



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



Programming Distributed Computing Platforms with COMPSs

Workflows & Distributed Computing Group

26-27/01/2021

On-line

Outline

Day 2

- Session 6 (9:30-11:00): Java & C++
 - Writing Java applications
 - Java Hands-on + debug
 - C++ Syntax
- Coffee break (11:00 – 11:30)
- Session 7 (11:30-13:00): COMPSs Advanced Features
 - Using binaries and MPI code, Fault Tolerance and Exception management, Numba
 - COMPSs execution environment
- Lunch break (13:00 – 14:30)
- Session 8 (14:30-16:30): Cluster Hands-on (MareNostrum)
- COMPSs Installation & Final Notes
- SLIDES
 - http://compss.bsc.es/releases/tutorials/tutorial-PATC_2021/

Language differences overview

- Model and concepts are the same in all the Languages
- Differences in task declaration and synchronizations

Language	Task declaration	Sychronization
Python	On method implementation	Explicit (compss_open, compss_wait_on, wait_on_file)
Java	Task Definition Interface	Implicit (except getFiles)
C++	Task Definition Interface	Explicit (compss_open, compss_wait_on, wait_on_file)

Java Syntax



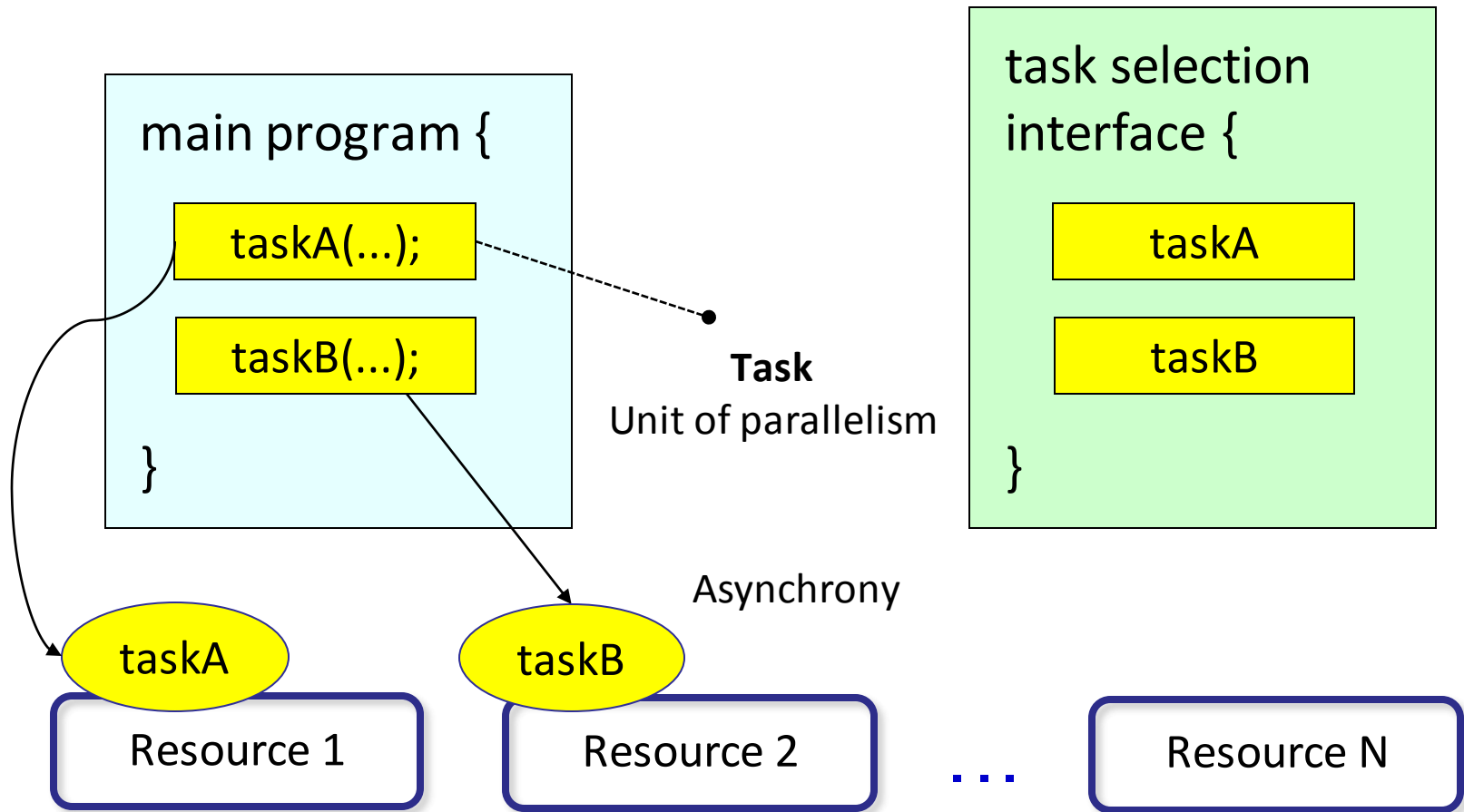
**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

Programming Steps

1. Identify tasks

2. Select tasks



Task Selection Interface

```
public interface SampleItf {  
    @Constraints(computingUnits = "1", memorySize = "0.5f")  
    @Method(declaringClass = "compss.Example")  
    void myMethod(  
        @Parameter(direction = INOUT) Reply r,  
        @Parameter(type = FILE, direction = OUT) String filename  
    );  
  
    @Service(namespace = "http://servicess.es/example",  
              name = "SampleService",  
              port = "SamplePort")  
    Reply myServiceOp(  
        @Parameter(direction = IN) Query q  
    );  
}
```

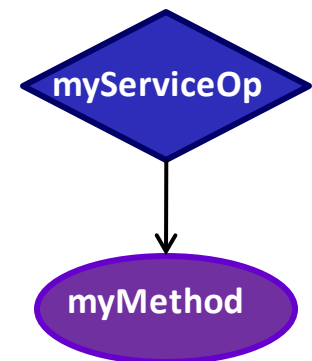
Main program

```
public class App {  
  
    public static void main(String[] args) {  
        Query query = new Query(...);  
  
        Reply reply = myServiceOp(query);  
  
        myMethod(reply, "out.txt");  
  
        reply.printToLog();  
    }  
}
```

Service task call

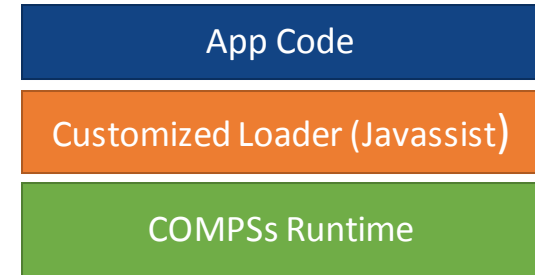
Method task call

Synchronization



Why we do not need to synchronize?

- Code instrumented with Javassist
 - Modified at loading time.



```
public class App {
    public static void main(String[] args) {
        COMPSsRuntime.start();
        Query query = new Query(...);
        Reply reply = myServiceOp(query); -> COMPSsRuntime.executeTask(...)
        myMethod(reply, "out.txt"); -> COMPSsRuntime.executeTask(...)
        COMPSsRuntime.getObject(reply);
        reply.printToLog();
        COMPSsRuntime.stop();
    }
}
```


COMPSs API calls

- There are some calls that can not be inferred and the user can use calling the COMPSs API
 - Static class COMPSs
- Barrier: wait for all tasks to finish
 - `COMPSs.barrier();`
- Deregister object
 - As objects are registered in the runtime. It prevents the Java GC to delete the object.
 - `COMPSs.deregisterObject(object);`
- Synchronize a file without opening
 - `COMPSs.getFile(filename);`

Java example



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

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);  
  
    }  
}
```

- Task Method

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

Sample Application (Interface)

- Task Annotation Interface

```
public interface SimpleItf {
```

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

Implementation

**Parameter
metadata**

Sample Application (Main Program)

- Main program NO CHANGES!
- No need to synchronize data COMPSs is doing itself!

```
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);  
  
    }  
    printCounters(counter1, counter2, counter3);  
}
```

← No need to synch

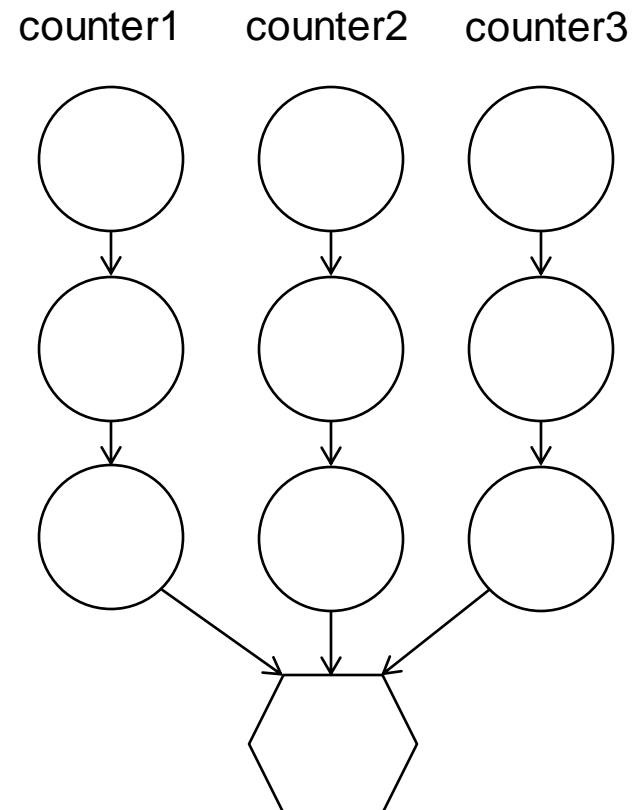
Sample Application: Task Graph

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

1st iteration

2nd iteration

3rd iteration



Java Hands-on

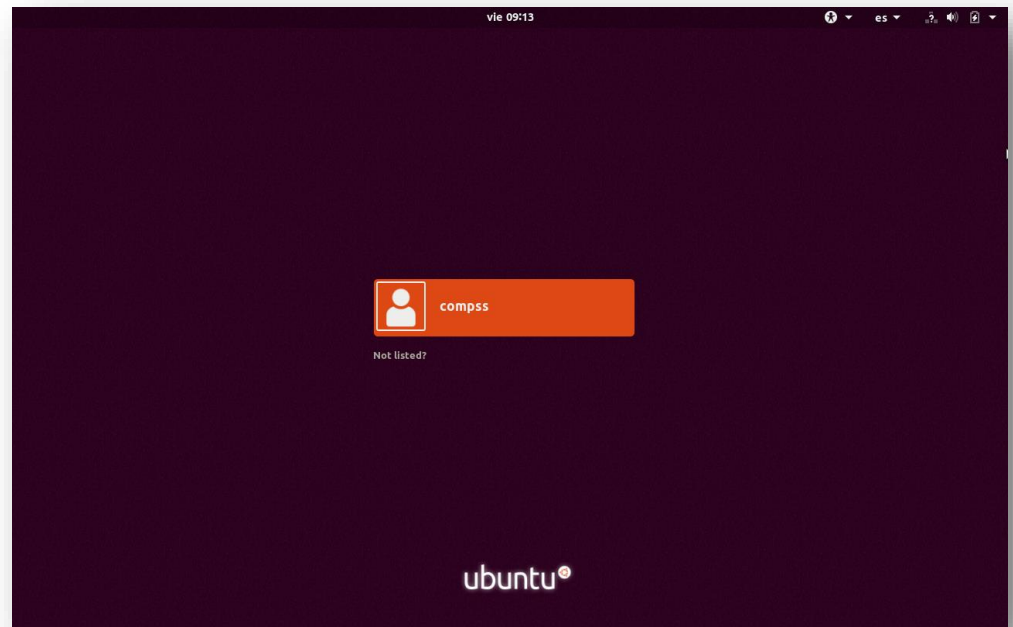


**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

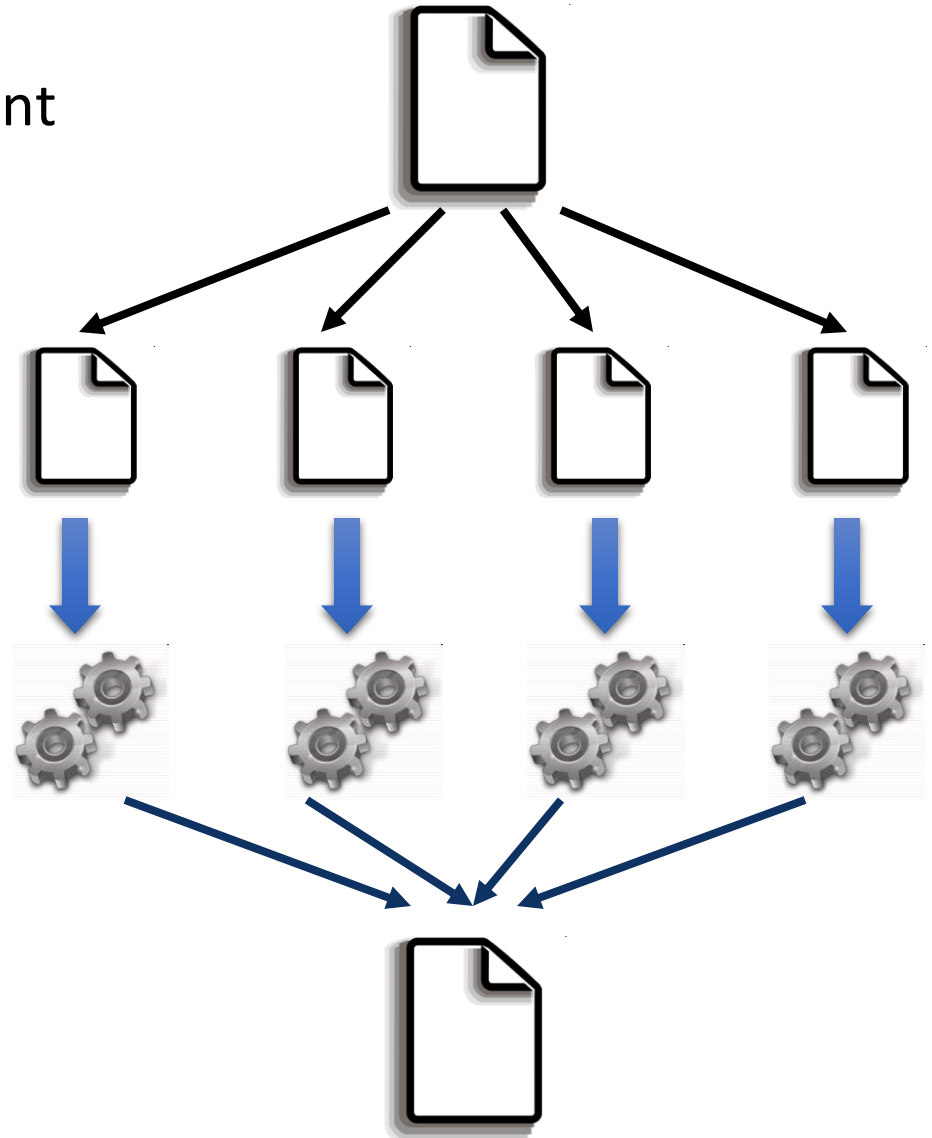
Hands-on environment

- Windows users: Start the Virtual Machine
 - User: **compss** `~/tutorial_apps/java`
 - Password: **compss2021**
- Open IDE (eclipse, netbeans,..)



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)



Compilation and Simple Execution

- Compilation
 - Run ***mvn clean install*** in `/home/compss/tutorial_apps/java/wordcount`
- Init the docker testing environment
 - `compss init -i compss/compss-tutorial:2.8`
- Use compss command to run the application
 - **compss run** [options] < FQDN app. classname> <application args>

Note: *compss run* command is used for running the application in the Docker testing environments in production installations this command has to be substituted by *runcompss*

- **Exercise:** Simple word count execution

- Usage:

`wordcount.uniqueFile.Wordcount <data_file> <block_size>`

```
$compss@bsc:~/> cd /home/compss/tutorial_apps/java/wordcount
$compss@bsc:/home/compss/tutorial_apps/java/wordcount/> compss run -classpath=jar/wordcount.jar
wordcount.uniqueFile.Wordcount data-set/file_small.txt 650
```



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: Result

```
$compss@bsc:~/tutorial_apps/java/wordcount/jar/> compss run -classpath=jar/wordcount.jar  
wordcount.uniqueFile.Wordcount data-set/file_small.txt 650
```

```
Executing cmd: runcompss --project=/project.xml --resources=/resources.xml ...
```

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

```
WARNING: COMPSs Properties file is null. Setting default values
```

```
[ API] - Starting COMPSs Runtime v2.6 (build xxxx)
```

```
DATA_FILE parameter value = data-set/file_small.txt
```

```
BLOCK_SIZE parameter value = 650
```

```
[LOG] Computing word count result
```

```
[LOG] Counted Words is : 247
```

```
[ API] - No more tasks for app 1
```

```
[ API] - Getting Result Files 1
```

```
[ API] - Execution Finished
```



Application Logs

Java Hands-on: Configuration

- Project.xml:
 - `comps exec cat /project.xml`

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

- Other optional parameters
 - User, AppDir, LibraryPath

Java Hands-On: Configuration

- Resources.xml:
 - compss exec cat /resources.xml

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

Affects to
application
parallelism

Simulating several workers

- Add workers:
 - Create other containers in your machine to run as workers

```
$compss@bsc:/home/compss/tutorial_apps/java/wordcount/> compss components add worker 2
```

- Check project and resources

```
$compss@bsc:/home/compss/tutorial_apps/java/wordcount/> compss exec cat /project.xml
```

```
$compss@bsc:/home/compss/tutorial_apps/java/wordcount/> compss exec cat /resources.xml
```

- Execute again

```
$compss@bsc:/home/compss/tutorial_apps/java/wordcount/> compss run -classpath=jar/wordcount.jar  
wordcount.uniqueFile.Wordcount data-set/file_small.txt 650
```


Java Hands-On: Monitoring

- The runtime of COMPSs provides real-time monitoring to follow the progress of the executions
 - Running tasks, resources usage, execution time per task, real-time execution graph, etc.
- Start monitor and open browser
 - **compss monitor start**
 - <http://localhost:8080/compss-monitor/>
- Activate monitoring with a `runcompss/compss run` flag
 - Setting a monitoring interval
 - **compss run --monitoring=<int>**
 - With a default monitoring interval
 - **compss run -m** (or) **compss run --monitoring**
- **Exercise:** run wordcount enabling monitoring

Note: in production installations
`/etc/init.d/compss-monitor start`

Note: `compss run` command is used for running the application in the Docker testing environments in production installations this command has to be substituted by `runcompss`

```
$compss@bsc:/home/compss/tutorial_apps/java/wordcount/jar/> compss run -m -classpath=jar/wordcount.jar  
wordcount.uniqueFile.Wordcount data-set/file_long.txt 350000
```



Java Hands-on: Graph generation

- To generate the graph of an application, it must be run with the monitor or graph flags activated
 - `compss run -m | -graph | -g`
- The graph will be stored in:
 - `.COMPSs/<APP_NAME>_<EX#>/monitor/complete_graph.dot`
- To convert the graph to a PDF format:
 - `compss gengraph <dot_file>`
- **Exercise:** generate the graph for the wordcount application

Remember: `compss run` command environments in production installations this command has to be substituted by `runcompss`

```
$compss@bsc:~/tutorial_apps/java/wordcount> compss run -g -classpath=jar/wordcount.jar wordcount.uniqueFile.Wordcount data-set/file_small.txt 650
```

```
... application execution ...
```

```
$compss@bsc:~/tutorial_apps/java/wordcount> compss gengraph .COMPSs/wordcount.uniqueFile.Wordcount_04/monitor/complete_graph.dot
```

```
Output file: .COMPSs/wordcount.uniqueFile.Wordcount_04/monitor/complete_graph.pdf
```

```
$compss@bsc:~/tutorial_apps/java/wordcount> evince .COMPSs/wordcount.uniqueFile.Wordcount_04/monitor/complete_graph.pdf
```



Java Hands-on: Debugging

- Different log levels activated as options
 - `--log_level=<level>`
(**off**: for performance | **info**: basic logging | **debug**: detect errors)
 - `-debug` or `-d`
- The output/errors of the main code of the application are shown in the console
- Logging files are stored by default in:
 - `$HOME/.COMPSSs/<APP_NAME>_XX`
 - Customizable with flag `-base_log_dir=<path>`
- Inside this folder, the user can check :
 - The output/error of a task # N : `jobs/jobN.[out|err]`
 - Messages from the COMPSSs : `runtime.log`
 - Worker: `$HOME/.COMPSSs/<app_name_XX>/workers`
or during the execution at `/<working_dir>/<uuid>/<node_name>/logs`
- **Exercise:** run wordcount with debugging

Note: By default in production installations the log folder is in `$HOME/.COMPSSs`. In the Docker environment, it is set to appear in `$PWD/.COMPSS`. In this case, when using Jupyter the log dir flag is not set and you have to do `pycompss exec cp /root/.COMPSSs` to see the logs

```
$$compss@bsc:/home/compss/tutorial_apps/java/wordcount/jar/> compss run -d -classpath=jar/wordcount.jar  
wordcount.uniqueFile.Wordcount /home/compss/workspace_java/wordcount/data/file_small.txt 650
```



Demo

- Common errors:
 - Exceptions
 - In main code
 - Within a task
 - Usage of non-serializable objects
 - As a parameters
 - As a return
 - Connectivity problems
 - The master can not connect to the worker
 - The worker can not connect to the master

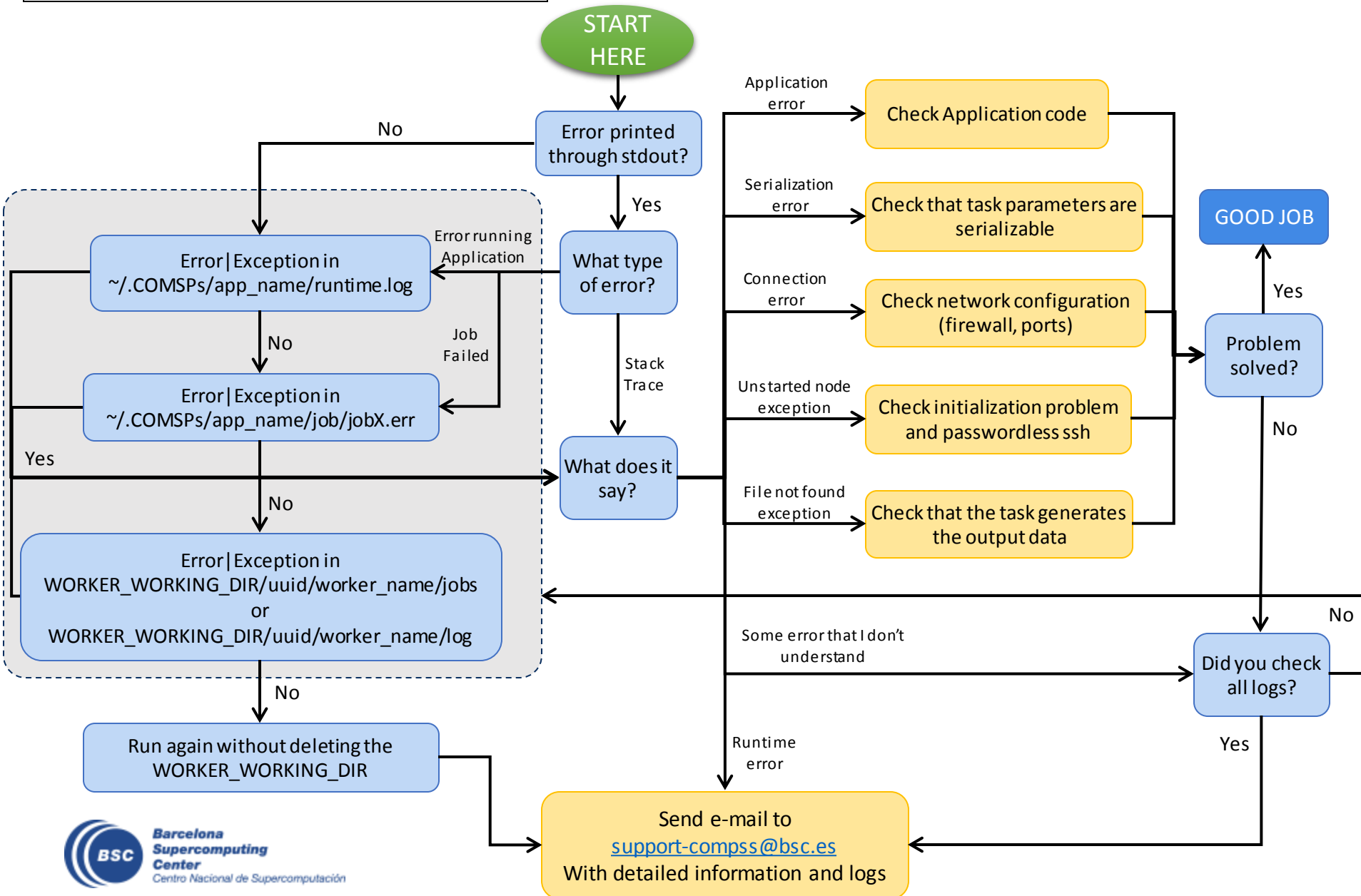


Default log locations:

MASTER LOGS: ~/.COMPSs

WORKER_WORKING_DIR = /tmp/COMPSsWorker/

Debugging process



C Syntax



**Barcelona
Supercomputing
Center**

Centro Nacional de Supercomputación

COMPSs C++ Binding

- Application Structure
- C Binding API
- Task definition /Supported data
- Compilation & Execution

Application Structure

- Main Code
 - <AppName>.cc
- Task definition interface
 - <AppName>.idl
- Task Implementation
 - <AppName>-functions.cc
- Auxiliary classes and methods
 - src folder

C++-Binding API

- Similar to the Python Binding
- Start/stop
 - **compss_on()** / **compss_off()**
- Synchronize and delete Objects
 - `template <class T> void compss_wait_on(T* &obj);`
 - `template <class T> T compss_wait_on(T &obj);`
 - `template <class T> int compss_delete_object(T* &obj);`
- Synchronize and delete files
 - `void compss_ifstream(char * filename, ifstream& ifs);`
 - `void compss_ofstream(char * filename, ofstream& ofs);`
 - `FILE* compss_fopen(char * filename, char * mode);`
 - `void compss_wait_on_file(char * filename);`
 - `void compss_delete_file(char * filename);`
- Barrier
 - `void compss_barrier();`

Task definition

- Task definition interface (IDL-like interface)
 - Task definition
 - [return_type|void] [static][class_name::]method_name(params...);
 - Param definition
 - [in|out|inout]data_type name
 - Supported Data types for Dependencies
 - Files: *file*, objects: *Class_name*, 1D Arrays: *type[#elems]*
 - Primitive data types only IN direction: *char*(string)*, *int*, *double*, *float*,...

```
interface example {
  @Constraints(ComputingUnits=2)
  void method(in f_in, out file f_out);
  //Expected C++ method: void method(char* f_in, char* f_out)

  ObjectEx ObjectEx:objectMethod(in inout ObjectEx accum,);
  //Expected C++ method: ObjectEx* ObjectEx:objectMethod(ObjectEx* accum)

  double[20] normal_method(in int n, in double[n] in_array);
  //Expected C++ method: double* normal_method(int n, double*)
}
```

Compilation & Execution

- Compilation

- Generate master/worker stubs (C++ do not support reflection)
- Command:
 - `comps_build_app <appName>`

- Execution

- Same as Python/Java (runcomps command)
 - `runcomps master/<appName> [app_args]`
- Require to set the AppDir
 - In the project.xml
 - Homogeneous cluster:
 - `runcomps -appdir`

```
<Project>
  <MasterNode/>
  <ComputeNode Name="localhost">
    <InstallDir>/opt/COMPSS/</InstallDir>
    <WorkingDir>/tmp/WorkerLocalhost/</WorkingDir>
    <Application>
      <AppDir>/home/tutorial_apps/c/matmul_objects/</AppDir>
    </Application>
  </ComputeNode>
</Project>
```

Exercise

- Matrix Multiplication with c-binding
 - `cd ~/tutorial_apps/c/matmul_object`
- Compile:
 - `compss exec compss_build_app Matmul`
- Execute:
 - `compss -appdir=/home/user/ master/Matmul 4 4 2.0`



**Barcelona
Supercomputing
Center**
Centro Nacional de Supercomputación



EXCELENCIA
SEVERO
OCHOA

THANK YOU!

support-compss@bsc.es

www.bsc.es