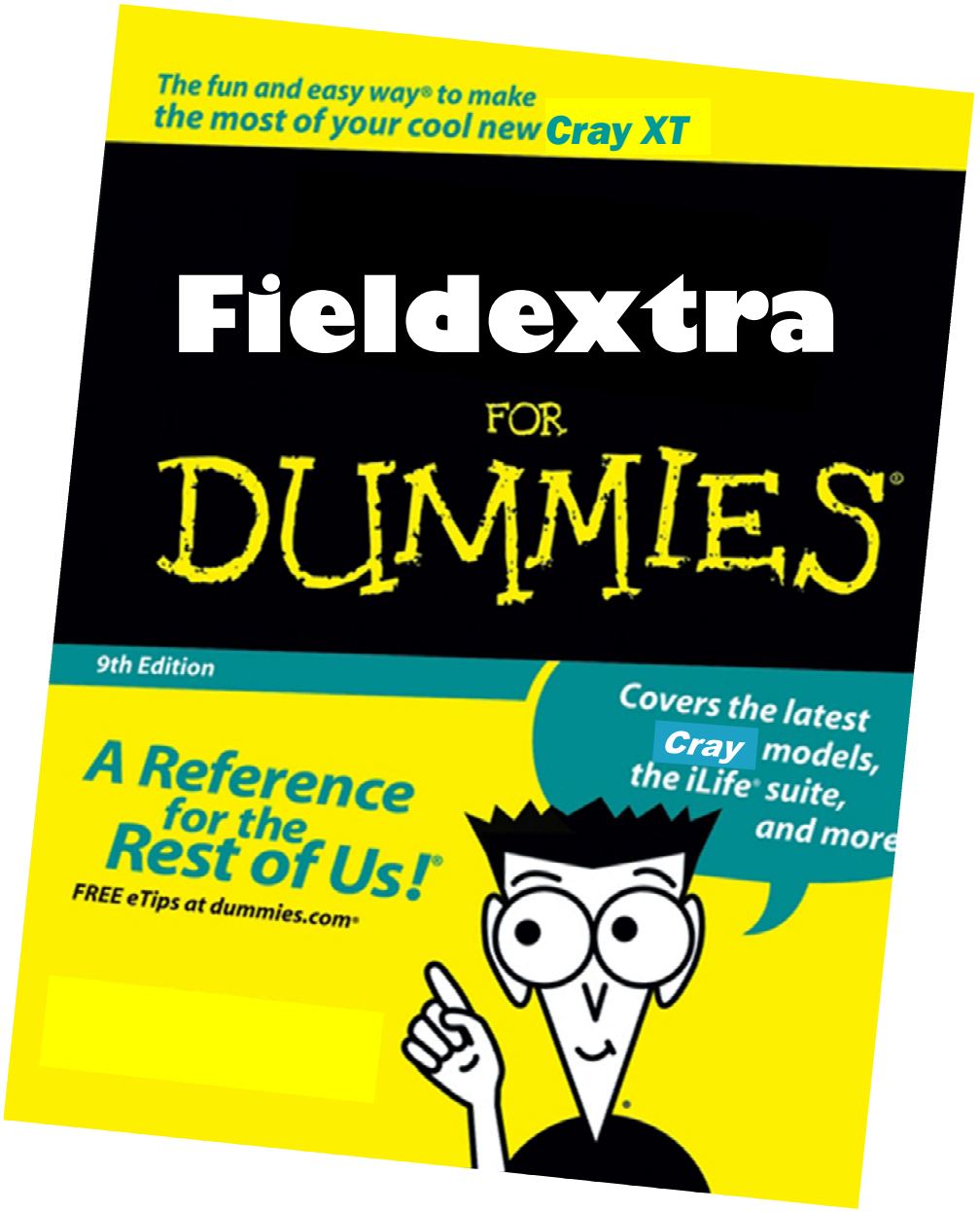
Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

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

CONSORTIUM FOR SMALL SCALE MODELING
COSMO



JM. Bettems &
P. Baumann
MeteoSwiss
February 2022

fieldextra v14.0.0

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

bettems@tsa: fieldextra **control_file**

fieldextra is the fortran program executable
control_file contains the namelist defining the program behaviour

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

Control_file

```
… HEADER …

&Process
  in_file = "./support/input/cosmo-e/000/1fff<DDHH>0000"
  tstart  = 3,  tstop = 9,  tincr = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG",  out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist  = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /


&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/


&Process out_field="TOT_PREC" /
```

Available in ./cookbook/meteogram_precipitation.nl

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

Define input and output characteristics, define domain subset

```
&Process
  in_file = "./support/input/cosmo-e/000/1fff<DDHH>0000"
  tstart  = 3,  tstop = 9,  tincr = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG",  out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist  = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /
```

```
&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/


&Process out_field="TOT_PREC" /
```

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

```
&Process
  in_file = "./support/input/cosmo-e/000/1fff<DDHH>0000"
  tstart  = 3,  tstop = 9,  tincr = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG",  out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist  = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /
```

Define fields to collect

```
&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
```

```
&Process out_field="TOT_PREC" /
```

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

```
&Process
  in_file = "./support/input/cosmo-e/000/1fff<DDHH>0000"
  tstart  = 3,  tstop = 9,  tincr = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG",  out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist  = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /

&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/

&Process out_field="TOT_PREC" /
```

Define operations to apply on collected fields (large choice of operators available)

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

```
&Process
  in_file = "./support/input/cosmo-e/000/1fff<DDHH>0000"
  tstart  = 3,  tstop = 9,  tincr = 3,
  out_file = "./support/results/meteogram_precipitation.txt"
  out_type = "METEOG",  out_type_fmt = "f71_dh_prec",
  out_type_text1 = "Precipitation rain+snow in the last 3 hours mm : mean over 5 gridpoints"
  locgroup = "nat"
  loclist  = "GVE", "DOL", "FRE", "NEU", "CDF", "CHA", "CGI", "PUY", "PAY" /


&Process in_field="RAIN_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="RAIN_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_GSP", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
&Process in_field="SNOW_CON", hoper = "c5", toper = "delta,3,hour" , toper_mask = "lead_time>3"/
```

```
&Process out_field="TOT_PREC" /
```

Define fields to compute
(refers to a fieldextra procedure, easily extensible)

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

- **control_file** contains the namelist defining the program behaviour

header	**&RunSpecification**
	&GlobalResource
	&GlobalSettings
	&ModelSpecification
product definition	**&Process** *(repeated)*

- **external resources**

definition of field names	**dictionary** in &GlobalResource
definition of locations	**location_list** in &GlobalResource

Commented example – Meteogram

Table of total precipitation

3-hourly precipitation sum, mean over 5 grid points, every 3 hours at chosen locations

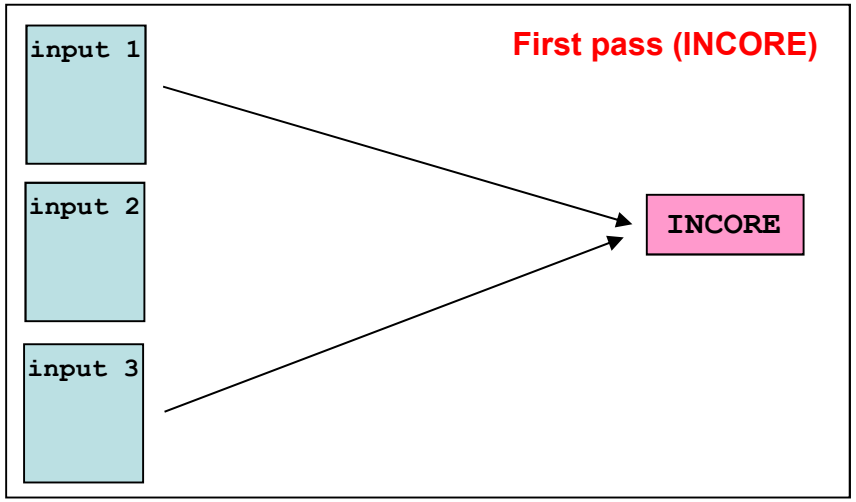
- program **diagnostic** and **profiling**

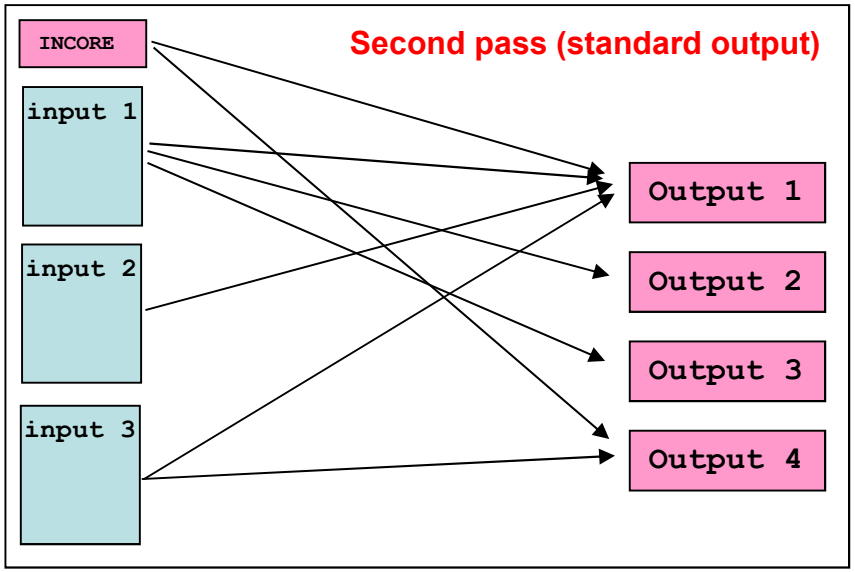
  standard error & output
  file ***fieldextra.diagnostic***

  controlled by the values of verbosity and additional_diagnostic
  verbosity = 'high' *in &RunSpecification*
  additional_diagnostic = .true. *in &RunSpecification*
  additional_profiling = .true. *in &RunSpecification*

Selected topics

Design – Input & output





- **INCORE storage** used to store resources for common operations (see next slide).

- Each **input** is read **once.**
- Storage is allocated **on demand** for each **output**.
- Each input record is evaluated, and dispatched in the corresponding output storage when requested (in memory).

→ **io optimization at the cost of memory usage !**

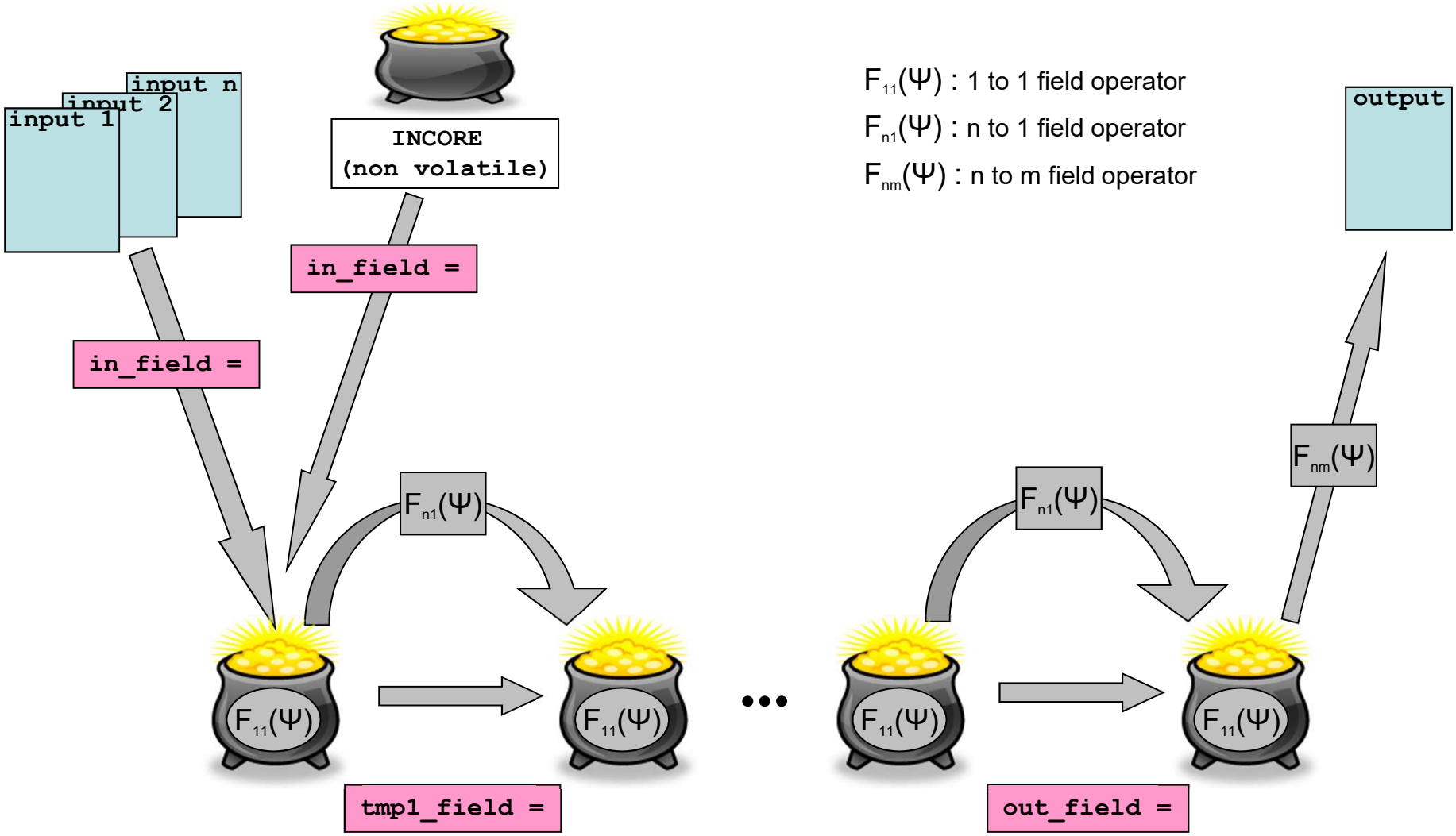
- When all data for some output have been collected, the corresponding data is written on disk and the memory is released.
- For output supporting append mode, data is processed piecewise after reading each related validation time (**‚just on time ' mode**).

Design – Incore storage

- **INCORE** global persistent storage is used to :
  - associate grid points to specified *locations* & *regions*
  - produce grid point *height information* for some output
  - specify model *base grid* when working with staggered fields, or fields defined on a larger domain
  - specify grid for *re-gridding*
  - *merge* and *compare* different fields
  - provide access to *programmatically derived* constant fields (see below)

- **Programmatically derived constant fields** will be available from INCORE storage when HSURF is present:
  - RLAT, RLON (geog. latitude, longitude [deg])
  - CLAT, CLON, ELAT, ELON, VLAT, VLON (geog. latitude, longitude [deg])
  - SWISS_WE / SWISS_SN (swiss coord. [m])
  - BOAGAW_WE / BOAGAW_SN (Gauss-Boaga coord., west sector [m])
  - BOAGAE_WE / BOAGAE_SN (Gauss-Boaga coord., east sector [m])
  - HHL / HFL (COSMO height of model levels [m])
  - T0FL, P0FL, DP0FL (COSMO reference atmosphere)

Design – Iterative processing (1)



Design – Iterative processing (2)

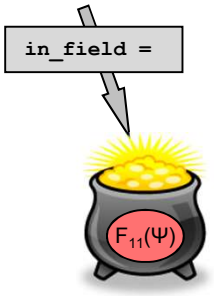
- For each output, define the set of **associated input files**
  - data can only be extracted from this set
  - *INCORE* storage can be part of this set

- **First iteration**: **collect** all necessary input fields
  - all fields must be *unique* (condition can be relaxed with out_duplicate_infield)
  - all field must be defined on a *compatible grid* (cropping & re-gridding are available)
  - fields required in a subsequent iteration must be collected at that stage

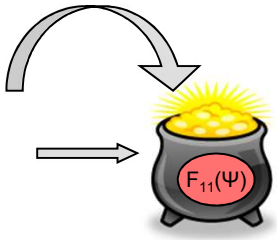
- **Next iterations (repeated up to 6 times)**: **collect** or **compute** fields
  - only data in *previous* recipient are available
  - if fields are not available, they must be computed:
    in this case the *parent fields* must be available in the previous recipient
  - the *main parent* (defined in dictionary or by the type of required operator)
    defines the characteristics of the produced field

- Only the last recipient is available for **output generation**
  - *all* fields available in the last recipient are written in the output file

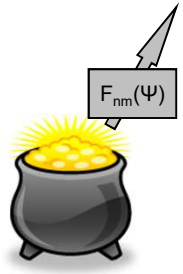
Design – Iterative processing (3)



First iteration:
Each extracted field may be transformed by one or more operators, in the order
regridding (*regrid*), change meta-information (*set_**),
merge (*merge_with*), compare (*compare_with*), lateral transform (*hoper*),
scale/offset, vertical transform (*voper, voper2 … voper5*),
temporal transform (*toper*), local transform (*poper, poper2 … poper5*),
spatial filter (**_filter*), reset identity (*new_field_id*)

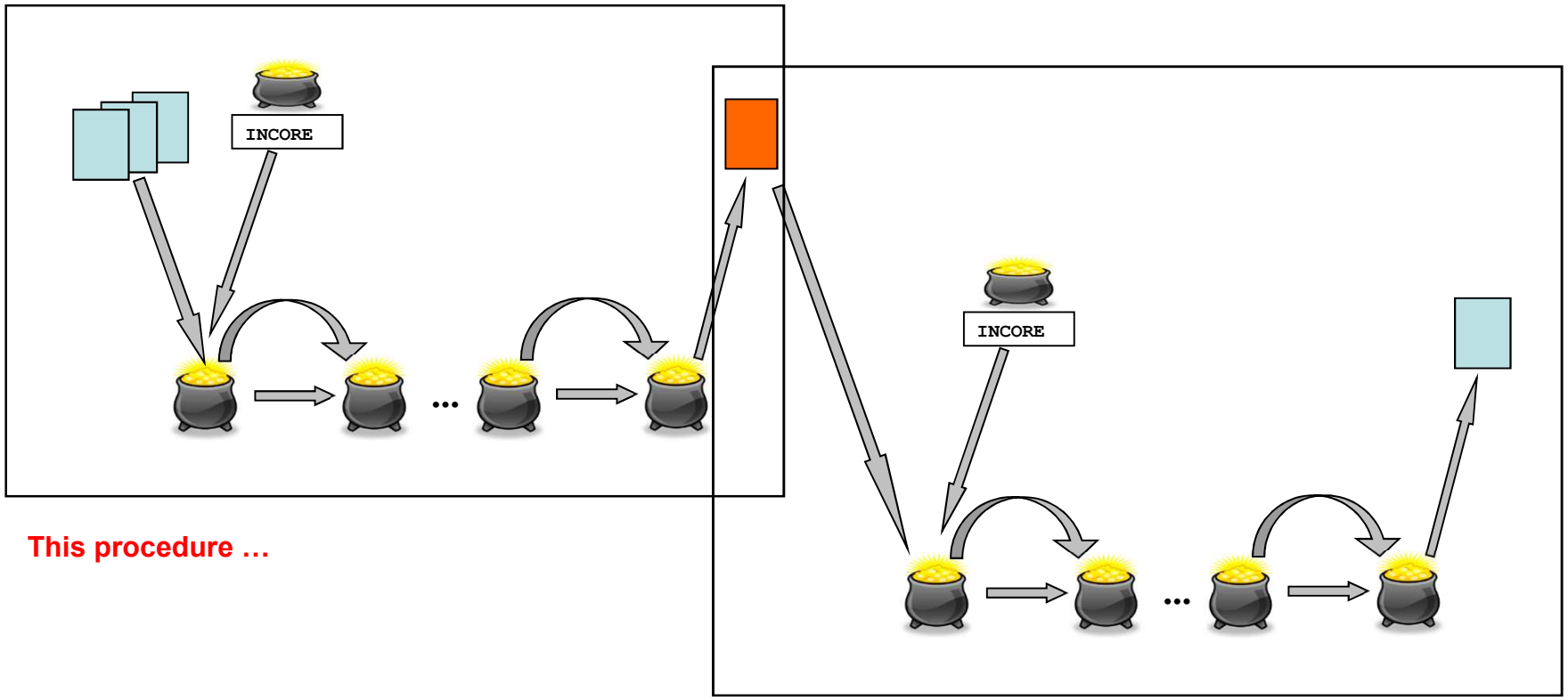


Next iteration(s):
Each extracted field may be transformed by one or more operators, in the order
lateral transform (*hoper*), *scale/offset*,
vertical transform (*voper, voper2 … voper5*), temporal transform (*toper*),
local transform (*poper, poper2 … poper5*), spatial filter (**_filter*),
change meta-information (*set_**) , reset identity (*new_field_id*)



After last iteration:
A last set of global operations may be applied, in the order
n to m operator (*out_postproc_module*), regridding (*out_regrid_target*),
reset meta-info (*set_* …), filter data (*out_filter_**)

Design – Iterative processing (4)

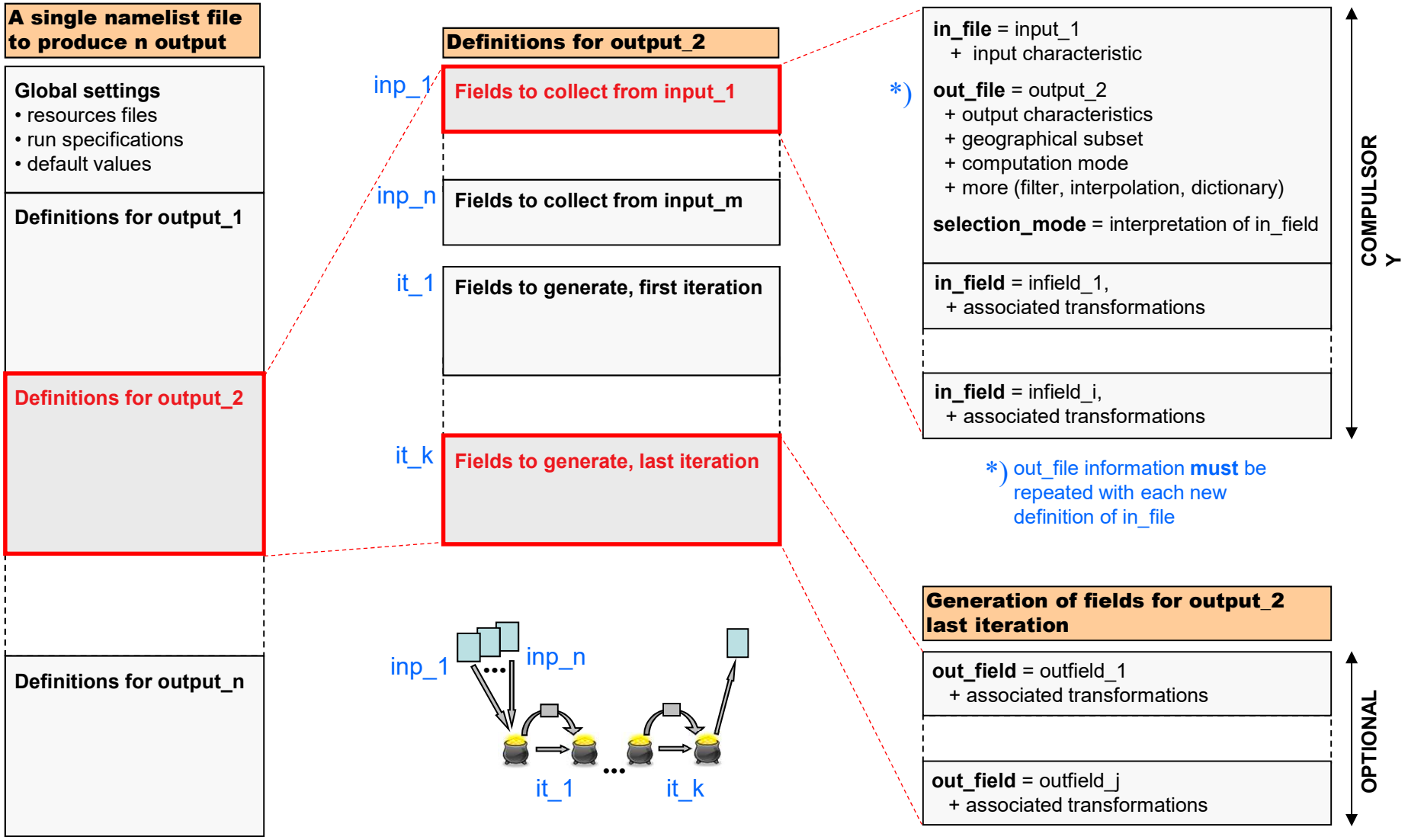


This procedure …

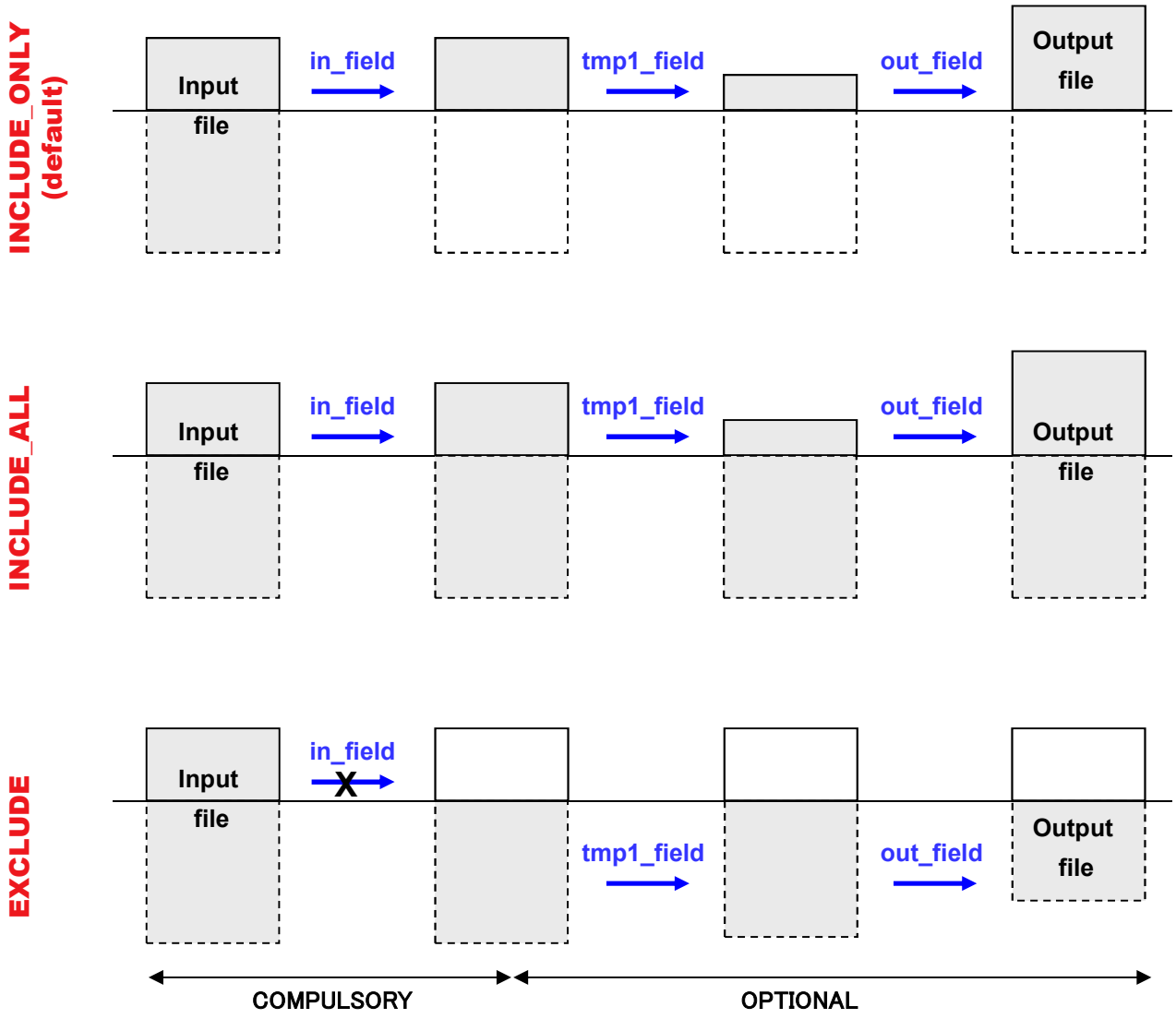
… can be repeated once!

- write a temporary GRIB file at the end of the first iteration
- use this temporary file as input for the second iteration
- dependencies are detected and files are processed in the correct order

Design – Namelist : basic



Design – Namelist : selection_mode

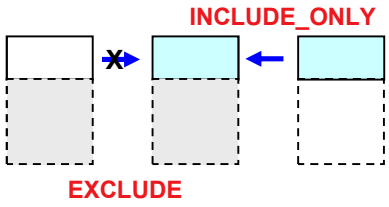


Typical usage:

Product generation

Transformation of input file

Merging 2 input



Design – Namelist : time levels (1)

- **A generic name may be used to loop over a set of input files**
  - typically to process a standard set of model output, characterized by one file per validation date
  - a key is inserted in the input file name (<DDHH>, <DDHHMMSS>, <YYYYMMDDHH:initial_date> …)
  - a list of times is defined explicitly (tlist) or by an implicit loop (tstart, tstop, tincr)

    → *the key is replaced in turn by each time, the same extraction pattern is applied on each input.*

- **Time operators may be applied on collected and generated fields**
  - all collected time levels are available, and only those

- **It is possible to filter the times collected in output**
  - another list of times defined by an implicit loop (out_tstart, out_tstop, out_tincr) is used, when available, to filter the list of times defined by (tlist) or (tstart, tstop, tincr)

    → *the validation dates available in output are those associated with the filtered input list*
  - this filter does not influence the set of time levels available for the time operators

Example: centered T2m mean, 6-hourly output		
	&Process	
	in_file="lfff**<DDHH>**0000“ , in_type="GRIB",	← **key in input file name**
	tstart=0, tstop=24, tincr=1,	← **implicit time loop**
	out_file=“product“ , out_type="GRIB1",	
	out_tstart=3, out_tincr=6 /	← **filter output date, implicit time loop**
	&Process	
	in_field= “T_2m“ , **toper**= “mean**,-3,3,1,hour**“ /	← **time operator**

Design – Namelist : time levels (2)

- **Instead of collecting all validation times in the same output, one file per validation time is created by using a generic name for the output file**
  - a key is inserted in the output file name (<DDHH>, <DDHHMMSS>, <YYYYMMDDHH:initial_date> …)
  - the list of times defined by (tlist) or by (tstart, tstop, tincr) is used to set the key values
  - filtering defined by (out_tstart ...) is respected

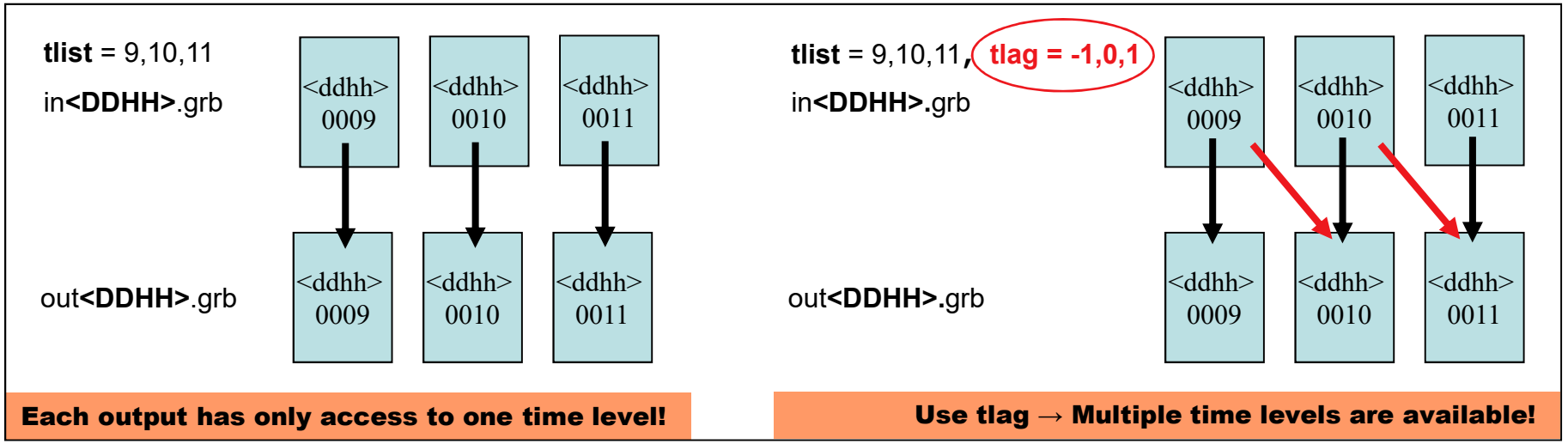
  - **in this case, the set of input files contributing to each output must be explicitely specified!**
    → use tlag (see next slide)

- **These mechanisms are based on the assumption that any file whose name matches**
  …<key>… , key in {DDHH, DDHHMMSS…}
  contains fields valid for the same date, and that the value of <key> represents this date!

Example: **centered T2m mean,** **6-hourly output,** **one output per date**	&Process in_file="lfff**<DDHH>**0000“ , in_type="GRIB", **tstart=0, tstop=24, tincr=1**, out_file=" product_**<DDHH>“** , out_type="GRIB1", **out_tstart=3**, **out_tincr=6, tlag=-3,3,1** / &Process in_field= “T_2m“ , **toper**= “mean**,-3,3,1,hour**“ /	← **key in output file name** ← **input/output association (for toper)**

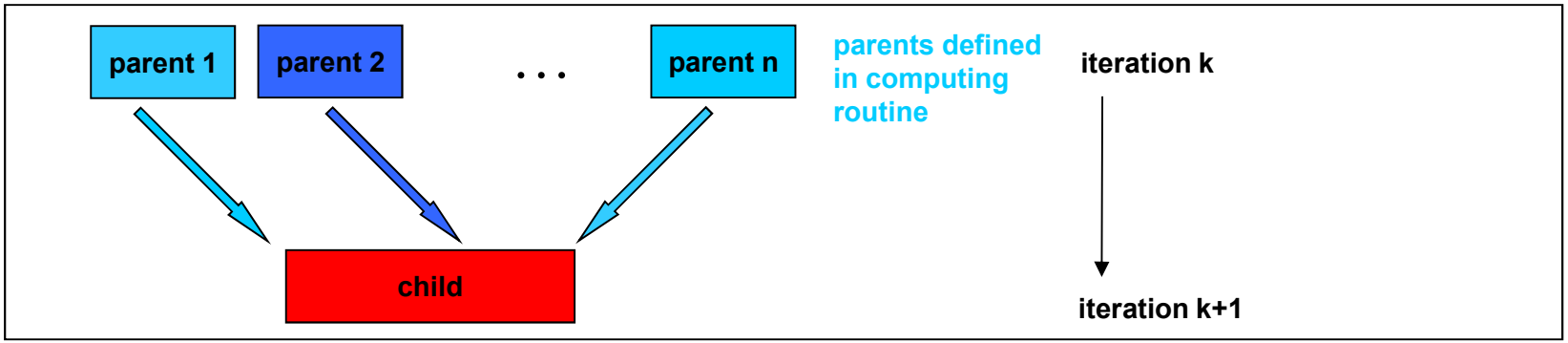
Design – Namelist : tlag

- **Explicit specification of contributing input files**
  - for each output file, fieldextra constructs the set of contributing input files
  - data can only be collected from this set
  - the set of contributing input files is not univoquely defined when multiple output files are defined within the same &Process group, which is the case when a time key is also used in the name of the output file
  - in this case a one to one correspondence is assumed, meaning that each output has only access to a single input, i.e. a single time level (see below)
  - when temporal operators requiring multiple time levels are used, the set of input files contributing to each output must be explicitly specified
  - this is done by using the namelist variable **tlag;** tlag defines an **interval of contributing input files** relative to the currently processed output (and refers to the list of times defined by tlist or tstart…)



Design – Computation of new fields (1)

Meteorological operator, activated via the name of the new field [$F_{n1}(\Psi)$]



Example

RELHUM_2M from T_2M and TD_2M

&Process in_field = "TD_2M" / ← parent 1

&Process in_field = "T_2M" / ← parent 2 (main parent, defined in field dictionary)

&Process out_field = "RELHUM_2M" / ← new field

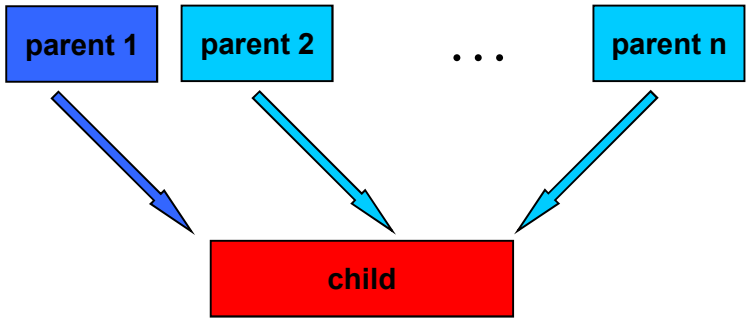
Implementation

Operators of common interest
- add new routine in fxtr_operator_generic
- extend case statement in fxtr_operator_generic:field_compute_generic

Operators of local interest
- add new routine in fxtr_operator_specific
- extend case statement in fxtr_operator_specific:field_compute_specific

Design – Computation of new fields (2)

Named operator, activated by setting "use_operator=…" [$F_{n1}(\Psi)$]



parents defined in computing routine

iteration k

↓

use_operator = "operator name"
use_tag = "parent 1, …, parent n"

↓

iteration k+1

Example

W_SO_CST from W_SO, FR_LAND, and SOILTYP

```
&Process in_field = "W_SO" /          ← parent 1 (main parent)
&Process in_field = "FR_LAND" /       ← parent 2
&Process in_field = "SOILTYP" /       ← parent 3

&Process out_field = "W_SO_CST",      ← new field
    use_operator = "wso_offset_v001", ← operator to use
    use_tag  = "W_SO,FR_LAND,SOILTYP" / ← list of parents (main parent first)
```

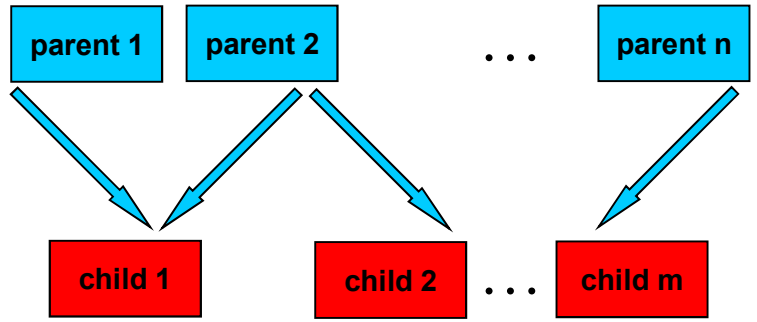
Implementation

Operators of common interest
- add new routine in fxtr_operator_generic
- extend case statement in fxtr_operator_generic:field_compute_generic
- extend module parameter fxtr_operator_generic:allowed_generic_nm_operator

Operators of local interest
- add new routine in fxtr_operator_specific
- extend case statement in fxtr_operator_specific:field_compute_specific
- extend module parameter fxtr_operator_specific:allowed_specific_nm_operator

Design – Computation of new fields (3)

Post-processing operator, activated by setting "out_postproc_module=…" [$F_{nm}(\Psi)$]



parents defined in pp routine

m ≠ n is possible, children defined in pp routine

final iteration

↓

out_postproc_module = "post-processing routine name"

↓

output file

Example

Transform wind and cloud cover to derive MeteoSwiss forecast matrix input

```
&Process
 in_file = "lfff<DDHH>0000",
 tstart = 0, tstop = 24, tincr = 1,
 locgroup = "nat",  loclist = "GVE",
 out_file = "forecast_matrix.txt",
 out_type = "FLD_TABLE",
 out_postproc_module = "pp_forecast_matrix" /
 …
 &Process out_field = "DD10MC_AM" /
 …
 &Process out_field = "DRSRRG_D" /
```

← operator applies to entire output

← parent 1

…

← parent n

Implementation

Operators of common interest

- add new post-processing routine in fxtr_operator_generic
- extend case statement in fxtr_operator_generic:data_postprocess_generic
- extend module parameter fxtr_operator_generic:allowed_generic_pp_procedure

Operators of local interest

- add new post-processing routine in fxtr_operator_specific
- extend case statement in fxtr_operator_specific:data_postprocess_specific
- extend module parameter fxtr_operator_specific:allowed_specific_pp_procedure

Design – Shared memory parallelism (1)

- Shared memory multitasking is available and implemented with **OpenMP directives**

- **File import :** *multiple input files are read and decoded concurrently*.
  In addition, in the case of fields defined on the native ICON grid:
  - *Parallel import and processing* of ICON grid definition
  - *Parallel re-gridding* from native ICON grid to any regular grid

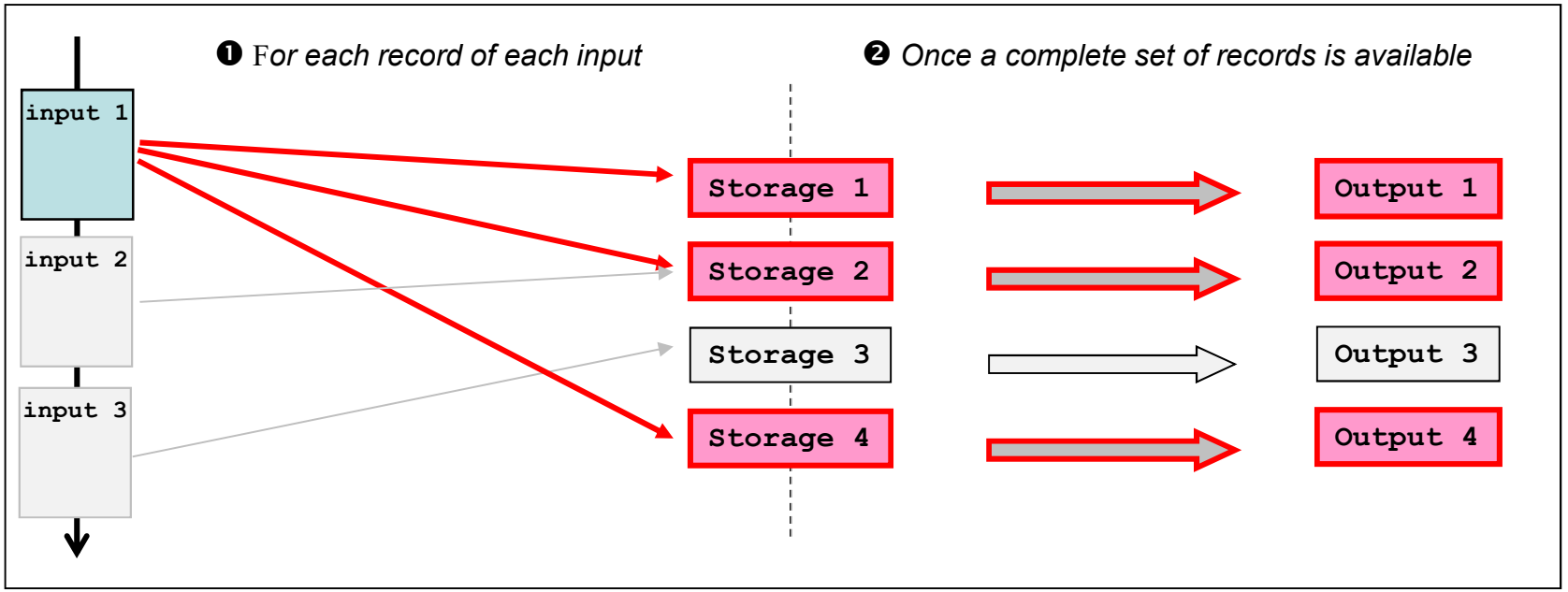
- **Product generation :** *two levels of parallelism* are implemented and can be simultaneously used
  - *parallel product generation, including export* (outer loop parallelism)
  - *parallelization of algorithms* used in product generation (inner loop parallelism)
- Two (exclusive) types of *algorithm parallelization* are available
  - *Grid points partitioning* (horizontal grid), if possible
  - Otherwise, *parallel computation of multiple 2D field lateral slices*, when the same operator is applied on multiple records within the current iteration

- Only shared memory parallelism

See the example./cookbook/multiple_products.nl

Design – Shared memory parallelism (2)

Parallel production of output (outer loop parallelism, marked with ——— below)



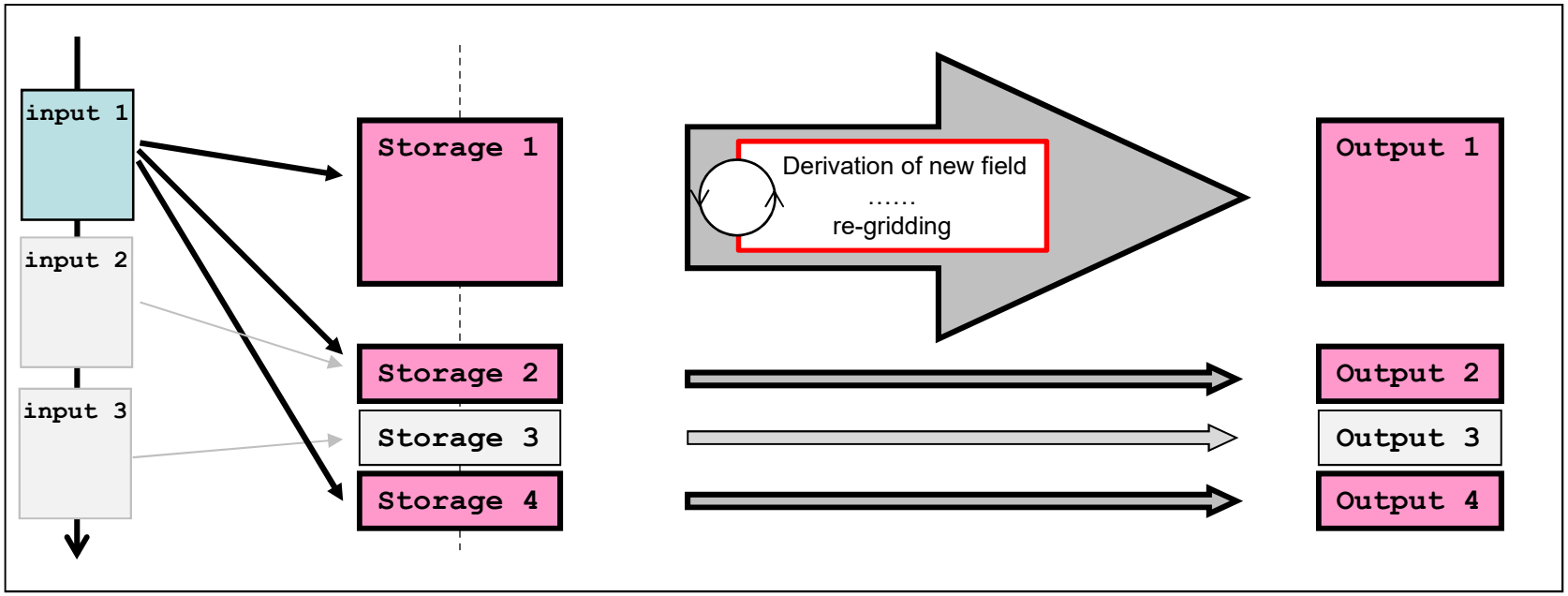
The following operations are applied in parallel (loop over output):

(1) For each record in turn :
check use of current record by output, process and store record

(2) Once a complete set of records is available :
iterative processing of fields , format and write output

Design – Shared memory parallelism (3)

Algorithm parallelization (inner loop parallelism, marked with ——— below)



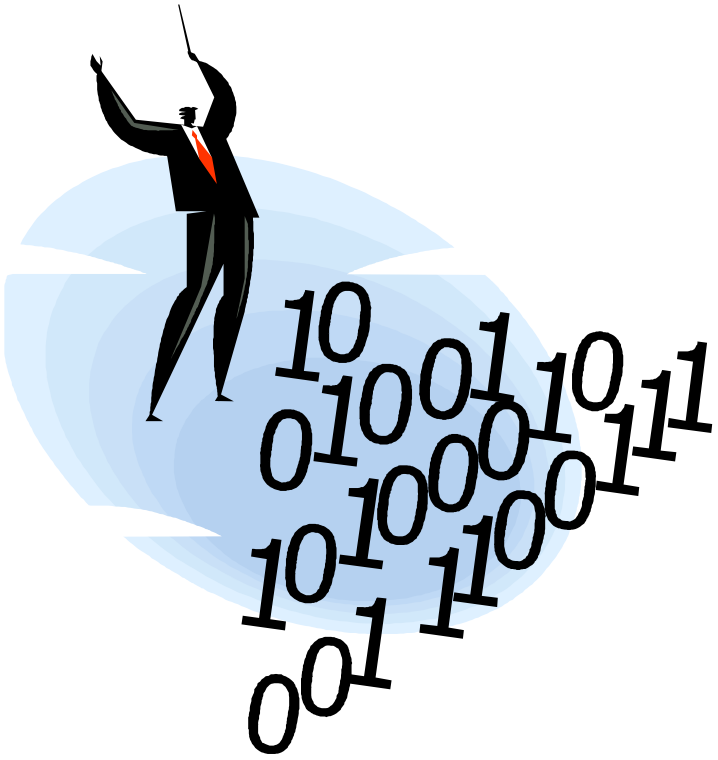
Within each processing iteration associated with each output,
for each operator in turn (hoper, poper...) :

parallel computation using grid points partitioning in (i,j) space, when no halo required

or

parallel computation using fields partitioning, when the same transformation is applied on multiple fields

Code structure



Modules

	PROGRAM:
Main	fieldextra
	MODULES (core functionality):
Parse namelist	fxtr_control,
Driver for product generation	fxtr_kernel,
Driver for field manipulation	fxtr_operator_main,
Transform field	fxtr_operator_column, fxtr_operator_regrid,
Compute new field	fxtr_operator_generic, fxtr_operator_specific, fxtr_operator_probability,
Support procedures (thermo…)	fxtr_operator_support,
Generate output	fxtr_write_generic, fxtr_write_obsolete, fxtr_write_specific
	MODULES (program specific support):
Type, symbolic constants …	fxtr_definition,
External resources	fxtr_resource_dictionary, fxtr_resource_gis, fxtr_resource_stat,
Storage / Meta info / Code profiling	fxtr_storage, fxtr_attribute, fxtr_profiling
	MODULES (generic support):
GRIB1 / GRIB2 / NetCDF	support_grib1, support_grib2, support_netcdf, support_blk_table
Vert. coordinates / ICON grid	support_vertical_mesh, support_icon_grid,
Storage / Code diagnostic	support_storage, support_diagnostic,
OpenMP	support_openmp
Date / Hor. Coordinates / …	support_datetime, support_gis, support_math, support_misc
	MODULES (imported from COSMO):
	cosmo_data_parameters
	cosmo_meteo_utilities, cosmo_pp_utilities, cosmo_utilities

Main data structure

TYPE ty_out_store (→ see fxtr_definition)

Variables of this type are used as
main repository for fields values and meta-information associated with each output.

- **Field values** are collected in **values(:,:,:)** array, where:
  first dim. is for *location* index , second dim. is for *field* index , third dim. is for *validation date* index

  A field in this context is a 2D field on a specific surface (ground, model, pressure…) and on a specific subgrid (cell, vertex, edge) of the horizontal base grid.
  A location is a grid point, but the set of used locations is not necessarily a rectangular domain.
  Note that the field values are stored in a 1-dimensional section of the values array.

  The number of locations for each subgrid is given by **nbr_gp(:)**, the number of fields by **nbr_field**, and the number of validation dates by **nbr_vdate**.

- The **characteristics of a field** are documented in
  field_id(:), field_epsid(:), field_pdfid(:), field_hgrid(:), field_level(:), field_product(:), field_trange(:,:) …
  (field_id(:)%name is field name, field_id(:)%tag is user defined tag)

  The **characteristics of a locations** are documented in
  gp_lat(:,:), gp_lon(:,:), gp_coord(:,:,:) …

  The **validation date** are documented in
  validation_date(:)

  Other information, **common** to all fields of the considered output, is available in:
  ofile_name, grid_hcoord, grid_vcoord, …

Main program

0. Initialization sequence, first part.
1. Read parameters defining program behaviour.
2. Initialization sequence, second part.

[input_file_group: DO] Loop through all groups of input files.
This loop is executed twice: first to collect fields for special output (INCORE, INSPECT), then to collect fields for standard output.

3. Generate output file
(just on time mode, all fields collected, last call).

[input_file_loop: DO] Loop through all input files in current group.

4. Skip or open input file.
4.1 Skip input when all associated input/output pairs are inactive
4.2 Select files to process
4.3 Wait for file
4.4 Process file
4.4.1 Detect type of first record, set calling order for API
GRIB file: DWD lib (GRIB1), ECMWF lib (GRIB2)
NetCDF file: NETCDF lib
BLK_TABLE file: internal API
[loop_over_api: DO] Loop over decoding API
4.4.2 Skip record if non matching API
4.4.3 Open file
4.4.4 Get global header

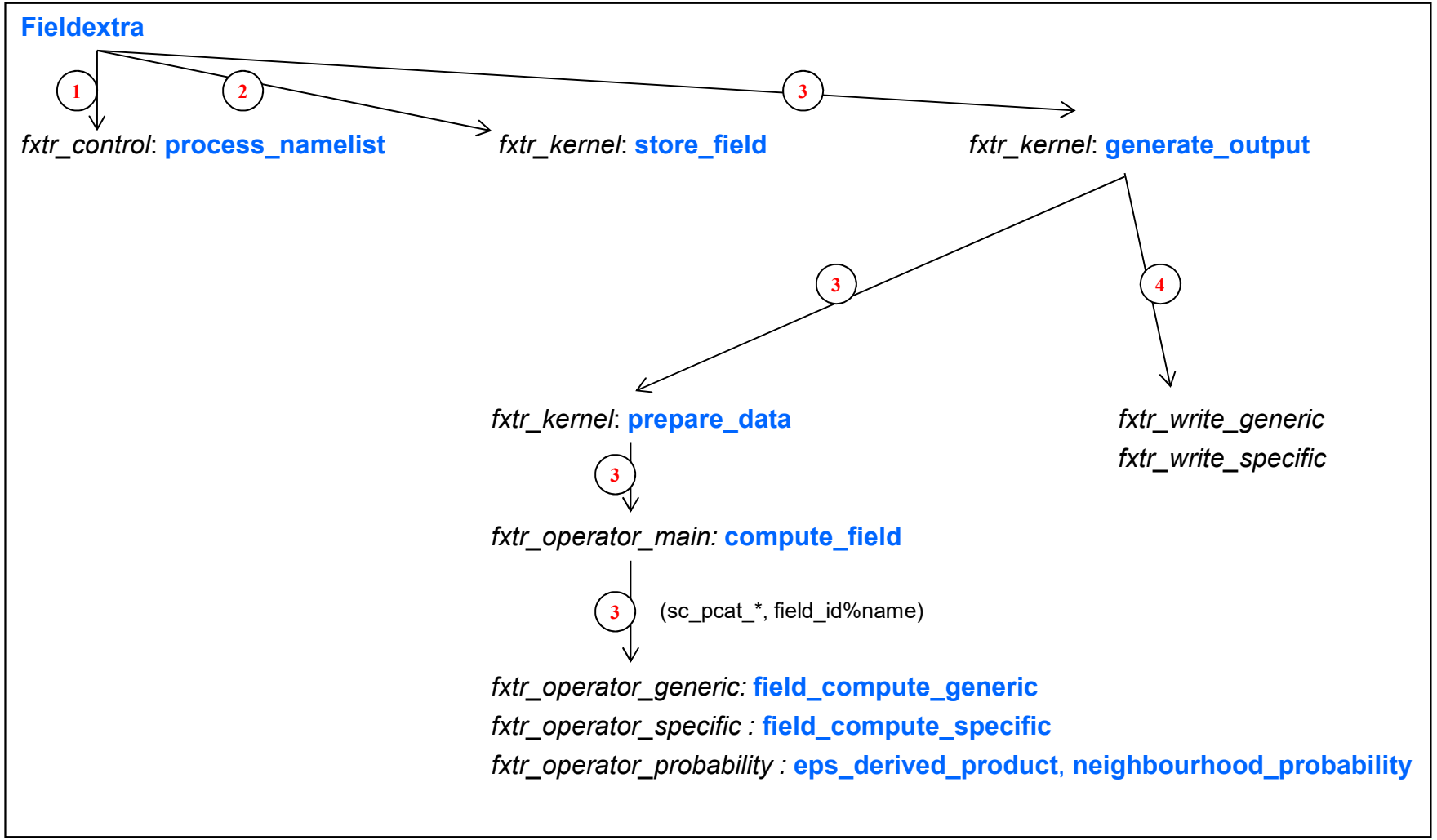
[input_records_loop: DO] Loop through all records in current input file.
5. Read and decode input field.
5.1 Clone cache (input missing)
5.2 Standard input file.
5.2.1 Read next record (skip data section, data will be read and decoded on request later on)
5.2.2 Decode meta-information
5.2.3 Process meta-information
5.2.4 Check meta-information
5.3 Pseudo input file __INCORE__ .

[output_file_loop: DO] Loop through all output files
6. Dispatch input field in output storage.
6.1 Does the current field contributes to the current output?
6.2 Unpack or generate field values
6.3 (On demand) crop field
6.4 (On demand) lateral regridding
6.5 Dispatch field in output storage.

[END DO output_file_loop, input_records_loop, loop_over_api, input_file_loop, input_file_group]

7. Operations requiring access to special storage (INCORE, INSPECT).
8. Diagnostic about missing fields.
9. (Repeat mode) store production diagnostic,
10. Final diagnostic, profiling and clean-up.

Calling tree : product generation



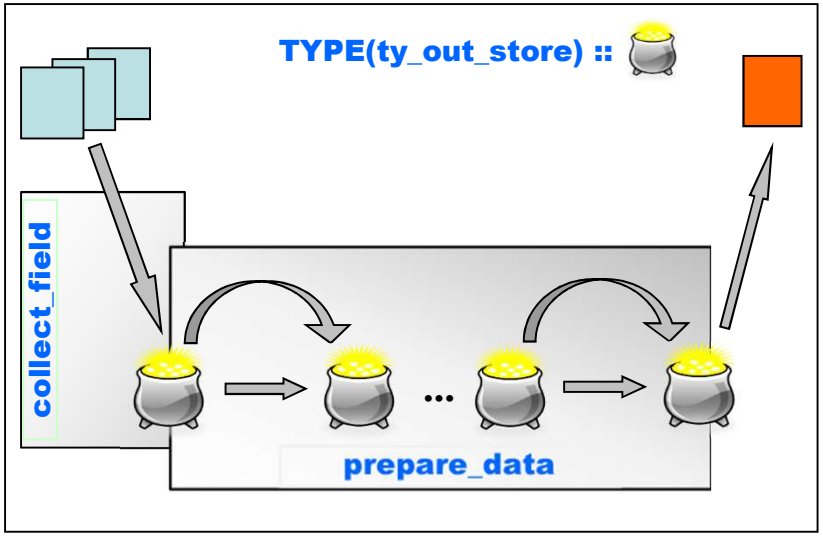
Iterative data processing : implementation

1. In main procedure

+ horizontal re-gridding (*in_regrid_target*)

2. In fxtr_kernel:store_field

+ modification of field meta-information (*set_units, set_reference_date* ...)
+ merge with another field (*merge_with*)
+ compare with another field (*compare_with*)
+ horizontal reduction of field (in advance of B2. when possible)



3. In fxtr_kernel:prepare_data

A. Processing of **constant fields** with respect to the time dimension

B. **Iterative processing of fields**

- **B1.1** build extended information about generated fields
  - **B1.1.1** look for field in previous set
  - **B1.1.2** look for main parent in previous set
  - **B1.1.3** derive relationship between child and parent
  - **B1.1.4** build full list of fields to generate/extract
- **B1.2** calculate new fields or copy fields from previous iteration
- **B2.** horizontal operator (*hoper*) and horizontal reduction of field
- **B3.** linear transformation (*scale, offset*)
- **B4.** transformation in a column (*voper*)
- **B5.** apply time operator (*toper*)
- **B6.** apply point operator (*poper*)
- **B7.** apply spatial filter (*in_filter* …)
- **B8.** reset field identity (*new_field_id*)
- **B9.** purge data from dates with inhibited print out

C. **Non iterative field transformations**

- C1. Reset field meta-information with user specified values (*set_units*, *set_reference_date* …)
- C2. Programmatic setting of some local meta-information
- C3. Check consistency of meta-information
- C4. Post-processing operator (*out_postproc_module*)
- C5. Re-gridding (*out_regrid_target*) *or* project on user specified domain (*slice …*)

D. **Prepare data for print out**

- D1. filter out undefined fields
- D2. filter out constant fields when requested
- D3. purge data from dates with inhibited print out
- D4. derive information common to all fields
- D5. update additional elements of data storage

User specific output format

Module fxtr_write_specific

SUBROUTINE write_specific

- ! 0.2 Decision about format

SUBROUTINE write_test

- ! ... List of expected fields
- ! 0.1 Initialize variables
- ! 0.2 Check usage of explicit locations
- ! 0.3 Check that data_pout content is supported by output type
- ! 0.4 Open output file
- ! 1. Look for data
- ! 2. Write header
- ! 3. Write data

Try to understand all details of the write_test routine,

clone this routine to produce your preferred format (TEST2) !

Some typical applications

Some typical applications
Pre-processing

- **Merge** surface temperature from IFS over sea and from COSMO over land to produce a single field suited for the assimilation cycle.
- **Interpolate** Swiss radar composite from kilometric grid onto the COSMO-2 grid for feeding the latent heat nudging process.
- **Interpolate** KENDA analysis **from regular grid to ICON triangular grid** to start re-forecasts for R&D
- **Rebuild EPS members** by adding EPS perturbation to a more recent determinist forecast.
- **Interpolate pollen fields from the ICON model** onto the COSMO grid to produce lateral boundary conditions for the COSMO model.
- **Upscale KENDA-1 analysis** from a 1.1km grid to a 2.2.km grid to produce initial conditions for COSMO-2E.

Some typical applications
Post-processing

- **Meteograms** (as text) at specified locations
- **Cropping** and **regridding** for user specific grids
- **Data thinning** of model output for verification purposes
- Computation of **geostrophic** wind and related quantities
- **Interpolation** of wind field on specified theta and PV surfaces
- **Split** file with multiple EPS members or validation dates in pieces
- **Fill holes** in a 2-dimensional field not defined everywhere (e.g HZEROCL)
- Mix multiple model output in a single XML file for **seamless forecast**
- **Convert** GRIB1 to GRIB2, incl. the specification of generalized height based vertical coord.
- **Kalman** correction at selected locations
- **MOS** based estimation of fields (including fields not part of model output!)
  - *Fieldextra expects the coefficients of the statistical model as external resource*
  - *Statistical filter computation is done outside of fieldextra!*

- Generation of **EPS products**
- Generate **lagged ensemble** from COSMO-2 rapid update cycle
- **Clone** missing member in COSMO-E output by changing member id

- **Real time production**: wait for model output, produce partial output every dH hours

Some typical applications
More complex products

- Generate a **soil type dependent field offset** and apply it to correct W_SO
- Find **3D location** of points where some conditions are fulfilled
  (e.g. over-saturation over ice and temperature above -20C)
- Compute spatially **upscaled EPS probability**
- Create a **bitmap** for the condition ‘probability of 6h sum of total precipitation exceeding 25mm is larger than 0’
- **Warn product** : compute region based quantiles of some fields under side conditions
  (e.g. 50% quantile of wind gust for all points below 800m where T_2m below 0C)
- **Freezing rain**: compute the integral of the temperature between the two lowest 0C isotherm in case of an inversion over a cold air pool
- **CAT for aviation**: compute indicators, find the height-surface of maximum CAT, compute the CAT category (low, medium, high) on this surface
- **FABEC product** : compute air density on a set of pressure and height above ground levels, interpolated on a geographical lat/lon grid, in GRIB 2
- **Monitoring** of model output : field values statistics, when values are outside of pre-defined validity range

Access, installation and usage

Access

- **Licenced software**
  - free to all COSMO members.
  - free licences for the **R&D community**, but without support.

- **Access**
  - **Master code repository on GitHub**
    https://github.com/COSMO-ORG/fieldextra (private repository)
  - **Package on COSMO web site**
    http://www.cosmo-model.org/content/support/software/default.htm
  - **Full installation at ECMWF on cca** (UNIX group cfxtra)
    /perm/ms/ch/ch7/projects/fieldextra
  - **Full installation at CSCS on tsa and daint** (UNIX group s83c)
    /project/s83c/fieldextra/{tsa,daint}

Access

Package on COSMO site

- **Tar file on COSMO web site, password protected**
  http://www.cosmo-model.org/content/support/software/default.htm
  - Source code for libraries (incl. config. script)
  - Source code for fieldextra
  - All necessary Makefiles (for gfortran, ifortran …)
  - All necessary resources (dictionary, location list ...)
  - Documentation (admin, compatibility, documentation)
  - Cookbook (used to validate installation)
  - Tools (including fx tools)

Fieldextra is only validated against the libraries and the associated resources included in the distribution package !

Installation

- **Follow steps in ./admin/INSTALLATION**
  - How to install, compile and test the code (almost all automatized)
  - Two cases are considered :
    - *Using **GitHub** (requires access to private repository)*
    - *Using **package** available on the COSMO web site*

- **New and modified features are documented in ./admin/HISTORY**

- **Backward compatibility issues are documented in ./compatibility files**

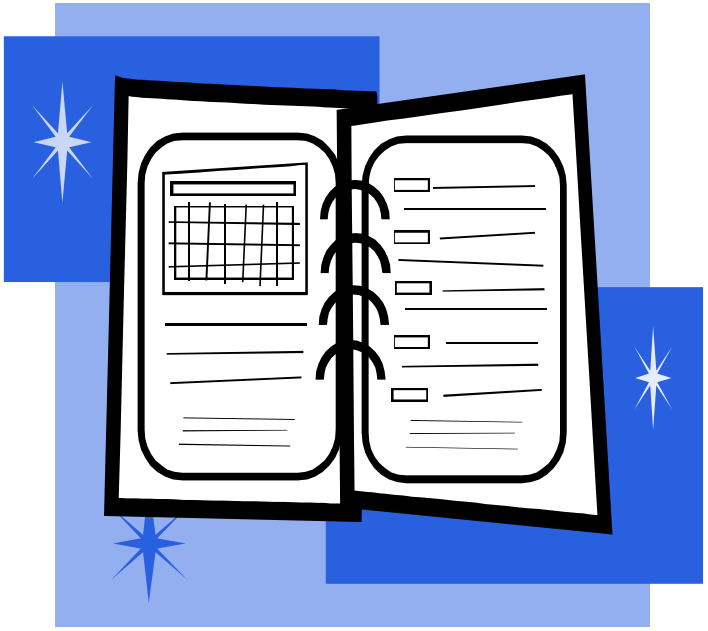
Usage

- **./cookbook**
  - Commented examples, the best way for learning how to use fieldextra
- **./documentation/FAQ**
  - Frequently asked questions
- **./documentation/README.user (and README.user.locale)**
  - Comprehensive description of functionalities
- **./documentation/README.grib_issues**
  - Summary of issues with GRIB 1 / 2 standards and with GRIB API
- **./admin/GRIB_POLICY**
  - GRIB 2 common COSMO policy

Things never work as planned …

- **Problem by installation ?**
  - carefully read and follow **./admin/INSTALLATION**
  - look at **./documentation/FAQ**
- **Namelist not working with newer release?**
  - consider documents in **./compatibility**
- **Problem by usage ?**
  - set **verbosity** to *high* (or *debug*) and **additional_diagnostic** to *true*
  - look at **./documentation/FAQ**
- **Do not know how to write the namelist for some application?**
  - get inspired by the **cookbook** examples
- **Get community support at** fieldextra@cosmo-model.org

Roadmap



What shall I expect next?

Release x.x

See

https://github.com/COSMO-ORG/fieldextra/milestones

Possible major developments

See

https://github.com/COSMO-ORG/fieldextra-wiki/wiki/Planning

E.g.

- Improve support for CI / CD
- Improve support of NetCDF format
- Support distributed memory parallelism (MPI)

Final discussion

Final discussion

Topics …

```
!+*****************************************************************************
SUBROUTINE generate_output(multi_pass_mode, just_on_time, last_call,              &
                           datacache, data_origin, tot_nbr_input,                 &
                           out_paths, out_types, out_modes,                       &
                           out_grib_keys, out_spatial_filters,                    &
                           out_subset_size, out_subdomain, out_gplist, out_loclist, &
                           out_data_reduction, out_postproc_modules,              &
                           nbr_gfield_spec, gen_spec, ierr, errmsg                )
!=============================================================================
!
! Root procedure to generate output files
!
!-----------------------------------------------------------------------------
 ! Dummy arguments
 LOGICAL, INTENT(IN)                          :: multi_pass_mode       ! Multiple pass mode?
 LOGICAL, DIMENSION(:), INTENT(IN)            :: just_on_time          ! True if prod. now
 LOGICAL, INTENT(IN)                          :: last_call             ! True if last call
 CHARACTER(LEN=*), INTENT(IN)                 :: datacache             ! Data cache file
 TYPE(ty_fld_orig), INTENT(IN)                :: data_origin           ! Data origin
 INTEGER, DIMENSION(:), INTENT(IN)            :: tot_nbr_input         ! Expected nbr. input
 CHARACTER(LEN=*), DIMENSION(:), INTENT(IN)   :: out_paths             ! Output files names
 TYPE(ty_out_spec), DIMENSION(:), INTENT(IN)  :: out_types             !   types
 TYPE(ty_out_mode), DIMENSION(:), INTENT(IN)  :: out_modes             !   modes
 INTEGER, DIMENSION(:,:), INTENT(IN)          :: out_grib_keys         !   grib specs
 INTEGER, DIMENSION(:), INTENT(IN)            :: out_subset_size       !   subset size
 INTEGER, DIMENSION(:,:), INTENT(IN)          :: out_subdomain         !   subdomain definition
 INTEGER, DIMENSION(:,:,:), INTENT(IN)        :: out_gplist            !   gp definition
 CHARACTER(LEN=*), DIMENSION(:,:), INTENT(IN) :: out_loclist           !   locations definition
 CHARACTER(LEN=*), DIMENSION(:,:), INTENT(IN) :: out_spatial_filters   ! Definition defining filter
 TYPE(ty_out_data_reduction), DIMENSION(:), INTENT(IN) :: out_data_reduction ! Data reduction method
 CHARACTER(LEN=*), DIMENSION(:,:), INTENT(IN) :: out_postproc_modules  ! Specific post processing
 INTEGER, DIMENSION(:,:), INTENT(IN)          :: nbr_gfield_spec       ! Specifications of
 TYPE(ty_fld_spec_root), DIMENSION(:), INTENT(IN) :: gen_spec          !+ fields to generate
 INTEGER, INTENT(OUT)                         :: ierr                  ! Error status
 CHARACTER(LEN=*), INTENT(OUT)                :: errmsg                !   error message

 ! Local parameters
 CHARACTER(LEN=*), PARAMETER          :: nm='generate_output: ' ! Tag

 ! Local variables
 LOGICAL                              :: exception_detected, exception, use_postfix
 LOGICAL                              :: unique_ftype, multiple_grid, exist
 LOGICAL, DIMENSION(3*mx_iteration+1) :: tmp_fddata_alloc, tmp_gpdata_alloc
 LOGICAL, DIMENSION(3*mx_iteration+1) :: tmp_value_alloc, tmp_flag_alloc
 INTEGER                              :: i1, i2, i3, i_fd, i_vd
 INTEGER                              :: nbr_input
 INTEGER                              :: out_idx, ios, idx_vd_defined
 CHARACTER(LEN=strlen)                :: messg, temporal_res, out_path
 TYPE(ty_fld_type)                    :: out_ftype



 ! Initialize variables
 !---------------------
 ierr = 0  ;  errmsg = ''
 exception_detected = .FALSE.
 tmp_fddata_alloc(:) = .FALSE.  ;  tmp_gpdata_alloc(:) = .FALSE.
 tmp_value_alloc(:) = .FALSE. ; tmp_flag_alloc(:) = .FALSE.


 ! Create/update data cache file
 !------------------------------
 ! The cache file must reflect the state of data(:) after the last call to
 ! collect_output (i.e. before any field manipulation done in prepare_pout)

 ! Loop over each output file
 !---------------------------
 output_file_loop: &
 DO i1 = 1, nbr_ofile
  out_idx = data(i1)%ofile_idx
  nbr_input = COUNT( data(i1)%ifile_used )

  ! Skip bogus output
  IF ( data(i1)%ofile_bogus ) CYCLE output_file_loop
  ! Skip completed output
  IF ( data(i1)%ofile_complete ) CYCLE output_file_loop
  ! Skip empty data array
  IF ( ALL(.NOT. data(i1)%defined) ) CYCLE output_file_loop
  ! Only prepare output when all possible associated data have been collected
  ! or when 'just on time' production is active
  IF ( .NOT. last_call                    .AND.                    &
       nbr_input < tot_nbr_input(out_idx) .AND.                    &
       .NOT. just_on_time(out_idx)            ) CYCLE output_file_loop

  ! At this point the corresponding output file will be produced
  ! Keep track of completed output file
  IF ( nbr_input >= tot_nbr_input(out_idx) ) data(i1)%ofile_complete = .TRUE.

  ! Build name of output, considering a possible temporary postfix
  use_postfix = .FALSE.
  IF ( LEN_TRIM(out_postfix) /= 0 .AND. data(i1)%ofile_usepostfix .AND.   &
       .NOT. (data(i1)%ofile_firstwrite .AND. data(i1)%ofile_complete)  ) &
                                                  use_postfix = .TRUE.
  out_path = out_paths(out_idx)
  IF ( use_postfix ) out_path = TRIM(out_path)//TRIM(out_postfix)
  ! Release memory allocated in previous call to prepare_pout (if any)
  DO i2 = 1, 3*mx_iteration+1
   IF ( tmp_value_alloc(i2) ) DEALLOCATE(data_tmp(i2)%values, data_tmp(i2)%defined)
   IF ( tmp_flag_alloc(i2) ) DEALLOCATE(data_tmp(i2)%flag)
   IF ( tmp_fddata_alloc(i2) ) THEN
    DEALLOCATE(data_tmp(i2)%field_type, data_tmp(i2)%field_origin,       &
data_tmp(i2)%field_name, data_tmp(i2)%field_grbkey,       &
               data_tmp(i2)%field_trange,                              &
               data_tmp(i2)%field_level, data_tmp(i2)%field_ltype,     &
               data_tmp(i2)%field_prob, data_tmp(i2)%field_epsid,      &
               data_tmp(i2)%field_vref, data_tmp(i2)%field_ngrid,      &
               data_tmp(i2)%field_scale, data_tmp(i2)%field_offset,    &
               data_tmp(i2)%field_vop, data_tmp(i2)%field_vop_usetag,  &
               data_tmp(i2)%field_vop_nlev, data_tmp(i2)%field_vop_lev, &
               data_tmp(i2)%field_pop, data_tmp(i2)%field_hop,         &
               data_tmp(i2)%field_top, data_tmp(i2)%nbr_level,         &
               data_tmp(i2)%level_idx, data_tmp(i2)%nbr_eps_member,    &
               data_tmp(i2)%eps_member_idx, data_tmp(i2)%field_idx      )
   ENDIF
   IF ( tmp_gpdata_alloc(i2) ) THEN
    DEALLOCATE(data_tmp(i2)%gp_coord, data_tmp(i2)%gp_idx,              &
               data_tmp(i2)%gp_lat, data_tmp(i2)%gp_lon, data_tmp(i2)%gp_h)
   ENDIF
  END DO

  ! Prepare data for print out (calculate new fields, ... ; populate data_pout)
  ! * Info message
  IF ( just_on_time(out_idx) ) THEN
   messg = ' (just on time output)'
  ELSE IF ( nbr_input >= tot_nbr_input(out_idx) ) THEN
   messg = ' (all associated input collected)'
  ELSE
   messg = ''
  ENDIF
```

Thank you for your attention!