

An Introduction to

SAGA

A Simple API for Grid Applications

Ole Weidner

oweidner@cct.lsu.edu CCT, LSU, Baton Rouge



AT LOUISIANA STATE UNIVERSITY

Outline

- The SAGA Standard
 - Overview and evolution
- The SAGA C++ Implementation
 - Overview
 - Functional packages
 - Middleware adaptors
 - Outlook and agenda

What is SAGA?

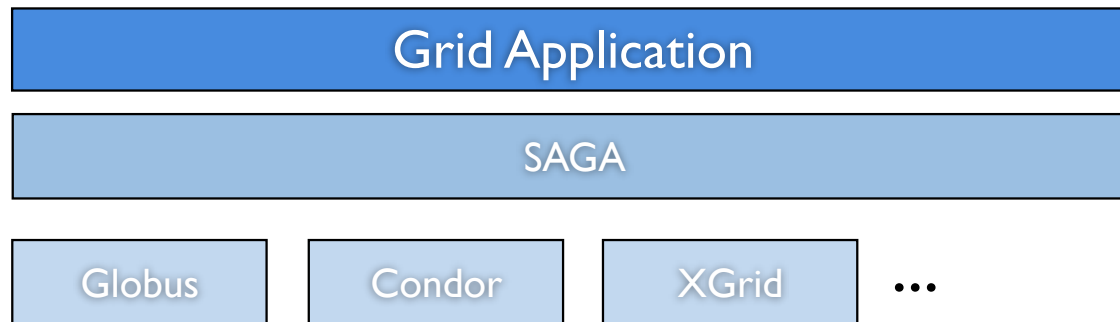
- A Grid API definition defined by the Open Grid Forum (OGF) aimed at high-level application developers who don't want to know anything about Grid computing but want to make use of distributed resources through the Grid
- A Grid API definition that is as easy and natural as possible without having to learn about underlying technologies or middleware

Advantages

- SAGA allows to write truly portable Grid applications which are not bound to a specific middleware but are rather middleware and platform independent
- SAGA allows rapid and natural Grid application development through its simple high-level API and incorporation of well know programming models

What SAGA is not

- No replacement for existing Grid middleware



- Not Bound to specific programming language
- Does not cover all available Grid functionality

Why SAGA?

- Current Grid Middleware APIs are not focused on application programming
- Groups are continually wrapping Grid technologies with application or domain specific APIs
- Several other projects developing generic APIs but there was no standardization process yet

SAGA Evolution

- Collection of community driven use-cases built the foundation for the SAGA standard
- Current standard includes support for
 - jobs, files/directories, replica, streams and rpc
- Version 1.0 is almost released
- Along with the standard we developed a first reference implementation that'll be available at the same time as the standard

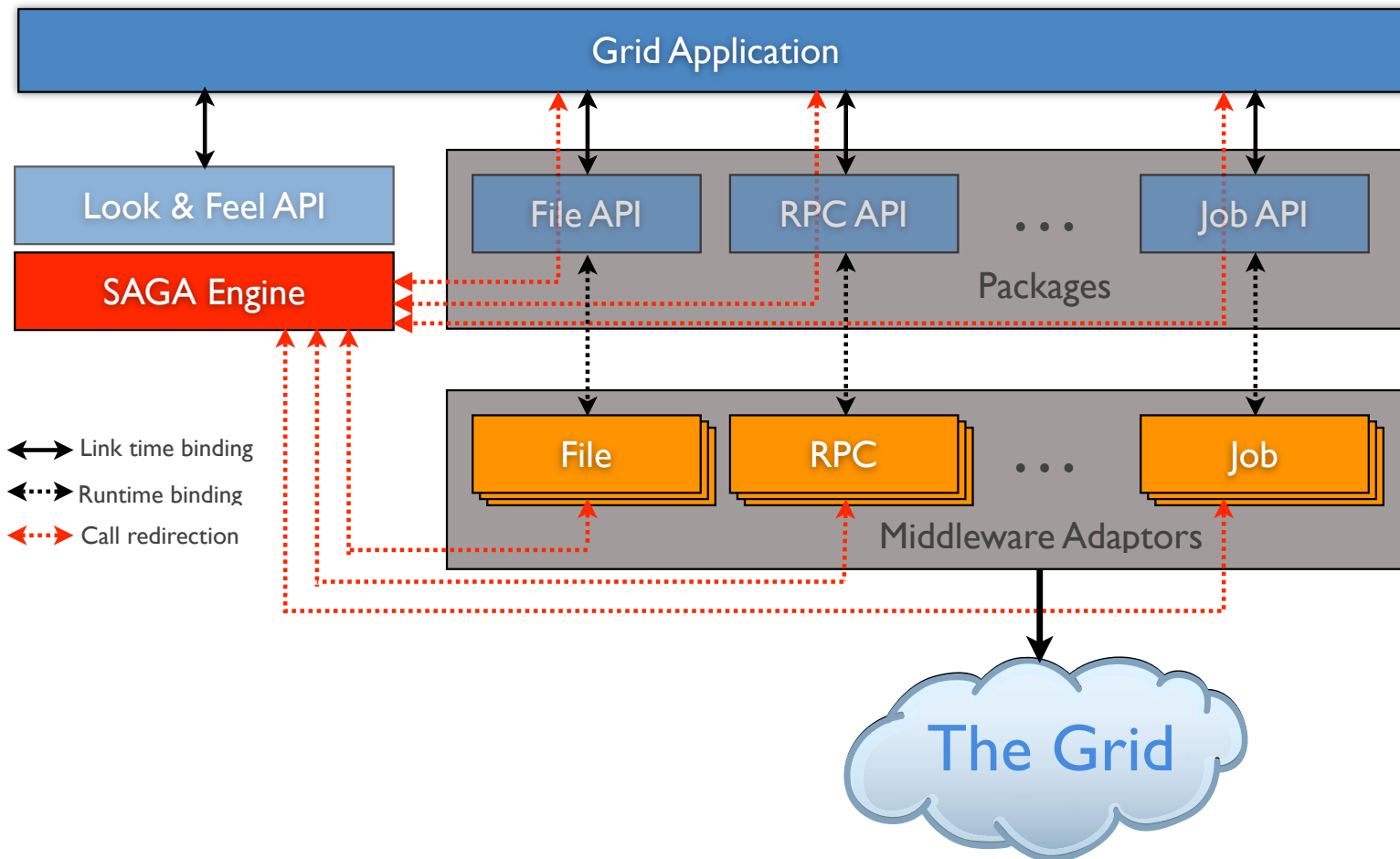
C++ Reference Implementation

- It is the first and (currently) only complete SAGA implementation
- Strictly adopts the complete SAGA standard
- Implementation aims to be as modular (=extensible) as possible
- Does not only implement the API but also a concept for dynamic middleware adaption

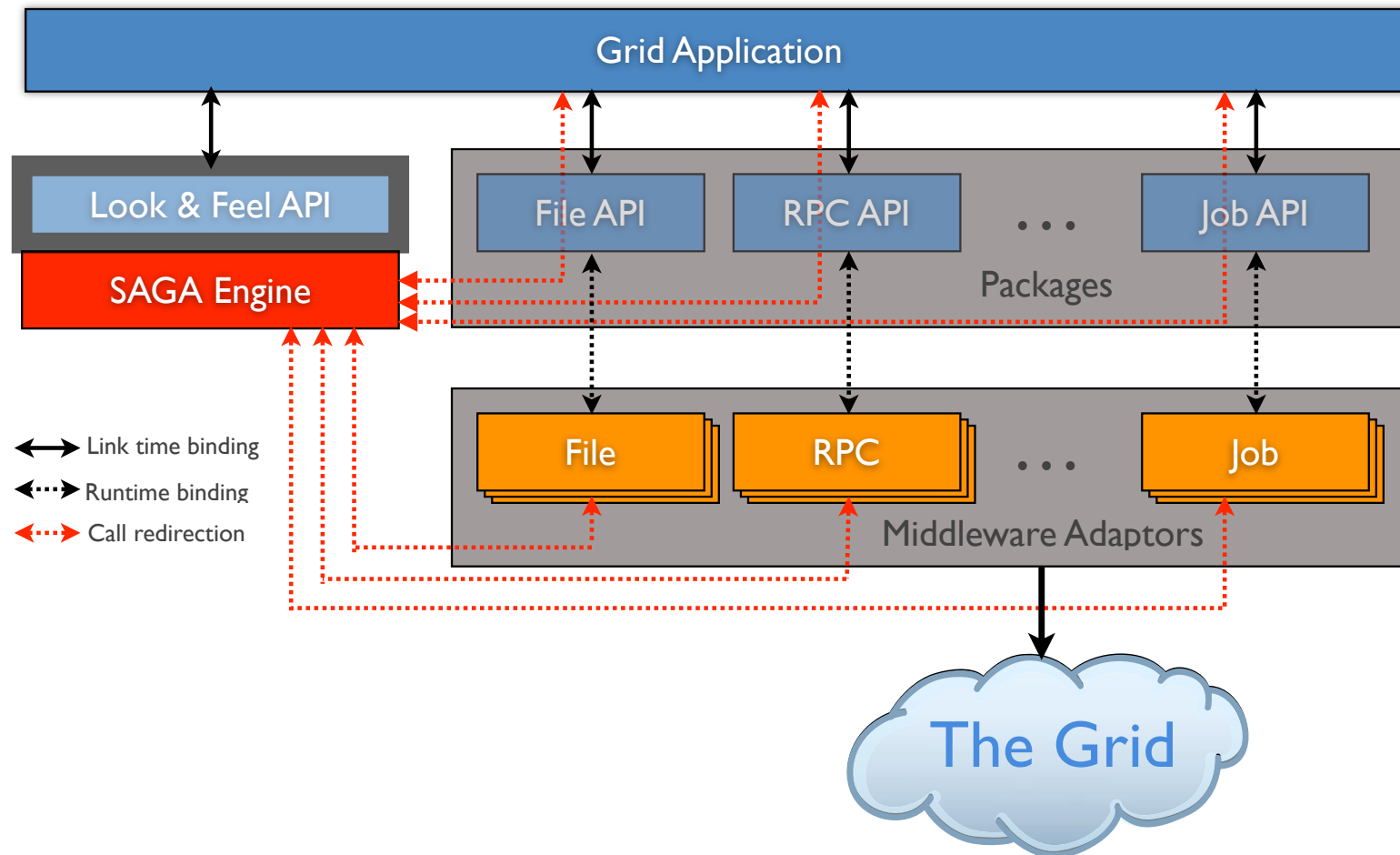
C++ Reference Implementation

- It is written in 100% platform independent C++ code using ANSI C++ and the Boost Project's C++ libraries
- The only platform-dependent part of the implementation are the middleware binding adaptors - some might be not be available on all platforms

The Big Picture



Look & Feel API



Look & Feel API Features

- Unified “look&feel” for all SAGA packages
- The API has a number of core features
 - Unified Error handling
 - Inherent asynchronism (task model)
 - Monitoring model

Error Handling

- Provides SAGA specific exception class
- Set of well defined codes used by all SAGA methods to indicate errors

Success	= 0	NotImplemented	= 1
IncorrectURL	= 2	IncorrectSession	= 3
AuthenticationFailed	= 4	AuthorizationFailed	= 5
PermissionDenied	= 6	BadParameter	= 7
IncorrectState	= 8	AlreadyExists	= 9
DoesNotExist	= 10	ReadOnly	= 11
Timeout	= 12	NoAdaptor	= 13
NoAdaptorInfo	= 14	Unexpected	= 15
NoSuccess	= 16		

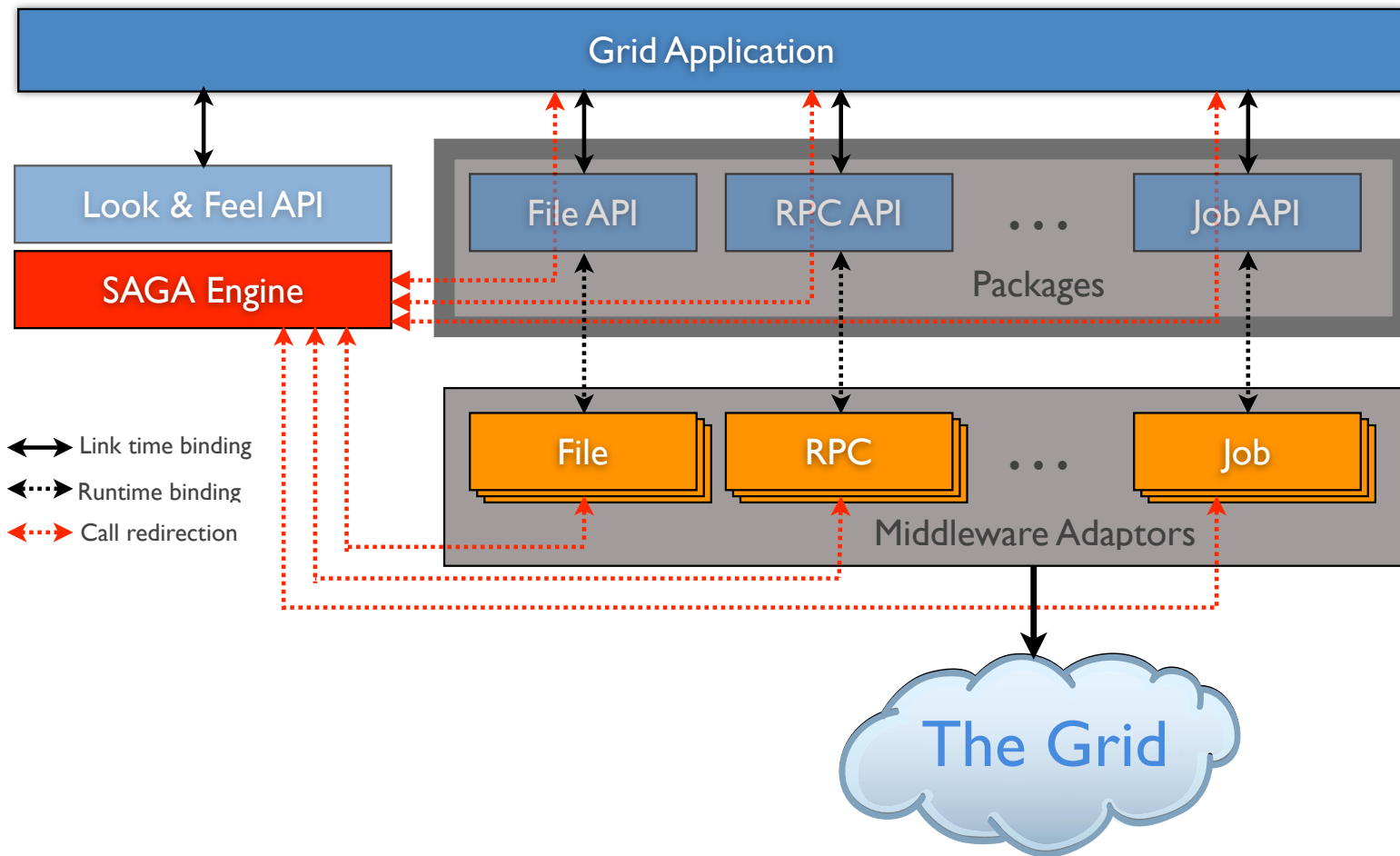
Task Model

- All SAGA objects implement the task model
- Every method has three “flavors”
 - 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

Monitoring Model

- Certain SAGA Objects implement the monitoring interface
- Objects can expose readable (monitor-able) or read-/writable (steerable) metrics (values) using the monitoring model
- Other objects can query these metrics or register a callback to get notified if a metric has changed

Functional Packages



Functional Packages

- A functional packages is a group of objects and methods for a specific Grid feature
- Implemented as C++ dynamic libraries
 - More control over features
 - Reduce memory footprint
- Easily vertically extensible
 - No horizontal code dependencies

Available Package

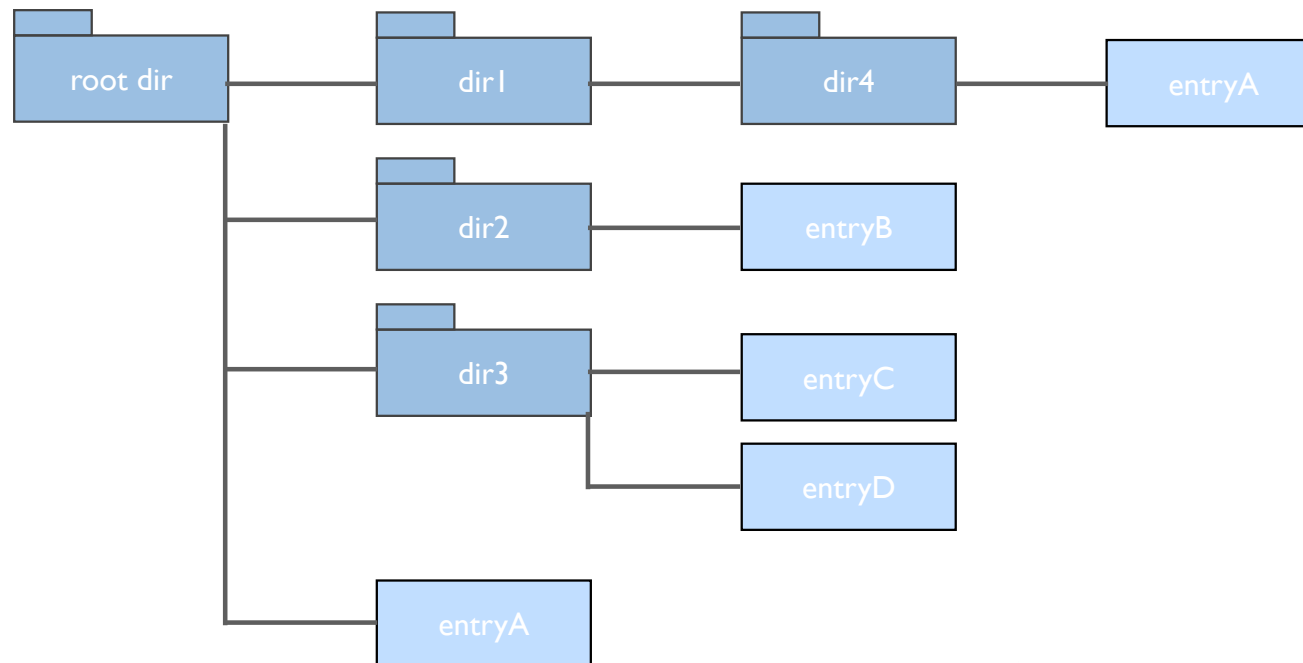
- Namespace Package
- Job Package
- File Package
- Logical File Package (replica)
- RPC Package
- Stream Package

Namespace Package API

- Fundamental data structure in SAGA
- Namespace representation as described by POSIX standard
- Defines hierarchical structure of directories and entries
 - directory may contain other directories or entries
 - entries can be any SAGA object

Namespace Package API

- Example namespace hierarchy



Namespace Package API

- Namespace API supports
 - symbolic links (pointer to entries or directories)
 - searching directories and entries
 - access control lists (ACL)
 - pattern based operations

Namespace Package API

- Pattern based operations support
 - Standard POSIX shell wildcards

<code>*</code>	matches any string
<code>?</code>	matches a single character
<code>[abc]</code>	matches any of a set of characters
<code>[a-z]</code>	matches any of a range of characters
<code>[!abc]</code>	matches none of a range of characters
<code>[!a-z]</code>	matches none of a range of characters
<code>{a,bc}</code>	matches any of a set of strings

- All path-based operation accept wildcards

Namespace Package API

- Access Control List (ACL) support
 - Implements permission model for namespaces
 - Entries or directories can have an ACL
 - POSIX-like ACL implementation

```
dn_user = "O=dutchgrid, O=vu, CN=Joe Doe";  
dn_group = "O=dutchgrid, O=vu, CN=*";  
dn_group = "O=dutchgrid, O=project-123, CN=*";  
dn_group = "O=*, O=project-123, CN=*";
```

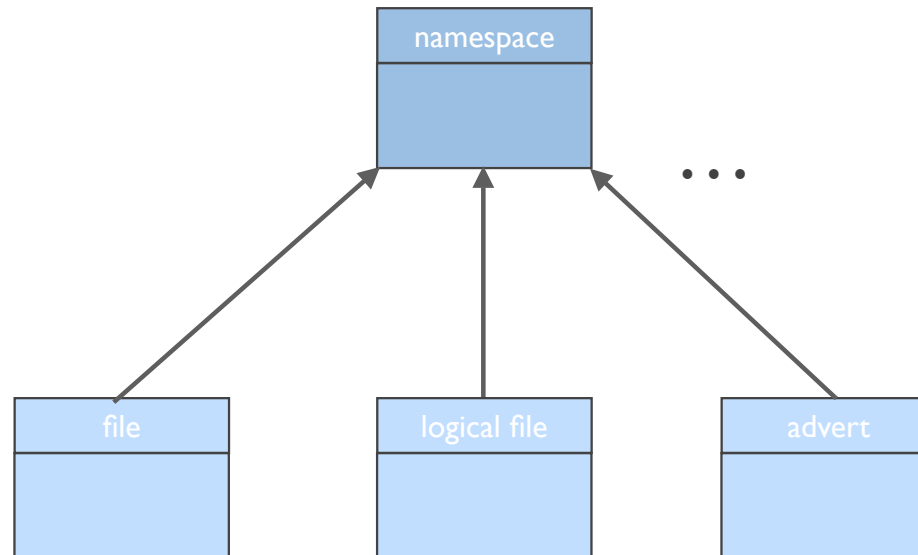
- Methods: [set_acl\(\)](#), [get_acl\(\)](#), [list_dn\(\)](#)

Namespace Package API

- Namespace entry inspection methods
 - `get_url()`, `get_cwd()`, `get_name()`, `exists()`, `is_link()`
- Navigation methods
 - `read_link()`, `change_dir()`, `list()`, `find()`
 - `get_num_entries()`, `get_entry()`
- Modification methods
 - `copy()`, `link()`, `move()`, `remove()`

Namespace Package API

- Why is namespace fundamental?



Namespace API Example

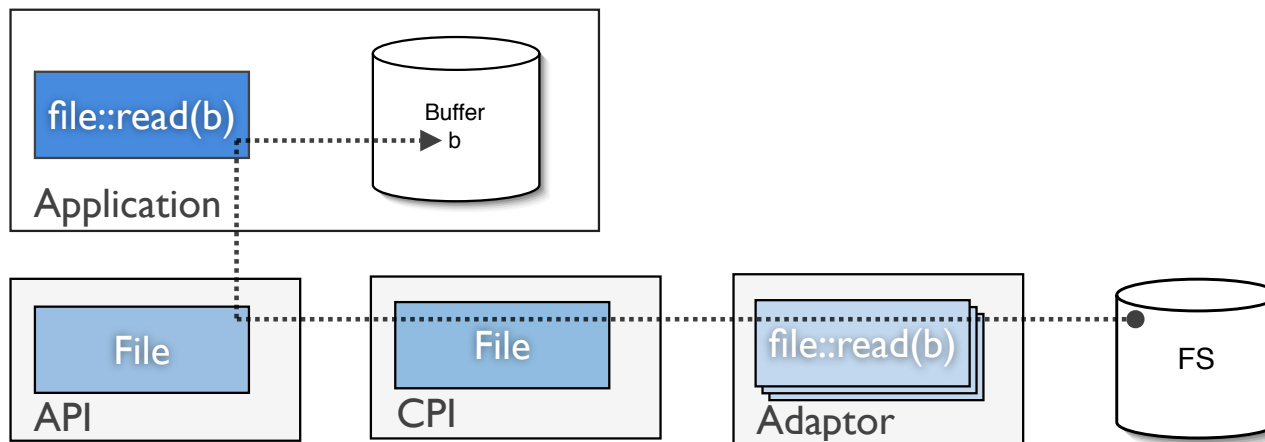
```
01: // Recursive Namespace structure listing
02: //
03: void list_dir( std::string & url) {
04:     saga::ns_dir dir (url);
05:
06:     for ( int i = 0; i < dir.get_num_entries (); i++ )
07:     {
08:         string name = dir.get_entry (i);
09:
10:         std::cout << name << std::endl
11:
12:         if( dir.is_dir(name) )
13:         {
14:             list_dir(name);
15:         }
16:     }
```

File Package API

- Namespace with file/dir objects as entries
- Extends namespace API with I/O methods
- POSIX-style zero-copy read/write
- API supports
 - Scattered I/O
 - Pattern-based I/O

File Package API

- Standard I/O methods
 - `read()`, `write()`, `seek()`
 - Read-/write- methods are zero-copy



File Package API

- Scattered I/O support (POSIX)
 - Read/write operates on buffer vectors
 - Cluster multiple I/O operations into a single call
 - Effective for large files
 - Ineffective for small files
 - Needs middleware support
 - Methods: `read_v()`, `write_v()`

File Package API

- Pattern-based I/O support
 - Idea and syntax based on FALLS (FAMiLy of Line Segments)
 - Describes a regular pattern on binary data
 - Read/write only data matching the pattern
 - Very effective on multi-layer files (e.g. image data)
 - Methods: `size_p()`, `read_p()`, `write_p()`

File API Example

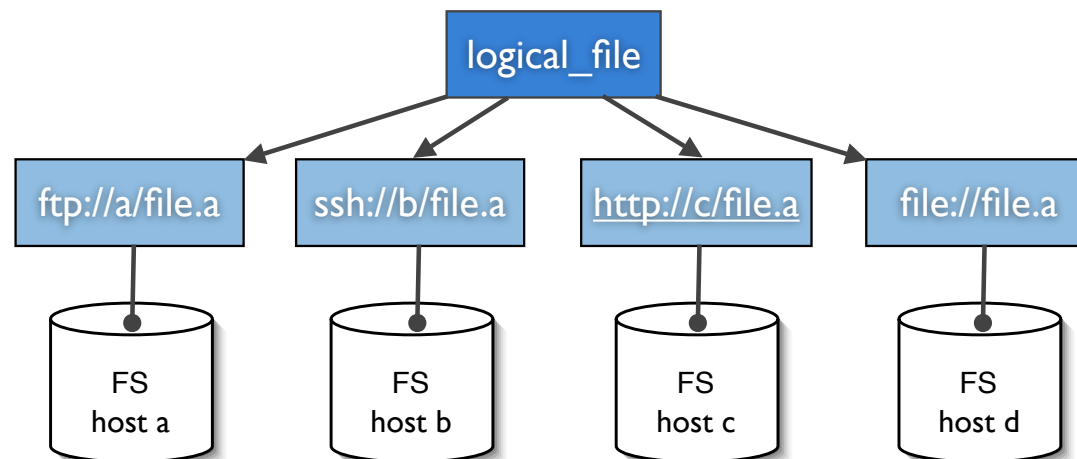
```
01: // Read the first 10 bytes of a file if file size > 10 bytes
02: //
03: saga::file my_file ("griftp://gridhub/~/result.dat);
04:
05: off_t size = my_file.get_size ();
06:
07: if ( size > 10 )
08: {
09:     char buffer[11];
10:     long buflen;
11:
12:     my_file.read (10, buffer, &buflen);
13: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

Logical File Package API

- Provides file and directory replica service interface (similar to Globus RLS)
- Implements SAGA namespace with logical file and directory objects as entries
- Allows metadata association through SAGA attribute interface
- Extends namespace API with methods for replica handling

Logical File Package API

- A logical file points to one or more copies of a physical file (replicas) or other logical files
- A physical file (replica) is a file system entry



Logical File Package API

- Replica management methods
 - `add_location()` adds physical file entry
 - `remove_location()` removes physical file entry
 - `update_location()` changes physical file entry
 - `list_locations()` lists all logical / physical mappings
 - `replicate()` create new physical file copy

Logical File API Example

```
01: // Replicate a logical file and check its size
02: //
03: saga::logical_file lf ("lfn://remote.catalog.net/tmp/file1");
04:
05: lf.replicate ("gsiftp://localhost.net/tmp/file.rep");
06:
07: saga::file f ("gsiftp://localhost.net/tmp/file.rep");
08:
09: std::cout << "size of local replica: "
10:           << f.get_size ()
11:           << std::endl;
```

Job Package API

- Job handling inspired by DRMAA
- Extends the SAGA task model API
- API supports
 - Job description
 - Job submission & control
 - Job reconnection
 - Job I/O redirection

Job Package API

- Job description
 - Based on SAGA attributes (key value pairs)
 - Well defined set of keys based on JSDL 1.0

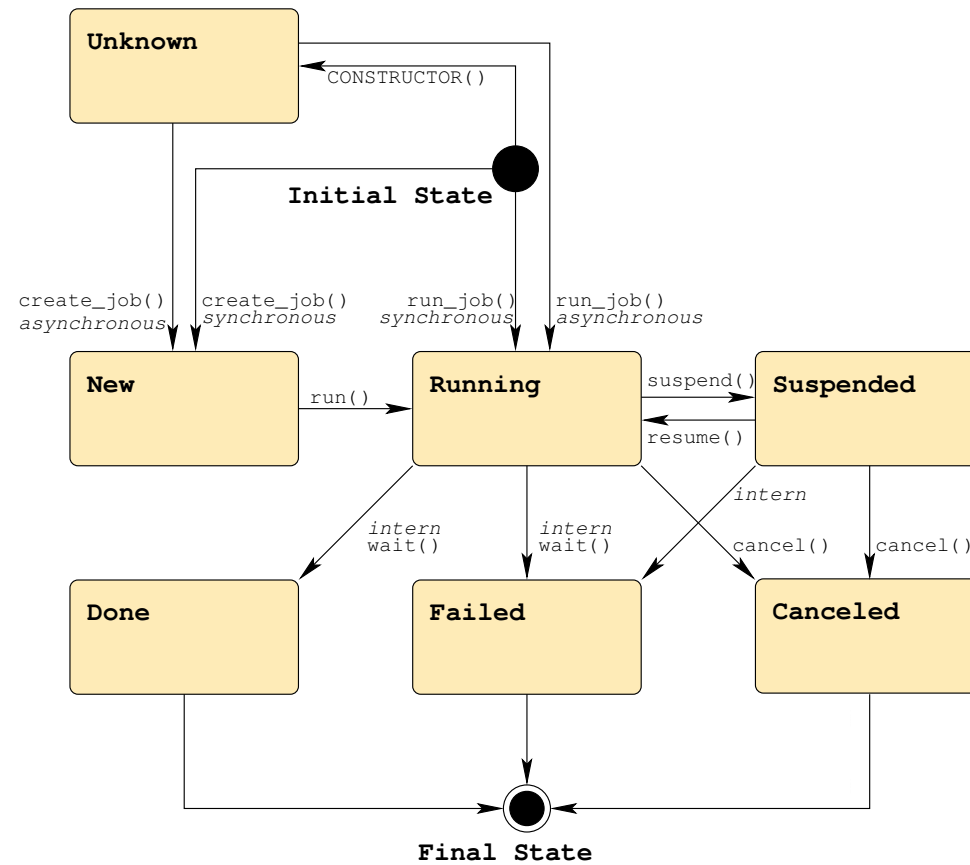
```
01: std::list <string> transfers;  
02: transfers.push_back ("infile > infile");  
03: transfers.push_back ("ftp://host.net/path/out << outfile");  
04:  
05: saga::job_description jobdef;  
06: jobdef.set_attribute ("Executable", "job.sh");  
07: jobdef.set_attribute ("TotalCPUCount", "16");  
08: jobdef.set_vector_attribute ("FileTransfer", transfers);
```

Job Package API

- Job submission
 - Methods: `create_job()`, `run_job()`
- Job state control
 - Extends SAGA task API - same “look&feel”
 - Methods: `run()`, `cancel ()`, `suspend()`, `resume()`,
- Support for checkpointing and job migration
 - Methods: `checkpoint()`, `migrate()`

Job Package API

- Job state model - extends task state model



Job Package API

- Job I/O redirection support
 - stdin, stdout and stderr available as C++ streams
 - Methods: `get_stdin()`, `get_stderr()`, `get_stdout()`
- Job reconnection support
 - (Re-)connect to running job based on JobID
 - Job doesn't need necessarily to be a "SAGA Job"

Job API Example

```
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: }
```

Stream Package API

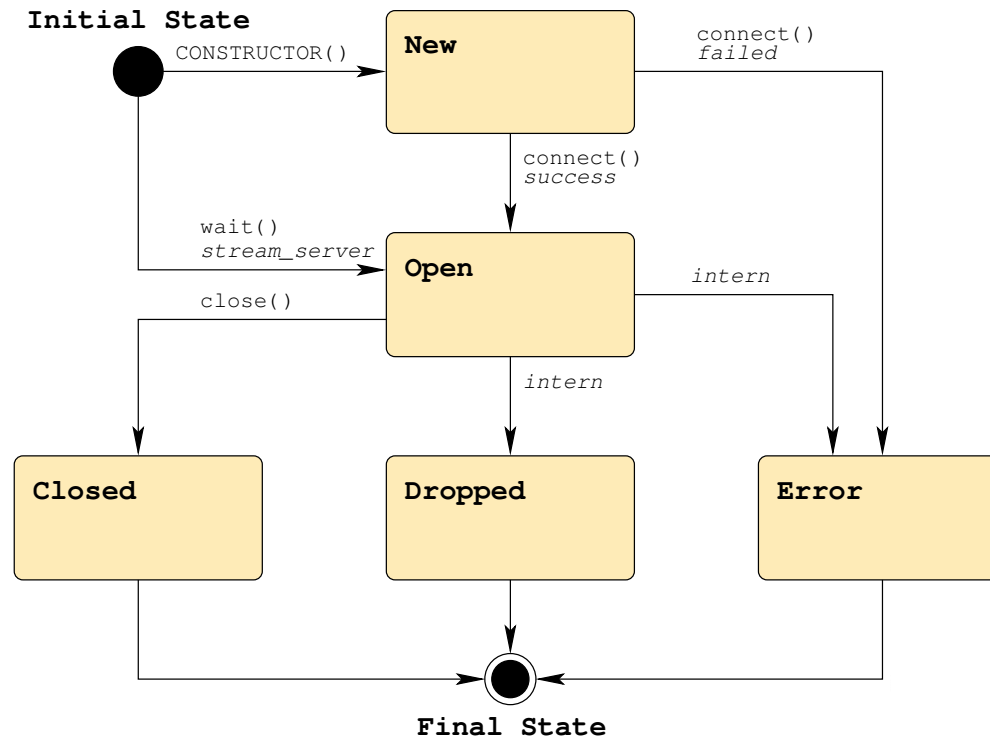
- Provides authenticated socket (endpoint) connection similar to BSD sockets
- Support hooks for authorization and encryption schemes
- Focus on simplicity, not performance - but can be fast anyway ;-)

Stream Package API

- Basic stream setup methods
 - Create waiting endpoint `serve()`
 - Create connecting endpoint `connect()`
 - Close endpoint connection `close()`
- Stream I/O methods
 - `read()`, `write()` reading/writing using raw buffers
 - `wait()` check if stream is ready for I/O

Stream Package API

- Stream connection state model



Stream API Example

```
01: // Open stream, read & write
02: //
03: int recvlen;
04: saga::stream s ("localhost:5000");
05:
06: s.connect ();
07: s.write ("Hello World!", 12);
08:
09: // blocking read, read up to 128 bytes
10: recvlen = s.read (buffer, 128);
```

RPC Package API

- Remote Procedure Calls based on OGF GridRPC standard
- Method is encoded in SAGA URL

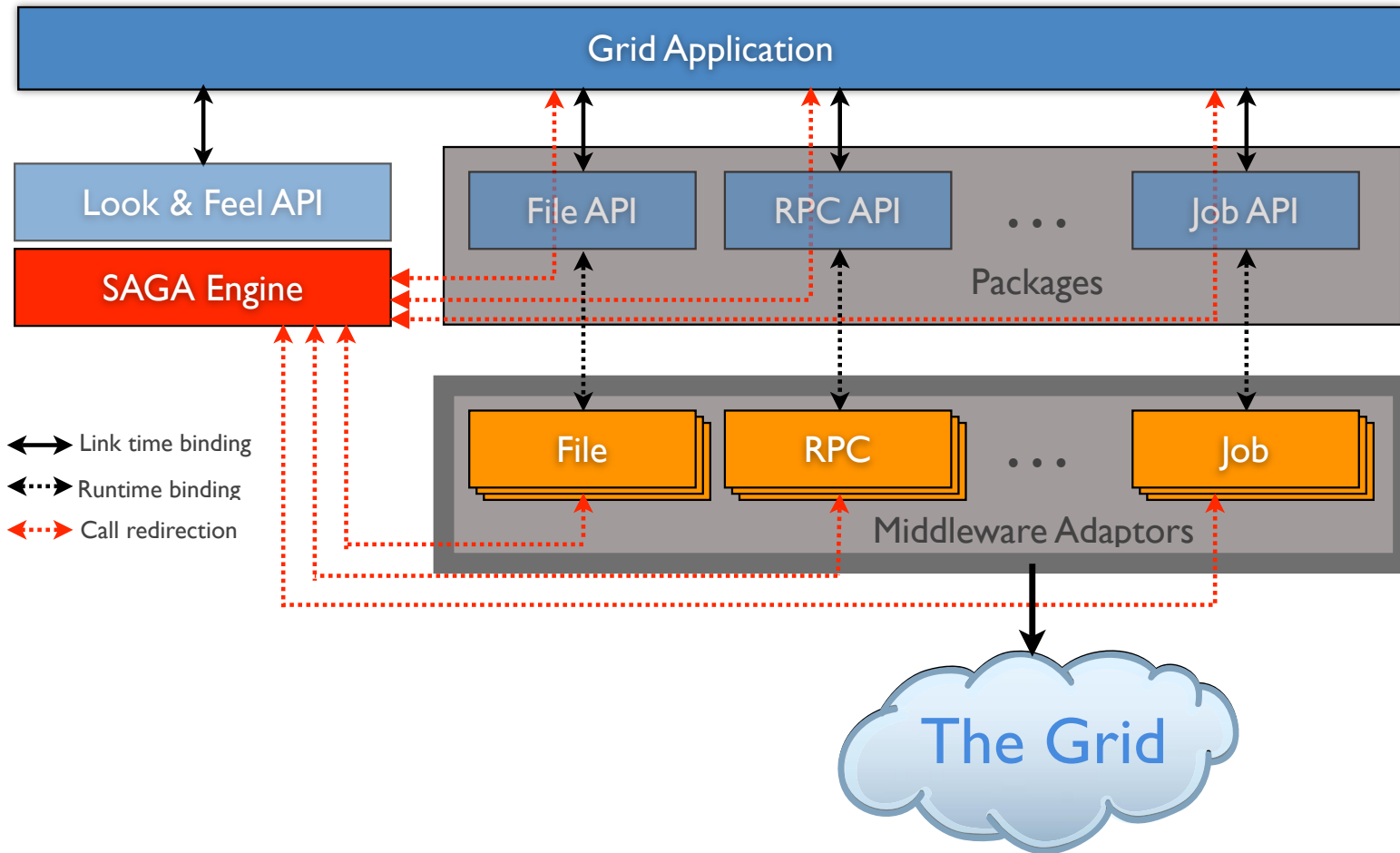
```
gridrpc://server.net:1234/my_function
```

- Array of structures contains parameters and return values

RPC API Example

```
01: // Call a remote procedure
02: //
03: rpc rpc ("gridrpc://fs0.das2.cs.vu.nl/matmul1");
04:
05: std::vector <saga::rpc::parameter> params (2);
06:
07: params[0].buffer = // ptr to matrix A
08: params[0].size = sizeof (buffer);
09: params[0].mode = saga::rpc::InOut;
10:
11: params[1].buffer = // ptr to matrix B
12: params[1].size = sizeof (buffer);
13: params[1].mode = saga::rpc::In;
14:
15: rpc.call (&params);
16:
17: // A now contains the result
```

Adaptors



Adaptors

- Packages are only as usefull as the adaptors which realize them!
- Package implementation almost done
NOW: focus on high-quality adaptor implementations
- Use of sophisticated conformance test suite to test every aspect of the adaptor implementation

File Adaptors

- Default File Adaptor
 - stable
 - Uses Boost Filesystem API - works everywhere
- Globus GridFTP File Adaptor
 - preview
 - Works where GT4 client API is available

Logical File Adaptors

- Default Logical File Adaptor
 - stable
 - Uses SOCI/SQL to create/manage replica catalog
 - Works already with SQLite and PostgreSQL
- Globus pre-OGSI RLS Adaptor
 - Not usable (in progress)

Job Adaptors

- Default Job Adaptor
 - stable
 - Uses threads to spawn new jobs - runs everywhere
- Globus pre-OGSI GRAM Job Adaptor
 - preview - some features missing

Stream Adaptors

- Default Stream Adaptor
 - Not implemented yet

RPC Adaptors

- Default RPC Adaptor
 - preview
 - Uses XMLRPC for procedure calling

Current SAGA Agenda

- First SAGA binary release in preparation
- OMII-UK Middleware Integration
 - SAGA recently got funded by OMII to develop a complete set of WS-based Globus adaptors together with Python/C API wrappers
 - Starts in April - Adaptor set will be ready within 12 months
- SAGA LONI deployment

Current SAGA Projects

- GridSAT/SAGA
 - Re-implementation of a grid-enabled parallel SAT solver using SAGA (UC San Diego)
 - First large-scale Grid application using SAGA

Thank you for your attention

More information can be found at

<http://saga.cct.lsu.edu>