The Distributed and Unified Numerics Environment (DUNE)

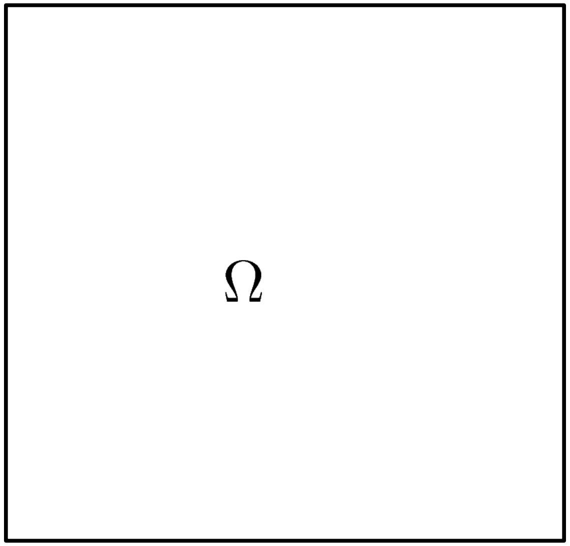
Oliver Sander, Freie Universität Berlin

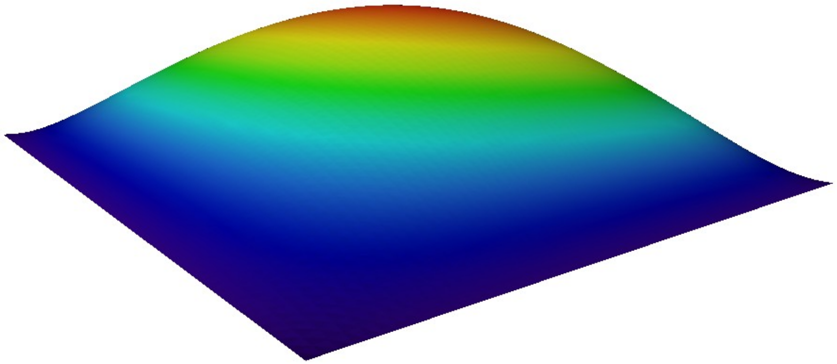
1. 12. 2011, SplineTalks

Partielle Differentialgleichungen

Zum Beispiel die Poisson-Gleichung:

$$-\Delta u = f \qquad \text{bzw.} \qquad -\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

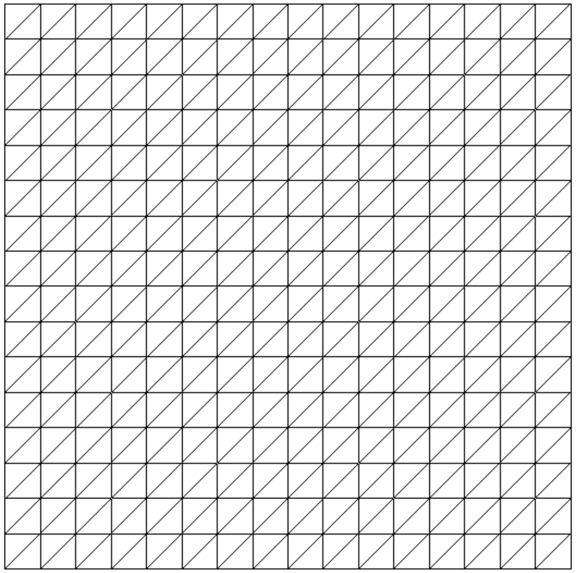


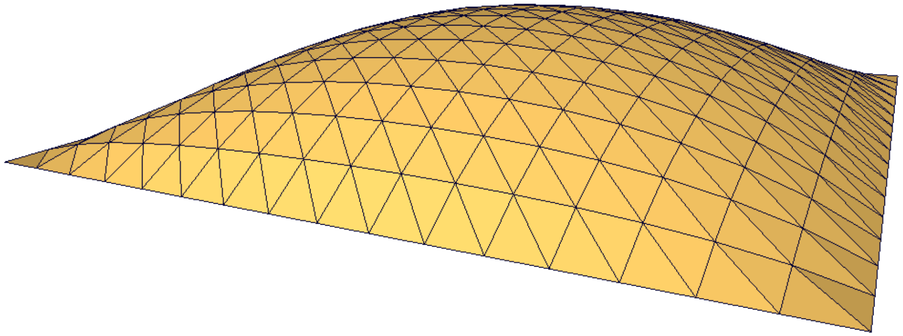


Finite Elemente

Methode zum Lösen von partiellen Differentialgleichungen

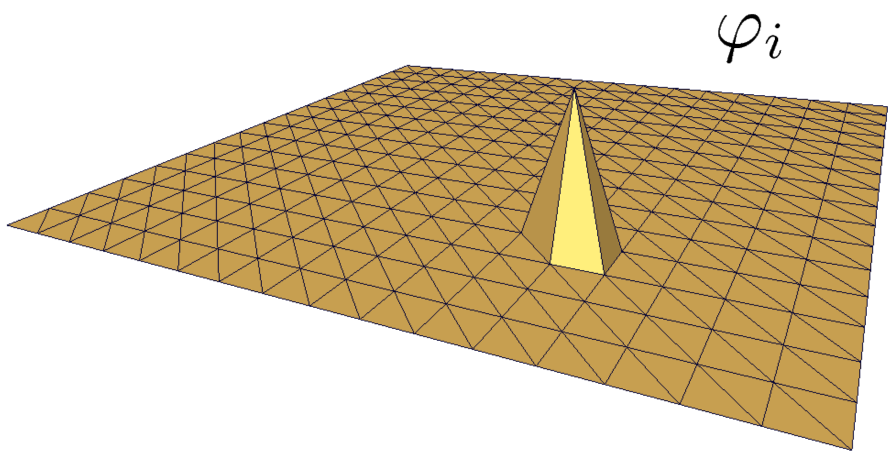
Suche approximative Lösung,
die stückweise affin auf einem gegebenen Gitter ist.





Algebraisches Problem

Knotenbasis:



Lineares Gleichungssystem: $A\bar{u} = b \qquad A \in \mathbb{R}^{n\times n}, \quad b \in \mathbb{R}^n$

$$A_{ij} = \int_\Omega \nabla\varphi_i \nabla\varphi_j \, dx \qquad b_i = \int_\Omega \varphi_i f \, dx$$

A ist dünnbesetzt, aber möglicherweise sehr(!) groß

(bis etwa $n \approx 10^{10}$)

Finite-Elemente-Software

Komponenten von Finite-Elemente-Software:

- Gitterverwaltung
- Lineare Algebra
- Assemblierer
- Löser für (lineare) Gleichungssysteme
- I/O, bzw. Visualisierung

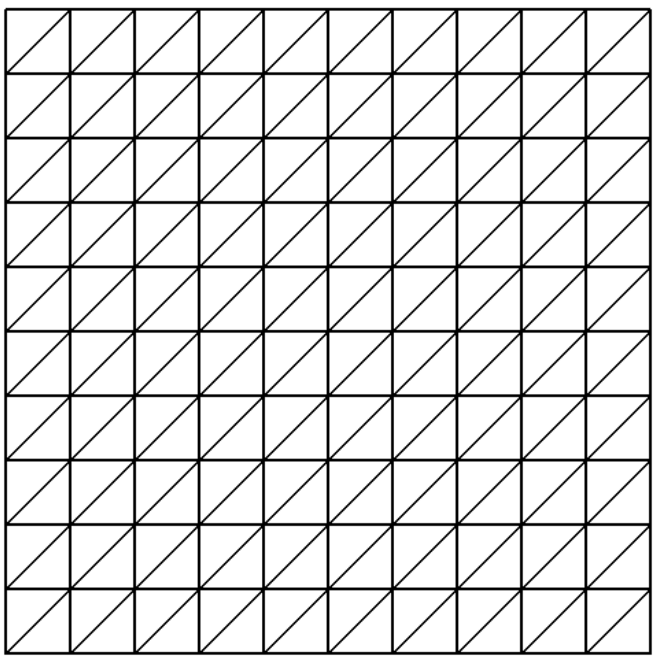
Steuerung über

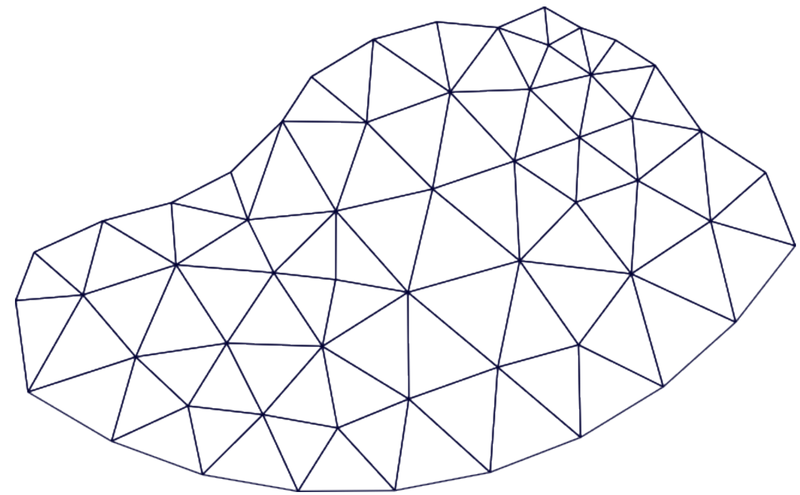
- GUI
- Skriptsprache
- API

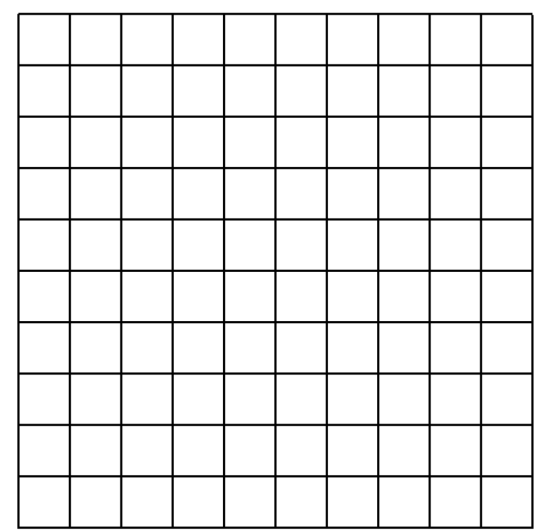
Jeder, der über Finite Elemente forscht, oder sie benutzt, braucht solch eine Software

- z.B. UG, deal II, Alberta, Dune, ...
- kommerzielle Codes, z.B. Ansys, Abaqus
- kleine, handgestrickte Forschungscodes

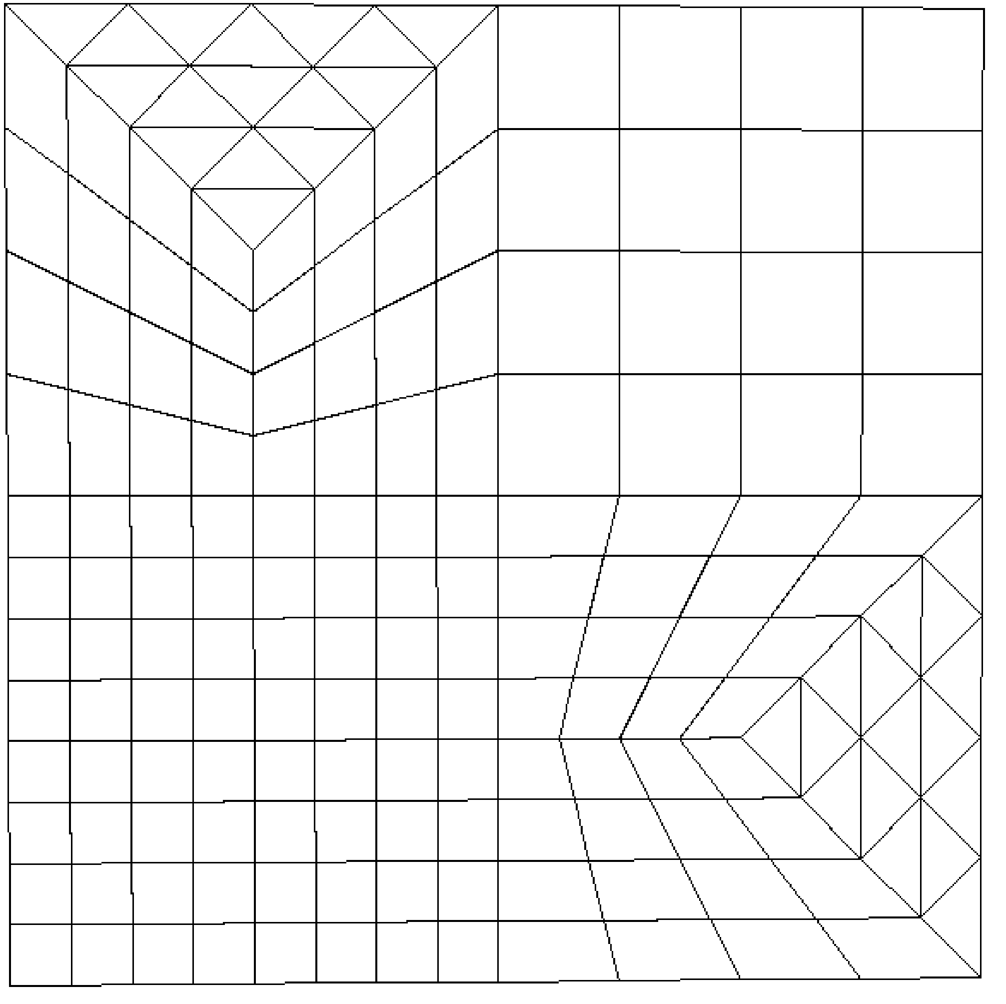
Gitter



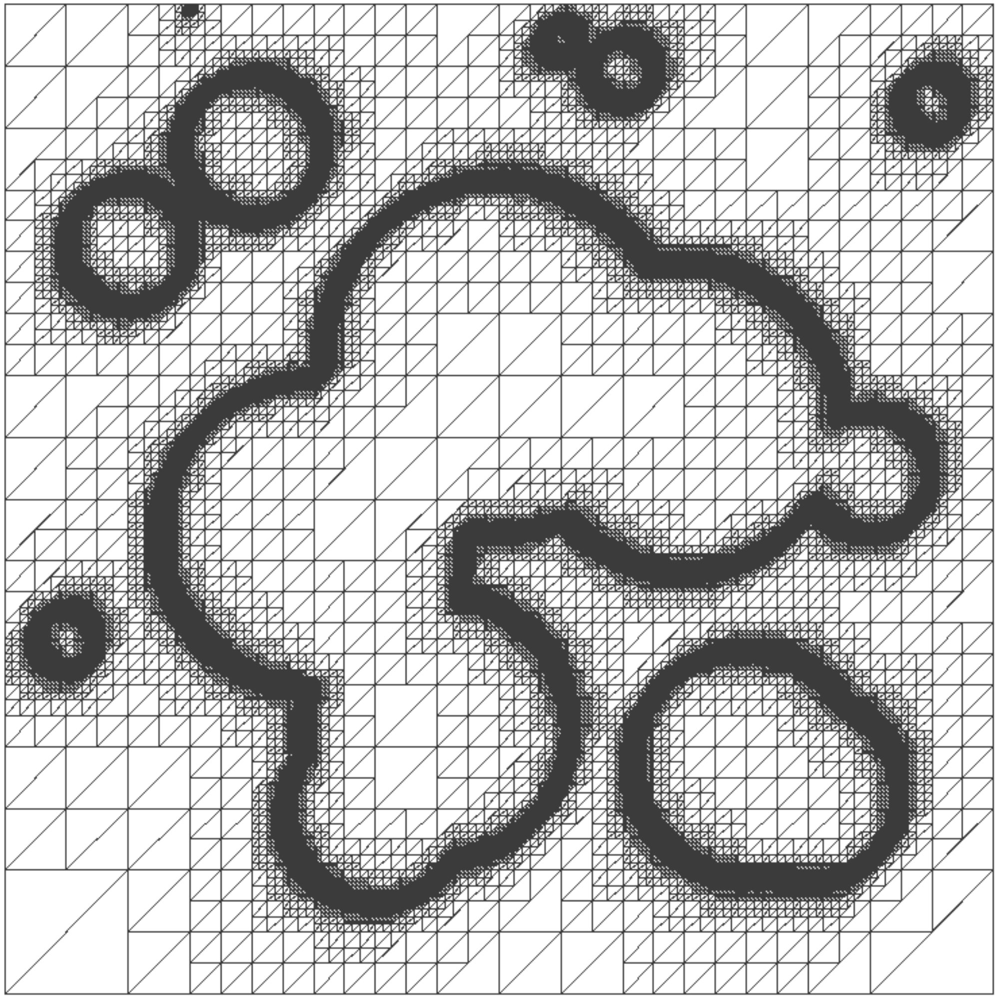




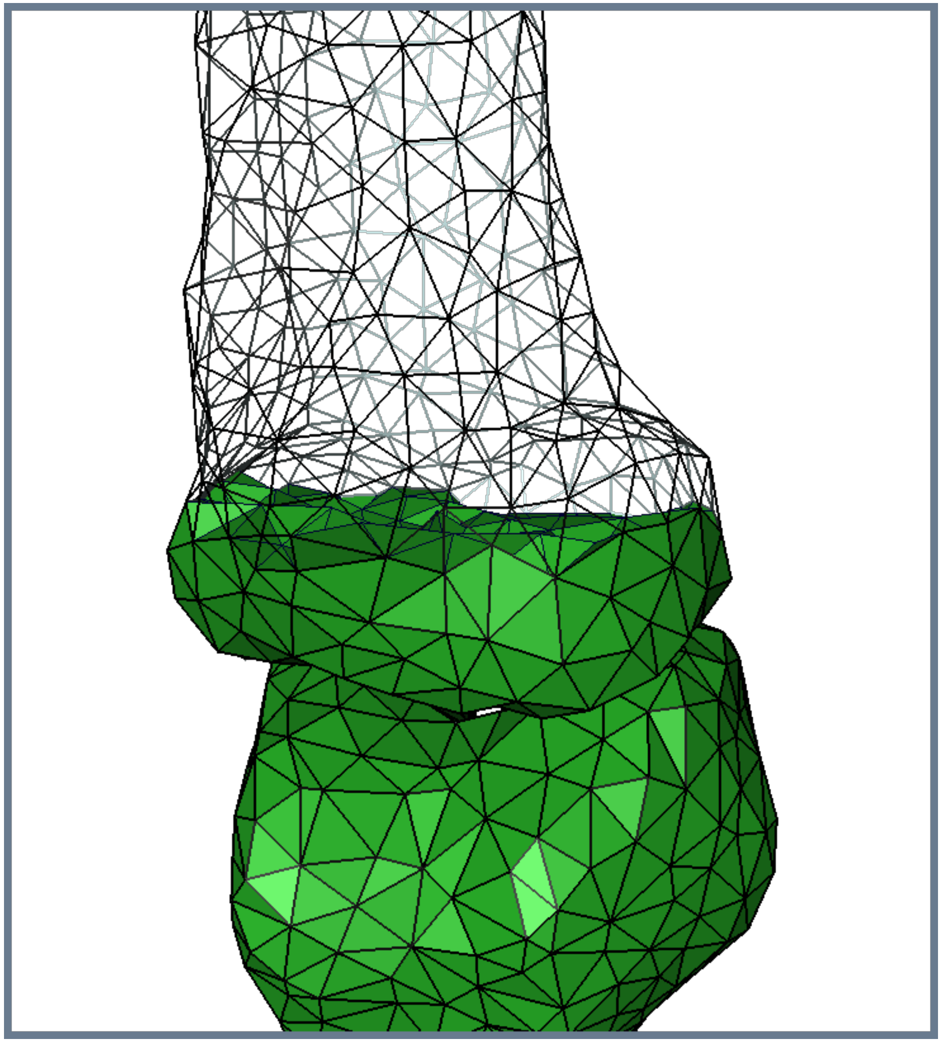
Gitter



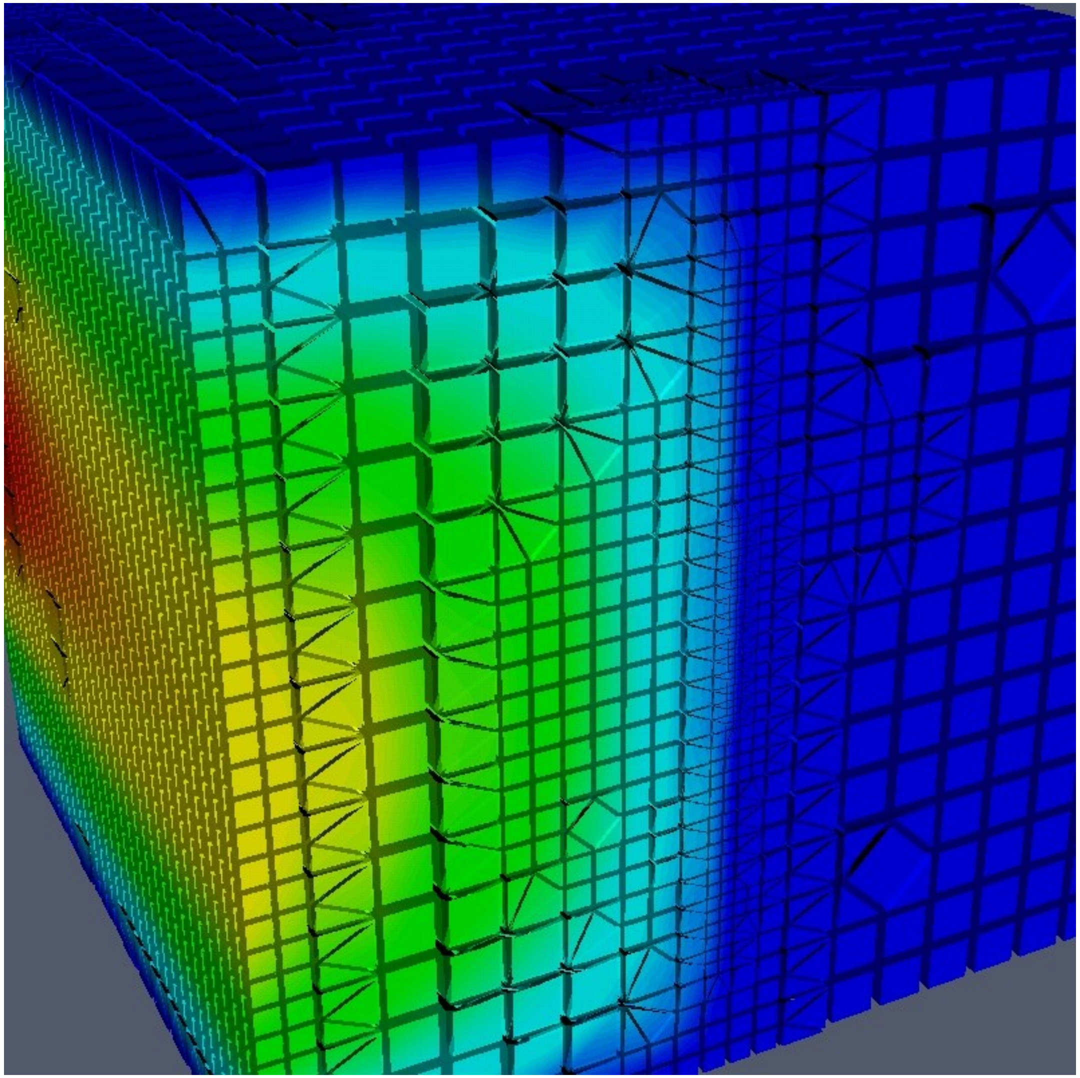
Gitter



Gitter



Gitter



Etwas Geschichte: UG

UG: Unstructured Grids

Entwickelt ab ca. 1995 in der AG von Prof. Wittum
an der Universität Heidelberg

ca. 300.000 Zeilen C-Code

komplett parallelisiert

eigene Skriptsprache (mit selbstgeschriebenem Parser)

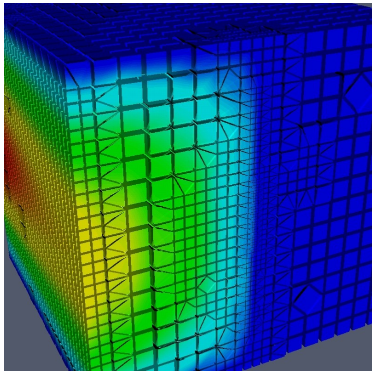
eigene Visualisierung

- sehr flexibel
- sehr portabel
- langsam
- schwer zu bedienen

Einer der erfolgreichsten FE-Forschungscodes

Die Grundidee von Dune

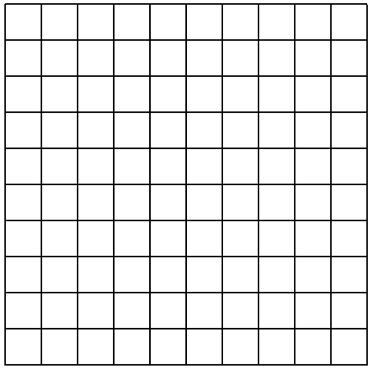
Eine Datenstruktur kann nie alle Nutzer glücklich machen:



- Flexible Implementierungen sind zu langsam
- schnelle Implementierungen sind zu unflexibel

Idee:

trenne Datenstrukturen und Algorithmen durch abstrakte Schnittstellen



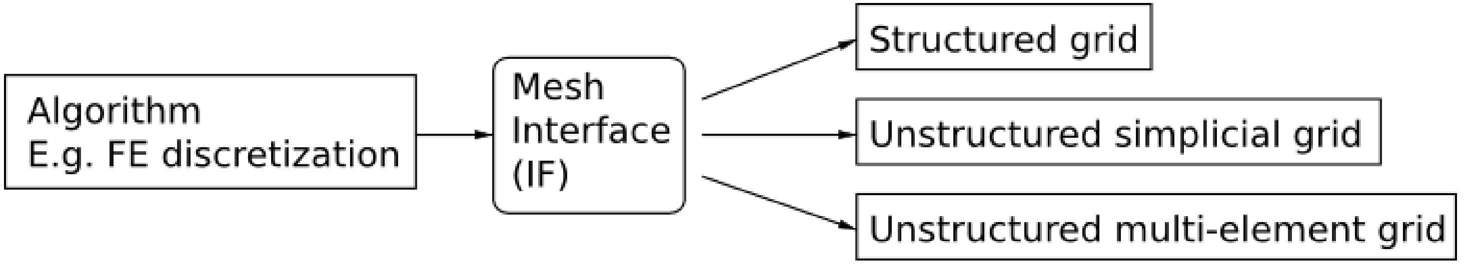
Drei Designziele:

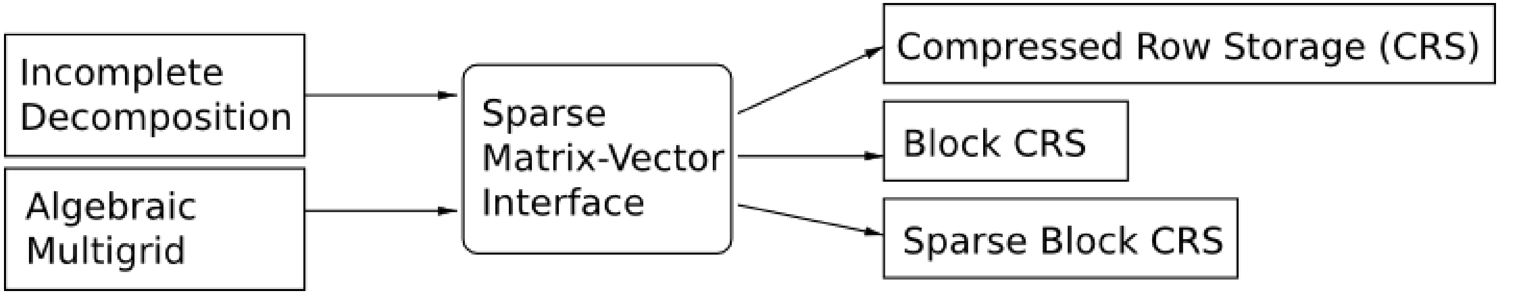
- Flexibilität in der Wahl der Datenstrukturen
- Modularität
- Effizienz

Concept I: Flexibility

Separate data structure and algorithms

- Determine what algorithms require from a data structure (`abstract interface')
- Formulate algorithms based in this interface
- Provide different implementations of the interface





Recycling von existierendem Code

Gittermanager sind teilweise extrem schwierig zu programmieren

- ➔ In UG stecken mehrere Dutzend Mannjahre

In Dune:
biete existierende Codes als Implementierung der Gitterschnittstelle an

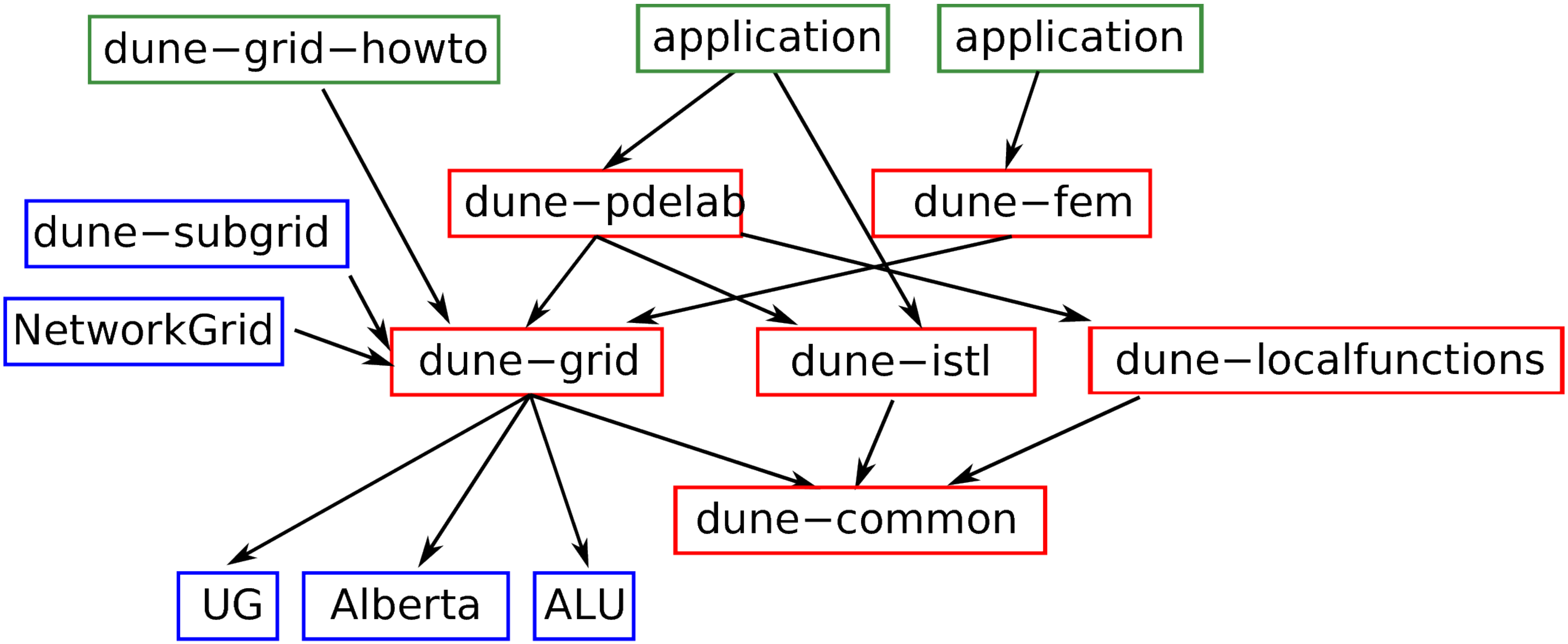
- Zwischenschicht teilweise sehr anspruchsvoll zu programmieren
- Arbeitsersparnis trotzdem immens
- Einfacherer Zugang zu existierenden Codes

Dune bietet

- UG
- Alberta,
- ALUGrid

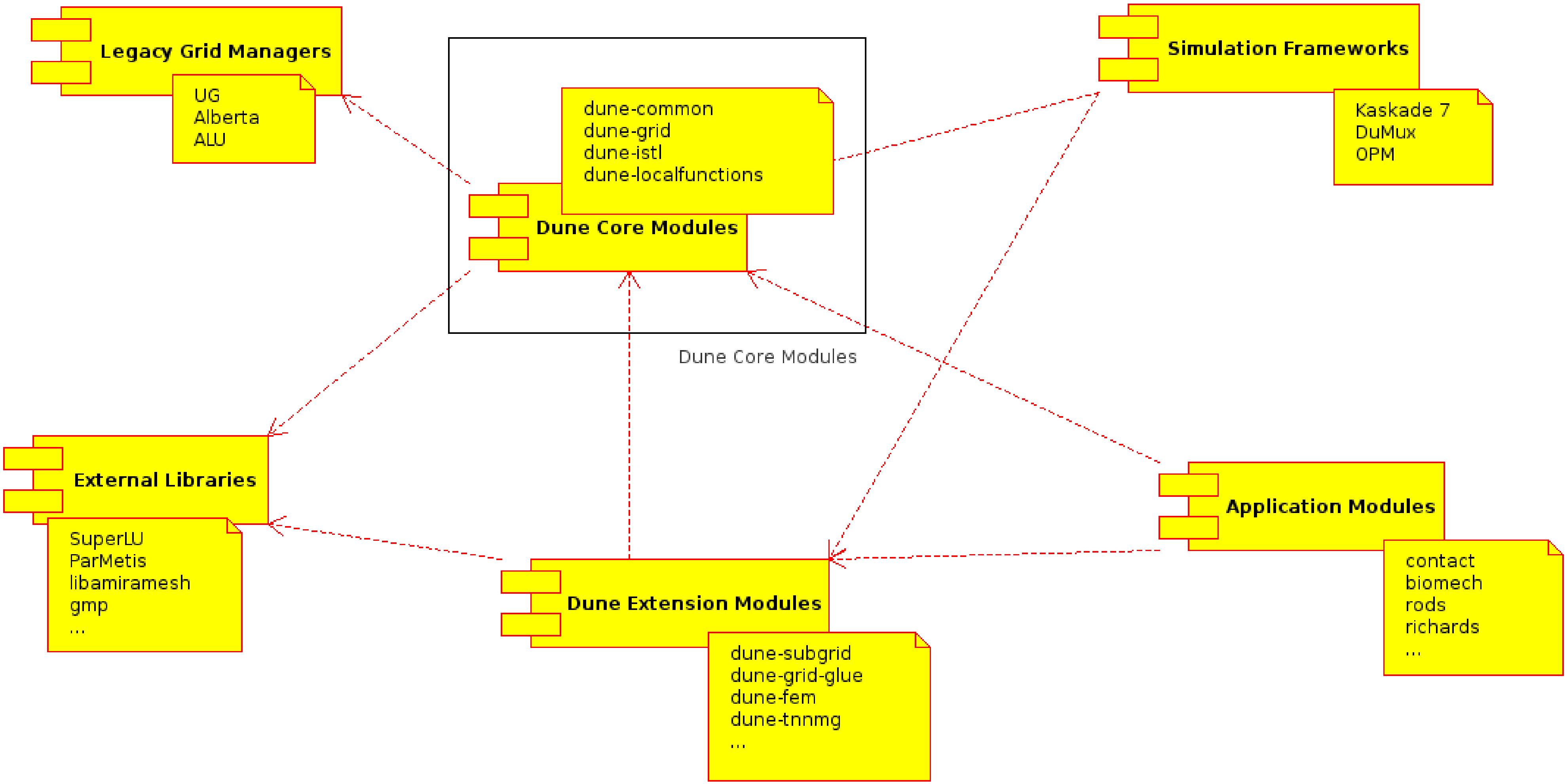
Concept II: Modularity

Dune is divided into modules



Package manager `dunecontrol` tracks and resolves inter-module dependencies

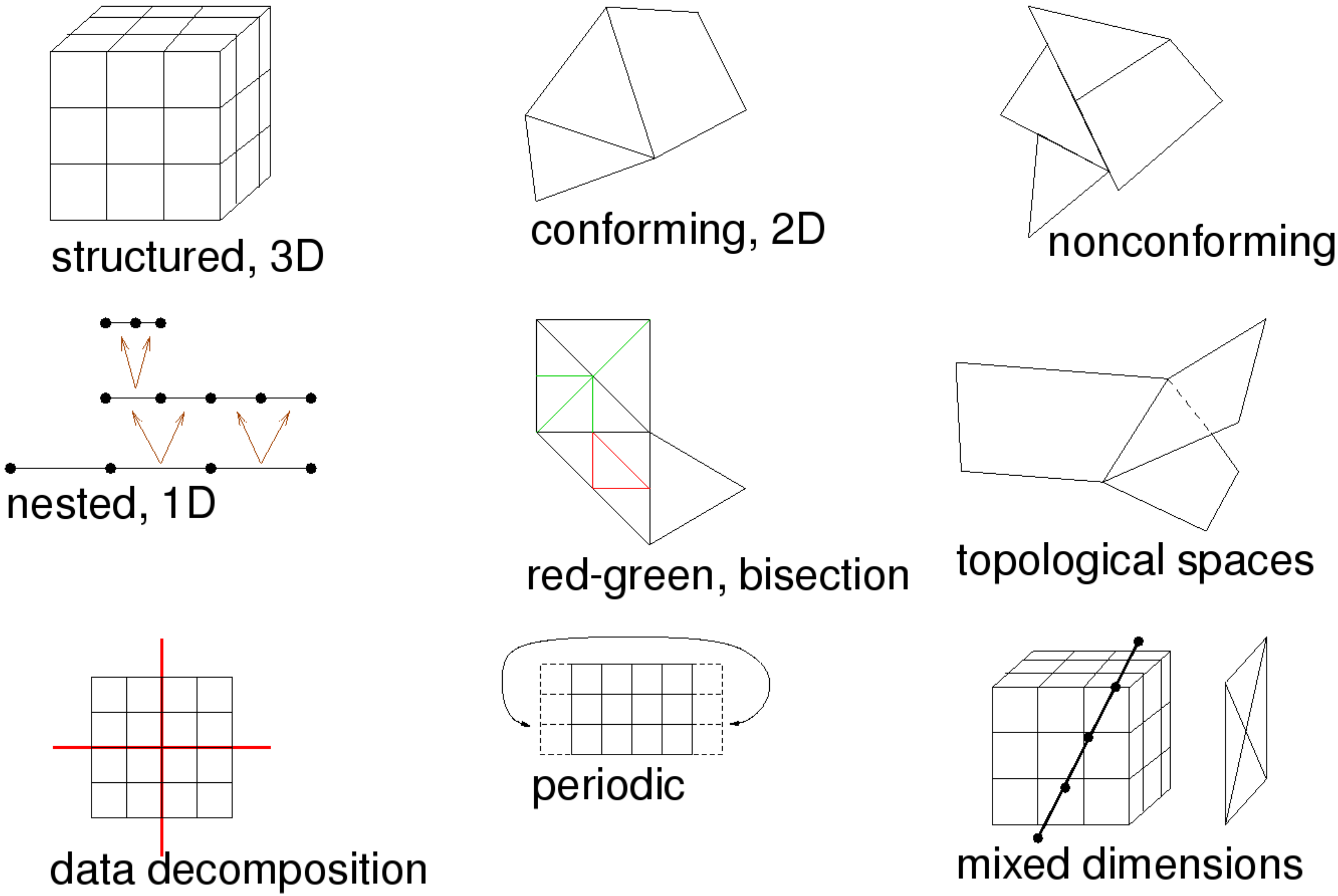
Concept II: Modularity



Concept III: Efficiency

kommt später

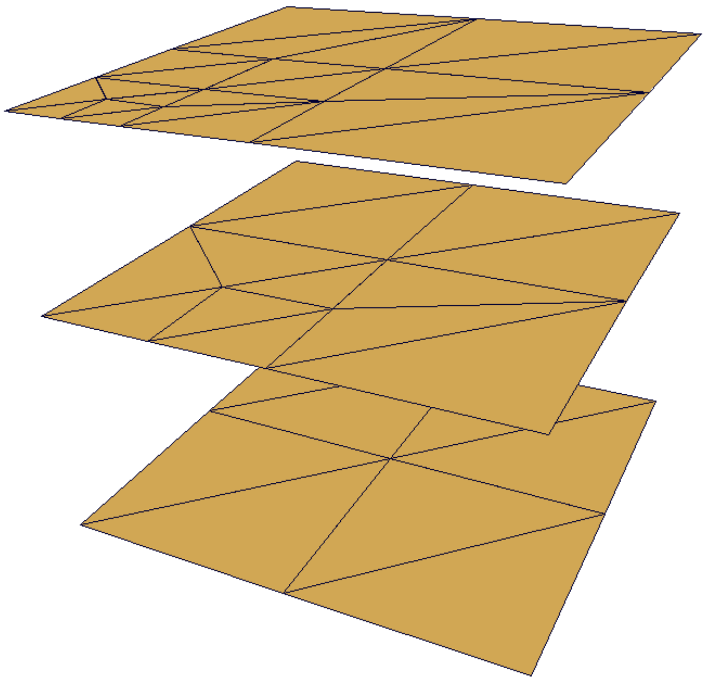
Scope of the Grid Interface



Formal Definition of a Grid

Grids in the DUNE sense are hierarchical!

A hierarchical grid consists of three things:



- A set of entity complexes

$$\mathcal{E} = \{E_0, \dots, E_k\}$$

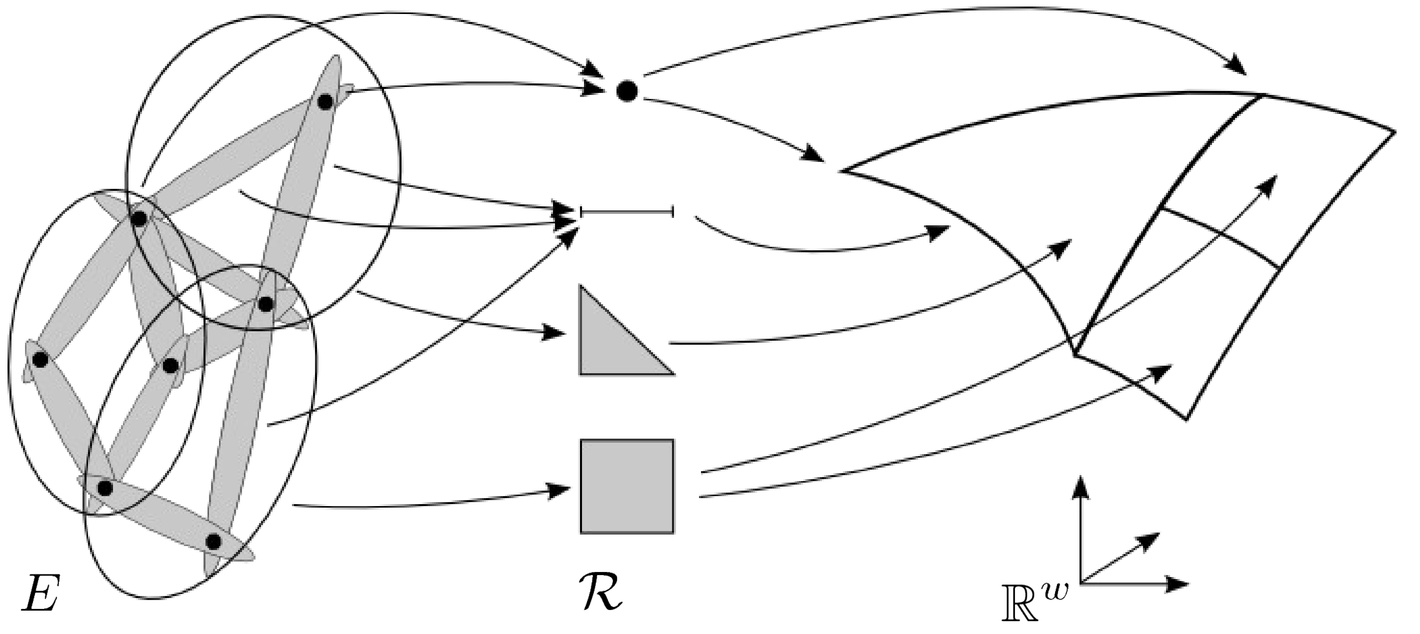
- A set of geometric realizations

$$\mathcal{M} = \{M_0, \dots, M_k\}$$

- A set of father relations

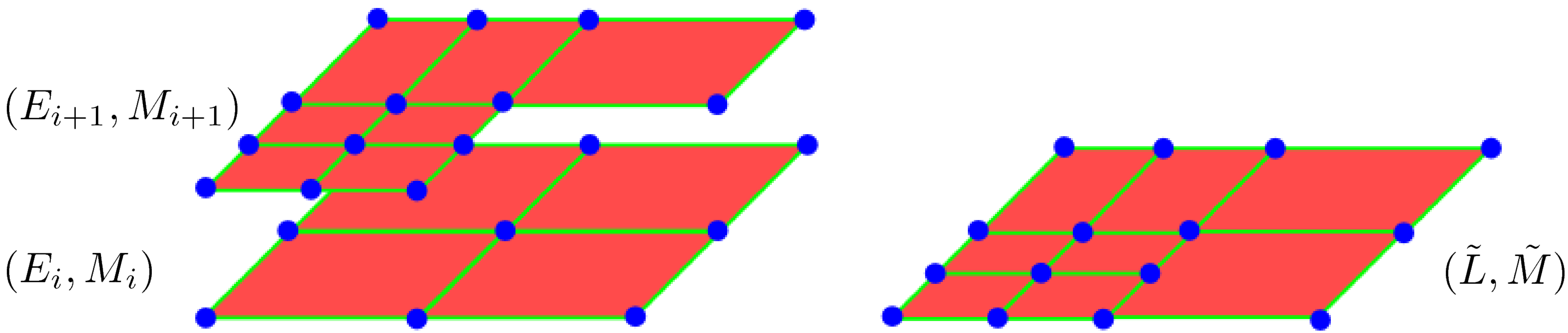
$$\mathcal{F} = \{F_0, \dots, F_{k-1}\}$$

Entity Complexes and Geometric Realizations



- Entity complex: set system of entities, topological information
- Reference elements: classify entities
- Geometric realization: maps from the reference elements into Euclidean space

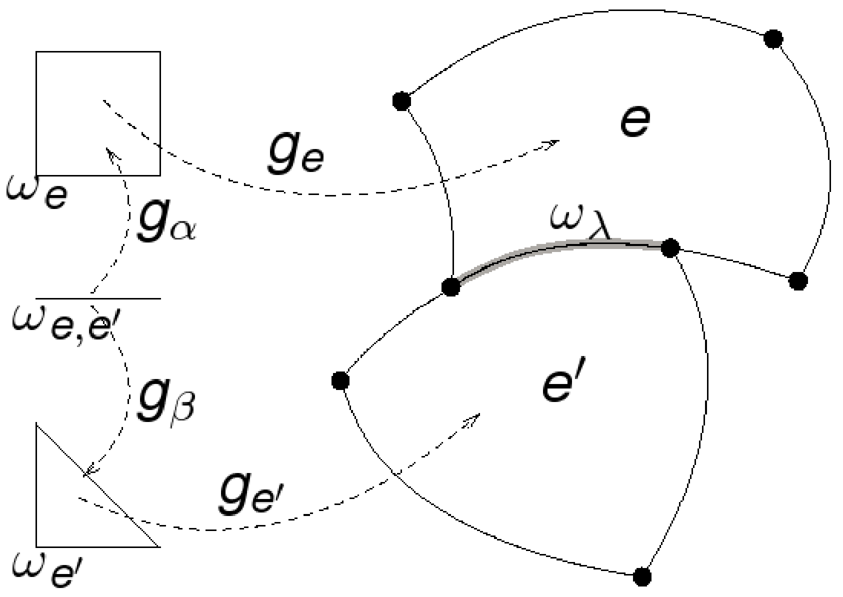
Father Relation



Hierarchical grid　　　　Leaf grid

- Connect two level grids with a father relation
- Only element father relation appears in the interface
- Leaf entities constitute the leaf grid

Intersections



- An d-1 dimensional point set shared by two elements.
- Described by transformations from a reference element
- Arbitrary nonconforming intersections can be handled.
- Leaf- and level-wise intersections

- Intersections with the domain boundary and the processor boundary

Implementation

- Mathematical definitions translate directly into C++ classes
- Implementations using static polymorphism
- Access to entities by STL-style iterators:
  `LevelIterator`, `LeafIterator`, `HierarchicIterator`,
  `IntersectionIterator`
- Arbitrary sets of grids can coexist in the same application
- Many templates, but few really evil tricks
- GNU AutoTools build system for each module
- Special package manager tracks inter-module dependencies
- Runs on most flavours of Unix
- Licence: LGPL + linking exception
- Surprisingly easy to use!

Grid Implementations

The following grid implementations are currently available:

- Dedicated Dune grid implementations:
  - `YaspGrid`, `SGrid`: structured grids
  - `OneDGrid`: fully adaptive one-dimensional grid
  - `NetworkGrid`: network of 1d grids in a 3d world
  - `CpGrid`: corner-point grid [from Rasmussen et al., Sintef]
- Legacy grid managers:
  - `UG`, `Alberta`, `ALUGrid`
- Meta grids:
  - `SubGrid`: select element subset and treat it like a new grid
  - `GeometryGrid`: supply grid with a new geometry
  - `PrismGrid`: turn any grid into a prism grid of one dimension higher
- Example implementation:
  - `IdentityGrid`

Code Example: Grid Creation

Create a structured grid:

```
const int dim =3;
typedef Dune :: SGrid < dim , dim > GridType;
Dune :: FieldVector < int , dim > N (3);
Dune :: FieldVector < GridType :: ctype , dim > L (-1.0);
Dune :: FieldVector < GridType :: ctype , dim > H ( 1.0);
GridType grid (N, L, H);
```

Create a UGGrid from a gmsh file:

```
const int dim =3;
typedef Dune :: UGGrid < dim > GridType;
GridType grid;
Dune :: GmshMeshReader<GridType>::read(grid, "filename");
```

For unstructured grid: general interface for grid creation

Code Example: Grid Traversal

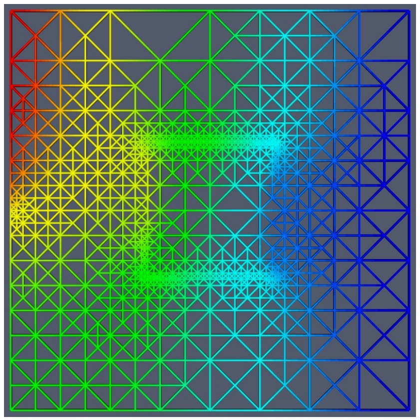
Iterate over all elements on the leaf grid

```
typedef GridType :: Codim <0>:: LeafIterator ElementLeafIterator;

for ( ElementLeafIterator it = grid . template leafbegin <0>();
       it != grid . template leafend <0>(); ++it )
  {
     std :: cout << " visiting element which is a " << it -> type ()
                 << std :: endl ;
  }
```

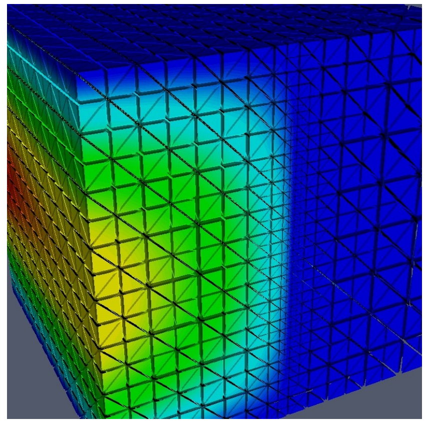
Iterate over all vertices on the leaf grid

```
typedef GridType :: Codim <dim> :: LeafIterator VertexLeafIterator;

for ( VertexLeafIterator it = grid . template leafbegin <dim>();
       it != grid . template leafend <dim>(); ++it )
  {
     std :: cout << " visiting vertex at " << it -> geometry () .corner(0)
                 << std :: endl;
  }
```

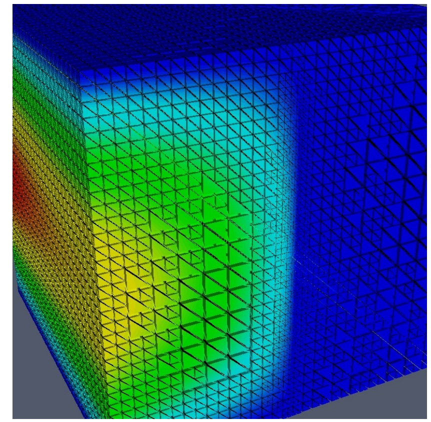
Example: Poisson Problem



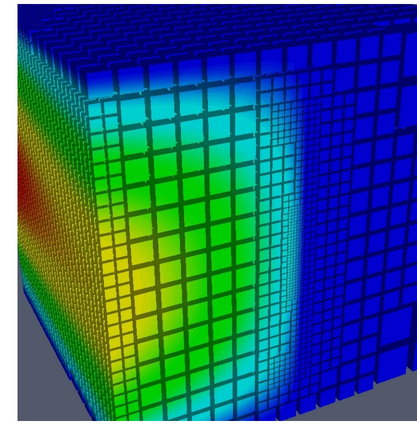
AlbertaGrid, 2d



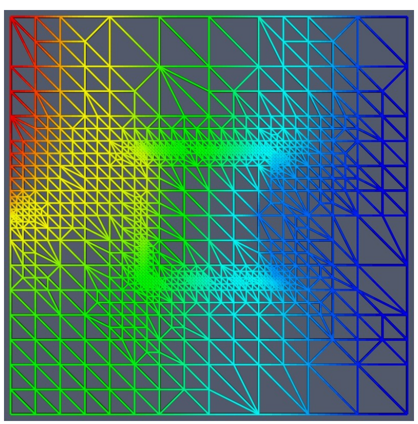
AlbertaGrid, 3d



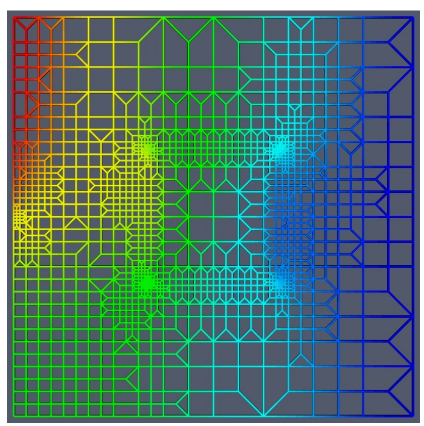
AluSimplexGrid, 3d



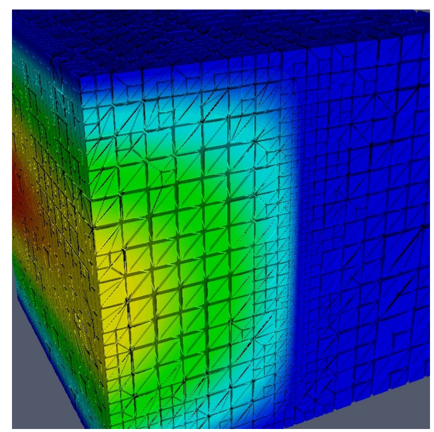
AluCubeGrid, 3d



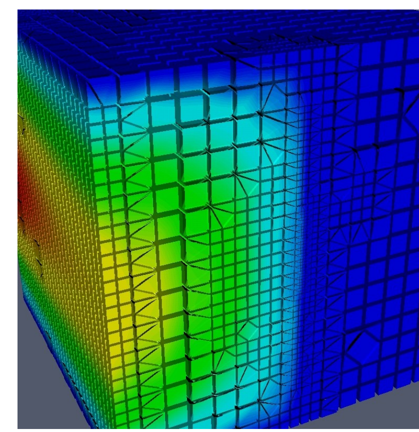
UGGrid, 2d, simplices



UGGrid, 2d, cubes

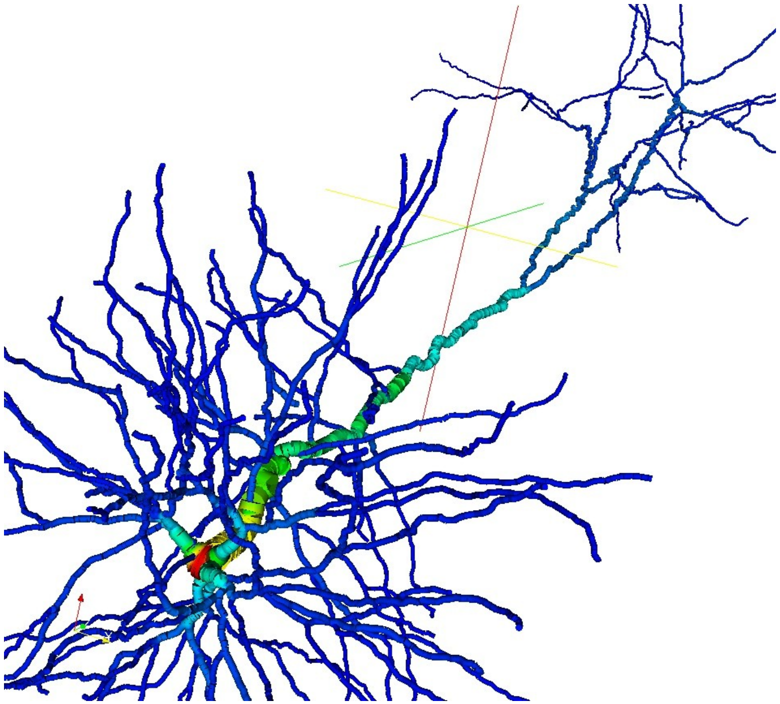


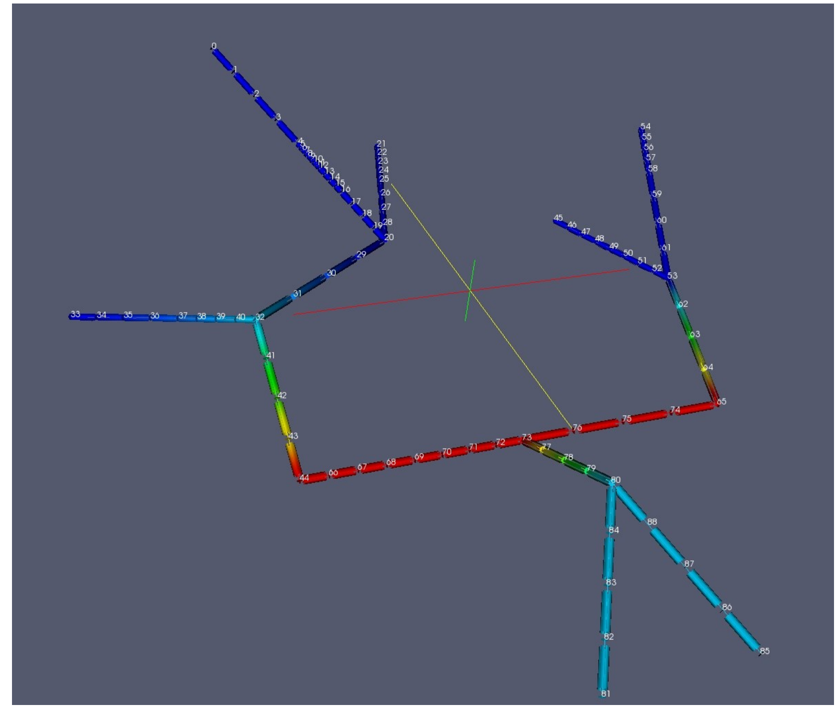
UGGrid, 3d, simplices



UGGrid, 3d, cubes

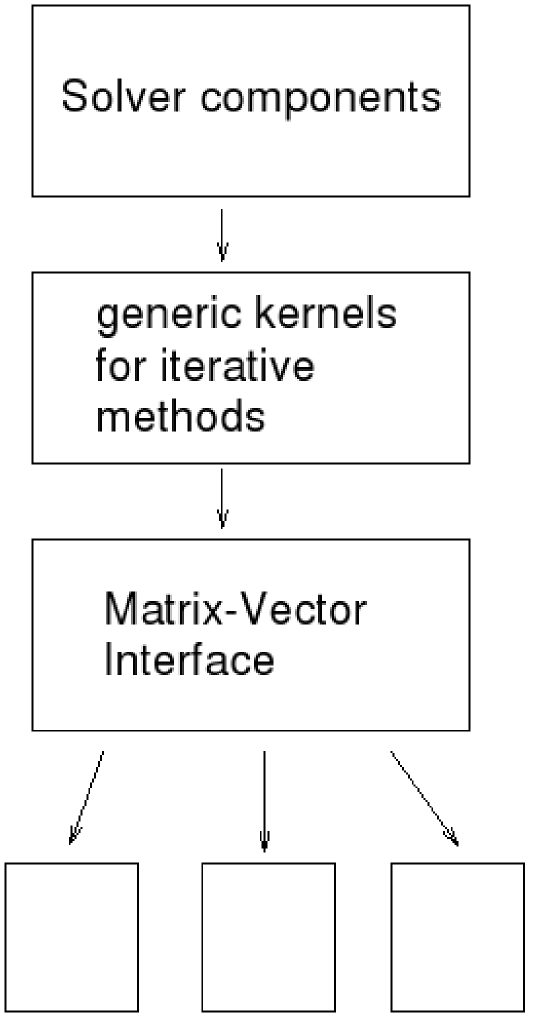
Example: NetworkGrid





- Dendritic tree of L5 B pyramidal neuron (reconstruction by Christiaan de Kock, MPIMF, Heidelberg)
- NetworkGrid simulator (Stefan Lang, Olaf Ippisch)

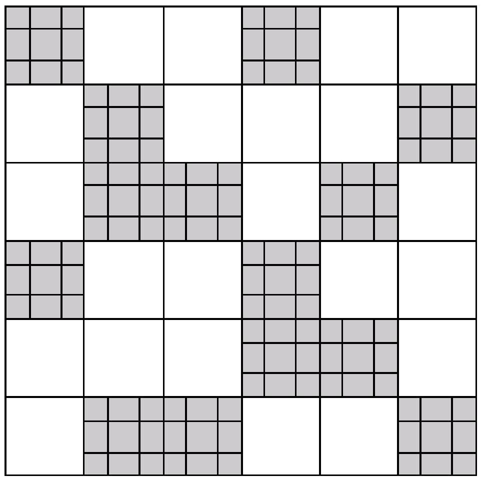
Linear Algebra: dune-istl



- There are already template libraries for linear algebra: MTL/ITL
- Existing libraries cannot efficiently use (small) structure of FE-Matrices
- Solver components: Based on operator concept, Krylov methods, (A)MG preconditioners
- Generic kernels: Triangular solves, Gauß-Seidel step, ILU decomposition
- Matrix-Vector Interface: Support recursively block structured matrices
- Various implementations of the interface are available

dune-istl is completely independent of dune-grid!

Block Structure in FE Matrices

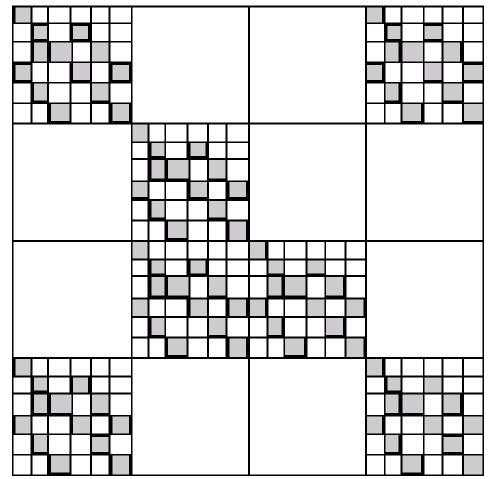


sparse block matrix

blocks are dense

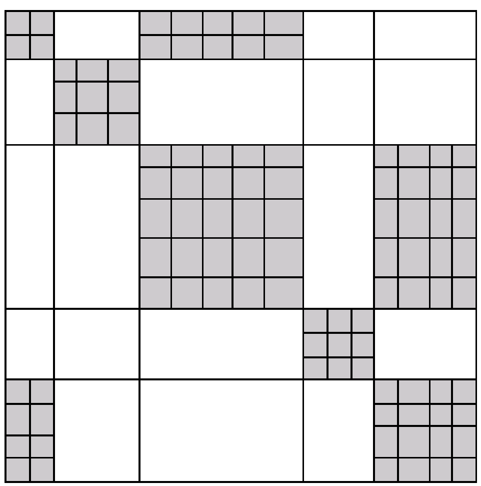
blocks have fixed size

DG fixed p



blocks are sparse

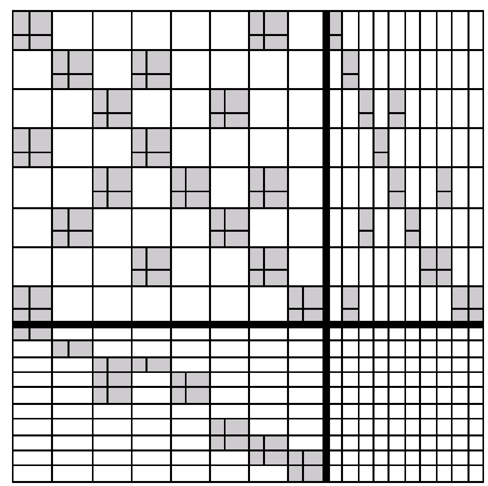
diffusion-reaction systems



blocks are dense

blocks have variable size

DG hp version



2x2 block matrix

each block is sparse

Taylor-Hood elements

Example Definitions

- A vector containing 20 blocks where each block contains two complex numbers using **double** for each component:

```
typedef FieldVector<complex<double>,2> MyBlock;
BlockVector<MyBlock> x(20);
x[3][1] = complex<double>(1,-1);
```

- A sparse matrix consisting of sparse matrices having scalar entries:

```
typedef FieldMatrix<double,1,1> DenseBlock;
typedef BCRSMatrix<DenseBlock> SparseBlock;
typedef BCRSMatrix<SparseBlock> Matrix;
Matrix A(10,10,40,Matrix::row_wise);
... // fill matrix
A[1][1][3][4][0][0] = 3.14;
```

Vector and Matrix Interface

Mainly taken from sparse BLAS

- Vector
  - Is a one-dimensional container
  - Sequential access
  - Random access
  - Vector space operations: Addition, scaling
  - Scalar product
  - Various norms
  - Sizes
- Matrix
  - Is a two-dimensional container
  - Sequential access using iterators
  - Random access
  - Organization is row-wise
  - Mappings $y = y + Ax$; $y = y + A^T x$; $y = y + A^H x$;
  - Solve, inverse, left multiplication
  - Various norms
  - Sizes

The need for speed

FE-Probleme können sehr groß werden

Zeitaufwand:

- implizite Verfahren:
  Lösen des linearen Gleichungssystems

$$A\bar{u} = b$$

- explizite Verfahren:
  Aufstellen der Matrix, Gitterverfeinerung

$$\bar{u}^{k+1} = \bar{u}^k + (b - A\bar{u}^k)$$

Deshalb:

- optimale Algorithmen und Datenstrukturen
- wissen, was man tut
- statischer Polymorphismus

Was kostet die Dune-Schnittstelle?

Für große Probleme: parallele Architekturen

- Shared-memory-Maschinen
- Verteiltes Rechnen

Dynamischer vs. statischer Polymorphismus

Dynamischer Polymorphismus

```
class GridBase
{
   virtual int dimension() = 0;
}

class My3dGrid : public GridBase
{
   virtual int dimension() {return 3;}
}

GridBase* myGrid = new My3dGrid …

int gridDim = myGrid->dimension();
```

Die Lehrbuchvariante, aber...

Langsam:

- Overhead durch Funktionsaufruf
- Pipeline stoppt wegen bedingtem Sprung

Dynamischer vs. statischer Polymorphismus

Statischer Polymorphismus

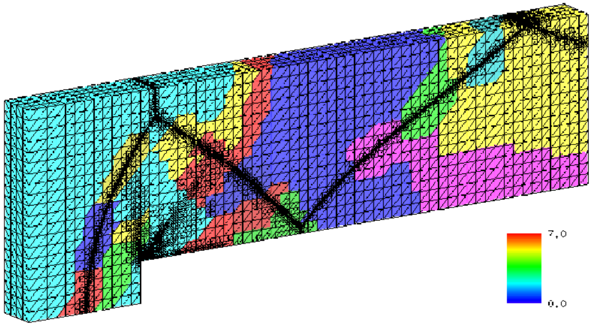
```
template <class GridImplementation>
class GridInterface
{
   int dimension() {return impl_.dimension();}

private:
   GridImplementation impl_;
}

class My3dGridImp
{
   int dimension() {return 3;}
}

GridInterface<My3dGridImp>* myGrid = new GridInterface<My3dGridImp> …

int gridDim = myGrid->dimension();
```

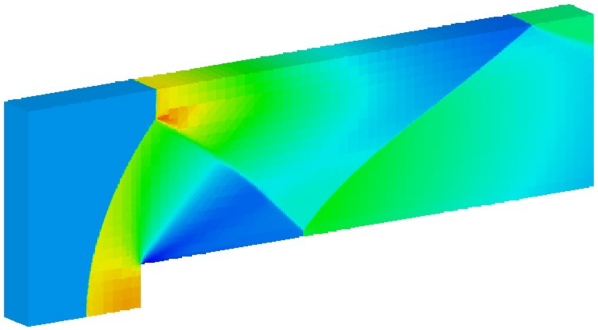
Zur Übersetzungszeit bekannt, welche Methode aufgerufen wird

- Kein bedingter Sprung
- Overhead des Funktionsaufrufs kann entfernt werden (Inlining)

Was kostet die Gitterschnittstelle?

ALUGrid direct vs. ALUGrid through DUNE



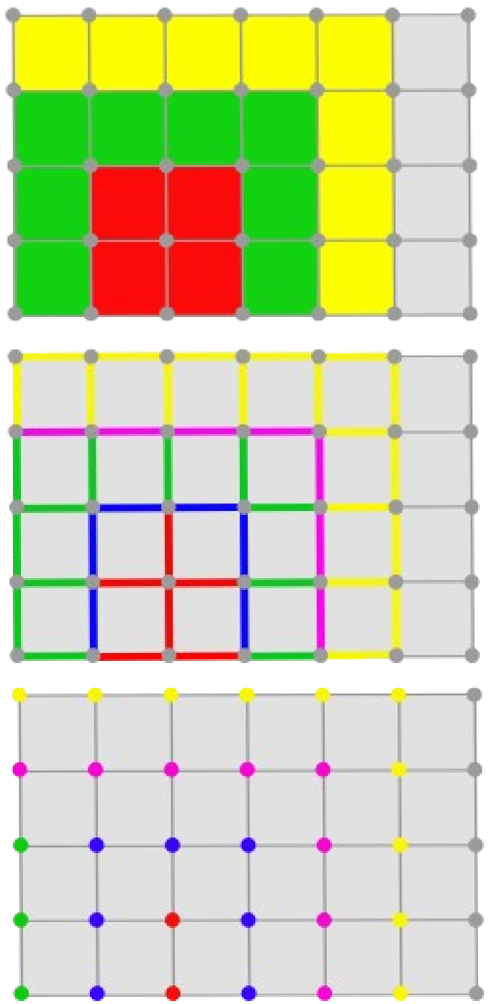


compressible Euler equations

P	flux	evolve	adapt.	total
4	7.8	-5.0	9.3	12
8	7,5	-5.0	9.2	12
16	6.9	-5.0	9.2	11
32	4.9	-5.0	9.1	9

relative performance loss [%]

Parallel Data Decomposition

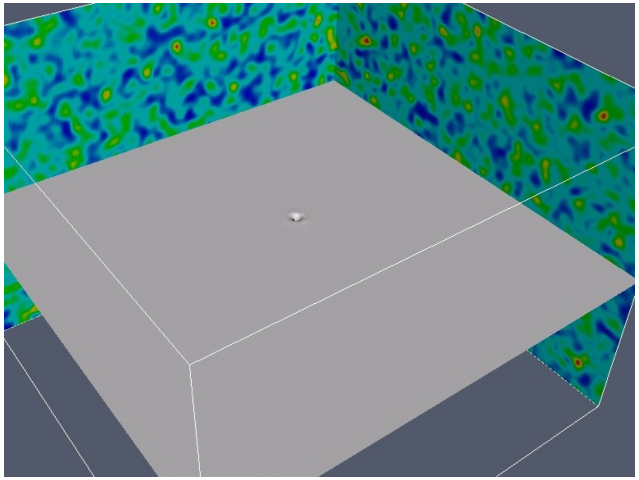


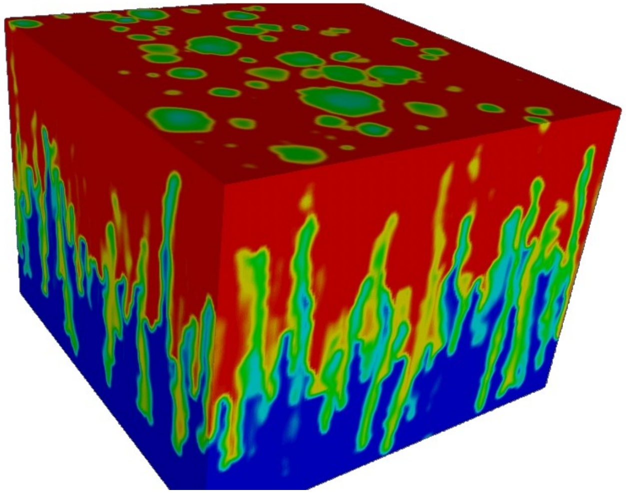
- Grid is mapped to $\mathcal{P} = \{0, \dots, P-1\}$.
- $E = \bigcup_{p \in \mathcal{P}} E|_p$ possibly overlapping.
- $\pi_p : E|_p \to$ "partition type".
- For codimension 0 there are three partition types:
  - *interior*: Nonoverlapping decomposition.
  - *overlap*: Arbitrary size.
  - *ghost*: Rest.
- For codimension > 0 there are two additional types:
  - *border*: Boundary of interior.
  - *front*: Boundary of interior+overlap.

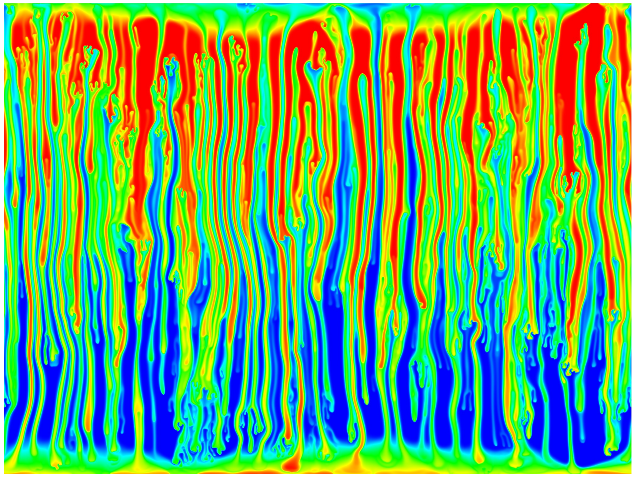
- Grid implementations organize communication and load balancing

Example: Parallel Computing

Density-driven flow (P. Bastian)







- cell-centered finite volume scheme
- YaspGrid, 8e8 cells, 384 processors
- 9000 timesteps, 3 days running time

Aktuell in Arbeit:
Rechnungen auf JUGENE (Jülich),
294.912 Prozessoren

Dune: Organisation

Gestartet: 2002 von Peter Bastian, mit Mario Ohlberger und Martin Rumpf

Entwickelnde Gruppen:
Berlin, Heidelberg, Münster, Freiburg, Warwick

- Homepage in Heidelberg
- User-Wiki in Münster

Mir bekannte Nutzer:

- Aachen, Basel, Bergen, Berlin, Erlangen, Freiburg, Graz, Heidelberg, Kaiserslautern, Münster, Nizza, Stuttgart, Oslo, Zürich, ...

Kommerzielle Nutzer: StatoilHydro, (Totalfina?)

- jährliches Entwicklertreffen
- jährlicher Dune-Kurs in Heidelberg
- Okt. 2010: erstes Dune-Usertreffen

http://www.dune-project.org

Projekte

Projekte ohne Mathe

- Debian-Paketierung
- Bindings an andere Programmiersprachen
  - z.B. Python, D, Matlab, ...
- Statisches Testen
- Verbesserungen am Buildsystem
- Mehr Dateiformate für Gitter-I/O
  - z.B. LGM
- Fehlende Features in UGGrid
  - Backup/Restore
  - Kommunikation auf Kanten und Seiten
  - Dynamische Lastverteilung

Projekte mit Mathe

Viele!