Adaptive multi-tier intelligent data manager for Exascale

admire-eurohpc.eu

ADMIRE Users Day

The Environmental Application

Diana Di Luccio, Ciro Giuseppe De Vita, Gennaro Mellone,
Livia Marcellino, Angelo Ciaramella, Giulio Giunta and **Raffaele Montella**

Consorzio Nazionale per l'Informatica
University of Naples "Parthenope"

December 12th 2023.

Barcelona Supercomputing Center

Grant Agreement number: 956748 — ADMIRE — H2020-JTI-EuroHPC-2019-1

Presentation outline

- Introducing the Environmental Application workflow
- The Environmental Application building blocks
- WaComM++: Water quality Community Model
- WaComM++: Designed with the computational malleability in mind
- DagOnStar: Directed Acyclic Graph on Anything
- Conclusions

How to use this presentation.

The full presentation is intended as ADMIRE dissemination material for an half day front seminar followed by half day hands on tutorial.

The Environmental App is provided as an example within the DagOnStar repository.

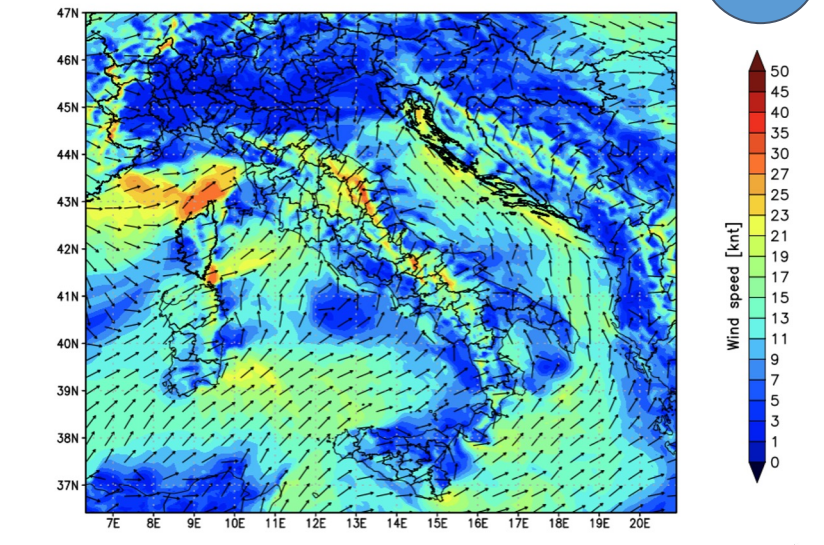
The WaComM++ demo is available as step by step hands-on tutorial on the WaComM++ repository.

Introducing Environmental Application

Sort of a “digital twin” of a real environment… but not just a *buzzword*!

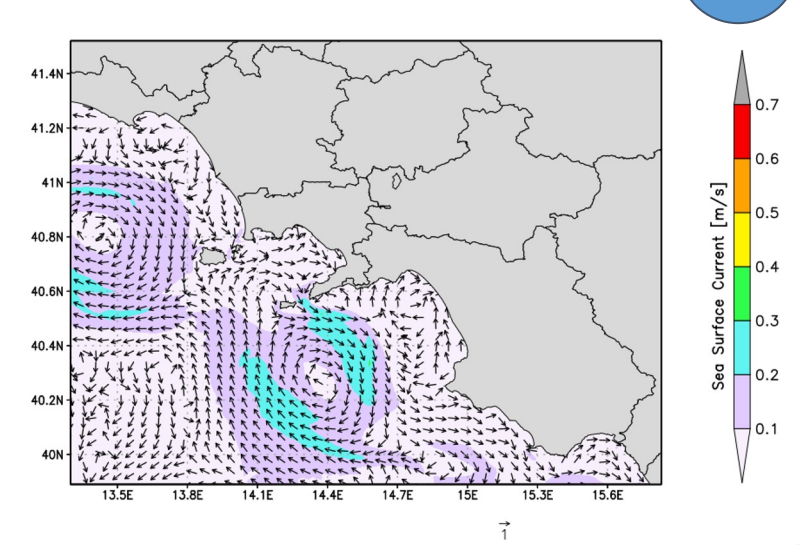
Products

1



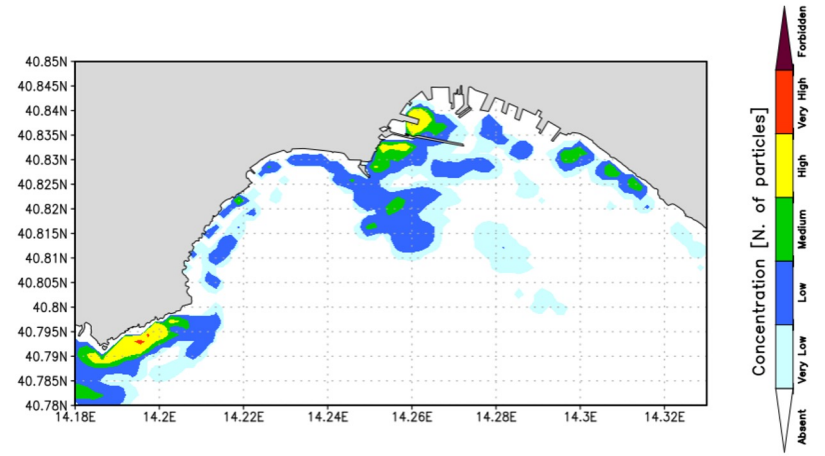
Weather forecasts (7d, 1000m)

2



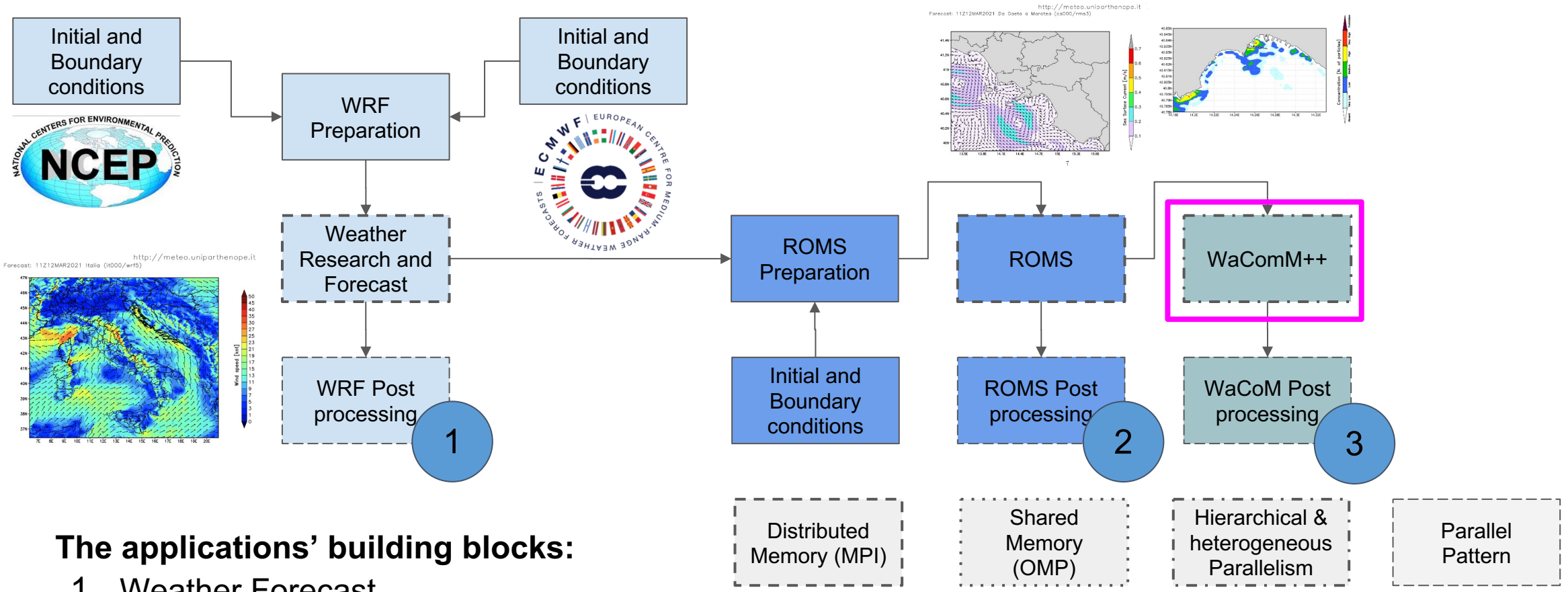
Sea currents (7d, 160m)

3



Inerts tracing (7d, 160m)

The workflow



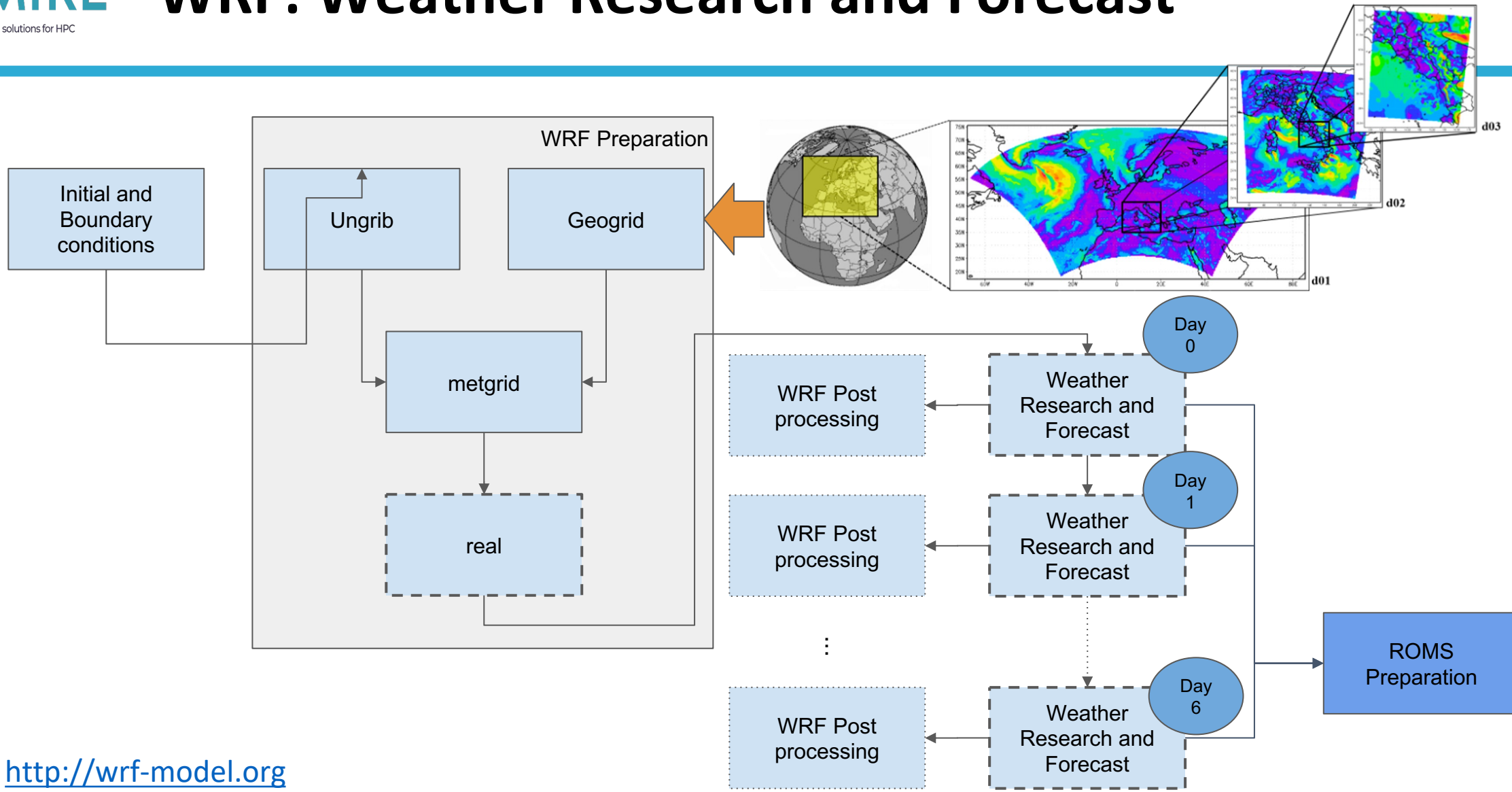
The applications' building blocks:

1. Weather Forecast
2. Ocean Dynamics Forecast
3. Inert Transport and Diffusion Forecast

The Environmental Application building blocks

Like a Lego toy but with a more challenging interface system

WRF: Weather Research and Forecast

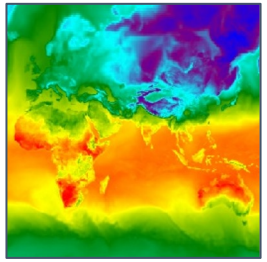


http://wrf-model.org

https://www.mmm.ucar.edu/models/wrf

WRF: Input & Output

input



Global Forecast System
National Centers for Environmental Prediction

Resolution: 0.5 degrees/3h
4 dataset per day
39 GB/run - 5.6 GB/day.

Storage: 107.4 GB/run

Scratch: 105 GB/run

output

Domain 1, 25Km, 1h, 168h

History:
2.2 GB/day, 15.4 GB/run

Archive:
539 MB/day, 3.8 GB/run

Restart:
0.4 GB/day, 2.8 GB/run

Domain 2, 5Km, 1h, 168h

History:
4.3 GB/day, 30.1 GB/run

Archive:
2 GB/day, 14 GB/run

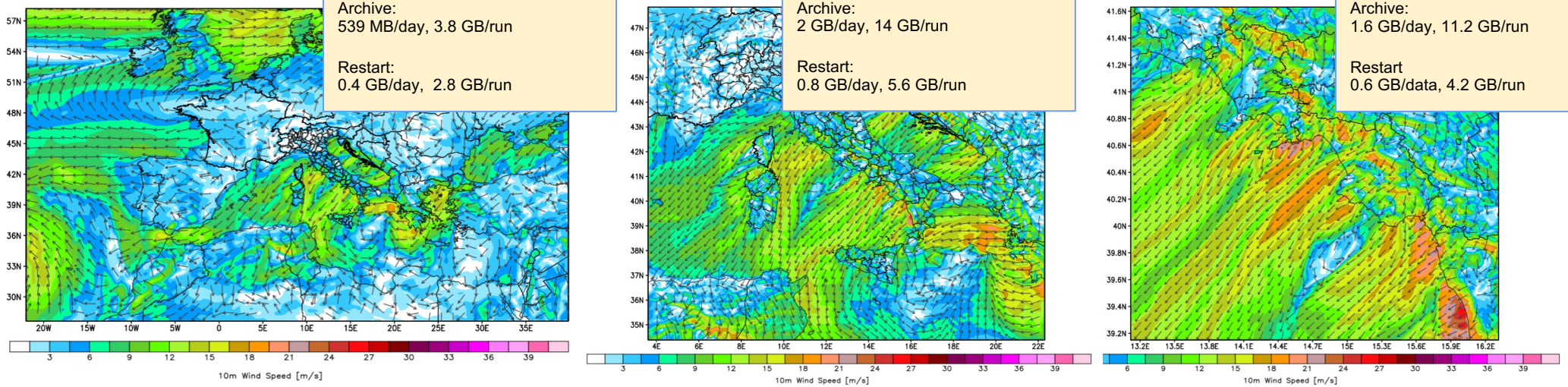
Restart:
0.8 GB/day, 5.6 GB/run

Domain 3, 1Km, 1h, 168h

History:
2.9 GB/day, 20.3 GB/run

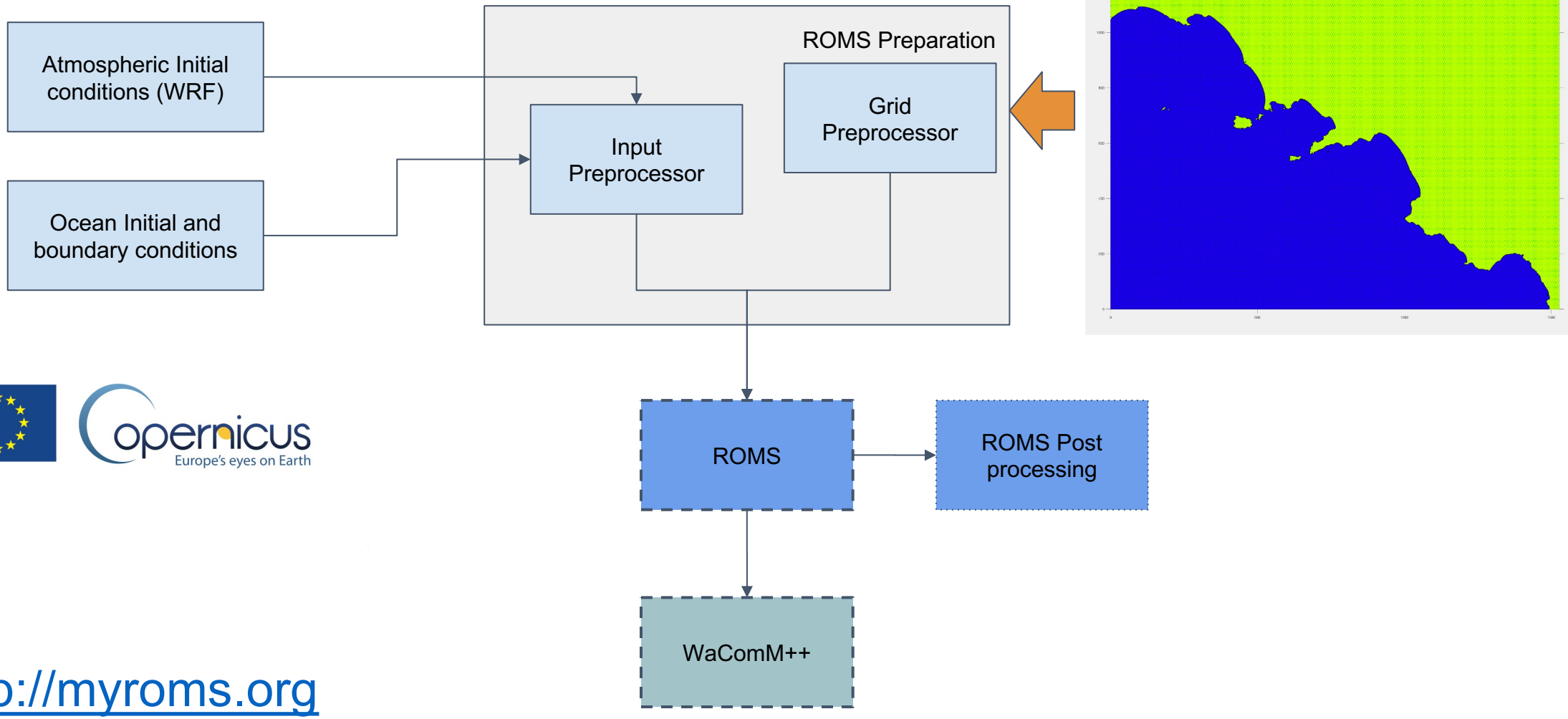
Archive:
1.6 GB/day, 11.2 GB/run

Restart
0.6 GB/data, 4.2 GB/run



https://www.mmm.ucar.edu/models/wrf

ROMS: Regional Ocean Model System

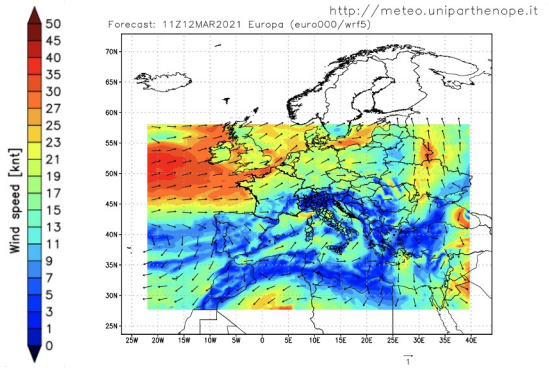


http://myroms.org

https://www.myroms.org

ROMS: Input & Output

input

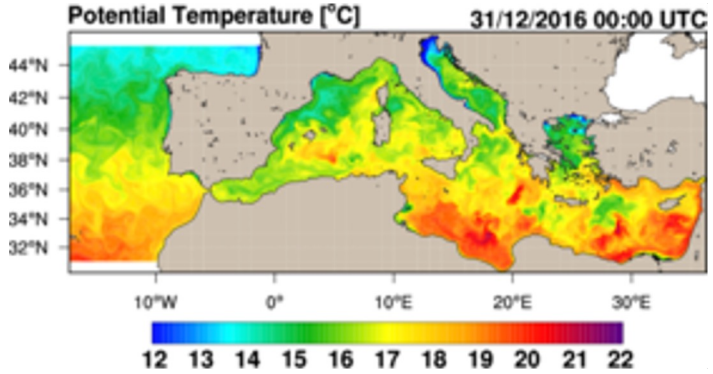


WRF - CMMMA

27 GB/run

Copernicus

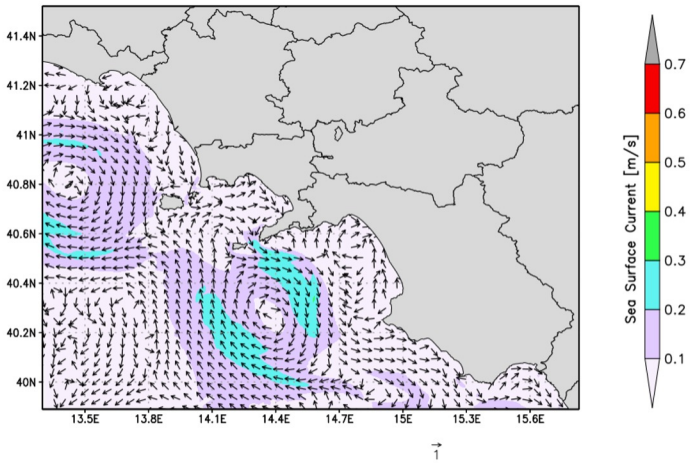
0.4 GB/run



output

Storage: 492 GB/run

Scratch: 332.5 GB/run



Domain 3, 160 m, 1h, 168h

History:
63 GB/day, 441 GB/run

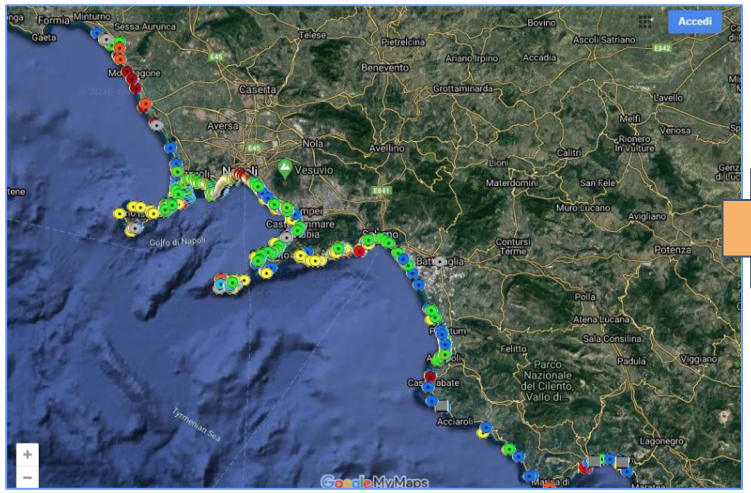
Archive:
7.3 GB/day, 51.1 GB/run

https://www.myroms.org

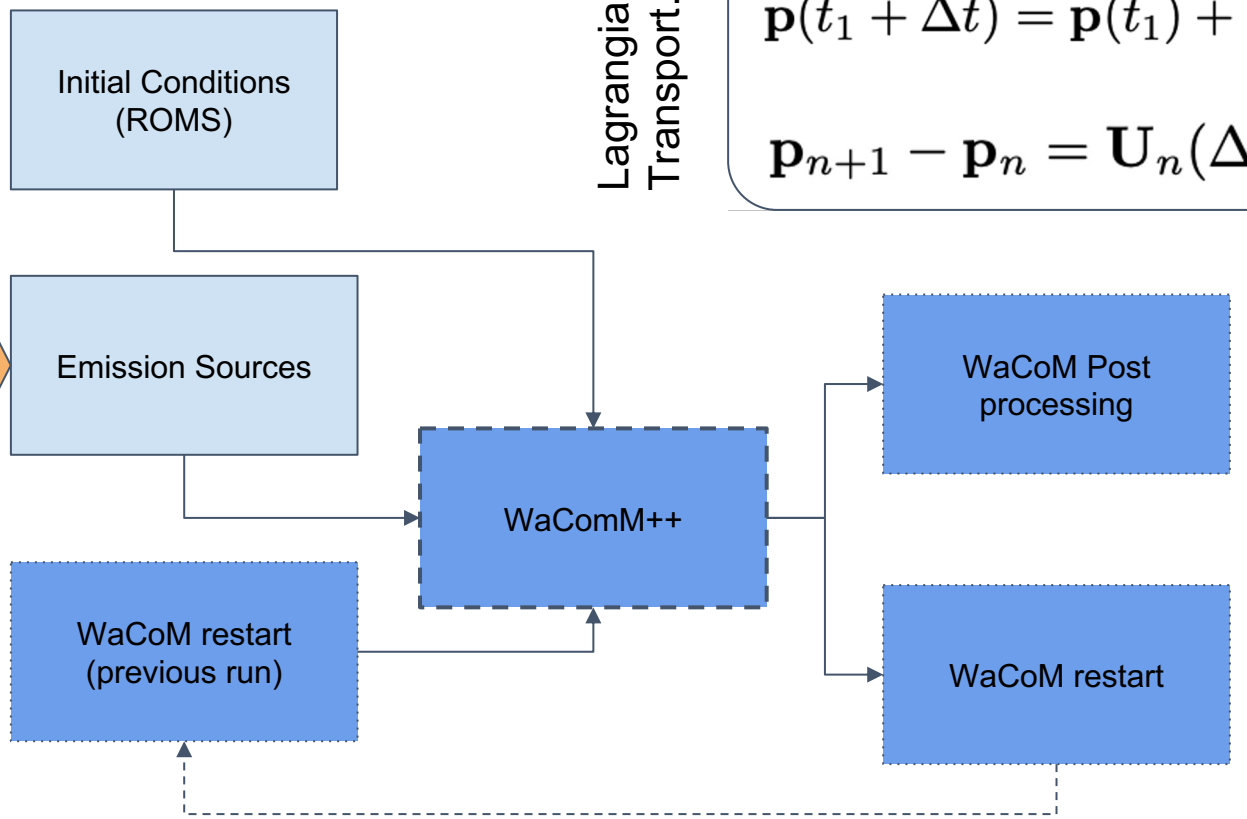
WaComM++:
Water quality Community Model

A highly scalable high-performance Lagrangian transport and diffusion model for marine pollutants assessment

WaComM++: Water quality Community Model ++



Sewer Outlets



Lagrangian Transport...

$$\mathbf{p}(t_1 + \Delta t) = \mathbf{p}(t_1) + \int_{t_1}^{t_1+\Delta t} \mathbf{u}(\mathbf{p}(t), t)dt$$

$$\mathbf{p}_{n+1} - \mathbf{p}_n = \mathbf{U}_n(\Delta t) + \rho_n$$

$$\sigma = \sqrt{2K\Delta t}$$

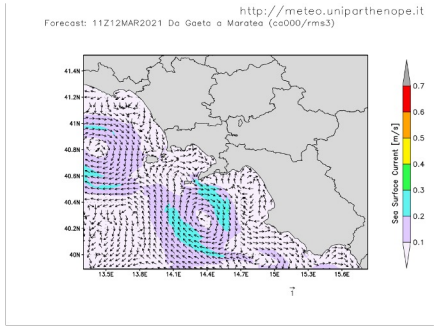
$$\sigma(z_n) = \sigma(0)\left(1 + \frac{z_n}{H}\right)$$

...and Diffusion.

https://github.com/ccmmma/wacommplusplus

WaComM++: Input & Output

input



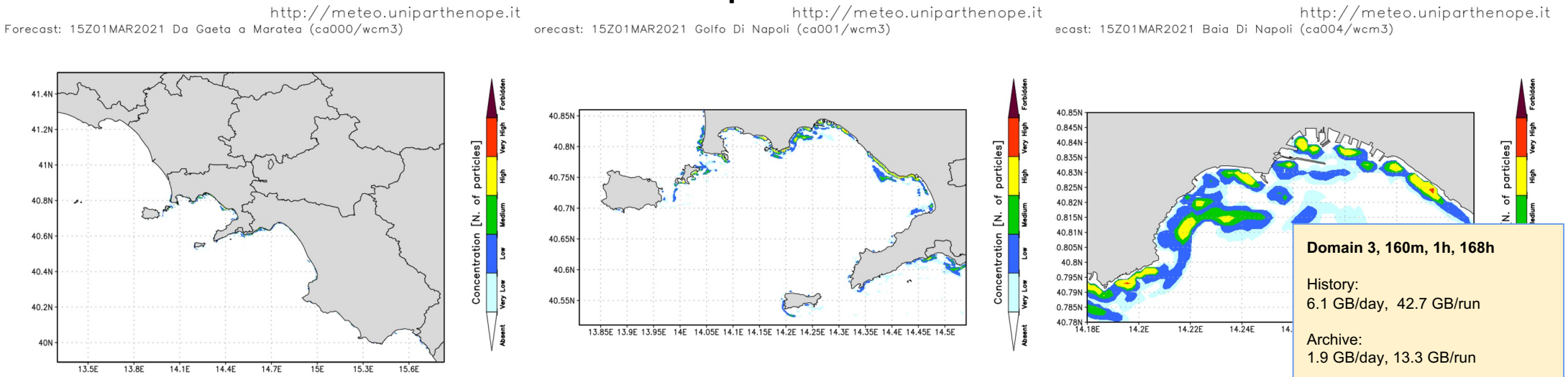
ROMS - CMMMA

305 GB/run

Storage: 68.4 GB/run

Scratch: 353 GB/run

output



Domain 3, 160m, 1h, 168h

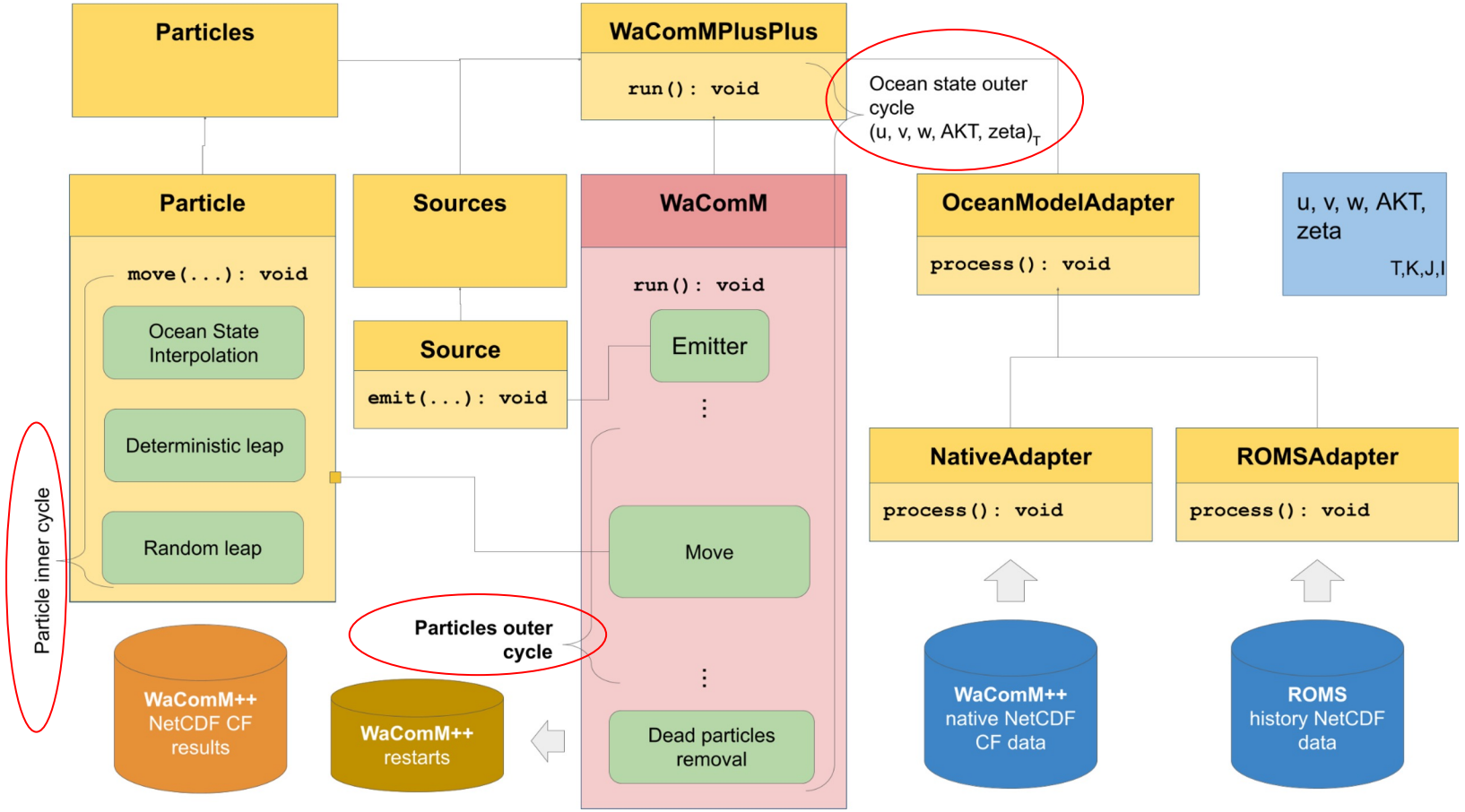
History:
6.1 GB/day, 42.7 GB/run

Archive:
1.9 GB/day, 13.3 GB/run

Restart:
1.8 GB/day, 12.4 GB/run

https://github.com/ccmmma/wacommpluspl

WaComM++ architecture



The overall computation is performed over three nested cycles:

Ocean state outer cycle: for each time-referenced dataset (usually 1-hour), a WaComM component is instanced.

Particles outer cycle: moves the particle to process using ocean data.

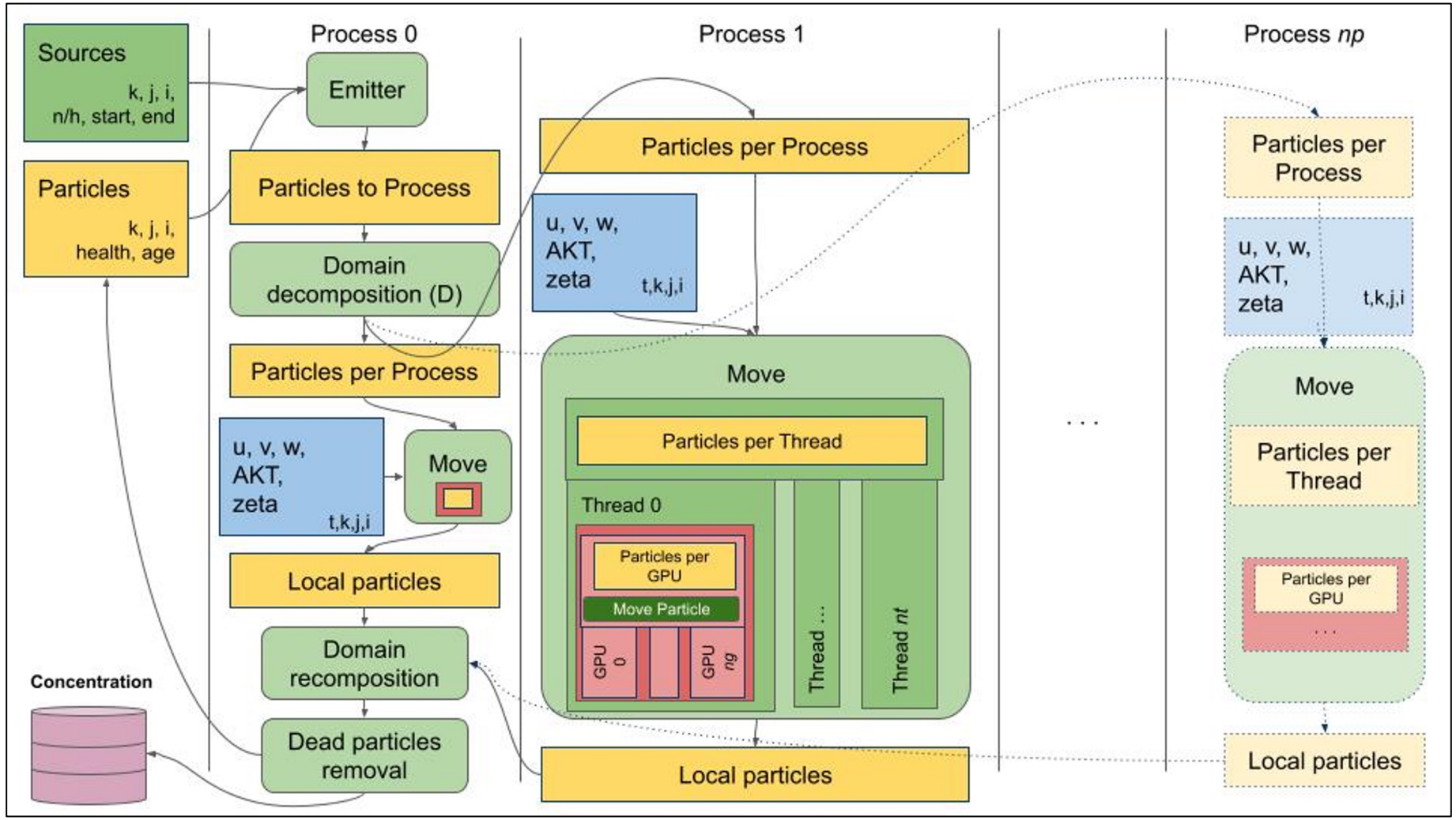
Particle inner cycle: moves the particle within the considered time slice, applying the Lagrangian transport and diffusion equations integrated on a given time step.

While time-dependent iterations characterize the ocean state outer cycle and the inner particle cycle, the particles' outer cycle has been hierarchically parallelized because each particle movement is independent of the others.

https://github.com/ccmmma/wacommplusplus

WaComM++ hierarchical parallelization schema

With multi-GPU paradigm.



WaComM++ evaluation configuration

We use the following configurations:

- 25 million particles spilled out by a single coastal source located in the Gulf of Napoli (Campania, Italy)
- no restart mode
- 24 h of simulation
- **Different parallelization schema**

PurpleJeans (HPC Tests):

2 Intel(R) Xeon(R)Gold 5218 CPU@2.30GHz 16 cores each)

4 Nvidia Tesla V100SXM232GB 5120 CUDA cores each)

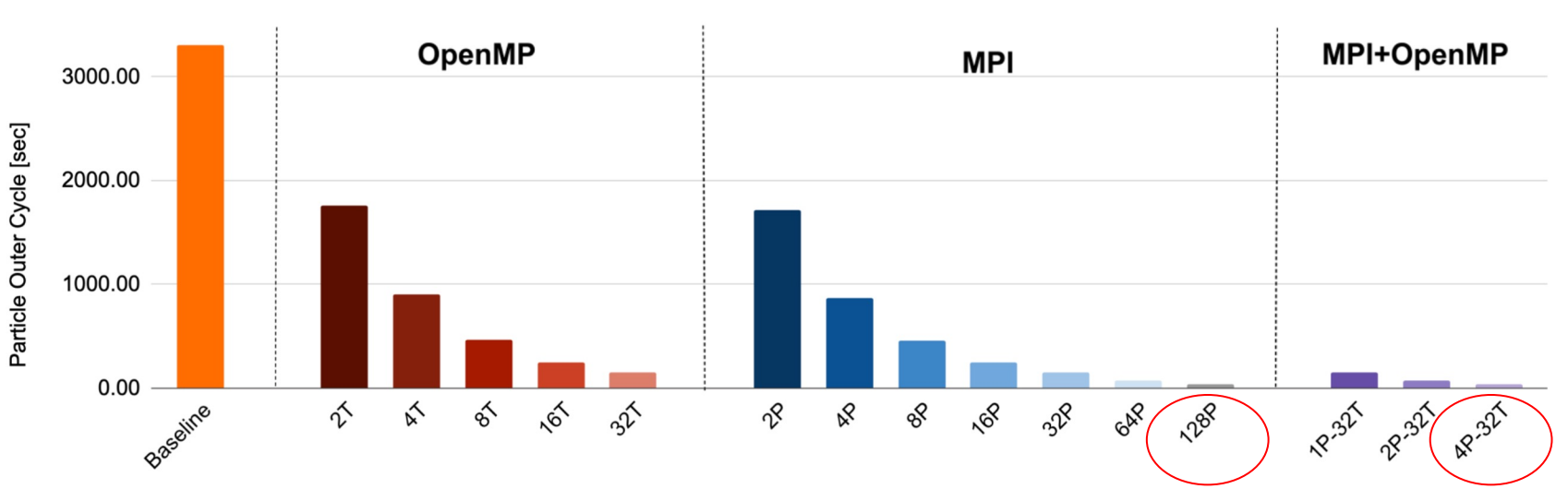
https://rcf.uniparthenope.it

WaComM++ evaluation configuration

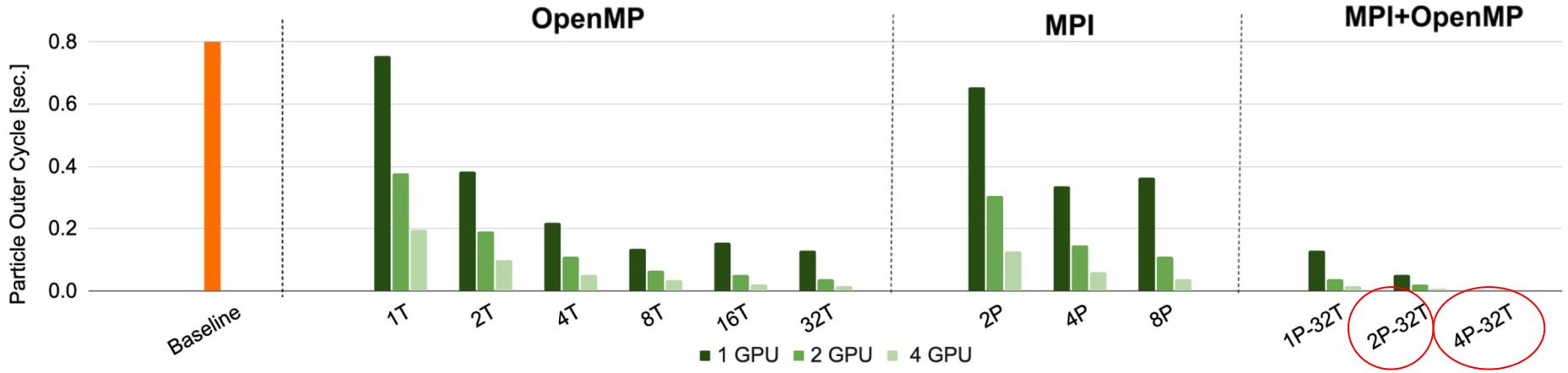
- **Baseline** → one process, only one thread, and no GPUs (sequential mode)
- **Distributed Memory (MPI approach)**: 1, 2, 4, 8, 16, 32, 64, and 128P on four computing nodes, considering only 1T.
- **Shared Memory (OpenMP approach)**: Considering only one MPI process, we used 1, 2, 4, 8, 16, and 32T on one computing node with 1P.
- **Shared Memory and CUDA (OpenMP-CUDA approach)**: We consider both the single GPU and the multi-GPU cases. A single process is tested from shared memory threads varying from 1 to 32T, sharing 1, 2, and 4G.
- **Distributed Memory and CUDA (MPI-CUDA approach)**: We consider both the single GPU and the multi-GPU cases. Multiple processes are tested, varying from 1 to 8P, using only one thread sharing 1, 2, and 4G.
- **Distributed Memory, Shared Memory, and CUDA (MPI-OpenMP-CUDA approach)**: We consider both the single GPU and the multi-GPU cases. A multiple 1 to 4P is tested using shared memory fixed on 32T sharing 1, 2, and 4G.

https://github.com/ccmmma/wacommplusplus

Wacomm++ evaluation



Particle Outer-Cycle execution times results without CUDA approach

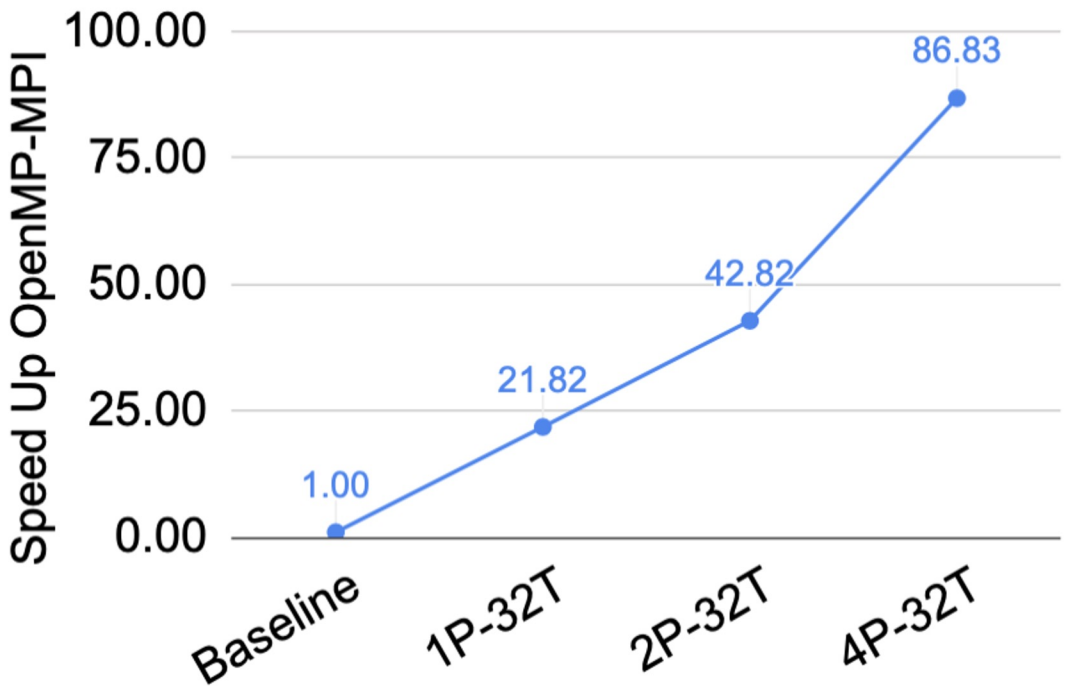


Particle Outer-Cycle execution times results using CUDA approach.

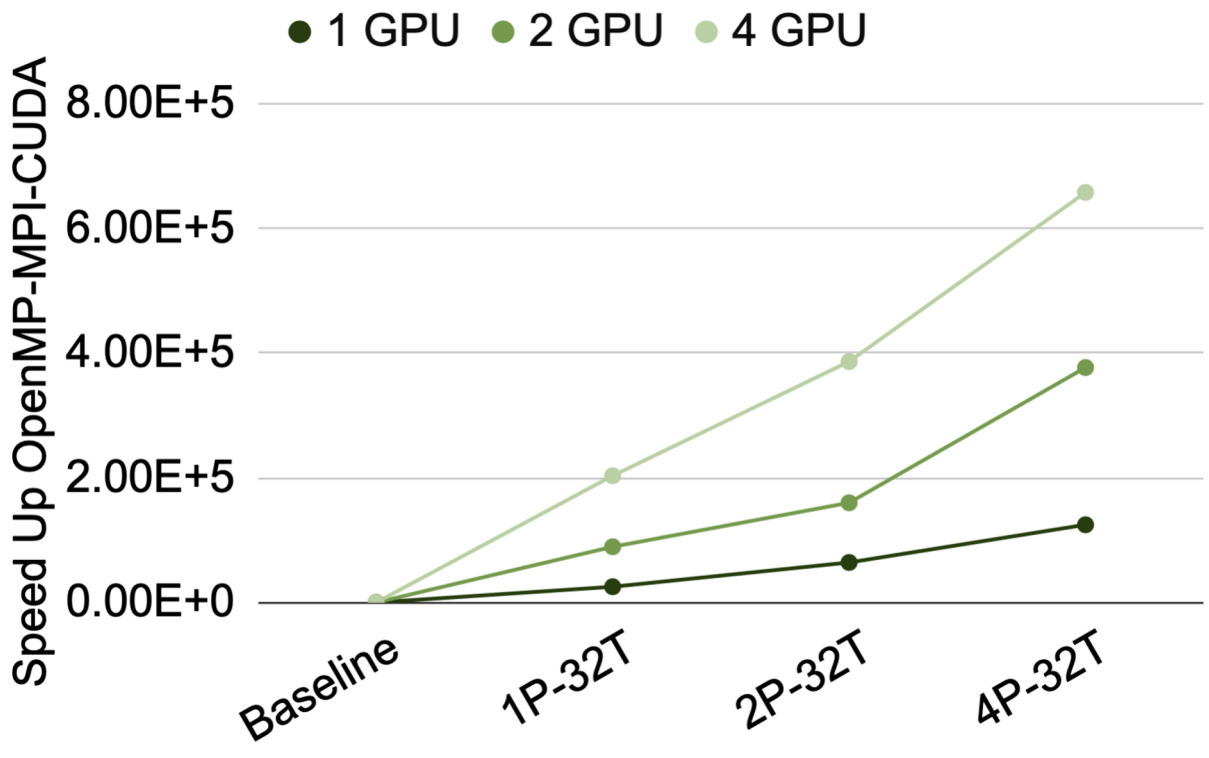
"P" is the number of MPI processes, "T" is the number of OpenMP processes, and "G" identifies the number of GPU devices considered for the computation.

https://github.com/ccmmma/wacommplusplus

Wacomm++ evaluation



Speed up of the OpenMP-MPI approach



Speedup of the OpenMP-MPI-CUDA approach

https://github.com/ccmmma/wacommplusplus

WaComM++: designed with the computational malleability in mind

Malleability is no good for all recipes, but for this one is really, really tasty!

Computational malleability in mind

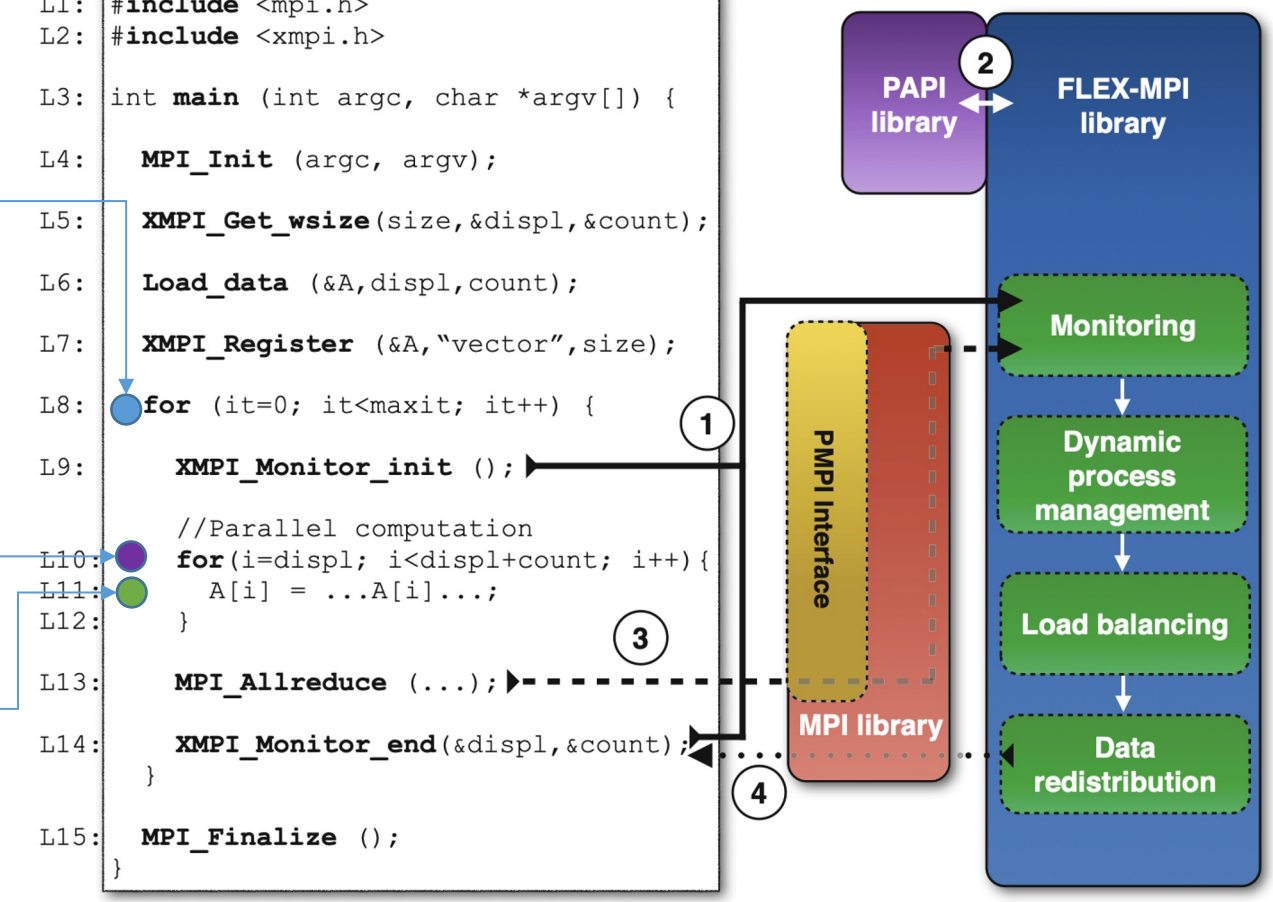
- 3 main cycles:

Ocean outer cycle (iterate over ocean status by time)

Outer particles cycle (compute in parallel each inert particle path)

Inner particle cycle (compute the particle movement, sequential)

```
L1:  #include <mpi.h>
L2:  #include <xmpi.h>
L3:  int main (int argc, char *argv[]) {
L4:    MPI_Init (argc, argv);
L5:    XMPI_Get_wsize(size,&displ,&count);
L6:    Load_data (&A,displ,count);
L7:    XMPI_Register (&A,"vector",size);
L8:    for (it=0; it<maxit; it++) {
L9:      XMPI_Monitor_init ();
         //Parallel computation
L10:     for(i=displ; i<displ+count; i++){
L11:       A[i] = ...A[i]...;
L12:     }
L13:     MPI_Allreduce (...);
L14:     XMPI_Monitor_end(&displ,&count);
       }
L15:   MPI_Finalize ();
     }
```



- Flex-MPI

Martin, Gonzalo, Maria-Cristina Marinescu, David E. Singh, and Jesus Carretero. "FLEX-MPI: an MPI extension for supporting dynamic load balancing on heterogeneous non-dedicated systems." In *European Conference on Parallel Processing*, pp. 138-149. Springer, Berlin, Heidelberg, 2013.

Malleable-WaComM++: Flex-MPI

```
1: T ← 0
2: MPI_Init(...)
3: p ← MPI_Comm_rank()
4: s ← MPI_Comm_size()
5: type ← EMPI_Get_type()
6: if type=EMPI_NATIVE then
7:     MPI_Barrier(EMPI_COMM_WORLD)
8: else
9:     T ← EMPI_Get_shared()
10: end if
11: while T ≤ T_max do
12:     if p = 0 then
13:         D_T ← emitNewParticles()
14:     end if
15:     displ, count ← EMPI_Get_wsize (p, s, D.size, ...)
16:     EMPI_Register_vector ("D",D_T, MPI_Datatype, D.size());
17:     if type=EMPI_NATIVE then
18:         MPI_Barrier(EMPI_COMM_WORLD)
19:     else
20:         T ← EMPI_Get_shared()
21:     end if
22:     EMPI_Monitor_init ();
23:     move(D_T, (u,v,w,zeta,AKT)_T, S)
24:     MPI_Allgatherv (D_T+desp, count, MPI_Datatype, ...);
25:     if p = 0 then
26:         removeDeadParticles(D_T)
27:         saveConcentration(D_T)
28:     end if
29:     count, desp, vcounts, displs ← EMPI_Monitor_end (rank, size, T, T_max, ...);
30:     EMPI_Free (D_T, "D")
31:     status ← EMPI_Get_status ()
32:     if state=EMPI_REMOVED then
33:         break
34:     end if
35:     T ← T + 1
36: end while
37: MPI_finalize()
```

Process initialization by means of malleability

Ocean state outer cycle

Particles Emission (Process 0)

Malleable problem data at T iteration

If the process is a malleable one, get data shared via Flex-MPI

Process Monitoring

Particles Inner Cycle

Particles Removing and I/O (Process 0)

If a Flex-MPI command for process remotion is received, exit the iteration cycle

Main issues:

the problem size varies between one iteration and another.

Process 0 does operations affecting the problema size.

https://github.com/ccmmma/wacommplusplus

Main issues & work in progress with the Flex-MPI team

- Simplified iteration schema:
  - Process 0:
    - Generate new particles
    - Prepare displacements and counts
  - Broadcast the counts to all processes
  - Scatter the particles to processes
  - Compute the particles' outer cycle
  - Gather the particles from processes
  - Process 0:
    - Remove died particles
    - Performs I/O

- Register/Unregister: the size changes at each iteration
- The displacement and the count is changed at each iteration by the process 0

https://github.com/ccmmma/wacommplusplus

Main issues & work in progress with the Flex-MPI team

- We worked tightly with the Flex-MPI team to move from the first malleable WaComM++ prototype to the full featured WaComM++

- WaComM++ can be build with Flex-MPI for production

```
find_library(EMPI_LIBRARY empi HINTS "path_EMPI_lib")
find_library(PAPI_LIBRARY papi HINTS "path_PAPI_lib")
find_library(GLPK_LIBRARY glpk HINTS "path_GLPK_lib")
find_library(ICC_LIBRARY icc HINTS "path_ICC_lib")
```

- Example: compile with OpenMP, and Flex-MPI support:

```
cmake -DUSE_OMP=ON -DUSE_MPI=OFF -DUSE_EMPI=ON -DUSE_CUDA=OFF -DDEBUG=OFF ..
make
mpirun -n np ./wacommplusplus
```

https://github.com/ccmmma/wacommplusplus

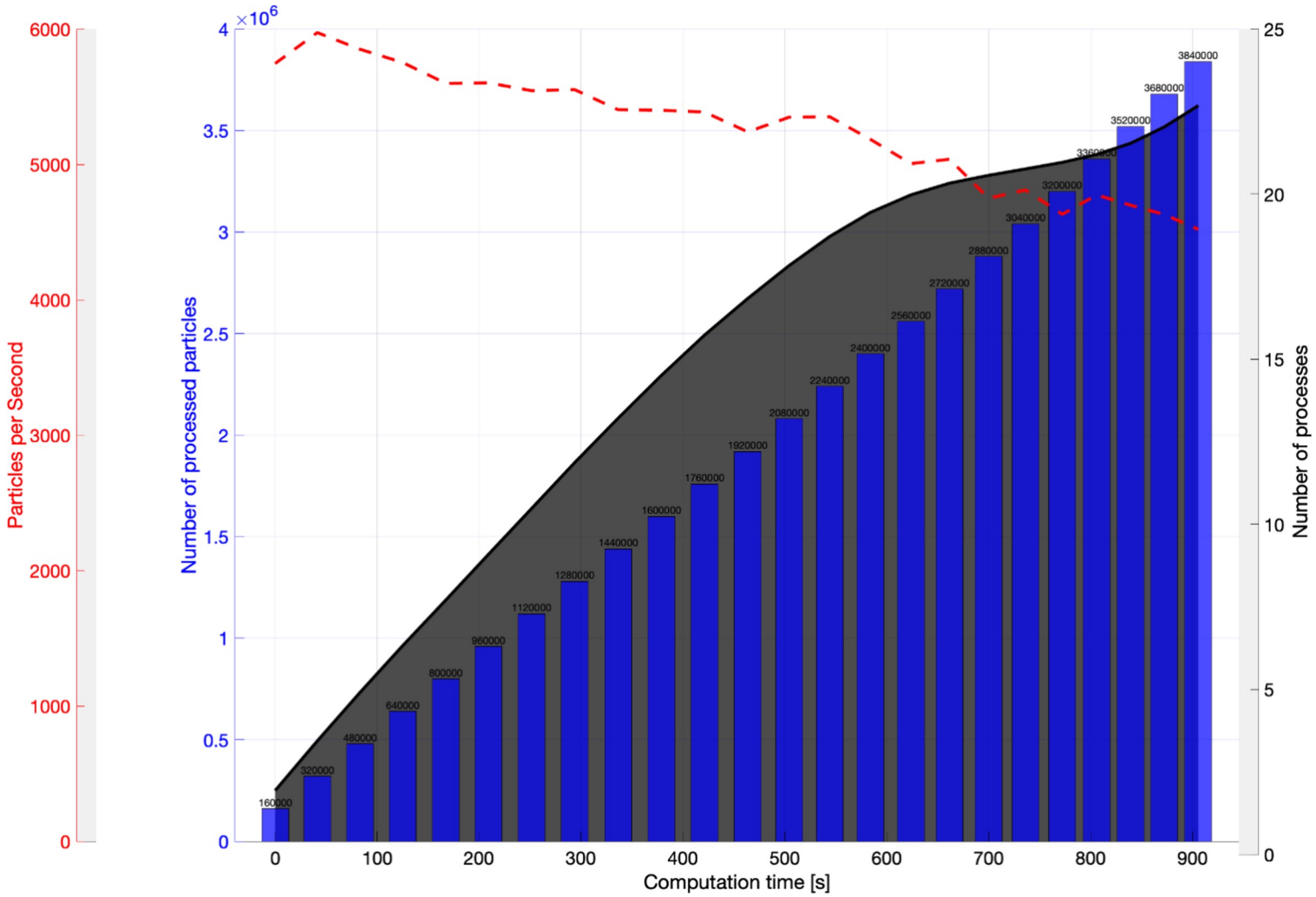
Flex-MPI: increasing processes scenario

- WaComM++ is configured to perform a 24h simulation (24 iterations).
- At each iteration 250k particles are added, finally some of them will not survive (health, borders closures).
- At each iteration the number of processes is increased by 1.

- The metric used for performance evaluation is the number of particles processed per second computed by the process 0 considering the time for particles generation, data distribution, computation and recollection.
- The time needed for I/O is not considered in this case.

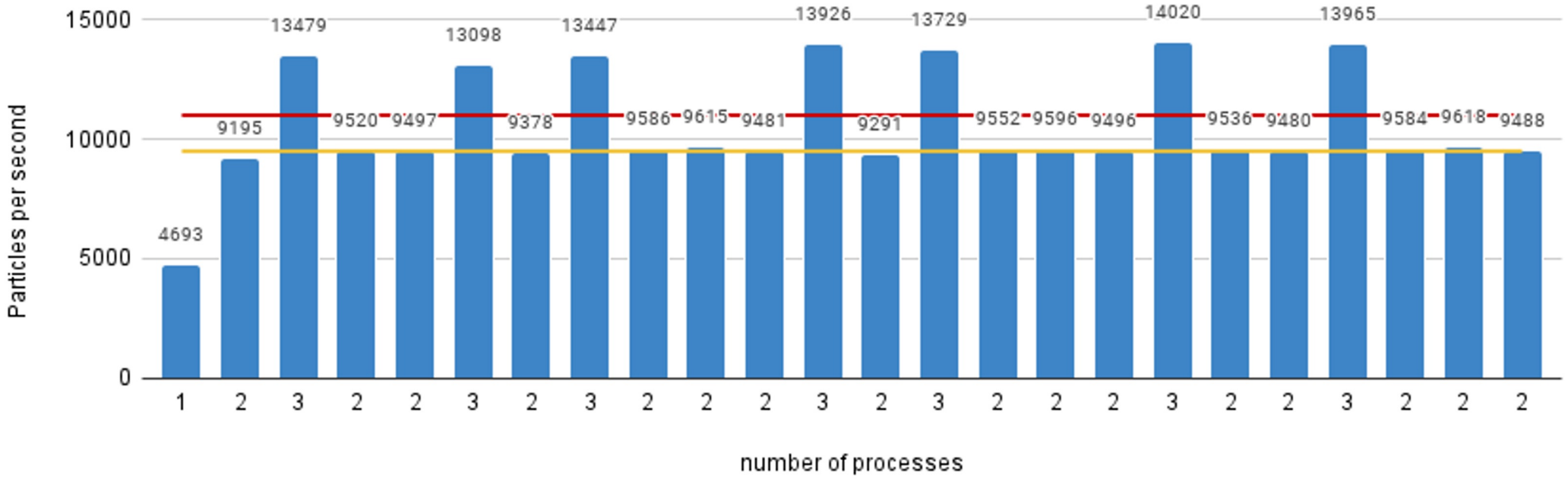
https://github.com/ccmmma/wacommplusplus

Flex-MPI: increasing processes scenario



- Simulation of 24 hours of 250K particles emitted per hour from a single emission point.
- The problem size is distributed on a new spawn process each simulated hour.

Flex-MPI: constant performance scenario

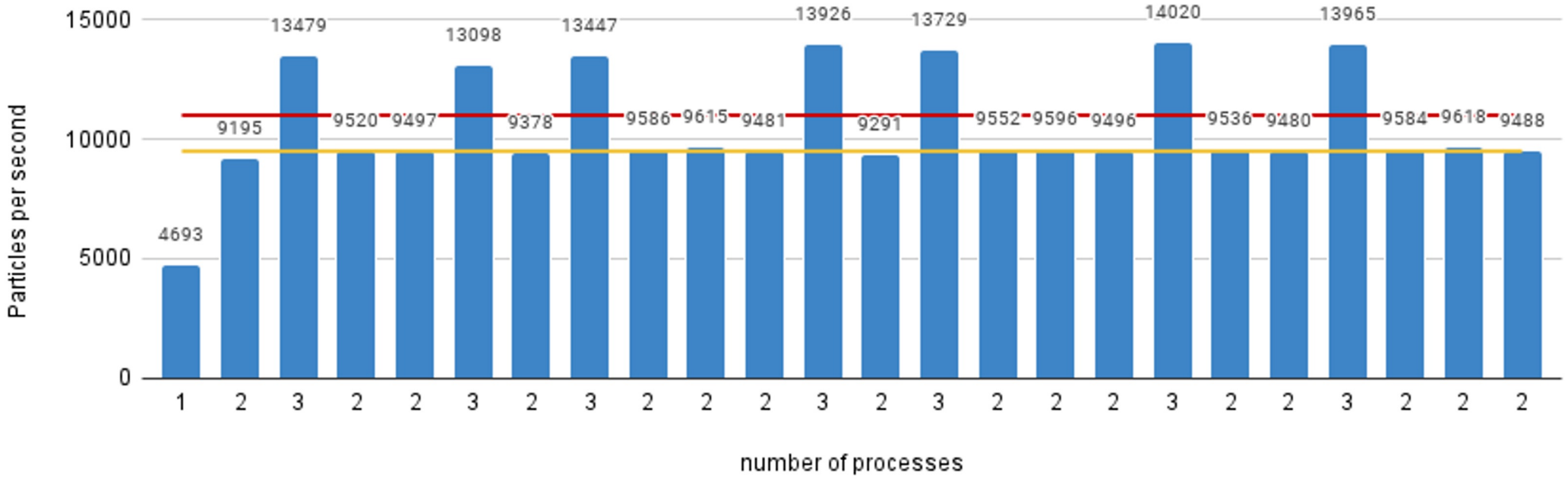


- WaComM++ is configured to perform a 24h simulation (24 iterations).

- At each iteration 250k particles are added, finally some of them will not survive (health, borders closures).

https://github.com/ccmmma/wacommplusplus

Flex-MPI: constant performance scenario



- The target performance metric has been constrained in the range between 9k particles per second up to 11k particles per second.
- The number of processes varies by means of stay in the costrains.
- The time needed for I/O is not considered in this case.

https://github.com/ccmmma/wacommplusplus

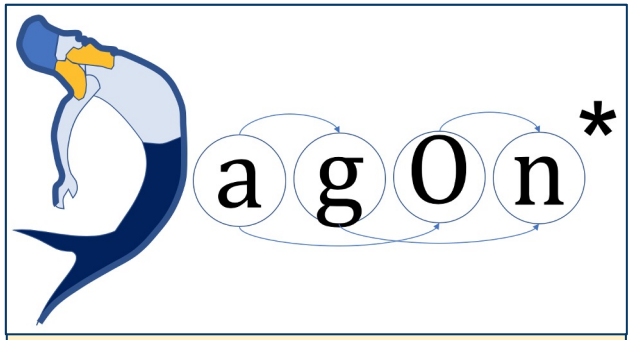
DagOnStar: Yet Another Python-Based Workflow Engine

…that works like a charm for computational environmental science applications…

Direct Acyclic Graphs as parallel jobs on anything

DagOnStar is a production-oriented workflow engine:

- **Integration** in the Python environment.
- **Minimal** footprint for external software components execution.
- **Avoiding** any **centered data management**.
- **Straightforward** definition of tasks:
  - Python scripts.
  - Web interaction.
  - External software components.
  - Parallel patterns.
- **Execution** sites **independence**:
  - Local / scheduler (SLURM).
  - Containers (Docker).
  - Clouds (AWS, OpenStack, DigitalOcean).
- **Similar products (short incomplete list):** Parsl, StreamFlow, …



Named after the Phoenician god-fish *Dagon* known by ancient Greeks as *Triton*.

NB: The **star** symbol * is the wildcard for **anything**.

https://github.com/dagonstar/

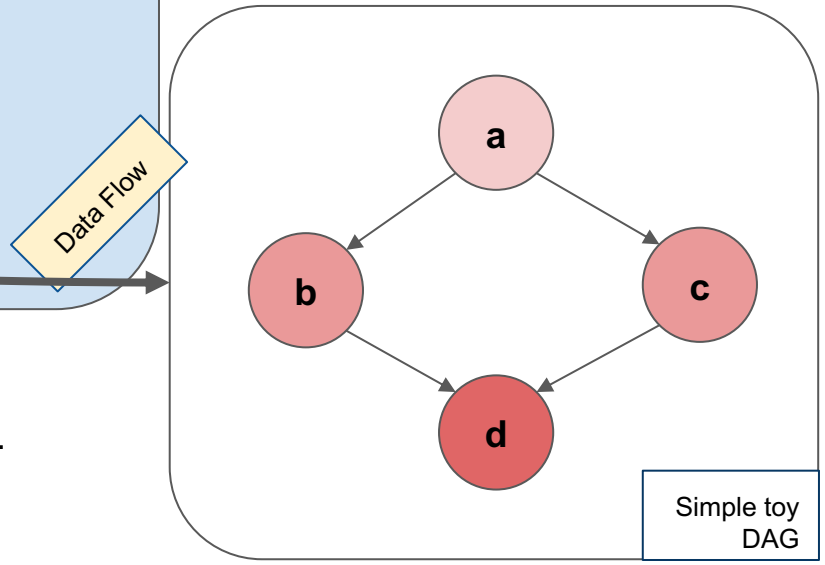
Programming Model

Python Script: "DagOnStar Hello World App"

```
from dagon import Workflow
from dagon.task import TaskType, DagonTask
…
workflow = Workflow("myapp")
workflow.add_task(DagonTask(TaskType.BATCH, "a", "..."))
workflow.add_task(DagonTask(TaskType.BATCH, "b", "workflow:///a"))
workflow.add_task(DagonTask(TaskType.BATCH, "c", "workflow:///a"))
workflow.add_task(DagonTask(TaskType.BATCH, "d", "workflow:///b
workflow:///c"))
workflow.run()
sys.exit(0)
```

Task Flow — Defined by task dependencies.

Data Flow — Defined by data dependencies.

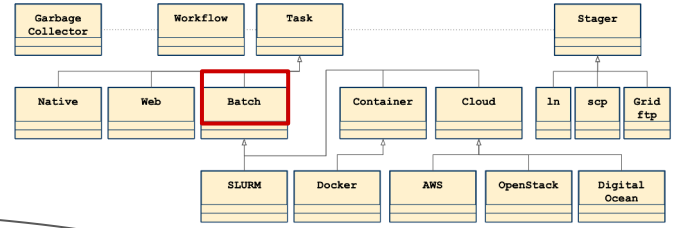


- Dealing with actual data files instead of high-level defined datasets.
- Performing backward data references in order to create dependencies.
- Having more Workflow instances in the same Python application.

DagOnStar has been designed by a computational environmental application friendly programming model.

The workflow:// schema

The **Batch** component takes charge of the management of data dependencies using the **workflow:// schema**.



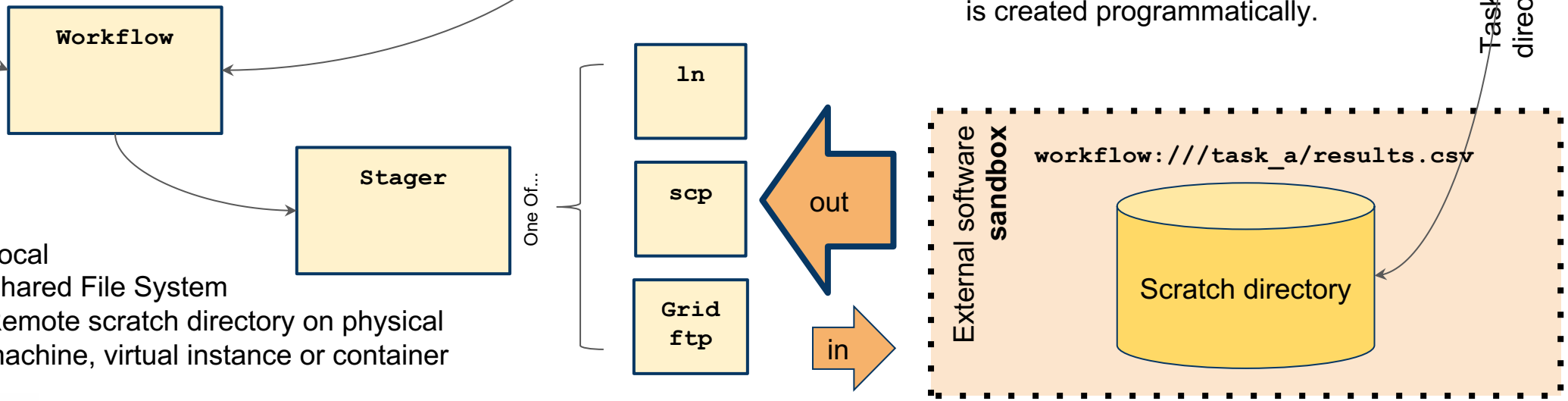
workflow://workflow_unique_name/task_unique_name/

The schema label

The workflow unique name
An UUID could be used
If empty means “current workflow”

The task unique name
Can be dynamically generated by the Python script when the workflow is created programmatically.

Task scratch directory root



- Local
- Shared File System
- Remote scratch directory on physical machine, virtual instance or container

https://github.com/dagonstar/

Conclusions

…everything has a start, sooner or late comes to the end…

Conclusion (1/2)

The Environmental Application is available as:

1. Example in the DagOnStar workflow engine repository:
   https://github.com/DagOnStar/dagonstar
   In both dry run (easy way) and real flavours (if you like playing hard).

2. As ADMIRE use case on the University of Torino HPC cluster (with computationally malleable WaComM++).

3. In routinary production on dedicated HPC resources (HPC-GPU BlackJeans, 650 CPU cores, 1 PB long term storage, http://rcf.uniparthenope.it)

Data Dissemination:

- Technical web portal
- Progressive Web Application
- Opendap Server
- Http
- Web APIs

WaComM++ step by step tutorial is available on the repository:
https://github.com/CCMMMA/wacommplusplus

Conclusion (2/2)

- It runs basically uninterrupted since the first prototype.
- *Almost* failsafe: time to recovery after a catastrophic event (full storage loss), less than 72 hours.
- Runs as DagOnStar application
- The WaComM++ transport and diffusion model has already used for different applications.