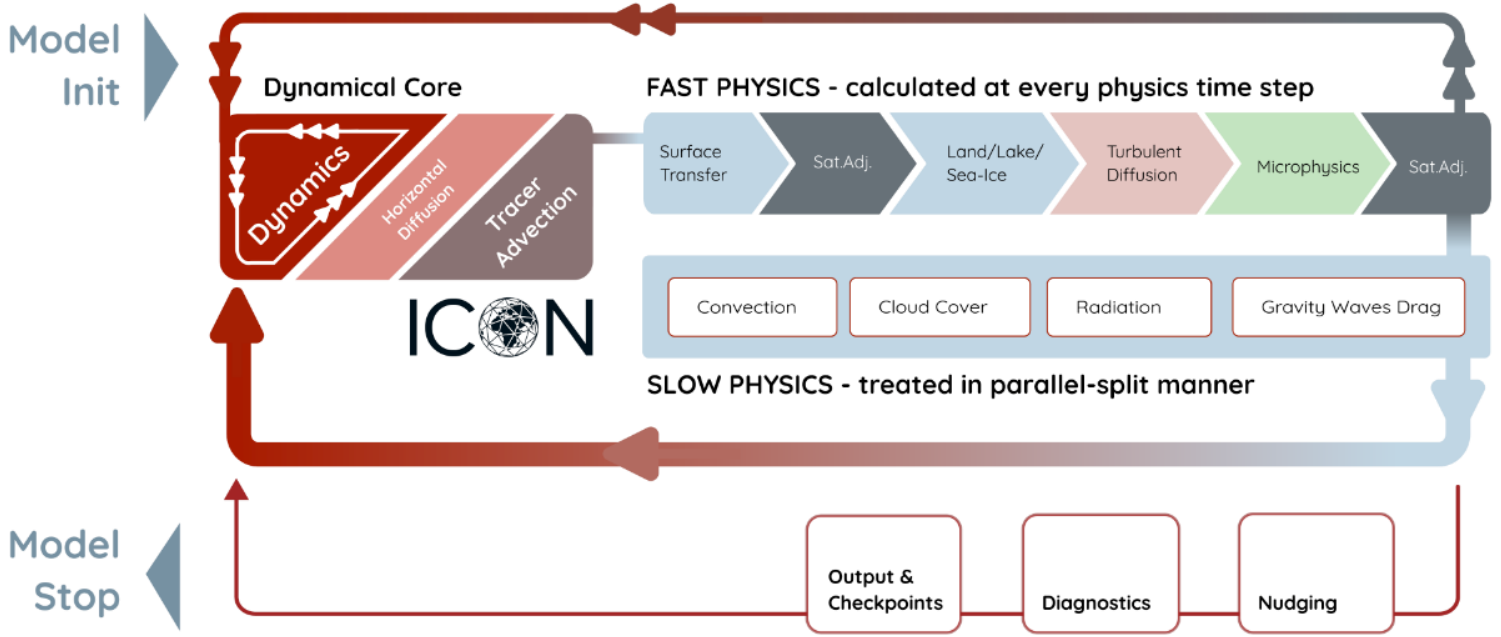
ComIn Plugins

ICON Community Interface

Mahnoosh Haghighatnasab and ComIn Developer Team
DWD, DLR, DKRZ, FZJ | DWD Academic ICON Course |
23 July 2025

ICON Control Flow

- **Release ICON 2024.10** Fortran and C code: **1,123,740** lines
- Where should my implementations go?
- Going through review process
- Difficult to maintain



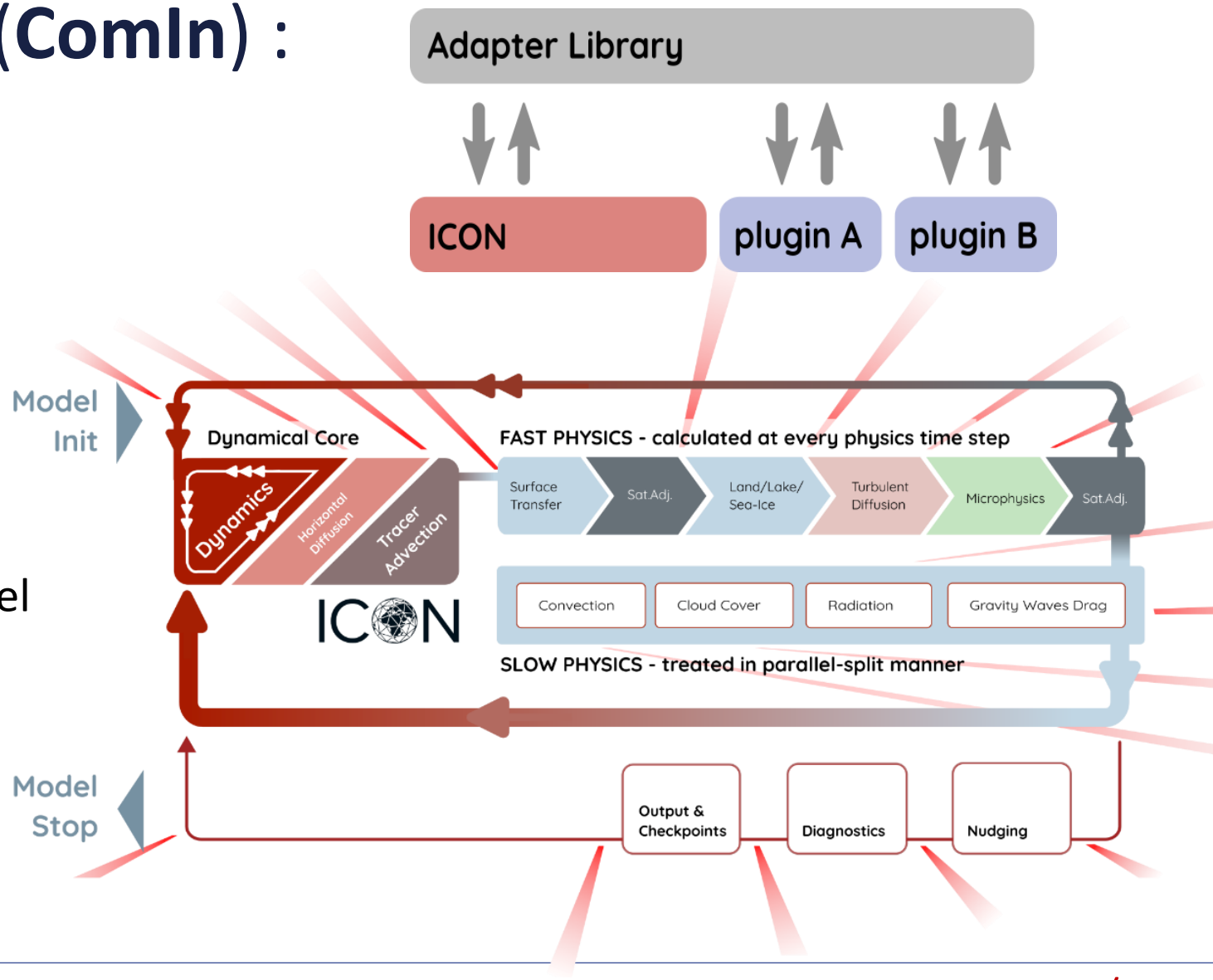
ICON Community Interface (**ComIn**) :

- Connects plugins to the ICON host model

- Plugin functions are called at pre-defined events (Entry Points)

- Regulates the access and creation of model variables

- Guaranties stable interface for external projects



Source: F.Prill (DWD)

Plugin mechanism for ICON

Behind the scenes: dynamic linking

- ComIn relies on dynamic linking to implement the plugin functionality.
- *Dynamic/shared linking:* operating system loads the necessary shared libraries into memory at runtime

Language interoperability in plugins

- **Shared libraries** for Fortran or C/C++ plugins
- **Python plugins** do not need any compilation process

ComIn Plugins Building Blocks

```python
@comin.EP_ATM_WRITE_OUTPUT_BEFORE
def python_diagfct():
    print("diagnostic function called!")
```

append subroutines to a callback register

```python
@comin.EP_ATM_WRITE_OUTPUT_BEFORE
def python_diagfct():
    print("diagnostic function called!")
```

append subroutines to a callback register

```python
@comin.EP_SECONDARY_CONSTRUCTOR
def constructor():
    handle = comin.var_get( [entrypoint], (varname, domain_id),
        comin.COMIN_FLAG_WRITE | comin.COMIN_FLAG_SYNC_HALO
    )
    ...
    np.asarray(handle) = some_value
```

read/write access to model variables

```python
@comin.EP_ATM_WRITE_OUTPUT_BEFORE
def python_diagfct():
    print("diagnostic function called!")
```

append subroutines to a callback register

```python
@comin.EP_SECONDARY_CONSTRUCTOR
def constructor():
    handle = comin.var_get( [entrypoint], (varname, domain_id),
        comin.COMIN_FLAG_WRITE | comin.COMIN_FLAG_SYNC_HALO
    )
    ...
    np.asarray(handle) = some_value
```

read/write access to model variables

```python
comin.var_request_add((varname, domain_id), False)
comin.metadata_set((varname, domain_id), tracer=True)
```

creating additional variables

```python
@comin.EP_ATM_WRITE_OUTPUT_BEFORE
def python_diagfct():
    print("diagnostic function called!")
```

append subroutines to a callback register

```python
@comin.EP_SECONDARY_CONSTRUCTOR
def constructor():
    handle = comin.var_get( [entrypoint], (varname, domain_id),
        comin.COMIN_FLAG_WRITE | comin.COMIN_FLAG_SYNC_HALO
    )
    ...
    np.asarray(handle) = some_value
```

read/write access to model variables

```python
comin.var_request_add((varname, domain_id), False)
comin.metadata_set((varname, domain_id), tracer=True)
```

creating additional variables

```python
domain = comin.descrdata_get_domain(domain_id)
clon = np.asarray(domain.cells.clon)
clat = np.asarray(domain.cells.clat)
```

descriptive data structures

contain information on the ICON setup, the computational grids, and the simulation

- ComIn is an interface not a coupler
- YAC instances coupler can be used in ComIn Plugins

```python
@comin.EP_ATM_YAC_DEFCOMP_AFTER
def yac_define_fields():
    yac_pres_sfc_source = yac.Field.create( ... )
    if rank == 0:
        yac_pres_sfc_target = yac.Field.create( ... )
        yac.def_couple(
            "comin_example_source", "comin_icon_grid", "pres_sfc",
            "comin_example_target", "comin_example_grid", "pres_sfc",
            ... )

@comin.EP_ATM_TIMELOOP_END
def process():
    yac_pres_sfc_source.put(np.ravel(comin_pres_sfc)[: domain.cells.ncells])
    if rank == 0:
        data, info = yac_pres_sfc_target.get()
        plt.imshow(np.reshape(data, [179, 360]))
        plt.savefig(f"pres_sfc.png")
```

Enable the plugin mechanism

- Make sure ICON is configured with `--enable-comin`
- In the Fortran Namelist:

```
&comin_nml
  plugin_list(1)%name           = "comin_plugin"
  plugin_list(1)%plugin_library = "libpython_adapter.so"
  plugin_list(1)%options        = "comin_plugin.py"
/
```

multiple plugins can be added

embedded Python interpreter

filename of Python script passed as an option

Entry point implementation

- Currently 41 entrypoints implemented in ICON
- New entry points can be easily introduced:
  `CALL icon_call_callback(…)`

- **Granularity**: above block loop level, only global variables are exposed, only process-local partitions can be accessed directly

- **ICON calls ComIn, not vice versa!**
  processes in the host model are not switched off, but can only be deactivated using ICON namelist switches

Project history and status

2022

Project started as a collaboration:
DWD, **DLR-IPA, DKRZ** and **FZJ**

2023

Design white paper and mock-up completed

Jan 2024

ComIn v0.1.0 is part of the first **ICON Open Source Release**

Oct 2024

ComIn v0.2.0 released

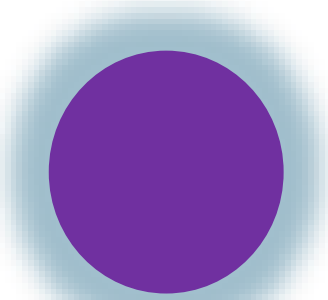
April 2025

Release ComIn **v0.3.0** (adds data types, C++ refactoring, …)

Project started as a collaboration:

DWD, **DLR-IPA, DKRZ** and **FZJ**

2022

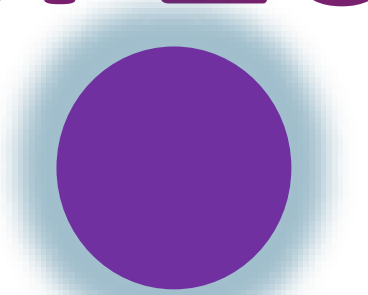


2023

Design white paper and mock-up completed

ComIn v0.1.0 is part of the first **ICON Open Source Release**

Jan 2024

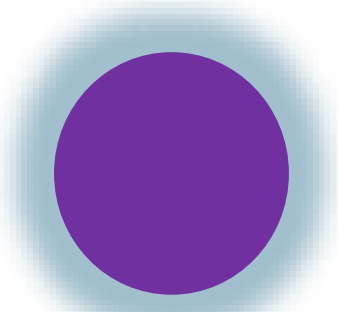


Oct 2024

ComIn v0.2.0 released

Release ComIn **v0.3.0**
(adds data types, C++
refactoring, …)

April 2025



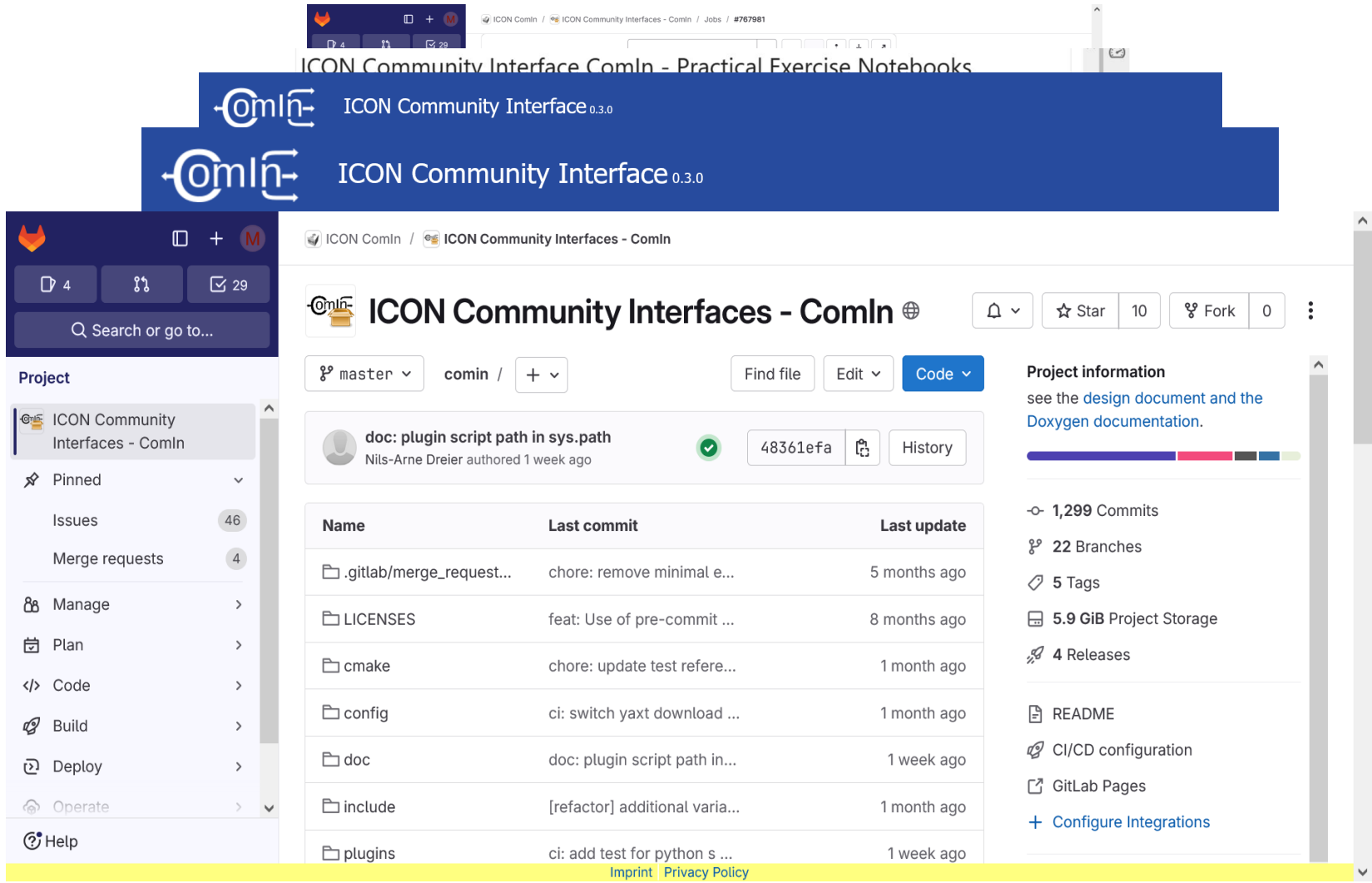
Key Resources

Key Resources

GitLab-Project

https://gitlab.dkrz.de/icon-comin/comin

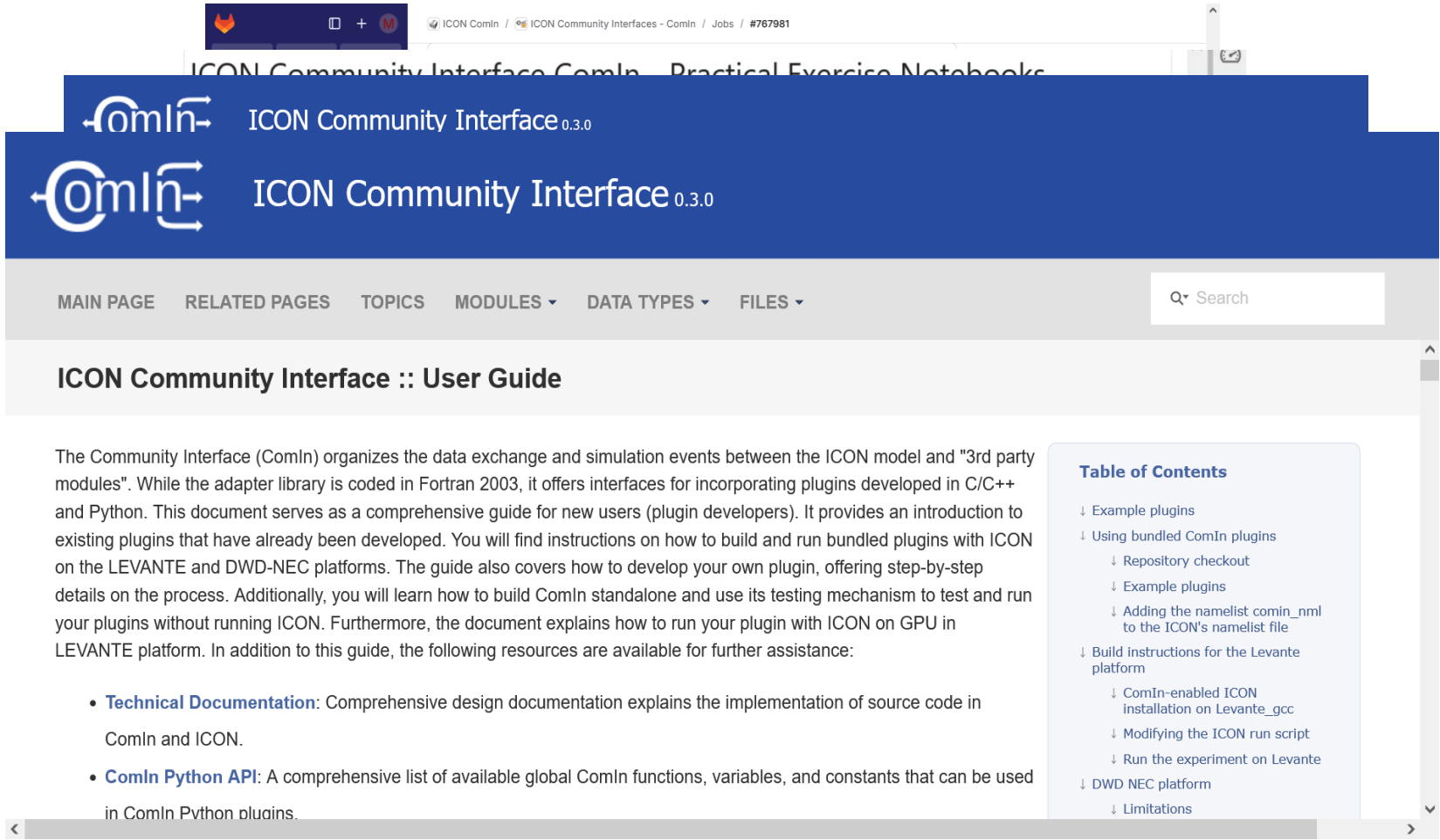
- Source code and release notes
- Plugin examples
- Tests



Key Resources

Extensive documentation

https://icon-comin.gitlab-pages.dkrz.de/comin/



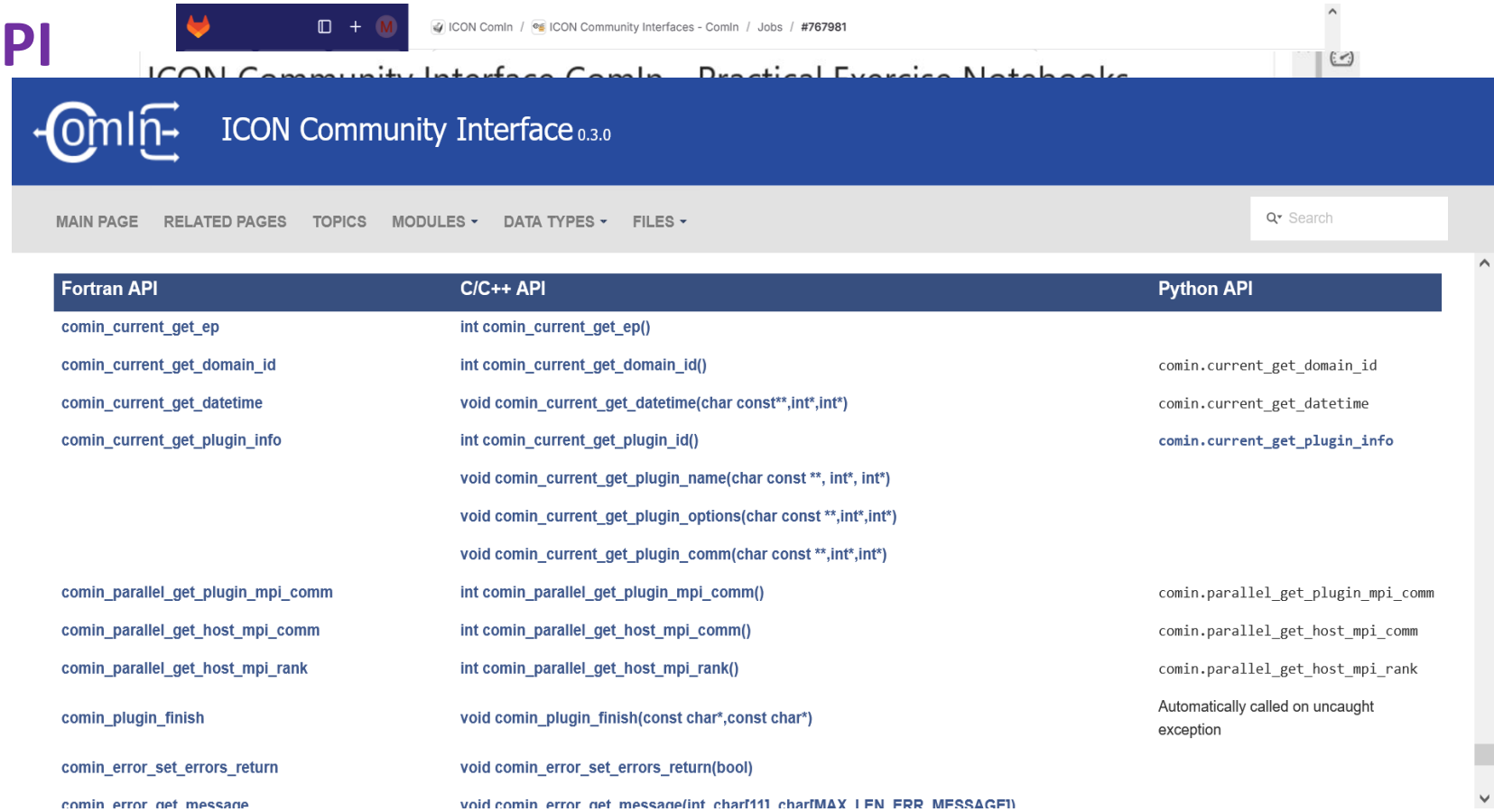
- User guide
- Design document
- Developer document
- Doxygen page
- ICON ComIn paper

Hartung, K.,et al. (2025). ICON ComIn–the ICON Community Interface. *Geoscientific Model Development*, *18*(4), 1001-1015.

Key Resources

Fortran, C/C++ and Python API

- Equivalent interfaces for plugins:
  - Python
  - Fortran
  - C/C++.



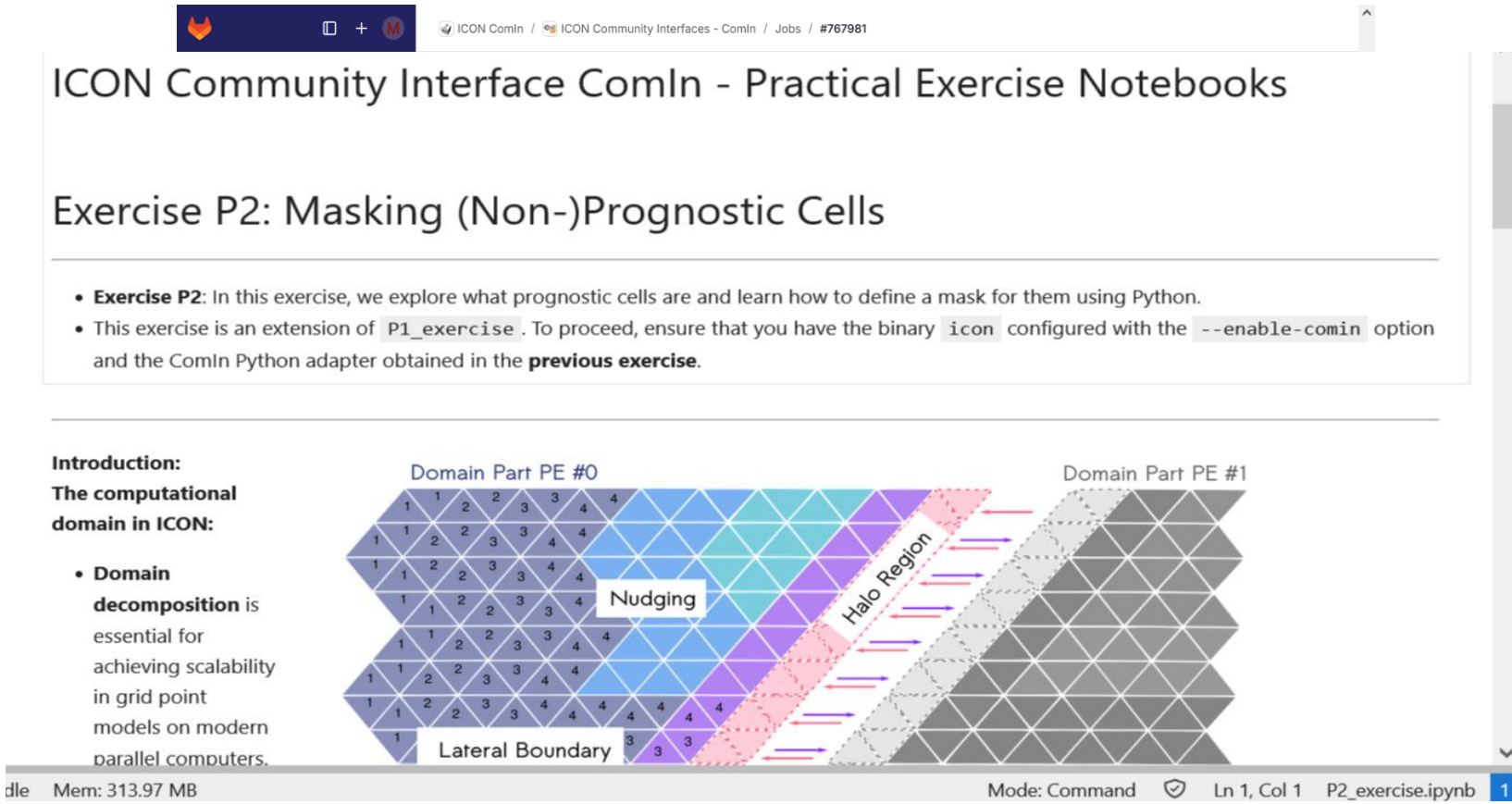
Fortran API	C/C++ API	Python API
comin_current_get_ep	int comin_current_get_ep()	
comin_current_get_domain_id	int comin_current_get_domain_id()	comin.current_get_domain_id
comin_current_get_datetime	void comin_current_get_datetime(char const**,int*,int*)	comin.current_get_datetime
comin_current_get_plugin_info	int comin_current_get_plugin_id()	comin.current_get_plugin_info
	void comin_current_get_plugin_name(char const **, int*, int*)	
	void comin_current_get_plugin_options(char const **,int*,int*)	
	void comin_current_get_plugin_comm(char const **,int*,int*)	
comin_parallel_get_plugin_mpi_comm	int comin_parallel_get_plugin_mpi_comm()	comin.parallel_get_plugin_mpi_comm
comin_parallel_get_host_mpi_comm	int comin_parallel_get_host_mpi_comm()	comin.parallel_get_host_mpi_comm
comin_parallel_get_host_mpi_rank	int comin_parallel_get_host_mpi_rank()	comin.parallel_get_host_mpi_rank
comin_plugin_finish	void comin_plugin_finish(const char*,const char*)	Automatically called on uncaught exception
comin_error_set_errors_return	void comin_error_set_errors_return(bool)	
comin_error_get_message	void comin_error_get_message(int, char[11], char[MAX_LEN_ERR_MESSAGE])	

Key Resources

ComIn Exercise Notebooks

https://gitlab.dkrz.de/icon-comin/comin-training-exercises

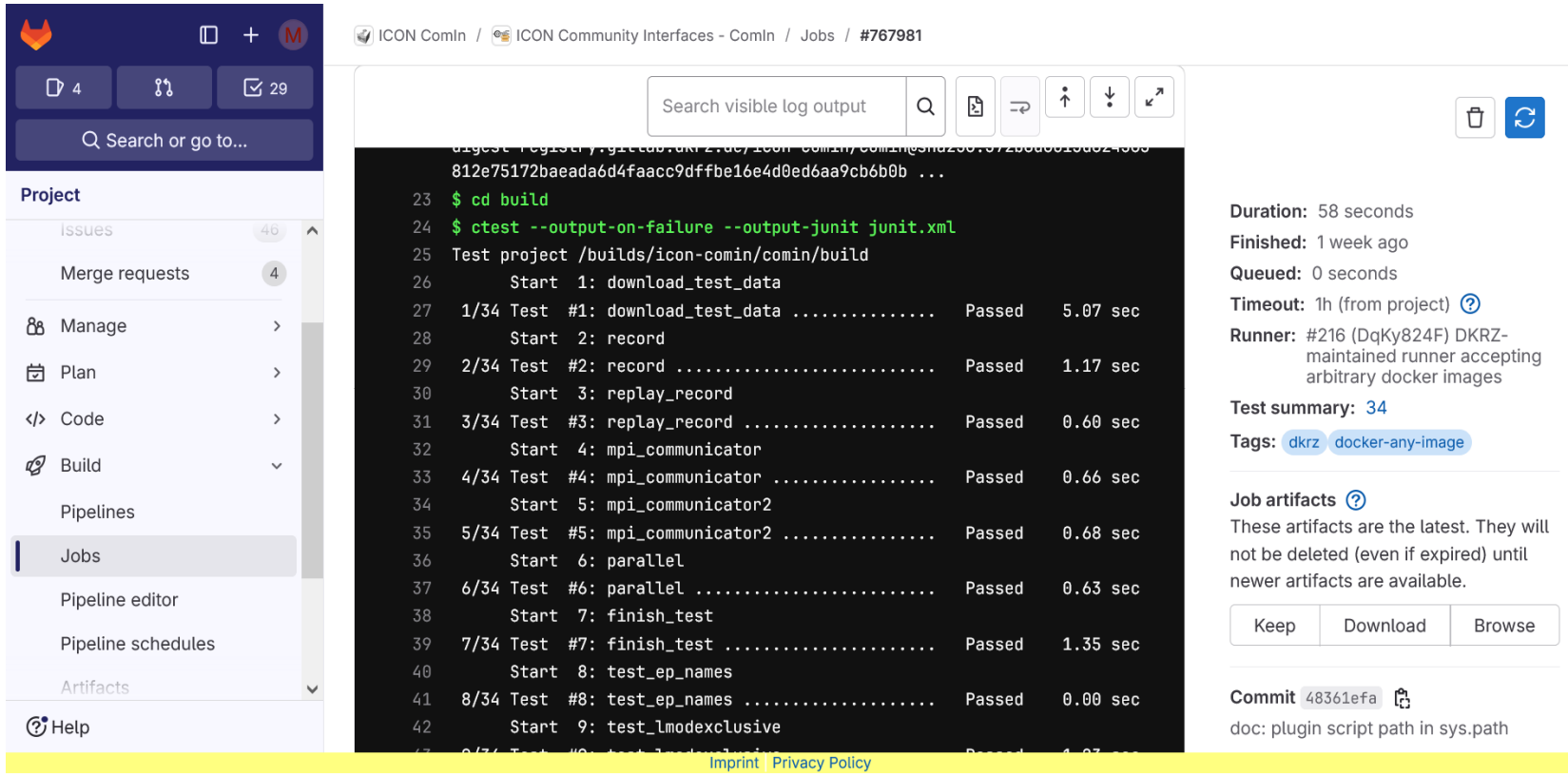
- Prepared for Levante platform

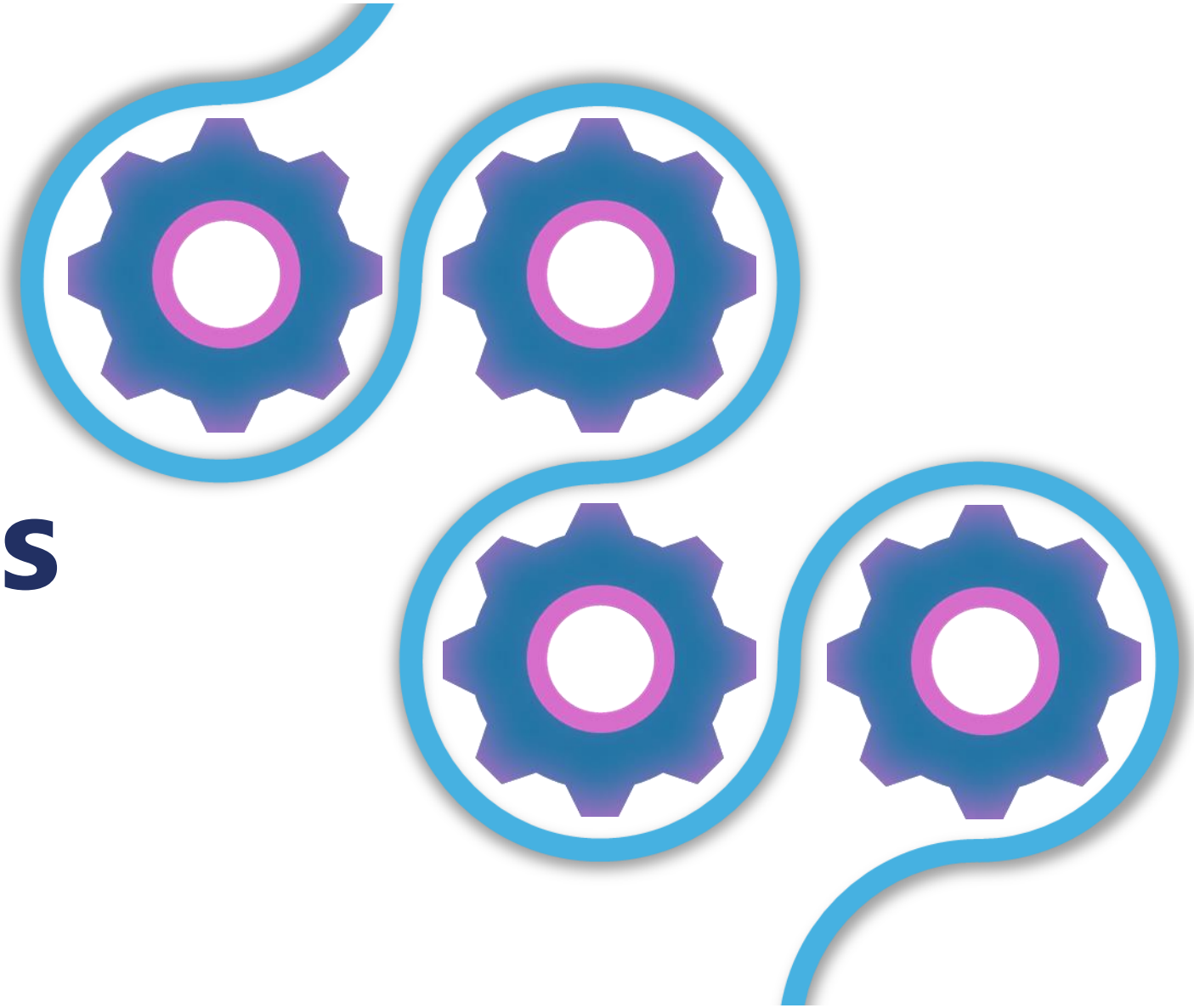


Key Resources

record & replay tool

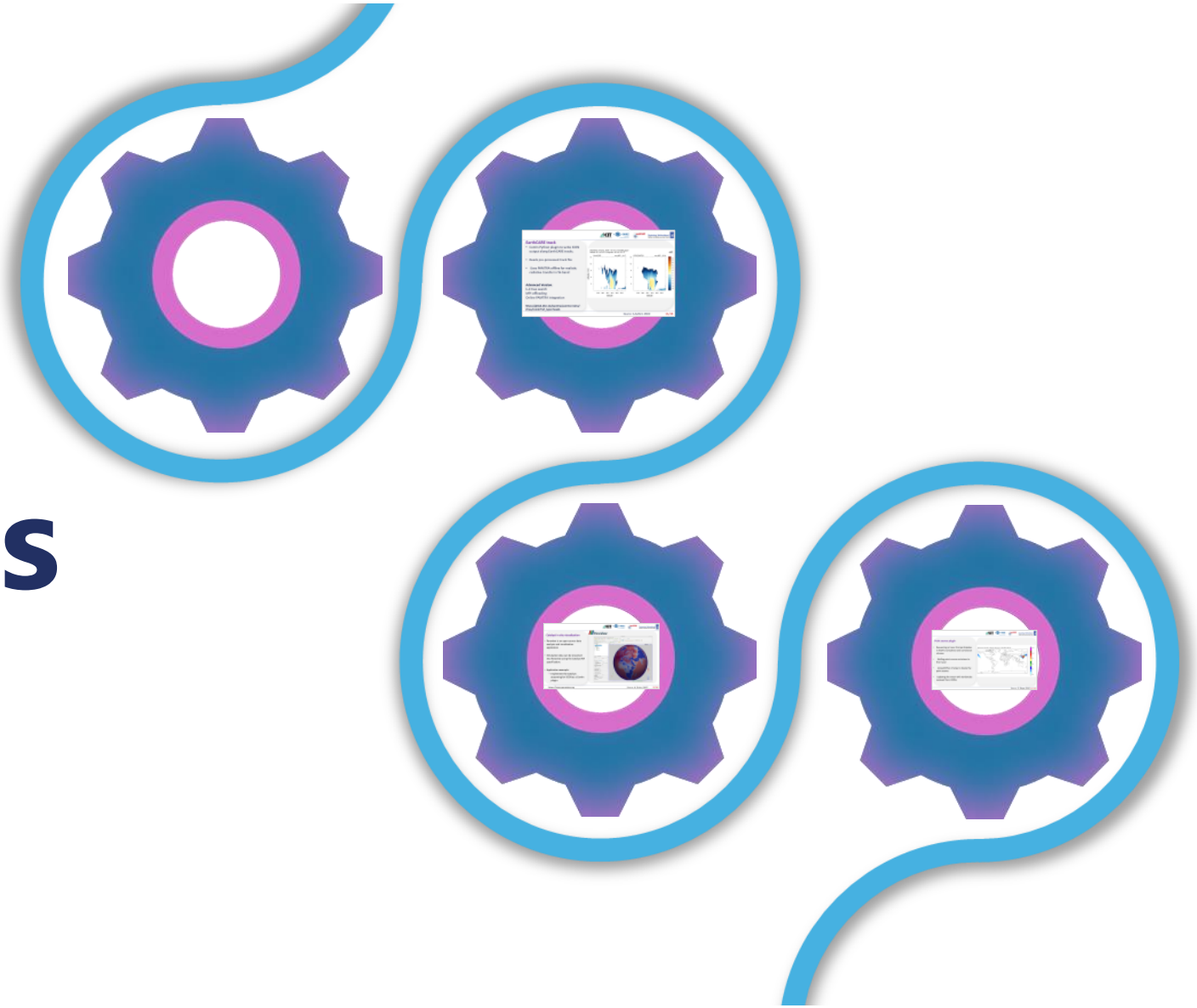
- Makes use of previously recorded ICON datasets
- **record & replay is** very helpful for developing plugins
- It can act as a **host model**
- Run plugin with *ctest* functionality





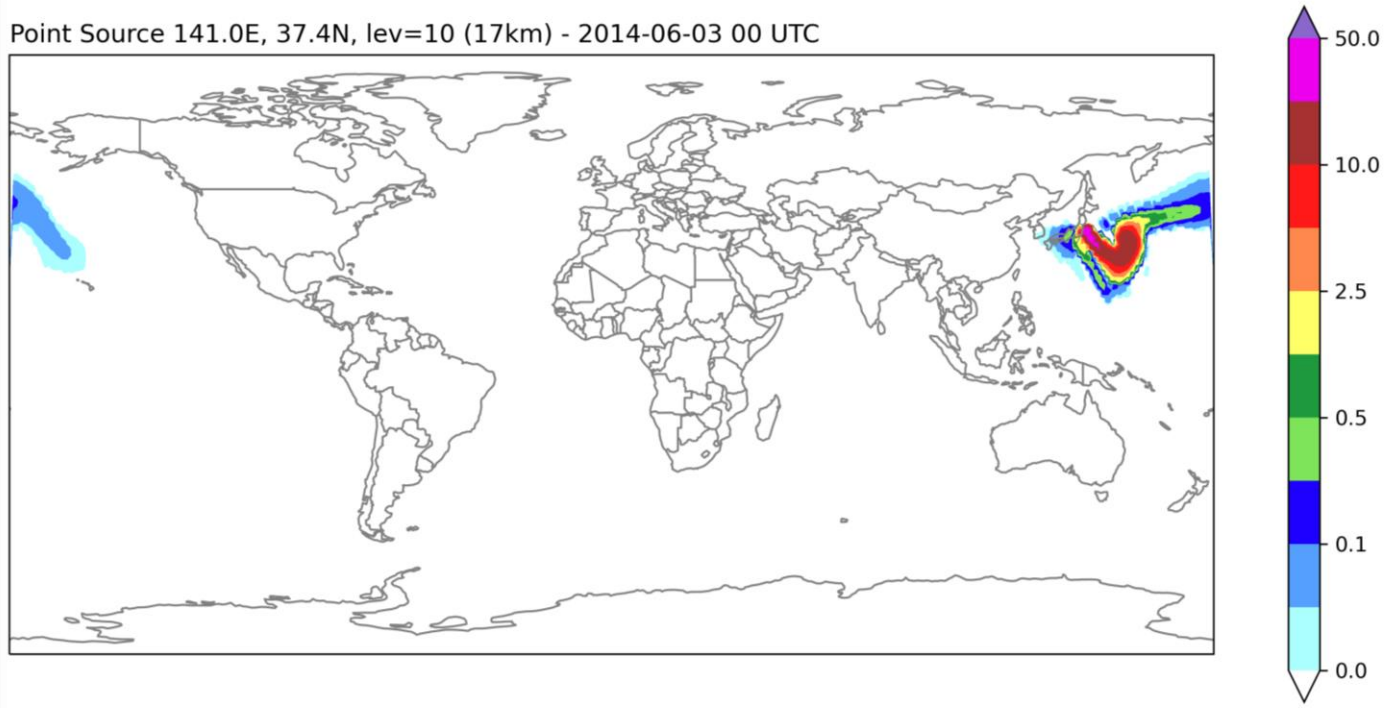
Applications

Applications



Point source plugin

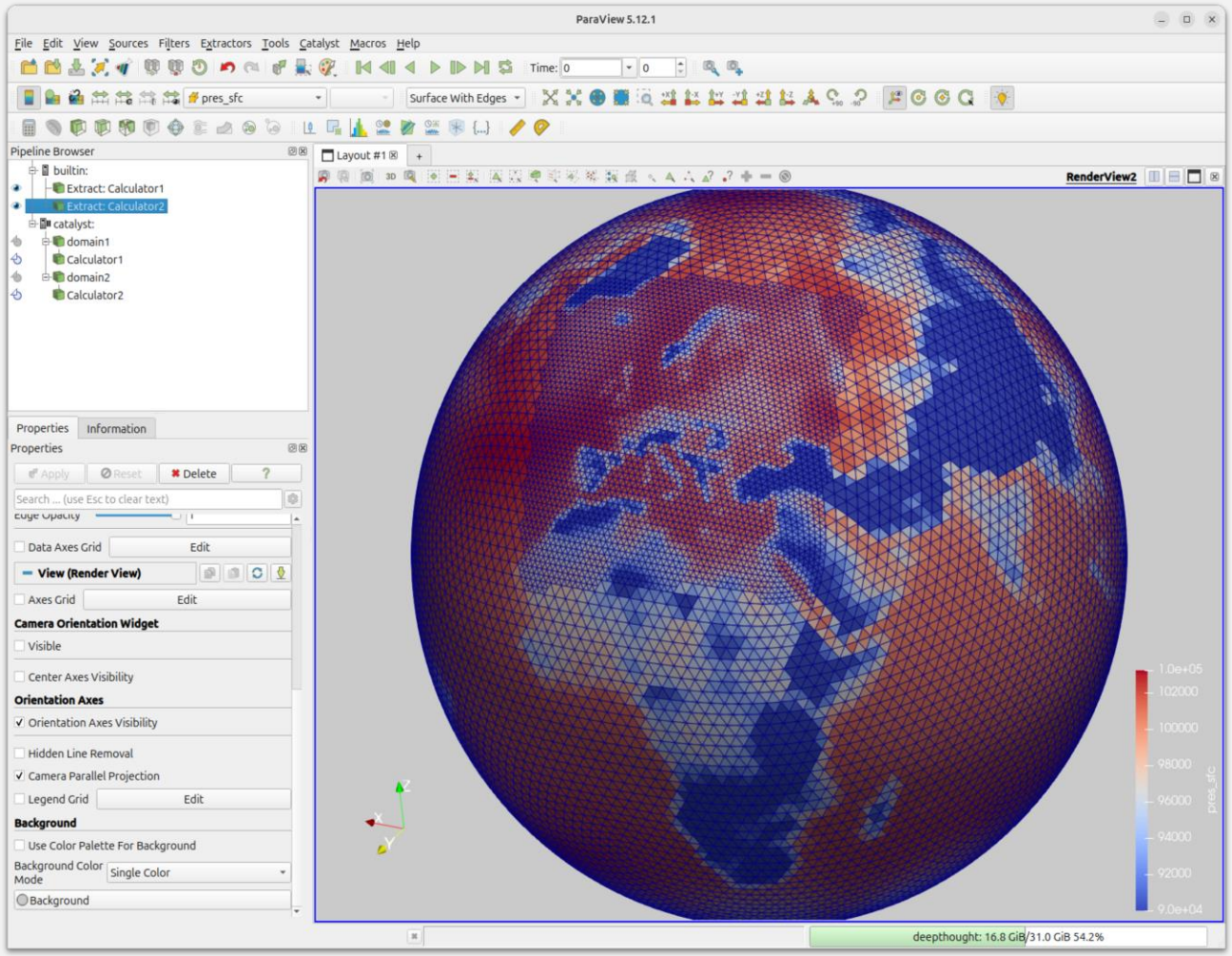
- Requesting a tracer that participates in ICON's turbulence and convection scheme
- Adding point source emissions to this tracer
- Using *KDTree* of *scipy* to locate the point source
- Updating the tracer with tendencies received from ICON's



Source : D. Rieger, DWD

Catalyst in situ visualization

- Paraview is an open-source data analysis and visualization application
- Simulation data can be streamed into Paraview using the Catalyst API specification.
- Application example:
  - Implement the Catalyst streaming for ICON as a ComIn plugin.



https://www.paraview.org

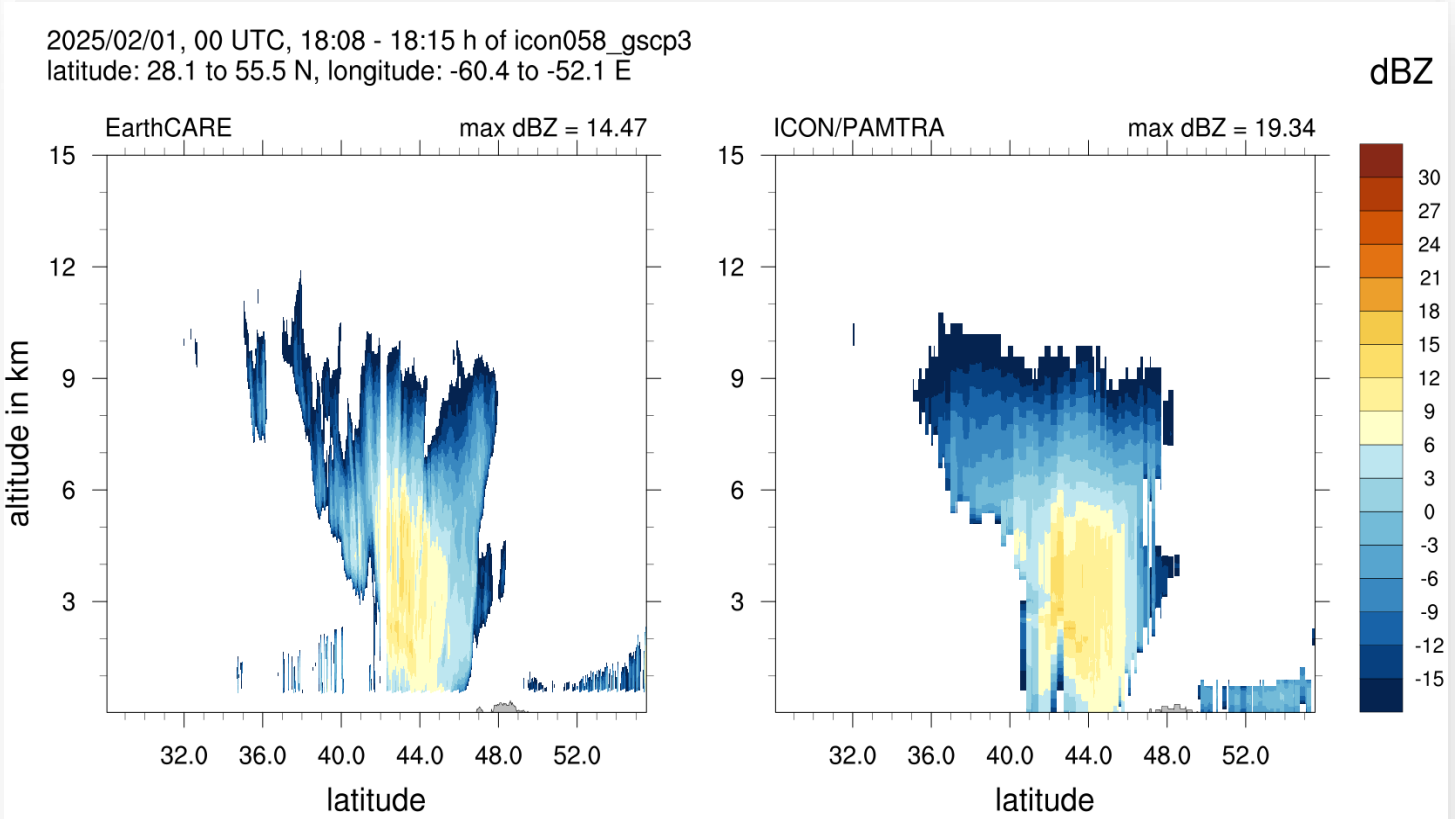
Source: N. Dreier, DKRZ

EarthCARE track

- ComIn Python plugin to write ICON output along EarthCARE tracks.
- Reads pre-processed track file
- Uses PAMTRA offline for realistic radiative transfer in W-band

Advanced Version:
k-d tree search
MPI offloading
Online PAMTRA integration

https://gitlab.dkrz.de/pamtra/pamtra-insitu/-/tree/comin?ref_type=heads



Source: A.Seifert, DWD

Team Members

Deutscher Wetterdienst (DWD)

Florian Prill

Mahnoosh Haghighatnasab

Daniel Rieger

Deutsches Zentrum für Luft- und Raumfahrt (DLR)

Bastian Kern

Kerstin Hartung

Patrick Jöckel

Deutsches Klimarechenzentrum (DKRZ)

Nils-Arne Dreier

Lakshmi Aparna Devulapalli

Wilton Jaciel Loch

Forschungszentrum Jülich (FZJ)

Astrid Kerkweg

Thank You

Mahnoosh Haghighatnasab
Deutscher Wetterdienst
E-mail:
mahnoosh.haghighatnasab@dwd.de

Contact: comin@icon-model.org

Further applications

- Initicon demonstrator Python plugin replace ICON's initialization module
- Machine learning applications
  → Open ICON project 2024 – 2027
- Messy