Generalitat de Catalunya
Departament de Recerca
i Universitats

GOBIERNO
DE ESPAÑA
MINISTERIO
DE CIENCIA
E INNOVACIÓN

UPC
UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

UNIÓN EUROPEA
Fondo Europeo de Desarrollo Regional

Programming complex workflows with PyCOMPSs

Rosa M Badia

CEEC webinar, December 16 2024

CEEC

Centre of Excellence in Exascale CFD

EuroHPC JU systems

	Status	Country	Peak performance	Architecture
Pre-Exascale				
LUMI	Operational	Finland	539.13 petaflops	64-core AMD EPYC™ CPUs + AMD Instinct™ GPU
Leonardo	Operational	Italy	315.74 petaflops	Intel Ice-Lake, Intel Sapphire Rapids + DIA Ampere
MareNostrum 5	Operational	Spain		Sapphire Rapids, NVIDIA Hopper, A Grace, Intel Emeralds, Intel
Petascale				
Meluxina	Op			MD EPYC + NVIDIA Ampere A100
Vega	Oper		10.05 petaflops	AMD Epyc 7H12 + Nvidia A100
Karolina	Opera	zech Republic	12.91 petaflops	AMD + Nvidia A100
Discoverer	Operational	Bulgaria	5.94 petaflops	AMD EPYC
Deucalion	Operational	Portugal	5.01 petaflops	A64FX, AMD EPYC, Nvidia Ampere

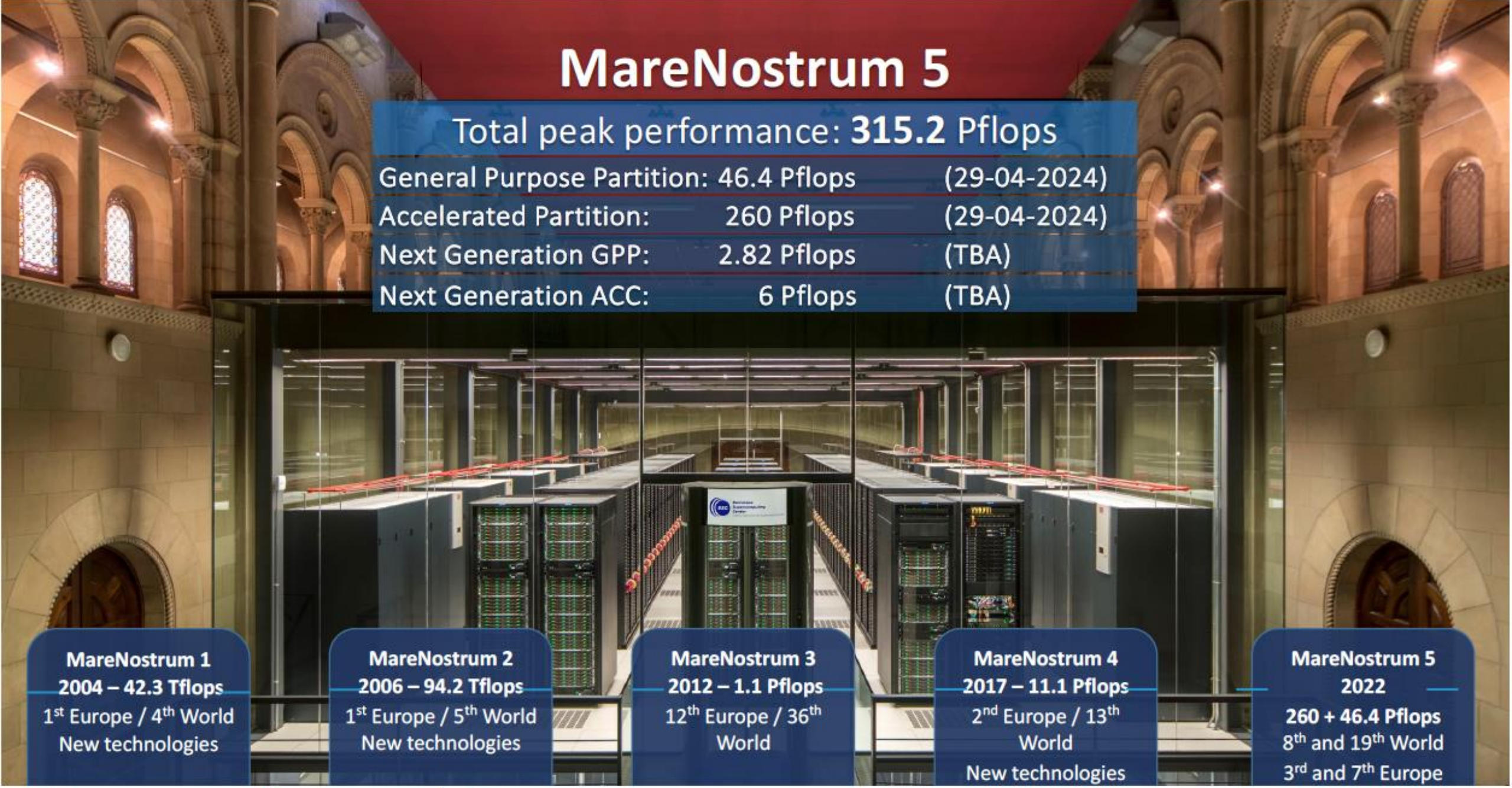
JUPITER, First European Exascale Supercomputer to be installed in Jülich, Germany

Centro Nacional de Supercomputación

https://eurohpc-ju.europa.eu/about/our-supercomputers_en

MareNostrum 5

Total peak performance: **315.2** Pflops		
General Purpose Partition:	46.4 Pflops	(29-04-2024)
Accelerated Partition:	260 Pflops	(29-04-2024)
Next Generation GPP:	2.82 Pflops	(TBA)
Next Generation ACC:	6 Pflops	(TBA)



MareNostrum 1
2004 – 42.3 Tflops
1st Europe / 4th World
New technologies

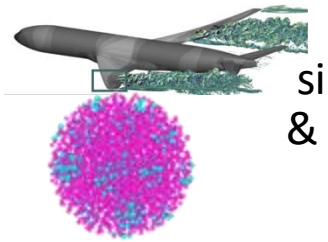
MareNostrum 2
2006 – 94.2 Tflops
1st Europe / 5th World
New technologies

MareNostrum 3
2012 – 1.1 Pflops
12th Europe / 36th World

MareNostrum 4
2017 – 11.1 Pflops
2nd Europe / 13th World
New technologies

MareNostrum 5
2022
260 + 46.4 Pflops
8th and 19th World
3rd and 7th Europe

Complex problems for complex computing infrastructures

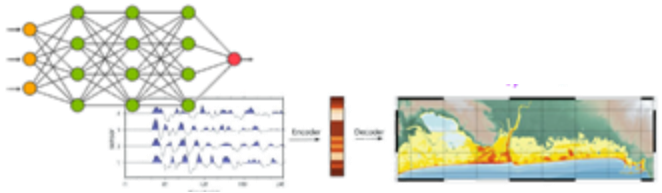


HPC simulation & modeling

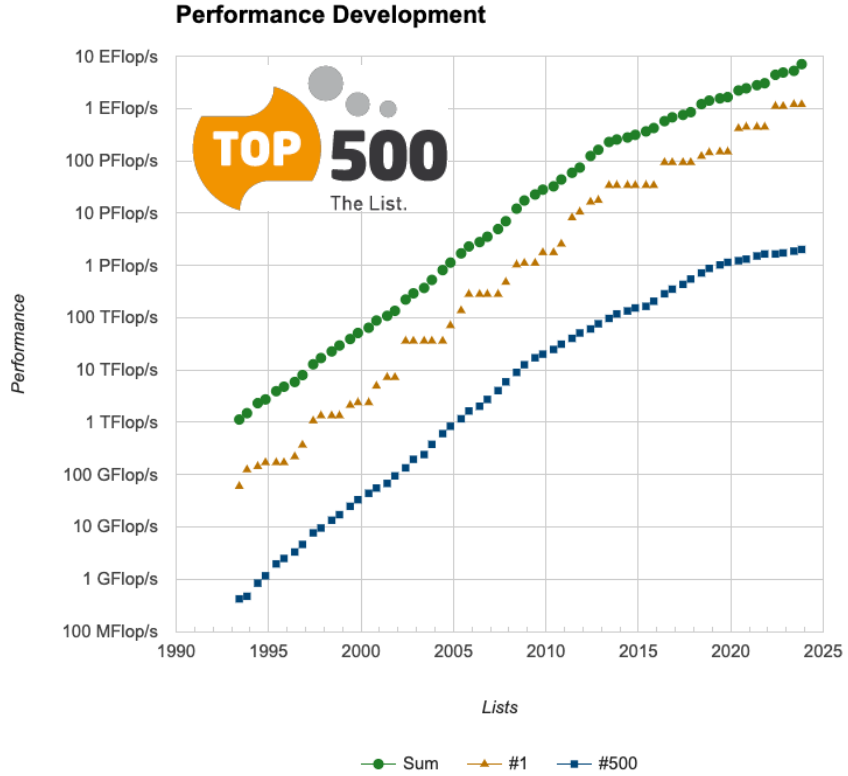
Data analytics

COMPLEX PROBLEMS

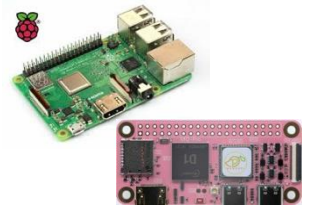
Artificial intelligence



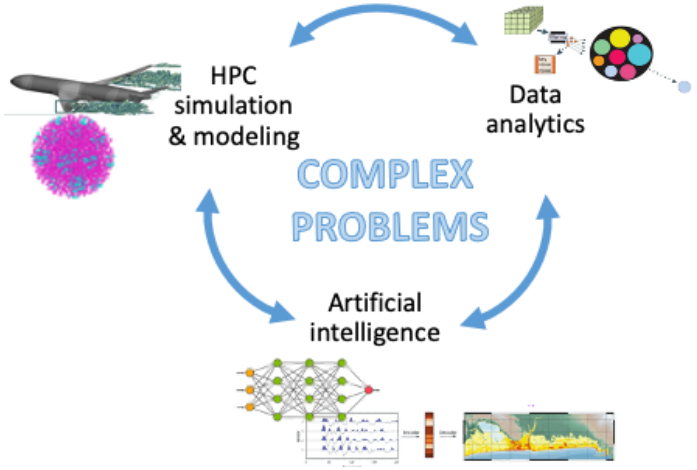
Barcelona Supercomputing Center
Centro Nacional de Supercomputación



Digital continuum

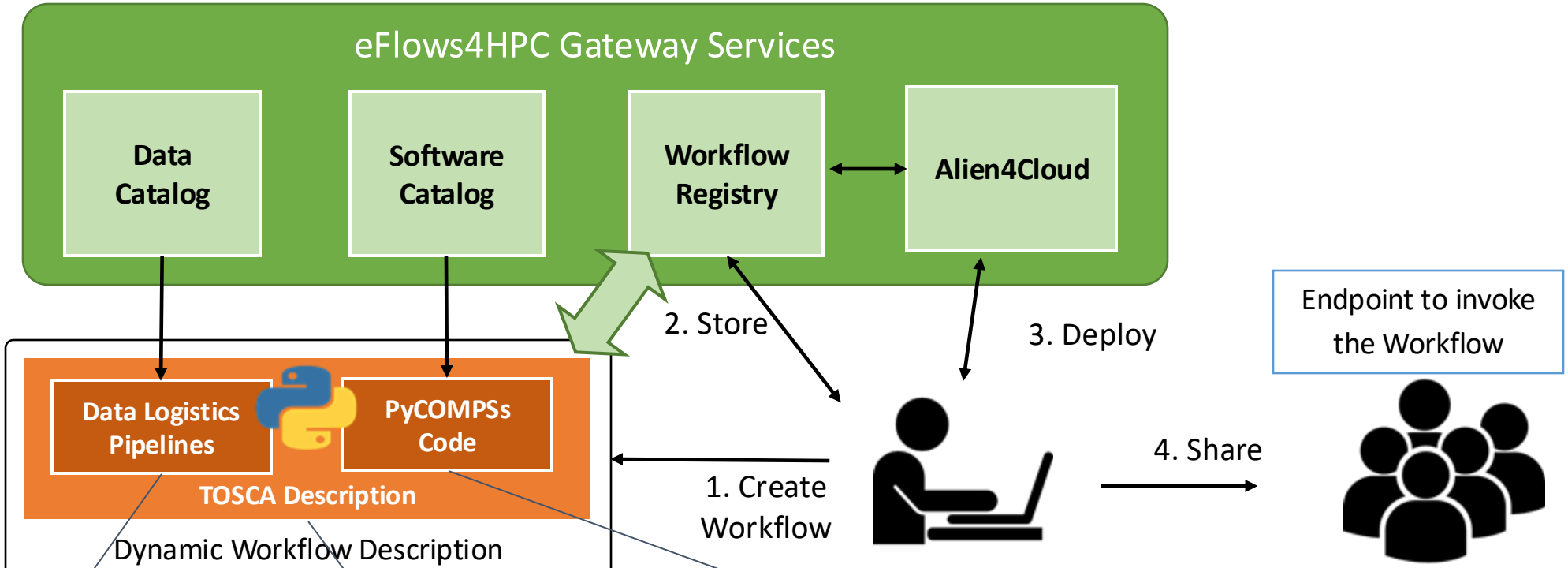


Workflow lifecyle challenges



- Workflow development
  - Different programming models and environments
- Workflow deployment
  - Can we make it easier to new HPC users?
- Workflow operation
  - Go beyond static workflows
  - Not only computational aspects, data management as well

HPCWaaS: Workflow lifecycle overview



Description of data movements as Python functions. Input/output datasets described at Data Catalog

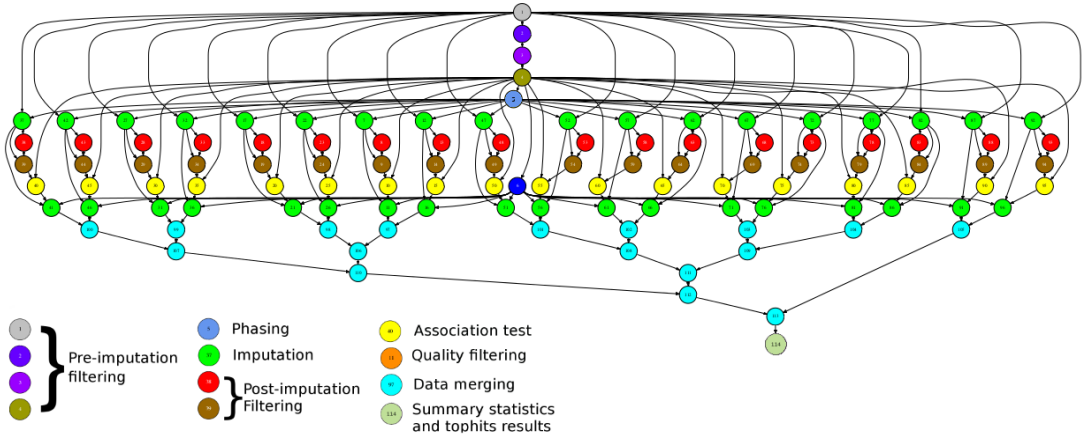
Computational Workflow as a simple Python script. Invocation of software described in the Software Catalog

Topology of the components involved in the workflow lifecycle and their relationship.

Workflows' development with PyCOMPSs

- Sequential programming, parallel execution
  - General purpose programming language + annotations/hints
- Task-based parallelization
  - Automatic generation of task graph
  - Coarse grain tasks: methods and web services
  - Sequential and parallel tasks
- Offers a shared memory illusion in a distributed system
  - Can address larger dataset than storage space
- Agnostic of computing platform
  - Clusters, clouds and cluster containers

```
@task(c=INOUT)
def multiply(a, b, c):
    c += a*b
```



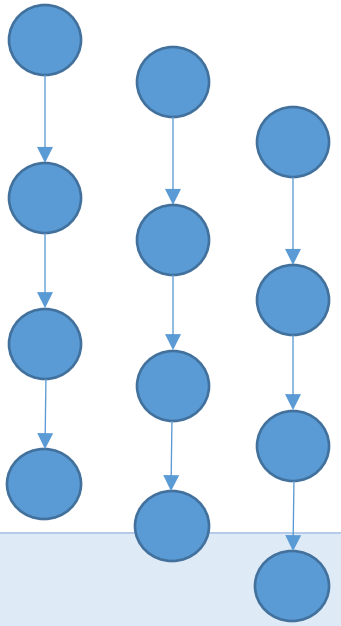
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PyCOMPSs syntax

- Use of **decorators** to annotate tasks and to indicate arguments directionality
- Small API for data synchronization

Tasks definition

```
@task(c=INOUT)
def multiply(a, b, c):
    c += a*b
```



Main Program

```
initialize_variables()
startMulTime = time.time()
for i in range(MSIZE):
   for j in range(MSIZE):
      for k in range(MSIZE):
         multiply (A[i][k], B[k][j], C[i][j])
compss_barrier()
mulTime = time.time() - startMulTime
```

Synchronization

- Main program and tasks do not share the same memory spaces
- The synchronization compss_wait_on waits for tasks generating the parameter to be finished and moves the data from the remote node to the node where the main program is executed:

```
a = compute (b)
#compute is a task, here we can not check the value of a
...
a = compss_wait_on (a)
#here we can check the value of a
if a:
    ...
```

```
startMulTime = time.time()
for i in range(SIZE):
    compute (A[i], B[i])
compss_barrier()
mulTime = time.time() - startMulTime
```

Tasks' constraints

- Constraints enable to define HW or SW features required to execute a task
  - Runtime performs the match-making between the task and the computing nodes
  - Support for multi-core tasks and for tasks with memory constraints
  - **Support for heterogeneity on the devices in the platform**

```
@constraint (MemorySize=6.0, ProcessorPerformance="5000", ComputingUnits="8")
@task (c=INOUT)
def myfunc(a, b, c):
    ...
```

```
@constraint (MemorySize=1.0, ProcessorType ="ARM", )
@task (c=INOUT)
def myfunc_other(a, b, c):
    ...
```

Linking with other programming mode

- A task can be more than a sequential function
  - A task in PyCOMPSs can be sequential, multicore or multi-node
  - External binary invocation: wrapper function generated automatically
  - Supports for alternative programming models: MPI and OmpSs
- Additional decorators:
  - @binary(binary="app.bin")
  - @mpi(binary="mpiApp.bin", runner="mpirun", processes=8)
  - @ompss(binary="ompssApp.bin")
- Can be combined with the @constraint and @implement decorators

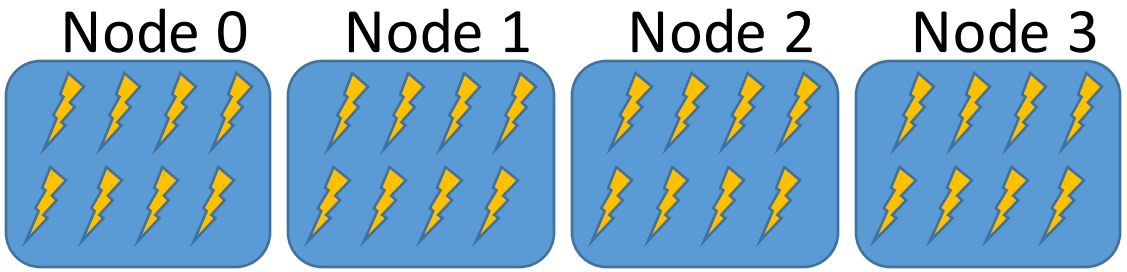
```
@binary(binary="app.bin", workingDir="/myApp")
@task()
def func(l):
    pass
```

Support for MPI tasks

- Resource manager aware of multi-node tasks

```
@mpi (binary=”mySimulator”, runner="mpirun", processes= ”32”, processes_per_node=8)
@task (returns=int, stdOutFile=FILE_OUT_STDOUT, stdErrFile=FILE_OUT_STDERR)
def nems(stdOutFile, stdErrFile):
    pass
```

Launches MPI execution with
32 processes
8 processes per node



MPMD applications

- The @mpmd_mpi decorator can be used to define Multiple Program Multiple Data (MPMD) MPI tasks

```
@mpmd_mpi(runner="mpirun", working_dir = {{working_dir_exe}},
          programs=[{binary="fesom.x", processes = "$FESOM_PROCS" },
                    {binary="oifs", args="-v ecmwf -e awi3", processes = "$OIFS_PROCS" },
                    {binary="rnfma", processes = "$RNFMA_PROCS"}])
@task(log_file={Type:FILE_OUT, StdIOStream:STDOUT}, working_dir_exe=DIRECTORY_INOUT)
def esm_simulation(log_file, working_dir_exe):
   pass
```

- As a result of the @mpmd_mpi annotation, the following commands will be generated:

```
> cd working_dir_exe; mpirun -n $FESOM_PROCS fesom.x : \
    -n $OIFS_PROCS oifs -v ecmwf -e awi3 : -n $RNFMA_PROCS rnfma
```

Tasks in container images

- Goal: enable tasks embedded in container images
  - @container decorator can be used together with the task annotation
  - Also support for user-defined tasks

```
@container(engine="DOCKER", image="ubuntu")
@binary(binary="ls")
@task()
def task_binary_empty():
    pass
```

```
@container(engine="DOCKER", image="compss/compss")
@task(returns=1, num=IN, in_str=IN, fin=FILE_IN)
def task_python_return_str(num, in_str, fin):
    print("Hello from Task Python RETURN")
    print("- Arg 1: num -- " + str(num))
    print("- Arg 1: str -- " + str(in_str))
    print("- Arg 1: fin -- " + str(fin))
    return "Hello"
```

Support for data streams

- Interface to support streaming data in tasks
- Task-flow and data-flow tasks live together in PyCOMPSs/COMPSs workflows
- Data-flow tasks persist while streams are not closed
  - Parameters can be one/multiple streams and non-streamed
- Runtime implementation based on Kafka

```
@task(fds=STREAM_OUT)
def sensor(fds):
        ...
    while not end():
        data = get_data_from_sensor()
        f.write(data)
    fds.close()
```

```
@task(fds_sensor=STREAM_IN, filtered=OUT)
def filter(fds_sensor, filtered):
        ...
    while not fds_sensor.is_closed():
        get_and_filter(fds_sensor, filtered)
```

Failure management

- Interface than enables the programmer to give hints about failure management

```
@task(file_path=FILE_INOUT, on_failure='CANCEL_SUCCESSORS',
time_out='$task_timeout')
def task(file_path):
    ...
    if cond :
        raise Exception()
```

- Options: RETRY, CANCEL_SUCCESSORS, FAIL, IGNORE
- Implications on file management:
  - i.e, on IGNORE, output files are generated empty
- **Possibility of ignoring part of the execution of the workflow, for example if a task fails in an unstable device**
- **Opens the possibility of dynamic workflow behaviour depending on the actual outcome of the tasks**

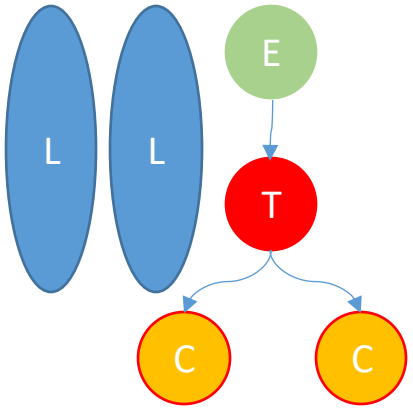
Timeouts and exceptions

- Timeouts can be defined for a task

```
@task(file_path=FILE_IN, time_out=200)
def time_out_task (file_path):
    ...
```

- Tasks can raise exceptions

```
@task(file_path=FILE_INOUT)
def comp_task(file_path):
    ...
    raise COMPSsException("Exception
raised")
```

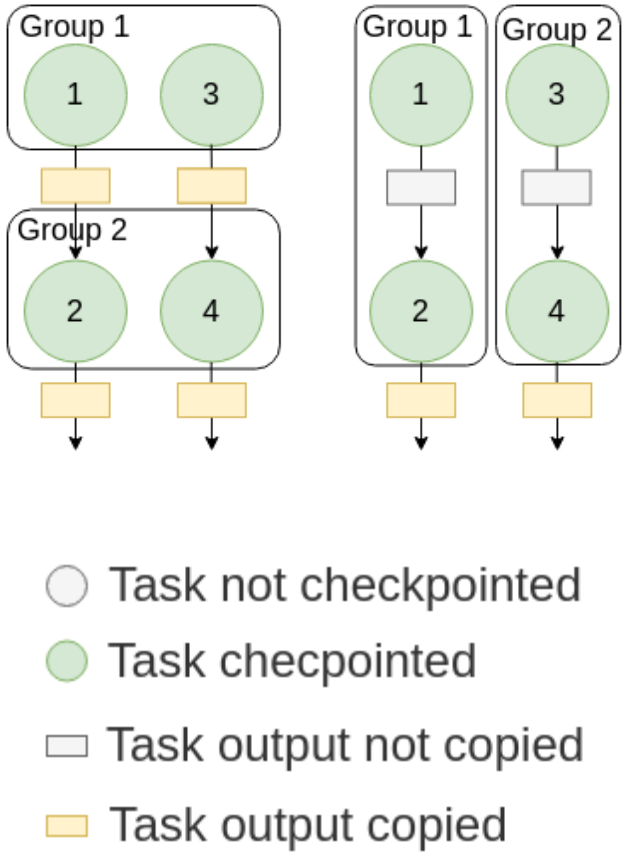


- Combined with groups of tasks enables to cancel the group of tasks on the occurrence of an exception

```
def test_cancellation(file_name):
    try:
        with TaskGroup('failedGroup'):
            long_task(file_name)
            long_task(file_name)
            executed_task(file_name)
            comp_task(file_name)
            cancelledTask(FILE_NAME)
            cancelledTask(FILE_NAME)

    except COMPSsException:
        print("COMPSsException caught")
        write_two(file_name)
```

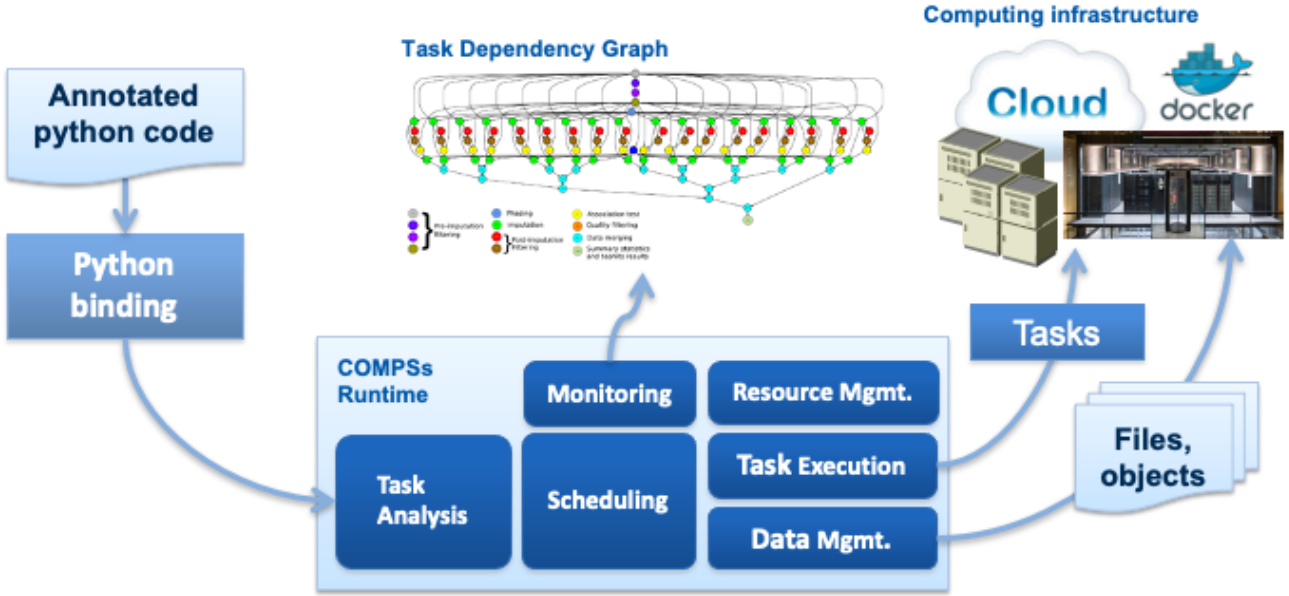
Checkpointing

- Allows the workflow re-execution avoiding the re-execution of finished tasks
- Asynchronous but may have some overhead
  - Save tasks results in a persistent storage
  - Trade-off between performance and time to recover
  - Establishing the right checkpoint granularity is important
- 3 mechanisms for automatic checkpointing
  - **Time:** periodically, COMPSs saves the last version produced for every value
  - **Finished tasks** : after the completion of X tasks, COMPSs saves the last version produced for every value
  - **Instantiation task groups**: Defines groups of tasks, COMPSs saves those data versions that are final results for the group
- Indicated by the developer using a provided API call `compss_snapshot()`
- No checkpoint inside the task: drawback for very large tasks.
  - Possible integration with internal checkpointing



PyCOMPSs features and runtime

- PyCOMPSs/COMPSs applications executed in distributed mode following the master-worker paradigm
  - Description of computational infrastructure in an XML file
- Sequential execution starts in master node and tasks are offloaded to worker nodes
- In clusters, whole COMPSs application deployed as a job allocation
- All data scheduling decisions and data transfers are performed by the runtime
- All data scheduling decisions and data transfers are performed by the runtime



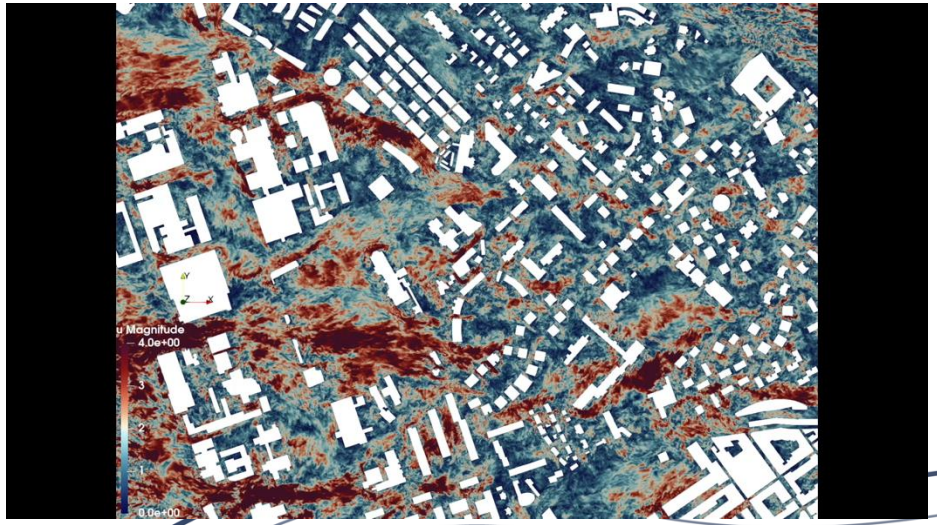
Workflow examples

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

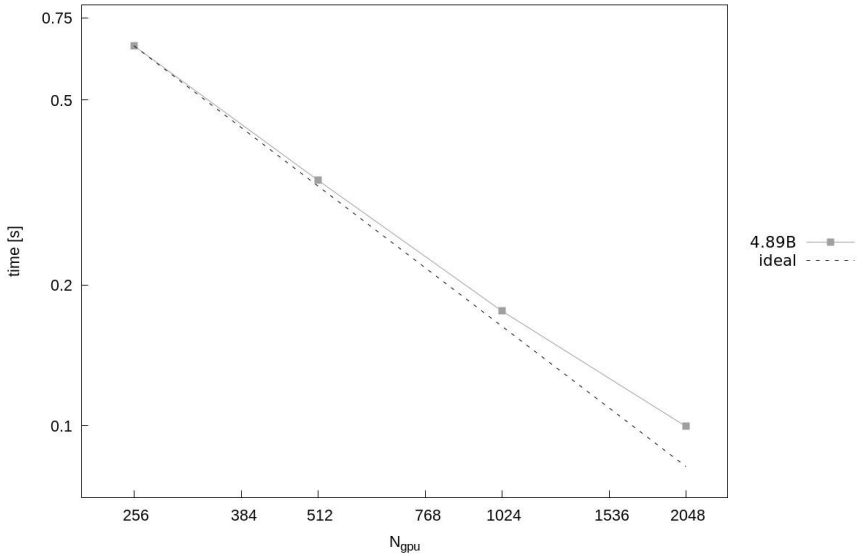
SOD2D: Spectral high-Order coDe 2 solve partial Differential equations*

- Language: Fortran
- GPU port path: OpenACC
- Required libs: HDF5, MPI
- Compressible and incompressible flows
- Git repo: https://gitlab.com/bsc_sod2d/sod2d_gitlab/
- … and btw, the code is 3D!

SOD2D: **S**pectral high-**O**rder co**D**e 2 solve partial **D**ifferential equations



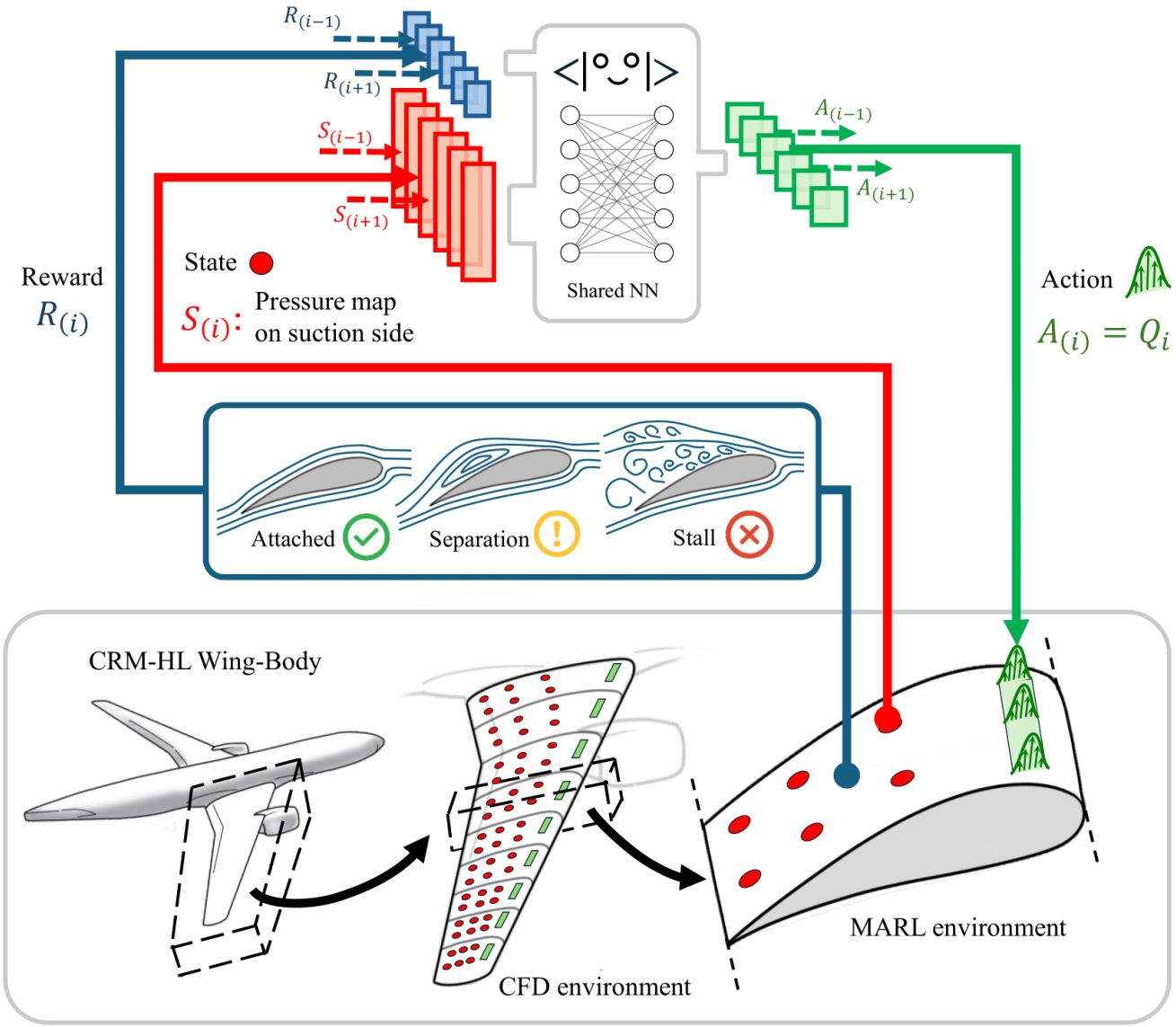
wmLES of the campus Nord of UPC, using 64 H100 and 500M DoF



Scalability of SOD2D in MN5

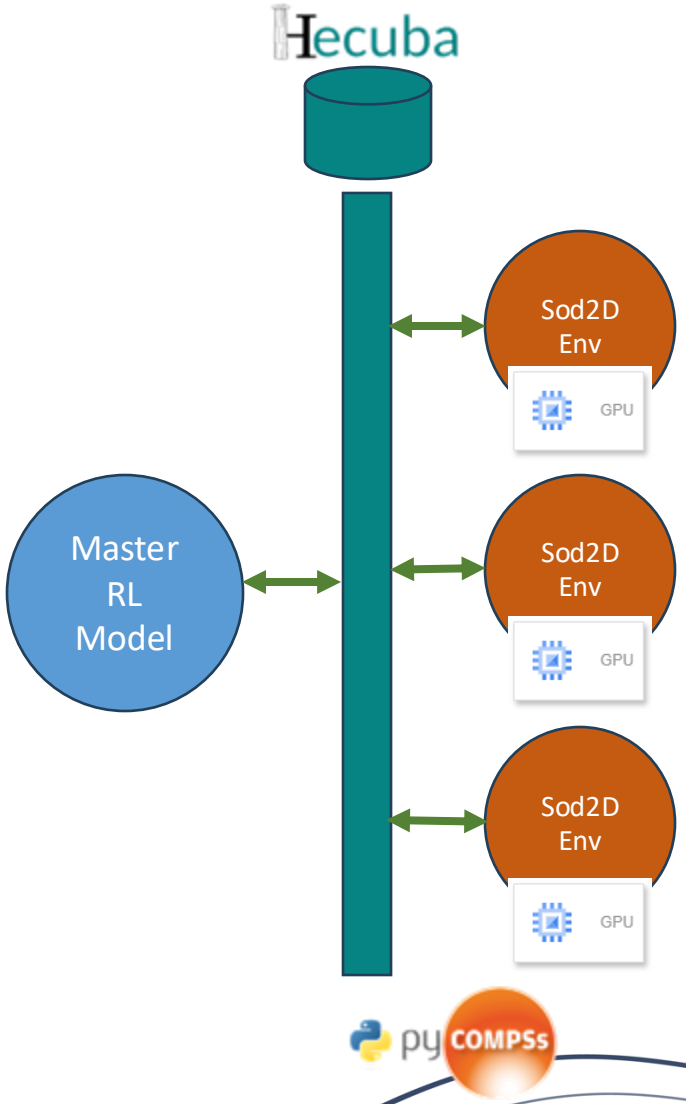
Paper: *https://doi.org/10.1016/j.cpc.2023.109067*
**Oriol Lehmkuhl (BSC) et al.*

The CEEC challenge



- Research in automatic ways to control the aerodynamic performance of aircrafts under dynamic extreme events
- Sod2D - RL model coupling using a multi-environment approach for accelerated training
  - Example: using 4 environments executed in 16 GPUs each
- Each environment sends its state and reward into the database
- The RL model reads state and reward and returns actions for each environment into the database
- Each environment reads the corresponding actions and advances the flow solution in time

Integration with PyCOMPSs and Hecuba




```
@constraint(processors=[
        {"processorType": "CPU", "computingUnits": "1"},
        {"processorType": "GPU", "computingUnits": "1"}
    ])
@mpi(
    binary="sod2d",
    runner="mpirun",
    processes=4,
    flags="-x SSOB={{db_address}} -x COMPSS_BINDED_GPUS",
    args="--restart_step={{restart_step}}
          --f_action={{f_action}}
          --t_episode={{t_episode}}
          --t_begin_control={{t_begin_control}}
          --tag={{tag}}"
)
@task()
def compss_sod2d(db_address, restart_step, faction, t_episode, t_begin_control, tag):
    pass
```

Integration with PyCOMPSs and Hecuba

Before: Redis

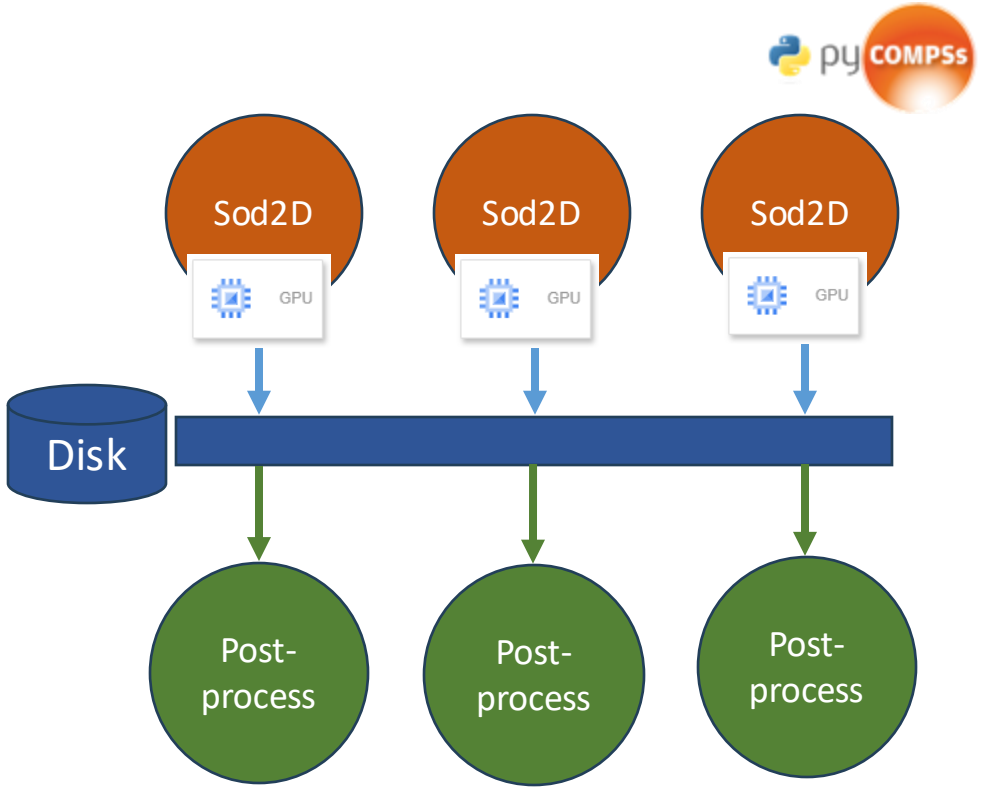
```
subroutine read_action(client, key)
   type(client_type), intent(inout) :: client
   character(len=*), intent(in) :: key ! actions name to read from database
   integer, parameter :: interval = 100 ! polling interval in milliseconds
   integer, parameter :: tries = 10000 ! huge(1) ! infinite number of polling tries
   logical :: exists ! receives whether the tensor exists
   logical :: is_error
   integer :: found, error

   ! wait (poll) until the actions array is found in the DB, then read, then delete
   if (mpi_rank .eq. 0) then
      found = client%poll_tensor(trim(adjustl(key)), interval, tries, exists)
      ! wait indefinitely for new actions to appear
      is_error = client%SR_error_parser(found)
      if (found /= 0) stop 'Error in SmartRedis read_action. Actions array not found.'
      error = client%unpack_tensor(trim(adjustl(key)), action_global, shape(action_global))
      is_error = client%SR_error_parser(error)
      if (error /= 0) stop 'Error in SmartRedis read_action. Tensor could not be unpacked.'
      error = client%delete_tensor(trim(adjustl(key)))
      is_error = client%SR_error_parser(error)
      if (error /= 0) stop 'Error in SmartRedis read_action. Tensor could not be deleted.'
   end if

   ! broadcast rank 0 global action array to all processes
   call mpi_bcast(action_global, action_global_size, mpi_datatype_real, 0, app_comm, error)
   !$acc update device(action_global(:))
end subroutine read_action
```

Hecuba hides the synchronization

Now: Hecuba

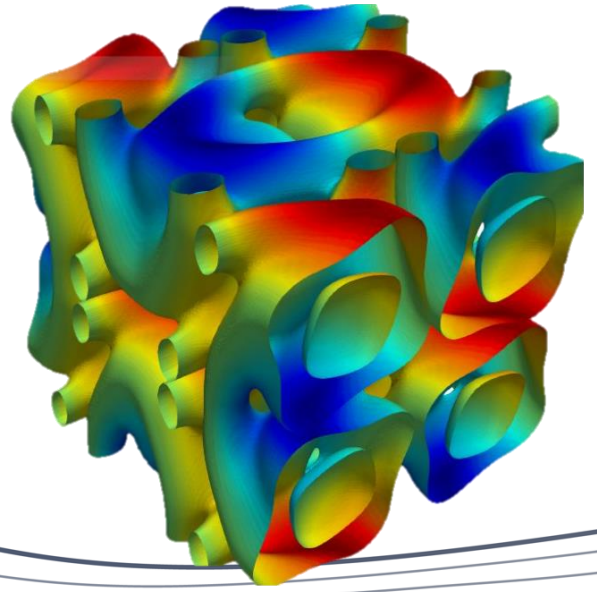
```
subroutine read_action_hecuba(client, key)
   character(len=*), intent(in) :: key ! actions name to read from database

   if (mpi_rank .eg. 0) then
      call hecuba_read_action(trim(adjustl(key)), action_global)
   end if

   ! broadcast rank 8 global action array to all processes
   call mpi_bcast(action_global, action _global_size, mpi_datatype_real, 0,
app_comm, error)
   !$acc update device(action_global(:))
end subroutine read_action_hecuba
```

CEEC Workflows: in-situ postprocessing of CFD simulations and visualization



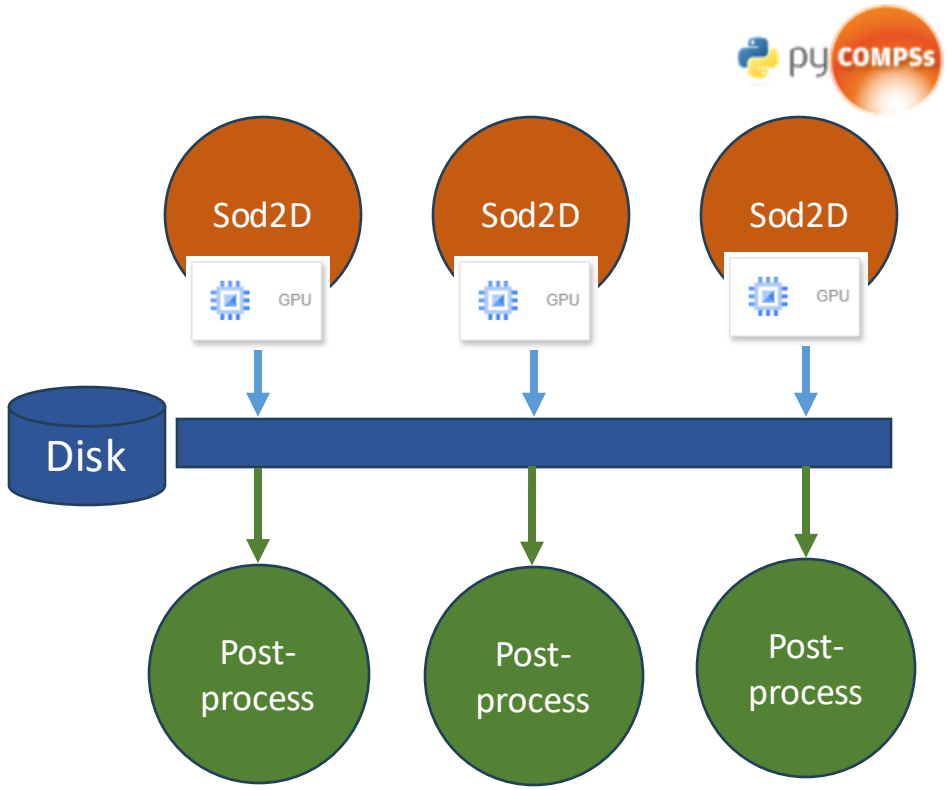
Q isocontours of a Taylor Green Vortex

- This postprocess includes:
  - Compute velocity gradients (with Gauss-Legendre-Lobatto quadrature)
  - Compute its second invariant (Q criterion)
  - Do isocontours of Q
  - Interpolate the velocity to the isocontours.

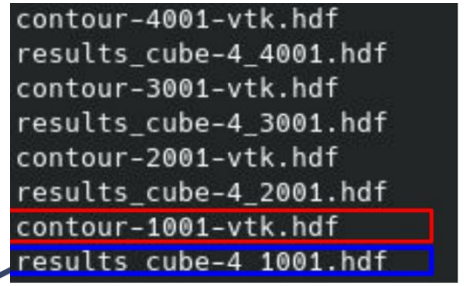


VIDEO

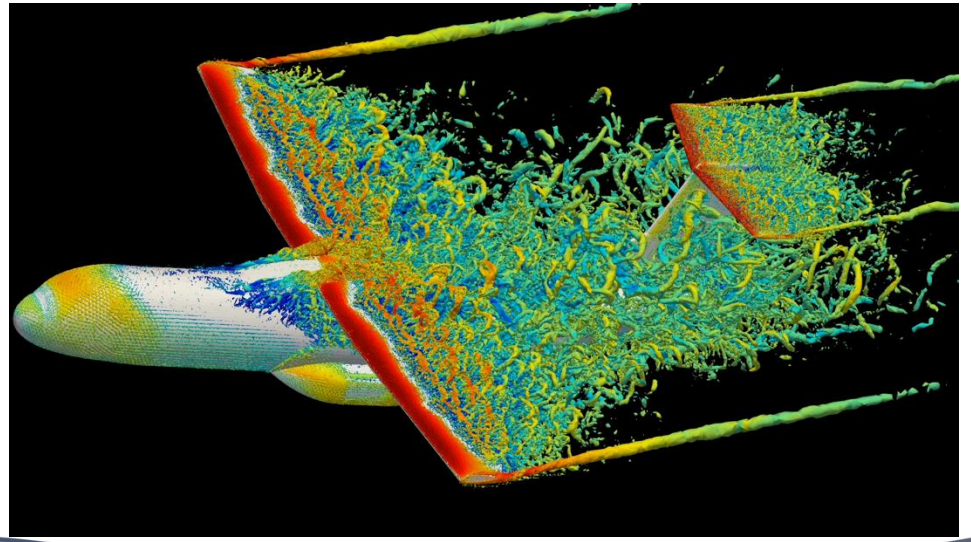
CEEC Workflows: in-situ postprocessing of CFD simulations and visualization



- Run CFD simulations using SOD2D on the GPU
- Uses stream interface of PyCOMPSs
- Every time a results file is saved, PyCOMPSs launches a task that postprocesses the results file:
  - Compute gradients of velocity
  - Q criterion isocontours
  - Extract slices of data
  - Plot over line



Postprocess file
Generated by SOD2D



wmLES simulation of a new concept Aircaft from Airbus, using 32 H100 and 200M DoF. Simulations carried out in the CDTI PTA 2023 (CETACEO) under the collaboration agreement between BSC and Airbus

CEEC Workflows: in-situ postprocessing of CFD simulations and visualization

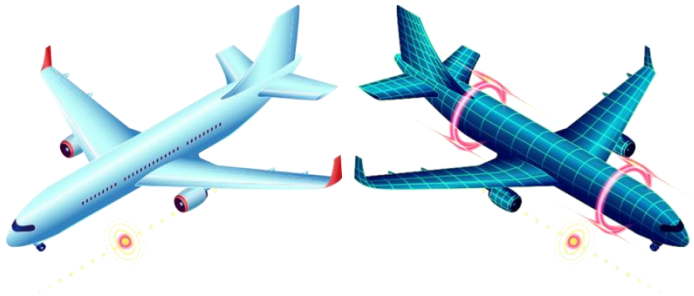

```
@constraint(processors=[{'processorType':'CPU', 'computingUnits':'$SOD2S_CORES_PER_PROC'},
                        {'processorType':'GPU', 'computingUnits':'1'}])
@mpi(runner="mpirun", flags="$SOD2D_MPI_FLAGS", binary="./sod2d", processes="$SOD2D_PROCS",
                    processes_per_node="$SOD2D_PPN", args="{{configuration}} {{fds}}",
          working_dir="{{working_dir}}")
@task(fds=STREAM_OUT)
def sod2d(fds, configuration, working_dir):
    pass
```

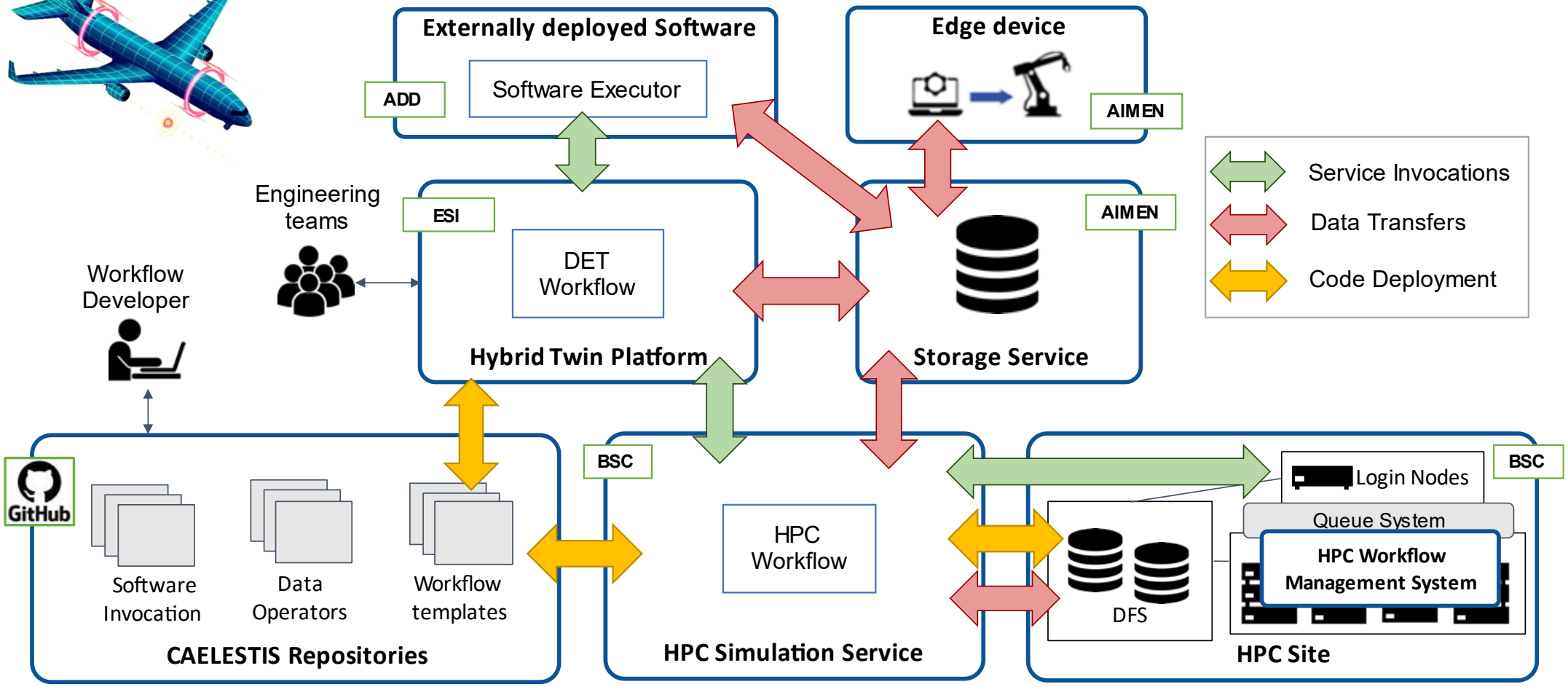
```
@constraint(computing_units="$VIZ_CORES_PER_PROC")
@mpi(runner="mpirun", binary="python", processes="$VIZ_PROCS", processes_per_node="$VIZ_PPN",
          args="{{script}} {{instant}}", working_dir="{{viz_dir}}")
@task()
def visualization(viz_dir, script, instant):
    pass
```

```
fds = FileDistroStream(base_dir=results_folder)
#run simulation
sod2d(fds, configuration, sod2d_workingdir)
while not fds.is_closed():
    new_files = fds.poll()
    #run a simulation for each created file
    for filename in new_files:
        visualization(viz_workingdir, filename)
```

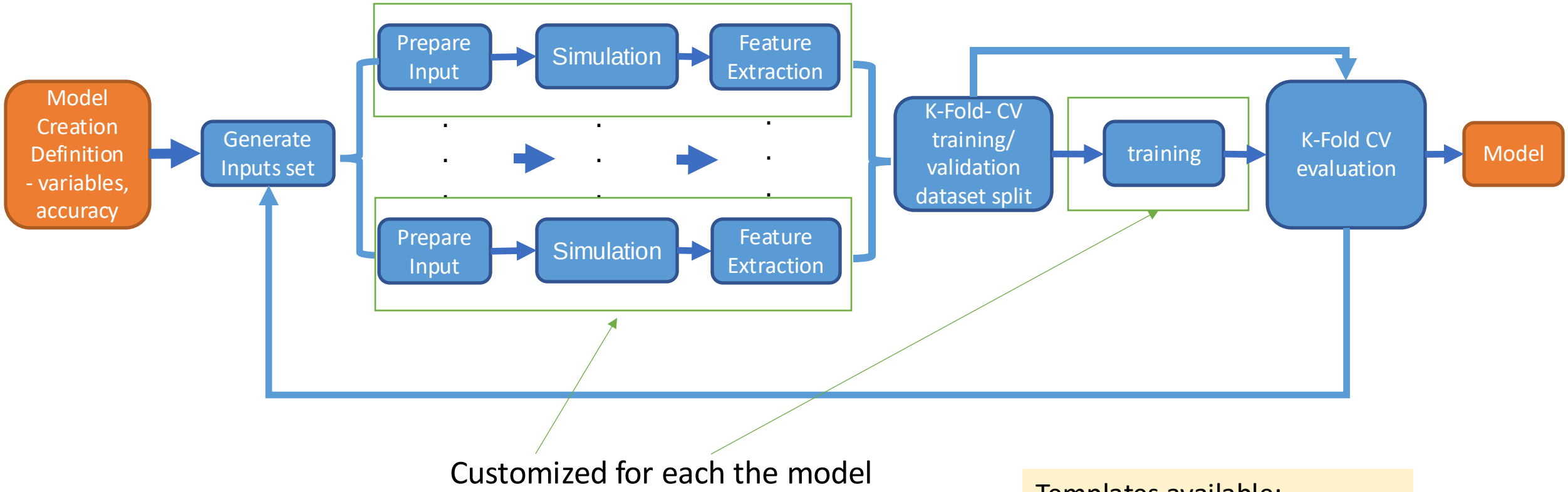

28

CAELESTIS Simulation Ecosystem Architecture





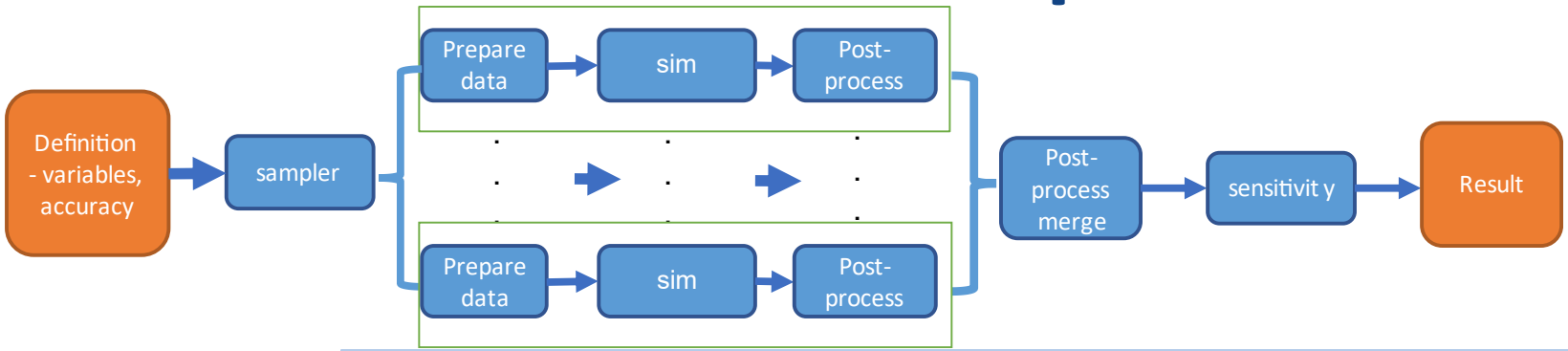
Workflow templates



Templates available:

- Sensitivity Analysis
- Uncertainty Quantification
- Model Training
- Monte Carlo

Workflow templates: main program



workflow.py

```
def execution(execution_folder, data_folder, phases, inputs, outputs, parameters):
    sample_set, problemDef = phase.run(phases.get("sampler"), inputs, outputs, parameters,  data_folder, locals())
    sample_set = compss_wait_on(sample_set)
    problemDef= compss_wait_on(problemDef)
    original_name_sim = parameters.get("original_name_sim")
    results_folder = execution_folder + "/results/"
    if not os.path.isdir(results_folder):
        os.makedirs(results_folder)
    write_file(results_folder, sample_set, "xFile.npy")
    y = []
    for i in range(sample_set.shape[0]):
        values = sample_set[i, :]
        name_sim = original_name_sim + "-s" + str(i)
        simulation_wdir = execution_folder + "/SIMULATIONS/" + name_sim + "/"
        prepare_out = phase.run(phases.get("prepare_data"), inputs, outputs, parameters, data_folder, locals())
        sim_out=phase.run(phases.get("sim"), inputs, outputs, parameters, data_folder, locals(), out=prepare_out)
        new_y = phase.run(phases.get("post_process"), inputs, outputs, parameters, data_folder, locals(), out=sim_out)
        y.append(new_y)
    out_post=phase.run(phases.get("post_process_merge"), inputs, outputs, parameters, data_folder, locals())
    write_file(results_folder, y, "yFile.npy")
    phase.run(phases.get("sensitivity"), inputs, outputs, parameters, data_folder, locals(), out=out_post)
    return
```

Workflow templates: phases

Simulation

```
@on_failure(management='IGNORE')
@mpi(runner="mpirun", binary="$ALYA_BIN", args="{{name_sim}}", processes=alya_procs,
processes_per_node=alya_ppn, working_dir="{{simulation_wdir}}")
@task(returns=1, time_out=alya_timeout)
def simulation(name_sim, simulation_wdir, **kwargs):
    return
```

Prepare data

```
def prepare_data(**kwargs):
    prepare_args = kwargs
    variables = vars_func(prepare_args)
    out1 = prepare_sld(prepare_args, variables)
    bool_template_fie=check_template_exist(prepare_args,"template_fie")
    bool_template_dom=check_template_exist(prepare_args,"template_dom")
    if bool_template_fie:
        out2 = prepare_fie(prepare_args, variables, out=out1)
    if bool_template_dom:
        out3 = prepare_dom(prepare_args, out=out1)
    return out3
```

```
@task(returns=1)
def prepare_sld(prepare_args, variables, **kwargs):
    original_name_sim = get_value(prepare_args, "original_name_sim")
    ...
```

```
@task(returns=1)
def prepare_fie(prepare_args, variables, **kwargs):
    template = get_value(prepare_args, "template_fie")
    ...
```

Workflow configuration: yaml file

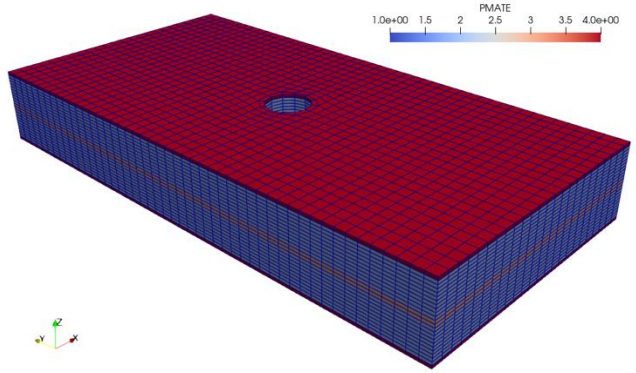
sensitivity.yaml

```
workflow_type: WORKFLOWS.SENSITIVITY_ANALYSIS.workflow.execution
phases:
  sampler:
    type: PHASES.SAMPLERS.morris.sampling
    arguments:
      - r: $parameters.r
      - p: $parameters.p
      - problem: $parameters.problema
  prepare_data:
    type: PHASES.BEFORESIMULATION.alya.prepare_data
    arguments:
      - mesh: $inputs.mesh_file
      - template_sld: $inputs.template_sld
      - template_dom: $inputs.template_dom
      - problem: $parameters.problem
      - simulation_wdir: $variables.simulation_wdir
      - values: $variables.values
      - name_sim: $variables.name_sim
      - original_name_sim: $variables.original_name_sim
  sim:
    type: PHASES.SIMULATIONS.alya.simulation
    arguments:
      - name_sim: $variables.name_sim
      - simulation_wdir: $variables.simulation_wdir
  post_process:
    type: PHASES.POSTSIMULATION.alya.collect_results
    arguments:
      - name_sim: $variables.name_sim
      - simulation_wdir: $variables.simulation_wdir
  post_process_merge:
    type: PHASES.POSTSIMULATION.alya.write_results
    arguments:
      - y: $variables.y
```

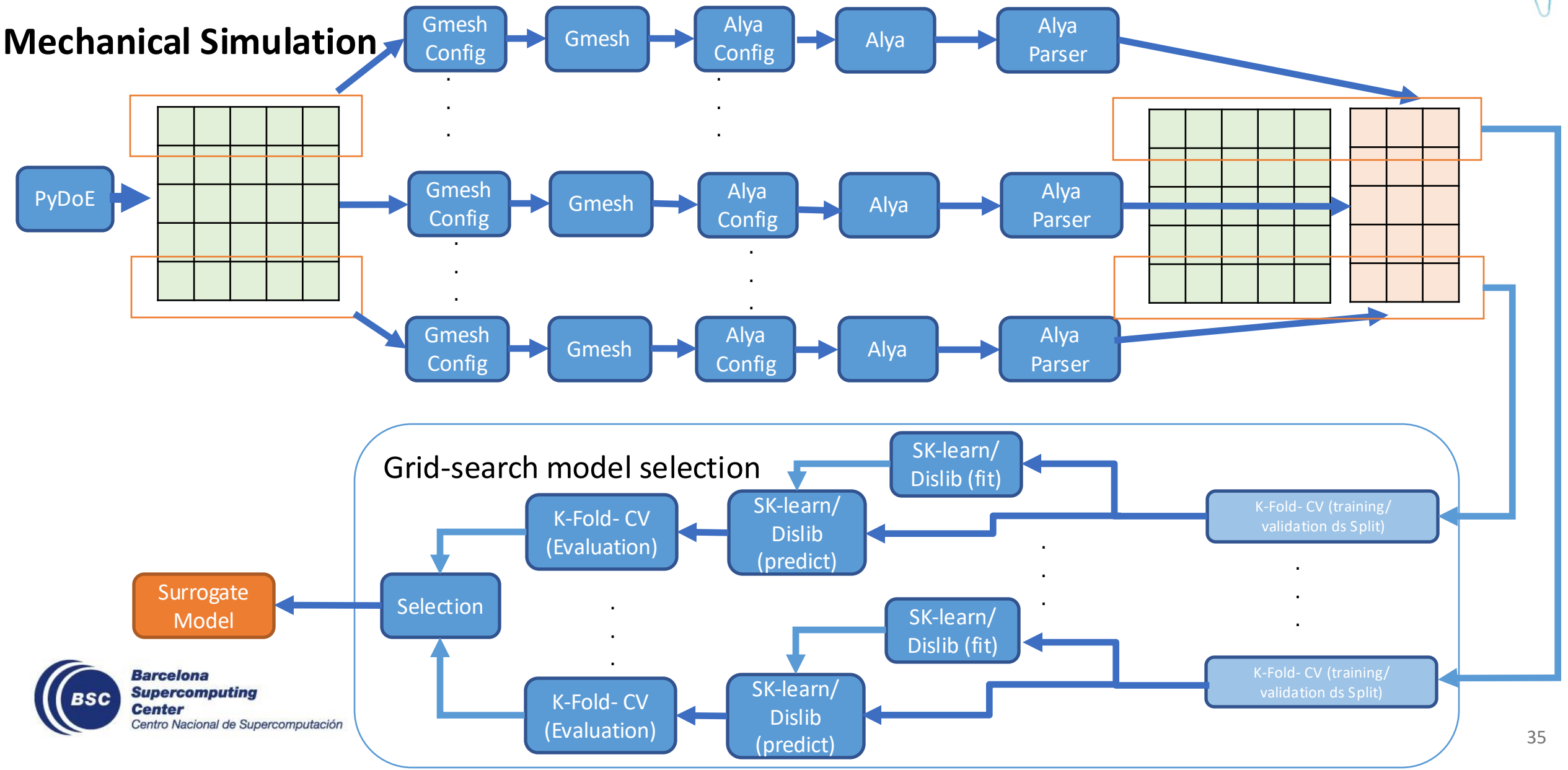
Actual problem: open hole tension

- Open hole geometry: test specimen with a hole in the middle
  - The simulation mechanically sets it under tension until it breaks
    - all virtually, numerically
- The workflow generates synthetic data which is simulated with Alya and trains an AI model
- The trained model is able to predict the maximum load at which the specimen will break given some inputs

- Mesh
  - Global element size: 0.5 mm x 0.5 mm x 0.13 mm
  - Total elements: 54332
  - Element types: Hexahedrons

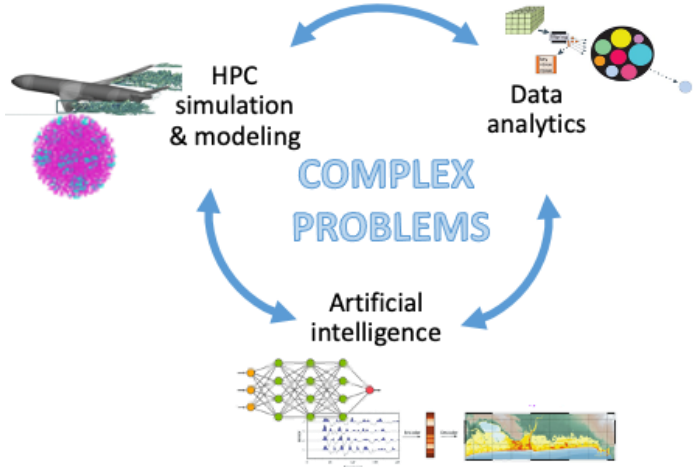


Specific workflow instance



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Final thoughts



- The growth of HPC systems will continue in the future
- Complex workflows will continue to evolve and need more powerful software tools and HPC systems
  - Hybrid workflows, combining classical + quantum
  - Deployed in the digital continuum
- PyCOMPSs is a programming environment that supports the development of this complex workflows
  - Coupling of HPC + AI / visualization
  - Dynamic changes

Further Information

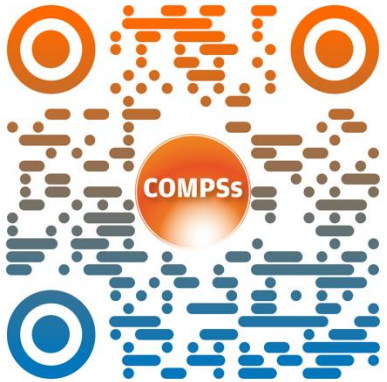
- Project page: http://www.bsc.es/compss
  - Documentation
  - Virtual Appliance for testing & sample applications
  - Tutorials
- Source Code

  https://github.com/bsc-wdc/compss
- Docker Image

  https://hub.docker.com/r/compss/compss
- Applications

  https://github.com/bsc-wdc/apps

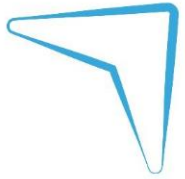
  https://github.com/bsc-wdc/dislib
- Dislib
  - https://dislib.readthedocs.io/en/latest/



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

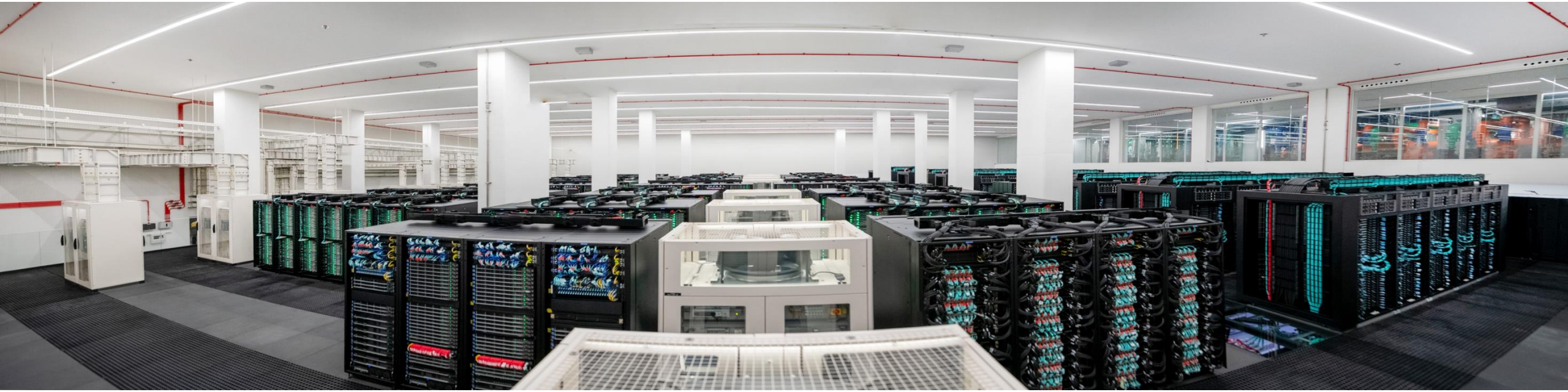
ACKs

HP2C-DT



CAELESTIS

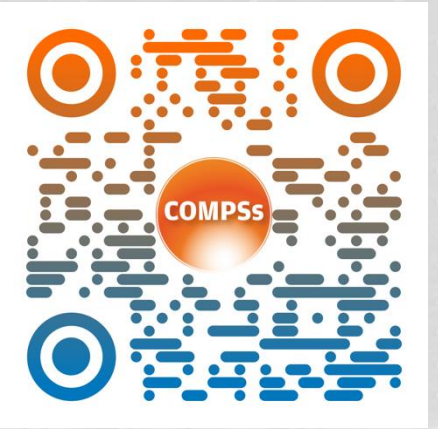
MareNostrum 5



rosa.m.badia@bsc.es

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Thanks!



rosa.m.badia@bsc.es