Towards high-fidelity industrial fluid dynamics simulations at high order

David Moxey
College of Engineering, Maths & Physical Sciences, University of Exeter

Outline

- Motivation
- What are high order methods and why are they useful?
- Challenges of higher order methods (and some solutions!)
- Nektar++: a spectral/*hp* element framework
- Applications

Lombard, Moxey et al., *AIAA J* (2016)

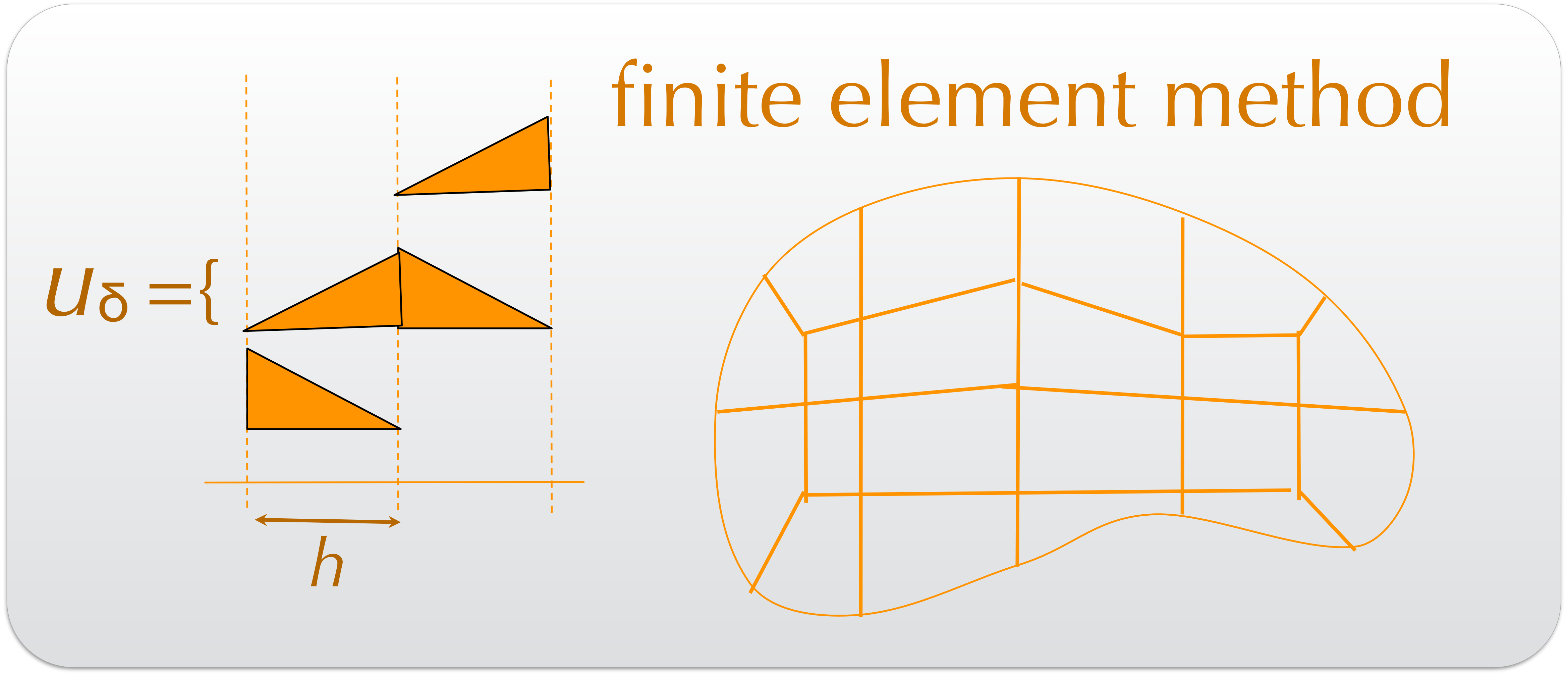
Increasing desire for **high-fidelity** simulation in high-end engineering applications.

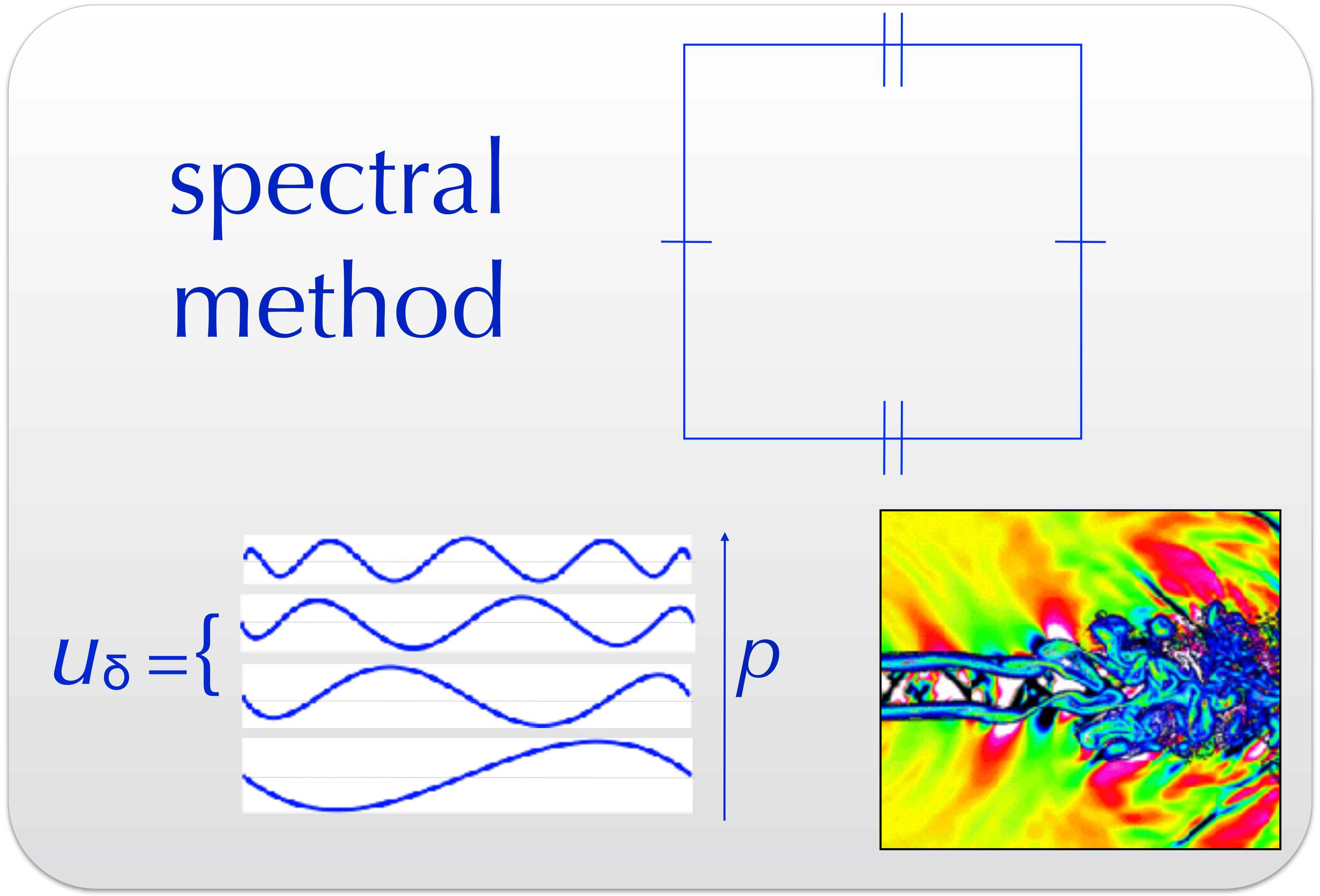
Want to accurately model difficult features:

- strongly separated flows
- feature tracking and prediction
- vortex interaction

My goal: develop methods and techniques for making LES **affordable**

What are high-order methods?





spatial flexibility (*h*)

+

accuracy (*p*)

→

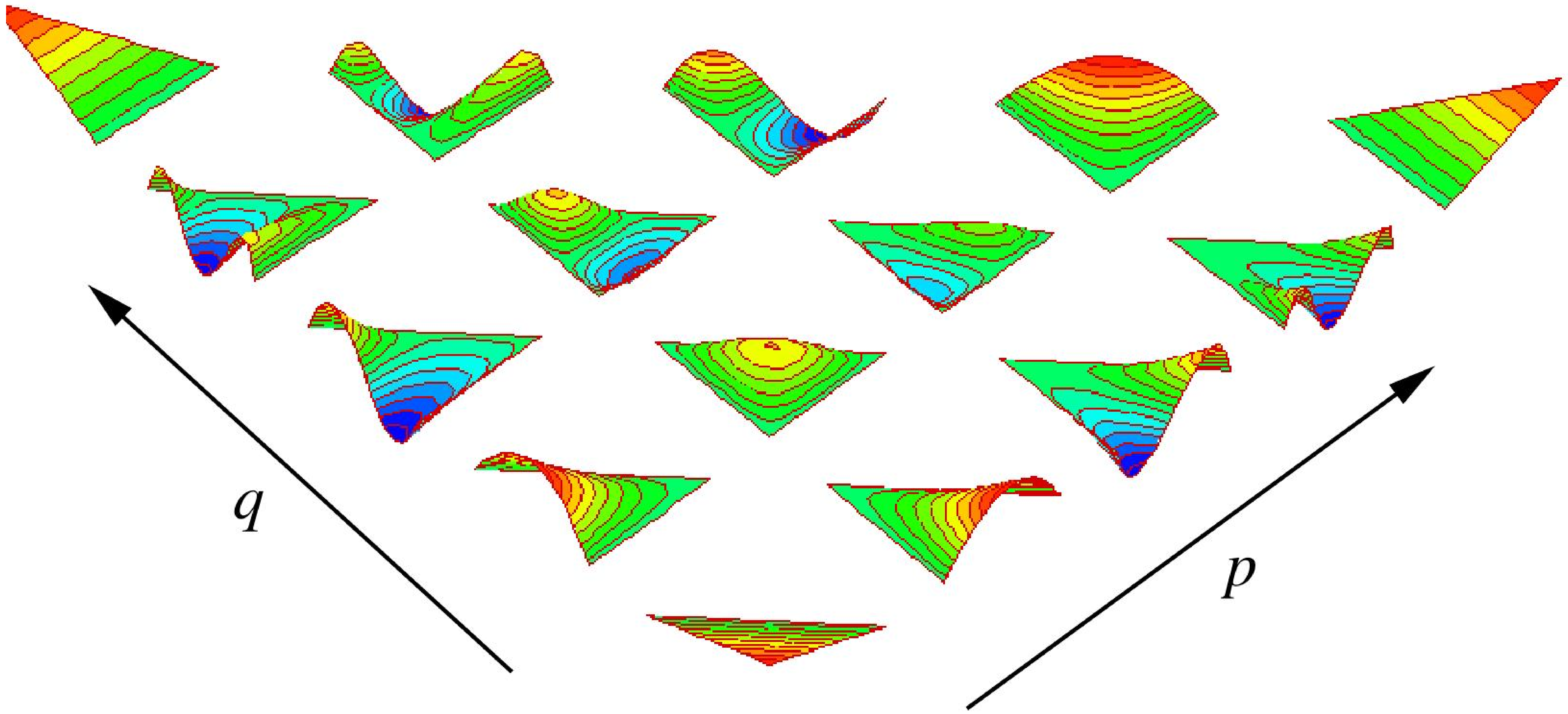
spectral/*hp* element

Higher-order expansions

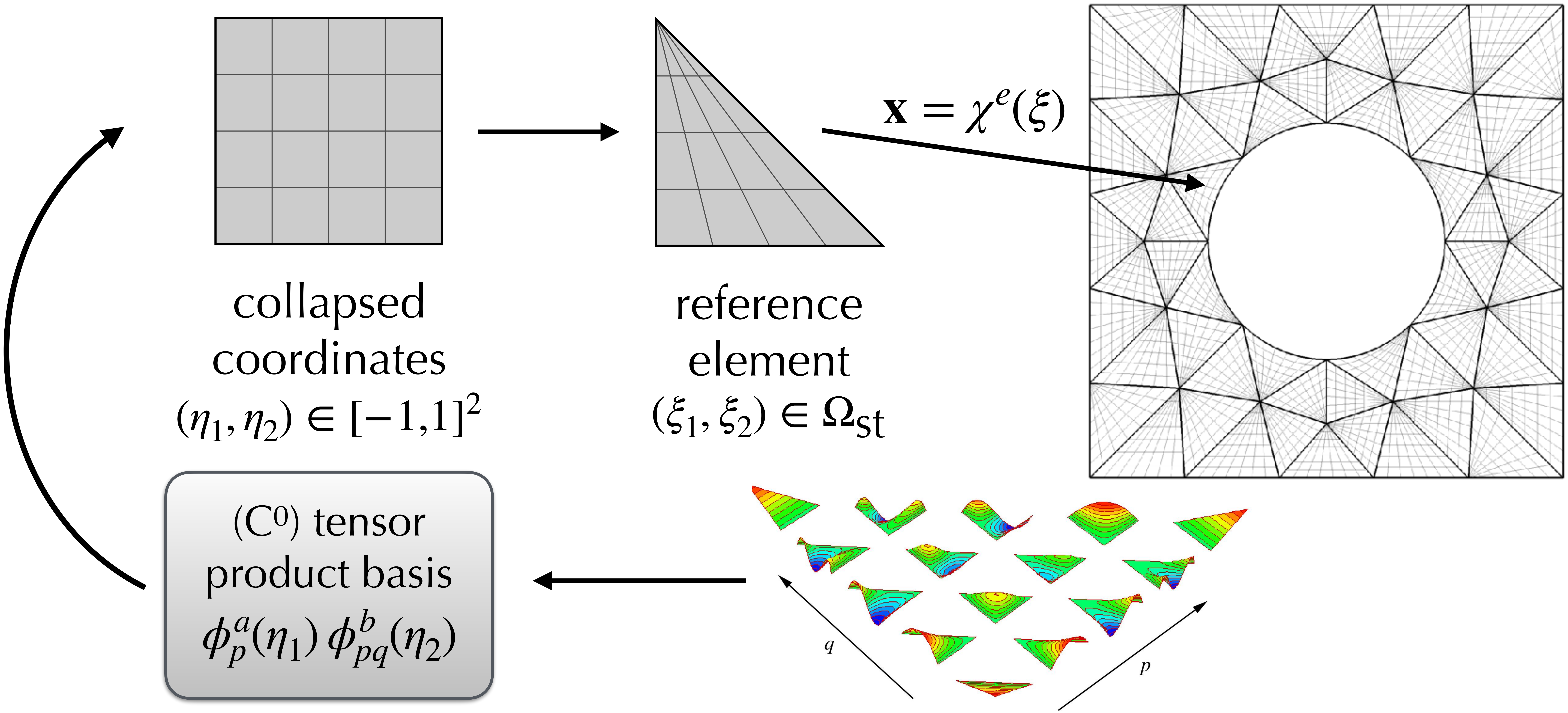
- Extend traditional FEM by adding higher order polynomials of degree *P* within each element.

- Traditional linear triangular elements have 3 degrees of freedom per element (each vertex).

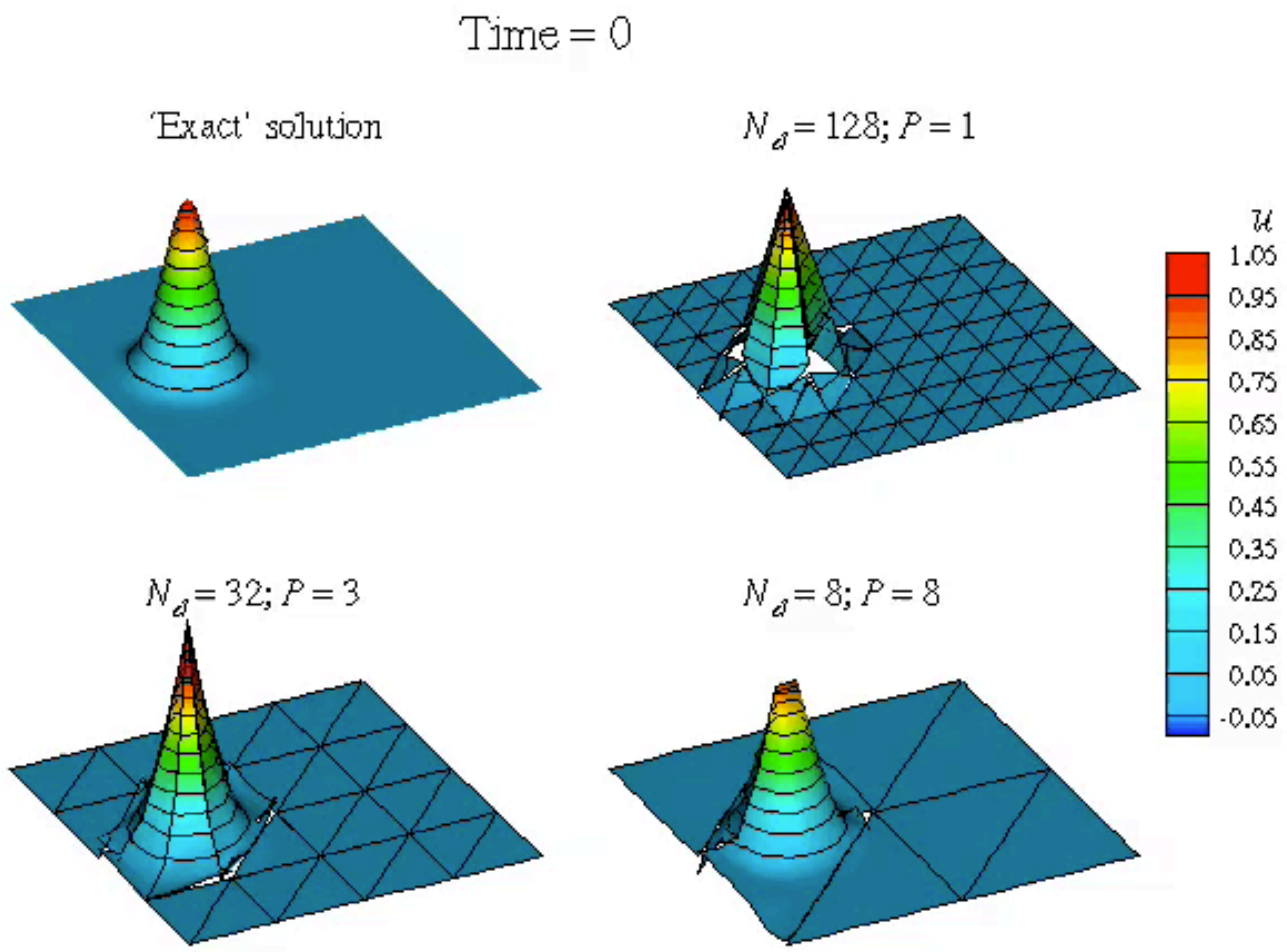
- High-order has $(P+1)(P+2)/2$ at a given order *P*.



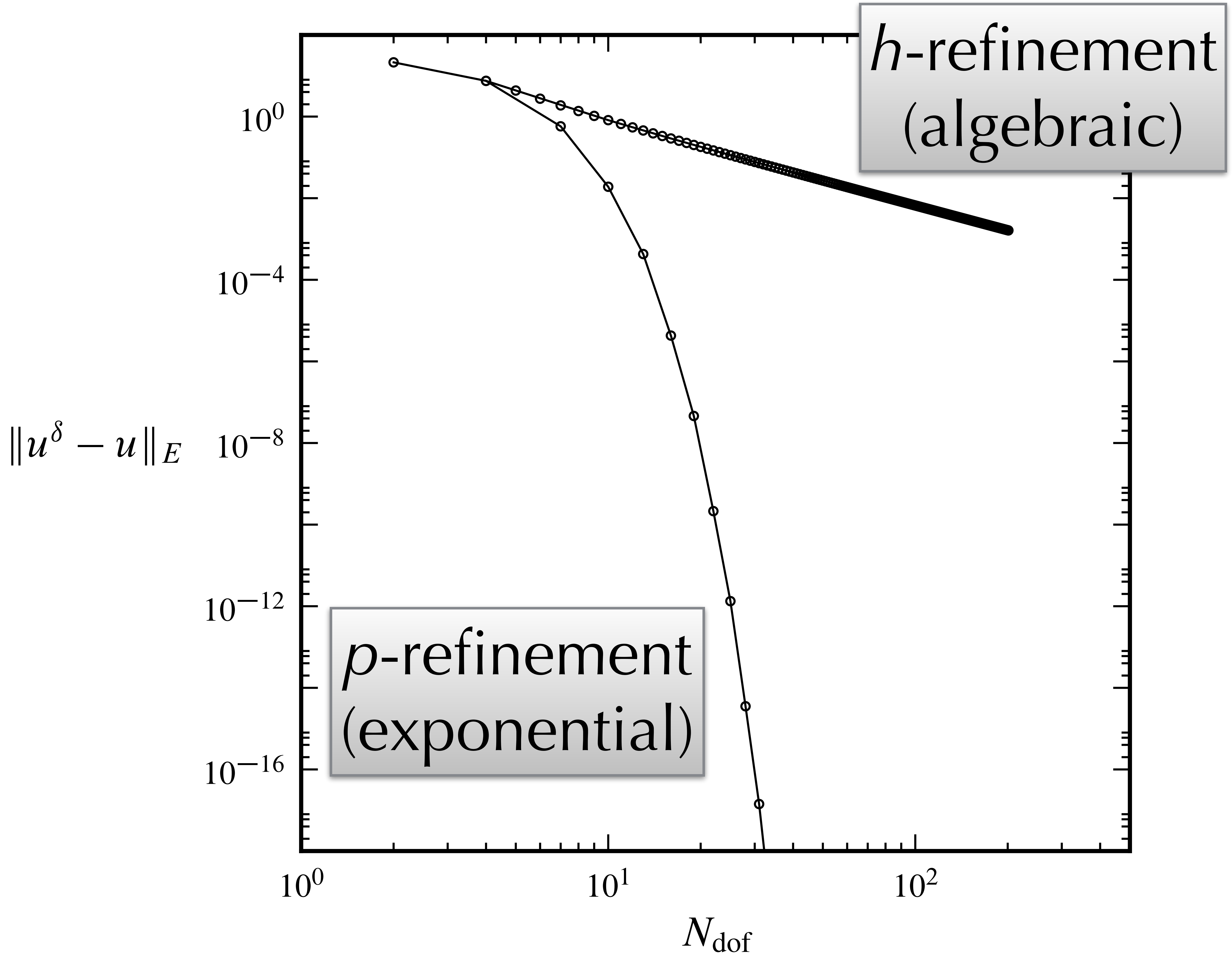
Spectral/*hp* element formulation



Why use a high-order method?



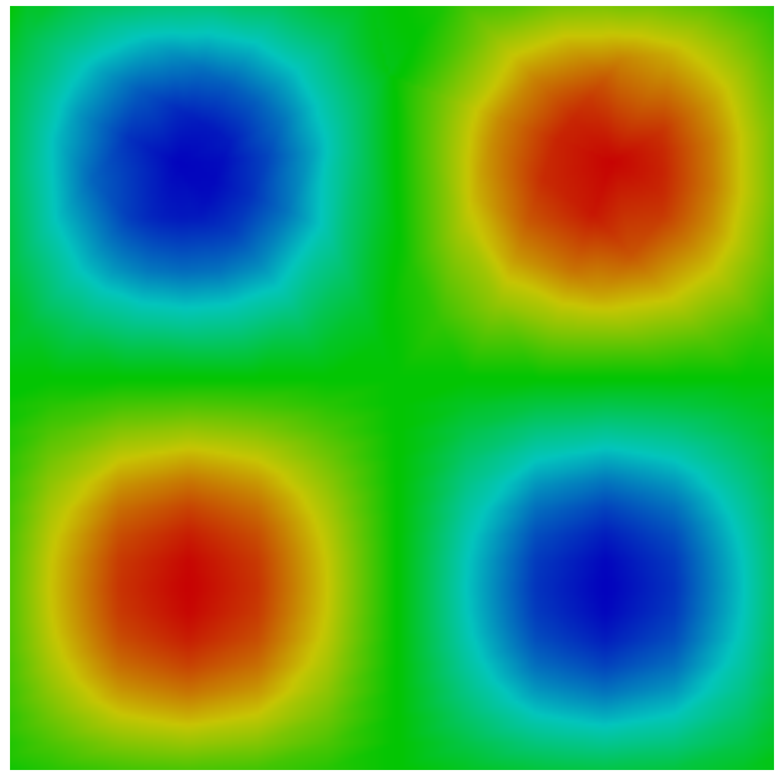
Why use a high-order method?



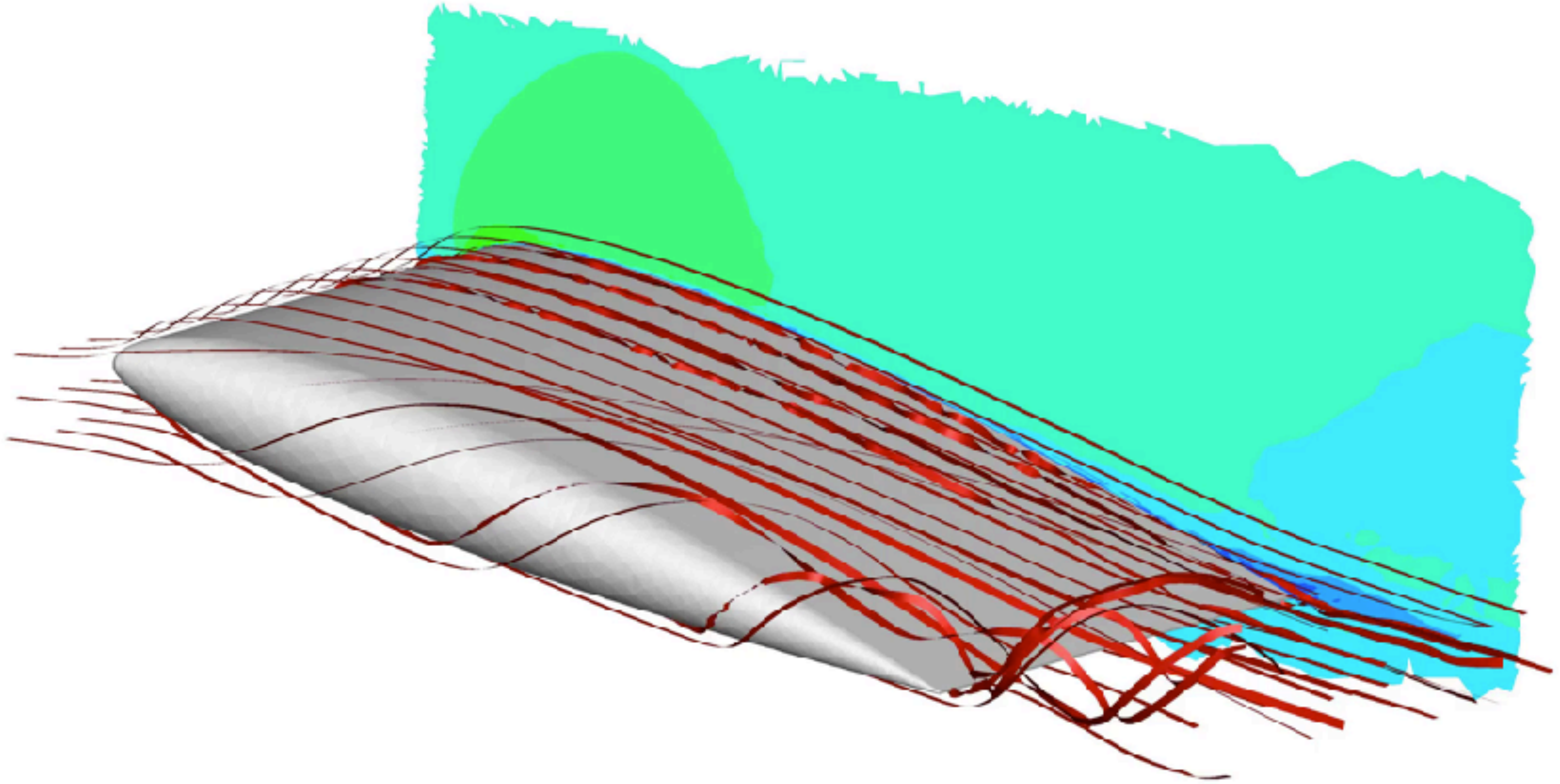
$$\nabla^2 u(x) - \lambda u(x) = -f(x)$$

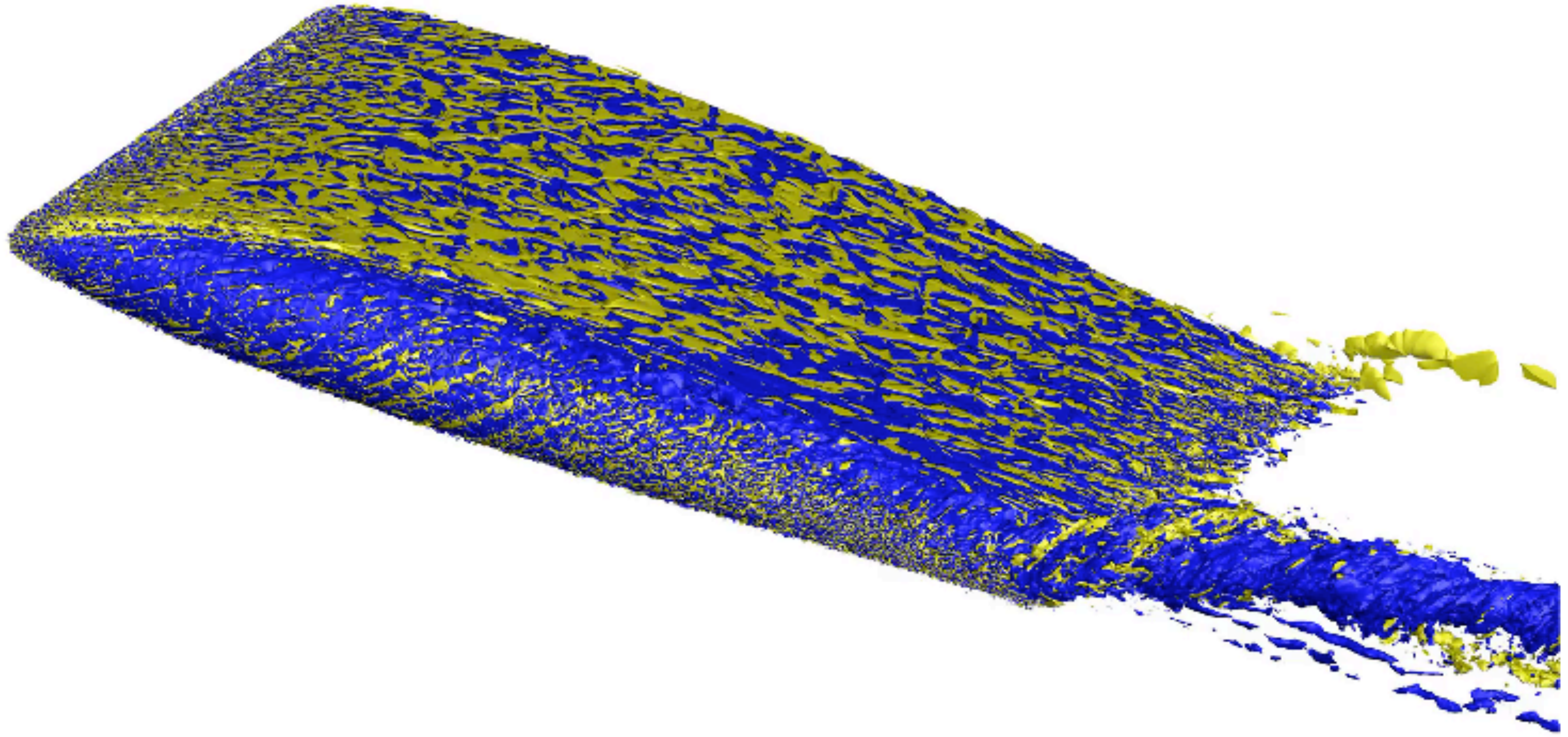
$$u(x) = \sin(\pi x)\sin(\pi y)$$

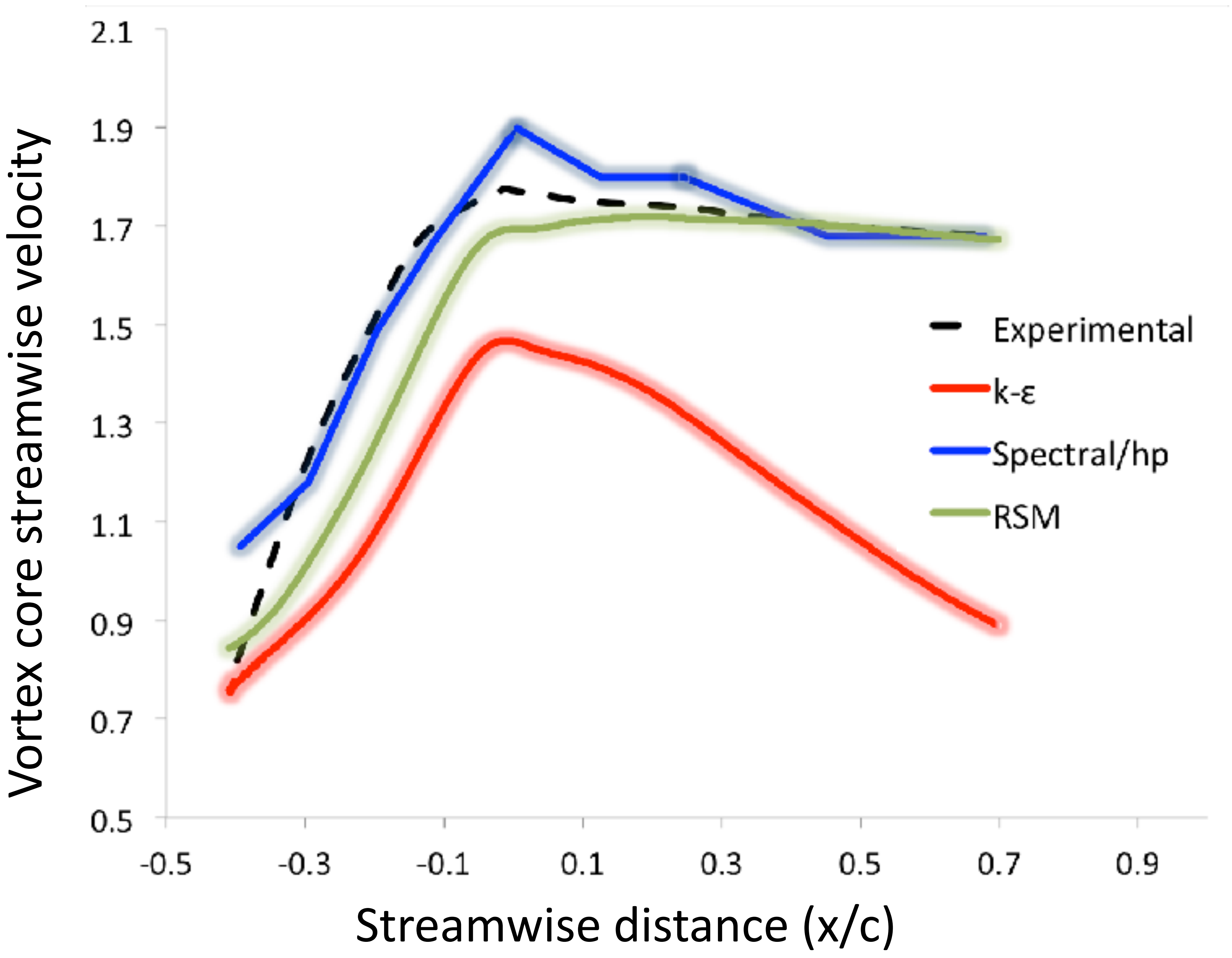
$$\Rightarrow f(x) = (\nabla^2 - \lambda)u(x)$$



NACA 0012 example







Lombard, Moxey, Hoessler, Dhandapani, Taylor and Sherwin
AIAA Journal (2016)

So why doesn't everyone use high-order?

Things I'll discuss today:

- Pre-processing (mesh generation), particularly for complex geometries.
- Efficiency & cost: linear algebra techniques & operator implementations.
- Difficulty and effort of implementation.

Other issues:

- Post-processing and visualisation, stability and robustness, preconditioning…

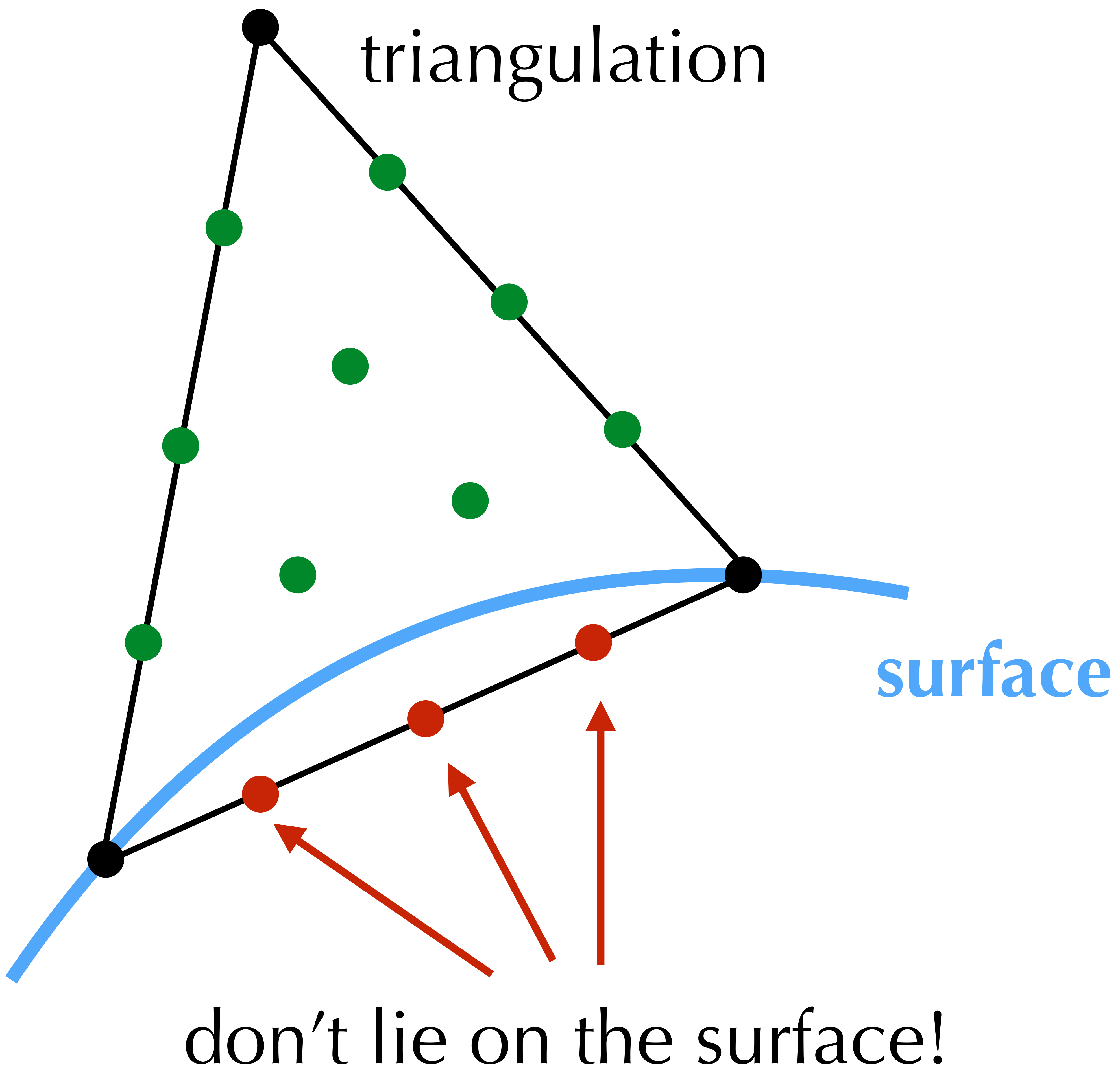
Challenge 1: high-order mesh generation

Complex geometries
look like this

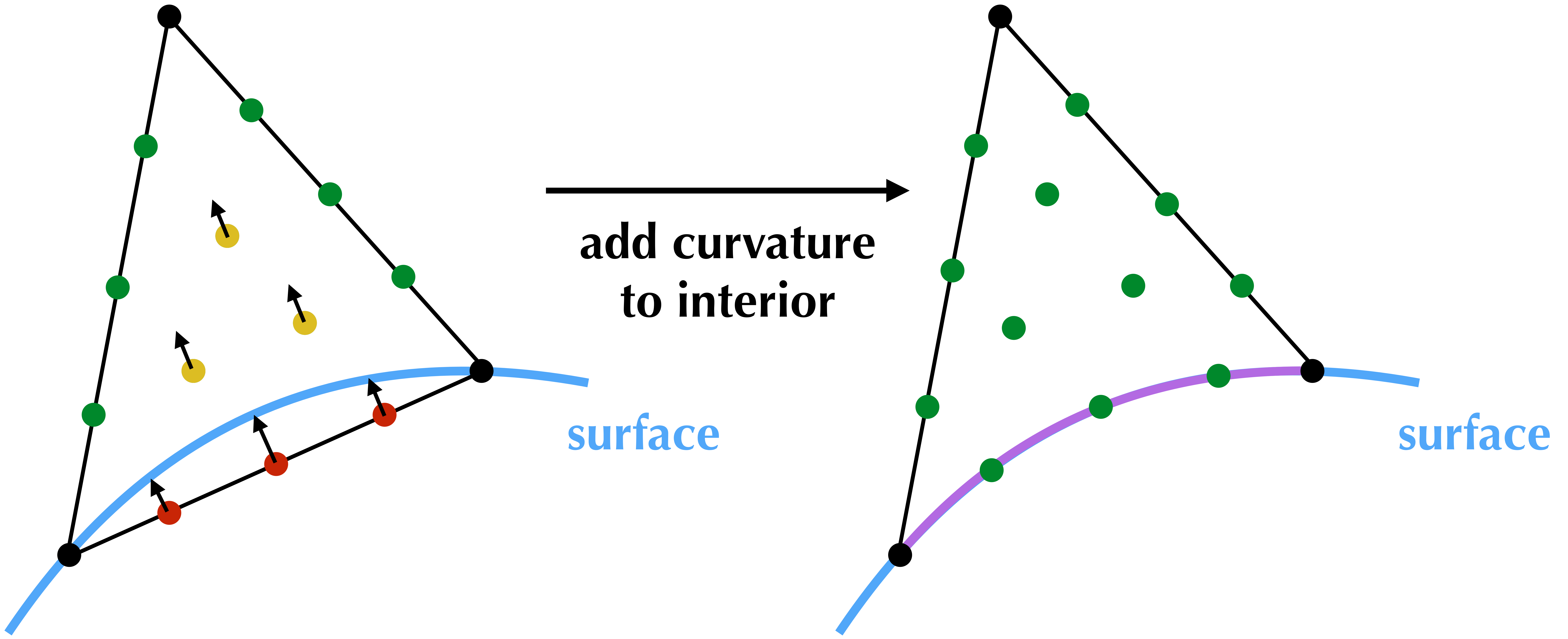
Not like this

High-order mesh generation

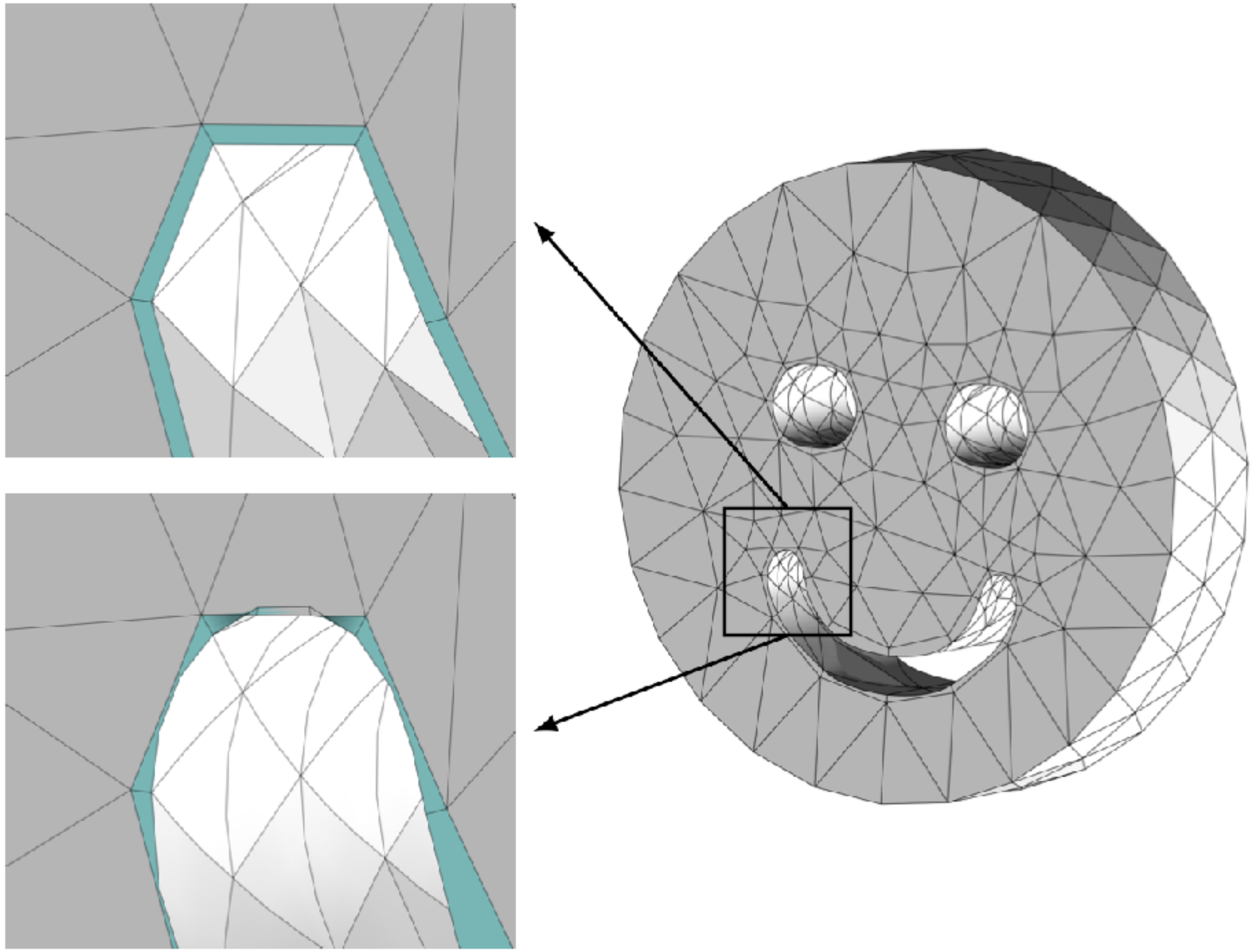
- Good quality meshes are **essential** to finite element and finite volume simulations.
- You can have a very fancy solver, but without a mesh you **can’t run your simulation**!
- At high orders we have an additional headache, as we must **curve the elements** to fit the geometry.



High-order mesh generation

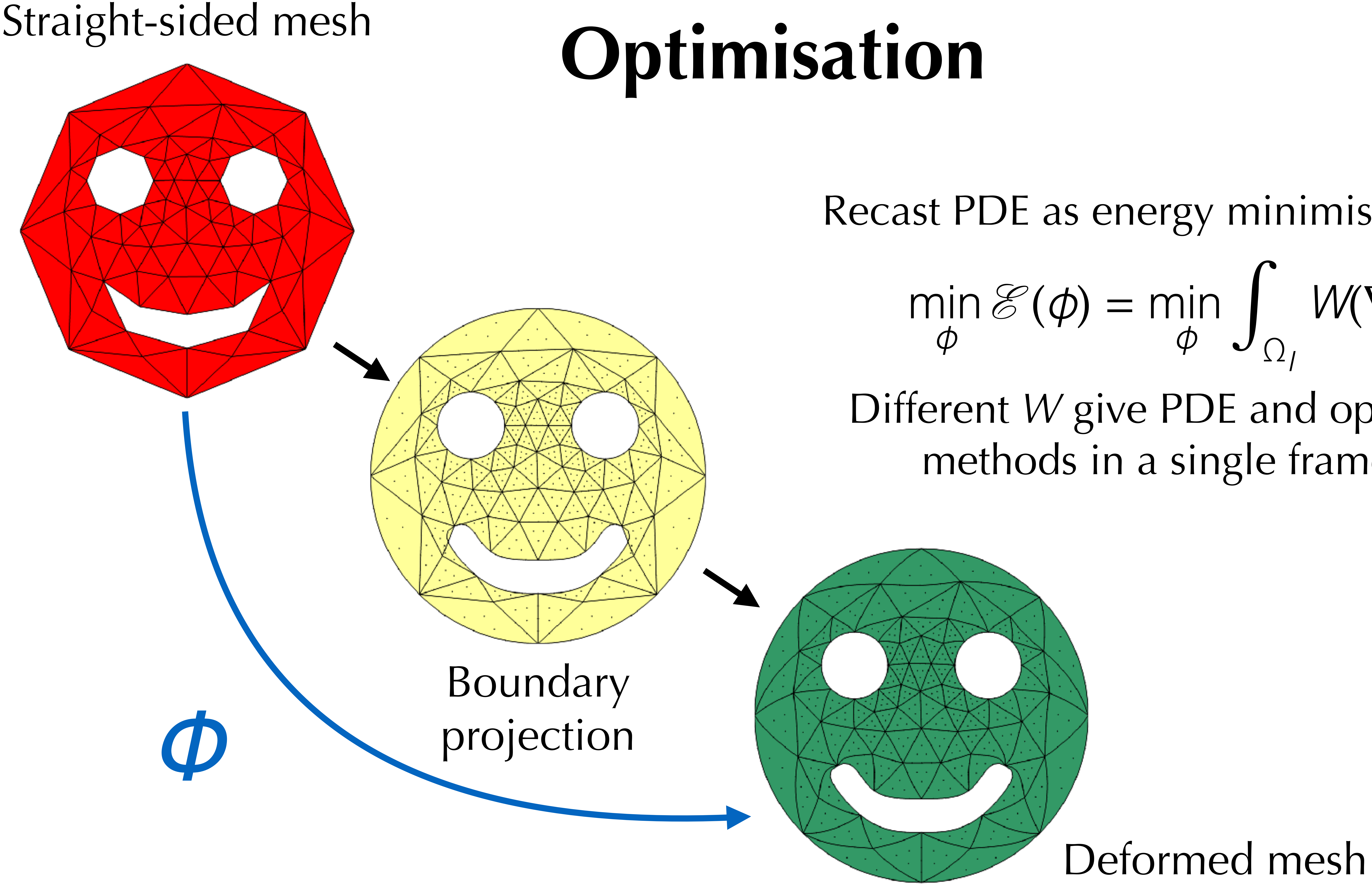


High-order mesh generation



- Curving coarse meshes leads to invalid elements.
- Most existing mesh generation packages cannot deal with this.
- Involves non-trivial optimisation procedure.
- Therefore a need to develop new techniques.

Optimisation

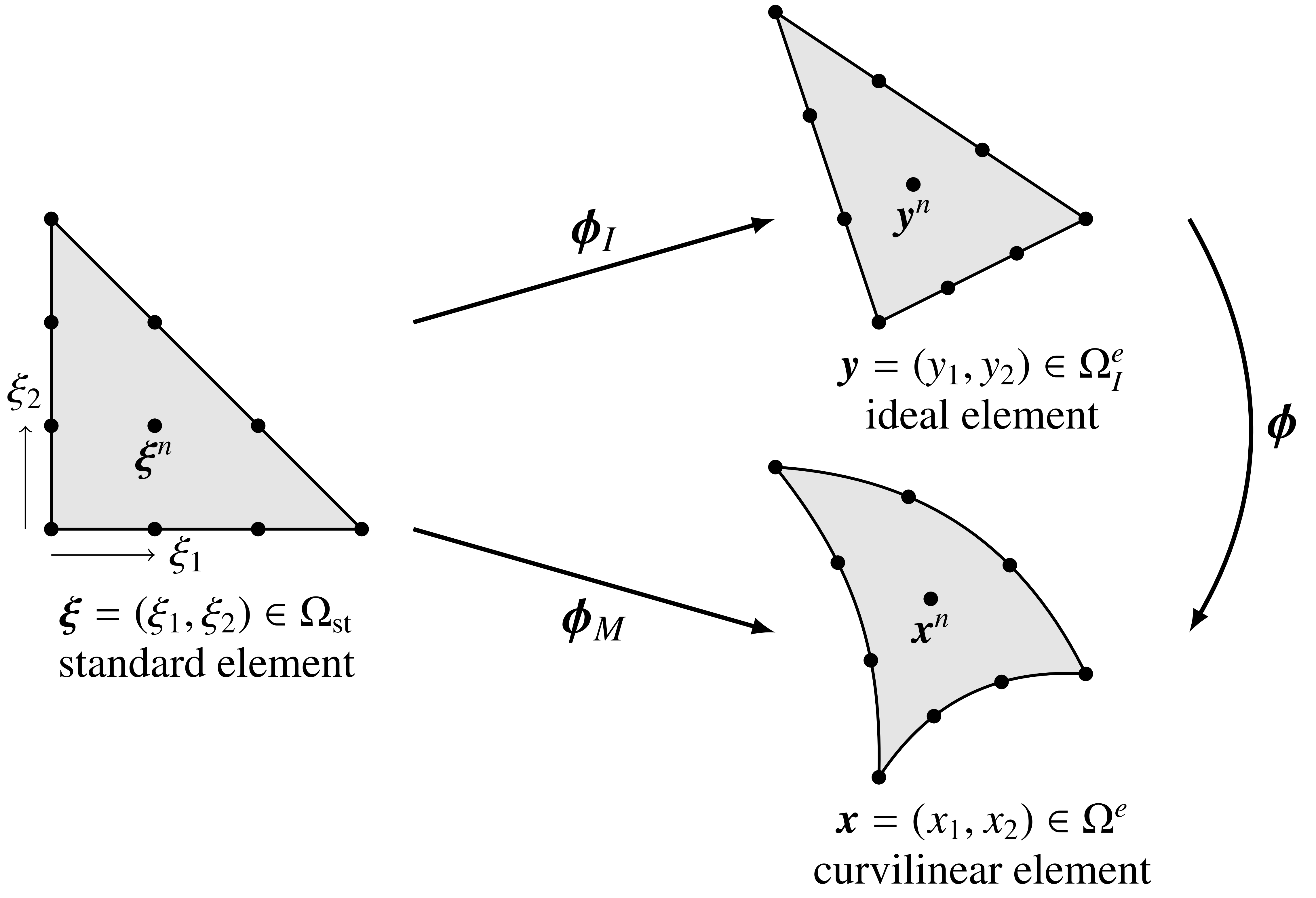


Recast PDE as energy minimisation: solve

$$\min_{\phi} \mathscr{E}(\phi) = \min_{\phi} \int_{\Omega_I} W(\nabla \phi)\, dy$$

Different W give PDE and optimisation methods in a single framework

Variational approach



$$\min_{\phi} \mathscr{E}(\phi) = \min_{\phi} \int_{\Omega_I} W(\nabla\phi)\, dy$$

$$W = \frac{\kappa}{2}(\ln J)^2 + \mu\, \mathbf{E} : \mathbf{E}; \quad \mathbf{E} = \frac{1}{2}(\mathbf{F}^t\mathbf{F} - \mathbf{I})$$

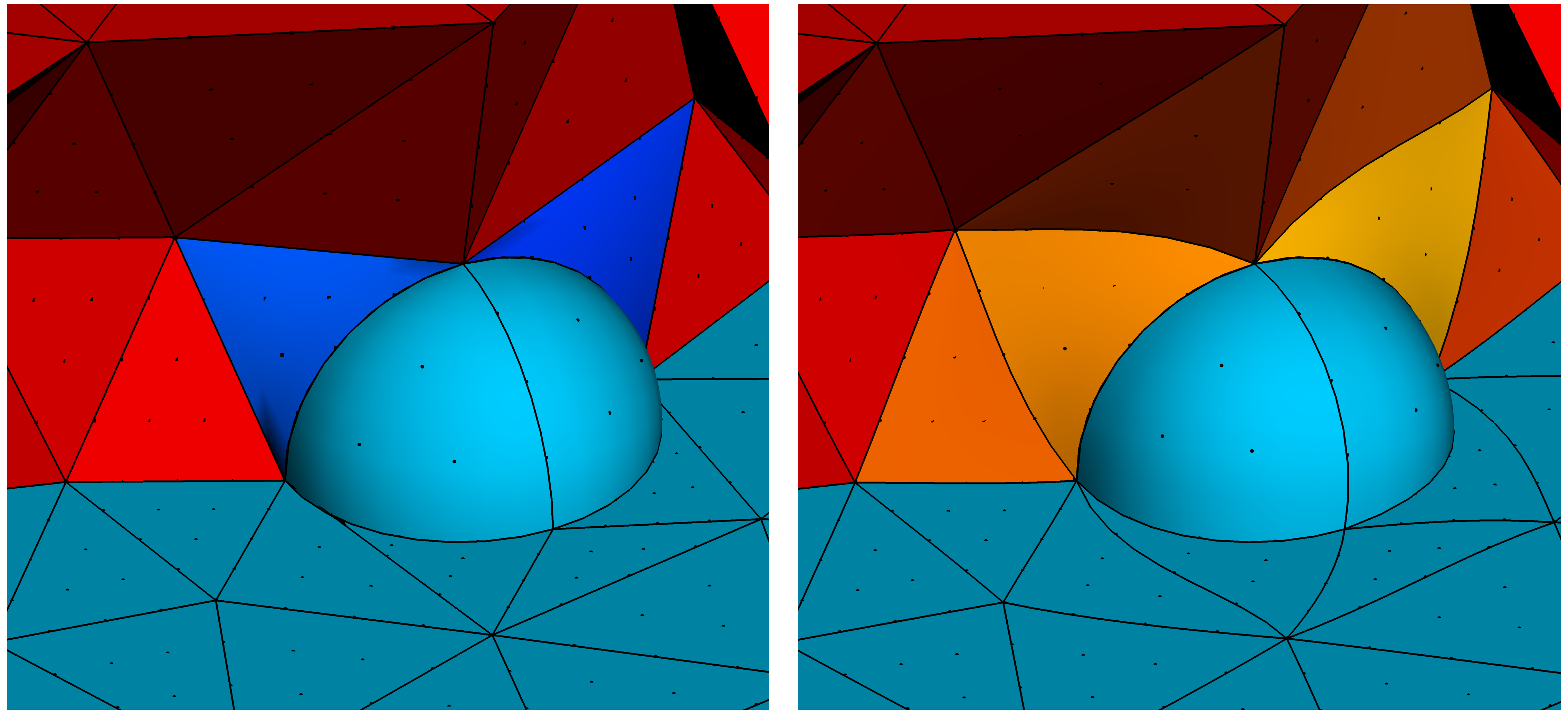
$$W = \frac{\mu}{2}(\mathbf{F} : \mathbf{F} - 3) - \mu \ln J + \frac{\lambda}{2}(\ln J)^2$$

$$W = J^{-1}\,(\mathbf{F} : \mathbf{F})$$

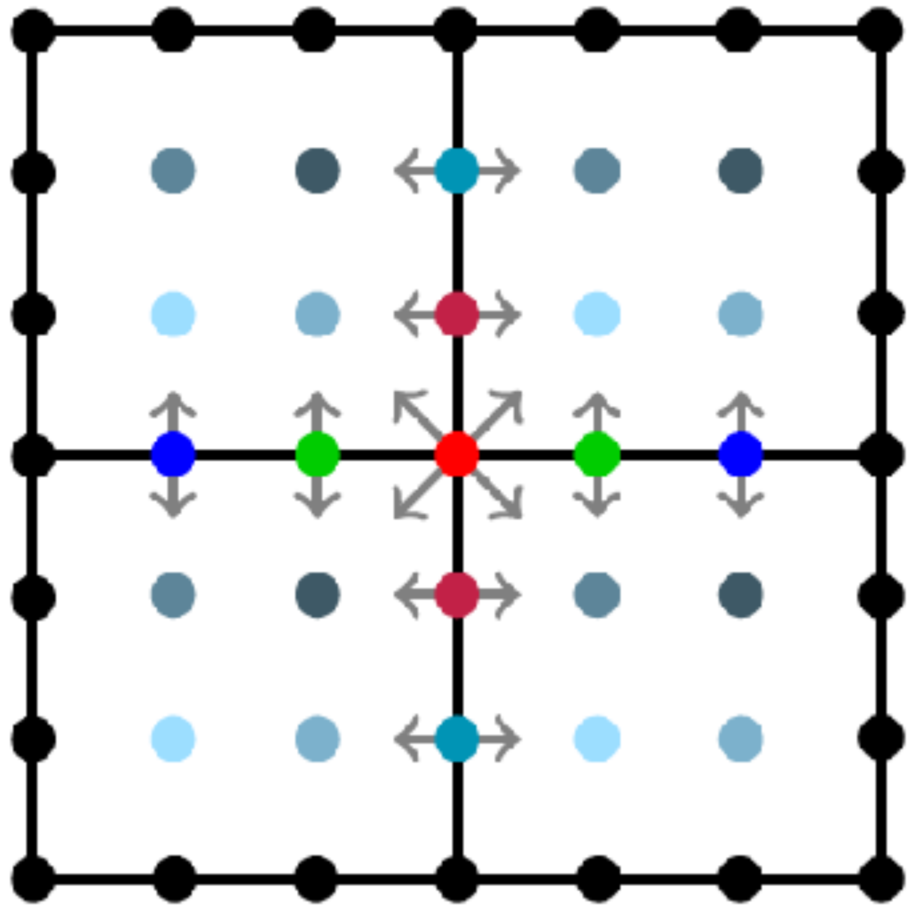
$$W = \frac{1}{d}|J|^{-d/2}(\mathbf{F} : \mathbf{F})$$

M. Turner, J. Peiró, D. Moxey, *Curvilinear mesh generation using a variational framework*
Computer Aided Design **103** 73-91 (2018)

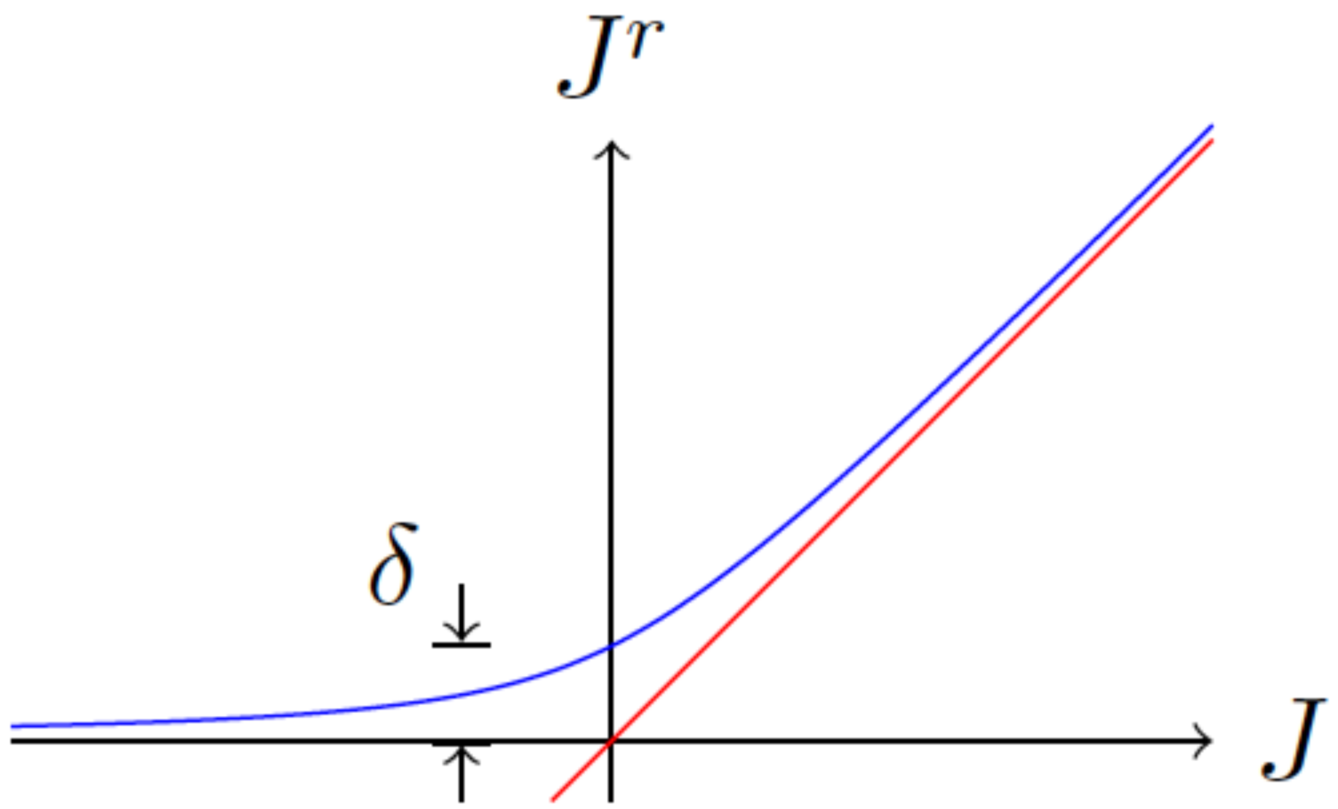
Benefits



CAD sliding

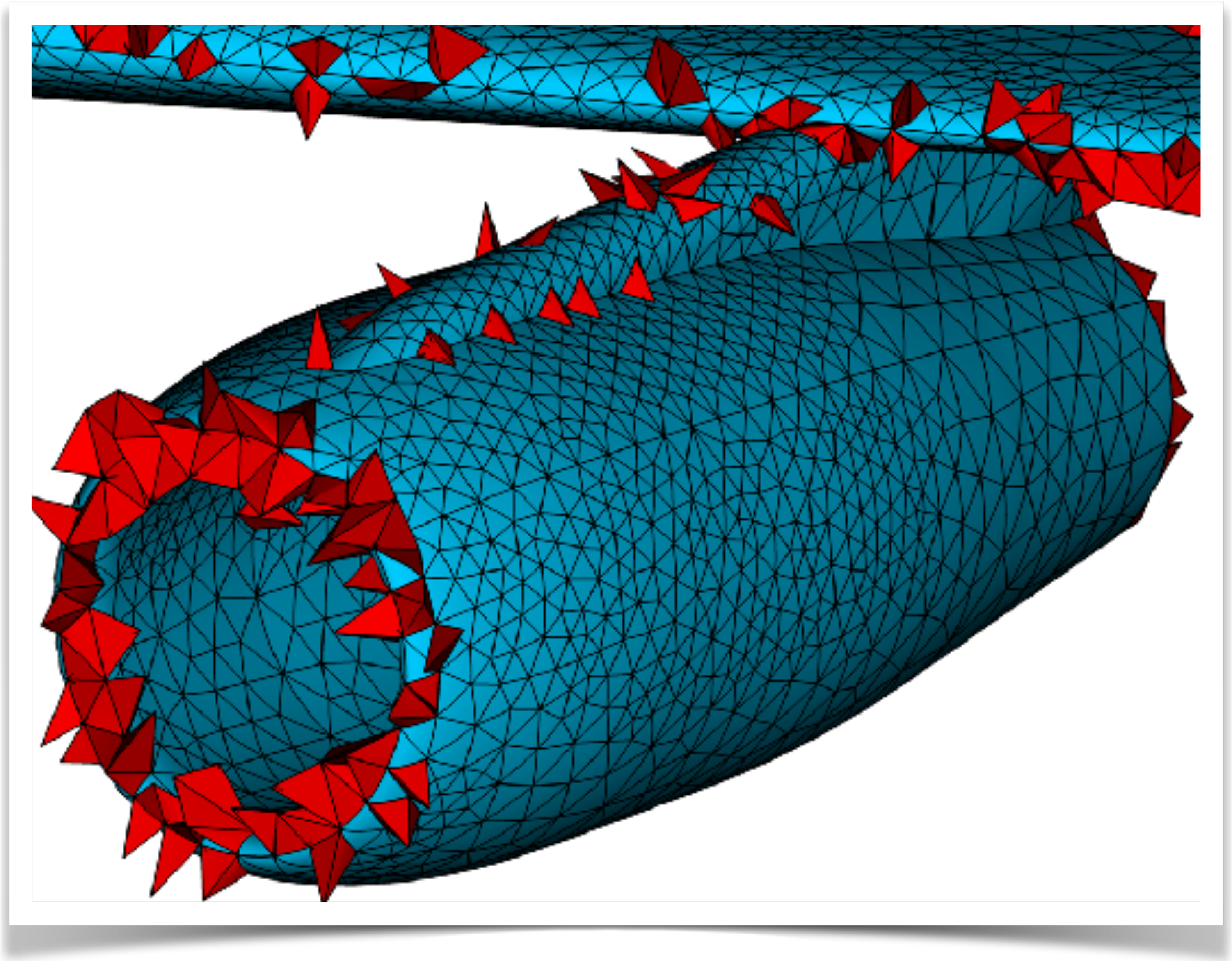


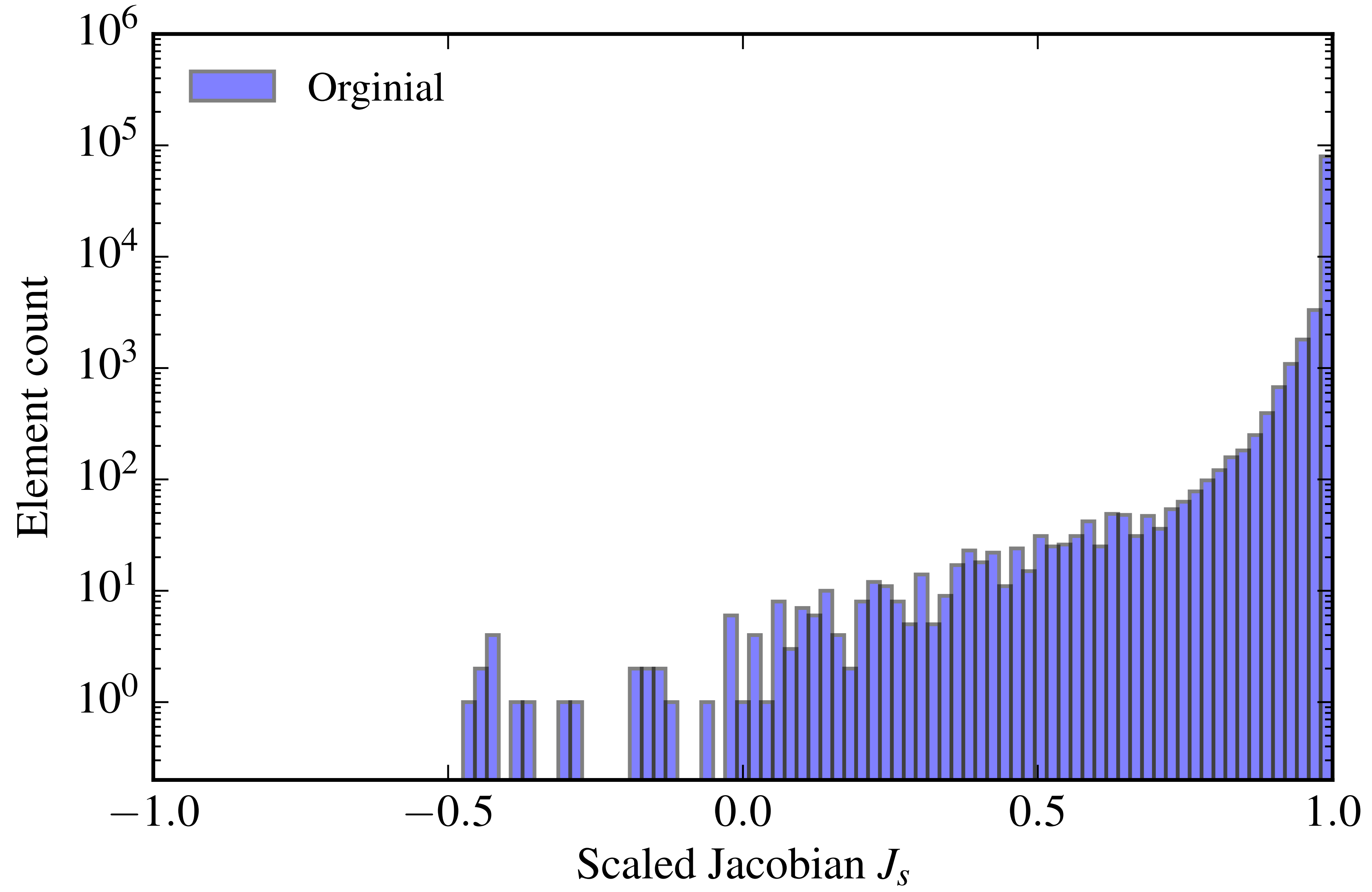
Multi-core parallelisation
relaxation optimisation approach

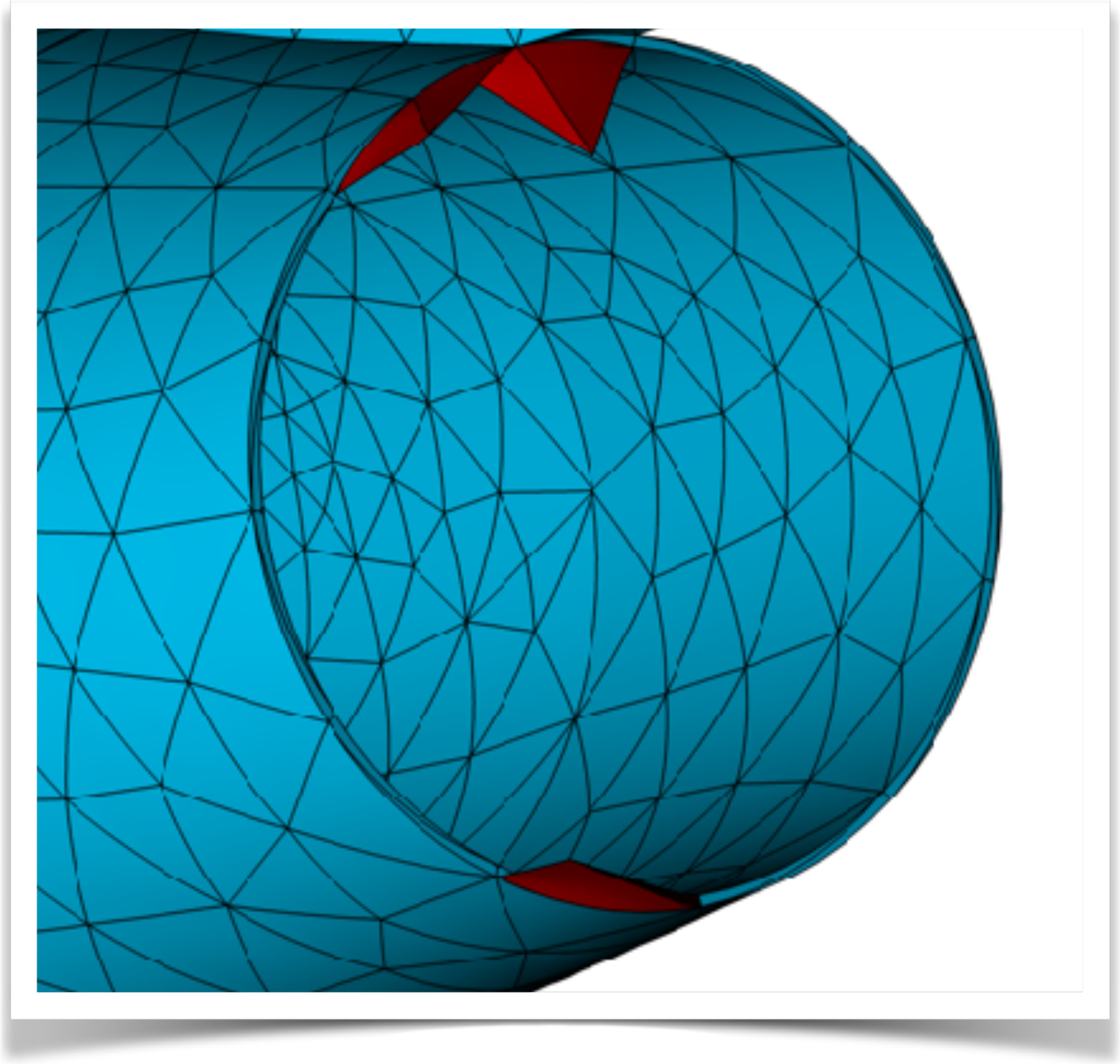


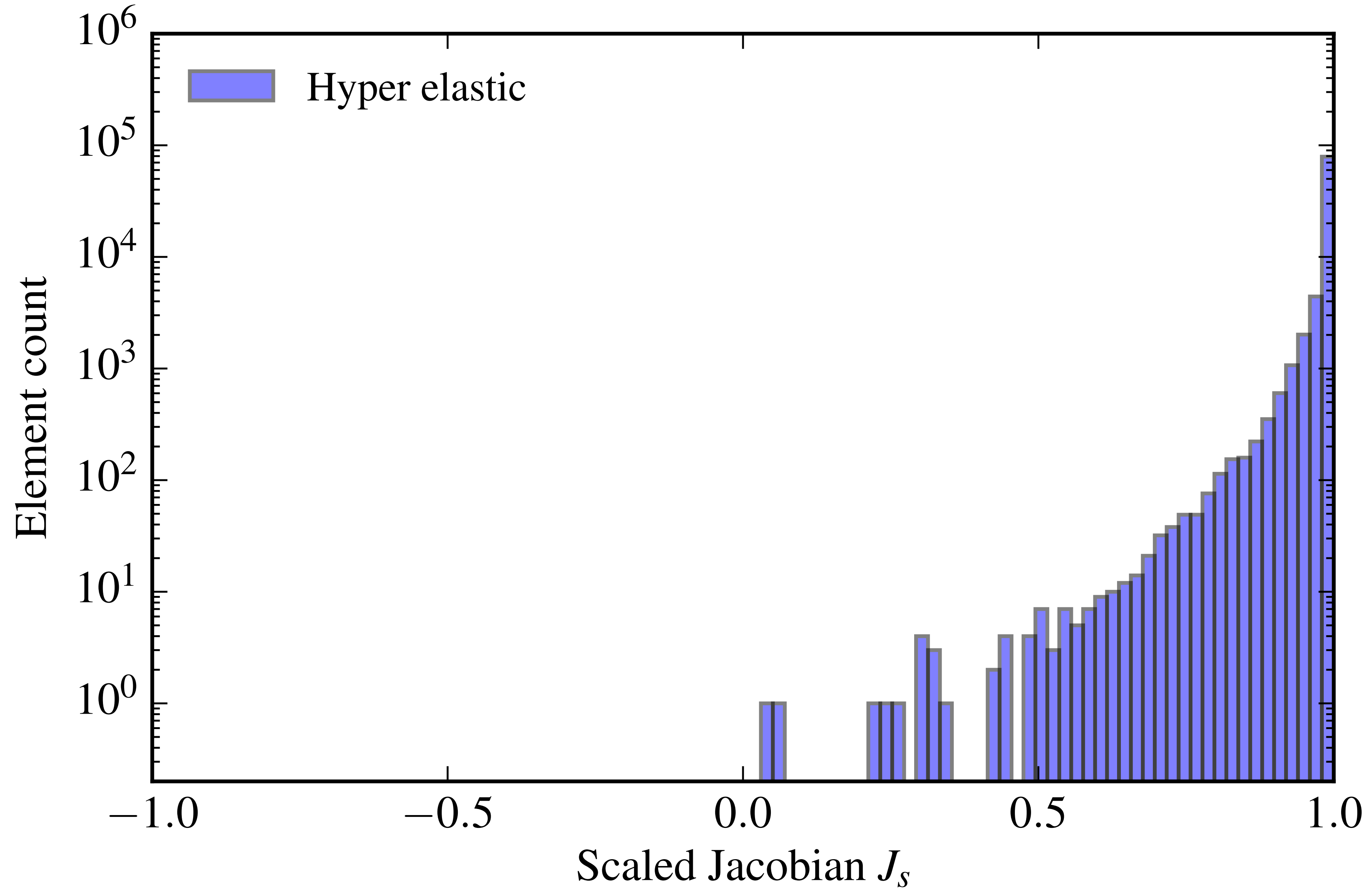
Untangles meshes
using Jacobian regularisation

Example: DLR F6 engine





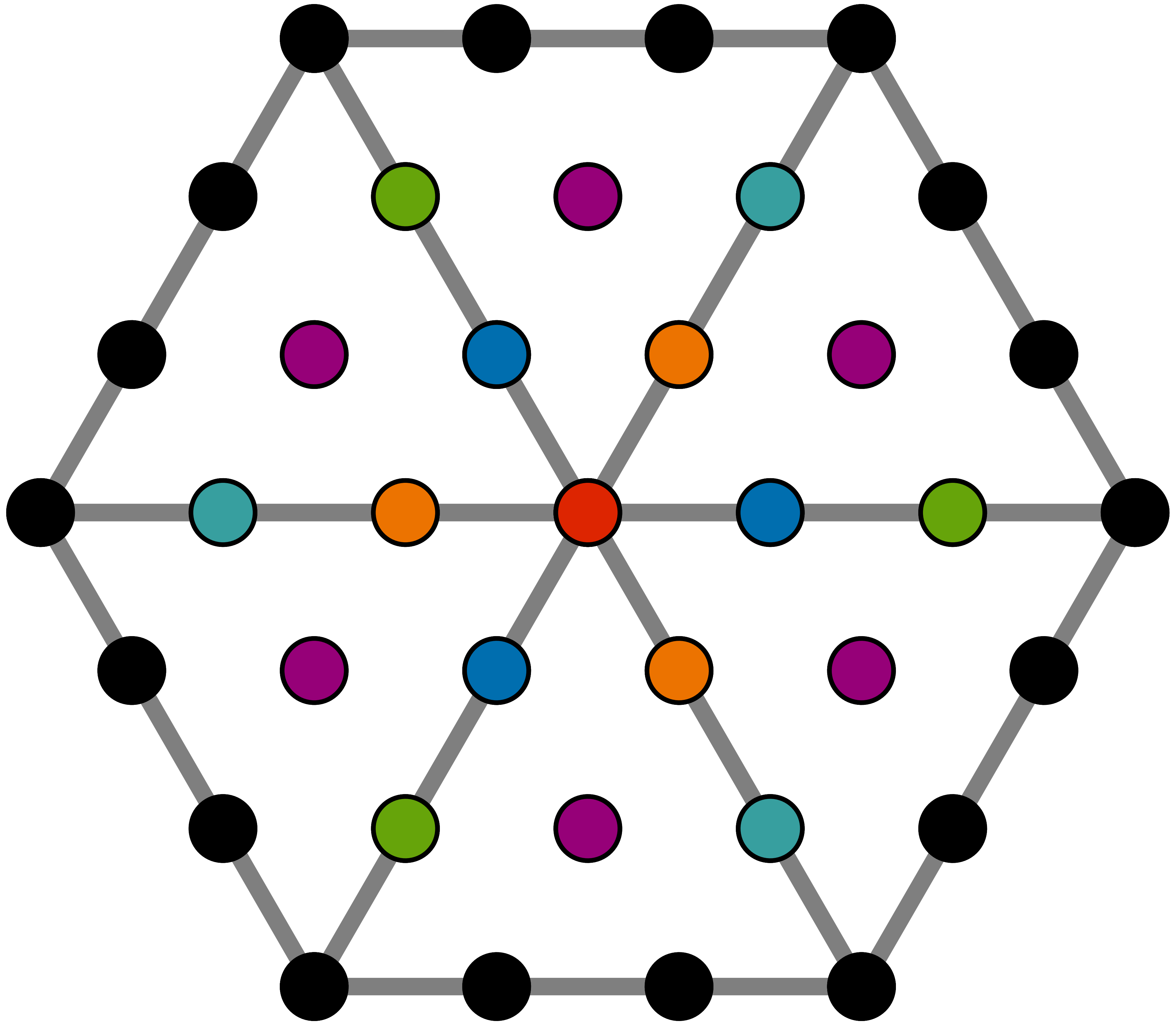




Speeding up optimisation

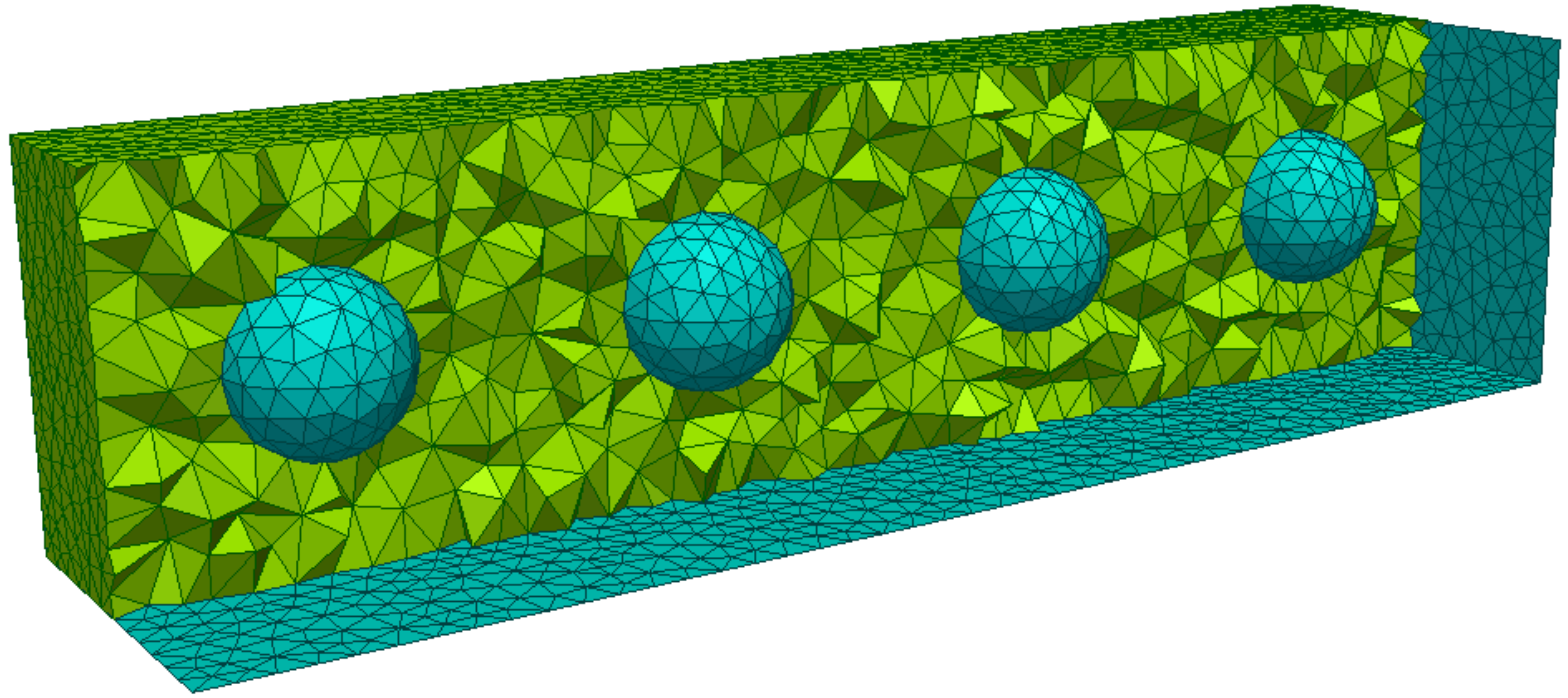
- Meshing usually accomplished on a single workstation, generally repeated as part of many design iterations.
- Optimisation process is resource intensive, but GPUs have lots of compute density.
- Can we leverage parallelism of the method effectively on a GPU?
- How do we do this in a code-friendly way?

Node colouring

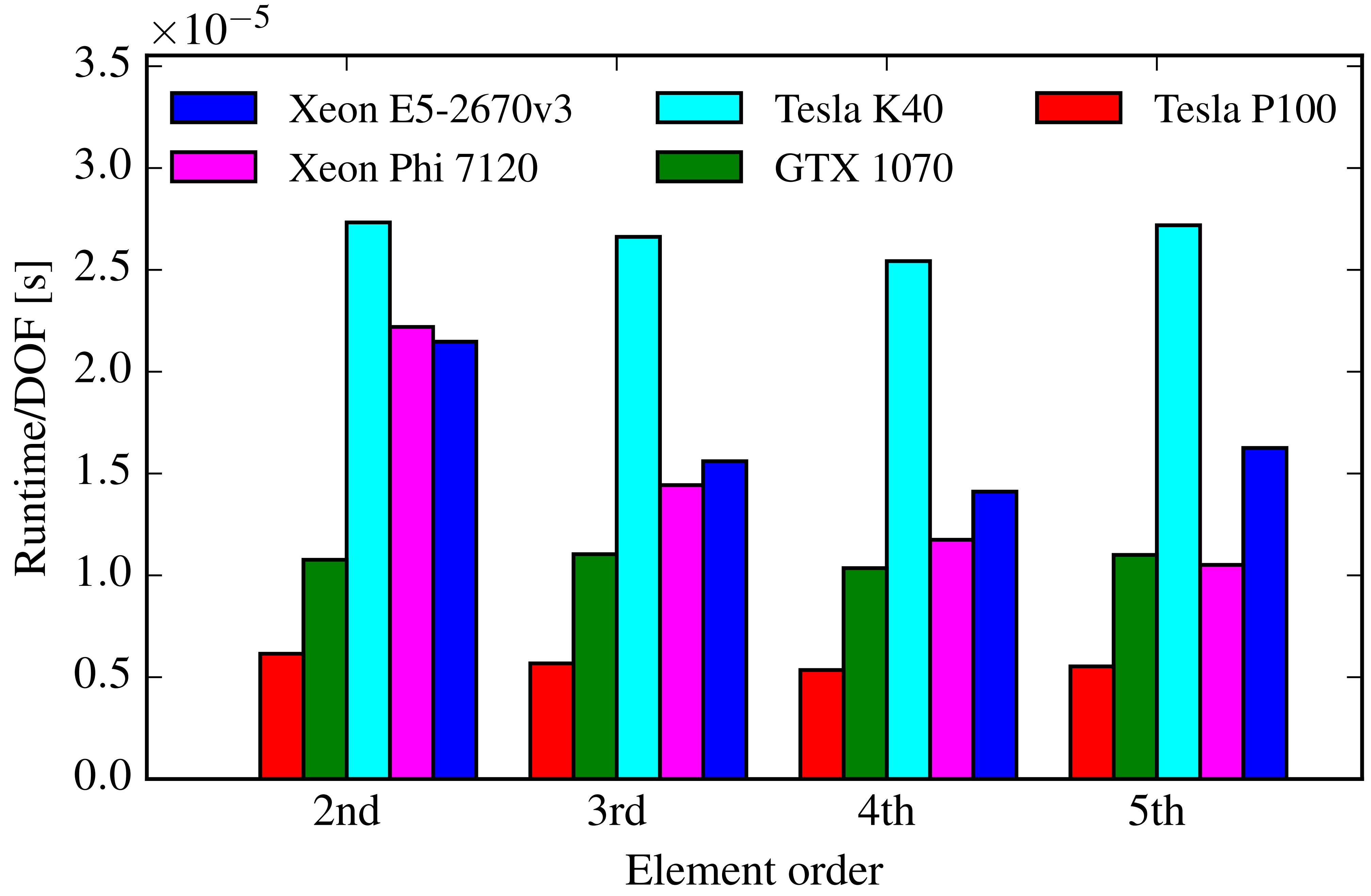


- For each node, solve local minimisation problem.
- Calculate functional + gradients analytically.
- Uses multi-level threading to exploit GPU hierarchy: use Kokkos.
- Iterate until global functional residual is small.

Results



Four spheres in a box, 33k tetrahedra, ~400k nodes at p = 5



Reasonably consistent runtimes per DoF across polynomial orders

Challenge 2: efficient implementation

- Today's computational hardware: lots of FLOPS available, but really hard to use them.
- Algorithms will only use hardware effectively if they are **arithmetically intense:** i.e. high ratio of FLOPS per byte of memory transfer.
- This is one of the reasons that current industry-standard CFD codes often do not make best use of hardware on offer.
- High-order has potential in this area through **matrix-free formulation** of the underlying operators.

Matrix-free FEM

- Common in FEM to compute (local or global) mass & stiffness matrices, e.g.

$$\mathbf{M}_{ij} = \int_{\Omega^e} \phi_i(x_1)\phi_j(x_2)\,\mathrm{d}x \qquad \mathbf{S}_{ij} = \int_{\Omega^e} \nabla\phi_i(x_1)\cdot\nabla\phi_j(x_2)\,\mathrm{d}x$$

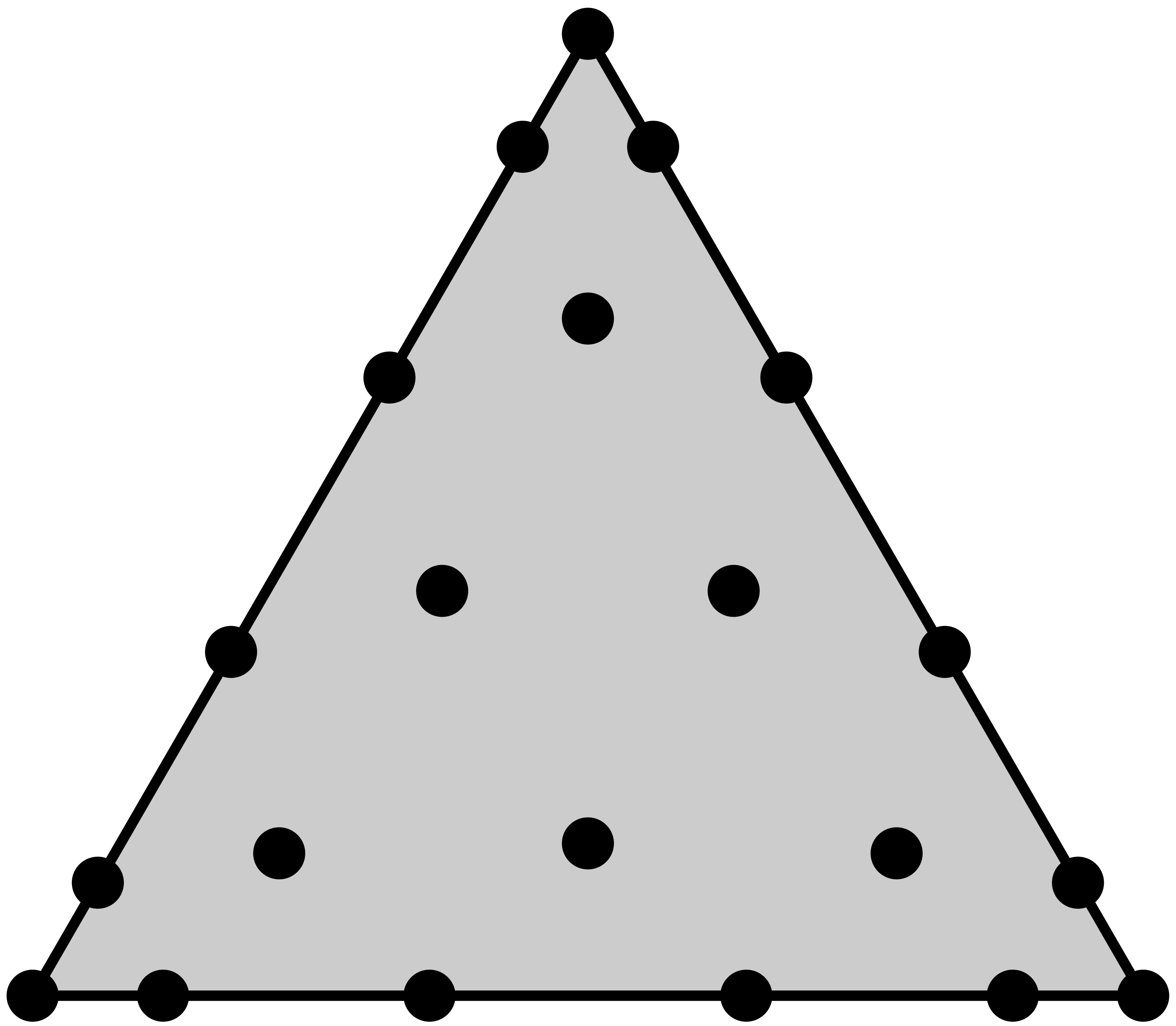
- For a hypercube: rank P^d, storage & multiplication cost $O(P^{2d})$.

- Entries computed using Gaussian quadrature: evaluation cost also $O(P^{2d})$; but the constant is important!

- Idea of matrix-free: compute *action* of local matrix by evaluating summations corresponding to integrals above to avoid memory transfer.

- Further efficiency if we use a **tensor product basis** to enable **sum-factorisation**.

Unstructured elements



P5 triangle, Fekete points

- Typically unstructured elements make use of Lagrange basis functions (although not always).
- Combine this with a suitable set of quadrature (cubature) points: no tensor-product structure.
- However, spectral/*hp* does have a tensor product structure!

$$u^{\delta}(\eta_1, \eta_2) = \sum_{p=0}^{P} \sum_{q=0}^{Q-p} \hat{u}_{pq} \phi_p^a(\eta_1) \phi_{pq}^b(\eta_2)$$

Sum-factorisation

Key to performance at high polynomial orders: complexity $O(P^{2d})$ to $O(P^{d+1})$!

$$u(\xi_{1i}, \xi_{2j}) = \sum_{p=0}^{P} \sum_{q=0}^{Q} \hat{u}_{pq} \phi_p(\xi_{1i}) \phi_q(\xi_{2j}) = \sum_{p=0}^{P} \phi_p(\xi_{1i}) \left[\sum_{q=0}^{Q} \hat{u}_{pq} \phi_q(\xi_{2j}) \right]$$

store this for each *p*

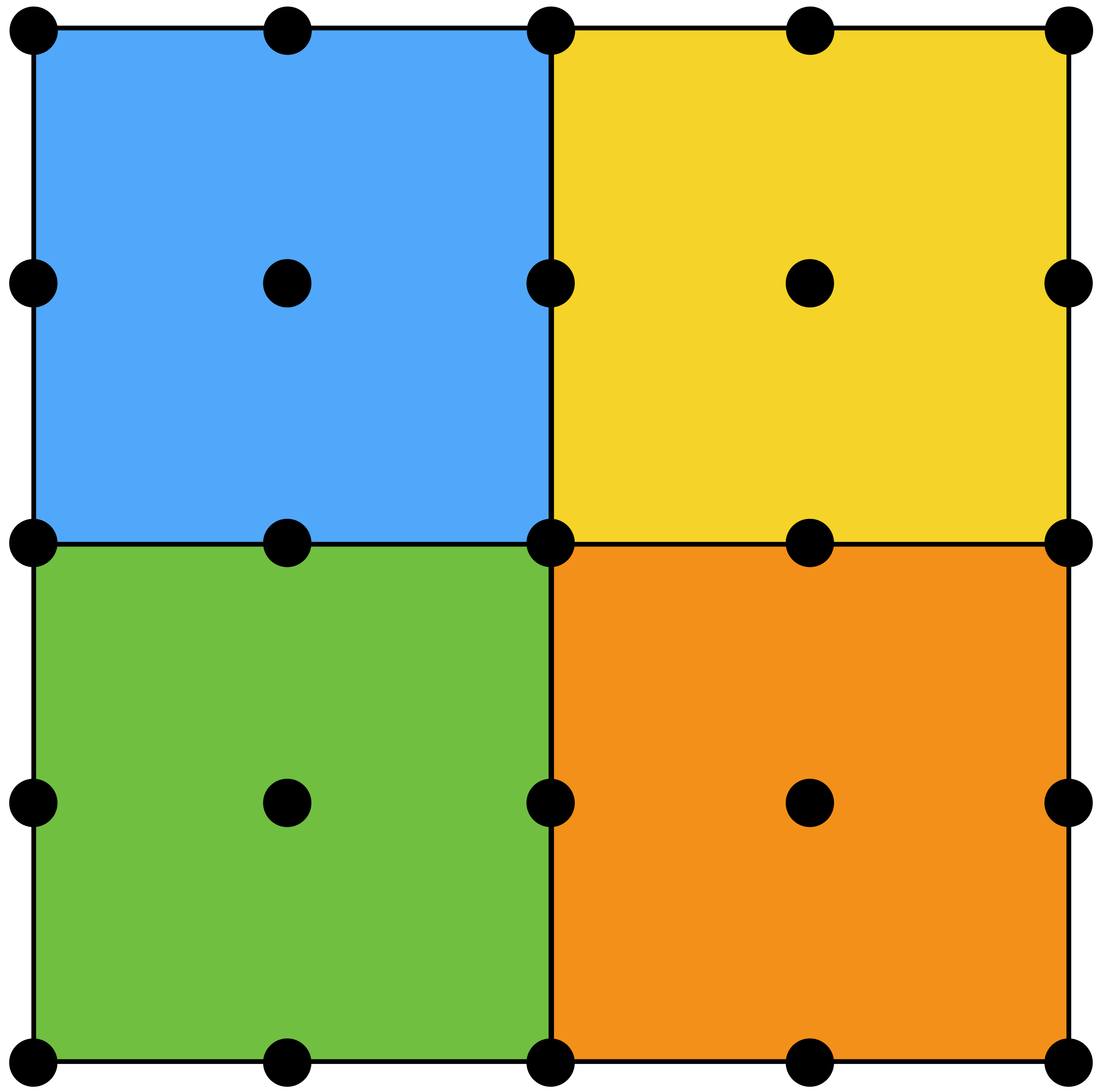
This works in essentially the same way for more complex indexing:

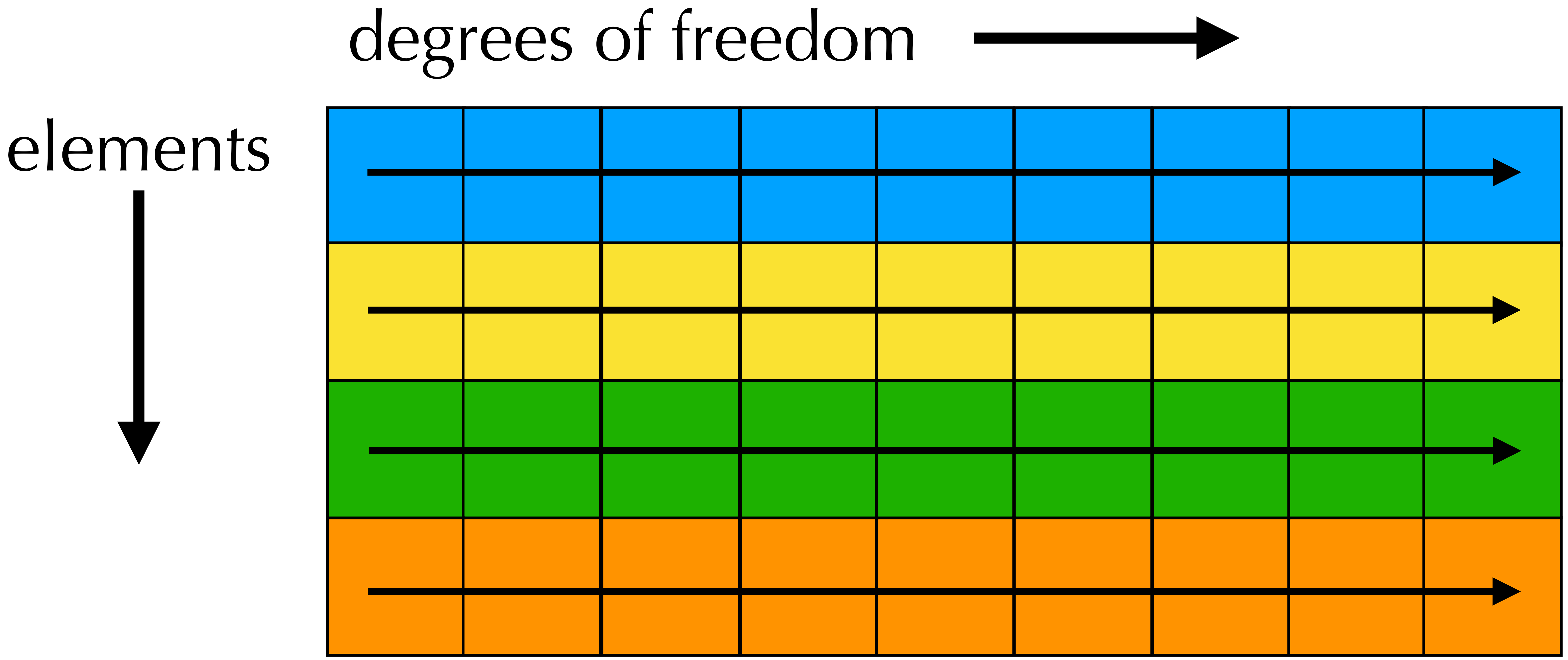
$$\sum_{p=0}^{P} \sum_{q=0}^{Q-p} \hat{u}_{pq} \phi^a_p(\xi_{1i}) \phi^b_{pq}(\xi_{2j}) = \sum_{p=0}^{P} \phi^a_p(\xi_{1i}) \left[\sum_{q=0}^{Q-p} \hat{u}_{pq} \phi^b_{pq}(\xi_{2j}) \right]$$

store this for each *p*

Data layout

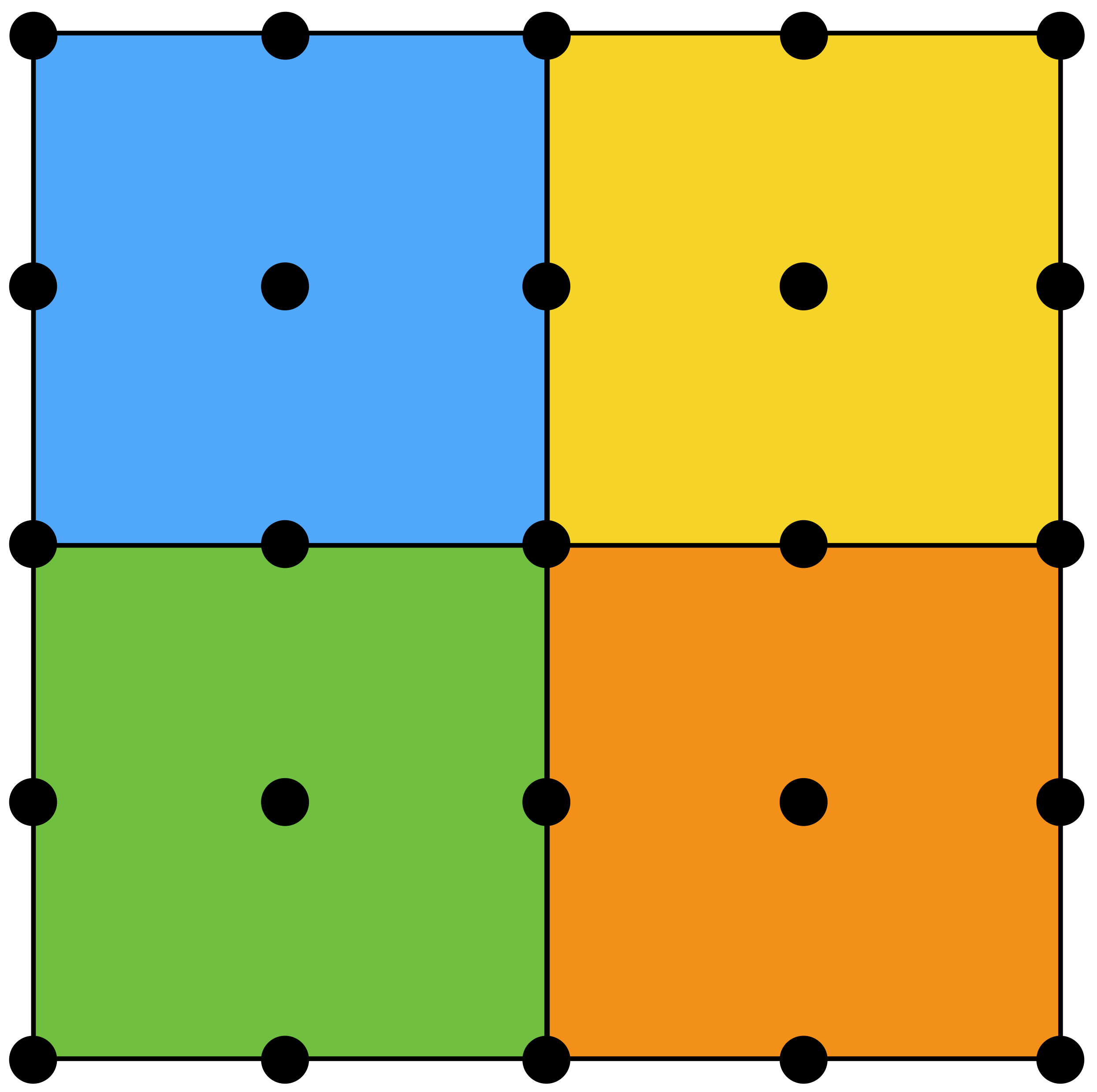
To exploit hardware, need to consider data layout:
natural to consider data element by element.

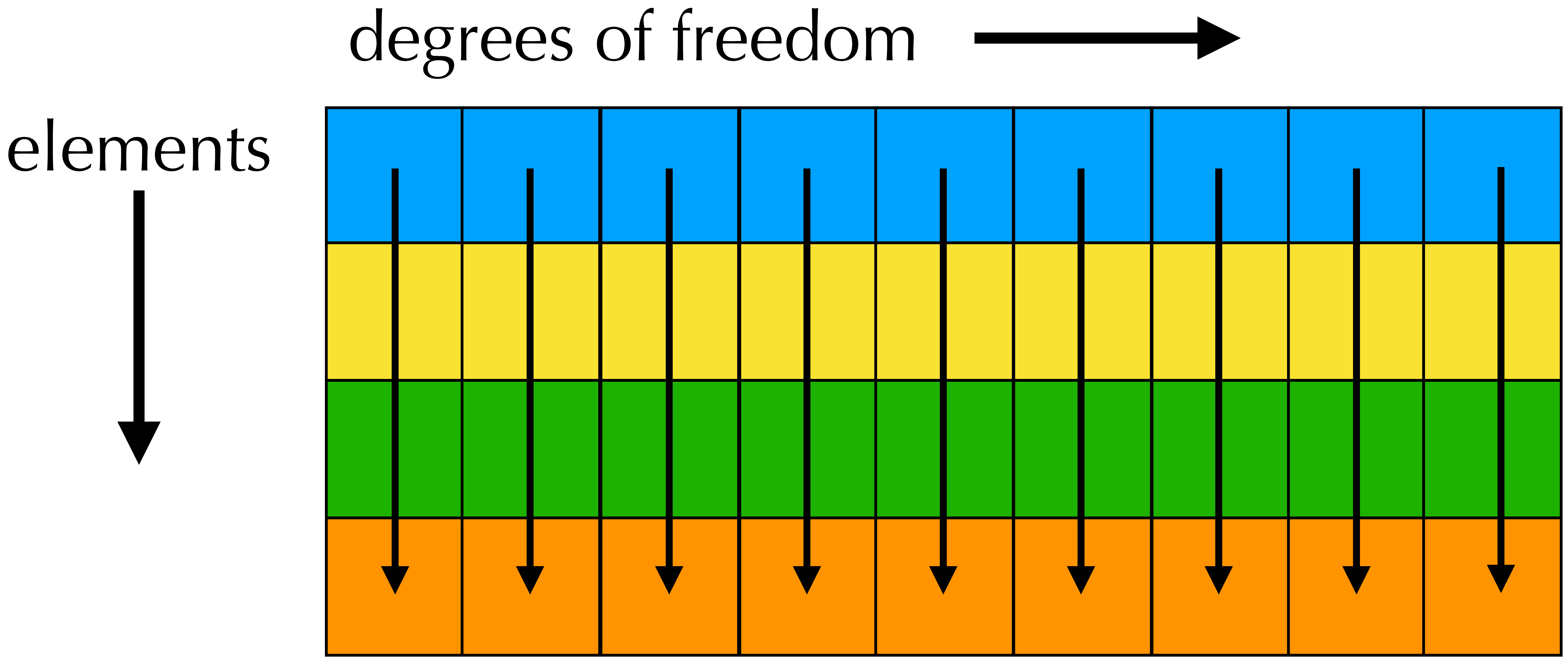




Data layout

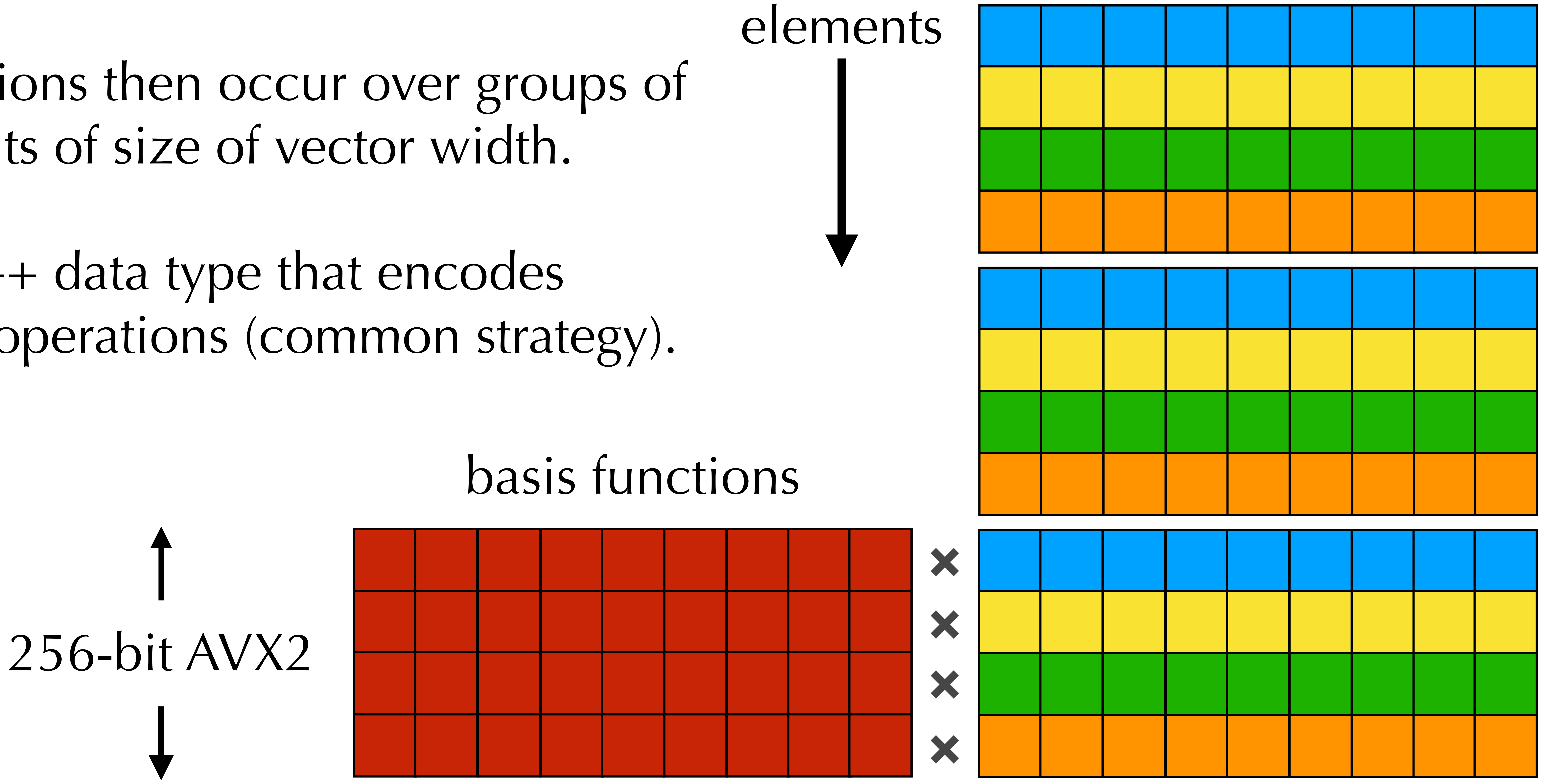
However, can exploit vectorisation by grouping DoFs by vector width.





Data layout

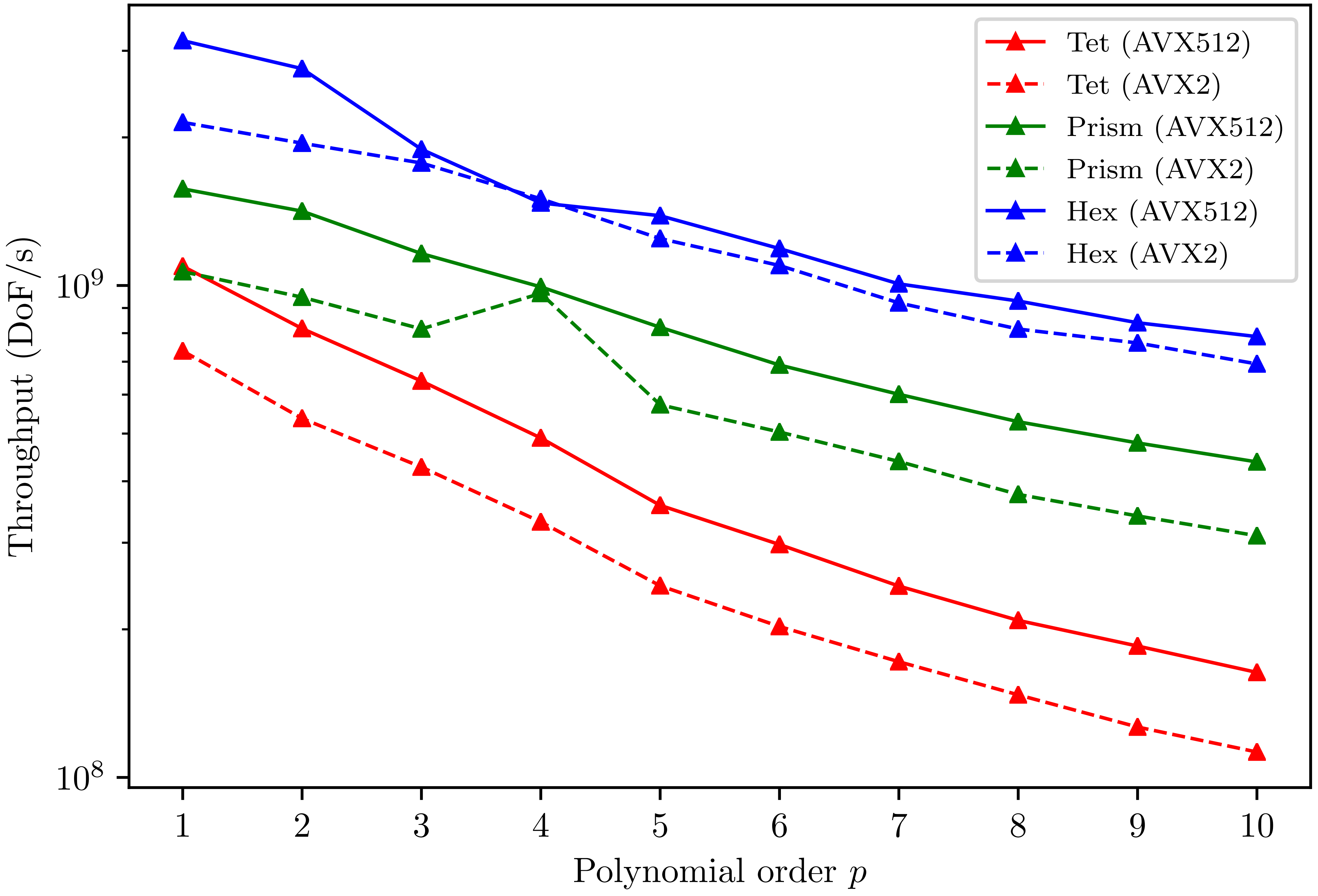
- Operations then occur over groups of elements of size of vector width.
- Use C++ data type that encodes vector operations (common strategy).



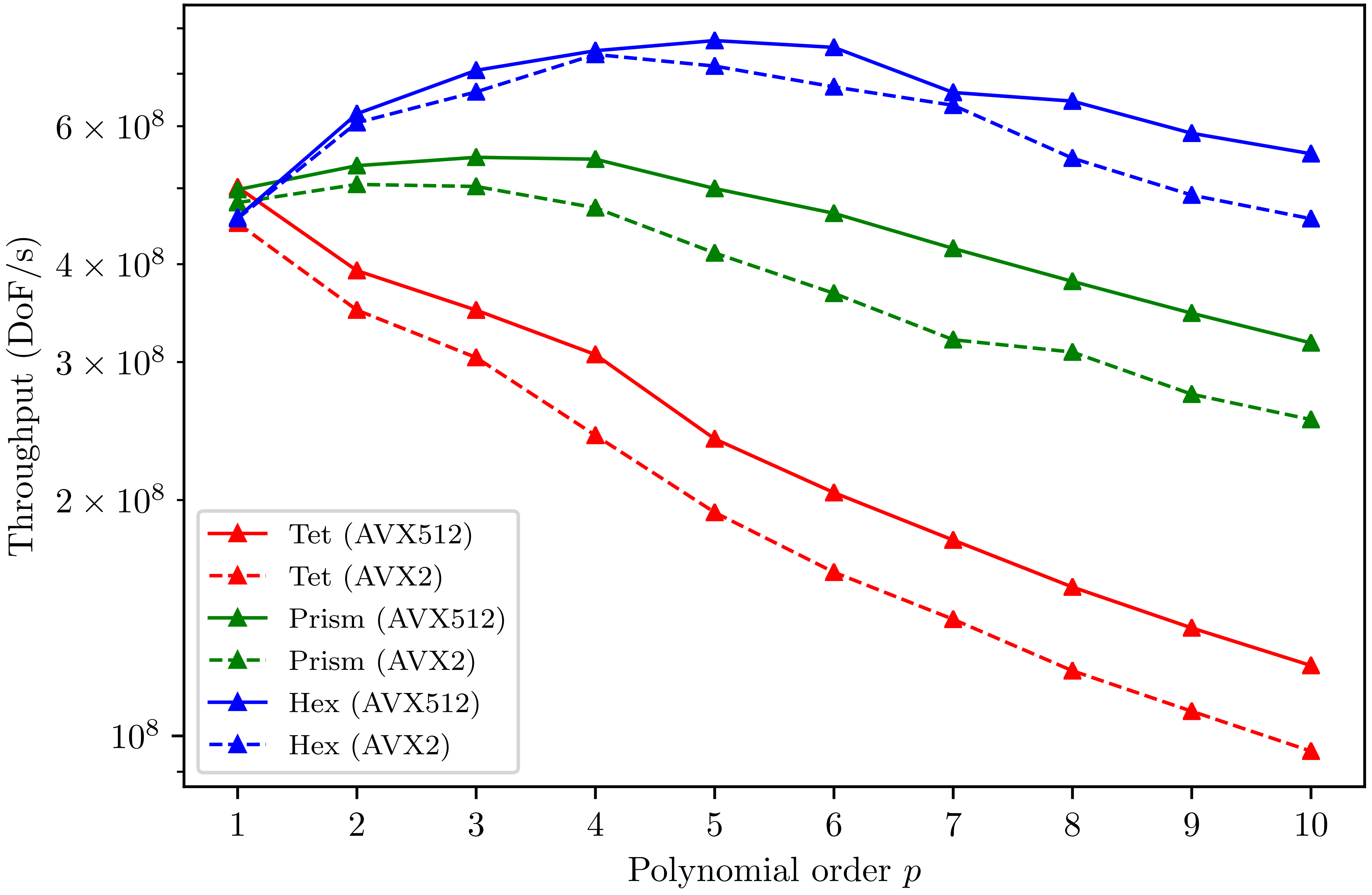
Assessing performance

- Various techniques used to assess kernel performance:
  - **Throughput**: number of local DoF/s processed, for a mesh whose sizes exceeds available cache.
  - GFLOP/s gives some indication of capabilities, provided we are not memory-bound.
  - Better is **roofline analysis**: where do we sit in terms of memory bandwidth to arithmetic intensity?
- Note all results for local elemental operation evaluation only.

Throughput (AVX512/AVX2, Skylake)

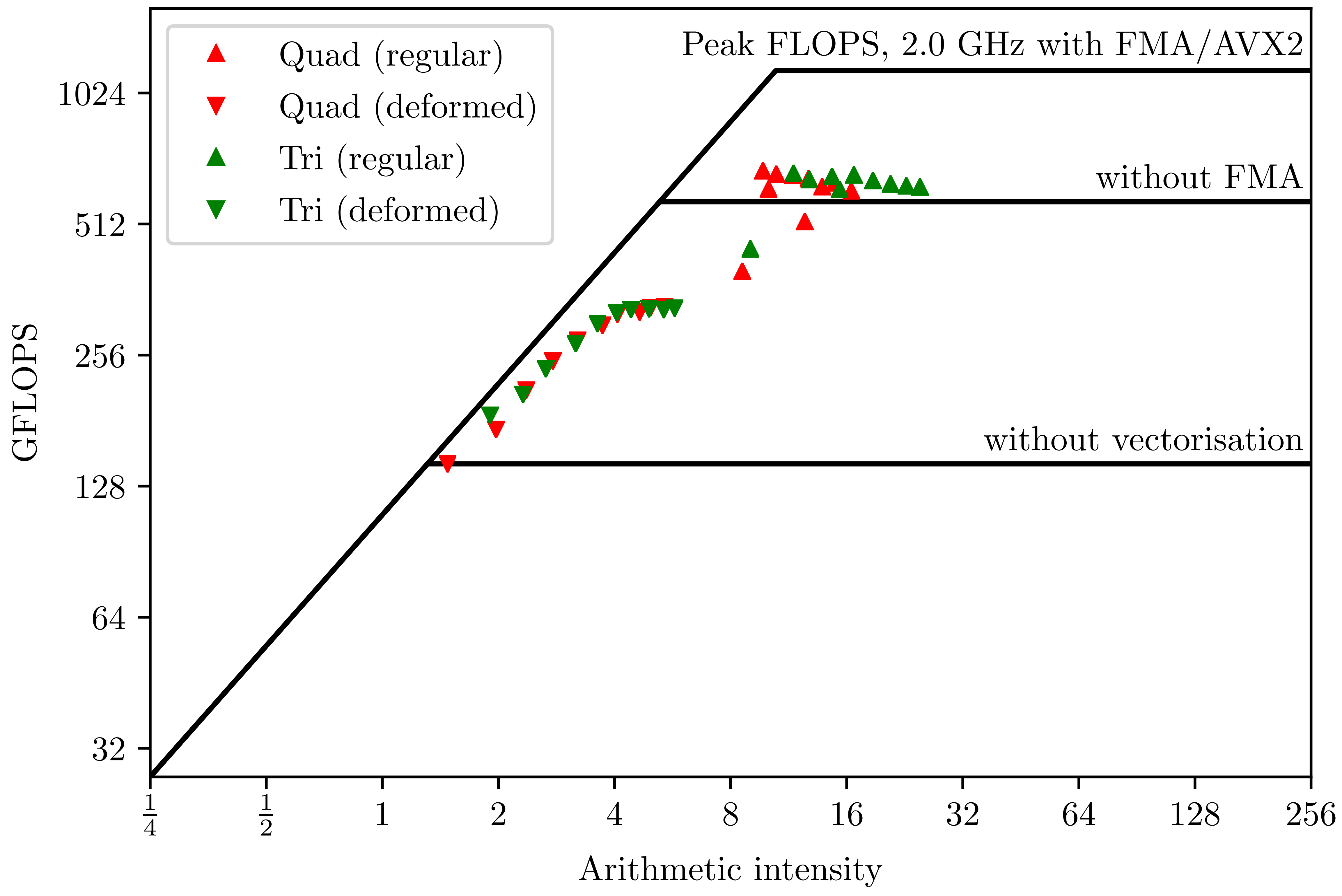


3D: ‘Regular’ elements

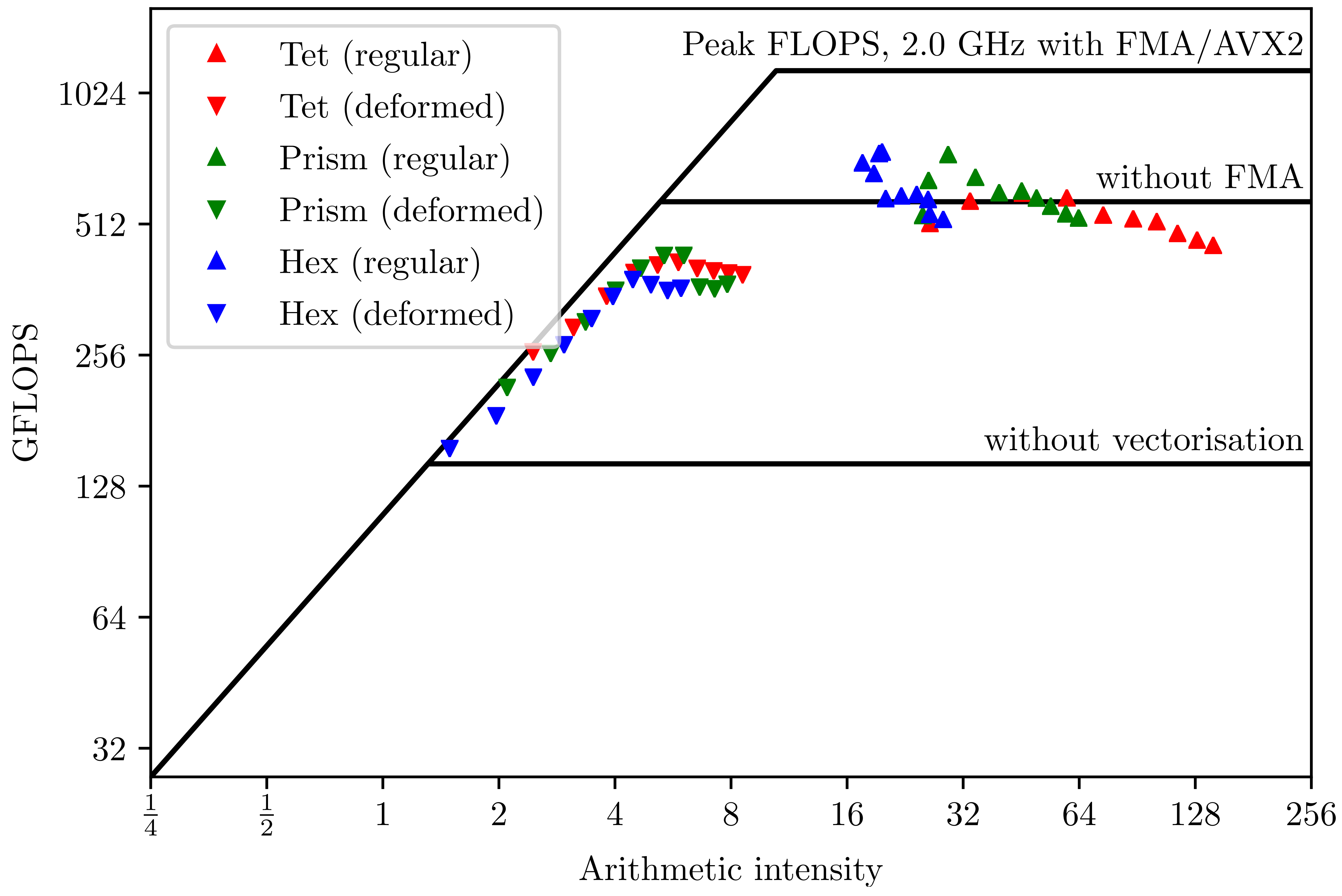


3D: ‘Deformed’ elements

Roofline results



2D: Quads, triangles



3D: Hexahedra, prisms, tetrahedra

Use of ~50-70% peak FLOPS for regular elements

Challenge 3: implementation effort

- High-order methods have potential to bring some nice numerical and computational benefits to bear on complex problems.
- Offer high(er) fidelity at equivalent or lower costs, as they have good implementation characteristics.
- However, one of the main barriers to using high-order methods is that they are **difficult to implement**.

Nektar++

spectral/hp element framework

Nektar++

spectral/hp element framework

- Nektar++ is an **open-source MIT-licensed framework** for high-order methods.
- **Arbitrary order** curvilinear meshes to support complex geometries in a wide range of application area**s** including incompressible/compressible fluids.
- **Wide range of discretisation choices:** CG/DG/HDG, Fourier, modal/nodal expansions, 1/2/3D, embedded manifolds.
- **Parallel MPI support**, scalable to many thousands of cores.
- Modern **C++11 API**, extensive testing, CI & distributed source control.

Development team

Mike Kirby

Spencer Sherwin

Chris Cantwell

David Moxey

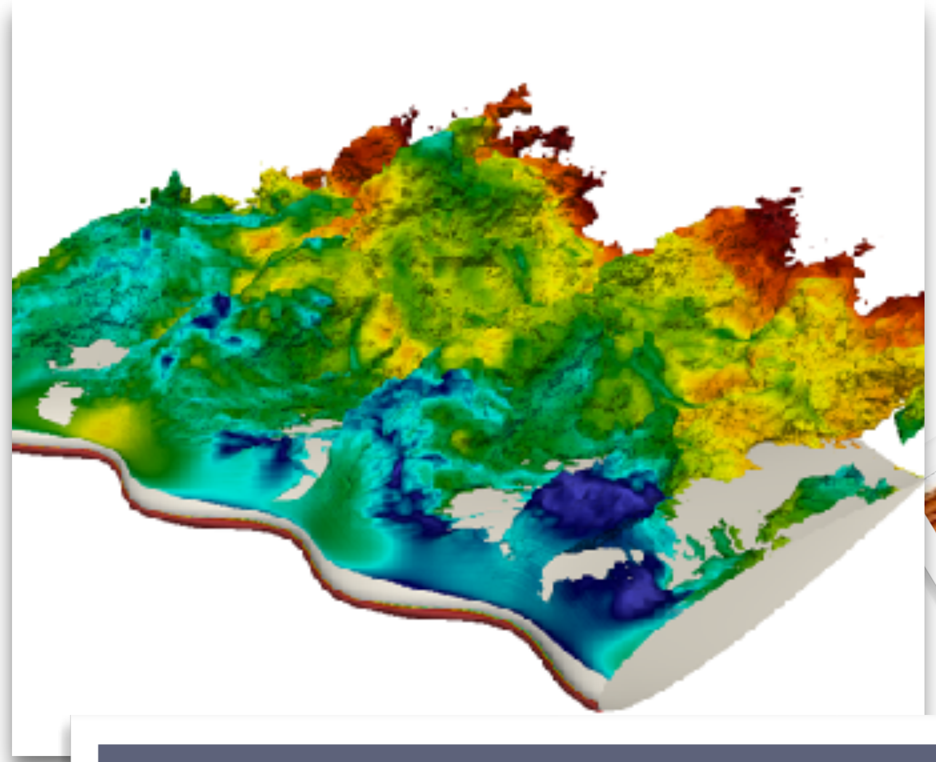
THE UNIVERSITY OF UTAH

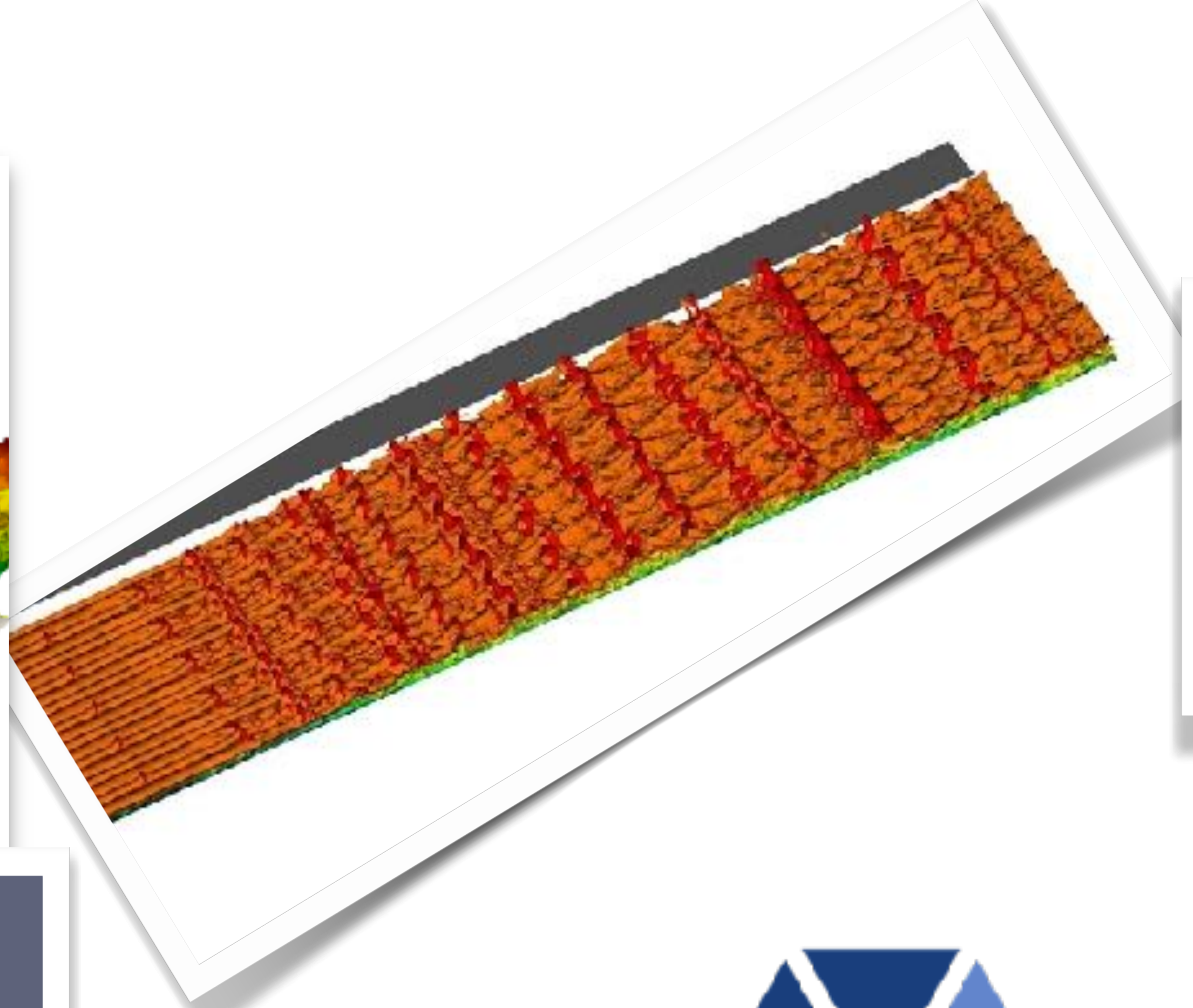
Imperial College London

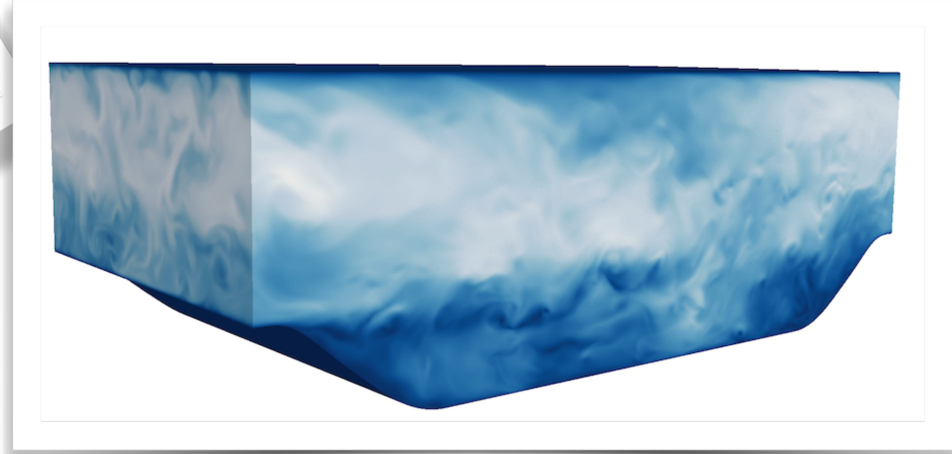
UNIVERSITY OF EXETER

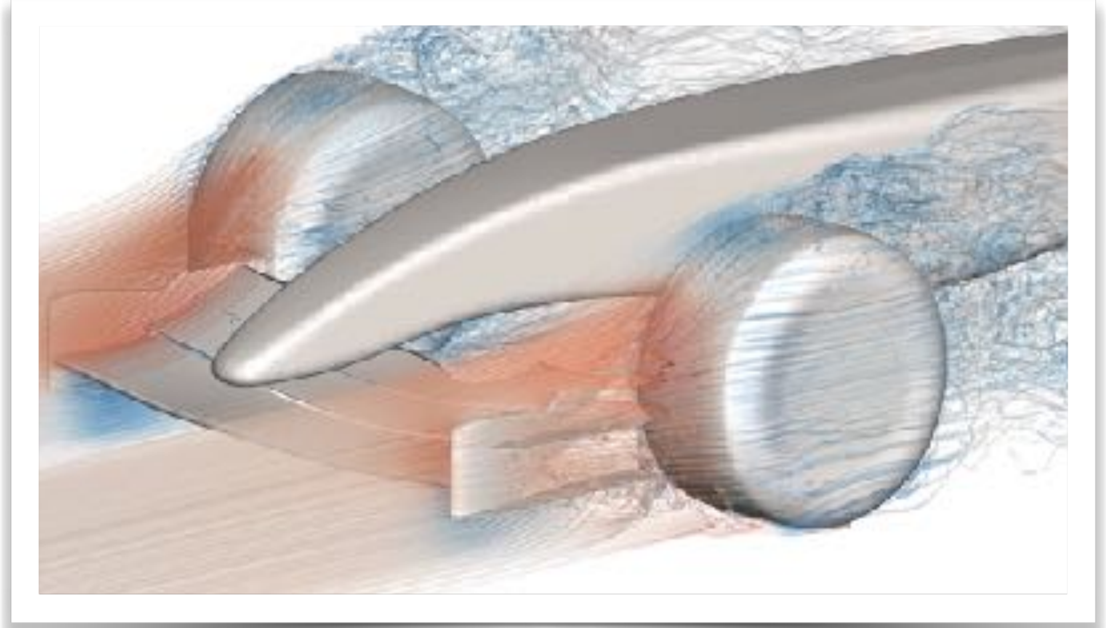
- **Project coordinators:** Joaquim Peiró, Gianmarco Mengaldo
- **Senior developers:** Kilian Lackhove, Douglas Serson, Giacomo Castiglioni

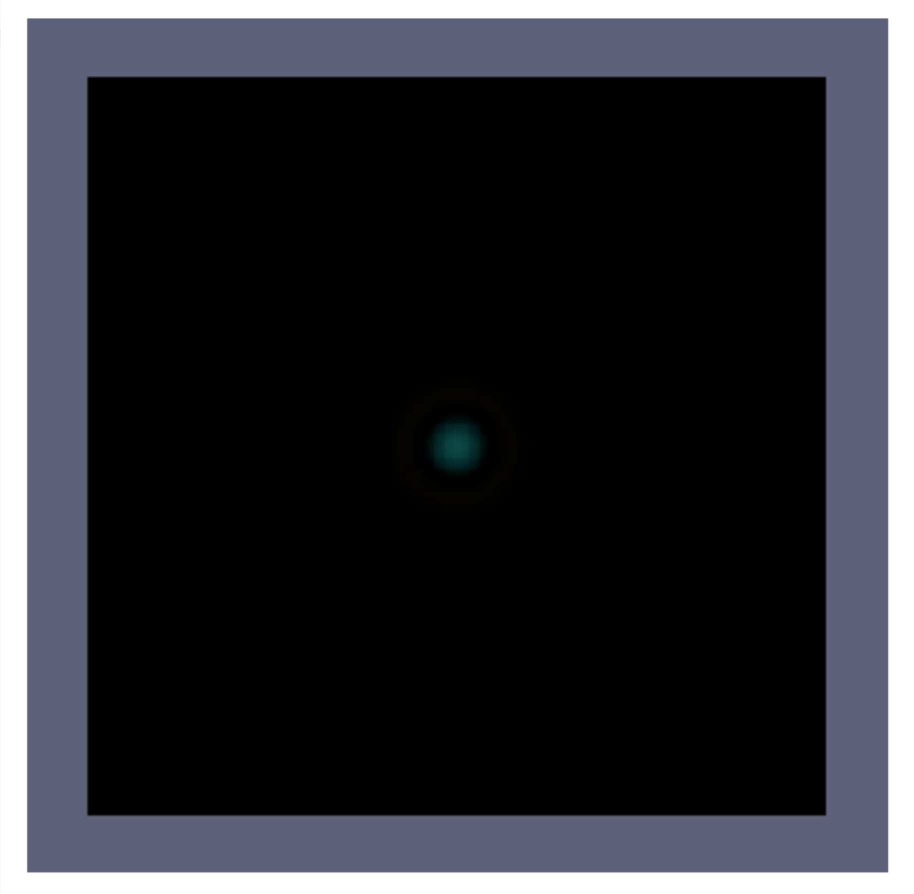
Some application areas



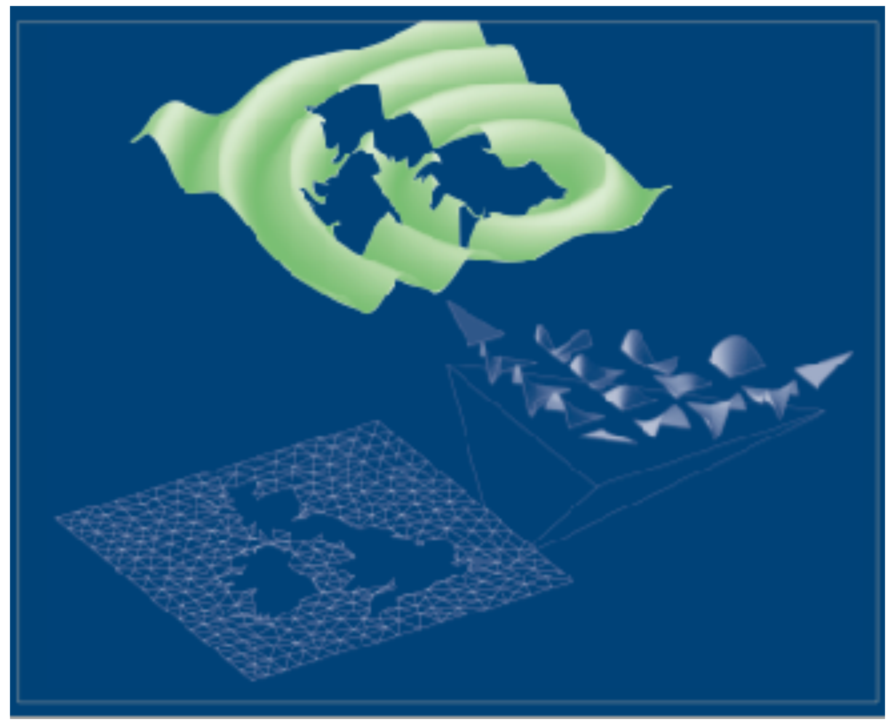


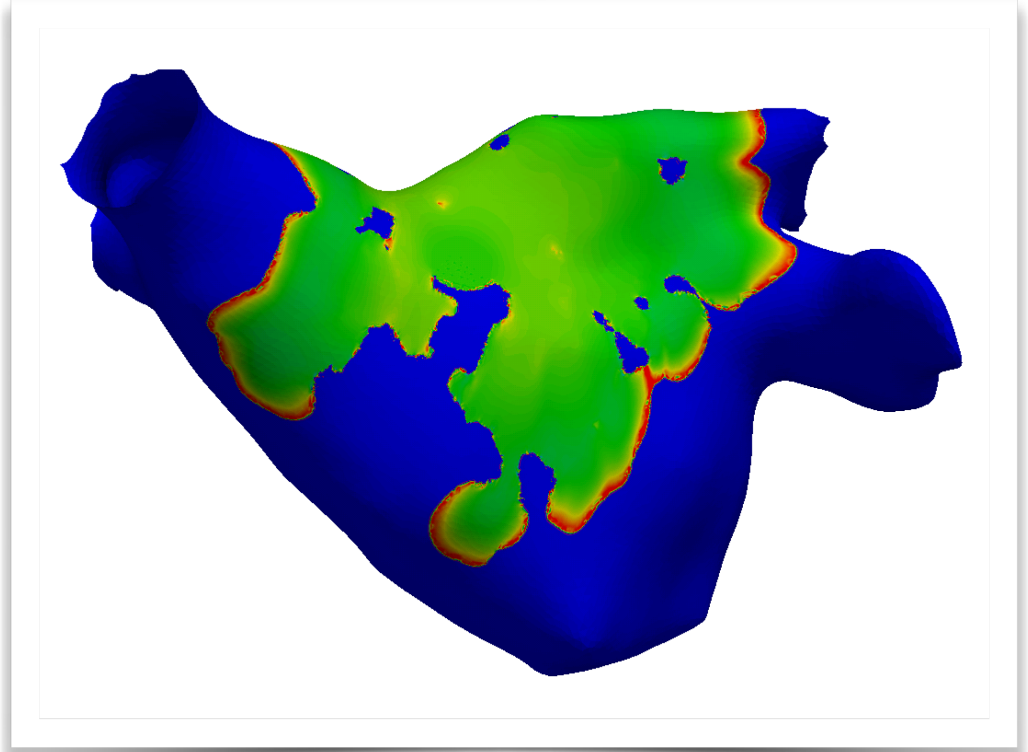


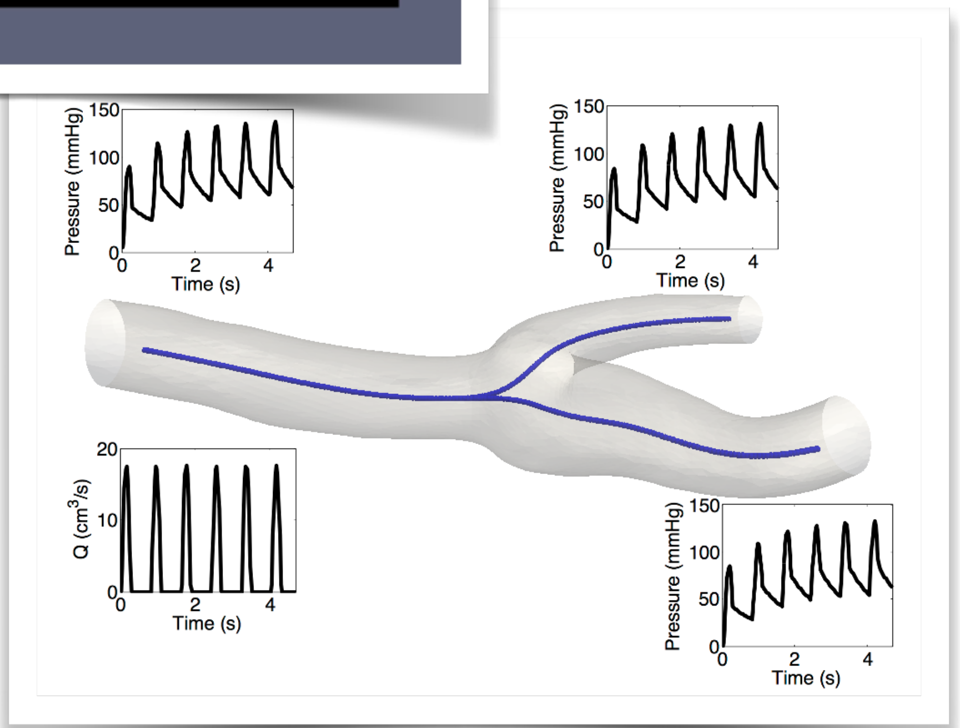


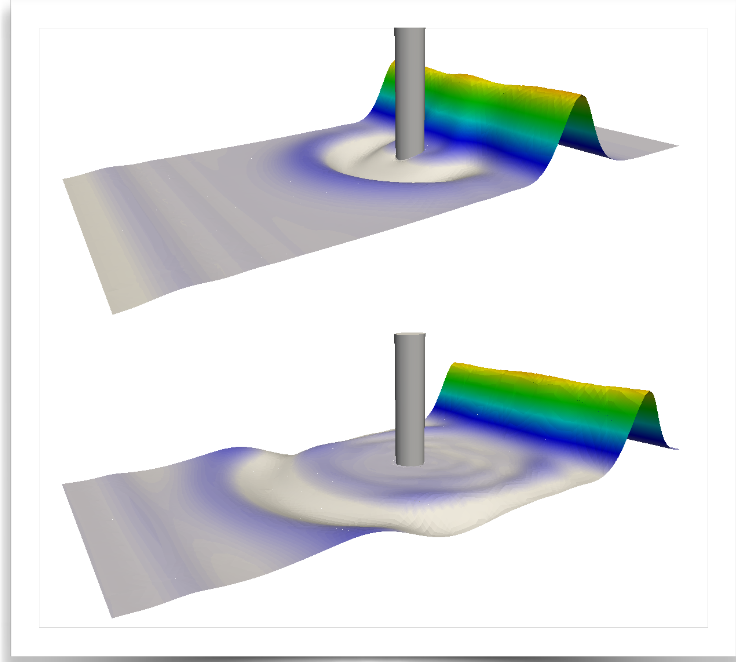


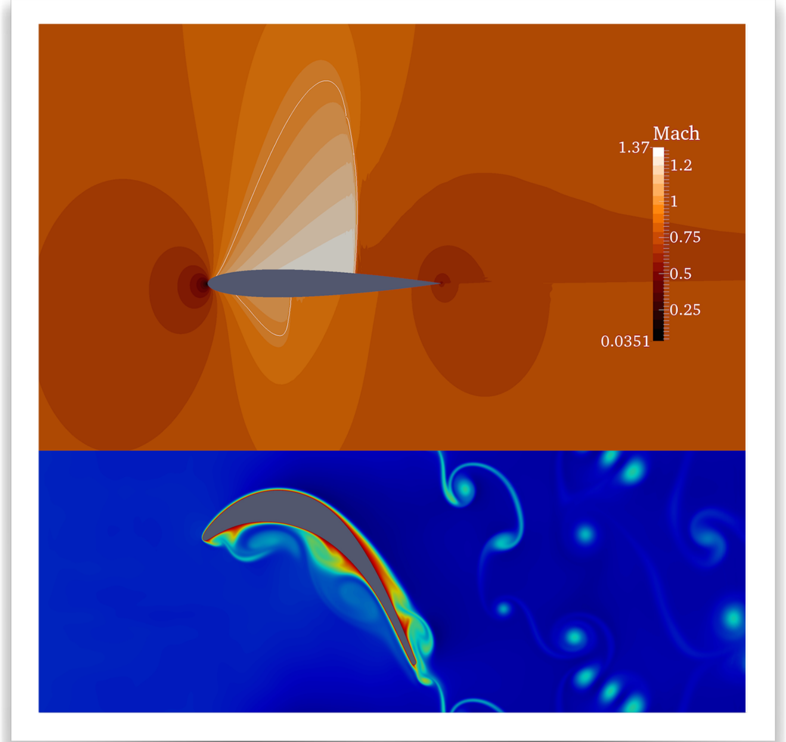
www.nektar.info

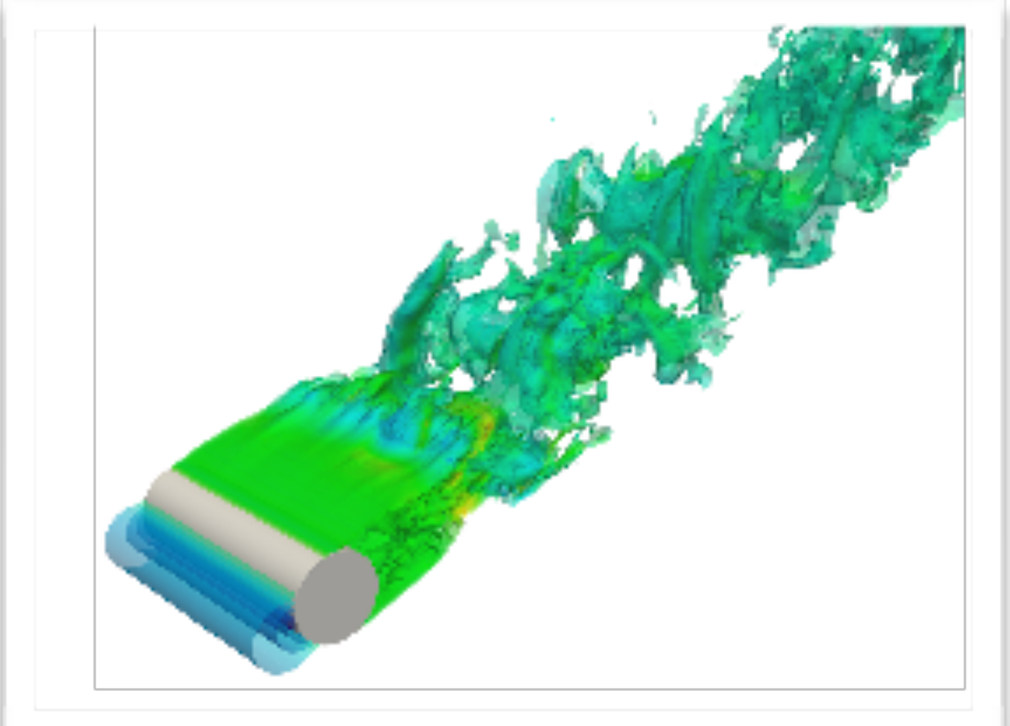




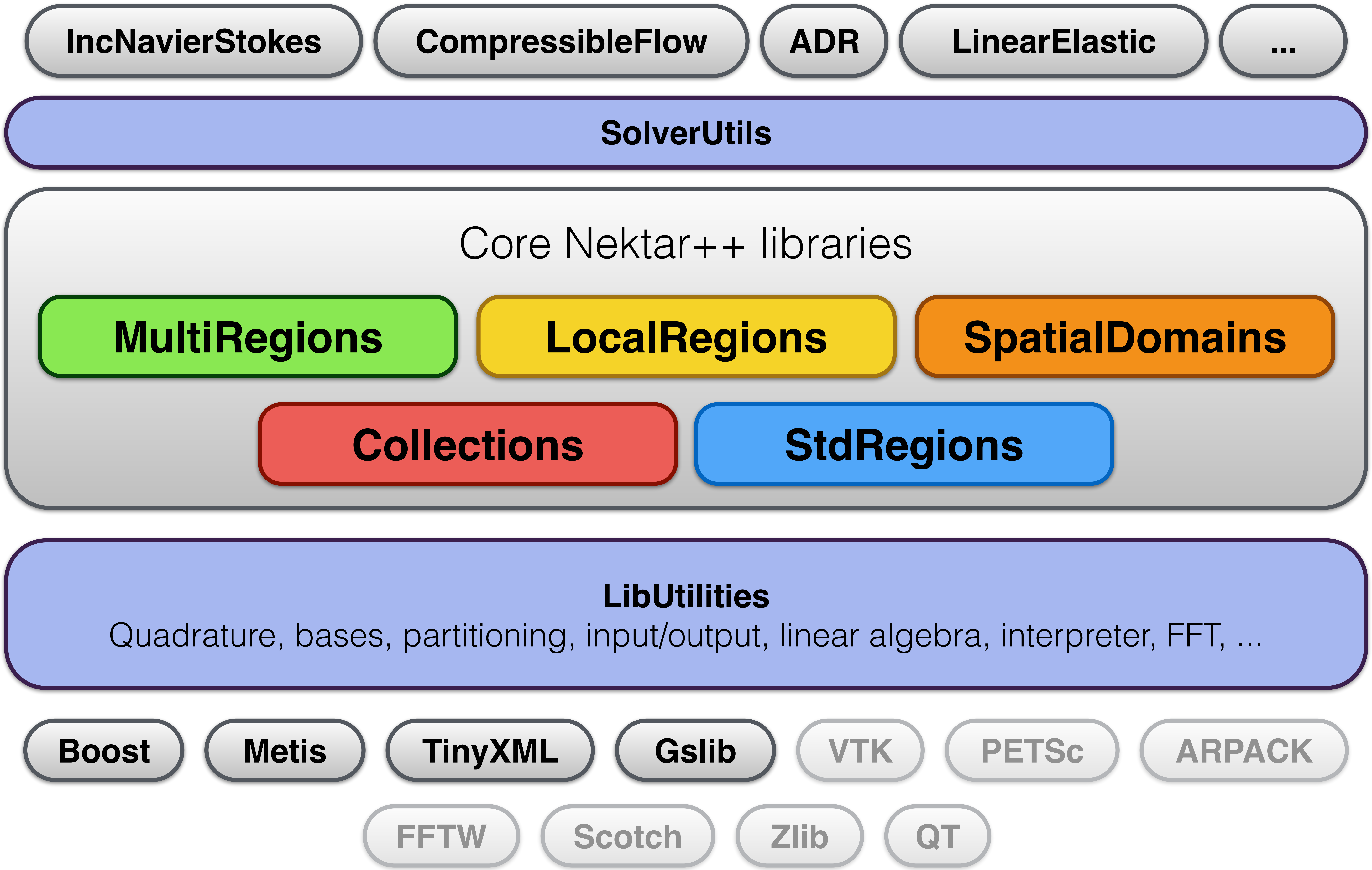




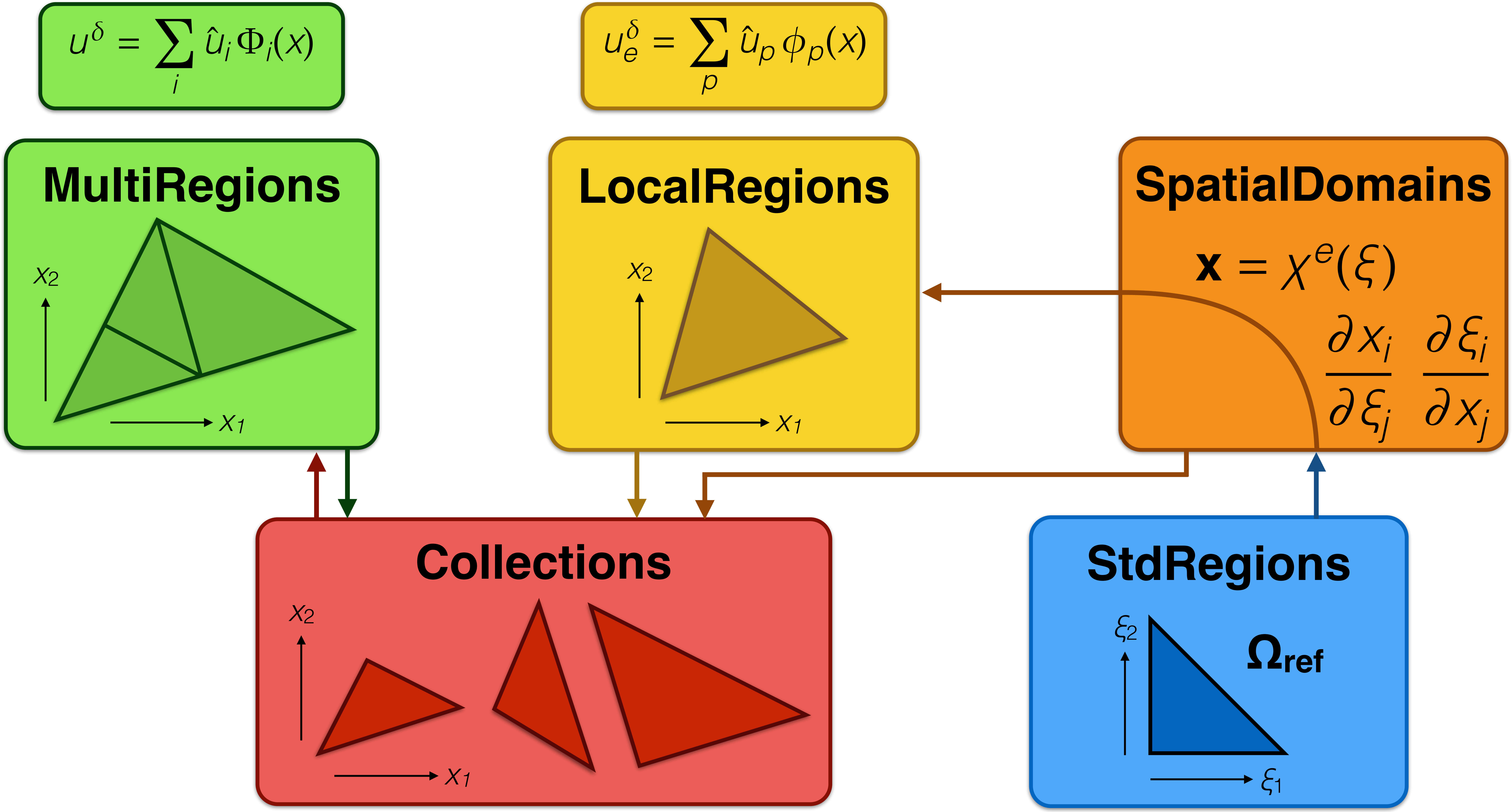




Framework design



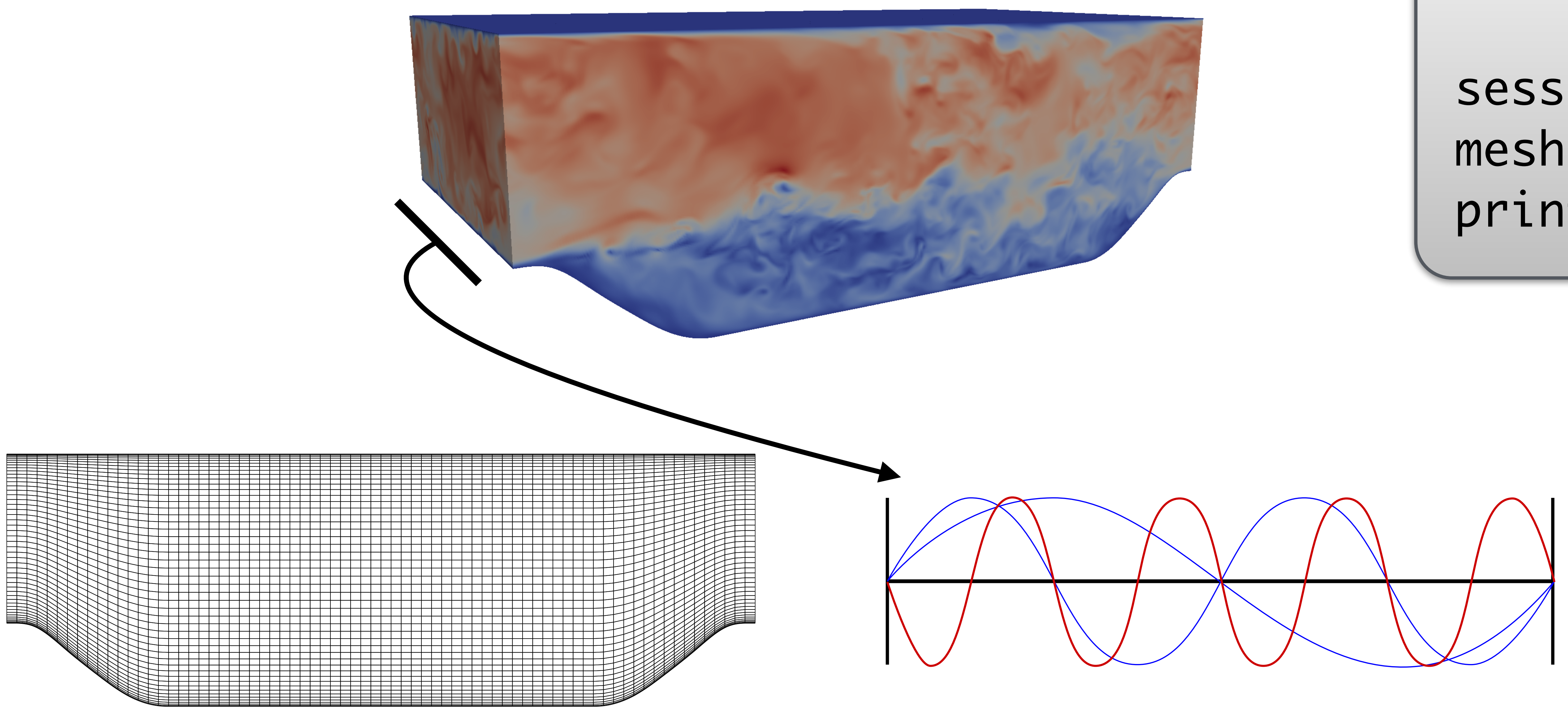
Framework design



Highlights from v5

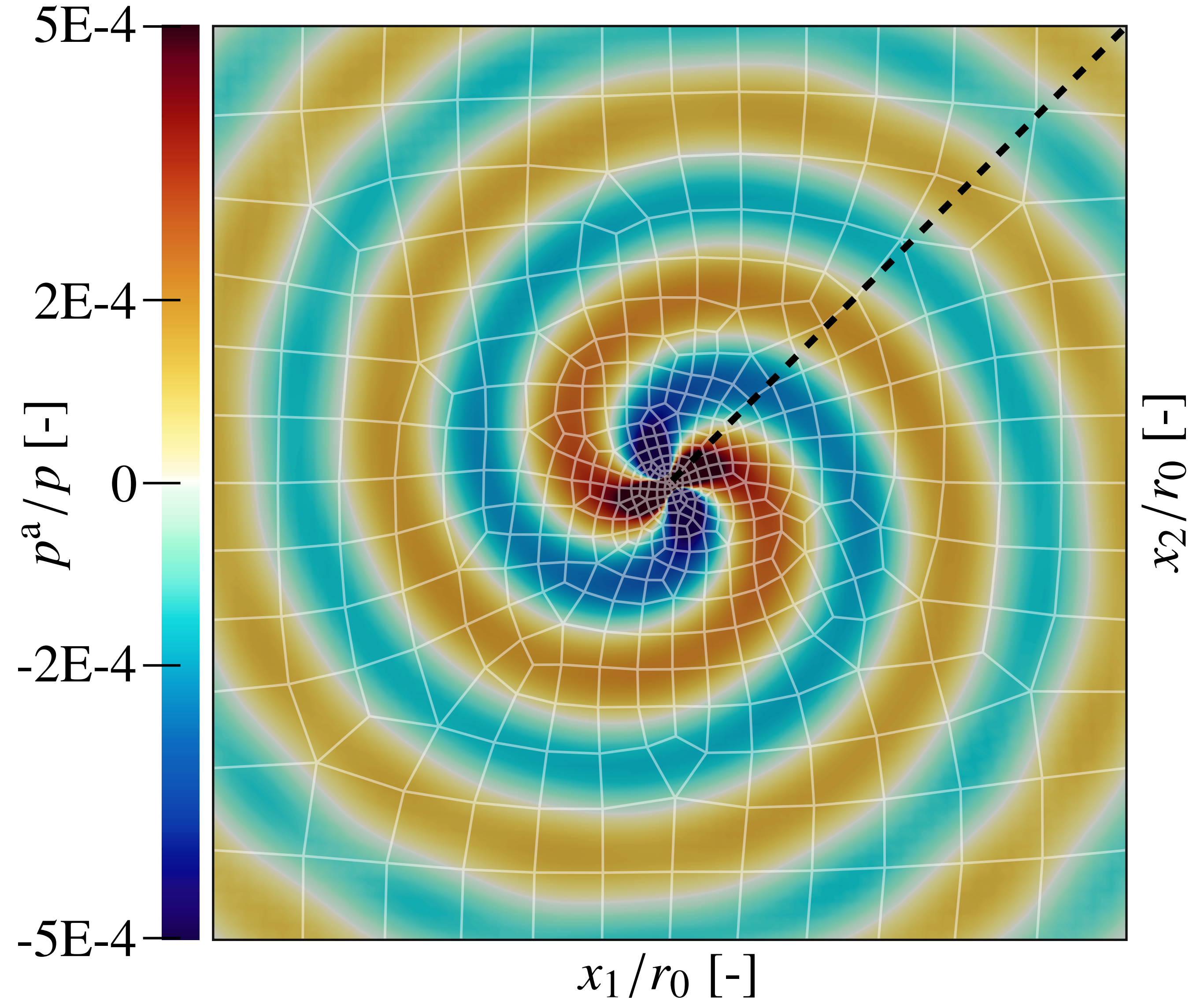
```python
from NekPy.LibUtilities import SessionReader
from NekPy.SpatialDomains import MeshGraph

session = SessionReader.CreateInstance(sys.argv)
mesh    = MeshGraph.Read(session)
print(mesh.GetMeshDimension())
```

Python interface



2D Spectral element mesh + 1D Fourier expansion

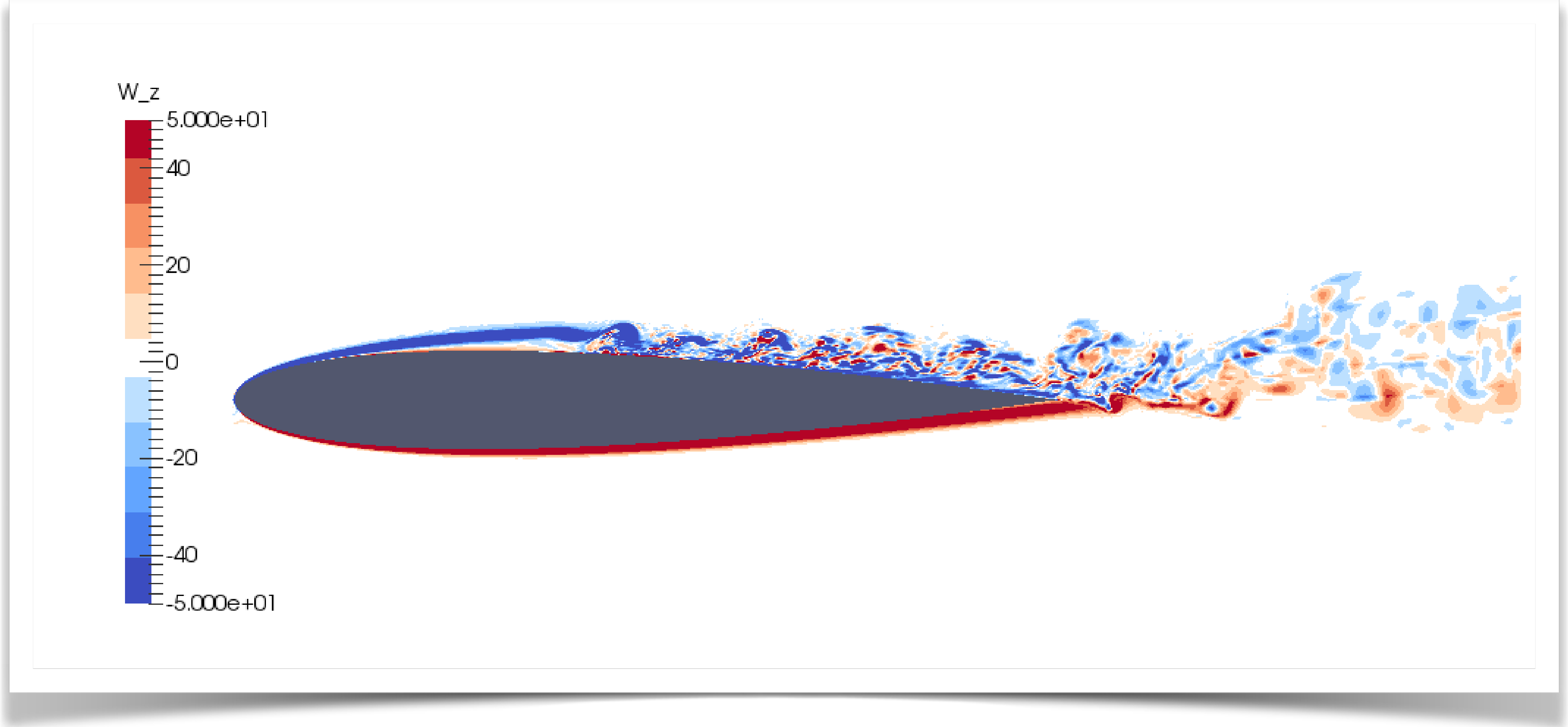
Hybrid discretisation

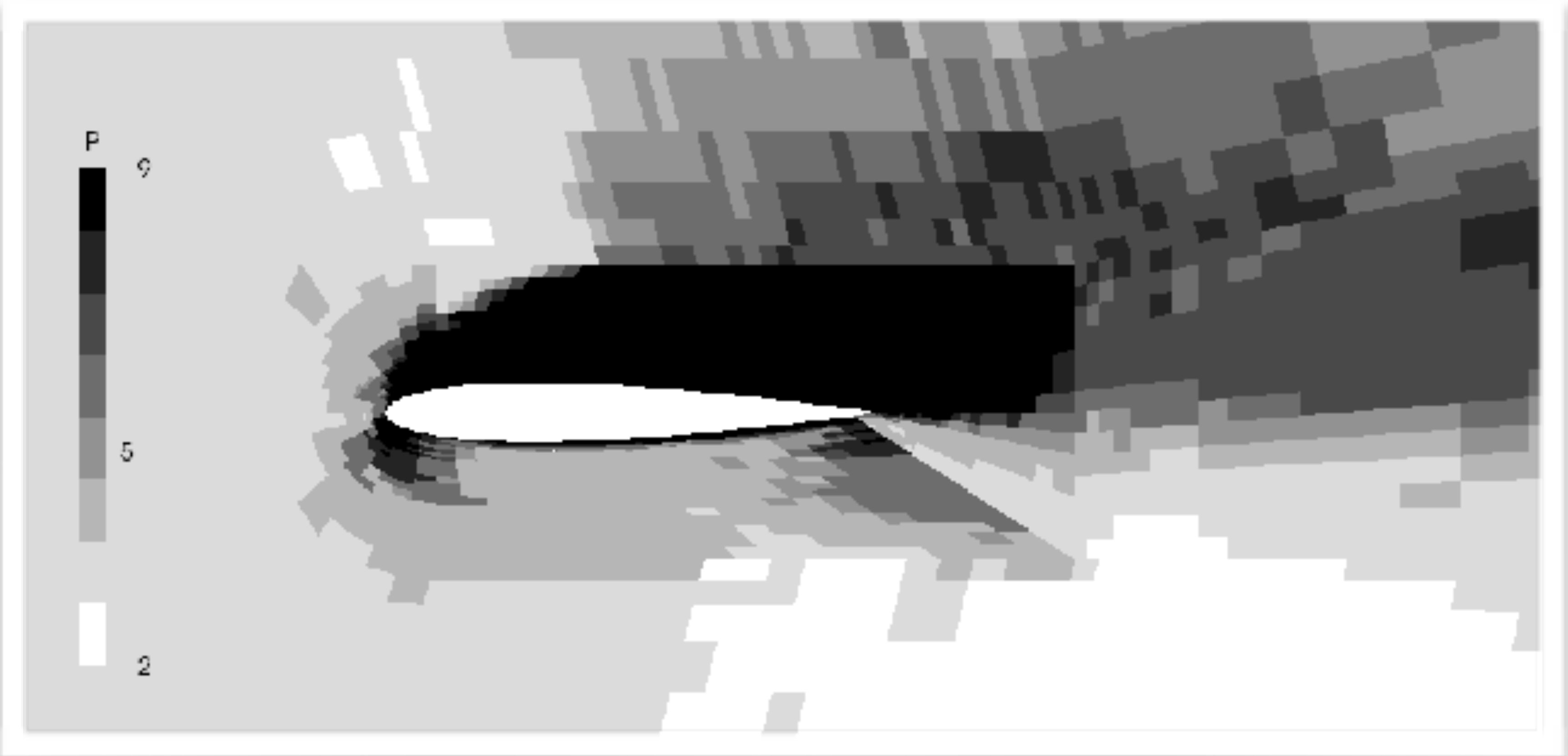


Acoustic solver

Moxey, Cantwell et al, arXiv 1906.03489

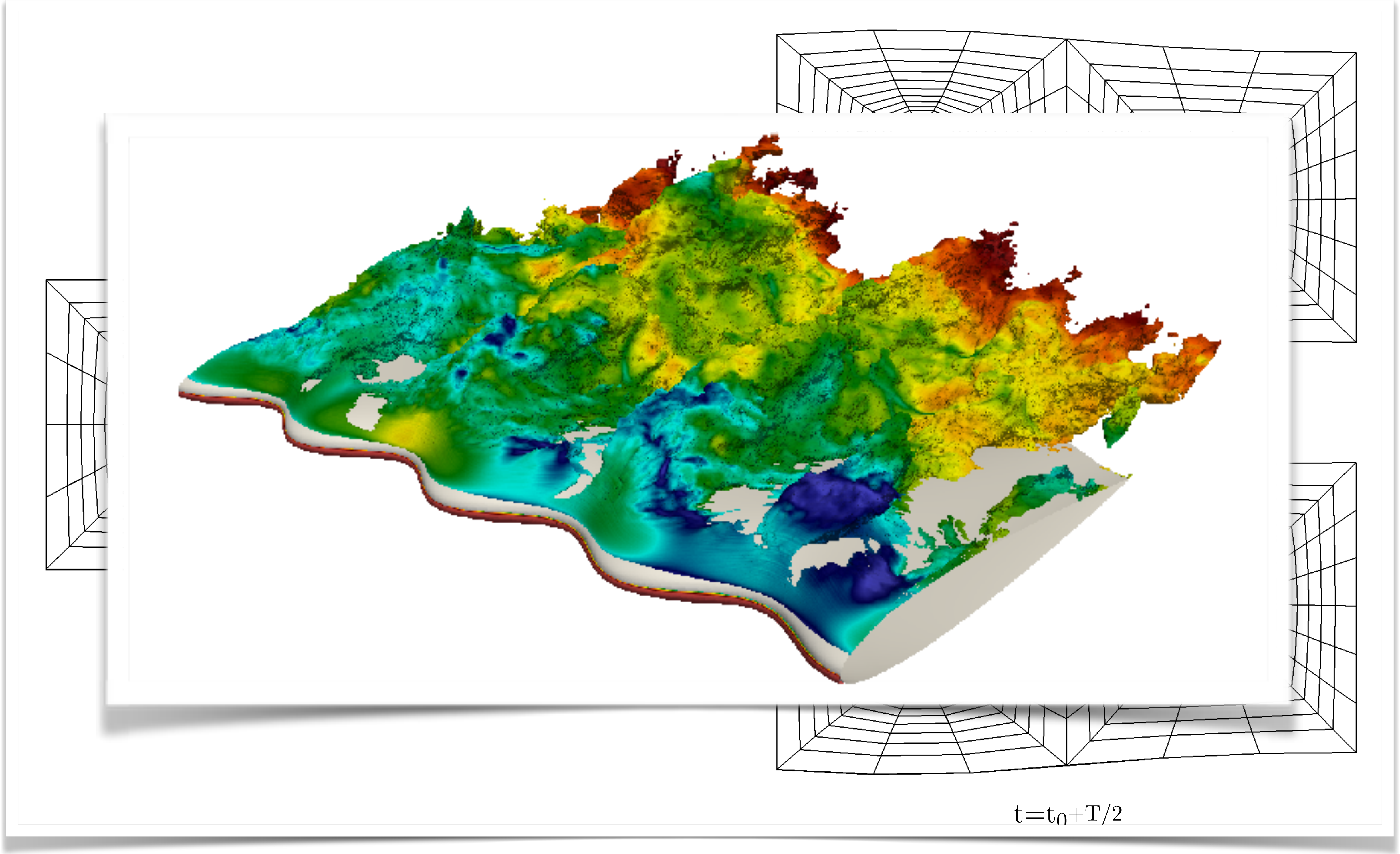
Highlights from v5





Spatially varying polynomial orders

D. Moxey et al, Spectral and High Order Methods for Partial Differential Equations ICOSAHOM 2016, pp. 63–79



Coordinate mapping

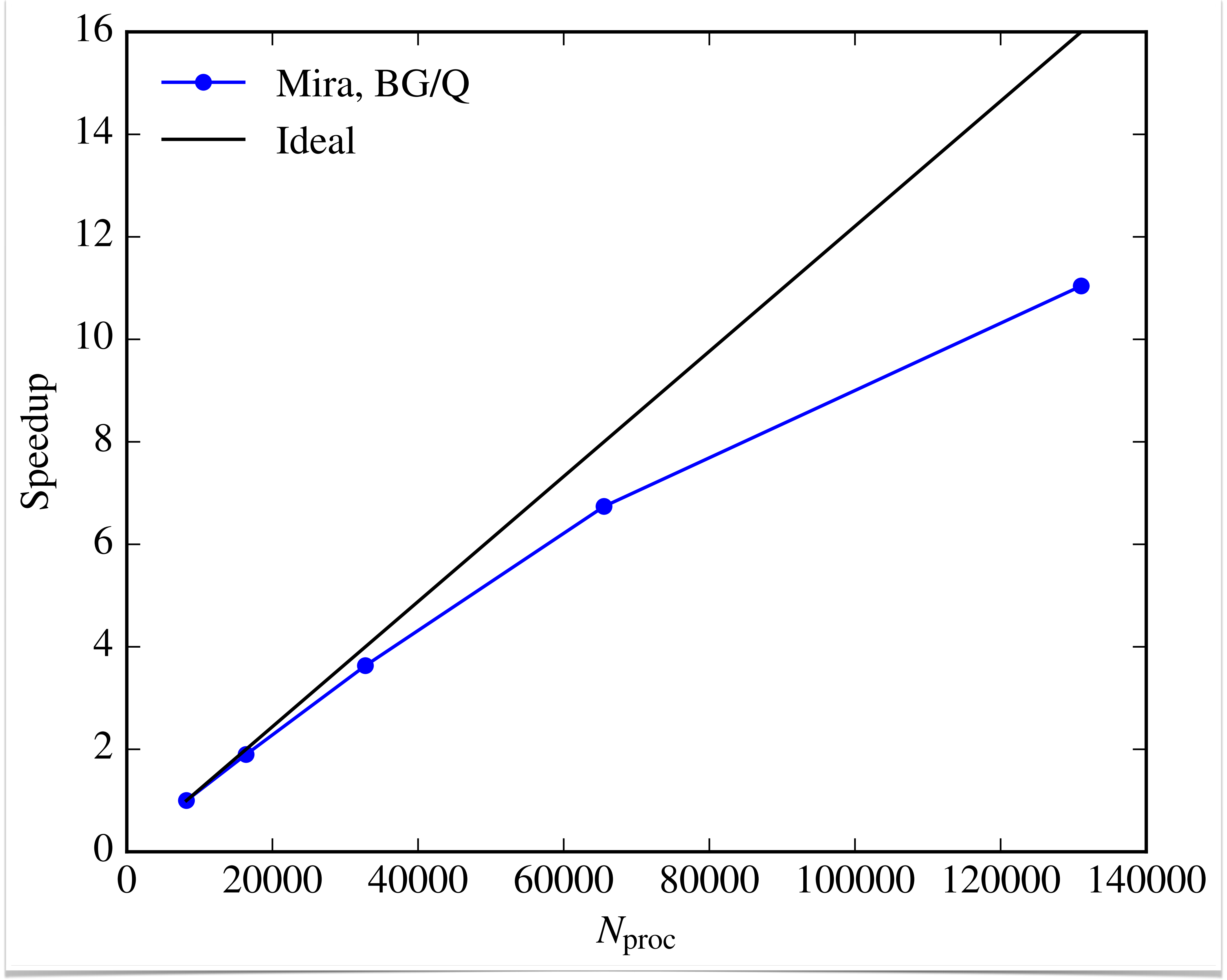
D. Serson, J. Meneghini, and S. Sherwin, J. Comp. Phys. **316**, 243-254 (2016)

High-order fluid simulations

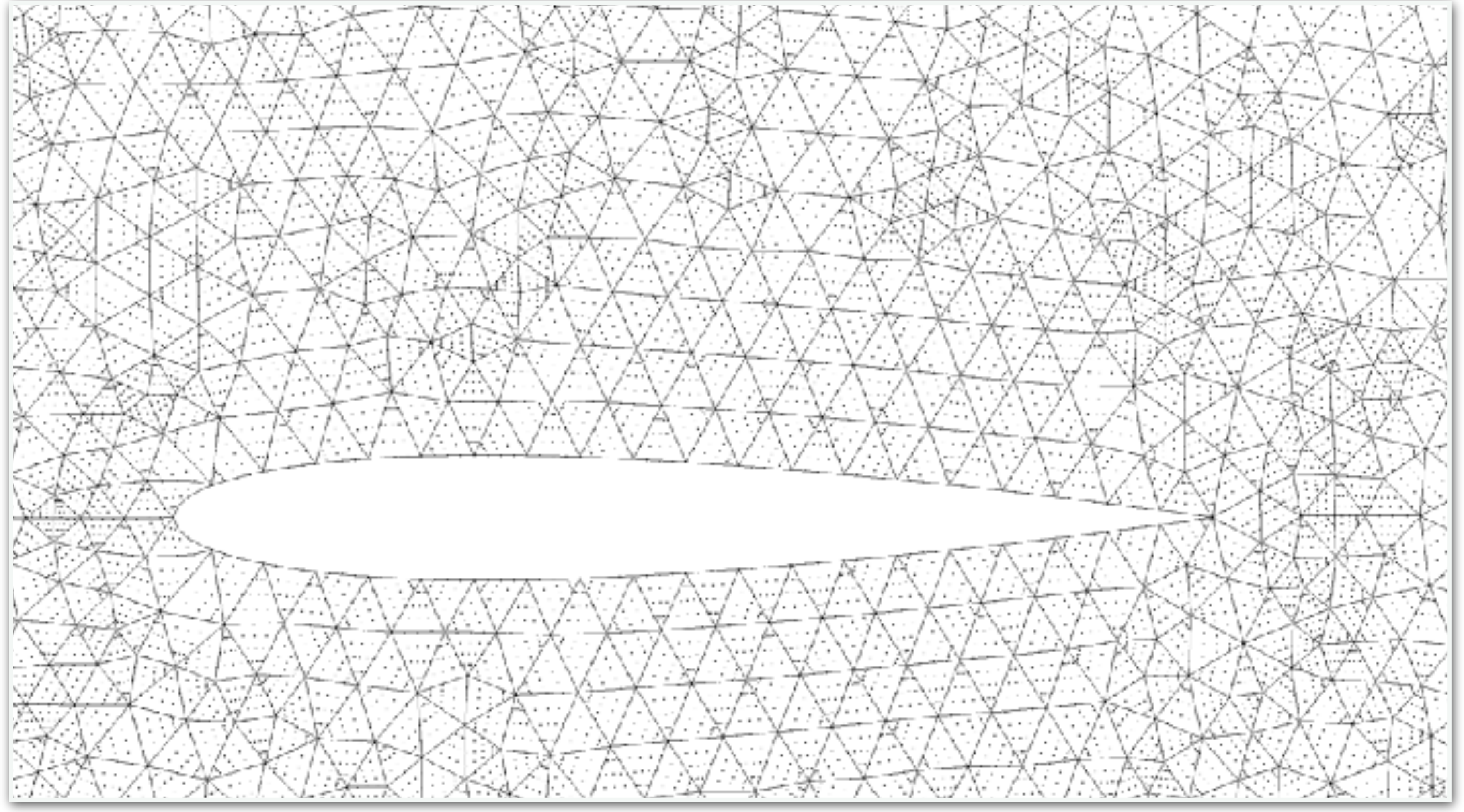
- Heavy development of both **compressible** and **incompressible flow** solvers and, with a particular focus on **high-fidelity** simulations.
- Consider inherently **unsteady flows**: investigate use of **implicit LES**.
- Our message: still computationally expensive & requires HPC, but **should not be prohibitive** and **should scale** with high-order simulations.

Solving at scale

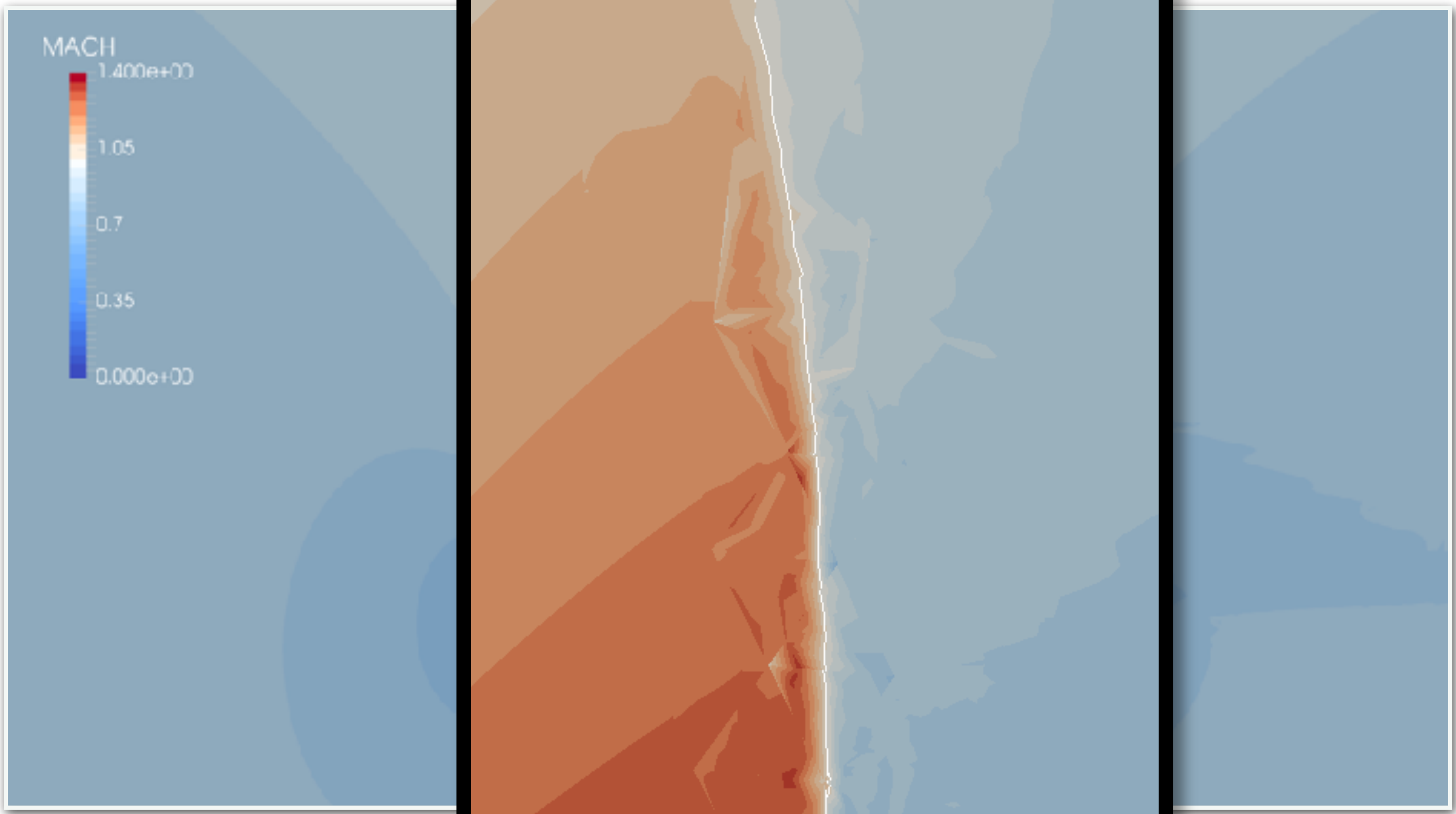
- Relying on HPC means we need efficient and scalable linear solvers.
- Mesh is decomposed across processors; local dense matrices formed for each element, communication with `gslib`.
- Core of the code scales well on Mira: test case of a ~5m element F1 geometry at fifth order.
- However still some work to do on scalable preconditioning!



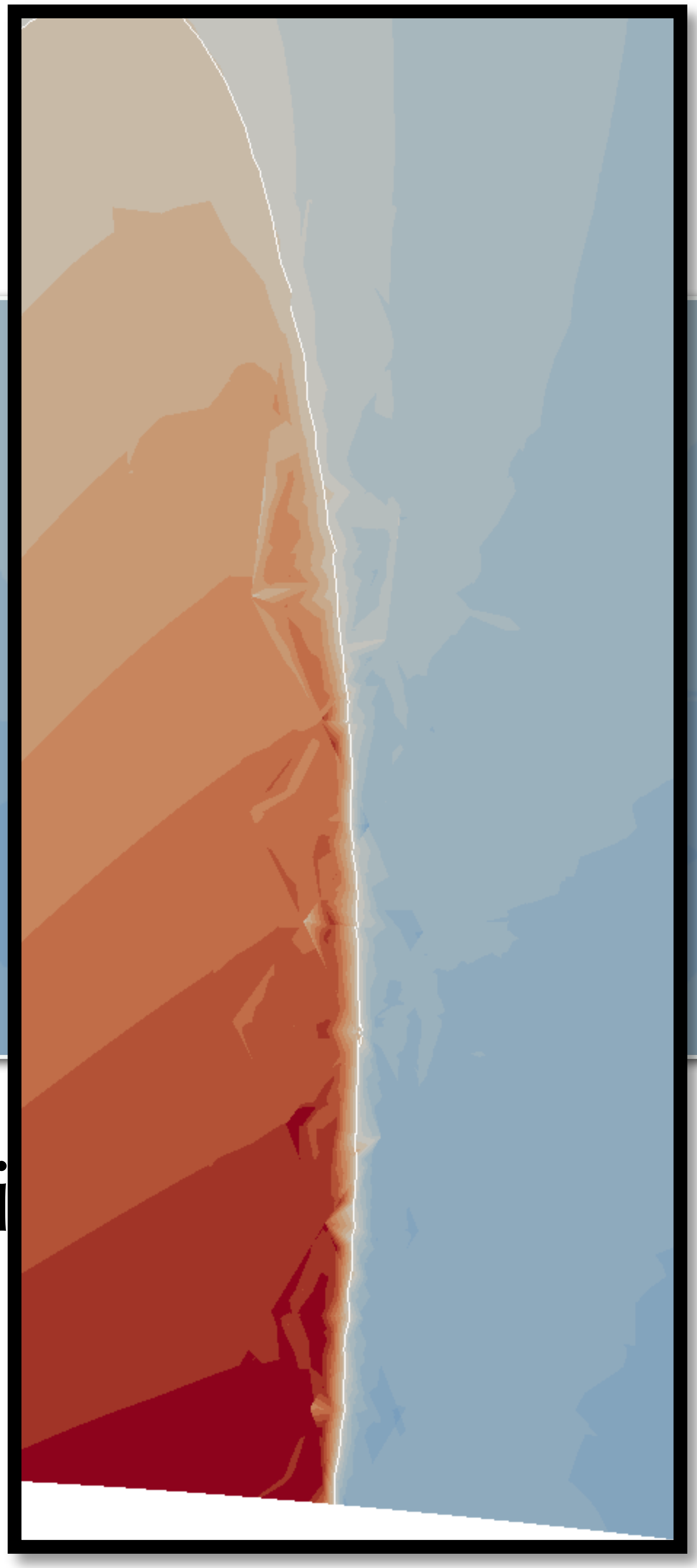
Example: NACA 0012 transonic



Starting mesh



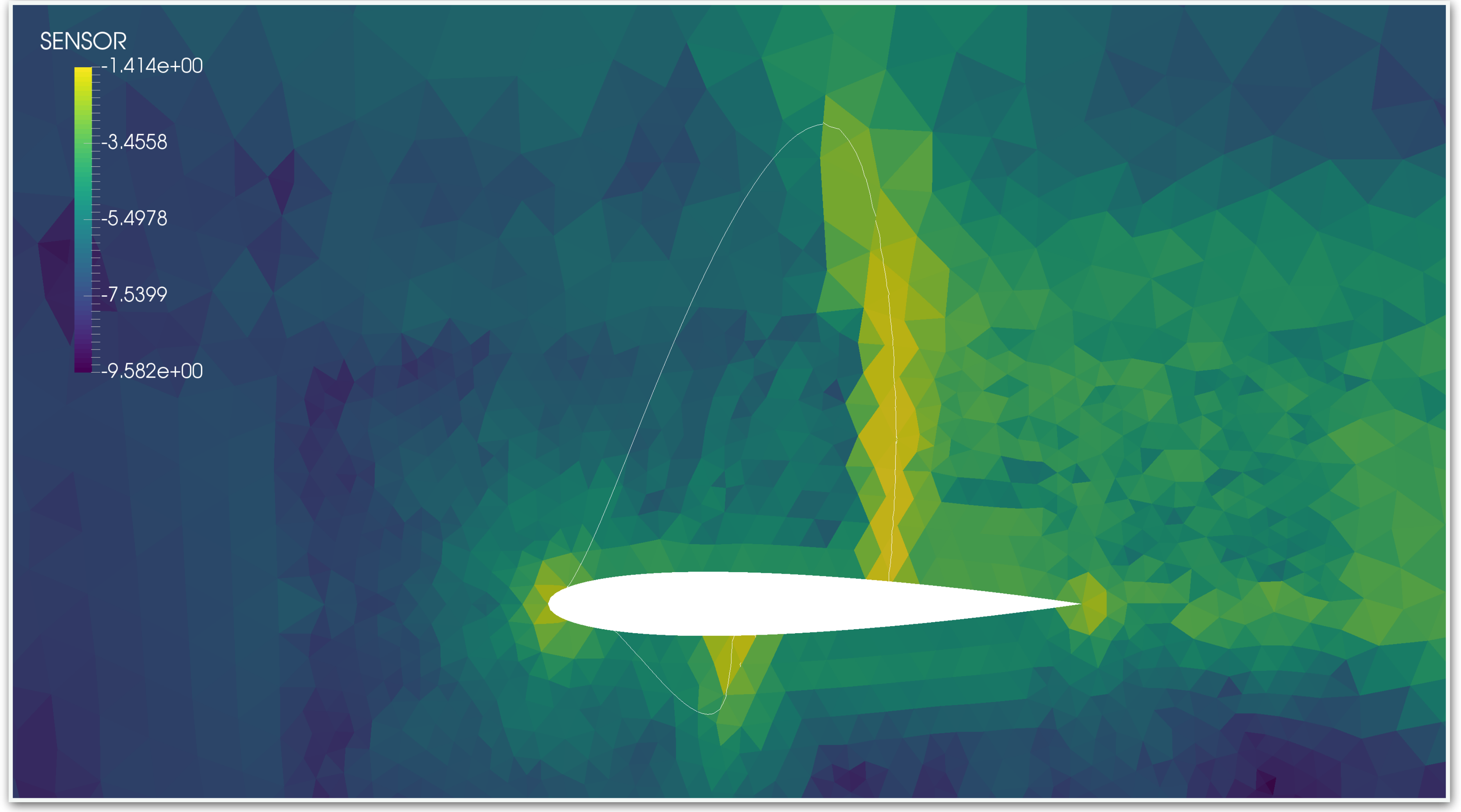
Ini



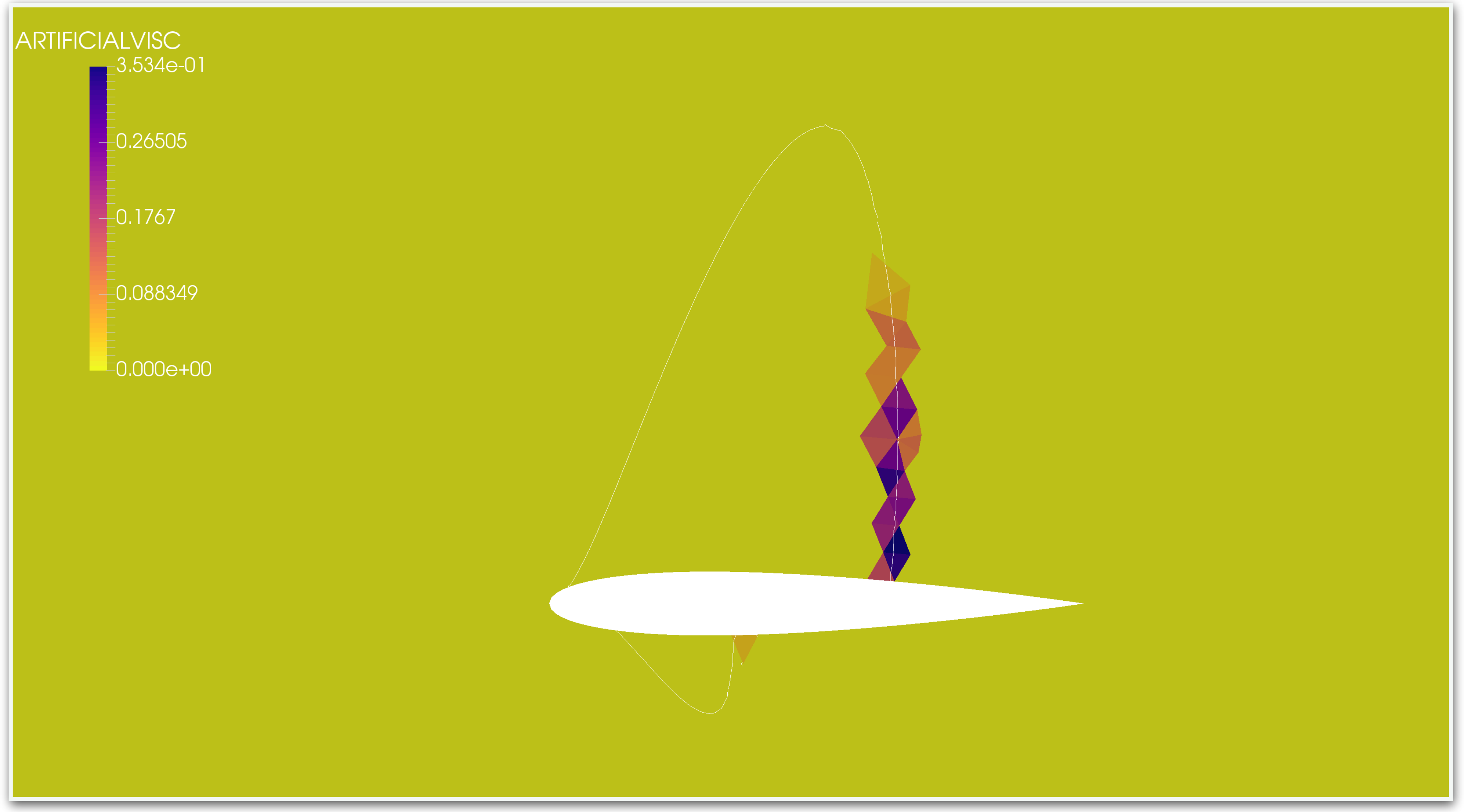
Ma = 0.8, 1.25° AoA

Marcon, Castiglioni, Moxey, Sherwin & Peiró, arXiv 1909.10973

Example: NACA 0012 transonic



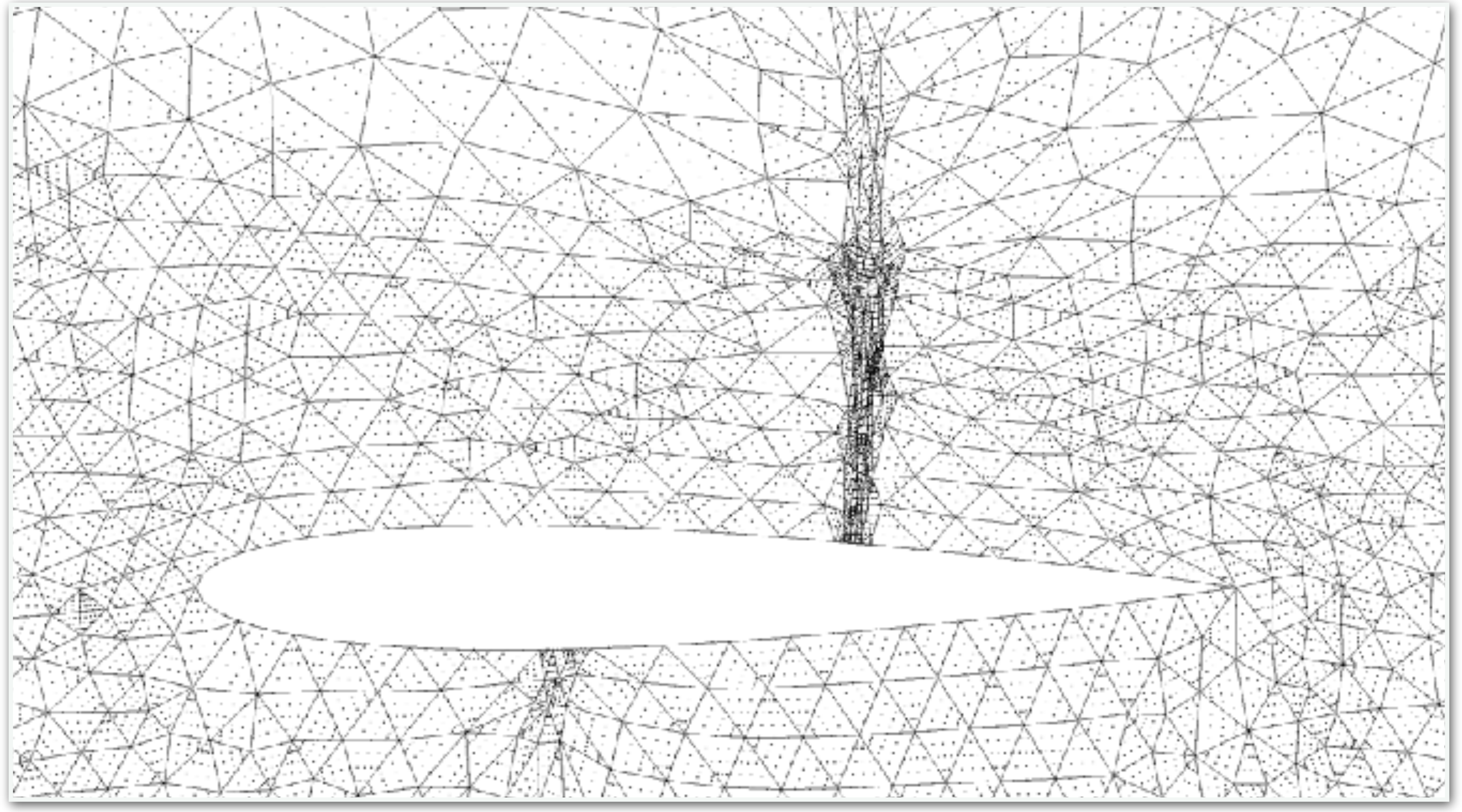
Discontinuity sensor



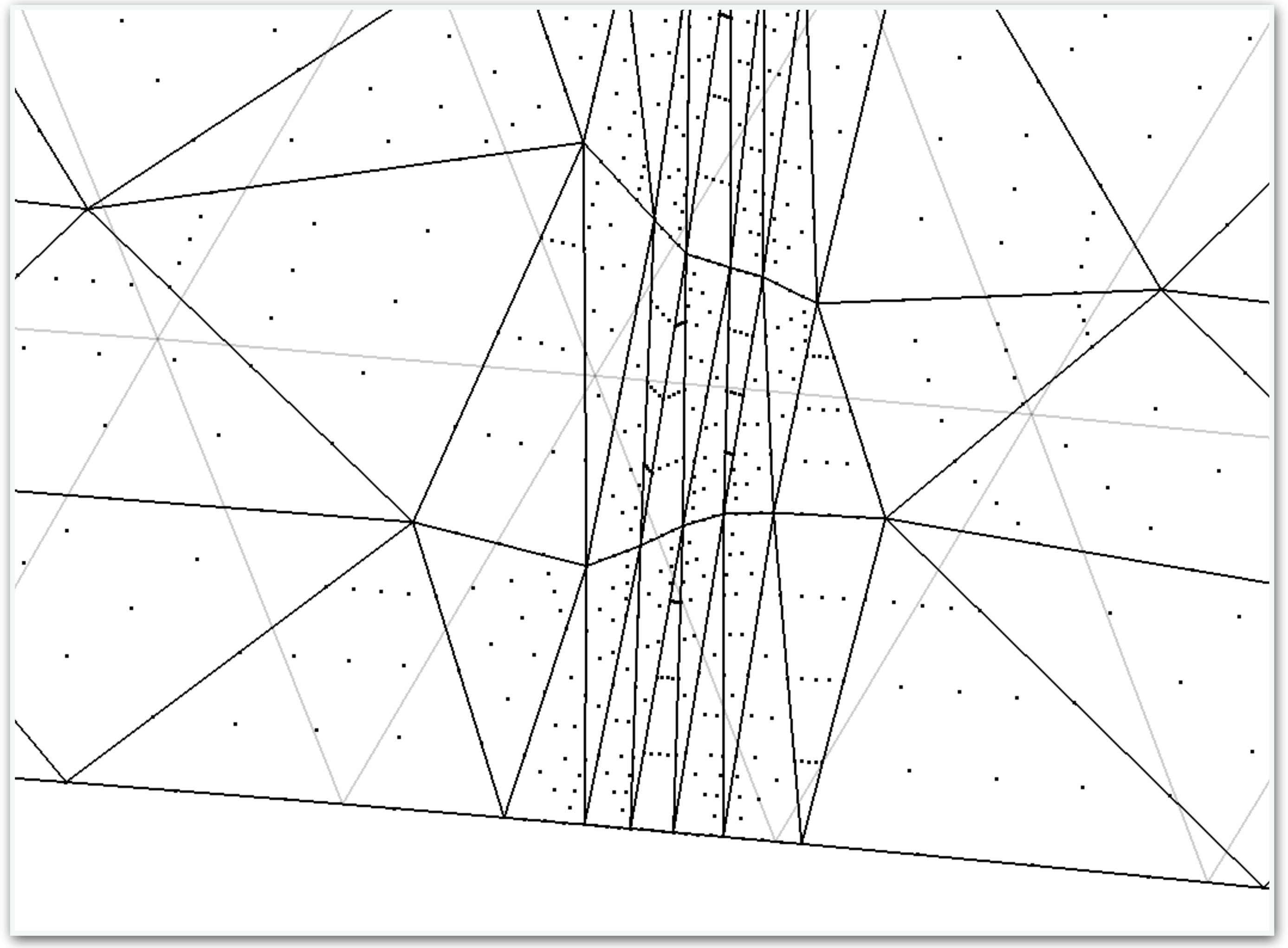
Artificial viscosity

Marcon, Castiglioni, Moxey, Sherwin & Peiró, arXiv 1909.10973

Example: NACA 0012 transonic



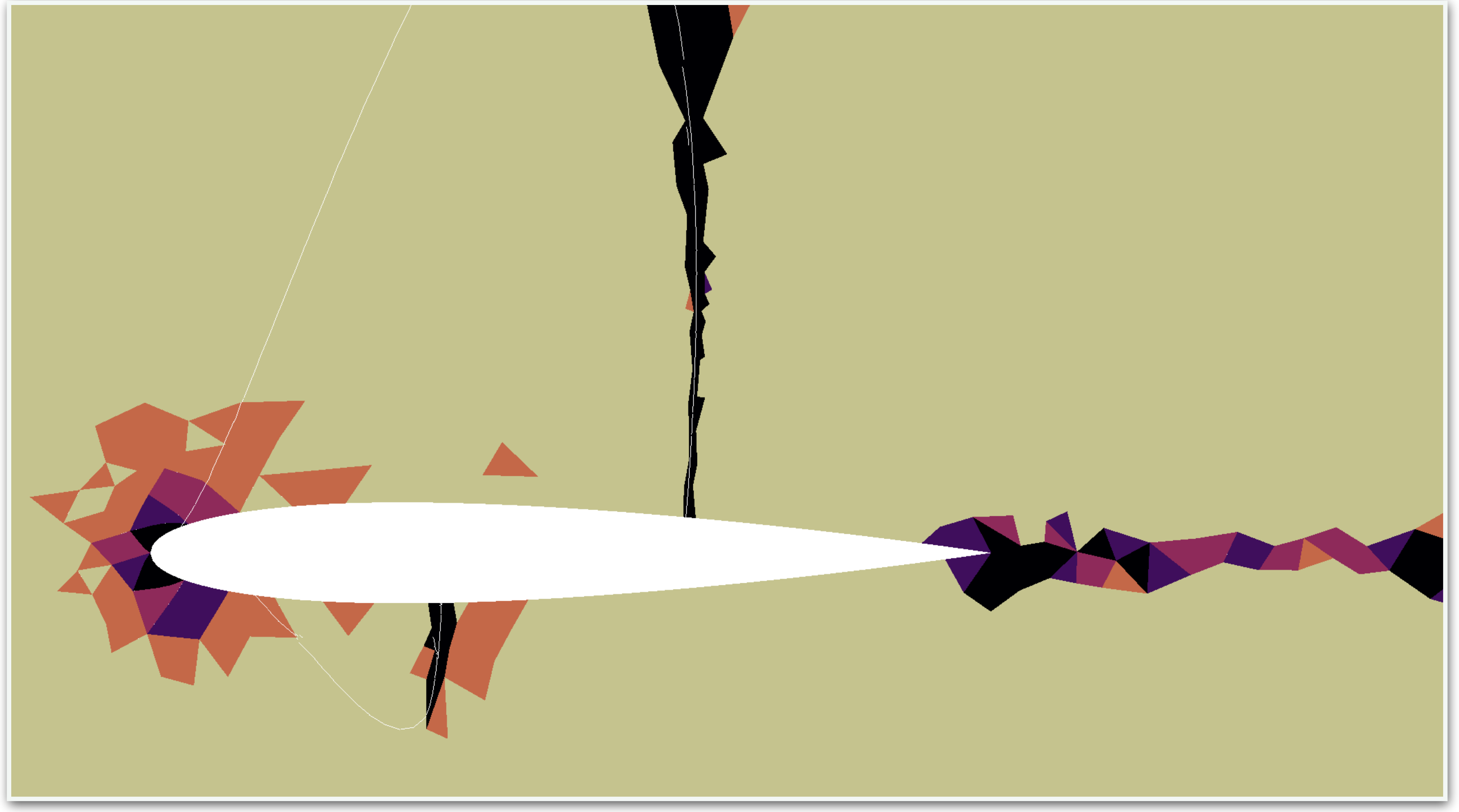
Calculate target size
& do r-adaptation



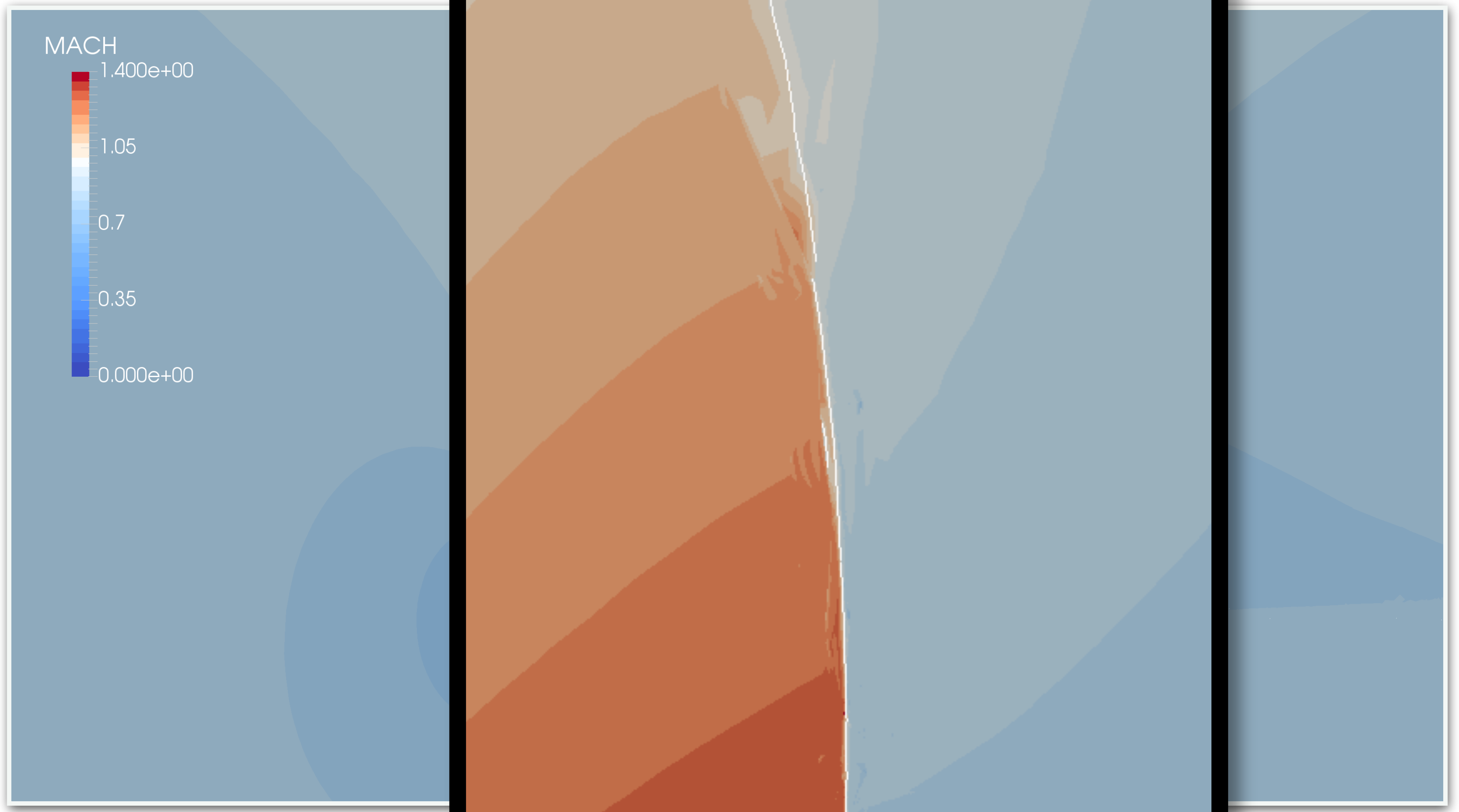
Use of CAD sliding

Marcon, Castiglioni, Moxey, Sherwin & Peiró, arXiv 1909.10973

Example: NACA 0012 transonic



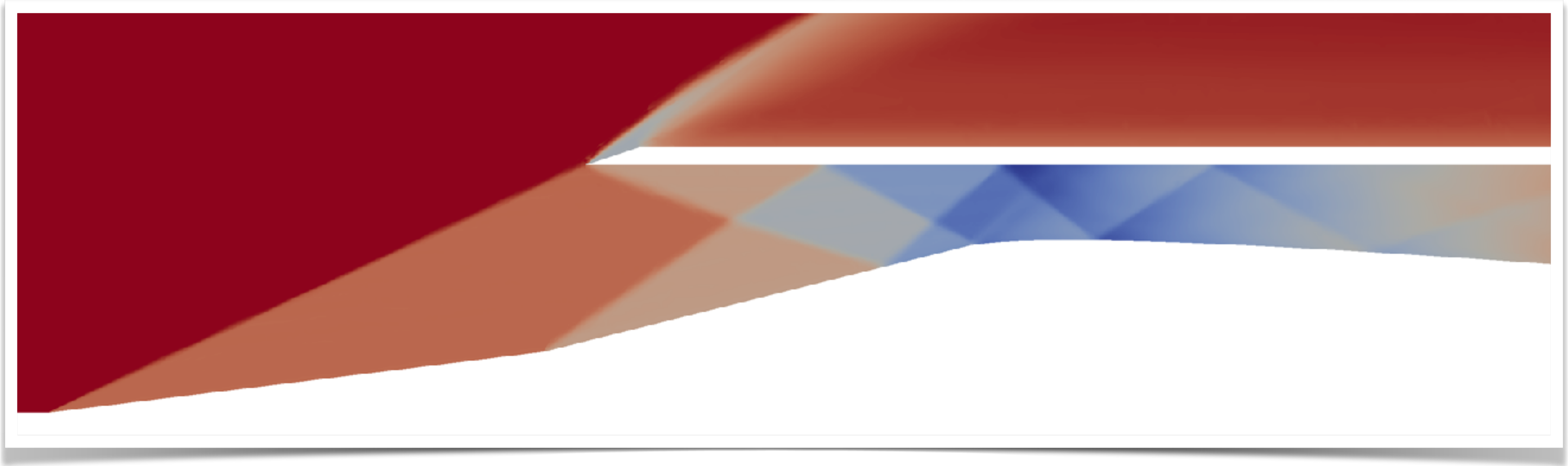
Translate to variable p

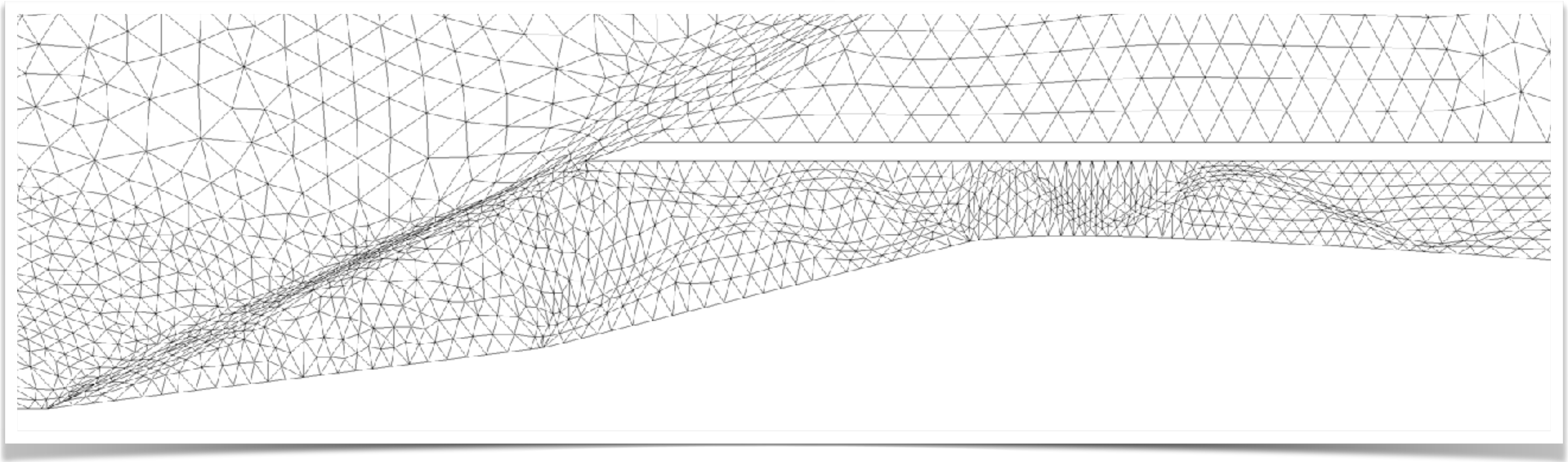


Improv … ng

Marcon, Castiglioni, Moxey, Sherwin & Peiró, arXiv 1909.10973

Supersonic example





Supersonic intake
Ma = 1.0

High-order splitting scheme

Navier–Stokes:

$$\partial_t \mathbf{u} + \mathbf{N}(\mathbf{u}) = -\nabla p + \nu \nabla^2 \mathbf{u}$$
$$\nabla \cdot \mathbf{u} = 0$$

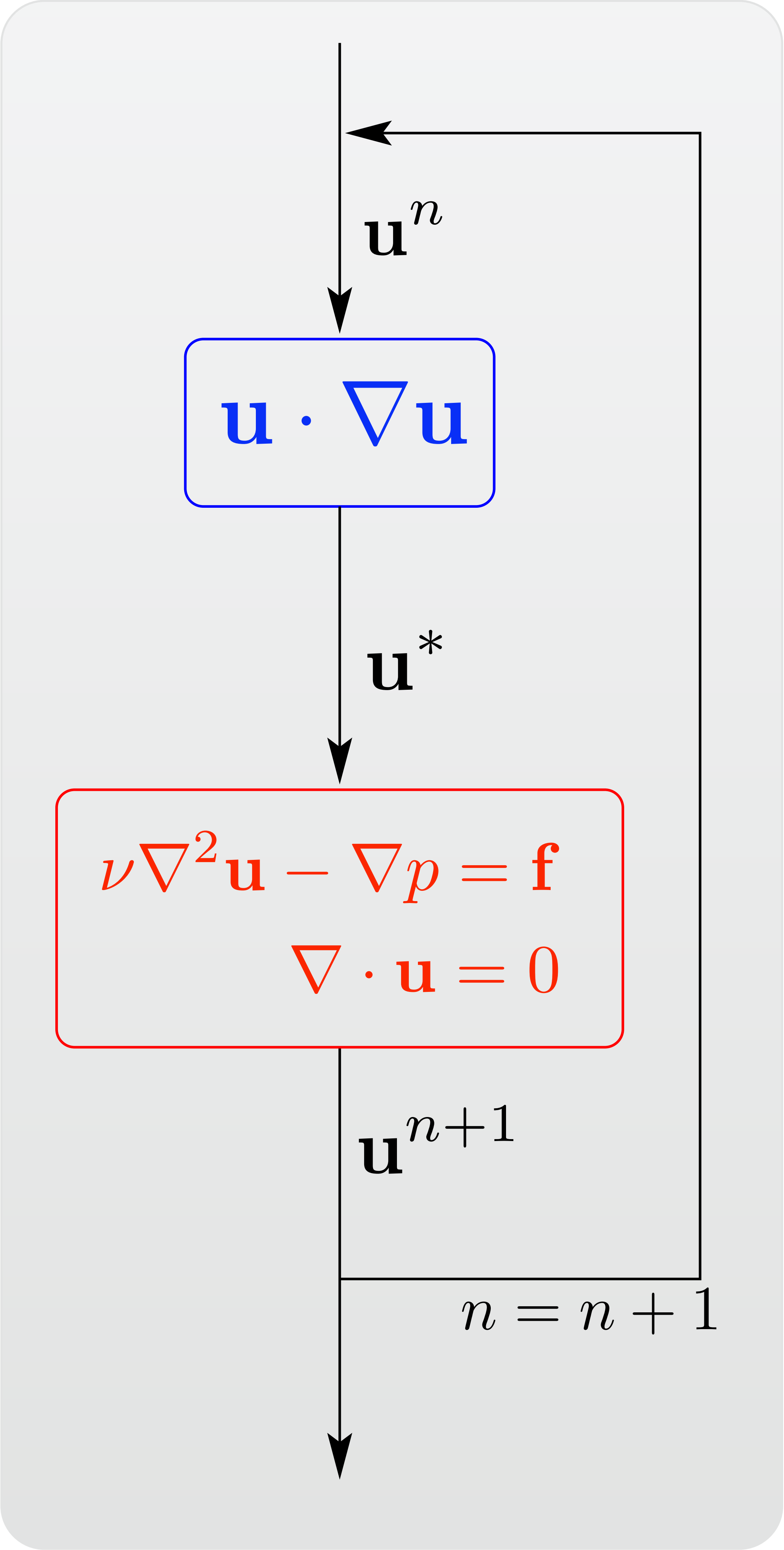
Velocity correction scheme *(aka stiffly stable)*:

Orszag, Israeli, Deville (90), Karnaidakis Israeli, Orszag (1991), Guermond & Shen (2003)

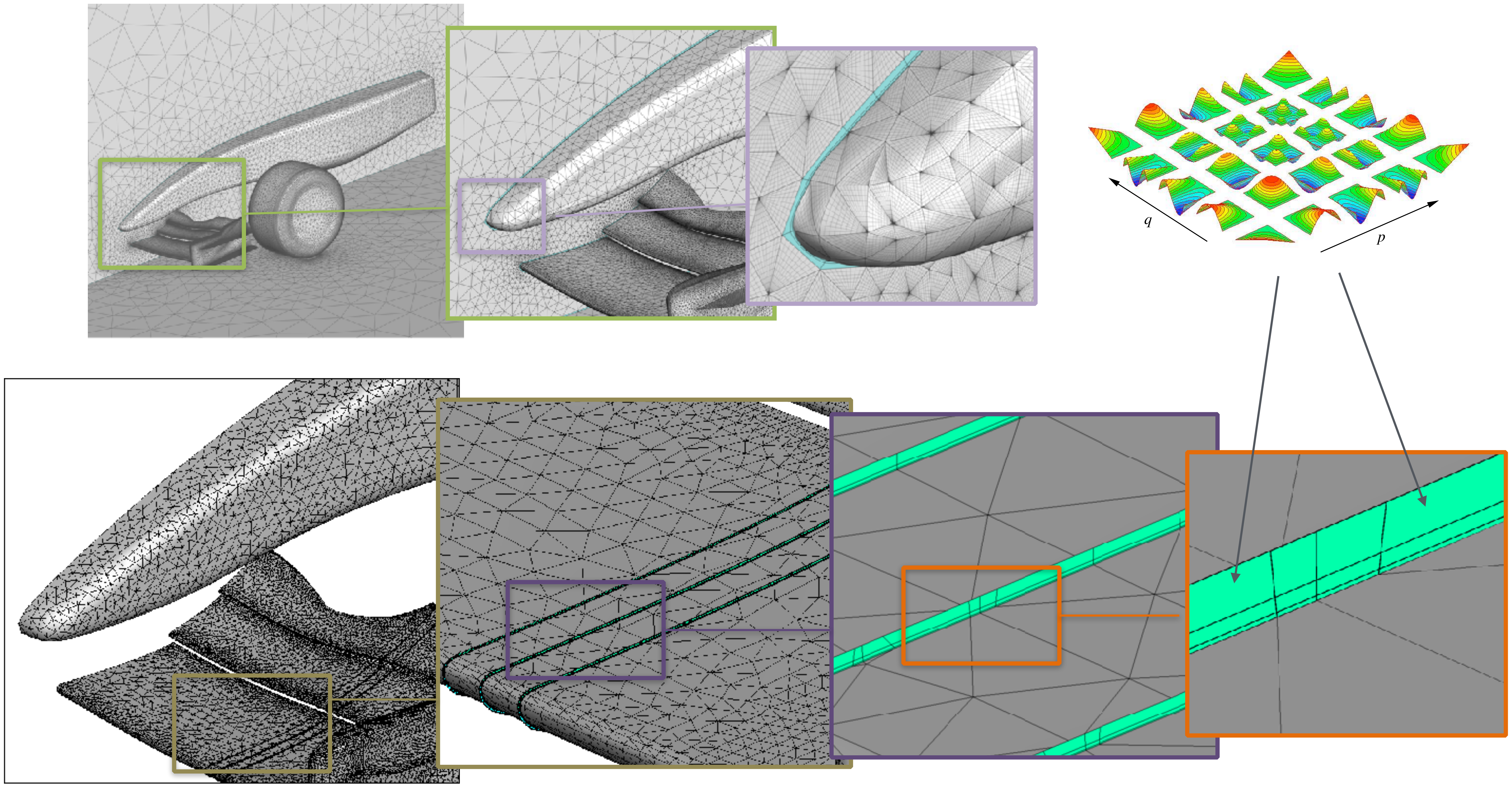
Advection: $u^* = -\sum_{q=1}^{J} \alpha_q \mathbf{u}^{n-q} - \Delta t \sum_{q=0}^{J-1} \beta_q \mathbf{N}(\mathbf{u}^{n-q})$

Pressure Poisson: $\nabla^2 p^{n+1} = \frac{1}{\Delta t} \nabla \cdot \mathbf{u}^*$

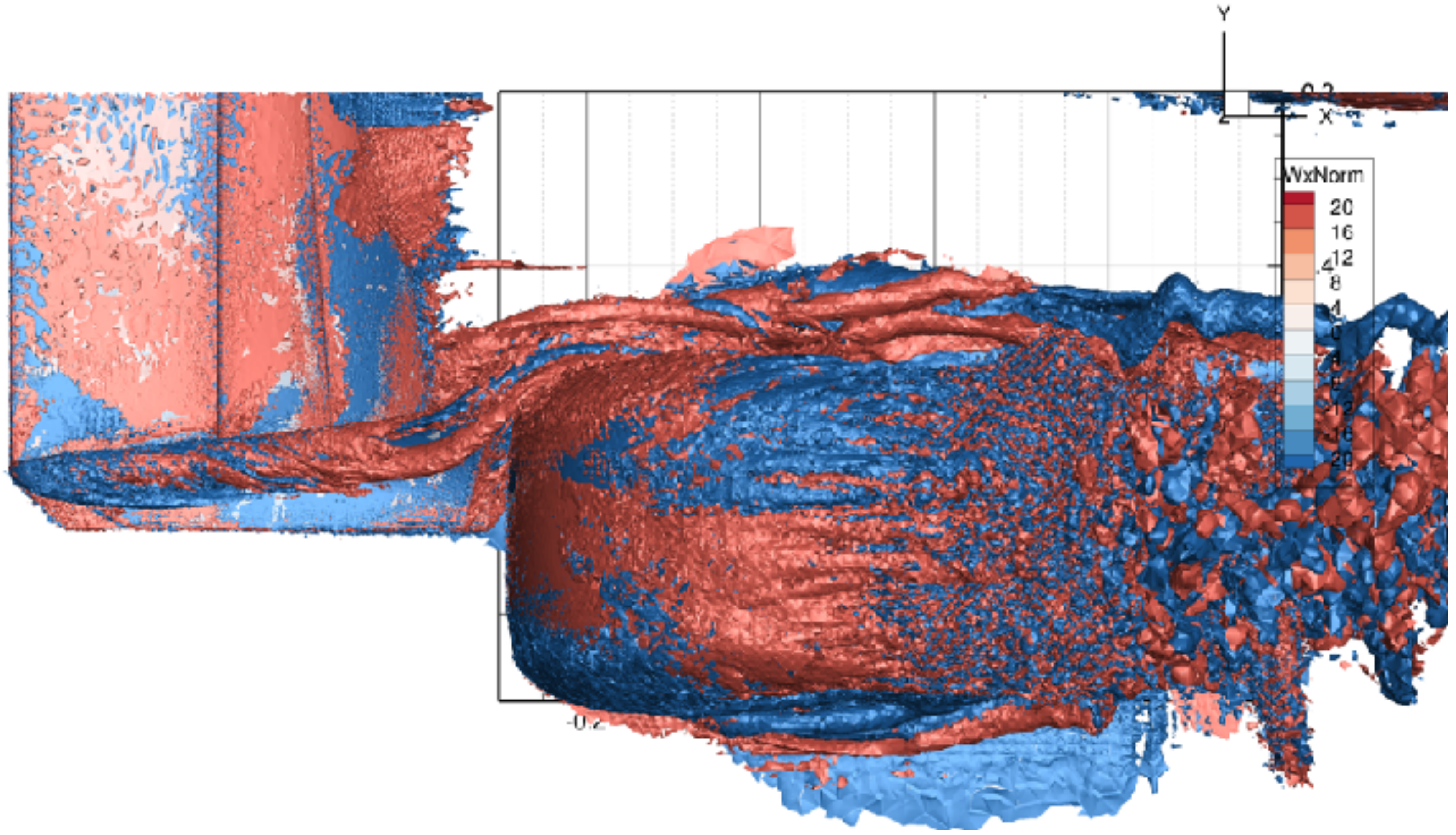
Helmholtz: $\nabla^2 \mathbf{u}^{n+1} - \frac{\alpha_0}{\nu \Delta t} \mathbf{u}^{n+1} = -\frac{\mathbf{u}^*}{\nu \Delta t} + \frac{1}{\nu} \nabla p^{n+1}$

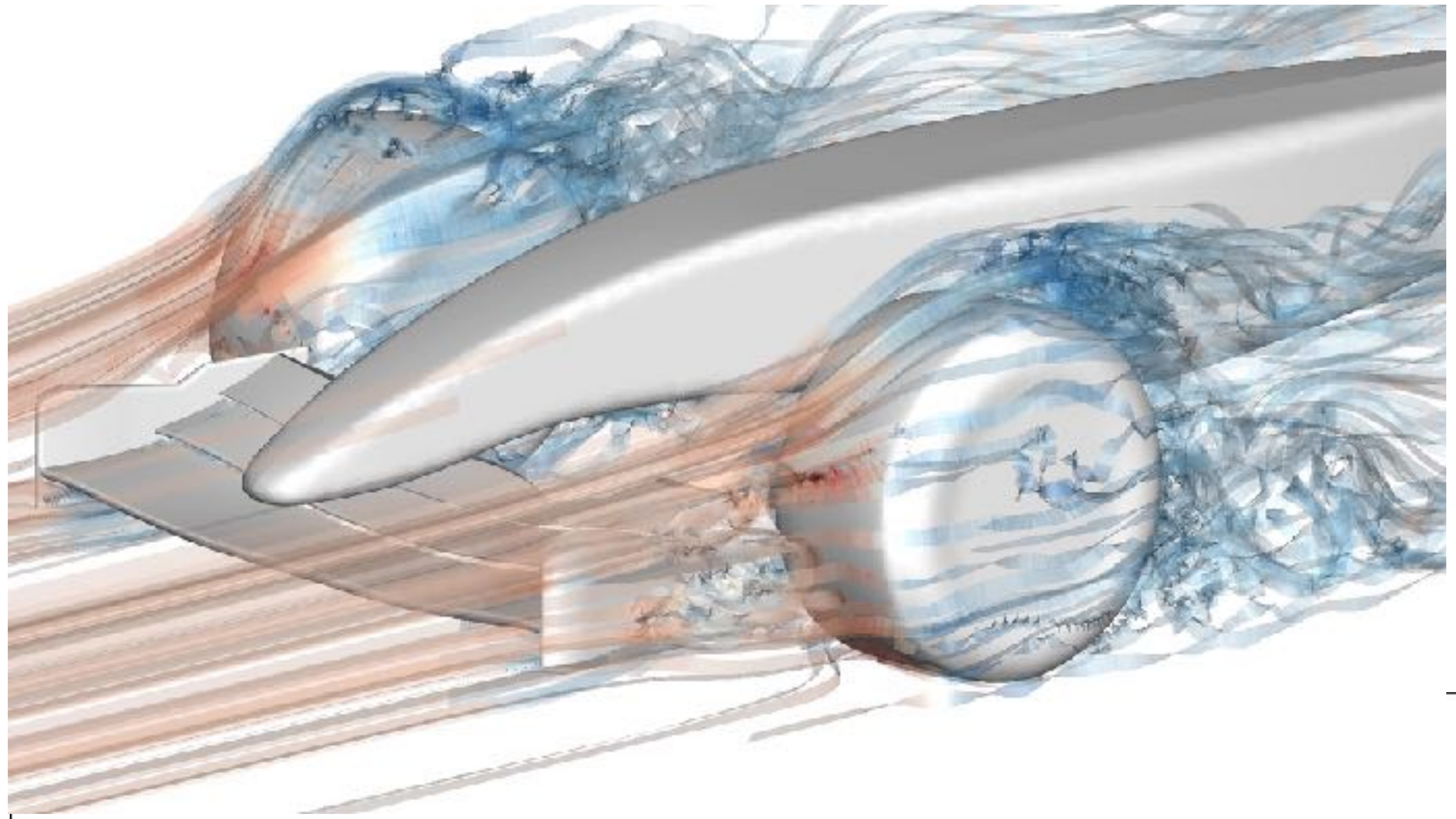


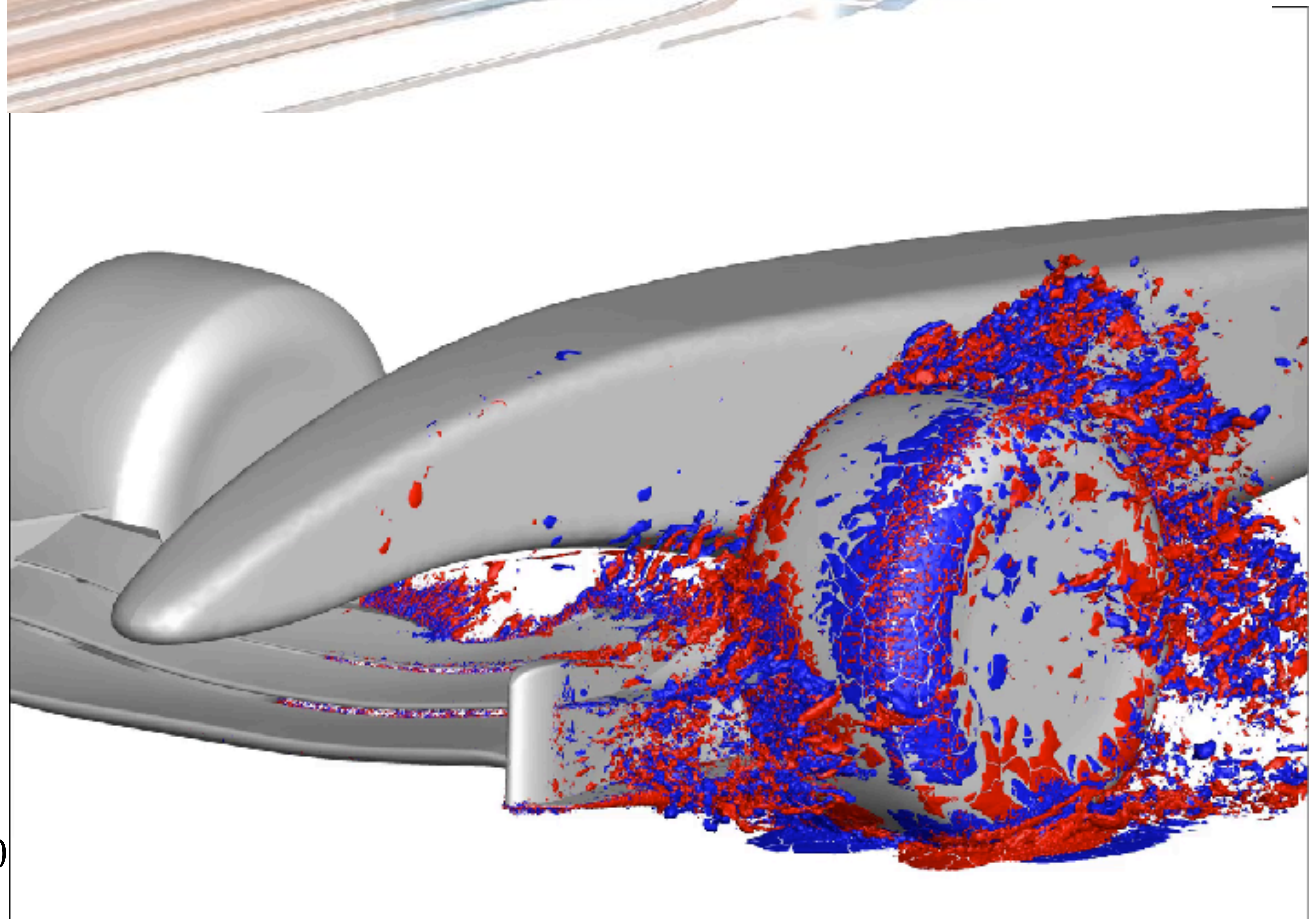
Meshing for F1 applications



More complex geometries



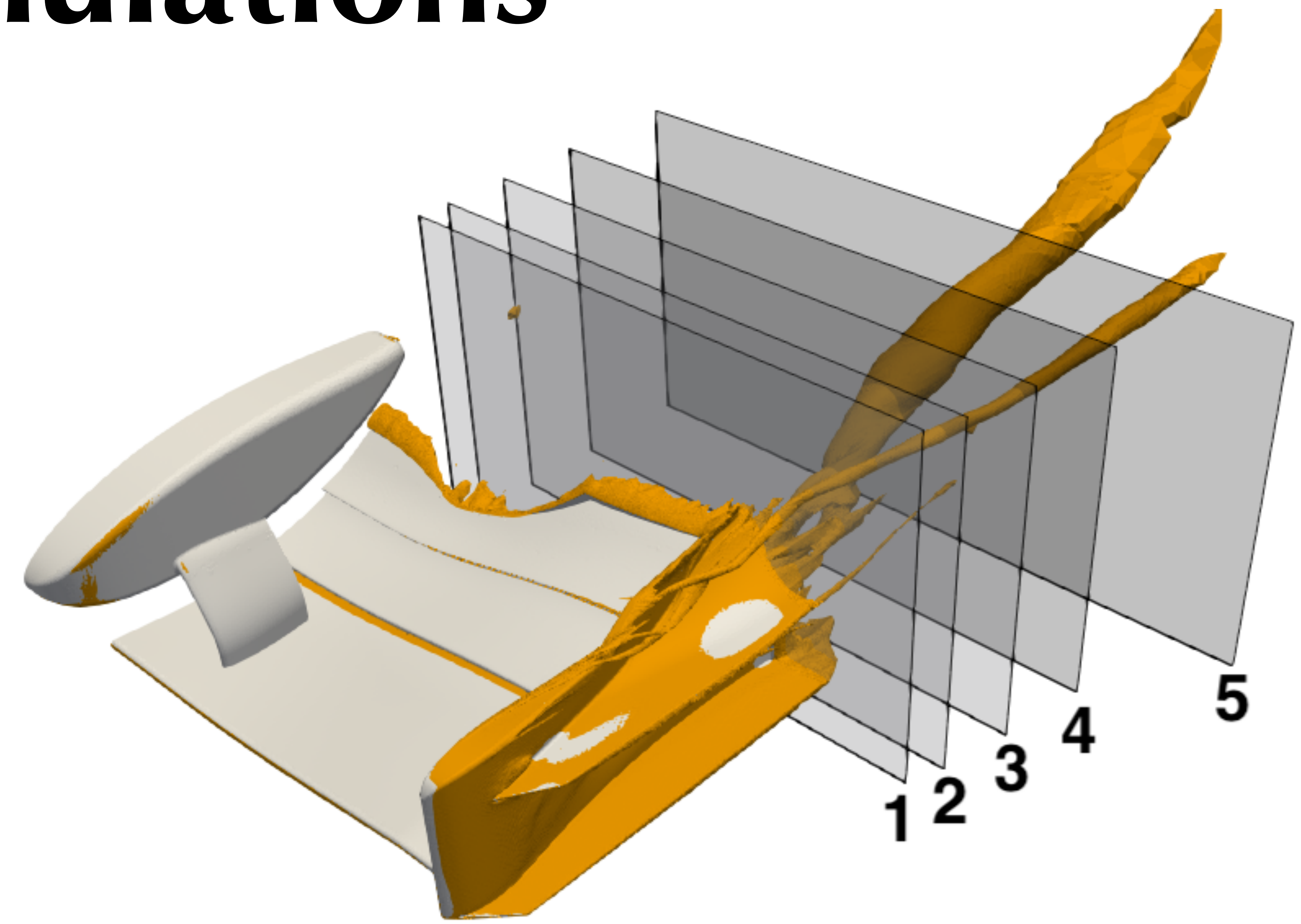


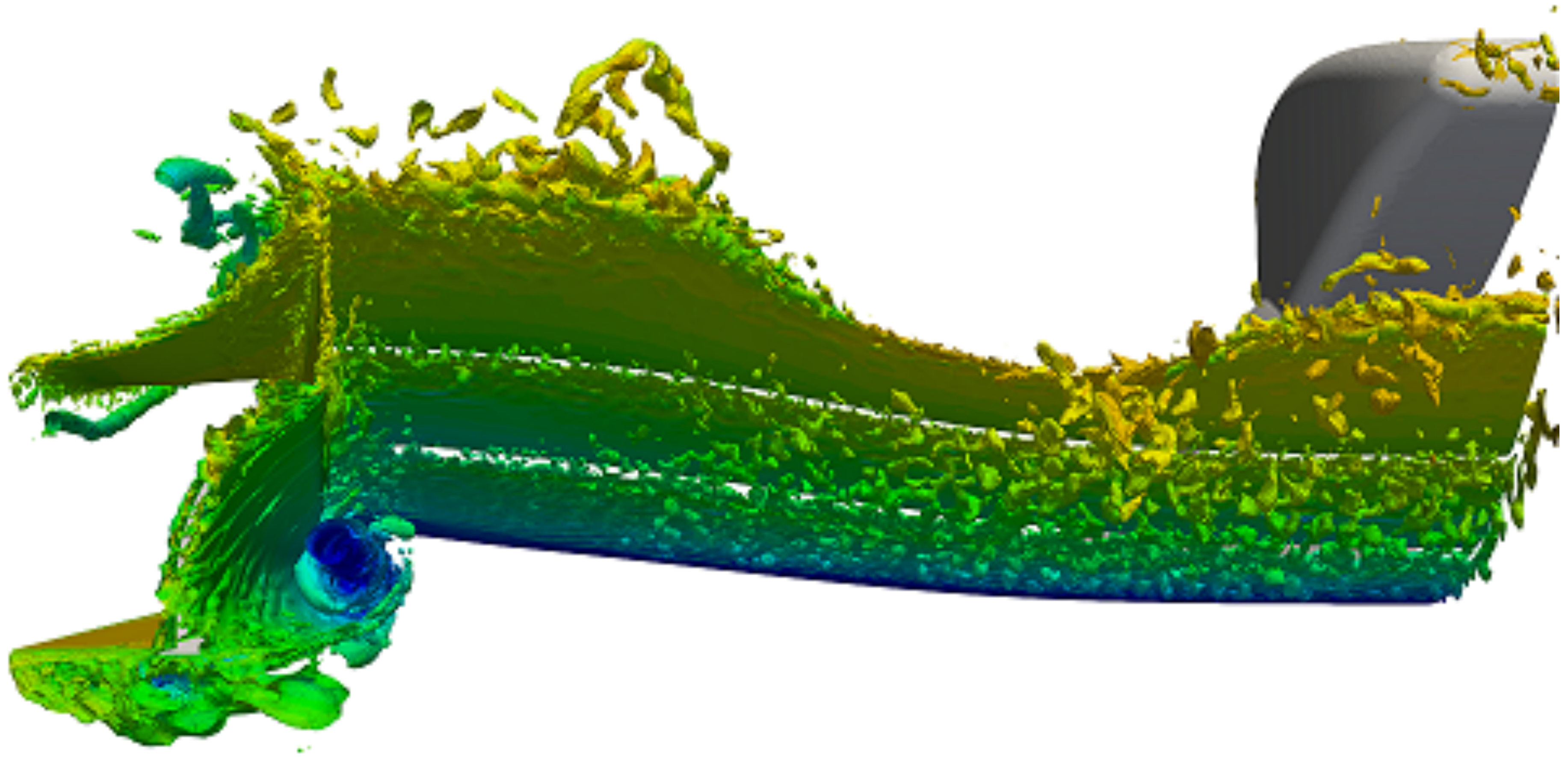


Supported by ARCHER leadership award (20m CPU hours)

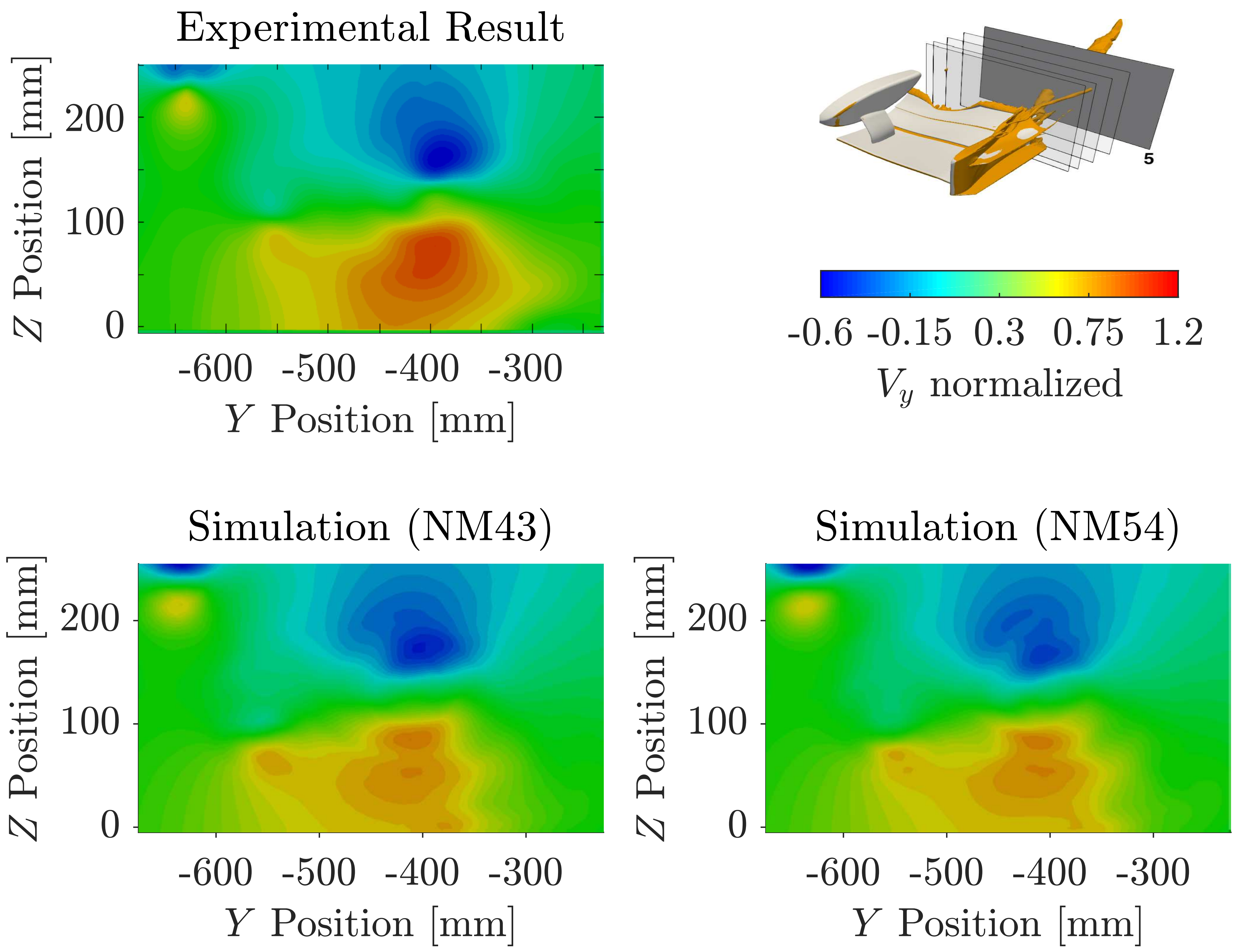
Recent F1 simulations

- F1 simulations highlight complex vortex interaction cases: ideal candidates for LES.
- Front wing simulations with experimental PIV datasets as new proposed benchmark case.
- Analysis found in Buscariolo, Hoessler, Moxey et al, arXiv 1909.06701.
- Datasets in DOI: 10.14469/hpc/6049

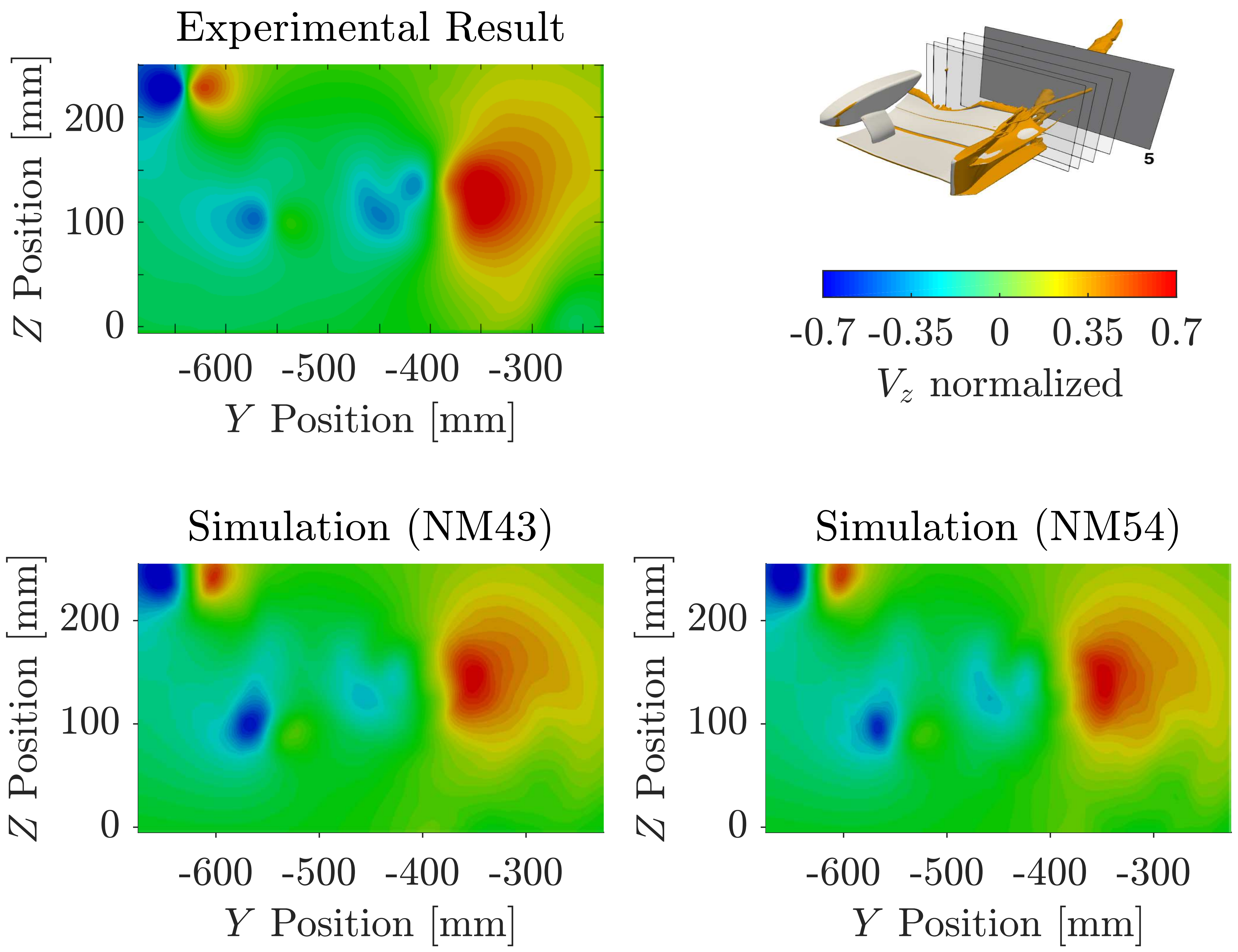




Comparison with experiment



u component



v component

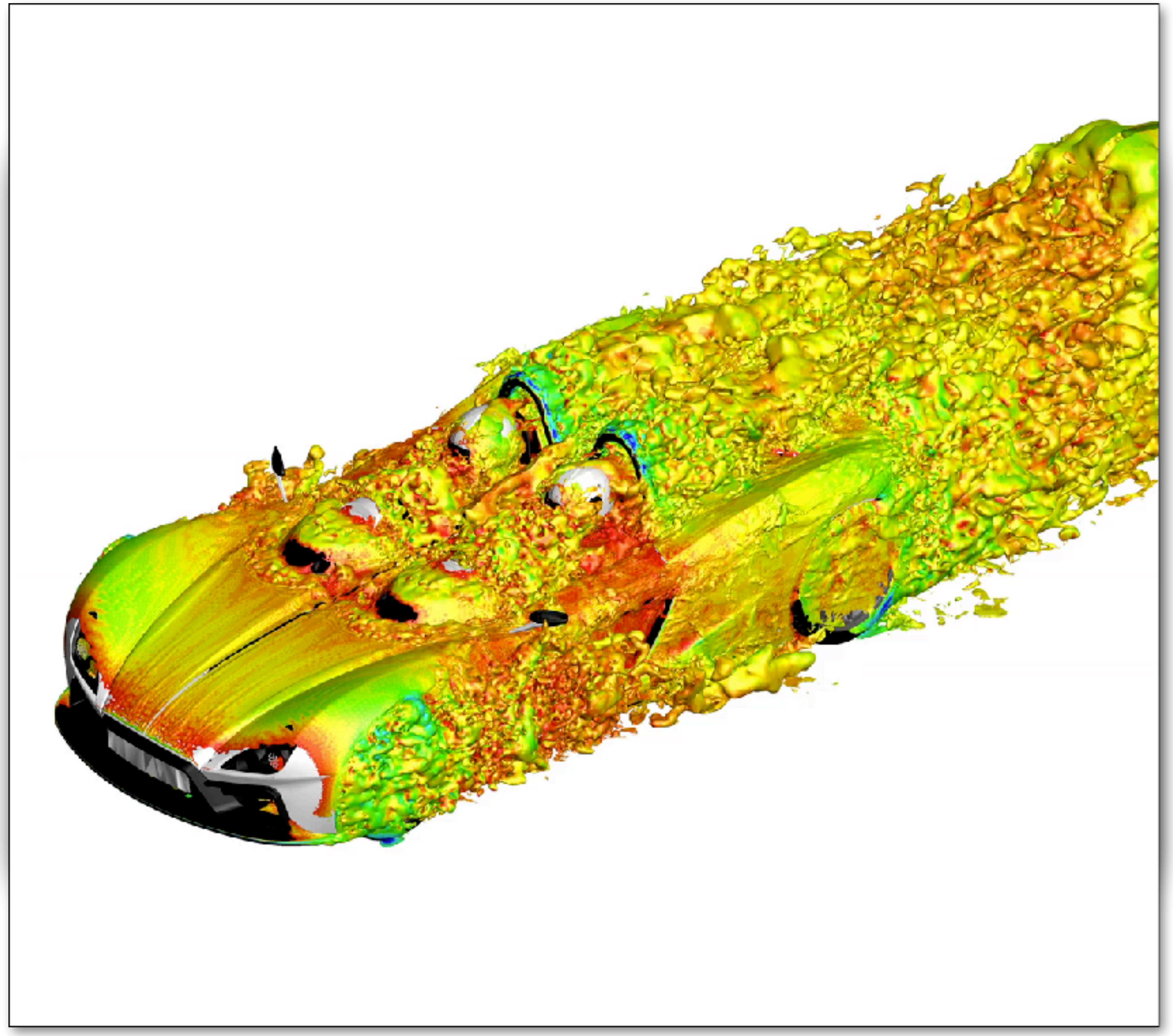
Elemental road racing car

- Most challenging case undertaken with Nektar++ to date (that I know of!)

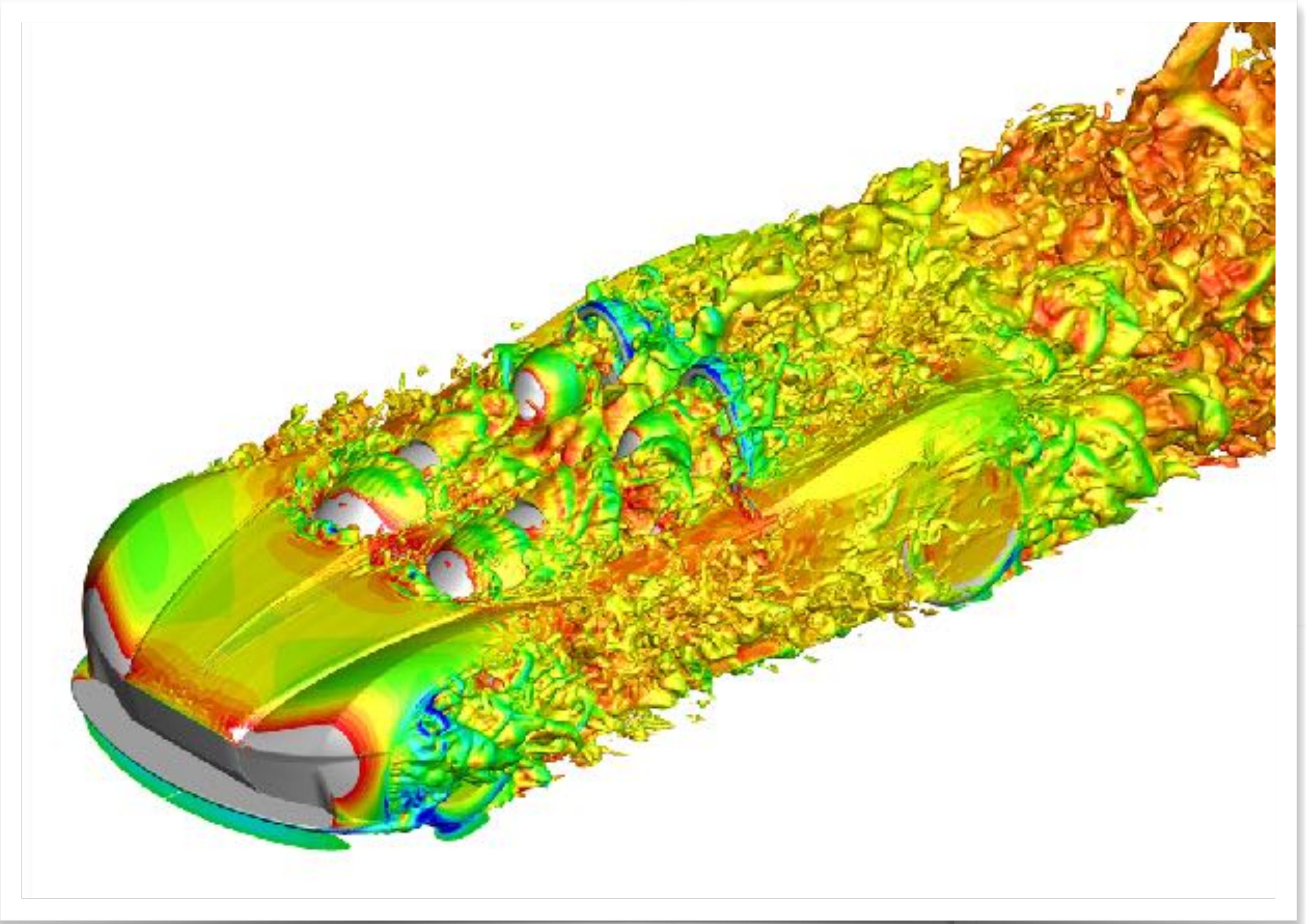
- Re ~ 1m, around 1bn dof.

- Simulated at $P = 5$ with a matching high-order mesh and SVV-LES.

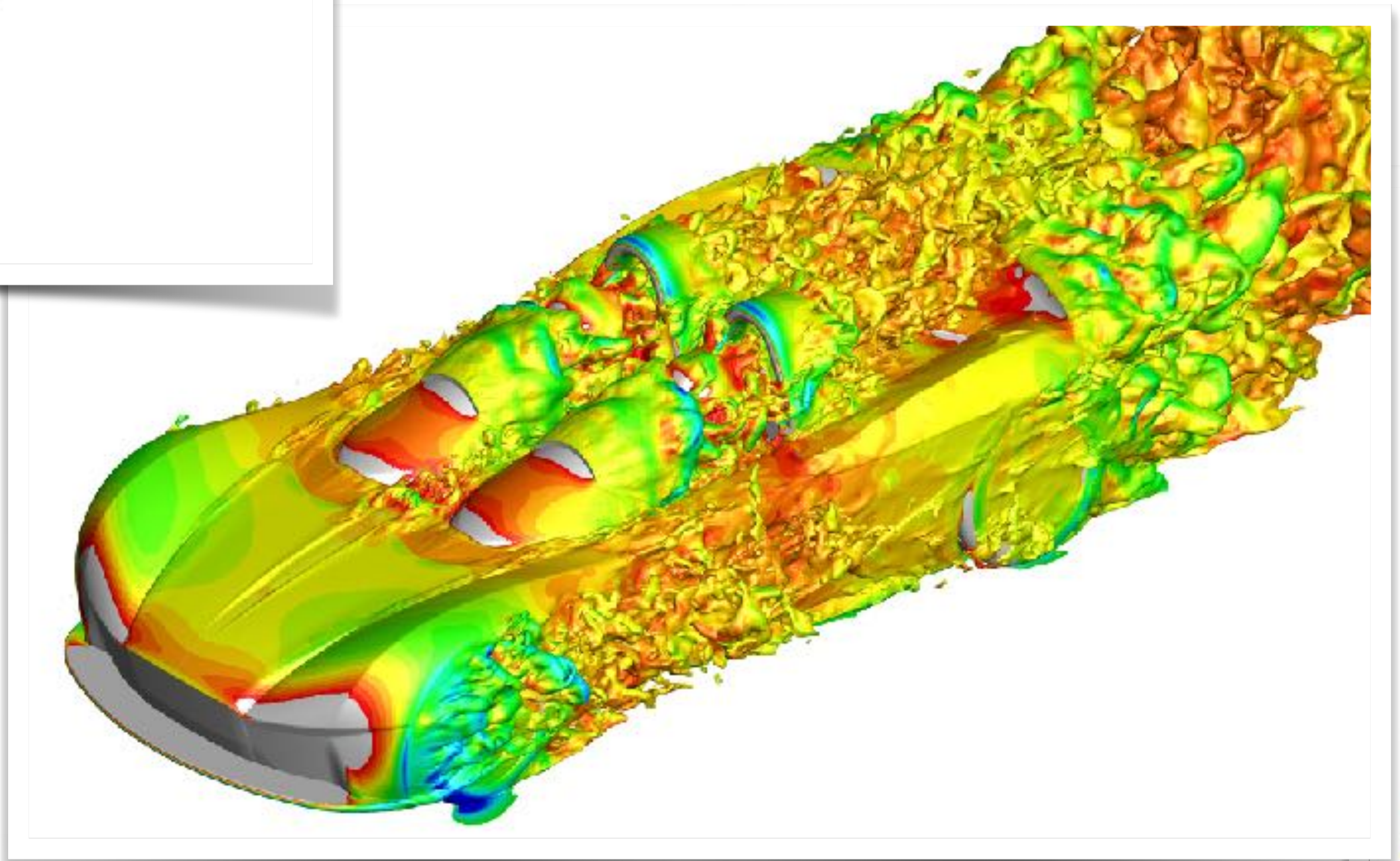
- Aim to identify aerodynamic issues and refine design.



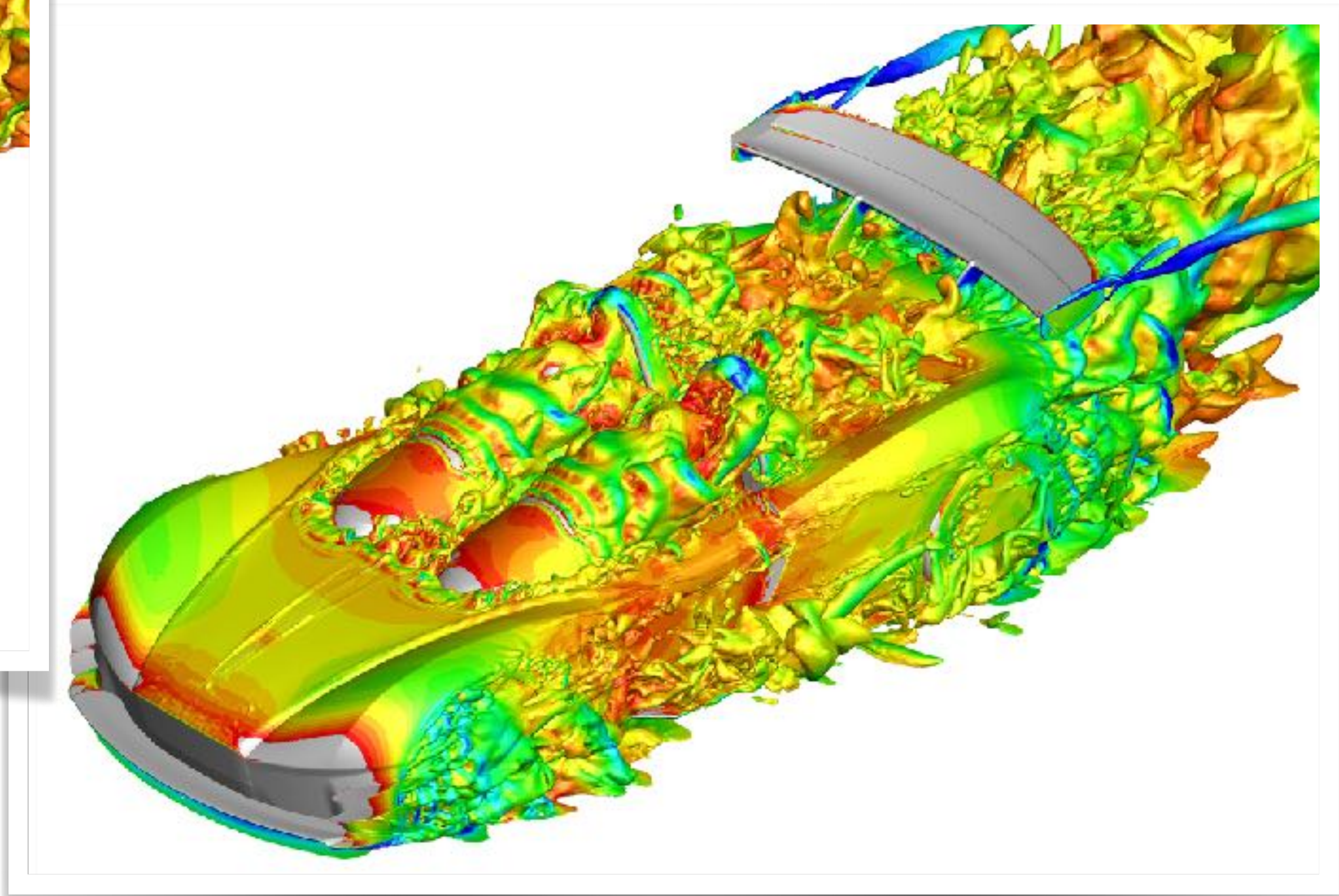
Elemental road race car



5th order
Re = 1m



Design 2: +33% Downforce



Design 3:   +270% Downforce

Moxey, Turner, Jassim, Taylor, Peiro & Sherwin

Summary

- We can certainly spectral/*hp* element techniques to challenging industrial flow problems and succeed!
- Accurate, transient flow modelling is an **enabling technology** for high-end engineering/physics.
- But… there is still a way to go yet!
  - Meshing for 3D geometries is a specialist skill.
  - Robustness still requires more analysis.

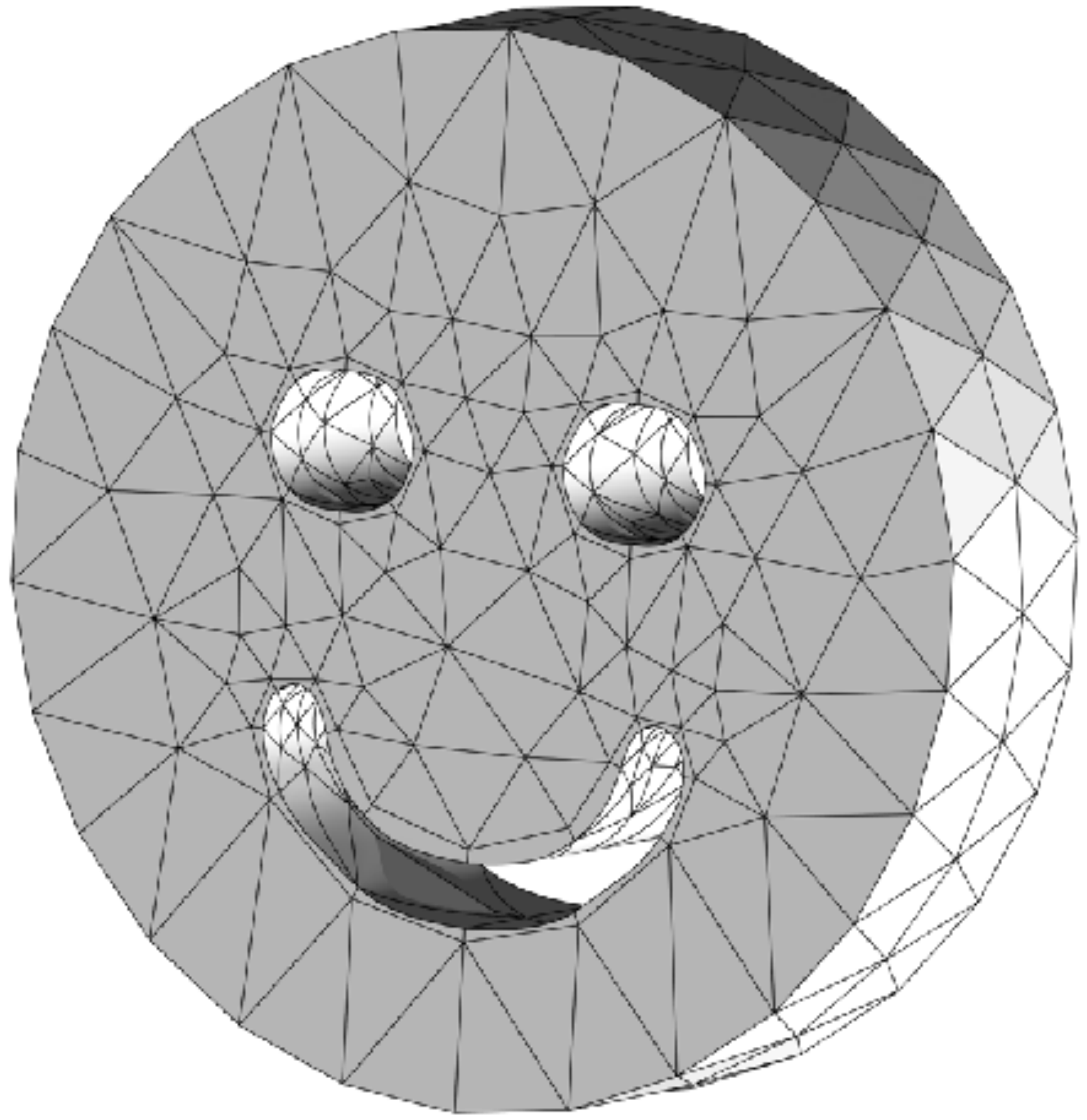
Thanks for listening!

https://davidmoxey.uk/

d.moxey@exeter.ac.uk

www.nektar.info

https://prism.ac.uk/



EPSRC
Pioneering research and skills

Nektar++: enhancing the capability and application of high-fidelity spectral/*hp* element methods

David Moxey[1], Chris D. Cantwell[2], Yan Bao[3], Andrea Cassinelli[2], Giacomo Castiglioni[2], Sehun Chun[4], Emilia Juda[2], Ehsan Kazemi[4], Kilian Lackhove[6], Julian Marcon[2], Gianmarco Mengaldo[7], Douglas Serson[2], Michael Turner[2], Hui Xu[5,2], Joaquim Peiró[2], Robert M. Kirby[8], Spencer J. Sherwin[2]