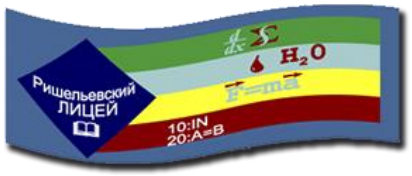


Ришельевский лицей
при Одесском национальном университете
1817...1989...

В дни успеха и печали
для нас ты - Alma Mater

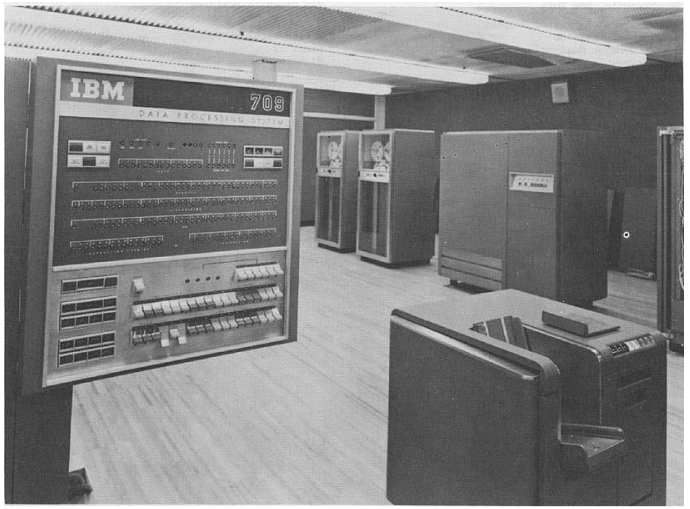
Параллельное программирование на языке Python в системе COMPSs

Зайцев Николай

История параллельного программирования

- Первым компьютером, использующим разрядно параллельную память и разрядно-параллельную арифметику, **IBM 704** (1955)

- Через несколько лет конструкторы присоединили 6 независимых процессоров ввода/вывода, которые после получения команд могли работать параллельно с основным процессором. Эта модель получила название **IBM 709** (1958)



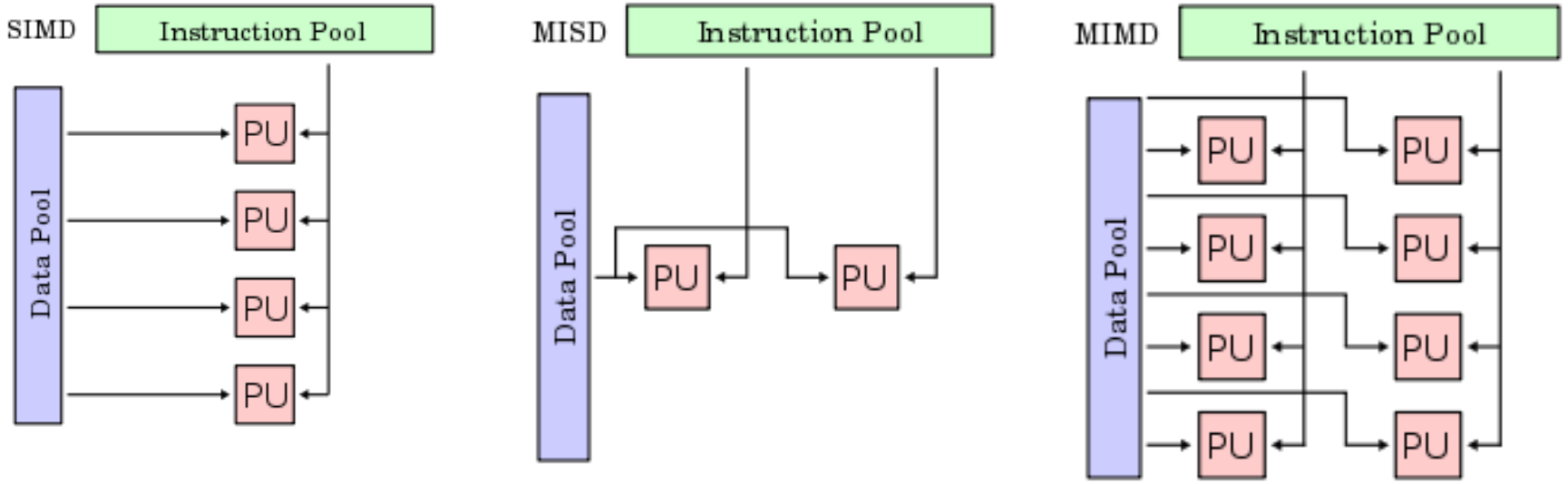
История параллельного программирования

Работы в направлении создания параллельных вычислительных систем в США и СССР велись интенсивно с 1960-х годов. Основные места в которых велись исследования по этой теме: Иллинойский университет, Burroughs Corporation, Вычислительный центр АН СССР.

История параллельного программирования

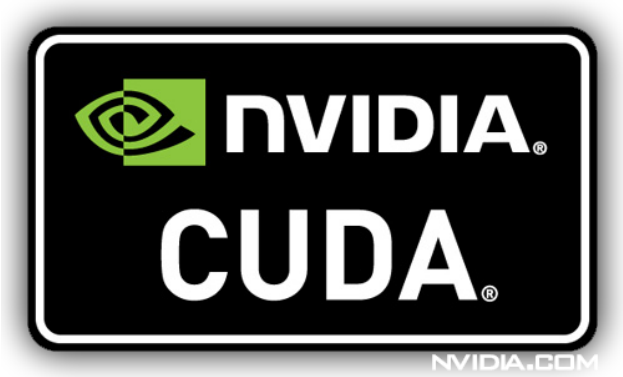
Принципы параллельных компьютерных вычислений:

- **SIMD** (Single Instruction, Multiple Data);
- **MISD** (Multiple Instruction stream, Single Data);
- **MIMD** (Multiple Instruction stream, Multiple Data).



Современные средства параллельного программирования

- MPI (Message Passing Interface)
- OpenMP
- Программно-аппаратная архитектура для графических процессоров (Для видеокарт фирмы Nvidia – CUDA, для AMD – OpenCL)
- PVM(Parallel Virtual Machine)



OpenCL

PVM

stands for

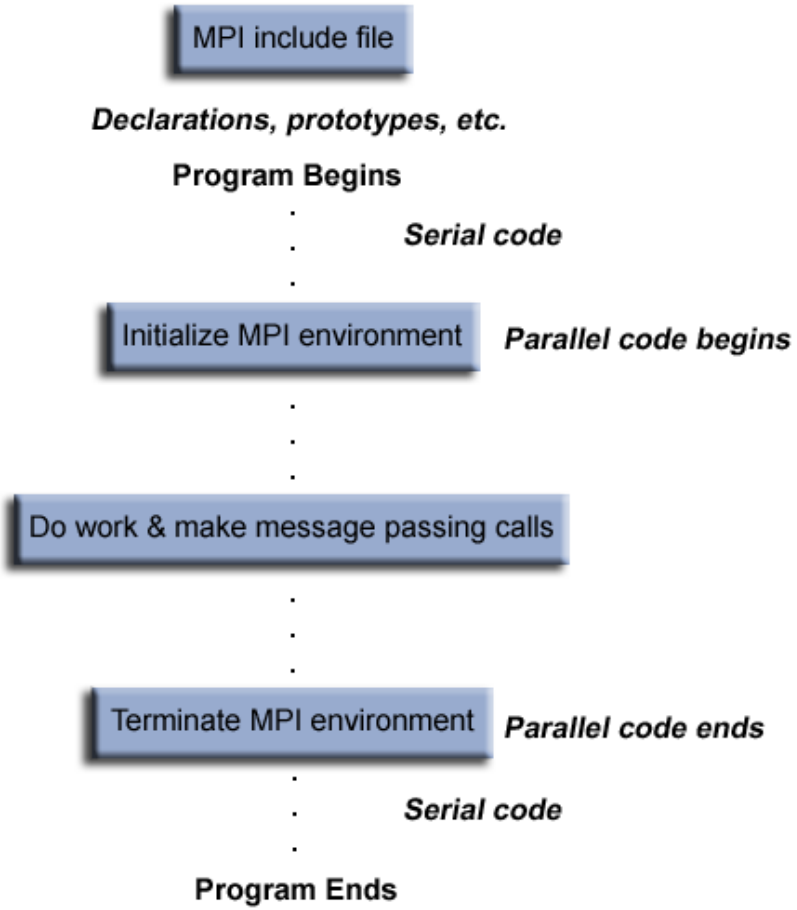
Parallel Virtual Machine

MPI

Первая версия MPI разрабатывалась Уильямом Гроуппом, Эвином Ласком и вышла в 1994.

Основными понятиями стандарта MPI являются: процесс, группа процессов, коммуникатор.

В первую очередь MPI ориентирован на системы с распределенной памятью в то время как OpenMP ориентирован на системы с общей памятью.



Пример программы на MPI

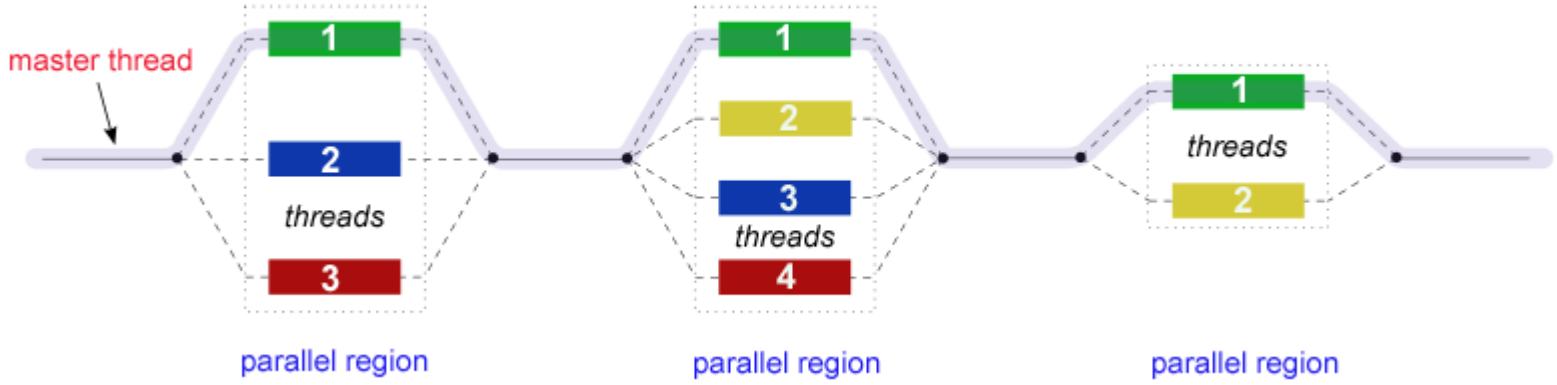
```
#include "mpi.h"
#include <stdio.h>
int main(int argc, char **argv)
{
   int rank;
   float a, b;
   MPI_Status status;
   MPI_Init(&argc, &argv);
   MPI_Comm_rank(MPI_COMM_WORLD, &rank);
   a = 0.0;
   b = 0.0;
   if(rank == 0)
   {
      b = 1.0;
      MPI_Send(&b, 1, MPI_FLOAT, 1, 5, MPI_COMM_WORLD);
      MPI_Recv(&a, 1, MPI_FLOAT, 1, 5, MPI_COMM_WORLD, &status);
   }
   if(rank == 1)
   {
      a = 2.0;
      MPI_Recv(&b, 1, MPI_FLOAT, 0, 5, MPI_COMM_WORLD, &status);
      MPI_Send(&a, 1, MPI_FLOAT, 0, 5, MPI_COMM_WORLD);
   }
   printf("process %d a = %f, b = %f\n", rank, a, b);
   MPI_Finalize();
}
```

OpenMP

Разработана компанией Architecture Review Board (ARB).

Первая версия появилась в 1997 году, предназначалась для языка Fortran.

OpenMP реализует параллельные вычисления с помощью много поточности, в которой «главный» (master) поток создает набор подчиненных (threads) легковесных процессов (нитей) и задача распределяется между ними.

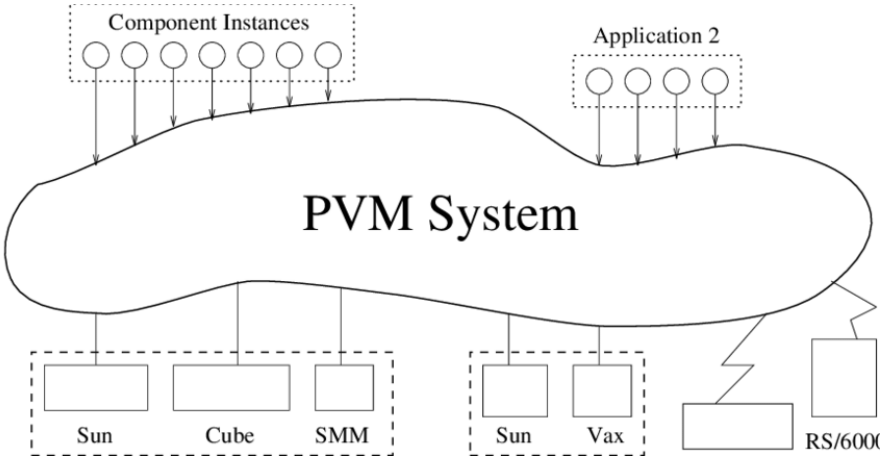


Пример программы на OpenMP

```
#include <stdio.h>
#include <omp.h>
int main(int argc, char *argv[])
{
   int A[10], B[10], C[10], i, n;
/* Заполним исходные массивы */
   for (i=0; i<10; i++){ A[i]=i; B[i]=2*i; C[i]=0; }
#pragma omp parallel shared(A, B, C) private(i, n)
   {
      /* Получим номер текущей нити */
      n=omp_get_thread_num();
#pragma omp for
      for (i=0; i<10; i++)
      {
         C[i]=A[i]+B[i];
         printf("Нить %d сложила элементы с номером %d\n",
                n, i);
      }
   }
}
```

PVM

Parallel Virtual Machine (PVM) - программный пакет, позволяющий объединять разнородный набор компьютеров в общий вычислительный ресурс («виртуальную параллельную машину»). Имеет более расширенные возможности в плане контроля вычислений: присутствует специализированная консоль управления параллельной системой и её графический эквивалент XPVM, позволяющий наглядно продемонстрировать работу всей системы.



Программирование на GPU

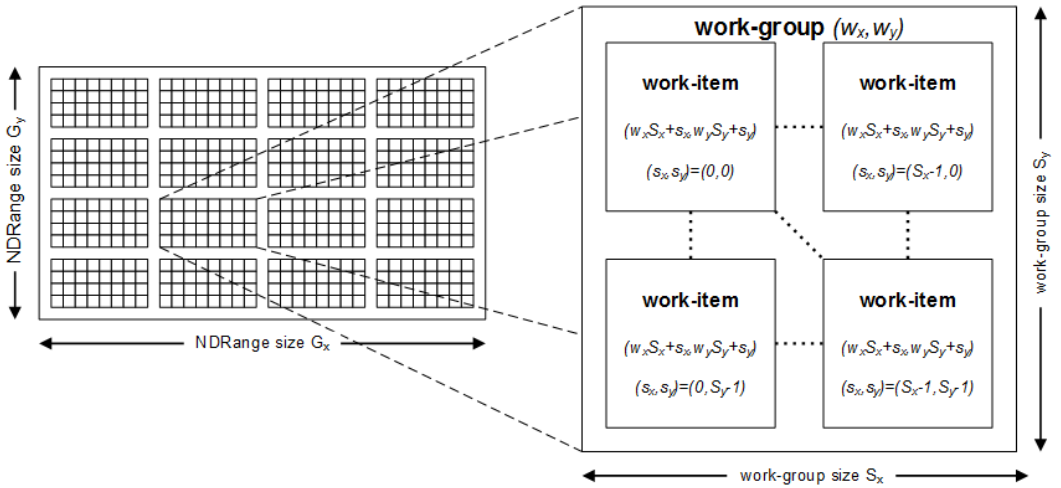
CUDA

Первоначальная версия CUDA SDK была представлена 15 февраля 2007 года.

В основе интерфейса лежит язык C с некоторыми расширениями.

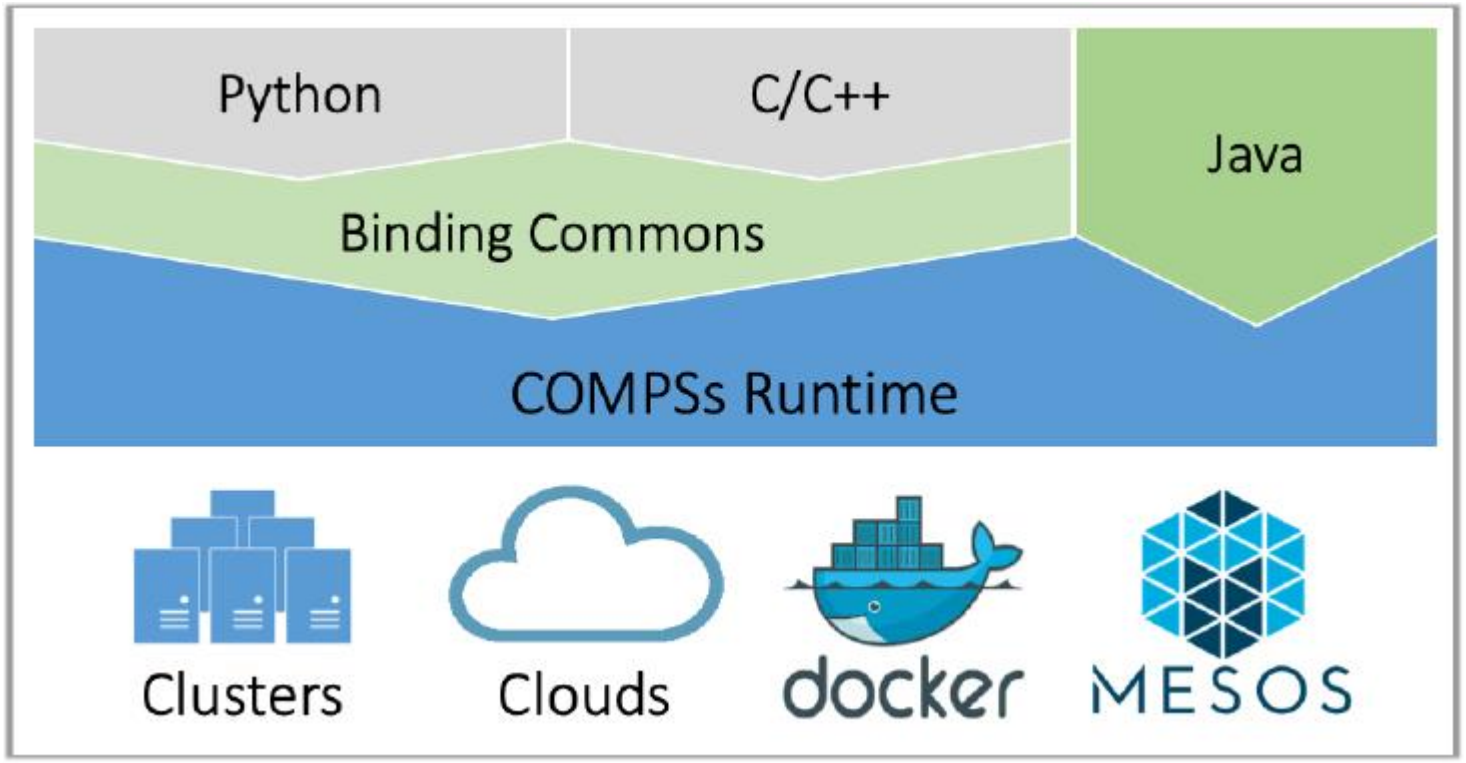
OpenCL

Первоначально был разработан в компании Apple Inc. Вскоре компания AMD решила поддержать разработку OpenCL. 16 июня 2008 года была образована рабочая группа разработки спецификаций OpenCL.



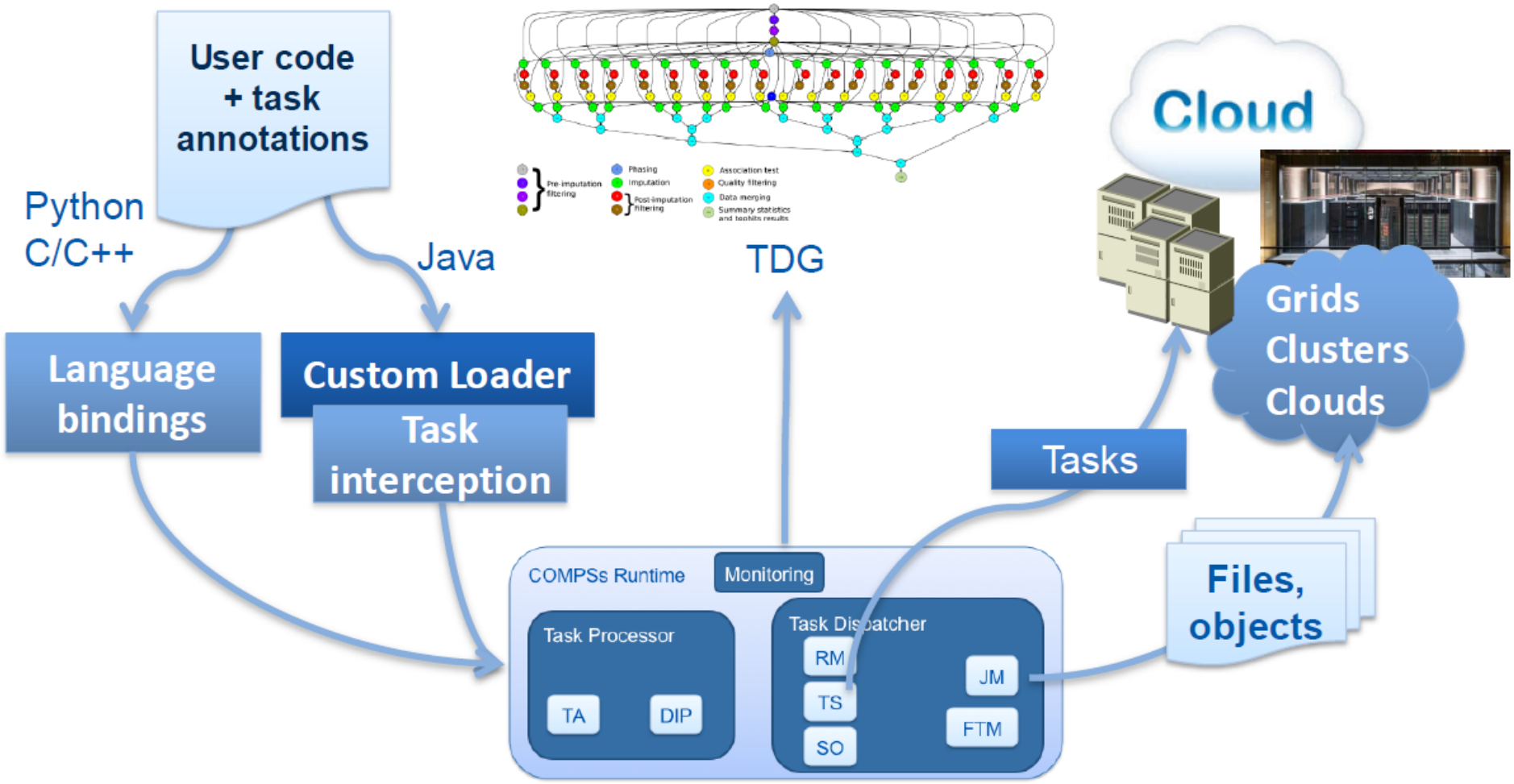
Основные идеи COMPSs

COMPSs поддерживает такие языки программирования как Java, Python и C/C ++. Модель программирования COMPS построена на использовании Runtime system, которая управляет несколькими аспектами выполнение заявок



Основные идеи COMPSs

Runtime system



Пример использования COMPSs на языке Python

Listing 1:

```
a=2.2
b=3.3
y=a*b+a/b
print(y)
>python ab.py
7.92666666667
```

Listing 2:

```
def mymul(x,y): return x*y
def myadd(x,y): return x+y
def mydiv(x,y): return x/y
a=2.2
b=3.3
z=myadd(mymul(a,b),mydiv(a,b))
print(z)
>python abf.py
7.92666666667
```

Код программы в системе COMPSs

Listing 3:

```
from pycompss.api.task import task                    # Import @task decorator
from pycompss.api.parameter import *                  # Import parameter metadata for
the @task decorator
from pycompss.api.api import compss_wait_on  # Import synnchronization function
@task(x=IN, y=IN, returns=float)
def mymul(x, y):
    return x * y
@task(x=IN, y=IN, returns=float)
def myadd(x, y):
    return x + y
@task(x=IN, y=IN, returns=float)
def mydiv(x, y):
    return x/y
if __name__ == '__main__':
    a = 2.2
    b = 3.3
    z = myadd(mymul(a, b), mydiv(a, b))
    z = compss_wait_on(z)
    print(z)
```

Результат

[INFO] Using default execution type: compss

[INFO] Using default location for project file: /opt/COMPSs/Runtime/configuration/xml/projects/default_project.xml

[INFO] Using default location for resources file: /opt/COMPSs/Runtime/configuration/xml/resources/default_resources.xml

[INFO] Inferred PYTHON language

---------- Executing abfc.py ----------

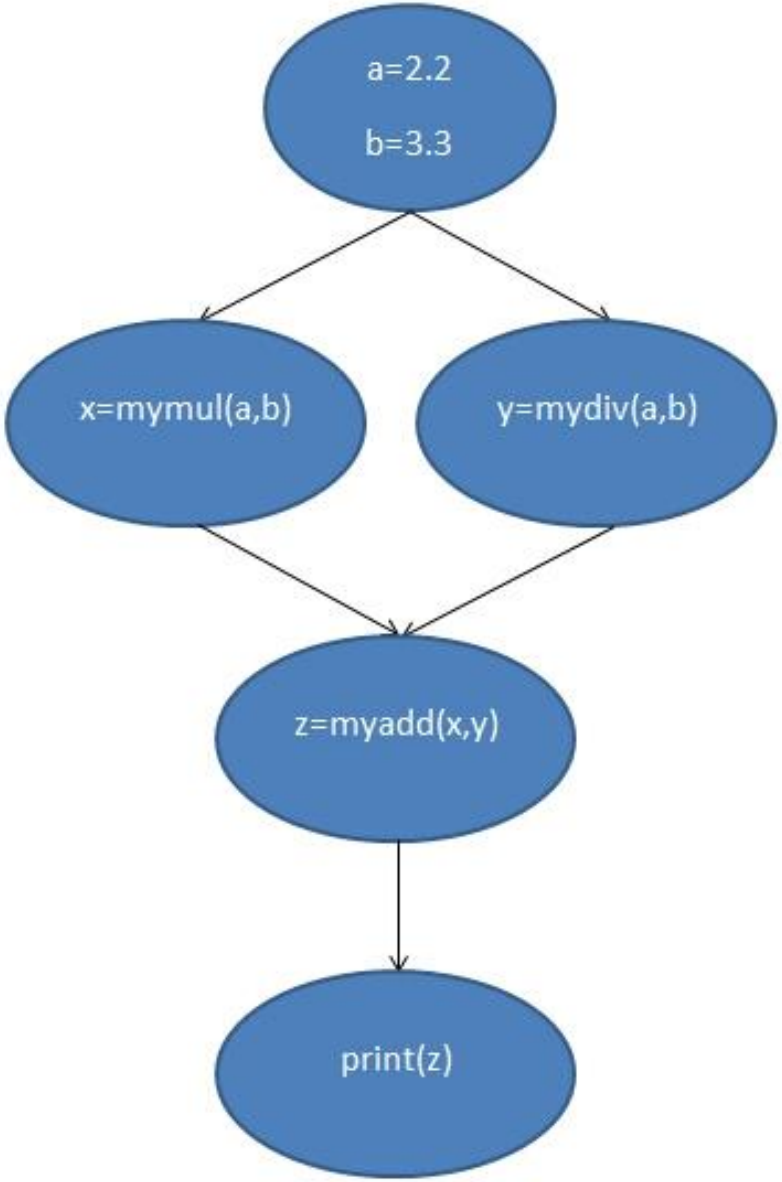
WARNING: COMPSs Properties file is null. Setting default values

[(660)   API] - Starting COMPSs Runtime v2.5.rc1907 (build 20190702-1710.rb9d4035fbe39f8f3d692d485d359964971080cc0)

7.92666666667

[(4601)   API] - Execution Finished

Граф информационных связей



Средства системы COMPSs

Декораторы	Дополнительные описания входных и выходных параметров функций программы
Библиотеки	Набор функций явного вызова COMPSs посредством API, например для синхронизации
Файл ресурсов	Описание общей организации вычислительного кластера, процессоров, памяти, ядер вычислительных узлов и топологии соединений
Файл проекта	Список и взаимодействие основных модулей пользователя а также использованных средств COMPSs

Декораторы

```
@task (f=FILE_INOUT)
def func(f, i):
        fd = open(f, 'r+')


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

@task(is distributed =True)
        def func():
        ...
```

Argument	Value
Formal parameter name	**(default: empty)**: The parameter is an object or a simple tipe that will be inferred. - IN: read-only parameter, all types. - INOUT: read-write parameter, all types except file (primitives, strings, objects). - OUT: write-only parameter, all types except file (primitives, strings, objects). - CONCURRENT : concurrent read-write parameter, all types except file (primitives, strings, objects). - FILE/FILE_IN: read-only file parameter. - FILE_INOUT: read-write file parameter. - FILE_OUT: write-only file parameter. - FILE_CONCURRENT: concurrent read-write file parameter. - COLLECTION_IN: read-only collection parameter (list). - COLLECTION_INOUT: read-write collection parameter (list). - Dictionary: {Type:(empty=object)/FILE/COLLECTION, Direction:(empty=IN)/IN/INOUT/OUT/CONCURRENT}
returns	int (for integer and boolean), long, float, str, dict, list, tuple, user-defined classes
target_direction	INOUT (default), IN or CONCURRENT
priority	True or False (default)
is_distributed	True or False (default)
is_replicated	True or False (default)
on_failure	'RETRY' (default), 'CANCEL_SUCCESSORS', 'FAIL' or 'IGNORE'

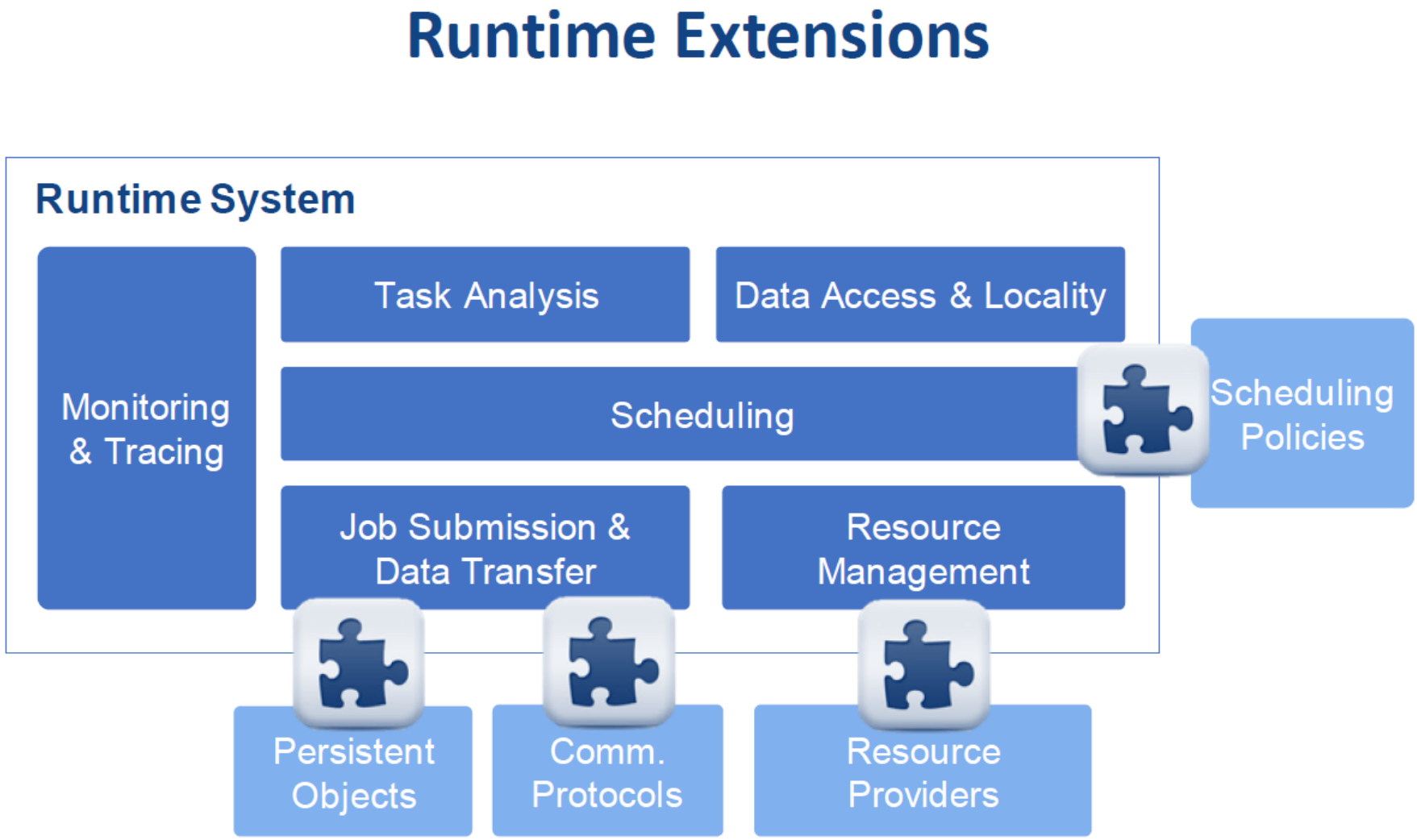
Table 1: Arguments of the @*task* decorator.

Декораторы

Decorator	Parameters
@binary	**binary**: (Mandatory) String defining the full path of the binary that must be executed. **working_dir**: Full path of the binary working directory inside the COMPSs Worker.
@ompss	**binary**: (Mandatory) String defining the full path of the binary that must be executed. **working_dir**: Full path of the binary working directory inside the COMPSs Worker.
@mpi	**binary**: (Mandatory) String defining the full path of the binary that must be executed. **working_dir**: Full path of the binary working directory inside the COMPSs Worker. **runner**: (Mandatory) String defining the MPI runner command. **computing_nodes**: Integer defining the number of computing nodes reserved for the MPI execution (only a single node is reserved by default).
@compss	**runcompss**: (Mandatory) String defining the full path of the runcompss binary that must be executed. **flags**: String defining the flags needed for the runcompss execution. **app_name**: (Mandatory) String defining the application that must be executed. **computing_nodes**: Integer defining the number of computing nodes reserved for the COMPSs execution (only a single node is reserved by default).
@multinode	**computing_nodes**: Integer defining the number of computing nodes reserved for the task execution (only a single node is reserved by default).

Table 2: @binary, @ompss, @mpi, @compss and @multinode decorators supported parameters.

Расширения системы выполнения



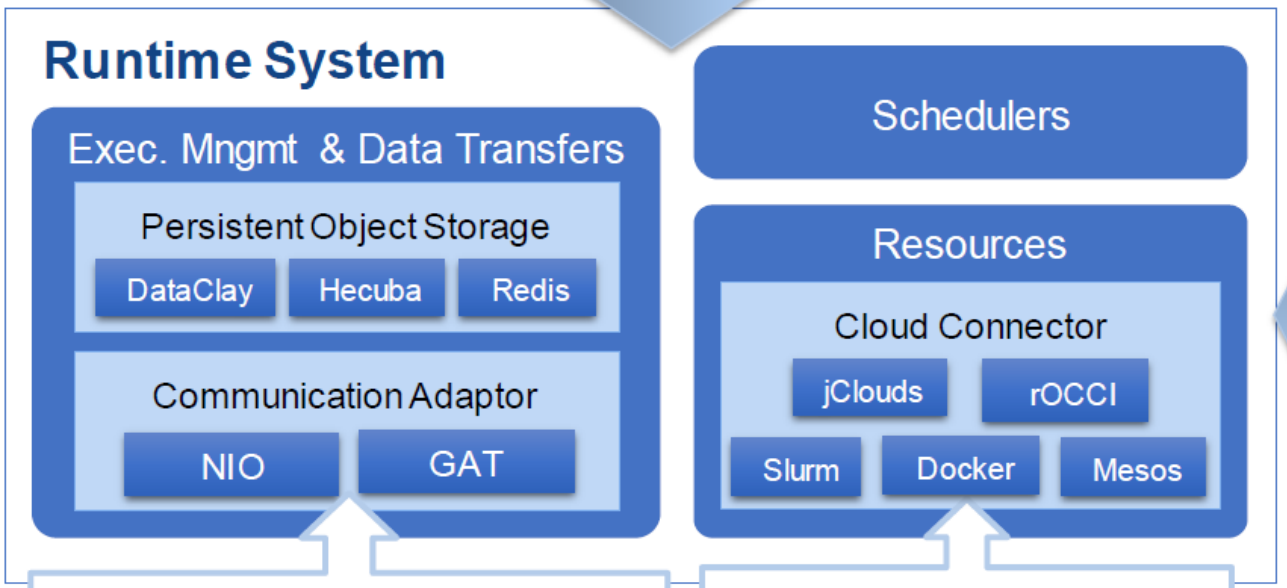
Конфигурация настроек выполнения

Execution Environments Configuration

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

runcompss options

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



resources.xml

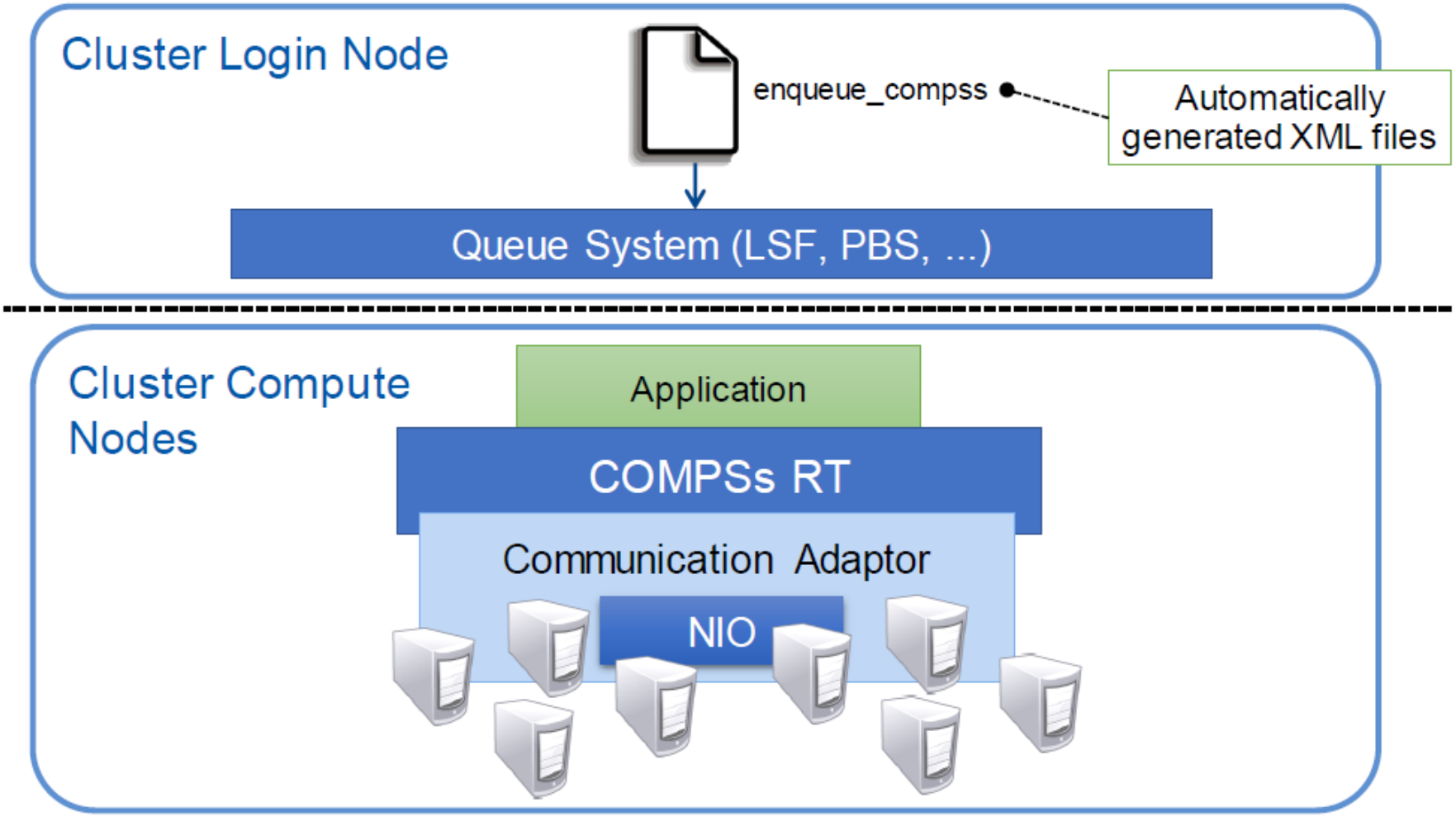
project.xml

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

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

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

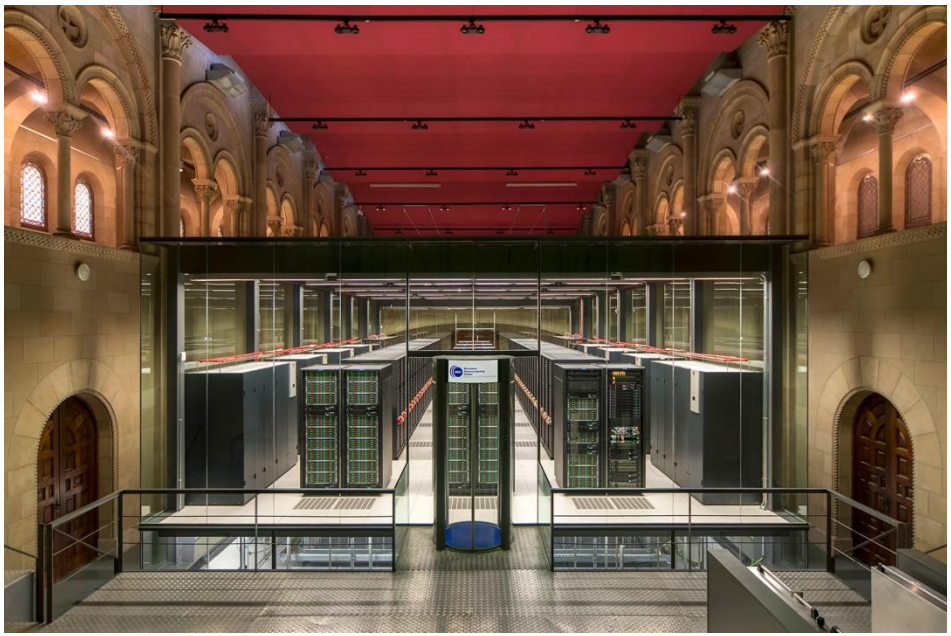
Схема системы COMPSs для кластеров



BSC

Barcelona Supercomputing Center
Centro Nacional de Supercomputación

Mare Nostrum (лат. – Средиземное море) – это символичное название для самого мощного суперкомпьютера в Испании который находится в BSC



Сертификат

PRACE

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

PATC@BSC Training Course

Certificado de Asistencia
Certificate of Attendance

PATC Course: Programming Distributed Computing Platforms with COMPSs

Dirección / Address: PATC at BSC - CNS
C/ Jordi Girona, 29
Edifici Nexus II
E-08034 Barcelona
País / *Country*: Spain

Certifica que / *hereby certifies that*:

El/la investigador/a / *the researcher Nikolay Zaitsev* **procedente de** / *coming from Richelieu Lyceum, Odessa,* **ha asistido a dicho evento en la siguiente fecha**/ *has attended this event on the following dates:*

Días / *day*	**Mes** / *month*	**Año** / *year*
29-30	*01*	*2019*

PATC Directora / *PATC Director*	*Maria-Ribera Sancho*

Firma / *signature:*

Sello / *Stamp:*

BSC Barcelona Supercomputing Center

Fecha / *date*: *11-02-2019*

Este documento no será válido si lleva cualquier tachadura o enmienda / *This document will not be valid if crossed out or corrected*

Спасибо за внимание!