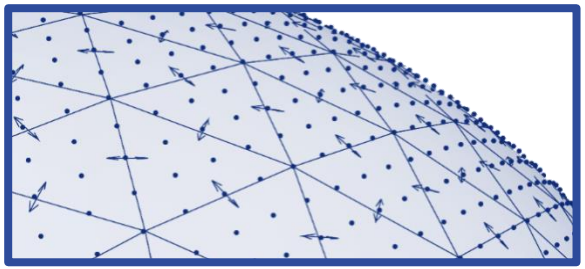
Deutscher Wetterdienst
Wetter und Klima aus einer Hand

ICON Community Interfaces

F. Prill & ICON ComIn project team



DWD, DLR, DKRZ | natESM training | 15 Nov 2023

Who uses and extends the ICON software?

ICON ecosystem, adaptation of https://dtcenter.org/community-code/common-community-physics-package-ccpp

Vision – instead of an outline

Low bar to entry for 3rd party contributions (‘plugins’).

- simplify “out of the box” exploration
  and long range research
- I/O mitigation
  - put post-processing into the model
  - tapping into new data sources
- enable the use of Python with its growing community
  in the atmospheric sciences

We invite you to participate with your own ideas!

Current status – tools for extending ICON

ICON code modules

Adapter code + external source code
(often **Git submodules**)

example: deep atmosphere DyCore

example: ICON-ART

Coupling software YAC [1]

exchange any 2d field between
source and target grids on the sphere

⇢ language interoperability: C, Python

example: ICON ocean waves model

These approaches do not solve the “orchestration problem”: how to organize the data exchange and simulation events. This is the purpose of ICON ComIn.

ICON ComIn is an interface, not a translation layer.

State of the art in ESM

Frameworks connecting 3rd party modules to a host model, using the dynamic loader of the OS:

- CCPP-Framework (common-community-physics-package) for NOAA models
  - developed since 2018
  - https://dtcenter.org/community-code/common-community-physics-package-ccpp

- Plume (plugin mechanism) for ECMWF models
  - since ~11/2022, still under heavy development, v0.2.0
  - https://doi.org/10.5194/egusphere-egu23-7944

ICON Community Interface project

Deutscher Wetterdienst
Wetter und Klima aus einer Hand

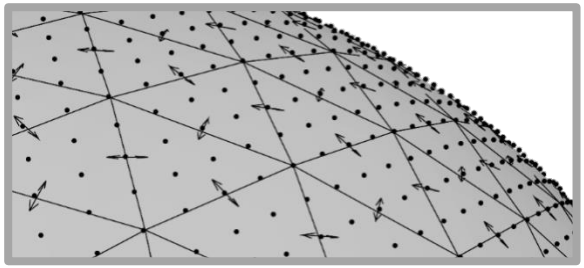
Provide a standardized public interface for optional third party code in ICON (‘plugins’)

- plugins easier to migrate to new ICON releases
- reduce maintenance for ICON as well as for third party code developers
- aligns with DestinE’s “full integration mode” and the modularization in the “Warmworld Faster” project.

Project team: DWD, DKRZ, DLR-IPA, …

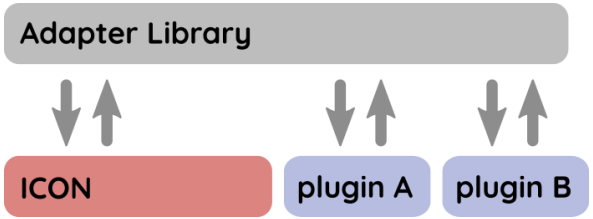
- Originally agreed as a cooperation between DWD and the DLR Institute of Atmospheric Physics in Nov 2021
- DKRZ, KIT have joined in the beginning of 2022
- First release: end of 2023

ComIn API



Adapter library and callback registry

- ComIn Callback Register: Subroutines of the plugins are called at pre-defined events during the ICON simulation. Plugins are attached to “entry points” without modification of the ICON model code.
- The ComIn Adapter Library is included by ICON and the plugins. It contains descriptive data structures and regulates the access and the creation of model variables.



Restrictions and design decisions (at least for v1.0)

- interfaces restricted to cell-based double precision vars
- references are preferred over values (copies)
- no asynchronicity between ICON and 3rd party modules
- plugin calls above the “block loop level”

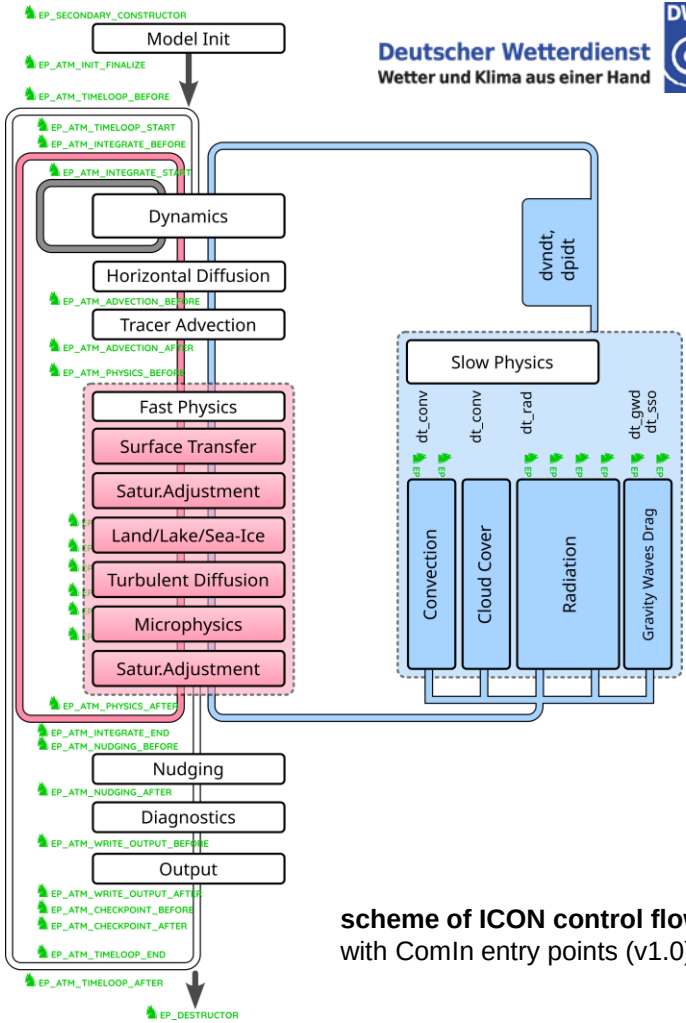
Plugins automatically compare the version info of “their” adapter libraries to ICON’s adapter library version. Minor versions: backward compatible.

Plugging into ICON's control flow

ComIn connects 3rd party modules to ICON, using the dynamic loader of the OS.

- configure option: `--enable-comin`
  switch off ComIn entirely by preprocessor
  `#ifndef __NO_ICON_COMIN__`
- Two entry points before the time loop:
  primary constructor – mandatory
  secondary constructor – optional, after allocation of ICON variable list
- Entry points: "before/after" syntax

  `EP_<COMP>_<PROCESS|LOOP>_[BEFORE|AFTER|START|END]`



scheme of ICON control flow
with ComIn entry points (v1.0)

Fortran namelist comin_nml

Deutscher Wetterdienst
Wetter und Klima aus einer Hand

Enable plugin in the host model via namelist.

ICON

```
&comin_nml
  plugin_list(1)%name           = "simple_fortran_plugin"
  plugin_list(1)%plugin_library = "libsimple_fortran_plugin.so"
/
```

multiple plugins can be attached

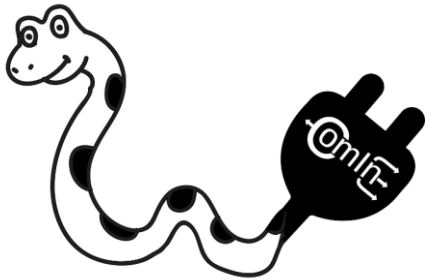
Optional: non-standard name of constructor, MPI communicator, user-specified plugin arguments (character string)

Writing a ComIn-compliant module

Plugins are built independently from ICON.

- ComIn contains an emulator of an NWP model for stand-alone development
- multi-language interface: Fortran, C/C++, Python

… see application example on later slides.



CMakeLists.txt – for plugin build

```
find_package(ComIn)
target_link_libraries(myproject comin)
```

… and that's it.

Python plugins do not need any compilation process at all.

Primary constructor example


```
TYPE(t_comin_descrdata_domain), POINTER :: p_patch
TYPE(t_comin_setup_version_info)        :: version

SUBROUTINE comin_main()  BIND(C)
  version = comin_setup_version_info()
  IF (version%version_no_major > 1) &
    CALL comin_finish("comin_main (simple_fortran_plugin)", "incompatible version!")

  CALL comin_request_add_var(                                              &
    &    t_comin_var_descriptor( id = 1, name = "simple_fortran_var" ),    &
    &    t_comin_var_metadata  (…), &
    &  ierr )

  CALL comin_callback_register(EP_SECONDARY_CONSTRUCTOR,   simple_fortran_seccstr, ierr)
  CALL comin_callback_register(EP_ATM_WRITE_OUTPUT_BEFORE, simple_fortran_diagfct, ierr)

  p_patch => comin_descrdata_get_domain(1, ierr)
END SUBROUTINE comin_main
```

Primary constructor example

Deutscher Wetterdienst
Wetter und Klima aus einer Hand

```
TYPE(t_comin_descrdata_domain), POINTER :: p_patch
TYPE(t_comin_setup_version_info)        :: version

SUBROUTINE comin_main() BIND(C)
  version = comin_setup_version_info()
  IF (version%version_no_major > 1) &
    CALL comin_finish("comin_main (simple_fortran_plugin)", "incompatible version!")

  CALL comin_request_add_var(                                              &
    &     t_comin_var_descriptor( id = 1, name = "simple_fortran_var" ),   &
    &     t_comin_var_metadata   (…), &
    &   ierr )

  CALL comin_callback_register(EP_SECONDARY_CONSTRUCTOR,   simple_fortran_seccstr, ierr)
  CALL comin_callback_register(EP_ATM_WRITE_OUTPUT_BEFORE, simple_fortran_diagfct, ierr)

  p_patch => comin_descrdata_get_domain(1, ierr)
END SUBROUTINE comin_main
```

primary constructor

request additional ICON variable

register function callbacks

get descriptive data structures

ComIn API

Nomenclature for procedures, variables and derived data types

`comin_<scope> / t_comin_<scope>`

scopes: “setup”, “callback”, “primaryconstructor”, “parallel”, “descrdata”, “var”, “request”, …

Examples:

- `comin_callback_register, comin_callback_get_ep_name`
- `comin_var_get`
- `comin_parallel_get_plugin_mpi_comm`

Descriptive data structures

- invariant global data, eg. grid topology
- simulation status information (current time step, domain, …)
- MPI parallelization info

Parallel execution

Data access limited to the local MPI process, but plugins may handle their own parallel communication.

- ComIn initialization harmonized with YAC
- reference implementation for MPI handshake in YAC and ComIn available in C, Python and Fortran (⇢ thanks to M. Hanke, DKRZ)
- Plugin-to-plugin communication: ComIn allows plugins to be attached process-wise, using the same plugin communicator
- next steps: extend API for GPU host-to-device transfer

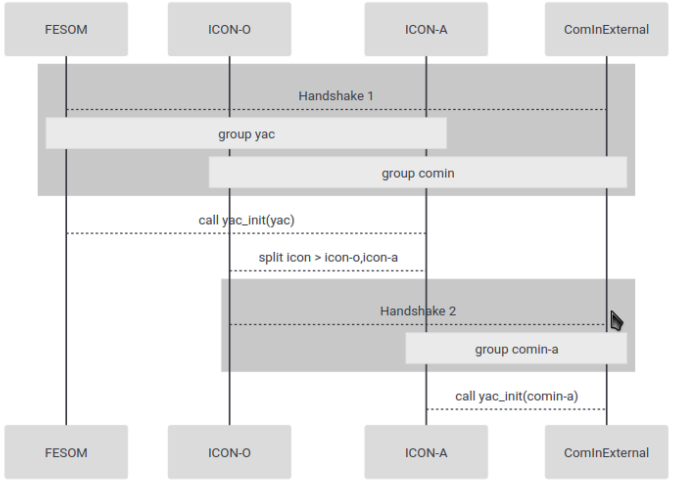
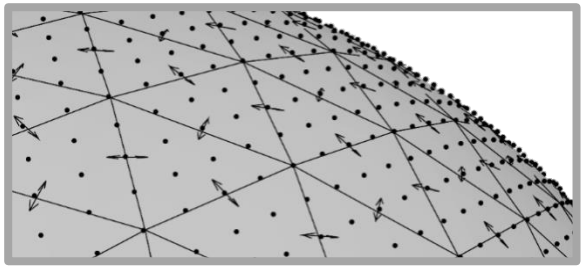


Fig.: Scheme of ComIn MPI handshake

Applications



Python example

Deutscher Wetterdienst
Wetter und Klima aus einer Hand

```
import comin
import numpy as np

print("Hello World!", file=sys.stderr)
tke_metadata = comin.Metadata()
comin.request_add_var("tke", 1, tke_metadata)

@comin.register_callback( ↵
  comin.EP_SECONDARY_CONSTRUCTOR)
def simple_python_constructor():
    global pres, tke
    print("PYTHON: secondary constructor")
    pres = comin.var_get( ↵
      [comin.EP_ATM_WRITE_OUTPUT_BEFORE],
      "pres", id=1)
    tke = comin.var_get(↵
      [comin.EP_ATM_WRITE_OUTPUT_BEFORE],
      "tke", id=1)
```

```
@comin.register_callback( ↵
  comin.EP_ATM_WRITE_OUTPUT_BEFORE)
def simple_python_diagfct():
    print("PYHTON: diagfct called!")
    print(np.asarray(pres))
    np.asarray(tke)[:] = 42.

@comin.register_callback(comin.EP_DESTRUCTOR)
def simple_python_destructor():
    print("Good bye!", file=sys.stderr)
```

Python ~ Lingua franca of atmospheric science and data science students.

Python example


```
import comin
import numpy as np

print("Hello World!", file=sys.stderr)
tke_metadata = comin.Metadata()
comin.request_add_var("tke", 1, tke_metadata)

@comin.register_callback( ↵
  comin.EP_SECONDARY_CONSTRUCTOR)
def simple_python_constructor():
    global pres, tke
    print("PYTHON: secondary constructor")
    pres = comin.var_get( ↵
      [comin.EP_ATM_WRITE_OUTPUT_BEFORE],
      "pres", id=1)
    tke = comin.var_get(↵
      [comin.EP_ATM_WRITE_OUTPUT_BEFORE],
      "tke", id=1)
```

primary constructor

request additional ICON variable

```
@comin.register_callback( ↵
  comin.EP_ATM_WRITE_OUTPUT_BEFORE)
def simple_python_diagfct():
    print("PYHTON: diagfct called!")
    print(np.asarray(pres))
    np.asarray(tke)[:] = 42.

@comin.register_callback(comin.EP_DESTRUCTOR)
def simple_python_destructor():
    print("Good bye!", file=sys.stderr)
```

register function callbacks through Python decorator

Python ~ Lingua franca of atmospheric science and data science students.

Application: Point source plugin

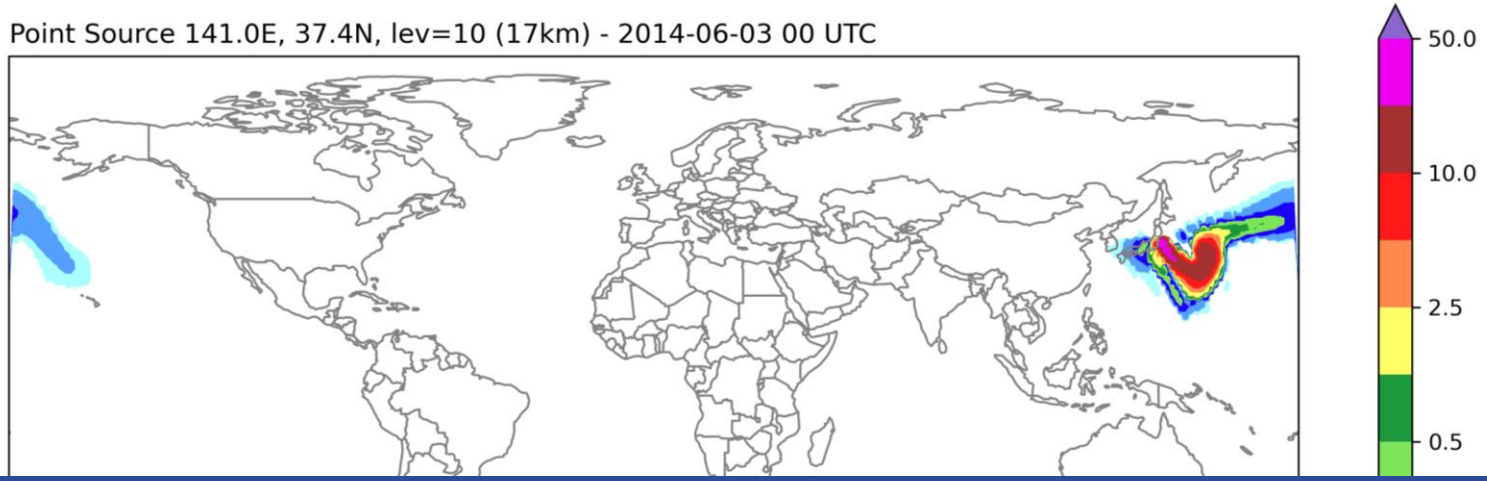


Fig.: D. Rieger, DWD

(conv/turbulent transport disabled)

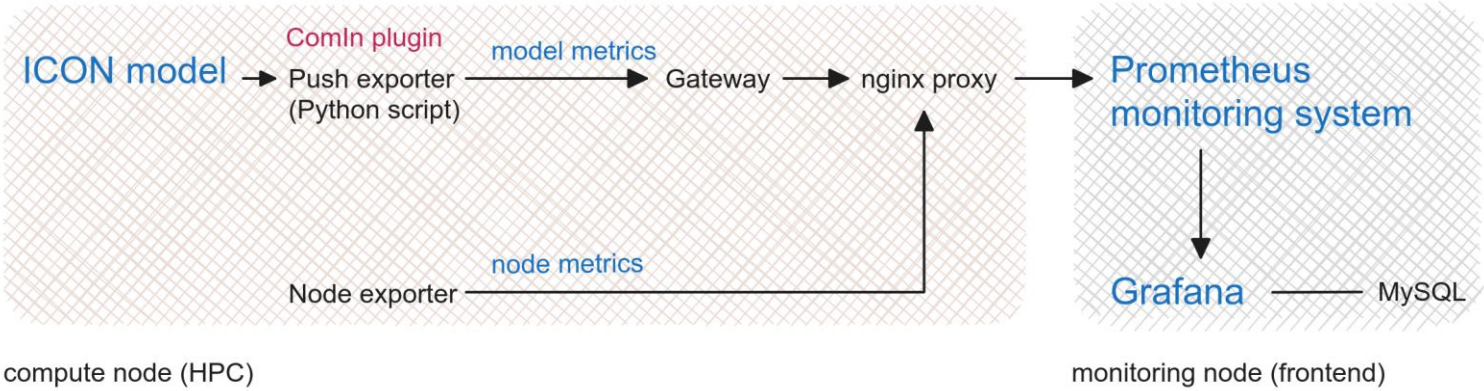
- request a tracer that participates in ICON's turbulence scheme
- add point source emissions to this tracer
- update the tracer with tendencies received from ICON's turbulence scheme
- exploit Python’s incredibly vast library of useful modules

Application: Telemetry

Prometheus + Grafana: popular monitoring system and time series DB

- “exporters” supply frontend with data (via HTTPS)
- e.g. Prometheus `node_exporter`:
  exporter for machine metrics

Proof-of-concept: connect Grafana to ICON with a ComIn exporter plugin

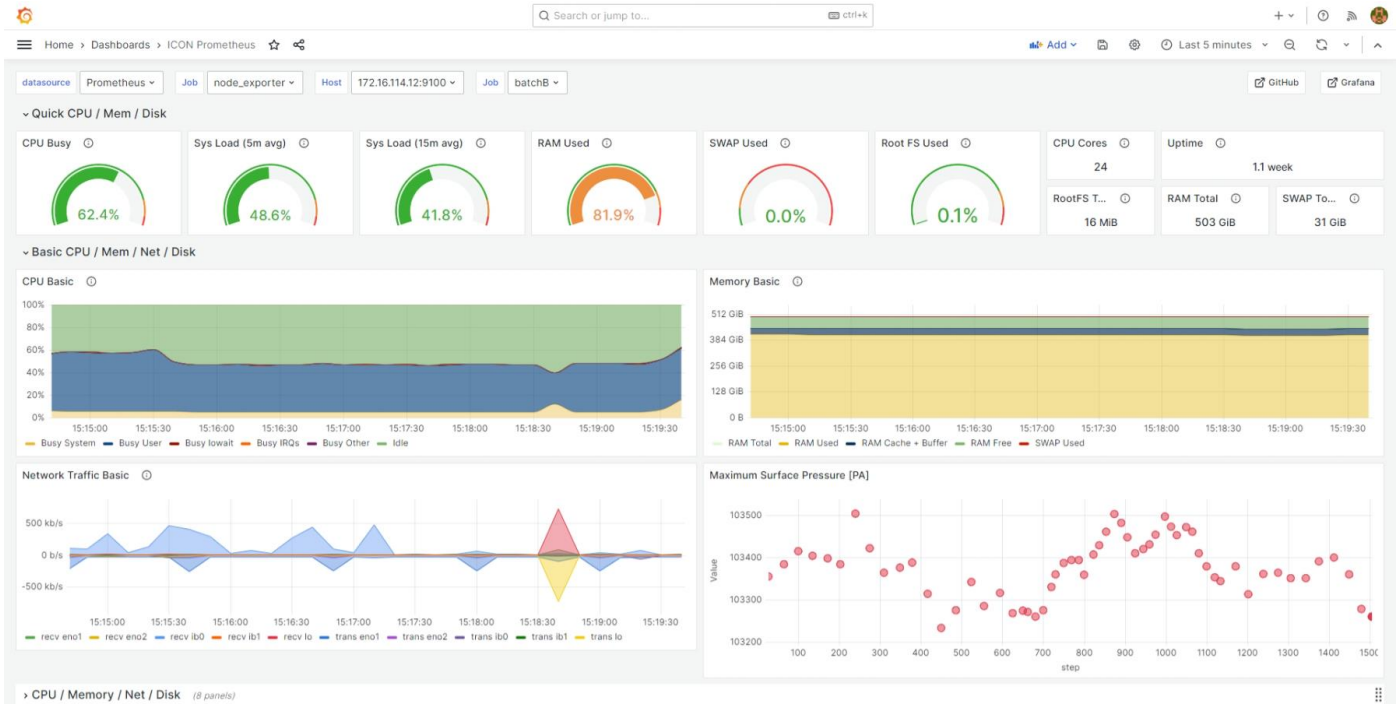


Source:
C. Eser (DWD-TI), FP

Prometheus exporter plugin

Deutscher Wetterdienst
Wetter und Klima aus einer Hand

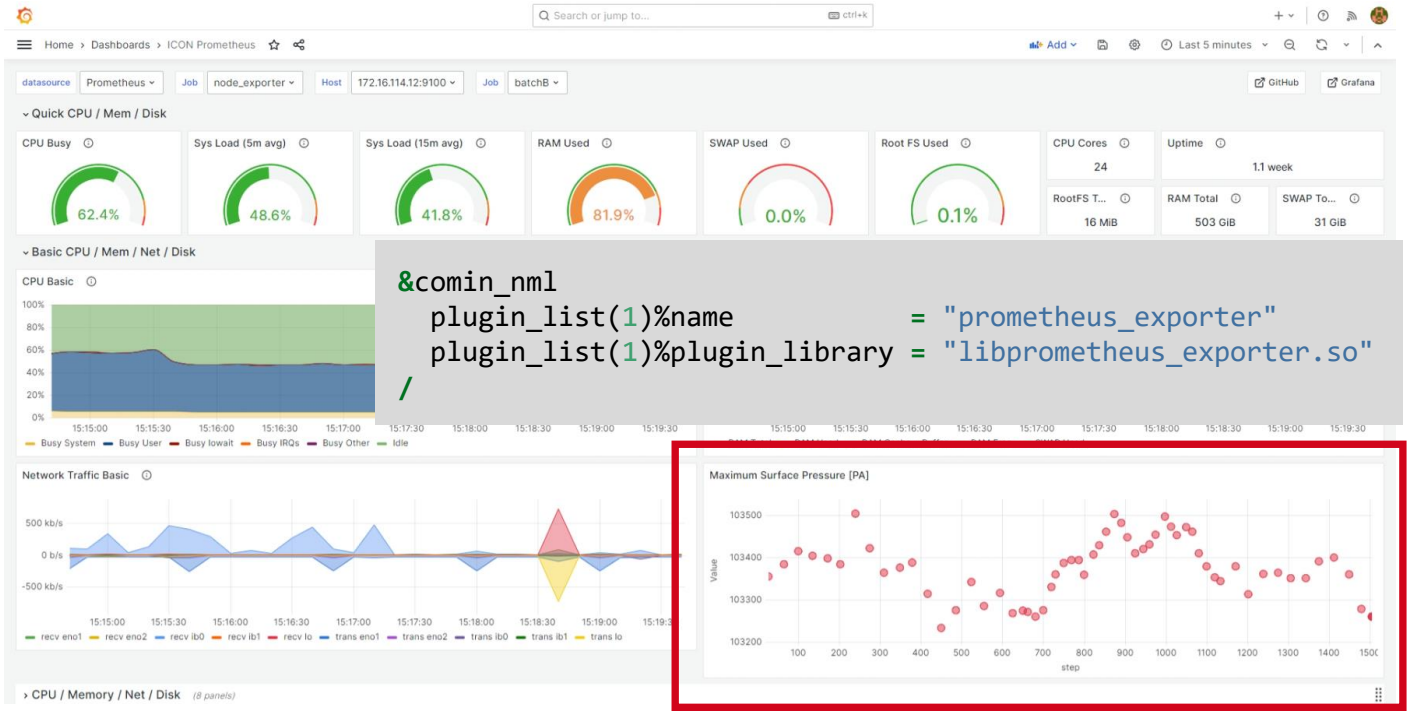
Flexible monitoring dashboard offered by Grafana:



Prometheus exporter plugin

Deutscher Wetterdienst
Wetter und Klima aus einer Hand

Flexible monitoring dashboard offered by Grafana:

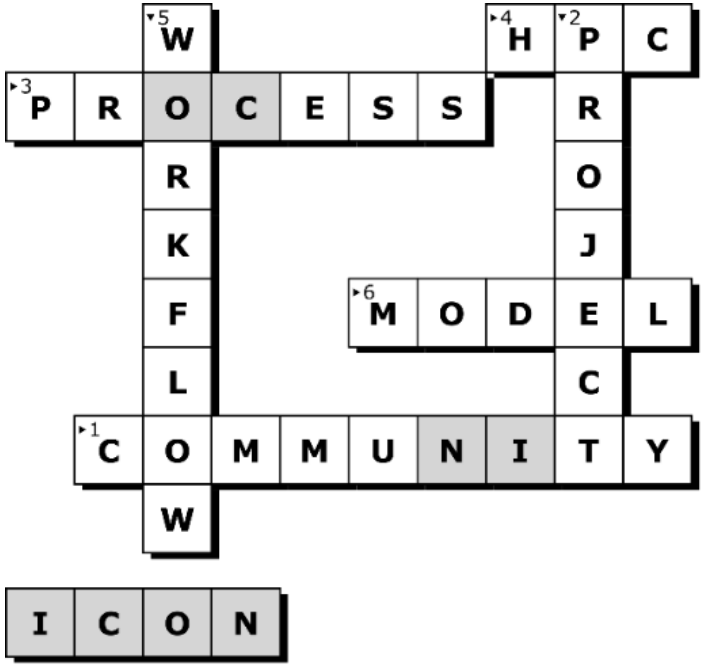



```
&comin_nml
  plugin_list(1)%name          = "prometheus_exporter"
  plugin_list(1)%plugin_library = "libprometheus_exporter.so"
/
```

Ex.: DWD NEC platform displaying ICON's maximum surface pressure

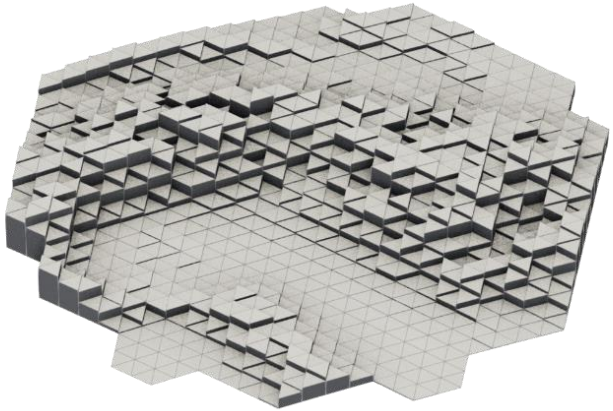
Further applications

- implementation of physics innovations, e.g. land, chemistry, hydrology
- data sources, e.g. horizontal or vertical interpolation of data
- light-weight data extraction
- in-situ analysis
- (helps to) express workflows as Python programs
- continuous online training for machine learning algorithms



TL;DR

- ICON becomes Open Source by beginning of 2024.
- Perfect time for a well-defined plug-in mechanism that makes ICON flexibly expandable.



Next steps

- First ComIn release version targeted for end of 2023.
- Planning for future releases already started.
- Restructure code modules already existing in ICON.
- natESM sprint for MESSy is a good example, replace current ICON 2.6.5 code instrumentation.

Lessons learned

- Cross-institutional projects: scientists have to be involved with the software engineering.

ComIn Gitlab area
https://gitlab.dkrz.de/icon-comin (restricted to ICON developers)

ICON feature branch
https://gitlab.dkrz.de/icon/icon-nwp/-/merge_requests/1038

Poster and presentation (03/2023)
https://go.dwd-nextcloud.de/index.php/s/kixRQ8gNF3tBgfH

Florian Prill

Met. Analyse und Modellierung
Deutscher Wetterdienst

e-mail: Florian.Prill@dwd.de