Hypersonic Task-based Research solver

Studying compressible reacting flows using Legion

Mario Di Renzo

Legion retreat, Dec 4-5, 2024

UNIVERSITÀ DEL SALENTO

Stanford ENGINEERING | Center for Turbulence Research

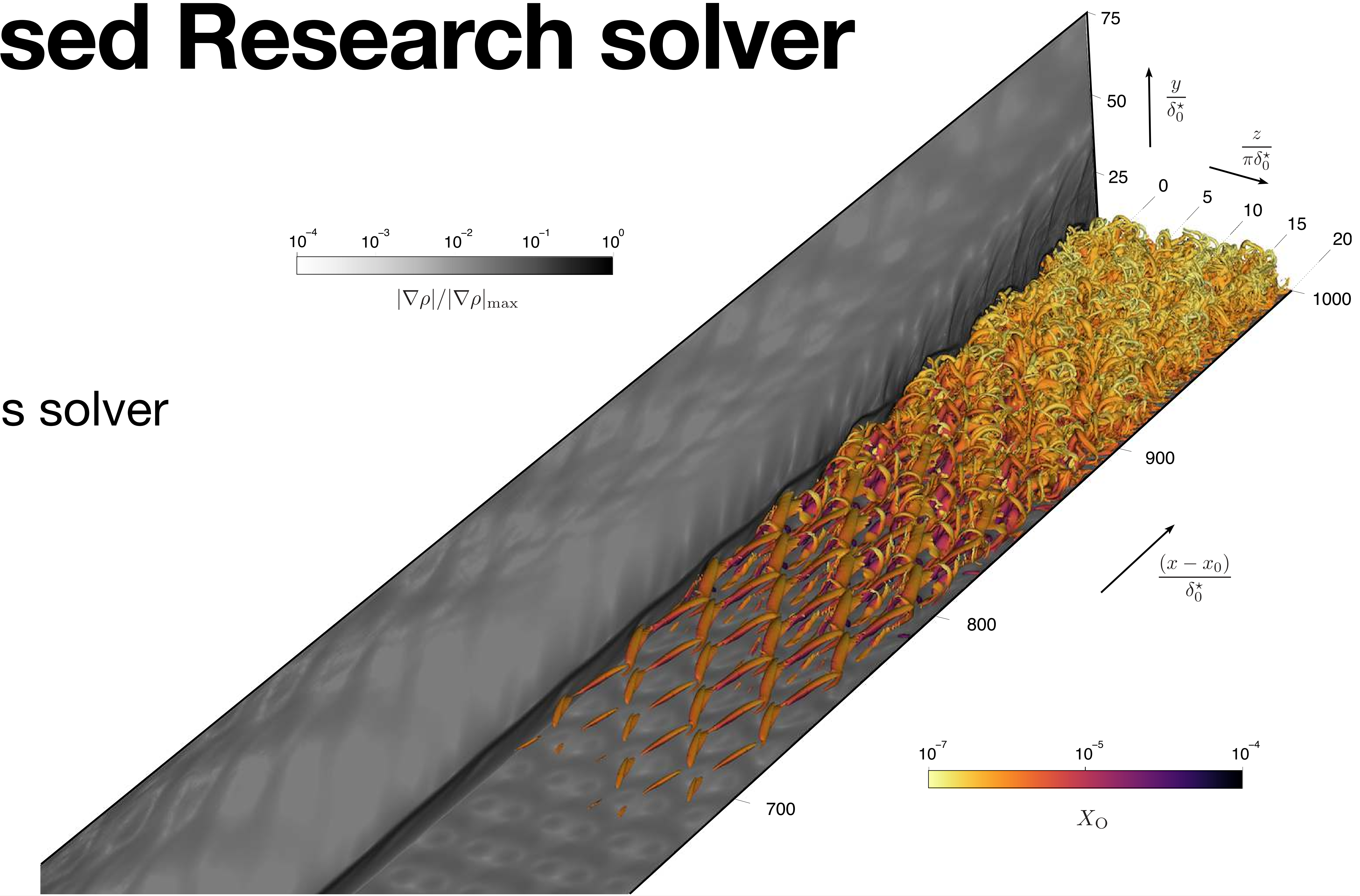
Content

What is HTR and its applications?

How is Legion leveraged to achieve performance?

The future of the HTR solver

The Hypersonic Task-based Research solver



Main features:

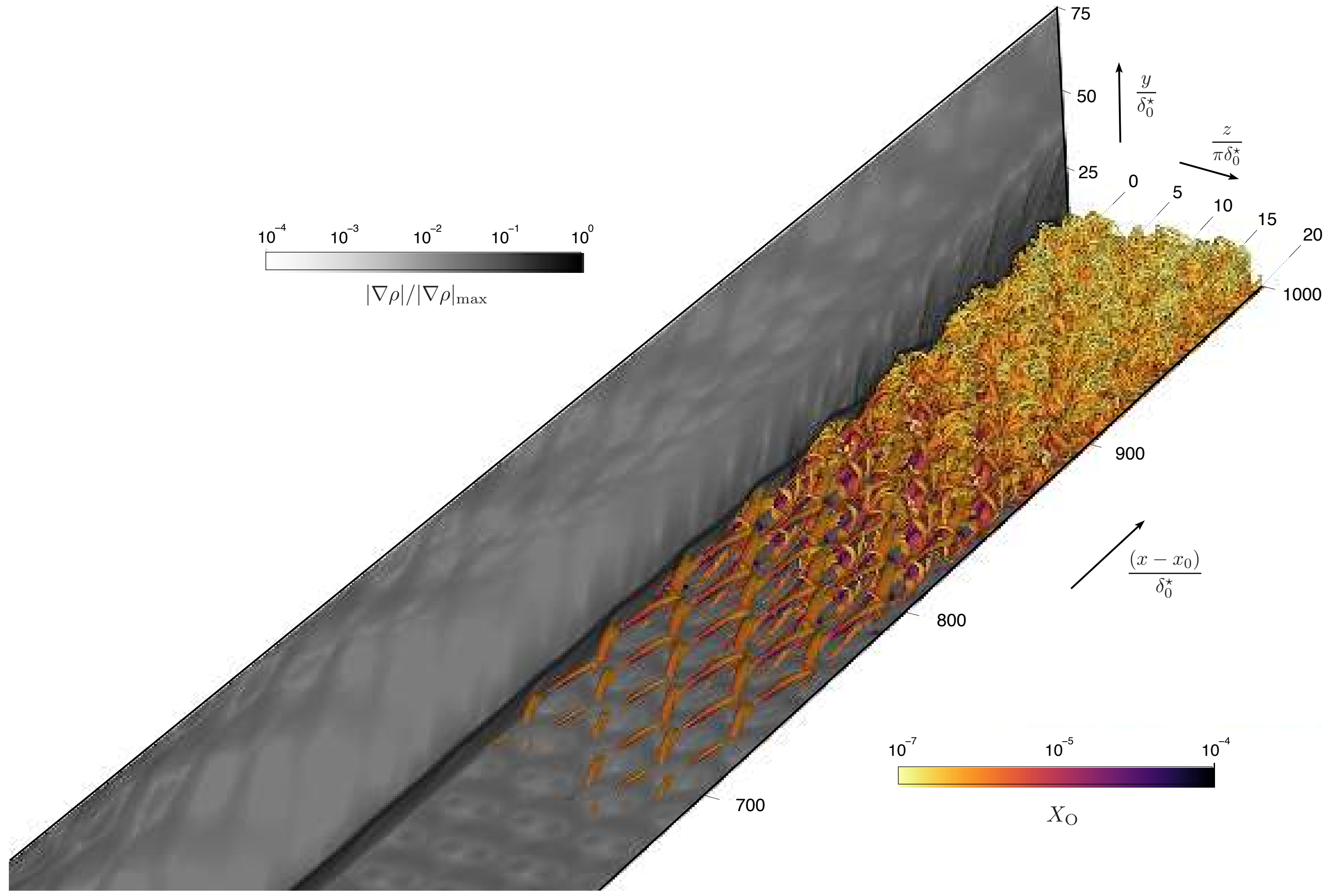
- Compressible multicomponent Navier—Stokes solver with finite-rate chemistry
- Formulated to handle curvilinear multiblock computational grids
- Compatible with NVIDIA and AMD GPUs

Further reading:

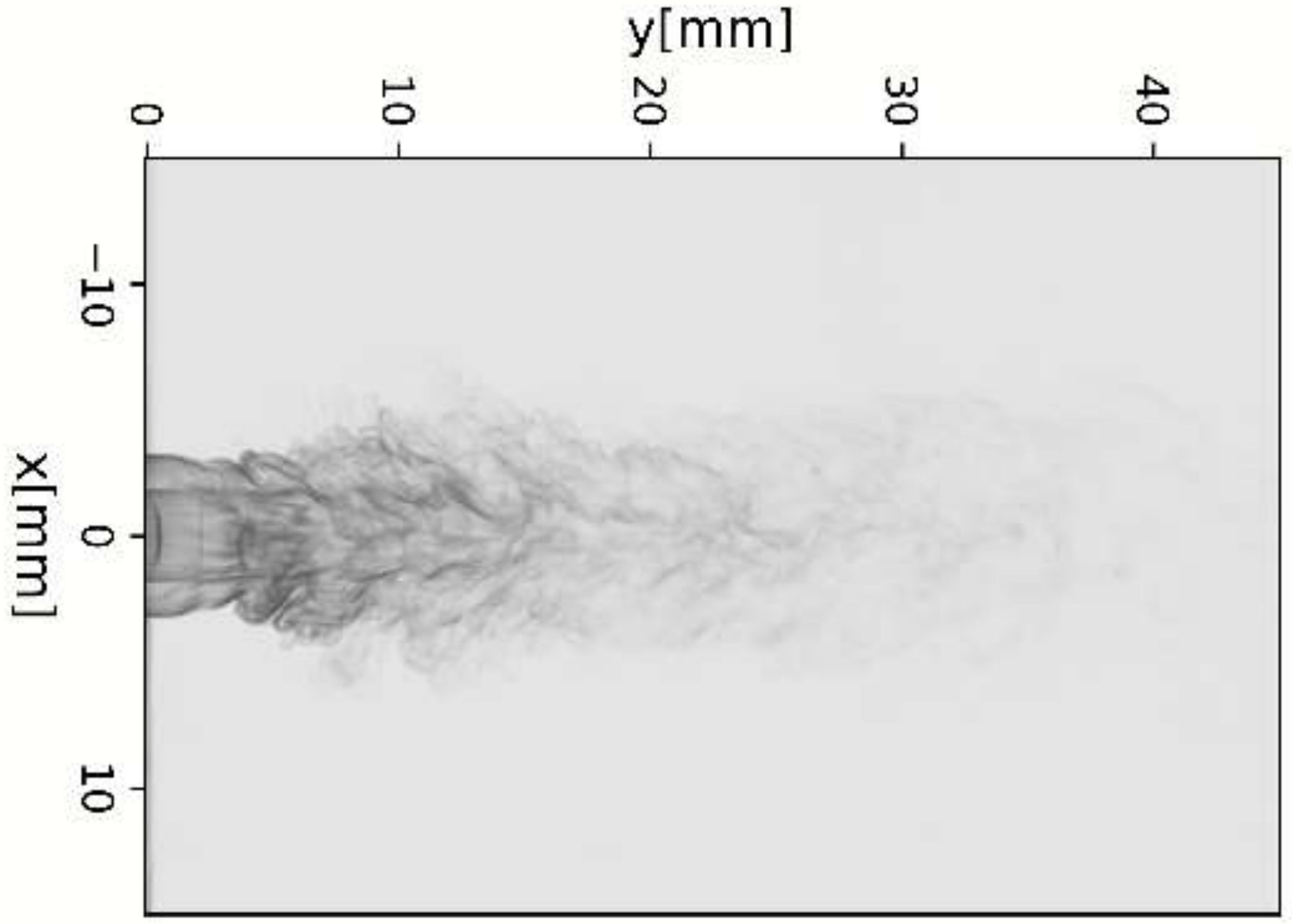
- Di Renzo, M., Fu, L. & Urzay, J. "HTR solver: An open-source exascale-oriented task-based multi-GPU high-order code for hypersonic aerothermodynamics." Computer Physics Communications 255 (2020), 107262
- Di Renzo, M. & Pirozzoli, S. "HTR-1.2 solver: Hypersonic Task-based Research solver version 1.2". Computer Physics Communications 261 (2021), p. 107733.
- Di Renzo, M. "HTR-1.3 solver: Predicting electrified combustion using the hypersonic task-based research solver". Computer Physics Communications 272 (2022), p. 108247.

Applications of the HTR solver

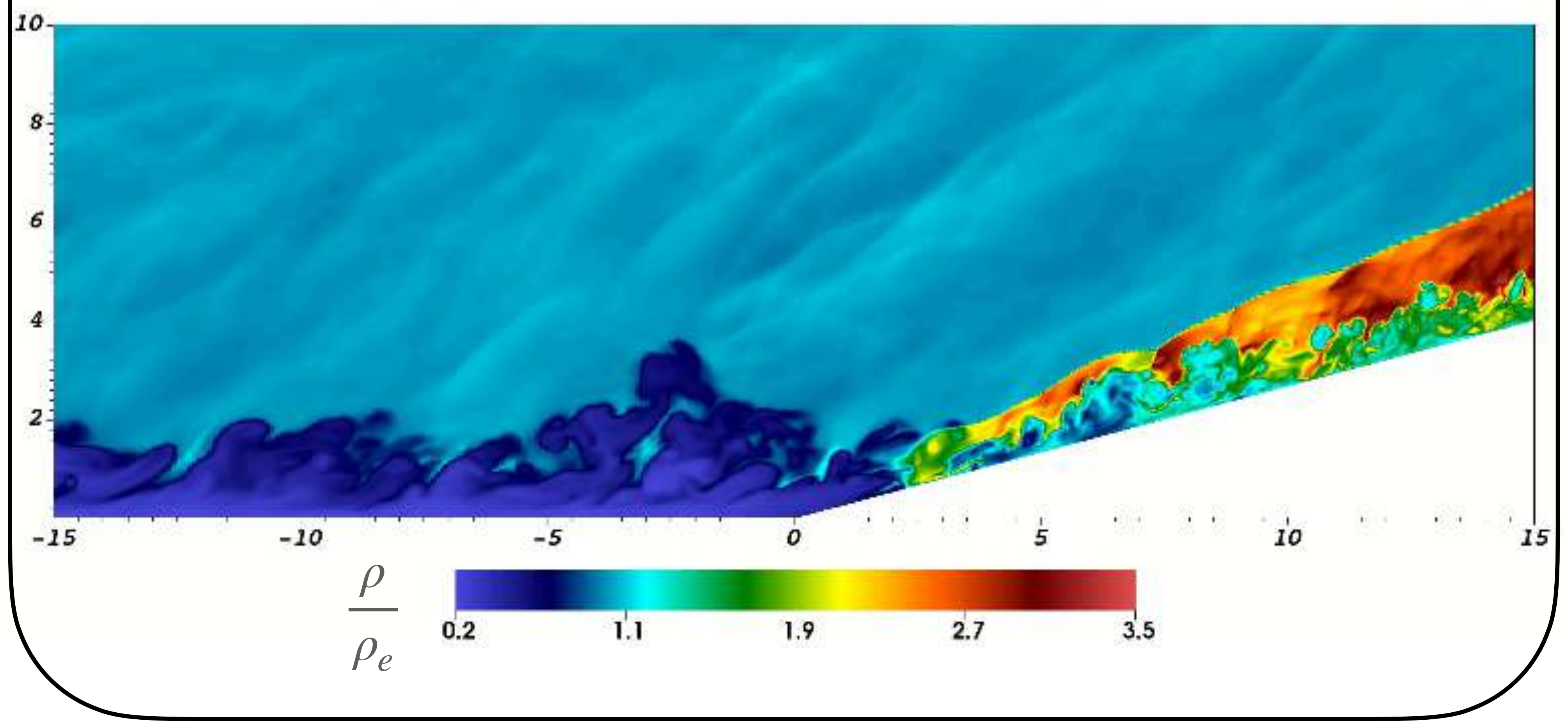
Hypersonic high-enthalpy boundary layers



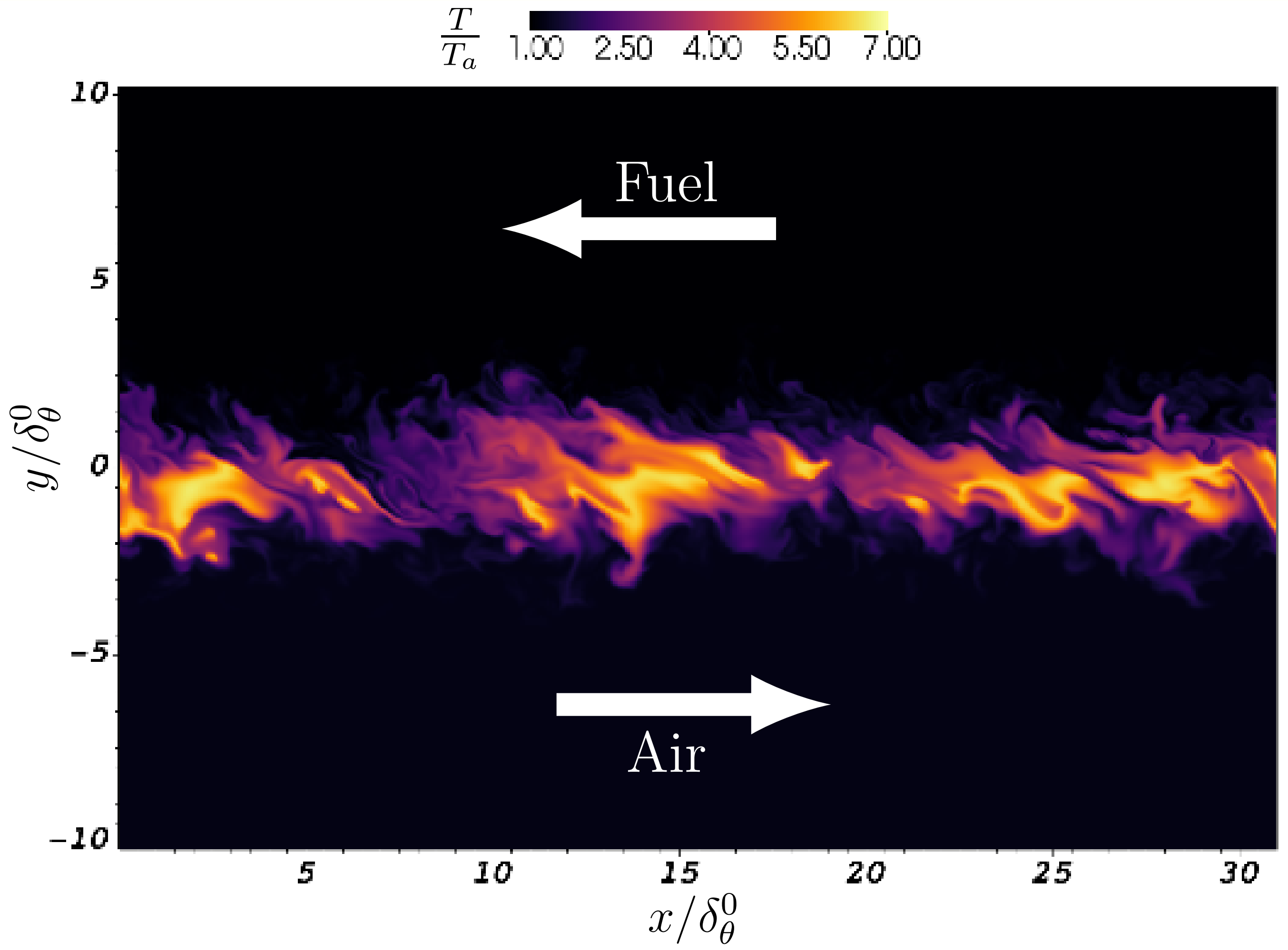
Rocket burner ignition



Shock-wave/turbulence interactions



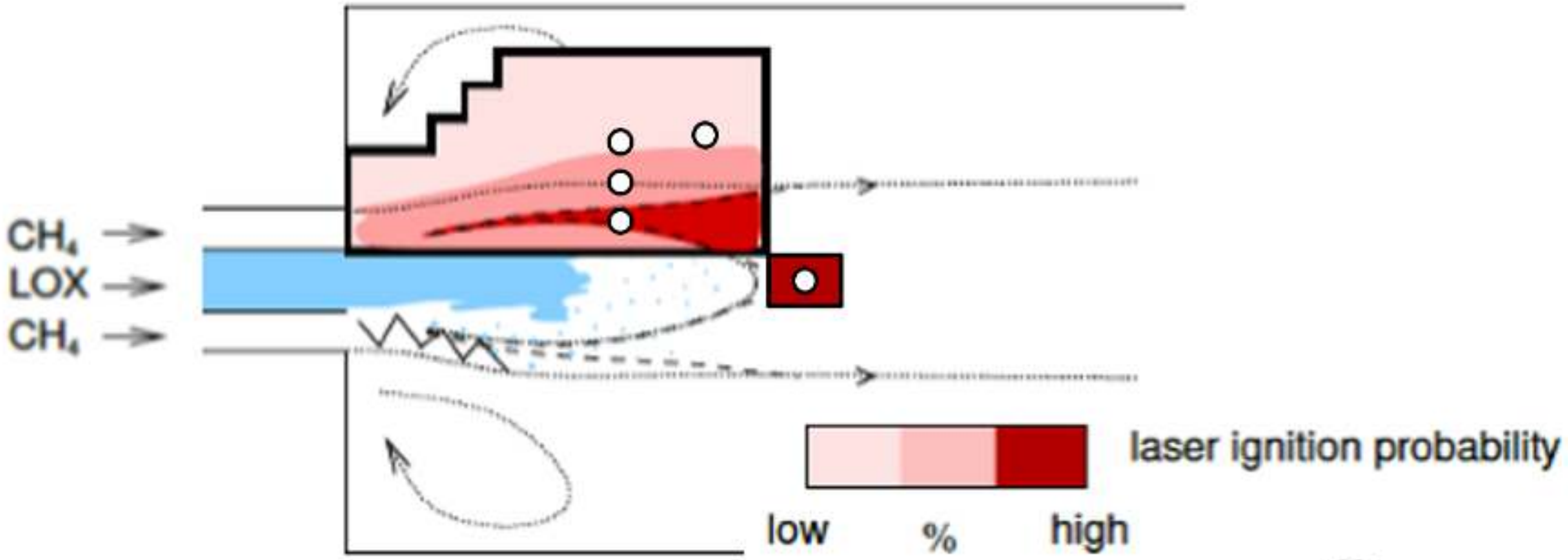
Electrified flame simulations

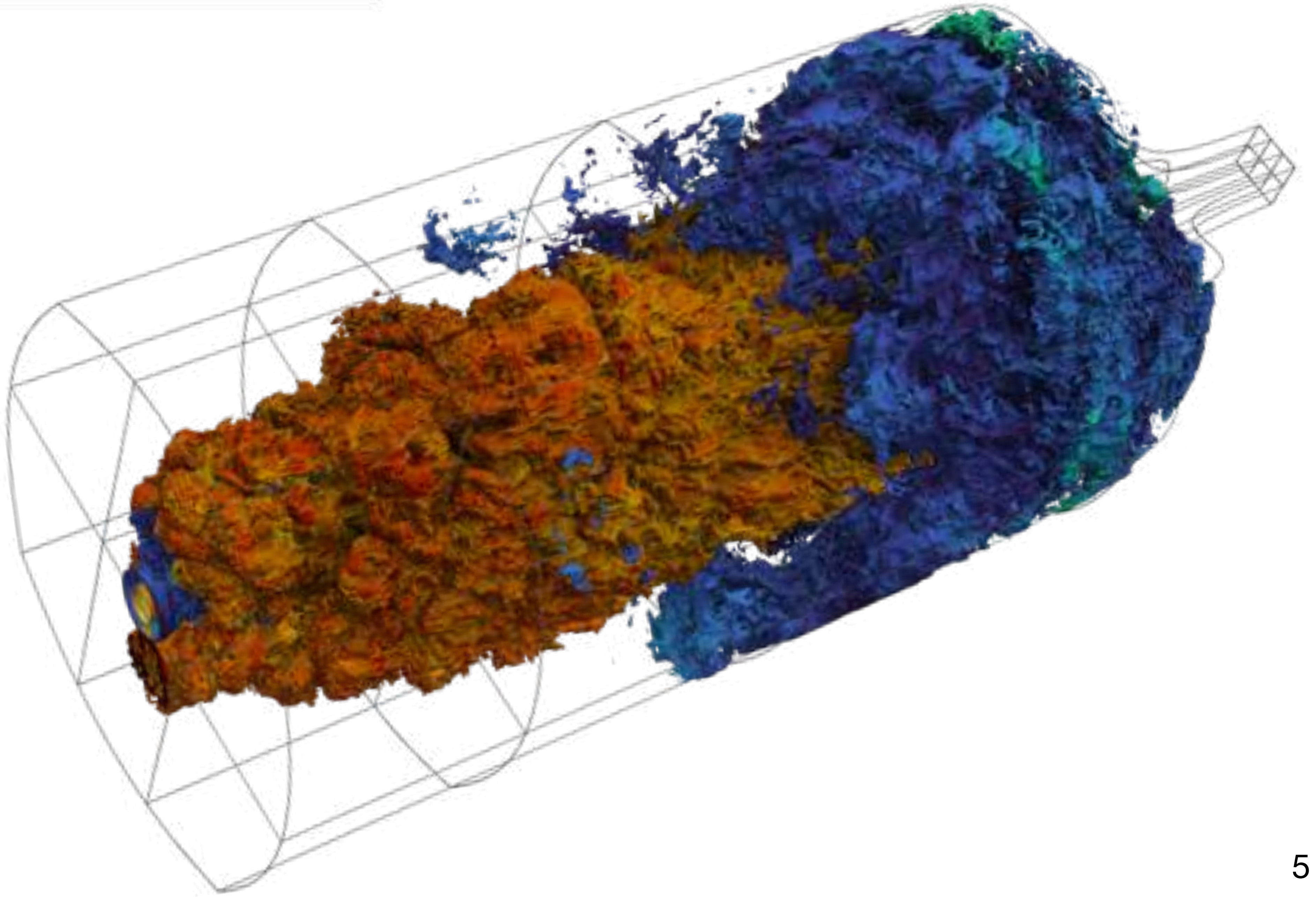


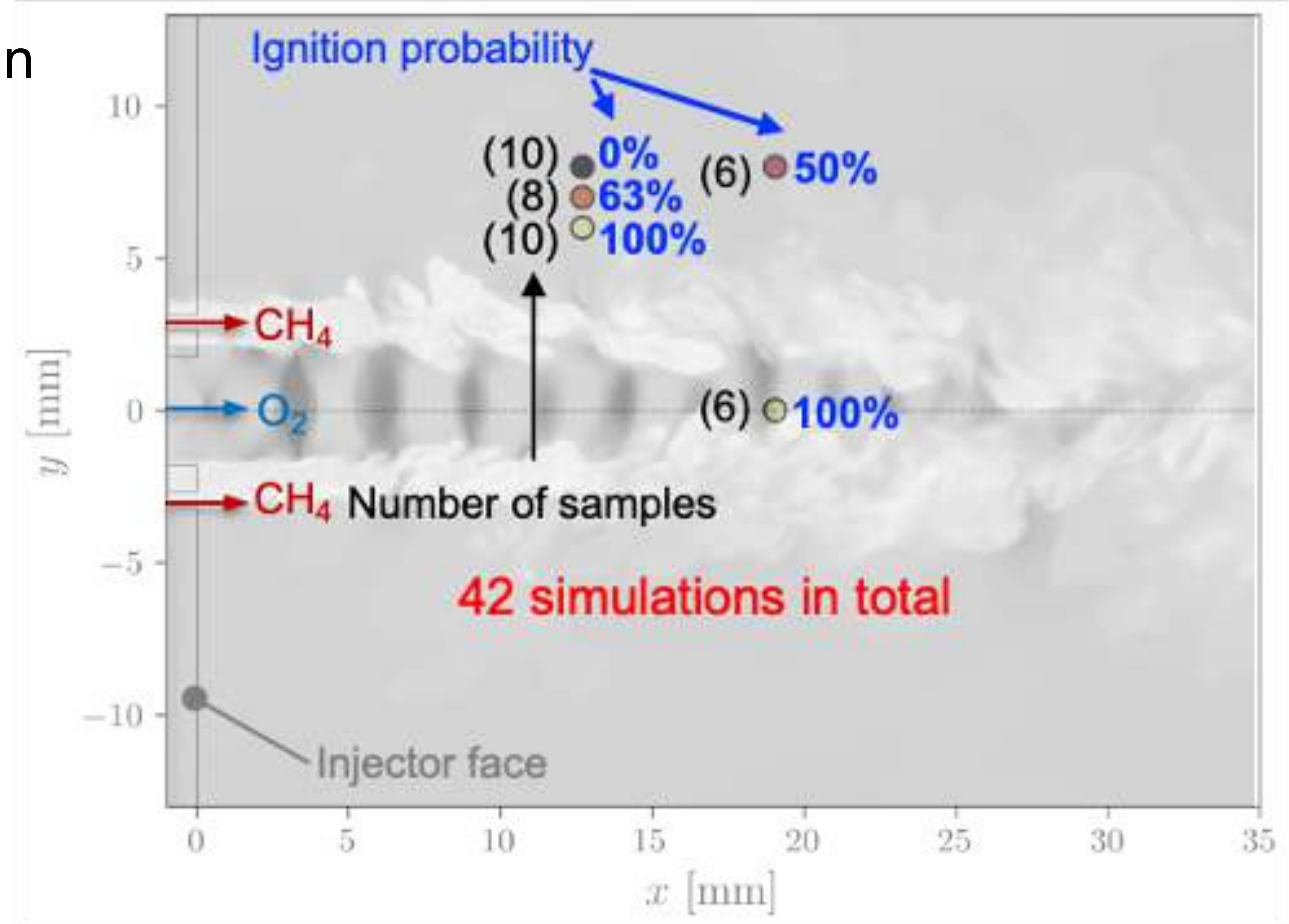
Integrated Simulations using Exascale Multiphysics Ensembles (INSIEME)

PI: Prof. Gianluca Iaccarino

Project Goal: build **probability maps** of laser-induced ignition success/failure in a rocket burner using computations validated by experiments



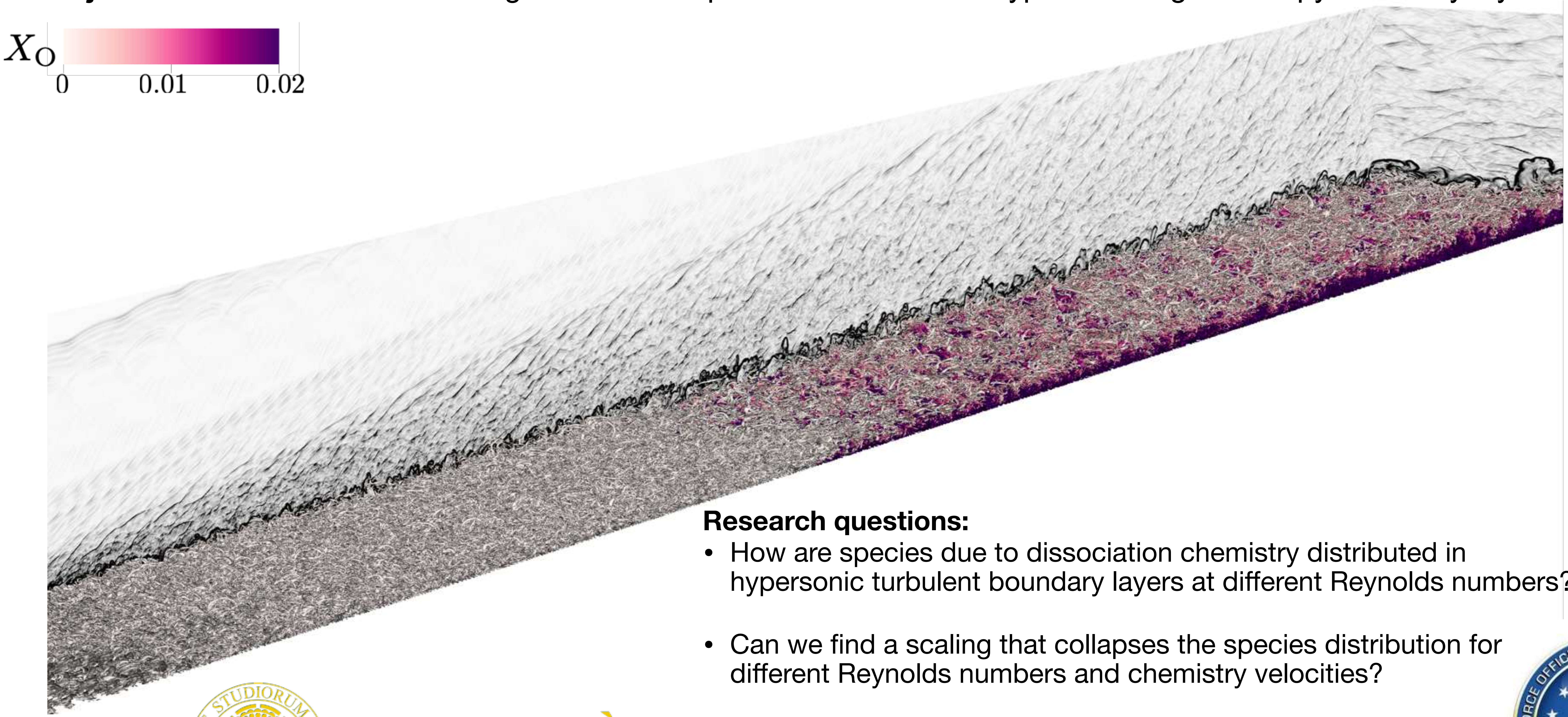




- Prediction of reliability of laser-based ignition of cryogenic propellants on a model rocket combustor
- Complex physics including compressible flow, phase change, turbulence, mixing, ionization, combustion
- Ignition probability maps obtained from O(105-106) concurrent multifidelity ensembles run on exascale machines with an efficient, portable HPC code

Properties of chemical species distribution in hypersonic boundary layers at high enthalpies
PI: Mario Di Renzo

Project Goal: determine the scaling of chemical species distribution in hypersonic high-enthalpy boundary layers



Research questions:

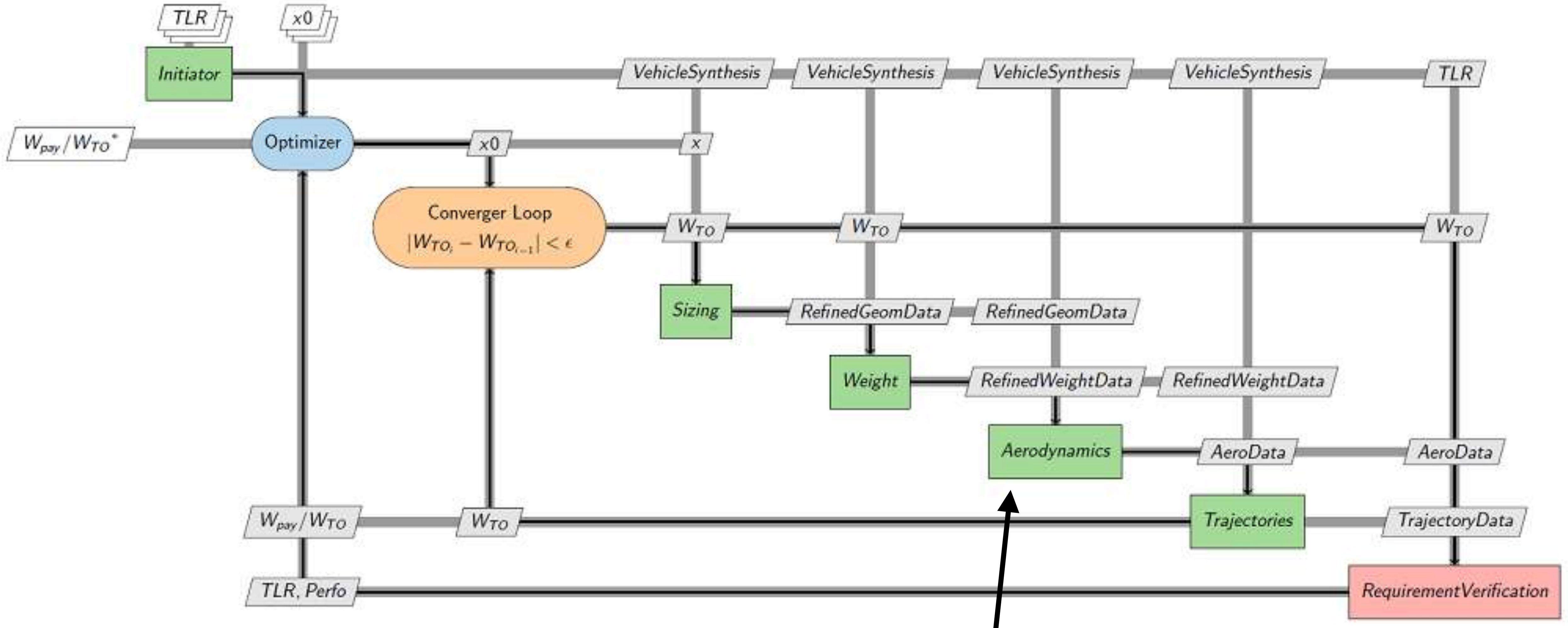
- How are species due to dissociation chemistry distributed in hypersonic turbulent boundary layers at different Reynolds numbers?
- Can we find a scaling that collapses the species distribution for different Reynolds numbers and chemistry velocities?
- Can we devise a reduced-order model for these distributions?

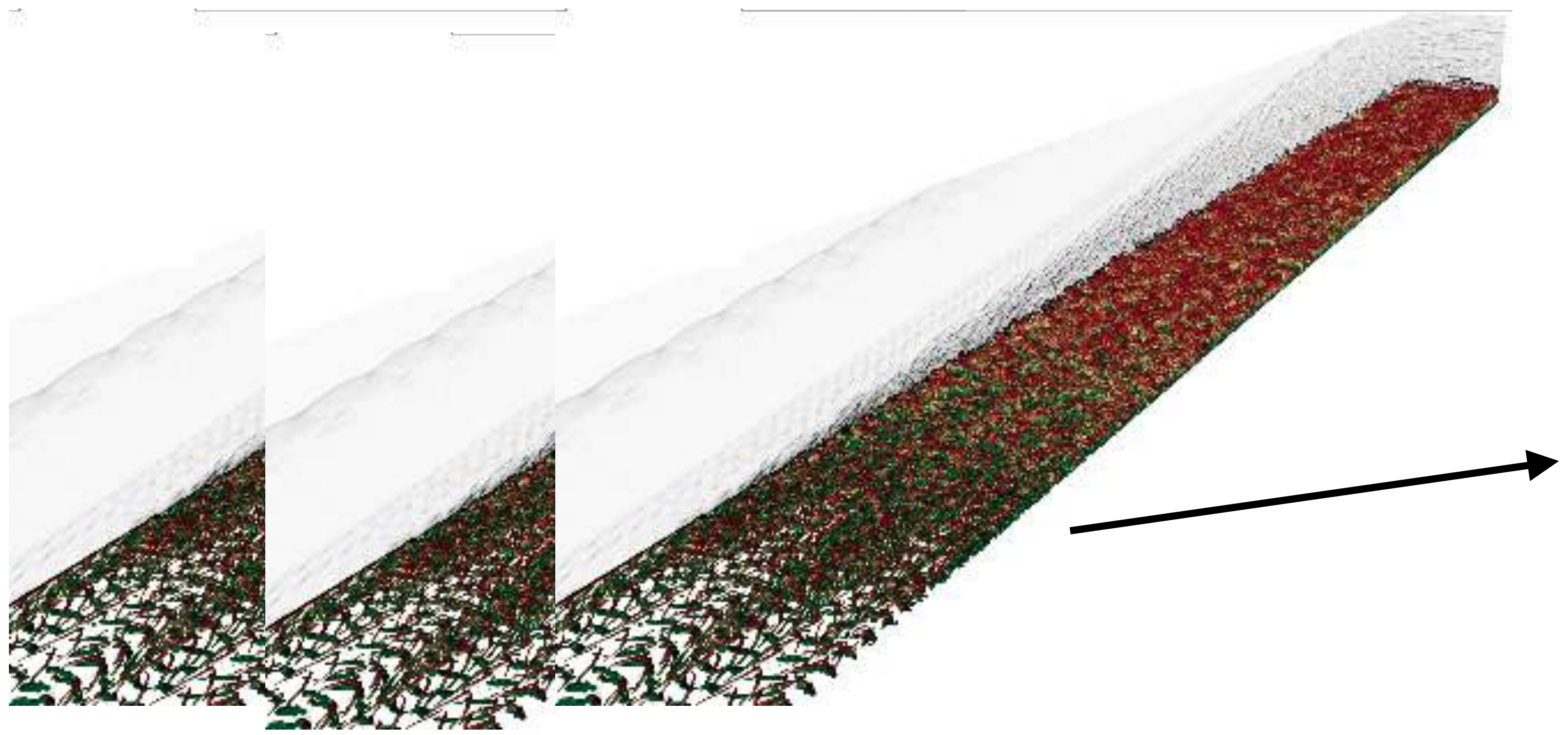
UNIVERSITÀ DEL SALENTO

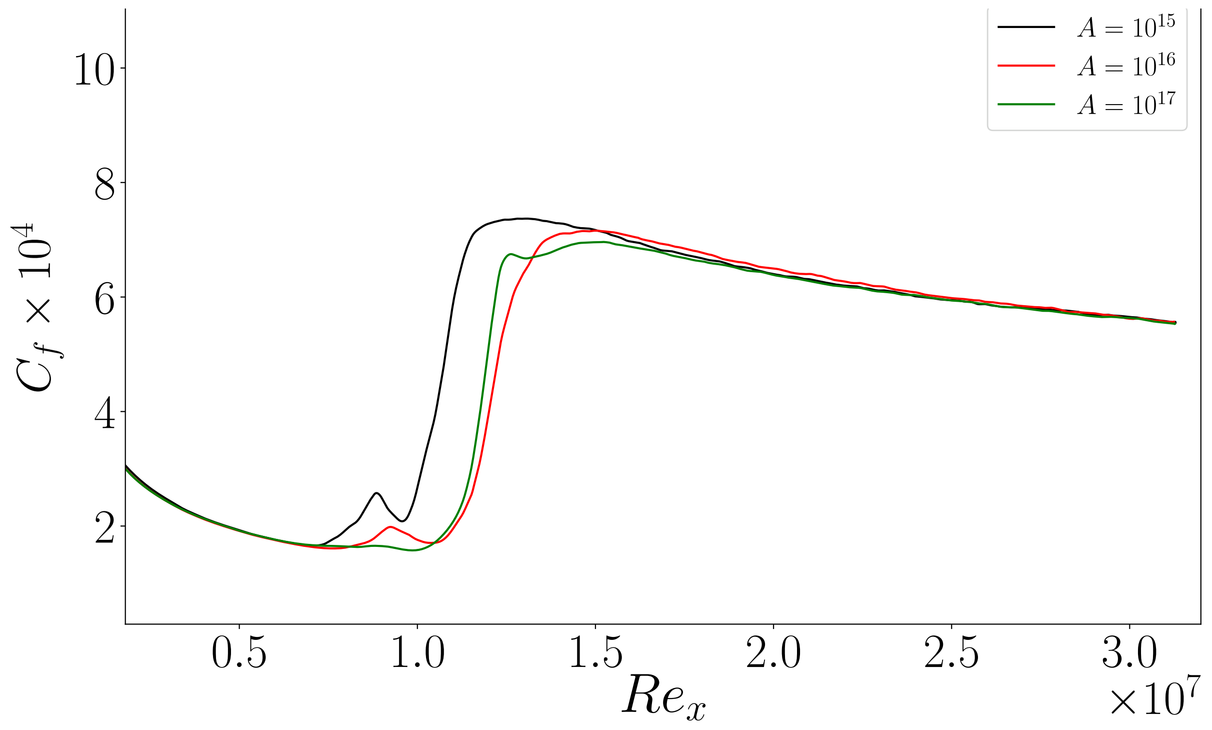
Integrated CONceptual DEsign tools for Suborbital vehicles (ICONDES)
PI: Mario Di Renzo

Project Goal: development of an integrated conceptual design tool for manned suborbital vehicles

- The design of suborbital vehicles involves multiple coupled disciplines:
  - Sizing and weight of the vehicle
  - Flight trajectory optimization
  - **Aerodynamic analysis**
- The HTR solver tunes the reduced aerodynamic models used during the conceptual design.







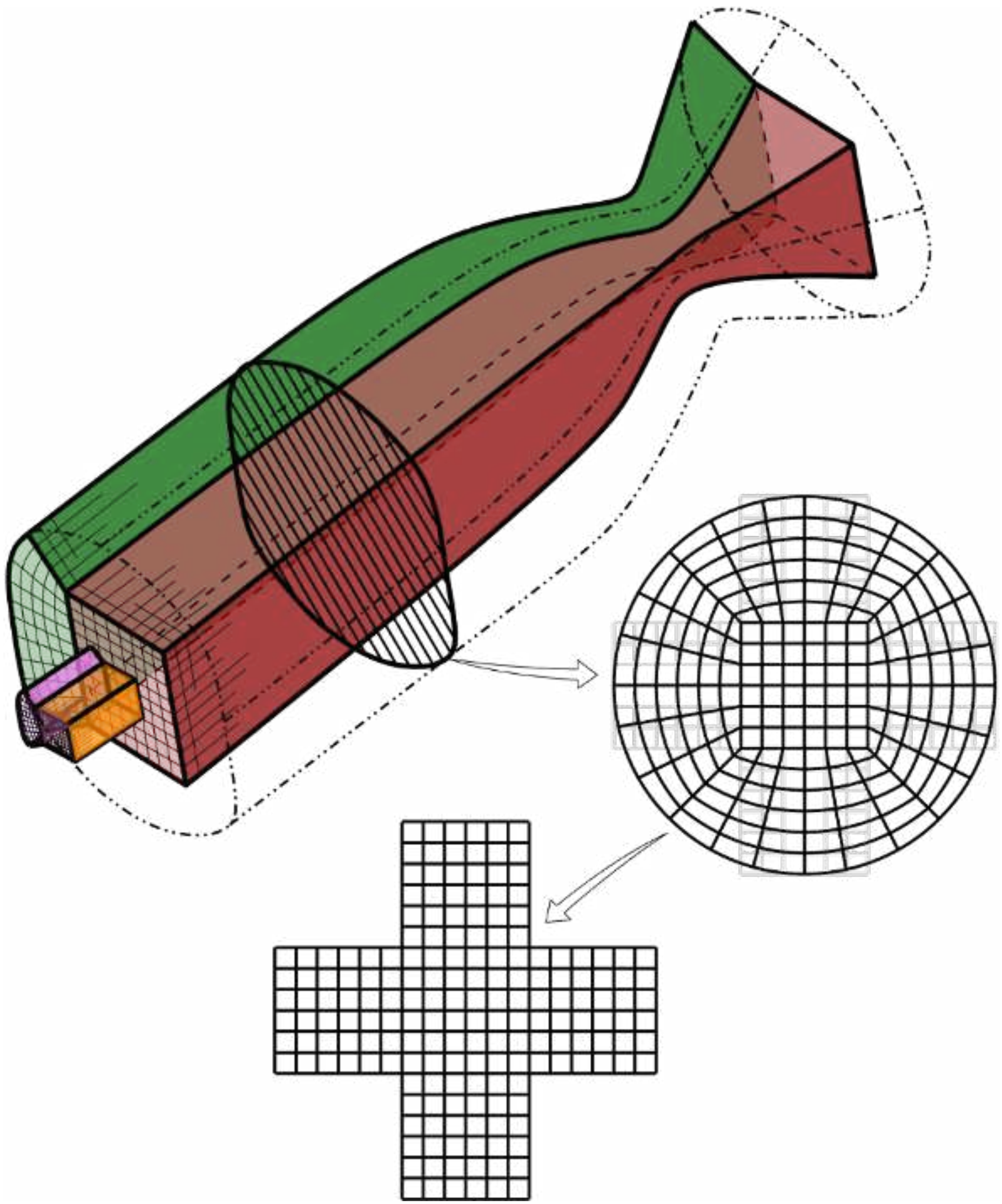
$$C_f = \frac{0.455}{(\log Re)^{2.58} + (1 + 0.144 Ma^2)^{0.65}}$$

Finanziato dall'Unione europea
NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani
PIANO NAZIONALE DI RIPRESA E RESILIENZA
UNIVERSITÀ DEL SALENTO

How is Legion leveraged to achieve performance?

Case study

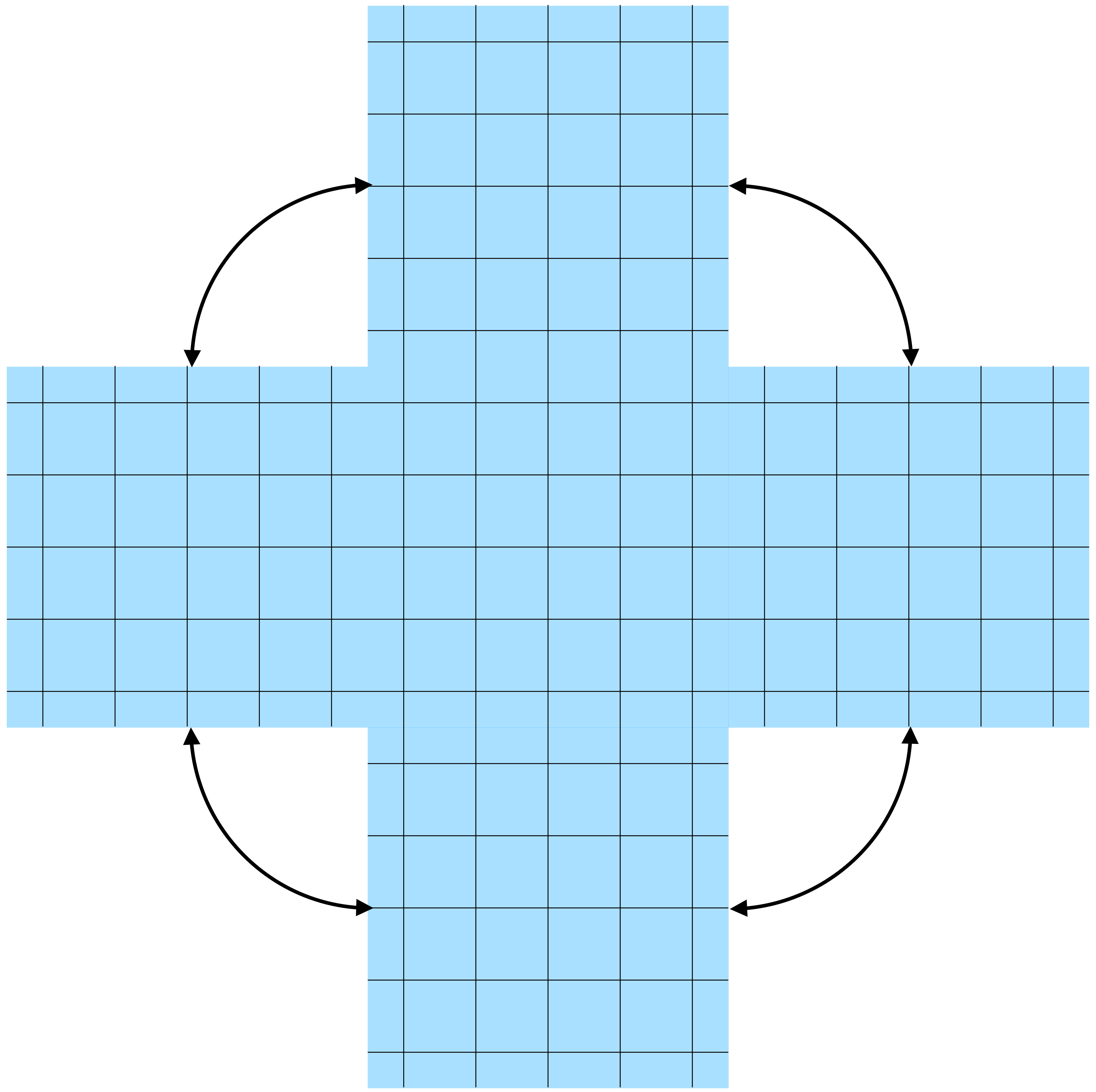
- The PSAAP III project investigates a rocket burner with complex geometry, where billions of degrees of freedom are necessary to describe the flow
- The index space of the computational domain utilized for these calculations features multiple blocks
- These blocks are connected in a non-trivial manner to describe the desired topology
- The HTR solver partitions this domain and executes several tasks with stencil accesses to update the solution



Credit: Alboreno Voci (albovoci@stanford.edu)

Case study

- The index space of a section of the PSAAP burner looks like a cross
- The lateral faces of the cross are connected to describe the domain topology

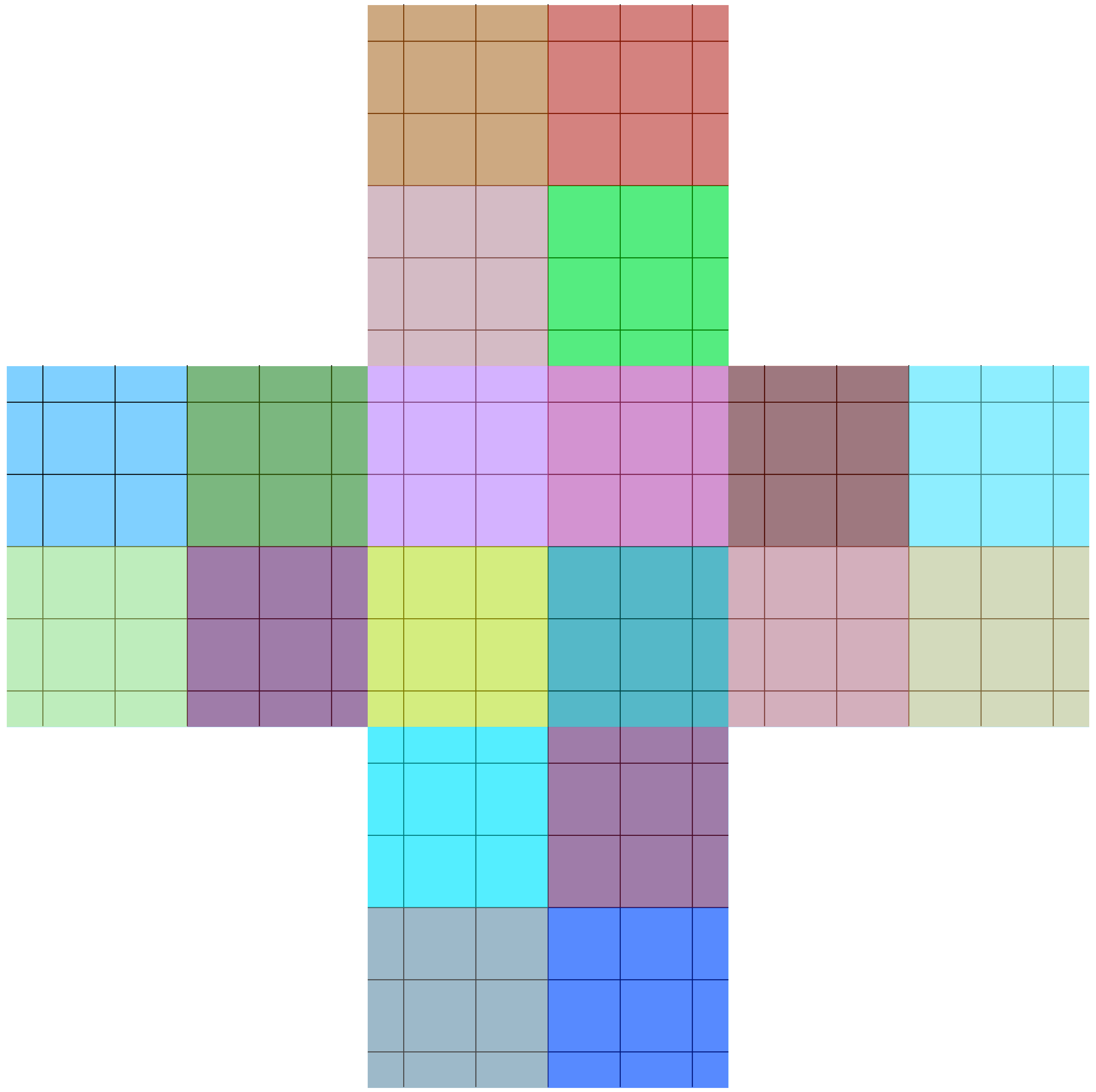


Credit: Alboreno Voci (albovoci@stanford.edu)

Dynamic scheduling

- The index space of a section of the PSAAP burner looks like a cross
- The lateral faces of the cross are connected to describe the domain topology
- HTR partitions this index space and solves a set of conservation equations within each partition

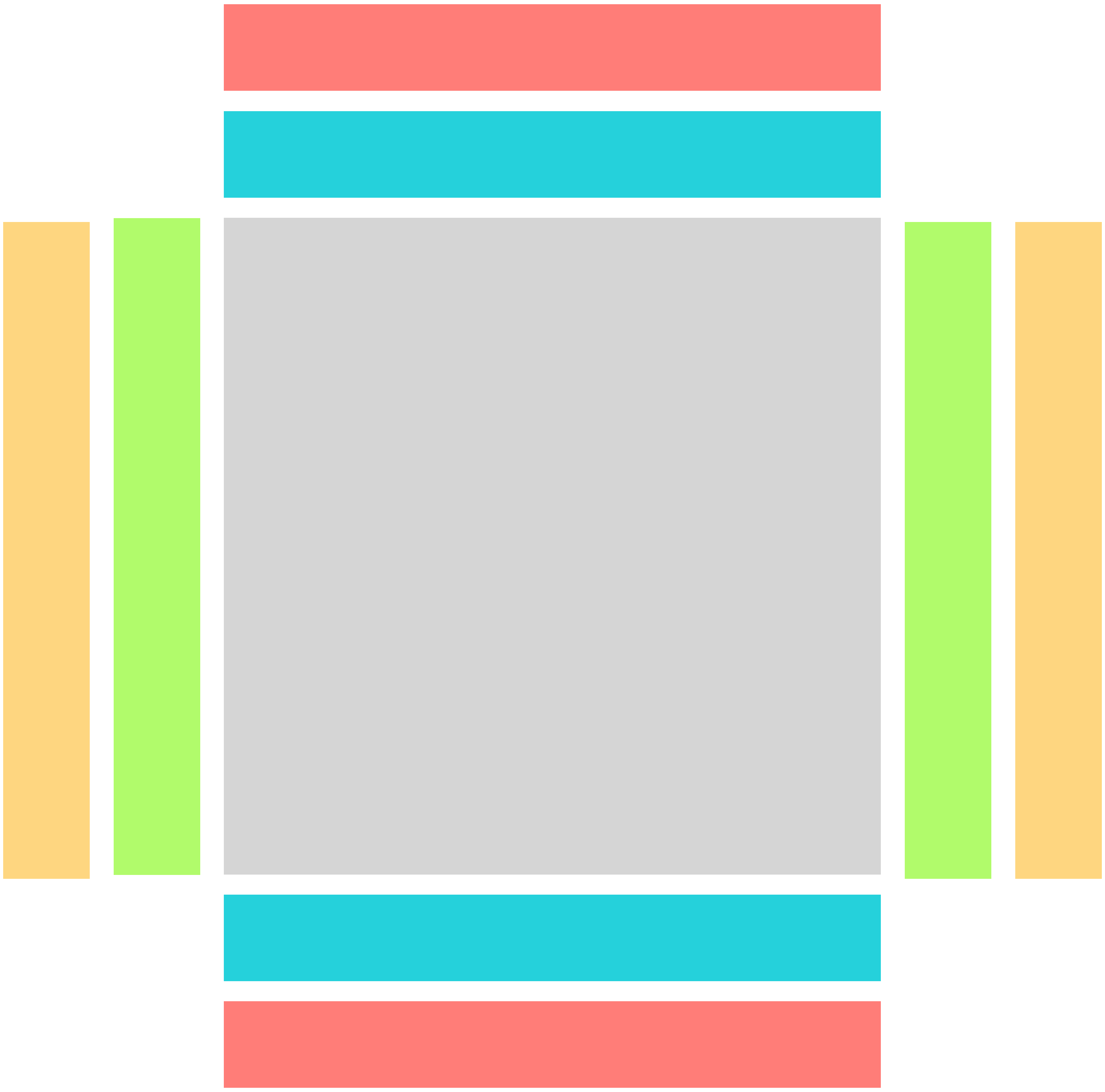
$$\frac{\partial \mathbf{Q}}{\partial t} = \frac{\partial \mathbf{F_E}}{\partial x} + \frac{\partial \mathbf{G_E}}{\partial y} + \frac{\partial \mathbf{H_E}}{\partial z} + \frac{\partial \mathbf{F_V}}{\partial x} + \frac{\partial \mathbf{G_V}}{\partial y} + \frac{\partial \mathbf{H_V}}{\partial z} + \ldots + \mathbf{\Omega}_{chem}$$



Dynamic scheduling

- The index space of a section of the PSAAP burner looks like a cross
- The lateral faces of the cross are connected to describe the domain topology
- HTR partitions this index space and solves a set of conservation equations within each partition

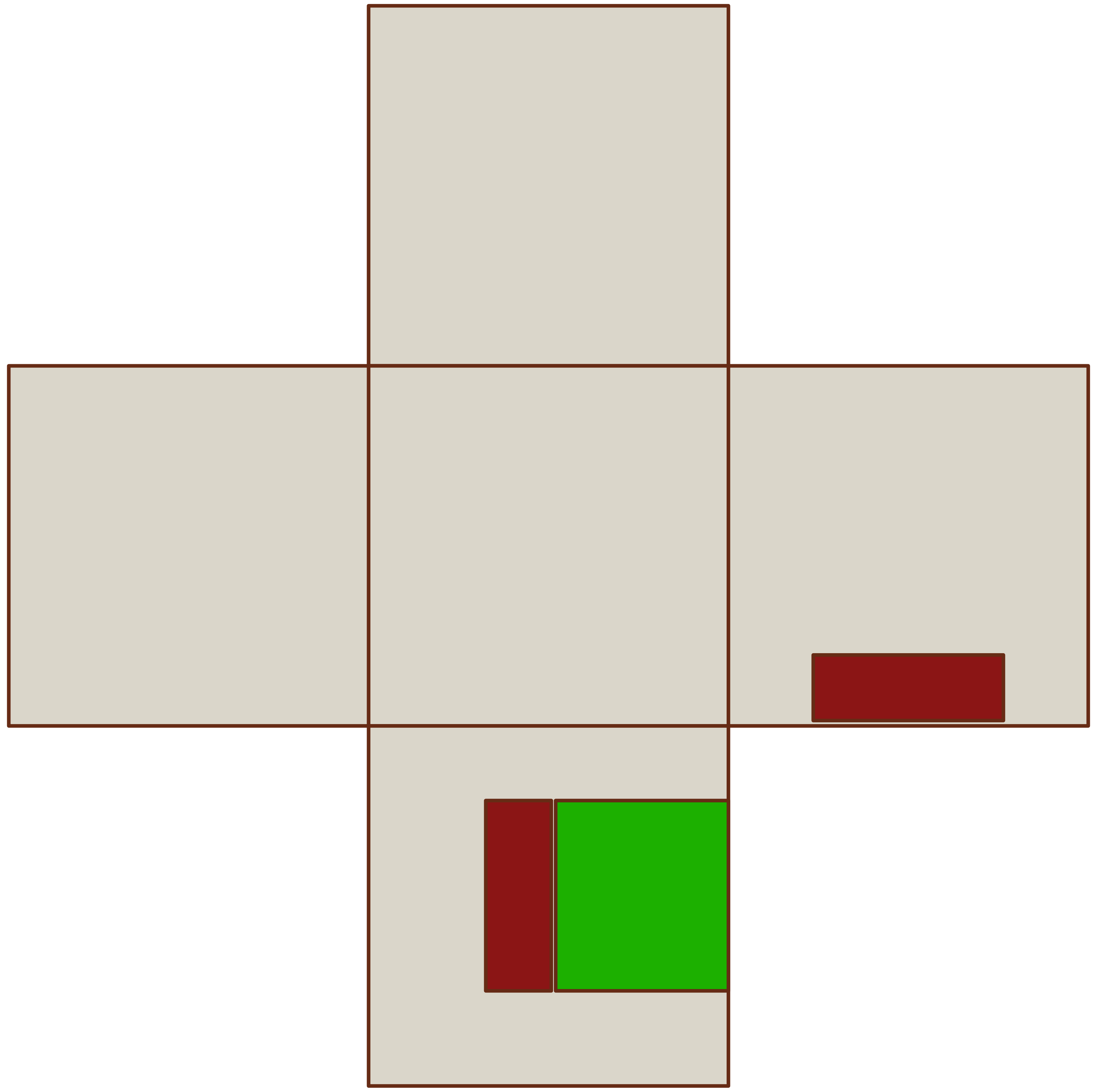
$$\frac{\partial \mathbf{Q}}{\partial t} = \frac{\partial \mathbf{F_E}}{\partial x} + \frac{\partial \mathbf{G_E}}{\partial y} + \frac{\partial \mathbf{H_E}}{\partial z} + \frac{\partial \mathbf{F_V}}{\partial x} + \frac{\partial \mathbf{G_V}}{\partial y} + \frac{\partial \mathbf{H_V}}{\partial z} + \ldots + \mathbf{\Omega}_{chem}$$



- Tasks compute and add to the right-hand side of the equation each term
- Required data might be readily available or not depending on partitioning and mapping
- Dynamically scheduling the tasks with an atomic coherence model significantly helps hiding communications

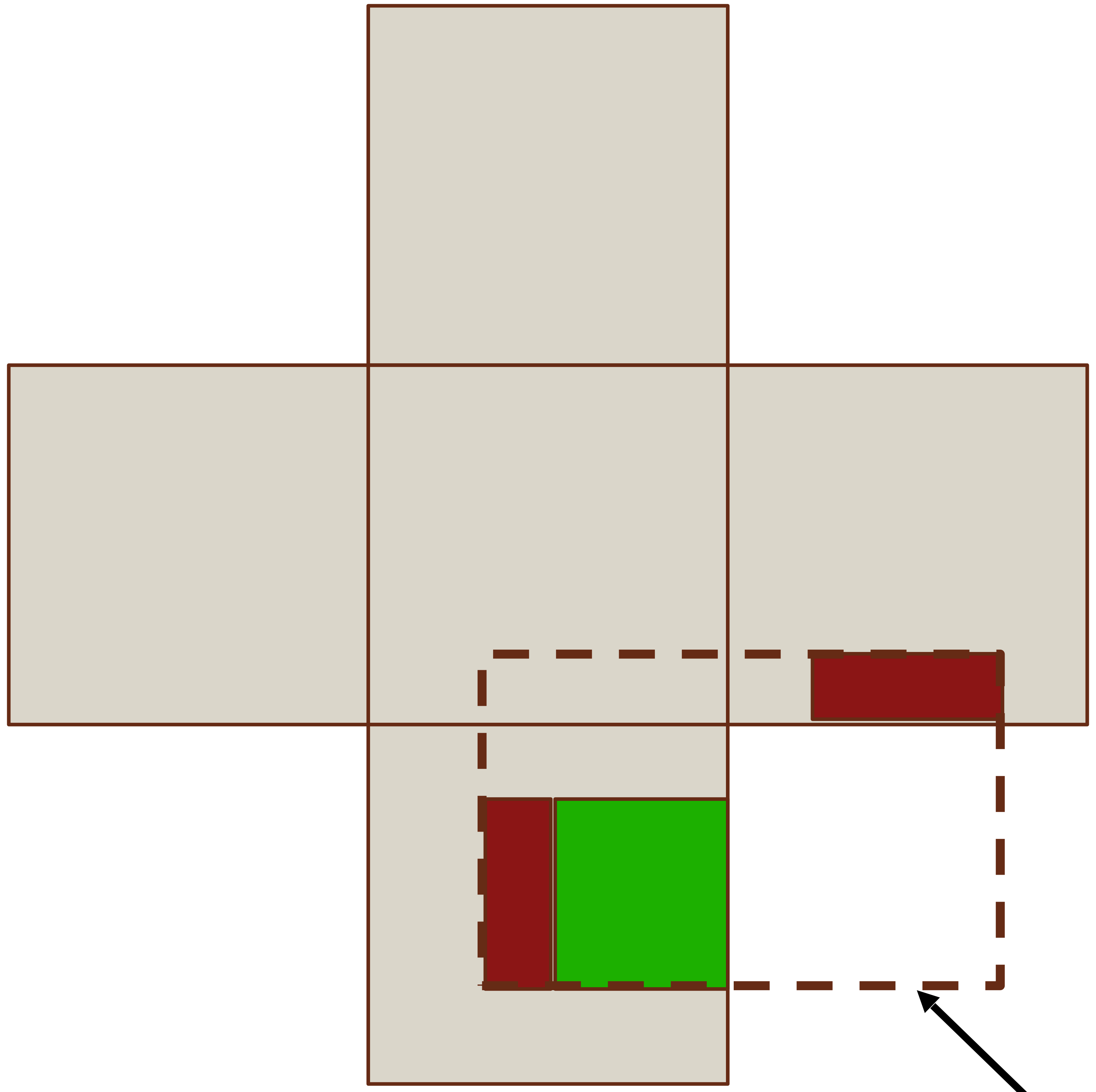
Instance padding

Problem: We need to execute a task on the green subregion. This task requires stencil accesses in the red subregions.



Instance padding

Problem: We need to execute a task on the green subregion. This task requires stencil accesses in the red subregions.



Physical instance allocated in memory

Baseline implementation:

- An affine accessor that spans the green and red regions is utilized

Pros:

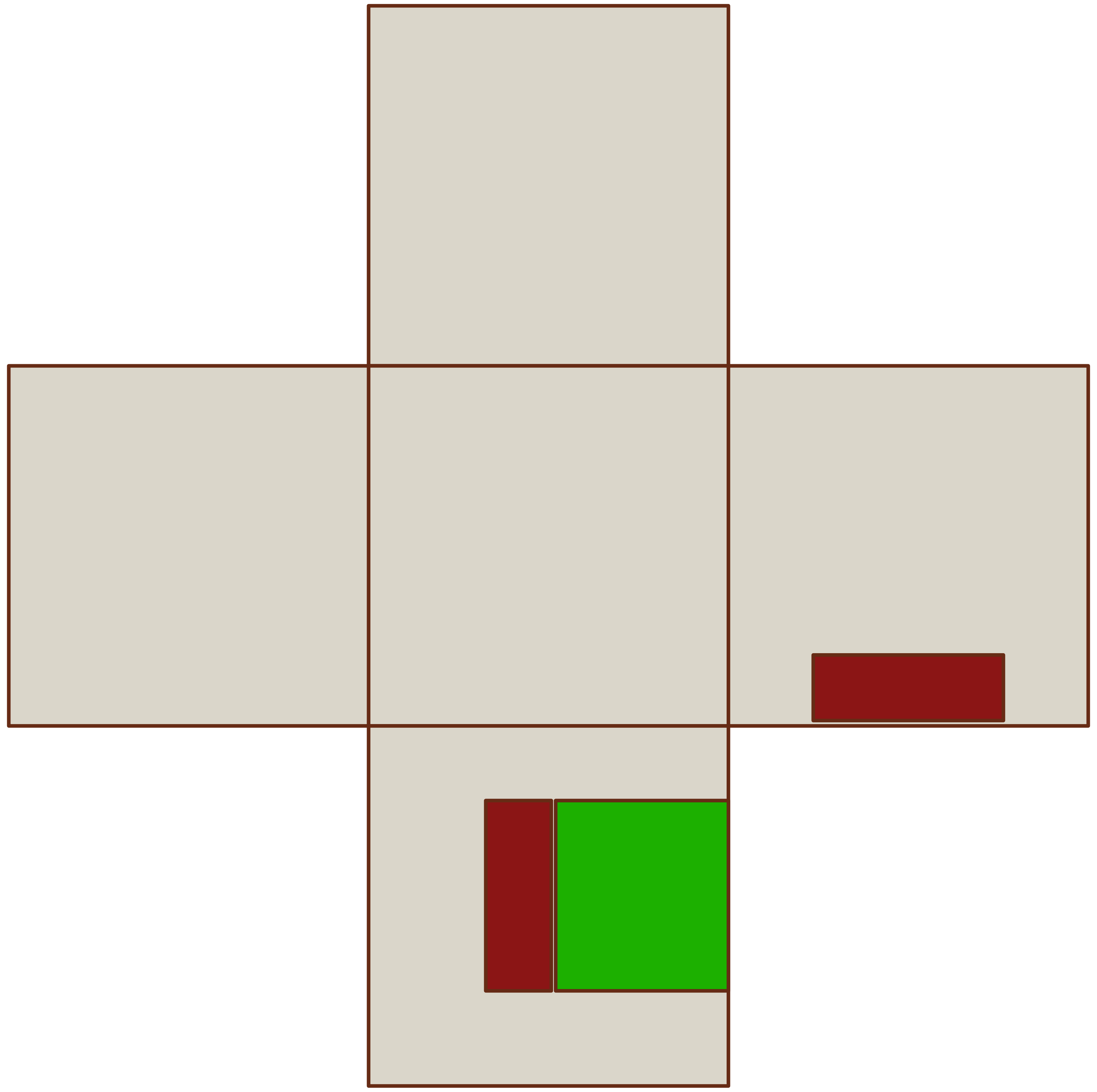
- Legion affine accessors are fast

Cons:

- Requires a lot of memory
- Computationally inefficient
- Non-trivial stencil point calculations

Instance padding

Problem: We need to execute a task on the green subregion. This task requires stencil accesses in the red subregions.

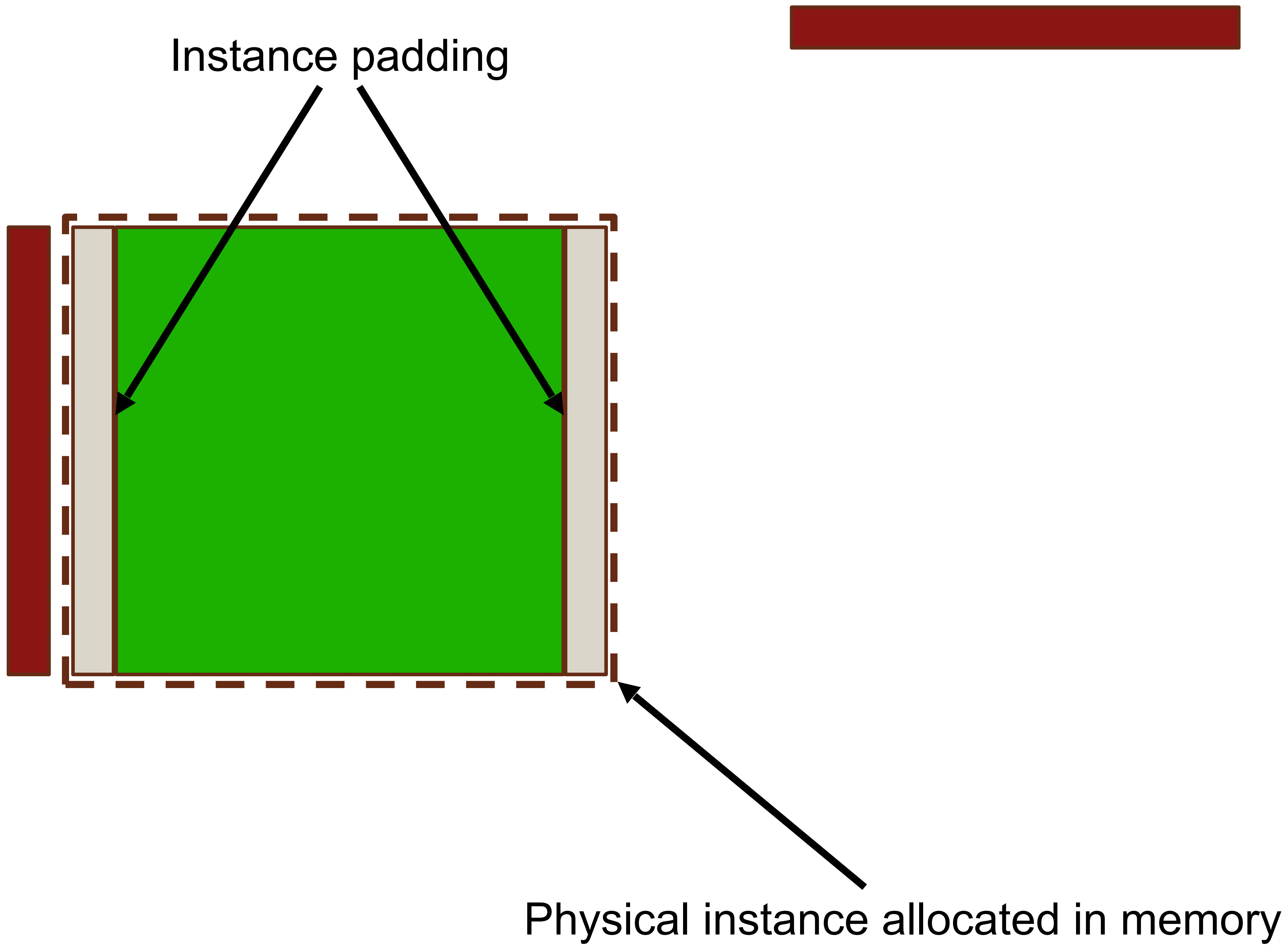


New implementation:

- A padded instance for the green region and a multi-affine accessor for the red regions are utilized.

Instance padding

Problem: We need to execute a task on the green subregion. This task requires stencil accesses in the red subregions.

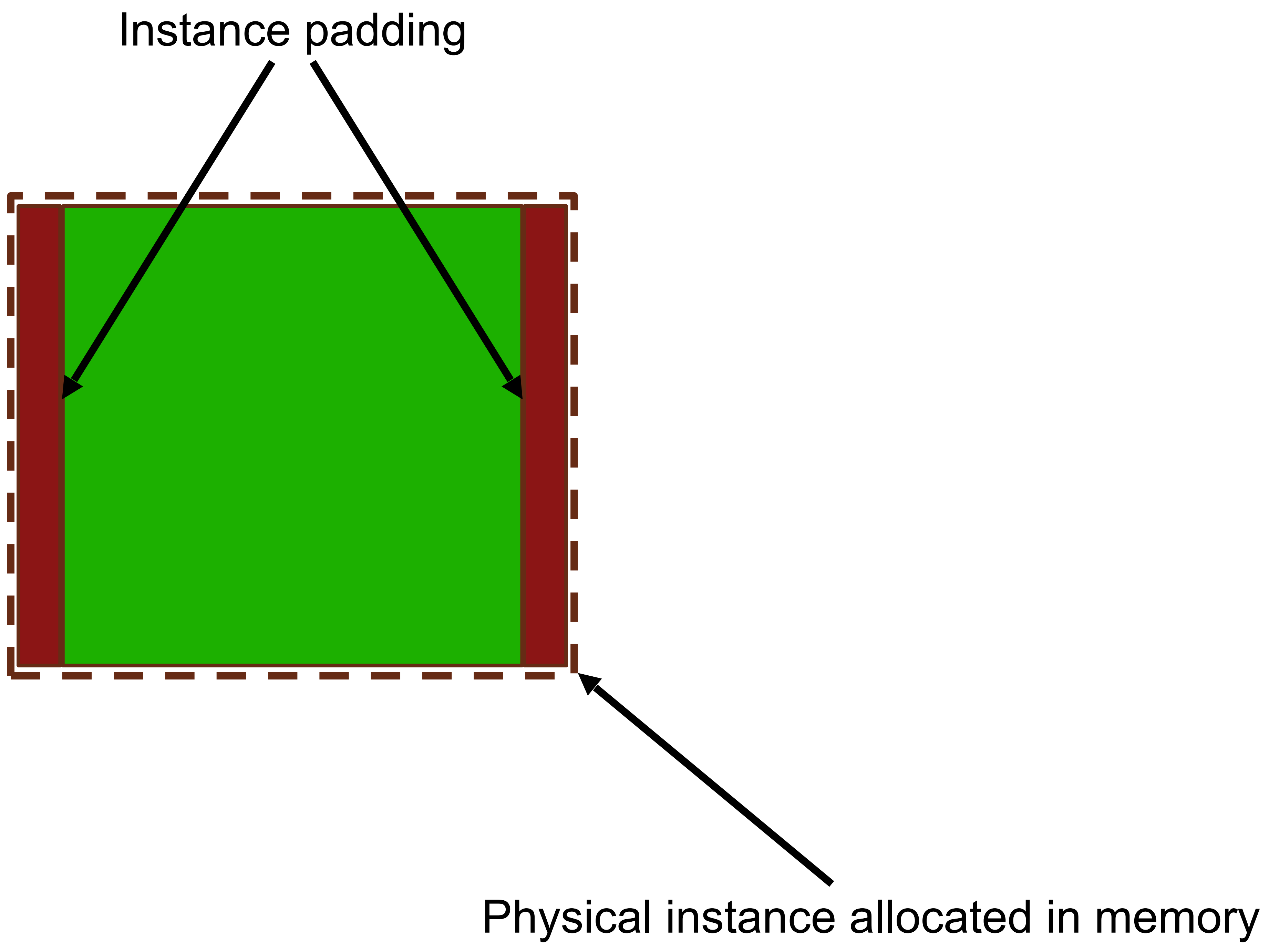


New implementation:

- A padded instance for the green region and a multi-affine accessor for the red regions are utilized.

Instance padding

Problem: We need to execute a task on the green subregion. This task requires stencil accesses in the red subregions.



New implementation:

- A padded instance for the green region and a multi-affine accessor for the red regions are utilized.
- Copy data from the red regions to the instance padding

Pros:

- Utilizes Legion affine accessors in the computationally intensive part of the calculation
- Simple stencil point calculations
- Improved affinity in memory access

The future of the HTR solver

Brief history of HTR

HTR 1.0

- Fully implemented in Regent
- Autoparallelized
- Could consider at most 5 species and a handful of reactions

HTR 1.2

- Manually parallelized
- Flux tasks ported to the C++ API to improve GPU kernel performance

HTR 1.3

- Chemistry tasks ported to the C++ API
- Tested with more than 40 species and 1000 reactions

HTR 1.4

- Most of the leaf tasks implemented in the C++ API
- Stable curvilinear coordinates formulation

HTR++

- The entire solver ported to the C++ API
- New portability layer

HTR++/Legion interface

Observations:

- HTR and HTR++ require only a small subset of the Legion runtime objects and execution options
- Most of HTR developers have a background in fluid dynamics

Objective: Hide some of the Legion runtime constructs and procedures from new developers while reducing the lines of code required by each operation

Strategy: Use a series of helper objects to ensure that Legion data structures are properly constructed, utilized, and deleted by HTR++.

```
//-----------------------------------------------------------------------------
// Utility that helps managing field spaces
//-----------------------------------------------------------------------------
class FieldSpaceManager {
public:
   //--------------------------------
   // Constructors
   //--------------------------------
   inline FieldSpaceManager(Context& ctx_, Runtime* runtime_)
       : owner(false),
         ctx(ctx_),
         runtime(runtime_){};
```

Snippet of the object that manages a Legion field space

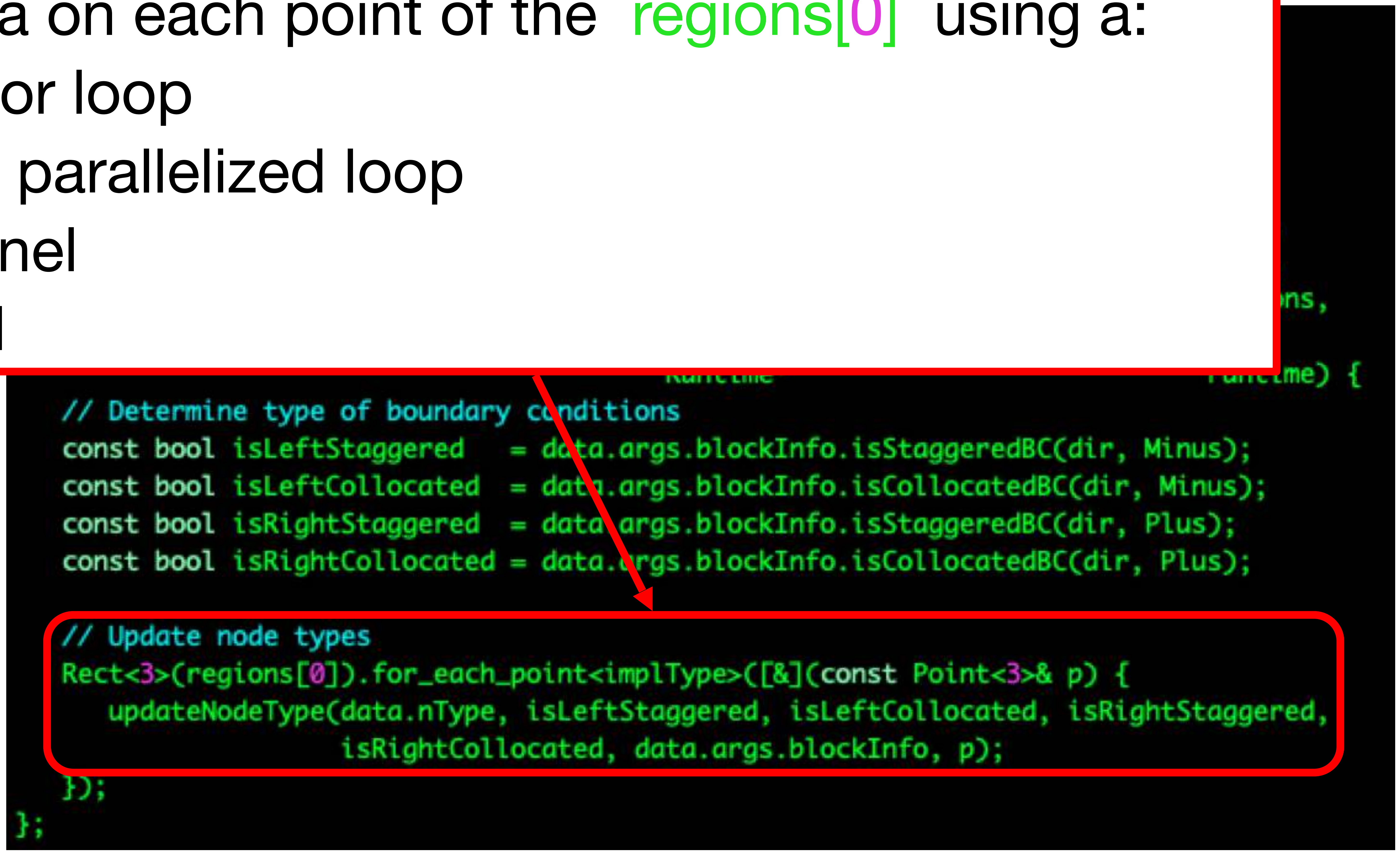
Byproduct: This interface could be generalized to allow HTR++ to be executed as a standalone application

HTR++ portability layer

t architectures

Depending on the value of implType , this function calls the provided lambda on each point of the regions[0] using a:

- A standard for loop
- An OpenMP parallelized loop
- A CUDA kernel
- A HIP kernel



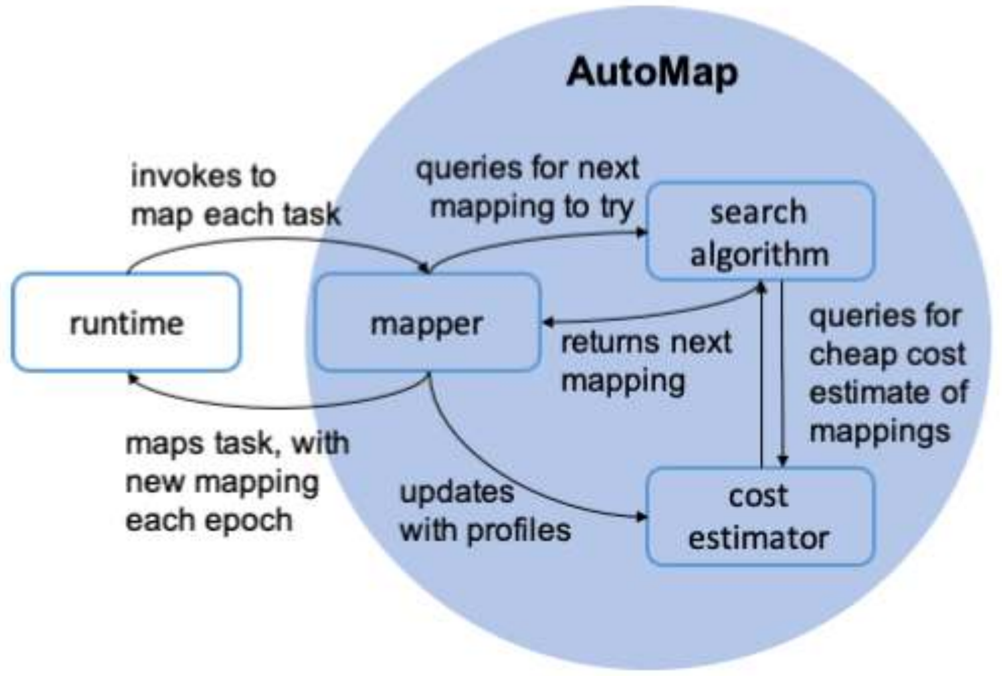
```
    // Determine type of boundary conditions
    const bool isLeftStaggered   = data.args.blockInfo.isStaggeredBC(dir, Minus);
    const bool isLeftCollocated  = data.args.blockInfo.isCollocatedBC(dir, Minus);
    const bool isRightStaggered  = data.args.blockInfo.isStaggeredBC(dir, Plus);
    const bool isRightCollocated = data.args.blockInfo.isCollocatedBC(dir, Plus);

    // Update node types
    Rect<3>(regions[0]).for_each_point<implType>([&](const Point<3>& p) {
       updateNodeType(data.nType, isLeftStaggered, isLeftCollocated, isRightStaggered,
                      isRightCollocated, data.args.blockInfo, p);
    });
};
```

Snippet of the implementation for the task initializes the grid operators for boundary regions

HTR++ portability layer — key features

- All the sources for each type of loop are implemented in the same place with clear advantages in terms of software management and interface design
- New models and features are only written in C++
- Porting to other architectures is very easy as it involves the inclusion of another kernel type in the `for_each_point` template function
- The implementation takes care of GPU reductions preserving the deferred execution model
- Developers can still implement their loops and OpenMP/CUDA/HIP kernels for debugging, performance, or exploration purposes

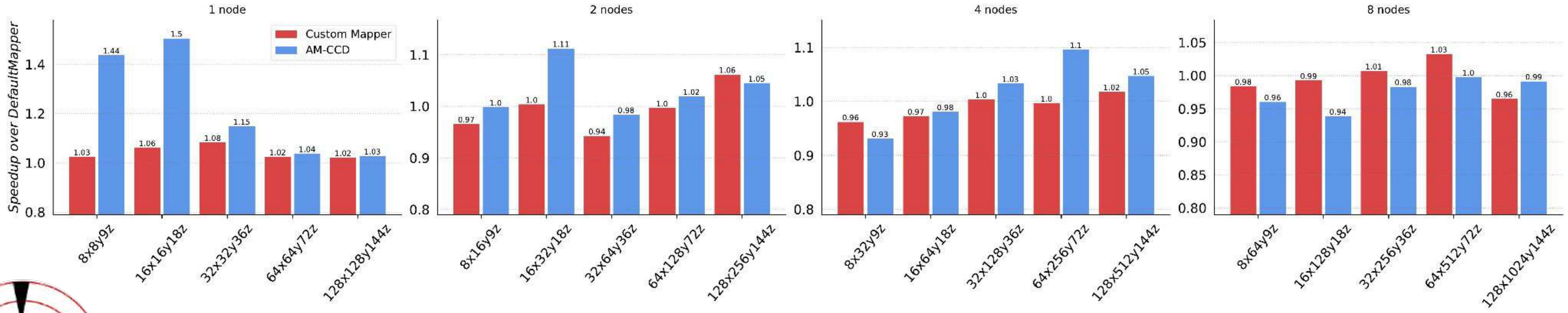
AutoMap



HTR has a sufficiently complex task graph that might have more than 2^{100} possible mappings

The HTR mapper using previous experience, measurement, or what we believe is the best execution algorithm

We are experimenting with the use of AutoMap to find better mappers



Credit: Thiago Teixeira (thiagoxt@stanford.edu)

Conclusions

- HTR is a compressible Navier—Stokes solver utilized for reacting flow simulations including combustion and hypersonics
- HTR has been natively written to be utilized with Legion and leverage the available task-based constructs
- HTR future developments involve a major porting to the C++ API and the use of new mappers found using AutoMap
- If you are interested or want to get involved in HTR, **please reach out!**