www.bsc.es


Barcelona Supercomputing Center
Centro Nacional de Supercomputación

# COMPSs Tutorial

February 2nd 2017, Barcelona

Adrià Aguilà, Pol Alvarez, Ramon Amela, Rosa M. Badia,
Javier Conejero, Sandra Corella, Jorge Ejarque
Daniele Lezzi, Francesc Lordan, Albert Serven,
Cristian Ramon-Cortes, Sergio Rodriguez

# Useful Information

- Slides:
  - http://compss.bsc.es/releases/tutorials/latest-tutorial.pdf
- COMPSs Virtual Appliance
  - http://compss.bsc.es/releases/vms/COMPSs-2.0-VM-tutorial.ova
  - **user:** compss
  - **password:** compss2017
- Wifi XSF:
  - **User:** xsf.convidat
  - **Password**: 2017Informatica

# Outline (Feb 2nd 2017)

- Roundtable(9:00 - 9:30): Presentation and background of participants
- Session 1 (9:30 – 11:00): Introduction to COMPSs
  - Programming model
    - Java Syntax
    - Demo: First Java example
    - Python syntax
    - Demo: First Python example
- Coffee break (11:00 – 11:30)
- Session 2 (11:30 – 13:00):
  - COMPSs execution environments
    - Demo: First example executed in MN
    - Demo: first example executed in cloud
  - Other sample codes

# Outline (Feb 2nd 2017)

- Lunch Break (13:00 -14:00)
- Session 3 (14:00 - 15:30) Hands-on I – Java
  - Virtual Machine Setup
  - Java Hands-on
    - Word-count taskified code
    - Configuration, monitoring, debugging
    - Graph generation
- Coffee break: 15:30 – 16:00
- Session 4 (16:00 – 18:00): Hands-on II – Python
  - Python Hands-on
    - Word-count without annotations
    - Annotate tasks
  - Cluster Hands-on
    - Execution in MN
    - Overview of tracing, trace analysis
- COMPSs Installation
- Final Notes

www.bsc.es


Barcelona Supercomputing Center

*Centro Nacional de Supercomputación*

# Introduction

# Motivation

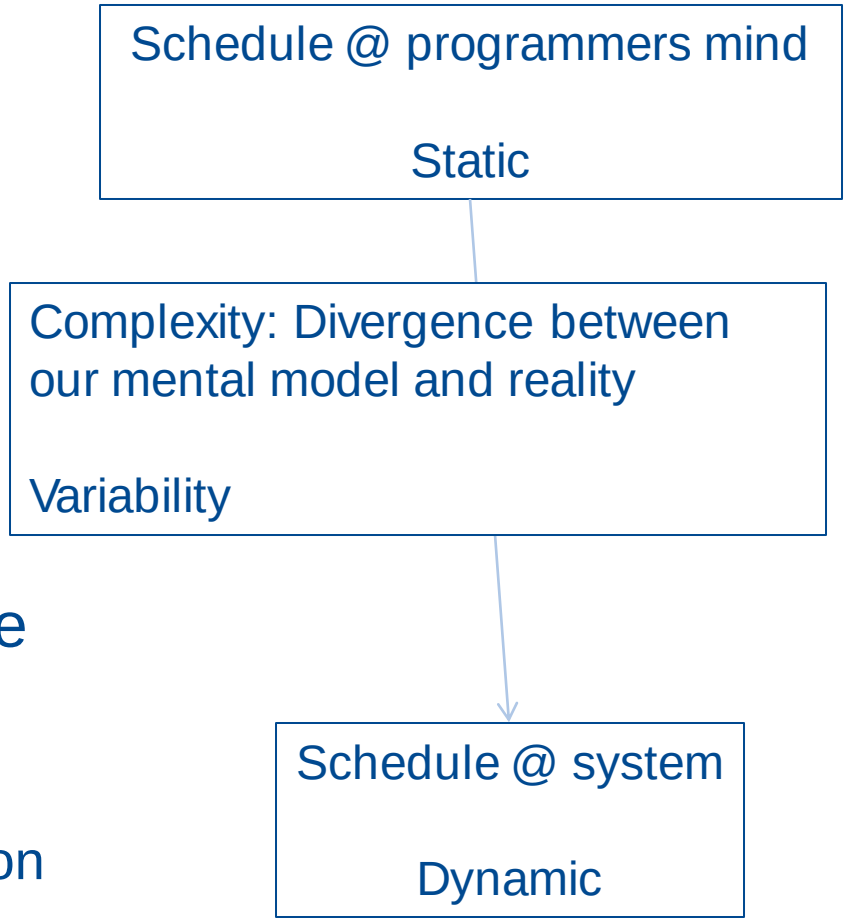
- New complex architectures constantly emerging
  - With their own way of programming them
    - Fine grain: e.g. APIs to run with GPUs, NVMs (Non-Volatile Memories)
    - Coarse grain: e.g. APIs to deploy in Clouds
  - **Difficult** for programmers
    - Higher learning curve / Time To Market (TTM)
    - 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

# Motivation

- 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

# The parallel programming revolution

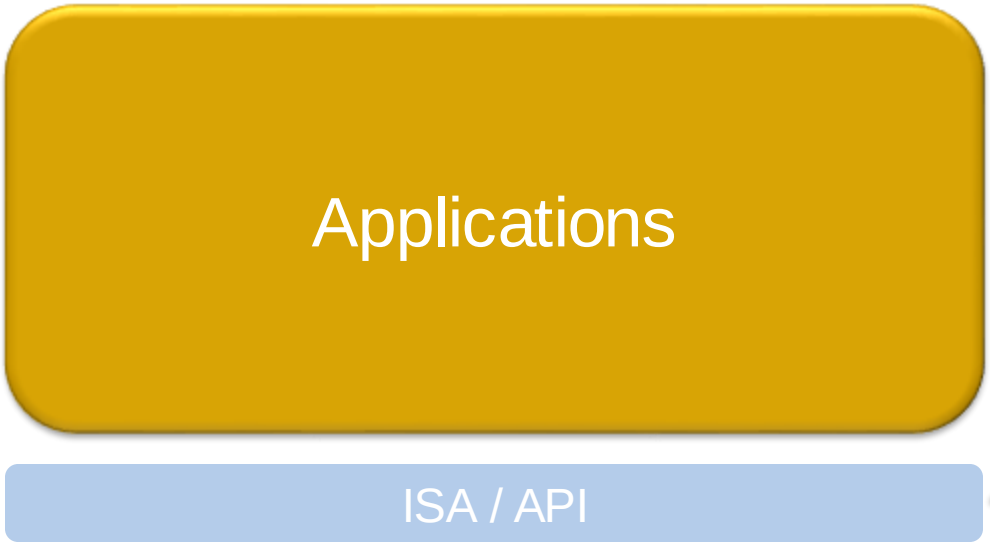
- Parallel programming in the past
  - Where to place data
  - What to run, where
  - How to communicate




- Parallel programming in the future
  - What do I need to compute
  - What data do I need to use
  - Hints (not necessarily very precise) on potential concurrency, locality,…
  - YOU PROGRAM **SEQUENTIALLY**!!!

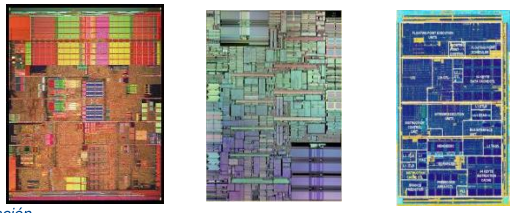
# Living in the programming revolution

- At the beginning there was one language




Programs "decoupled" from hardware

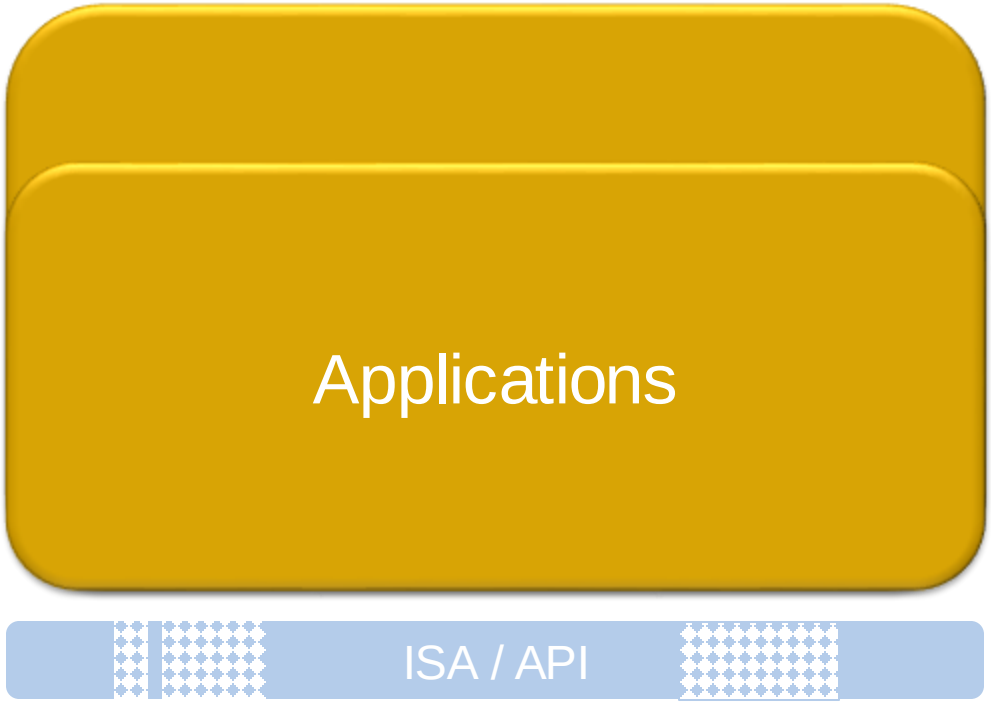
Simple interface Sequential program



ILP

# Living in the programming revolution

- Multicores made the interface to leak




Application logic
+
**Platform specificities**

Address spaces (hierarchy, transfer), control flows, …

# BSC Vision in the programming revolution (StarSs)

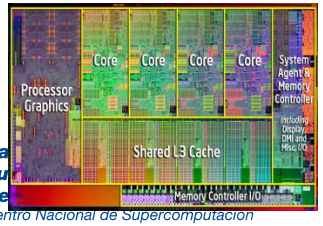
- Need to decouple again

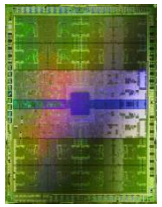
Applications

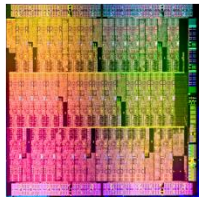
PM: High-level, clean, abstract interface

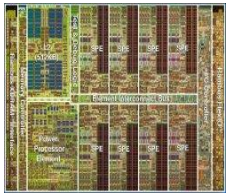
Power to the runtime

ISA / API







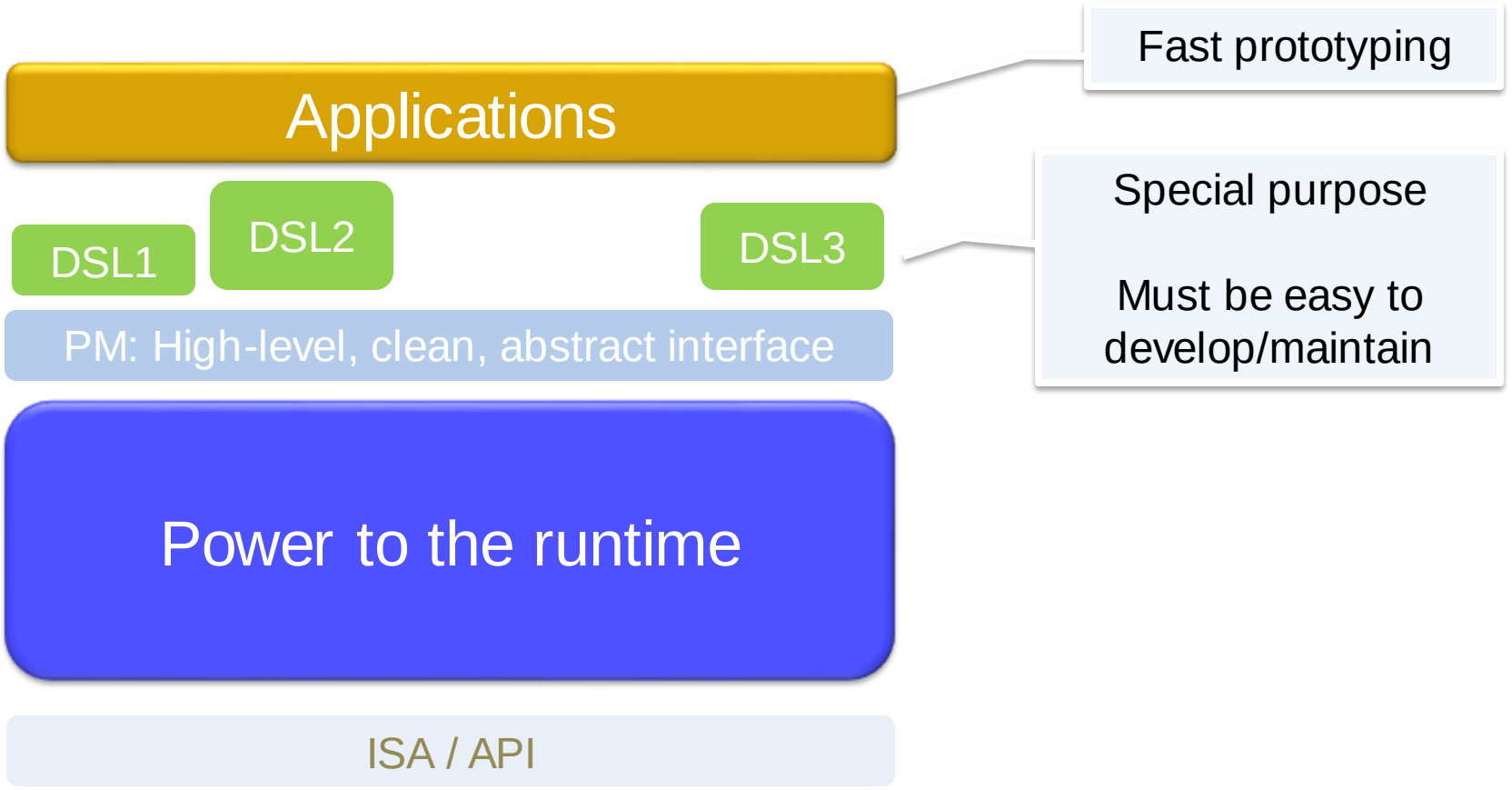
Application logic

Arch. independent

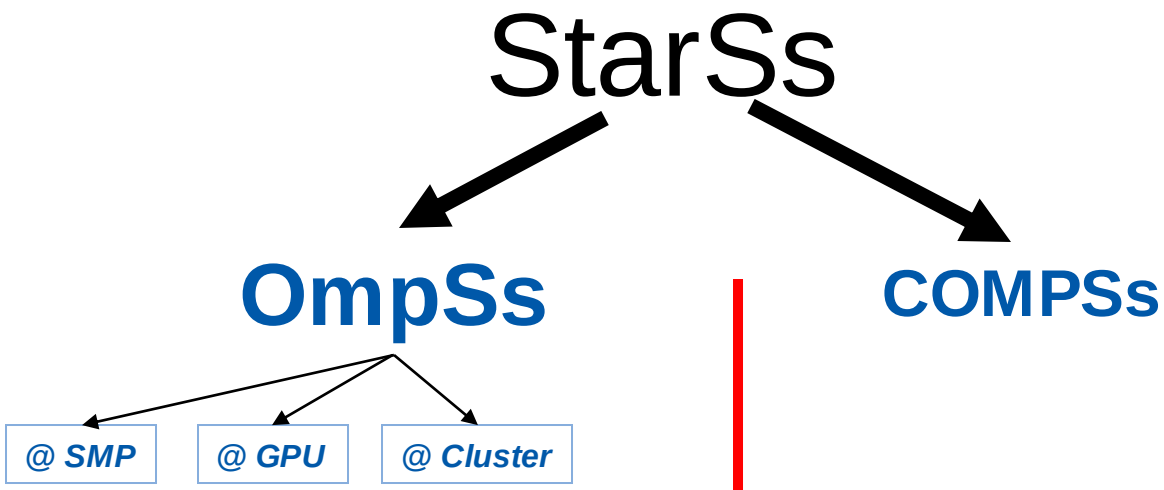
General purpose
Task based
Single address space

"Reuse" architectural ideas under new constraints (Out-Of-Order execution)

# BSC Vision in the programming revolution (StarSs)



# The StarSs "Granularities"




| | |
|---|---|
| **Average task Granularity**: | |
| 100 microseconds – 10 milliseconds | 10 ms - 1 day |
| **Address space to compute dependences**: | |
| Memory | Files, Objects, NVMs |
| **Language bindings**: | |
| C, C++, FORTRAN | Java, C/C++, Python |

**SMPs, Clusters** **Clusters, Clouds**

www.bsc.es


**Barcelona Supercomputing Center**

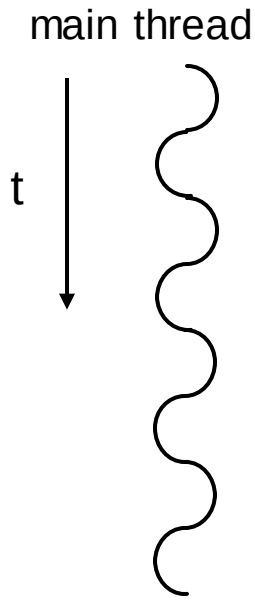
*Centro Nacional de Supercomputación*

# Let's narrow the StarSs idea…for Distributed Architectures

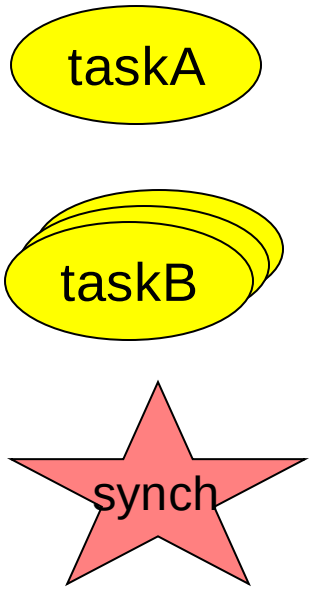
- Cluster / Cloud applications are complex to develop
  - Even more if you want to run things in parallel
  - **Goal 1: Keep a Sequential Programming Paradigm**
    - Writing an application for a computational distributed infrastructure should be as easy as writing a sequential application
  - **Goal 2: Exploit parallelism**
    - Run it as fast as possible
- Target applications: composed of tasks, most of them repetitive
  - Granularity of the tasks: enough to be distributed (simulators, …)
  - Data: files, objects, arrays and primitive types

# Programming Model: Properties (I)

- Based on sequential programming
  - No APIs, no threading, no messaging
  - No parallel constructs, no pragmas
  - Sequential consistency




```
Main Program {
   taskA(data1);

   for (int i=0; i< N; i++)
     taskB(data1, data2);

   if (condition)
     process(data2);
}
```



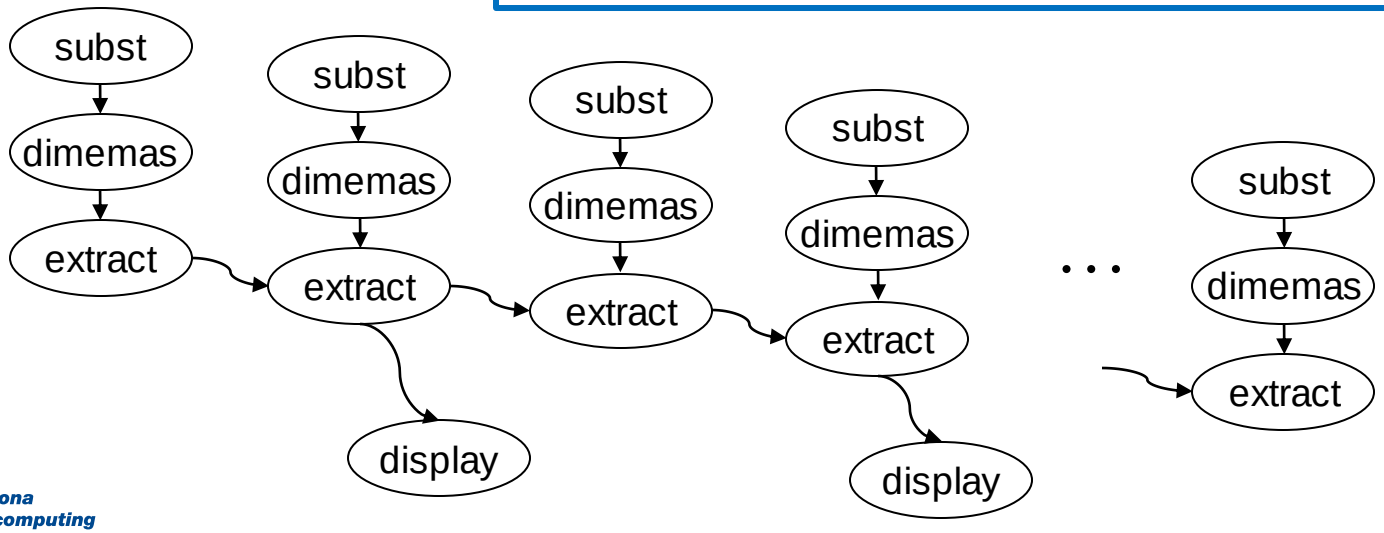
# Programming Model: Dependency detection

- Automatic on-the-fly creation of a task dependency graph

Main Program

```
for (int i = 0; i < N; i++) {
   newBWD = random();
   subst(refCFG, newBWD, newCFG);
   dimemas(newCFG, trace, dimOUT);
   extract(newBWD, dimOUT, finalOUT);
   if (i % 2 == 0) display(finalOUT);
}
```

OUT
IN
INOUT



# Runtime System

Application

Task Selection Interface

**Runtime System**


Grid


Cluster


Cloud

# Supported Features

- Basic Features:
  - Data dependency analysis
  - Data transfer
  - Task scheduling
  - Resource management
  - Results collection
  - Fault tolerance
  - Method and Web Service Tasks
- Advanced Features:
  - Shared disks support
  - Constraints based scheduling
  - Task versioning support

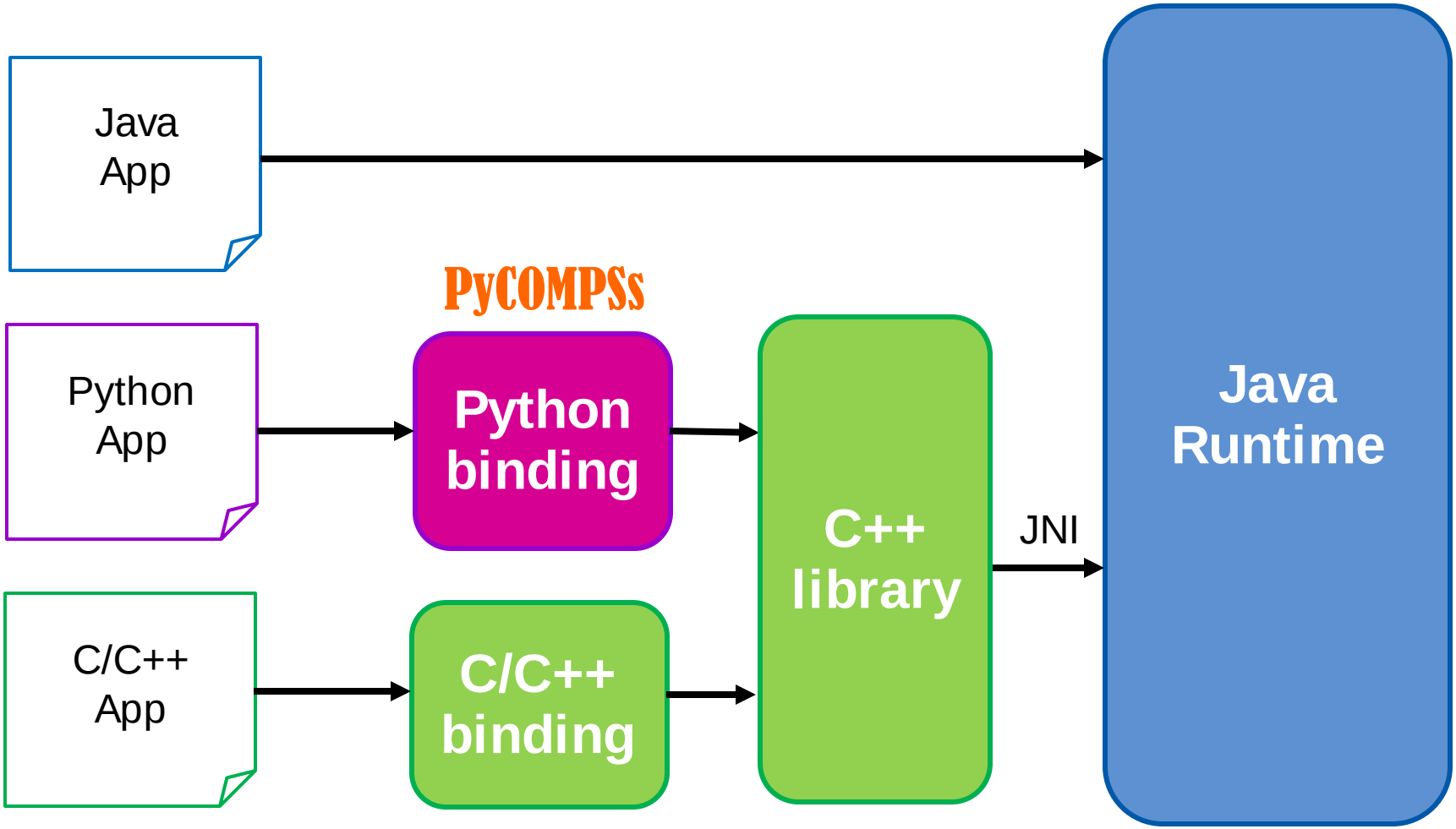
www.bsc.es


Barcelona Supercomputing Center

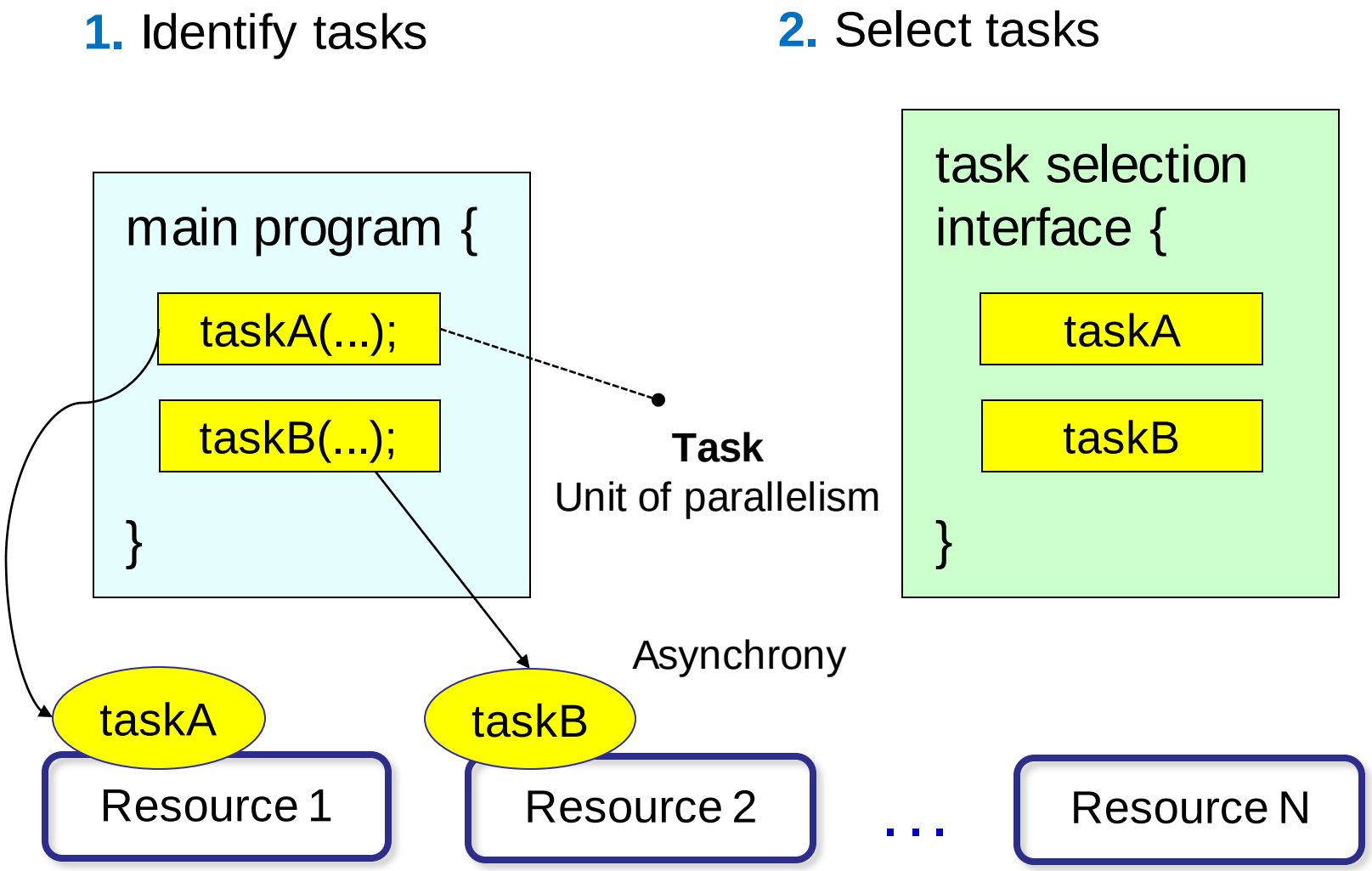
*Centro Nacional de Supercomputación*

# Java Syntax

# COMPSs Bindings



# Programming Model: Steps



# Programming Model: Task selection interface

```
public interface SampleItf {
   @Constraints(computingUnits = "1", memorySize = "0.5f")
   @Method(declaringClass = "servicess.Example")
    void myMethod(
        @Parameter(direction = INOUT)
        Reply r
    );

   @Service(namespace = "http://servicess.es/example",
            name = "SampleService",
            port = "SamplePort")
    Reply myServiceOp(
        @Parameter(direction = IN)
        Query q
    );
}
```

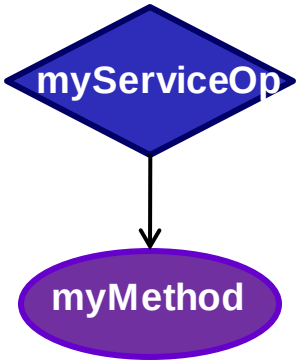
# Programming Model: Regular Main program

```
public class App {

   public static void main(String[] args) {
       Query query = new Query(…);
       Reply reply = myServiceOp(query);

       myMethod(reply);

       reply.printToLog();
   }
}
```

Service task call

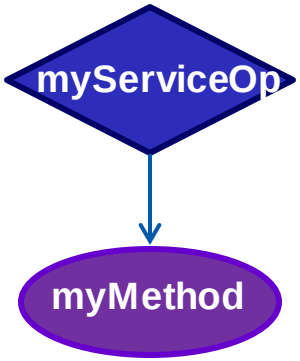
Method task call

Synchronization

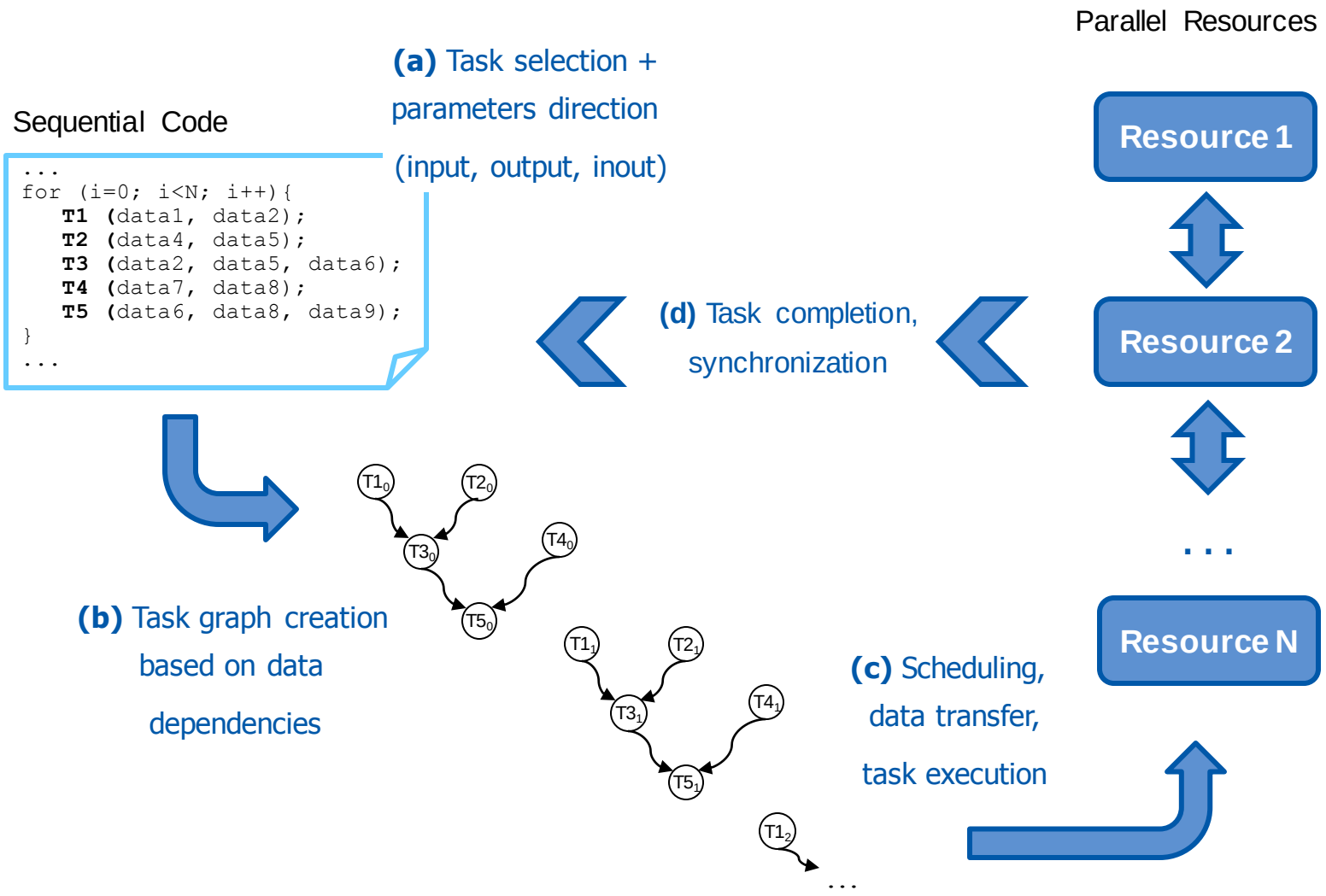


# Programming Model: Service Operation

```
@WebService
public class ServiceApp {
   @Orchestration
   public static void sampleComposite() {
       Query query = new Query(…);
       Reply reply = myServiceOp(query);

       myMethod(reply);

       reply.printToLog();
   }
}
```



# Programming Model: Summary




# JAVA EXAMPLE

# Programming Model: Sample Application

- Main Program

```
public static void main(String[] args) {
        String counter1 = args[0], counter2 = args[1],
               counter3 = args[2];

        initializeCounters(counter1, counter2, counter3);

        for (i = 0; i < 3; i++) {
          increment(counter1);
          increment(counter2);
          increment(counter3);
        }
}
```

- Subroutine

```
public static void increment(String counterFile) {
        int value = readCounter(counterFile);
        value++;
        writeCounter(counterFile, value);
}
```

# Programming Model: Sample App (Interface)

- Task Annotation Interface

```
public interface SimpleItf {

        @Method(declaringClass = "SimpleImpl")
        void increment(
                @Parameter(type = FILE, direction = INOUT)
                String counterFile
        );

}
```

**Implementation**

**Parameter metadata**

# Programming Model: Sample App (Main Program)

- Main program NO CHANGES!

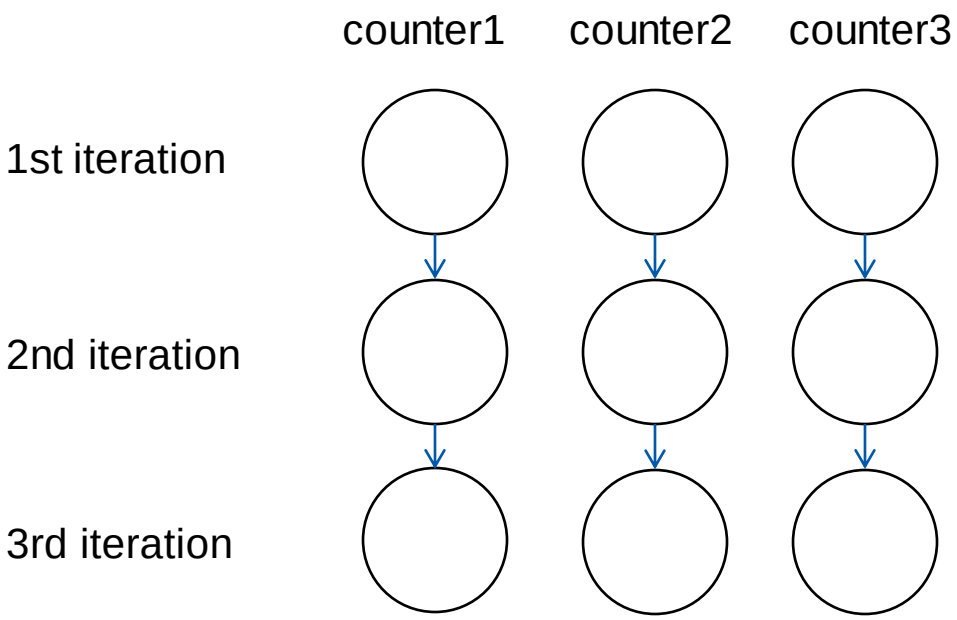
```
public static void main(String[] args) {
        String counter1 = args[0], counter2 = args[1],
               counter3 = args[2];

        initializeCounters(counter1, counter2, counter3);

        for (i = 0; i < 3; i++) {
          increment(counter1);
          increment(counter2);
          increment(counter3);
        }
}
```

# Programming Model: Task Graph

## Main Loop

```
for (i = 0; i < 3; i++) {
        increment(counter1);
        increment(counter2);
        increment(counter3);
}
```

## Task Graph




# Python Syntax

# Why Python?

- Python is powerful... and fast;
  plays well with others;
  runs everywhere;
  is friendly & easy to learn;
  is Open. *

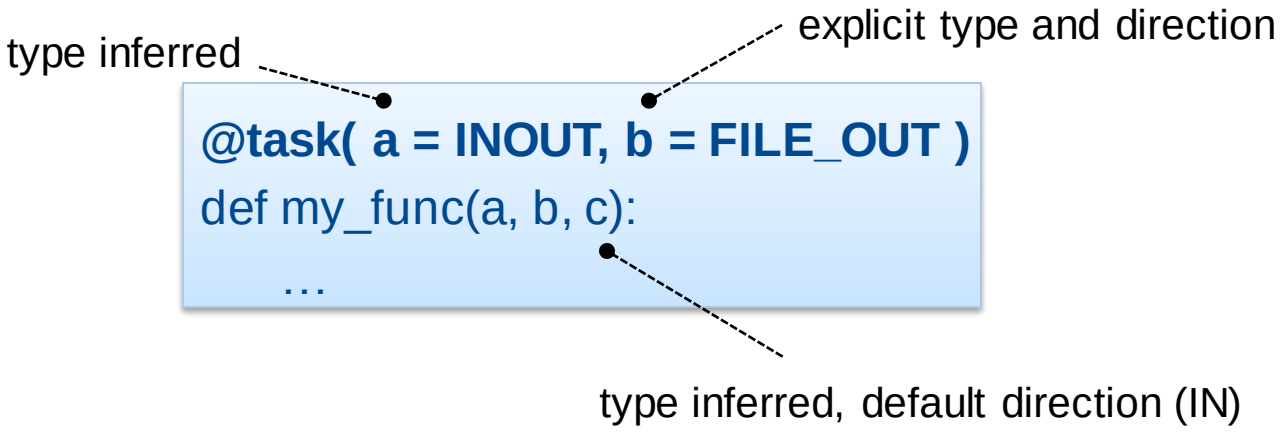

- Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in languages such as C
- Large community using it, including scientific and numeric
- Object-oriented programming and structured programming are fully supported
- Large number of software modules available (38,000 as of January 2014)



* From python.org

# PyCOMPSs: Task definition

- Task definition with Python decorators
  - Provide information about task parameters (*TYPE_DIRECTION*):
    - Type
      - Only mandatory for files
      - Inferred for the rest of the types
    - Direction
      - Default IN (read-only)
      - Mandatory for INOUT (read-write) and OUT (write-only)



# PyCOMPSs: Task definition (II)

- The @task decorator: special arguments
  - Type of the return value → mandatory if a value is returned

```
@task(returns = int)
def ret_func():
    return 1
```

  - The funcion may return more than one value:

```
@task(returns = (int, list))
def ret_func():
    return 1, [2, 3]
```

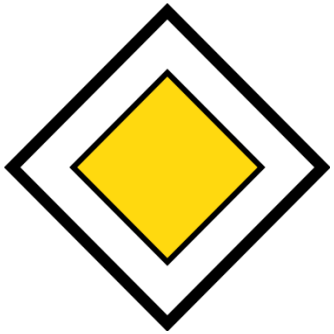
# PyCOMPSs: Task definition (III)

- The @task decorator: special arguments
  - Does the task modify the callee object? → default True

```
class MyClass(object):
    @task(isModifier = False)
    def instance_method(self):
        … # self is NOT modified here
```

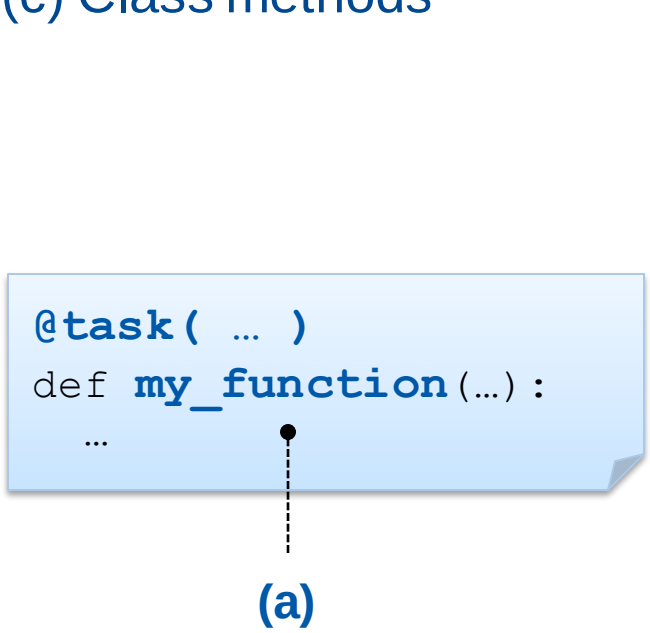

  - Is it a priority task? → Default False

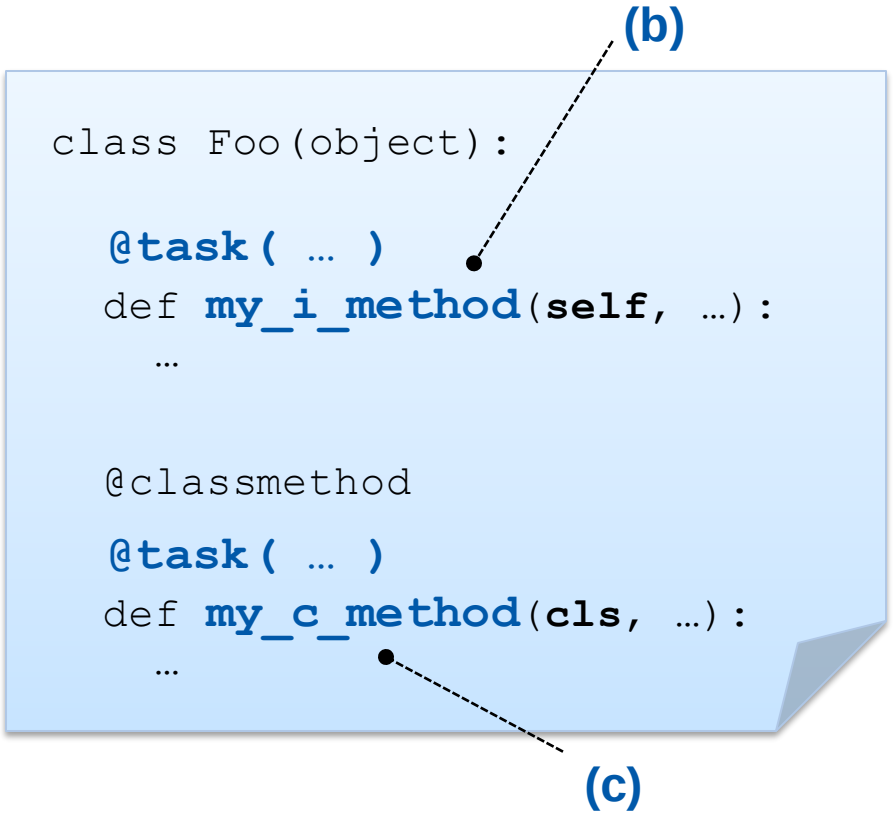
```
@task(priority = True)
def prio_func():
    …
```



# PyCOMPSs: Task types

- What can be selected as a task?
  - (a) Functions
  - (b) Instance methods
  - (c) Class methods



# PyCOMPSs: Main program → Synchronization API

- Data created or updated by a task can be used in the main program of the application
  - But we need to synchronize first!

- Three API methods for synchronization
  - *compss_open* → files

```
my_file = 'file.txt'
func(my_file)
fd = compss_open(my_file)
…
```

*func* is a task that modifies *my_file*

  - *compss_wait_on* → objects

```
my_obj = MyClass()
my_obj.method()
my_obj = compss_wait_on(my_obj)
…
```

*method* is a task that modifies *my_obj*

  - *waitForAllTasks()*
    - *Barrier - does not synchronize data → useful for measuring time*

# PyCOMPSs: Main program → Future objects

- Mechanism to make asynchronous those tasks that return a value
  - Synchronization is only triggered when necessary
- The future object is a representative of the object yet to be generated

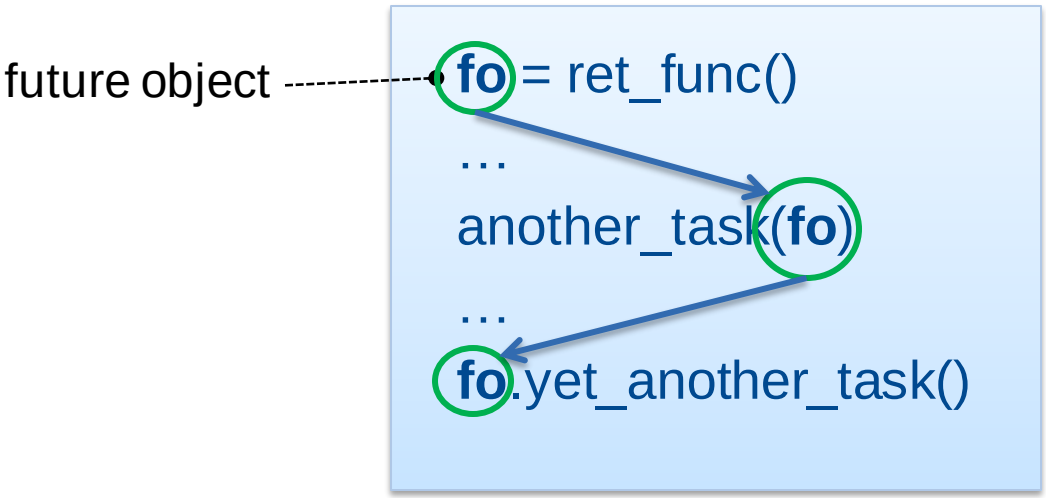
```
@task(returns = MyClass)
def ret_func():
    return MyClass(…)
```

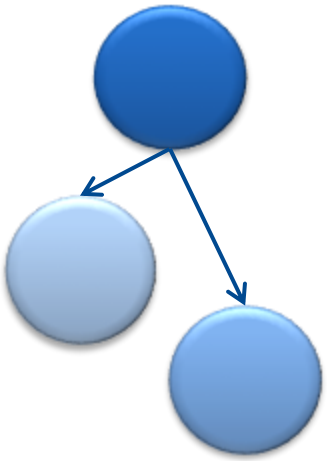
```
if __main__==‘__main__’:
    o = ret_func()
```

future object

# PyCOMPSs: Main program → Future objects (II)

- A future object can be involved in a subsequent task call
  - PyCOMPSs will automatically enforce the dependency



- Synchronization from main program (same as other objects):

```
fo = ret_func()
…
out = compss_wait_on(fo)
```

# PyCOMPSs Constraints

- Enables definition of tasks' constraints
  - Resource to execute the task should meet the constraint
- Decorator definition:
  - @constraint(constraint1="value1", constraint2="value2", …)
- Examples of supported constraints:
  - ProcessorArchitecture
  - ComputingUnits
  - MemorySize
  - AppSoftware

```
@constraint(ComputingUnits="8")
@task(A=INOUT, priority=True)
def potrf(A):
    A = dpotrf(A, lower=True)[0].tolist()
```

# PyCOMPSs: Wrap-up example

- Invoke tasks as Python functions/methods
- API for data synchronization

Main Program

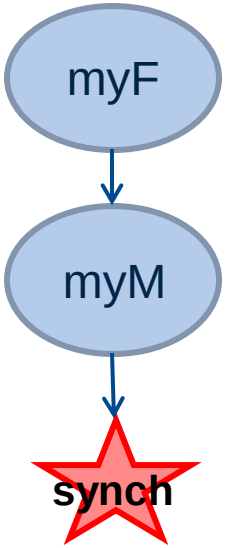
```
foo = Foo()
myFunction(foo)
foo.myMethod()
…
foo = compss_wait_on(foo)
foo.bar()
```

- Task selection in function definition (decorators)

Function definition

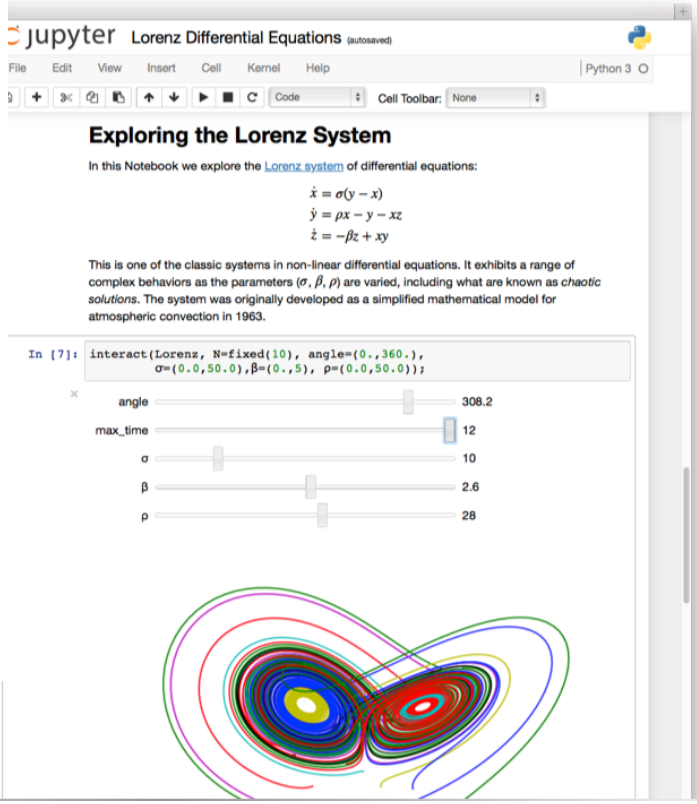
```
@task( par = INOUT )
def myFunction(par):
  …
```

```
class Foo(object):
  @task()
  def myMethod(self):
    …
```



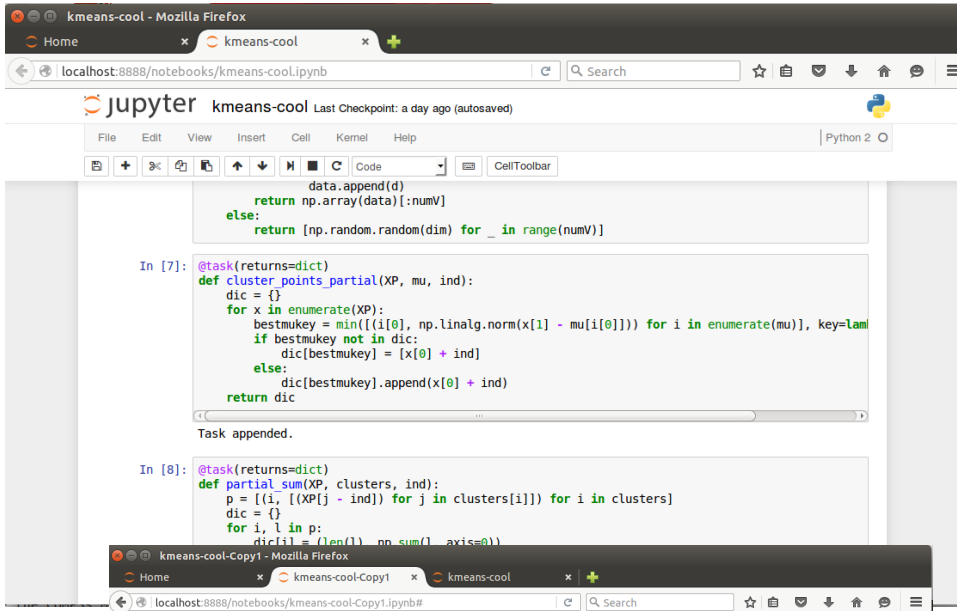
# Jupyter notebook

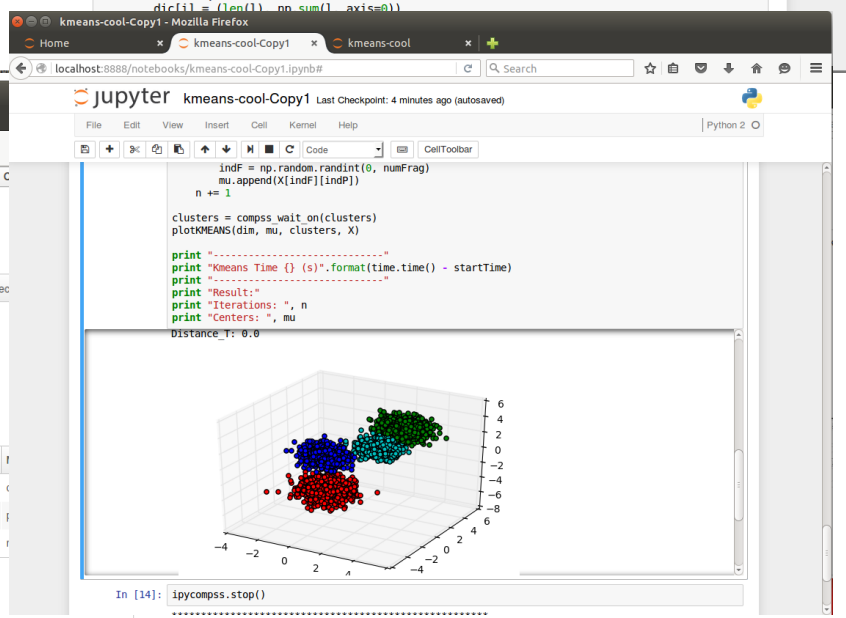
- The Jupyter Notebook is a web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.
- Uses include: data cleaning and transformation, numerical simulation, statistical modeling, machine learning and much more.
- Runs Python –sequential
- Prototype of PyCOMPSs integrated with Jupyter notebook
  - Runs in parallel in local node and can offload tasks to external nodes

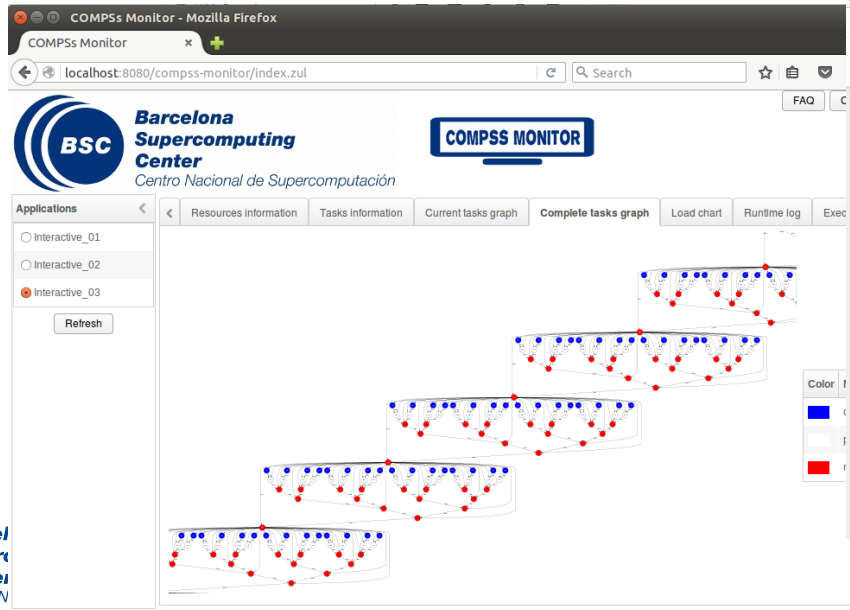



# PyCOMPSs @ Jupyter notebook

- Runtime started from notebook
- PyCOMPSs tasks registered and send to workers
- Apps can be configured to generate trace, graph and to be monitored



# Demo Python

- Using Jupyter notebook and Monitor
- First example: simple example
  - Goal: show PyCOMPSs syntax and how to execute in the notebook
  - Step by step, describe each code block
  - Execute with the monitor
- Second example:
  - Goal: show an example with dependences
  - Describe each code block
  - Execute and show the task-graph in the monitor
  - Plot the results

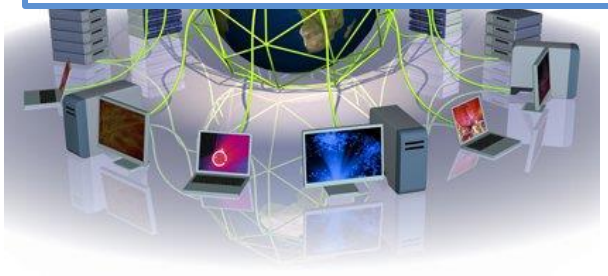

# COMPSs Execution Environments

# Runtime System

Application

Task Selection Interface

How can I select the execution platform?

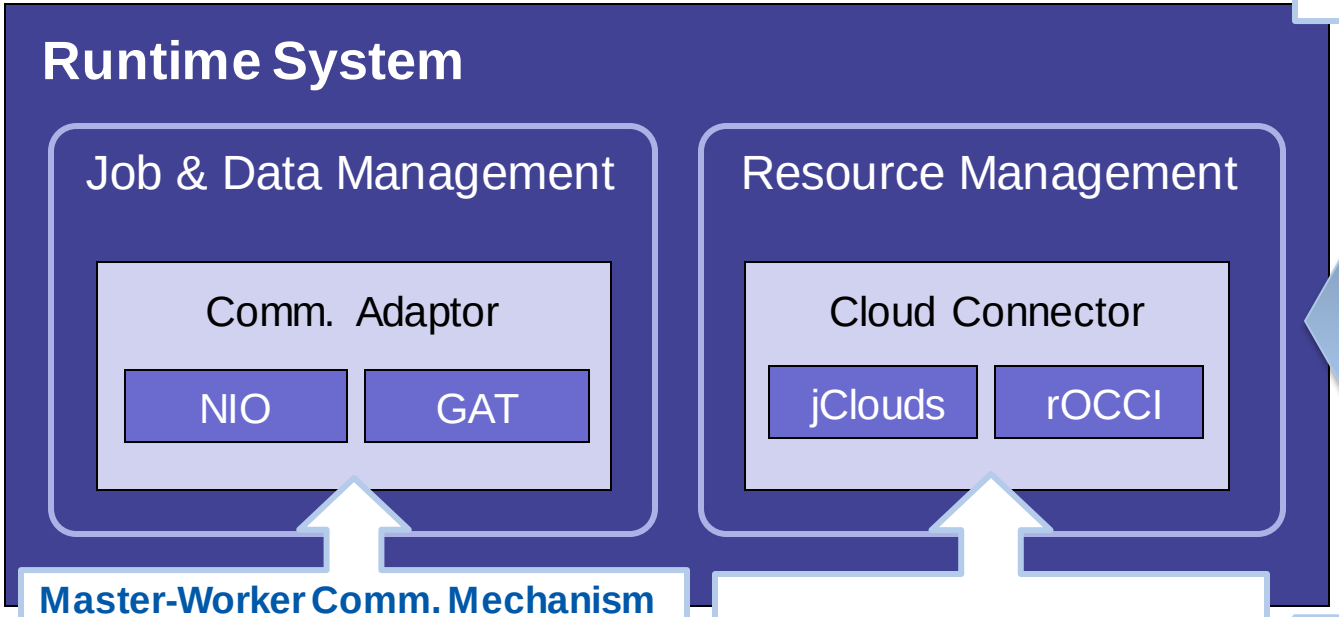


Grid

Cluster


Cloud

# Execution Environment Configuration Overview




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

resources.xml

project.xml

**Master-Worker Comm. Mechanism**
- GAT: Restrictred environments (only ssh access) and Grid Middleware
- NIO: Efficient Persistent workers implementation
- Controlled and secured environments
- Provided as execution command argument

**Resource Scalability**
- jClouds: access to most of commercial public clouds
- rOCCI: OGF standard
- Extensible (support others..)
- Described in the resources and project files

**Application Exec. Description**
- Selection of resources
- Application Code Location
- Working directory
- Provided as execution command argument

# Basic Execution Examples

- Remote host
  - Demo Matrix Multiplication execution in bscgrid06
- Clusters
  - Demo Matrix Multiplication execution in MN
- Cloud
  - Demo Matrix Multiplication execution in Google Compute Engine

# Demo application: Block Matrix multiplication

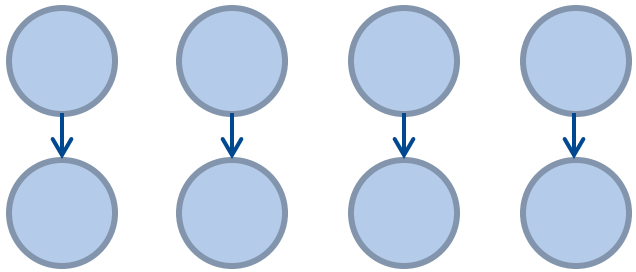
## Code

```
//Main Code
for (int i = 0; i < MSIZE; i++){
        for (int j = 0; j < MSIZE; j++){
                for (int k = 0; k < MSIZE; k++){
                        multiplyAccumulative(C[i][j], A[i][k], B[k][j] );
                }
        }
}
```

```
//Task Code
public static void multiplyAccumulative ( Block c, Block a, Block b )
{
        for( int i = 0; i < c.bRows; i++ )
                for( int j = 0; j < c.bCols; j++ )
                        for ( int k = 0; k < c.bCols; k++ )
                                c.data[i][j] += a.data[i][k] * b.data[k][j];
}
```

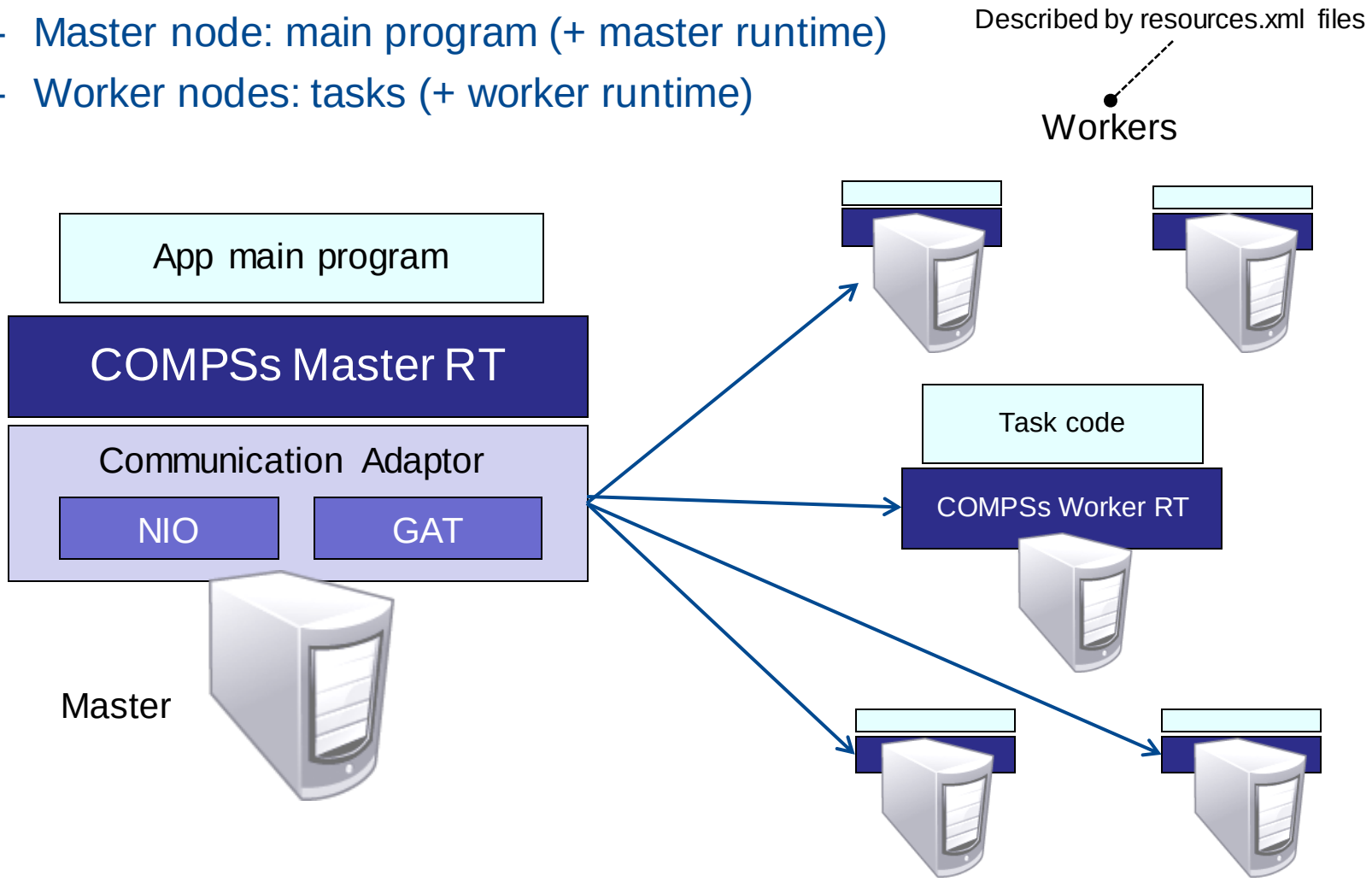
## Example

- Task dependency graph for matrices of 2x2 blocks
- Dynamically Generated



# COMPSs in remote hosts (interactive)

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



# Configuration: Resources Specification

## Resources.xml

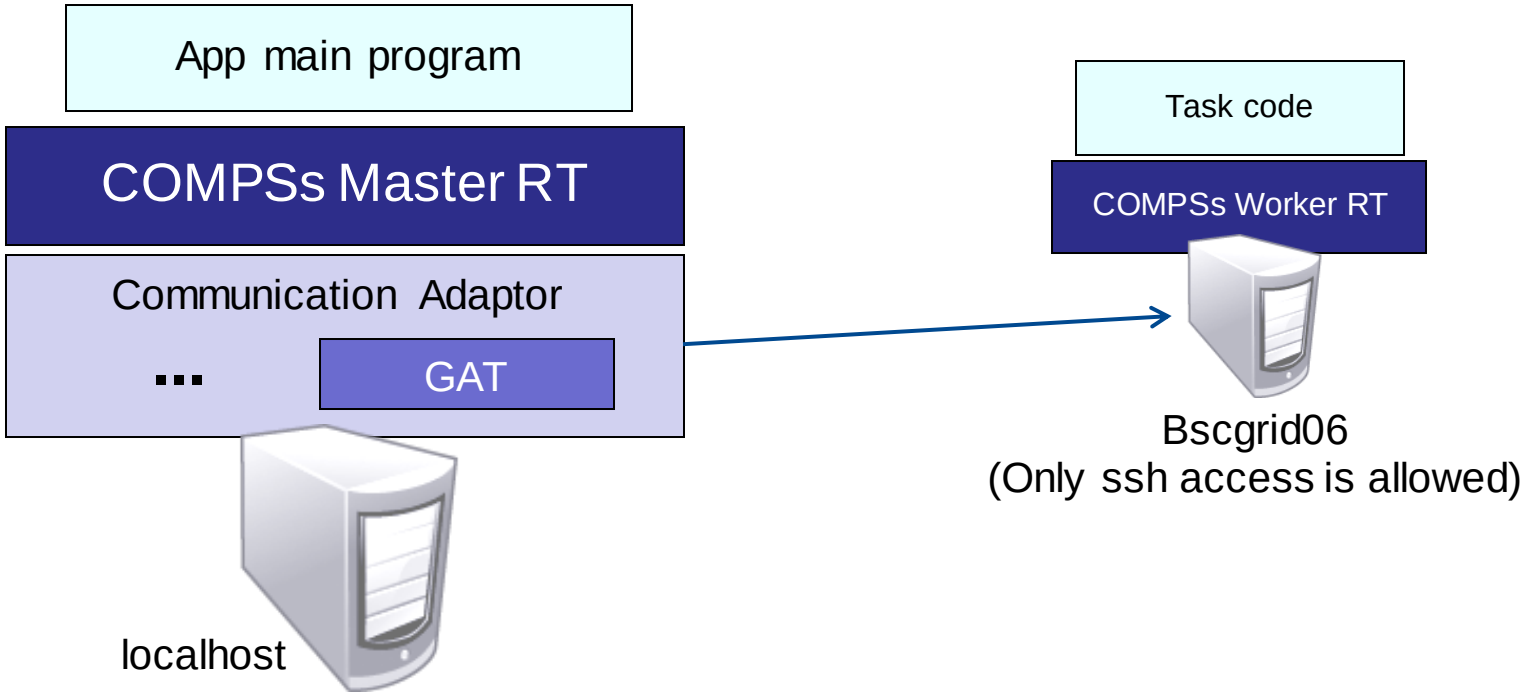
```
<?xml version="1.0" encoding="UTF-8"?>
<ResourceList>
    <!--Description for any physical node-->
    <ComputeNode Name="172.20.200.18">
      <Processor Name="P1">
          <ComputingUnits>4</ComputingUnits>
          <Architecture>amd64</Architecture>
          <Speed>3.0</Speed>
      </Processor>
      <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>
       </CompuNode>

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

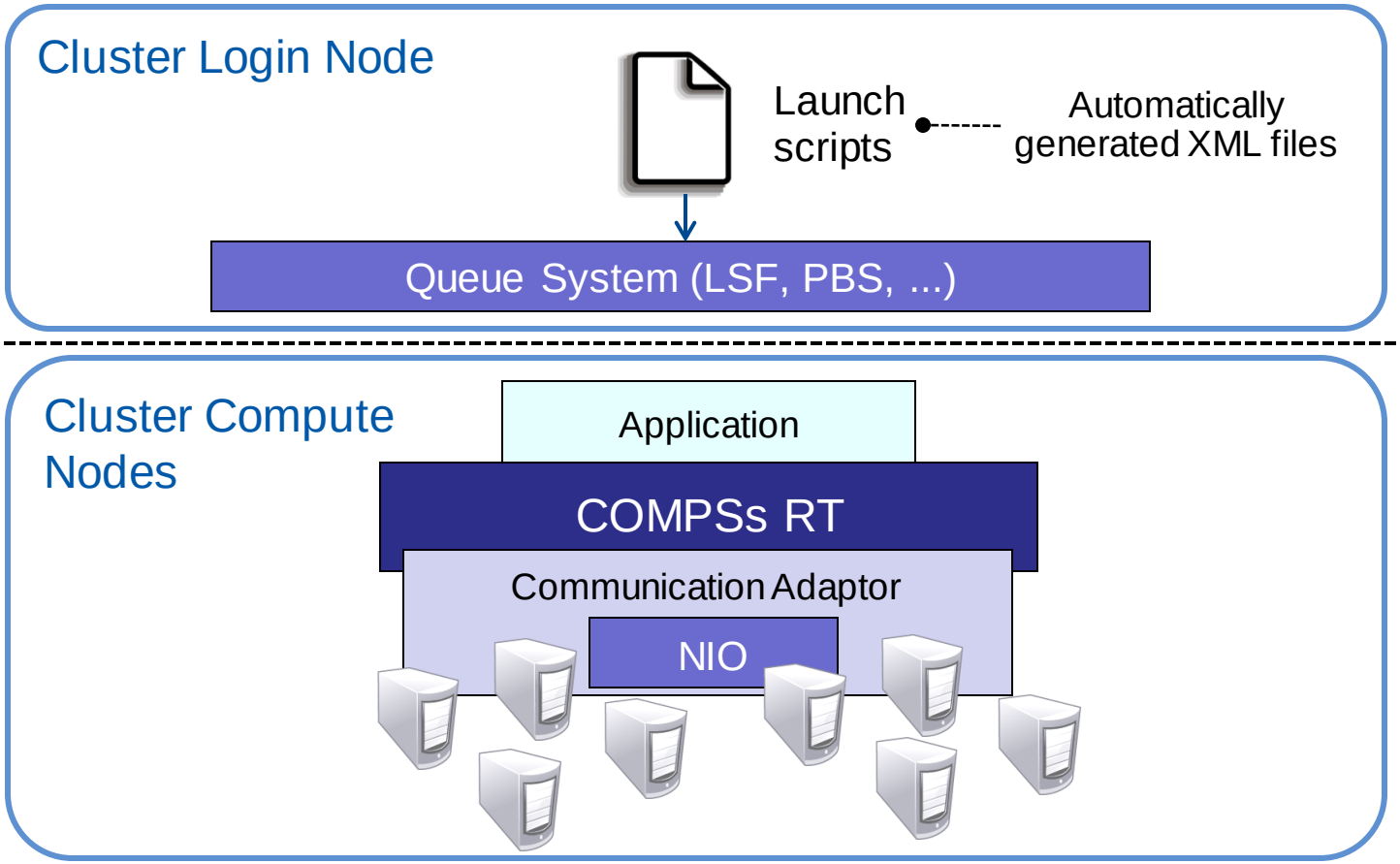
# DEMO: COMPSs using Remote hosts (interactive)




- Demo:
  - Deploy code in worker
  - Run the application with specific resources and project.xml and GAT the adaptor

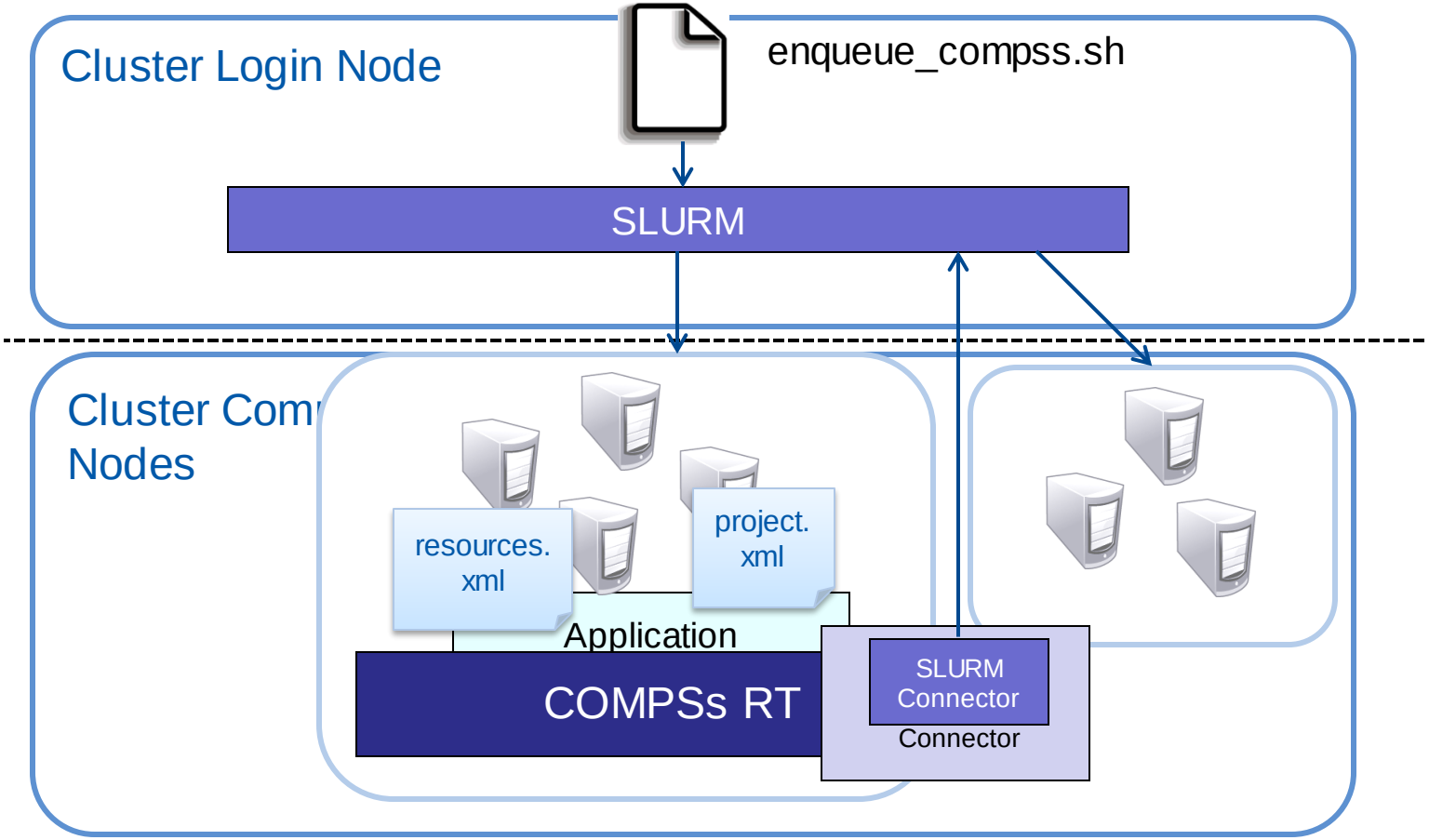
# COMPSs in a Cluster (queue system)

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

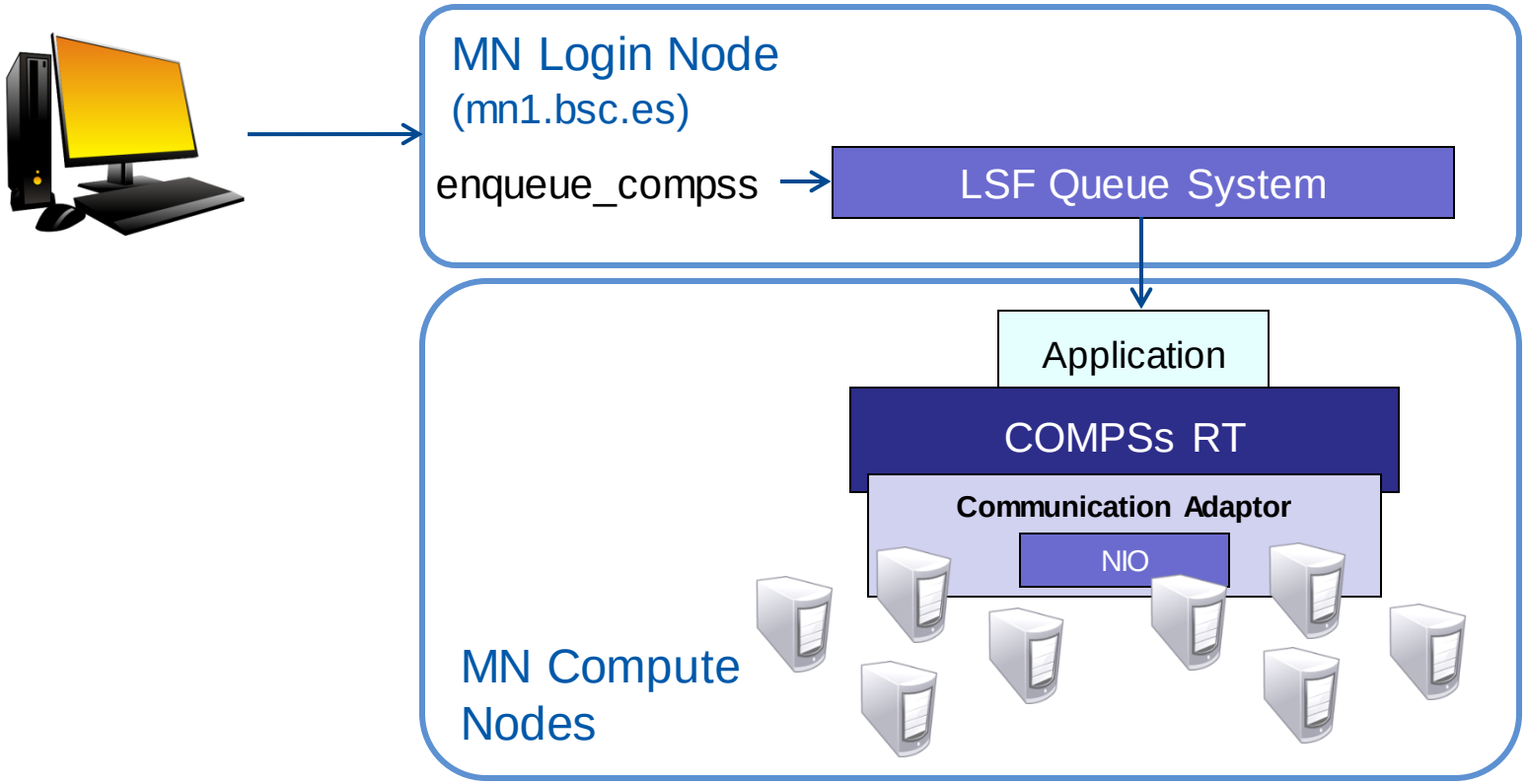


# COMPSs in a Cluster (queue system)

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



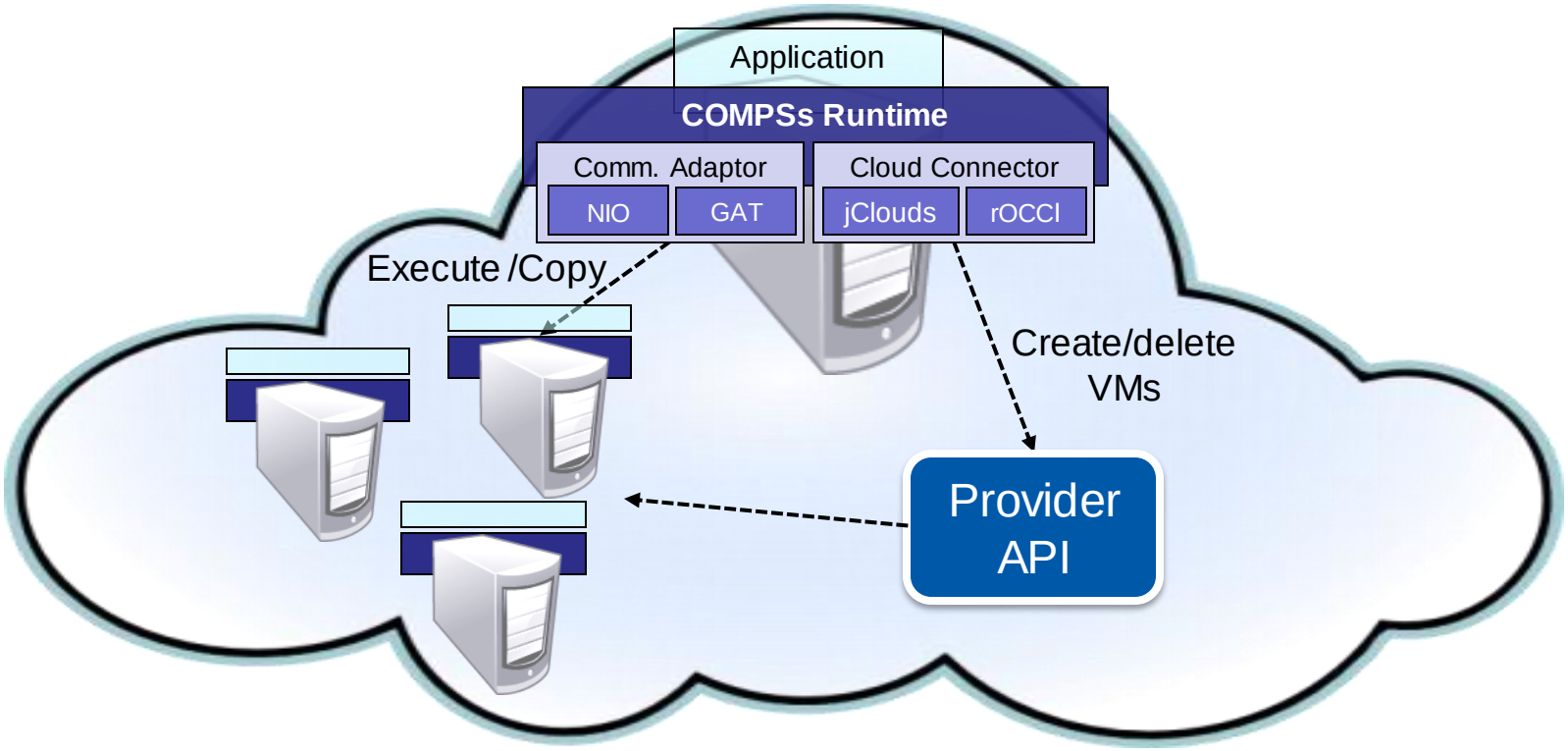
# COMPSs in a Cluster (queue system)




- DEMO:
  - Deploy app in MN
  - Connect login node
  - Launch “enqueue_compss” script requesting the number of nodes

# COMPSs in Clouds

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



# Cloud Configuration: Resources Specification

## Resources.xml

```
<ResourceList>
    <CloudProvider name="BSCCloud">
        <Endpoint>
            <Server>https://bscgrid20.bsc.es:11443</Server>
            <ConnectorJar>con-rocci.jar</ConnectorJar>
            <ConnectorClass>es.bsc.conn.rocci.ROCCI</ConnectorClass>
        </Endpoint>
        <Images>
                <Image name="debianbase">
                    <CreationTime>120</CreationTime>
                    <Adaptors>…
                    <OperatingSystem>...
                    <Software>…
                </Image>
                ..
        </Images>
        <InstanceTypes>
                <InstanceType Name="bsc.small">
                    <Processor>...
                    <Memory>…
                </InstanceType>
                  …
        </InstanceTypes>
    </CloudProvider>
</ResourceList>
```

# Cloud Configuration: Project Specification

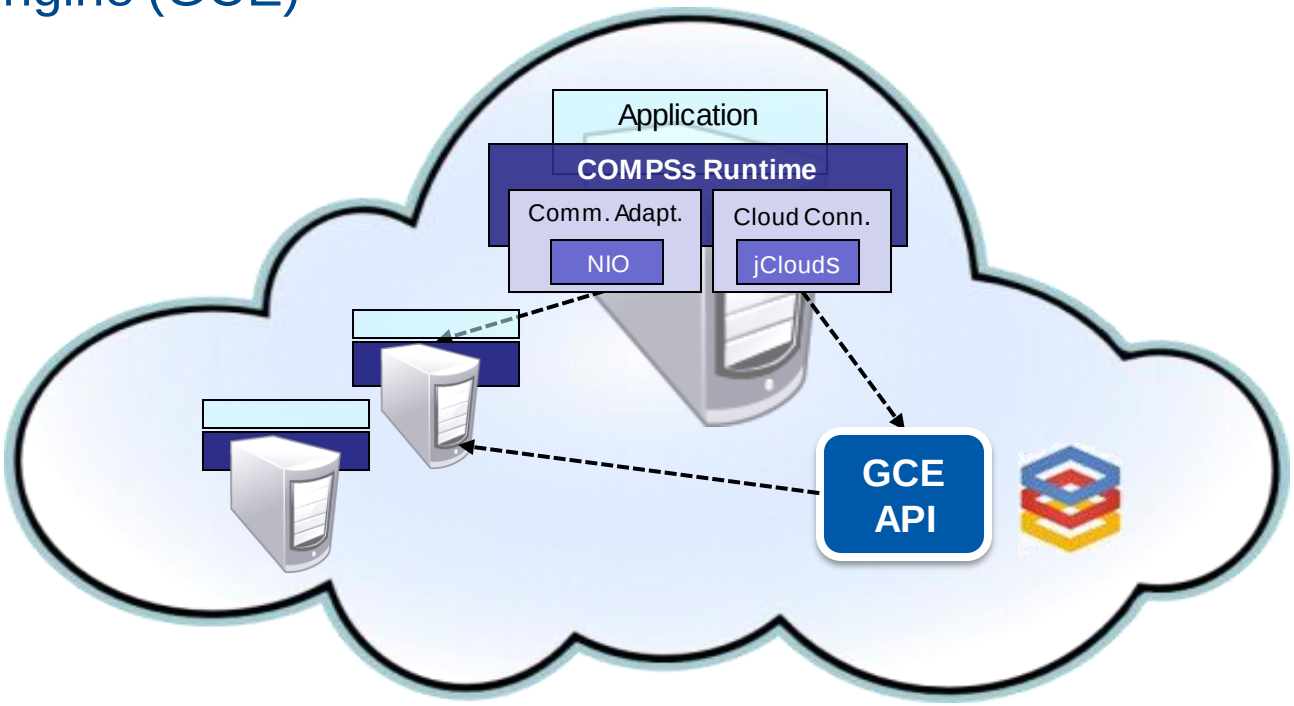
## Project.xml

```
<Project>
 <Cloud>
        <InitialVMs>0</InitialVMs>
        <minVMCount>2</minVMCount>
        <maxVMCount>5</maxVMCount>
        <Provider name="BSCCloud">
            <LimitOfVMs>5</LimitOfVMs>
            <Property>
                <Name>user-cred</Name>
                <Value>/home/.../cert.pem</Value>
            </Property>
            <Property>
                <Name>user</Name>
                <Value>userbsc</Value>
            </Property>
    ...
```

```
...
            <ImageList>
                <Image name="debianbase">
                    <InstallDir>/opt/COMPSs/</InstallDir>
                    <WorkingDir>/tmp/</WorkingDir>
                    <Application>
                        <AppDir>/home/user/AppName</AppDir>
                    <User>user</User>
                    <Package>
                        <Source>/home/…/AppName.tar.gz</Source>
                        <Target>/home/user/</Target>
                    </Package>
                </Image>
            </ImageList>
            …
            <InstanceTypes>
                <InstanceType name="bsc.small"/>
            </InstanceTypes>
        </Provider>
    </Cloud>
</Project>
```

# DEMO: COMPSs in Clouds

- Execution of COMPSs applications in Google Compute Engine (GCE)



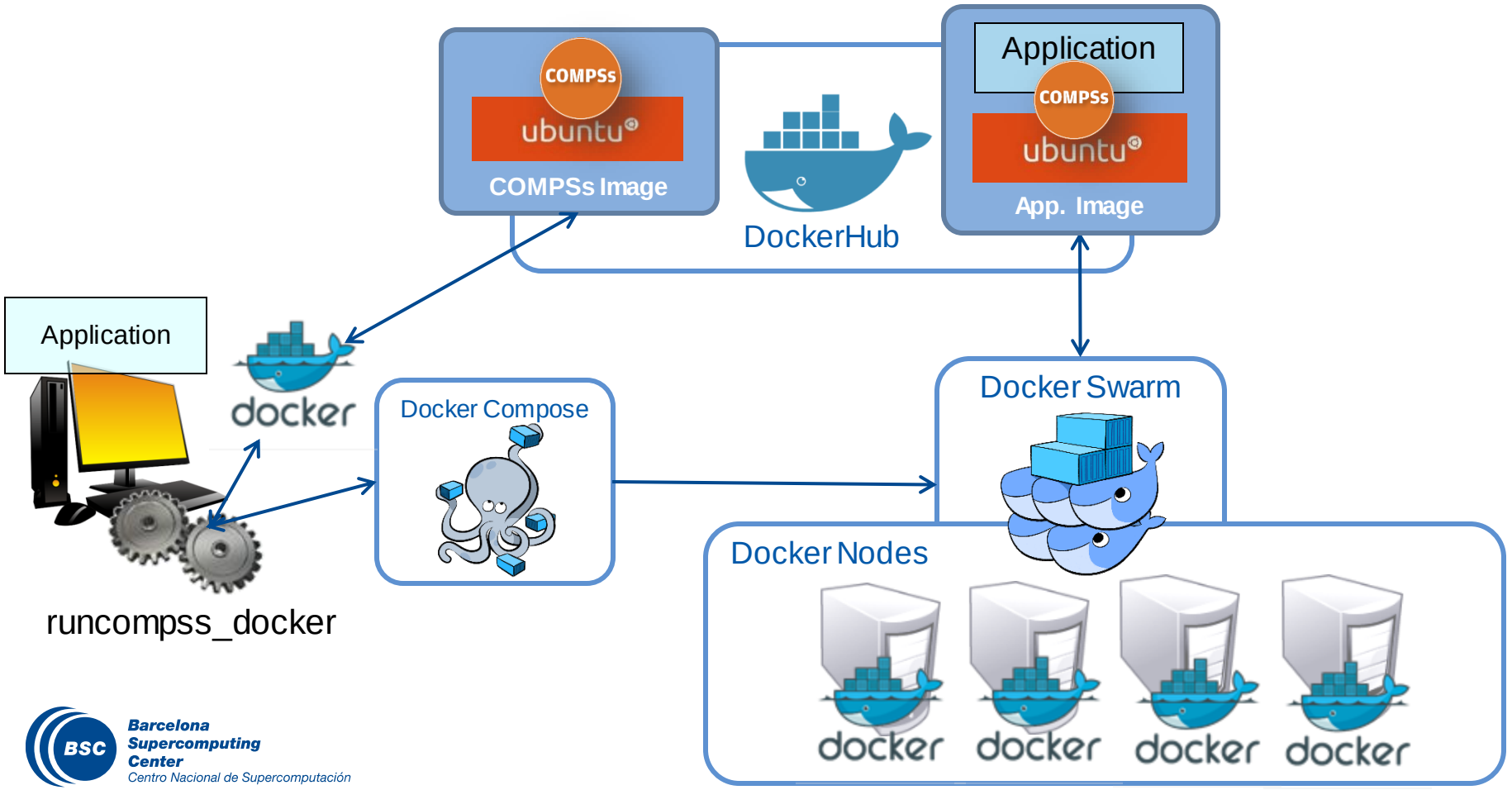

https://www.youtube.com/watch?v=XGaqUje_2zY

# Advanced Usage Examples

- COMPSs with Docker
- Combine Different Environment
  - Cloud Bursting
  - Multiple Grids
- COMPSs for scaling Web Service
- Real Applications implemented with COMPSs

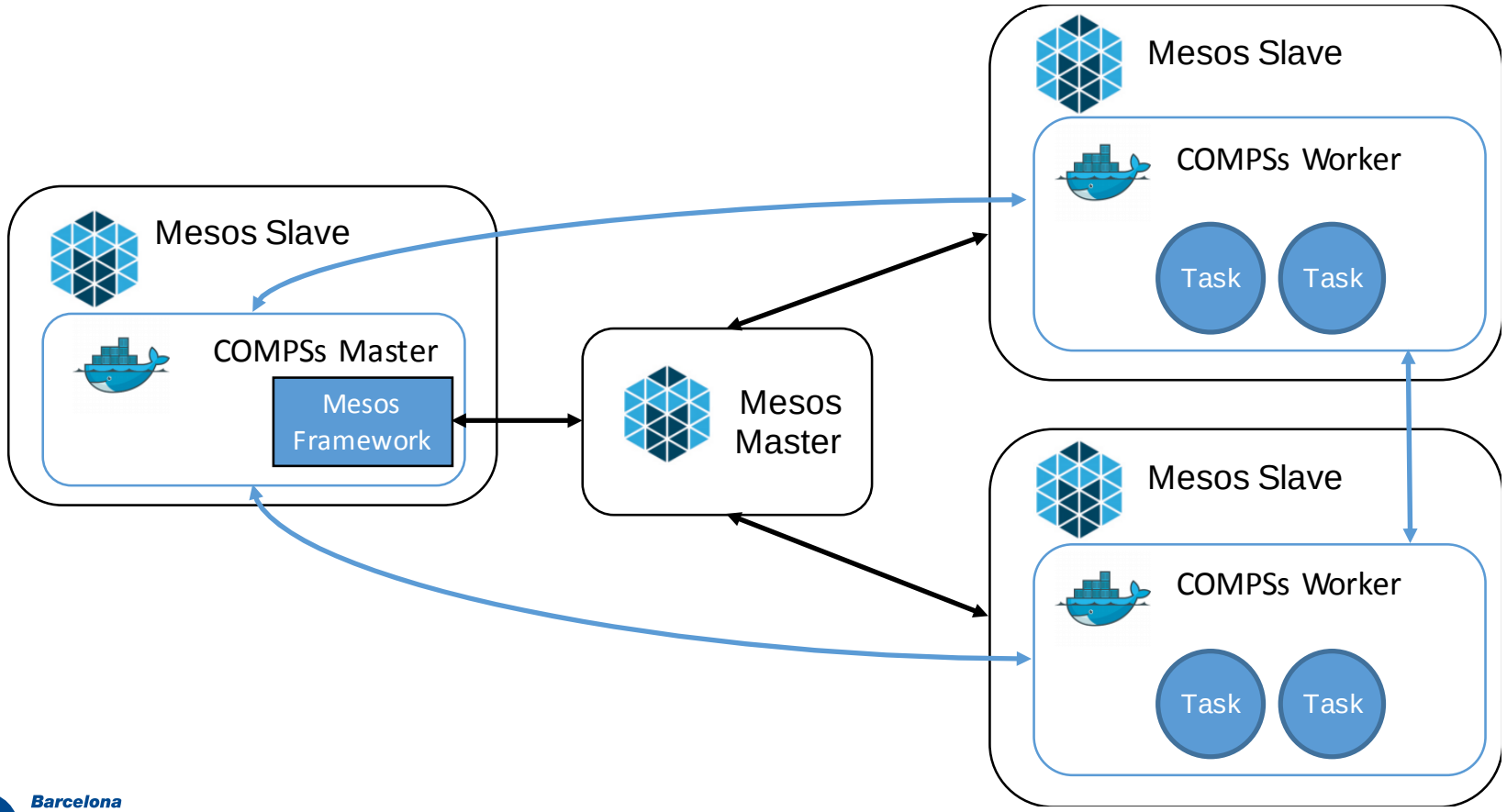
# COMPSs with Docker

- Keep as transparent for the user as posible
  - Same as running a local compss application (runcompss command)
- Deploy applications as a set of docker container



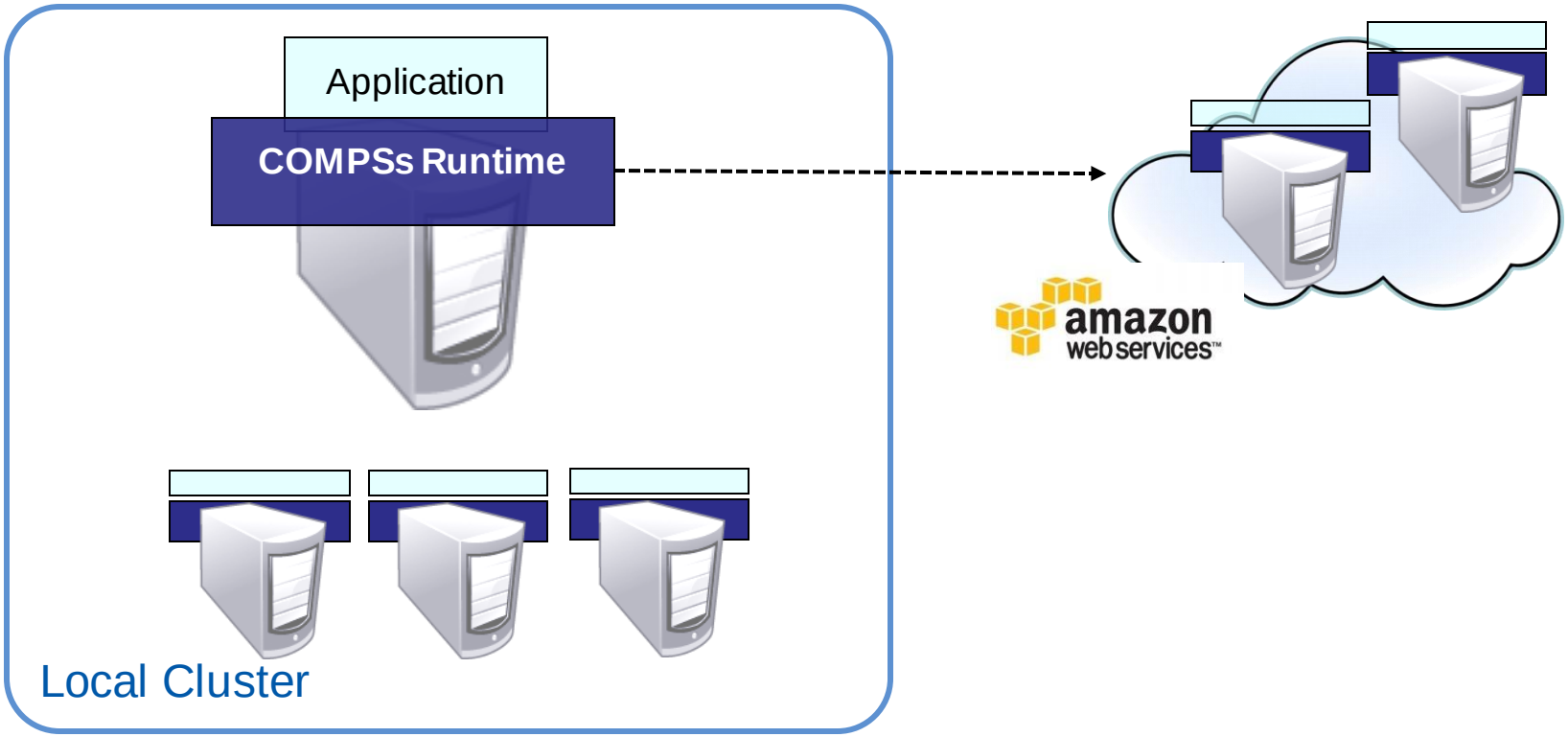
# COMPSs with Docker

- The COMPSs runtime register itself as a Mesos Framework and negotiates the use of resources with the Mesos Master.
- The number and type of nodes requested depends on the actual load.
- Both the COMPSs Master and the workers are executed in Docker containers, managed by Mesos, thus allowing a completely transparent deployment of the applications.



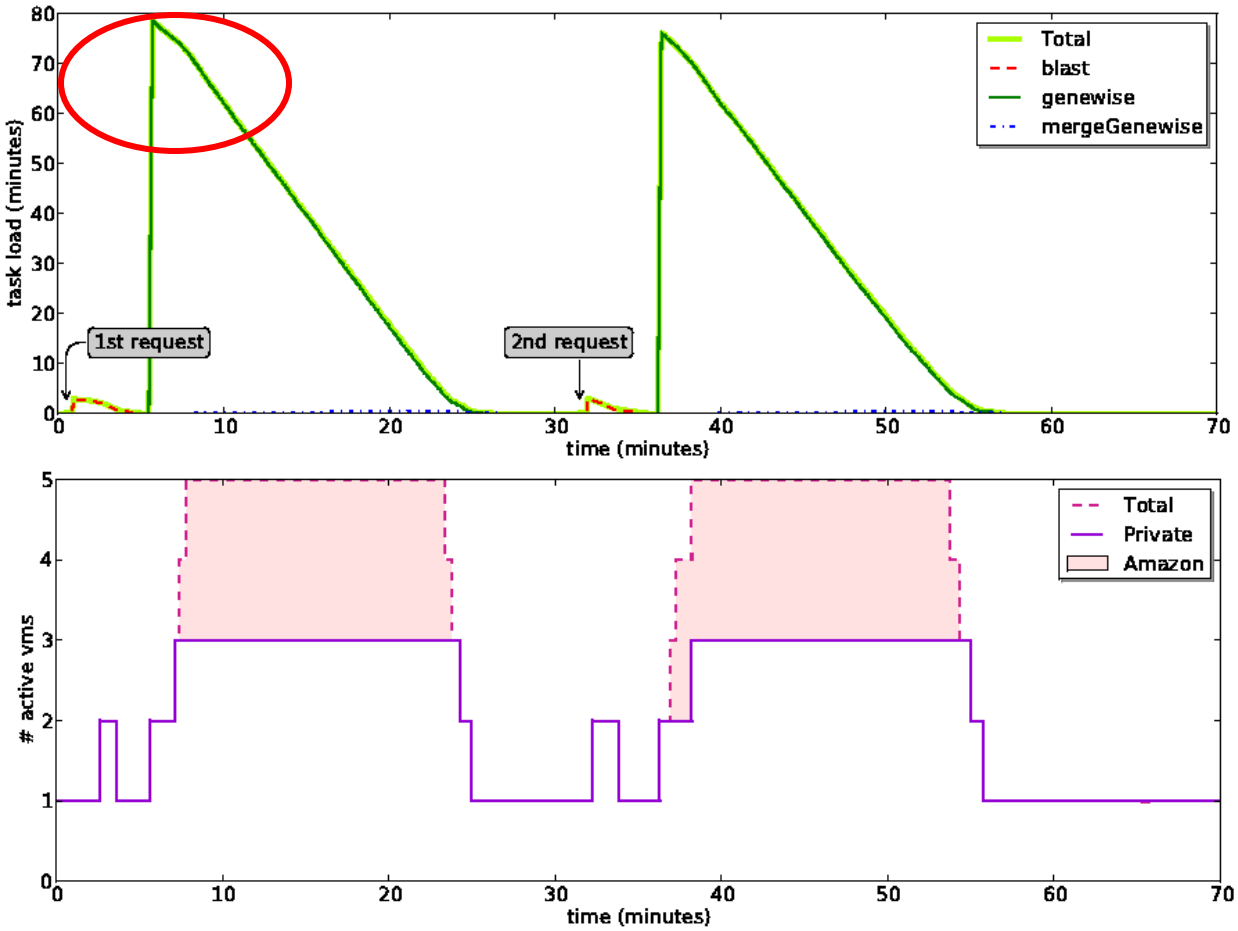
# Cloud Bursting

- Execution of COMPSs applications in Clouds
  - Select de connector to interact Cloud providers connectors, SSH

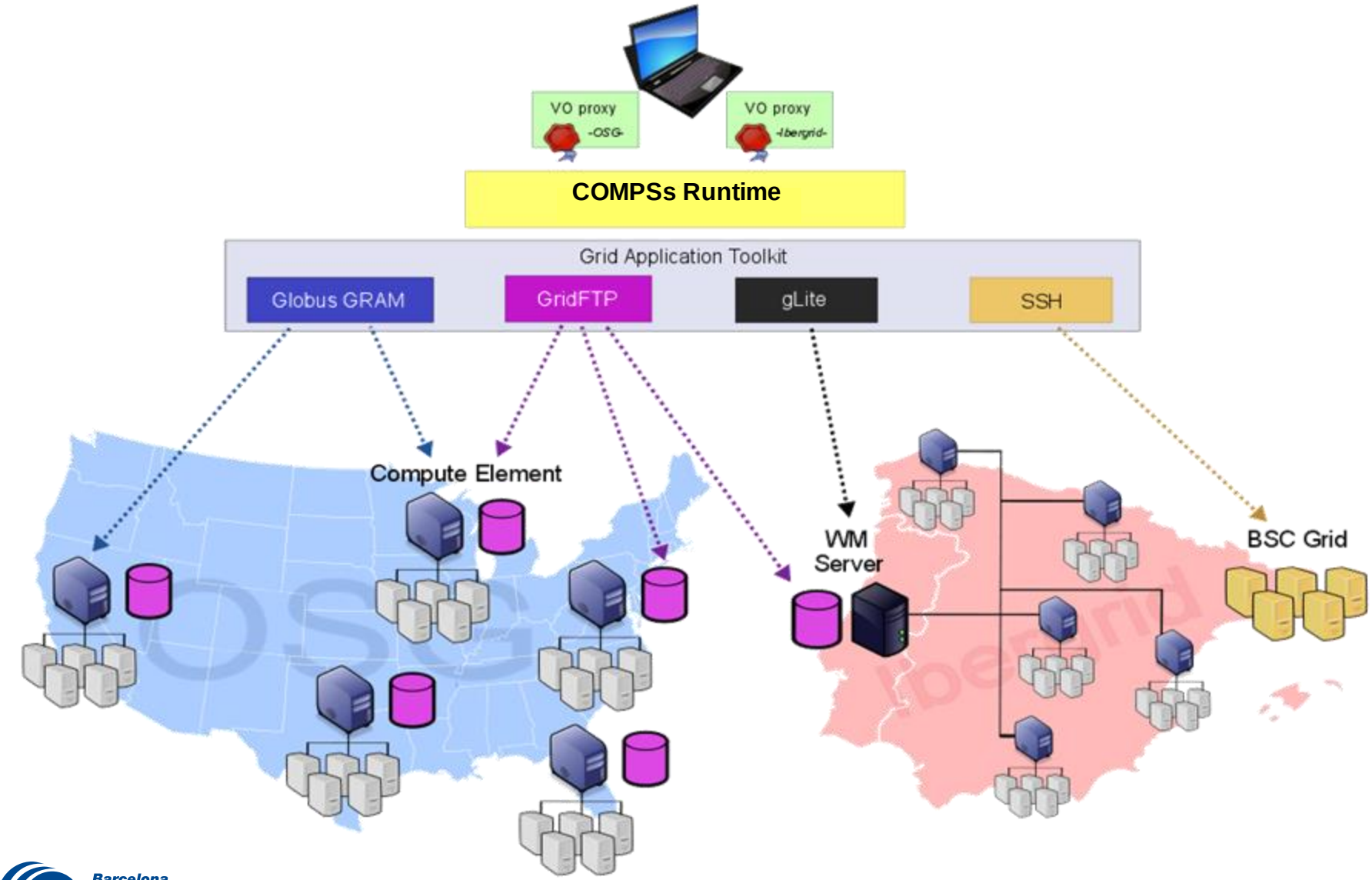


# Cloud Bursting

- Increase/decrease number of VMs depending on task load
- Bursting to Amazon EC2 to face peak load

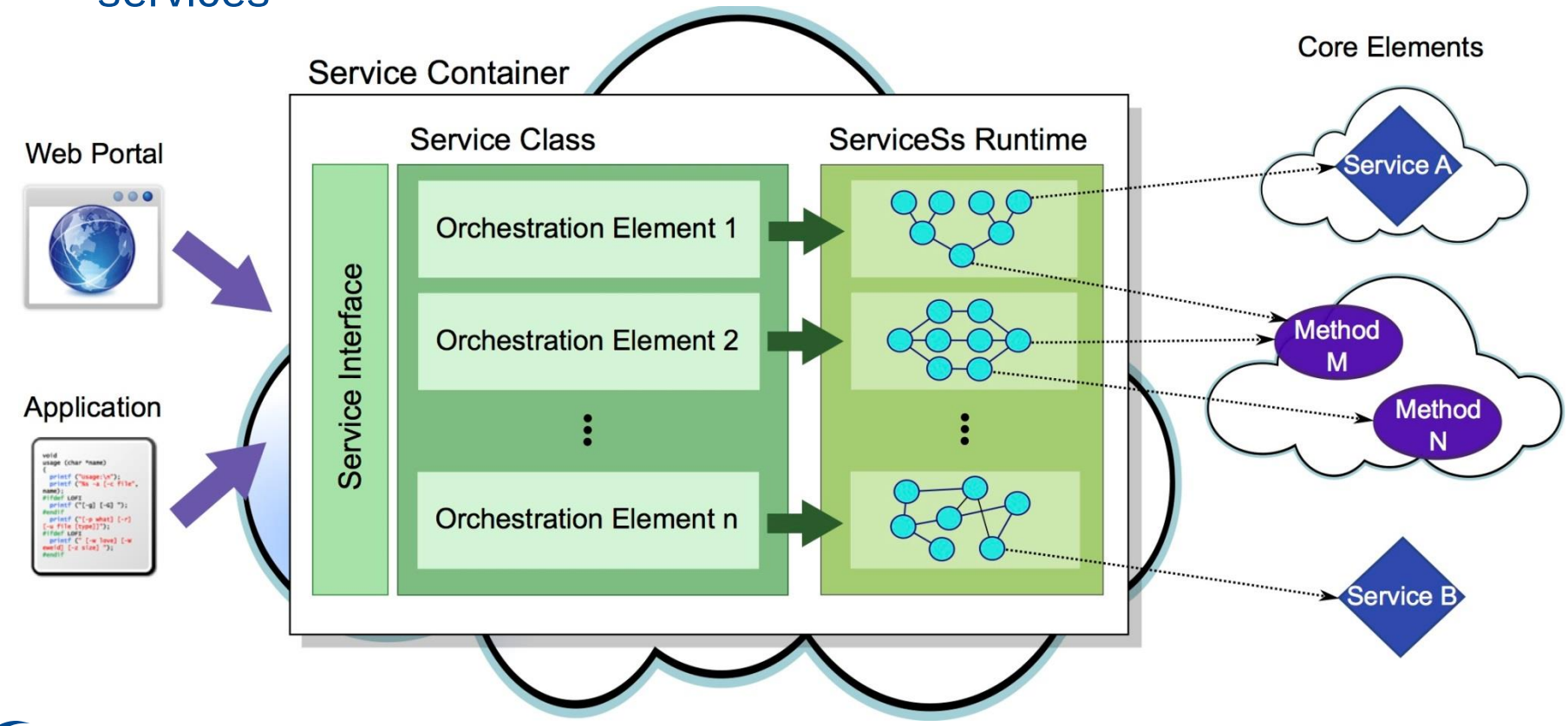


# COMPSs in multiple Grids



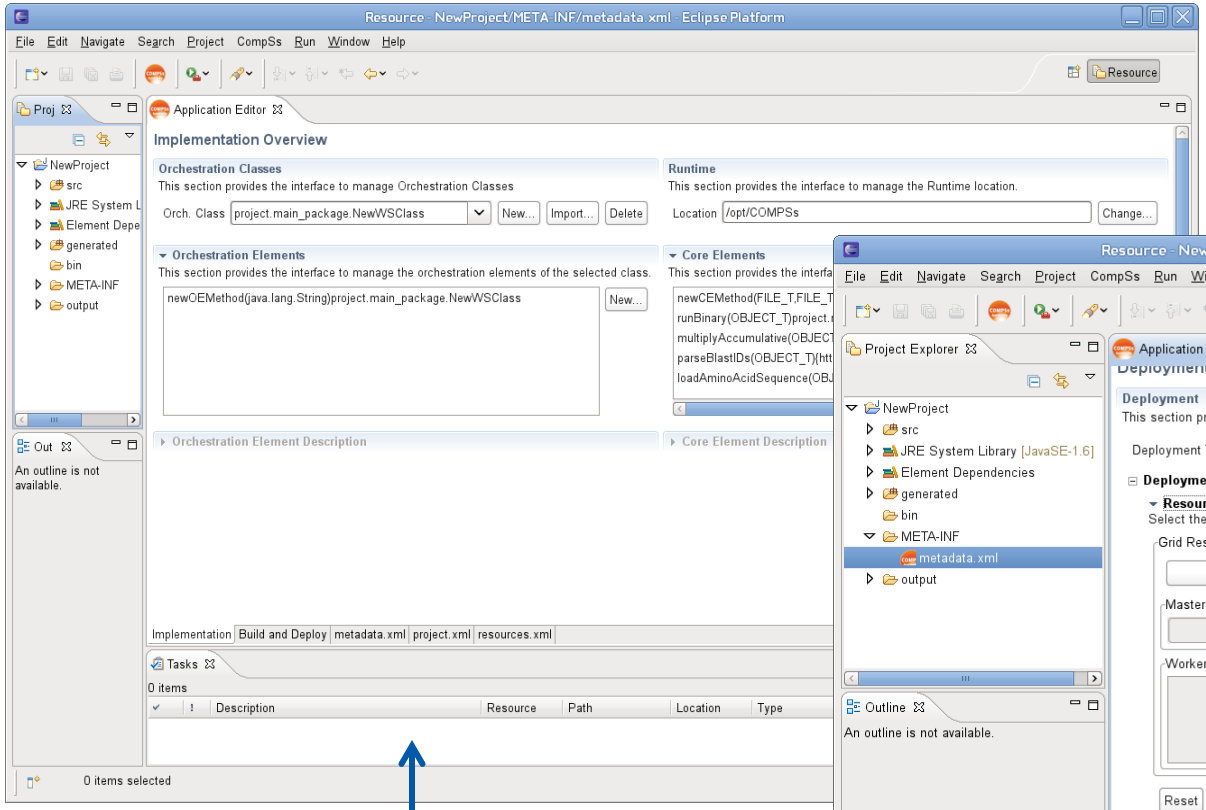
# Web service implementations with COMPSs

- A WS method implements a worklfow of tasks
- Different invocations generate different tasks
- Runtime manages the execution of the different calls in the available services



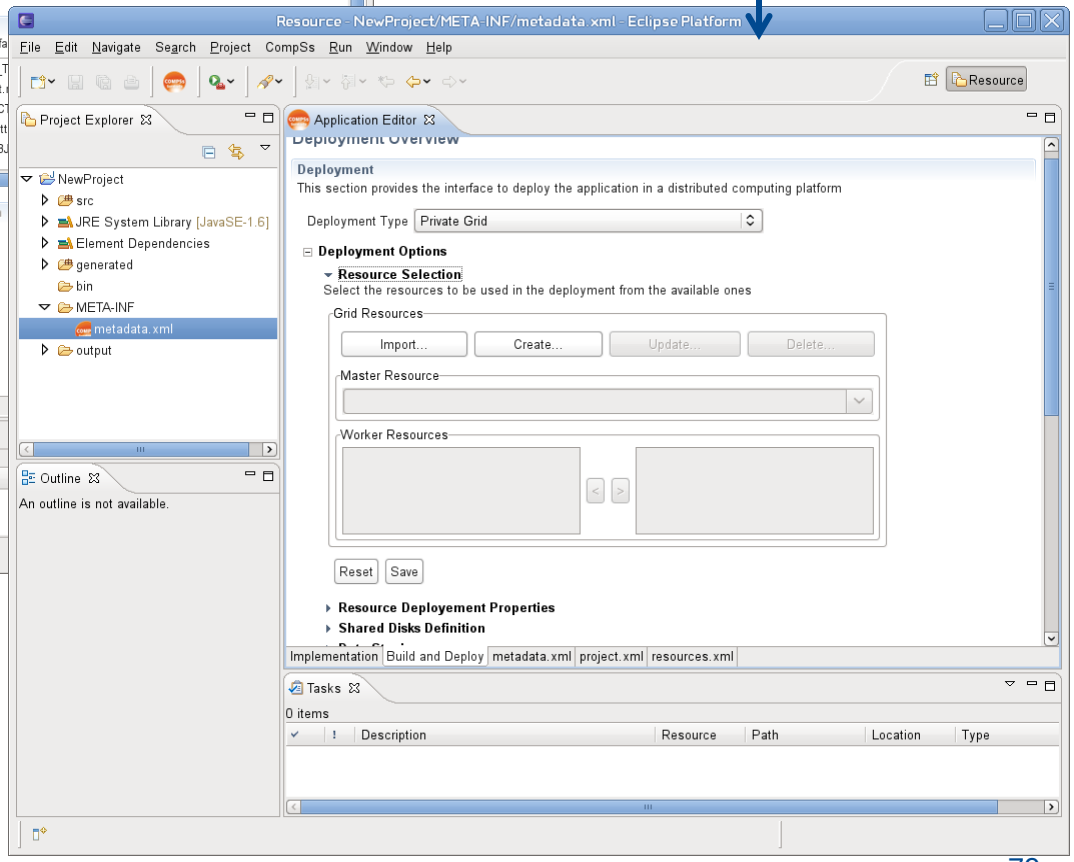
# IDE for COMPSs applications

## IDE for implementing and deploying applications




Building & Deployment:

- Generate Packages
- Define hosts & Deploy




Tasks Definition:

- Service Operations (Orchestration
- Tasks (Core Element)

# Applications using COMPSs

- Personalized medicine
  - eIMRT: planning radiotherapy treatments
- Earth Science
  - HRT: modeling global ocean-atmosphere circulation
- 3D render
  - LuxRender: renderize architectural designs
- Civil Engineering
  - EnergyPlus: modeling airflows in buildings
  - Architrave: force-effects on buildings
- Social Networks
  - SocialSensor: Tweets analysis
  - Buaala: Recommendations System

# Applications using COMPSs

- BioInformatics
  - Discrete: simulate molecular dynamics for proteins
  - Blast: alignments of protein sequences
  - Hmmer: alignment of protein sequences
  - GeneDetect: genetics algorithm
  - GUIDANCE: GWAS Analysis
  - QSAR: drug design
  - REPET
  - ABYSS
  - PyMDSetup: Molecular dynamics workflow
- Deep Learning
  - Tiramisu: Image patterns analytics


# Other sample codes

# Matrix multiply in Java

```
for (int i = 0; i < MSIZE; i++){
    for (int j = 0; j < MSIZE; j++){
        for (int k = 0; k < MSIZE; k++){
            MatmulImpl.multiplyAccumulative( _C[i][j], _A[i][k], _B[k][j] );
        }
    }
}
```

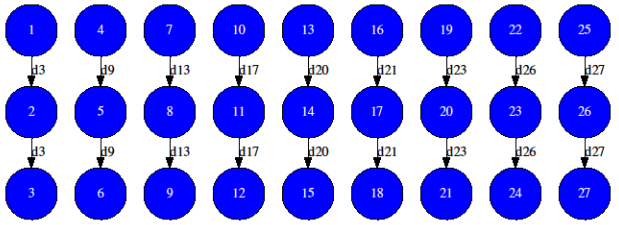
```
public static void multiplyAccumulative( String f3, String f1,
String f2 )
{
    Block a = new Block( f1 );
    Block b = new Block( f2 );
    Block c = new Block( f3 );
    c.multiplyAccum( a, b );
    try
        …
}

public void multiplyAccum ( Block a, Block b )
{
    for( int i = 0; i < this.bRows; i++ )          // rows
        for( int j = 0; j < this.bCols; j++ )      // cols
            for ( int k = 0; k < this.bCols; k++ )// cols
                this.data[i][j] += a.data[i][k] * b.data[k][j];
}
```

# Matrix multiply in Java

```
package matmul;

import integratedtoolkit.types.annotations.Constraints;
import integratedtoolkit.types.annotations.Method;
import integratedtoolkit.types.annotations.Parameter;
import integratedtoolkit.types.annotations.Parameter.*;


public interface MatmulItf {
@Constraints(processorCoreCount = 4, memoryPhysicalSize = 1.5f)
@Method(declaringClass = "matmul.MatmulImpl")
void multiplyAccumulative(
    @Parameter(type = Type.FILE, direction = Direction.INOUT)
    String file1,

    @Parameter(type = Type.FILE, direction = Direction.IN)
    String file2,

    @Parameter(type = Type.FILE, direction = Direction.IN)
    String file3
);

}
```

# Matrix multiply with constraints in Python

```
from pycompss.api.constraint import constraint
from pycompss.api.task import task
from pycompss.api.parameter import INOUT

@constraint(ComputingUnits="8")
@task(c = INOUT)
def multiply(a, b, c):
    import numpy
    c += a*b
```

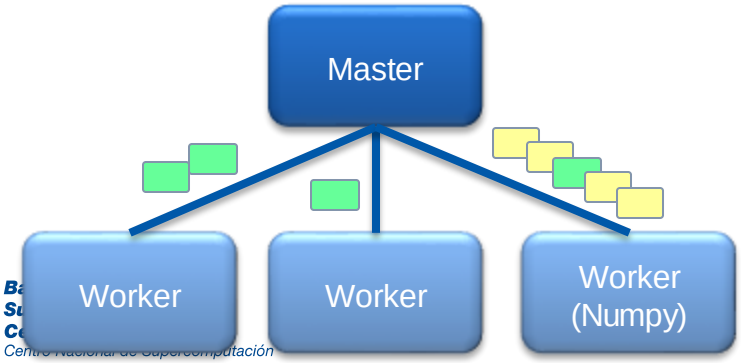



```
    args = sys.argv[1:]
    MSIZE = int(args[0])
    BSIZE = int(args[1])

    A = B = C = []
    # Initialize A, B & C as np.array(BSIZE, BSIZE)
    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])
```

# PyCOMPSs constraints

```
@task(returns=str)
def mutate(source):
    charpos = random.randint(0, len(source) - 1)
    parts = list(source)
    parts[charpos] = chr(ord(parts[charpos]) +
                         random.randint(-1,1))
    return ''.join(parts)
```

```
@constraint(AppSoftware="Numpy")
@task(returns=int)
def fitness(source, target):
    import numpy
    fitval = 0
    a = numpy.array([ord(i) for i in source])
    b = numpy.array([ord(i) for i in target])
    fitval = numpy.linalg.norm(a-b)
    return fitval
```




```
fitval = fitness(source, target)
i = 0
while True:
    i += 1
    m = mutate(source)
    fitval_m = fitness(m, target)
    fitval_m = compss_wait_on(fitval_m)
    if fitval_m < fitval:
        fitval = fitval_m
m = compss_wait_on(source)
```

# Sample code: Kmeans @ PyCOMPSs

```
from pycompss.api.api import compss_wait_on
size = int(numV / numFrag)

X = [genFragment(size, dim) for _ in range(numFrag)]
mu = init_random(dim, k)
oldmu = []
n = 0
startTime = time.time()
while not has_converged(mu, oldmu, epsilon, n, maxIterations):
    oldmu = mu
    clusters = [cluster_points_partial(X[f], mu, f * size) for f in range(numFrag)]
    partialResult = [partial_sum(X[f], clusters[f], f * size) for f in range(numFrag)]

    mu = merge_reduce(reduceCentersTask, partialResult)
    mu = compss_wait_on(mu)
    mu = [mu[c][1] / mu[c][0] for c in mu]
    n += 1
return (n, mu)
```

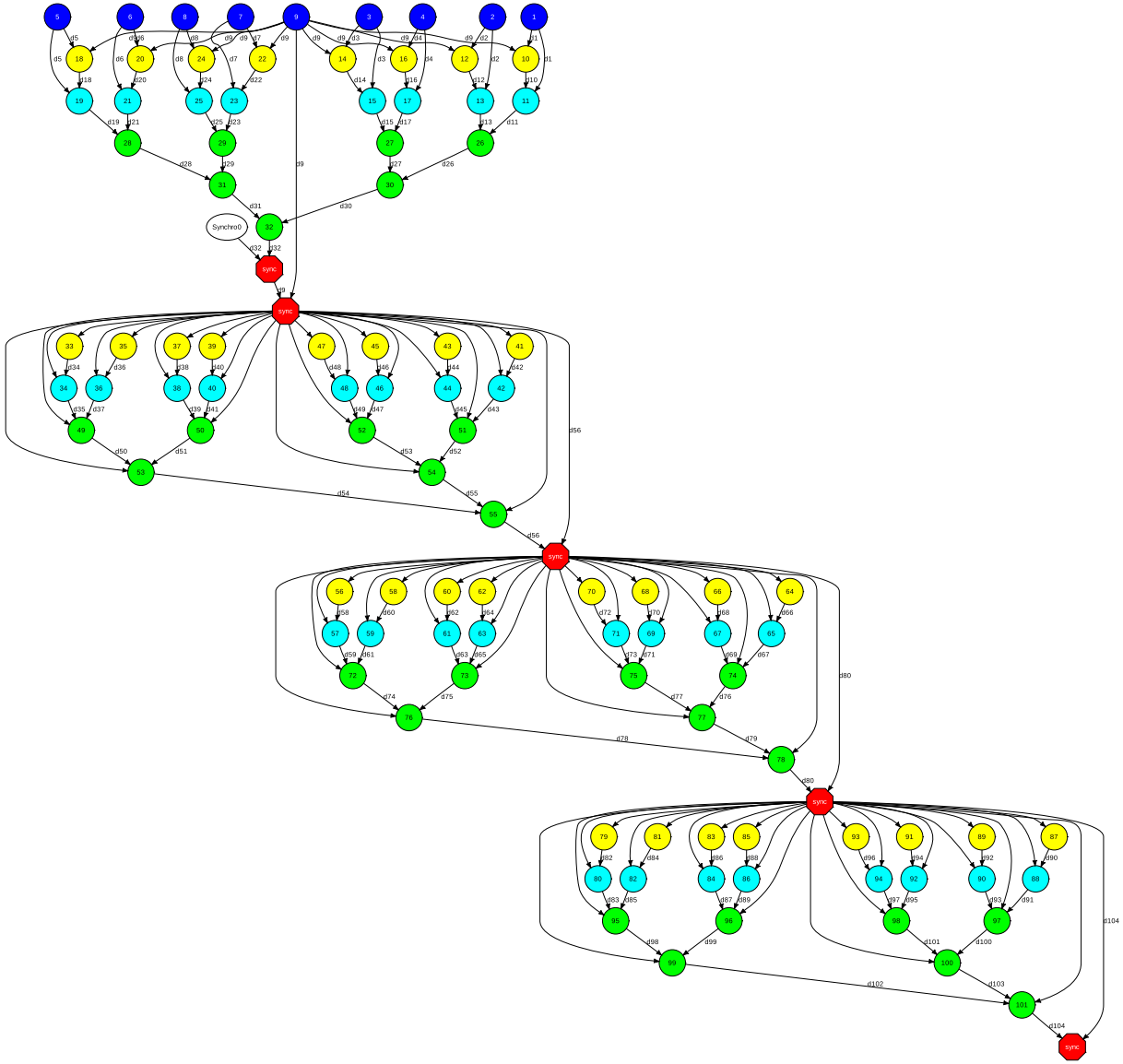
```
@task(returns=dict, priority=True)
def reduceCentersTask(a, b):
    for key in b:
        if key not in a:
            a[key] = b[key]
        else:
            a[key] = (a[key][0] + b[key][0],
                      a[key][1] + b[key][1])
    return a
```

```
@task(returns=dict)
def partial_sum(XP, clusters, ind):
    import numpy as np
    XP = np.array(XP)
    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(returns=dict)
def cluster_points_partial(XP, mu, ind):
    import numpy as np
    dic = {}
    XP = np.array(XP)
    for x in enumerate(XP):
        bestmukey = min([(i[0], np.linalg.norm(x[1] - mu[i[0]]))
                         for i in enumerate(mu)], key=lambda t: t[1])[0]
        if bestmukey not in dic:
            dic[bestmukey] = [x[0] + ind]
        else:
            dic[bestmukey].append(x[0] + ind)
    return dic
```

# Sample code: Kmeans @ PyCOMPSs

- Task graph:
  - 8 fragments
  - 4 iterations



- Computation of mutual cross-correlations between all pairs of a set of spike data
- Also computes the cross-correlations for surrogate data sets for each neuron pair

Sequential code

```
f = open('./spikes.dat', 'r')
spikes = pickle.load(f)
f.close()
#preallocate result variables
num_ccs = (num_neurons**2 - num_neurons)/2
cc_orig = zeros((num_ccs,2*maxlag+1))
cc_surrs = zeros((num_ccs,2*maxlag+1,num_surrs))
idxrange = range(num_bins-maxlag,num_bins+maxlag+1)
row = 0

#for all pairs ni,nj such that nj > ni
for ni in range(num_neurons-1):
  for nj in range(ni+1,num_neurons):
                cc_orig[row,:] = correlate(spikes[ni,:],spikes[nj,:],…
                num_spikes_i = sum(spikes[ni,:])
                num_spikes_j = sum(spikes[nj,:])
                for surrogate in range(num_surrs):
                  surr_i = zeros(num_bins)
                  surr_i[random.random_integers(0,num_bins-1,num_spikes_i)] = 1
                  surr_j = zeros(num_bins)
                  surr_j[random.random_integers(0,num_bins-1,num_spikes_j)] = 1
                  cc_surrs[row,:,surrogate] = correlate(surr_i,surr_j,"full")[idxrange]
                row = row +1
#save results
f = open('./result_cc_originals.dat','w')
pickle.dump(cc_orig,f)
f.close()
f = open('./result_cc_surrogates.dat','w')
pickle.dump(cc_surrs,f)
f.close()
```

# Neuroscience Data Processing @ Parallel Python

Main program

```
…
# tuple of all parallel python servers to connect with
ppservers = ('comp1.my-network', 'comp2.my-network' …

if len(sys.argv) > 1:
    ncpus = int(sys.argv[1])
    #creates jobserver with ncpus workers
    job_server = pp.Server(ncpus, …
else:
    #creates jobserver with workers automatically detected
    job_server = pp.Server(ppservers=ppservers,  …

#wait for servers to come up
time.sleep(5)

#calculate number of nodes in total
nlocalworkers = job_server.get_ncpus()
activenodes = job_server.get_active_nodes()
workerids = activenodes.keys()
nworkers=sum( [activenodes[workerids[i]] for i in
range(len(workerids))] ) + nlocalworkers
num_ccs = (num_neurons**2 - num_neurons)/2

#calculate number of pairs each worker should process
step = ceil(float(num_ccs)/nworkers)
start_idx = 0
end_idx = 0
starts = zeros((nworkers+1,))
…
```

Explicit resources declaration

Function definition

```
def cc_surrogate_range(start_idx, end_idx, seed, num_neurons,
num_surrs, num_bins, maxlag):
  …
```

Main program (cont)

```
for worker in range(nworkers):
    start_idx = end_idx
    end_idx = int(min((worker+1)*step,num_ccs))
    …
    depfuncs = ()
    depmodules = "numpy","pickle",
    jobs.append(job_server.submit(cc_surrogate_range,…
    …
cc_original = zeros((num_ccs,2*maxlag+1))
cc_surrs = zeros((num_ccs,2*maxlag+1,2))
for worker in arange(nworkers):
    start = starts[worker]
    end = starts[worker + 1]
    result = jobs[worker]()
    cc_original[start:end,:] = result[0]
    cc_surrs[start:end,:,:] = result[1]

f = open('./result_cc_originals.dat','w')
pickle.dump(cc_original,f)
f.close()
f = open('./result_cc_surrogates_conf.dat','w')
pickle.dump(cc_surrs,f)
f.close()
```

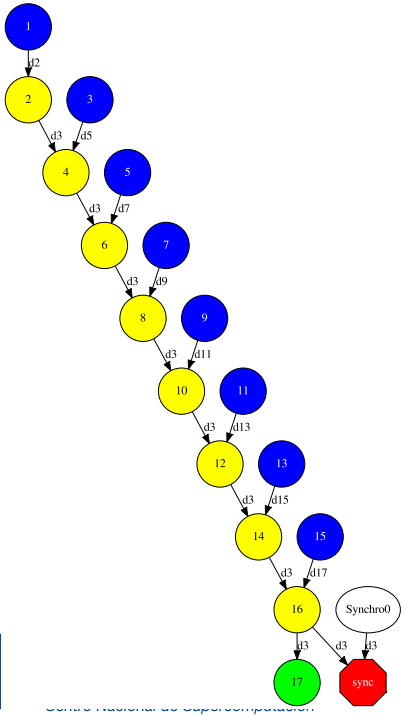
Explicit fork join

Data back

# Neuroscience Data Processing @ PyCOMPSs

Main program

```
Import sys
from pycompss.api.api  import compss_wait_on

num_frags = int(sys.argv[1])

#calculate number of pairs per fragment
num_ccs = (num_neurons**2 - num_neurons)/2
step = ceil(float(num_ccs)/num_frags)
start_idx = 0
end_idx = 0

seed = 2398645
delta = 1782324
```




Tasks definition

```
@task(cc_original = INOUT, cc_surrs = INOUT, priority = True)
def gather(result, cc_original, cc_surrs, start, end):
    cc_original[start:end,:] = result[0]
    cc_surrs[start:end,:,:] = result[1]

@task(returns = list)
def cc_surrogate_range(start_idx, end_idx, seed, num_neurons,
num_surrs, num_bins, maxlag):
    …
```

Main program (cont)

```
cc_original = zeros((num_ccs,2*maxlag+1))
cc_surrs = zeros((num_ccs,2*maxlag+1,2))
for frag in range(num_frags):
    start_idx = end_idx
    end_idx = int(min((frag+1)*step,num_ccs))
    result = cc_surrogate_range(start_idx, end_idx, seed, …
    gather(result, cc_original, cc_surrs, start_idx, end_idx)
    seed = seed + delta

f = open('./result_cc_originals.dat','w')
cc_original = compss_wait_on(cc_original)
pickle.dump(cc_original,f)
f.close()

f = open('./result_cc_surrogates_conf.dat','w')
cc_surrs = compss_wait_on(cc_surrs)
pickle.dump(cc_surrs,f)
f.close()
```


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación
HANDS-ON

# Hands-On: Overview

- COMPSs Virtual Machine Set-up
- Java Hands-on
  - Compilation & Execution
  - Configuration
  - Monitoring, debugging, graph generation
- Python Hands-on
  - Jupyter-notebook
  - Annotate tasks in Python
  - Execution in MareNostrum
  - Overview of tracing and trace analysis

# COMPSs development VM Installation

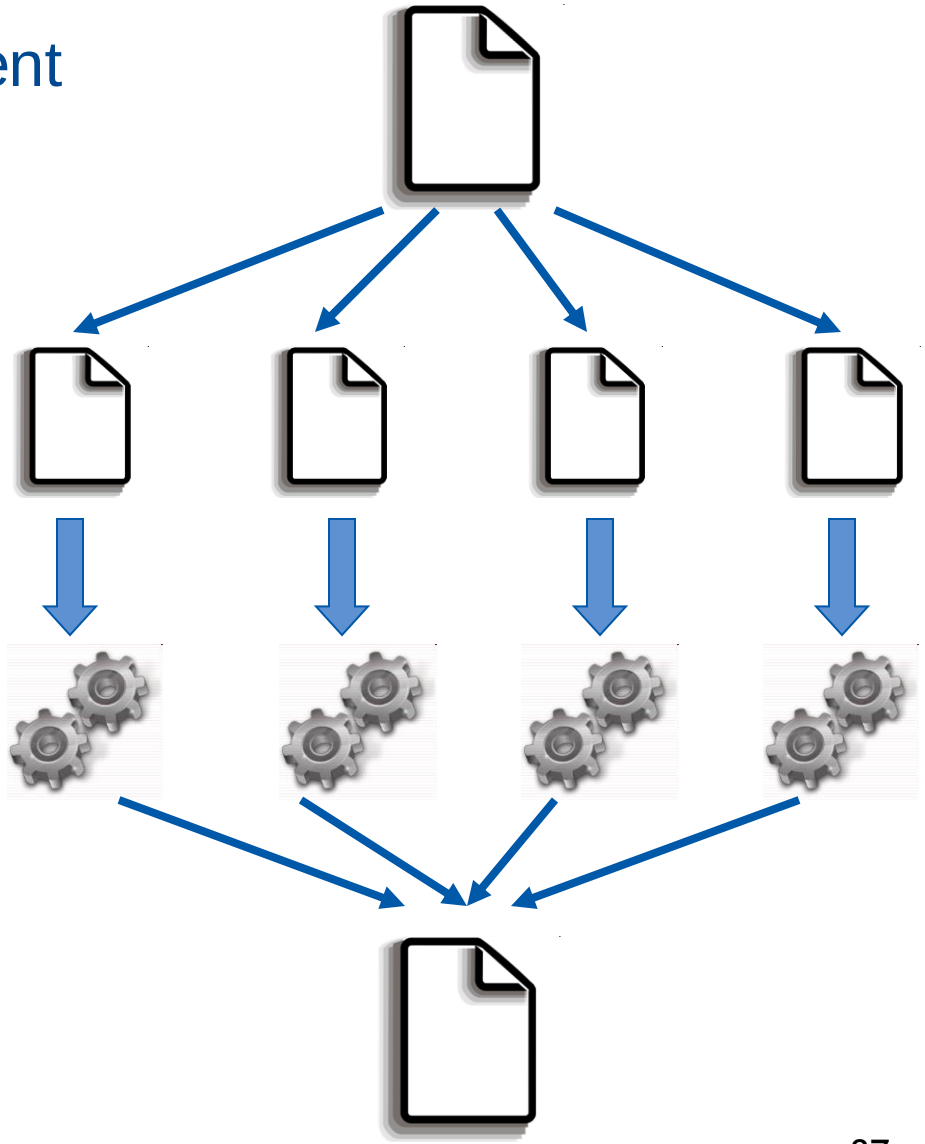
- COMPSs Virtual Appliance
  - Available from website:

    http://compss.bsc.es/releases/vms/COMPSs-2.0-VM-tutorial.ova
- VirtualBox: Import Virtual Appliance…


# Java Hands-on

# Word Count

- Counting words of a document
- Parallelization
  - Split documents in blocks
  - Count words of Blocks
  - Merge results



# Java Hands On: Exercise

- Complete the Word Count parallelization with COMPSs
  - Level 0: No Java background
    - Look the implementation (wordcount project)
  - Level 1: Basic Java background
    - Define methods in the interface (wordcount_sequential)
  - Level 2: Java background
    - Define methods in the interface and complete the part of the main code with helper methods (wordcount_blanks)

# Java Hands On: Exercise Solution

## Main Code

```
private static void computeWordCount() {
        HashMap<String, Integer> result  = new HashMap<String, Integer>();
        int start = 0;
        for (int i = 0; i < NUM_BLOCKS; ++i) {
                HashMap<String, Integer> partialResult = wordCountBlock(DATA_FILE, start, BLOCK_SIZE);
                start = start + BLOCK_SIZE;
                result = mergeResults(result, partialResult);
        }
        System.out.println("[LOG] Counted Words is : " + result.keySet().size());
}
```

## Interface

```
public interface WordcountItf {
        @Method(declaringClass = "wordcount.uniqueFile.Wordcount")
        public HashMap<String, Integer> mergeResults(
                @Parameter HashMap<String, Integer> m1,
                @Parameter HashMap<String, Integer> m2
        );

        @Method(declaringClass = "wordcount.uniqueFile.Wordcount")
        HashMap<String, Integer> wordCountBlock(
                @Parameter(type = Type.FILE, direction = Direction.IN) String filePath,
                @Parameter int start,
                @Parameter int bsize
        );
}
```

# Java Hands-on: Compilation and Simple Execution

- Compilation (Eclipse IDE)
  - Package Explorer -> Project (wordcount) -> Export… (Solution)
- Use runcompss command to run the application
  - runcompss [options] < FQDN app. classname> <application args>

- ***Exercise:*** Simple wordcount execution
  - Usage:

    wordcount.uniqueFile.Wordcount <data_file> <block_size>


```
$compss@bsc:~/> cd ~/workspace_java/wordcount/jar
$compss@bsc:~/workspace_java/wordcount/jar/> runcompss wordcount.uniqueFile.Wordcount
    /home/compss/workspace_java/wordcount/data/file_short.txt 650
```

# Java Hands-on: Result

```
$compss@bsc:~/workspace_java/wordcount/jar/> runcompss wordcount.uniqueFile.Wordcount
    /workspace_java/wordcount/data/file_short.txt 500

Using default location for project file:
    /opt/COMPSs/Runtime/scripts/user/../../configuration/xml/projects/project.xml
Using default location for resources file:
    /opt/COMPSs/Runtime/scripts/user/../../configuration/xml/resources/resources.xml

----------------- Executing wordcount.uniqueFile.Wordcount -------------------------

WARNING: IT Properties file is null. Setting default values
[   API]  -  Deploying COMPSs Runtime v2.0 (build xxxx)
[   API]  -  Starting COMPSs Runtime v2.0 (build xxxx)
DATA_FILE parameter value = /home/compss/workspace_java/wordcount/data/file_short.txt
BLOCK_SIZE parameter value = 650
[LOG] Computing word count result
[LOG] Counted Words is : 250
[   API]  -  No more tasks for app 1
[   API]  -  Getting Result Files 1
[   API]  -  Execution Finished

------------------------------------------------------------
```

# Java Hands-on: Configuration

- Project.xml: /opt/COMPSs/Runtime/configuration/xml/projects/project.xml

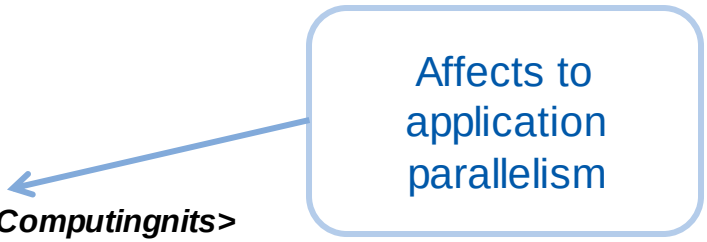
```
<?xml version="1.0" encoding="UTF-8"?>
<Project>
      <MasterNode>
      <ComputeNode  Name="localhost">
            <InstallDir>/opt/COMPSs/</InstallDir>
            <WorkingDir>/tmp/COMPSsWorker</WorkingDir>
      </ComputeNode>
</Project>
```

  - Other optional parameters
    - User, AppDir, LibraryPath

# Java Hands-On: Configuration

- Resources.xml: /opt/COMPSs/Runtime/configuration/xml/resources/default_resources.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ResourceList>
    <!--Description for any physical node-->
    <ComputeNode Name="localhost">
        <Processor Name="Main">
            <ComputingUnits>4</Computingnits>
        </Processor>
        <Memory>
            <size>8</size>
        </Memory>
        <Storage>
            <size>50</size>
        </Storage>
        <Adaptors>
            <Adaptor Name="integratedtoolkit.nio.master.NIOAdaptor">
                …
            <Adaptor Name="integratedtoolkit.gat.master.GATAdaptor">
                …
        </Adaptors>
    </ComputeNode>
<ResourceList>
```



# Java Hands-On: Monitoring

- The runtime of COMPSs provides real-time monitoring
  - http://localhost:8080/compss-monitor/
  - If not started run as root:
    - /etc/init.d/compss-monitor start
- The user can log-in and follow the progress of the executions
  - Running tasks, resources usage, execution time per task, real-time execution graph, etc.
- Activate monitoring with a runcompss flag
  - Setting a monitoring interval
    - runcompss **--monitoring=<int>**
  - With a default monitoring interval
    - runcompss **–m** (or) runcompss **--monitoring**
- ***Exercise:*** run wordcount enabling monitoring


```
$compss@bsc:~/> cd ~/workspace_java/wordcount/jar
$compss@bsc:~/workspace_java/wordcount/jar/> runcompss –m wordcount.uniqueFile.Wordcount
    /home/compss/workspace_java/wordcount/data/file_long.txt 250000
```

# Java Hands-on: Debugging

- Different log levels activated as runcompss options
  - runcompss **--log_level**=<level>
    (**off**: for performance | **info**: basic logging | **debug**: detect errors)
  - runcompss **–debug** or runcompss **-d**
- The output/errors of the main code of the application are shown in the console
- Other logging files are stored in:
  - $HOME/.COMPSs/<APP_NAME>_XX
- Inside this folder, the user can check the following:
  - The output/error of a task # N : ***/jobs/jobN.[out|err]***
  - Messages from the COMPSs : ***runtime.log***
  - Task to resources allocation: ***resources.log***
- ***Exercise:*** run wordcount with debugging


```
$compss@bsc:~/> cd ~/workspace_java/wordcount/jar
$compss@bsc:~/workspace_java/wordcount/jar/> runcompss –d wordcount.uniqueFile.Wordcount
    /home/compss/workspace_java/wordcount/data/file_short.txt 650
```

# Java Hands-on: Graph generation

- To generate the graph of an application, it must be run with the monitor or graph flags activated
  - runcompss **-m** (or) runcompss **–graph** (or) runcompss **-g**
- The graph will be stored in:
  - $HOME/.COMPSs/<APP_NAME>_XX/monitor/complete_graph.dot
- To convert the graph to a PDF format use gengraph command
  - Usage: gengraph <dot_file>
- **Exercise:** generate the graph for the wordcount application


```
$compss@bsc:~/> cd ~/workspace_java/wordcount/jar
$compss@bsc:~/workspace_java/wordcount/jar/> runcompss –g wordcount.uniqueFile.Wordcount
     /home/compss/workspace_java/wordcount/data/file_short.txt 650

… application execution …

$compss@bsc:~/workspace_java/wordcount/jar/> cd ~/.COMPSs/wordcount.uniqueFile.Wordcount_04/monitor
$~/.COMPSs/wordcount.uniqueFile.Wordcount_04/monitor> gengraph complete_graph.dot
Output file: complete_graph.pdf
$~/.COMPSs/wordcount.uniqueFile.Wordcount_04/monitor> evince complete_graph.pdf
```

www.bsc.es


Barcelona Supercomputing Center

*Centro Nacional de Supercomputación*

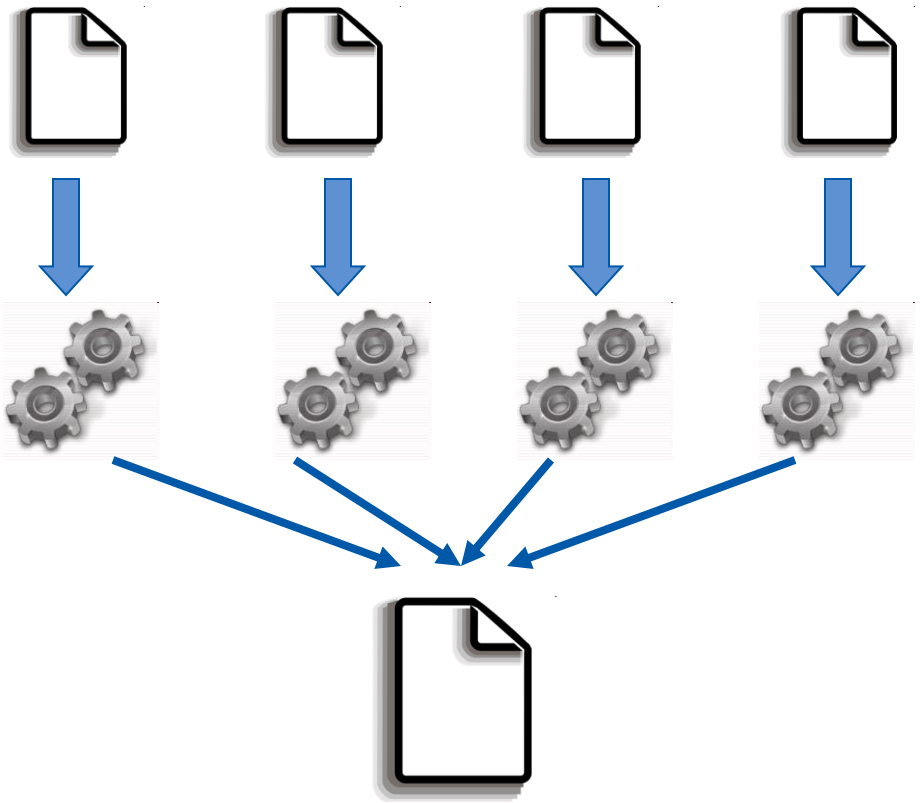
# Python Hands-on

# PyCOMPSs Hands On: Exercise

- Complete the WordCount parallelization with PyCOMPSs:
  - **In Jupyter-Notebook**
    - **Task definition with Python decorators**
    - **Local execution**
  - Execution in MareNostrum III
  - Overview of tracing:
    - Trace analysis

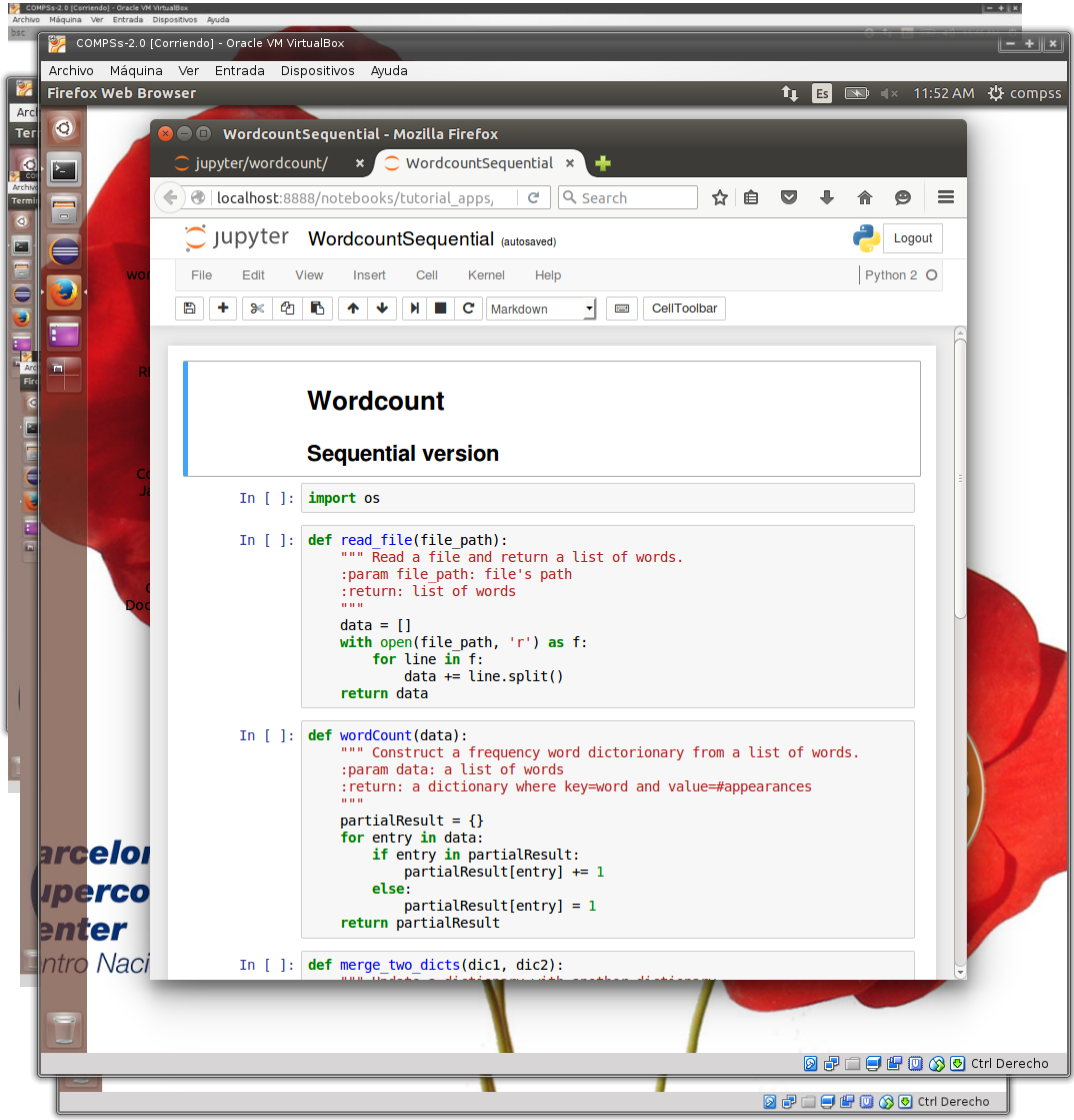
# Word Count

- Counting words of a set of documents
- Parallelization
  - Phase 1: Count words of a set of documents
  - Phase 2 : Merge results



# PyCOMPSs and Jupyter-Notebook

- Start the Virtual Machine
  - User: compss
  - Password: compss2017
- Open a console
- Start jupyter-notebook
  - $ jupyter-notebook
- Look for the sequential wordcount project:
  - Level 0: No Python Background
    - Wordcount.ipynb
  - Level 1: Python Background
    - WordcountSequential.ipynb



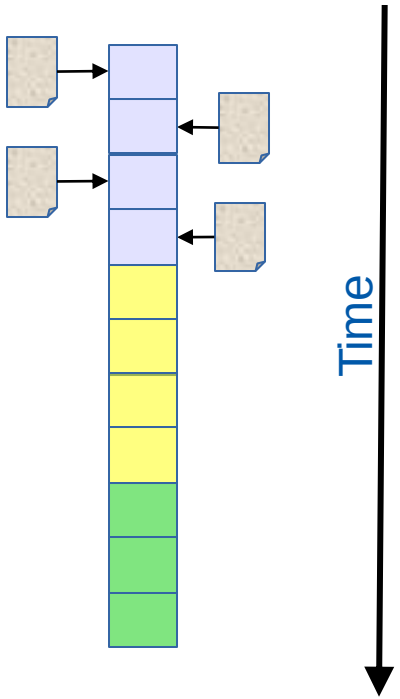
# WordCount @ Sequential (WordcountSequential.ipynb)

```
def read_file(file_path):
    data = []
    with open(file_path, 'r') as f:
        for line in f:
            data += line.split()
    return data
```

```
def wordCount(data):
    partialResult = {}
    for entry in data:
        if entry in partialResult:
            partialResult[entry] += 1
        else:
            partialResult[entry] = 1
    return partialResult
```

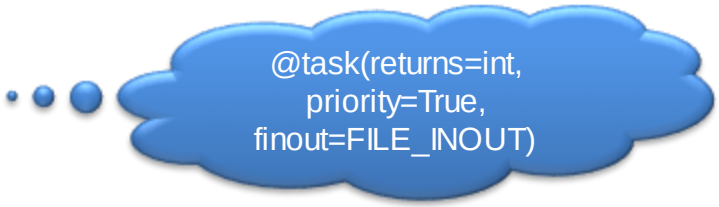
```
def merge_two_dicts(dic1, dic2):
    for k in dic2:
        if k in dic1:
            dic1[k] += dic2[k]
        else:
            dic1[k] = dic2[k]
    return dic1
```

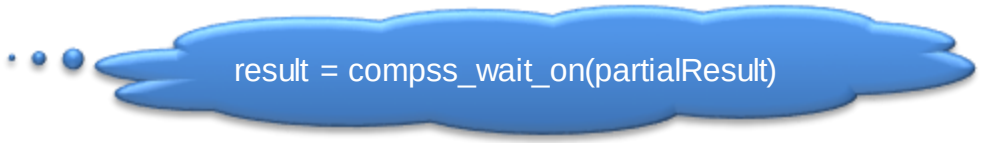
```
if __name__ == "__main__":
    pathDataset = sys.argv[1]
    # Construct a list with the file's paths from the dataset
    paths = []
    for fileName in os.listdir(pathDataset):
        paths.append(os.path.join(pathDataset, fileName))

    # Read file's content
    data = map(read_file, paths)

    # From all file's data execute a wordcount on it
    partialResult = map(wordCount, data)

    # Accumulate the partial results to get the final result.
    result = reduce(merge_two_dicts, partialResult)
```



# PyCOMPSs cheatsheet

- Important Modules
  - Interactive: **pycompss.interactive**
    - start(debug=<True|*False*>, monitor=<int>, graph=<True|*False*>, trace=<True|*False*>, taskCount=<int>)
    - stop(sync=<True|False>)
  - Constraint decorator: **pycompss.api.constraint**
  - Task decorator: **pycompss.api.task**
    - Keywords:
      - returns=<return_type>
      - priority=<True|False>
      - function_var_name=<parameter>
  - Task parameters: **pycompss.api.parameter**
    - *IN*, OUT, INOUT
    - FILE, FILE_IN, FILE_OUT, FILE_INOUT
  - API: **pycompss.api.api**
    - compss_open(file_name, mode='r')
    - compss_delete(file_name)
    - waitForAllTasks()
    - compss_wait_on(object)

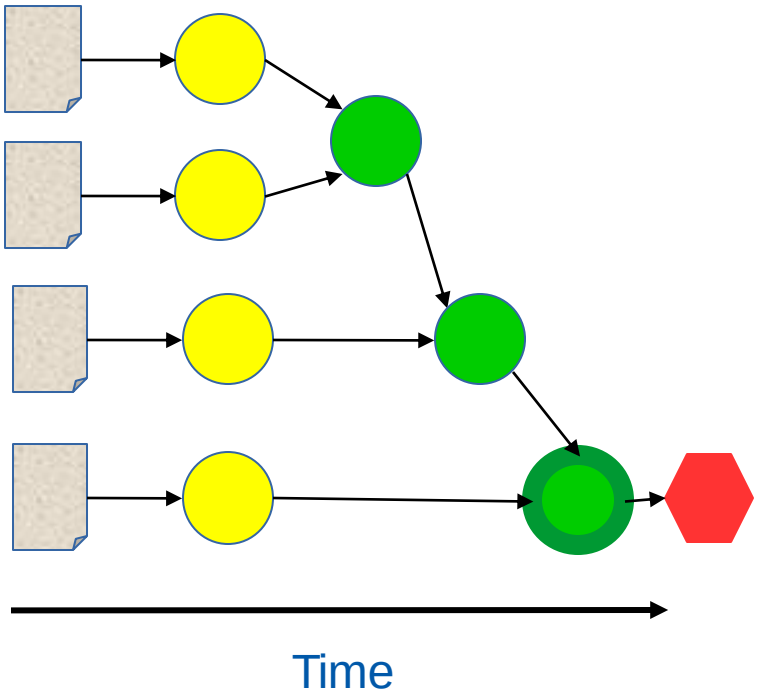
# WordCount @ PyCOMPSs (option1)

```
def read_file(file_path):
    data = []
    with open(file_path, 'r') as f:
        for line in f:
            data += line.split()
    return data
```

```
@task(returns=dict)
def wordCount(data):
    partialResult = {}
    for entry in data:
        if entry in partialResult:
            partialResult[entry] += 1
        else:
            partialResult[entry] = 1
    return partialResult
```

```
@task(returns=dict, priority=True)
def merge_two_dicts(dic1, dic2):
    for k in dic2:
        if k in dic1:
            dic1[k] += dic2[k]
        else:
            dic1[k] = dic2[k]
    return dic1
```

```
if __name__ == "__main__":
    from pycompss.api.api import compss_wait_on
    pathDataset = sys.argv[1]
    # Construct a list with the file's paths from the dataset
    paths = []
    for fileName in os.listdir(pathDataset):
        paths.append(os.path.join(pathDataset, fileName))

    # Read file's content
    data = map(read_file, paths)

    # From all file's data execute a wordcount on it
    partialResult = map(wordCount, data)

    # Accumulate the partial results to get the final result.
    result = reduce(merge_two_dicts, partialResult)

    # Wait for result
    result = compss_wait_on(result)
```



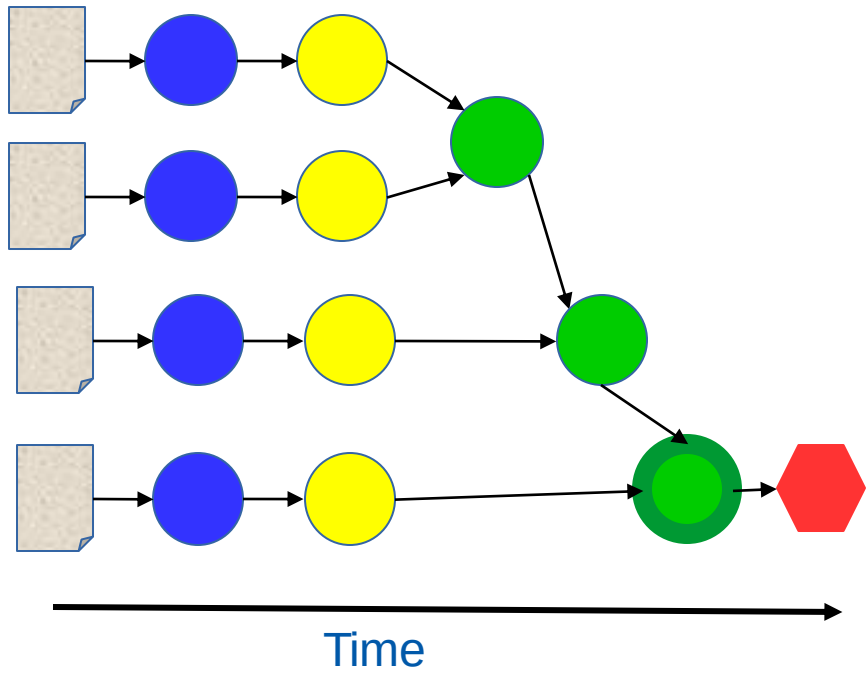
# WordCount @ PyCOMPSs (option 2)

```
@task(returns=list,
      file_path=FILE_in)
def read_file(file_path):
    data = []
    with open(file_path, 'r') as f:
        for line in f:
            data += line.split()
    return data
```

```
@task(returns=dict)
def wordCount(data):
    partialResult = {}
    for entry in data:
        if entry in partialResult:
            partialResult[entry] += 1
        else:
            partialResult[entry] = 1
    return partialResult
```

```
@task(returns=dict, priority=True)
def merge_two_dicts(dic1, dic2):
    for k in dic2:
        if k in dic1:
            dic1[k] += dic2[k]
        else:
            dic1[k] = dic2[k]
    return dic1
```

```
if __name__ == "__main__":
    from pycompss.api.api import compss_wait_on
    pathDataset = sys.argv[1]
    # Construct a list with the file's paths from the dataset
    paths = []
    for fileName in os.listdir(pathDataset):
        paths.append(os.path.join(pathDataset, fileName))

    # Read file's content
    data = map(read_file, paths)

    # From all file's data execute a wordcount on it
    partialResult = map(wordCount, data)

    # Accumulate the partial results to get the final result.
    result = reduce(merge_two_dicts, partialResult)

    # Wait for result
    result = compss_wait_on(result)
```



# PyCOMPSs Hands On: Exercise

- Complete the WordCount parallelization with PyCOMPSs:
  - In Jupyter-Notebook
    - Task definition with Python decorators
    - Local execution
  - **Execution in MareNostrum III**
  - Overview of tracing:
    - Trace analysis

# Execution in MareNostrum III

- How to connect to MN3?
  - > ssh -X nct01XXX@mn3.bsc.es
  - Password: p@tcc0mps.XXX

- Update .bashrc
  - Edit: **.bashrc**
  - Add: "**module load COMPSs/2.0**" at the end
  - Execute: **source .bashrc**

- Where is the source code?
  - **cd**
  - **cp /gpfs/projects/nct01/nct01001/source/* .**

launch_pycompss.sh
wordcount.py
wordcountPyCOMPSs.sh
wordcount2PyCOMPSs.sh

- Where is the dataset?
  - **cp -r /gpfs/projects/nct01/nct01001/dataset .**

- Available editors
  - vi
  - emacs

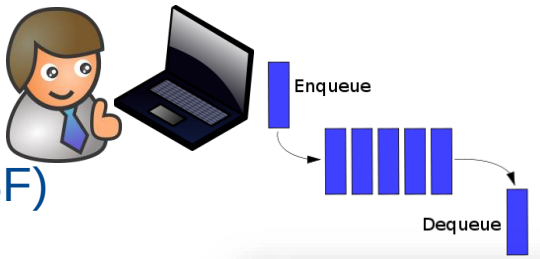
# WordCount @ Sequential

- Remember the dataset path
- How to launch with python sequentialy?
  - > **python wordcount.py /gpfs/projects/nct01/nct01001/dataset/4/**

  Results:

```
user@login3:~> python wordcount.py /path/to/dataset/
Elapsed Time (s)
0.959941864014
Words: 2571768
```

- Submit jobs to MareNostrum III
  - All jobs should be submited to the queuing system (LSF)
  - We will use a launcher script: **launch_pycompss.sh**
  - Useful commands:
    - bjobs – This command shows the status of the job.
    - bkill jobId – This command kills a job with id 'jobId'.




# Execution in MareNostrum III - HandsOn

## launch_pycompss.sh

```
#/bin/bash

enqueue_compss \
  --exec_time=10 \
  --reservation=nct74_rsv01 \
  --num_nodes=2 \
  --tasks_per_node=16 \
  --lang=python \
  --tracing=true \
  --graph=true \
  /home/nct01/nct01XXX/wordcountPyCOMPSs.py /gpfs/home/nct01/nct01XXX/dataset/64files
```

## Parameters:

- num_nodes: amount of nodes where to execute (1 master + 1 worker).
- tasks_per_node: amount of tasks that can be processed in parallel (1-16).
- Dataset path: **/gpfs/home/nct01/nct01XXX/dataset/64files**

## How to execute with PyCOMPSs?

- **./launch_pycompss.sh**

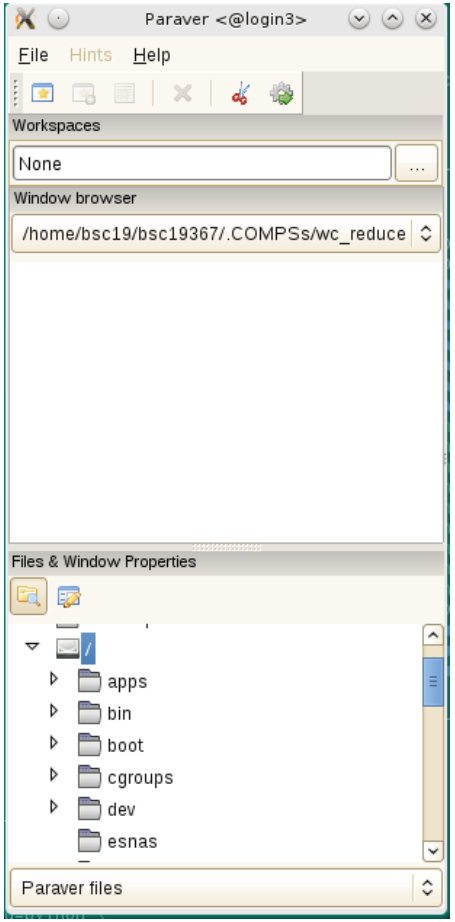
# PyCOMPSs Hands On: Exercise

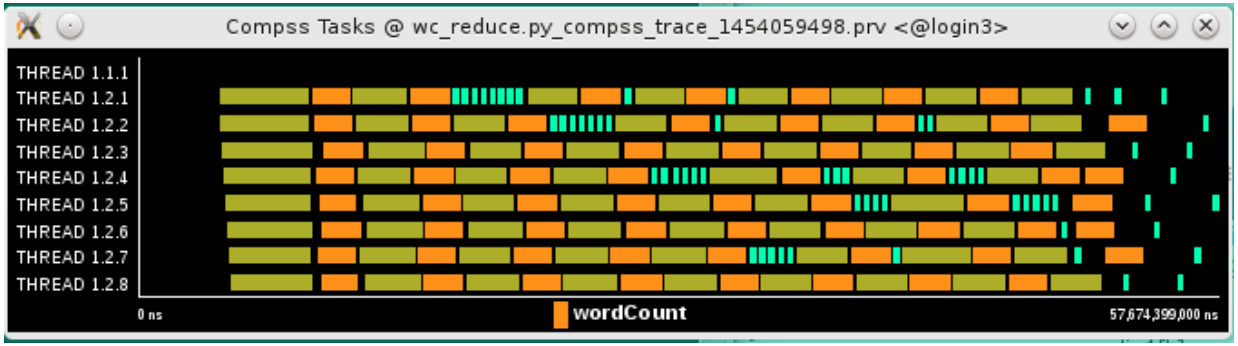
- Complete the WordCount parallelization with PyCOMPSs:
  - In Jupyter-Notebook
    - Task definition with Python decorators
    - Local execution
  - Execution in MareNostrum III
  - **Overview of tracing:**
    - **Trace analysis**

# Wordcount @ Performance Analysis

- Paraver is the BSC tool for trace visualization
  - Trace events are encoding in Paraver (.prv) format by Extrae
  - Paraver is a powerful tool for trace visualization.
  - An experimented user could obtain many different views of the trace events.

- For more information about Paraver visit:
  - http://www.bsc.es/computer-sciences/performance-tools/paraver



# Wordcount @ Performance Analysis

- COMPSs can generate post-execution traces of the distributed execution of the application
  - Useful for performance analysis and diagnosis
- How it works?
  - Task execution and file transfers are application events
  - An XML file is created at workers to keep track of these events
  - At the end of the execution all the XML files are merged to get the final trace file
  - COMPSs uses Extrae tool to dynamically instrument the application
    - In a worker:
      - Extrae keeps track of the events in an intermediate file
    - In the master:
      - Extrae merges the intermediate files to get the final trace file

# Wordcount @ Performance Analysis

```
---------- Executing wc_reduce.py ----------

Welcome to Extrae 3.3.0 (revision 3966 based on extrae/trunk)
Extrae: Generating intermediate files for Paraver traces.
Extrae: Intermediate files will be stored in /.statelite/tmpfs/gpfs/home/bsc19/bsc19000/Apps/WC/src/tutorial
Extrae: Tracing buffer can hold 100000 events
Extrae: Tracing mode is set to: Detail.
Extrae: Successfully initiated with 1 tasks and 1 threads

[ API] - Starting COMPSs Runtime v2.0 (build 20161222-1250.r2680)
...
[ API] - No more tasks for app 0
[ API] - Getting Result Files 0
[ API] - Execution Finished
...

Extrae: Application has ended. Tracing has been terminated.

merger: Output trace format is: Paraver
merger: Extrae 3.3.0 (revision 3966 based on extrae/trunk)

mpi2prv: Selected output trace format is Paraver
mpi2prv: Parsing intermediate files
mpi2prv: Generating tracefile (intermediate buffers of 745642 events)

mpi2prv: Congratulations! ./trace/wc_reduce.py_compss_trace_1453885329.prv has been generated.
```

Extrae starts before the user application execution

COMPSs runtime starts

The application finishes and the tracing process ends

COMPSs runtime ends

The merge process starts

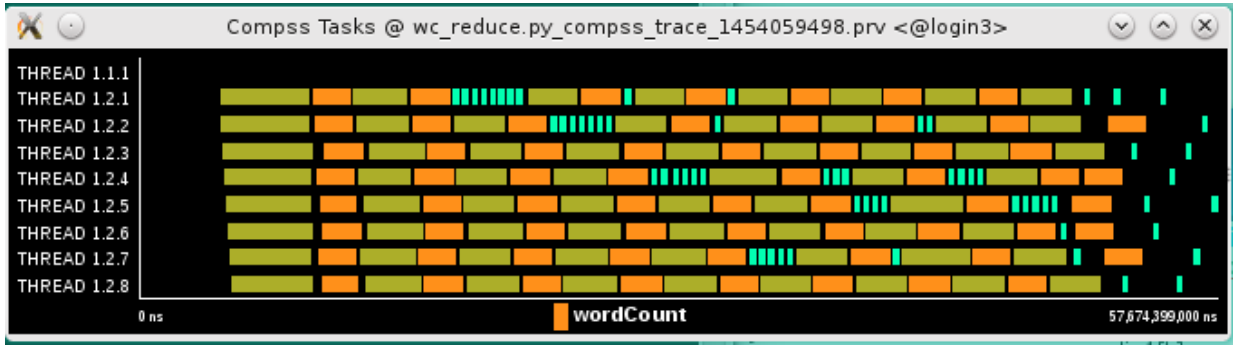
Intermediate trace files are processed

The final trace file is generated

# WordCount @ Performance Analysis

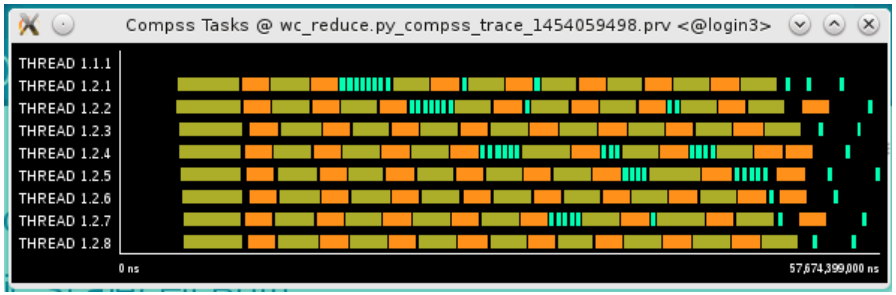
- Open Paraver
  - > module load paraver
  - > cd $HOME/.COMPSs/{JobID}
  - > wxparaver trace/*.prv

- COMPSs provides some configuration files to automatically obtain the view of the trace
  - File/Load Configuration... (/gpfs/apps/MN3/COMPSs/2.0/Dependencies/paraver/cfgs/compss_tasks.cfg)

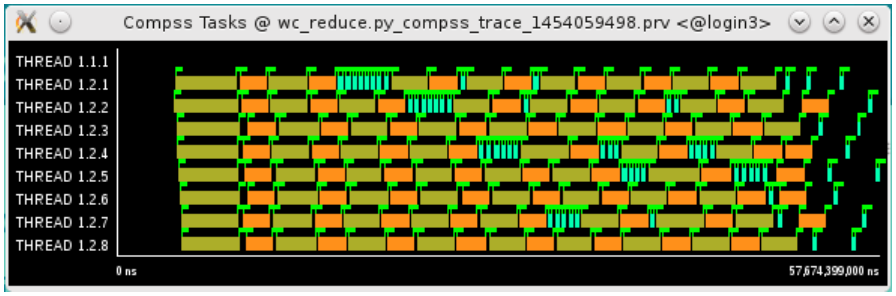


# Wordcount @ Performance Analysis

- Fit window
  - Right click on the trace window
  - Fit Semantic Scale/ Fit Both




- View Event flags
  - Right click on the trace window
  - View / Event Flags



# Wordcount @ Performance Analysis

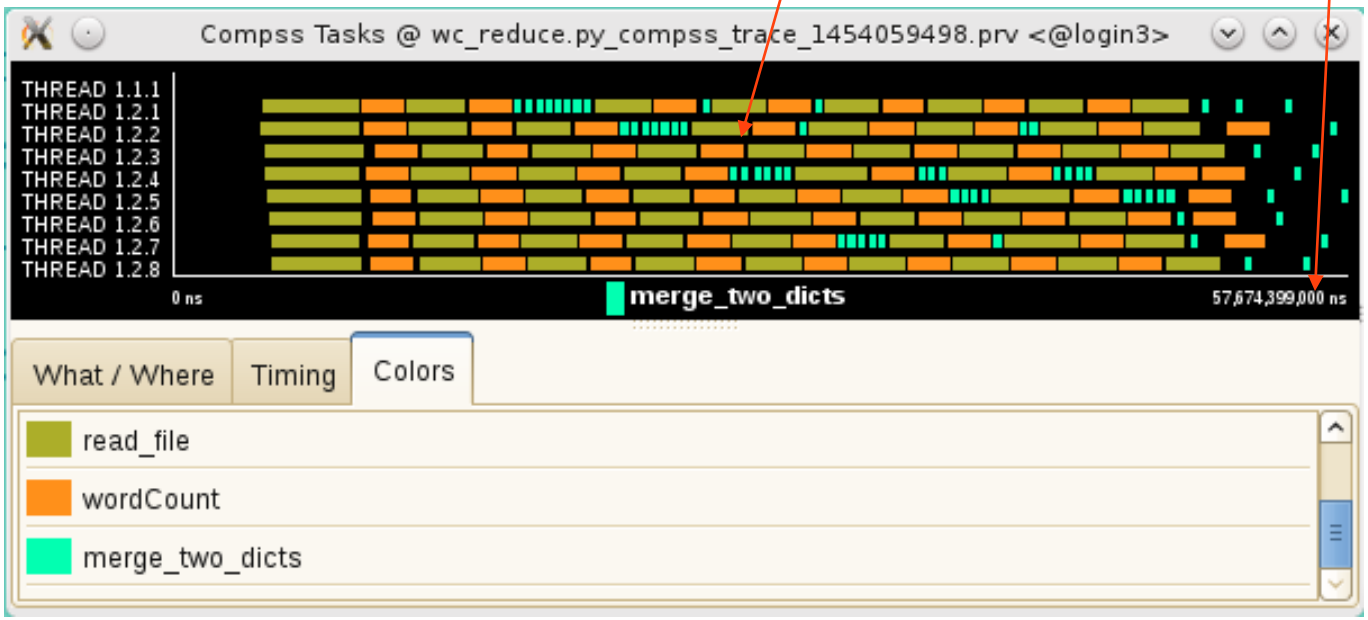
## Show info Panel

- Right click on the trace window
- Check info panel option
- Select Colors tab of the panel

Tasks

Execution time

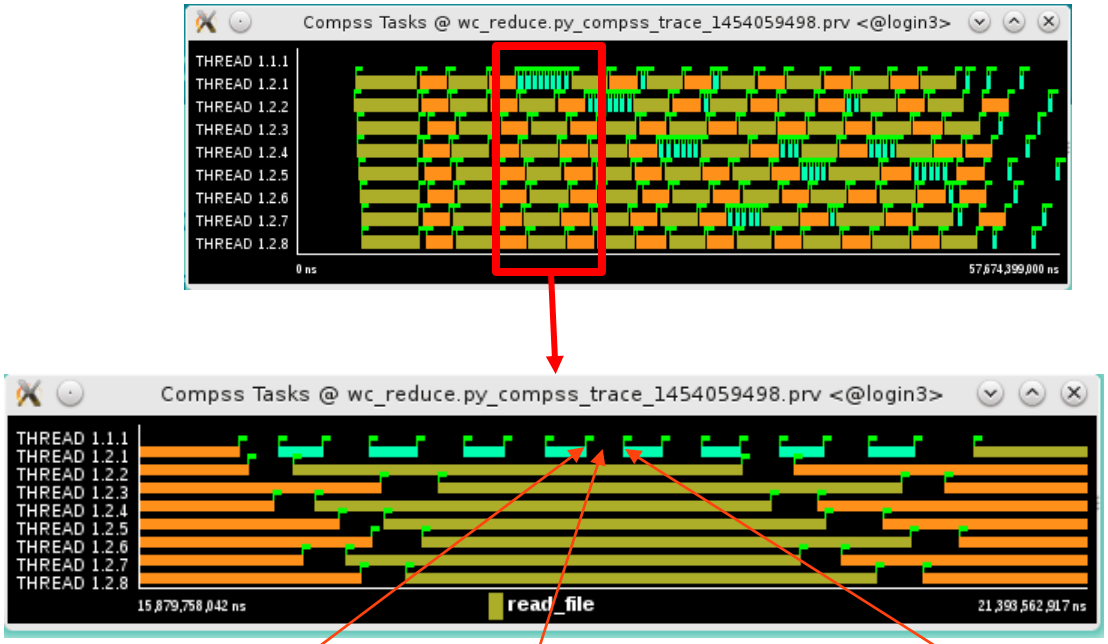
Threads




Legend with task names

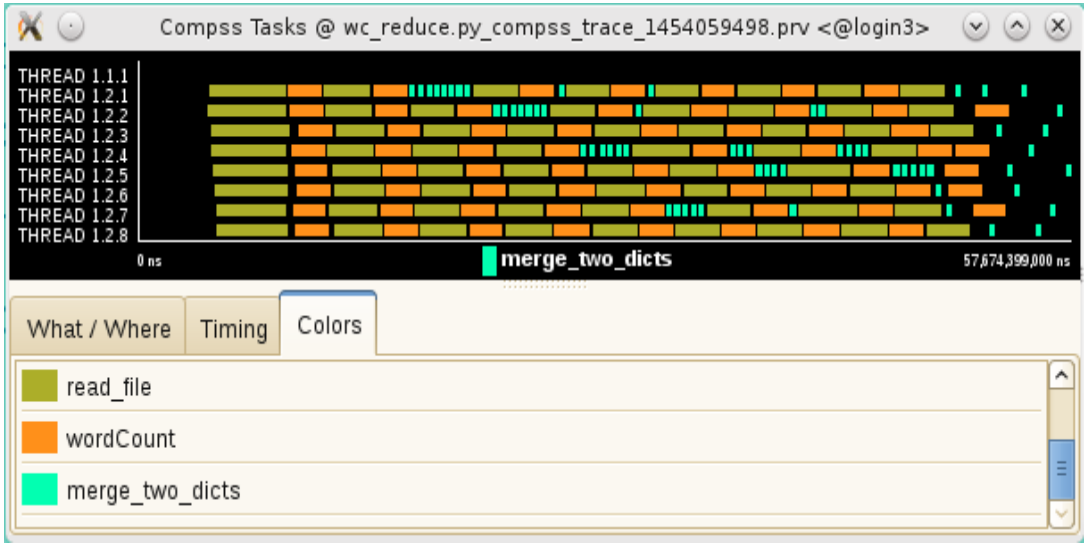
# Wordcount @ Performance Analysis

- Zoom to see details
  - Select a region in the trace window to see in detail
  - And repeat the process until the needed zoom level
  - The undo zoom option is in the right click panel



# Wordcount @ Performance Analysis

- Summarizing:
  - Lines in the trace:
    - One line for the master
    - N lines for the workers
- Meaning of the colours:
  - Black: idle
  - Other colors: task running
    - see the color legend
- Flags (events):
  - Start / end of task




# INSTALLATION

# Installation

- Installation manual:
  - http://compss.bsc.es/releases/compss/latest/docs/COMPSs_Installation_Manual.pdf
- Source code:
  - http://compss.bsc.es/ (Downloads Section – Source)
- Packages and repositories:
  - http://compss.bsc.es/ (Downloads Section – Repository references)
    - Debian based: apt-get install compss-framework
    - Zypper based: zypper install compss-framework
    - Yum based: yum install compss-framework
- Supercomputers:
    - $ wget http://compss.bsc.es/repo/sc/stable/COMPSs_2.0.tar.gz
    - $ tar -xvzf COMPSs_2.0.tar.gz
    - $ cd COMPSs
    - $ ./install <targetDir>
- Pip:
    - sudo -E pip install compss –v
    - source /etc/profile.d/compss.sh


BSC
Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

# FINAL NOTES

# Final Notes

- 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
  - Web services

# Final Notes

- Project page:
  - http://www.bsc.es/compss
- Direct downloads page:
  - http://www.bsc.es/computer-sciences/grid-computing/comp-superscalar/download
    - Virtual Appliance for testing & sample applications
    - Tutorials
    - Red-Hat & Debian based installation packages
    - Source Code
- Application Repository
  - http://compss.bsc.es/projects/bar/wiki/Applications
    - Several examples of applications developed with COMPSs

# Final Notes

- Looking for developers:
  - https://www.bsc.es/join-us/job-offers/job-offers-list/15cswdcdev

# Projects where COMPSs is used/developed








www.bsc.es


**Barcelona Supercomputing Center**
*Centro Nacional de Supercomputación*

# Thank you!

For further information please contact

support-compss@bsc.es