Atlas, a library for flexible Earth system modelling

ECMWF Training course:

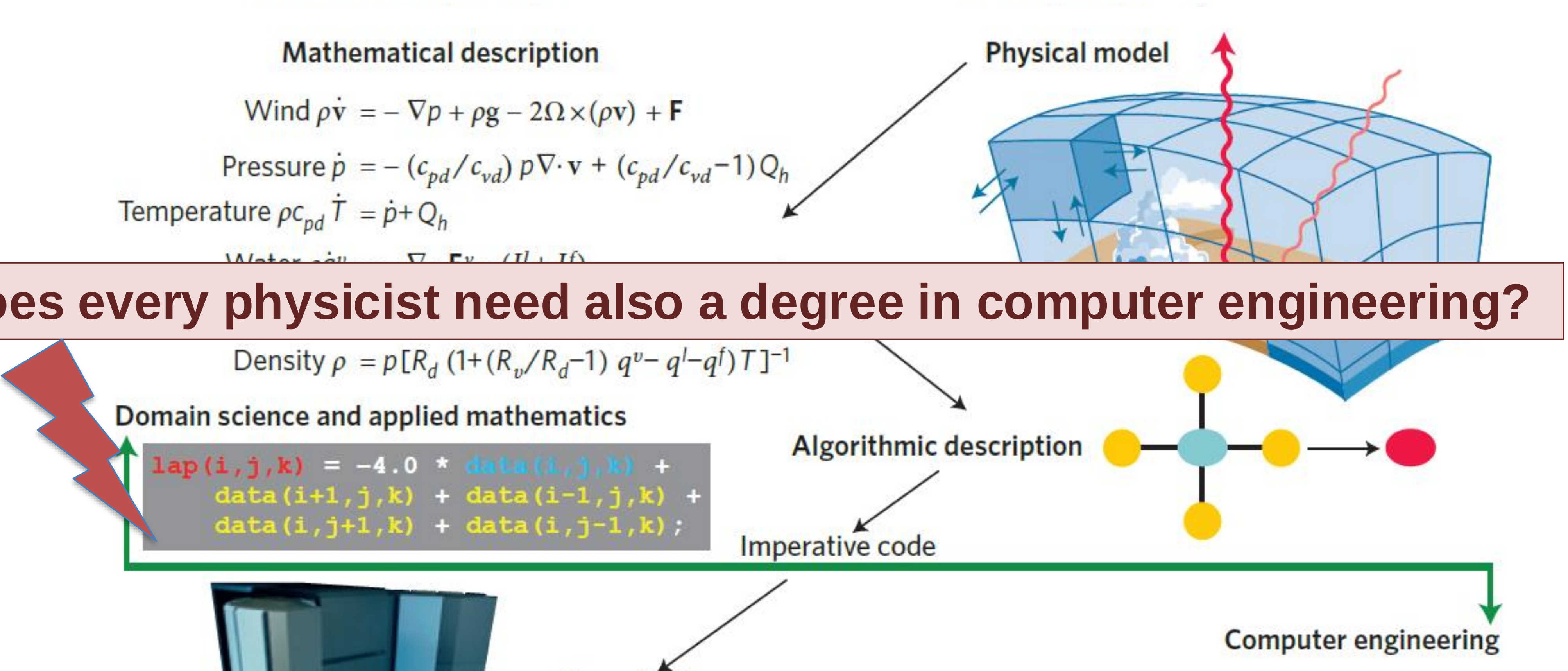
Numerical Methods for Numerical Weather Prediction

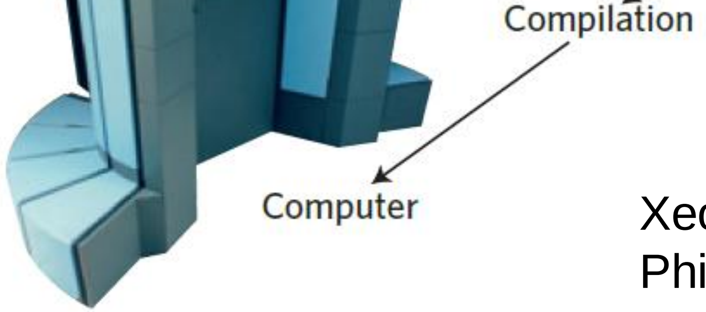
Willem Deconinck
willem.deconinck@ecmwf.int

ECMWF

Traditional science workflow

[Schulthess 2015]
NATURE PHYSICS



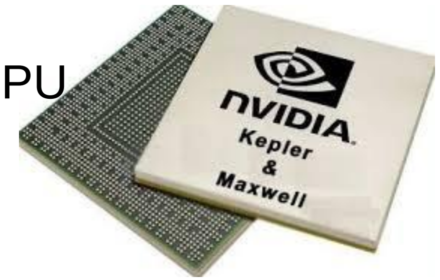


Does every physicist need also a degree in computer engineering?

Compilers can no longer do everything…

Xeon Phi

GPU



FPGA

What are the options?

Separation of concerns !!!

OpenACC
Directives for Accelerators

OpenMP

NVIDIA CUDA

MPI

Huge exercise to port all model components to (each?) hardware

Flexible model development on abstractions

Atlas

Development of toolchain to abstract hardware, memory, numerics

Road blocks

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), www.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
- Support structured and unstructured grids (global as well as **LAM**)

ECMWF

Atlas typical workflow

Mesh

Octahedral Gaussian grid (O16)

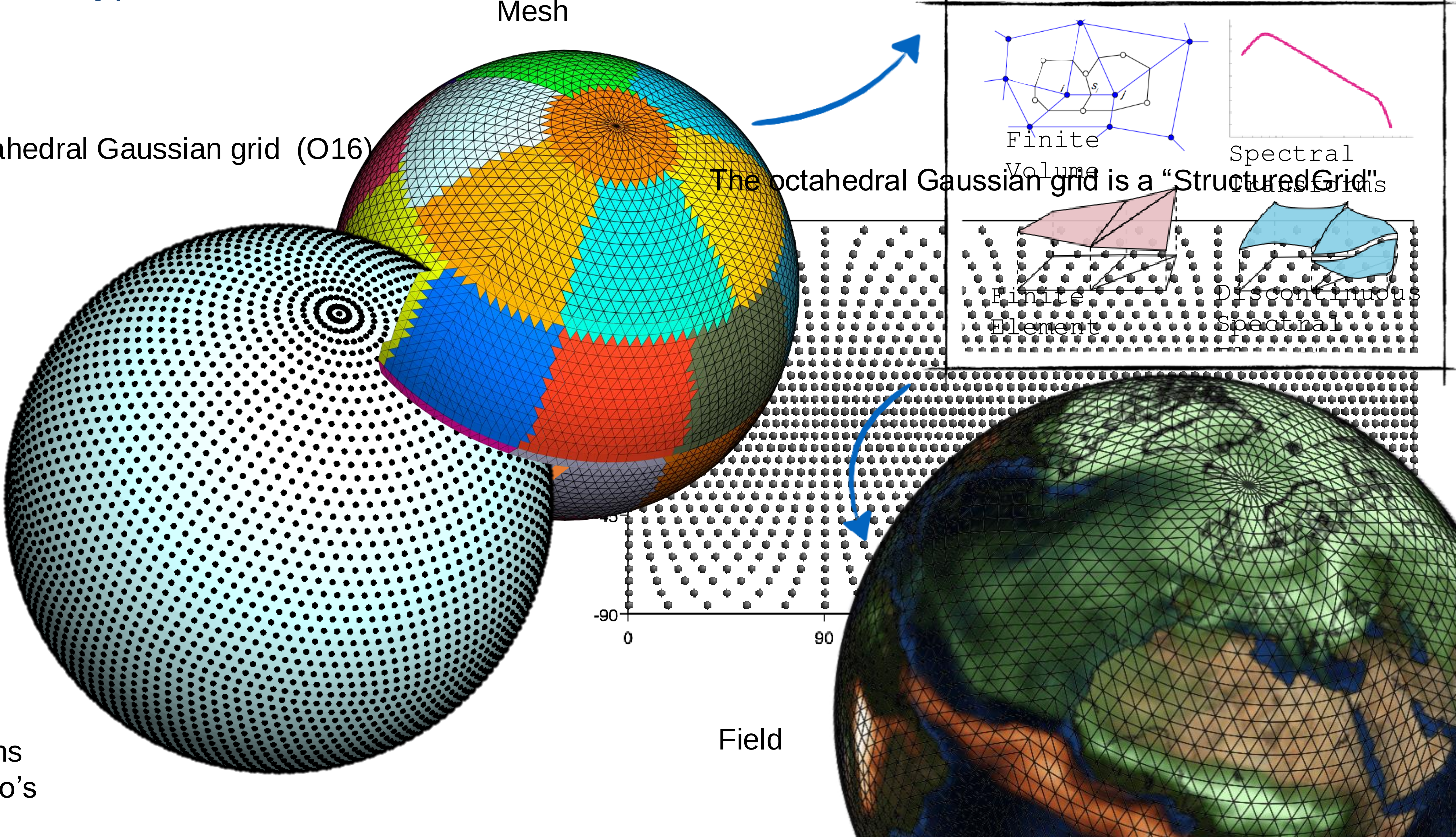
Grid

Partitions
with halo’s

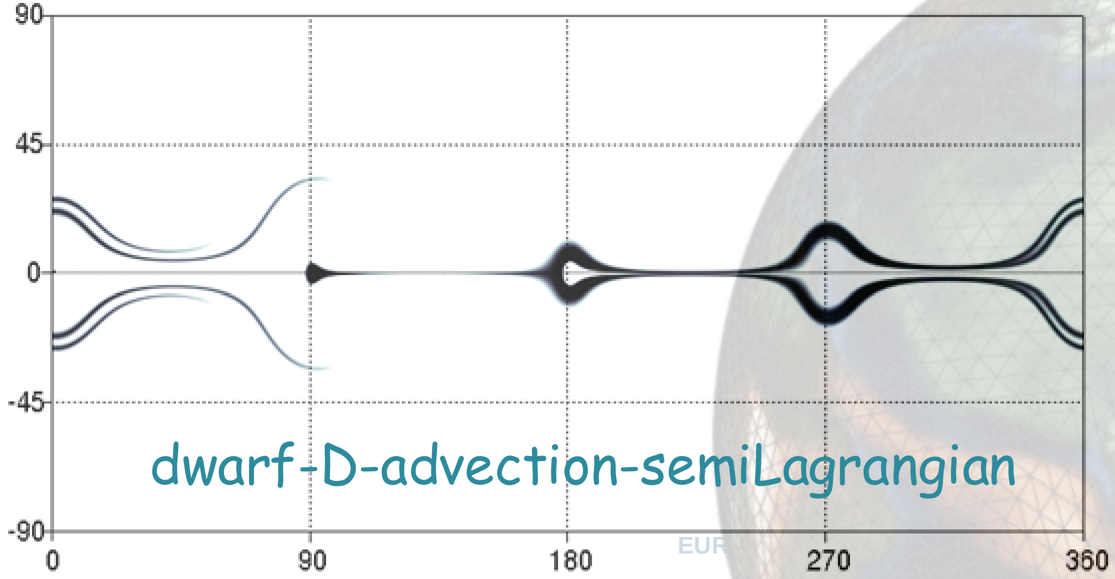
FunctionSpace

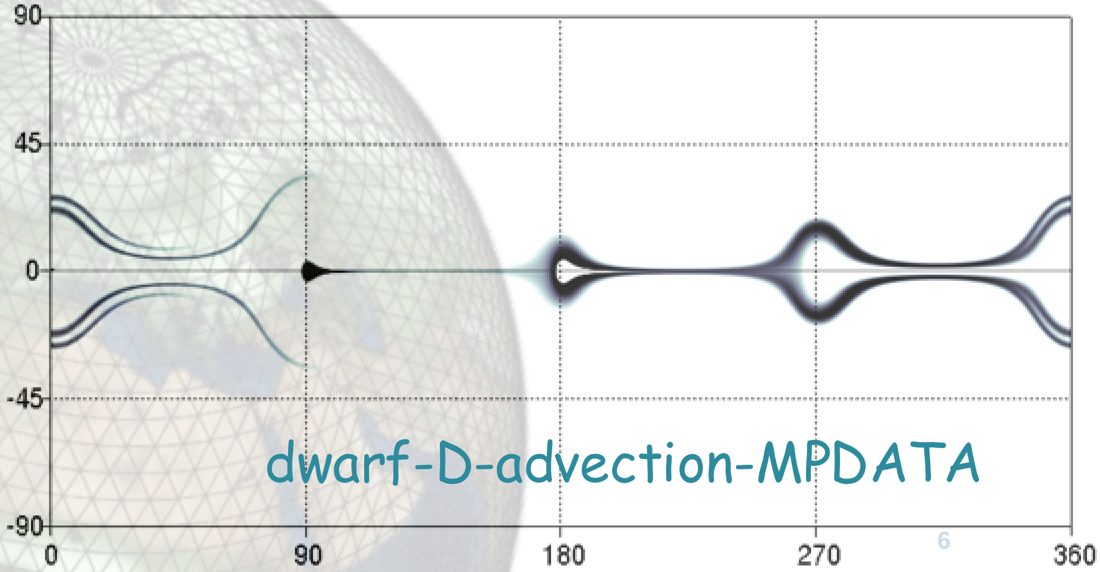
The octahedral Gaussian grid is a “StructuredGrid"

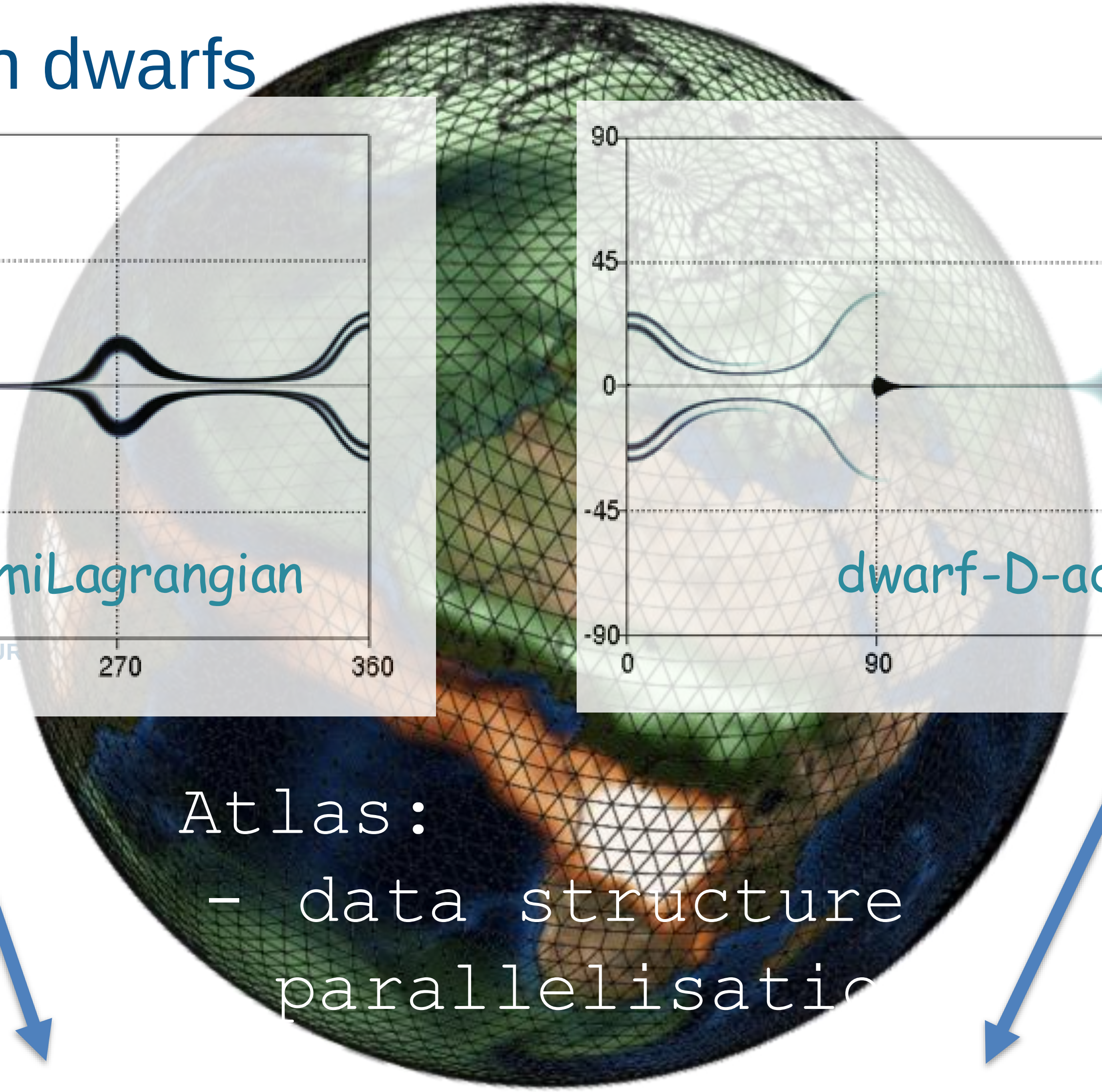
Field

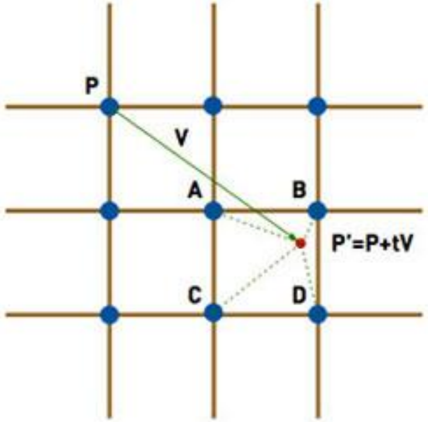


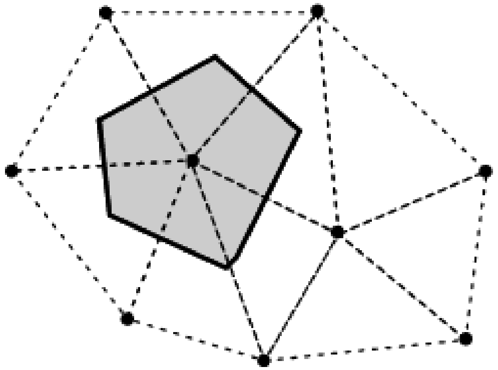
ESCAPE: Advection dwarfs







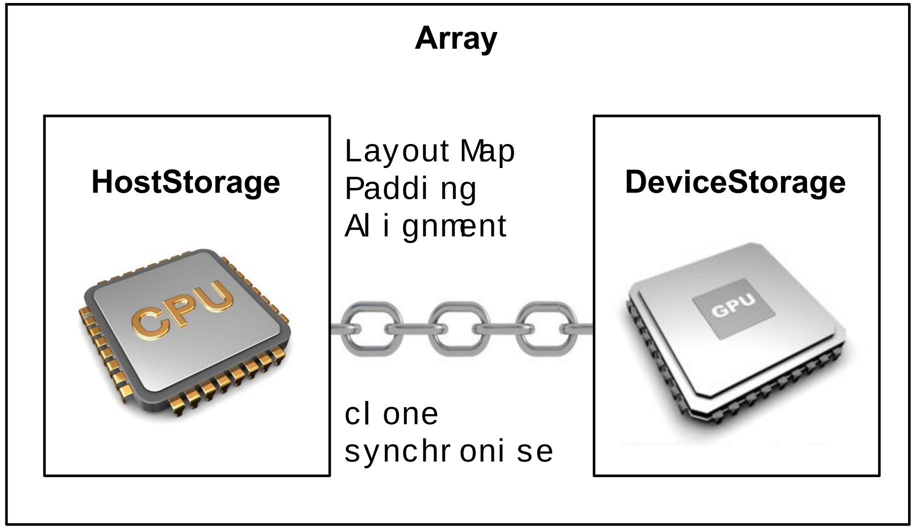




Advection abstraction in IFS based on Atlas

Atlas on GPUs

- **Two linked memory spaces: host (CPU) and device (GPU)**
- **Built on memory resource abstractions**
  - **Memory pools**
  - **CUDA (Nvidia) or HIP (AMD)**
- **Asynchronous execution via ‘streams’**



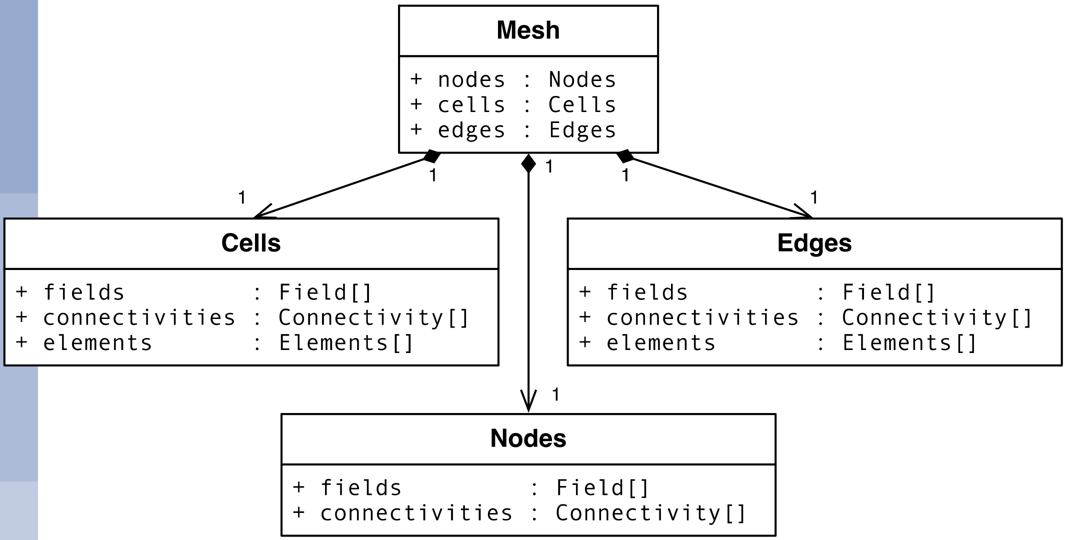
C++ example

```
// 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) = ...
}}

// Allocate and copy field to device
field.updateDevice();

// 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  (OR Kokkos!)
some_cuda_kernel<<<1,1>>>(device_view);
```

Atlas on GPUs with OpenACC for Fortran

- **GPU enabled data structures**

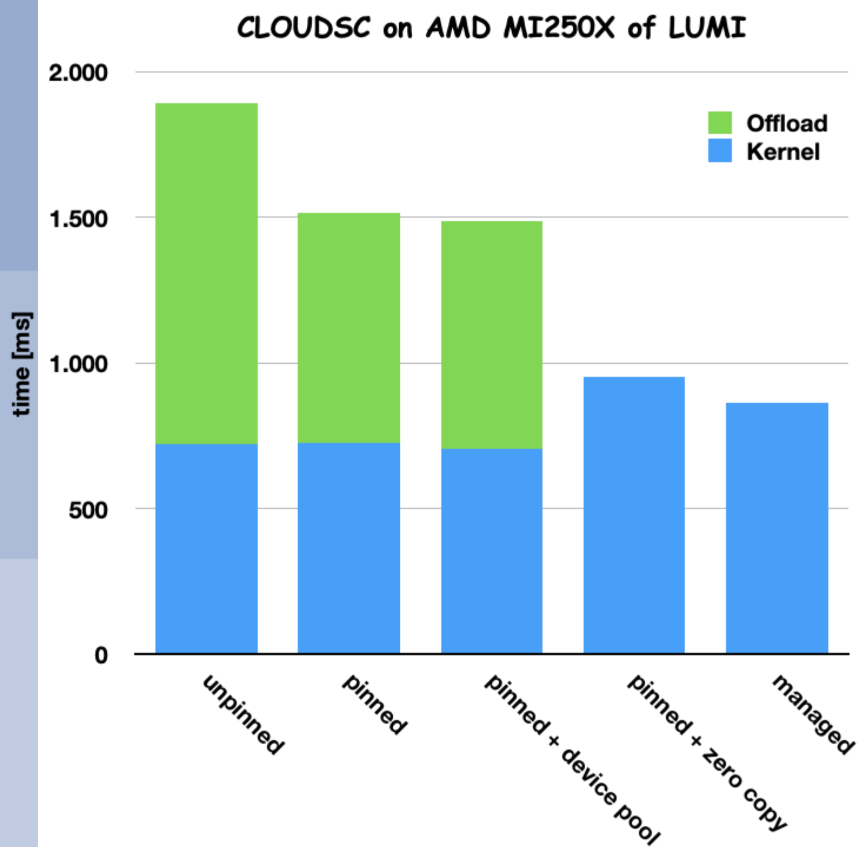
- Cloning mesh to device recursively clones all encapsulated components to device

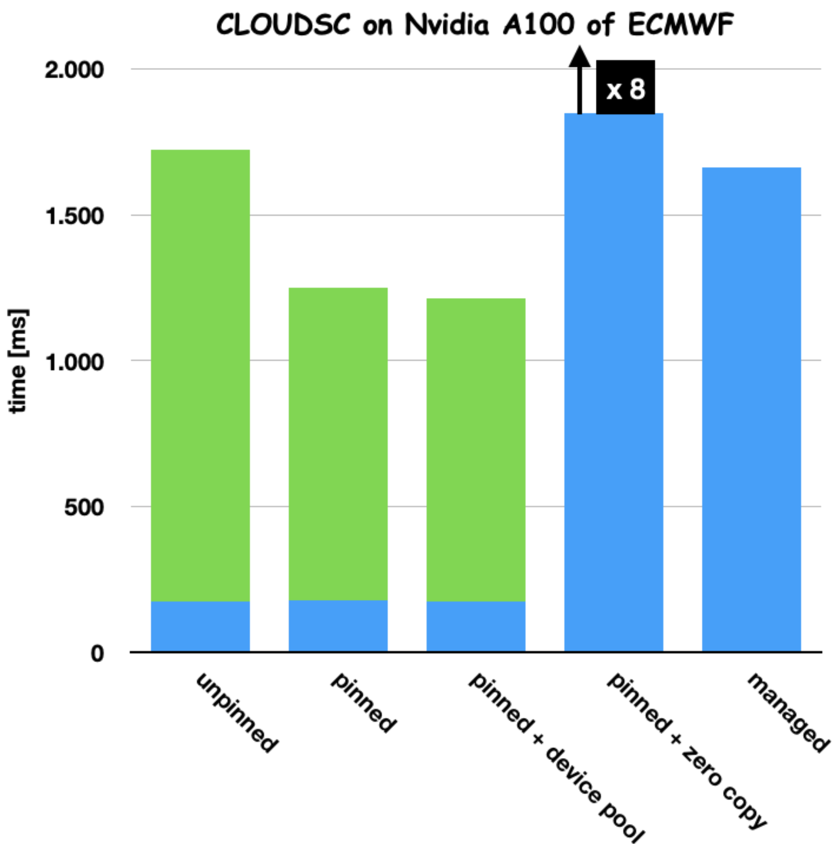


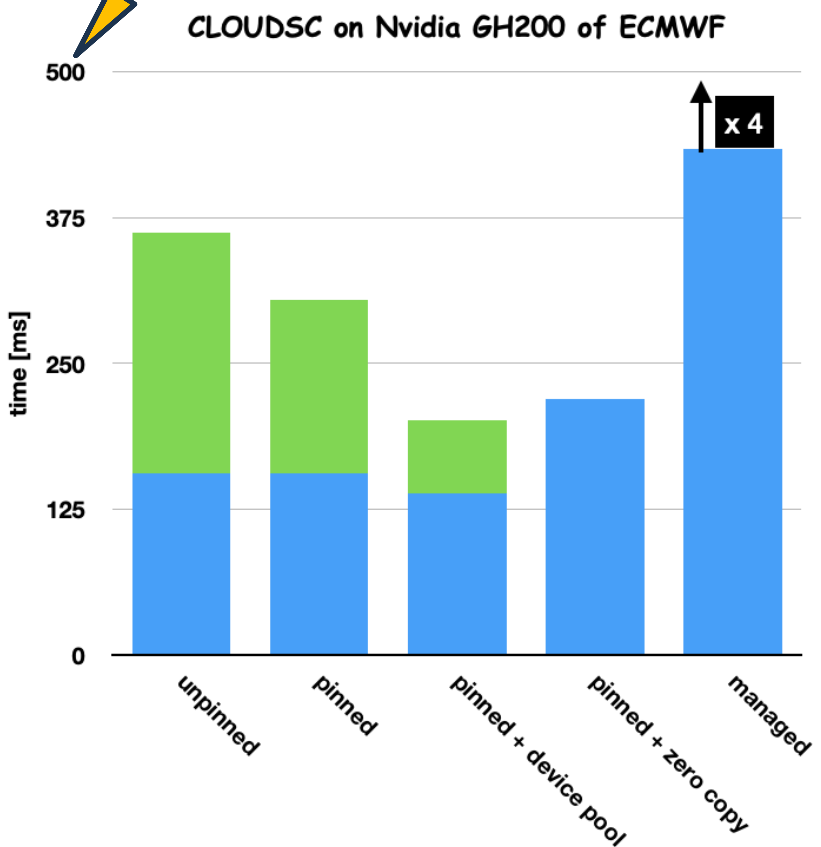

```fortran
type(atlas_Mesh) :: mesh           ! Assume created
type(atlas_mesh_Nodes) :: nodes ! Nodes in the Mesh
type(atlas_Field) :: field_xy    ! Coordinate field of nodes
real(8), pointer :: xy           ! Raw data pointer

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

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

call field_xy%update_device() ! Copy field 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
```

Playing with different memory resources and offloading strategies

- CLOUDSC: ECMWF cloud microphysics scheme with 262144 columns (NPROMA=64)
- Memory management, offloading and GPU data access via Atlas
- Fortran OpenACC with !$ACC DATA DEVICEPTR clause
- Different memory allocation and offloading strategies on AMD MI250X (LUMI), Nvidia A100, Nvidia GH200
  → Strategy which works best for one GPU is not what works best for another!



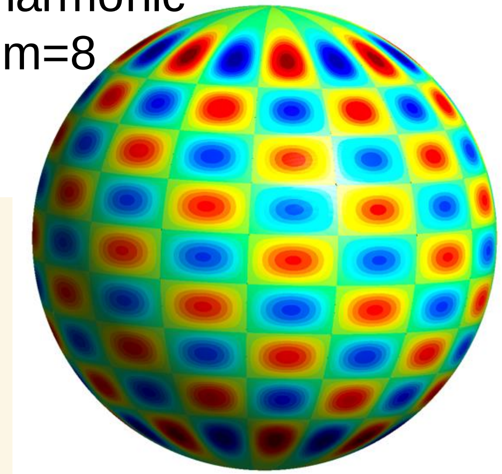




atlas4py

- atlas4py = Python bindings for Atlas
- Developed and hosted by CSCS (github.com/GridTools/atlas4py)
  - Minimum requirement for FVM+GT4Py
  - Agreed to be contributed to our ECMWF hosted atlas repository by CSCS shortly
- More bindings exist and are in process of merging:
  Interpolation, Spectral transforms,
  Halo exchanges, KD-Tree-search, Partitioners,
  MeshGenerators, FunctionSpaces
- Opens up various **parallel** distributed research and data analysis opportunities, with operability with Fortran and C++ algorithms.
- New model development with driver layers and data management in Python but with computations in C++ or Fortran or DSL or … ?

Spherical harmonic
n=16, m=8

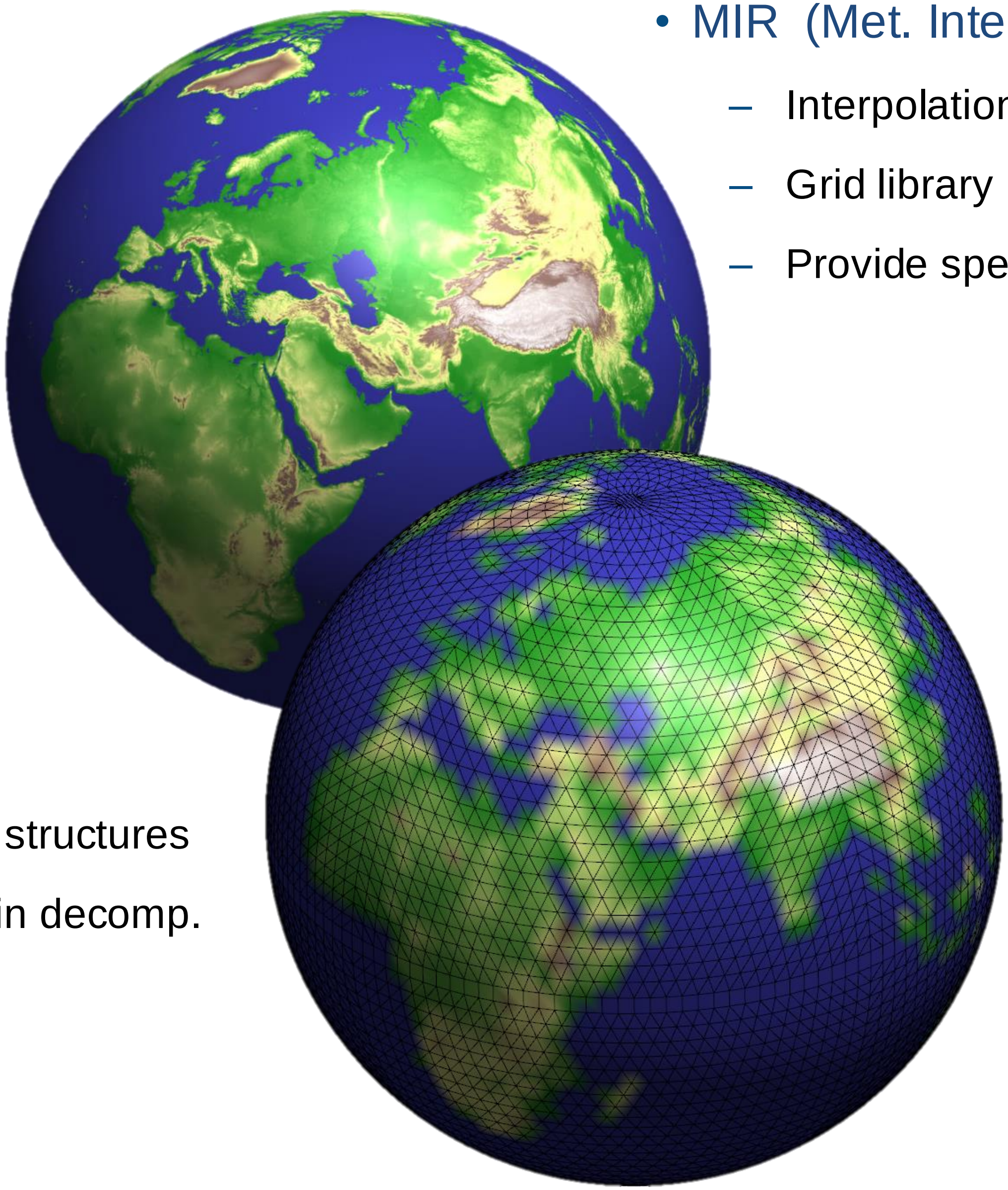


```python
import atlas4py as atlas

atlas.initialize()

grid       = atlas.Grid("O1280")
truncation = 1279

# Create function spaces
fs_sp = atlas.functionspace.Spectral(truncation)
fs_gp = atlas.functionspace.StructuredColumns(grid)

# Create fields (already distributed)
field_sp = fs_sp.create_field(name="sp")
field_gp = fs_gp.create_field(name="gp")

# Initial condition for spectral field
set_spherical_harmonic(field_sp, n=16, m=8)

# Spectral transform
trans = atlas.Trans(grid, truncation)
trans.invtrans(field_sp, field_gp)

# Visualisation of gridpoint field
mesh = atlas.MeshGenerator(type='structured').generate(grid)
atlas.Gmsh("field_gp.msh", coordinates='xyz').write(mesh).write(field_gp)

atlas.finalize()
```

custom python function using numpy array

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



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

- FVM
  - Object Oriented data structures
  - Parallelisation: domain decomp.