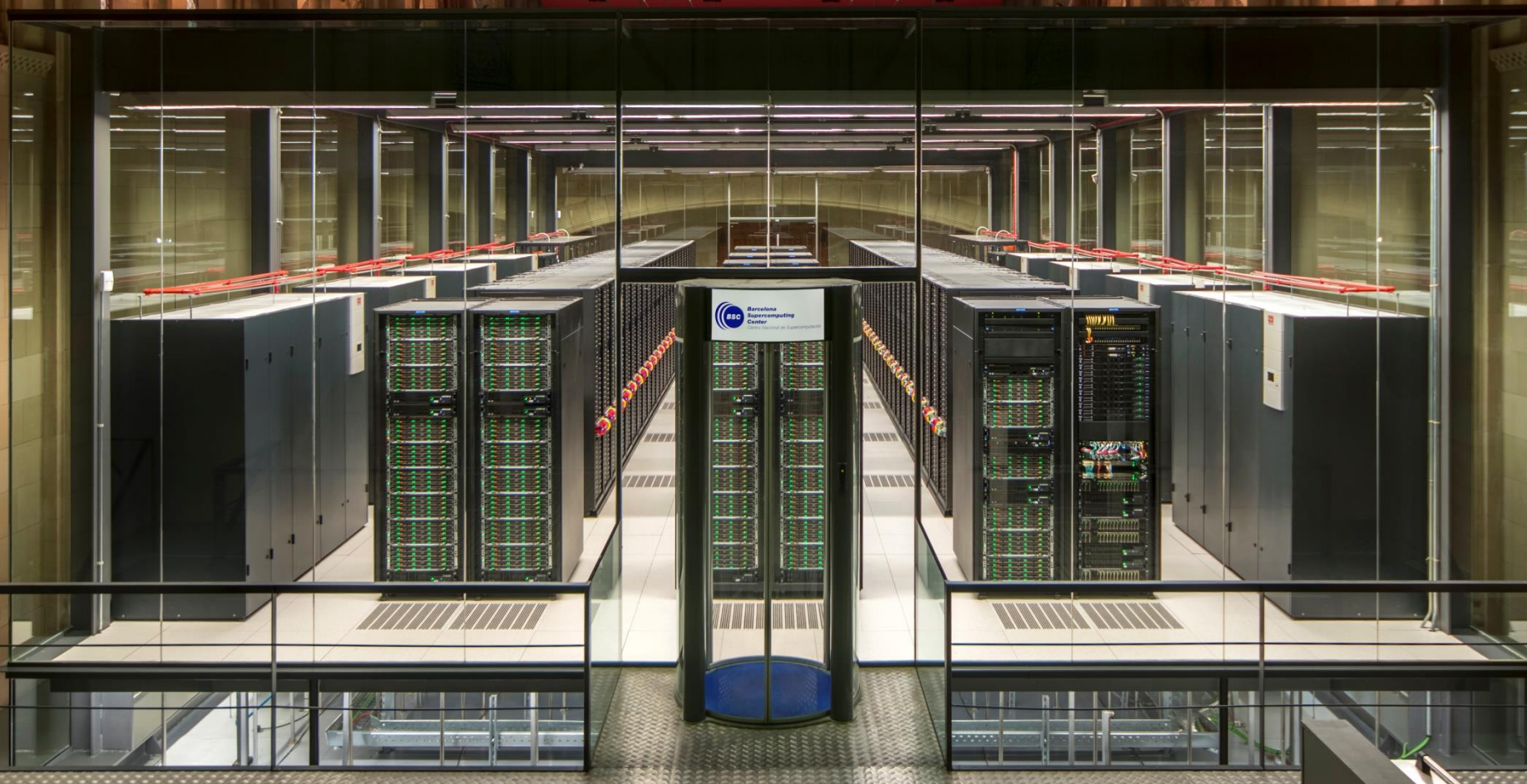




**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



Generalitat de Catalunya
**Departament de Recerca
i Universitats**



GOBIERNO
DE ESPAÑA
MINISTERIO
DE CIENCIA
E INNOVACIÓN



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH



UNIÓN EUROPEA
Fondo Europeo de Desarrollo Regional



Optimising workflow lifecycle management: development, HPC-ready containers deployment and reproducibility

Raül Sirvent, Rosa M Badia

18/11/2024

SC24 tutorial, Atlanta, 18 Novembre 2024

Tutorial website

https://github.com/bsc-wdc/Tutorial_SC24



Agenda

8:30 – 8:45	Overview of tutorial agenda	Rosa M Badia
8:45 – 9:15	Part 1.1: Hybrid HPC+AI+DA workflow development with PyCOMPSs <ul style="list-style-type: none">- Context of the workflows at BSC- Overview of workflow development with PyCOMPSs- Extensions for the integration of HPC with AI and DA- Sample workflows	Rosa M Badia
9:15 – 9:45	Part 1.2: Workflows' reproducibility through provenance <ul style="list-style-type: none">- Motivation for workflow provenance- Design of the recording mechanism- Sharing experiments for reproducibility	Raül Sirvent
9:45 - 10:00	Part 1.3: HPC ready container images <ul style="list-style-type: none">- Motivation for architecture specific containers- Overview of the Container Image Creation service- Example of HPC ready container generation- Workflow example for hands-on	Rosa M Badia
10:00 - 10:30	Coffee break	

Agenda

10:30 – 10:45	Hands-on preparation (credentials distribution, how to access, etc)	All presenters
10:45 – 11:15	Part 2.1: Hands-on session: Sample workflows with PyCOMPSs, execution with containers, task-graph generation, tracefile generation (optional)	Rosa M Badia
11:15 – 11:55	Part 2.2: Hands-on session: How to automatically record workflow provenance and use it to share experiments in WorkflowHub	Raül Sirvent
11:55 - 12:00	Tutorial conclusions	All presenters

EuroHPC JU systems

Pre-Exascale
Petascale

	Status	Country	Peak performance	Architecture
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 + NVIDIA Ampere
MareNostrum 5	Operational	Spain	250.00 petaflops	Intel Sapphire Rapids, NVIDIA Hopper, AMD Instinct MI300X, NVIDIA Grace, Intel Emeralds, Intel Xeon
Meluxina	Operational	France	10.05 petaflops	AMD EPYC + NVIDIA Ampere A100
Vega	Operational	Germany	10.05 petaflops	AMD Epyc 7H12 + Nvidia A100
Karolina	Operational	Czech 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 under installation in Jülich, Germany



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

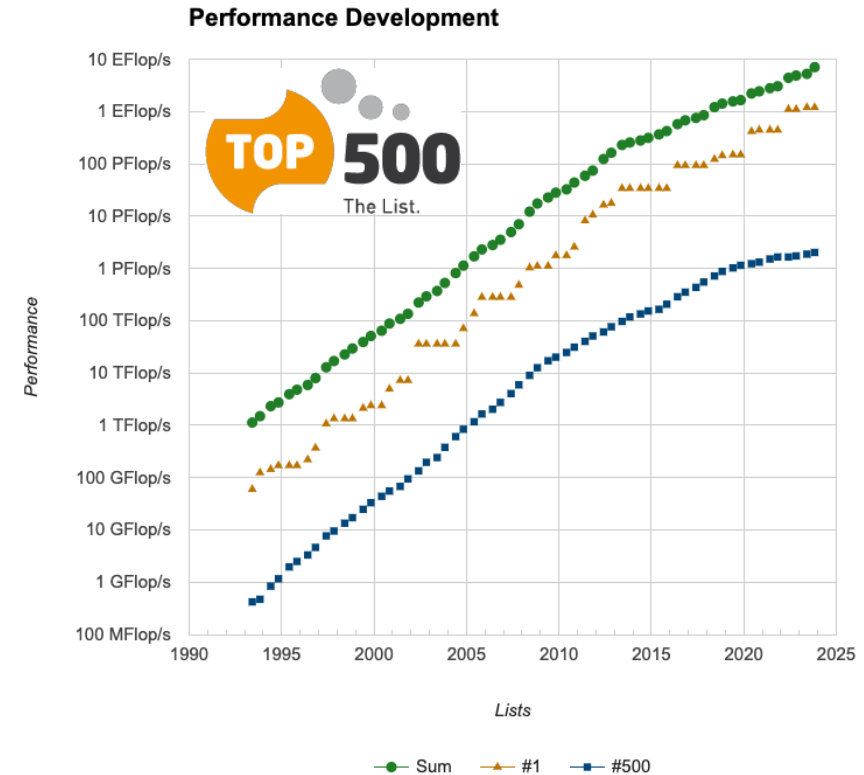
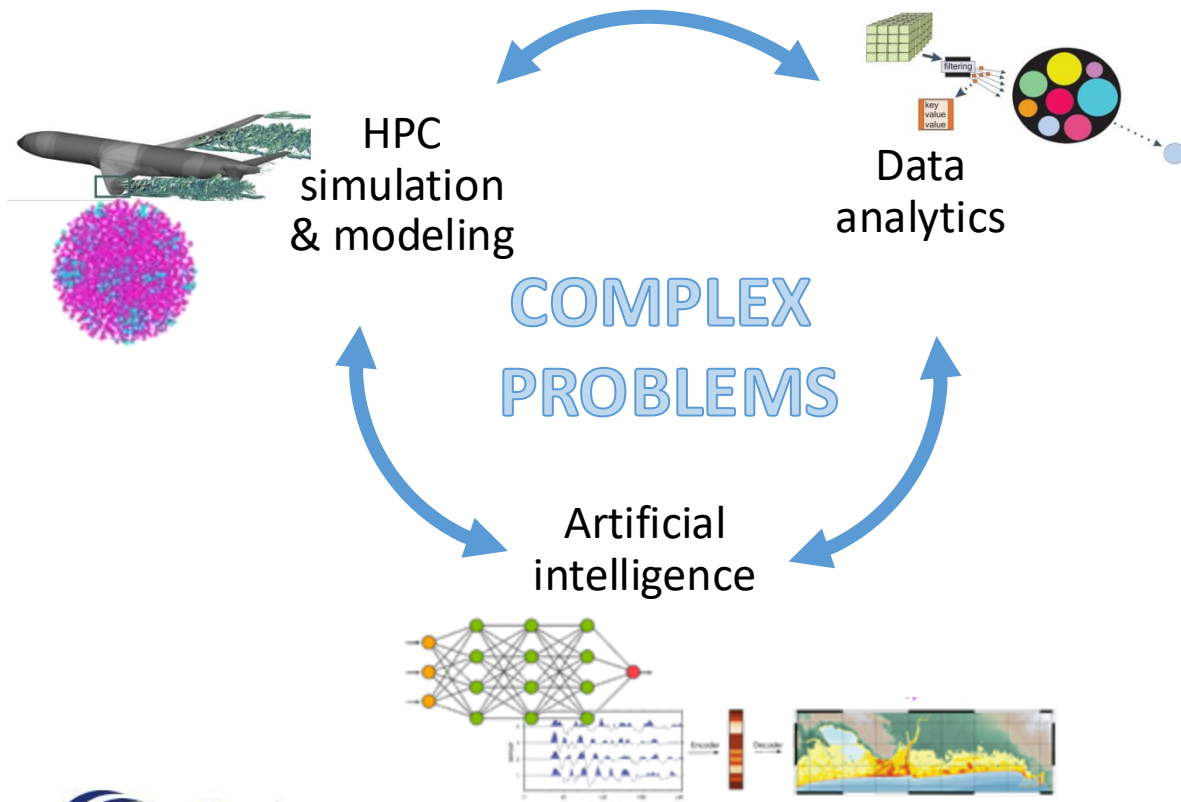
MareNostrum 2
2006 – 94.2 Tflops
1st Europe / 5th World
New technologies

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

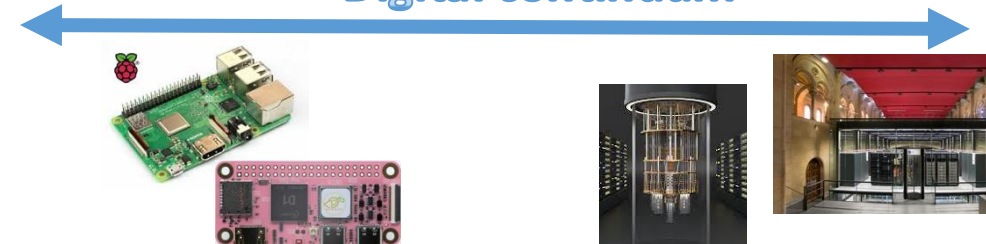
MareNostrum 4
2017 – 11.1 Pflops
2nd Europe / 13th World
New technologies

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

Complex problems for complex computing infrastructures

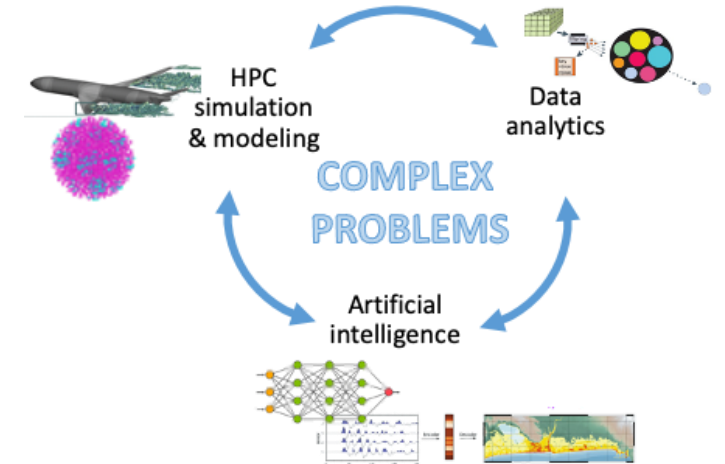


Digital continuum



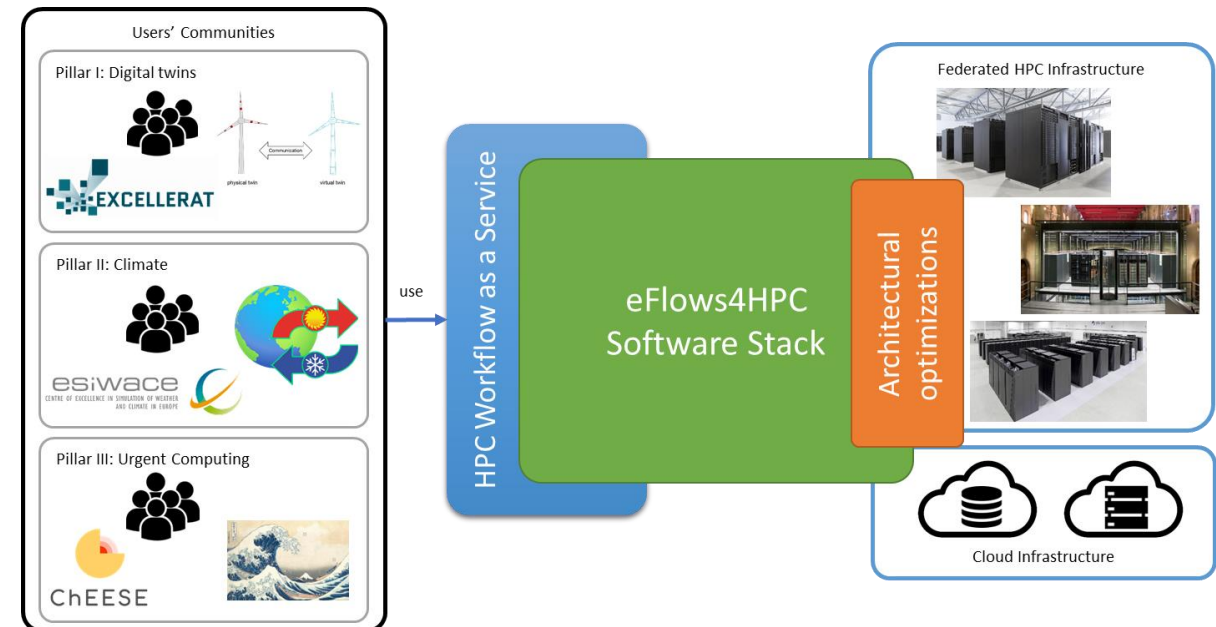
Workflow lifecycle challenges

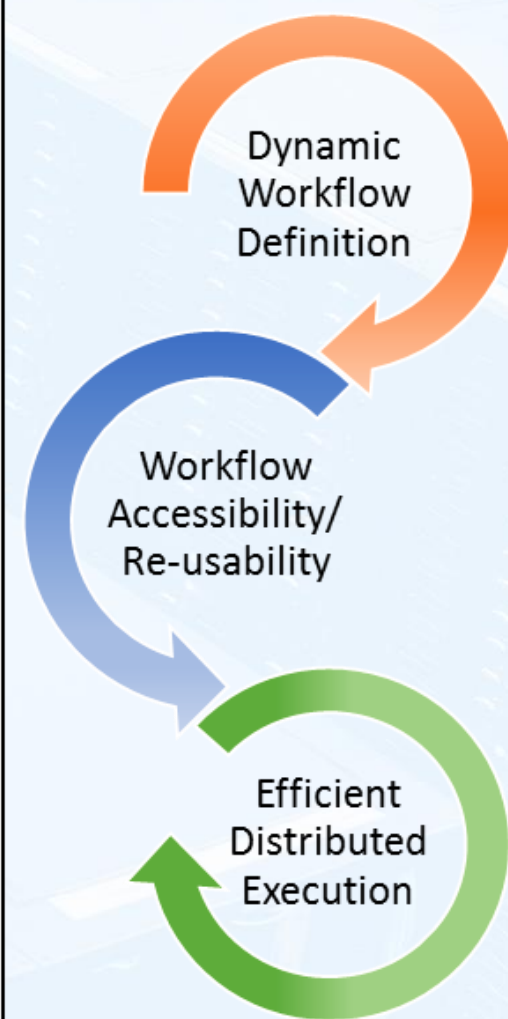
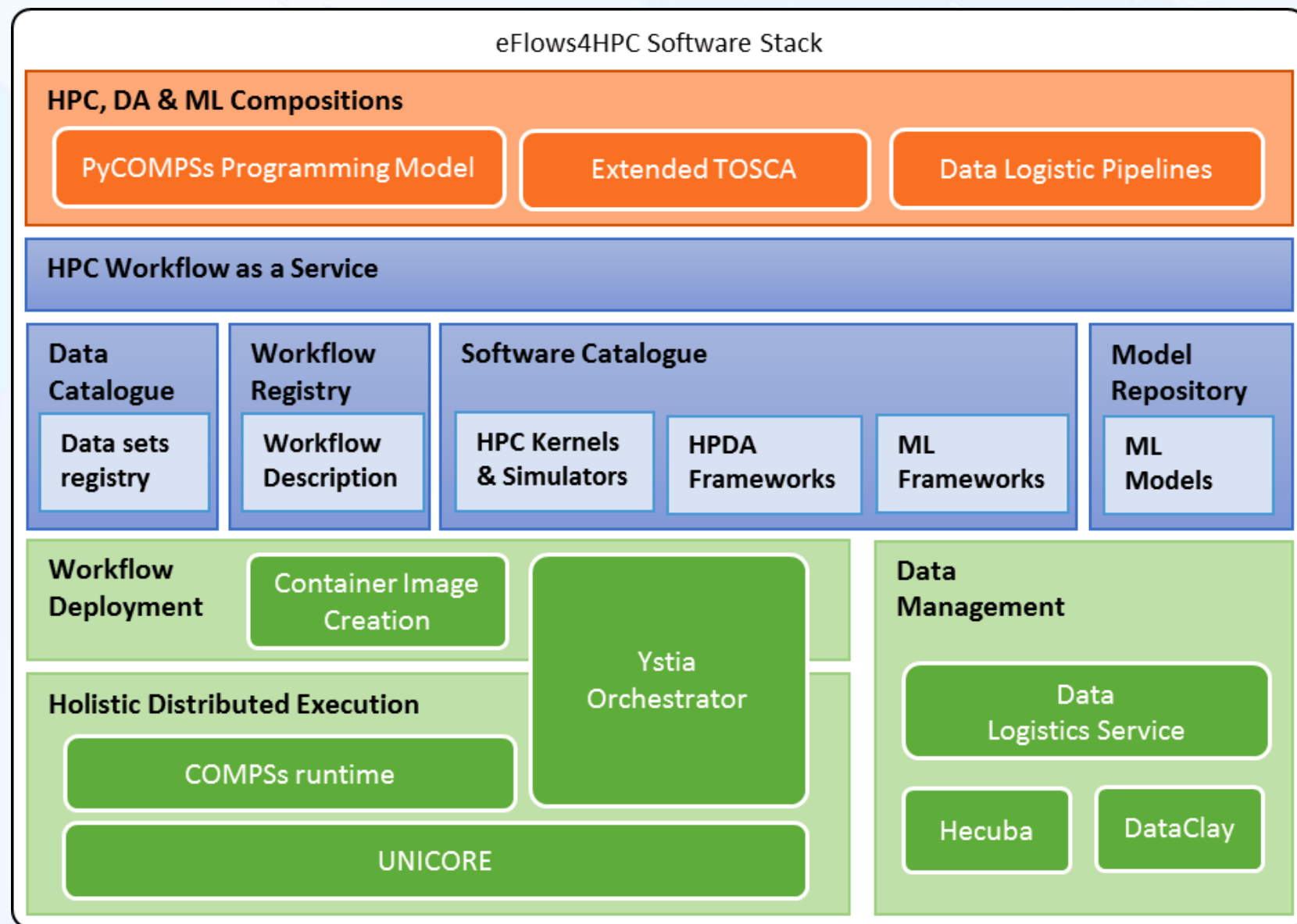
- Workflow development
 - Different programming models and environments
 - Workflow deployment
 - Can we make it easier to new HPC users?
 - Workflow operation
 - Go beyond static workflows
 - Not only computational aspects, data management as well
- Sample projects:



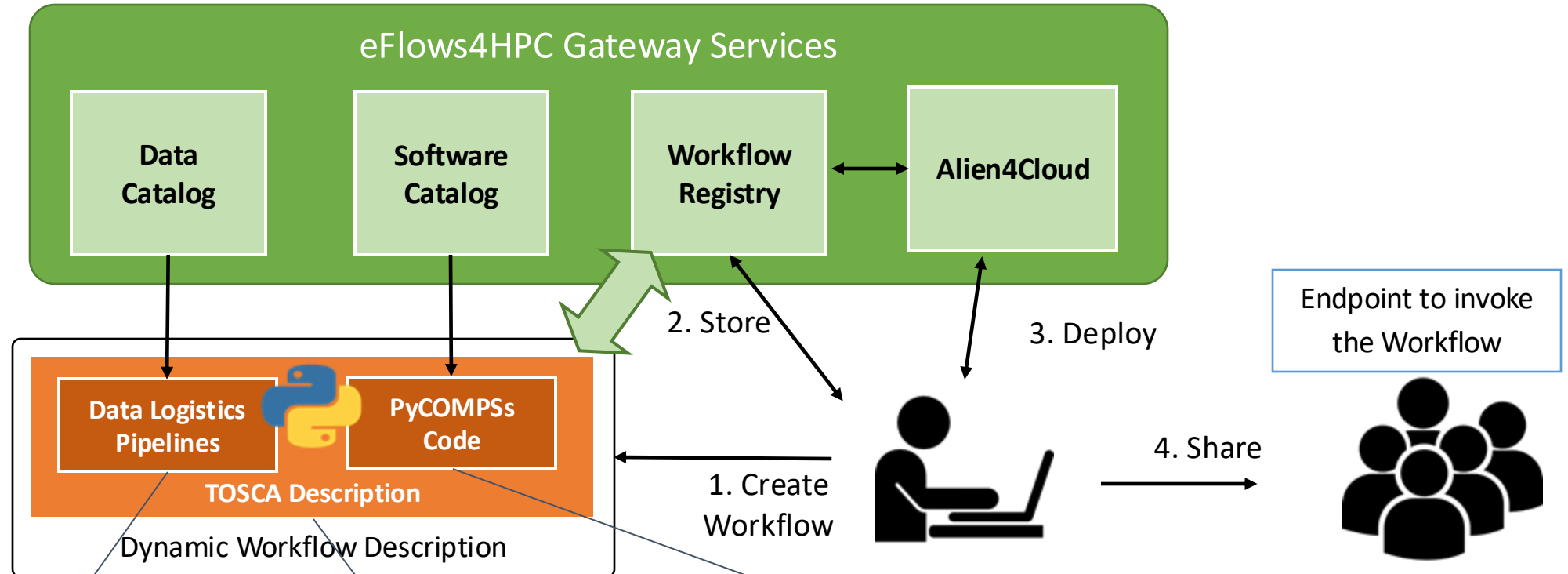
eFlows4HPC in a nutshell

- Software tools stack that makes easier the development and management of complex workflows:
 - Combine different aspects
 - HPC, AI, data analytics
 - Reactive and dynamic workflows
 - Autonomous workflow steering
 - Full lifecycle management
 - Not just execution
 - Data logistics and Deployment
- HPC Workflows as a Service:
 - Mechanisms to make easier the use and reuse of HPC by wider communities
- Architectural Optimizations:
 - Selected HPC – AI Kernels Optimized for GPUs, FPGA, EPI
- Validation Pillar's
 - End-user workflows linked to CoEs





HPCWaaS: Workflow lifecycle overview



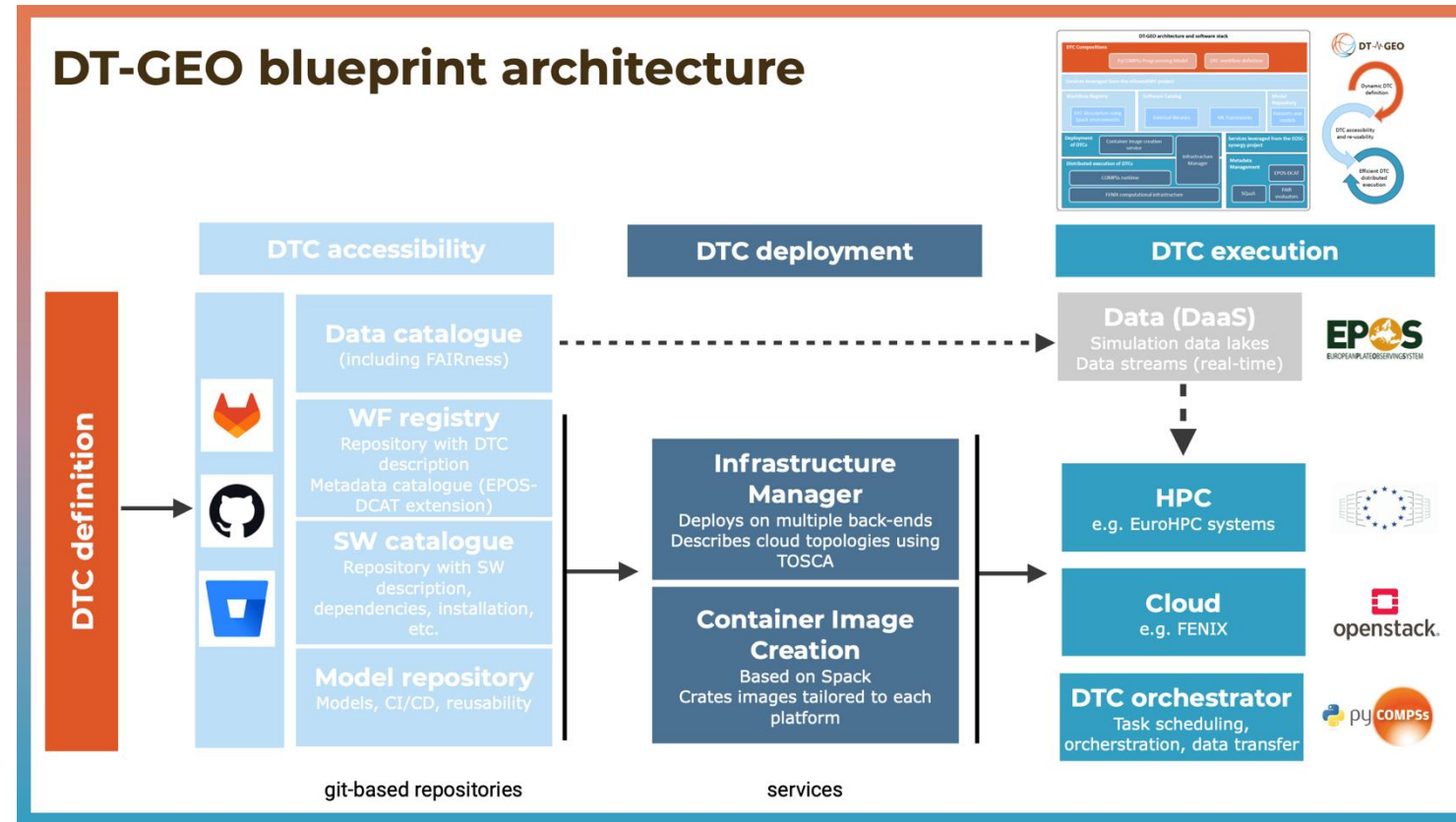
Description of data movements as Python functions.
Input/output datasets described at Data Catalog

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

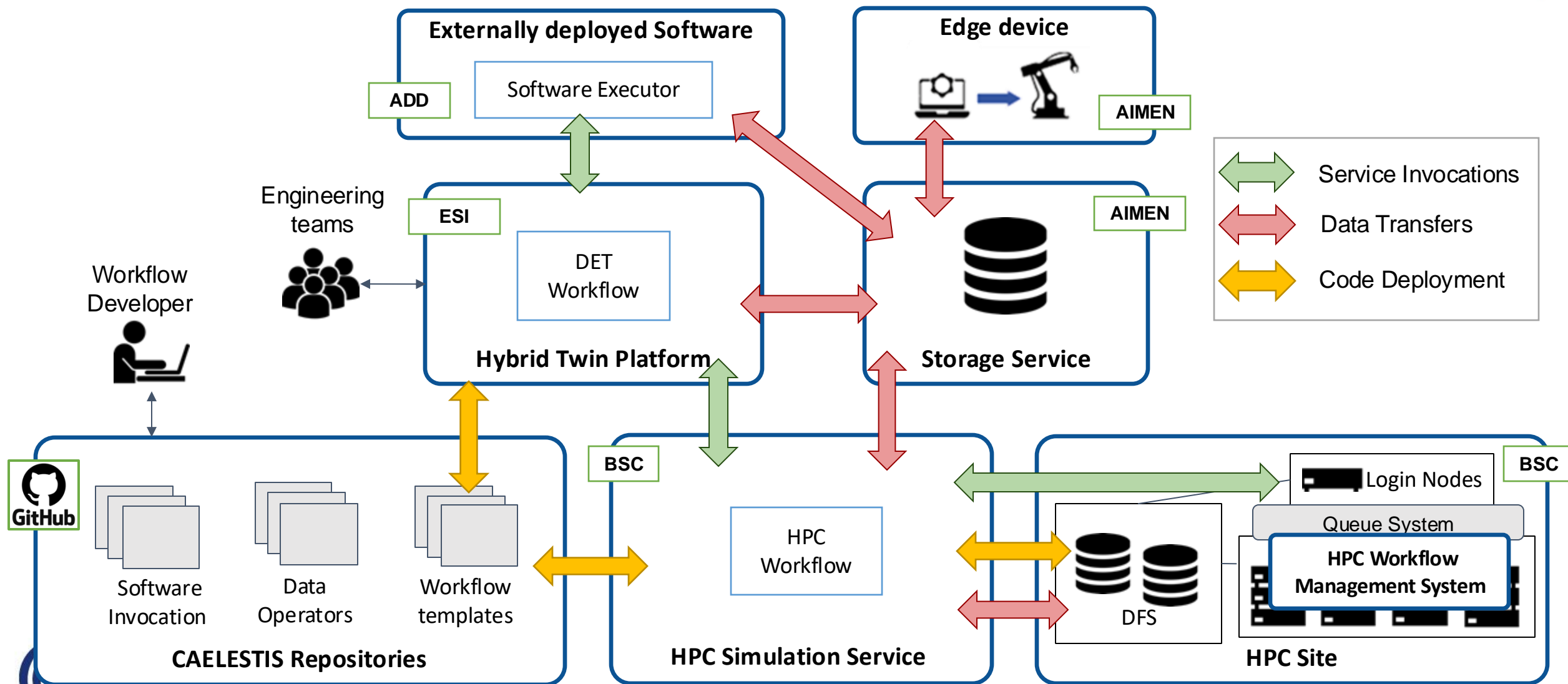
Topology of the components involved in the workflow
lifecycle and their relationship.

DT-GEO: implementing a geophysical extremes digital twin

- 12 Digital Twin Components (DTCs) addressing specific hazardous phenomena from
 - Volcanoes,
 - Tsunamis
 - Earthquakes, and
 - Anthropogenically-induced extremes
- DTCs implemented as workflows, many of them inheriting the eFlows4HPC architecture and services



CAELESTIS Simulation Ecosystem Architecture



Integrating different computations in PyCOMPSs



**Barcelona
Supercomputing
Center**

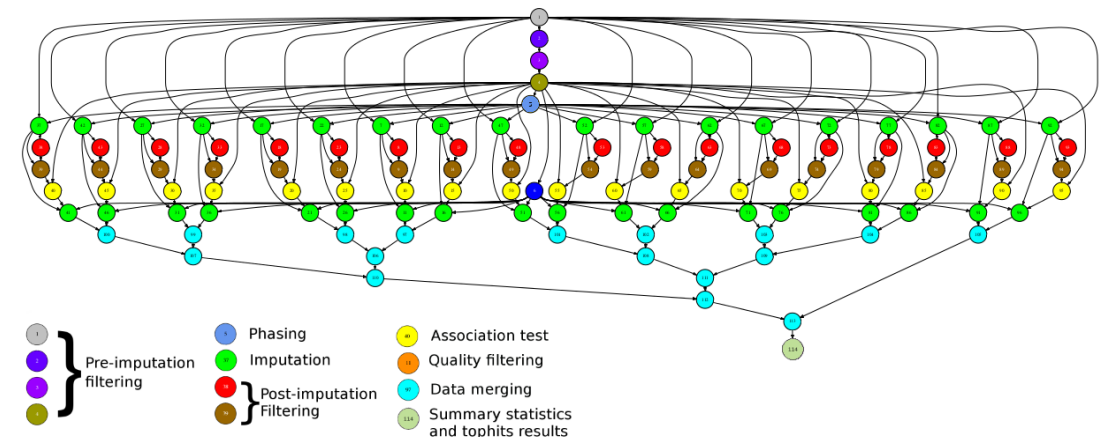
Centro Nacional de Supercomputación

Workflows in 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
- Based in Python
 - Further extended in eFlows4HPC for better integration of HPC, AI and Big Data

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



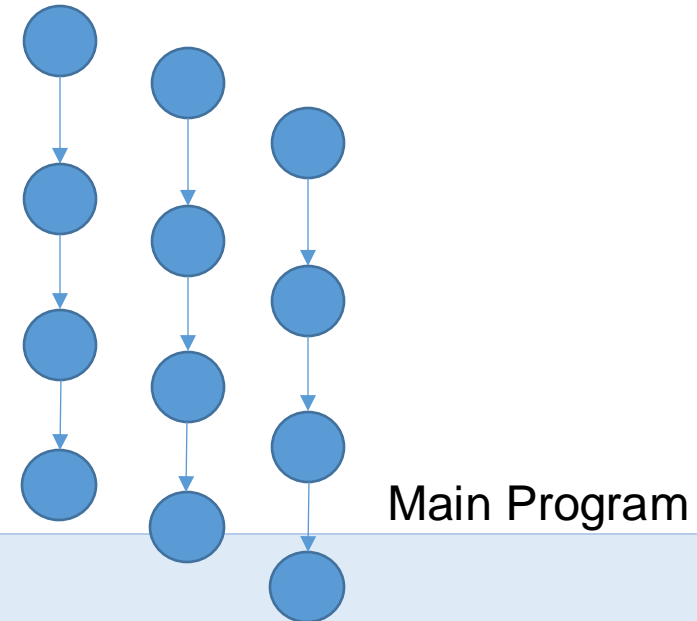
PyCOMPSs syntax



- 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
```



```
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
```


Other interesting annotations

Task constraints: enable to define HW or SW requirements

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

Support for other programming models

```
@constraint (computingUnits= "248")
@mpi (binary="mySimulator", runner="mpirun", computingNodes= "16", ...)
@task (returns=int, stdoutFile=FILE_OUT_STDOUT, ...) def
nems(stdoutFile, stderrFile):
    pass
```

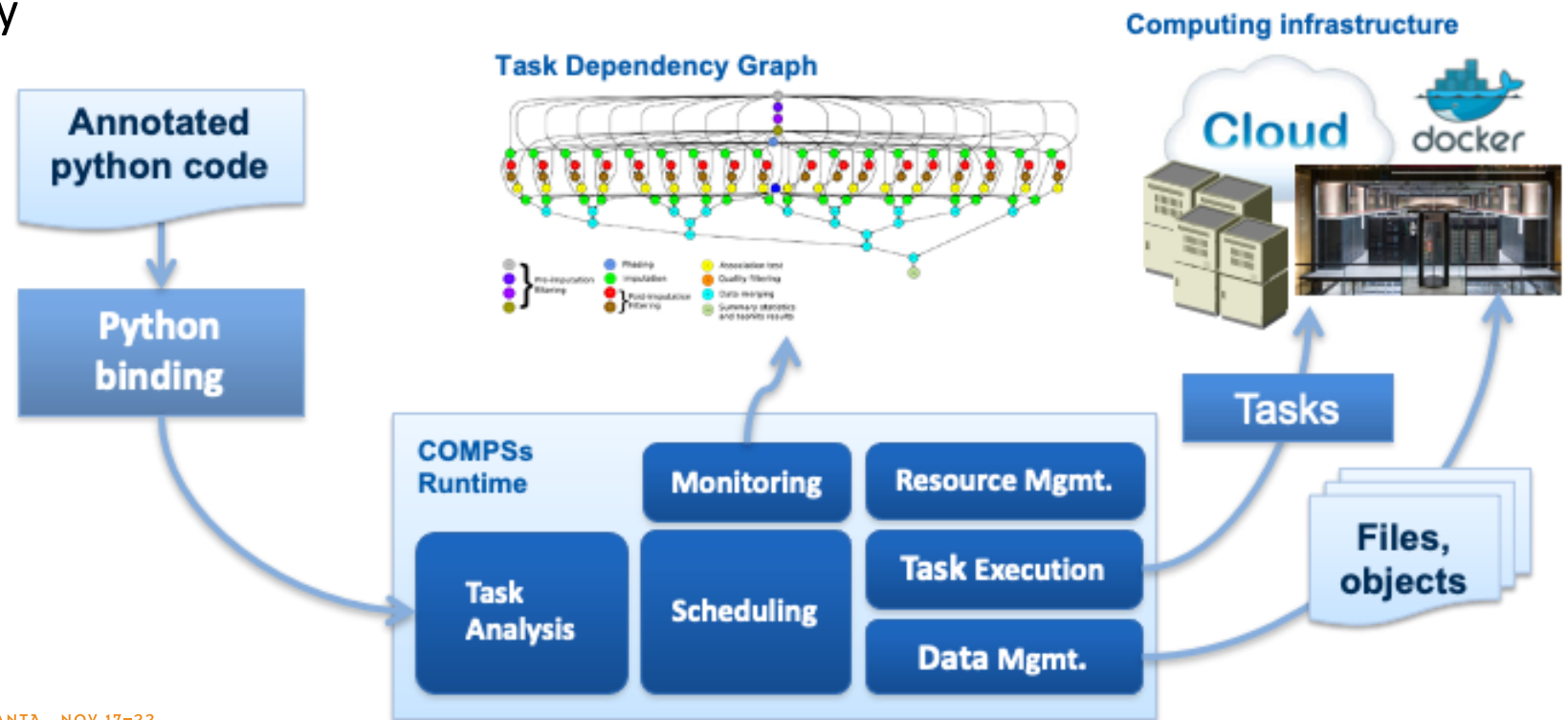
Task failure management

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

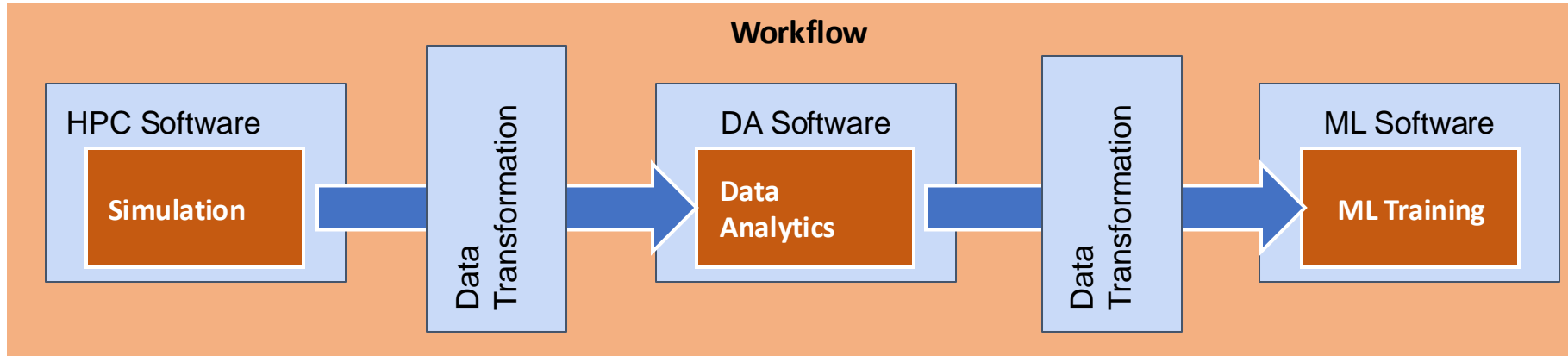
PyCOMPSs runtime



- Runtime deployed as a distributed master-worker
 - Description of computational infrastructure in an XML file
- Sequential execution starts in master node and tasks are offloaded to worker nodes
- All data scheduling decisions and data transfers are performed by the runtime
- Support for horizontal elasticity
- Support for containers



Interfaces to integrate HPC/DA/ML



- Goal:
 - Reduce the required glue code to invoke multiple complex software steps
 - Developer can focus in the functionality, not in the integration
 - Enables reusability
- Two paradigms:
 - Software invocation
 - Data transformations

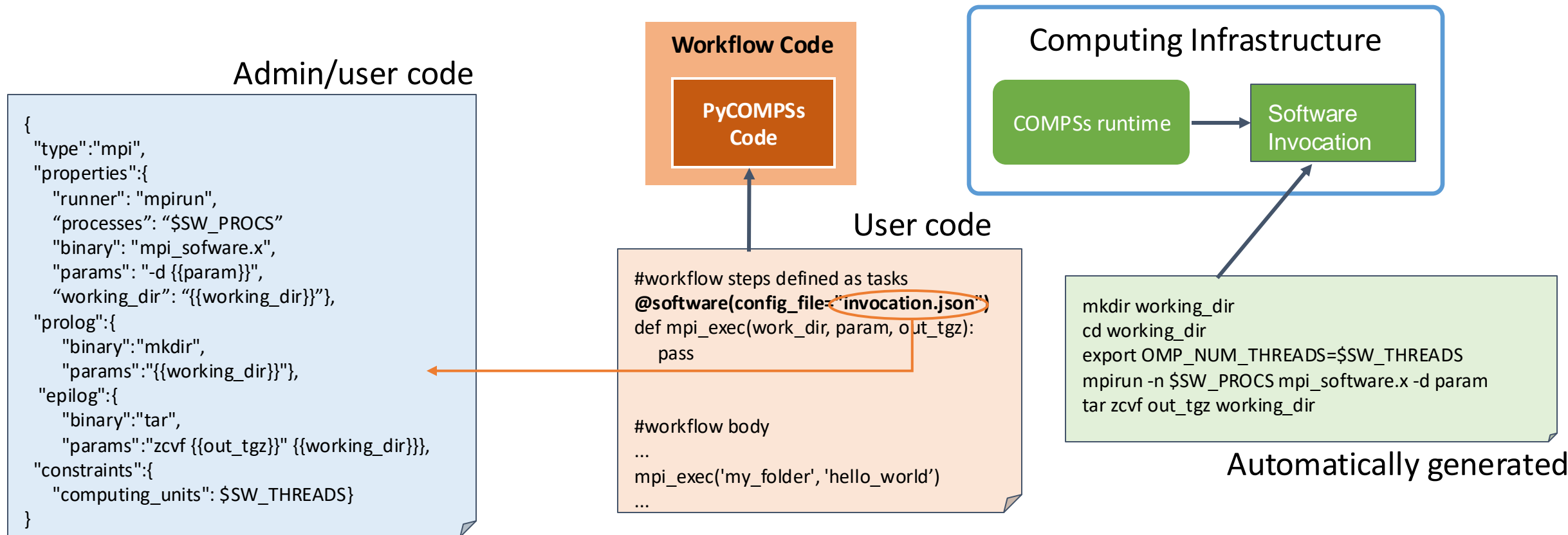
workflow steps defined as tasks

```
@data_transformation (input_data, transformation description)
@software (invocation description)
def data_analytics (input_data, result):
    pass

simulation (input_cfg, sim_out)
data_analytics (sim_out, analysis_result)
ml_training (analysis_result, ml_model)
```

workflow body

Software Invocation description



Software invocation
description
Stored in software catalog

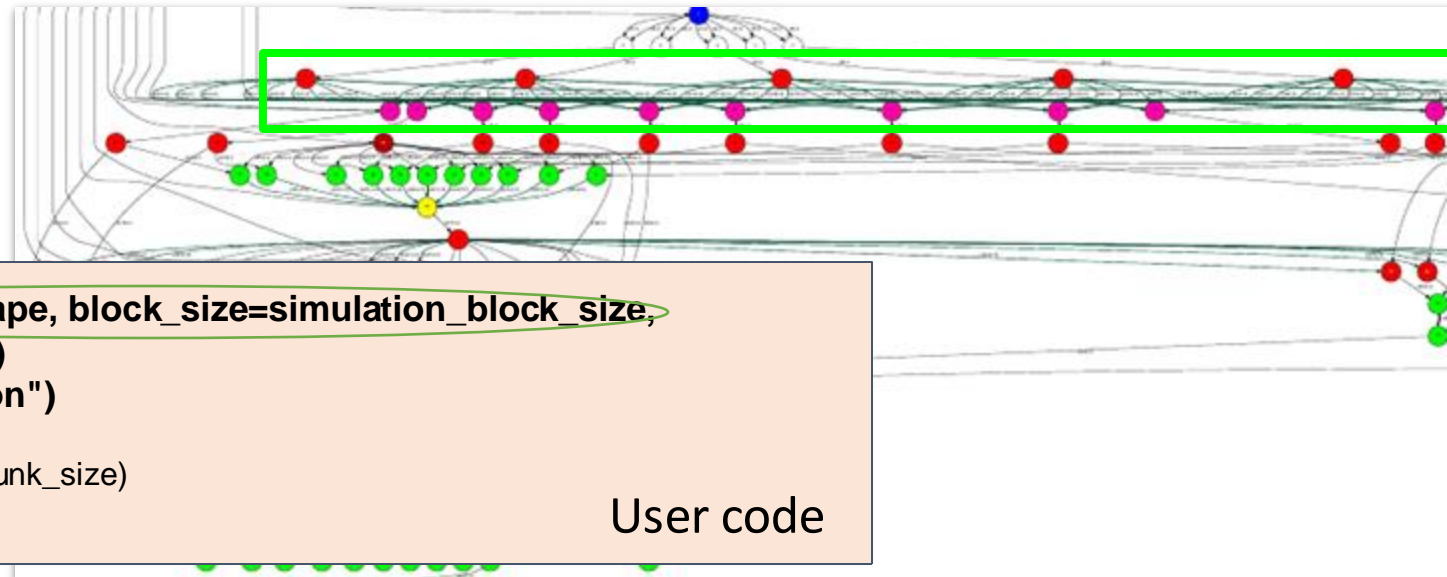
- Converts a Python function of a software invocation to a PyCOMPSs task
- Takes information from the description in json
- Enables reuse in multiple workflows

Data transformations

- A data transformation changes the data without requiring extra programming from the developer

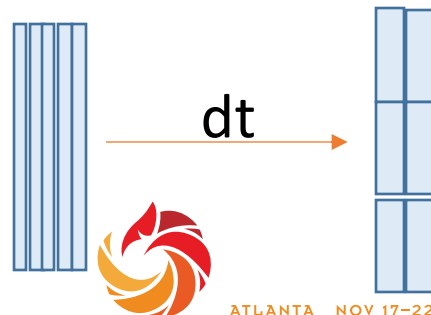
Admin/user code

```
def load_blocks_rechunk(blocks, shape, block_size, new_block_size):  
    ...  
    SnapshotMatrix = load_blocks_array (final_blocks, shape, block_size);  
    return SnapshotMatrix
```



```
@dt("blocks", load_blocks_rechunk, shape=expected_shape, block_size=simulation_block_size,  
    new_block_size=desired_block_size, is_workflow=True)  
@software(config_file = SW_CATALOG + "/dislib/dislib.json")  
def rSVD(blocks, desired_rank=30):  
    u,s = rsvd(blocks, desired_rank, A_row_chunk_size, A_column_chunk_size)  
    return u
```

User code



```
...  
model, parameters = load_model_parameters(model_file)  
for cfg in sim_cfgs:  
    sim_results.append(execute_FOM_instance(model,parameters,[cfg]))  
rom = rSVD(sim_results, desired_rank)  
...
```

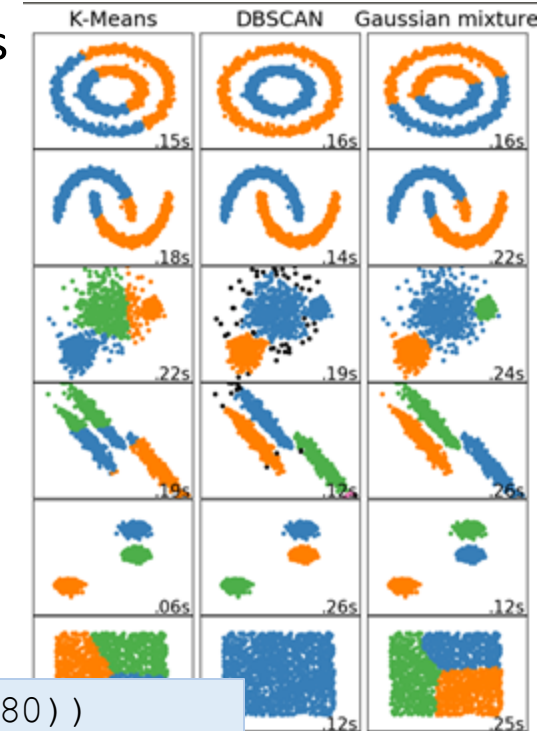
User code

Main workflow program

Dislib: parallel machine learning



- dislib: Collection of machine learning algorithms developed on top of PyCOMPSs
 - Unified interface, inspired in scikit-learn (fit-predict)
 - Based on a distributed data structure (ds-array)
 - Unified data acquisition methods
 - Parallelism transparent to the user – PyCOMPSs parallelism hidden
 - Open source, available to the community
- Provides multiple methods:
 - Data initialization
 - Clustering
 - Classification
 - Model selection, ...

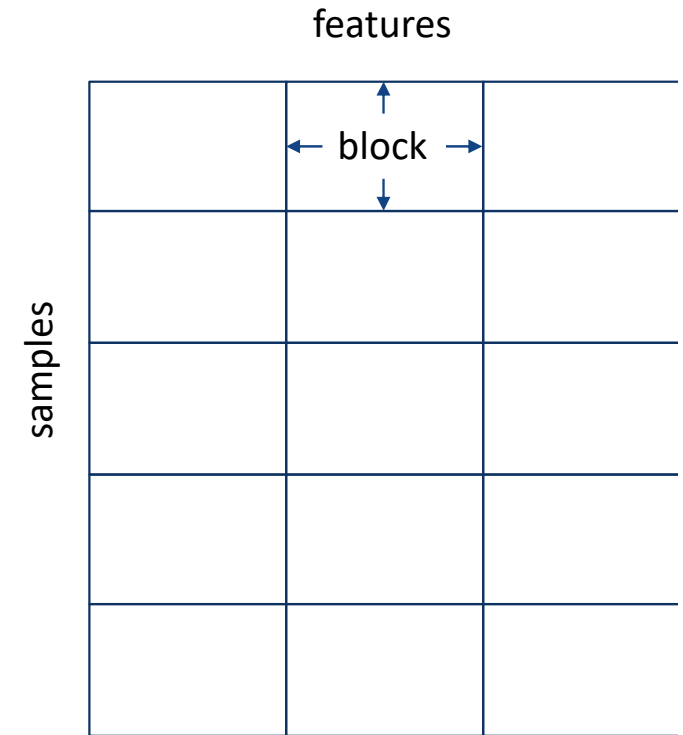


```
x = load_txt_file("train.csv", (10, 780))
x_test = load_txt_file("test.csv", (10, 780))

kmeans = KMeans(n_clusters=10)
kmeans.fit(x)
kmeans.predict(x_test)
```


Dislib data structure: Distributed arrays (ds-arrays)

- 2-dimensional structure (i.e., matrix)
 - Divided in blocks (NumPy arrays)
- Works as a regular Python object
 - But not stored in local memory!
- Methods for instantiation and slicing with the same syntax of numpy arrays:
 - Internally parallelized with PyCOMPSs:
 - Loading data (e.g. from a text file)
 - Indexing (e.g., `x[3]`, `x[5:10]`)
 - Operators (e.g., `x.min()`, `x.transpose()`)
- ds-arrays can be iterated efficiently along both axes
- Samples and labels can be represented by independent distributed arrays



Internally parallelized with PyCOMPSs

Computes pair wise distances of points to centers and accumulates new values to compute new centers (partials)

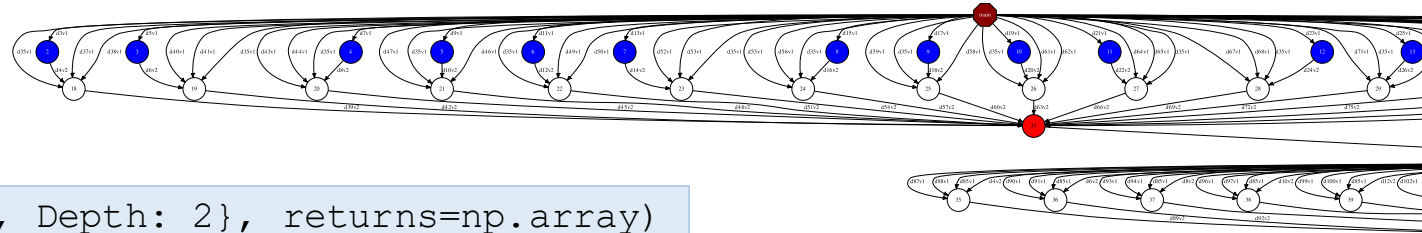
```
@task(blocks={Type: COLLECTION_IN, Depth: 2}, returns=np.array)
def _partial_sum(blocks, centers):
    ...
    return partials
```

```
@task(returns=dict)
def _merge(*data):
    ...
    return accum
```

Reduces values of centers through merge task

```
def fit(self, x, y=None):
    """ Compute K-means clustering.
    old_centers = None
    iteration = 0

    while not self._converged(old_centers, iteration):
        old_centers = self.centers.copy()
        partials = []
        for row in x._iterator(axis=0):
            partial = _partial_sum(row._blocks, old_centers)
            partials.append(partial)
        self._recompute_centers(partials)
        iteration += 1
    self.n_iter = iteration
    return self
```



x: ds-array

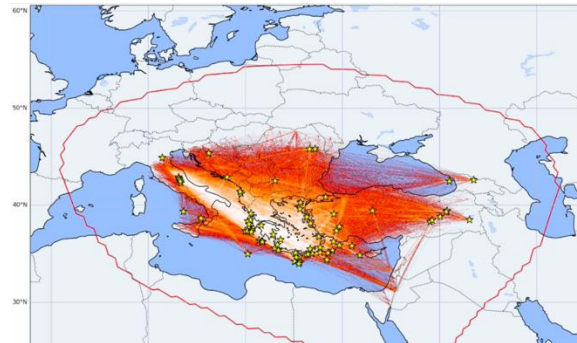
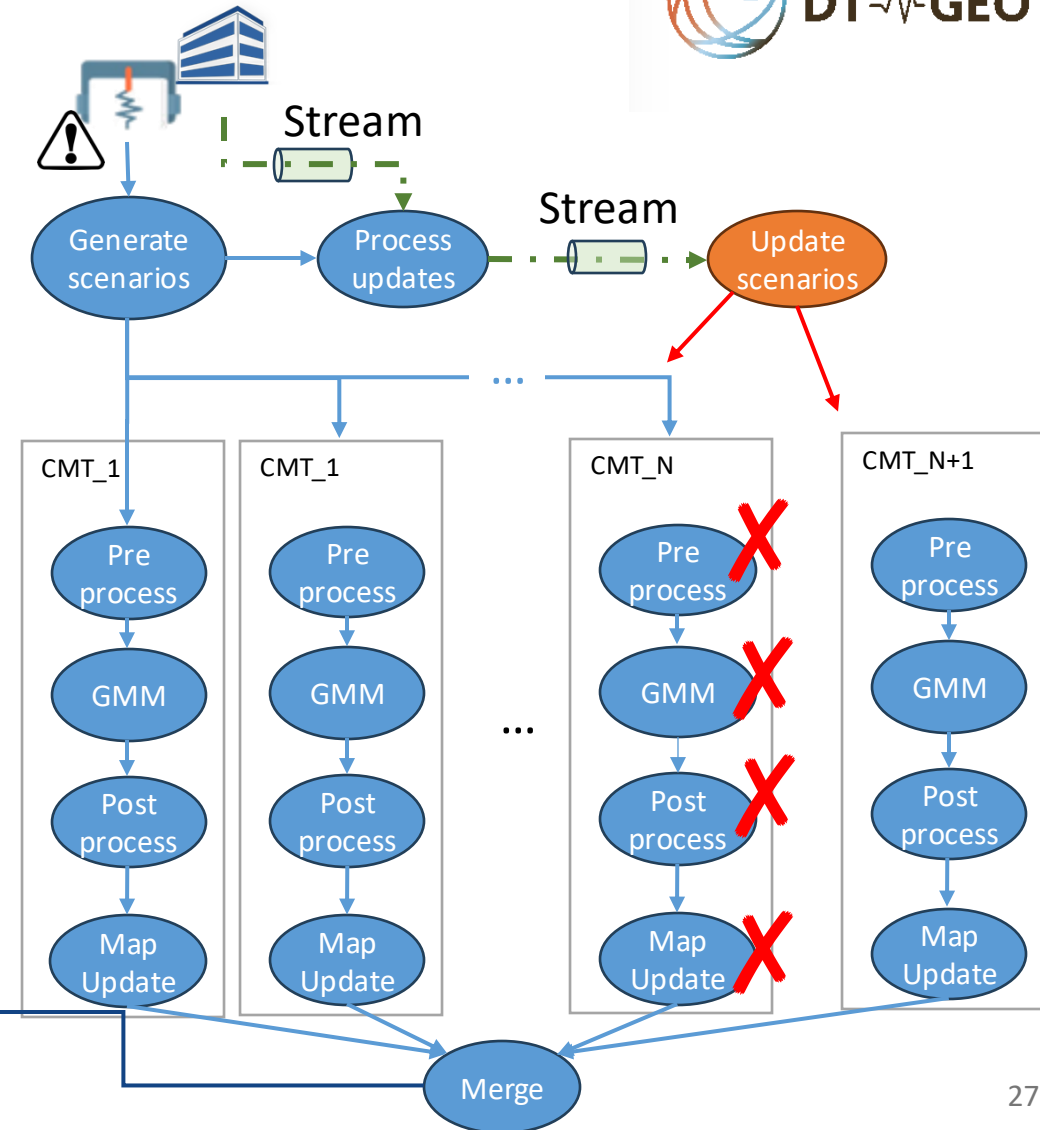
Sample workflows

- UCIS4EQ – Earthquake simulation – eFlows4HPC and DT-GEO
- Digital Twin Component (DTC) for Volcanoes workflow – DT-GEO
- Surrogate model creation – CAELESTIS

Event-driven cancellation/creation

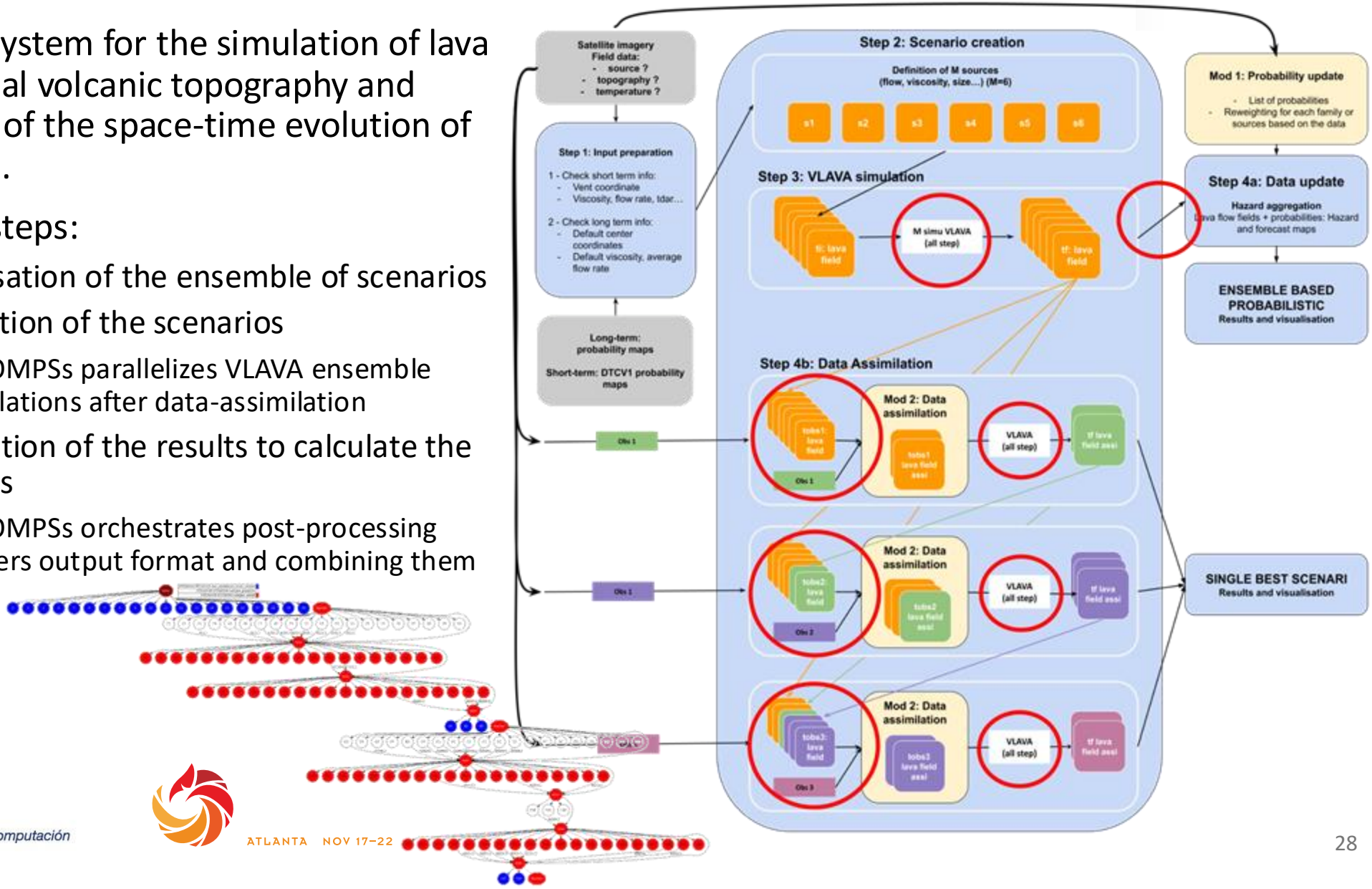
UCIS4EQ: HPC-based urgent seismic simulation workflow

- Evaluation of scenarios after the occurrence of a seismic event
- Combines multiple web services and HPC simulation (Salvus)
- Workflow Dynamicity:
 - Usage of **data streaming** for communication of events
 - On event occurrence API supports:
 - **Dynamic cancellation** of task groups
 - **Dynamic creation** of new set of tasks



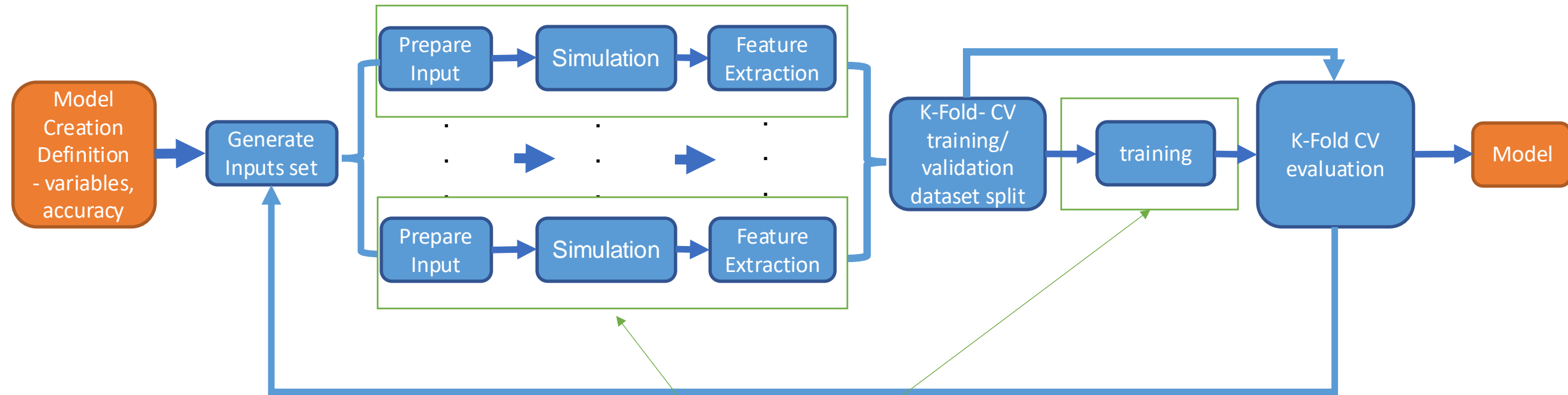
DTC Volcanoes

- Automated system for the simulation of lava flows on a real volcanic topography and visualization of the space-time evolution of the lava field.
- Three main steps:
 1. Initialisation of the ensemble of scenarios
 2. Simulation of the scenarios
 - PyCOMPSs parallelizes VLAVA ensemble simulations after data-assimilation
 3. Aggregation of the results to calculate the forecasts
 - PyCOMPSs orchestrates post-processing gathers output format and combining them



Workflow templates

Surrogate Model Creation Workflow

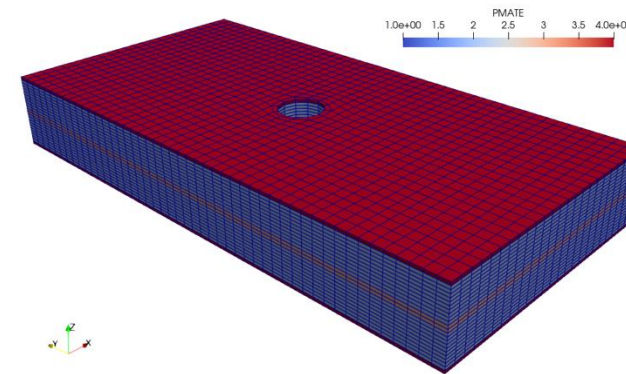


Customized for each the model

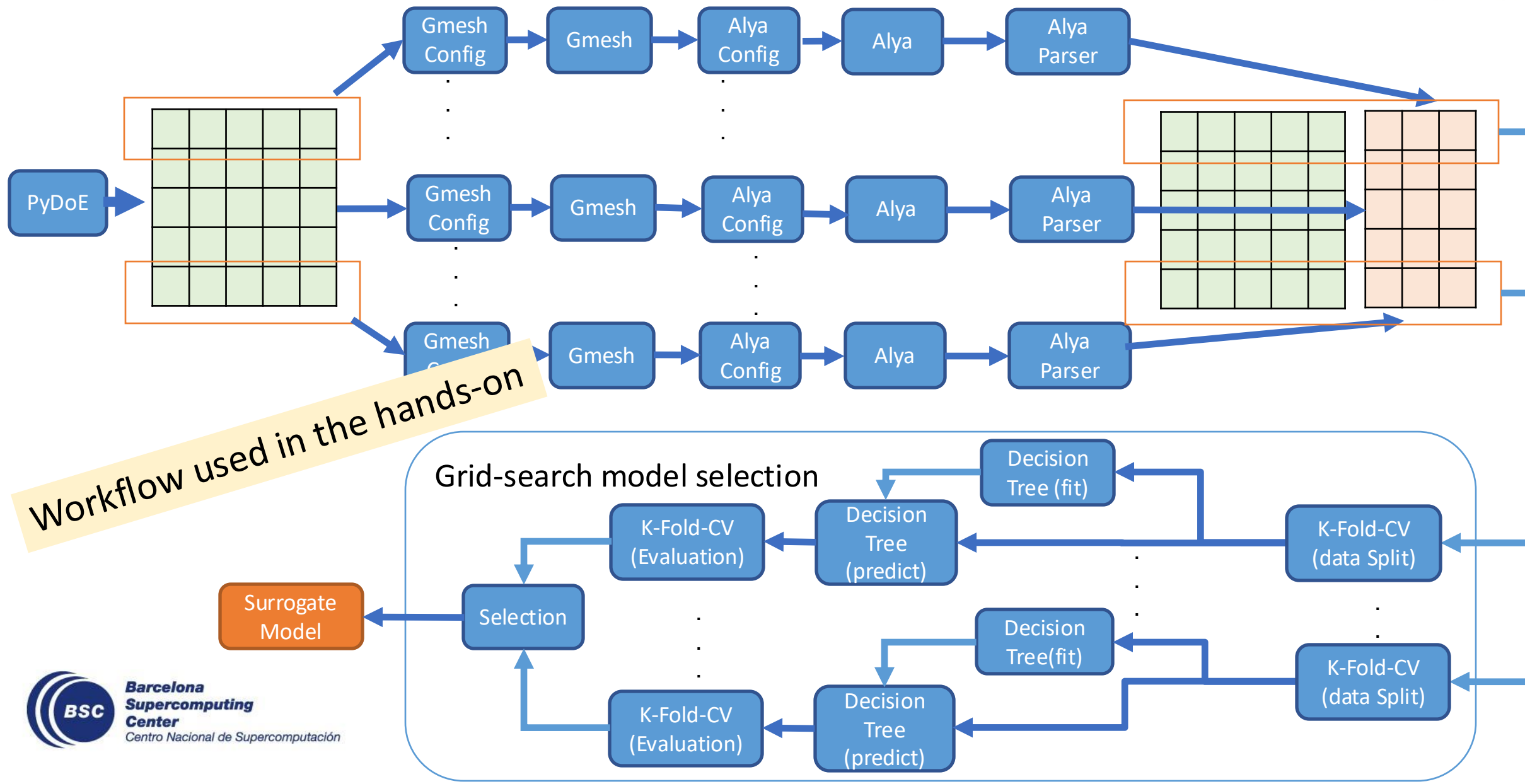
Actual problem: open hole tension

- 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 subsequently trains a 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



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



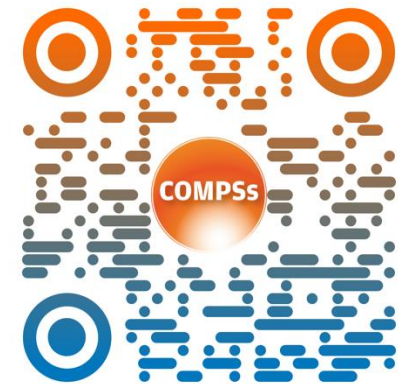
<https://github.com/bsc-wdc/apps>

<https://github.com/bsc-wdc/dislib>



- Dislib

<https://dislib.readthedocs.io/en/latest/>



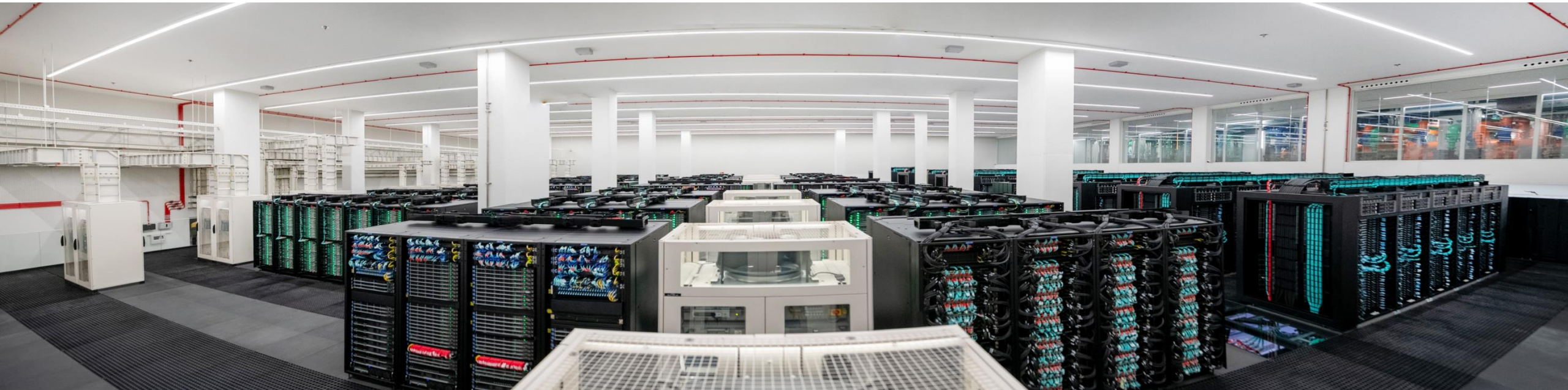
ACKs



HP2C-DT



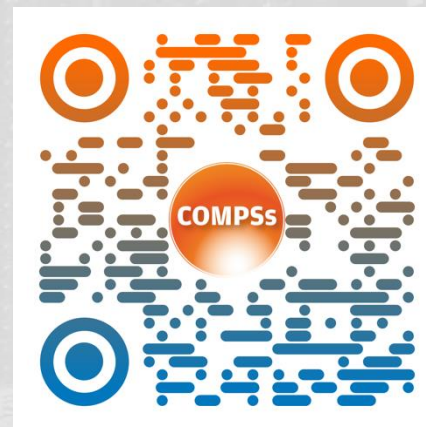
MareNostrum 5





**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

Thanks!



Visit us in booth #3549!

rosa.m.badia@bsc.es