Towards high-fidelity industrial fluid dynamics simulations at high order

David Moxey
Department of Engineering, King's College London
College of Engineering, Mathematics & Physical Sciences, University of Exeter

Platform for Advanced Scientific Computing

6th July 2021

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

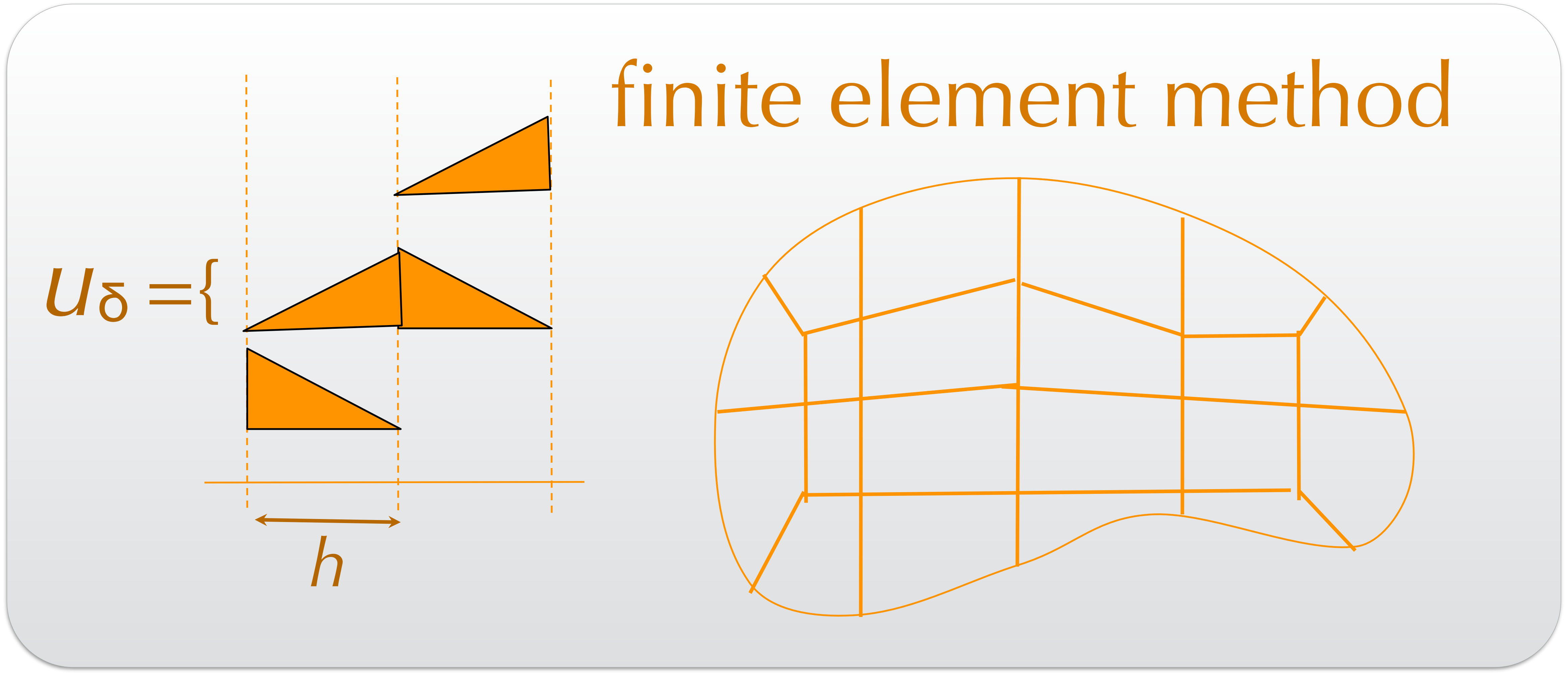
Increasing desire for **high-fidelity** CFD simulations in high-end aeronautics applications.

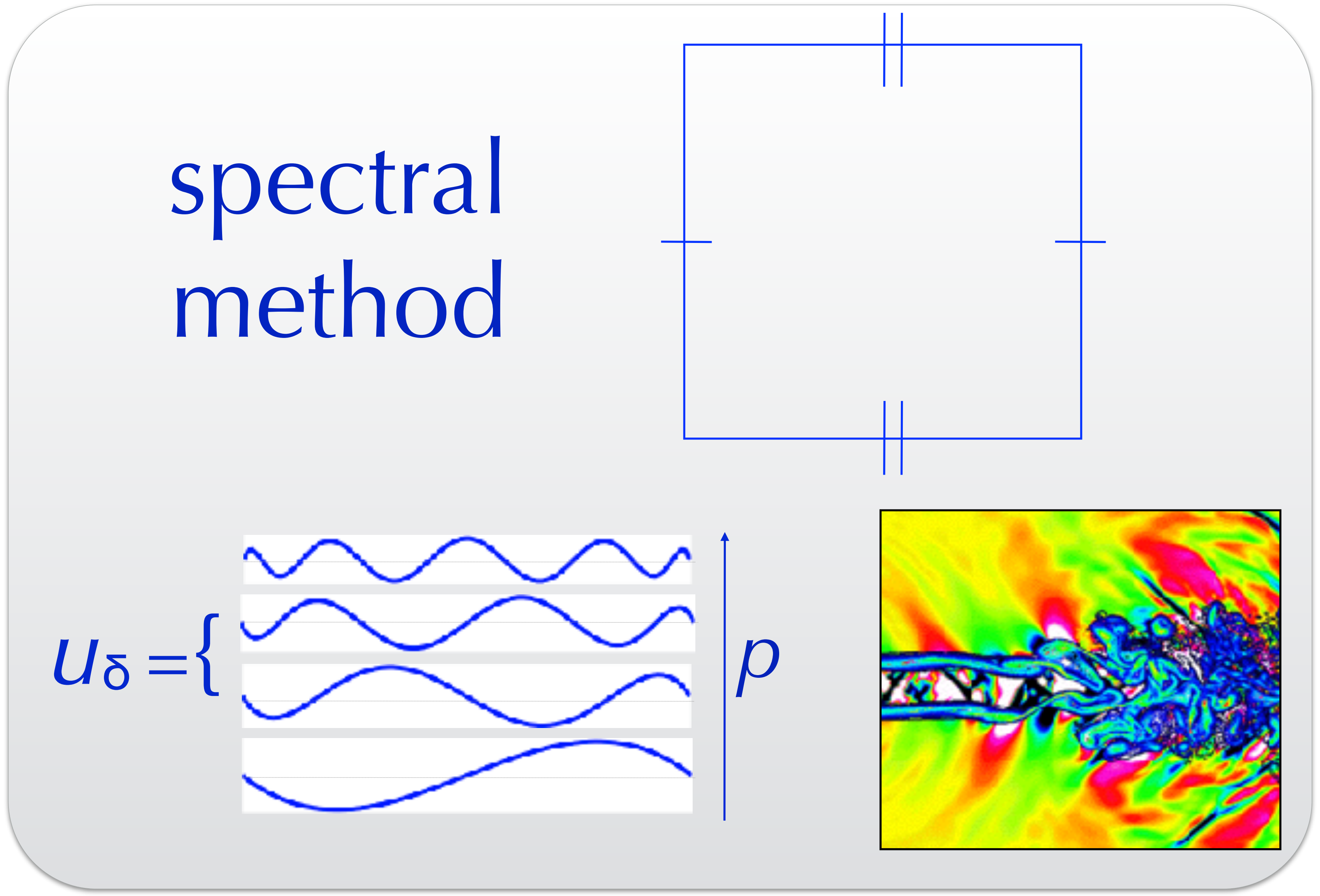
Want to accurately model 'difficult' features:

- strongly separated flows, vortex interaction;
- feature tracking and prediction.

Goal: develop methods and techniques for making LES affordable & routine, based on high-order finite element methods.

What are high-order methods?





spatial flexibility (*h*)

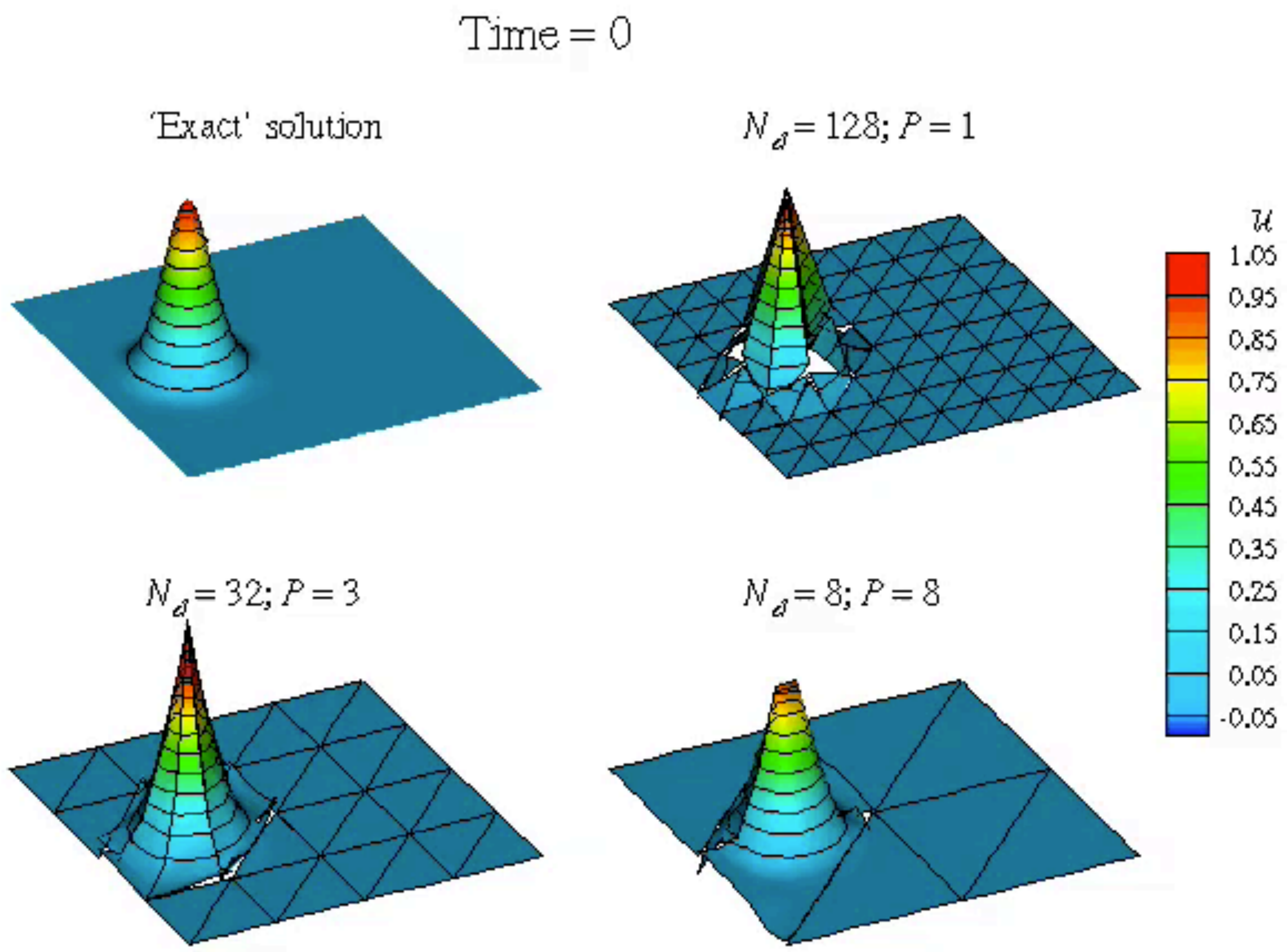
+

accuracy (*p*)

⟶

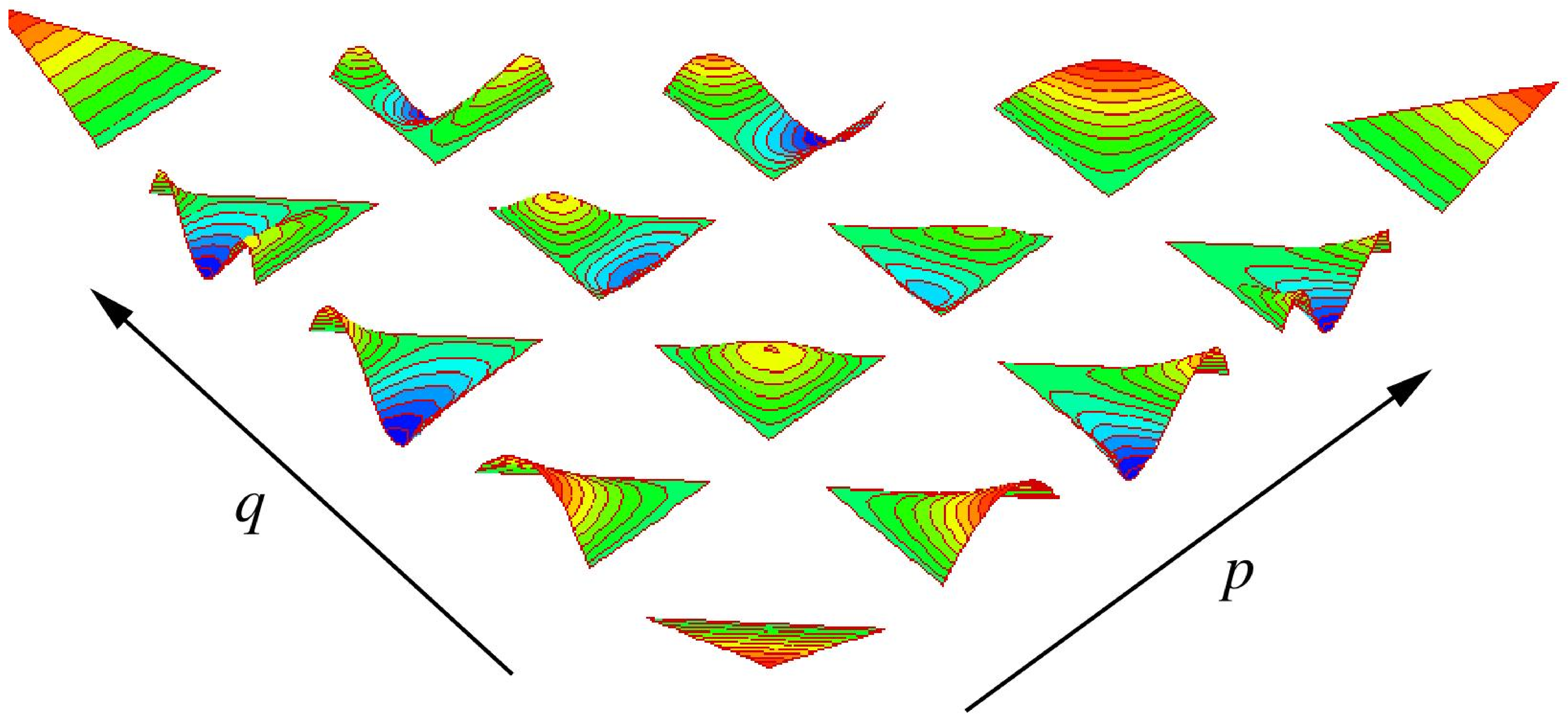
spectral/*hp* element

Why use a high-order method for CFD?



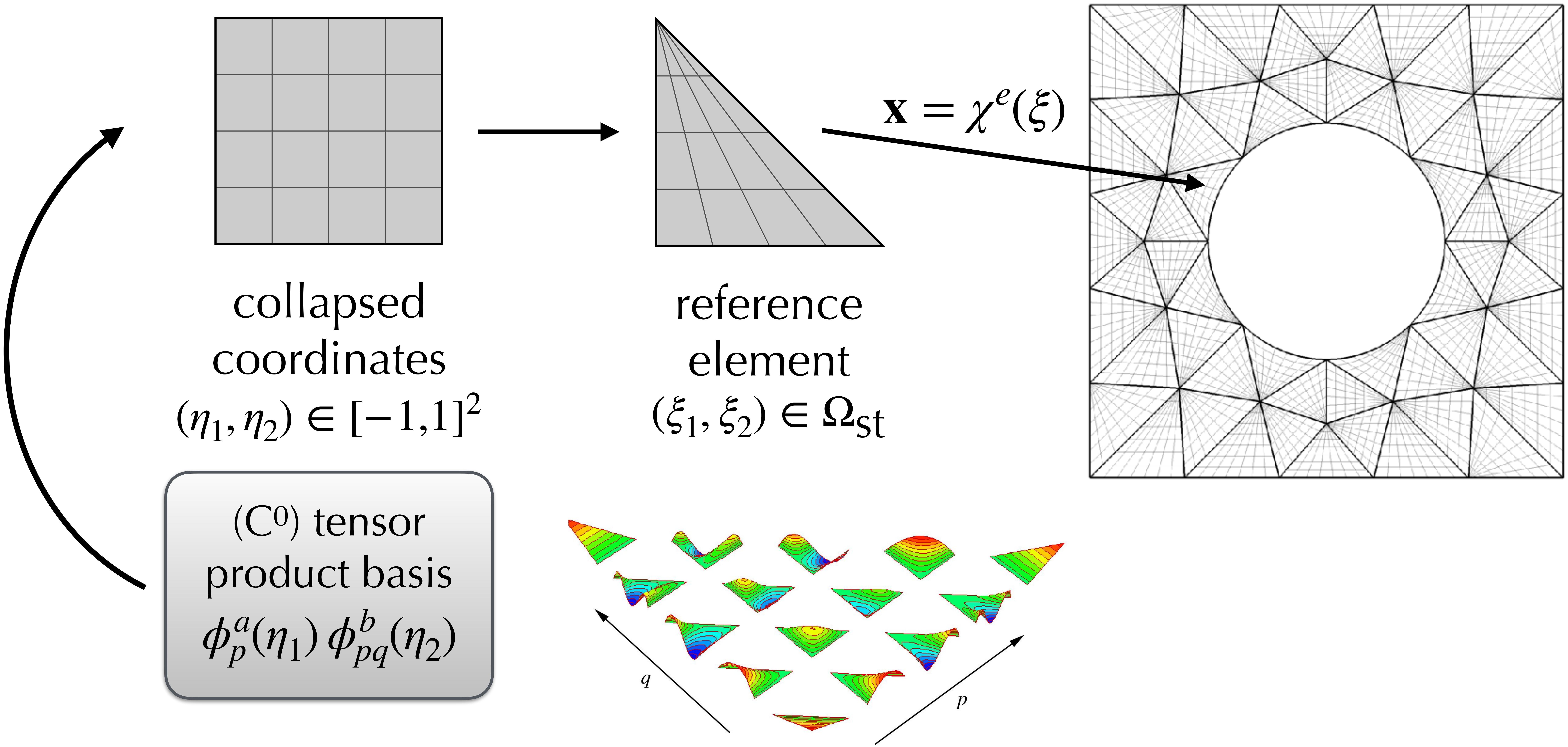
The spectral/*hp* element method

- Extend traditional FEM by adding higher order polynomials of degree P within each element.
- e.g. high-order triangle has $(P+1)(P+2)/2$ degrees of freedom at a given order P.
- Increased accuracy: more done per degree of freedom, increased **arithmetic intensity**.



spectral/*hp* element basis functions

The spectral/*hp* element method



Challenges

High order methods can provide a lot of benefits for these problems:

- ✓ very high accuracy over time/space;
- ✓ excellent for tracking transient structures;
- ✓ tuneable arithmetic intensity: overcome flops/byte barrier;
- ✓ geometric flexibility.

but there are also some challenges…

- ➤ implementation difficulty;
- ➤ extra work per degree of freedom means computational efficiency is important;
- ➤ mesh generation;
- ➤ other numerical challenges, e.g. scalable preconditioning.

"Defining" features of spectral/hp method

Generally *not* collocated

$$u(\xi_{1i}, \xi_{2j}) = \sum_{n=0}^{P^2} \hat{u}_n \phi_n(\xi) = \sum_{p=0}^{P} \sum_{q=0}^{Q} \hat{u}_{pq} \phi_p(\xi_{1i}) \phi_q(\xi_{2j})$$

quadrature points

modal coefficients

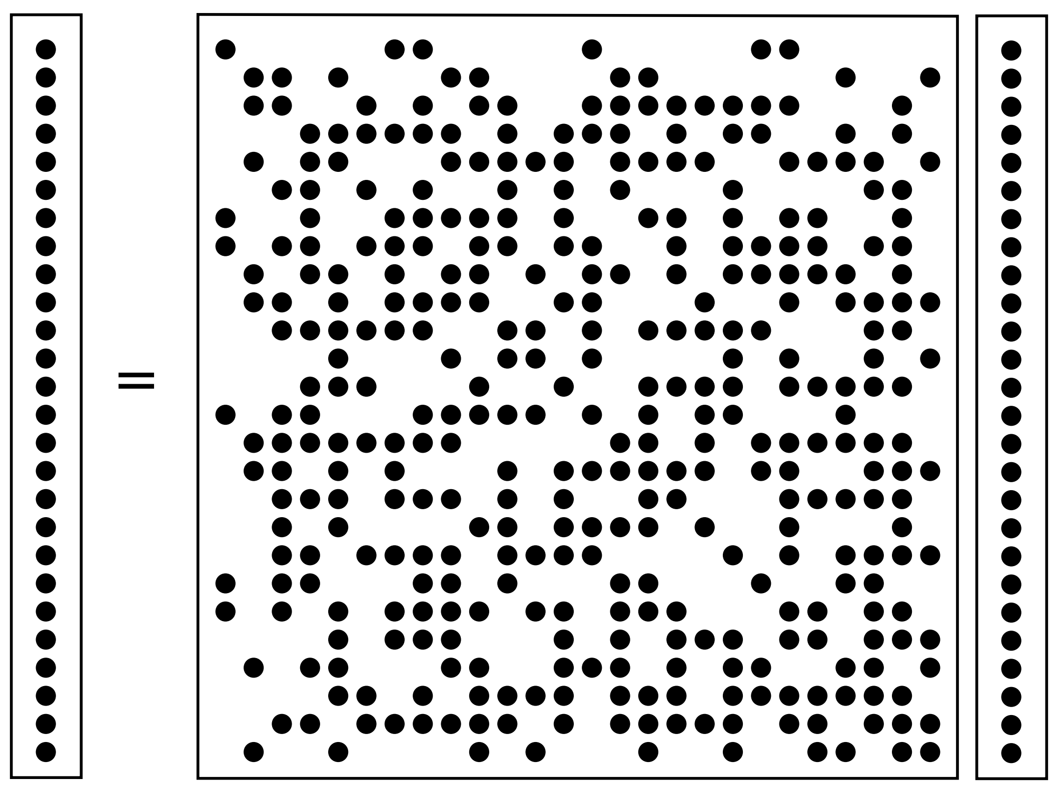
Uses tensor products of 1D basis functions, even for non-tensor product shapes

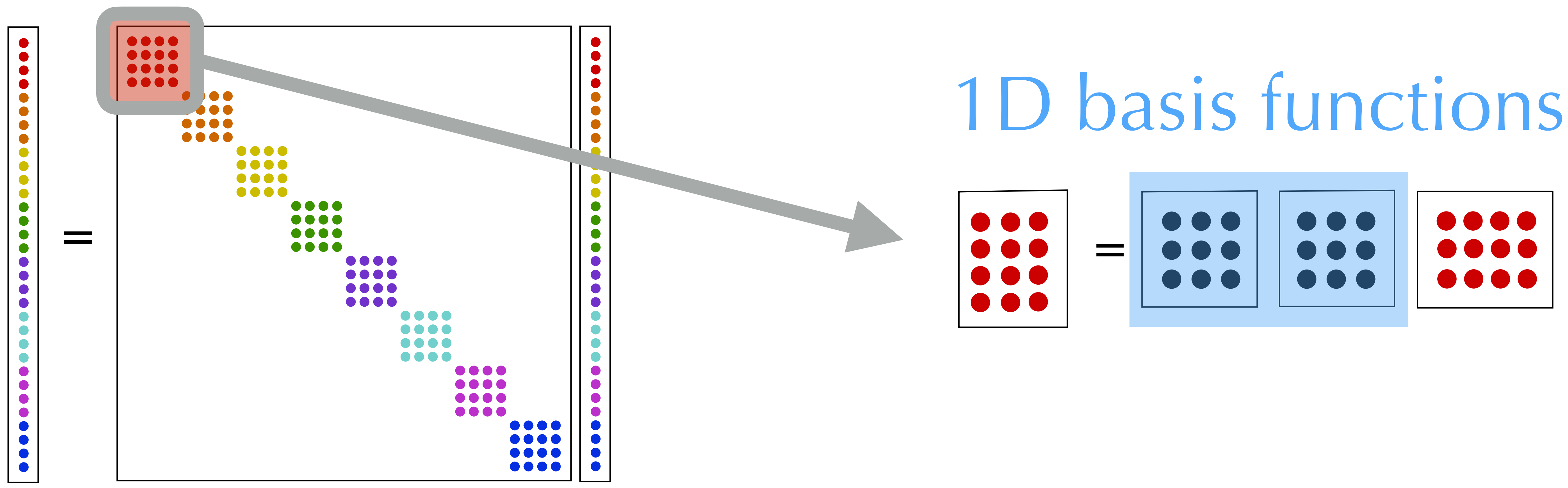
$$u(\xi_{1i}, \xi_{2j}, \xi_{3j}) = \sum_{p=0}^{P} \sum_{q=0}^{Q-p} \sum_{r=0}^{R-p-q} \hat{u}_{pqr} \phi_p^a(\xi_{1i}) \phi_{pq}^b(\xi_{2j}) \phi_{pqr}^c(\xi_{3k})$$

basis function indexing harder

Implementation choices

Finite element operation evaluations (e.g. mass matrix) form bulk of simulation cost; however can be evaluated in several ways.





Global matrix
assemble a sparse matrix

Local evaluation
create elemental dense matrices + assembly map

Matrix free
no local matrices at all
sum factorisation speedup

→ increasing arithmetic intensity

Sum-factorisation

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

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

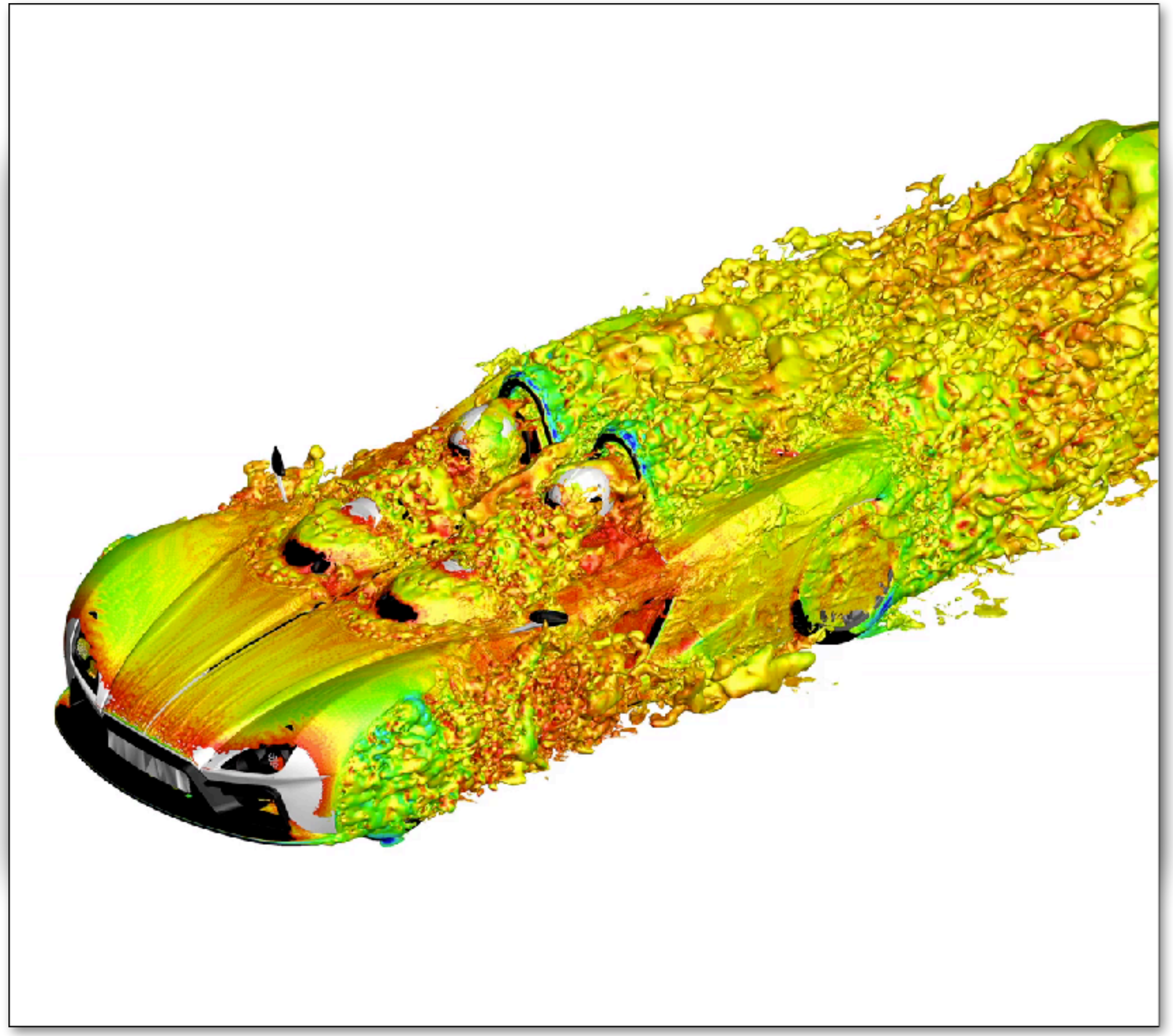
store this

Works in essentially the same way for more complex indexing for e.g. triangles:

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

store this

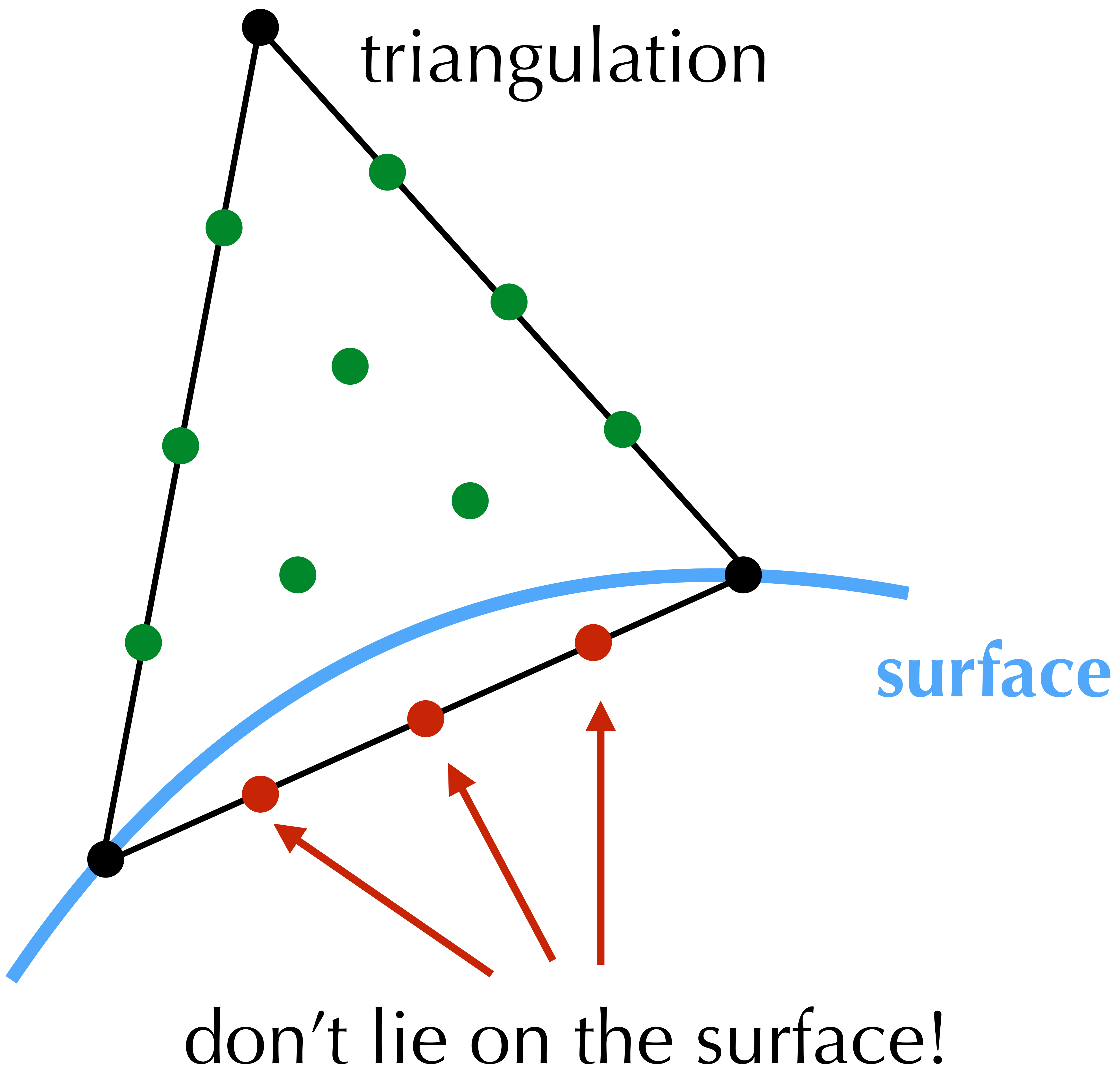
Unstructured simulations



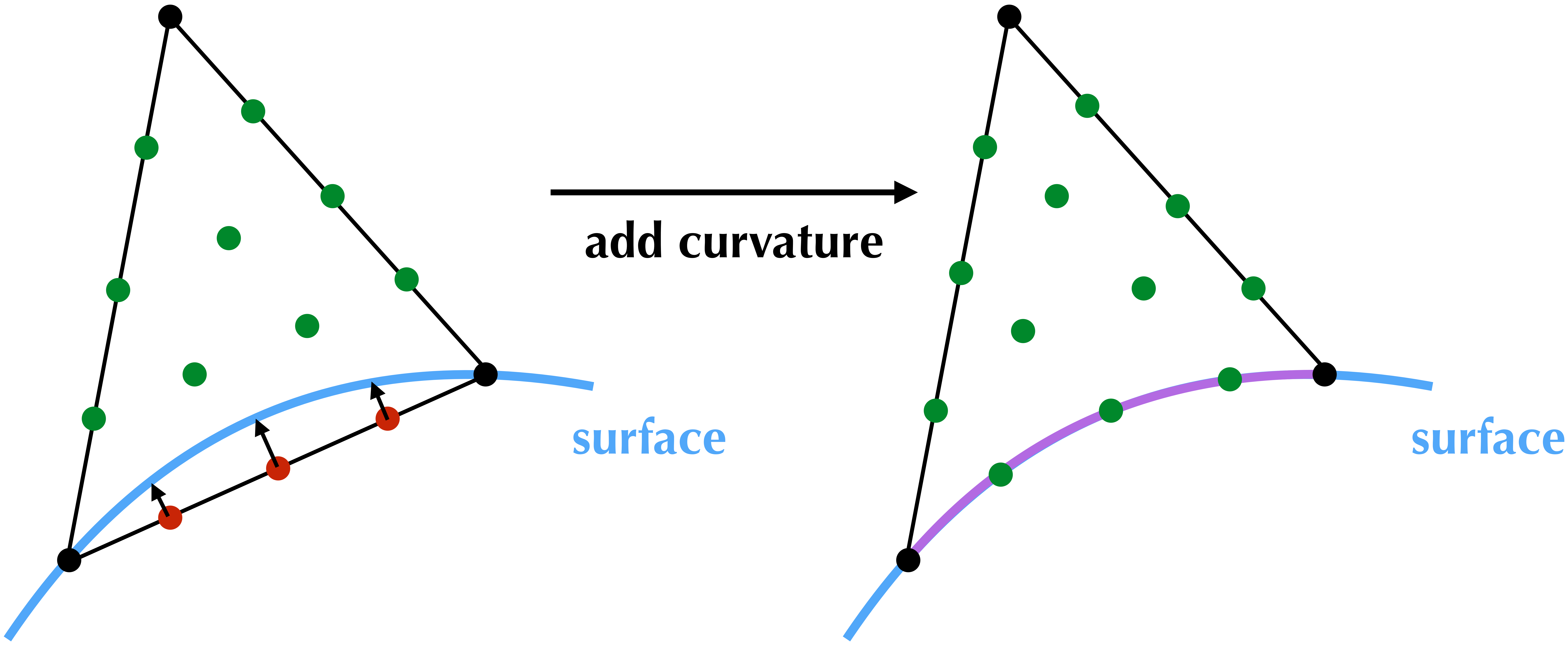
- Common knowledge: hex/quad elements yield best performance.
- However complex geometries presently require meshes of 'unstructured' elements: tets, prisms, etc.
- Potentially use tensor-product basis on unstructured elements, enable matrix-free operators + sum factorisation.

High-order mesh generation

- Good quality meshes are **essential** to simulation results.
- We can have a very fancy solver, but if there's no mesh then you can't run your simulation!
- At high orders we have an additional headache, as we must **curve the elements** to fit the geometry.



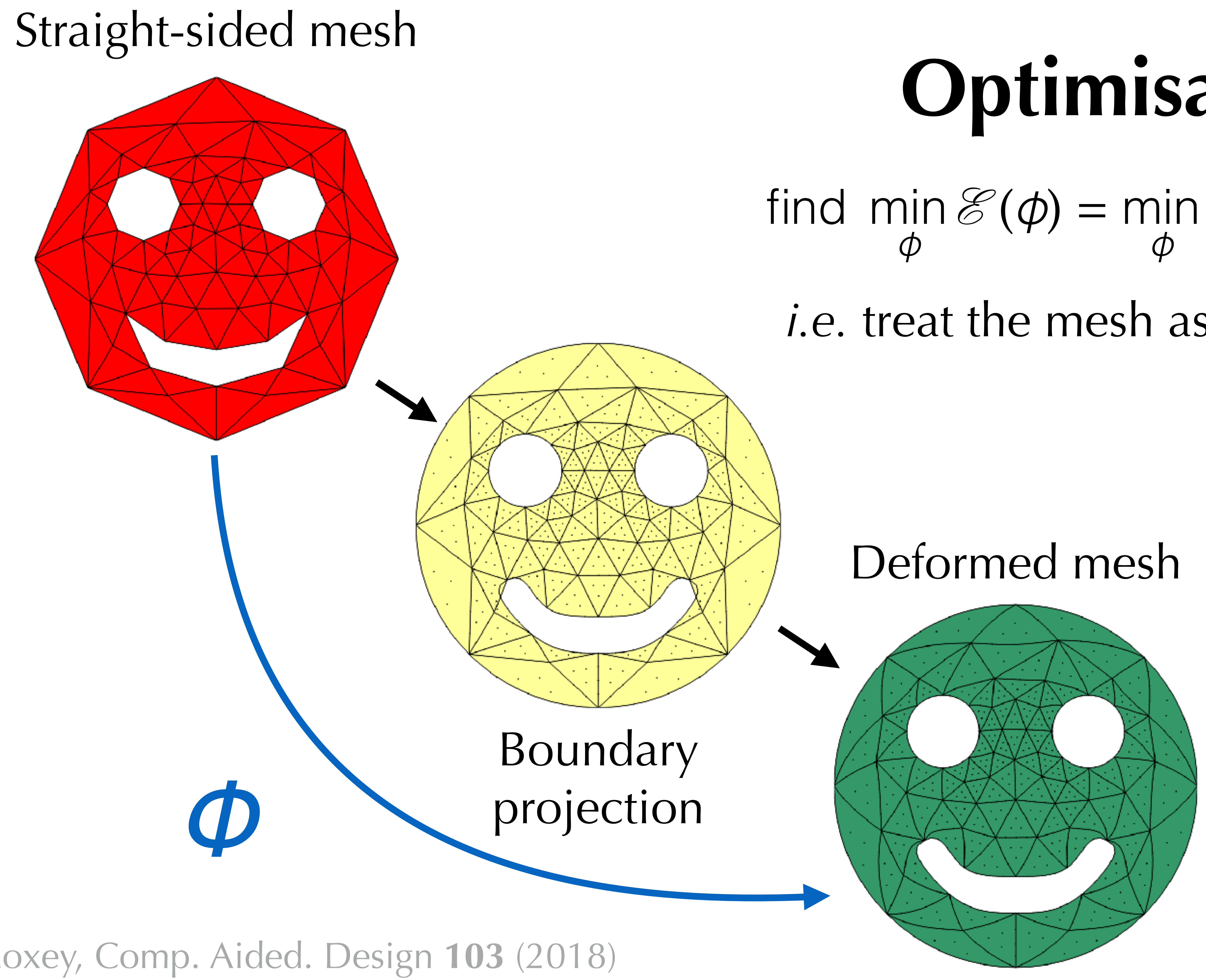
High-order mesh generation



Optimisation

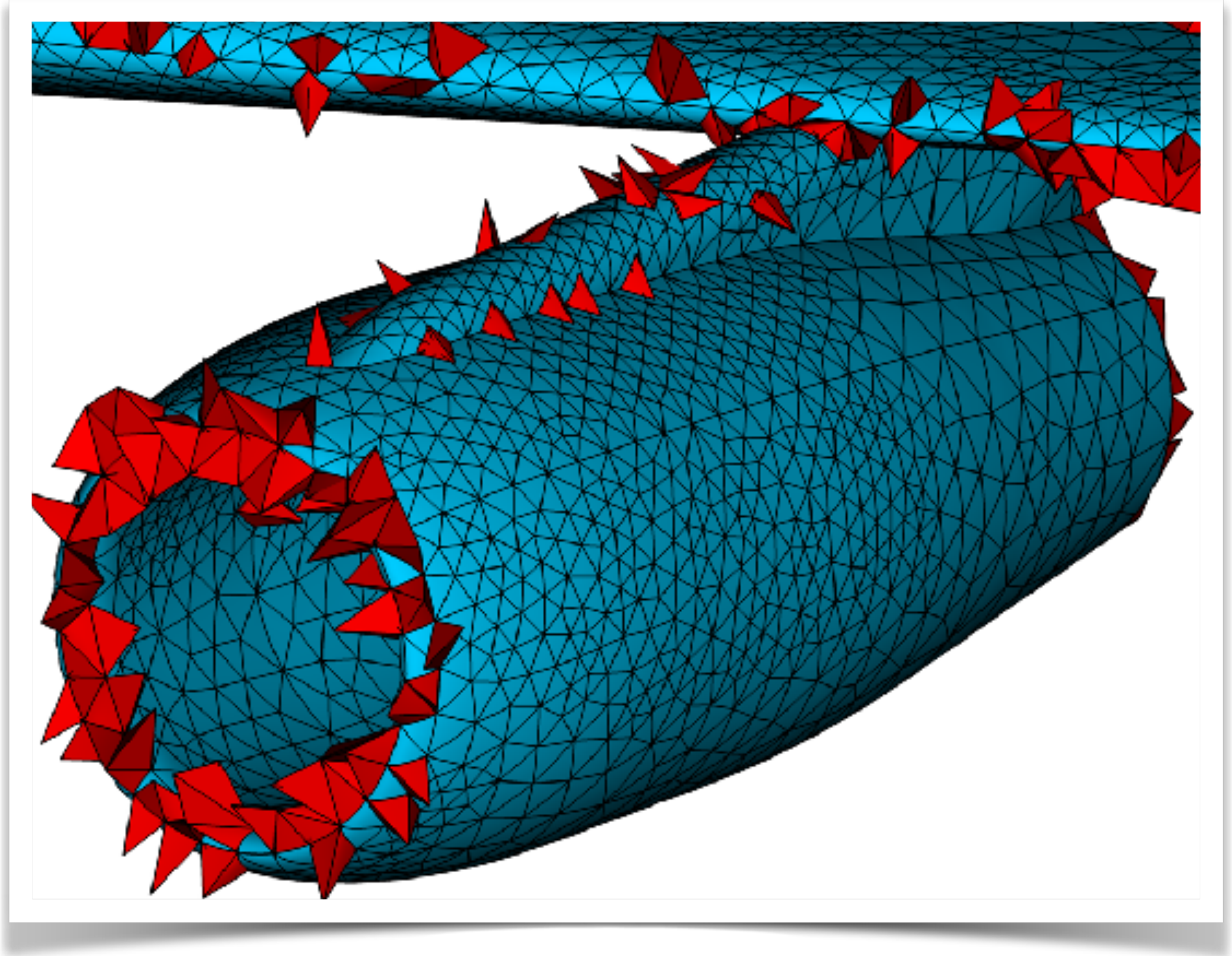
$$\text{find } \min_{\phi} \mathscr{E}(\phi) = \min_{\phi} \int_{\Omega_I} W(\nabla\phi)\, dy$$

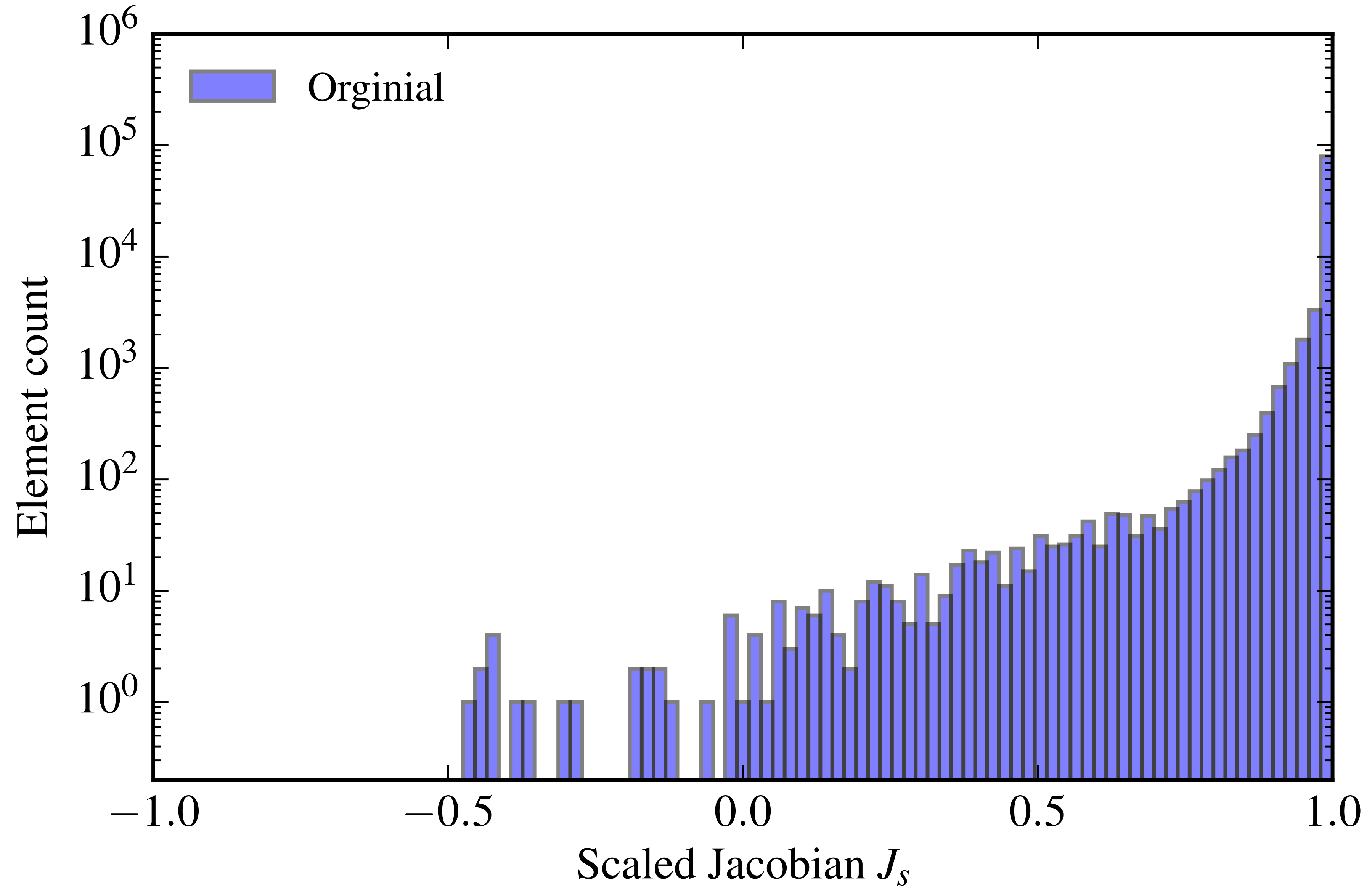
i.e. treat the mesh as a solid body

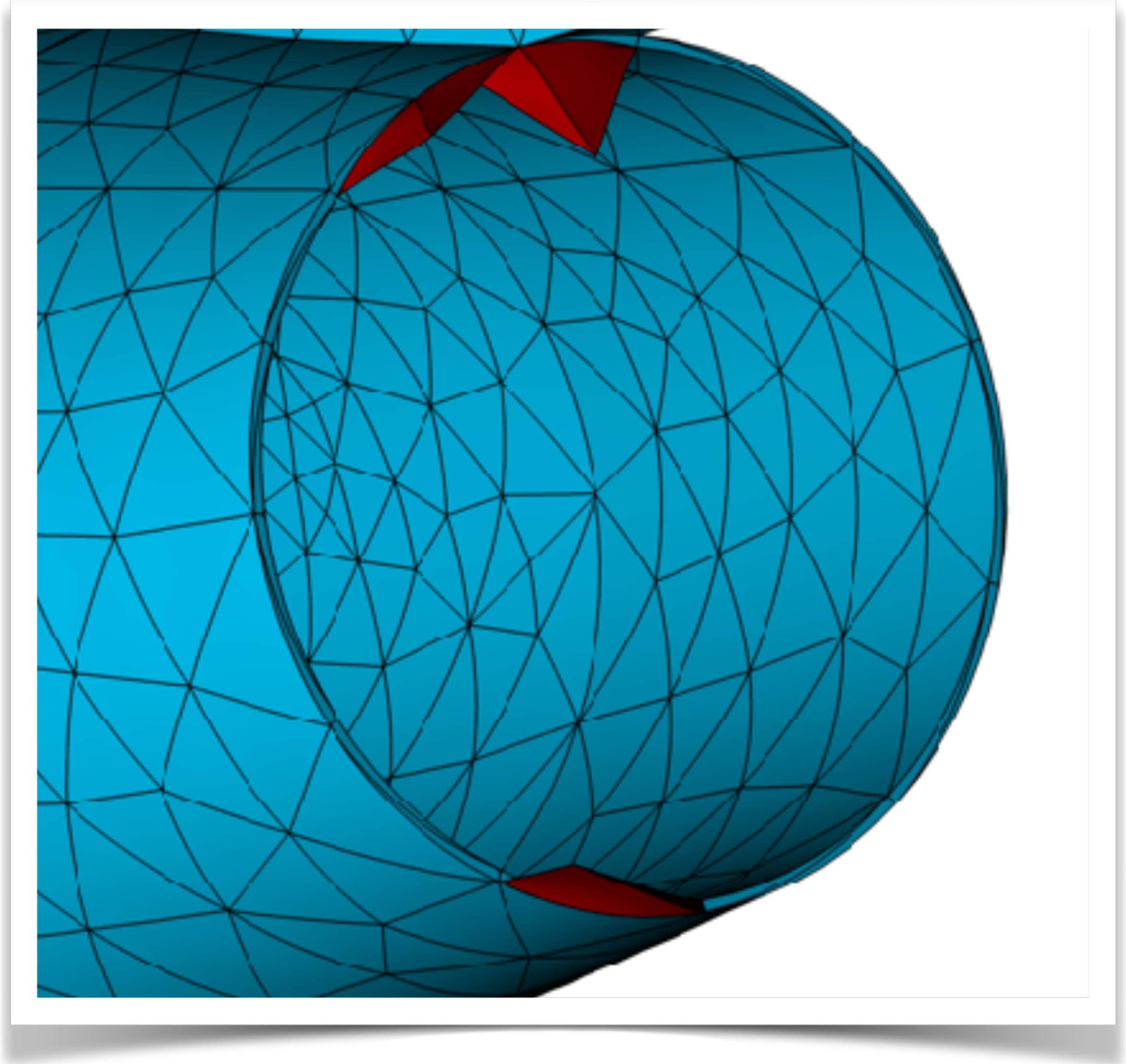


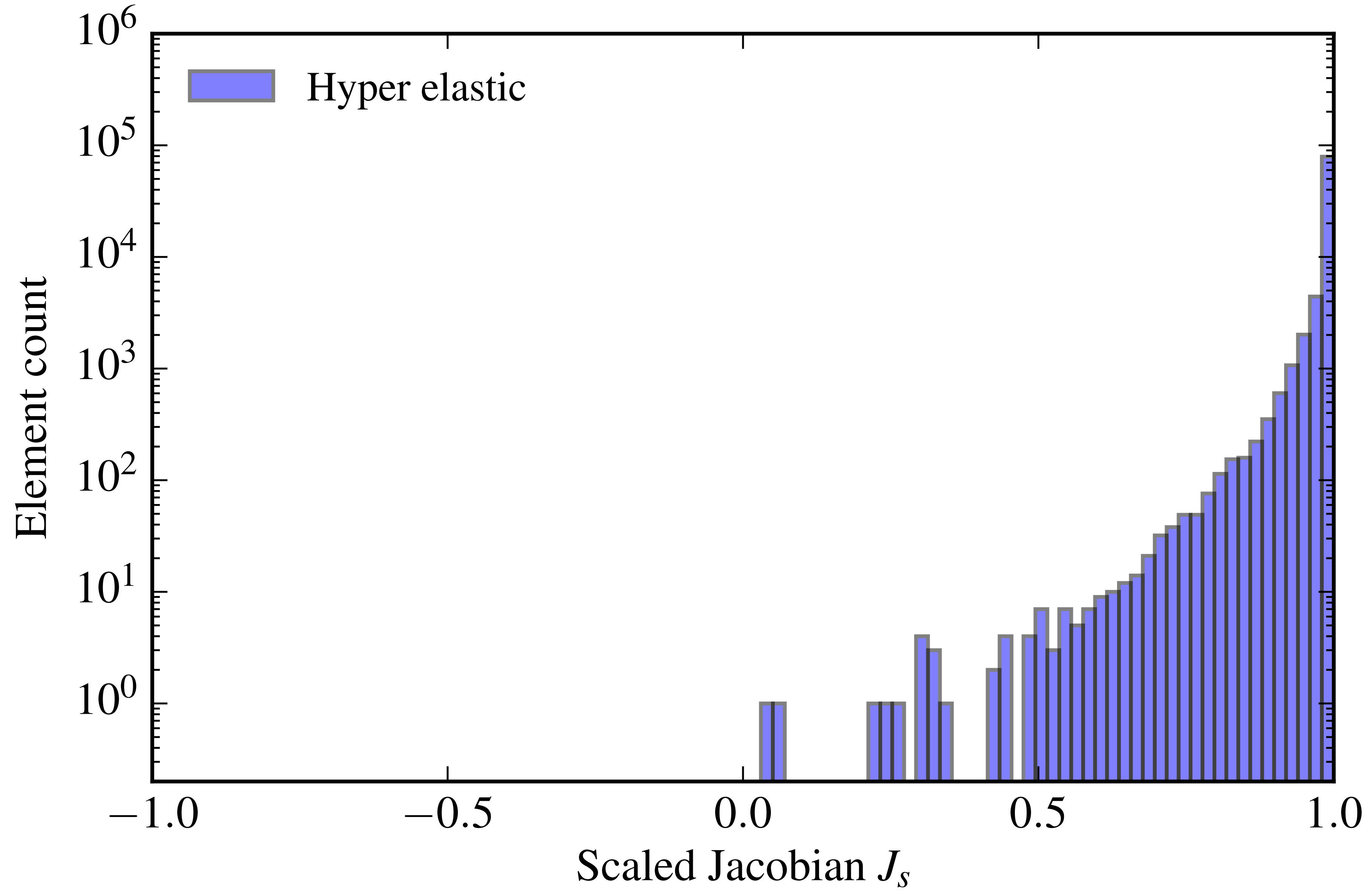
Turner, Peiro, Moxey, Comp. Aided. Design **103** (2018)

Example: DLR F6 engine

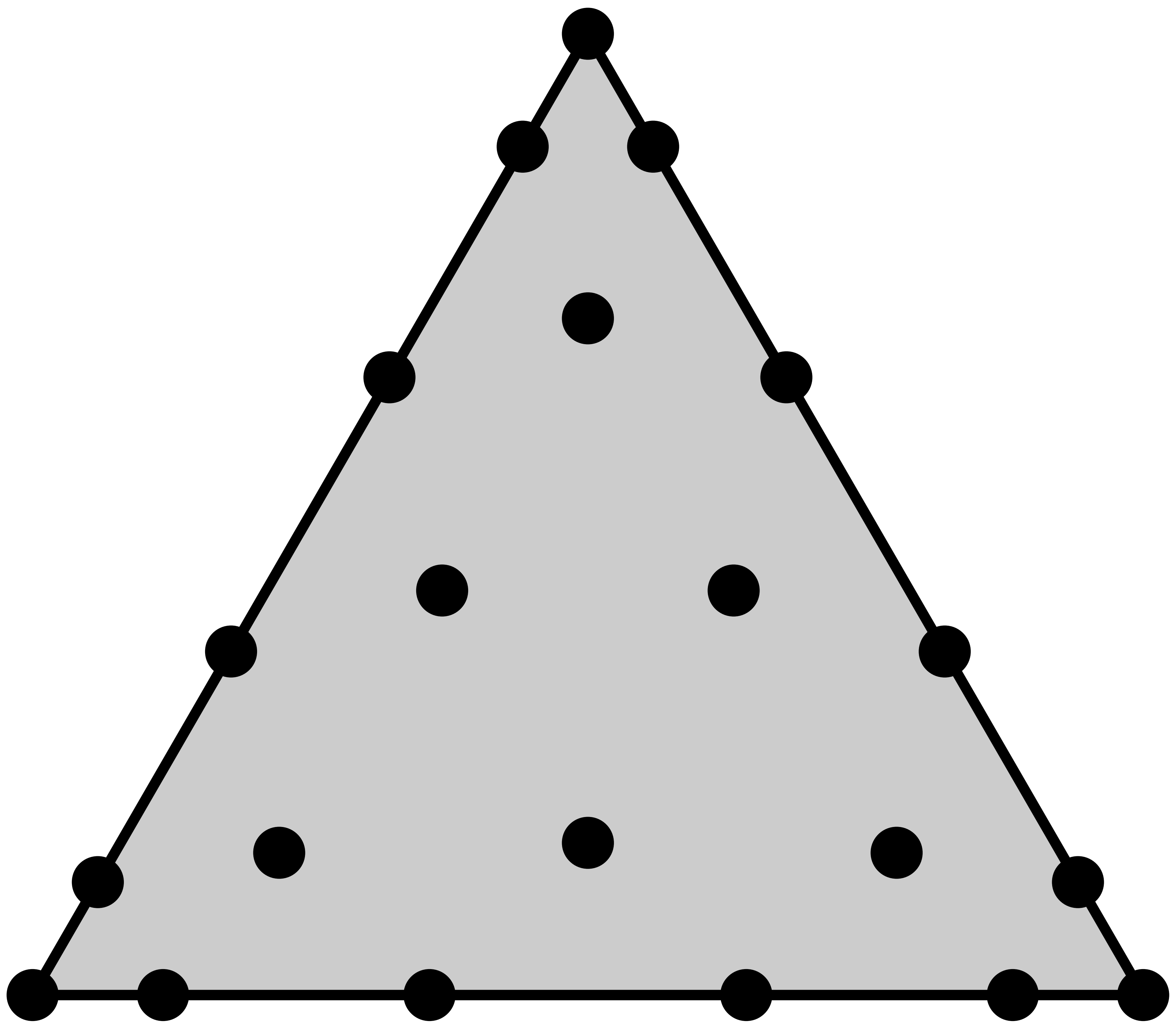








Unstructured elements



P5 triangle, Fekete points

- Unstructured elements generally make use of Lagrange basis functions.
- Combine this with a suitable set of cubature points: either collocated or not.
- However this loses tensor-products structure: i.e. no sum factorisation possible.

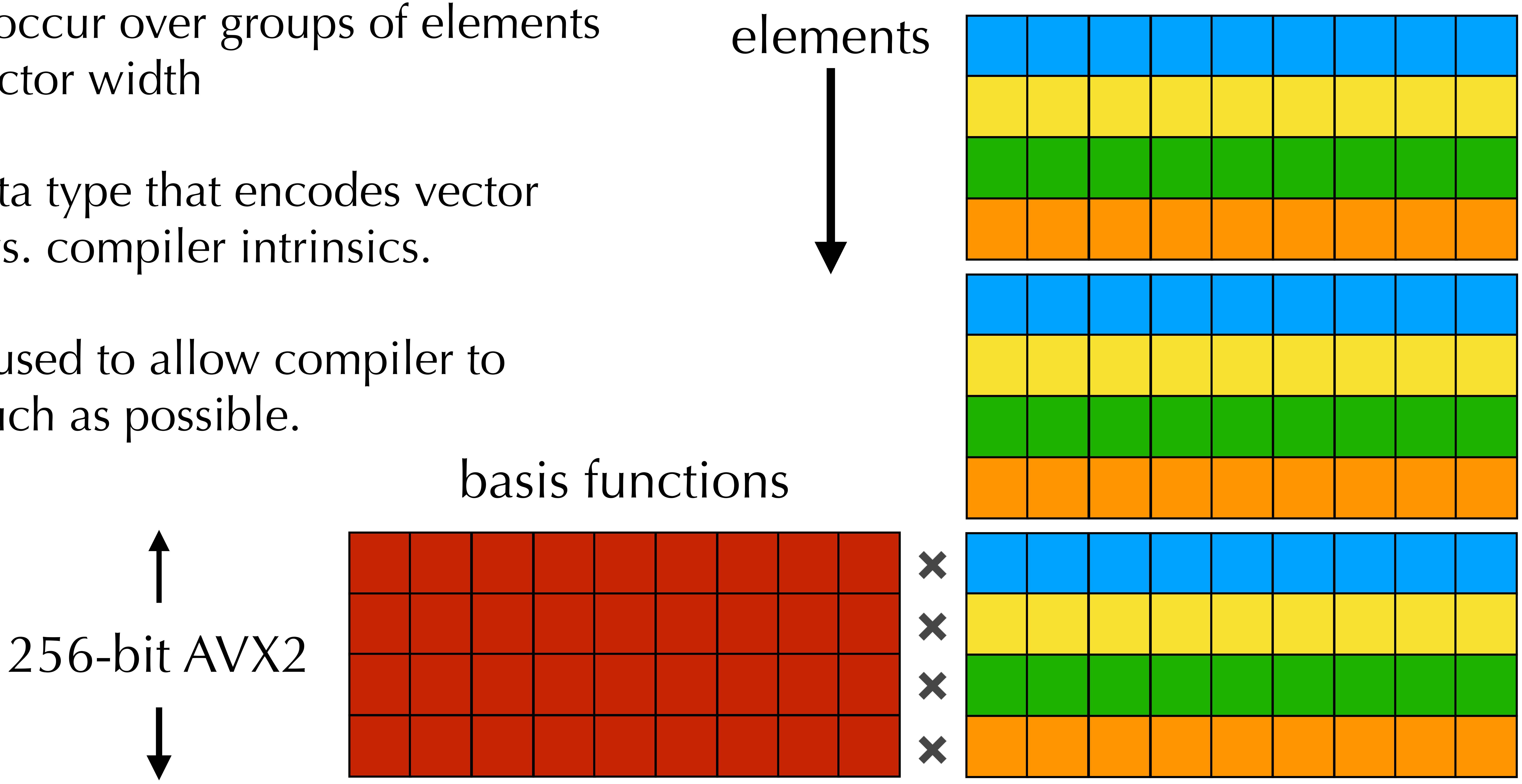
Key questions

- Under spectral/*hp* approach, sum-factorisation matrix-free operators are certainly possible for any element type. Some questions:
    - How much performance do we lose relative to hex/quad?
    - How should SIMD be used?
- Developed a benchmarking utility for Helmholtz operator to test viability of this approach: used for implicit solve in incompressible N-S equations.

$$\nabla^2 u - \lambda u = f(x) \quad \rightarrow \quad (\mathbf{L} + \lambda \mathbf{M})\hat{\mathbf{u}} = \hat{\mathbf{f}}$$

Data layout

- Operations occur over groups of elements of size of vector width
- Use C++ data type that encodes vector operations vs. compiler intrinsics.
- Templating used to allow compiler to unroll as much as possible.



Implementation particulars

- Hand-written kernels for each element type to implement three key components: **interpolation**, **derivatives** and **inner products/integration**.

- Written using C++: templating on (potentially heterogeneous) polynomial order, quadrature order and vector width.

- Also templates on **affine elements** (spatially-constant Jacobian) vs. **curvilinear** (spatially-varying); no consideration for Cartesian meshes.

- Templating gives *significant* improvements in runtime performance, particularly for complex loop structures found in this regime.

x86-64 CPU Tests

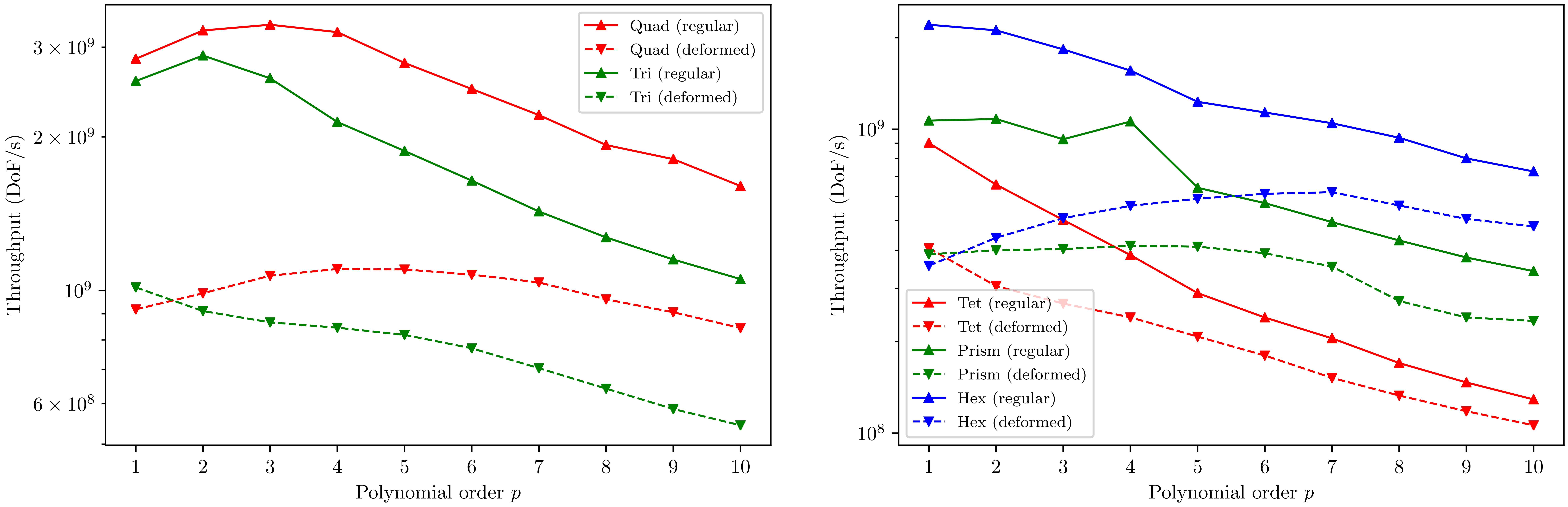
- Benchmarking of Helmholtz operator performed on two x86 architectures with varying SIMD widths.

	Broadwell (AVX2)	Skylake (AVX512)
Model	E5-2697 v4	Xeon Gold 6130
Clock speed(s)	2.3 / 2.0 GHz standard / AVX2	2.1 / 1.7 / 1.3 GHz standard / AVX / AVX512
Cores / sockets	18 / 2	16 / 2
Max node GFLOP/s	1,152	870 (AVX2) 1,331 (AVX512)

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: C^0 work in progress.

Throughput (AVX2, Broadwell)

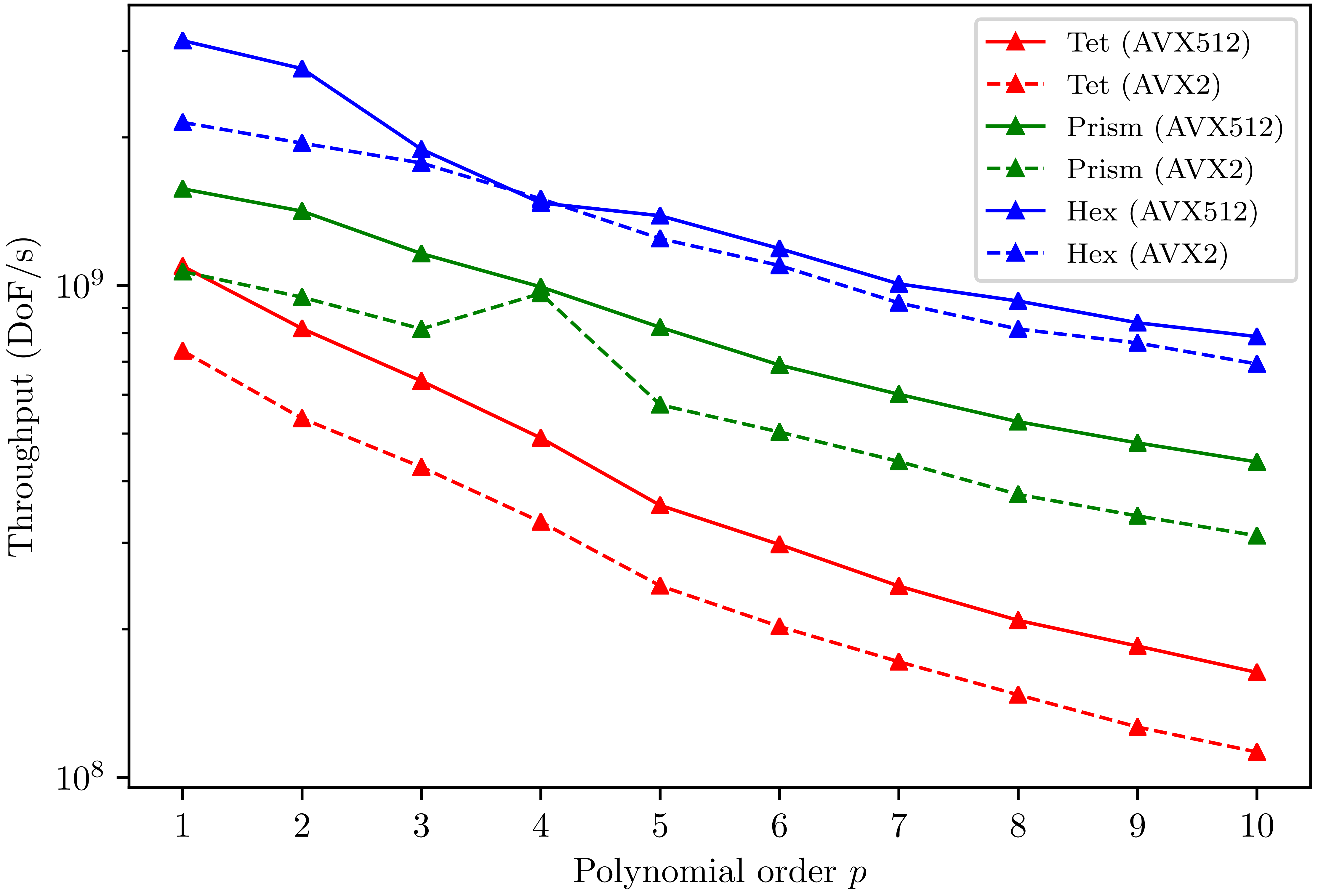


2D: Quads, triangles

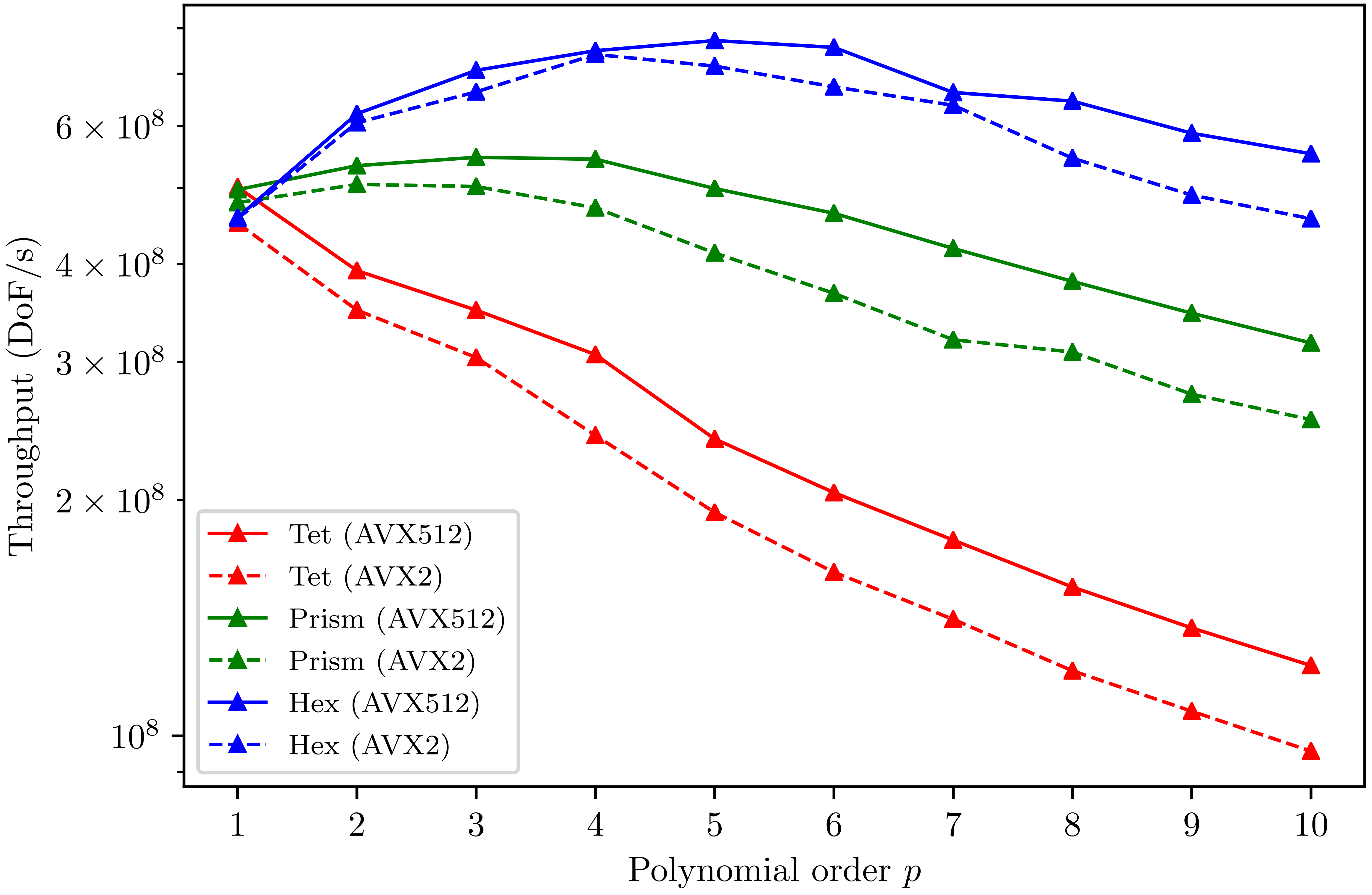
3D: Hexahedra, prisms, tetrahedra

Moxey, Amici, Kirby, SIAM J. Sci. Comput., **42** (3), 2020

Throughput (AVX512/AVX2, Skylake)



3D: 'Regular' elements



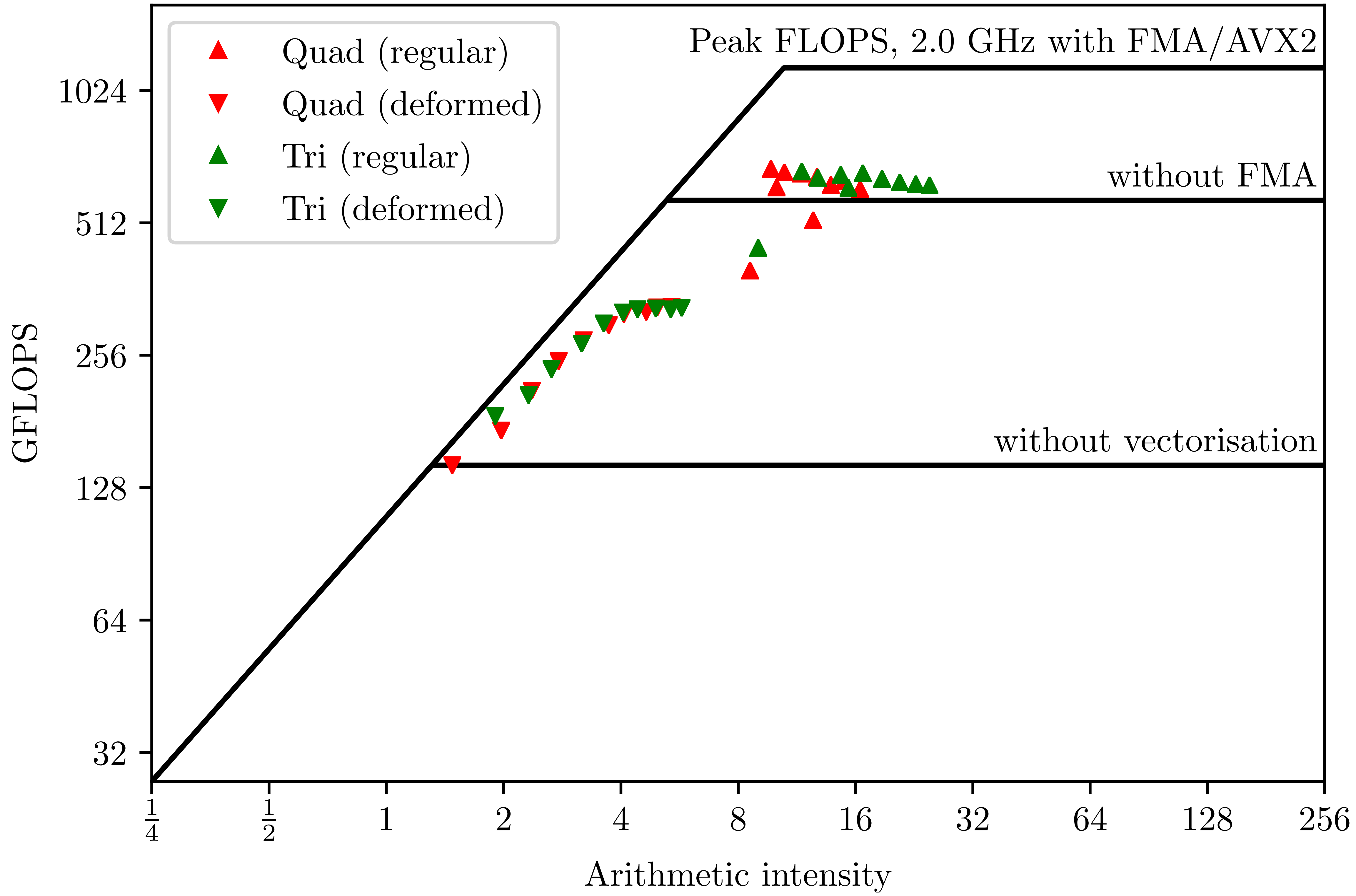
3D: 'Deformed' elements

Moxey, Amici, Kirby, SIAM J. Sci. Comput., **42** (3), 2020

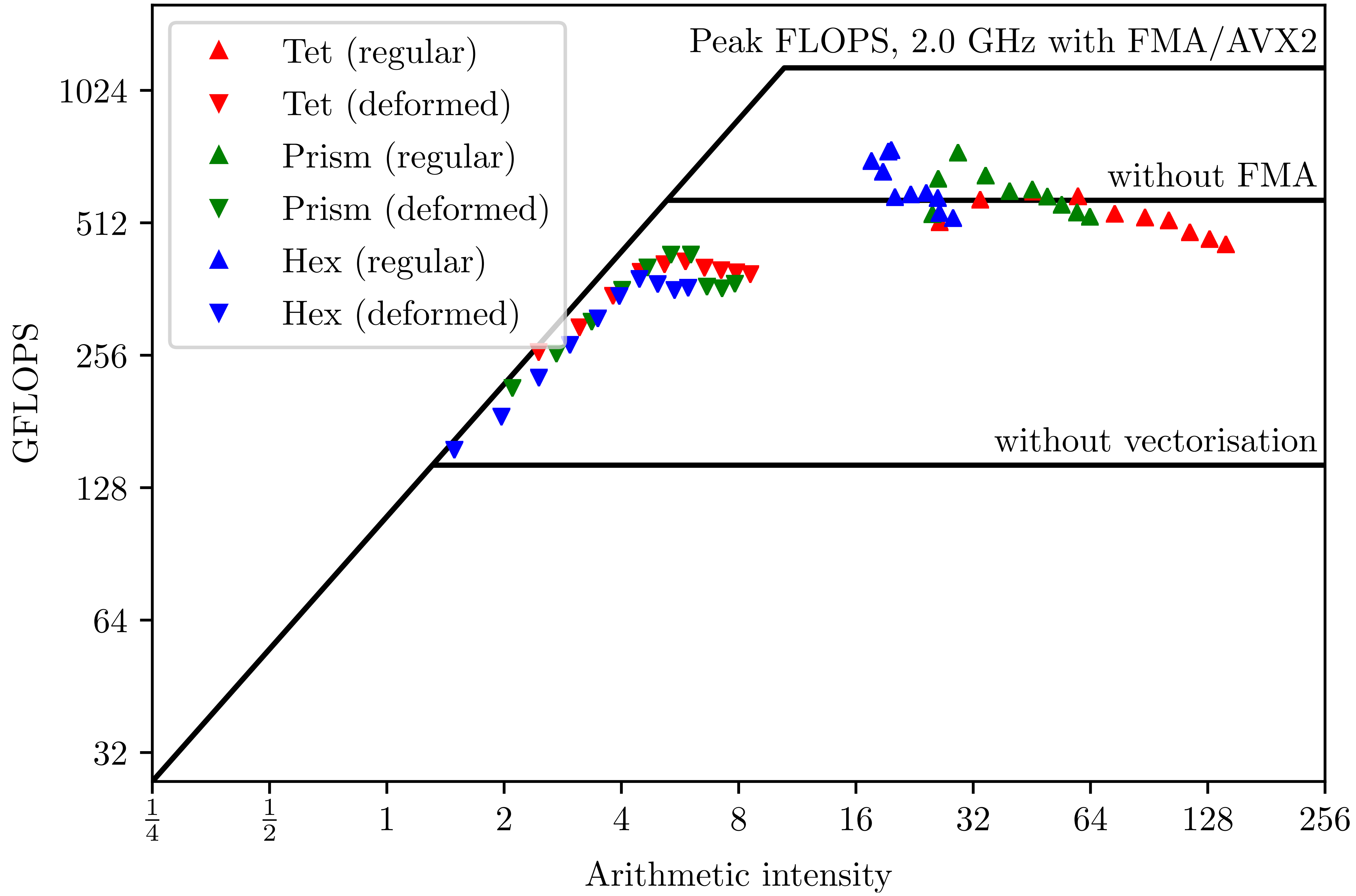
Some clear trends

- This behaves pretty much as you might anticipate:
    - For regular elements, clear hierarchy of element type/dimension, where throughput is lost as dimension/complexity of indexing increases.
    - Regular elements outperform deformed elements due to increased memory bandwidth.
    - Relative performance gap between deformed elements decreases at moderate polynomial orders.

Roofline results



2D: Quads, triangles



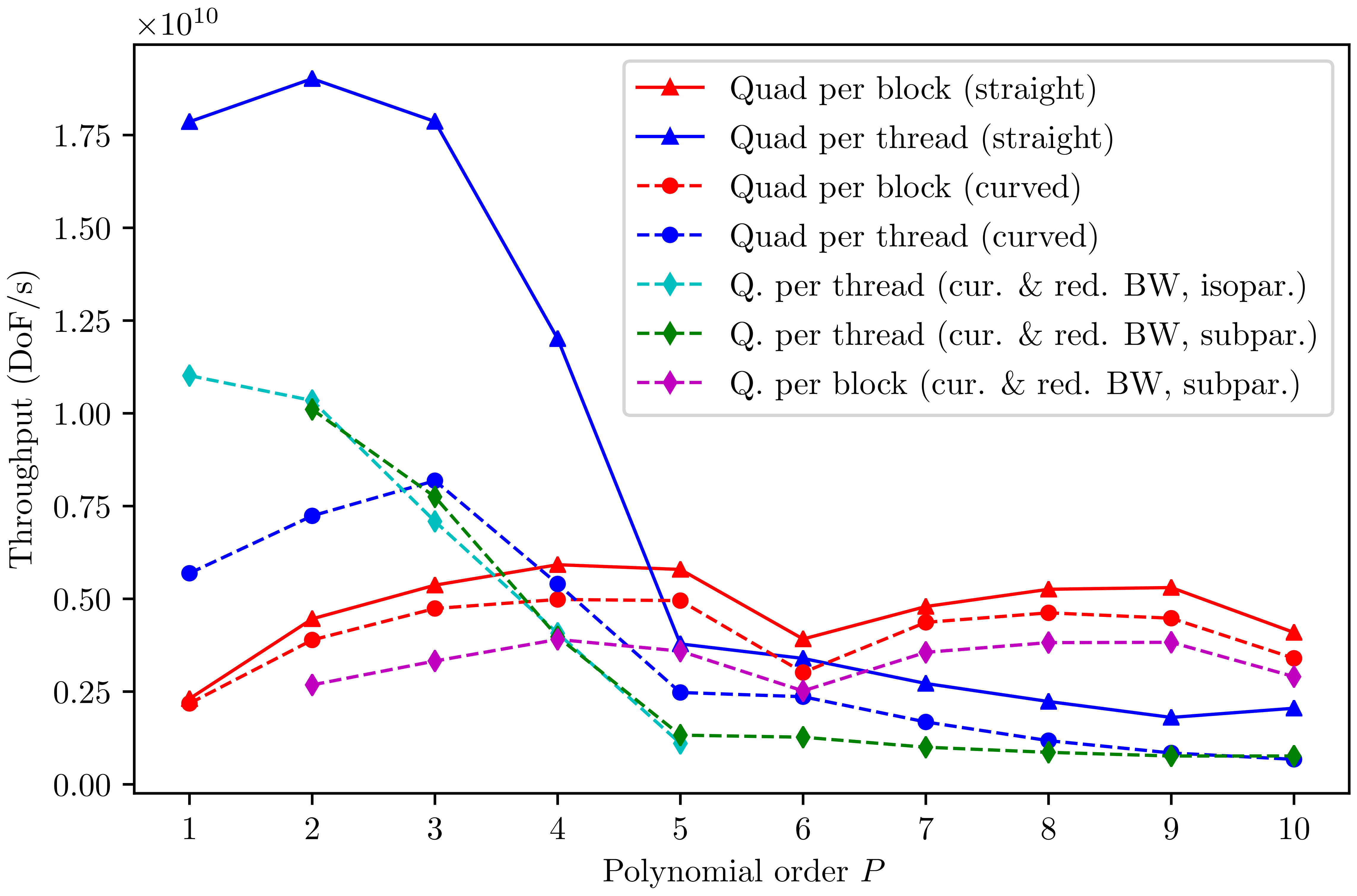
3D: Hexahedra, prisms, tetrahedra

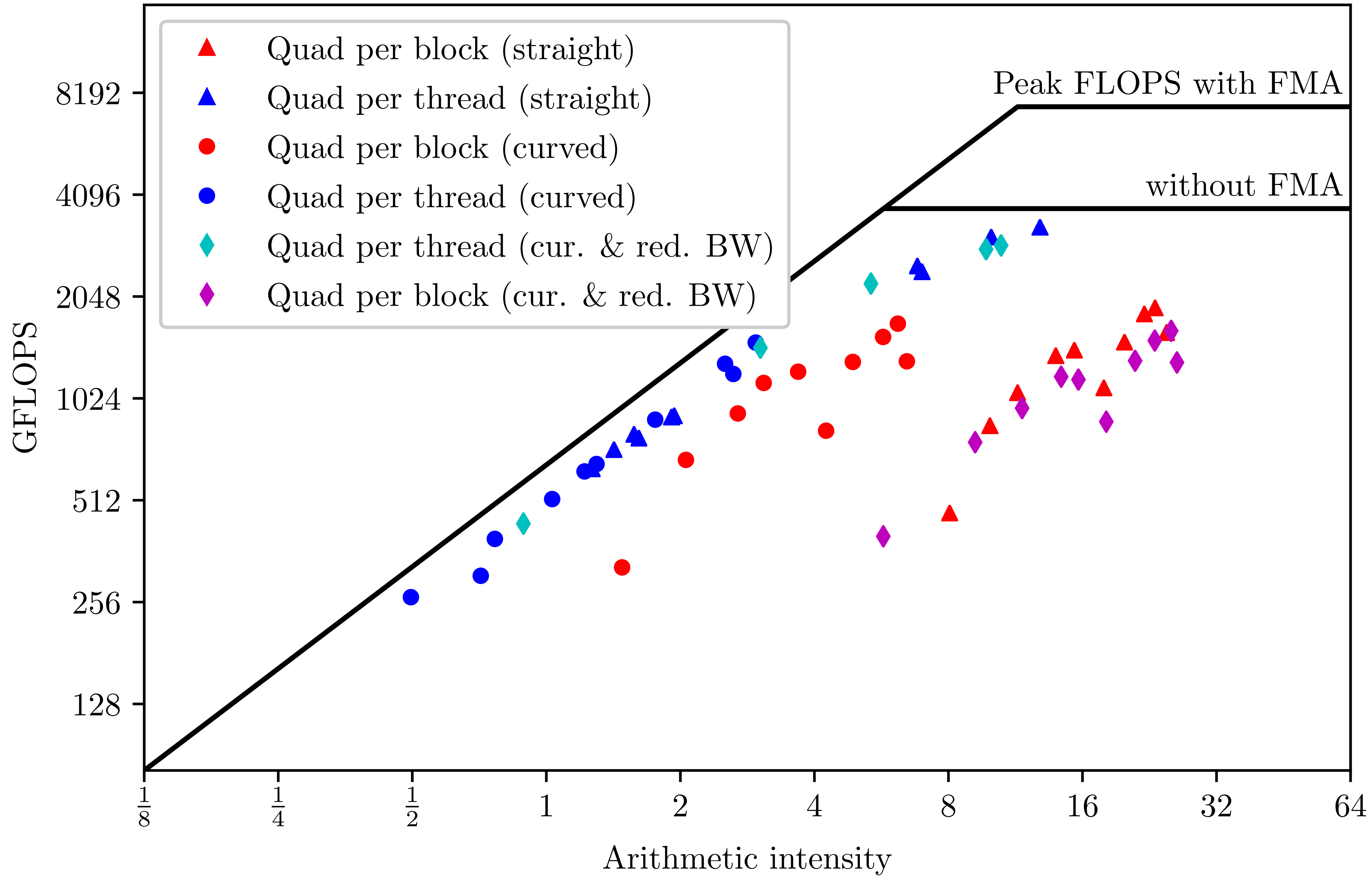
Moxey, Amici, Kirby, SIAM J. Sci. Comput., **42** (3), 2020

What about GPUs?

- More of a work in progress. Good indication that these techniques translate to GPU architectures, but more care required.
- Central issue is that as vector width increases, so too does cache pressure.
- Therefore a need for multiple strategies as polynomial order increases:
  - *per thread* parallelism: one element per thread, handles work at all solution points in the element (as in CPU tests).
  - *per block* parallelism: one element per SMX unit, then one thread per solution point.

Results: quadrilateral elements





Results from Titan V benchmarking
Similar trends for triangular elements

Eichstadt, Peiró, Moxey,
to be submitted

Summary

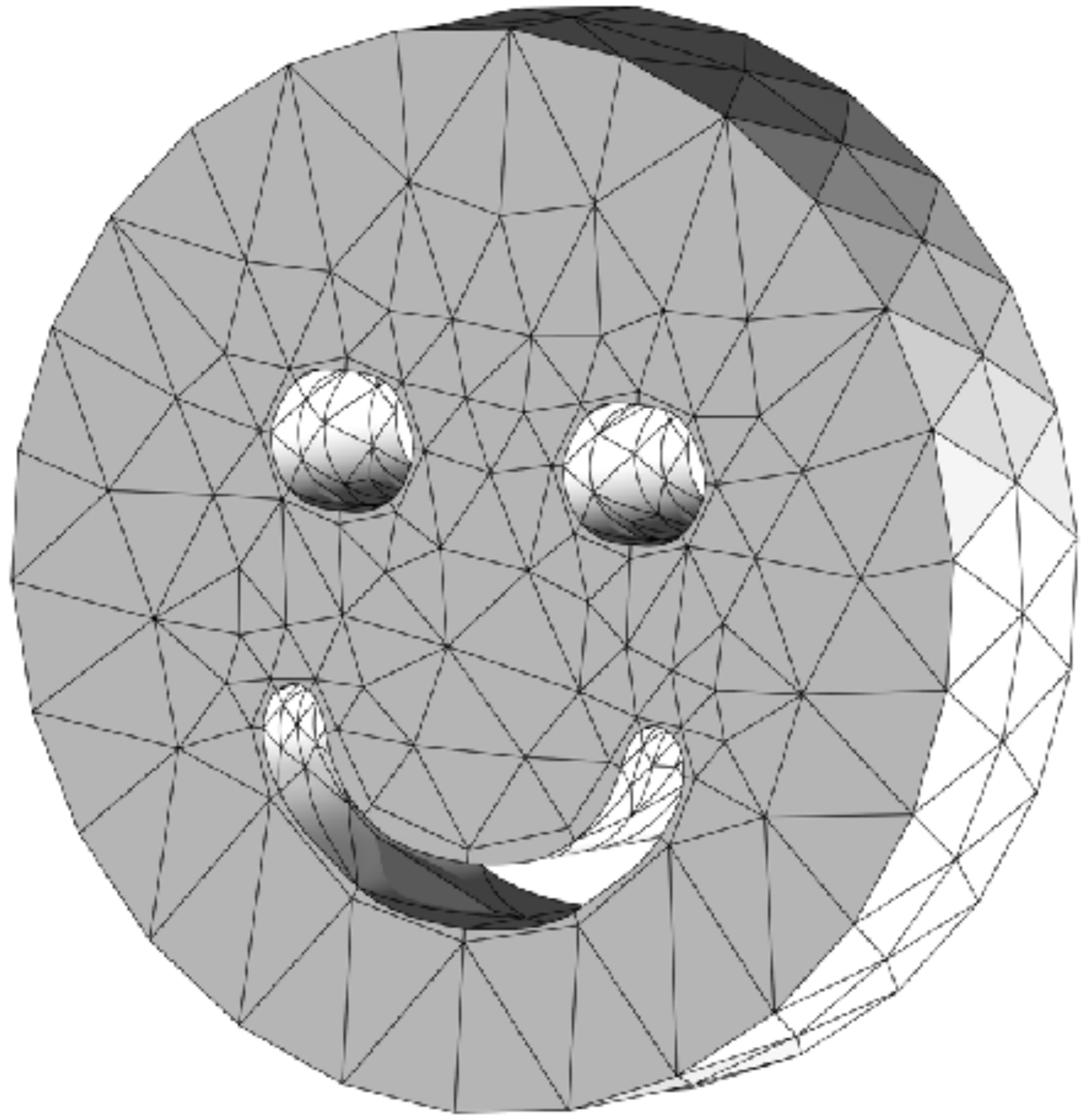
- Efficient matrix-free implementations of key finite element operators are certainly achievable on modern architectures for ‘unstructured’ elements.
- Inevitable drop in performance from quads/hexahedra: complexity of indexing, additional cache pressure, etc.
- However relative performance of e.g. hex/prism and quad/tri is actually pretty good, particularly for deformed elements; important for e.g. boundary layer problems with large proportion of BL prisms.
- Mesh generation still a key problem, GPU in 3D still to do.

Thanks for listening!

https://davidmoxey.uk/

david.moxey@kcl.ac.uk
d.moxey@exeter.ac.uk

www.nektar.info



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]