

# 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

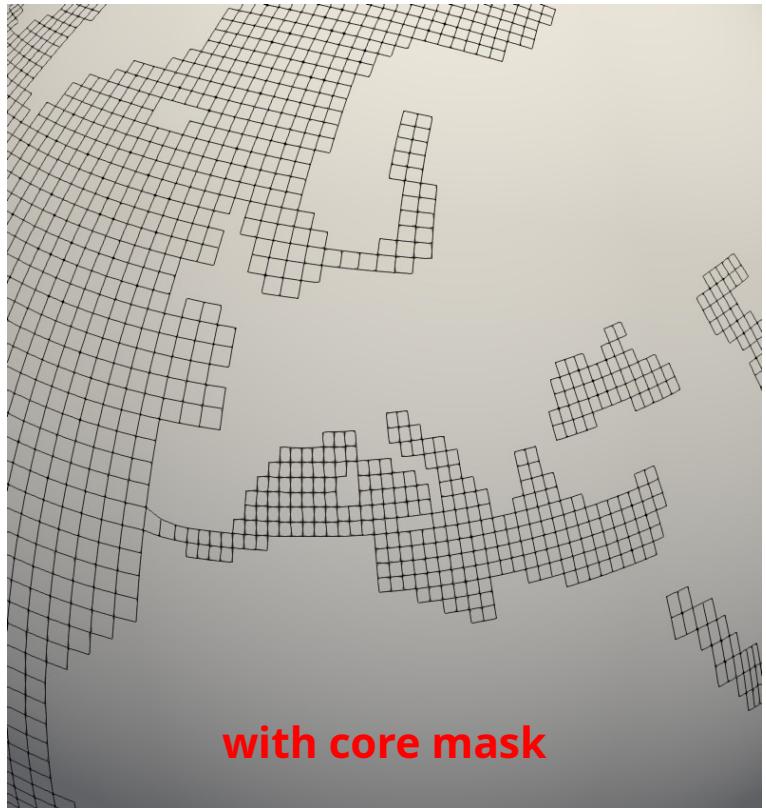
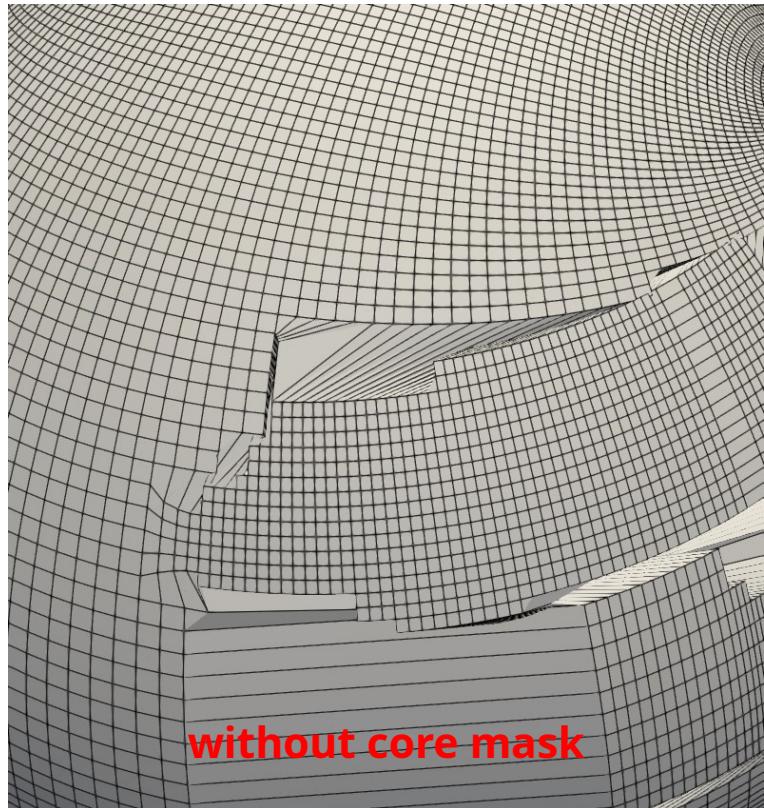
## Field mask

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

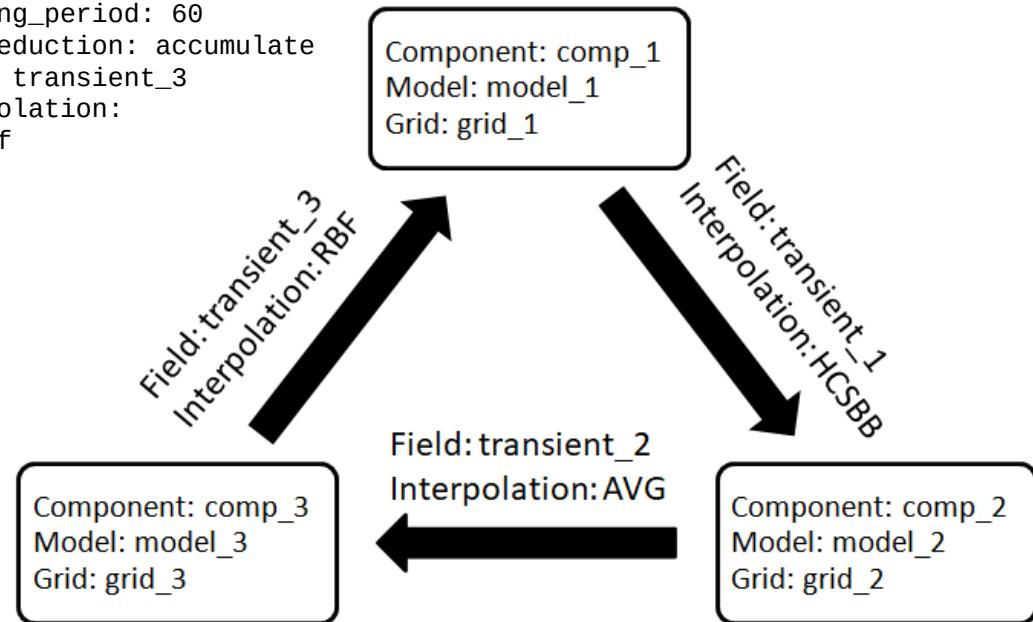
- In atmo/ocean coupling
  - deactivate land points in global atmo grid
- Halos
  - deactivate halos for outgoing fields  
→ send only valid data
  - activate halos for ingoing fields  
→ no halo update required after coupling

# 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://dkrz-sw.gitlab-pages.dkrz.de/yac/dd/dfa/yaml\\_file.html](https://dkrz-sw.gitlab-pages.dkrz.de/yac/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
timestep_unit: second
calendar: proleptic-gregorian
coupling:
# comp_1 -> comp_2
- src_component: comp_1
  src_grid: grid_1
  tgt_component: comp_2
  tgt_grid: grid_2
  coupling_period: 60
  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
# comp_3 -> comp_1
- src_component: comp_3
  src_grid: grid_3
  tgt_component: comp_1
  tgt_grid: grid_1
  coupling_period: 60
  time_reduction: accumulate
  field: transient_3
  interpolation:
    - rbf
```



# Configuration files example

```

definitions:
  atm2oce: &atm2oce
    src_component: atmos
    src_grid: icon_atmos_grid
    tgt_component: ocean
    tgt_grid: icon_ocean_grid
    time_reduction: average
    src_lag: 1
    tgt_lag: 1
  oce2atm: &oce2atm
    src_component: ocean
    src_grid: icon_ocean_grid
    tgt_component: atmos
    tgt_grid: icon_atmos_grid
    time_reduction: average
    src_lag: 1
    tgt_lag: 1
  atm2riv: &atm2riv
    src_component: atmos
    src_grid: icon_atmos_grid
    tgt_component: HD
    tgt_grid: HD_GRID
    time_reduction: average
    src_lag: 1
    tgt_lag: 1
  riv2oce: &riv2oce
    src_component: HD
    src_grid: HD_GRID
    tgt_component: ocean
    tgt_grid: icon_ocean_grid
    time_reduction: average
    src_lag: 1
    tgt_lag: 1

interp_stacks:
  hcsbb_interp_stack: &hcsbb_interp_stack
    interpolation:
      - bernstein_bezier
      - nnn:
          n: 4
          weighted: arithmetic_average
      - fixed:
          user_value: -999.9
  conserv_interp_stack: &conserv_interp_stack
    interpolation:
      - conservative:
          order: 1
          enforced_conservation: false
          partial_coverage: true
          normalisation: fracarea
      - fixed:
          user_value: -999.9
  conserv_interp_dest: &conserv_interp_dest
    interpolation:
      - conservative:
          order: 1
          enforced_conservation: false
          partial_coverage: true
          normalisation: destarea
  spmap_interp_stack: &spmap_interp_stack
    interpolation:
      - source_to_target_map:
          spread_distance: 0.0
          max_search_distance: 0.0
      - fixed:
          user_value: 0.0

timestep_unit: ISO_format
calendar: proleptic-gregorian
coupling:
  - <<: [ *atm2oce, *hcsbb_interp_stack ]
    coupling_period: "PT30M"
    field: [surface_downward_eastward_stress,
            surface_downward_northward_stress]
  - <<: [ *atm2oce, *conserv_interp_stack ]
    coupling_period: "PT30M"
    field: [surface_fresh_water_flux,
            total_heat_flux,
            atmosphere_sea_ice_bundle]
  - <<: [ *oce2atm, *conserv_interp_stack ]
    coupling_period: "PT30M"
    field: [sea_surface_temperature,
            ocean_sea_ice_bundle]
  - <<: [ *atm2riv, *conserv_interp_dest ]
    coupling_period: "P01D"
    field: [surface_water_runoff,
            soil_water_runoff]
  - <<: [ *riv2oce, *spmap_interp_stack ]
    coupling_period: "P01D"
    field: river_runoff

```

# Definition of couples

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

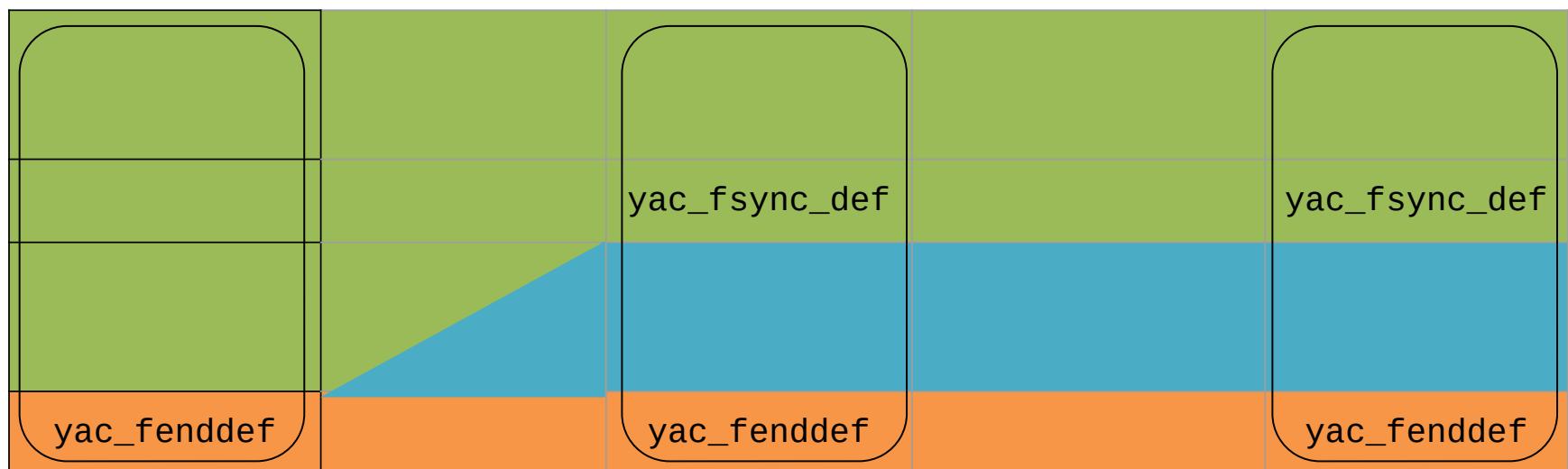
# Definition of couples via user interface

```
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](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_cdef_datetime ( start_datetime, end_datetime );
CALL yac_fdef_datetime ( start_datetime = start_datetime, &
end_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 components 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

Each process can be a communicator part of a component. 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 an algorithm that implements a MPI handshake algorithm that can generate multiple different communicators in a single collective operation. This is the solution we use in the DKRZ coupled run configuration, which implements a MPI handshake algorithm that can generate multiple different communicators in a single collective operation.

As long as DKRZ has MPI Handshake implemented, YAC will use it. Once all components are initialized, MPI Handshake will be called.

### MPI

Once all components are initialized, MPI Handshake will be called. The MPI Handshake will be called for each component.

For each component, MPI Handshake will be called. The MPI Handshake will be called for each component.

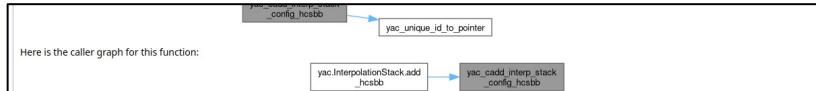
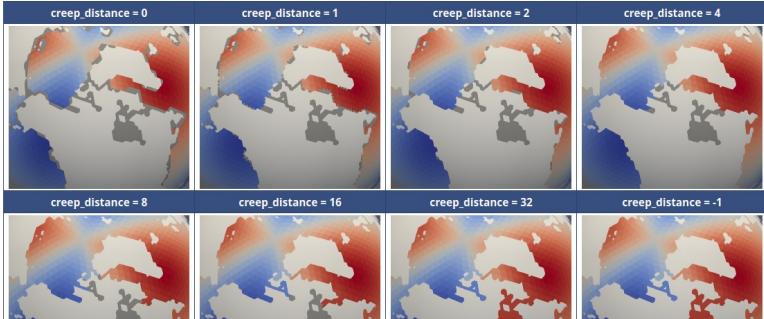
### Options

- 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 interpolated with this method.

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



### yac\_cadd\_interp\_stack\_config\_nnn()

void yac\_cadd\_interp\_stack\_config\_nnn ( int interp\_stack\_config\_id,

int type,

Table of Contents

↳ Introduction

↳ MPI Handshake Interface

↳ Language interoperability

↳ MPI Handshake

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

↳ YAC API

↳ Examples

↳ Interoperability

↳ Type

# Initial MPI communicator splitting

- initials 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