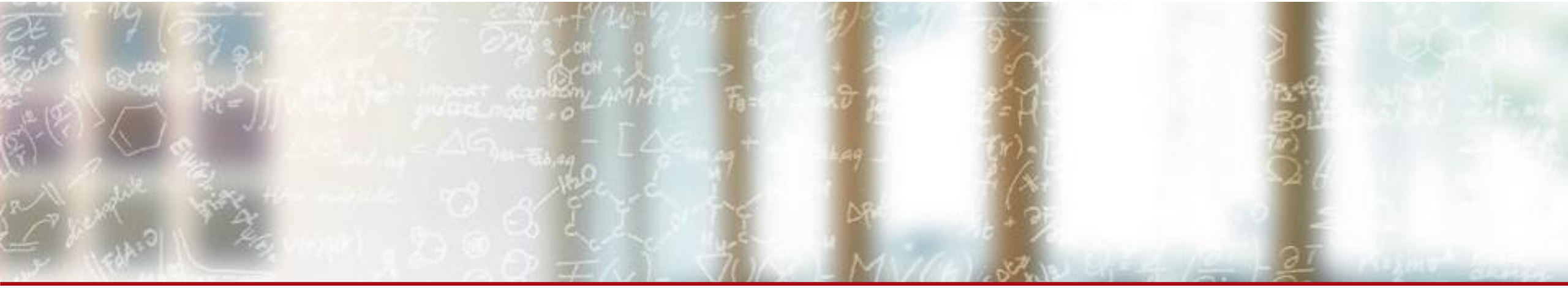
CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

ETH zürich



GridTools High-level HPC Libraries for Weather and Climate

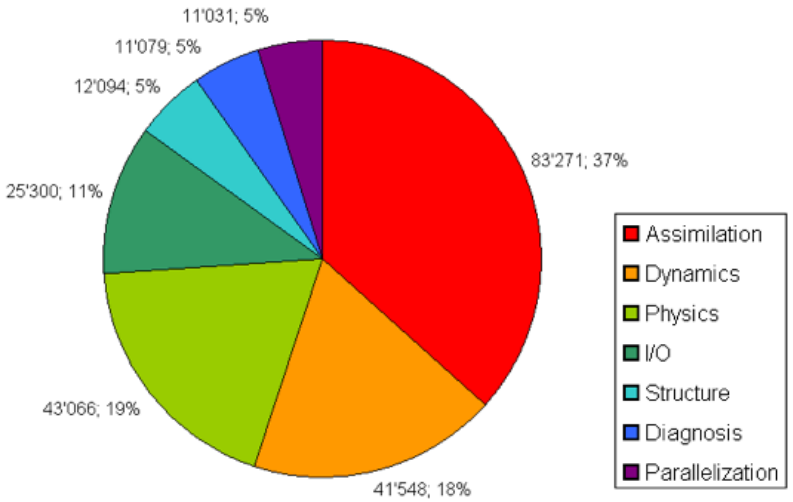
Hannes Vogt, CSCS

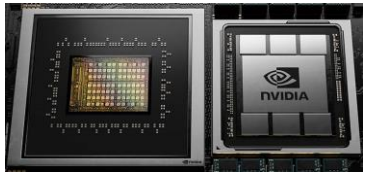
March 18, 2022

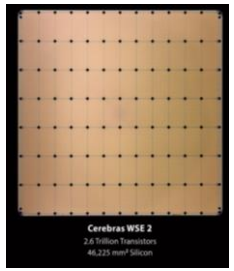
Why?

- Historically monolithic Fortran models were tuned/rewritten for target hardware

- E.g. COSMO:
  About 250’000 lines of Fortran code







Who and What?

- Contributors and Partners

- Weather and Climate models

ICON

FVM for IFS

The GridTools Framework

- Goals:
  - Domain-scientist-friendly languages/APIs for Weather and Climate applications
  - HPC: achieve performance portability

- GridTools C++:
  - Successor of STELLA for COSMO
  - Embedded Domain Specific Language
- GT4Py
  - Cartesian GT4Py
  - Declarative GT4Py
- Other tools

Warning: There will be code!
https://github.com/havogt/gt_framework_talk

Model component development

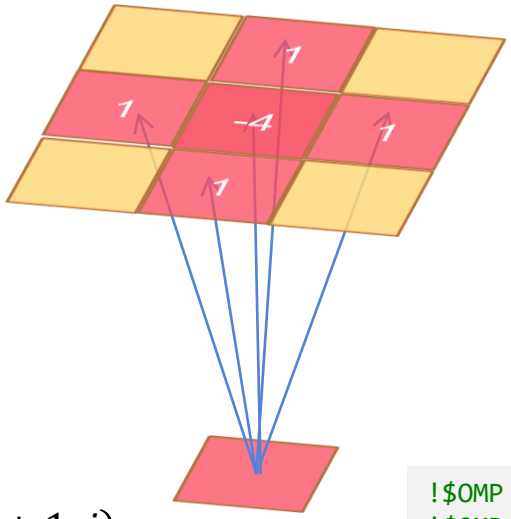
Mathematical Formulation

Discretization

Prototype Implementation

HPC Implementation

$$\Psi = \Delta_{x,y}\phi$$



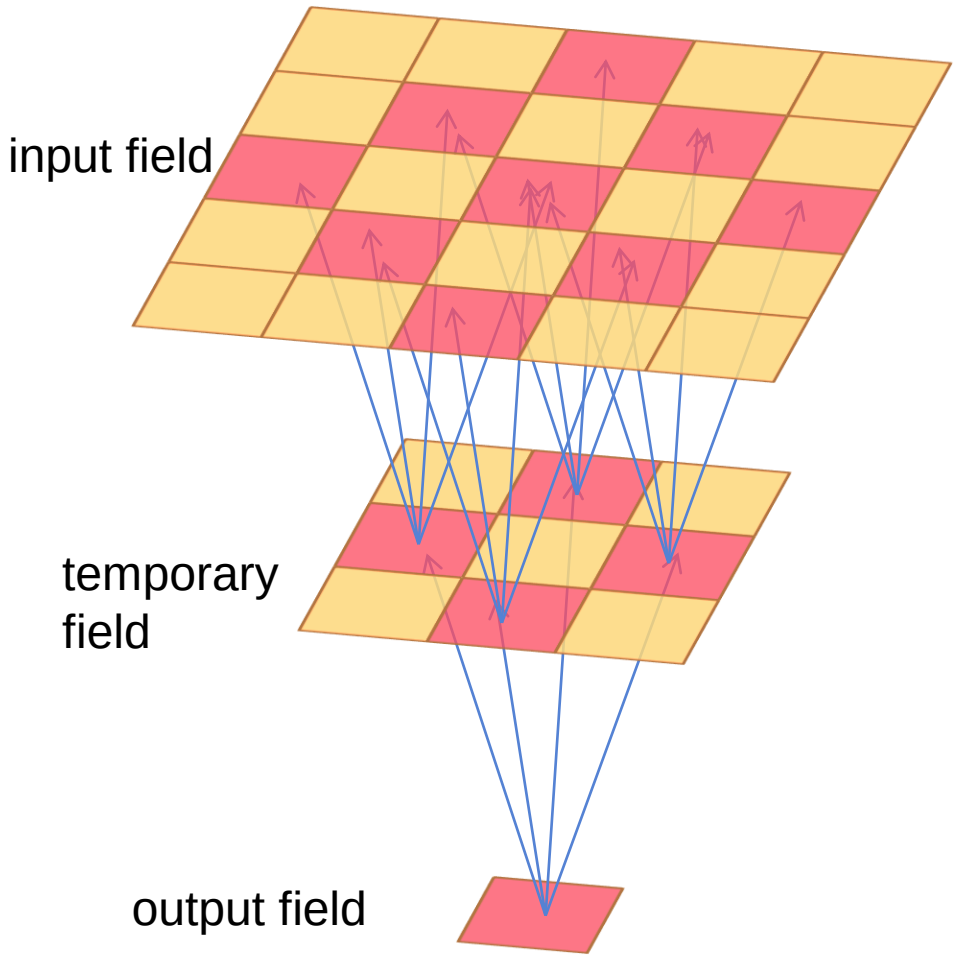
$$\Psi(i,j) = \frac{1}{h^2}\{-4 * \phi(i,j) + \phi(i+1,j) + \phi(i-1,j) + \phi(i,j+1) + \phi(i,j-1)\}$$

```python
import numpy as np

def laplacian(inp: np.ndarray):
    return (
        -4.0 * inp[1:-1, 1:-1]
        + inp[2:, 1:-1] + inp[:-2, 1:-1]
        + inp[1:-1, 2:] + inp[1:-1, :-2]
    )
```

```
!$OMP PARALLEL
!$OMP DO PRIVATE(jb,i_startidx,i_endidx,je,jk)
  DO jb = i_startblk, i_endblk
    ! ...
!$ACC PARALLEL IF( i_am_accel_node .AND. acc_on )
#ifdef __LOOP_EXCHANGE
   !$ACC LOOP GANG
    DO je = i_startidx, i_endidx
     !$ACC LOOP VECTOR
      DO jk = slev, elev
#else
!CDIR UNROLL=3
   !$ACC LOOP GANG
    DO jk = slev, elev
      !$ACC LOOP VECTOR
      DO je = i_startidx, i_endidx
#endif
         ! here goes the laplacian
      END DO
    END DO
!$ACC END PARALLEL
  END DO
!$OMP END DO NOWAIT
!$OMP END PARALLEL
```

Inspired by slides from O. Fuhrer (MeteoSwiss)

Example: Laplacian of a Laplacian




```
template <class In, class Out, class Tmp>
void laplap_naive(std::array<std::size_t, 3> sizes, In const &input,
                  Out &output, Tmp &&tmp) {
  for (std::size_t i = 1; i < sizes[0] - 1; ++i) {
    for (std::size_t j = 1; j < sizes[1] - 1; ++j) {
      for (std::size_t k = 0; k < sizes[2]; ++k) {
        tmp[i][j][k] = -4.0 * input[i][j][k] //
                       + input[i + 1][j][k]  //
                       + input[i - 1][j][k]  //
                       + input[i][j + 1][k]  //
                       + input[i][j - 1][k];
      }
    }
  }
  for (std::size_t i = 2; i < sizes[0] - 2; ++i) {
    for (std::size_t j = 2; j < sizes[1] - 2; ++j) {
      for (std::size_t k = 0; k < sizes[2]; ++k) {
        output[i][j][k] = -4.0 * tmp[i][j][k] //
                          + tmp[i + 1][j][k]  //
                          + tmp[i - 1][j][k]  //
                          + tmp[i][j + 1][k]  //
                          + tmp[i][j - 1][k];
      }
    }
  }
}
```

Example: Laplacian of a Laplacian

```cpp
int main() {
  constexpr std::size_t Nx = 64;
  constexpr std::size_t Ny = 64;
  constexpr std::size_t Nz = 64;

  auto input = new double[Nx][Ny][Nz];
  auto result = new double[Nx][Ny][Nz];
  auto tmp = new double[Nx][Ny][Nz];

  constexpr std::array<std::size_t, 3> sizes{Nx, Ny, Nz};
  laplap_naive(sizes, input, result, tmp);
  // delete
}
```

memory layout

loop structure

explicit loop bounds

```cpp
template <class In, class Out, class Tmp>
void laplap_naive(std::array<std::size_t, 3> sizes, In const &input,
                  Out &output, Tmp &&tmp) {
  for (std::size_t i = 1; i < sizes[0] - 1; ++i) {
    for (std::size_t j = 1; j < sizes[1] - 1; ++j) {
      for (std::size_t k = 0; k < sizes[2]; ++k) {
        tmp[i][j][k] = -4.0 * input[i][j][k] //
                       + input[i + 1][j][k]  //
                       + input[i - 1][j][k]  //
                       + input[i][j + 1][k]  //
                       + input[i][j - 1][k];
      }
    }
  }
  for (std::size_t i = 2; i < sizes[0] - 2; ++i) {
    for (std::size_t j = 2; j < sizes[1] - 2; ++j) {
      for (std::size_t k = 0; k < sizes[2]; ++k) {
        output[i][j][k] = -4.0 * tmp[i][j][k] //
                          + tmp[i + 1][j][k]  //
                          + tmp[i - 1][j][k]  //
                          + tmp[i][j + 1][k]  //
                          + tmp[i][j - 1][k];
      }
    }
  }
}
```

memory access for temporary

Example: Laplacian of a Laplacian

```cpp
int main() {
  constexpr std::size_t Nx = 64;
  constexpr std::size_t Ny = 64;
  constexpr std::size_t Nz = 64;

  auto input = new double[Nx][Ny][Nz];
  auto result = new double[Nx][Ny][Nz];
  auto tmp = new double[Nx][Ny][Nz];

  constexpr std::array<std::size_t, 3> sizes{Nx, Ny, Nz};
  laplap_naive2(sizes, input, result, tmp);
  // delete
}
```

```cpp
using domain_t = std::array<std::array<std::size_t, 2>, 3>;

template <class In, class Out>
void lap(domain_t domain, In const &in, Out &out) {
  for (std::size_t i = domain[0][0]; i < domain[0][1]; ++i) {
    for (std::size_t j = domain[1][0]; j < domain[1][1]; ++j) {
      for (std::size_t k = domain[2][0]; k < domain[2][1]; ++k) {
        out[i][j][k] = -4.0 * in[i][j][k] //
                       + in[i + 1][j][k]  //
                       + in[i - 1][j][k]  //
                       + in[i][j + 1][k]  //
                       + in[i][j - 1][k];
      }
    }
  }
}

template <class In, class Out, class Tmp>
void laplap_naive2(std::array<std::size_t, 3> sizes, In const &input,
                   Out &output, Tmp &&tmp) {
  lap(domain_t{{1, sizes[0] - 1}, {1, sizes[1] - 1}, {0, sizes[2]}}, input,
      tmp);
  lap(domain_t{{2, sizes[0] - 2}, {2, sizes[1] - 2}, {0, sizes[2]}}, tmp,
      output);
}
```

Example: Laplacian of a Laplacian – more abstractions

```cpp
int main() {
  constexpr std::size_t Nx = 64;
  constexpr std::size_t Ny = 64;
  constexpr std::size_t Nz = 64;

  auto input = new double[Nx][Ny][Nz];
  auto result = new double[Nx][Ny][Nz];
  auto tmp = new double[Nx][Ny][Nz];

  constexpr std::array<std::size_t, 3> sizes{Nx, Ny, Nz};
  laplap_naive2(sizes, input, result, tmp);
  // delete
}
```

```cpp
using domain_t = std::array<std::array<std::size_t, 2>, 3>;

template <class Fun, class... Args>
void apply(domain_t domain, Fun &&fun, Args &&...args) {
  for (std::size_t i = domain[0][0]; i < domain[0][1]; ++i) {
    for (std::size_t j = domain[1][0]; j < domain[1][1]; ++j) {
      for (std::size_t k = domain[2][0]; k < domain[2][1]; ++k) {
        fun(std::forward<Args>(args)..., i, j, k);
      }
    }
  }
}

constexpr auto lap3 = [](auto const &in, auto &out, std::size_t i,
                         std::size_t j, std::size_t k) {
  out[i][j][k] = -4.0 * in[i][j][k] //
                 + in[i + 1][j][k]  //
                 + in[i - 1][j][k]  //
                 + in[i][j + 1][k]  //
                 + in[i][j - 1][k];
};

template <class In, class Out, class Tmp>
void laplap_naive3(std::array<std::size_t, 3> sizes, In const &input,
                   Out &output, Tmp &&tmp) {
  apply(domain_t{{{1, sizes[0] - 1}, {1, sizes[1] - 1}, {0, sizes[2]}}}, lap3,
        input, tmp);
  apply(domain_t{{{2, sizes[0] - 2}, {2, sizes[1] - 2}, {0, sizes[2]}}}, lap3,
        tmp, output);
}
```

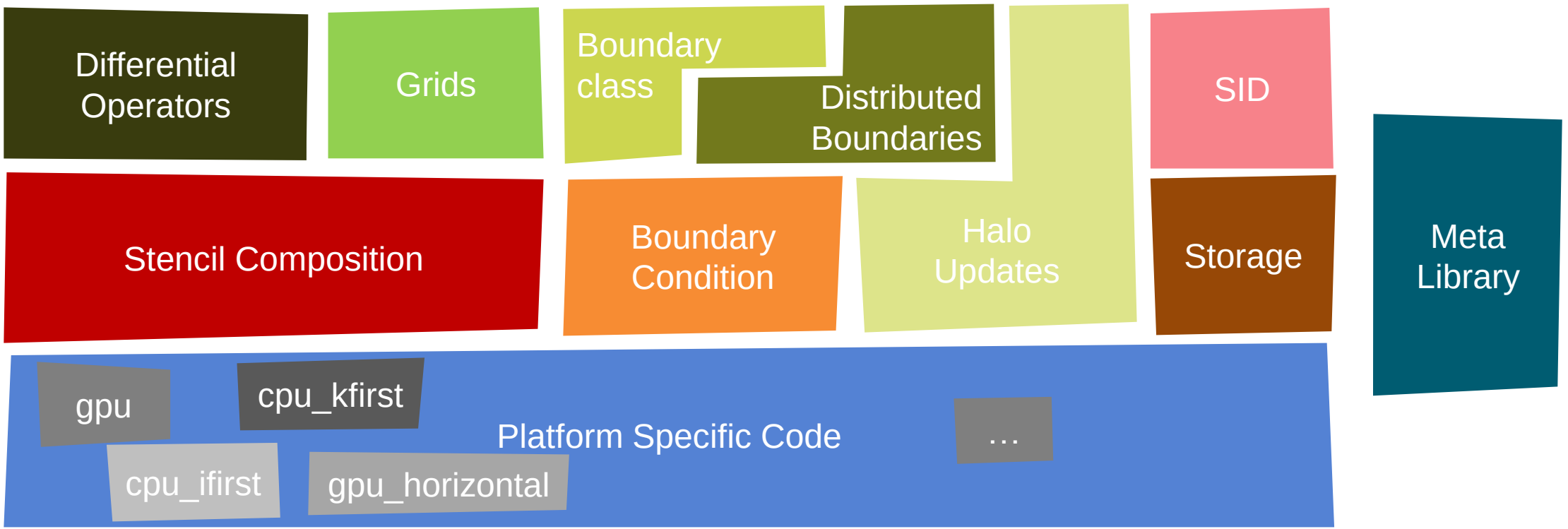
GridTools C++ Example

```
int main() {
  constexpr std::size_t Nx = 64;
  constexpr std::size_t Ny = 64;
  constexpr std::size_t Nz = 20;

  auto builder =
      storage::builder<storage::gpu>.type<double>().dimensions(Nx, Ny,
                                                               Nz);

  auto input = builder.build();
  auto result = builder.build();

  halo_descriptor di{2, 2, 2, Nx - 2 - 1, Nx};
  halo_descriptor dj{2, 2, 2, Nx - 2 - 1, Nx};
  auto grid = make_grid(di, dj, Nz);

  laplap_gridtools(grid, input, result);
}
```

```
using namespace gridtools;
using namespace stencil;

struct lap {
  using in = cartesian::in_accessor<0, extent<-1, 1, -1, 1>>;
  using out = cartesian::inout_accessor<1>;

  using param_list = make_param_list<in, out>;

  template <class Eval> GT_FUNCTION static void apply(Eval &&eval) {
    eval(out()) = -4.0 * eval(in()) + eval(in(1, 0)) + eval(in(-1, 0)) +
                  eval(in(0, 1)) + eval(in(0, -1));
  }
};
```

```
template <class Grid, class In, class Out>
void laplap_gridtools(Grid const &grid, In &&in, Out &&out) {
  auto spec = [](auto in, auto out) {
    GT_DECLARE_TMP(double, tmp);
    return execute_parallel()
        .ij_cached(tmp)
        .stage(lap{}, in, tmp)
        .stage(lap{}, tmp, out);
  };

  run(spec, stencil::gpu{}, grid, in, out);
}
```

GridTools C++

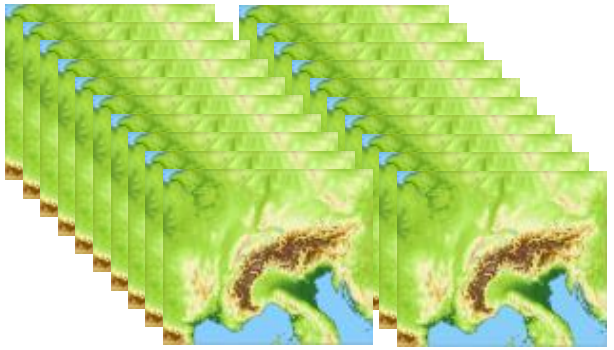


- Status: stable, https://github.com/GridTools/gridtools/, BSD-3 license
  - First release in April 2019
  - Current release: v2.1.0

Application: COSMO

- Dynamical core based on GridTools, since COSMO v5.7
- provides GPU support
- Operational at MeteoSwiss since 2019

- Testcase with COSMO-E on Piz Kesch using 4 K80s (8 GK210)
- 582 x 390 x 60 gridpoints
- Total runtime for 360 timesteps (2 hours simulated time)



COSMO-E *(in production at MeteoSwiss from 2016 - 2019)*
5 days forecasts, 2x per day, 2.2km grid size, 21 members

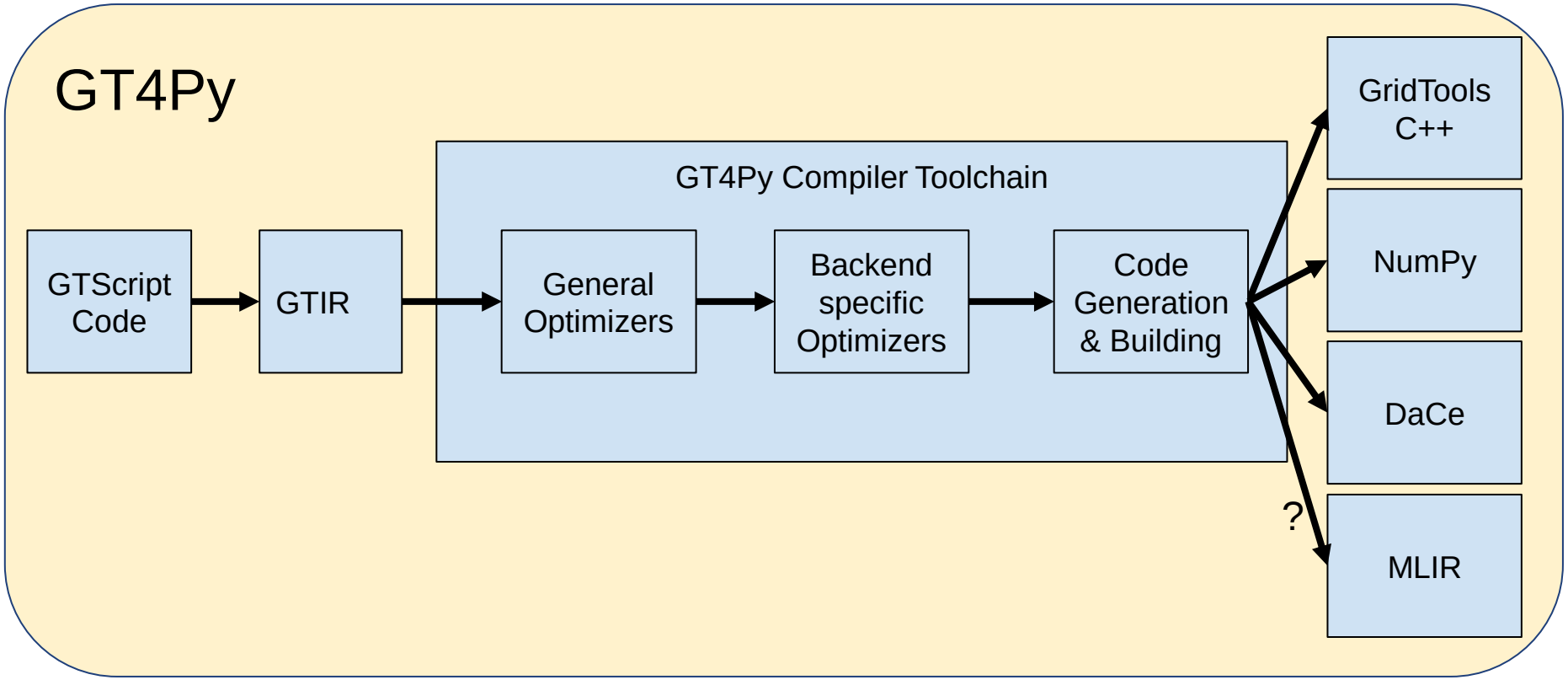
Value Type	Implementation	COSMO Total Time	Dynamical Core
Float	STELLA	141	66.2
	GridTools	**131** (-7 %)	**53.6** (-19 %)
Double	STELLA	204	101.3
	GridTools	**192** (-6 %)	**86.1** (-15 %)

GridTools C++ Pros and Cons

- Positive
  - Achieved performance portability:
    - Single source code
    - Single place to switch architecture
  - Embedded in C++: all you need is a C++ compiler
    - Escaping the DSL is trivial
  - **Weather model in operations at MeteoSwiss since 2019**
- Negative
  - Boiler-plate from embedding in C++
  - Compile-time, C++ template error messages
  - Some optimizations are not possible
    - e.g. #pragmas on specific variables
    - Transformation between compute-on-the-fly and staged execution
  - **We did not manage to convince the domain scientist to use this language**

Cartesian GT4Py

- Originally: a Python frontend for GridTools C++
- Now a compiler-like toolchain



schematic overview

Cartesian GT4Py Example

- Live demo

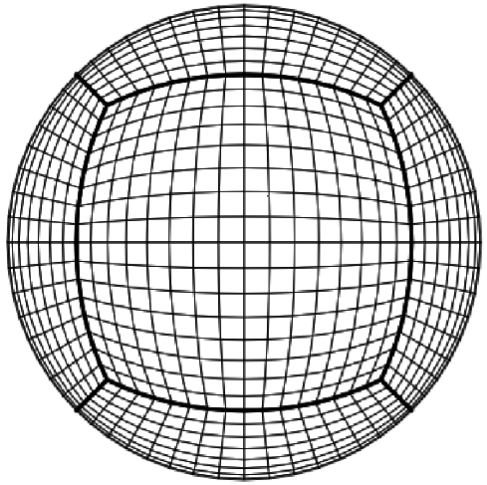
Cartesian GT4Py Pros and Cons

- Status
  - https://github.com/GridTools/gt4py, beta, release planned for H2-2022
- Positive
  - Improved productivity compared to GridTools C++
    - More concise, easier to read syntax
    - Better error messages
- Negative
  - Debuggability of the DSL:
    - Debugger steps through generated code, not the GTScript source
  - Composability is limited to gtscript.functions, not possible across vertical loops
    - For optimizability: stencil should be as big as possible
    - For testing: stencil should be reasonably small
  - GTScript language with mutable fields requires complicated restrictions for parallelization

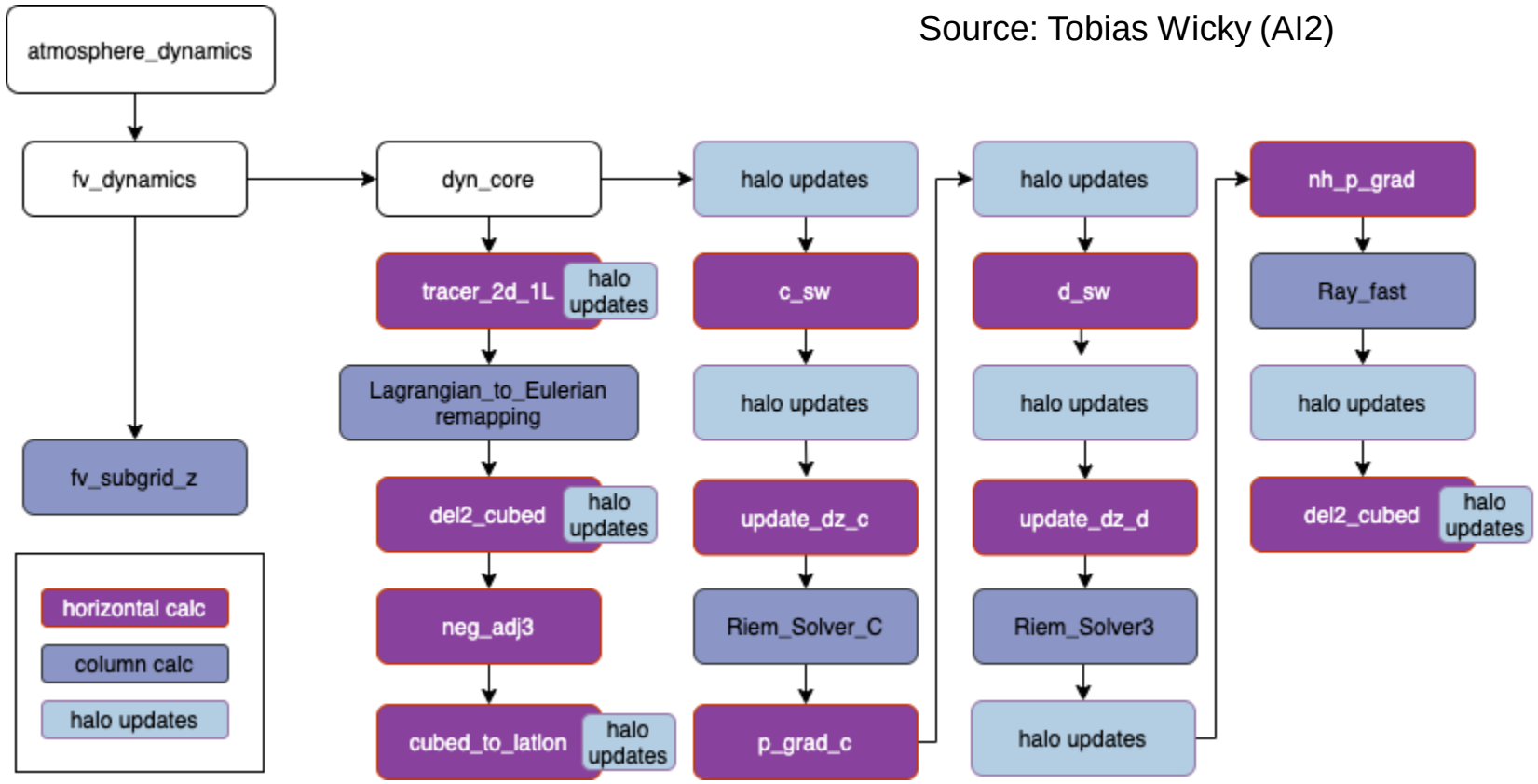
```
def shifted_self_assignment(field):
    with computation(PARALLEL), interval(...):
        field = field[1,0,0]
```

Excursion: FV3 with GT4Py and DaCe

- 30k lines of Python code
- 13 modules
- 180 stencils



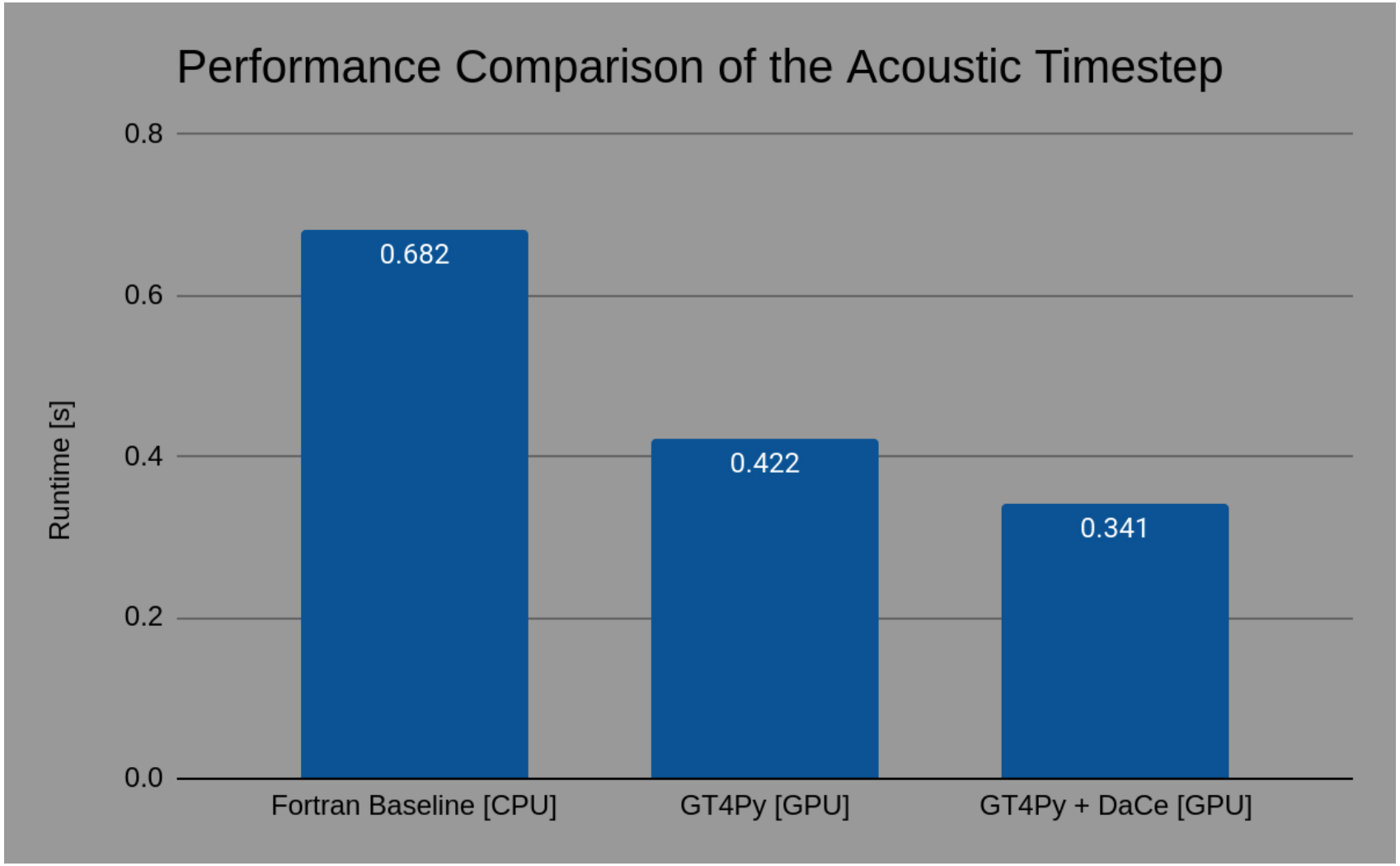
Source: Tobias Wicky (AI2)



See https://www.youtube.com/watch?v=2FA6Eu7msXU for details

Excursion: FV3 with GT4Py and DaCe

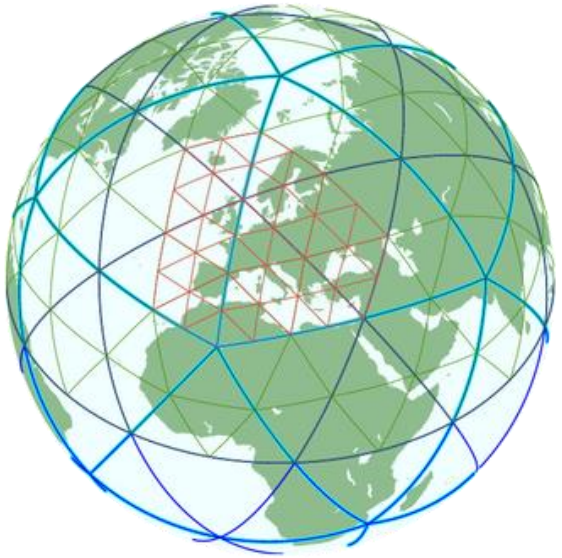
preliminary results



Source:
Tobias Wicky (AI2)

Declarative GT4Py

- Functional stencil language
  - Overcomes composability issue
  - Removes weird language limitations
- With a thin interface layer with mutable field
  - Allows embedding into existing applications



- Support for non-Cartesian/unstructured meshes
- Embedded execution
  - Improved debuggability

Declarative GT4Py: Example

under active development

```python
@field_operator
def lap(in_field) :
    return (
        -4.0 * in_field + in_field(I + 1) + in_field(J + 1)
        + in_field(I - 1) + in_field(J - 1)
    )

@field_operator
def laplap(in_field):
    return lap(lap(in_field))

@program
def laplap_program(in_field, out_field):
    laplap(in_field, out=out_field[2:-2, 2:-2])
```

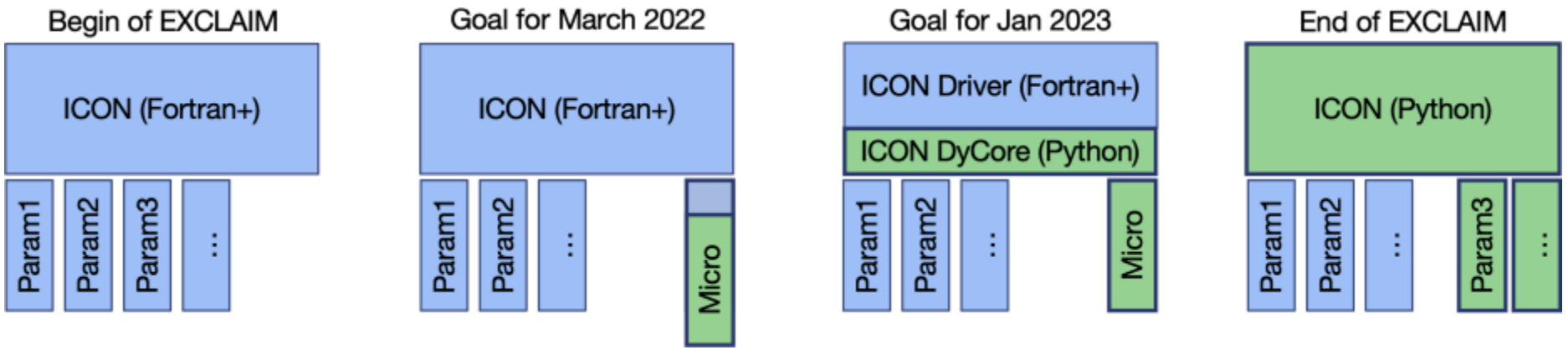
```python
def main():
    shape = (20, 20)
    as_ij = np_as_located_field(IDim, JDim)
    input = as_ij(np.fromfunction(lambda x, y: x**5 + y**5, shape))

    result = as_ij(np.zeros_like(input))
    laplap_program(input, result, offset_provider={"I": IDim, "J": JDim})
```

- Status
  - https://github.com/GridTools/gt4py, branch “functional”, early alpha eventually planned to be released as GT4Py v2

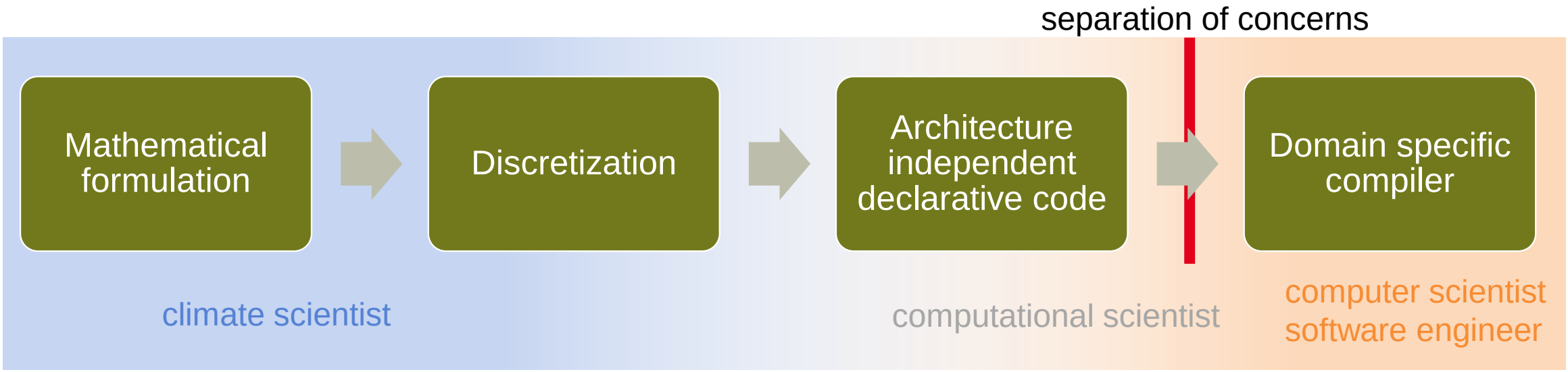
Outlook: EXCLAIM

https://exclaim.ethz.ch

- Project funded by ETH Zurich to provide a platform for climate science
- Scientific goal: high-resolution global simulations at 3 km resolution
- Full model driven by Python based on Declarative GT4Py

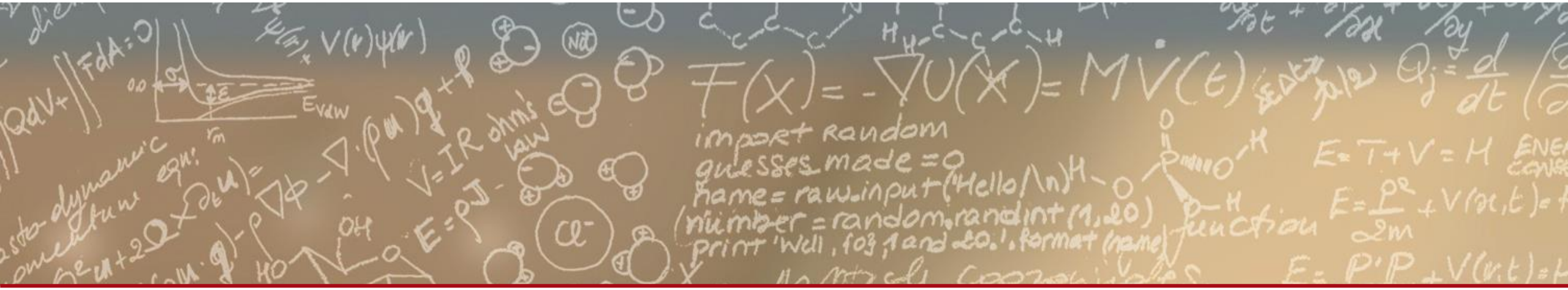


Conclusion: Model component development - our vision



Inspired by slides from O. Fuhrer (MeteoSwiss)

- Challenges
  - Trade-off between convenience (maybe habbit) and expressive semantics
  - Generality vs. domain specificness



Thank you for your attention.