CENTER FOR COMPUTATION
& TECHNOLOGY

Grid Interoperability at the Application Level using SAGA

Shantenu Jha[12*], H Kaiser[1], A Merzky[1], O Weidner[1]

[1]Center for Computation and Technology, LSU
[2] Department of Computer Science, LSU

[*]Also with NeSC, Edinburgh and UC-London

Outline

- Interoperability: An Application's Perspective
  - Application vs Service
  - Grid Aware vs Unaware
- Simple API for Grid Applications(SAGA): Introduction
- SAGA: Interoperability at different levels
- SAGA in relation to GIN
- Application Example(s)

Motivation

- GIN:
  - Interoperation at the service level
  - Bottom-up approach
- Specific Application Projects:
  - "Federating Grids": SPICE, Cactus BH simulation
    - not scalable
  - Top-down approach
- e-Science Applications must be able to utilise infrastructure
  - Handle current heterogeneity and future developments
- "Real" eScience Applications: Interoperability(!)

CENTER FOR COMPUTATION & TECHNOLOGY

"Global" Grid Infrastructure

US TeraGrid

Starlight (Chicago)

SDSC

NCSA

PSC

Netherlight (Amsterdam)

UKLight

App Sci

NGS

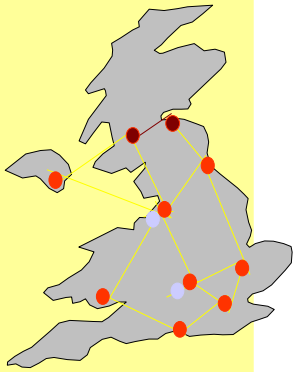
HPCx

Leeds

Manchester

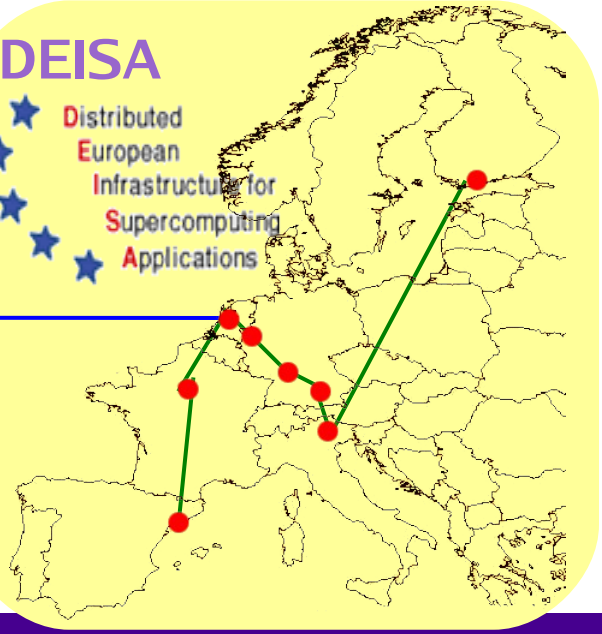
Oxford

RAL



DEISA

Distributed European Infrastructure for Supercomputing Applications

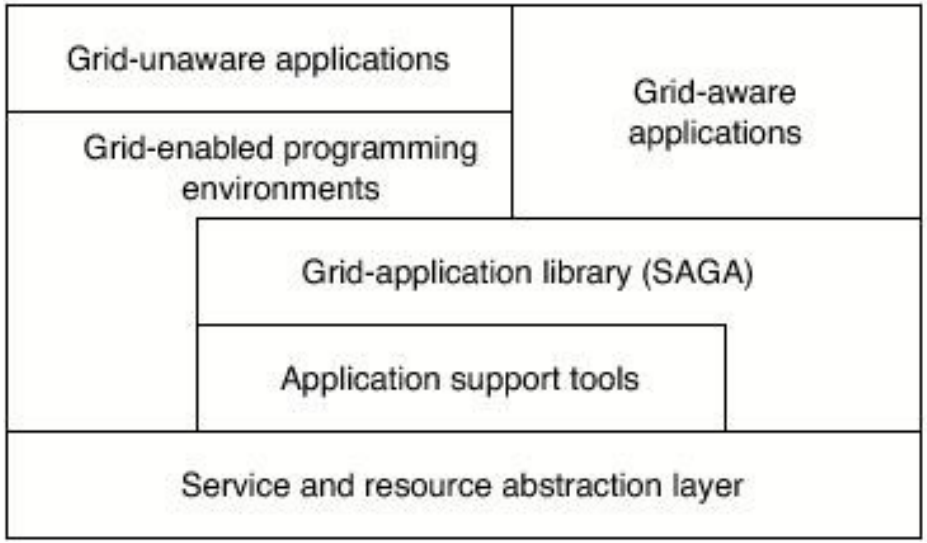


All sites connected by production network

Computation Network PoP Visualization

Application Level Interoperability (1)

Rough Taxonomy of Applications



- Some applications are Grid-unaware and want to remain so
  - Use tools/environments (e.g, NanoHub, GridChem)
  - May run on Grid-aware/Grid-enabled environments (e.g. Condor) or programming environment (e.g, MPICH-G2)
- Some applications are explicitly Grid-aware
  - Control, Interact & Exploit distributed systems at the *application level*

Application Level Interoperability (2)

- Some features (not definition!) of ALI:
  - Beyond compiling, no further changes for platform
  - Automated, scalable and extensible solutions to use new resources
    - Not via bilateral or bespoke arrangements
  - Semantics of any services are consistent (eg error handling)
  - ALI often requires QoS, co-scheduling
    - eg. remote vizualisation, multiple-resources
  - In general, GU applications require mostly SLI
    - establishing interoperabilty for GU is easier than GA applications

SAGA: In a Nutshell

- A lack of:
  - Programming interface that provides common grid functionality with the correct level of abstractions?
  - Ability to hide underlying complexities, varying semantics, heterogenities and changes from application program(er)
- Simple, integrated, stable, uniform, high-level interface
- Simplicity: Restricted in scope, 80/20
- Measure(s) of success:
  - Does SAGA enable quick development of “new” distributed applications?
  - Does it enable greater functionality using less code?

CENTER FOR COMPUTATION & TECHNOLOGY

Copy a File: Globus GASS

```
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);
}

if ( source_url.scheme_type != GLOBUS_URL_SCHEME_GSIFTP &&
     source_url.scheme_type != GLOBUS_URL_SCHEME_FTP    &&
     source_url.scheme_type != GLOBUS_URL_SCHEME_HTTP   &&
     source_url.scheme_type != GLOBUS_URL_SCHEME_HTTPS  ) {
  printf ("can not copy from %s - wrong prot\n", source_URL);
  return (-1);
}
globus_gass_copy_handleattr_init  (&gass_copy_handleattr);
globus_gass_copy_attr_init        (&source_gass_copy_attr);

globus_ftp_client_handleattr_init (&ftp_handleattr);
globus_io_fileattr_init           (&io_attr);

globus_gass_copy_attr_set_io      (&source_gass_copy_attr, &io_attr);
                                   &io_attr);
globus_gass_copy_handleattr_set_ftp_attr
                                  (&gass_copy_handleattr,
                                   &ftp_handleattr);
globus_gass_copy_handle_init      (&gass_copy_handle,
                                   &gass_copy_handleattr);

if (source_url.scheme_type == GLOBUS_URL_SCHEME_GSIFTP ||
    source_url.scheme_type == GLOBUS_URL_SCHEME_FTP    ) {
  globus_ftp_client_operationattr_init (&source_ftp_attr);
  globus_gass_copy_attr_set_ftp (&source_gass_copy_attr,
                                 &source_ftp_attr);
}
else {
  globus_gass_transfer_requestattr_init (&source_gass_attr,
                                  source_url.scheme);
  globus_gass_copy_attr_set_gass(&source_gass_copy_attr,
              &source_gass_attr);
}

output_file = globus_libc_open ((char*) target,
              O_WRONLY | O_TRUNC | O_CREAT,
              S_IRUSR  | S_IWUSR | S_IRGRP |
              S_IWGRP);
if ( output_file == -1 ) {
  printf ("could not open the file \"%s\"\n", target);
  return (-1);
}
/* convert stdout to be a globus_io_handle */
if ( globus_io_file_posix_convert (output_file, 0,
                                   &dest_io_handle)
     != GLOBUS_SUCCESS) {
  printf ("Error converting the file handle\n");
  return (-1);
}

result = globus_gass_copy_register_url_to_handle (
         &gass_copy_handle, (char*)source_URL,
         &source_gass_copy_attr, &dest_io_handle,
         my_callback, NULL);
if ( result != GLOBUS_SUCCESS ) {
  printf ("error: %s\n", globus_object_printable_to_string
          (globus_error_get (result)));
  return (-1);
}
globus_url_destroy  (&source_url);
return (0);
}
```

SAGA Example: Copy a File

High-level, uniform

```
#include <string>
#include <saga/saga.hpp>

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;
  }
}
```

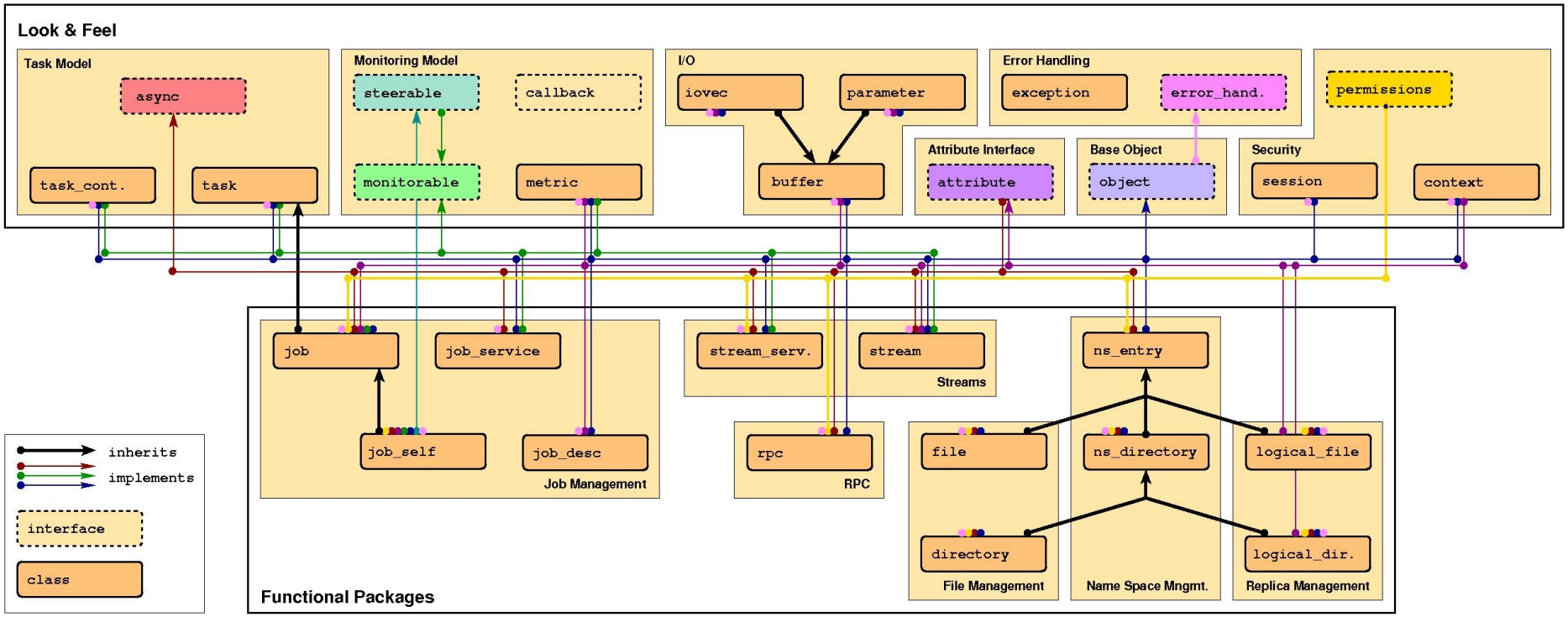
- Provides the high level abstraction, that application programmers need; will work across different systems
- Shields gory details of lower-level middle-ware and system issues
- Like MapReduce – leave details of distribution *etc.* out

SAGA: Scope

- Is:
  - Simple API for Grid-Aware Applications
    - Deal with distributed infrastructure explicitly
  - High-level (= application-level) abstraction
  - An uniform interface to different middleware(s)
  - Client-side software
- Is NOT:
  - Middleware
  - A service management interface!
  - Does not hide the resources - remote files, job (but the *details)*

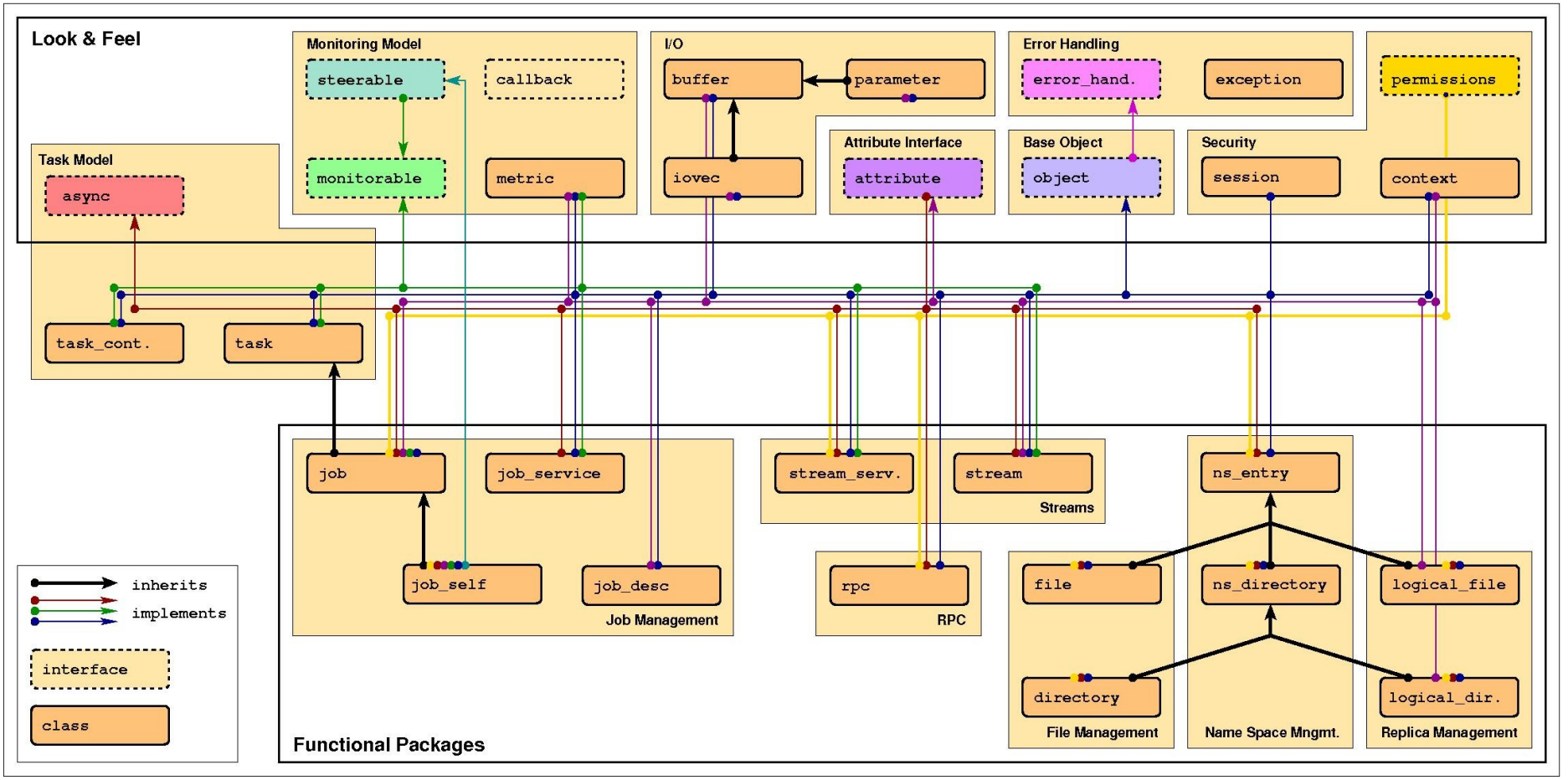
CENTER FOR COMPUTATION & TECHNOLOGY

SAGA: Class Diagram



CENTER FOR COMPUTATION & TECHNOLOGY

SAGA: Class Diagram (2)



CENTER FOR COMPUTATION & TECHNOLOGY

SAGA API: Towards a Standard

Standards help Interoperability

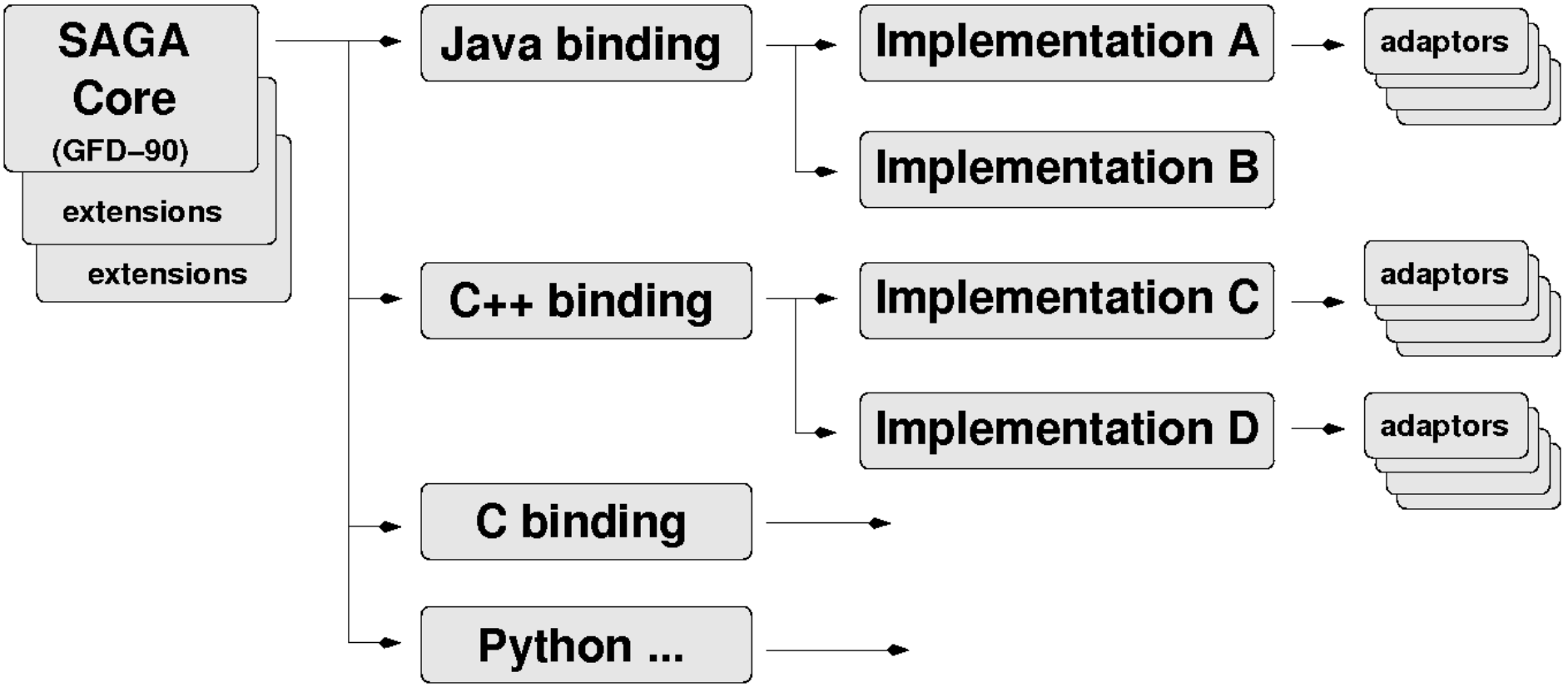
- The need for a standard programming interface
  - "Go it alone" versus "Community" model
  - Reinventing the wheel again, yet again, and again
  - MPI as a useful analogy of community standard
  - OGF the natural choice; establish SAGA-RG
- "Tedium" of the standardisation process?
  - Not all technology needs to be standardised upfront
  - Standardisation not a guarantee to success
- Requirements Document
  - Design and requirements derived from 23 Use Cases
  - Different projects, applications and functionality

SAGA API: Design & Specification

Process enabling Interoperability

- Open Grid Forum: Design and requirements derived from 23 Use Cases (some "correlated")
  - Different projects, applications and functionality
    - Biological, Coastal-modelling, visualization ..
  - Functional Areas: Job Mgmt, Resource Mgmt, Data Mgmt, Logical Files, Streams....
  - Non-functional Areas: Asynchronous, QoS, Bulk
- Interface is language independent, object-oriented and each sub-system is independent
- Specificed using Scientific Interface Description Language (SIDL)
  - extensible and easily implementable

The SAGA Landscape



SAGA Interface: Providing Interoperabiltiy
Job Submission API

```
01: // Submitting a simple job and wait for completion
02: //
03: saga::job_description jobdef;
04: jobdef.set_attribute ("Executable", "job.sh");
05:
06: saga::job_service js;
07: saga::job job = js.create_job ("remote.host.net", jobdef);
08:
09: job.run();
10:
11: while( job.get_state() == saga::job::Running )
12: {
13:   std::cout << “Job running with ID: “
14:             << job.get_attribute(“JobID”) << std::endl;
15:   sleep(1);
16: }
```

SAGA Implementation

How it supports Interoperability

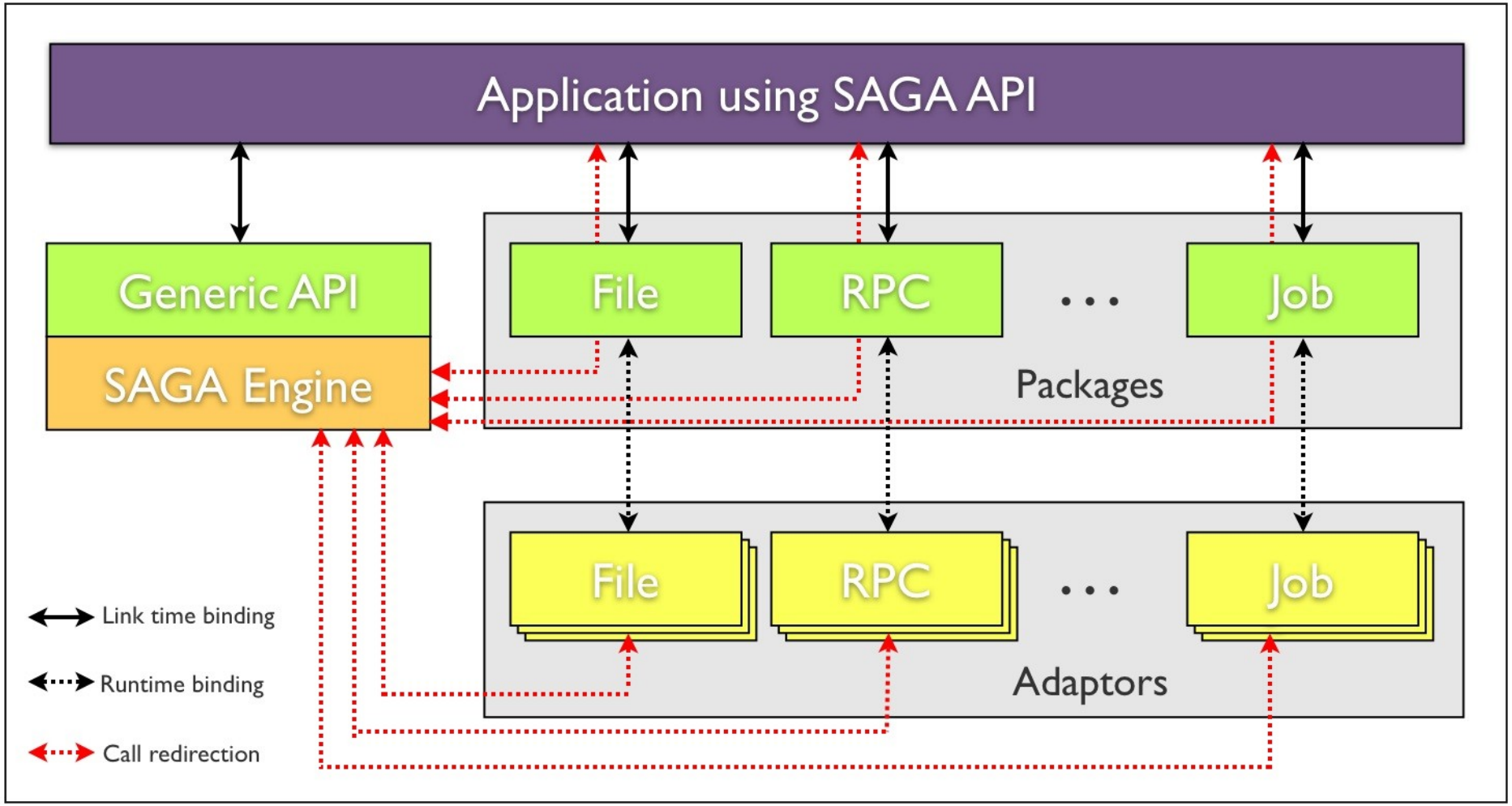
- Requirements
- Overall Architecture
  - Horizontal, Extensibility, Vertical
- Generic Call Routing
  - Route API calls to appropriate MW adaptors; underlying technology is Abstract Objects
- Lessons Learnt
  - Uniform implementations for different languages; C, C++, Python and Java
  - Flexible adaptor selection (more than one)

SAGA Implementation (2)
Some requirements to support interoperability

- Non-trivial set of requirements:
  - Allow heterogenous middleware to co-exist
  - Cope with evolving grid environments; dyn resources
  - Future SAGA API extensions
  - Portable, syntactically and semantically platform independent; permit latency hiding mechanisms
  - Ease of deployment, configuration, multiple-language support, documentation etc.
  - Provide synchronous, asynchronous & task versions

  Portability, modularity, flexibility, adaptabilty, extensibility

SAGA C++ (LSU)


Application using SAGA API
Generic API
SAGA Engine
File
RPC
...
Job
Packages
File
RPC
...
Job
Adaptors
Link time binding
Runtime binding
Call redirection

SAGA Implementation
How the Architecture Supports Interoperability

- Horizontal Extensibility – API Packages
  - Current packages:
    - file management, job management, remote procedure calls, replica management, data streaming
    - Steering, information services, checkpoint in pipeline
- Vertical Extensibility – Middleware Bindings
  - Different adaptors for different middleware
  - Set of 'local' adaptors
- Extensibility for Optimization and Features
  - Optimization: e.g. Bulk optimization, modular design
  - Features: e.g latency hiding, late binding etc.
  - PIMPL = Private (API..) + Implementation (Engine)

CENTER FOR COMPUTATION & TECHNOLOGY

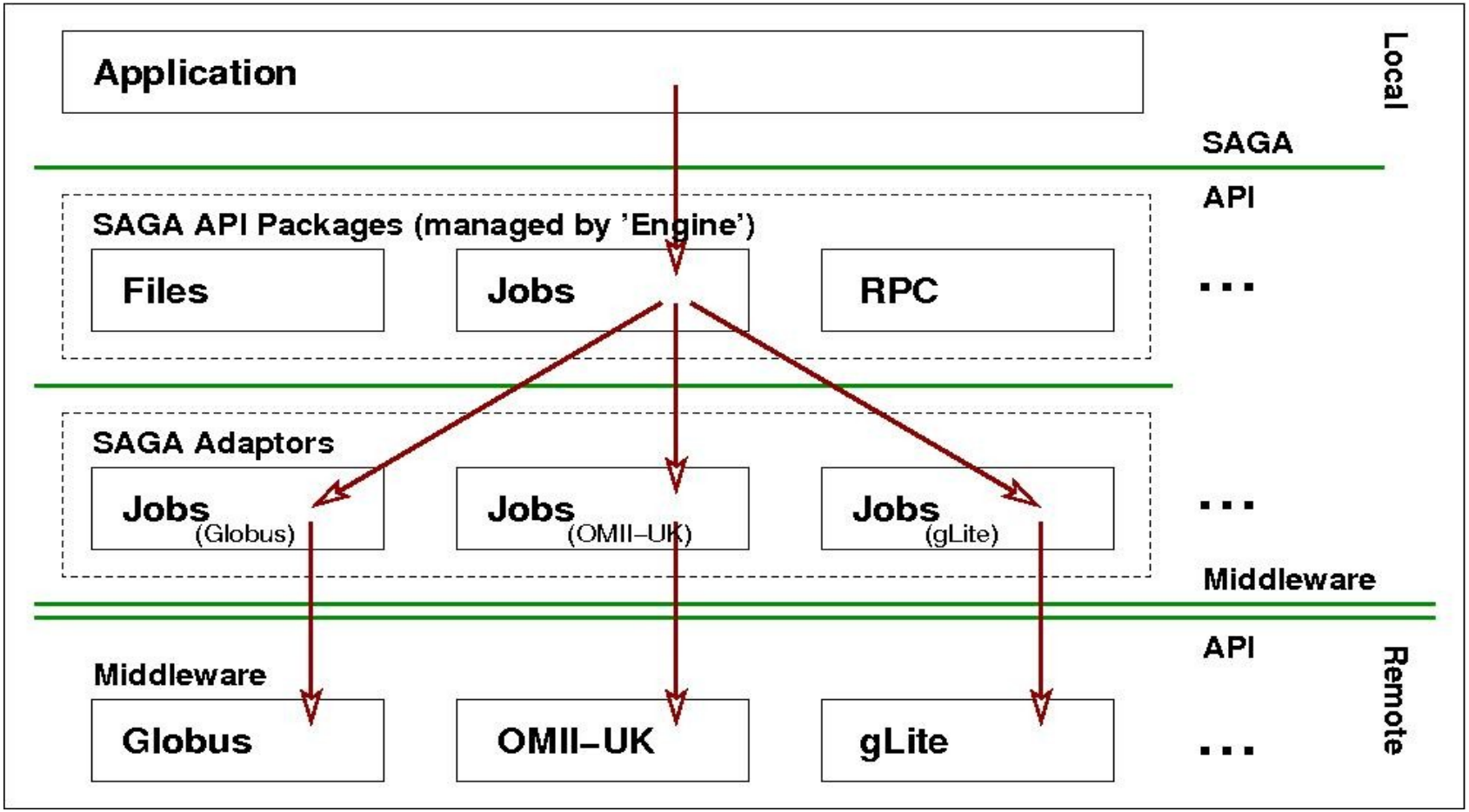
SAGA Implementation

How it supports Interoperability

- Requirements
- Overall Architecture
  - Horizontal, Extensibility, Vertical
- Generic Call Routing
  - Route API calls to appropriate MW adaptors; underlying technology is Abstract Objects
- Lessons Learnt
  - Uniform implementations for different languages; C, C++, Python and Java
  - Flexible adaptor selection (more than one)

SAGA: Interoperable Job Submission

Role of Adaptors (middleware binding)

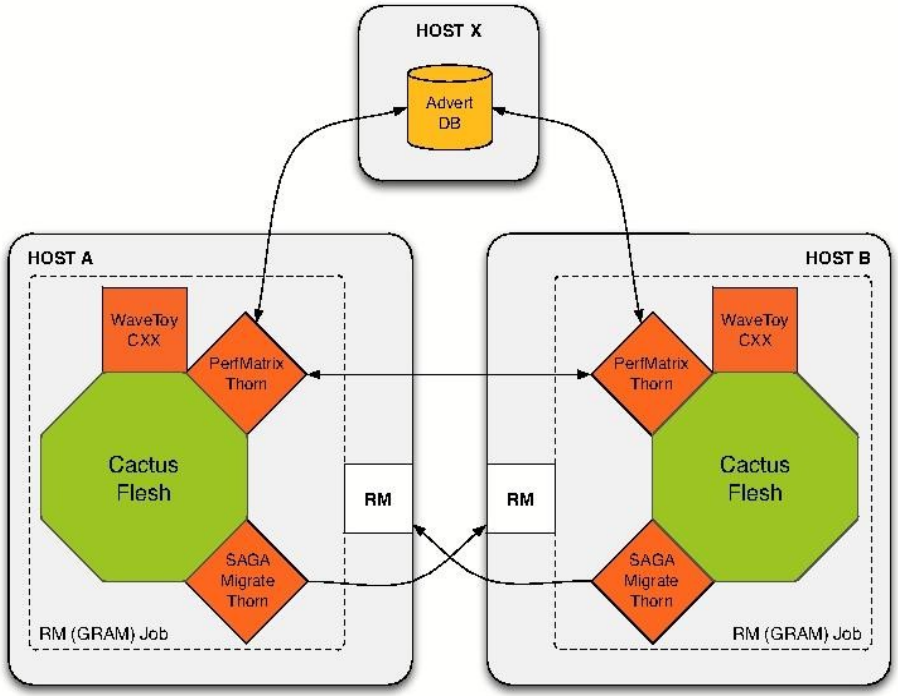


SAGA and (the five areas of) GIN

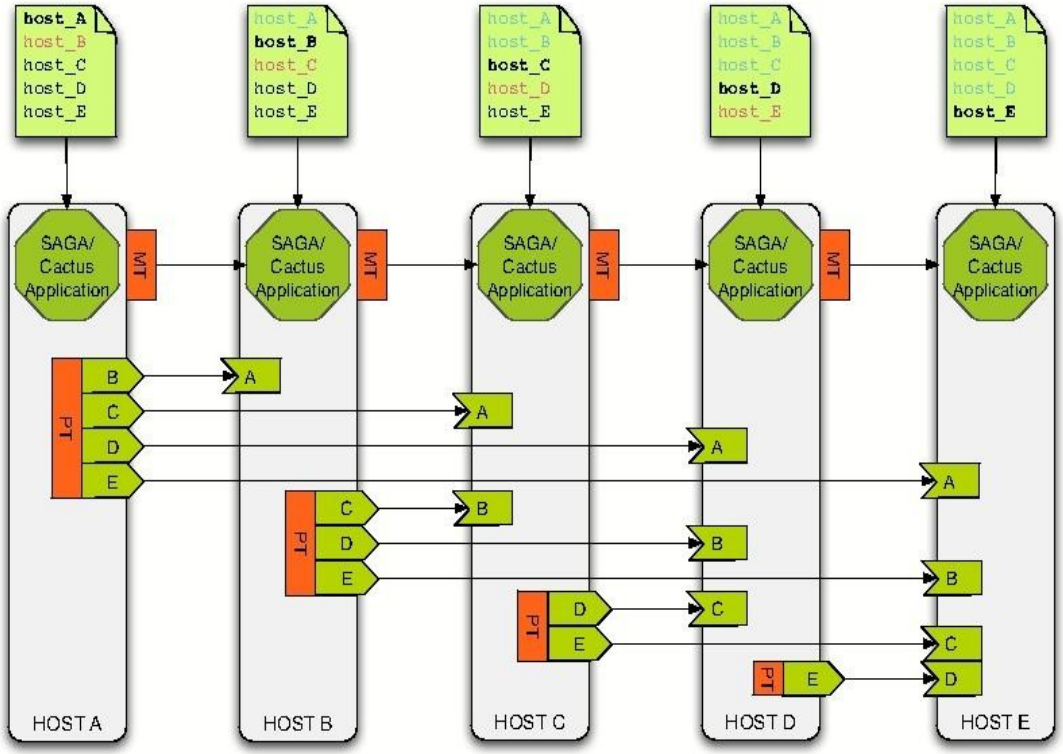
- SAGA and ...
  - Job Submission & Management
    - Provides same functionality as OGSA-BES, but at a higher level of abstraction, e.g., job id w/o contacting job service
    - Same job state model as BES (not a random coincidence)
  - Data Management and Movement
    - Data movement protocol agnostic
    - Supports pseduo-schema (any://remote.host.net/dir/file.typ) and allows implementation to determine
  - Authorization and Identity Management
    - Essentially not exposed at application development level
  - Cross Grid Applications
  - Information Services and Modelling

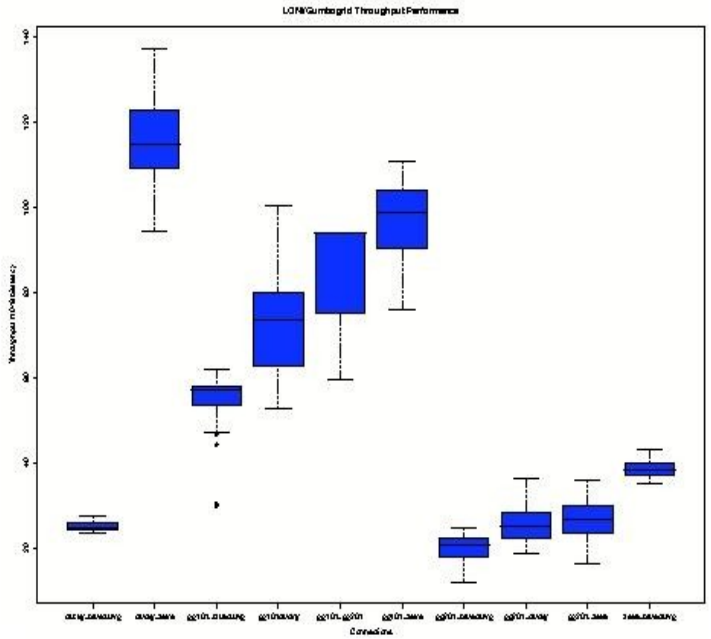
Network Performance Aware Application (1)

- Capable of acquiring application-specific network characteristic data and determining ideal migration target across heterogeneous grids without changes in the application



Network Performance Aware Application (2)





Future Work

- Many applications under development...
- Middleware bindings (adaptors): Key to “realising' interoperability
- Use SAGA based applications in the next generation of GIN demonstrations(!)
  - Suggestions for applications?
  - US TeraGrid and EU Deisa?

CENTER FOR COMPUTATION & TECHNOLOGY

Acknowledgments

- Funding Agencies:
  - UK EPSRC:
    - Through OMII-UK “OMII SAGA” project
    - Through NeSC, Edinburgh Distributed Programming Abstractions
  - US NSF
    - Cybertools projects
  - CCT Internal Funds

Acknowledegments: The SAGA Team

CENTER FOR COMPUTATION & TECHNOLOGY

Hartmut Kaiser

Thilo Kielmann

Andre Merzky

Ceriel Jacobs

Ole Weidner

Kees Verstop

+ Many other students

Upcoming API Extensions

- MessageBus
  - Structured data transfer, also many-to-many
- Service Discovery
  - Based on GLUE schema
- Adverts
  - Persistent storage of application-level data
- Checkpointing/Recovery
  - Based on GridCPR