GT4Py

A Python framework for performance portable weather and climate applications

Till Ehrengruber

ETH zürich

GridTools 4python

XCLAIM

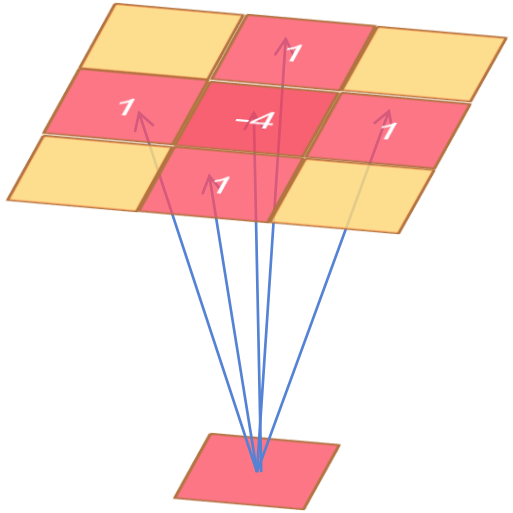
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Vison

Make it **easy** to write **performant** weather and climate applications

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

Prototypical example: 2D Laplacian



$$f \mapsto \Delta f$$

Mathematical formulation

$$\Delta f_{i,j} \approx -4f_{i,j} + f_{i-1,j} + f_{i+1,j} + f_{i,j-1} + f_{i,j+1}$$

How do we achieve good performance?

How do we achieve good performance?

⇒ increase data locality

Traditional approach

```
for (int i = 0; i < n; ++i) {
  for (int j = 0; j < n; ++j) {
    g(i, j) = -4*f(i, j) + f(i-1, j) + f(i, j-1)
                         + f(i+1, j) + f(i, j+1);
  }
}
```

Traditional approach

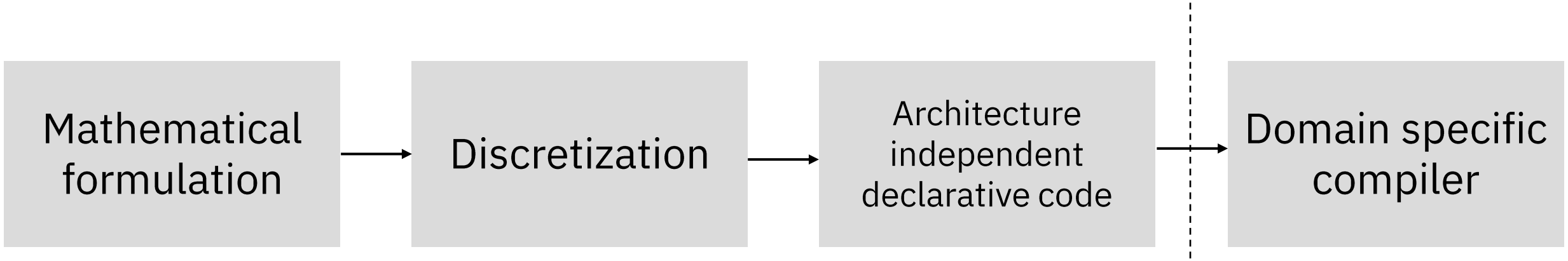
```
#pragma omp parallel for collapse(2)
for (int i = 0; i < n; i+=block_size) {
  for (int j = 0; j < n; j+=block_size) {
    for (int ii = i; ii < i+block_size; ++ii) {
      for (int jj = j; jj < j+block_size; ++jj) {
        g(ii, jj) = -4*f(ii, jj) + f(ii-1, jj) + f(ii, jj-1)
                                 + f(ii+1, jj) + f(ii, jj+1);
      }
    }
  }
}
```

Traditional approach: Optimization directives

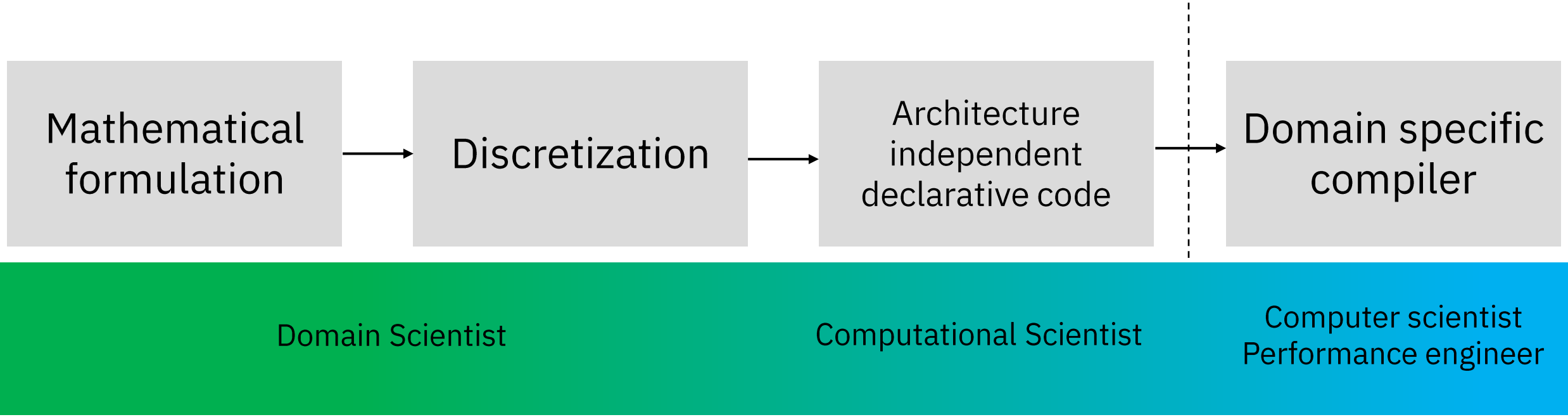
Model	Number of directives
FVM (Fortran)	1365 x `OMP PARALLEL`
ICON (Fortran)	1243 x `ACC PARALLEL`
GFS-FV3 (Fortran)	333 x `OMP PARALLEL`

Optimization directives are not scalable!

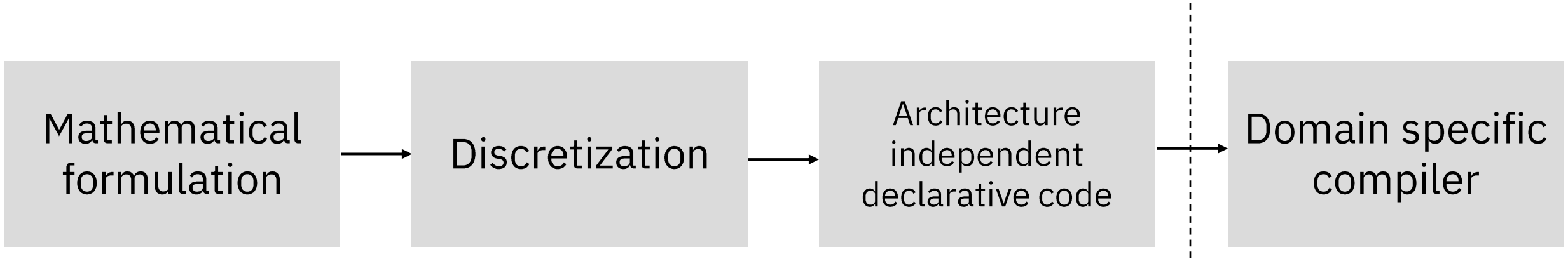
Separation of concerns



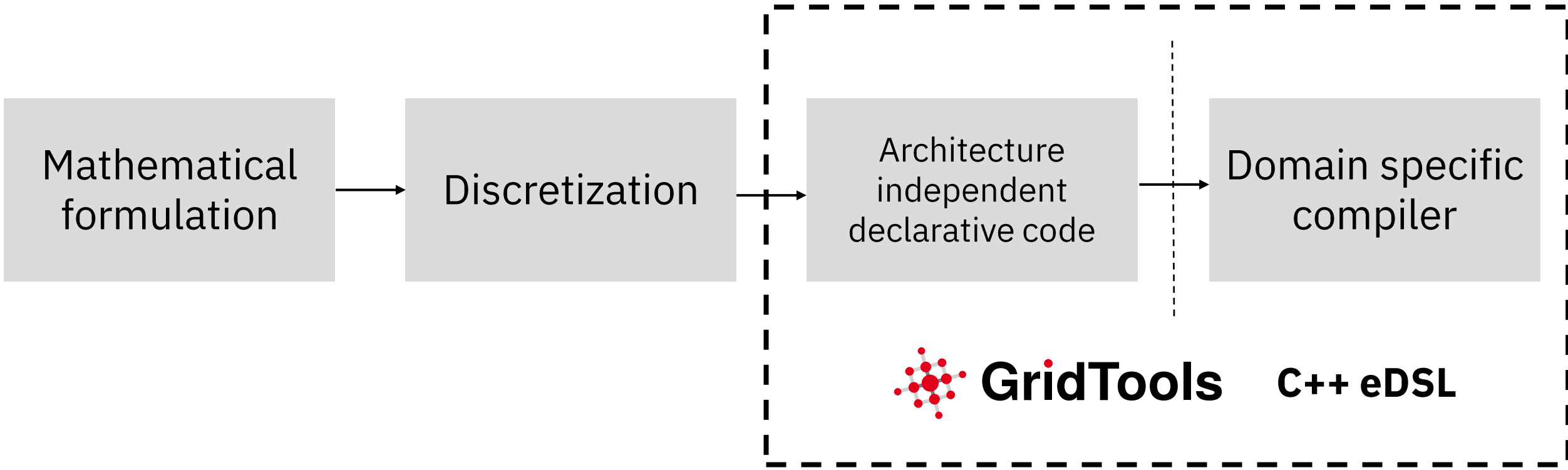
Separation of concerns



Separation of concerns

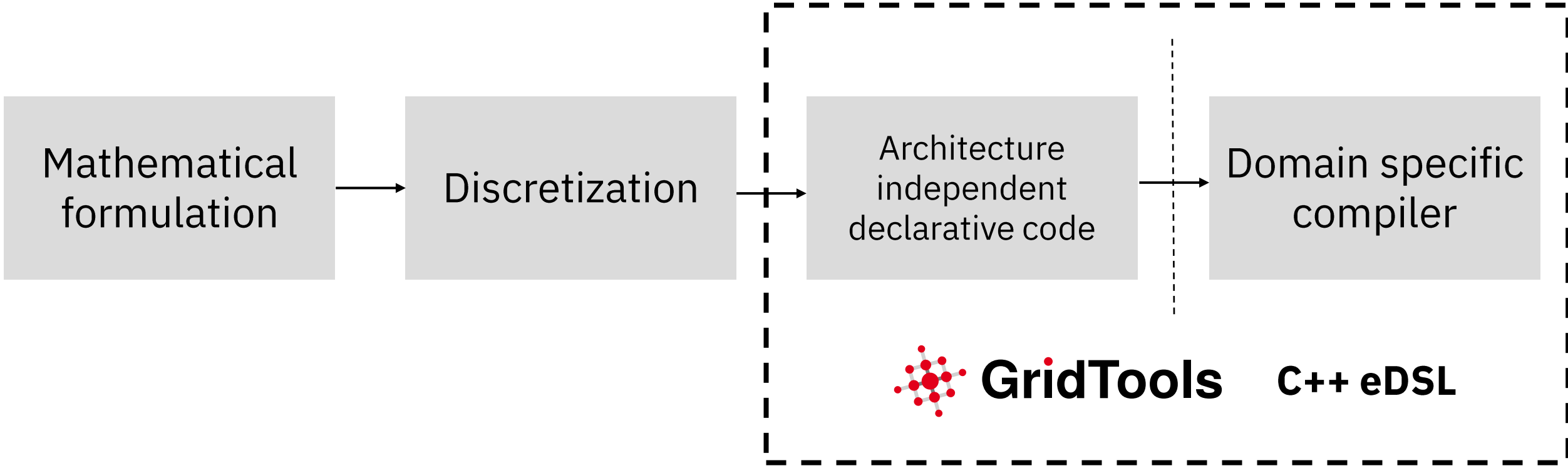


GridTools C++ eDSL



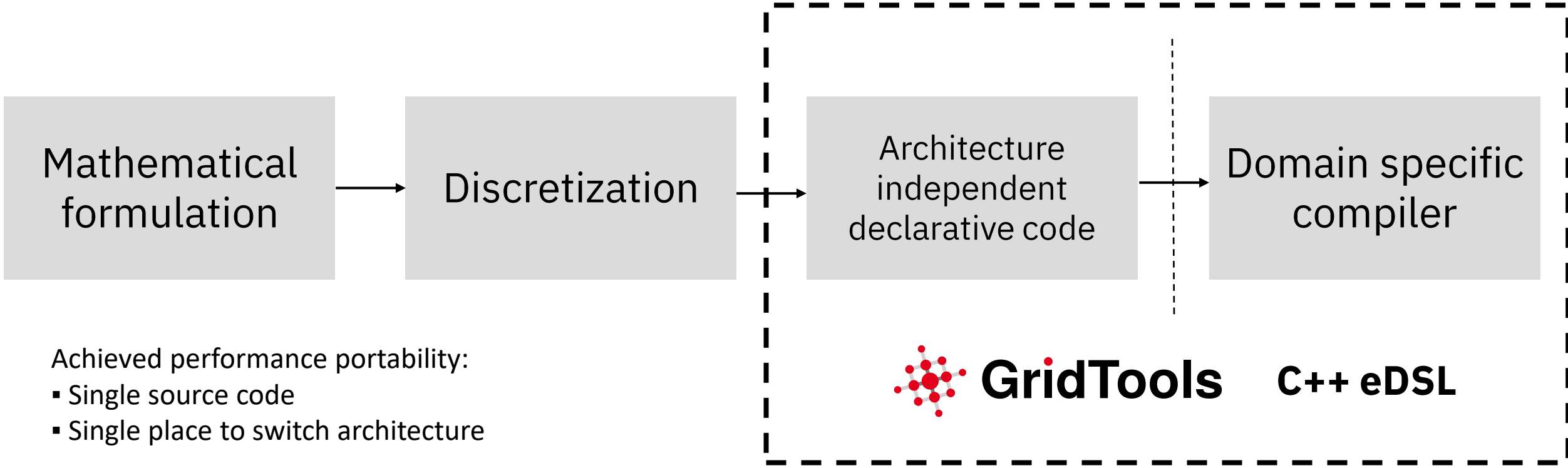
GridTools C++ eDSL

Operational at MeteoSwiss since 2019 (COSMO)



GridTools C++ eDSL

Operational at MeteoSwiss since 2019 (COSMO)

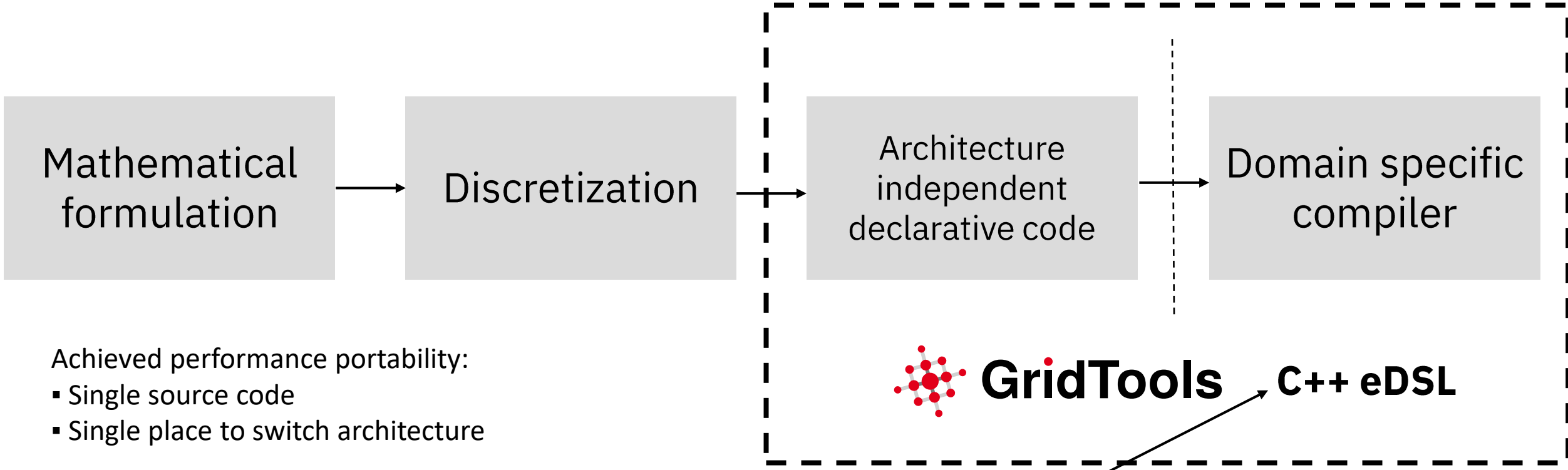


Achieved performance portability:

- Single source code
- Single place to switch architecture

GridTools C++ eDSL

Operational at MeteoSwiss since 2019 (COSMO)

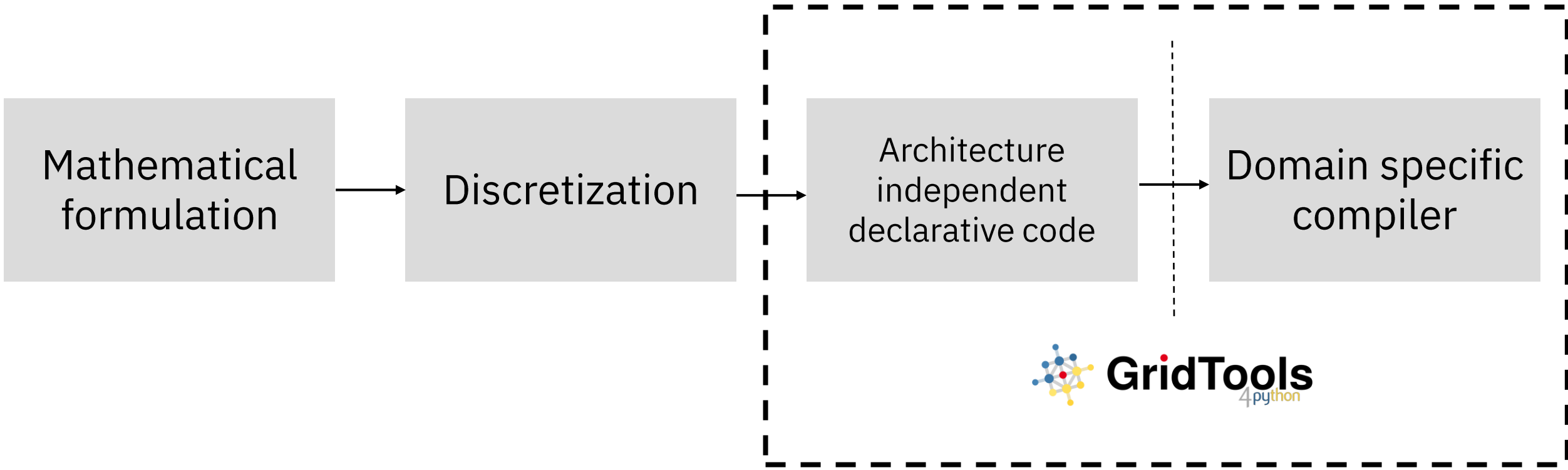


Achieved performance portability:

- Single source code
- Single place to switch architecture

We did not manage to convince the domain scientist to use this language

GT4Py


Mathematical formulation
Discretization
Architecture independent declarative code
Domain specific compiler
GridTools
4python

GT4Py

- Domain scientists write high-level code specifying the algorithm and the mathematical operations
  - No need to pollute the code with high-performance optimizations techniques
  - Hardware-agnostic description of the computation ➜ performance portability
- GT4Py works as an optimizing compiler with several backends
  - Code generation optimized for a specific architecture
  - Backend selects implementation strategy (parallelism, memory layout, etc.)
  - Backends can be added to provide efficient implementations for new platforms
  - Leverages knowledge of the typical computation patterns in the domain
- Smooth integration with Python scientific ecosystem provides access to a huge collection of Python libraries
  - data processing, analysis, visualization, etc.

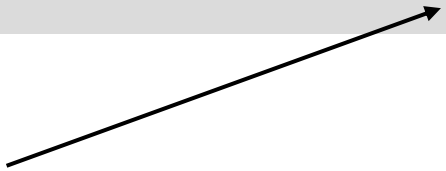
Laplacian in GT4Py

Mathematical formulation

$$\Delta f \approx -4f_{i,j} + f_{i-1,j} + f_{i+1,j} + f_{i,j-1} + f_{i,j+1}$$

GT4Py code

```
@field_operator
def laplacian(f: Field[[I, J], float]):
  return -4*f + f(I-1) + f(I+1) + f(J-1) + f(J+1)
```



Hardware-agnostic description
of the computation

Operators

Field operator (`@field_operator`)

Covering most patterns of explicit finite-difference and finite-volume discretizations, multiple field operations can be grouped together into a field operator.

```
@field_operator
def edge_average(
  vertex_field: Field[[Vertex], float]
) -> Field[[Edge], float]:
  return 0.5*(vertex_field(E2V[0])+vertex_field(E2V[1]))
```

Field operators are composable, allowing the description of high-level operators from basic building blocks.

```
laplap = laplacian(laplacian(f))
```

Scan operator (`@scan_operator`)

Scan operators are useful for expressing computations with dependencies across an entire dimension, which commonly occur in implicit solvers and physical parametrizations. The output from the previous level (i.e., `k+1` or `k−1`, depending on the direction) is used by a scalar function to derive a new value for the current grid point, iteratively building up a complete field.

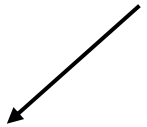
```
@scan_operator(axis=KDim, forward=True, init=0.0)
def simple_scan_operator(
    carry: float, current_value: float
) -> float:
    return carry + current_value


result = simple_scan_operator(inp_field)
```

Hardware Portability

```
@field_operator
def laplacian(f: Field[[I, J], float]):
    ...
```

Hardware Portability

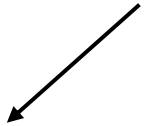
Single line change to switch to a different hardware architecture



```
@field_operator(backend=gtfn_cpu)
def laplacian(f: Field[[I, J], float]):
    ...
```

Hardware Portability

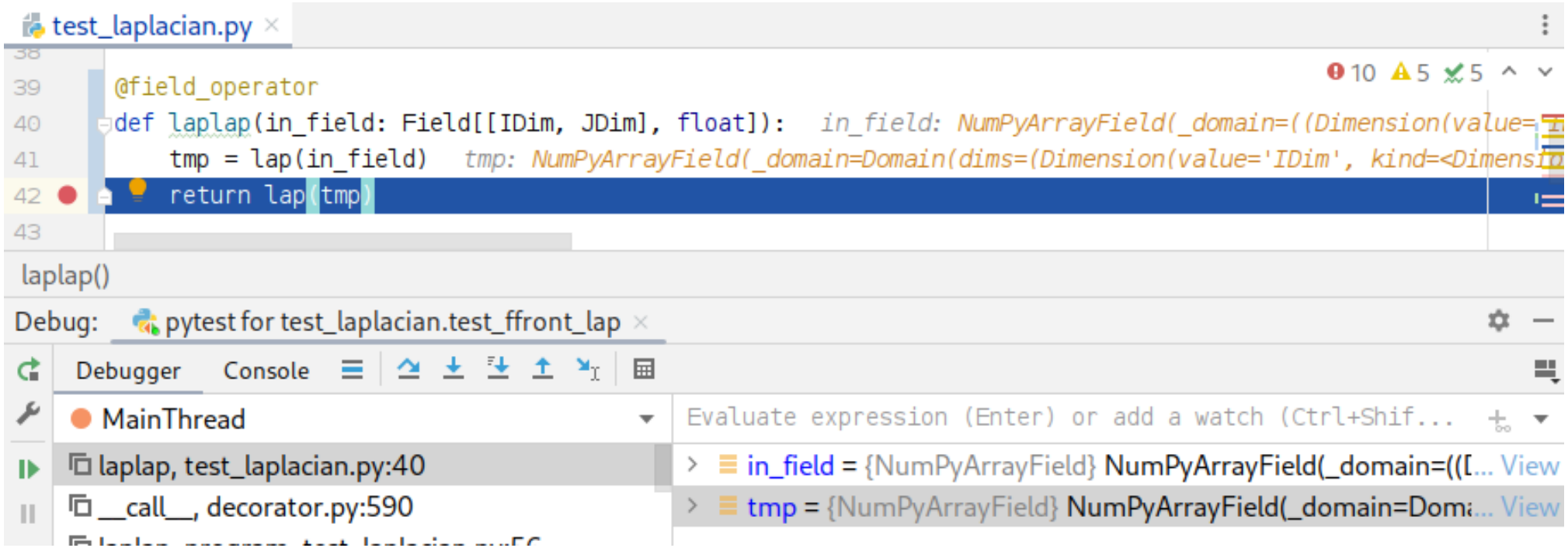
Single line change to switch to a different hardware architecture



```
@field_operator(backend=gtfn_gpu)
def laplacian(f: Field[[I, J], float]):
    ...
```

Debuggability

Intermediate results can be inspected using regular debugging tools (at the cost of performance).

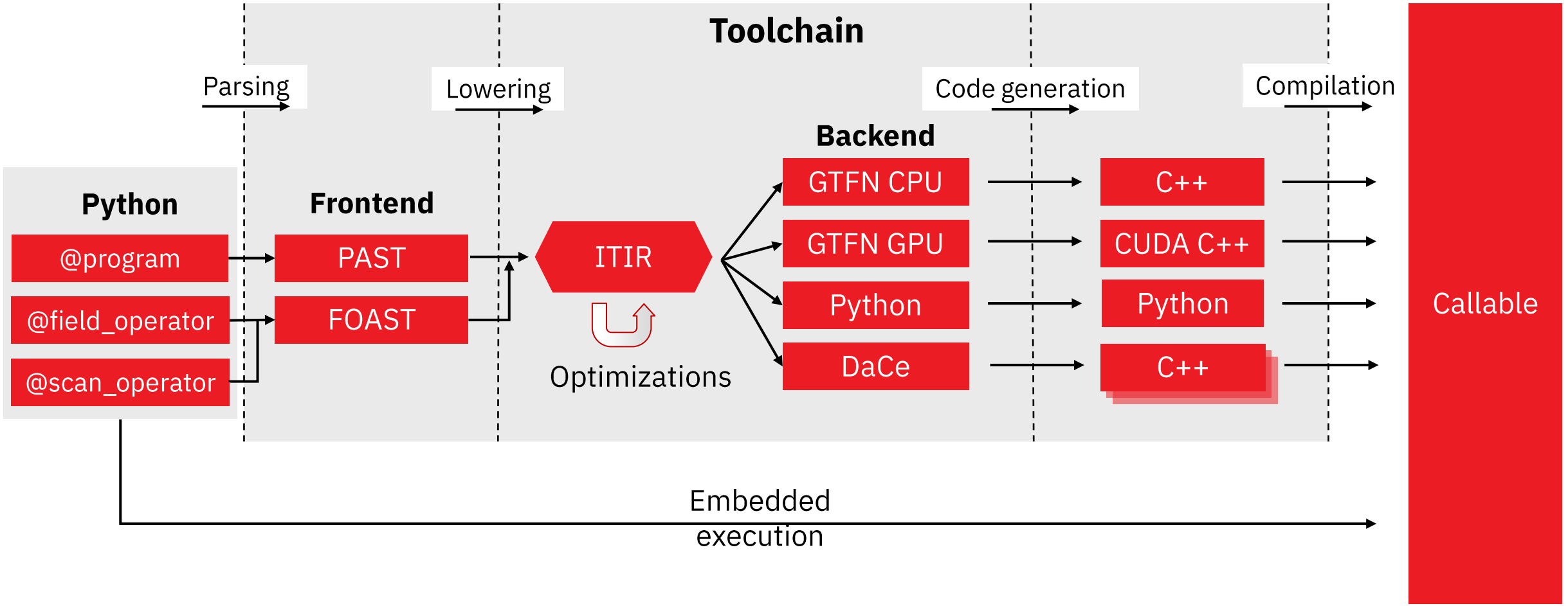


Interoperability

GT4Py can be integrated into existing Fortran (or C++) code

- Fortran driver, Python operators
- Incremental transition process towards Python
  - Approach taken in Exclaim project (ICON)
- Allows continuous validation during porting
- Some limitations persist
  - Memory layout fixed until everything ported (or costly reshuffling / H2D transfers)

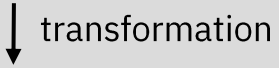
GT4Py Toolchain



Sample optimization: Constant folding

Pattern

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

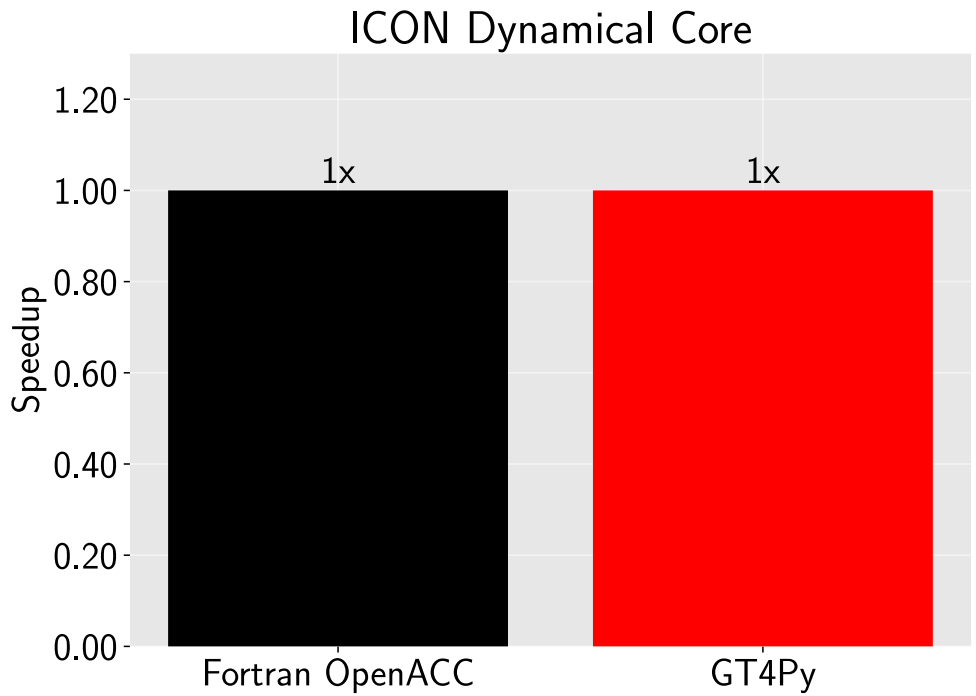


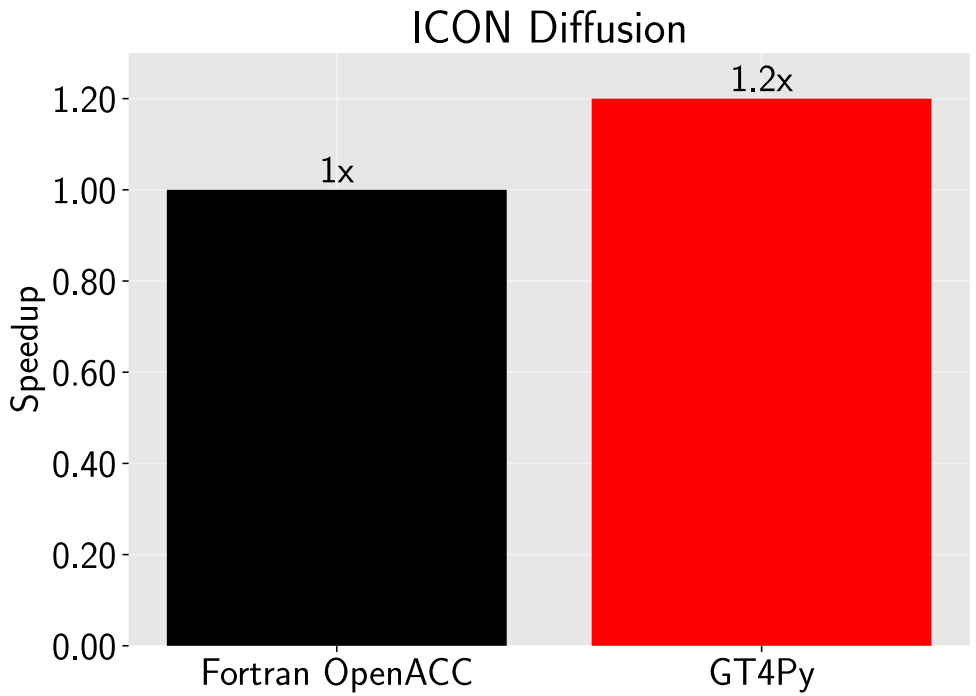

```
true_branch
```

Optimization pass (excerpt)

```
if (
    isinstance(new_node.fun, itir.SymRef)
     and new_node.fun.id == "if_"
     and isinstance(new_node.args[0], itir.Literal)
):
    if new_node.args[0].value == "True":
        new_node = new_node.args[1]
    else:
        new_node = new_node.args[2]
```

Performance

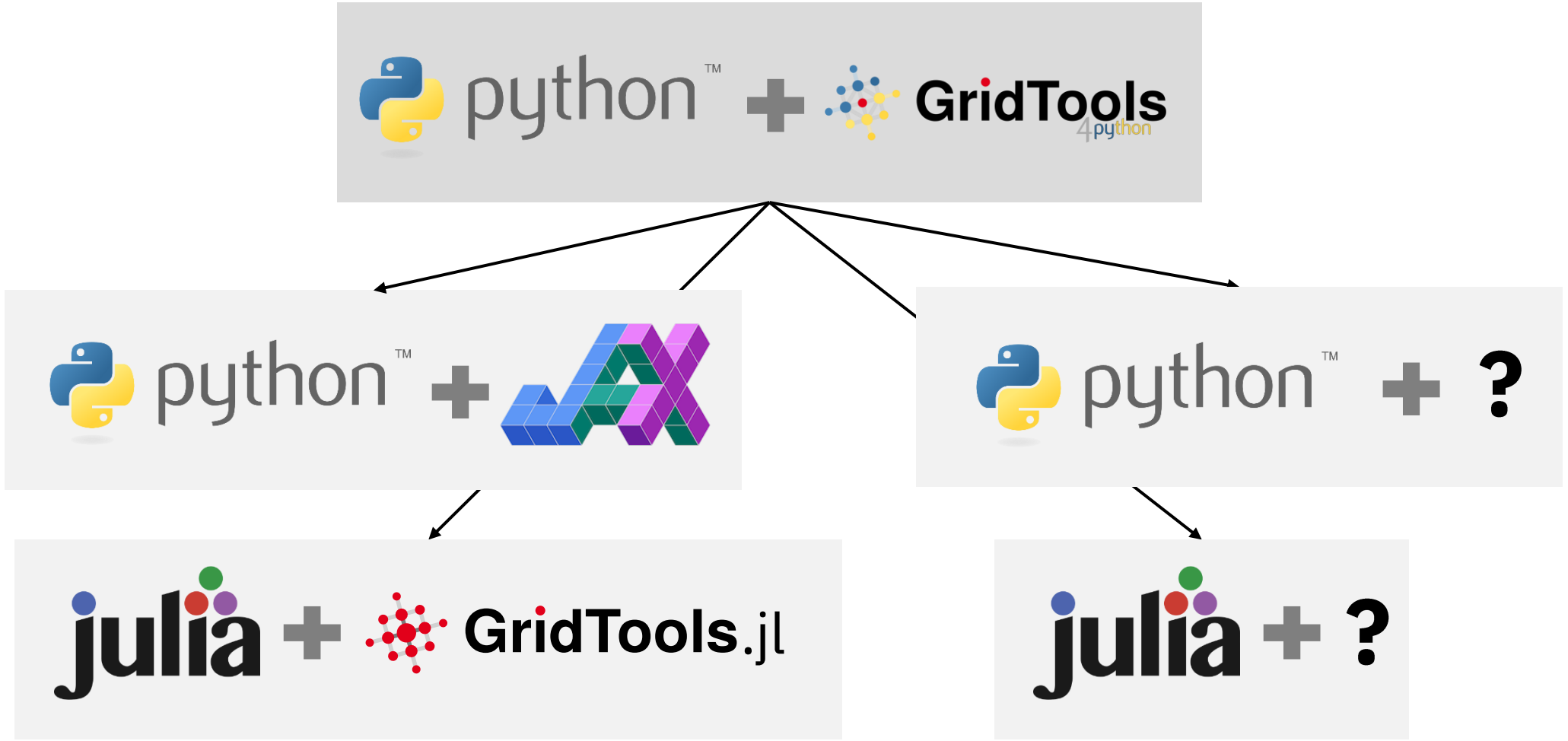




Hardware:
NVIDIA® V100 GPUs 32GB
2 x Intel® Xeon® Skylake 6134
384 GB DDR4

Software:
cuda/11.2.2
nvhpc/21.2-cuda-11.2

GT4Py is not a dead end!



NWP & climate applications

- **ICON** atmospheric model dynamics and physics ported to declarative GT4Py (V2) by MeteoSwiss, EXCLAIM project at ETH Zurich
  ➜ See Daniel Hupp's poster
- **FVM** for IFS porting and further development with GT4Py (V1+V2) by ECMWF, CSCS, and PASC-funded project KILOS at ETH Zurich.
  ➜ See Christian Kühnlein's talk
- **Pace** is GT4Py (V1) implementation of the FV3GFS / SHiELD atmospheric model of NOAA and GFDL by Allen Institute for AI (AI2)
  ➜ See Oliver Elbert's talk
  ➜ See Christopher Kung's talk

Thanks!

https://github.com/GridTools/gt4py

M. Bianco[1], Nina Burgdorfer[3], T. Ehrengruber[1], N. Farabullini[2], A. Gopal[2], R. Häuselmann[1], Daniel Hupp[3], P. Kardos[2], S. Kellerhals[2], M. Luz[2], C. Müller[3], E. G. Paredes[1] , M. Roethlin[3] , H. Vogt[1]

[1] Swiss National Supercomputing Center, CSCS
[2] Center for Climate Systems Modeling, ETH Zurich
[3] Federal Office of Meteorology and Climatology, MeteoSwiss

ETH zürich

XCLAIM

Platform for Advanced Scientific Computing

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre