CENTER FOR COMPUTATION
& TECHNOLOGY

SAGA

A Simple API for Grid Applications

Introduction to the SAGA API

omii-uk
www.omii.ac.uk

Outline

- SAGA Standardization
- API Structure and Scope (C++)
- API Walkthrough
- SAGA SoftwareComponents
  - Command Line Tools
  - Python API bindings
  - C++ API bindings
  - [Java API bindings]

SAGA: Teaser

```
// SAGA: File Management example

#include <saga/saga.hpp>

int main ()
{
  saga::filesystem::directory dir ("any://remote.host.net//data/");

  if ( dir.exists ("a") && ! dir.is_dir ("a") )
  {
    dir.copy ("a", "b", saga::filesystem::Overwrite);
  }

  list <saga::url> names = dir.find ("*-{123}.txt");

  saga::filesystem::directory tmp  = dir.open_dir ("tmp/", saga::fs::Create);
  saga::filesystem::file      file = dir.open     ("tmp/data.txt");

  return 0;
}
```

DC-APIs: some observations

- diversity of (Grid) middleware implies diversity of APIs
- middleware APIs are often a by-product
- APIs are difficult to sync with middleware development, and to stay **simple**
- successful APIs generalize programming concepts
  - MPI, CORBA, COM, RPC, PVM, SSH, …
- no new API standards for distributed computing
  - !standard: Globus, gLite, Unicore, Condor, iRods, …

Open Grid Forum (OGF)

- The Open Grid Forum (aka GF, EGF, GGF, EGA) standardizes distributed computing infrastructures/MW
- e.g. GridFTP, JSDL, OCCI, …
- focuses on interfaces, but also protocols, architectures, APIs
- driven by academia, but some buy-in / acceptance in industry
- cooperation with SDOs like SNIA, DMTF, IETF, etc.

APIs within OGF

- OGF focuses on services
- numerous service interfaces
  - often WS-based, but also REST, others
- some effort on higher level APIs
  - Distributed Resource Management Application API (DRMAA)
  - Remote Procedure Calls (GridRPC)
  - Checkpoint and Recovery (GridCPR)
  - [Job Submission and Description Language (JSDL)]

OGF: DRMAA

- implementable on all major resource management services
- simple means to define and submit jobs
- basic job management features (status, kill)
- job templates for bulk job management
- DRMAA.v2 is expected by end of 2010 (oops)
  - OO
  - extended
  - SAGA aligned !

DRMAA Example

```
drmaa_job_template_t * job_template;

if ( ! ( job_template = create_job_template (exe, 5, 0) ) )
{
  fprintf (stderr, "create_job_template failed\n");
  return 1;
}

while ( ( drmaa_errno = drmaa_run_job (job_id,
                                       sizeof (jobid)-1,
                                       job_template,
                                       diagnosis,
                                       sizeof (diagnosis)-1) )
         == DRMAA_ERRNO_DRM_COMMUNICATION_FAILURE )
{
 fprintf (stderr, "drmaa_run_job failed: %s\n", diagnosis);
 sleep (1);
}
```

OGF: GridRPC

- standardizes the three existing RPC implementations for Grids (Ninf-G, DIET)
- example of *'gridified API'*
- simple: get function handle, call function
- explicit support for asynchronous method calls
- GridRPC.v2 adds support for remote persistent data handles
  - SAGA aligned !

OGF: GridRPC

```
double A[N*N], B[N*N], C[N*N];

initMatA (N, A);
initMatB (N, B);

grpc_initialize (argv[1]);

grpc_function_handle_t         handle;
grpc_function_handle_default (&handle, "mat_mult");

if ( grpc_call (&handle, N, A, B, C) != GRPC_NO_ERROR )
{
  exit (1);
}

grpc_function_handle_destruct (&handle);
grpc_finalize ();
```

OGF: GridCPR (Checkpoint & Recovery)

- Grids seem to favor application level checkpointing
- GridCPR
  - allows to manage checkpoints
  - defines an architecture, service interfaces, and scope of client API
  - SAGA aligned !
- not many implementations exist, usage declining
  - virtualized hardware makes CPR somewhat superfluous

OGF: JSDL

- extensible XML based language for describing job requirements
- does not cover resource description (on purpose) does not cover workflows, or job dependencies etc (on purpose)
- JSDL is extensible (ParameterSweep, SPMD, ...)
- top-down approach
- SAGA leans on JSDL for job description
  - future revisions of SAGA will support JSDL directly

OGF: JSDL

```
<jsdl:JobDefinition>
 <JobDescription>
  <Application>
   <jsdl-posix:POSIXApplication>
    <Executable>/bin/date</Executable>
   </jsdl-posix:POSIXApplication>
  </Application>
  <Resources ...>
   <OperatingSystem>
    <OperatingSystemType>
     <OperatingSystemName>LINUX</OperatingSystemName>
    </OperatingSystemType>
   </OperatingSystem>
  </Resources>
 </JobDescription>
<jsdl:JobDefinition>
```

OGF: top-down vs. bottom-up

- bottom-up often agrees on (semantic) LCD + backend specific extensions
- top-down usually focuses on semantics of application requirements

- bottom-up tends to be more powerful
- top-down tends to be simpler and more concise

- ***we very much prefer top-down!***

OGF: Summary

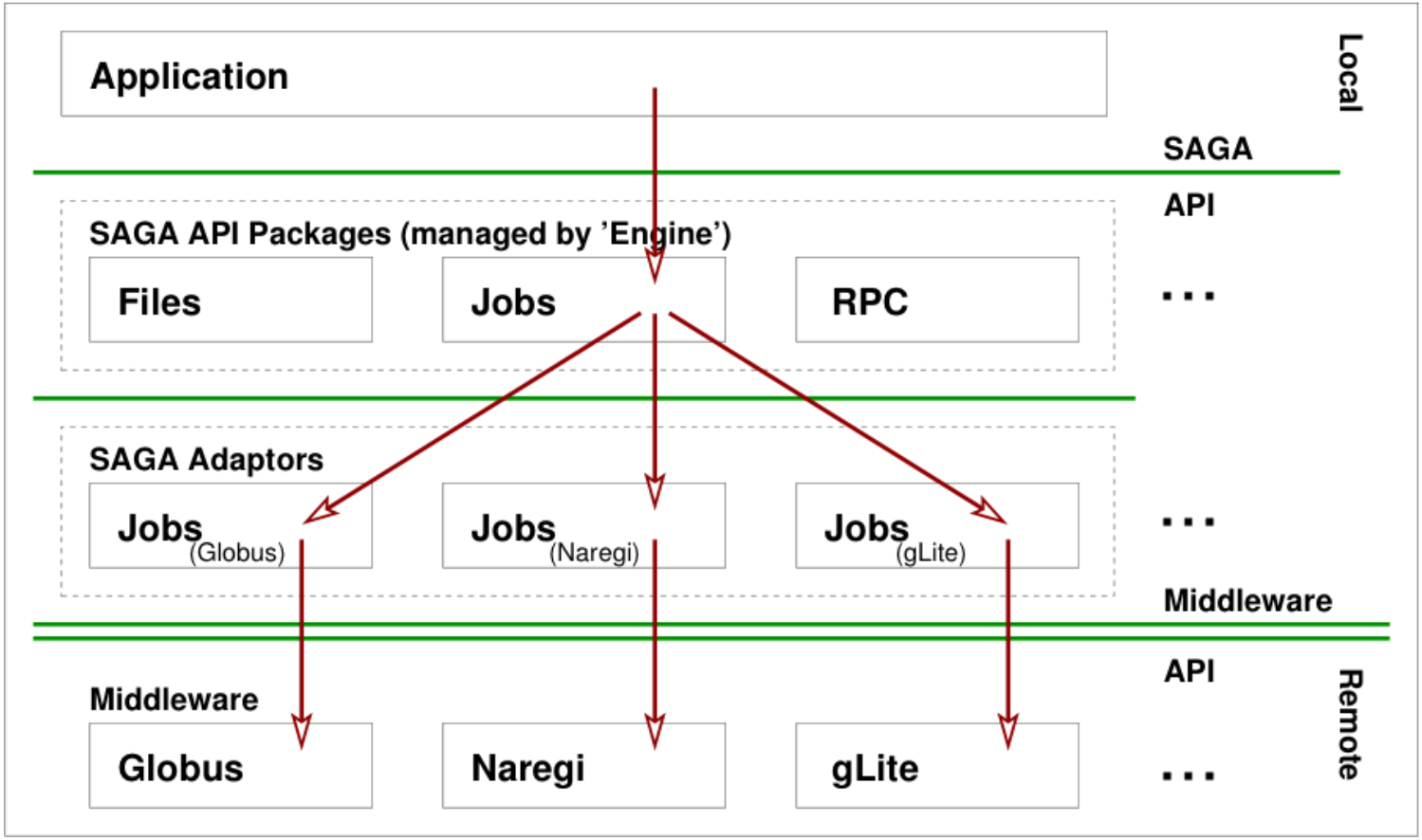
- some high-level API specs exist in OGF, and are successful
- OGF APIs do not cover the complete OGF scope
- the various API standards are disjoint
- APIs are defined bottom-up
- WSDL & Co cannot replace application level APIs

- **SAGA tries to address these issues**

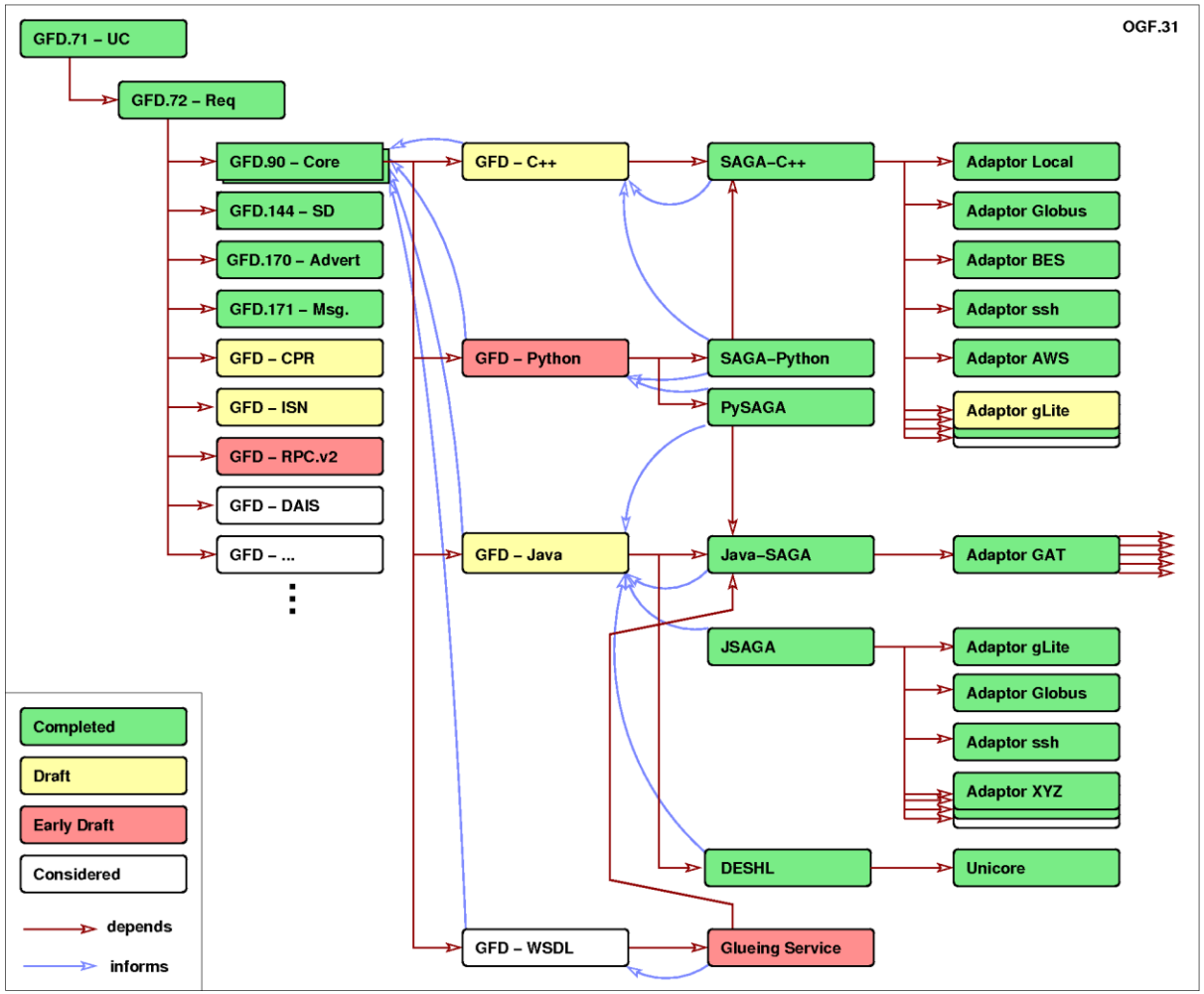
SAGA Design Principles

- SAGA: Simple API for Grid Applications
  - OGF approach to a uniform API layer (facade)
- top-down approach
  - use case driven!
  - defines application level abstractions
- governing principle: 80:20 rule
  - simplicity versus control!
- extensible
  - stable look & feel
  - API packages
- API Specification is language independent (IDL)
  - Renderings exist in C++, Python, Java
  - focus today on C++ or Python

Implementation



SAGA API Landscape



SAGA Intro: Example 1

```
// SAGA: File Management example

saga::filesystem::directory dir ("any://remote.host.net//data/");

if ( dir.exists ("a") && ! dir.is_dir ("a") )
{
  dir.copy ("a", "b", Overwrite);
}

list <saga::url> names = dir.find ("*-{123}.txt");

saga::filesystem::directory tmp  = dir.open_dir ("tmp/", Create);
saga::filesystem::file      file = dir.open     ("tmp/data.txt");

file.copy (“txt/data.bak”);
```

SAGA Intro: Example 1

- API is clearly POSIX (libc + shell) inspired
- where is my security??
- what is 'any://' ???
- usage should be intuitive (hopefully)
- correct level of abstraction?

SAGA Intro: Example 2

```
// SAGA: Job Submission example

saga::job::description jd;     // details on later slides

saga::job::service js ("any://remote.host.net/");
saga::job::job     j = js.create_job (jd);

j.run ();

cout << "Job State: " << j.get_state () << endl;

j.wait ();

cout << "Retval " << j.get_attribute ("ExitCode") << endl;
```

SAGA Intro: Example 2'

```
// SAGA: Job Submission example




saga::job::service js ("any://remote.host.net");
saga::job::job     j = js.run_job ("touch /tmp/touch.me");



cout << "Job State: " << j.get_state () << endl;

j.wait ();

cout << "Retval " << j.get_attribute ("ExitCode") << endl;
```

SAGA Intro: Example 2

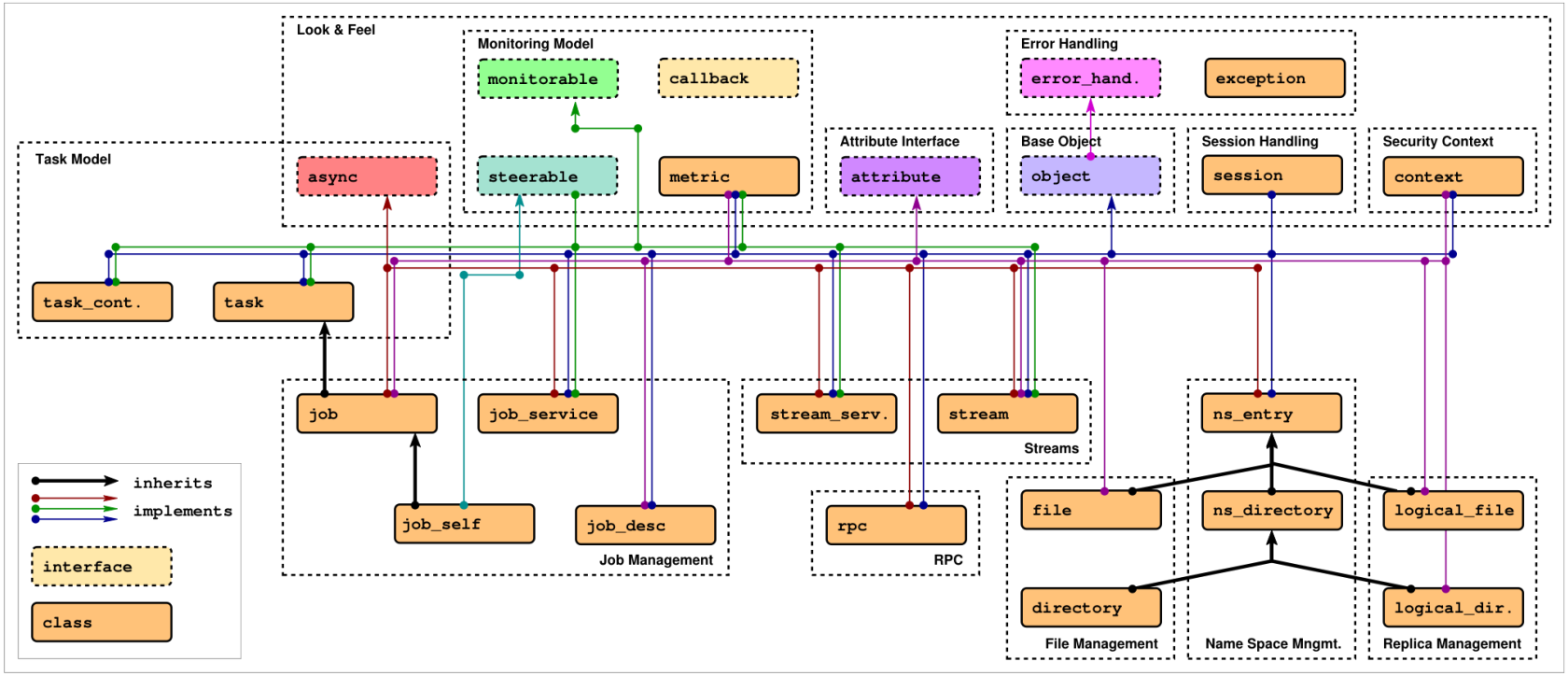
- stateful objects!
- yet another job description language? :-(
- many hidden/default parameters
  - keeps call signatures small
- 'any://' again!
- TIMTOWTDI (there is more than one way to do it)

SAGA Intro: 10.000 feet

- object oriented:
  - uses inheritance and interfaces
  - very moderate use of templates though!
- functional and non-functional elements strictly separated
  - functional API:
    - typically mappable to remote operations
    - ordered in API 'Packages': extensible
  - non-functional API:
    - typically not mappable to explicit remote operations
    - "look & feel": orthogonal to functional API
    - security, asynchronous ops, notifications, ...
- few inter-package dependencies - allows for partial implementations

SAGA Class Hierarchy

SAGA: Class hierarchy



SAGA: Class hierarchy

Functional API Packages

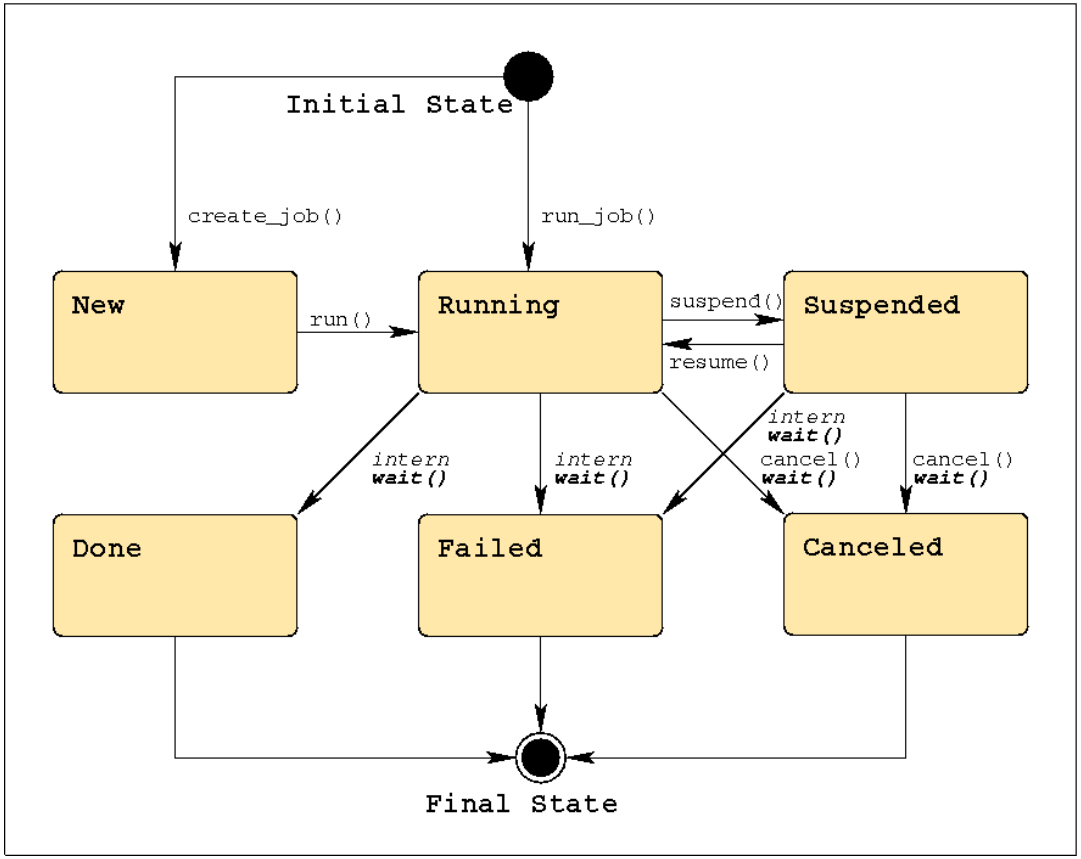
SAGA Job Package: Overview

- running jobs is **use case #1**
- all middlewares support it, one way or the other
- well established patterns exist
  - job description
  - job state
  - submission endpoint
  - ...

SAGA Job Package: Example 1

```
saga::job::description jd;
saga::job::service     js ("gram://remote.host.net");
saga::job              j = js.create_job (jd);

j.run ();

cout << "Job State: " << j.get_state () << endl;

j.wait ();

cout << "Retval " << j.get_attribute ("ExitCode") << endl;
```

SAGA Job Package: job states



SAGA Job Package: job operations

```
j.run     ();
j.wait    ();
j.cancel  ();

j.suspend ();
j.resume  ();

j.signal     (SIGUSR1);
j.checkpoint ();
j.migrate    (jd);
```

SAGA Job Package: job description

```
saga::job::description jd;

jd.set_attribute ("Executable",       "/bin/tail");
jd.set_attribute ("WorkingDirectory", "data/");
jd.set_attribute ("Cleanup",          "False");

// pseudo code *blush*
jd.set_vector_attribute ("Arguments",    ["-f", "my_log"]);
jd.set_vector_attribute ("Environment",  ["TMPDIR=/tmp/"]);
jd.set_vector_attribute ("FileTransfer", ["my_log >> all_logs"]);
```

SAGA
A Simple API for Grid Applications

SAGA Job Package: job description

Executable	WorkingDirectory	TotalPhysicalMemory
Arguments	*Interactive*	CPUArchitecture
Environment	Cleanup	OperatingSystemType
CandidateHosts	Input	*Queue*
SPMDVariation	Output	JobProject
TotalCPUCount	Error	*JobContact*
NumberOfProcesses	*JobStartTime*	FileTransfer
ProcessesPerHost	WallTimeLimit	
ThreadsPerProcess	TotalCPUTime	

SAGA Job Package: job description

- leaning heavily on JSDL, but flat, borrows from DRMAA
- mixes hardware, software and scheduling attributes!
- cannot be extended
- no support for 'native' job descriptions (RSL, JDL, ...) - **yet**
- only 'Executable' is required
- backend MAY ignore unsupported keys

  cd /tmp/data && rm -rf *

SAGA Job Package: job service

```
saga::job::service js ("gram://remote.host.net/");

vector <string> ids = js.list ();  // list known jobs

while ( ids.size () )
{
  string    id = ids.pop_back ();  // fetch one job id
  saga::job j  = js.get_job (id);  // reconnect to job

  cout << id << " : " << j.get_state () << endl;
}
```

SAGA Job Package: job service

- represents a specific job submission endpoint
- job states are maintained on that endpoint (usually)
- full reconnect may not be possible (I/O streaming)
- lifetime of state up to backend
- reconnected jobs may have different job description (lossy translation)

SAGA Namespace Package

- interfaces for managing entities in name spaces
- files, replicas, information, resources, steering parameter, checkpoints, . . .
- manages hierarchy (mkdir, cd, ls, . . .)
- entries are assumed to be opaque (copy, move, delete, ...)

SAGA Namespace Package: example

```
saga::name_space::directory d ("ssh://remote.host.net//data/");

if ( d.is_entry ("a") && ! d.is_dir ("a") )
{
  d.copy ("a", "../b");
  d.link ("../b", "a", Overwrite);
}

list <saga::url> names = d.find ("*-{123}.text.");

saga::name_space::directory tmp = d.open_dir ("tmp/data/1",
                                    saga::name_space::CreateParents);

saga::name_space::entry data = tmp.open ("data.txt");

data.copy ("data.bak", Overwrite); // uses cwd
```

SAGA Namespace Package

- name space entries are opaque: the name space package can never look inside
- directories *are* entries (inheritance)
- inspection: `get_cwd()`, `get_url()`, `get_name()`, `exists()`, `is_entry()`, `is_dir()`, `is_link()`, `read_link()`
- manipulation: `create()`, `copy()`, `link()`, `move()`, `remove()`
- permissions: `permissions_allow()`, `permissions_deny()`
- wildcards are supported (POSIX influence...)

SAGA Filesystem Package

- implements name space interface
- adds access to **content** of namespace::entries (files)
- POSIX oriented: `read()`, `write()`, `seek()`
- optimizations: for distributed file access:
  - scattered I/O
  - pattern based I/O
  - extended I/O (from GridFTP)

SAGA Filesystem Package: Example

```
saga::filesystem::file f ("any://remote.host.net/data/data.bin");

char mem[1024];
saga::mutable_buffer buf (mem);

if ( f.get_size () >= 1024 )
{
  buf.set_data (mem + 0, 512);
  f.seek (512, saga::filesystem::Start);
  f.read (buf);
}

if ( f.get_size () >= 512 )
{
  buf.set_data (mem + 512, 512);
  f.seek (0, saga::filesystem::Start);
  f.read (buf);
}
```

SAGA Filesystem Package

- provides access to the **content** of filesystem entries (sequence of bytes)
- saga buffers are used to wrap raw memory buffers
- saga buffers can be allocated/managed by the SAGA implementation
- several incarnations of read/write: posix style, scattered, pattern based

SAGA Filesystem Package: Flags

```
enum flags {
  None            =    0,
  Overwrite       =    1,
  Recursive       =    2,
  Dereference     =    4,
  Create          =    8,
  Exclusive       =   16,
  Lock            =   32,
  CreateParents   =   64,
  Truncate        =  128,  // not on name_space
  Append          =  256,  // not on name_space
  Read            =  512,
  Write           = 1024,
  ReadWrite       = 1536,  // Read | Write
  Binary          = 2048   // only on filesystem
}
```

SAGA Advert Package

- persistent storage of application level information
- semantics of information defined by application
- allows storage of serialized SAGA objects (object persistency)
- ***very useful for bootstrapping and coordinating distributed application components***

SAGA Advert Package: Example

```
// example for browing my task adverts
saga::advert::directory todo ("any//remote.host.net/my_tasks/");

// pseudo vector code
list <saga::url> urls = todo.find ("*", ["priority=urgent"]);

while ( urls.size () )
{
  saga::advert ad (urls.pop_front ());
  std::cout << ad.get_attribute ("title")       << std::endl;
  std::cout << ad.get_attribute (« deadline")   << std::endl;
  std::cout << ad.get_attribute ("description") << std::endl << std::endl;
}
```

SAGA Advert Package: Example

```
// master side code: advertise (publish) a saga::file instance
saga::file f (url);

saga::advert ad ("any//remote.host.net/files/my_file_ad", Create);

ad.store_object (f);
```

```
// client side code: retrieve file instance
saga::advert ad ("any//remote.host.net/files/my_file_ad");

saga::file f = ad.retrieve_object ();
```

SAGA: Class hierarchy

Look & Feel Packages

SAGA Session: Example – default session

```
saga::ns_dir dir ("any://remote.host.net//data/");

if ( dir.is_entry ("a") && ! dir.is_dir ("a") )
{
  dir.copy ("a", "../b");
  dir.link ("../b", "a", Overwrite);
}

list <saga::url> names = dir.find ("*-{123}.text.");

saga::name_space::directory sub   = dir.open_dir ("tmp/");
saga::name_space::entry     entry = dir.open     ("data.txt");

entry.copy ("data.bak", Overwrite);
```

SAGA Session: Properties

- by default hidden (default session is used)
- session is identified by lifetime of security credentials, and by objects in this session (jobs etc.)
- session is used on object creation (optional)
- saga::context can attach security tokens to a session
  - the default session has default contexts

SAGA Session: Example – explicit session

```
saga::context c1 (saga::context::X509);
saga::context c2 (saga::context::X509);

c2.set_attribute ("UserProxy", "/tmp/x509up_u123.special");

saga::session s;

s.add_context (c1);
s.add_context (c2);

saga::name_space::dir dir (s, "any://remote.host.net/data/");
```

SAGA Session: Lifetime

```
saga::filesystem::file f;

{
  saga::context c (saga::context::X509);

  c.set_attribute ("UserProxy", "/tmp/x509up_u123.special");

  saga::session s;  s.add_context (c);

  saga::filesystem::dir d (s, "gridftp://remote.host.net/data/");

  s.remove_context (c1);

  f = d.open (“a");
}

f.copy ("b");  // this works – session and context are sticky!
```

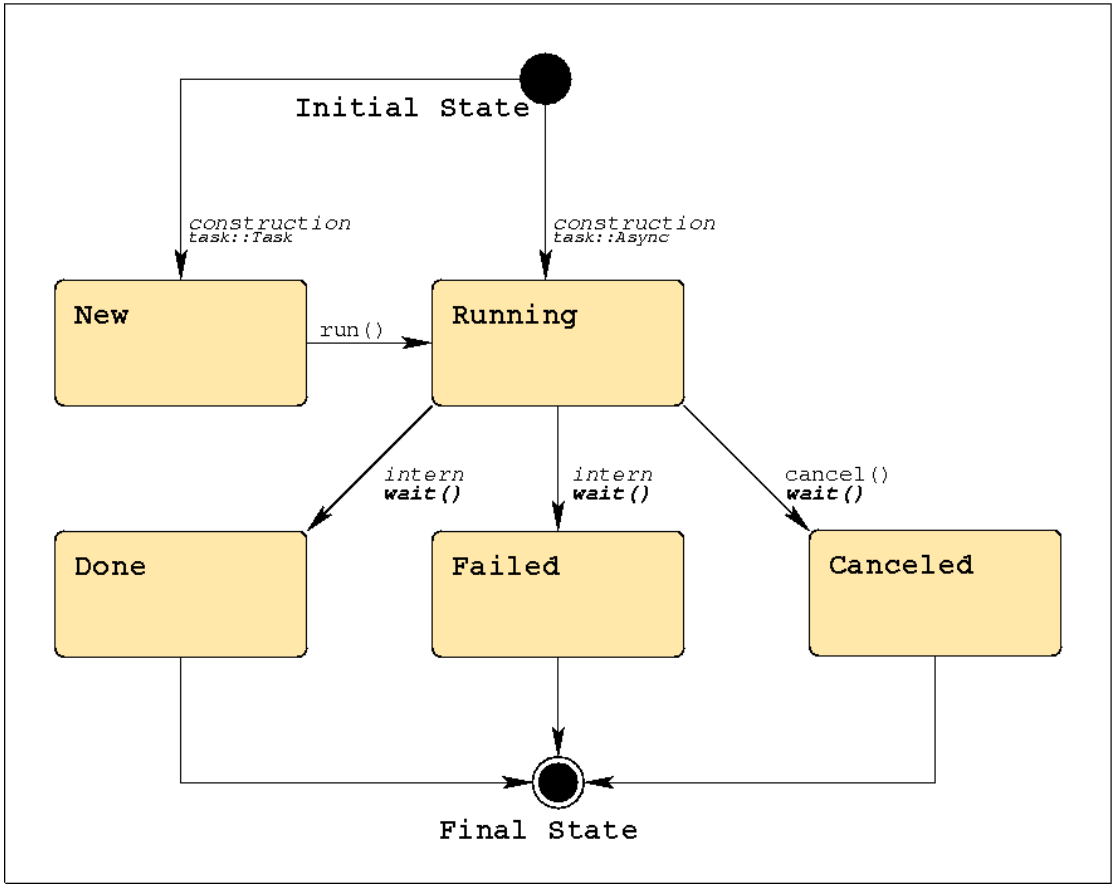
SAGA Session: Lifetime

```
// as a rule: don’t worry about object lifetime too much...

saga::session s;
saga::context c (saga::context::X509);

c.set_attribute ("UserProxy", "/tmp/x509up_u123.special");
s.add_context (c);

saga::filesystem::dir  d (s, "gridftp://remote.host.net/data/");
saga::filesystem::file f = d.open (“a");

f.copy ("b");
```

SAGA Tasks

- asyncronous operations are a MUST in distributed systems
- `saga::task` represents an asyncronous operation (e.g. `file.copy ()`)
- `saga::task_container` manages multiple tasks
- tasks are stateful (similar to jobs)

SAGA Tasks: States



SAGA Tasks

- different versions for each method call: `Sync, Async, Task`
- signature basically the same
- differ in state of the task returned by that method
  - `Sync:` task is `Done` `-> create () ; run () ; wait () ;`
  - `Async:` task is `Running` `-> create () ; run () ;`
  - `Task:` task is `New` `-> create () ;`
- delayed exception delivery

```
if ( saga::task::Failed == task.get_state () )
{
    task.rethrow ();
}
```

SAGA Tasks

- different versions for each method call: `Sync, Async, Task`
- signature basically the same
- differ in state of the task returned by that method
  - `Sync:` task is `Done` -> `get_result ();`
  - `Async:` task is `Running` -> `wait(); get_result ();`
  - `Task:` task is `New` -> `run(); wait(); get_result ();`
- delayed exception delivery

```
if ( saga::task::Failed == task.get_state () )
{
    task.rethrow ();
}
```

SAGA Task: Example

```
// normal method call, synchronous
/* void */ file.copy ("data.bak");

// async versions, never throw (use 'rethrow()' on failure)
saga::task t1 = file.copy     <saga::task::Sync>  ("data.bak.1");
saga::task t2 = file.copy     <saga::task::Async> ("data.bak.2");
saga::task t3 = file.copy     <saga::task::Task>  ("data.bak.3");

// t1: Done
// t2: Running
// t3: New

t3.run ();    // t3 now Running, too

t2.wait ();
t3.wait ();

// t1, t2, t3 are final (Done or Failed)
```

SAGA Task: Example

```
// normal method call, synchronous
size_t s = file.get_size ();

// async versions, never throw (use 'rethrow()' on failure)
saga::task t1 = file.get_size <saga::task::Sync>  ();
saga::task t2 = file.get_size <saga::task::Async> ();
saga::task t3 = file.get_size <saga::task::Task>  ();

// get_result: implies wait() and rethrow(), and thus can throw!
size_t s1 = t1.get_result <size_t> ();
size_t s2 = t2.get_result <size_t> ();
size_t s3 = t3.get_result <size_t> ();
```

SAGA Task Container: Example

```
// create task container
saga::task_container tc;

// add tasks
tc.add (t1);
tc.add (t2);
tc.add (t3);

// collective operations on all tasks in container
tc.run ();

saga::task done_task = tc.wait (saga::task::Any);

tc.wait (saga::task::All);
```

SAGA Task Container: Tasks and Jobs

```
// NOTE:
// class saga::job : public saga::task
// task container can thus manage tasks *and* jobs:

saga::task task = file.copy <saga::task::Async> ("b");
saga::job  job  = js.run_job ("remote.host.net", "/bin/date");

saga::task_container tc;

tc.add (task);
tc.add (job);

tc.wait (saga::task::All);
```

SAGA Task Container: Bulk Operations

```
saga::task_container tc;

tc.add (js.create_job (jd_1));
tc.add (js.create_job (jd_2));
tc.add (js.create_job (jd_3));
// ...
tc.add (js.create_job (jd_n));

tc.run ();

tc.wait (saga::task::All);
```

SAGA: also available

- monitoring
- notifications
- attributes
- exceptions

Questions?

CENTER FOR COMPUTATION
& TECHNOLOGY

SAGA

A Simple API for Grid Applications

Components of SAGA



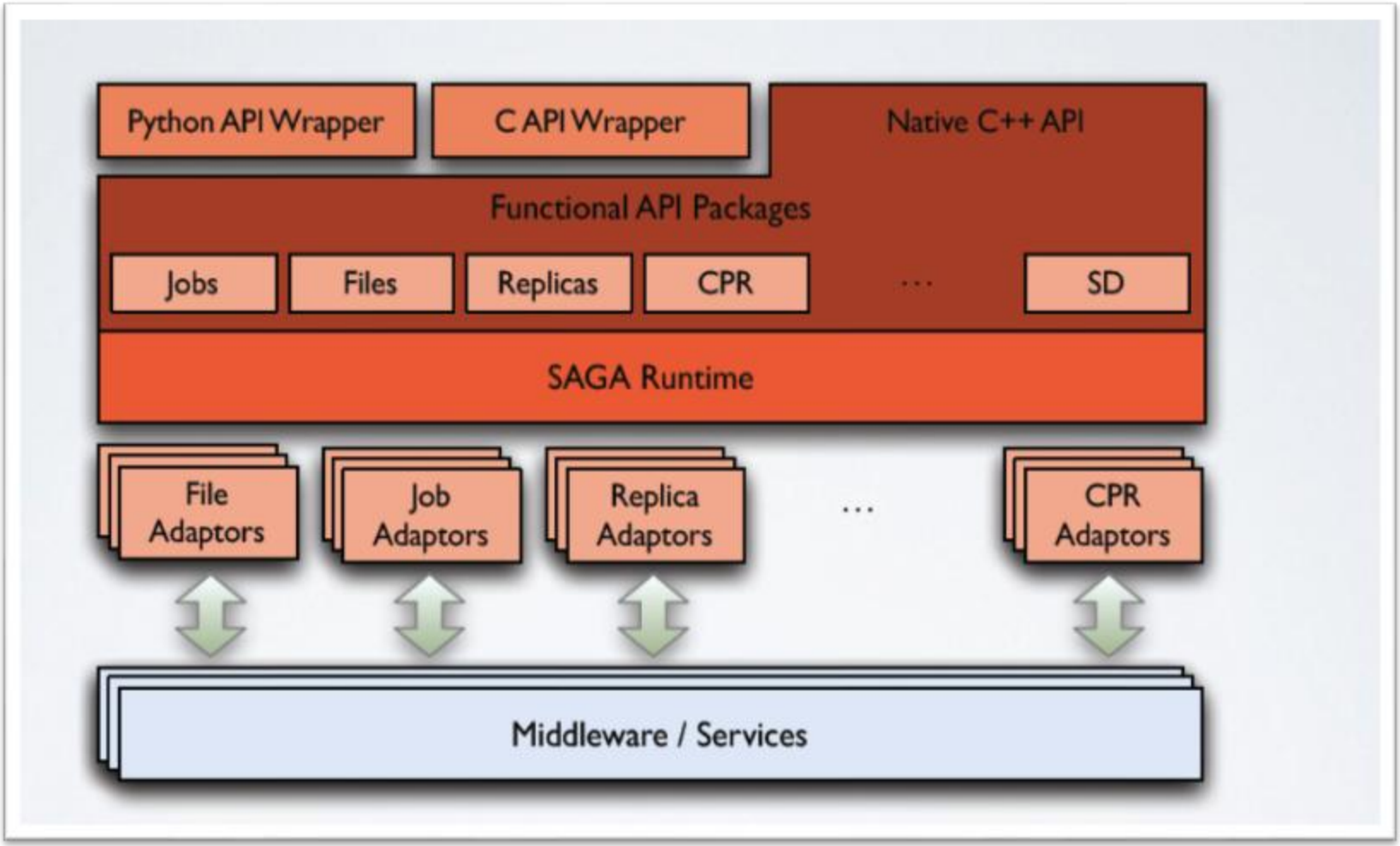
Outline

- SAGA command line tools
- SAGA Python API
- SAGA C++ API

Documentation

- General information
  - https://www.saga-project.org/
- API documentation
  - C++
    - http://static.saga.cct.lsu.edu/apidoc/cpp/latest/
  - Python
    - http://static.saga.cct.lsu.edu/apidoc/python/latest/
- Programmers manual
  - http://static.saga.cct.lsu.edu/docs/programming_guide/saga_programming_guide.pdf

SAGA: Architecture



Three Ways to Use SAGA

	Local Adaptors	Globus Adaptors	SSH Adaptors	…
	file://localhost/... any://localhost/...	gram://remotehost/... any://remotehost/…	ssh://remotehost/… any://remotehost/…	
C++	using saga::job::job; **job.run** (cmd);			
Python	import saga.job **js.run** (cmd)			
Shell	**saga-job run** host cmd			

Three Ways to Use SAGA

	Local Adaptors	Globus Adaptors	SSH Adaptors	…
	file://localhost/... any://localhost/...	gram://remotehost/... any://remotehost/…	ssh://remotehost/… any://remotehost/…	
C++	using saga::filesystem::directory; **dir.copy** (src, dest)			
Python	import saga.filesystem **dir.copy** (src, dest)			
Shell	**saga-file copy** src dest			

Command line tools

- SAGA comes with simple command line tools that allow to access basic package functionality.
- The source code is very simple and a great starting point to explore the SAGA package APIs:
  - saga-file $SAGA_ROOT/tools/clutils/filesystem/
  - saga-job $SAGA_ROOT/tools/clutils/job/
  - saga-advert $SAGA_ROOT/tools/clutils/advert/
  - saga-shell $SAGA_ROOT/tools/shell/

Command line tool: saga-file

- Supported protocols
  - Depends on SAGA adaptors
  - Also available: Globus GridFTP, Curl (subset), KFS, Amazon EC2, Opencloud (Sector/Sphere), Hadoop (HDFS)
- Supported commands:

Command	Arguments
copy	<url from> <url to>
move	<url from> <url to>
remove	<url>
cat	<url>
list_dir	<url>

Command line tool: saga-job

- Supported protocols
  - Depends on SAGA adaptors
  - Also available: Globus Gram, Condor, OMII-GridSAM, LSF, Amazon EC2, Opencloud (Sector/Sphere)
- Supported commands:

Command	Arguments
run	<rm url> <command> <arguments>
submit	<rm url> <command> <arguments>
state	<rm url> <jobid>
suspend	<rm url> <jobid>
resume	<rm url> <jobid>
cancel	<rm url> <jobid>

Command line tool: saga-advert

- What is it?
  - Central data store with
    - Hierachical keys
    - Attributes
  - Filesystem like structure
- Supported protocols
  - Depends on SAGA adaptors
  - Local adaptor:
    - Local backend: SQLite3
    - Remote backend: PostgreSQL
  - Also available: Hadoop H-Base, Hypertable

Command line tool: saga-advert

Command	Arguments
list_directory	<advert-url> <pattern>
add_directory remove_directory	<advert-url>
add_entry remove_entry	<advert-url>
store_string	<advert-url> <string>
retrieve_string	<advert-url>
list_attributes	<advert-url>
set_attribute	<advert-url> <key> <value>
remove_attribute	<advert-url> <key>

Command line tool: saga-shell

- All in one of all command line tools as mentioned earlier
- Keeps context in between commands
- Navigate (remote) filesystems (advert, replica too!)
- Launch (remote) jobs, uses io redirection to access in/out
- All commands are implemented using SAGA

Command line tool: saga-shell

Type	Commands
File system navigation	pwd, ls, mv, cp, cd, mkdir, rmdir, touch, cat
Job package	run, suspend, resume, kill, status, ps
replica	rep_find, rep_list, rep_add, rep_remove, rep_update, rep_replicate
environment	setenv, getenv, env
permissions	add_proxy, remove_proxy

Python API Example: File Package

- copy a file

```
import saga

src  = saga.url ("file://localhost/etc/passwd")
dst  = saga.url ("file://localhost/tmp/passwd-copy")

f    = saga.filesystem.file (src, saga.filesystem.Read)

f.copy (dst)
```

Python API Example: File Package

- get a directory file listing

```
import saga

src       = saga.url ("file://localhost/opt/")
d         = saga.filesystem.directory (src)
names     = d.list ('*')

for name in names:
    ns = saga.name_space.entry  (name)

    if    ns.is_dir ():   print 'd ',  name
    elif ns.is_link ():  print '->', ns.read_link ()
    else:                print '  ', name
```

Python API Example: Job Package #1

- submit a job

```
import saga

js_url          = saga.url ("fork://localhost/")
job_service     = saga.job.service (js_url)
job_desc        = saga.job.description ()

job_desc.executable = "/bin/touch"
job_desc.arguments = ["-a", "touche"]

my_job          = job_service.create_job (job_desc)

my_job.run ()
```

Python API Example: Advert Package

- Create and modify an advert entry

```
# host/process A
import saga
import time
name      = saga.url ("advert://localhost/myentry")
e         = saga.advert.entry (name, saga.advert.ReadWrite|saga.advert.Create)

e.set_attribute ("started", time.strftime ("%a, %d %b %Y %H:%M:%S +0000",
                            time.gmtime())

# host/process B
import saga
name      = saga.url ("advert://localhost/myentry")
e         = saga.advert.entry (name)
print "started: " + e.get_attribute ("started")
```

Additional Resources: Programmers Guide

- set of very small and easy examples, one for each package/paradigm
  - file_copy, file_copy (async)
  - error handling
  - attributes
  - stream (server/client)
- http://static.saga.cct.lsu.edu/docs/programming_guide/saga_programming_guide.pdf

Example 1: hello_world

- hello world
  - launch 3 jobs on different machines
    - Execute "/bin/echo"
  - no job dependency
  - each job returns its passed input argument
    - "Hello"
    - "distributed"
    - "world!"
  - results printed as soon as jobs finish

Example 1: hello_world

- hello world
  - prints "Hello distributed world"
  - demonstrates
    - How to launch a remote job using SAGA job_service
    - Pass arguments using the command line
    - Collect result by output redirection
- the source code can be found here (see 'Example1'):
  - https://svn.cct.lsu.edu/repos/saga/core/trunk/examples/tutorial
  - the example uses localhost to spawn children
    - for remote execution change HOST1, HOST2, HOST3 from "localhost" to 'something else' (e.g. ssh://…)

Example 2: chaining_jobs

- launch 3 jobs on 3 different machines
- output of previous job is needed to launch next job
- simple sequential execution, but SAGA style
- demonstrates
  - how to launch a job using SAGA job_service
  - how to feed input to launched job
  - how to collect output
- launched job: `/usr/bin/bc`
- increment the number passed as the argument
  - pass returned incremented number to next job

Example 3: depending_jobs

- coordinating information from advert service
- launch a single job sequentially on a set of remote resources
  - simulating checkpointing/relaunching on different resource (migration)
- maintain a single result value in advert service
  - Gets written by one job, and read by the next
- demonstrates
  - how to launch remote jobs while maintaining state
  - assembling argument lists
- result is left in advert service, but accessed afterwards

Integrated Demo: Mandelbrot

- http://cyder.cct.lsu.edu/saga-interop/mandlebrot/demo/

Questions | Comments ?

- We have covered:
  - SAGA command line tools
  - SAGA Python API
  - SAGA C++ API
  - Examples
- Check out the tutorial website for more details and examples:

  http://saga.cct.lsu.edu/software/cpp/documentation/tutorials/loni-training-2010