION_AT_TRANSITION_ON`. In the past, the `ContactTractionElement` computed the forces as in the case `TRACTION_AT_TRANSITION_ON = False`. In the other case, the traction force is not set equal to zero, if the `ContactTractionElement` is located at the transition from “no contact” to “contact” (see red arrow below). The new default case is `TRACTION_AT_TRANSITION_ON = True`.

`TRACTION_AT_TRANSITION_ON = False` `TRACTION_AT_TRANSITION_ON = True`

Resultant pressure force Applied pressure Contact tool

Test cases

Edit

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- An outdated technical report from 1992 printed on paper.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Regression testing?

- Mandatory.
- In-house script.

Documentation?

- A wiki system on the web.



Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Input files?

- Home-made input files.
- Error-prone obscure syntax.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- A wiki system on the web.

Extensibility?

- None.

Portability

Rule: The code should be easily built/tested on any (recent) platform



- ✓ Most popular OS
- ✗ Libraries should be recompiled



- ✓ Popular OS
- ✗ Libraries should be recompiled



- ✓ OS of supercomputers
- ✗ Less user-friendly



It is very important to let developers work in their preferred environment (which is rarely Linux), then build their code on supercomputers for heavy calculations.



Main problems

- the build system should be multi-platform (CMake).
- the dependencies should be limited and multi-platform.

Visualisation

Example: Real-time visualisation during the simulations

Initially, 2 aims

- To see what we are doing, anytime.
- To make the software usage easier for students and the industry.

Features

- 1 thread exclusively dedicated to graphics running in parallel.
- The state of the memory is continuously displayed in 3D.
- Nothing is written to disk.

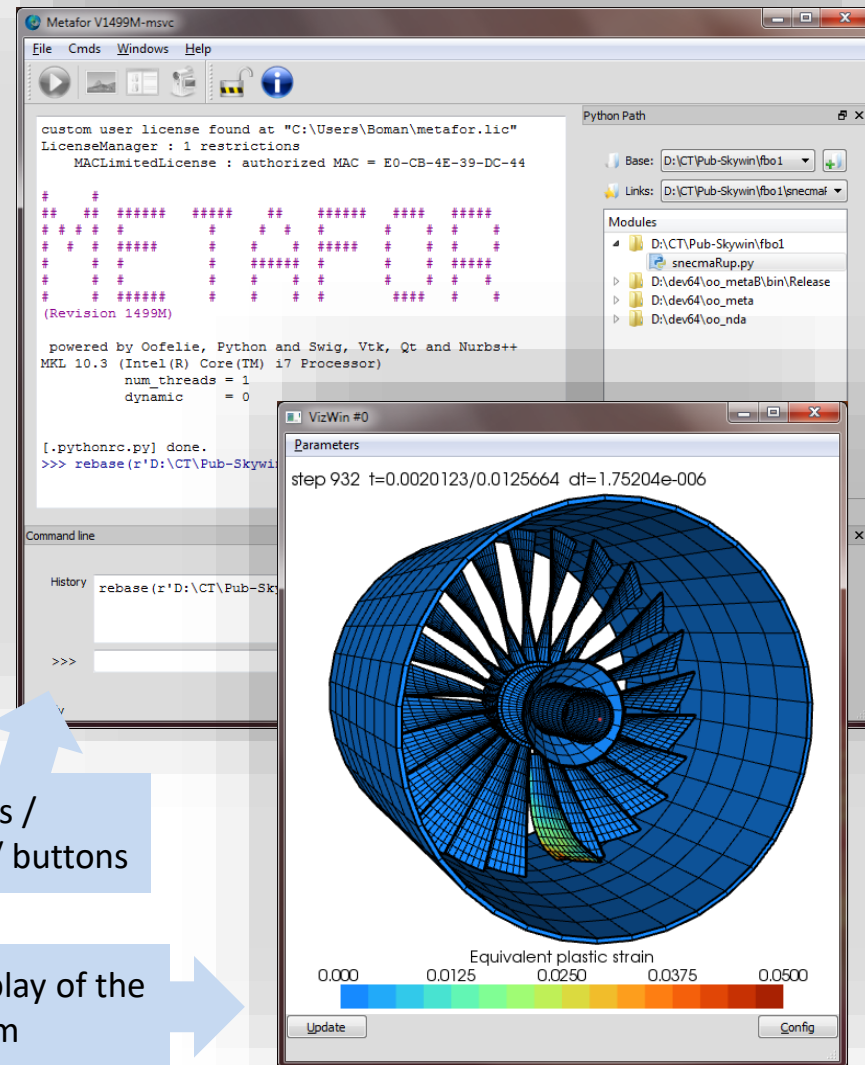
Multi-platform libraries:



windows /
menus / buttons



3D display of the
problem



State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Depends on a library only available as binaries on supercomputers.
- Post-processing with a non-free product.

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- A wiki system on the web.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Runs on any recent desktop computer and supercomputers.
- Own GUI / post-processor,

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- A wiki system on the web.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Runs on any recent desktop computer and supercomputers.
- Own GUI / post-processor,

Regression testing?

- Mandatory.
- In-house script.

Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- A wiki system on the web.

Extensibility?

- None.



Input files



Python scripts as input files

old home-made syntax

```
.poi
i 1 x 0.0 y 0.0
i 2 x 1.0 y 0.0
i 3 x 1.0 y 1.0
i 4 x 0.0 y 1.0

.cur
i 1 points 1 2
i 2 points 2 3
i 3 points 3 4
i 4 points 4 1
```



python syntax

```
geo = domain.getGeometry()

p1 = geo.add(Point(1, 0.0, 0.0))
p2 = geo.add(Point(2, 1.0, 0.0))
p3 = geo.add(Point(3, 1.0, 1.0))
p4 = geo.add(Point(4, 0.0, 1.0))

c1 = geo.add(Line(1, p1, p2))
c2 = geo.add(Line(2, p2, p3))
c3 = geo.add(Line(3, p3, p4))
c4 = geo.add(Line(4, p4, p1))
```

- Parsing requires a lot of code.
- Error handling?
- New syntax to learn.

- Almost no source code at all!
- Error handling by python.
- Same syntax as internal C++.

Input files



Python scripts as input files

old home-made syntax

```
.poi
i 1 x 0.0 y 0.0
i 2 x 1.0 y 0.0
i 3 x 1.0 y 1.0
i 4 x 0.0 y 1.0

.cur
i 1 points 1 2
i 2 points 2 3
i 3 points 3 4
i 4 points 4 1
```

python syntax

```
geo = domain.getGeometry()

def createCube(no, Lx, Ly, ox=0.0, oy=0.0):
    p1 = geo.add(Point(1, ox, 0.0))
    p2 = geo.add(Point(2, ox+Lx, 0.0))
    p3 = geo.add(Point(3, ox+Lx, oy+Ly))
    p4 = geo.add(Point(4, ox, oy+Ly))

    c1 = geo.add(Line(1, p1, p2))
    c2 = geo.add(Line(2, p2, p3))
    c3 = geo.add(Line(3, p3, p4))
    c4 = geo.add(Line(4, p4, p1))

for i in range(5):
    createSquare(no=i, Lx=1.0, Ly=1.0, ox=i*2)

import myroutines
myroutines.createBeam()
```

"for" loop

call to external
modules

call to a function

parameters

What if I want to add:

- parameters?
- functions?
- branches, loops?
- ...

- all these features come at no cost!

Input files



No cost? Really?

Yes! a Python class is **automatically** created by SWIG (free tool) for each C++ class.

materials.i

```
%module materials
%{
#include "ElasticMat.h"
%}

#include "ElasticMat.h"
```


SWIG
Simplified Wrapper and
Interface Generator

_materials.pyd

materials.py

ElasticMat.h

```
class ElasticMat
{
    double E, nu;
public:
    ElasticMat(double, double);
    T computeStress(T &strain);
    ...
};
```




INPUT FILE

```
from materials import *

mat = ElasticMat(E, nu)

model.setMaterial(mat)
model.run()
```



State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Runs on any recent desktop computer and supercomputers.
- Own GUI / post-processor,

Regression testing?

- Mandatory.
- In-house script.

Input files?

- Home-made input files.
- Error-prone obscure syntax.

Documentation?

- A wiki system on the web.

Extensibility?

- None.



State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Runs on any recent desktop computer and supercomputers.
- Own GUI / post-processor,

Regression testing?

- Mandatory.
- In-house script.

Input files?

- Python.

Documentation?

- A wiki system on the web.

Extensibility?

- None.



State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Runs on any recent desktop computer and supercomputers.
- Own GUI / post-processor,

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Python.

Documentation?

- A wiki system on the web.

Extensibility?

- None.

Extensibility

1. Extensibility with “Inheritance” in C++

- In any object-oriented language, you can extend a class using “inheritance”.
- Example: extension of an elastic material to handle elasto-plasticity:

ELASTIC MATERIAL

```
class ElasticMat
{
    double E, nu;
public:
    virtual
    T computeStress(T &strain);
};
```

ELASTO PLASTIC MATERIAL

```
class ElPlastMat : public ElasticMat
{
    double sig_yield, hardening;
public:
    virtual
    T computeStress(T &strain);
};
```

Purpose?

- The code using the material can be written without any conditional statements and should never be modified.
 - ➔ “if the material is elastic, do this.... else, do that” is avoided.
- But, of course, the new code should be recompiled.


Extensibility

2. Inheritance of C++ classes in Python : “user subroutines”

Thanks to SWIG, the derived class can be written in Python!

ELASTIC MATERIAL (written in C++)


```
class ElasticMat
{
    double E, nu;
public:
    virtual
    T computeStress(T &strain);
};
```



This python code will be called from C++!

ELASTO-PLASTIC MATERIAL (derived in Python)

```
from materials import *
class ElPlastMat(ElasticMat):
    def computeStress(self, strain):
        # compute stresses
        # from strains
        # using python cmds here..
        return stress
```



A new constitutive law is available without any compiler!

✓ Very useful for students (FYP)

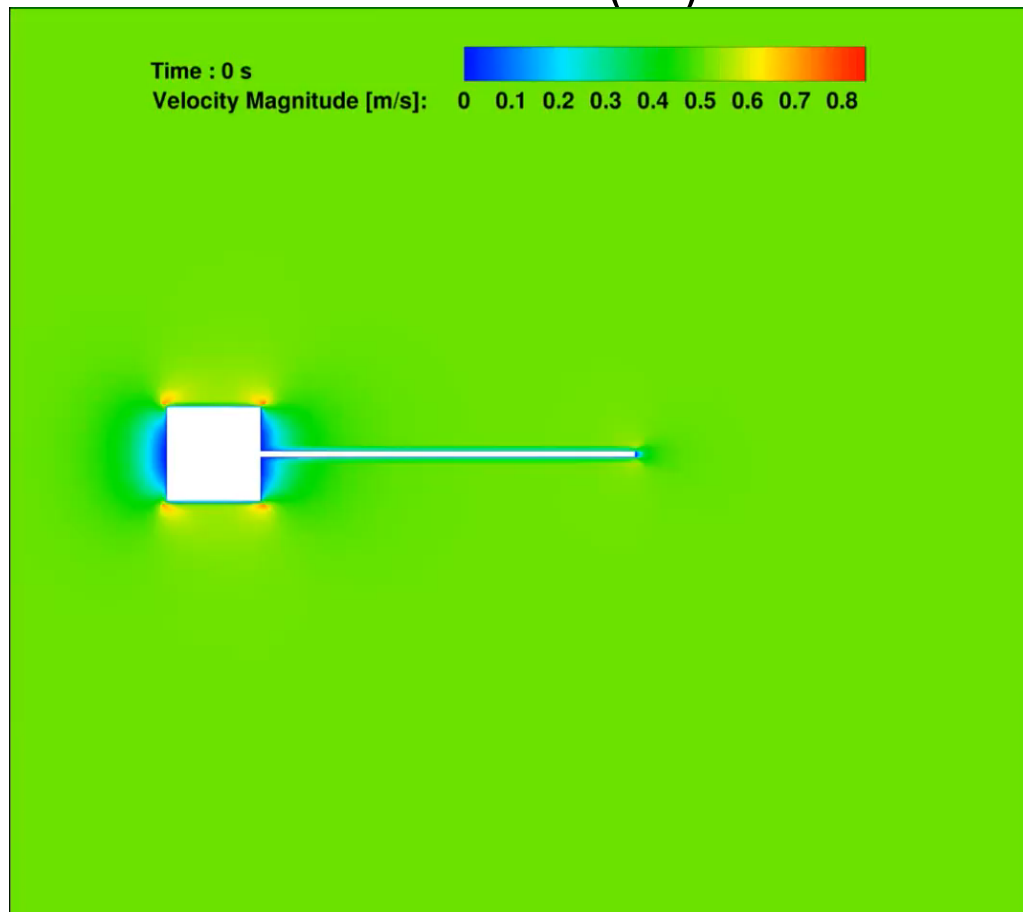
Extensibility

3. Extensibility via “python as a glue language”: coupling with another solver



David Thomas

fluid-structure interaction (FSI)



Prof. J.-P. Ponthot's lab

metafor.py

Metafor
Solid solver (C++)



fluid –
structure
solver



Prof. V. Terrapon's lab

SU2.py

SU2
Fluid solver (C)

Extensibility

Summary – Advantages of the Python interface

- **Less source code:** no need to code a home-made parser,
- **Safety:** errors are never ignored and can be correctly handled.
- **Complete Language:** possibility to use loops, conditional statements, objects, functions, ... in the input file.
- **Extensibility:** inheritance of C++ classes.
- **Glue language:** call to external programs, solvers.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Runs on any recent desktop computer and supercomputers.
- Own GUI / post-processor,

Regression testing?

- Mandatory.
- In-house script.



Input files?

- Python.

Documentation?

- A wiki system on the web.

Extensibility?

- None.

State of Metafor

Source Code?

- Modular (C++).
- Source Code Management with Subversion.

Portability?

- Runs on any recent desktop computer and supercomputers.
- Own GUI / post-processor,

Regression testing?

- Mandatory.
- In-house script.

Input files?

- Python.

Documentation?

- A wiki system on the web.

Extensibility?

- Python user-subroutines.
- Coupling with external solvers.



Scope

1. Practical Management of Simulation Codes

- Metafor... from 1992 to 2018

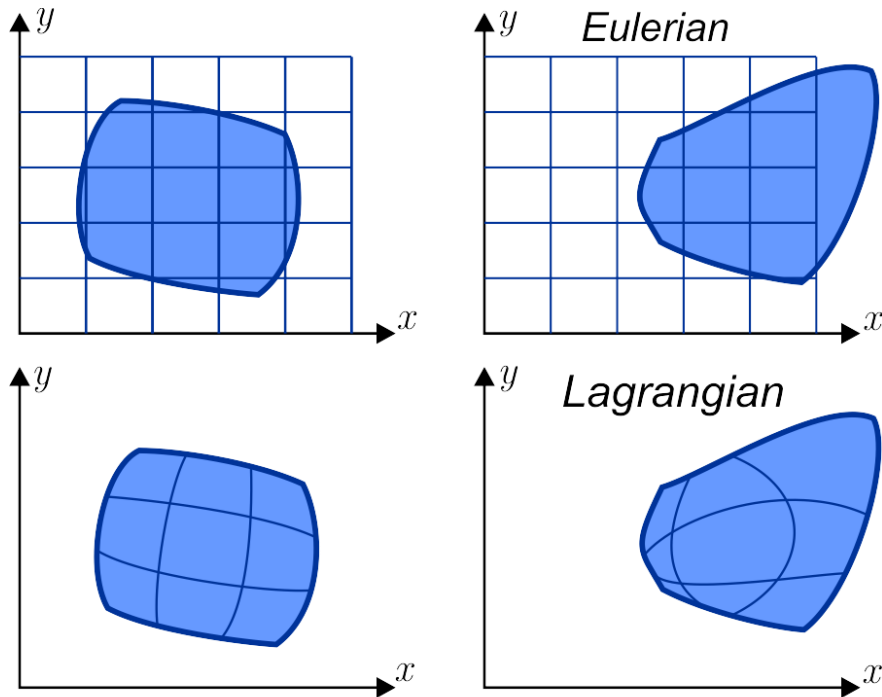
2. Numerical Applications

- Introduction to ALE formalism
- Thixoforming
- Continuous Roll forming
- Friction Stir Welding
- Additive manufacturing

3. Conclusions

Introduction to ALE formalism

Kinematic description of the motion



Eulerian formalism

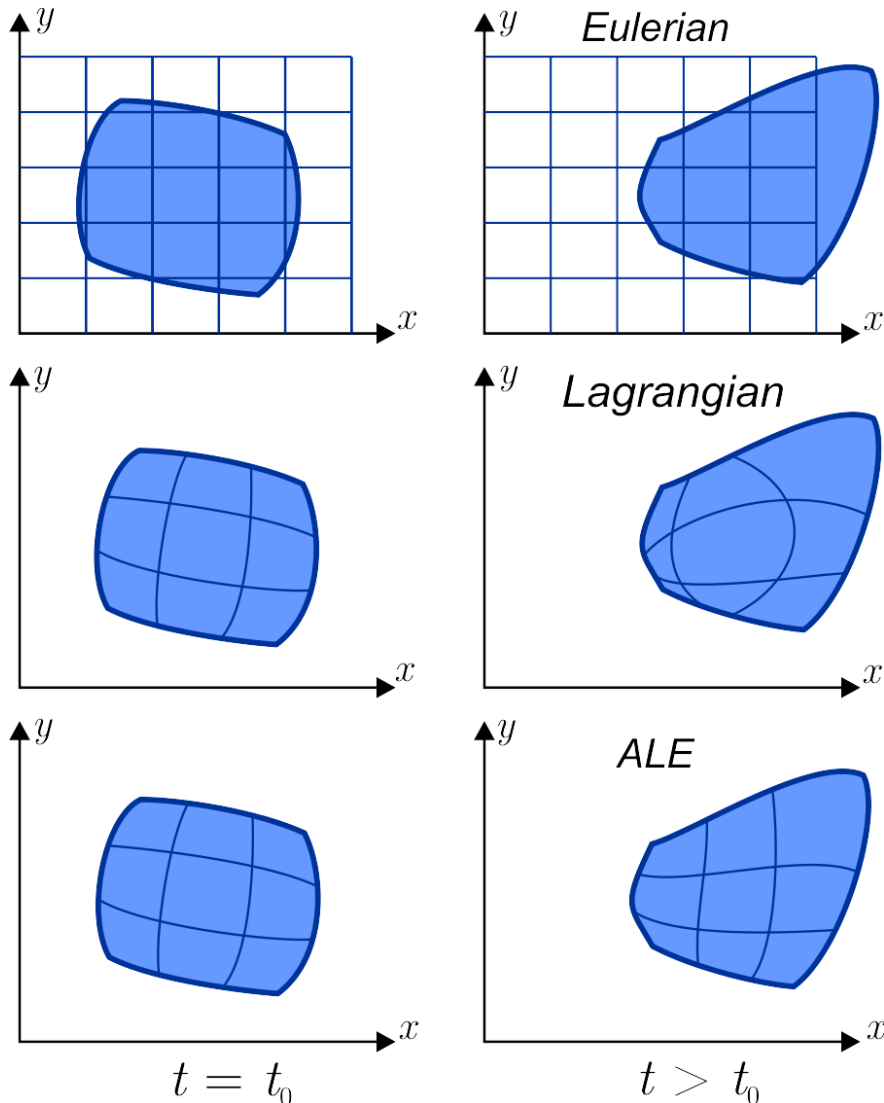
- ✓ Undistorted mesh
- ✗ Free boundaries are difficult to track
- ✗ History-dependent materials are difficult to handle

Lagrangian formalism

- ✗ The mesh can be rapidly distorted
- ✓ Free boundaries are automatically computed
- ✓ History-dependent materials are easier to handle

Introduction to ALE formalism

Kinematic description of the motion



Arbitrary Lagrangian Eulerian (ALE) formalism

- Extension of both previous formalisms
- The mesh motion is uncoupled from material motion
- ALE can be crudely seen as a continuous remeshing procedure
- Mesh topology does not change
- Remapping of variables is faster than classical remeshing

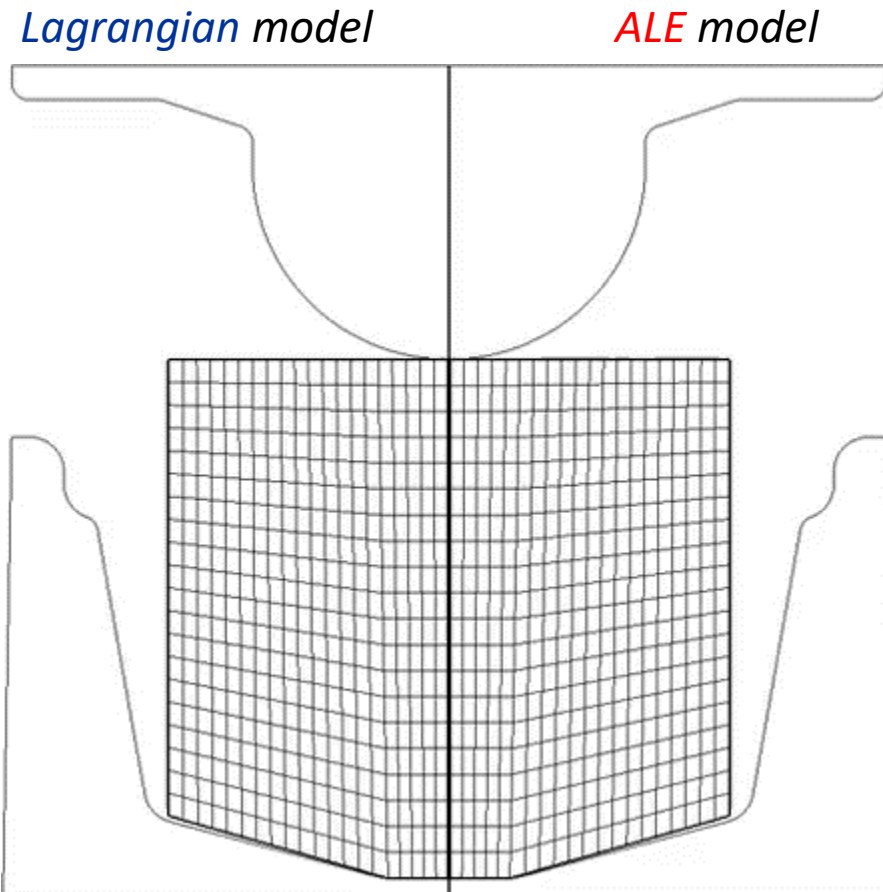
2 Families of ALE Problems

1. Problems involving excessive mesh distortion

Benefits of ALE vs. Lagrangian models

- Helps us keep well-shaped elements despite large deformations
- Most often remeshing is completely avoided

Example: axisymmetric forging

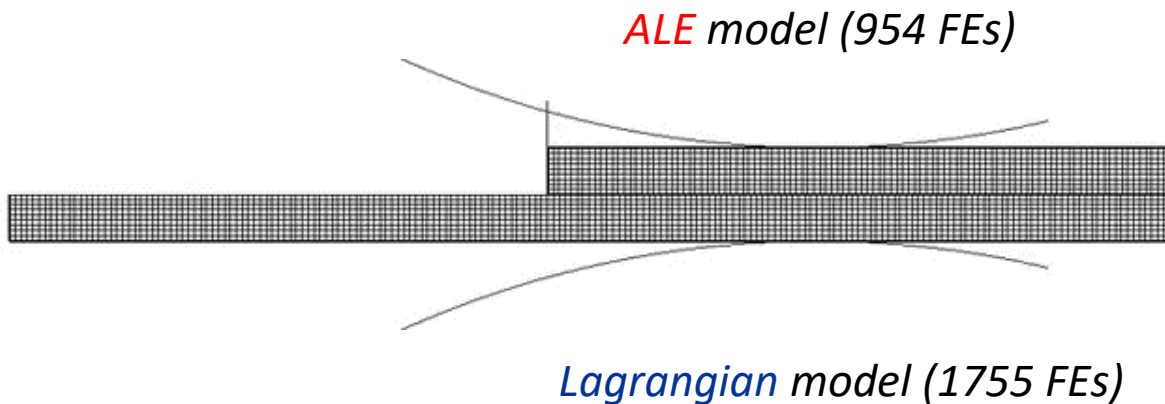


2 Families of ALE Problems

2. “Quasi-Eulerian” models

Example: cold rolling

↑
free
└ *fixed* →



Benefits of ALE vs. Lagrangian models

- The size of the model is decreased in the flow direction.
- Loading is easier.
- The element size may be optimised in the flow direction.
- The contact regions do not change.

Scope

1. Practical Management of Simulation Codes

- Metafor... from 1992 to 2018

2. Numerical Applications

- Introduction to ALE formalism
- **Thixoforming**
- Continuous Roll forming
- Friction Stir Welding
- Additive manufacturing

3. Conclusions

Thixoforming

What is thixotropy?

- A thixotropic material behaves...
 - ... like a solid, at rest (a billet can sustain its own weight).
 - ... like a liquid, during shearing (can be cut easily).
- Common examples: clays, muds, paints, tomato ketchup.
- Some alloys (Magnesium, Aluminium, Steels) exhibit a semi-solid behaviour in a narrow range of T.



What is thixoforming?

- Taking advantage of this semi-solid state.
- The process combines advantages of casting and forging.

Thixofforming

Constitutive behaviour

“Extended yield criterion” for thixotropy

$$\bar{f} = \bar{\sigma}^{VM} - \sigma_y - \eta \dot{\varepsilon}^{vp}$$

modified yield stress

$$\sigma_y = \sigma_y(\dot{\varepsilon}^{vp}, \bar{\varepsilon}^{vp}, \lambda, f_l)$$

apparent viscosity

$$\eta = \eta(\dot{\varepsilon}^{vp}, \bar{\varepsilon}^{vp}, \lambda, f_l)$$



R. Koeune

2 NEW INTERNAL
VARIABLES

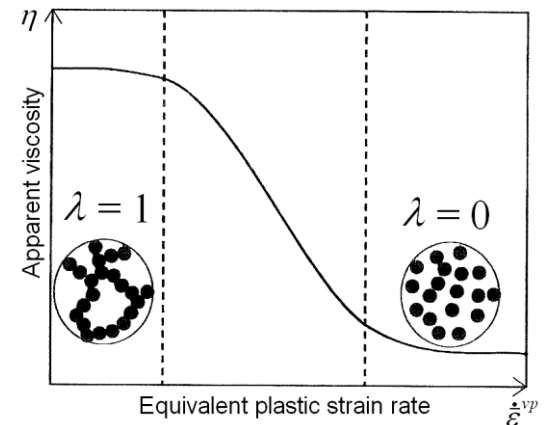
liquid fraction
(phase change)

$$f_l = f_l(T)$$

cohesion degree
(microstructural state)

$$\lambda = \lambda(\dot{\varepsilon}^{vp}, f_l)$$

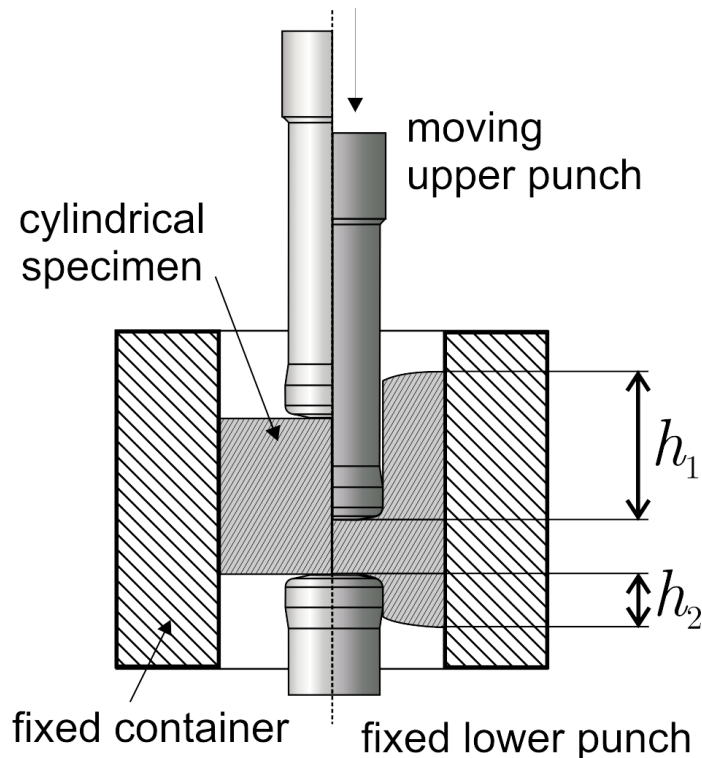
Both liquid and solid formalisms are contained in the proposed models!



Thixoforming

Double cup extrusion – process description

- Production of an axisymmetric H-shape part, starting from a cylinder.
- Temperature controlled by induction heating.
- Experiments made in PIMW, Liège in collaboration with ENSAM, Metz.
- Material: 100Cr6 steel alloy.



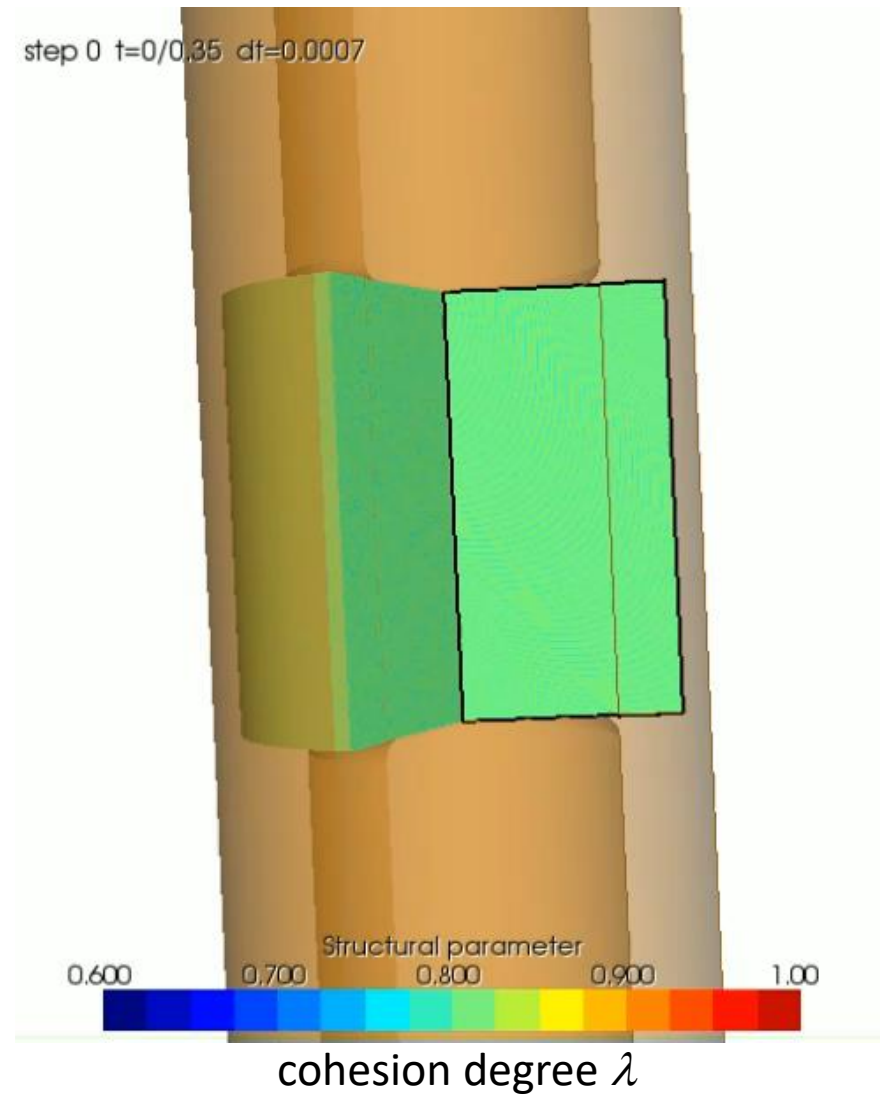
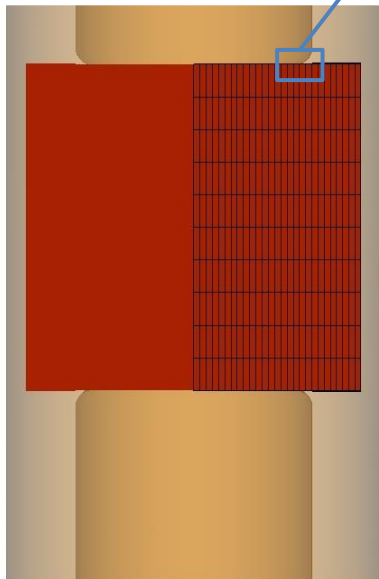
Example of (half) part

Thixofforming

Double cup extrusion – mesh management

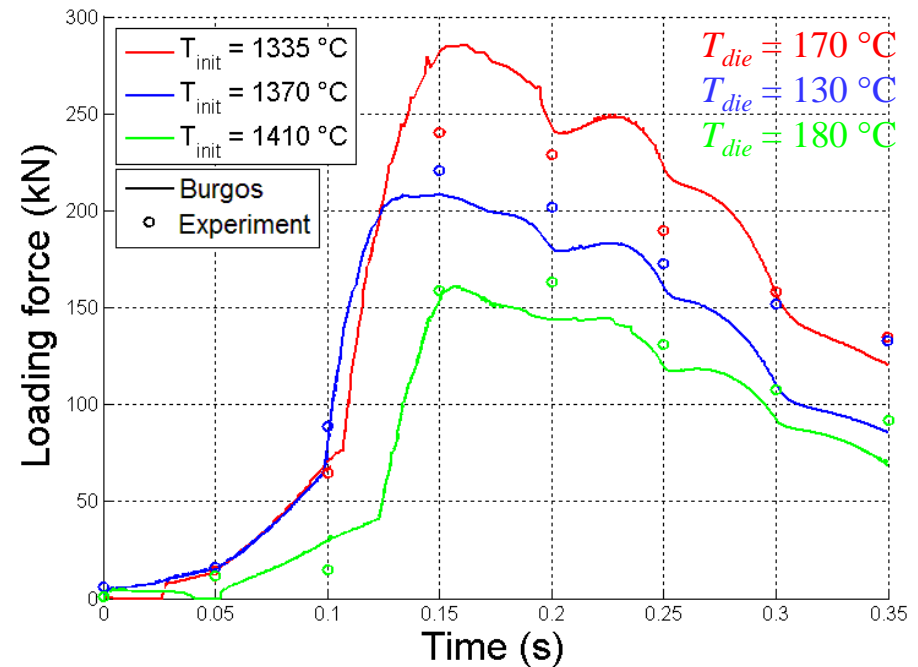
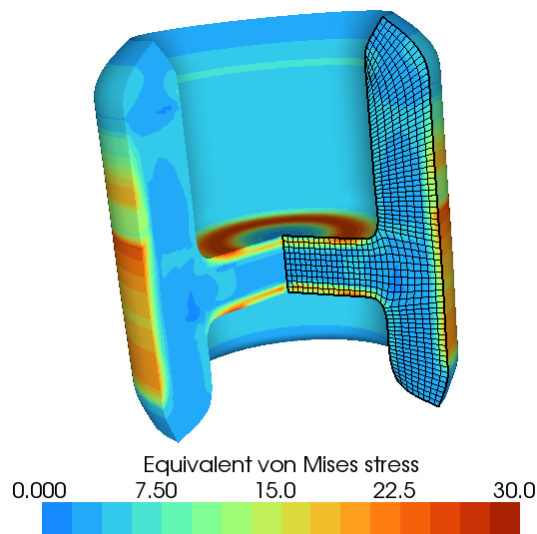
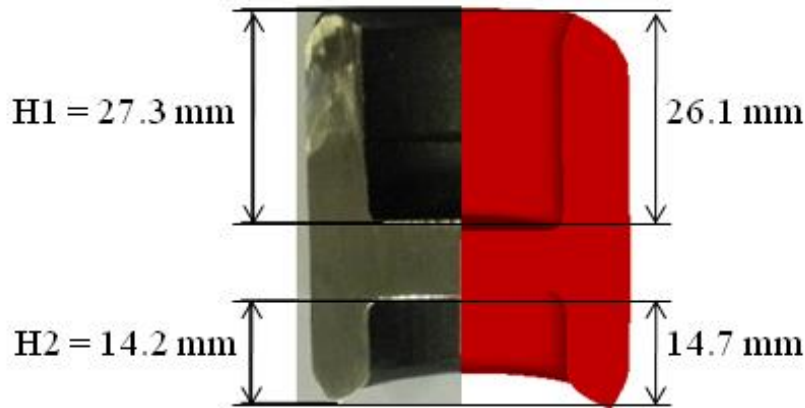
ALE mesh management
(numerical trick):

→ An initial thin squeezed mesh is added to the top of the billet



Thixoforming

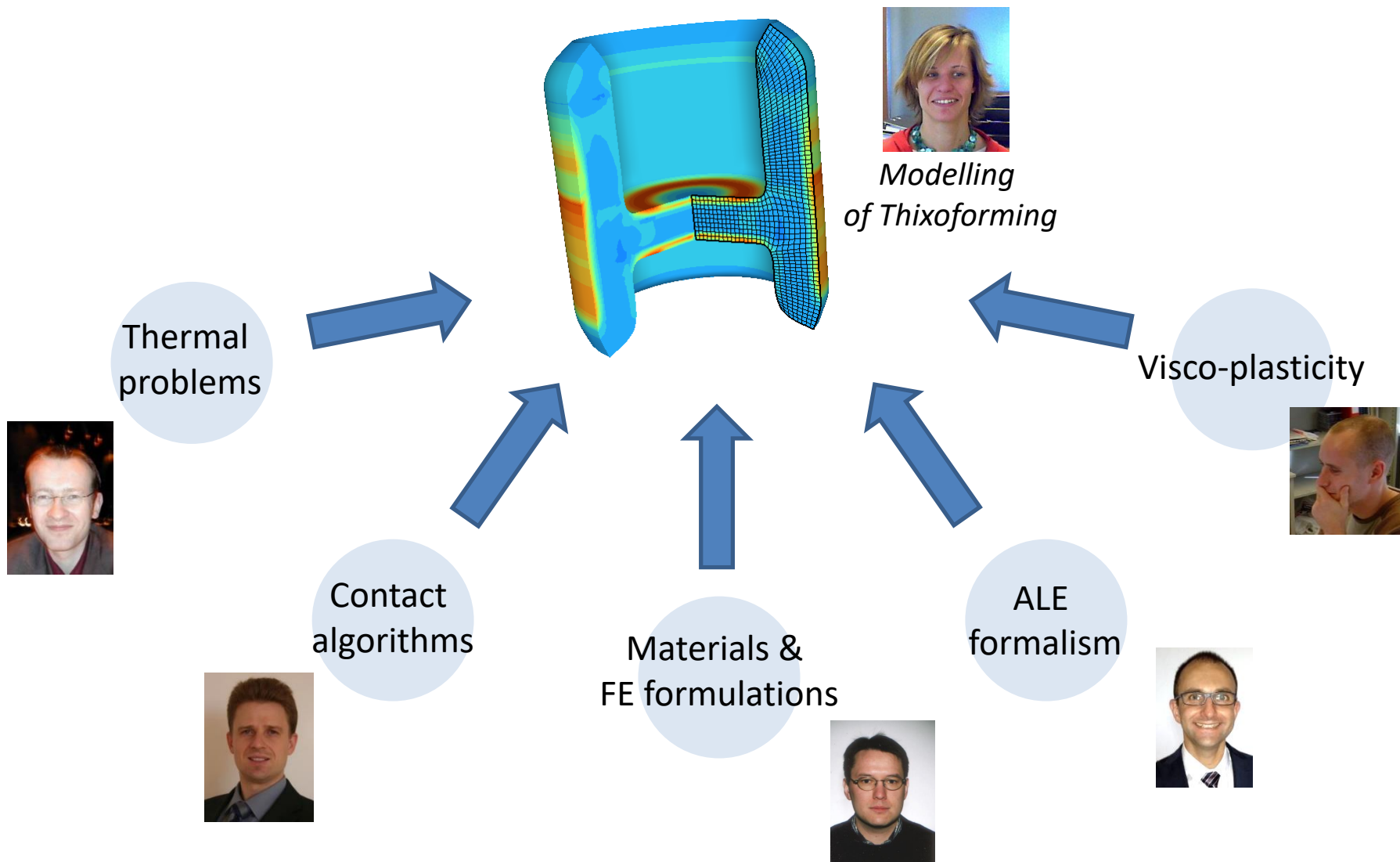
Double cup extrusion - results



- Good agreement between experimental and numerical results.
- The model is able to predict the **residual stresses** after unloading and cooling to room temperature.

Thixoforming

A PhD project built from the results of other projects



Scope

1. Practical Management of Simulation Codes

- Metafor... from 1992 to 2018

2. Numerical Applications

- Introduction to ALE formalism
- Thixoforming
- **Continuous Roll forming**
- Friction Stir Welding
- Additive manufacturing

3. Conclusions

Continuous Roll forming

Process description

A metal strip is incrementally bent by sets of rolls (called “forming stands”) until the desired cross section is obtained

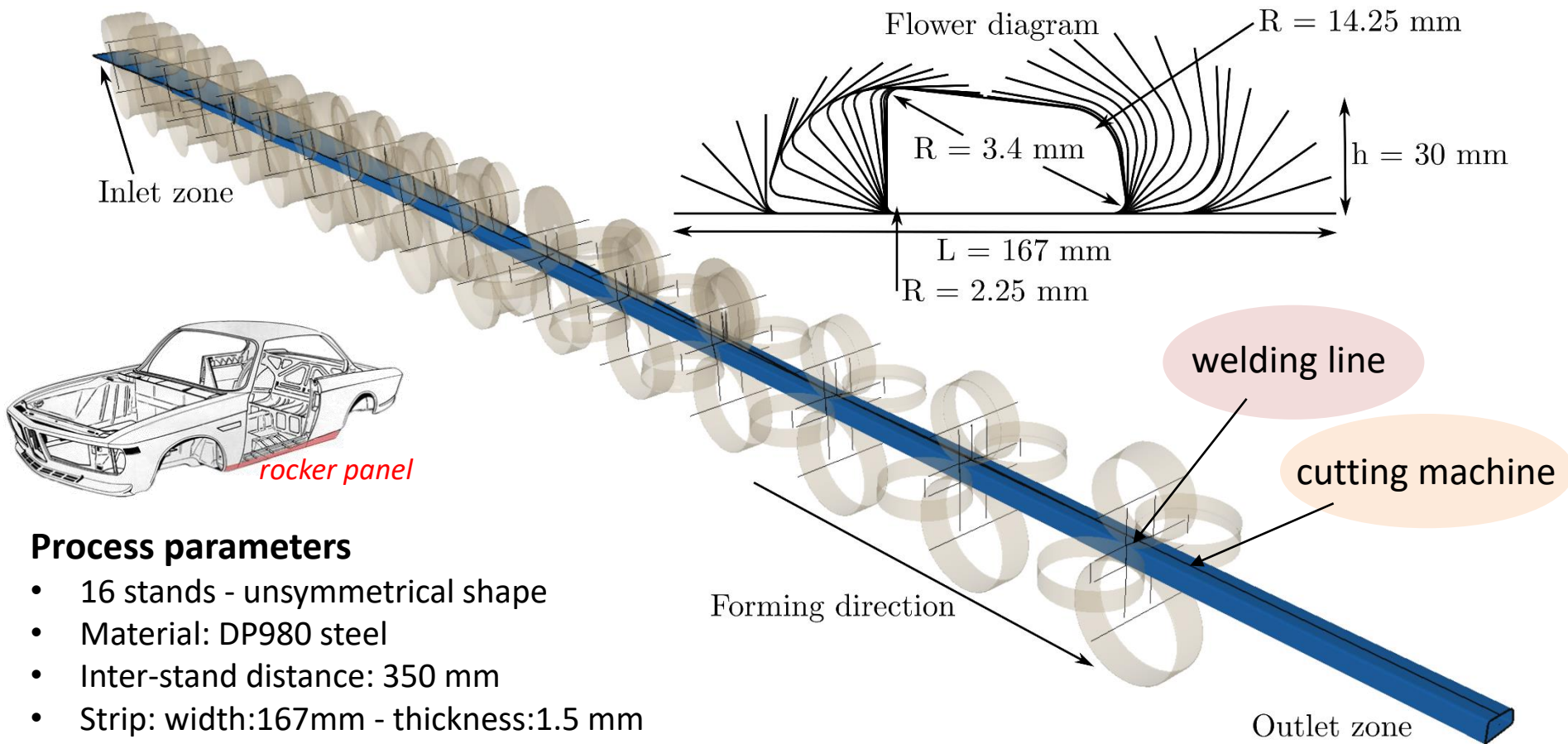
➔ Key advantage: OK for Advanced High Strength Steels



FE Modelling of roll forming is essential
(roll design, prediction of defects, residual stresses, springback, etc.)
... but very time-consuming!

Continuous Roll forming

Forming of a rocker panel (closed section)



Process parameters

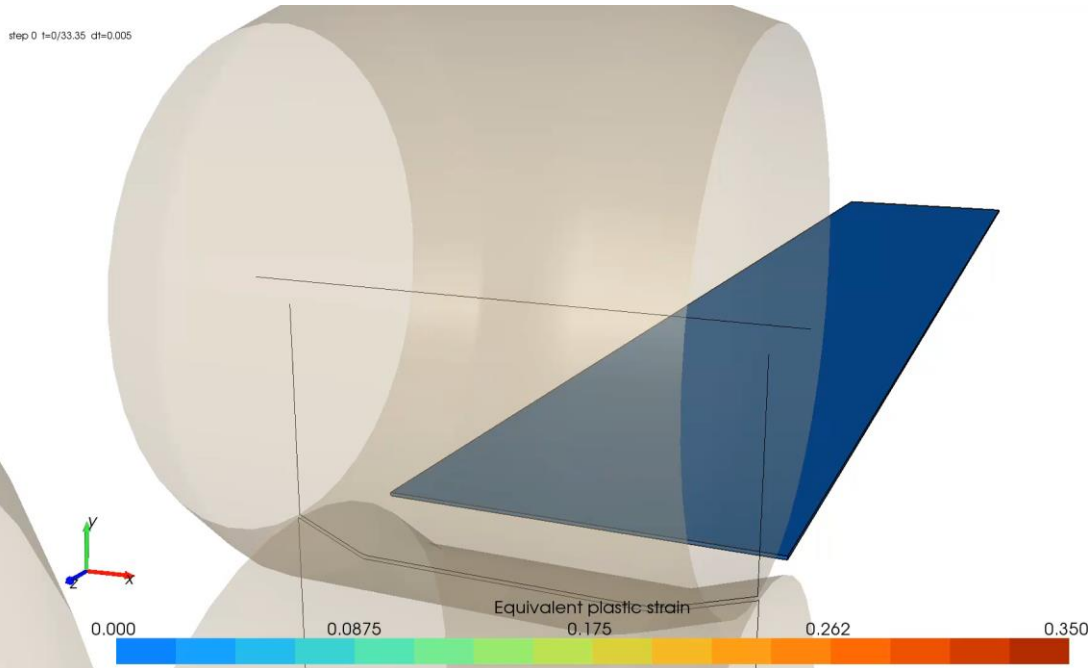
- 16 stands - unsymmetrical shape
- Material: DP980 steel
- Inter-stand distance: 350 mm
- Strip: width:167mm - thickness:1.5 mm
- Length of the model: 6.26m
- Sheet velocity: $v = 200 \text{ mm/s}$
- Coulomb friction $\mu = 0.2$

Continuous Roll forming

Drawbacks of Lagrangian modelling



ArcelorMittal



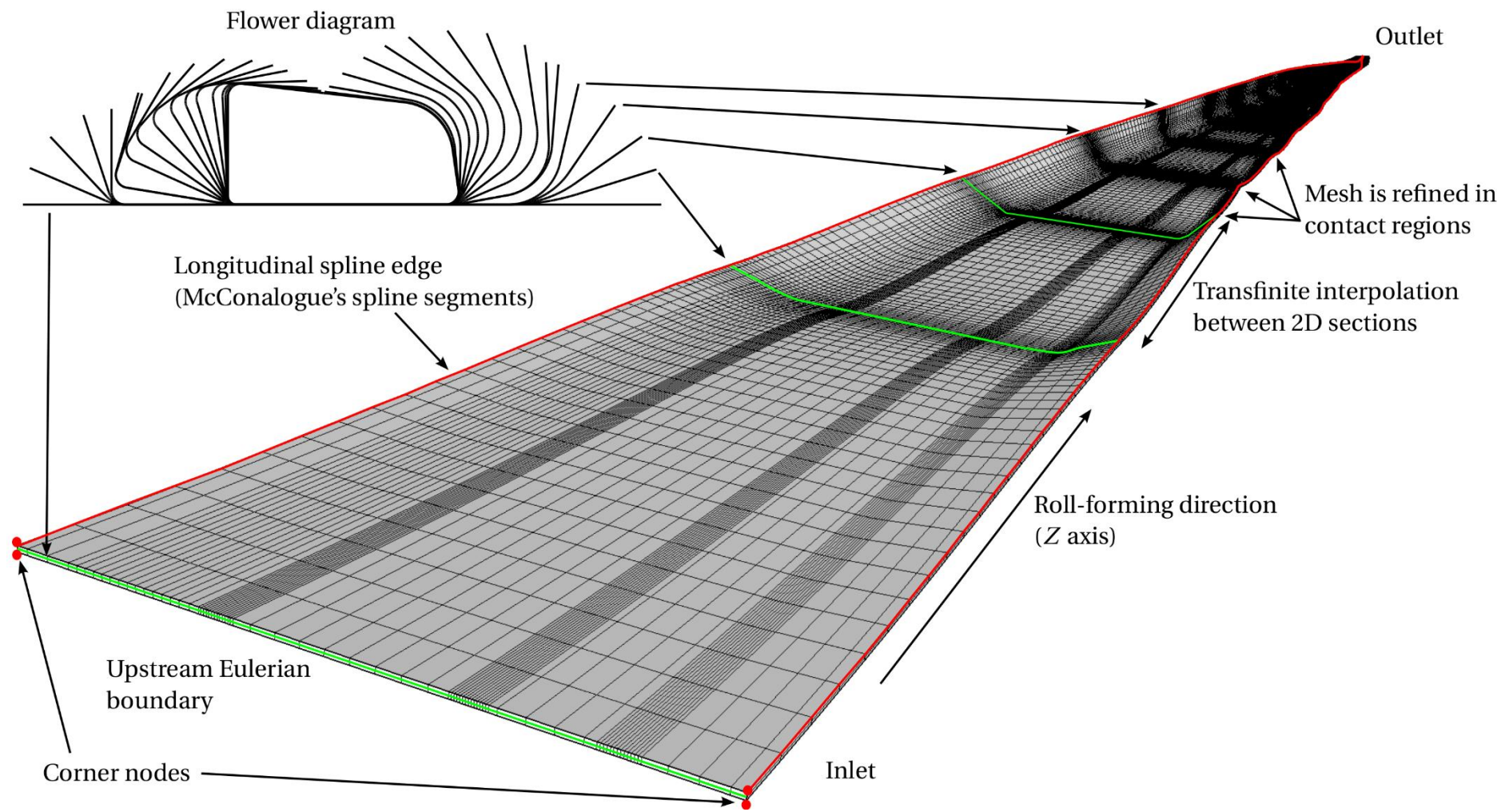
In a Lagrangian model, the continuous strip is modelled by a sheet of limited length.

- Non-realistic boundary conditions or friction coefficients to make the sheet move.
- **The sheet hits the stands and gets sometimes blocked.**
 - ➔ Incremental modifications of the geometry of the tools by trial and error.
- How to model the welding and post-cut operation?

The setup of the Lagrangian model is **very tricky** and time-consuming!

Continuous Roll forming

Initial ALE mesh: interpolation of the flower diagram



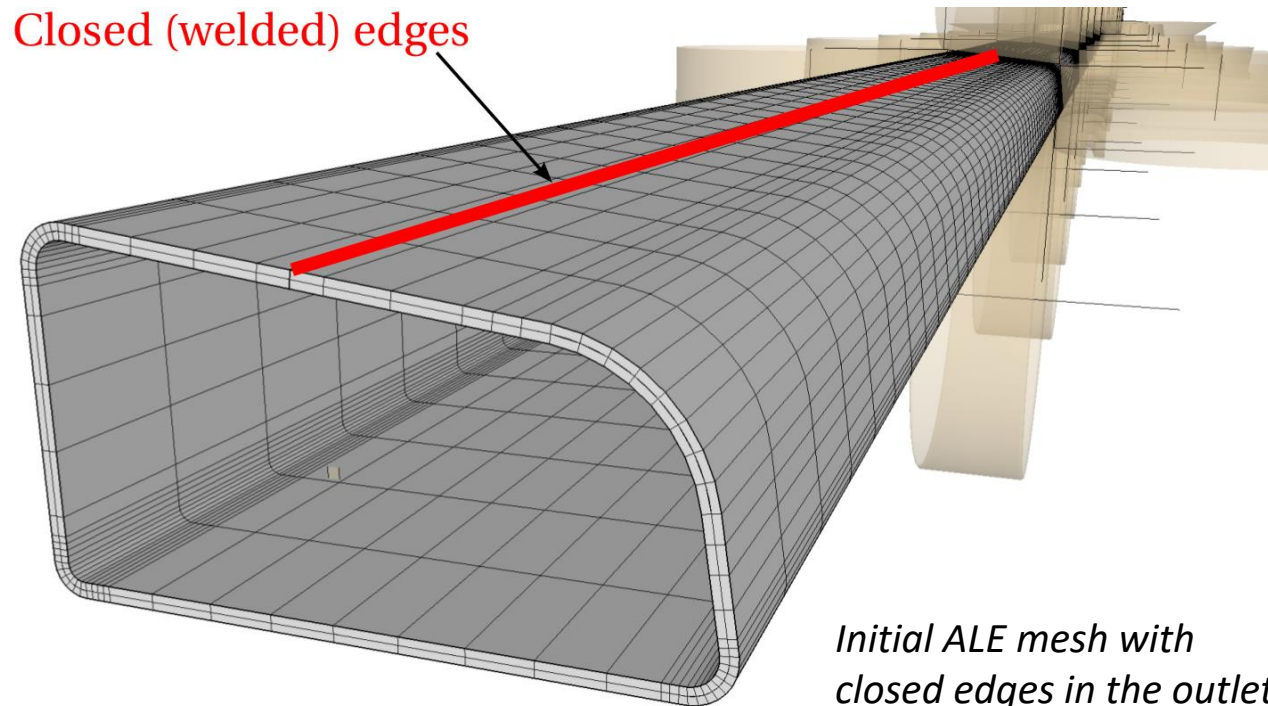
Continuous Roll forming

Modelling of inline welding in ALE

- Actual physics in the welding process is **heavily simplified**.
- The initial mesh is built as a closed mesh.



Y. Crutzen



Initial ALE mesh with closed edges in the outlet.

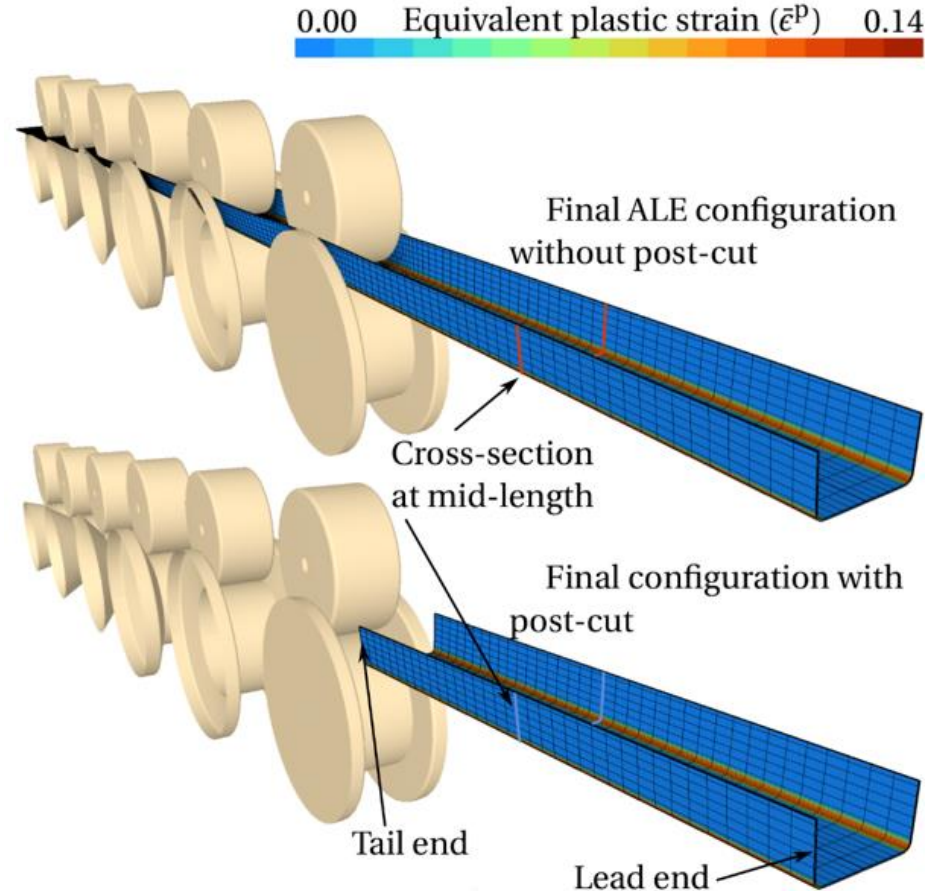
Continuous Roll forming

Modelling of the post-cut operation in ALE



Y. Crutzen

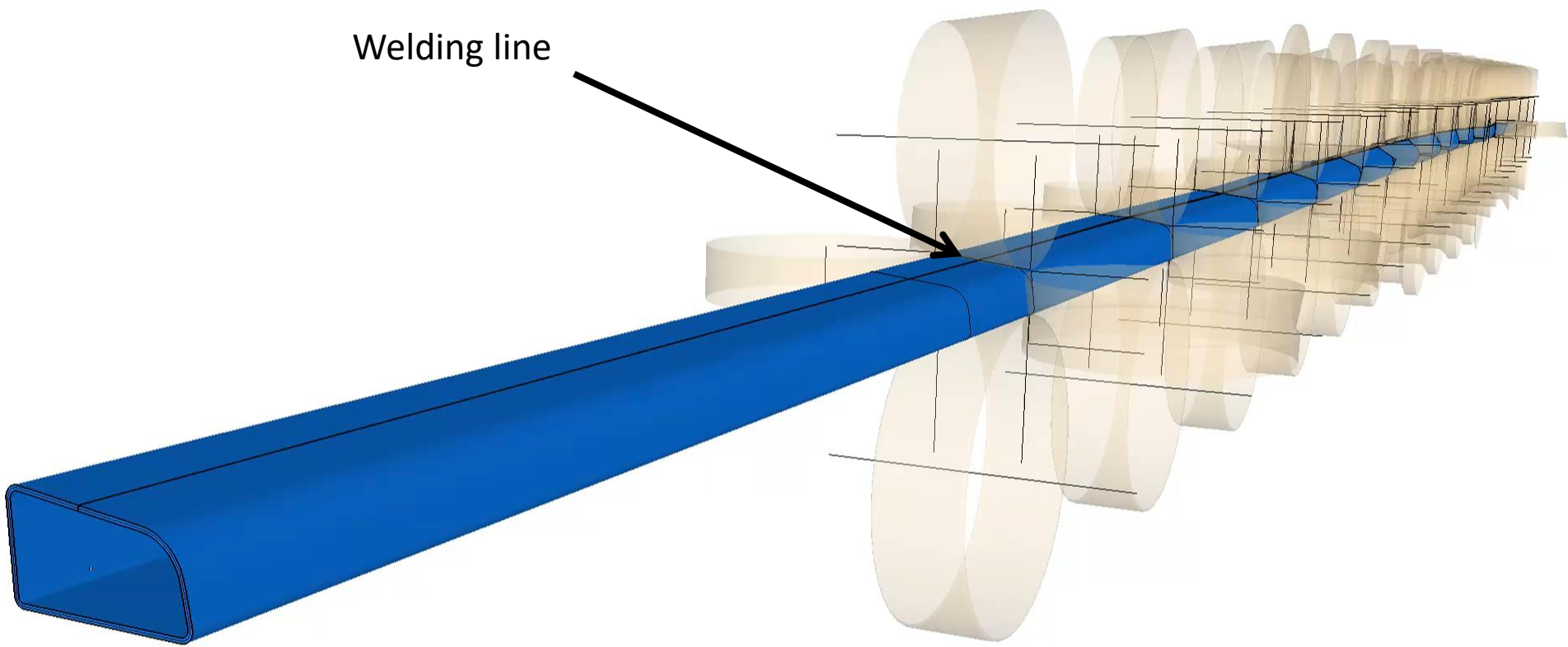
- Aims:
 - release the final product from the forming rolls
 - Accurate computation of the springback
- **Perfect cut-off is assumed:** deactivate the finite elements located inside the roll-forming machine.
- Switch from ALE to **purely Lagrangian** formalism.
- Implicit dynamic integration scheme with **viscous damping to kill out the oscillations.**



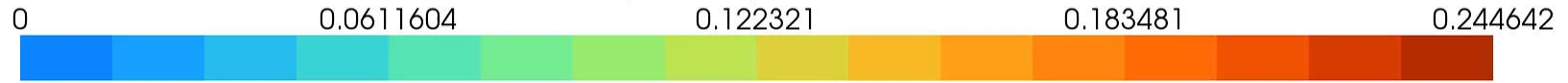
Continuous Roll forming

View of ALE results from a downstream point of view

Welding line



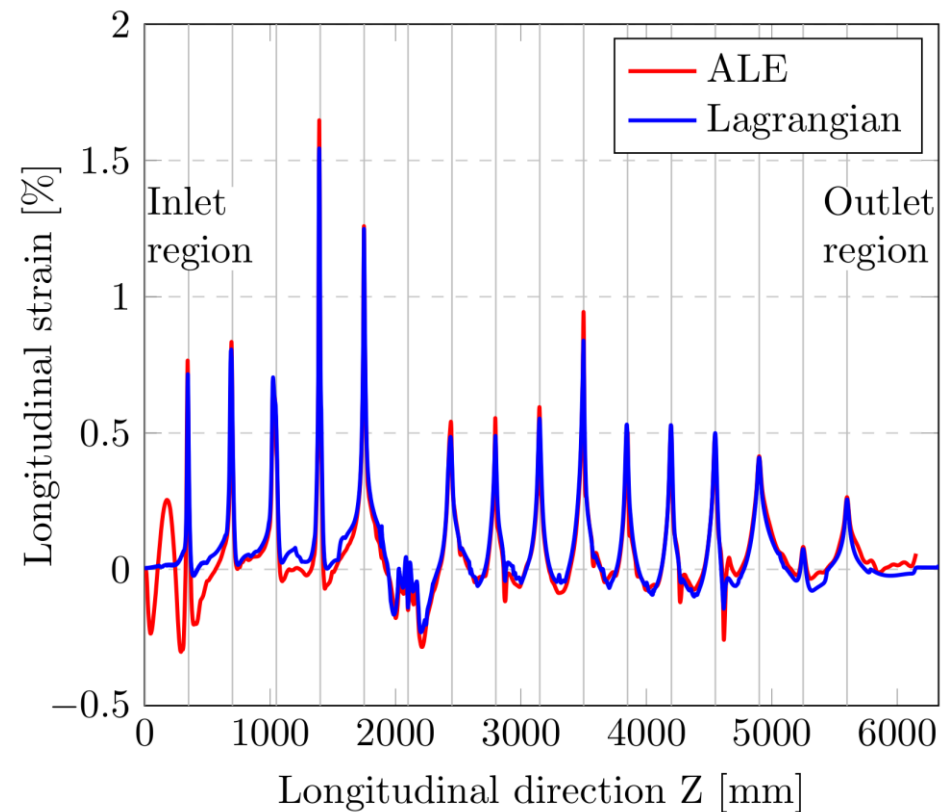
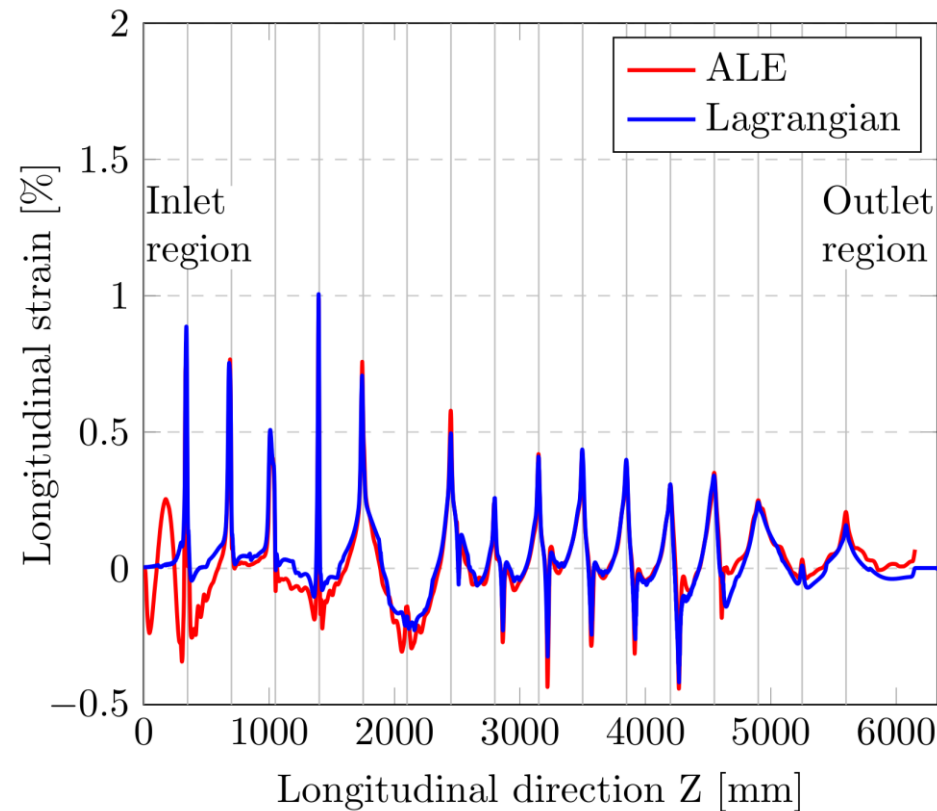
Equivalent plastic strain



Continuous Roll forming

Comparison to Lagrangian results

Longitudinal strain along the 2 edges

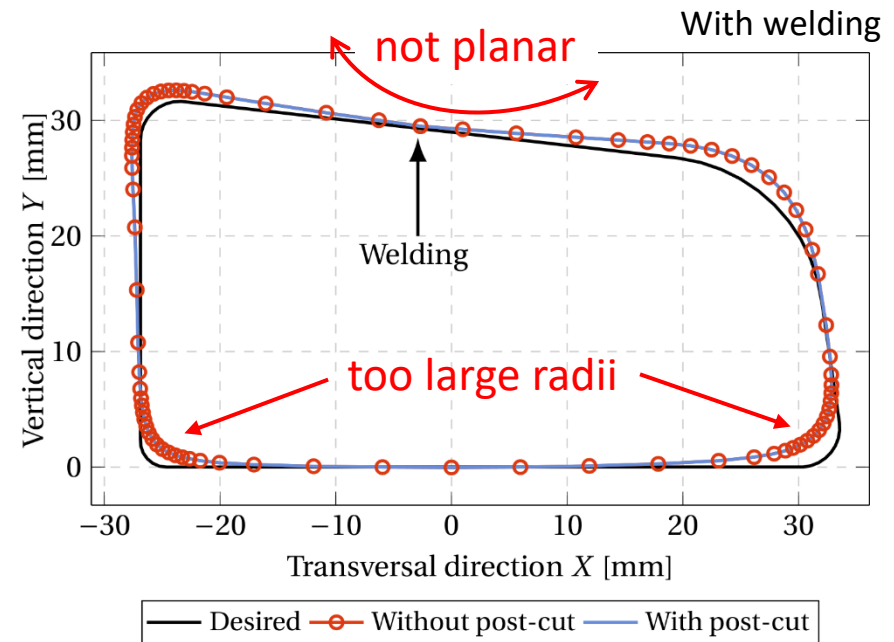
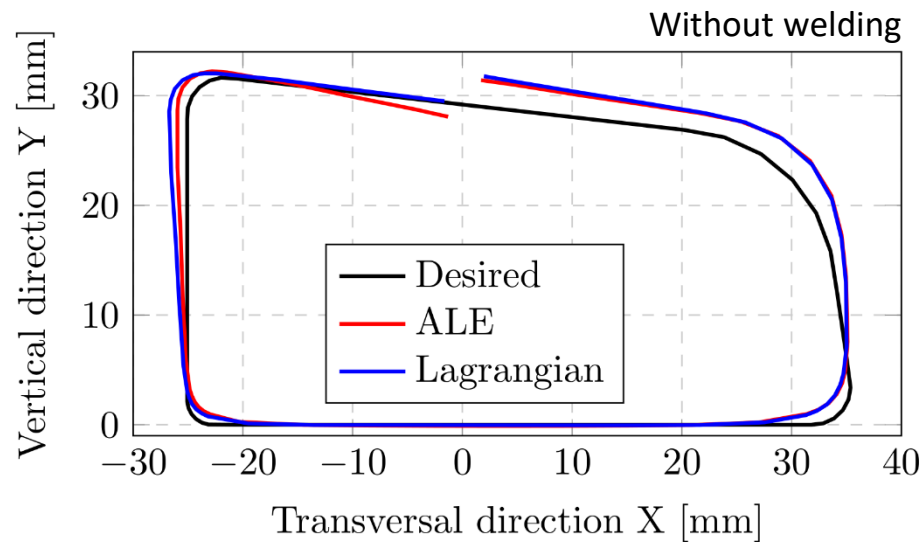


- ALE and Lagrangian curves are very **similar**, except in the inlet region.
- Maxima values are close to each other.

Continuous Roll forming

Influence of the welding operation and analysis of the final section

Final shape of the cross-section extracted at mid-length of the product

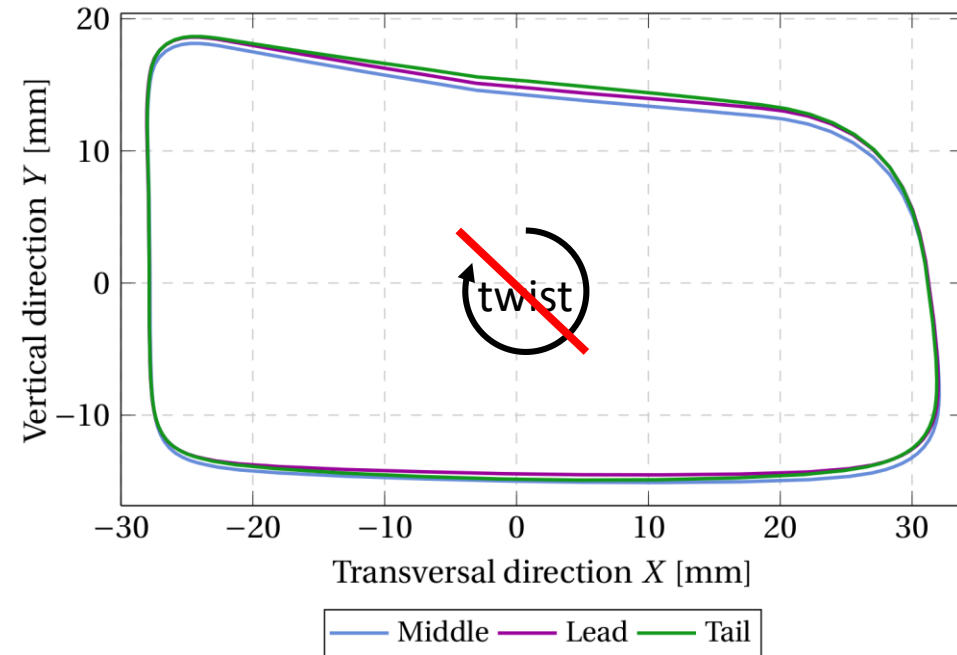
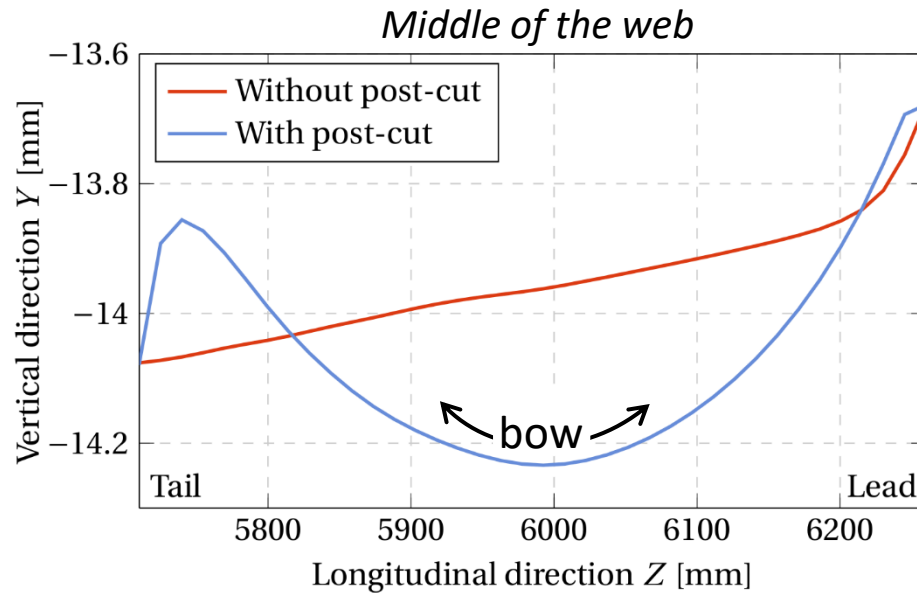


Shape defects predicted by the model:

- Bending radii in the lower part are much larger than the desired ones.
- Upper part is not perfectly planar.

Continuous Roll forming

Analysis of the final shape of the cut-to-length strip

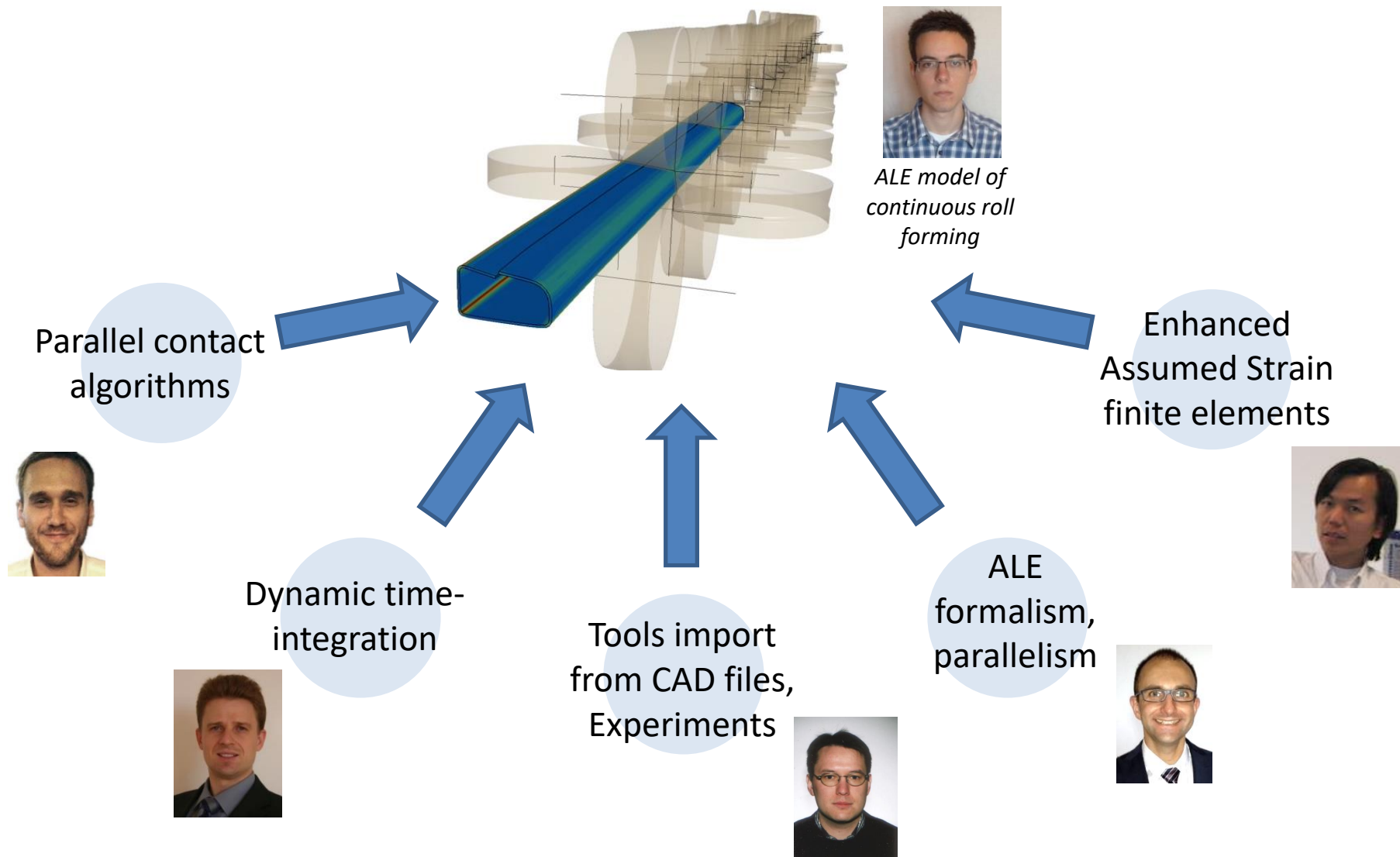


Shape defects predicted by the model:

- A slight longitudinal **bow defect** appears after post-cut.
- No significant **twist defect** (torsion)

Continuous Roll forming

A project built from the results of other projects



Scope

1. Practical Management of Simulation Codes

- Metafor... from 1992 to 2018

2. Numerical Applications

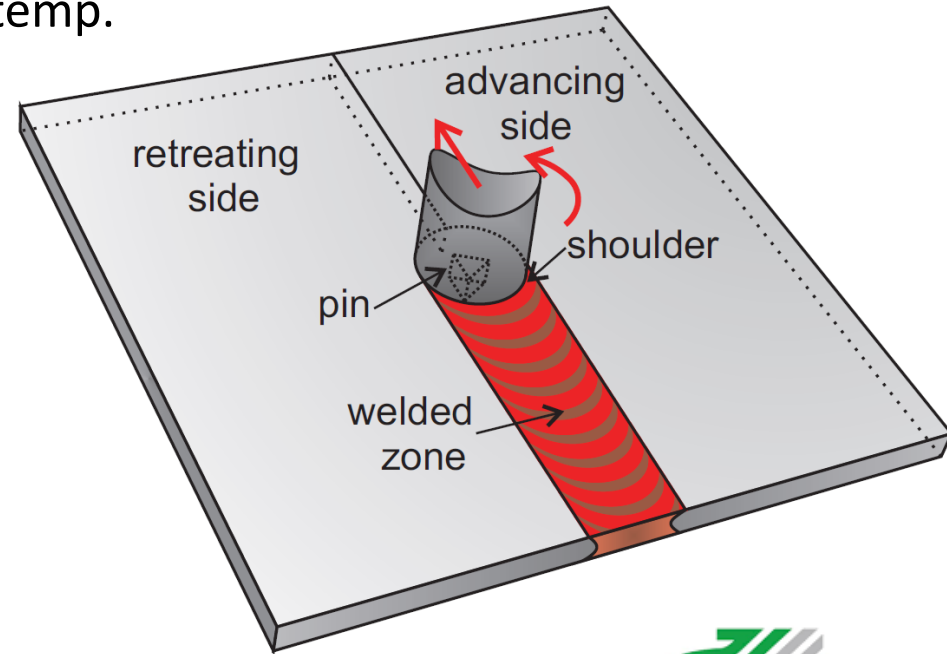
- Introduction to ALE formalism
- Thixoforming
- Continuous Roll forming
- **Friction Stir Welding**
- Additive manufacturing

3. Conclusions

Friction Stir Welding

Process description

- Solid-state joining process → better than classical welding.
- $T_{\text{process}} < T_{\text{melting}}$.
- Non-consumable tool.
- Mechanical intermixing + friction heating.
- Well suited for alloys with low fusion temp.



FSW of aluminium alloy 2024

- rotation: 500 RPM
- advance: 25 cm/min
- tool angle: 1.5°

Friction Stir Welding

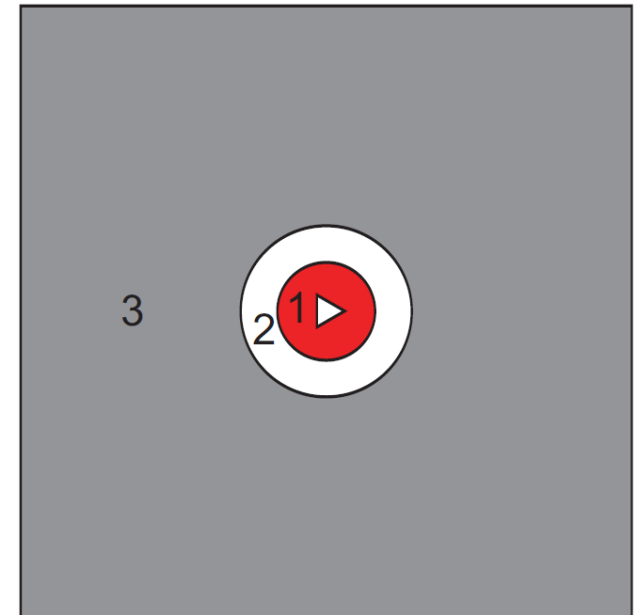
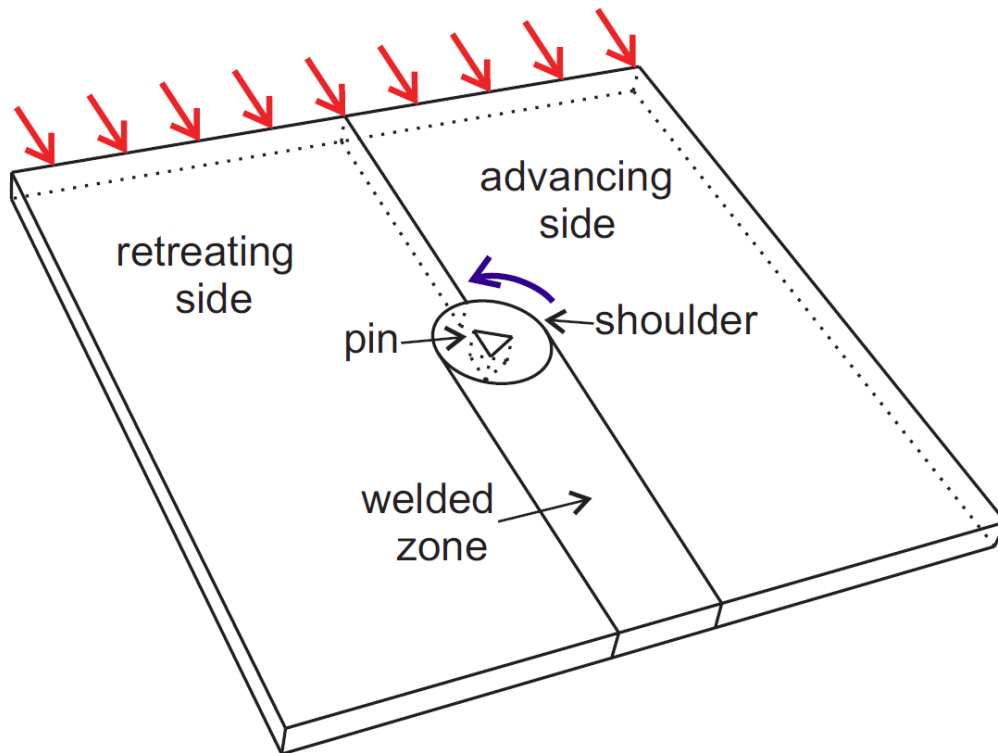
FSW modelling with the ALE formalism

Mesh management – 3 zones

1. The mesh moves with the tool – ALE formulation (red)
2. Remeshing (transition zone - white)
3. Fixed mesh – Eulerian formulation (grey)

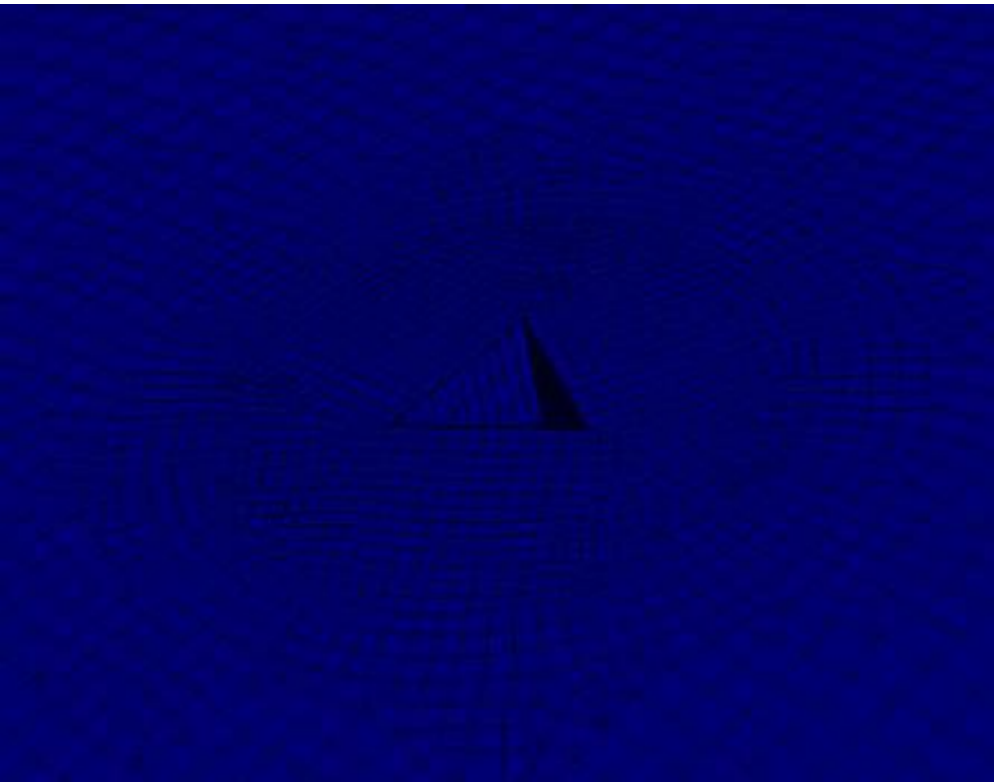


P Bussetta



Friction Stir Welding

FSW (ALE) – Example of mesh motion



Constitutive law:

Norton-Hoff
$$S = 2\mu D \left(\sqrt{3} \sqrt{\frac{2}{3} D : D} \right)^{m-1}$$



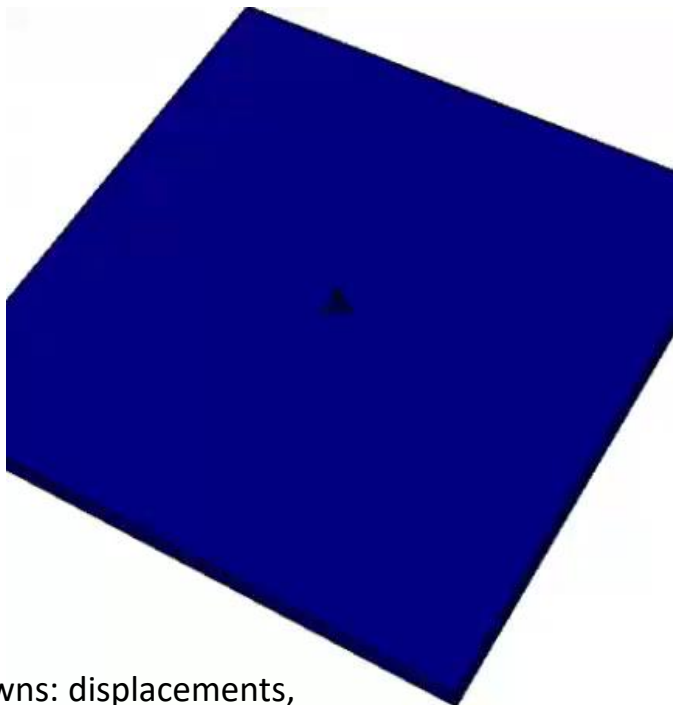
Friction Stir Welding

Comparison between a solid and a fluid approach



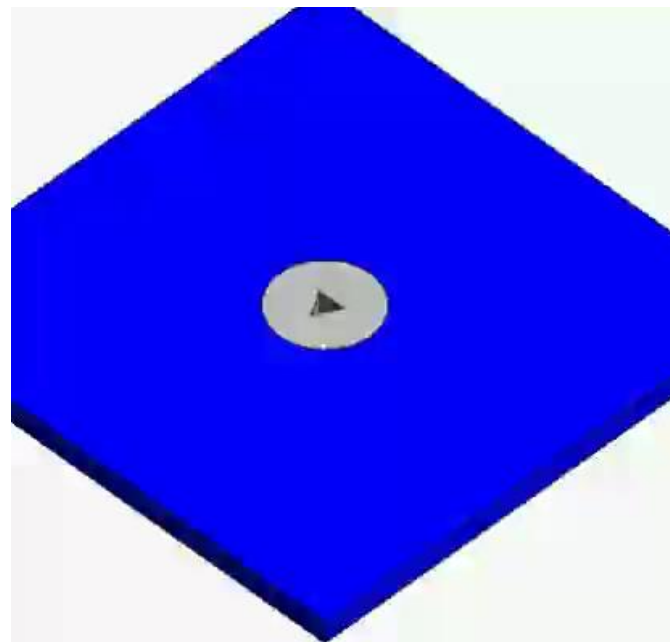
collaboration with UPC, Barcelona

Solid approach – Metafor (ULiège)

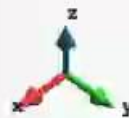


unknowns: displacements,
temperature

Fluid approach – COMET (UPC)

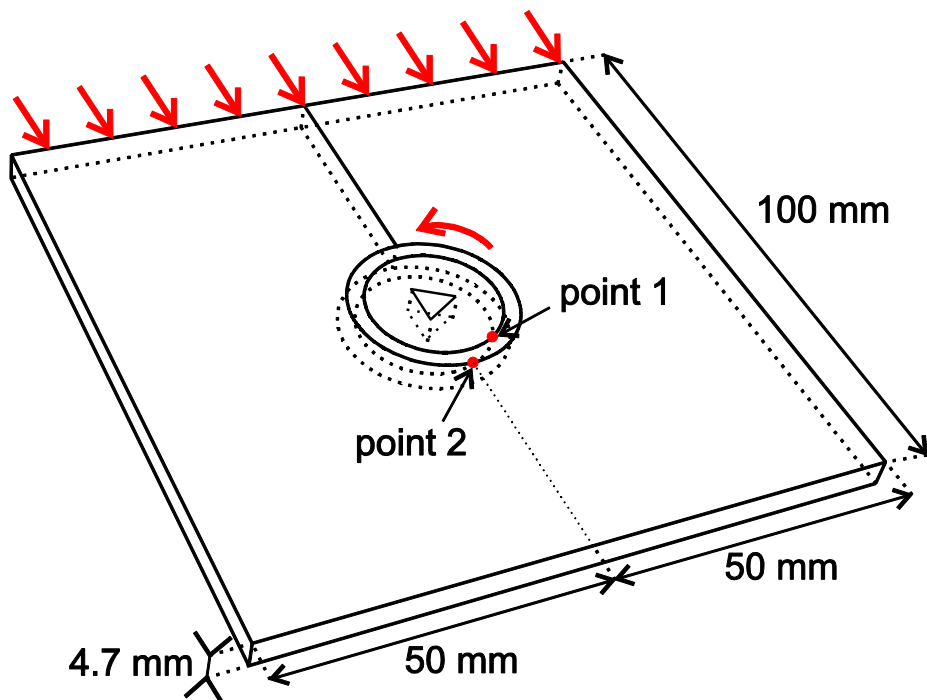


unknowns: pressure,
velocity,
temperature

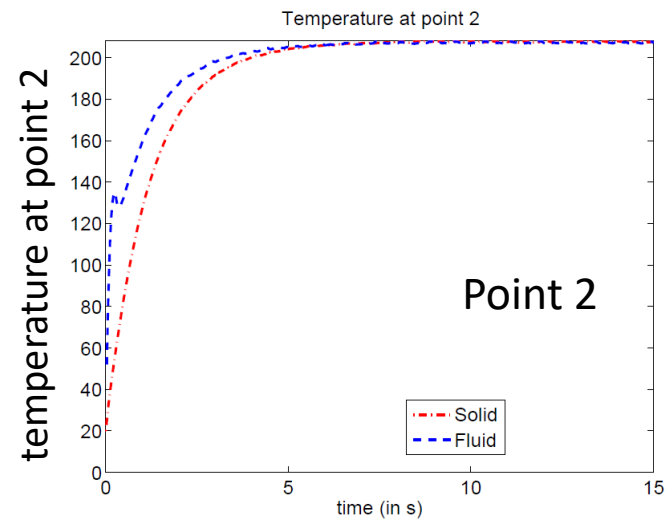
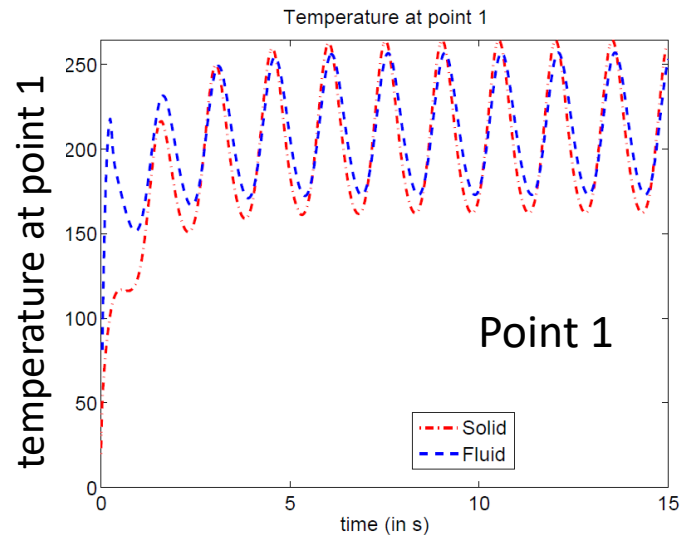


Friction Stir Welding

Comparison between a solid and a fluid approach



- Temperature evolution is similar with both approaches.



Friction Stir Welding

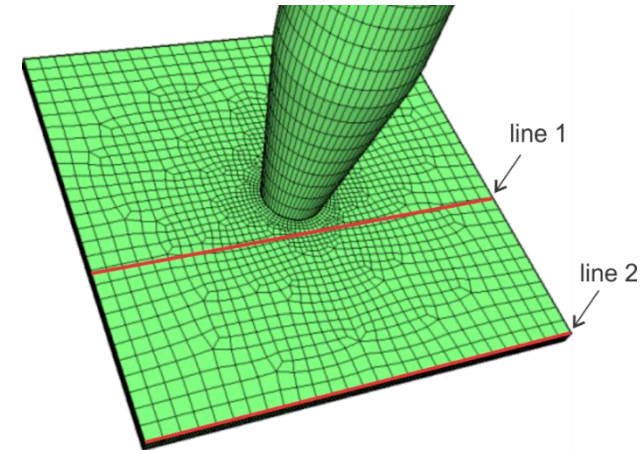
Residual stresses with the solid approach

Plates (AA2024): thermo-**elasto**-viscoplastic

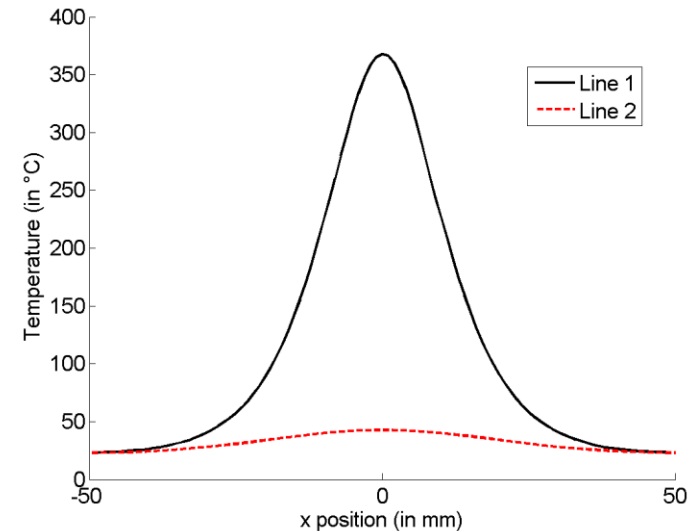
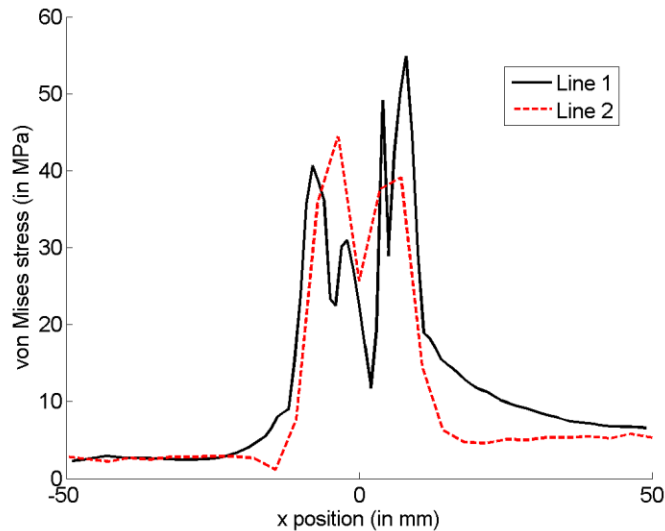
$$\sigma_y = \sigma_0(T) + A(T)(1 - e^{\xi \bar{\epsilon}}) + K e^{\frac{mQ}{RT}} \dot{\epsilon}^m$$

Sonne et al (2013) *JMPT* 213

- $\xi = 25$
- $K = 1.5 \text{ MPa s}^m$ (viscosity parameter)
- $m = 0,12$ (strain rate sensitivity parameter)
- $Q = 155 \text{ kJ/mol}$ (activation energy)
- Young modulus = $E(T)$
- Poisson ratio = 0.33



Stresses & temp. during the process



Friction Stir Welding

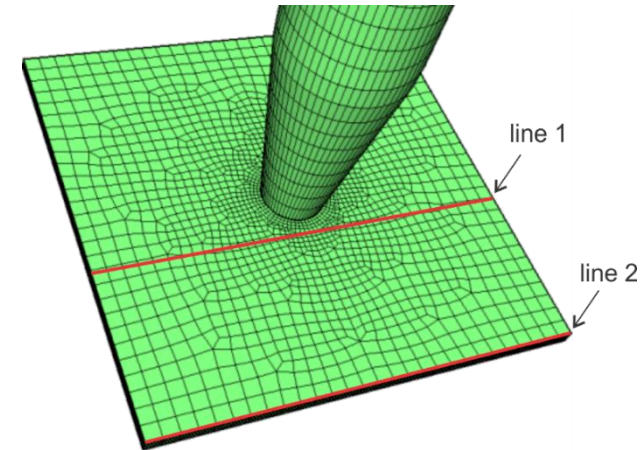
Residual stresses with the solid approach

Plates (AA2024): thermo-**elasto**-viscoplastic

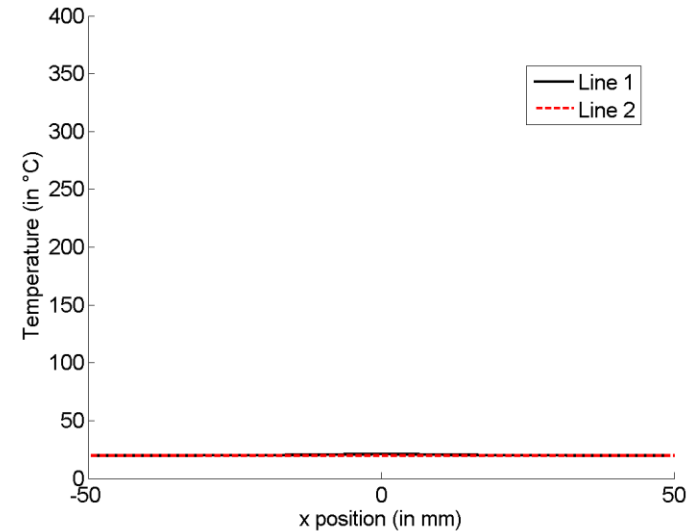
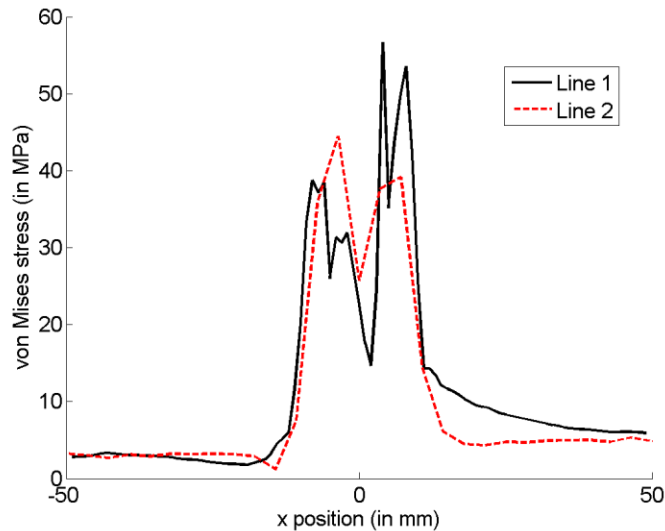
$$\sigma_y = \sigma_0(T) + A(T)(1 - e^{\xi \bar{\epsilon}}) + K e^{\frac{mQ}{RT}} \dot{\epsilon}^m$$

Sonne et al (2013) *JMPT* 213

- $\xi = 25$
- $K = 1.5 \text{ MPa s}^m$ (viscosity parameter)
- $m = 0,12$ (strain rate sensitivity parameter)
- $Q = 155 \text{ kJ/mol}$ (activation energy)
- Young modulus = $E(T)$
- Poisson ratio = 0.33

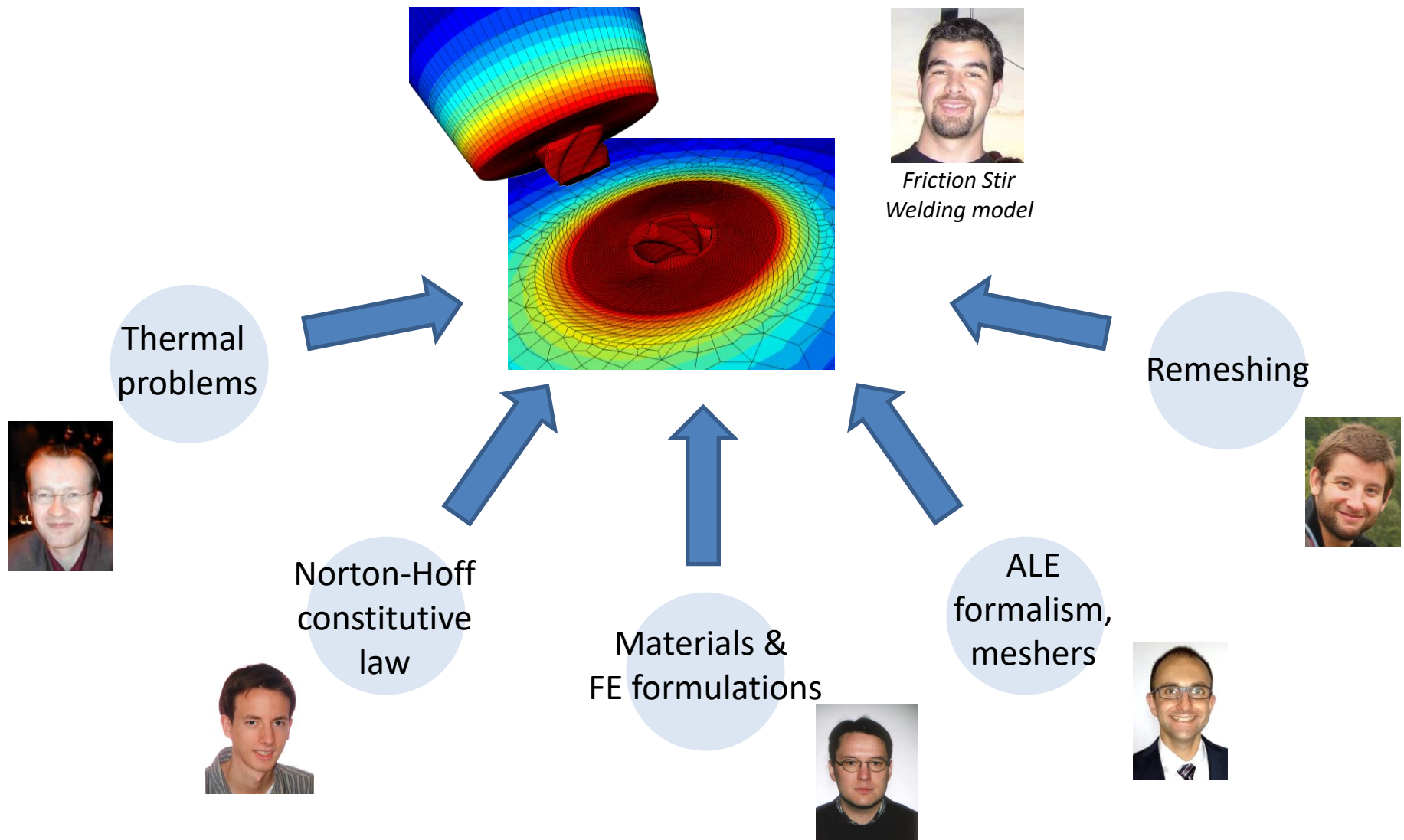


Stresses & temp. after the process and cooling



Friction Stir Welding

A project built from the results of other projects



Scope

1. Practical Management of Simulation Codes

- Metafor... from 1992 to 2018

2. Numerical Applications

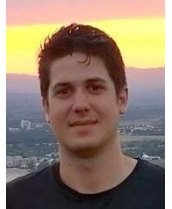
- Introduction to ALE formalism
- Thixoforming
- Continuous Roll forming
- Friction Stir Welding
- Additive manufacturing

3. Conclusions

Additive Manufacturing

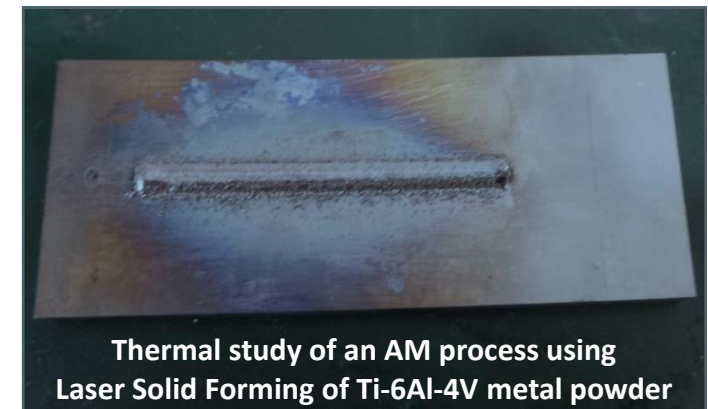
Context and challenges

- New PhD thesis started a few months ago: Very first model of an **laser solid forming** with Metafor.
- The thesis focuses on **mesh and geometry management**.
- Material law?
 - for now: thermal calculations only.
 - future: reuse of the constitutive law developed in the frame of thixoforming.
- Can we extend the “element deletion” algorithm (developed to compute cracks) to an “**element addition**” algorithm?



C. Laruelle

*Reference results from Chiumenti et al.
(UPC, Barcelona)*

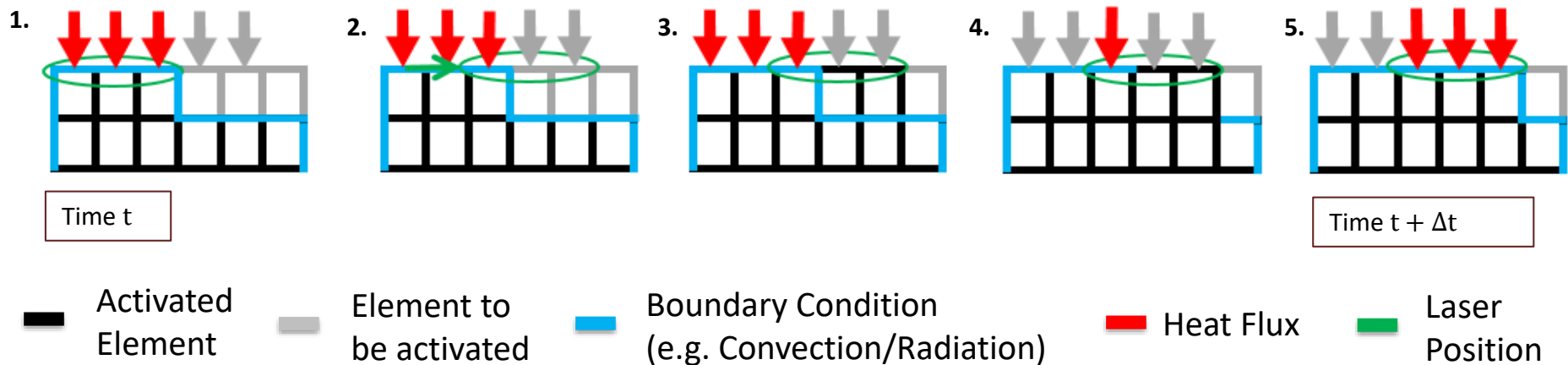


Thermal study of an AM process using
Laser Solid Forming of Ti-6Al-4V metal powder

Additive Manufacturing

Mesh management technique

Computation of new active mesh and boundary conditions

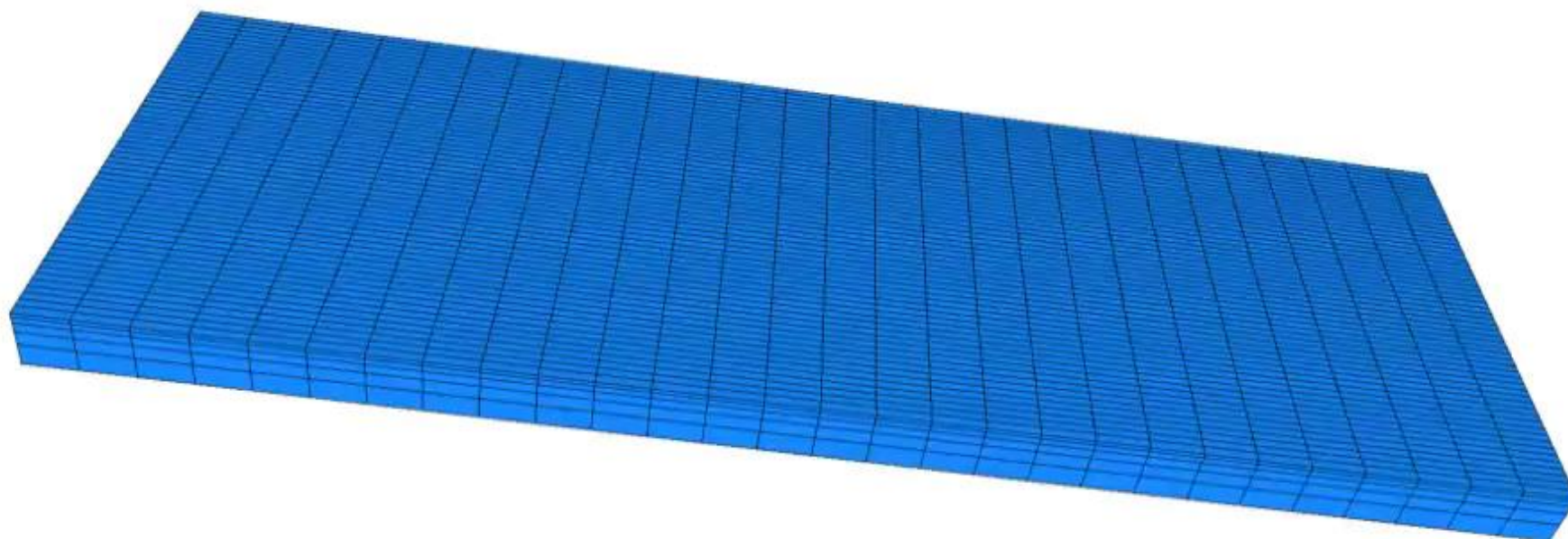


- Finite elements and boundary conditions are all created and deactivated at the start of the simulation.
- Activation/Deactivation of finite elements and boundary conditions based on the current laser position.

Additive Manufacturing

Numerical results – Time evolution of the temperature field

step 0 t=0/1586.58 dt=0



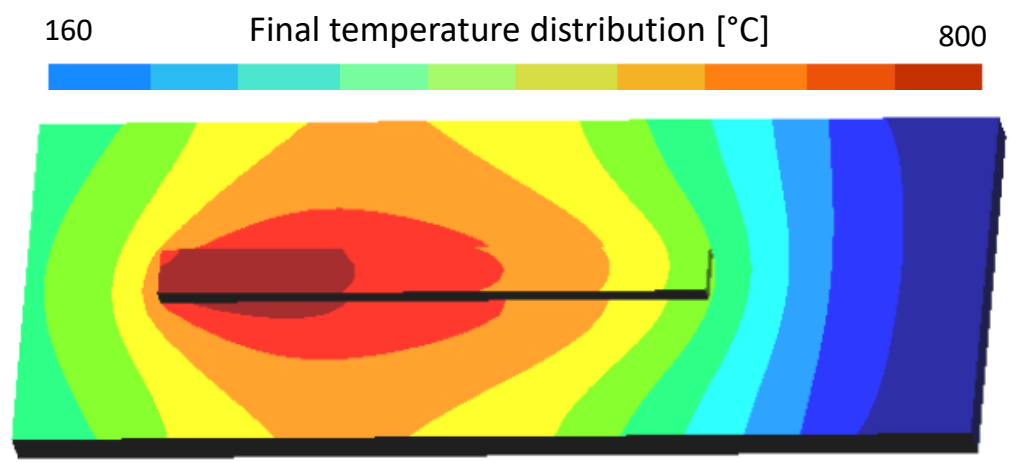
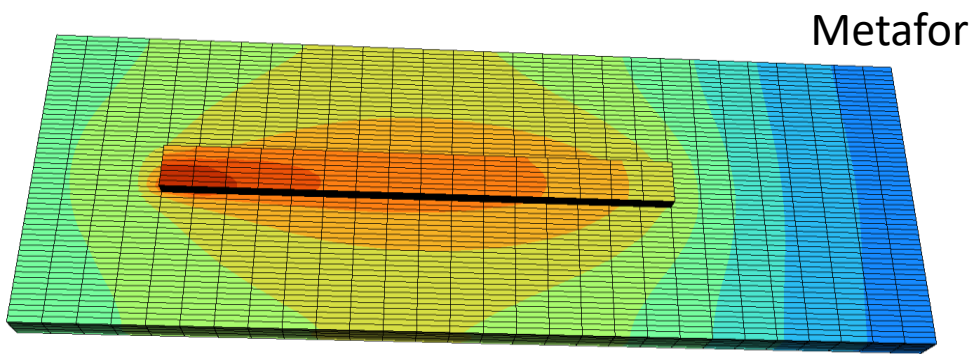
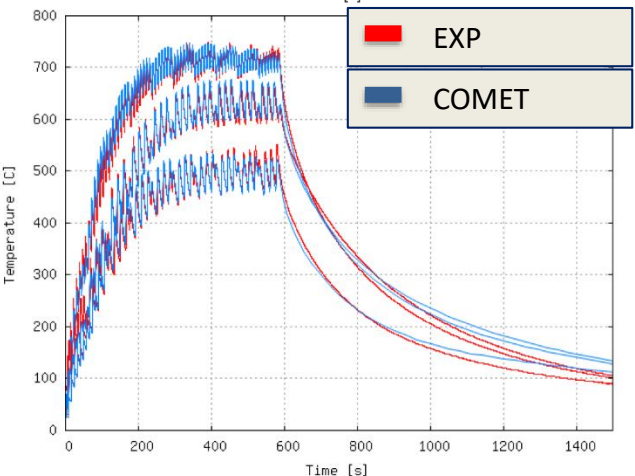
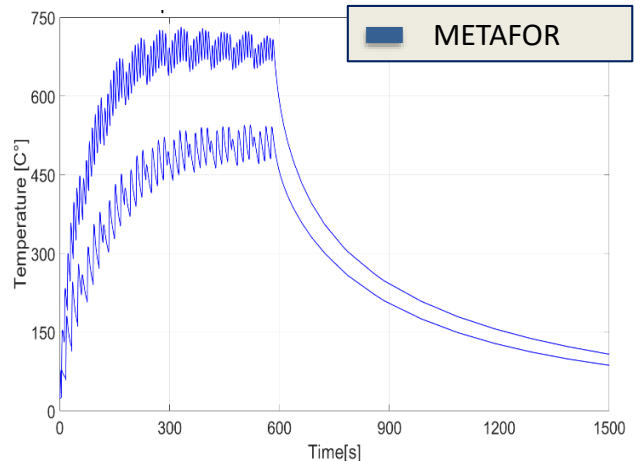
Additive Manufacturing

Results – comparison with literature



Good agreement of the temperature evolution between COMET, Metafor and experimental measurements.

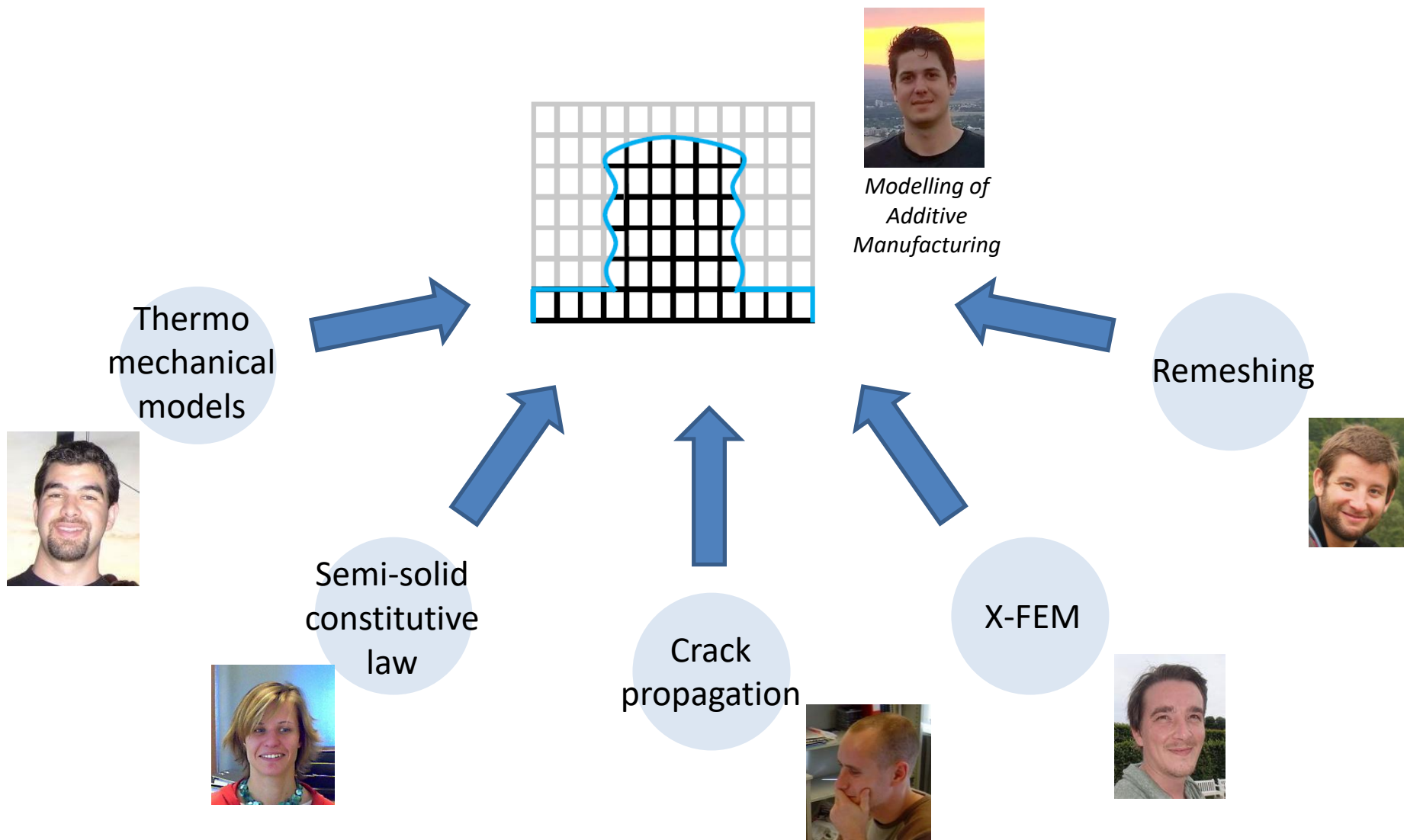
thermocouples on the lower surface



COMET

Additive Manufacturing

A project built from the results of other projects



Scope

1. Practical Management of Simulation Codes

- Metafor... from 1992 to 2018

2. Numerical Applications

- Introduction to ALE formalism
- Thixoforming
- Continuous Roll forming
- Friction Stir Welding
- Additive manufacturing

3. Conclusions

Conclusions

- The development of complex models/algorithms is always the **sum of the work of many researchers**.
- Keeping their source code clean, effective, reliable, robust and easily extensible is a difficult task which is **usually underestimated** in the academic world.
- A lot of simulation codes are **continuously lost** and projects based on previous work do not reach their goals.
- The presented tools and the resulting methodology have the advantage to be rather simple and allow the researchers to **spend more time on “science” and less time on “coding”**.