Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Persistent Memory Programming with PyCOMPSs

Javier Alvarez, Rosa M. Badia, **Javier Conejero**, Jorge Ejarque, Daniele Lezzi, Francesc Lordan, Cristian Ramon-Cortes

Workflows & Distributed Computing Group

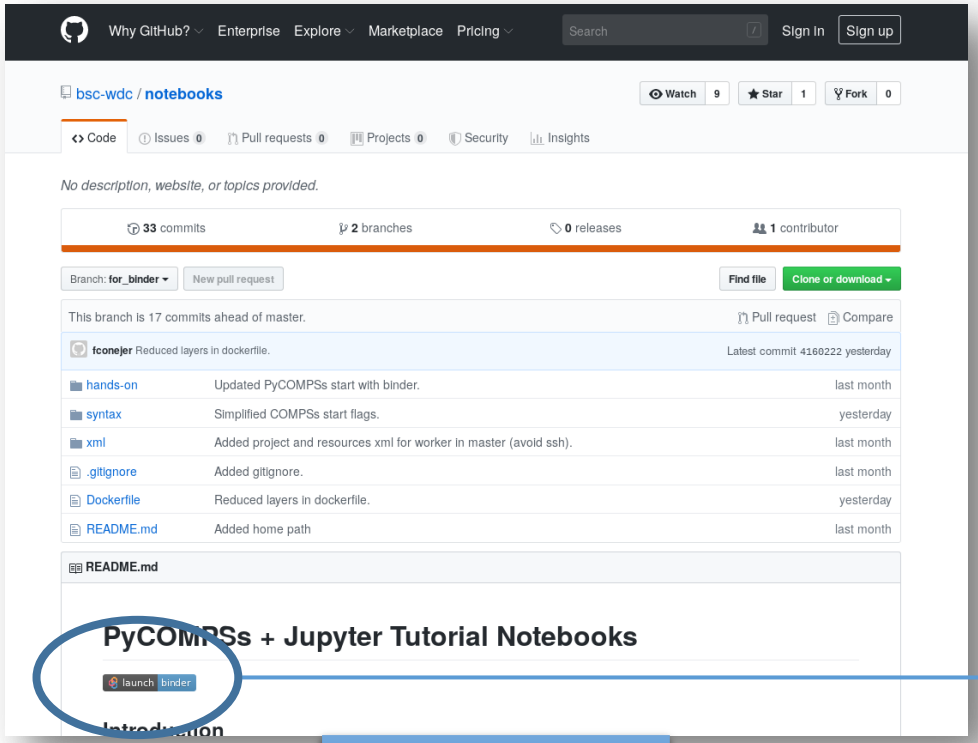
Supercomputing 2019 (SC19)

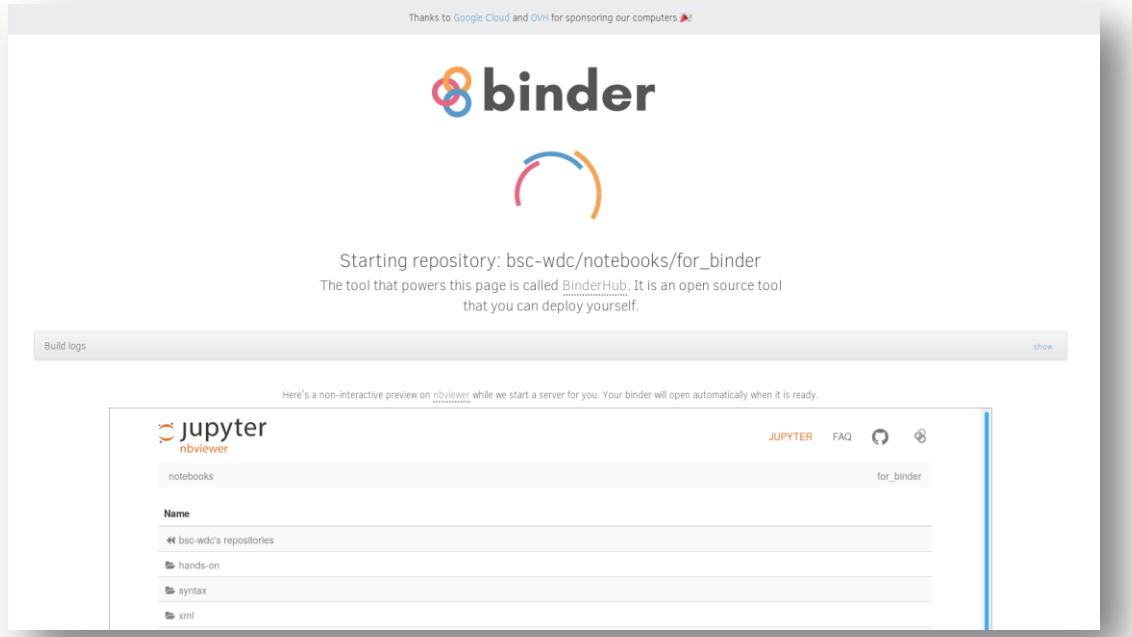
18/11/2019 Denver (CO) USA

Before we start

- Binder
- Docker
- VM

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





Docker and VM Instructions

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

What is PyCOMPSs?

PyCOMPSs is a **task based programming model** that enables the **parallel execution** of Python scripts by annotating methods with **task decorators**. At run time, PyCOMPSs' runtime identifies tasks' data dependencies, schedules and executes them in **distributed environments**.

- Outline:
  - Introduction
  - PyCOMPSs syntax through Jupyter notebooks
  - COMPSs interaction with HPC infrastructures and NVRAM features
  - Real application demo (hands-on)
  - PyCOMPSs integration with Persistent Storage

Introduction

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Issues!

- New complex architectures constantly emerging
  - With their own way of programming them
    - Fine grain: e.g. 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

Objective

- Create tools that make user's life easier
  - Intermediate layer: let the difficult parts to those tools
    - Act on behalf of the user
    - Distributing the work through resources
    - Dealing with architecture specifics
    - Automatically improving performance
  - Tools for visualization
    - Monitoring
    - Performance analysis

BSC vision on programming models

Applications

PM: High-level, clean, abstract interface

Power to the runtime

API

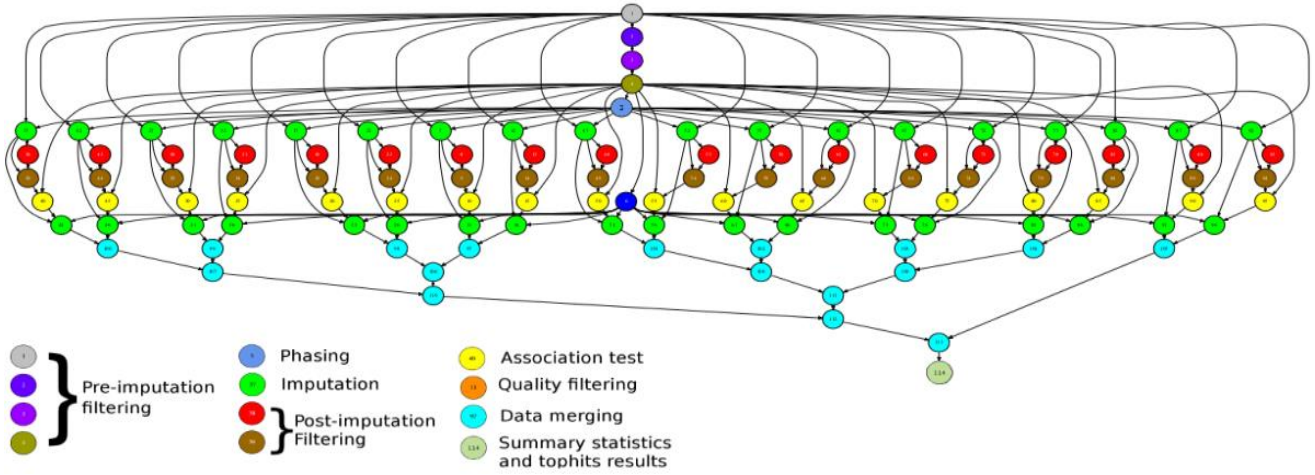
Program logic independent of computing platform

General purpose
Task based
Single address space

Intelligent runtime, parallelization, distribution, interoperability

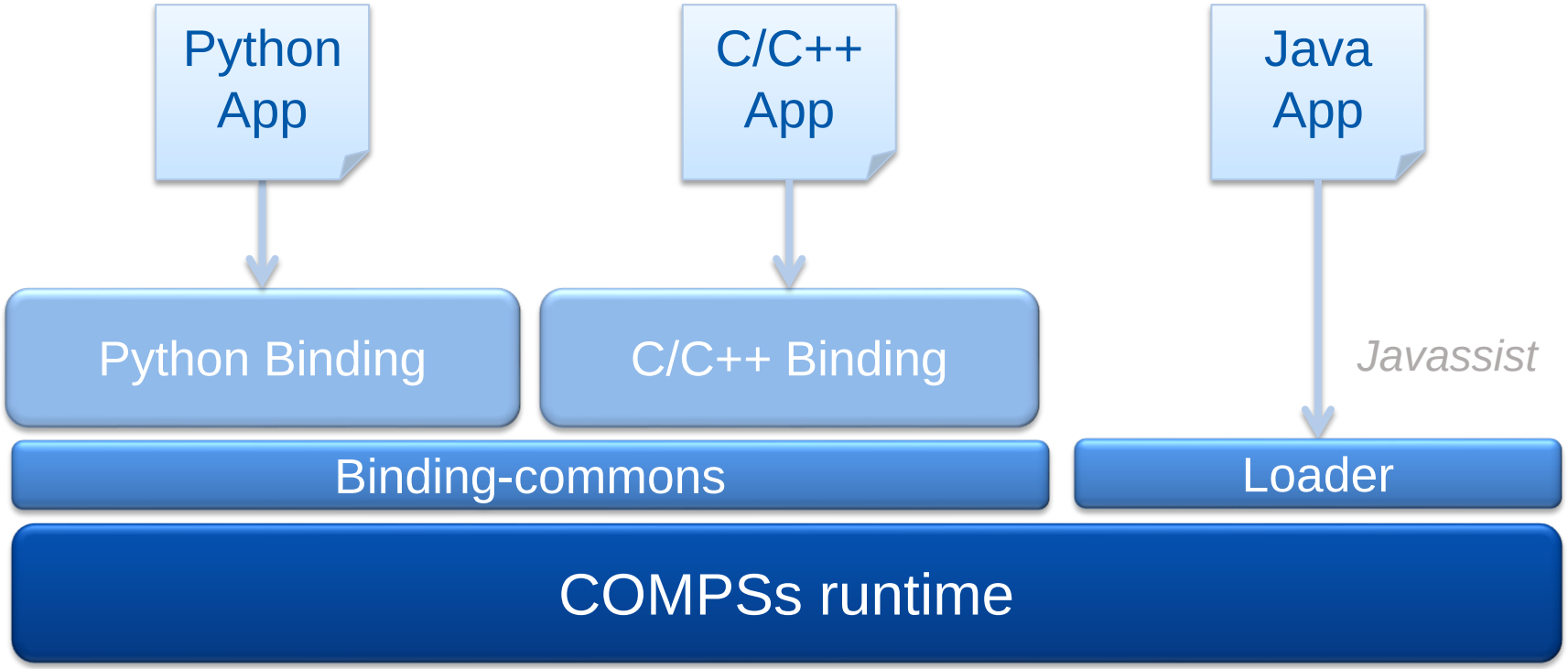
Programming with PyCOMPSs

- Sequential programming
- General purpose programming language + annotations/hints
  - To identify tasks and directionality of data
- Task based: task is the unit of work
- Simple linear address space
- Builds a task graph at runtime that express potential concurrency
  - Implicit workflow
- Exploitation of parallelism
  - … and of distant parallelism
- Agnostic of computing platform
  - Enabled by the runtime for clusters, clouds and grids

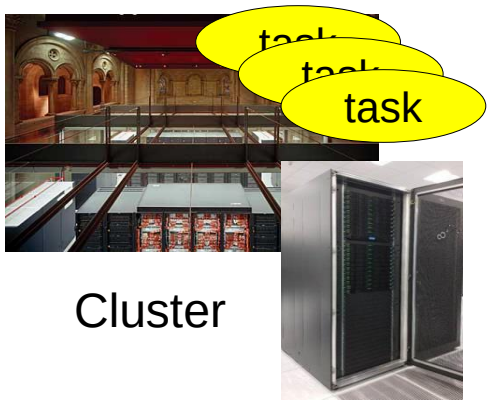


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

COMPSs Architecture

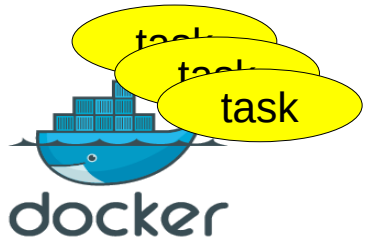


Grid



Cluster

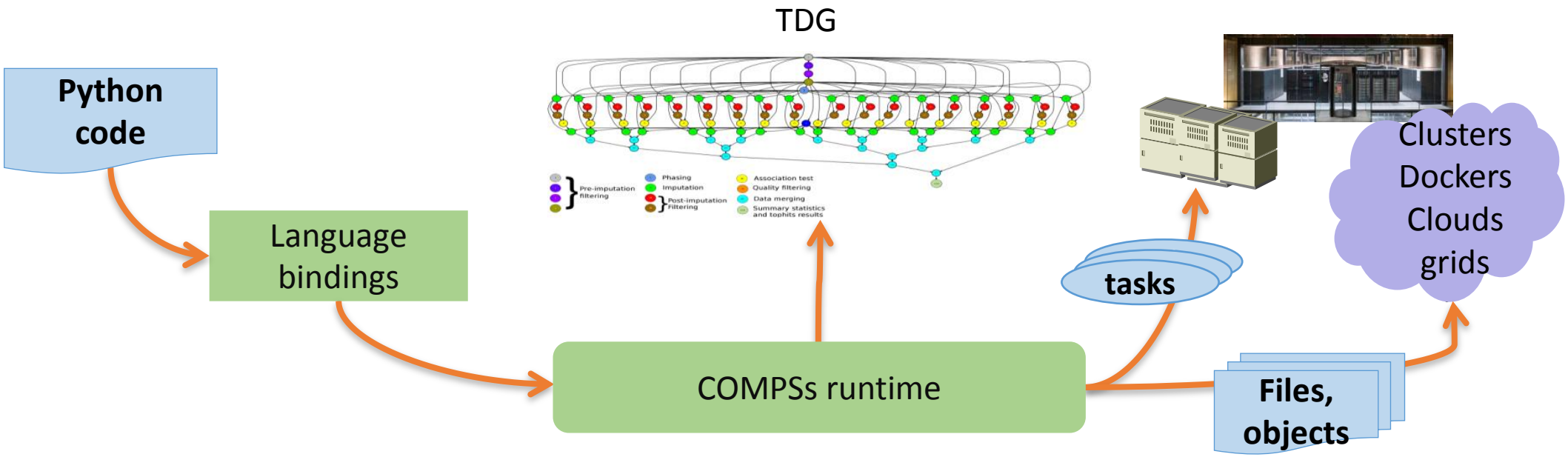
Cloud



Containers

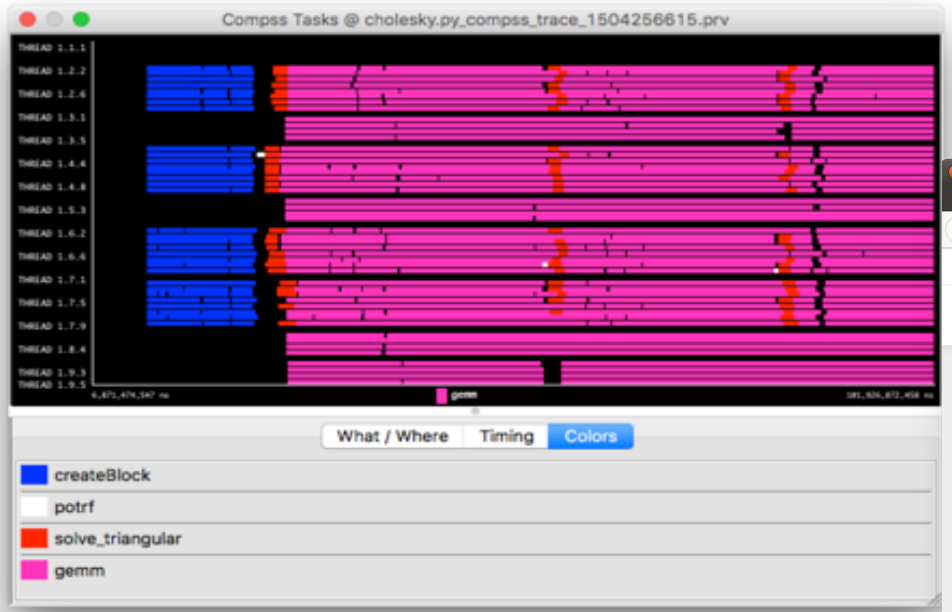
PyCOMPSs Runtime

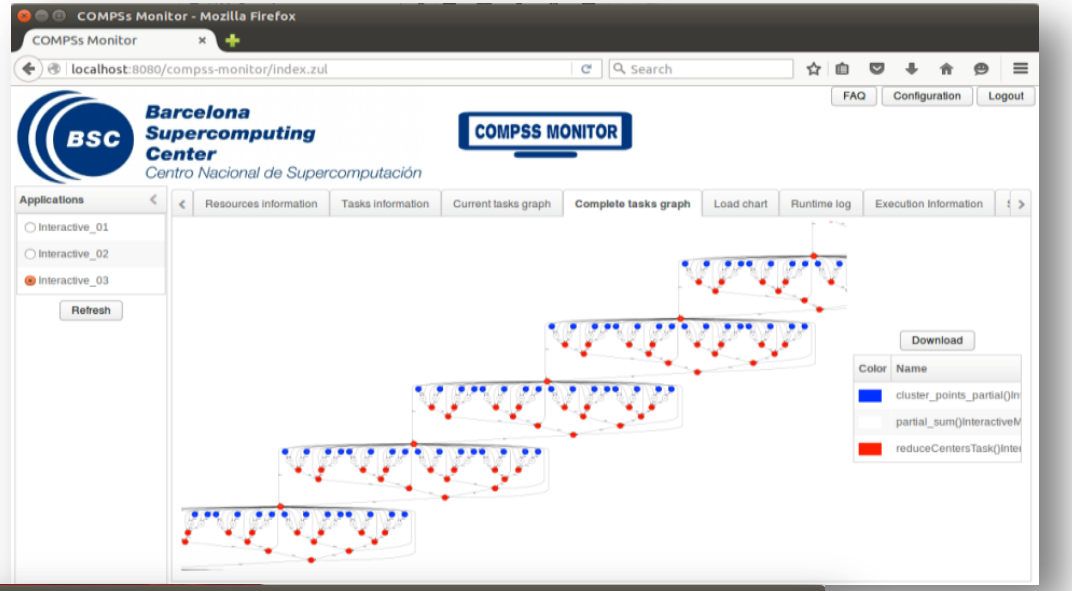
- Sequential execution starts in master node
- Tasks are offloaded to worker nodes
- All data scheduling decisions and data transfers performed by runtime

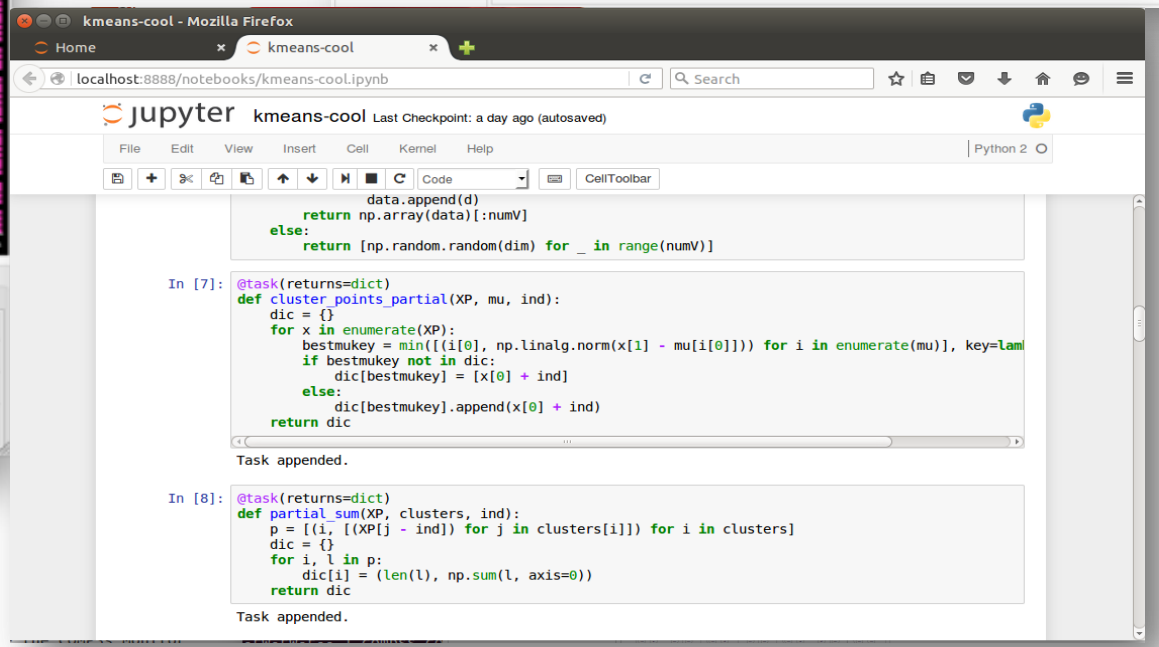


PyCOMPSs development environment

- Runtime monitor
- Paraver traces
- Jupyter-notebooks integration







Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PyCOMPSs
Syntax

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PyCOMPSs Syntax

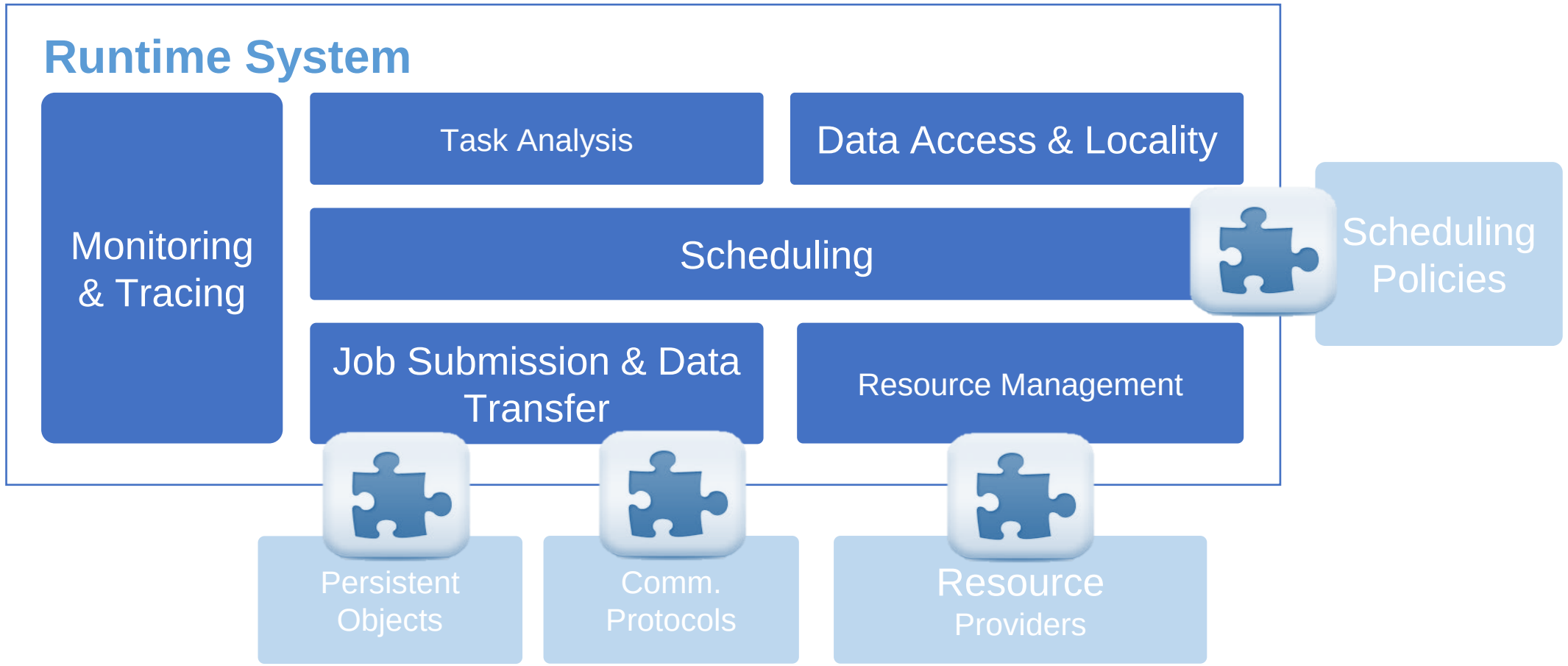
Instructions and Notebooks

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

COMPSS Runtime and execution environments

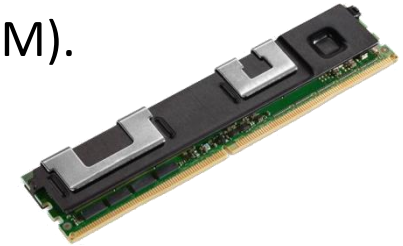
Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Runtime Architecture



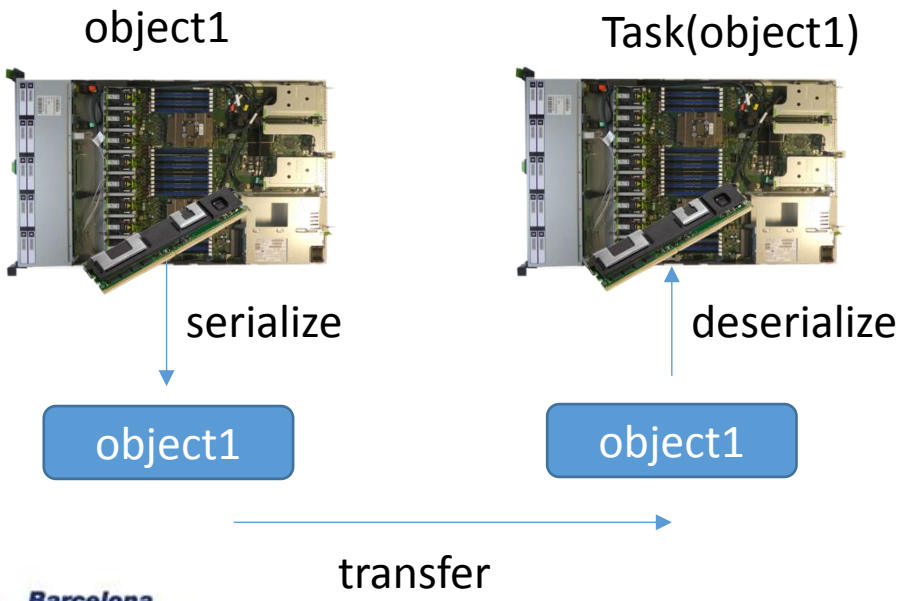
Data management in COMPSs

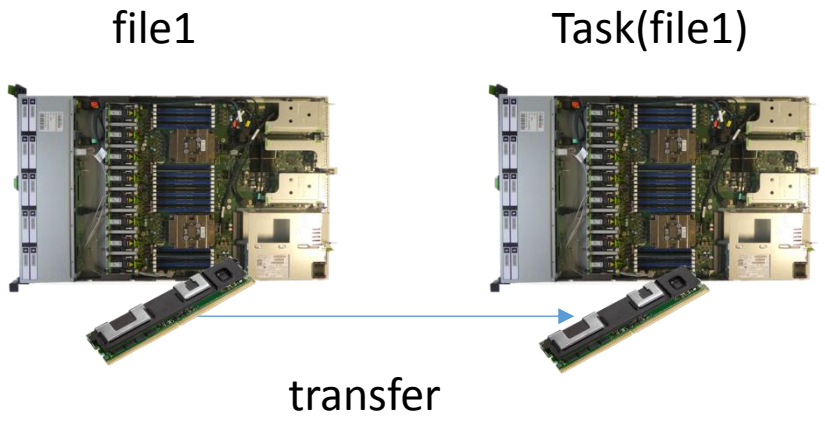
- COMPSs runtime offers the view of a single memory space and storage system
  - Can address memory spaces much larger than the initial space available in a single node
  - This larger memory space is made available to the application in a transparent way
- Data management
  - Objects can be either created by the main program or tasks
  - When accessed by tasks executed in different worker nodes, the runtime will be in charge of transferring the data between them
  - Files and/or objects can be renamed/versioned
  - Renaming enables further parallelism
  - Versioning reduces the number of required transfers
- Runtime supports shared and distributed filesystems, such as NVRAM (2LM).
  - In the first case, no file transfers are required



Data management in COMPSs

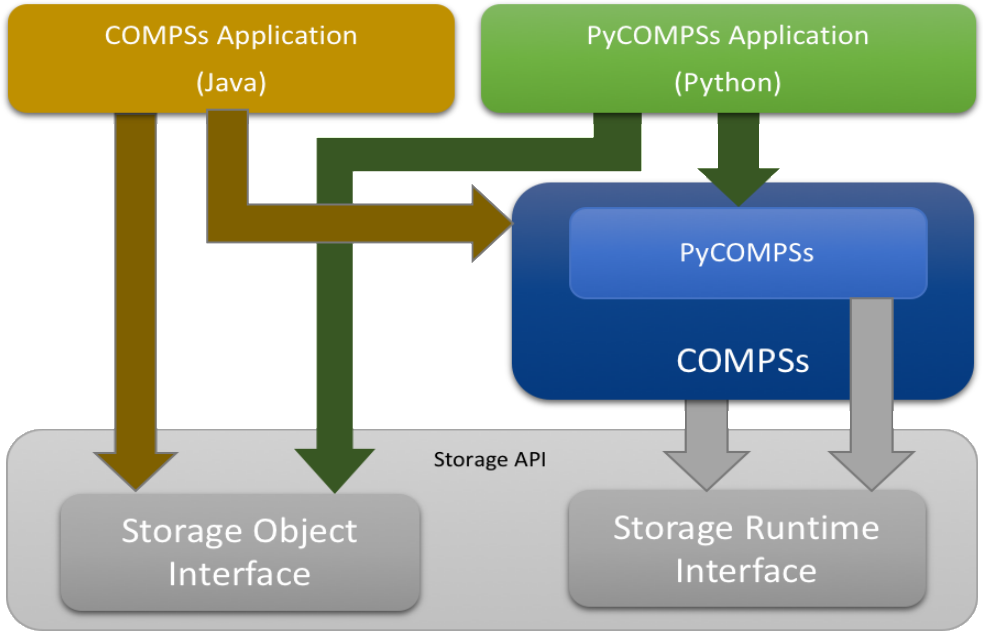
- Serialization
  - Due to different address spaces between different nodes, objects in memory need to be transferred
  - Before being transferred are serialized
  - Only for objects, not for files
  - For Python, Pickle/Dill/Numpy libraries are used





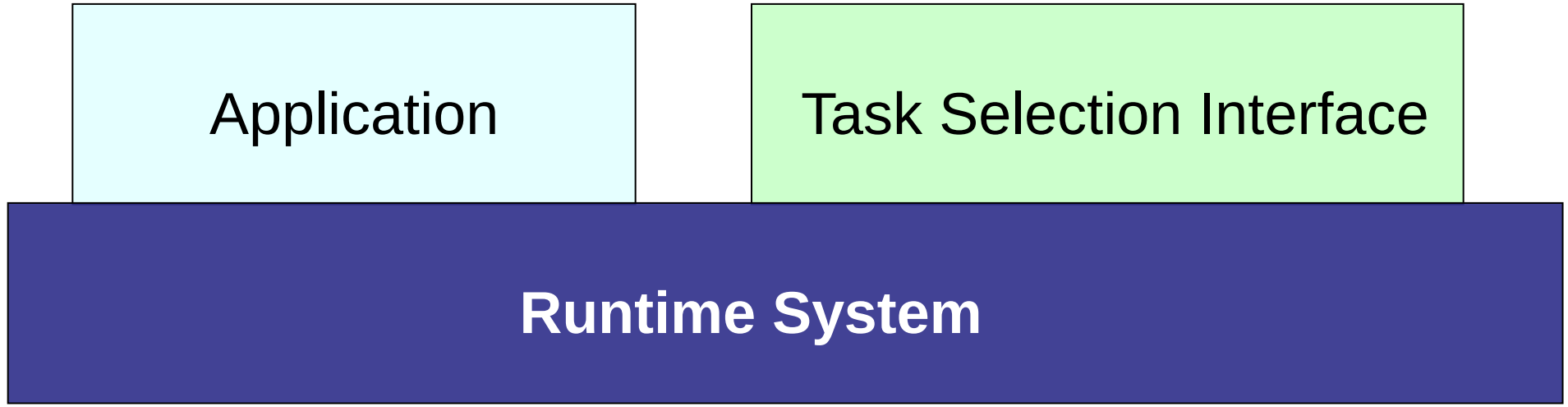
Integration with Storage: Storage API

- Integration of programming model with new storage management platforms
- Data made persistent, application agnostic of this persistency
- Enables reducer-consumer schemes
- Enables in-situ operations



dataClay Hecuba redis

Runtime System

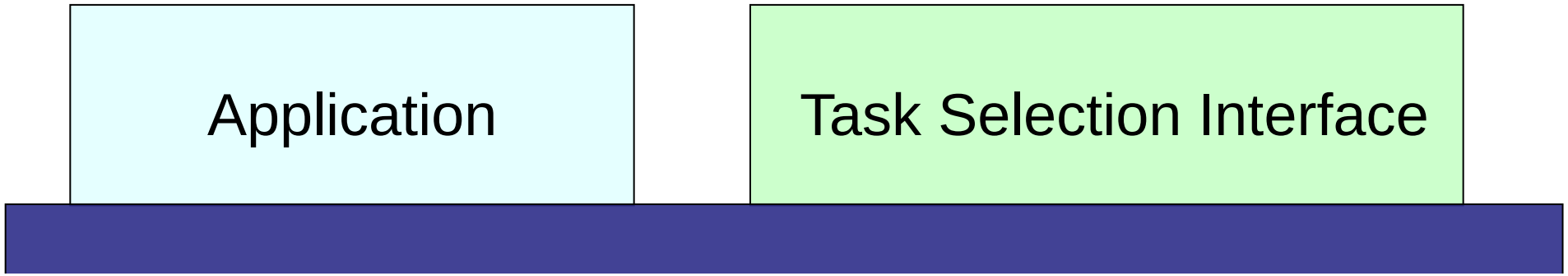


Grid

Cluster

Cloud

Runtime System



How do I select the execution platform?

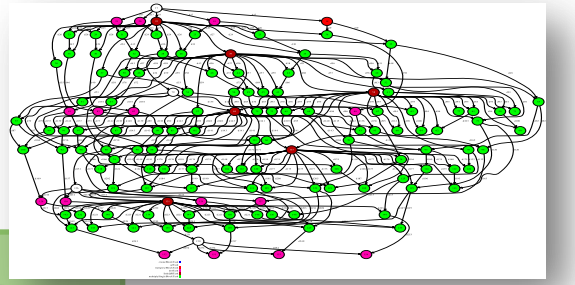
Grid

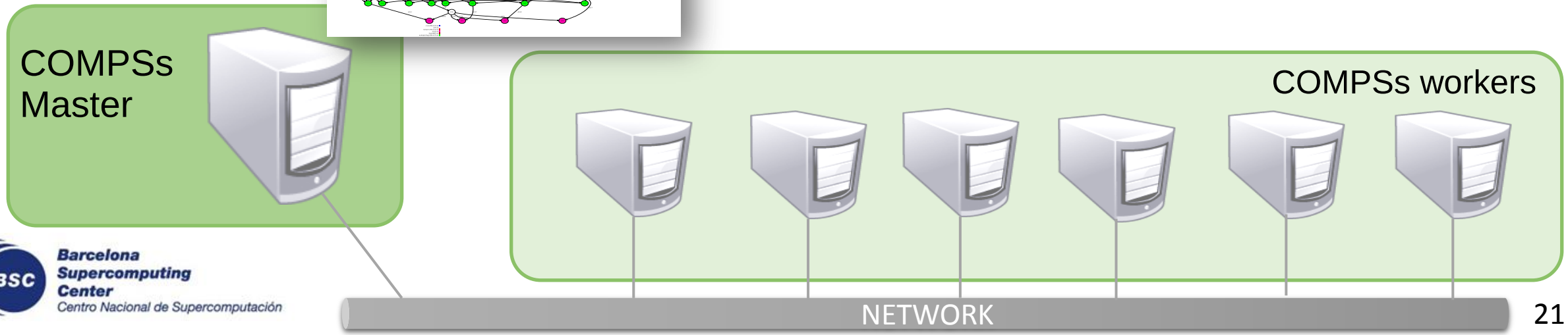
Cluster

Cloud

Execution environment

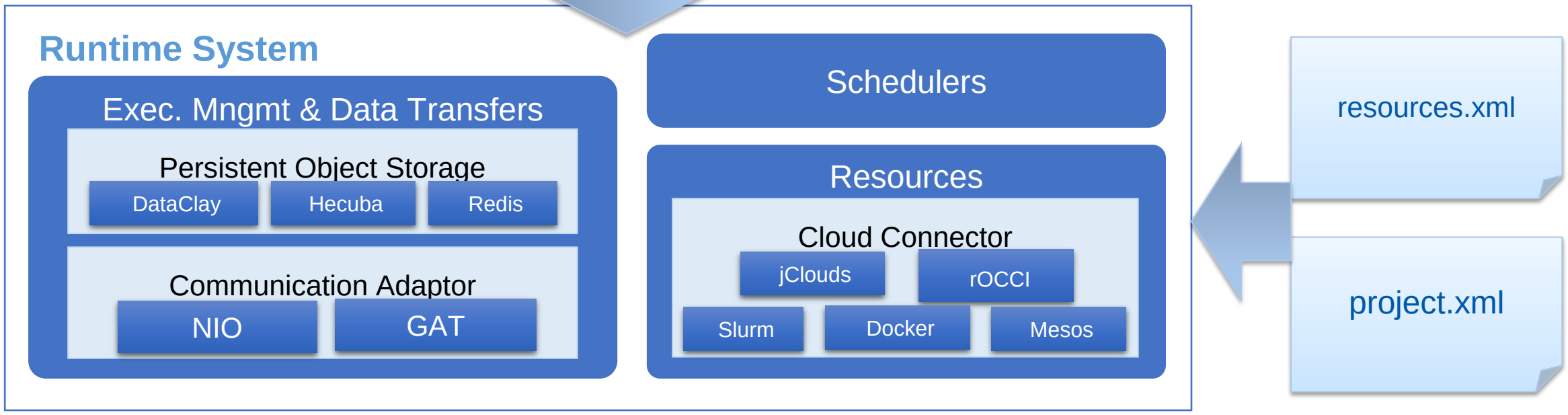
- COMPSs runtime is able to execute in distributed computing platforms (clusters, clouds, contained manager clusters)
- Main program + runtime started in master node
  - Takes care of tasks scheduling, data transfers, etc.
- Tasks executed in worker nodes





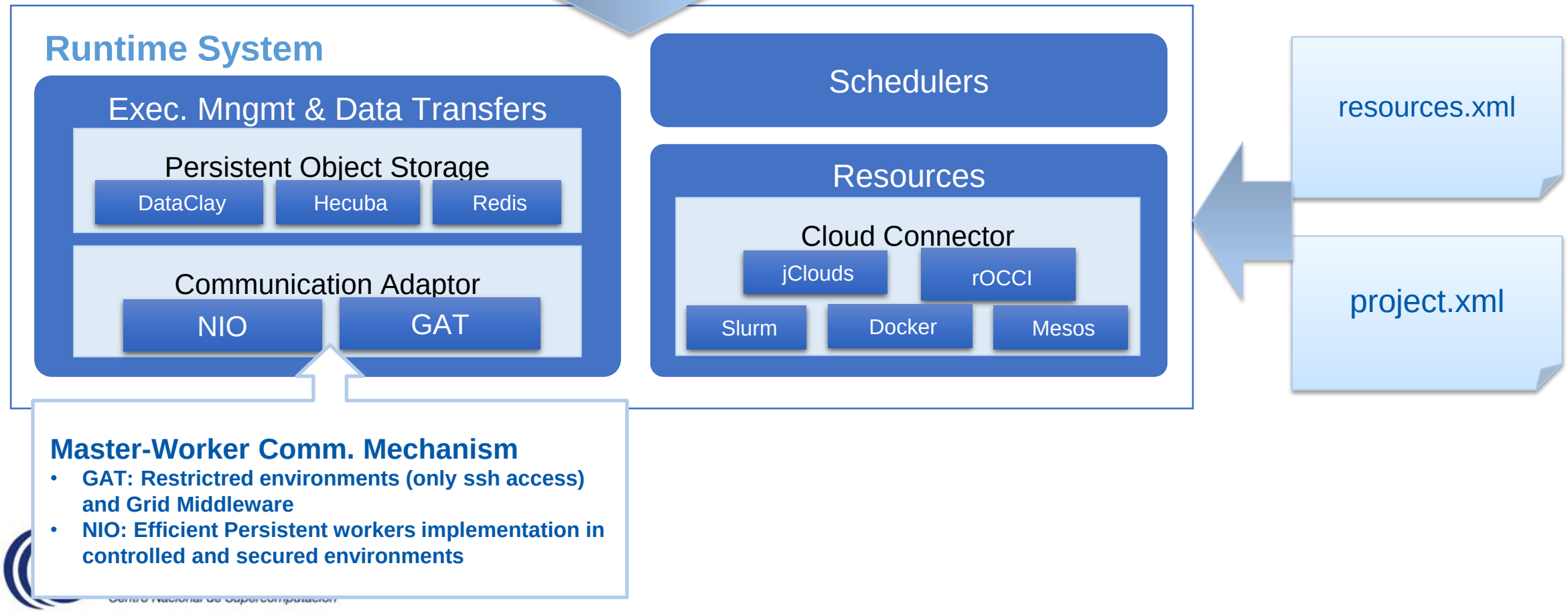
Execution Environments Configuration

runcompss options



Execution Environments Configuration

runcompss options

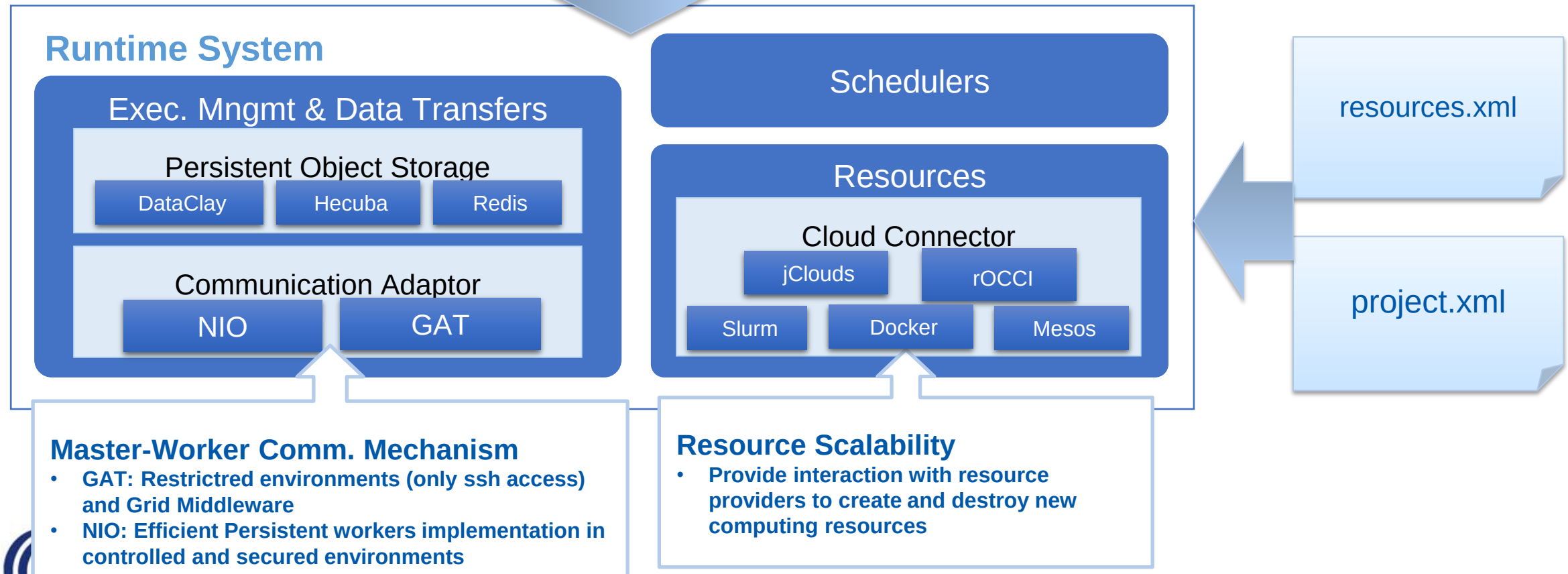


Master-Worker Comm. Mechanism

- **GAT: Restrictred environments (only ssh access) and Grid Middleware**
- **NIO: Efficient Persistent workers implementation in controlled and secured environments**

Execution Environments Configuration

runcompss options



Master-Worker Comm. Mechanism

- **GAT: Restrictred environments (only ssh access) and Grid Middleware**
- **NIO: Efficient Persistent workers implementation in controlled and secured environments**

Resource Scalability

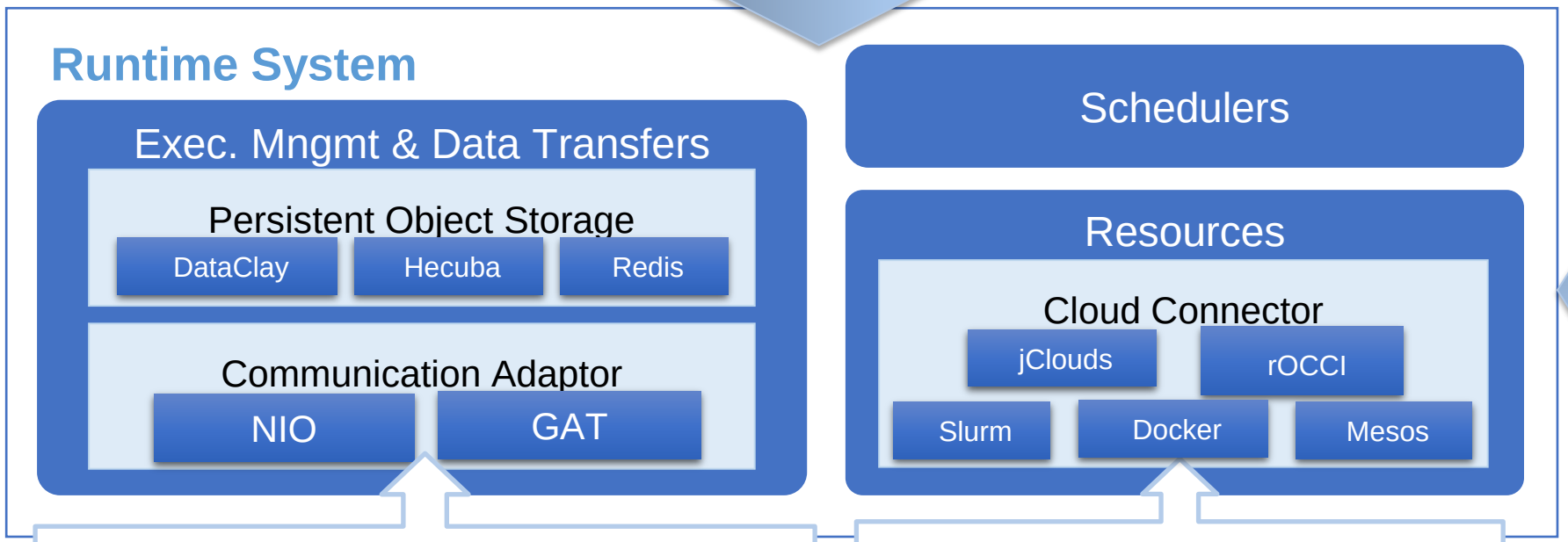
- **Provide interaction with resource providers to create and destroy new computing resources**

Execution Environments Configuration

Specifies:

- **Resources and Project files**
- **Scheduler, Comm. Adaptor**
- **Persistent object storage**

runcompss options



resources.xml

project.xml

Master-Worker Comm. Mechanism

- **GAT: Restrictred environments (only ssh access) and Grid Middleware**
- **NIO: Efficient Persistent workers implementation in controlled and secured environments**

Resource Scalability

- **Provide interaction with resource providers to create and destroy new computing resources**

Execution Environments Configuration

Specifies:

Resources and Project files
Scheduler, Comm. Adaptor
Persistent object storage

runcompss options

Infrastructure Description

- **Describe the available resource in the infrastructure**
- **Describe Cloud Providers: Images and VM Templates**

resources.xml

project.xml

Runtime System

Exec. Mngmt & Data Transfers

Persistent Object Storage

DataClay | Hecuba | Redis

Communication Adaptor

NIO | GAT

Schedulers

Resources

Cloud Connector

jClouds | rOCCI

Slurm | Docker | Mesos

Master-Worker Comm. Mechanism

- **GAT: Restrictred environments (only ssh access) and Grid Middleware**
- **NIO: Efficient Persistent workers implementation in controlled and secured environments**

Resource Scalability

- **Provide interaction with resource providers to create and destroy new computing resources**

Execution Environments Configuration

Specifies:

- Resources and Project files
- Scheduler, Comm. Adaptor
- Persistent object storage

runcompss options

Infrastructure Description

- Describe the available resource in the infrastructure
- Describe Cloud Providers: Images and VM Templates

Runtime System

Exec. Mngmt & Data Transfers

Persistent Object Storage

DataClay | Hecuba | Redis

Communication Adaptor

NIO | GAT

Schedulers

Resources

Cloud Connector

jClouds | rOCCI | Slurm | Docker | Mesos

resources.xml

project.xml

Master-Worker Comm. Mechanism

- GAT: Restrictred environments (only ssh access) and Grid Middleware
- NIO: Efficient Persistent workers implementation in controlled and secured environments

Resource Scalability

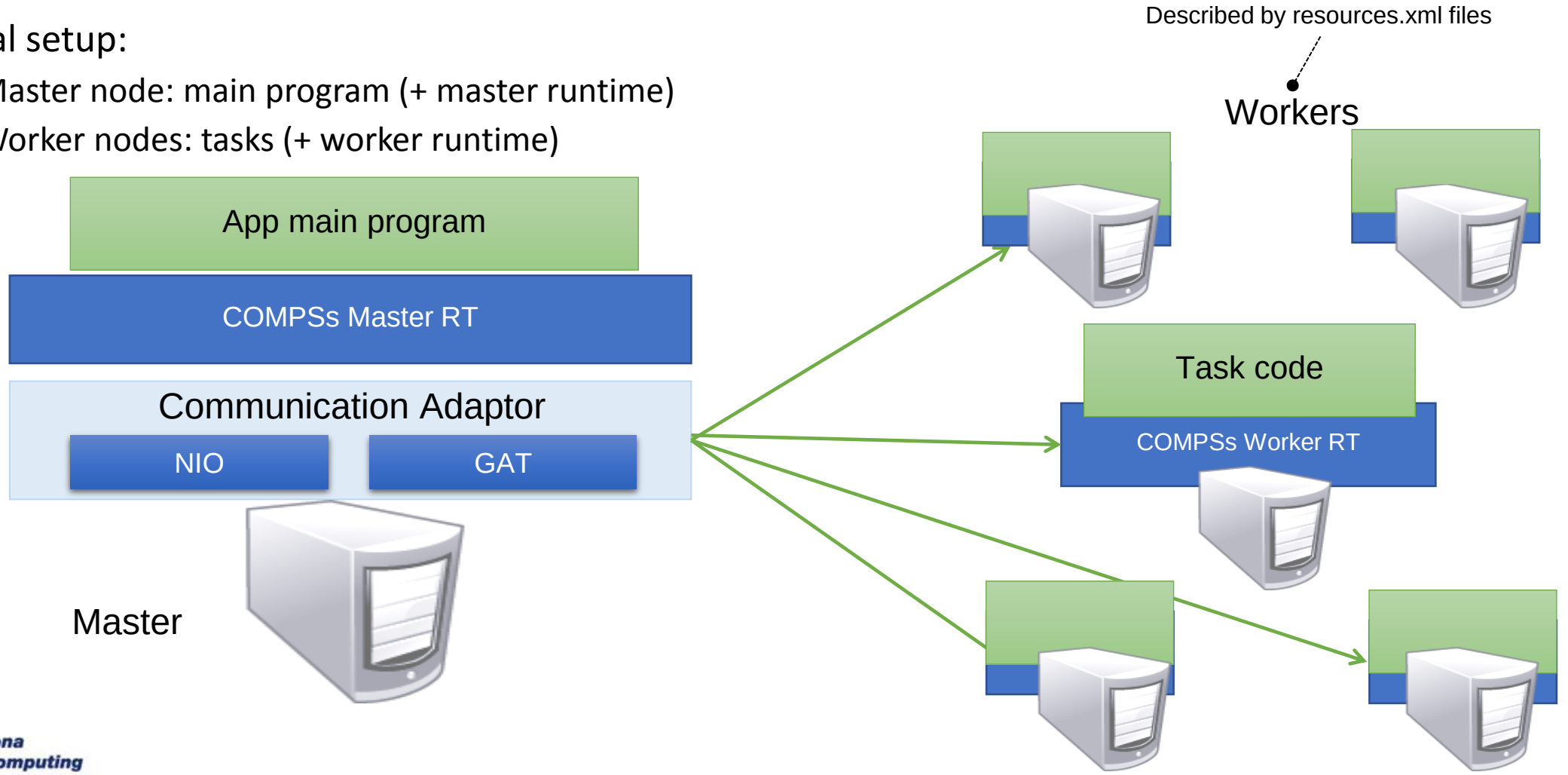
- Provide interaction with resource providers to create and destroy new computing resources

Application Exec. Desc.

- Selection of resources
- Application Code Location
- Working directory
- Provided as execution command argument

COMPSs @ Interactive Hosts

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



Configuration: Resources Specification

Resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ResourceList>
   <!--Description for any physical node-->
   <ComputeNode Name="172.20.200.18">
     <Processor Name="P1">
        <ComputingUnits>4</ComputingUnits>
        <Architecture>amd64</Architecture>
        <Speed>3.0</Speed>
     </Processor>
     <Memory>
        <Size>256.2</Size>
        <Type>Non-volatile</Type>
     </Memory>
     <Storage>
        <Size>2000.0</Size>
     </Storage>
     <OperatingSystem>
        <Type>Linux</Type>
        <Distribution>OpenSUSE</Distribution>
        <Version>13.2</Version>
     </OperatingSystem>
                              …
     <Software>
        <Application>Java</Application>
        <Application>Python</Application>
     </Software>
     <Adaptors>
        <Adaptor Name="integratedtoolkit.nio.master.NIOAdaptor">
           <SubmissionSystem>
              <Interactive/>
           </SubmissionSystem>
           <Ports>
              <MinPort>43001</MinPort>
              <MaxPort>43002</MaxPort>
           </Ports>
        </Adaptor>
     </Adaptors>
   </ComputeNode>

   <ComputeNode Name="172.20.200.19">
        …
   </ComputeNode>
<ResourceList>
```

Configuration: Project Specification

Project.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<Project>
      <!--Description of used nodes in an application and where is the application installed-->
      <ComputeNode Name="172.20.200.18">
            <InstallDir>/opt/COMPSs/</InstallDir>
            <WorkingDir>/tmp/</WorkingDir>
            <Application>
                  <AppDir>/home/user/apps/app_A/</AppDir>
                  <LibraryPath>/home/user/apps/app_A/lib</LibraryPath>
                  <Classpath>/home/user/apps/app_A/clases/</Classpath>
                  <Pythonpath>/home/user/apps/app_A/clases/py<Pythonpath>
            </Application>
       </ComputeNode>

       <ComputeNode Name="172.20.200.19">
          …
       </ComputeNode>
       ….
</Project>
```

Constraints matching

- Constraints matching mechanism
  - Enables to choose the optimal resource for each task type
- Applications describe constraints with constraint interface
- The resources description indicates resources available in each host
- Runtime does the matching before doing scheduling

Constraints matching examples

Resource.xml

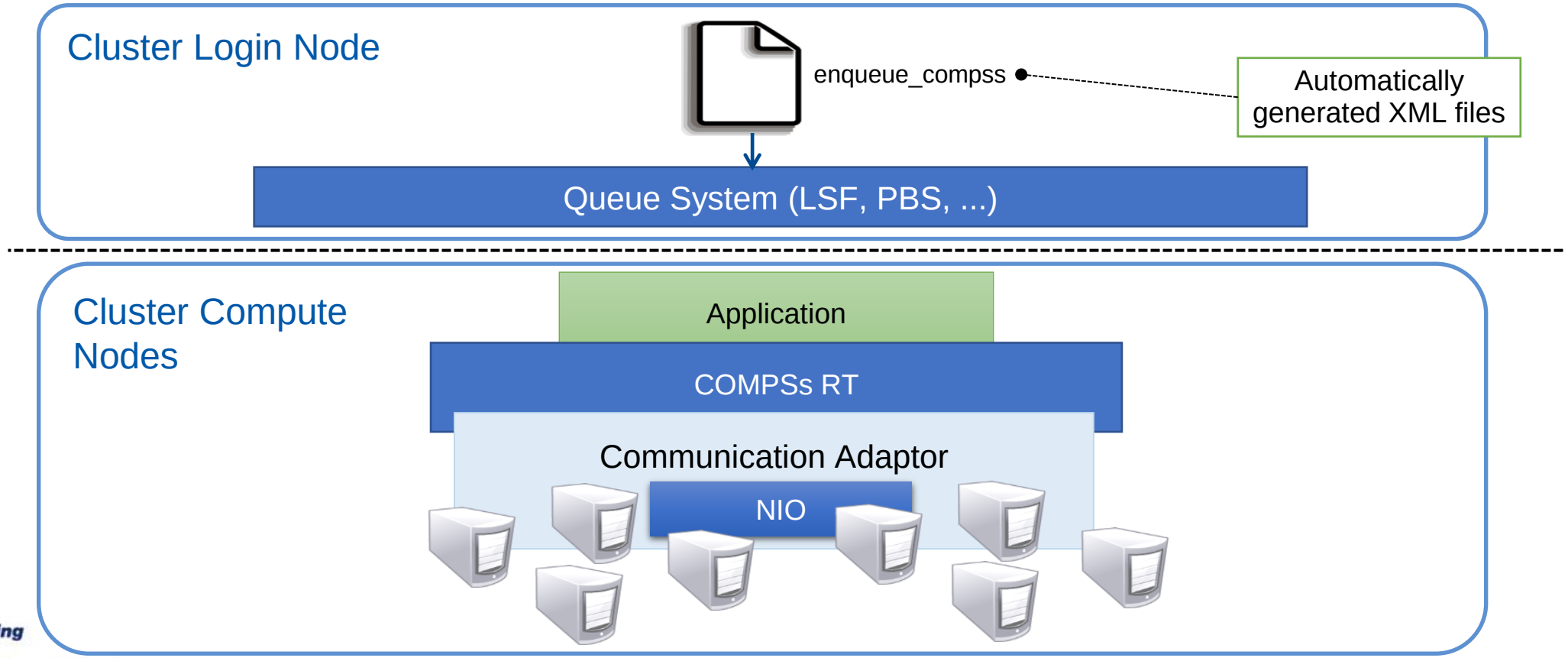
```
<ComputeNode Name="172.20.200.18">
  <Processor Name="P1">
    <ComputingUnits>4</ComputingUnits>
    <Architecture>amd64</Architecture>
    <Speed>3.0</Speed>
  </Processor>
  <Memory>
    <Size>256.2</Size>
    <Type>Non-volatile</Type>
  </Memory>
  <Storage>
    <Size>2000.0</Size>
  </Storage>
  <OperatingSystem>
    <Type>Linux</Type>
    <Distribution>OpenSUSE</Distribution>
    <Version>13.2</Version>
  </OperatingSystem>
                    …
```

Python decorator

```
@constraint(memory_size="64",
            memory_type="Non-volatile")
@task(A=INOUT, priority=True)
def potrf(A):
    A.dpotrft(lower=True)
```

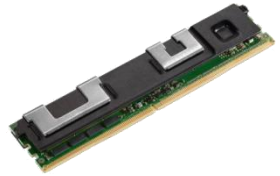
COMPSs@Cluster

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



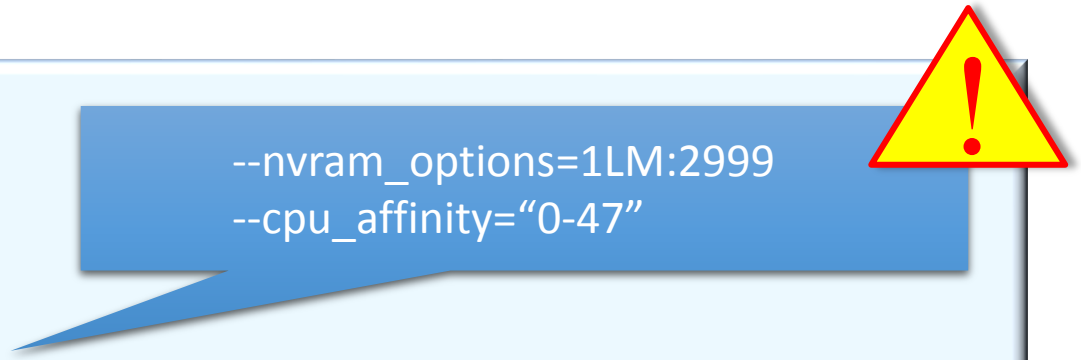
COMPSs in a Cluster (with job scheduler)

NVRAM equipped!



- Use of enqueue_compss command (instead of runcompss)
- Generates project.xml and resources.xml
- Launches application in the allocation:

```
enqueue_compss \
 --exec_time=10 \
 --num_nodes=6 \
 --worker_in_master_cpus=48 \
 --cpus_per_node=48 \
 --master_working_dir=. \
 --worker_working_dir=scratch \
 --lang=python \
 --comm=integratedtoolkit.nio.master.NIOAdaptor \
 --tracing=true \
 --graph=true \
 /home/user/src/wordcount.py /lustre/projects/user/dataset
```

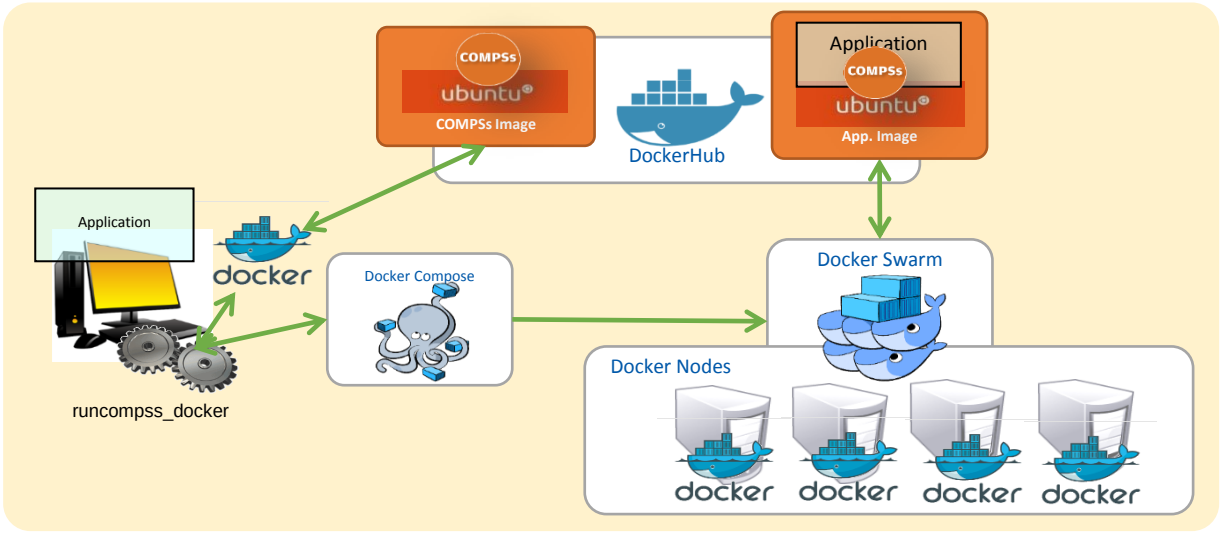


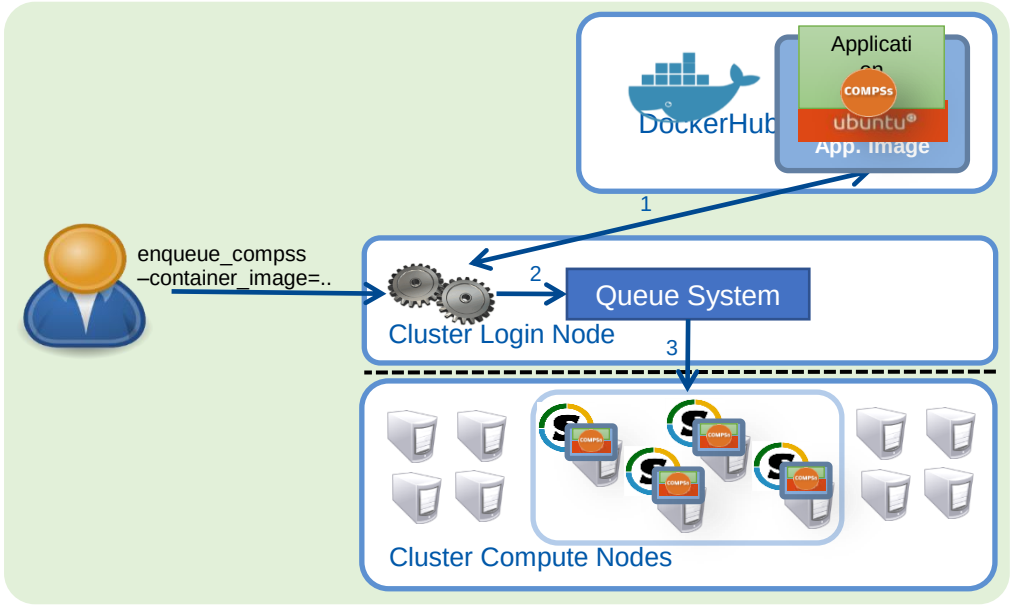
- Type "enqueue_compss" to see help and options

COMPSs …

Extra features and other environments

- Elasticity with SLURM:
  - Expand
  - Reduce
- COMPSs with Docker
  - Keep as transparent for the user as possible
    - Same as running a local COMPSs application
  - Deploy applications as a set of docker container
- COMPSs with Singularity
  - Execute applications from a container image in HPC cluster
  - Can be also used in combination with the cluster elasticity
- **And more features!!!**



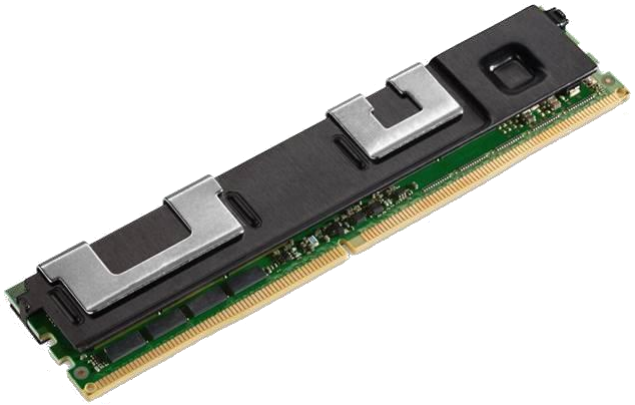


Application demo

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Kmeans @ nextgenio Cluster

- Review Kmeans source code
- enqueue_compss usage
- Job submission
- Logs



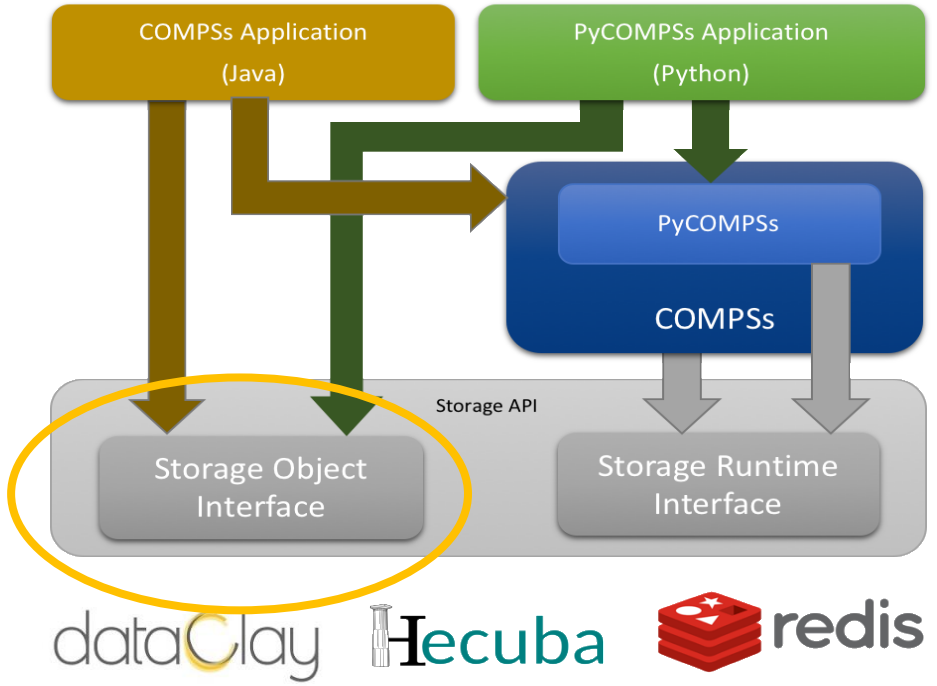
PyCOMPSs integration with Persistent Storage

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

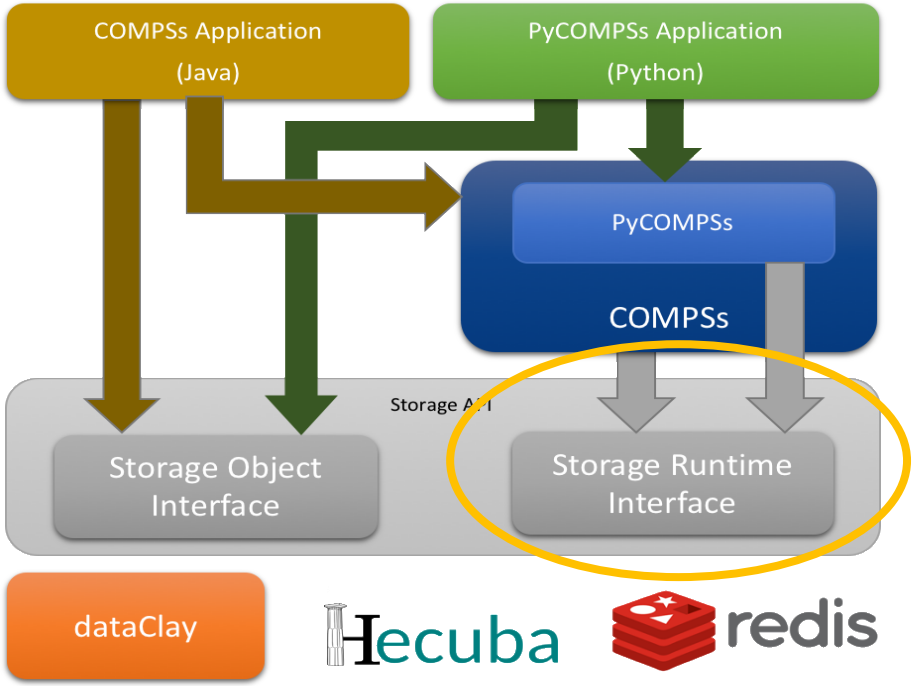
Storage Interface

Method	Python Signature
Constructor	`def __init__(self)`
Make persistent	`def make_persistent(self)` `    # Does return?`
Delete Persistent	`def delete_persistent(self)` `    # Does return?`
Get ID	`def getID(self)` `    # returns the id`

Get by Alias	`def get_by_alias(name)`
	`    # returns the object by name`
	`    # Called using the class`



Storage Interface



Method	Objective	Python API	Java API
init	Do any initialization action before starting to execute the application.	■	■
finish	Do any finalization action after executing the application.	■	■
getLocations	Retrieve the locations where a particular object is.		■
getByID	Retrieve an object from its identifier.	■	■
newReplica	Create a new replica of an object in the datastore.		■
newVersion	Create a new version of an object in the datastore.		■
consolidateVersion	Consolidate a version of an object in the datastore.		■
executeTask	Execute the task into the datastore		■
getResult	Retrieve the result of the execution into the datastore		■
TaskContext	Define a task context (task enter/exit actions)	■	

Wordcount

```
from my_collections import MyWords, MyResult...

if __name__ == "__main__":
    data = MyWords.get_by_alias("experiment1")
    r = MyResult();
    r.make_persistent("result1");
    for block in data.blocks:
        partial_result = word_count(block)
        reduce_count(r, partial_result)
    r = compss_wait_on(r)
    print r
```

```
@task(returns=dict, data=IN)
def word_count(data):
    """Unmodified word_count function"""
    map_count = defaultdict(int)
    for word in data:
        map_count[word] += 1
    return map_count
```

```
from dataclay import StorageObject
class MyWords(StorageObject):
    """
    @ClassField blocks list
    """

    @dclayMethod()
    def __init__(self):
        (…)

    …
```

Final notes

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Take-away messages

- Sequential programming approach
- Parallelization at task level
- Transparent data management and remote execution
- Can operate on different infrastructures:
  - Cluster
  - Grid
  - Cloud (Public/Private)
    - PaaS
    - IaaS
  - Containers

PyCOMPSs at SC19

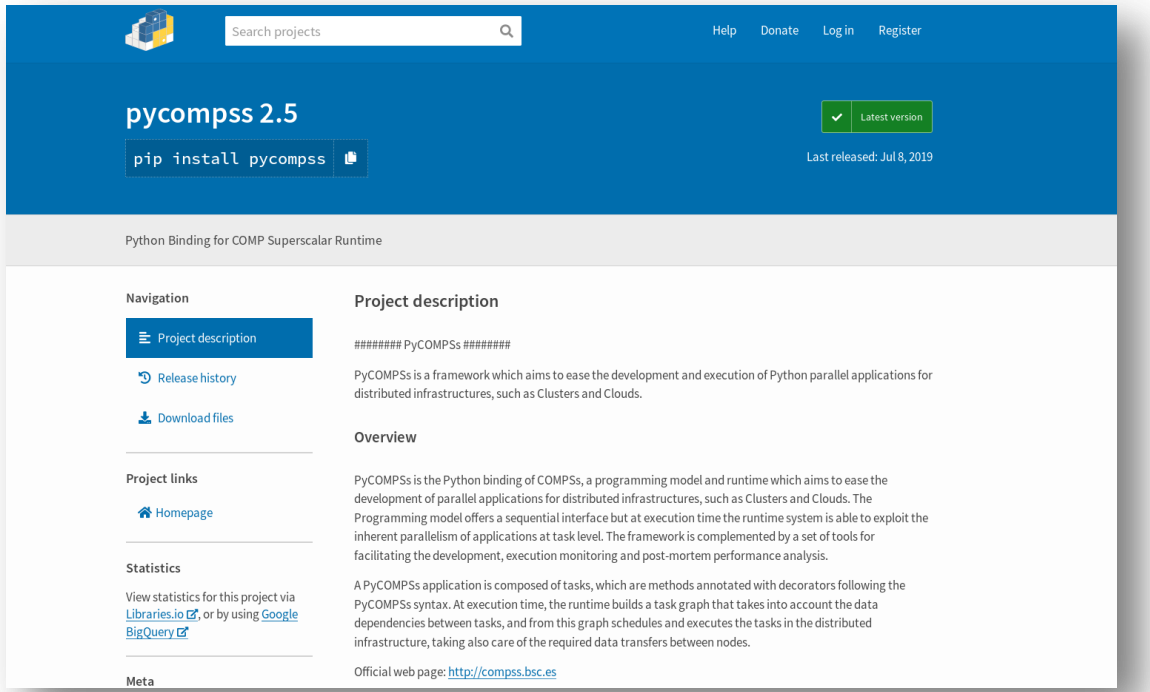
- Demos at the BSC booth (#1975):
  - Tuesday, 14:00 – 14:30: **Accelerating parallel code with PyCOMPSs and Numba**
  - Wednesday, 13:00 – 13:30: **Parallel machine learning with dislib**
  - Thursday, 11:00 – 11:30: **Fault-tolerance mechanisms for dynamic PyCOMPSs workflows**

Further Information

- Project page:
  - http://www.bsc.es/compss
- Direct downloads page:
  - https://www.bsc.es/research-and-development/software-and-apps/software-list/comp-superscalar/downloads
    - Virtual Appliance for testing & sample applications
    - Tutorials
    - Source Code
- Application Repository
  - http://compss.bsc.es/projects/bar/wiki/Applications
    - Several examples of applications developed with COMPSs

PyCOMPSs - PIP install

- Release of PyCOMPSs pip package to enable automatic installation with "pip install".
- https://pypi.python.org/pypi/pycompss/2.5/
- Documentation for the package



Projects where COMPSs is used/developed

TANGO

bioexcel
Center of Excellence for Computational Biomolecular Research

CLASS

nextgenio

EXPERTISE

LANDSUPPORT

ExaQUte
Exascale Quantification of Uncertainties for Technology and Science Simulation

FUJITSU

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA SEVERO OCHOA

Thank you

- Booth #1975 -

support-compss@bsc.es

javier.conejero@bsc.es