Generic API concepts in ESM – lessons learned from MESSy*

15.11.2023, natESM Training2023, Hamburg

Patrick Jöckel

*Modular Earth Submodel System (http://www.messy-interface.org)

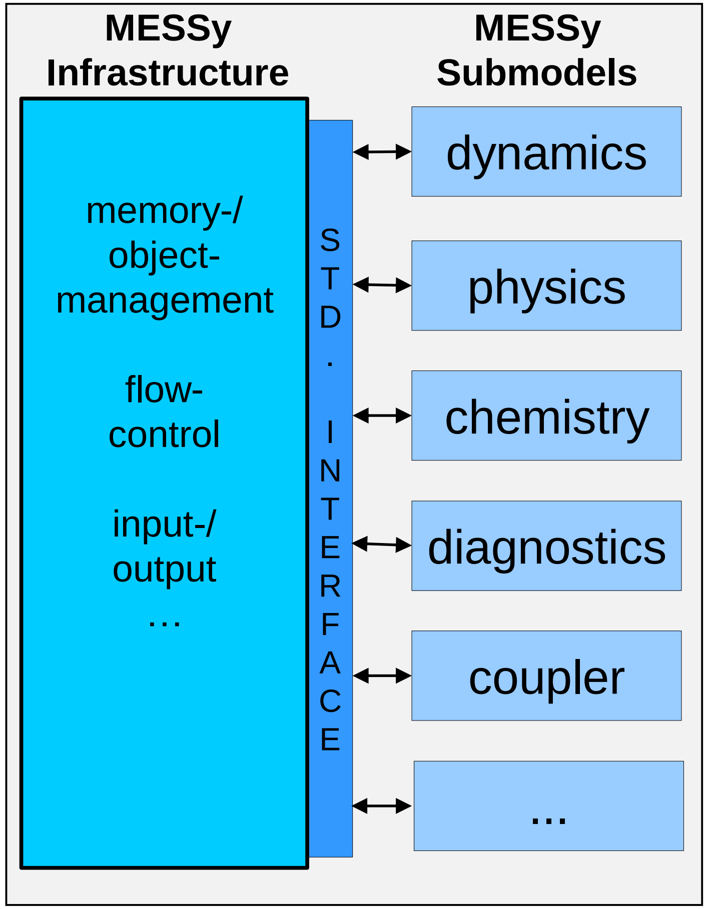
The Modular Earth Submodel System (in a nutshell)

MESSy is a ***software*** providing a ***framework*** for a standardized, ***bottom-up*** implementation of Earth System Models (or parts of those) with flexible complexity.

“Bottom-up” means, the MESSy software provides an ***infrastructure*** with generalized ***interfaces*** for the standardized control and interconnection (=coupling) of “low-level ESM components” (dynamical cores, physical parameterizations, chemistry packages, diagnostics etc.) which are called submodels.

MESSy comprises currently ~ 160 submodels (i.e., coded MESSy conform):

- infrastructure (= the framework) submodels
- diagnostic submodels
- atmospheric chemistry related submodels
- model physics related submodels



Some background information & history

You start with: Legacy (ESM) Codes

- ECHAM
- COSMO
- CESM1
- ICON

The aim is to …

- implement “atmospheric chemistry”
  - climate feedback (large / long scale)
  - air quality (small /short(er) scale)
  - → large flexibility required

Some background information & history

You start with: Legacy (ESM) Codes

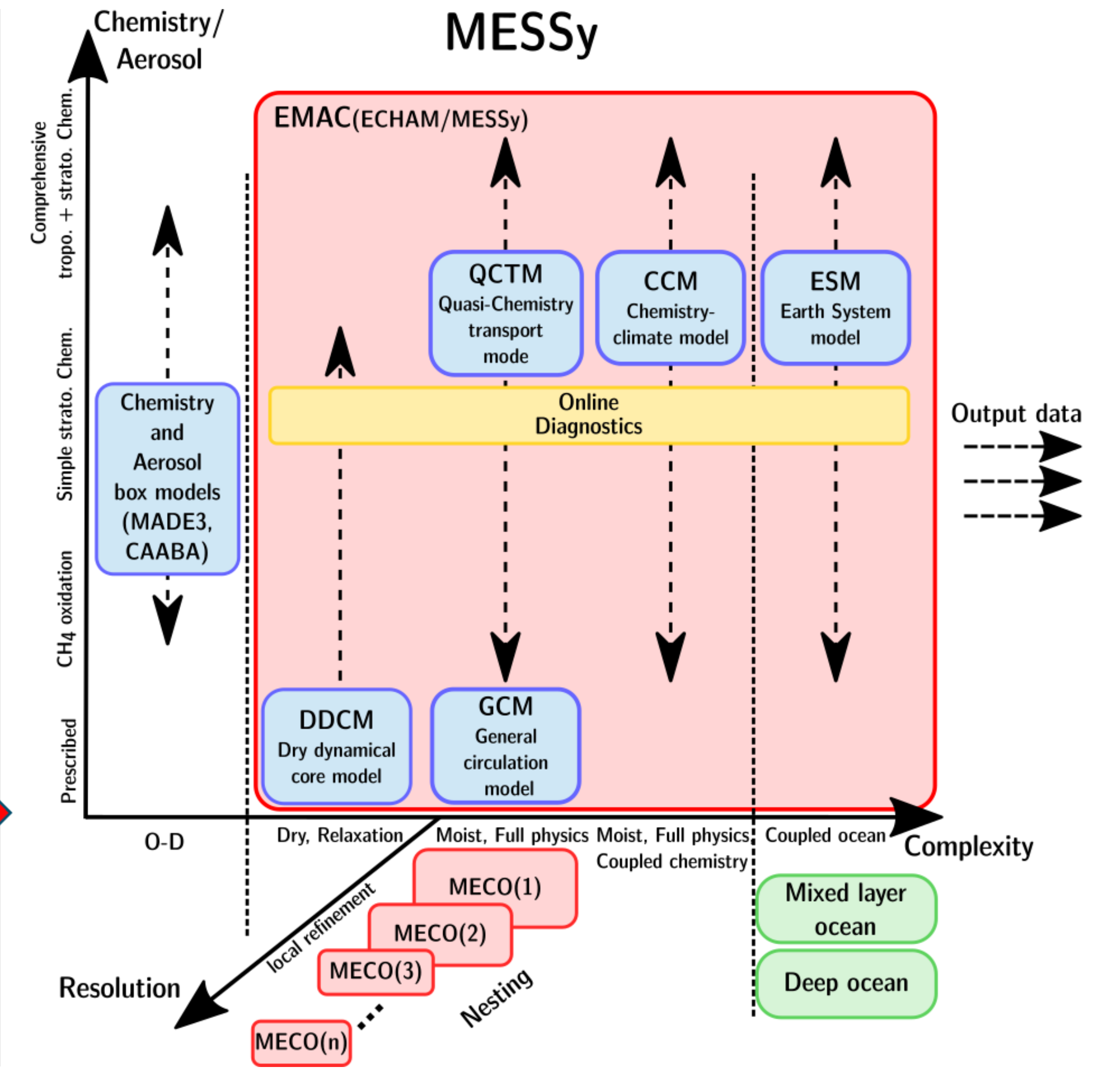
- ECHAM
- COSMO
- CESM1
- ICON

The aim is to …

- implement "atmospheric chemistry"
  - climate feedback (large / long scale)
  - air quality (small /short(er) scale)
  - → large flexibility required

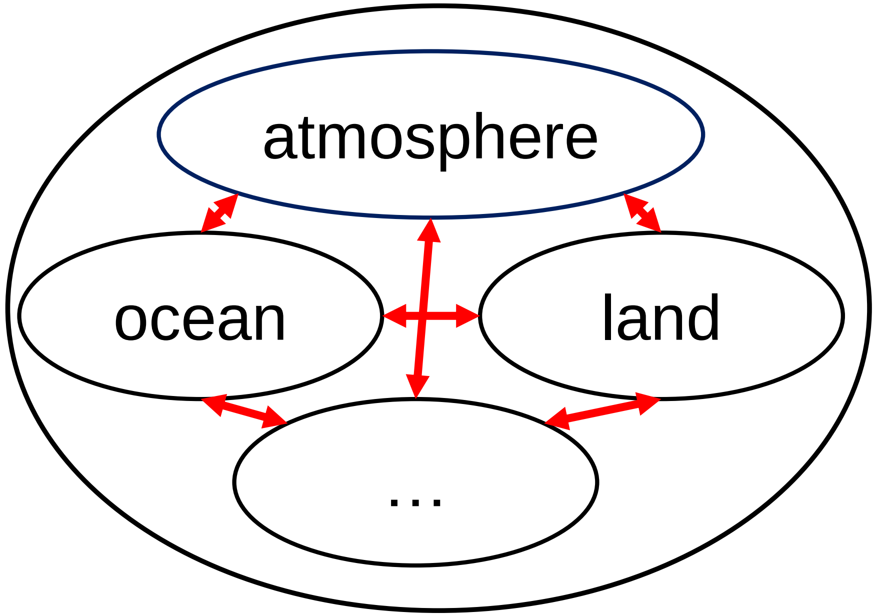
Not only for atmospheric chemistry! …

The result is … an integrated framework …



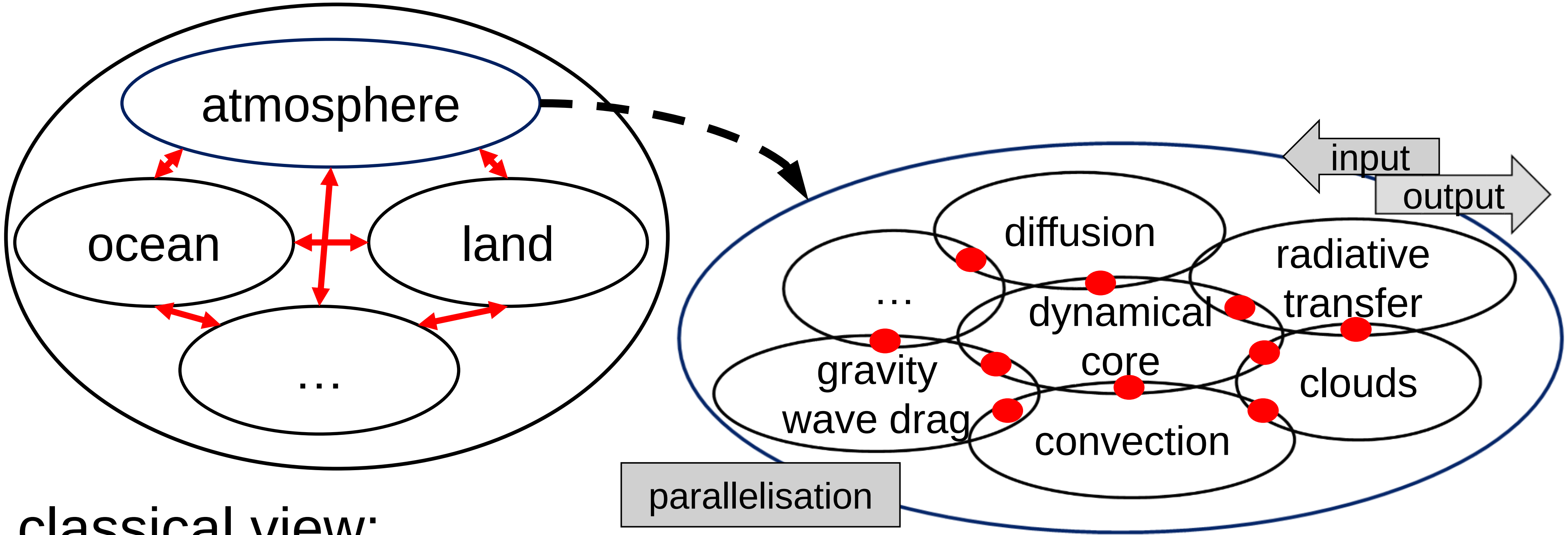
WHICH WAY TO GO?

Earth System model components



classical view:
coupled compartments

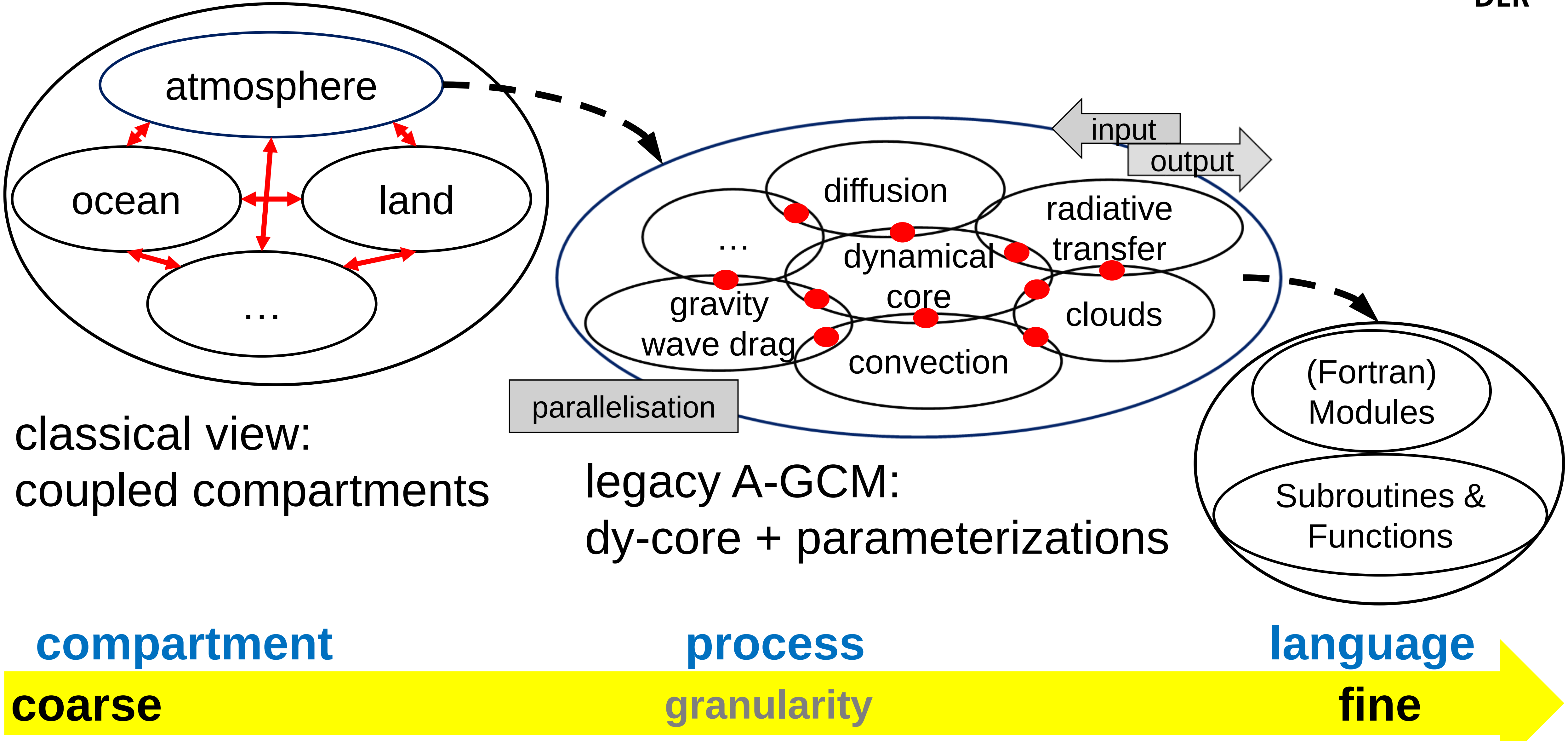
Earth System model components



classical view:
coupled compartments

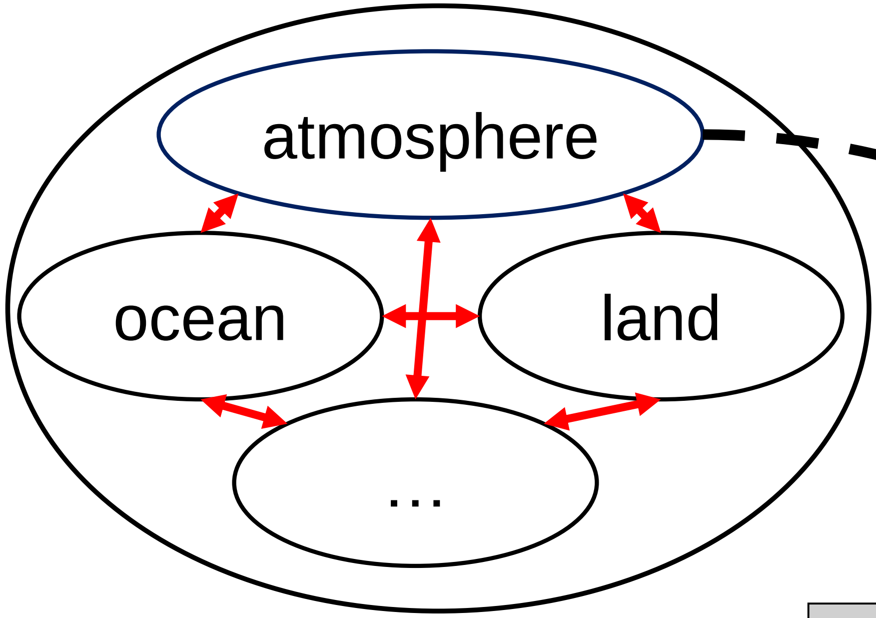
legacy A-GCM:
dy-core + parameterizations

Earth System model components

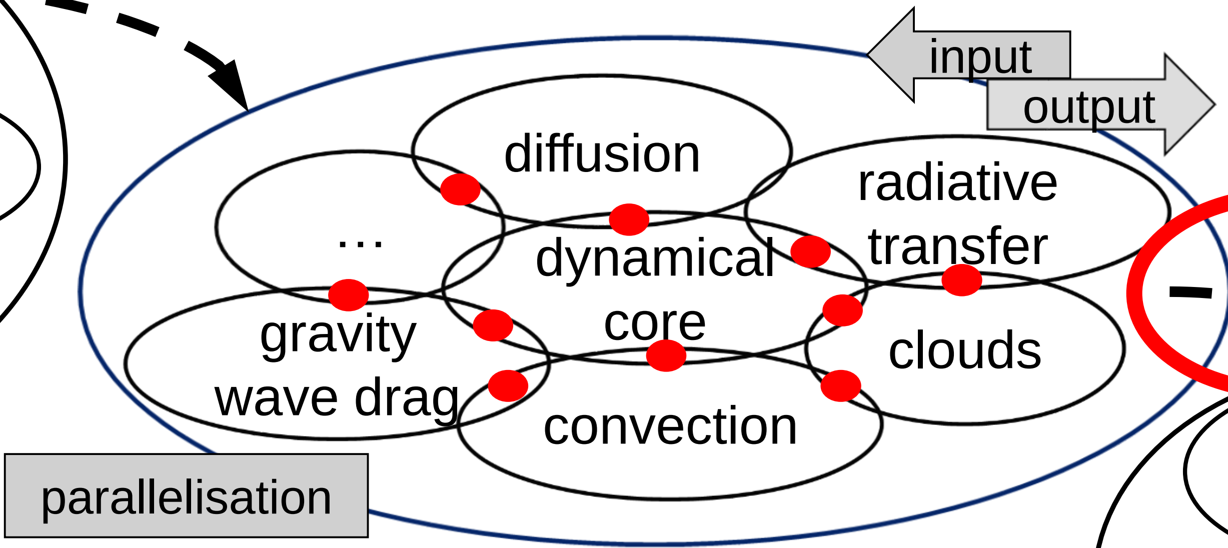


Earth System model components

How are the various components connected ?

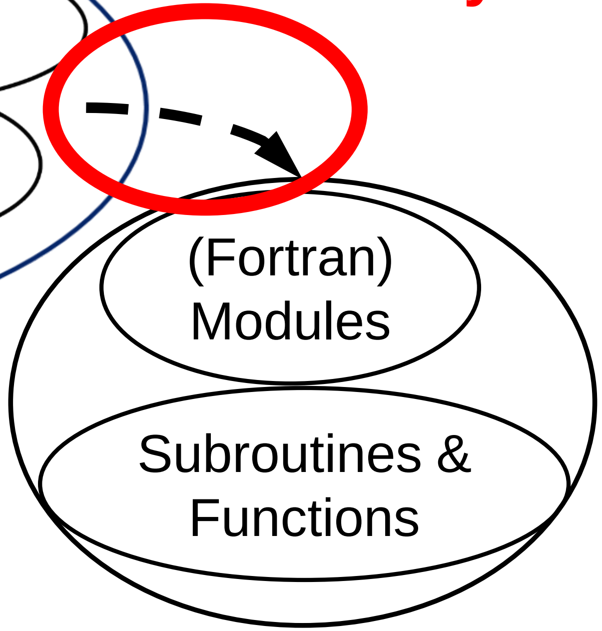


classical view:
coupled compartments



legacy A-GCM:
dy-core + parameterizations

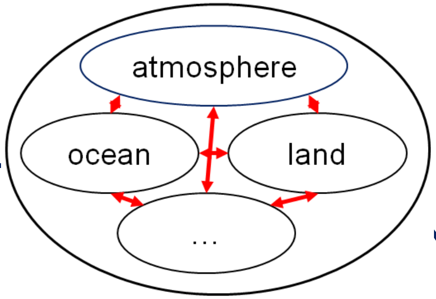
From messy to MESSy!

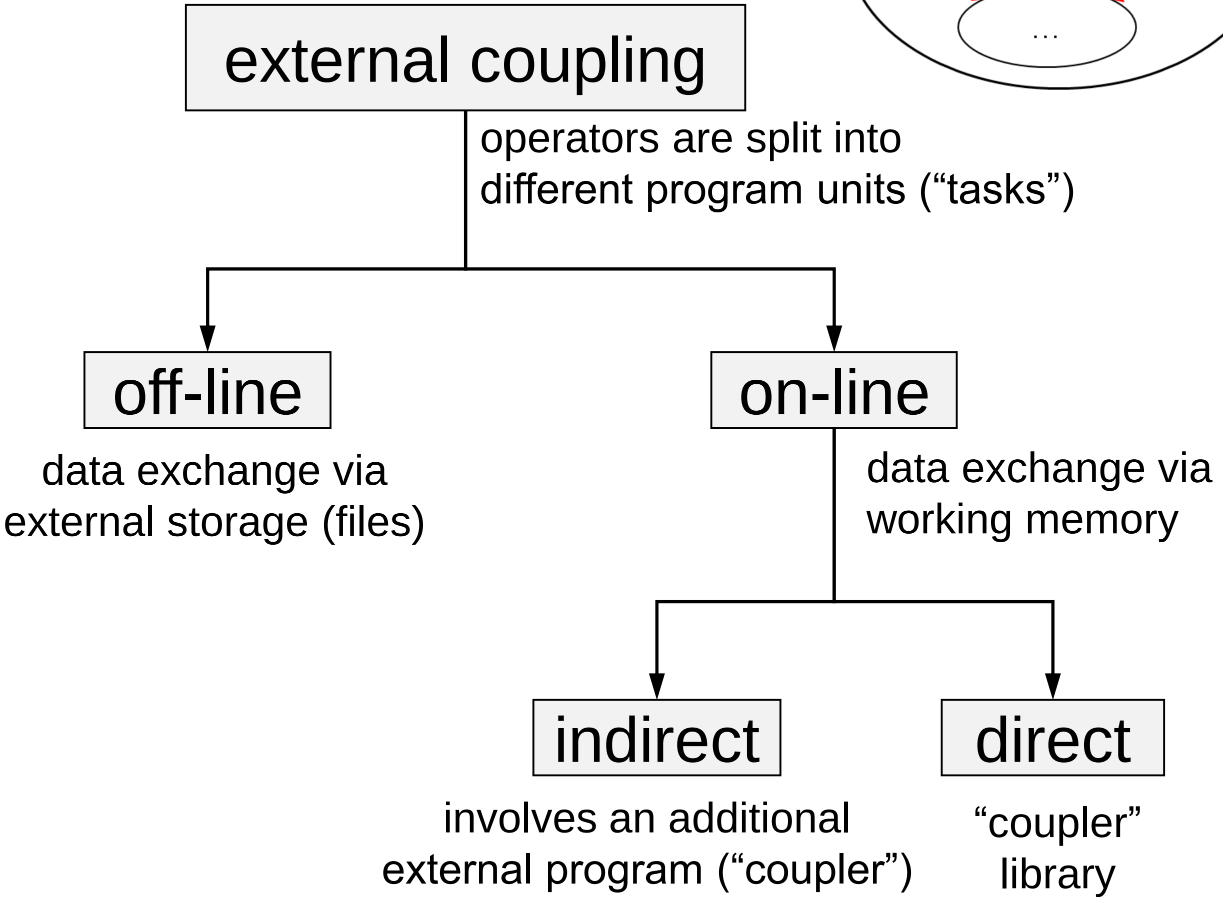


compartment **process** **language**

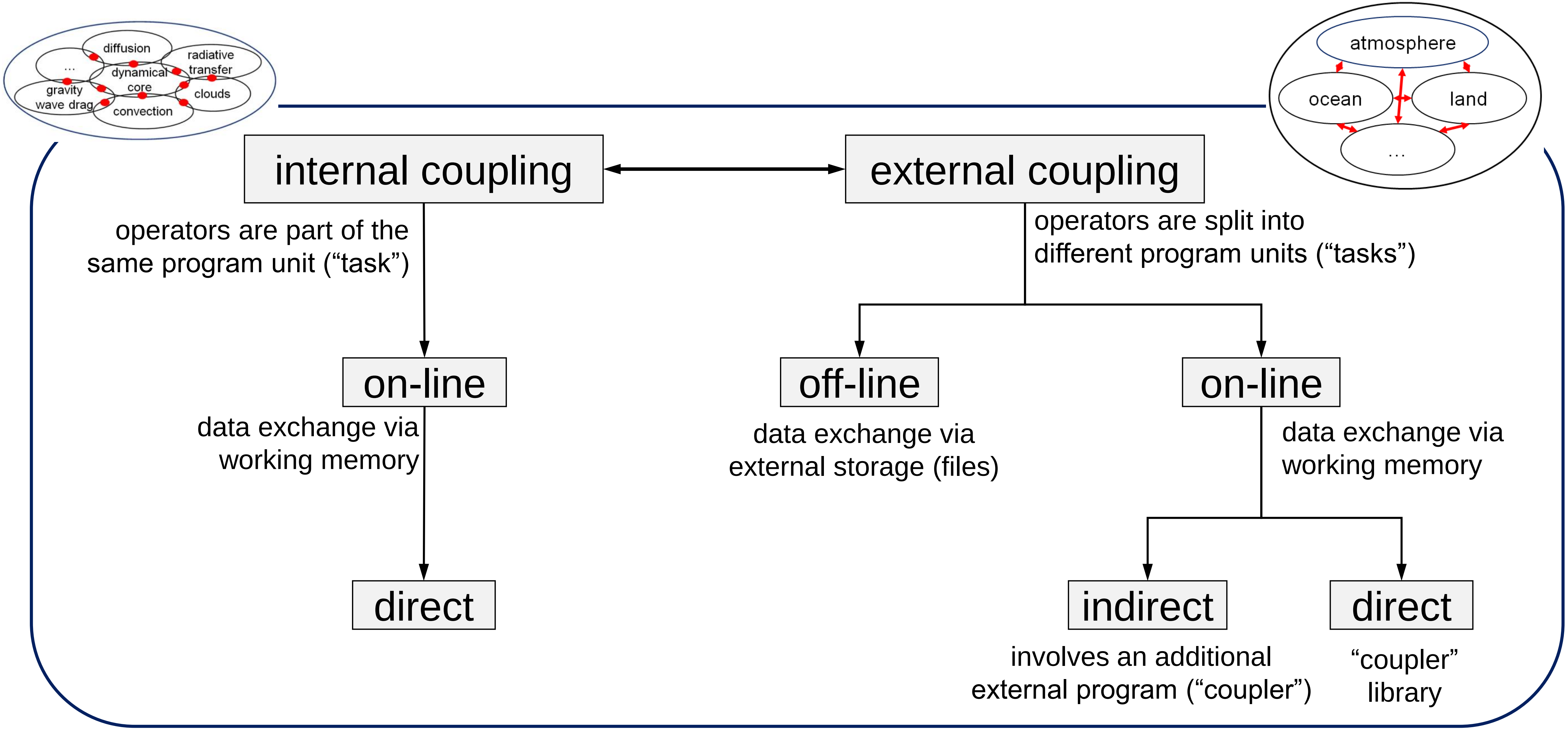
coarse **granularity** **fine**

Coupling (1-way & 2-way)

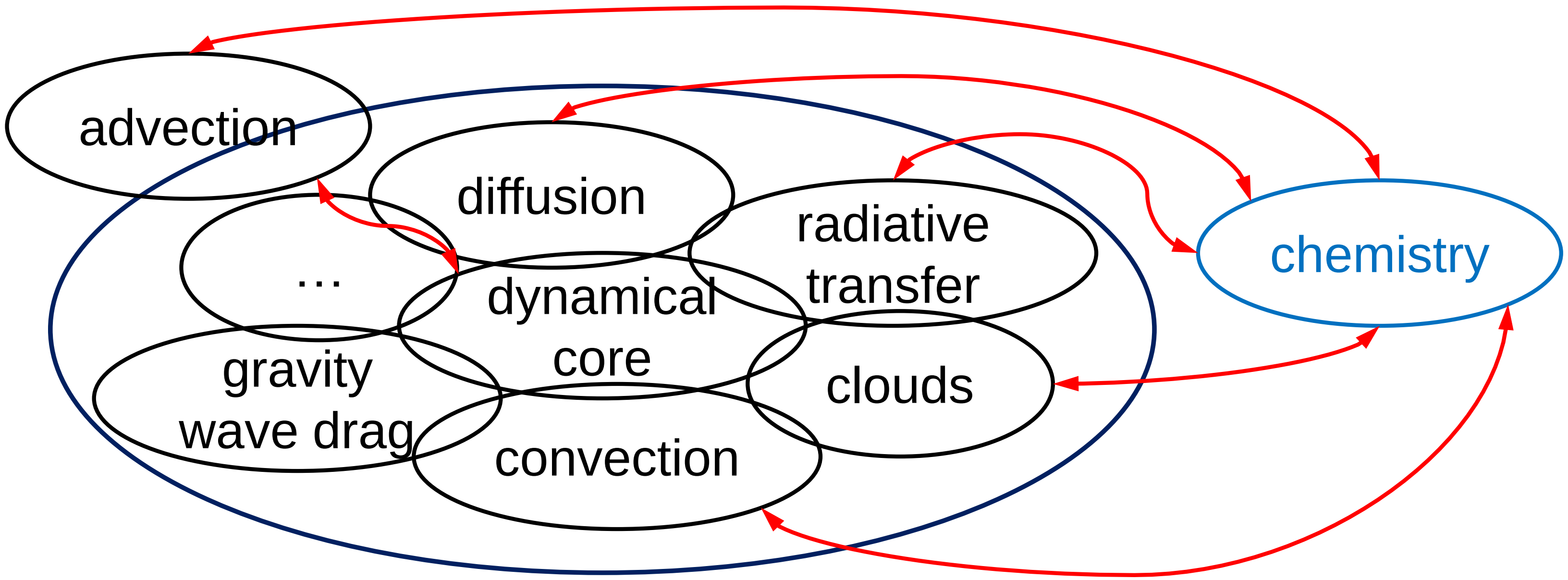




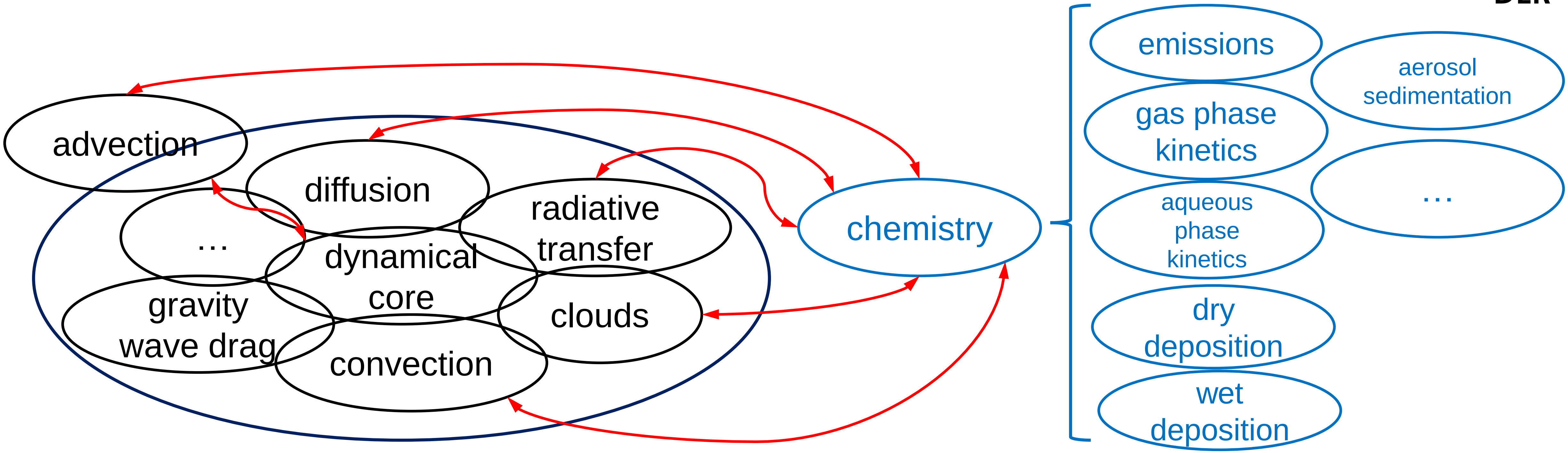
Coupling (1-way & 2-way)



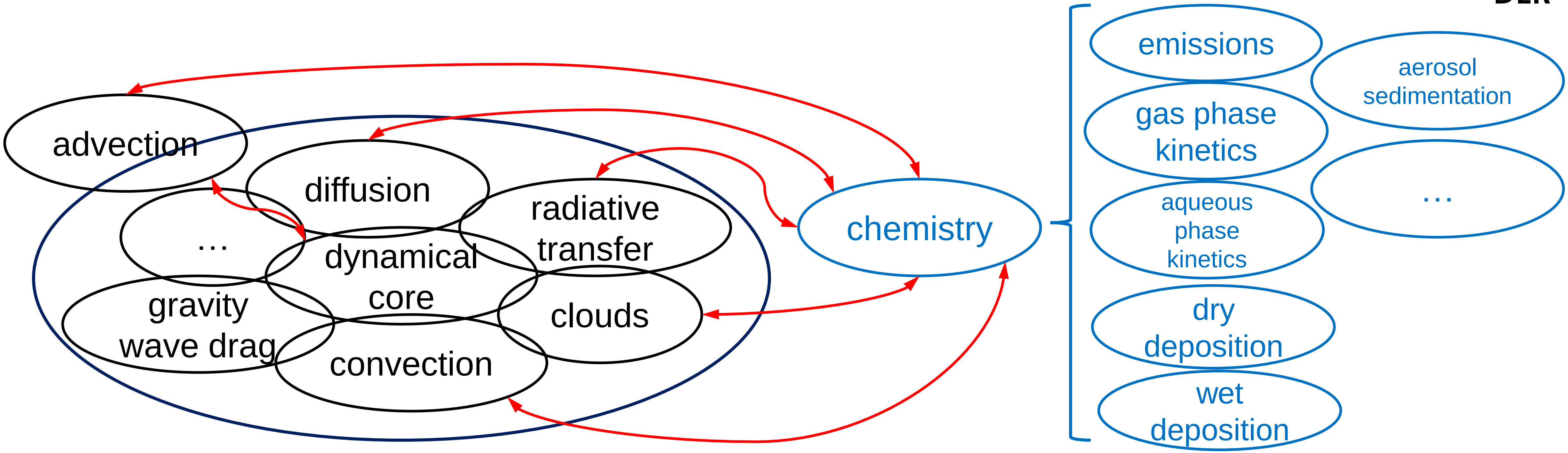
Adding atmospheric chemistry & aerosol



Adding atmospheric chemistry & aerosol

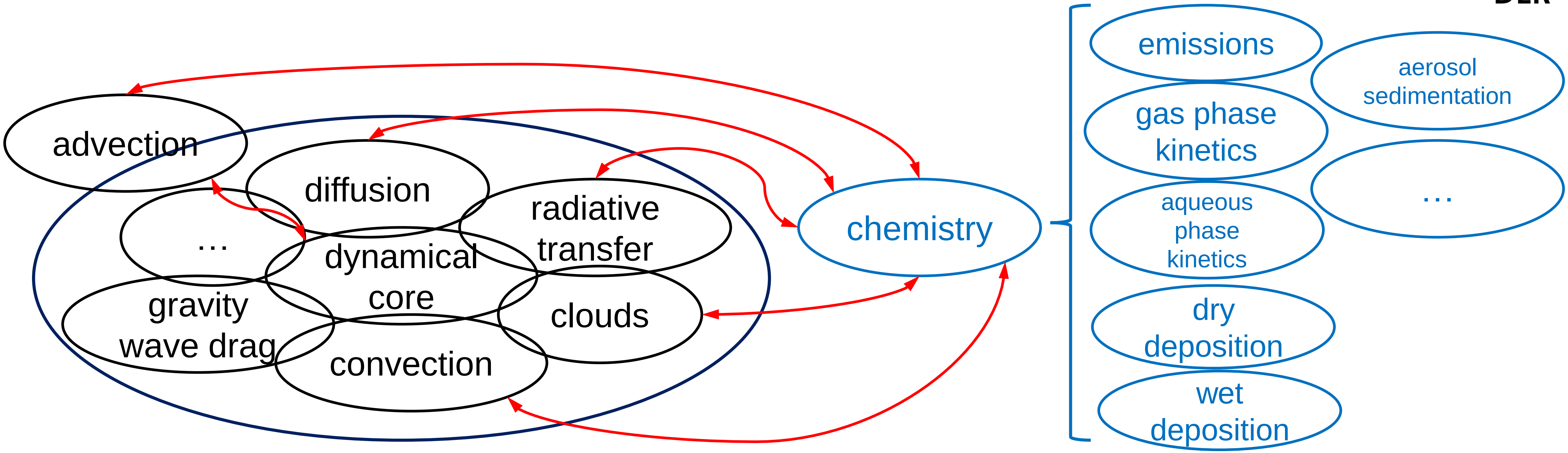


Adding atmospheric chemistry & aerosol



- 2-way (!) exchange
- large number of 3D fields to exchange
- interaction with many physical processes
- numerous additional processes

Adding atmospheric chemistry & aerosol



- 2-way (!) exchange
- large number of 3D fields to exchange
- interaction with many physical processes
- numerous additional processes

- "external coupling" not feasible
- "internal coupling" required
- but implementation into "legacy" code is not desirable, e.g.
  → maintenance
  → flexibility

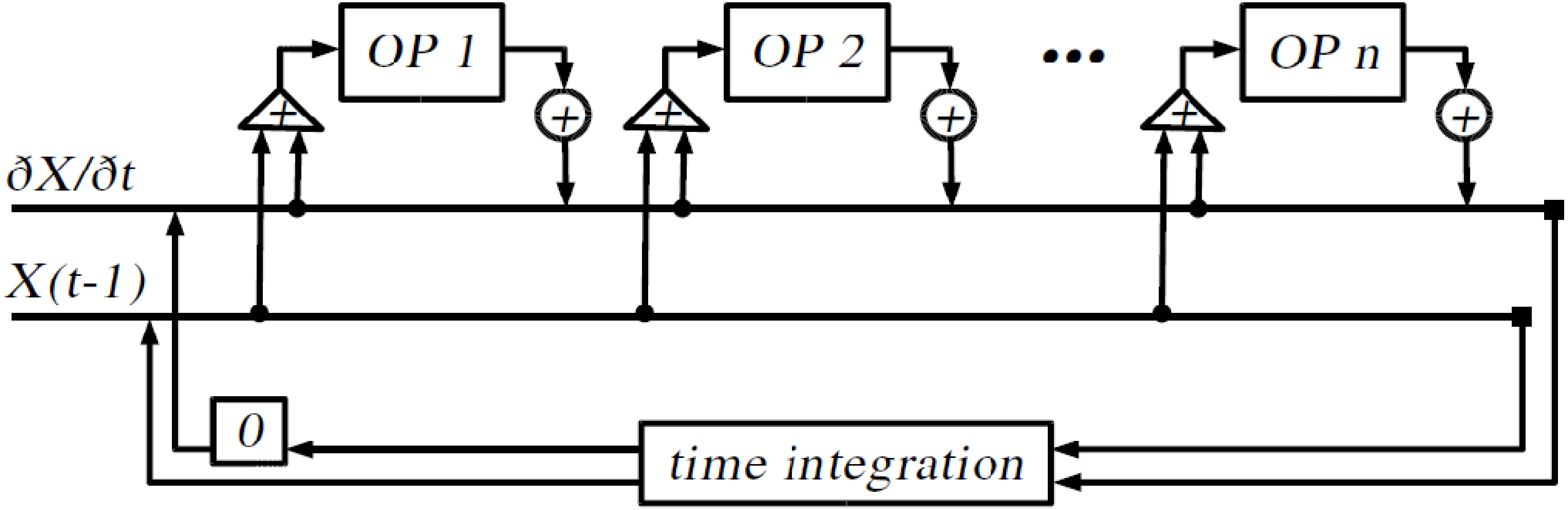
WHICH WAY OUT?

Photo by Viktor Forgacs on Unsplash

The fundamental concept: Operator splitting
(a numerical method to solve differential equations in “steps”)

… a very helpful approach:

- different PDE terms express different physics (e.g., on different time-scales)
- different numerical methods for different process descriptions required



The fundamental concept: Operator splitting
(a numerical method to solve differential equations in “steps”)

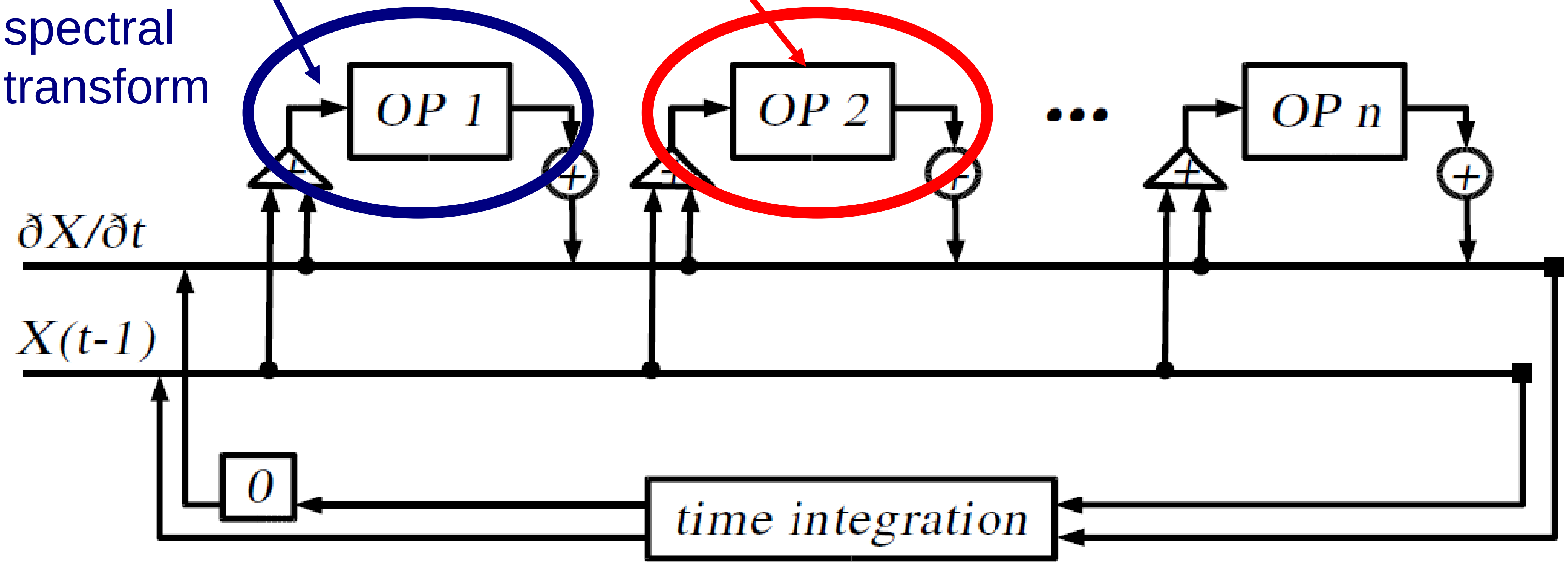
Example:

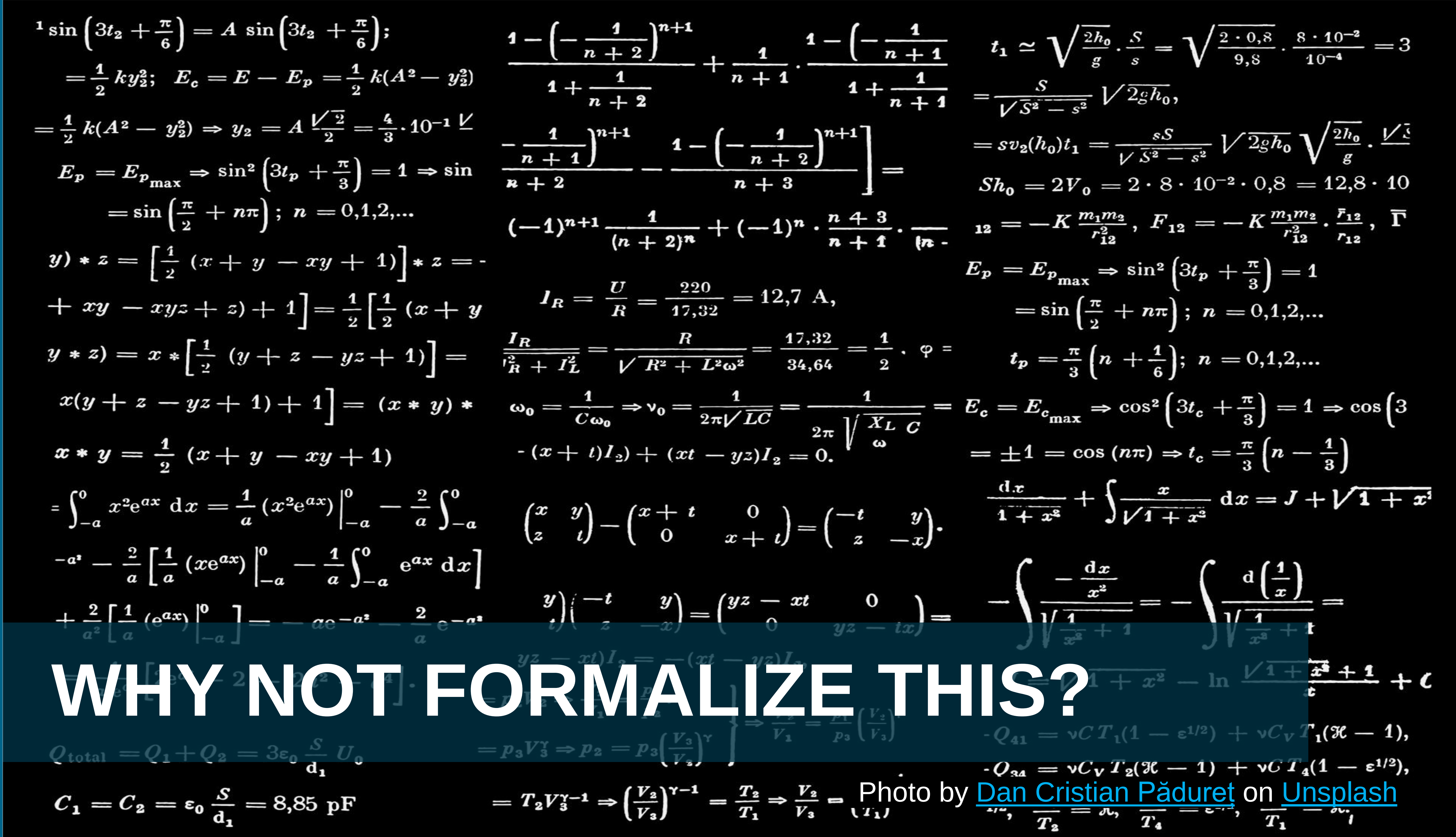
basic (dynamical) equations → coupled PDE system

reaction kinetics (chemistry) → coupled ODE system

Rosenbrock-3 (auto)

spectral transform

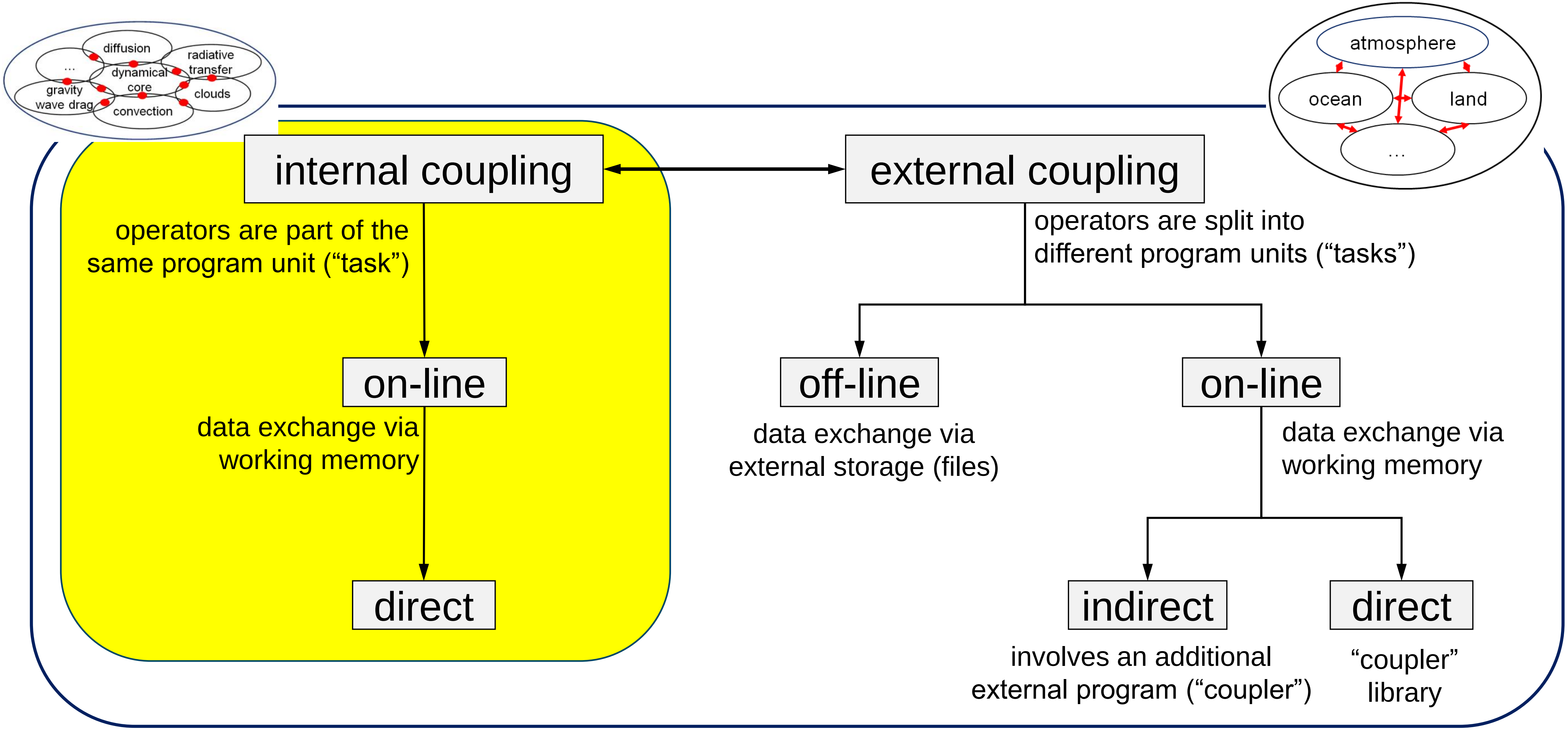




WHY NOT FORMALIZE THIS?

Photo by Dan Cristian Păduret on Unsplash

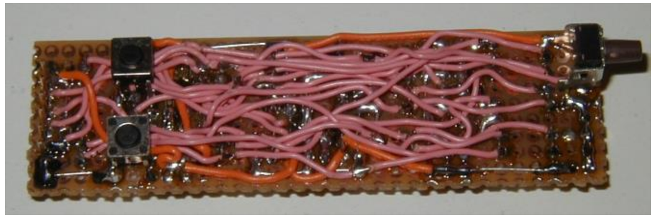
Coupling


diffusion
radiative transfer
...
dynamical core
gravity wave drag
clouds
convection
atmosphere
ocean
land
...
internal coupling
external coupling
operators are part of the same program unit ("task")
operators are split into different program units ("tasks")
on-line
data exchange via working memory
direct
off-line
data exchange via external storage (files)
on-line
data exchange via working memory
indirect
involves an additional external program ("coupler")
direct
"coupler" library

Internal coupling (online & direct)

“unstructured”

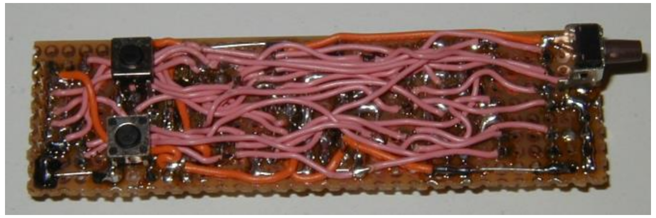
pure language level
(subroutines and modules)



Internal coupling (online & direct)

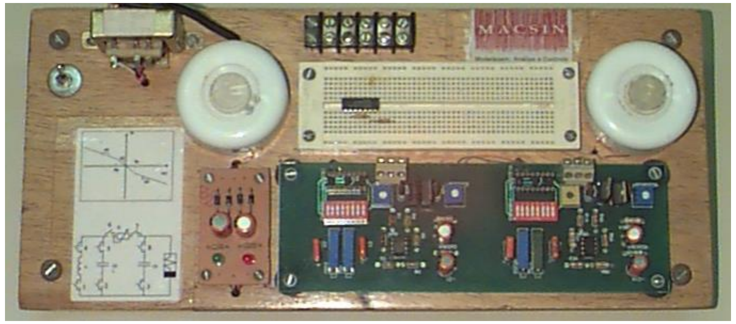
“unstructured”

pure language level
(subroutines and modules)



“structured”

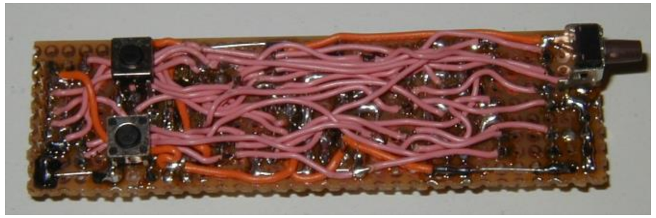
+ coding standard
+ (some) internal infrastructure
  - memory management
  - time management
  - I/O
  - orchestration / run control
  - check-pointing
  - grid structures
  - tracers
  - …
- **model specific!**



Internal coupling (online & direct)

DLR

“unstructured”	“structured”	“formalised”
pure language level (subroutines and modules)	+ coding standard + (some) internal infrastructure ▪ memory management ▪ time management ▪ I/O ▪ orchestration / run control ▪ check-pointing ▪ grid structures ▪ tracers ▪ … ▪ **model specific!**	+ **model independent formulation** + standard libraries + … ↓ **integrated framework**



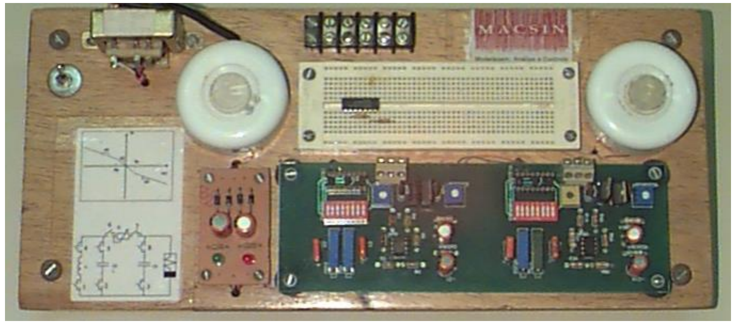


Photo by Vishnu Mohanan on Unsplash

Internal coupling (online & direct)

DLR

“unstructured”	“structured”	“formalised”
pure language level (subroutines and modules)	+ coding standard + (some) internal infrastructure ▪ memory management ▪ time management ▪ I/O ▪ orchestration / run control ▪ check-pointing ▪ grid structures ▪ tracers ▪ … ▪ **model specific!**	+ **model independent formulation** + standard libraries + … **integrated framework**

decreasing “*spaghetti-ness*” of code

increasing “separation of concerns”

Integrated Frameworks

- Examples:
  - Earth System Modelling Framework (ESMF) [*https://www.earthsystemcog.org/projects/esmf/*]
  - Modular Earth Submodel System (MESSy) [*https://www.messy-interface.org*]
  - Community Earth System Model (CESM) [*http://www.cesm.ucar.edu/models/ccsm4.0/*]
  - Flexible Modelling System (FMS) [*https://www.gfdl.noaa.gov/fms/*]
  - …

Advantages:
- fine granular structure of models
  - full "separation of concerns"
  - natural "dwarfs"
    (for porting and optimisation)
- well defined (model independent!) API
- high flexibility
- code sharing possible
- readable code structure

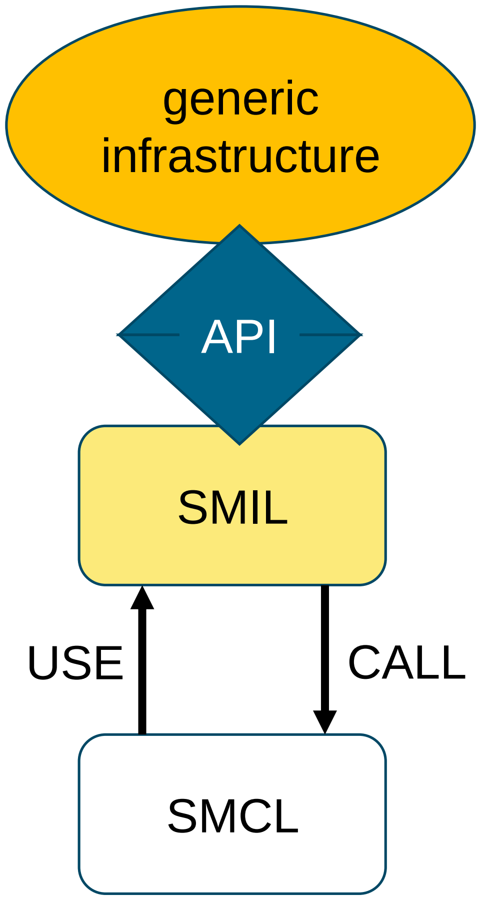
Disadvantages:
- intrusive for "legacy" codes
- model specific "infrastructure" components need to be abandoned / doubled & connected

```
      repr%pdecomp%start(i,jr) = start(i,repr%order_mem2out(jr))
      repr%pdecomp%cnt(i,jr)   = cnt(i,repr%order_mem2out(jr))
      cs1 = cs1 + repr%pdecomp%cnt(i,jr)
      repr%pdecomp%ml(i,jr)  = ml(i,repr%order_mem2out(jr))
      repr%pdecomp%mu(i,jr)  = mu(i,repr%order_mem2out(jr))
      IF (repr%pdecomp%cnt(i,jr) /= 0) &
           cs2 = cs2 + ( repr%pdecomp%mu(i,jr) - repr%pdecomp%ml(i,jr) + 1)
   END DO segment_loop
   IF (cs1 /= cs2) THEN
      WRITE(*,*) 'ERROR: (REPRESENTATION ',TRIM(repr%name) &
           ,', RANK ',jr,'): ',cs1,' =/= ',cs2
      status = 2024 ! REPRESENTATION SEGMENTATION MISMATCH
      RETURN
   END IF
END DO rank_loop

! CHECK MEMORY SIZE
IF (zlchk) THEN
```

NOW, WHAT ABOUT MESSy?

Some generic concepts of MESSy: Separation & Modularisaton

1. process (granularity!) → operator
   - operator output entirely determined by input (→ dwarfs, tests, …)
   - 2 software layers (Fortran modules) for each operator
     - “connection” via API to generic infrastructure
     - “machinery” (independent of infrastructure)
2. “generic infrastructure”:
   - generic data-types (dimensions, representations, attributes, **objects**, grids, …)
   - I/O, memory management, orchestration, …
   - API for “**wireless**” access to **objects**



Some generic concepts of MESSy: Wireless connections

WIRED

```
MODULE CHEM

USE PHYS, ONLY: temperature

...

! do something with temperature

END MODULE CHEM
```

```
MODULE PHYS

USE CHEM, ONLY: do_something

...

 CALL do_something(…, temperature, …)

END MODULE PHYS
```

- compile-time dependency of modules CHEM & PHYS
- ***variable*** name “hard-wired”
- often long lists of parameters at subroutine call
- → highly inflexible, maximally intrusive

Some generic concepts of MESSy: Wireless connections

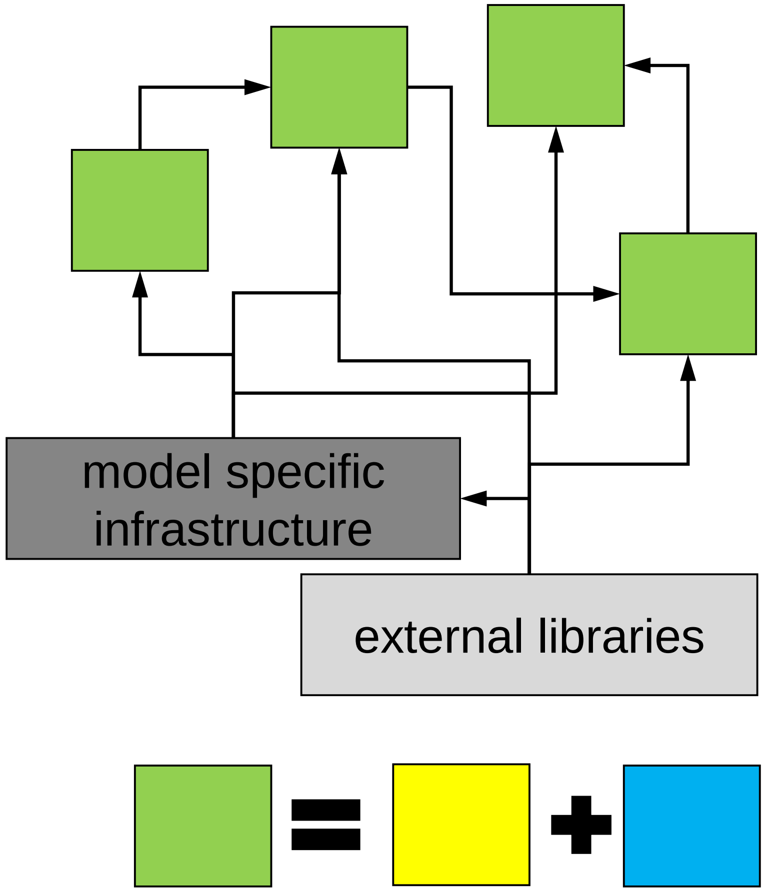
```
MODULE PHYS

USE INFRASTRUCTURE, ONLY: object_define
...

CALL object_define('phys','temperature',...)
...

END MODULE PHYS
```

```
MODULE CHEM

USE INFRASTRUCTURE, ONLY: object_access

REAL(dp), POINTER, DIMENSION(:,:,:) :: temp
...

CALL object_access('phys','temperature',temp)
...

! do something with local pointer temp

END MODULE CHEM
```

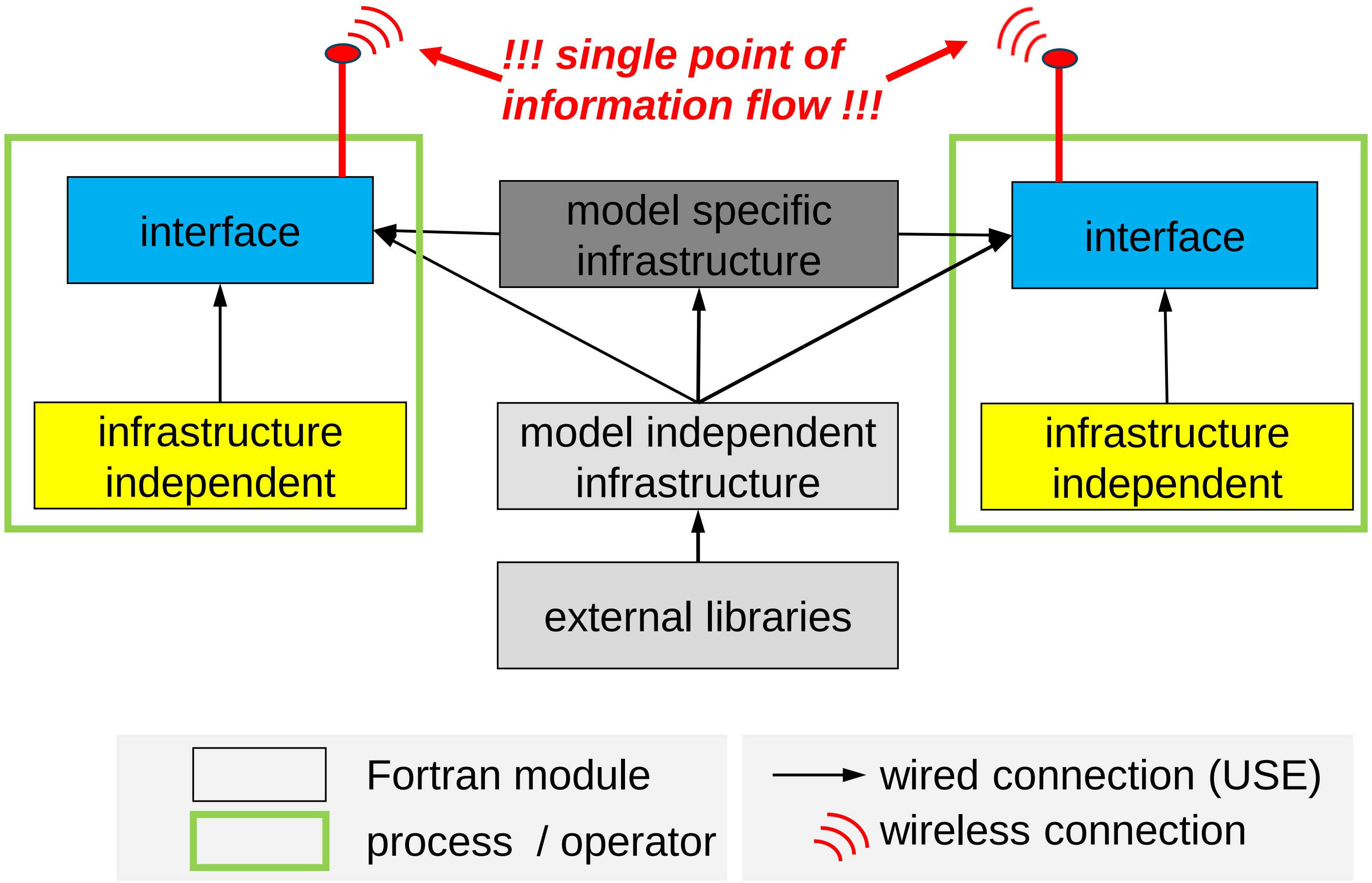
- no compile-time dependency of CHEM & PHYS
- dependency on common INFRASTRUCTURE (API)
- ***object*** name (convention, standardisation)
- *object meta information* can be easily added (e.g. units)

Some generic concepts of MESSy: Hierarchical software layering

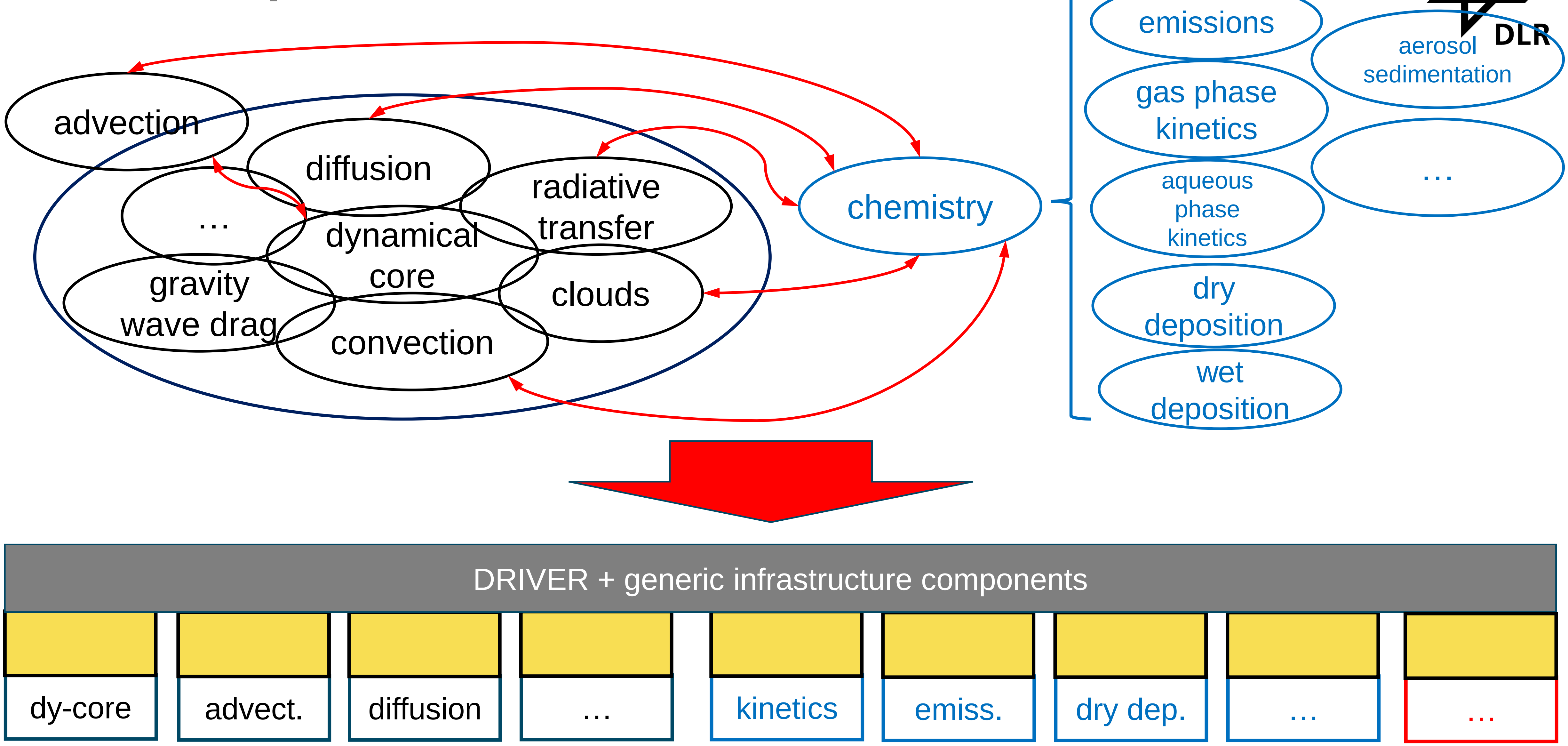
non-hierarchical software ~~layering~~

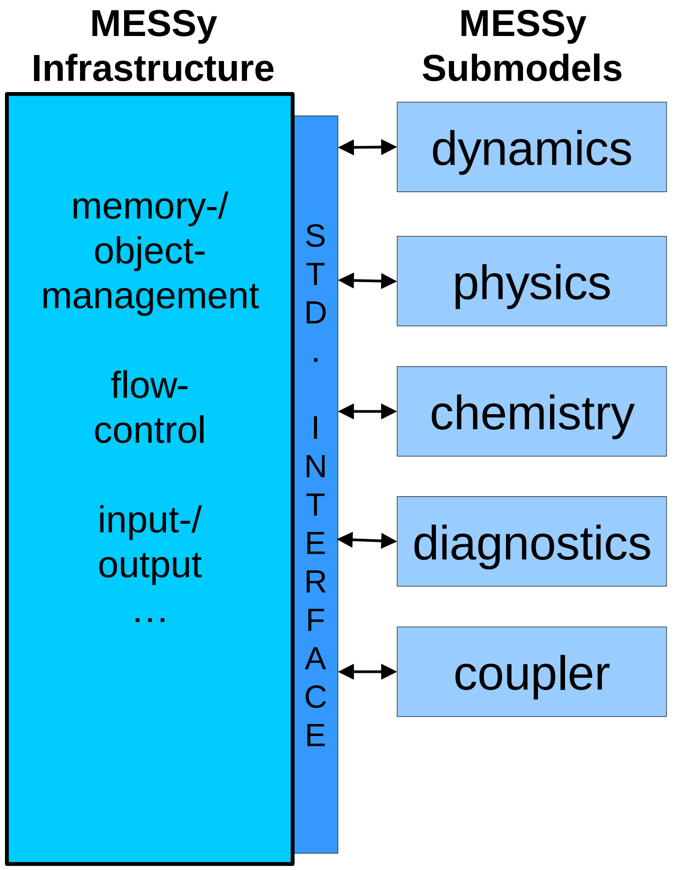


hierarchical software layering



In consequence …

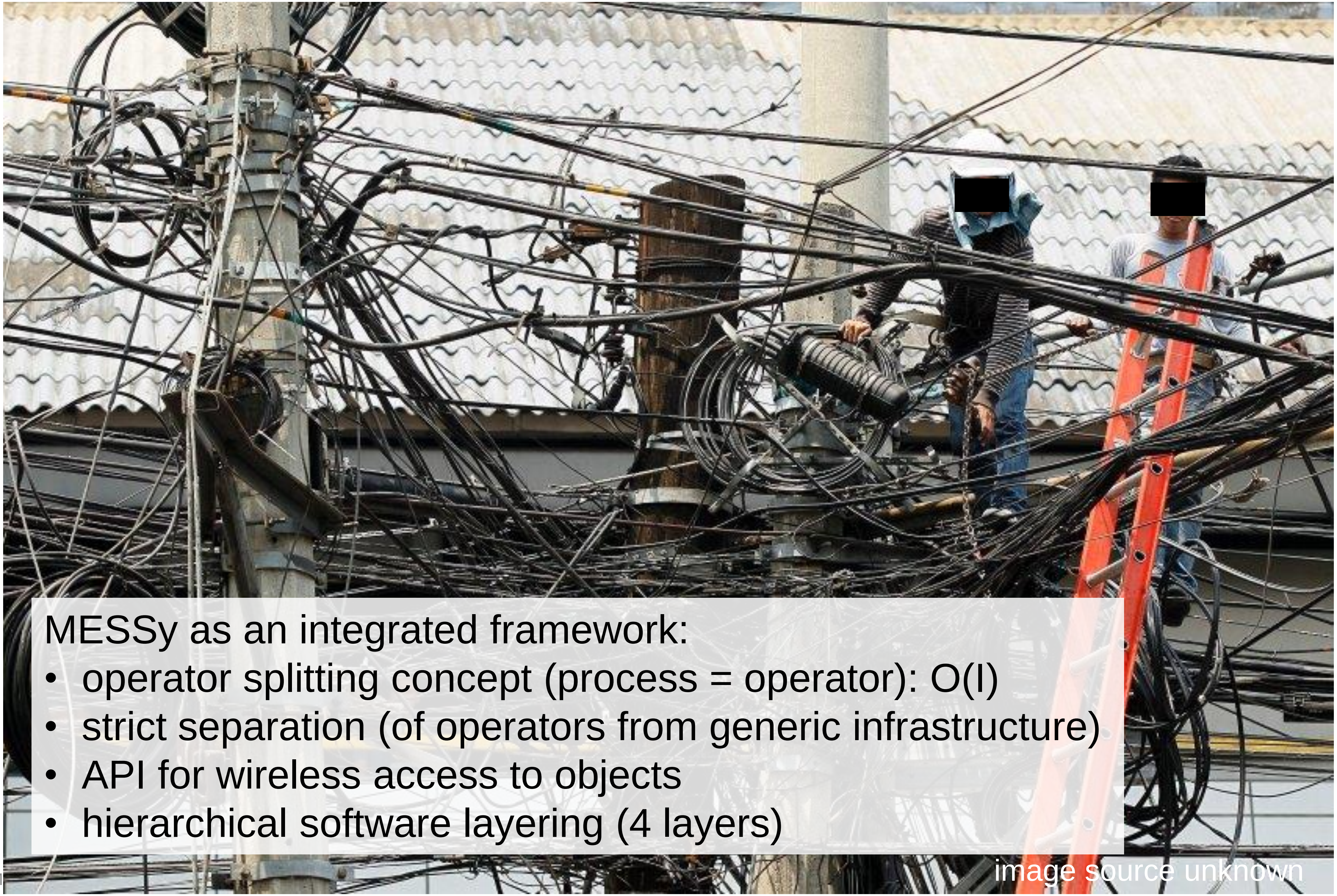




MESSy as an integrated framework:

- operator splitting concept (process = operator): O(I)
- strict separation (of operators from generic infrastructure)
- API for wireless access to objects
- hierarchical software layering (4 layers)

Thank you very much for your attention!



MESSy as an integrated framework:

- operator splitting concept (process = operator): O(I)
- strict separation (of operators from generic infrastructure)
- API for wireless access to objects
- hierarchical software layering (4 layers)

image source unknown

Impressum

Thema: Generic API concepts in ESM – lessons learned from MESSy

Datum: 15.11.2023

Autor: Patrick Jöckel

Institut: Institut für Physik der Atmosphäre

Bildcredits: Alle Bilder „DLR (CC BY-NC-ND 3.0)“