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** - 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** - 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** - 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

	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 + NVIDIA Ampere
MareNostrum 5	Operational	Spain		Sapphire Rapids, NVIDIA Hopper, Grace, Intel Emeralds, Intel
Petascale Meluxina				AMD EPYC + NVIDIA Ampere A100
Vega	Op		10.05 petaflops	AMD Epyc 7H12 + Nvidia A100
Karolina	Ope	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

Centro Nacional de Supercomputación

Complex problems for complex computing infrastructures

HPC simulation & modeling

Data analytics

COMPLEX PROBLEMS

Artificial intelligence

Performance Development

TOP 500 The List.

- Sum
- #1
- #500
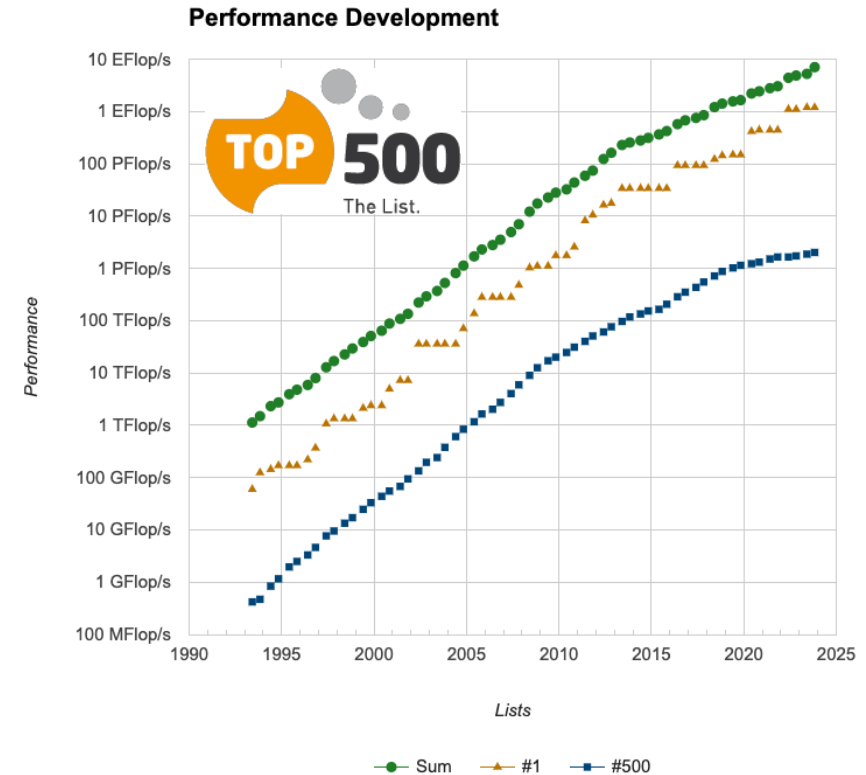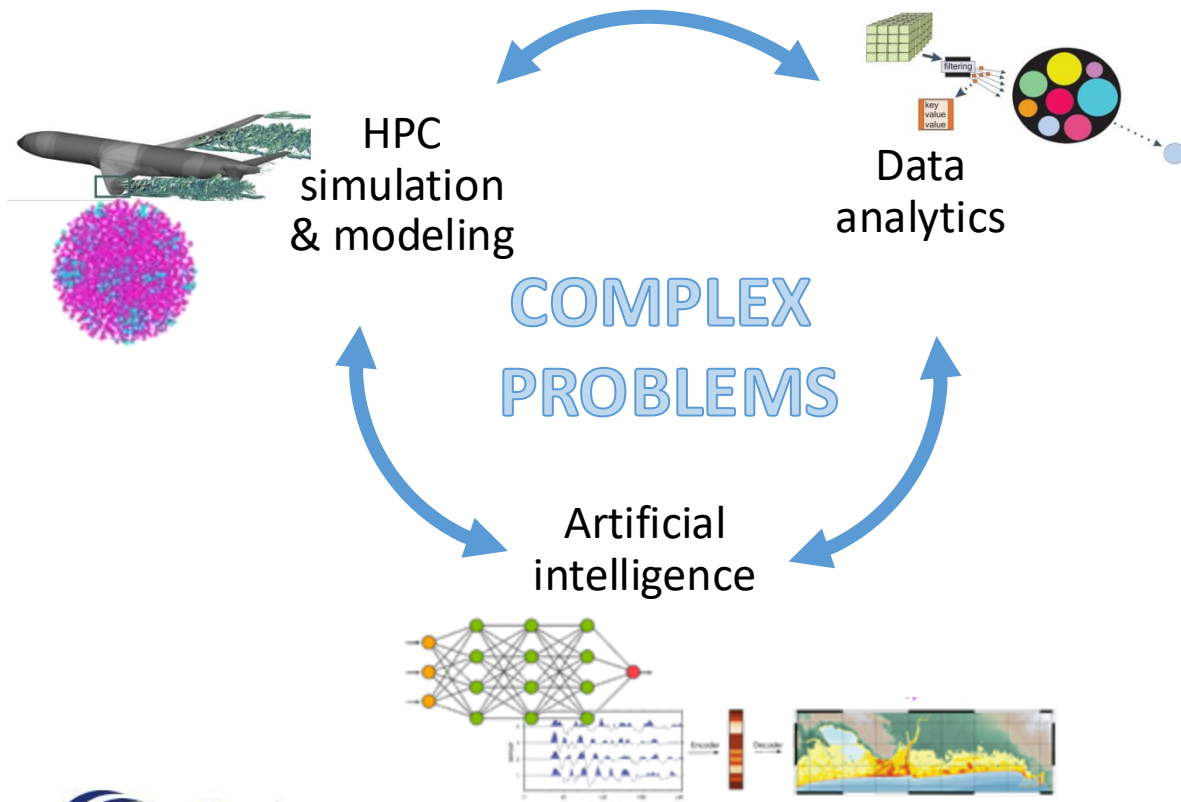
Digital continuum
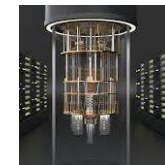
Workflow lifecyle challenges

- 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

- Sample projects:
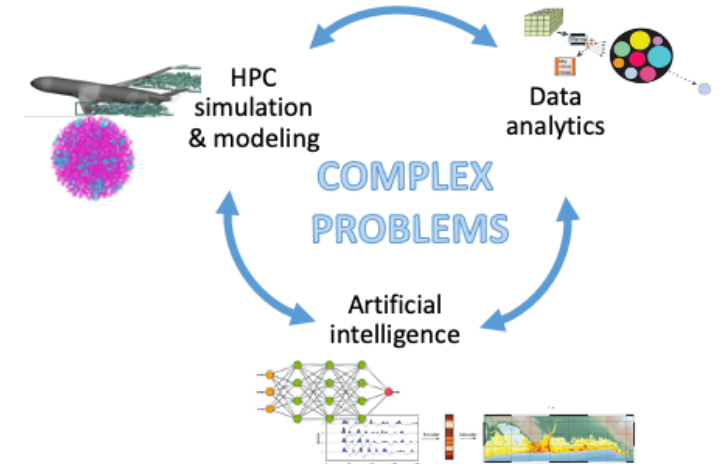
eFlows4HPC in a nutshell

- 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
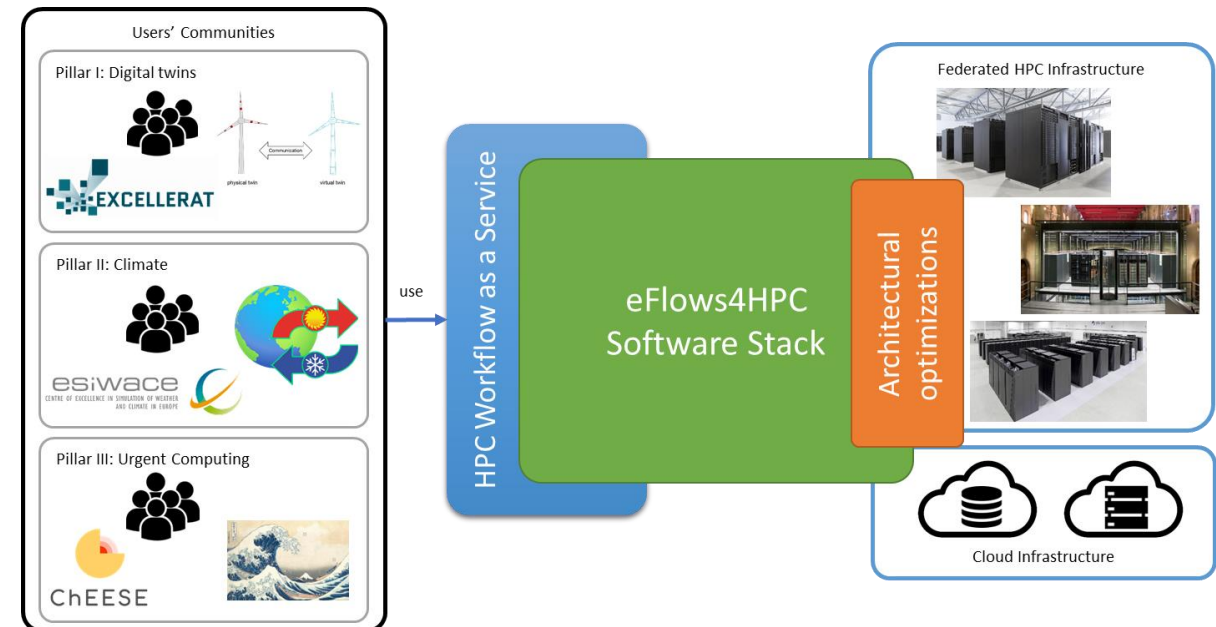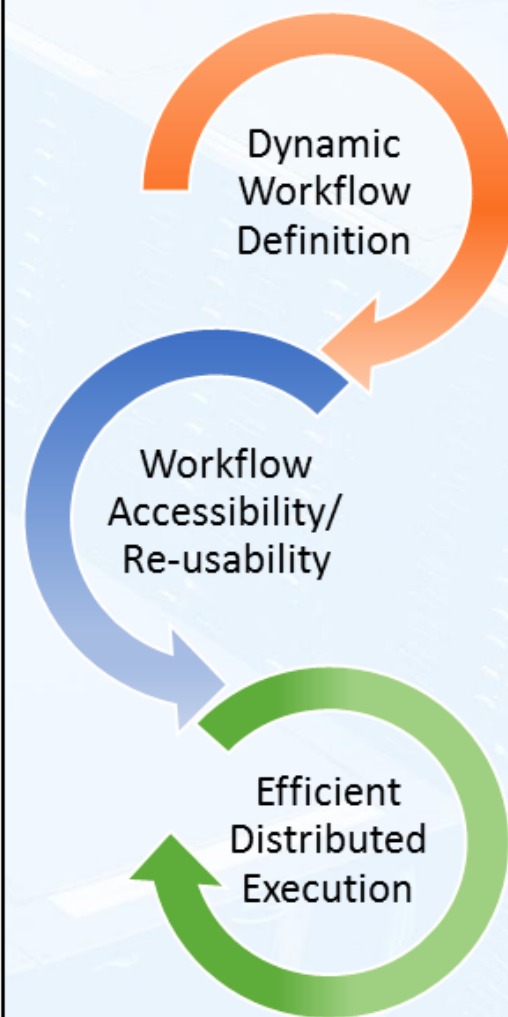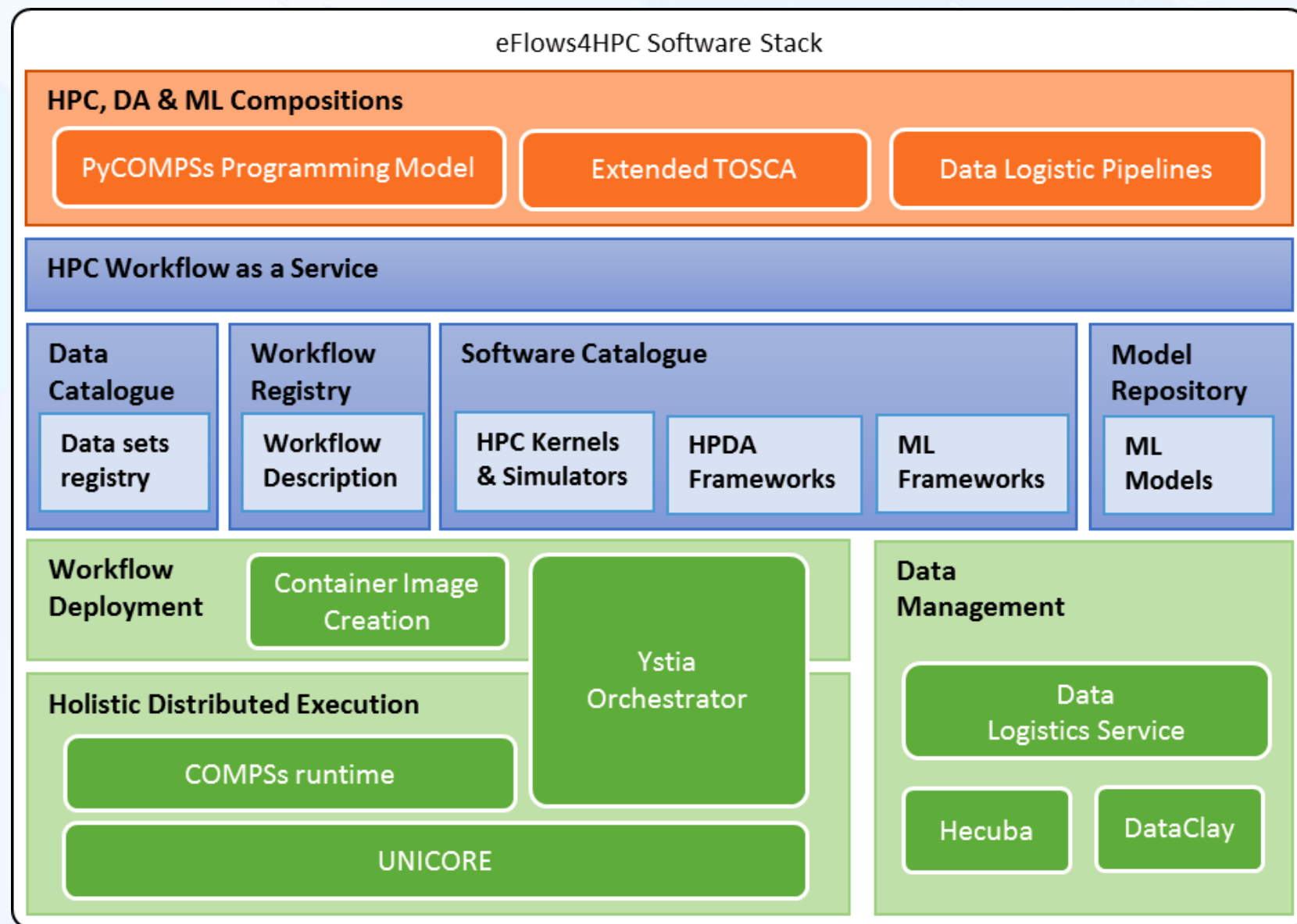
HPCWaaS: Workflow lifecycle overview

eFlows4HPC

eFlows4HPC Gateway Services

Data Catalog

Software Catalog

Workflow Registry ← → **Alien4Cloud**

Dynamic Workflow Description

TOSCA Description

Data Logistics Pipelines

PyCOMPSs Code

2. Store

3. Deploy

1. Create Workflow

4. Share

Endpoint to invoke the Workflow
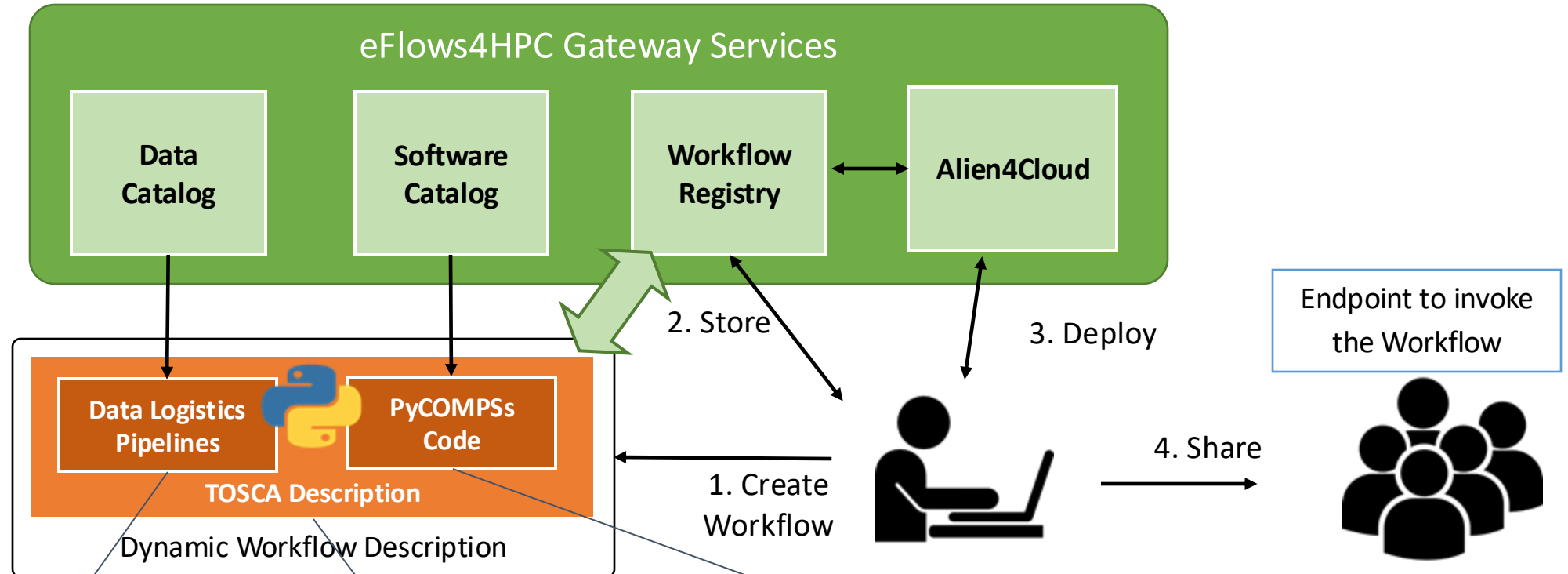
Description of data movements as Python functions. Input/output datasets described at Data Catalog

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

DT-GEO blueprint architecture

DTC accessibility — DTC deployment — DTC execution

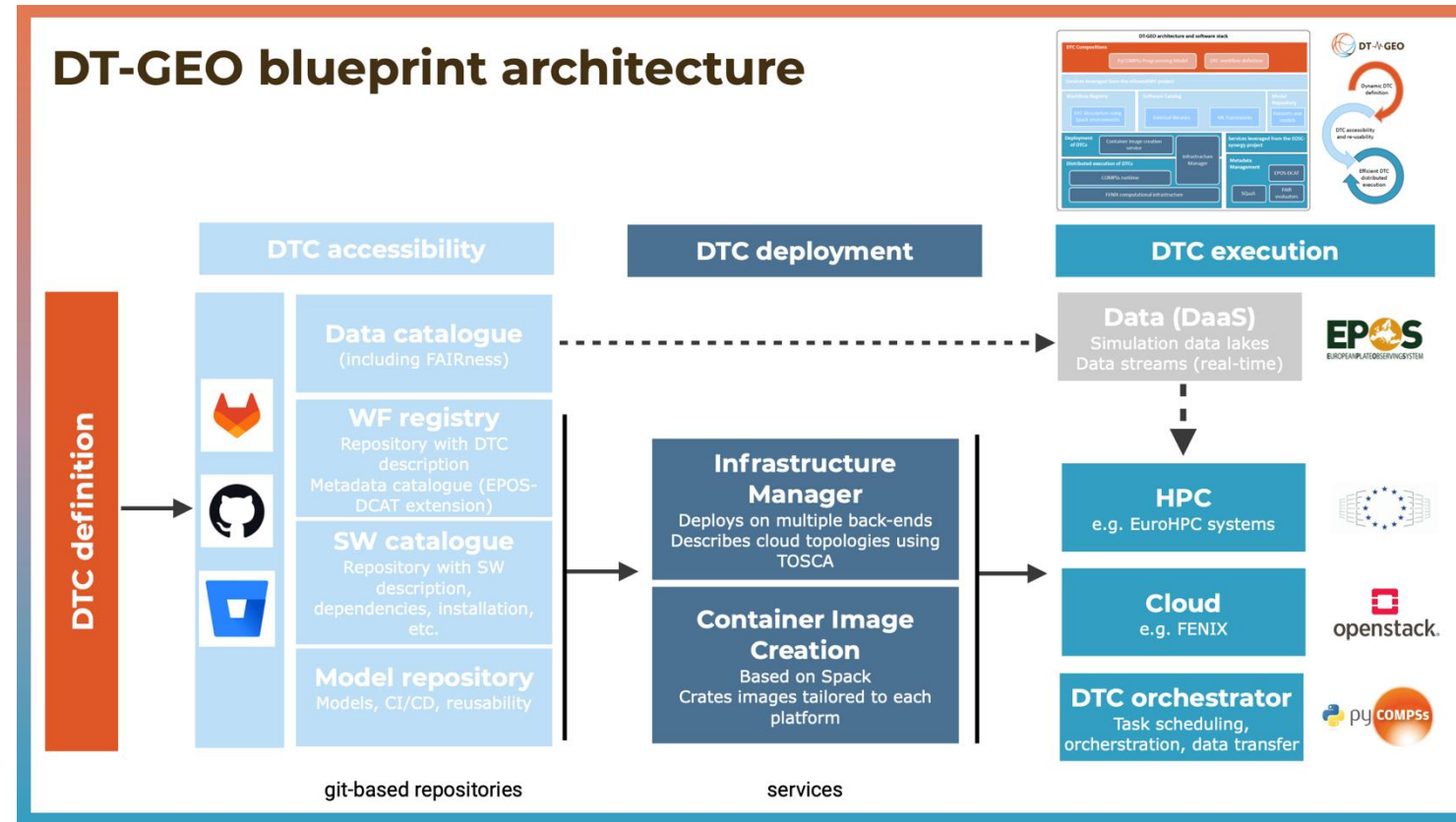DTC definition

Data catalogue (including FAIRness)

WF registry
Repository with DTC description
Metadata catalogue (EPOS-DCAT extension)

SW catalogue
Repository with SW description, dependencies, installation, etc.

Model repository
Models, CI/CD, reusability

Infrastructure Manager
Deploys on multiple back-ends
Describes cloud topologies using TOSCA

Container Image Creation
Based on Spack
Crates images tailored to each platform

Data (DaaS)
Simulation data lakes
Data streams (real-time)

EPOS
EUROPEAN PLATE OBSERVING SYSTEM

HPC
e.g. EuroHPC systems

Cloud
e.g. FENIX

openstack

DTC orchestrator
Task scheduling, orchestration, data transfer

py COMPSs

git-based repositories   services

CAELESTIS Simulation Ecosystem Architecture
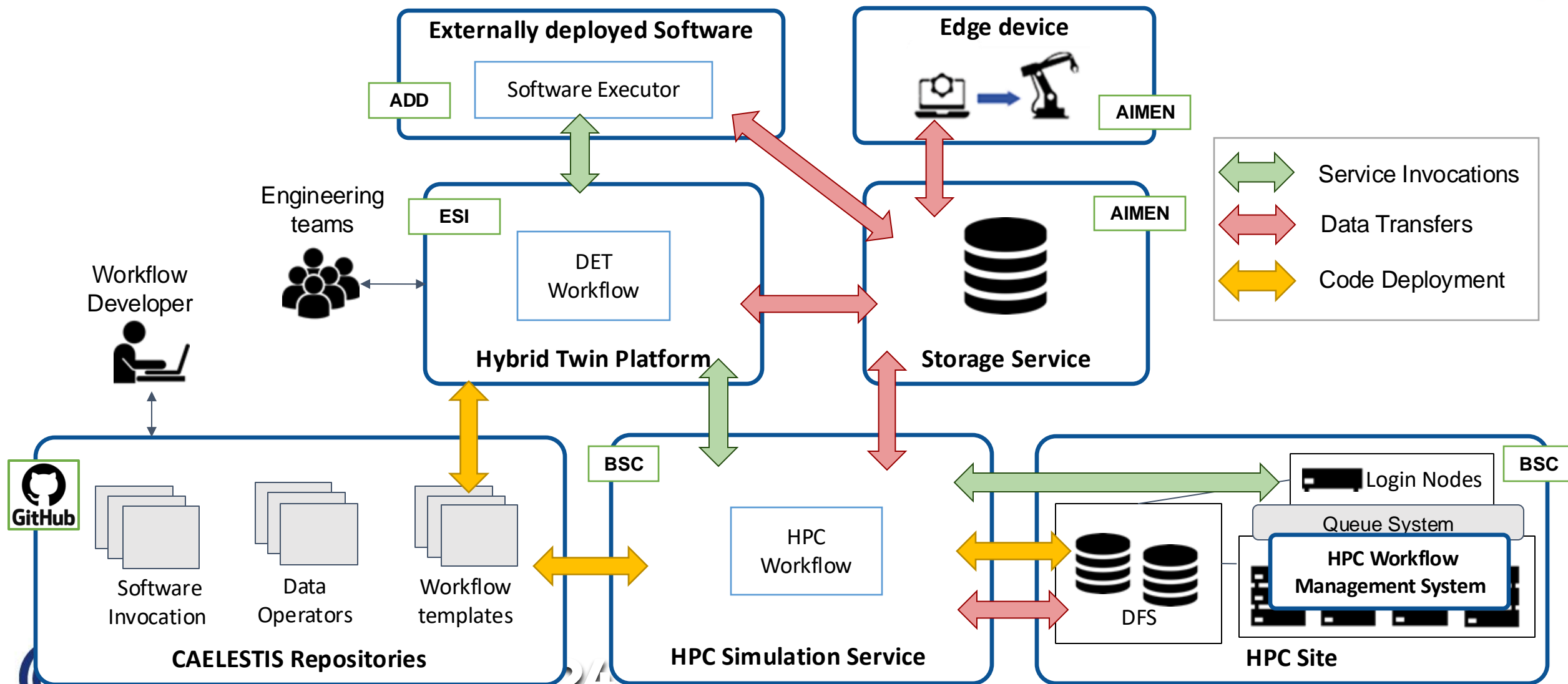
Externally deployed Software

ADD

Software Executor

Edge device

AIMEN

Engineering teams

ESI

DET Workflow

AIMEN

Workflow Developer

Hybrid Twin Platform

Storage Service

Service Invocations

Data Transfers

Code Deployment

GitHub

Software Invocation

Data Operators

Workflow templates

BSC

HPC Workflow

HPC Simulation Service

Login Nodes

BSC

Queue System

HPC Workflow Management System

DFS

HPC Site

CAELESTIS Repositories

Center
Centro Nacional de Supercomputación

A workflow example in the hands-on!
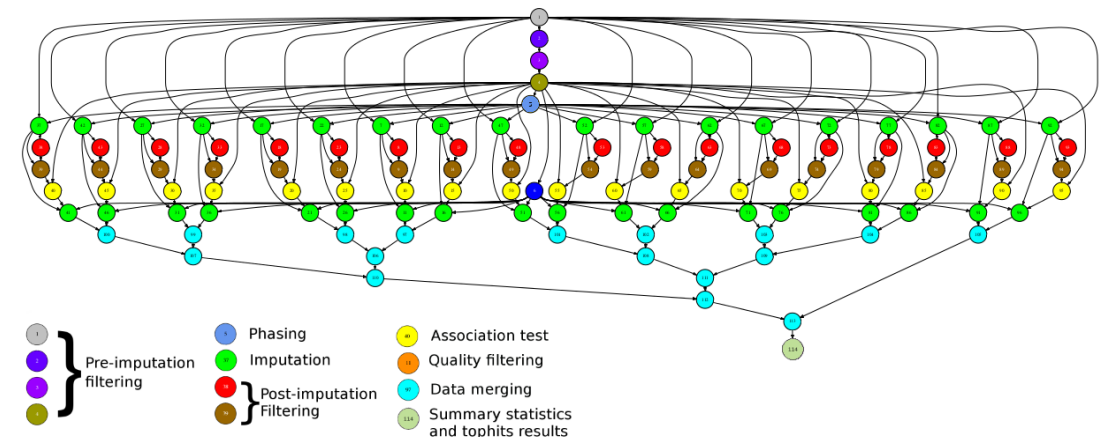
14

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
```


Pre-imputation filtering
Phasing
Association test
Imputation
Quality filtering
Post-imputation Filtering
Data merging
Summary statistics and tophits results

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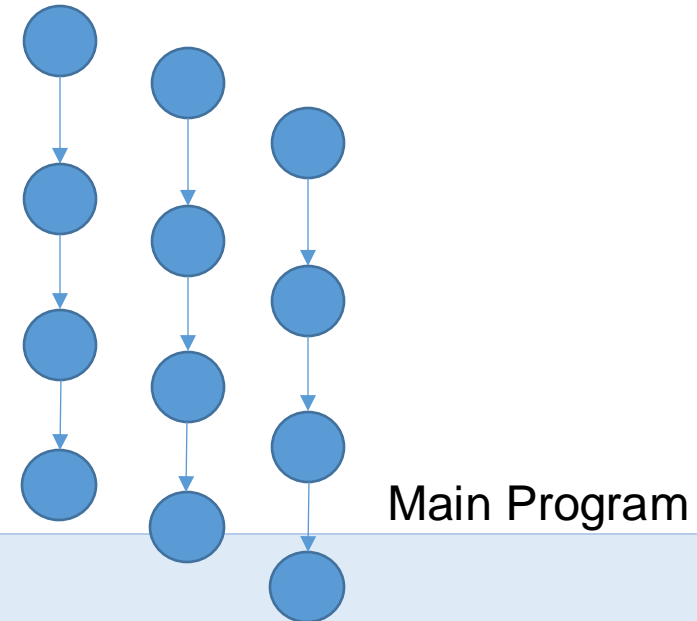
```
initialize_variables()
startMulTime = time.time()
for i in range(MSIZE):
    for j in range(MSIZE):
        for k in range(MSIZE):
            multiply (A[i][k], B[k][j], C[i][j])
compss_barrier()
mulTime = time.time() - startMulTime
```

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()
```
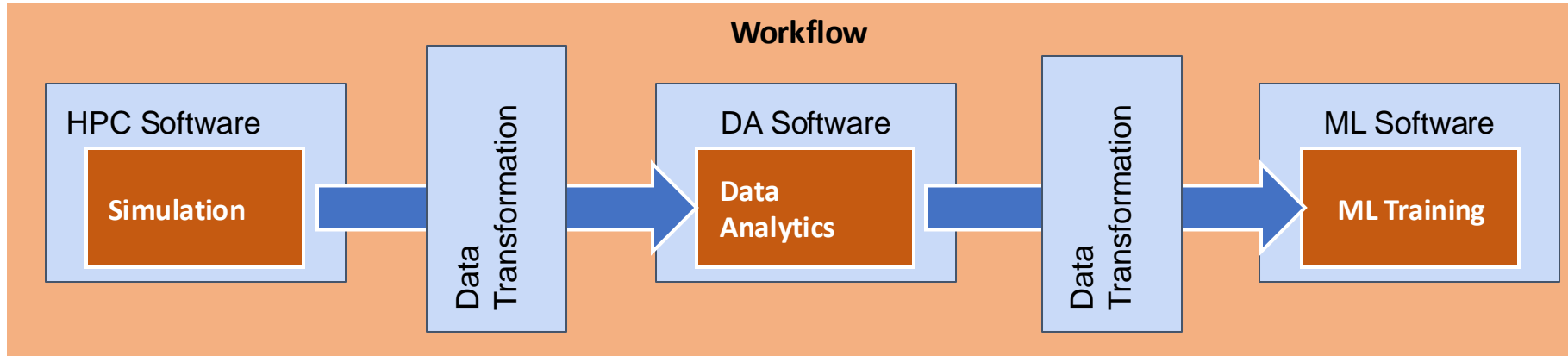
PyCOMPSs runtime

- 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

*https://compss-doc.readthedocs.io/
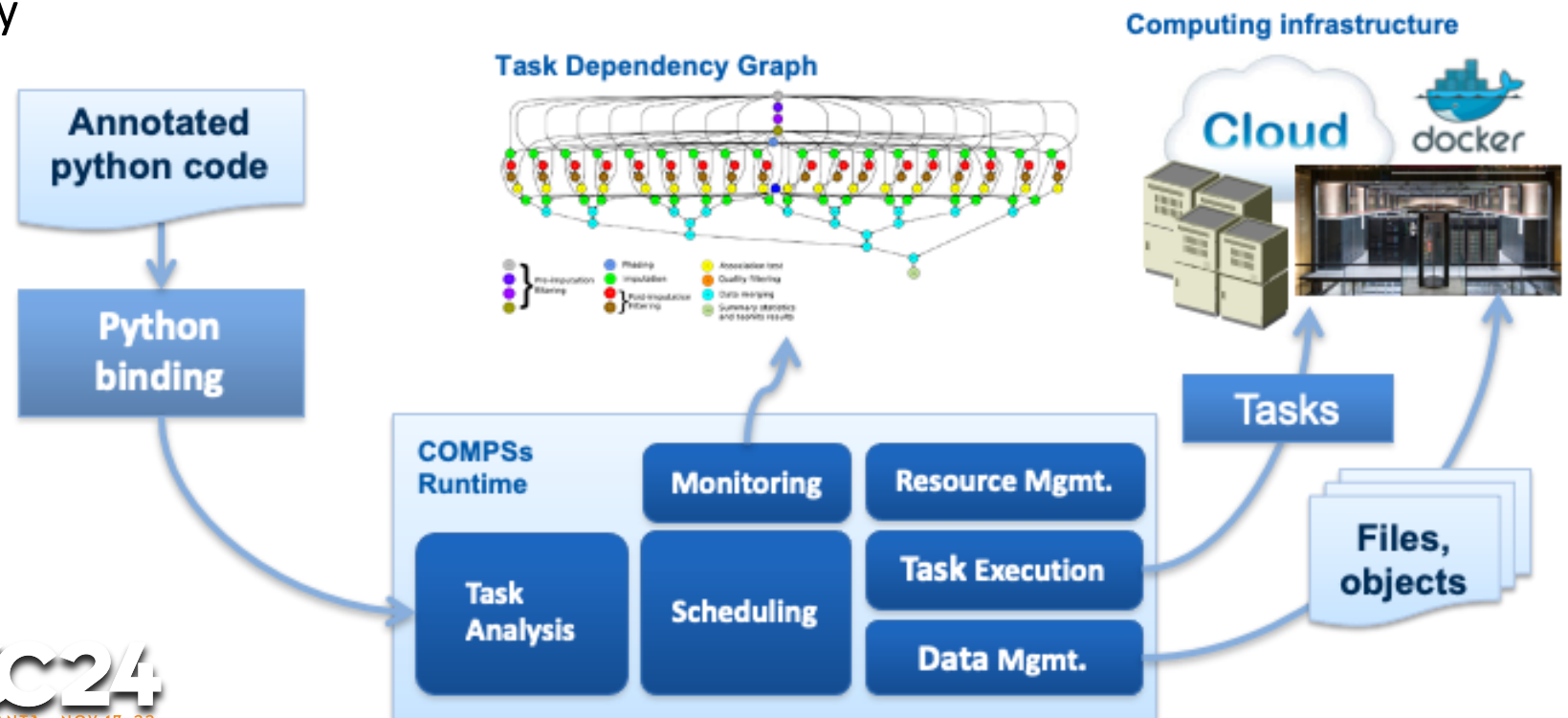
Interfaces to integrate HPC/DA/ML

eFlows4HPC

Workflow

HPC Software		DA Software		ML Software
Simulation	Data Transformation	**Data Analytics**	Data Transformation	**ML Training**

workflow steps
defined as tasks

- 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:
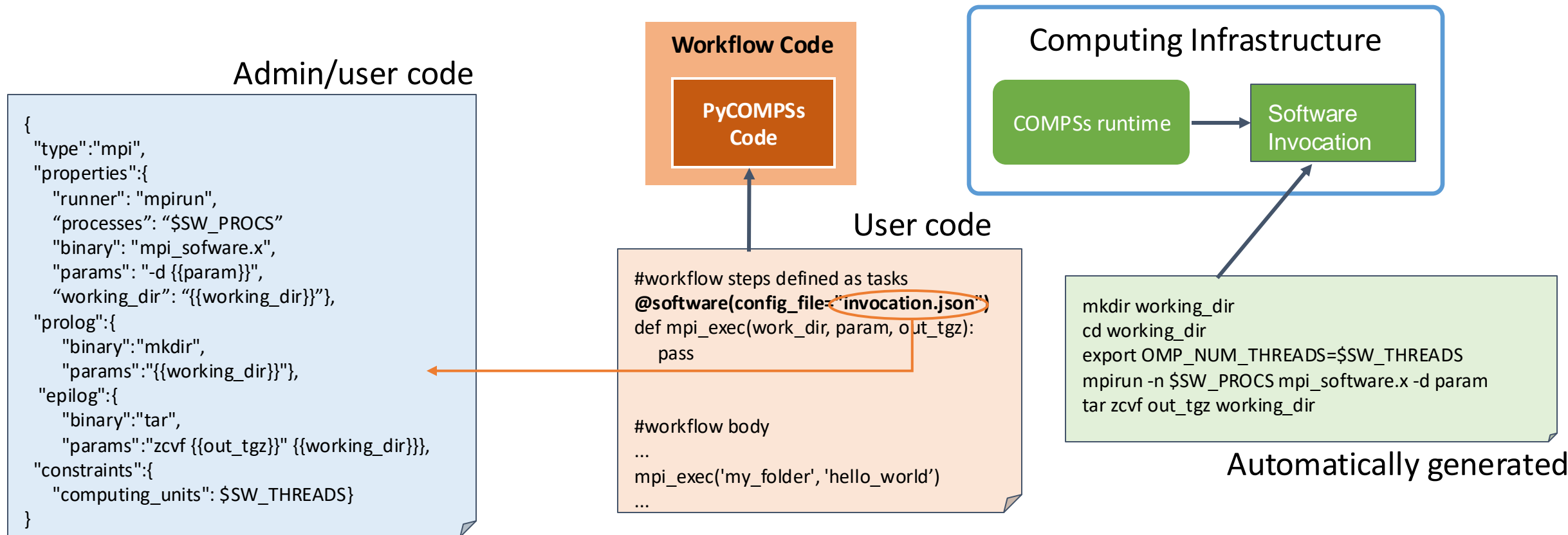 - Software invocation
  - Data transformations

workflow body

```
@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)
```

SC24
ATLANTA NOV 17-22

Software Invocation description

eFlows4HPC

Admin/user code

```
{
  "type":"mpi",
  "properties":{
    "runner": "mpirun",
    "processes": "$SW_PROCS"
    "binary": "mpi_sofware.x",
    "params": "-d {{param}}",
    "working_dir": "{{working_dir}}"},
  "prolog":{
    "binary":"mkdir",
    "params":"{{working_dir}}"},
  "epilog":{
    "binary":"tar",
    "params":"zcvf {{out_tgz}}" {{working_dir}}},
  "constraints":{
    "computing_units": $SW_THREADS}
}
```

Software invocation description
Stored in software catalog

Workflow Code

PyCOMPSs Code

Computing Infrastructure

COMPSs runtime → Software Invocation

User code

```
#workflow steps defined as tasks
@software(config_file="invocation.json")
def mpi_exec(work_dir, param, out_tgz):
    pass


#workflow body
...
mpi_exec('my_folder', 'hello_world')
...
```

```
mkdir working_dir
cd working_dir
export OMP_NUM_THREADS=$SW_THREADS
mpirun -n $SW_PROCS mpi_software.x -d param
tar zcvf out_tgz working_dir
```

Automatically generated

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

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

SC24 ATLANTA NOV 17-22

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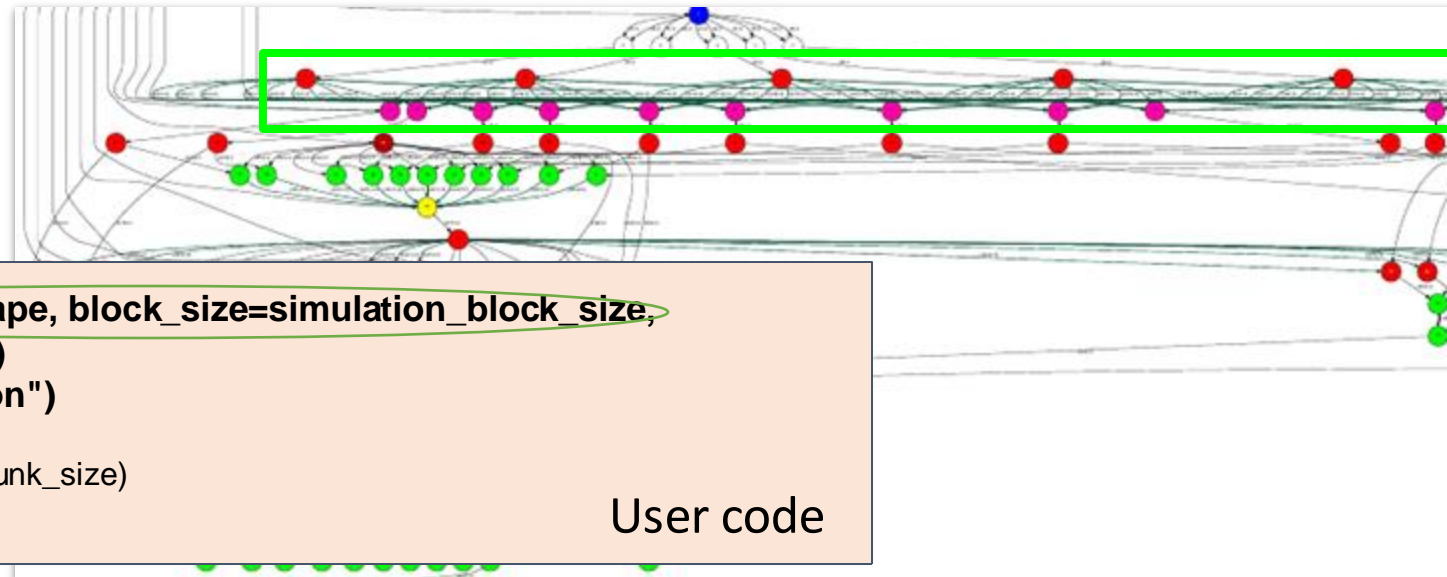
```
@dt("blocks", load_blocks_rechunk, shape=expected_shape, block_size=simulation_block_size,
    new_block_size=desired_block_size, is_workflow=True)
@software(config_file = SW_CATALOG + "/dislib/dislib.json")
def rSVD(blocks, desired_rank=30):
    u,s = rsvd(blocks, desired_rank, A_row_chunk_size,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

User code

dt

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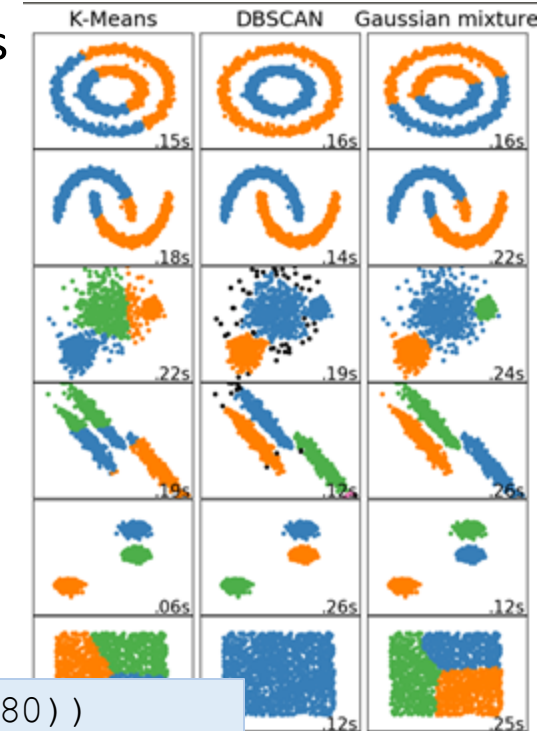
```
x = load_txt_file("train.csv", (10, 780))
x_test = load_txt_file("test.csv", (10, 780))

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

features

block

samples
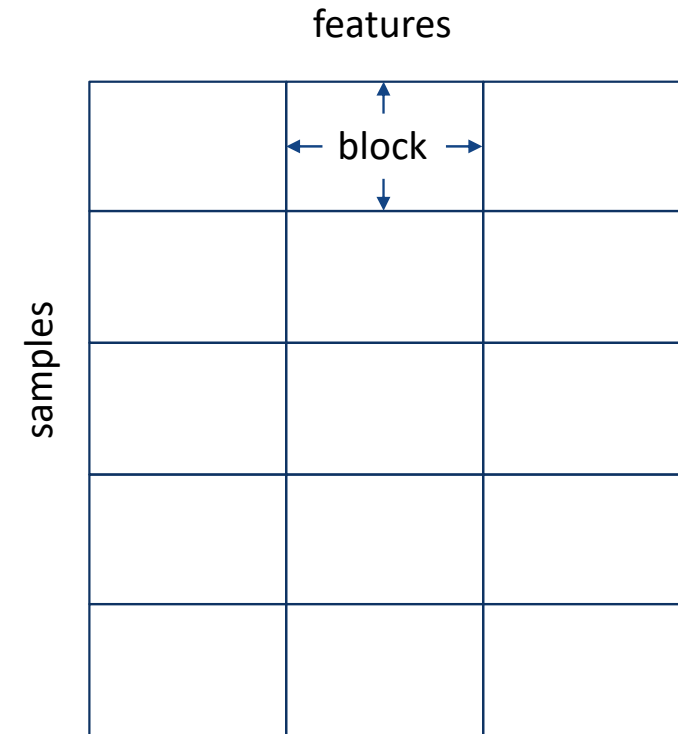
Internally parallelized with PyCOMPSs

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


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

x: ds-array

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

Reduces values of centers through merge task

```python
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
```
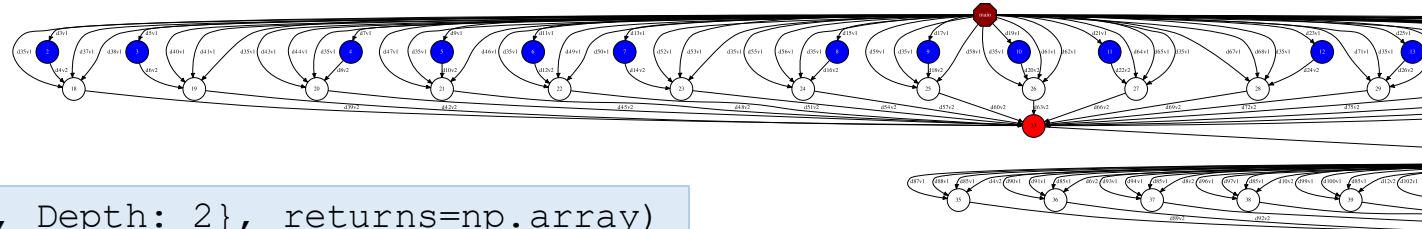
Sample workflows

- 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
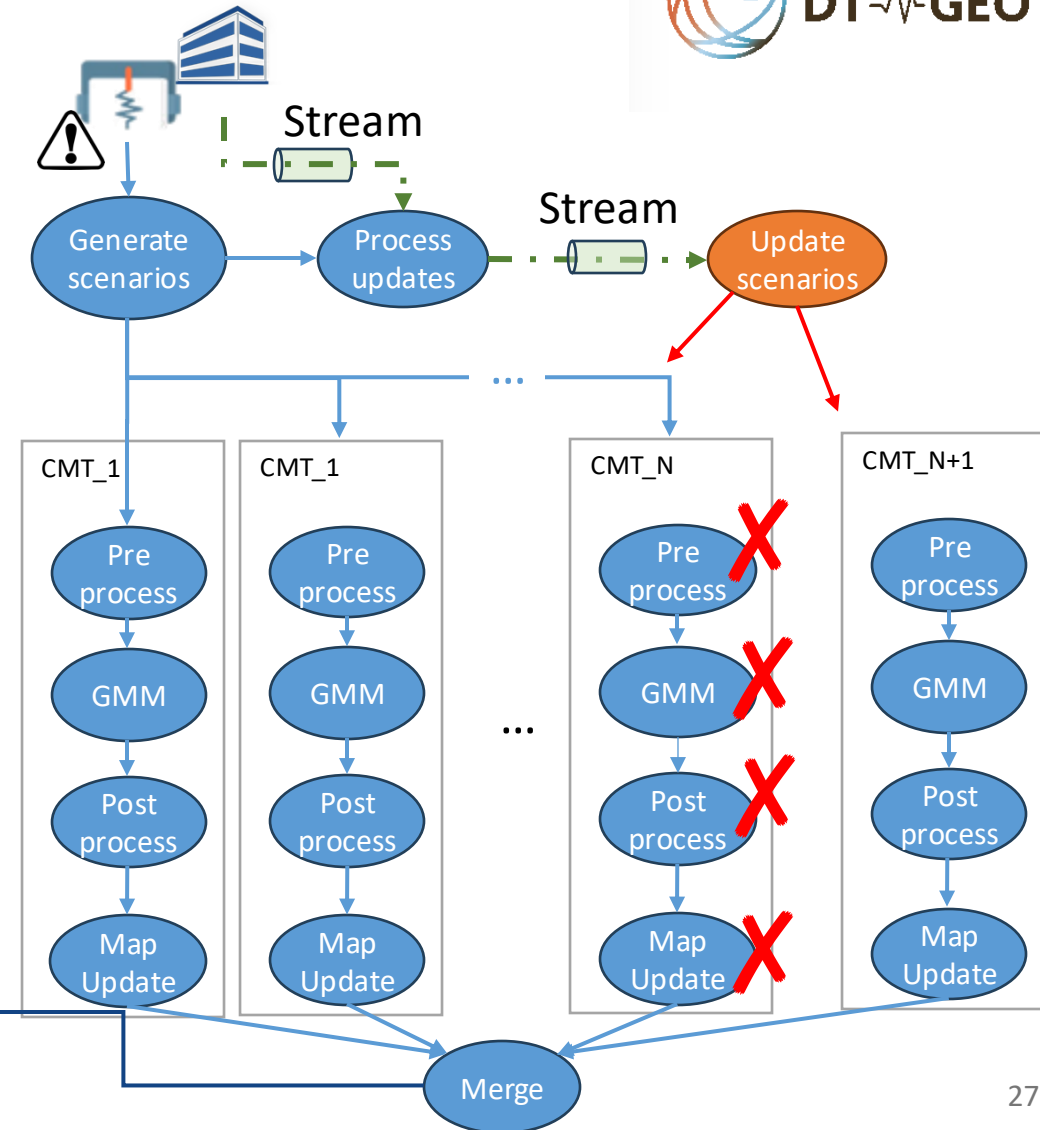
DTC Volcanoes

- 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
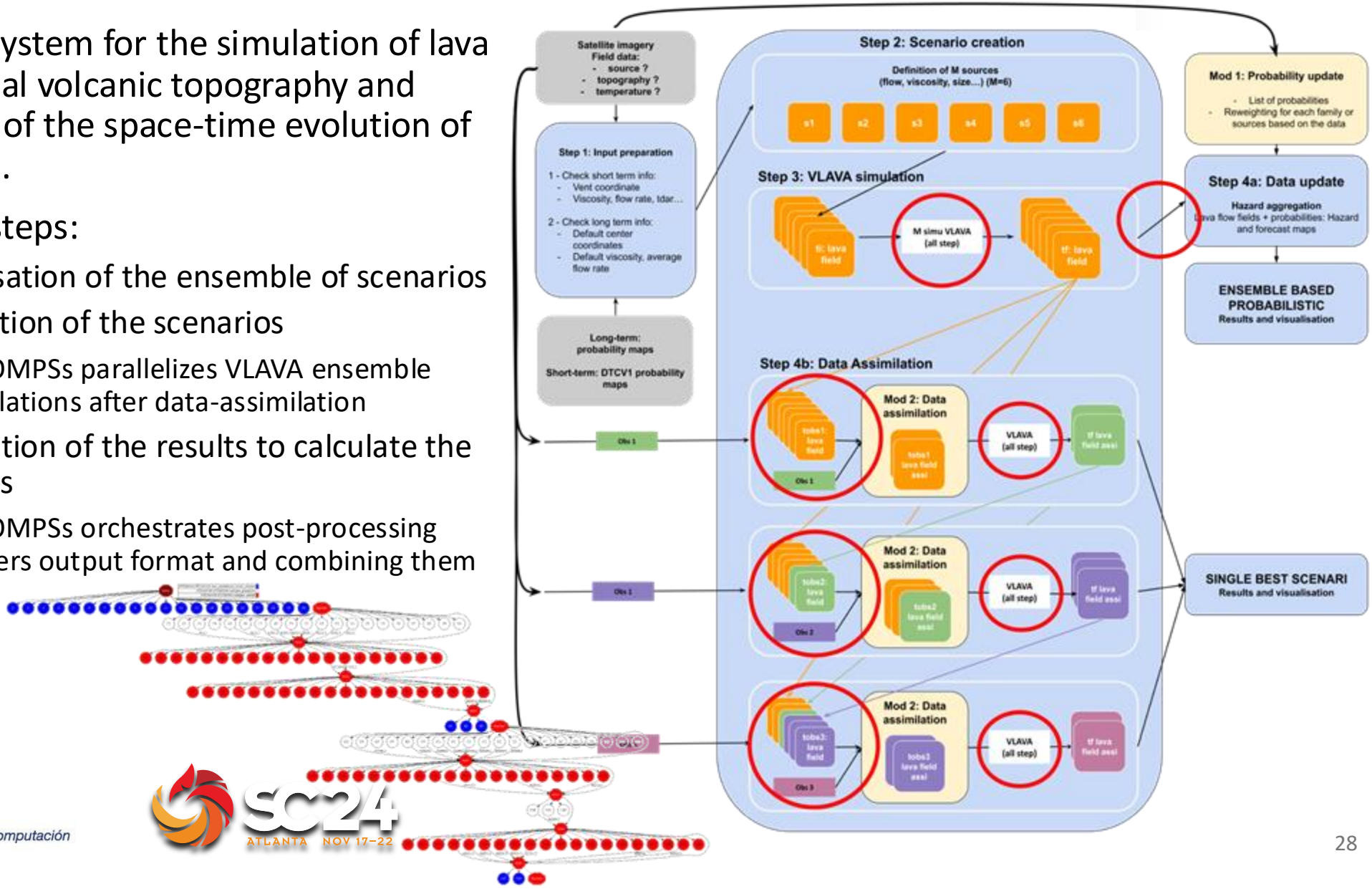
Workflow templates

Surrogate Model Creation Workflow
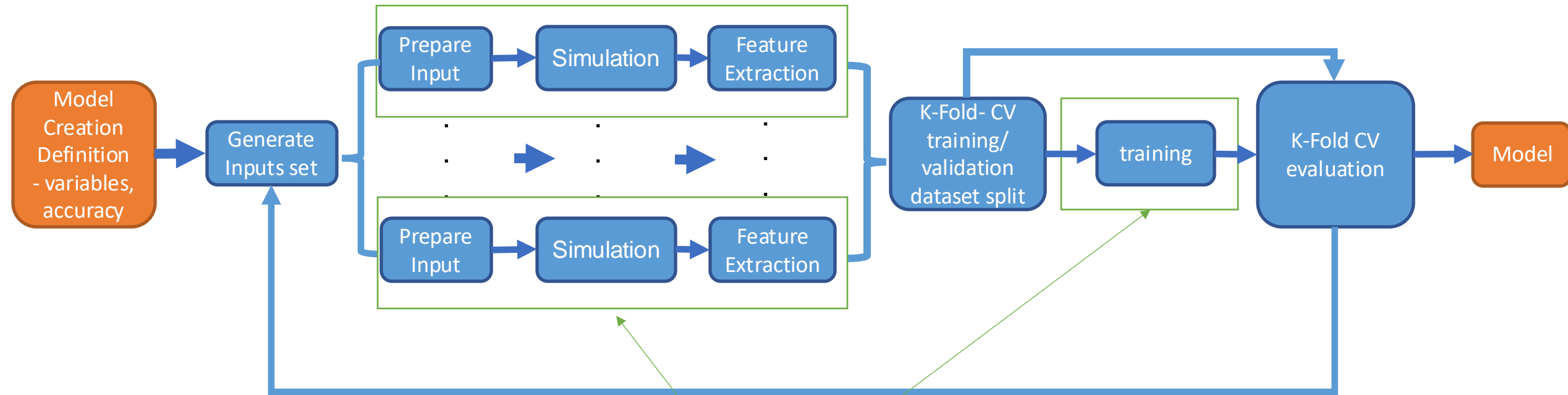
Customized for each the model

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 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
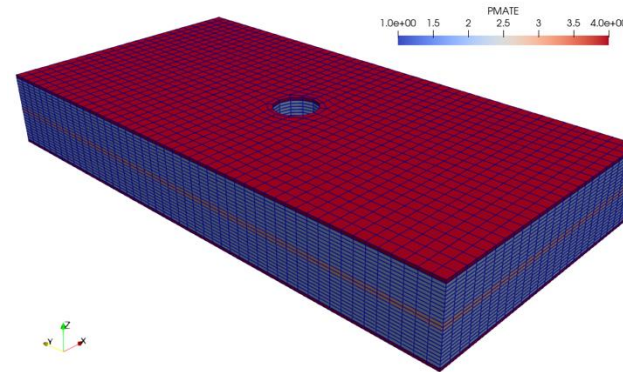
Specific workflow instance
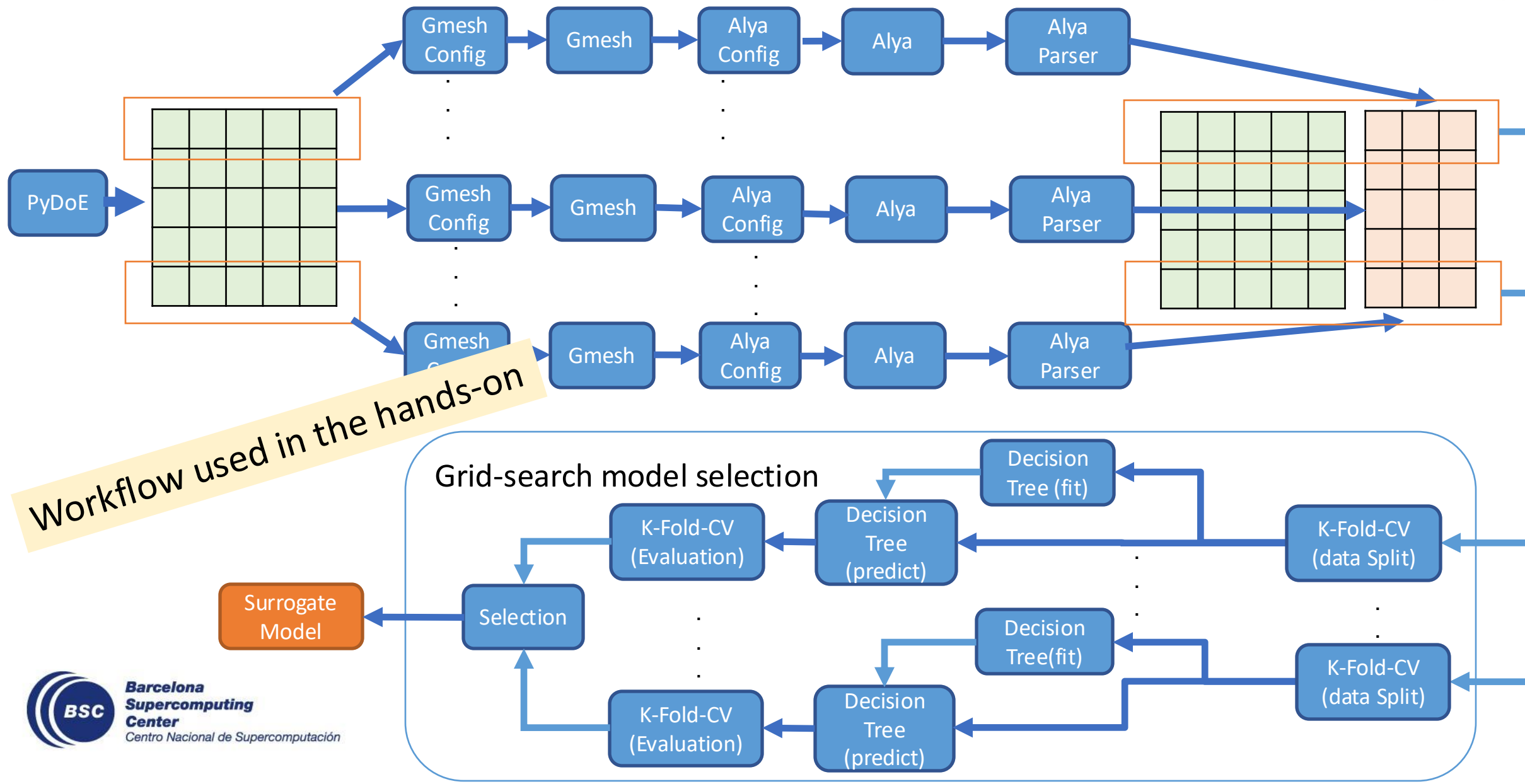
Further Information

- 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

MareNostrum 5

rosa.m.badia@bsc.es