Recent Advances in the Gridap Ecosystem: Scalable Solvers, Hybrid Methods, & Model Reduction

Santiago Badia Jordi Manyer Nicholas Mueller

Monash University, Melbourne, Australia
Delft University of Technology, Delft, Netherlands

Julia4PDEs 2026 · Vrije Universiteit Amsterdam · March 26–27, 2026

Outline

1. Gridap.jl

2. Hybrid Polytopal Methods

3. Scalable Geometric Solvers

4. Reduced order modelling

What is Gridap?

Gridap

- Finite Element (in very broad sense!) framework 100% in Julia
- First commit 15-3-2019 (7yo now!)
- 6,400+ commits, 100,000+ code lines, 50+ contributors
- Compatible FEM (grad, div, curl), DG, adaptivity, hybridisable methods (HDG, HHO, ...), unfitted FEM, surface PDEs, geometric multigrid, BDDC, ROM...
- 850 stars at Github (gold standard deal.ii 1.6k)
- Users: Google-X, Princeton, MIT, INRIA, Caltech...

S. Badia and F. Verdugo (2020). "Gridap: An extensible Finite Element toolbox in Julia". *Journal of Open Source Software* 5.52, p. 2520

The Ecosystem

Gridap

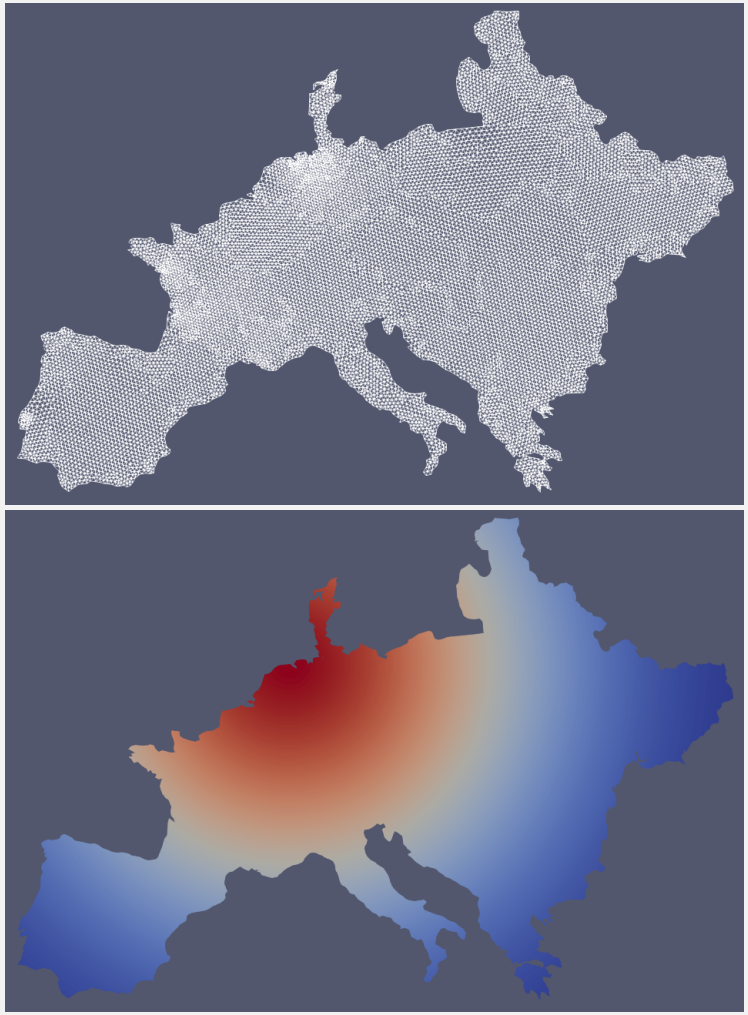
- `Gridap.jl` — core library
- `GridapDistributed.jl` — MPI parallelism through `PartitionedArrays.jl`
- `GridapSolvers.jl` — Scalable geometrical solvers
- `GridapROMs.jl` — Reduced order modelling
- ... and much more! Find out more about the **ecosystem** at `https://gridap.github.io/Gridap.jl/stable/ecosystem/`.

F. Verdugo and S. Badia (2022). "The software design of Gridap: a finite element package based on the Julia JIT compiler". *Computer Physics Communications* 276, p. 108341

What is Gridap?

High-level interface for PDE discretisations and solution, which closely follows the **mathematical formulation**:

```
using Gridap
using GridapGmsh
using DrWatson

u(x) = exp(-((x[1] - 38)^2 + (x[2] - 46)^2) / 1000)
model = GmshDiscreteModel(datadir("europe.msh"))

order = 1
reffe = ReferenceFE(lagrangian, Float64, order)
V = FESpace(model, reffe;dirichlet_tags="boundary")
U = TrialFESpace(V, u)

Ω  = Triangulation(model)
dΩ = Measure(Ω, 2*(order-1))
f(x) = -Δ(u)(x)
a(u,v) = ∫( ∇(u) ⋅ ∇(v) )dΩ
l(v)   = ∫( f⋅v )dΩ

op = AffineFEOperator(a, l, U, V)
uh = solve(op)
writevtk(
  Ω, datadir("europe"),
  cellfields=["uh"=>uh, "u"=>u, "eh" => uh-u]
)
```



Leveraging Julia's just-in-time compilation and lazy evaluations

Outline

1. Gridap.jl

2. Hybrid Polytopal Methods

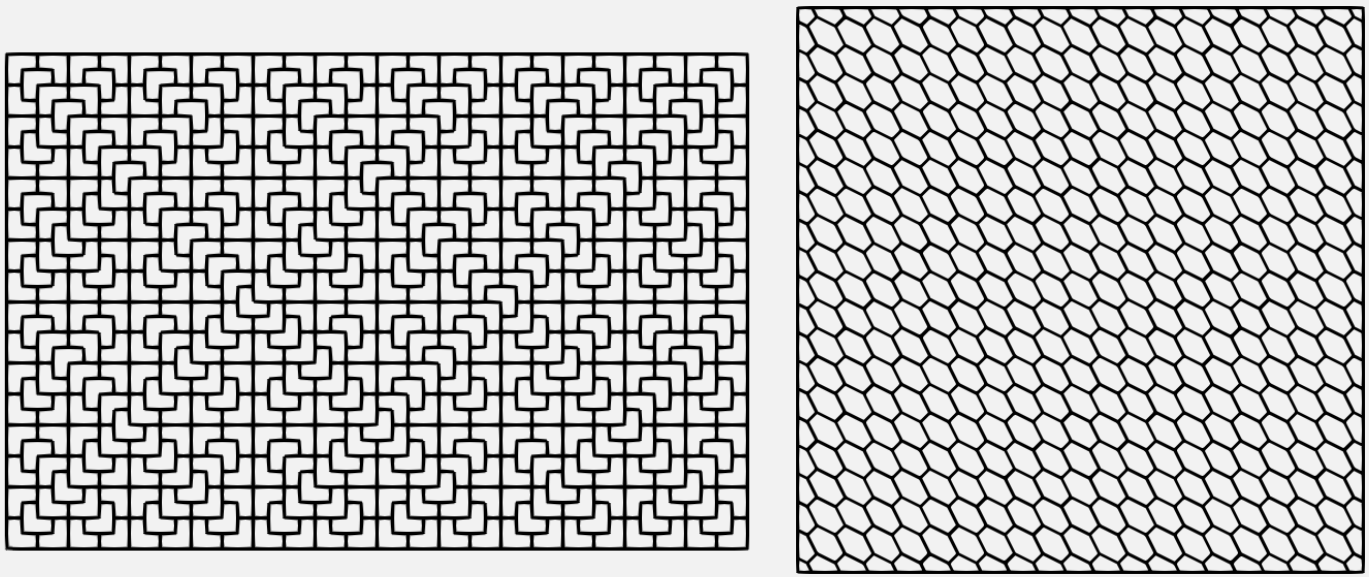
3. Scalable Geometric Solvers

4. Reduced order modelling

Overview: polytopal methods in Gridap

Available in **Gridap v0.19.8+**.

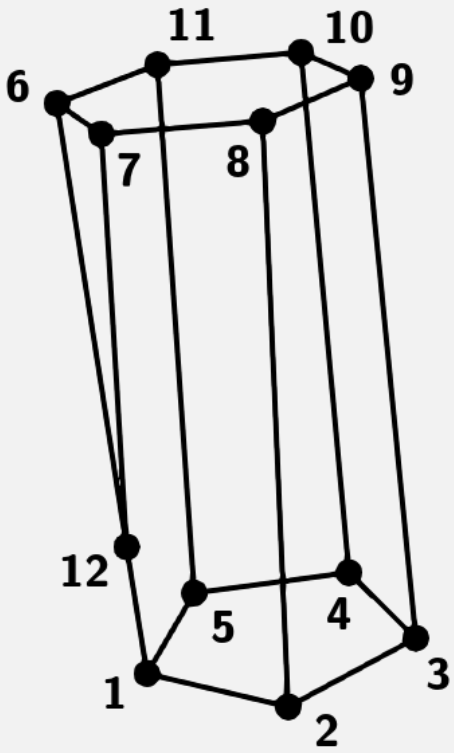
- General polyhedra and polytopal meshes.
- Patch integration and local assembly and solves.
- Support for hybrid methods: HDG, HHO, ... on polytopal meshes.

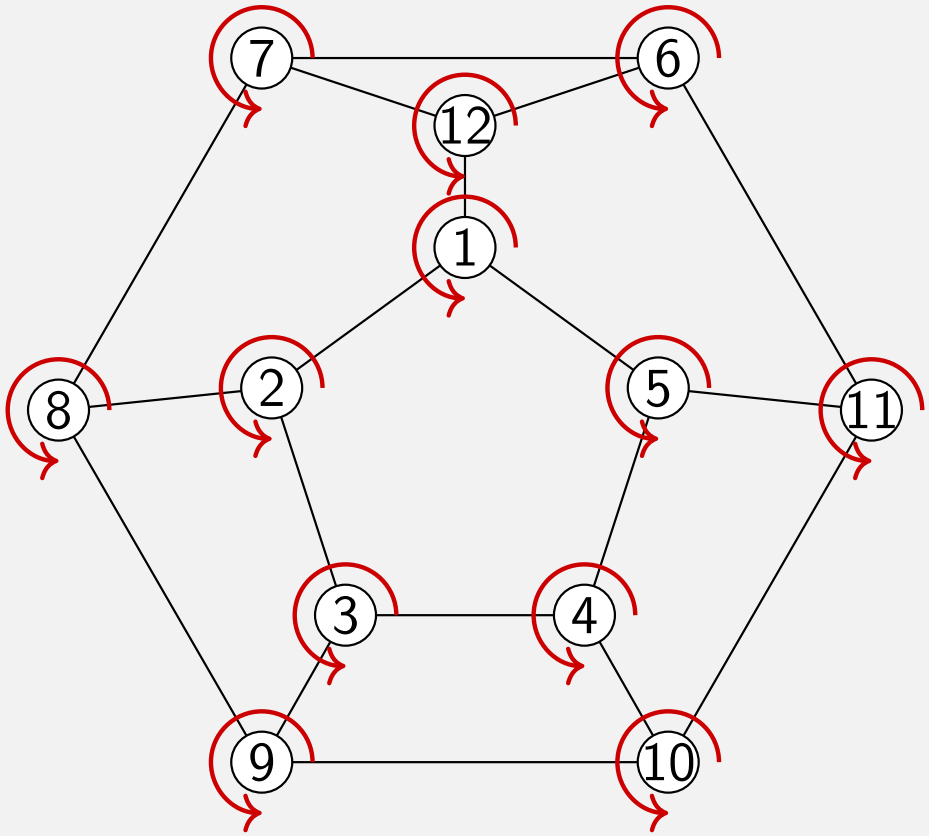


Find out more about the new features in J. Manyer, J. Tushar, and S. Badia (2026). *A natural language framework for non-conforming hybrid polytopal methods in Gridap.jl*. arXiv: 2603.00880 [cs.MS].

Polyhedron representation

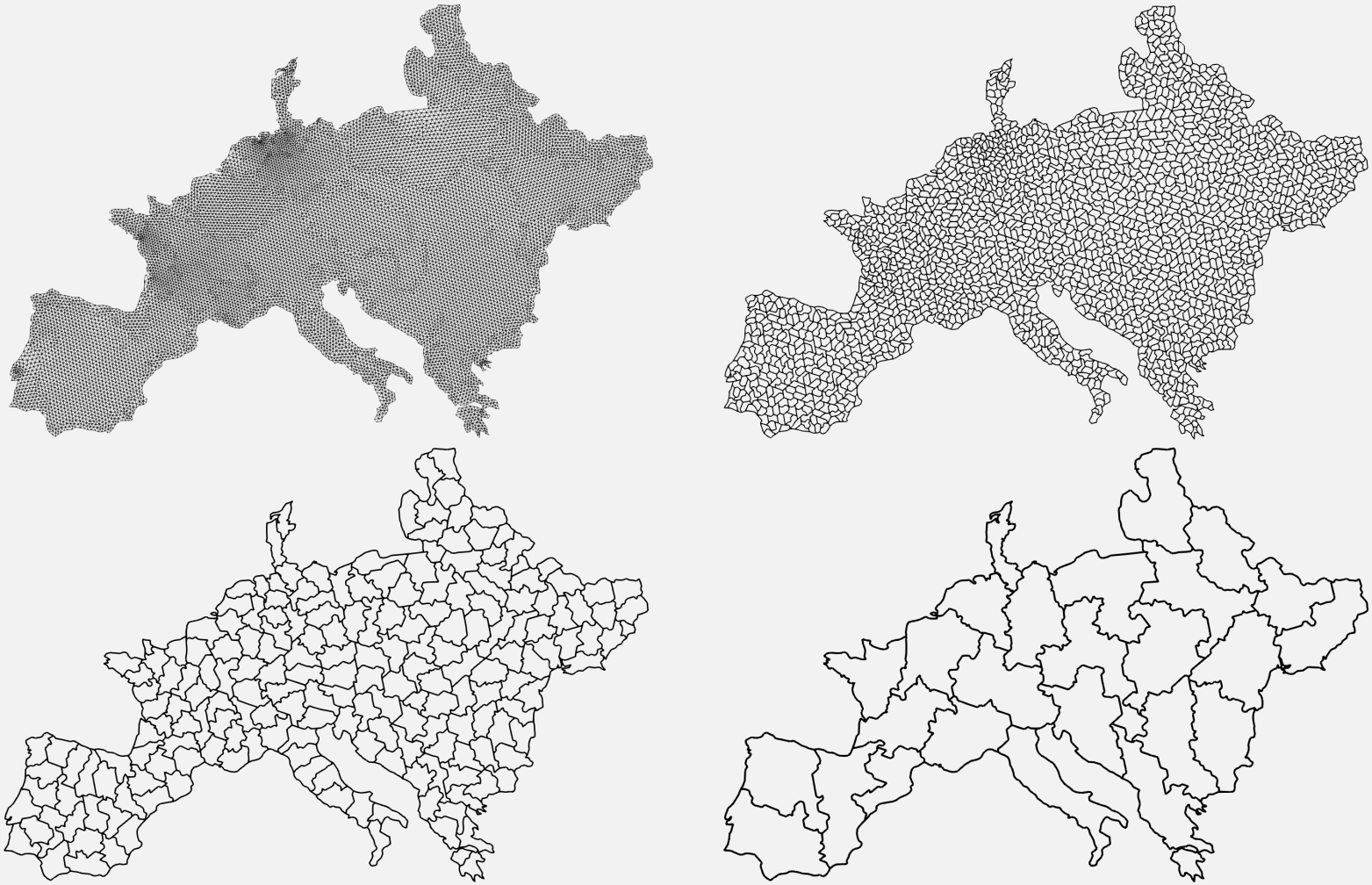
Polygons/Polyhedrons as **rotation systems**.





S. Badia, P. A. Martorell, and F. Verdugo (2022). "Geometrical discretisations for unfitted finite elements on explicit boundary representations". *Journal of Computational Physics* 460, p. 111162

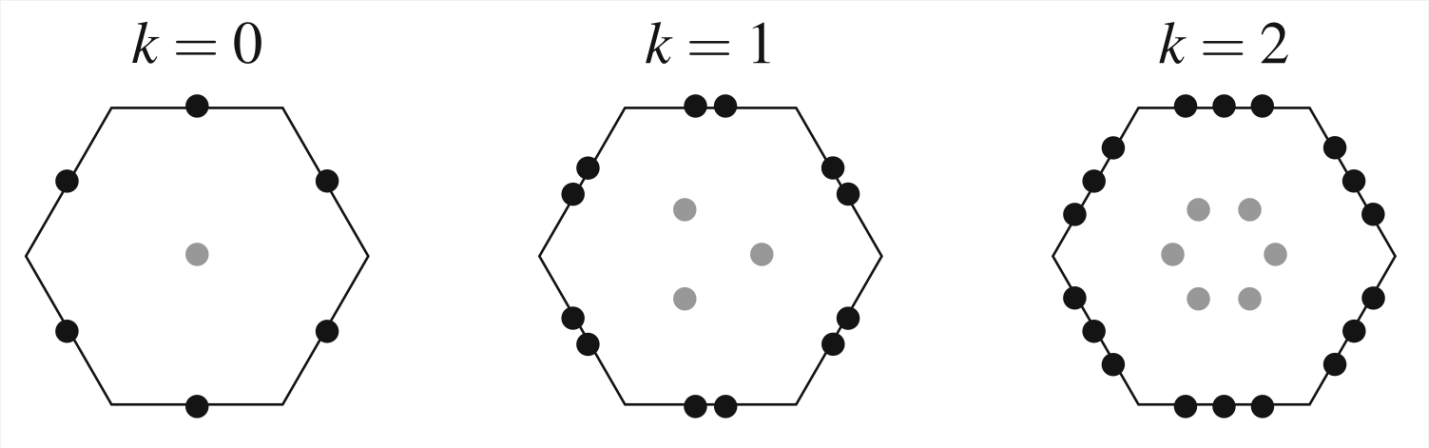
Agglomeration: building mesh hierarchies



Reducing computational cost without perturbing the geometry

Hybrid non-conforming methods

Hybridizable Discontinuous Galerkin (**HDG**), Hybrid High-Order (**HHO**), ...



- Unknowns tied to **cells and faces**.
- **Static condensation**: cell unknowns eliminated locally.
- **Local solves**: local problems have to be solved on each cell during assembly.

Static condensation

On each cell:

$$\begin{bmatrix} A_{ii} & A_{ib} \\ A_{bi} & A_{bb} \end{bmatrix} \begin{bmatrix} u_i \\ u_b \end{bmatrix} = \begin{bmatrix} f_i \\ f_b \end{bmatrix}$$

Global system $Su_b = g$, with

$$S = A_{bb} - A_{bi}A_{ii}^{-1}A_{ib}$$
$$g = f_b - A_{bi}A_{ii}^{-1}f_i$$

- Requires **local assembly** and **local solves** on each cell.
- Both u_i and u_b can be **multi-variable**.

```
Ui = ... # Eliminated space
Ub = ... # Retained space
style = BlockMultiFieldStyle(2)
X = MultiFieldFESpace([Ui, Ub]; style)

# Define the sub-domains
ptopo = PatchTopology(model)

a((ui,ub),(vi,vb)) = ... # Bilinear form
l((vi,vb)) = ... # Linear form

op = StaticCondensationOperator(ptopo,X,a,l)
ui, ub = solve(op)
```

- `ptopo`: Patch Topology, encodes local subdomain information
- `Ui`, `Ub` - any FESpace
- `BlockMultiFieldStyle` - first block eliminated, second block retained

Local operators

Example: HHO reconstruction
$\sim$ elliptic projection on each cell

Maps $(u_t, u_f) \to r$ solving

$$\int_T \nabla r \cdot \nabla w = \int_T \nabla u_t \cdot \nabla w + \int_{\partial T} (u_f - u_t) \nabla w \cdot n,$$

$$\int_T r = \int_T u_t \quad \text{(via Lag. mult.)}$$

Constrained local solve on each cell:

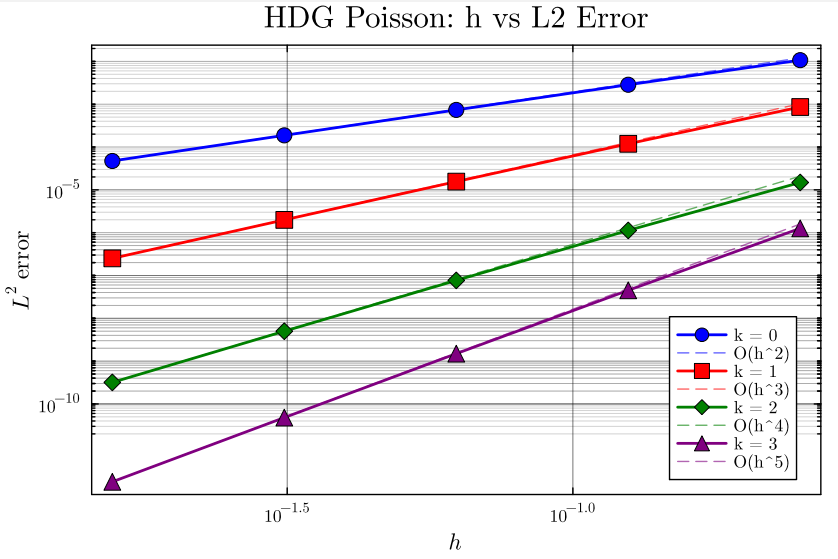
$$\begin{bmatrix} A_{rr} & A_{r\lambda} \\ A_{\lambda r} & 0 \end{bmatrix} \begin{bmatrix} r \\ \lambda \end{bmatrix} = \begin{bmatrix} A_{rt} & A_{rf} \\ A_{\lambda t} & 0 \end{bmatrix} \begin{bmatrix} u_t \\ u_f \end{bmatrix}$$

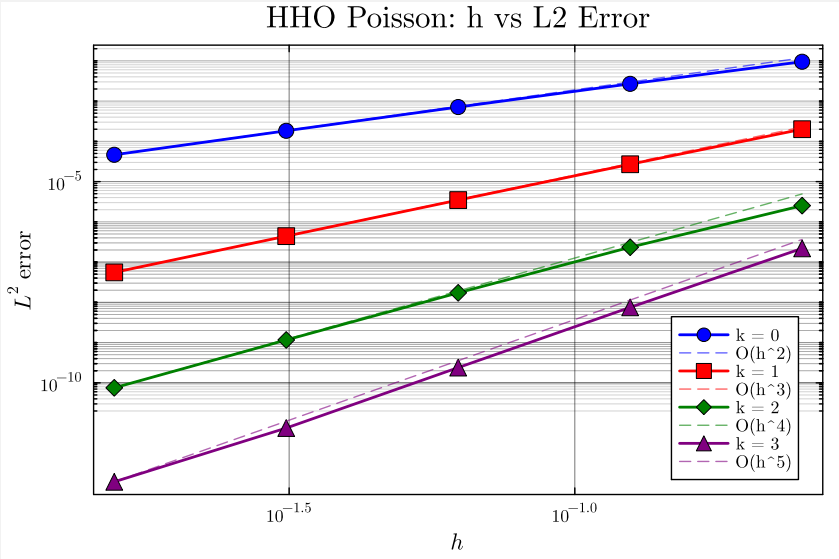
Cell-wise solution r used **within the global bilinear form**.

```
function reconstruction_operator(Ω,ptopo,X,order)
  L = PolytopalFESpace(Ω,Float64,order+1)
  Λ = PolytopalFESpace(Ω,Float64,0)

  Ωp = PatchTriangulation(Ω,ptopo)
  Γp = PatchBoundaryTriangulation(Ω,ptopo)
  dΩp = Measure(Ωp,2*(order+1))
  dΓp = Measure(Γp,2*(order+1))
  n = get_normal_vector(Γp)
  lhs((u,λ),(v,μ))   = ∫(∇(u)⋅∇(v) + μ⋅u + λ⋅v)dΩp
  rhs((uT,uF),(v,μ)) = ∫(∇(uT)⋅∇(v) + uT⋅μ )dΩp +
                       ∫((uF - uT)⋅(n⋅∇(v)))dΓp

  style = BlockMultiFieldStyle(2,(1,1))
  W = MultiFieldFESpace([L,Λ];style)
  R = LocalOperator(
    LocalPenaltySolveMap(), ptopo, W, X, lhs, rhs;
    space_out = L
  )
  return R
end

R = reconstruction_operator(Ω, ptopo, X, order)
function a(x,y)
  RuT, RuF = R(x)
  RvT, RvF = R(y)
  return ∫((∇(RuT)+∇(RuF)) ⋅ (∇(RvT)+∇(RvF)))dΩp
end
```

Summary: polytopal hybrid methods in Gridap

- Full explanation and drivers in:
  J. Manyer, J. Tushar, and S. Badia (2026). *A natural language framework for non-conforming hybrid polytopal methods in Gridap.jl*. arXiv: 2603.00880 [cs.MS]
- Interactive drivers can be found in `https://gridap.github.io/Tutorials/dev/`.





Outline

1. Gridap.jl

2. Hybrid Polytopal Methods

3. Scalable Geometric Solvers

4. Reduced order modelling

. . . why another solver library?

Algebraic solvers are abundant and library-agnostic:

- **Julia:** `IterativeSolvers.jl`, `Krylov.jl`, `AlgebraicMultigrid.jl`, `PartitionedSolvers.jl`, `...`
- **Artifacts:** PETSc, Trilinos, Pardiso, . . .

However: Discretization-aware geometrical solvers are the only option for complex PDEs (multi-variable, indefinite, . . .).

GridapSolvers.jl: `https://github.com/gridap/GridapSolvers.jl`

Unified and composable interface for algebraic and geometric solvers for the Gridap ecosystem.

J. Manyer, A. F. Martín, and S. Badia (2024). "GridapSolvers.jl: Scalable multiphysics finite element solvers in Julia". *Journal of Open Source Software* 9.102, p. 7162

Some applications

Used accross a wide range of applications: `GridapMHD.jl`, `GridapGeosciences.jl`, `GridapTopOpt.jl`, ...

- **Block preconditioners** for multi-physics problems
- **Geometric Multigrid** for conforming discretizations (Lagrangian, RT, ND) and non-conforming hybrid methods (HDG, HHO) on agglomerated meshes.
- **Domain Decomposition** methods: Overlapping Schwarz, BDDC (conforming, hybrid), Auxiliary space methods, ...

Huge composability and flexibility: users can easily build their own preconditioners by combining algebraic and geometric components, and leveraging the high-level interface to the discretization.

We will focus on two examples: **Block preconditioners for Stokes** and **GMG for hybrid non-conforming methods**.

Block preconditioners

Example: Inc. Stokes flow

$$\begin{bmatrix} A & B^T \\ B & 0 \end{bmatrix} \begin{bmatrix} u \\ p \end{bmatrix} = \begin{bmatrix} f \\ 0 \end{bmatrix}$$

Block preconditioner:

$$P^{-1} = \begin{bmatrix} A^{-1} & 0 \\ 0 & \tilde{S}^{-1} \end{bmatrix}$$

with $\tilde{S} = M_p \sim -BA^{-1}B^T$.

Extremely flexible framework:

- Diagonal / Triangular
- Blocks from matrix or user-defined
- Single / Multi-variable
- Linear / Non-linear

```
reffe_u = ReferenceFE(
  lagrangian,VectorValue{2,Float64},2)
reffe_p = ReferenceFE(
  lagrangian,Float64,1;space=:P)
V = TestFESpace(
  model,reffe_u,dirichlet_tags="boundary")
Q = TestFESpace(model,reffe_p;constraint=:zeromean)
U = TrialFESpace(V,u)

style = BlockMultiFieldStyle()
X = MultiFieldFESpace([U,Q];style)
Y = MultiFieldFESpace([V,Q];style)

dΩ = Measure(Triangulation(model),4)
a((u,p),(v,q)) = ∫(∇(v)⊙∇(u))dΩ
               - ∫((∇⋅v)*p + (∇⋅u)*q)dΩ
l((v,q)) = ∫(v⋅f)dΩ
op = AffineFEOperator(a,l,X,Y)

solver_u = PETScLinearSolver()
solver_p = CGSolver(JacobiLinearSolver())
blocks = [LinearSystemBlock(),
          BiformBlock((p,q)->∫(p*q)dΩ,Q,Q)]
P = BlockDiagonalSolver(blocks,[solver_u,solver_p])

solver = FGMRESSolver(30,P)
uh, ph = solve(solver,op)
```

Traditional Geometric Multigrid

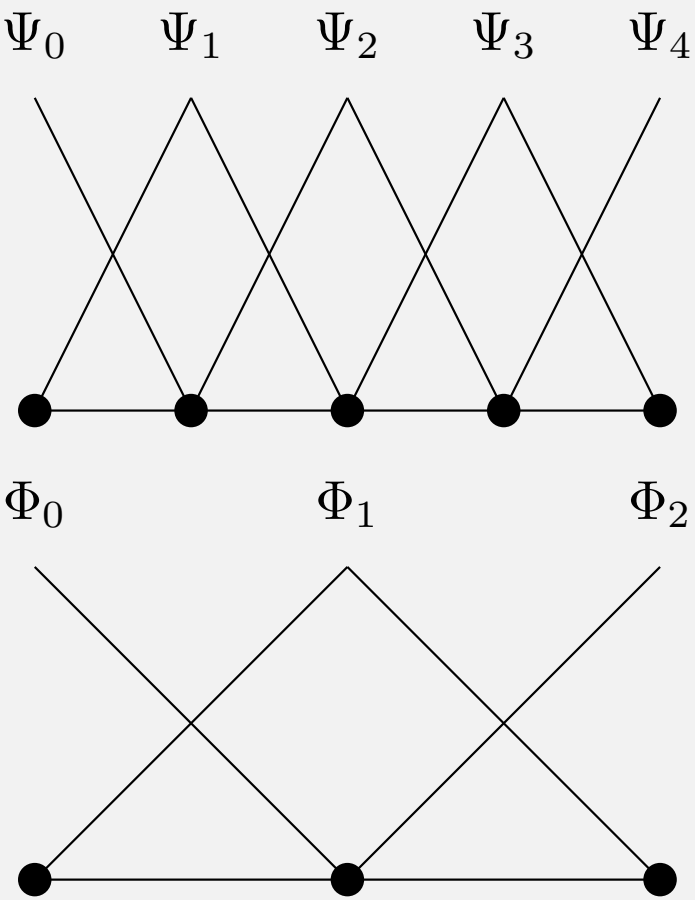
Traditionally designed for nodal finite elements on strcutured meshes:

Ingredients:

- Smoother
- Prolongation/restriction operators
- Coarse grid solver

Main idea:

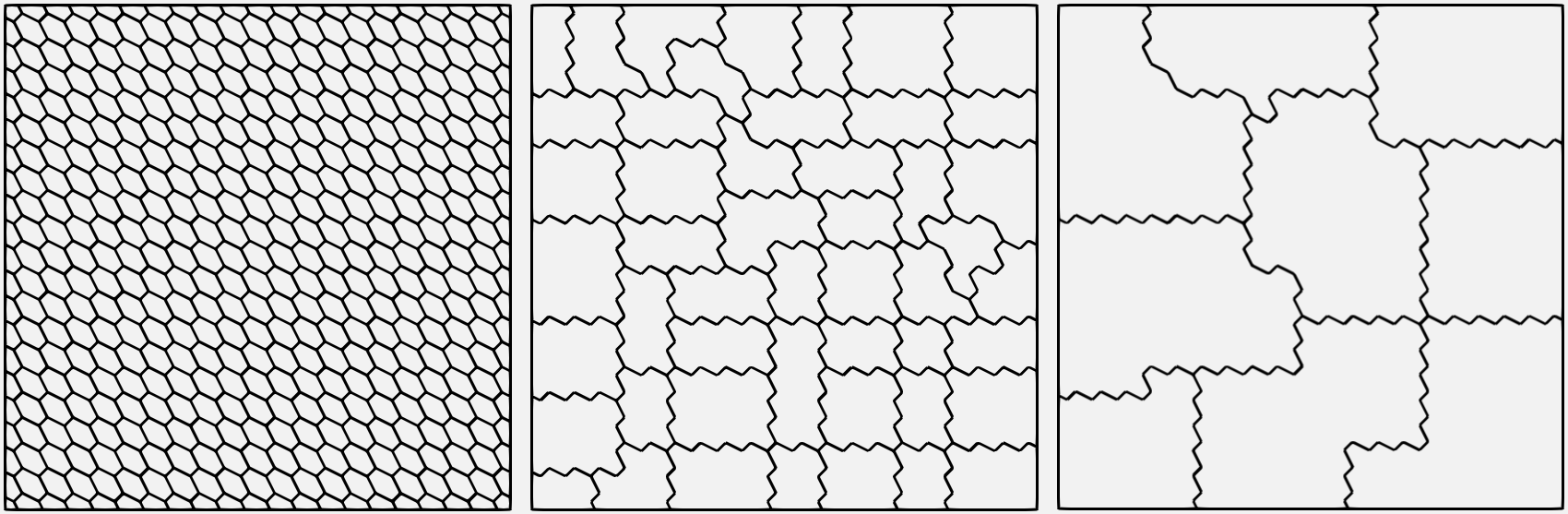
- Smoothers see **high-frequency** error components.
- Coarse grid solver sees **low-frequency** error components.



GMG for hybrid non-conforming methods

Idea: extend the traditional GMG ingredients to the hybrid polytopal setting on agglomerated meshes.

Challenge: We work with statically-condensed systems.



S. Badia and J. Manyer (2026). *Geometric Multigrid solvers for Hybrid High-Order methods on polytopal meshes*. arXiv: 2603.00860 [math.NA]

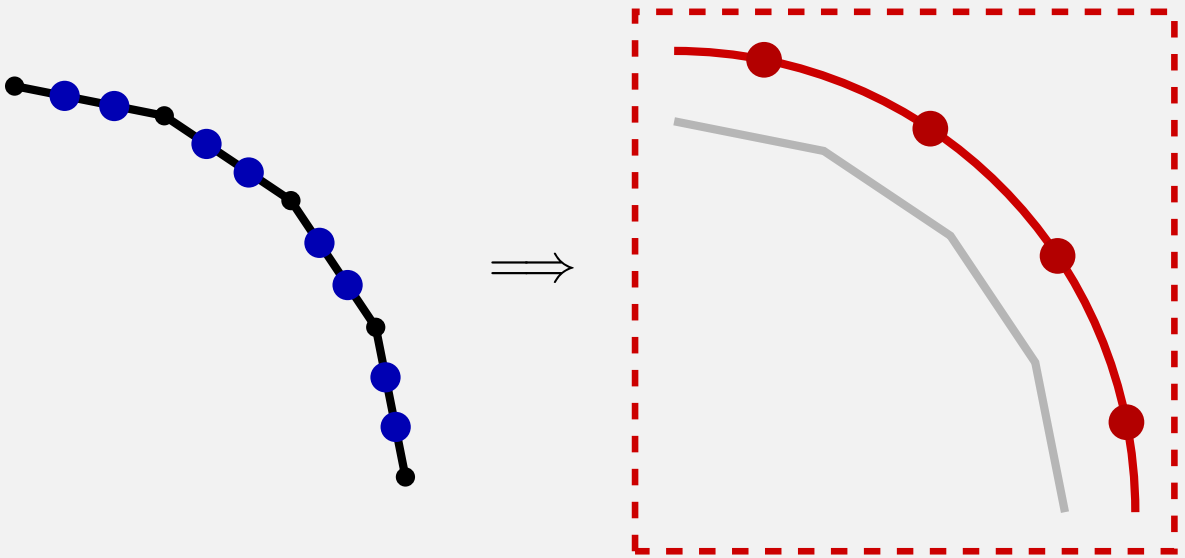
Minimal interface spaces

As cells get coarsened, the number of planar faces per cell grows. To have a scalable GMG, we need to keep the number of DOFs per interface **bounded**.

$$\mathcal{P}^k(F) = \mathbb{P}^0(\Omega)|_F + \nabla\mathbb{P}^{k+1}(\Omega)|_F \cdot \boldsymbol{n_F} \subseteq [\mathbb{P}^k(F)]^d \cdot \boldsymbol{n_F}$$

Construction:

$$m_F(\boldsymbol{p}, \boldsymbol{q}) = \int_F (\boldsymbol{p} \cdot \boldsymbol{n_F})(\boldsymbol{q} \cdot \boldsymbol{n_F}) \;\rightarrow\; \text{QR}$$



L. Yemm (2022). "A New Approach to Handle Curved Meshes in the Hybrid High-Order Method". *Foundations of Computational Mathematics* 24, pp. 1049–1076

Prolongation operators

Challenge: How to go from coarse to fine skeleton?

- **Step 1:** coarse skeleton $\rightarrow$ coarse cell (reconstruction)
- **Step 2:** coarse cell $\rightarrow$ fine skeleton (averaging)

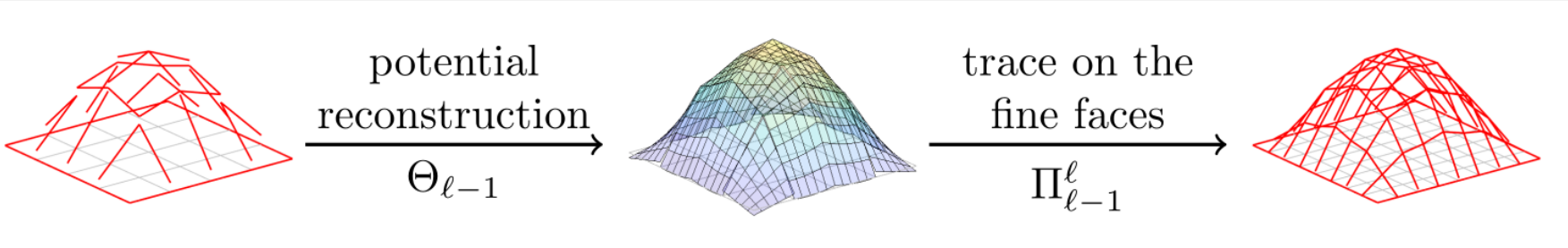
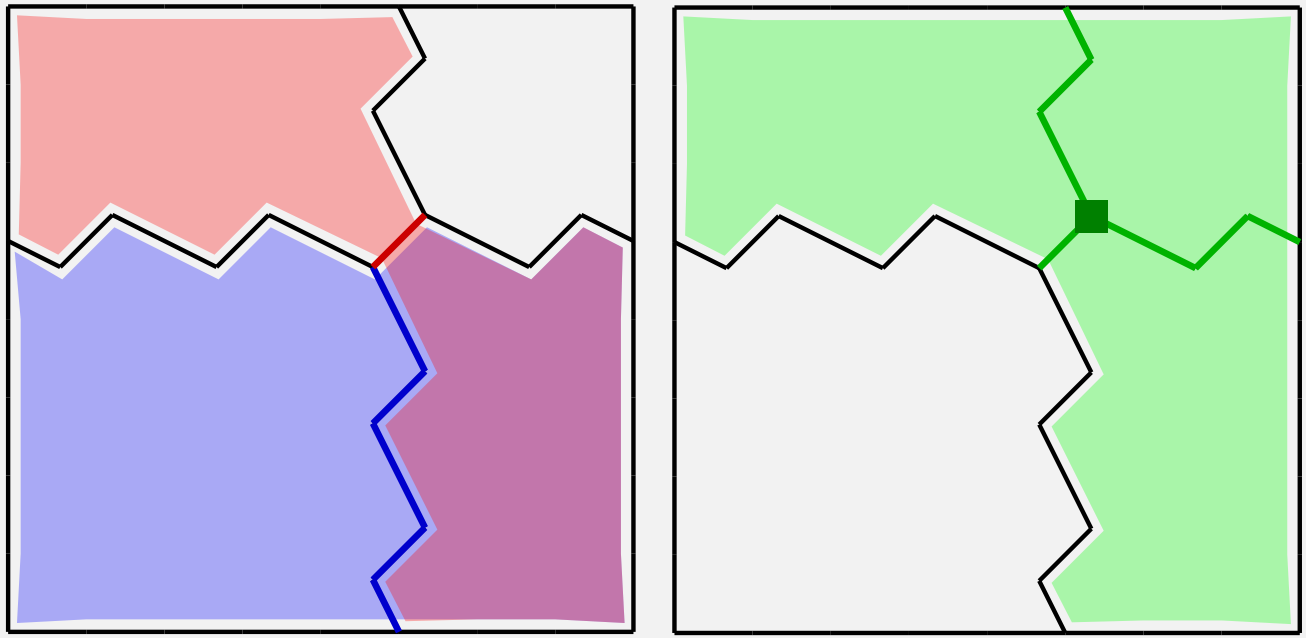


Figure taken from: D. A. Di Pietro et al. (2021). "An H-Multigrid Method for Hybrid High-Order Discretizations". *SIAM Journal on Scientific Computing* 43.5, S839–S861. eprint: https://doi.org/10.1137/20M1342471

Smoothers

Challenge: We coarsen the cells, but faces remain unchanged.
$\rightarrow$ Traditional smoothers are not robust.

Solution: Interface-wide patch-based smoothers.
$\rightarrow$ (Left) edge patch, (Right) vertex patch.



Results - 2D agglomerated voronoi meshes

	n_c	k	$I^R_{\ell-1}$					$I^U_{\ell-1}$				
			2	3	4	5	6	2	3	4	5	6
FP	10201	0	27	28	26	24	-	29	31	30	26	-
		1	28	26	26	26	-	26	27	26	24	-
		2	29	31	30	30	-	28	30	30	28	-
	19881	0	28	28	27	25	25	31	32	31	27	26
		1	28	27	27	27	27	27	28	28	25	24
		2	30	32	31	31	31	29	30	29	28	27
	40401	0	19	19	20	19	18	20	21	22	20	19
		1	19	18	19	19	19	19	19	20	18	18
		2	20	21	22	22	22	19	20	21	21	20
VP	10201	0	19	19	18	17	-	19	20	19	19	-
		1	17	17	17	17	-	17	17	17	17	-
		2	15	15	16	16	-	15	15	15	15	-
	19881	0	19	19	18	18	18	21	21	20	19	19
		1	17	17	17	18	18	18	17	17	17	17
		2	15	16	16	16	16	15	15	15	16	16
	40401	0	14	13	13	13	13	14	14	15	14	14
		1	11	11	11	11	11	12	12	12	12	12
		2	10	10	10	10	10	10	10	10	10	10

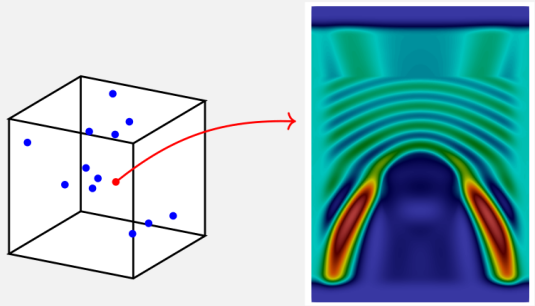
More results and details in S. Badia and J. Manyer (2026). *Geometric Multigrid solvers for Hybrid High-Order methods on polytopal meshes*. arXiv: 2603.00860 [math.NA].

Outline

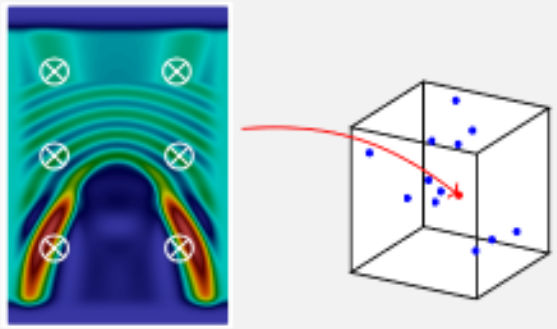
ROMs for parameterised PDEs

- ROMs: **Low-dimensional approximations** of mappings between quantities defined through parameterised PDEs
- Objective: cost reduction for **multi-query** evaluations
- Common applications:

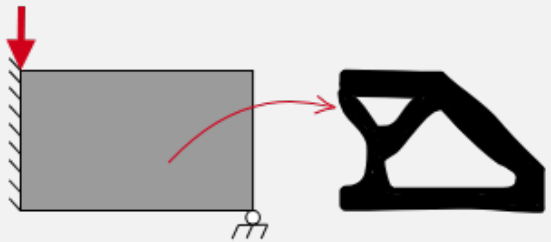
Forward problems
parameter $\mapsto$ state



Inverse problems
observations $\mapsto$ parameter

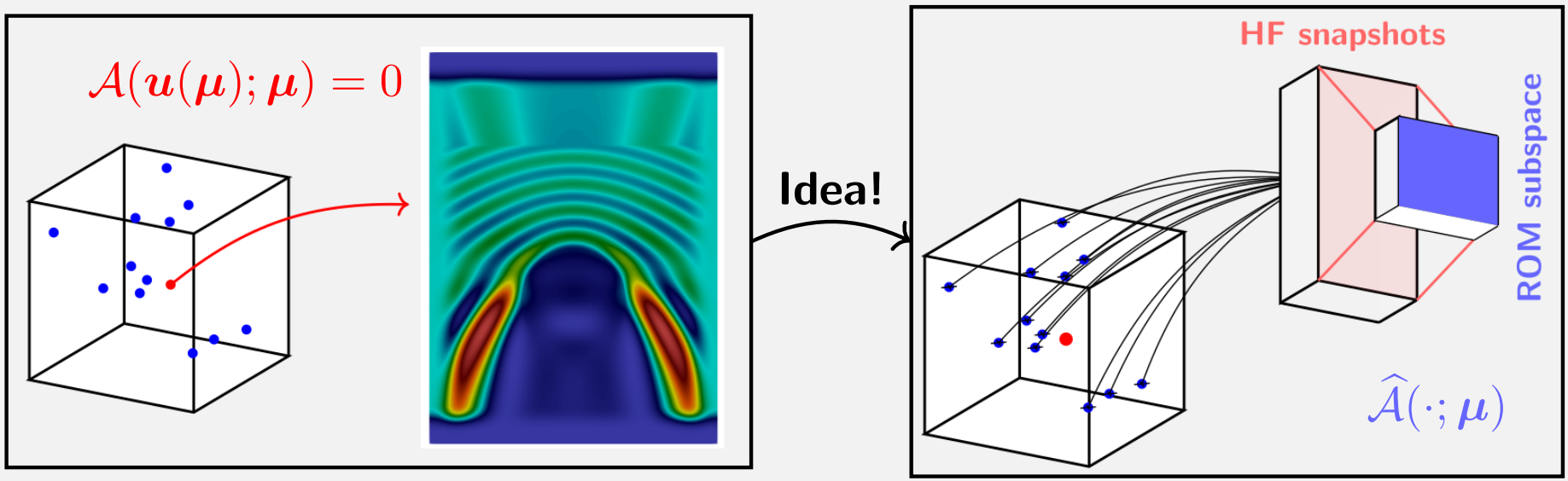


Optimal control
shape $\mapsto$ optimal objective



Linear ROMs for forward problems

Expensive (even with Gridap) **Reduction**



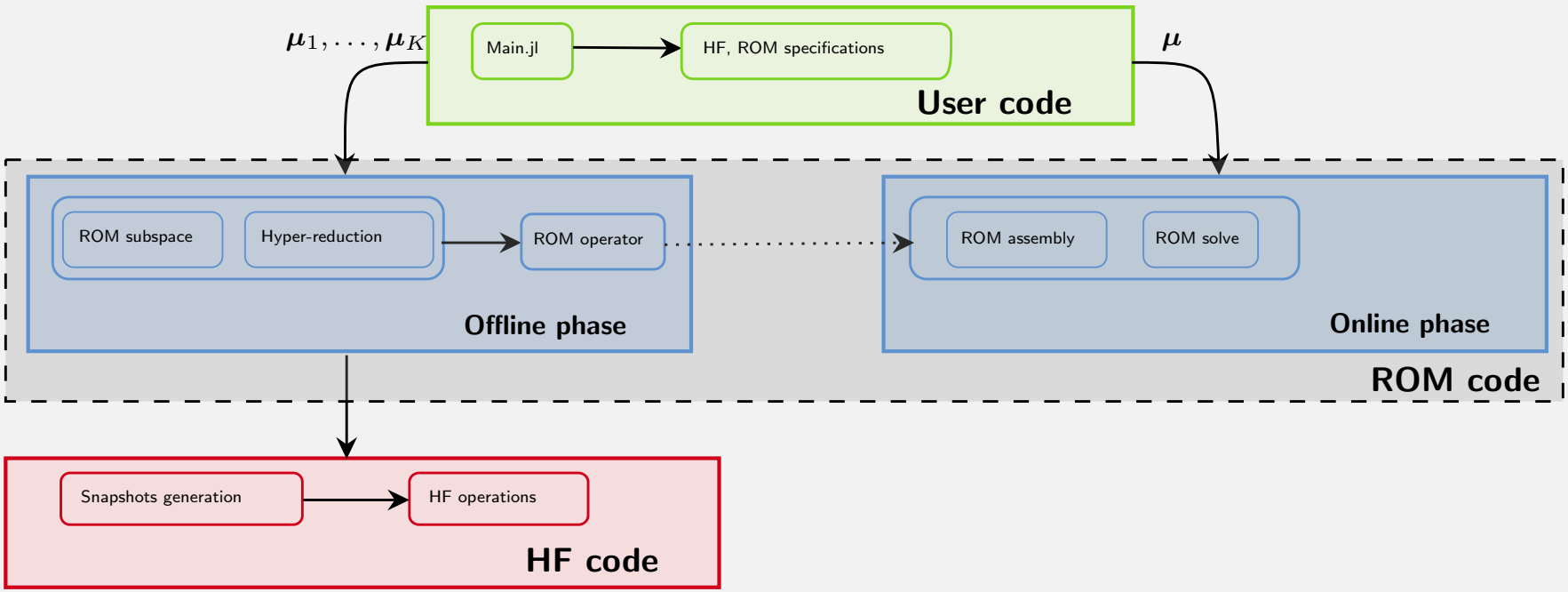
For any value of $\boldsymbol{\mu}$:

- **ROM approximation:** seek $\widehat{\boldsymbol{u}}(\boldsymbol{\mu}) : \widehat{\mathcal{A}}(\widehat{\boldsymbol{u}}(\boldsymbol{\mu});\boldsymbol{\mu}) = 0$
- Hopefully: $\widehat{\boldsymbol{u}}(\boldsymbol{\mu}) \approx \boldsymbol{u}(\boldsymbol{\mu})$

N. Mueller and S. Badia (2026). "GridapROMs.jl: Efficient reduced order modelling in the Julia programming language". *Computer Physics Communications* 320, p. 109985

Workflow

- **Offline-online splitting**:
  - Expensive offline phase $\Longrightarrow$ compute μ-independent quantities $\Longrightarrow$ **performed only once**
  - Cheap online phase $\Longrightarrow$ compute μ-dependent quantities $\Longrightarrow$ repeat **for any new μ**



- Offline phase is the **computational bottleneck**

GridapROMs.jl: efficient ROMs in Julia

- **Motivation**: ROMs are
  - often implemented *ad-hoc*
  - relatively unexplored in Julia (`ModelOrderReduction.jl`, `SciML.jl`)
- **Native integration** with Gridap.jl
  - inherits the **expressiveness** and **flexibility**
  - reuses & extends lazy machinery for parameterised PDEs for **efficiency**
- Implements several ROM techniques: POD, Tucker, tensor-train strategies, Discrete Empirical Interpolation Method (DEIM), Optimal Sparse Subset Selection (SSOR), etc.
- **Advanced capabilities**: parameterised geometries, saddle-point stabilisation, local subspaces, distributed-in-memory generation of snapshots
- **Applicability**: linear, nonlinear, single-field, multi-field, steady, unsteady, coupled problems

Implementation: naive Gridap vs GridapROMs

Naive for loop with Gridap

```
pspace = ... # some 2D parametric space
Vₕ = ... # some test FE space
μ = realization(pspace;nparams=10)
for μᵢ in μ
    α(x) = μ[2]
    g(x) = μᵢ[1]*x
    a(u,v) = ∫(α*∇(v)⋅∇(u))dΩ
    f(v) = ∫(v)dΩ
    Uₕ = TrialFESpace(Vₕ,g)
    feop = AffineFEOperator(a,f,Uₕ,Vₕ)
    uh = solve(feop)
end
```

GridapROMs

```
# parameterized functions
g(x) = μ -> μ[1]*x
gₚ(μ) = parameterize(g,μ)
α(x) = μ -> μ[2]
αₚ(μ) = parameterize(α,μ)

# parameterized forms
aₚ(μ,u,v) = ∫(αₚ(μ)*∇(v)⋅∇(u))dΩ
fₚ(μ,v) = ∫(v)dΩ

# parameterized HF operator
Uₕₚ = ParamTrialFESpace(test,gₚ)
opₚ = ParamLinearOperator(aₚ,fₚ,pspace,Uₕₚ,Vₕ)

# parameterized HF solution
uₚ, = solve(LUSolver(),opₚ,μ)
```

Gridap	GridapROMs
–	AbstractParamRealization
Function	AbstractParamFunction
AbstractArray	AbstractParamArray
LazyArray	LazyArray
TrialFESpace	ParamTrialFESpace
FEOperator	ParamFEOperator
(Grid)	(ParamGrid)

Performance: naive Gridap vs GridapROMs

- Assembly of residuals and Jacobians for several values of the parameters for the steady Navier-Stokes equation

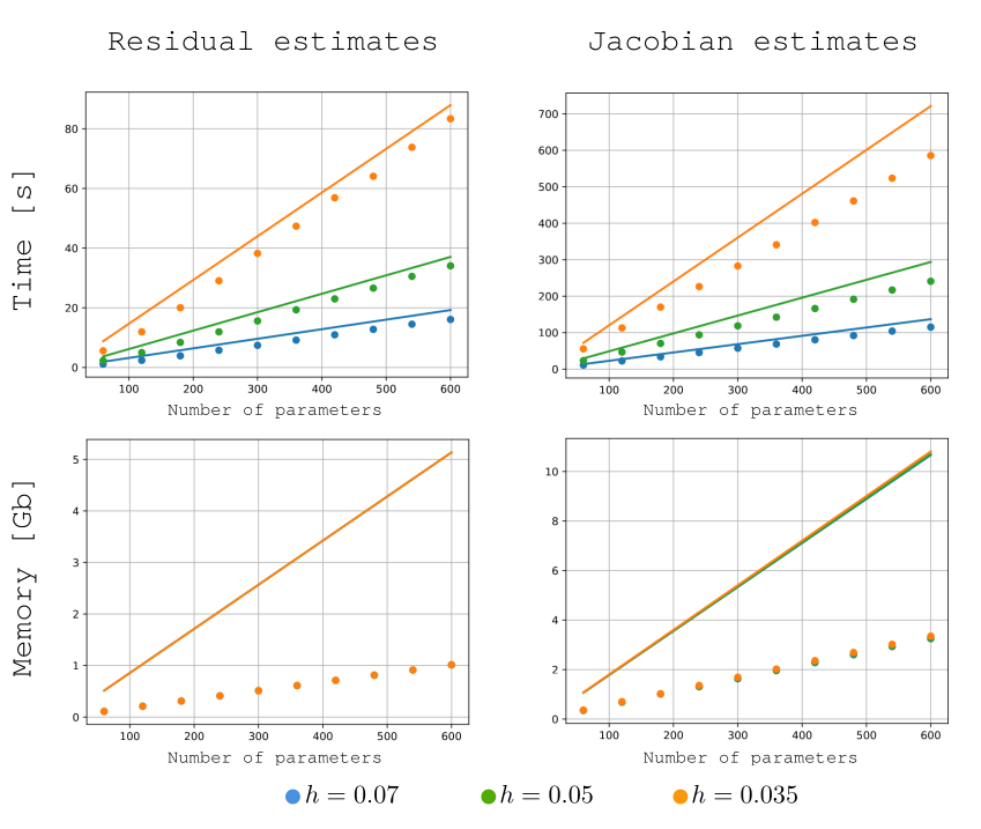


Figure: Solid lines: naive Gridap code; circles: GridapROMs code

- Noticeable **speedup in memory** due to **reuse of caches** across different parameters

Offline-online splitting in GridapROMs

Offline phase

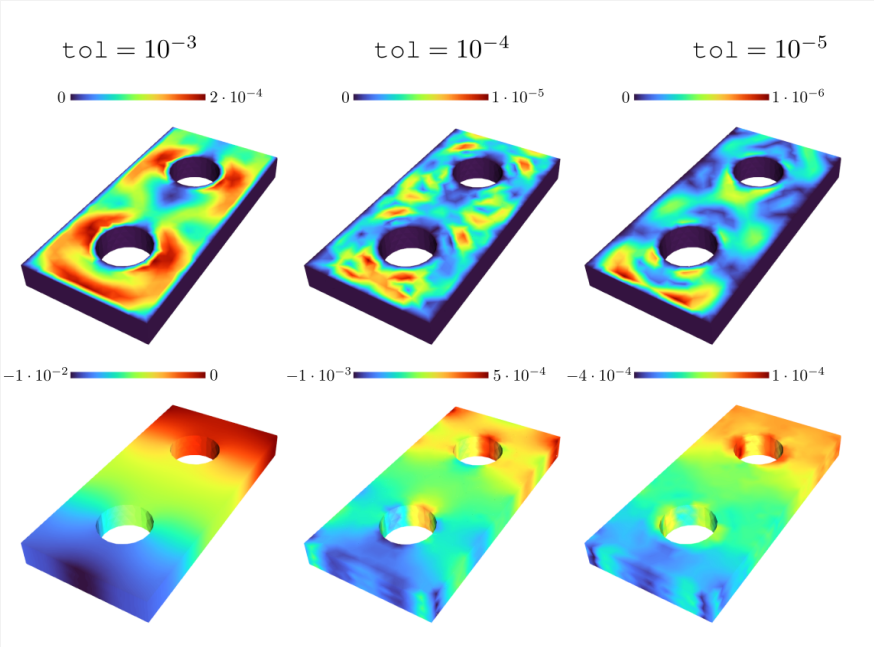
```
# ROM specs
tol = 1e-4
slvr = LUSolver()
red_sol = PODReduction(tol;nparams=20)
rbslvr = RBSolver(
  slvr,red_sol;
  nparams_jac=5,nparams_res=1
)

################ offline phase ################
# generate snapshots
snaps, = solution_snapshots(rbslvr,op_p)

# reduction HF spaces and weak form
Û,V̂  = reduced_spaces(rbslvr,op_p,snaps)
â,f̂  = reduced_weak_form(rbslvr,op_p,Û,V̂ ,snaps)
rbop_p = RBOperator(op_p,Û,V̂ ,â,f̂ )

# alternatively, use some syntactic sugar
# rbop_p = reduced_operator(rbslvr,op_p,snaps)
################ offline phase ################
```

Online phase

```
################ online phase #################
# generate online parameters
μ_on = realization(pspace;sampling=:uniform)

# call reduced solver
x̂_on,rbstats = solve(rbslvr,rbop_p,μ_on)
################ online phase #################

################ postprocess ##################
# call HF solver
x_on,stats = solve(slvr,op_p,μ_on)

# this returns the errors and speedups
eval_performance(
  rbslvr,op_p,rbop_p,x_on,x̂_on,stats,rbstats
)
################ postprocess ##################
```

- Check out the repo at `github.com/gridap/GridapROMs.jl/` for more details
- More complex examples are described in N. Mueller, S. Badia, and Y. Zhao (2026). "Reduced basis solvers for unfitted methods on parameterized domains". *Computer Methods in Applied Mechanics and Engineering* 451, p. 118610

Example: 3D unsteady Navier-Stokes flow ($Re = 100$)



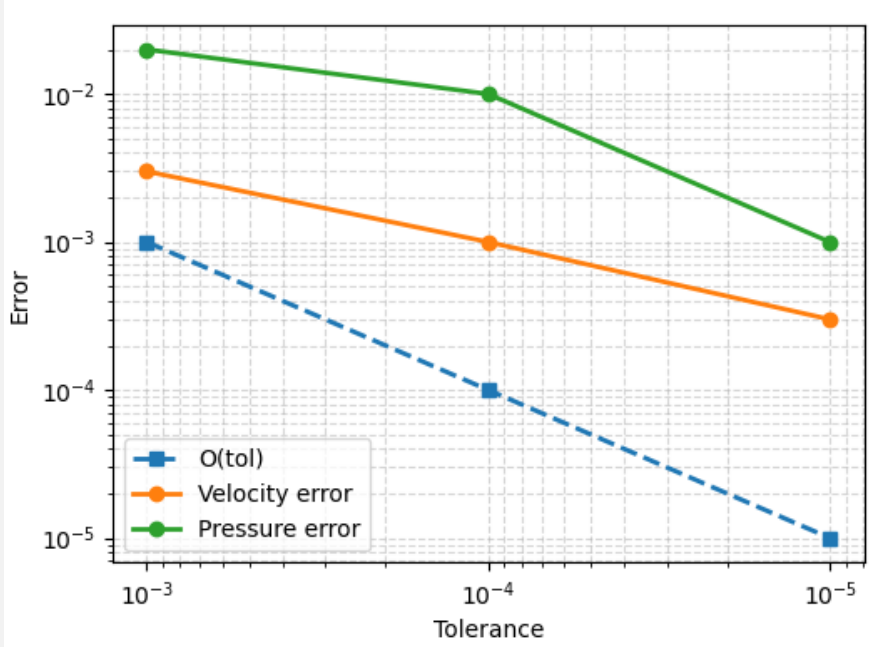


Figure: Left: point-wise error at a fixed time; right: space-time relative error.

tol = 10^{-3}		tol = 10^{-4}		tol = 10^{-5}	
SU-T	**SU-M**	**SU-T**	**SU-M**	**SU-T**	**SU-M**
$3 \cdot 10^5$	$7 \cdot 10^2$	$3 \cdot 10^5$	$6 \cdot 10^2$	$2 \cdot 10^5$	$4 \cdot 10^2$

Table: Speedup in execution time (SU-T), and memory usage (SU-M)

Summary

Support for polytopal methods

J. Manyer, J. Tushar, and S. Badia (2026). *A natural language framework for non-conforming hybrid polytopal methods in Gridap.jl*. arXiv: 2603.00880 [cs.MS]

GridapSolvers.jl

J. Manyer, A. F. Martín, and S. Badia (2024). "GridapSolvers.jl: Scalable multiphysics finite element solvers in Julia". *Journal of Open Source Software* 9.102, p. 7162

GridapROMs.jl

N. Mueller and S. Badia (2026). "GridapROMs.jl: Efficient reduced order modelling in the Julia programming language". *Computer Physics Communications* 320, p. 109985