Imperial College London

Fluid-Structure Interaction Problems using SU2 and External Finite-Element Solvers

R. Sanchez[1], D. Thomas[2], R. Palacios[1], V. Terrapon[2]

[1]Department of Aeronautics, Imperial College London
[2]Department of Mechanical and Aerospace Engineering, University of Liege

First SU2 Annual Developers Meeting
TU Delft, 6 September 2016

Imperial College London

Outline

Imperial College
London

Objectives

- Development of FSI capabilities in SU2
- Target: static and dynamic aeroelastic simulation
- ALE solution from CFD solver needs:
  - Mesh deformation
  - Interface (interpolation)
  - Structural solver (FEA)
- Two approaches to the structural solver
  - Native implementation within SU2 source code
  - Python wrapper to couple with third-party solvers

SU2
The Open-Source CFD Code

http://su2.stanford.edu

Native fluid solver[1]

Arbitrary Lagrangian-Eulerian (ALE) formulation:

$$\frac{\partial \mathbf{U}}{\partial t} + \nabla \cdot \mathbf{F}^{\mathbf{c}}(\mathbf{U}, \dot{\mathbf{u}}_{\mathbf{\Omega}}) - \nabla \cdot \mathbf{F}^{\mathbf{v}}(\mathbf{U}) - \mathbf{Q} = \mathbf{0} \qquad \text{in } \Omega_f \times [0, t]$$

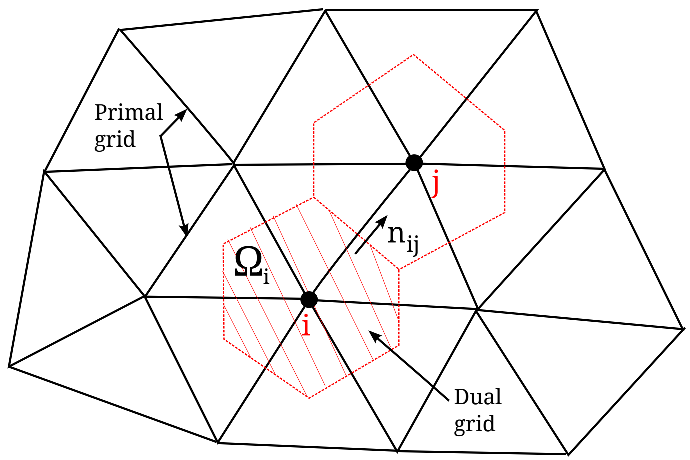
Space integration

- Dual-grid, edge-based discretization
- Vertex-based control volumes

Time integration

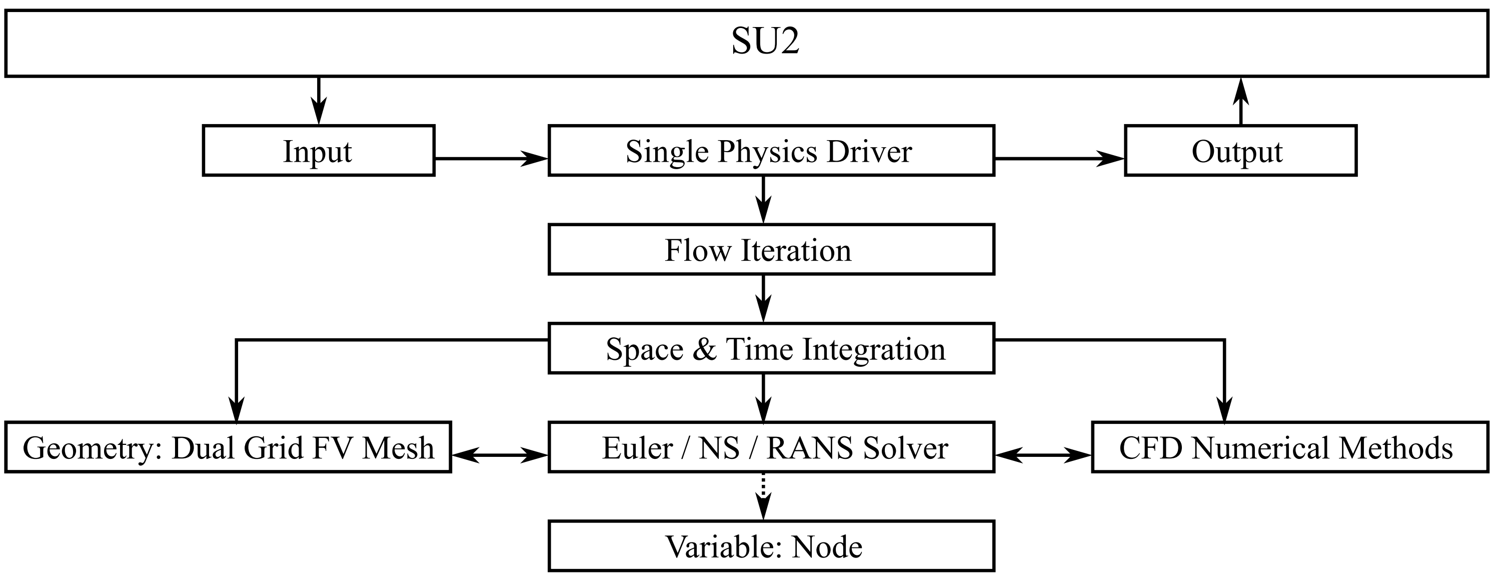
- Dual time-stepping strategy

$$\frac{\partial \mathbf{U}}{\partial \tau} + R^*(\mathbf{U}) = 0$$



[1] Palacios, F. *et al.* (2013)

Native fluid solver: code structure



Imperial College London

Solid mechanics

$$\left.\begin{array}{l}\text{Large deformations}\\ \text{Complex material behaviour}\end{array}\right\} \rightarrow \text{Finite deformation framework}$$

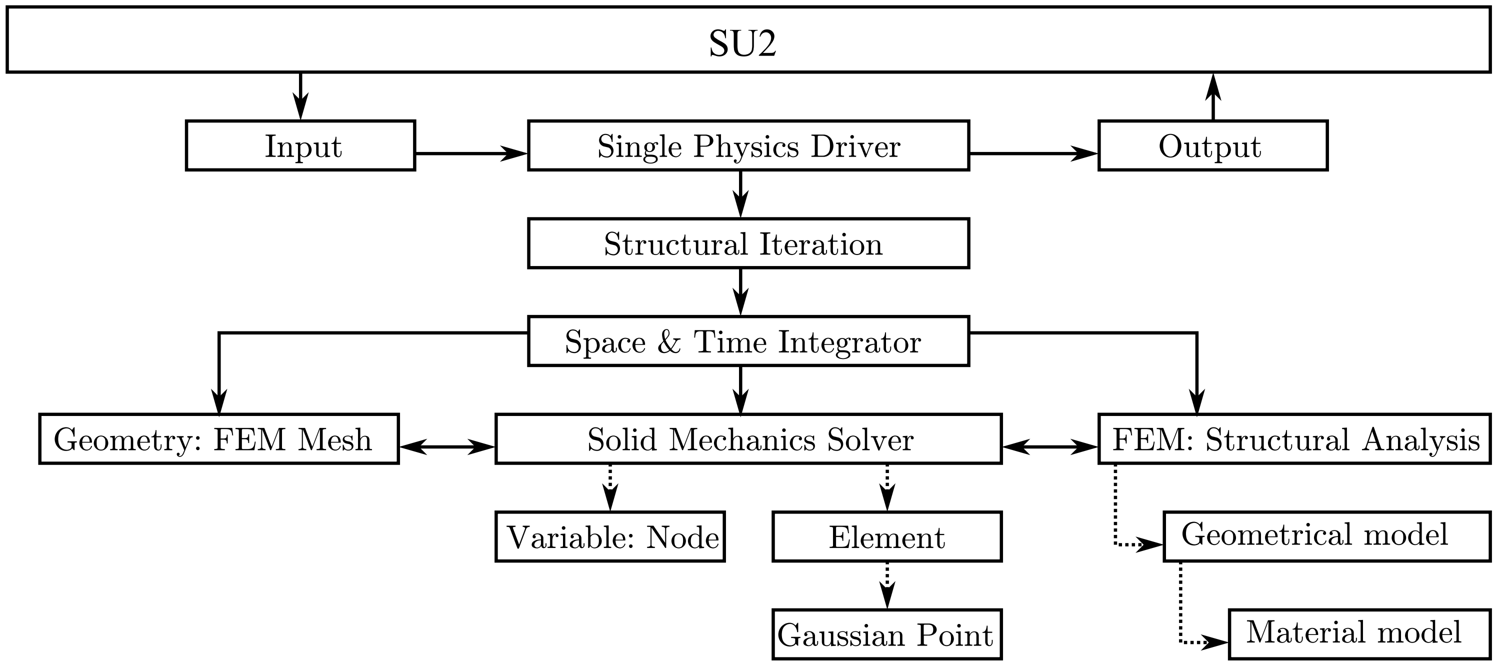
For static problems:

$$\begin{aligned}\mathscr{S}(\mathbf{x}) &= \delta W^{int} && -\delta W^{ext} && = \mathbf{0}\\ &= \int_v \boldsymbol{\sigma} : \delta \mathbf{d}\ dv && - \left(\int_v \mathbf{f} \cdot \delta \mathbf{v}\ dv + \int_{\partial v} \mathbf{t} \cdot \delta \mathbf{v}\ da \right) && = \mathbf{0}\end{aligned}$$

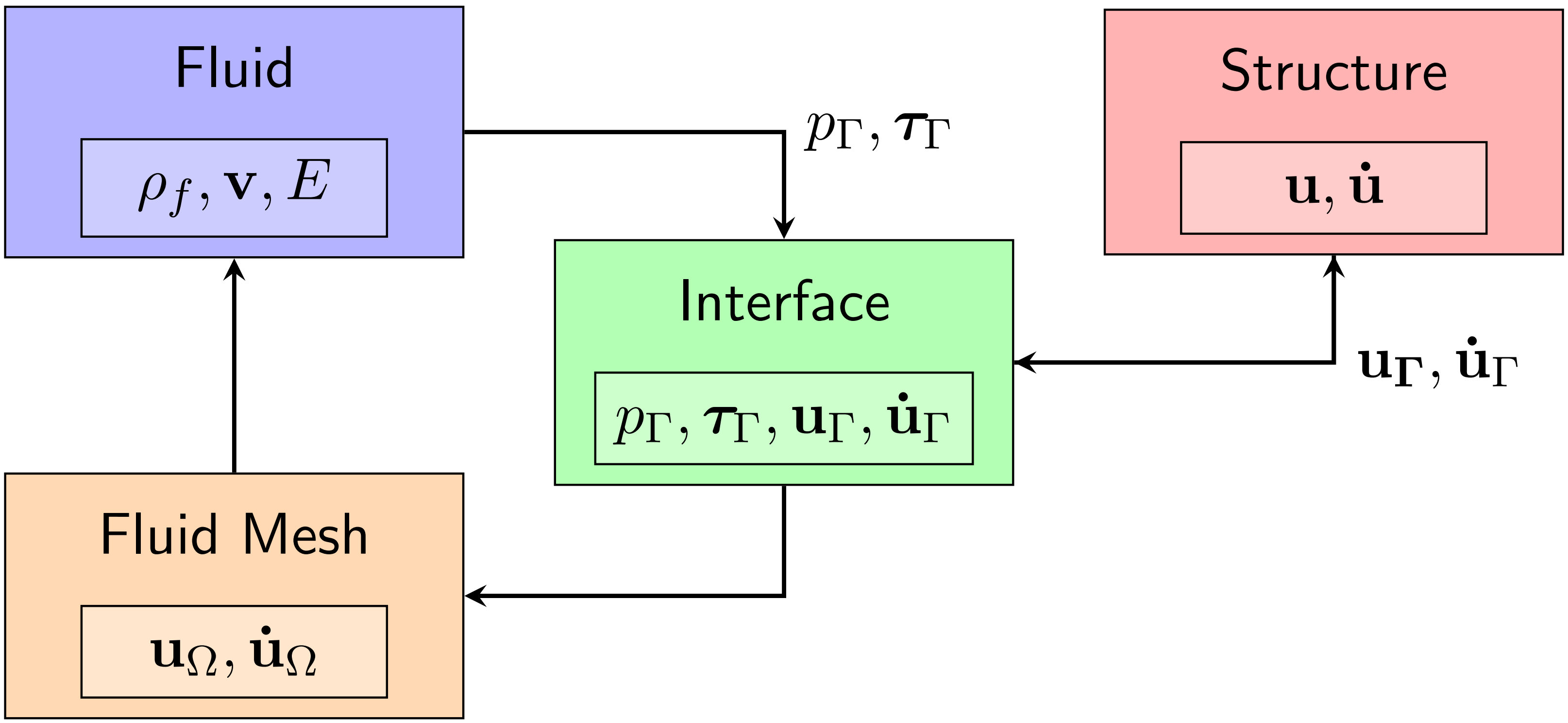
Linearising, $\dfrac{\partial \mathscr{S}(\mathbf{x})}{\partial \mathbf{x}} \Delta \mathbf{x} = \mathbf{K} \Delta \mathbf{x} = -\mathscr{S}(\mathbf{x})$ where $\mathbf{K} = \mathbf{K}_c + \mathbf{K}_{\boldsymbol{\sigma}} - \mathbf{K}_{ext}$

Integration using Finite Element Method
After inertia is added, time integration using Newmark and generalized-α methods

Native structural solver: code structure

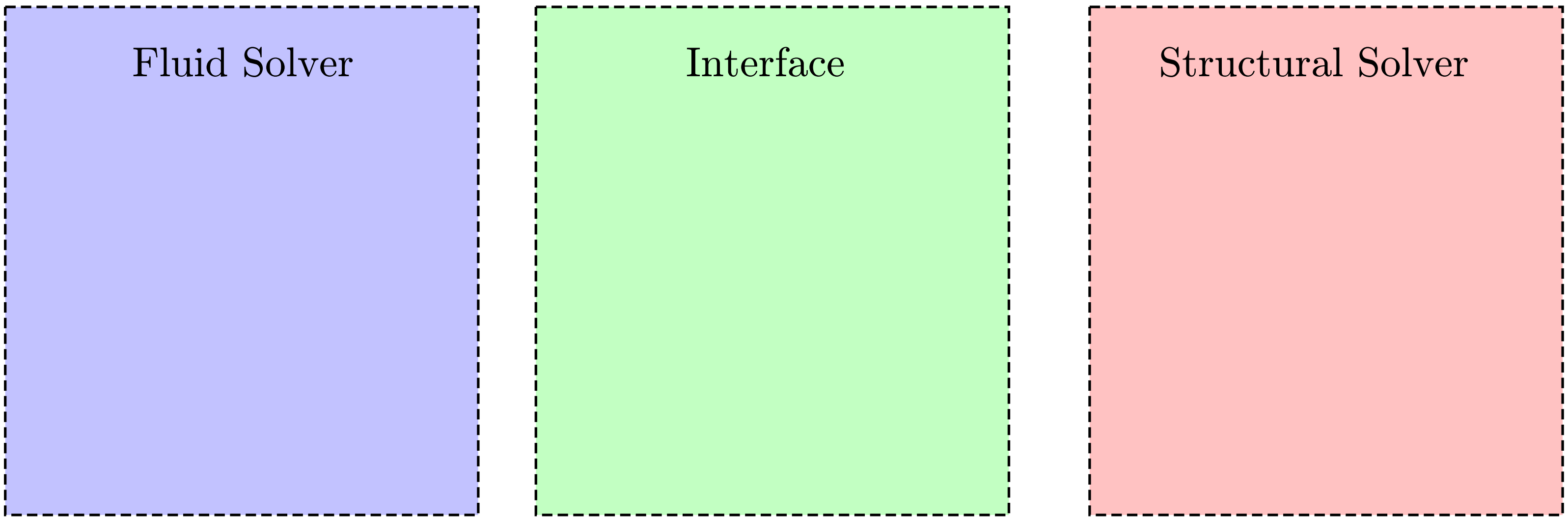


Native FSI solver*

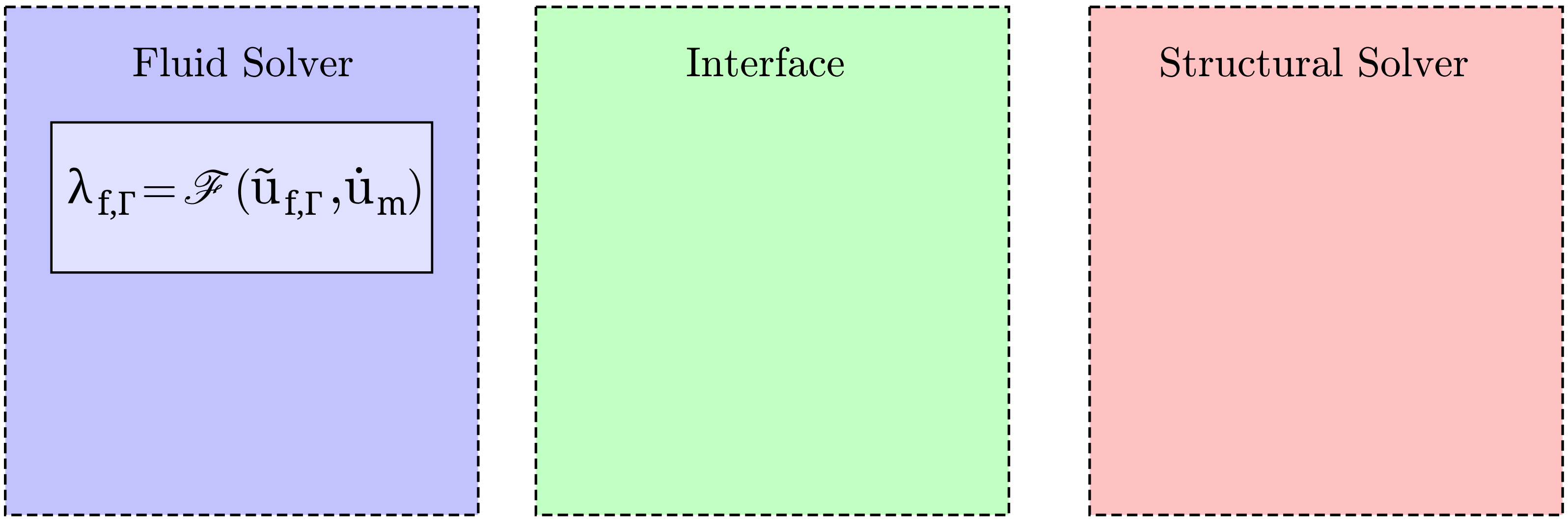


* Sanchez, R. *et al.* (SciTech, 2016)

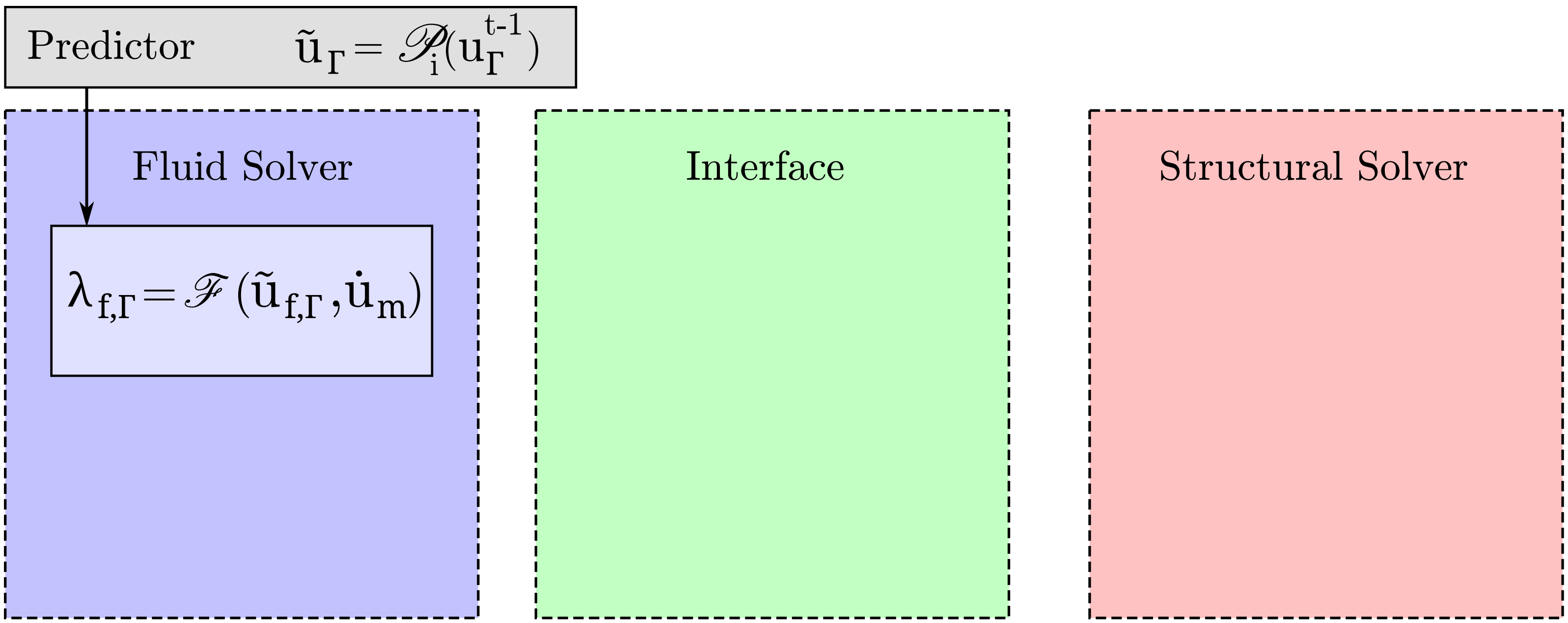
Native FSI solver: Time coupling



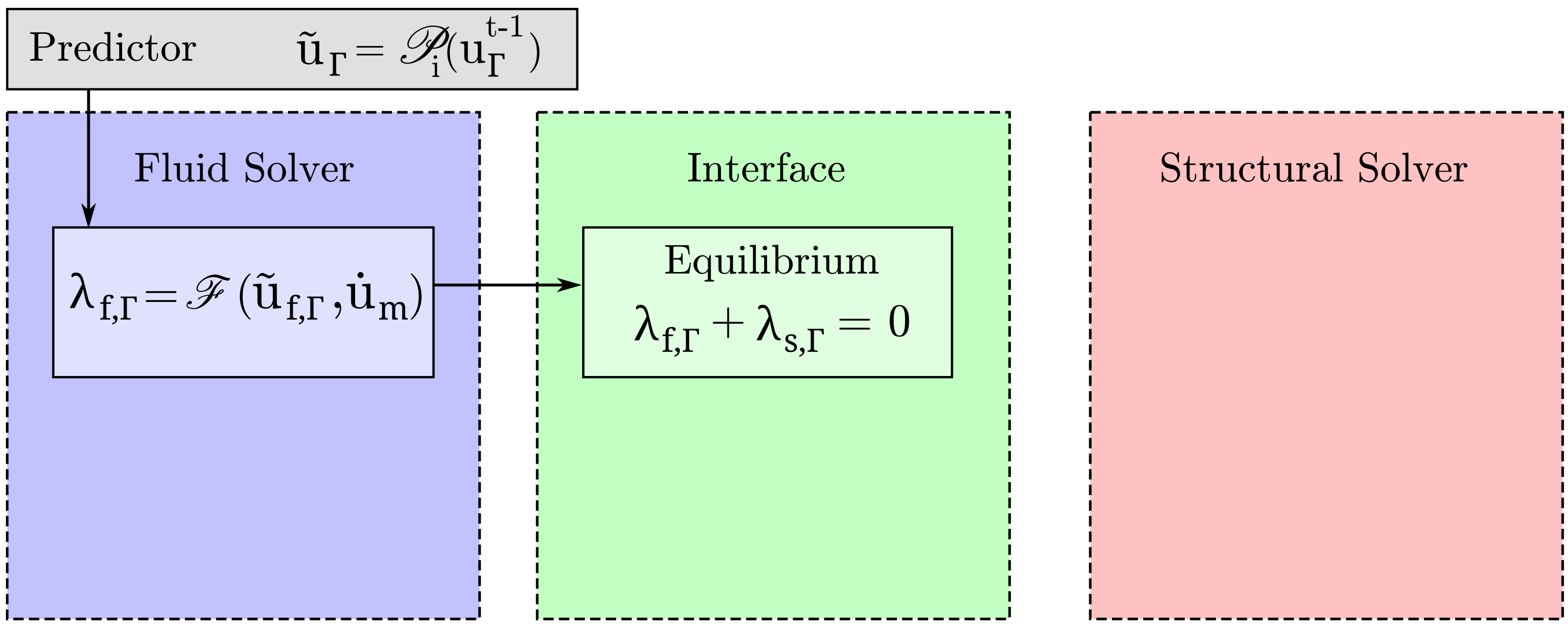
Native FSI solver: Time coupling



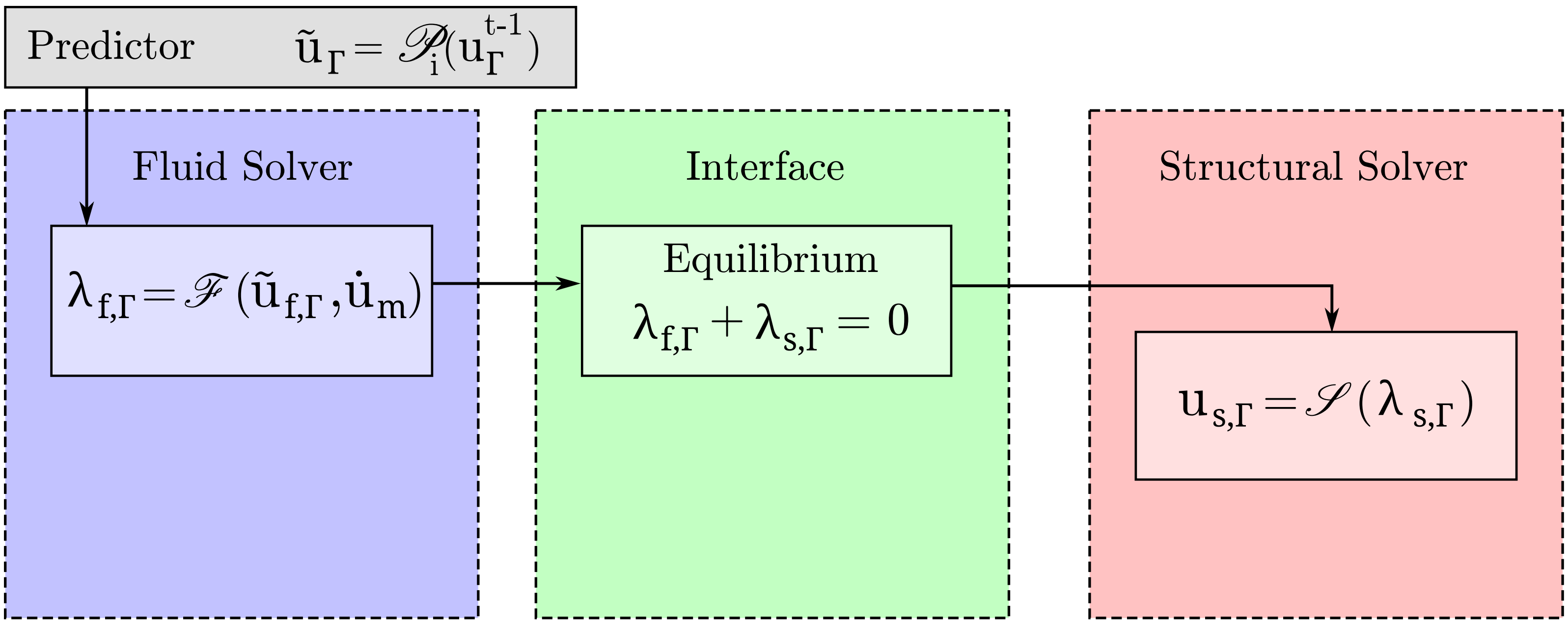
Native FSI solver: Time coupling



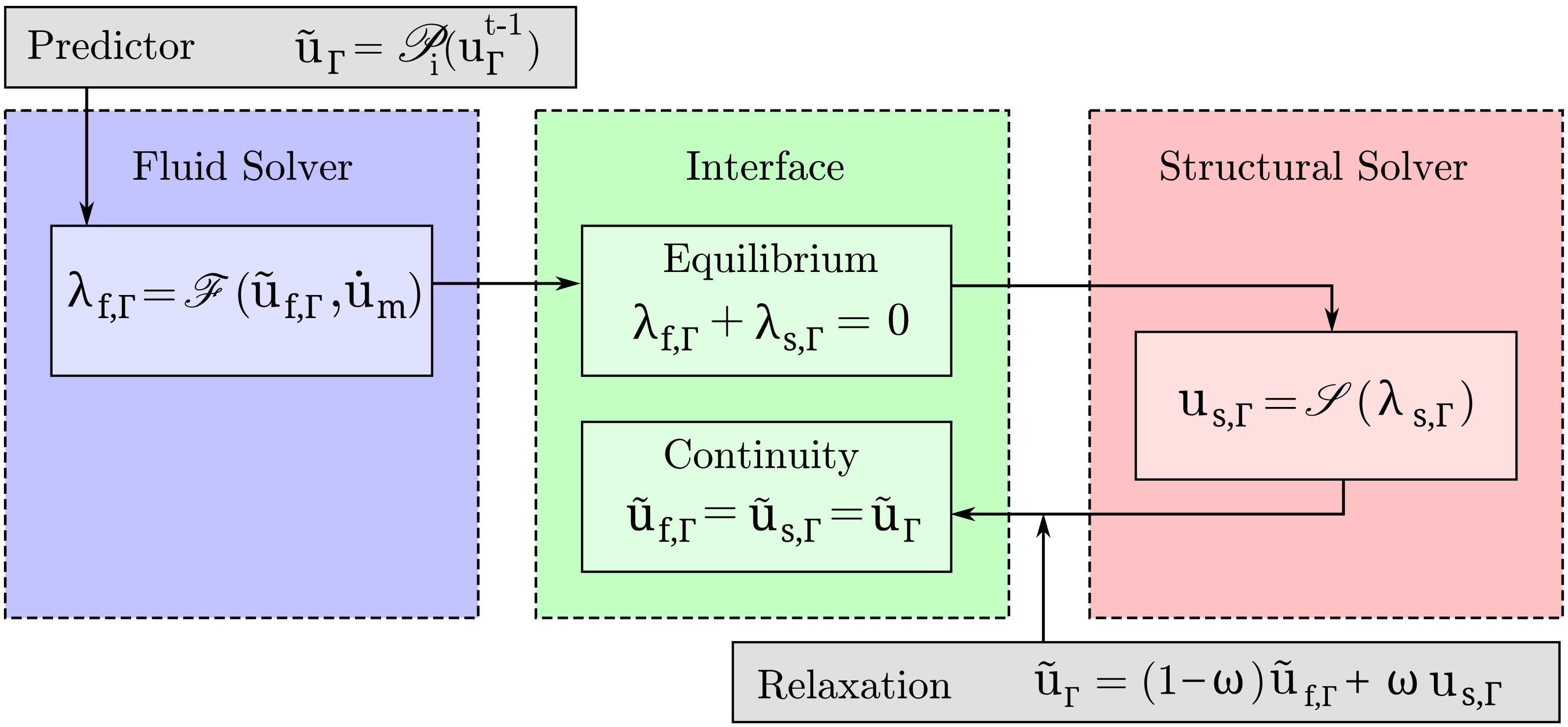
Native FSI solver: Time coupling



Native FSI solver: Time coupling

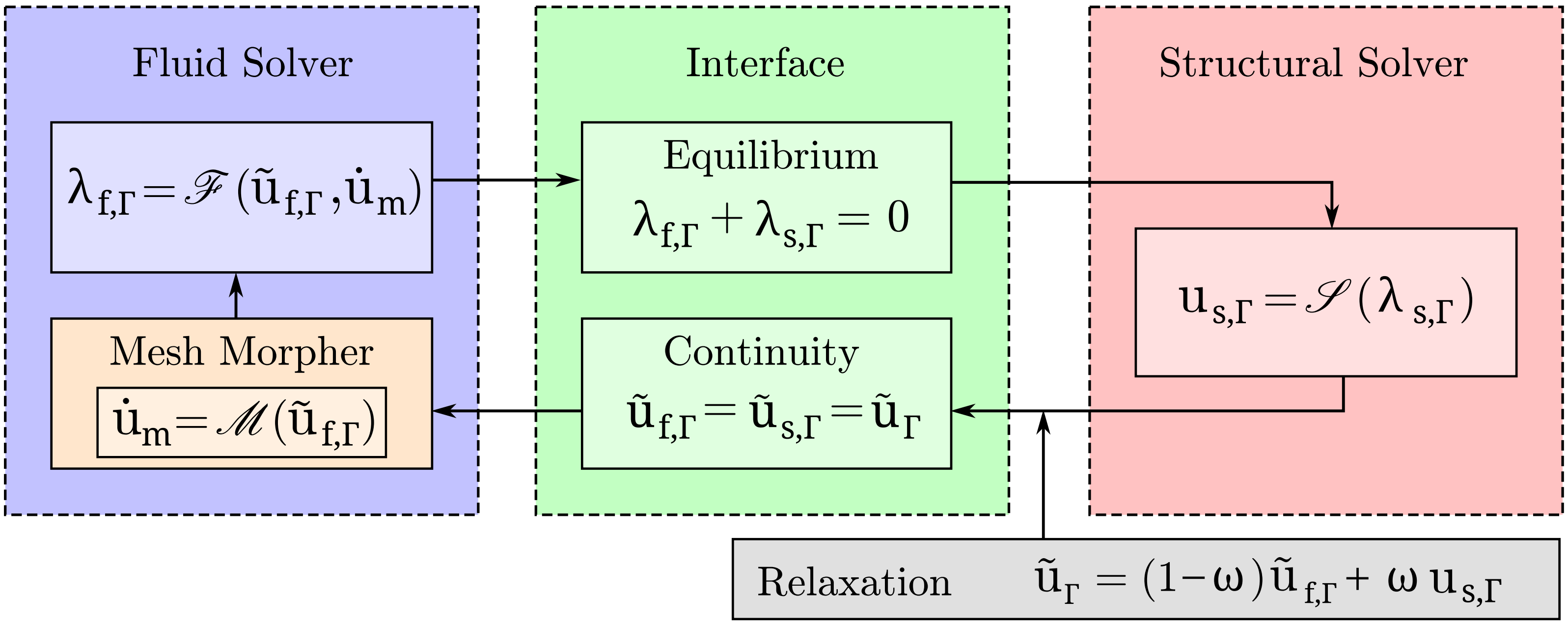


Native FSI solver: Time coupling

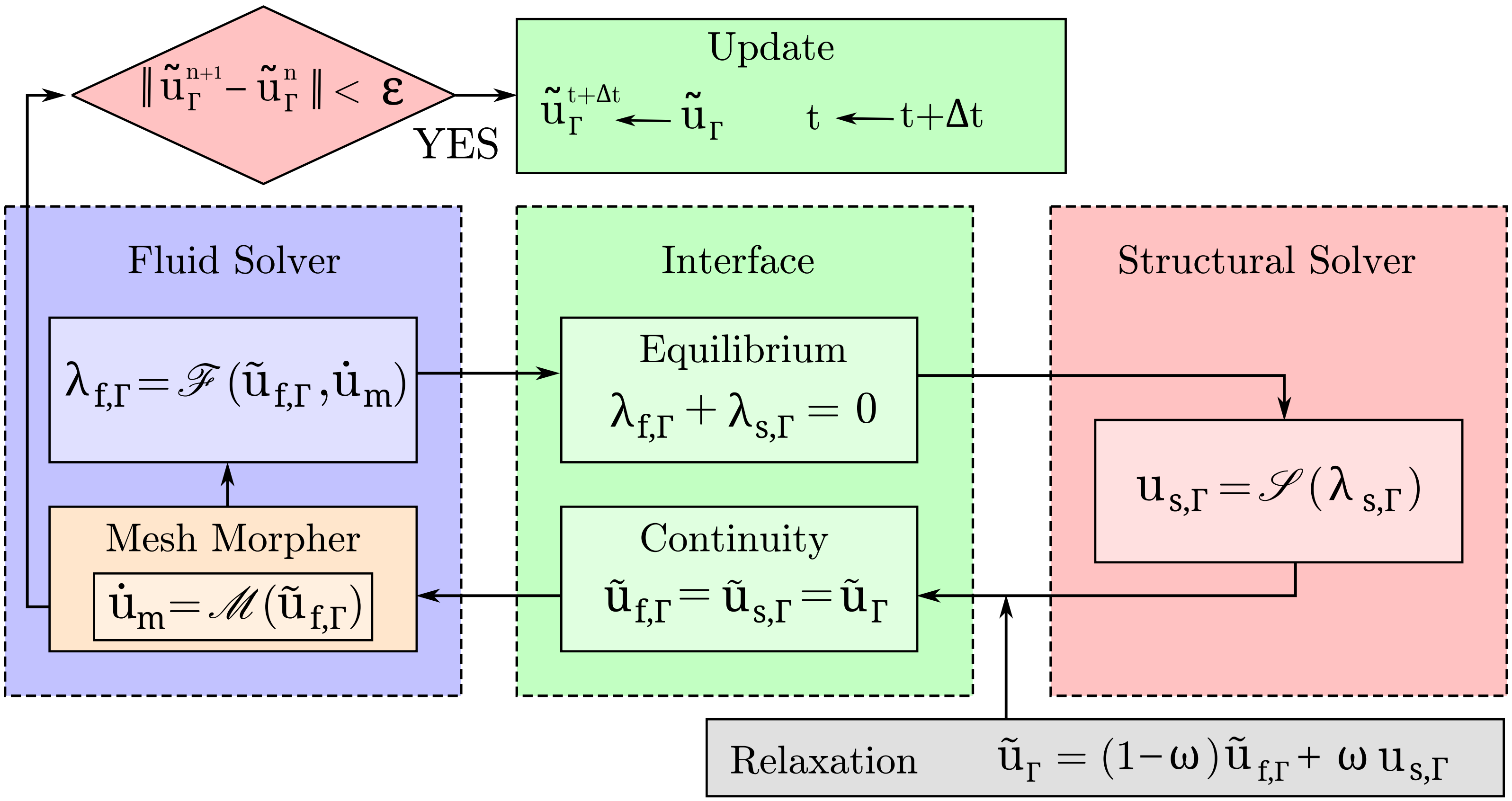


Imperial College London

Native FSI solver: Time coupling

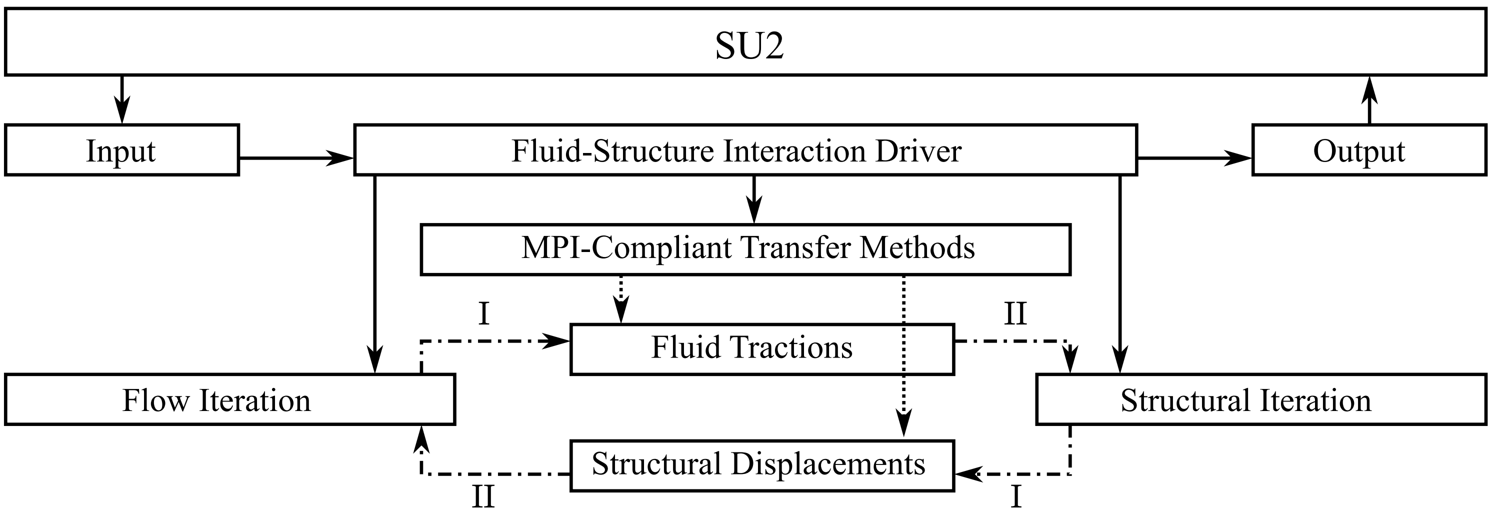


Native FSI solver: Time coupling

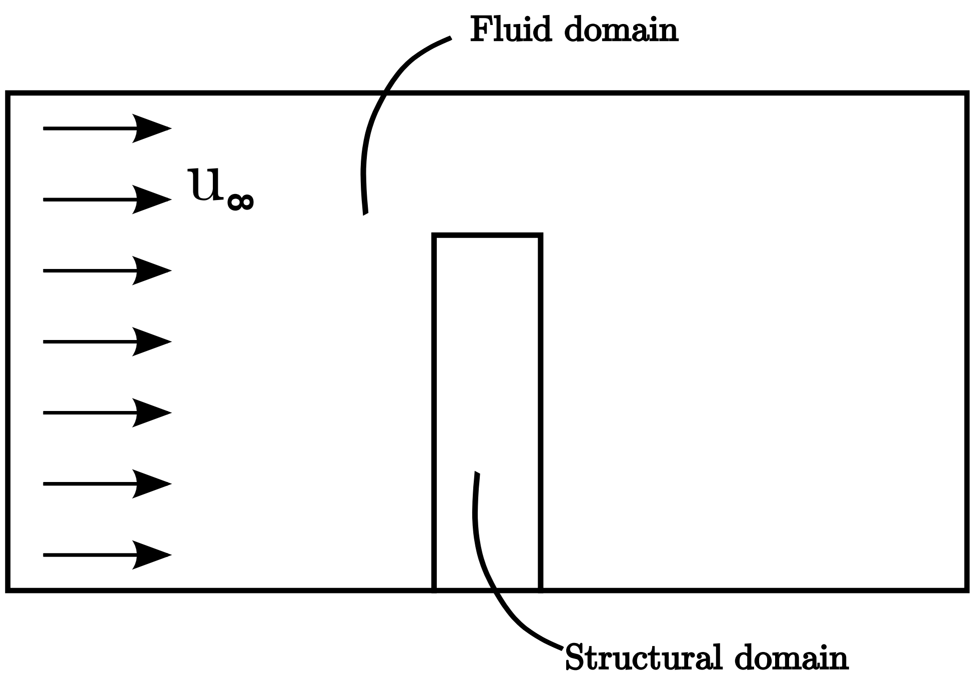


Imperial College London

Native FSI solver: Time coupling structure

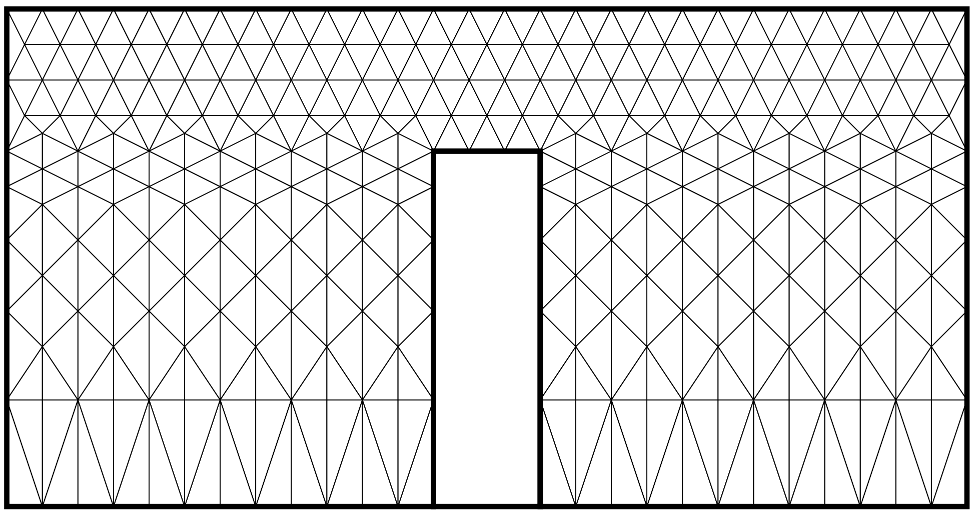


Native FSI solver: Spatial coupling



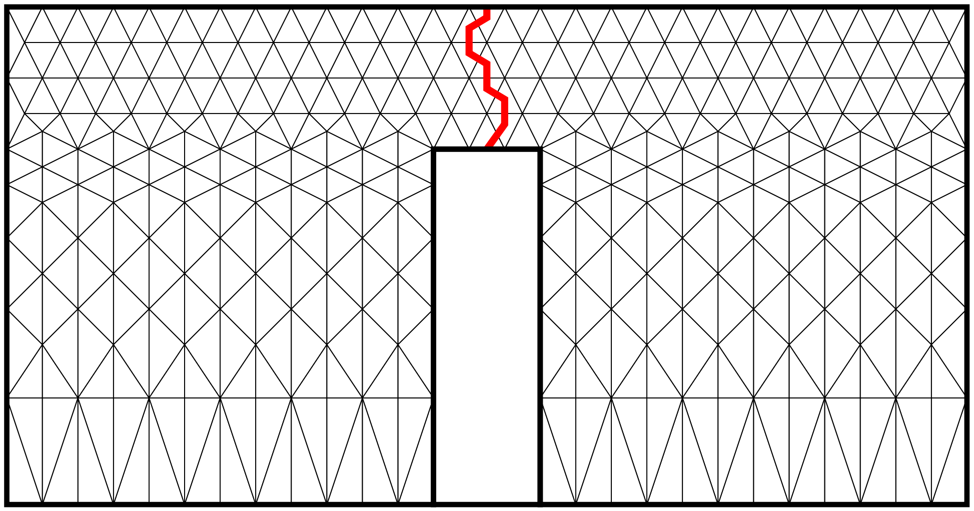
Imperial College London

Native FSI solver: Spatial coupling



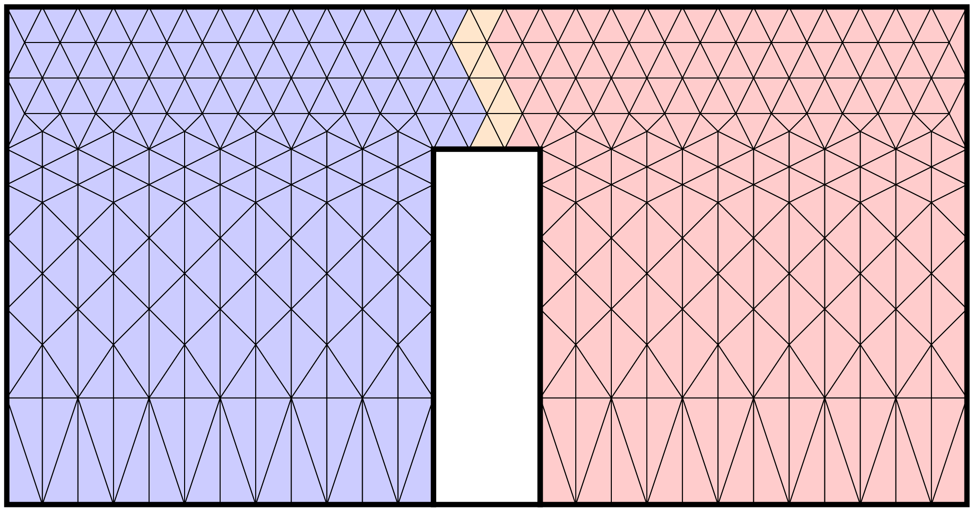
Imperial College
London

Native FSI solver: Spatial coupling



Automatic domain decomposition

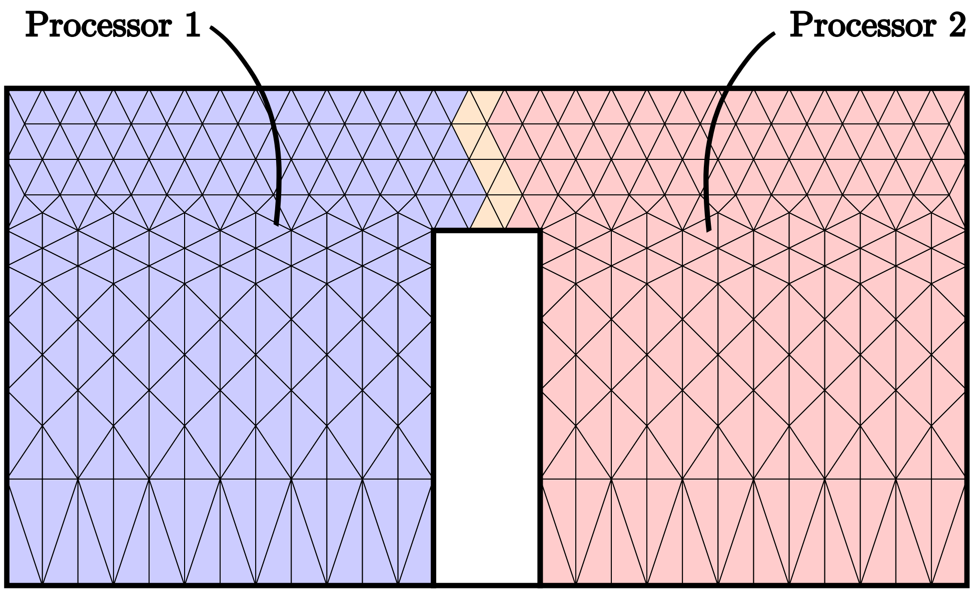
Native FSI solver: Spatial coupling



Automatic
domain
decomposition

Imperial College
London

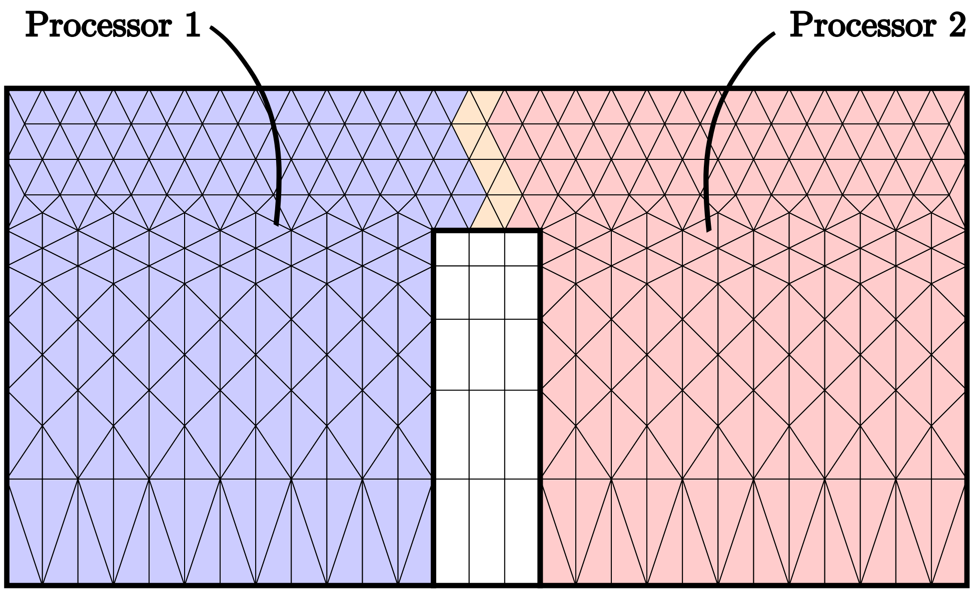
Native FSI solver: Spatial coupling



Automatic domain decomposition

Imperial College
London

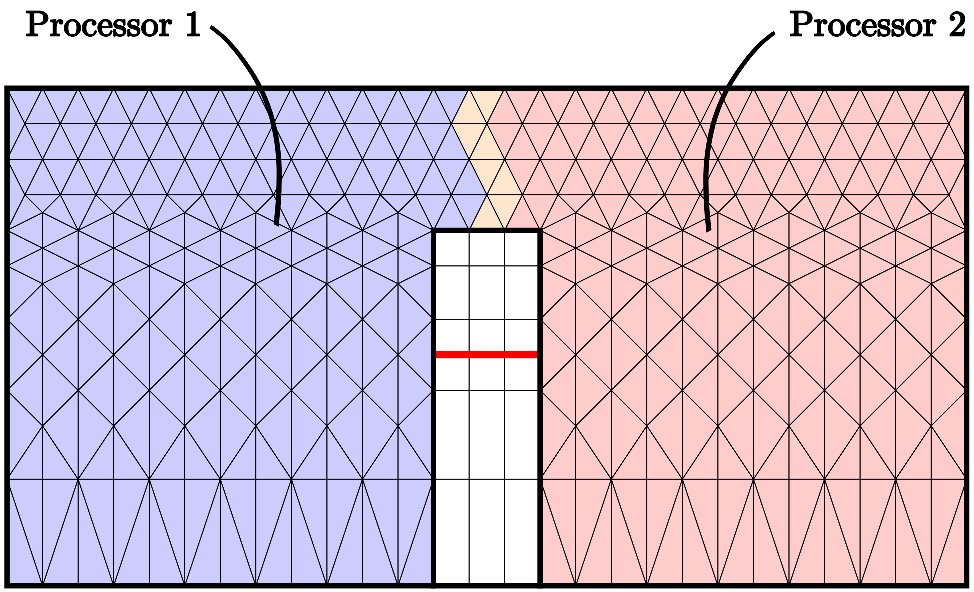
Native FSI solver: Spatial coupling



Automatic domain decomposition

Imperial College London

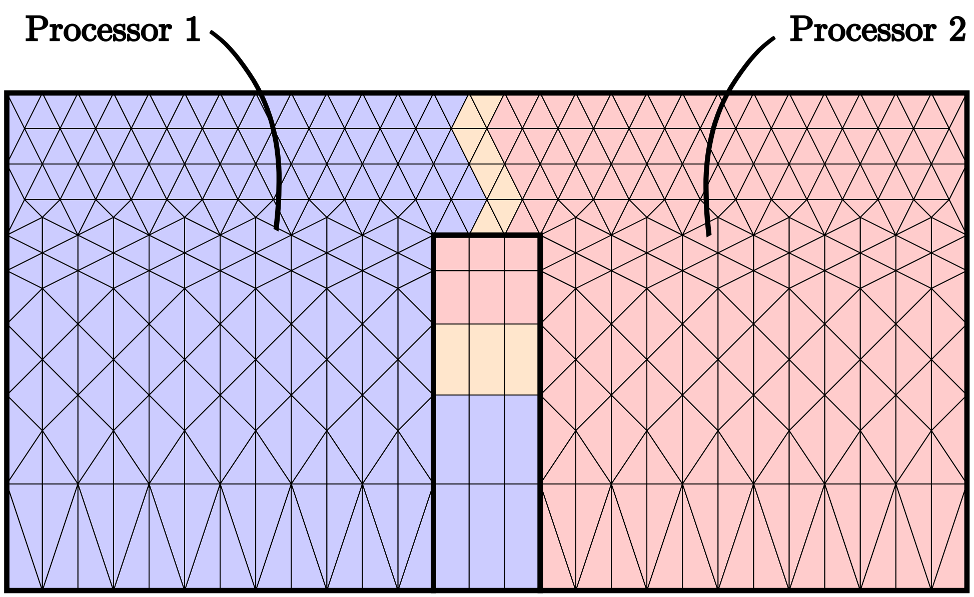
Native FSI solver: Spatial coupling



Automatic domain decomposition

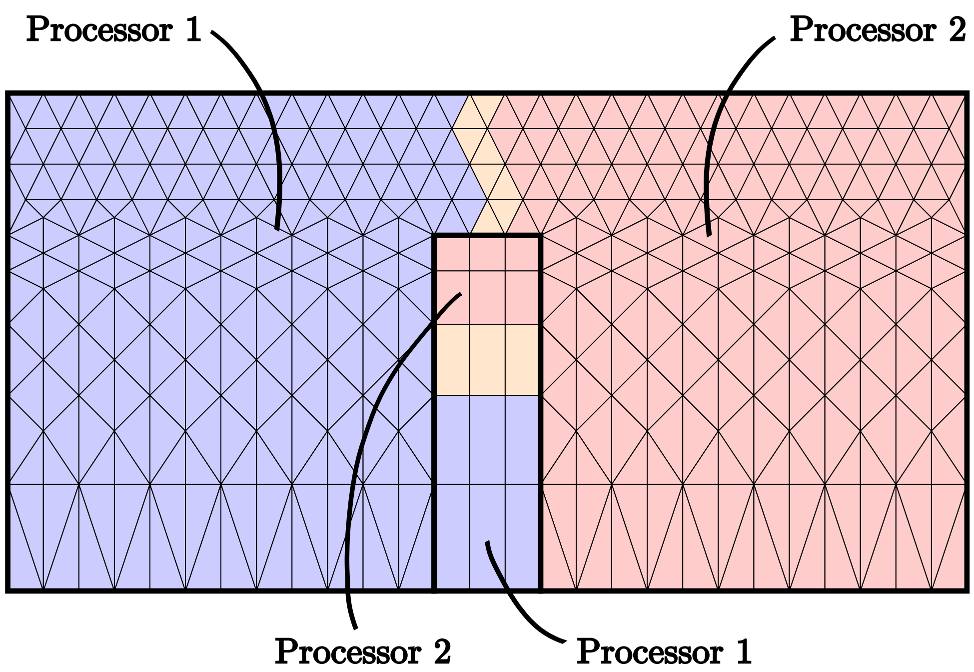
Imperial College London

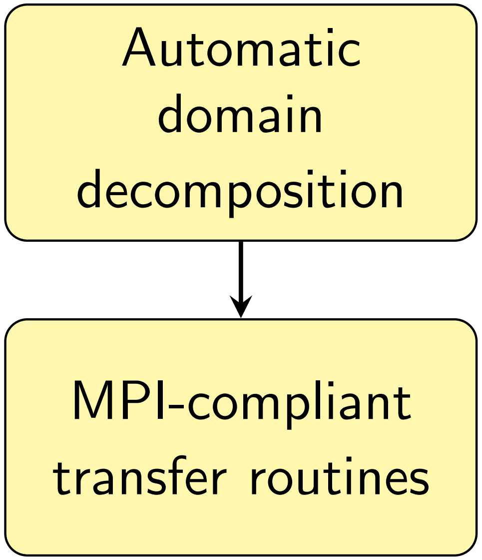
Native FSI solver: Spatial coupling



Automatic domain decomposition

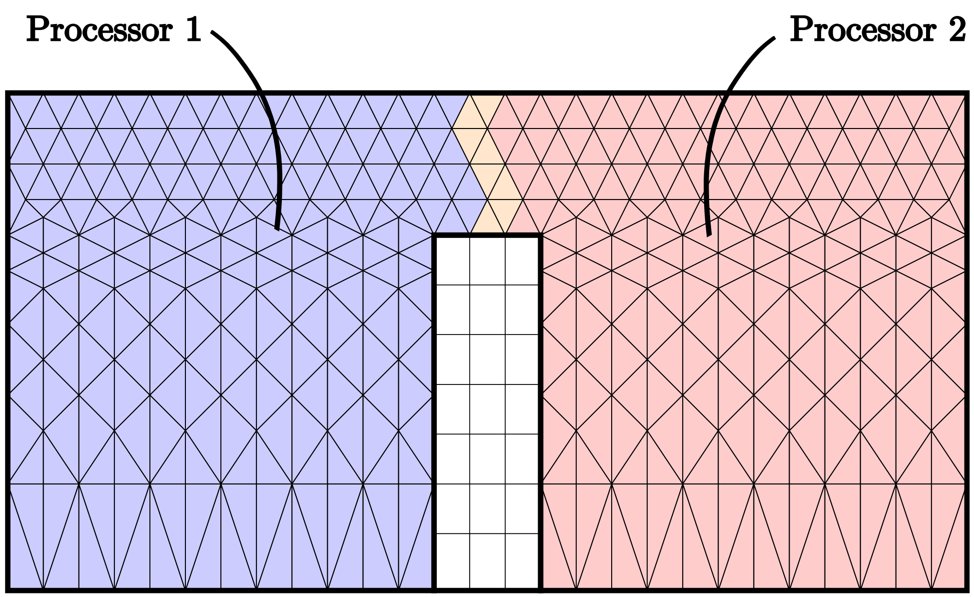
Native FSI solver: Spatial coupling





Imperial College London

Native FSI solver: Spatial coupling



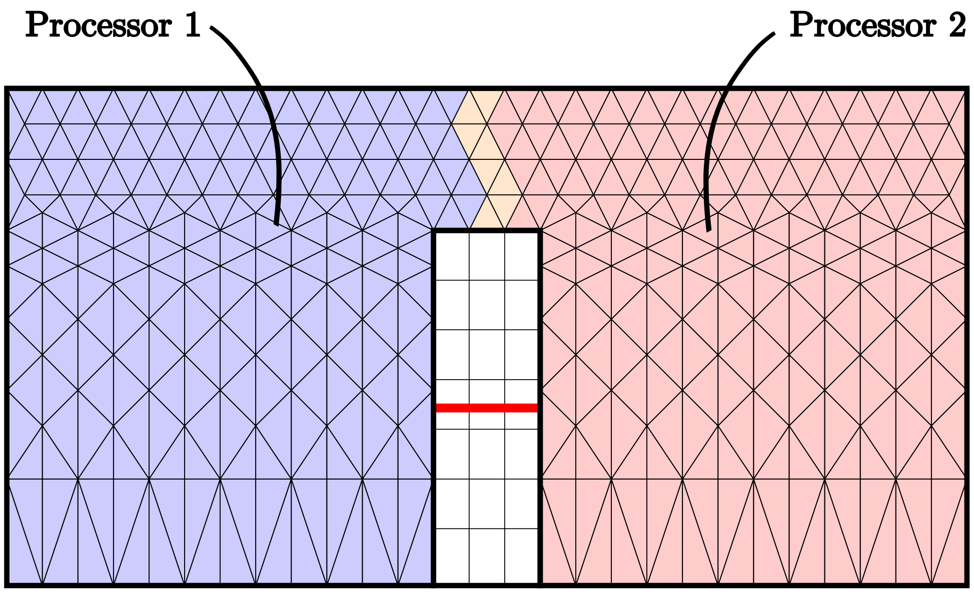
Automatic domain decomposition

↓

MPI-compliant transfer routines

Imperial College London

Native FSI solver: Spatial coupling

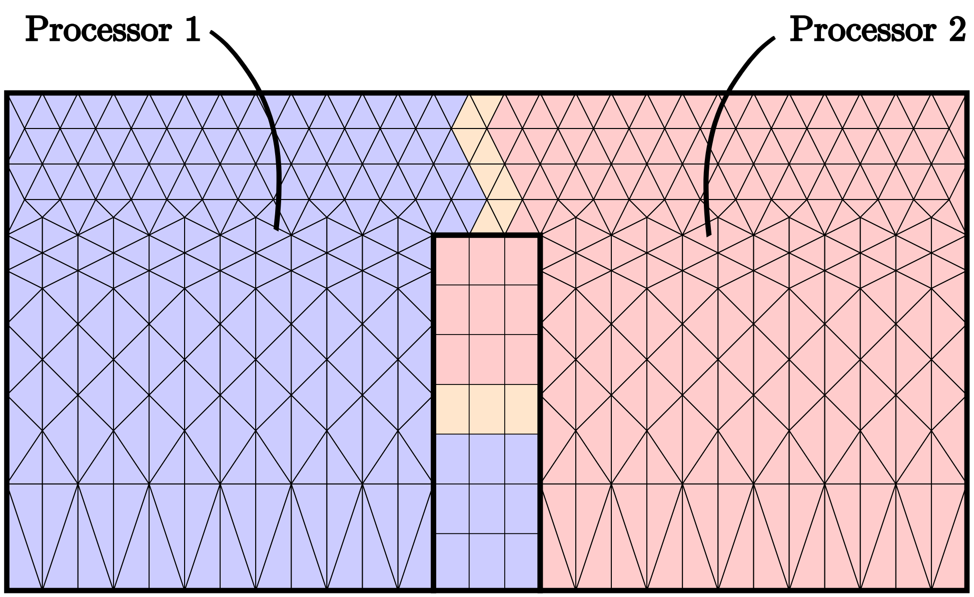


Automatic domain decomposition

↓

MPI-compliant transfer routines

Native FSI solver: Spatial coupling

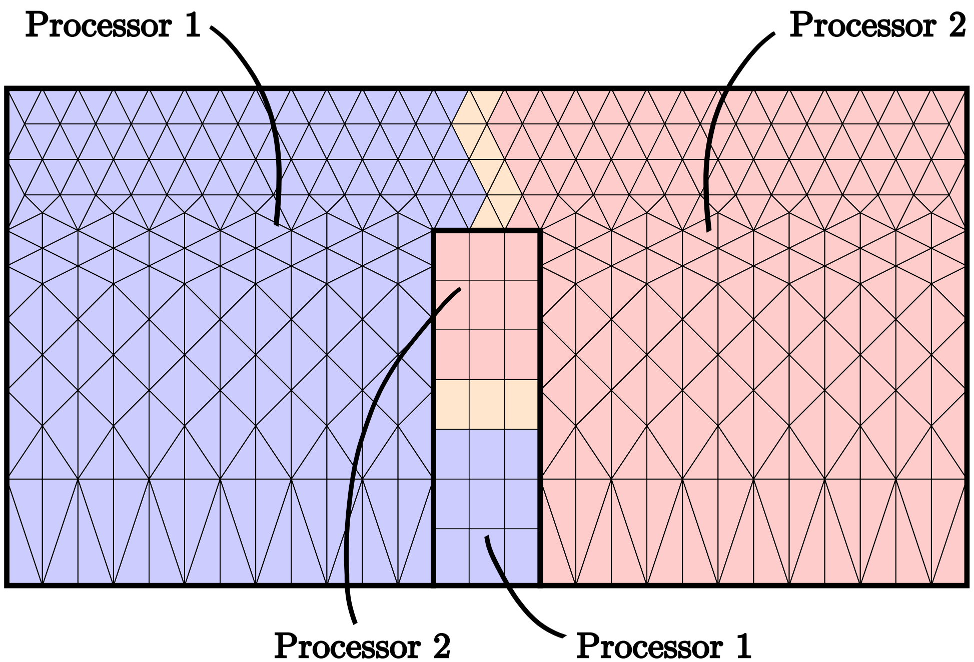


Automatic domain decomposition

↓

MPI-compliant transfer routines

Native FSI solver: Spatial coupling



Automatic domain decomposition

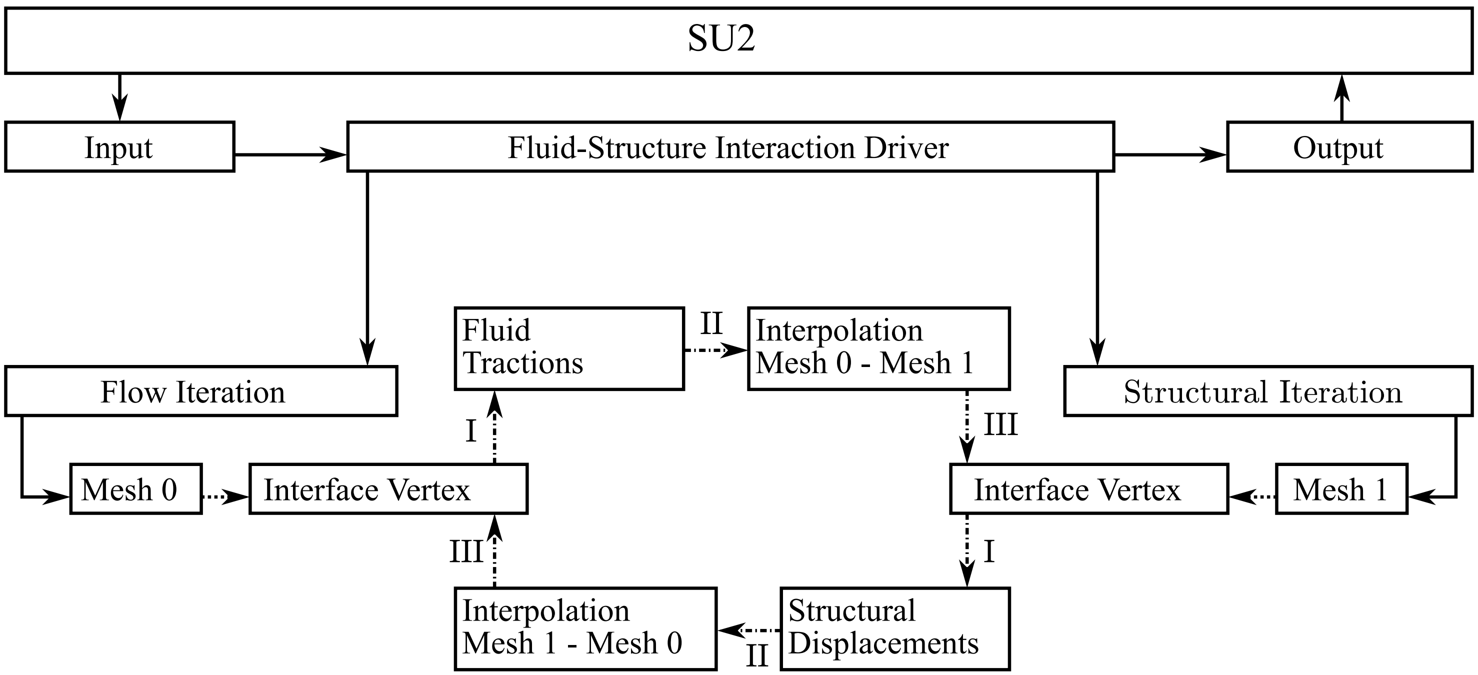
↓

MPI-compliant transfer routines

Interpolation

$$\mathbf{u}_f = \mathbf{H}\mathbf{u}_s$$
$$\boldsymbol{\lambda}_s = \mathbf{H}^T\boldsymbol{\lambda}_f$$

Native FSI solver: Spatial coupling structure

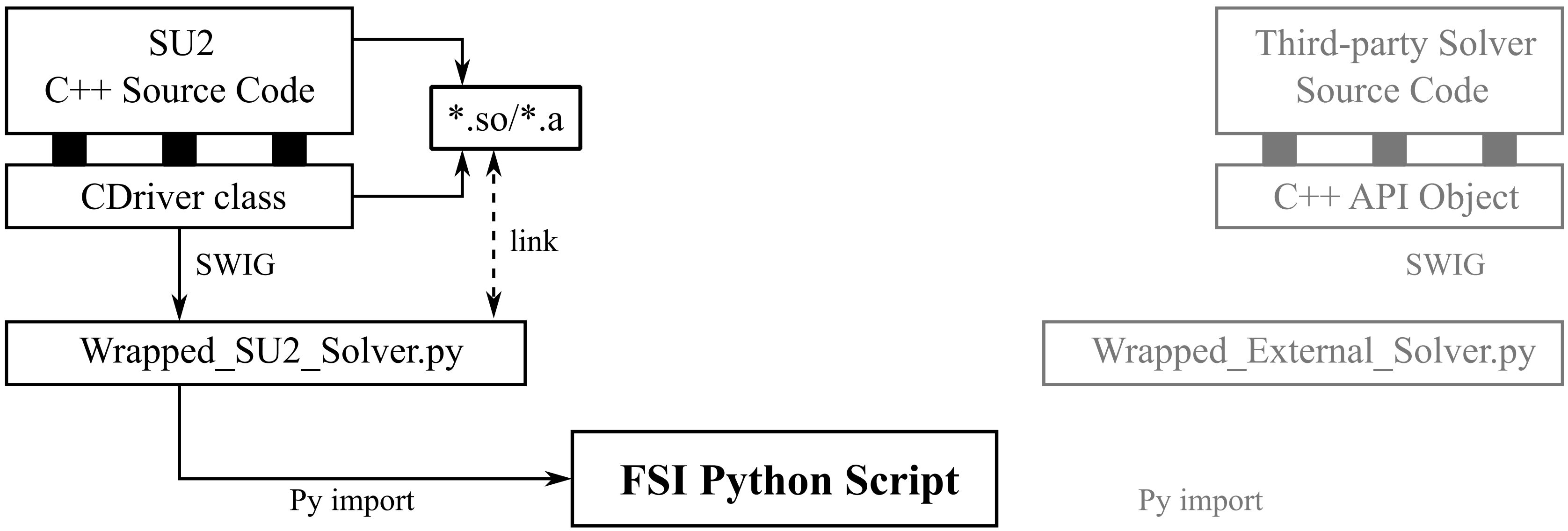


Python wrapper for external solvers

- Couple SU2 with a third-party solver: improved flexibility.
- **Python wrapper**: synchronizes the solvers within a single environment.
- Python wrapper calls all C++ functions in the new CDriver class.
- Additional compilation using SWIG: API $\rightarrow$ importable python object.
- Communication between solver performed through memory.
- Examples of coupling in this work:
  - Coupling with an in-house spring analogy solver.
  - Coupling with a third-party structural solver: TACS (Kennedy & Martin, 2014).

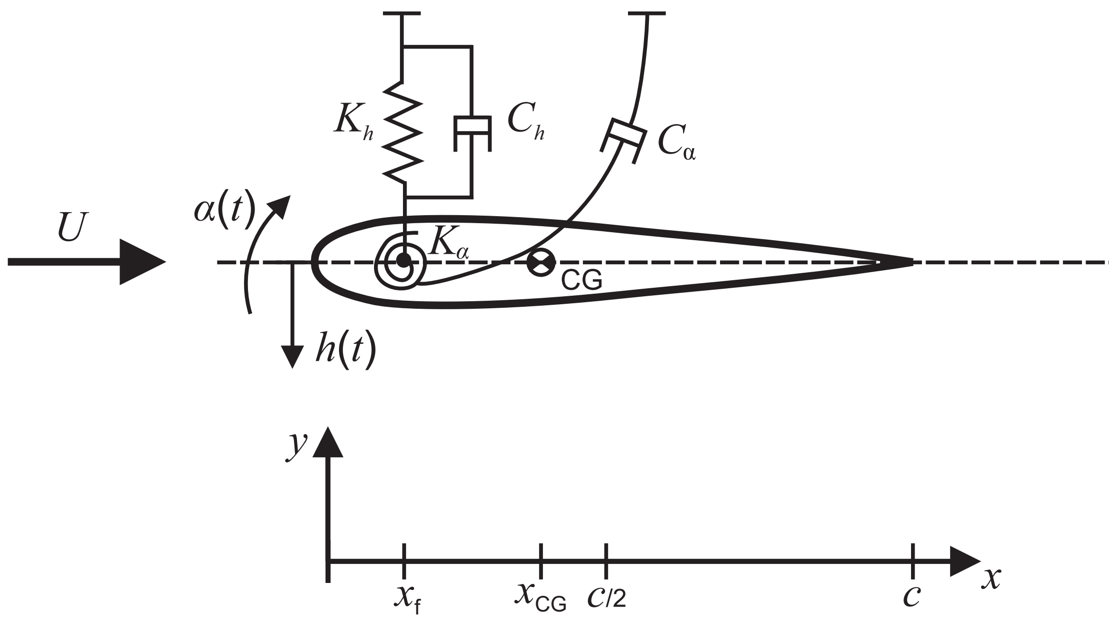
Imperial College London

Python wrapper: code structure



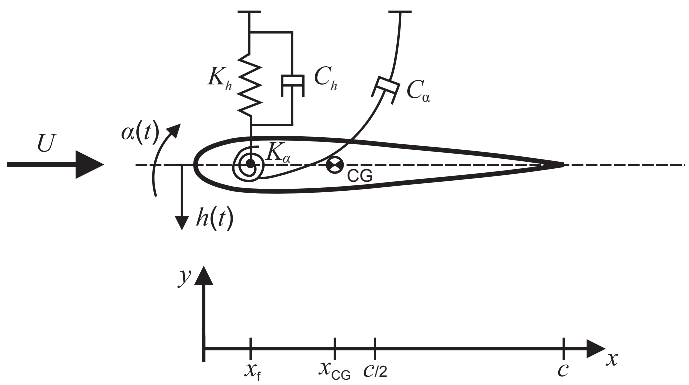
Imperial College
London

FSI solver for rigid body applications



$$
\begin{aligned}
m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\
S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha \alpha &= M
\end{aligned}
$$

FSI solver for rigid body applications

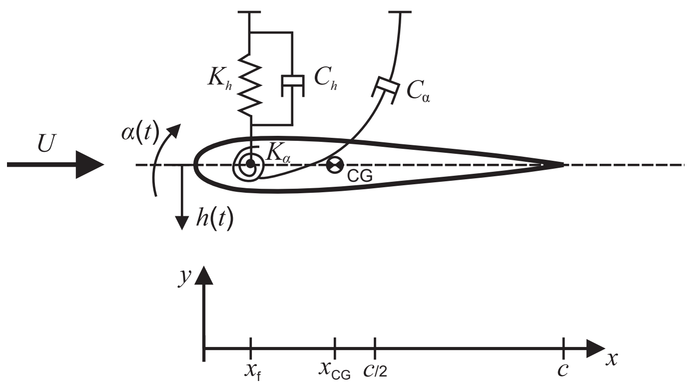


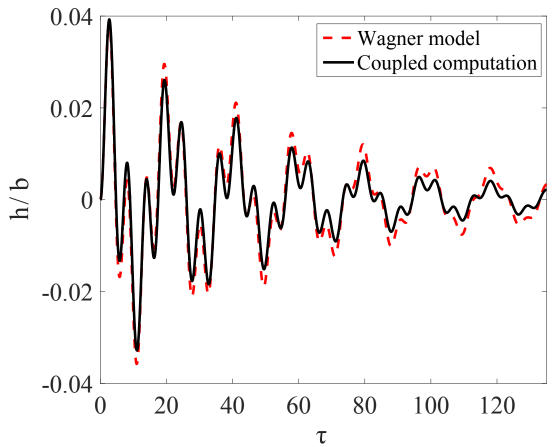
$$
\begin{aligned}
m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\
S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M
\end{aligned}
$$

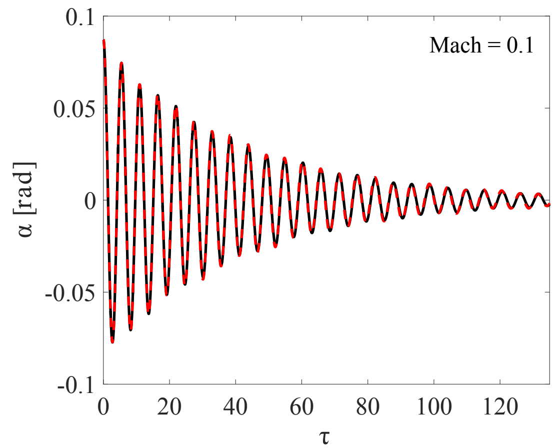
NACA 0012 using the Python wrapper

- Two-dimensional aeroelastic problem
- $k-\omega$ SST turbulence model
- Sub- and post-critical conditions
- Re $= 4 \cdot 10^6$ for $c = 1$ m
- Initial pitch angle 5°
- $\omega_\alpha = 45$ rad/s
- Strongly-coupled scheme

FSI solver for rigid body applications



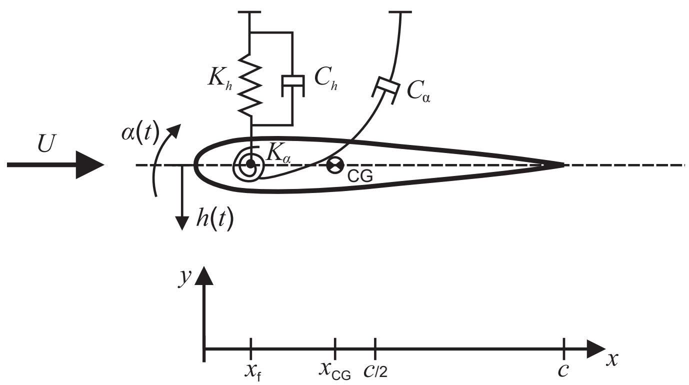




$$
\begin{aligned}
m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\
S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M
\end{aligned}
$$

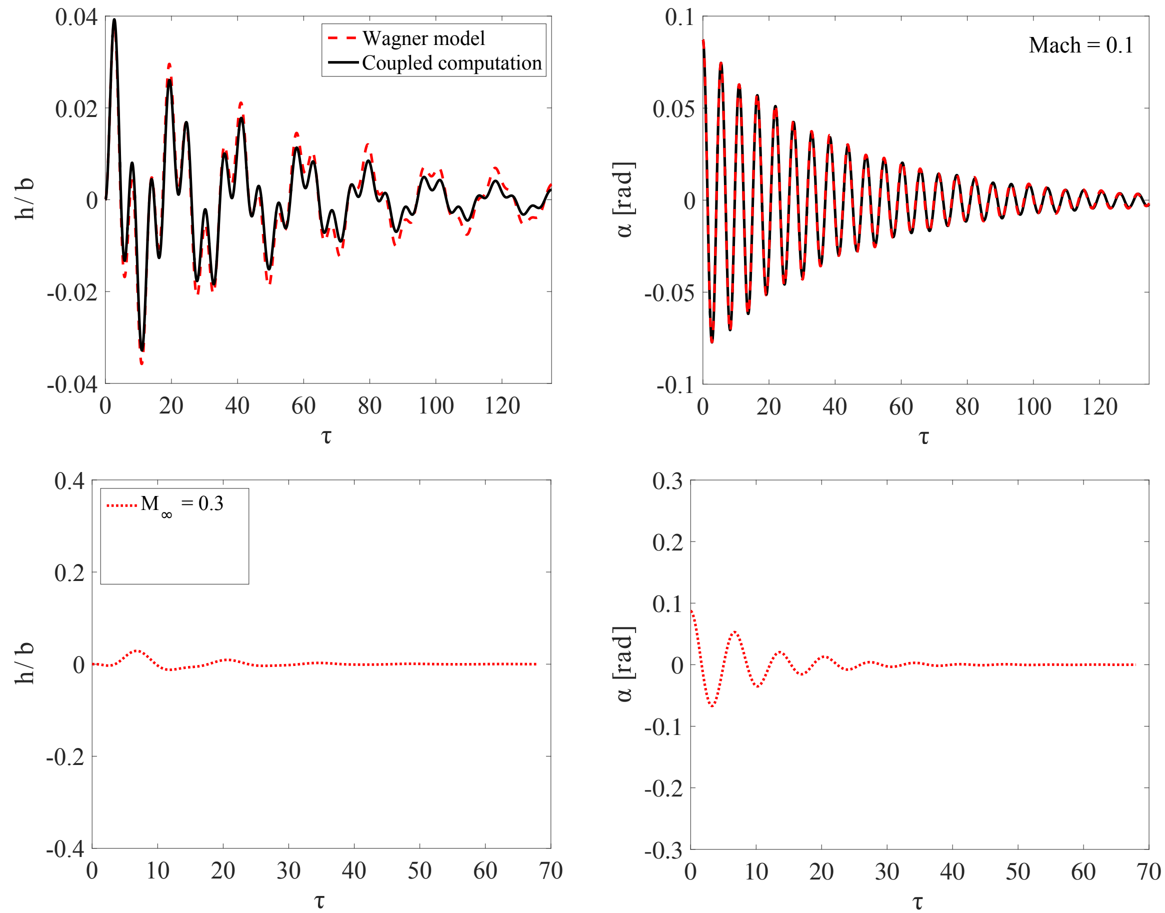
- NACA 0012 (Python wrapper)

FSI solver for rigid body applications

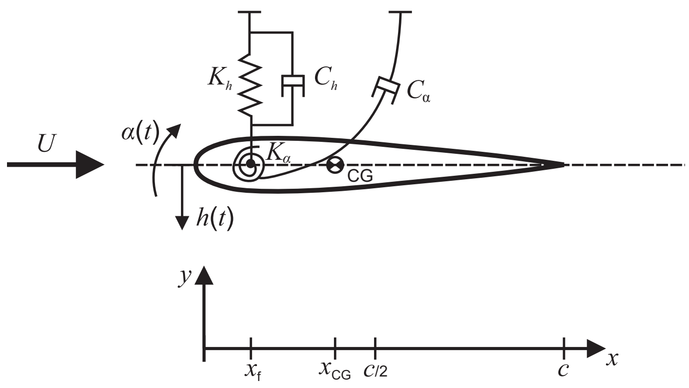


$$\begin{aligned} m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\ S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M \end{aligned}$$

- NACA 0012 (Python wrapper)

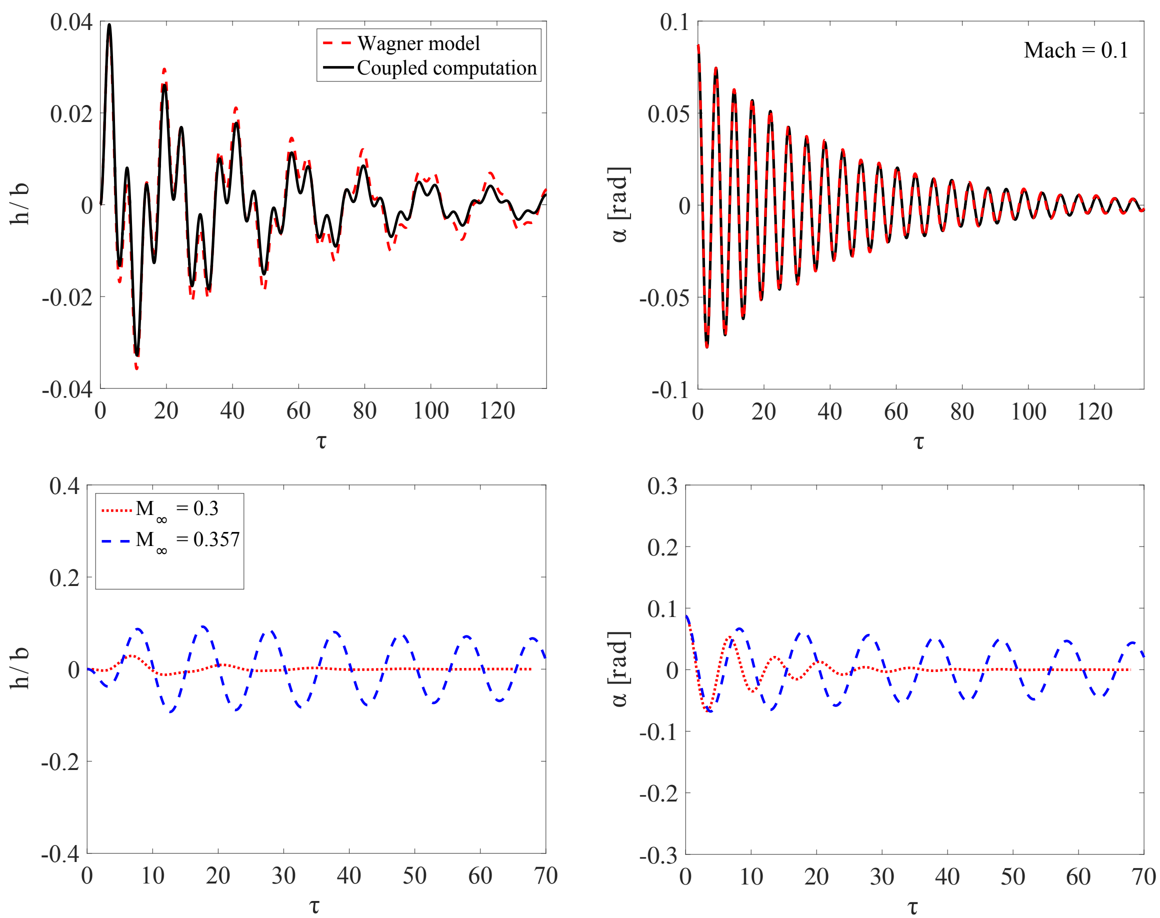


FSI solver for rigid body applications

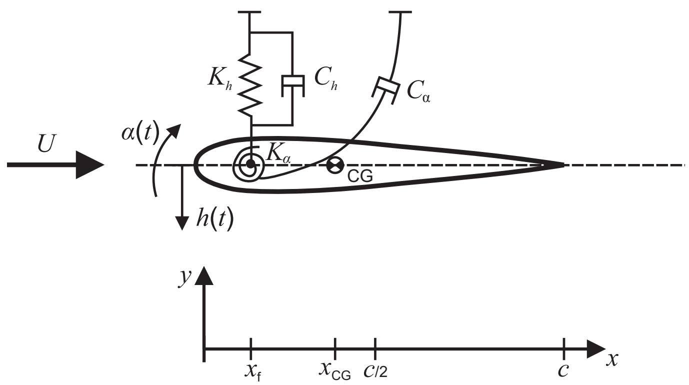


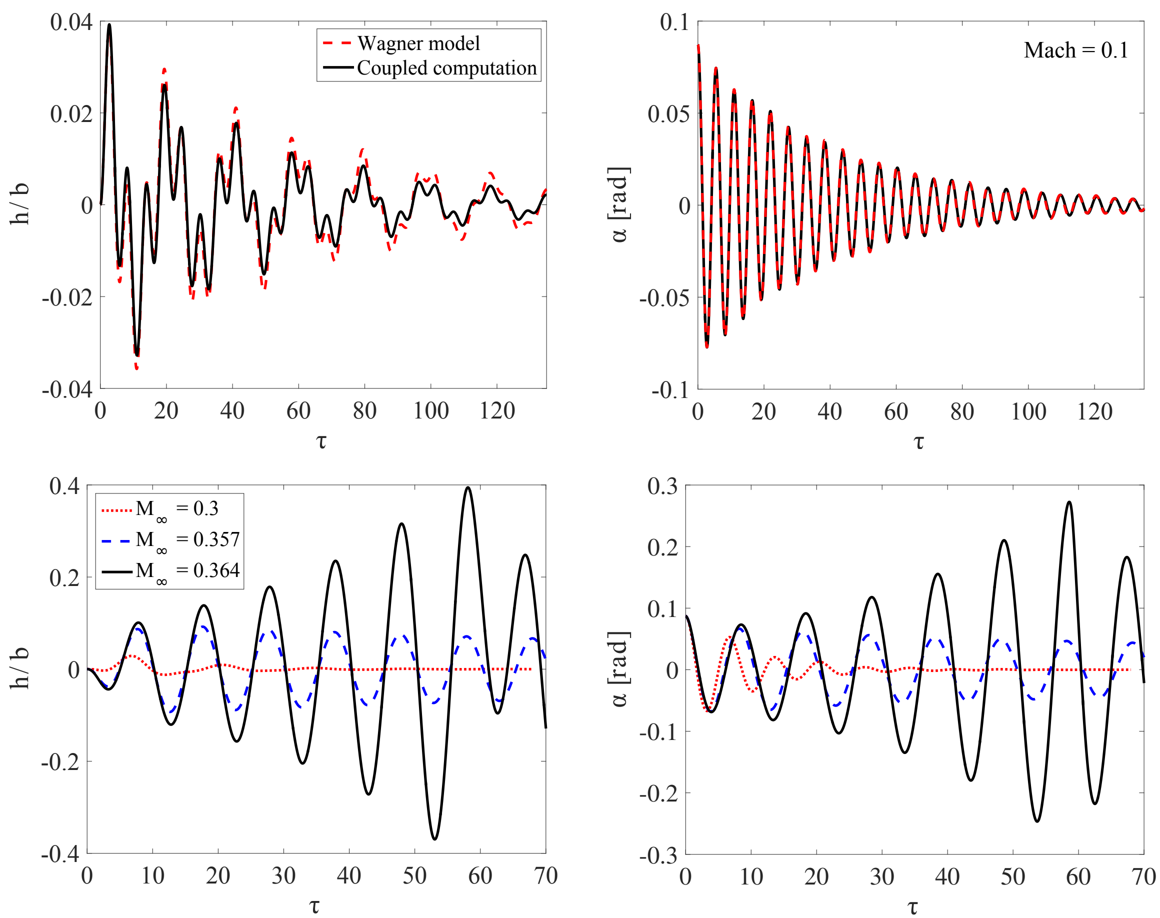
$$
\begin{aligned}
m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\
S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha \alpha &= M
\end{aligned}
$$

- NACA 0012 (Python wrapper)



FSI solver for rigid body applications

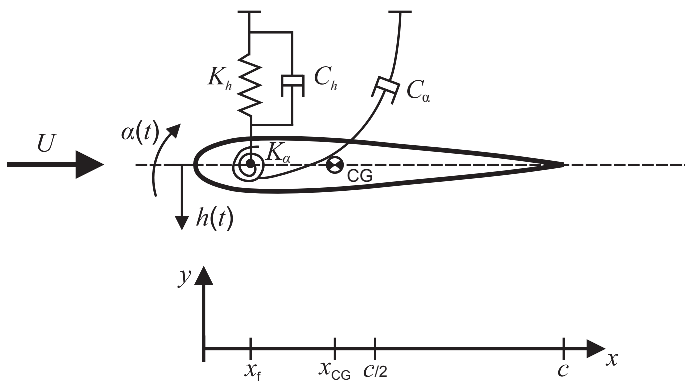


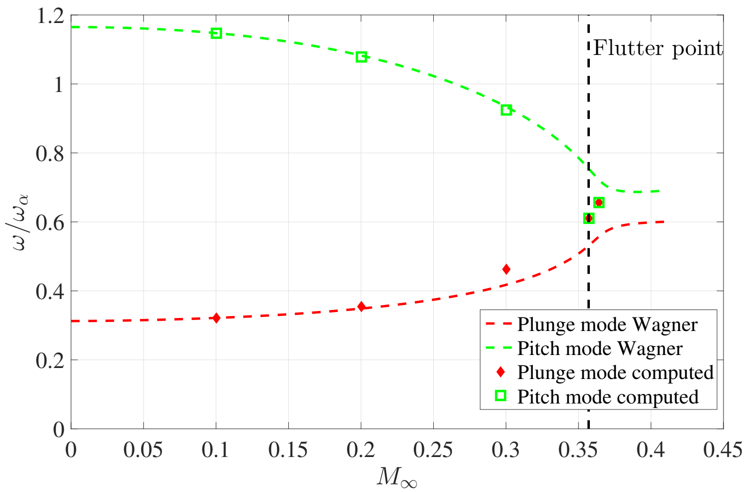


$$
\begin{aligned}
m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\
S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M
\end{aligned}
$$

- NACA 0012 (Python wrapper)

FSI solver for rigid body applications



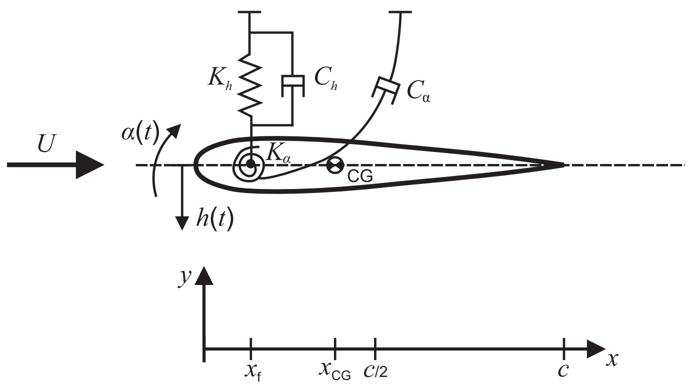


$$\begin{aligned} m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\ S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M \end{aligned}$$

- NACA 0012 (Python wrapper)

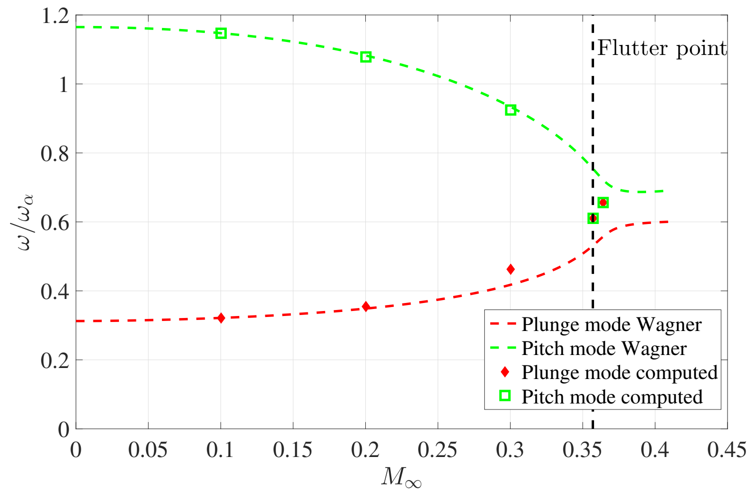
Imperial College
London

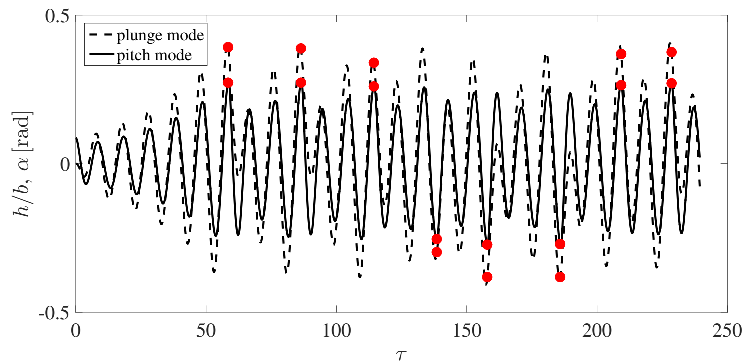
FSI solver for rigid body applications



$$
\begin{aligned}
m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\
S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M
\end{aligned}
$$

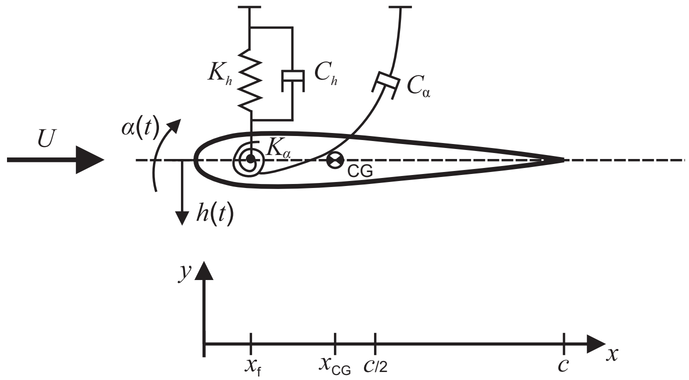
- NACA 0012 (Python wrapper)





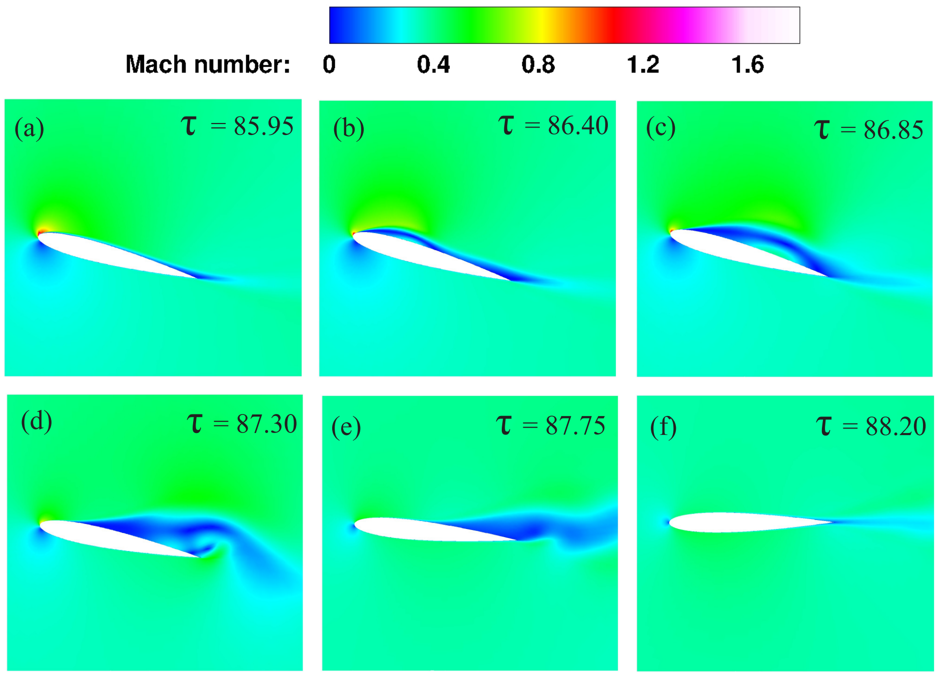
Imperial College London

FSI solver for rigid body applications

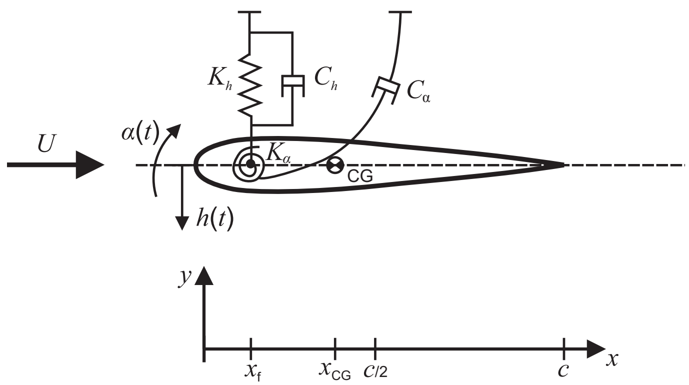


$$\begin{aligned} m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\ S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M \end{aligned}$$

- NACA 0012 (Python wrapper)



FSI solver for rigid body applications



$$\begin{aligned} m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\ S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M \end{aligned}$$

✓ NACA 0012 (Python wrapper)

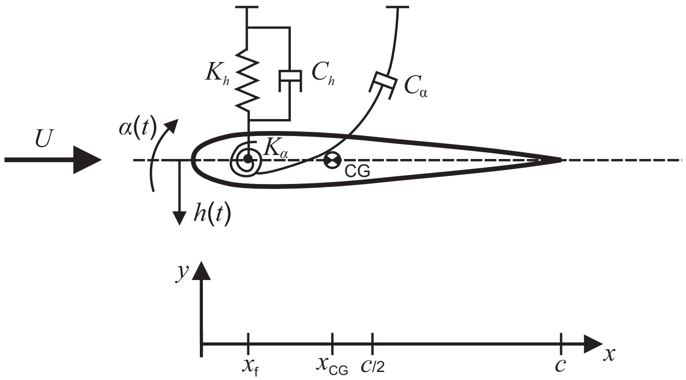
Isogai Wing Section using the Python wrapper

- Two-dimensional aeroelastic problem
- Swept back wing: elastic axis $x_f = -b$
- Transonic regime: $M_\infty = 0.7$ - 0.9
- $\omega_\alpha = 100$ rad/s
- Strongly-coupled scheme
- Speed index:

$$V^* = \frac{U_\infty}{b\omega_\alpha\sqrt{\mu}}$$

Imperial College
London

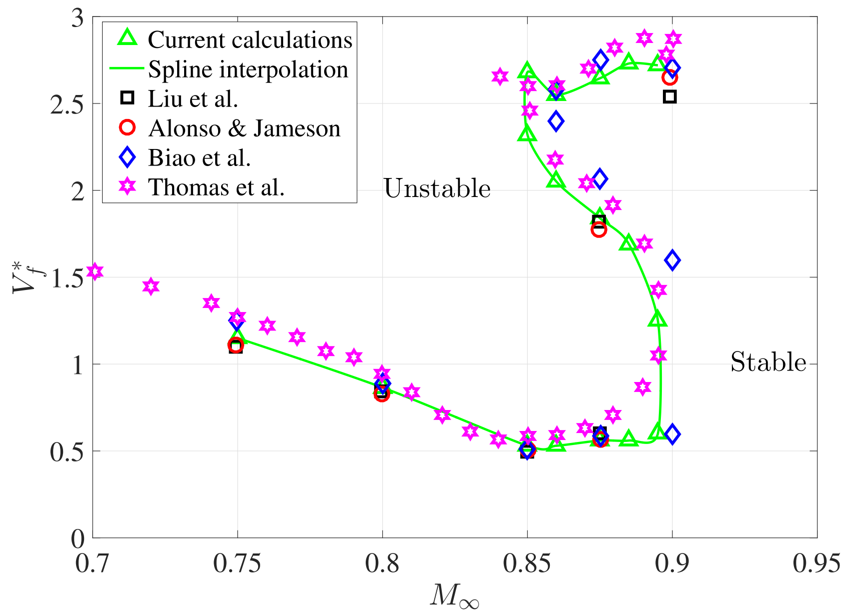
FSI solver for rigid body applications



$$
\begin{aligned}
m\ddot{h} + S\ddot{\alpha} + C_h\dot{h} + K_h h &= -L \\
S\ddot{h} + I_f\ddot{\alpha} + C_\alpha\dot{\alpha} + K_\alpha\alpha &= M
\end{aligned}
$$

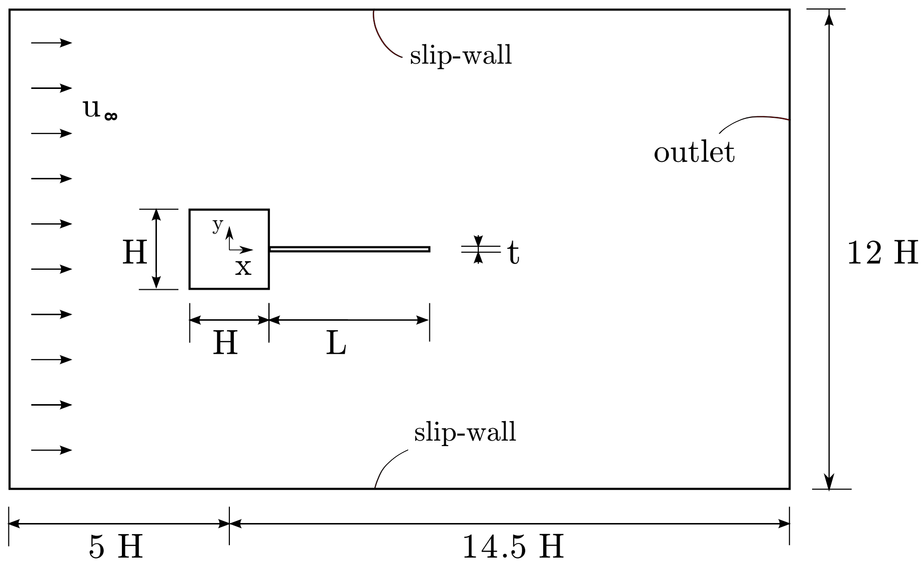
✓ NACA 0012 (Python wrapper)

- Isogai Wing Section (Python wrapper)



Imperial College London

Native FSI solver with deformable solids



Geometry:

$H = 1$ cm
$L = 4$ cm
$t = 0.06$ cm

Fluid properties

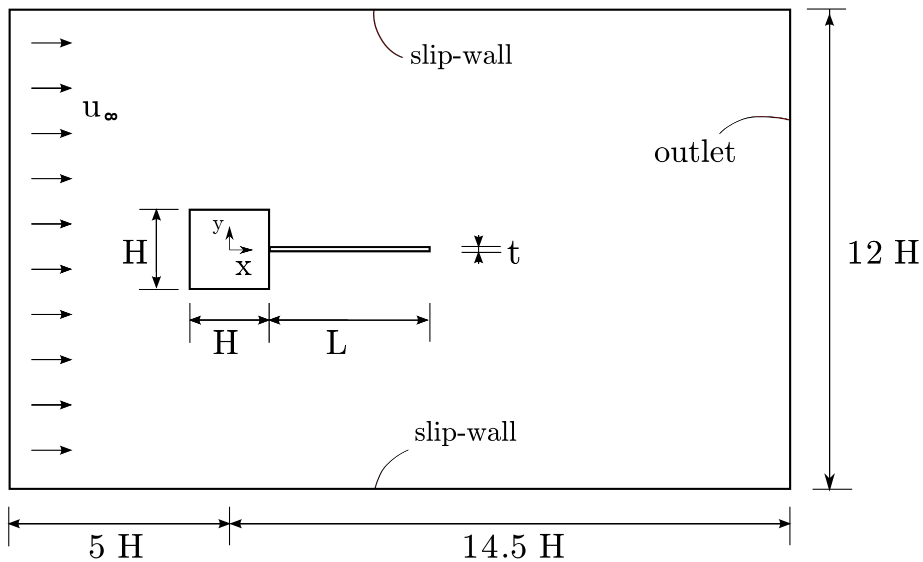
$u_\infty = 0.513$ m/s
$\rho_f = 1.18$ kg/m^3
$\mu_f = 1.82 \cdot 10^5$ kg/m·s
$Re = 332$

Structural properties

$E = 2.50 \cdot 10^5$ Pa
$\rho_s = 100$ kg/m^3
$\nu = 0.35$

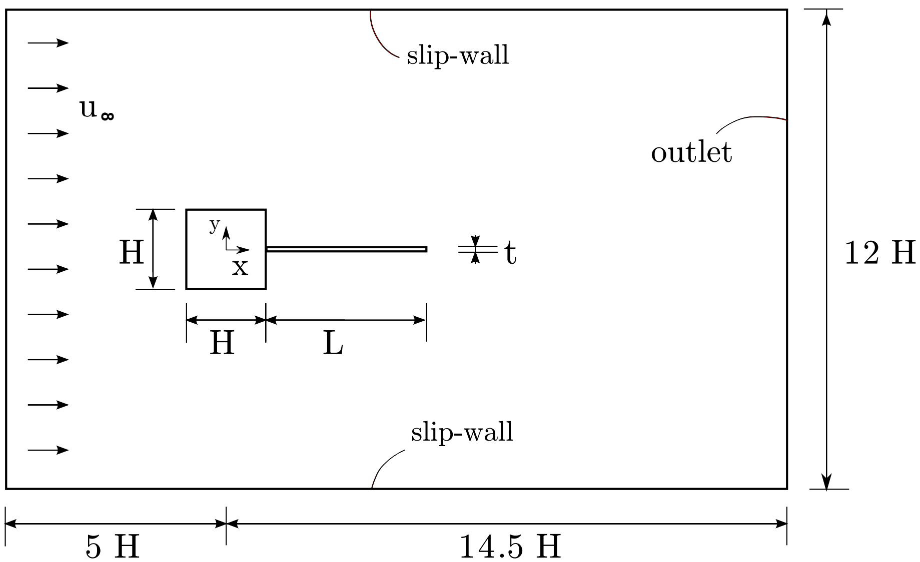
Imperial College London

Native FSI solver with deformable solids

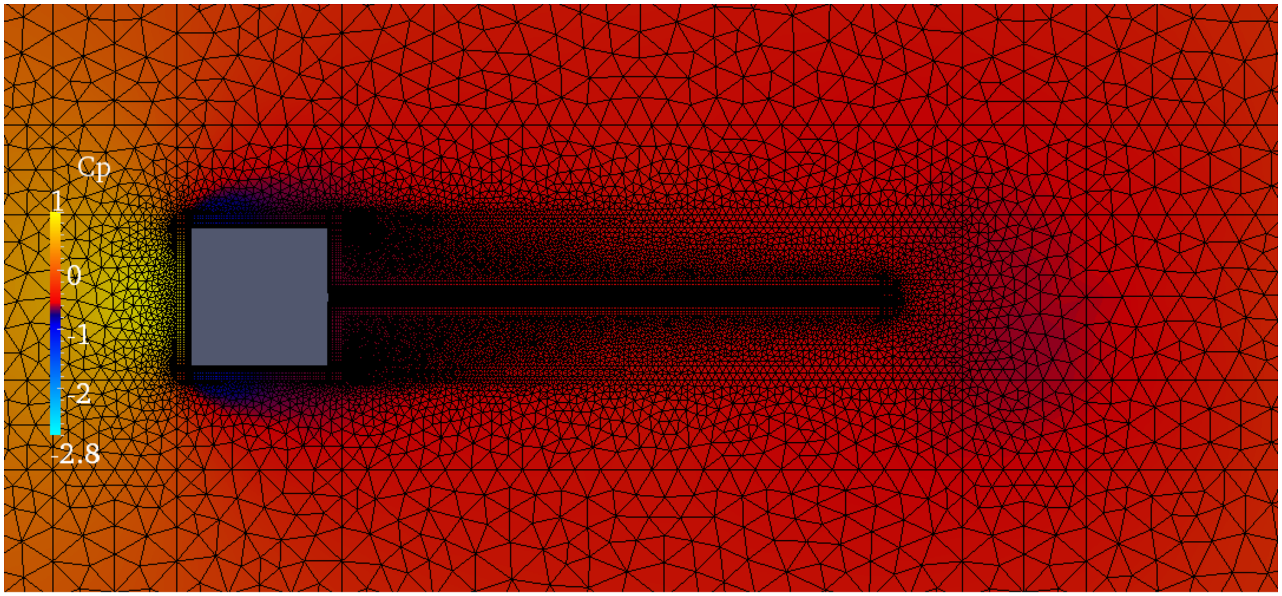


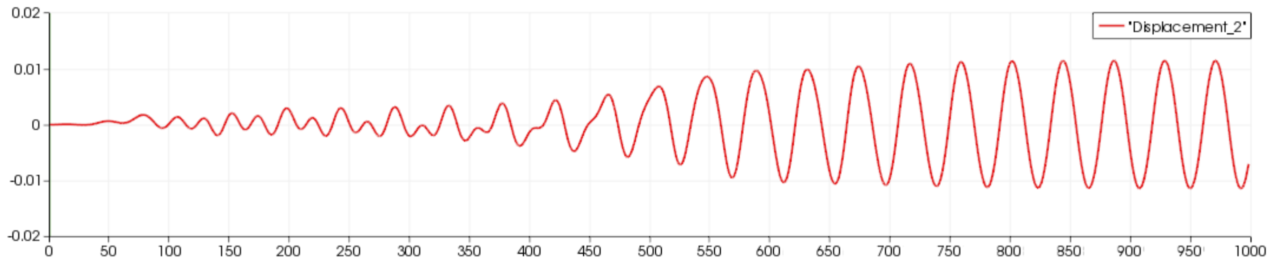
	$\bar{f}$ (Hz)	d_{max}(cm)
Wall and Ramm (1998)	3.08	1.31

Native FSI solver with deformable solids

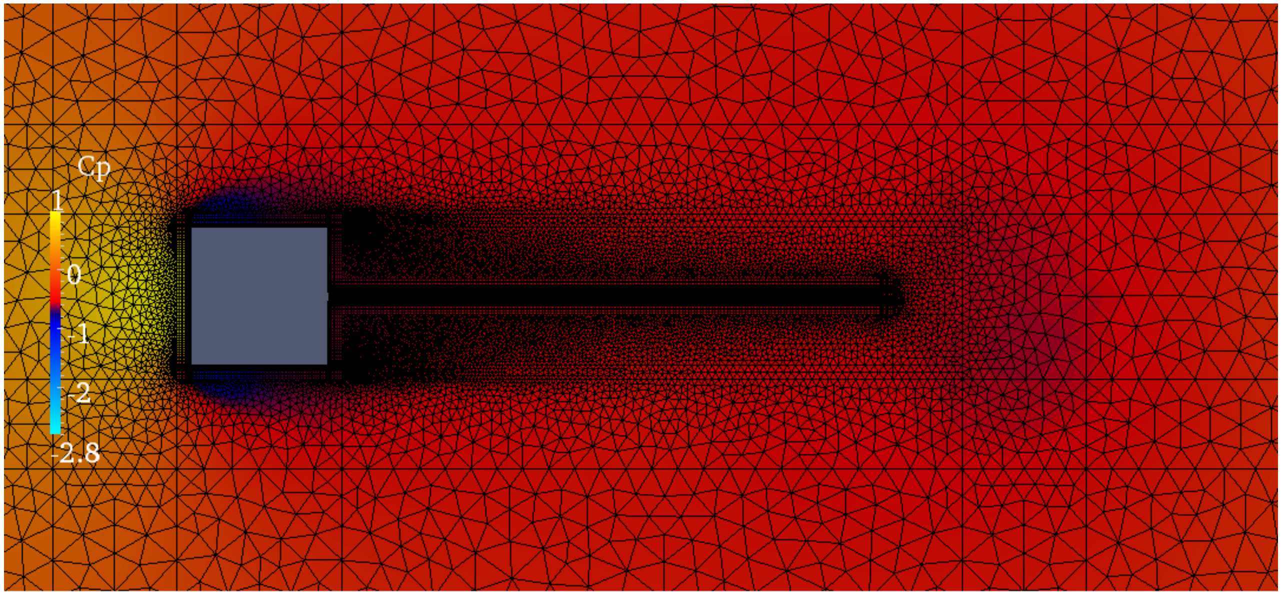


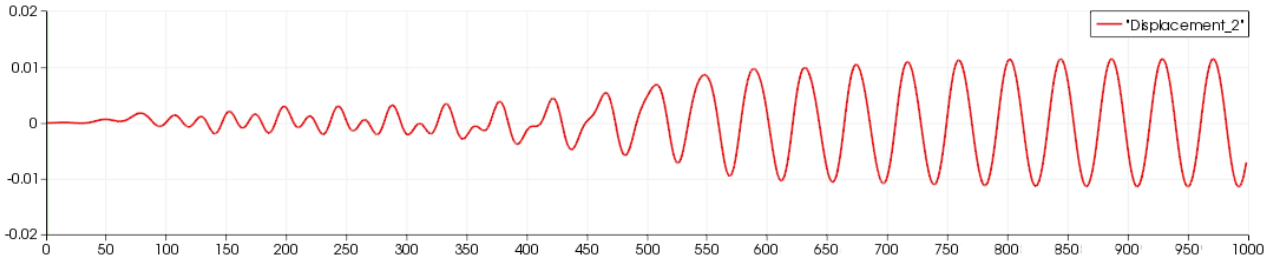
	$\bar{f}$ (Hz)	d_{max}(cm)
Wall and Ramm (1998)	3.08	1.31
Matthies and Steindorf (2003)	2.99	1.34
Dettmer and Peric (2006)	2.96	1.1
	3.31	1.4
Wood et al. (2008)	2.78	1.1
	3.13	1.2
Kassiotis et al. (2011)	3.17	1.0
Habchi et al. (2013)	3.25	1.02
Froehle and Persson (2014)	3.18	1.12


Cp
1
0
-1
-2
-2.8

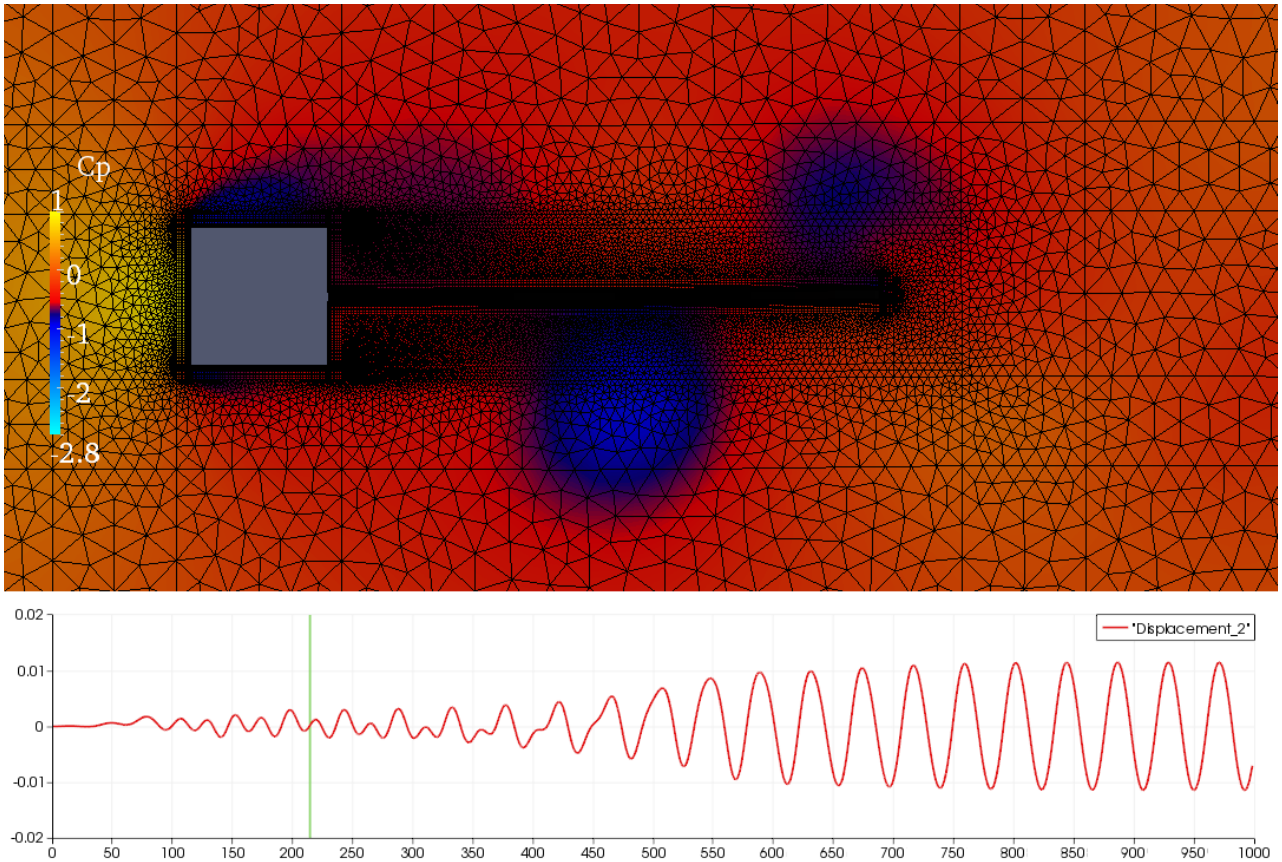

'Displacement_2'
0.02
0.01
0
-0.01
-0.02
0 50 100 150 200 250 300 350 400 450 500 550 600 650 700 750 800 850 900 950 1000

Imperial College
London



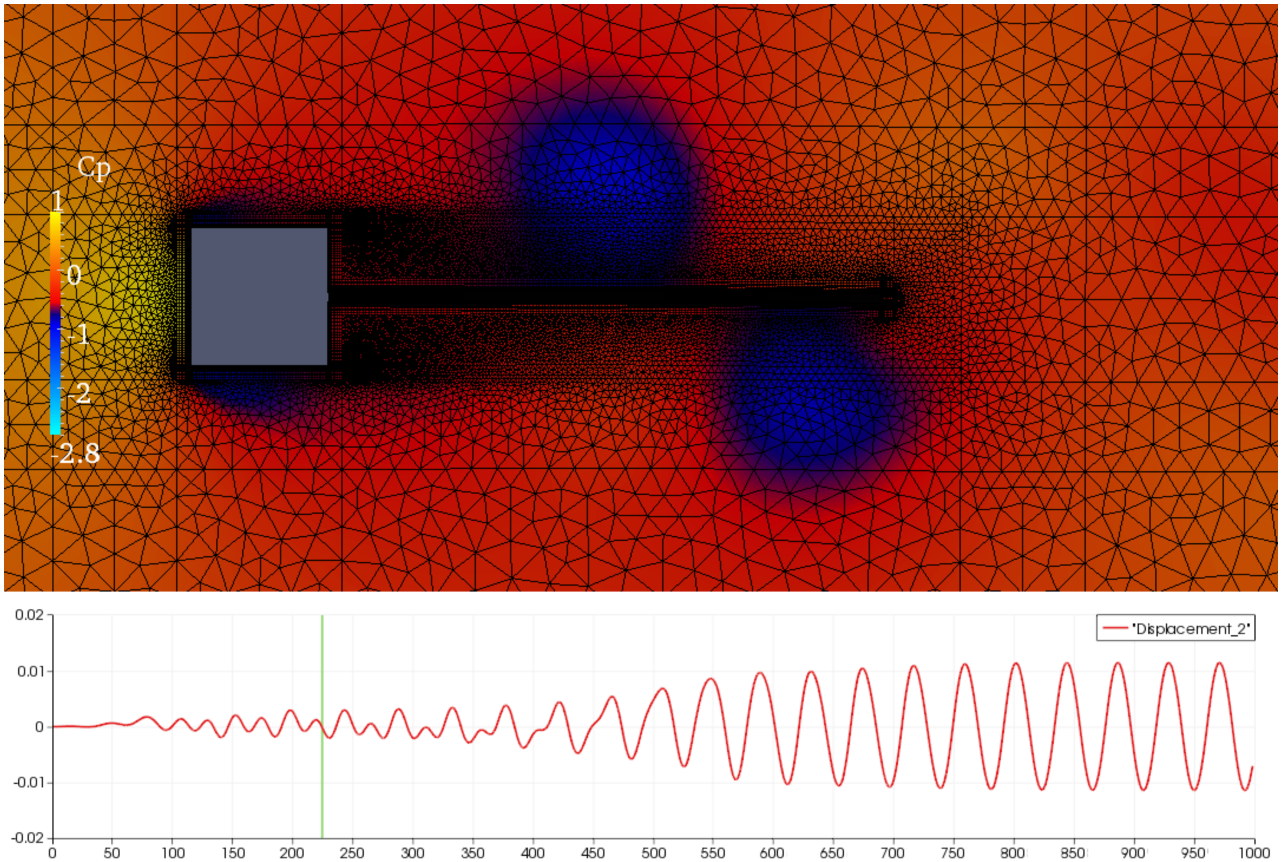


Imperial College
London



Flow-driven vibrations

Imperial College
London

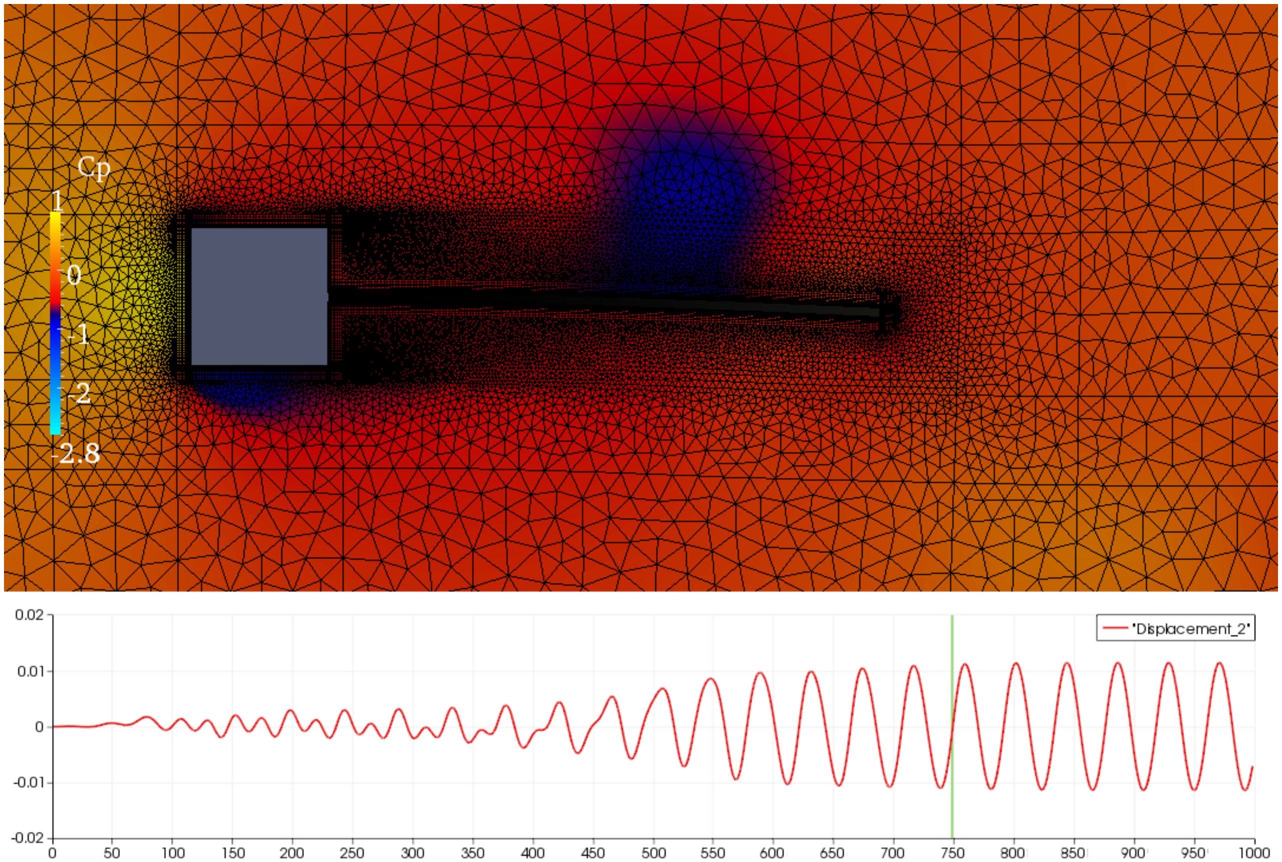


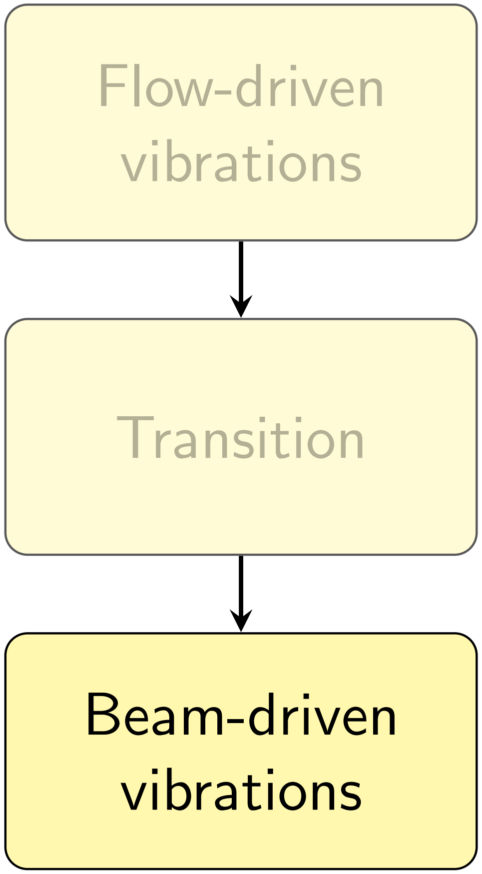
Flow-driven vibrations

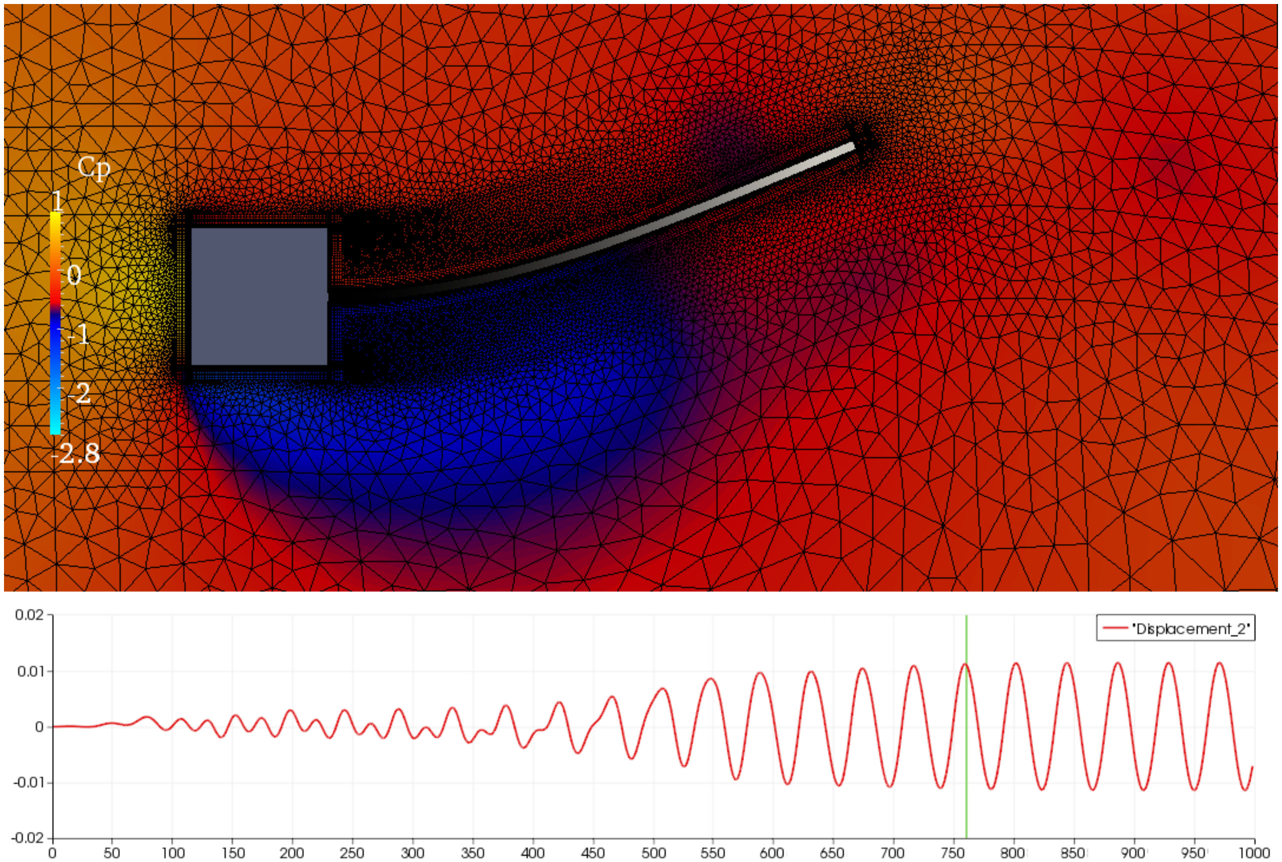
↓

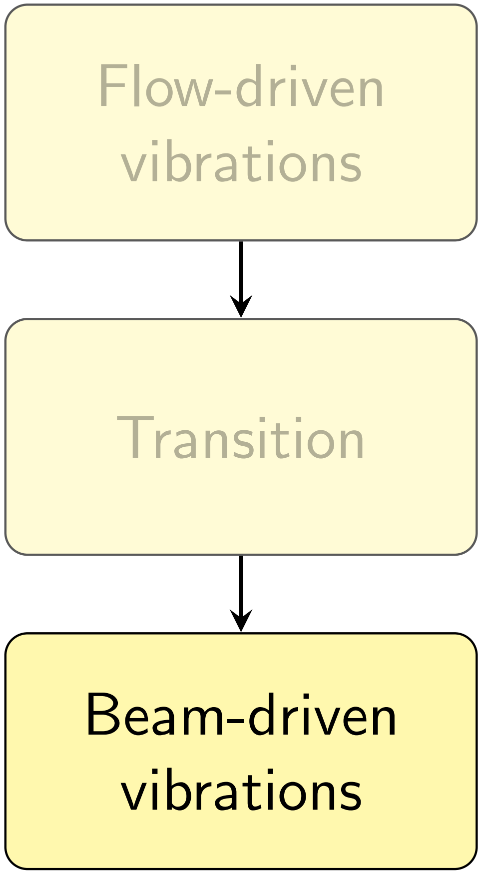
Transition

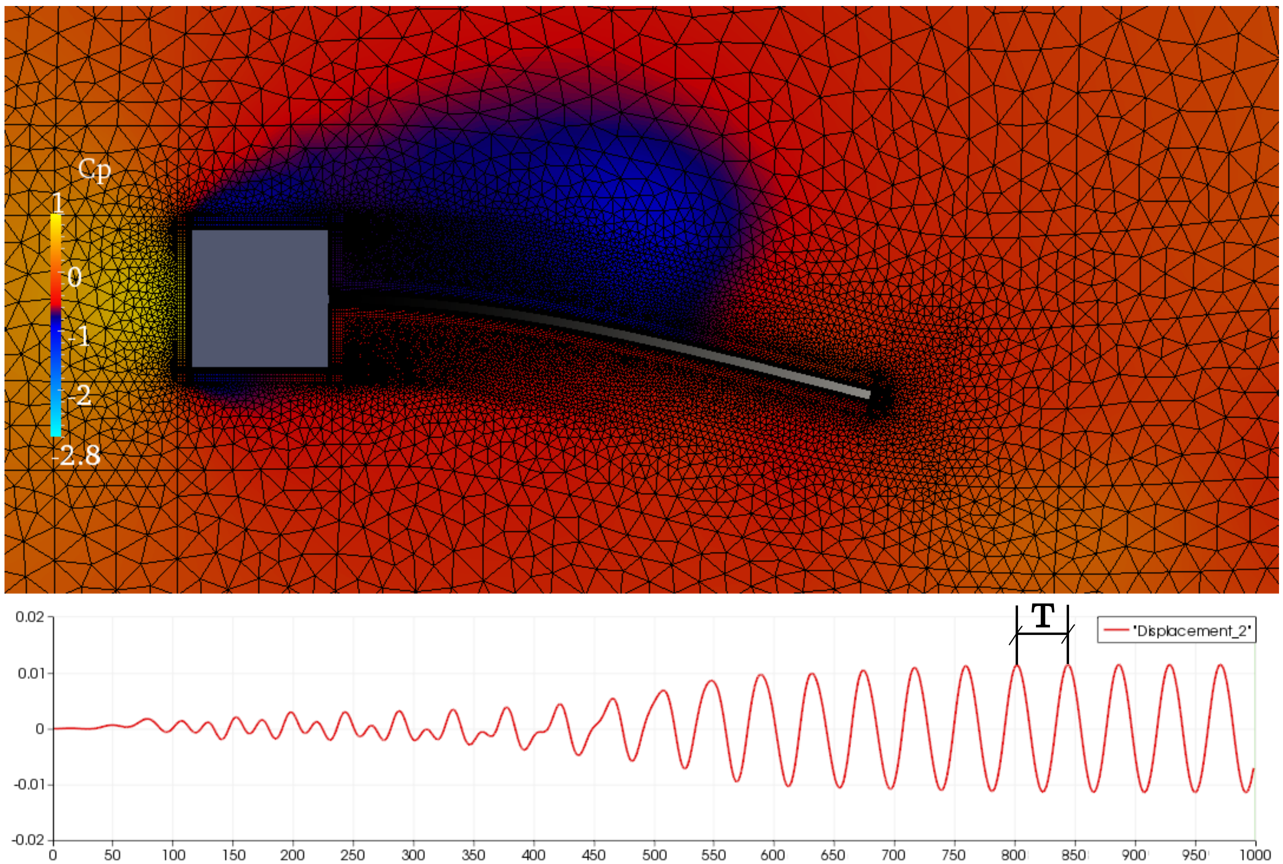
Imperial College
London






Cp
1
0
-1
-2
-2.8
'Displacement_2'
0.02
0.01
0
-0.01
-0.02
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000


Flow-driven vibrations
Transition
Beam-driven vibrations



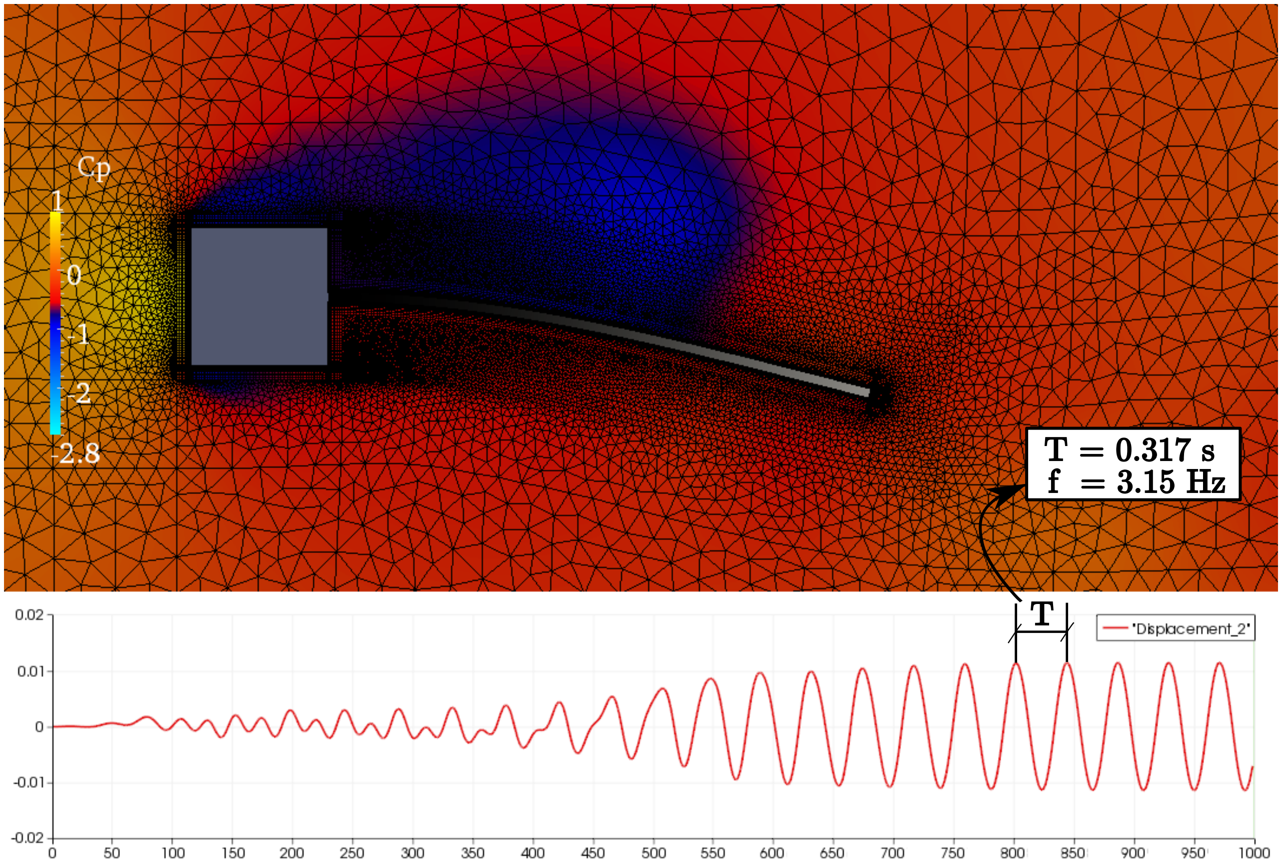
Flow-driven vibrations

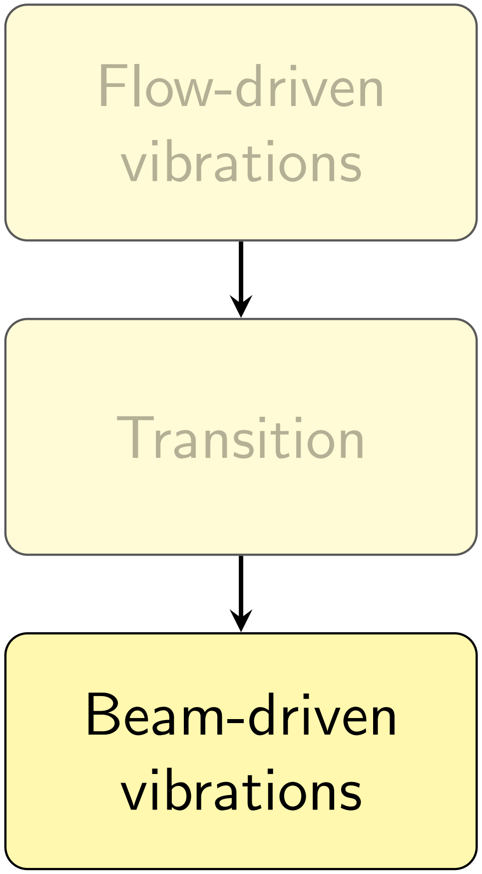
↓

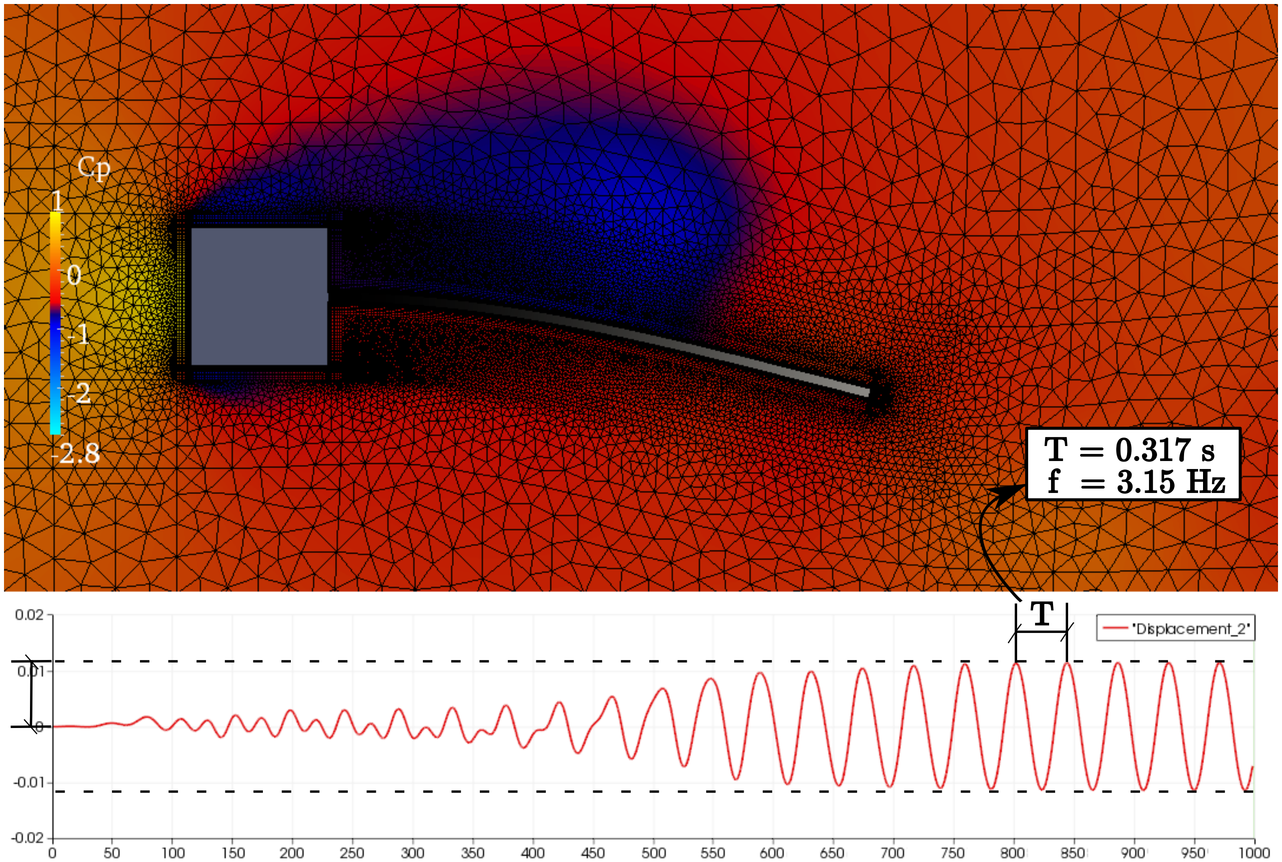
Transition

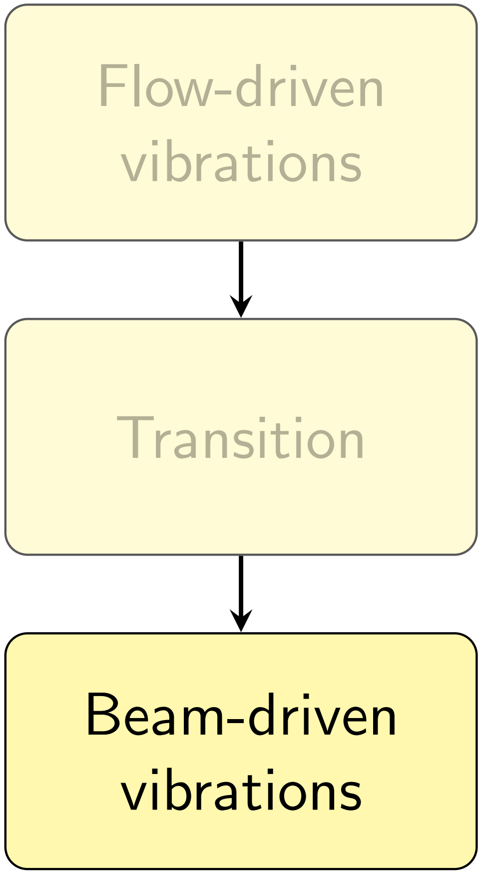
↓

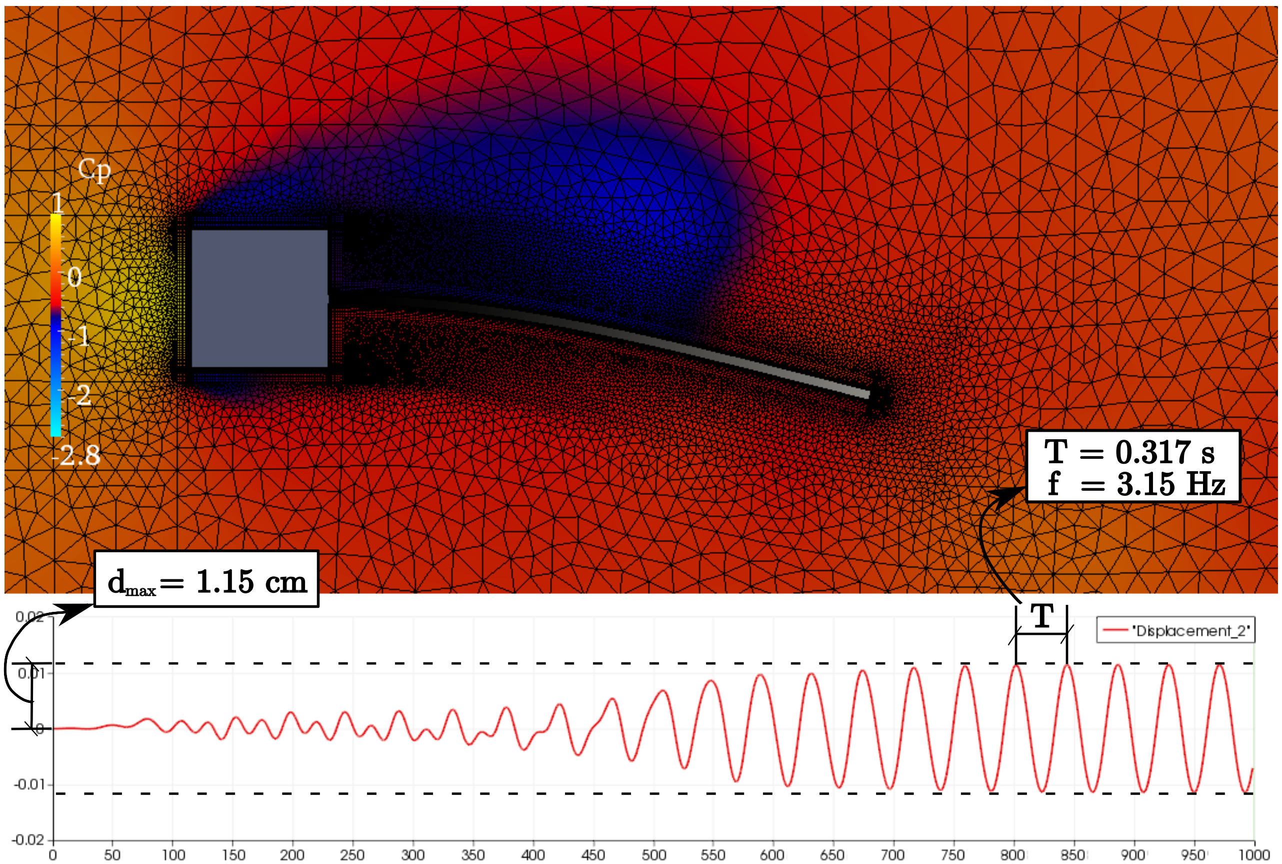
Beam-driven vibrations


Cp
1
0
-1
-2
-2.8
T = 0.317 s
f = 3.15 Hz
T
'Displacement_2'
0.02
0.01
0
-0.01
-0.02
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000


Flow-driven vibrations
Transition
Beam-driven vibrations


Cp
1
0
-1
-2
-2.8
T = 0.317 s
f = 3.15 Hz
T
'Displacement_2'
0.02
0.01
0
-0.01
-0.02
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
900
950
1000


Flow-driven vibrations
Transition
Beam-driven vibrations



Flow-driven vibrations

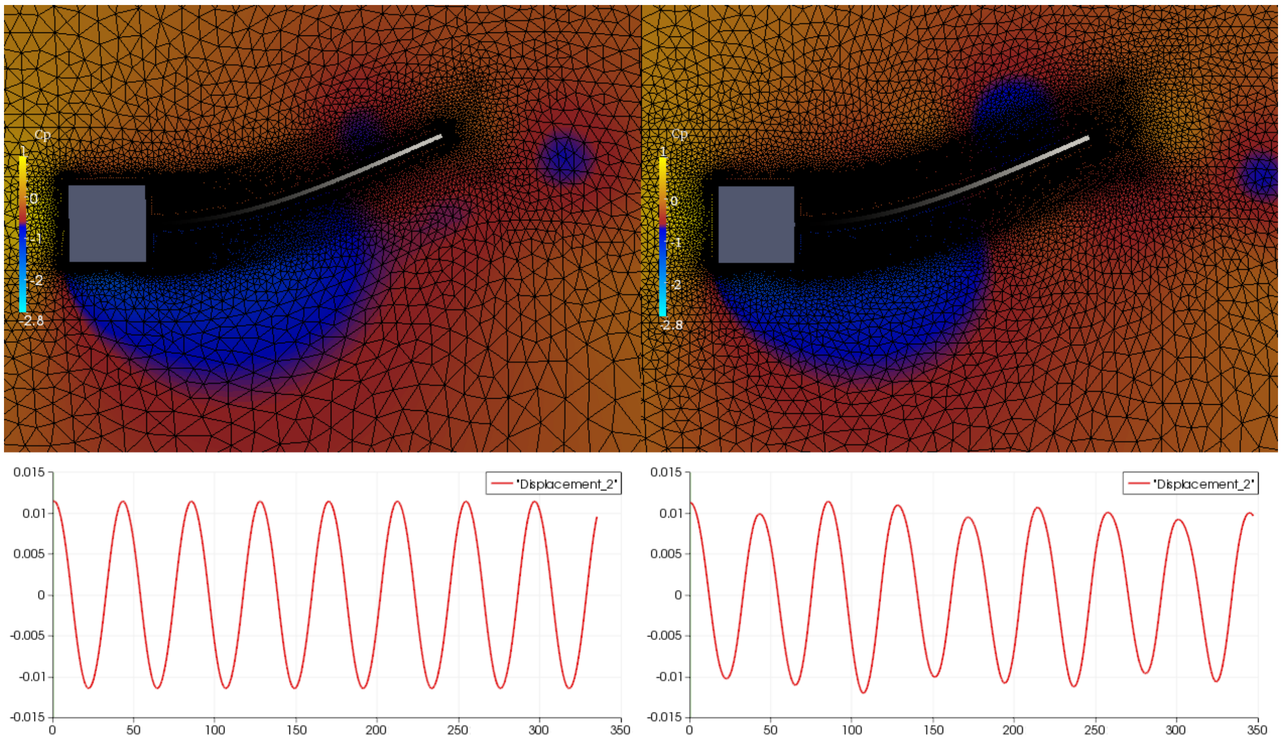
↓

Transition

↓

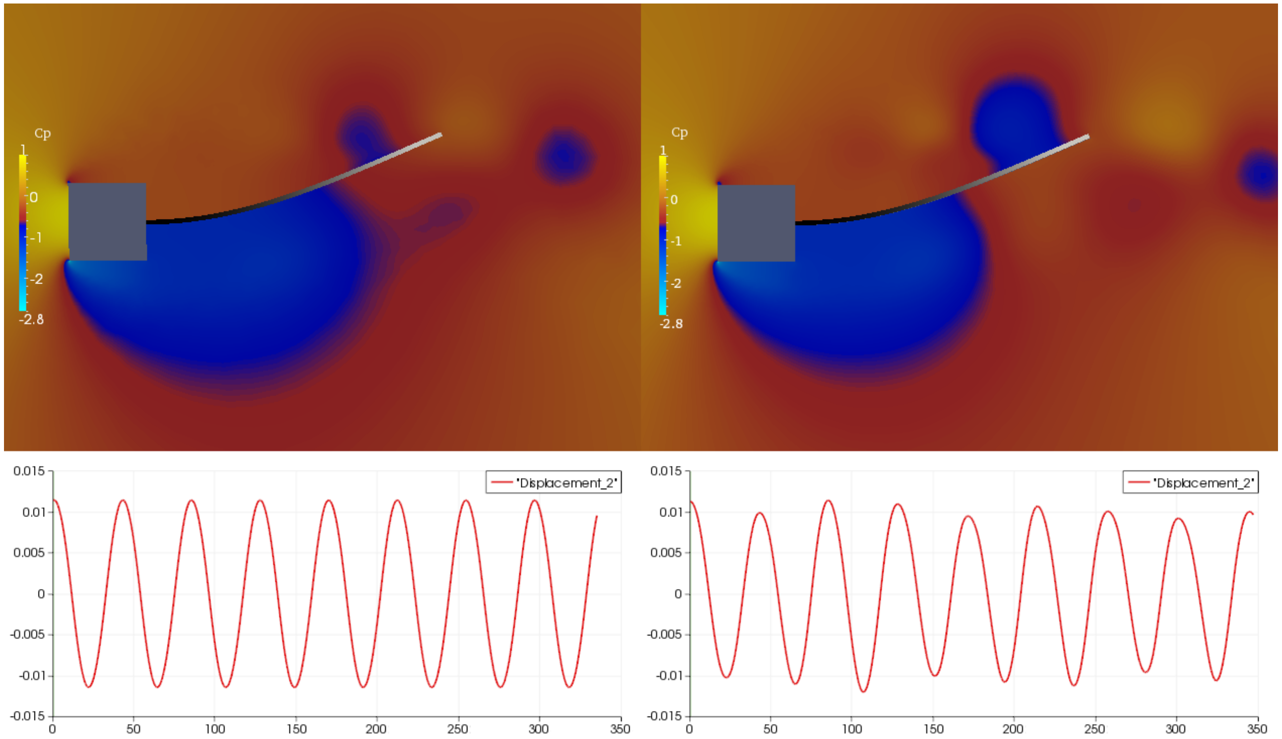
Beam-driven vibrations

Native FSI solver with deformable solids



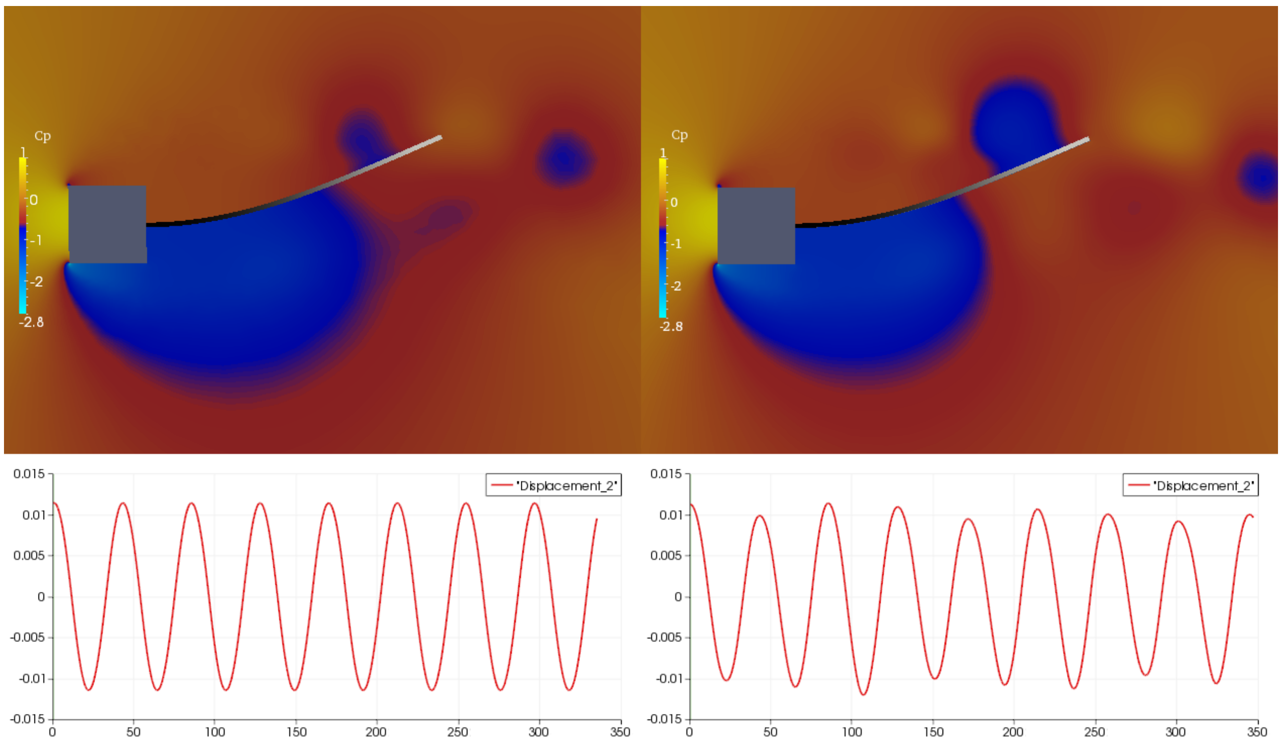
Imperial College London

Native FSI solver with deformable solids

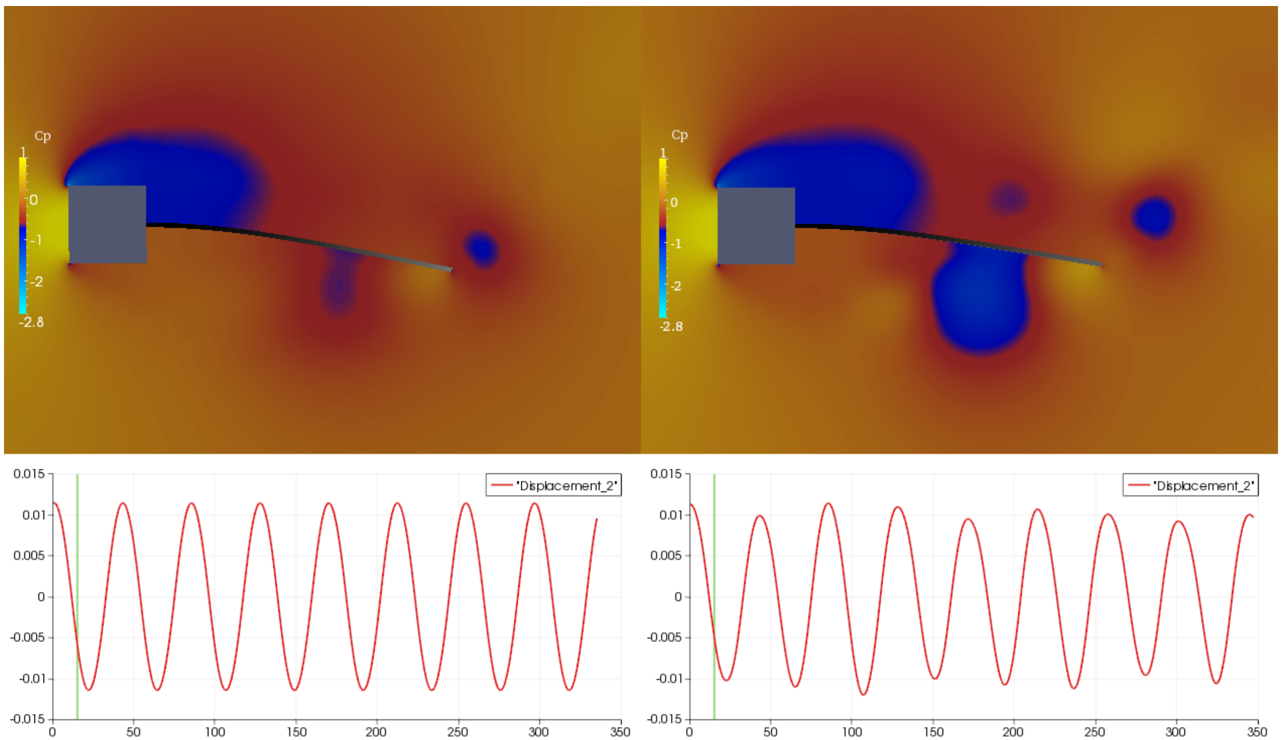


Imperial College London

Native FSI solver with deformable solids



Native FSI solver with deformable solids



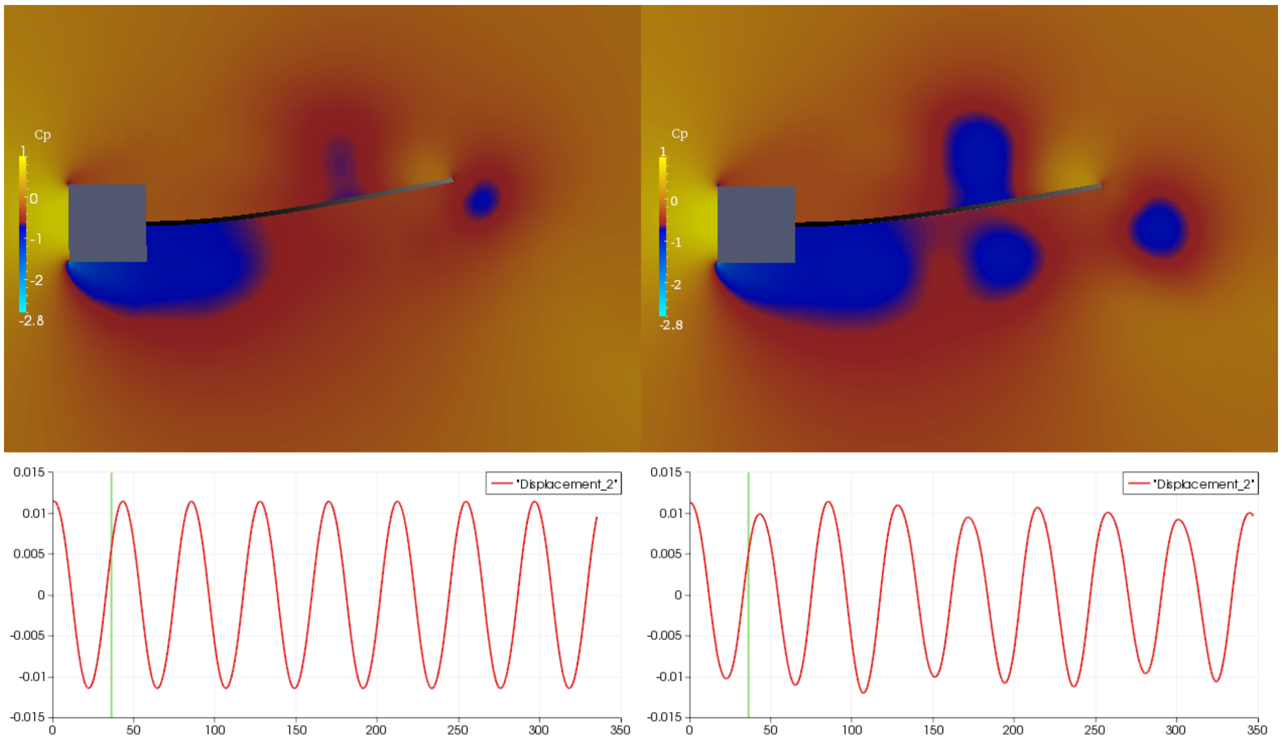
Complex vortical structures

Large displacements

Strong FSI couplings

Imperial College London

Native FSI solver with deformable solids



Complex vortical structures

Large displacements

Strong FSI couplings

Conclusions

1. Open-source FSI solvers tailored to computational aeroelasticity.
2. Two approaches adopted
   - Natively embedded solver
   - Python code wrapper for improved flexibility
3. Software architecture ready and demonstrated on first applications. Modularity has been preserved.
4. Source code already merged. Some first tutorials are available (more to come).