Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

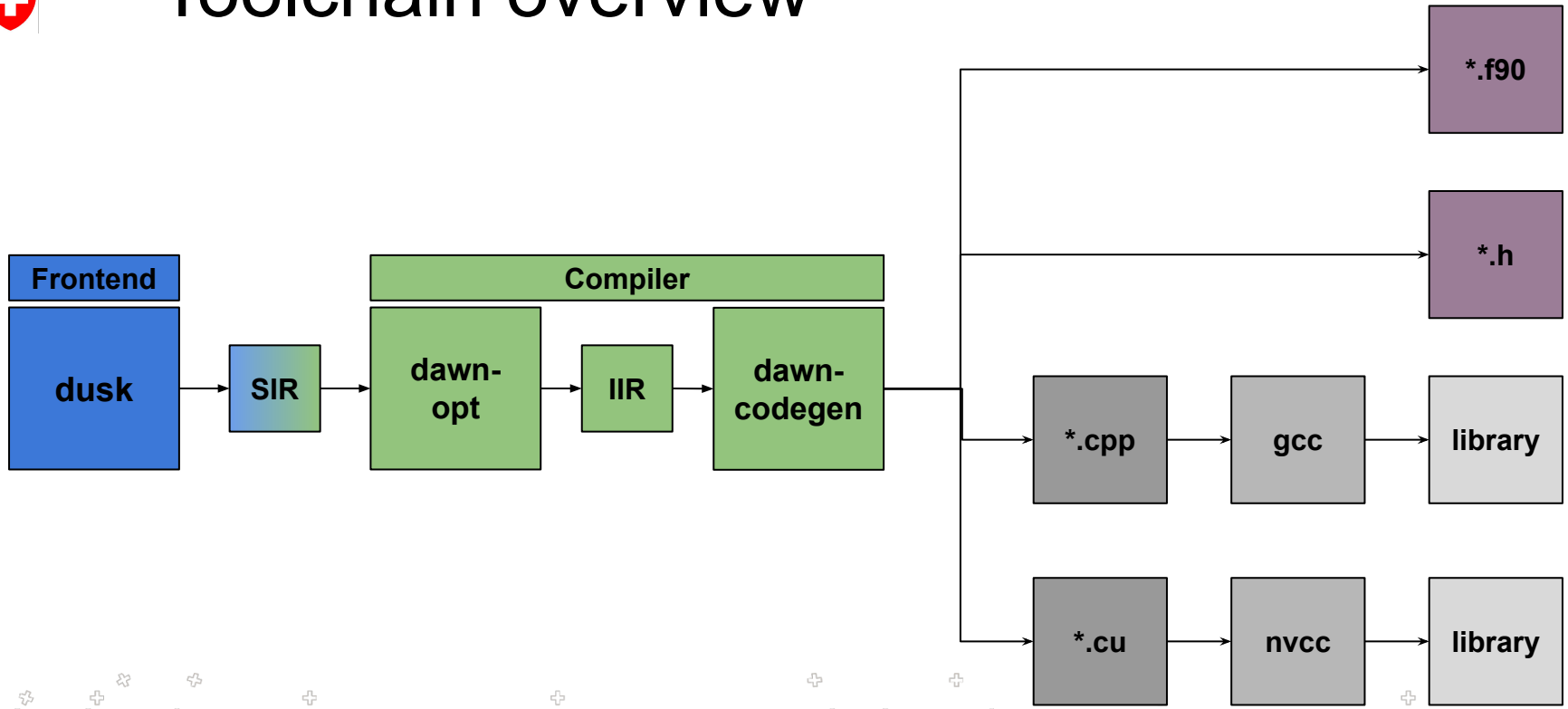
Swiss Confederation

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

Integration

How to use the toolchain and how to interoperate with existing model code (in our case ICON's)

Toolchain overview



CLI Tool: dusk-front

Usage: `dusk-front [-h] input_script.py`

Translates a Python embedded DSL (Dusk) script into SIR (in JSON format) and prints it to stdout. Errors and warnings go to stderr.

Output usually piped to other tools or dumped to file:

```
dusk-front stencils.py | ...
dusk-front stencils.py > stencils.sir
```

CLI Tool: dawn-opt

Usage:

```
dawn-opt [OPTIONS...] [SIR or IIR file. If unset, reads from stdin]
```

Optimizer component of Dawn. Prints code-generation ready IIR (JSON format) to stdout, starting from either SIR or intermediate IIR.

Many options to tune optimizations and to dump intermediate states or analyses.

Add `--default-opt` to enable most of the optimizations, otherwise unoptimized (but parallel-valid) IIR is generated.

Also here output usually piped or dumped to file:

```
... | dawn-opt --default-opt | ...
... | dawn-opt > stencils.iir
```

CLI Tool: dawn-codegen

Usage:

```
dawn-codegen [OPTIONS...] [IIR file. If unset, reads from stdin]
```

Example:

```
... | dawn-codegen -b cuda-ico -o stencils_cuda.cu
... | dawn-codegen -b naive-ico -o stencils_cxx.cpp
```

Also allows fine-tuning for a specific GPU architecture.

Two important options of dawn-codegen are presented in the next slides...

Generating C header

On top of the `.cpp` / `.cu` implementation code, `dawn-codegen` can generate a C header containing the run functions' declarations of each stencil, through the `--output-c-header` option.

Example:

```
... | dawn-codegen -b cuda-ico -o stencils.cu
--output-c-header stencils.h
```

Generating C header

stencils.h:

```
extern "C" {
  double run_my_stencil(dawn::GlobalGpuTriMesh  *mesh, int k_size,
                        ::dawn::float_type  *field1, ::dawn::float_type  *field2, ...);
}
```

stencils.cu:

```
...
double run_my_stencil(dawn::GlobalGpuTriMesh  *mesh, int k_size,
                      ::dawn::float_type  *field1, ::dawn::float_type  *field2, ...)
{ ... }
```

Generating F90 interface

It can also generate Fortran 90 compatible interfaces (based on the `ISO_C_BINDING` module) with the `--output-f90-interface` option.

Example:

```
... | dawn-codegen -b cuda-ico -o stencils_cuda.cu
--output-f90-interface stencils_cuda.f90
```

Generating F90 interface

stencils.f90:

```
module stencils
use, intrinsic :: iso_c_binding
implicit none

interface
   real(c_double) function run_my_stencil(mesh,k_size,field1,field2)  bind(c)
     use, intrinsic :: iso_c_binding
     type(c_ptr), value, target :: mesh
     integer(c_int), value, target :: k_size
     real(c_double), dimension(:,:), target :: field1
     real(c_double), dimension(:,:), target :: field2
   end function
 end interface
end module
```

CLI Toolchain

All together the tools are used in the following manner:

```
dusk-front stencils.py | dawn-opt | dawn-codegen -b
cuda-ico -o stencils_cuda.cu --output-c-header
stencils_cuda.h --output-f90-interface stencils_cuda.f90
```

Alternatively, if one doesn't have particular tuning needs, the `dusk` end-to-end tool simplifies the usage:

```
dusk -b ico-cuda stencils.py
```

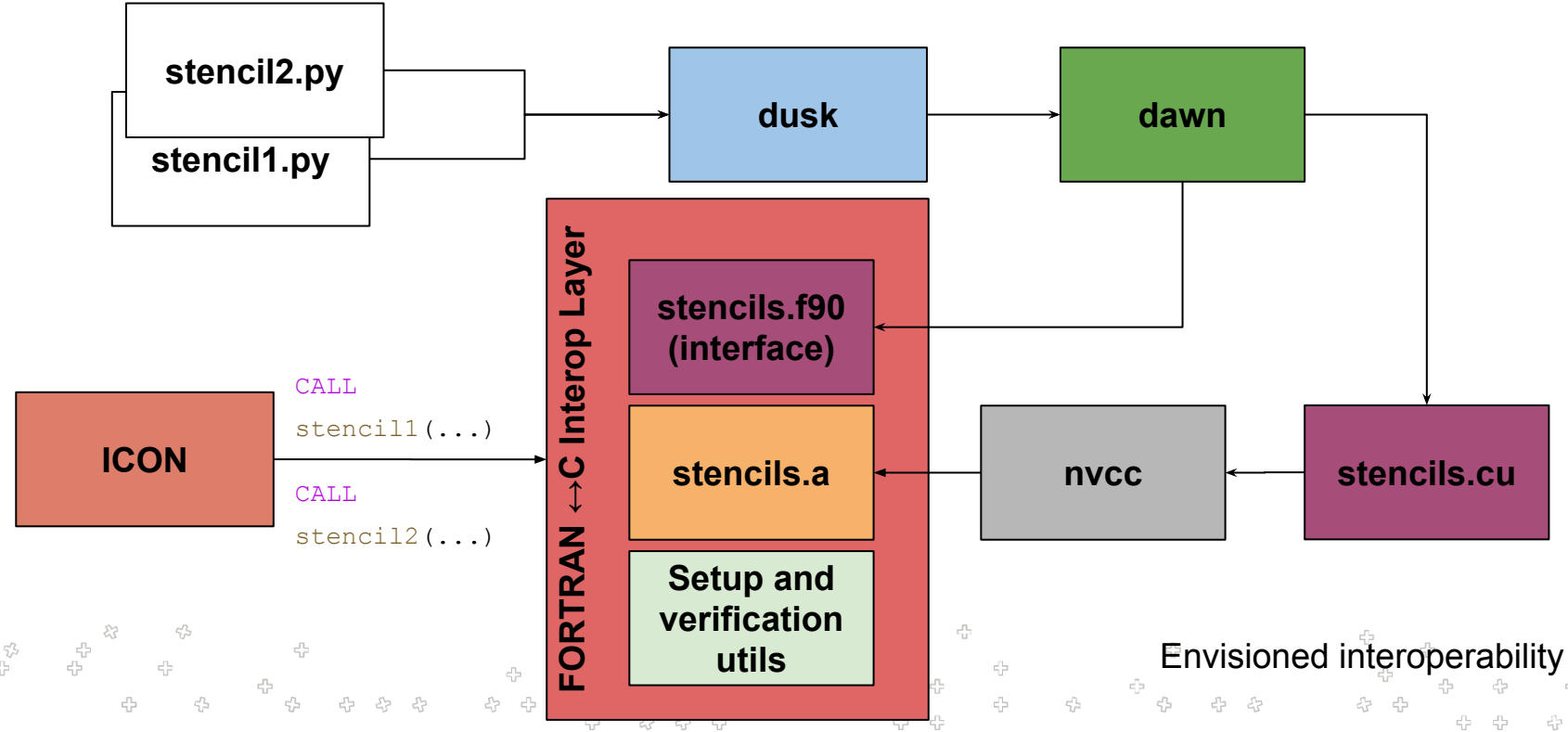
Python APIs

The same steps shown above can be done also through our Python APIs:

```
dusk.transpile()
dawn4py.lower_and_optimize()
dawn4py.optimize()
dawn4py.codegen()
dawn4py.compile()
...
```

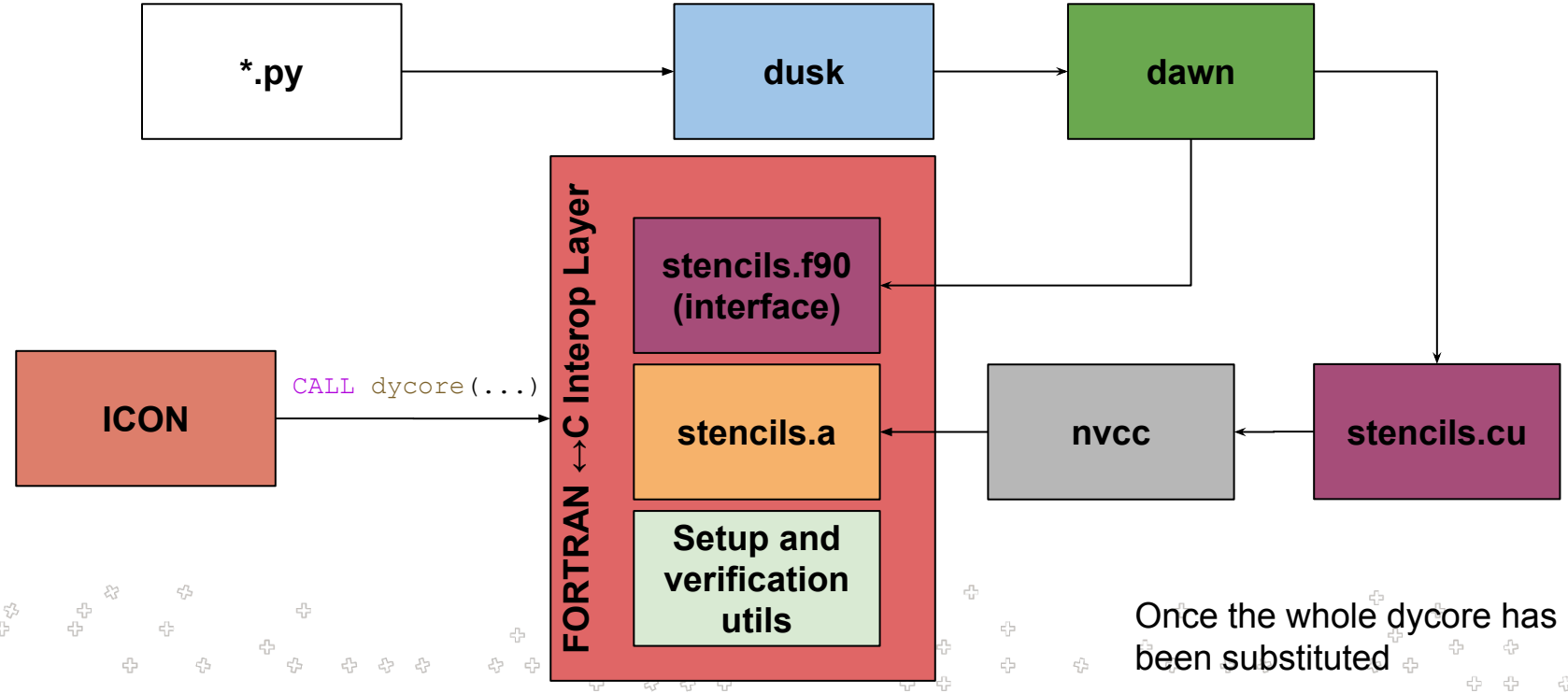
This allows to do everything within the same python shell.

Interoperability: Overview



Envisioned interoperability

Interoperability: Overview



Once the whole dycore has been substituted

Interoperability: Example

```
SUBROUTINE vn_adv_horizontal(
            ... )
! <SNIP>
!==========================================
! Calculate the gradient of kinetic energy,
! and accumulate velocity tendency
!==========================================

 CALL grad_fd_norm( ... )
```

```
SUBROUTINE grad_fd_norm( ... )

   !$ACC PARALLEL
   !$ACC LOOP GANG
   DO je = i_startidx, i_endidx
     !$ACC LOOP VECTOR
     DO jk = slev, elev
      grad_norm_psi_e(je,jk,jb) =  &
         &  ( psi_c(iidx(je,jb,2),jk,iblk(je,jb,2)) - &
         &    psi_c(iidx(je,jb,1),jk,iblk(je,jb,1)) )  &
         &  * inv_dual_edge_length(je,jb)
     ENDDO
   END DO
   !$ACC END PARALLEL
 END DO
```

Interoperability: Example

```
SUBROUTINE vn_adv_horizontal(
            ... )
! <SNIP>
!==========================================
! Calculate the gradient of kinetic energy,
! and accumulate velocity tendency
!==========================================

 CALL grad_fd_norm( ... )
```

Stencil is rewritten in dusk

```
grad_norm_psi_e = sum_over(
      Cell > Edge,
      psi_c,
      weights=[1/lhat, -1/lhat]
)
```

Interoperability: Example

Code is generated using dawn

```
SUBROUTINE vn_adv_horizontal(
            ... )
! <SNIP>
!==========================================
! Calculate the gradient of kinetic energy,
! and accumulate velocity tendency
!==========================================

 CALL grad_fd_norm( ... )
```

Interface

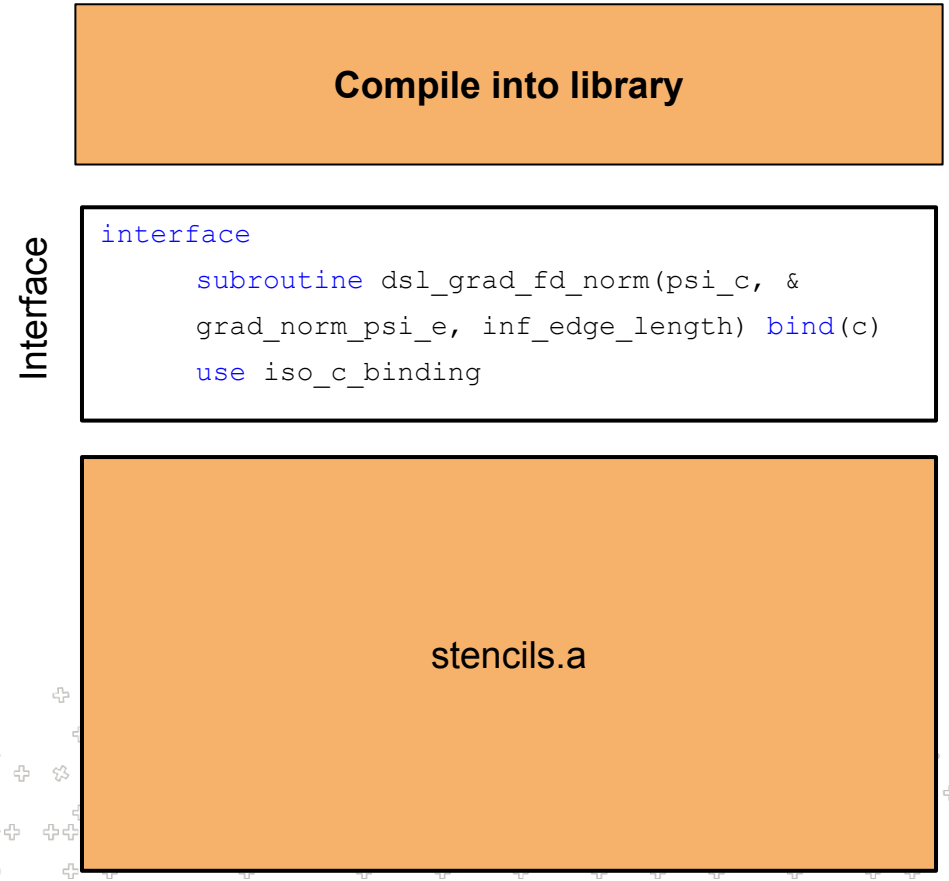
```
interface
      subroutine dsl_grad_fd_norm(psi_c, &
      grad_norm_psi_e, inf_edge_length) bind(c)
      use iso_c_binding
```

Implementation

```
void dsl_grad_fd_norm(
  double* psi_c,
  double* grad_norm_psi_e,
  double* inf_edge_length) {
 for(int k = 0; k < k_size; ++k) {
   for(auto const& loc : getEdges(m_mesh)) {
     for(auto inner_loc :
       grad_norm_psi_e(loc, k) = reduce(...
```

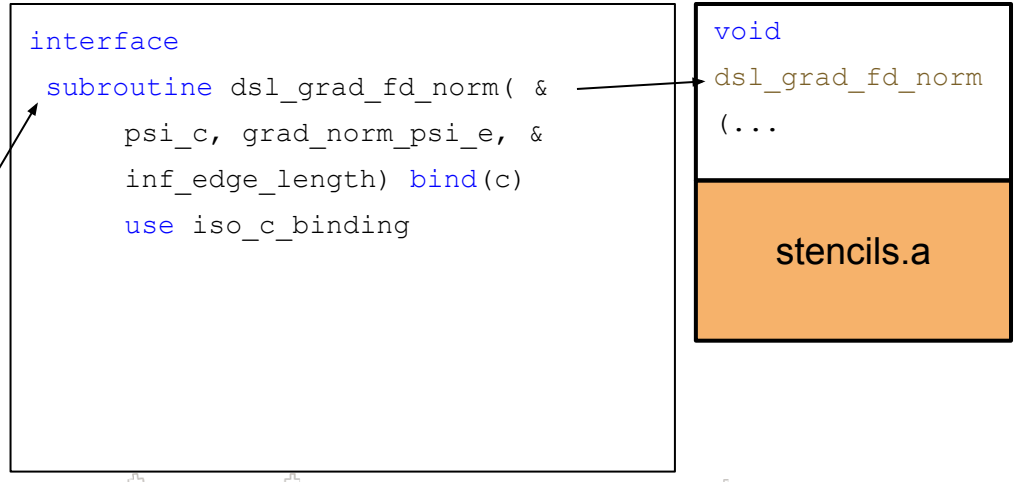
Interoperability: Example

```
SUBROUTINE vn_adv_horizontal(
           ... )
! <SNIP>
!==========================================
! Calculate the gradient of kinetic energy,
! and accumulate velocity tendency
!==========================================

 CALL grad_fd_norm( ... )
```



Interoperability: Example

```
SUBROUTINE vn_adv_horizontal(
           ... )
! <SNIP>
!==========================================
! Calculate the gradient of kinetic energy,
! and accumulate velocity tendency
!==========================================

 CALL dsl_grad_fd_norm( psi_c,
                        grad_norm_psi_e,
                        inf_edge_length )
```



Interoperability: Fields

The model (ICON) imposes a specific format of fields.

A requirement is to ensure that the generated code is going to be compatible.

There are at least two valid options:

- Do a format conversion while switching from model code to generated code and viceversa.
- Allow parametrization of the backend in order to produce code that is already compatible.

We are going along the second way, betting on the advantage of not having a considerable overhead to deal with.

DSL-ifying ICON

It’s worth mentioning that we aim at progressively substituting all the *dycore* stencils of the **OpenACC**-accelerated version of ICON.

Fields are already in GPU memory when switching to DSL.

One challenge is to be able to verify the validity of the computations that we substitute as we proceed.

It’s also very interesting to monitor the overall performance (time elapsed) of the dycore while we progress.

We are devising a process that allows to fulfil both requirements within the same instance of the DSL-ified ICON code.

Q&A

Questions?

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

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

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