Coupling Earth System Model Components using the Atlas library

Workshop on surface process coupling and its interactions with the atmosphere

Willem Deconinck, Slavko Brdar, Pedro Maciel
willem.deconinck@ecmwf.int

Bonn, Germany | 9-10 April 2025

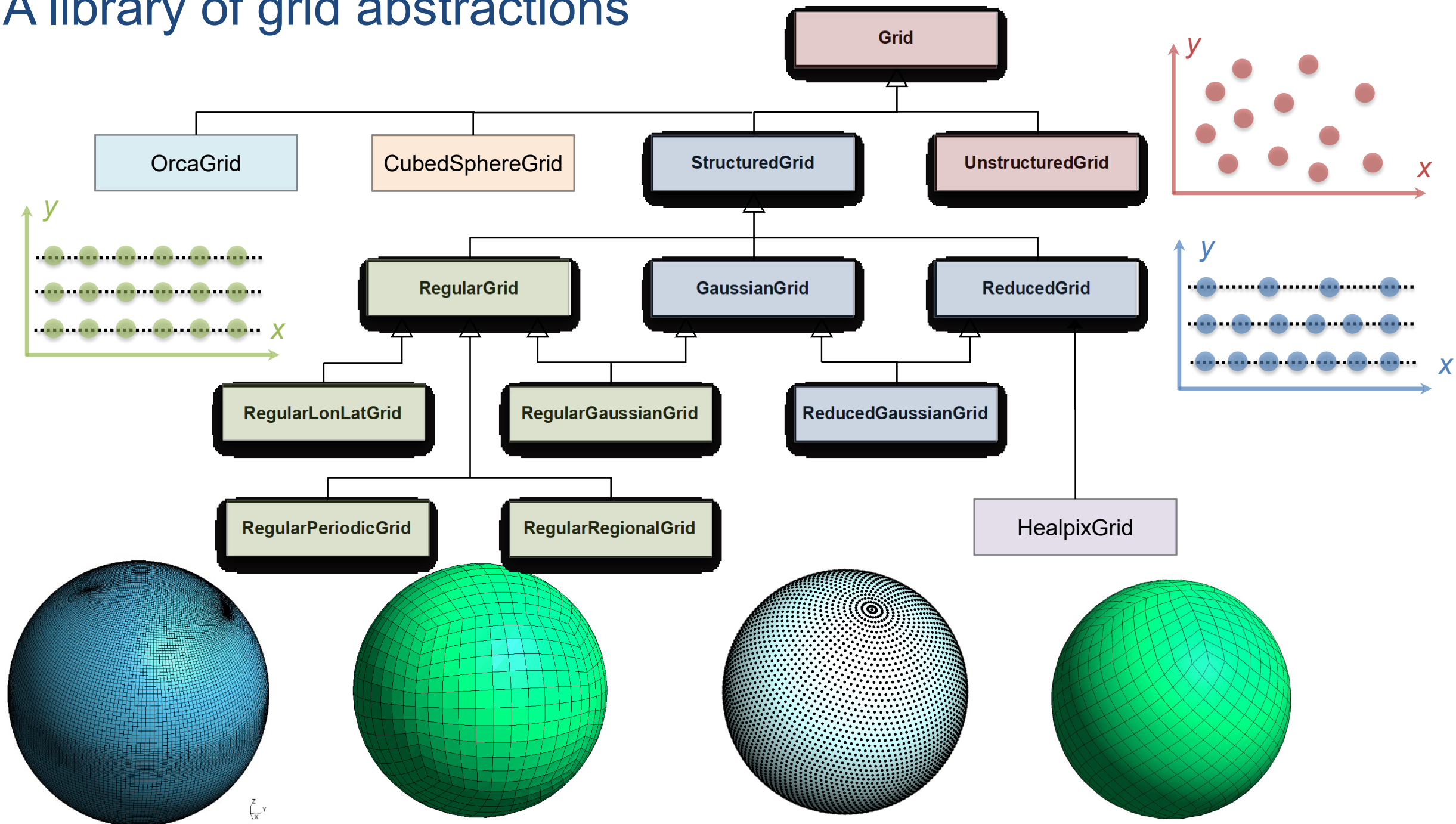
ECMWF

Funded by the European Union
Destination Earth implemented by ECMWF esa EUMETSAT

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

- Open source (Apache 2) : github.com/ecmwf/atlas
- Data structures to enable new numerical algorithms, e.g. based on unstructured meshes
  - Support structured and unstructured grids
  - Support **globa**l as well as **regional** grids
  - Grids with projections
- Separation of concerns
  - Grids / Mesh generation
  - Parallelisation (domain decomposition, halo exchanges)
  - Accelerator-aware data structures (GPU/CPU/…)
- Readily available operators
  - Remapping and interpolation
  - Gradient, divergence, laplacian
  - Spherical Harmonics transforms

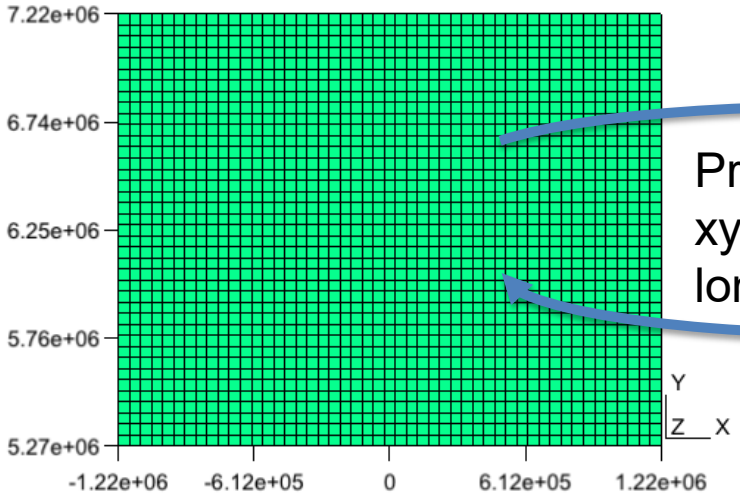
A library of grid abstractions



Support for Projections and regional grids

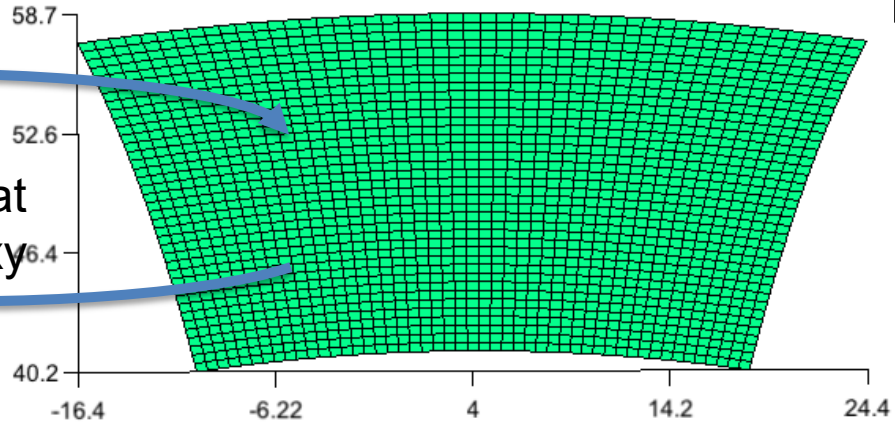
Grid coordinates

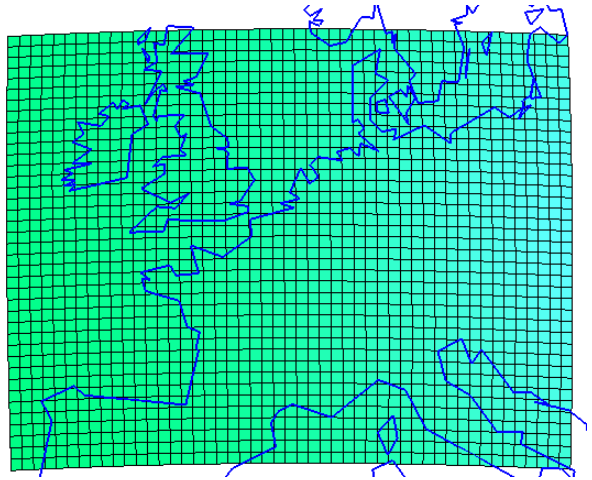
Lambert projection → metres



Projection:
xy_to_lonlat
lonlat_to_xy

Geographic coordinates (degrees)





Grid creation:

```
Grid grid ( "O1280" );
```

Sample of grid identifiers:

O1280	octahedral reduced gaussian grid
H256	healpix grid
eORCA12_T	extended ORCA 1/12 deg ocean grid, T arrangement

Grid iteration:

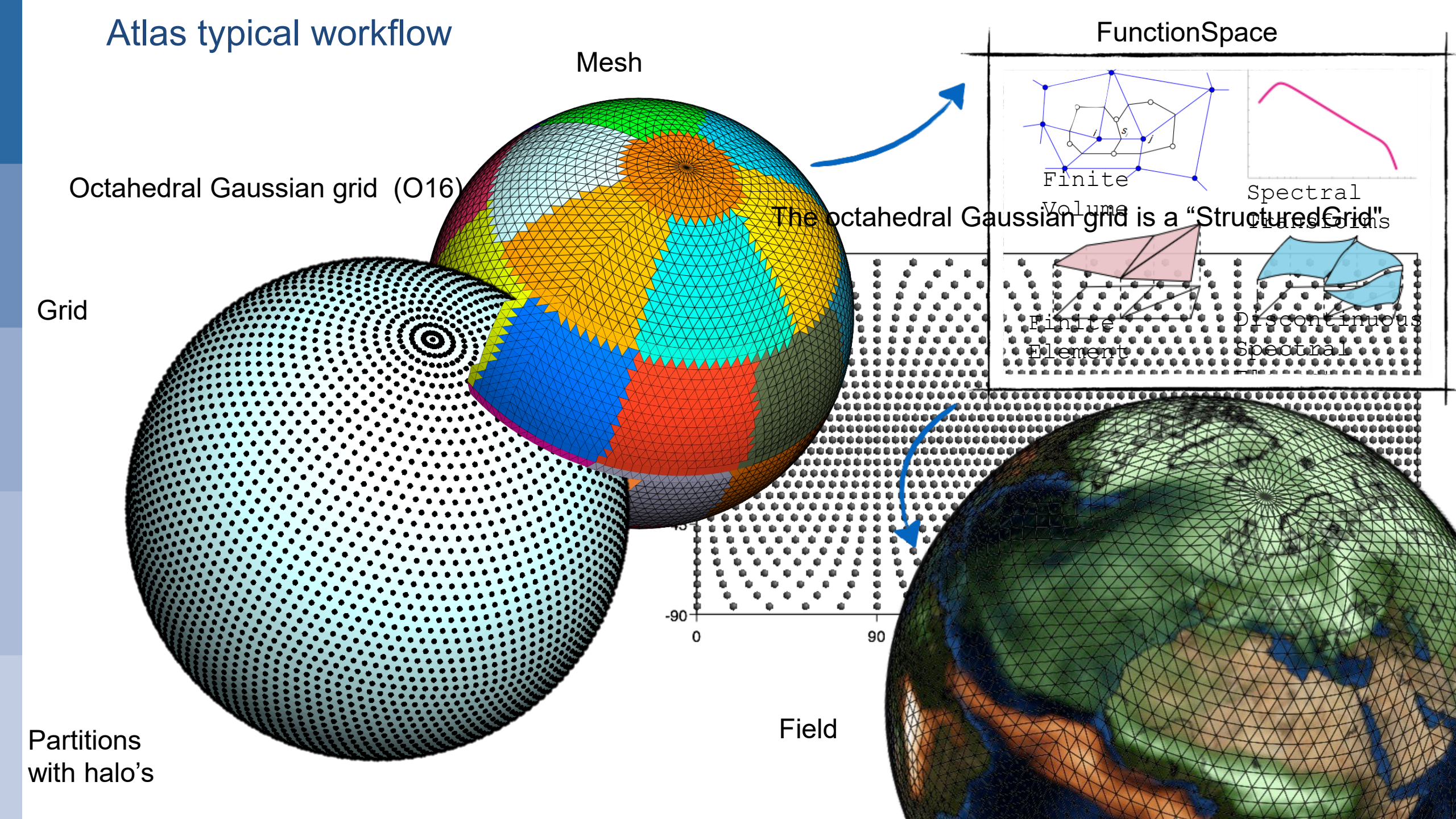
Grid coordinates (x,y)

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

Geographic coordinates (lon,lat)

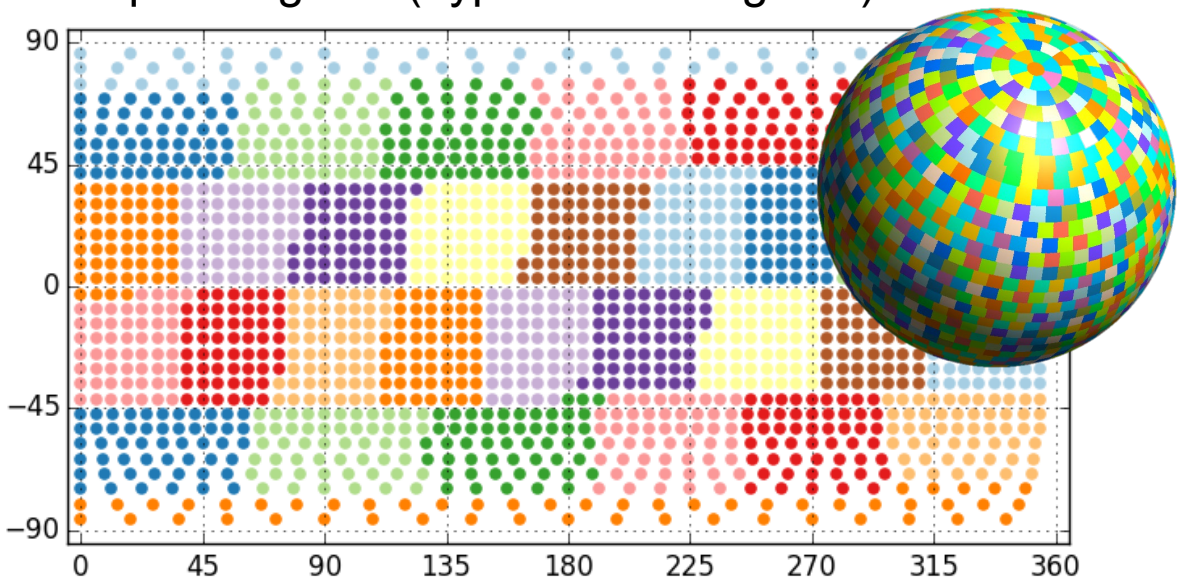
```
for( PointLonLat p : grid.lonlat() ) {
    double lon = p.lon();
    double lat = p.lat();
}
```

Atlas typical workflow

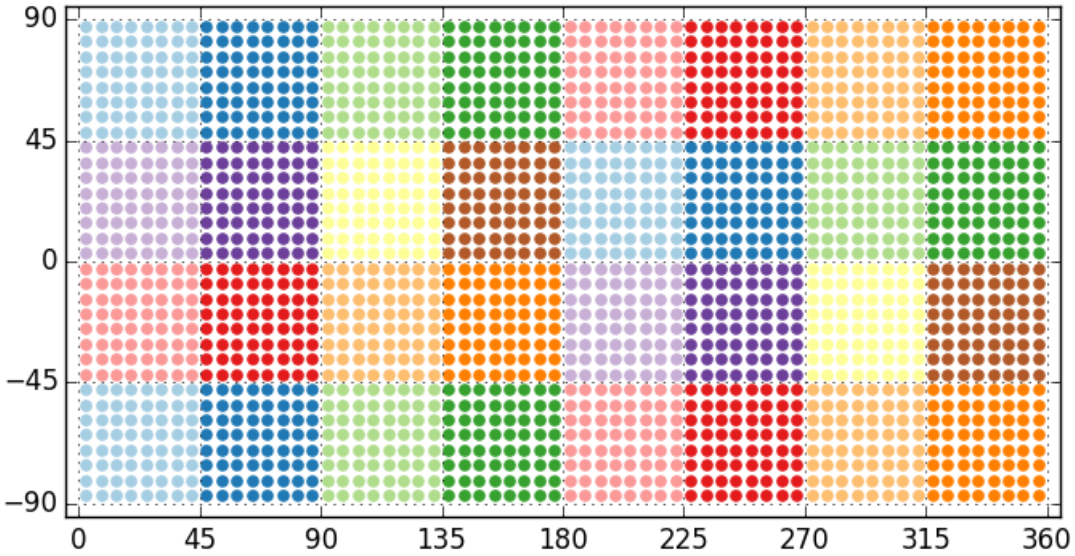


Multiple domain decomposition strategies

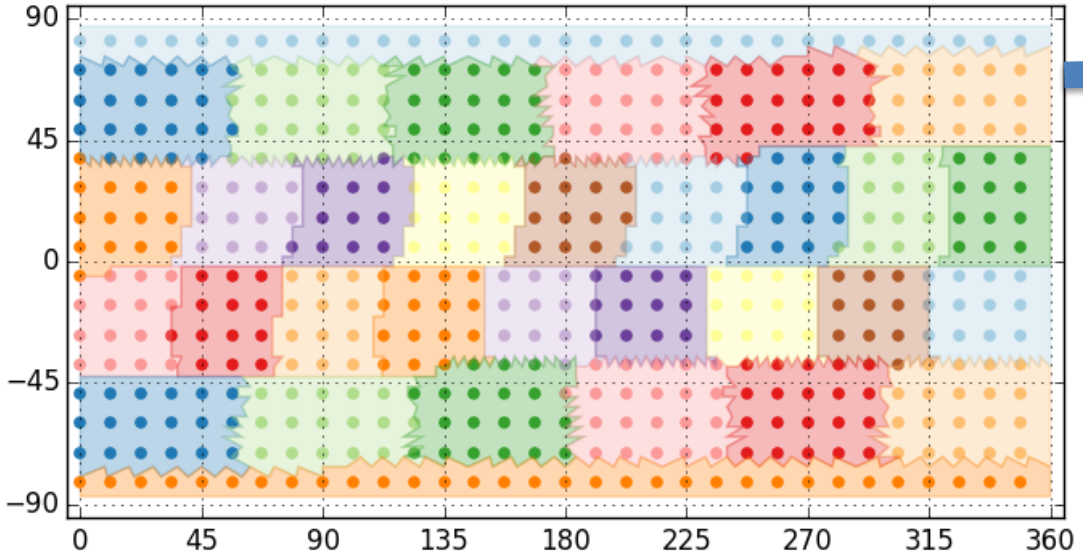
Equal Regions (typical for IFS grids)



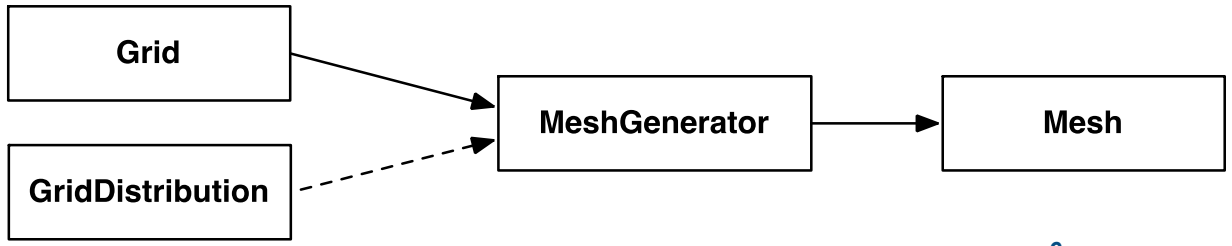
Checkerboard (typical for regional grids)



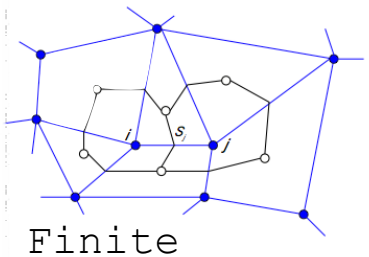
Matching



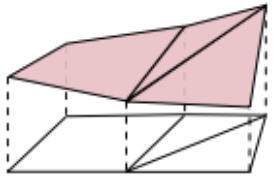
Any grid can follow the domain decomposition of an already distributed mesh or function space



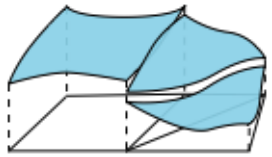
FunctionSpaces: Discretisation specific knowledge



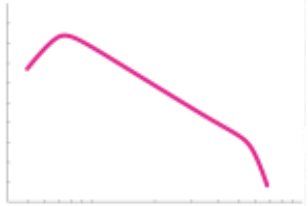
Finite Volume



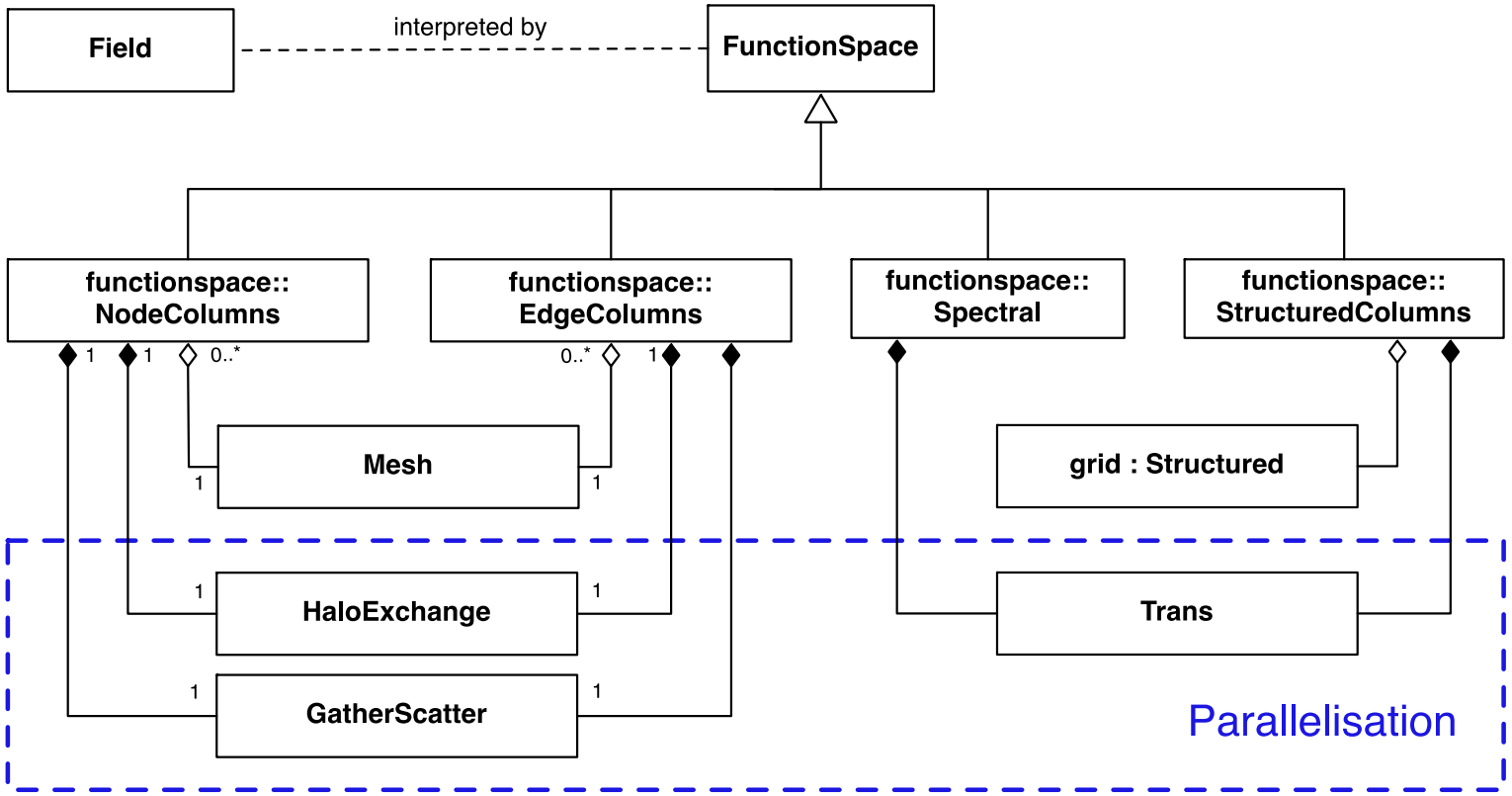
Finite Element



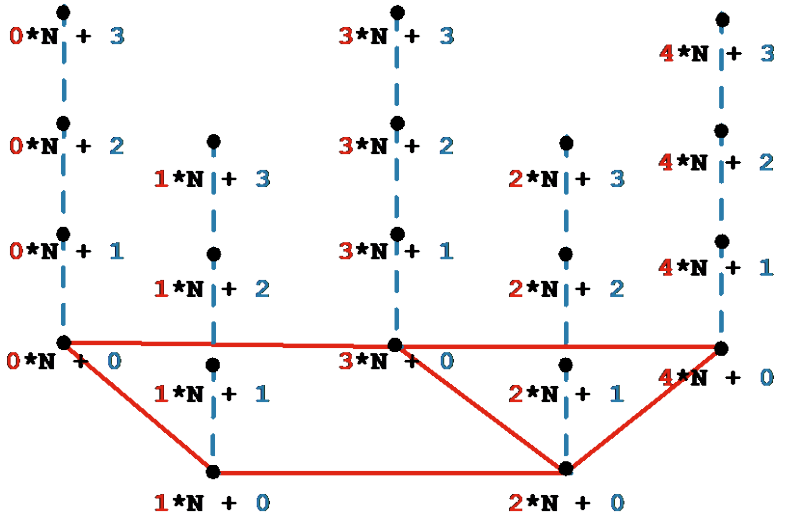
Discontinuous Spectral Element



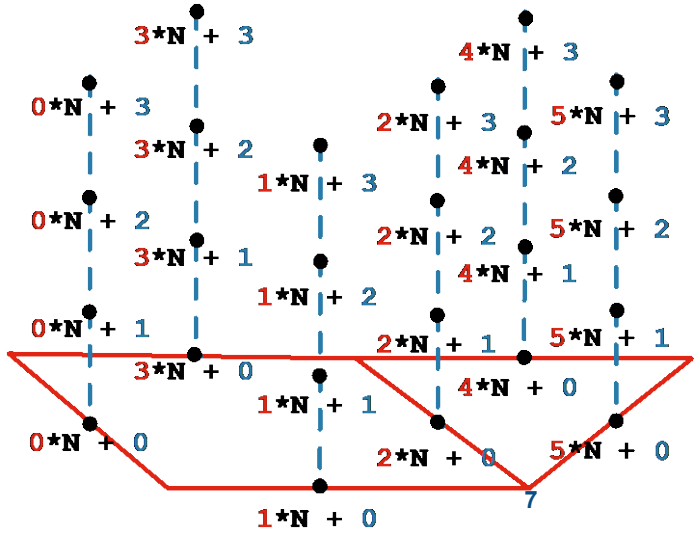
Spectral Transforms



NodeColumns



EdgeColumns

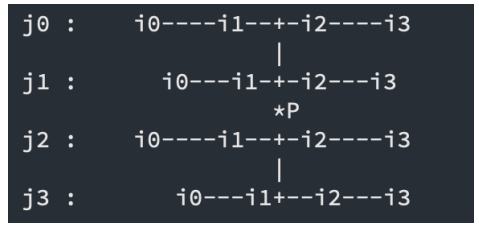


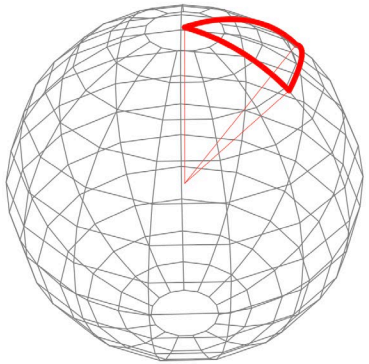
Interpolation

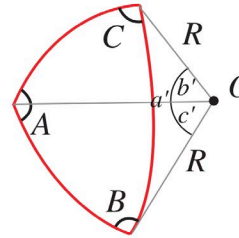
Interpolators can be setup with two functionspaces: for source and target:

```
Interpolation interpolation( configuration, source_functionspace, target_functionspace);
```

Atlas provides a number of interpolation methods







“structured”: bilinear, quasicubic, cubic

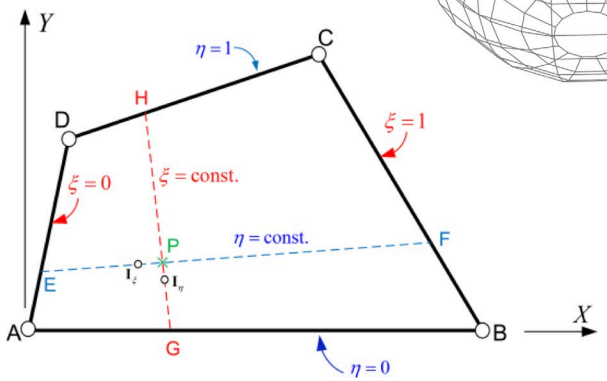
- Meshless, for structured grids (functionspace=StructuredColumns)
- Matches IFS semi-lagrangian interpolation schemes
- Computed in lonlat coordinates.

“finite-element”:

- Requires mesh (functionspace=NodeColumns)
- Find point in spherical elements, and use linear interpolation
- Computed in 3D coordinates

“unstructured-bilinear-lonlat”:

- Follows “finite-element” approach but adapted to match NEMO’s bilinear interpolation method
- Computed in lonlat coordiantes



Interpolation

"(k-) nearest neighbour":

- Meshless, kD-tree based. Functionspace=any
- Closest "k" points are collected
- Distance based weighting

"grid-box-average":

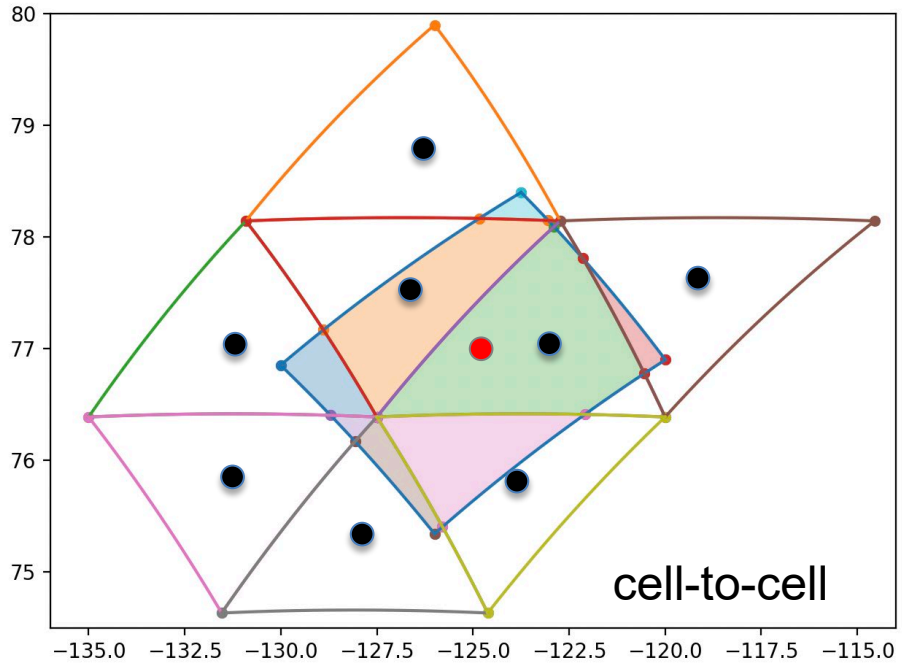
- Conservative interpolation
- Source AND target must be a StructuredGrid
- Grid-boxes around source and target grid points are created and intersecting areas are used to compute weights.

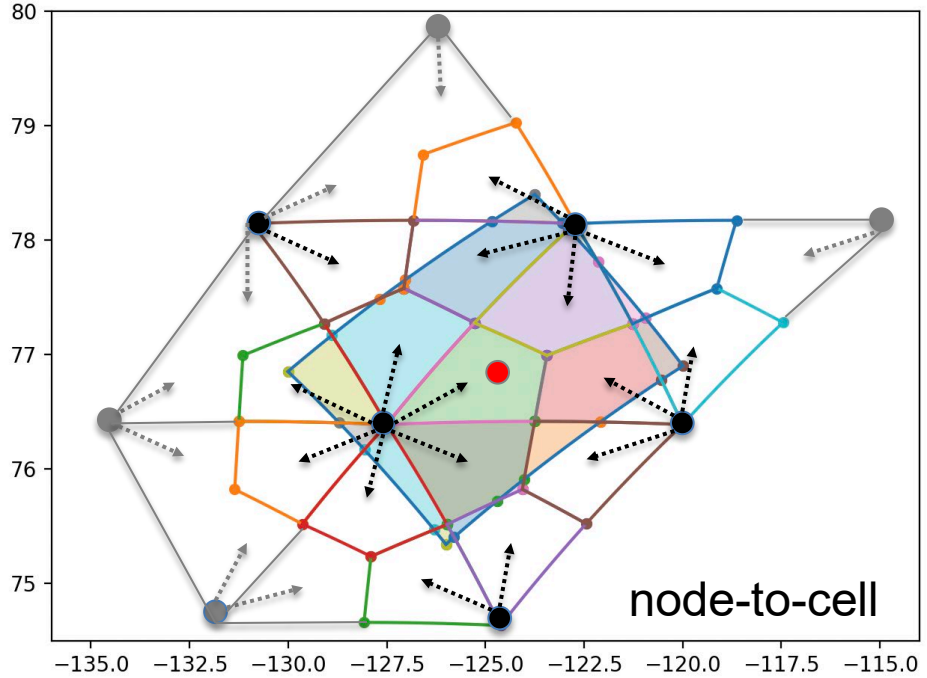
"conservative-spherical-polygon"

- Can be used for unstructured grids
- Conservative with option for 1st and 2nd order accuracy, based on spherical polygon intersections
- Source and target must be mesh-based with convex spherical polygon elements;
- Source and Target values may be either defined as a element-quantity or a node quantity.

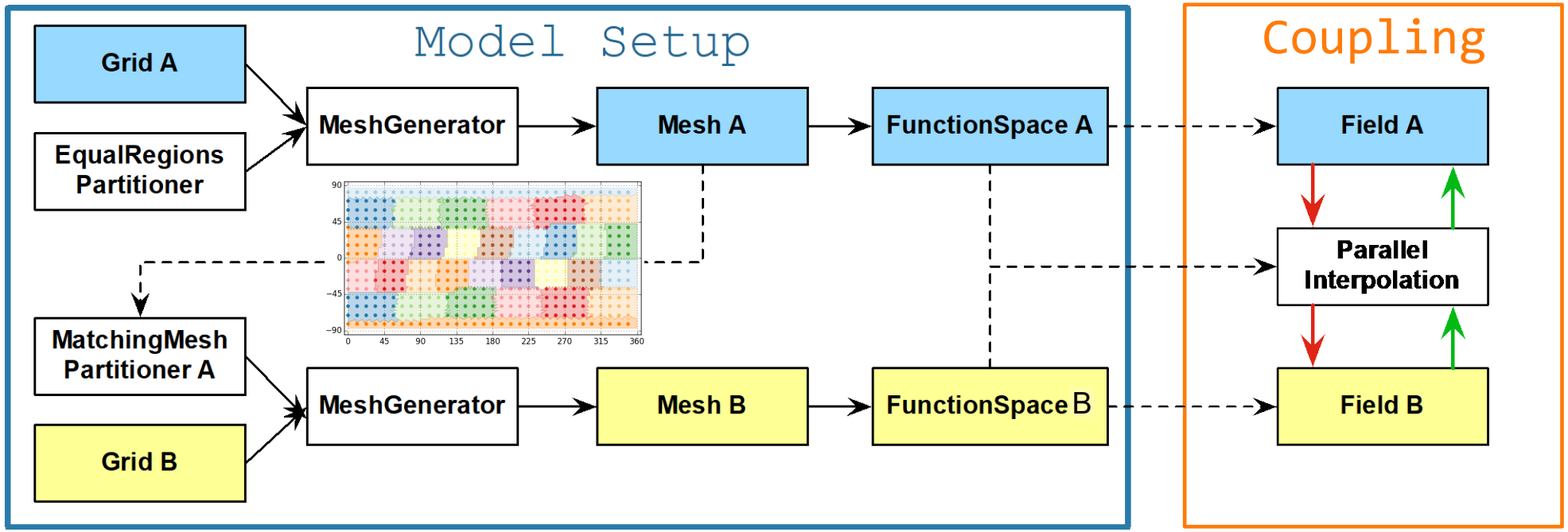
Stencil for octahedral to healpix grid.

Shaded are the intersections that form the target cell





Interpolation 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)
```

Caveats:

- Grid A must contain Grid B
- Mesh A must have sufficiently large halos for interpolation stencil computation

Interpolation with caching

In many instances computing the interpolation stencils and weights is costly, especially for high-resolution grids and limited parallelisation. This cost can be reduced when optionally using caching.

1. Write cache
   - Create interpolator, which contains a partitioned interpolation matrix on each MPI rank
   - Exctract and gather global interpolation matrix to 1 MPI rank (mpi_root)
   - Write global interpolation matrix with 1 MPI rank (mpi_root)

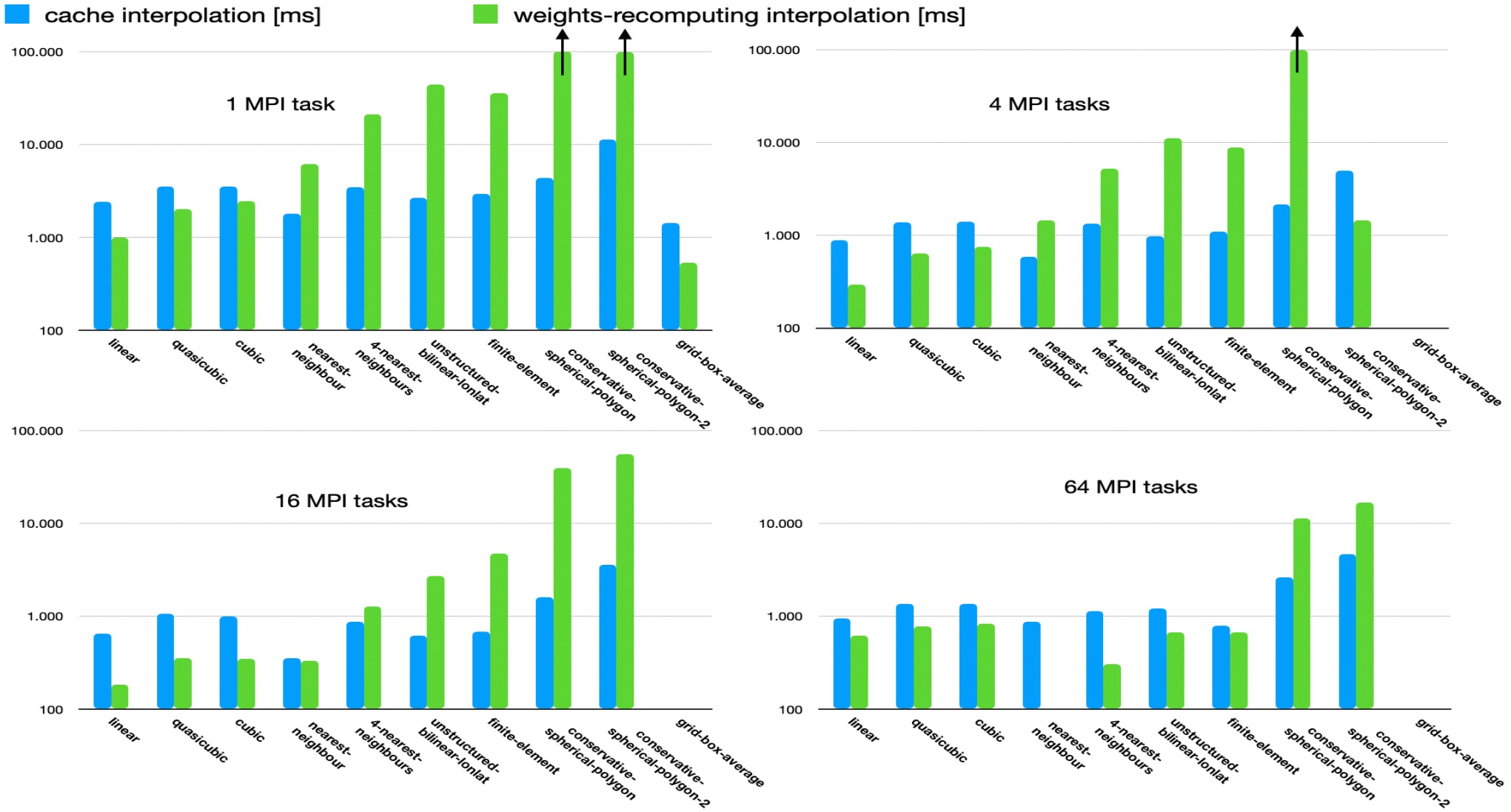
```
Interpolation interpolator(config, source, target);
InterpolationMatrix global_matrix = assemble_interpolation_matrix(interpolation, mpi_root);
write_interpolation_matrix("interpolation_matrix.dat", global_matrix, mpi_root);
```

2. Create interpolator using cache
   - Read global interoplation matrix with 1 MPI rank (mpi_root)
   - Distribute global interpolation matrix to partitioned interpolation matrices on each MPI rank
   - Create interpolator but use already provided matrix

```
InterpolationMatrix global_matrix = read_interpolation_matrix("interpolation_matrix.dat", mpi_root);
InterpolationMatrix matrix = distribute_interpolation_matrix(global_matrix, source, target);
Interpolation interpolator (config, source, target, matrix);
```

Interpolation with caching

Various timings and speed-ups can be reported. For some methods, like structured interpolation methods, weight and stencil computations are very fast, and caching does not increase performance. Reading and distributing matrix also costs!

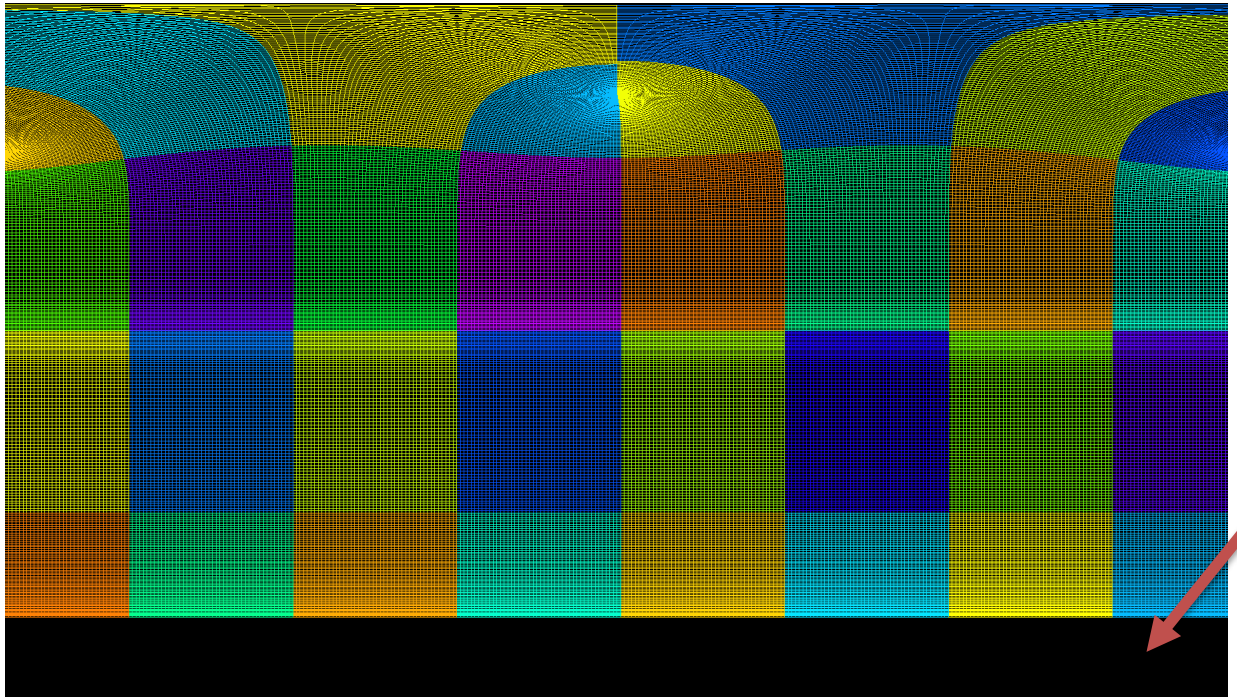


Benefits using Matching partitioner

1. Interpolation weights are computed in parallel, staying within each MPI rank
2. Source fields may only require a halo-exchange → minimum parallel communication

Drawbacks

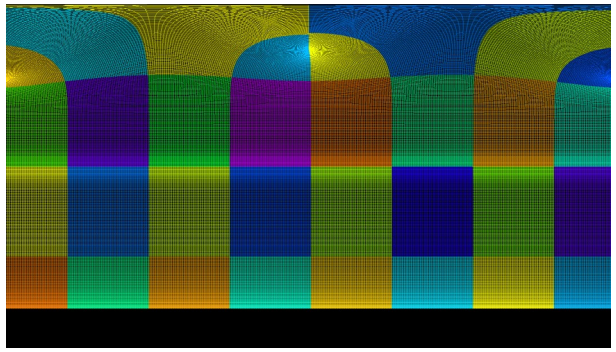
1. Possibly very large and sometimes unknown source halo requirements (e.g. conservative interpolation)
2. In general model components use their own domain decomposition matching the numerics and grid best
3. ORCA grid does not span entire globe.
   ORCA to Gaussian gid → ERROR: Several Gaussian grid points cannot be assigned a matching partition

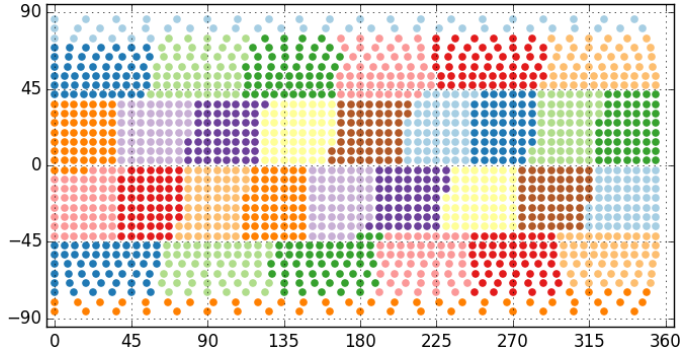


Points on target grid that fall here cannot be assigned any partition

Non-matching domain decomposition

1. Source and Target may have independent domain decomposition



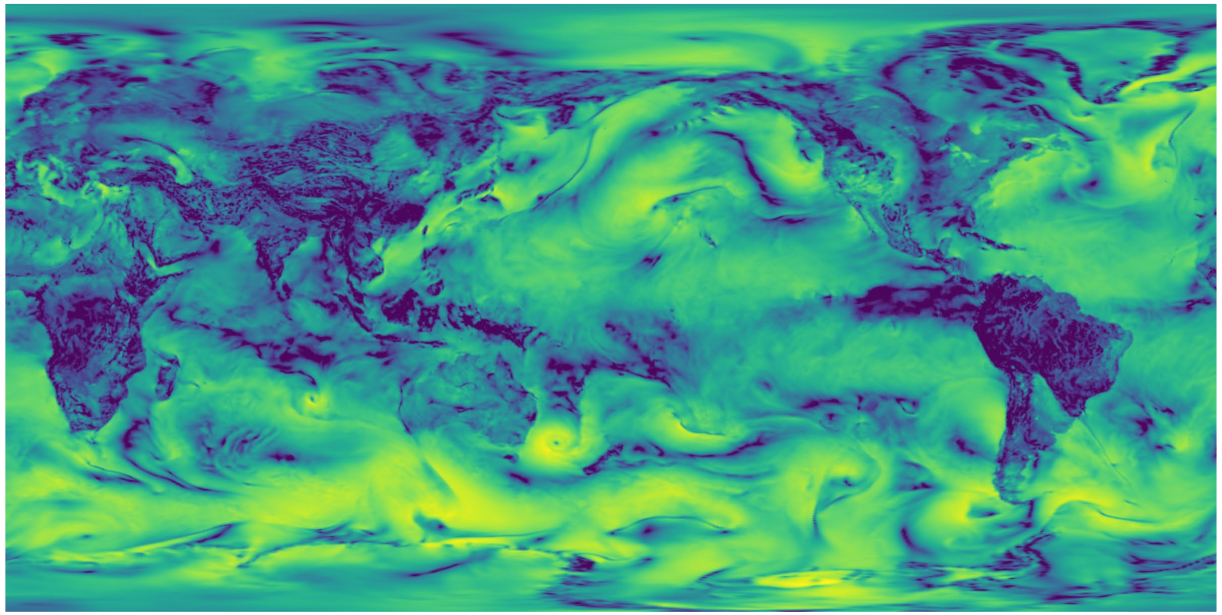


2. No halos are required.
3. Reuse same cached global interpolation matrix described earlier!
4. Distribute the global interpolation matrix to target domain decomposition.
5. Collect information on each target partition on which MPI ranks the source stencils are located, and create a “Collect” communication pattern, which collects source field values from all required MPI ranks to satisfy the interpolation stencils
6. Apply an interpolation operator using the weights and collected source field values

Some interpolation algorithms which only implemente computation of interpolation weights in serial may now find use in parallel applications.

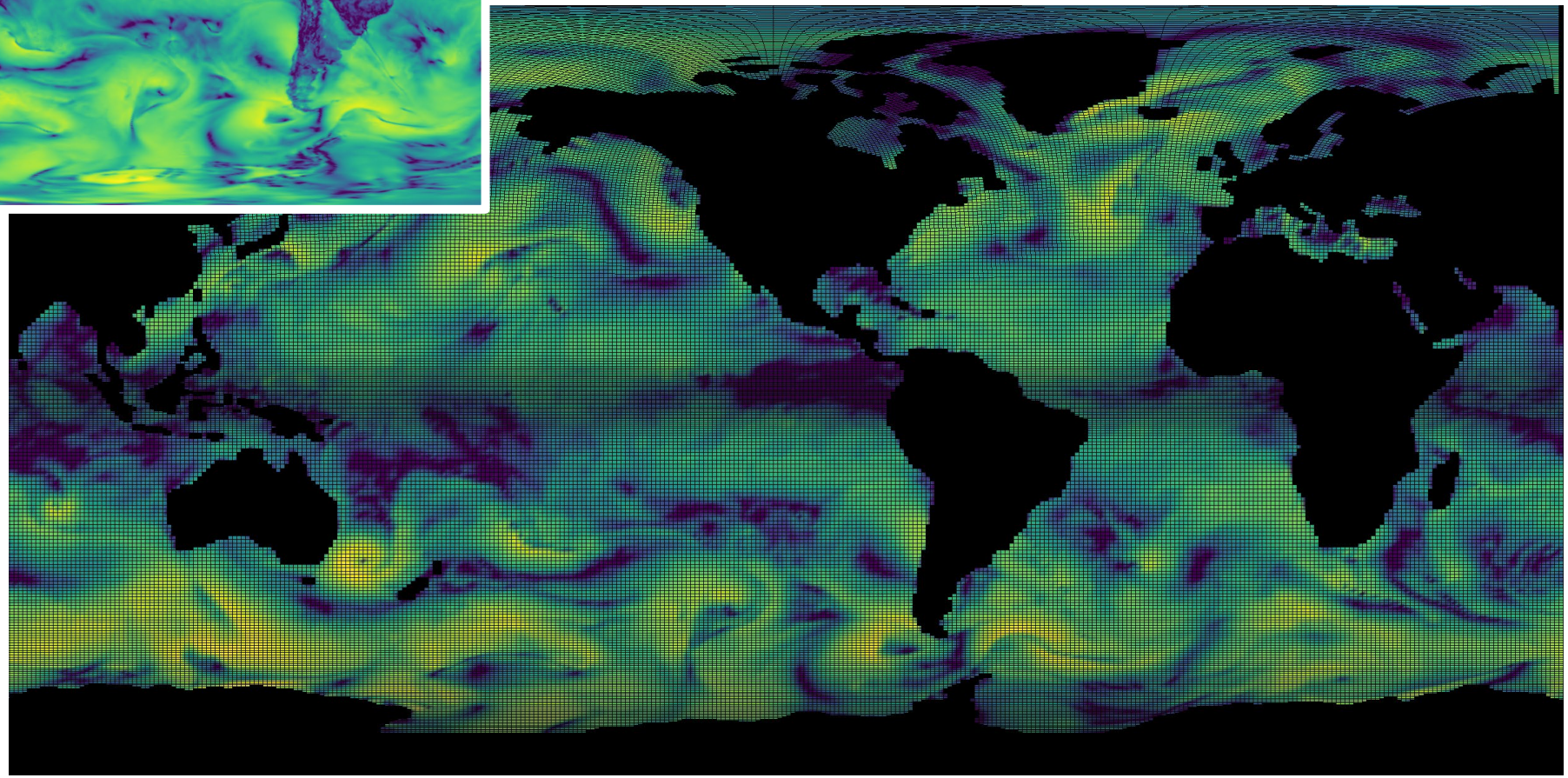
Example 1: From atmosphere to ocean

Field: 10m wind magnitude, reanalysis 2025.04.01 at 00:00



source:
O360 (0.25 deg)

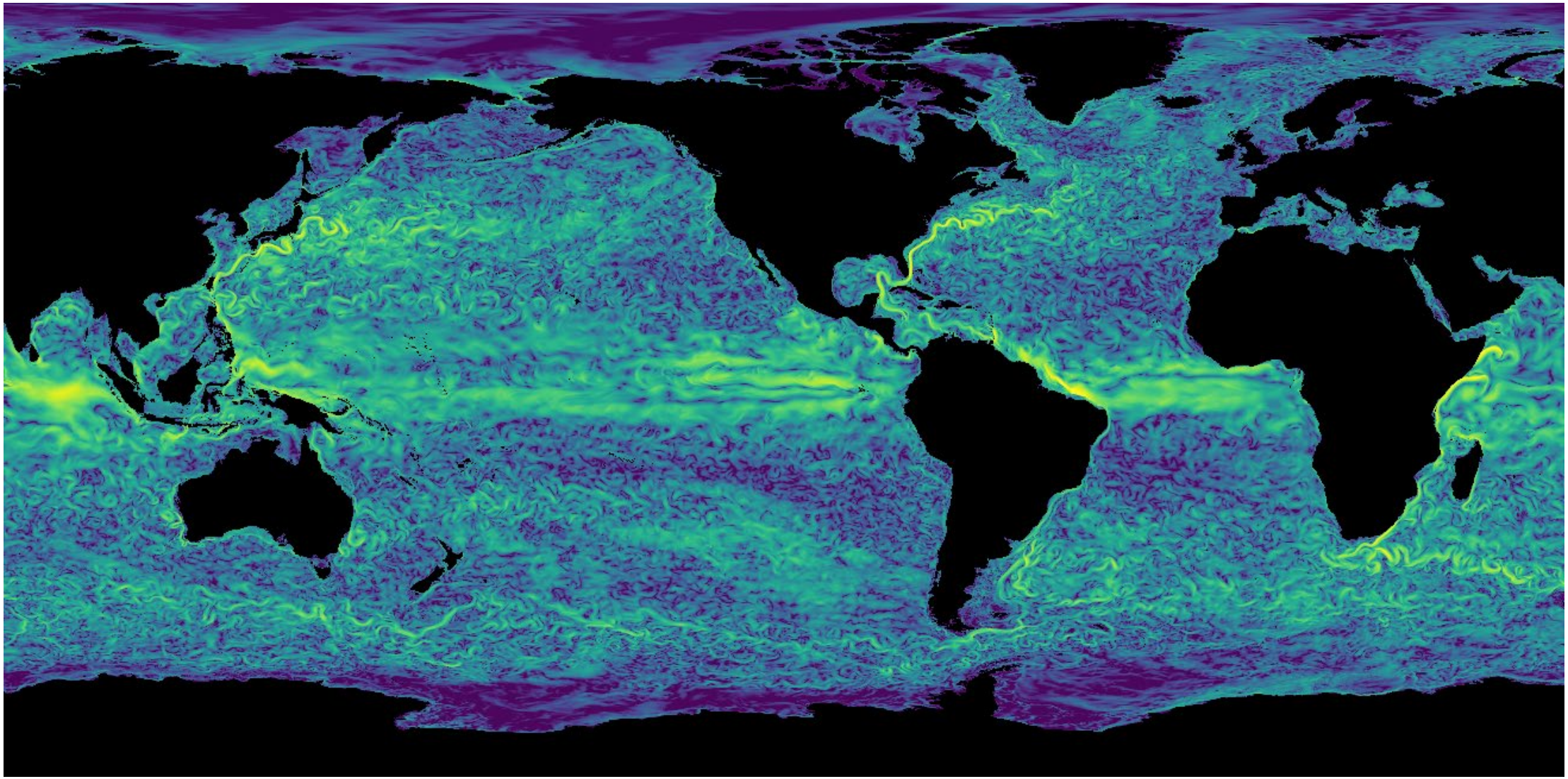
target: ORCA1_F



interpolation:
structured-bicubic

Example 2: From ocean to atmosphere

IFS CY49R2
TcO1279 atmosphere + eORCA12 ocean

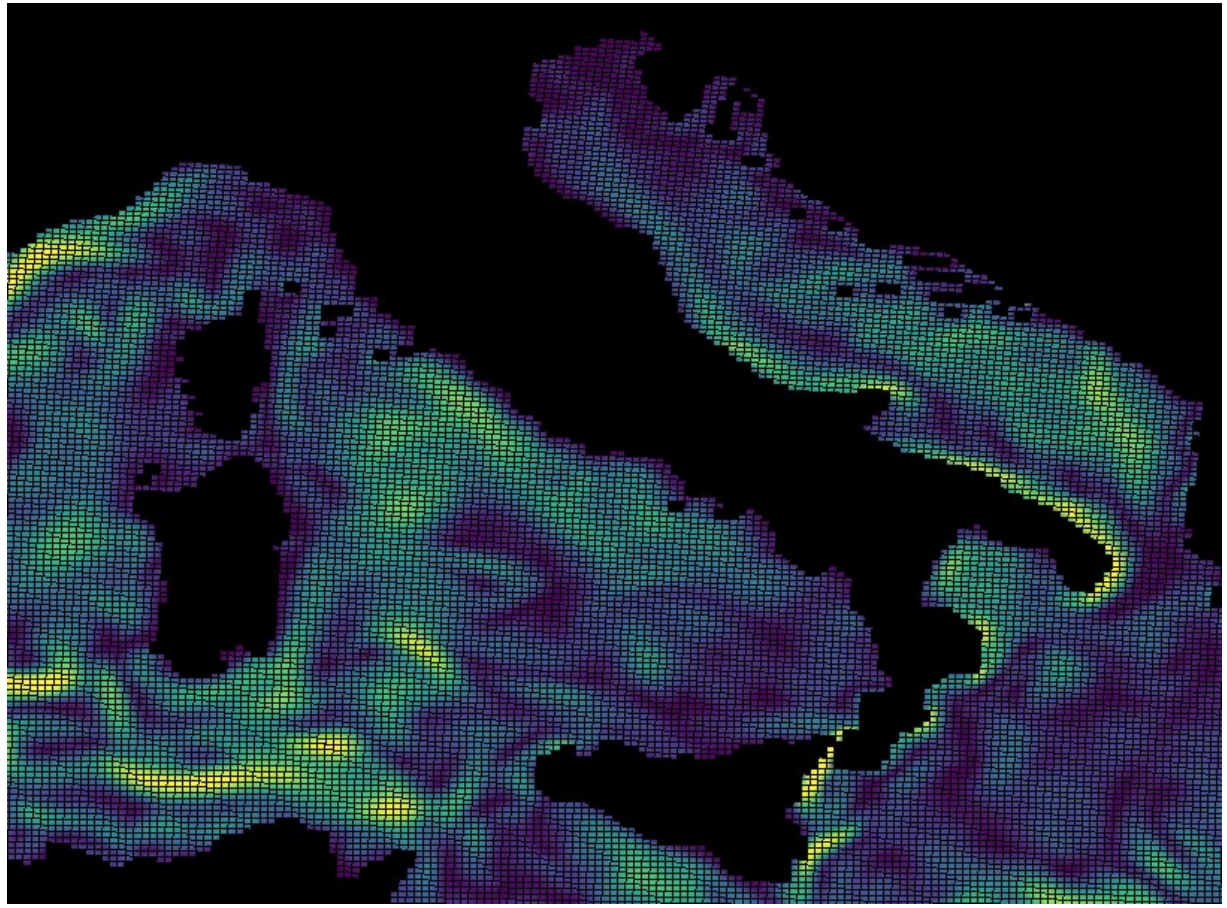


source: eORCA12_T (1/12 degree)

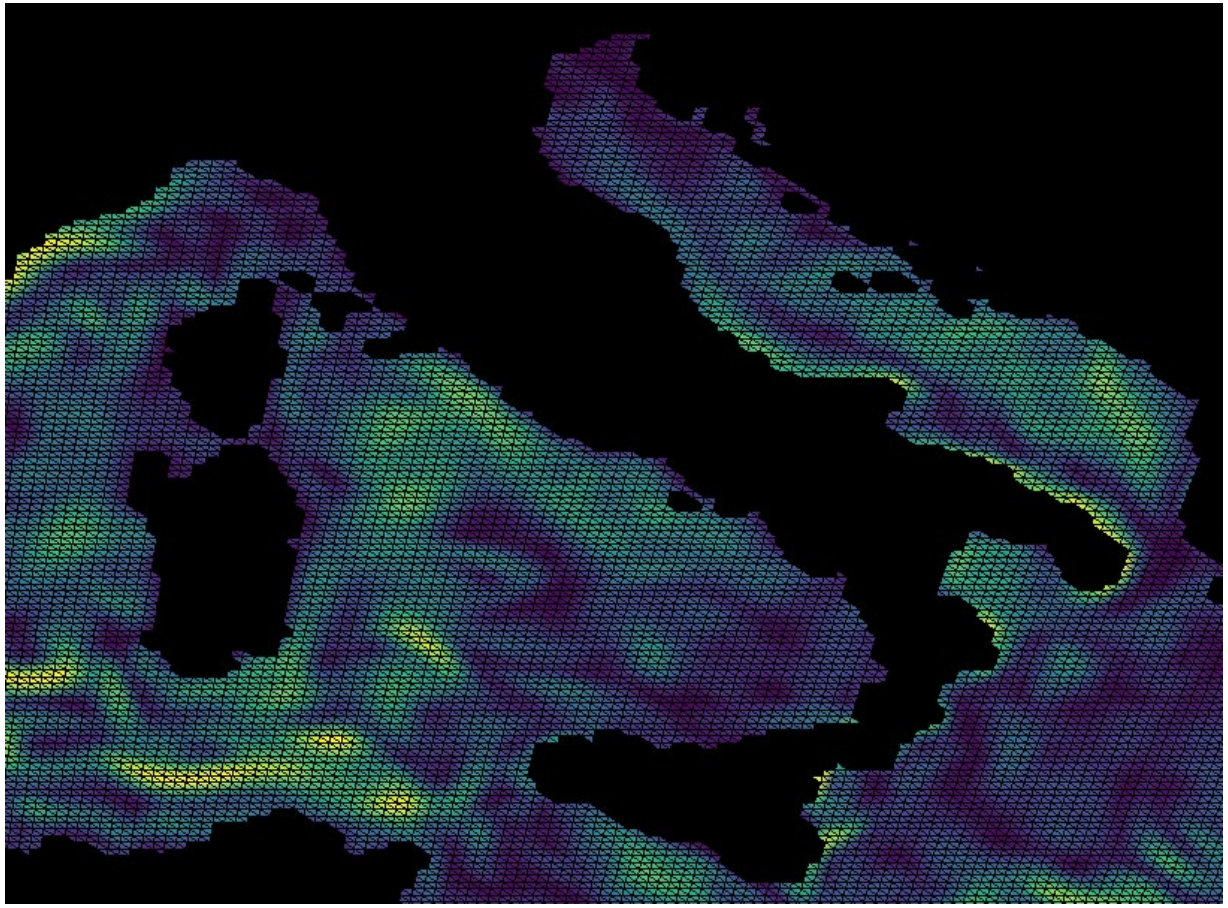
Sea velocity magnitude

Example 2: From ocean to atmosphere

Ocean: eORCA12 (1/12 degree)



Atmosphere: O1280 (9km)

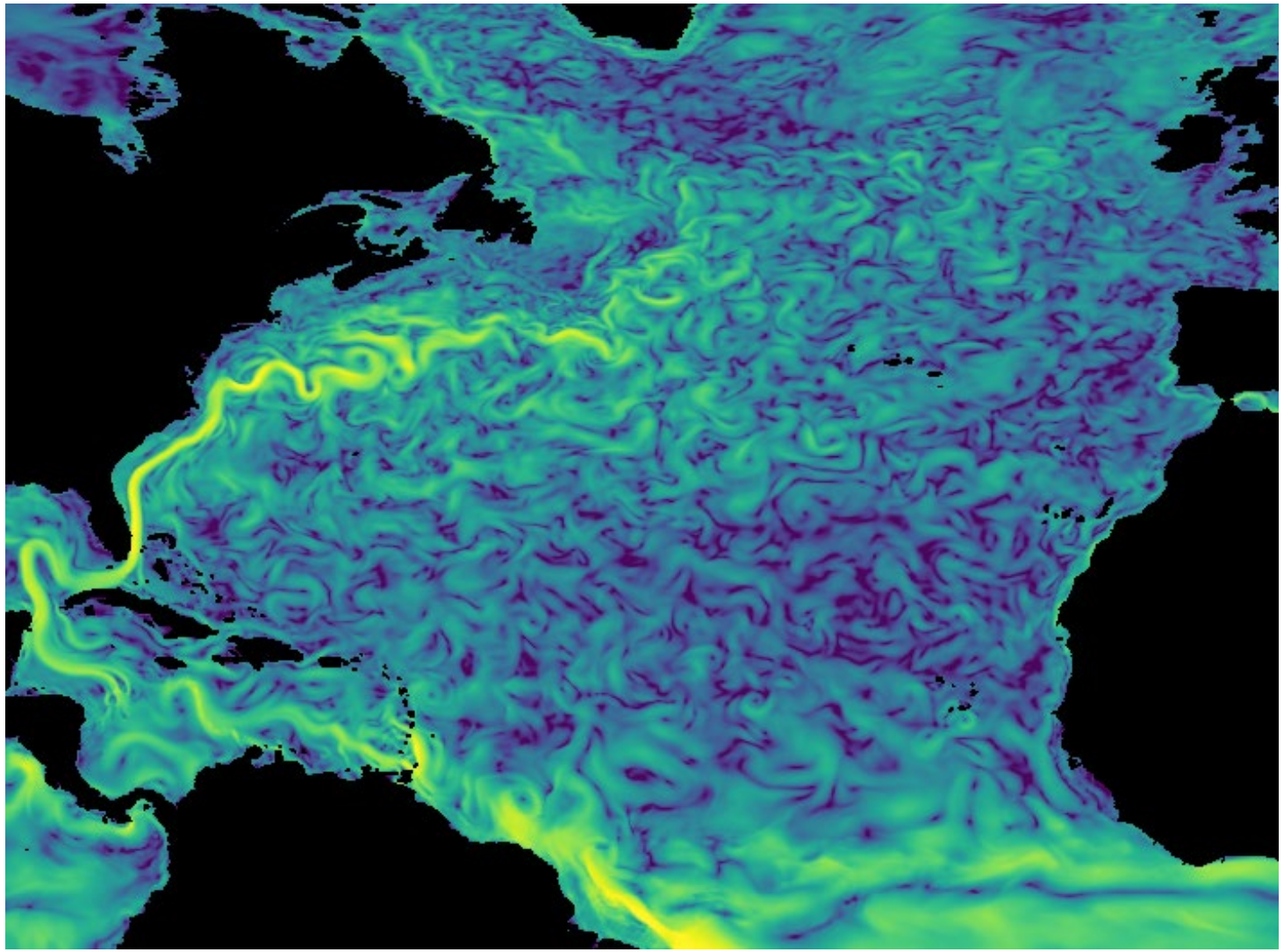


similar in resolution

Example 2: From ocean to atmosphere

Interplation method: conservative-spherical-polygon

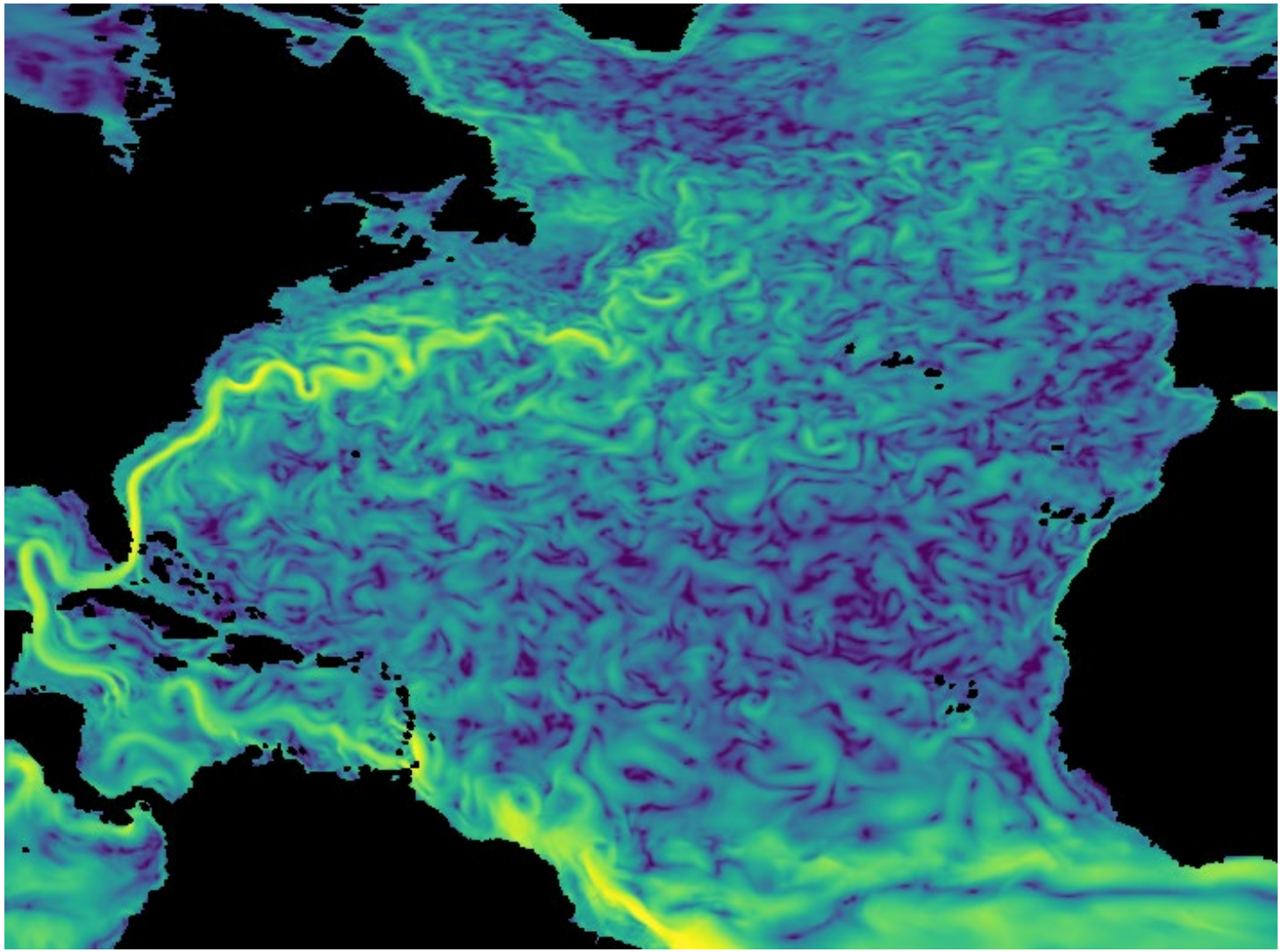
O1280



Example 2: From ocean to atmosphere

Interplation method: conservative-spherical-polygon

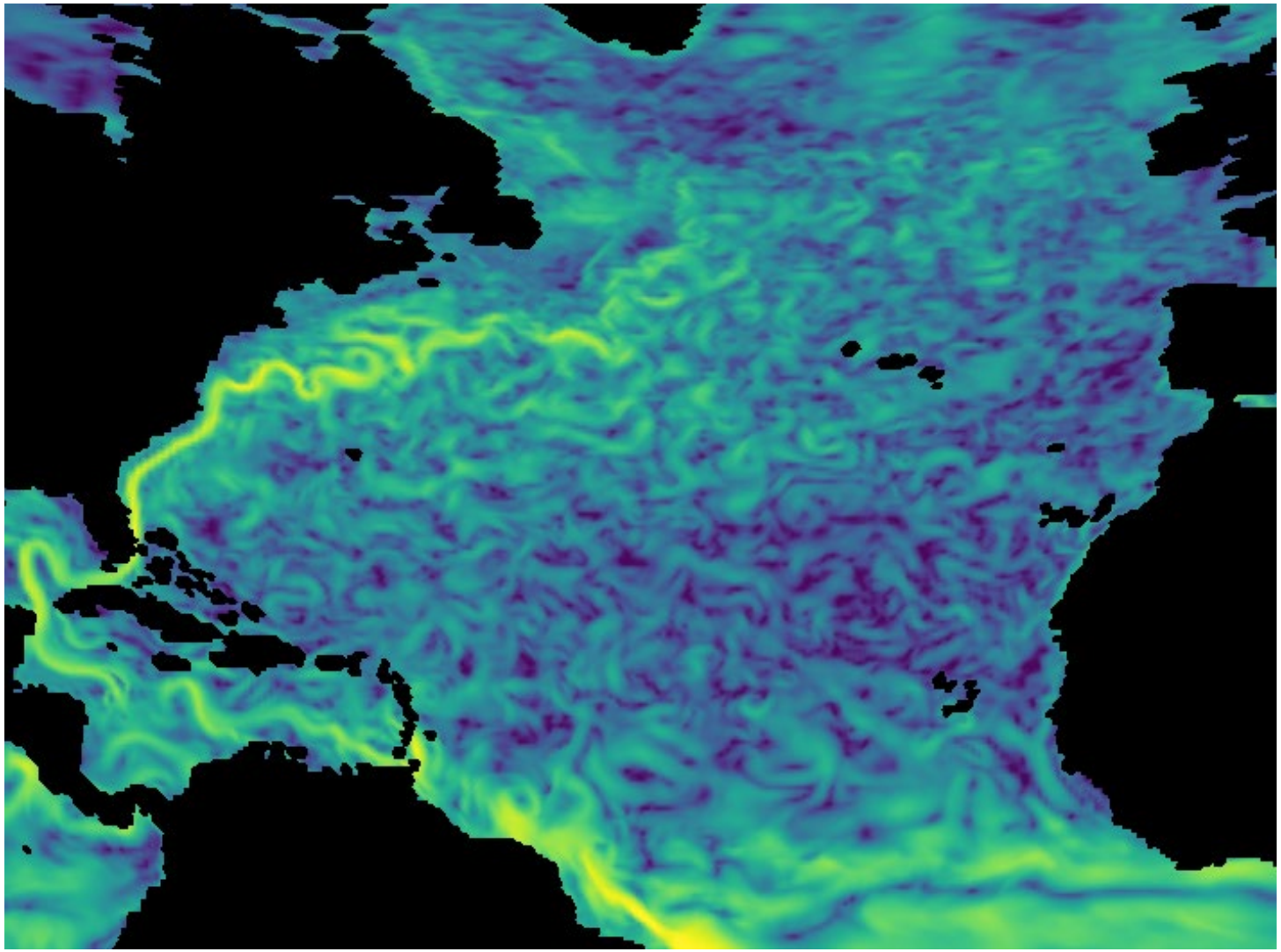
O640



Example 2: From ocean to atmosphere

Interplation method: conservative-spherical-polygon

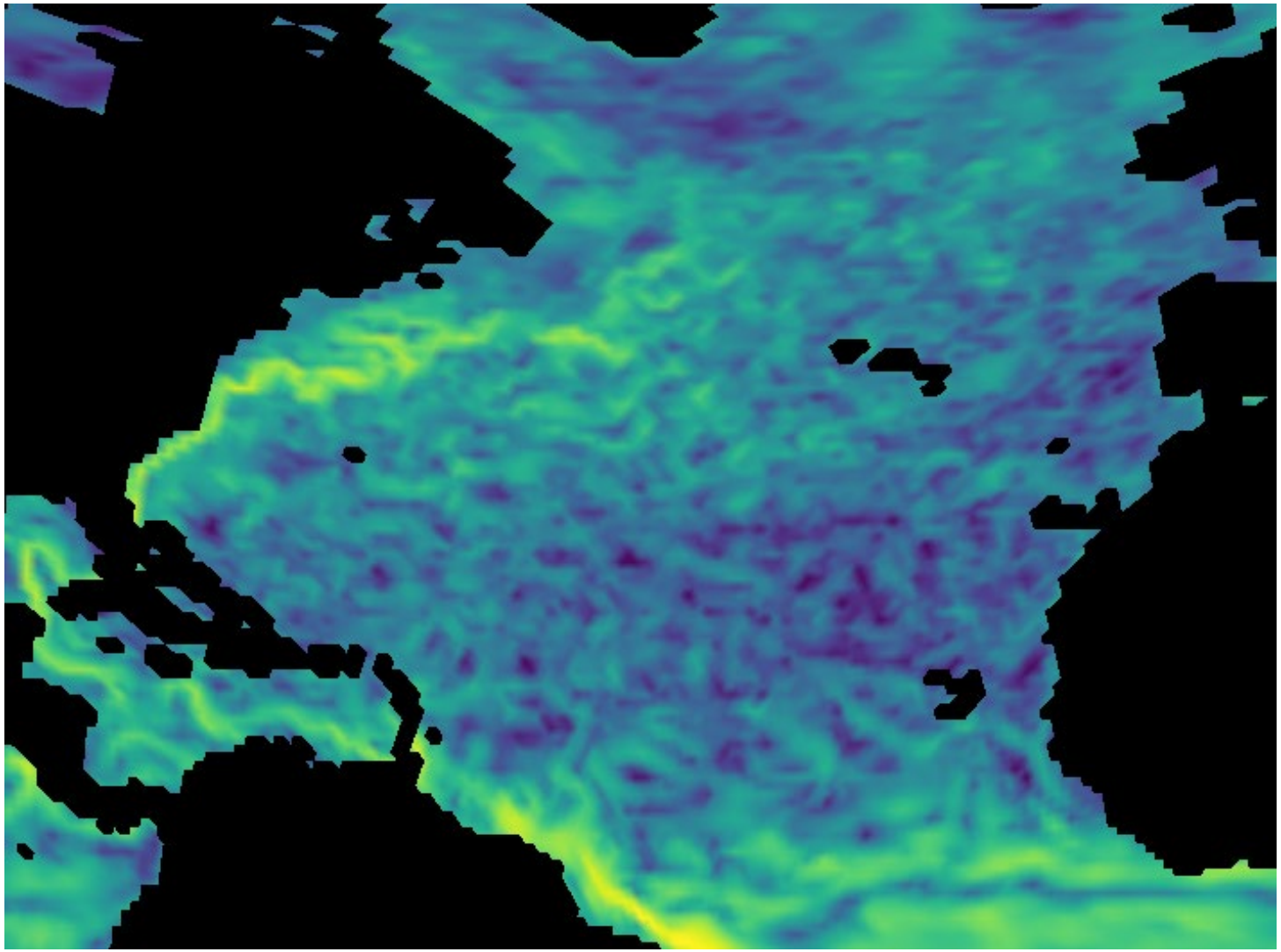
O320



Example 2: From ocean to atmosphere

Interplation method: conservative-spherical-polygon

O160

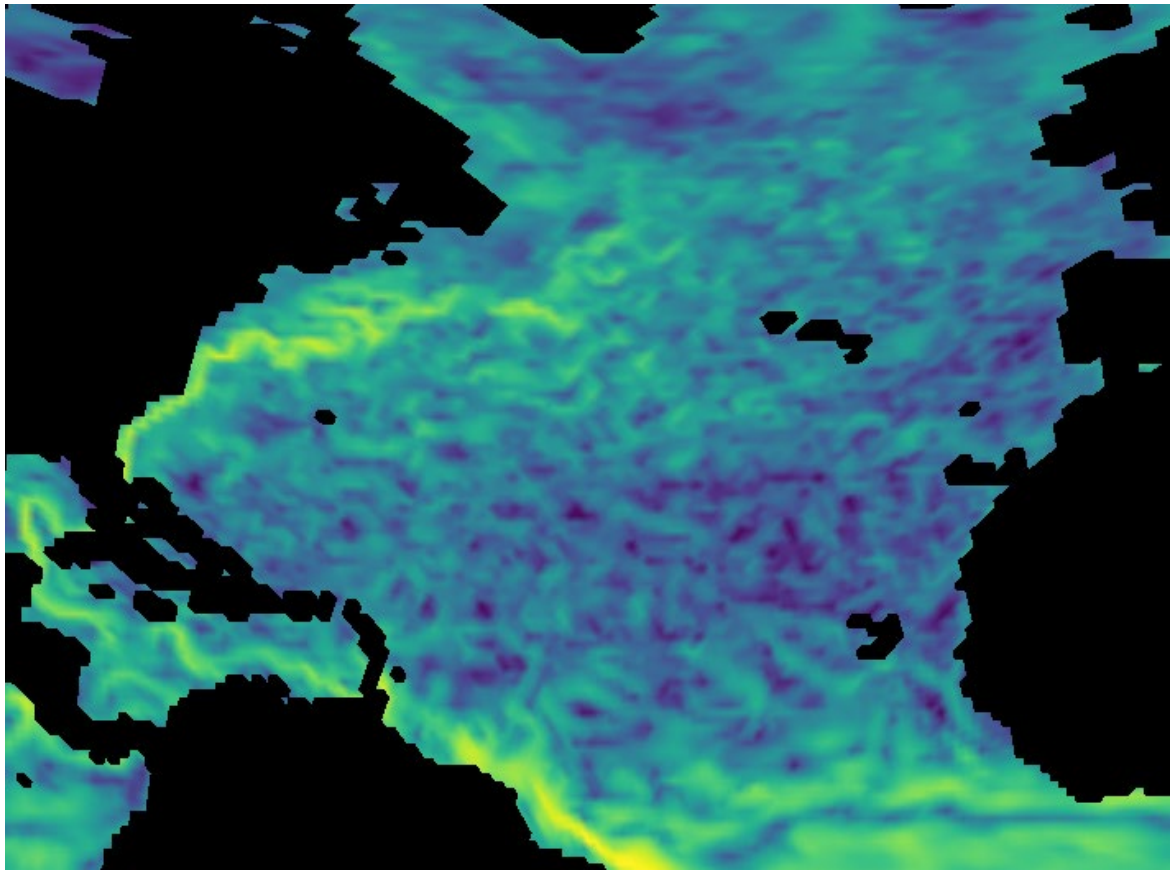


Example 2: From ocean to atmosphere

Field: Sea velocity magnitude, from

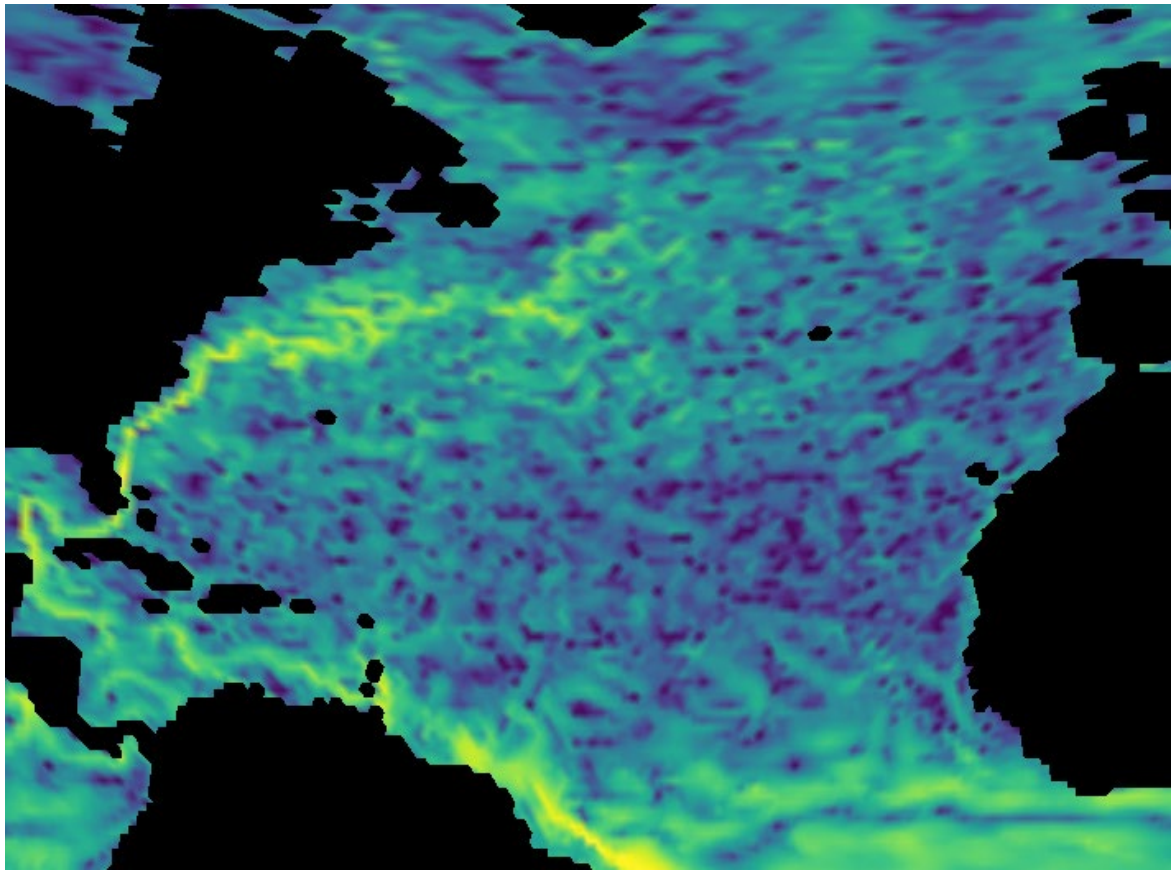
Compare interpolation eORCA12 to O160: conservative vs non-conservative:

conservative-spherical-polygon



- Smooth, filtering the high-frequencies
- Shape preserving

unstructured-bilinear-lonlat



- Preserves maxima / minima
- Noisy, unfitered

Next steps, near term

- Implement use of land-sea mask
- Use alternative fallback interpolation method at coast-lines
  - where target elements are not fully covered by source elements
  - where source fields contain missing values
- Integrate with IFS
- Warmworld project: Use FESOM unstructured ocean mesh
- Use Atlas to couple ecrad (radiation model)

Next steps, longer term

- Implement capability to couple ESM components which use different MPI communicators with different MPI_SIZE.

Coupling Earth System Model Components using the Atlas library

Workshop on surface process coupling and its interactions with the atmosphere

Bonn, Germany | 9-10 April 2025

Willem Deconinck, Slavko Brdar, Pedro Maciel

willem.deconinck@ecmwf.int

ECMWF

Thank you for your attention!

Funded by the European Union
Destination Earth
implemented by ECMWF esa EUMETSAT