SSH GRID superscalar: a tailored version for clusters

Jorge Ejarque
Rosa M. Badia, Pieter Bellens, Josep M. Perez,
Jesus Labarta, Marc de Palol, Raül Sirvent, Enric Tejedor
Barcelona Supercomputing Center (BSC-CNS)
Technical University of Catalonia (UPC)

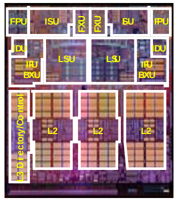
Outline

- GRID superscalar overview
- SSH version design
- MareNostrum Execution Scenarios
- Scheduling Policies
- Conclusions and Future Work

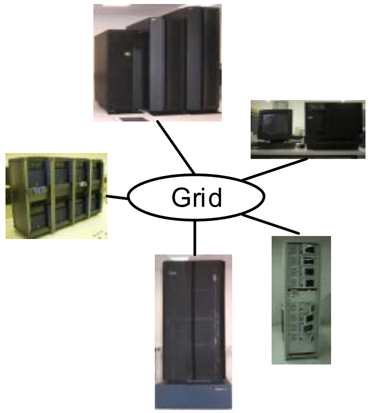
GRID superscalar overview

- Ease the programming of GRID applications

- Basic idea:



≅



ns ➔ seconds/minutes/hours

GRID superscalar overview

- Reduce the development complexity of Grid applications to the minimum
  - Writing an application for a computational Grid may be as easy as writing a sequential application
- Target applications: composed of tasks, most of them repetitive
  - Granularity of the tasks of the level of simulations or programs

GRID superscalar overview

- Three components:
  - Main program
  - Subroutines/functions
  - Interface Definition Language (IDL) file
- Programming languages:
  - C/C++, Perl, Java
  - Prototype version for shell script

GRID superscalar overview

- Master code

```
GS_On();
for (int i = 0; i < MAXITER; i++) {
   newBWd = GenerateRandom();
   subst (referenceCFG, newBWd, newCFG);
   dimemas (newCFG, traceFile, DimemasOUT);
   post (newBWd, DimemasOUT, FinalOUT);
   if(i % 3 == 0) Display(FinalOUT);
}
fd = GS_FOpen(FinalOUT, R);
printf("Results file:\n"); present (fd);
GS_FClose(fd);
GS_Off(0);
```

- Interface Definition Language (IDL) file
  - In/Out/InOut files or scalars
  - The functions listed will be executed in a remote server in the Grid.

```
interface OPT {
void subst(in File referenceCFG, in double latency, in double bandwidth, \
    out File newCFG);
void Dimemas(in File cfgFile, in File traceFile, in double goal, out File \
    DimemasOUT);
void post(in double bw, in File DimemasOUT, inout File resultFile);
};
```

GRID superscalar overview

- Subroutines/functions

```
void dimemas(char *newCFG, char *traceFile, char *DimemasOUT)
{
   char command[500];

   putenv("DIMEMAS_HOME=/usr/local/cepba-tools");
   sprintf(command, "/usr/local/cepba-tools/bin/Dimemas -o %s %s",
         DimemasOUT, newCFG );
   GS_System(command);
}
```

```
void display(char *toplot)
{
...
oplot);
```

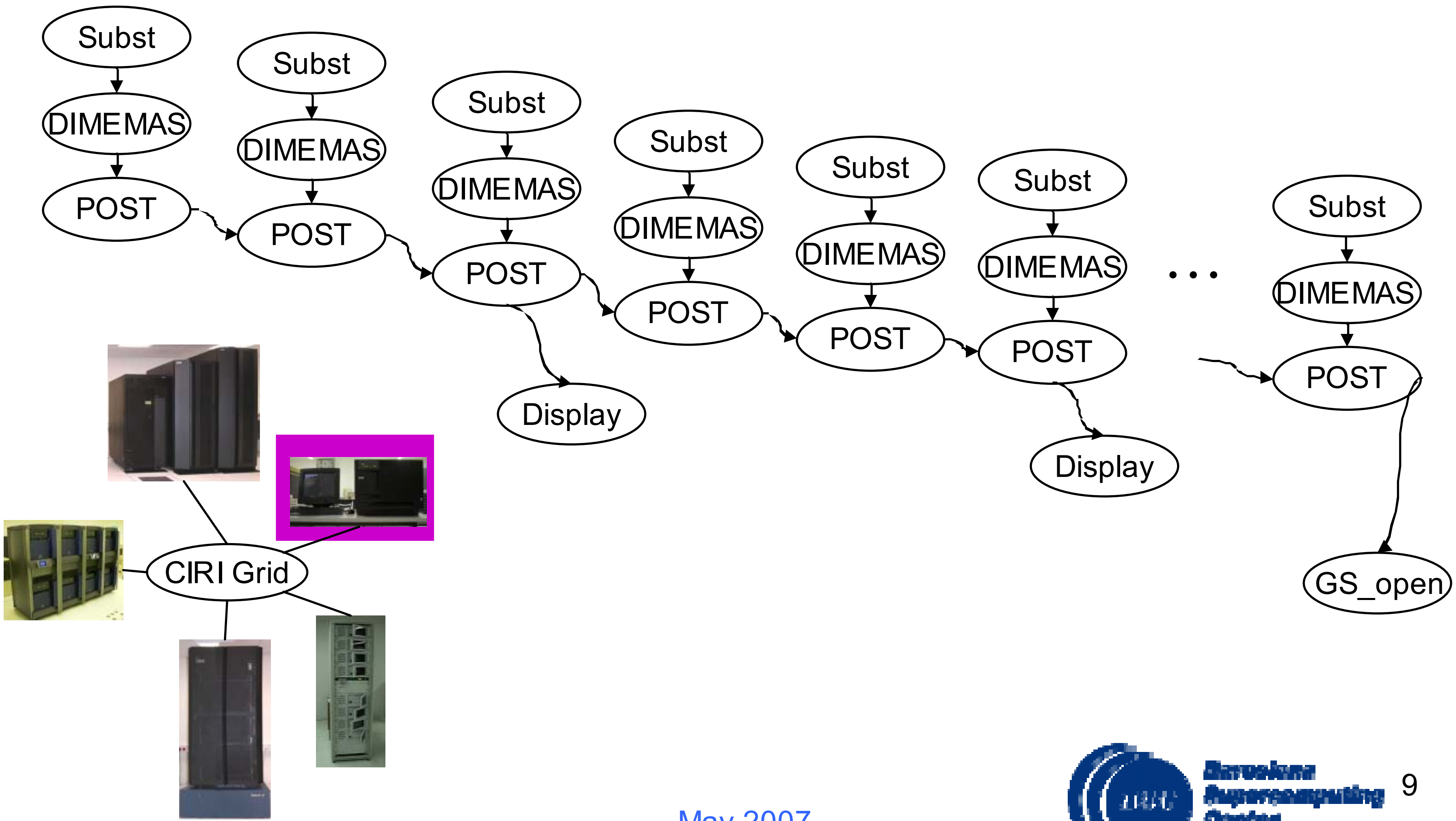
```
void concat(char *f1, char *f2, char *fout){
FILE *fp;
int i,j,k;

for (i=1; i<1000; i++)
        for (j=0; j<1000; j++)
                k= j%i;

fp = fopen(fout,"w");
fprintf(fp,"Call to concat(%s, %s, %s)\n", f1, f2, fout);
fclose(fp);

}
```

GRID superscalar overview

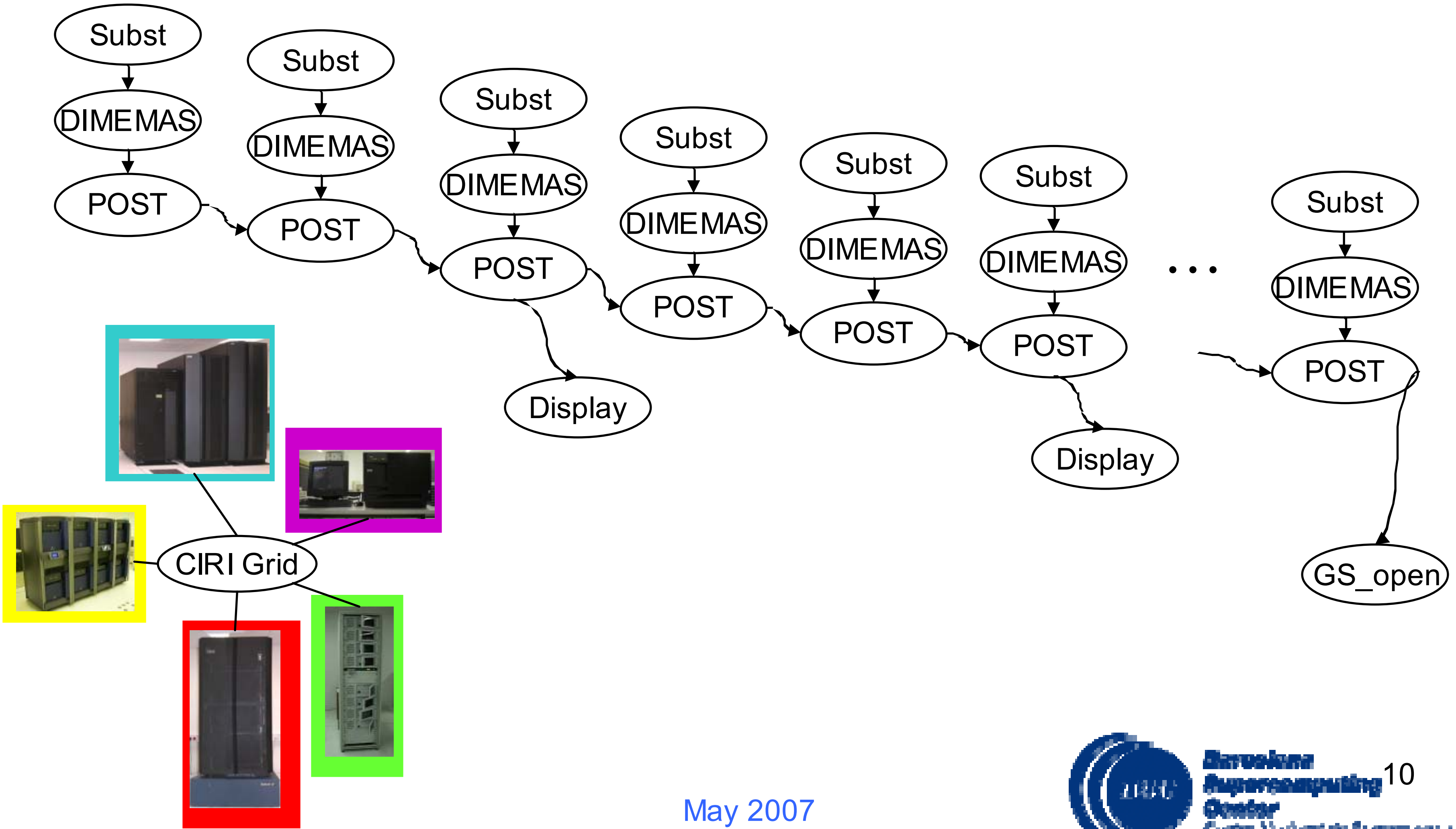
Input/output data

```
for (int i = 0; i < MAXITER; i++) {
   newBWd = GenerateRandom();
   subst (referenceCFG, newBWd, newCFG);
   dimemas (newCFG, traceFile, DimemasOUT);
   post (newBWd, DimemasOUT, FinalOUT);
   if(i % 3 == 0) Display(FinalOUT);
}
fd = GS_Open(FinalOUT, R);
printf("Results file:\n"); present (fd);
GS_Close(fd);
```

GRID superscalar overview



GRID superscalar overview



SSH/SCP version design

Motivation

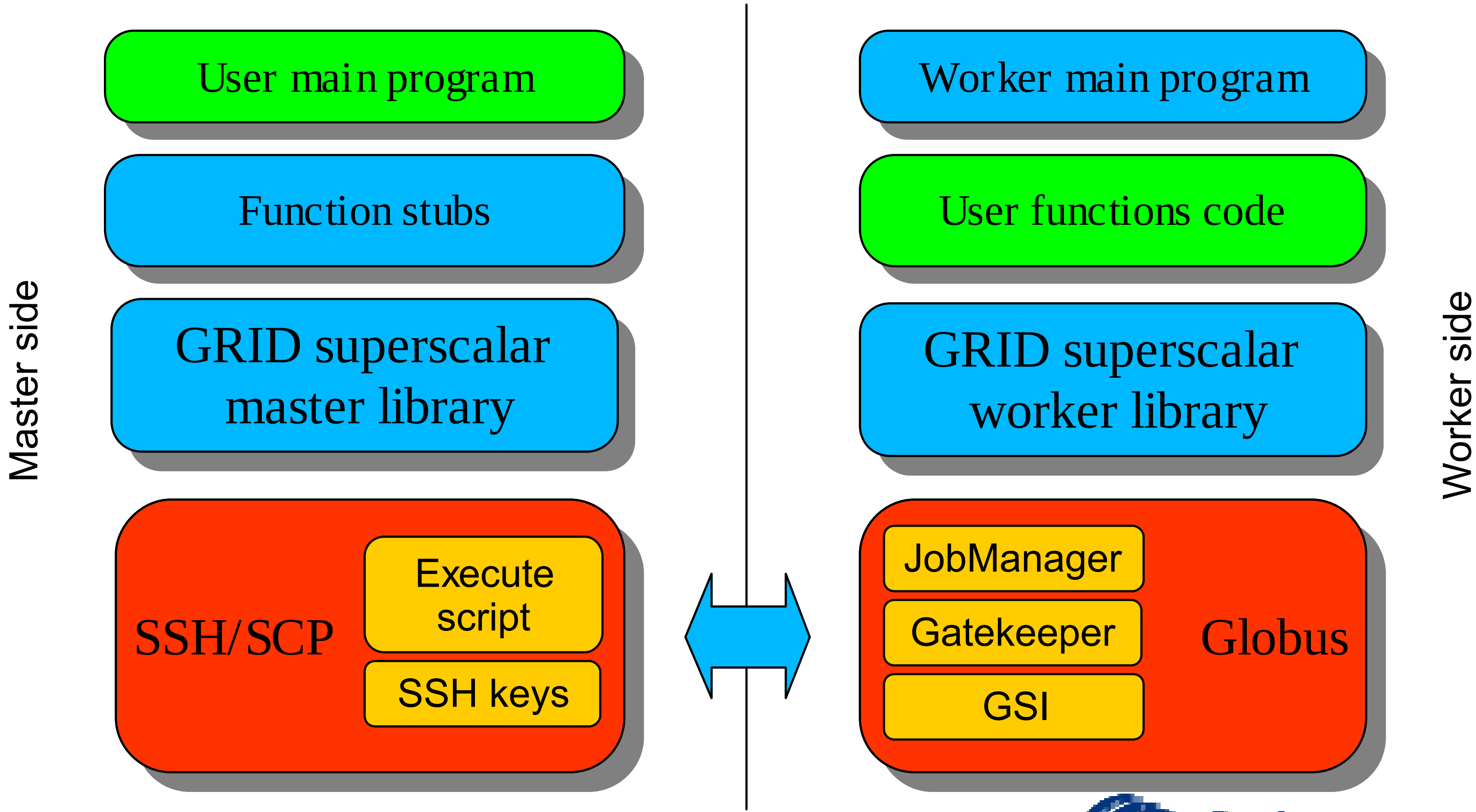
- Ease the programming of Grid and cluster applications

- Avoid Grid middleware problems
  - Difficult to install
  - Not available in all systems
  - Reduce the overhead

- Why SSH/SCP
  - SSH and SCP are available in most of modern systems
  - Robust
  - Secure

SSH/SCP version design

GRID superscalar application architecture

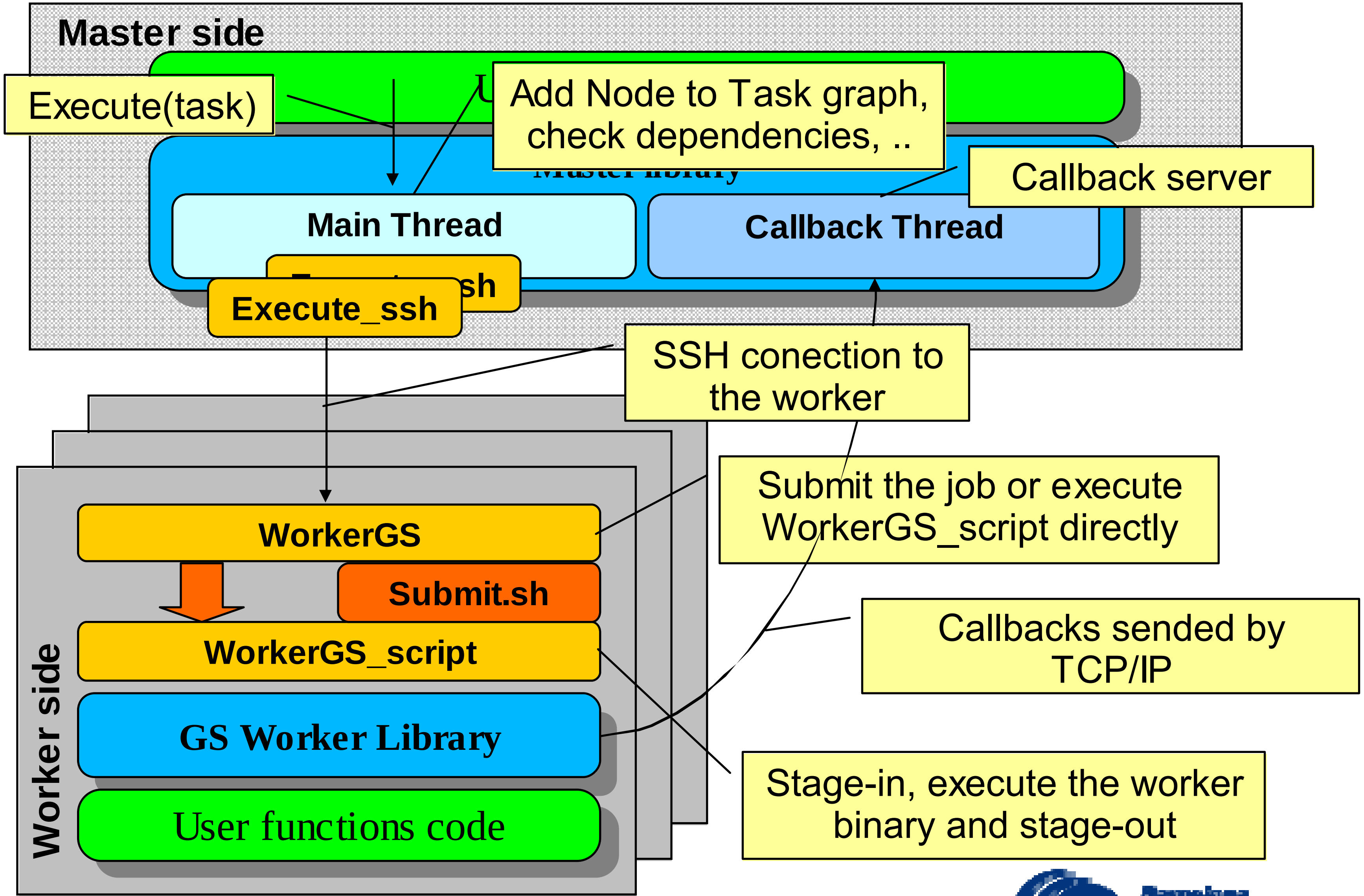


SSH/SCP version design

- The same user interface as in globus version
  - User code compatible with other versions
  - No changes are needed
  - The same code generation tools

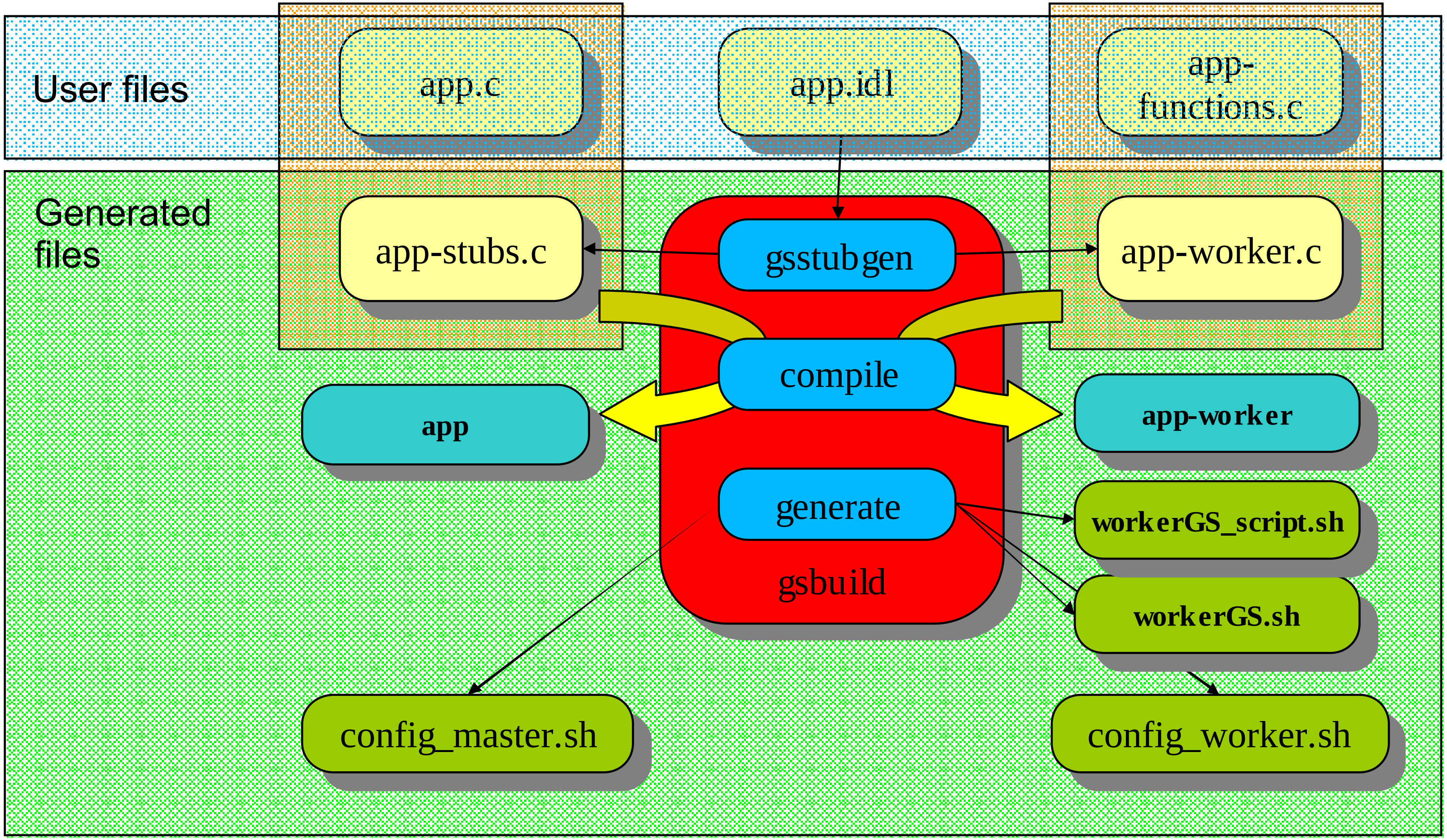
- Required middleware functionalities substituted by scripts with ssh/scp calls
  - Possibility to choose other remote commands (rsh, rcp)
  - All scripts are auto generated at compile or configuration time

SSH/SCP version design



Barcelona Supercomputing Center
Centro Nacional de Supercomputación

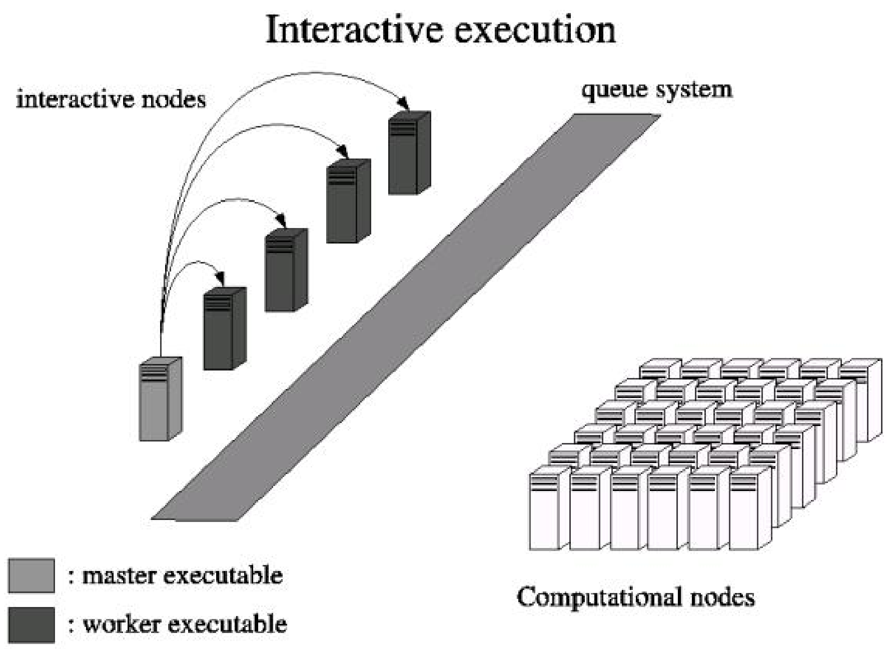
SSH version design: gsbuild

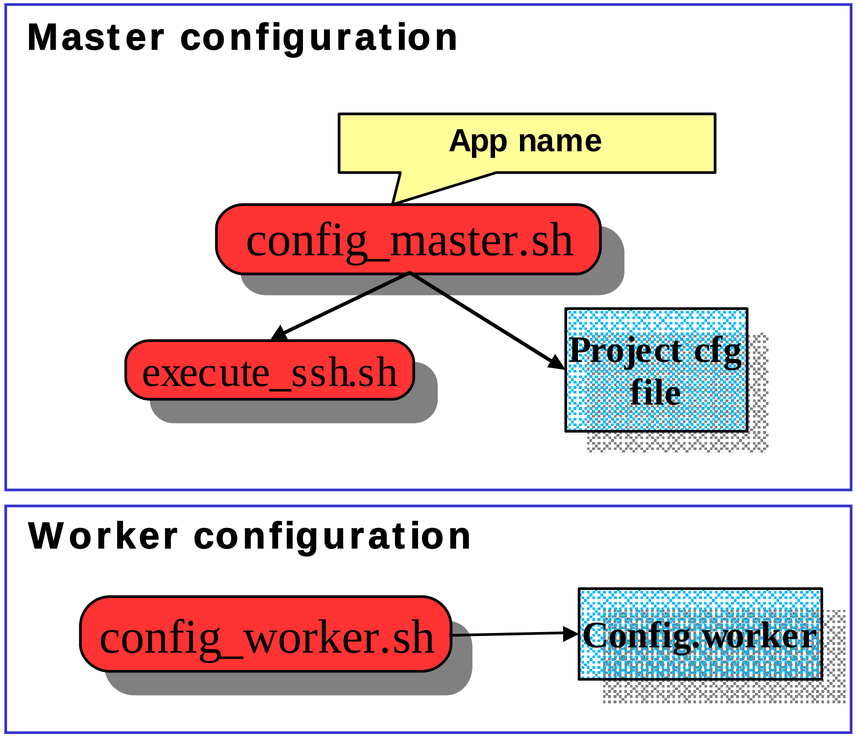


MareNostrum Execution Scenarios

- Deployment tool not needed in clusters
- Configuration scripts
  - Generate the execution and submission script
  - The user can define:
    - Remote commands
    - Queue system
    - Environment variables (PATH, LIBS, CLASSPATH,...)
- Four different execution scenarios
  - Interactive
  - Enqueue the whole application
  - Master interactive, queued workers
  - Queued master , queued workers

MareNostrum Execution Scenarios





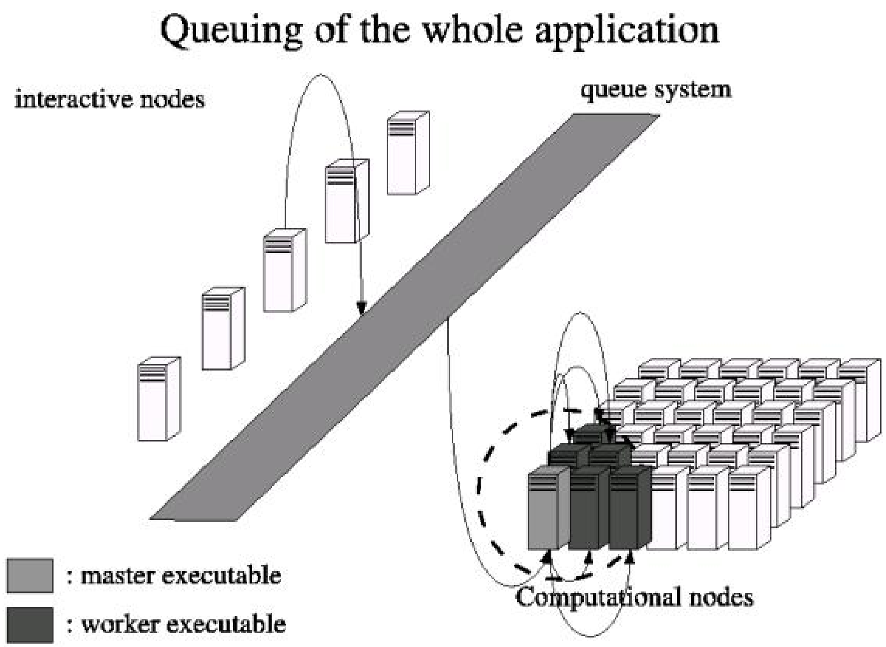
Advantages

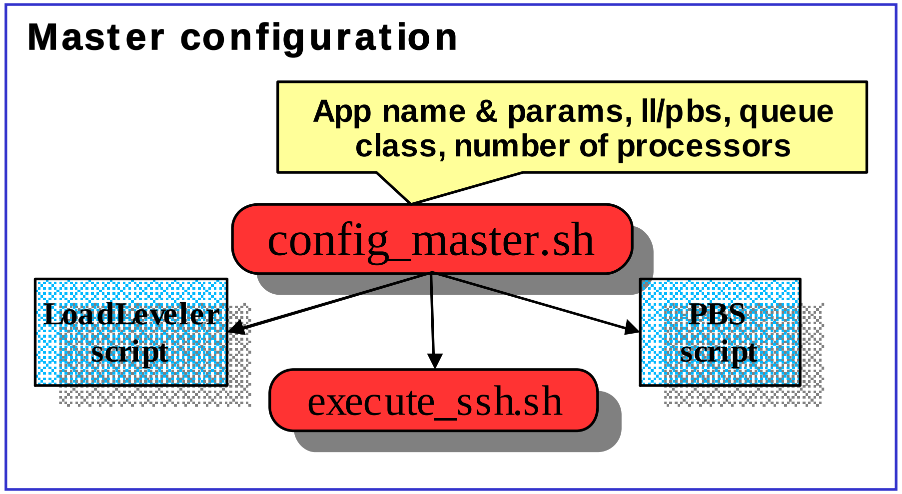
- Suitable for application debugging
- Fast execution

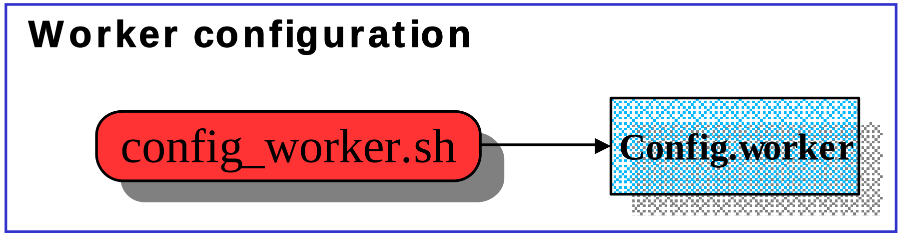
Drawbacks

- Users' executions are not isolated
- CPU time limited in interactive nodes

MareNostrum Execution Scenarios







Advantages

- Suitable for production executions
- Suitable for applications with fine grained tasks

Drawbacks

- All resources allocated during the whole execution
- Queue overhead at the beginning
- Difficult to allocate large number of nodes

Queued workers scenarios

- Need to add the worker queue class in config_master
- Master interactive, queued workers
  - Isolate the computational jobs (only master in interactive nodes)
  - Overhead of queuing each task (suitable for applications with coarse grained tasks)
- Master queued, queued workers
  - Reduces the initial overhead (easy to allocate single slot for the master only)
  - Overhead of queuing each task (suitable for applications with coarse grained tasks)
  - Resource reuse (not all resources allocated during the whole application)

Scheduling policies

- Time duration estimation
- FIFO
- Weight graph

Scheduling policies

- Useful for executions in a grid
  - The classAd library is used to match resource properties with task requirements
  - If more than one resources fulfils the constraint, the resource which minimizes this formula is selected:

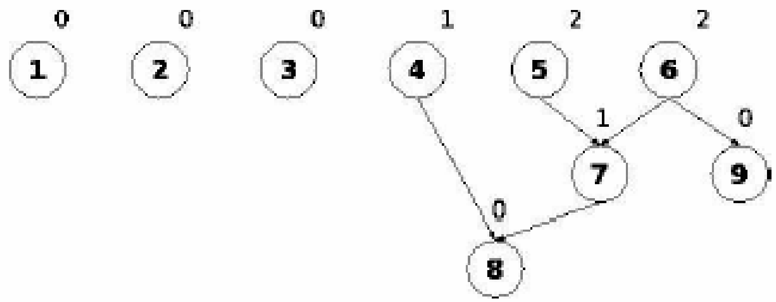
$$f(t,r) = \alpha \, FT(r) + \beta \, ET(t,r)$$

    - t = task
    - r = resource
    - FT = File transfer time to resource r
    - ET = Execution time of task t on resource r (using user provided cost function)

Scheduling policies: Cluster

- FIFO policy
  - Do not create overhead

- Weight-based policy
  - Assign a weight for each tasks depending how critical is this tasks in the graph (number of successors)
  - Execute first the most critical tasks



Barcelona Supercomputing Center

Scheduling Policies

- FIFO vs Weight-based
  - Cholesky decomposition that works in blocks
  - 22 processors
- Results
  - Weight assignment and decision time negligible comparing with the task duration
  - Speed up between 1-1.2

	8x8	13x13	18x18	23x23	28x28	32x32
Total Tasks	191	636	1481	2851	4871	7039
Assign Weights	$41 \mu sec$	$163 \mu sec$	$482 \mu sec$	$1224 \mu sec$	$2481 \mu sec$	$3857 \mu sec$
Mean Decision	$\backsim 1\ \mu sec$	$\backsim 2\ \mu sec$	$\backsim 2\ \mu sec$	$\backsim 2\ \mu sec$	$\backsim 3\ \mu sec$	$\backsim 3\ \mu sec$

Policies	Matrix block sizes				
	8x8	13x13	18x18	23x23	28x28
FIFO	158 *sec*	712 *sec*	863 *sec*	1591 *sec*	2214 *sec*
WEIGHT	152 *sec*	589 *sec*	721 *sec*	1395 *sec*	2036 *sec*
Speedup	1,04	1.20	1.19	1.14	1.09

Conclusions and Future work

- SSH GRID superscalar
  - The required middleware functionalities substituted by script with ssh/scp calls
  - Relieves from the middleware overhead
  - Valid paradigm also for cluster applications

- Ongoing and Future Work
  - Fault tolerance mechanisms (currently, task level checkpointing)
  - Extend the MareNostrum solution to the Spanish Supercomputing Network (*Red Española de Supercomputación*)

More information

- GRID superscalar home page:
  www.bsc.es/grid/grid_superscalar