Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Home Affairs FDHA
Federal Office of Meteorology and Climatology MeteoSwiss

Learning to forget

Lessons from adopting and maintaining a weather and climate model for heterogeneous HPC systems

Oliver Fuhrer, MeteoSwiss

Contributions from X. Lapillonne[1], C. Osuna[1], M. Bianco[2],L. Benedicic[2], T. Schulthess[2,3]

[1]MeteoSwiss, [2]CSCS, [2]ITP ETH Zurich,

ADAC Workshop, 20.6.2018

Can you spot the weather model?

Reality

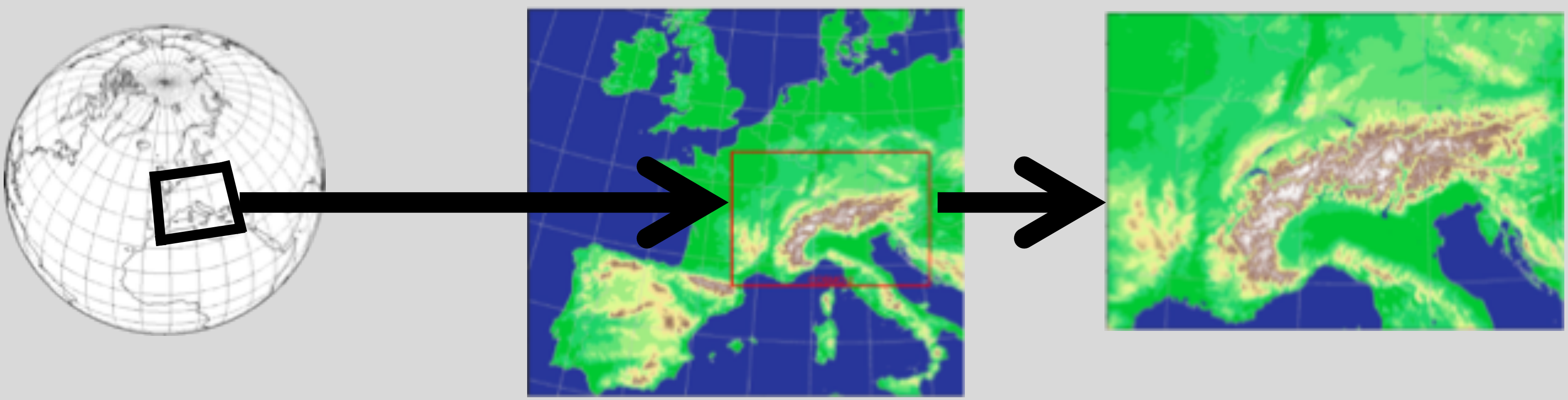
Where did we start in 2010?

Operations in 2010

EZMWF-Modell
16 km grid spacing
2 x per day
10 day forecast

COSMO-7
6.6 km grid spacing
3 x per day
3 day forecast

COSMO-2
2.2 km grid spacing
8 x per day
33 h forecast

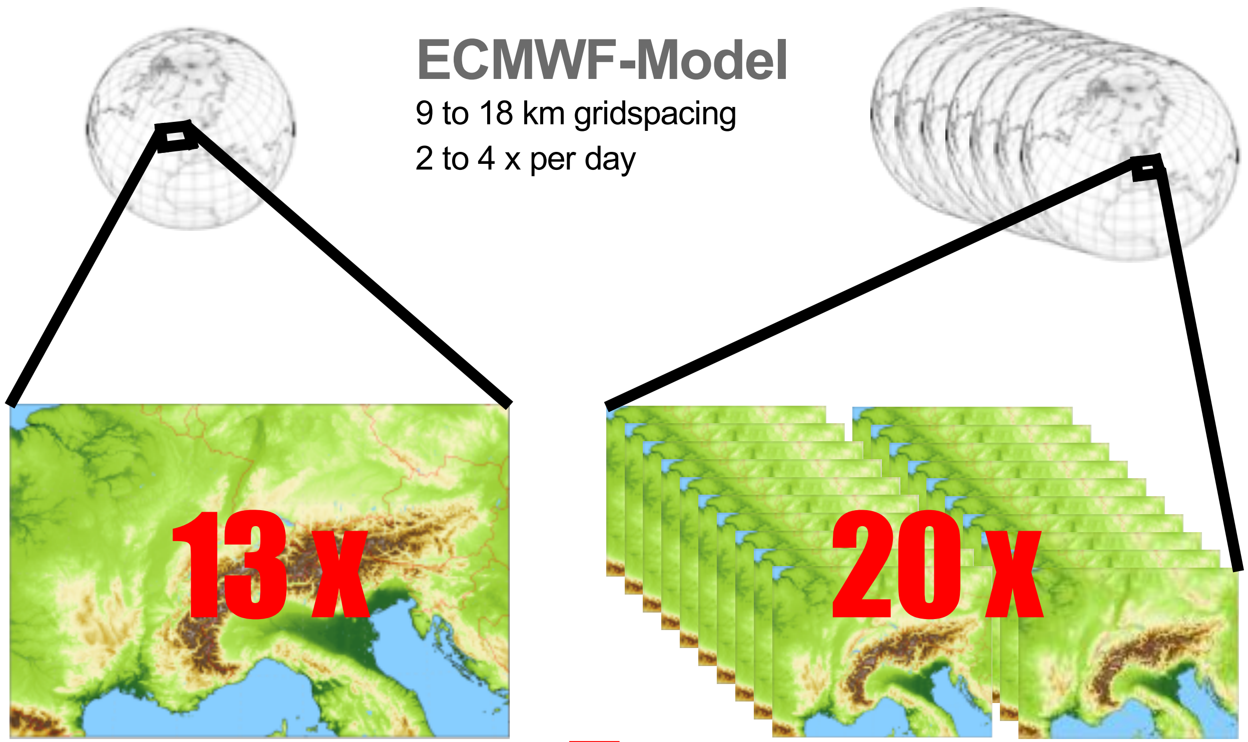


Strategy for next-generation

(computational effort relative to operational system) = 40 x

ECMWF-Model

9 to 18 km gridspacing
2 to 4 x per day



COSMO-1

1.1 km gridspacing
8 x per day
1 to 2 d forecast

COSMO-E

2.2 km gridspacing
2 x per day
5 d forecast
21 members

Ensemble data as imi ion: LETKF 7 x

Business as usual

Cray XE6 (Albis/Lema)
Current operational system at CSCS

Next System
Accounting for Moore's law

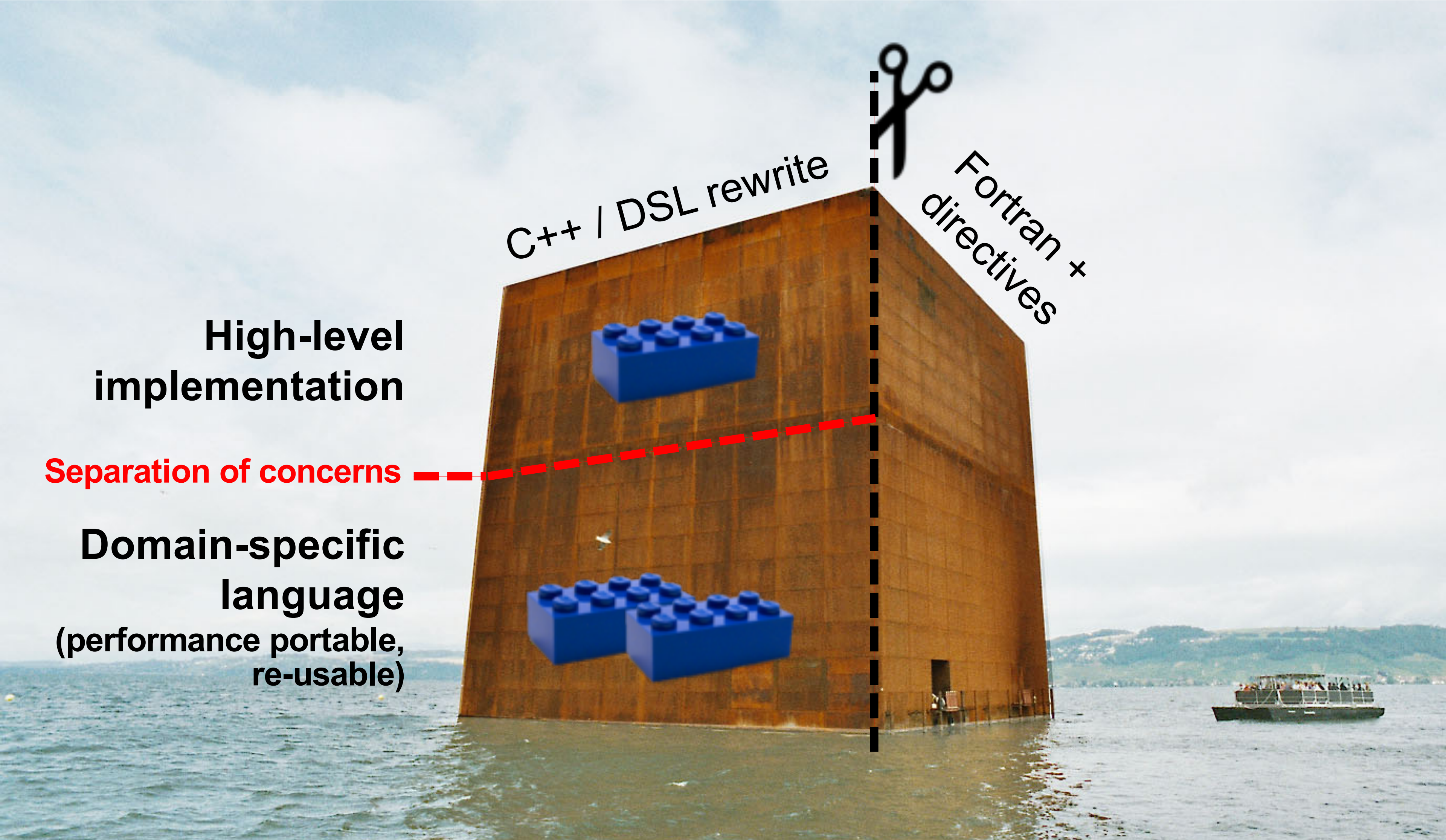
COSMO Model

Up or down?

- **Increase level of abstraction**
  - Hide implementation details
  - Can be disruptive
- **Decrease level of abstraction**
  - Add implementation details
  - Often incremental

Refactoring effort

Fuhrer et al. 2014 (doi: 10.14529/jsfi140103)
Gysi et al. 2015 (doi: 10.1145/2807591.2807627)



Results (COSMO-E benchmark)

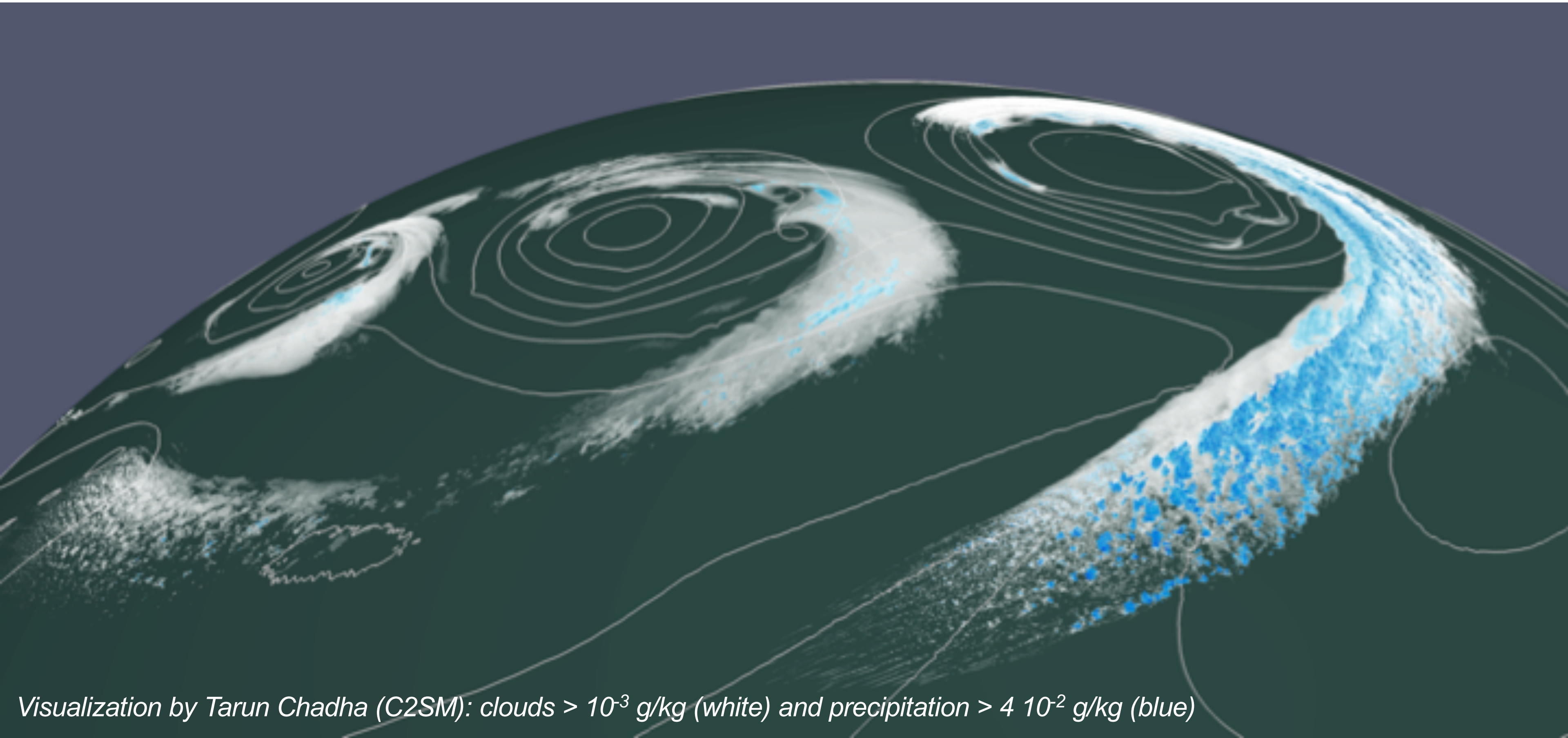
Co-design (simultaneous software, hardware and workflow re-design) allowed MeteoSwiss to increase computational load by 40x within 4–5 years

4x through investement into software.

	Piz Dora (old code)	Piz Kesch (new code)	Fa
Time-to-solution	~26 CPUs	~7 GPUs	**3.7 x**
Energy-to-solution	10 kWh	2.1 kWh	**4.8 x**
Size of system (cabinets)	1.4	0.38	**3.8 x**

Applications

- Used in production @MeteoSwiss
- Near-global simulations on full Piz Daint (4’888 GPU nodes, Fuhrer et al. 2018 GMD)



Visualization by Tarun Chadha (C2SM): clouds > 10^{-3} g/kg (white) and precipitation > 4 10^{-2} g/kg (blue)

Fortran + MPI + OpenACC + ...

is not the solution!

Software productivity gap!

```
durian:code fuhrer$ cloc E3SM/
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Fortran 90                    2812         244713         354664        1125623
C                             1149         154791         250480         648455
Fortran 77                     498          68946          98188         287216
HTML                           441           4457           2927         162587
XML                            914          15353           6546         142869
Bourne Shell                   404          22088          25988         130941
C/C++ Header                   399          11006          21174          60059
m4                              50           5161           1170          49869
Python                         329          12158          15054          45945
Perl                           172          14406          21944          41502
TeX                            172           4979           3830          29693
CMake                          422           4679           6771          26563
...
-------------------------------------------------------------------------------
SUM:                          8408         586306         832057        2835375
-------------------------------------------------------------------------------
```

Software productivity?

```
#ifdef _OPENACC
!$ACC DATA PCOPYIN( psi_c ), PCOPYOUT( grad_norm_psi_e ), IF( i_am_accel_node .AND. acc_on )
!ACC_DEBUG UPDATE DEVICE( psi_c ), IF( i_am_accel_node .AND. acc_on )
!$ACC PARALLEL &
!$ACC PRESENT( ptr_patch, iidx, iblk, psi_c, grad_norm_psi_e ), &
!$ACC IF( i_am_accel_node .AND. acc_on )

!$ACC LOOP GANG
#else
!$OMP PARALLEL

!$OMP DO PRIVATE(jb,i_startidx,i_endidx,je,jk) ICON_OMP_DEFAULT_SCHEDULE
#endif
  DO jb = i_startblk, i_endblk

  CALL get_indices_e(ptr_patch, jb, i_startblk, i_endblk, &
                     i_startidx, i_endidx, rl_start, rl_end)

!$ACC LOOP VECTOR COLLAPSE(2)
#ifdef __LOOP_EXCHANGE
    DO je = i_startidx, i_endidx
      DO jk = slev, elev
#else
    DO jk = slev, elev
      DO je = i_startidx, i_endidx
#endif
      !
      ! compute the normal derivative
      ! by the finite difference approximation
      ! (see Bonaventura and Ringler MWR 2005)
      !
       grad_norm_psi_e(je,jk,jb) =  &
          &  ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) - &
          &    psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )  &
          &  * ptr_patch%edges%inv_dual_edge_length(je,jb)

      ENDDO

    END DO

  END DO
#ifdef _OPENACC
!$ACC END PARALLEL
!ACC_DEBUG UPDATE HOST( grad_norm_psi_e ), IF( i_am_accel_node .AND. acc_on )
! Add $ser directives here
!$ACC END DATA
#else
!$OMP END DO NOWAIT

!$OMP END PARALLEL
#endif
```

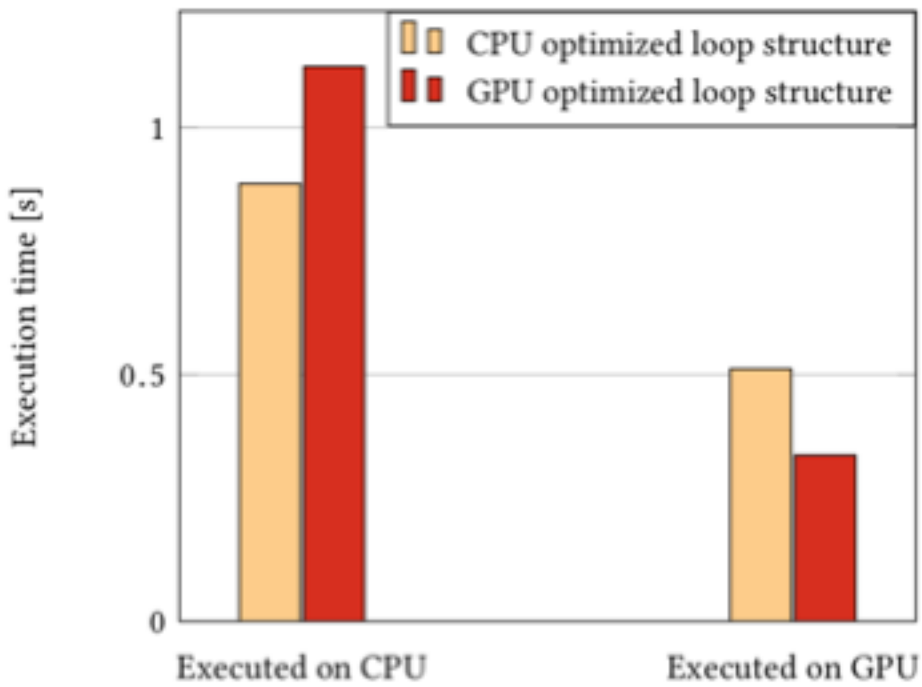
Efficiency myth

	runtime	occupancy	DRAM throughput read	DRAM throughput write	shared memory	register usage
non-blocked (naive)						
K20X	0.53 ms	0.266	>75.1 GB/s	>35.5 GB/s	0 B	47-53
K20	0.68 ms	0.285	>39.1 GB/s	>26.3 GB/s	0 B	37-44
blocked						
K20X	0.90 ms	0.283	13.9 GB/s	62.9 GB/s	0 B	73
K20	0.69 ms	0.591	12.7 GB/s	63.1 GB/s	4 B	46
shared						
K20	0.54 ms	0.600	15.9 GB/s	16.1 GB/s	4.272 KB	39
shared-3D						
K20	0.56 ms	0.670	15.4 GB/s	16.1 GB/s	4.272 KB	34
STELLA						
K20X	0.29 ms	[illegible]				
K20	0.35 ms	0.90				

Heavily optimized code is typically faster than Fortran + OpenACC, but also unreadable and unmaintainable!

Performance portability myth

- Radiation scheme on CPU (Intel E5-2690v3 “Haswell”) and GPU (NVIDIA Tesla K80) using Fortran + OpenMP + OpenACC



Lappillonne and Fuhrer, PPL, 2018
Clement et al. 2018, PASC’18

DSL in C++ may also not be the solution!

STELLA

Data fields

```
IJKRealField lapfield, datafield;
```

```
enum { data, lap };

template<typename TEnv>
struct Laplacian
{
 STENCIL_STAGE(TEnv)
 STAGE_PARAMETER(FullDomain, data)
 STAGE_PARAMETER(FullDomain, lap)

 static void Do(Context ctx, FullDomain)
 {
   ctx[lap::Center()] =
     -4.0 * ctx[data::Center()]
     + ctx[data::At(iplus1)]
     + ctx[data::At(iminus1)]
     + ctx[data::At(jplus1)]
     + ctx[data::At(jminus1)];
 }
};
```

Loop body

```
Stencil stencil;
StencilCompiler::Build(
  stencil,
  "Example",
  calculationDomainSize,
  StencilConfiguration<Real, BlockSize<32,4> >(),
  pack_parameters(
    Param<lap,  cInOut>(lapfield),
    Param<data, cIn>.  (datafield)
  ),
  define_loops(
    define_sweep<cKIncrement>(
      define_stages(
        StencilStage<
          Laplacian, IJRange<cComplete,0,0,0,0>,
          KRangeFullDomain >()
      )
    )
  )
);
```

Stencil

Run

```
for(int step = 0; step < numOfSteps; ++step)
{
  stencil.Apply();
}
```

STELLA DSL in C++ critique

+

- C++ well supported language
- Abstract (some) hardware dependent details
- High efficiency
- DSL allows for (partial) separation of concerns between domain-scientist and computer scientist

-

- C++ not well accepted by weather and climate community (boiler plate)
- No introspection / global optimization
- Backends complicated to implement (template meta-programming)

No turn key solution!

Is it time for a SDK for weather and climate?

Developmet Workflow

“scientist”

Mathematical Model

$$T = T(x,z,t)$$
$$T(x,z,t=0) = T_0(x,z)$$
$$\frac{\partial T}{\partial t} = \alpha\,\nabla^2 T$$

↓

Discretization, Solver

$$\nabla^2 T_{i,k} = \frac{T_{i+1,k} + T_{i-1,k} + T_{i,k+1} + T_{i,k-1} - 4T_{i,k}}{\Delta^2}$$

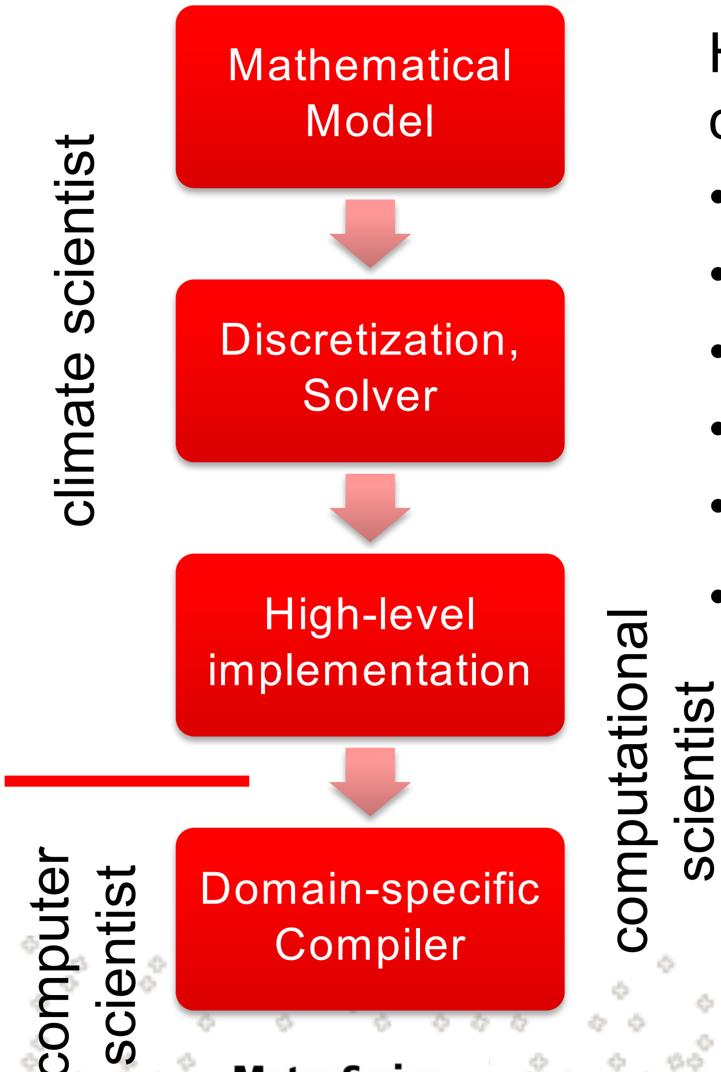
↓

Implementation

software engineer

```
do k = 2, nk-1
  do i = 2, ni-1
    lap(i,k) = ( T(i+1,k) + T(i-1,k) + &
                 T(i,k+1) + T(i,k-1) - &
               4*T(i,k) )/dx/dy
  end do
end do
```

We need to learn to forget



High-level language for weather and climate

- No explicit data structure
- No loops / execution schedule
- No explicit threading / vectorization
- No directives
- No HW-dependent details
- …

Separation of concerns

GridTools Framework

- "SDK for weather and climate science"
- Joint development between CSCS / MeteoSwiss
- Domain-specific for Earth system model components
- Regional (production) and global grids (prototype)
- Multiple APIs (C++, Python, gtclang)

Example (GT4Py)

```python
# Definitions function
def horizontal_diffusion(data, weight):
    i = gt.Index()
    j = gt.Index()

    laplacian = gt.Equation()
    flux_i = gt.Equation()
    flux_j = gt.Equation()
    diffusion = gt.Equation()

    # Laplacian operator
    laplacian[i, j] = -4.0 * data[i, j] + (data[i-1, j]
                                           + data[i+1, j]
                                           + data[i, j-1]
                                           + data[i, j+1])
    # Horizontal flux
    flux_i[i, j] = laplacian[i+1, j] - laplacian[i, j]
    # Vertical flux
    flux_j[i, j] = laplacian[i, j+1] - laplacian[i, j]
    # Diffusion
    diffusion[i, j] = weight[i, j] * (flux_i[i-1, j]
                                      - flux_i[i, j]
                                      + flux_j[i, j-1]
                                      - flux_j[i, j])
    return diffusion

# Create computation domain
my_domain = gt.domain.Rectangle((2, 2), (61, 61))

# Create stencil object
stencil = gt.Stencil(definitions_func=horizontal_diffusion,
                     inputs={"data": array_a,
                             "weight": array_b},
                     outputs={"diffusion": array_out},
                     domain=my_domain,
                     mode=gt.mode.ALPHA)
```

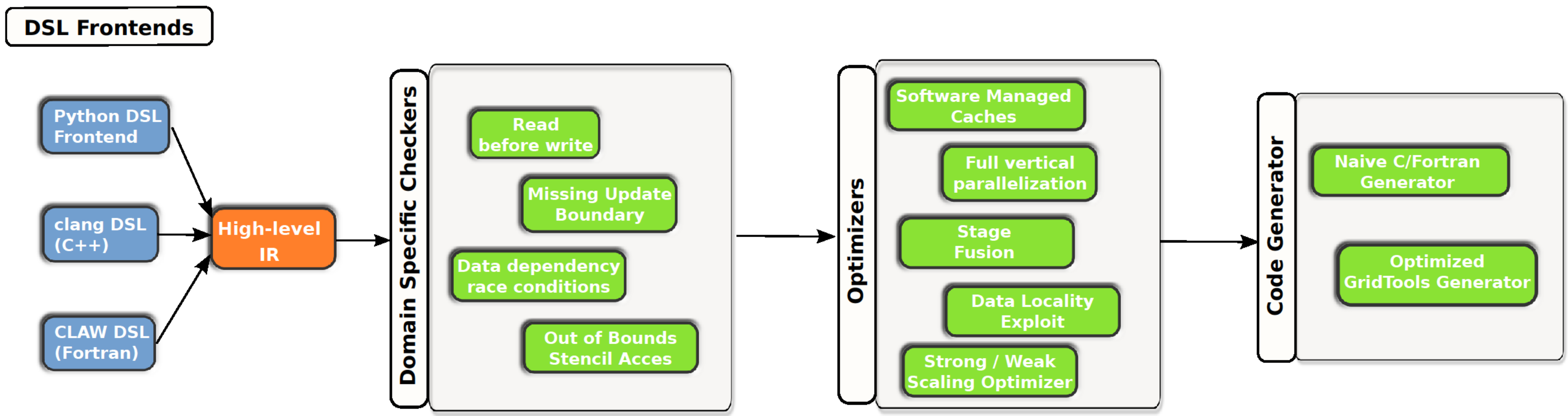
- Mathematical operators (stencils)
- Data fields
- Region over which operator is applied
- Boundary conditions (not shown in exmpale)
- High-level, declarative syntax
- Numpy as well as high-performance backends (x86 multi-core, NVIDIA GPU, Xeon Phi)

Example (gtclang)

```
function avg {
  offset off
  storage in

  avg = 0.5 * ( in(off) + in() )
}

function coriolis_force {
  storage fc, in

  coriolis_force = fc() * in()
}

operator coriolis {
  storage u_tend, u, v_tend, v, fc

  vertical_region ( k_start , k_end ) {
      u_tend += avg(j-1, coriolis_force(fc, avg(...
      v_tend -= avg(i-1, coriolis_force(fc, avg(...
    }
  }
}
```

- Higher productivity and code safety
- Based on LLVM/Clang compiler framework
- Allows for global optimization ”across kernels”
- Generates efficient code for x86 multi-core,NVIDIA GPUs, Intel Xeon Phi, and ARM (prototype)
- 4x – 6x reduction in LOC

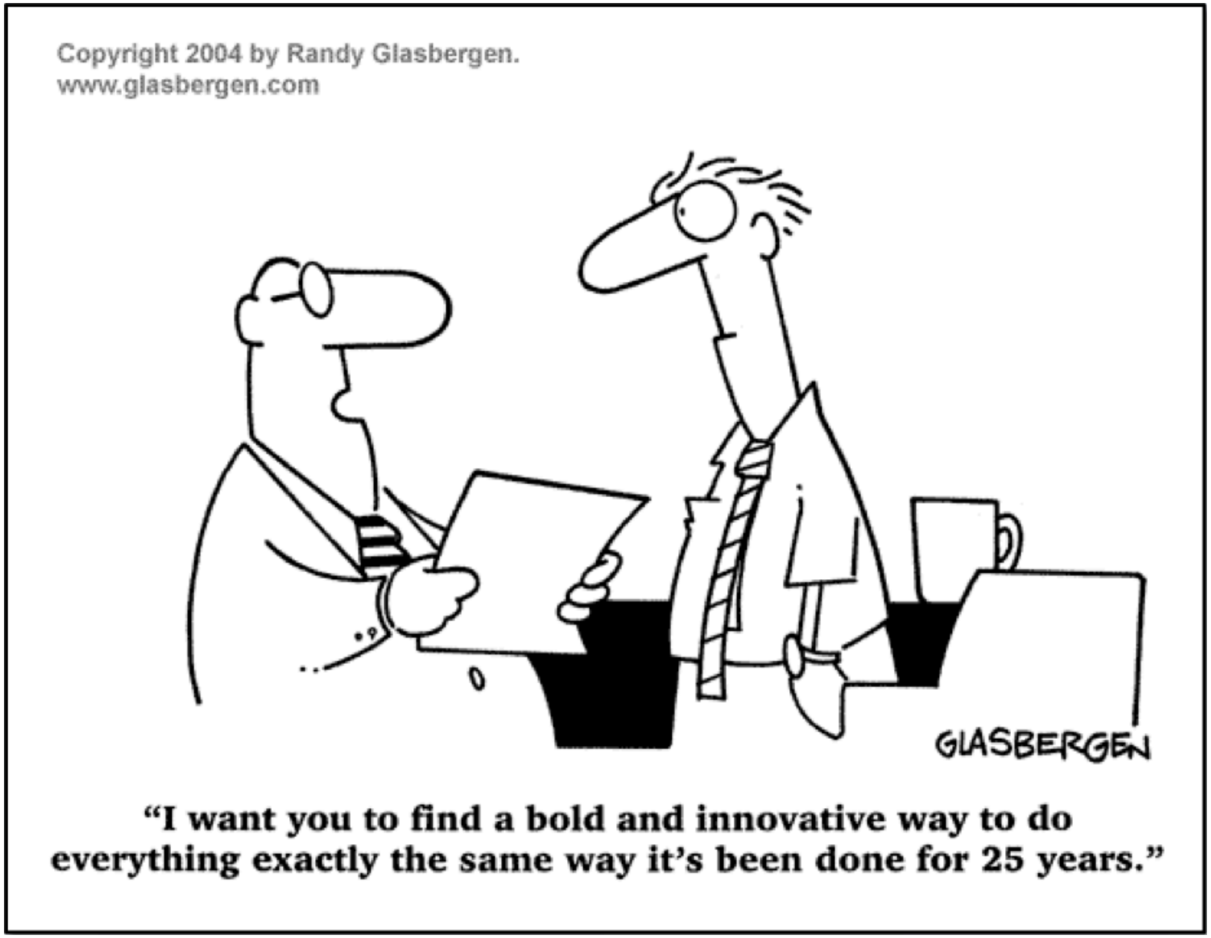
Compiler Toolchain



Summary

- Weather and climate community is struggling to keep up
  - very few models able to target multiple hardware architectures
  - software productivity gap
  - no turn-key solutions
- Large software effort for COSMO to enable efficient execution on multiple hardware architectures
- Learnings
  - It's not about porting, it's about maintaining
  - Fortran + directives exacerbate problem (ad-interim solution?)
  - DSL embedded C++ not readily adopted by domain-scientists
  - We need a high-level software development environment for weather and climate

Thank you! Questions?



MeteoSwiss
Operation Center 1
CH-8058 Zurich-Airport
T +41 58 460 91 11
www.meteoswiss.ch

MeteoSvizzera
Via ai Monti 146
CH-6605 Locarno-Monti
T +41 58 460 92 22
www.meteosvizzera.ch

MétéoSuisse
7bis, av. de la Paix
CH-1211 Genève 2
T +41 58 460 98 88
www.meteosuisse.ch

MétéoSuisse
Chemin de l'Aérologie
CH-1530 Payerne
T +41 58 460 94 44
www.meteosuisse.ch