Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

PyCOMPSs introduction

Rosa M. Badia, Javier Conejero

Workflows & Distributed Computing Group

21/03/2022

Barcelona

Outline

Day 1

12:30 – 13:00 PyCOMPSs introduction I

- Overview
- Hands-on example 1

12:30 – 13:00 PyCOMPSs introduction II

- Hands-on example 2
- Hands-on example 3
- Conclusion

SLIDES

- https://tinyurl.com/2dxac2uw

PYCOMPSS OVERVIEW

Motivation

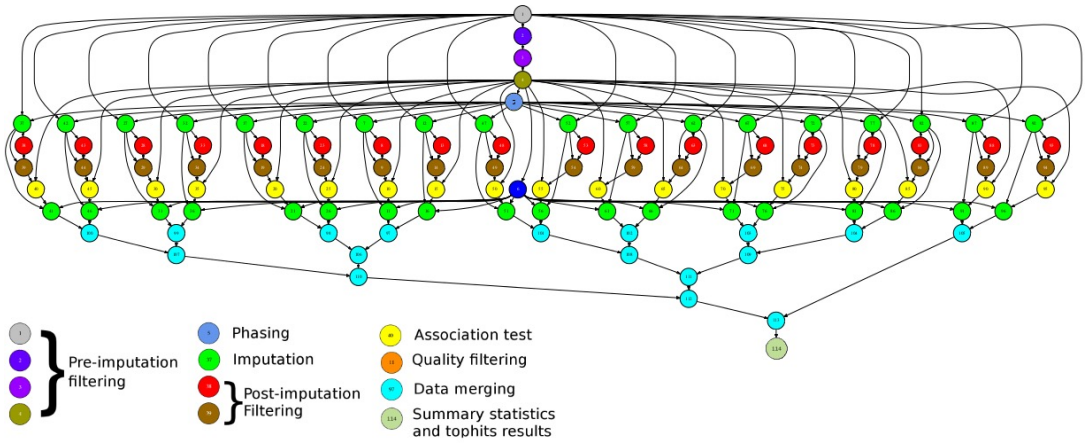
- Create tools that make developers' life **easier**
  - Allow developers to focus on their problem
  - Provide the intermediate layer logic:
    - Act on behalf of the user
    - Automatic parallelization
    - Distribute the work through compute resources
    - Automatic data transfers
    - Deal with architecture specifics
    - Automatically improve performance
  - Tools for visualization
    - Monitoring
    - Performance analysis
  - Integration of computational workloads, with machine learning and data analytics

Goal of this lesson

- Give an overview of PyCOMPSs
- Exemplify the use of PyCOMPSs as a tool to bundle multiple jobs into one in MareNostrum 4
- More information about PyCOMPSs can be find in our website, including other larger tutorials: compss.bsc.es

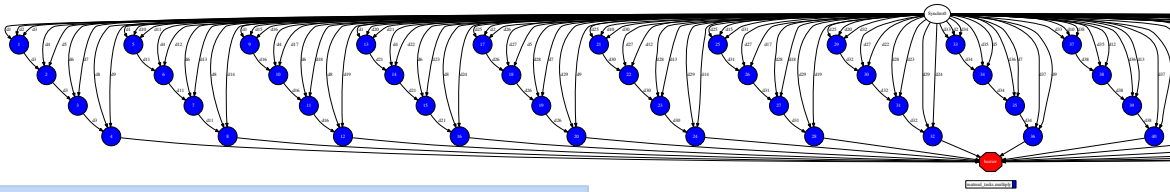
Programming with PyCOMPSs

- Sequential programming, parallel execution
- General purpose programming language + annotations/hints
  - To identify tasks and directionality of data
- Task based: task is the unit of parallelism
- Builds a task graph at runtime that express potential concurrency
  - And data dependencies
- Offers a shared memory illusion to applications in a distributed system
  - The application can address larger data storage space: support for Big Data apps
- Agnostic of computing platform
  - Enabled by the runtime for clusters, clouds and container managed clusters



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
```

Binary decorator

- Tasks can describe its behaviour as a Python method
- Or can invoke an external binary:

```
@binary(binary="mybinary.bin")
@task()
def binary_func():
     pass
```

  - The name of the binary is indicated in the decorator
  - Body of the task is empty
- PyCOMPSs generates at runtime the necessary wrapper code to invoke the binary

Binary decorator: parallel tasks

- Binary tasks can use more than 1 core
  - OpenMP, threaded, etc
- The number of threads (cores) to use are indicated with a constraint:

```
@contraint(computing_units=”8")
@binary(binary=”parallel_binary.bin")
@task()
def binary_func():
     pass
```

- Maximum value in a MareNostrum 4 node is 48 cores

Binary decorator: memory constraints

- Binary tasks may need a larger amount of memory
- Memory constraints can be indicated as well in the decorators (units = GB)

```
@contraint(memory_size=6.0)
@binary(binary="mem_intensive_binary.bin")
@task()
def binary_func():
     pass
```

- MareNostrum 4 nodes has two memory capacities:
  - Regular nodes: 96 GB (average 2GB per core)
  - High-memory nodes: 384 GB (average 8GB per core)

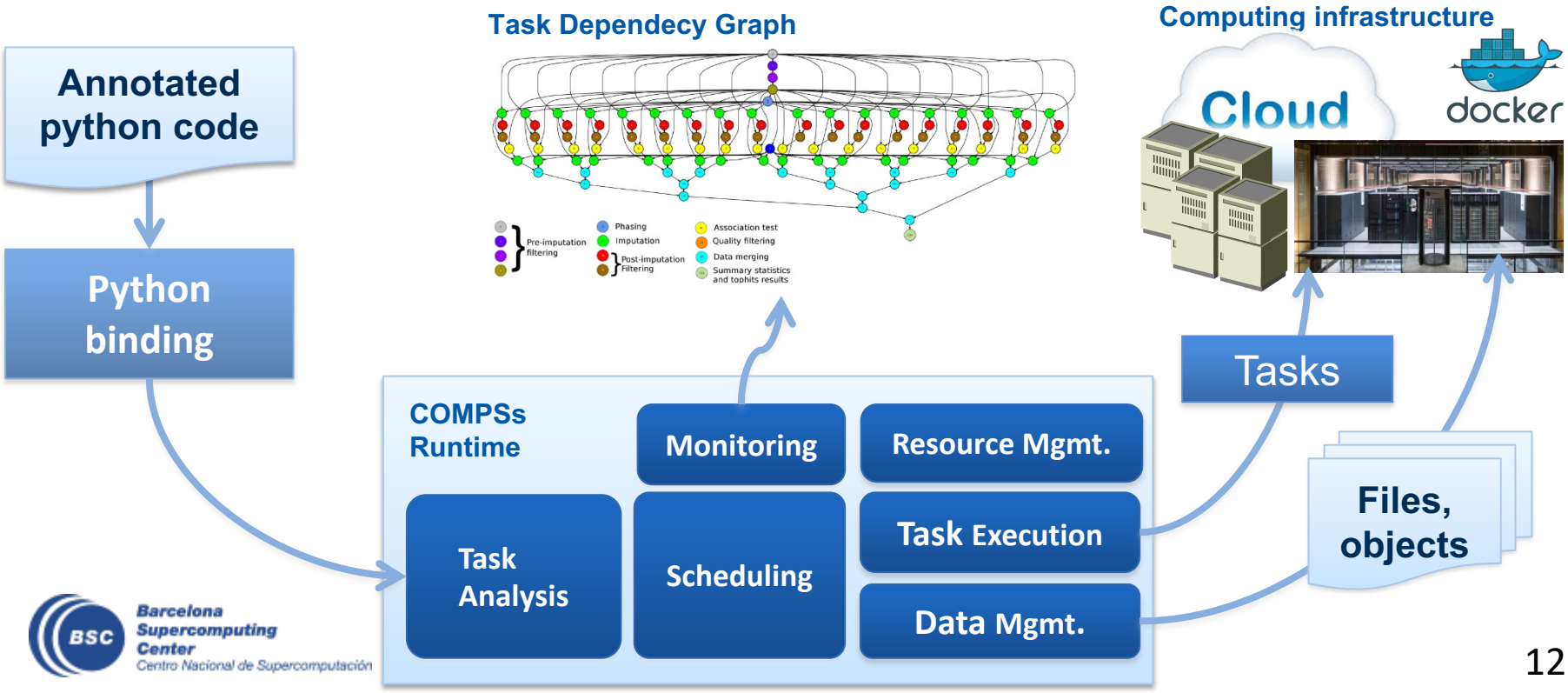
MPI decorator

- When the external binary tasks is an MPI application spanning multiple nodes

```
@mpi(binary="mpiApp.bin", runner="mpirun", processes=96)
@task()
def mpi_func():
    pass
```

PyCOMPSs runtime in MN4

- PyCOMPSs applications executed as a single job in the queuing system (slurm) using N nodes
- One node acts as master and execute user code and runtime
  - Rest of the nodes acts as workers
  - Part of master node can act as a worker also
- Two options for data management: shared_disk or local (default)

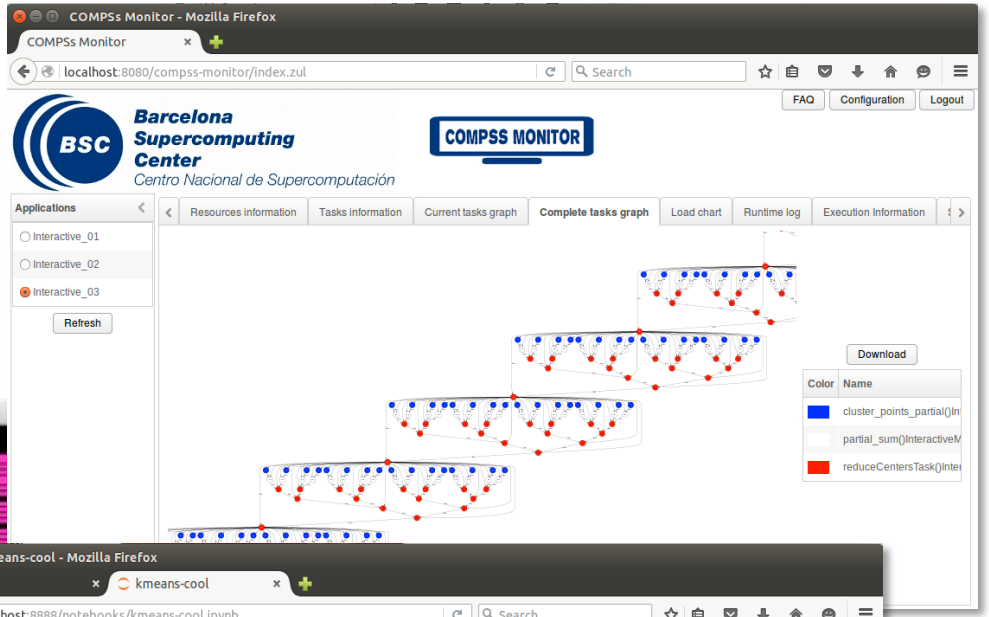


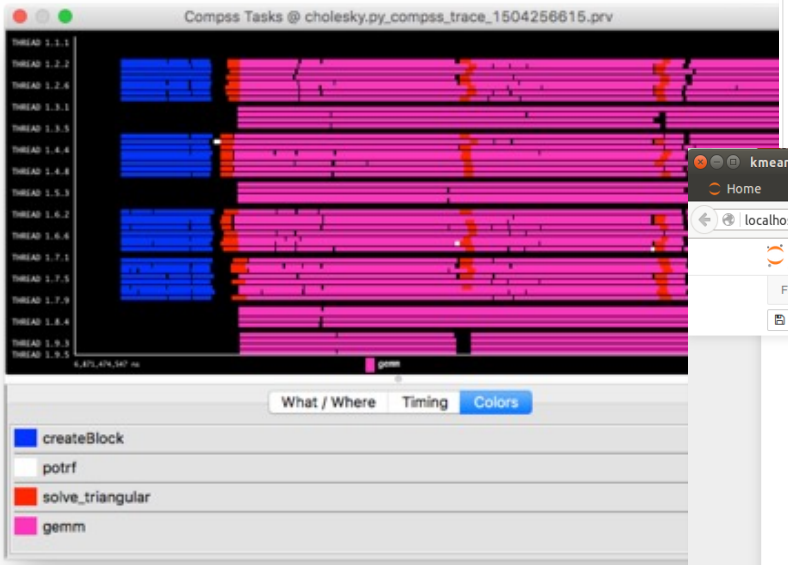
Launching a PyCOMPSs application in MN4

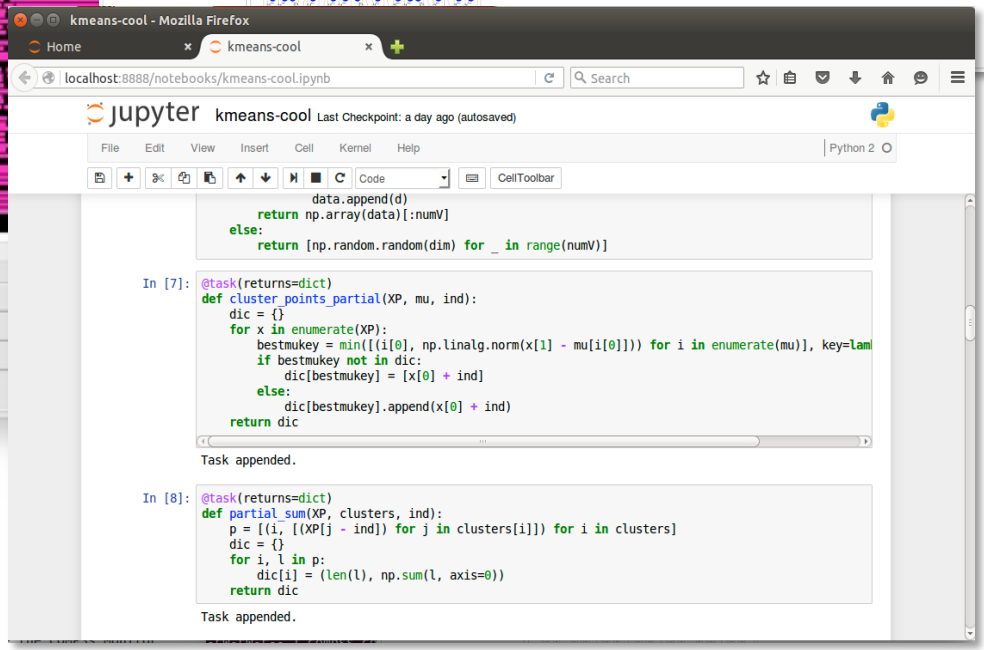
- PyCOMPSs is installed and frequently updated in MareNostrum 4
  - /apps/COMPSs/
- Available as a module (multiple versions, by default will load the latest stable version)
  - module load COMPSs
- PyCOMPSs CLI provides a command line interface for execution:
  - pycompss --help
- Sample command line to launch a PyCOMPSs application:
  - pycompss job submit -g --qos=debug --exec_time=10 --num_nodes=1 grep.py

PyCOMPSs development environment

- Runtime monitor
- Paraver traces
- Jupyter-notebooks integration







Projects where COMPSs is used/developed

eFlows4HPC

LAND SUPPORT

AI SPRINT
AI IN SECURE PRIVACY-PRESERVING COMPUTING CONTINUUM

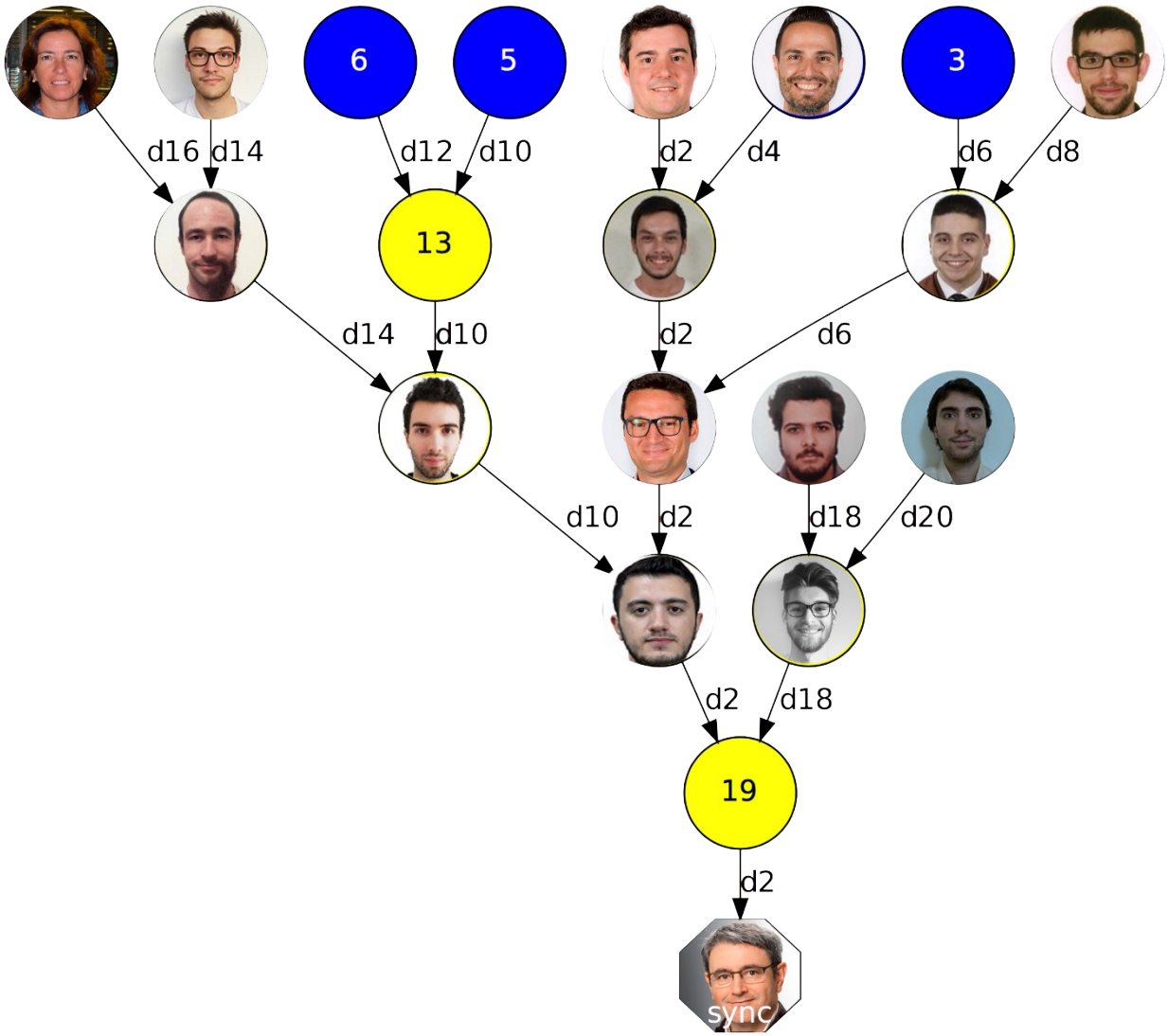
ELASTIC

bioexcel
Center of Excellence for Computational Biomolecular Research

Per Med CoE
HPC/Exascale Centre of Excellence in Personalised Medicine

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

The WDC team



http://compss.bsc.es