

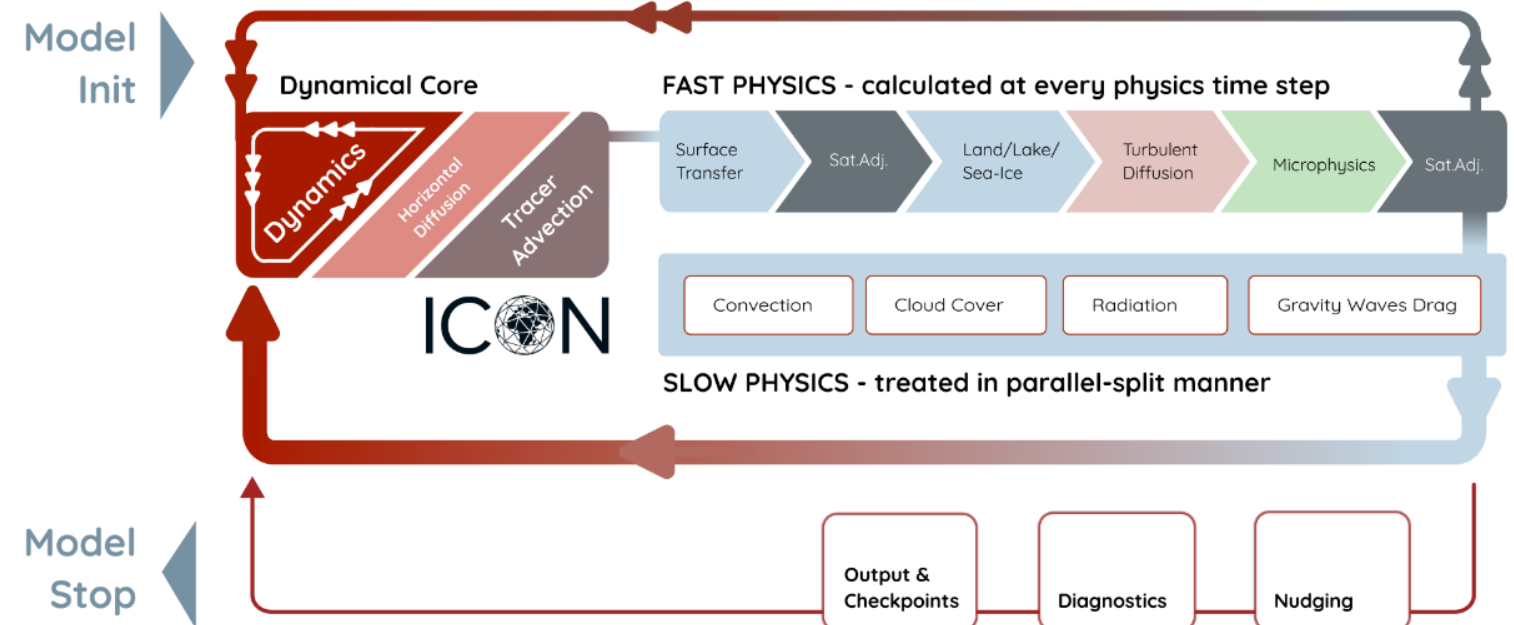
ComIn Plugins

ICON Community Interface

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

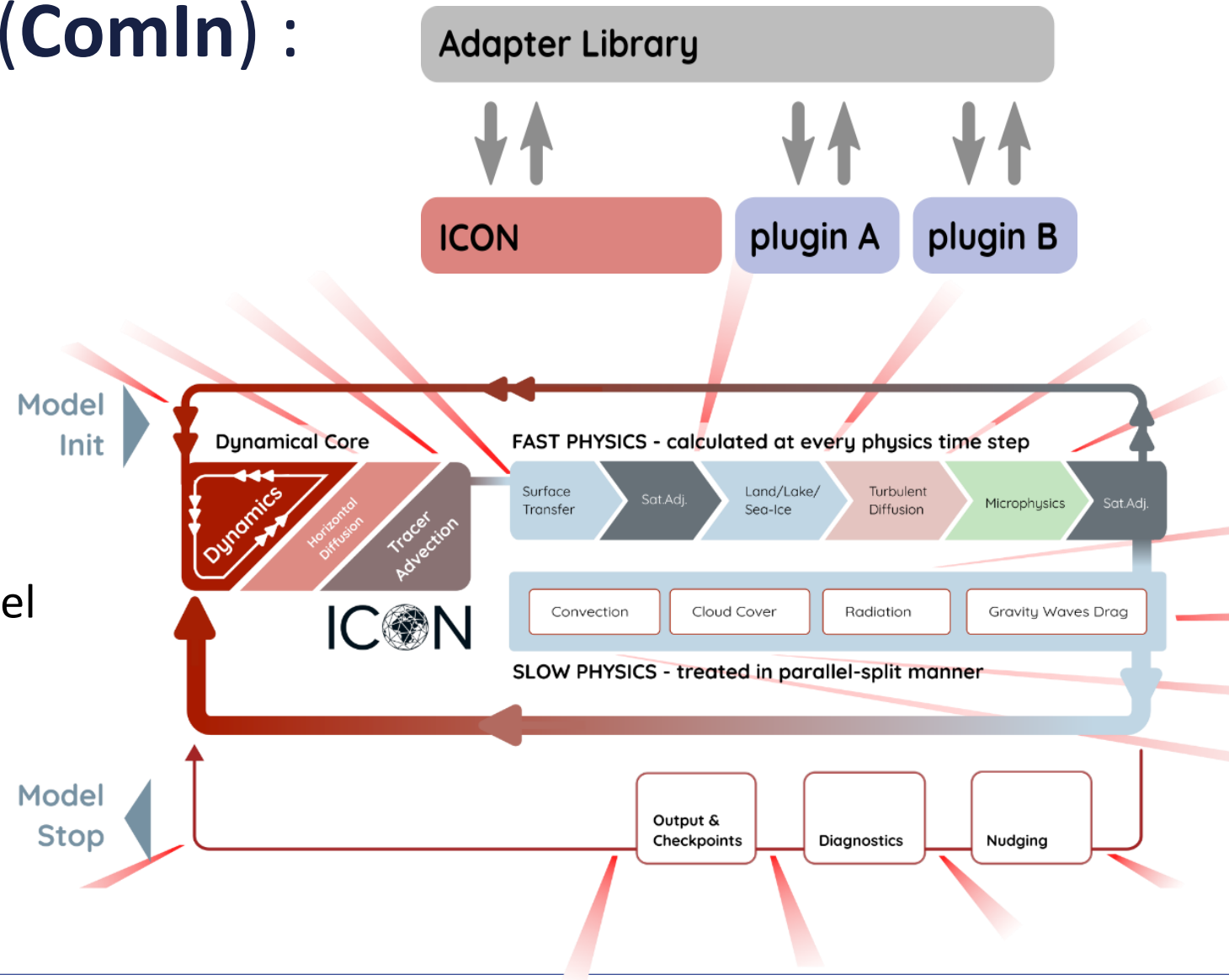
ICON Control Flow

- **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



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

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

append subroutines to a callback register

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

append subroutines to a callback register

```
@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

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

append subroutines to a callback register

```
@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

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

creating additional variables


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

append subroutines to a callback register

```
@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

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

creating additional variables

```
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

```
@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



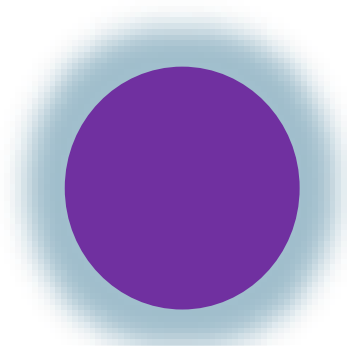
Project history and status

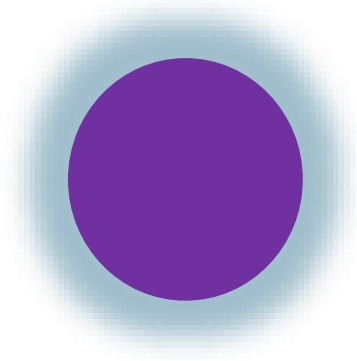


Project started as a
collaboration:

DWD, DLR-IPA, DKRZ and FZJ

2022



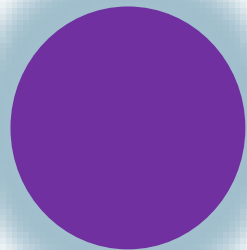


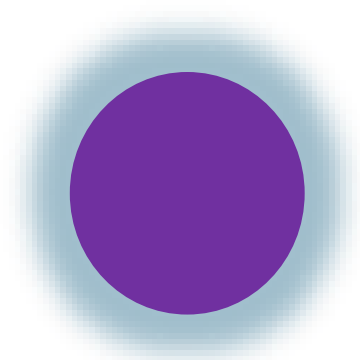
2023

Design white paper and
mock-up completed

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

Jan 2024



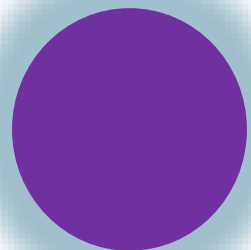


Oct 2024

ComIn v0.2.0 released

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

April 2025



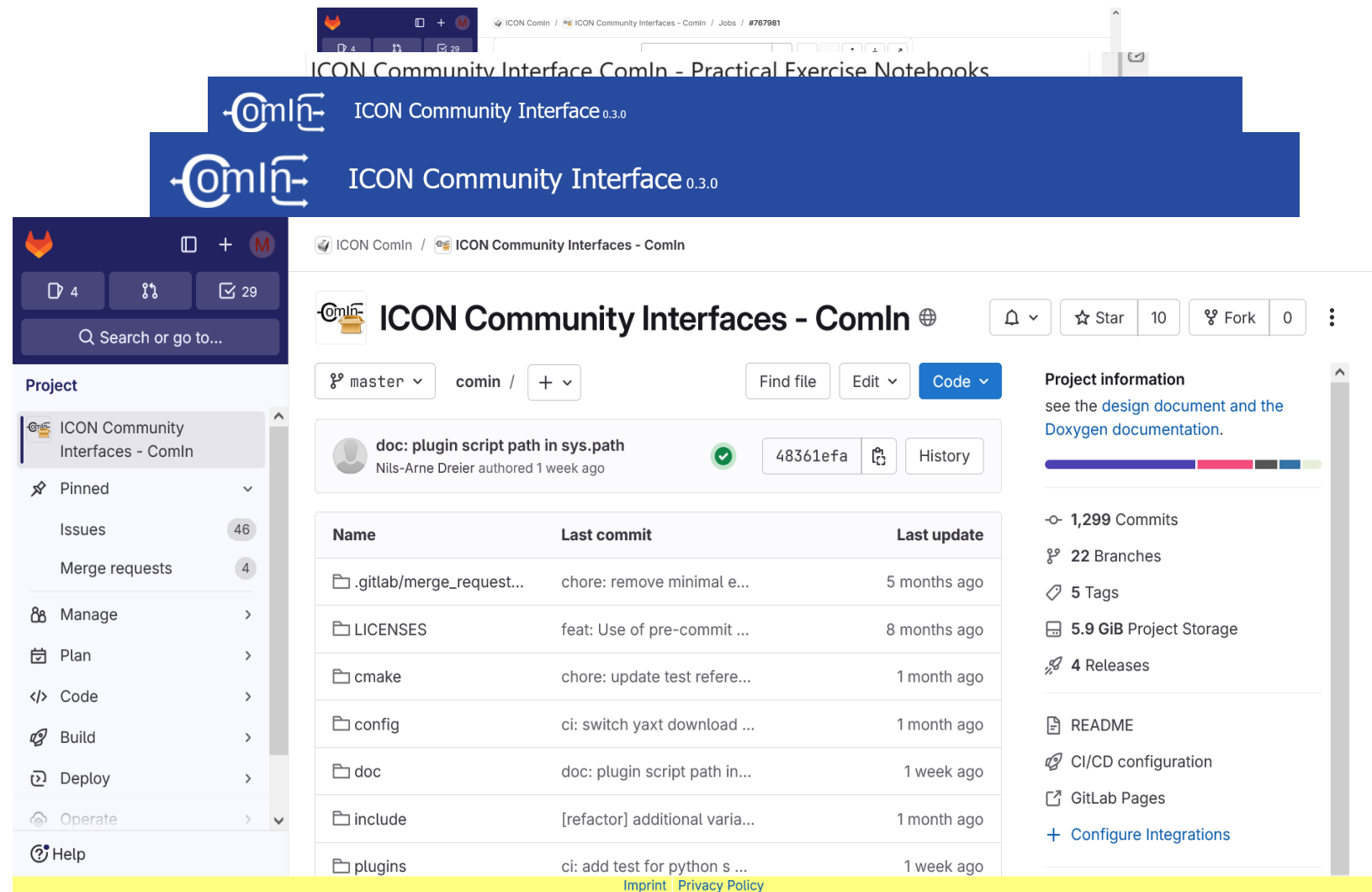
Key Resources

Key Resources

GitLab-Project

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

- Source code and release notes
- Plugin examples
- Tests



The screenshot shows the GitLab web interface for the project 'ICON Community Interfaces - ComIn'. The left sidebar contains navigation links: Project, Pinned, Issues (46), Merge requests (4), Manage, Plan, Code, Build, Deploy, Operate, and Help. The main content area displays the project name, a dropdown for the 'master' branch, and a list of files with their last commit and update dates. A specific commit 'doc: plugin script path in sys.path' by Nils-Arne Dreier is highlighted. The right sidebar provides project statistics and links to documentation and integrations.

Name	Last commit	Last update
.gitlab/merge_request...	chore: remove minimal e...	5 months ago
LICENSES	feat: Use of pre-commit ...	8 months ago
cmake	chore: update test refere...	1 month ago
config	ci: switch yaxt download ...	1 month ago
doc	doc: plugin script path in...	1 week ago
include	[refactor] additional varia...	1 month ago
plugins	ci: add test for python s ...	1 week ago

Project information:
 see the [design document](#) and the [Doxygen documentation](#).

1,299 Commits
 22 Branches
 5 Tags
 5.9 GiB Project Storage
 4 Releases

README
 CI/CD configuration
 GitLab Pages
[Configure Integrations](#)

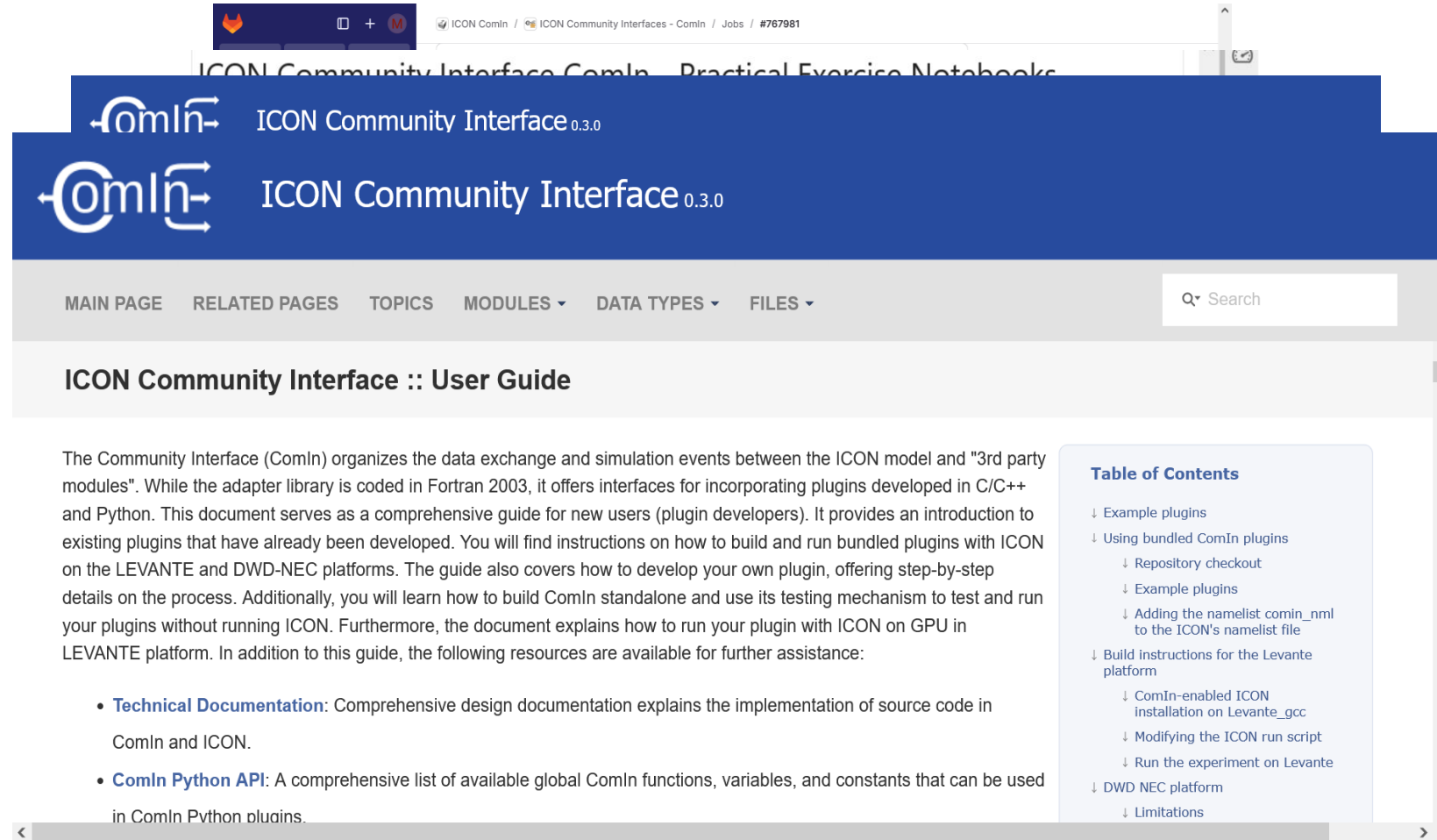
Key Resources

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.

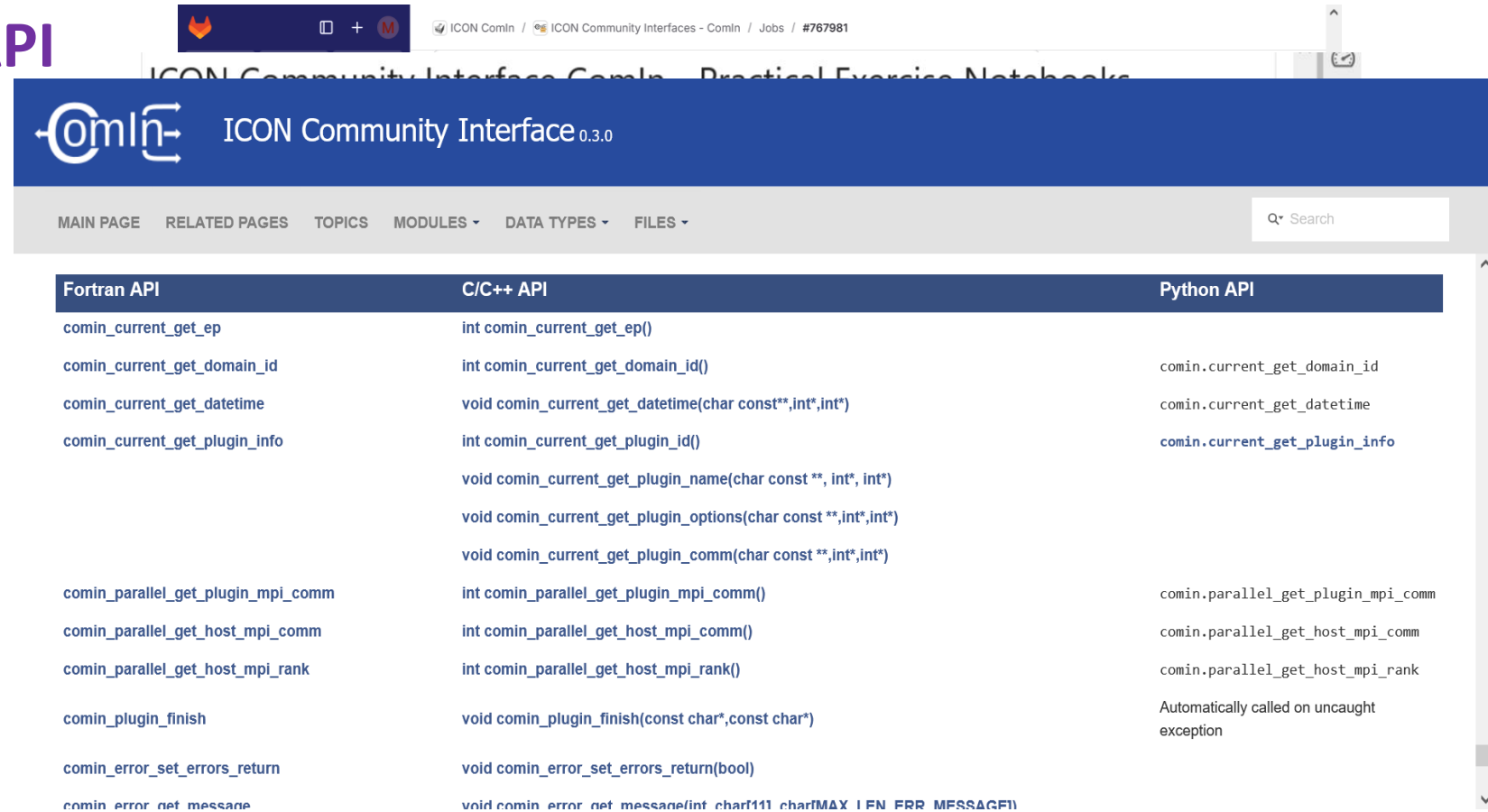


The screenshot shows the web interface for the ICON Community Interface (ComIn) version 0.3.0. The header is dark blue with the ComIn logo and version number. Below the header is a navigation bar with links: MAIN PAGE, RELATED PAGES, TOPICS, MODULES, DATA TYPES, and FILES. A search bar is on the right. The main content area is titled "ICON Community Interface :: User Guide". It contains a paragraph introducing the ComIn interface and its purpose. To the right of the main text is a "Table of Contents" sidebar with links to various sections: Example plugins, Using bundled ComIn plugins, Repository checkout, Example plugins, Adding the namelist comin_nml to the ICON's namelist file, Build instructions for the Levante platform, ComIn-enabled ICON installation on Levante_gcc, Modifying the ICON run script, Run the experiment on Levante, DWD NEC platform, and Limitations.

Key Resources

Fortran, C/C++ and Python API

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



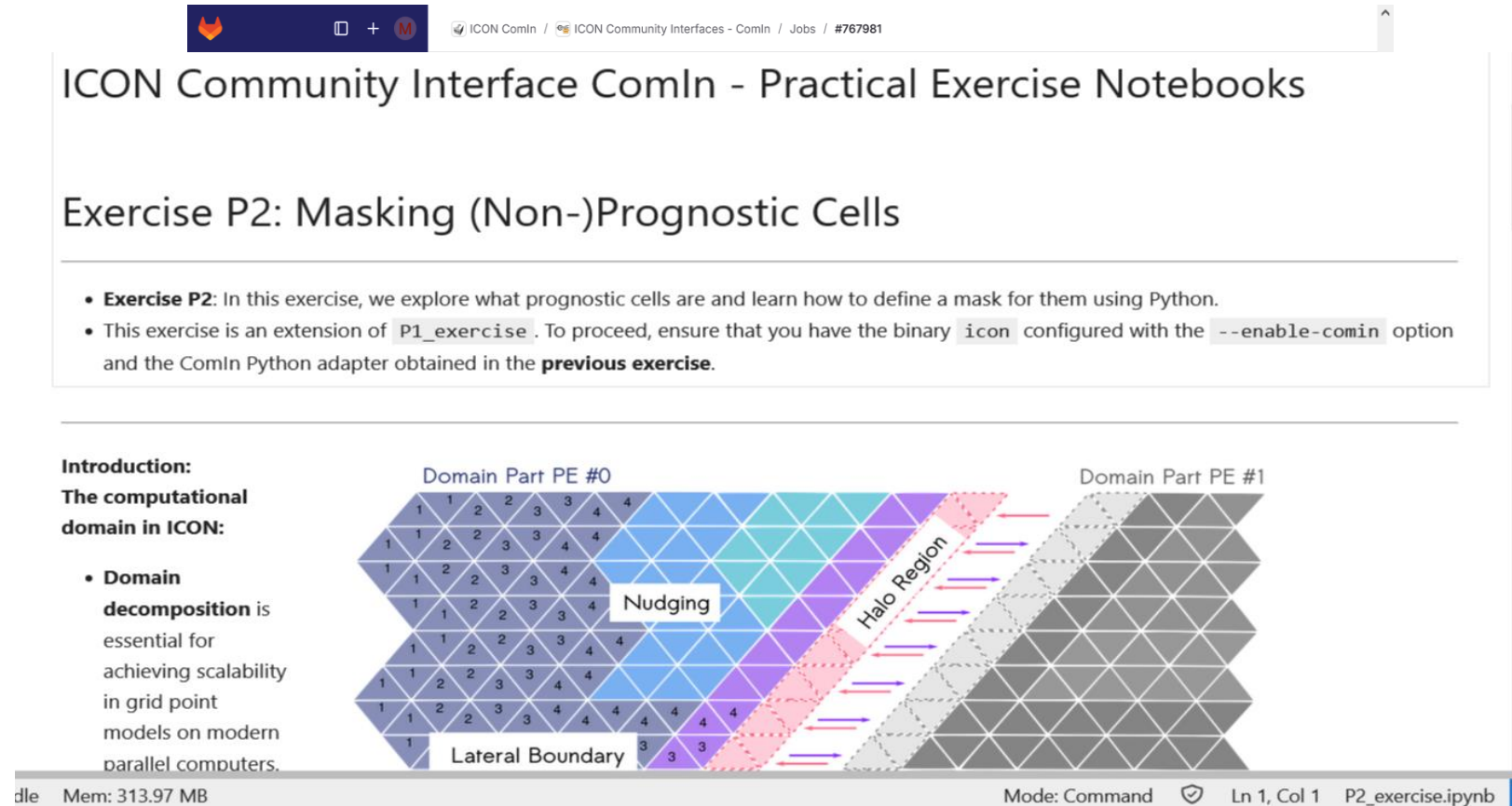
Fortran API	C/C++ API	Python API
comin_current_get_ep	int comin_current_get_ep()	
comin_current_get_domain_id	int comin_current_get_domain_id()	comin.current_get_domain_id
comin_current_get_datetime	void comin_current_get_datetime(char const**,int*,int*)	comin.current_get_datetime
comin_current_get_plugin_info	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



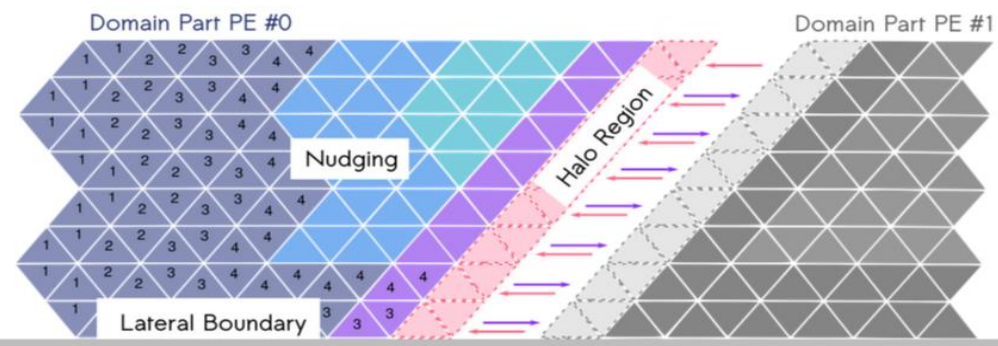
ICON Community Interface ComIn - Practical Exercise Notebooks

Exercise P2: Masking (Non-)Prognostic Cells

- **Exercise P2:** In this exercise, we explore what prognostic cells are and learn how to define a mask for them using Python.
- This exercise is an extension of `P1_exercise`. To proceed, ensure that you have the binary `icon` configured with the `--enable-comin` option and the ComIn Python adapter obtained in the **previous exercise**.

Introduction:
The computational domain in ICON:

- **Domain decomposition** is essential for achieving scalability in grid point models on modern parallel computers.



Domain Part PE #0

Domain Part PE #1

Nudging

Halo Region

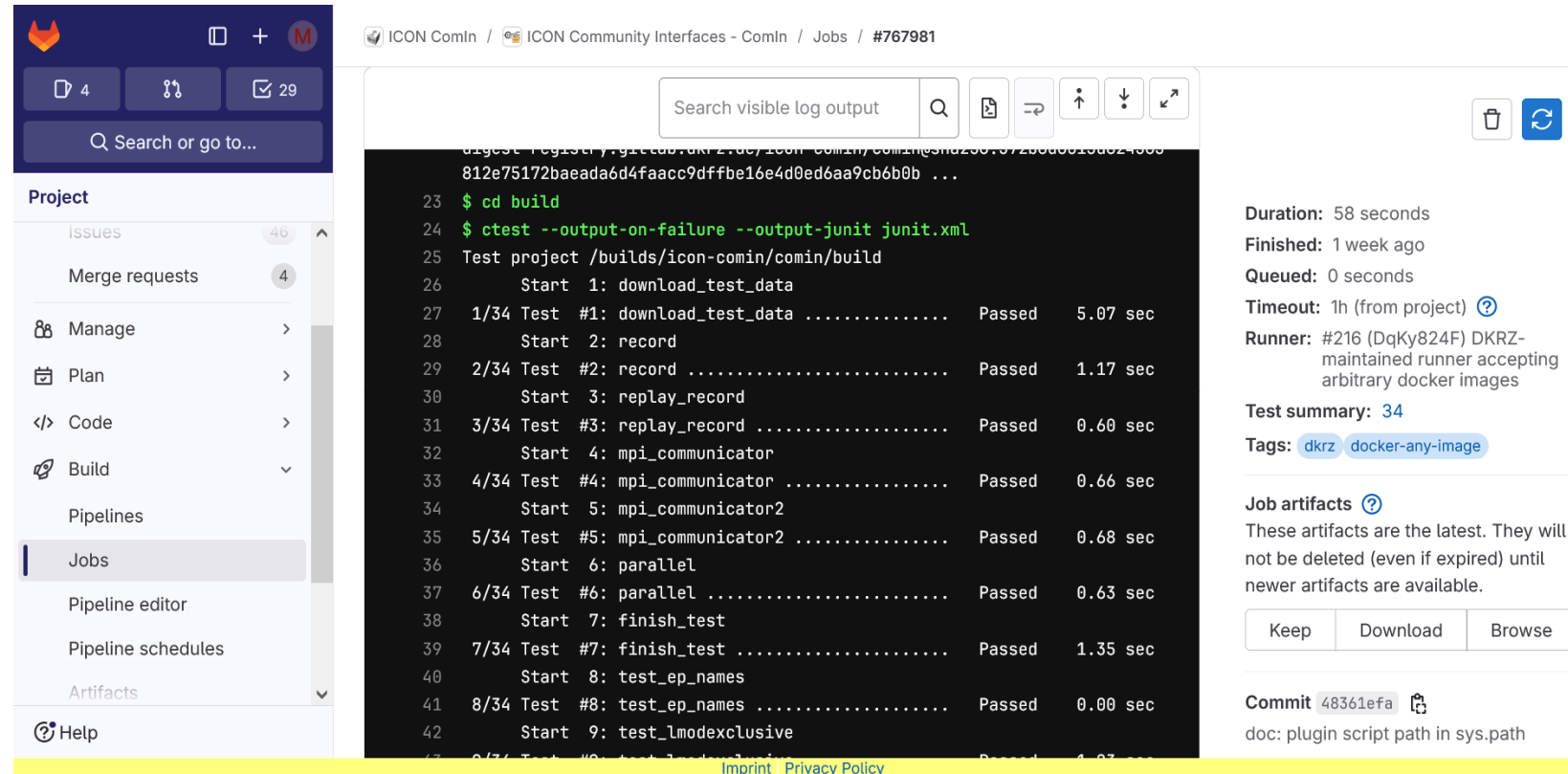
Lateral Boundary

dle Mem: 313.97 MB Mode: Command Ln 1, Col 1 P2_exercise.ipynb 1

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



The screenshot displays the ICON Comin web interface for job #767981. The left sidebar shows navigation options: Project, Issues (46), Merge requests (4), Manage, Plan, Code, Build, Pipelines, Jobs (selected), Pipeline editor, Pipeline schedules, Artifacts, and Help. The main content area shows the job log with a search bar and various icons. The log output includes commands like `$ cd build`, `$ ctest --output-on-failure --output-junit junit.xml`, and a list of test results. The right sidebar provides a summary of the job, including duration, finish time, runner information, and tags.

Job Summary:

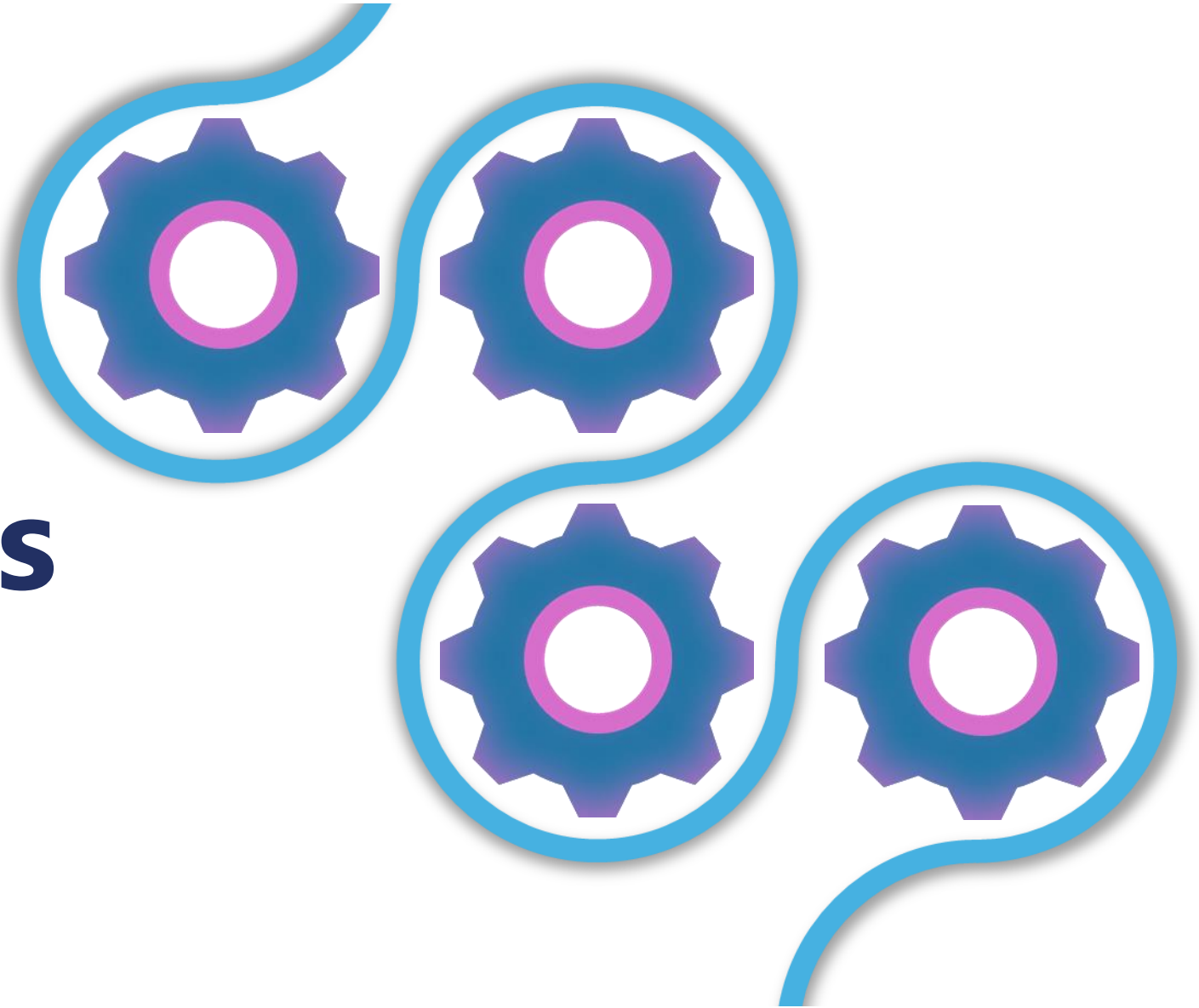
- Duration: 58 seconds
- Finished: 1 week ago
- Queued: 0 seconds
- Timeout: 1h (from project)
- Runner: #216 (DqKy824F) DKRZ-maintained runner accepting arbitrary docker images
- Test summary: 34
- Tags: `dkrz`, `docker-any-image`

Job artifacts: These artifacts are the latest. They will not be deleted (even if expired) until newer artifacts are available.

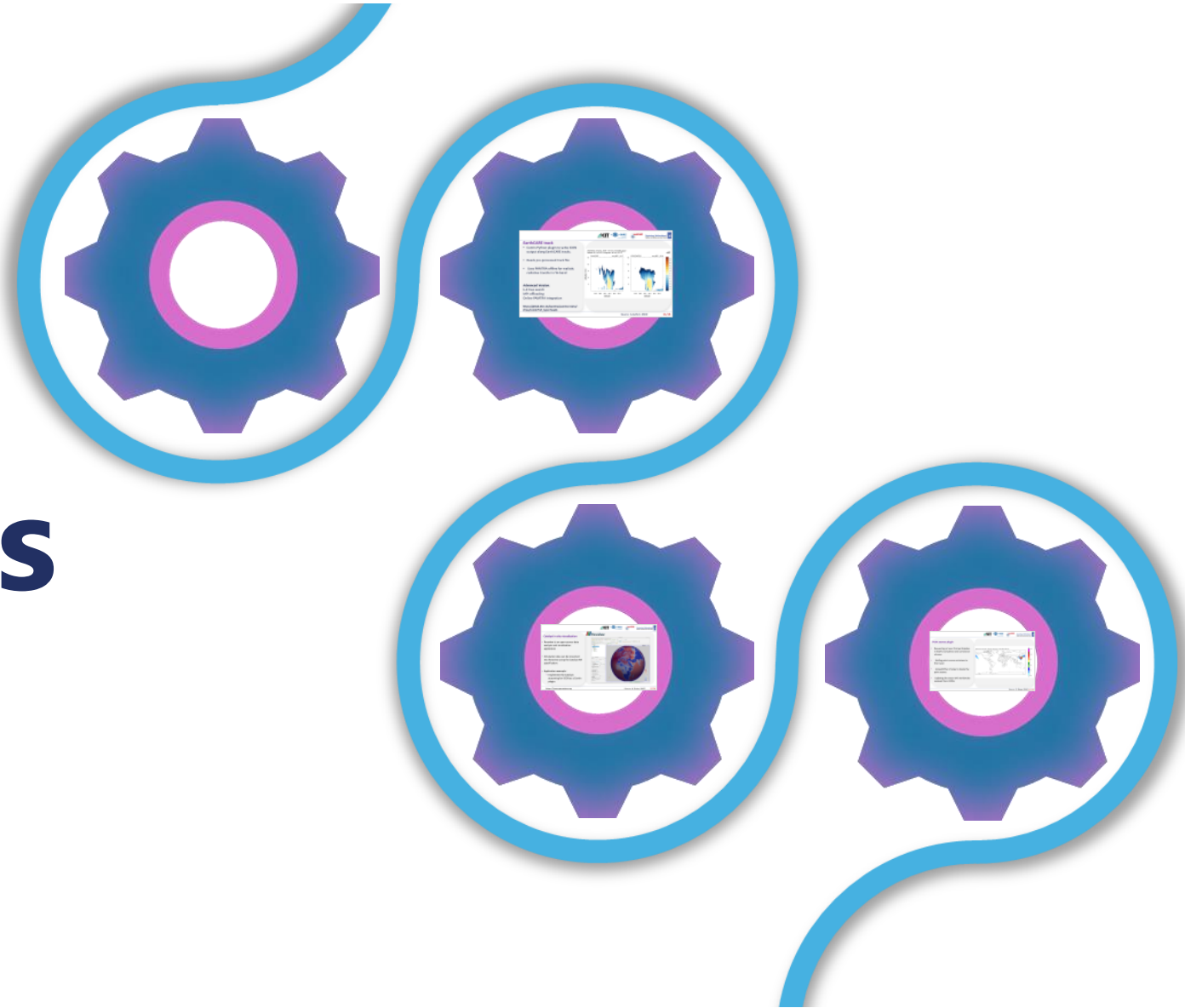
Commit: 48361efa

doc: plugin script path in sys.path

Applications



Applications



Point source plugin

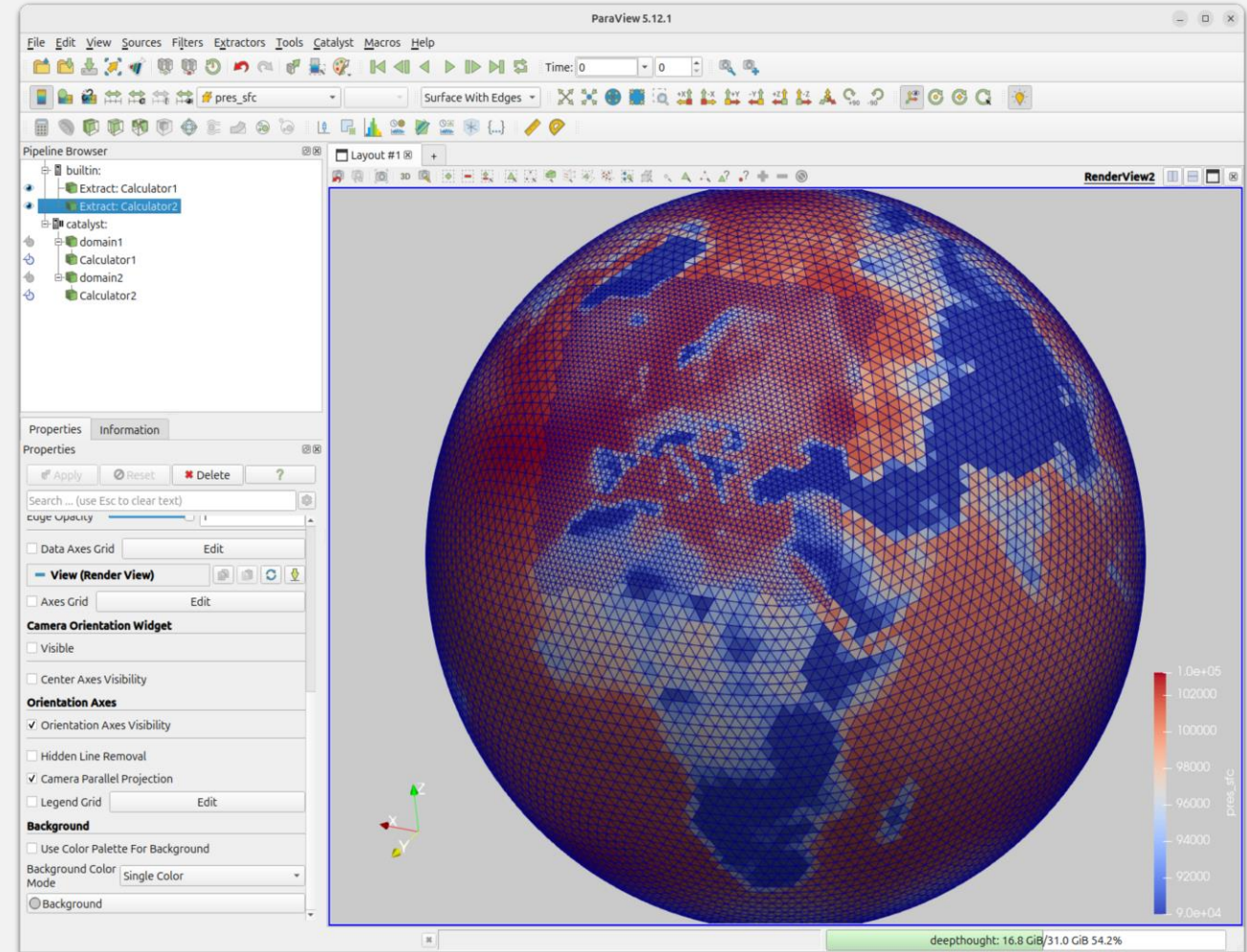
- Requesting a tracer that participates in ICON's turbulence and convection scheme
- Adding point source emissions to this tracer
- Using *KDTree* of *scipy* to locate the point source
- Updating the tracer with tendencies received from ICON's

Point Source 141.0E, 37.4N, lev=10 (17km) - 2014-06-03 00 UTC



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.



EarthCARE track

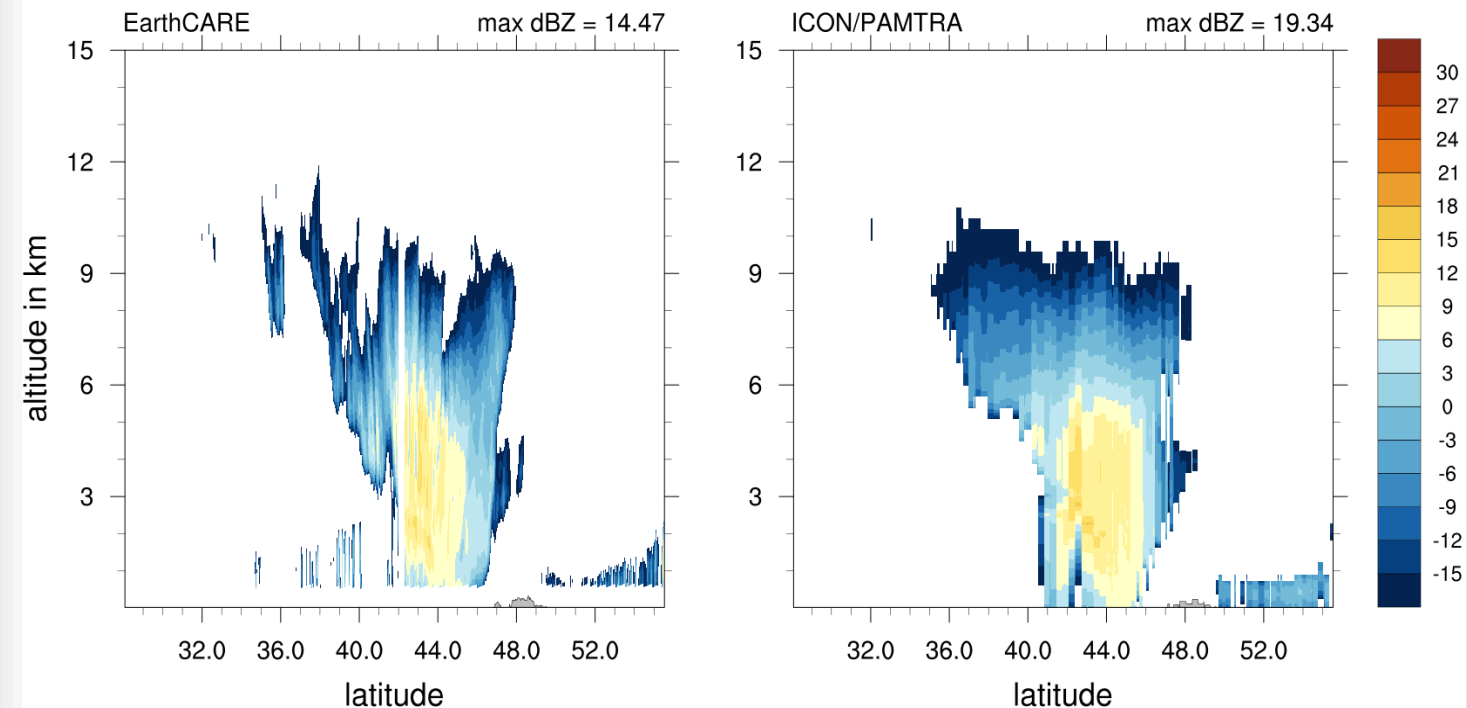
- ComIn Python plugin to write ICON output along EarthCARE tracks.
- Reads pre-processed track file
- Uses PAMTRA offline for realistic radiative transfer in W-band

Advanced Version:

k-d tree search
MPI offloading
Online PAMTRA integration

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

2025/02/01, 00 UTC, 18:08 - 18:15 h of icon058_gscp3
latitude: 28.1 to 55.5 N, longitude: -60.4 to -52.1 E



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

