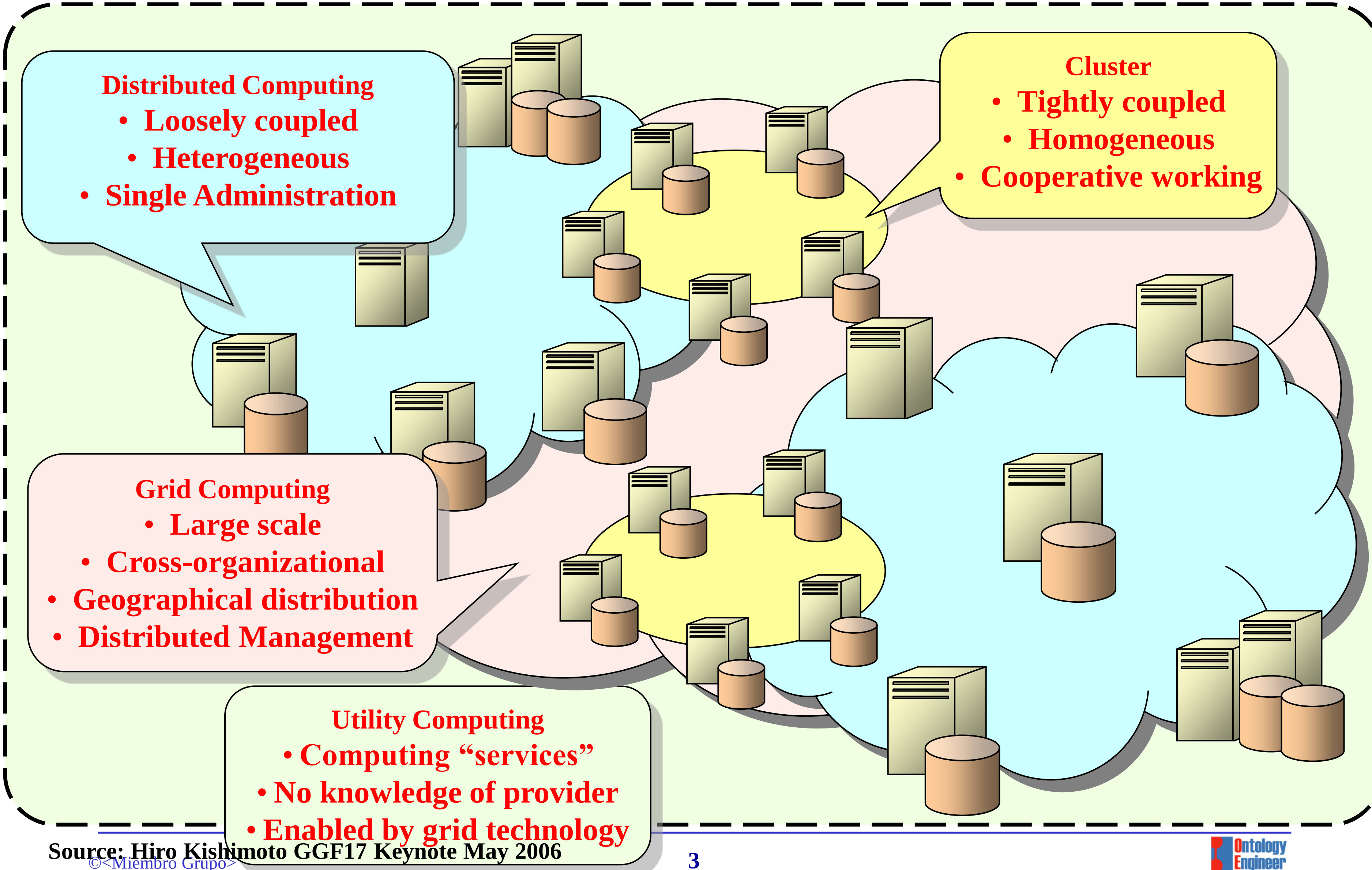
International Summer School on Grid Computing 2008

Carlos Buil Aranda

Table of Contents

Introduction - Grid & Related Paradigms



Source: Hiro Kishimoto GGF17 Keynote May 2006

Introduction – The Grid

- What does the Grid provide?
  - Combine different and heterogeneous systems
    - Heterogeneity of data, systems, instruments, research processes, etc.
  - Grids provide virtual communities (virtual homogeneity)
  - Component for e-Infrastructure
- Why use/build Grids?
  - New Research: enabling new ways of doing research in many disciplines
  - Economic reasons
  - Computer science reasons (distributed systems problems)

Introduction - eScience

- What is e-Science?
  - E-Science is the application of the Grid infrastructure in order to obtain a better, faster or new research result in ALL research disciplines
- What is e-Infrastructure?
  - "e-Infrastructure is the term used for the distributed computing infrastructure that provides shared access to large data collections, advanced ICT tools for data analysis, large-scale computing resources and high performance visualisation"
  - Not only Grids but also networks, data centre and specially enabling collaboration techniques.
- Collaboration

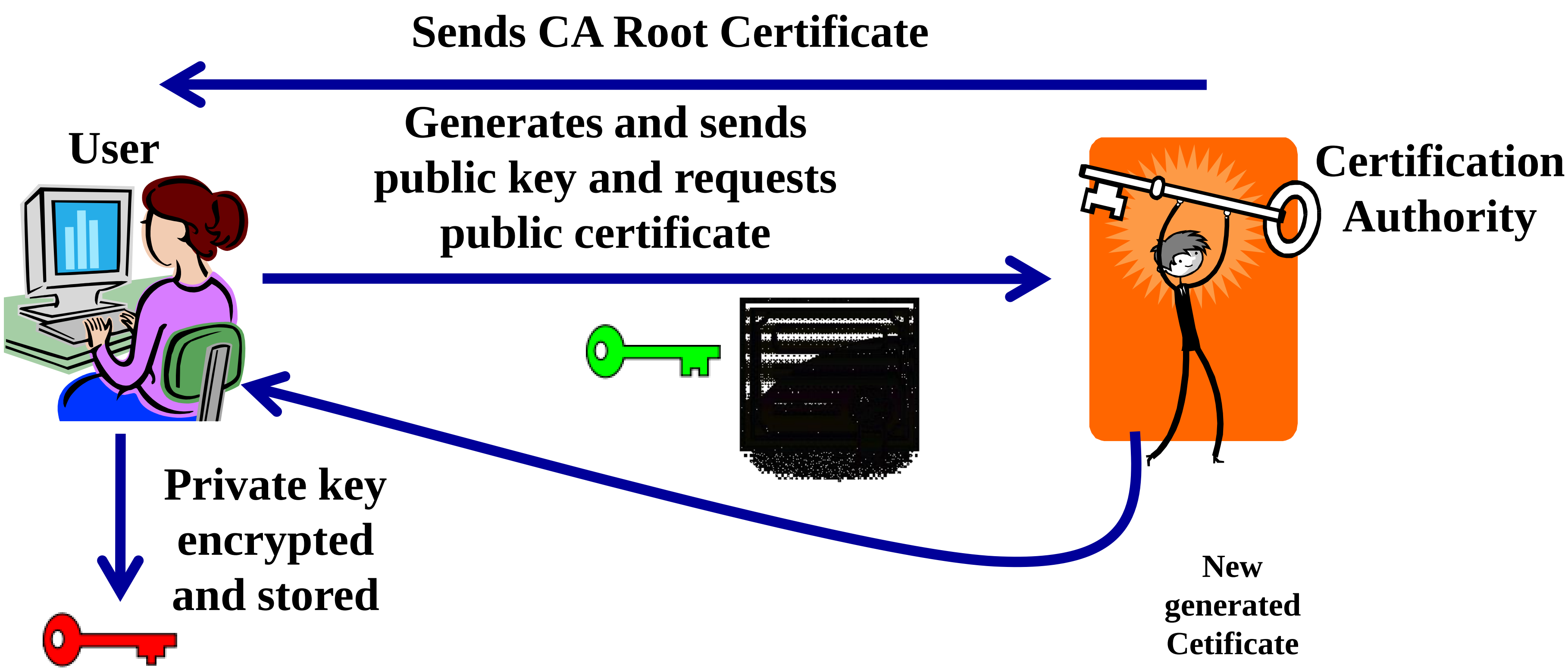
Table of Contents

Grid Security

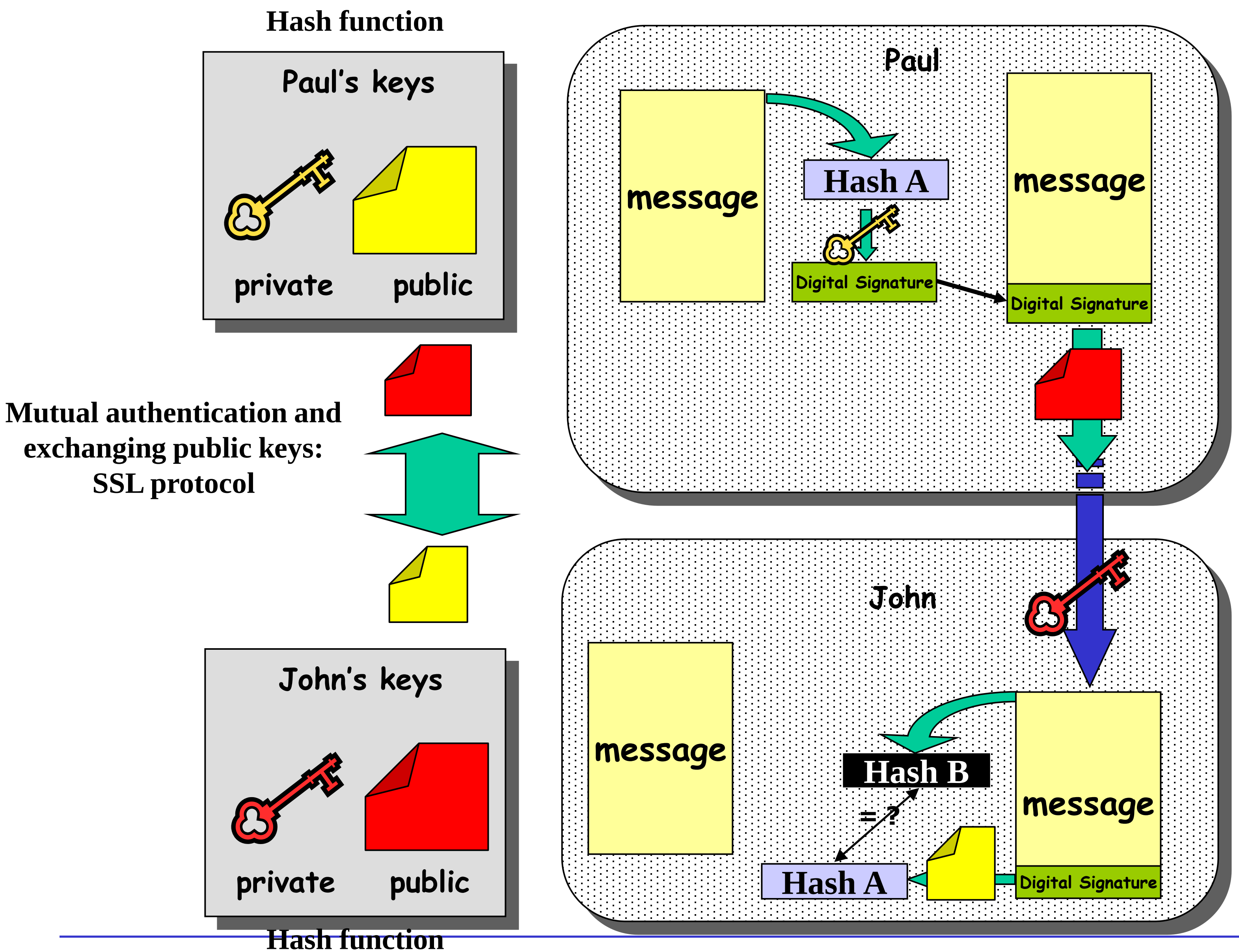
- Different clusters access to other clusters information
- Why should the OEG cluster trust the cluster UPC in Barcelona? Who do I know they are who they say they are?
  - Sensitive information must be protected
  - The cluster resources must be protected
- Communication happens through internet
  - Internet is not the most secure channel
- Two levels of security
  - Network level security (Authentication)
  - VO level security (Authorization)

Grid Security Infrastructure (GSI)

- Security at network level: Public Key Infrastructure (PKI)
- Based on pair of keys: Public and Private keys



PKI in action – the big picture



Grid Security

- VO Security
  - To what resources can user X access?
  - Authorization problem
    - Keeping a database on every site
    - Keeping a list centrally
- Delegation of access rights
  - Proxy certificates

Distributed Systems Principles

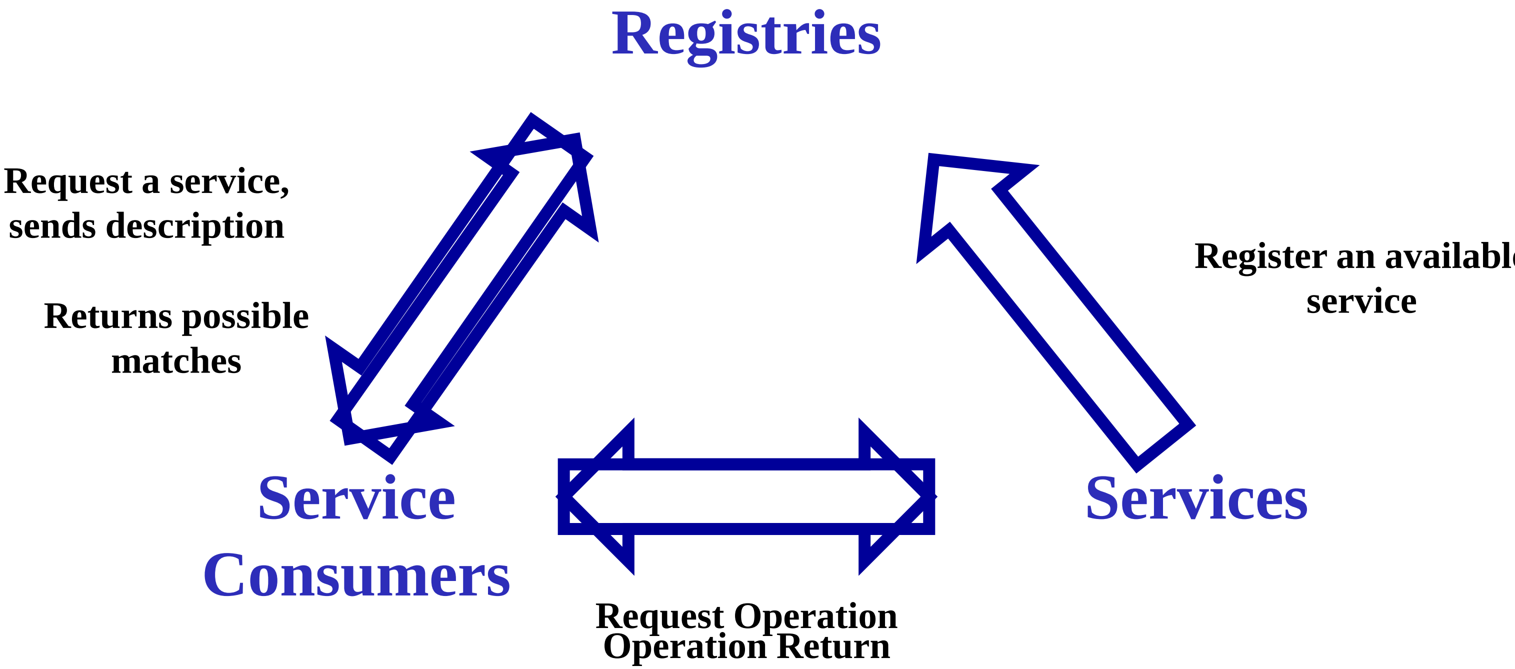
P.H. Enslow, *"What is a Distributed Data Processing System?"* Computer, January 1978

- High Availability and Reliability
- High System Performance
- Ease of Modular and Incremental Growth
- Automatic Load and Resource Sharing
- Good Response to Temporary Overloads
- Easy Expansion in Capacity and/or Function

Distributed Systems Principles (problems)

- Distributed systems problems
  - Lack of Knowledge
    - Due to technical reasons (latency, failures, tec.), human reasons (incomplete knowledge, lack of understanding, poor models)
  - Heterogeneity
    - Hardware heterogeneity (computer architectures, storage systems, input instruments), OS heterogeneity, different implementations systems
  - Latency
    - It always be there (and probably it will get worse due to geographic scale, system complexity, processing time)
  - Unreliability
    - Failures can happen (Network outages, Power outages, software errors, etc.)
  - Autonomy & Change
    - Changes on local systems, services

Distributed Systems Principles (Service Oriented Architecture)



Computational Resources & Job management

- Multiprocessor systems
  - Shared memory
  - Distributed memory
  - Distributed shared memory
- Usage
  - submission of jobs by the users
  - Processing of these jobs
  - High amount of jobs, resources are not so large
- Ways for managing submission of jobs is required
  - Time sharing
  - Space sharing

Job submission

- High heterogeneity of computational resources:
  - Different types of machines, OS, requirements for submission, memory, cores, duration of the tasks, etc.
- Abstraction is needed for submitting jobs
  - JSDL: Job Submission Description Language
  - OGF standard
  - Every system has its own submission language
- JSDL
  - XML based
  - Defines the job requirements (type of processor, OS, memory, etc.) and parameters (inputs, outputs, etc.)
  - It is not possible to submit jobs directly to resources

JSDL Example

```xml
<jsdl:JobDefinition xmlns="http://www.example.org/"
    xmlns:jsdl="http://schemas.ggf.org/jsdl/2005/11/jsdl"
    xmlns:jsdl-posix="http://schemas.ggf.org/jsdl/2005/11/jsdl-posix"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <jsdl:JobDescription>
    <jsdl:JobIdentification>
      <jsdl:JobName>Simple Application GW Template vs JSDL</jsdl:JobName>
      <jsdl:Description> This is a simple example to describe the main
            differences between GW Template and the JSDL schema.
      </jsdl:Description>
    </jsdl:JobIdentification>
    <jsdl:Application>
      <jsdl:ApplicationName>ls</jsdl:ApplicationName>
      <jsdl-posix:POSIXApplication>
        <jsdl-posix:Executable>/bin/ls</jsdl-posix:Executable>
        <jsdl-posix:Argument>-la file.txt</jsdl-posix:Argument>
        <jsdl-posix:Environment name="LD_LIBRARY_PATH">/usr/local/lib</jsdl-posix:Environment>
        <jsdl-posix:Input>/dev/null</jsdl-posix:Input>
        <jsdl-posix:Output>stdout.${JOB_ID}</jsdl-posix:Output>
        <jsdl-posix:Error>stderr.${JOB_ID}</jsdl-posix:Error>
      </jsdl-posix:POSIXApplication>
    </jsdl:Application>
    <jsdl:Resources>
      <jsdl:CandidateHost>
        <jsdl:HostName>*.dacya.ucm.es</jsdl:HostName>
      </jsdl:CandidateHost>
      <jsdl:CPUArchitecture>
        <jsdl:CPUArchitectureName>x86_32</jsdl:CPUArchitectureName>
      </jsdl:CPUArchitecture>
    </jsdl:Resources>
```

Execution & Job management

- Initiating/submitting, monitoring and managing jobs
  - Translate job to the target system language, submit the job, get the job status, cache the job if something goes wrong, etc.
- Handling and organizing all the job data
  - Manage the job data (internal directories, intermediate data used by the jobs, etc.). All these data is preserved/organized after the successful job completion.
- Coordinating and scheduling jobs
  - Coordinate workflow resources, manage the initiation of the workflow execution, monitor the workflow, etc.

High Throughput Computing

- Provide to the users a high amount of computational resources
  - Goal: be able to maintain a high number of processes running with high performance and reliability
  - Reliability is a key question in these systems
- High amount of resources
- High availability of these resources
  - The greater the degree of replication of resources, the better the ability of the system to maintain high reliability and performance
- Principal unity of control
  - One unit that leads the goal of all resources working together
- Transparency for the user
  - Remember VOs?
- Component autonomy
  - The resources are independent of allowing or not the use of their capabilities

Distributed Data Management

- Data characteristics
  - Distributed, produced in large quantities, etc.
  - Where the data is used?
    - Scientific applications consume large amounts of data
  - Problems
    - Storage is not a problem anymore
    - Computing power is still a problem
      - But not a big deal, we still produce large amounts of data but storage systems and processing systems are growing
    - New problem: the power needed to analyse/use the data

Principles of Distributed Data Management (I)

- Data Processing, data access, data replication
- Data processing
  - Co-scheduling: moving computation to data
  - Desirable for large amounts of data
  - Problems
    - assure required data near execution host is online
    - second code near data without long wait time
    - Clean up after execution or failure
    - Evtl. licence availability
    - ~70% of all failures in distributed systems are due to failure to properly manage data
  - Complexities
    - some data can not be distributed
    - metadata stores use to be central (information about the execution of a process, for example)

Principles of Distributed Data Management (II)

- Remote Data Access
  - Streaming
    - Direct access to the information: it is processed when it is accessed/created
  - Caching
    - Use local data caches
  - Difficulties
    - Data validity
- Replication
  - Keeping replicas of data in many places in a Grid
    - It is possible to keep data close to the computation system allowing to:
    - Improving throughput and efficiency
    - Reducing latency
    - Both problems previously stated (see previous slide)
  - Replica problems
    - Consistency of replicas
    - Locating the replicas
    - Where do I store replicas? Country problems, legal issues
    - Semantically equivalent but not identical replicas

Principles of Distributed Data Management (III)

- Space Management
  - assure that target has enough space
  - output can be written
- File Transfer
  - Grid ftp
  - Reliable GridFTP
  - System's own implementations
- Trust
  - Site policies: Do not trust the users
    - Who can access what information
  - VO policies: Do not trust the sites
    - Accessing sensitive data
    - Managing user and group at VO level
  - An agreement must be achieved

Distributed File Management

- Distributed File Systems
  - Wide Area File Access
  - Storage Resource Manager SRM interface to File Storage
    - Exists OGF Standard
  - Difficulties
    - Scaling issues (servers, clients, updates), secutiry
- Managed Storage Systems
  - Stores data in the order of Petabytes
  - Total-throughput scales with the size of the installation
  - Supports several hundreds to thousands of clients
  - Adding / removing storage nodes w/o system interruption

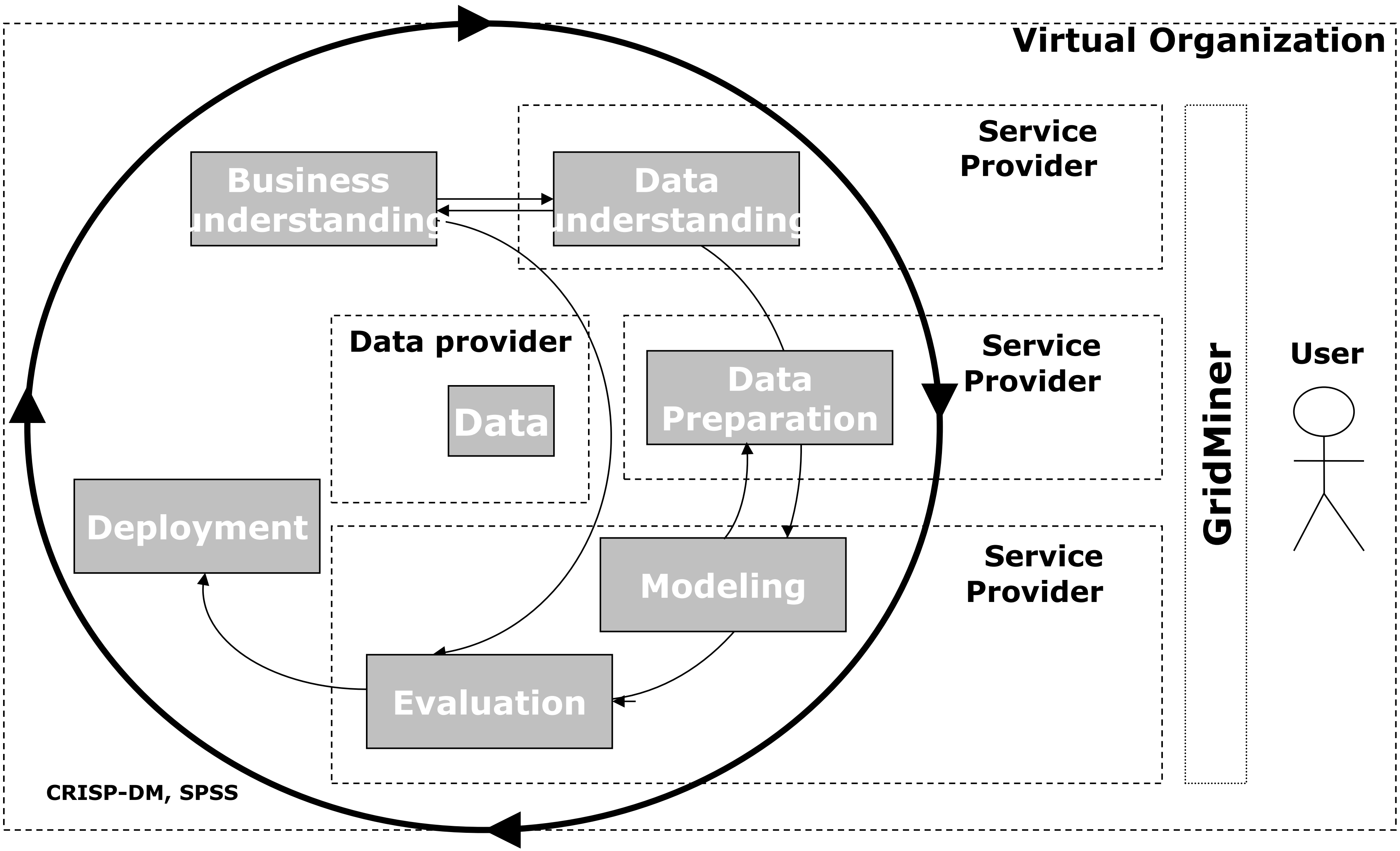
Others

- Google File System
  - MapReduce parallel data programming model
    - Allows for automatic parallelization of data intensive tasks
    - Generate intermediate key/value pairs for a regular key/value
    - Reduce(merge) all generated ones back together
  - GoggleFS filesystem – Open Hadoop implementation
- Hadoop
  - NOT for Wide Area yet
  - Built with reliability in mind for commodity hardware
  - Optimized for streaming access, not generic posix
  - Built in java for large files
  - Write once read many patterns best
  - Very new, changing fast
  - Watch out for scaling

Distributed Data Mining

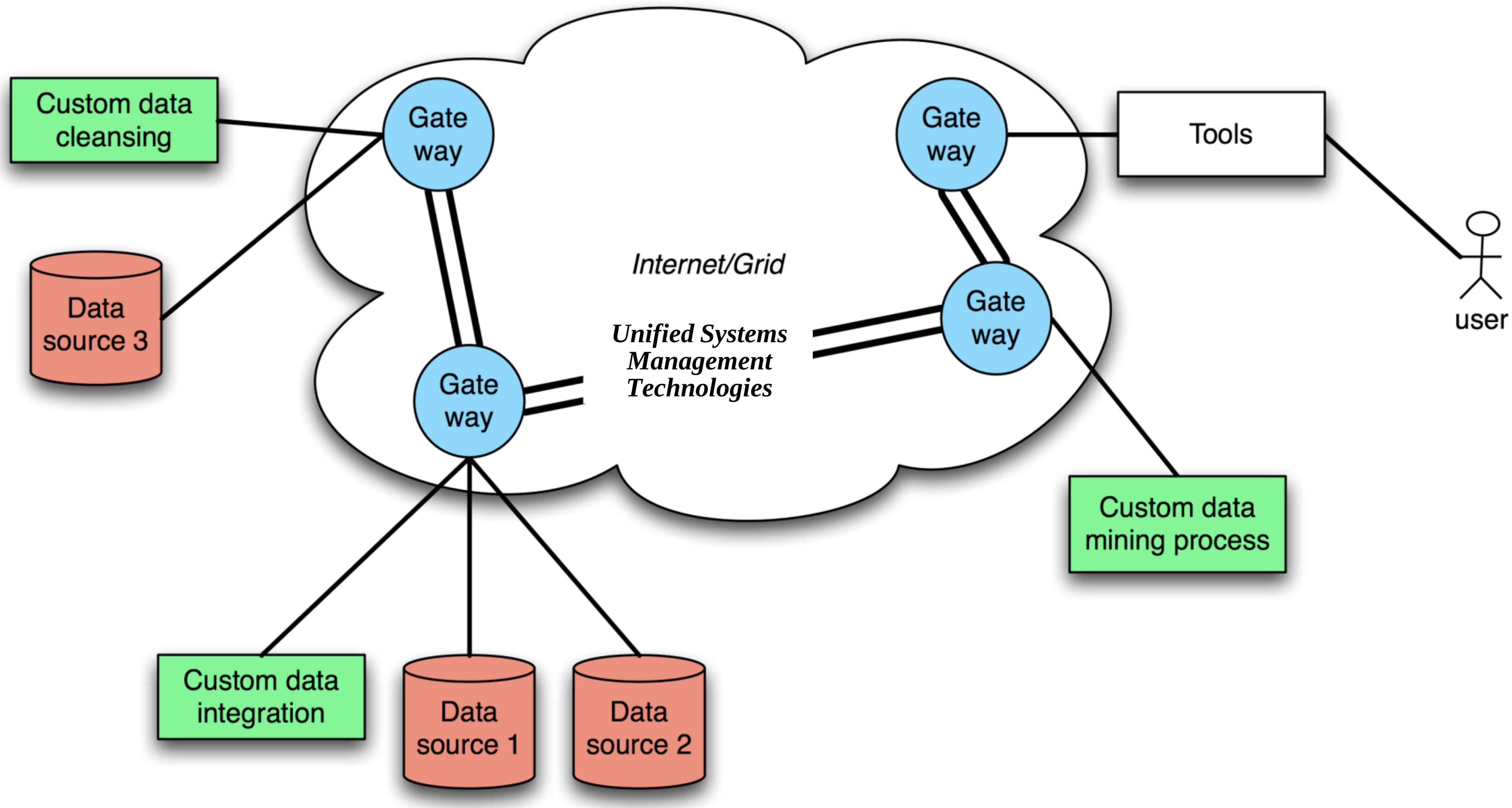
- Large amounts of distributed data
- It is possible to extract useful information from the data generated
- Possibility of distributing data mining processes
  - Adaptation of data mining processes to a distributed environment
- ADMIRE
  - Advanced Data Mining and Integration research for Europe
  - Accelerate access to and increase the benefits from data exploitation;
  - Deliver consistent and easy to use technology for extracting information and knowledge;
  - Cope with complexity, distribution, change and heterogeneity of services, data, and processes, through abstract view of data mining and integration; and
  - Provide power to users and developers of data mining and integration processes.

GridMiner Data Mining Model



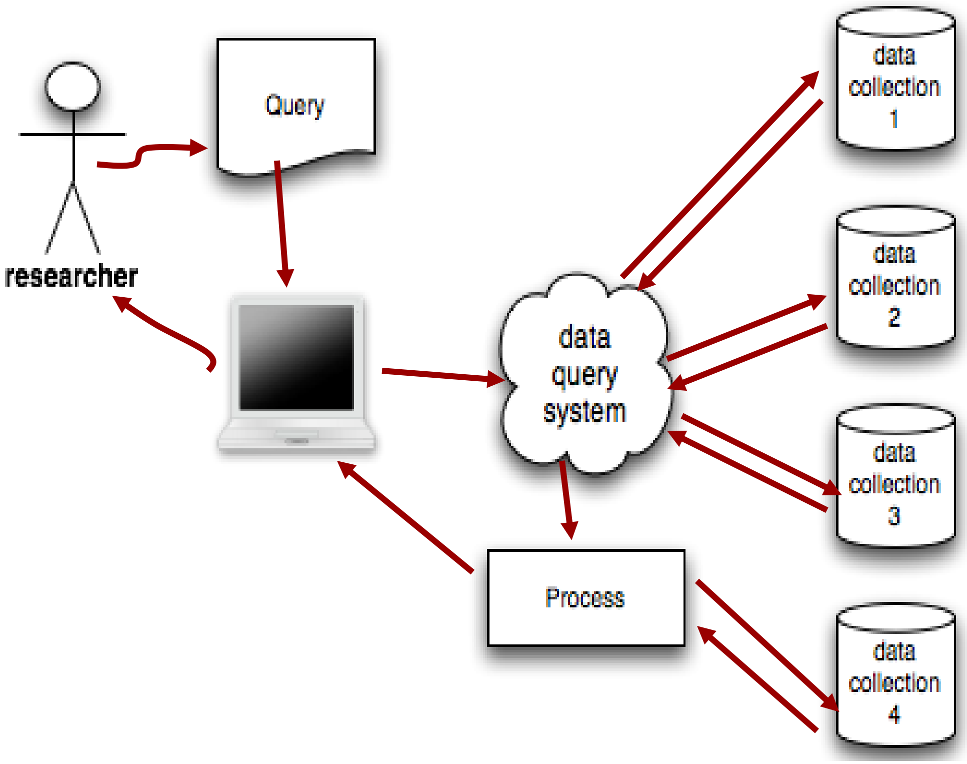
Slide by Peter Brezany – ISSGC08 Balatonfüred, Hungary

ADMIRE’s High-Level Architecture



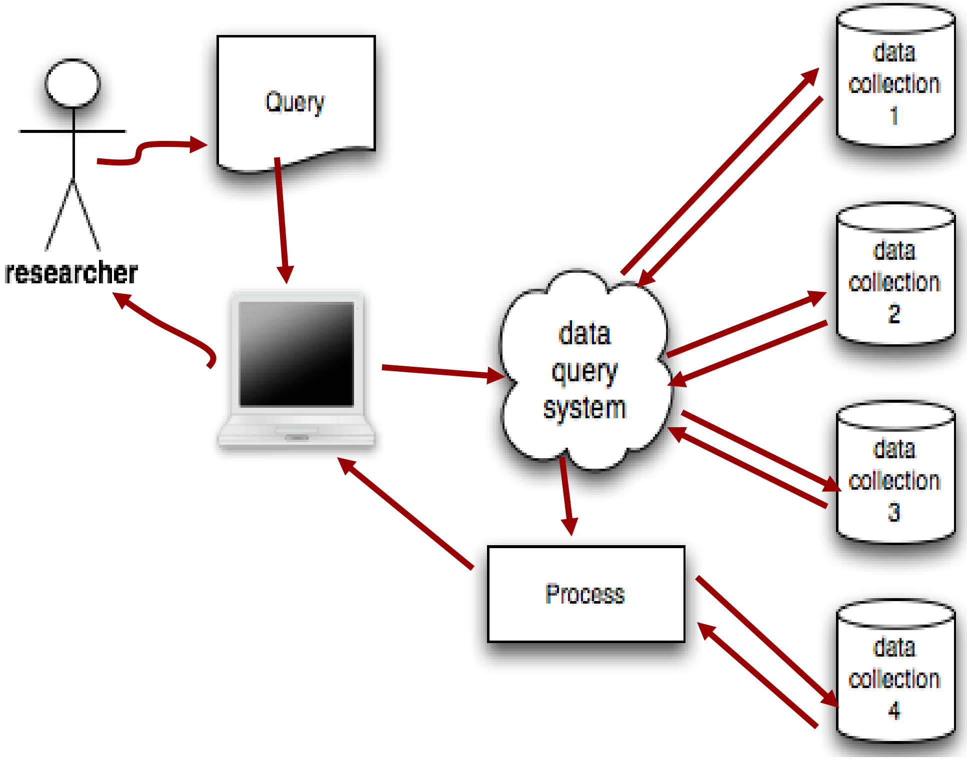
Data Access and Integration

- Gather data from different and distributed sources
- Scientists/researchers make queries to different data sources
- DAI system distributes the queries
- DAI system integrates and process the results to form the requested data
- The data is processed as the user demanded



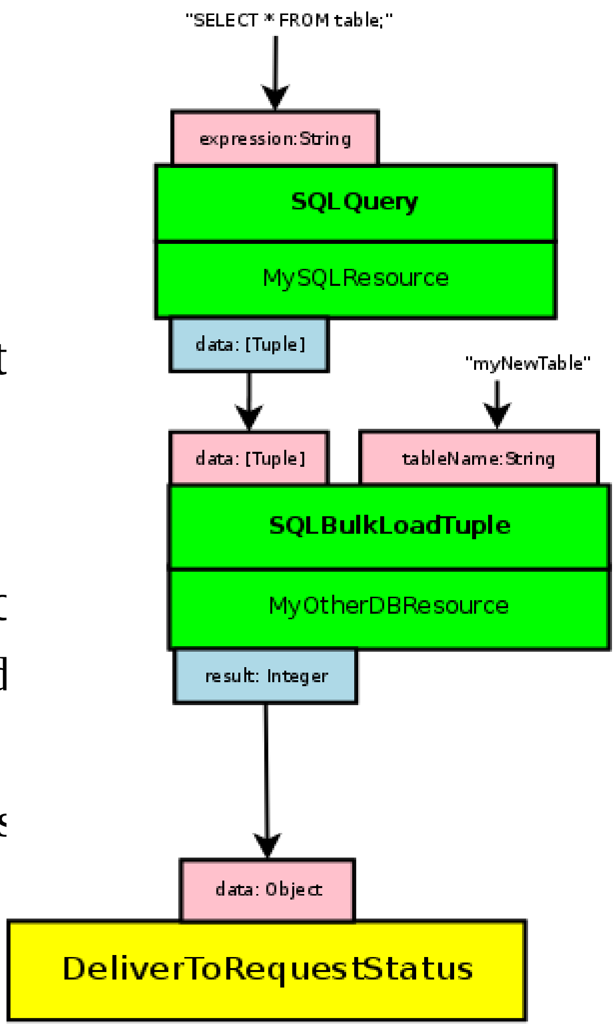
Data Access and Integration

- Gather data from different and distributed sources
- Scientists/researchers make queries to different data sources
- DAI system distributes the queries
- DAI system integrates and process the results to form the requested data
- The data is processed as the user demanded



OGSA-DAI

- OGSA-DAI is
  - An extensible framework that allows to
  - Access, integrate, transform and deliver
  - Distributed and heterogeneous sources of dat
- OGSA-DAI key elements are
  - Resources (Data request execution resource,
  - Activities (=operations or named unit of func
    - Activity connectors for connecting inputs and
  - Workflows (=composition of activities)
    - Pipeline workflow (A set of chained activities data flowing between the activities)
    - Sequence workflow
    - Parallel workflow



Desktop Grids

- Resource Donors and Users
  - if U ~ D => generic Grid model
  - if U >> D => service Grid model
  - if U << D => desktop Grid model
- Generic Grids
  - Anyone can donate resources, heterogeneity, run own applications
- Service Grids
  - Push Model
  - Guaranteed service
  - Many users: anyone can use the assigned resources for running their own applications

Desktop Grids (II)

- Desktop Grid model
- Dynamic resource donation
  - Based on desktop computers
  - More donors of resources than consumers
  - Not guaranteed service
  - Pull model
  - Not large number of grid applications
- BOINC (Berkeley Open Infrastructure for Network Computing)
  - A donor (participant) can define the distribution of idle CPU time for different projects

Table of Contents

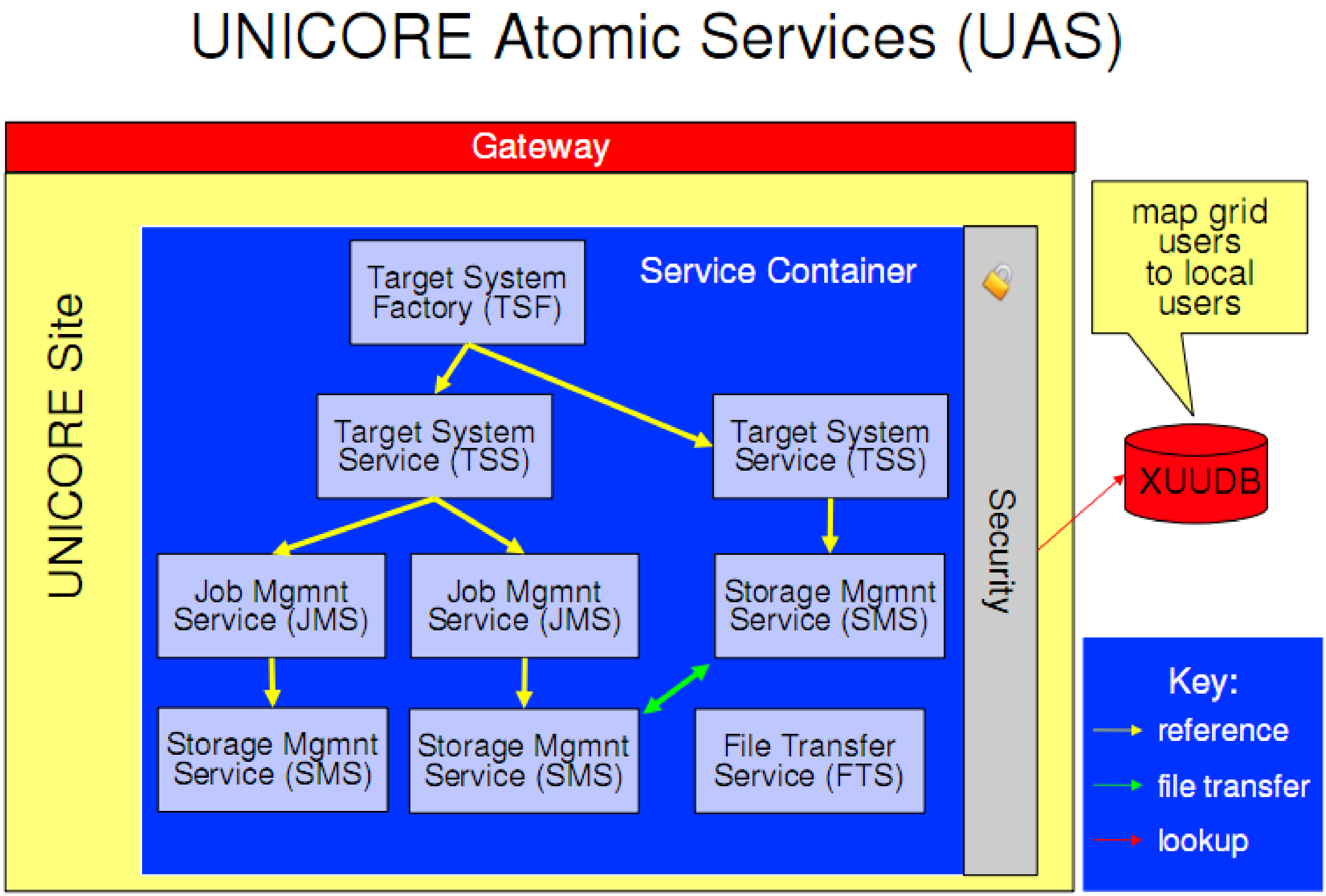
Grid Middleware

- Middleware between the computational resources and the users (computer scientists, researchers)
- The middleware interacts with both
  - User requirements
  - Available resources
- Provides
  - Security infrastructure, job management, data management, job monitoring
- Different implementations for different Grid systems
  - Supercomputers, large clusters, high amount of personal computers,

UNICORE

- Uniform Interface to Computing Resources
- UNICORE is a Grid middleware that provides seamless, secure, and intuitive access to distributed Grid resources
  - Grid resources: supercomputers or clusters systems
- Submitting a job
  - Client establishes a SSL connection with the UNICORE gateway
  - XUUDB checks the user
  - UNICOREX send the job to TSI
- TSI (Target System Interface)
  - Executes the job

UNICORE (II)


UNICORE Atomic Services (UAS)
Gateway
UNICORE Site
Service Container
Target System Factory (TSF)
Target System Service (TSS)
Target System Service (TSS)
Job Mgmnt Service (JMS)
Job Mgmnt Service (JMS)
Storage Mgmnt Service (SMS)
Storage Mgmnt Service (SMS)
Storage Mgmnt Service (SMS)
File Transfer Service (FTS)
Security
map grid users to local users
XUUDB
Key:
reference
file transfer
lookup

System 1 Submit job file

```
{
  Executable: ”/bin/date”,
  Arguments: [”-v”, ”-d”],
  Environment: [”SHELL=/bin/bash”, ”JAVA_OPTS=-v”],
  Imports: [
    { File: myfile, To: uspaceFile },
  ],
  Exports: [
    { File: uspaceOutFile, To: ”localName” },
    { File: otherUspaceOutFile,  To: ”otherLocalName”},
  ],
  Stage in: [
            {
              From: ”RBYTEIO:http://localhost:8080/XNJS/etcetc”,
              To: ”uspaceFileName”
            }
  ],
   Stage out: [
            {
              From: stdout,
              To: ”RBYTEIO:http://someserver/someresource”
            }
  ],
  Resources: {
    Memory: 128000000,
    CPUs: 32,
    Nodes: 4,
    Runtime: 3600
  }
User name: fred,
User email: fred@fred.invalid
Name: my test job,
Description: a sample job,
}
```

Condor

- Grid middleware for HTC systems
- Installed in a large number of computer
- Key concepts:
  - Matchmaker
    - Get an agreement between users of resources and providers of resources (machines)
    - Machine requirements: only jobs from OEG computers, only work from 6pm to 4am
    - Users requirements: to be executed only on Linux machines, dual core processor
    - ClassAd file represents the state of jobs and machines
  - Submitting a job
    - Selecting the job universe (vanilla, Java, standard, MPI, etc.)
    - Using a job description file, possibility of monitoring jobs
  - Job dependencies (DAGMan)
- High importance to the system reliability
  - Job checkpoints, parameter sweep management (clusters and processes)

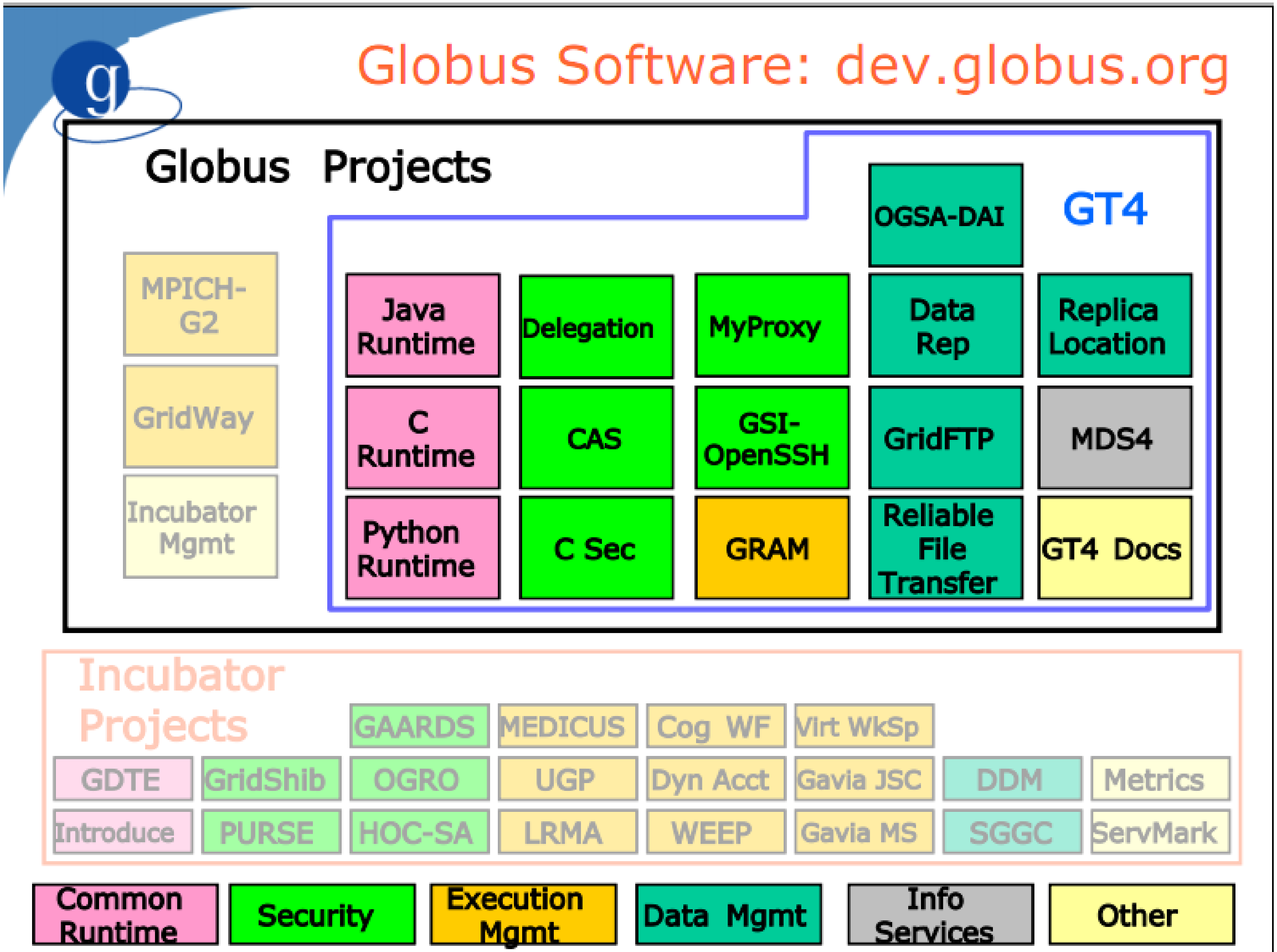
System 2 Submit job file

```
Universe      = vanilla
Executable    = my_job
Log           = my_job.log
InitialDir    = run_$(Process)
ShouldTransferFiles   = IF_NEEDED
Transfer_input_files  = dataset$(Process),
common.data
Requirements = Memory >= 256 && Disk > 10000
Rank = (KFLOPS*10000) + Memory
Queue 600
```

Globus

- Globus is a Grid middleware that provides means for building collaborative and distributed applications
  - It is a middleware that solves the problem of heterogeneity between user applications and computing resources
- Core runtime
- Security (Authentication and authorization)
  - GSI (Grid Security Infrastructure), X509 standard, VOMS (Virtual Organization Membership Service)
- Execution management
  - Supports the GRAM interface (Grid Resource Allocation and Management)
  - It provides a single protocol for communicating with different batch/cluster job schedulers.
- Data management
  - GridFTP (XIO), RFT, Replica Location Service, OGSA-DAI
- Monitoring
  - Monitoring and Discovery System (MDS4): information providers, collective services, clients

GLOBUS (II)

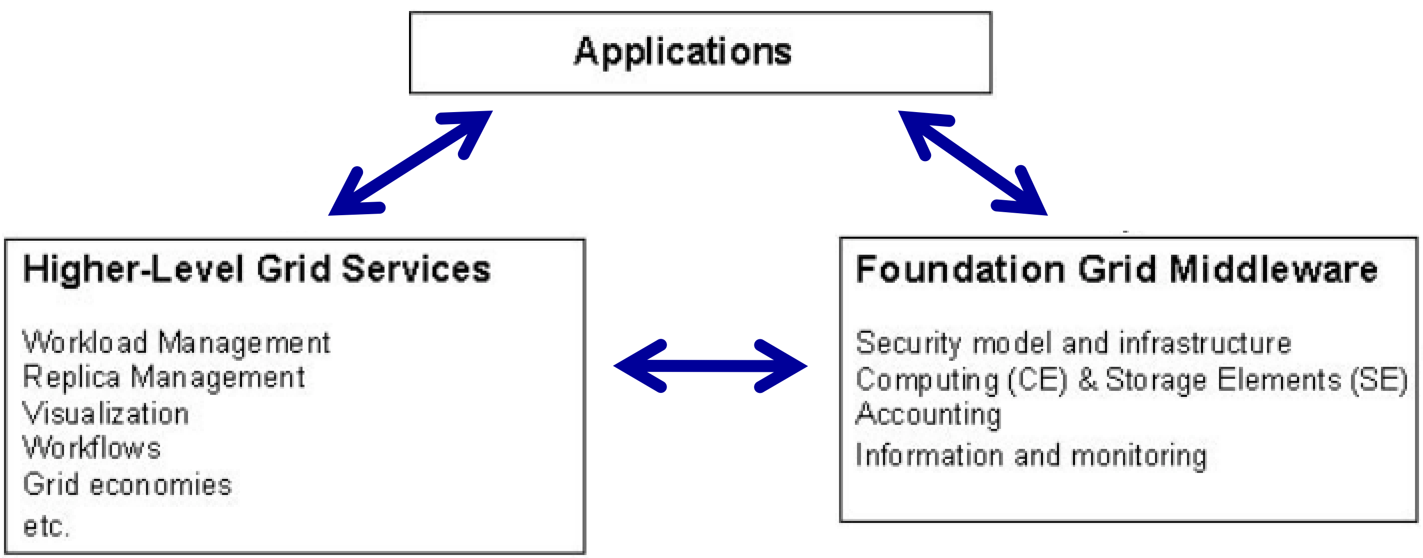

Globus Software: dev.globus.org
Globus Projects
MPICH-G2
GridWay
Incubator Mgmt
GT4
OGSA-DAI
Java Runtime
Delegation
MyProxy
Data Rep
Replica Location
C Runtime
CAS
GSI-OpenSSH
GridFTP
MDS4
Python Runtime
C Sec
GRAM
Reliable File Transfer
GT4 Docs
Incubator Projects
GAARDS
MEDICUS
Cog WF
Virt WkSp
GDTE
GridShib
OGRO
UGP
Dyn Acct
Gavia JSC
DDM
Metrics
Introduce
PURSE
HOC-SA
LRMA
WEEP
Gavia MS
SGGC
ServMark
Common Runtime
Security
Execution Mgmt
Data Mgmt
Info Services
Other

System 3 Submit job file

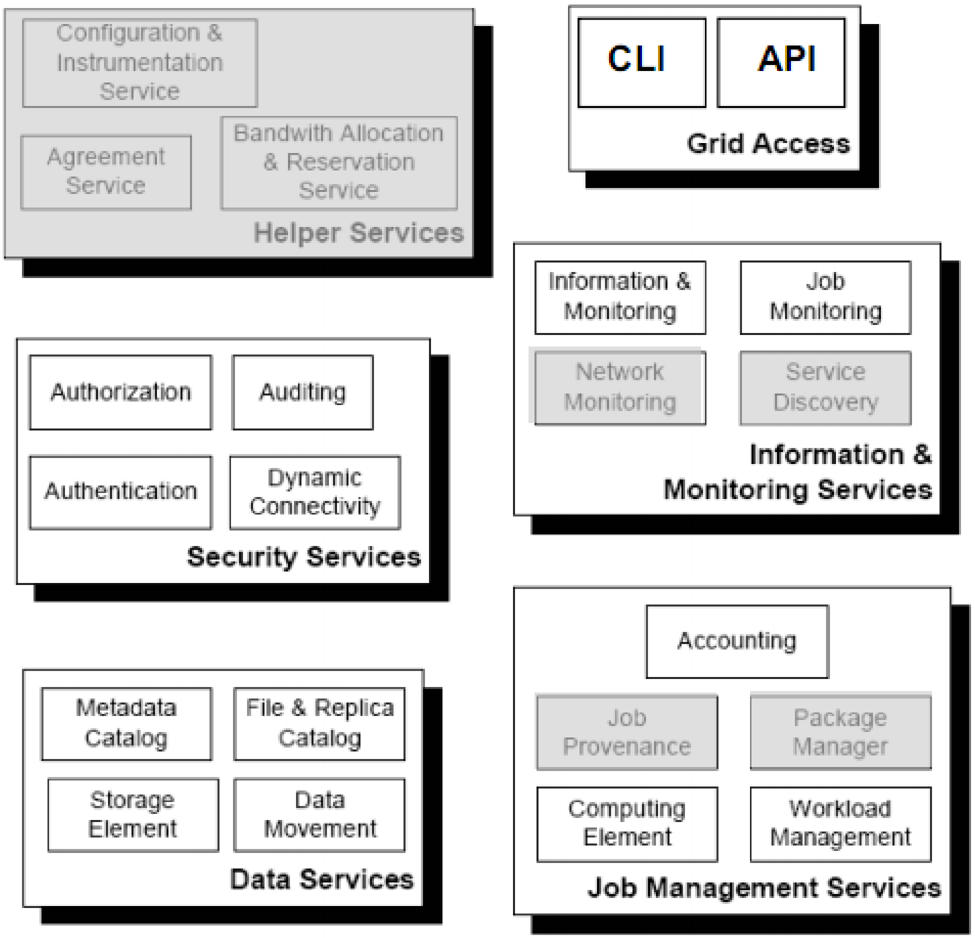
```
<job>
   <executable>${GLOBUS_USER_HOME}/genr</executable>
   <argument>5</argument>
   <argument>4</argument>
   <stdout>${GLOBUS_USER_HOME}/genr.out.${GLOBUS_USER_NAME}</stdout>
   <fileStageIn>
      <transfer>
         <sourceUrl>gsiftp://tc02.nesc.ed.ac.uk:2812/tmp/my_genr</sourceUrl>
         <destinationUrl>file:///${GLOBUS_USER_HOME}/genr</destinationUrl>
      </transfer>
   </fileStageIn>
   <fileStageOut>
      <transfer>
         <sourceUrl>file:///${GLOBUS_USER_HOME}/genr.out.${GLOBUS_USER_NAME}</sourceUrl>
         <destinationUrl>gsiftp://tc02.nesc.ed.ac.uk:2812/tmp/</destinationUrl>
      </transfer>
   </fileStageOut>
   <fileCleanUp>
      <deletion>
         <file>file:///${GLOBUS_USER_HOME}/genr</file>
      </deletion>
      <deletion>
         <file>file:///${GLOBUS_USER_HOME}/genr.out.${GLOBUS_USER_NAME}</file>
      </deletion>
   </fileCleanUp>
</job>
```

gLite

- Is a Grid middleware powering the EGEE infrastructure
  - EGEE is the largest interdisciplinary Grid infrastructure in the world
  - Provides the necessary components to process the users jobs (security, workflow management, metadata management, etc.
  - Middleware Structure:



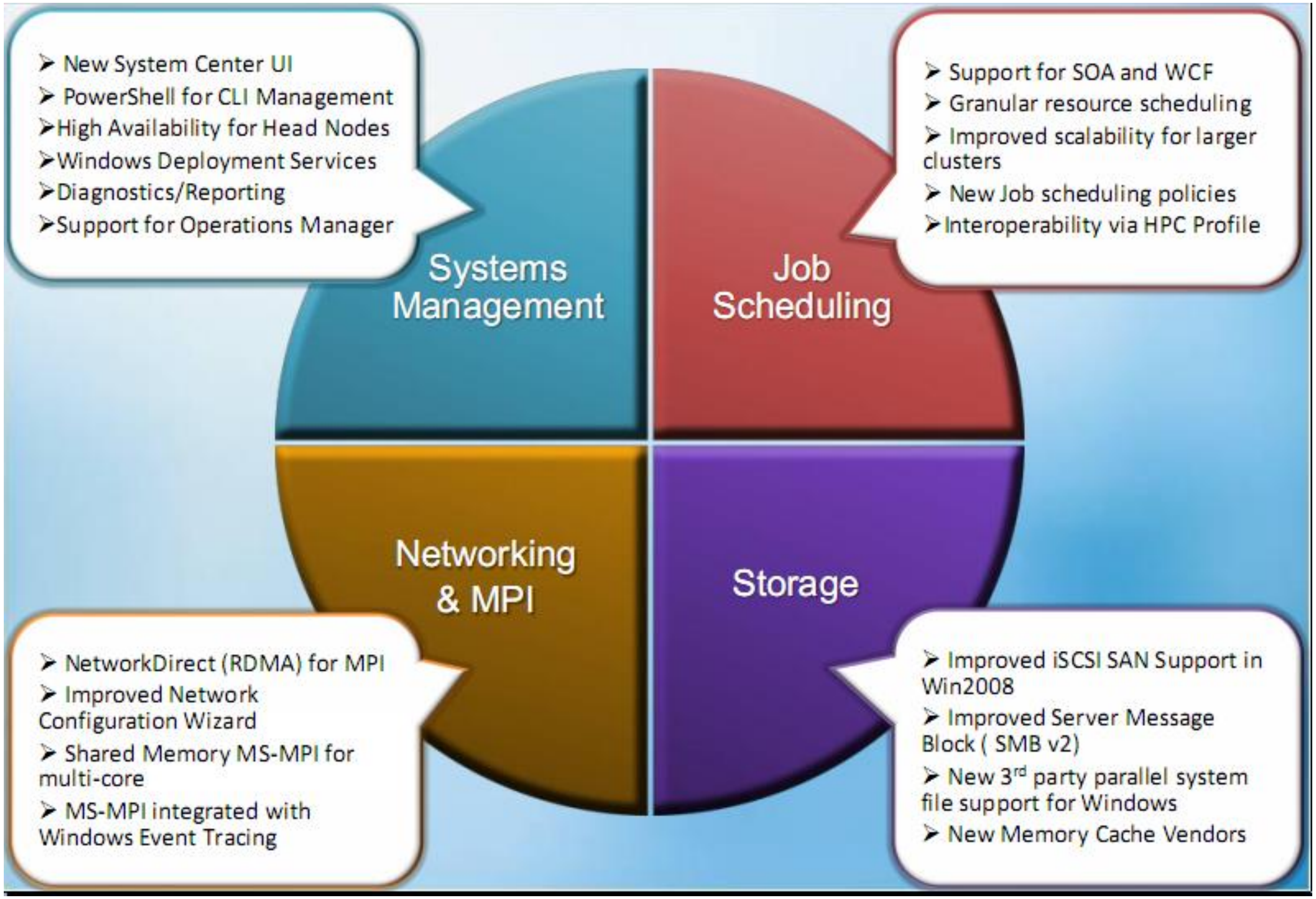
gLite Services Decomposition



System 4 Submit job file

```
Type = "Job";
JobType = "Normal";
Executable = "startGen4.sh";
Environment = {"CLASSPATH=./gfal.jar:./gint.jar","
      LD_LIBRARY_PATH=.:$LD_LIBRARY_PATH",
      "LCG_GFAL_VO=gilda","LCG_RFIO_TYPE=dpm"};
Arguments = " 0 0 10 4 10000 aliserv6.ct.infn.it
      lfn:/grid/gilda/valeria/2000pillar.dat /gilda/issgc07/";
StdOutput = "sample.out";
StdError = "sample.err";
InputSandbox = {"startGen4.sh","gint.jar","gfal.jar",
      "libGFalFile.so"};
OutputSandbox = {"sample.err","sample.out", "sample.log"};
Requirements = Member("GLITE-3_0_0"
      ,other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

MS-HPC



Ontology Engineering Group

MS-HPC (II)

- "Engagement with standards"
  - High heterogeneity of computer systems (different types of computers with different types of OS)
  - Use of standards for achieving interoperability
  - File sharing
  - SUA (Subsytem for Unix-based Applications)
  - SCOM (Systems Centre Operations management)
    - Monitoring
  - MS-HPC support standard specifications
    - Basic Execution Service (BES), JSDL, etc.

System 5 Submit job file

```
<jsdl:JobDefinition>
    <jsdl:JobDescription>
            <jsdl:JobIdentification>
             </jsdl:JobIdentification>
            <jsdl:Application>
                <jsdl-hpcp:HPCProfileApplication name="HostDate" >
                    <jsdl-hpcp:Executable>C:\GridSchool\sfk\scan.bat
                      </jsdl-hpcp:Executable>
                    <jsdl-hpcp:Input></jsdl-hpcp:Input>
                    <jsdl-hpcp:Output>output.txt</jsdl-hpcp:Output>
                </jsdl-hpcp:HPCProfileApplication>
            </jsdl:Application>
        <jsdl:Resources>
            <jsdl:ExclusiveExecution>false</jsdl:ExclusiveExecution>
            <jsdl:TotalCPUCount/>
        </jsdl:Resources>
        <jsdl:DataStaging>
        </jsdl:DataStaging>
    </jsdl:JobDescription>
</jsdl:JobDefinition>
```

Submission script

```
#!/usr/bin/env perl

open(SUBMIT, ">submit-big");
print SUBMIT "Universe   = java\n";
print SUBMIT "Executable = sfkscanner.jar\n";
print SUBMIT "jar_files  = sfkscanner.jar,issgc_sfk_nesc.jar\n";
print SUBMIT "Log        = explorer.log\n";
print SUBMIT "Output     = explorer.\$(PROCESS).output\n";
print SUBMIT "Error      = explorer.\$(PROCESS).error\n";
print SUBMIT "transfer_input_files    = BoxData.txt,PillarsData2.txt.en\n";
print SUBMIT "should_transfer_files   = YES\n";
print SUBMIT "when_to_transfer_output = ON_EXIT\n";
print SUBMIT "\n";

#my $y;
for ($x = -15; $x < -11.12; $x += 0.1) {
   for ($y = 9861.9; $y < 9862.49 ; $y += 0.1 ) {
#        my $xx = $x + 10; #-15.0
#        my $yy = $y + 10; #9861.9
#       my $zz = $z;      #-11.12
#       my $ff = $f;      #9862.49
        print SUBMIT "Arguments  = uk.ac.nesc.toe.sfk.radar.Scanner ";
        print SUBMIT "$x $y -11.12 9862.49 0.001\n";
        print SUBMIT "Queue\n";
        print SUBMIT "\n";
        }

}
close(SUBMIT);
exit 0;
```

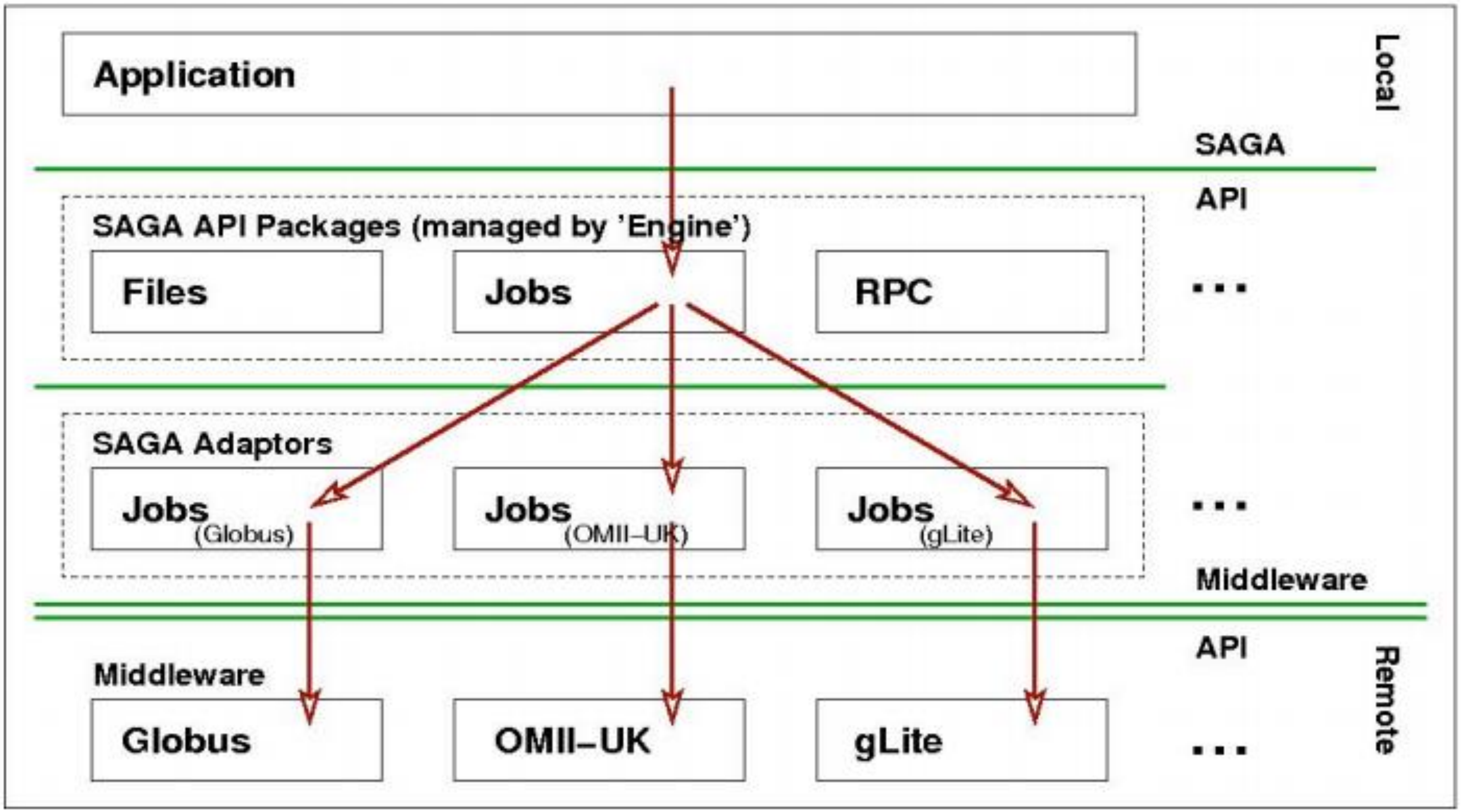
Table of Contents

- Introduction
- Grid Technologies
  - Security
  - Job submission
  - HTP (High Throughput Computing)
  - Grid Architectures
  - Distribution of data
  - Desktop Grids
- Grid Middleware
  - UNICORE
  - Condor
  - Globus
  - gLite
  - MS-HPC
- Middleware between Grid Middleware and end users
  - Services and Workflows

SAGA

- Simple API for Grid Applications (SAGA)
  - SAGA is a simple API (as it sounds)
  - SAGA is an emerging standard
- What is the Grid (brief reminder)
  - Dynamic
  - Heterogeneous
  - Complex
- What SAGA provides?
  - A high level abstraction hiding middleware details
  - Details of distribution are hidden
  - Simple API to access the existing Grid middleware
  - Is a client side software

SAGA Architecture



SAGA Example

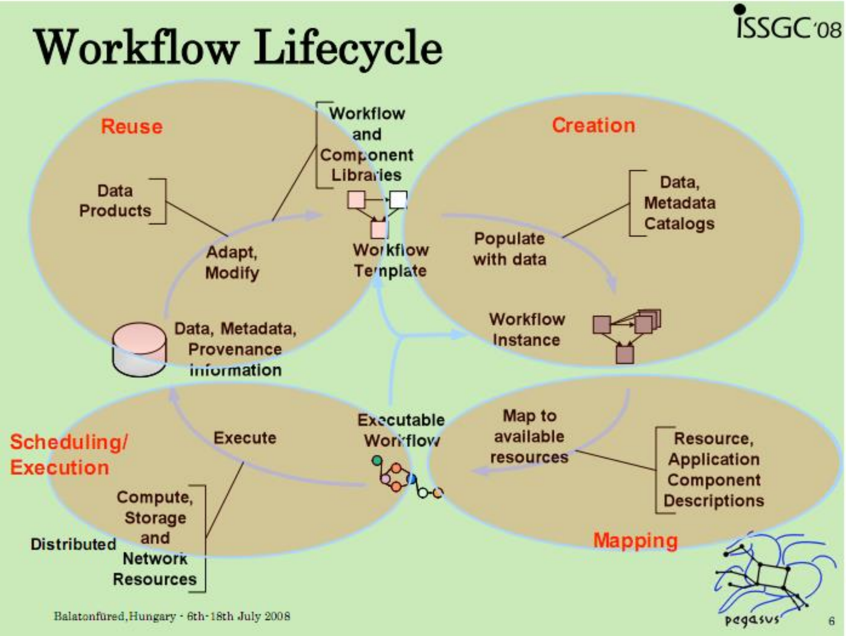
```
void copy_file(std::string source_url,
std::string target_url)
{
  try {
    saga::file f(source_url);
    f.copy(target_url);
  }
  catch (saga::exception const &e) {
    std::cerr << e.what() << std::endl;
  }
}
```

```
int copy_file (char const* source,   char const* target)
{
globus_url_t                    source_url;
globus_io_handle_t              dest_io_handle;
globus_ftp_client_operationattr_t  source_ftp_attr;
globus_result_t                 result;
globus_gass_transfer_requestattr_t source_gass_attr;
globus_gass_copy_attr_t         source_gass_copy_attr;
globus_gass_copy_handle_t       gass_copy_handle;
globus_gass_copy_handleattr_t   gass_copy_handleattr;
globus_ftp_client_handleattr_t  ftp_handleattr;
globus_io_attr_t                io_attr;
int                             output_file = -1;

if ( globus_url_parse (source_URL, &source_url) !=
GLOBUS_SUCCESS
  printf ("can not parse source_URL \"%s\"\n",
source_URL);
  return (-1);
}
…
```

Workflow Principles

- Scientific workflows
  - Allow to compose applications from other execution units/programs
  - Provide automation for the applications
  - Types of workflow applications
    - provide a service to a community
    - supporting community based analysis
    - process large amounts of shared data on shared resources
  - Critical issues
    - Find the right components
    - set the right parameters
    - find the right data
    - connect appropriate pieces together
- Workflow lifecycle
  - Creation
  - Mapping
  - Planning/execution
  - Reuse



Workflow principles (lifecycle I)

- Creation
  - Find the right components
  - set the right parameters
  - find the right data
  - connect appropriate pieces together
- Mapping
  - Map tasks to available resources
  - Where to run the computation units?
  - What data do I have to access?

Workflow principles (lifecycle II)

- Scheduling/execution
  - Consists in the execution of the planned workflows in other systems
    - Workflow systems send tasks to systems like Condor or Globus (depending on what specific functionality want to use)
- Reuse
  - how to find what is already there?
  - how to determine the quality of what is already there?
  - how to invoke an existing workflow?
  - how to share a workflow with a colleague?
  - how to share a workflow with a competitor?

Conclusions (I)

- Grid is about collaboration and sharing resources
  - Computational resources (large amount of computers, large systems)
  - Data resources
- New ways for doing research in many and different areas
  - High use of computing resources
  - Data intensive applications
  - Distributed applications
  - Grid creates virtual homogeneity of resources (virtualization)
  - E-Science, e-Infrastructure, grid security
- Virtual homogeneity
  - Standards

Conclusions (II)

- Systems provide similar functionalities but:
  - UNICORE focused in large scale systems (high computing power)
  - Condor focused in high amount of jobs and computers
  - Globus focused in high amount of computers
  - gLite high amount of computers/large infrastructure
- Same as data management in Grids:
  - Every problem (high rate of data generation, high amounts of data, high distribution of data) has its own solution
- Workflow systems allow to compose and submit complex jobs

Distributed Reasoning

- Distribution of data/ontologies
  - Where are the classes/properties/relations
  - Where are the instances
  - How to access them
  - Availability of the resources
- Distribution of processes
  - What processes can I distribute?
  - Where do I send these processes?
  - Reliability of the system
  - Gather and join the results
  - Possible use of workflow systems/scripts?