Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

HP2C

Adapting the COSMO weather and climate model for hybrid architectures.

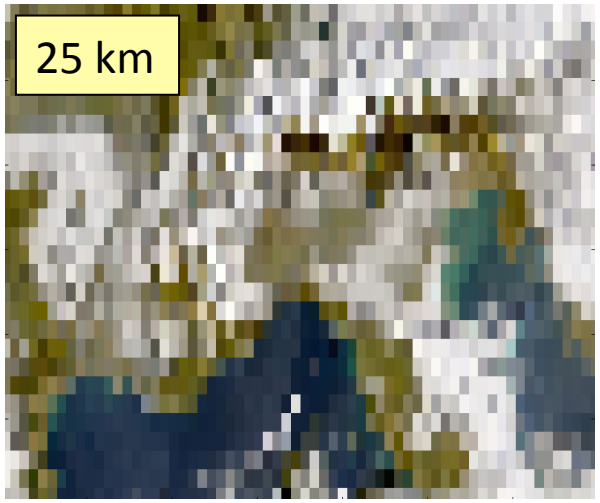
Overview

- Weather and climate modeling
- COSMO model
- Porting approaches
- Performance results
- Discussion

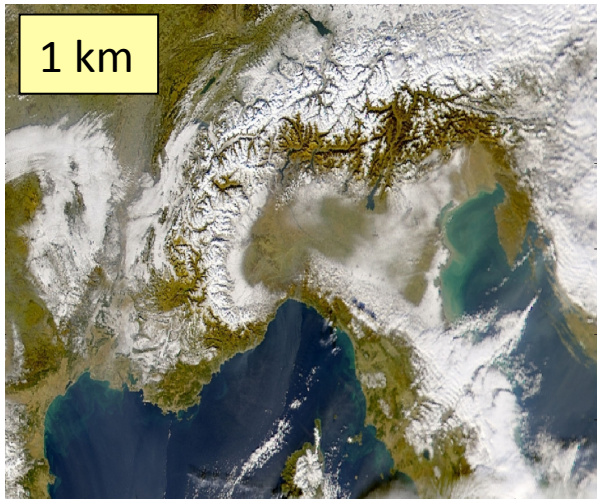
Why do weather and climate simulations require increasing amount of HPC resources?

Higher resolution

Global climate model



Global weather model

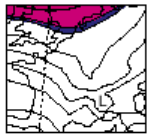
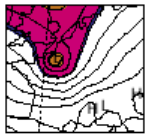


Regional weather model

Resolution is of key importance to reduce the uncertainty of weather and climate model predictions
(e.g. regional weather, extreme events, underlying processes)

2 x horizontal resolution ≈ 8 x computational cost

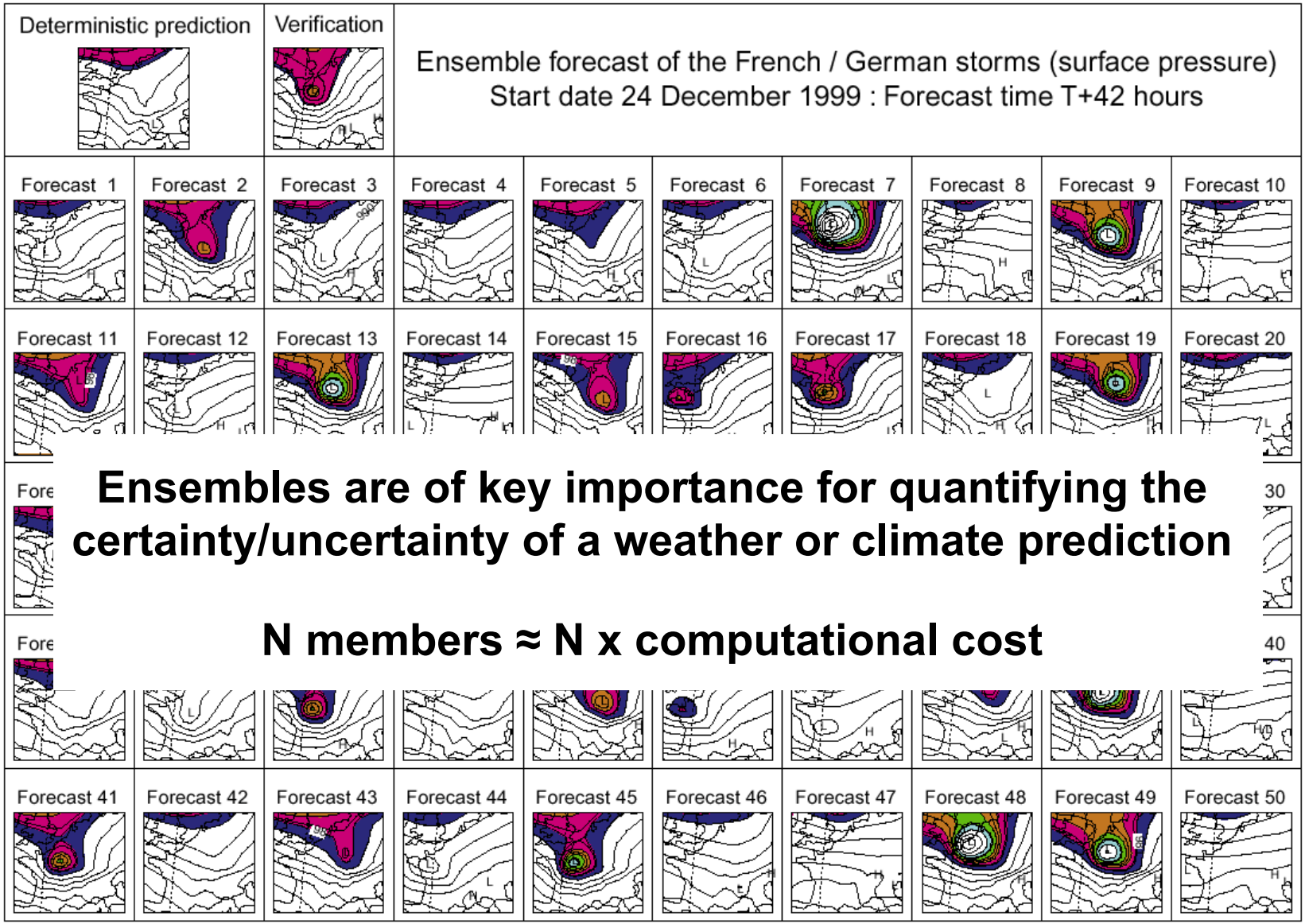
Single forecast (T+42h)

Deterministic prediction	Verification	Ensemble forecast of the French / German storms (surface pressure) Start date 24 December 1999 : Forecast time T+42 hours
		

"The overall loss in Europe as a whole came to €6bn, making Lothar the largest loss ever to be carried by the insurance industry following a European windstorm event." (Munich Re, 2002)

ECMWF, Munich Re (2002)

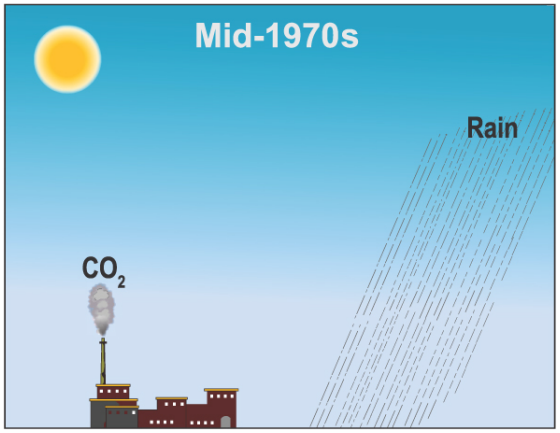
Ensemble forecast (T+42h)

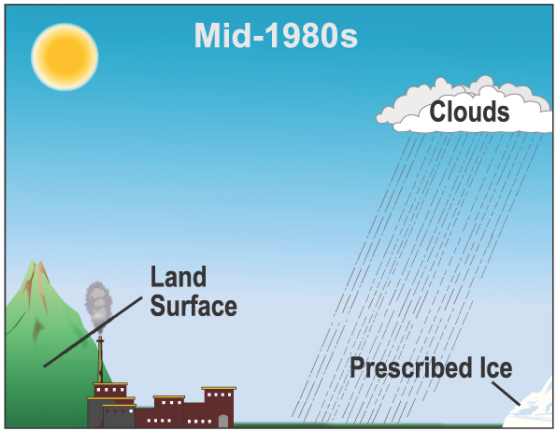


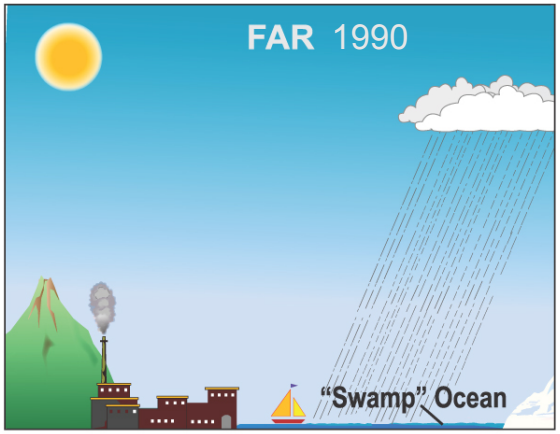
Ensembles are of key importance for quantifying the certainty/uncertainty of a weather or climate prediction

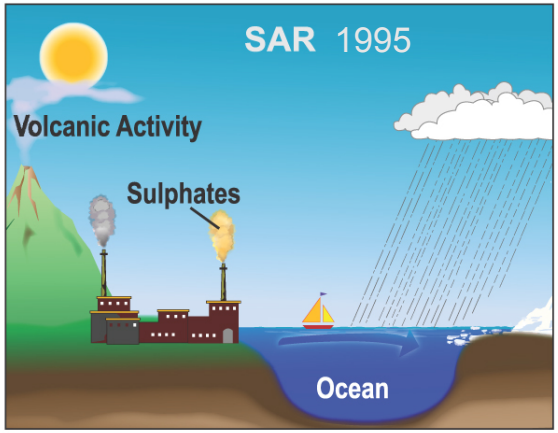
N members ≈ N x computational cost

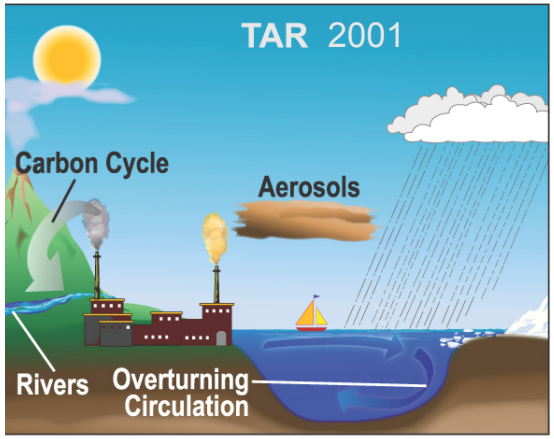
Model complexity

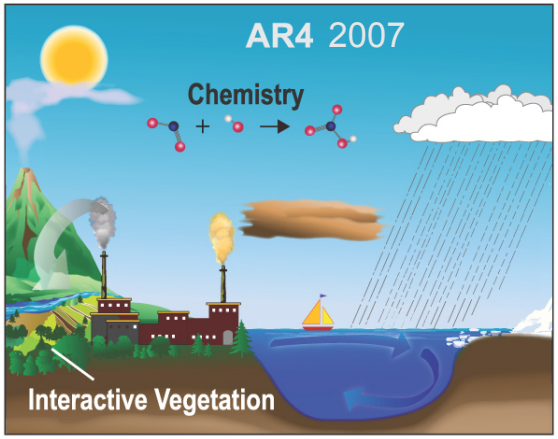








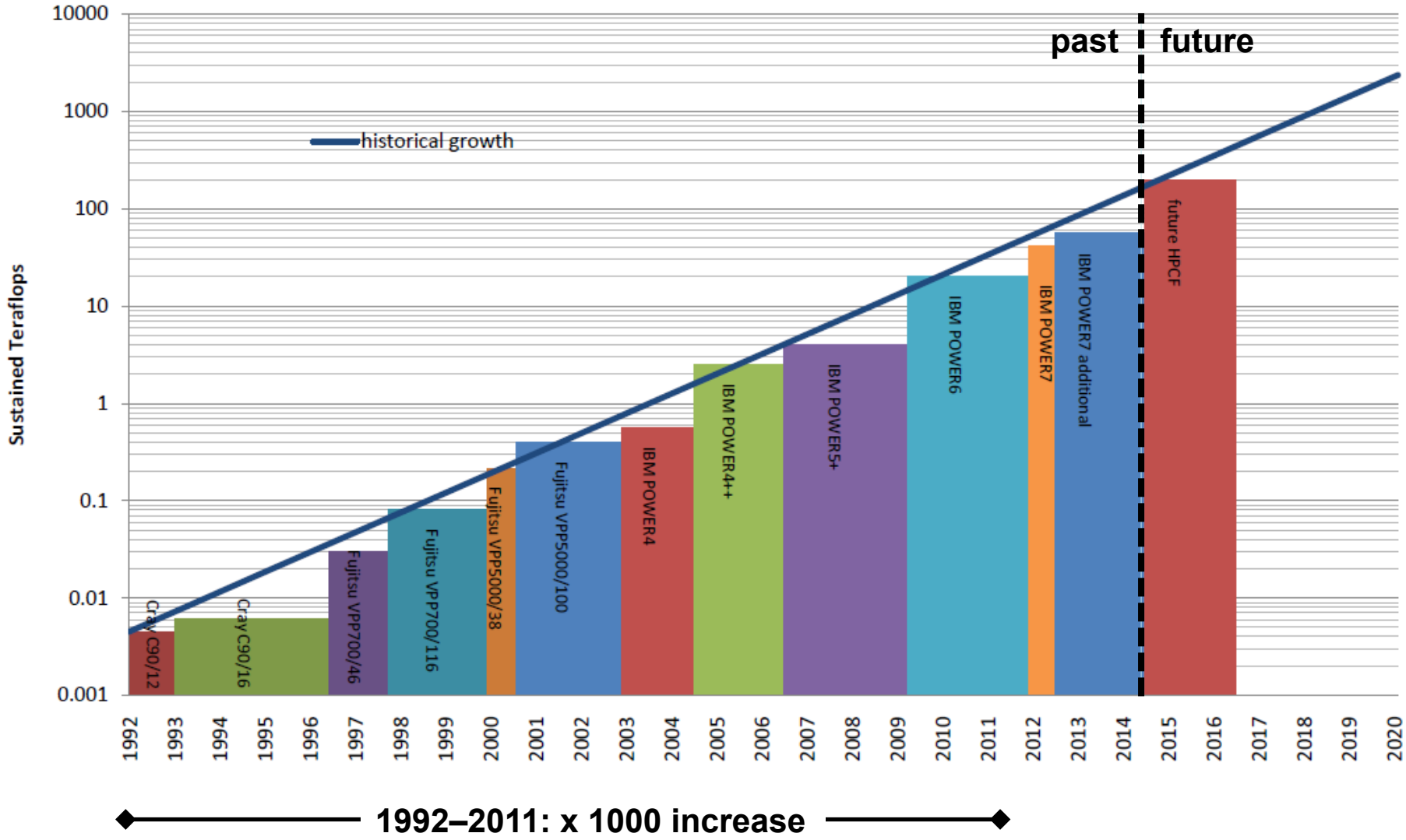




IPCC, AR4

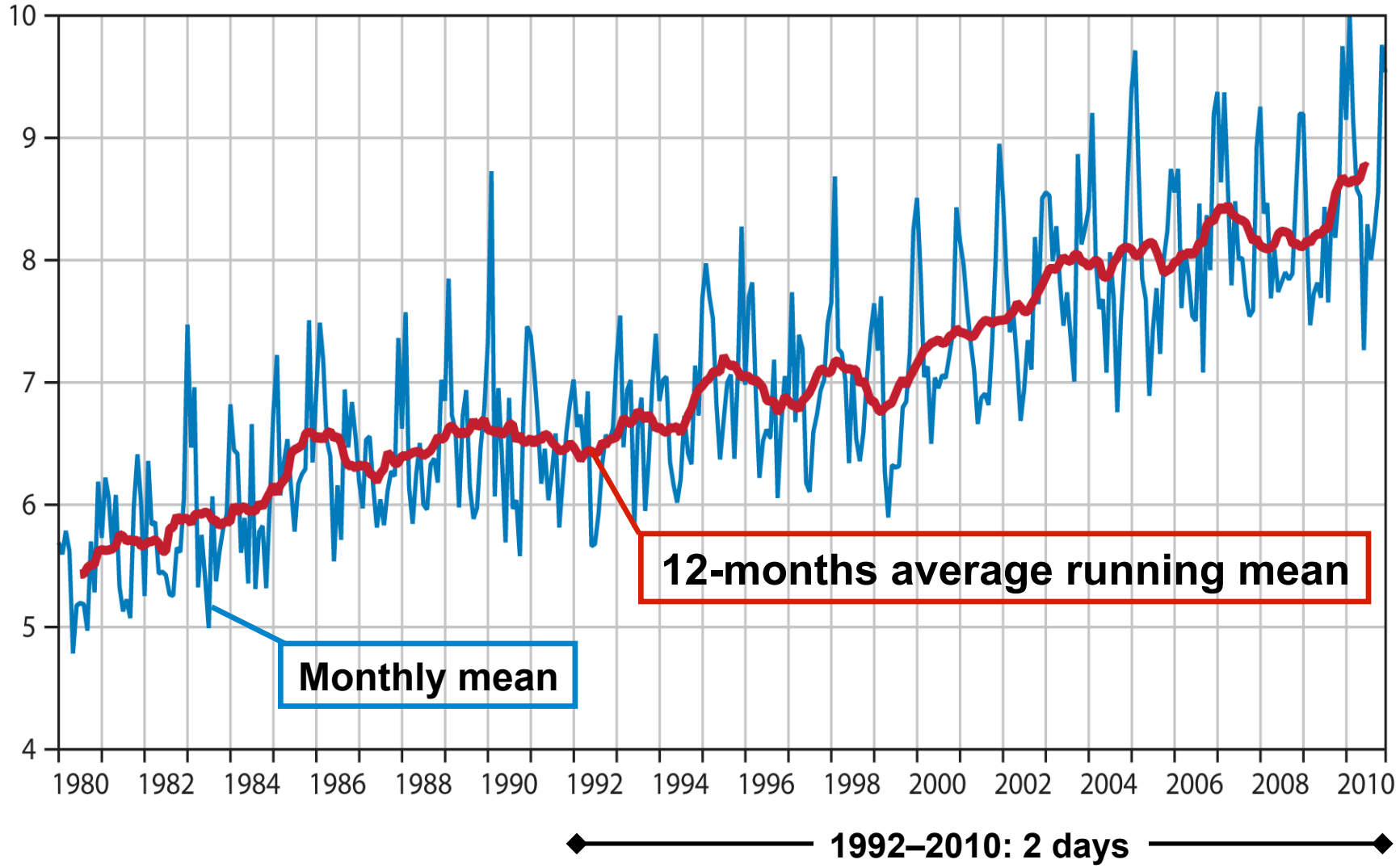
What is the benefit of increasing HPC resources for weather and climate simulation?

ECWMF sustained performance

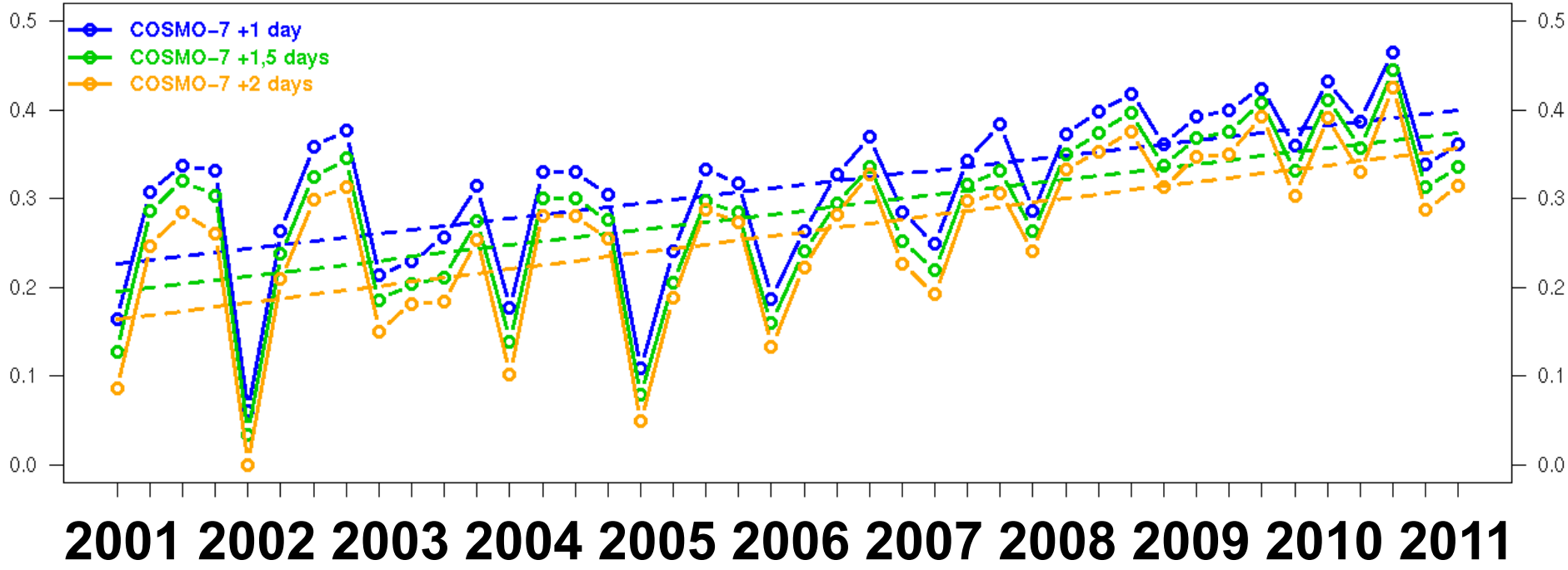


Long-term evolution of global forecast

IFS, ACC 500hPa reaches 60%



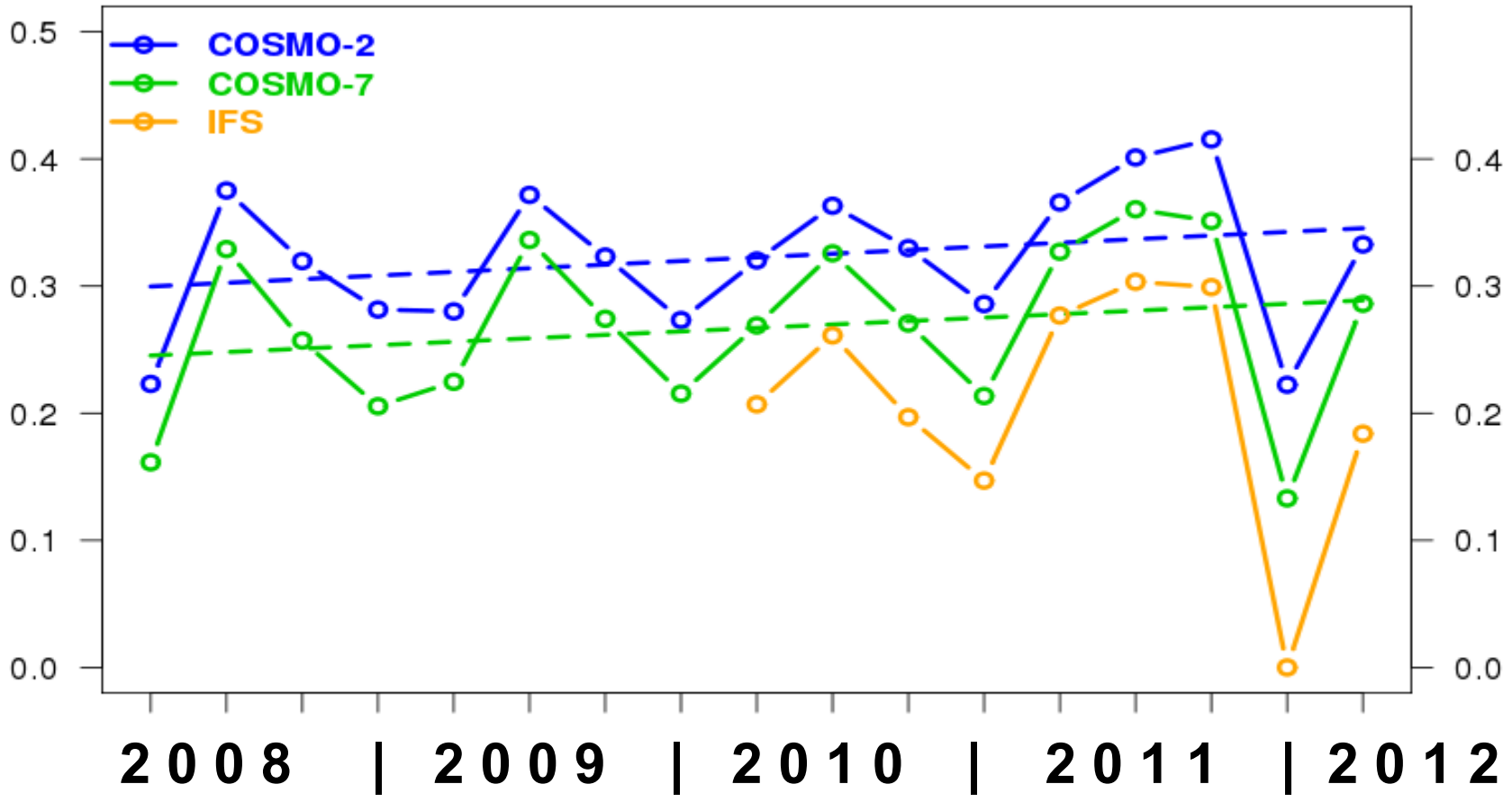
Long-term evolution of regional forecast
(Temperature, seasonal verification)



Verification against observations over Europe

0 = lowest score / 1 = perfect forecast

Benefit of regional models
(Wind, comparison IFS / COSMO-7 / COSMO-2)



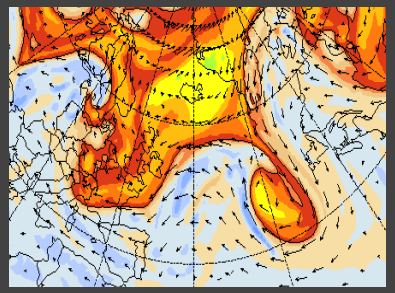
Verification against observations over Switzerland

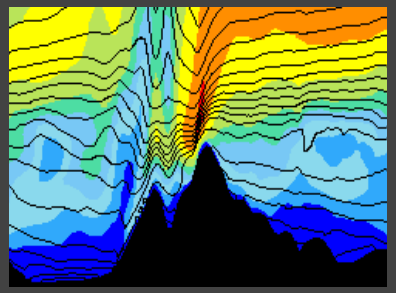
0 = lowest possible score / 1 = perfect forecast

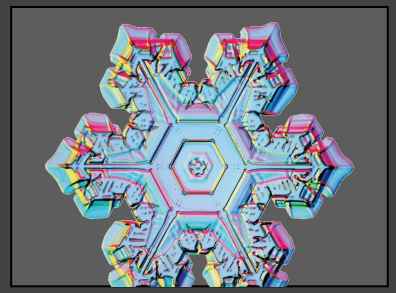
Regional weather and climate modeling using the COSMO model

The weather and climate system
(multi-scale, multi-physics)

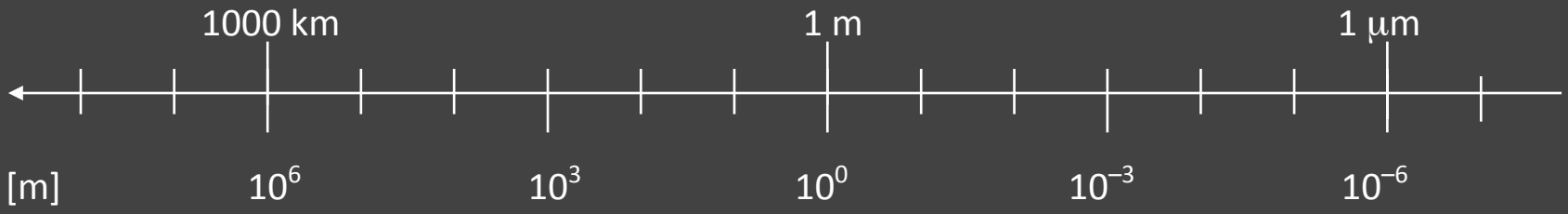
10^6 m = 1 Mm

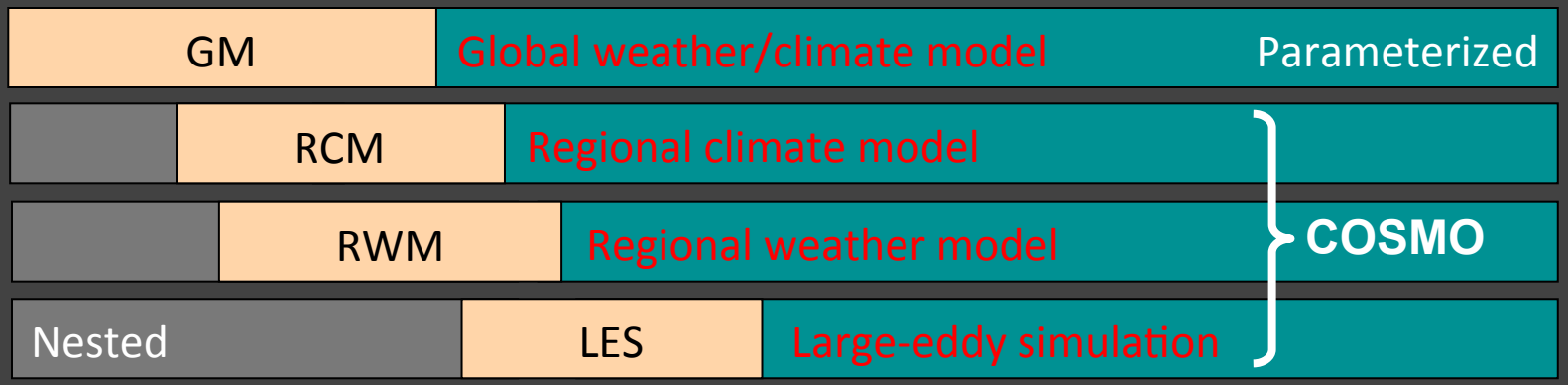






10^{-6} m = 1 μm





C. Schär (ETH)

The modeling chain

ECMWF-Model

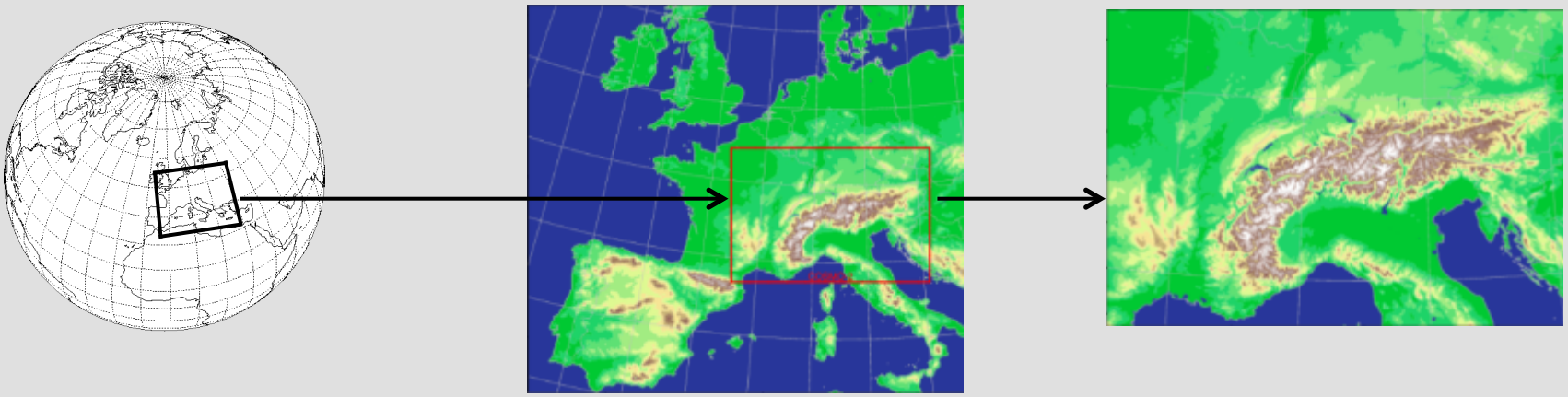
dx = 16 km
2 x per day
10 day forecast

COSMO-7

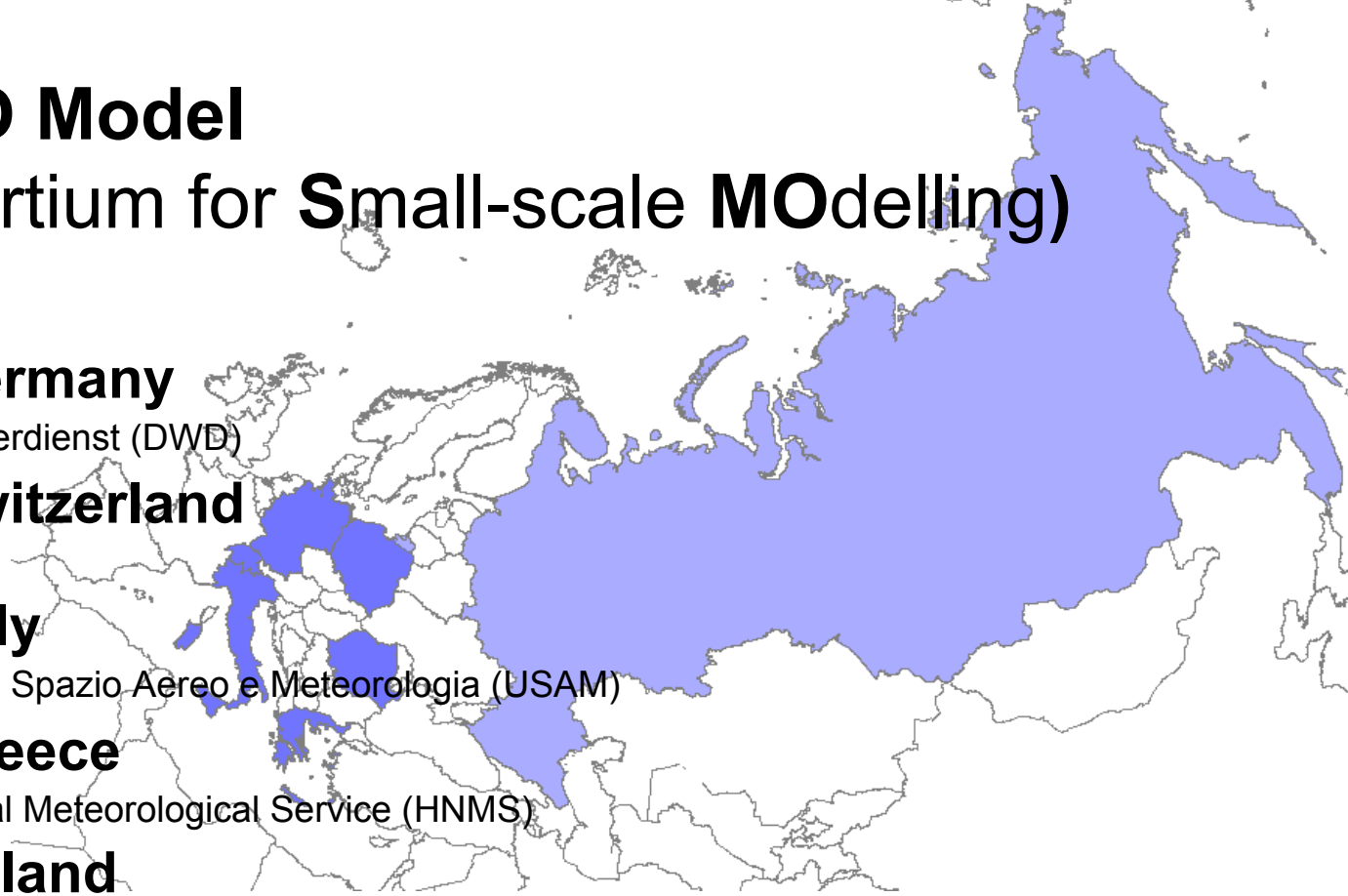
dx = 6.6 km
3 x per day
3 day forecast

COSMO-2

dx = 2.2 km
8 x per day
33 h forecast



COSMO Model
(COnsortium for Small-scale MOdelling)



Germany

Deutscher Wetterdienst (DWD)

Switzerland

MeteoSwiss

Italy

Ufficio Generale Spazio Aereo e Meteorologia (USAM)

Greece

Hellenic National Meteorological Service (HNMS)

Poland

Institute for Meteorology and Water Management (IMGW)

Romania

National Meteorological Administration (NMA)

Russia

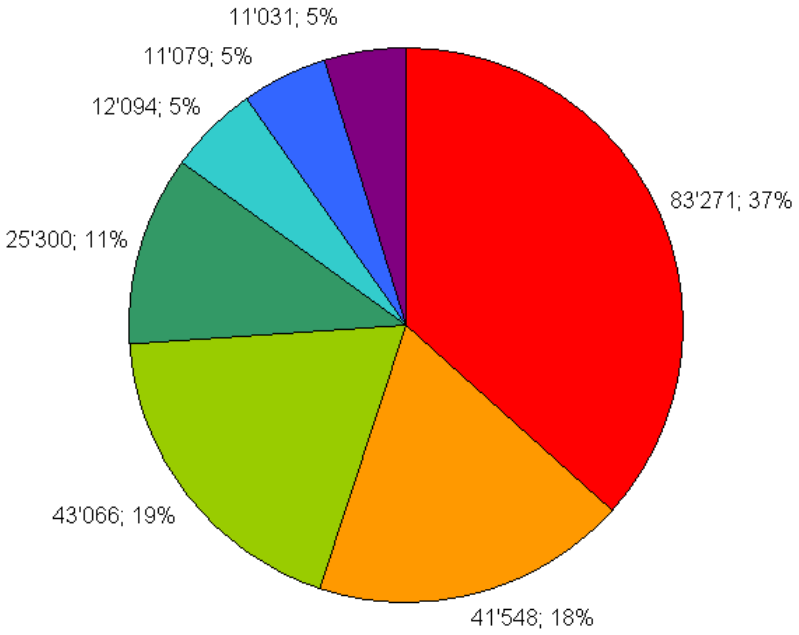
Federal Service for Hydrometeorology and Environmental Monitoring (Roshydromet)

…used at over 70 universities

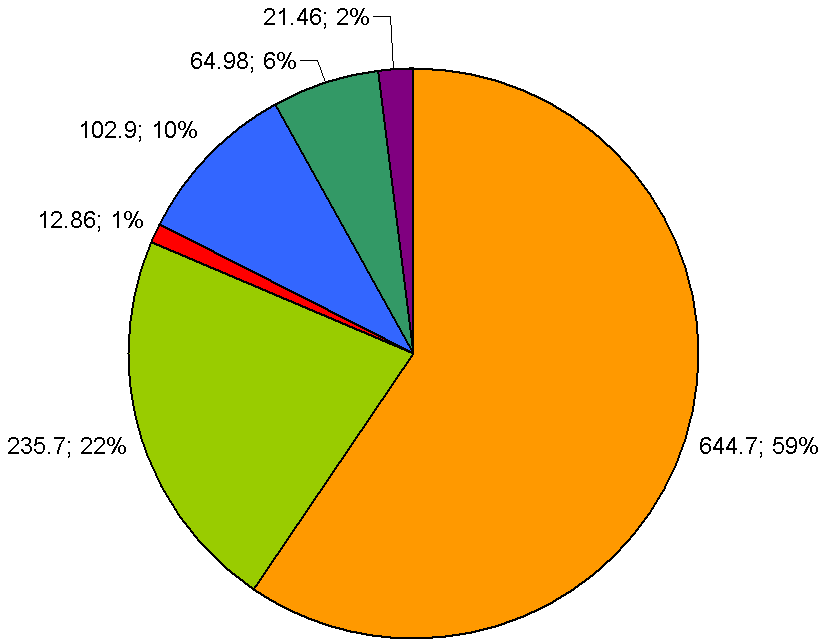
Current trunk version

- Monolithic
- 250’000 lines of Fortran 90 code

% LOC



% Runtime



Dynamics

wind

pressure

temperature

water

density

$$\rho\frac{d\mathbf{v}}{dt} = -\nabla p + \rho\mathbf{g} - 2\boldsymbol{\Omega}\times(\rho\mathbf{v}) - \nabla\cdot(\underline{\mathbf{T}})$$

$$\frac{dp}{dt} = -(c_{pd}/c_{vd})p\nabla\cdot\mathbf{v} + (c_{pd}/c_{vd}-1)Q_h$$

$$\rho c_{pd}\frac{dT}{dt} = \frac{dp}{dt} + Q_h$$

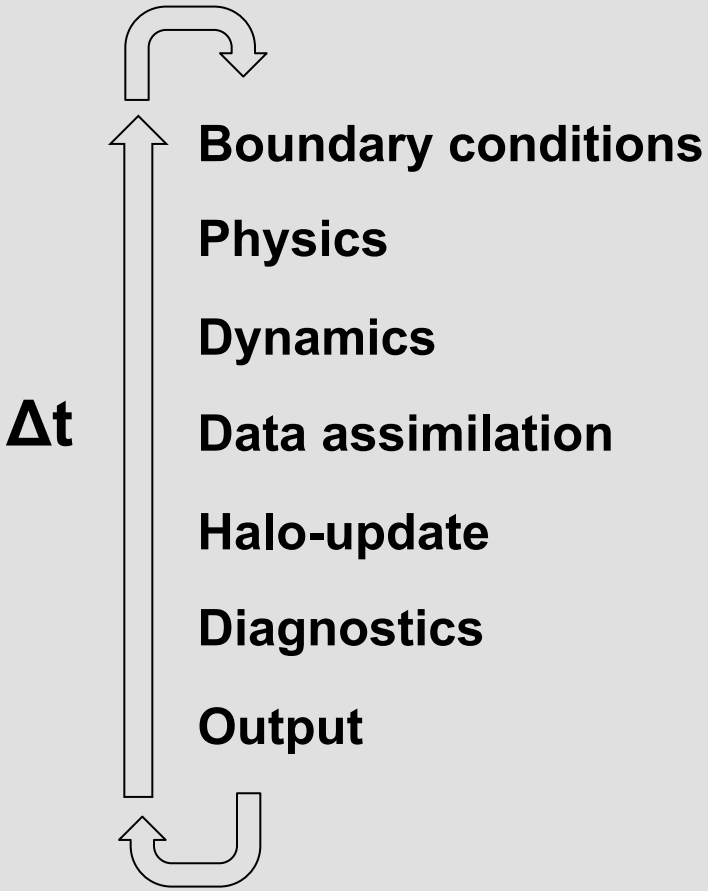
$$\rho\frac{dq^v}{dt} = -\nabla\cdot\mathbf{F}^v - (I^l + I^f)$$

$$\rho\frac{dq^{l,f}}{dt} = -\nabla\cdot(\mathbf{P}^{l,f} + \mathbf{F}^{l,f}) + I^{l,f}$$

$$\rho = p\{R_d(1+(R_v/R_d-1)q^v - q^l - q^f)T\}^{-1}$$

COSMO Workflow

Initialization



Properties

- Sequential workflow
- PDEs
- Structured grid
- Finite differences
- Low compute intensity
- Compact stencils

Cleanup

How to adapt COSMO for current and future hybrid architectures?

Challenges

- **Timescales**
  - Model: ~ 10 years + 20 years
  - Hardware: ~ 4 years
- **Effort and Complexity**
  - > 250’000 LOC
  - Heterogeneous collection of modules
  - Moving target
- **Community code**
  - Single source
  - Performance portability
  - Maintainability
  - Usability

Acceleration with GPUs for COSMO?

- CPUs and accelerators have separate memories
- Transfer of data on each timestep too expensive

*

Part	Time/Δt
Dynamics	**166 ms**
Physics	**56 ms**
Total	**253 ms**

vs

§

Transfer of all prognostic variables

68 ms

All code which touches the prognostic variables on every timestep has to be ported

Porting approach

- **Dynamics**
  - 40k lines, 60% runtime
  - Few developers
  - Performance critical
  - Stencils

Rewrite
- full port
- DSL

- **Rest of code (physics, assimilation, …)**
  - 130k lines, 25% runtime
  - Several developers
  - “Plug-in” from other models
  - Only partly performance critical
  - Point-wise or column-based

Fortran + MPI + “X”
- keep source
- partial port
- directives

The Fortran + MPI + “X” Approach

OpenACC porting

Strategy

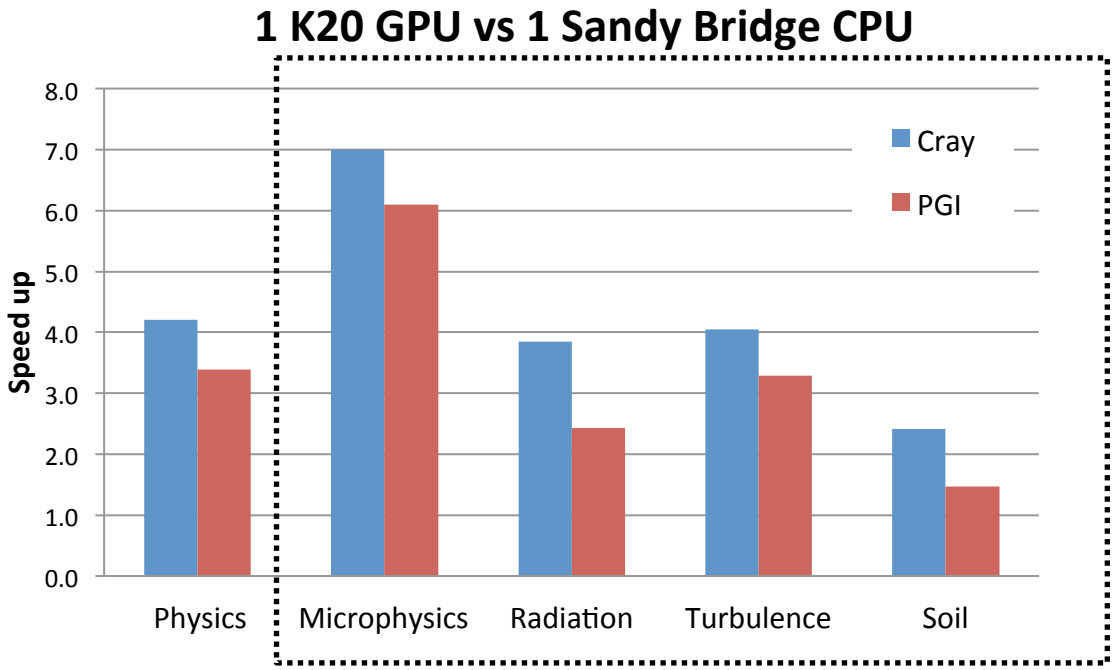
- Parallelization in horizontal (1 thread per column)
- All data resident on GPU
- Avoid data creation as much as possible
- Restructure/optimize time-critical parts
- Use > 1 compiler

Effort

- 60k lines required porting
- 3600 directives

Performance results (Physics)

- Test domain: 263x92x60, 1h simulation.
- Socket to Socket comparison



- Varying optimization effort
- Overall speedup between 3x and 4x (no data transfer)

Fortran + MPI + OpenACC Approach

Pros

- Good performance possible
- Retain original language
- Incremental porting possible
- “Rapid” porting of non time-critical parts
- Very good interaction with compiler builders
- Accessible to non-expert programmers

Cons

- No free lunch!
- Data management is hard
- Single source not always possible
- Performance portability
- Different compiler behavior
- Implementations maturing

Rewrite of dynamics

Strategy

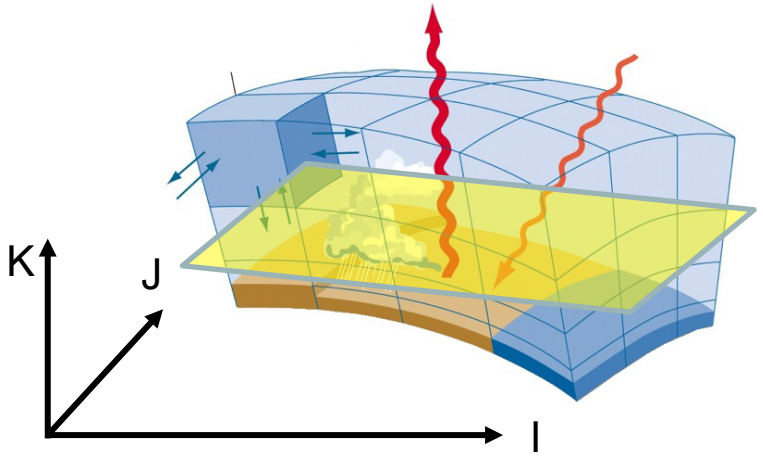
- Code analysis and performance model
- Build re-usable software components

Effort

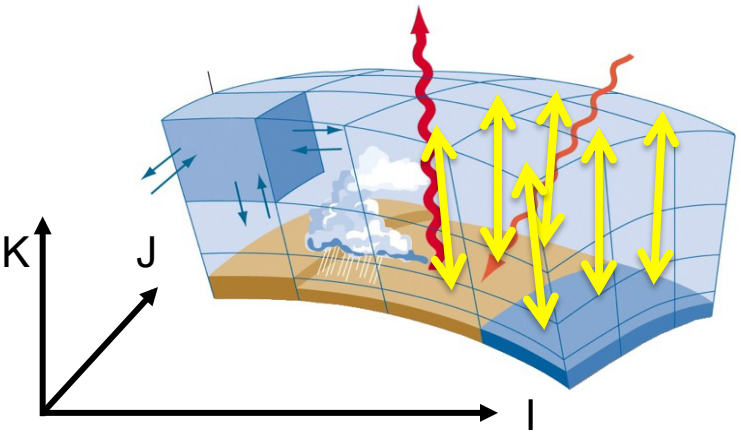
- 40k lines rewritten

Dynamics: Algorithmic motifs

1. **Finite difference stencil computations**
   - Focus on horizontal IJ-plane accesses
   - No loop carried dependencies



2. **Tri-diagonal solves**
   - vertical K-direction stencils
   - Loop carried dependencies in K



Dynamics: Learnings

- **Fortran code is a mix** of mathematical model, numerical discretization, solution algorithm, and hardware dependent implementation details
- Strongly **memory bandwidth bound**
- Optimizations are **hardware dependent** and **increase code complexity**
- Consequences
  - Hard to achieve performance portability with a single source code!
  - Hard to understand, modify and debug
  - Hard to re-use
  - Hard to attract good programmers

Solutions?

- Good compromise (if it exists!)
- Several efficient source codes
- Separate model and algorithm from hardware specific implementation and optimization using a DSL

DSL Approach

- Domain-specific language (DSL)

  A language that exploits domain knowledge for both **productivity** and **efficiency** at the cost of **generality**

- Several flavors
  - Source-to-source translation (Liszt, ATMOL, ICON, PATUS, …)
  - embedded in host language (STELLA, Physis, …)
  - compiler extension (Halide, Pochoir, …)

- **STELLA** (Stencil Loop Language) is a DSL for solving PDEs on structured grids
  - embedded in C++ using template meta-programming

Example: Laplacian

- Operator has two main components
  - Loop-logic defining the stencil application domain and order
  - Stencil defining the operator to be applied

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

STELLA (user code)

```
enum { data, lap };

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

  static void Do()
  {
    lap::Center() =
      -4.0 * data::Center() +
      data::At(iplus1) +
      data::At(iminus1) +
      data::At(jplus1) +
      data::At(jminus1);
  }
};
```

```
IJKRealField lapfield, datafield;
Stencil stencil;

StencilCompiler::Build(
pack_parameters(
    Param<lap, cInOut>(lapfield),
    Param<data, cIn>(datafield)
  ),
  concatenate_sweeps(
    define_sweep<KLoopFullDomain>(
      define_stages(
        StencilStage<Laplace, IJRangeComplete>()
      )
    )
  )
);

stencil.Apply();
```

STELLA (user code)

Stencil

```
enum { data, lap };

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

  static void Do()
  {
    lap::Center() =
      -4.0 * data::Center() +
      data::At(iplus1) +
      data::At(iminus1) +
      data::At(jplus1) +
      data::At(jminus1);
  }
};
```

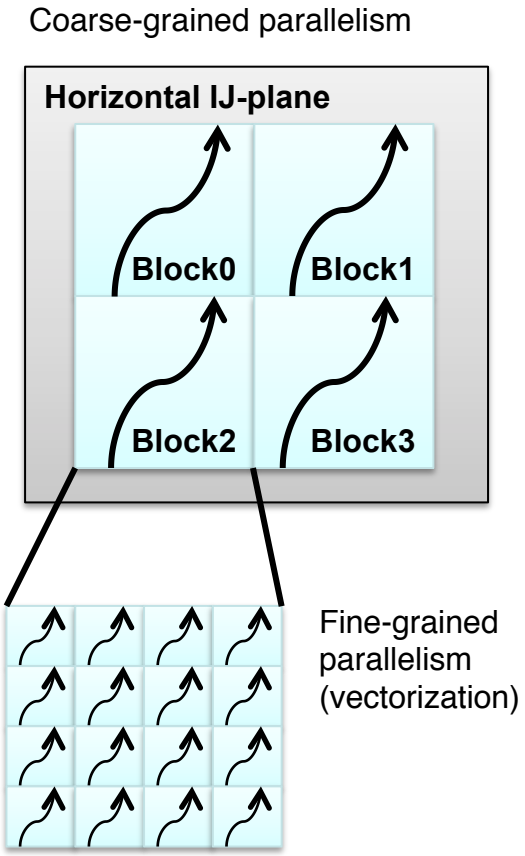
Loop-logic

```
IJKRealField lapfield, datafield;
Stencil stencil;

StencilCompiler::Build(
pack_parameters(
    Param<lap, cInOut>(lapfield),
    Param<data, cIn>(datafield)
  ),
  concatenate_sweeps(
    define_sweep<KLoopFullDomain>(
      define_stages(
        StencilStage<Laplace, IJRangeComplete>()
      )
    )
  )
);

stencil.Apply();
```

STELLA backends

A single switch chooses the STELLA backend

- The same high-level user code can be compiled using different backends
- **CPU (x86 multi-core)**
  - kij-storage
  - ij-blocking
  - Coarse: OpenMP threads
  - Fine: vectorization by compiler
- **GPU (NVIDIA)**
  - ijk-storage
  - Coarse: CUDA thread blocks
  - Fine: CUDA threads
  - software managed caching



DSL Approach

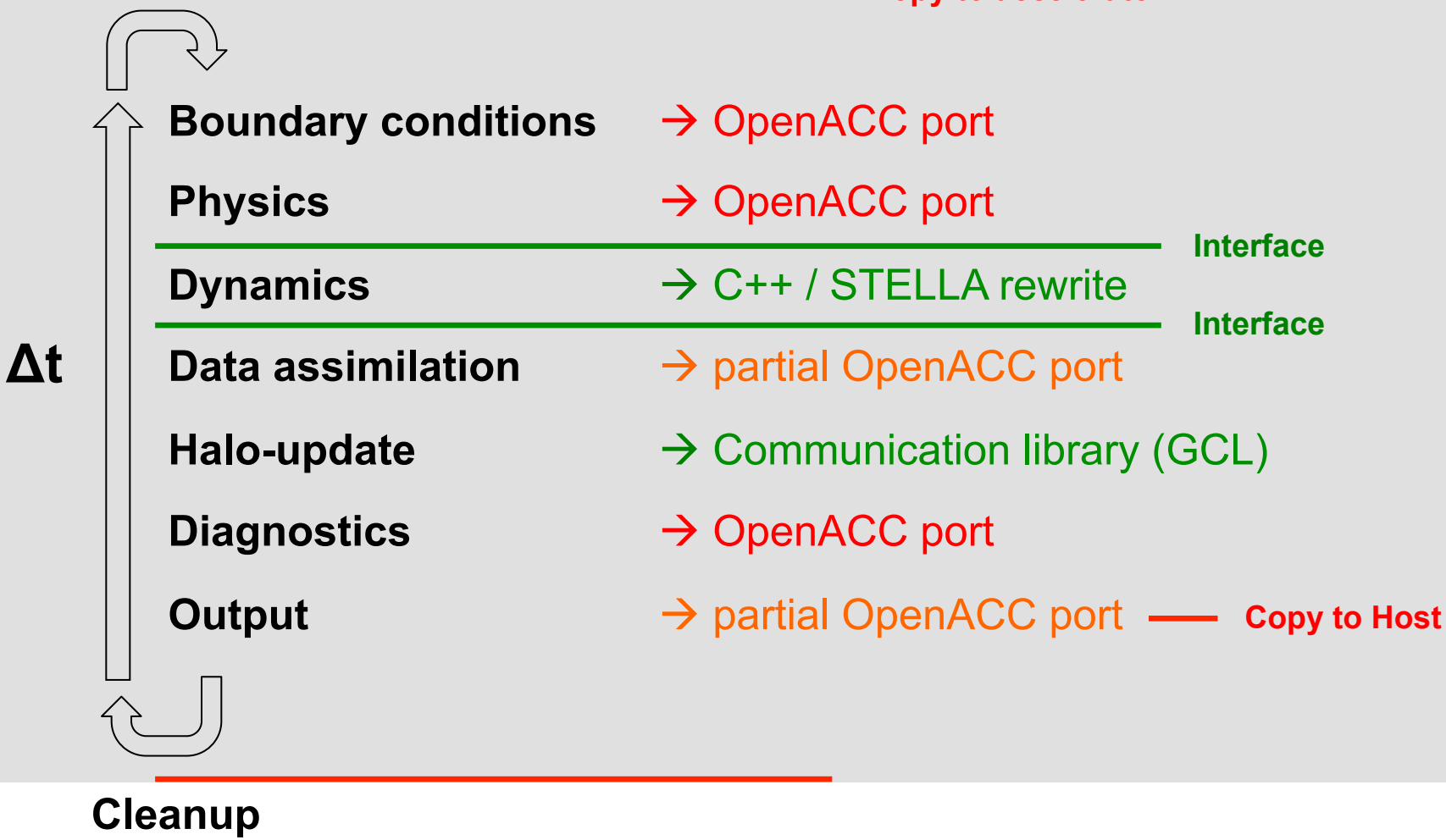
Pros

- Performance portability
- Separation of concerns (algorithm / implementation strategy)
- Single user code
- DSL enforces coding conventions
- Accessible to non-expert programmers

Cons

- Paradigm shift
- Maintenance of DSL library
- Backends require expert knowledge
- Fixed set of features
- Loss of abstraction painful

Integration



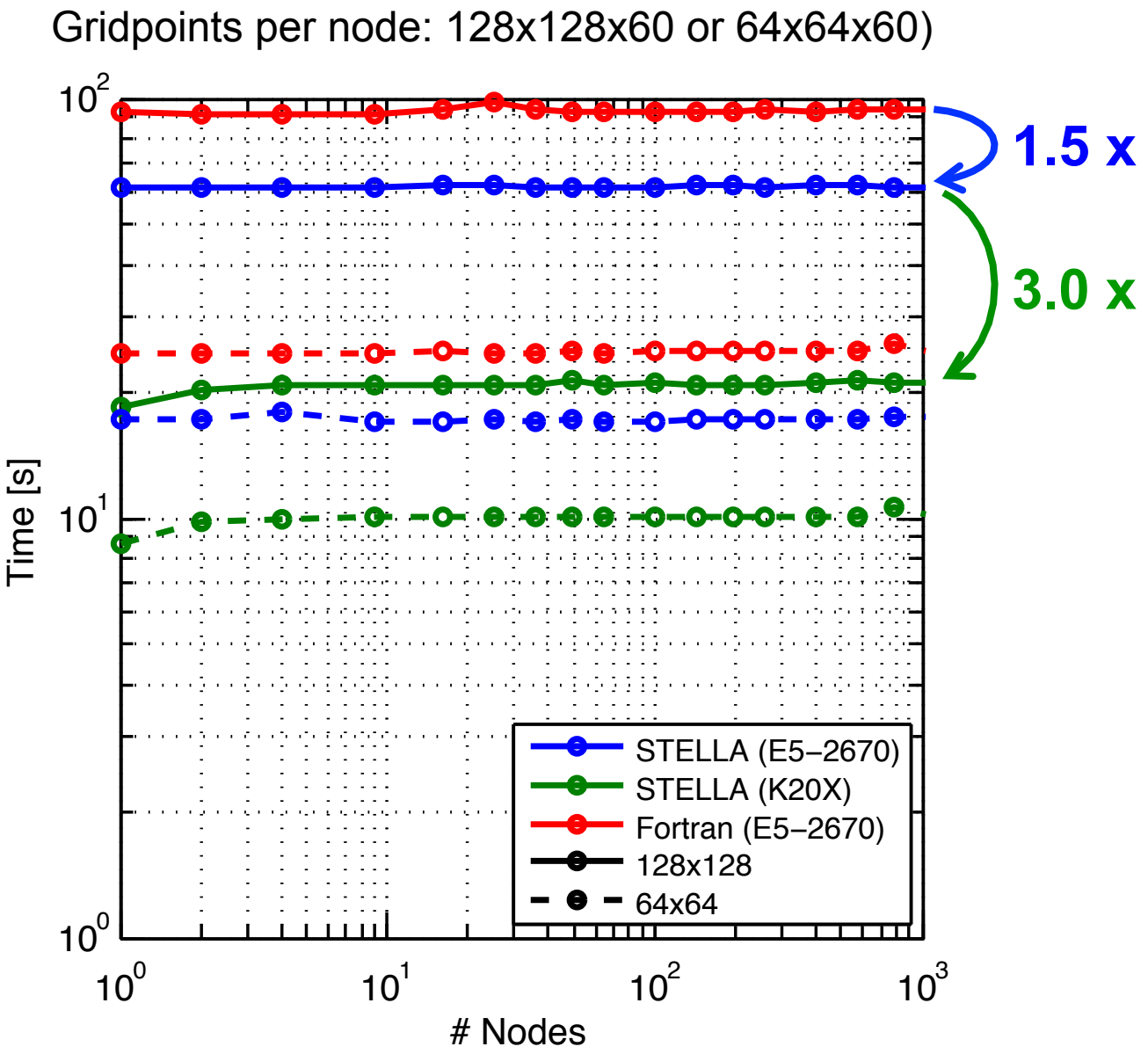
Integration and Testing

- **Interoperability** of key importance
  - Fortran & C++
  - OpenACC & CUDA
  - Middleware (STELLA, GCL) & legacy code
  - Compiler 1 & Compiler 2

- **Bit-reproducibility** useful
  - different data distributions
  - different hardware architectures

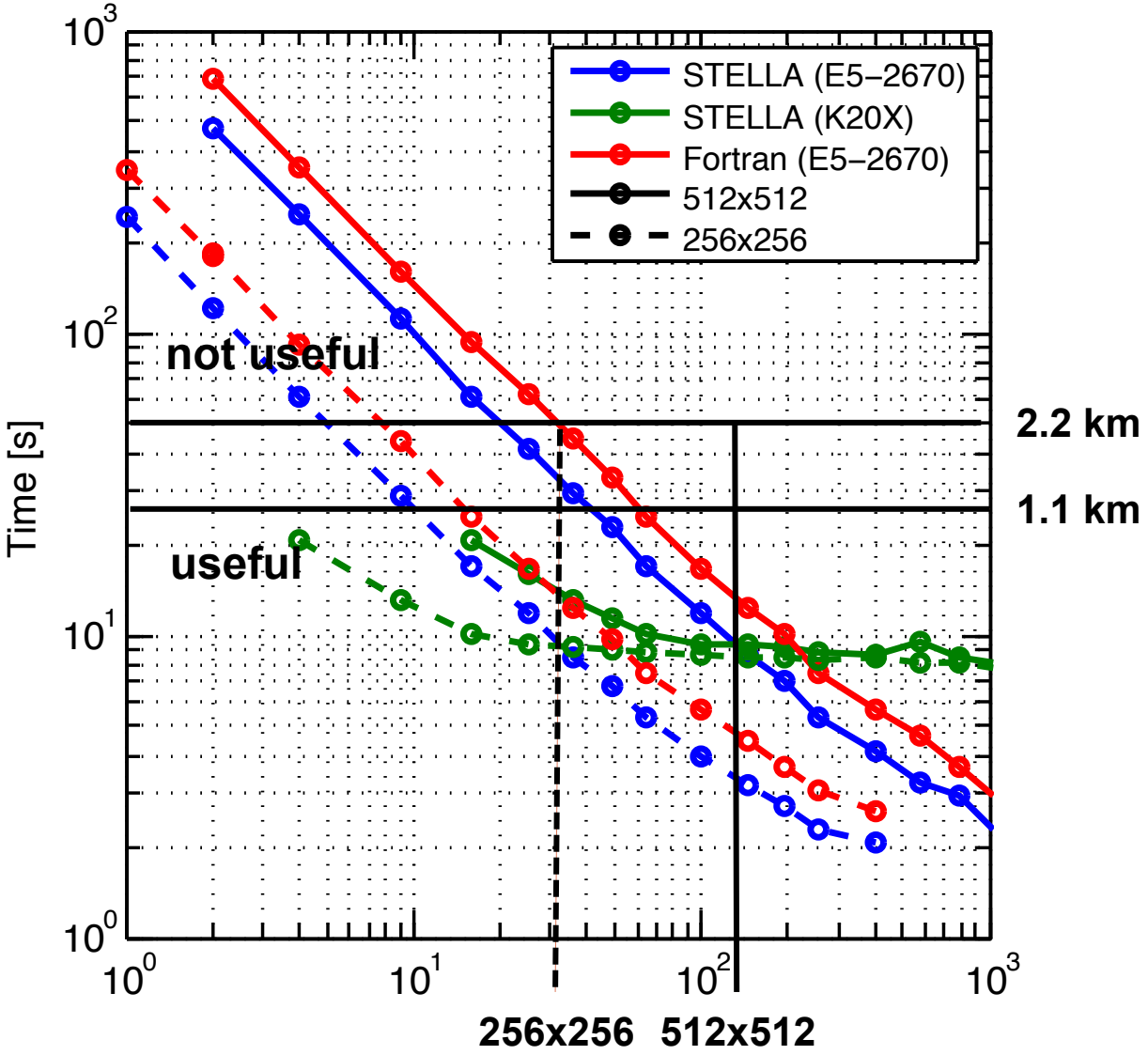
Performance results on Piz Daint

Weak scaling (Piz Daint, no I/O)



Strong scaling (Piz Daint, no I/O)

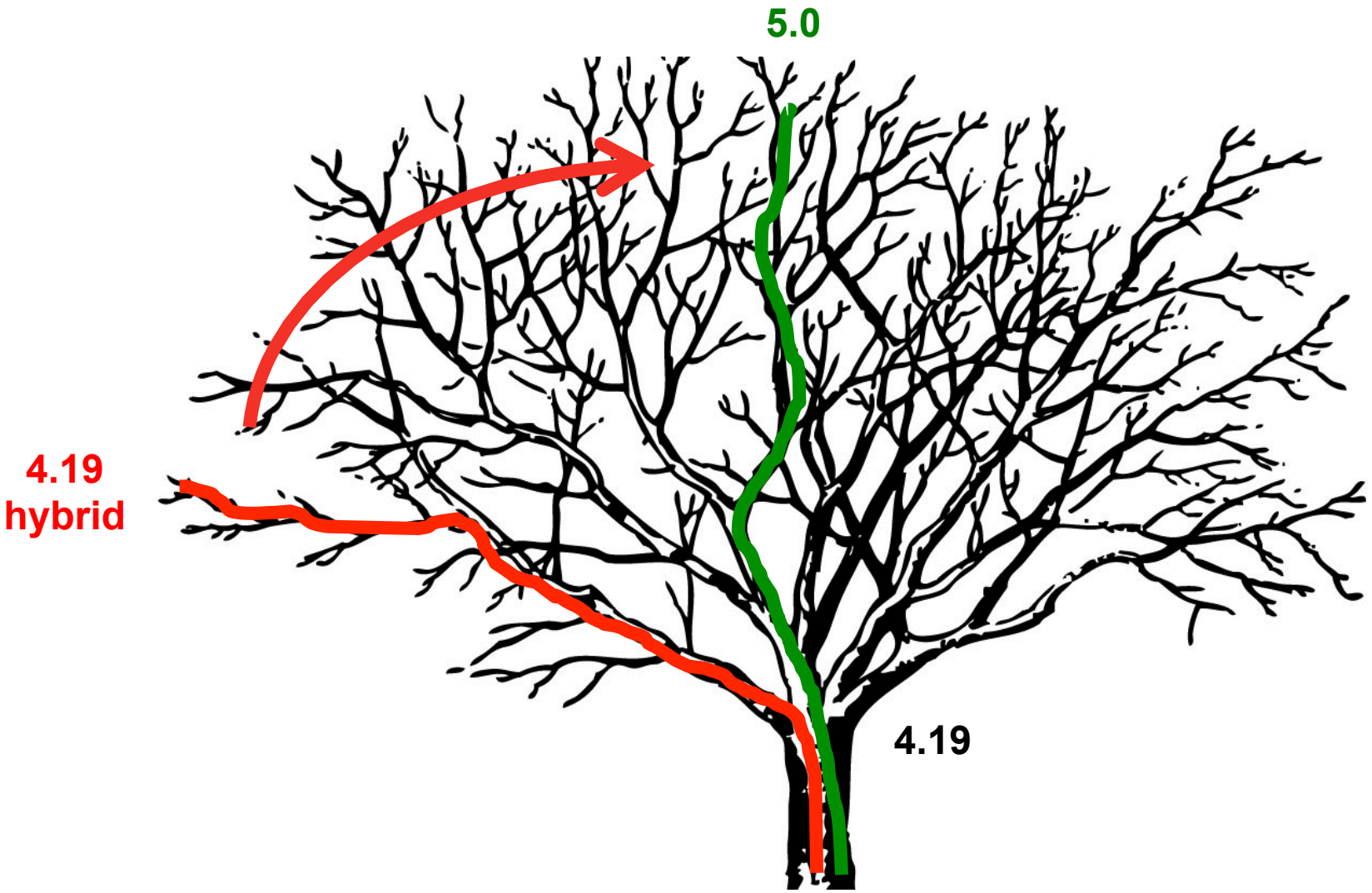
Total number of gridpoints: 512x512x60 or 256x256x60)



Conclusions

- Changing hardware architectures require to continually rewriting/refactoring our codes
- Model codes are growing in length and complexity (changing a wheel on a running car)
- No consensus has emerged to deliver both high performance with high programmer productivity
- **Developing re-usable, performance portable and highly optimized middleware can help**
- **DSLs are an interesting way forward**
- Several efforts in weather/climate (ATMOL, ICON DSL, STELLA, …)

Ongoing



Merge back to trunk until end of 2014

Next steps

- **PASC GridTools Project**
  Generalize STELLA to grids on a sphere (global models) and other science domains (e.g. seismology)
- **DSL for physical parametrizations?**
  - Loop reordering
  - Data structures
  - Demotion/Promotion of arrays

Thank you!

Cloud-Resolving Simulation of Winter Storm Kyrill

David Leutwyler, Oliver Fuhrer, Christoph Schär, Andrea Arteaga, Isabelle Bey, Mauro Bianco, Ben Cumming, Tobias Gysi, Xavier Lapillonne, Daniel Lüthi, Carlos Osuna, Anne Roches, Thomas Schulthess

ETH zürich

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

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

CSCS
Centro Svizzero di Calcolo Scientifico
Swiss National Supercomputing Centre

C2SM
Center for Climate Systems Modeling

Software Architecture

“Algorithm”
Little or no hardware dependent code
Maintained by domain-scientist

Separation of concerns

“Middleware”
Re-usable software components
Domain-specific
Inter-operable
Performance portable
Performance optimizable
Maintained by computer-scientist

Separation of concerns