# Non-intrusive Multirate Time-Integration for High-Order accurate Compressible Fluid Dynamics with Trixi.jl

**Daniel Doehring**[1]

in collaboration with

Michael Schlottke-Lakemper[2], Gregor Gassner[3], and Manuel Torrilhon[1]

[1] **ACoM, RWTH Aachen University** [2] HPSC, University of Augsburg [3] CDS, University of Cologne

2024/07/02

Applied and Computational Mathematics | RWTH AACHEN UNIVERSITY

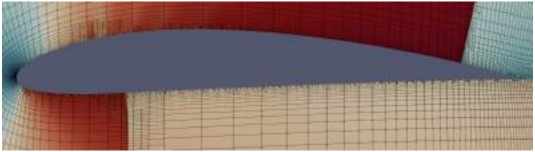
# Outline

1. About Trixi.jl
2. Multirate Time-Integration with P-ERK
3. Applications
4. Conclusion & Outlook

# About Trixi.jl

**Trixi.jl** → `https://github.com/trixi-framework/Trixi.jl`

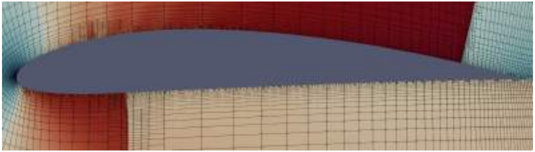
- is a Julia package for simulation of compressible flows
  → Euler, NSF, (VR)MHD, APE



# About Trixi.jl

**Trixi.jl** → https://github.com/trixi-framework/Trixi.jl

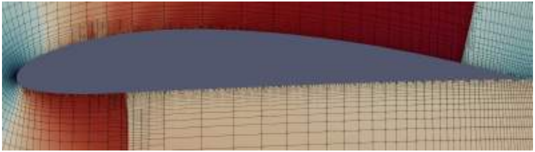
- is a Julia package for simulation of compressible flows
  → Euler, NSF, (VR)MHD, APE
- implements the DGSEM method



# About Trixi.jl

**Trixi.jl** → https://github.com/trixi-framework/Trixi.jl

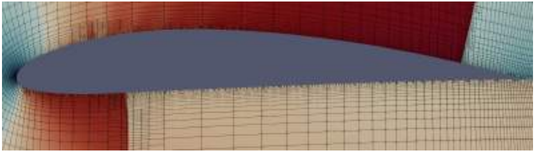
- is a Julia package for simulation of compressible flows
  → Euler, NSF, (VR)MHD, APE
- implements the DGSEM method
- is *h*-adaptive (AMR) through `p4est` and `t8code`



# About Trixi.jl

**Trixi.jl** → `https://github.com/trixi-framework/Trixi.jl`

- is a Julia package for simulation of compressible flows
  → Euler, NSF, (VR)MHD, APE
- implements the DGSEM method
- is *h*-adaptive (AMR) through `p4est` and `t8code`
- discretizes PDEs via the **method-of-lines (MoL)** approach



# About Trixi.jl

## Method-of-Lines approach

- PDE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i\big(\mathbf{u}, \nabla \mathbf{u}\big) = \mathbf{0}$$

```julia
using Trixi # Trixi.jl is reg. Julia package

# Define PDE
equations = LinearScalarAdvectionEquation1D(1.0)
```

*Actual runnable Julia code snippet*

# About Trixi.jl

## Method-of-Lines approach

- PDE → ODE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i\big(\mathbf{u}, \nabla \mathbf{u}\big) = \mathbf{0}$$

$$\overset{\text{DG}}{\Rightarrow} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{U}(t) = \mathbf{F}\big(\mathbf{U}(t)\big)$$

```
using Trixi # Trixi.jl is reg. Julia package

# Define PDE
equations = LinearScalarAdvectionEquation1D(1.0)


# Vanilla DGSEM with k = 3, Rusanov/LLF flux
solver = DGSEM(polydeg = 3,
               surface_flux = flux_lax_friedrichs)
# 16-cell discretization of Ω = (−1, 1)
mesh = StructuredMesh((16,), (-1,), (1,))
# Some initial condition function
initial_condition = initial_condition_const

# Perform MoL/Semidiscretization
semi = SemidiscretizationHyperbolic(mesh,
         equations,
         initial_condition, solver)

# Set up ODE problem
tspan = (0, 1)
ode = semidiscretize(semi, tspan)
```

*Actual runnable Julia code snippet*

# About Trixi.jl

## Method-of-Lines approach

- PDE → ODE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i\big(\mathbf{u}, \nabla \mathbf{u}\big) = \mathbf{0}$$

$$\overset{\text{DG}}{\Rightarrow} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{U}(t) = \mathbf{F}\big(\mathbf{U}(t)\big)$$

a) Hand-off ODE problem to `OrdinaryDiffEq.jl`

```julia
using Trixi # Trixi.jl is reg. Julia package

# Define PDE
equations = LinearScalarAdvectionEquation1D(1.0)


# Vanilla DGSEM with k = 3, Rusanov/LLF flux
solver = DGSEM(polydeg = 3,
               surface_flux = flux_lax_friedrichs)
# 16-cell discretization of Ω = (−1, 1)
mesh = StructuredMesh((16,), (-1,), (1,))
# Some initial condition function
initial_condition = initial_condition_const

# Perform MoL/Semidiscretization
semi = SemidiscretizationHyperbolic(mesh,
          equations,
          initial_condition, solver)

# Set up ODE problem
tspan = (0, 1)
ode = semidiscretize(semi, tspan)
```

*Actual runnable Julia code snippet*

# About Trixi.jl

## Method-of-Lines approach

- PDE $\rightarrow$ ODE

$$\partial_t \mathbf{u}(t, \mathbf{x}) + \partial_{x_i} \mathbf{f}_i\big(\mathbf{u}, \nabla \mathbf{u}\big) = \mathbf{0}$$

$$\overset{\text{DG}}{\Rightarrow} \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{U}(t) = \mathbf{F}\big(\mathbf{U}(t)\big)$$

a) Hand-off ODE problem to `OrdinaryDiffEq.jl`

b) Multirate Time-Integration via partitioned Runge-Kutta

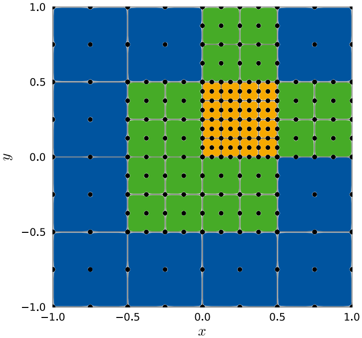
```julia
using Trixi # Trixi.jl is reg. Julia package

# Define PDE
equations = LinearScalarAdvectionEquation1D(1.0)


# Vanilla DGSEM with k = 3, Rusanov/LLF flux
solver = DGSEM(polydeg = 3,
               surface_flux = flux_lax_friedrichs)
# 16-cell discretization of Ω = (−1, 1)
mesh = StructuredMesh((16,), (-1,), (1,))
# Some initial condition function
initial_condition = initial_condition_const

# Perform MoL/Semidiscretization
semi = SemidiscretizationHyperbolic(mesh,
         equations,
         initial_condition, solver)

# Set up ODE problem
tspan = (0, 1)
ode = semidiscretize(semi, tspan)
```

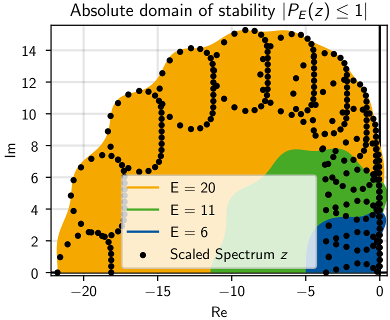
*Actual runnable Julia code snippet*

# Contents

# Stabilized Multirate Time-Integration

Achieve multirate time-integration by using **stabilized/optimized** schemes in regions with higher characteristic speeds (stricter CFL)




For convection dominated problems:

$$\Delta t \overset{!}{\leq} C_t(E) \cdot C_x(k) \min_i \underbrace{\min_j \frac{\Delta x_i}{|\mu_j(\mathbf{u}_i)|}}_{=:a_i}$$

# Stabilized Multirate Time-Integration

**Partition** ODE according to characteristic speeds $a_i$

$$\mathbf{U}'(t) = \begin{pmatrix} \mathbf{U}^{(1)}(t) \\ \vdots \\ \mathbf{U}^{(R)}(t) \end{pmatrix}' = \begin{pmatrix} \mathbf{F}^{(1)}\big(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)\big) \\ \vdots \\ \mathbf{F}^{(R)}\big(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)\big) \end{pmatrix} = \mathbf{F}\big(t, \mathbf{U}(t)\big)$$

# Stabilized Multirate Time-Integration

**Partition** ODE according to characteristic speeds $a_i$

$$\mathbf{U}'(t) = \begin{pmatrix} \mathbf{U}^{(1)}(t) \\ \vdots \\ \mathbf{U}^{(R)}(t) \end{pmatrix}' = \begin{pmatrix} \mathbf{F}^{(1)}\big(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)\big) \\ \vdots \\ \mathbf{F}^{(R)}\big(t, \mathbf{U}^{(1)}(t), \dots, \mathbf{U}^{(R)}(t)\big) \end{pmatrix} = \mathbf{F}\big(t, \mathbf{U}(t)\big)$$

and employ different RKMs

$$\mathbf{K}_i^{(r)} = \mathbf{F}^{(r)}\left(t_n + c_i^{(r)} \Delta t, \mathbf{U}_n + \Delta t \sum_{j=1}^{S} \sum_{k=1}^{R} a_{i,j}^{(k)} \mathbf{K}_j^{(k)}\right),$$

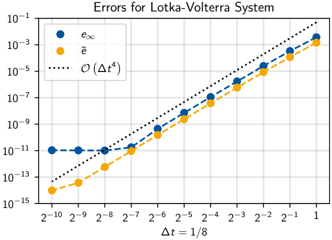
$$\mathbf{U}_{n+1} = \mathbf{U}_n + \Delta t \sum_{i=1}^{S} \sum_{k=1}^{R} b_i^{(r)} \mathbf{K}_i^{(r)}$$

optimized for the spectrum $\sigma\big(J(\mathbf{U}_0)\big) = \sigma\left(\frac{\partial \mathbf{F}}{\partial \mathbf{U}}\Big|_{\mathbf{U}_0}\right)$.

# Multirate Time-Integration with P-ERK

**Paired-Explicit Runge-Kutta (P-ERK) methods**[1] are a class of partitioned RKMs that satisfy

- Order conditions: $p = 2$[1], $p = 3$[2] or $p = 4$[3]




[1] Vermeire. JCP. 2019

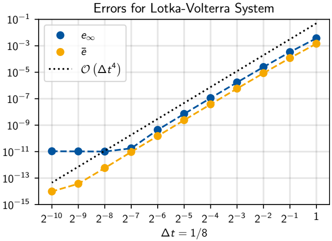
[2] Nasab, Vermeire. JCP. 2022

[3] Doehring et al. To Appear. 2024

[4] Hundsdorfer, Ketcheson, Savostianov. J. Sci. Comput. 2015

# Multirate Time-Integration with P-ERK

**Paired-Explicit Runge-Kutta (P-ERK) methods**[1] are a class of partitioned RKMs that satisfy




- Order conditions: $p = 2$[1], $p = 3$[2] or $p = 4$[3]
- Conservation[4]

| | P-ERK$_{4;\{6,10,16\}}$ | SSP$_{4;10}$ |
|---|---|---|
| $e_\rho^{\text{Cons}}$ | $3.78 \cdot 10^{-13}$ | $2.11 \cdot 10^{-13}$ |
| $e_{\rho v_x}^{\text{Cons}}$ | $5.70 \cdot 10^{-14}$ | $1.26 \cdot 10^{-13}$ |
| $e_{\rho v_y}^{\text{Cons}}$ | $6.53 \cdot 10^{-14}$ | $1.29 \cdot 10^{-13}$ |
| $e_{\rho e}^{\text{Cons}}$ | $9.40 \cdot 10^{-13}$ | $3.66 \cdot 10^{-12}$ |

[1] Vermeire. JCP. 2019

[2] Nasab, Vermeire. JCP. 2022

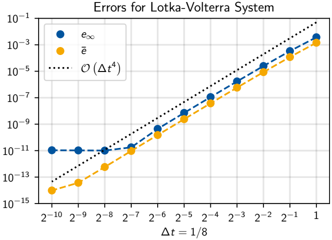
[3] Doehring et al. To Appear. 2024

[4] Hundsdorfer, Ketcheson, Savostianov. J. Sci. Comput. 2015

# Multirate Time-Integration with P-ERK

**Paired-Explicit Runge-Kutta (P-ERK) methods**[1] are a class of partitioned RKMs that satisfy

- Order conditions: $p = 2$[1], $p = 3$[2] or $p = 4$[3]
- Conservation[4]
- Internal Consistency[4]




| | P-ERK$_{4;\{6,10,16\}}$ | SSP$_{4;10}$ |
|---|---|---|
| $e_\rho^{\text{Cons}}$ | $3.78 \cdot 10^{-13}$ | $2.11 \cdot 10^{-13}$ |
| $e_{\rho v_x}^{\text{Cons}}$ | $5.70 \cdot 10^{-14}$ | $1.26 \cdot 10^{-13}$ |
| $e_{\rho v_y}^{\text{Cons}}$ | $6.53 \cdot 10^{-14}$ | $1.29 \cdot 10^{-13}$ |
| $e_{\rho e}^{\text{Cons}}$ | $9.40 \cdot 10^{-13}$ | $3.66 \cdot 10^{-12}$ |

| Method | $\|e_\rho(x,y)\|_{L^\infty(\Omega)}$ |
|---|---|
| P-ERK$_{4;6}$ | $1.1497 \cdot 10^{-5}$ |
| P-ERK$_{4;10}$ | $1.1497 \cdot 10^{-5}$ |
| P-ERK$_{4;16}$ | $1.1497 \cdot 10^{-5}$ |
| P-ERK$_{4;\{6,10,16\}}$ | $1.1499 \cdot 10^{-5}$ |

[1] Vermeire. JCP. 2019

[2] Nasab, Vermeire. JCP. 2022

[3] Doehring et al. To Appear. 2024

[4] Hundsdorfer, Ketcheson, Savostianov. J. Sci. Comput. 2015

# Multirate Time-Integration with P-ERK

## Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- `Elements`

# Multitrate Time-Integration with P-ERK

## Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- `Elements`
- `Interfaces`

# Multirate Time-Integration with P-ERK

## Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- `Elements`
- `Interfaces`
- `Mortars`

# Multitrate Time-Integration with P-ERK

## Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- `Elements`
- `Interfaces`
- `Mortars`
- `Boundaries`

# Multirate Time-Integration with P-ERK

## Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- `Elements`
- `Interfaces`
- `Mortars`
- `Boundaries`

$\Rightarrow$ Assign these to partitions $r = 1, \ldots, R$

# Multirate Time-Integration with P-ERK

## Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- `Elements`
- `Interfaces`
- `Mortars`
- `Boundaries`

⇒ Assign these to partitions $r = 1, \ldots, R$

```julia
function calc_volume_integral!(du, u, mesh, nonconservative_terms, equations,
                               volume_integral::VolumeIntegralWeakForm,
                               dg::DGSEM, cache)
  # Loop over all elements in the cache
  @threaded for element in eachelement(dg, cache)
    weak_form_kernel!(du, u, element, mesh, nonconservative_terms,
                      equations, dg, cache)
  end
  return nothing
end
```

# Multirate Time-Integration with P-ERK

## Non-intrusiveness

`Trixi.jl` operates on four (convenience) data structures

- `Elements`
- `Interfaces`
- `Mortars`
- `Boundaries`

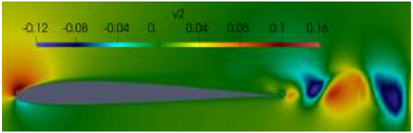
$\Rightarrow$ Assign these to partitions $r = 1, \ldots, R$

```
function calc_volume_integral!(du, u, mesh, nonconservative_terms, equations,
                               volume_integral::VolumeIntegralWeakForm,
                               dg::DGSEM, cache, elements_r::Vector{Int64})
  # Loop over elements of the r'th level
  @threaded for element in elements_r
    weak_form_kernel!(du, u, element, mesh, nonconservative_terms,
                      equations, dg, cache)
  end
  return nothing
end
```

# Contents

# Application: Laminar flow around 2D SD7003 airfoil

- $\text{Re} = 10^4, \text{Ma} = 0.2$
- $k = 3, p = 4, 7605$ quads
- $\Omega = [-20, 40] \times [-20, 20]$




| Source | $\overline{C}_L$ | $\overline{C}_D$ |
|---|---|---|
| P-ERK $p = 4$ | 0.3827 | 0.04995 |
| P-ERK $p = 2$[a] | 0.3841 | 0.04990 |
| P-ERK $p = 3$[b] | 0.3848 | 0.04910 |
| Uranga et al.[c] | 0.3755 | 0.04978 |
| López-Morales et al.[d] | 0.3719 | 0.04940 |

[a] Vermeire. JCP. 2019

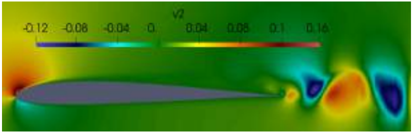
[b] Nasab, Vermeire. JCP. 2022

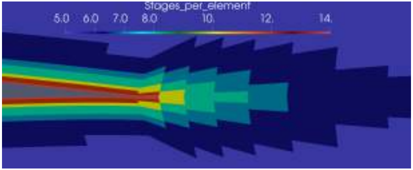
[c] Uranga et al. Int. J. Num Methods Eng. 2011

[d] López-Morales et al. 32nd AIAA AAC. 2014

# Application: Laminar flow around 2D SD7003 airfoil

- $\mathrm{Re} = 10^4, \mathrm{Ma} = 0.2$
- $k = 3, p = 4$, 7605 quads
- $\Omega = [-20, 40] \times [-20, 20]$




| Source | $\overline{C}_L$ | $\overline{C}_D$ |
|---|---|---|
| P-ERK $p = 4$ | 0.3827 | 0.04995 |
| P-ERK $p = 2$[a] | 0.3841 | 0.04990 |
| P-ERK $p = 3$[b] | 0.3848 | 0.04910 |
| Uranga et al.[c] | 0.3755 | 0.04978 |
| López-Morales et al.[d] | 0.3719 | 0.04940 |

| Method | $\tau / \tau^\star$ | $N_{\mathrm{RHS}} / N^\star_{\mathrm{RHS}}$ |
|---|---|---|
| P-ERK$_{4;\{5,6,\dots,14\}}$ | 1.0 | 1.0 |
| P-ERK$_{4;12}$ | 1.37 | 2.05 |
| RK$_{4;4}$ | 2.20 | 2.33 |

*(Lower is better)*

[a] Vermeire. JCP. 2019

[b] Nasab, Vermeire. JCP. 2022

[c] Uranga et al. Int. J. Num Methods Eng. 2011

[d] López-Morales et al. 32nd AIAA AAC. 2014

# Application: Euler-Acoustic Simulation

## Co-rotating vortices

- Acoustic-Purturbation Equations (APE)[1]

[1] Ewert, Schröder. JCP. 2003

[2] Schlottke-Lakemper et al. Computers & Fluids. 2017

# Application: Euler-Acoustic Simulation

## Co-rotating vortices

- Acoustic-Purturbation Equations (APE)[1]
- with Lamb-vector source term from CEE

[1] Ewert, Schröder. JCP. 2003

[2] Schlottke-Lakemper et al. Computers & Fluids. 2017

# Application: Euler-Acoustic Simulation

## Co-rotating vortices

- Acoustic-Purturbation Equations (APE)[1]
- with Lamb-vector source term from CEE
- Hybrid method[2]: CAA + CFD

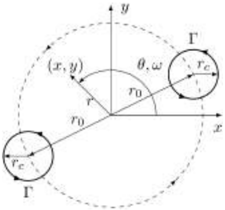
[1] Ewert, Schröder. JCP. 2003

[2] Schlottke-Lakemper et al. Computers & Fluids. 2017

# Application: Euler-Acoustic Simulation

## Co-rotating vortices

- Acoustic-Purturbation Equations (APE)[1]
- with Lamb-vector source term from CEE
- Hybrid method[2]: CAA + CFD



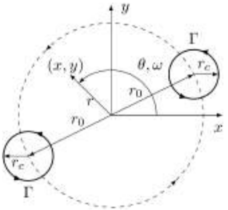

[1] Ewert, Schröder. JCP. 2003

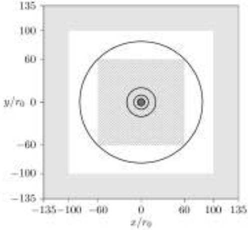
[2] Schlottke-Lakemper et al. Computers & Fluids. 2017

# Application: Euler-Acoustic Simulation

## Co-rotating vortices

- Acoustic-Purturbation Equations (APE)[1]
- with Lamb-vector source term from CEE
- Hybrid method[2]: CAA + CFD



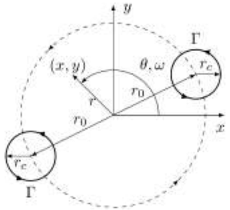

[1] Ewert, Schröder. JCP. 2003

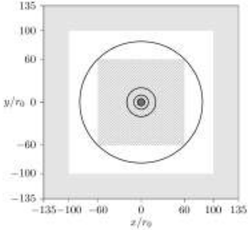
[2] Schlottke-Lakemper et al. Computers & Fluids. 2017

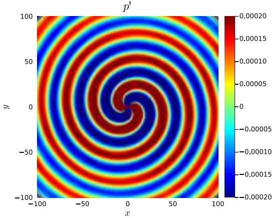
# Application: Euler-Acoustic Simulation

## Co-rotating vortices

- Acoustic-Purturbation Equations (APE)[1]
- with Lamb-vector source term from CEE
- Hybrid method[2]: CAA + CFD




[1] Ewert, Schröder. JCP. 2003

[2] Schlottke-Lakemper et al. Computers & Fluids. 2017

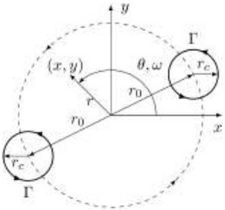
# Application: Euler-Acoustic Simulation

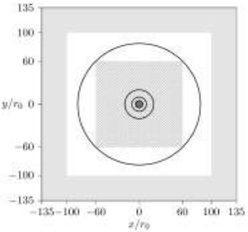
## Co-rotating vortices

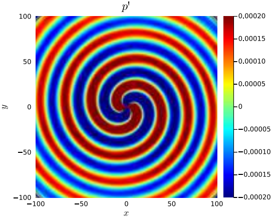
- Acoustic-Purturbation Equations (APE)[1]
- with Lamb-vector source term from CEE
- Hybrid method[2]: CAA + CFD

| Method | $\tau / \tau^\star$ | $N_{\text{RHS}} / N^\star_{\text{RHS}}$ |
|---|---|---|
| P-ERK$_{4;\{5,6,8,13\}}$, P-ERK$_{4;\{5,6,9,14\}}$ | 1.0 | 1.0 |
| P-ERK$_{4;13}$, P-ERK$_{4;14}$ | 1.60 | 1.87 |
| NDB$_{4;14}$ | 1.80 | 2.20 |
| TD$_{4;8}$ | 2.62 | 3.02 |
| CFR$_{4;6}$ | 3.82 | 4.26 |
| RK$_{4;4}$ | 4.72 | 4.52 |

*(Lower is better)*




[1] Ewert, Schröder. JCP. 2003

[2] Schlottke-Lakemper et al. Computers & Fluids. 2017

# Contents

# Conclusion & Outlook

**Summary:**

- Non-intrusive multirate time-integration

**Up next:**

# Conclusion & Outlook

**Summary:**

- Non-intrusive multirate time-integration
  $\rightarrow$ MoL and stabilized schemes

**Up next:**

# Conclusion & Outlook

**Summary:**

- Non-intrusive multirate time-integration
  $\rightarrow$ MoL and stabilized schemes
  $\rightarrow$ High-order in time & space

**Up next:**

# Conclusion & Outlook

**Summary:**

- Non-intrusive multirate time-integration
  $\rightarrow$ MoL and stabilized schemes
  $\rightarrow$ High-order in time & space
  $\rightarrow$ Conservative & Consistent

**Up next:**

# Conclusion & Outlook

**Summary:**

- Non-intrusive multirate time-integration
  $\rightarrow$ MoL and stabilized schemes
  $\rightarrow$ High-order in time & space
  $\rightarrow$ Conservative & Consistent

**Up next:**

- $p = 4$ P-ERK schemes with error-based timestep control

# Conclusion & Outlook

**Summary:**

- Non-intrusive multirate time-integration
  → MoL and stabilized schemes
  → High-order in time & space
  → Conservative & Consistent

**Up next:**

- $p = 4$ P-ERK schemes with error-based timestep control
- Combination of local & multirate time-stepping

# Thank you for your attention!

Questions?


GitHub Repository: https://github.com/trixi-framework/Trixi.jl

Documentation: https://trixi-framework.github.io/Trixi.jl/stable/

Slack: https://join.slack.com/t/trixi-framework/