



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



How COMPSs can be useful at ET-EIB

Raül Sirvent

Workflows & Distributed Computing Group

May 2023

Barcelona

ET-EIB Workshop Aachen (March 2023)

- Intro: “Provide a software framework allowing traceability and reproducibility, efficient job submission and data access” – **COMPSs provenance recording, COMPSs runtime system**
- Division 1: “We should promote grid computing from the beginning to not run into the same issues LIGO and Virgo are facing with pipelines that can not run on the grid, because they rely on features that are not part of the grid infrastructure” – **COMPSs applications’ portability**
- Need to understand better the use cases to see real possibilities

Motivation

- New complex architectures constantly emerging
 - With their own way of programming them
 - Fine grain: e.g. Programming models and APIs to run with GPUs, NVMs (Non-Volatile Memories)
 - Coarse grain: e.g. APIs to deploy in Clouds
 - **Difficult** for programmers
 - Higher learning curve / Time To Market (TTM)
 - What about non computer scientists???
 - **Difficult** to understand what is going on during execution
 - Was it fast? Could it be even faster? Am I paying more than I should? (**Efficiency**)
 - Tune your application for each architecture (or cluster)
 - E.g. partitioning data among nodes

Motivation

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

COMPSs

Programming Model and Runtime System



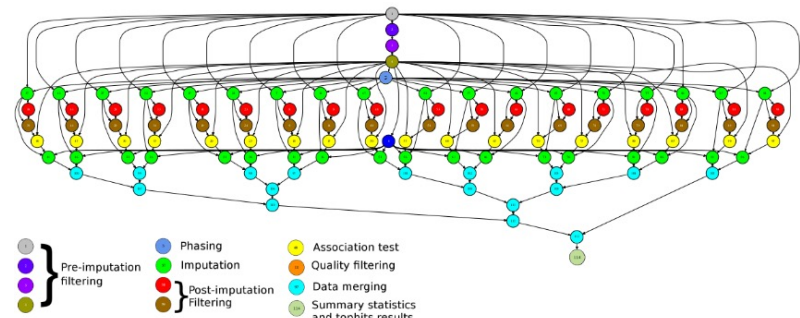
**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

Programming with PyCOMPSs/COMPSs

- Sequential programming, parallel execution
- General purpose programming language + annotations/hints
 - To identify tasks and directionality of data
 - Task based: task is the unit of work
- Builds a task graph at runtime
 - Express potential concurrency
 - Exploitation of parallelism
- Offers a shared memory illusion to applications in a distributed system
 - The application can address larger data storage space:
 - Support for Big Data apps
- Simple linear address space
- Agnostic of computing platform
 - Runtime takes all scheduling and data transfer decisions

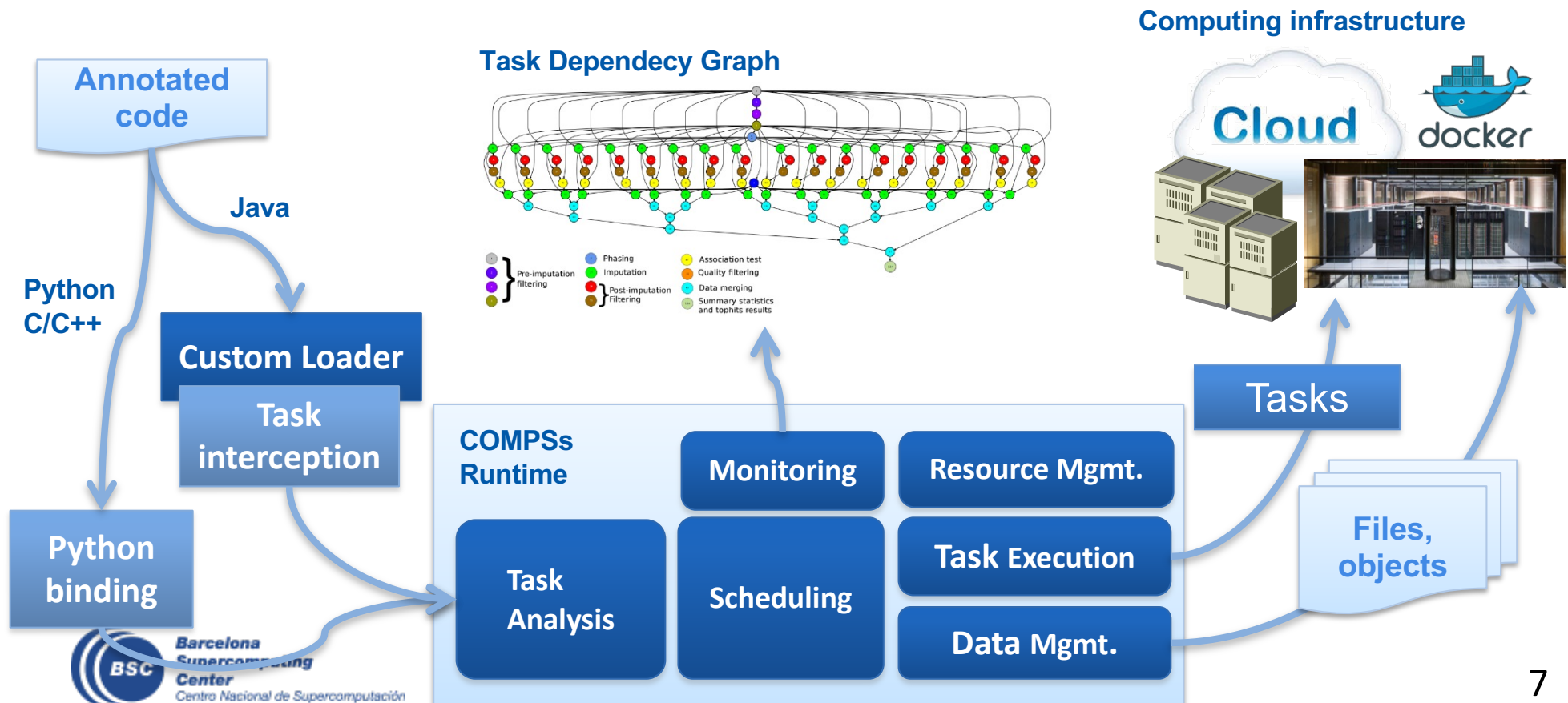
```
initialize_variables()  
for i in range(MSIZE):  
    for j in range(MSIZE):  
        for k in range(MSIZE):  
            multiply (A[i][k], B[k][j], C[i][j])
```

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



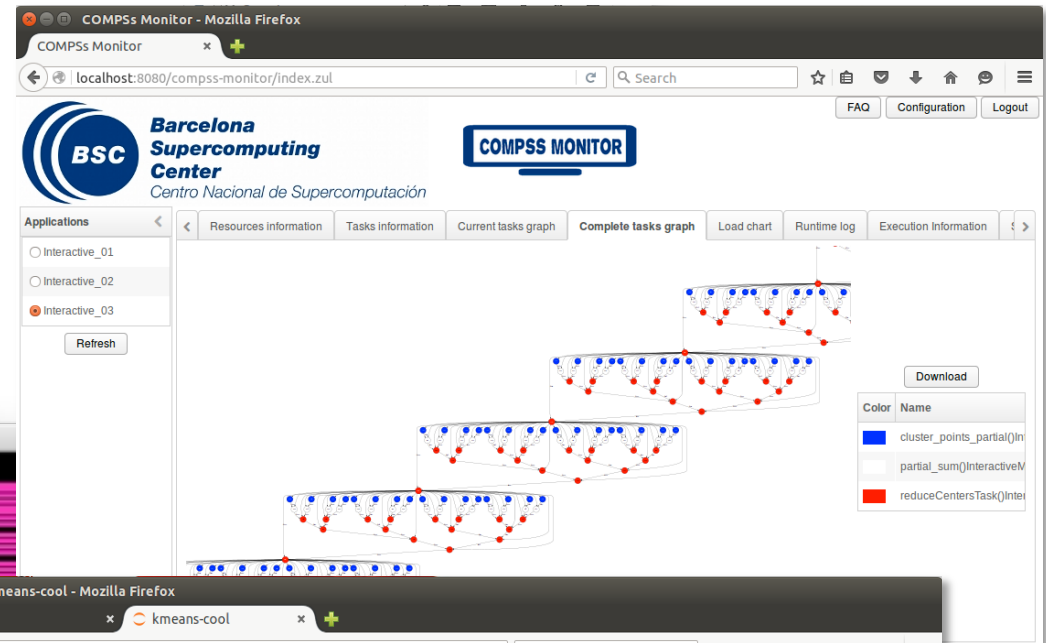
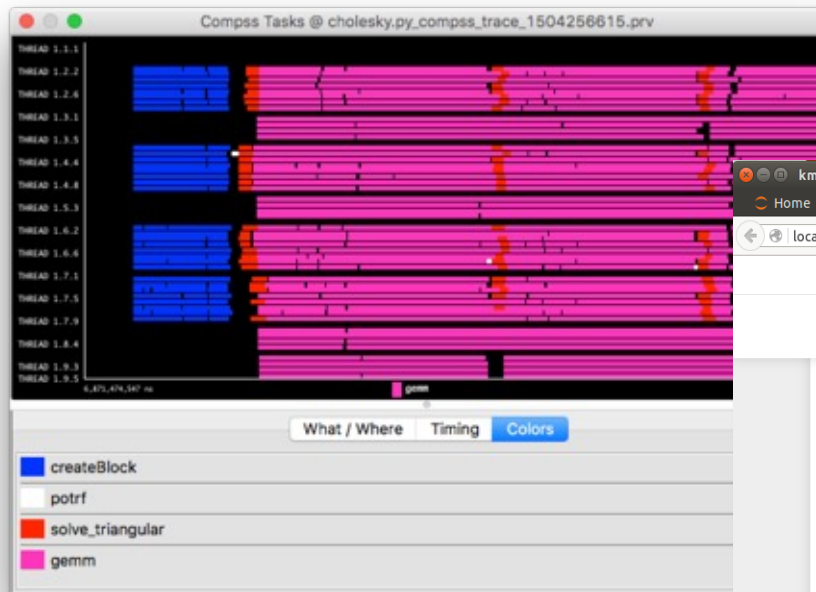
COMPSs runtime

- PyCOMPSs/COMPSs applications executed in distributed mode following the master-worker paradigm
- Sequential execution starts in master node
- Tasks are offloaded to worker nodes
- All data scheduling decisions and data transfers are performed by the runtime



PyCOMPSs development environment

- Runtime monitor
- Paraver traces (HPC!!!)
- Jupyter-notebooks integration



Jupyter Notebook interface showing Python code for kmeans-cool. The code defines a task function for cluster_points_partial and a partial_sum function. The notebook output shows the code being executed and the task being appended.

```
data.append(d)
return np.array(data)[:numV]
else:
return [np.random.random(dim) for _ in range(numV)]

In [7]: @task(returns=dict)
def cluster_points_partial(XP, mu, ind):
dic = {}
for x in enumerate(XP):
bestmukey = min([(i[0], np.linalg.norm(x[1] - mu[i[0]])) for i in enumerate(mu)], key=lambda i, l: (i[0], np.linalg.norm(x[1] - mu[i[0]]))])
if bestmukey not in dic:
dic[bestmukey] = [x[0] + ind]
else:
dic[bestmukey].append(x[0] + ind)
return dic
Task appended.

In [8]: @task(returns=dict)
def partial_sum(XP, clusters, ind):
p = [(i, [(XP[j] - ind) for j in clusters[i]]) for i in clusters]
dic = {}
for i, l in p:
dic[i] = (len(l), np.sum(l, axis=0))
return dic
Task appended.
```


COMPSs Execution Environments (Portability)



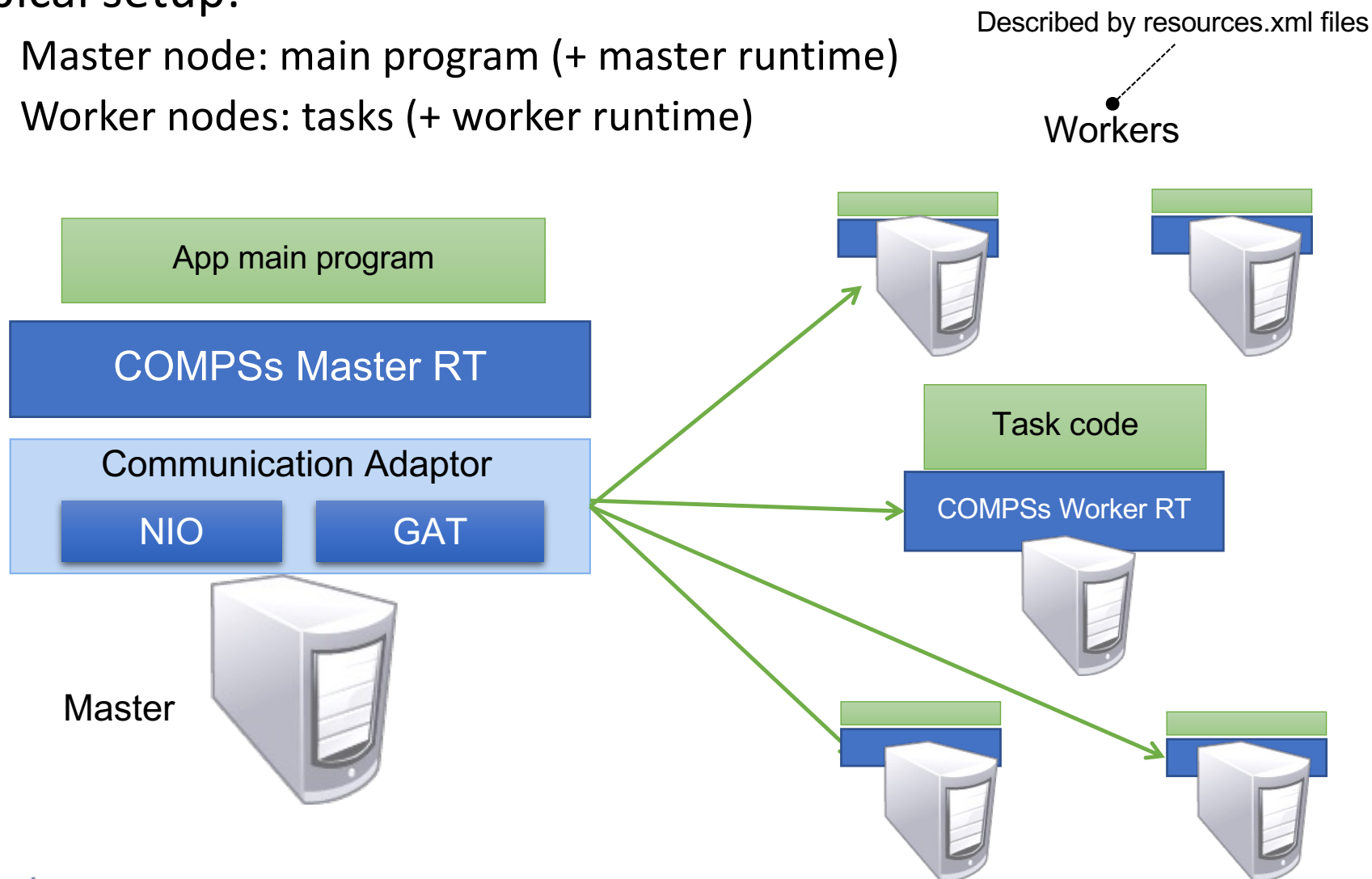
**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

Basic Execution Environments

- Interactive Computing Nodes
- Clusters (interaction with batch jobs systems)
- Distributed: Grids/Clouds (interaction with Grid/Cloud Provider APIs)

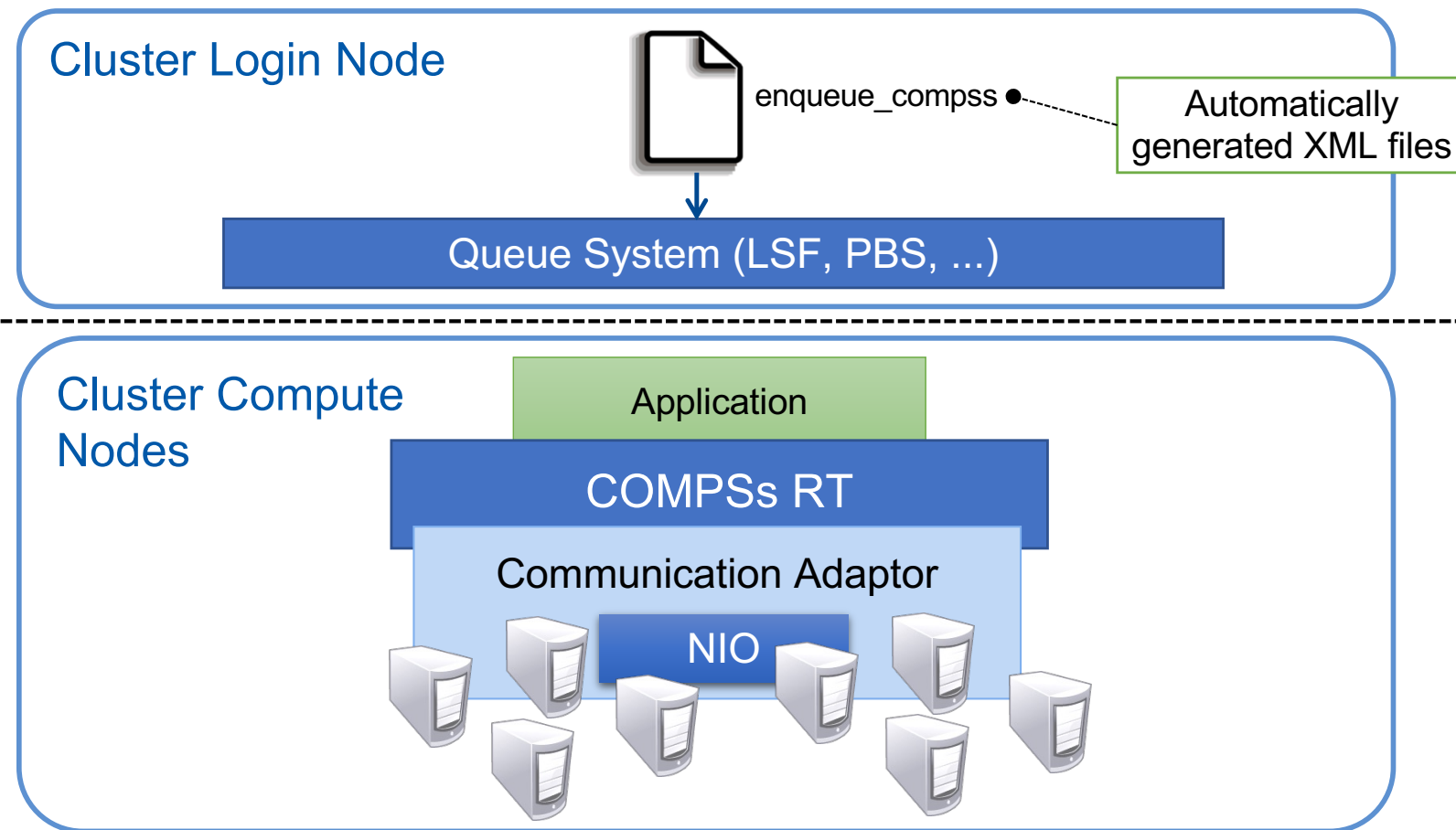
COMPSs @ Interactive Hosts

- Typical setup:
 - Master node: main program (+ master runtime)
 - Worker nodes: tasks (+ worker runtime)



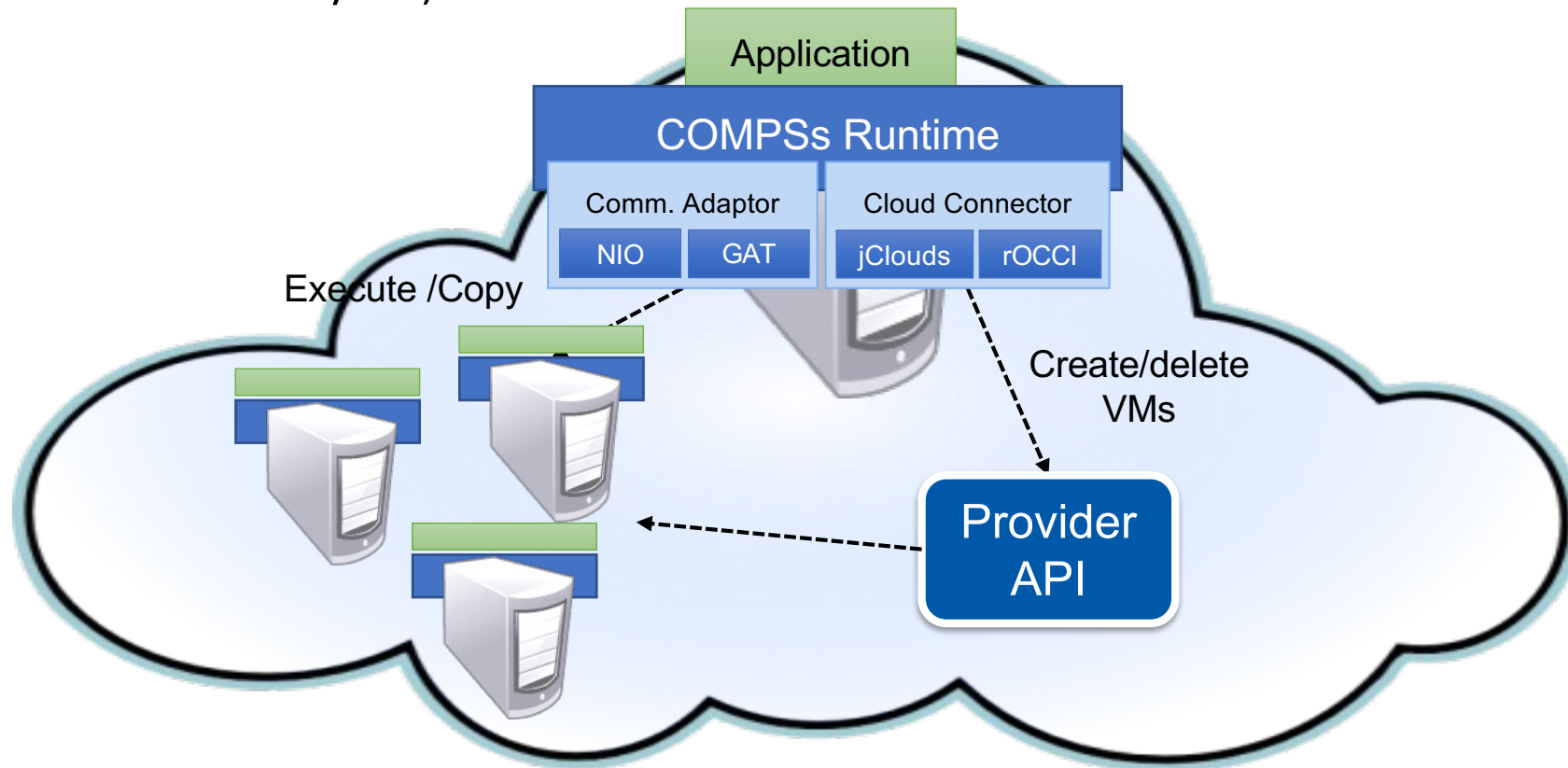
COMPSs@Cluster

- Execution divided in two phases
 - Launch scripts queue a whole COMPSs app execution
 - Actual execution starts when reservation is obtained



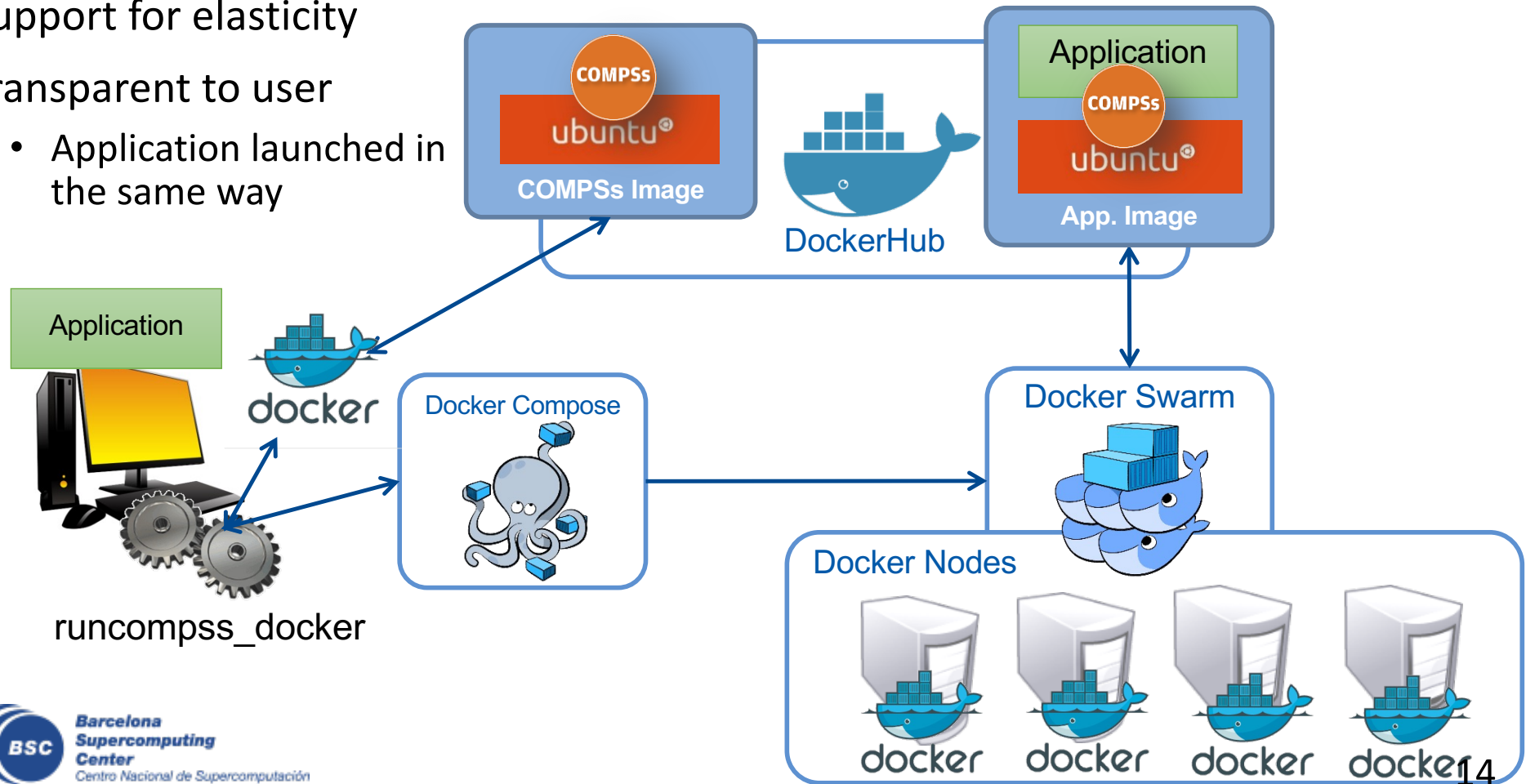
COMPSs@Grid/Cloud

- Execution of COMPSs applications in Grids/Clouds
 - Select the connector to interact with the Grid/Cloud provider
 - Adaptor to communicate VMs (NIO if provider supports firewall management, GAT if only ssh)



COMPSs runtime: containers

- Two alternatives
 - Whole application deployed as a container
 - Support for Docker, Singularity and other container engines
 - Individual tasks can be in containers
- Support for elasticity
- Transparent to user
 - Application launched in the same way



COMPSs Unique Advanced Features for ET



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

Automatic Workflow Provenance Recording

- Record details of your COMPSs (and dislib) workflow executions

- Ensure **reproducibility** and **replicability** of the results
- **RO-Crate** workflow profile format used (simple and interoperable)



- Usage: **-p** or **--provenance** flag with *compss run*

- **Input:** Simple YAML file to describe the application and its authors
- **Output:** Resulting package with: Application source files, Graph image, Application profiling, RO-Crate metadata file
- RO-Crate metadata includes:

- References to detected inputs and outputs used by the workflow (and their details), but **files are not included** (avoid moving large datasets)
- Application details, COMPSs version used, hostname, ...

- More info:

https://compss-doc.readthedocs.io/en/stable/Sections/05_Tools/04_Data_Provenance.html



Multiple Task Implementations

- Multiple implementations for a task with the same objective, but with different constraints (e.g. specific libraries, hardware)
- Transparently, the runtime will invoke the implementation that fulfils the constraints within each resource

```
@implement(source_class="sourcemodule", method="main_func")
@constraint(app_software="numpy")
@task(returns=list)
def myfunctionWithNumpy(list1, list2):
    # Operate with the lists using numpy
    return resultList

@task(returns=list)
def main_func(list1, list2):
    # Operate with the lists using built-int functions
    return resultList
```

Failure Management & Dynamic Workflow

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

```
@task(file_path=FILE_INOUT, on_failure='CANCEL_SUCCESSORS')
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**

Leveraging NUMBA

- Just in time compilation for Python and NumPy code

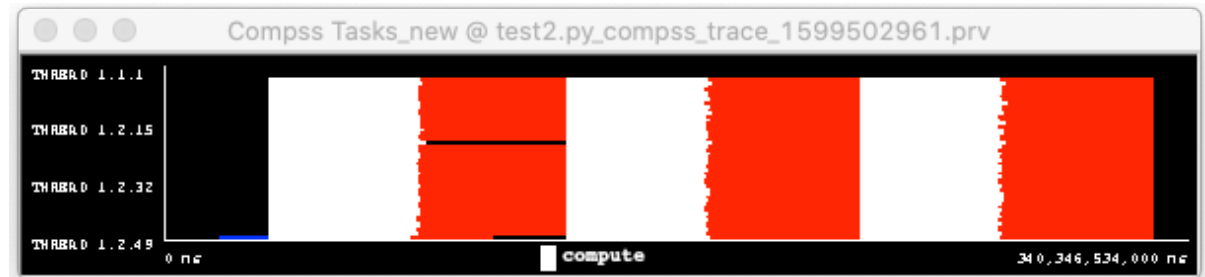
```
@task (returns=1)  
def ident_loops(x):  
    r = np.empty_like(x)  
    n = len(x)  
    for i in range(n):  
        r[i] = np.cos(x[i]) ** 2 + np.sin(x[i]) ** 2  
    return r
```

```
@task(returns=1, numba=True)  
def ident_loops_jit(x):  
    r = np.empty_like(x)  
    n = len(x)  
    for i in range(n):  
        r[i] = np.cos(x[i]) ** 2 + np.sin(x[i]) ** 2  
    return r
```

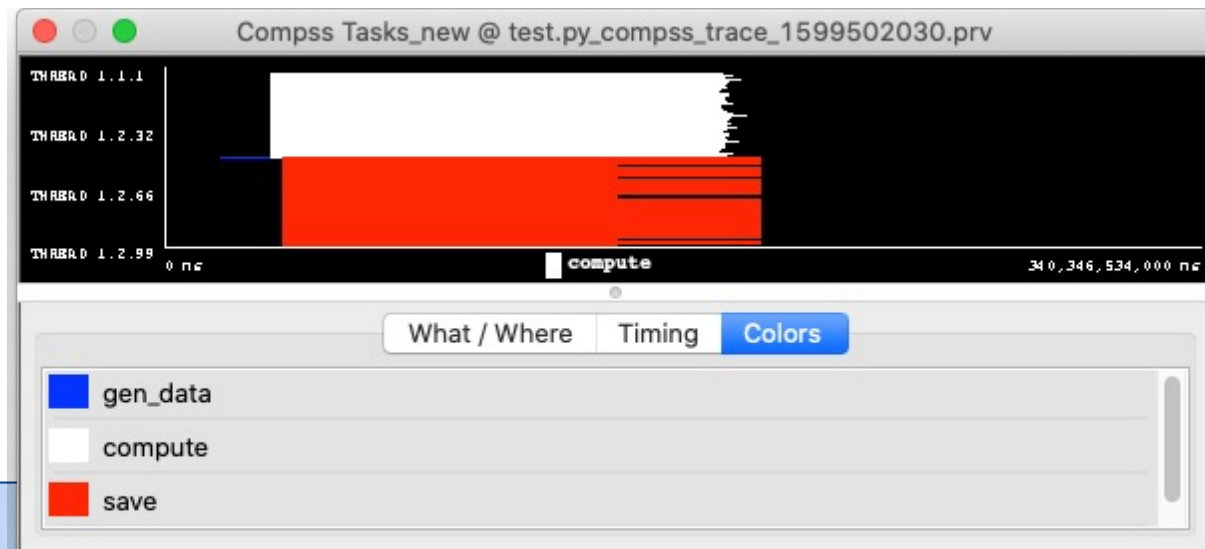
Data Filtering Tasks (IO Tasks)

- IO tasks can run in parallel with compute tasks using virtual IO resources
- IO tasks can use a specific amount of BW

Without IO resources



With IO resources



```
@constraint (storageBW=9)
```

```
@IO()
```

```
@task()
```

```
def save(data, itr):
```

```
...
```

```
    f = open(dest, "w")
```

```
    f.write(data)
```

```
    f.flush()
```

```
...
```

```
@task(returns=list)
```

```
def compute()
```

```
...
```

```
    for i ...
```

```
        #some computation
```

```
...
```

```
    for j in range(N):
```

```
        for i in range(48):
```

```
            c[i] = compute()
```

```
        for i in range(48):
```

```
            save(c[i], i*(j+1))
```

Other Potentially Useful Advanced Features



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

Other decorators: 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
 - Example: 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):  
    ...
```

Other decorators: linking with other programming models

- 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 OpenMP/OmpSs**
- Additional decorators:
 - `@binary(binary="app.bin")`
 - `@ompss(binary="ompssApp.bin")`
 - `@mpi(binary="mpiApp.bin", runner="mpirun", processes=8)`
- Can be combined with the `@constraint` and `@implement` decorators

```
@constraint (computingUnits= "8")
@mpi (runner="mpirun", processes= "16", ...)
@task (returns=int, stdoutFile=FILE_OUT_STDOUT, ...)
def nems(stdoutFile, stderrFile):
    pass
```

Support for data streams

- New 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)
```


Integration with persistent memory

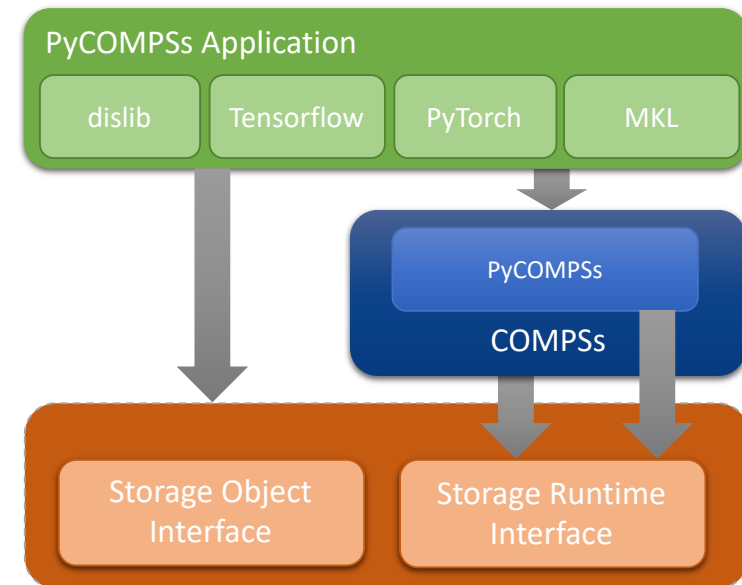
- Programmer may decide to make persistent specific objects in its code
- Persistent objects are managed same way as regular objects
- Tasks can operate with them

```
a = SampleClass ()
a.make_persistent()
Print a.func (3, 4)

a.mytask()
compss_barrier()

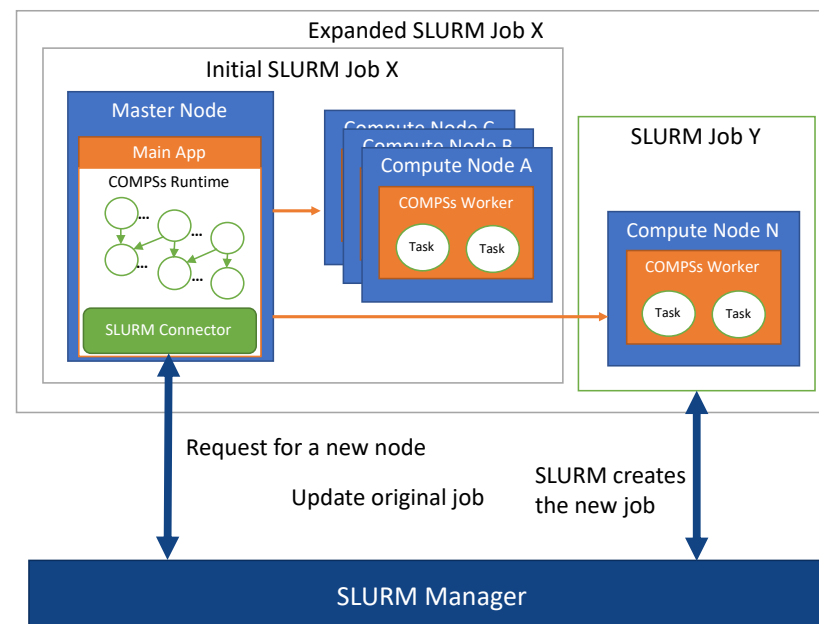
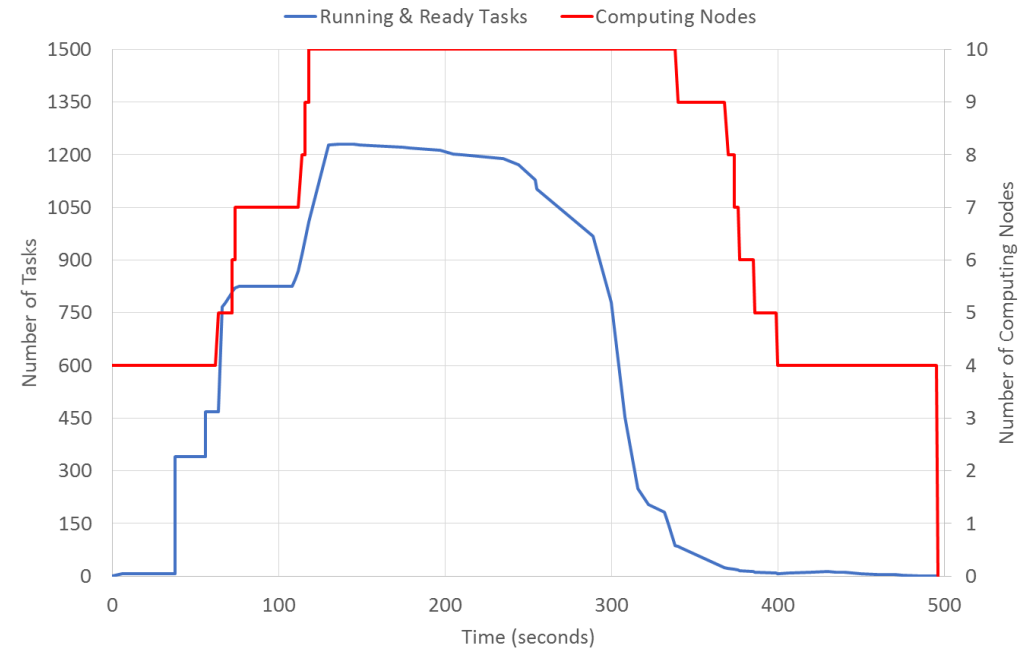
o = a.another_object
```

- **Objects can be accessed/shared transparently in a distributed computing platform**



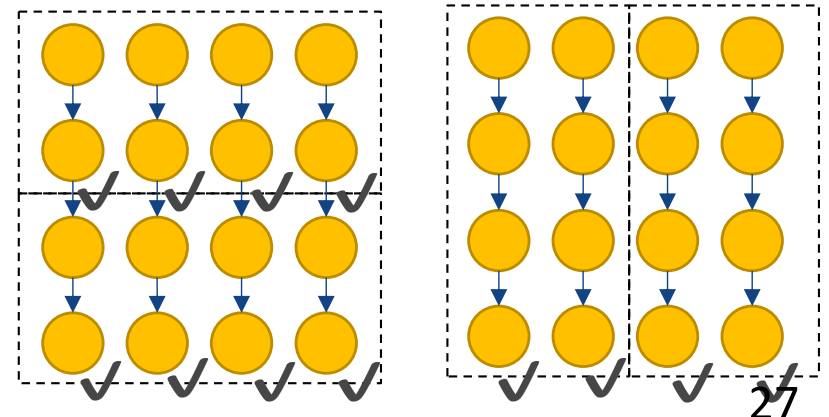
Support for elasticity

- Possibility to adapt the computing infrastructure depending on the actual workload
- Now also for SLURM managed systems
- Feature that contributes to a more effective use of resources



Checkpointing

- Mechanism to recover from failures
 - Allows the workflow re-execution avoiding the re-execution of finished tasks
- Asynchronous but with 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 a final result of the group
- Indicated by the developer with API
- Extensible Policies
 - customize group creations



Timeouts and exceptions

- Timeouts can be defined for each 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
- Can be combined to dynamically make decisions depending on the actual behavior

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

Final notes








**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación

Take-away messages

- Sequential programming approach (Java, Python, C++)
- Parallelization at task level
- Transparent data management and remote execution
- Easily integrate legacy applications (binaries, MPI)
- Can operate on different infrastructures:
 - Cluster
 - Grid
 - Cloud (Public/Private)
 - Containers
- Dislib for Machine learning on top of PyCOMPSs

Further Information

- Project page: <http://www.bsc.es/compss>
 - Virtual Appliance for testing & sample applications
 - Tutorials
-  YouTube Channel (Demos and Tutorials)
 - <https://www.youtube.com/@compsuperscalar3152>
- Documentation: <https://compss-doc.readthedocs.org>
- 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>



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



EXCELENCIA
SEVERO
OCHOA

THANK YOU!

support-compss@bsc.es

www.bsc.es