

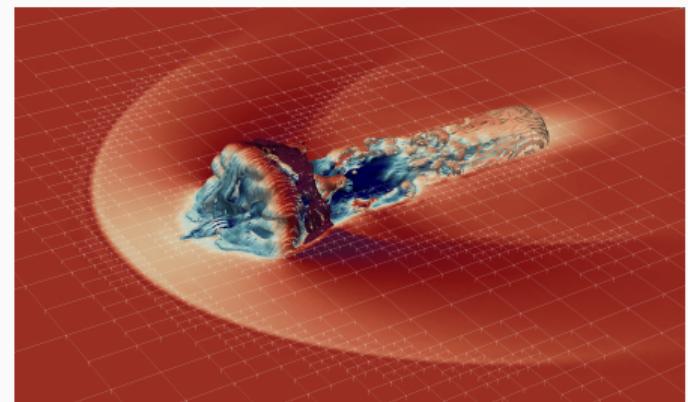
Adaptive and extendable numerical simulations with Trixi.jl

Michael Schlottke-Lakemper^{*}, Hendrik Ranocha[†]

JuliaCon 2021, 28th–30th July 2021

^{*} University of Cologne, Germany

[†] University of Münster, Germany



Main contributors



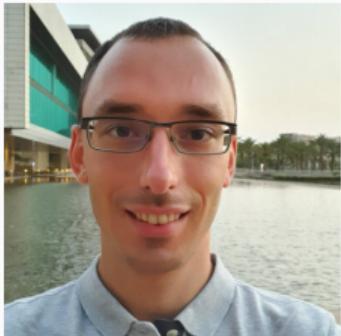
Michael Schlottke-Lakemper



Gregor J. Gassner



Andrew R. Winters



Hendrik Ranocha



Erik Faulhaber



Jesse Chan

How it got started (in January 2020)

- FLUXO, a fast, parallel Fortran 3D fluid dynamics code: the [race horse](#)



How it got started (in January 2020)

- FLUXO, a fast, parallel Fortran 3D fluid dynamics code: the [race horse](#)



- The [one-trick ponies](#): singular purpose, various languages, discard after use



What should we do? Write more code!

The new code should be

- Extendable
 - Useful for experimentation and research

What should we do? Write more code!

The new code should be

- Extendable
 - Useful for experimentation and research
- Easy to use
 - Low barrier for new users/students

What should we do? Write more code!

The new code should be

- Extendable
 - Useful for experimentation and research
- Easy to use
 - Low barrier for new users/students
- Fast
 - Suitable for laptop science and production simulations
 - High-performance implementation with adaptivity

What should we do? Write more code!

The new code should be

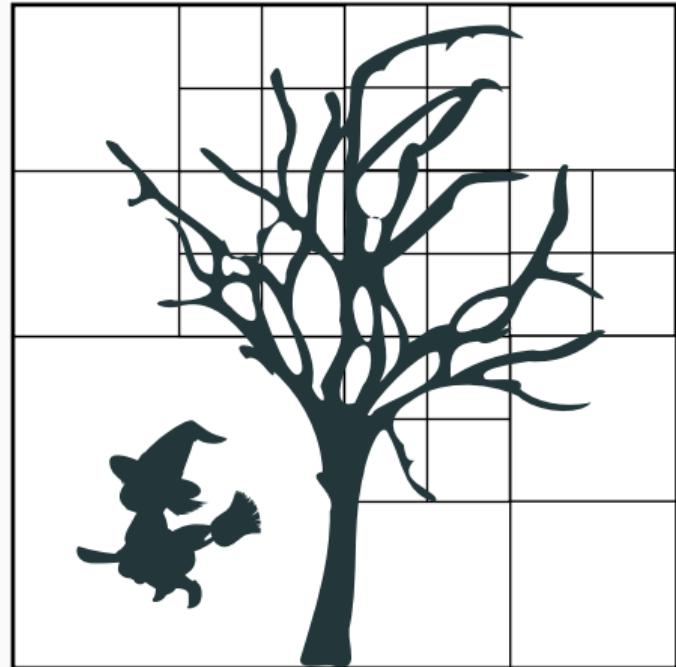
- Extendable
 - Useful for experimentation and research
- Easy to use
 - Low barrier for new users/students
- Fast
 - Suitable for laptop science and production simulations
 - High-performance implementation with adaptivity

Is this even achievable with a **single** code?

Where do we stand today?

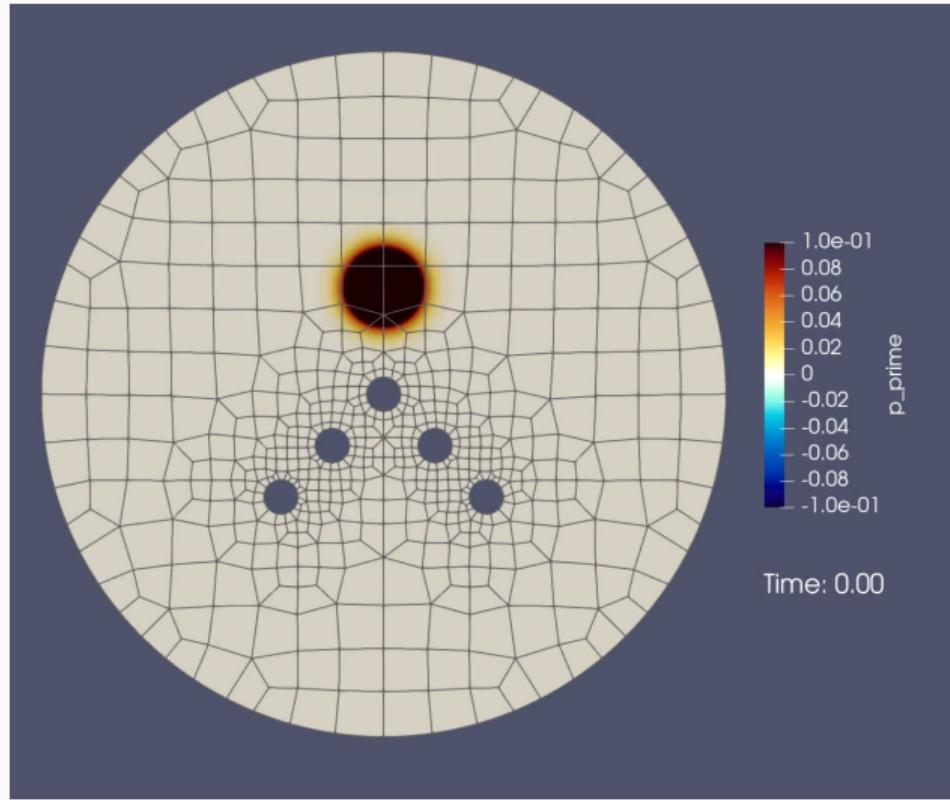
Trixi.jl

- Adaptive numerical simulation framework for hyperbolic partial differential equations
- 6 core developers, 12 contributors, 30k+ lines
- 18+ students, 2 published papers
- Integration with Julia ecosystem:
 - OrdinaryDiffEq.jl: time integration
 - ForwardDiff.jl: automatic differentiation
 - Plots.jl, Makie.jl: plotting
 - LoopVectorization.jl: performance
 - Polyester.jl: parallelization



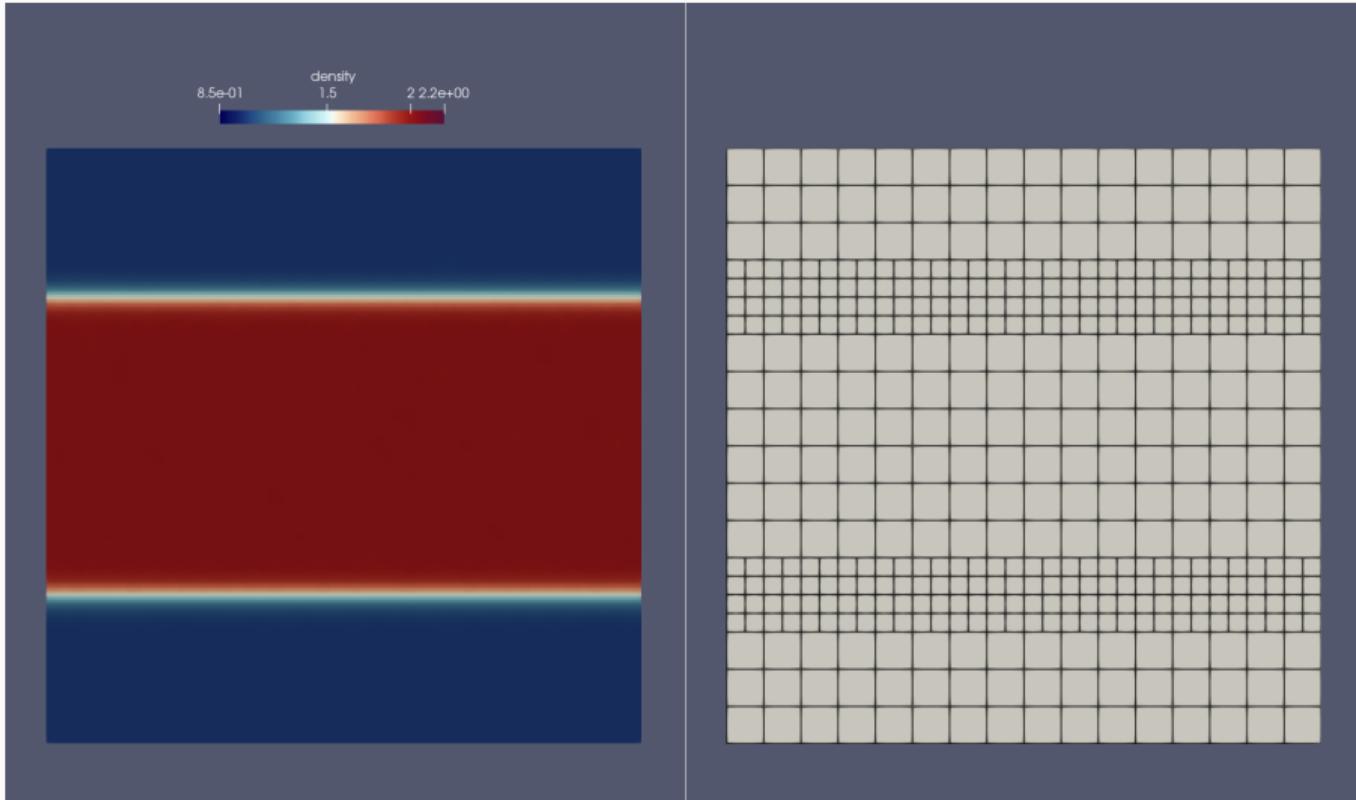
<https://github.com/trixi-framework/Trixi.jl>

Acoustic wave scattering on a curved unstructured mesh

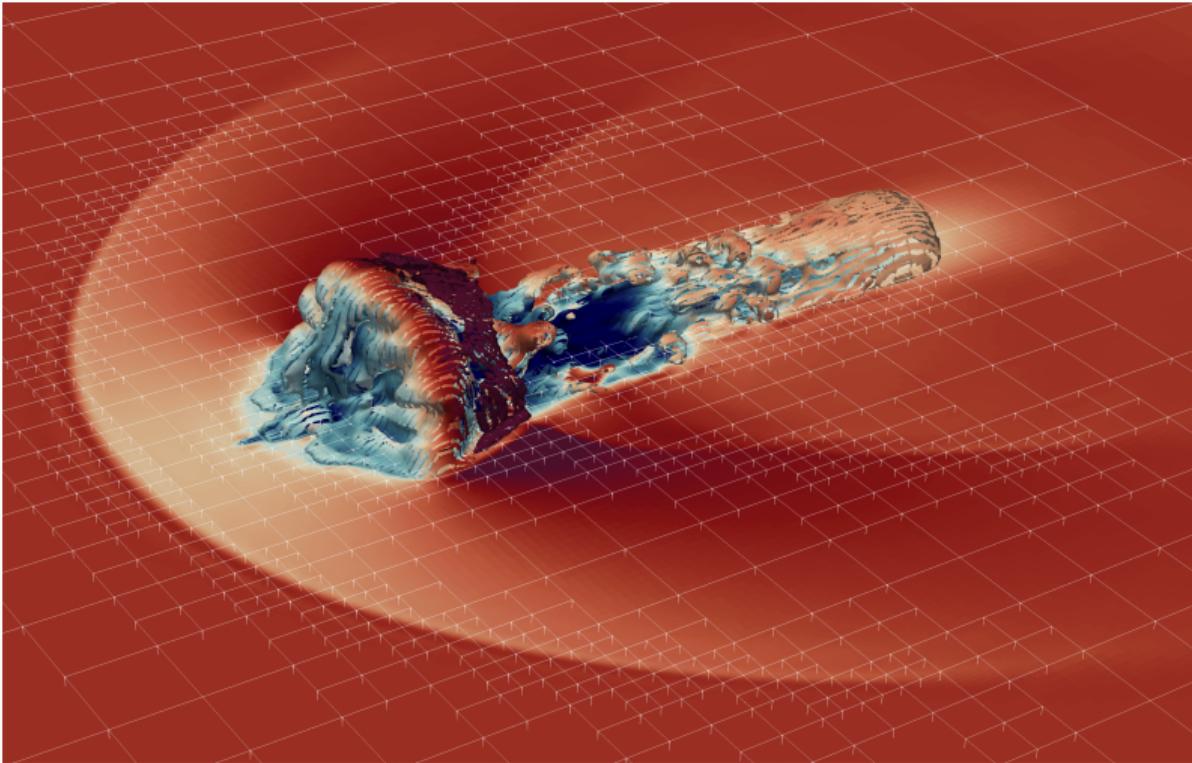


Credit: Andrew R. Winters, Lars Christmann

Kelvin-Helmholtz flow instability with adaptive mesh refinement



Cold gas cloud interacting with hot supersonic flow



Live demonstration

The screenshot shows a Jupyter Notebook interface with a presentation slide. The top menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar below the menu contains various icons for file operations like opening, saving, and running cells. The status bar indicates "Trusted" and "Julia 1.6.1". The main content area displays a slide with the title "Adaptive and extendable numerical simulations with Trixi.jl". The text describes Trixi.jl as a numerical simulation framework for adaptive, high-order discretizations of conservation laws, highlighting its modular architecture and ease of extension. It mentions the speaker's names, Michael Schlotte-Lakemper and Hendrik Ranocha, and provides links for follow-along and MyBinder access.

Adaptive and extendable numerical simulations with Trixi.jl

Trixi.jl is a numerical simulation framework for adaptive, high-order discretizations of conservation laws. It has a modular architecture that allows users to easily extend its functionality and was designed to be useful to experienced researchers and new users alike. In this talk, we will give an overview of Trixi's current features, present a typical workflow for creating and running a simulation, and show how to add new capabilities for your own research projects.

Michael Schlotte-Lakemper, Hendrik Ranocha

- Follow along at <https://git.io/Jcl6G>
- Launch MyBinder at <https://tinyurl.com/3zpp2ksw>

Jupyter notebook and presentation available at

<https://github.com/trixi-framework/talk-2021-juliacon>

(directly launch notebook: <https://tinyurl.com/3zpp2ksw>)

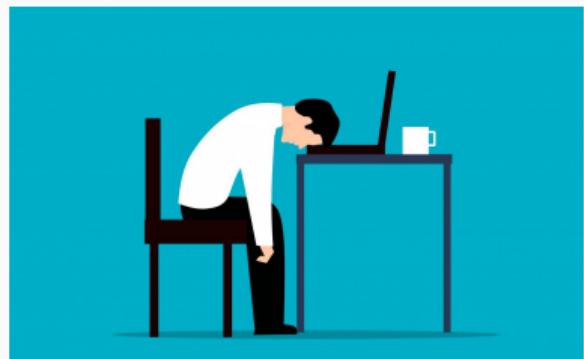
What about the “*look and feel*” performance?

- **Startup latency** is annoying (Julia v1.6.1/Trixi v0.3.40)
 - Install Trixi + OrdinaryDiffEq + Plots: 4 minutes
 - First result: 45 seconds
 - First plot: 20 seconds
 - Second result + plot: **0.04** seconds



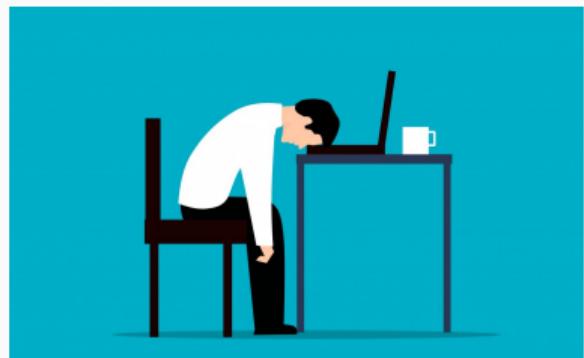
What about the “*look and feel*” performance?

- **Startup latency** is annoying (Julia v1.6.1/Trixi v0.3.40)
 - Install Trixi + OrdinaryDiffEq + Plots: 4 minutes
 - First result: 45 seconds
 - First plot: 20 seconds
 - Second result + plot: **0.04** seconds
- Reason: code compilation at first use → cache



What about the “*look and feel*” performance?

- **Startup latency** is annoying (Julia v1.6.1/Trixi v0.3.40)
 - Install Trixi + OrdinaryDiffEq + Plots: 4 minutes
 - First result: 45 seconds
 - First plot: 20 seconds
 - Second result + plot: **0.04** seconds
- Reason: code compilation at first use → cache
- **Upside:** REPL + Revise.jl takes away (most of) the pain



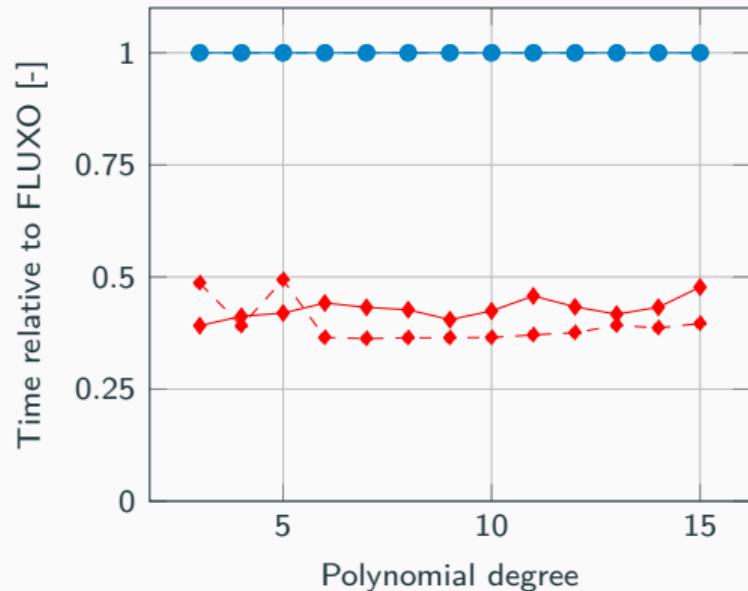
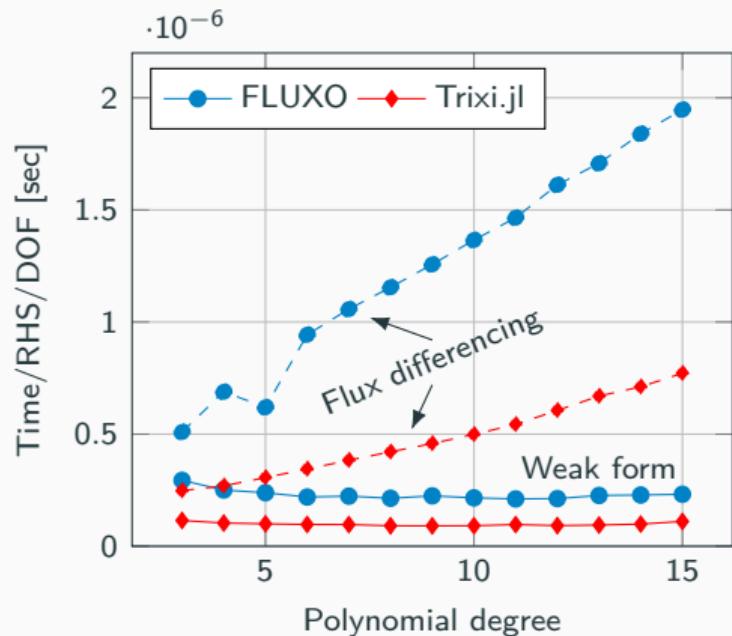
What about *writing fast code*?

- Efficient Julia code requires new performance intuition
 - Everything relevant for other compiled languages
 - + avoid unwanted allocations
 - + avoid type instabilities

What about *writing fast code*?

- Efficient Julia code requires new performance intuition
 - Everything relevant for other compiled languages
 - + avoid unwanted allocations
 - + avoid type instabilities
- **Upside:** many Julia-based tools for introspection and performance optimization

What about *serial* performance?



- Performance comparison with optimized Fortran-based FLUXO code (**lower is better**)
- Trixi: **faster** for **realistic example** (3D compressible Euler on curved mesh)

What about *parallel* performance for *simulation science*?

- Shared-memory parallelization **works well**
- Not many MPI-based massively parallel **simulation** codes out there (yet)
- Open question: **How to integrate MPI** without creating a “Frankencode”?
- New momentum from projects such as
<https://github.com/CliMA/ClimateMachine.jl>



Parallel performance: **the verdict is still out**



What about *ease of use* when installing Trixi?

- Built-in package manager `Pkg` handles everything
- Install Trixi + dependencies + postprocessing tools with `two lines of code`

```
julia> import Pkg  
julia> Pkg.add(["Trixi", "Trixi2Vtk", "OrdinaryDiffEq", "Plots"])
```

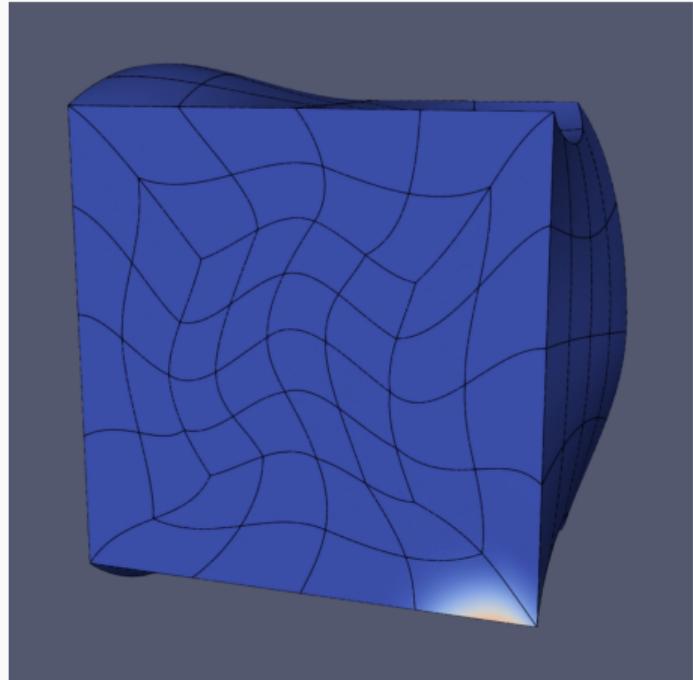
What about *binary dependencies*?

- Install [pre-compiled binaries](#) via Pkg
- Can be used for libraries and executables

Example: adaptive meshes with [p4est](#)

- Wrapper package [P4est.jl](#)
- Auto-installs OS-specific binaries
- Works on Linux, MacOS, Windows

<https://github.com/trixi-framework/P4est.jl>



Credit: Erik Faulhaber

What about *reproducibility*?

- Provision **reproducible** compute environments
- All information in **Project.toml** + **Manifest.toml**
- Excellent for **reproducible science**:
 - Paper #1: <https://git.io/JYBtP>
 - Paper #2: <https://git.io/JYBtA>
 - Talk #1: <https://git.io/JqnxE>
 - Talk #2: <https://git.io/JcLMy>
 - This talk: <https://git.io/JcL6G>

The screenshot shows a search results page on GitHub for the query "reproducible". The results are displayed in five cards, each representing a repository:

- A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics**
README.md | License MIT | DOI 10.5281/zenodo.3998635
- Preventing pressure oscillations does not fix local linear stability issues of entropy-based split-form high-order schemes**
README.md | License MIT | DOI 10.5281/zenodo.8554368
- Julia for adaptive high-order multi-physics simulations**
README.md | License MIT | Search Issues
- Introduction to Julia and Trixi, a numerical simulation framework for hyperbolic PDEs**
README.md | License MIT | Search Issues
- JuliaCon 2021: Adaptive and extendable numerical simulations with Trixi.jl**
README.md | License MIT | Search Issues

What about *extendability*?

- **Multiple dispatch** + Just-Ahead-Of-Time compilation
⇒ No difference between standard library code, package code, own code
- **Easily** extend existing functionality and **remain fast**

[Trixi.jl](#)

```
1 calc_volint(solver::DGSEM, volint_type::WeakForm, equations)
2 calc_volint(solver::DGSEM, volint_type::StrongForm, equations)
```

What about *extendability*?

- **Multiple dispatch** + Just-Ahead-Of-Time compilation
⇒ No difference between standard library code, package code, own code
- **Easily** extend existing functionality and **remain fast**

`Trixi.jl`

```
1 calc_volint(solver::DGSEM, volint_type::WeakForm, equations)
2 calc_volint(solver::DGSEM, volint_type::StrongForm, equations)
```

`User code`

```
3 calc_volint(solver::DGSEM, volint_type::StrongForm, equations::CompressibleEuler)
```

What about *extendability*?

- Flow-gravity simulation with **shock capturing** and **adaptive mesh refinement**
- Multi-physics coupling with **less than 350 lines** of code

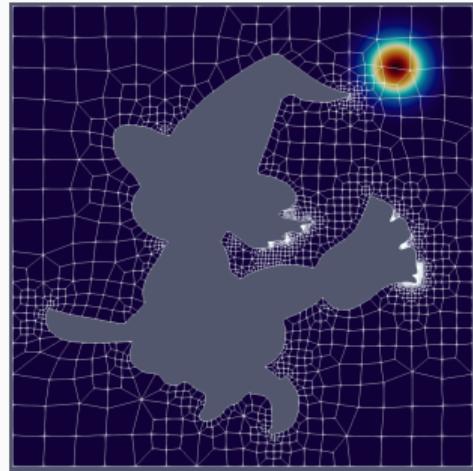


Schlottke-Lakemper, Winters, Ranocha, Gassner, Journal of Computational Physics, 2021.

[doi:10.1016/j.jcp.2021.110467](https://doi.org/10.1016/j.jcp.2021.110467) | arXiv:2008.10593

Conclusions and outlook

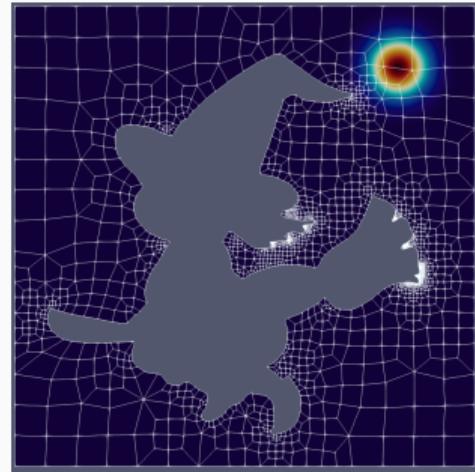
- [Trixi.jl](#): adaptive high-order numerical simulations
- Focus on [extendability](#), [ease of use](#), [performance](#)
- Julia [well-suited](#) for simulation science
(with small caveats)
- Future: [MPI parallelization](#), more physics, GPUs



Credit: Andrew R. Winters

Conclusions and outlook

- **Trixi.jl**: adaptive high-order numerical simulations
- Focus on **extendability, ease of use, performance**
- Julia **well-suited** for simulation science
(with small caveats)
- Future: **MPI parallelization**, more physics, GPUs



Credit: Andrew R. Winters

Thank you for your interest!

Get in touch: join us on Trixi's Slack or open an issue!

<https://github.com/trixi-framework/Trixi.jl>