



# An Introduction to the Simple API for Grid Applications (SAGA)

**Thilo Kielmann**

**VU University, Amsterdam**

[kielmann@cs.vu.nl](mailto:kielmann@cs.vu.nl)





- The Simple API for Grid Applications (SAGA)
  - Motivation & Scope
  - SAGA as an OGF Standard
- The SAGA Landscape
  - Interfaces
  - Language Bindings
- SAGA Implementations
  - Engine with Adaptors
  - C++, Java, Python

# SAGA in a nutshell



- **A programming interface for grid applications**
    - provides common grid functionality
    - simple (80/20 rule, limited in scope)
    - integrated (“consistent”)
    - stable: does not change (incompatibly)
    - uniform, across middleware platforms
    - high level, what applications need
-

# Copy a File: Globus GASS



VU university *amsterdam*

```
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 middleware system
- Like MapReduce – leave details of distribution *etc.* out

# What SAGA is and is not



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



- 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
  - Quick skim through the Requirements document
  - Design and requirements derived from 23 use cases
  - Different projects, applications and functionality



SourceForge : Project Home - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://forge.ogf.org/sf/projects/saga-rg


GridForge Home Projects Search

Shortcut: home User: kielmann Password: \*\*\*\*\* Log In Help

Project Home Tracker Documents Tasks Source Code Discussions File Releases Wiki Project Admin OGFCalendar CalTest

Project: SAGA-RG Project Home

**Project Home**



**SAGA-RG**

Simple API for Grid Applications RG

Project Created: 05/19/2004

[Project News](#) (0 Items)

**Project Members**

Total Project Members: 22

Project Administrators:

- [Andre Merzky](#)
- [Ole Weidner](#)
- [Pascal Kleijer](#)
- [Shantenu Jha](#)
- [Thilo Kielmann](#)
- [Tom Goodale](#)

**Project News:**

There are no News Items.

**Project Home Page**

**SAGA Documents and Specifications:**

- [SAGA UseCase Document](#)
- [SAGA Requirement Specification](#)
- [SAGA Core API Specification](#)
- [SAGA Core API Specification Errata \(Wiki\)](#)

**SAGA Implementations:**

- SAGA Java Implementation at EPCC/DEISA:** <http://deisa-ira7.forge.nesc.ac.uk/> (partial SAGA implementation, Files and Jobs)
- SAGA C++ Reference Implementaion:** <http://saga.cct.lsu.edu/>
- SAGA Java Reference Implementation:** <http://saga.cct.lsu.edu/>

**SAGA SVN:**

- Anonymous access is via:
- `cvs -d :pserver:cvs_anon@cvs.cct.lsu.edu:/projects login`
- with the password 'anon', followed by:
- `cvs -d :pserver:cvs_anon@cvs.cct.lsu.edu:/projects co SAGA-RG`
- If you need write access to the CVS, please contact [Andre Merzky](#)

POWERED BY Dell Hosted & Managed By LEXTECH

[Contact Webmaster](#) | [Report a problem](#) | [GridForge Help](#)

# SAGA API: Design & Specification



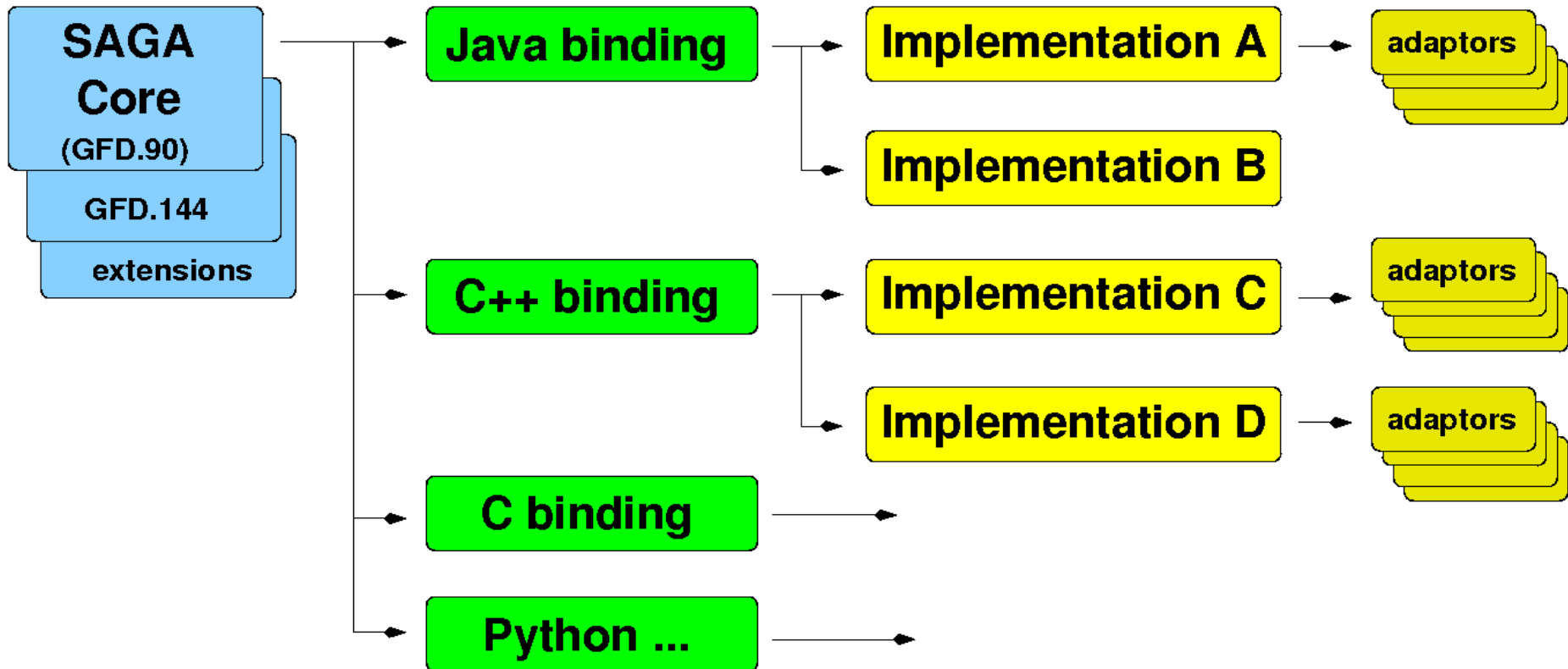
- Input SAGA Design Team (OGF, Berkeley, VU, LSU)
- Priority and partition
  - Functional Areas: Job Mgmt, Resource Mgmt, Data Mgmt, Logical Files, Streams....
  - Non-functional Areas: Asynchronous, QoS, Bulk
- Object-oriented; each sub-system is independent
- Interface is language independent; specified using Scientific Interface Description Language (SIDL)
  - Easy to map into specific language
  - Extensible and easily implementable
  - Scope for language specific features in binding

# Outline



- The Simple API for Grid Applications (SAGA)
  - Motivation & Scope
  - SAGA as an OGF Standard
- The SAGA Landscape
  - Interfaces
  - Language Bindings
- SAGA Implementations
  - Engine with Adaptors
  - C++, Java, Python

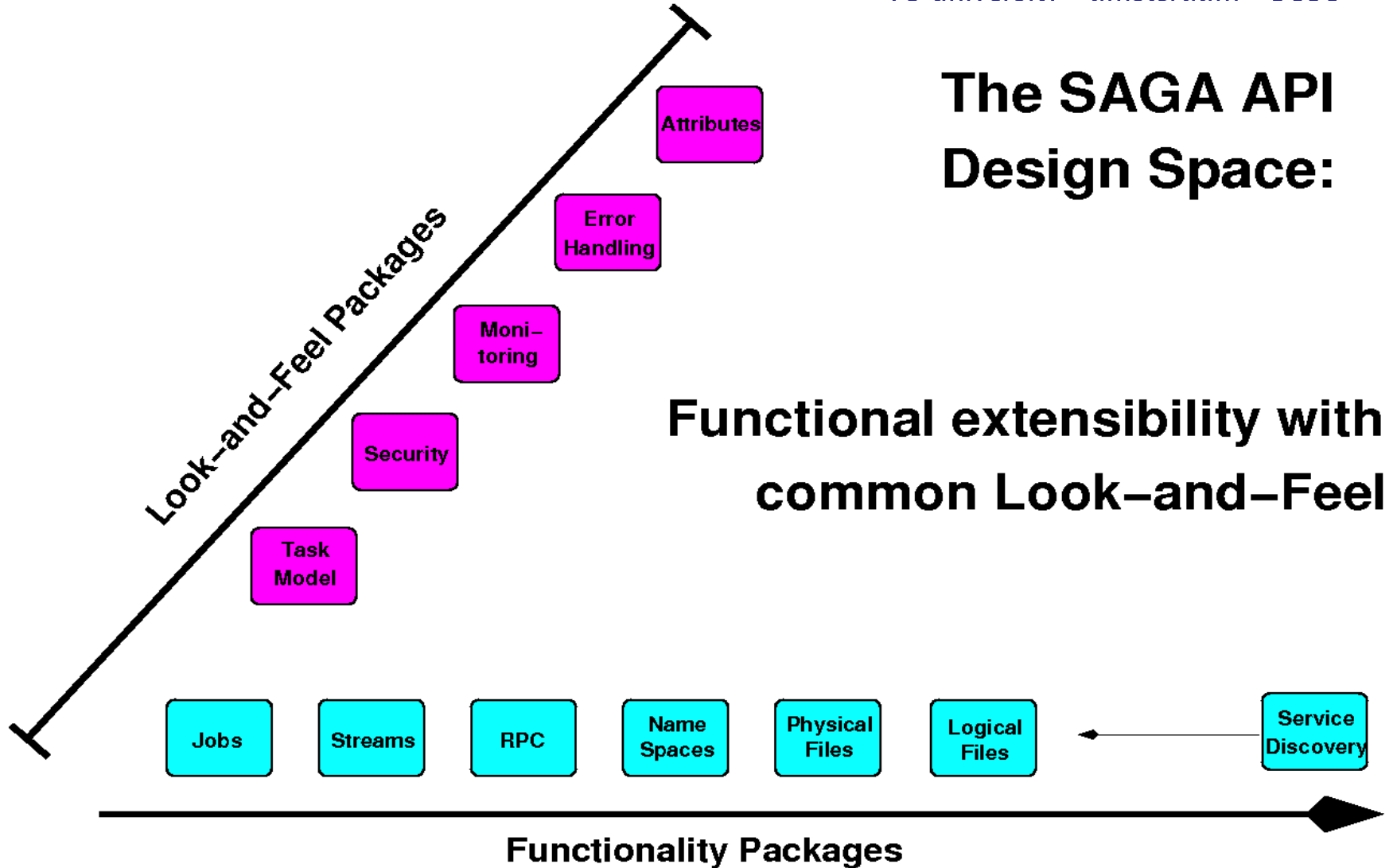
# The SAGA Landscape



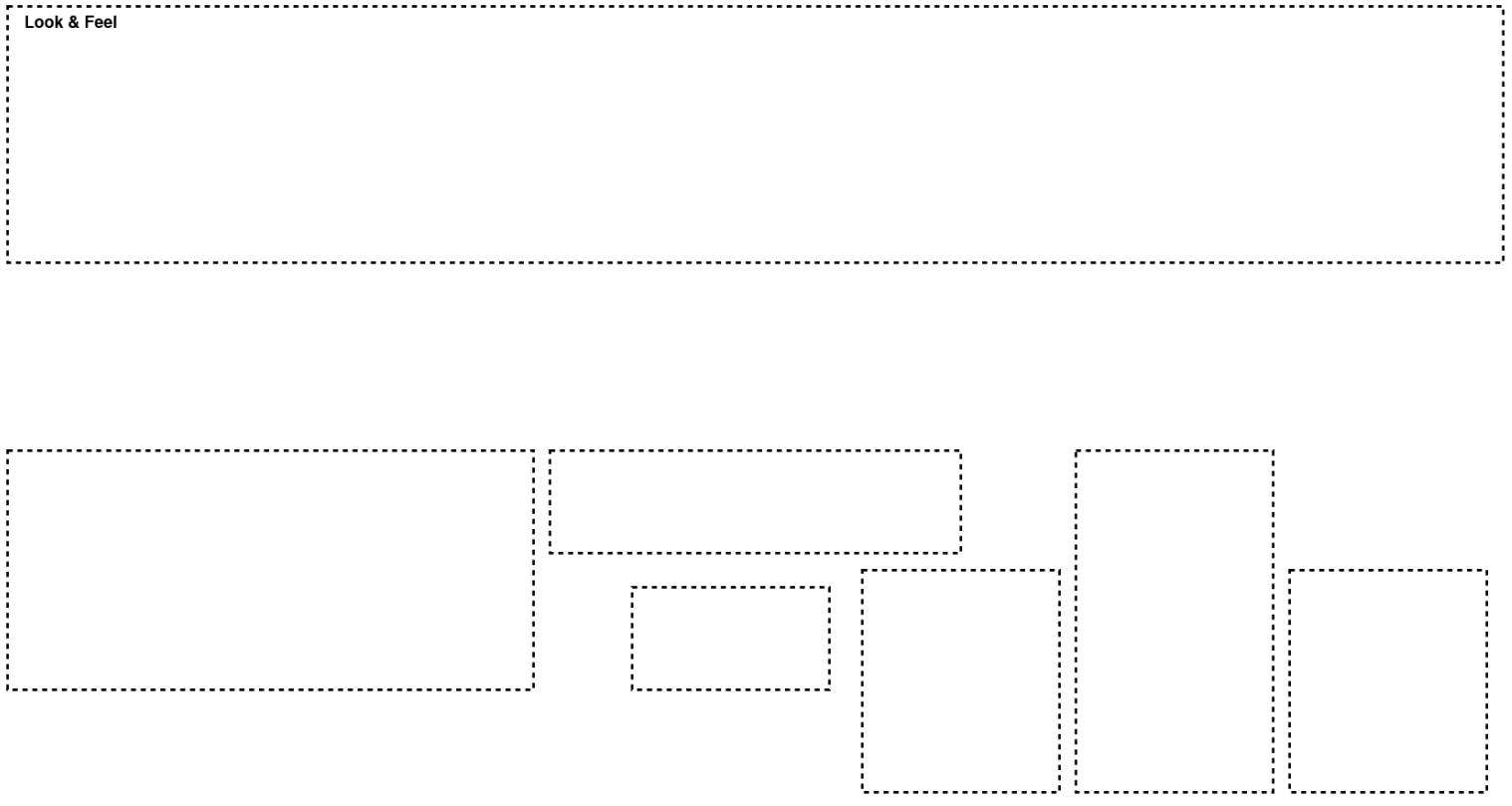
# SAGA API Design Overview



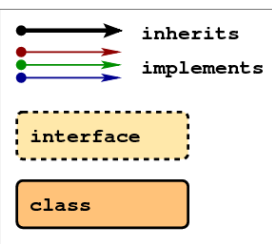
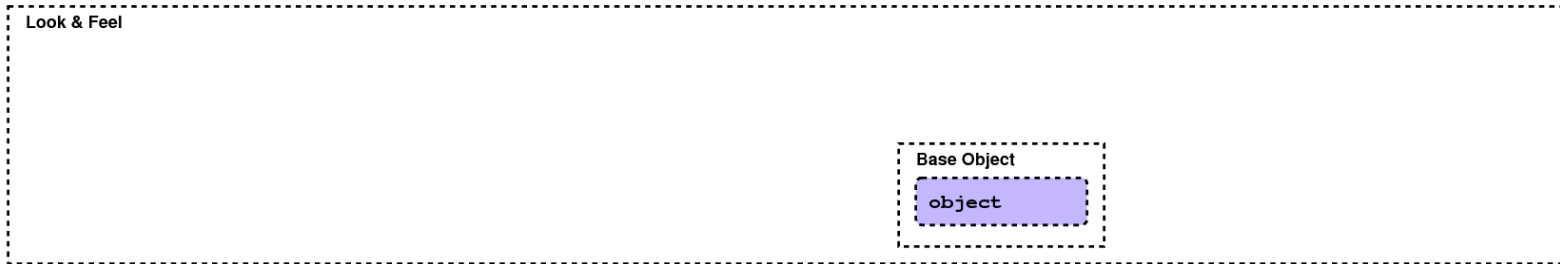
## The SAGA API Design Space:



# SAGA Interface Hierarchy

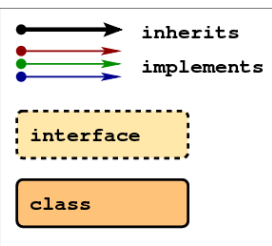
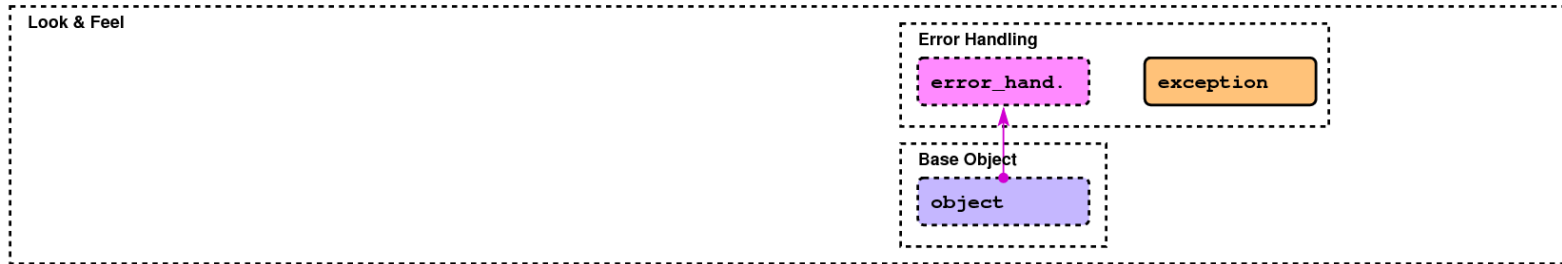


**Look and feel:** Top level Interfaces; Core SAGA objects needed by other API packages that provide specific functionality -- capability providing packages e.g., jobs, files, streams, namespaces etc.

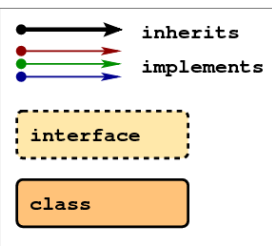
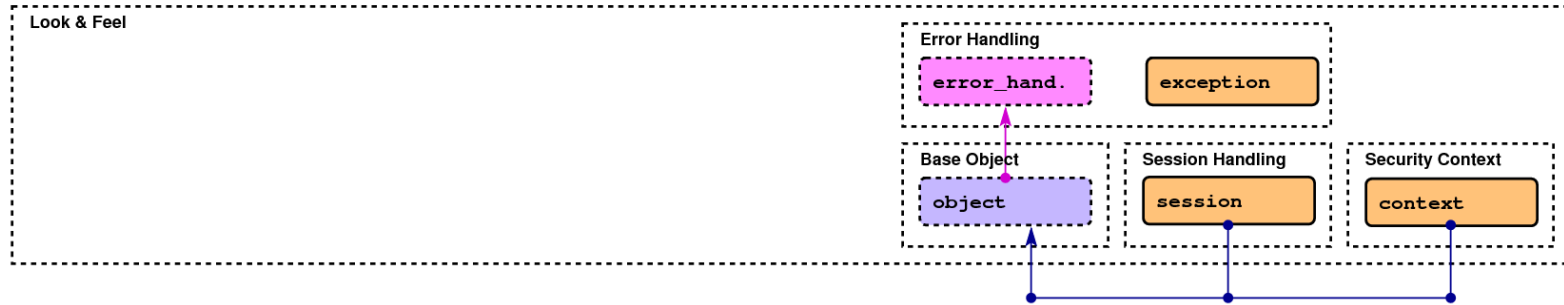


The common root for all SAGA classes.  
Provides unique ID to maintain a list of SAGA objects. Provides methods (get\_id()) essential for all SAGA objects

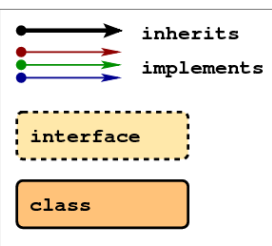
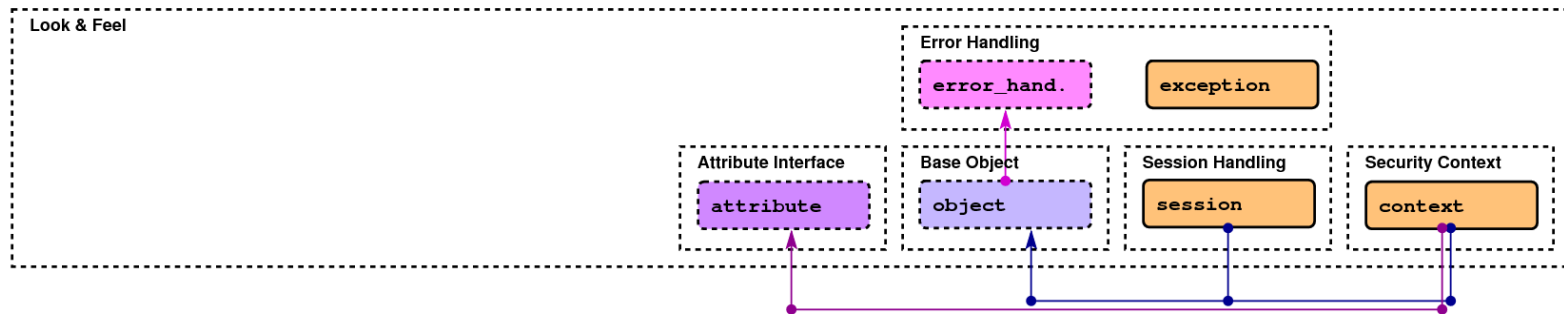
# Errors and Exceptions



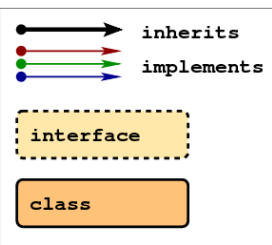
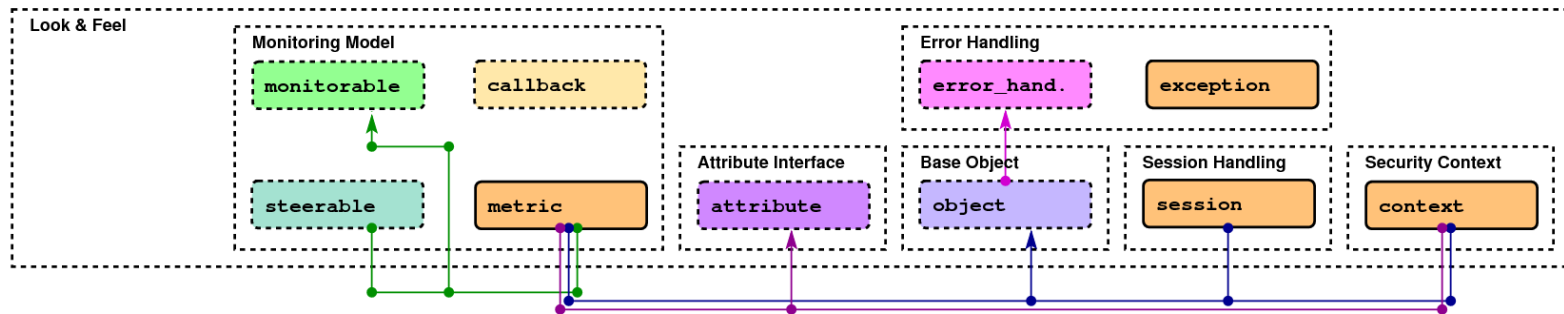
SAGA defines a hierarchy of exceptions  
(and allows implementations to fill in specific details)



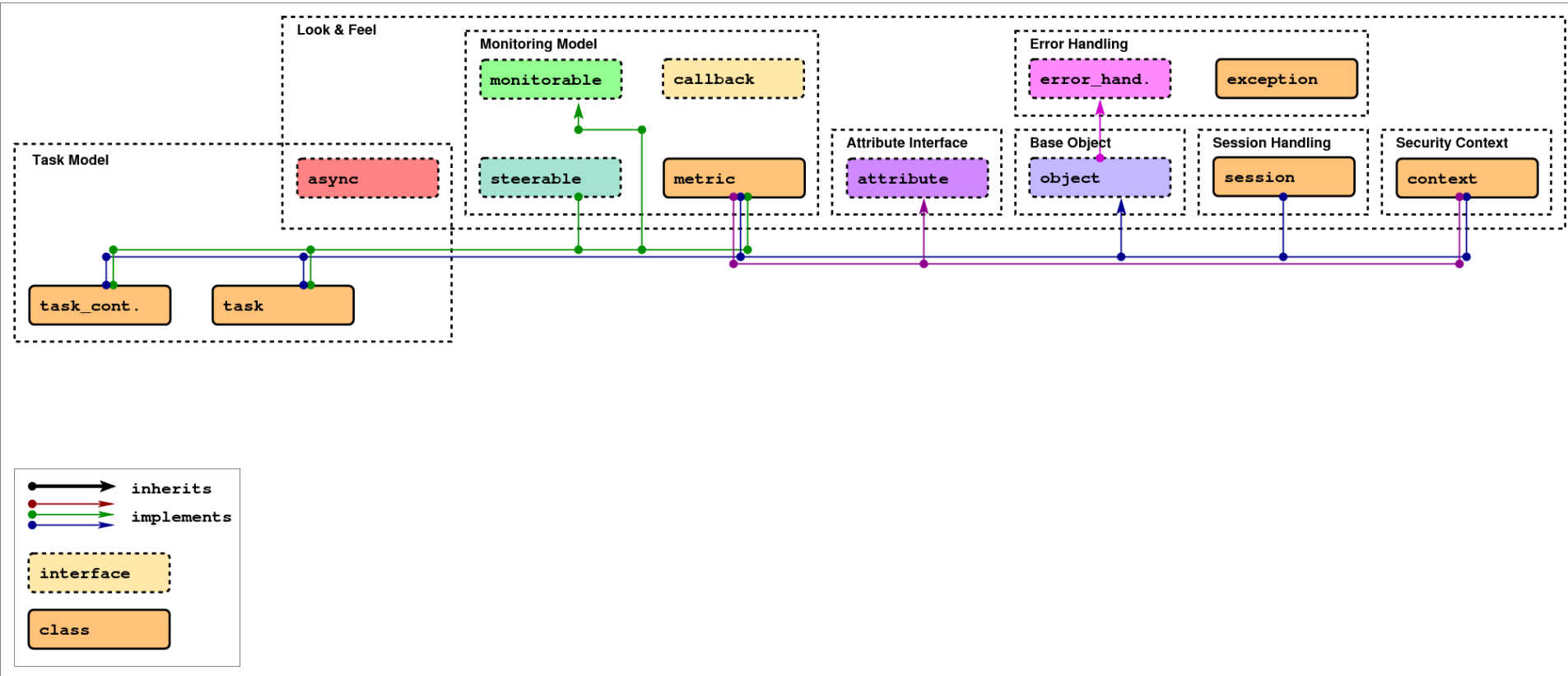
Context provides functionality of a session handle and isolates independent sets of SAGA objects. Only needed if you wish to handle multiple credentials. Otherwise *default context* is used.



Where attributes need to be associated with objects, e.g. Job-submission. Key-value pairs, e.g. for resource descriptions attached to the object.



**Metric** defines application-level data structure(s) that can be monitored and modified (steered). Also, task model requires state monitoring.



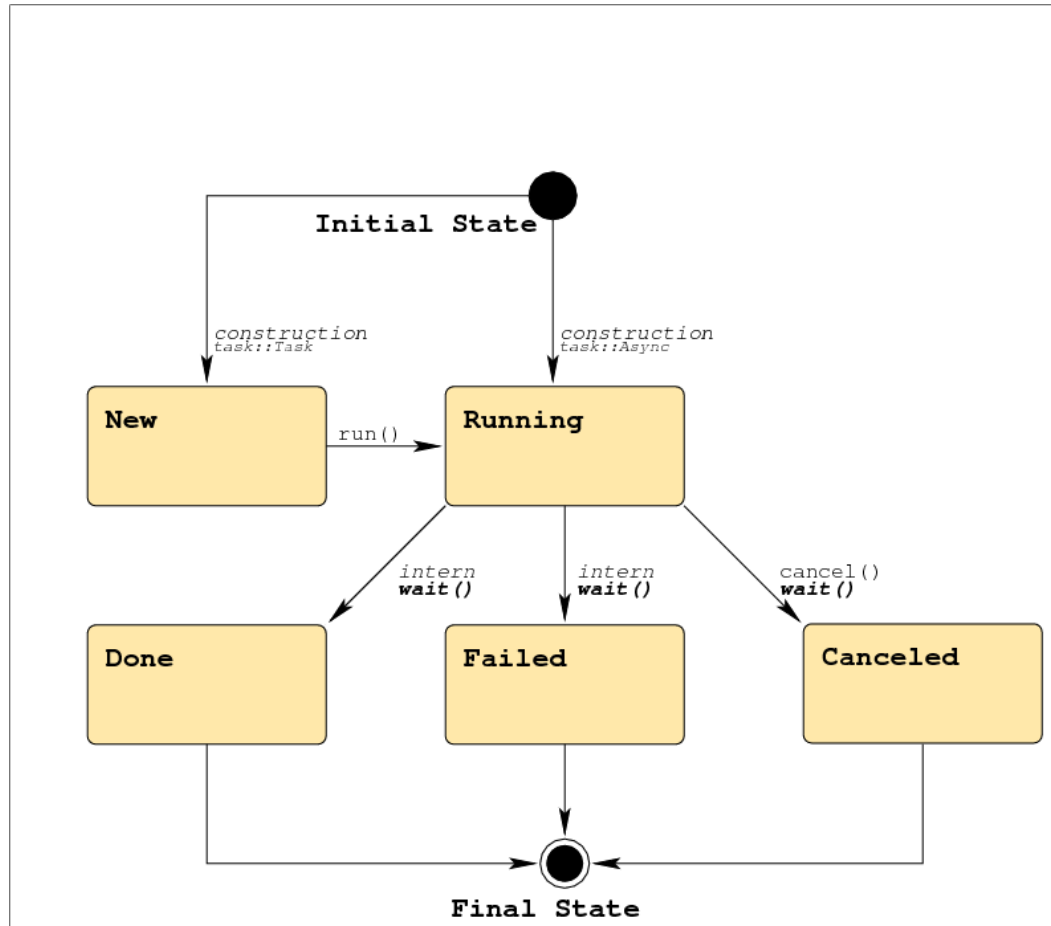
Most calls can be synchronous, asynchronous, or tasks (need explicit start.)

# SAGA Task Model



- All SAGA objects implement the task model
- Every method has three “flavours”
  - synchronous version - the implementation
  - asynchronous version - synchronous version wrapped in a task (thread) and started
  - task version - synchronous version wrapped in a task but not started (task handle returned)
- Adaptor can implement own async. version

# SAGA Task Model



# SAGA Task Model



```
file f ("any://host.net//data/src.dat");

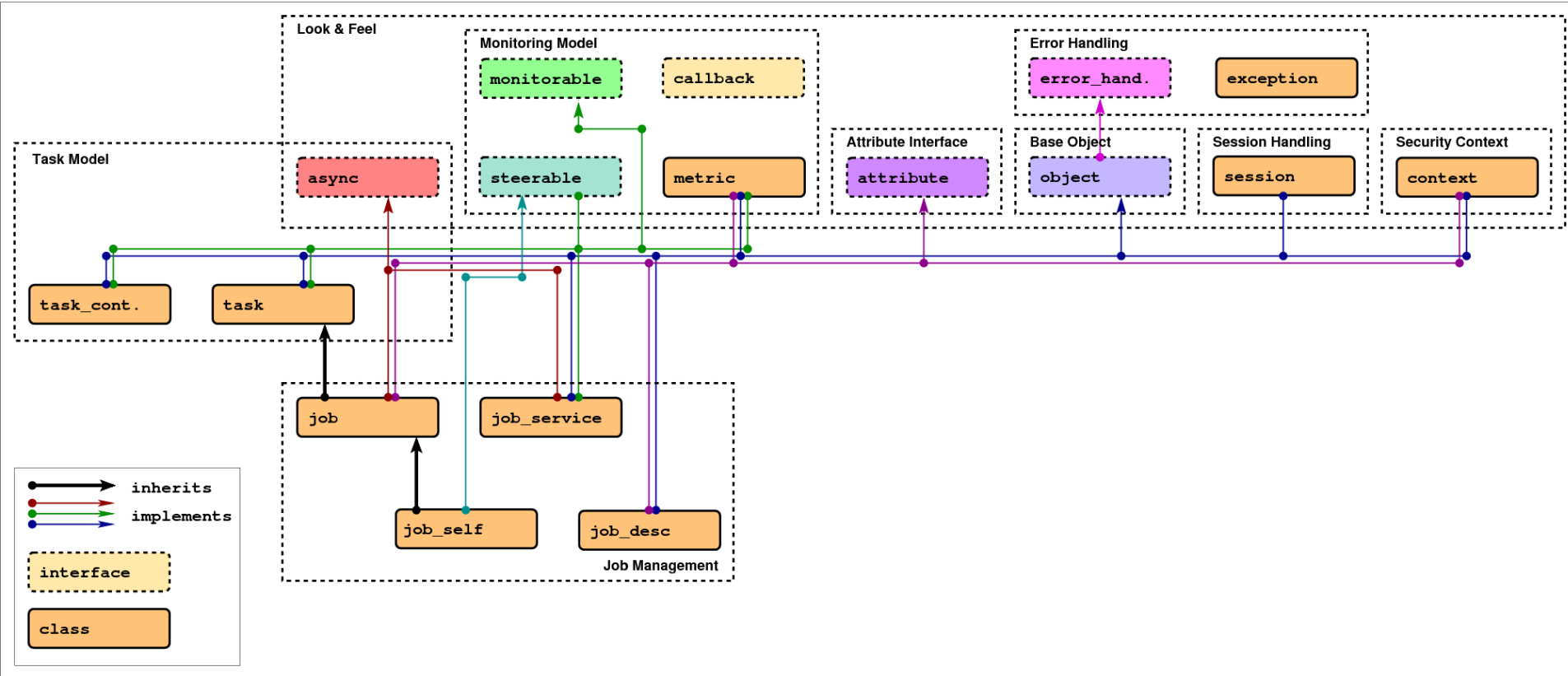
// normal sync version of the copy method
f.copy ("any://host.net//data/dest1.dat");

// the three task versions of the same method
task t1 = f.copy <task::Sync> ("any://host.net//data/dest2.dat");
task t2 = f.copy <task::ASync> ("any://host.net//data/dest3.dat");
task t3 = f.copy <task::Task> ("any://host.net//data/dest4.dat");

// task states of the returned saga::task
// t1 is in 'Finished' or 'Failed' state
// t2 is in 'Running' state
// t3 is in 'New' state

t3.run ();

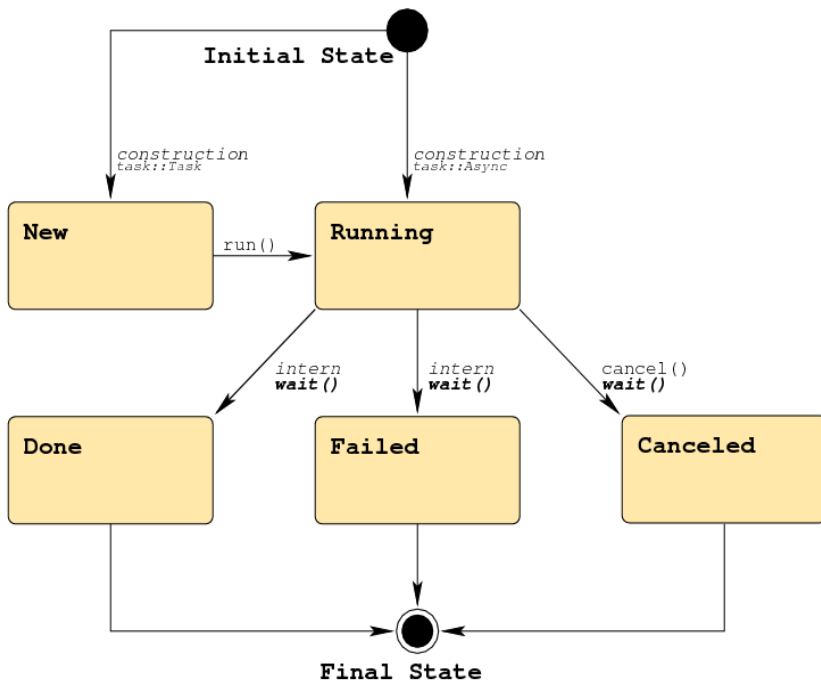
t2.wait ();
t3.wait ();
```



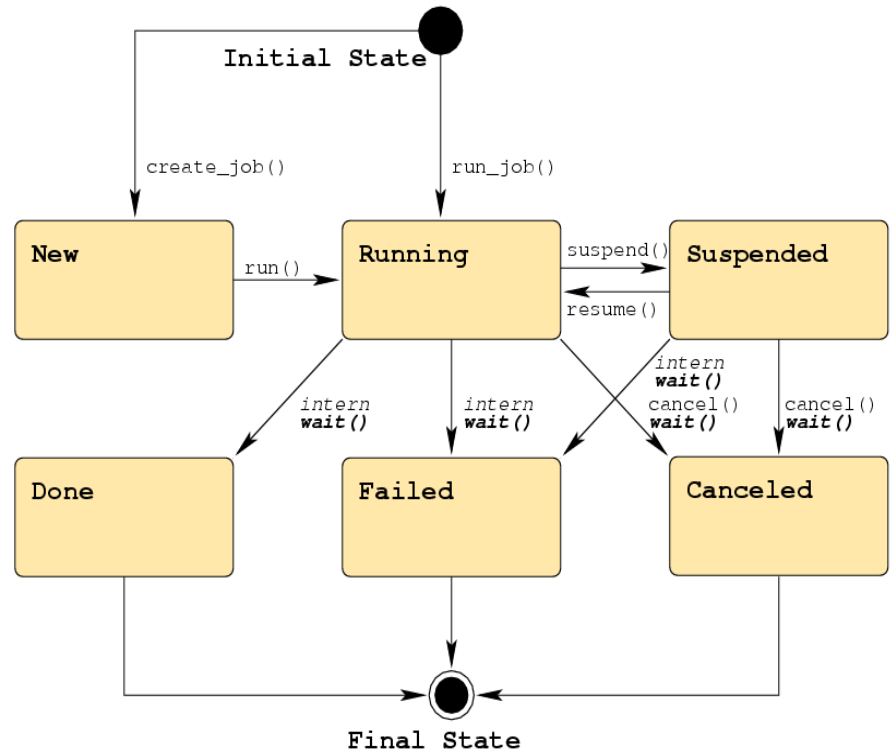
Jobs are submitted to run somewhere in the grid.



## Tasks:



## Jobs:



# Job Submission API



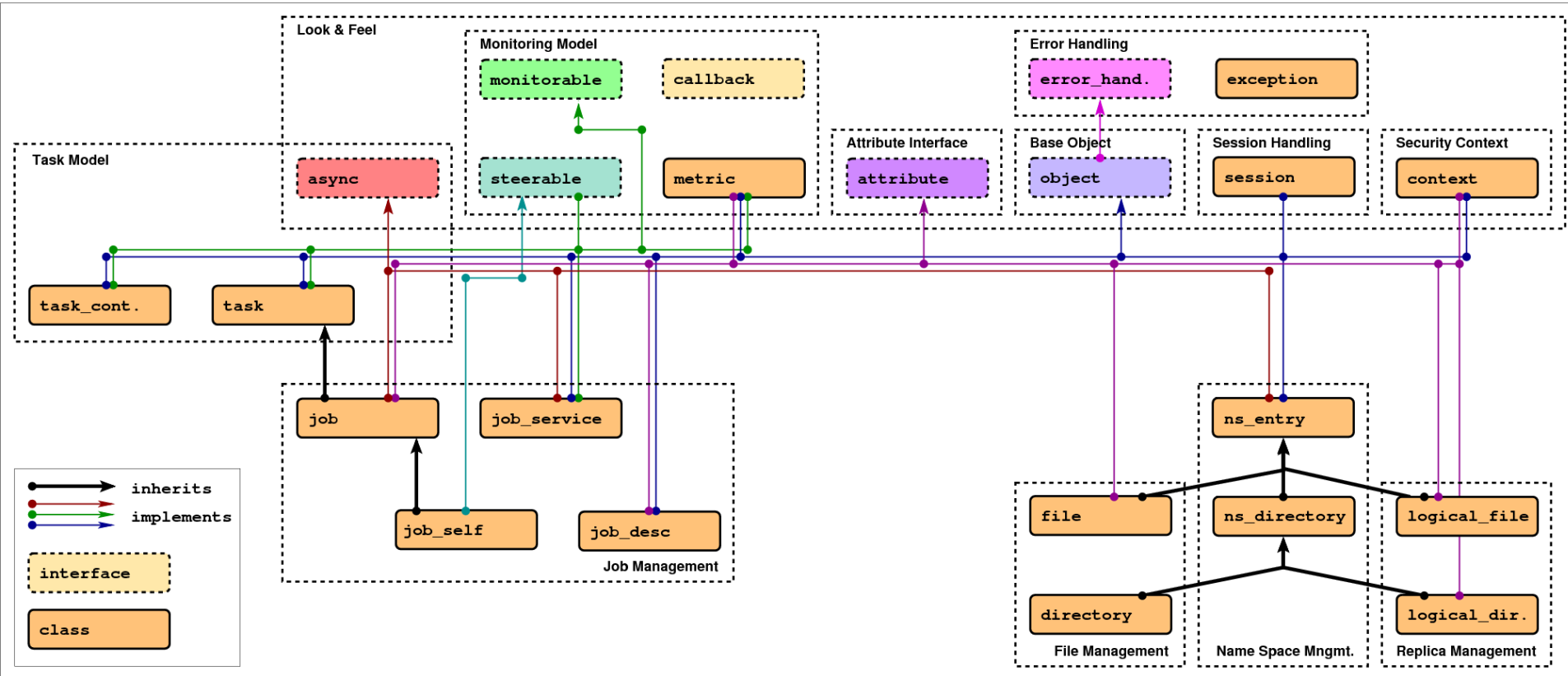
```
01: // Submitting a simple job and wait for completion
02: //
03: saga::job_description jobdef;
04: jobdef.set_attribute ("Executable", "job.sh");
05:
06: saga::job_service js;
07: saga::job job = js.create_job ("remote.host.net", jobdef);
08:
09: job.run();
10:
11: while( job.get_state() == saga::job::Running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

# Job Submission API



`job_service` uses `job_description` to create a `job`

- `job_description` attributes are based on JSDL [OGF, GFD.56]
- State model is based on OGSA BES [OGF, GFD.108]
- `job_self` represents the SAGA application



Both for physical and replicated (“*logical*”) files

# File API Example

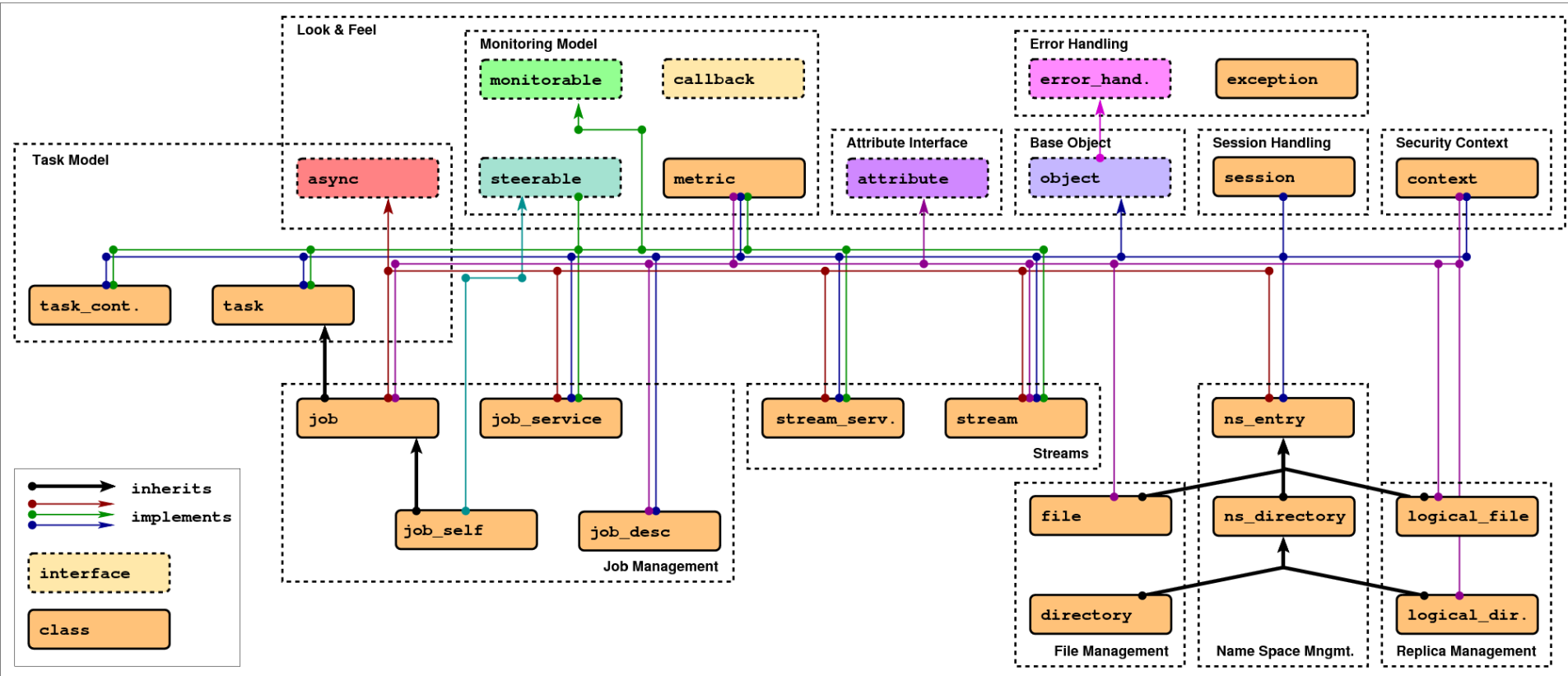


```
01: // Read the first 10 bytes of a file if file size > 10 bytes
02: //
03: saga::file my_file ("gridftp://gridhub/~/result.dat");
04:
05: off_t size = my_file.get_size ();
06:
07: if ( size > 10 )
08: {
09:     char buffer[11];
10:     long buflen;
11:
12:     my_file.read (10, buffer, &buflen);
13:
14:     if ( buflen == 10 )
15:     {
16:         std::cout << buffer << std::endl;
17:     }
18: }
```

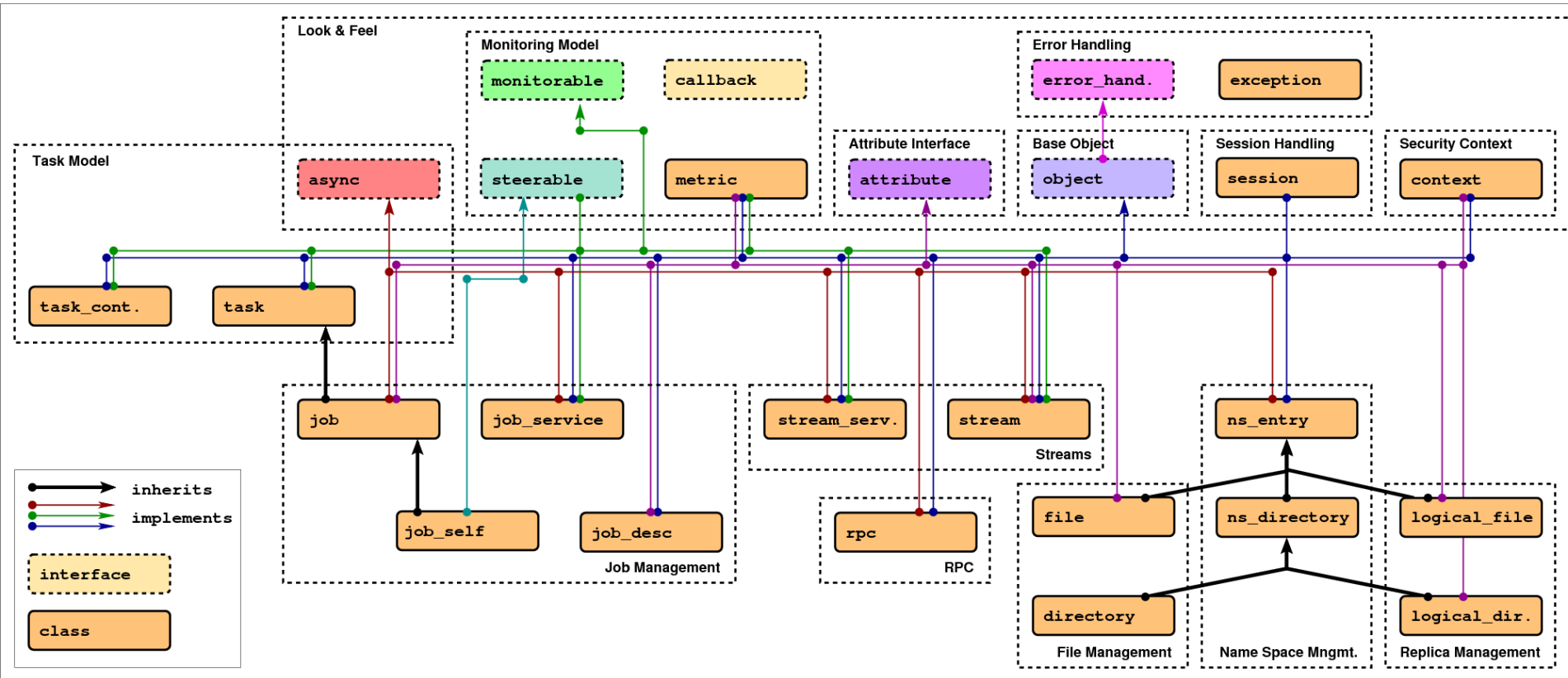
# saga\_file\_read.cpp



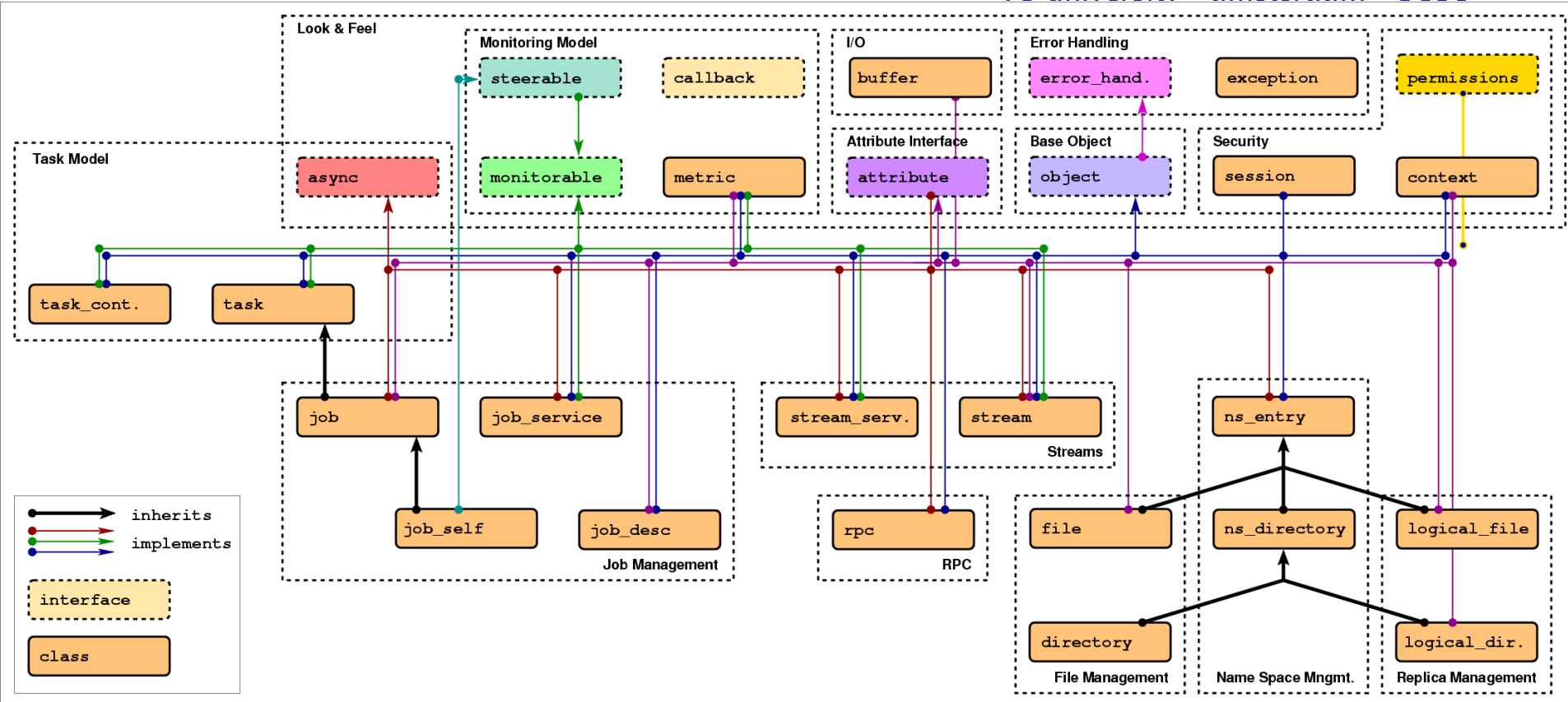
```
01: #include <string>
02: #include <saga/saga.hpp>
03:
04: int main( int argc, char* argv[] )
05: {
06:     if ( argc < 2 ) {
07:         std::cout << "\nUsage: " << argv[0] << " [URL] \n" << std::endl;
08:     }
09:     else {
10:         saga::size_t readbytes = 0;
11:         char inbuff[64];
12:
13:         saga::file f (argv[1], saga::file::Read);
14:
15:         while (readbytes = f.read (saga::buffer(inbuff)))
16:             std::cout << std::string(inbuff,readbytes) << std::flush;
17:     }
18:     return 0;
19: }
```



Simple, data streaming end points.



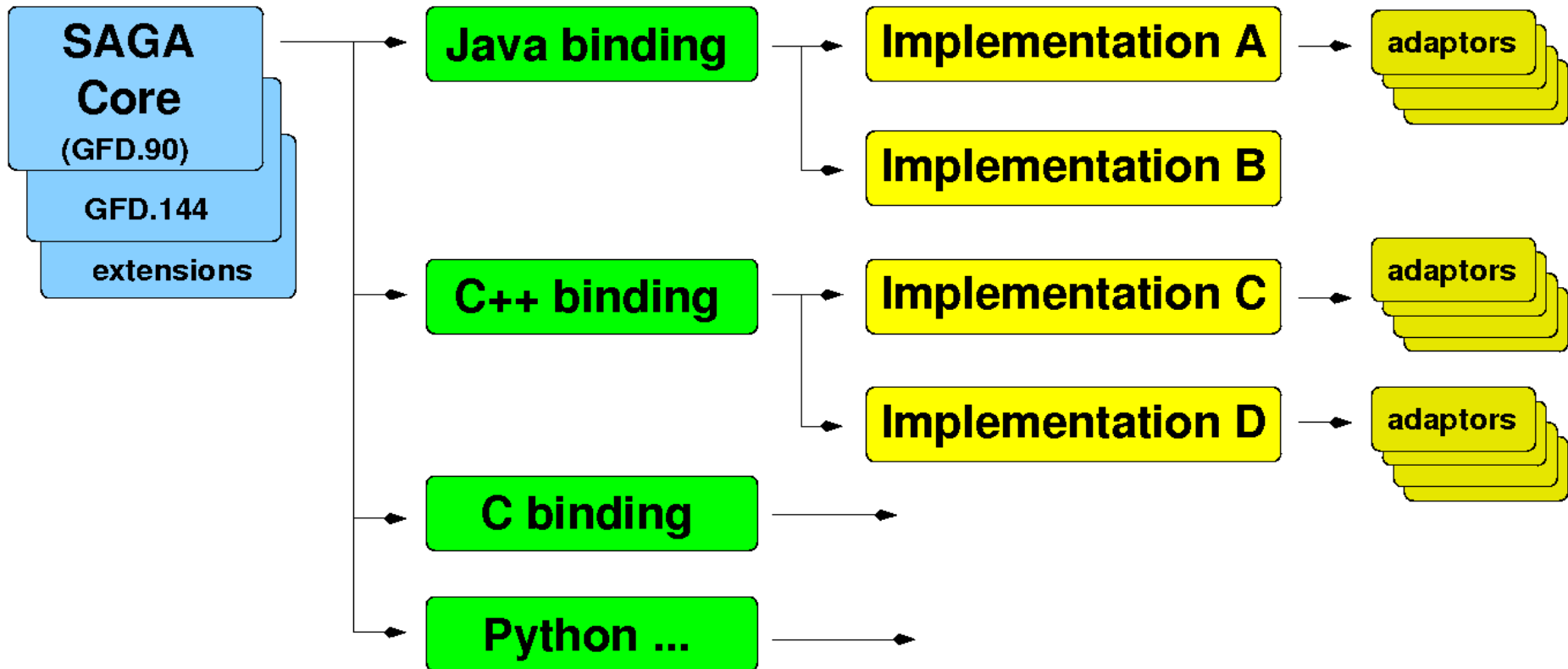
A rendering of GridRPC [OGF, GFD.052]



Permissions for access rights

Buffers for I/O operations

# The SAGA Landscape



# SAGA Language Bindings



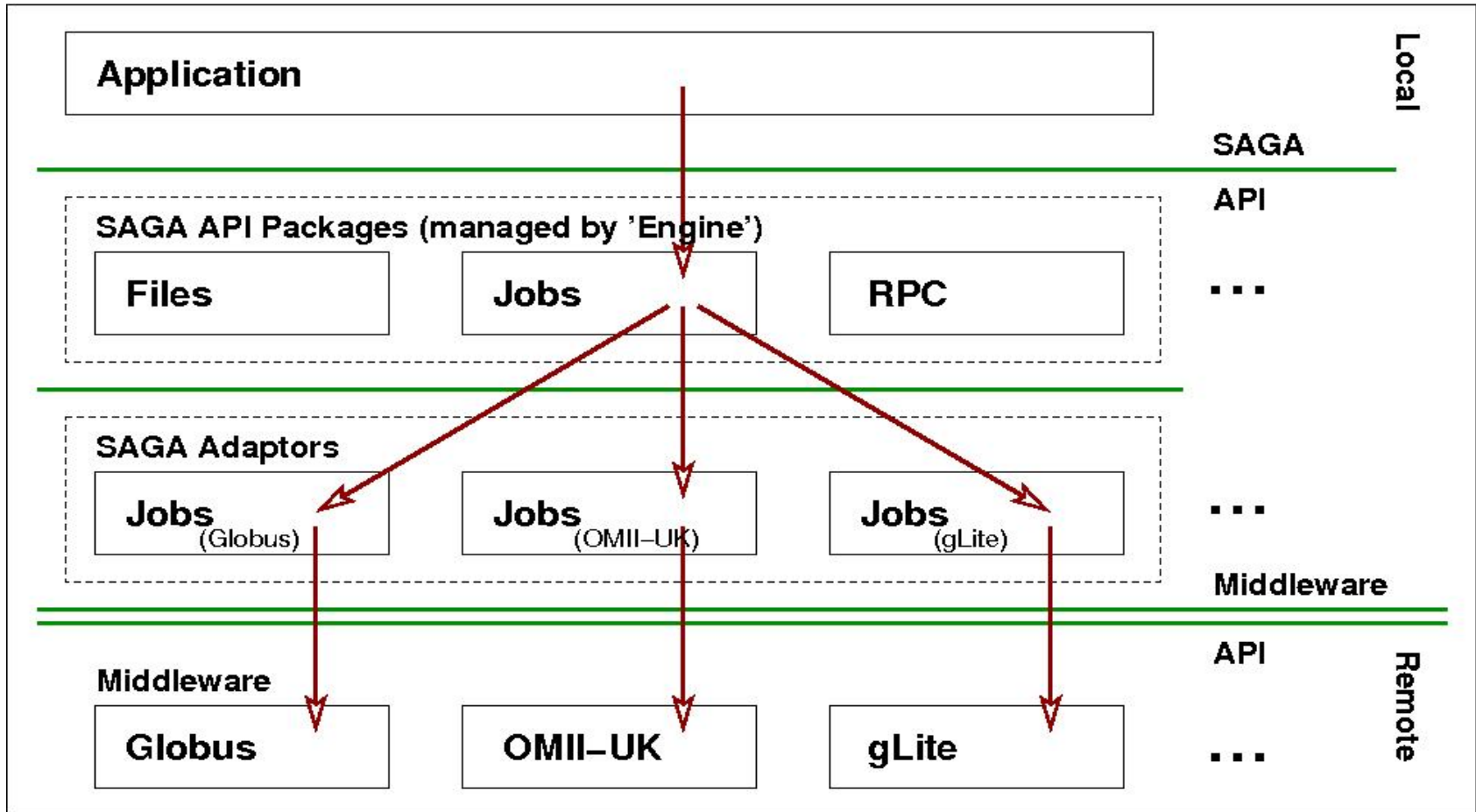
- ❑ For C++, the binding currently is implicitly defined by the reference implementation
- ❑ For Java, a language binding has been defined
  - used by the VU reference implementation
  - still needs to go through OGF's standardization publication process
- ❑ For Python, a language binding has been defined
  - same story as with Java...
- ❑ Language bindings currently are the “weak spot” in the standardization process



- The Simple API for Grid Applications (SAGA)
  - Motivation & Scope
  - SAGA as an OGF Standard
- The SAGA Landscape
  - Interfaces
  - Language Bindings
- SAGA Implementations
  - Engine with Adaptors
  - C++, Java, Python



- Non-trivial set of requirements:
  - Allow heterogeneous 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





- 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
  - Bulk optimization, modular design

# Implementations



- OGF Standard: Two independent implementations of a specification are required
- VU: Java
  - Part of the OMII-UK Project
  - Builds on JavaGAT
- LSU: C++
  - Developed (originally) with/at VU
- VU/LSU: Python
  - wrappers on top of C++ and Java SAGA





## □ C++

- local, Globus 3 and 4, OMII-UK GridSAM, GridFTP, Globus RLS

## □ Java

- local, JavaGAT, OMII-UK GridSAM, XMLRPC

## □ Python

- via C++ or Java

# Other Implementations & Adaptors



- DEISA: Java library
  - JRA7: DEISA (UNICORE) files and jobs
- NAREGI: Java library
  - NAREGI services
- Univ of Virginia: Genesis-II adaptors for Java
- IN2P3 (Lyon): JSAGA (interoperability)
- More Unicore Adaptor – volunteer(s)!!!
- Others that we are not aware of ??

# Upcoming API Extensions



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

**SAGA is very much a work in progress at all levels**

# Finally, is SAGA Simple?



- It depends: It is certainly not simple to implement!
  - Grids are complex and the complexity needs to be addressed somewhere, by someone!
  - Pain using the middleware goes into the SAGA engine and adaptors.
- But it is simple to use!!
  - Functional Packages (specific calls), Look & Feel
  - Somewhat like MPI - most users only need a very small subset of calls



- ❑ It's all about **Applications**
- ❑ SAGA: major ongoing, intellectual OGF effort
- ❑ Still many threads under development: API-level extensions, SAGA Engine & Adaptors
- ❑ SAGA is available
  - across programming languages
  - across different middlewares and their versions
  - from multiple, independent providers
- ❑ ***<http://saga.cct.lsu.edu/>***

# Acknowledgements



- The SAGA Team, at and with OGF:
    - Andre Merzky, Shantenu Jha, Pascal Kleijer, Malcolm Illingworth, Hartmut Kaiser, Ole Weidner, Stephan Hirmer, Cerial Jacobs, Kees Verstoep
  - OMII-UK
  - The European Commission via grants to
    - The CoreGRID network of excellence
    - The XtreemOS project
  - The Dutch VL-e project
  - CCT @ LSU
-