National Aeronautics and Space Administration

Software Design of the HyperSolve CFD Tool

Clark Pederson
NASA Langley Research Center

STO-AVT-358 Lecture Series
18-21 May 2026
The von Karman Institute for Fluid Dynamics

Standing on the shoulders of giants

I owe a great debt to the developers who trained me:

Dr. Kyle Thompson
Research AST,
NASA Langley Research Center

Dr. Matthew O'Connell
Research AST,
NASA Langley Research Center

Dr. Bil Kleb
Senior Researcher,
NASA Langley Research Center

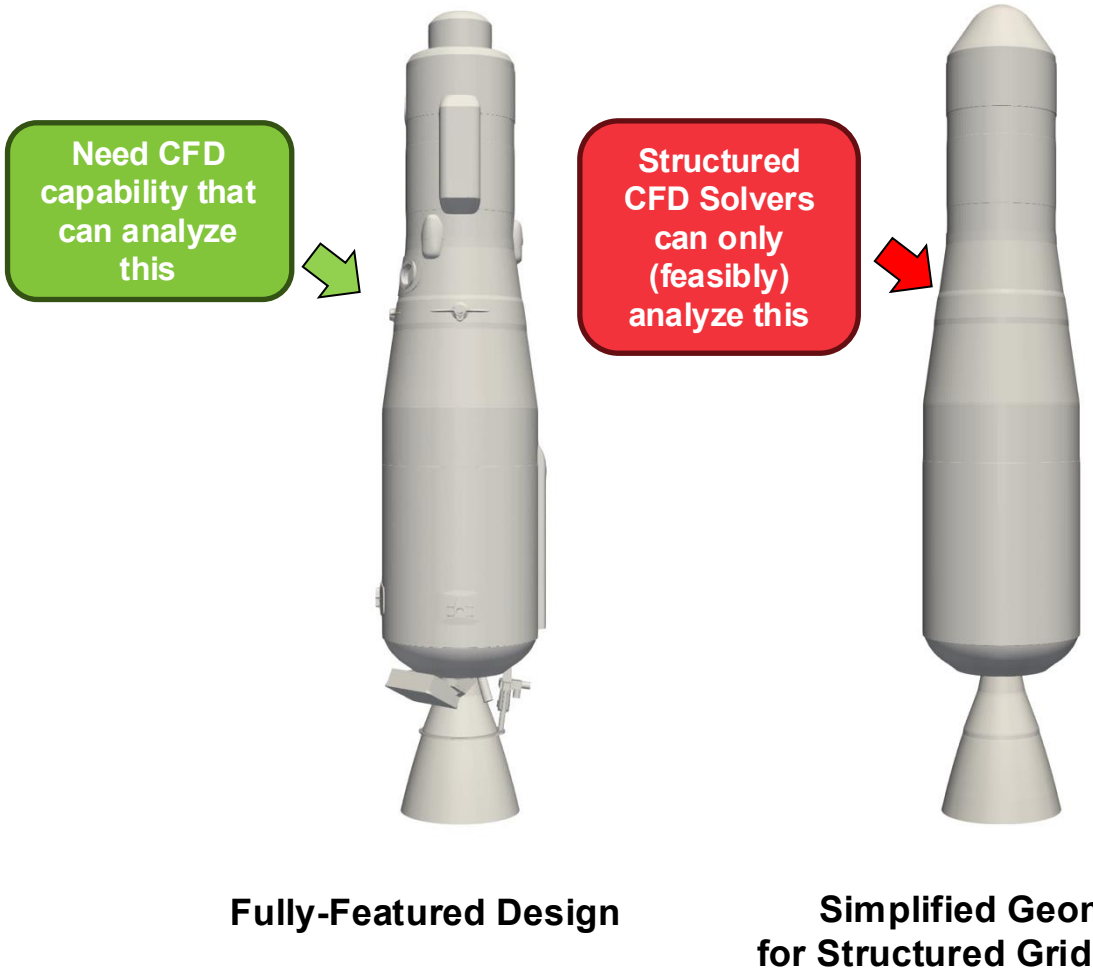
Dr. Cameron Druyor
Formerly NASA
Now at NextSilicon

Dr. Mike Park
Formerly NASA
Now at Flexcompute

Lecture Outline

1. Aeroheating on Complex Geometry
   - Motivation for HyperSolve
   - Sketch-to-Solution Workflow
2. HyperSolve Numerical Methods
3. HyperSolve Nonlinear Solver
4. HyperSolve Software Design and Software Principles
   - SOLID design prinicples
   - Testing infrastructure

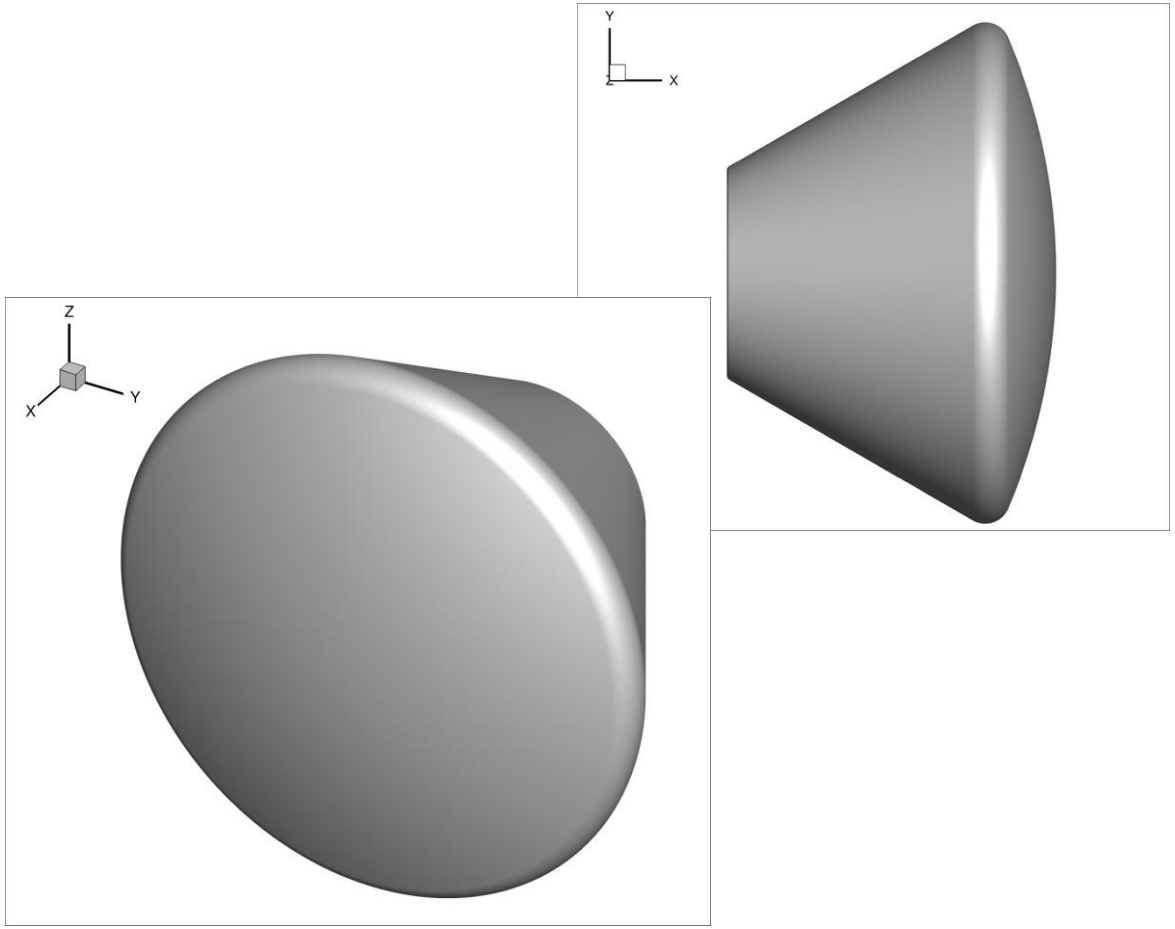
Motivation



- For design studies, we need quick turn around on a variety of geometries.
- Projects frequently ask about the heating on complex geometry:
  - Protuberances
  - Cavities
- Manually creating structured grids is time-consuming and error-prone.

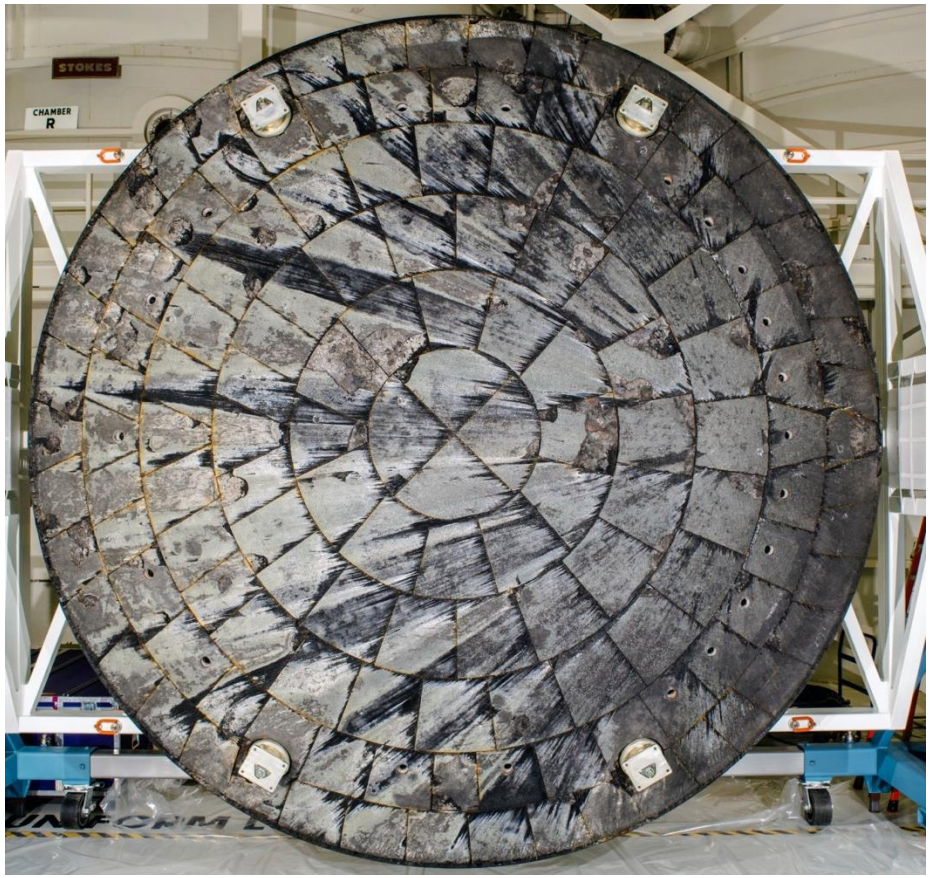
Motivation: Orion Heatshield

Simplified design



Simplified axisymmetric Orion heatshield

Reality

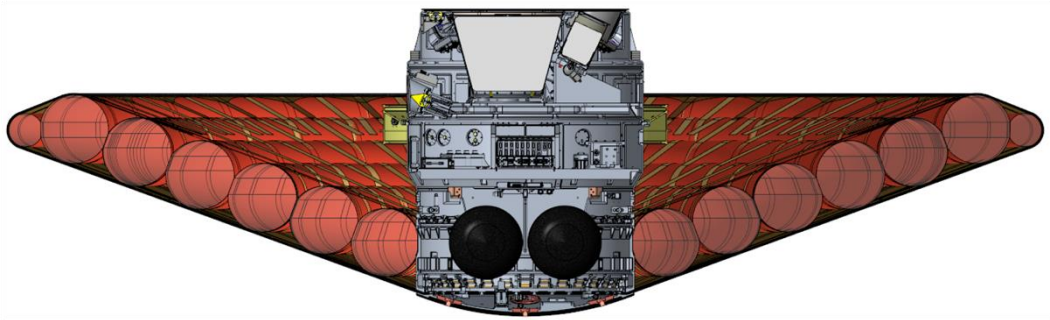


Actual heatshield from Artemis I mission
Credit: NASA

Challenge: Aeroheating on Complex Geometry

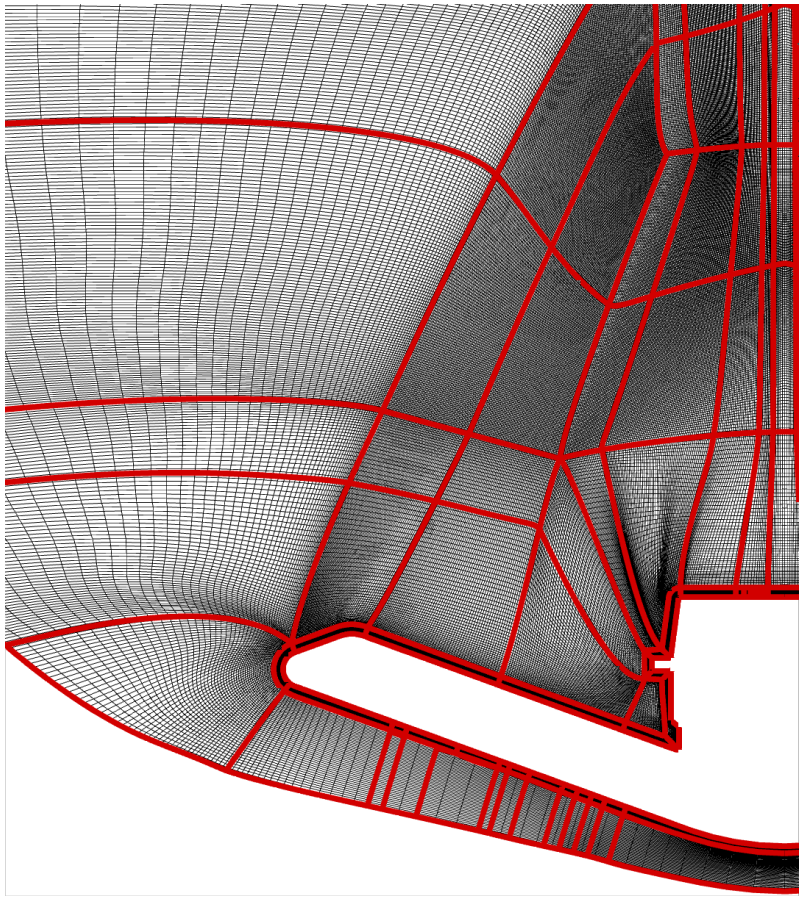
Multiblock Structured CFD

- Consistent accuracy
- “Artisan grids:” novel block topologies enable meshing nonsimple geometry
- Labor-intensive mesh generation



Cross-section of the LOFTID vehicle.

“Stacked Block” Structured Mesh Example



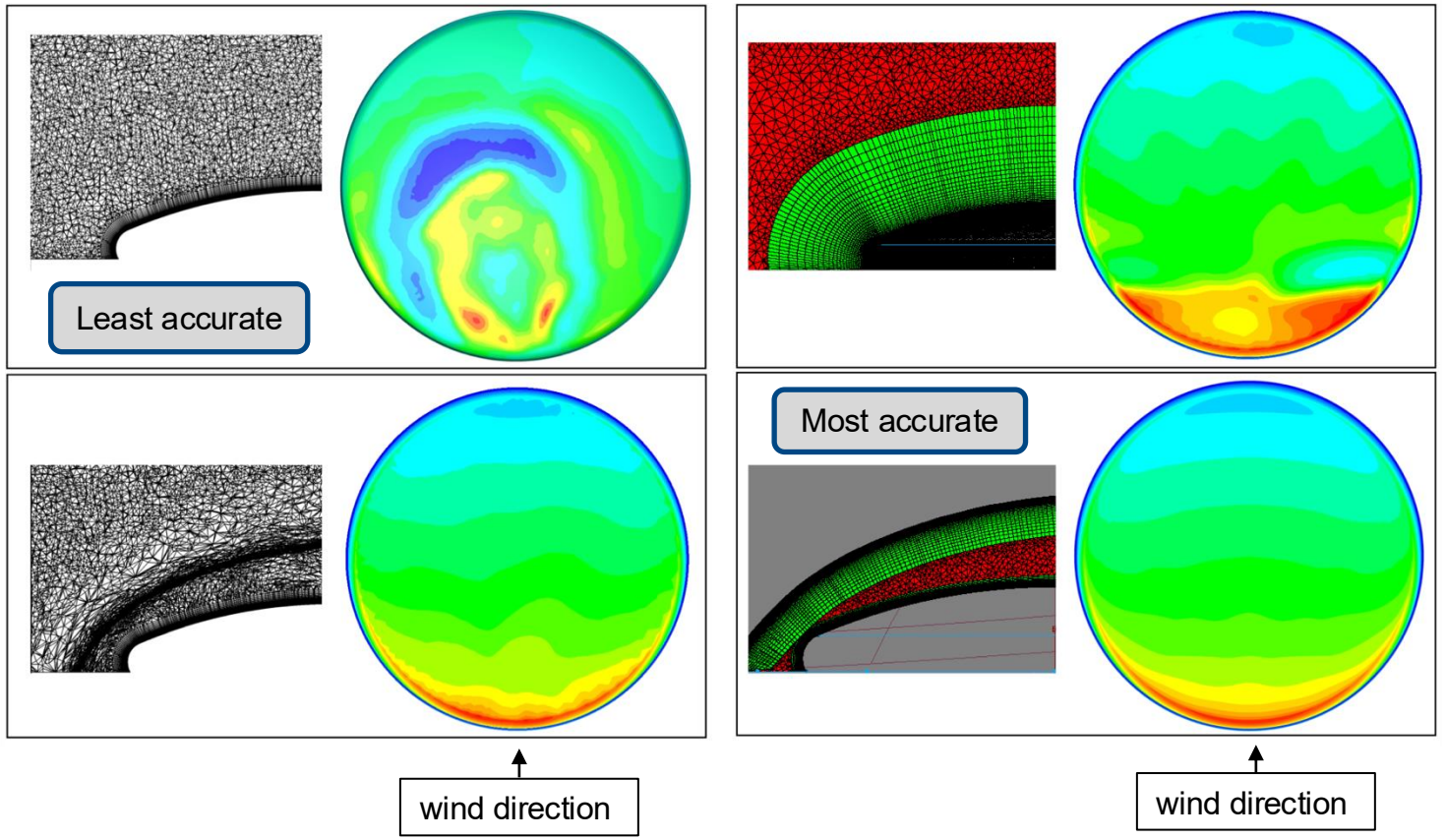
Thompson, Kyle B., Adam J. Wise, and Brian R. Hollis. "Improvements to the LAURA Mesh Adaptation Algorithm." *AIAA AVIATION 2023 Forum*. 2023.

Challenge: Aeroheating on Complex Geometry

Unstructured CFD

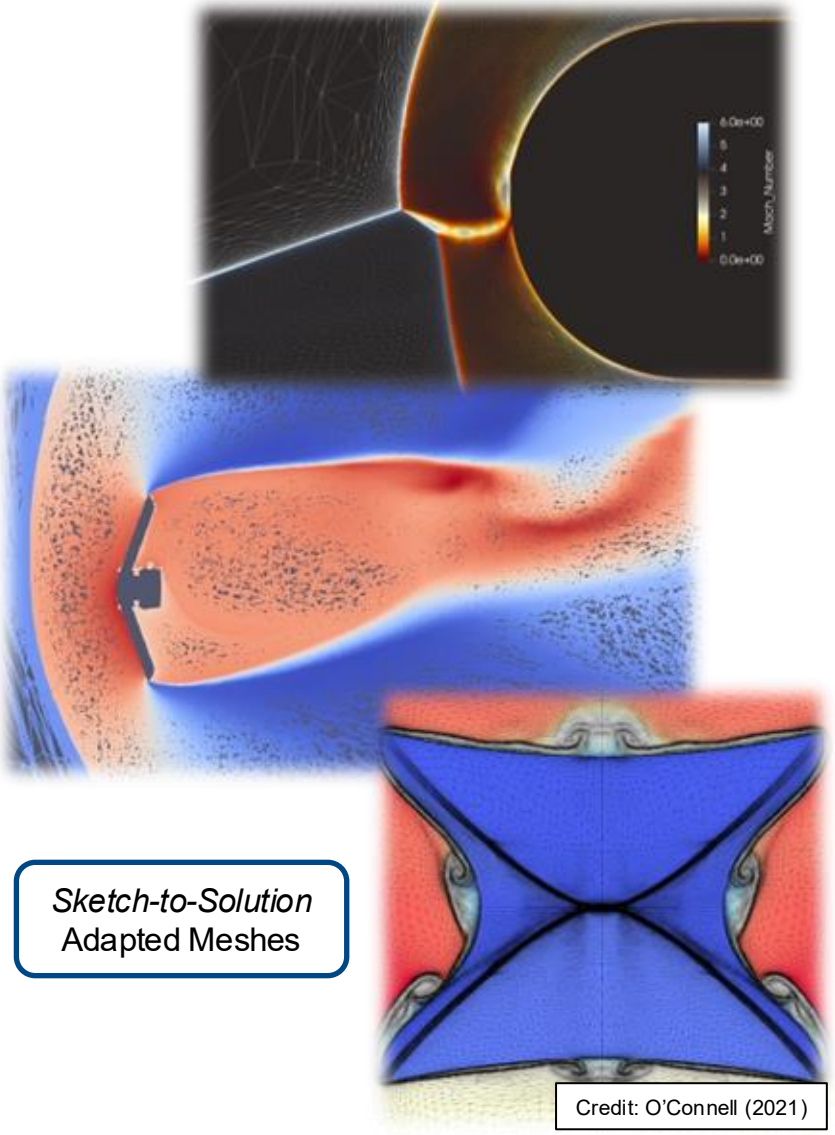
- Simplifies mesh generation
- Sensitivity to the mesh topology

Spherical heat shield surface heat flux contours on mixed-element meshes



Unstructured Mesh Adaptation

- Simplex meshes are meshes with triangles in 2D or tetrahedra in 3D
- They are “tetrahedral all the way to the wall.” There is no structured boundary-layer mesh with prism elements.
- These meshes allow for simple and robust mesh generation and adaptation.



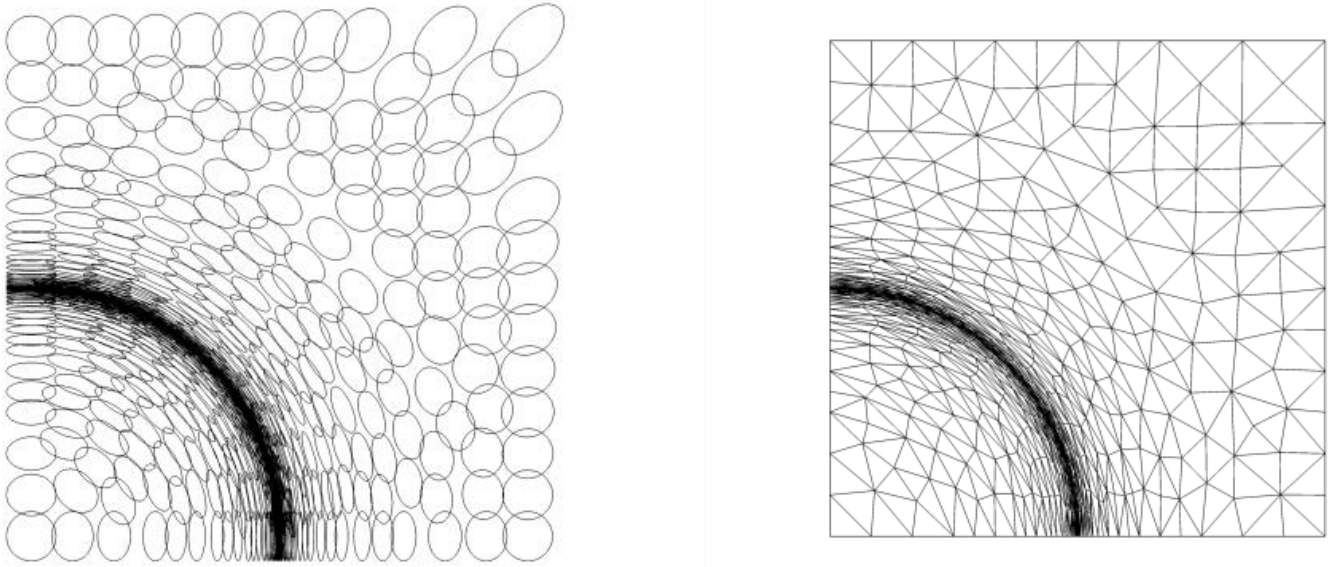
Sketch-to-Solution
Adapted Meshes

Credit: O’Connell (2021)

[1] Kleb, W., Park, M., Wood, W., Bibb, K., Thompson, K., and Gomez, R., “Sketch-to-Solution: An Exploration of Viscous CFD with Automatic Grids,” AIAA Paper 2019–2948, 2019.

Metric Based Unstructured Mesh Adaptation

- Metric field
  - Describes a request of grid density, stretching, and orientation
  - Multiscale metric[1] constructed to control solution interpolation error
    - Requires Hessian of a scalar field

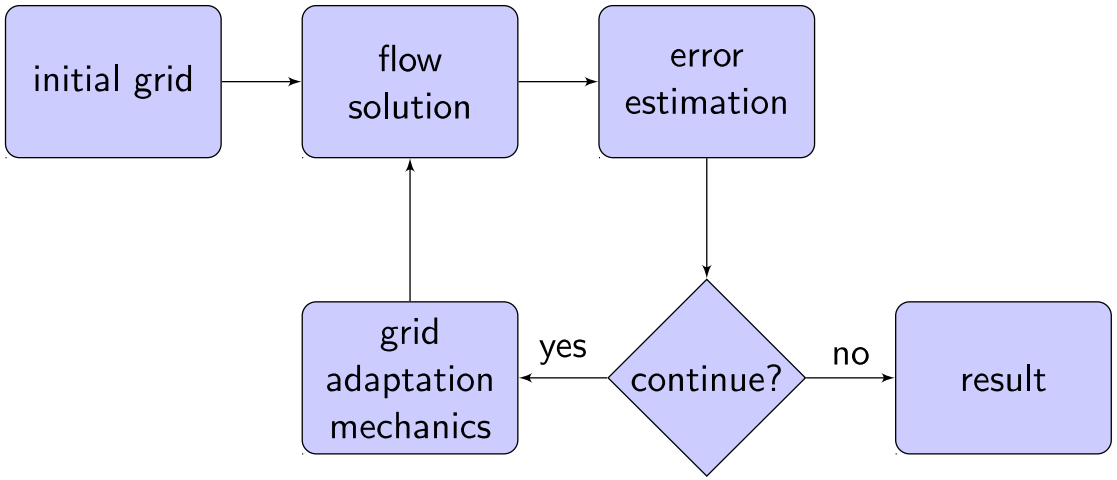


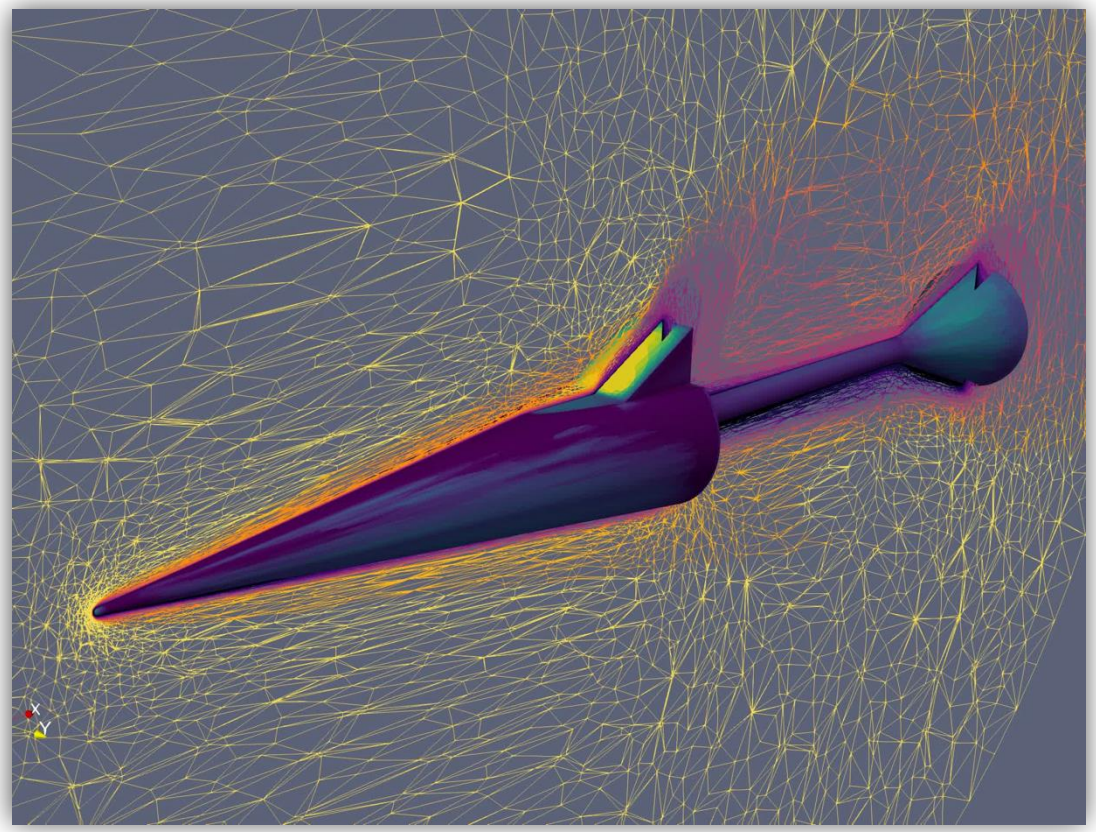
2D Metric Field Rendered as Ellipses and Unit Grid

[1]Alauzet and Losielle, JCP 229(3), 2010.

Sketch-to-Solution Process

Methodology: Solution-based mesh adaptation process



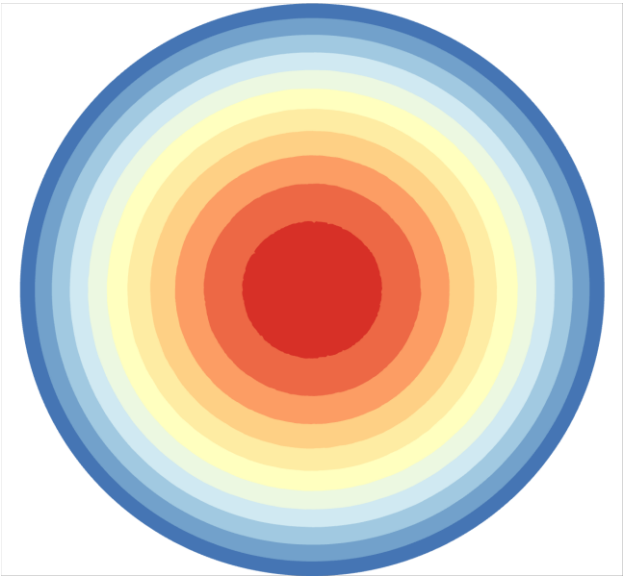


Numerical Challenges with Simplex Meshes

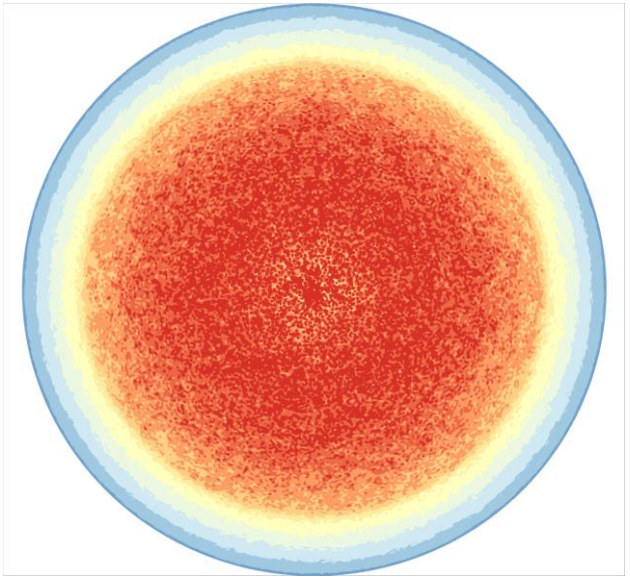
Traditional numerical schemes have mixed results on simplex meshes:

- Pressure can be predicted accurately
- Gradient quantities (such as skin friction or heating) have larger errors

Numerical methods that work well on regular cartesian grids suffer a loss of accuracy on unstructured meshes.



Pressure



Heat Flux

Can we improve our numerics to achieve higher accuracy heating on simplex meshes?

Part 1 Summary

- **Challenge: surface heat flux strongly dependent on mesh topology**
  - Most CFD for aerothermal analysis at NASA still relies on expert grid generation practitioners

- Automated mesh adaptation is a promising approach:
  - Mesh adaptation used successfully for aerodynamic predictions
  - Development has lagged for aerothermodynamic predictions
  - Sketch-to-Solution workflow can automatically adapt meshes via a scalar field provided by the CFD solver

QUESTIONS?

PART 2: HYPERSOLVE NUMERICAL METHODS

Economical Unstructured CFD Methods

Goal: Improve surface heat flux accuracy on simplex meshes without greatly increasing the degrees of freedom compared to traditional 2nd-order finite-volume schemes

Diffusion Problems as Hyperbolic PDEs

Diffusion Equation	Hyperbolic Diffusion Equation
$u_t = \nu u_{xx}$	$u_t = \nu\, p_x$ $p_t = (u_x - p)/T_r$

Recast a diffusion equation as a first-order system

- Gradient variables solved in additional equation
- New system of equations can be discretized using any method (finite-volume, DG, etc.)

Benefits of this reformulation include:

- Uniform order-of-accuracy of gradients and primal solution variables
- Faster iterative convergence, due to the removal of second-derivatives

Nakashima, Y., Norihiko W., and Nishikawa, H.. "Hyperbolic Navier-Stokes solver for three-dimensional flows." AIAA Paper 2016-1101, 2011.

Reformulated Heat Flux System (RHFS)

Only split off the temperature gradient as a first-order system[1]

- Pro: Only three additional equations required – less memory required
- Con: No iterative convergence acceleration, due to velocity second-derivatives remaining in the system

Navier-Stokes

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0$$
$$\partial_t (\rho u) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} - \tau) + \nabla p = 0$$
$$\partial_t (\rho E) + \nabla \cdot (\rho H \mathbf{v} - \tau \mathbf{v} + \mathbf{q}) = 0$$

Reformulated Heat Flux System (RHFS)

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0$$
$$\partial_t (\rho u) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} - \tau) + \nabla p = 0$$
$$\partial_t (\rho E) + \nabla \cdot (\rho H \mathbf{v} - \tau \mathbf{v} + \mathbf{q}) = 0$$

3 additional heat flux equations →

$$\frac{T_h}{\mu_h} \partial_t (\mathbf{q}) - \nabla T = -\frac{\mathbf{q}}{\mu_h}$$

[1]Thompson, K., Nishikawa, H., and Padway, E., "Economical Third-Order Methods for Accurate Surface Heating Predictions on Simplex Element Meshes.", AIAA Paper 2023-2629, 2023.

Implicit Edge-Based Gradients (IEBG)

Alternatively, use the same methodology to upgrade the accuracy of the solution gradients for use in a conventional numerical scheme[1]

- Pro: Additional storage required independent of number of equations
- Con: Requires solving a linear system for the gradient of each solution variable (only the RHS changes per solution variable)

Steady Hyperbolic-form of Laplace Equation

$$\cancel{\partial_t u}^{\,0} = \nabla \cdot \mathbf{g}$$

$$\cancel{\partial_t \mathbf{g}}^{\,0} = \frac{1}{T_r}\left[(\nabla u)^t - \mathbf{g}\right]$$

Discretize and solve for gradient variable (**g**) with solution fixed

Implicit Edge-Based Gradients (Linear System of Equations)

$$\underbrace{\frac{1}{V_j}\sum_{k\in k_j}\frac{c\mathbf{g}_j + (2-c)(\mathbf{g}_k - \nabla\mathbf{g}_j^{LSQ}\cdot\Delta\mathbf{r}_{jk})}{2}V_{jk}}_{\text{LHS (Does not change)}} = \underbrace{\frac{1}{V_j}\sum_{k\in k_j}\frac{u_L+u_R}{2}}_{\text{RHS (Changes per variable)}}$$

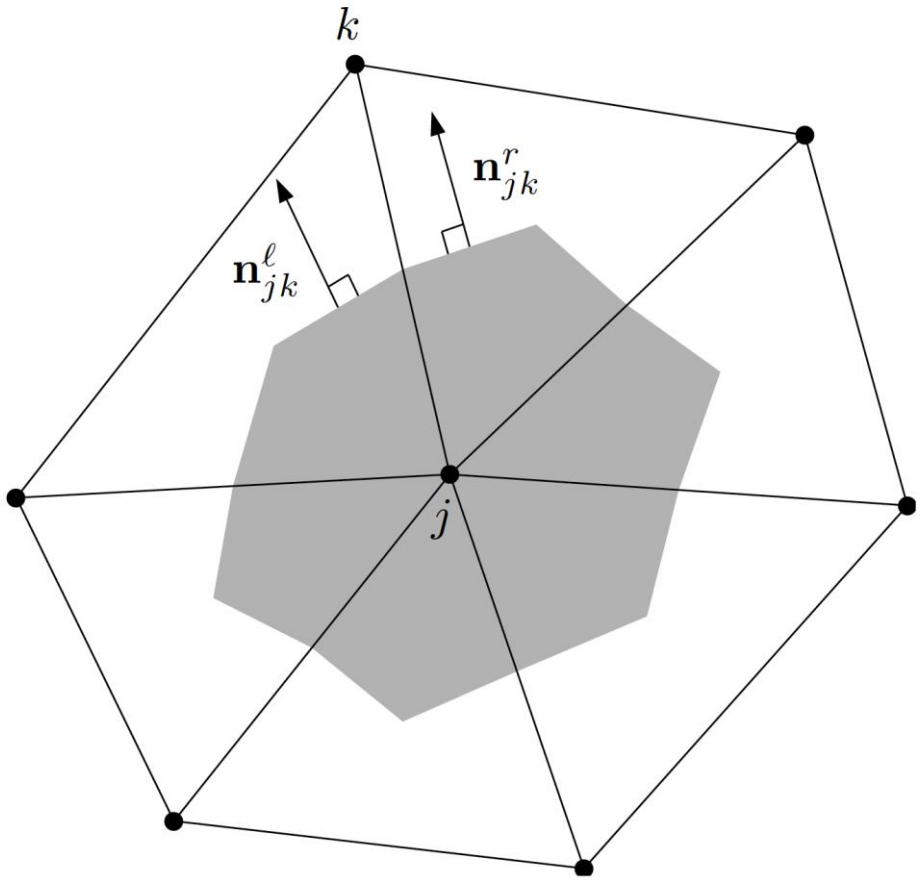
[1]Nishikawa, H. "Updates to Implicit Edge-Based Gradient Methods.", AIAA Paper 2024-1748, 2024.

Edge-Based Finite Volume Discretization

- Median Dual Control Volume
  - Nodal storage of solution
  - Fluxes evaluated at edge midpoints
  - Dual face directed area:

$$\mathbf{n}_{jk} = \mathbf{n}_{jk}^{l} + \mathbf{n}_{jk}^{r}$$

- Higher-order boundary closure via quadrature formulas[1]



[1]H. Nishikawa., "Beyond interface gradient: A general principle for constructing diffusion schemes.", AIAA Paper 2010-5093, 2010.

Economical 3rd-order Finite Volume Methods

3rd-order convective fluxes

Katz and Sankaran JCP 2011

$$\frac{1}{2}\left(\mathbf{F_L}+\mathbf{F_R}\right)-\frac{1}{2}\mathbf{D}$$

$$\mathbf{F_L}=\mathbf{F}_j+\frac{1}{2}\left(\frac{\partial\mathbf{F}}{\partial\mathbf{U}}\right)_j\boxed{\nabla\mathbf{U}_j}\cdot\mathbf{e}_{jk}$$

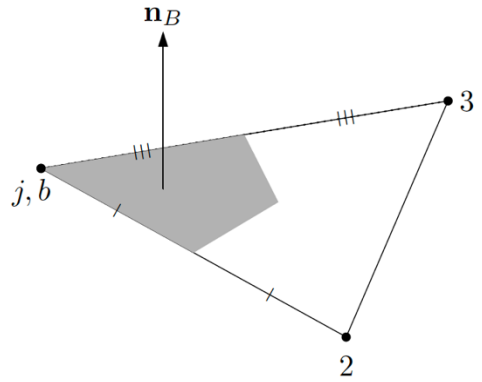
$$\mathbf{F_R}=\mathbf{F}_k-\frac{1}{2}\left(\frac{\partial\mathbf{F}}{\partial\mathbf{U}}\right)_k\boxed{\nabla\mathbf{U}_k}\cdot\mathbf{e}_{jk}$$

Evaluated using QLSQ

3rd-order-preserving boundary quadrature

Nishikawa, JCP 2014

$$\left(\frac{1}{2}\mathbf{F}_{jB}+\frac{1}{4}\mathbf{F}_{j2}+\frac{1}{4}\mathbf{F}_{j3}\right)\cdot\mathbf{n_B}$$



Boundary Quadrature at Node *j*

3rd-order source term quadrature

Nishikawa and Liu, JCP 2017

$$\int_{V_j}\mathbf{S}\,dV=\frac{1}{60}\sum\left(13\mathbf{S}_j+3\partial\mathbf{S}_{jk}-3\mathbf{S}_k\right)\left(\mathbf{e}_{jk}\cdot\mathbf{n}_{jk}\right)$$

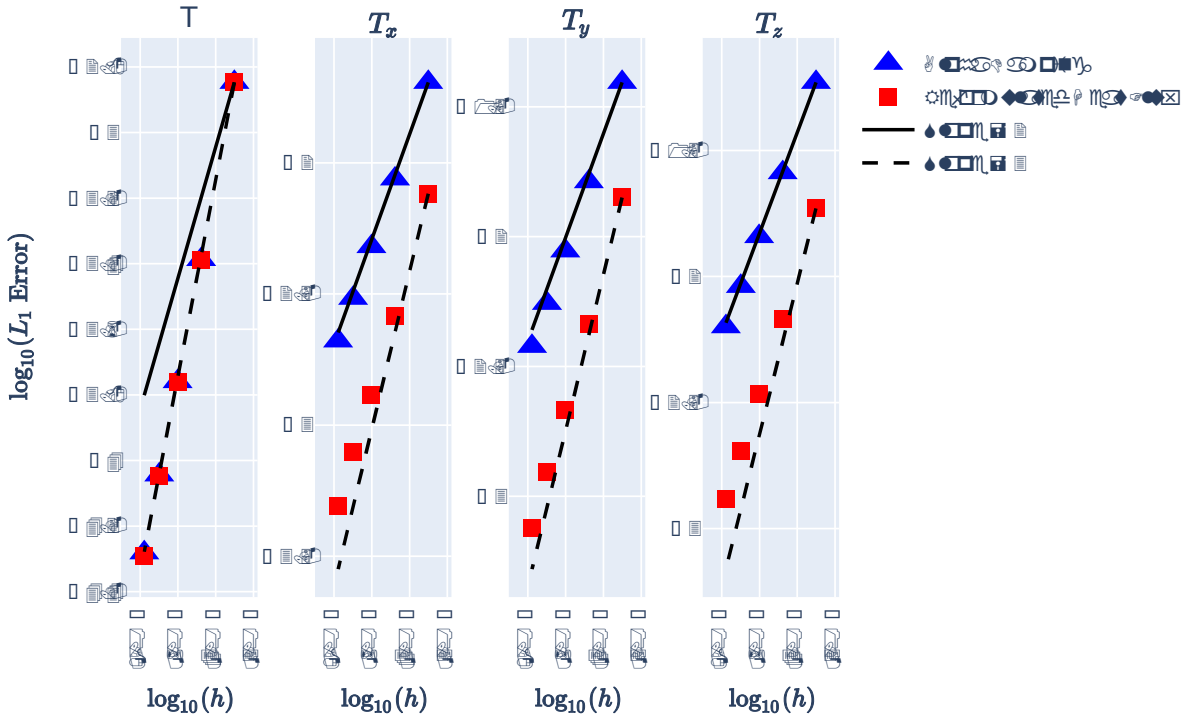
Same DoF as 2nd-order FV!

Order of Accuracy Verification

Order of accuracy verified using Method of Manufactured Solutions

Manufactured Solution

$$\rho = 0.11\ \mathrm{e}^{x} + 0.12\ \mathrm{e}^{y} + 0.13\ \mathrm{e}^{z} + 1.23$$
$$u = 10\ \mathrm{e}^{x} + 20\ \mathrm{e}^{y} + 30\ \mathrm{e}^{z} + 100$$
$$v = 10\ \mathrm{e}^{x} + 20\ \mathrm{e}^{y} + 30\ \mathrm{e}^{z} + 200$$
$$w = 10\ \mathrm{e}^{x} + 20\ \mathrm{e}^{y} + 30\ \mathrm{e}^{z} + 300$$
$$T = 10\ \mathrm{e}^{x} + 20\ \mathrm{e}^{y} + 30\ \mathrm{e}^{z} + 350$$



Using QLSQ, RHFS achieves >2nd-order temperature gradient accuracy

- Relatively low cost: three additional equations/variables per mesh vertex

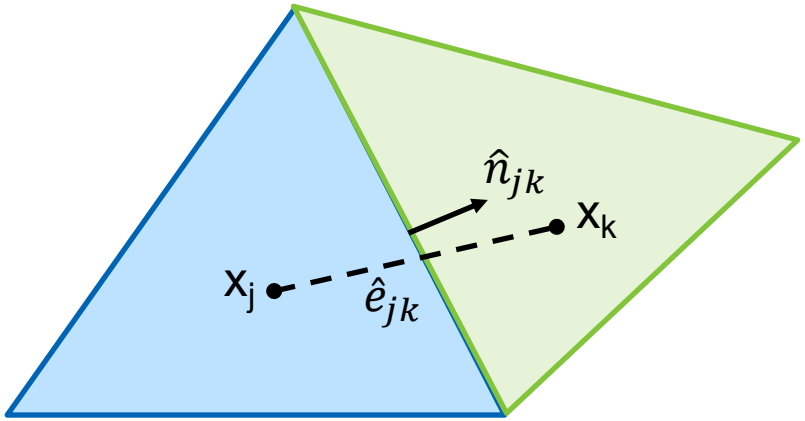
Viscous Scheme

Viscous fluxes can be expressed as a product of diffusivity (ν) and a gradient term:

$$\vec{F} = -\nu \nabla \mathrm{u}$$

This flux is then projected onto a face:

$$\phi_{jk} = -\nu \nabla \mathrm{u} \cdot \hat{n}_{jk}$$



A simple choice for a viscous scheme would be:

$$\phi_{jk} = -\nu \overline{\nabla u} \cdot \hat{n}_{jk}$$

where $\overline{\nabla u}$ is the average of the gradients at x_j and x_k.

This simple and consistent scheme is not stable for skewed meshes, where $\hat{n}_{jk}$ is not aligned with $\hat{e}_{jk}$.

Alpha-Damping Scheme

Nishikawa [1,2] derived a general formula for 2nd to 4th order accurate viscous schemes on a compact stencil:

$$\phi = -\nu\overline{\nabla u}\cdot\hat{n}_{jk} - \frac{\nu\alpha}{\left|\hat{e}_{jk}\cdot\hat{n}_{jk}\right|}\left[\frac{u_k - u_j}{\ell_{jk}} - \overline{\nabla u}\cdot\hat{e}_{jk}\right]$$

Average gradient, projected onto face

Gradient along edge

Average gradient, projected onto edge

where $\ell_{jk} \equiv \left|x_k - x_j\right|$ and α is a free parameter.

Setting $\alpha = 4/3$ can yield 4th order accuracy on specific types of meshes at the expense of monotonicity preservation. Other common choices:

- $\alpha = 1$
- $\alpha = \hat{e}_{jk}\cdot\hat{n}_{jk}/\left|\hat{e}_{jk}\cdot\hat{n}_{jk}\right|$ (see [3])

1. [Nishikawa, 2011] *"Robust and Accurate Viscous Discretization via Upwind Scheme - I: Basic Principle"*
2. [Nishikawa, 2010] *"Beyond Interface Gradient: A General Principle for Constructing Diffusion Schemes"*
3. [Haselbacher and Blazek, 2000] *"Accurate and Efficient Discretization of Navier-Stokes Equations on Mixed Grids"*

A Review of Roe Fluxes

The typical Roe flux can be expressed as:

$$\phi_{jk} = \frac{1}{2}\left(F(w_L) + F(w_R)\right) - \frac{1}{2}|A(w_L, w_R)|(u_R - u_L)$$

where F is the inviscid flux function, $|A|$ is a Jacobian matrix, w_L and w_R are the primitive variables reconstructed at the faces:

$$w_L = w_j + \nabla w_j \cdot (x_c - x_j), \quad w_R = w_k + \nabla w_k \cdot (x_c - x_k),$$

u_j and u_k are the solution values at the nodes, and u_L and u_R are the conservative variables computed from w_L and w_R.

The reconstruction can cause problems for A if the linear reconstruction results in unrealizable states, such as negative density. Falling back on first order accuracy ($w_L = w_j$) is common but not necessary.

Robust Roe Flux

The typical Roe flux can be expressed as:

$$\phi_{jk} = \frac{1}{2}\left(F(w_L) + F(w_R)\right) - \frac{1}{2}|A(w_L, w_R)|(u_R - u_L)$$

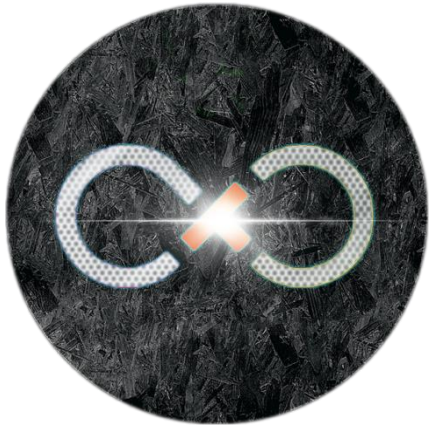
Nishikawa [1] proposed calculating the Jacobian matrix using the nodal values, instead of reconstructed values:

$$\phi_{jk} = \frac{1}{2}\left(F(w_L) + F(w_R)\right) - \frac{1}{2}\left|A\left(u_j, u_k\right)\right|(u_R - u_L)$$

This allows for unrealizable reconstructed states (such as w_R and w_L), while maintaining 2nd order accuracy.

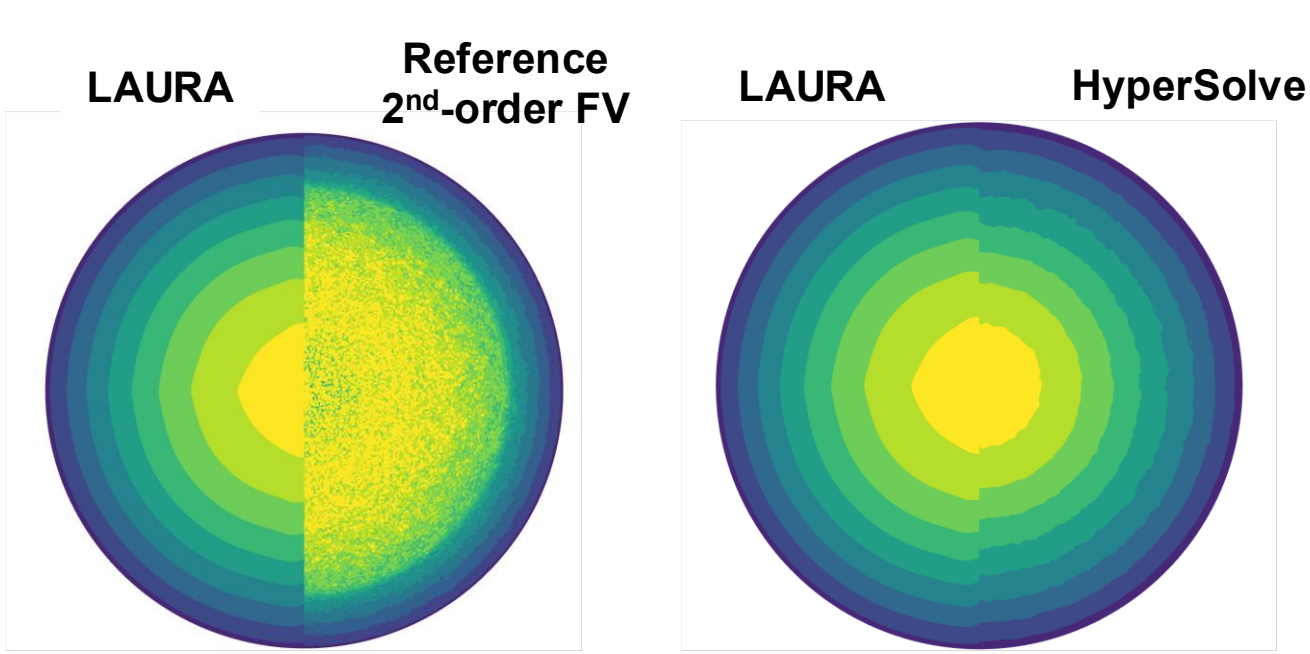
HyperSolve CFD Solver

- HyperSolve is a three-dimensional finite-volume solver being used to investigate the efficacy of advanced numerical methods on unstructured meshes
- HyperSolve CFD Solver Capabilities
  - 2nd/3rd-order edge-based, finite-volume discretizations
  - Implicit edge-based gradients
  - In-core mesh adaptation for multiblock structured grids
  - Thermally perfect gas and reacting flow in thermal nonequilibrium
  - 1-eqn and 2-eqn RANS turbulence model variants
  - Jacobian-Free Newton-Krylov nonlinear solver with nonlinear preconditioning
  - Performance portability to GPU hardware via Kokkos library[1]
  - Ray tracing and radiative heat flux
- Provided through T-infinity plugins
  - Unstructured mesh adaptation
- Core mission of HyperSolve to is to provide accurate surface heat transfer predictions for NASA Entry vehicles, with a focus on unstructured mesh adaptation

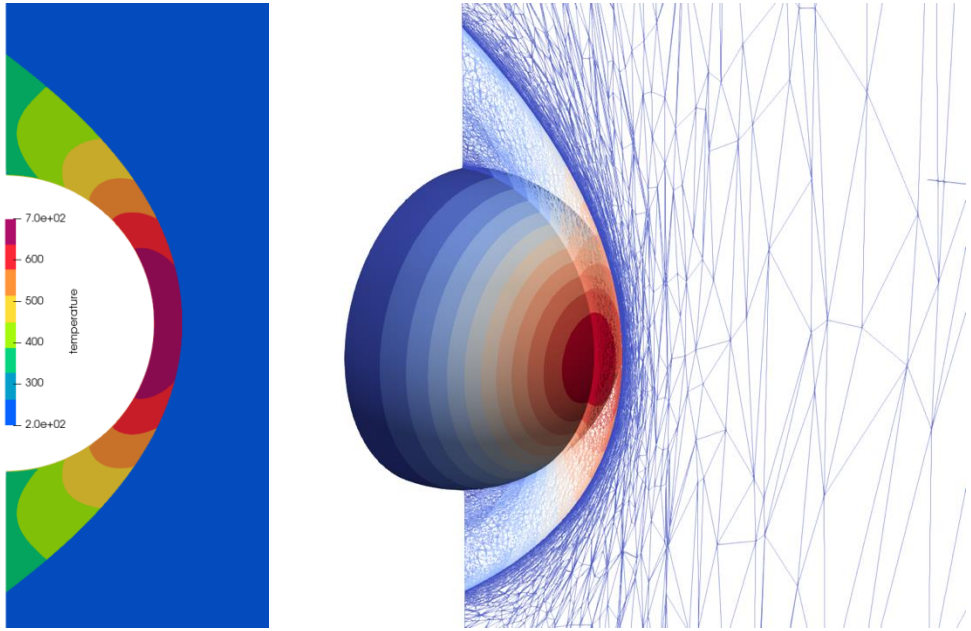


1. Trott et al. "Kokkos 3: Programming Model Extensions for the Exascale Era". *IEEE Transactions on Parallel and Distributed Systems, 2022*

1 km/s 3D Laminar Flow over a Hemisphere



Surface Heat Flux



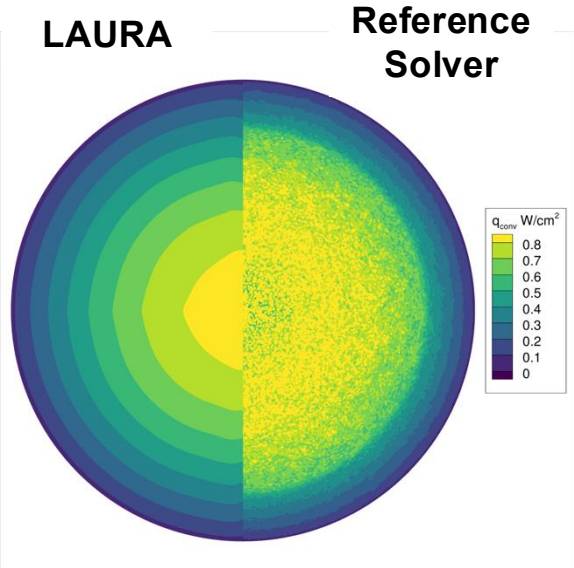
Fully-Tetrahedral Mesh
(Adapted via Sketch-to-Solution)

LAURA structured solver used as benchmark for assessing accuracy
HyperSolve RHFS formulation provides more accurate heat transfer predictions[1]

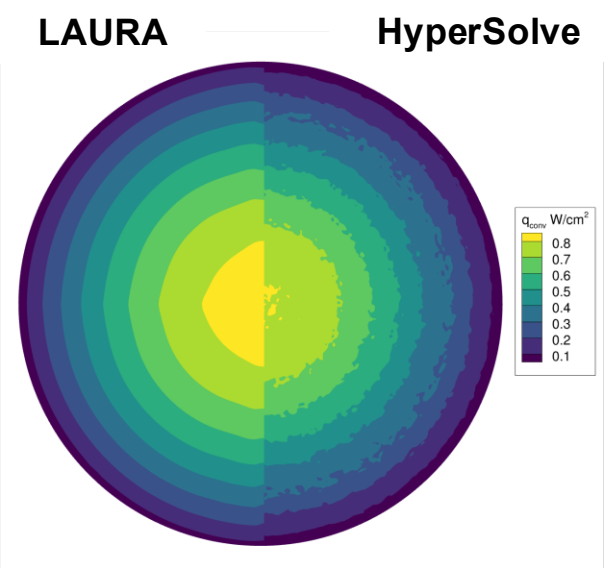
- HyperSolve uses the *refine* library for mesh adaptation, with temperature-Hessian metric
- HyperSolve RHFS scheme enables using the Sketch-to-Solution workflow for aerothermodynamic analysis

[1]Thompson, K., Nishikawa, H., and Padway, E., "Economical Third-Order Methods for Accurate Surface Heating Predictions on Simplex Element Meshes.", AIAA Paper 2023-2629, 2023.

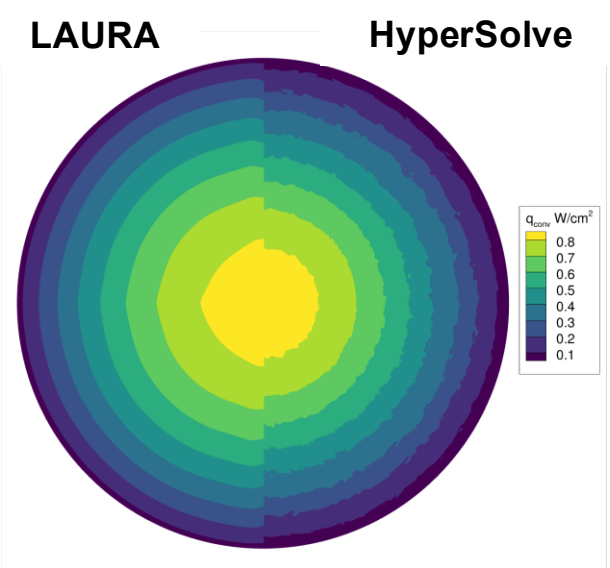
1 km/s 3D Laminar Flow over a Hemisphere



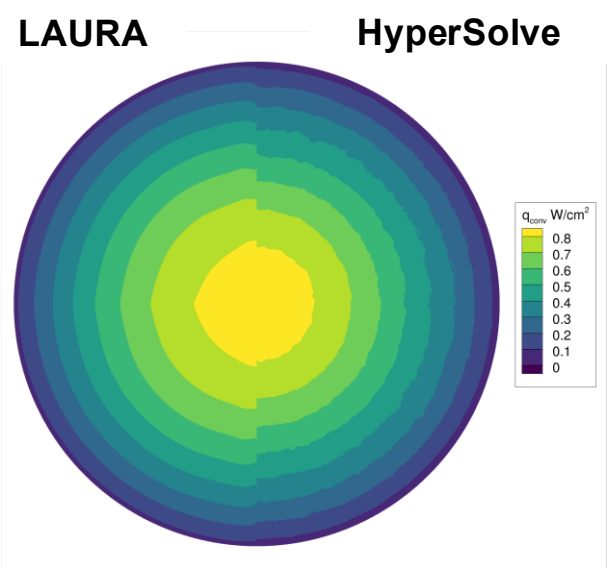
P1 Continuous Galerkin



Linear LSQ Gradients + Alpha-Damping



Quadratic LSQ Gradients + Alpha-Damping



Quadratic LSQ Gradients + RHFS

Least Expensive
Least Accurate

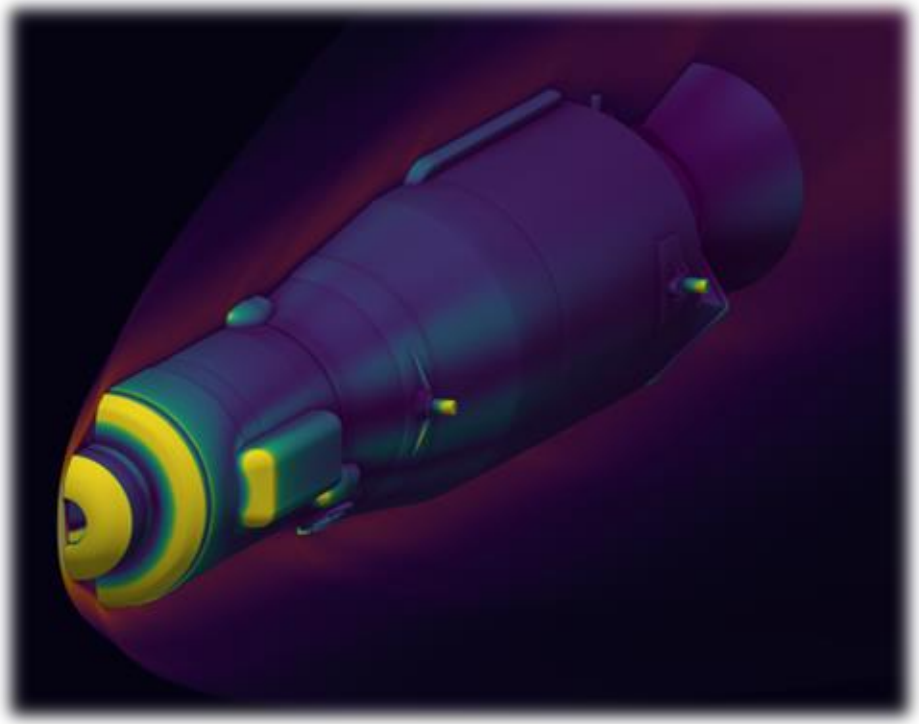
Most Expensive
Most Accurate

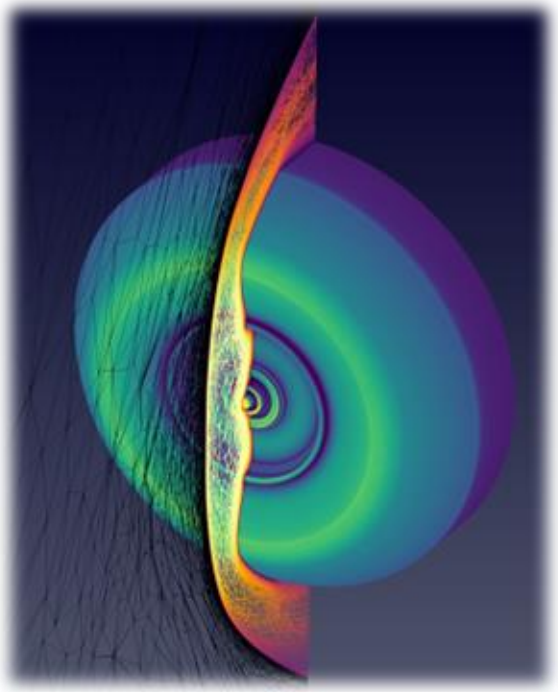
Important to note:

- Alpha-Damping consistently outperforms P1 Continuous Galerkin
- Quadratic LSQ Alpha-Damping and RHFS results are similar
- Similar degrees of freedom solved by LAURA and HyperSolve

Mars Ascent Vehicle

- HyperSolve has been enabling for aerothermal analysis on higher fidelity vehicle geometry
  - Grid generation cost reduced to CAD requirements (i.e., watertight

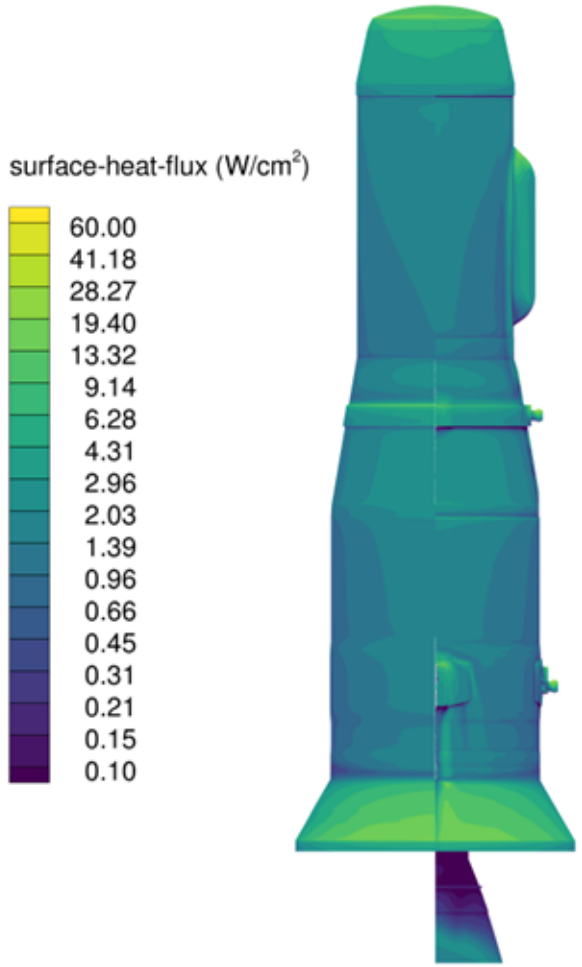


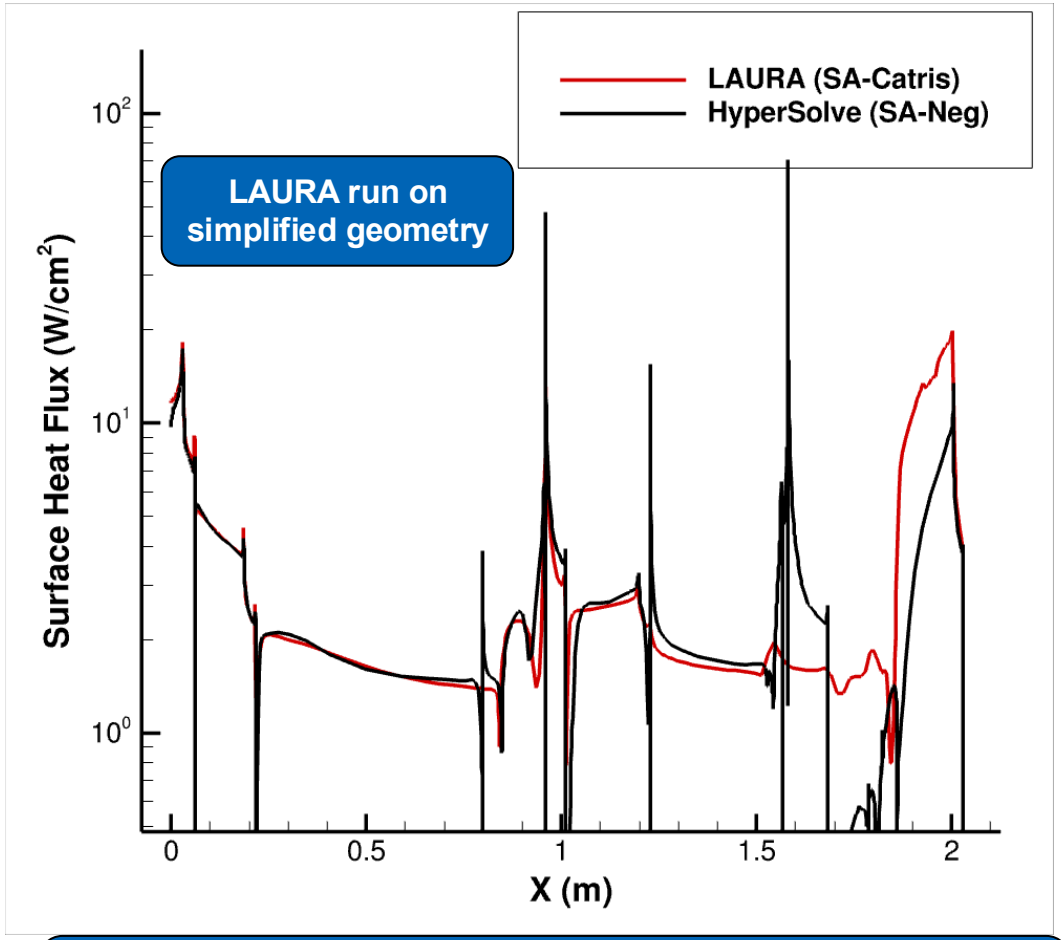


Solver	Nominal Time to Solution (Including Gridding)
LAURA	2 Weeks
HyperSolve	1-2 Days

Mars Ascent Vehicle results previously presented:
Thompson, K., Hinkle, D., and O'Connel, Matthew. "HyperSolve: An Automated Mesh Adaptation Workflow for Aerothermal Analysis" 20th International Planetary Probe Workshop, June 14, 2024, Williamsburg, VA

Mars Ascent Vehicle Surface Heating





Differences with increasing X are due to turbulence model and geometry differences

Mars Ascent Vehicle results previously presented:
Thompson, K., Hinkle, D., and O'Connel, Matthew. "HyperSolve: An Automated Mesh Adaptation Workflow for Aerothermal Analysis" 20th International Planetary Probe Workshop, June 14, 2024, Williamsburg, VA

QUESTIONS?

PART 3: NONLINEAR SOLVER

Newton Methods

For a steady-state problem, we are trying to solve a PDE that can be expressed as:

$$R(Q) = 0$$

where R is the nonlinear residual of the discretized PDEs and Q is the state vector.

Using Newton's method, we can solve this as:

$$\frac{\partial R}{\partial Q}\Delta Q = -R(Q^n)$$

where $\Delta Q = Q^{n+1} - Q^n$ and n is the current iteration.

Jacobian-Free Newton-Krylov (JFNK)

For a preconditioned Krylov method such as GMRES, we do not need the Jacobian for the linear solver. Instead we just need the matrix-vector product.

In JFNK, we approximate the matrix-vector product as:

$$\frac{\partial R}{\partial Q}\Delta Q \approx \frac{R(Q+\varepsilon\Delta Q)-R(Q)}{\varepsilon}$$

This is a powerful nonlinear solver method that allows deep convergence. Advantages include:

- JFNK can be faster than traditional linear solvers if repeated residual evaluation is cheaper than exact Jacobian construction.
- You can avoid common approximations in Jacobians that can stall convergence.

Pseudo-Transient Continuation

Newton or quasi-newton methods are powerful, but other techniques can be more robust or efficient for initial transients. The solution: globalization techniques.

For example, one can perform time-marching using a Backward Euler method:

$$\left(\frac{V}{\Delta t}+\frac{\partial R}{\partial Q}\right)\Delta Q=-R(Q^n)$$

where V is the volume of the computational cells and Δt is the timestep.

- Small timesteps can help early in convergence
- With large timesteps, this scheme recovers a Newton method

Other techniques

Line search

- Iterative nonlinear solvers give you a “direction” and a “step size.”
- What if taking the full step size results in:
  - an unrealizable state or
  - a high residual?
- You can backtrack along the line and take a smaller “step size” that is realizable or has a lower residual.

Other techniques

Local CFL and adaptive CFL

- You want to take large timesteps where possible, but smaller timesteps where there are undershoots or equation stiffness
- Adaptively ramp the timestep based on linear/nonlinear solver success
- You can also estimate a timestep limit based on the largest timestep that will keep the solution physical in a forward Euler technique:

$$\Delta t < r\rho^n / R^n$$

Where *r* is a relaxation parameter, ρ^n is the density, and R^n is the density residual.

- We have used a timestep limit based on temperature.
- More complex timestep limits are also possible (see discussion in [1])

1. [Luke2003] Luke, Edward A., et al. "A chemically reacting flow solver for generalized grids." *AIAA Journal* (2003).

Part 3 Summary

- There is no one-size-fits-all nonlinear solver
- Newton-methods are powerful and allow deep convergence when accurate Jacobians are used
- Time-marching with implicit methods is more robust and economical under certain conditions
- Pseudo-transient continuation bridges implicit time-marching techniques and Newton method

QUESTIONS?

PART 4: HYPERSOLVE SOFTWARE DESIGN AND PRINCIPLES

CFD 2030 Vision Report

"NASA should develop and maintain an integrated simulation and software development infrastructure to enable rapid CFD technology maturation."

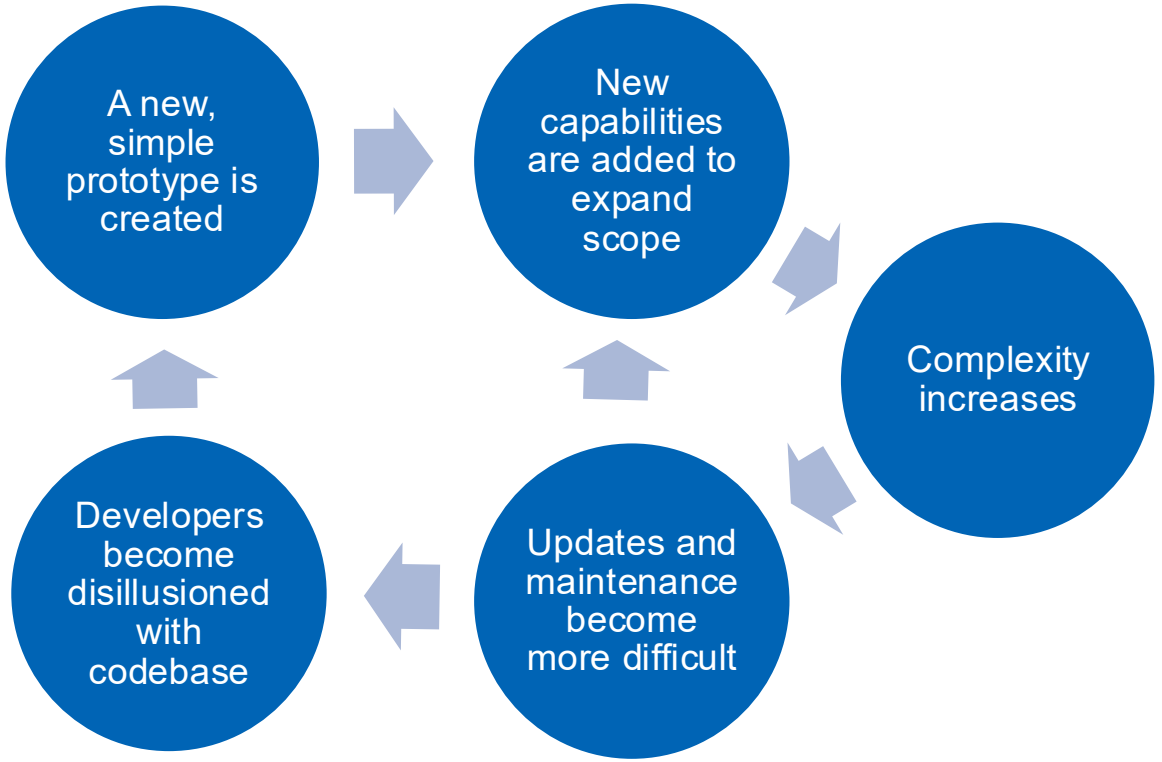
- CFD 2030 Vision Report[1] Programmatic Recommendation #2

HyperSolve developers are attempting to guide new software development efforts *and* add benefits to existing software tools keeping this recommendation in mind.

[1]Slotnick, Jeffrey P., et al., "CFD vision 2030 study: a path to revolutionary computational aerosciences." No. NF1676L-18332. 2014.

Reality of Long-term Software Development

- Many research scientists are not software engineers by training; they are either unaware of or do not prioritize long-term software craftsmanship.
- Software always grows in scope to meet new demands.
- Work must be done to fight the disorder and complexity of the system:
  - Define interfaces
  - Write unit tests
  - Refactor
  - Delete unused code



Why write abstractions and interfaces?

- Software should scale!
  - Scale with computer resources. (Parallel Scalability)
  - Scale with human resources. (Human Scalability)

- Challenges:
  - Problems are so complex, no one can be an expert in every aspect.
    - Grid generation, thermochemistry, radiation, turbulence, and nonlinear solvers are all very deep fields.
  - Experts solving problems from a different domain of expertise is suboptimal.

We should agree up-front on simple conceptual models that allow experts from different fields to collaborate.

SOLID Software Design Principles

SOLID Design Principles

- **S**ingle Responsibility Principle
  - Software entities should have only one reason to change.
- **O**pen Closed Principle
  - Software entities should be open for extension but closed for modification.
- **L**iskov Substitution Principle
  - Subtypes must be substitutable for their base types.
- **I**nterface Segregation Principle
  - Clients should not be forced to depend on methods that they do not use.
- **D**ependency Inversion Principle
  - High-level modules should not depend on low-level modules. Both should depend on abstractions.
  - Abstractions should not depend on details. Details should depend on abstractions.

SOLID Software Design Principles

SOLID Design Principles

- **S**ingle Responsibility Principle
  - Software entities should have only one reason to change.
- **O**pen Closed Principle
  - Software entities should be open for extension but closed for modification
- **L**iskov Substitution Principle
  - Subtypes must be substitutable for their base types.
- **I**nterface Segregation Principle
  - Clients should not be forced to depend on methods that they do not use.

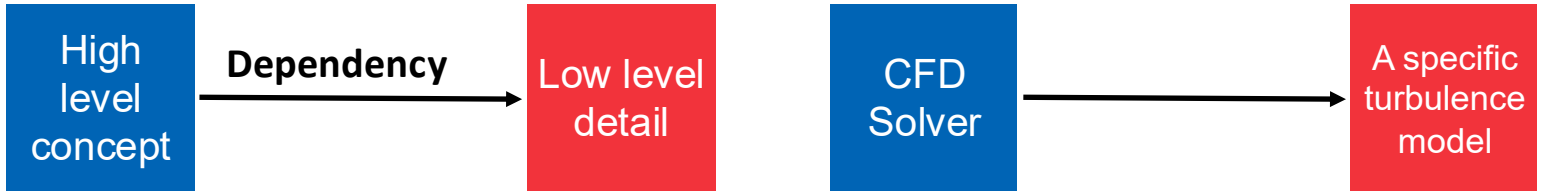
- **D**ependency Inversion Principle
  - High-level modules should not depend on low-level modules. Both should depend on abstractions.
  - Abstractions should not depend on details. Details should depend on abstractions.

Abstractions as Interfaces

- Interfaces are defined in C++ abstract base classes.
- They represent high level concepts using the minimal, yet sufficient set of information.

```
class MeshAdaptationInterface {
public:
   virtual std::shared_ptr<MeshInterface> adapt(
      std::shared_ptr<MeshInterface> mesh,
      MPI_Comm comm,
      std::shared_ptr<FieldInterface> metric,
      Json options) const = 0;
};
```

Direct Coupling



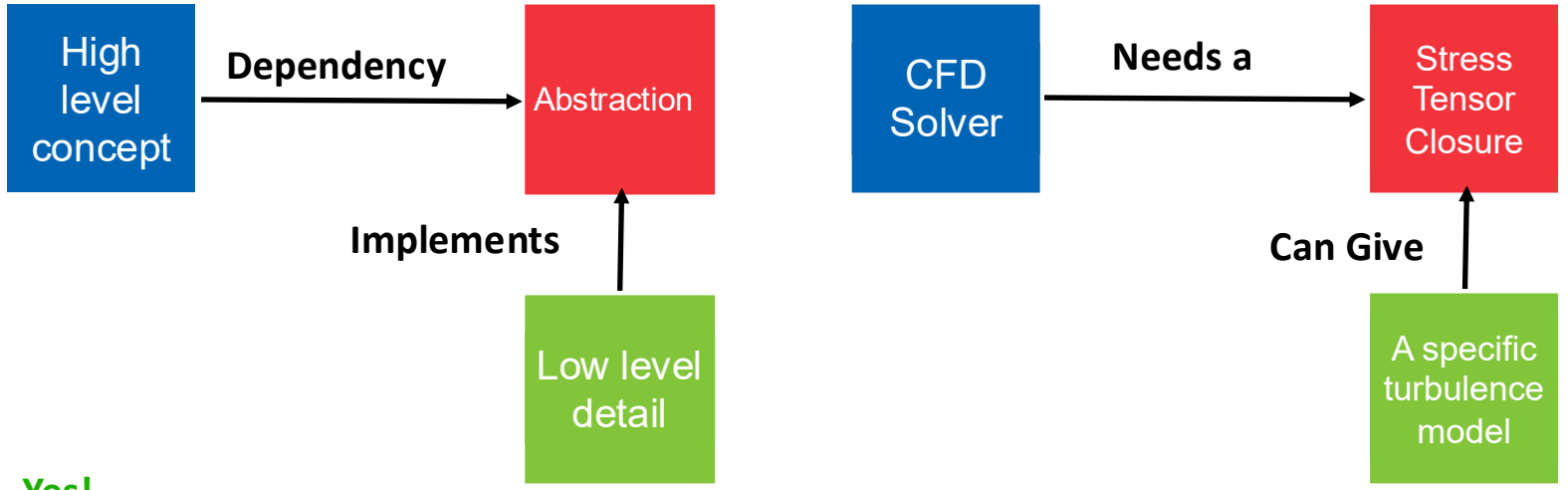
No!

Causes tight coupling,
hard to maintain,
high cognitive load.

”Did I find all the if-then branches for SA?
Or did I miss one somewhere?”

```
if (turbulence_model == "SA") {
  turb_avg = sum(...);
  ft1 = ...;
  ...
  tstress = ...;
} else if (turbulence_model == "SST") {
  ...
} else if (turbulence_model == "BSL") {
  ...
} else ... {
  ...
}
```

Dependency Inversion



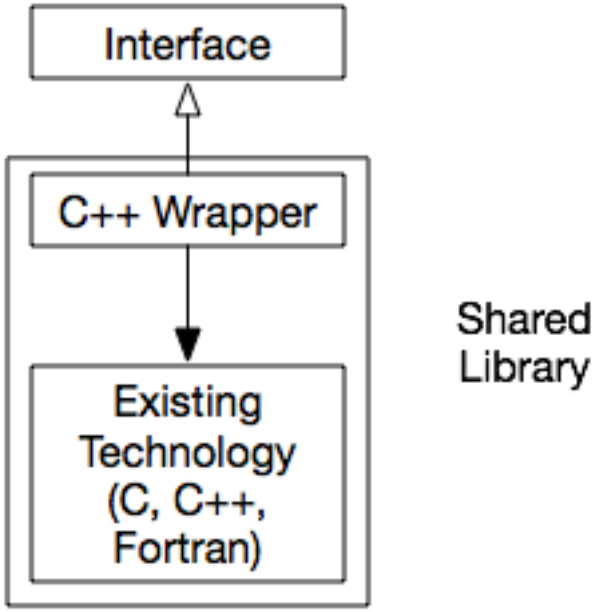
Yes!

Forces modularity,
easy to extend,
lower cognitive load

```
turbulent_stress = turb_model->computeTurbulentStress(
                                                density,
                                                u,
                                                v,
                                                w,
                                                &turb_model_vars);
```

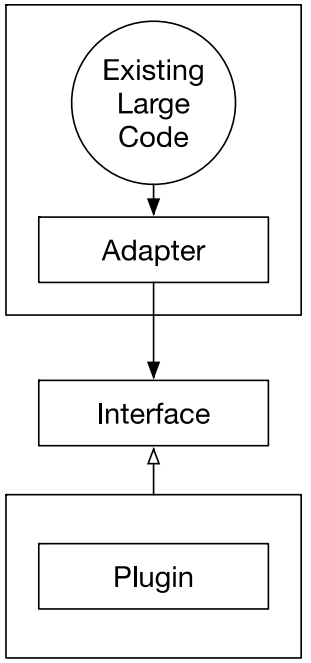
Plugins

- Plugins implement the interface.
- Plugins are C++ wrapper classes.
- Underlying implementation can be in C, C++, Fortran.
- Plugins are compiled as a shared library.



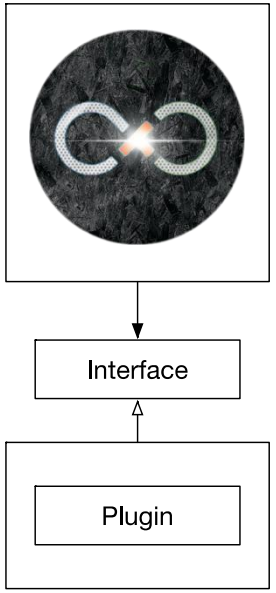
Using Plugins Directly

- Existing codes can incrementally add features over time adopting only some interfaces as needed.
- Need mesh adaptation?
  - Add new features by calling code to use the mesh adaptation interface.
  - Any compliant mesh adaptation plugin will work with that interface.



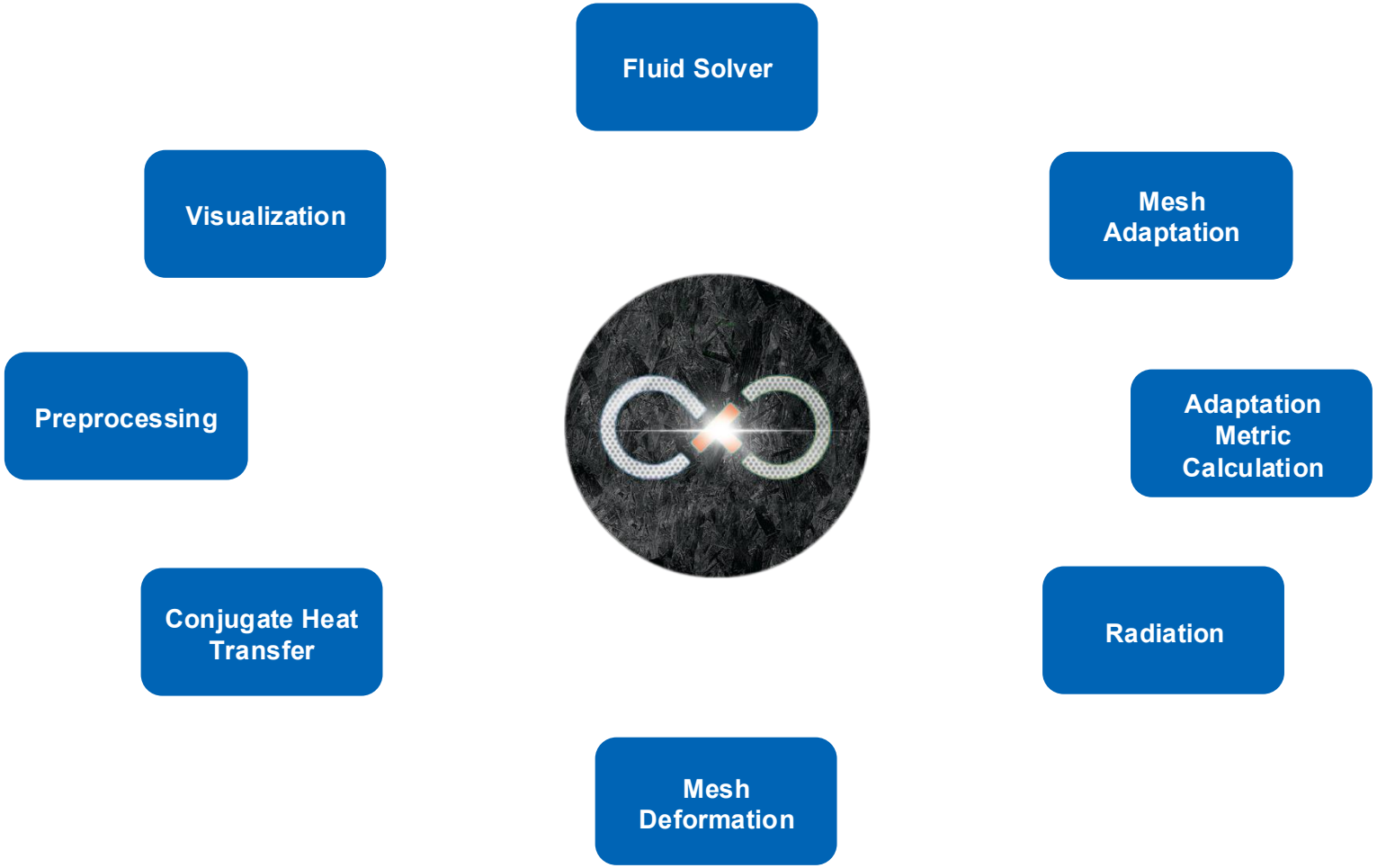
Using T-infinity Runtime

- HyperSolve uses the T-infinity runtime[1] to load plugins flexibly
  - C++ / Python bindings available.
  - Write driver scripts to set up complex cases. Typically using less than 100 lines of code.
  - Plugins are automatically wrapped by the T-infinity runtime.
  - Wrap technology once in C++, use it in C++ or Python automatically.

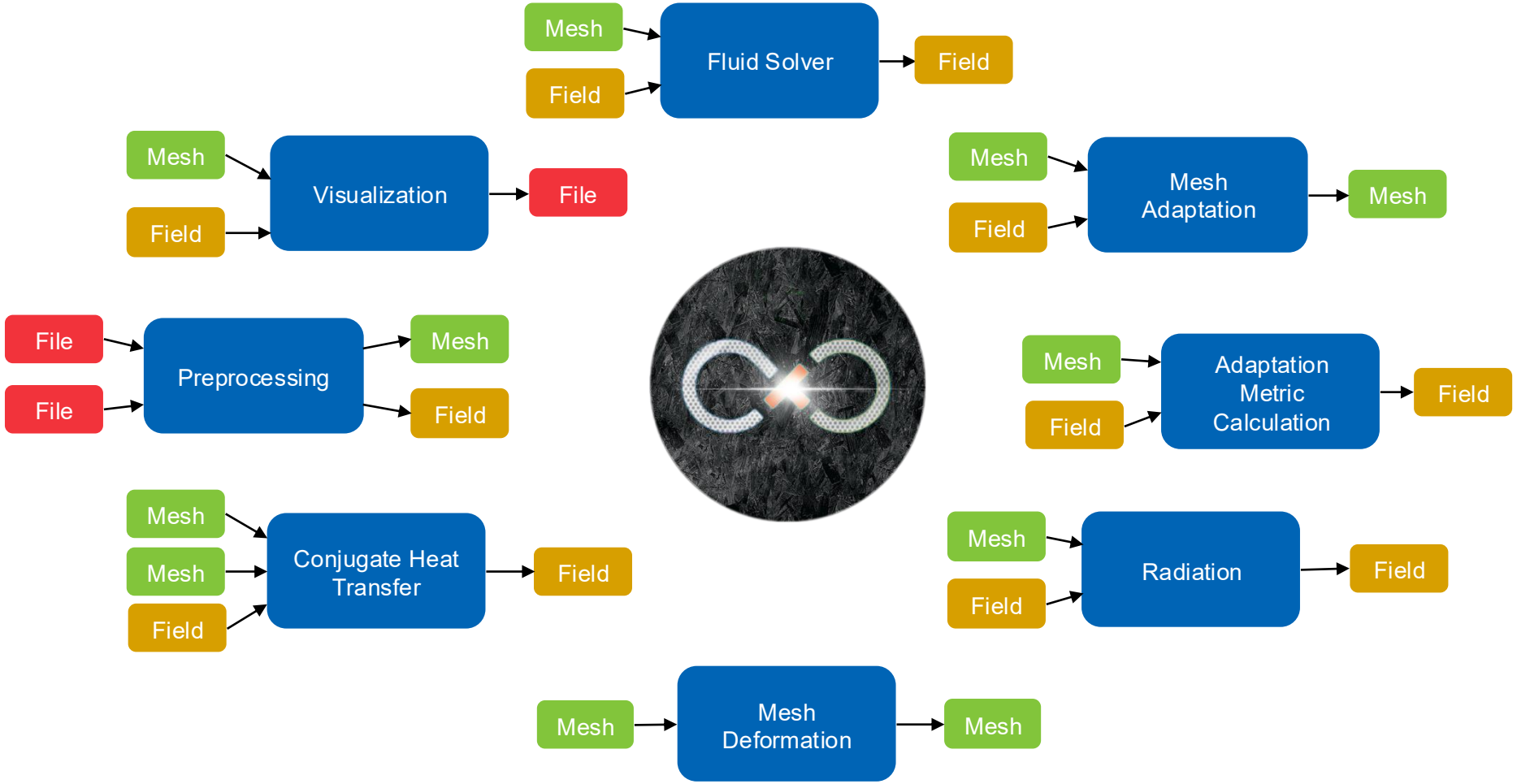


[1]O'Connell, Matthew, et al. "Application of the dependency inversion principle to multidisciplinary software development." AIAA Paper 2018-3856, 2018.

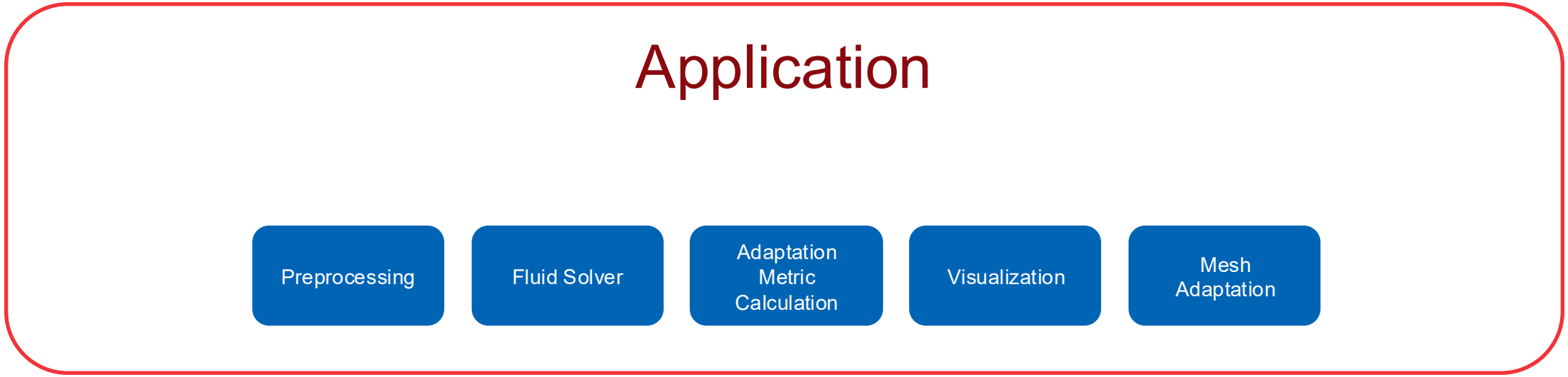
Building a CFD Solver Through Plugins



Unifying inputs and outputs gives flexibility

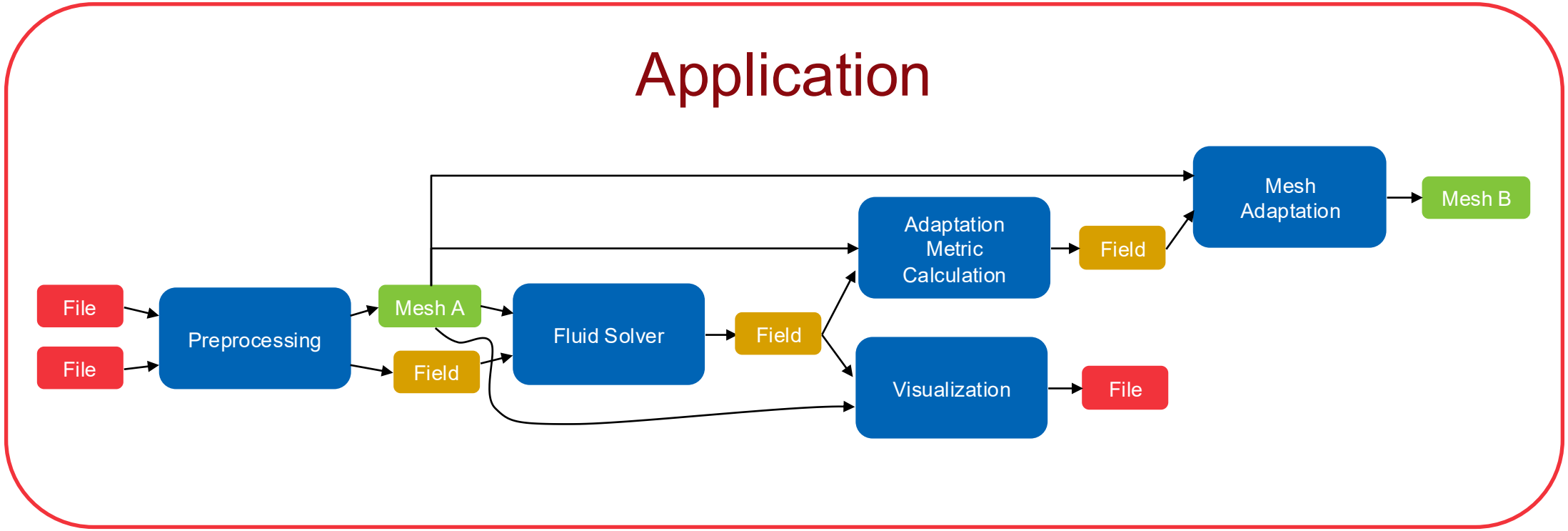


Flexible Uses: Cohesive Orchestrating Application

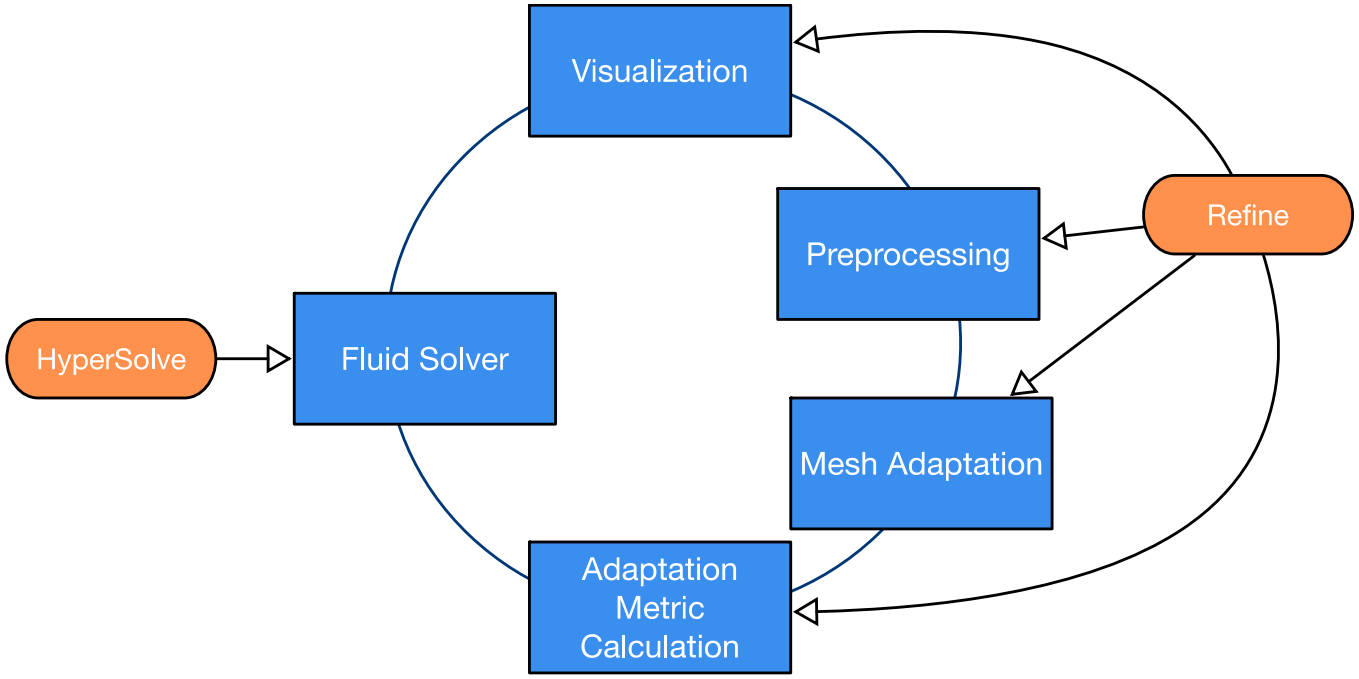


- The interfaces are designed to be independent, yet collaborative.
- The output of one interface is the input of another.
- Orchestrating applications can be written without adapting data.

Unifying inputs and outputs gives flexibility



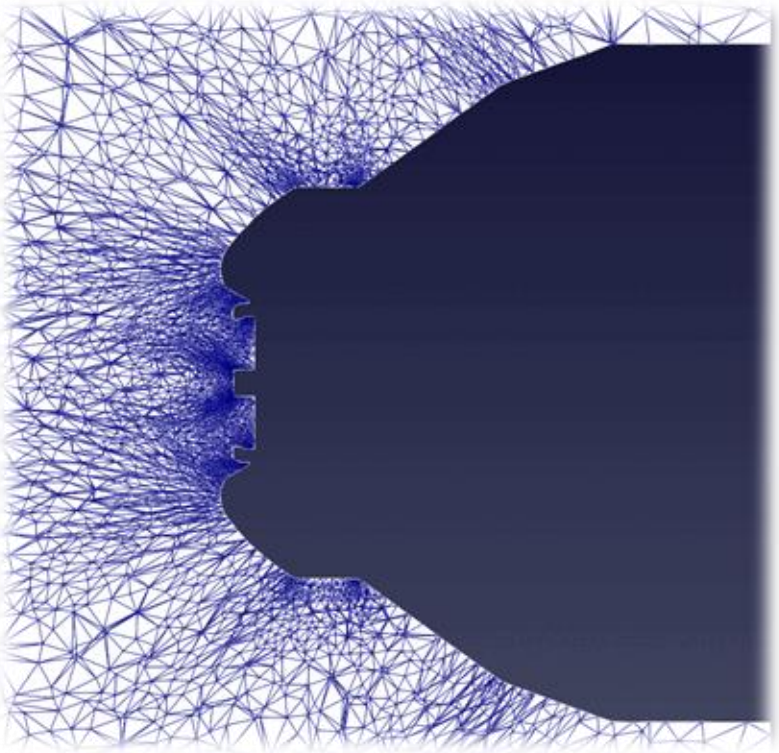
Pick and Choose: Mesh Adaptation



Hessian to metric-based adaptation.

Solver: HyperSolve

Adaptation, Visualization, Preprocessing: Refine



Example of mesh adaptation on the nose of the Mars Ascent Vehicle

Plugin Architecture: Pros and Cons

Pros

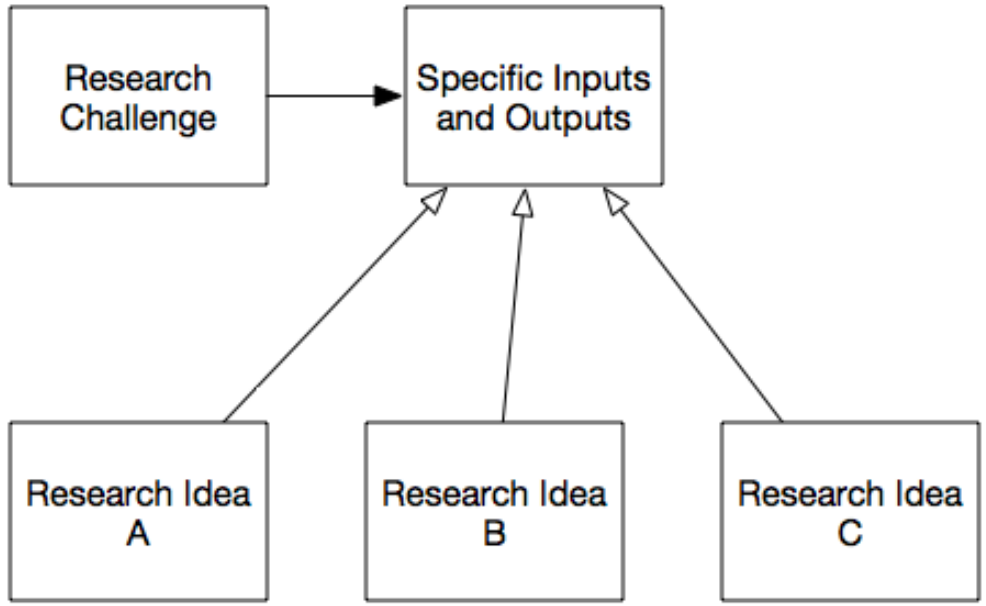
- Plugin architecture is very flexible. Easy to whip up complex simulations.
- Inherently scriptable and easily extendable into other python workflows.
- Substitutability gives robustness. If one plugin doesn't solve your problem, you can swap to another.
- Developing new plugins is fast because you can leverage existing work and only add what's new.
- Each component can be developed and tested individually!

Cons

- Plugins still need to be compiled, with a compatible set of compilers, and external libraries
- Compiling with -fPIC (Position Independent Code) appears to have a performance penalty of a few percent (4-6%).

Dependency Inversion for Research

- There is a clear separation from application developers using an interface, and research teams looking to implement it.
- Independent research teams can work on their own implementation, but by following the set interface they can all be interchanged for comparison.
- Technology improvements can be developed and tested in isolation without worrying about the complexity of the software that surrounds it.
- When the time comes, it is trivial to integrate that new technology into a larger body of work.



Alternatives to Virtual Dispatch

If you absolutely cannot use abstract base classes, plugins, or pointers, there are other techniques for SOLID software design.

Needing top-level performance is not an excuse for poor software architecture.

```
template <typename ProductionModel>
class SST {
 public:
  SST() = default;
  double calculateProduction(double k,
                             double omega,
                             double shear,
                             double vorticity) {
    const double eddy_viscosity = k / omega;
    return production.calc(eddy_viscosity, shear, vorticity);
  };
 private:
  ProductionModel production;
};
```

```
struct KatoLaunderProduction {
  double calc(double eddy_viscosity,
              double shear,
              double vorticity) {
    return eddy_viscosity * shear * vorticity;
  }
};

struct VorticityProduction {
  double calc(double eddy_viscosity,
              double shear,
              double vorticity) {
    return eddy_viscosity * vorticity * vorticity;
  }
};
```

```
double k = 0.1; double omega = 10.0;
double shear = 1.0; double vorticity = 0.1;

SST<VorticityProduction> modelA();
auto Pk = modelB.calculateProduction(k, omega, shear, vorticity);
std::cout << "Production with SST-Vm is: " << Pk << std::endl;

SST<KatoLaunderProduction> modelB();
Pk = modelB.calculateProduction(k, omega, shear, vorticity);
std::cout << "Production with SST-KLm is: " << Pk << std::endl;
```

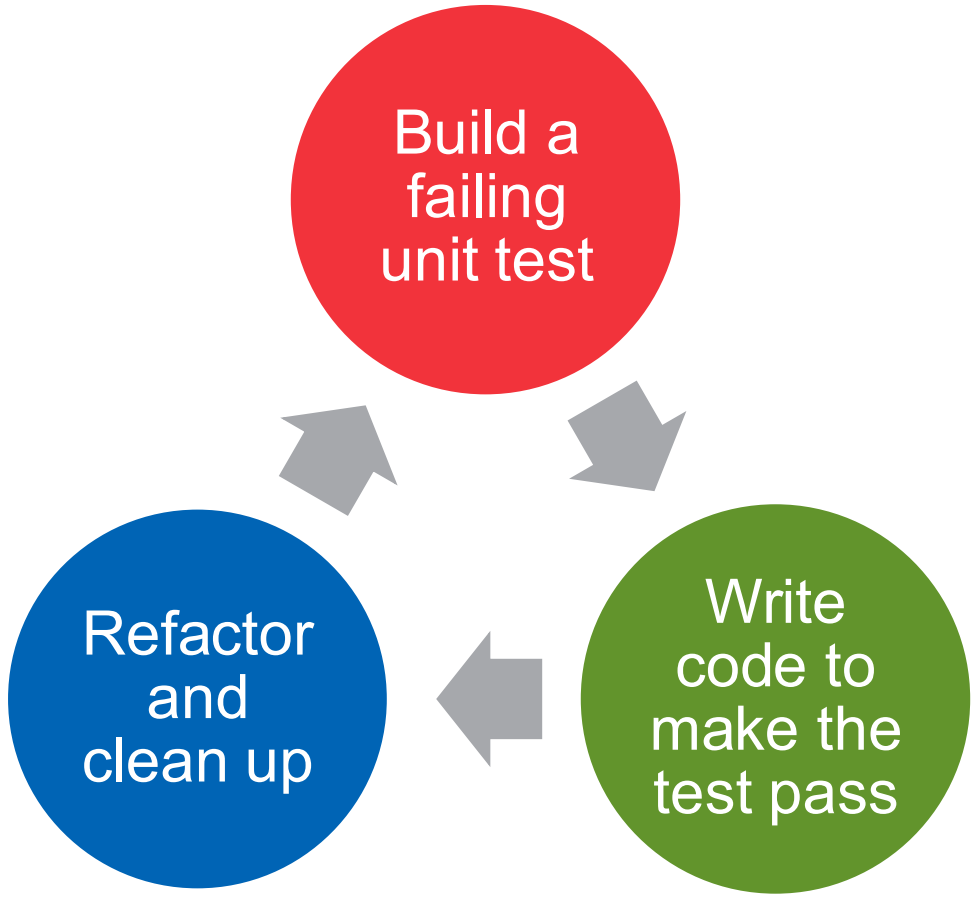
Polymorphism handled at compile-time, not runtime.

Note: This is an example that demonstrates concepts used in HyperSolve. It is **not** a release of actual HyperSolve code.

Test Driven Development[1]

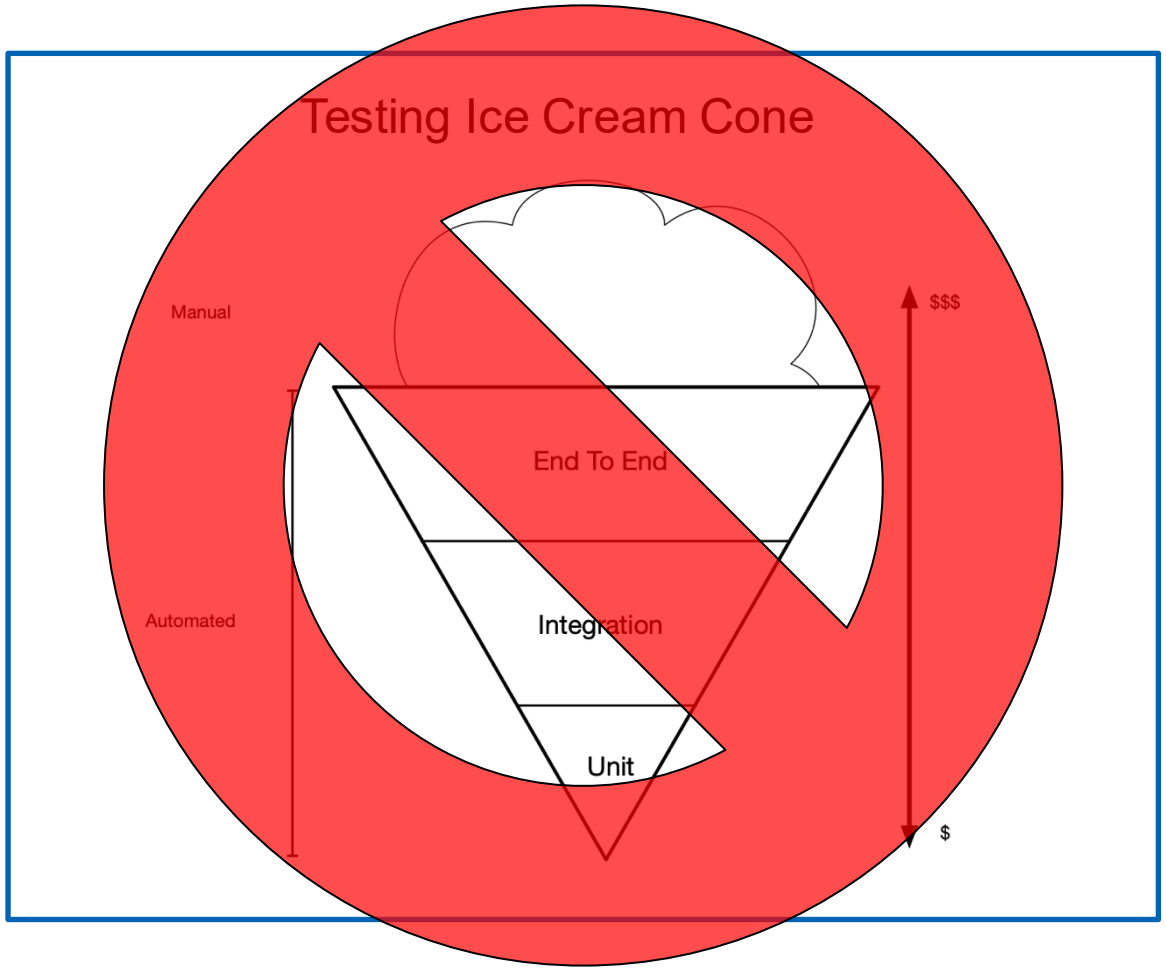
Put simply, tests for software should always be written first not last!

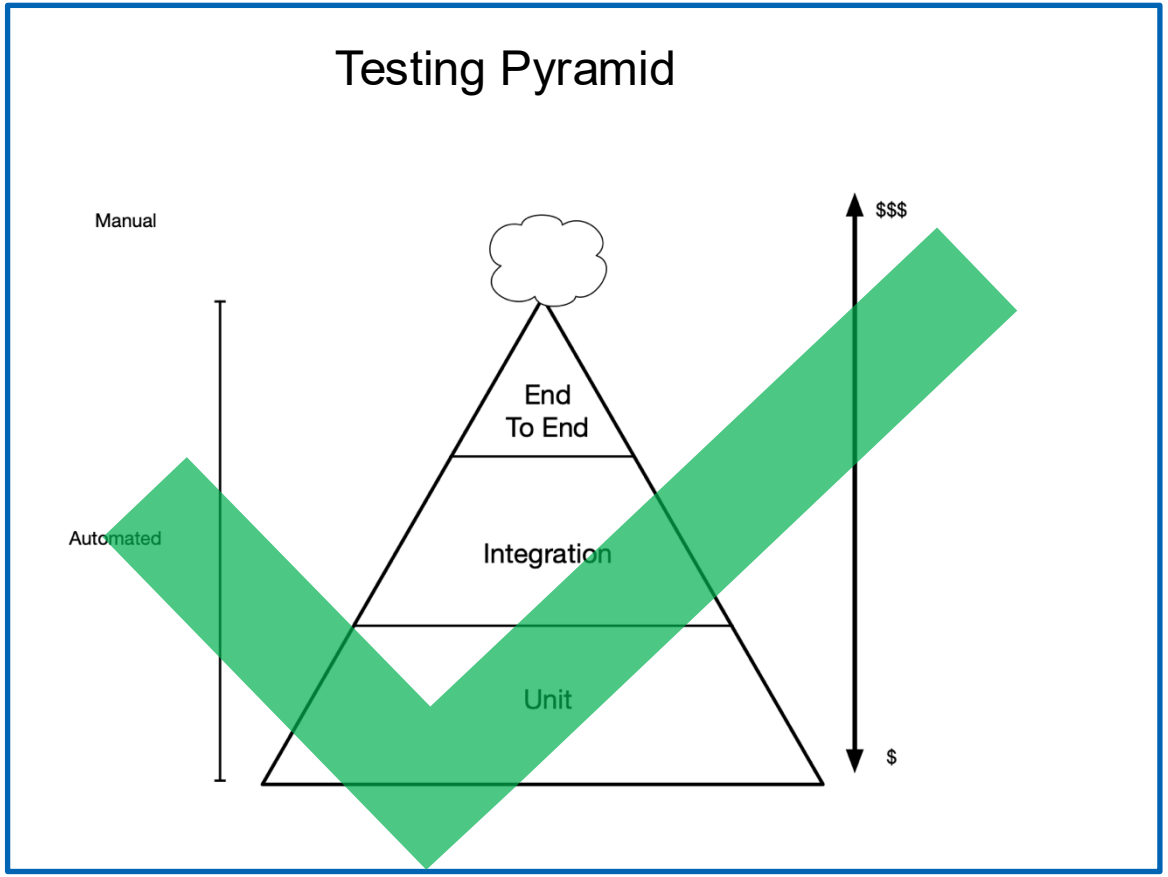
This process is typically iterative, where a failing test for a desired software component (i.e., function, class, etc.) is written first and then made to pass by implementing the software component



[1]Beck, K., “Test Driven Development: By Example.”, Addison-Wesley Longman, Amsterdam. 2002

Types of Software Testing





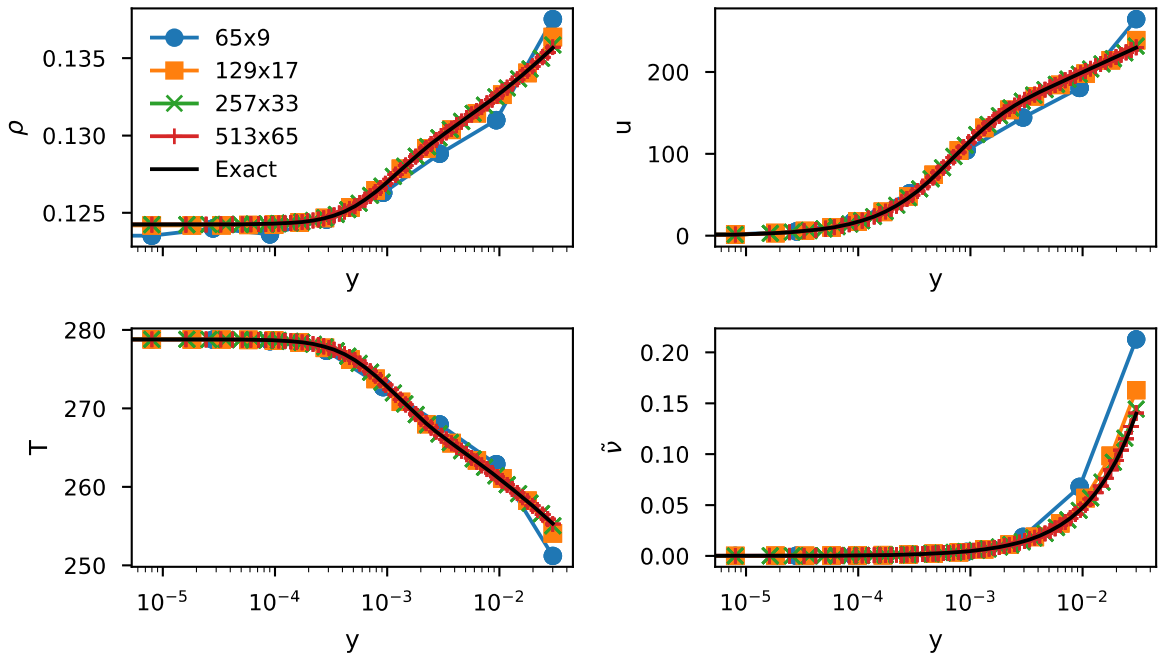
Unit Testing Example

```
TEST_CASE("Compute directed area of triangle") {
  Point a = {...};
  Point b = {...};
  Point c = {...};
  expected_tri_area = ...;
  REQUIRE(expected_tri_area == computeTriangleArea(a, b, c));
}
```

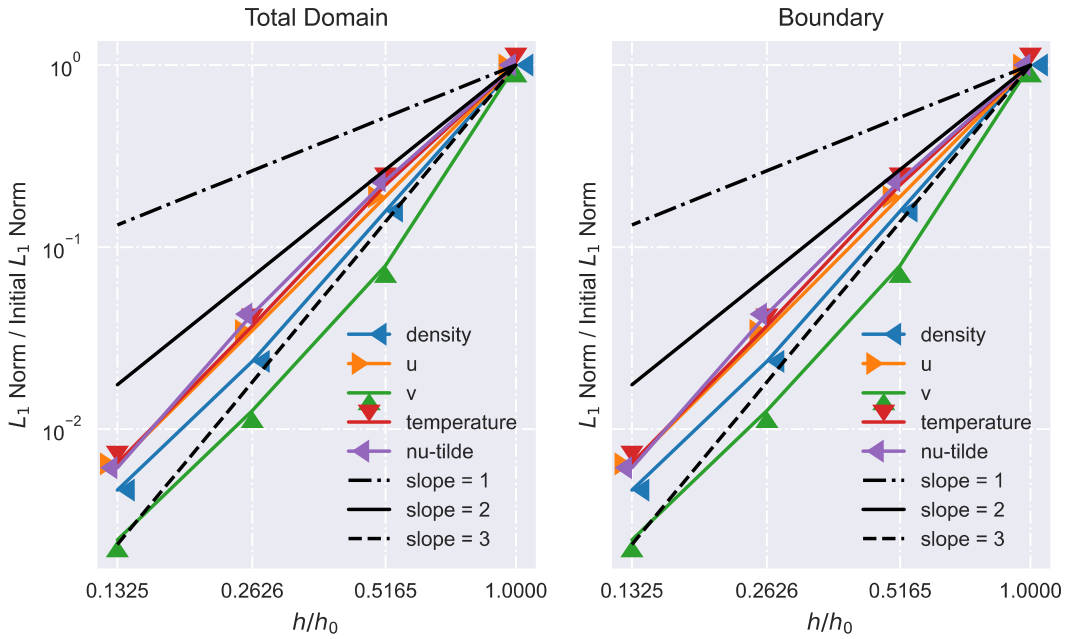
These tests should be (relatively) small and clear test one thing (i.e., function, class behavior, etc.)

Integration Testing Example

The order-of-accuracy of the implementation of the FANS equations in HyperSolve was verified[1] using with "physical" or "wall-bounded" manufactured solutions[2], which use an idealized boundary layer.



Profiles of density, streamwise velocity, temperature, and the turbulent variable in the wall-normal direction (y)



L_1 error under grid refinement. Both the L_1 error and the grid scale are normalized by the values on the coarsest grid.

[1]Thompson, K., et al. "Validation of the HyperSolve CFD Solver for Entry Descent and Landing Applications." AIAA Paper 2024-1973, 2024.
[2]Oliver, T., Estacio-Hiroms, K., Malaya, N., and Carey, G., "Manufactured Solutions for the Favre-Averaged Navier-Stokes Equations with Eddy-Viscosity Turbulence Models," AIAA Paper 2009-1575, 2012.

End-to-End Testing Example

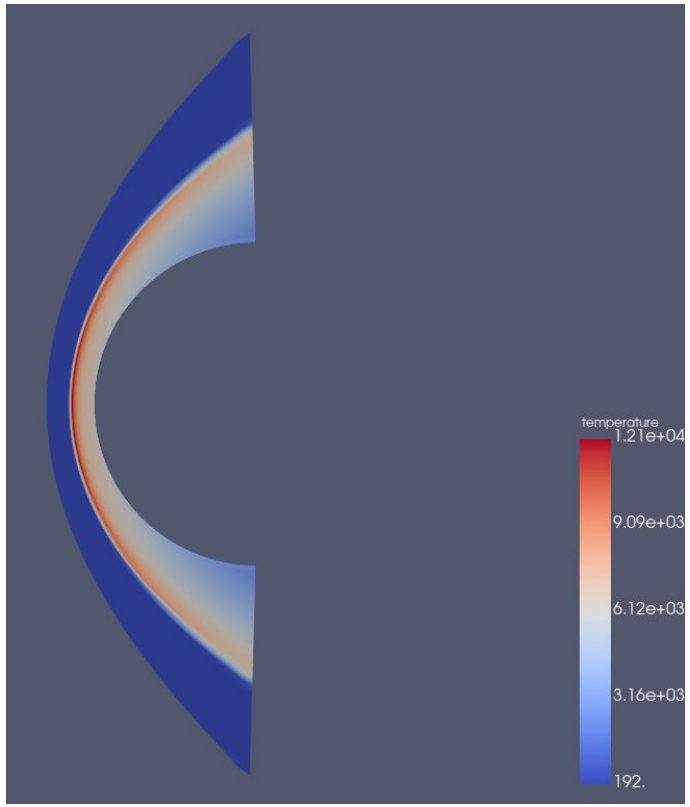
This is a simple example of a regression test we run regularly where HyperSolve is run on a ***structured mesh*** and compared against a LAURA solution on that same mesh.

Exercised Capabilities

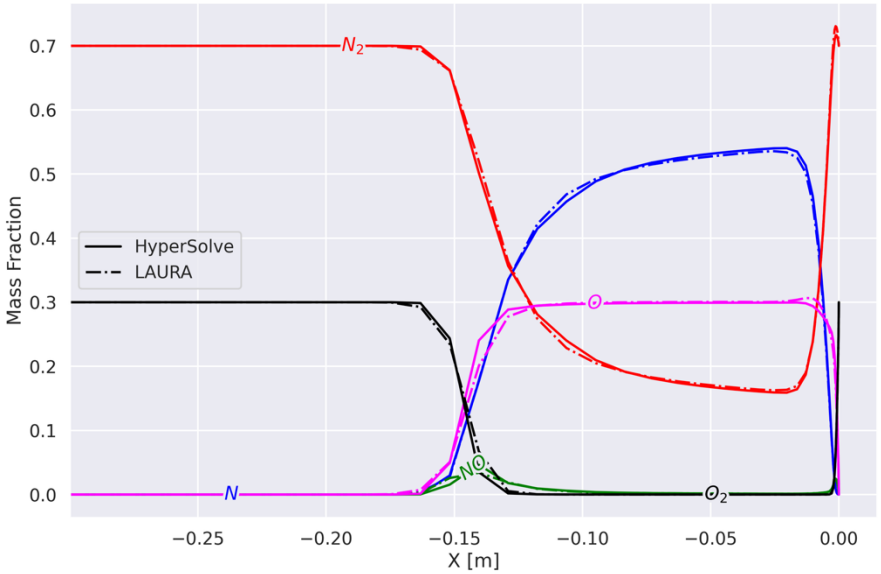
- 5-Species Air Model
- Wall Conditions:
  - Isothermal T = 500 K
  - Super-catalytic surface

Freestream Conditions	
Density	0.0001 *kg/m³*
Velocity	8000 *m/s*
Temperature	200.0 *K*

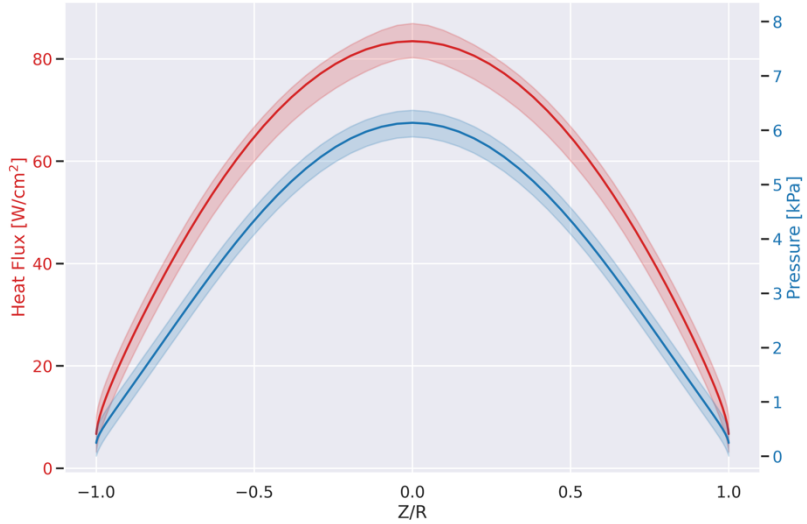
Species	Freestream Mass Fraction
N_2	0.70
O_2	0.30
N	1E-20
O	1E-20
NO	1E-20



Temperature field predicted by HyperSolve.



Comparison of mass fractions along the stagnation line HyperSolve (solid lines) and LAURA (dashed lines).



Comparison of heat flux and pressure between HyperSolve (lines) and LAURA +/- 5% (shaded regions).

Observations

Common (but bad) assumptions:

- This is the only application running.
- It is only running one instance at a time.
- It is only running on this computer.

Reduce assumptions as much as possible.

- The next developer won't know them.
- You will forget them.
- They cause undocumented software couplings.
- They will cause bugs.
- Write tests to ensure assumptions are met. Give human readable errors when they are violated.
- Often more general implementations are easier to implement and test.
- Use well documented conventions when not practical. Even better if those conventions are community wide.

Part 4 Summary

- The Dependency Inversion Principle: rely upon abstractions, not concrete implementations in software
  - Abstractions lower cognitive load by decoupling software components
  - HyperSolve uses this to great effect in a plugin ecosystem
  - Allows researchers to build libraries that are easily incorporated later

- Test Driven Development: write tests before the actual code
  - Prioritize writing tests based on the Testing Pyramid, not the Testing Ice Cream Cone

Acknowledgments

- This work is funded and supported by :
  - The Entry Modeling and Instrumentation Project within the NASA Game Changing Development Program, Space Technology Mission Directorate.
  - The High-Speed Flight Project within the NASA Aeronautics Research Mission Directorate.
  - The NASA Langley Science Directorate and the NASA Science Mission Directorate.

- Thanks to Kyle Thompson, Bil Kleb, Matt O'Connell, Chip Jackson, Drew Hinkle, Amar Potturi, and Bill Wood for their review of this lecture and for their contributions to development of the HyperSolve CFD tool.

QUESTIONS?

BACKUP SLIDES

Application to Navier-Stokes Equations

- Most recent and advanced application of hyperbolic methodology to compressible Navier-Stokes equations involves solving four times more equations than the standard system[1]
  - Memory requirement is a concern for multispecies flows

Navier-Stokes

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = 0$$
$$\partial_t (\rho u) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} - \tau) + \nabla p = 0$$
$$\partial_t (\rho E) + \nabla \cdot (\rho H \mathbf{v} - \tau \mathbf{v} + \mathbf{q}) = 0$$

Hyperbolic Navier-Stokes (HNS20)

$$\partial_t \rho + \nabla \cdot (\rho \mathbf{v}) = \nabla \mathbf{r}$$
$$\partial_t (\rho u) + \nabla \cdot (\rho \mathbf{v} \otimes \mathbf{v} - \tau) + \nabla p = 0$$
$$\partial_t (\rho E) + \nabla \cdot (\rho H \mathbf{v} - \tau \mathbf{v} + \mathbf{q}) = 0$$

9 velocity gradient equations → $\frac{T_v}{\mu_v} \partial_t (\mathbf{g}) - \nabla \mathbf{v} = -\frac{\mathbf{g}}{\mu_v}$

3 heat flux equations → $\frac{T_h}{\mu_h} \partial_t (\mathbf{q}) - \nabla T = -\frac{\mathbf{q}}{\mu_h}$

3 density gradients equations → $\frac{T_r}{\mu_r} \partial_t (\mathbf{r}) - \nabla \rho = -\frac{\mathbf{r}}{\mu_r}$

[1]Nakashima, Y., Norihiko W., and Nishikawa, H.. "Hyperbolic Navier-Stokes solver for three-dimensional flows." AIAA Paper 2016-1101, 2011.

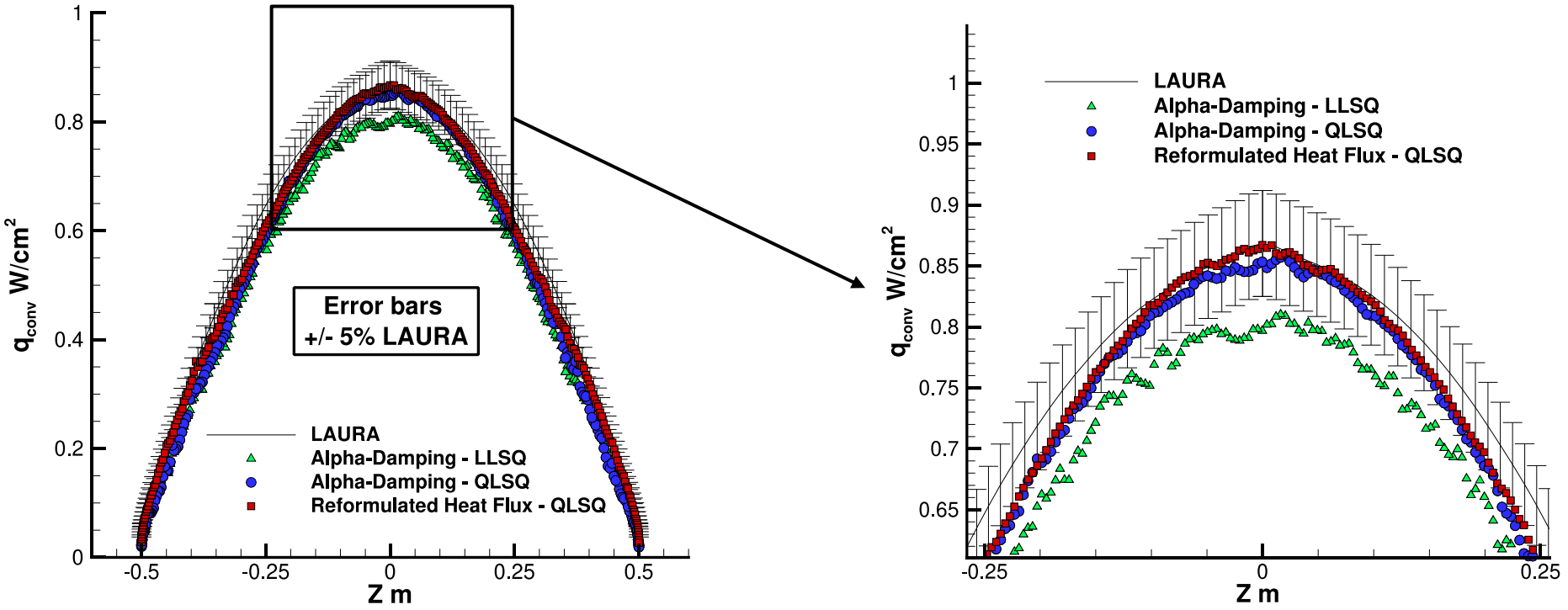
Pick and Choose: Mesh Adaptation


```
1  comm = getCommunicator()
2  preprocessor = getPreProcessor('path/to/PreProcessor/plugin')
3  mesh = preprocessor.importMesh('MeshFilename', comm)
4  metric_calculator = getMetricCalculator('path/to/MetricCalculator/plugin')
5  mesh_adaptation_plugin = getMeshAdaptation('path/to/Adaptation/plugin')
6
7  for step in number_of_adaptation_cycles
8    fluid_solver = getFluidSolver(mesh, comm, 'path/to/FlowSolver/plugin')
9    fluid_solver.setSolution(solution) unless step == 0
10   fluid_solver.solve()
11   mach = fluid_solver.field('mach')
12   metric = metric_calculator.calculate(mesh, comm, mach)
13   new_mesh = mesh_adaptation_plugin.adapt(mesh,comm,metric)
14
15   solution_variable_names = fluid_solver.expectsSolutionAs()
16   solution = fluid_solver.extractFields(solution_variable_names)
17   interpolator = getInterpolator(mesh, new_mesh, comm, refine_dir, refine_name)
18   interpolated_solution = interpolator.interpolate(solution)
19   fluid_solver.unloadPlugin()
20   mesh = new_mesh
21   solution = interpolated_solution
22 end
```

Example taken from: O'Connell, Matthew, et al. "Application of the dependency inversion principle to multidisciplinary software development." AIAA Paper 2018-3856, 2018.

1 km/s 3D Laminar Flow over a Hemisphere

Heat flux at Y=0 centerline slice



Both alpha-damping and RHFS schemes within 5% of stagnation heat flux predicted by LAURA with QLSQ

- Smoother heat flux profile with RHFS than fully-alpha-damping schemes