Atlas, a library for NWP and climate modelling

EWGLAM / SRNWP October 2017, Reading

By Willem Deconinck
willem.deconinck@ecmwf.int

Thanks to: Daan Degrauwe[2], Michail Diamantakis[1], Christian Kühnlein[1], Pedro Maciel[1], Gianmarco Mengaldo[1], Andreas Müller[1], Carlos Osuna[3]

[1] : ECMWF ; [2] : RMI ; [3] : MeteoSwiss

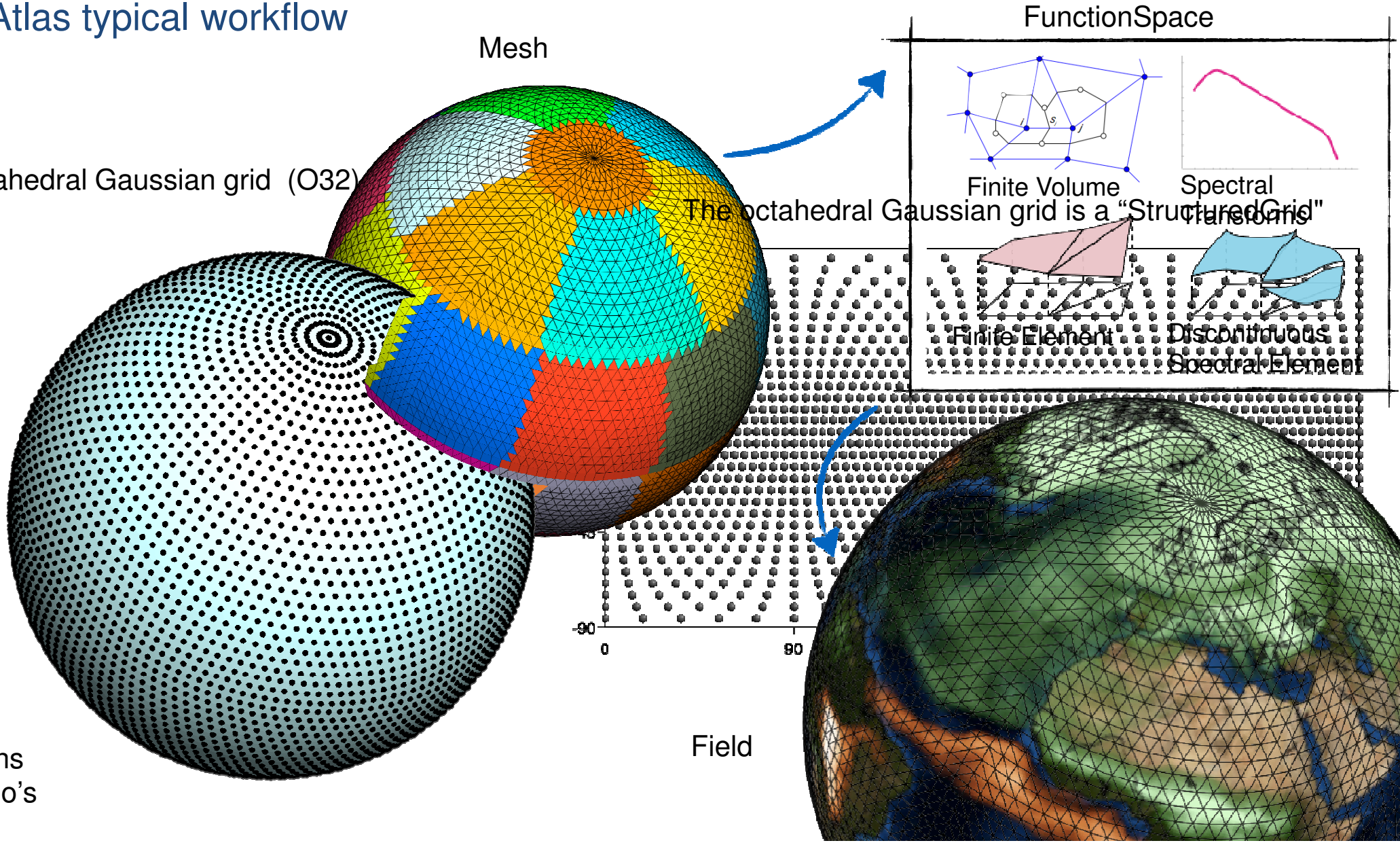
ECMWF

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

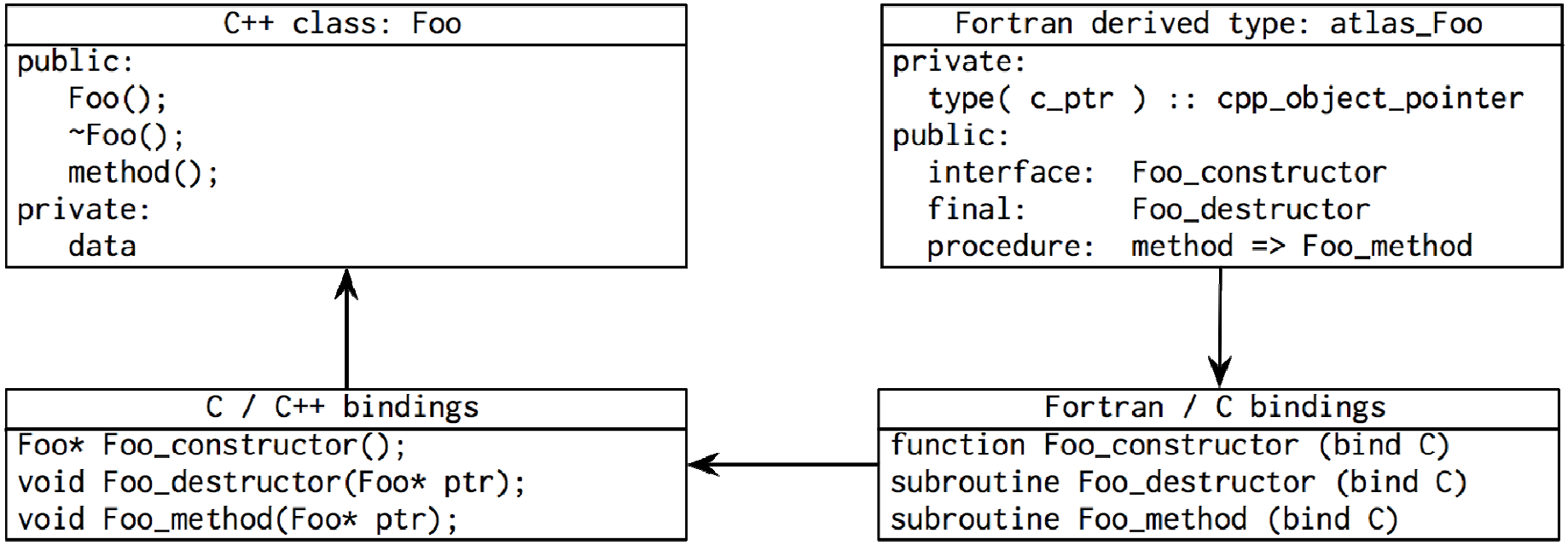
- A new foundation built with future challenges for HPC in mind
- Modern C++ library implementation with modern Fortran 2008 (OOP) interfaces → integration in existing models
- Open-source, very soon publicly available
- Data structures to enable new numerical algorithms, e.g. based on unstructured meshes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

ECMWF

Atlas typical workflow



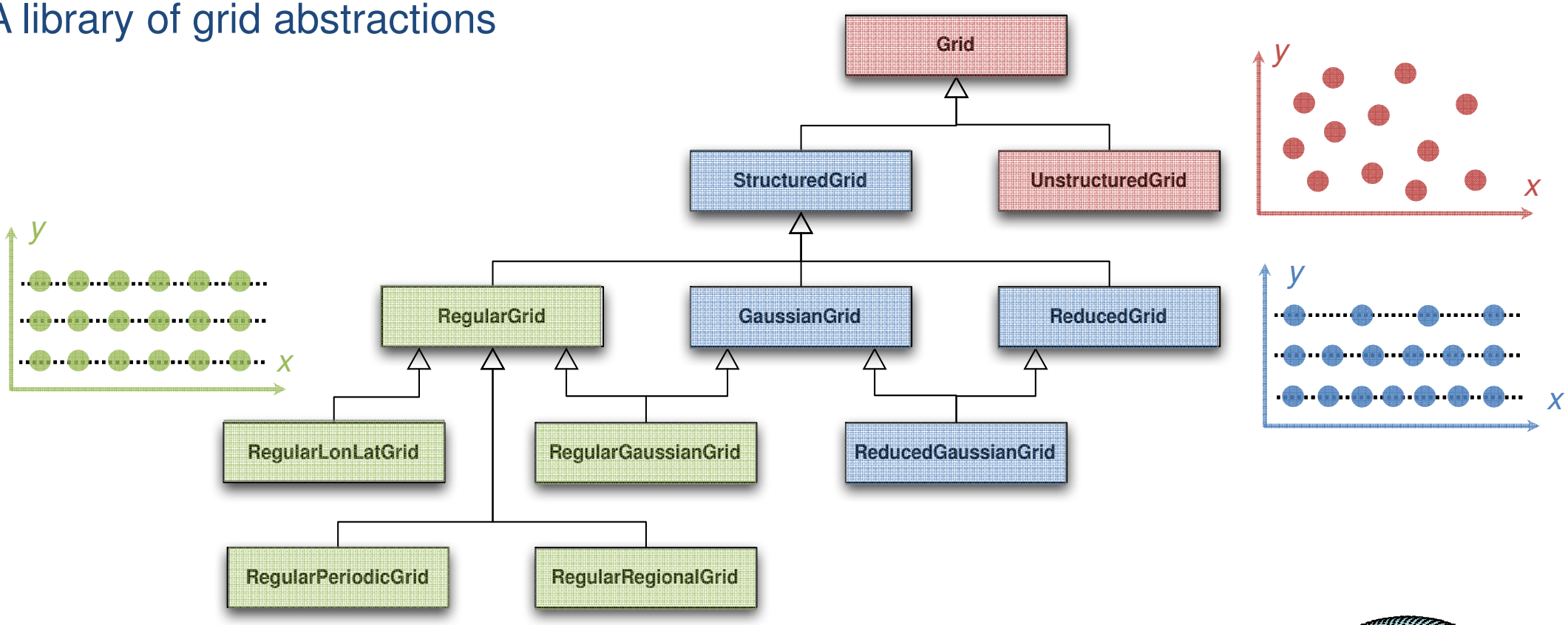
Object oriented design C++ / Modern Fortran




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

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

A library of grid abstractions



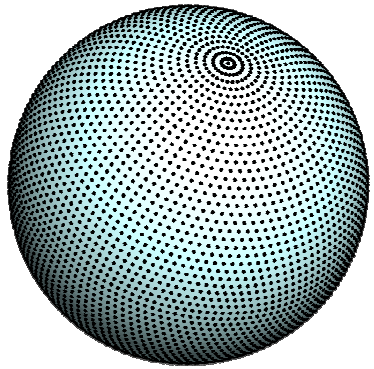
Example creation of operational octahedral reduced Gaussian grid unique identifier

C++

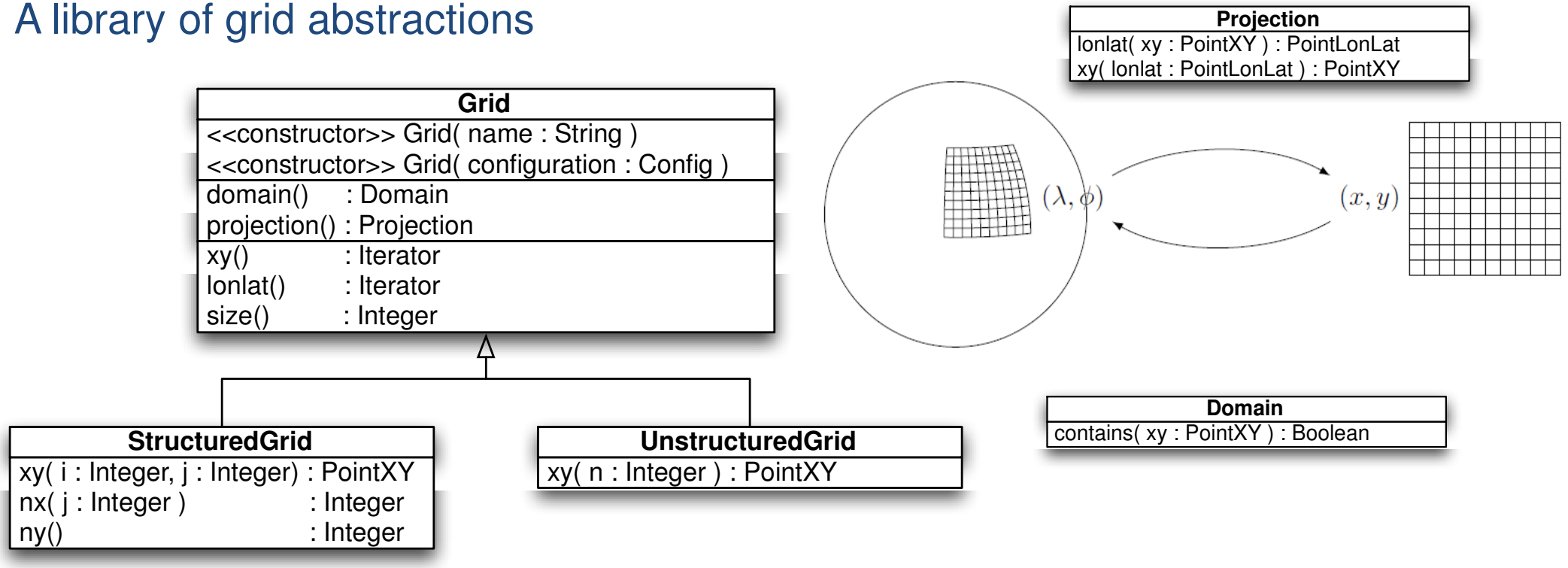
```
atlas::Grid grid;
grid = atlas::Grid ( “O1280” )
```

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

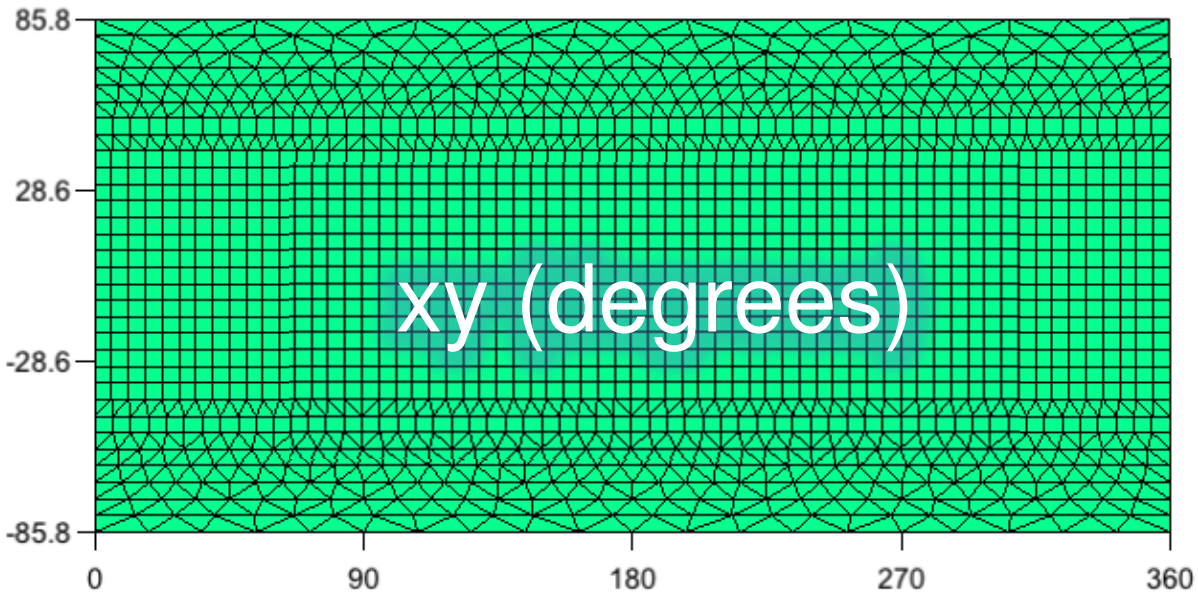
Grid coordinates (x,y)

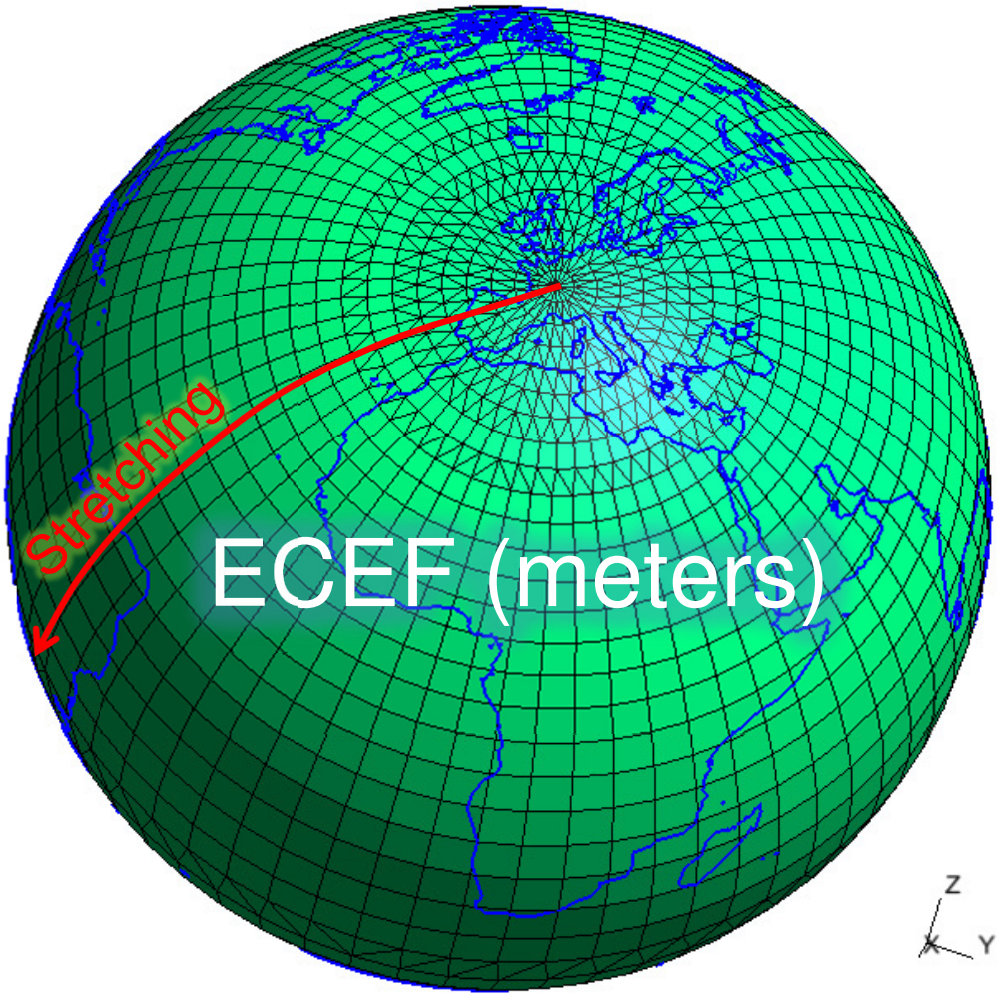
```
for( PointXY p : grid.xy() ) {
    double x = p.x();
    double y = p.y();
}
```

Geographic coordinates (lon,lat)

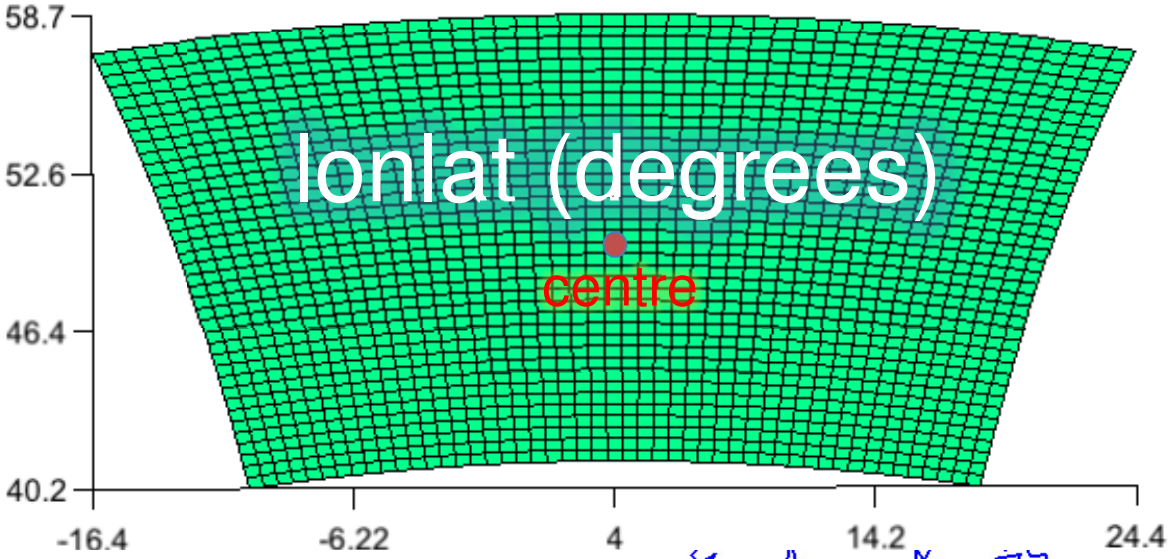
```
for( PointLonLat p : grid.lonlat() ) {
    double lon = p.lon();
    double lat = p.lat();
}
```

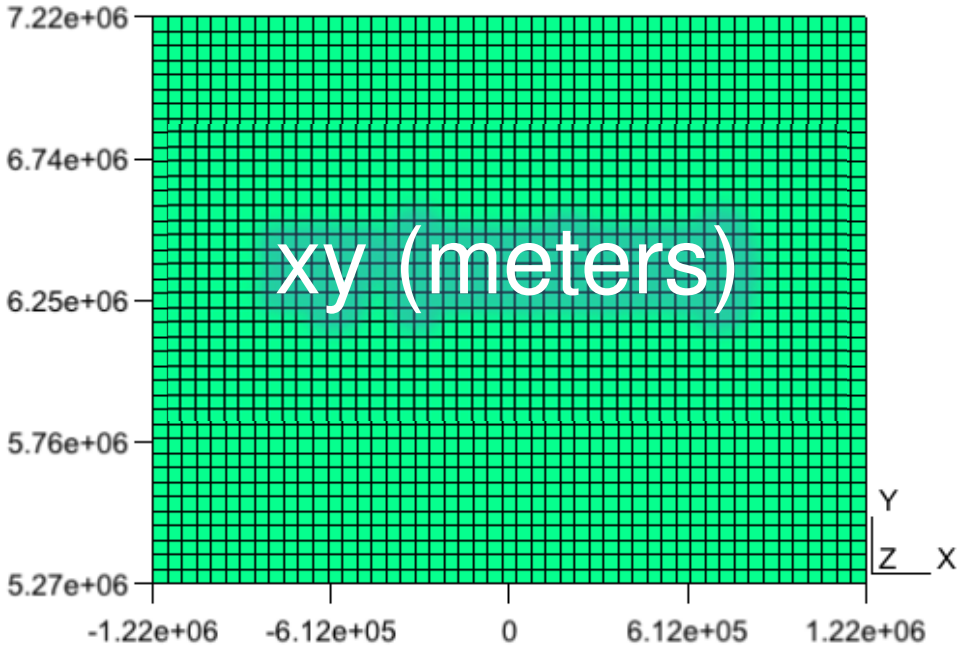
```
{
   "type" : "classic_gaussian",
   "N" : 16
   "projection" : {
      "type" : "rotated_schmidt",
      "north_pole"  : [3,47]
      "stretching_factor" : 2
   }
}
```

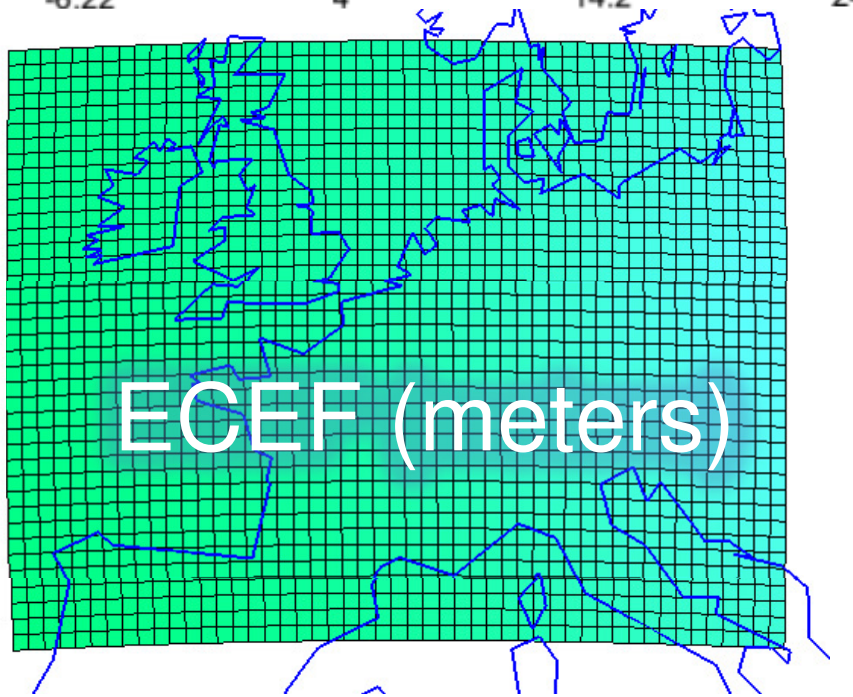




```
{
   "type" : "regional",
   "nx" : 50,   "dx" : 50000,
   "ny" : 40,   "dy" : 50000,
   "lonlat(centre)" : [4,50],
   "projection" : {
      "type" : "lambert",
      "latitude1"  : 50,
      "longitude0" : 4
   }
}
```





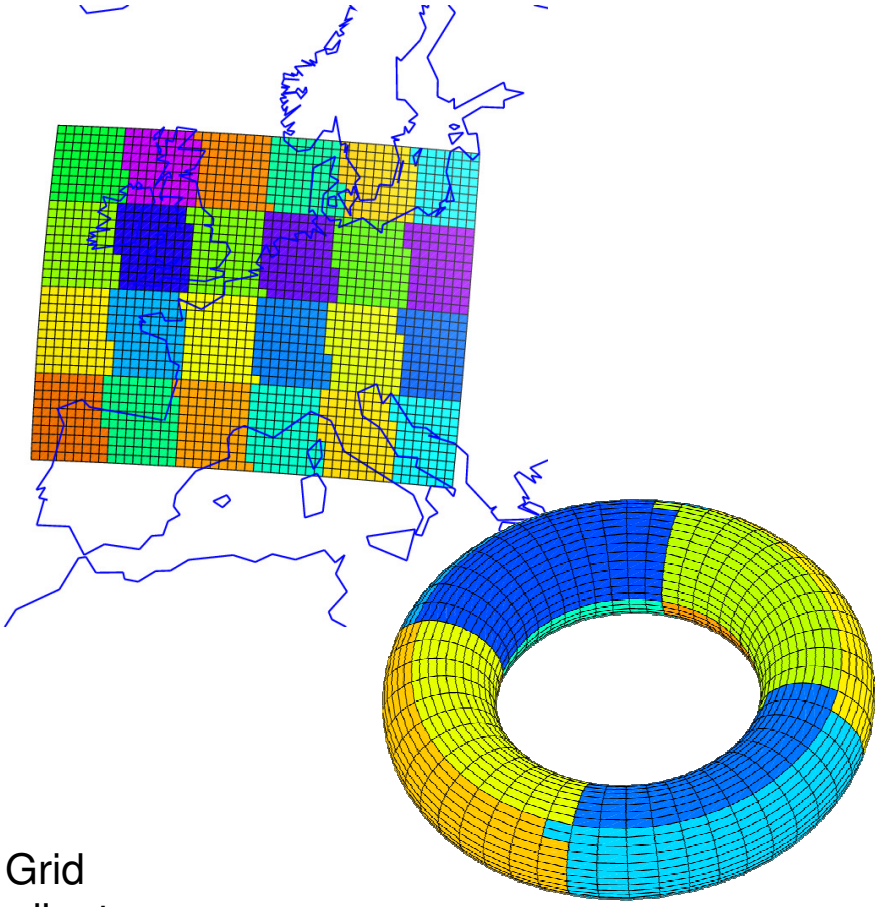


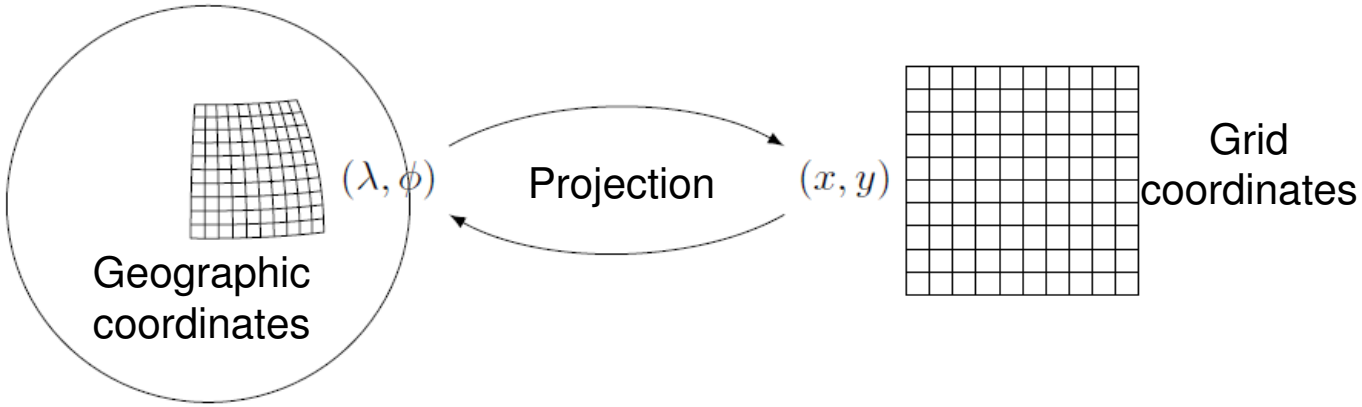
Atlas for LAM

Thanks to Daan Degrauwe (RMI)

Current capabilities:

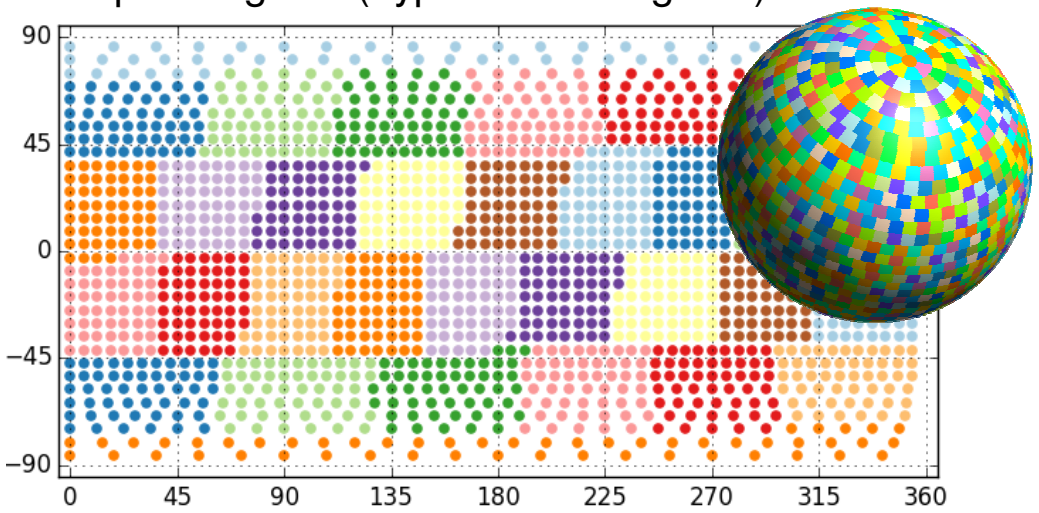
- Generation of parallel meshes for LAM grids
- Projections: Mercator, Lambert, Schmidt, Polar stereographic
  + Rotations thereof if applicable
- Checkerboard domain decomposition
- Periodicity in X and Y (biperiodic)



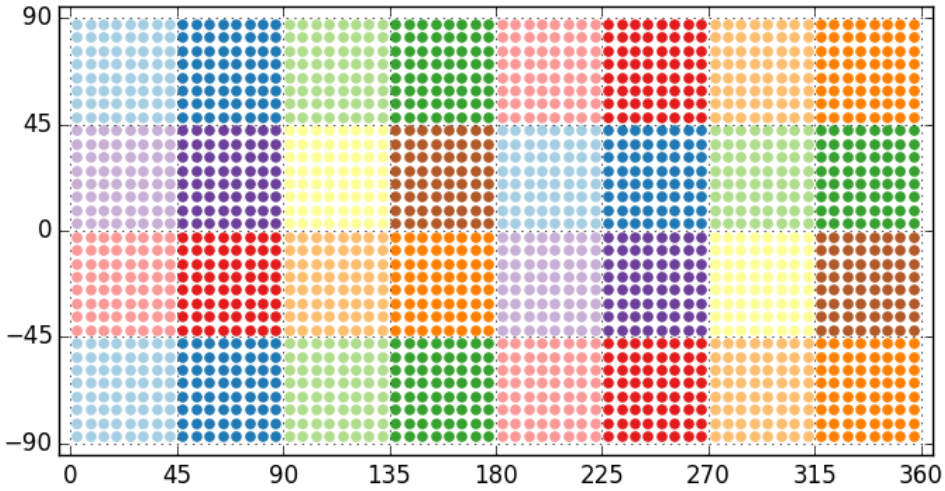


Multiple domain decomposition strategies

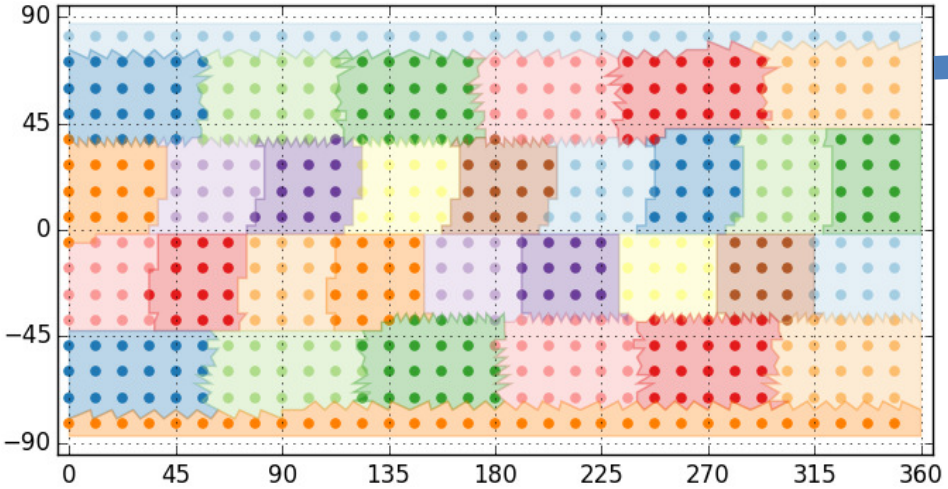
Equal Regions (typical for IFS grids)



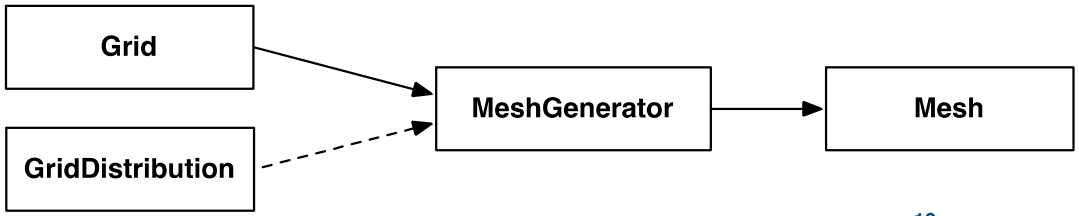
Checkerboard (typical for regional grids)



Matching Mesh

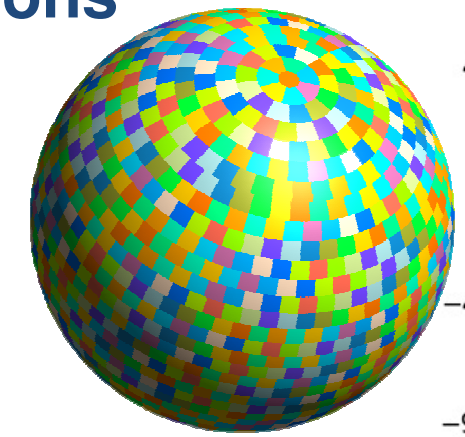


Any grid can follow the domain decomposition of an already distributed mesh

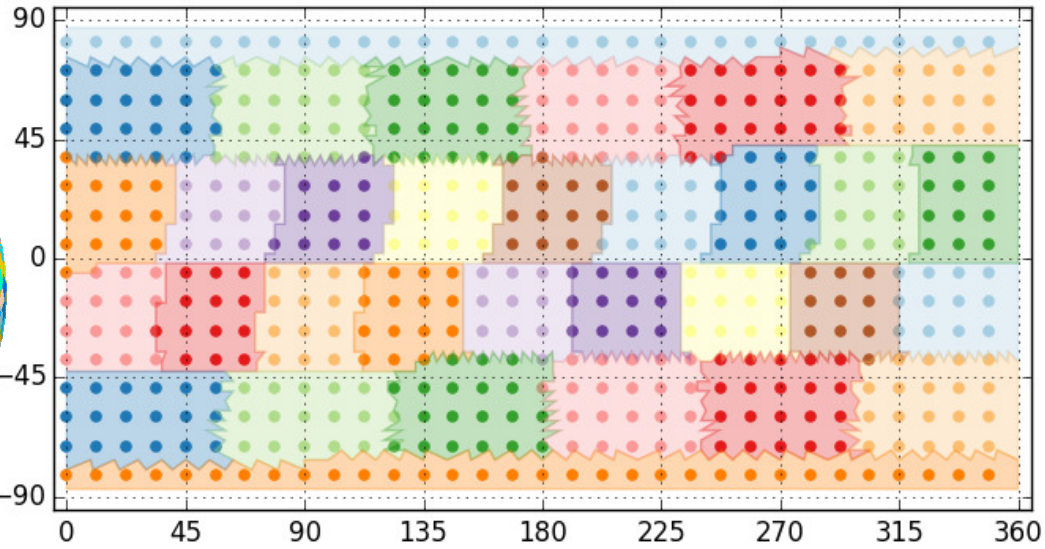


Remapping, using matching domain decompositions

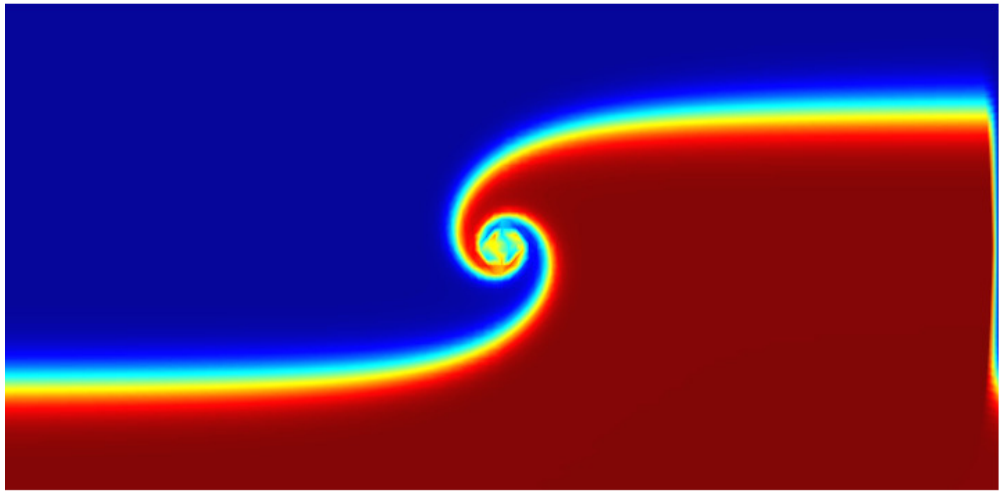
- Domain decompositions match to avoid MPI communication
- Parallel interpolation:
  - nearest neighbour
  - k-nearest neighbour
  - linear element-based

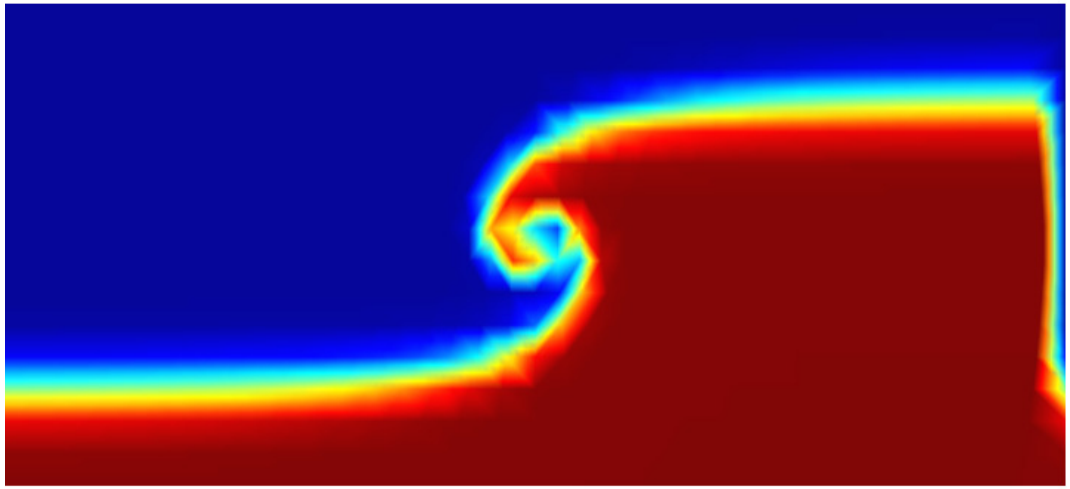


Thanks to Pedro Maciel (ECMWF)

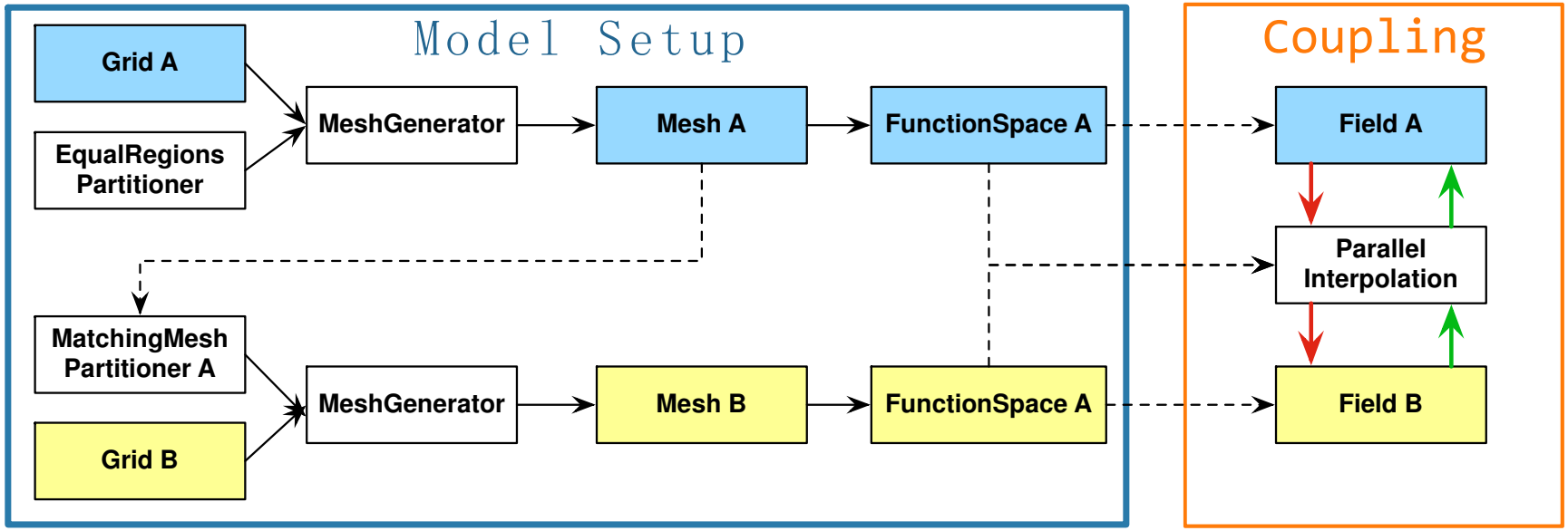


Example interpolation: O32 to F8





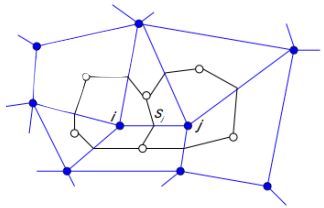
Remapping using matching domain decompositions



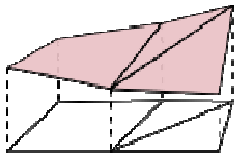

```
grid_A          = atlas_Grid("O1280")
partitioner_A   = atlas_EqualRegionsPartitioner()
mesh_A          = meshgenerator % generate(grid_A, partitioner_A)
fs_A            = atlas_functionspace_NodeColumns(mesh_A)

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)
```

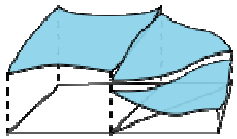
FunctionSpaces: Discretisation specific knowledge



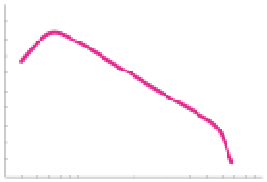
Finite Volume



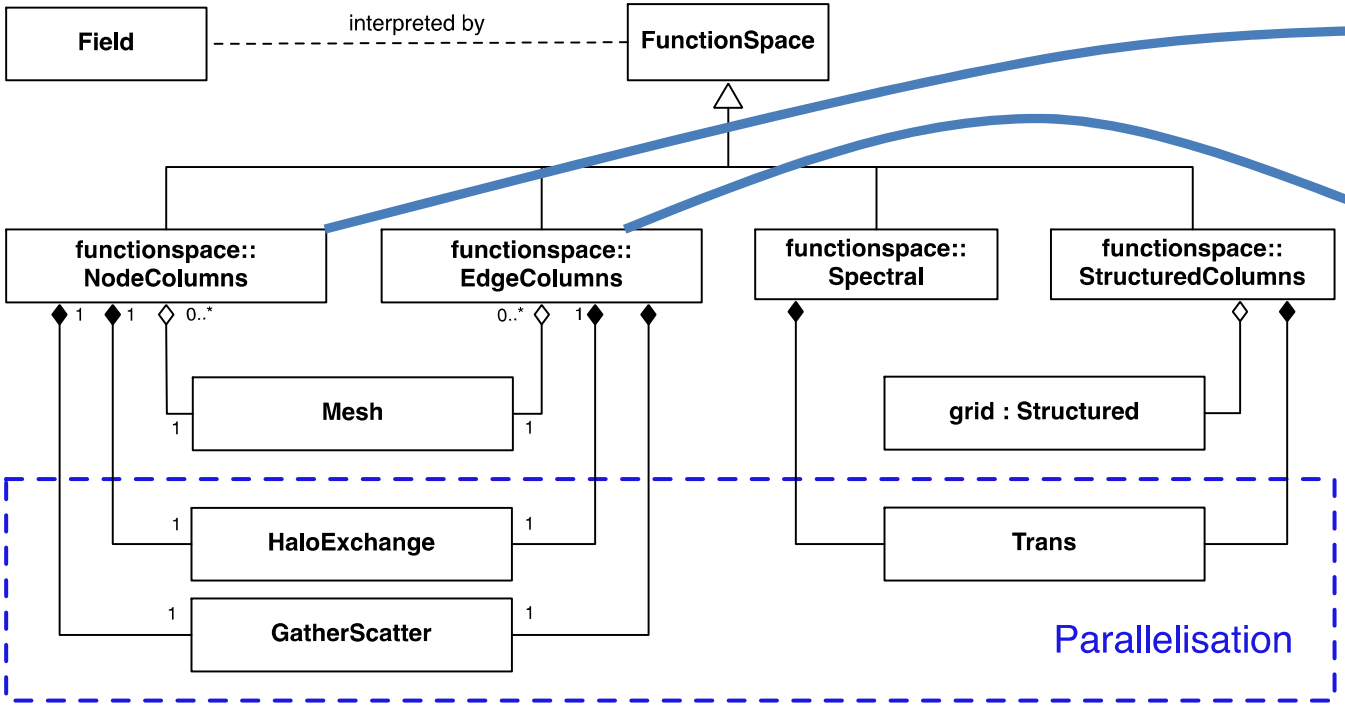
Finite Element

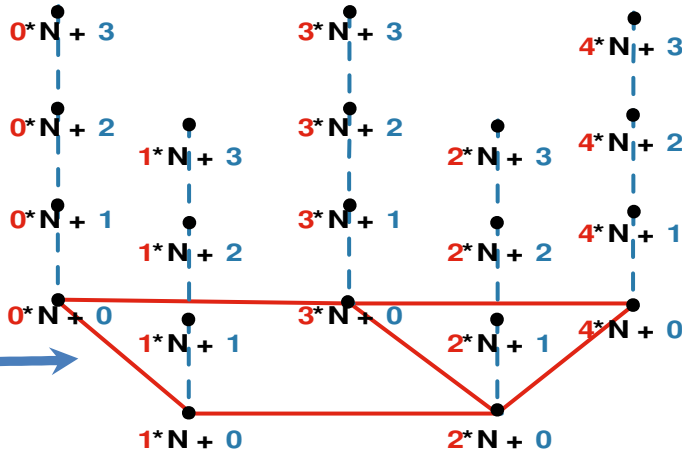


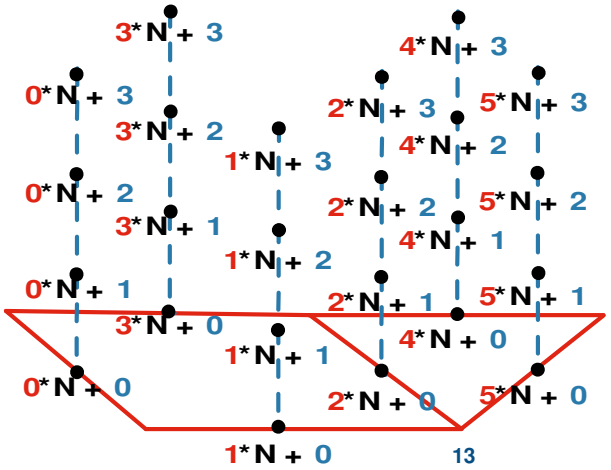
Discontinuous Spectral Element



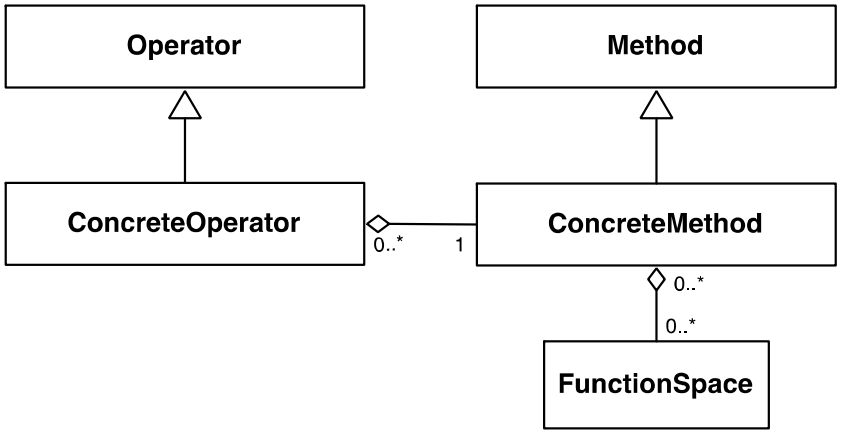
Spectral Transforms

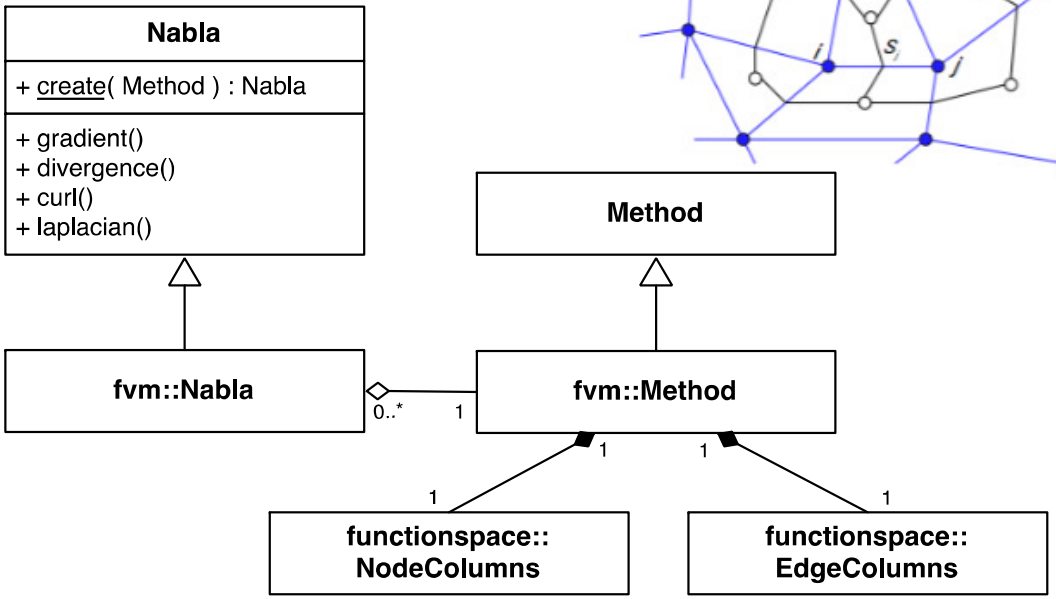


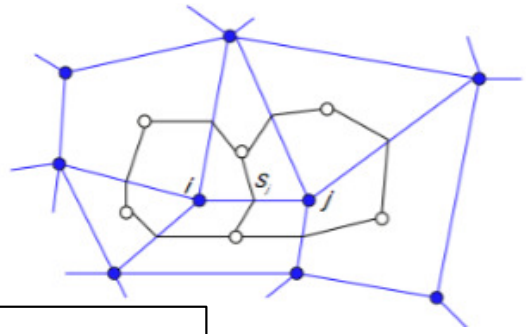




Mathematical operations





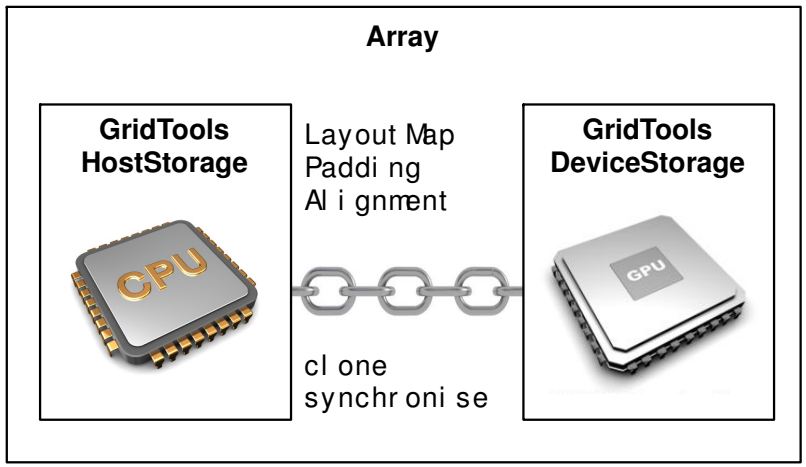


Example Fortran code to create and apply the Nabla operator to compute gradient and laplacian

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

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

Atlas on GPUs

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



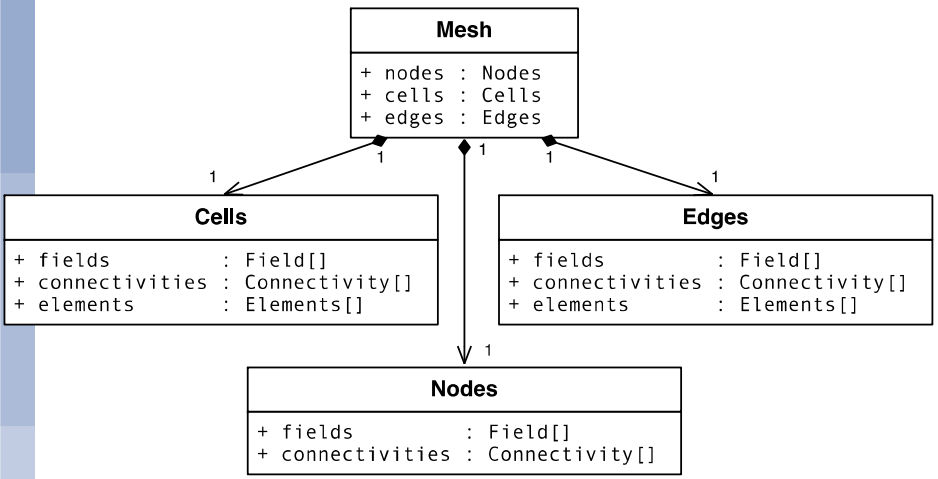

```
// Create field (double precision, with 2 dimensions)
auto field = Field( datatype("real64"), shape(Ni,Nj) );

// 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);
```

Thanks to Carlos Osuna (MeteoSwiss)

Atlas on GPUs

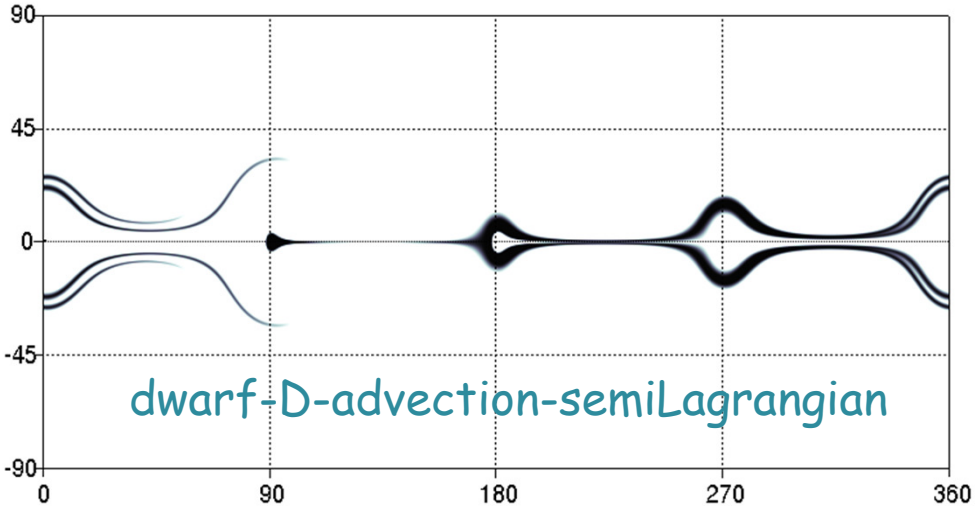
- **GPU enabled data structures**

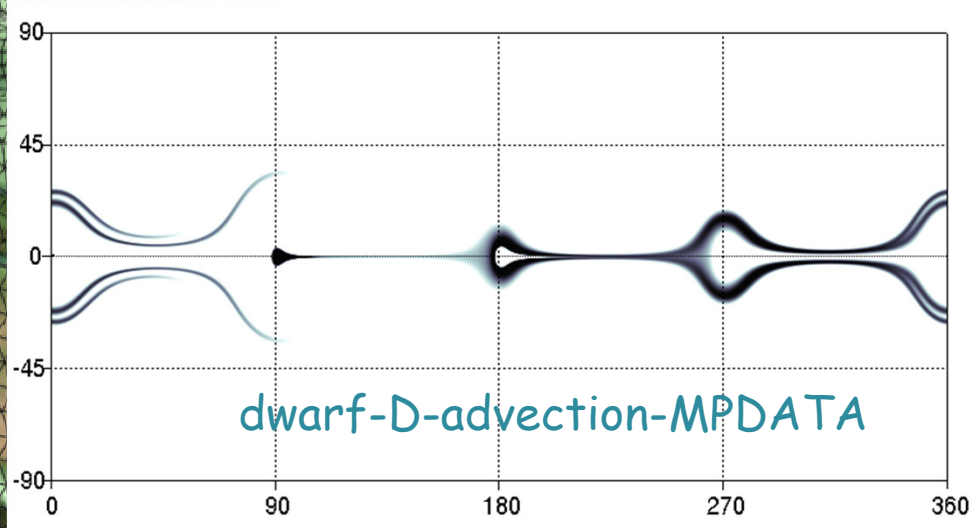
- Cloning mesh to device recursively clones all encapsulated components to device

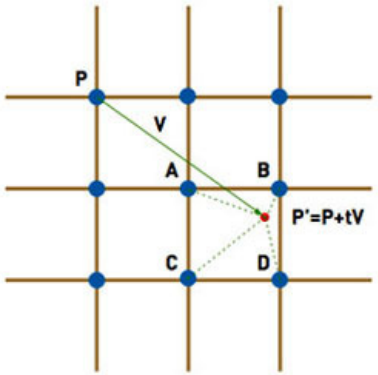


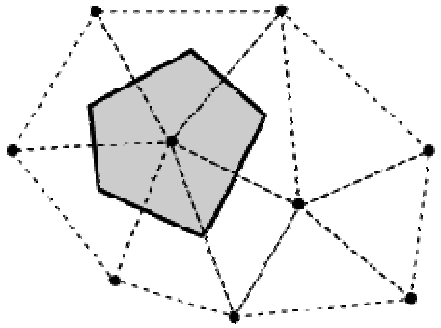

```
type(atlas_Mesh) :: mesh         ! Assume created
type(atlas_mesh_Nodes) :: nodes  ! Nodes in the Mesh
type(atlas_Field) :: field_xy    ! Coordinate field of nodes
real(8), pointer :: xy           ! Raw data pointer

nodes = mesh%nodes()
field_xy = nodes%xy()

call mesh%clone_to_device()     ! Copy entire mesh to GPU
call field_xy%device_data(xy)   ! GPU data pointer

!$acc data present(xy)
!$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 mesh%sync_host_device()
```

IFS and Advection dwarfs









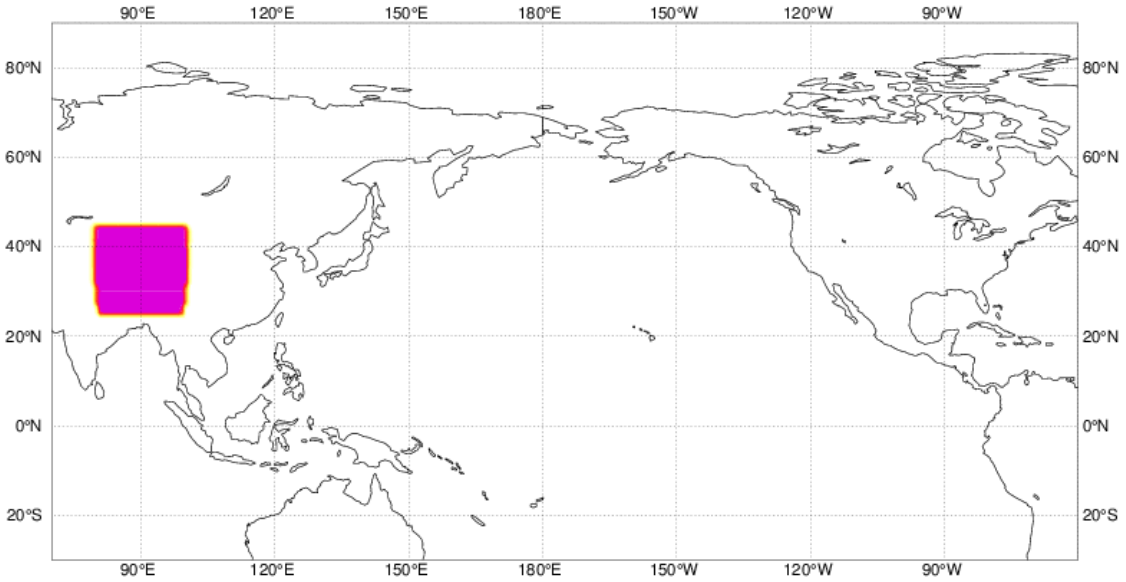
Advection abstraction in IFS based on Atlas

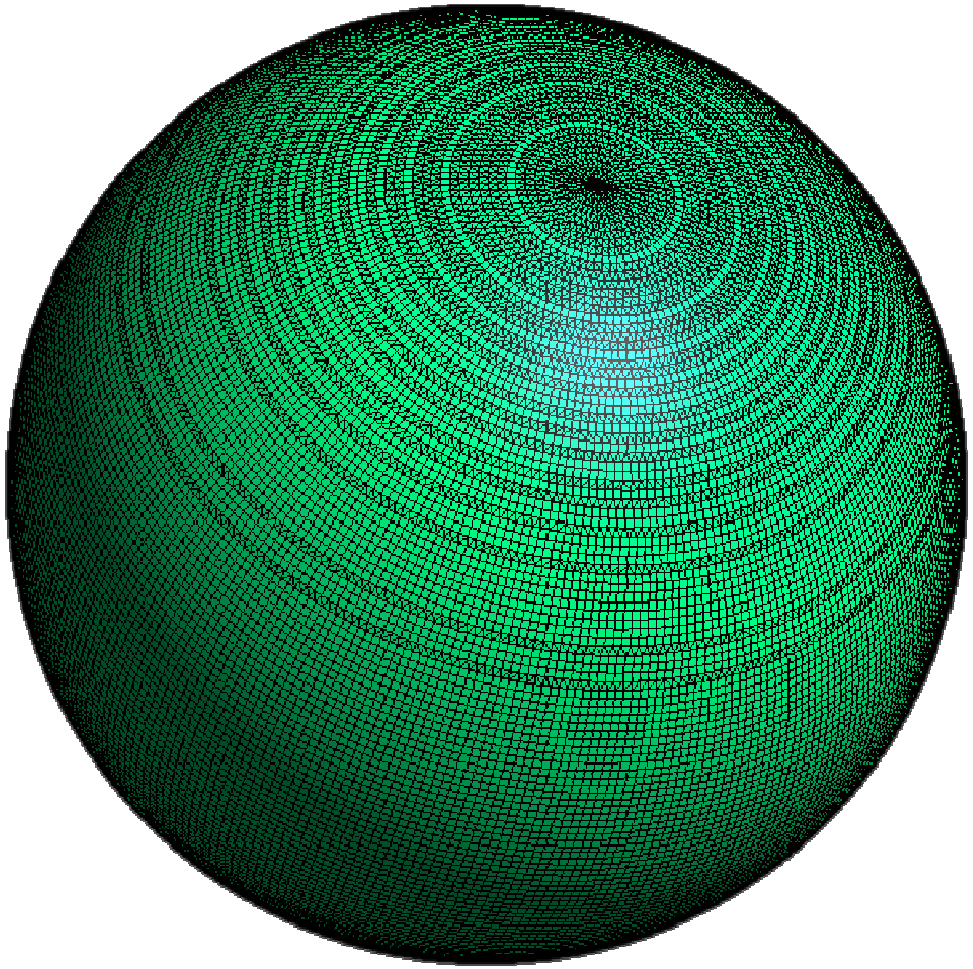
ECMWF

Thanks to Michail Diamantakis (ECMWF)

Passive tracer advection

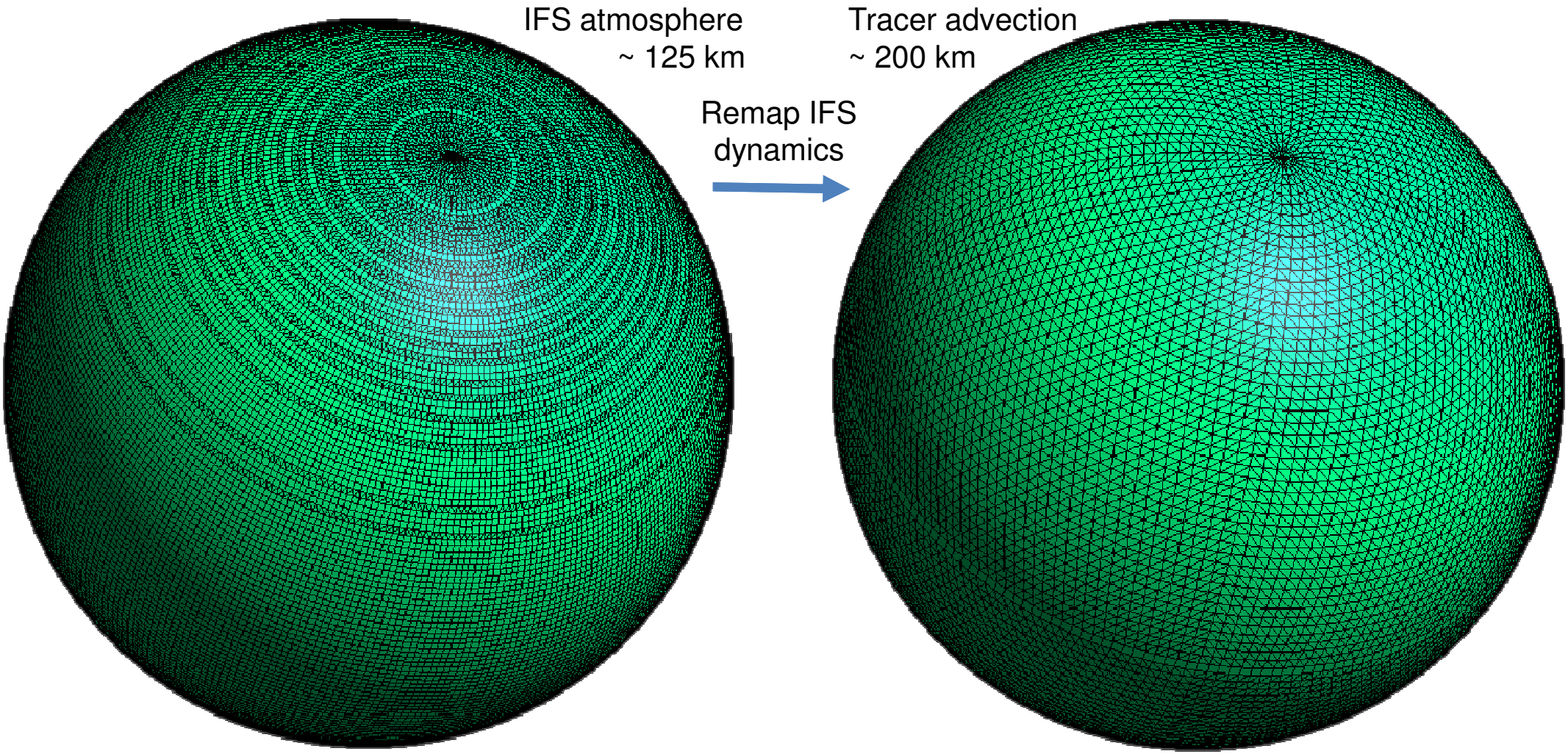
Tracer initialised over Asia – 125 km model resolution (N80)

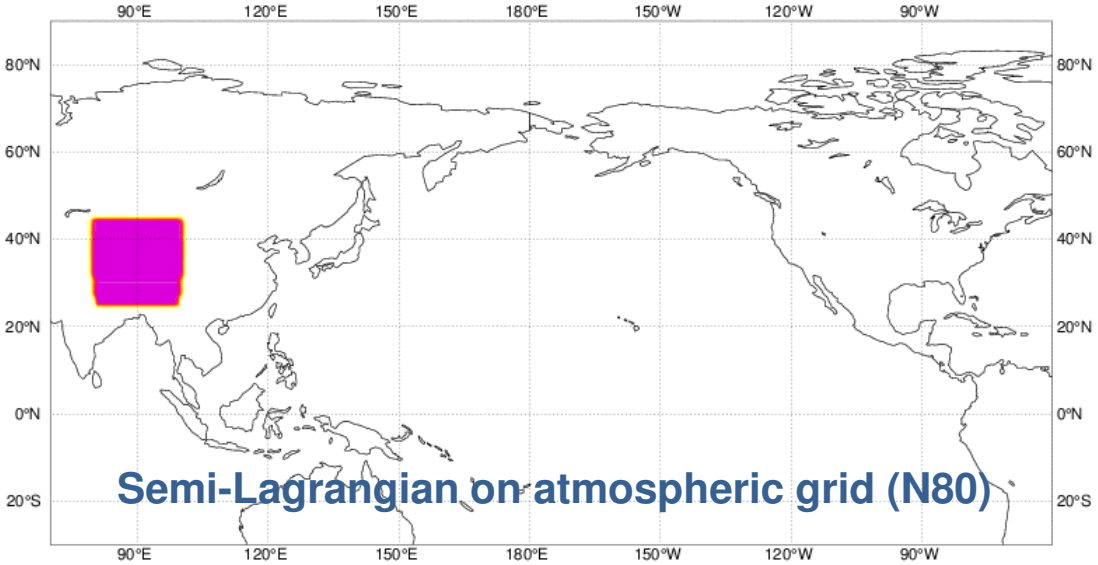




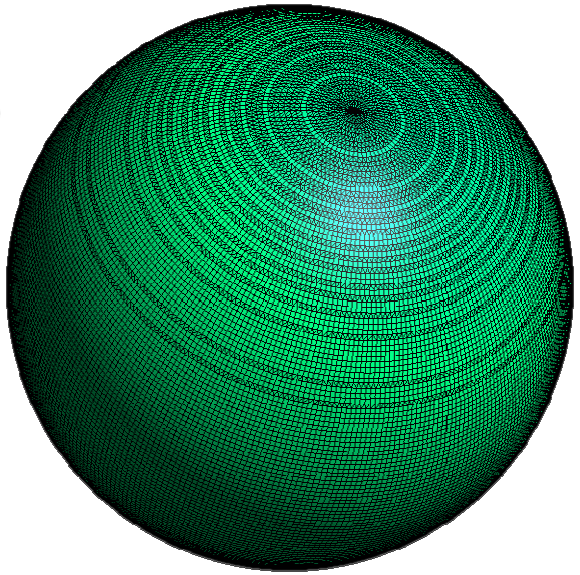
Very preliminary result,
proof of concepts

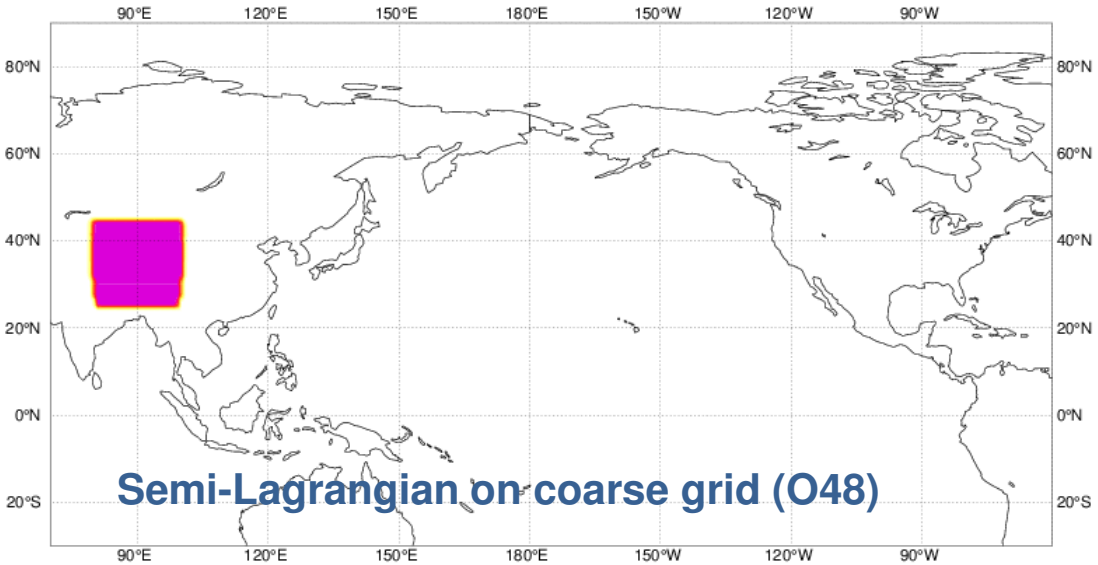
Can we use a coarser resolution for advection, but with wind from fine resolution?



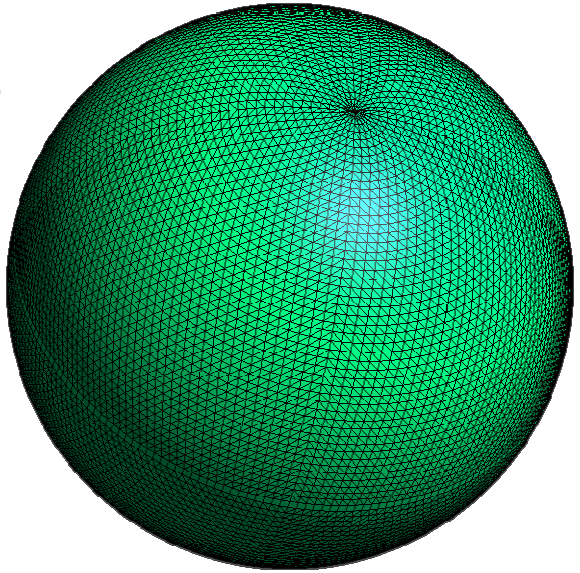


N80
~ 125 km



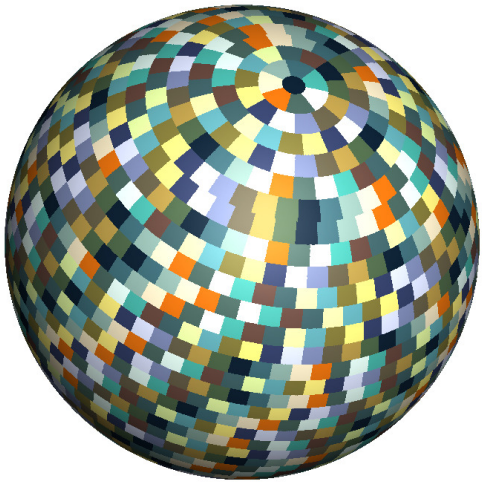


O48
~ 200 km

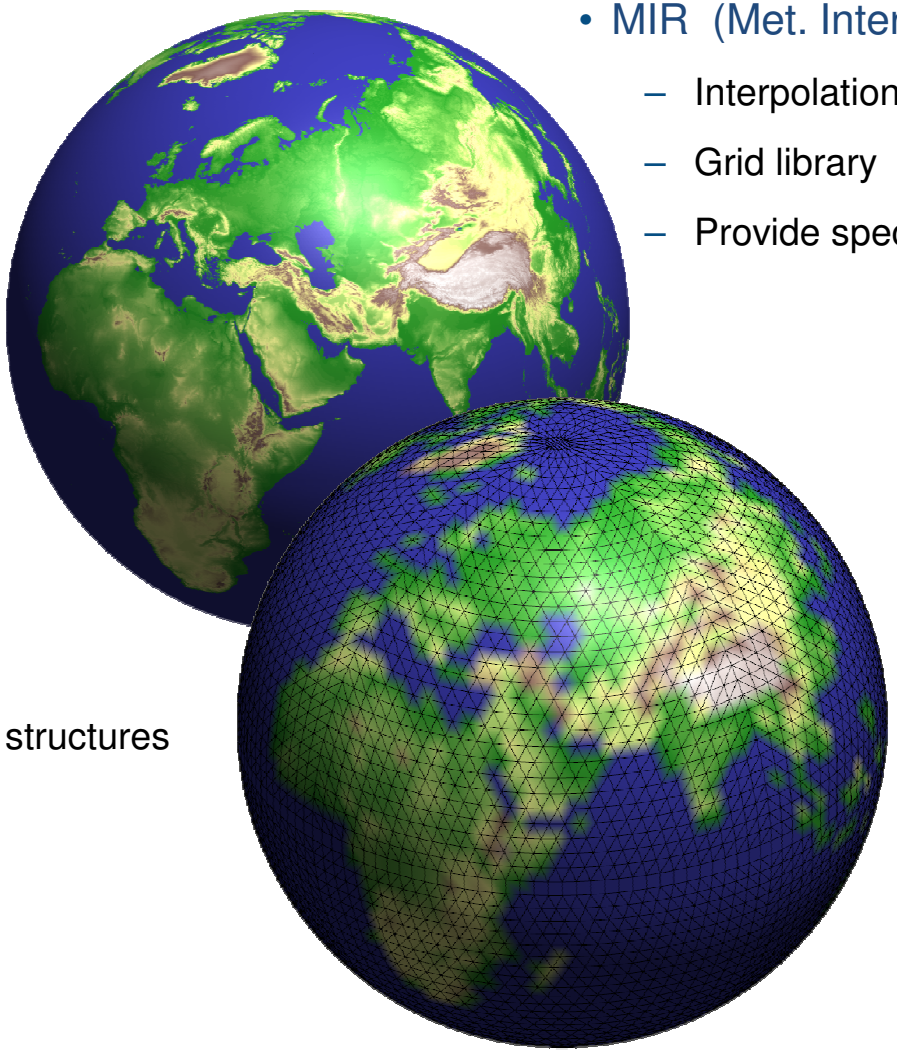


Atlas not the solution (i.e. not the library to develop in), but enabling new research

- ESCAPE dwarfs
  - Object Oriented data structures
  - LAM grids
  - GPU aware memory storage
    - IFS (currently optional)
      - Grid-point derivatives
      - Parallel interpolations
      - Multiple grids / coupling



- FVM
  - Object Oriented data structures
  - Parallelisation



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

- MARS
- MetView
- prodgen

Atlas, a library for NWP and climate modelling

EWGLAM / SRNWP October 2017

By Willem Deconinck

willem.deconinck@ecmwf.int

Thank you!

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

FUNDED BY
THE EUROPEAN UNION