SAGA: Simple Examples, Programming Manual SAGA-Shell, Example Applications

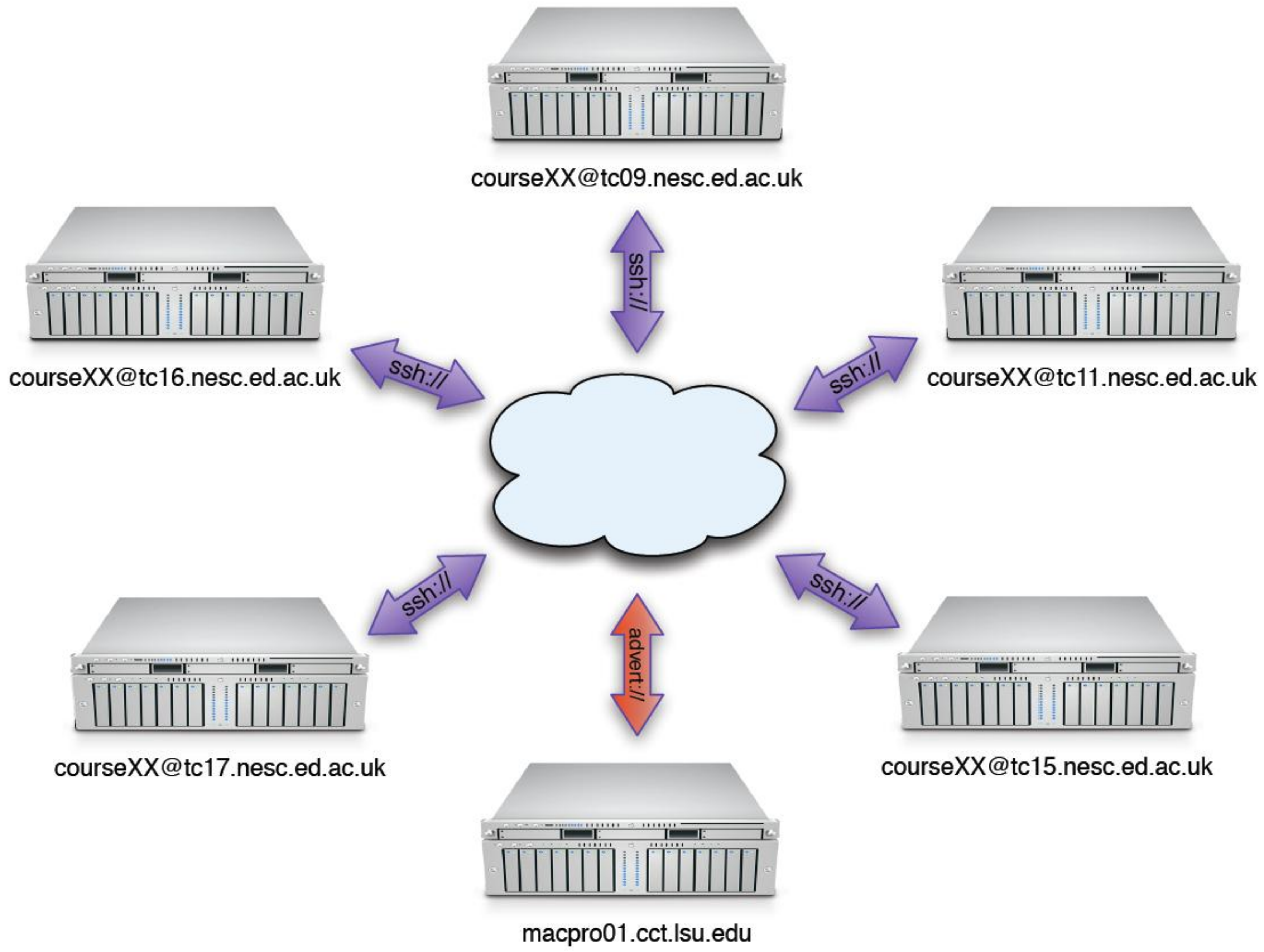
NeSC 2009, September 3rd/4th

Hartmut Kaiser, Shantenu Jha, Ole Weidner, Andre Merzky, Andre Luckow

Agenda

- Overview, resources, accounts, etc.
- SAGA command line tools
- Examples

Infrastructure


courseXX@tc09.nesc.ed.ac.uk
ssh://
courseXX@tc16.nesc.ed.ac.uk
ssh://
ssh://
courseXX@tc11.nesc.ed.ac.uk
ssh://
advert://
ssh://
courseXX@tc17.nesc.ed.ac.uk
courseXX@tc15.nesc.ed.ac.uk
macpro01.cct.lsu.edu

Login

- Login (ssh) to one of the following machines:
  - tc09.nesc.ed.ac.uk
  - tc11.nesc.ed.ac.uk
  - tc15.nesc.ed.ac.uk
  - tc16.nesc.ed.ac.uk
  - tc17.nesc.ed.ac.uk
- Credentials:
  - course01, course02,…, course12

Enter the SAGA world

```
source /usr/local/saga/share/saga/saga-env.sh
```

Make sure everything works:

```
saga-job run ssh://user@tcXX.nesc.ed.ac.uk /bin/hostname

saga-advert list_directory advert://FIXME/FIXME
```

Enter the SAGA world

- If something goes wrong:

```
setenv SAGA_VERBOSE = 0…6
```

- Will print logging information about adaptors, settings, API calls, etc.

Documentation

- General information
  - http://faust.cct.lsu.edu/trac/saga/wiki/NeSC2009

- API documentation
  - http://saga.cct.lsu.edu/cpp/apidoc/
- Programming manual
  - http://tinyurl.com/saga-manual

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/saga/tools/cltools/file/
saga-job	$SAGA_ROOT/saga/tools/cltools/job/
saga-advert	$SAGA_ROOT/saga/tools/cltools/advert/
saga-replica	$SAGA_ROOT/saga/tools/cltools/replica/
saga-shell	$SAGA_ROOT/saga/tools/shell/

Command line tools

- ‘Shell bindings’
  - Package specific (file, job, advert, replica)
- SAGA shell
  - All in one solution
  - Filesystem navigation (filesystem, advert, replica)
  - Job launching
  - Scripting

Command line tool: saga-file

- Supported protocols
  - Depends on SAGA adaptors
  - We will use ssh and local 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
  - We will use ssh and local 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
  - We will use local adaptor
    - Local backend: SQLite3
    - Remote backend: PostgreSQL
  - Also available: Hadoop H-Base, Hypertable

Command line tool: saga-advert

- Supported commands:

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-replica

- What is it?
  - Central data store allowing to amp logical file names to a set of physical files (i.e. different instances of same file on different machines)
    - Hierachical keys
    - Attributes
  - Filesystem like structure
- Supported protocols
  - Depends on SAGA adaptors
  - We will use local adaptor
    - Local backend: SQLite3
    - Remote backend: PostgreSQL
  - Also available: Globus RLS (subset)

Command line tool: saga-replica

- Supported commands:

Command	Arguments
list_directory	<lfn> <pattern>
add_directory remove_directory	<lfn>
add_lfn remove_lfn list_pfns	<lfn>
add_pfn	<lfn> <pfn>
remove_pfn	<lfn> <pfn>
list_attributes	<lfn>
set_attribute	<lfn> <key> <value>
remove_attribute	<lfn> <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

- Some of the supported commands

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

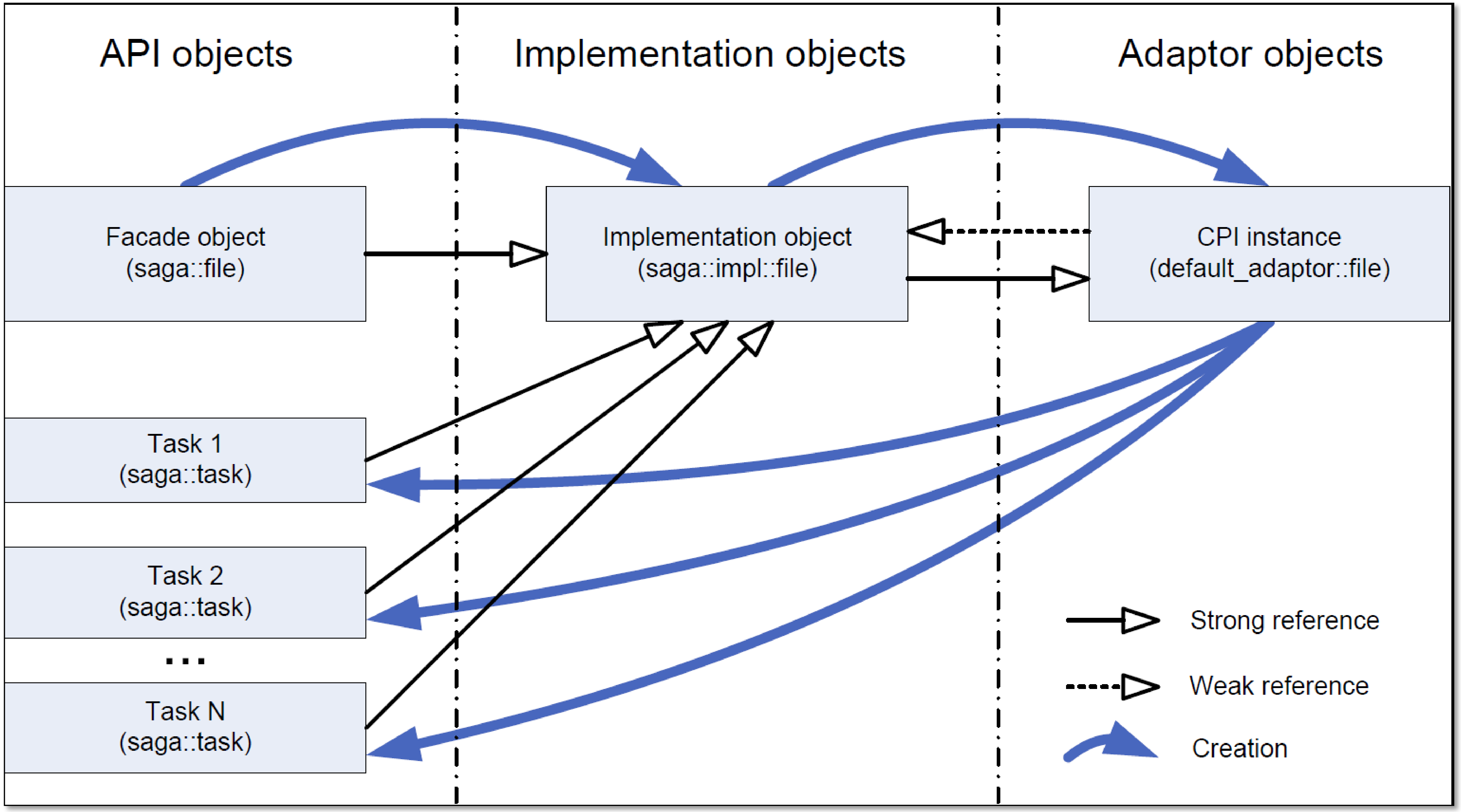
Hands on

- Try and run command line tools
- Copy a file, move it, delete it
- Run a job (/bin/sleep 10), monitor its status
- Play with advert service, create, directories, entries, store date, attributes

General Guidelines

- Pimpl paradigm, shared_ptr
- Sync/async API’s
- Task container
- Error handling

Pimpl paradigm, shared_ptr



Pimpl paradigm, shared_ptr

- All SAGA API objects are very lightweight (except saga::url)
  - Cheap to copy, passed as arguments, or stored in containers

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

void copy(saga::filesystem::file f, saga::url const& target)
{
    f.copy(target);
}

int main(int argc, char* argv[])
{
    saga::url source("ssh://tc17/etc/passwd");
    saga::url target(".");
    saga::filesystem::file f(source, saga::filesystem::Read);

    copy(f, target);
    return 0;
}
```

Sync/Async API's

- Almost all API objects expose 4 different sets of API functions:
  - Synchronous
    - Returns result synchronously
    - saga::off_t file::get_size();
  - Task based
    - Returns handle to deferred result (using a task object)
    - Asynchronous
      - saga::task not running yet
      - saga::task file::get_size<saga::task::Async>();
    - Task
      - saga::task already running
      - saga::task file::get_size<saga::task::Task>();
    - Synchronous
      - saga::task guaranteed to be finished
      - saga::task file::get_size<saga::task::Sync>();

Sync/Async API's

- Synchronous

```
saga::url source("ssh://tc17/etc/passwd");
saga::filesystem::file f(source, saga::filesystem::Read);

saga::off_t size = f.get_size();
```

- Asynchronous

```
saga::url source("ssh://tc17/etc/passwd");
saga::filesystem::file f(source, saga::filesystem::Read);

saga::task t = f.get_size<saga::task::ASync>();
t.run();

saga::off_t size = t.get_result<saga::off_t>();
```

Sync/Async API's

- Asynchronous creation of objects
- Factory functions

```
// create task
saga::task t =
    saga::filesystem::file::create<saga::task::ASync>("fileurl");

// ...
saga::filesystem::file f =
    t.get_result<saga::filesystem::file>();
```

Task container

- Special container allowing to handle many tasks as one

```
// create a list of tasks to run jobs
saga::job::service js("somehost");
saga::task_container tc;
for (int i = 0; i < num; ++i)
{
    tc.add_task(js.run_job<Task>(execs[i], hosts[i]));
}

// execute tasks
tc.run();

// wait for any of the tasks to finish
std::vector<saga::task> finished =
    tc.wait(saga::task_container::Any));
```

Error Handling

- Error handling is currently purely exception based

```
try {
    saga::filesystem:file f("non-existing file");
    // …
}
catch (saga::exception const& e) {
    std::cerr << e.what() << std::endl;
}
```

- Also:

```
e.get_error();               // error code
e.get_message();             // top level message
e.get_object();              // get failing API object
e.get_all_exceptions();      // get list of exceptions
e.get_all_messages();        // get list of messages
```

A Simple SAGA Application

- Simple file copy example: copy.cpp

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

int main(int argc, char* argv[])
{
    saga::url source("ssh://tc17/etc/passwd");
    saga::url target(".");
    saga::filesystem::file f(source, saga::filesystem::Read);

    f.copy(target);
    return 0;
}
```

Compiling and Linking a SAGA Application

- Simple file copy example: Makefile

```
SAGA_SRC = $(wildcard *.cpp)
SAGA_ADD_BIN_OBJ = $(SAGA_SRC:%.cpp=%.o)
SAGA_BIN = copy

include $(SAGA_LOCATION)/share/saga/make/saga.application.mk
```

- Use saga-config

```
g++ -Wall `saga-config --cxxflags` `saga-config --lflags` copy.cpp
```

- Or do it the hard way:

```
g++ -Wall –I$SAGA_LOCATION/include –pthread \
-L$SAGA_LOCATION/lib \
-lsaga_engine –lsaga_package_file copy.cpp
```

Running a SAGA Application

- Make sure that the SAGA libraries can be found by the loader.
- Use: /usr/local/saga/share/saga/saga-env.sh
- If something goes wrong - use SAGA_VERBOSE:

```
SAGA_VERBOSE=6 ./copy
```

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)

Hands on

- Try compiling and running other examples
  - Urls
  - Packages: file, job, replica
  - Modify stream server/client (see next slide)

Hands on

- Modify stream_server and stream_client to establish connection using advert service
- Server:

```
saga::stream::server s("tcp://tc17");
saga::advert::entry e
    ("advert://macpro01.cct.lsu.edu/NeSC2009/exercise_3/<uid>");
e.store_object(s);
...
```

- Client:

```
saga::advert::entry e
    ("advert://macpro01.cct.lsu.edu/NeSC2009/exercise_3/<uid>");
saga::stream::stream c (e.retrieve_object());
c.connect();
...
```

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!"
  - Jobs are launched in parallel (in separate threads)
  - As soon as result is collected it's printed on local console

Example 1: hello_world

- Hello world
  - Arbitrary sequence of results
    - Optimally: "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’):
  - http://faust.cct.lsu.edu/trac/saga/wiki/NeSC2009
  - The example uses localhost to spawn childs
  - For remote execution change HOST1, HOST2, HOST3 from "localhost" to "[tc11, tc15, tc16, or tc17].nesc.ed.ac.uk"

Hands on

- Compile and run example locally
- Modify the code to run it remotely
- Compile and run example remotely
- Run other remote executables
- Modify hello_world to produce consistent results
  - “Hello distributed world!”
- …

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 2: chaining_jobs

- Pass input
  - Command line (same as before)
  - Stage in of input files
    - Using job package API
    - Using file package API
  - Input redirection
  - Using Stream API
  - Using Advert service
- Collect output
  - Output redirection
  - Stage out of output files
    - Using job package API
    - Using file package API
  - Using Stream API
  - Using Advert service

Hands on

- Coordinate execution sequence of multiple jobs
  - 3 jobs are running, but wait for a flag in advert service to be set by previous job to continue

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 job using SAGA job, while maintaining environment
  - Assembling argument lists
- Result is left in advert service, but accessed afterwards

Thanks! Questions?

http://saga.cct.lsu.edu