LUND UNIVERSITY

# Introduction to Python Bindings for Dune-Fem

Claus Führer, Robert Klöfkorn, Viktor Linders

# Distributed and Unified Numerics Environment

A set of C++ modules for implementing grid-based numerical methods

**Distributed** development of **distributed** code for **distributed** machines.

**Core developers:**

- In Berlin: Gräser
- In Dresden: Sander, Praetorius, Burchardt
- In Heidelberg: Bastian, Blatt, Kempf
- In Lund: RK
- In Münster: Ohlberger, Engwer, Fahlke
- in Stuttgart: Grüninger
- In Warwick: Dedner
- ...

Dune papers with $> 1000$ citations together (Google scholar)


Bastian, Blatt, Dedner, Engwer, K., Ohlberger, Sander. A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part I: Abstract Framework. Computing, 2008.

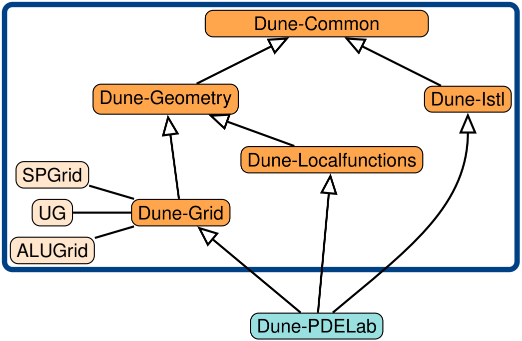
Bastian, Blatt, Dedner, Engwer, K., Kornhuber, Ohlberger, Sander. A Generic Grid Interface for Parallel and Adaptive Scientific Computing. Part II: Implementation and Tests in DUNE. Computing, 2008.

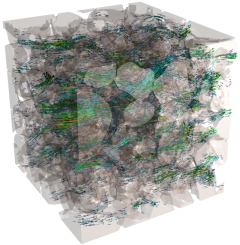
Dedner, K., Nolte, Ohlberger. A generic interface for parallel and adaptive discretization schemes: abstraction principles and the Dune-Fem module. Computing, 2010.

Bastian, Blatt, Dedner, Dreier, Engwer, Fritze, Gräser, Kempf, K., Ohlberger, Sander. The DUNE Framework: Basic Concepts and Recent Developments, CAMWA, 2020.


For details visit: `https://dune-project.org`

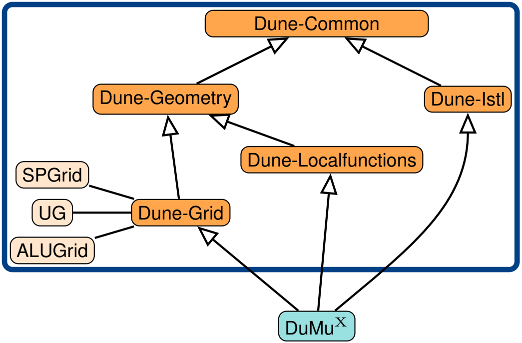
# Dune Modules

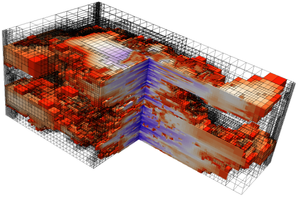


- Dune-Common basic infrastructure and build system
- Dune-Geometry implementation of generic geometry classes
- Dune-Grid abstract grid interface
- Dune-Istl Iterative Solver Template Library (Krylov, PAMG, ...)
- Dune-Localfunctions implementation of shape functions, ...
- Dune-PDELab discretization module

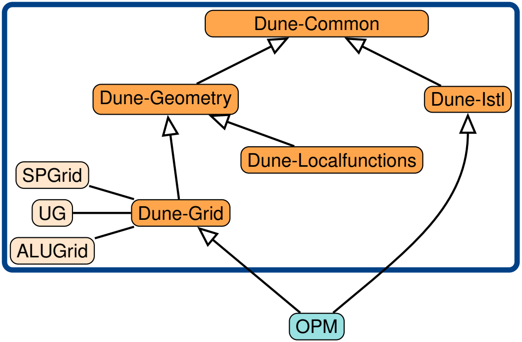
# Dune Modules

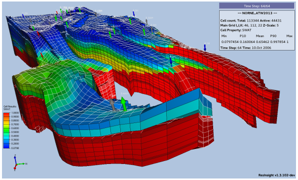


- Dune-Common basic infrastructure and build system
- Dune-Geometry implementation of generic geometry classes
- Dune-Grid abstract grid interface
- Dune-Istl Iterative Solver Template Library (Krylov, PAMG, ...)
- Dune-Localfunctions implementation of shape functions, ...
- DuMu$^{X}$ flow and transport processes in porous media

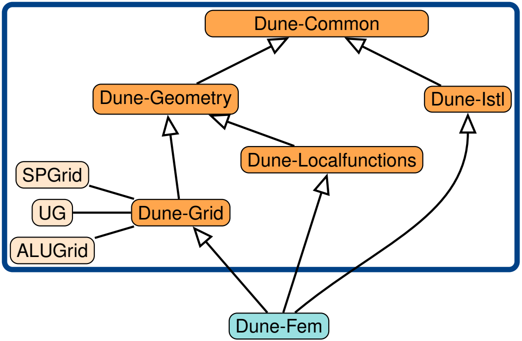
# Dune Modules



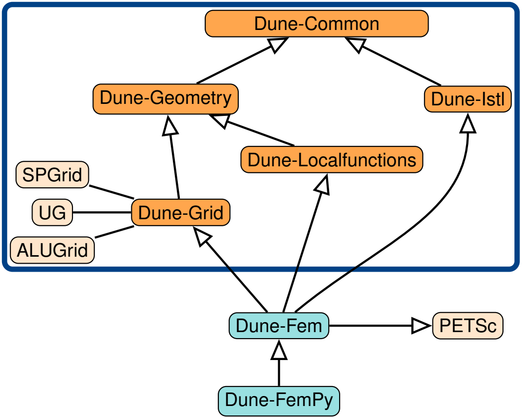
- Dune-Common basic infrastructure and build system
- Dune-Geometry implementation of generic geometry classes
- Dune-Grid abstract grid interface
- Dune-Istl Iterative Solver Template Library (Krylov, PAMG, ...)
- Dune-Localfunctions implementation of shape functions, ...
- Open Porous Media Initiative (SINTEF, Equinor, NORCE and others)

# Dune Modules



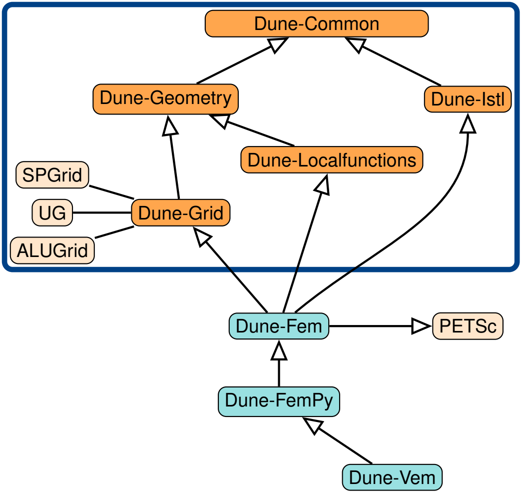

- Dune-Common basic infrastructure and build system
- Dune-Geometry implementation of generic geometry classes
- Dune-Grid abstract grid interface
- Dune-Istl Iterative Solver Template Library (Krylov, PAMG, ...)
- Dune-Localfunctions implementation of shape functions, ...
- Dune-Fem discretization module
  - discrete function spaces
  - data management for adaptivity
  - efficient communication (oberver pattern)
  - data I/O and checkpointing
  - ...

# Dune Modules



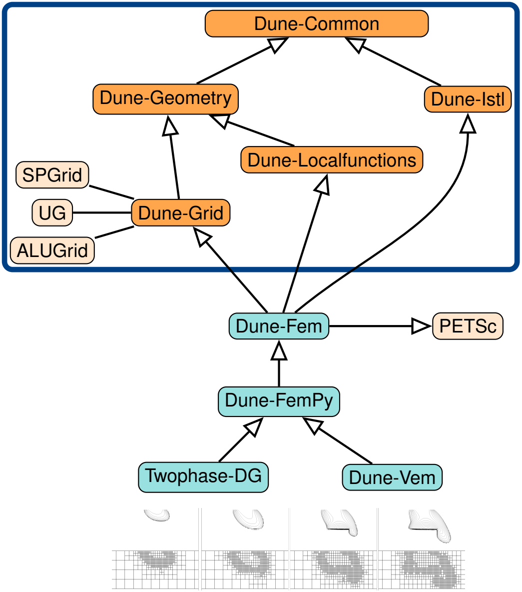

- Dune-Common
- Dune-Geometry
- Dune-Grid
- Dune-Istl
- Dune-Localfunctions
- Dune-Fem discretization module
- Dune-FemPy python bindings for Dune-Fem using FENICS UFL for variational description of PDEs.

# Dune Modules




- Dune-Common
- Dune-Geometry
- Dune-Grid
- Dune-Istl
- Dune-Localfunctions
- Dune-Fem discretization module
- Dune-FemPy python bindings for Dune-Fem using FENICS UFL for variational description of PDEs.
- Dune-Vem implementation of Virtual Element method

# Dune Modules




- Dune-Common
- Dune-Geometry
- Dune-Grid
- Dune-Istl
- Dune-Localfunctions
- Dune-Fem discretization module
- Dune-FemPy python bindings for Dune-Fem using FENICS UFL for variational description of PDEs.
- Dune-Vem implementation of Virtual Element method
- Twophase-DG implements twophase flow using DG discretizations

# Statistics about Dune (Feb 2022)

Core modules (`https://www.openhub.net/p/dune-project`)

- 52 person-years (resulting costs of 2.78 million dollar (55 000 per year)
- overall $\approx 307\,000$ lines
  - $\approx 199\,000$ lines of code
  - $\approx 60\,000$ lines of comments
  - ...

# Statistics about Dune (Feb 2022)

**Core modules** (`https://www.openhub.net/p/dune-project`)

- 52 person-years (resulting costs of 2.78 million dollar (55 000 per year)
- overall $\approx 307\,000$ lines
  - $\approx 199\,000$ lines of code
  - $\approx 60\,000$ lines of comments
  - ...

Further remarks:

- Development started in 2002
- license GLP with linking exception (LGPL in the future)
- DUNE 2.8 release of core modules
  - Project homepage (`https://dune-project.org`)
  - Mailing lists
  - gitlab for bug tracking, development, and discussion (`https://gitlab.dune-project.org`)
  - Automated testing system (CI)
- Yearly Dune Summer Schools starting 2007



LUND UNIVERSITY

# Statistics about Dune (Feb 2022)

**Core modules (`https://www.openhub.net/p/dune-project`)**

- 52 person-years (resulting costs of 2.78 million dollar (55 000 per year)
- overall $\approx 307\,000$ lines
  - $\approx 199\,000$ lines of code
  - $\approx 60\,000$ lines of comments
  - ...

Further remarks:

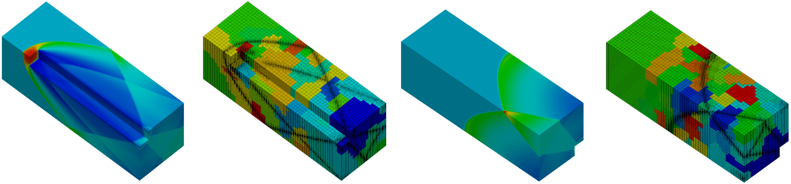
- Development started in 2002
- license GLP with linking exception (LGPL in the future)
- DUNE 2.8 release of core modules
  - Project homepage (`https://dune-project.org`)
  - Mailing lists
  - gitlab for bug tracking, development, and discussion (`https://gitlab.dune-project.org`)
  - Automated testing system (CI)
- Yearly Dune Summer Schools starting 2007
- **Coming up: Introduction to Dune, 4-8 April in Lund**

# Forward Facing Step 3d

## Simulation details (XC4000 SCC Karlsruhe, 2008):

- stabilized DG approach with quadratic polynomials ($k = 2 \hat{=} 50$ ukn. per cell)
- **fully unstructured** hexahedral grid (ALUGrid, also tetras)
- non-conforming grid adaptation in parallel (MPI)
  with dynamic load balancing (METIS)
- final adapted grid contains about $4.5$ **million** grid cells (uniform grid about $95$ million cells)
- **Adaptation** $5$**%** and **load-balancing** about $10$-$15$**%** of one timestep
- Load-balancing takes place approx. every $100$**th timestep**

| | speedup for one timestep | | | ODE solving per timestep | |
|---|---|---|---|---|---|
| $K$ | $S_{128\to K}$ | $\frac{128}{K} S_{128\to K}$ | $K$ | $S_{128\to K}$ | $\frac{128}{K} S_{128\to K}$ |
| 128 | | | 128 | | |
| 256 | 1.97 | 0.985 | 256 | 1.98 | 0.99 |
| 512 | 3.73 | 0.933 | 512 | 3.85 | 0.963 |





LUND UNIVERSITY

# Similar Software Packages

FEniCS project

deal.II

Firedrake

OpenFOAM

COMSOL

- Abaqus
- Calfem
- many more

Design goals: **Flexibility** and Efficiency and Modularity

- Separate grid structure and data
- Define abstract interfaces for each part (grid, discrete functions...)

Design goals: **Flexibility** and Efficiency and Modularity

- Separate grid structure and data
- Define abstract interfaces for each part (grid, discrete functions...) Example: Entity class is realized using the `generic bridge pattern`

```
class Entity< class Impl >
{
  Impl impl_;
public:
  bool isLeaf() const { return impl_.isLeaf(); }
  Geometry geometry () const { return impl_.geometry(); }
};
```

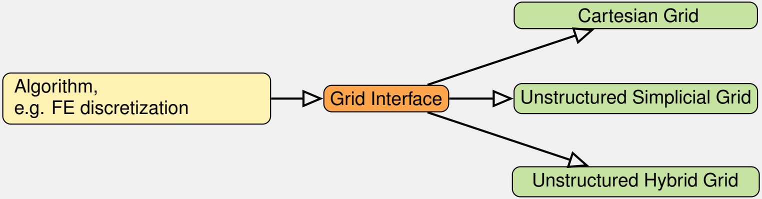
### Design goals: **Flexibility** and **Efficiency** and Modularity

- Separate grid structure and data
- Define abstract interfaces for each part (grid, discrete functions...)
- Base interface on mathematical formalism
  - `Grid` represents mathematical object $\mathcal{G}$
  - `Entity` represents mathematical object $E$
  - `DiscreteFunctionSpace` represents mathematical object $V_{\mathcal{G}}^{k}$

## Design goals: Flexibility and **Efficiency** and Modularity

- Separate grid structure and data
- Define abstract interfaces for each part (grid, discrete functions...)
- Base interface on mathematical formalism

1. Determine what algorithms require from grid and data structure to operate efficiently
2. Formulate algorithms based on this interface
3. Provide different implementations of the interface



# Dune-Fem

1. **Discrete spaces and discrete functions**
   - discrete function spaces (Lagrange, DG, ...)
   - discrete functions (adaptive DF, block vector DF, ...)
   - caching of basis functions
2. **Discretization schemes**
   - Lagrange FEM (generic, almost arbitrary order ... )
   - Finite Volume (first and second order)
   - Discontinuous Galerkin (various basis functions)
3. **implemented Runge Kutta solvers**
   - explicit Strong-Stability-Preserving Runge Kutta (SSP-RK) up to ord. $3$
   - Diagonally Implicit Runge Kutta (DIRK) methods up to order $3$
   - Semi Implicit Runge Kutta (SIRK) methods up to order $3$
4. **implemented Inverse Operators**
   - Generic Krylov methods
   - Dune-Istl (Krylov methods, ILU, AMG, ...)
   - PETSc (Krylov methods, ILU, HyPre, ML, ...)
   - SuiteSparse (direct solvers)
   - AMGX (GPU based, work in progress)
5. **Misc**
   - Restriction/prolongation strategies
   - DoF handling (automatic resize and DoF-compress)
   - Data I/O and check-pointing
   - Communication patterns
   - ...

# Discrete function spaces and discrete functions

Given a Grid $\mathcal{G}$ (called `GridView` or `Grid` or `GridPart`).
**A discrete function space is:** $\mathcal{D}_{\mathcal{G}} := \left(V_{\mathcal{G}}, (\mathcal{B}_E)_{E\in\mathcal{G}}, (\mu_E)_{E\in\mathcal{G}}\right)$

- $dim < \infty$ FunctionSpace
- BasisFunctionSet
- DofMapper
- Communication pattern
- ....

# Discrete function spaces and discrete functions

Given a Grid $\mathcal{G}$ (called `GridView` or `Grid` or `GridPart` ).
**A discrete function space is:** $\mathcal{D}_{\mathcal{G}} := \left(V_{\mathcal{G}}, (\mathcal{B}_E)_{E\in\mathcal{G}}, (\mu_E)_{E\in\mathcal{G}}\right)$

- $dim < \infty$ FunctionSpace
- BasisFunctionSet
- DofMapper
- Communication pattern
- ....

# Discrete function spaces and discrete functions

Given a Grid $\mathcal{G}$ (called `GridView` or `Grid` or `GridPart` ).
**A discrete function space is:** $\mathcal{D}_\mathcal{G} := \left(V_\mathcal{G}, (\mathcal{B}_E)_{E\in\mathcal{G}}, (\mu_E)_{E\in\mathcal{G}}\right)$

- $dim < \infty$ FunctionSpace
- BasisFunctionSet
- DofMapper
- Communication pattern
- ....

# Discrete function spaces and discrete functions

Given a Grid $\mathcal{G}$ (called `GridView` or `Grid` or `GridPart` ).
**A discrete function space is:** $\mathcal{D}_\mathcal{G} := \big(V_\mathcal{G}, (\mathcal{B}_E)_{E\in\mathcal{G}}, (\mu_E)_{E\in\mathcal{G}}\big)$

- $dim < \infty$ FunctionSpace
- BasisFunctionSet
- DofMapper
- Communication pattern
- ....

**A discrete function is:** $u_\mathcal{G} \in V_\mathcal{G}$ and

$$u_\mathcal{G} = \sum_{\psi\in\mathcal{B}_\mathcal{G}} u_\psi\, \psi \qquad\qquad u_{\mathcal{G}|E} = \sum_{\psi\in\mathcal{B}_\mathcal{G}} u_\psi\, \psi_{|E} = \sum_{i\in I_E} u_i^E\, \varphi_i^E\,.$$

with $u_i^E = u_{\mu_E(i)}$

- DoF data storage (e.g. `double*` , `BlockVector` , `numpy.array` )
- LocalFunction
- ....


LUND UNIVERSITY

# Discrete function spaces and discrete functions

Given a Grid $\mathcal{G}$ (called `GridView` or `Grid` or `GridPart` ).
**A discrete function space is:** $\mathcal{D}_{\mathcal{G}} := \big(V_{\mathcal{G}}, (\mathcal{B}_E)_{E\in\mathcal{G}}, (\mu_E)_{E\in\mathcal{G}}\big)$

- $dim < \infty$ FunctionSpace
- BasisFunctionSet
- DofMapper
- Communication pattern
- ....

**A discrete function is:** $u_{\mathcal{G}} \in V_{\mathcal{G}}$ and

$$u_{\mathcal{G}} = \sum_{\psi\in\mathcal{B}_{\mathcal{G}}} u_\psi\, \psi \qquad\qquad u_{\mathcal{G}|E} = \sum_{\psi\in\mathcal{B}_{\mathcal{G}}} u_\psi\, \psi_{|E} = \sum_{i\in I_E} u_i^E\, \varphi_i^E\, .$$

with $u_i^E = u_{\mu_E(i)}$

- DoF data storage (e.g. `double*` , `BlockVector` , `numpy.array` )
- LocalFunction
- ....

# Discrete function spaces and discrete functions

Given a Grid $\mathcal{G}$ (called `GridView` or `Grid` or `GridPart` ).
**A discrete function space is:** $\mathcal{D}_{\mathcal{G}} := \big(V_{\mathcal{G}}, (\mathcal{B}_E)_{E\in\mathcal{G}}, (\mu_E)_{E\in\mathcal{G}}\big)$

- $dim < \infty$ FunctionSpace
- BasisFunctionSet
- DofMapper
- Communication pattern
- ....

**A discrete function is:** $u_{\mathcal{G}} \in V_{\mathcal{G}}$ and

$$u_{\mathcal{G}} = \sum_{\psi\in\mathcal{B}_{\mathcal{G}}} u_\psi\,\psi \qquad\qquad u_{\mathcal{G}|E} = \sum_{\psi\in\mathcal{B}_{\mathcal{G}}} u_\psi\,\psi_{|E} = \sum_{i\in I_E} u_i^E\,\varphi_i^E\,.$$

with $u_i^E = u_{\mu_E(i)}$

- DoF data storage (e.g. `double*` , `BlockVector` , `numpy.array` )
- LocalFunction
- ....

# Discrete function spaces and discrete functions

Given a Grid $\mathcal{G}$ (called `GridView` or `Grid` or `GridPart` ).
**A discrete function space is:** $\mathcal{D}_\mathcal{G} := \left(V_\mathcal{G}, (\mathcal{B}_E)_{E\in\mathcal{G}}, (\mu_E)_{E\in\mathcal{G}}\right)$

- $dim < \infty$ FunctionSpace
- BasisFunctionSet
- DofMapper
- Communication pattern
- ....

**A discrete function is:** $u_\mathcal{G} \in V_\mathcal{G}$ and

$$u_\mathcal{G} = \sum_{\psi\in\mathcal{B}_\mathcal{G}} u_\psi\,\psi \qquad\qquad u_{\mathcal{G}|E} = \sum_{\psi\in\mathcal{B}_\mathcal{G}} u_\psi\,\psi_{|E} = \sum_{i\in I_E} u_i^E\,\varphi_i^E\,.$$

with $u_i^E = u_{\mu_E(i)}$

- DoF data storage (e.g. `double*` , `BlockVector` , `numpy.array` )
- LocalFunction
- ....

A. Dedner, R. Klöfkorn, M. Nolte, M. Ohlberger. ***A generic interface for parallel and adaptive scientific computing: Abstraction principles and the*** **Dune-Fem** ***module.*** Computing, 2010.

# How does it look like in Python

Generally:

```
from dune.grid import cartesianDomain
```

Create a domain $\Omega := [0,0] \times [2.5, 2.5] \subset \mathbb{R}^2$ with 100 cells in each direction

```
domain   = dune.grid.cartesianDomain([0,0],[2.5,2.5],[100,100])
```

Create a gridView (and implicitly a hierarchical grid – convenience)

```
gridView = dune.alugrid.aluConformGrid(domain)
```

Possible grid implementation:

```
function            module
------------------------------
aluGrid             dune.alugrid
aluConformGrid      dune.alugrid
aluCubeGrid         dune.alugrid
aluSimplexGrid      dune.alugrid
onedGrid            dune.grid
yaspGrid            dune.grid
...                 ...
```

# DiscreteFunctionSpace

Create a **scalar** discrete function space

```
space = dune.fem.space.lagrange( gridView, order=1 )
```

Create a **vector valued** discrete function space with $d \geq 1$

```
space = dune.fem.space.lagrange( gridView, dimRange=d, order=1 )
```

Available discrete function spaces:

```
function            module
--------------------------------
lagrange            dune.fem.space
lagrangehp          dune.fem.space
dgonb               dune.fem.space
dgonbhp             dune.fem.space
dglegendre          dune.fem.space
dglegendrehp        dune.fem.space
dglagrange          dune.fem.space
finiteVolume        dune.fem.space
p1Bubble            dune.fem.space
bdm                 dune.fem.space
raviartThomas       dune.fem.space
rannacherTurek      dune.fem.space
```

Special discrete function spaces:

```
combined            dune.fem.space
product             dune.fem.space
```

# DiscreteFunctionSpace and Storage

Create a **scalar** discrete function space with storage **numpy**:

```
space = dune.fem.space.lagrange( gridView, order=1 )
```

Create a **scalar** discrete function space with storage **istl**:

```
space = dune.fem.space.lagrange( gridView, order=1,
    storage="istl" )
```

Available storages:

```
storage            solvers from...
--------------------------------------------------------
numpy (default)    dune-fem, suitesparse, scipy with numpy
istl               dune-istl
petsc              PETSc, and petsc4py
```

**Note:** All storages can use all solvers by "paying" with an internal copy operation!

# DiscreteFunction

Creating a discrete function with interpolate and passing a `float` vector of length **dimRange**, i.e. $u_h = 0$:

```
uh   = space.interpolate( [0], name="u" )
```

Alternatively, pass an ufl expression to interpolate or use copy:

```
x = ufl.SpatialCoordinate(space)
initial_u = ufl.conditional(x[1]>1.25,1,0)
initial_v = ufl.conditional(x[0]<1.25,0.5,0)

uh   = space.interpolate( initial_u, name="u" )
uh_n = uh.copy()
vh   = space.interpolate( initial_v, name="v" )
vh_n = vh.copy()
```

Interpolate an ufl expression into an existing discrete function:

```
expr = ufl.conditional( ... )
uh.interpolate( expr )
```

# Operators

Operators map discrete functions (or grid functions) from a domain space to a range space, i.e. $\mathcal{L} : V \longrightarrow W$ (sometimes $V = W$)

```
# from before create gridView, space, and discrete function
domain   = dune.grid.cartesianDomain([0,0],[2.5,2.5],[100,100])
gridView = dune.alugrid.aluConformGrid(domain)
space = dune.fem.space.lagrange( gridView, order=1 )

u = ufl.TrialFunction(space)
v = ufl.TestFunction(space)

uh = space.interpolate( [0], name="u" )

# create model, here ufl expression
a = inner(grad(u), grad(v)) * dx

# create operator, here in practice V = W
laplace = dune.fem.operator.galerkin( a )
```

# Operators $V \neq W$

Creating an operator $\mathcal{L} : V \longrightarrow W$

```
# from before create gridView, space, and discrete function
fvspace = dune.fem.space.finiteVolume(uh.space.grid)
estimate = fvspace.interpolate([0], name="estimate")

u = ufl.TrialFunction(space)

chi = ufl.TestFunction(fvspace)
hT  = ufl.MaxCellEdgeLength(fvspace.cell())

estimator_ufl = hT**2 * div(grad(u)) * chi * dx
estimator = dune.fem.operator.galerkin(estimator_ufl)


estimator(uh, estimate)
```

# Schemes or Dune-FemPy $\neq$ Fenics

A scheme implements the weak form of where the integrands represent the mathematical model:

```
# from before create gridView, space, and discrete function
domain   = dune.grid.cartesianDomain([0,0],[2.5,2.5],[100,100])
gridView = dune.alugrid.aluConformGrid(domain)
space = dune.fem.space.lagrange( gridView, order=1 )
uh   = space.interpolate( [0], name="u" )

# create model, here ufl expression
a = inner(grad(u), grad(v)) * dx

# also work with a == rhs
scheme = dune.fem.scheme.galerkin( a==0, space, solver="cg" )

# solve laplace using cg solver from dune-istl
scheme.solve(target = uh )
```