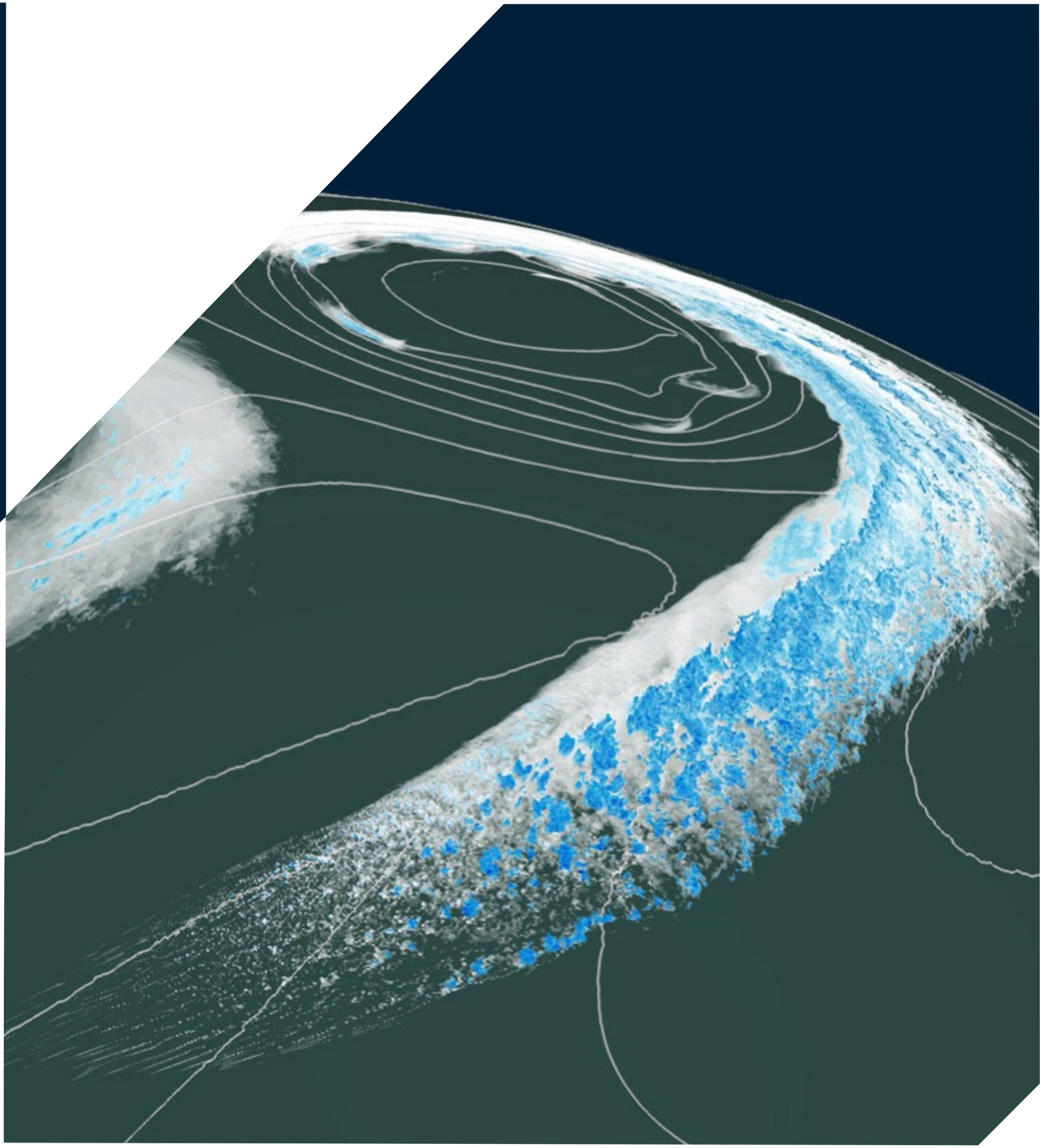
Learning how to forget

How high-level abstractions can help bridge the gap between productivity and performance.

Oliver Fuhrer, DOE ESMD/E3SM PI Meeting, 10/28/20

VULCAN



Who is Vulcan?

Vulcan was founded by Paul and Jody Allen, and is their estate's company and engine for philanthropic efforts.

Multiple tech4good projects focusing on oceans, climate, conservation, and communities.

Vulcan Climate Modeling (VCM)

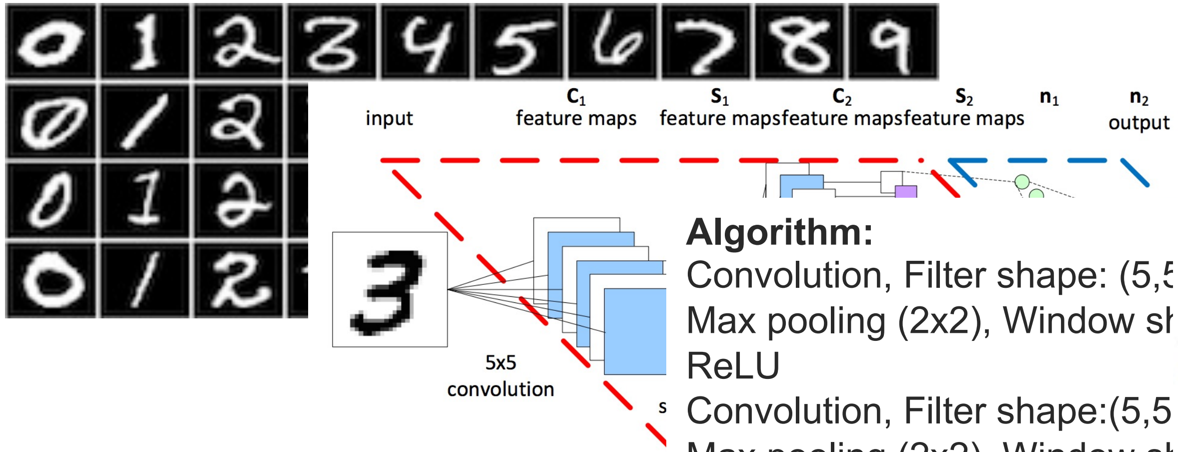
Pilot started June 2019, co-led by Chris Bretherton and Oliver Fuhrer

Main partners: NOAA GFDL in Princeton, CSCS and MeteoSwiss in Switzerland

The Semantic Gap

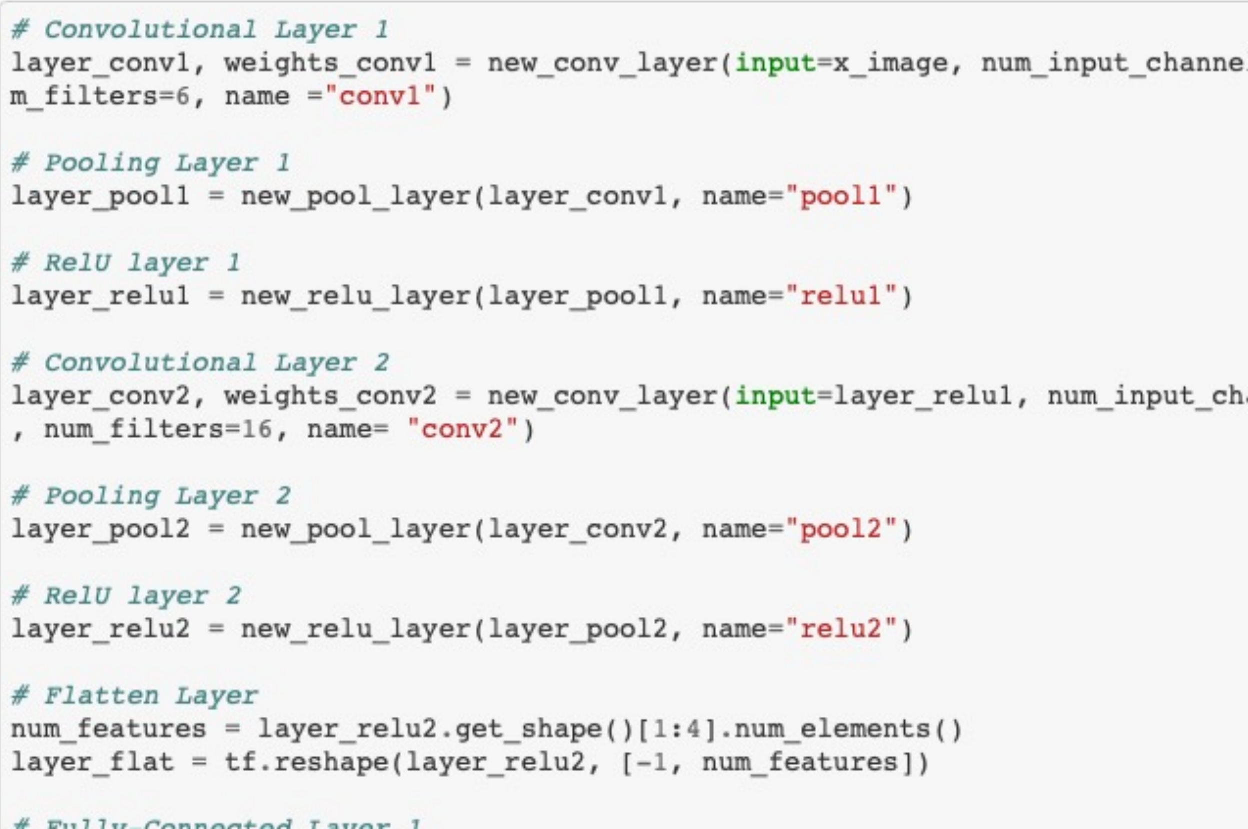
THE SEMANTIC GAP
APPLICATION DOMAIN
EXECUTION DOMAIN

Machine Learning Scientist



Algorithm:
Convolution, Filter shape: (5,5
Max pooling (2x2), Window sh
ReLU
Convolution, Filter shape:(5,5
Max pooling (2x2), Window sh
ReLU
Fully Connected Layer (128)
ReLU
Fully Connected Layer (10)
Softmax

Implementation (Python + Tensorflow):

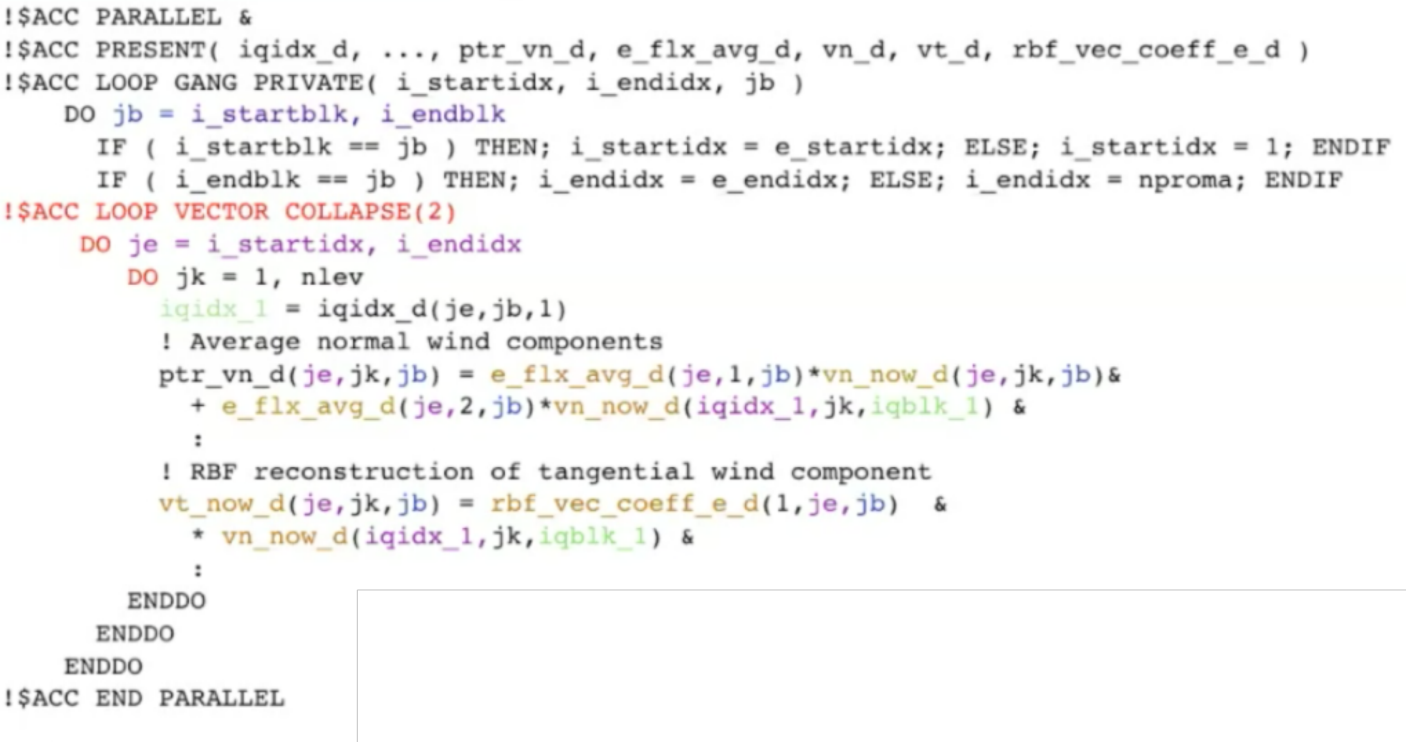


```
# Convolutional Layer 1
layer_conv1, weights_conv1 = new_conv_layer(input=x_image, num_input_channe
m_filters=6, name ="conv1")

# Pooling Layer 1
layer_pool1 = new_pool_layer(layer_conv1, name="pool1")

# RelU layer 1
layer_relu1 = new_relu_layer(layer_pool1, name="relu1")

# Convolutional Layer 2
layer_conv2, weights_conv2 = new_conv_layer(input=layer_relu1, num_input_ch
, num_filters=16, name= "conv2")

# Pooling Layer 2
layer_pool2 = new_pool_layer(layer_conv2, name="pool2")

# RelU layer 2
layer_relu2 = new_relu_layer(layer_pool2, name="relu2")

# Flatten Layer
num_features = layer_relu2.get_shape()[1:4].num_elements()
layer_flat = tf.reshape(layer_relu2, [-1, num_features])

# Fully-Connected Layer 1
```

Climate Scientist

$$\vec{u}(\vec{x}) = \sum_{i=1}^{k} \lambda_i \phi_i(\vec{x}) \vec{n}_i$$

Implementation (Fortran + OpenACC):



```
!$ACC PARALLEL &
!$ACC PRESENT( iqidx_d, ..., ptr_vn_d, e_flx_avg_d, vn_d, vt_d, rbf_vec_coeff_e_d )
!$ACC LOOP GANG PRIVATE( i_startidx, i_endidx, jb )
    DO jb = i_startblk, i_endblk
      IF ( i_startblk == jb ) THEN; i_startidx = e_startidx; ELSE; i_startidx = 1; ENDIF
      IF ( i_endblk == jb ) THEN; i_endidx = e_endidx; ELSE; i_endidx = nproma; ENDIF
!$ACC LOOP VECTOR COLLAPSE(2)
     DO je = i_startidx, i_endidx
        DO jk = 1, nlev
          iqidx_1 = iqidx_d(je,jb,1)
          ! Average normal wind components
          ptr_vn_d(je,jk,jb) = e_flx_avg_d(je,1,jb)*vn_now_d(je,jk,jb)&
            + e_flx_avg_d(je,2,jb)*vn_now_d(iqidx_1,jk,iqblk_1) &
            :
          ! RBF reconstruction of tangential wind component
          vt_now_d(je,jk,jb) = rbf_vec_coeff_e_d(1,je,jb)  &
            * vn_now_d(iqidx_1,jk,iqblk_1) &
            :
        ENDDO
      ENDDO
    ENDDO
!$ACC END PARALLEL
```

Summary

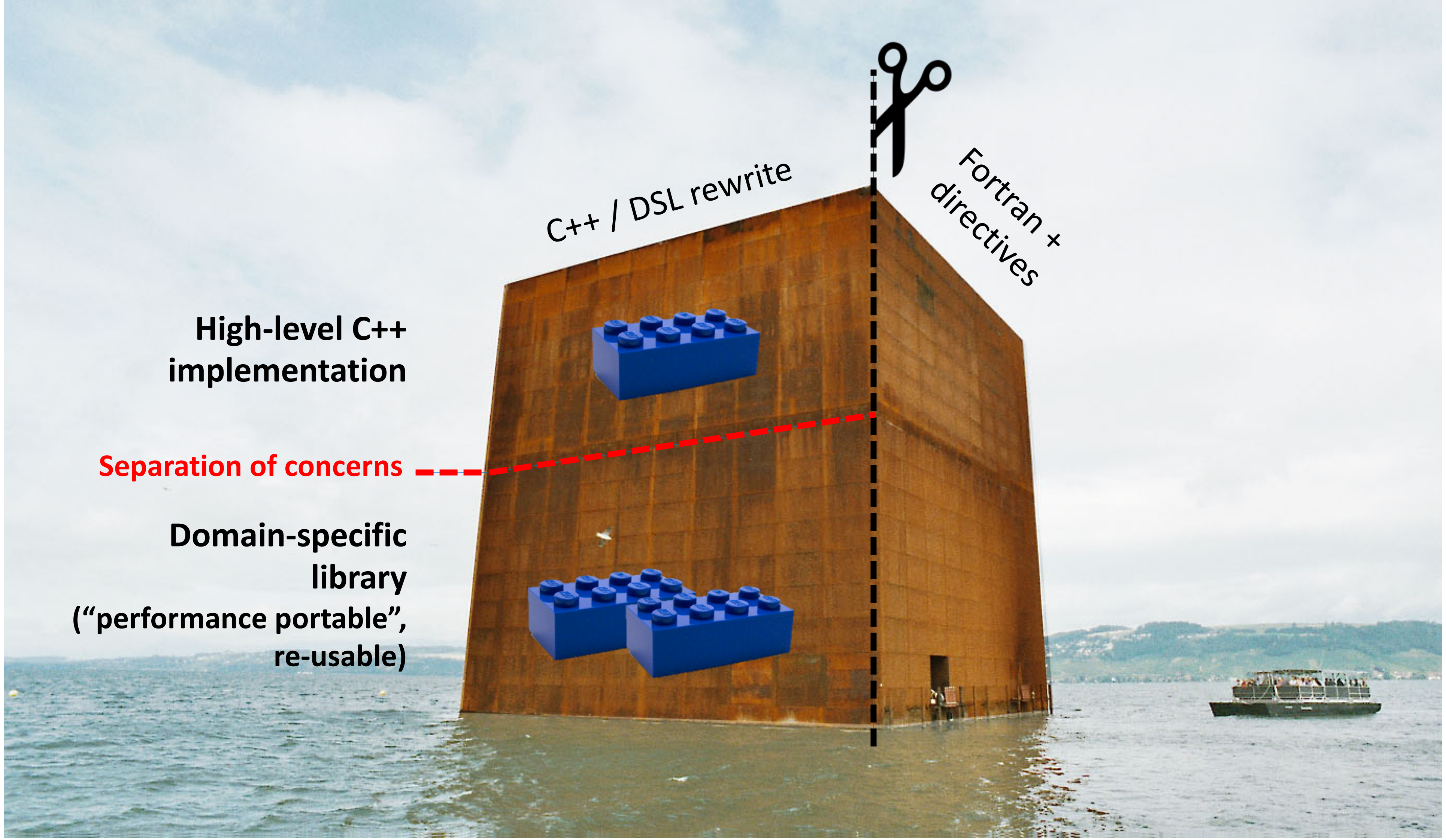
1. Domain-specific languages have the potential for achieving a good balance between performance, portability and productivity (at the price of generality).

2. Optimizing for data-movement on different hardware targets can require a higher-level of abstraction in our codes.

3. No turn-key solutions, but we can build on existing tools and libraries.

(Fuhrer et al. 2014; Gysi et al. 2016)

Experience with COSMO (since 2010)


C++ / DSL rewrite
Fortran + directives
High-level C++ implementation
Separation of concerns
Domain-specific library ("performance portable", re-usable)

Near-global Simulations

(Fuhrer et al. 2018; Schulthess et al. 2019)

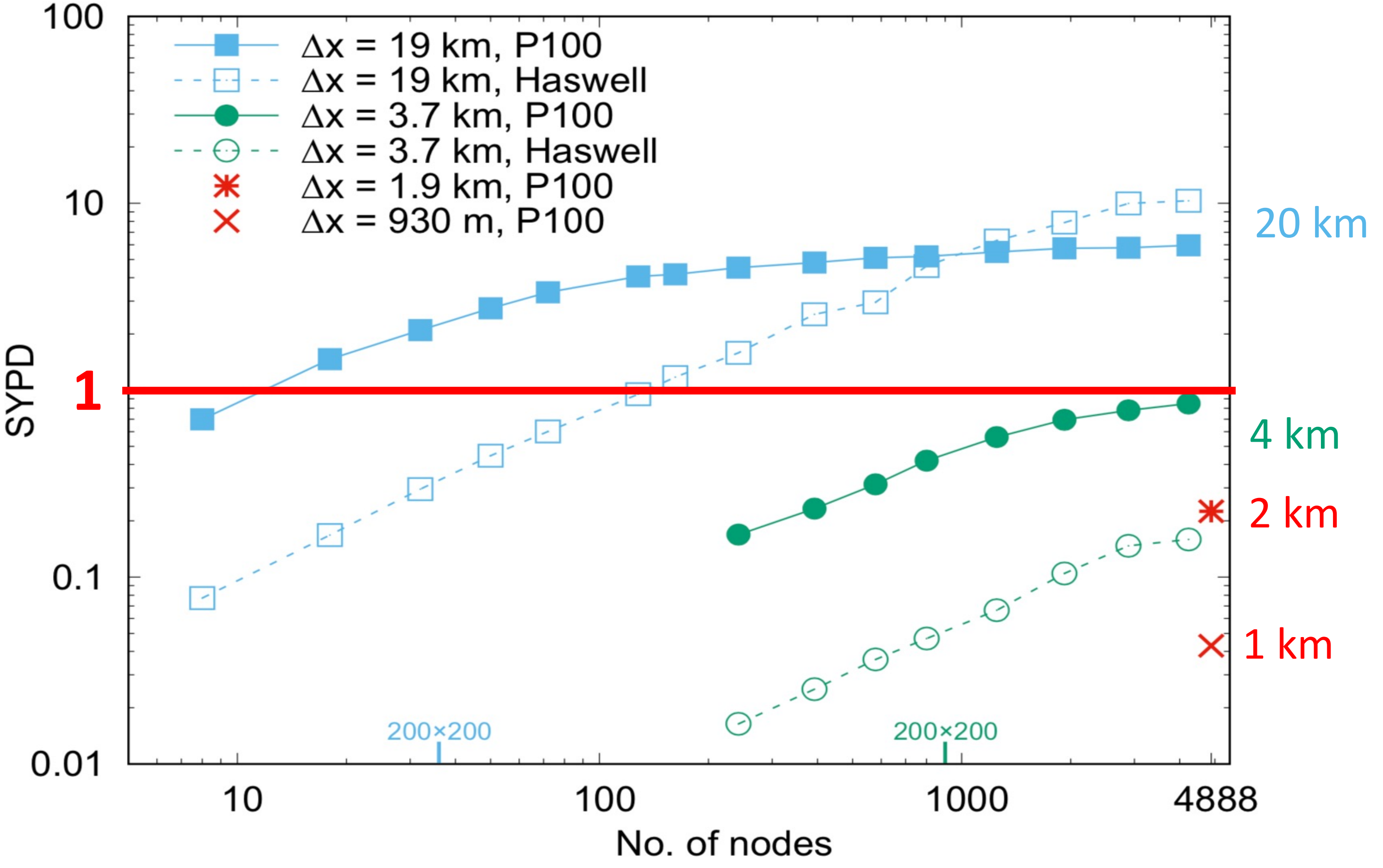
Feasible to port & run an uncoupled climate model on a modern, GPU-accelerated supercomputer.

In the linear scaling regime, GPU simulation is **3x faster** and **7x more energy efficient.**

GPUs need at least 128 x 128 x 80 to perform well. CPUs scale better.

Fast enough (0.25 SYPD) for AMIP-type simulations at 2 km grid spacing.

Strong Scalability (20 km, 4 km, 2 km, 1 km)

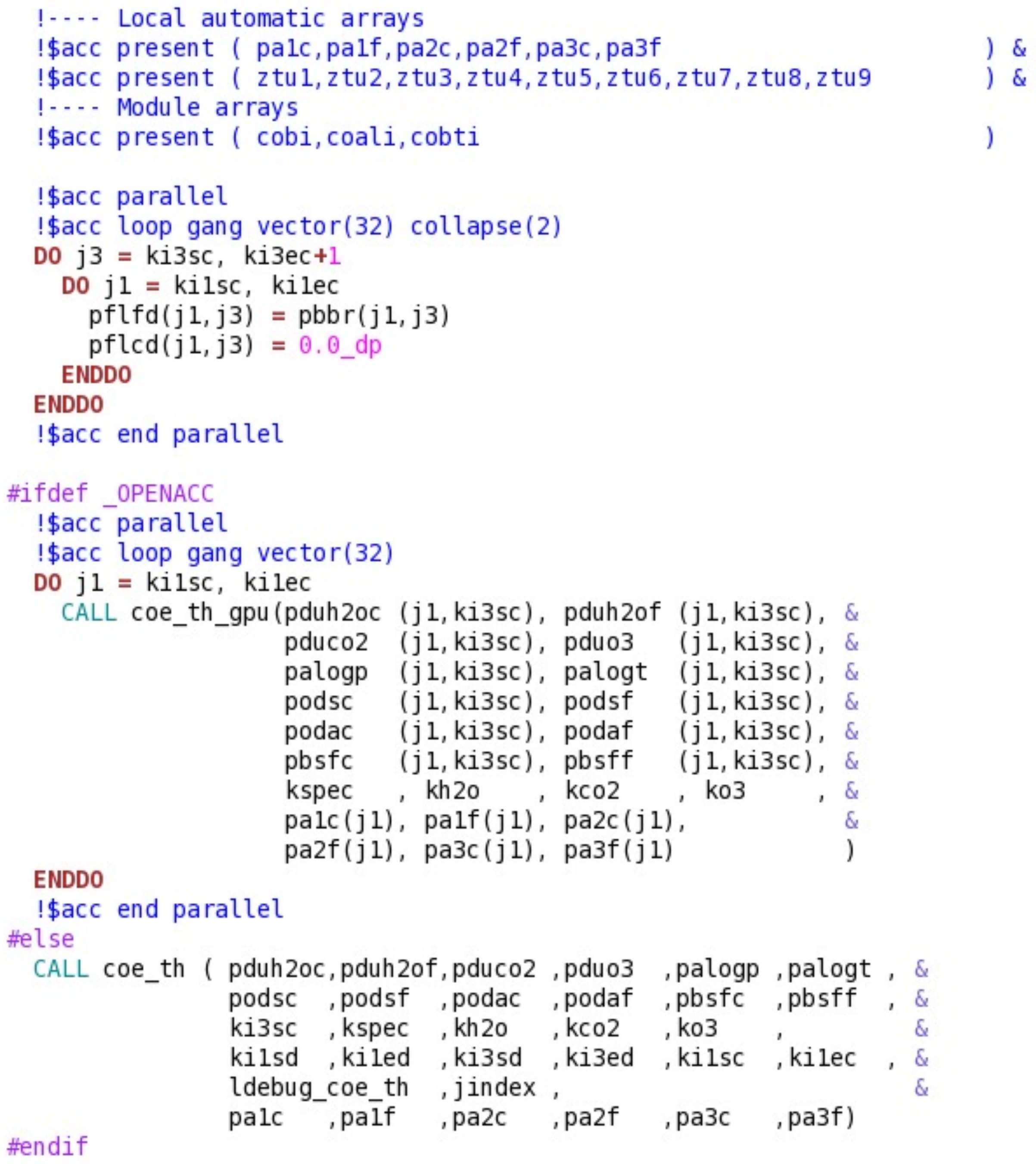


Fortran + X is worse!

Directives are not comments, they are code! (But some developers might ignore them.)

We're adding hardware specific code to an already large code base!

We cannot achieve performance portability by adding more detail.



```
  !---- Local automatic arrays
  !$acc present ( pa1c,pa1f,pa2c,pa2f,pa3c,pa3f                      ) &
  !$acc present ( ztu1,ztu2,ztu3,ztu4,ztu5,ztu6,ztu7,ztu8,ztu9        ) &
  !---- Module arrays
  !$acc present ( cobi,coali,cobti                                    )

  !$acc parallel
  !$acc loop gang vector(32) collapse(2)
  DO j3 = ki3sc, ki3ec+1
    DO j1 = ki1sc, ki1ec
      pflfd(j1,j3) = pbbr(j1,j3)
      pflcd(j1,j3) = 0.0_dp
    ENDDO
  ENDDO
  !$acc end parallel

#ifdef _OPENACC
  !$acc parallel
  !$acc loop gang vector(32)
  DO j1 = ki1sc, ki1ec
    CALL coe_th_gpu(pduh2oc (j1,ki3sc), pduh2of (j1,ki3sc), &
                    pduco2  (j1,ki3sc), pduo3   (j1,ki3sc), &
                    palogp  (j1,ki3sc), palogt  (j1,ki3sc), &
                    podsc   (j1,ki3sc), podsf   (j1,ki3sc), &
                    podac   (j1,ki3sc), podaf   (j1,ki3sc), &
                    pbsfc   (j1,ki3sc), pbsff   (j1,ki3sc), &
                    kspec   , kh2o    , kco2    , ko3     , &
                    pa1c(j1), pa1f(j1), pa2c(j1),           &
                    pa2f(j1), pa3c(j1), pa3f(j1)            )
  ENDDO
  !$acc end parallel
#else
  CALL coe_th ( pduh2oc,pduh2of,pduco2 ,pduo3  ,palogp ,palogt , &
                podsc  ,podsf  ,podac  ,podaf  ,pbsfc  ,pbsff  , &
                ki3sc  ,kspec  ,kh2o   ,kco2   ,ko3    ,         &
                ki1sd  ,ki1ed  ,ki3sd  ,ki3ed  ,ki1sc  ,ki1ec  , &
                ldebug_coe_th  ,jindex ,                         &
                pa1c   ,pa1f   ,pa2c   ,pa2f   ,pa3c   ,pa3f)
#endif
```

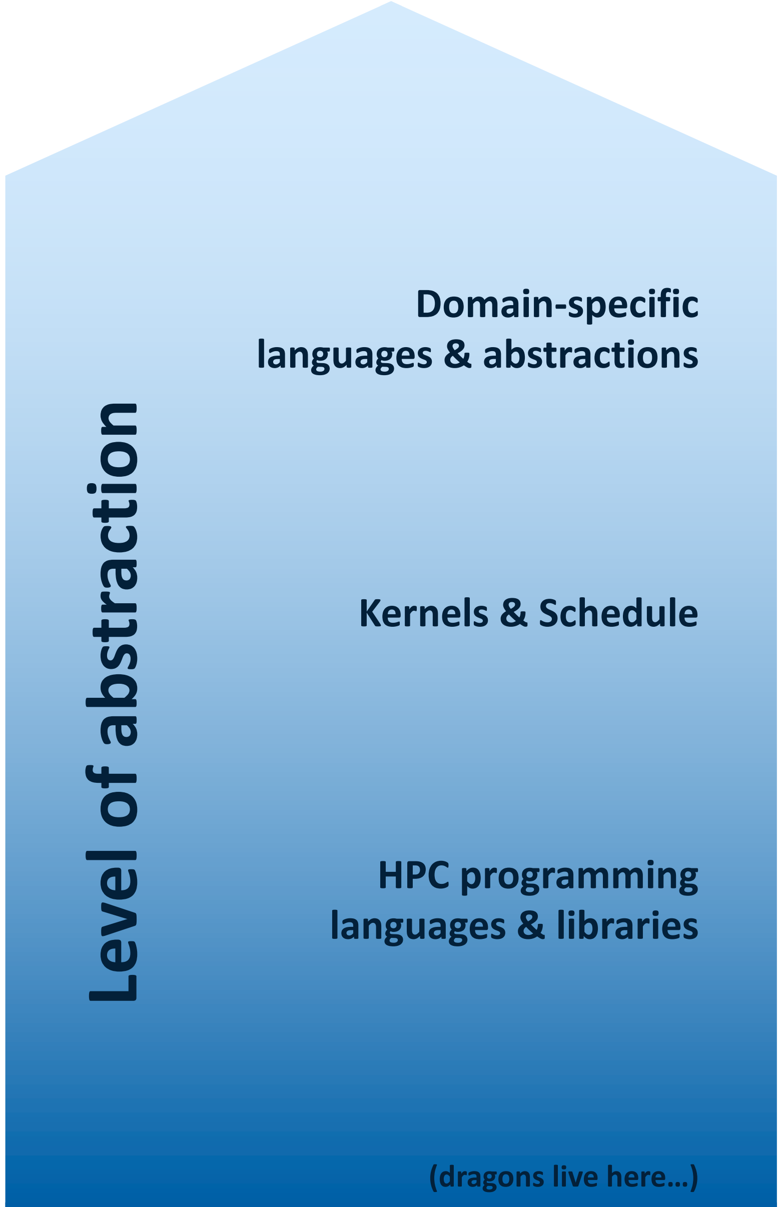
Templated C++ is a mixed bag

- Increasing the level of abstraction improved the performance as well as the maintainability of the code base.
- Investments into software can pay off more than investments into hardware.
- A higher-level of abstraction opens up entirely new avenues to inspect, analyze and instrument the code.

- Using a C++ library (e.g. GridTools C++) which abstracts kernels and schedule does not help with aggressive data-flow optimizations.
- Heaviliy templated C++ is a programming model that is not readily adopted by domain scientists.
- Other gotchas (e.g. boilerplate, debugging)

See also Schaer et al. 2019 MWR, Clement et al. 2019 SuperFri, Thaler et al. 2019, Schulthess et al. 2019, Clement et al. 2018, Fuhrer et al. 2018, Lapillonne et al. 2017, Leutwyler et al. 2016, Gysi et al. 2015, Cumming et al. 2014, Fuhrer et al. 2014, Lapillonne et al. 2014

Taxonomy of Abstractions



Level of abstraction
Domain-specific languages & abstractions
Kernels & Schedule
HPC programming languages & libraries
(dragons live here...)

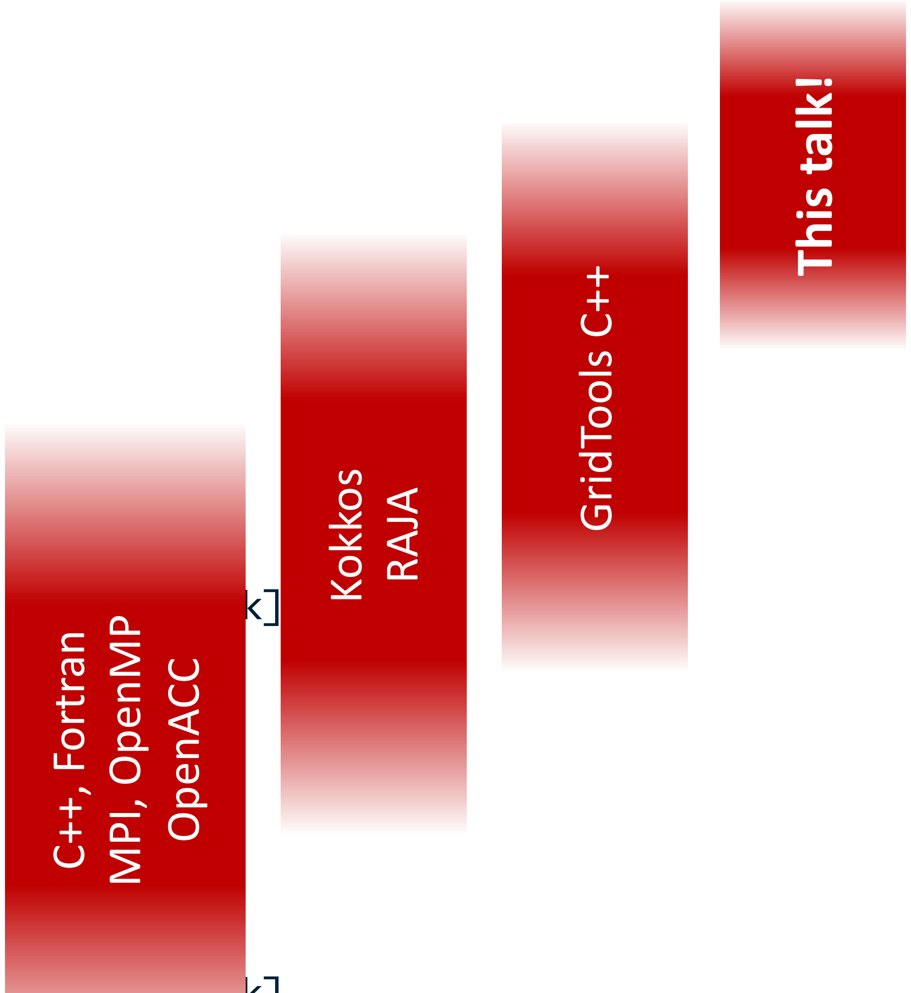
$$\nabla^2 u$$

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{\Delta x^2}$$

```
on domain:
  u[i+1] – 2 u[i] + u[i-1]
```

```
for i = 1, ni
  for j = 1, nj
    for k = 1, nk
      u[i+1,j,k] – 2 u[i,j,k]          k]
```

```
!$acc parallel present(u)
!$acc loop vector collapse(3)
for i = 1, ni
  for j = 1, nj
    for k = 1, nk
      u[i+1,j,k] – 2 u[i,j,k]          k]
```



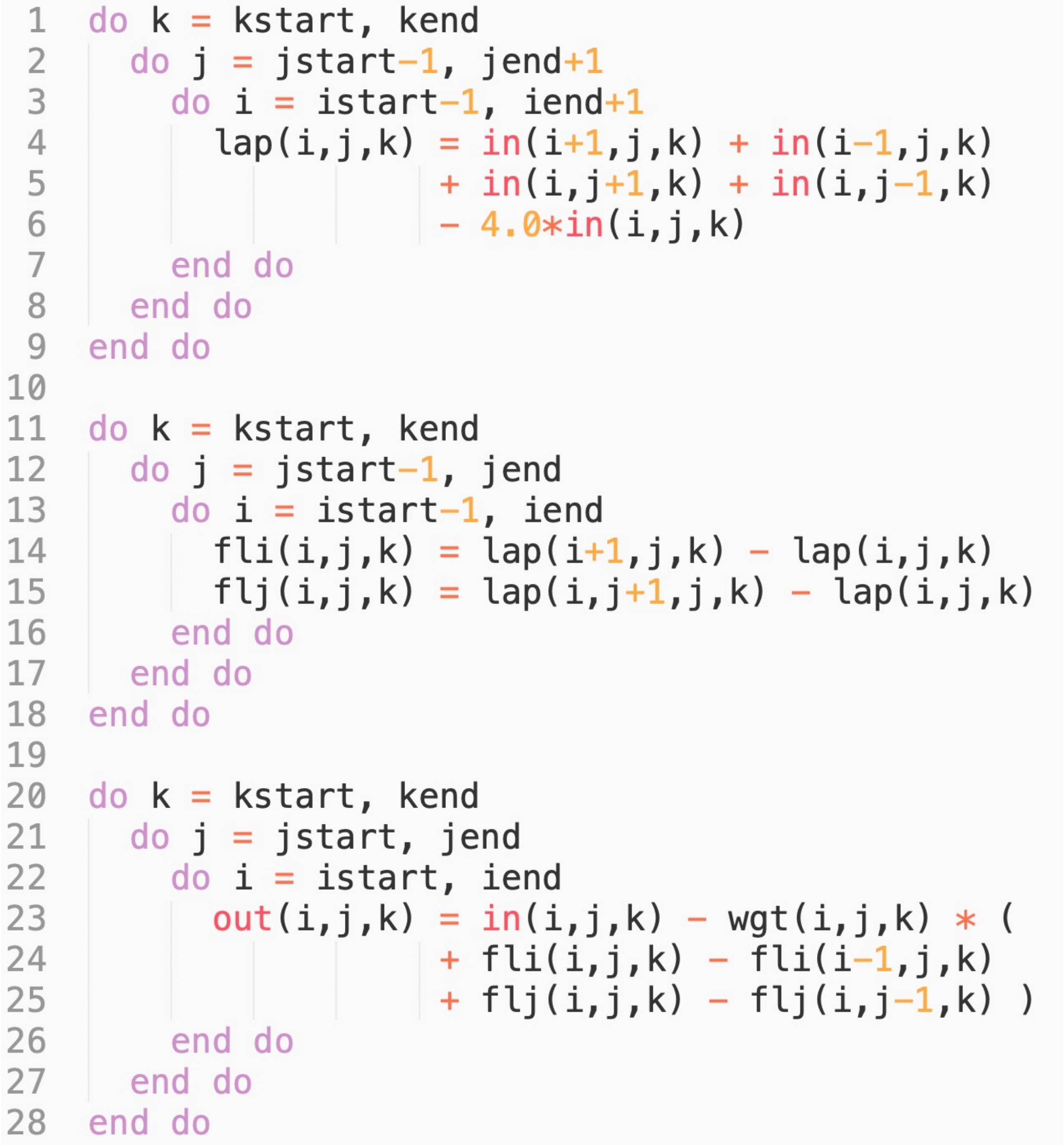
Horizontal Diffusion

$$L = \Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2}$$

$$F_x = \frac{\partial L}{\partial x}$$

$$F_y = \frac{\partial L}{\partial y}$$

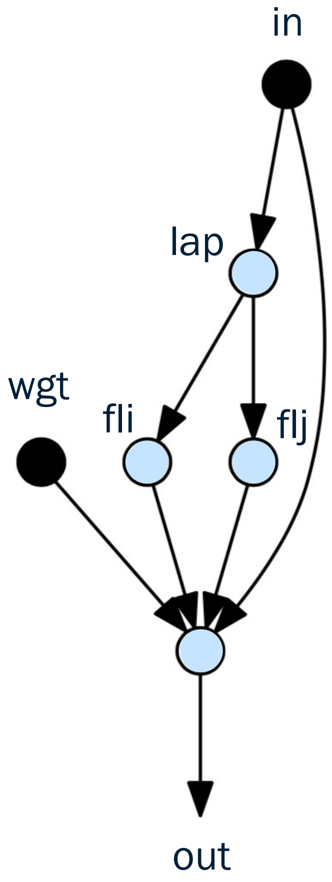
$$\begin{aligned}\frac{\partial\phi}{\partial t} &= -\alpha\nabla^4\phi \\ &= -\alpha\left(\frac{\partial F_x}{\partial x} + \frac{\partial F_y}{\partial y}\right)\end{aligned}$$



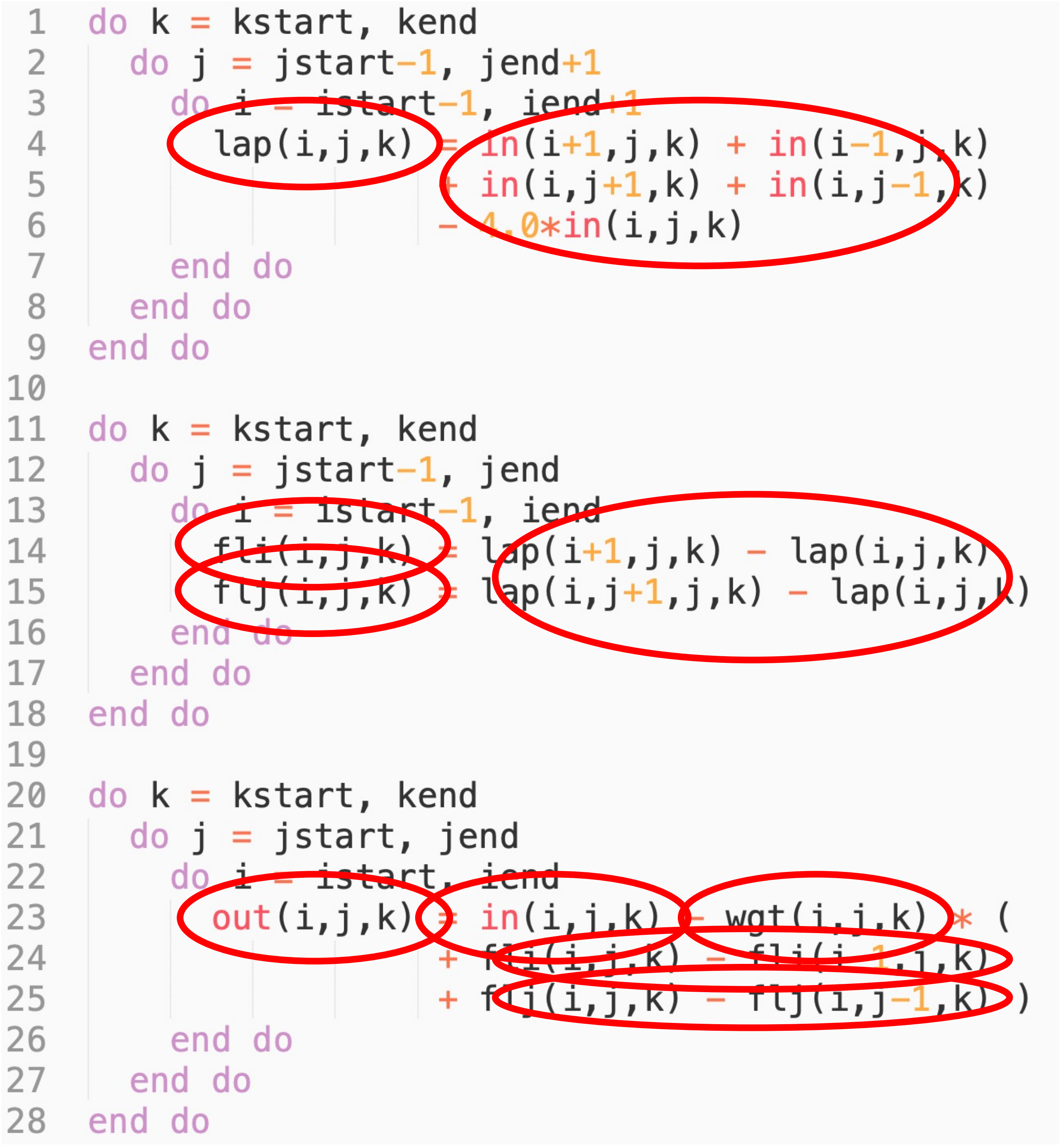
```fortran
do k = kstart, kend
  do j = jstart-1, jend+1
    do i = istart-1, iend+1
      lap(i,j,k) = in(i+1,j,k) + in(i-1,j,k)
                 + in(i,j+1,k) + in(i,j-1,k)
                 - 4.0*in(i,j,k)
    end do
  end do
end do

do k = kstart, kend
  do j = jstart-1, jend
    do i = istart-1, iend
      fli(i,j,k) = lap(i+1,j,k) - lap(i,j,k)
      flj(i,j,k) = lap(i,j+1,j,k) - lap(i,j,k)
    end do
  end do
end do

do k = kstart, kend
  do j = jstart, jend
    do i = istart, iend
      out(i,j,k) = in(i,j,k) - wgt(i,j,k) * (
                 + fli(i,j,k) - fli(i-1,j,k)
                 + flj(i,j,k) - flj(i,j-1,k) )
    end do
  end do
end do
```

Data-flow graph

Already 10 reads/writes in a very simple operator!



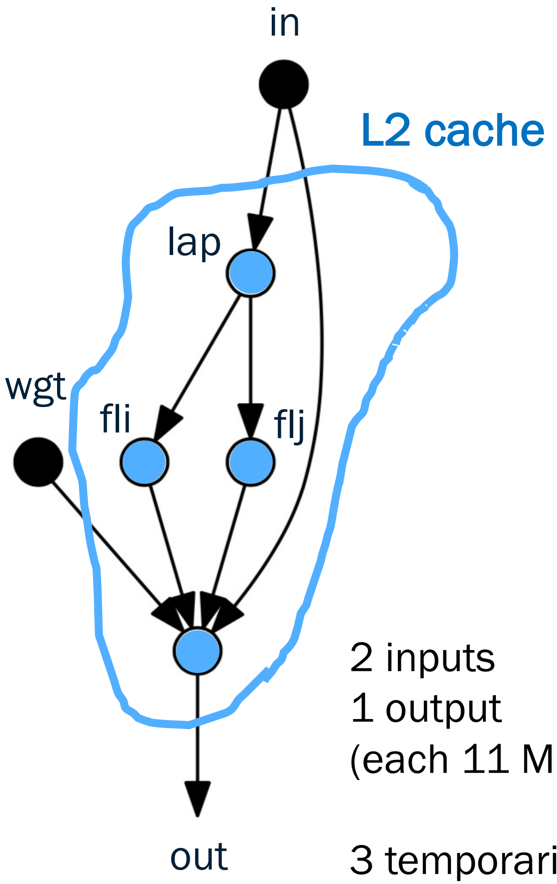
2 inputs
1 output
3 temporaries
(each 11 MB in memory)



```
do k = kstart, kend
  do j = jstart-1, jend+1
    do i = istart-1, iend+1
      lap(i,j,k) = in(i+1,j,k) + in(i-1,j,k)
                 + in(i,j+1,k) + in(i,j-1,k)
                 - 4.0*in(i,j,k)
    end do
  end do
end do

do k = kstart, kend
  do j = jstart-1, jend
    do i = istart-1, iend
      fli(i,j,k) = lap(i+1,j,k) - lap(i,j,k)
      flj(i,j,k) = lap(i,j+1,j,k) - lap(i,j,k)
    end do
  end do
end do

do k = kstart, kend
  do j = jstart, jend
    do i = istart, iend
      out(i,j,k) = in(i,j,k) - wgt(i,j,k) * (
                 + fli(i,j,k) - fli(i-1,j,k)
                 + flj(i,j,k) - flj(i,j-1,k) )
    end do
  end do
end do
```

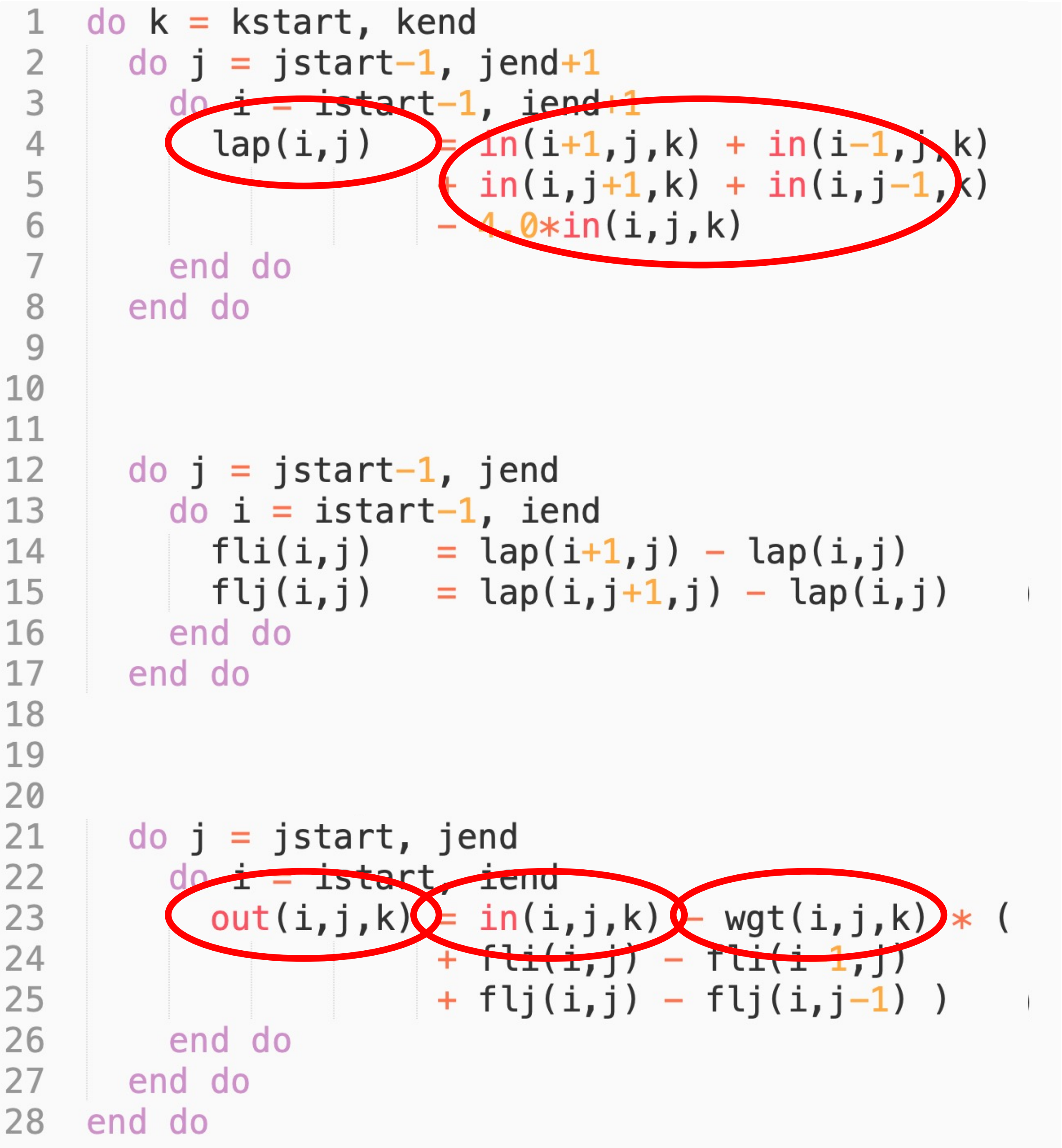
(adapted from Gysi et al, 2015)

Data-locality optimization



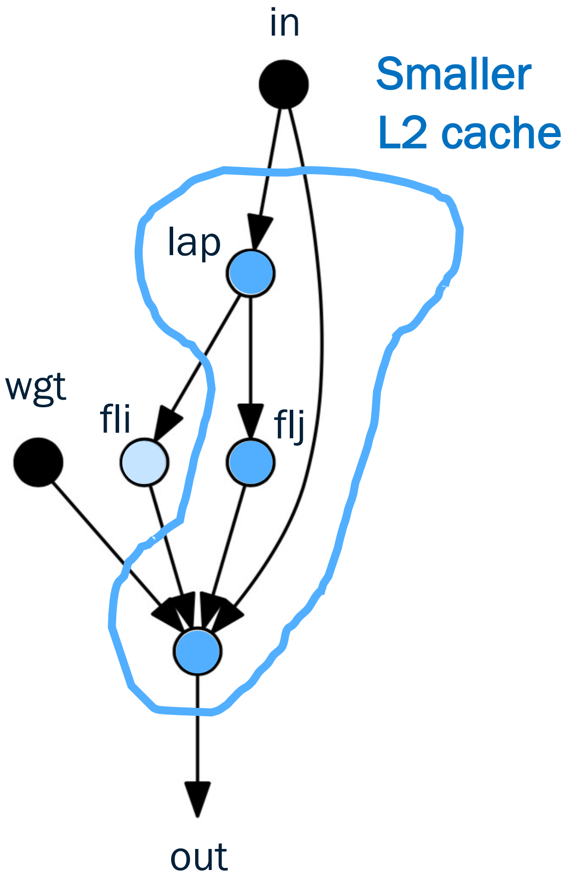
2 inputs
1 output
(each 11 MB in memory)

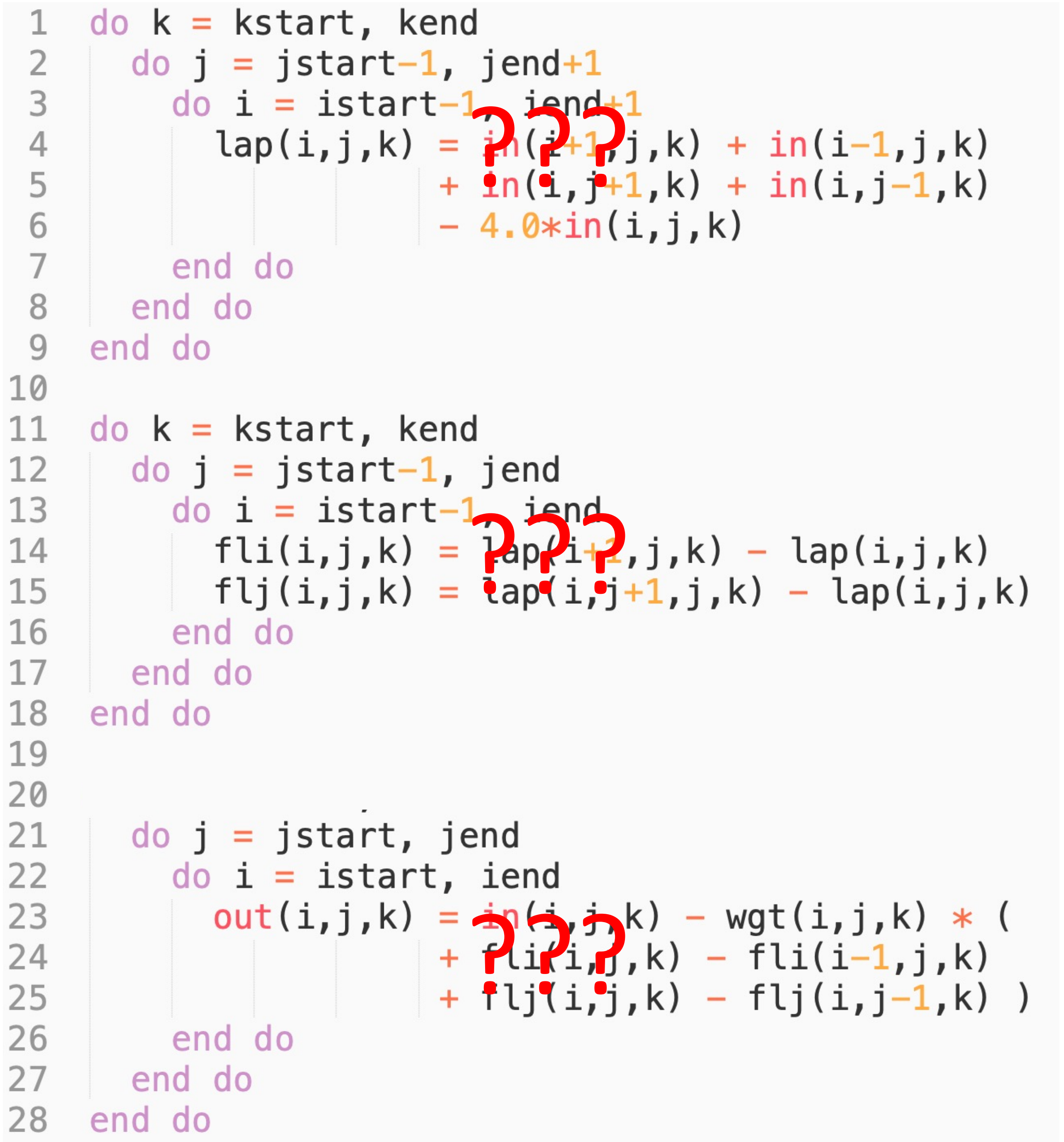
3 temporaries
(each 64 KB in memory)



```
do k = kstart, kend
  do j = jstart-1, jend+1
    do i = istart-1, iend+1
      lap(i,j)   = in(i+1,j,k) + in(i-1,j,k)
                 + in(i,j+1,k) + in(i,j-1,k)
                 - 4.0*in(i,j,k)
    end do
  end do



  do j = jstart-1, jend
    do i = istart-1, iend
      fli(i,j)   = lap(i+1,j) - lap(i,j)
      flj(i,j)   = lap(i,j+1,j) - lap(i,j)
    end do
  end do



  do j = jstart, jend
    do i = istart, iend
      out(i,j,k) = in(i,j,k) - wgt(i,j,k) * (
                 + fli(i,j) - fli(i-1,j)
                 + flj(i,j) - flj(i,j-1) )
    end do
  end do
end do
```

Impact of hardware



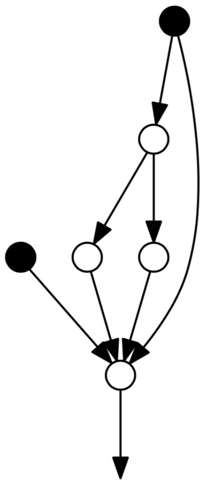


```
do k = kstart, kend
  do j = jstart-1, jend+1
    do i = istart-1, iend+1
      lap(i,j,k) = in(i+1,j,k) + in(i-1,j,k)
                 + in(i,j+1,k) + in(i,j-1,k)
                 - 4.0*in(i,j,k)
    end do
  end do
end do

do k = kstart, kend
  do j = jstart-1, jend
    do i = istart-1, iend
      fli(i,j,k) = lap(i+1,j,k) - lap(i,j,k)
      flj(i,j,k) = lap(i,j+1,j,k) - lap(i,j,k)
    end do
  end do
end do


  do j = jstart, jend
    do i = istart, iend
      out(i,j,k) = in(i,j,k) - wgt(i,j,k) * (
                 + fli(i,j,k) - fli(i-1,j,k)
                 + flj(i,j,k) - flj(i,j-1,k) )
    end do
  end do
end do
```

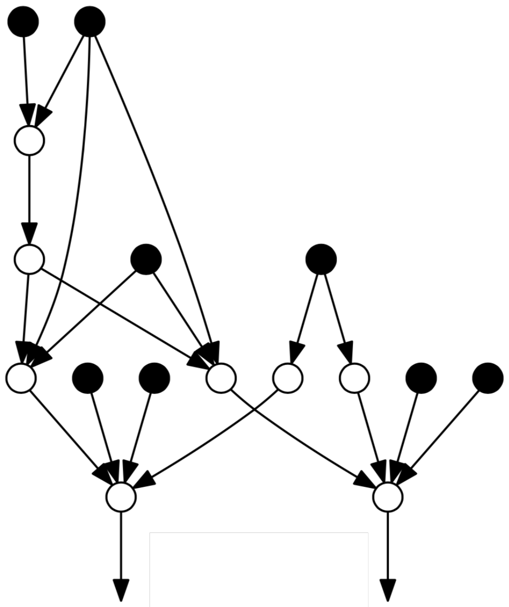
Loss of domain logic

- Optimal partitioning of graph and code generation are challenging tasks
- ”Optimal” code (inlining, loop-fusion, reordering, over-computation, …) does not retain domain logic (operators), changes with hardware target and can be hard to read.

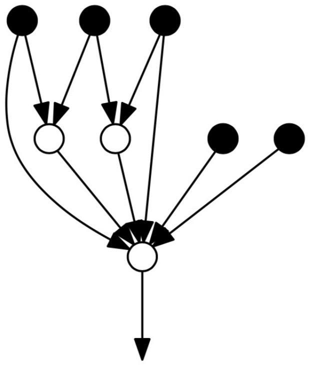
hordiff



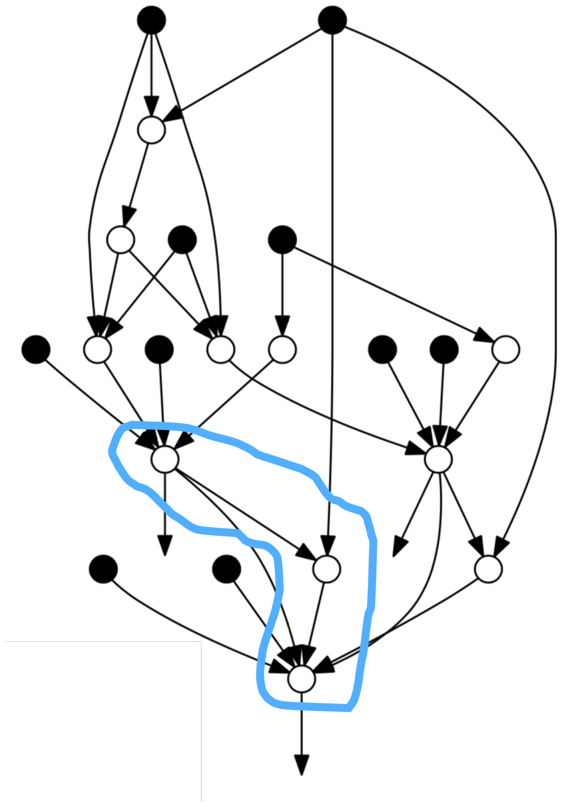
u,v-update



div



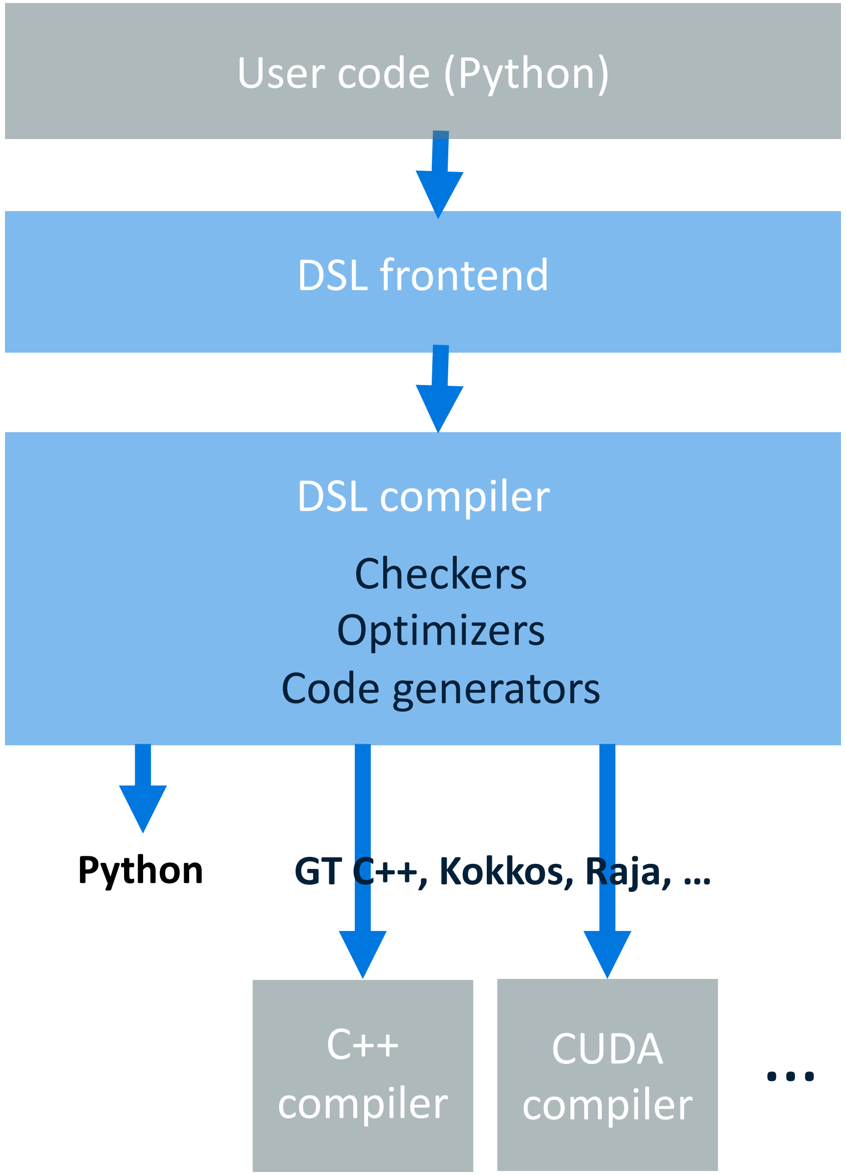
u,v-update & div



So what can we do?

DSL in Python

- A domain-specific language (DSL) is a programming language with concepts tailored to a specific application domain
- A compiler is responsible for transforming the high-level specification into code that runs on HPC cluster (code generation)
- By sacrificing generality, we can improve productivity, performance, and portability.
- User can specify execution backend (e.g. Python, optimized CPU, optimized GPU, …)
- Generate readable, efficient code by leveraging existing efforts (e.g. GridTools C++, Kokkos, Raja)



User code (Python)

DSL frontend

DSL compiler

Checkers
Optimizers
Code generators

Python

GT C++, Kokkos, Raja, …

C++ compiler

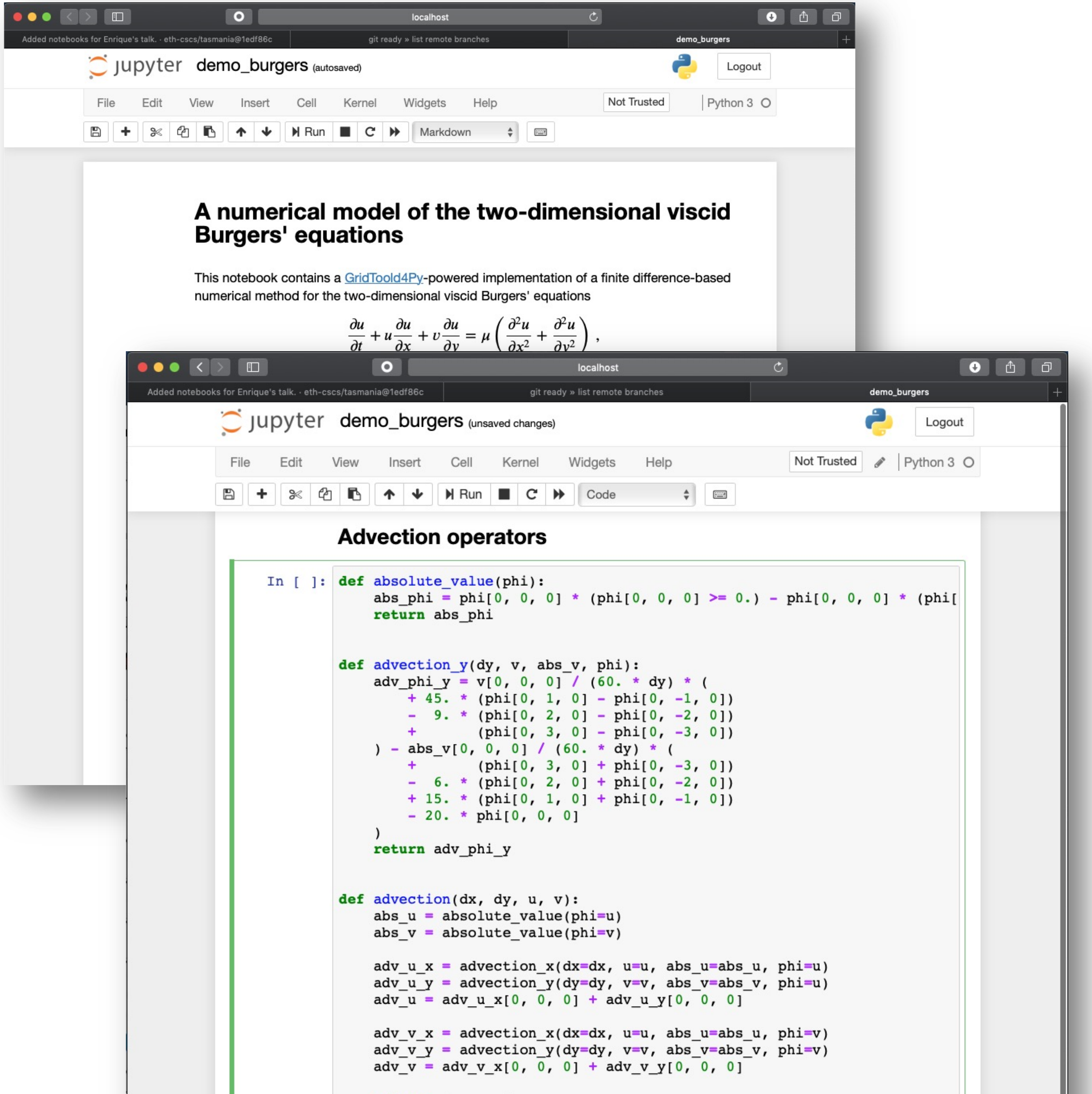
CUDA compiler

…

Why Python?

Large and growing community in the atmospheric and data science.

Mature integration with IDEs, huge ecosystem of packages, integration with visualization, interactive workflows using Jupyter notebooks on HPC clusters, …



A numerical model of the two-dimensional viscid Burgers' equations

This notebook contains a GridToold4Py-powered implementation of a finite difference-based numerical method for the two-dimensional viscid Burgers' equations

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right),$$

Advection operators

```
In [ ]: def absolute_value(phi):
            abs_phi = phi[0, 0, 0] * (phi[0, 0, 0] >= 0.) - phi[0, 0, 0] * (phi[
            return abs_phi


        def advection_y(dy, v, abs_v, phi):
            adv_phi_y = v[0, 0, 0] / (60. * dy) * (
                + 45. * (phi[0, 1, 0] - phi[0, -1, 0])
                -  9. * (phi[0, 2, 0] - phi[0, -2, 0])
                +       (phi[0, 3, 0] - phi[0, -3, 0])
            ) - abs_v[0, 0, 0] / (60. * dy) * (
                +       (phi[0, 3, 0] + phi[0, -3, 0])
                -  6. * (phi[0, 2, 0] + phi[0, -2, 0])
                + 15. * (phi[0, 1, 0] + phi[0, -1, 0])
                - 20. * phi[0, 0, 0]
            )
            return adv_phi_y


        def advection(dx, dy, u, v):
            abs_u = absolute_value(phi=u)
            abs_v = absolute_value(phi=v)

            adv_u_x = advection_x(dx=dx, u=u, abs_u=abs_u, phi=u)
            adv_u_y = advection_y(dy=dy, v=v, abs_v=abs_v, phi=u)
            adv_u = adv_u_x[0, 0, 0] + adv_u_y[0, 0, 0]

            adv_v_x = advection_x(dx=dx, u=u, abs_u=abs_u, phi=v)
            adv_v_y = advection_y(dy=dy, v=v, abs_v=abs_v, phi=v)
            adv_v = adv_v_x[0, 0, 0] + adv_v_y[0, 0, 0]
```

GT4Py (https://github.com/GridTools/gt4py)

- Joint open-source development with CSCS and MeteoSwiss.
- Features
  - No explicit parallelism or optimization
  - No loops
  - No explicit data-storage layout
  - Overhead-free functions (operators)

advection.py

```
import gt4py.gtscript as gtscript

@gtscript.function
def advection_x(dx, u, phi):
    adv_phi_x = u[0, 0, 0] / (60. * dx) * (
        + 45. * (phi[1, 0, 0] - phi[-1, 0, 0])
        -  9. * (phi[2, 0, 0] - phi[-2, 0, 0])
        +       (phi[3, 0, 0] - phi[-3, 0, 0])
    ) – abs(u[0, 0, 0]) / (60. * dx) * (
        +       (phi[3, 0, 0] + phi[-3, 0, 0])
        -  6. * (phi[2, 0, 0] + phi[-2, 0, 0])
        + 15. * (phi[1, 0, 0] + phi[-1, 0, 0])
        - 20. * phi[0, 0, 0]     )
    return adv_phi_x

@gtscript.stencil(backend=“numpy”)
def advection(
    in_u: gtscript.Field[np.float64],
    out_adv: gtscript.Field[np.float64],
    *,
    dx: float
):
    with computation(PARALLEL), interval(...):
        out_adv = advection_x(dx=dx, u=in_u, phi=u)
```

Demonstrate on FV3GFS

Apply the GT4Py DSL to port the FV3GFS model to Python.

Conduct a km-scale global simulation to provide a dataset to the the machine learning colleagues.

Current status

- Dynamical core (FV3) rewritten and validated
- 3 physical parametrizations ported, 1 on-going, 2 outstanding

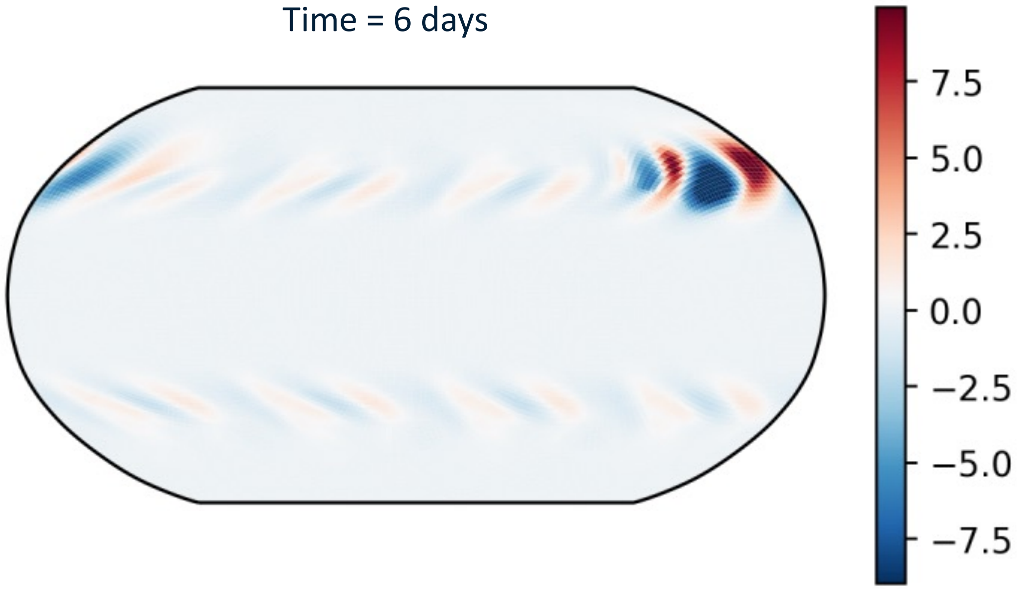
Dynamical Core (https://github.com/VulcanClimateModeling/fv3core)

- Dynamical core rewritten and validated using Python, x86 CPU and NVIDIA GPU backends.
- Parallelized using MPI from Python

- Simultaneous development of DSL frontend and compiler (e.g. corners & edges).
- **Next:** Refactoring for new DSL features. Performance improvements.

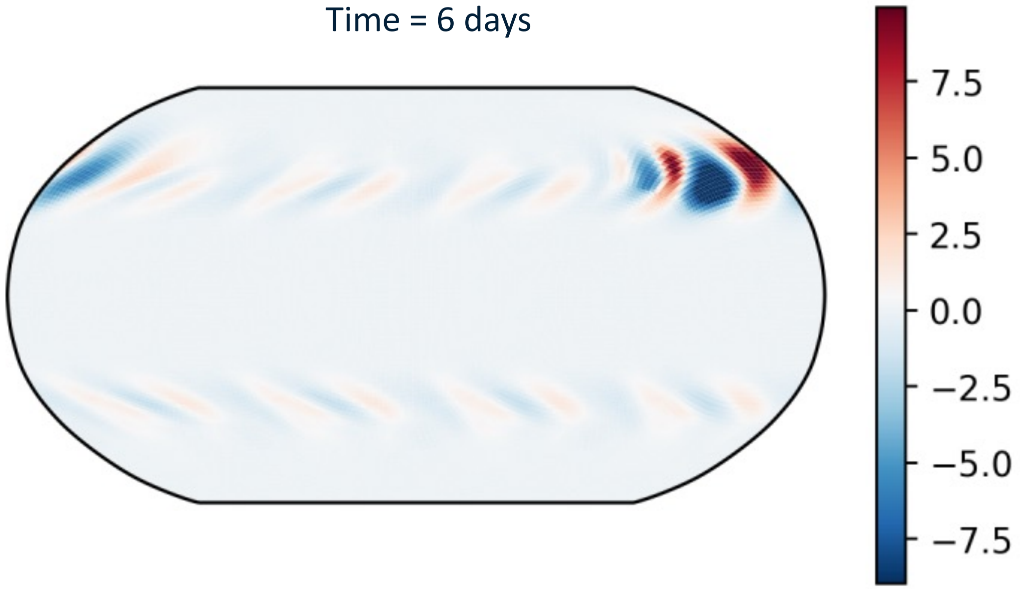
FV3 (Fortran)

Temperature anomaly [K]
Time = 6 days



FV3 (Python)

Temperature anomaly [K]
Time = 6 days



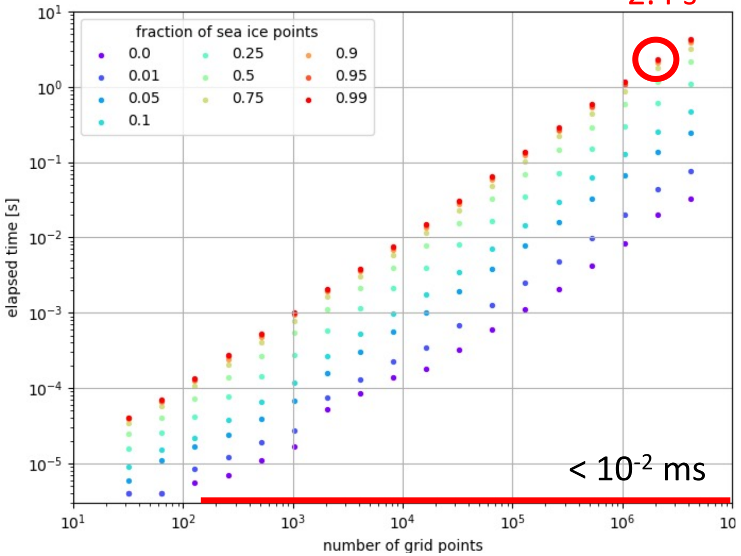
Performance (preliminary!)

Physical parameterization from FV3GFS ported by bachelor student as a course project

Performance results on a single node of Piz Daint supercomputer

CPU = Intel Xeon E5-2690 v3 (12 cores, @ 2.6 GHz)

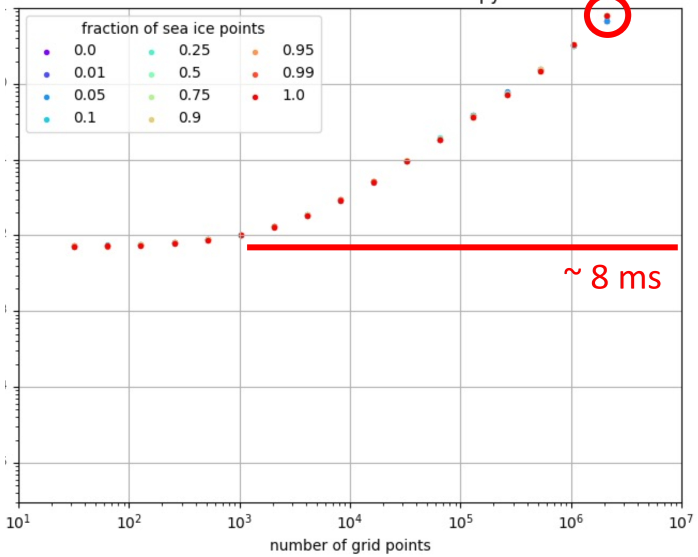
GPU = NVIDIA Tesla P100



Original (Fortran)

2.4 s

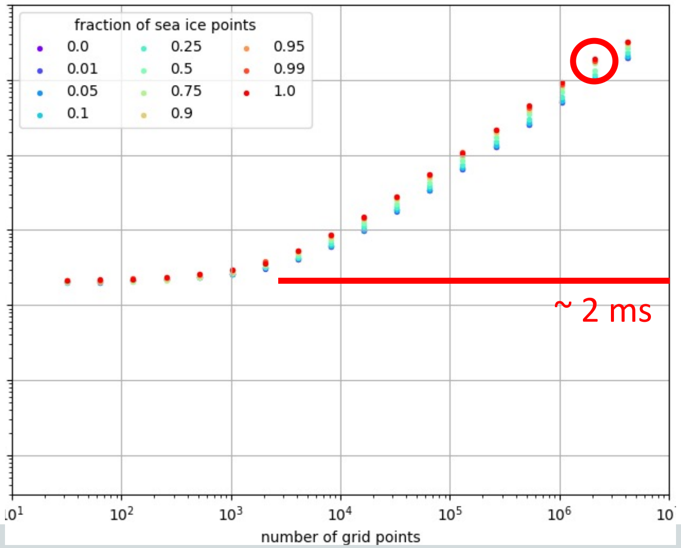
< 10^{-2} ms



Python (numpy)

7.5 s

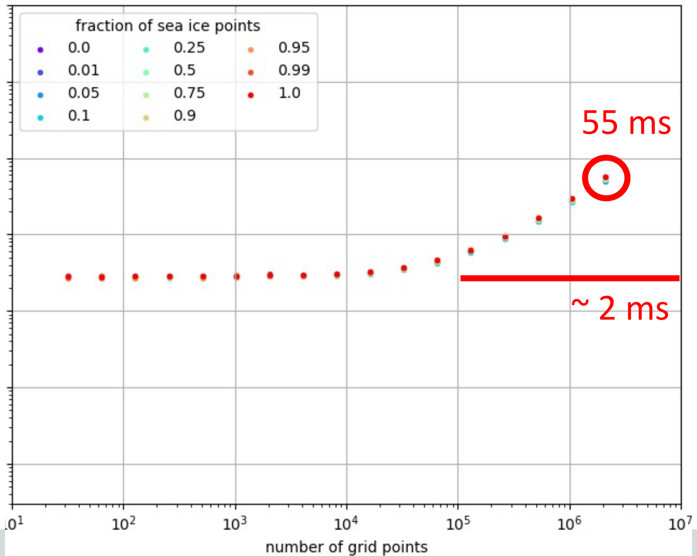
~ 8 ms



CPU (x86)

1.9 s

~ 2 ms



GPU (CUDA)

55 ms

~ 2 ms

Python wrapped FV3GFS

- Allows to run FV3GFS in parallel from Python
- Simply replace the dynamical core with our DSL version for validation.
- Many more use-cases
  - Integration with Python code (e.g. ML training, visualization, …)
  - Easy access to model for students
  - Rapid prototyping of new developments
  - …

basic_model.py

```python
import os
from mpi4py import MPI
import fv3gfs

if __name__ == '__main__':
    comm = MPI.COMM_WORLD
    rank = comm.Get_rank()

    fv3gfs.initialize()

    for i in range( fv3gfs.get_step_count() ):
        fv3gfs.step_dynamics()
        fv3gfs.step_physics()
        fv3gfs.save_restart()

    fv3gfs.cleanup()
```

Summary

1. Domain-specific languages have the potential for achieving a good balance between performance, portability and productivity (at the price of generality).

2. Optimizing for data-movement on different hardware targets can require a higher-level of abstraction in our codes.

3. No turn-key solutions, but we can build on existing tools and libraries.