Programming with PyCOMPSs: beyond MPI+X

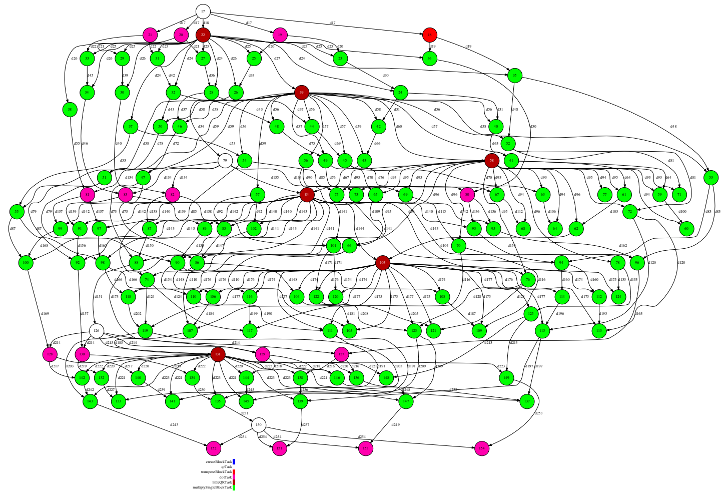
Rosa M. Badia

Workflows & Distributed Computing Group

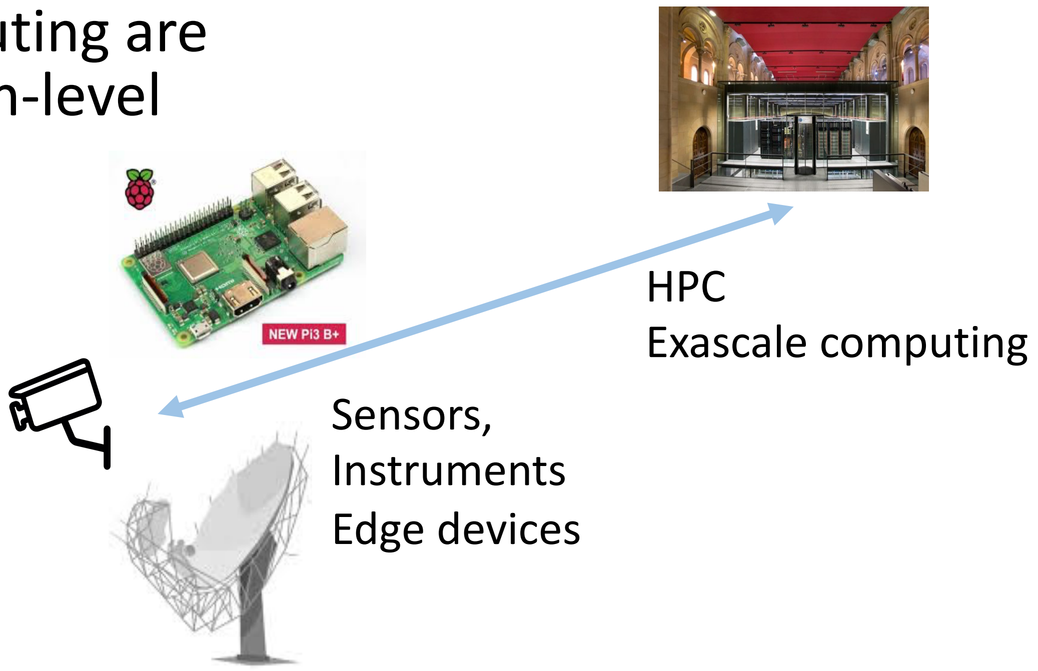
17/11/2019 Denver (CO) USA

Supercomputing 2019 (SC19)

PyCOMPSs/COMPSs ambition

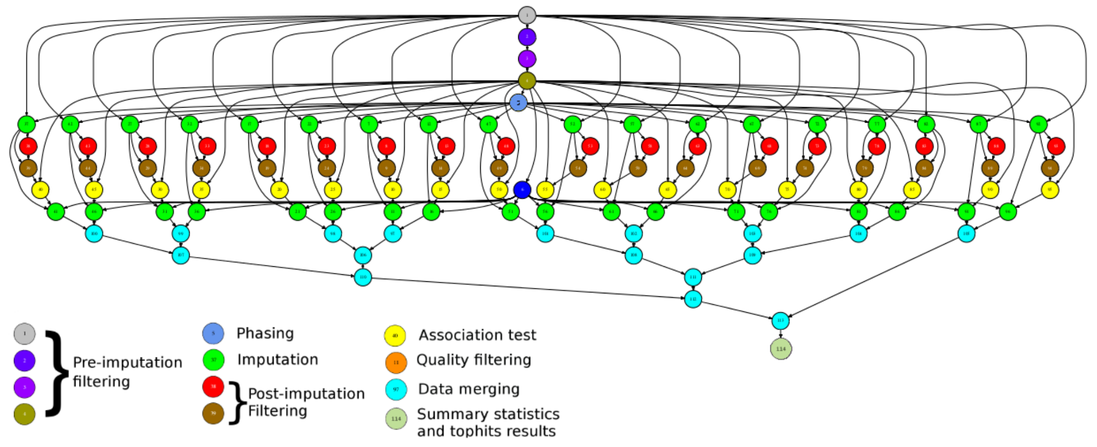


- Complex infrastructures with multiple and heterogeneous components
- Complex applications, composed of multiple components and pieces of software
- How to describe the workflows in such environment?
- Holistic approach where both data and computing are integrated in a single flow built on simple, high-level interfaces
  - Integration of computational workloads, with machine learning and data analytics
  - Intelligent runtime that can make scheduling and allocation, data-transfer, and other decisions



Programming with PyCOMPSs/COMPSs

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



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

What is a PyCOMPSs/COMPSs task?

- Tasks can be sequential, multi-threaded, multi-node
- Tasks may have constraints/requirements
- Tasks may have multiple versions
- Tasks may read/write streamed data

```
@implement (source class="myclass", method="myfunc")
@constraint (MemorySize=1.0, ProcessorType ="ARM")
@task (c=INOUT)
def myfunc_other (a, b, c):
    ...
```

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

Hybrid Task Flows and Data Flows:
new types of pipelines

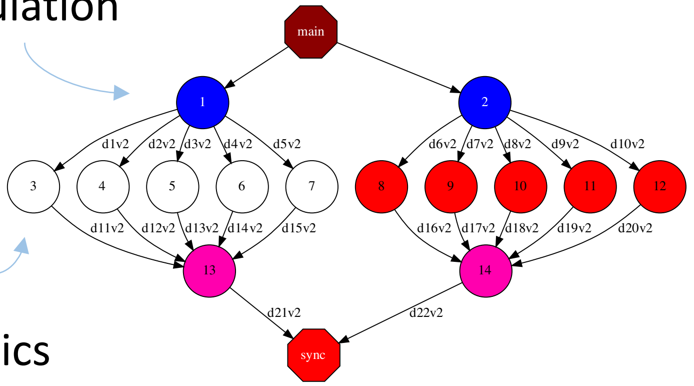
```
@task(fds=STREAM_OUT)
def sensor(fds):
    while not end():
        data = get_data_from_sensor()
        f.write(data)
        f.flush()
    fds.close()
```

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

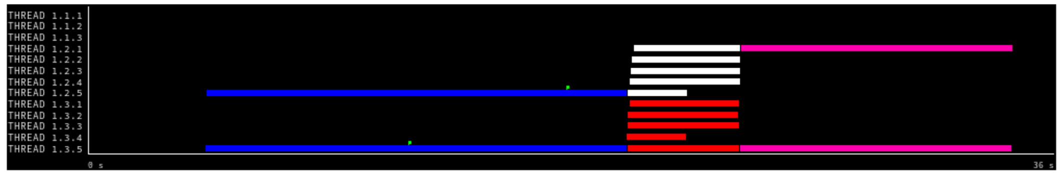
Use case: workflow that includes simulations, analytics and streaming

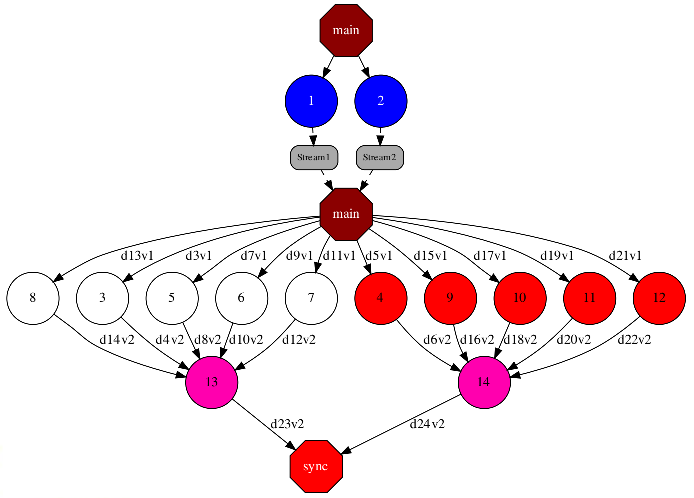
- Sample case: simulations generating at given steps data to be processed by some analytics

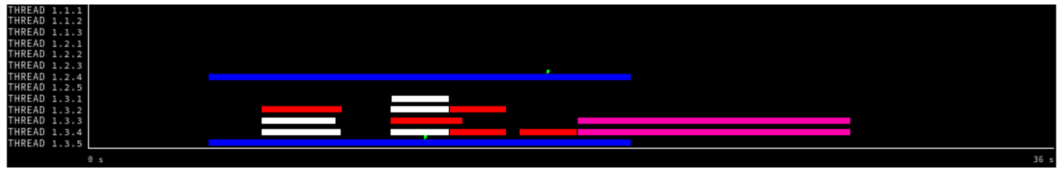
simulation



Analytics

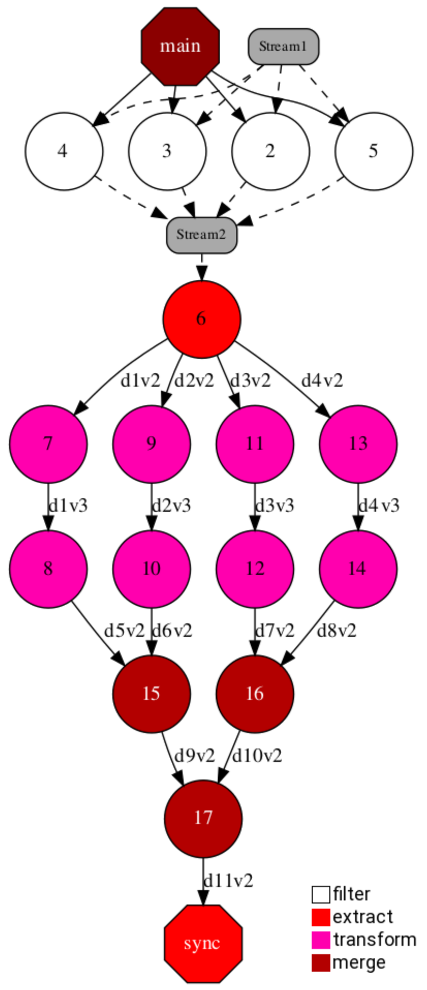






Use case: External streams

- Data produced by external streams (i.e. IoT sensors)
- Multiple cases:
  - Stream data produced by a single task and consumed by many tasks (one to many)
  - Produced by a many tasks and consumed by a single task (many to one)
  - Produced by many tasks and consumed by many tasks (many to many)
- Consumer mode can be configures to process data at least once, at most once, or exactly once
- Hybrid Task-based Workflows and Dataflows
- Data is processed from external streams and the resources can be adapted (elasticity) according to the workload



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PyCOMPSs at SC19

- Tutorial *Practical Persistent Memory Programming,* Monday 8:30
- Demos at the BSC booth (#1975):
  - Tuesday, 14:00 – 14:30: Accelerating parallel code with PyCOMPSs and Numba
  - Wednesday, 13:00 – 13:30: Parallel machine learning with dislib
  - Thursday, 11:00 – 11:30: Fault-tolerance mechanisms for dynamic PyCOMPSs workflows

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Thank you

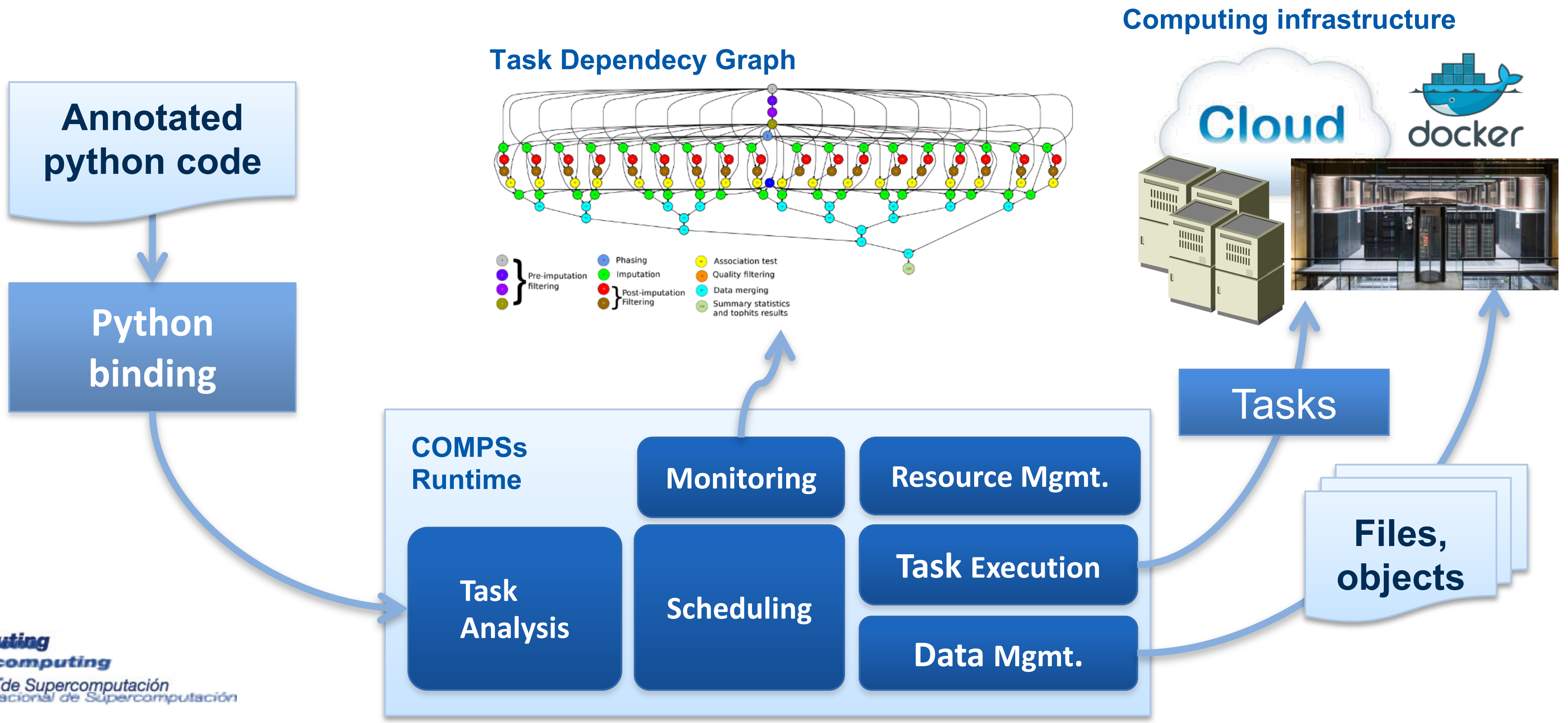
- Booth #1975 -

support-compss@bsc.es

rosa.m.badia@bsc.es

PyCOMPSs/COMPSs runtime

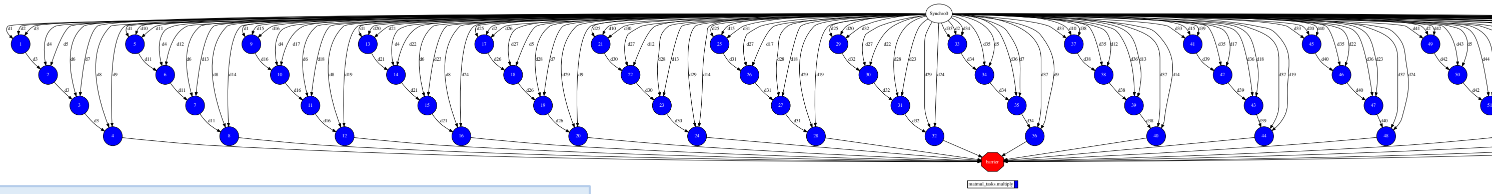
- PyCOMPSs/COMPSs applications executed in distributed mode following the master-worker paradigm
  - Description of computational infrastructure in an XML file
- Sequential execution starts in master node and tasks are offloaded to worker nodes
- All data scheduling decisions and data transfers are performed by the runtime



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, MKLProc):
    os.environ["MKL_NUM_THREADS"]=str(MKLProc)
    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], MKLProc)
compss_barrier()
mulTime = time.time() - startMulTime
```

- In terms of the overall format, let's plan on each of you starting with a very quick (~5-6 minutes) talk that focuses on:
  - An application that highlights the use of an alternative to MPI+X, and
  - Why you think it presents a distinct advantage to "tried-and-true" MPI+X (yes, in case you are wondering, auto-correct did not save me from almost sending this email with "tired-and-true" 😊).

- Feel free to order these two points however you see fit to presenting your position. You can consider this as your "elevator speech". Obviously, the goal here is to get people fired up (fellow panelists, audience, moderator, etc.) to drive the discussions for the remainder of the session.