

# OGF21

## Software Providers Track

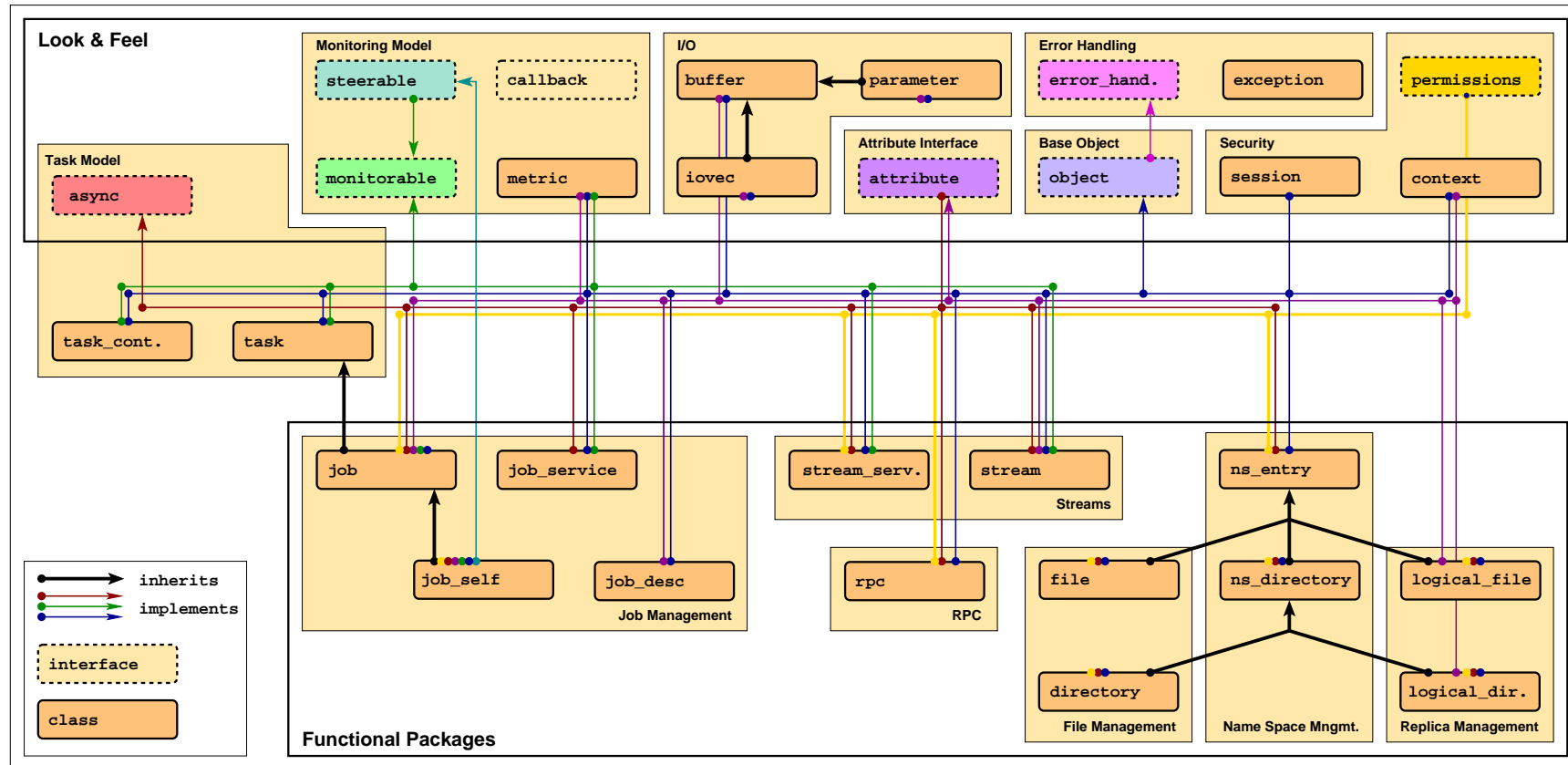
SAGA Overview / C++ Implementation

# SAGA overview

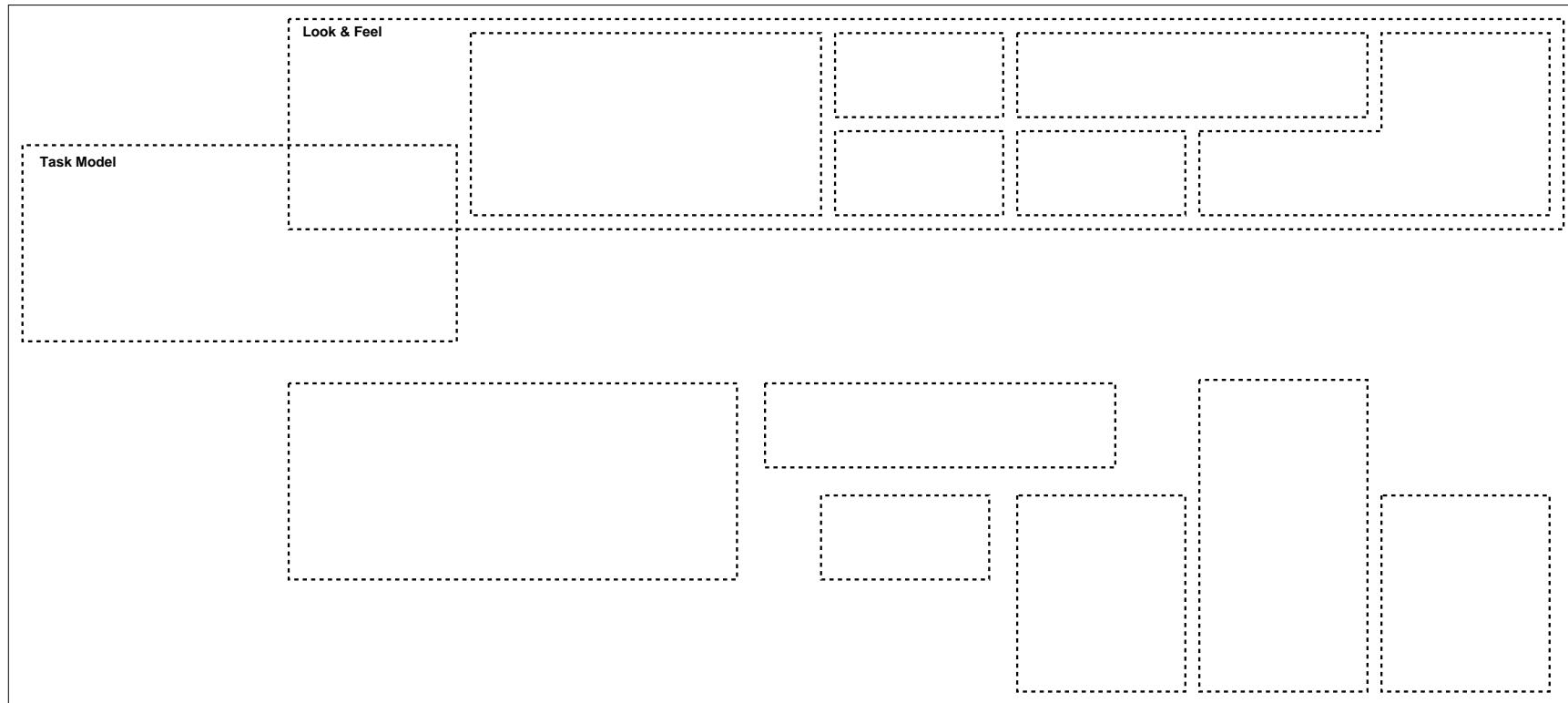
---

- SAGA API scope
- class hierarchy
- code examples
- planned extensions
- implementation status
- tutorial

# SAGA: Overview



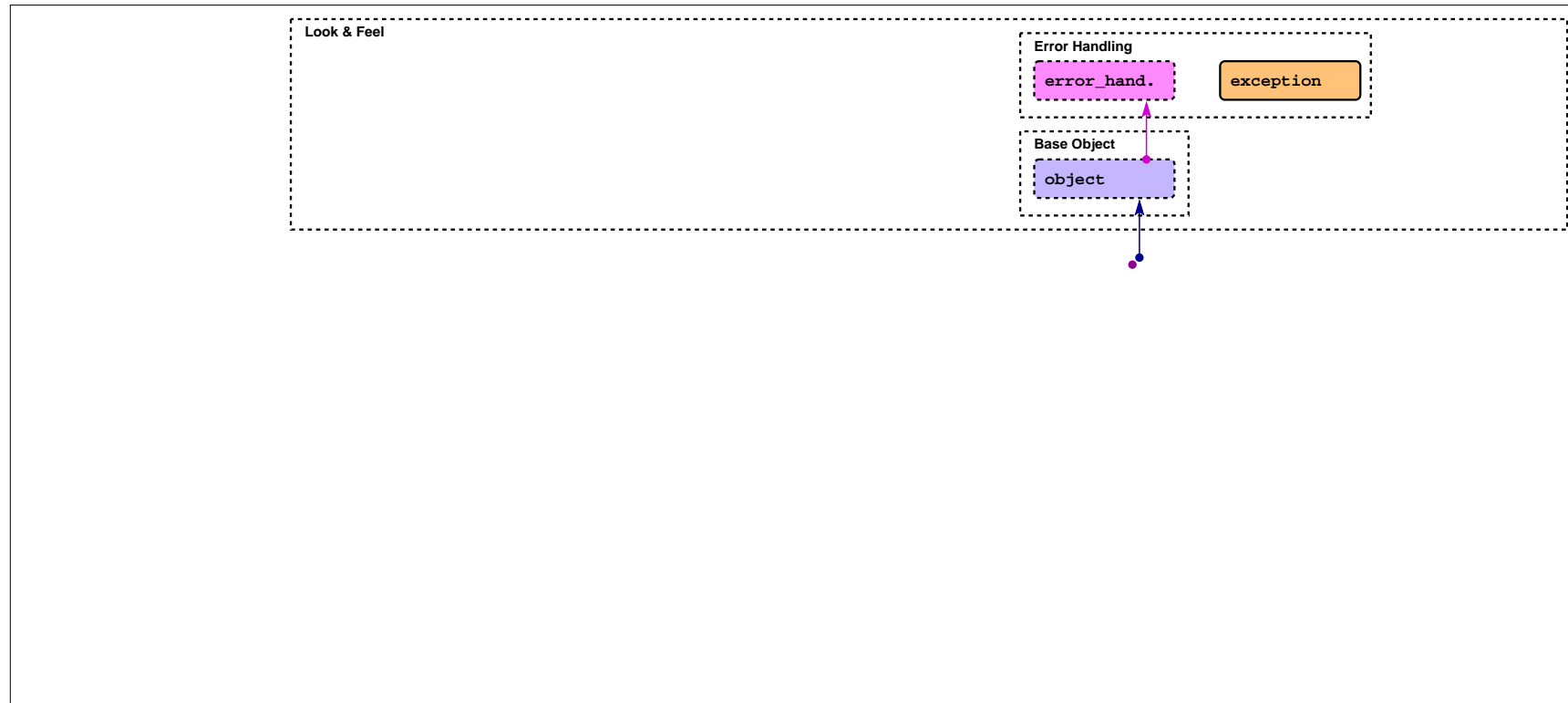
# SAGA: Class hierarchy



## SAGA Structure

Look & Feel versus functional packages

# SAGA: Class hierarchy

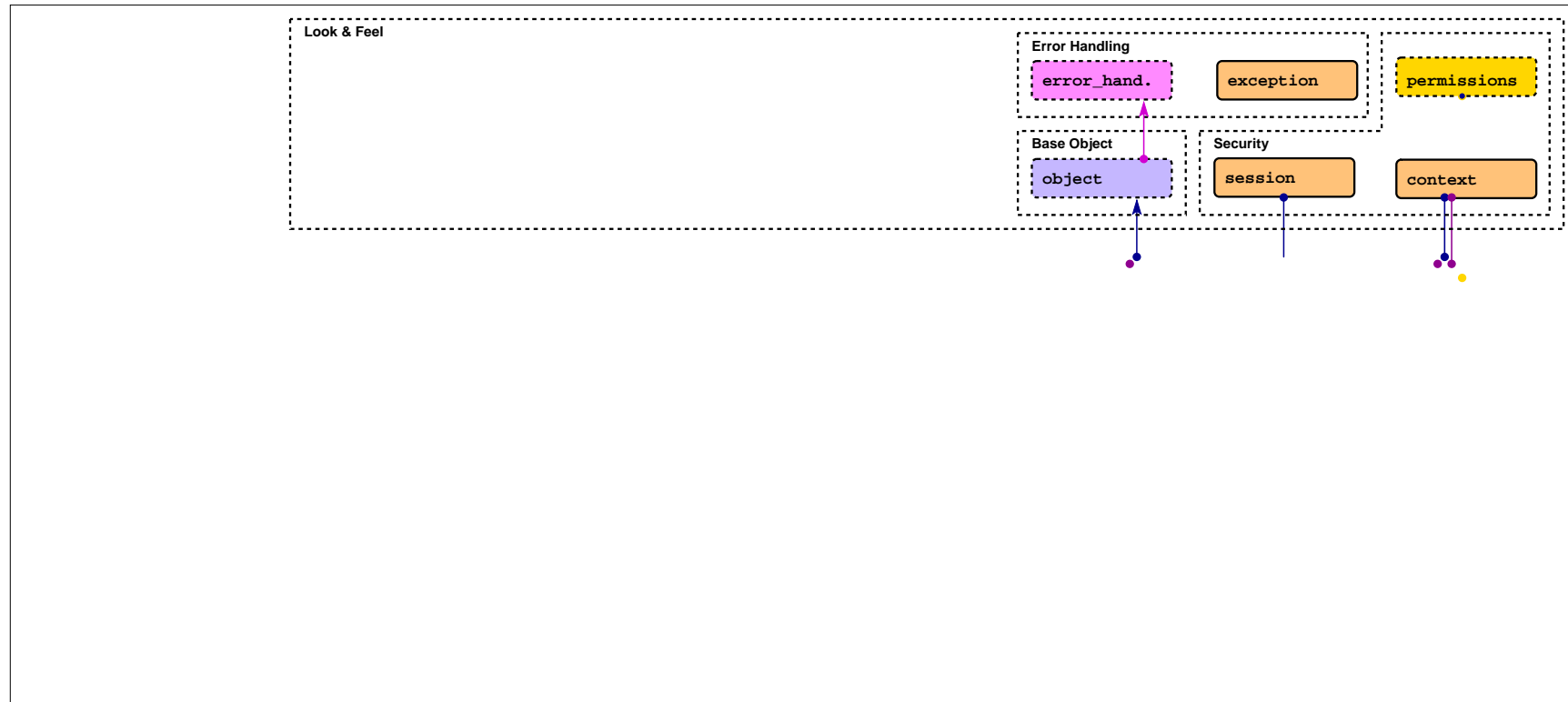


## SAGA Look & Feel:

`saga::object` allows for object uuids, `clone()` etc.

errors are based on exceptions or error codes.

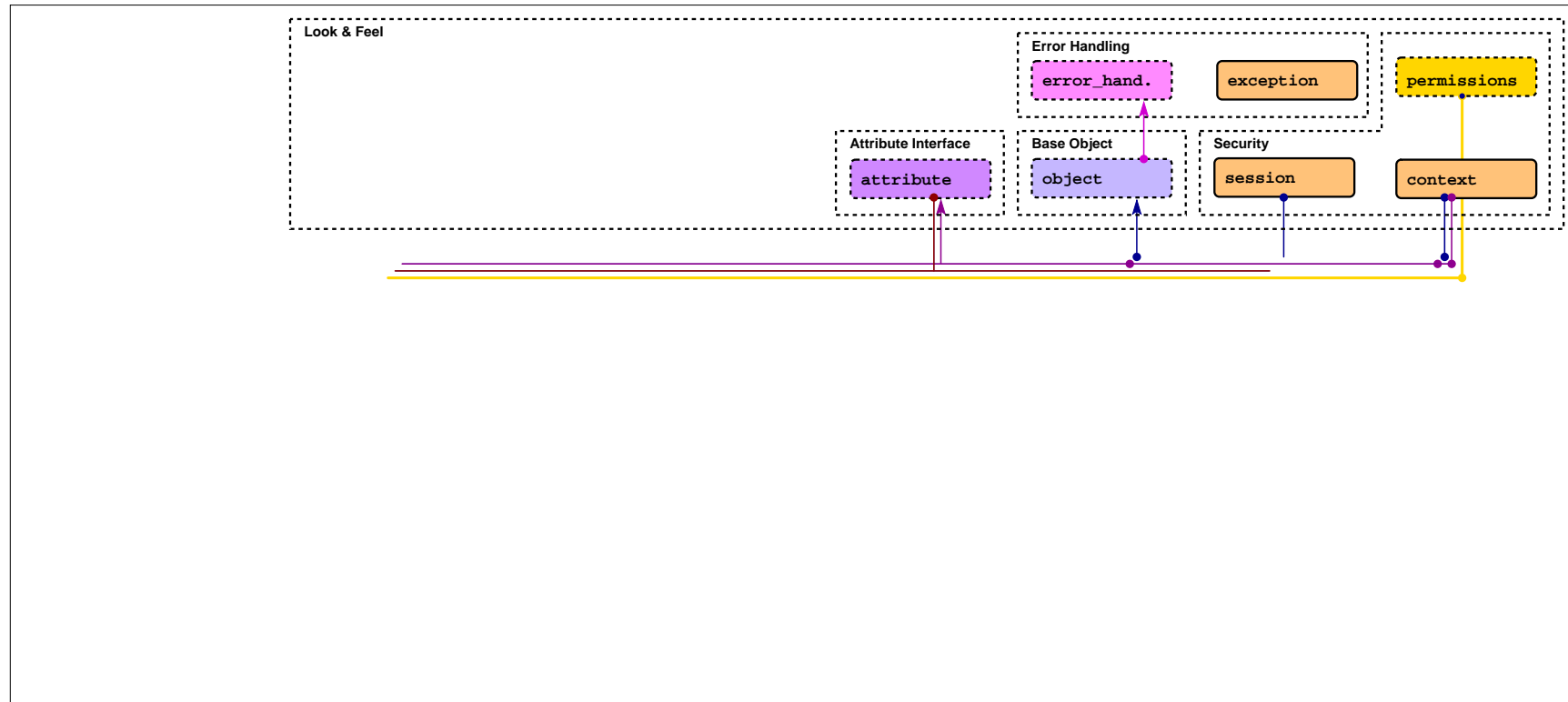
# SAGA: Class hierarchy



## SAGA Look & Feel:

session and credential management is hidden by default.

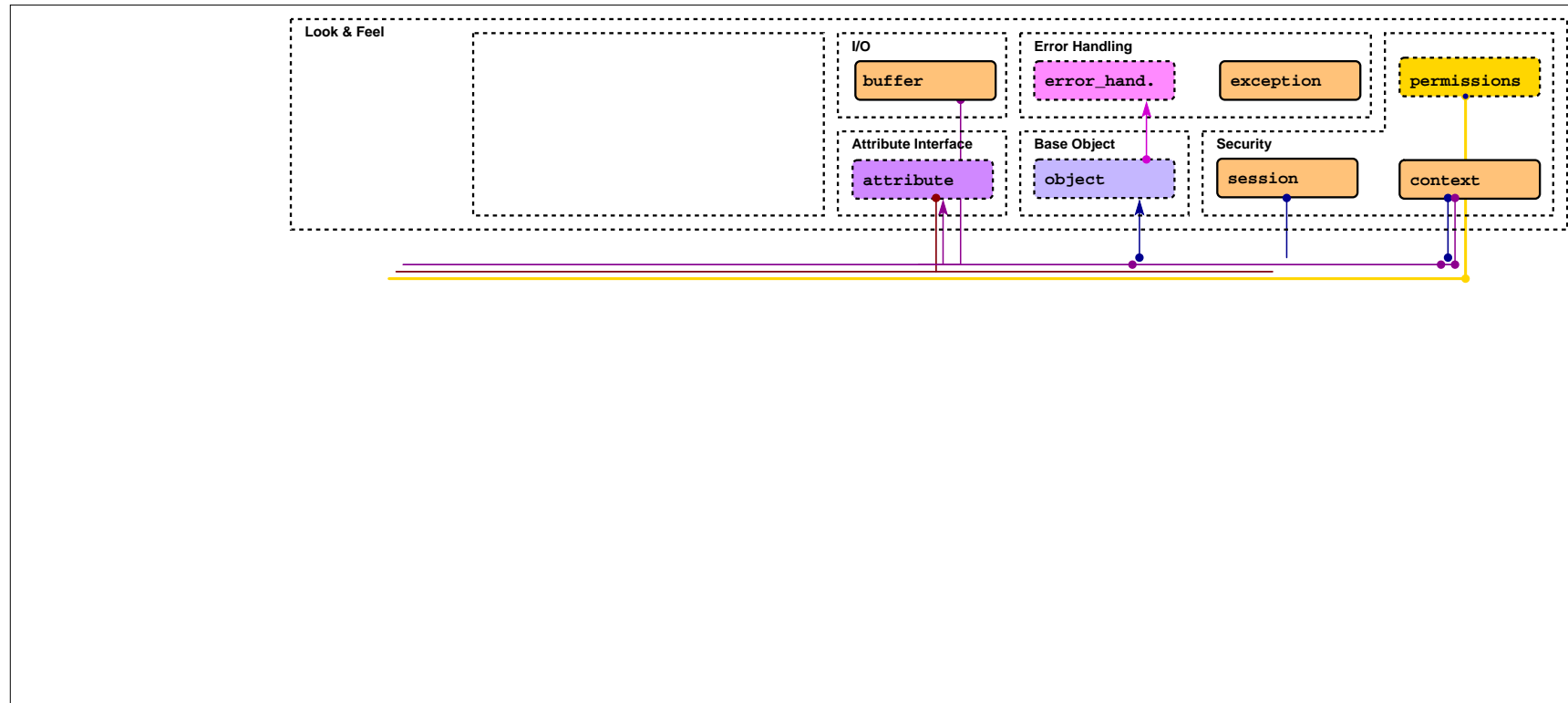
# SAGA: Class hierarchy



## SAGA Look & Feel:

Attribute interface for meta data.

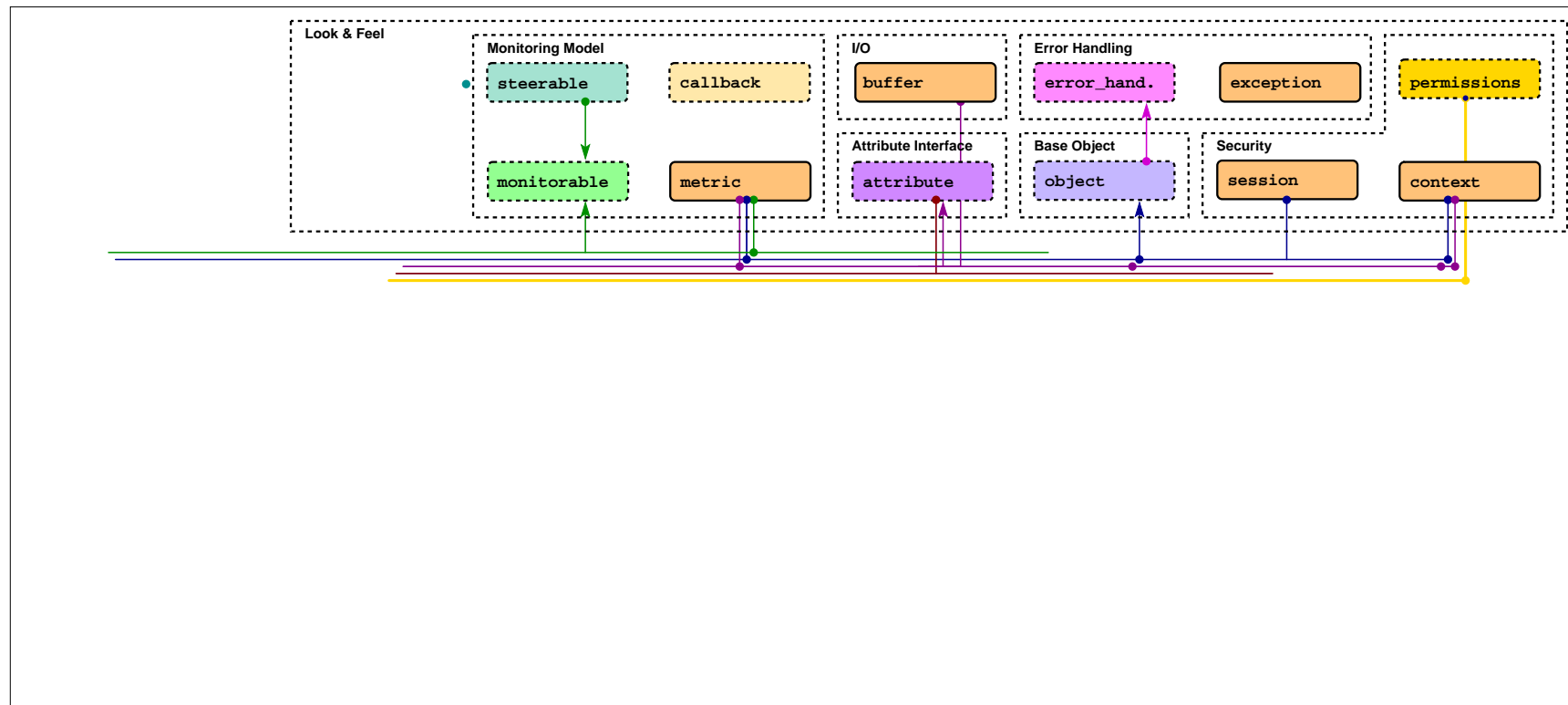
# SAGA: Class hierarchy



## SAGA Look & Feel:

buffers provide a uniform data interface for I/O

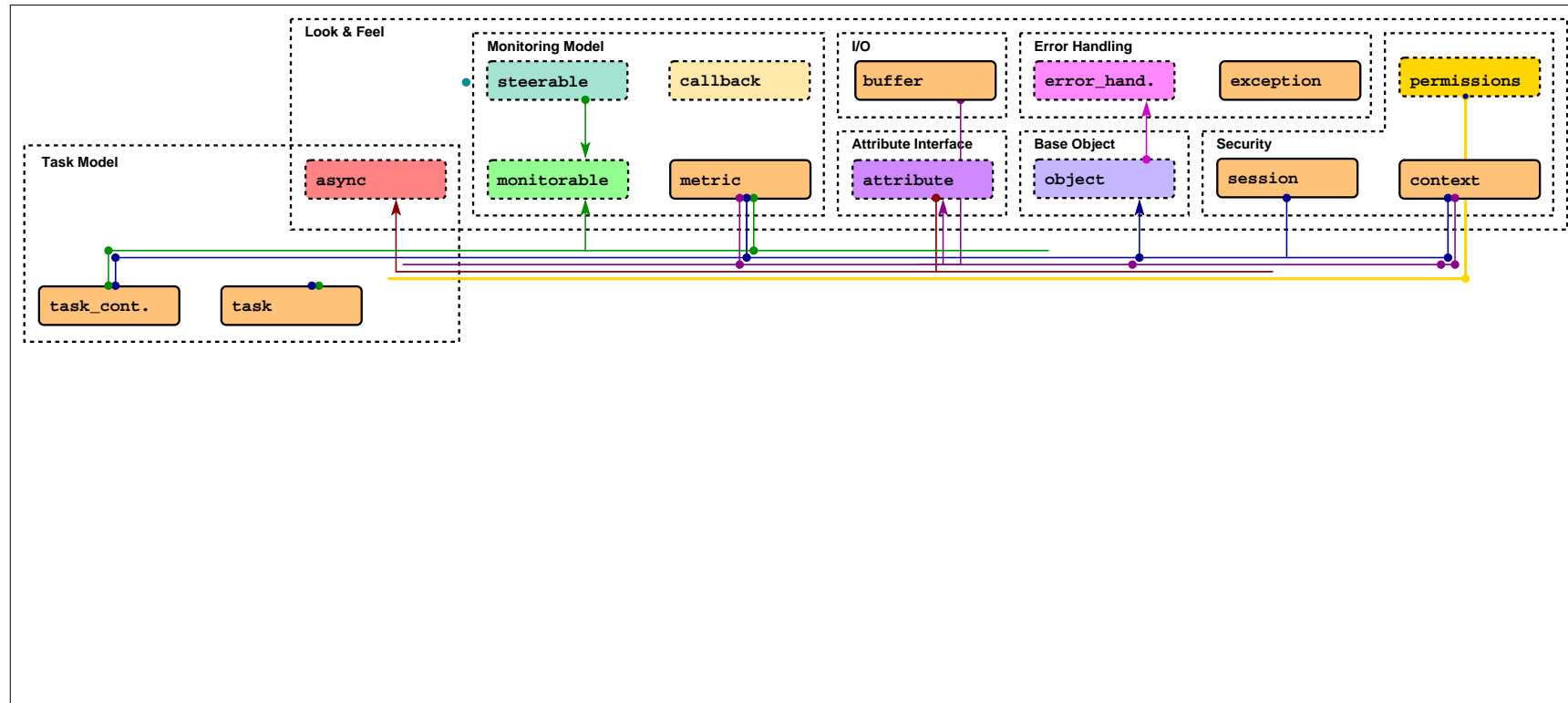
# SAGA: Class hierarchy



## SAGA Look & Feel:

Monitoring includes asynchronous notifications.

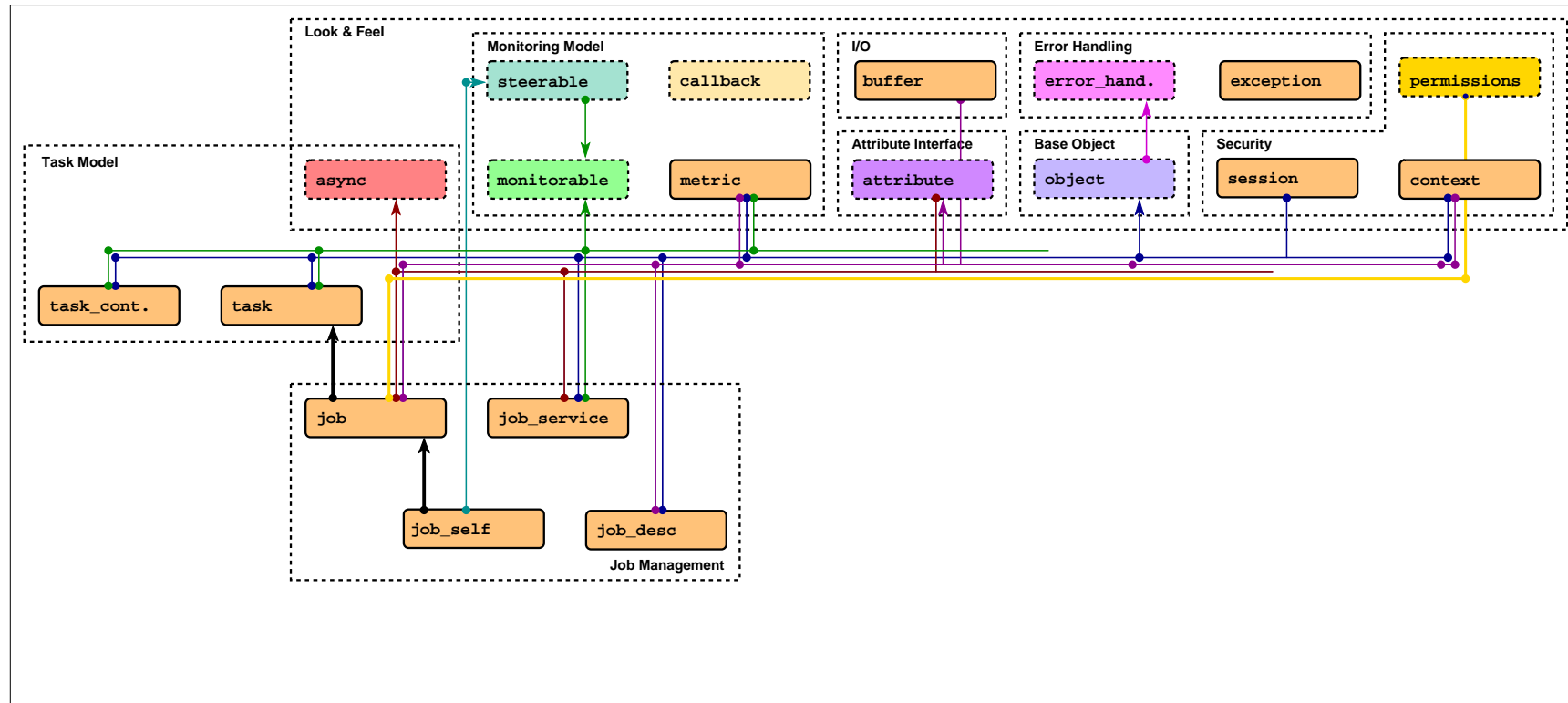
# SAGA: Class hierarchy



## SAGA Look & Feel:

the task model adds asynchronous operations.

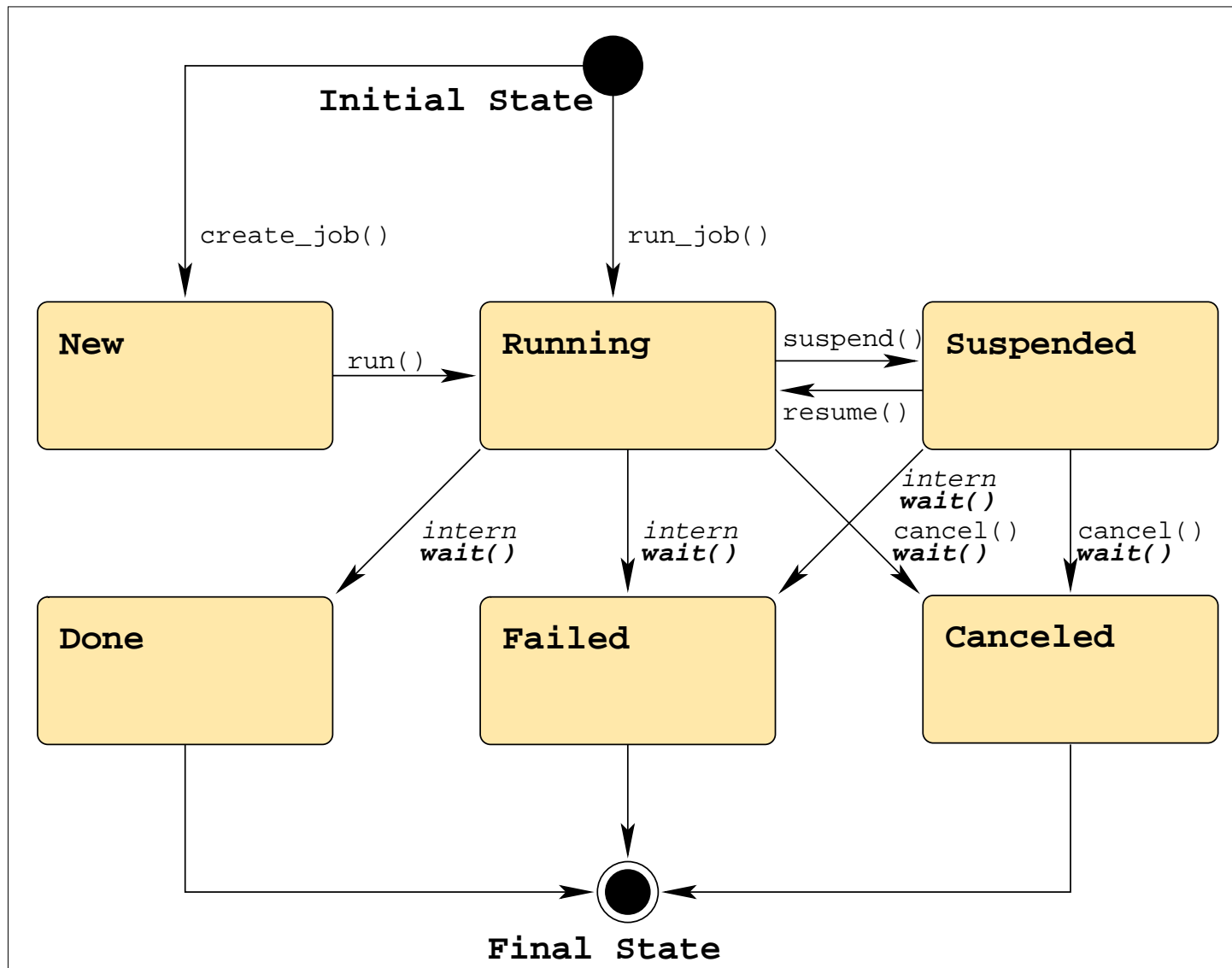
# SAGA: Jobs



# SAGA: Jobs

- `job_service` used `job_description` to create job
- `job_description` attributes are based on JSDL
- state model is based on BES
- `job_self` represents the SAGA application
- job submission and management, but no resource discovery, job dependencies, or workflows

# SAGA: Job States



# SAGA Examples: Jobs

\_\_\_\_\_ job submission \_\_\_\_\_

```
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 Examples: Jobs

jobs (cont.)

```
saga::job j = js.create_job (jd);  
  
j.run ();  
  
j.suspend ();  
j.resume ();  
  
j.checkpoint ();  
  
j.migrate (jd);  
  
j.signal (SIGUSR1);  
  
j.cancel ();
```

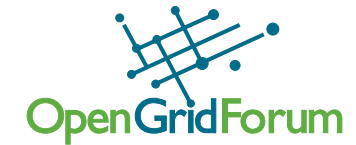
# SAGA Examples: Job Descr.

\_\_\_\_\_ job description - JSDL based \_\_\_\_\_

```
saga::job_description jd;

jd.set_attribute ("Executable",      "/bin/tail");
jd.set_attribute ("Arguments",      "-n, 20, -f, all.log");
jd.set_attribute ("Environment",    "TMPDIR=/tmp/");
jd.set_attribute ("WorkingDirectory", "data/");
jd.set_attribute ("FileTransfer",    "last.log >> all.log");
jd.set_attribute ("Cleanup",        "False");
```

# SAGA Examples: Job Descr.

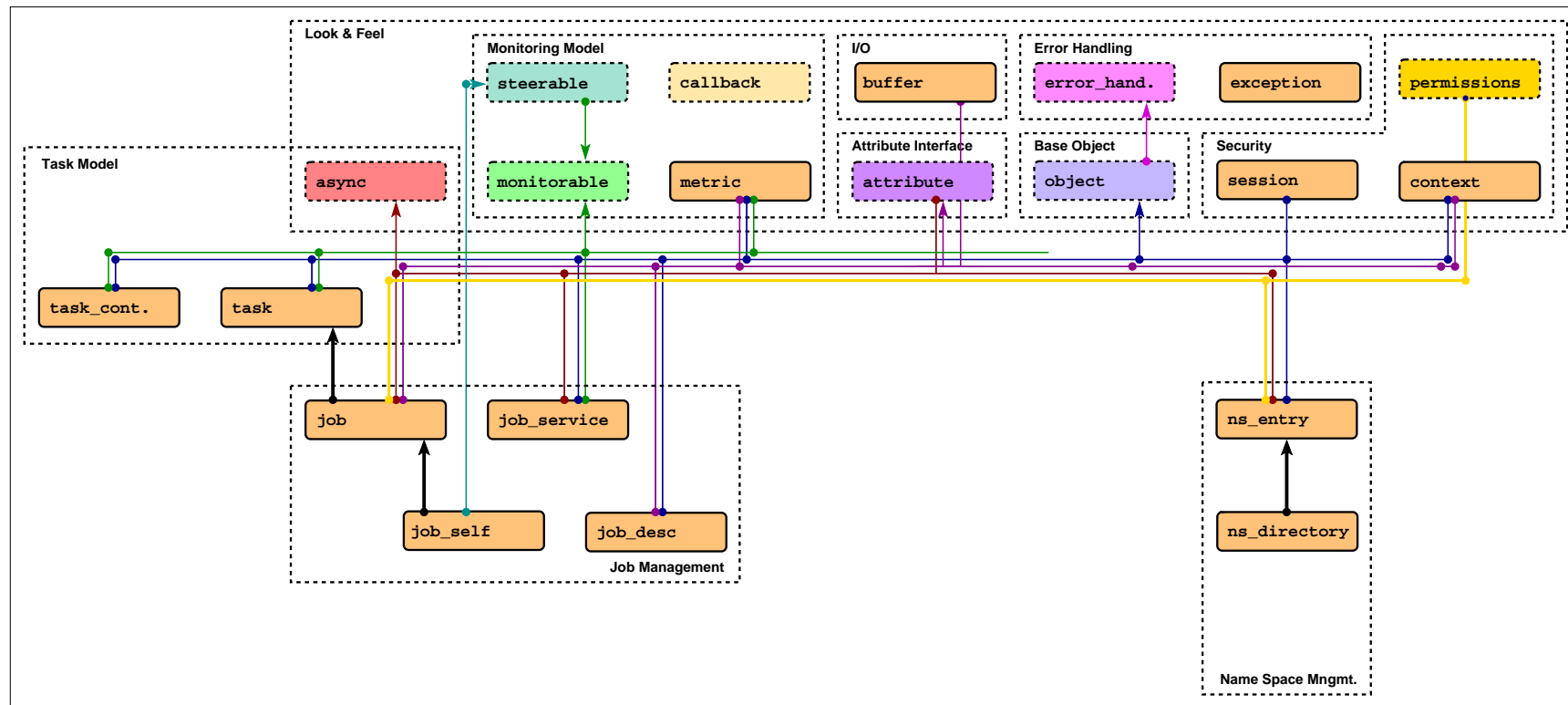


## SAGA JD attributes:

Executable	Environment	<i>JobStartTime</i>
Arguments	WorkingDirectory	TotalCPUTime
SPMDVariation	<i>Interactive</i>	TotalPhysicalMemory
TotalCPUCount	Input	CPUArchitecture
NumberOfProcesses	Output	OperatingSystemType
ProcessesPerHost	Error	CandidateHosts
ThreadsPerProcess	FileTransfer	<i>Queue</i>
	Cleanup	<i>JobContact</i>

*Italic:* not explicitly supported by JSDL

# SAGA: Name Spaces



# SAGA: Name Spaces

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

# SAGA Examples: NameSpaces



name space management

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

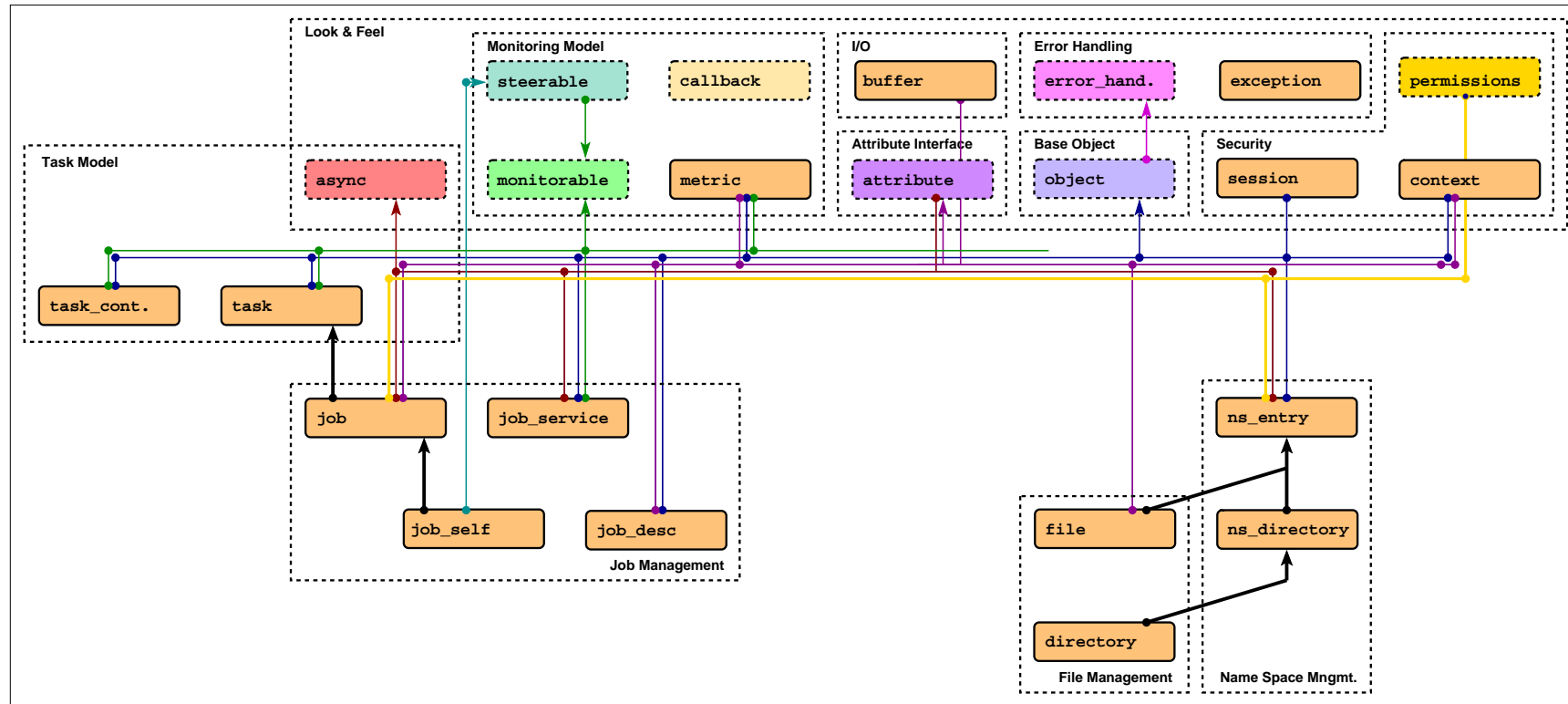
if ( dir.is_entry ("a") && ! dir.is_dir ("a") )
{
    dir.copy ("a", "../b");
    dir.link ("../b", "a", Overwrite);
}

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

saga::ns_dir tmp = dir.open_dir ("tmp/", DeReference);
saga::ns_entry entry = dir.open ("tmp/data.txt");

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

# SAGA: Files



# SAGA: Files

- implements name space interface, and adds access to content of NS entries (files)
- Posix oriented: read, write seek
- Grid optimizations: scattered I/O, pattern based I/O, extended I/O

# SAGA Examples: Files

file access

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

char buf[100];

if ( f.get_size () >= 223 )
{
    int pos = f.seek (123, Current);
    int len = f.read (saga::buffer (buf), 100);
}
```

# SAGA Examples: Files

\_\_\_\_\_ file access - scattered I/O \_\_\_\_\_

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

saga::ivec ivecs[100];

ivecs[i].buffer = saga::buffer ();
ivecs[i].offset = 1;
ivecs[i].leng_in = 10;

if ( f.get_size () >= 223 )
{

    f.read_v (ivecs);
}
```

# SAGA Examples: Files

file access - pattern based I/O

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

char buf[100];

string pattern ("(0,17,36,6,(0,0,2,6))");

if ( f.get_size () >= 223 )
{
    int len = f.read_p (pattern, saga::buffer (buf));
}
```

# SAGA Examples: Files

\_\_\_\_\_ file access - extended I/O \_\_\_\_\_

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

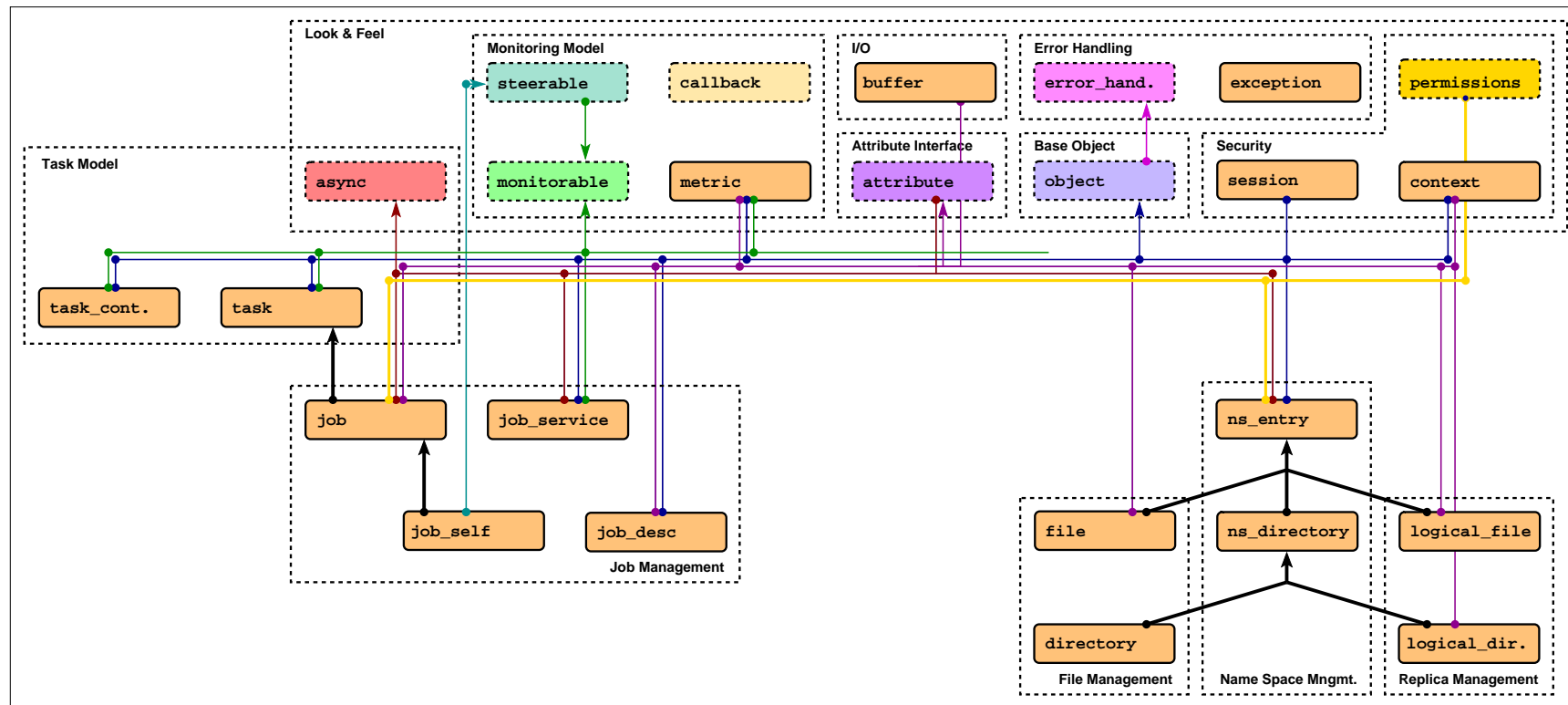
char buf[100];

string mode ("JPEG-crop");
string spec ("coord=0,0,10,10");

if ( f.get_size () >= 223 )
{

    int len = f.read_e (mode, spec, saga::buffer (buf));
}
```

# SAGA: Replicas



# SAGA: Replicas

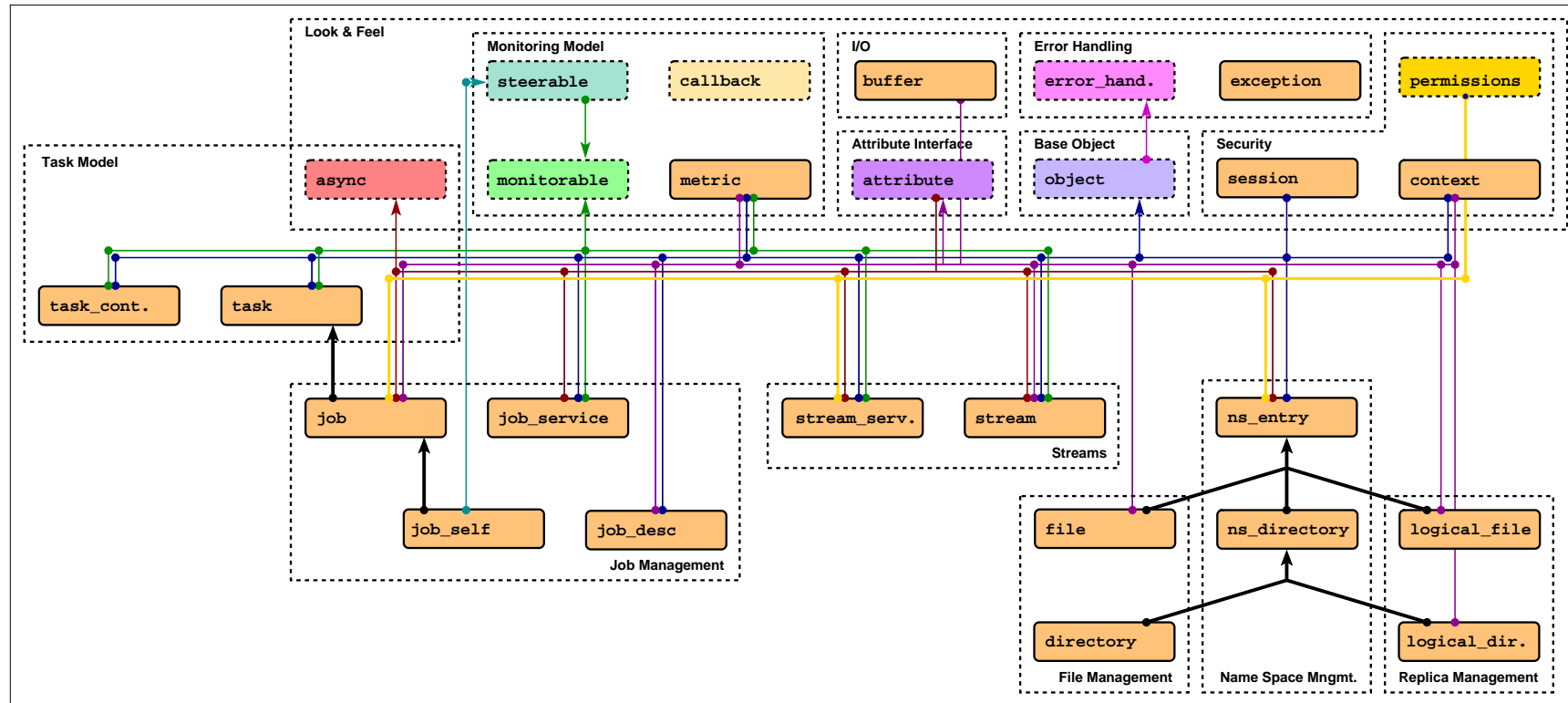
- implements name space interface, and adds access to properties of NS entries (logical files / replicas)
- O/REP oriented: list, add, remove replicas; manage meta data
- Grid optimizations are hidden (replica placement strategies, consistency and version management, . . . )

# SAGA Examples: Replicas

\_\_\_\_\_ replica meta data \_\_\_\_\_

```
saga::logical_directory dir ("raptor://remote.host.net/data/");  
  
list <string> files = dir.find ("*", "type=jpg");  
  
while ( file.size () )  
{  
    saga::logical_file lf (file.pop_front ());  
  
    lf.replicate ("file://localhost/data/all_jpg/", Overwrite);  
}
```

# SAGA: Streams



# SAGA: Streams

- simple and BSD socket oriented
- not supposed to replace MPI etc, but allows for simple application level communication
- potentially slow, but can be implemented efficiently

# SAGA Examples: Streams

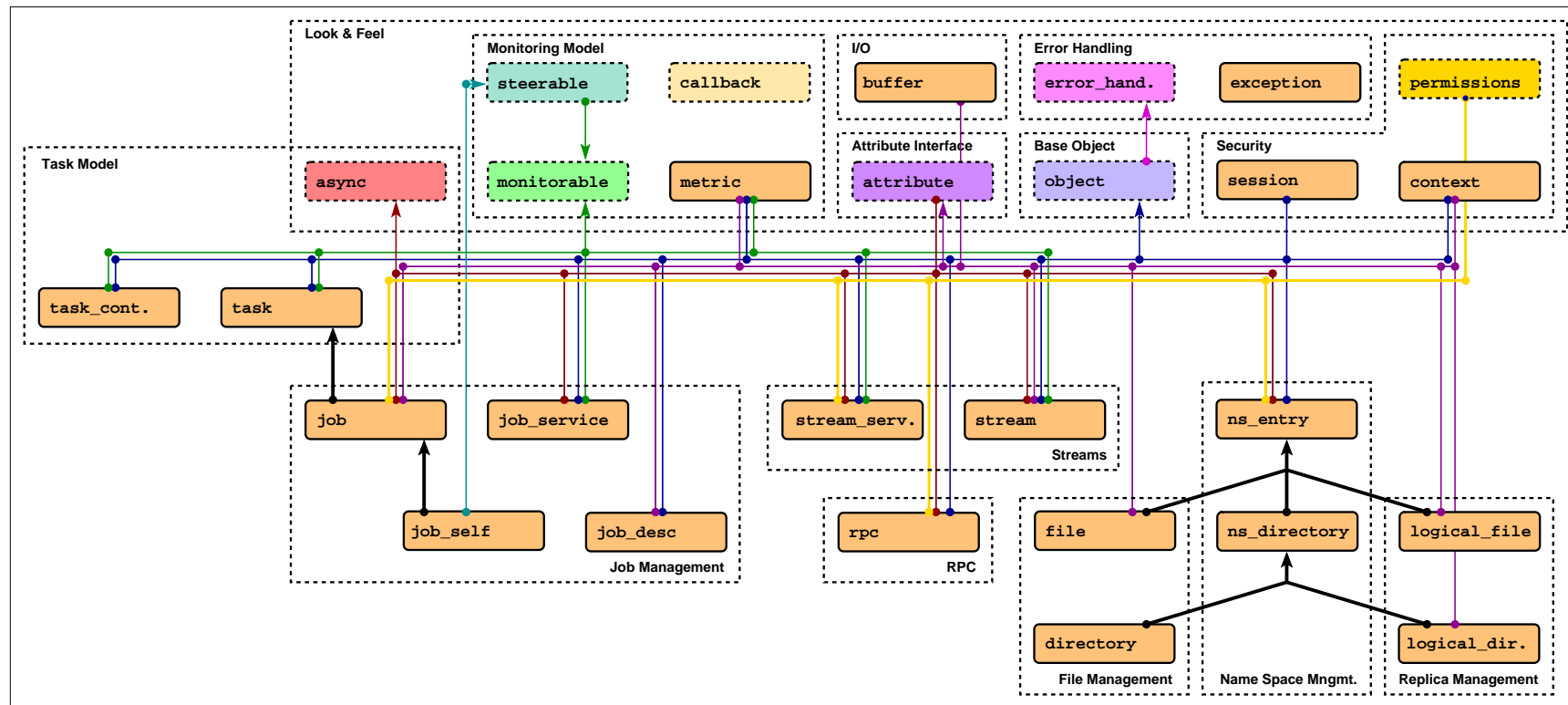
stream server

```
saga::stream_service ss ("tcp://localhost:1234");  
  
saga::stream_client sc = ss.serve ();  
  
sc.write ("Hello client", 13);
```

stream client

```
char buf [13];  
saga::stream_client sc ("tcp://remote.host.net:1234");  
  
sc.connect ();  
sc.read (saga::buffer (buf), 13);  
  
cout << buf << endl;
```

# SAGA: RPC



# SAGA: RPC

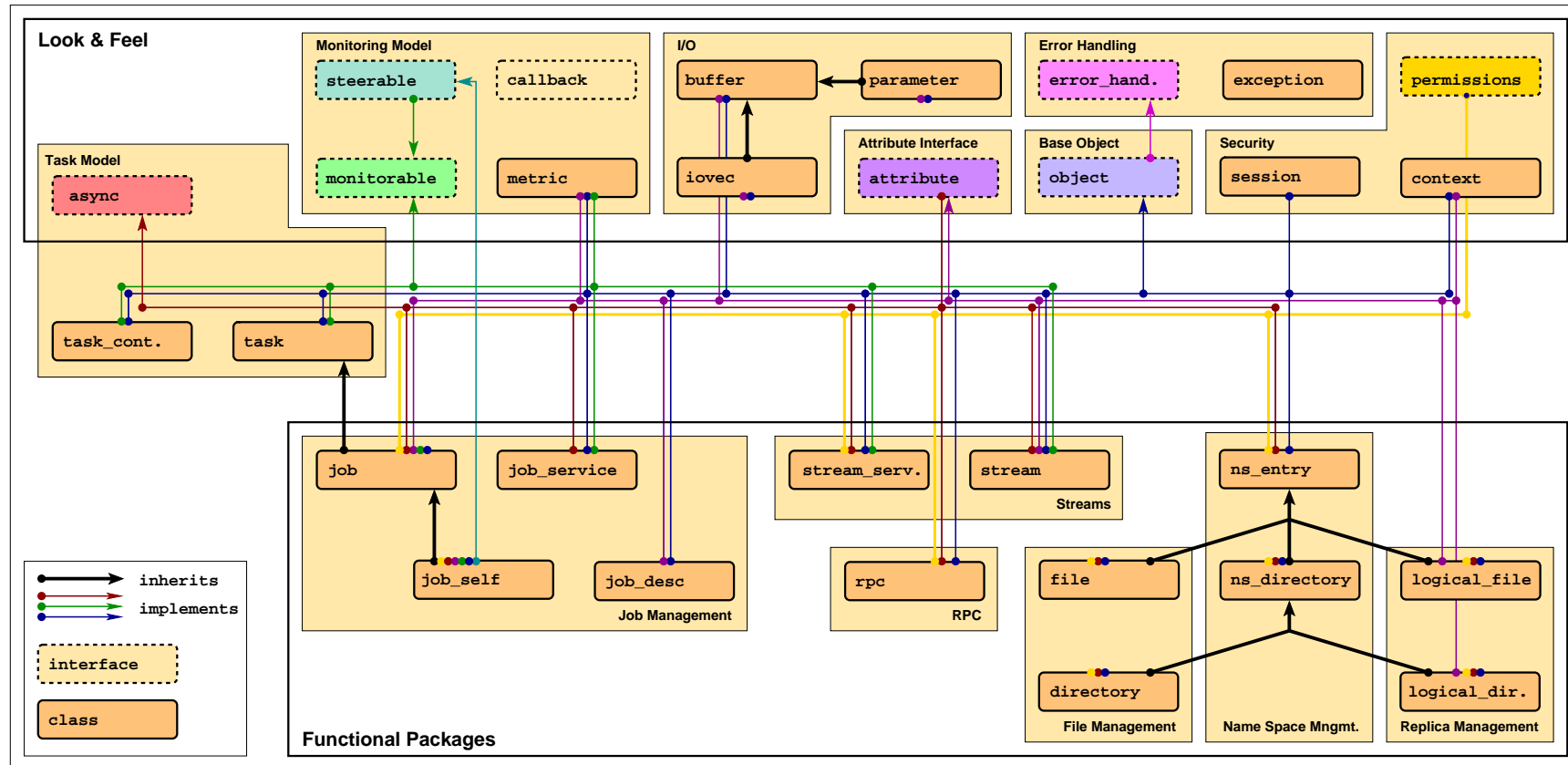
- maps GridRPC standard into the SAGA look & feel
- parameters are stack of structures (similar to scattered I/O)
- future revision will work on optimized data handling

# SAGA Examples: RPC

remote procedure call

```
saga::rpc rpc ("ninf://remote.host.net:1234/random");  
  
list <saga::rpc::parameter> params;  
params.push_back (new saga::rpc::parameter (Out, 10));  
  
rpc.call (params);  
  
cout << "found random number: " << atoi (param.buffer) << endl;  
  
delete (params.pop_front ());
```

# SAGA: Session and Context



# SAGA: Session Management



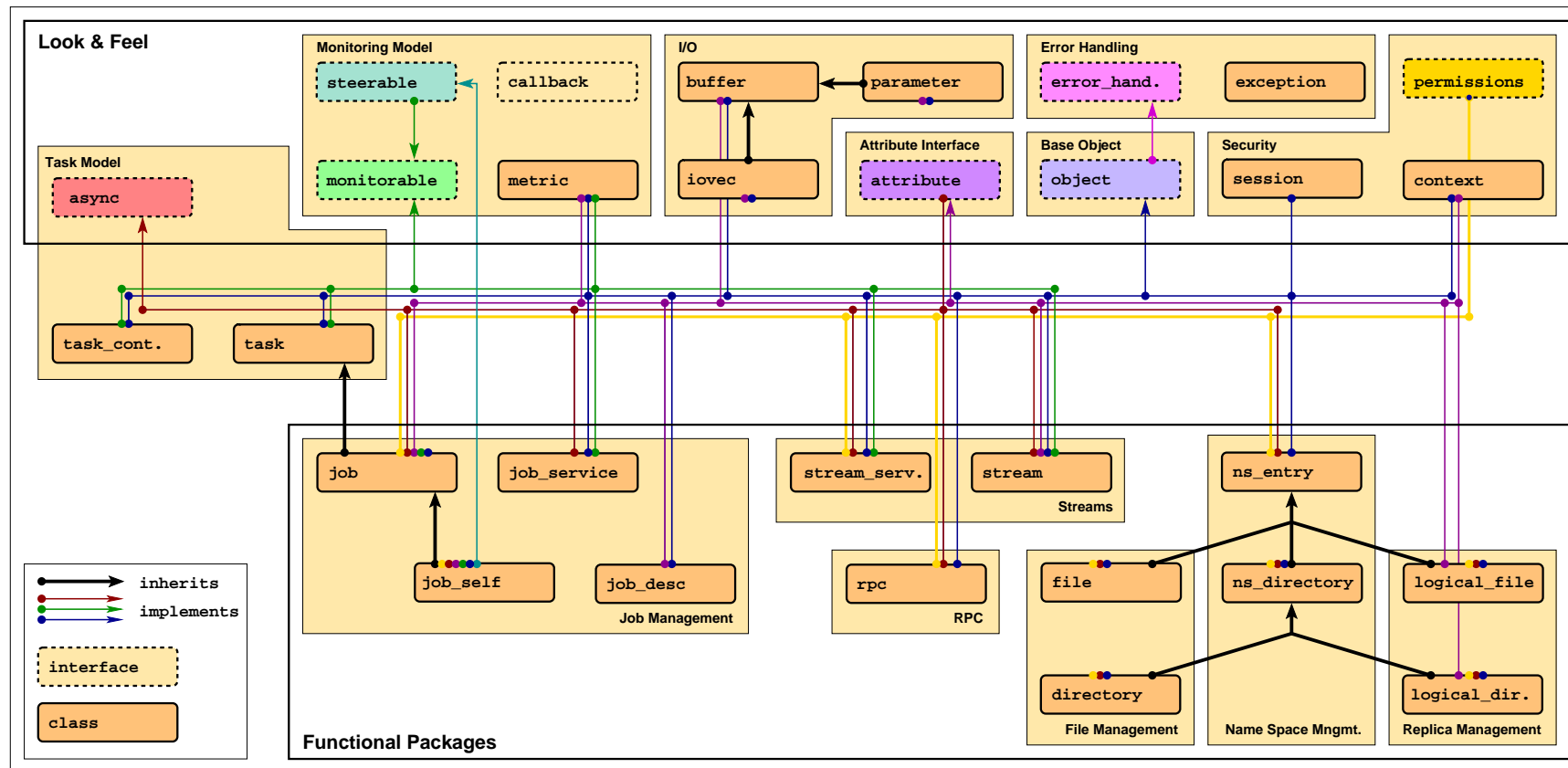
- by default hidden (default session is used)
- session is identified by lifetime of security credentials and by objects in this session (jobs etc.)
- session is used on object creation (optional)
- `saga::context` is used to attach security tokens to a session
- the default session has default contexts

# SAGA Examples: Session

context management

```
saga::context c1 ("Globus");  
saga::context c2 ("Globus");  
  
c2.set_attribute ("UserProxy", "/tmp/x509up_u123.special");  
  
saga::session s;  
  
s.add_context (c1);  
s.add_context (c2);  
  
saga::ns_dir dir (s, "any://remote.host.net/data/");
```

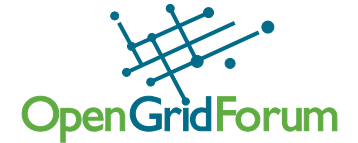
# SAGA: Monitoring



# SAGA: Monitoring

- monitoring of Grid entities (jobs, files, ...)
- monitoring of interactions (task state, notification, ...)
- `monitorables` have metrics
- `metrics` can be pulled, or subscribed to (callbacks)
- some metrics can be written (basic steering)

# SAGA Examples: Monitoring



———— pull monitoring ————

```
saga::job job = js.create_job (jd);

job.run ();

saga::metric m = job.get_metric ("MemoryUsage");

while ( 1 )
{
    cout << "Memory Usage: " << m.get_value () << endl;
    sleep (1);
}
```

# SAGA Examples: Monitoring

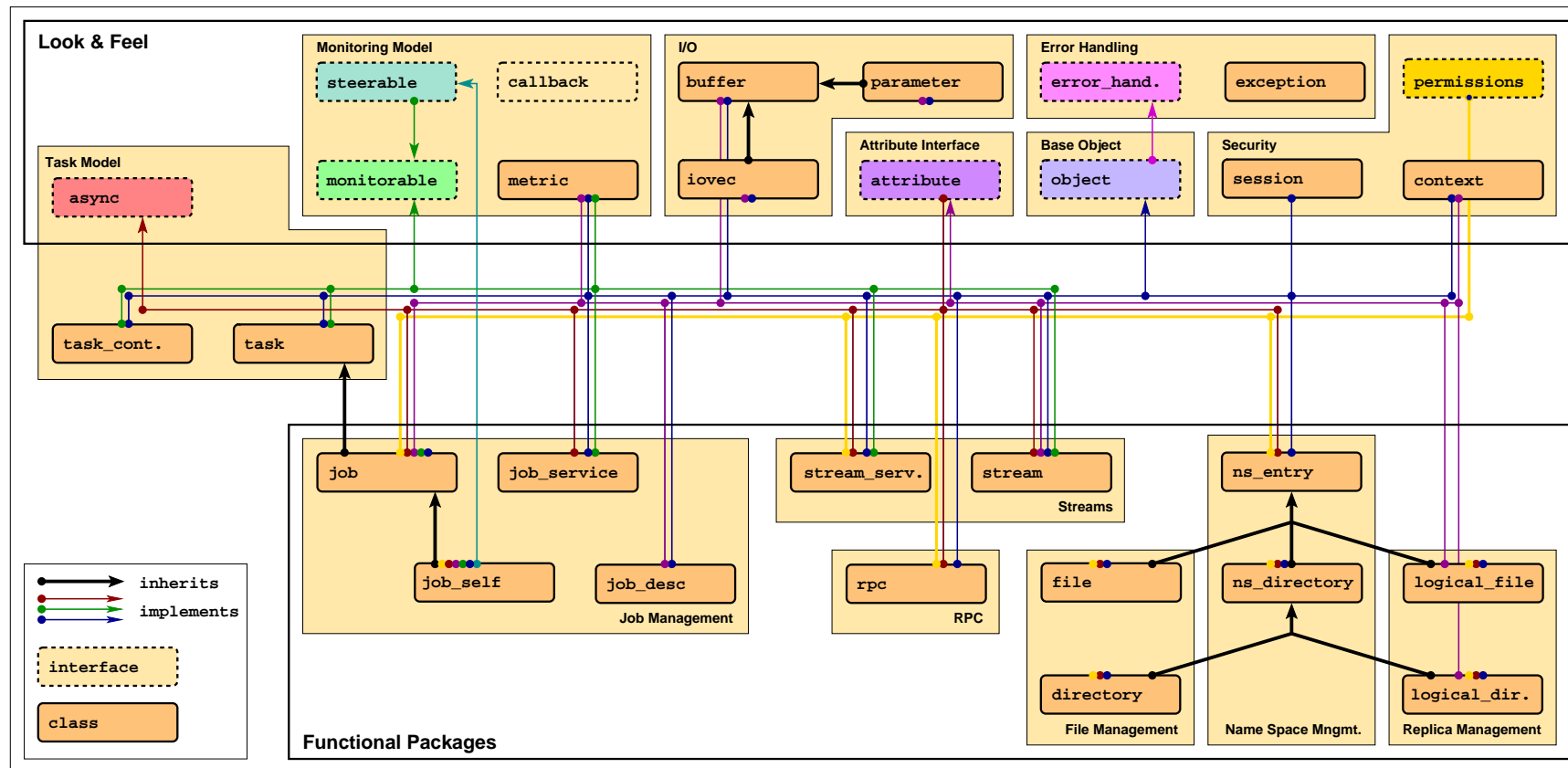


```
class my_cb : public saga::callback
{
public:
    bool cb (saga::monitorable obj,
            saga::metric      m,
            saga::context      c)
    {
        cout << "Memory Usage: " << m.get_value () << endl;
        return (true);
    }
};

my_cb cb;
saga::job job = js.run_job (jd);

job.add_callback ("MemoryUsage", cb);
```

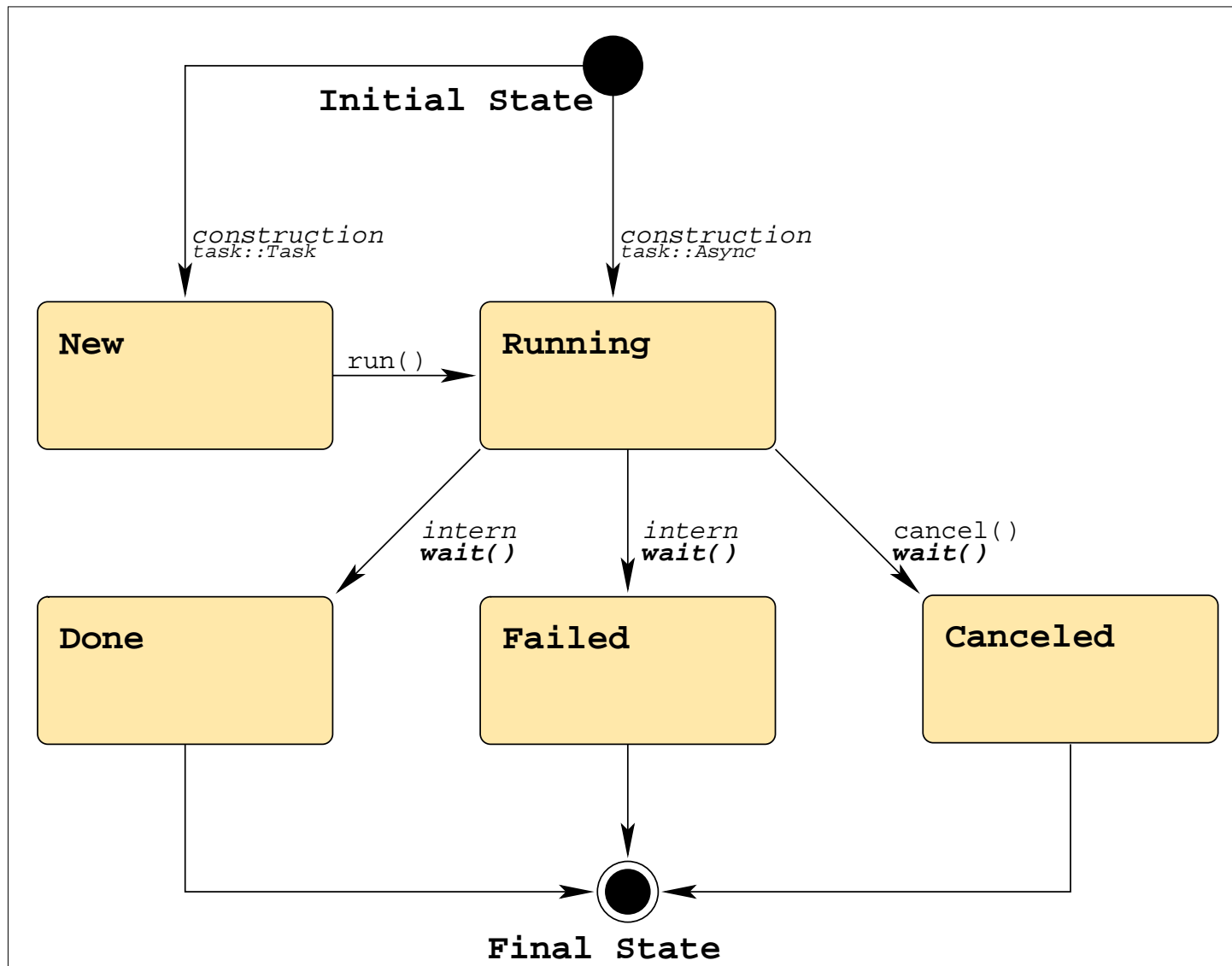
# SAGA: Tasks



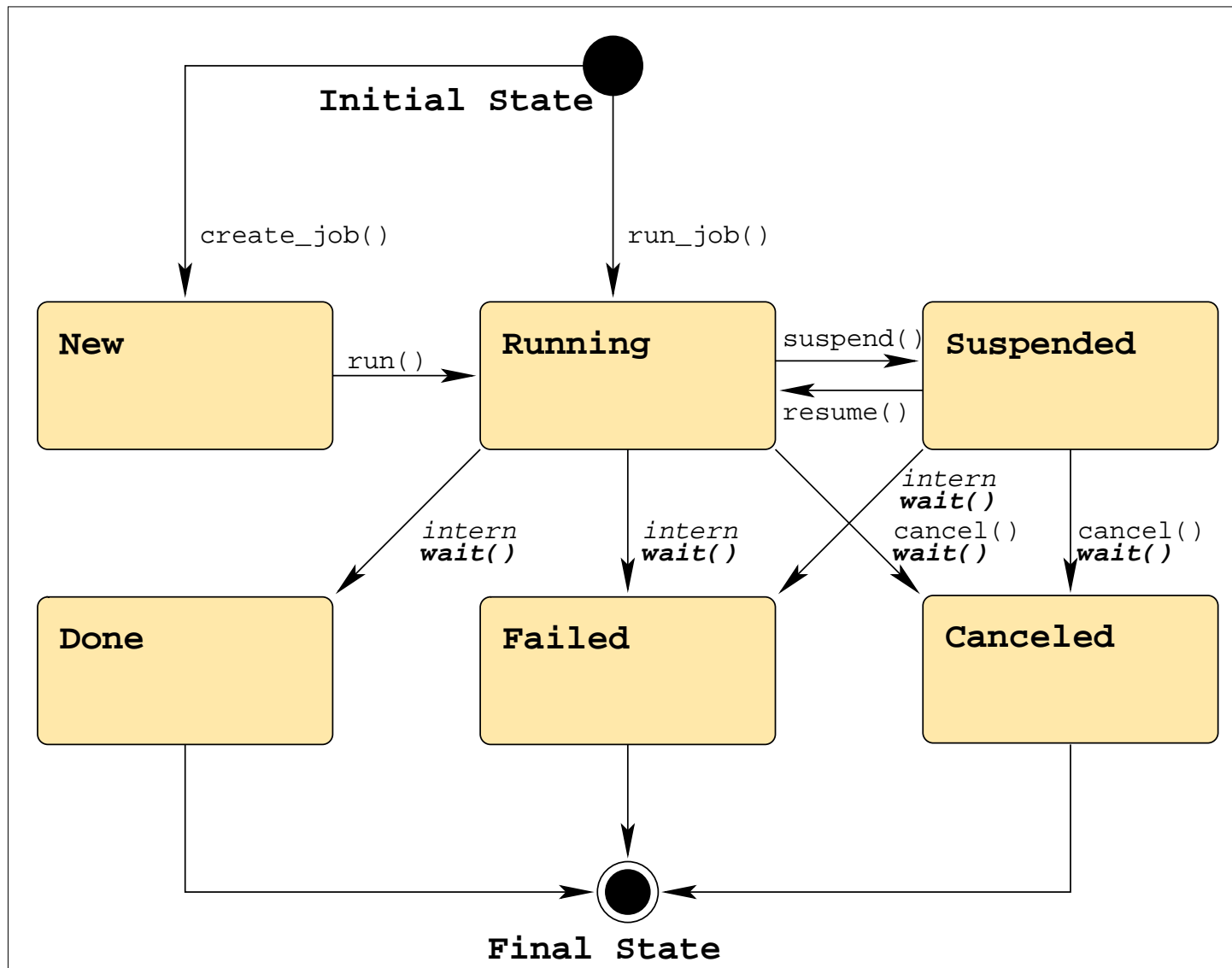
# SAGA: Tasks

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

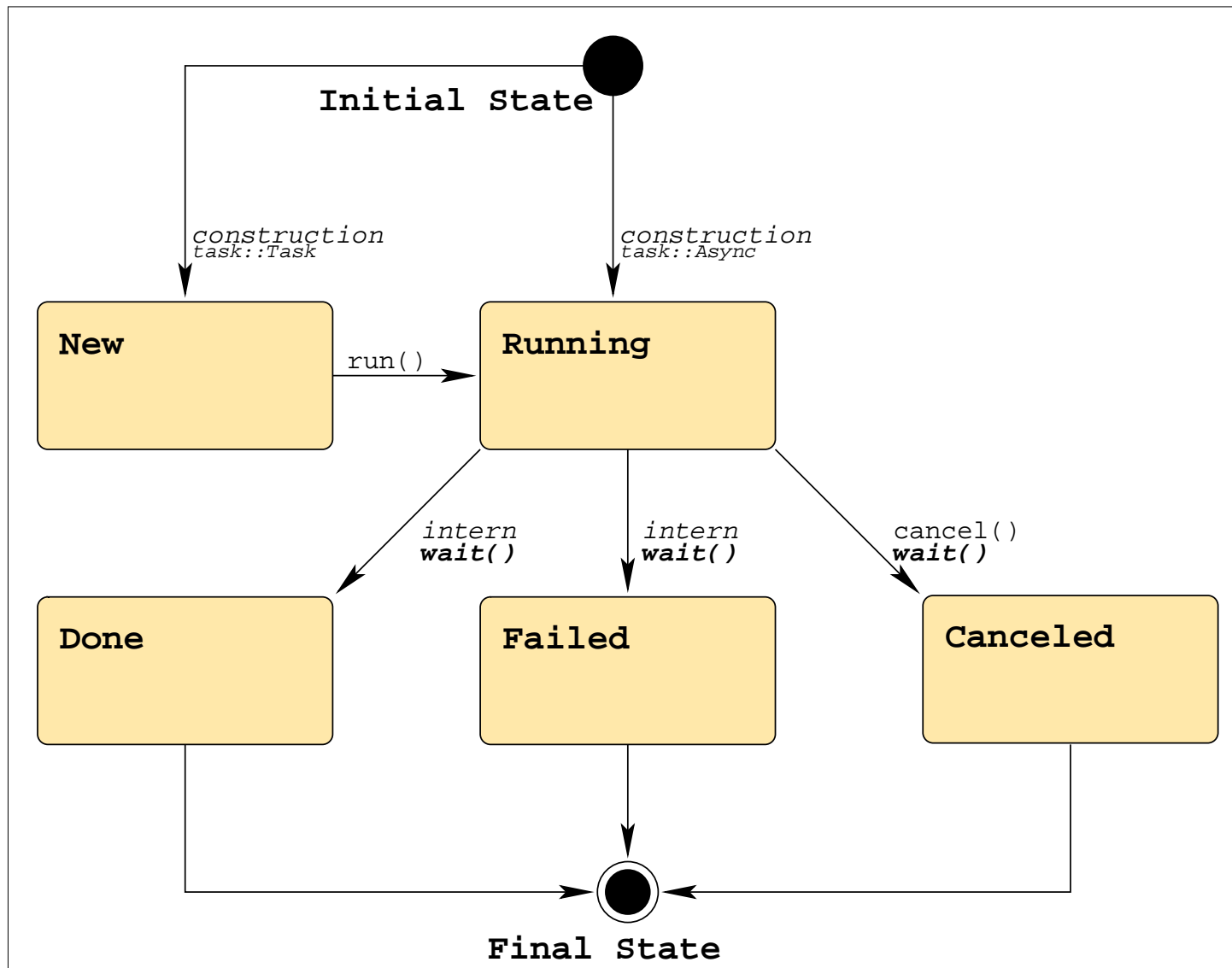
# SAGA: Task States



# SAGA: Job States



# SAGA: Task States



# SAGA: Tasks

- different versions for each method call: sync, async, task
- signature *basically* the same
- differ in state of task representing that method

# SAGA Examples: Tasks

```
_____ tasks (ii) _____  
  
saga::file file ("gsiftp://remote.host.net/data/data.bin");  
  
// normal, synchronous  
ssize_t size_0 = file.get_size ();  
  
// async versions  
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> ();  
  
// wait...  
  
ssize_t size_1 = t1.get_result <ssize_t> ();  
ssize_t size_2 = t2.get_result <ssize_t> ();  
ssize_t size_3 = t3.get_result <ssize_t> ();
```

# SAGA Examples: Tasks

```
tasks (iii)

t3.run ();

cout << t3.get_state () << endl; // Running

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

// t1, t2, t3: Done (or Failed...)
```

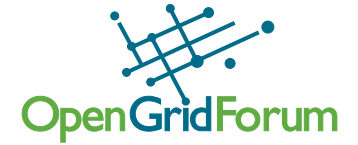
# SAGA Examples: Tasks

tasks container

```
saga::task_container tc;  
  
tc.add (t1);  
tc.add (t2);  
tc.add (t3);  
  
tc.run  ();  
  
saga::task done_task = tc.wait (Any);  
  
tc.wait (All);
```

# SAGA planned extensions

---



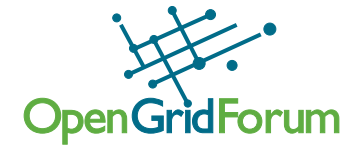
- service discovery
- message based communication
- information service (Advert Service)
- checkpoint & recovery (GridCPR)

# SAGA v2: Service Discovery



- allows to discover service endpoints
- resulting URLs are to be used in SAGA object construction (e.g. for `job_service` etc)
- information model is modeled after GLUE
- led by Steve Fisher (Rutherford Lab, UK)

# SAGA v2: Service Discovery



## Service Discovery

```
saga::discoverer d (session_handle);

string svc_filter = "Type = 'RB' AND NAME = 'CERN-PROD-rb'";
string vo_filter  = "VO IN ('atlas', 'dteam')";
string data_filter = "RunningJobs > 10"

vector <saga::service_description> slist =
    d.list_services (svc_filter, vo_filter, data_filter);

cout << "Total number of services found = "
    << slist.size()
    << endl;
```

# SAGA v2: Message Bus

- cover a variety of communication patterns
  - reliable/unreliable
  - point-2-point/pub-sub
  - ordered/unordered
  - with/without checksums
  - ...
- implies protocols, but is silent about interop
- async zero copy implementation is possible

# SAGA v2: Messages

———— Messaging server ————

```
saga::sender snd ("tcp://localhost:1234",  
                 Reliable | Ordered | P2P);  
saga::msg msg ("Hellow World");  
  
sc.send (msg);
```

———— Messaging client ————

```
saga::receiver rec ("tcp://remote.host.net:1234",  
                   Reliable | Ordered | P2P);  
  
saga::msg = rec.receive (); // internal buffer allocation
```

# SAGA v2: Adverts

- persistent storage of application level information
- semantics of information undefined (app!)
- possibly allows storage of serialized SAGA objects (object persistency)

# SAGA v2: Adverts

## Adverts

```
saga::advert_directory adir ("any//remote.host.net/data/");  
  
list <string> adverts = adir.find ("*", "type=jpg");  
  
while ( adverts.size () )  
{  
    saga::advert ad (averts.pop_front ());  
  
    ad.get_attribute ("description");  
}
```

# SAGA v2: Adverts

## Adverts

```
saga::file    f (url);  
saga::advert  ad ("any//remote.host.net/my_files/", Create);  
  
ad.attach ("my_file", f);
```

```
-----  
  
saga::advert  ad ("any//remote.host.net/my_files/");  
saga::file    f = ad.get_attachement ("my_file");
```

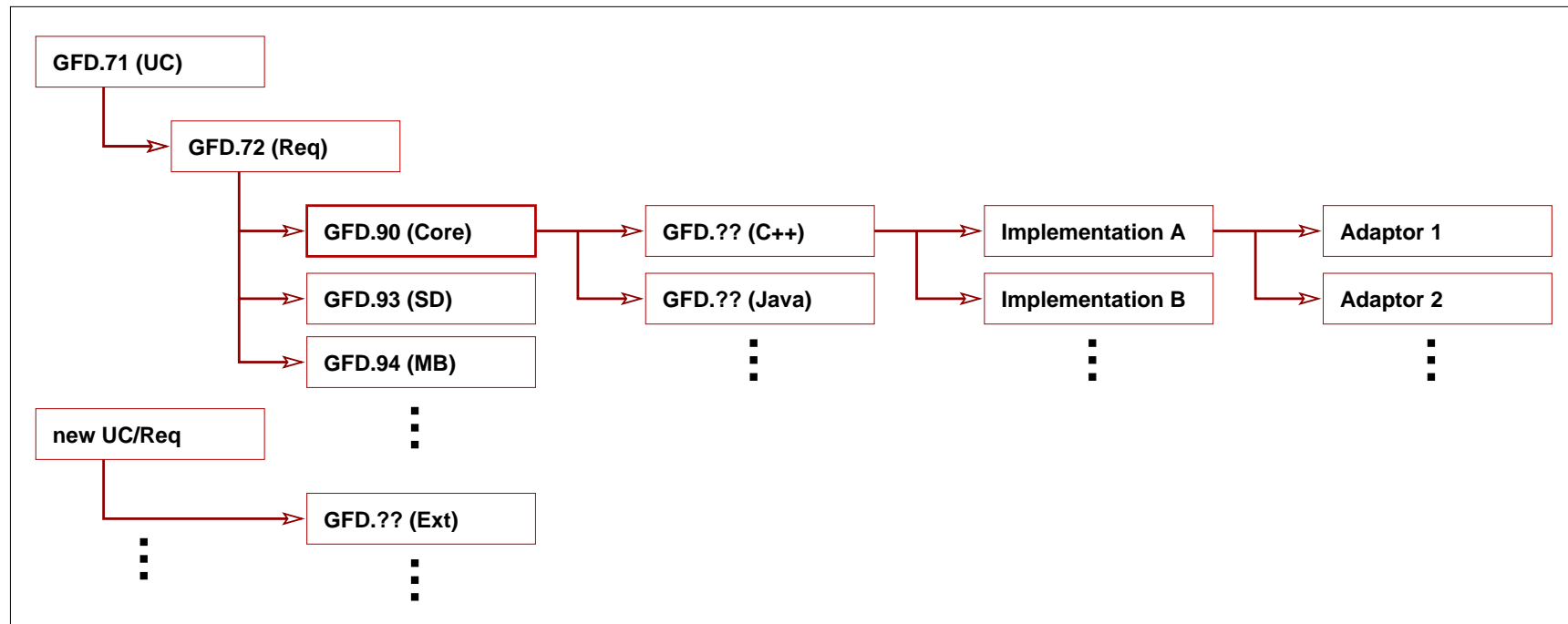
# SAGA v2: CPR

- no examples yet, API in flux
- allows to manage (find, move, stage, archive) checkpoints
- allows to trigger checkpointing of jobs
- probably name space based, with notification on CP creation

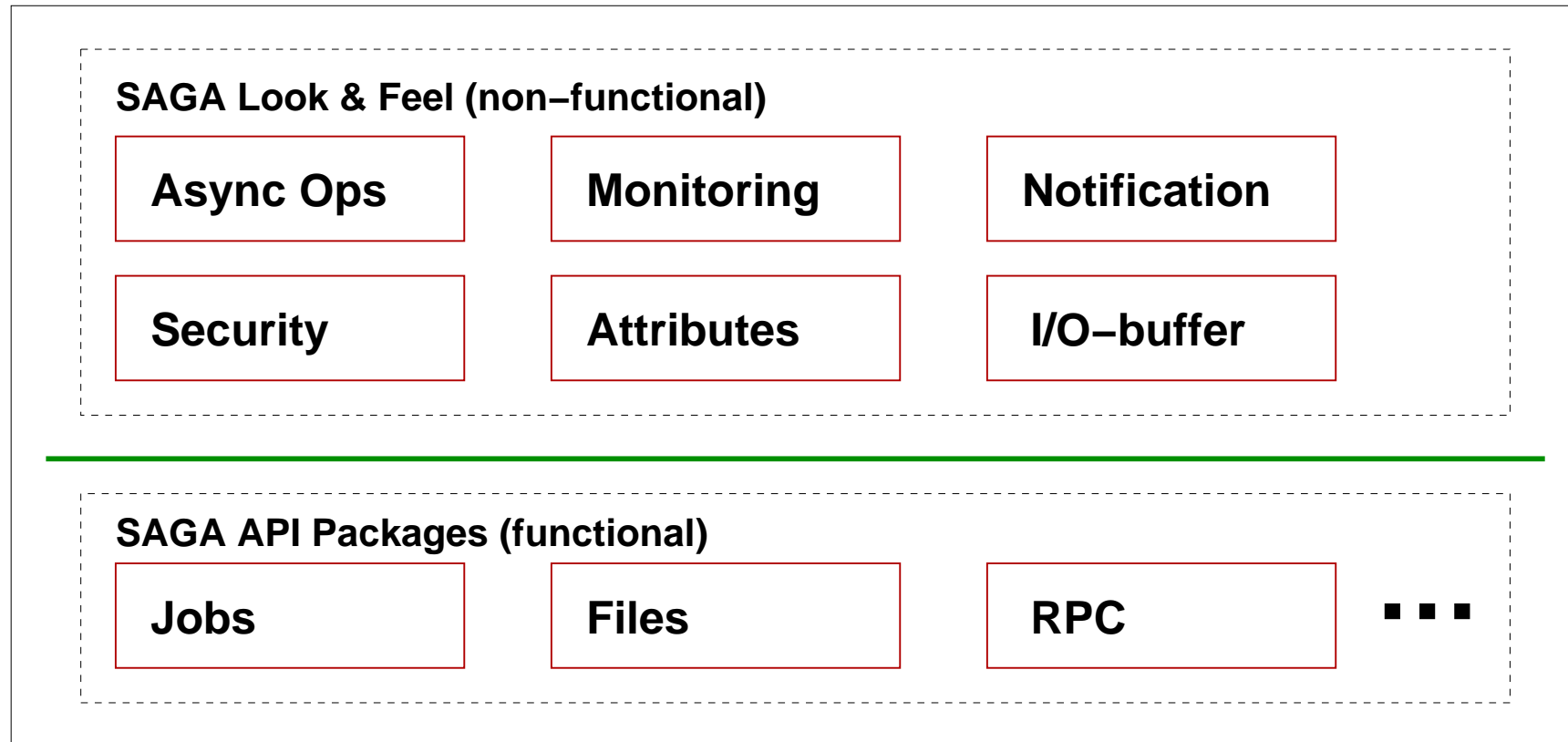
# Questions about API?

# Comments?

# SAGA: Structure



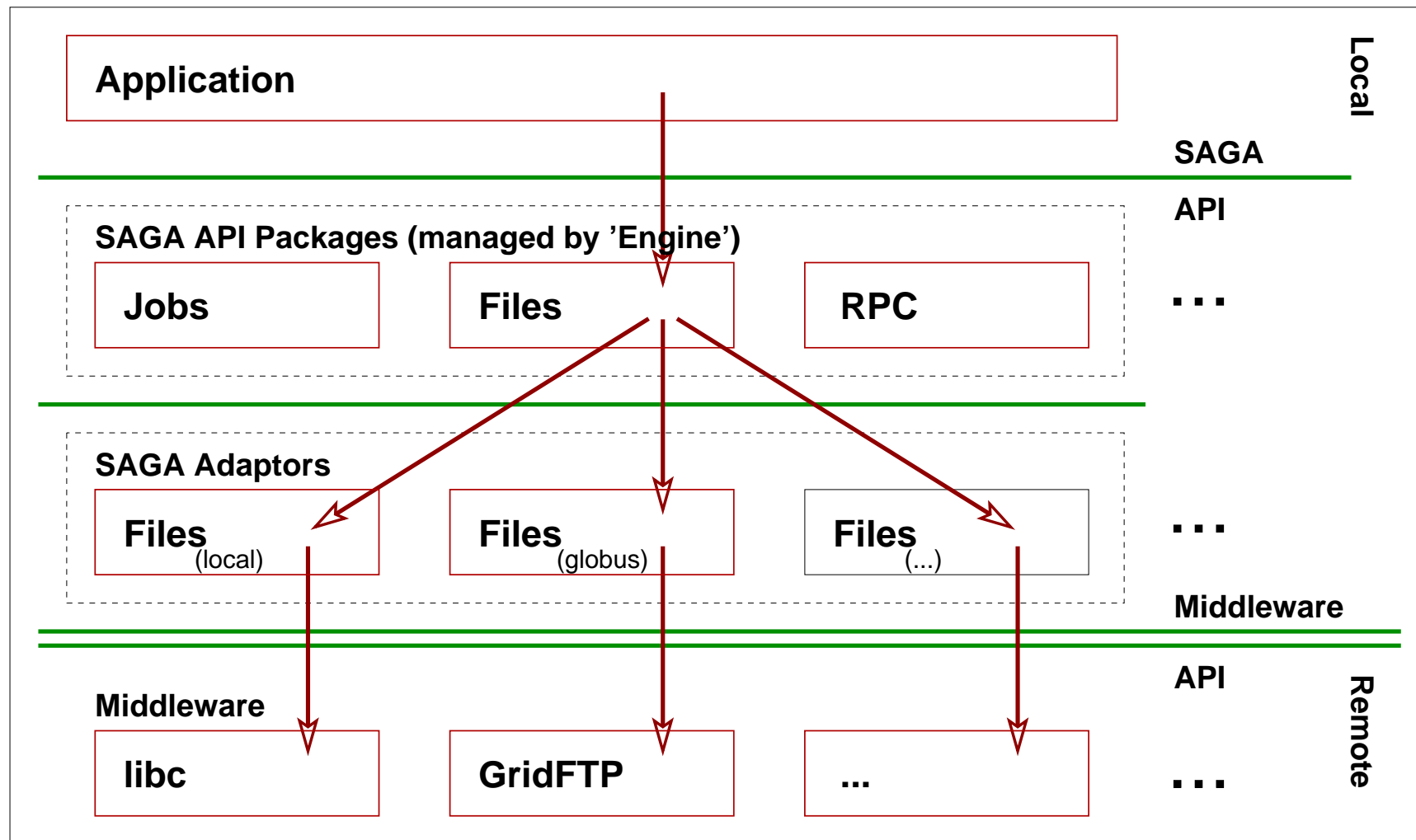
# SAGA: Structure/Scope



# Implementations

- implementations will reflect the SAGA structure, and are as extensible as the spec
- reference implementations in C++ and Java do that
- reference implementations also bind to multiple backends (adaptor per package)

# SAGA: Structure/Scope





# Implementation Status – C++



- reference implementation by CCT/LSU
- engine is **done**
- packages: all from core spec + adverts **done**
- other language bindings **planned** on top (C, Python, Perl, .Net/C#)
- working hard on getting documentation in place

# C++ Status - Adaptors

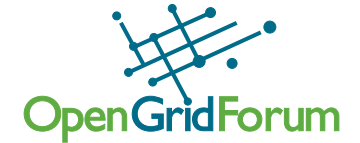
- local adaptors: **done**
- Globus adaptors (2.x/3.x - pre-WS): **Beta timeline:** next weeks
- GridSam adaptors (OMII-UK): **Alpha timeline:** demo at SC07
- LSF started
- wish list: Globus-WS (4.x), gLite, Unicore, ssh, ...

# C++ Implementation Details



- Requirements
  - boost (1.33.1, 1.34.1, ...)
  - optional: soci, sqlite3, xmlrpc, Globus, gSoap
- Installation
  - `http://saga.cct.lsu.edu/`
  - `svn co https://svn.cct.lsu.edu/repos/saga/`
  - `cd saga/trunk`
  - `./configure --prefix='pwd' /install && make && make install`
  - `setenv SAGA_LOCATION, LD_LIBRARY_PATH`
  - **Look at the examples/tutorial!**

# Contact / Acks



<http://forge.ggf.org/sf/projects/saga-rg>

<http://forge.ggf.org/sf/projects/saga-core-wg>

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

## **Supported by:**

CCT/LSU, VU, GridLab, CoreGrid, OMII-UK, VLe,  
XtreemOS, OGF

## **Thanks to:**

Hartmut Kaiser / Ole Weidner (CCT/LSU)

Ceriel Jacobs / Kees Verstoep / Rob v. Nieuwpoort (VU)

# Questions?