Developing ECMWF's next-generation performance-portable atmospheric dynamical core

Sara Faghin-Naini[1]

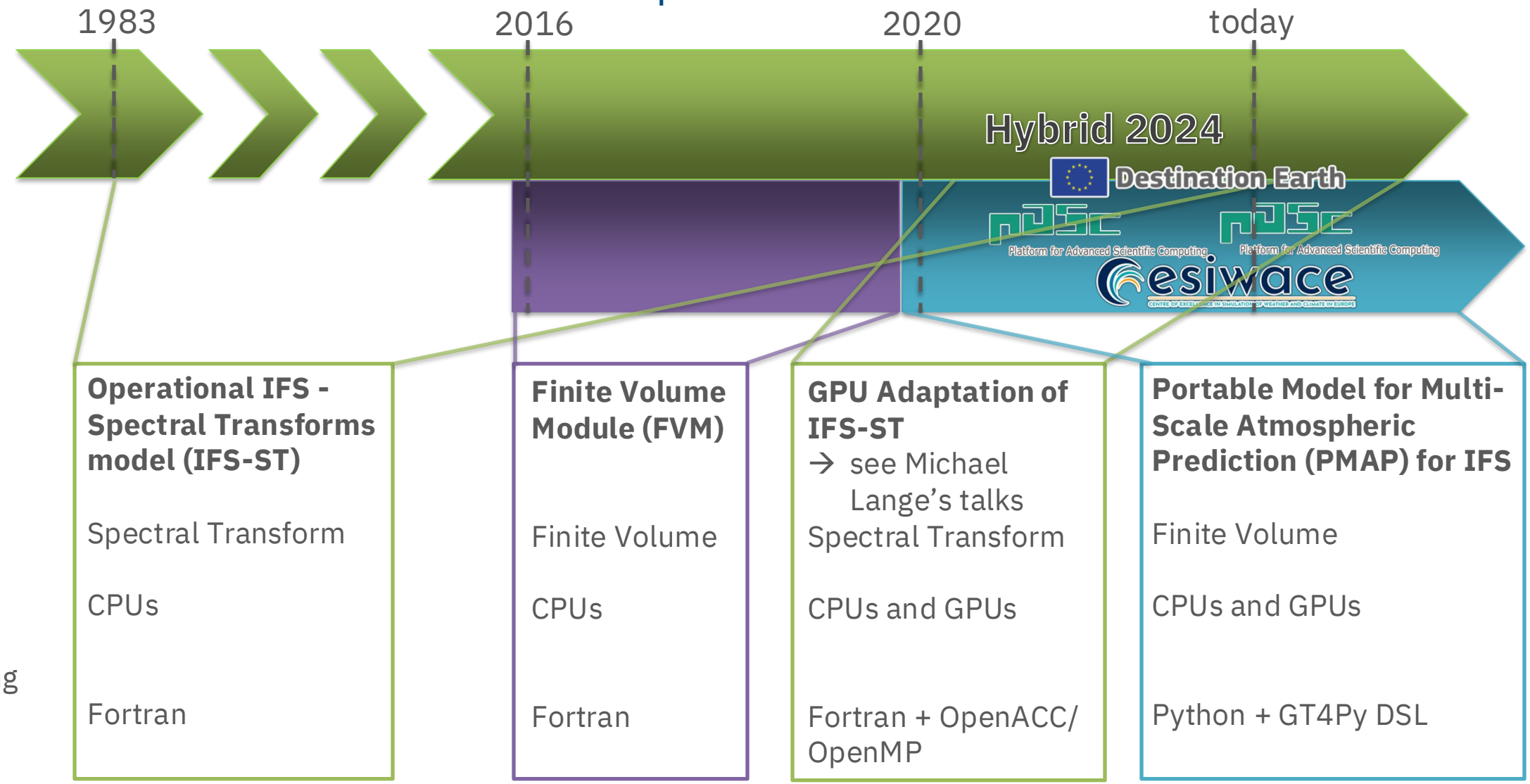
sara.faghihnaini@ecmwf.int

In collaboration with:
Till Ehrengruber[2], Lukas Papritz[3, 1], Stefano Ubbiali[3], Peter Dueben[1], Christian Kühnlein[1]

[1] European Centre for Medium-Range Weather Forecasts
[2] Swiss National Supercomputing Centre (CSCS)
[3] ETH Zurich

ESiWACE3 has received funding from the European High Performance Computing Joint Undertaking (EuroHPC JU) and the European Union (EU) under grant agreement No 101093054.

Numerical models and hardware adaptation at ECMWF



	Operational IFS - Spectral Transforms model (IFS-ST)	**Finite Volume Module (FVM)**	**GPU Adaptation of IFS-ST** → see Michael Lange's talks	**Portable Model for Multi-Scale Atmospheric Prediction (PMAP) for IFS**
Numerical scheme:	Spectral Transform	Finite Volume	Spectral Transform	Finite Volume
Hardware:	CPUs	CPUs	CPUs and GPUs	CPUs and GPUs
Programming language:	Fortran	Fortran	Fortran + OpenACC/ OpenMP	Python + GT4Py DSL

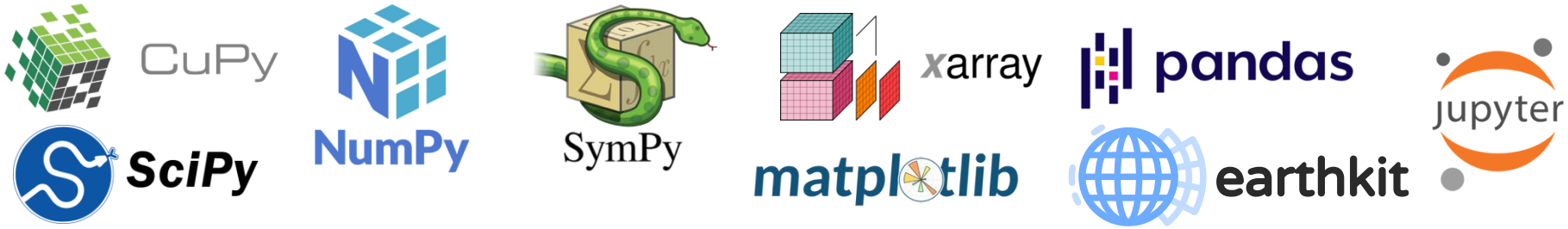
Portable Model for Multi-Scale Atmospheric Prediction (PMAP)

- Compact-stencil dynamical core in a high-level **Python-based** framework, designed to provide sustainable **performance-portability** leveraging **GT4Py** (GridTools for Python)
- **Joint effort** of
  - ECMWF (Core & ESiWACE3)
  - CSCS and
  - ETH Zurich (PASC-project)

  to port FVM to Python using the GT4Py domain specific library
- Solves the **non-hydrostatic fully-compressible equations** using **3D semi-implicit time-integration** and **finite-volume non-oscillatory advection**
- Model formulation supports **unstructured and structured grids, conservation, 3D diffusion and subgrid-scale turbulence, scalability, steep orography at sub-kilometer resolution**,…
- **Two configurations:**
  - **PMAP-LES**: **regional** model on quadrilateral grids with Large-Eddy Simulation capabilities
    → see Stefano Ubbiali's talk
  - **PMAP-GO**: **Global** dynamical core on the IFS **Octahedral** grid

Why GT4Py?

- **GT4Py** is embedded in the **Python** eco-system



- **Machine Learning** interoperability: **automatic differentiation** using JAX straight forward

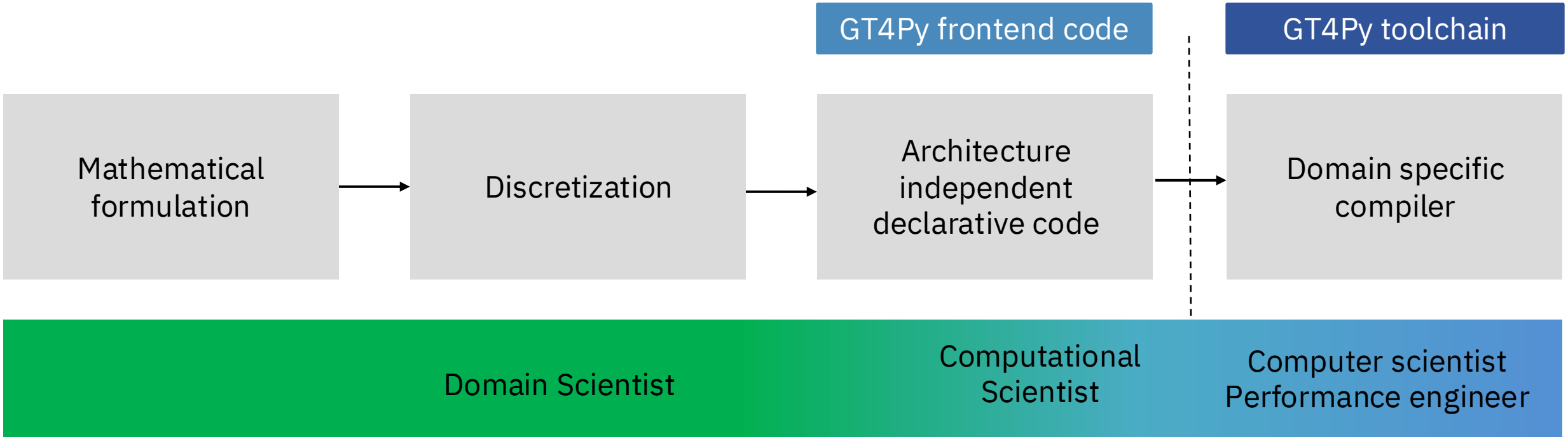
- **Portability**:
    - Example: **Porting to AMD GPUs**: ecosystem supported on AMD → model runs **without any code changes** after providing the software stack

Further NWP models using GT4Py:

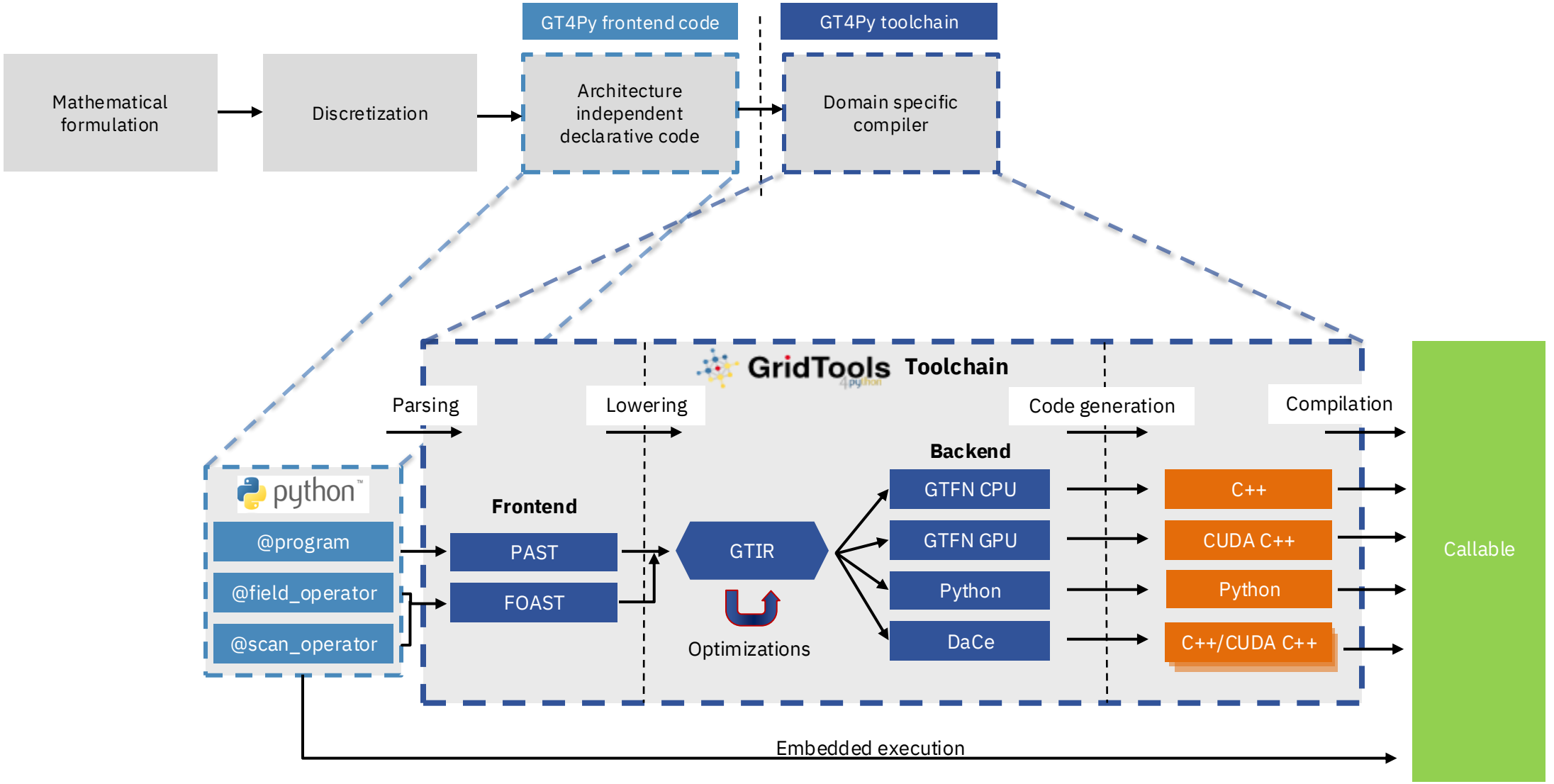
- **ICON** atmospheric model dynamics and physics ported to GT4Py (next) by MeteoSwiss, EXCLAIM project at ETH Zurich and CSCS → see Daniel Hupp's and Magdalena Luz's talk
- **Pace** is a GT4Py (cartesian) implementation of the FV3GFS / SHiELD atmospheric model of NOAA/GFDL by Allen Institute for AI (AI2)

Separation of concerns

- **Domain scientists** write **high-level code** specifying the algorithm and the mathematical operations
- **GT4Py optimizing toolchain** transforms high-level code into **efficient, architecture-tuned** implementations



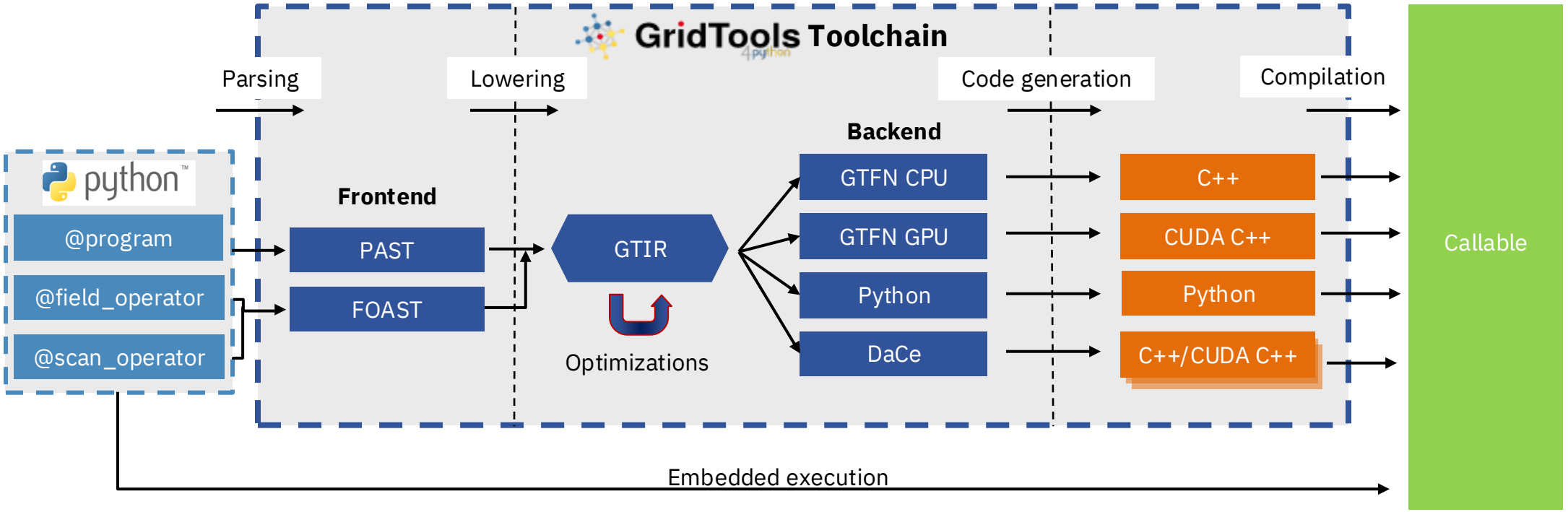
GT4Py toolchain



sara.faghihnaini@ecmwf.int

GT4Py toolchain

- **Code generation** optimized **for** a **specific architecture**, leveraging **knowledge** of the **typical computation patterns in the domain**
- **Backend selects implementation strategy** (parallelism, memory layout, etc.)
- Has built-in integration to the **DaCe** (Data-Centric Parallel Computing) library
- **Backends** can be **added** to provide efficient implementations **for new platforms**



GT4Py simple illustration: advection-diffusion equation

$$\frac{\partial u}{\partial t} = -a\frac{\partial u}{\partial x} + D\frac{\partial^2 u}{\partial x^2}$$

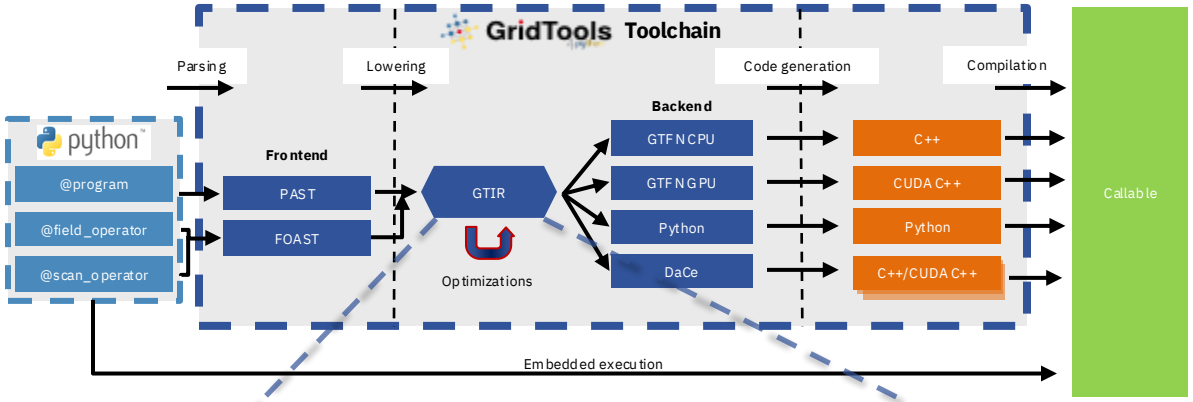
Discretization with first order space derivatives ($\Delta x = 1$) and forward Euler in time:

$$u_i^{n+1} = u_i^n - a\,\Delta t \cdot (u_i^n - u_{i-1}^n) + D\,\Delta t \cdot (u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

$a, D \in \{0, 1\}$ used to dis-/enable advection and diffusion

```
@gtx.field_operator(
    static_params=["enable_advection",
                   "enable_diffusion"])
def solve_advection_diffusion(
        u: gtx.Field[[I], float],
        dt: float,
        enable_advection: bool,
        enable_diffusion: bool,
):
    u_new = u
    if enable_advection:
        u_new = u_new - dt * (u - u(I - 1))
    if enable_diffusion:
        u_new = u_new +
          dt * (u(I + 1) - 2.0 * u + u(I - 1))
    return u_new
```

GT4Py optimization pass: constant folding (excerpt)




```
if_(True, true_branch, false_branch)
```

```
if (
    cpm.is_call_to(node, "if_")
    and isinstance(node.args[0], itir.Literal)
):
    if new_node.args[0].value == "True":
        return node.args[1]
    else:
        return node.args[2]
```

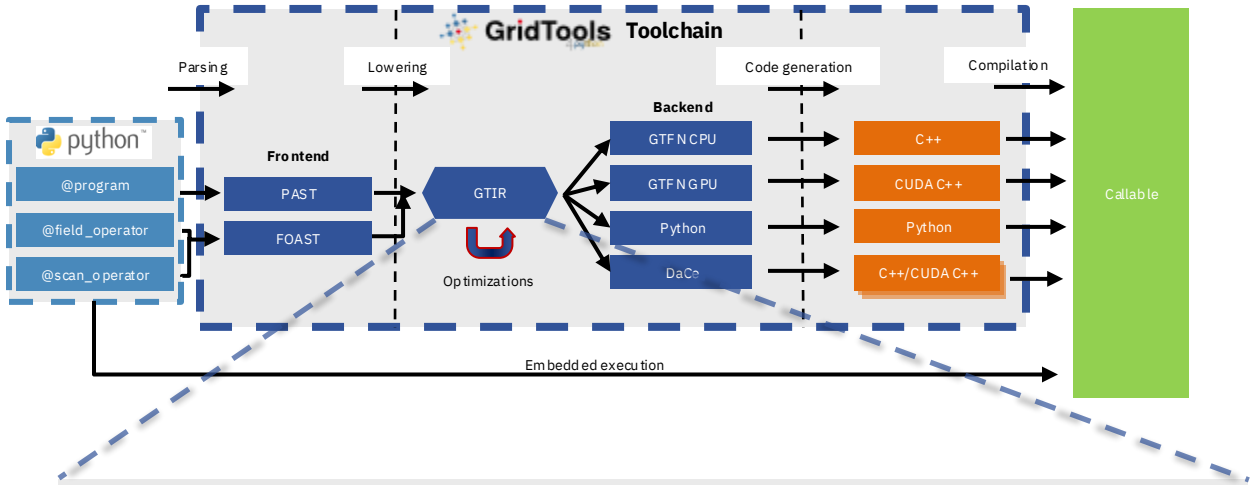
```
true_branch
```

- Thanks to our **functional** intermediate representation (**IR**) **no side effects** need to be considered in all passes
- **Further optimizations**:
  - Temporary fusion
  - Function inlining
  - Common subexpression elimination
  - …

GT4Py toolchain example

- Possible to specify **static parameters**, which are just-in-time- or pre-compiled

```
@gtx.field_operator(
    static_params=["enable_advection",
                   "enable_diffusion"])
def solve_advection_diffusion(
        u: gtx.Field[[I], float],
        dt: float,
        enable_advection: bool,
        enable_diffusion: bool,
):
    u_new = u
    if enable_advection:
        u_new = u_new - dt * (u - u(I - 1))
    if enable_diffusion:
        u_new = u_new +
            dt * (u(I + 1) - 2.0 * u + u(I - 1))
    return u_new
```

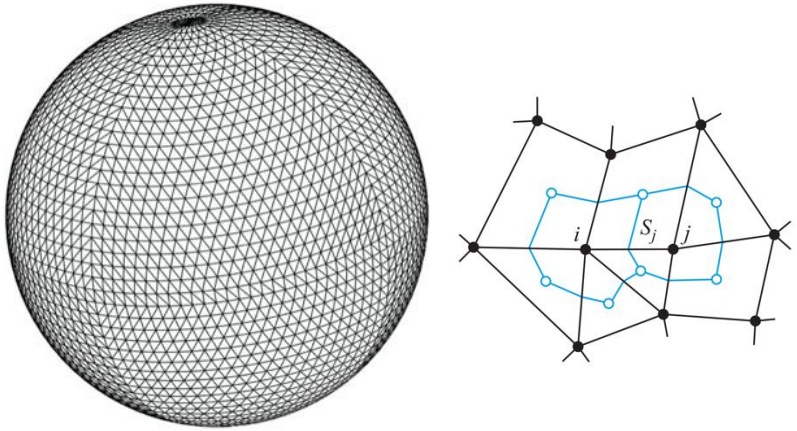



```
if_(enable_advection, true_branch, false_branch)
                          ↓
if_(False, true_branch, false_branch)
```

PMAP-GO details

- **Global dynamical core** using GT4Py.next
- Local **low-volume communication** footprint
- Using **height-based terrain-following vertical coordinates**
- Supports **semi-implicit integration with adaptive time step**
- Executes entirely in either **double or single precision**
- Supports **CPUs** and **GPUs** via different GT4Py backends
- Uses median-dual FV mesh built around the nodes of the **octahedral reduced Gaussian grid**
- Grid generation with ECMWF's Atlas[1] library and **atlas4py**[2] Python bindings

```
from atlas4py import StructuredGrid
grid = StructuredGrid("O24")
mesh = AtlasMesh.generate(grid)
```



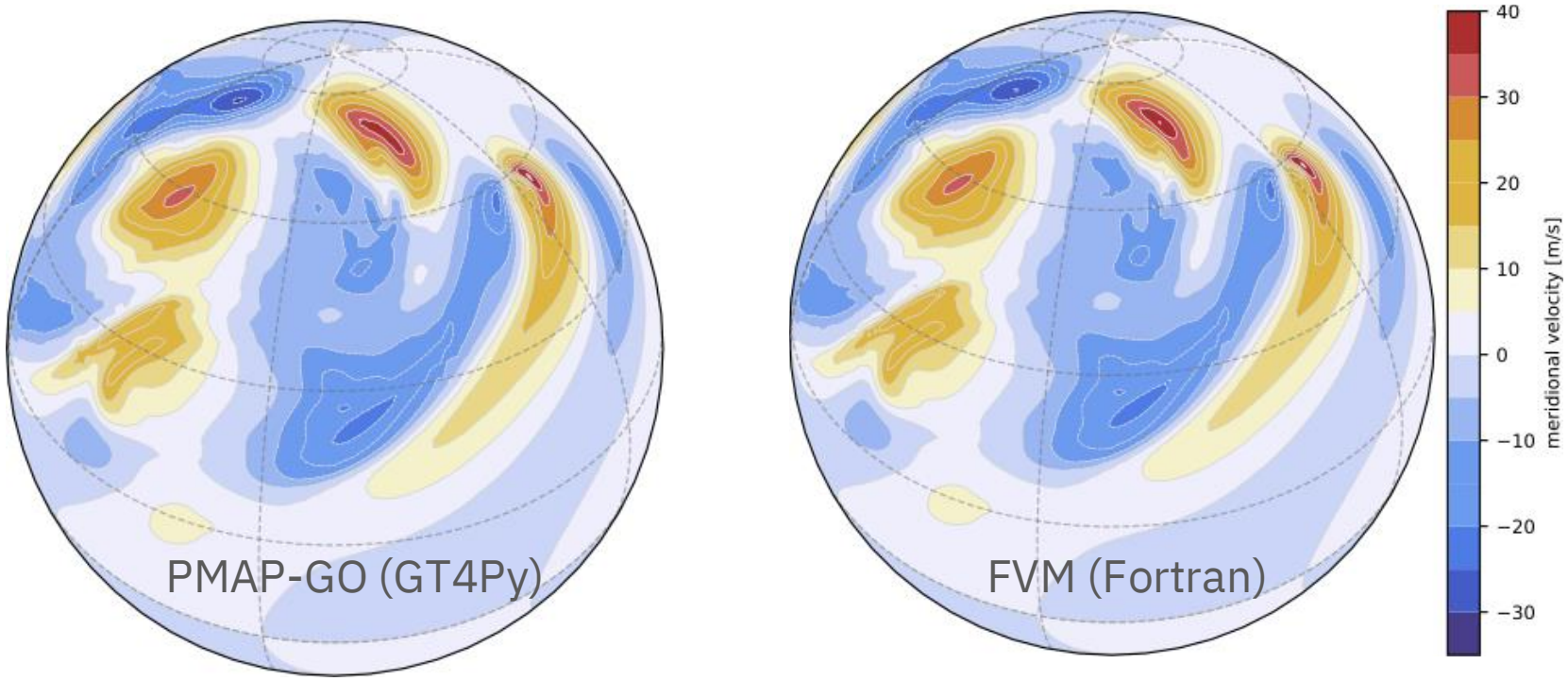
Octahedral grid O24 (left) and schematic of the median-dual mesh (right).

[1] Atlas: W. Deconinck et al. “Atlas : A library for numerical weather prediction and climate modelling”. In: Computer Physics Communications 220 (2017) DOI: 10.1016/j.cpc.2017.07.006

[2] atlas4py: https://github.com/GridTools/atlas4py

PMAP-GO simulations

- PMAP reproduces FVM and at the same time provides a fully portable future-proof model in Python

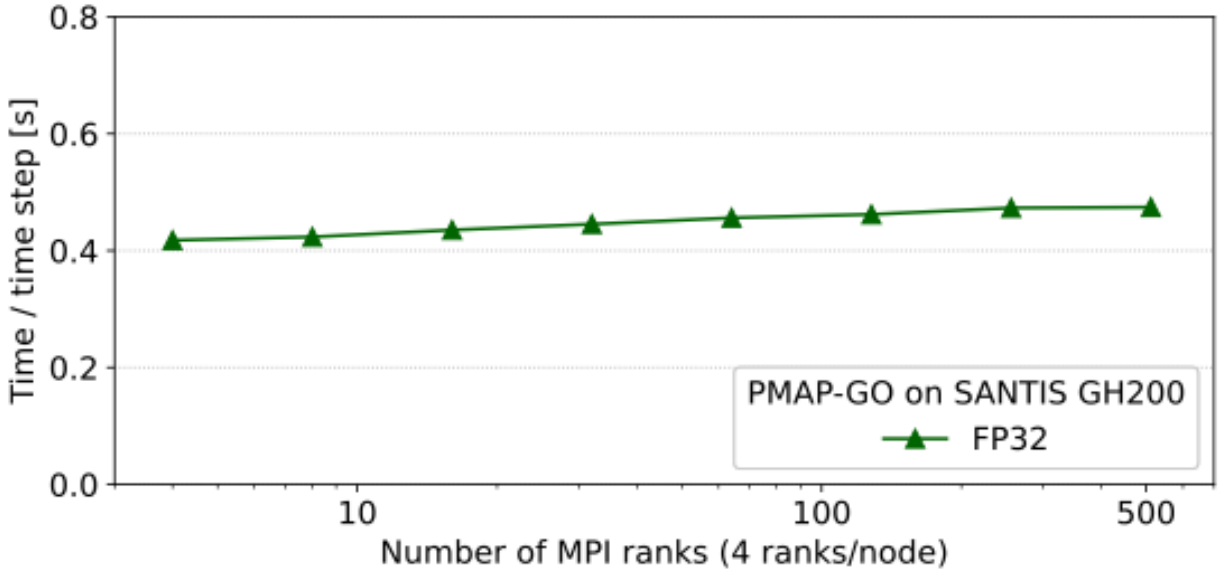


DCMIP-2016 baroclinic wave benchmark with left: PMAP-GO (GT4Py) and right: FVM (Fortran) on the same O160 grid: meridional wind (m/s, shaded) on day 15 at an altitude of 2km.

sara.faghihnaini@ecmwf.int

PMAP-GO scaling results

- **Weak scalability** of the **dynamical core**
- **Distributed memory implementation** using the exascale-ready library **GHEX**[1]
- Significant **optimizations** are still **ongoing**



DCMIP-2016 baroclinic wave benchmark: early weak scaling results on the CSCS Alps-Santis cluster (Grace Hopper GH200 GPUs).
Computational grids: O640 (for 4 ranks) to O7240 (for 512 ranks)

[1]GHEX: https://github.com/ghex- org/GHEX

GridTools 4python

https://github.com/GridTools/gt4py

ECMWF esiwace CENTRE OF EXCELLENCE IN SIMULATION OF WEATHER AND CLIMATE IN EUROPE

Co-funded by the European Union

EuroHPC Joint Undertaking

CSCS Centro Svizzero di Calcolo Scientifico Swiss National Supercomputing Centre

ETH zürich

PASC Platform for Advanced Scientific Computing

Thank you for listening!