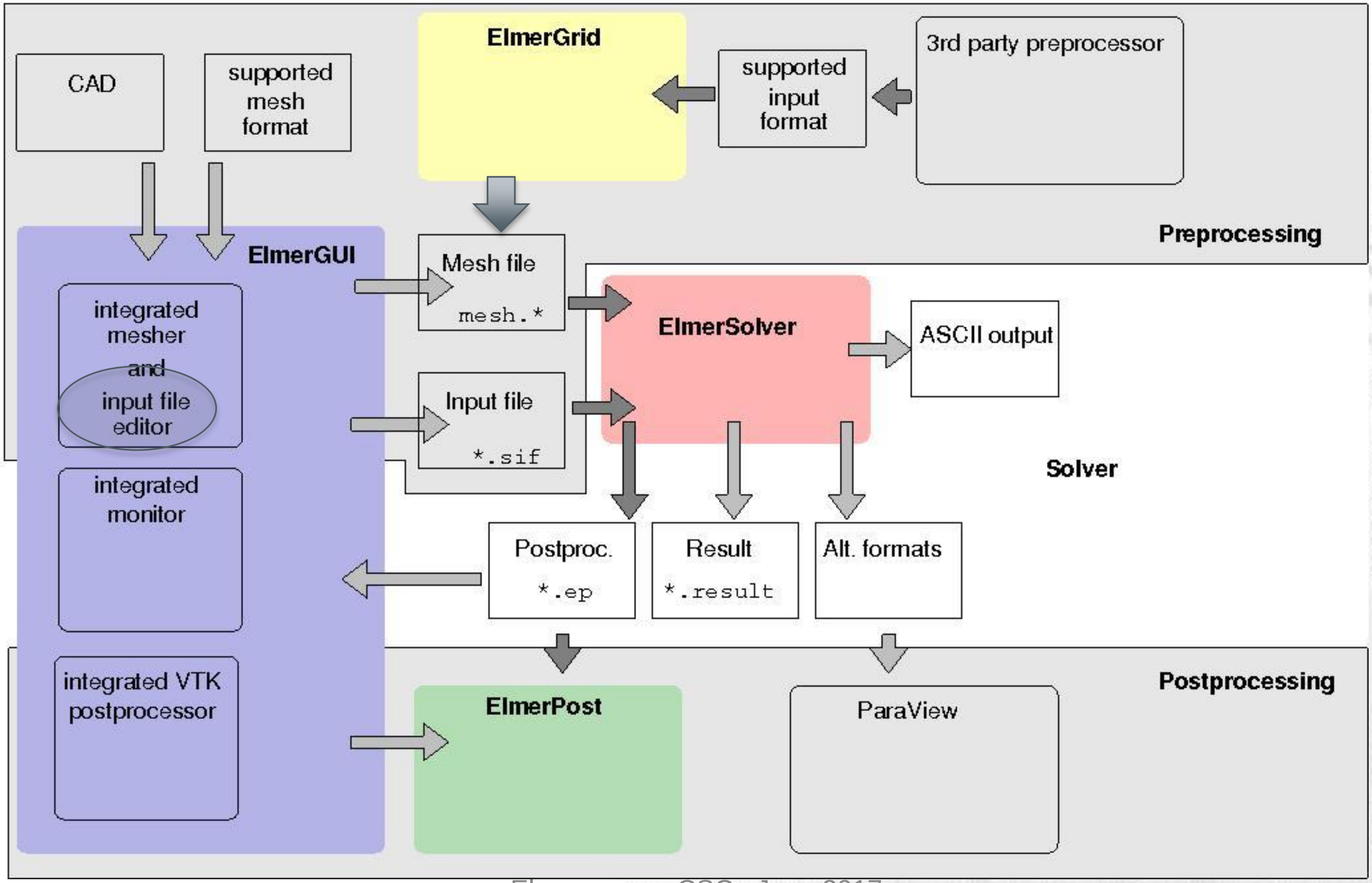
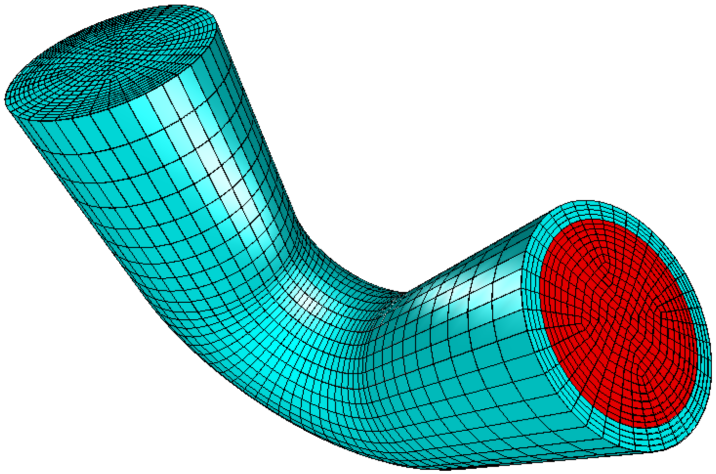
Thermal flow in a curved pipe – Explaining basic structure of an Elmer simulation

Elmer Team
CSC – IT Center for Science Ltd.

Elmer - Modules



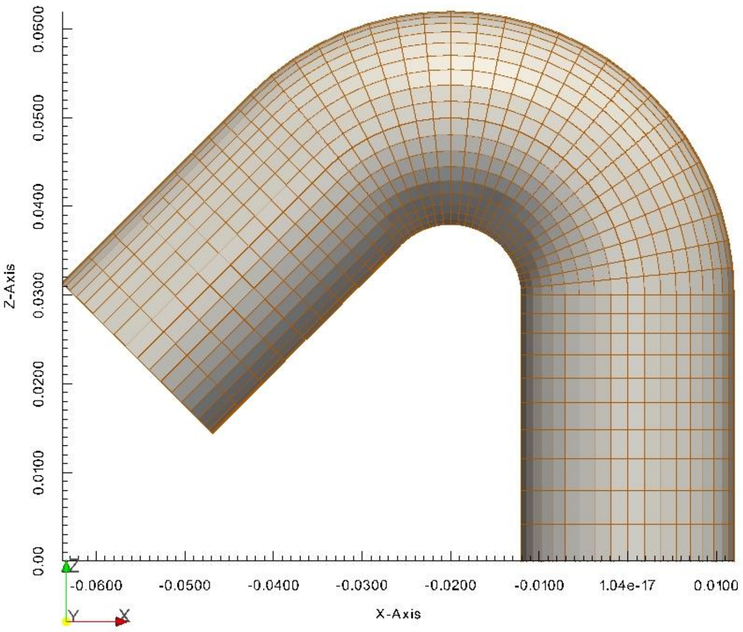
The problem



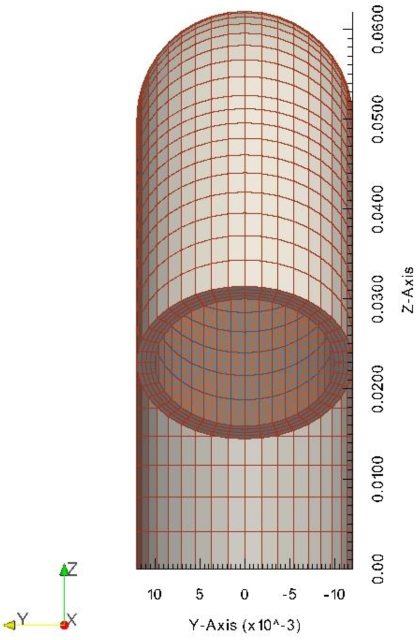
This is the current Tutorial 8
Thermal Flow in a curved pipe
in ElmerTutorials.pdf
(from nic.funet.fi)

- Pipe consisting of solid (iron) wall filled with fluid (water)
- We have a hot (350 K) inflow on one side of the pipe and cool the outside of the pipe at 300 K
- We prescribe inflow profile of water
- We are interested in steady state solution

The problem



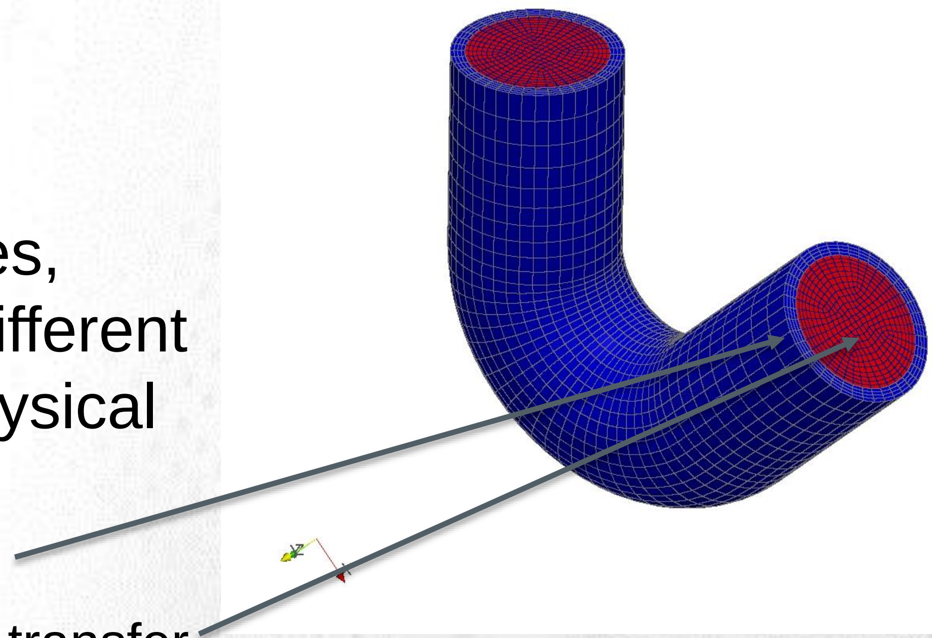
Z-Axis
0.00
0.0100
0.0200
0.0300
0.0400
0.0500
0.0600
-0.0600
-0.0500
-0.0400
-0.0300
-0.0200
-0.0100
1.04e-17
0.0100
X-Axis
Z
Y
X



Z
Y
X
10
5
0
-5
-10
Y-Axis (x10^-3)
0.00
0.0100
0.0200
0.0300
0.0400
0.0500
0.0600
Z-Axis

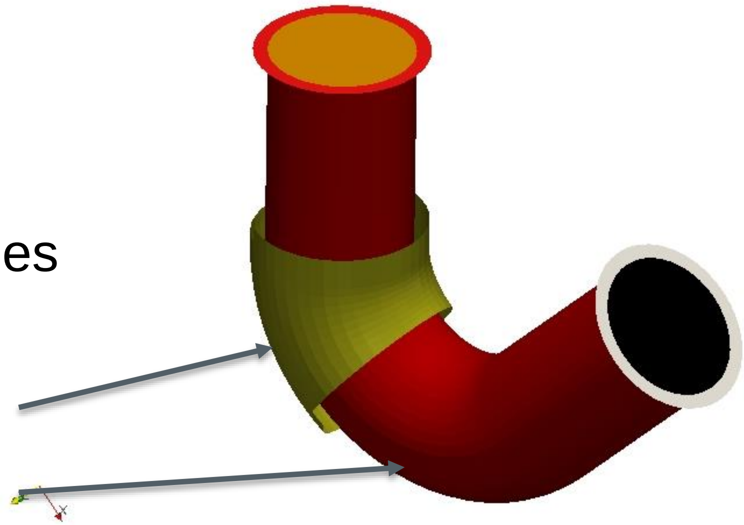
On bodies

- A **Body** is a distinguishable part of the computational domain
  - Geometry
  - Physical model(s)
  - Material properties
- Here we have two bodies, because we have two different materials (+ different physical models)
  - Solid (iron): heat transfer
  - Fluid (water): flow + heat transfer



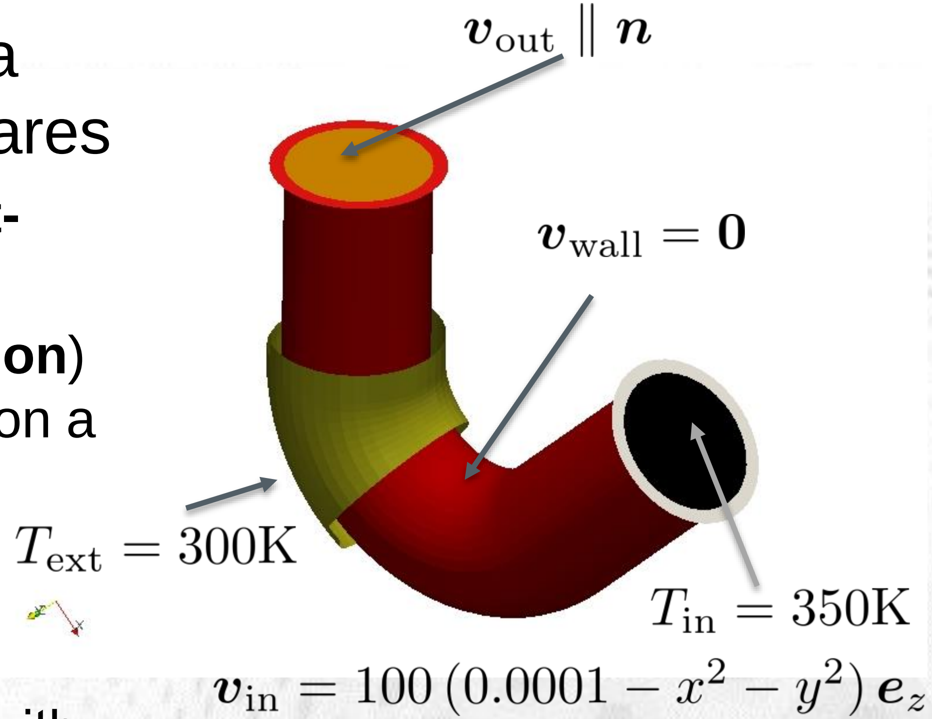
On boundaries

- A **Boundary** is a distinguishable lower-dimensional entity of the computational domain
  - In 3D: surfaces, lines and nodes
  - In 2D: lines and nodes
  - Can confine a body (external)
  - Can be situated in between 2 bodies (internal)
- Here we have several outside- and internal surface **boundaries**
  - can be viewed with ParaView



On boundaries

- A **Boundary Condition** is a set of instructions that declares
  - values of variables (**Dirichlet-condition**) or their normal
  - gradients (**Neumann-condition**) or mixed (**Robin-condition**) on a boundary
- Mind: BC's can apply to multiple boundaries
  - Don't interchange boundary with boundary condition



Suggestion: if you want to, you can start a little bit easier by just imposing a constant inflow velocity of 0.01

On solvers

- We talk of **Solvers** in terms of different physical models formulated by PDE's

- Heat transfer

$$\rho\, c\, (\partial T / \partial t + \boldsymbol{u} \cdot \nabla T) = \nabla \cdot (\kappa \nabla T) + \rho \sigma$$

- Navier-Stokes

$$\rho\, (\partial \boldsymbol{u} / \partial t + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla\, p + \nabla \cdot (\mu \dot{\boldsymbol{\epsilon}}(\boldsymbol{u})) + \rho \boldsymbol{f}$$

Material

- A **Material** defines the physical parameters

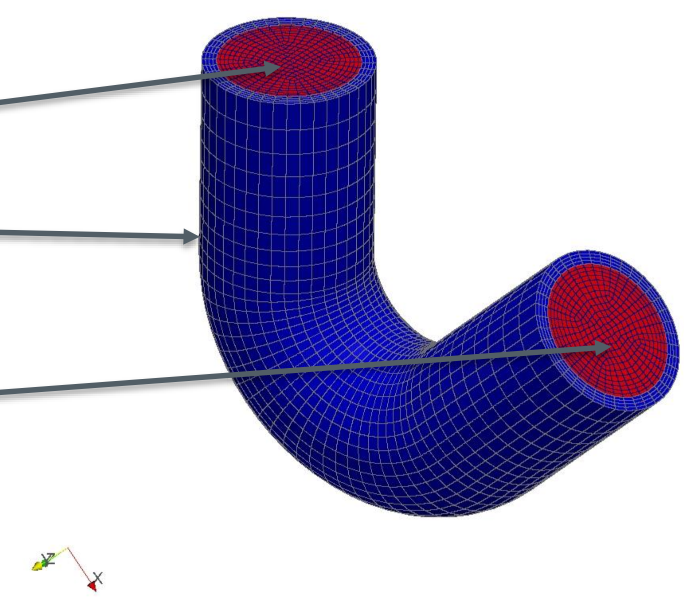
- Heat transfer

  $\rho,\ c,\ \kappa$

- Navier-Stokes

  $\rho,\ \mu$

- In our case we used material library in GUI



Bodyforce

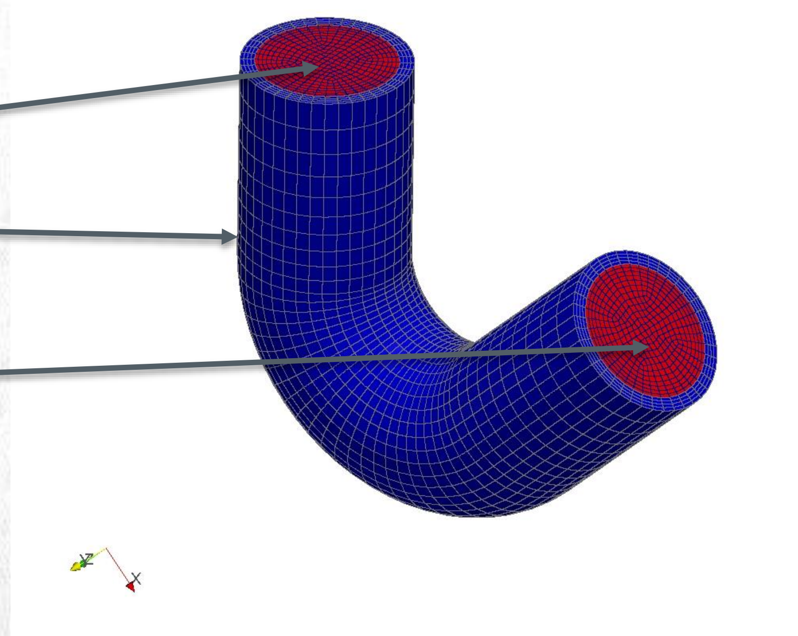
- A **Body Force** defines the right-hand side of the equations
- Heat transfer

  σ

- Navier-Stokes

  f

- Just theoretical, as we do not apply in this case



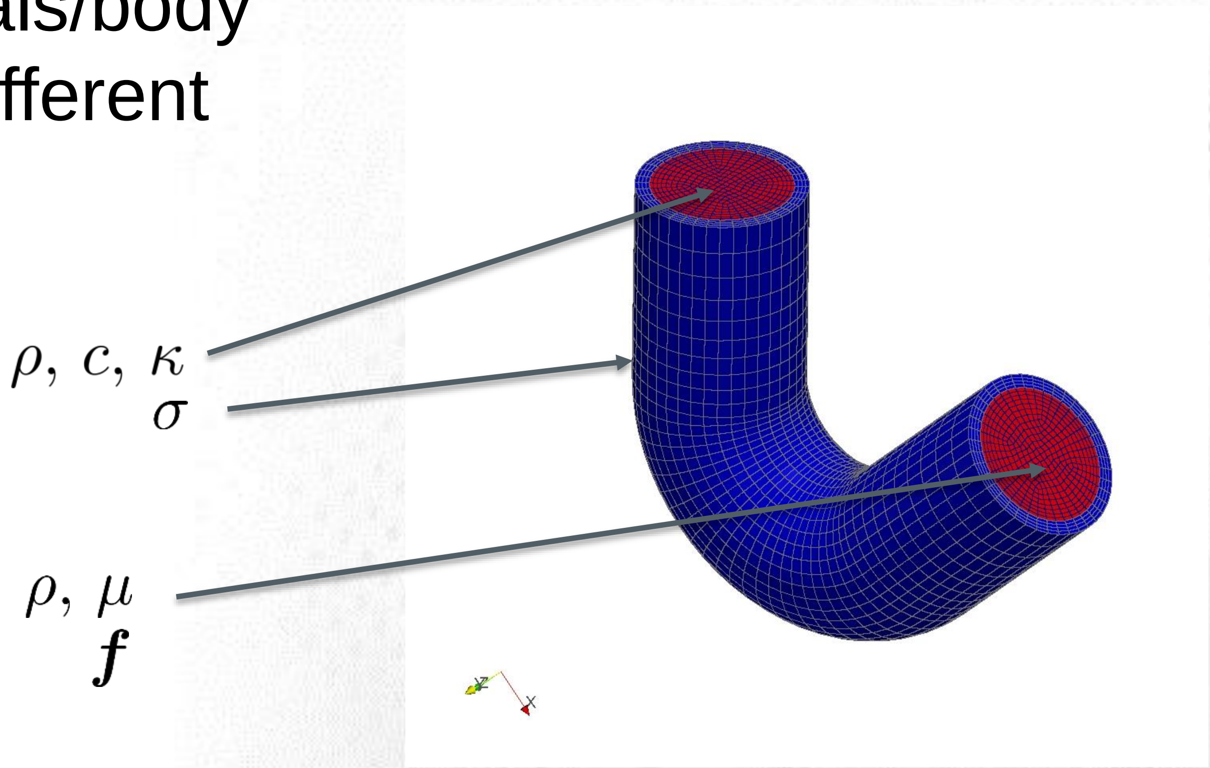
Equation

- An **Equation** assigns the solvers/materials/body forces to the different bodies
- Heat transfer

$$\rho\, c\, (\partial T/\partial t + \boldsymbol{u} \cdot \nabla T) = \nabla \cdot (\kappa \nabla T) + \rho \sigma$$

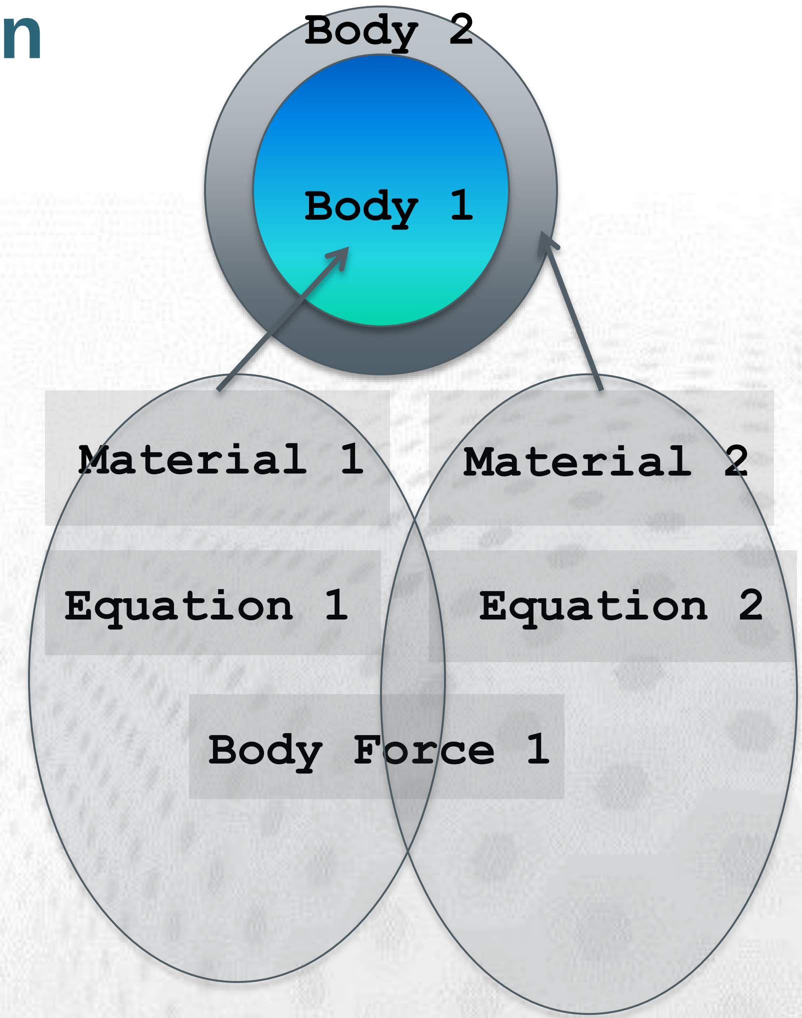
- Navier-Stokes

$$\rho\, (\partial \boldsymbol{u}/\partial t + \boldsymbol{u} \cdot \nabla \boldsymbol{u}) = -\nabla\, p + \nabla \cdot (\mu \dot{\boldsymbol{\epsilon}}(\boldsymbol{u})) + \rho \boldsymbol{f}$$



Equation

- Each **Body** has to have an **Equation** and **Material** assigned
- **Body Force**, **Initial Condition** are optional
- Two bodies can have the same **Material/Equation/ Body Force/Initial Condition** section assigned



Further settings to change

- Setup
  - Change **`case.ep`** into **`case.vtu`** in order to obtain output for ParaView
  - For restart, type into Free text input field:

    `Output File = case.result`

- Equation
  - Heat and Flow
  - Tab: Heat Equation
  - Edit Solver Settings
  - The Material parameters for heat transfer are constant. Hence this is a linear problem in terms of the variable Temperature:

    `Nonlinear System Max Iterations = 20 → 1`

Thermal flow in a curved pipe - Variations on the tutorial case using modifications of the text input file: coupling, MATC, User Defined Functions

Elmer Team
CSC – IT Center for Science Ltd.

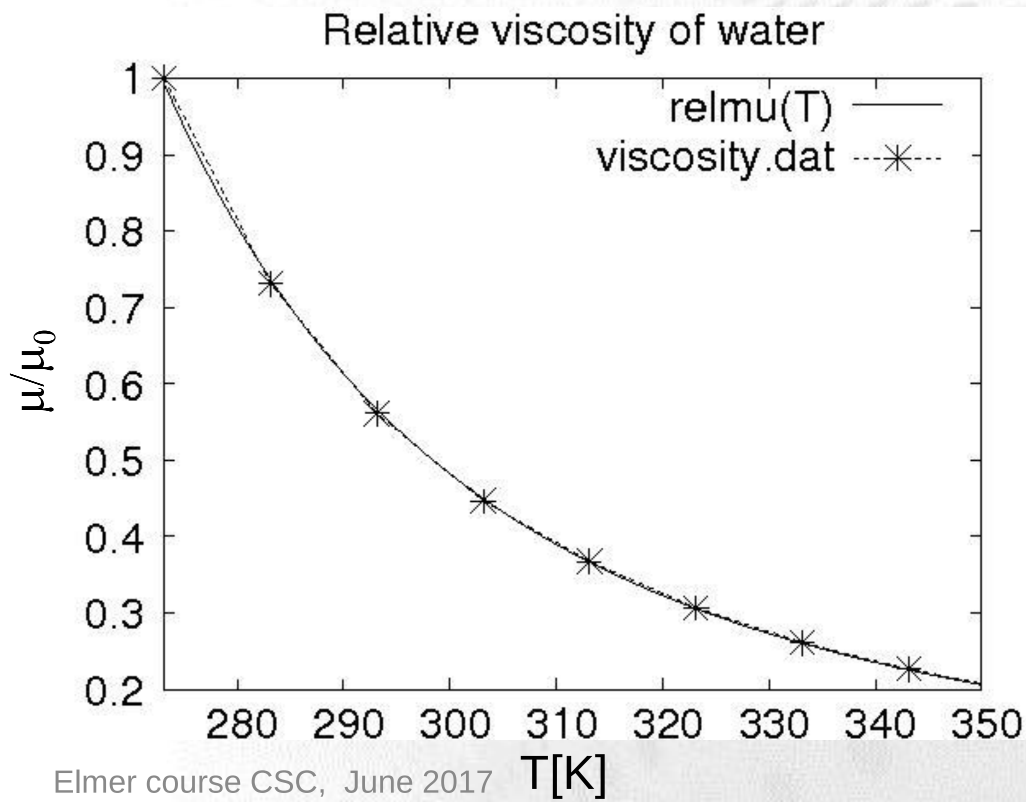
Variations – 2 way coupling

- Temperature dependence of the viscosity for liquid water

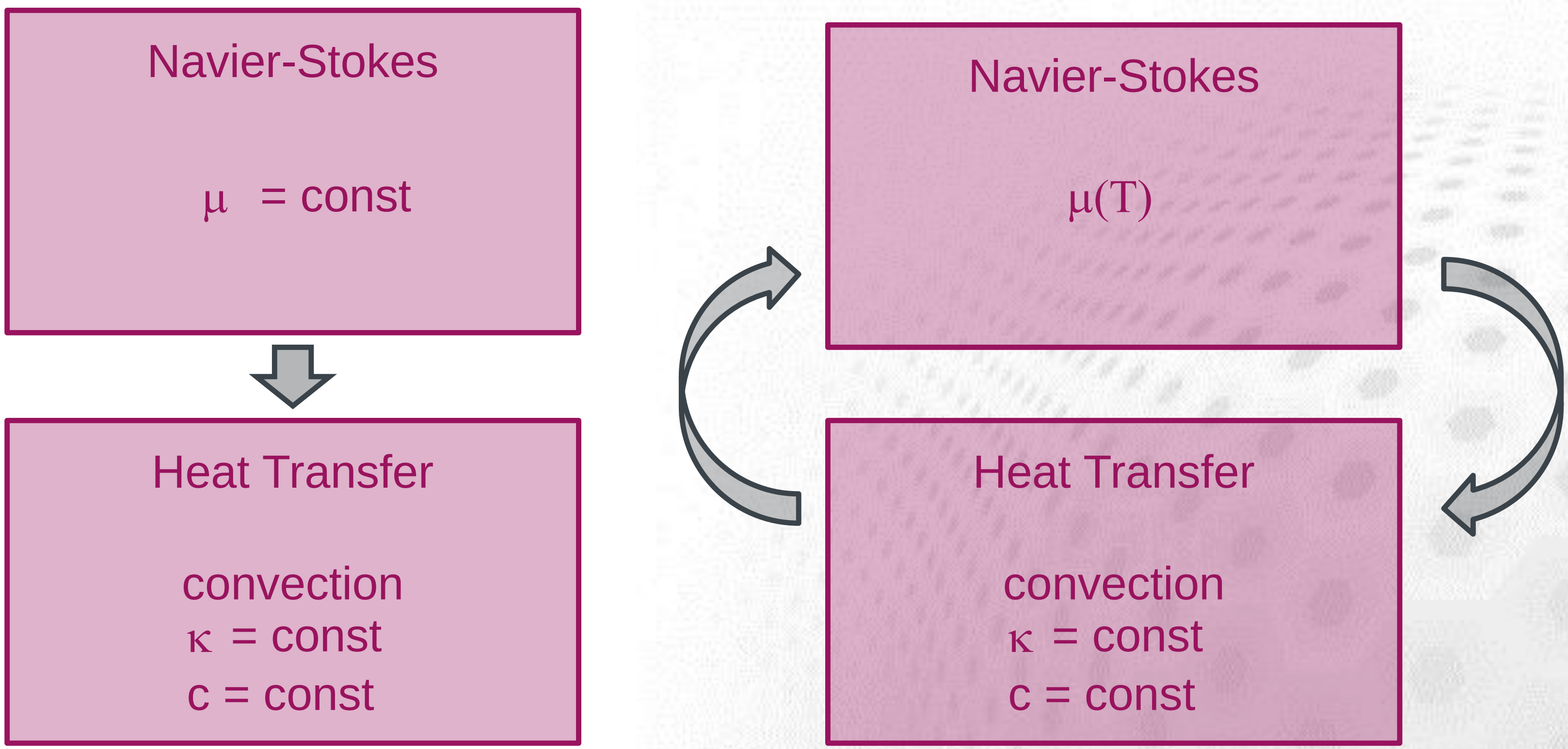
$$\mu/\mu_0 = \exp(-1.704 \; -5.306 \; 273.15/T + 7.003 \; (273.15/T)^2)$$

viscosity.dat

```
273.15  1.788e-3
283.15  1.307e-3
293.15  1.003e-3
303.15  0.799e-3
313.15  0.657e-3
323.15  0.548e-3
333.15  0.467e-3
343.15  0.405e-3
353.15  0.355e-3
363.15  0.316e-3
373.15  0.283e-3
```



Variations – 2 way coupling



`Steady State Max Iterations = 1 → 50`

Variations – 2 way coupling

- Copy the original solver input file (SIF)
- Open in editor of your choice (e.g., gedit)
  - apply the changes as suggested
  - change names of output files!
  - Include restart from earlier case:

```
Restart File = case.result
Restart Position = 0
```

  - The last line restarts from the last entry it found in **case.result**

Array 1

- Piecewise linear interpolation
- Alternative:

  `Real cubic`

  interpolates using cubic splines
- See SIF:

`coupled_array.sif`

```
Material 1
  Name = "Water (room temperature)"
  Viscosity = Variable Temperature
    Real
      273.15 1.788e-3 ! 0 Celsius
      283.15 1.307e-3
      293.15 1.003e-3
      303.15 0.799e-3
      313.15 0.657e-3
      323.15 0.548e-3
      333.15 0.467e-3
      343.15 0.405e-3
      353.15 0.355e-3
      363.15 0.316e-3
      373.15 0.283e-3 ! 100 Celsius
   End
```

Variations – 2 way coupling

- Save under case **`coupled_array.sif`**
- Run the case in serial:

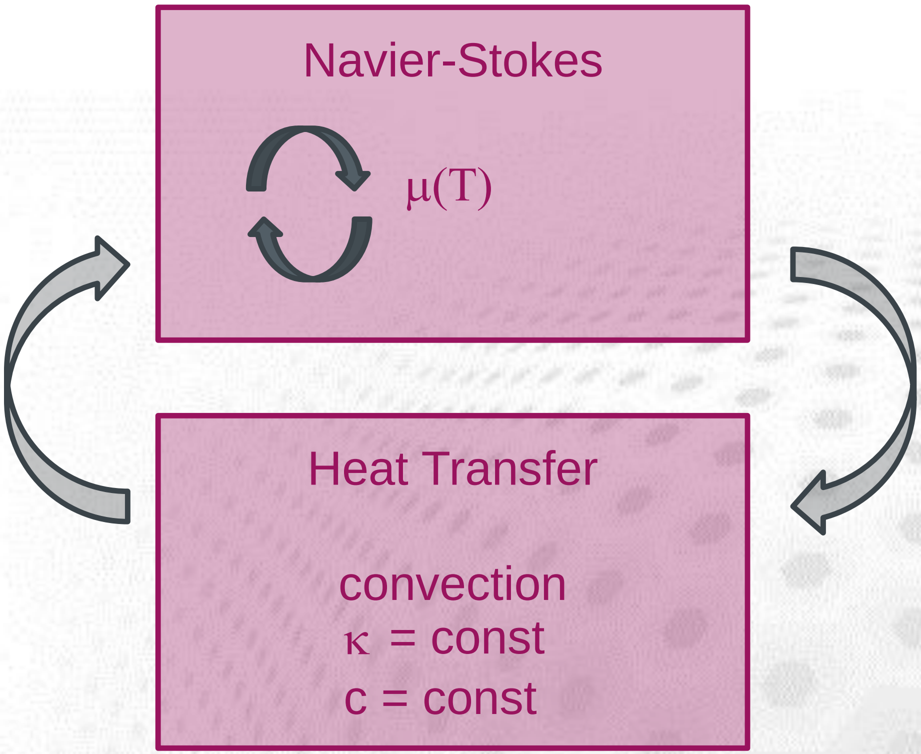
```
ElmerSolver coupled_array.sif >
  coupled_array.log &
```

  – Redirect output (good for checking performance)

Array 2

- Same as before, but now we switch to only one non-linear iteration for Navier-stokes
- Create new SIF:

`coupled_array_var.sif`



`Nonlinear System Max Iterations = 50 → 1`

MATC function

- Declare outside sections:
  - Constant **`mu0`**
  - Function **`relativevisc`**
- Call both using MATC from within **`Material 1`**

```
$ mu0 =  1.788e-3
$ function relativevisc(T){\
  a = -1.704;\
  b = -5.306;\
  c = 7.003;\
  z = 273.15/T;\
  _relativevisc = exp(a + b * z + c *(z^2));\
}
```

```
Material 1
  Name = "Water (room temperature)"
  Viscosity = Variable Temperature
    Real MATC "mu0 * relativevisc(tx)"
```

User Defined Function (UDF)

- Write a simple UDF in Fortran 90 that returns the value of viscosity from a given value of temperature **`viscosity1.f90`**
  - Pre-defined Header:

```
FUNCTION getWaterViscosity( Model, N, temperature ) &
RESULT(viscosity)
  USE DefUtils
  IMPLICIT NONE
  !------------ external variables ---------------------------
  TYPE(Model_t) :: Model
  INTEGER :: N
  REAL(KIND=dp) :: temperature, viscosity
```

NB for F90: exponential function … exp()   multiplication … *

User Defined Function (UDF)

- Compile it:

```
elmerf90 viscosity1.f90 –o viscosity1
```

- Re-write the Material 1 section:

```
Material 1
  Name = "Water (room temperature)"
  Viscosity = Variable Temperature
    Procedure "viscosity1" "getWaterViscosity"
```