



Max-Planck-Institut
für Meteorologie



The Coupling Library YAC

Moritz Hanke (DKRZ), René Redler (MPI-M), and Nils-Arne Dreier (DKRZ)



YAC-Team

- Main developer:
 - Moritz Hanke (DKRZ)
- With contributions from:
 - René Redler (MPI-M)
 - Nils-Arne Dreier (DKRZ)
 - Teresa Holfeld (MPI-M, student assistant)
 - Maxim Yastremsky (MPI-M, student assistant)
 - Thomas Jahns (DKRZ)
 - Uwe Schulzweida (MPI-M)
 - Hendryk Bockelmann (DKRZ)
 - Jörg Behrens (DKRZ)
 - Sergey Kosukhin (MPI-M)

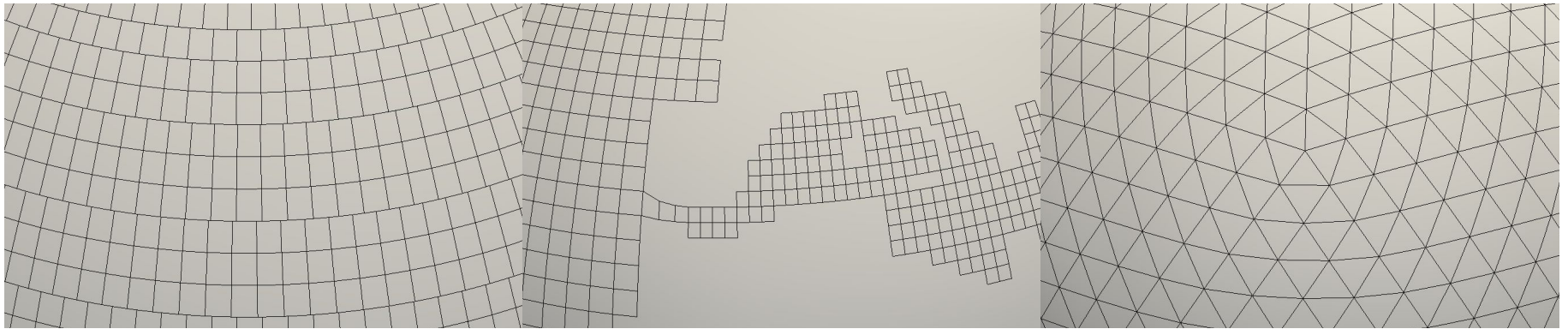
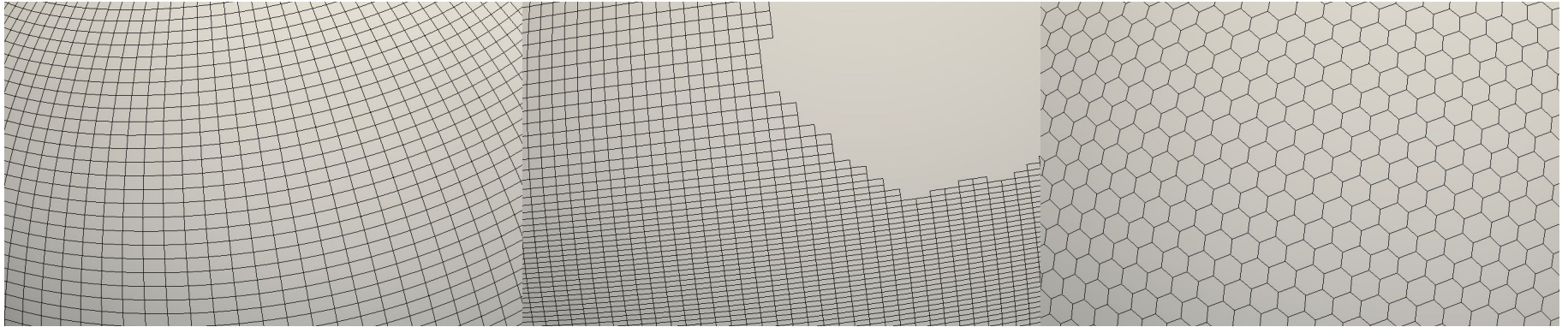
Introduction

- What does a coupler do?
 - exchange of data between independent components (e.g. atmosphere and ocean) at predefined time intervals
 - components can use different grids
 - coupler takes care of regridding between both grids
 - components can have different decompositions
 - coupler takes care of data redistribution
 - components can have different exchange periods
 - coupler takes care of matching of the exchanges and data aggregation if necessary

Introduction

- Library linked to components
 - Written in C
 - Unit tests cover 99.5% of the code
 - C-, Fortran-, and Python-Interface
- Supports all common grid types
- Provides various 2D-interpolation schemes
 - All computation on the unit sphere using cartesian coordinates
- Parallel online weight computation
- Licenced unter BSD 3-Clause License
- Used in official ICON-release (but developed independently)
- runs on Piz Daint¹, JUWELS², Levante³, LUMI⁴, MAC OS, and Linux systems with little to no porting effort

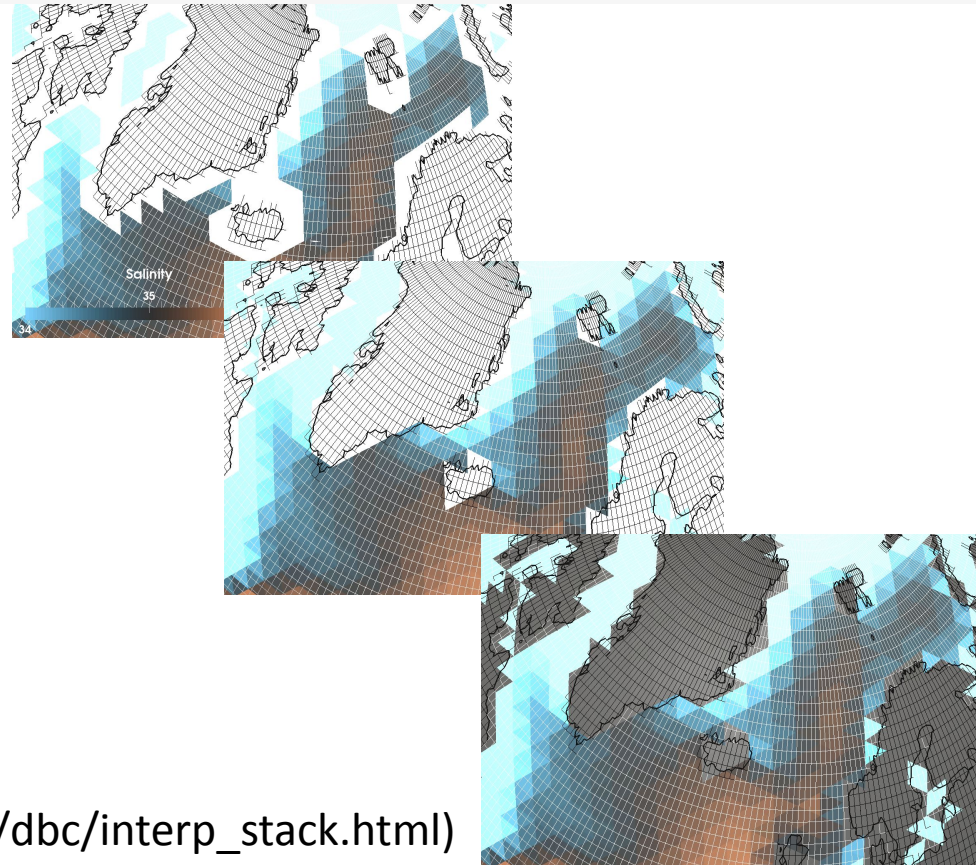
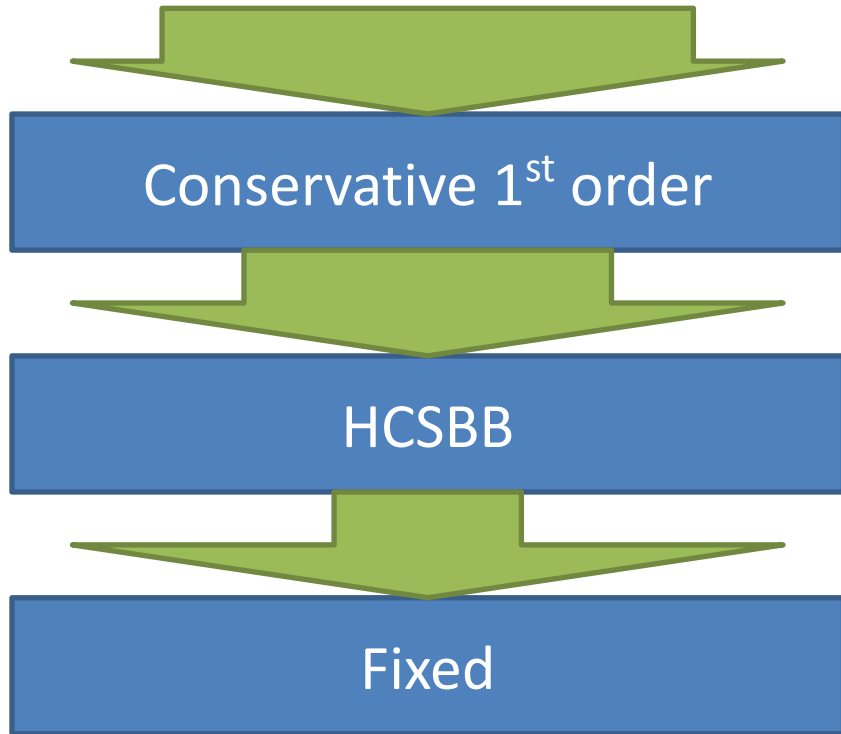
Supported grids



Interpolations

- 2D inter- and extrapolation
 - Various methods with different properties
 - All grid combinations are supported
- Fields can be defined on cells, vertices, or edges

Interpolation Stack

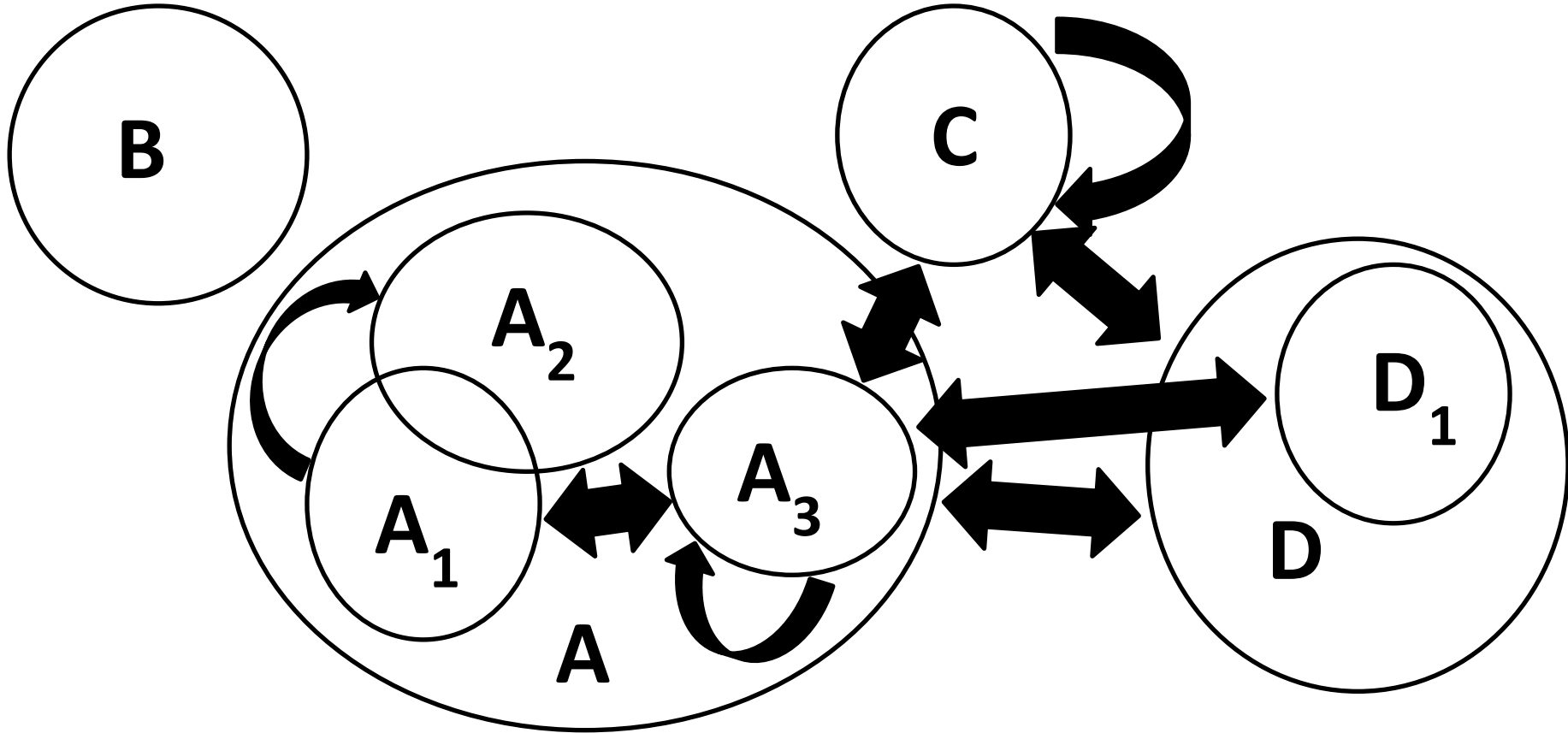


(https://dkrz-sw.gitlab-pages.dkrz.de/yac/db/dbc/interp_stack.html)

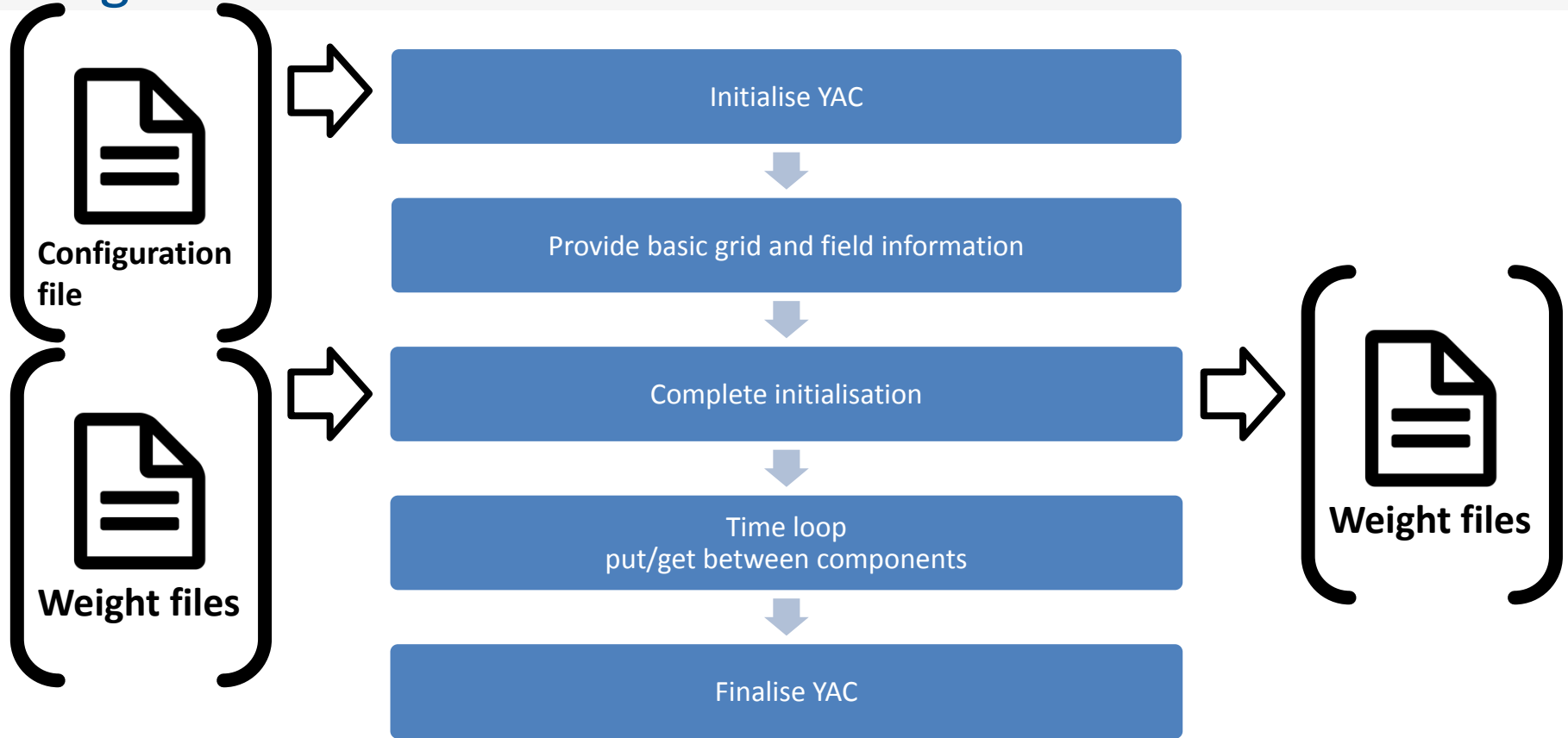
Interpolation Quality

- Benchmark on regridding quality by CERFACS in the frame of ISENES-Project
 - “This benchmark leads us to conclude that YAC, ESMF, and XIOS can all three be considered as high-quality regridding libraries [...]”
 - Valcke, S.; Piacentini, A.; Jonville, G. Benchmarking Regridding Libraries Used in Earth System Modelling. Math. Comput. Appl. 2022, 27, 31.
<https://doi.org/10.3390/mca27020031>

Supported coupling configurations



Usage



Fortran example

```
USE mo_yac_finterface  
  
CALL yac_finit()  
CALL yac_fdef_calendar(&  
    YAC_PROLEPTIC_GREGORIAN)
```

Fortran example

```
CALL yac_fread_config_yaml ( &  
    "coupling.yaml")
```

Fortran example

```
INTEGER :: comp_id
INTEGER :: comp_comm

CALL yac_fdef_comp( &
    "atmosphere", comp_id)
CALL yac_fget_comp_comm( &
    comp_id, comp_comm)
```

Fortran example

```
INTEGER, PARAMETER :: nbr_vertices = ...
INTEGER, PARAMETER :: nbr_cells = ...
INTEGER, PARAMETER :: nbr_vertices_per_cell = ...
REAL, ALLOCATABLE :: vertex_lon(nbr_vertices)
REAL, ALLOCATABLE :: vertex_lat(nbr_vertices)
INTEGER, ALLOCATABLE :: &
    cell_to_vertex(nbr_vertices_per_cell,nbr_cells)

INTEGER :: grid_id

CALL yac_fdef_grid( &
    "atmosphere_grid", nbr_vertices, nbr_cells, &
    nbr_vertices_per_cell, vertex_lon, vertex_lat, &
    cell_to_vertex, grid_id)
```

Fortran example

```
REAL, ALLOCATABLE :: cell_lon(nbr_cells)
REAL, ALLOCATABLE :: cell_lat(nbr_cells)

INTEGER :: cell_point_id

CALL yac_fdef_points( &
    grid_id, nbr_cells, &
    YAC_LOCATION_CELL, &
    cell_lon, cell_lat, &
    cell_point_id)
```

Fortran example

```
INTEGER, PARAMETER :: collection_size = ...
```

```
INTEGER :: taux_field_id
```

```
CALL yac_fdef_field( &  
  "TAUX", comp_id, &  
  (/cell_point_id/), 1, &  
  collection_size, "600", &  
  YAC_TIME_UNIT_SECOND, &  
  taux_field_id)
```


Fortran example

```
CALL yac_fenddef()
```

Fortran example

```
REAL :: tau_x_buffer(nbr_cells, &  
                    collection_size)  
INTEGER :: info, error  
  
DO t = 1, ntimes  
    ...  
    CALL yac_fput( &  
        tau_x_field_id, nbr_cells, &  
        collection_size, tau_x_buffer, &  
        info, error)  
    ...  
END DO
```

Fortran example

```
REAL :: tau_x_buffer(nbr_cells, &  
                    collection_size)  
INTEGER :: info, error  
  
DO t = 1, ntimes  
    ...  
    CALL yac_fget( &  
        tau_x_field_id, nbr_cells, &  
        collection_size, tau_x_buffer, &  
        info, error)  
    ...  
END DO
```

Fortran example

```
CALL yac_ffinalize()
```

End

- Questions?
- Download: <https://gitlab.dkrz.de/dkrz-sw/yac>
- Documentation: <https://dkrz-sw.gitlab-pages.dkrz.de/yac/>
- References
 - M. Hanke, R. Redler, T. Holfeld und M. Yastremsky, 2016: YAC 1.2.0: new aspects for coupling software in Earth system modelling. *Geoscientific Model Development*, 9, 2755-2769, <https://doi.org/10.5194/gmd-9-2755-2016>
 - M. Hanke und R. Redler, 2019: New features with YAC 1.5.0. *Reports on ICON*, No 3. https://doi.org/10.5676/DWD_pub/nwv/icon_003
 - E. Kritsikis, M. Aechtner, Y. Meurdesoif, and T. Dubos: Conservative interpolation between general spherical meshes, *Geosci. Model Dev.*, 10, 425–431, <https://doi.org/10.5194/gmd-10-425-2017>, 2017
 - Xiaoyu Liu, Larry L. Schumaker, Hybrid Bézier patches on sphere-like surfaces, *Journal of Computational and Applied Mathematics*, Volume 73, Issues 1–2, 1996, Pages 157-172, ISSN 0377-0427, [https://doi.org/10.1016/0377-0427\(96\)00041-6](https://doi.org/10.1016/0377-0427(96)00041-6)



Max-Planck-Institut
für Meteorologie



The Coupling Library YAC

Moritz Hanke (DKRZ), René Redler (MPI-M), and Nils-Arne Dreier (DKRZ)



Topics

- Masks types
- Configuration files
- Definition of couples
- Synchronisation of definitions
- Querying of definitions

Masks types

Core mask

defined per grid

masked out cells/vertices/edges are completely ignored by YAC

used to mask out degenerated and duplicated cells/vertices/edges

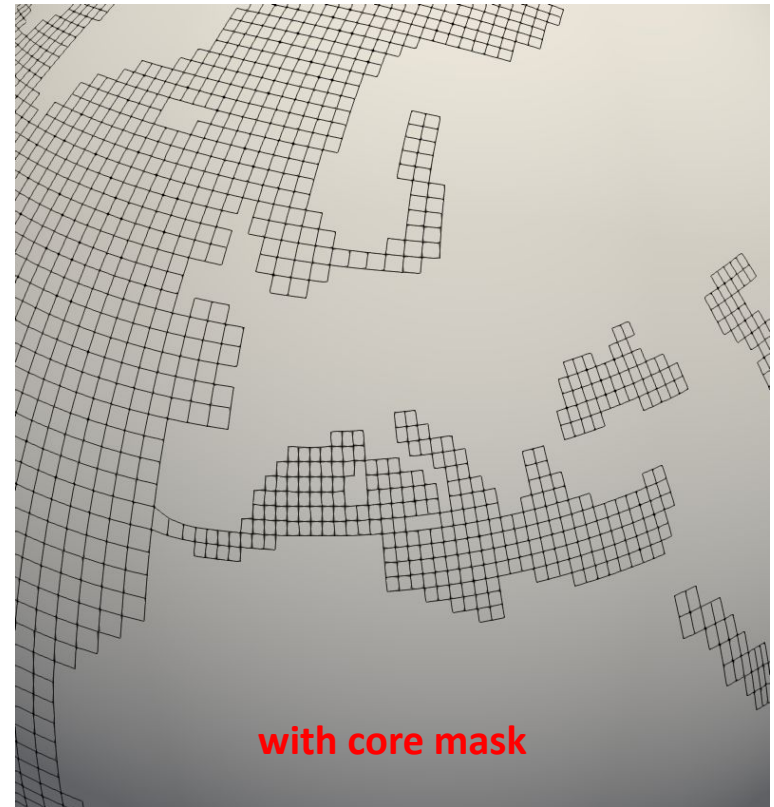
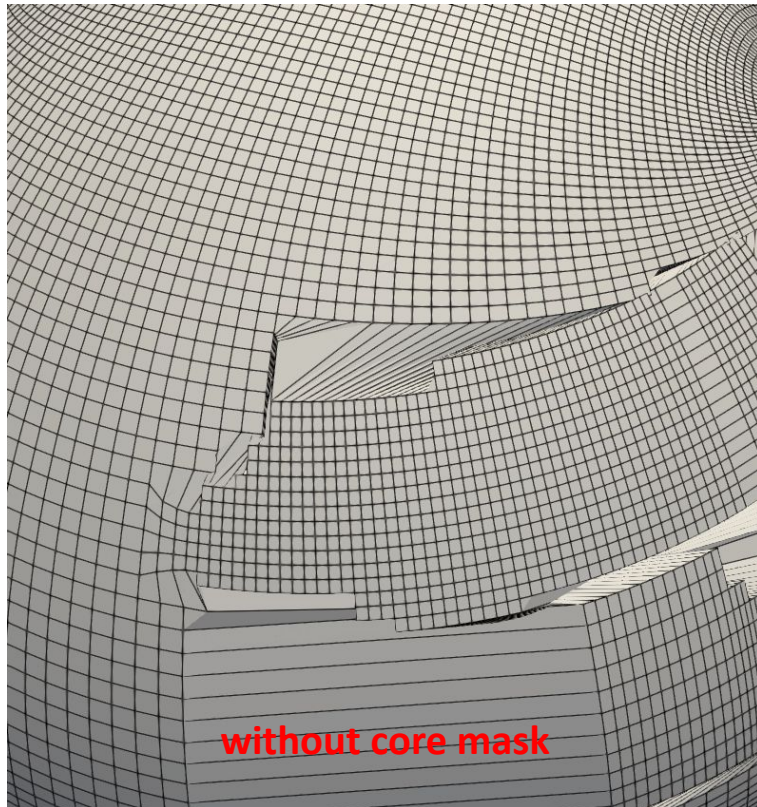
Field mask

defined per points or per field

mask out cells/vertices/edges are ignored in the weight computation

used to mask out cells/vertices/edges that have no valid data assigned to them (e.g. halos) or that should not receive data

Core mask example



Field mask application

- In atmo/ocean coupling
 - deactivate land points in global atmo grid
- Halos
 - deactivate halos for outgoing fields
→ send only valid data
 - activate halos for ingoing fields
→ no halo update required after coupling

Configuration files

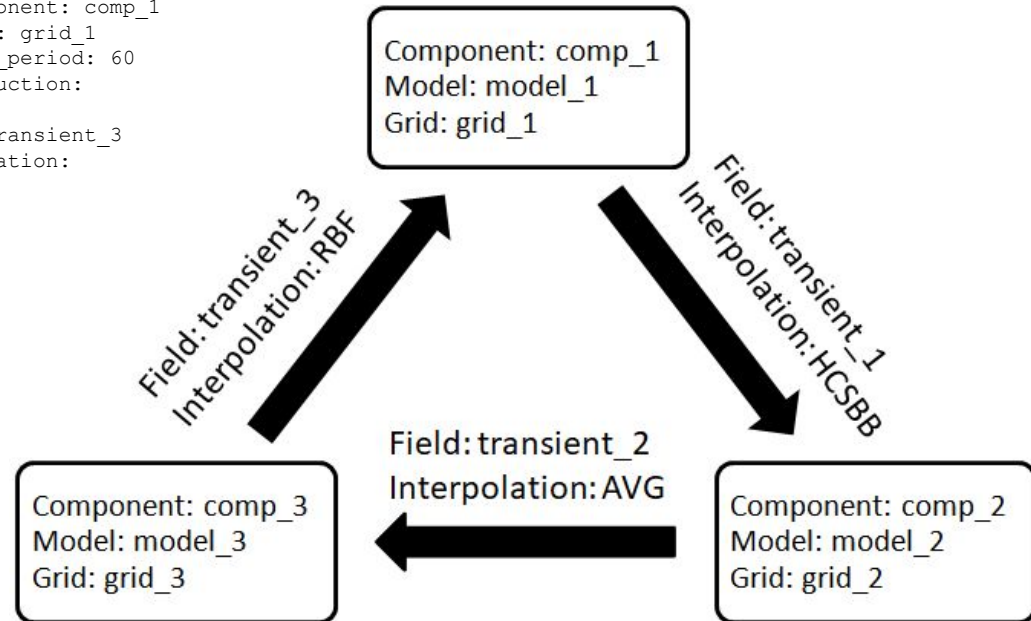
- Contains information about
 - (optional) start- and end date of the run
 - (optional) calendar to be used
 - which fields are supposed to be coupled
 - what interpolation is supposed to be used
 - at which frequency the coupling is supposed to be executed
- Have to be read in by at least one process
- One or more configuration files can be read by arbitrary processes
- Full support of YAML Version 1.2
- Documentation at:
https://yac.gitlab-pages.dkrz.de/YAC-dev/dd/dfa/yaml_file.html

Configuration files example

```

start_date: 2008-03-09T16:05:07    # comp_3 -> comp_1
end_date: 2008-03-10T16:05:07      - src_component: comp_3
timestep_unit: second              src_grid: grid_3
calendar: proleptic-gregorian      tgt_component: comp_1
coupling:                          tgt_grid: grid_1
# comp_1 -> comp_2                  coupling_period: 60
- src_component: comp_1            time_reduction:
  src_grid: grid_1                 accumulate
  tgt_component: comp_2           field: transient_3
  tgt_grid: grid_2                interpolation:
  coupling_period: 60              - rbf
  time_reduction: accumulate
  field: transient_1
  interpolation:
    - bernstein_bezier
# comp_2 -> comp_3
- src_component: comp_2
  src_grid: grid_2
  tgt_component: comp_3
  tgt_grid: grid_3
  coupling_period: 60
  time_reduction: accumulate
  field: transient_2
  interpolation:
    - average

```



Configuration files example

```

definitions:
- &time_config
  src_lag: 1
  tgt_lag: 1
  coupling_period: 3600
- &config_model
  <<: *time_config
  time_reduction: none
  interpolation:
    - nnn
    - fixed:
        user_value: -999.0
- &config_io
  <<: *time_config
  time_reduction: accumulate
  interpolation:
    - conservative:
        enforced_conservation: false
        normalisation: fracarea
        partial_coverage: false
    - fixed:
        user_value: -999.0

start_date: 2008-03-09T16:05:07
end_date: 2008-03-10T16:05:07
timestep_unit: second
calendar: proleptic-gregorian
coupling:
- src_component: dummy_atmosphere
  src_grid: dummy_atmosphere_grid
  tgt_component: dummy_ocean
  tgt_grid: dummy_ocean_grid
  <<: *config_model
  field: [surface_downward_eastward_stress,
          surface_downward_northward_stress,
          surface_fresh_water_flux,
          surface_temperature,
          total_heat_flux,
          atmosphere_sea_ice_bundle]
- src_component: dummy_ocean
  src_grid: dummy_ocean_grid
  tgt_component: dummy_atmosphere
  tgt_grid: dummy_atmosphere_grid
  <<: *config_model
  field: [sea_surface_temperature,
          eastward_sea_water_velocity,
          northward_sea_water_velocity,
          ocean_sea_ice_bundle]

```

Definition of couples

- Couples can be defined in definition phase by
 - reading of configuration file
 - call to user interface routine
`yac_fdef_couple`

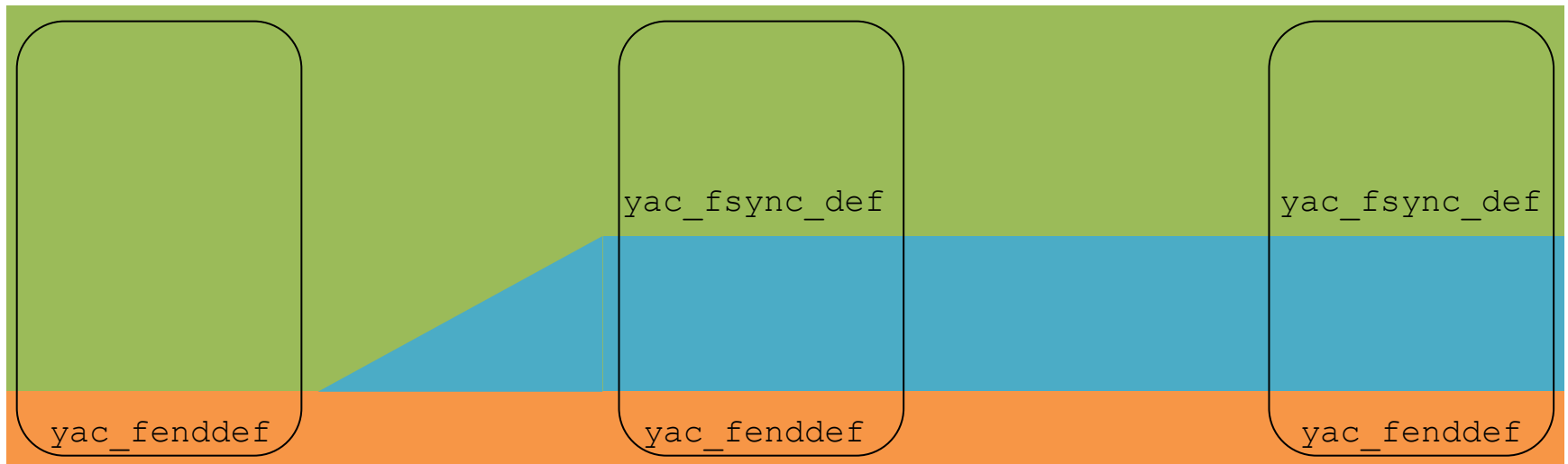
Definition of couples via user interface

```
subroutine yac_fdef_couple (                                &
  src_comp_name, src_grid_name, src_field_name,          &
  tgt_comp_name, tgt_grid_name, tgt_field_name,          &
  coupling_timestep, time_unit, time_reduction,          &
  interp_stack_config_id, src_lag, tgt_lag,              &
  weight_file, mapping_side, scale_factor,              &
  scale_summand, src_mask_names, tgt_mask_name )
```

! *: optional arguments

Synchronisation of definitions

Definition phase (grids, fields, and couples)



Querying of definitions

- YAC internally keeps global configuration information about all components, grids, and fields on each process
- Each processes can query about this information
- Examples:
 - Is component “atmo” defined
 - Has component “ocean” defined field “sea_surface_temperature”
 - What is the collection size of field “total_heat_flux” on component “atmo”

End

- Questions?
- Download: <https://gitlab.dkrz.de/dkrz-sw/yac>
- Documentation: <https://dkrz-sw.gitlab-pages.dkrz.de/yac/>
- References
 - M. Hanke, R. Redler, T. Holfeld und M. Yastremsky, 2016: YAC 1.2.0: new aspects for coupling software in Earth system modelling. *Geoscientific Model Development*, 9, 2755-2769, <https://doi.org/10.5194/gmd-9-2755-2016>
 - M. Hanke und R. Redler, 2019: New features with YAC 1.5.0. Reports on ICON, No 3. https://doi.org/10.5676/DWD_pub/nwv/icon_003
 - E. Kritsikis, M. Aechtner, Y. Meurdesoif, and T. Dubos: Conservative interpolation between general spherical meshes, *Geosci. Model Dev.*, 10, 425–431, <https://doi.org/10.5194/gmd-10-425-2017>, 2017
 - Xiaoyu Liu, Larry L. Schumaker, Hybrid Bézier patches on sphere-like surfaces, *Journal of Computational and Applied Mathematics*, Volume 73, Issues 1–2, 1996, Pages 157-172, ISSN 0377-0427, [https://doi.org/10.1016/0377-0427\(96\)00041-6](https://doi.org/10.1016/0377-0427(96)00041-6)

YAC in OASIS

- OASIS3-MCT 6.0 planned for end 2024
 - contains optional online weight computation by YAC

Documentation

```
const char * start_datetime = "01-01-1850T00:00:00";
const char * end_datetime = "31-12-1850T00:00:00";
// Both arguments are optional (can be NULL)
yac_def_datetime ( start_datetime, end_datetime );

character(len=YAC_MAX_CHARLEN) :: start_datetime
character(len=YAC_MAX_CHARLEN) :: end_datetime
start_datetime = '01-01-1850T00:00:00'
end_datetime = '31-12-1851T00:00:00'
yac_def_datetime ( start_datetime, &
                  end_datetime )
```

A coupled run configuration may consist of multiple executables or programs, e.g. model.a.x and model.b.x. If the processes of a single executable have to register multiple component individual communicator that contain only the processes of their respective executable in order to be able to determine the component associated to each process.

Initialising YAC contains more information on how to handle more complex setups such as the one described above.

The Definition Phase

Component Introduction

In complex coupled run configurations with multiple different executables, a common problem is the initial MPI communicator splitting. At the start of the run multiple communicators have to be built, for example one for each executable or for groups of executables. These communicators are required by the models themselves and by libraries used by one or more of the models (e.g. coupler or IO).

Each library and/or model can implement its own algorithm for splitting the initial MPI_COMM_WORLD. However, this can lead to conflicts and deadlocks between the different algorithms.

YAC provides the `split_comm_world` function which implements a MPI handling algorithm that generates multiple different communicators in a clean, collective manner.

As long as DKRZ has MPI Hand

Once, all components MPI

void yac MPI Co size 1 char * MPI Co

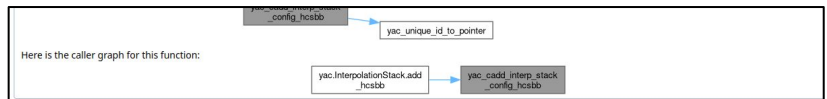
• com

• n

• gro

• gro

For each e



```
void yac_cadd_interp_stack_config_nnn ( int interp_stack_config_id,
```

```
• yac_cget_comps_comm
• yac_cget_comps_comm_instance
• yac_fget_comps_comm (*)
• yac.YAC.get_comps_comm
```

Reading configuration file

```
• yac_cread_config_yaml
• yac_cread_config_yaml_instance
• yac_cread_config_json
• yac_cread_config_json_instance
• yac_fread_config_yaml (*)
• yac_fread_config_json (*)
• yac.YAC.read_config_yaml
```

Interpolation Introduction to exemplary coupled model setup

The following picture shows an exemplary coupled model setup. It consists of two main components (Atmosphere and Ocean) and a number of other processes not involved in any coupling (for example I/O-Server).

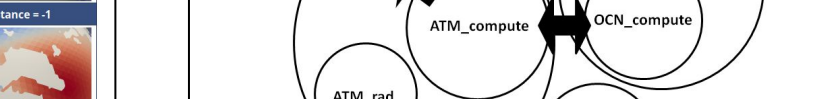
The Atmosphere itself is comprised of two subcomponents: **compute** (simulating the atmosphere), **rad** (simulating radiation), and **out** (doing output). The components **compute** and **rad** communicate through other means than YAC (for example yac!). Via the **out** component data from **compute** are written to the file system. Since in this setup the communication between **compute** and **out** is handled by YAC, **out** can use a different grid than **compute** and/or only a subsection of the area covered by **compute**.

The setup of the Ocean component is similar to Atmosphere. But all its processes are either **compute** or **out**.

In addition to the communication within the main components, the Atmosphere **compute** and the Ocean **compute** have a bidirectional exchange between themselves through YAC.

Definition

```
• yac_c
• yac_c
• yac_c
```



Options

- Creep distance (default: creep_distance: -1)
- Valid range: -1 <= creep_distance

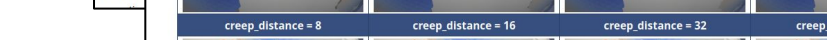
Is the number of iterations for the creep fill algorithm. A creep distance of -1 signals the interpolation to execute the algorithm until no additional target points can be iter this method.

Target field with additional creep fill interpolation (grey cell are not being interpolated)

creep_distance = 0 creep_distance = 1 creep_distance = 2 creep_distance = 4



creep_distance = 8 creep_distance = 16 creep_distance = 32 creep_distance = -1



Initial MPI communicator splitting

- initial communicator splitting is done by YAC using an MPI handshake algorithm
 - <https://gitlab.dkrz.de/dkrz-sw/mpi-handshake>
- processes not using YAC can take part in this splitting by using this algorithm, which is independent from YAC