Atlas, a library for NWP and climate modelling

5th ENES HPC Workshop
17-18 May 2018, Lecce

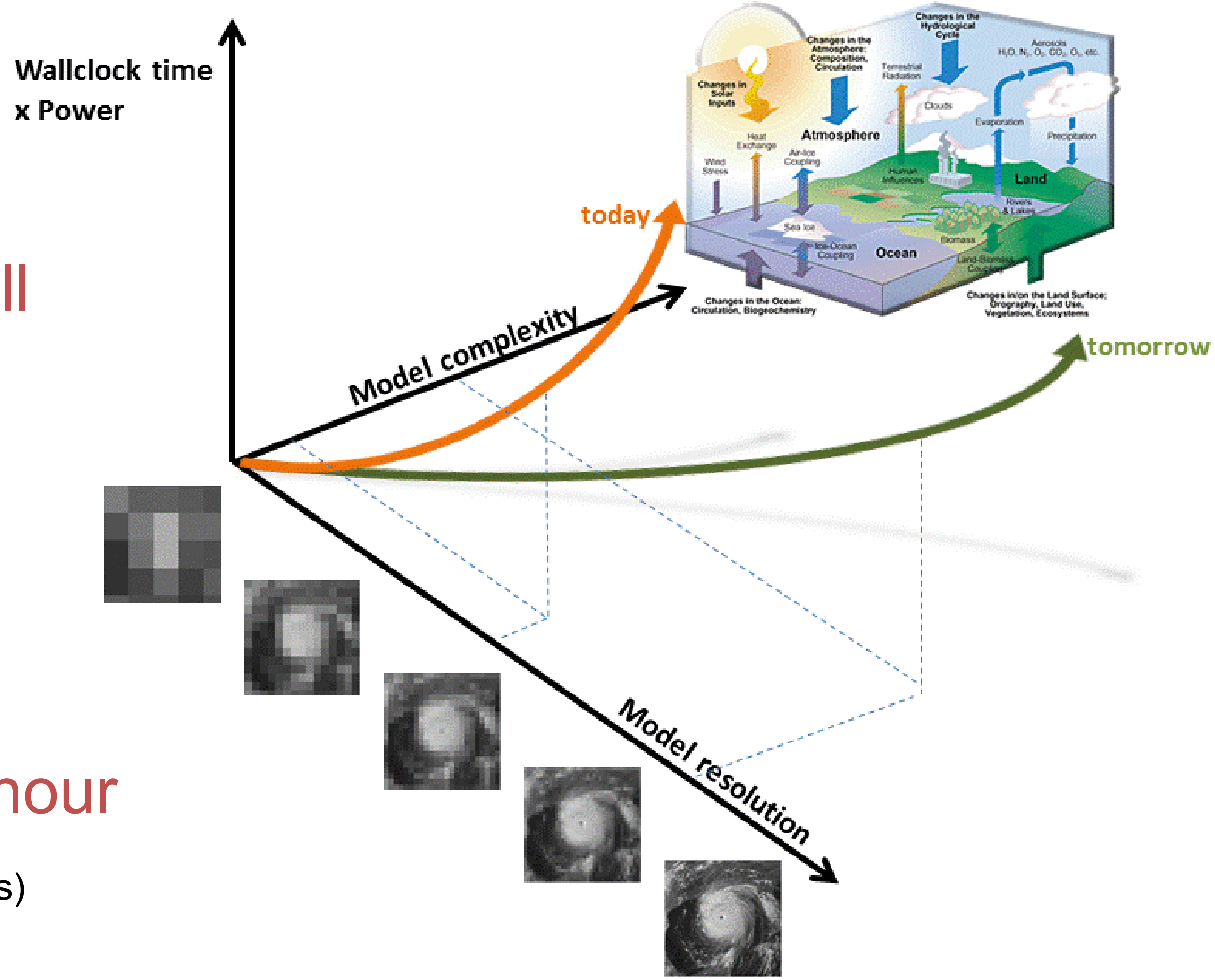
By Willem Deconinck
willem.deconinck@ecmwf.int

ECMWF

© ECMWF maggio 18, 2018

Growth in model resolution and model complexity

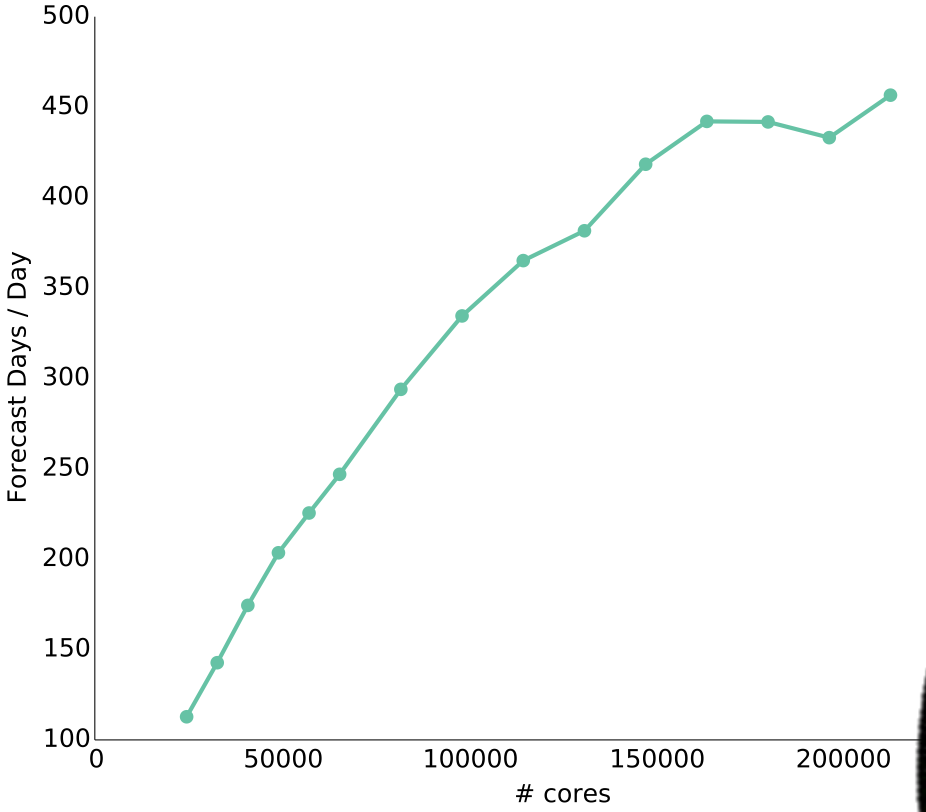
Improving forecast skill

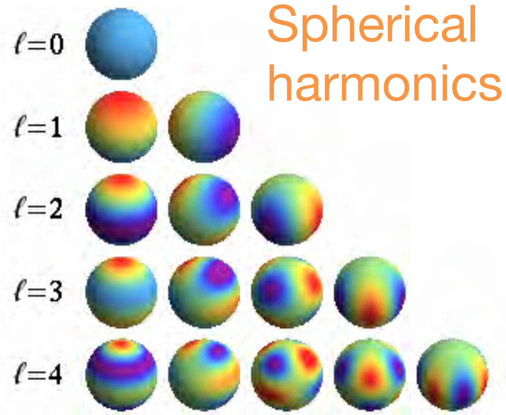


Limited by ability to forecast 10 days in 1 hour

(currently with 360 Cray XC40 nodes)

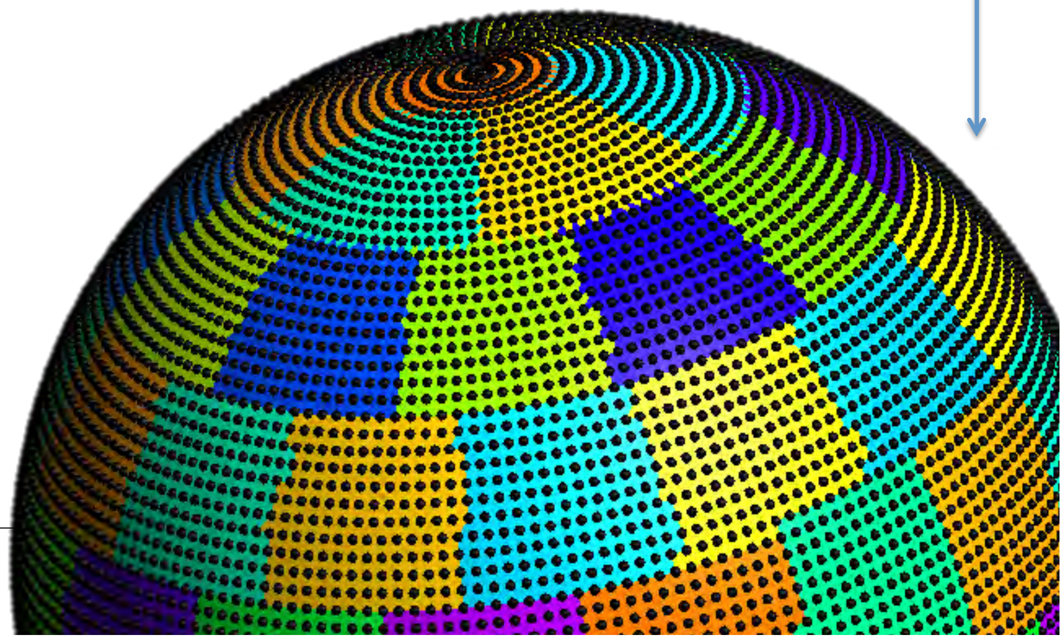
IFS faces a (long-term) scalability problem





transform

global all-to-all communications



Atlas, a library for NWP and climate modelling – *Deconinck et al. 2017, J-CPC*

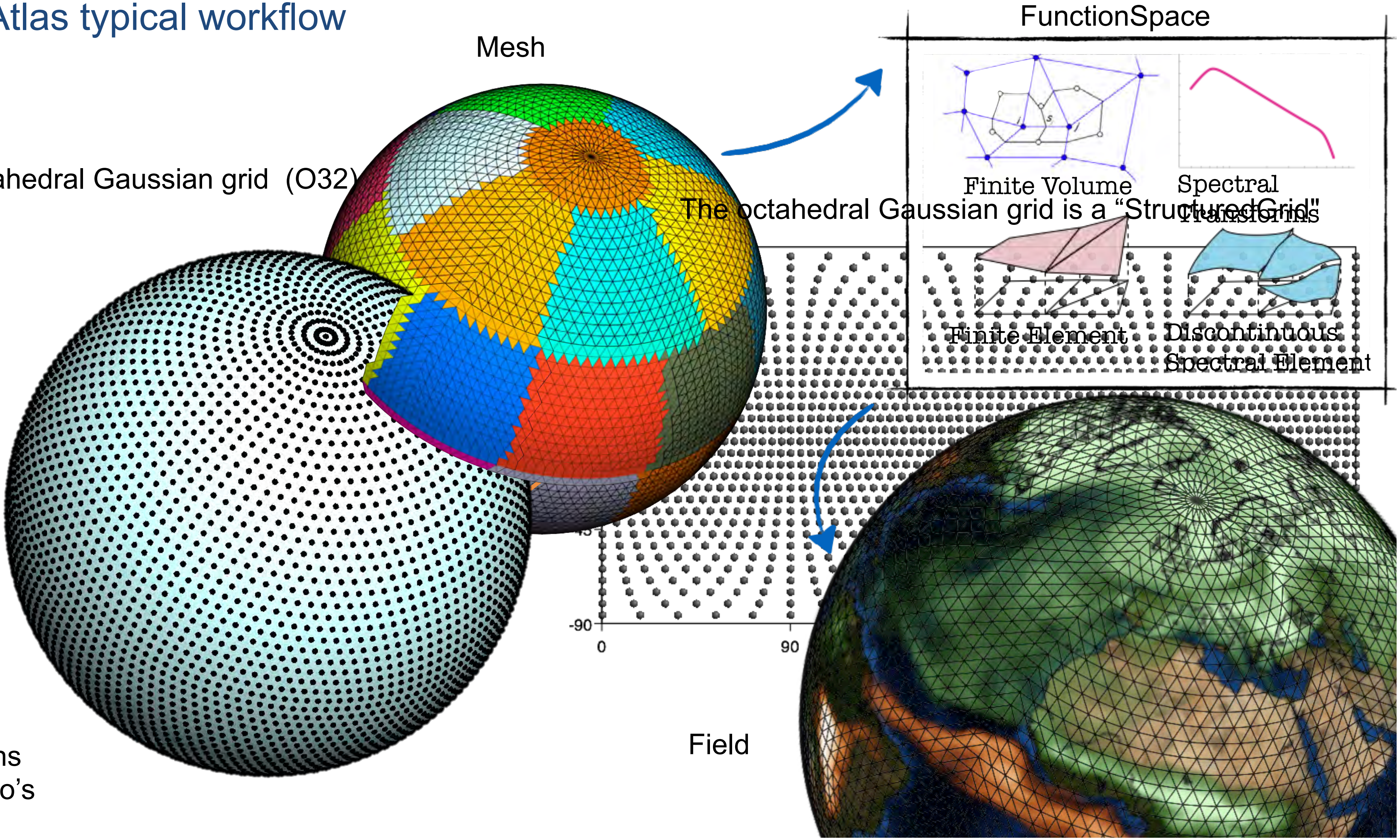
- A new foundation built with future challenges for HPC in mind
- Modern C++ library implementation with modern Fortran 2008 (OOP) interfaces → integration in existing models
- Open-source (Apache 2.0), http://github.com/ecmwf/atlas
- Data structures to enable new numerical algorithms, e.g. based on unstructured meshes
- Separation of concerns:
  - Parallelisation
  - Accelerator-awareness (GPU/CPU/…)
- Readily available operators
  - Remapping and interpolation
  - Gradient, divergence, laplacian
  - Spherical Harmonics transforms
- Support structured and unstructured grids (**global** as well as **regional**)

Atlas typical workflow

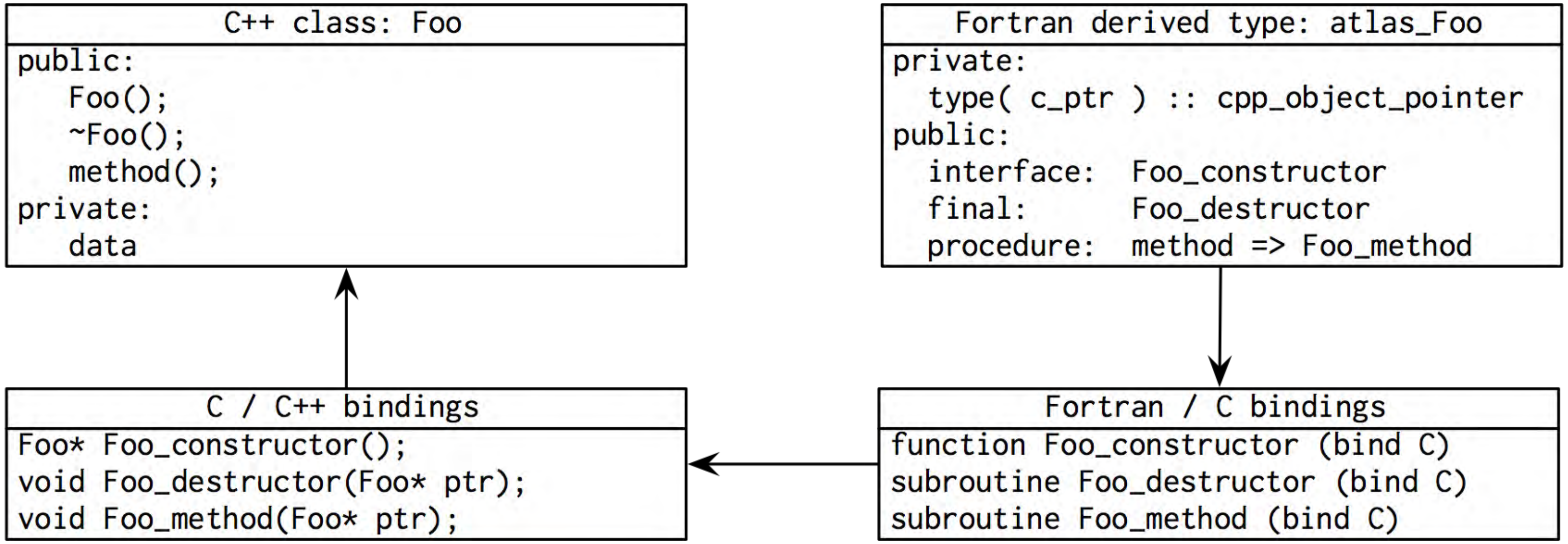
Octahedral Gaussian grid (O32)

Grid

Partitions
with halo's



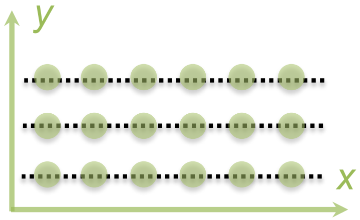
Object oriented design C++ / Modern Fortran

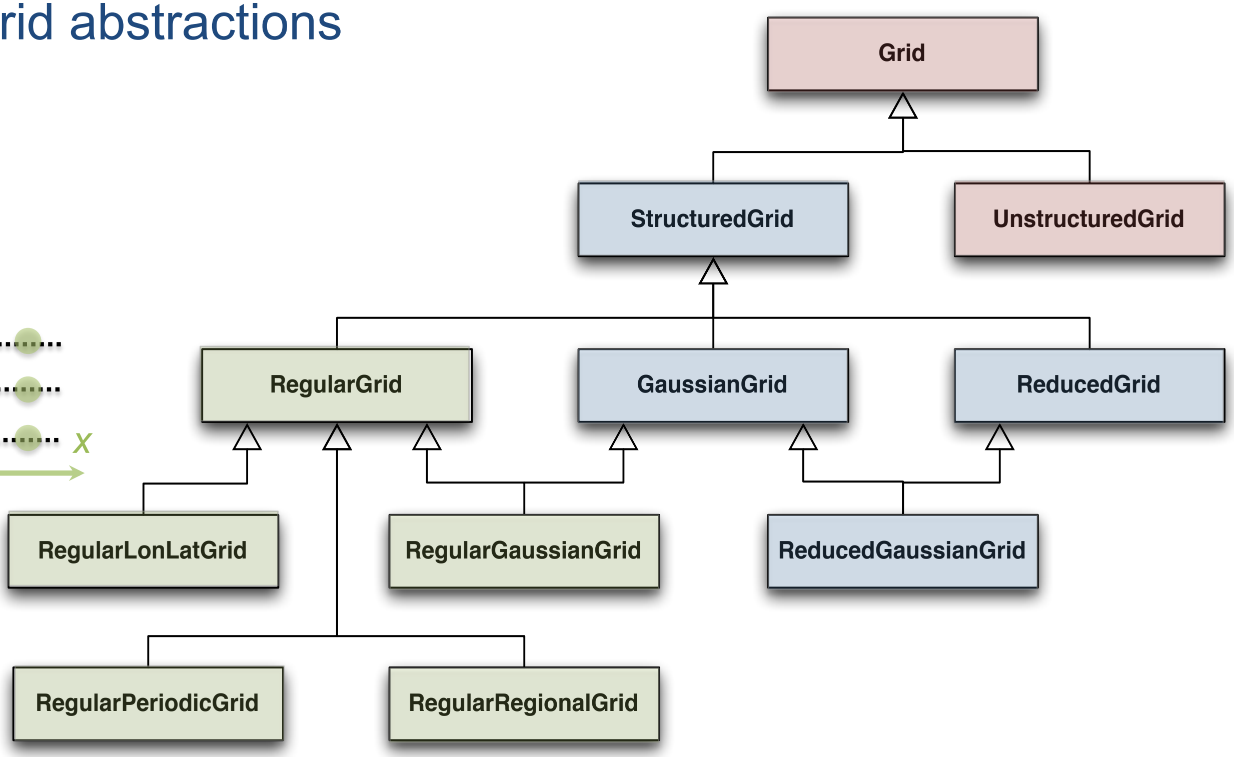


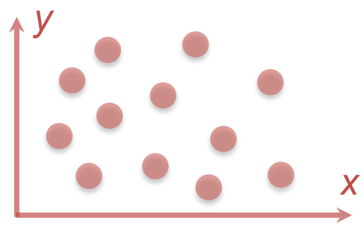

```cpp
Foo foo;
foo.method();
```

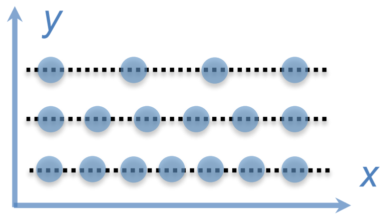
```fortran
type(atlas_Foo) :: foo
foo = atlas_Foo()
call foo%method()
```

A library of grid abstractions









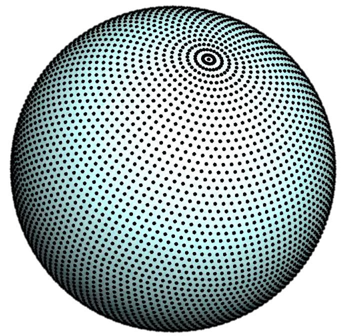
Example creation of operational octahedral reduced Gaussian grid using unique identifier

C++

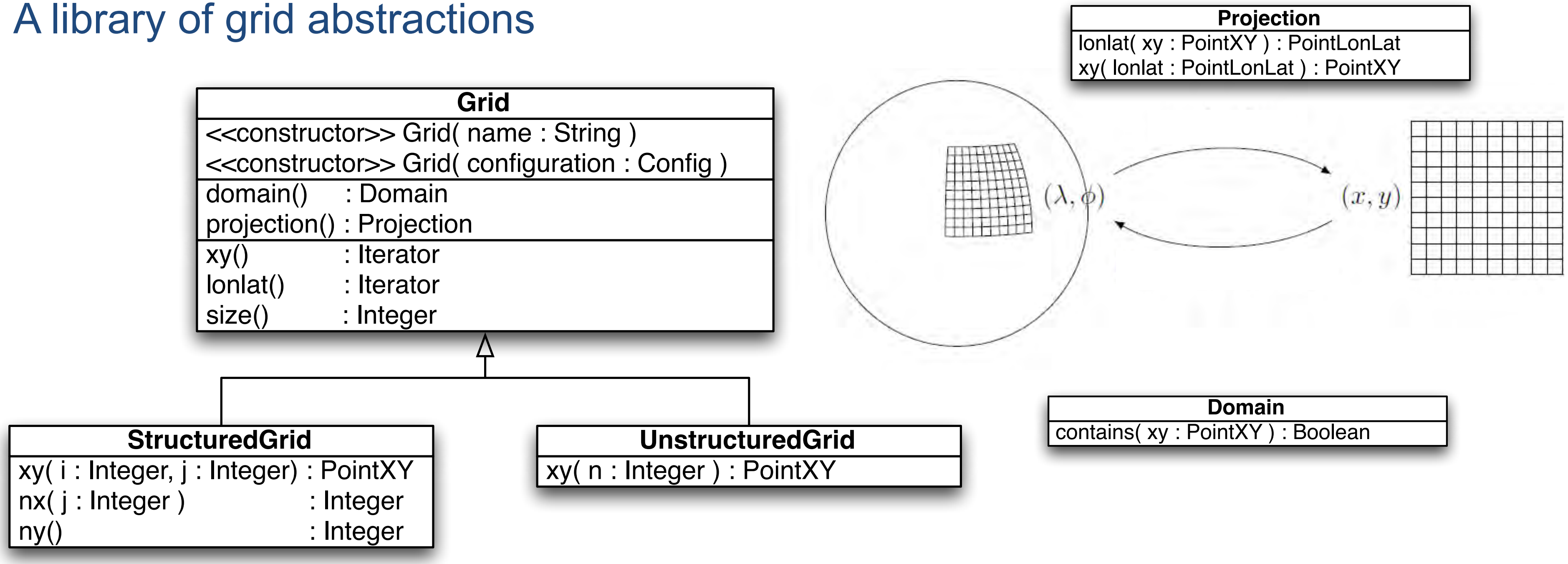
```
atlas::Grid grid;
grid = atlas::Grid ( “O1280” )
```

Fortran

```
type(atlas_Grid) :: grid
grid = atlas_Grid( “O1280” )
```



A library of grid abstractions



Iterating over all grid points, regardless of used projection, structure, domain

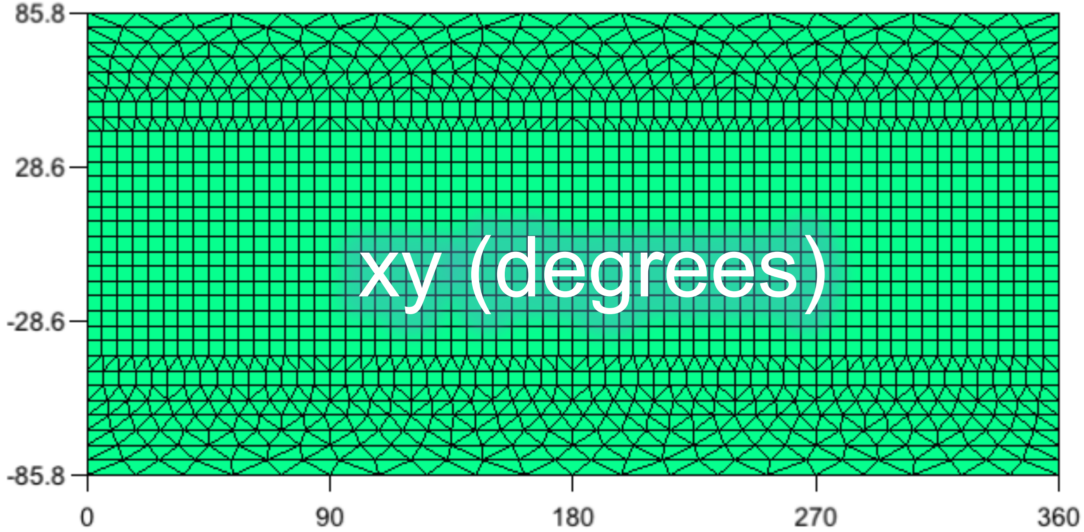
Grid coordinates (x,y)

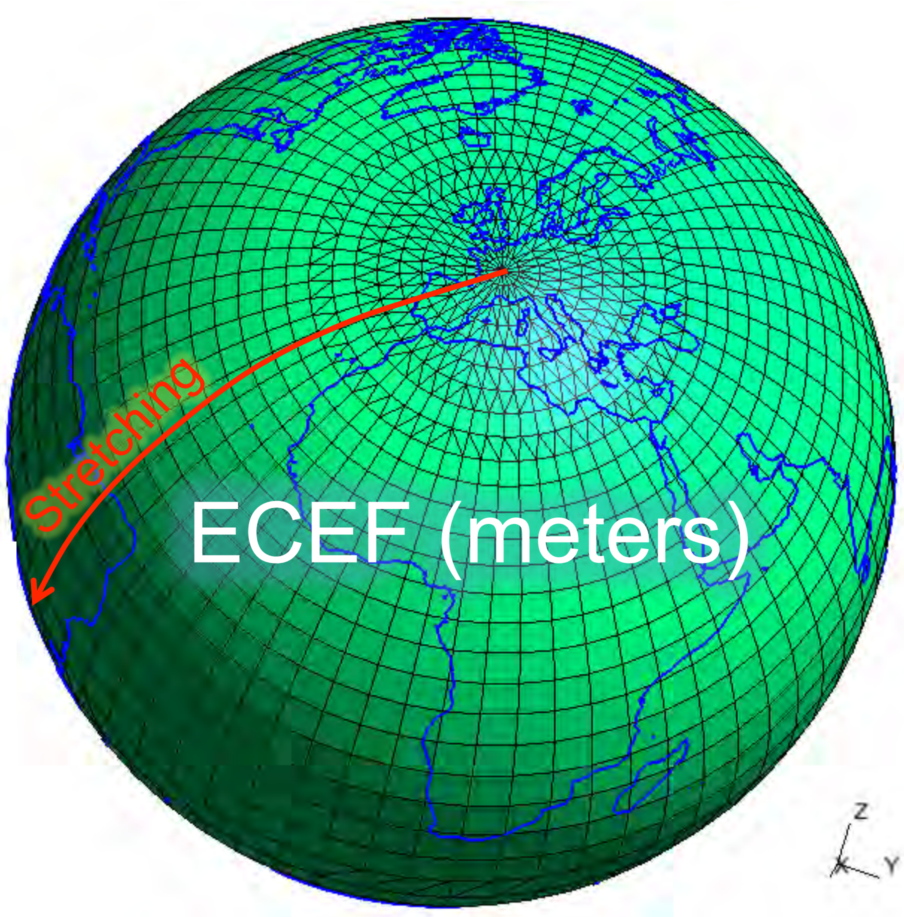
```
for( PointXY p : grid.xy() ) {
    double x = p.x();
    double y = p.y();
}
```

Geographic coordinates (lon,lat)

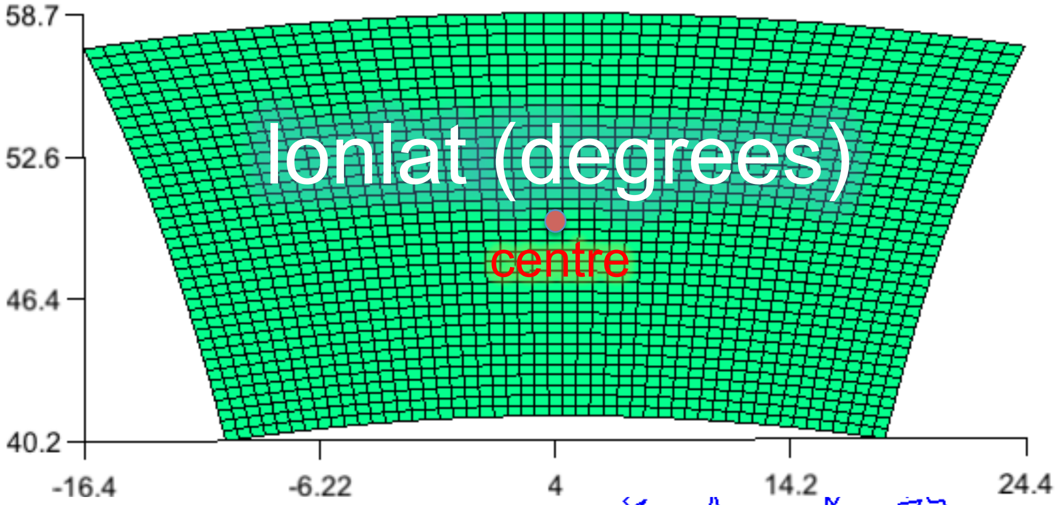
```
for( PointLonLat p : grid.lonlat() ) {
    double lon = p.lon();
    double lat = p.lat();
}
```

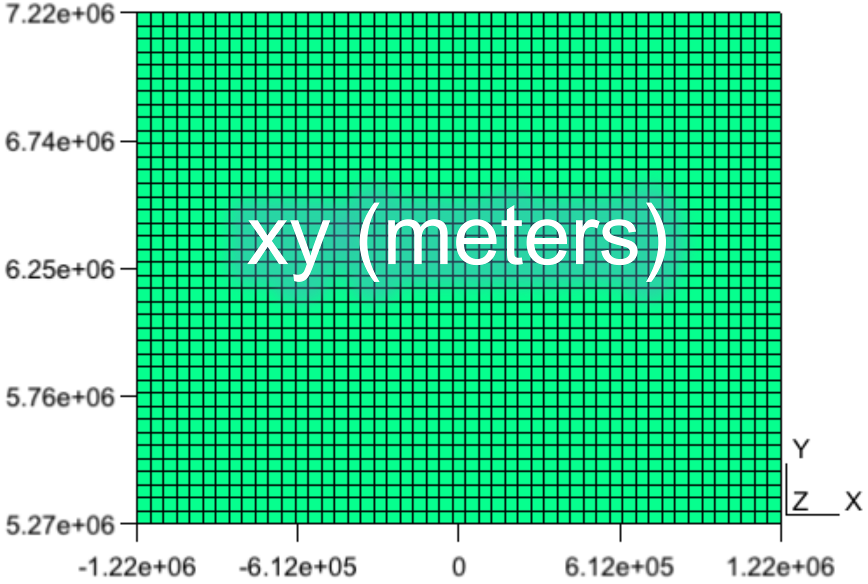
```
{
  "type" : "classic_gaussian",
  "N" : 16
  "projection" : {
     "type" : "rotated_schmidt",
     "north_pole"  : [3,47]
     "stretching_factor" : 2
  }
}
```

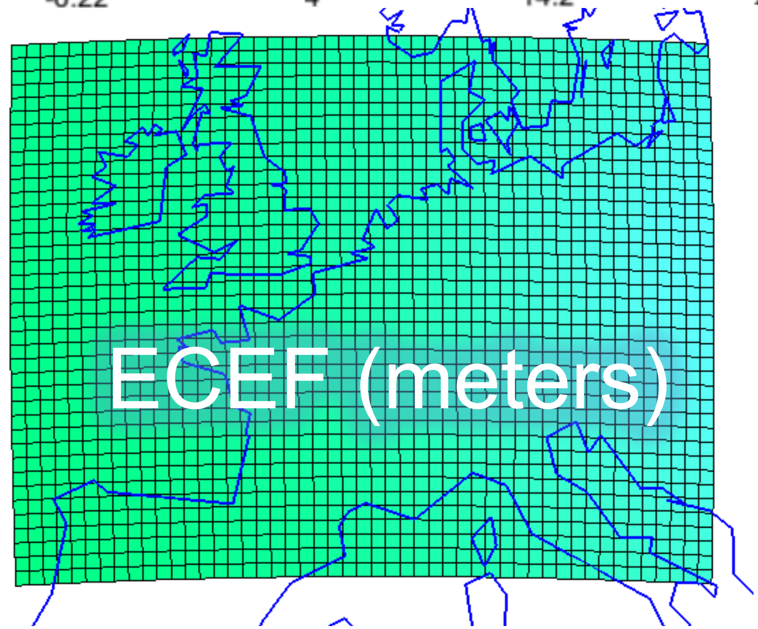




```
{
   "type" : "regional",
   "nx" : 50,   "dx" : 50000,
   "ny" : 40,   "dy" : 50000,
   "lonlat(centre)" : [4,50],
   "projection" : {
      "type" : "lambert",
      "latitude1"   : 50,
      "longitude0" : 4
   }
}
```

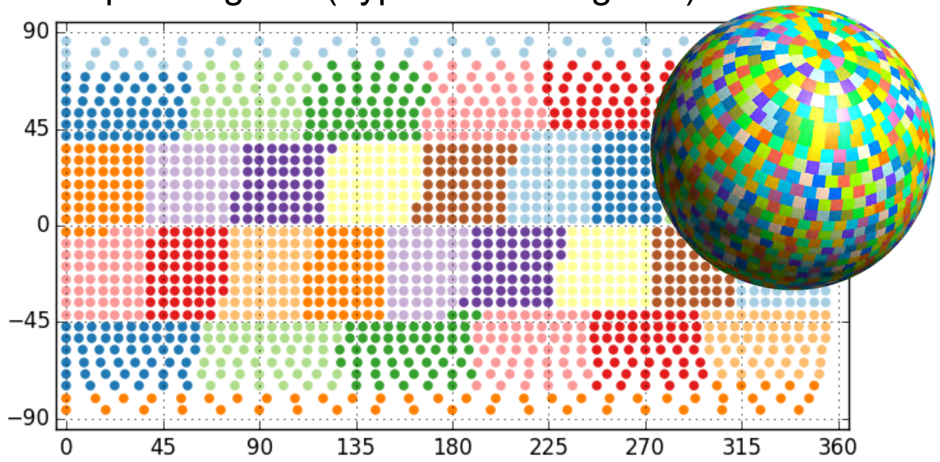




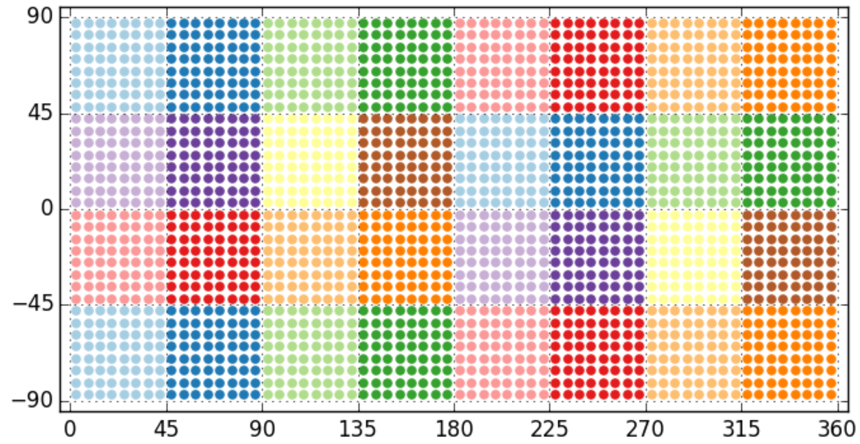


Multiple domain decomposition strategies

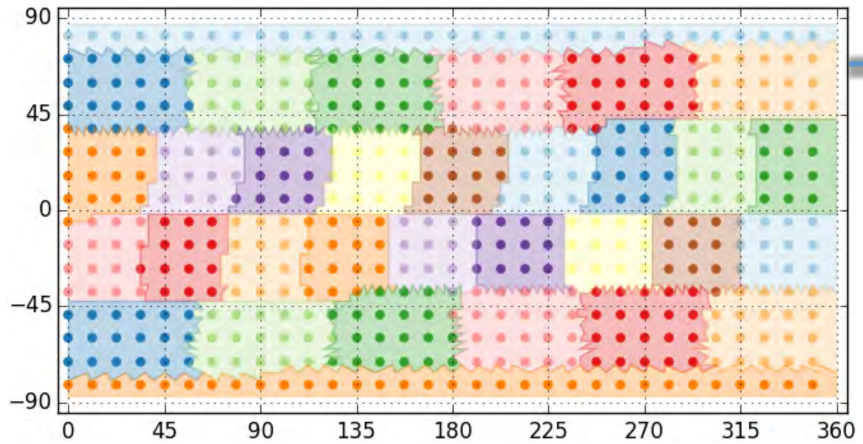
Equal Regions (typical for IFS grids)



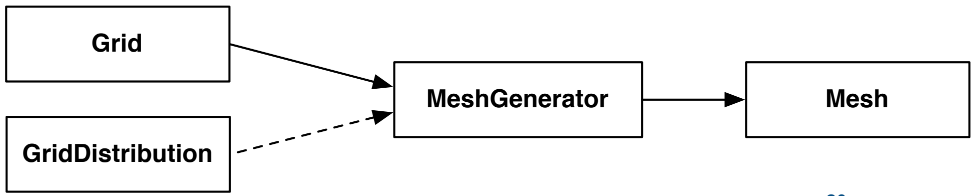
Checkerboard (typical for regional grids)



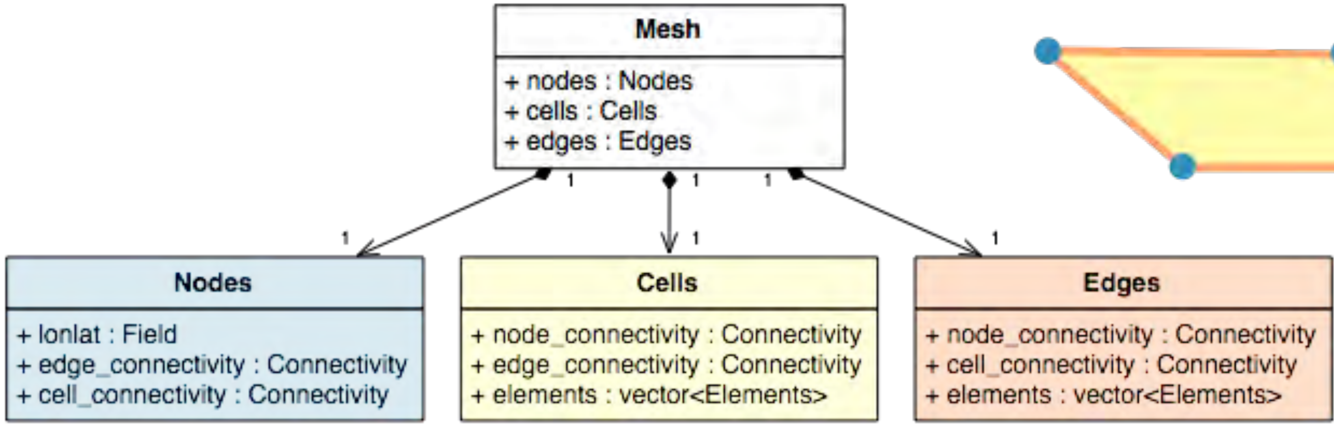
Matching Mesh

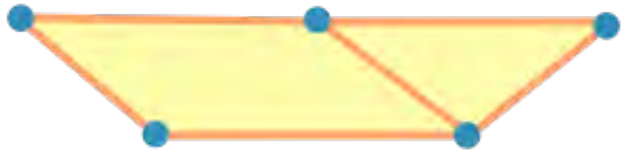


Any grid can follow the domain decomposition of an already distributed mesh

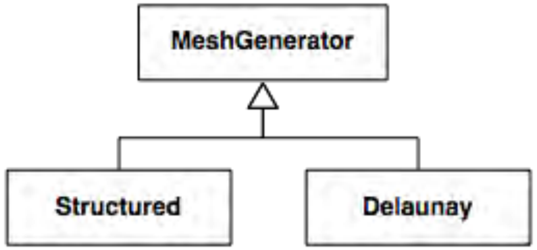


Mesh: connecting points (nodes) by edges and cells using connectivity tables





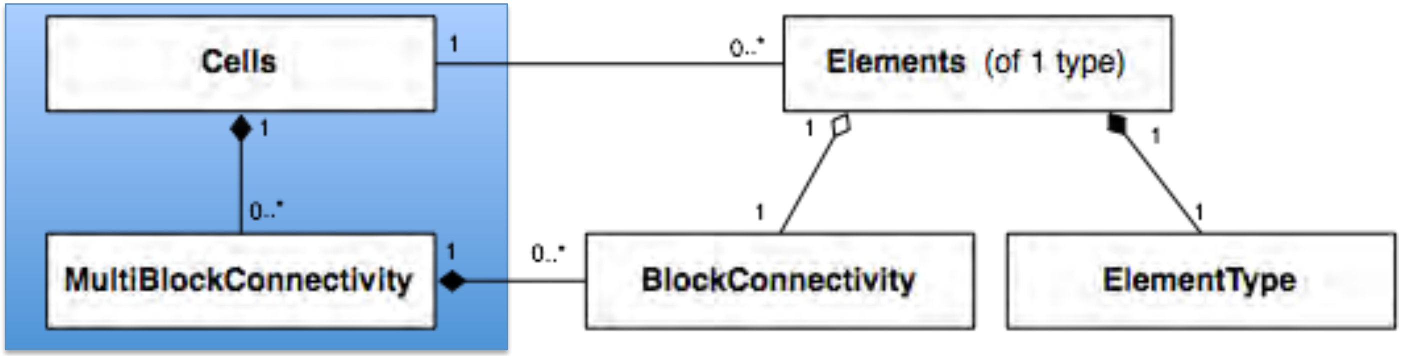
Mesh generation




```
MeshGenerator meshgen(“structured”);
Mesh mesh = meshgen.generate(grid);
```

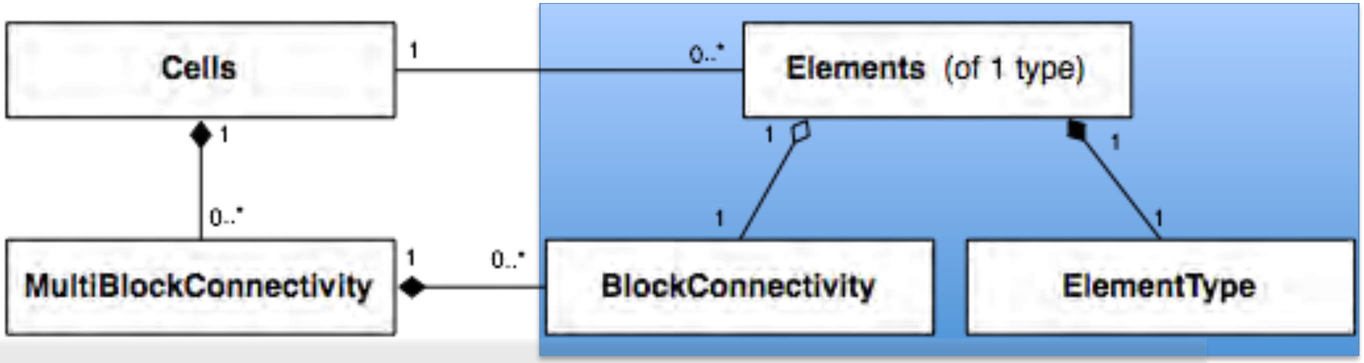
```
type(atlas_MeshGenerator) :: meshgen
type(atlas_Mesh)          :: mesh
meshgen = atlas_MeshGenerator(“structured”)
mesh = meshgen%generate(grid)
```

Example "for" over all elements



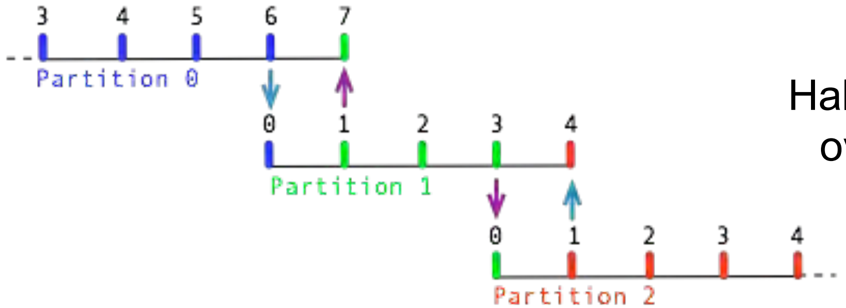

```
MultiBlockConnectivity& node_connectivity = mesh.cells().node_connectivity();

// Loop over ALL elements
for( size_t jelem=0; jelem<mesh.cells().size(); ++jelem ) {

    // Compute average over cell
    double cell_average = 0;
    for( size_t jnode=0; jnode<mesh.cells().nb_nodes(jelem);
        cell_average += field( node_connectivity(jelem,jnode) );
    cell_average /= double(mesh.cells().nb_nodes(jelem));
}
```

Example "for" per element type

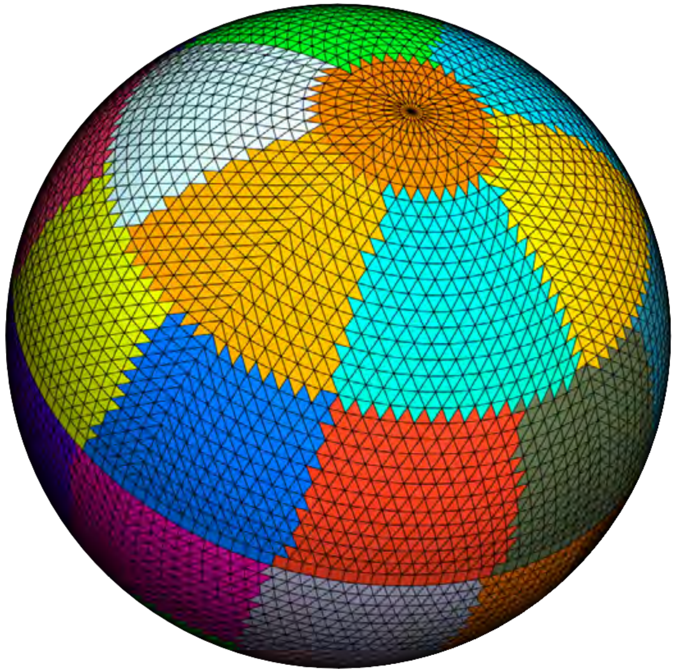


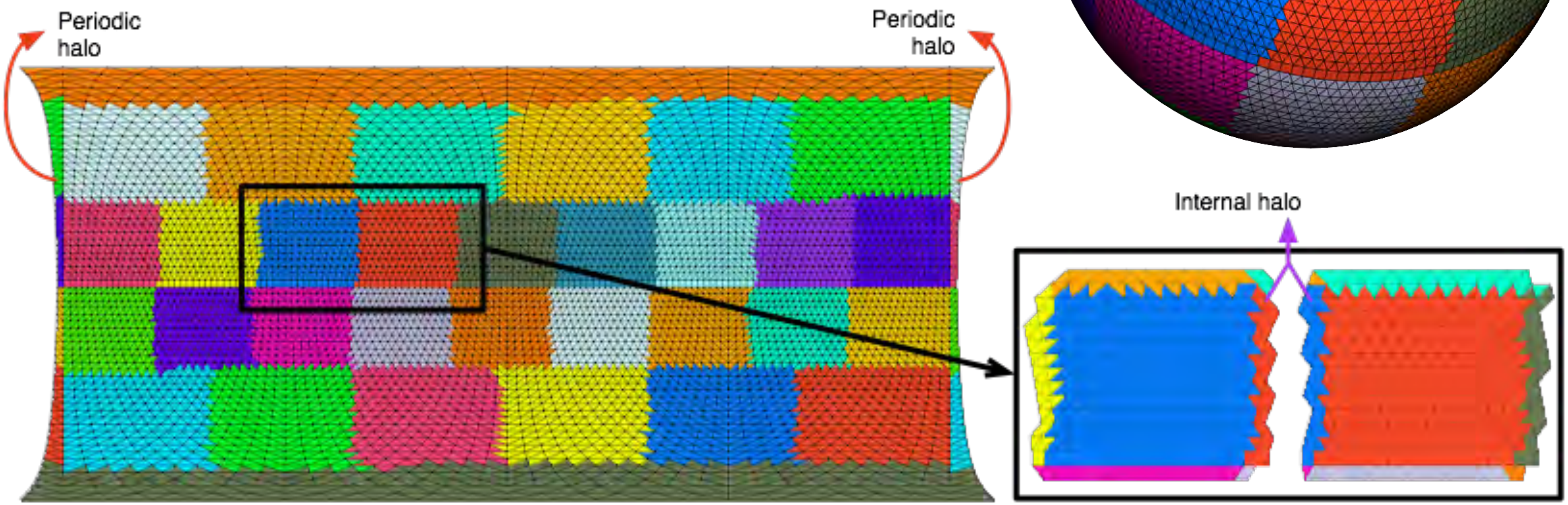

```
// Loop over element types
for( size_t jtype=0; jtype<mesh.cells().nb_types(); ++jtype ) {

   Elements& elements = mesh.cells().elements(jtype);
   BlockConnectivity& node_connectivity = elements.node_connectivity();

   // Loop over elements of one type
   for( size_t jelem=0; jelem<elements.size(); ++jelem ) {

      // Compute average over cell
      double cell_average = 0;
      for( size_t jnode=0; jnode<elements.nb_nodes();
         cell_average += field( node_connectivity(jelem,jnode) );
      cell_average /= double(elements.nb_nodes());
   }
}
```

Halo exchanges

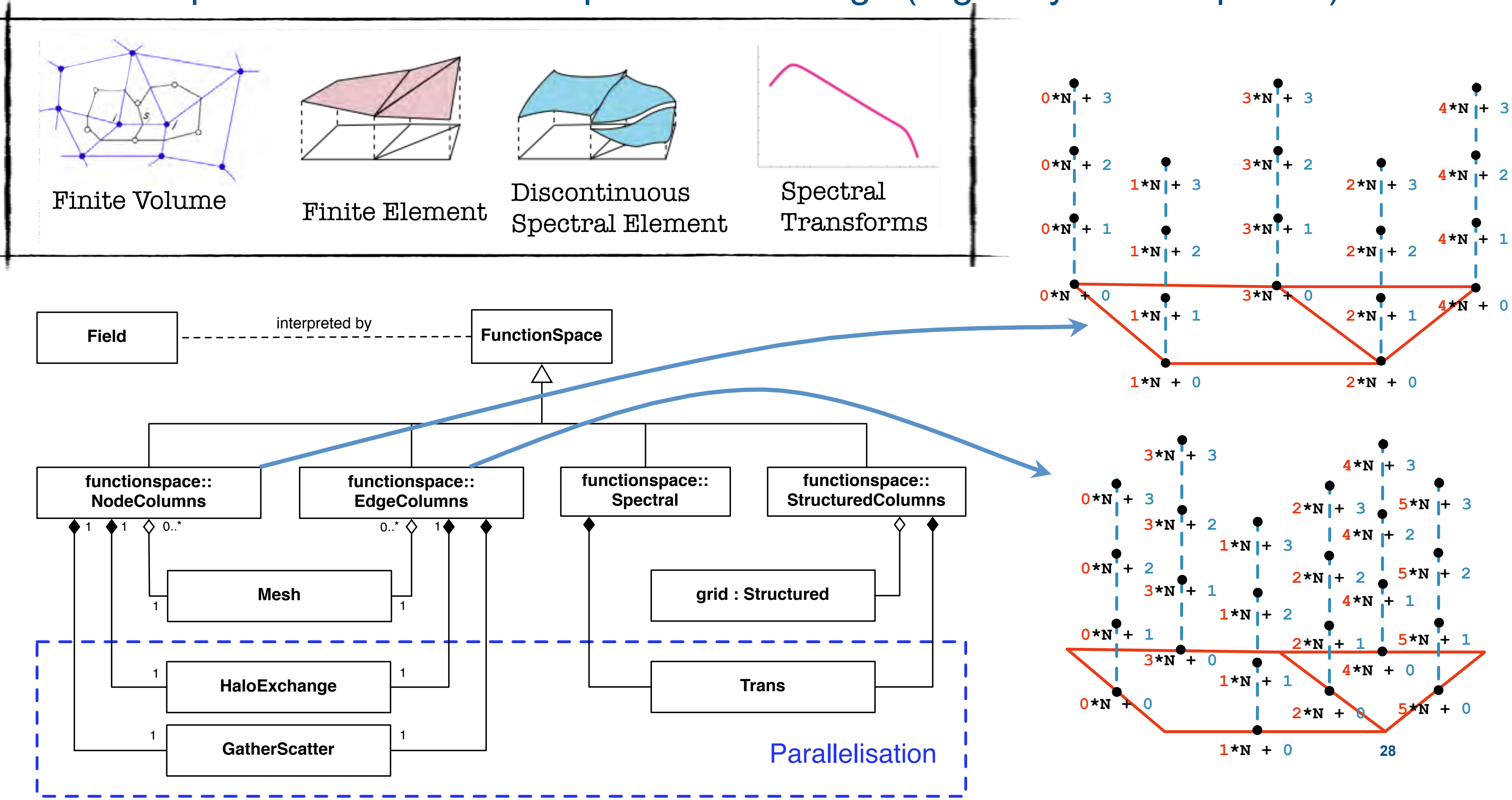


HaloExchange in overlap regions



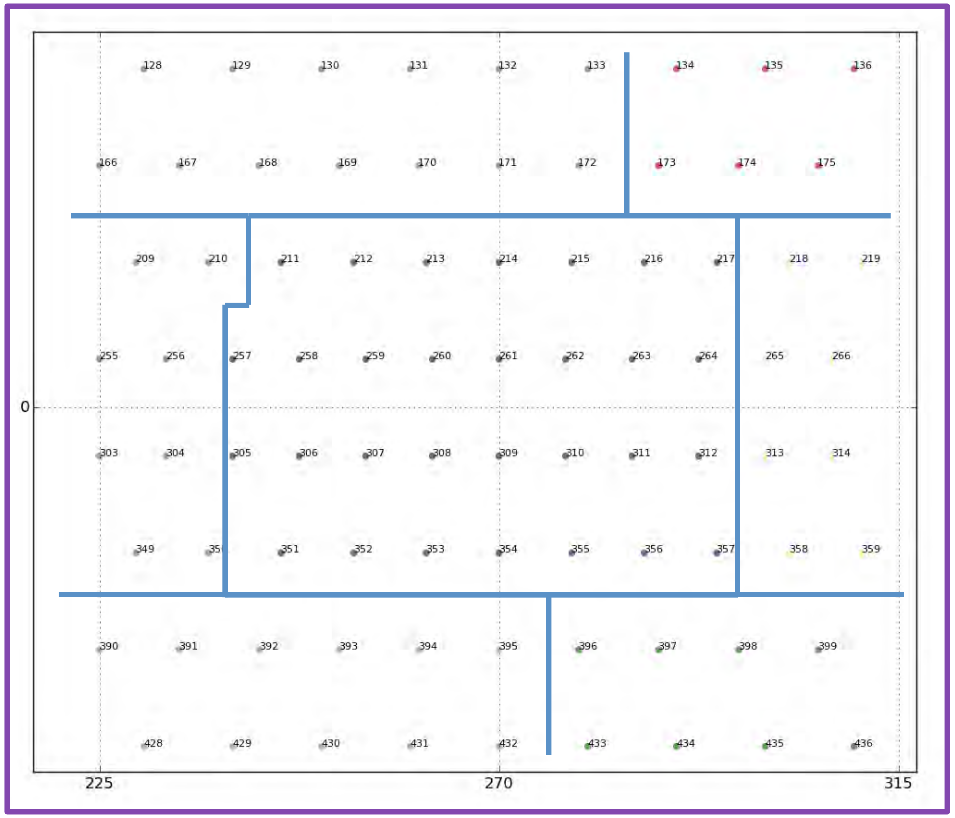


FunctionSpaces: Discretisation specific knowledge (arguably model specific)



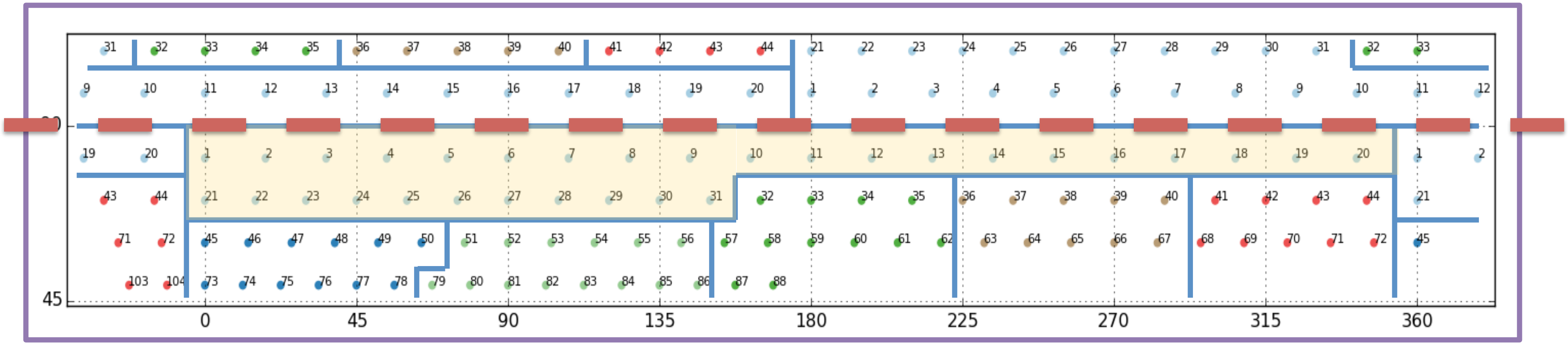
StructuredColumns FunctionSpace

- What?
  - Discretisation of Fields without a Mesh (only StructuredGrid, cfr. IFS)
  - Distributed view of horizontal grid, plus structured vertical levels
- Halos
  - Fast algorithm to create halo
  - HaloExchange capabilities !!!
  - Halo's over pole !!!



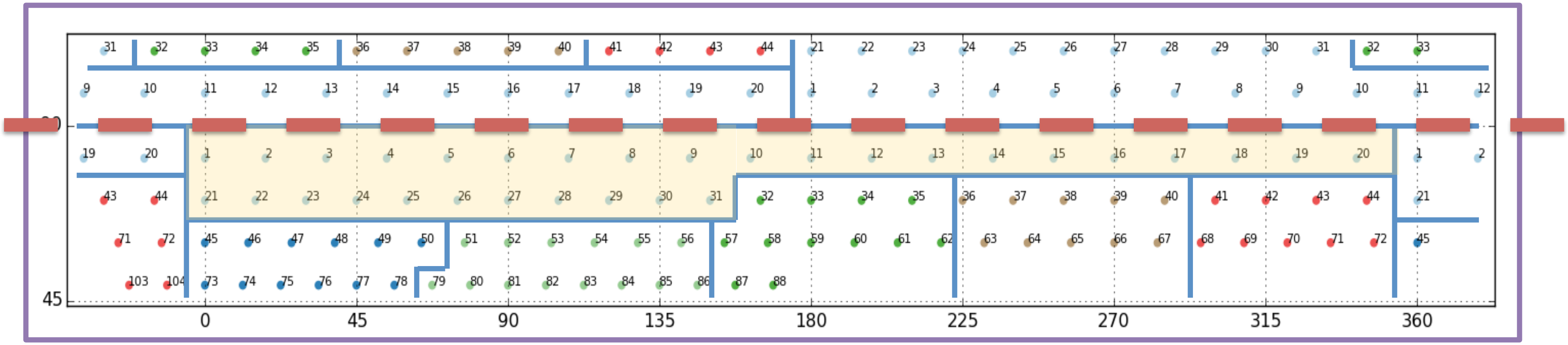
StructuredColumns FunctionSpace

- What?
  - Discretisation of Fields without a Mesh (only StructuredGrid, cfr. IFS)
  - Distributed view of horizontal grid, plus structured vertical levels
- Halos
  - Fast algorithm to create halo
  - HaloExchange capabilities !!!
  - Halo’s over pole !!!

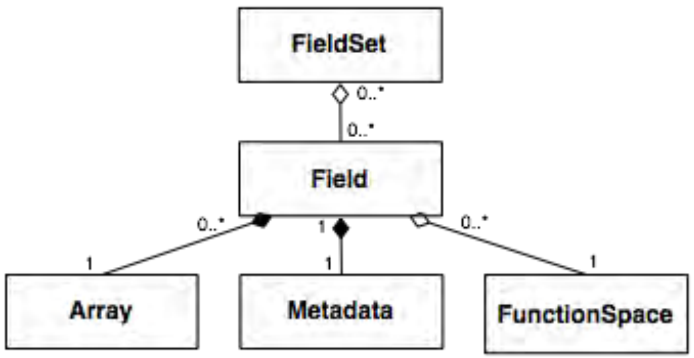


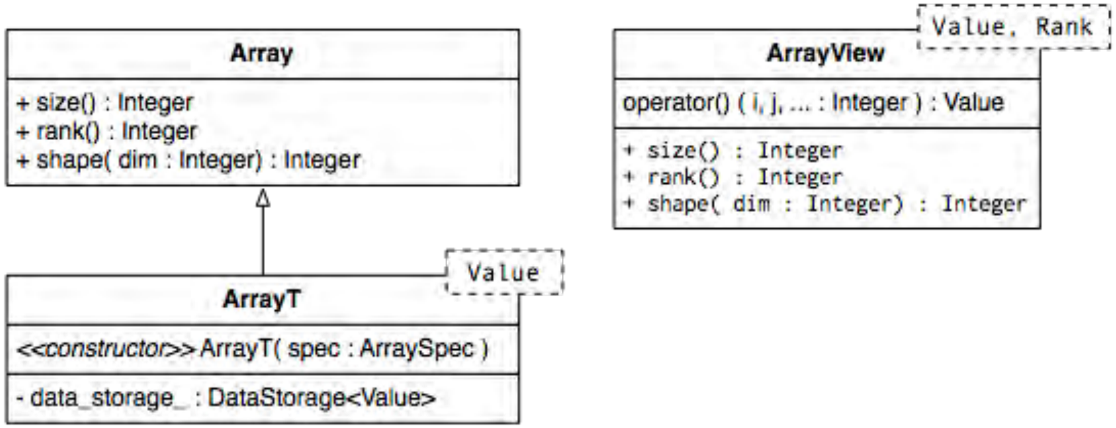
StructuredColumns FunctionSpace

- How?
  - Mapping (i , j) → local index
  - Indirect addressing but… increased flexibility
    - Room to optimise cache efficiency using space-filling curve
    - One-to-one mapping with mesh-based functionspace (excluding halo)
    - Halo points added at the end



Field: container for data with metadata

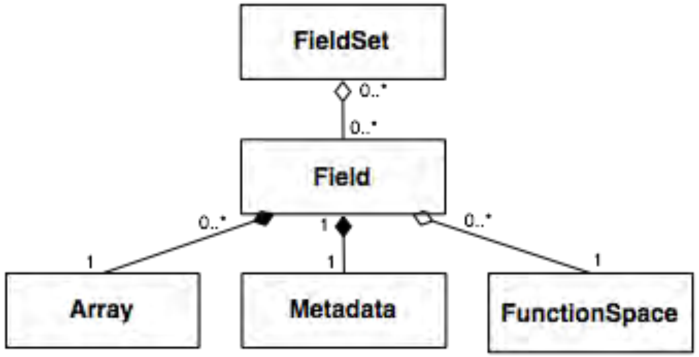


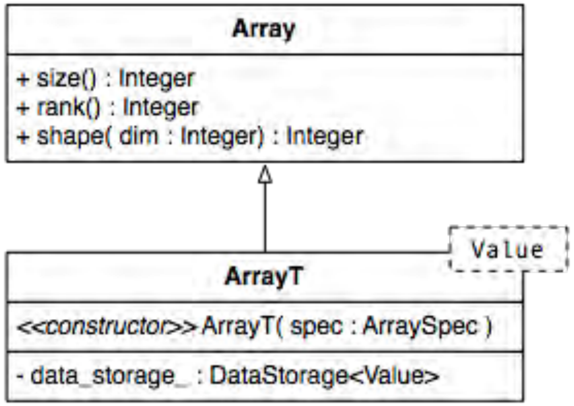


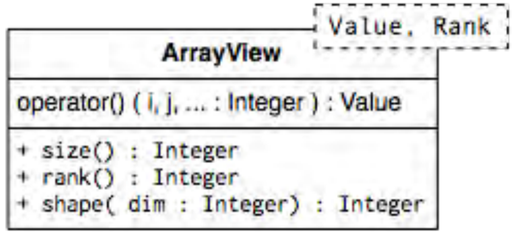

```
Field field = Field(
   /* name  */  "P",
   /* kind  */  make_datatype<double>(),
   /* shape */  make_shape(npts,nlev) );

field.metadata().set("units","hPa")
field.metadata().set("time",450)
```

```
type(atlas_Field)    :: field
type(atlas_Metadata) :: field_metadata

field = atlas_Field(                &
           name="P",                &
           kind=atlas_real(jprb),   &
           shape=[nlev,npts] )

field_metadata = field%metadata()
call field_metadata%set("units","hPa")
call field_metadata%set("units",450)
```

Acces to field data







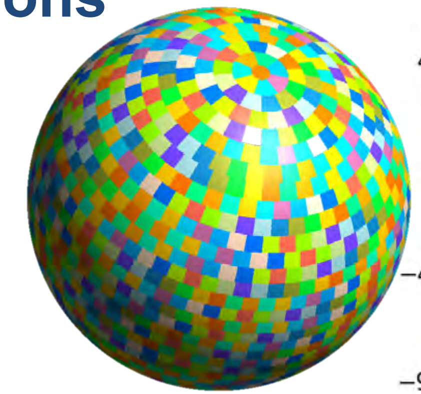

```
auto field_data =
        make_view<double,2>( field );

for( int jnode=0; jnode<npts; ++jnode ) {
    for( int jlev=0; jlev<nlev; ++jlev ) {
        field_data(jnode,jlev) = …
    }
}
```

```
real(8), pointer :: field_data(:,:)

call make_view( field, field_data )
do jnode=1,npts
    do jlev=1,nlev
        field_data(jlev,jnode) = …
    enddo
enddo
```

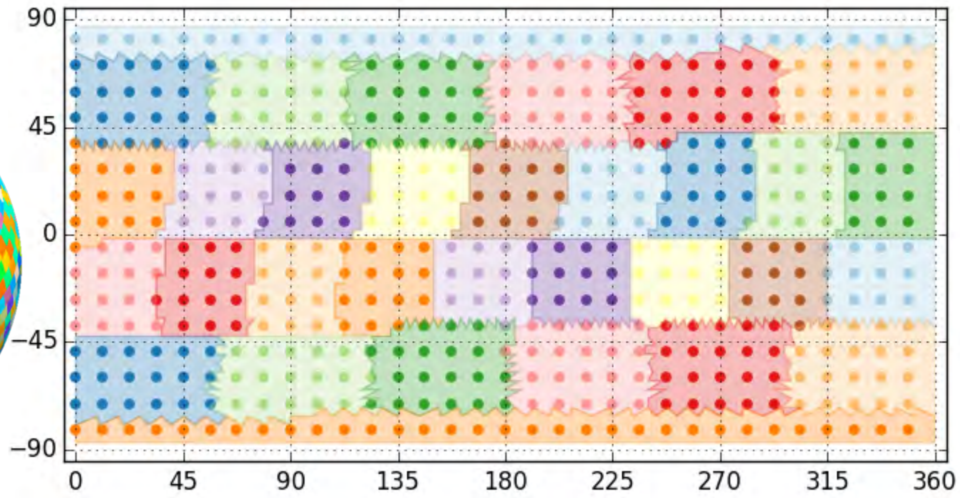
Remapping, using matching domain decompositions

- Domain decompositions match to avoid MPI communication

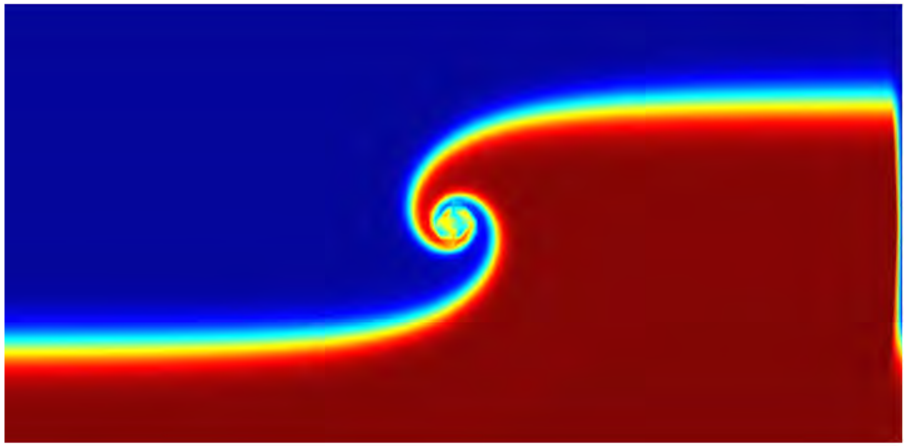
- Parallel interpolation:
  - nearest neighbour
  - k-nearest neighbour
  - linear element-based

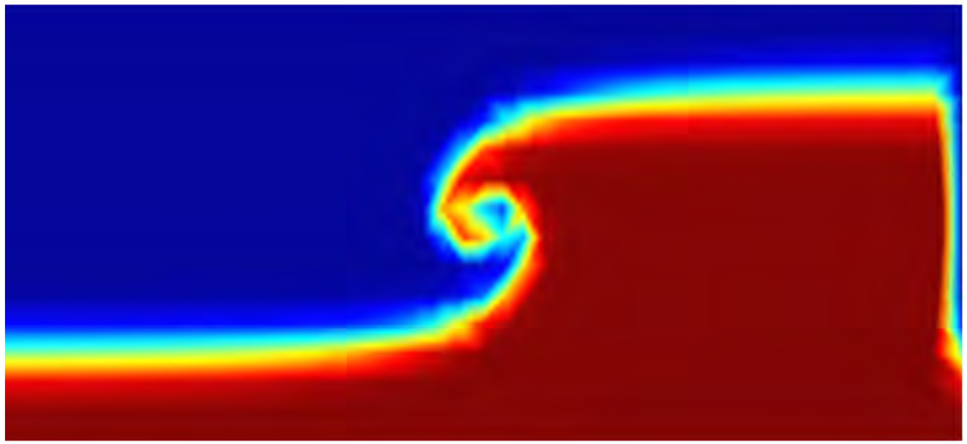


Thanks to Pedro Maciel (ECMWF)

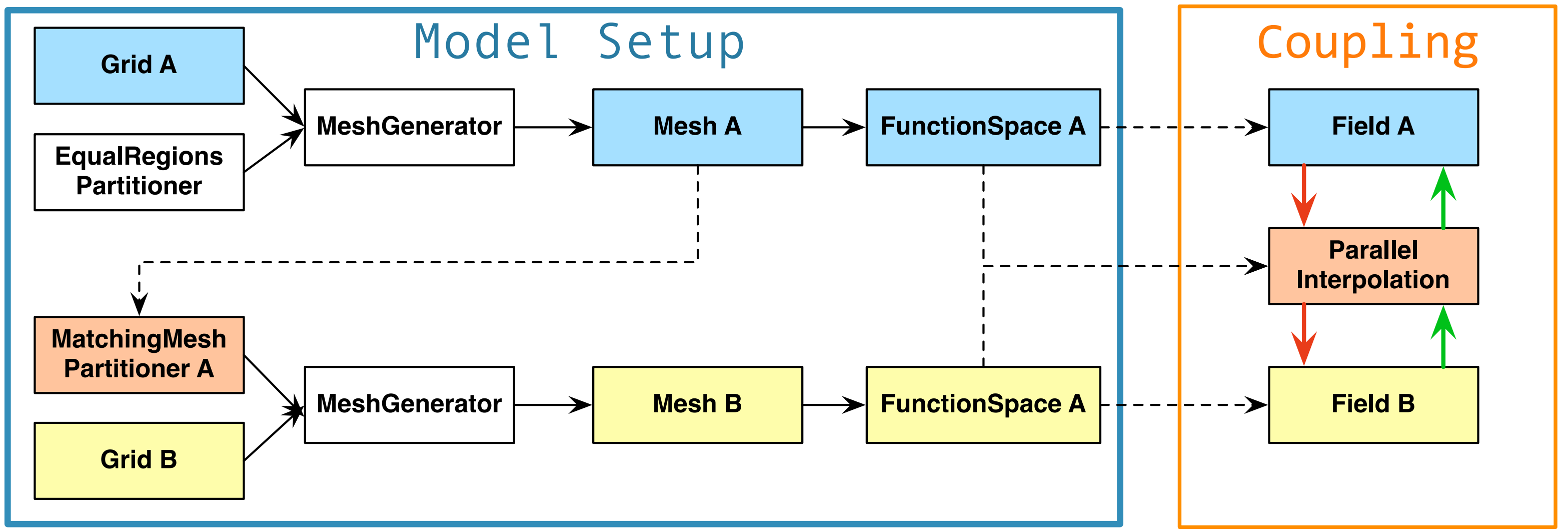


Example interpolation: O32 to F8





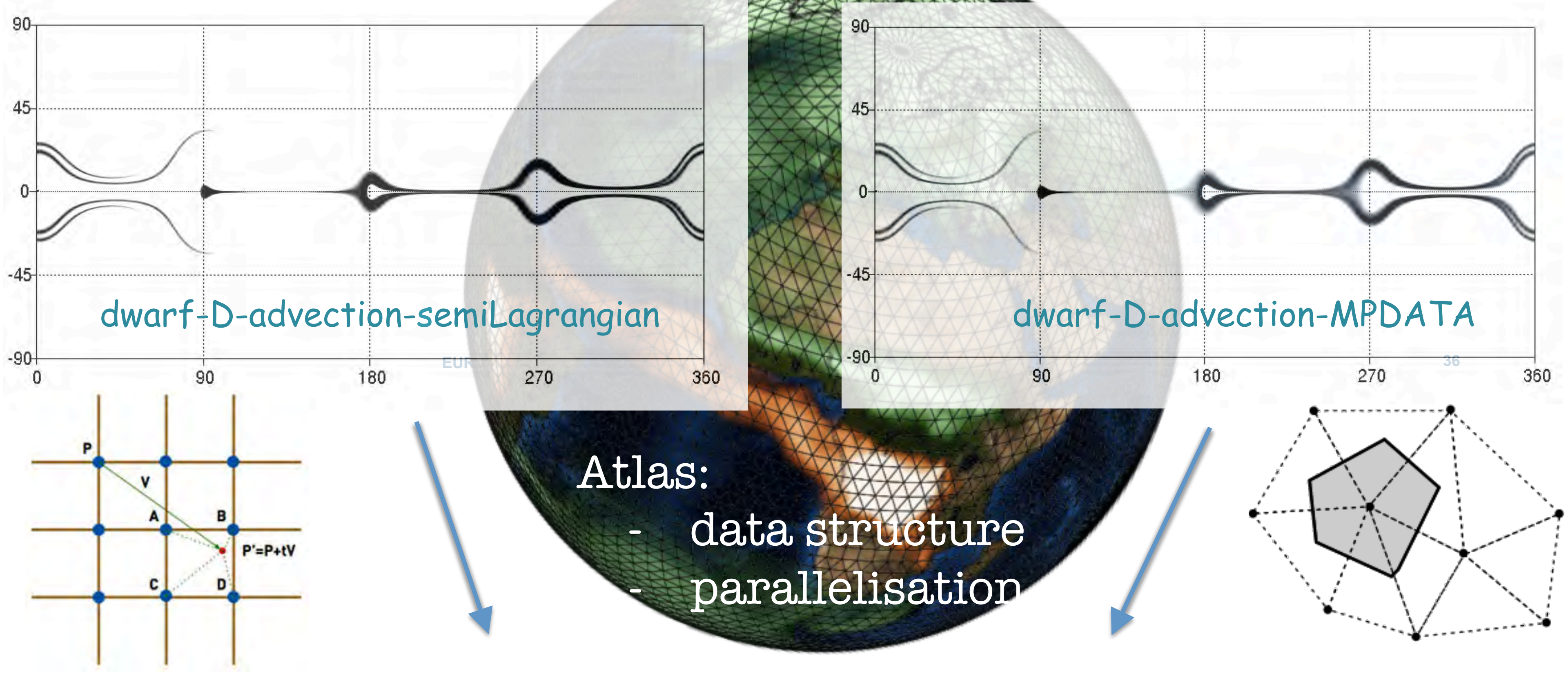
Remapping using matching domain decompositions




```
grid_A          = atlas_Grid("O1280")
partitioner_A   = atlas_EqualRegionsPartitioner()
mesh_A          = meshgenerator % generate(grid_A, partitioner_A)
fs_A            = atlas_functionspace_NodeColumns(mesh_A)

grid_B          = atlas_Grid("O640")
partitioner_B   = atlas_MatchingMeshPartitioner(mesh_A)
mesh_B          = meshgenerator % generate(grid_B, partitioner_B)
fs_B            = atlas_functionspace_NodeColumns(mesh_B)

interpolation_AB = atlas_Interpolation(type="finite-element",
                                       source=fs_A, target=fs_B)
```

```
call interpolation_AB %
              execute(field_A, field_B)
```

IFS and Advection dwarfs

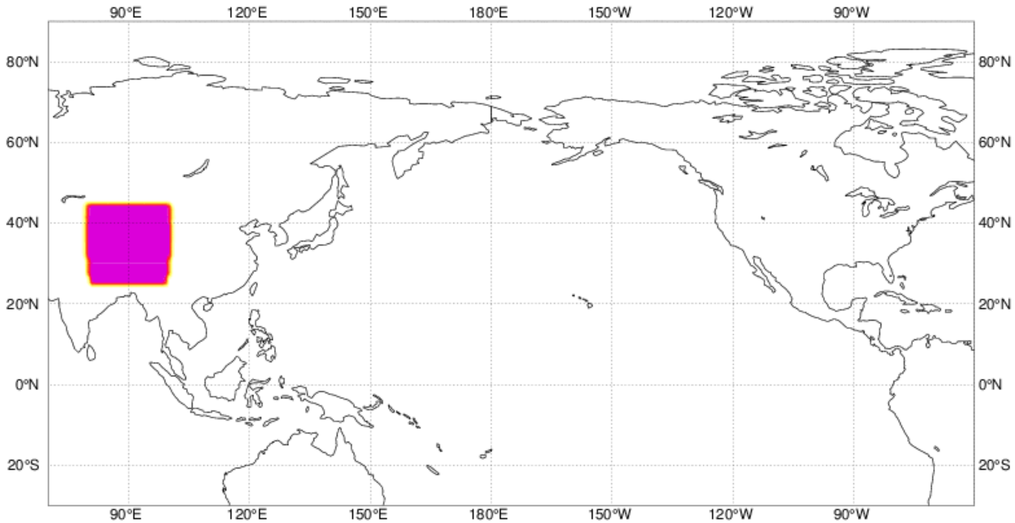


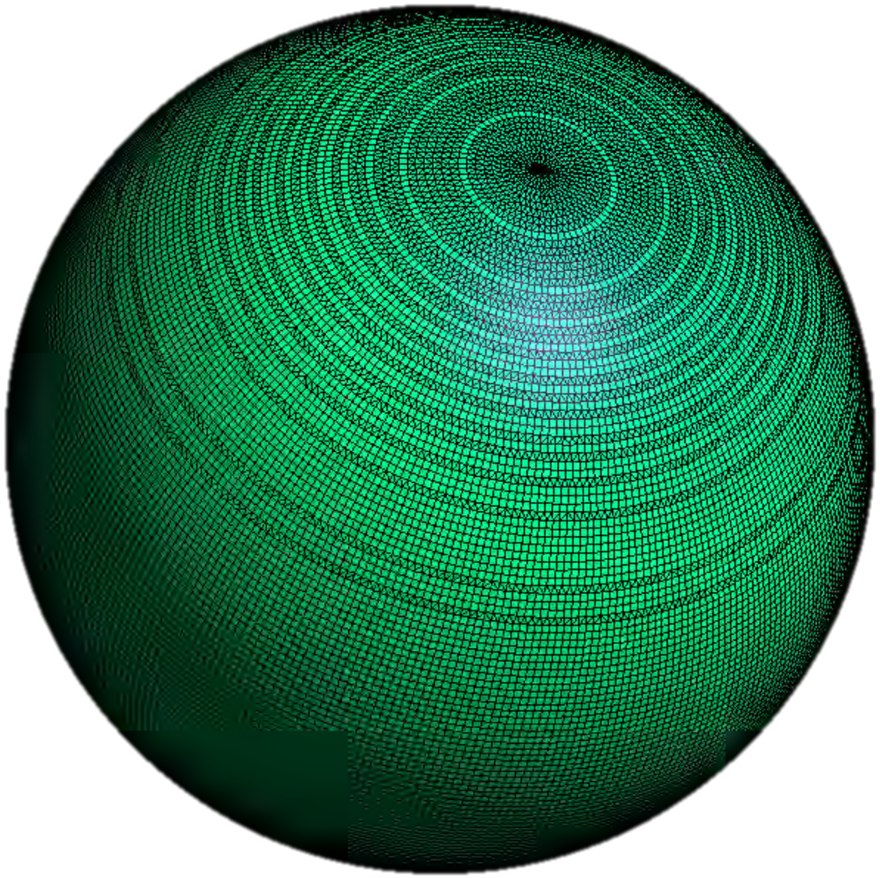
Advection abstraction in IFS based on Atlas

Thanks to Michail Diamantakis (ECMWF)

Passive tracer advection

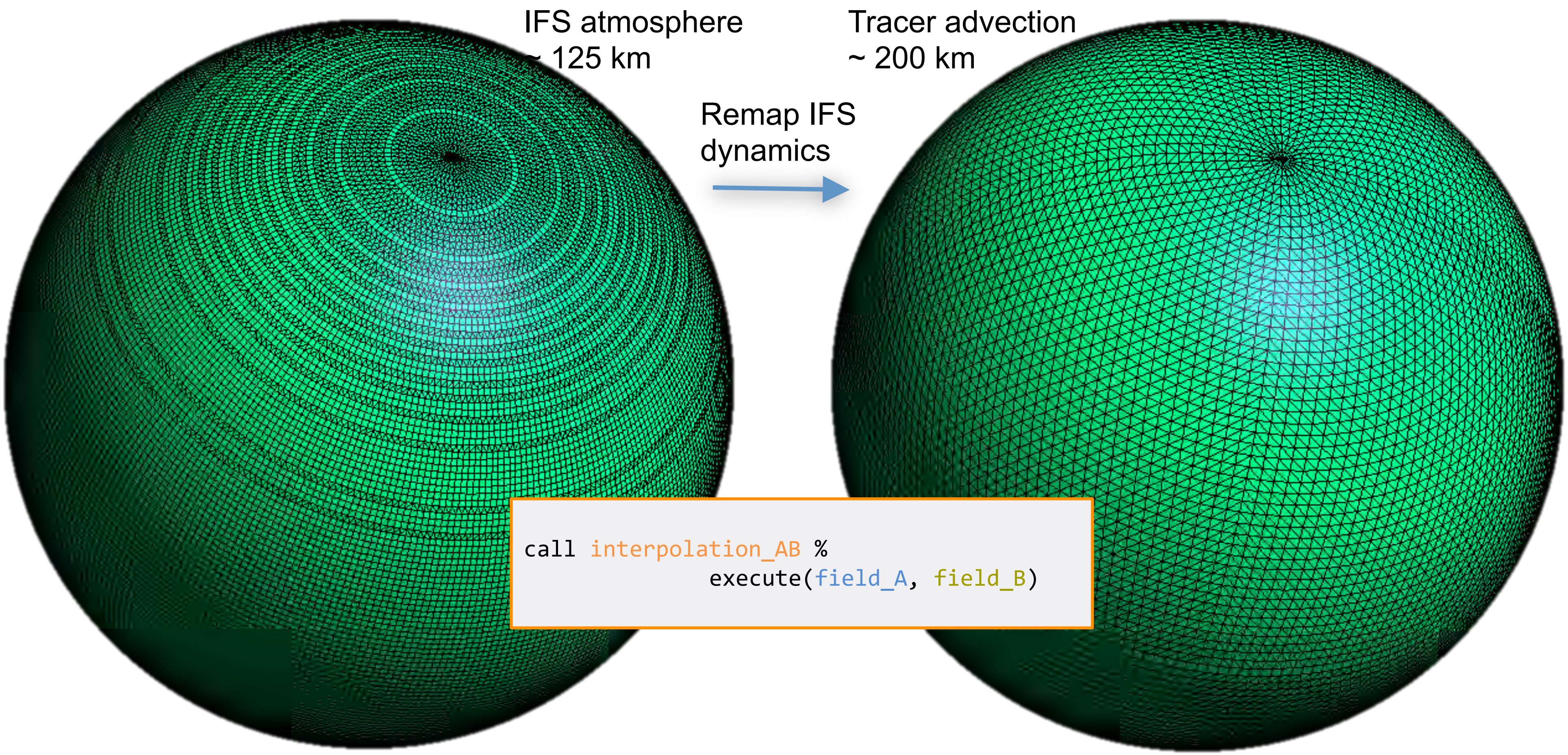
Tracer initialised over Asia – 125 km model resolution (N80)



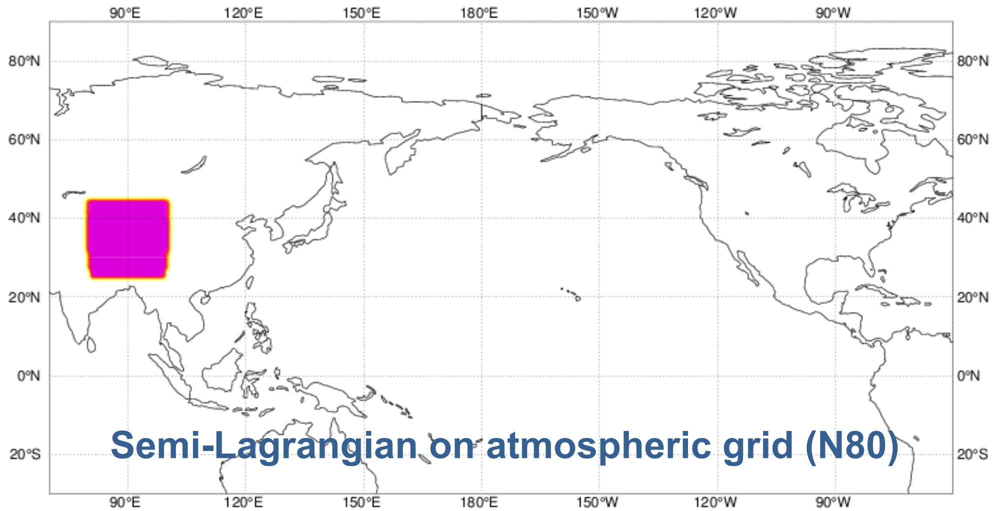


proof of concepts!
(read: poor Fortran code was abused in the process
to get quick results)

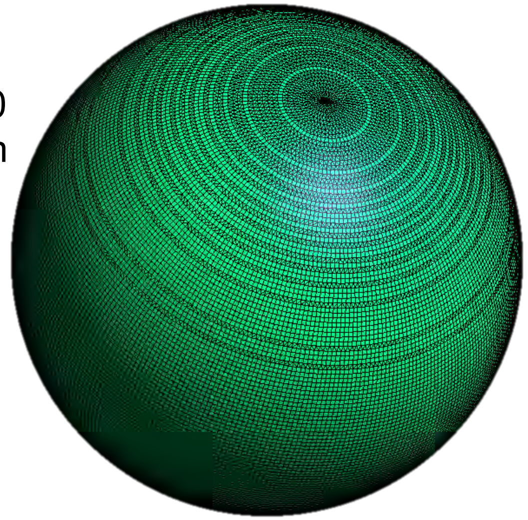
Can we use a coarser resolution for advection, but with wind from fine resolution?

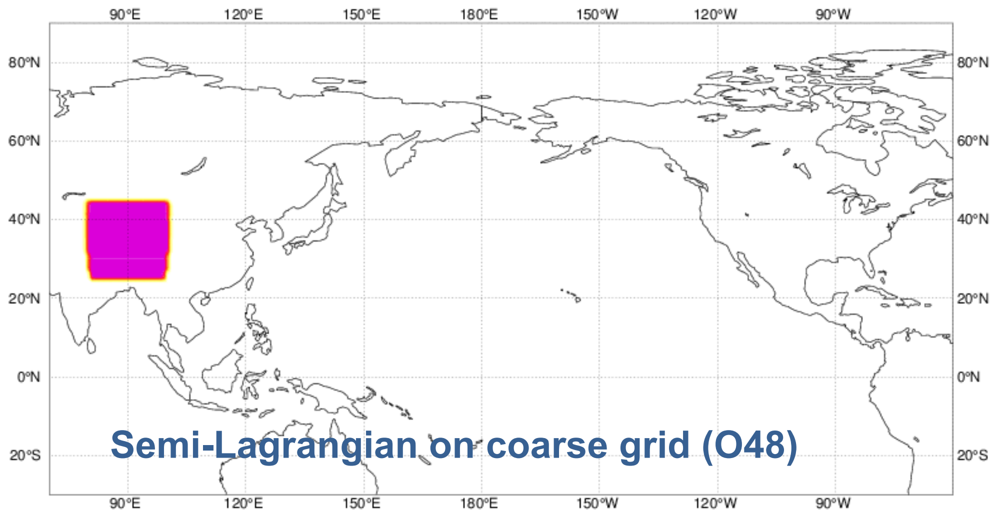



```
call interpolation_AB %
          execute(field_A, field_B)
```

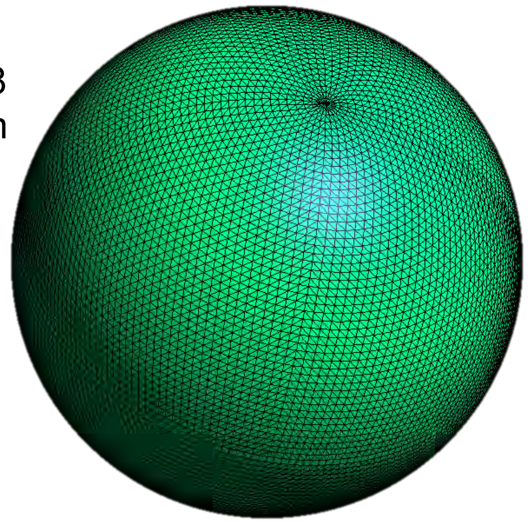


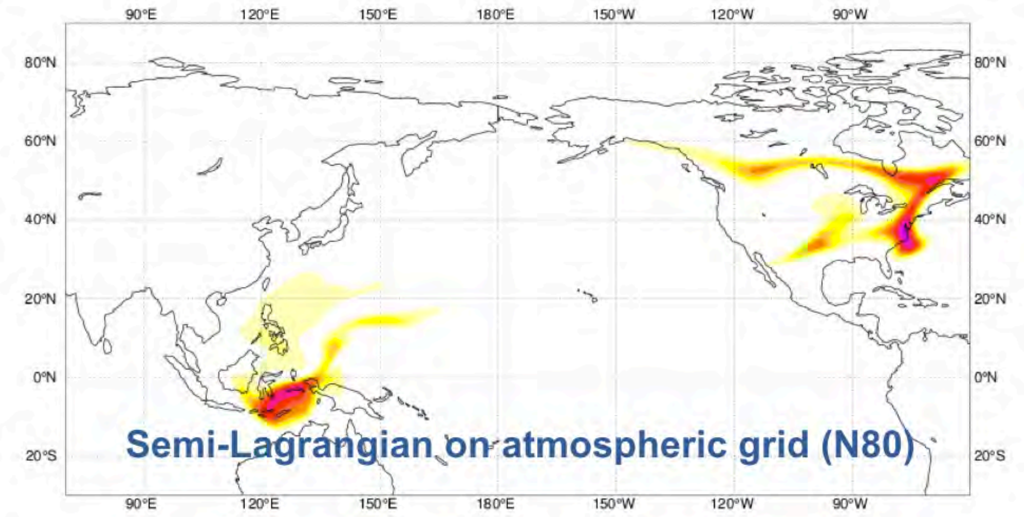
N80
~ 125 km



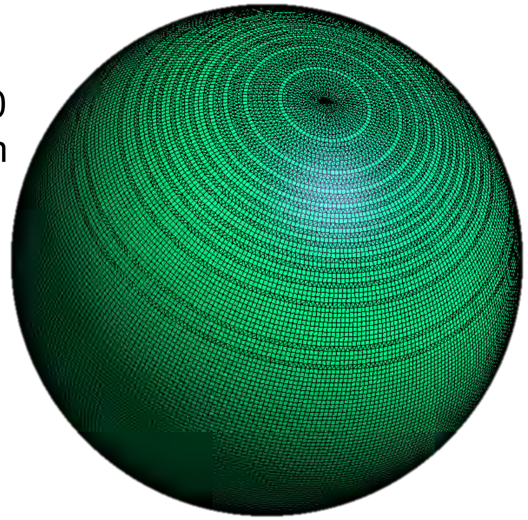


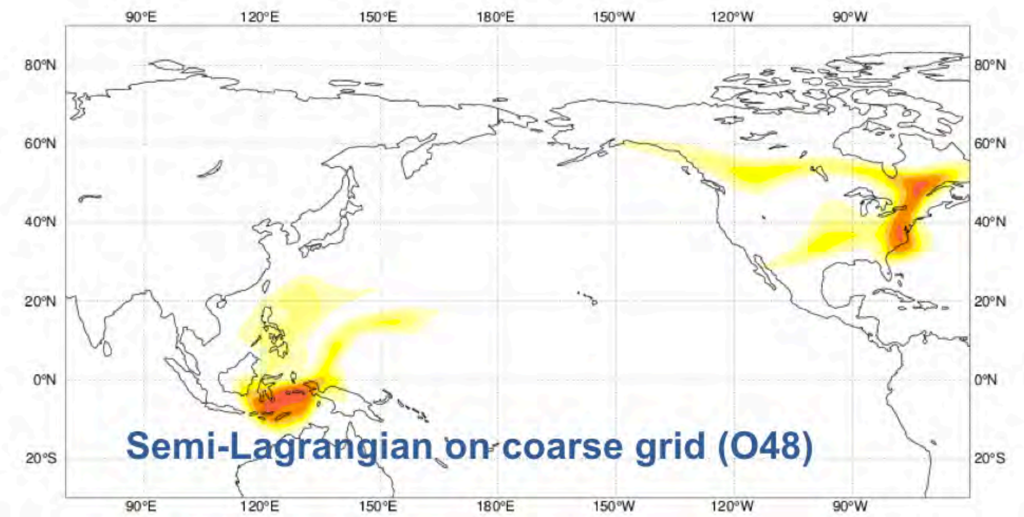
O48
~ 200 km



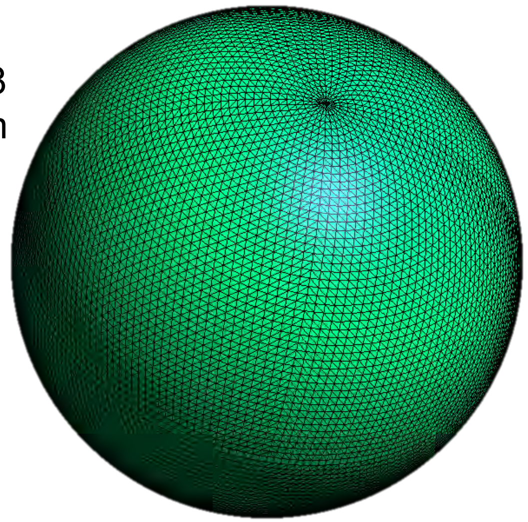


N80
~ 125 km





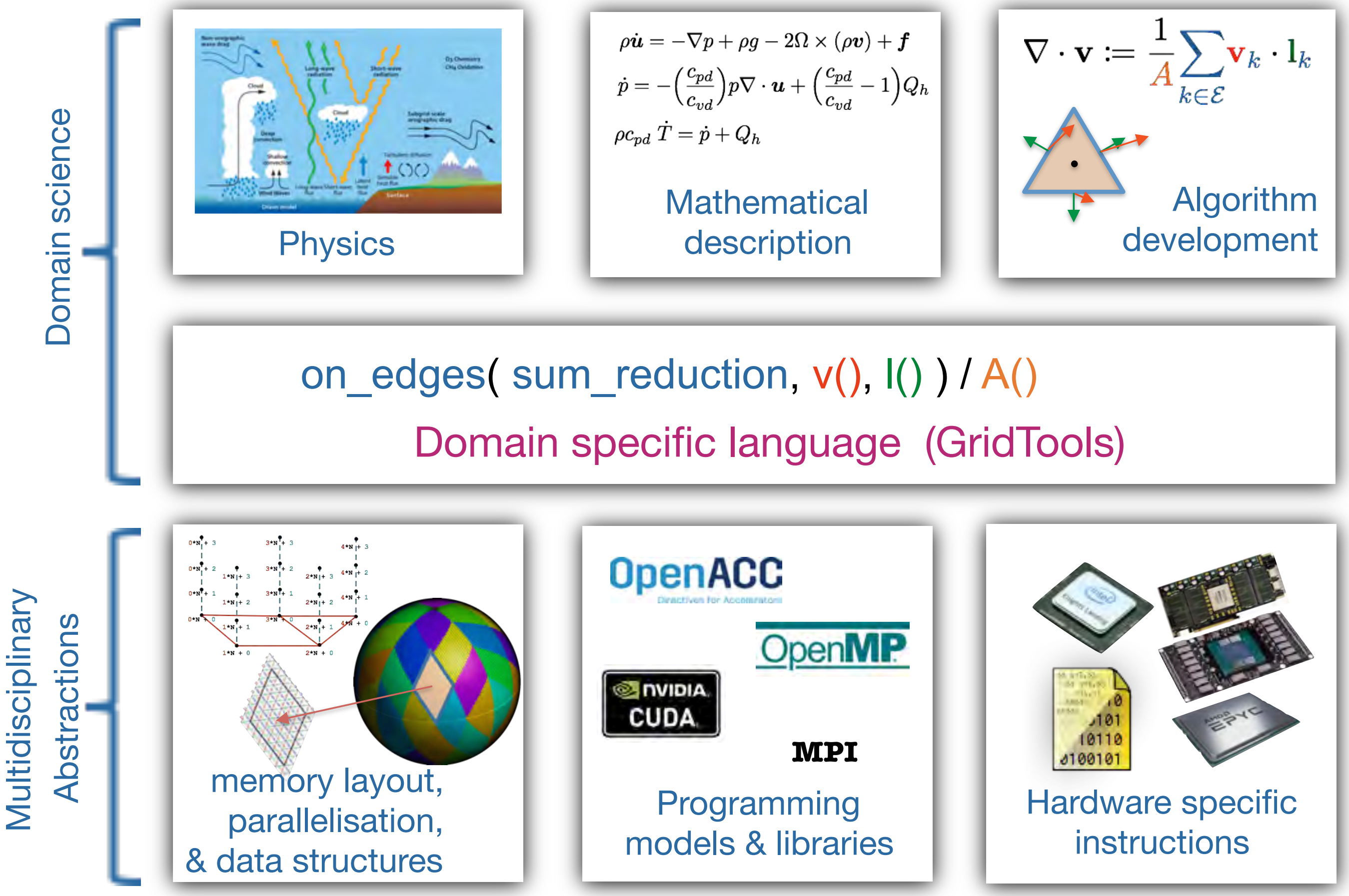
O48
~ 200 km



Atlas – GridTools integration

What is GridTools?

GridTools is a Domain-Specific-Language (DSL), developed by MeteoSwiss and CSCS, to express computational kernels that can be run on different hardware (CPU / GPU / MIC-KNL).



Atlas – GridTools integration

- Prior to ESCAPE, GridTools only worked on **specific regular grids** as used in COSMO
- In ESCAPE we extended this for **unstructured meshes based on Atlas**

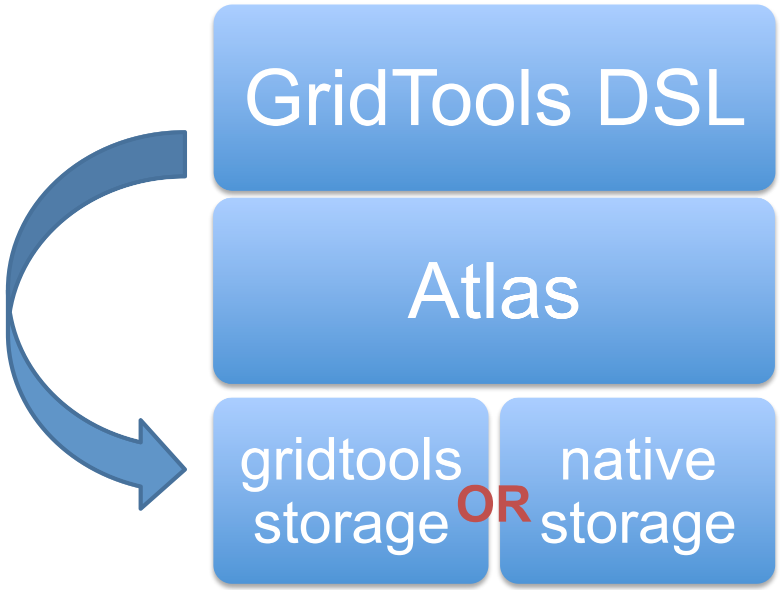
- GridTools has an internal data-storage framework that handles copying the data back-and-forth between the host (CPU) and device (e.g. GPU), and the DSL uses this extensively.

- **Challenge: How can we allow the GridTools DSL to recognise Atlas fields instead?**

Approach

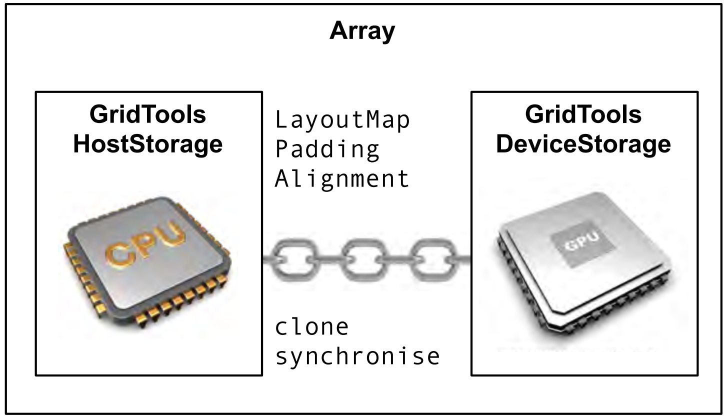
- Encapsulate the gridtools_storage features as a standalone module.

- Atlas Arrays and Fields can be compiled with gridtools_storage module for its internal storage.



Atlas on GPUs

- **Two linked memory spaces:**
  host (CPU) and device (GPU)
- **Built on GridTools storage layer**



Thanks to Carlos Osuna (MeteoSwiss)

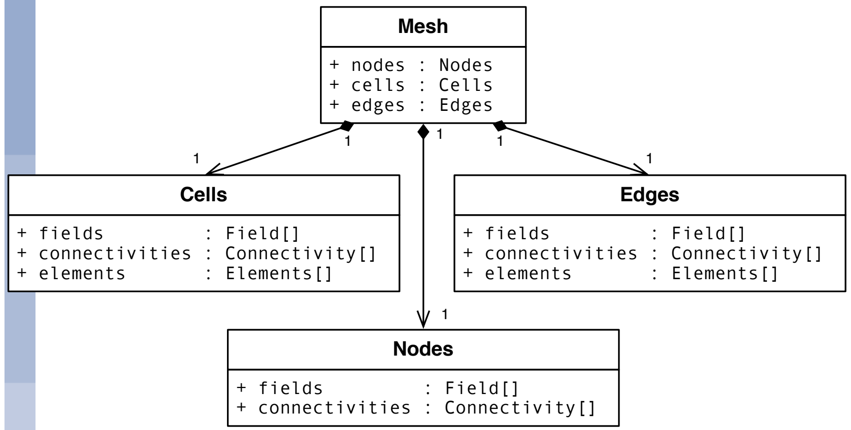
C++ example

```
// Create field (double precision, with 2 dimensions)
auto field = Field( datatype("real64"), shape(Ni,Nj) );

// Create a host view to interpret as 2D Array of doubles
auto host_view = make_host_view<double,2>(field);

// Modify data on host
for ( int i=0; i<Ni; ++i ) {
  for ( int j=0; j<Nj; ++j ) {
    host_view(i,j) = ...
}}

// Synchronise memory-spaces
field.syncHostDevice();

// Create a device view to interpret as 2D Array of doubles
auto device_view = make_device_view<double,2>(field);

// Use e.g. CUDA to process the device view...
some_cuda_kernel<<<1,1>>>(device_view);

// ... or GridTools or Kokkos
```

Atlas on GPUs with OpenACC for Fortran

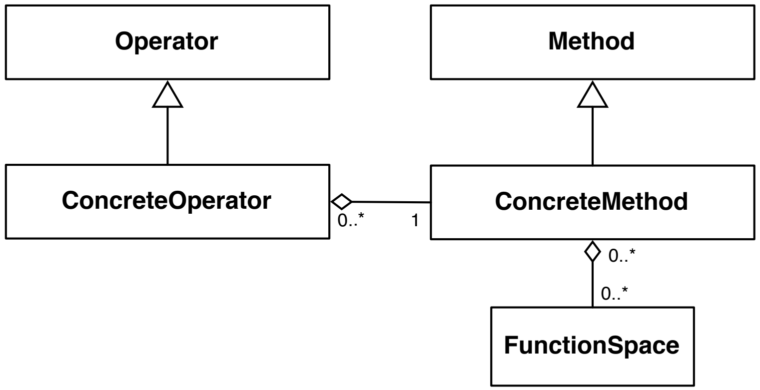
- **GPU enabled data structures**

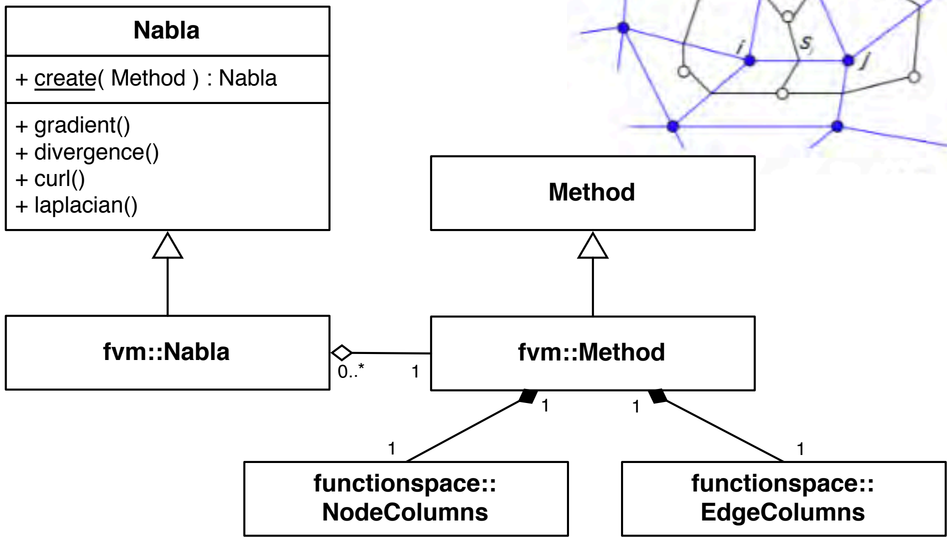
- Cloning mesh to device recursively clones all encapsulated components to device

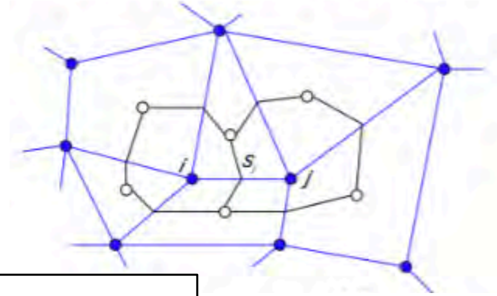



```fortran
type(atlas_Mesh) :: mesh             ! Assume created
type(atlas_mesh_Nodes) :: nodes ! Nodes in the Mesh
type(atlas_Field) :: field_xy    ! Coordinate field of nodes
real(8), pointer :: xy           ! Raw data pointer

!-----------------------------------------------------------
!

nodes    = mesh % nodes()            ! Access nodes
field_xy = nodes % xy()              ! Access coordinate field
call make_view( field_xy, xy )       ! Access raw data

call mesh % update_device()          ! Copy entire mesh to GPU

!$acc data present(xy)
!$acc kernels
do j=1,nodes % size()                ! Operate on GPU data
   xy(1,j) = ...                     ! e.g. modify X-coordinate
enddo
!$acc end kernels
!$acc end data

call field_xy % update_host()        ! Update changed field
```

Mathematical operations







Example Fortran code to create and apply the Nabla operator to compute gradient and laplacian

```
method = atlas_fvm_Method( mesh, levels=137 )
nabla  = atlas_Nabla( method )

call nabla%gradient ( scalarfield, gradfield )
call nabla%laplacian( scalarfield, laplfield )
```

Spectral Transforms in Atlas

C++ example

```
Grid grid("O1280");

StructuredColumns gp ( grid, levels(137) );
Spectral sp( 1279, levels(137) );

Field gpfield = gp.createField<double>();
Field spfield = sp.createField<double>();

Trans trans( gp, sp );   // TCo1279

trans.dirtrans( gpfield, spfield );
trans.invtrans( spfield, gpfield );
```

- Grid (Octahedral Gaussian O1280)
- FunctionSpaces gp and sp
- Creation of fields through FunctionSpace
- Internally sets up IFS-trans
- Transforms

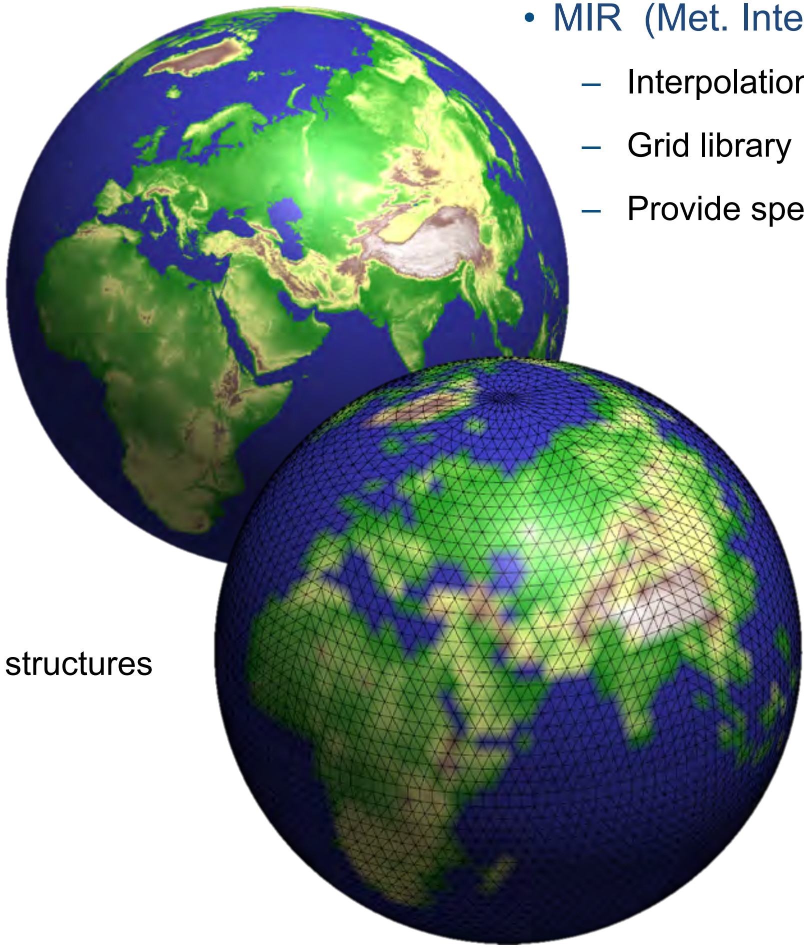
MIR

Similar in Fortran

FVM

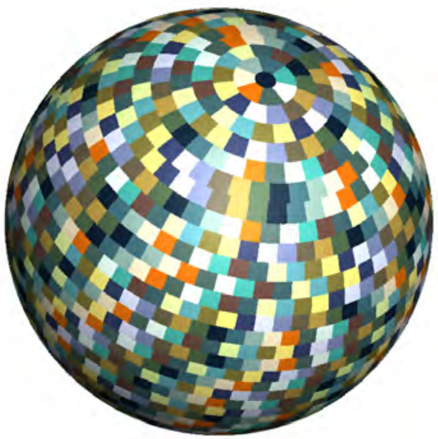
Atlas not the solution (i.e. not the library to develop in), but enabling new research

- ESCAPE dwarfs
  - Object Oriented data structures
  - LAM grids
  - GPU aware memory storage
    - IFS (not operational)
      - Grid-point derivatives
      - Parallel interpolations
      - Multiple grids / coupling



- MIR (Met. Interpol. & Regrid.)
  - Interpolation
  - Grid library
  - Provide spectral transforms

  - MARS
  - MetView
  - prodgen



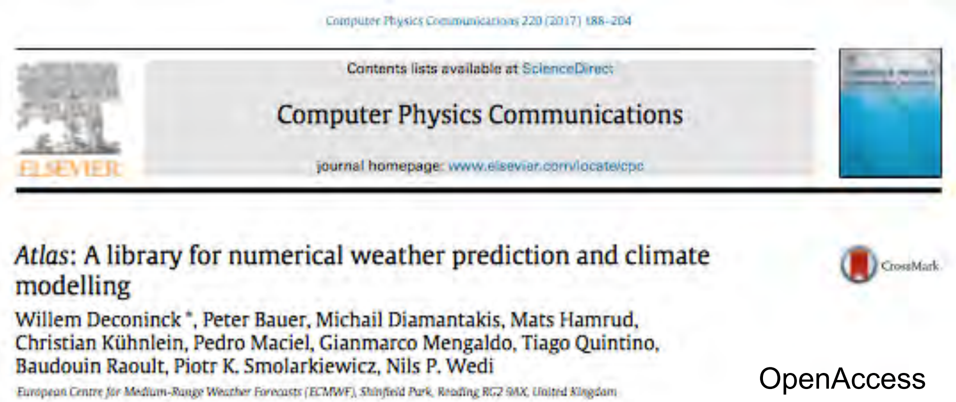
- FVM
  - Object Oriented data structures
  - Parallelisation

Atlas, a library for NWP and climate modelling

5th ENES HPC Workshop
17-18 May 2018, Lecce

By Willem Deconinck

willem.deconinck@ecmwf.int


Computer Physics Communications 220 (2017) 188–204

Contents lists available at ScienceDirect

Computer Physics Communications

journal homepage: www.elsevier.com/locate/cpc

ELSEVIER

Atlas: A library for numerical weather prediction and climate modelling

Willem Deconinck *, Peter Bauer, Michail Diamantakis, Mats Hamrud, Christian Kühnlein, Pedro Maciel, Gianmarco Mengaldo, Tiago Quintino, Baudouin Raoult, Piotr K. Smolarkiewicz, Nils P. Wedi

European Centre for Medium-Range Weather Forecasts (ECMWF), Shinfield Park, Reading RG2 9AX, United Kingdom

OpenAccess

ESCAPE has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement no. 671627.

Thank you!

ESCAPE

FUNDED BY
THE EUROPEAN UNION

© ECMWF maggio 18, 2018