The Modular Earth Submodel System (MESSy)

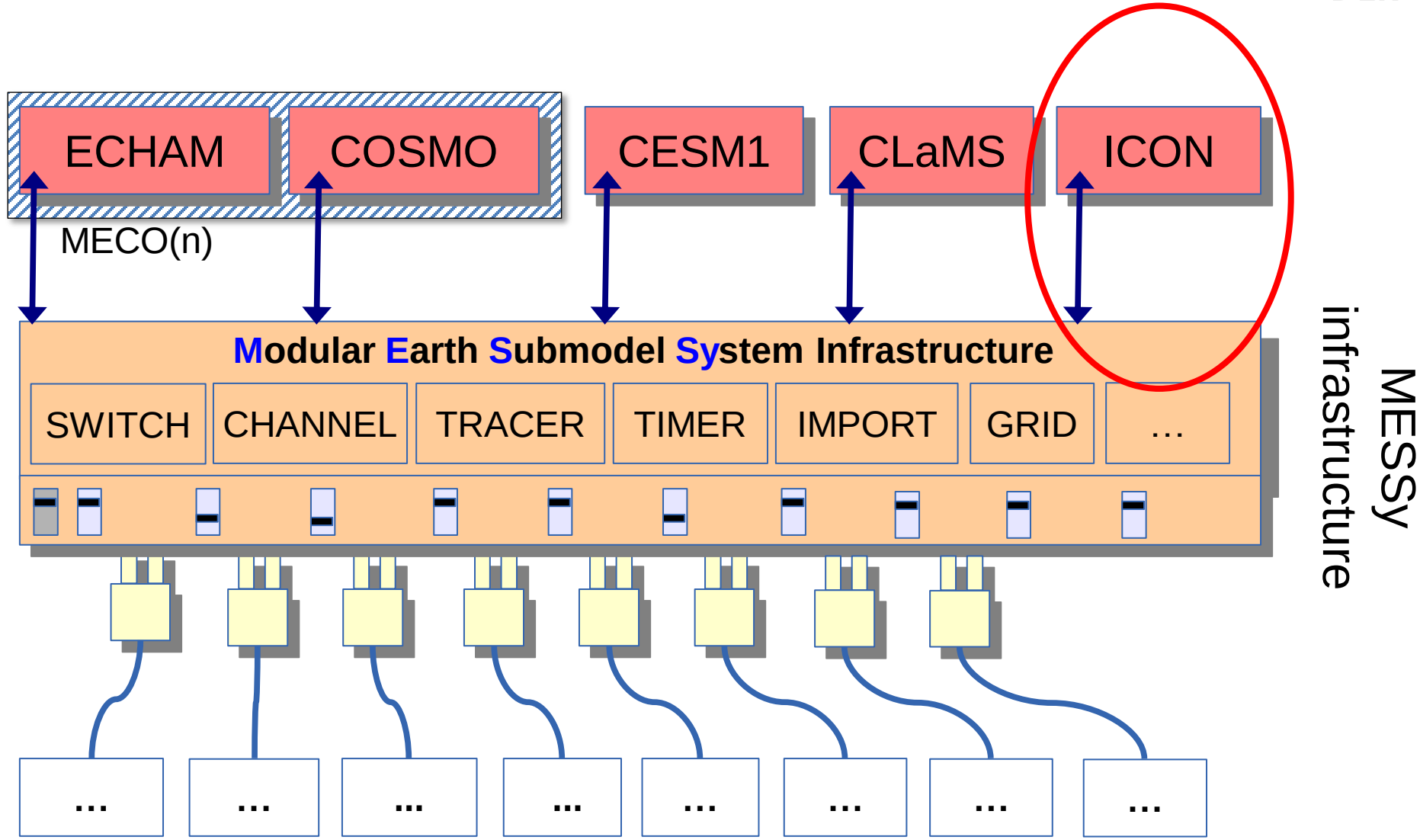
Long Lasting Interface Concept on Its Way into the Future

MESSy - Modular Earth Submodel System

https://www.messy-interface.org/

> 20 partner institutes

Framework to couple scientific codes to numerical weather prediction and climate models



P. Jöckel

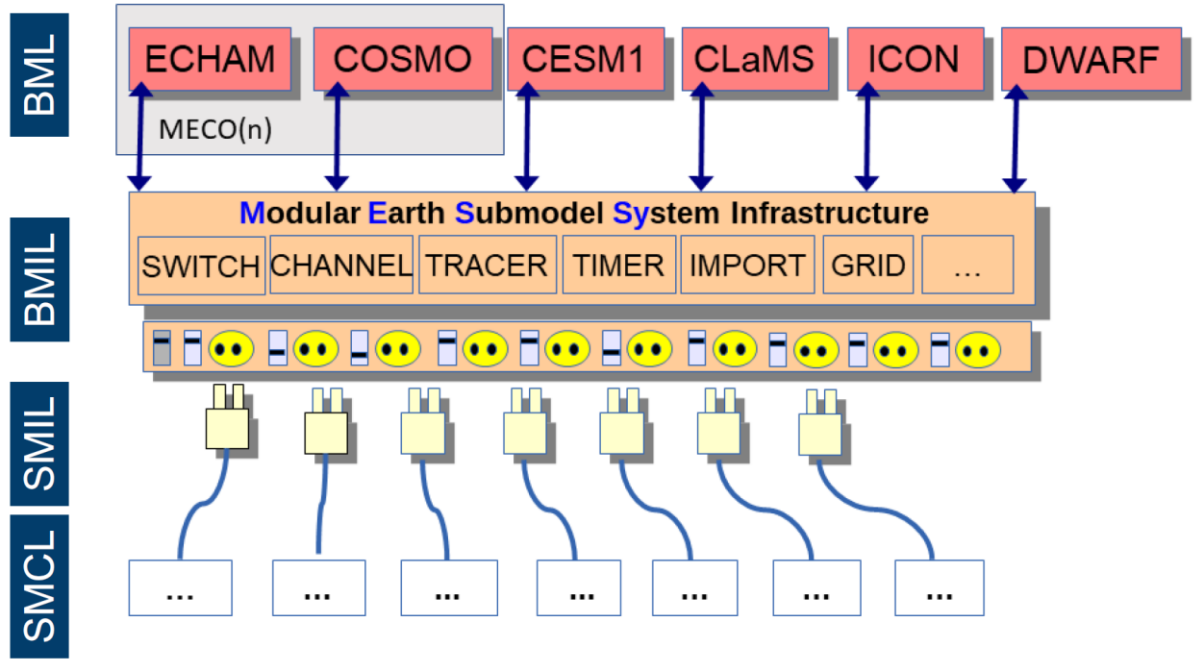
MESSy – a short introduction

Modular Earth Submodel System

- modular design supports range of applications, for example range of scales and operational modes
- strict separation of process description from model infrastructure
- software engineered code*
- code hosted at *gitlab.dkrz.de*
- wiki in addition to manuals for submodels document the code

*code organized in 4-layer structure, object oriented approach (as far as possible), “operator splitting” concept

MESSy code structure

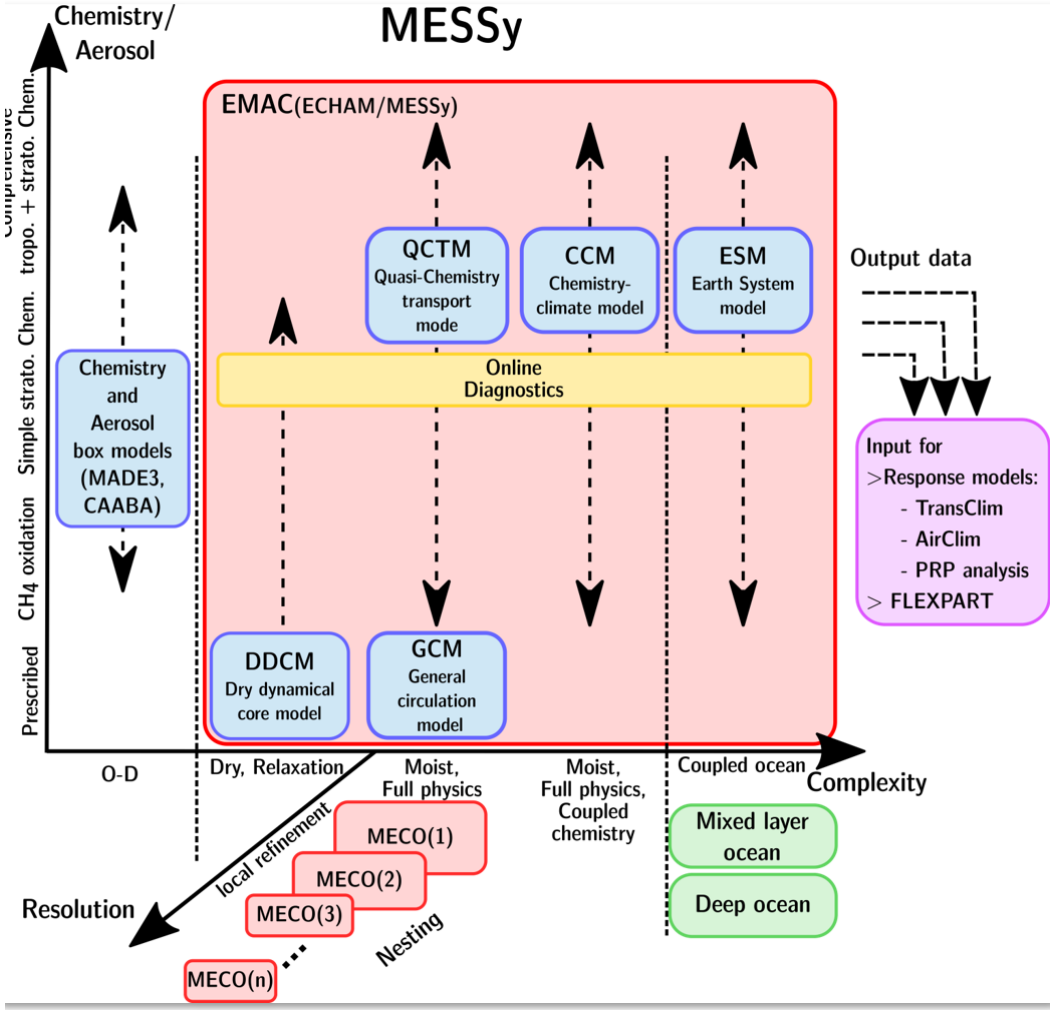


MESSy: range of applications

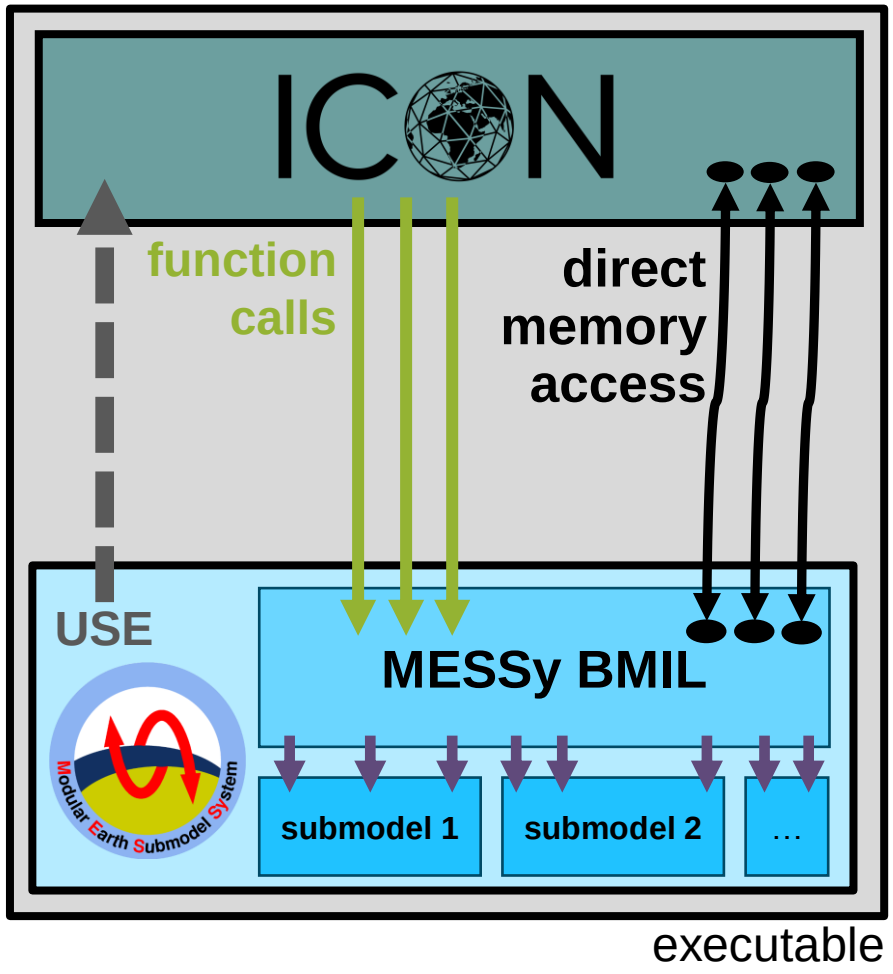
from local air quality modelling and campaign forecasts to climate projections

from idealized setups, through quasi chemistry-transport model mode and/or setups with specified dynamics, to chemistry-climate model setups with coupled ocean

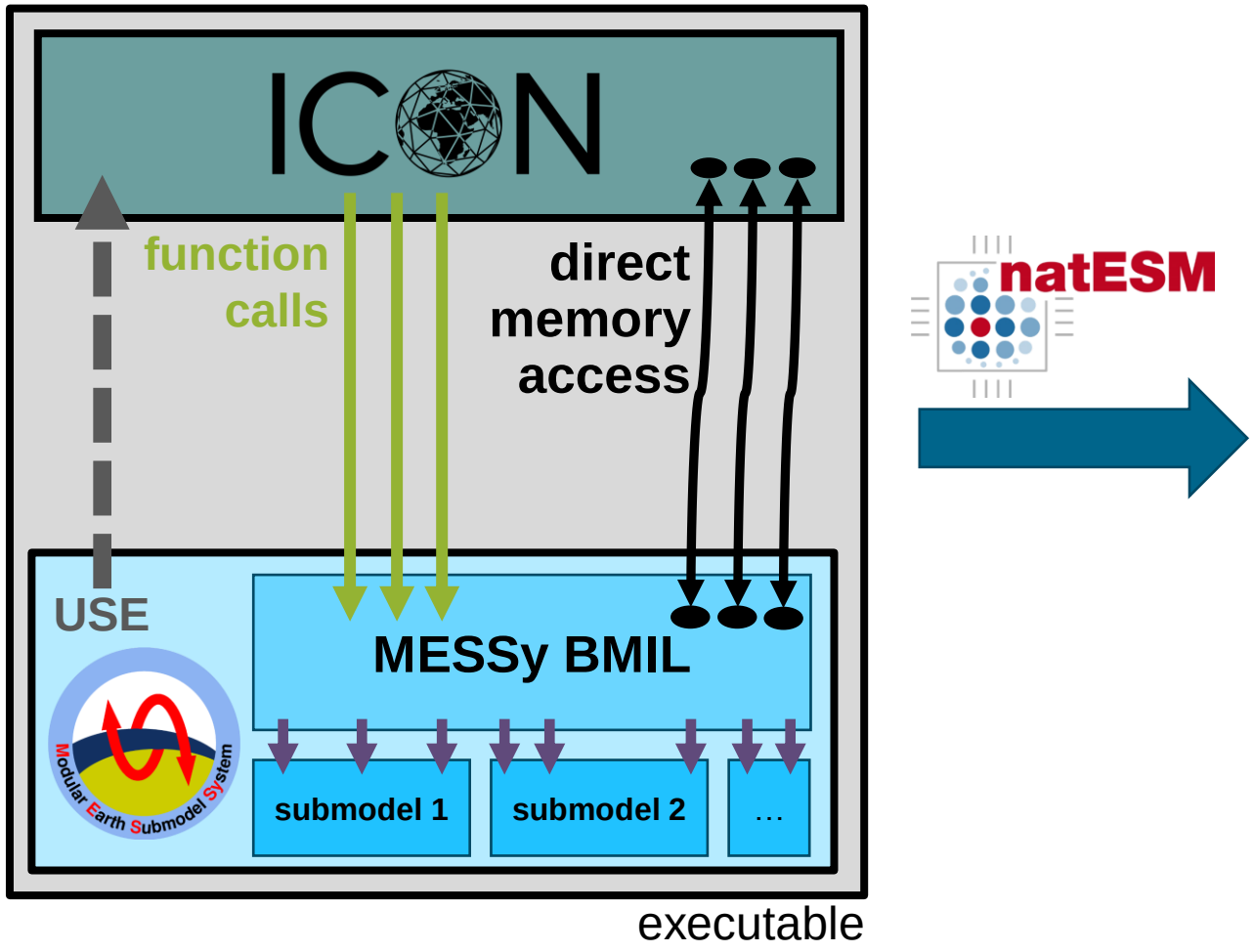
from regional to global modelling



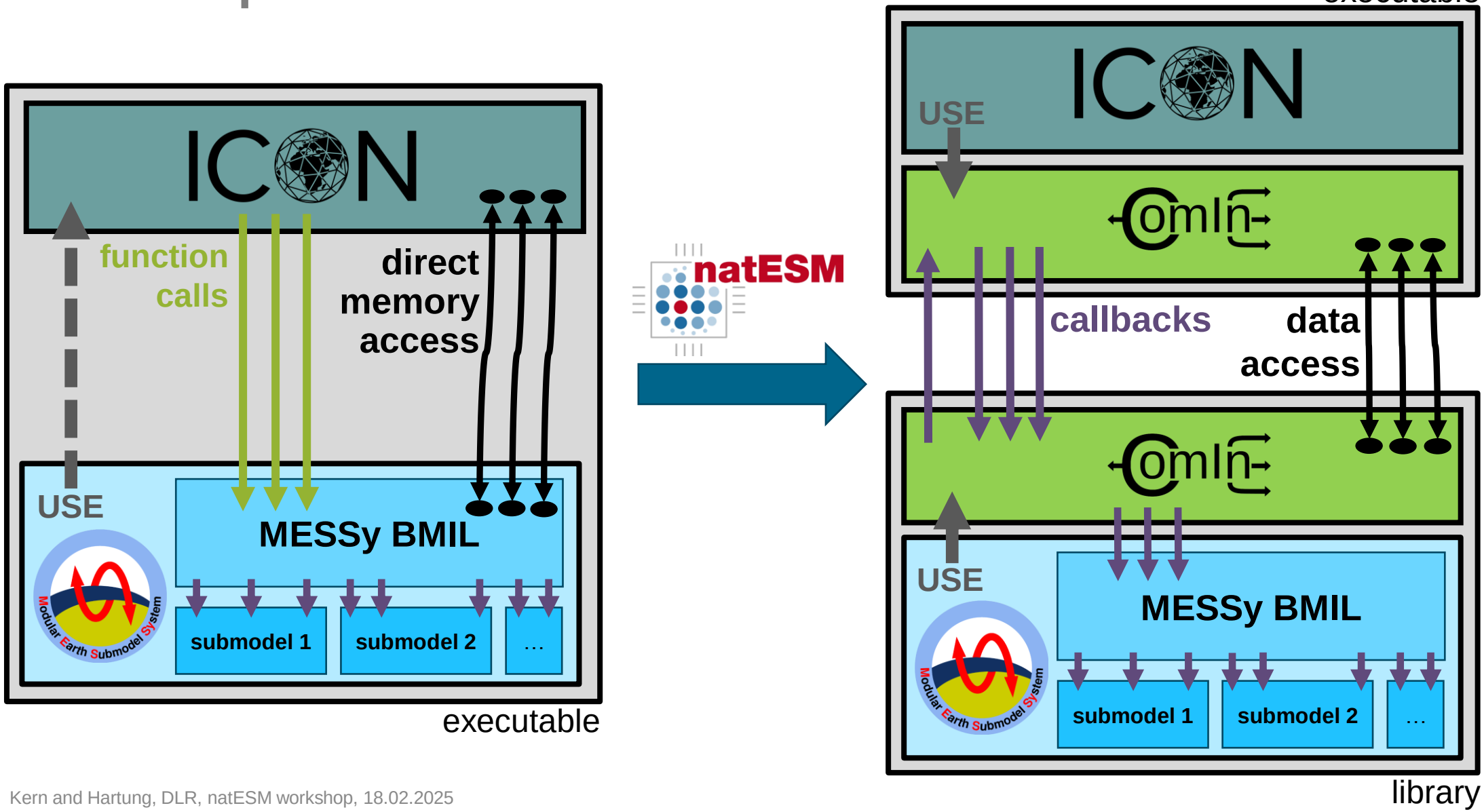
ComIn/MESSy
natESM sprints



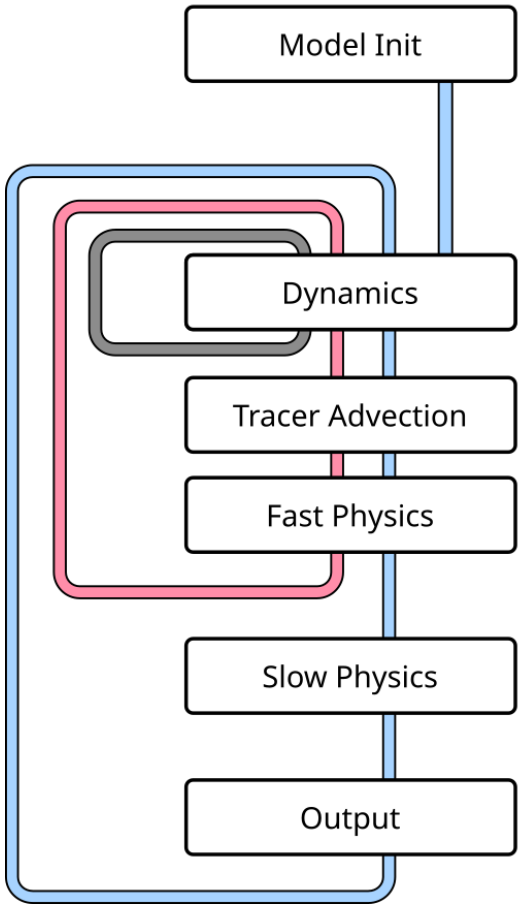
ComIn/MESSy
natESM sprints



ComIn/MESSy natESM sprints



MESSy modifications and workarounds



Infrastructure/general:

- adapt CHANNEL to work with ComIn pointer wrapper
- access to local variables/routines
- MPI implementation
- convective tracer transport

Init:

- allocate tracer memory en bloc (number calculated during runtime)
- register new variables/tracers for restart

Time loop:

- additional process tendencies
- masking of (processes over specific) regions

Florian Prill, ICON tutorial

Recipe to prepare software as ComIn plugin

1. Prepare code as shared library (unless using Python, where everything is easier).
2. Decide which entry points will be accessed and which data need to be added to ICON (mainly tracers). From this point onwards descriptive data can be accessed. This already defines the content of the primary constructor.
3. Decide which ICON data the plugin should access. In case of a larger/ more complex plugin: decide how to associate existing data structures with those provided by ComIn. This is a main component of the secondary constructor.
4. Associate routines registered to entry points with plugin routines.
5. Update ICON runsript (*comin_nml* section) to apply plugin (prepared as shared library).

Open questions, requirements from Plugin-side

Mainly from a subjective MESSy POV

- Should I develop a MESSy submodel or a ComIn plugin?
  - Additional (infrastructure) components provided from MESSy
  - ComIn interfaces for various programming languages
- Enhanced restart requirements?
  - Plan: introduce entry points, so plugins can call functions in specific cases (checkpointing, restart)
- Local loop variables
  - Do your plugins need variables, not exposed via the ICON (global) variable lists?
  - For MESSy/ComIn: code patches for ICON. If a variable is needed by several ComIn plugins: discussion with ICON developers, if it could be made available
- “Running plugins” in specific regions?
  - “masking of regions” / workaround: overwriting ICON’s variable fields
- More requirements coming to mind?

Impressum

Thema:	**Lorem ipsum consetetur sadipscing** Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat et justo duo dolores
Datum:	2023-01-01 (JJJJ-MM-TT)
Autor:	Vorname Nachname
Institut:	Lorem ipsum dolor sit amet
Bildquellen:	Alle Bilder „DLR (CC BY-NC-ND 3.0)“, sofern nicht anders angegeben