THE UNIVERSITY of EDINBURGH
informatics

edinburgh
data-intensive
research

Beyond the SAGA

From Simple APIs to Dynamic Abstractions

Outline

I. A Simple API for Grid Applications

II. Distributed Programming Abstractions (DPA)

III. A Good Research Idea ?

IV. A *Somewhat* Concrete Plan

What is SAGA ?

- Simple API for Grid (Distributed) Application
- Open Grid Forum community standard (GFD-R-P.90)
  - Describes a language-independent (SIDL), object oriented API for high-level tasks *considered useful* in distributed applications, like job submission, file transfer, communication, etc...
- Implementations of the GFD-R-P.90 standard:
  - JSAGA (Centre de Calcul IN2P3/CNRS, Lyon, France)
  - JavaSAGA (Vrije Universiteit, Amsterdam, NL)
  - C++ / Python Implementation (LSU, Baton Rouge, USA)

A Brief History

edinburgh data-intensive research

- 1988: Start of the Condor project
- 1998: Globus 1.0 released
- 2002: GridLab's Grid Application Toolkit (GAT)
- 2004: OGF (GGF) SAGA working group formed
- 2006: SAGA C++ development of the reference implementation starts at Louisiana State University
- 2010: GFD-90 1.0 standard (SAGA) released by OGF
- 2011: SAGA is being deployed across US TeraGrid, deployment on EGI, FutureGrid and XD is under review

A Brief History

edinburgh data-intensive research

A Brief History

A Brief History

edinburgh data-intensive research

- 1988: Start of the Condor project
- 1998: Globus 1.0 released
- 2002: GridLab's Grid Application Toolkit (GAT)
- 2004: OGF (GGF) SAGA working group formed
- 2006: SAGA C++ development of the reference implementation starts at Louisiana State University
- 2010: GFD-90 1.0 standard (SAGA) released by OGF
- 2011: SAGA is being deployed across US TeraGrid, deployment on EGI, FutureGrid and XD is under review

Why a Middleware-Independent API ?

edinburgh data-intensive research

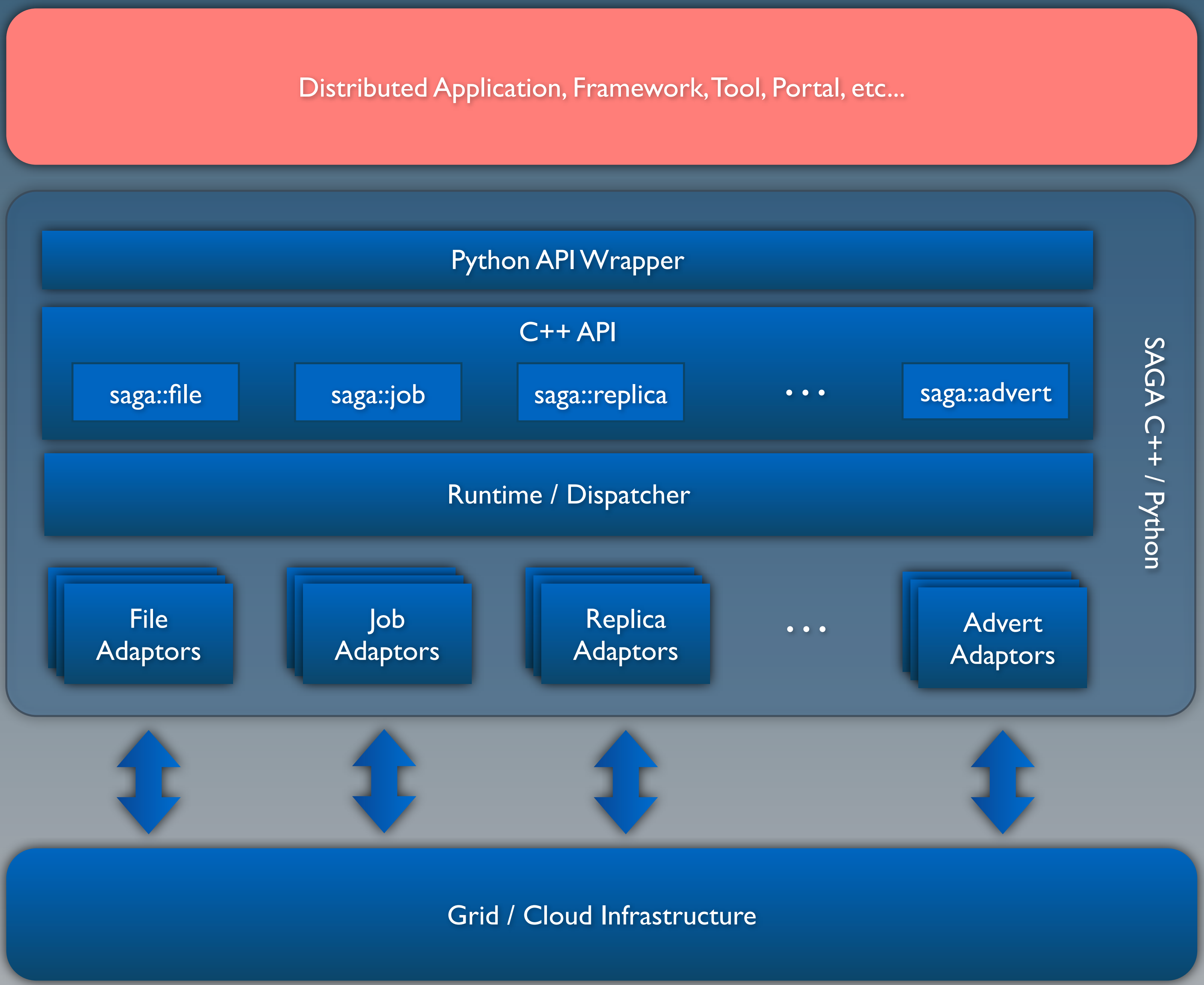
- (Lifetime of applications) > (Lifetime of infrastructure and interfaces)
- Portability / adaptability = protection of assets
- Opens new opportunities for large-scale distributed systems research and evaluation
- Newly emerging extreme-scale simulations may have to span (scale-out) across several different infrastructures
- Distributed computing's counterpart of MPI ?

LSU’s C++ / Python SAGA

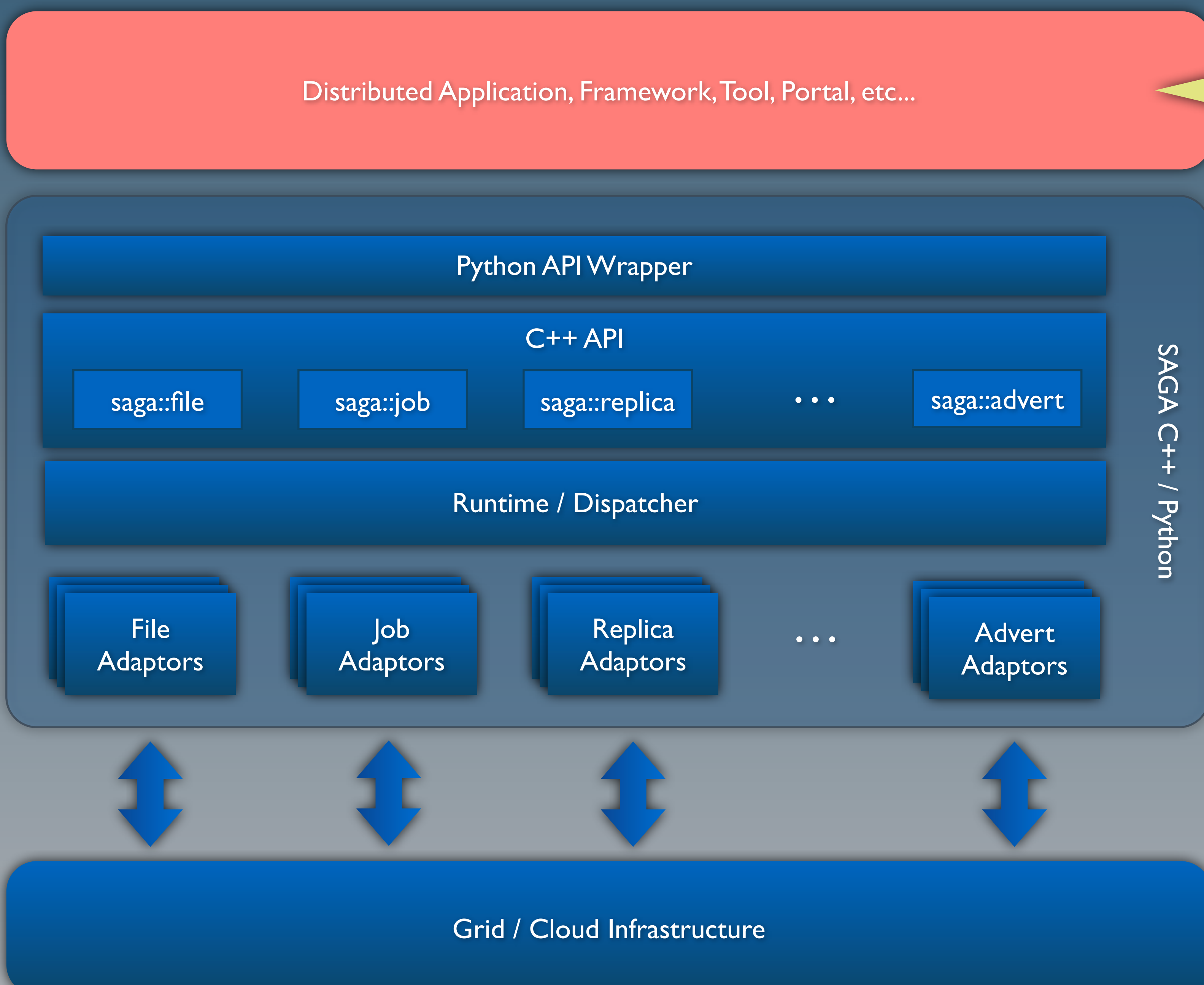
- Designed after the *adaptor pattern*
- A set of C++ libraries and headers grouped into functional packages
- *Adaptors* (plug-ins) that provide access to distributed middleware (Globus, gLite, Condor, etc...)
- A light-weight runtime/dispatcher that the right adaptor for an API call at runtime
- No services, daemons, etc...

How Does it Work ?

How Does it Work ?

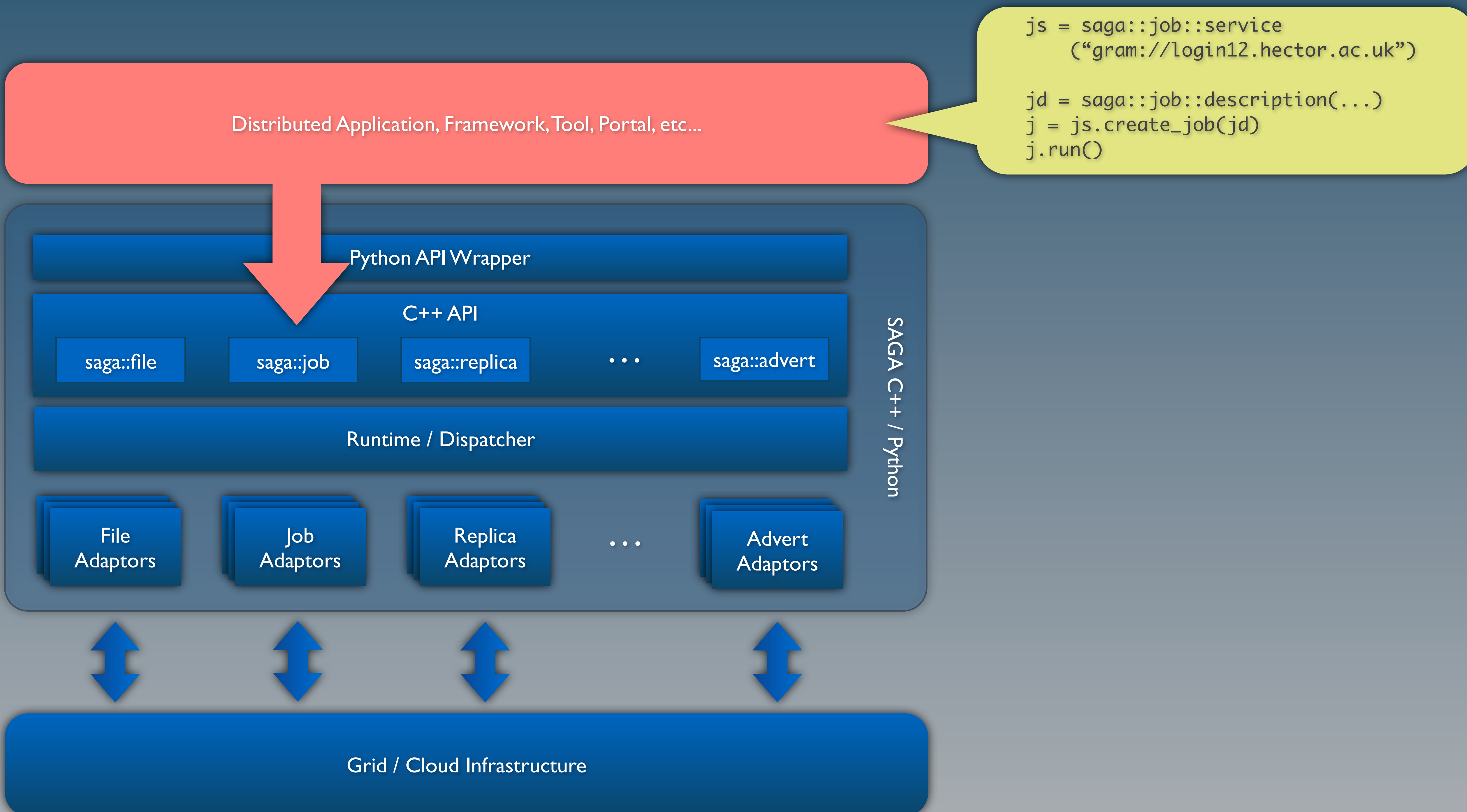


How Does it Work ?



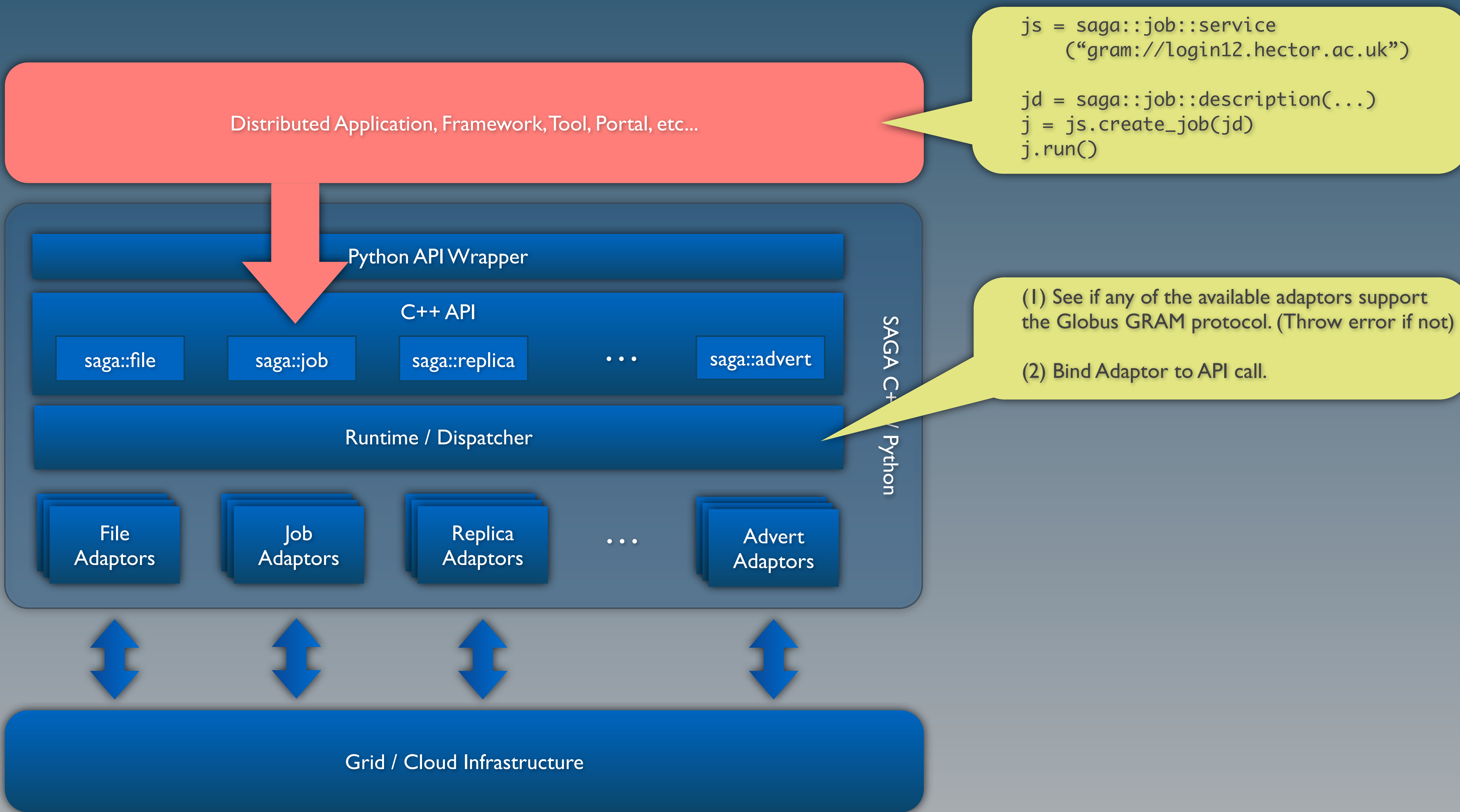

```
js = saga::job::service
    ("gram://login12.hector.ac.uk")

jd = saga::job::description(...)
j = js.create_job(jd)
j.run()
```

How Does it Work ?



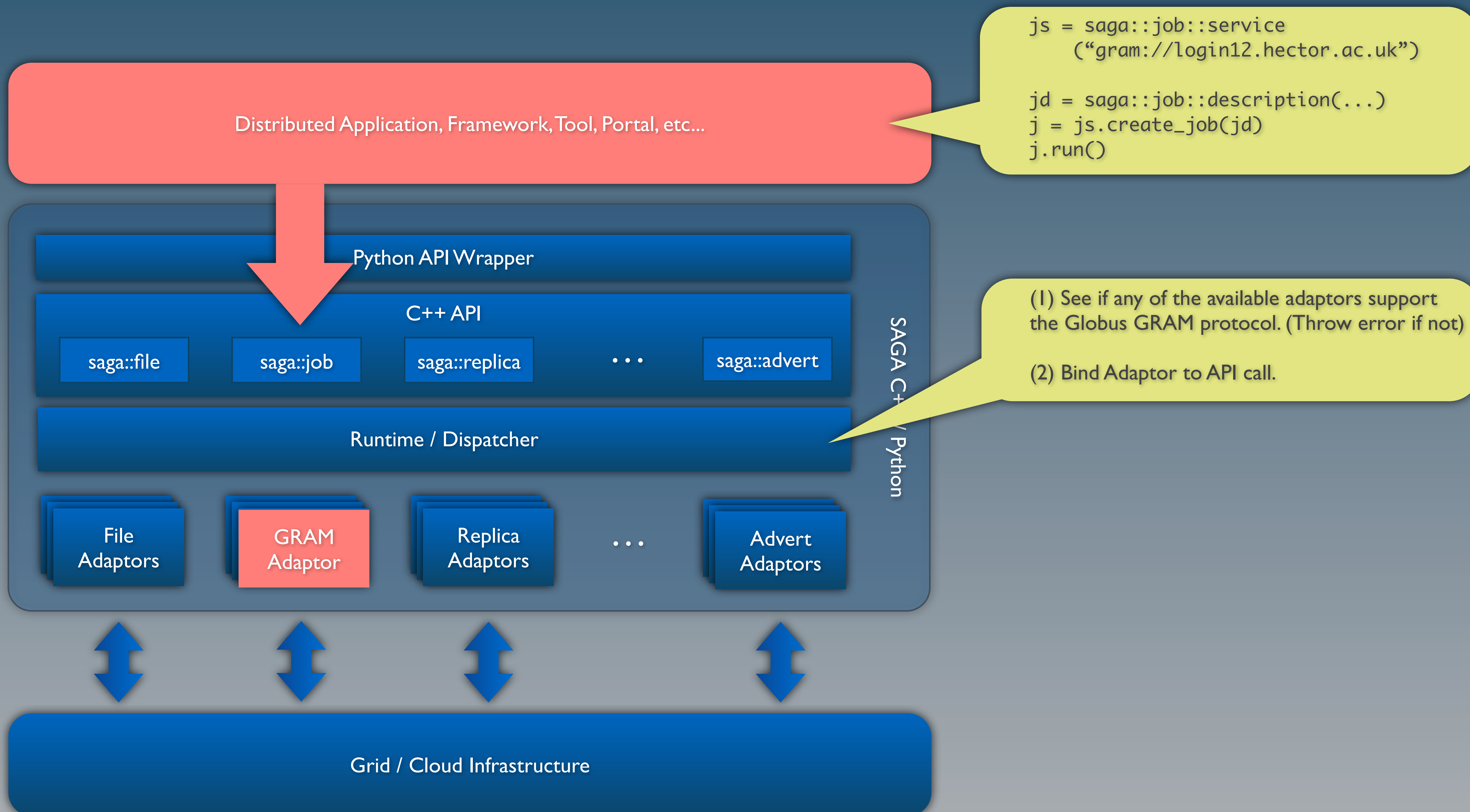
How Does it Work ?

edinburgh
data-intensive
research

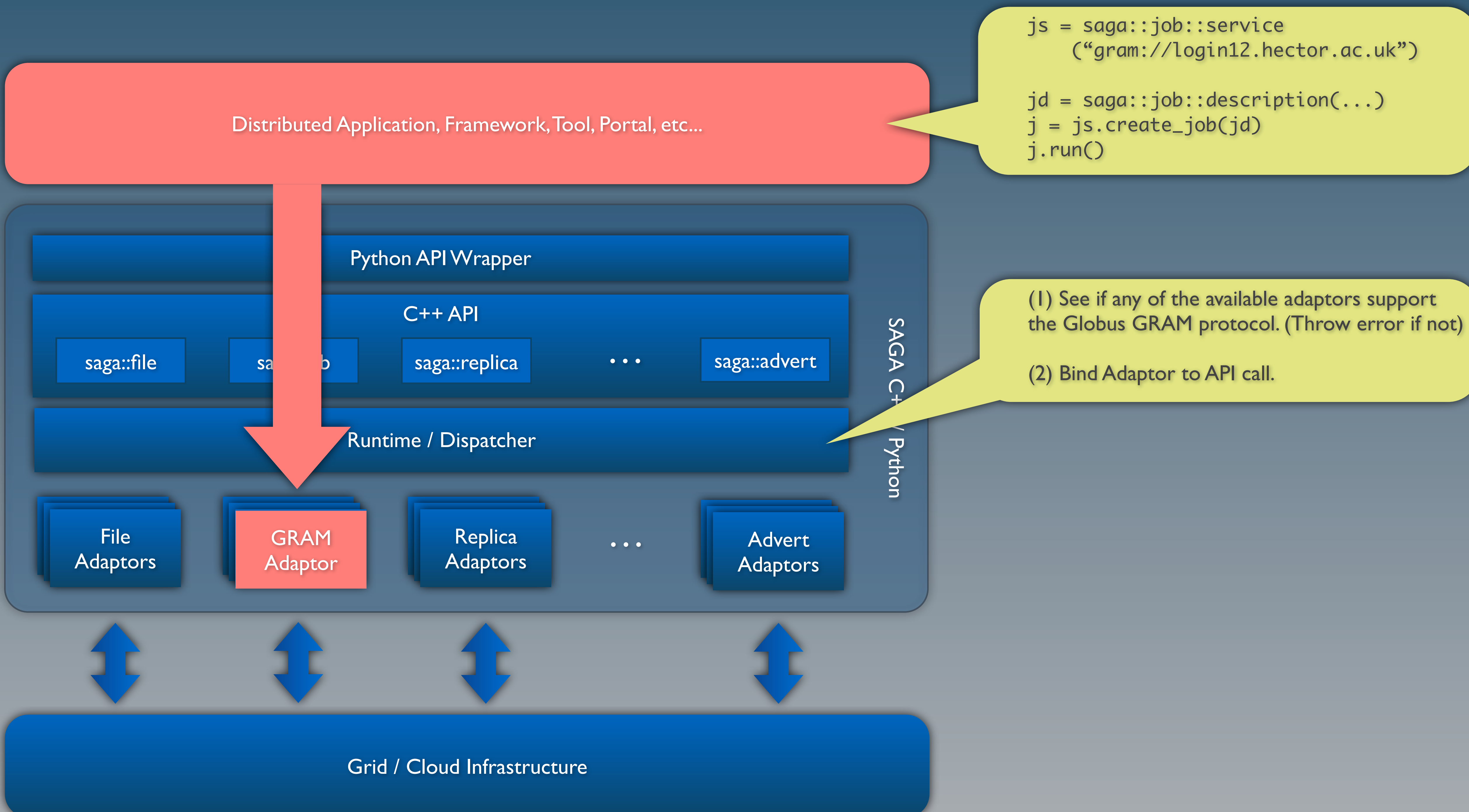


How Does it Work ?

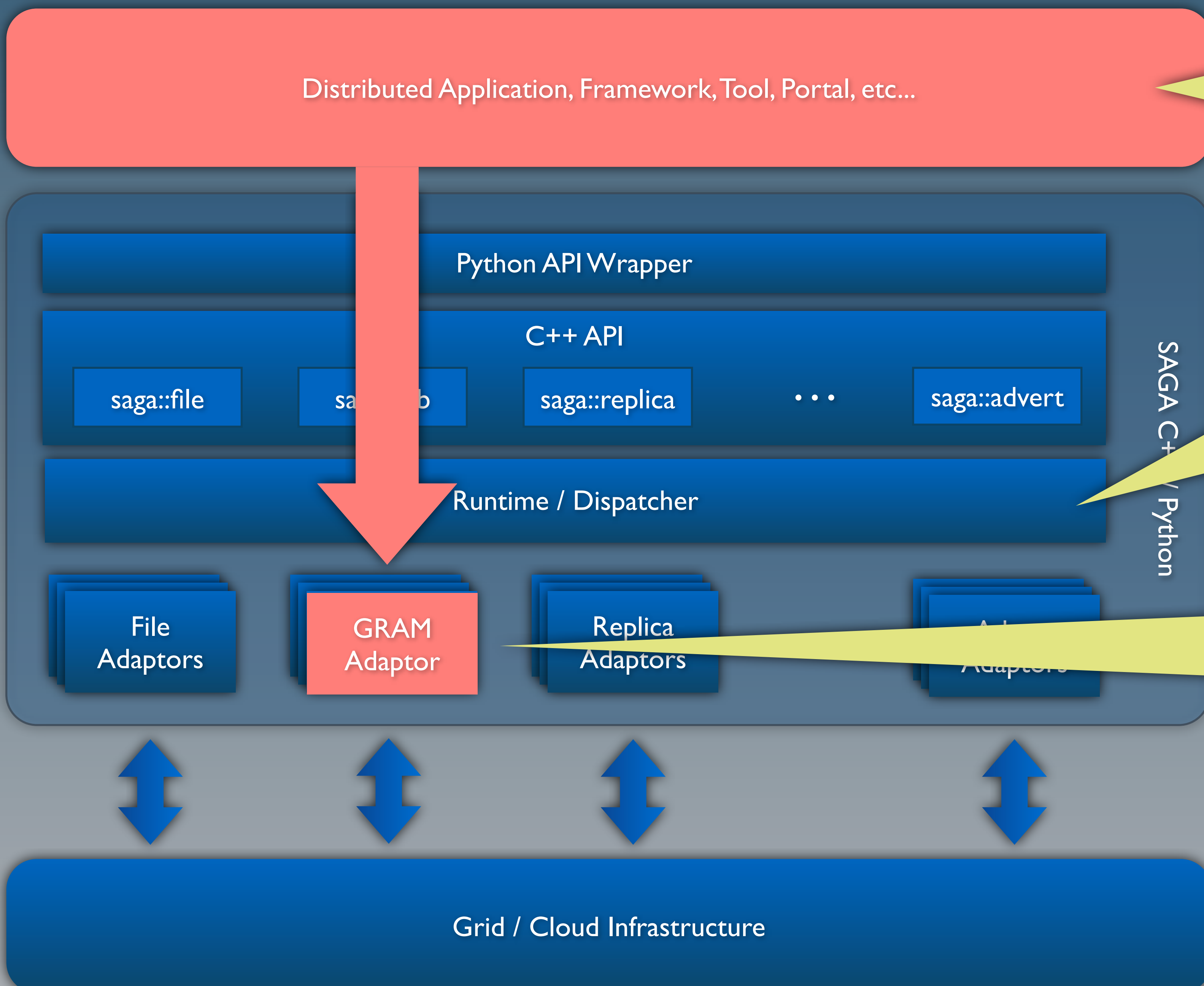
edinburgh data-intensive research



How Does it Work ?



How Does it Work ?




```
js = saga::job::service
    ("gram://login12.hector.ac.uk")

jd = saga::job::description(...)
j = js.create_job(jd)
j.run()
```

(1) See if any of the available adaptors support the Globus GRAM protocol. (Throw error if not)

(2) Bind Adaptor to API call.

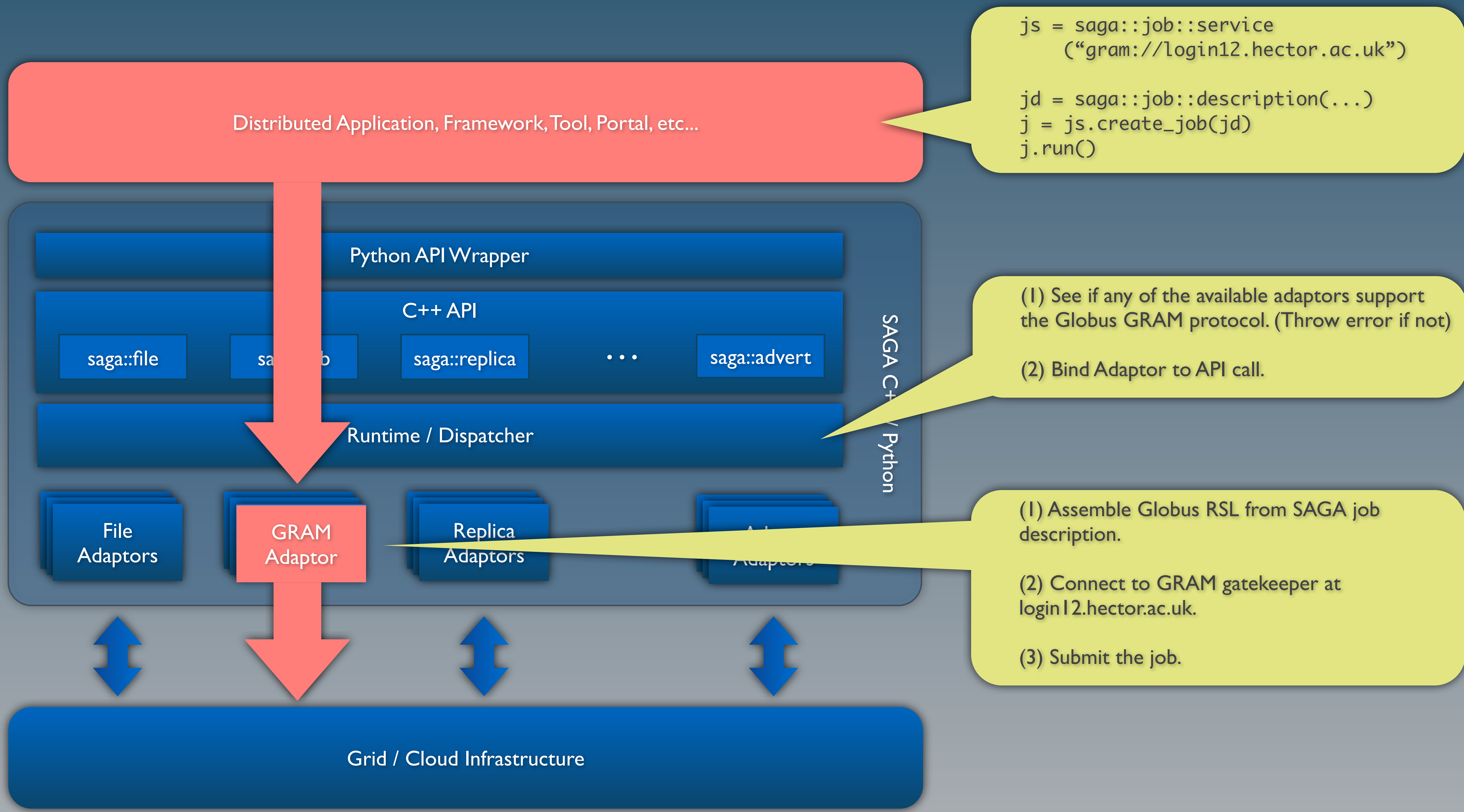
(1) Assemble Globus RSL from SAGA job description.

(2) Connect to GRAM gatekeeper at login12.hector.ac.uk.

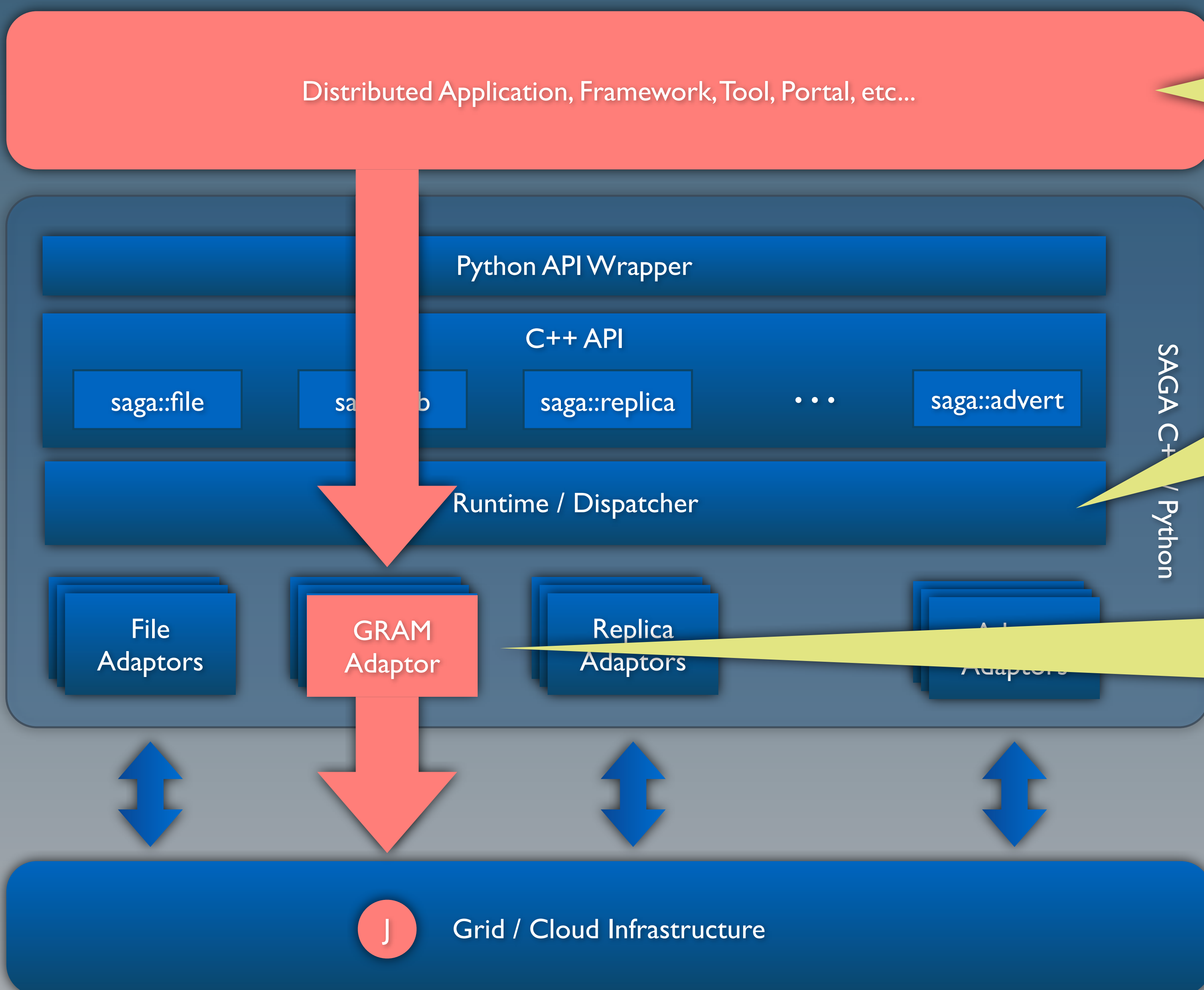
(3) Submit the job.

How Does it Work ?

edinburgh data-intensive research



How Does it Work ?




```
js = saga::job::service
    ("gram://login12.hector.ac.uk")

jd = saga::job::description(...)
j = js.create_job(jd)
j.run()
```

(1) See if any of the available adaptors support the Globus GRAM protocol. (Throw error if not)

(2) Bind Adaptor to API call.

(1) Assemble Globus RSL from SAGA job description.

(2) Connect to GRAM gatekeeper at login12.hector.ac.uk.

(3) Submit the job.

API Packages

- saga::advert - *Advert Service* Access API
- saga::filesystem - Filesystem Access API
- saga::job - Job Submission & Management API
- saga::replica - Replica Catalog Management API
- saga::rpc - Remote Procedure Call API
- saga::sd - Service Discovery API
- saga::stream - Data Stream Client & Server
- API Extensions under development: saga::messaging

Supported Middleware

edinburgh data-intensive research

- saga::job - Arc, Amazon EC2, Condor, Eucalyptus, Globus GRAM (2&5), Fork, gLite, SSH, Nimubs, OGSA BES, PBS (Pro), Platform LSF, SSH, TORQUE, ...
- saga::filesystem - Globus GridFTP, Hadoop HDFS, Local Filesystem, SSHFS, ...
- saga::replica - Globus RLS, SQL Replica Service
- saga::advert - SQL Advert Service
- saga::stream - TCP-based
- saga::sd - gLite SD

A Small Dilemma ?

- After several years of prototyping, testing and hardening, we presented SAGA as the holy grail of distributed computing to the user communities
- Lots of advertising, demos, workshops, tutorials
- But: Uptake very slow and not as expected
- Almost by accident, the problem got solved with the "SAGA *Big-Job*" framework

A Small Dilemma ?


```
/*
#------------------------------------------------------------------------------------------#
#                          gridftpClientToServer.C                                           #
#------------------------------------------------------------------------------------------#
#  Description : Transfer the data file from client site to each remote site.             #
#                and return the status of the file transfer.                              #
#                                                                                          #
#  Input       : Source URL string of the source server , and the Destination URL        #
#                string of the destination server .                                       #
#                The format for URL string is                                             #
#                Source URL      : full path of the file to transferred                   #
#                Destination URL : gsiftp://server-hostname/fullpathname/ofthefile/tobetransfered
#
# Output      : Displays whether the transfer is SUCCESSFUL/FAILED.                      #
#                                                                                          #
#------------------------------------------------------------------------------------------#
*/

#include <fcntl.h>
#include"iomanip"

#include"globus_ftp_client.h"
#include"globus_common.h"

#include<iostream>
#include<string>
#include<fstream>

#include"common.C"
#include"globusCallback.C"
using namespace std;
//FILE *fd;
#define MAX_BUFFER_SIZE 2048
#define ERROR -1
#define SUCCESS 0


class GridFTP:public InputConfig,public GlobusCallback {

        globus_ftp_client_handleattr_t            hattr;
        globus_ftp_client_operationattr_t         oattr;
        globus_ftp_client_handle_t                handle;
        globus_byte_t                             buffer[MAX_BUFFER_SIZE];
        globus_size_t                             buffer_length ;
        globus_result_t                           status;
        char *                                    tmpstr;

public :
        GridFTP();
        ~GridFTP();
        int gridftpClientToServer(string  ,string);

};

/*-----------------------------------------------------------------------------------------------------

Function     : done_cb(...)

Used         : gridftpClientToServer(...),gridftpThirdPartyTransfer(...),gridftpServerToClient(...)

Description  : It is a callback function ,it is called when the transfer is
               completely finished, i.e. both the data channel and control channel
               exchange.
               Here it simply sets a global variable (done) to true so the main
               program will exit the while loop.
----------------------------------------------------------------------------------------------------*/

namespace gridftp_cb {

               static void done_cb ( void *user_arg , globus_ftp_client_handle_t *handle , globus_object_t *err) {

            GridFTP *monitor=(GridFTP*)user_arg;

               char * tmpstr;
               if ( err ){
                              cout<<"\t\t Status : File Transferred Failed "<<endl;
                              cout<<"\t\t ERROR :"<<globus_object_printable_to_string(err)<<endl;
                 }
               else
                              cout<<"\t\t Status : File Transferred Successfully"<<endl;

                              monitor->setDoneValue();
               return;
               };

/*-----------------------------------------------------------------------------------------------------

Function    : data_cb(...)

Used        : gridftpClientToServer(..)

Description : read or write operation in the FTP Client library is asynchronous.A
              callback of this type is passed to such data  operation function calls.
              It is called when the user supplied buffer has been successfully
              transferred to the kernel.
              Note: That does not mean it has been successfully transmitted,instead it
              just reads the next block of data and calls register_write/register_read again.
-----------------------------------------------------------------------------------------------------*/

static void data_cb (void *user_arg , globus_ftp_client_handle_t *handle,globus_object_t * err, globus_byte_t *buffer,
                     globus_size_t length , globus_off_t offset,globus_bool_t  eof)
{
        if ( err ) {
                cout<<"\t\t ERROR :"<<globus_object_printable_to_string(err)<<endl;
        }
        else {
                if ( !eof ) {
                        FILE *fd = (FILE *) user_arg;
                        int rc;
                        rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
                        if ( ferror(fd) != SUCCESS) {
                                 cout<<"\n\t\t Error : Function data_cb\n"<<"\t\t Error code :"<< errno<<endl;
                                return;
                        }
           globus_ftp_client_register_write(handle, buffer,rc,offset + length,feof(fd) != SUCCESS,                                                   data_cb, (void *) fd);
                }
         }
         return;
  };
```

```
/*------------------------------------------------------------------------------------------
 Class       : GridFTP

Function     : GridFTP ( Constructor of class GridFTP )

Description : Used to
              a) Activate the GridFTP client module
              b) Initialize the mutex lock and condition variables
              c) Initialize the GridFTP client handle
------------------------------------------------------------------------------------------*/

GridFTP::GridFTP() {

        buffer_length = MAX_BUFFER_SIZE;
        status = (globus_result_t)globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);

        if ( status != GLOBUS_SUCCESS ) {

                tmpstr = globus_object_printable_to_string(globus_error_get(status));
                cout<<"\n\t Error: Failed to load GLOBUS_FTP_CLIENT_MODULE.\n\t Error Code "<<status<<"\n\t"<<tmpstr<<endl;
                exit(1);
        }

};
/*-----------------------------------------------------------------------------------------
Class        : GridFTP

Function     : ~GridFTP (Destructor of class GridFTP )

Description : Used to
              a) Destroy the GridFTP client handle
              b) Deactivate the GridFTP client module
-----------------------------------------------------------------------------------------*/

GridFTP::~GridFTP() {
        globus_module_deactivate_all();

};
/*------------------------------------------------------------------------------------------
cLass         : GridFTP
Function      : gridftpClientToServer(...)
Description   : To transfer the file from client site to remote site
Input         : a) Source file path which is to transferred
                b) Destination url to store the file
                c) LogFile
------------------------------------------------------------------------------------------*/

int GridFTP::gridftpClientToServer(string  src,string  dst) {

        int rc;
        FILE *fd;

        /* Initialize the handle attribute */
        if (globus_ftp_client_handleattr_init(&hattr) != GLOBUS_SUCCESS) {
                cout<<"\n\t\t ERROR : Failed to activate the ftp client handleattr\n";
                return 1;
        }

        /* Initialize the operation attribute */
        if (globus_ftp_client_operationattr_init(&oattr) != GLOBUS_SUCCESS) {
                cout<<"\n\t\t ERROR : Failed to initialize operationattr\n";
                return 1;
        }

        /* Initalize the handle */
   if (globus_ftp_client_handle_init(&handle,&hattr) != GLOBUS_SUCCESS) {
                cout<<"\n\t\t ERROR : Failed to initialize the handle\n";
                return 1;
        }

        continueOnCond();

          fd = fopen(src.c_str(),"r");
          if ( fd == NULL ) {
              cout<<"Error in opening local file"<<src;
             return 1;
          }

        /* Gridftp API call to start the put operation */
        status = globus_ftp_client_put(&handle,dst.c_str(),GLOBUS_NULL,GLOBUS_NULL,gridftp_cb::done_cb,this);
        if ( status != GLOBUS_SUCCESS ) {

           globus_object_t * err;
           err = globus_error_get(status);
           cout<<endl;
           fprintf(stderr, "\tError : %s", globus_object_printable_to_string(err));
           done = GLOBUS_TRUE;
        }
        else {
           rc = fread(buffer,1,MAX_BUFFER_SIZE,fd);
           globus_ftp_client_register_write(
                           &handle,
                           buffer,
                           rc,
                           0,
                           feof(fd) != SUCCESS,
                           gridftp_cb::data_cb,
                           (void *) fd);
           }

           /* lock on condition */
           waitOnCond();
            fclose(fd);

        globus_ftp_client_handle_destroy(&handle);
        return 0;
};


/* start of main */
int main(int argc , char **argv) {

              int                     hostCount;
        string             destinationUrl,sourceUrl,tempUrl;
        string             sourceFile,destinationFile,outputFile,tmpFile;

        /* Creating the object */
        GridFTP        gridftp;
```

A Small Dilemma ?

```
  /*
#-------------------------------------------------------------------------------------#
#                          gridftpClientToServer.C                                      #
#-------------------------------------------------------------------------------------#
#  Description : Transfer the data file from client site to each remote site.        #
#                and return the status of the file transfer.                          #
#                                                                                     #
#  Input       : Source URL string of the source server , and the Destination URL    #
#                string of the destination server .                                   #
#                The format for URL string is                                         #
#                Source URL      : full path of the file to transferred               #
#                Destination URL : gsiftp://server-hostname/fullpathname/ofthefile/tobetransfered
#
# Output      : Displays whether the transfer is SUCCESSFUL/FAILED.                   #
#                                                                                     #
#-------------------------------------------------------------------------------------#
*/

#include <fcntl.h>
#include"iomanip"

#include"globus_ftp_client.h"
#include"globus_common.h"

#include<iostream>
#include<string>
#include<fstream>

#include"common.C"
#include"globusCallback.C"
using namespace std;
//FILE *fd;
#define MAX_BUFFER_SIZE 2048
#define ERROR -1
#define SUCCESS 0


class GridFTP:public InputConfig,public GlobusCallback {

        globus_ftp_client_handleattr_t          hattr;
        globus_ftp_client_operationattr_t       oattr;
        globus_ftp_client_handle_t              handle;
        globus_byte_t                           buffer[
        globus_size_t                           buffer_
        globus_result_t                         status;
        char *                                  tmpstr;

public :
        GridFTP();
        ~GridFTP();
        int gridftpClientToServer(string  ,string);

};

/*--------------------------------------------------------

Function     : done_cb(...)

Used         : gridftpClientToServer(...),gridftpThirdPar

Description  : It is a callback function ,it is called wh
               completely finished, i.e. both the data ch
               exchange.
               Here it simply sets a global variable (dor
               program will exit the while loop.
--------------------------------------------------------

namespace gridftp_cb {

               static void done_cb ( void *user_arg , glo

            GridFTP *monitor=(GridFTP*)user_arg;

               char * tmpstr;
               if ( err ){
                              cout<<"\t\t Status : File T
                              cout<<"\t\t ERROR :"<<globu
                 }
               else
                              cout<<"\t\t Status : File T

                              monitor->setDoneValue();
               return;
               };

/*-----------------------------------------------------------------------------------------------------

Function    : data_cb(...)

Used        : gridftpClientToServer(..)

Description : read or write operation in the FTP Client library is asynchronous.A
              callback of this type is passed to such data  operation function calls.
              It is called when the user supplied buffer has been successfully
              transferred to the kernel.
              Note: That does not mean it has been successfully transmitted,instead it
              just reads the next block of data and calls register_write/register_read again.
------------------------------------------------------------------------------------------------------*/

static void data_cb (void *user_arg , globus_ftp_client_handle_t *handle,globus_object_t * err, globus_byte_t *buffer,
                     globus_size_t length , globus_off_t offset,globus_bool_t  eof)
{
        if ( err ) {
                cout<<"\t\t ERROR :"<<globus_object_printable_to_string(err)<<endl;
        }
        else {
                if ( !eof ) {
                        FILE *fd = (FILE *) user_arg;
                        int rc;
                        rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
                        if ( ferror(fd) != SUCCESS) {
                                cout<<"\n\t\t Error : Function data_cb\n"<<"\t\t Error code :"<< errno<<endl;
                               return;
                        }
          globus_ftp_client_register_write(handle, buffer,rc,offset + length,feof(fd) != SUCCESS,                                    data_cb, (void *) fd);
                }
         }
         return;
  };
```

```
    /*----------------------------------------------------------------------------------------------
 Class       : GridFTP

Function     : GridFTP ( Constructor of class GridFTP )

Description : Used to
              a) Activate the GridFTP client module
              b) Initialize the mutex lock and condition variables
              c) Initialize the GridFTP client handle
-----------------------------------------------------------------------------------------------*/

GridFTP::GridFTP() {

        buffer_length = MAX_BUFFER_SIZE;
        status = (globus_result_t)globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);

        if ( status != GLOBUS_SUCCESS ) {

                tmpstr = globus_object_printable_to_string(globus_error_get(status));
                cout<<"\n\t Error: Failed to load GLOBUS_FTP_CLIENT_MODULE.\n\t Error Code "<<status<<"\n\t"<<tmpstr<<endl;
                exit(1);
        }

};
/*-------------------------------------------------------------------------------------------
Class        : GridFTP

Function     : ~GridFTP (Destructor of class GridFTP )

Description : Used to
              a) Destroy the GridFTP client handle
              b) Deactivate the GridFTP client module
```

```
                                    /*
#-------------------------------------------------------------------------------------#
#                          gridftpClientToServer.C                                    #
#-------------------------------------------------------------------------------------#
#  Description : Transfer the data file from client site to each remote site          #
#                and return the status of the file transfer.                          #
#                                                                                     #
#  Input       : Source URL string of the source server , and the Destination URL    #
#                string of the destination server .                                   #
#                The format for URL string is                                         #
#                Source URL      : full path of the file to transferred               #
#                Destination URL : gsiftp://server-hostname/fullpathname/ofthefile/tobetrad   #
#                                                                                     #
# Output      : Displays whether the transfer is SUCCESSFUL/FAILED.                   #
#                                                                                     #
#-------------------------------------------------------------------------------------#
                                    */
```

```
           globus_object_t * err;
           err = globus_error_get(status);
           cout<<endl;
           fprintf(stderr, "\tError : %s", globus_object_printable_to_string(err));
           done = GLOBUS_TRUE;
        }
        else {
           rc = fread(buffer,1,MAX_BUFFER_SIZE,fd);
           globus_ftp_client_register_write(
                          &handle,
                          buffer,
                          rc,
                          0,
                          feof(fd) != SUCCESS,
                          gridftp_cb::data_cb,
                          (void *) fd);
           }

           /* lock on condition */
           waitOnCond();
            fclose(fd);

        globus_ftp_client_handle_destroy(&handle);
        return 0;
};


/* start of main */
int main(int argc , char **argv) {

               int                     hostCount;
        string             destinationUrl,sourceUrl,tempUrl;
        string             sourceFile,destinationFile,outputFile,tmpFile;

        /* Creating the object */
        GridFTP          gridftp;
```

A Small Dilemma ?

edinburgh data-intensive research

```
/*
#-----------------------------------------------------------------------------------------#
#                           gridftpClientToServer.C                                         #
#-----------------------------------------------------------------------------------------#
#  Description : Transfer the data file from client site to each remote site.              #
#                and return the status of the file transfer.                               #
#                                                                                          #
#  Input       : Source URL string of the source server , and the Destination URL          #
#                string of the destination server .                                        #
#                The format for URL string is                                              #
#                Source URL      : full path of the file to transferred                    #
#                Destination URL : gsiftp://server-hostname/fullpathname/ofthefile/tobetransfered
#
# Output      : Displays whether the transfer is SUCCESSFUL/FAILED.                        #
#                                                                                          #
#-----------------------------------------------------------------------------------------#
*/

#include <fcntl.h>
#include"iomanip"

#include"globus_ftp_client.h"
#include"globus_common.h"

#include<iostream>
#include<string>
#include<fstream>

#include"common.C"
#include"globusCallback.C"
using namespace std;
//FILE *fd;
#define MAX_BUFFER_SIZE 2048
#define ERROR -1
#define SUCCESS 0


class GridFTP:public InputConfig,public GlobusCallback {

        globus_ftp_client_handleattr_t          hattr;
        globus_ftp_client_operationattr_t       oattr;
        globus_ftp_client_handle_t              handle;
        globus_byte_t                           buffer[
        globus_size_t                           buffer_
        globus_result_t                         status;
        char *                                  tmpstr;

public :
        GridFTP();
        ~GridFTP();
        int gridftpClientToServer(string  ,string);

};

/*----------------------------------------------------------

Function     : done_cb(...)

Used         : gridftpClientToServer(...),gridftpThirdPar

Description  : It is a callback function ,it is called wh
               completely finished, i.e. both the data ch
               exchange.
               Here it simply sets a global variable (don
               program will exit the while loop.
----------------------------------------------------------

namespace gridftp_cb {

               static void done_cb ( void *user_arg , glo

            GridFTP *monitor=(GridFTP*)user_arg;

               char * tmpstr;
               if ( err ){
                          cout<<"\t\t Status : File T
                          cout<<"\t\t ERROR :"<<globu
                 }
               else
                          cout<<"\t\t Status : File T

                          monitor->setDoneValue();
               return;
               };

/*-----------------------------------------------------------------------------------------------

Function    : data_cb(...)

Used        : gridftpClientToServer(..)

Description : read or write operation in the FTP Client library is asynchronous.A
              callback of this type is passed to such data  operation function calls.
              It is called when the user supplied buffer has been successfully
              transferred to the kernel.
              Note: That does not mean it has been successfully transmitted,instead it
              just reads the next block of data and calls register_write/register_read again.
-----------------------------------------------------------------------------------------------*/

static void data_cb (void *user_arg , globus_ftp_client_handle_t *handle,globus_object_t * err, globus_byte_t *buffer,
                     globus_size_t length , globus_off_t offset,globus_bool_t  eof)
{
        if ( err ) {
                cout<<"\t\t ERROR :"<<globus_object_printable_to_string(err)<<endl;
        }
        else {
                if ( !eof ) {
                        FILE *fd = (FILE *) user_arg;
                        int rc;
                        rc = fread(buffer, 1, MAX_BUFFER_SIZE, fd);
                        if ( ferror(fd) != SUCCESS) {
                                cout<<"\n\t\t Error : Function data_cb\n"<<"\t\t Error code :"<< errno<<endl;
                               return;
                        }
           globus_ftp_client_register_write(handle, buffer,rc,offset + length,feof(fd) != SUCCESS,          data_cb, (void *) fd);
                }
         }
         return;
  };
```

```
/*--------------------------------------------------------------------------------------
 Class       : GridFTP

Function     : GridFTP ( Constructor of class GridFTP )

Description : Used to
              a) Activate the GridFTP client module
              b) Initialize the mutex lock and condition variables
              c) Initialize the GridFTP client handle
-----------------------------------------------------------------------------------------*/

GridFTP::GridFTP() {

        buffer_length = MAX_BUFFER_SIZE;
        status = (globus_result_t)globus_module_activate(GLOBUS_FTP_CLIENT_MODULE);

        if ( status != GLOBUS_SUCCESS ) {

                tmpstr = globus_object_printable_to_string(globus_error_get(status));
                cout<<"\n\t Error: Failed to load GLOBUS_FTP_CLIENT_MODULE.\n\t Error Code "<<status<<"\n\t"<<tmpstr<<endl;
                exit(1);
        }

};
/*--------------------------------------------------------------------------------------
Class        : GridFTP

Function     : ~GridFTP (Destructor of class GridFTP )

Description : Used to
              a) Destroy the GridFTP client handle
              b) Deactivate the GridFTP client module

...

           globus_object_t * err;
           err = globus_error_get(status);
           cout<<endl;
           fprintf(stderr, "\tError : %s", globus_object_printable_to_string(err));
           done = GLOBUS_TRUE;
        }
        else {
           rc = fread(buffer,1,MAX_BUFFER_SIZE,fd);
           globus_ftp_client_register_write(
                          &handle,
                          buffer,
                          rc,
                          0,
                          feof(fd) != SUCCESS,
                          gridftp_cb::data_cb,
                          (void *) fd);
           }

           /* lock on condition */
           waitOnCond();
            fclose(fd);

        globus_ftp_client_handle_destroy(&handle);
        return 0;
};


/* start of main */
int main(int argc , char **argv) {

              int                     hostCount;
        string              destinationUrl,sourceUrl,tempUrl;
        string              sourceFile,destinationFile,outputFile,tmpFile;

        /* Creating the object */
        GridFTP         gridftp;
```

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

int main (int argc, char** argv)
{
  try {
    saga::filesystem::file f( std::string(argv[1]) );
    f.copy( std::string(argv[2]) );
  }
  catch(saga::exception const & e)
  {
    std::cerr << “Error: “ << e.what() << std::endl;
  }
}
```

A Small Dilemma ?

- After several years of prototyping, testing and hardening, we presented SAGA as the holy grail of distributed computing to the user communities
- Lots of advertising, demos, workshops, tutorials
- But: Uptake very slow and not as expected
- Almost by accident, the problem got solved with the "SAGA *Big-Job*" framework

SAGA *Big-Job*

- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling

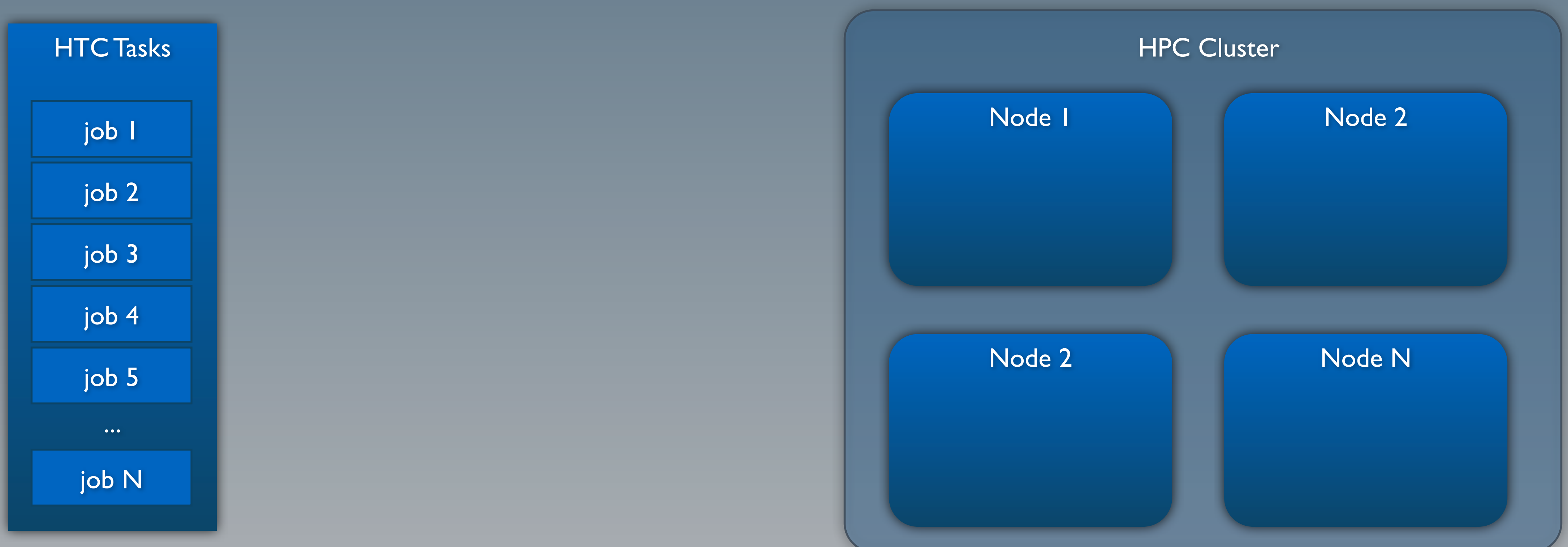
SAGA *Big-Job*

- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling

ole.weidner@ed.ac.uk

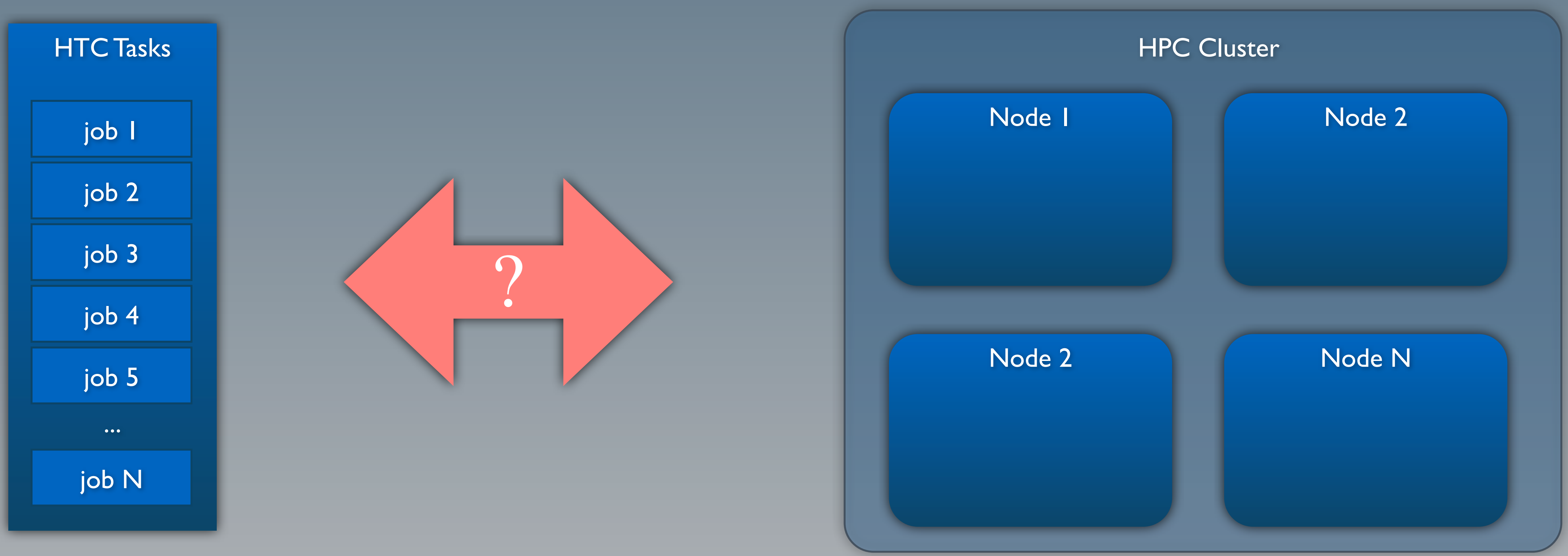
SAGA *Big-Job*

- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



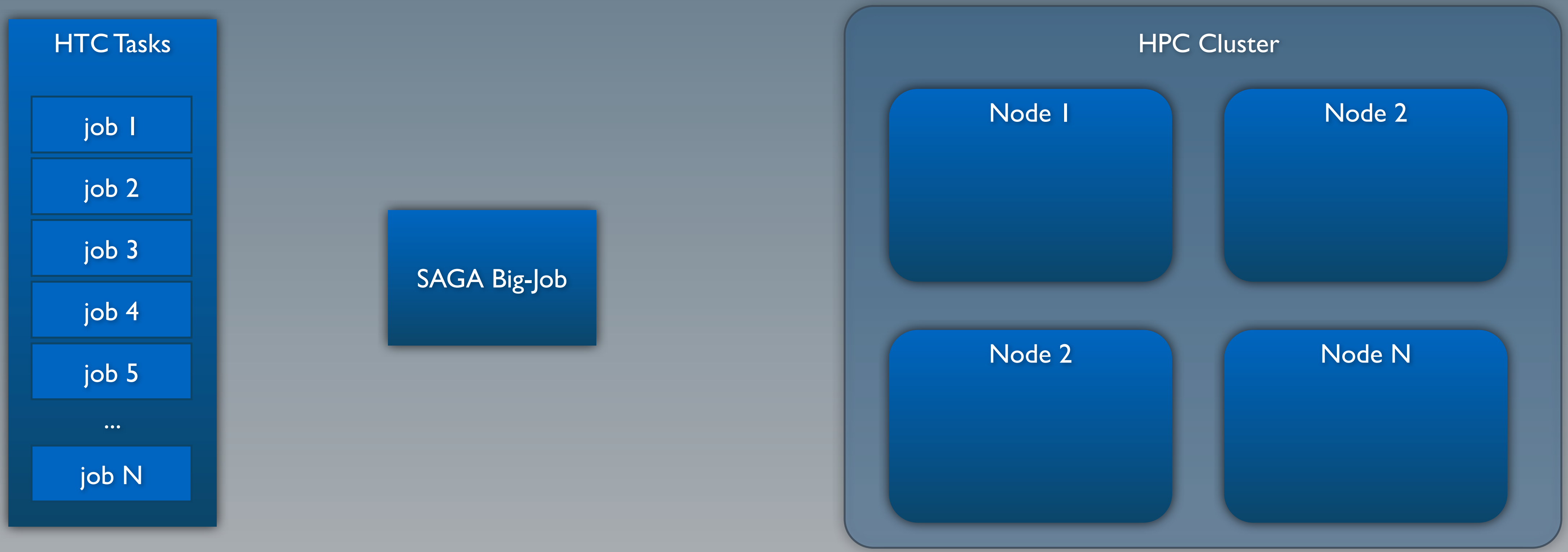
SAGA *Big-Job*

- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



SAGA *Big-Job*

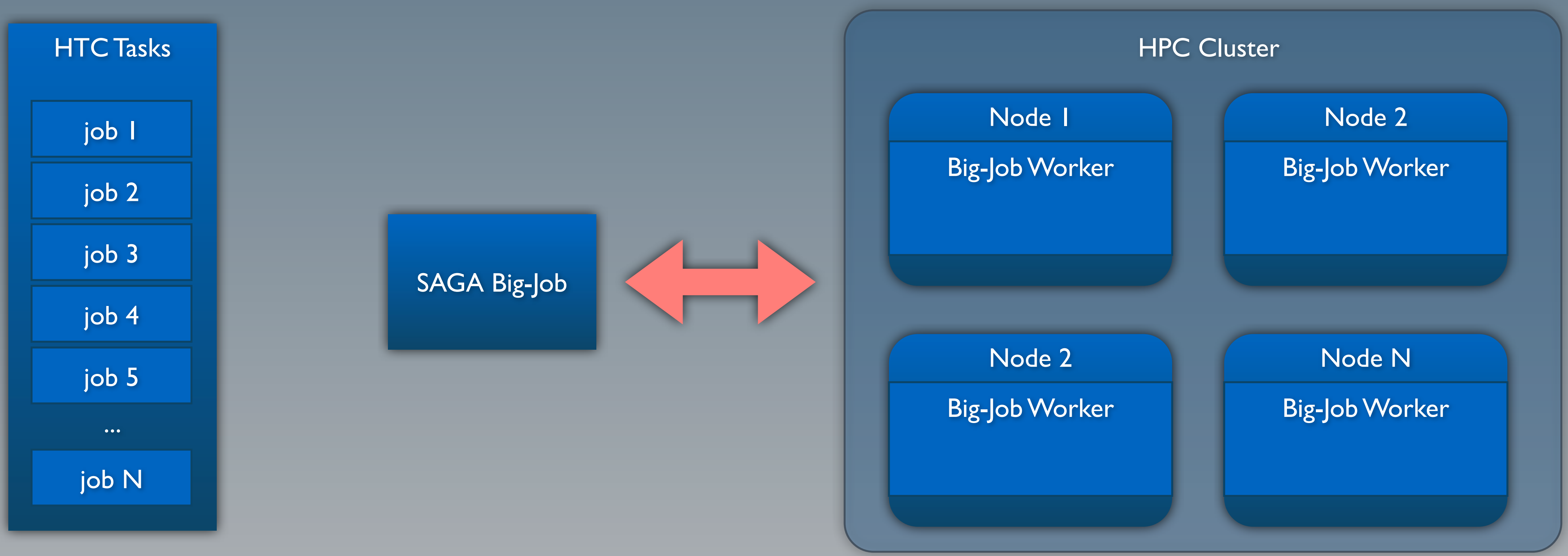
- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



ole.weidner@ed.ac.uk

SAGA *Big-Job*

- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling

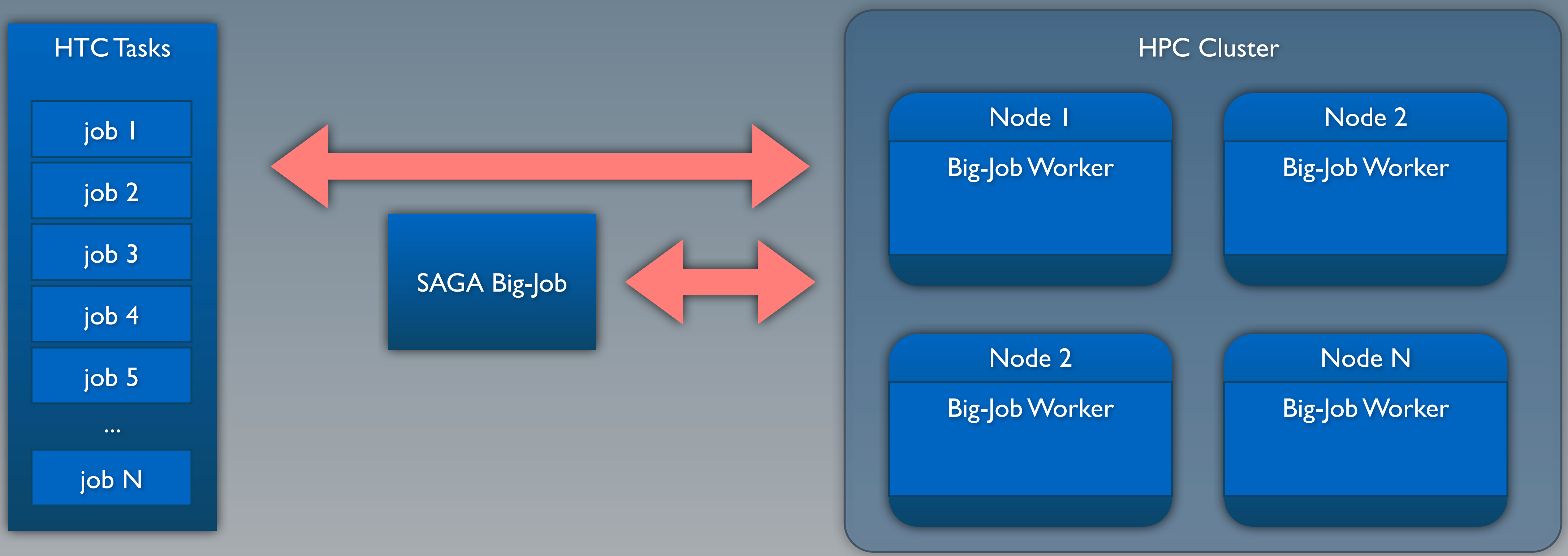


ole.weidner@ed.ac.uk

SAGA *Big-Job*

edinburgh data-intensive research

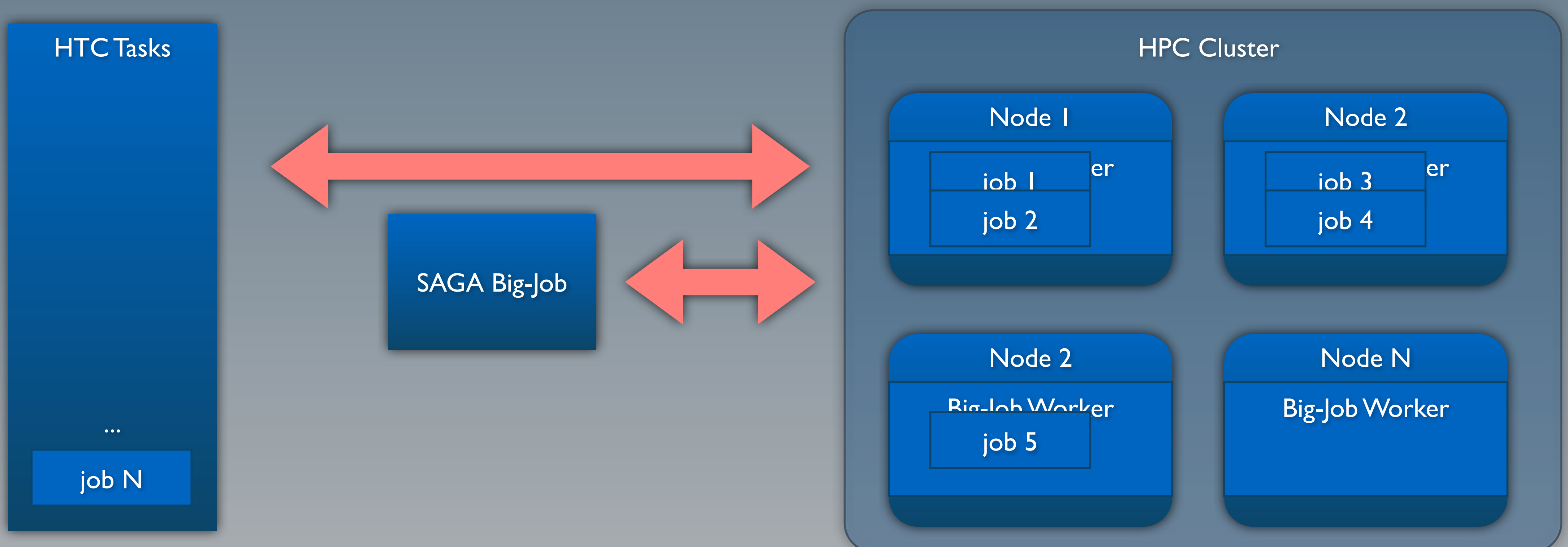
- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



ole.weidner@ed.ac.uk

SAGA *Big-Job*

- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling



SAGA *Big-Job*

edinburgh data-intensive research

- *Big-Job* (a.k.a. *glide-in*, or *pilot-job*) is a simple distributed programming abstraction written in SAGA (Python)
- It allows to run lots of HTC jobs (often single-core) transparently on HPC machines using overlay scheduling

- *Big-Job* suddenly sparked a lot of interest in SAGA
  - Makes a huge problem simply disappear
  - Allows to use legacy code (non-intrusive)
  - Runs everywhere - even on Hector (CRAY) and EC2

Abstractions, Abstractions, Abstractions !

- Apparently, we didn't understand the user communities and their requirements properly
- Users (especially the non-technical users) don't want another API. They want simple solutions for their every day problems. Abstractions can help !
- It turns out that SAGA is perfect to develop distributed programming abstractions:
  - Hides specific middleware implementation details
  - Allows concurrent cross-infrastructure resource usage
  - Optimisation and "adaptation" can happen "behind the scenes"

Abstractions, Abstractions, Abstractions !

- Abstractions: from generic to specialised:
    - Pilot-Job (a.k.a. Big-job, a.k.a. Glide-in)
    - Pilot-Data (data affinity)
    - Master-Worker
    - Workflow (a.k.a. DAG)
    - Peer-to-Peer
    - Replica exchange
    - Map-Reduce
    - All-Pairs
- See eSI 3DPAS theme: http://www.esi.ac.uk/research-themes/5

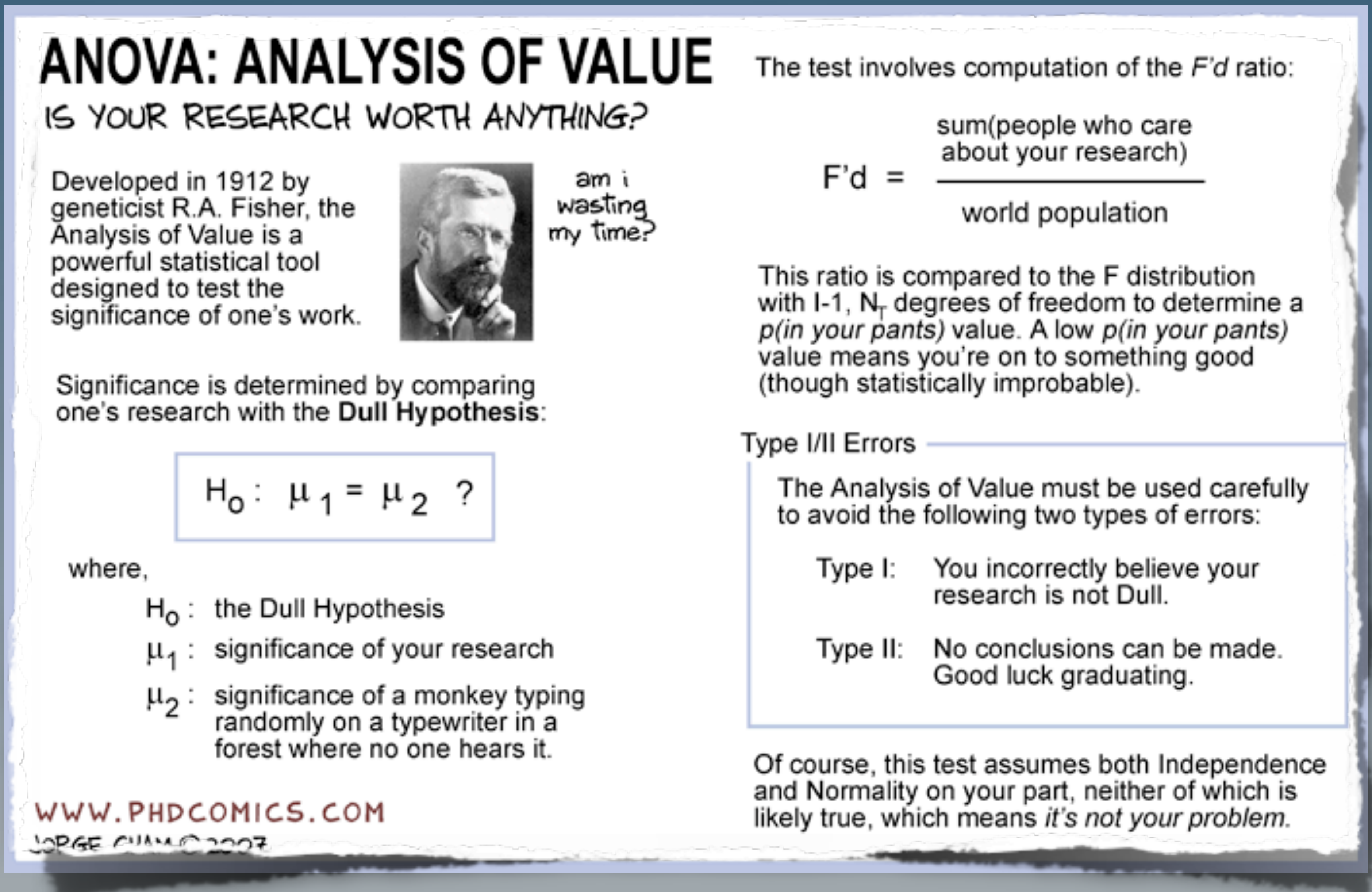
Abstractions, Abstractions, Abstractions !

edinburgh data-intensive research

- Abstractions: from generic to specialised:
  - Pilot-Job (a.k.a. Big-job, a.k.a. Glide-in)
  - Pilot-Data (data affinity)
  - Master-Worker
  - Workflow (a.k.a. DAG)
  - Peer-to-Peer
  - Replica exchange
  - Map-Reduce
  - All-Pairs
- See eSI 3DPAS theme: http://www.esi.ac.uk/research-themes/5

Is There Potential for (PhD) Research ?

edinburgh data-intensive research



Is There Potential for (PhD) Research ?

- Previous work and experience has shown the relevance and importance of abstractions in distributed computing
  - Especially interesting: with abstractions, optimisation can be done "behind the scenes" (hidden from the user)
- Some fundamental questions:
  - What are interesting upcoming challenges in future distributed applications and systems ?
  - Can some of the challenges possibly be address by using distributed programming abstractions ?
  - What is the current state of research in distributed programming abstractions ?

What are the Upcoming Challenges ?

edinburgh
data-intensive
research

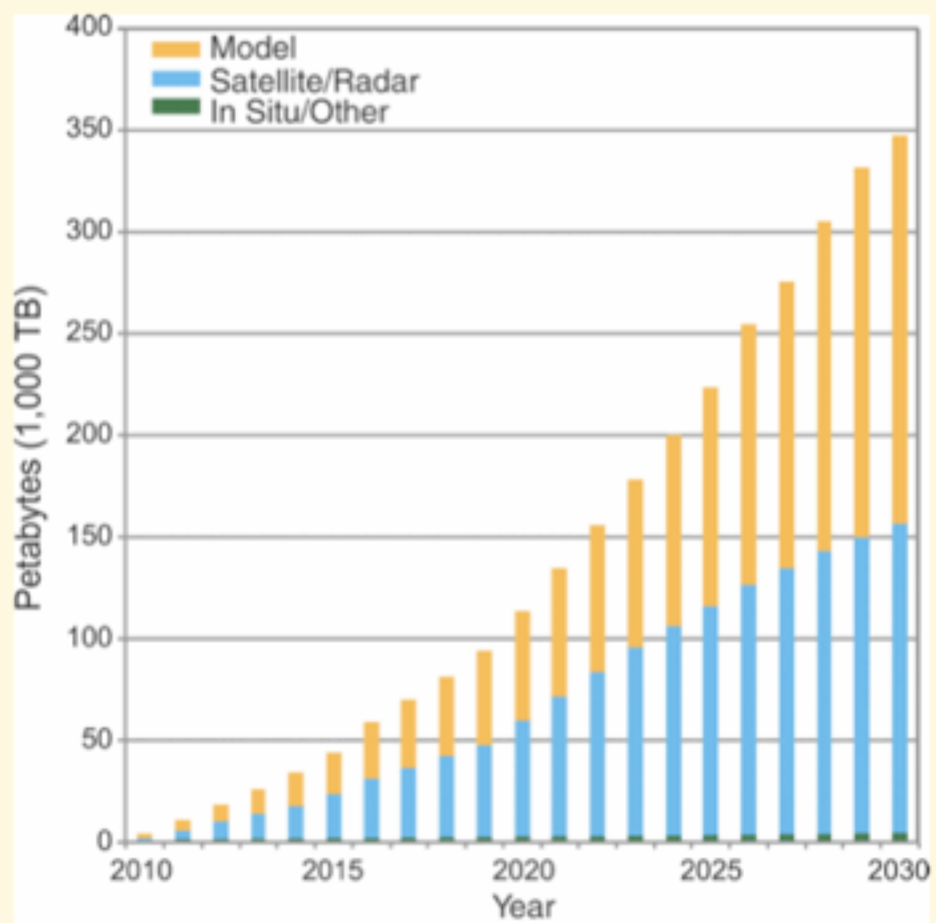
- The *Data Deluge*
  "The amount of genomic data available for study is increasing at a rate similar to that of Moore's Law" http://www.mcs.anl.gov/uploads/cels/papers/P1238.pdf
  - *"Store now - process later"* might not always be an option
  - *Adaptive* algorithms will be required to handle vast amounts of dynamic (streaming) data
  - Methods to characterise/predict data and especially dynamic changes in data will play a key role
- Infrastructure Challenge
  - Applications will have to spread and run across different heterogeneous infrastructures (possibly even simultaneously)
  - Probably other objectives besides *"minimise makespan"*

What are the Upcoming Challenges ?

edinburgh
data-intensive
research

... available for st...

...'s Law" http://www.m...



Projected increase in global climate data holdings for climate models, remotely sensed data, and in situ.

heterogeneou... ...n simultaneously)

- Probably othe... ...*makespan"*

What are the Upcoming Challenges ?

edinburgh data-intensive research

- The *Data Deluge*

  "The amount of genomic data available for study is increasing at a rate similar to that of Moore's Law" http://www.mcs.anl.gov/uploads/cels/papers/P1238.pdf

  - *"Store now - process later"* might not always be an option
  - *Adaptive* algorithms will be required to handle vast amounts of dynamic (streaming) data
  - Methods to characterise/predict data and especially dynamic changes in data will play a key role

- Infrastructure Challenge
  - Applications will have to spread and run across different heterogeneous infrastructures (possibly even simultaneously)
  - Probably other objectives besides *"minimise makespan"*

Can Distributed Abstractions Help ?

- There might be lots of new challenges, but from an application perspective things will mostly stay the same (!)
  - Applications will still be using common patterns and abstractions
  - But: input and objectives might change and they might change *DYNAMICALLY*
- Once we have understood the details and dynamics of these new challenges, we can *encapsulate* them inside distributed programming abstractions
- If this hasn't been done, it could be a nice intellectual and practically relevant contribution to the field

A Case for Dynamic Optimisation

edinburgh data-intensive research

- DPAs supporting **dynamic** optimisation to address dynamic data challenges:
  - Adaptive execution strategy based on input, output and system characteristics
  - Ability to "understand" and "predict" data (characteristics)
- DPAs supporting **autonomous** adaption to address rising complexity in infrastructure

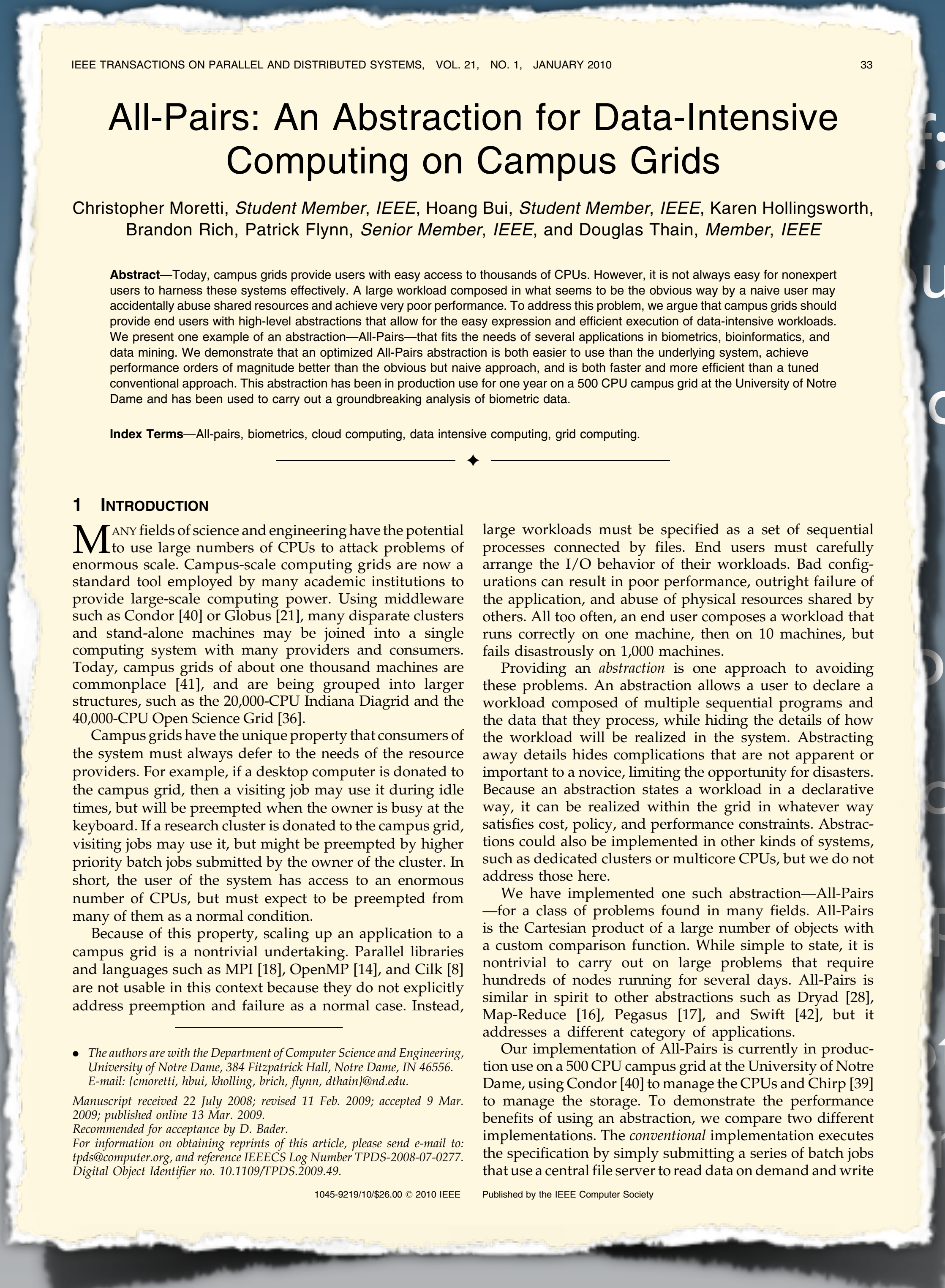
Current State of Affairs

edinburgh data-intensive research

- There doesn't seem to be a lot of work that revisits DPAs in the context of:
  - Streaming input / output data
  - Data with dynamically changing characteristics
  - Autonomic | adaptive | dynamic | optimisation
- Lots of work in static optimisation ("*fine-tuning*")
  - Optimise throughput for workload X on (idealised) platform Y
  - Usually confined to a specific middleware
- There seems to be a lot of uncharted territory, but that remains to be proved (more reading!)

Current State of Affairs

edinburgh
data-intensive
research


IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 21, NO. 1, JANUARY 2010 33

All-Pairs: An Abstraction for Data-Intensive Computing on Campus Grids

Christopher Moretti, *Student Member*, *IEEE*, Hoang Bui, *Student Member*, *IEEE*, Karen Hollingsworth, Brandon Rich, Patrick Flynn, *Senior Member*, *IEEE*, and Douglas Thain, *Member*, *IEEE*

Abstract—Today, campus grids provide users with easy access to thousands of CPUs. However, it is not always easy for nonexpert users to harness these systems effectively. A large workload composed in what seems to be the obvious way by a naive user may accidentally abuse shared resources and achieve very poor performance. To address this problem, we argue that campus grids should provide end users with high-level abstractions that allow for the easy expression and efficient execution of data-intensive workloads. We present one example of an abstraction—All-Pairs—that fits the needs of several applications in biometrics, bioinformatics, and data mining. We demonstrate that an optimized All-Pairs abstraction is both easier to use than the underlying system, achieve performance orders of magnitude better than the obvious but naive approach, and is both faster and more efficient than a tuned conventional approach. This abstraction has been in production use for one year on a 500 CPU campus grid at the University of Notre Dame and has been used to carry out a groundbreaking analysis of biometric data.

Index Terms—All-pairs, biometrics, cloud computing, data intensive computing, grid computing.

1 Introduction

Many fields of science and engineering have the potential to use large numbers of CPUs to attack problems of enormous scale. Campus-scale computing grids are now a standard tool employed by many academic institutions to provide large-scale computing power. Using middleware such as Condor [40] or Globus [21], many disparate clusters and stand-alone machines may be joined into a single computing system with many providers and consumers. Today, campus grids of about one thousand machines are commonplace [41], and are being grouped into larger structures, such as the 20,000-CPU Indiana Diagrid and the 40,000-CPU Open Science Grid [36].

Campus grids have the unique property that consumers of the system must always defer to the needs of the resource providers. For example, if a desktop computer is donated to the campus grid, then a visiting job may use it during idle times, but will be preempted when the owner is busy at the keyboard. If a research cluster is donated to the campus grid, visiting jobs may use it, but might be preempted by higher priority batch jobs submitted by the owner of the cluster. In short, the user of the system has access to an enormous number of CPUs, but must expect to be preempted from many of them as a normal condition.

Because of this property, scaling up an application to a campus grid is a nontrivial undertaking. Parallel libraries and languages such as MPI [18], OpenMP [14], and Cilk [8] are not usable in this context because they do not explicitly address preemption and failure as a normal case. Instead, large workloads must be specified as a set of sequential processes connected by files. End users must carefully arrange the I/O behavior of their workloads. Bad configurations can result in poor performance, outright failure of the application, and abuse of physical resources shared by others. All too often, an end user composes a workload that runs correctly on one machine, then on 10 machines, but fails disastrously on 1,000 machines.

Providing an *abstraction* is one approach to avoiding these problems. An abstraction allows a user to declare a workload composed of multiple sequential programs and the data that they process, while hiding the details of how the workload will be realized in the system. Abstracting away details hides complications that are not apparent or important to a novice, limiting the opportunity for disasters. Because an abstraction states a workload in a declarative way, it can be realized within the grid in whatever way satisfies cost, policy, and performance constraints. Abstractions could also be implemented in other kinds of systems, such as dedicated clusters or multicore CPUs, but we do not address those here.

We have implemented one such abstraction—All-Pairs—for a class of problems found in many fields. All-Pairs is the Cartesian product of a large number of objects with a custom comparison function. While simple to state, it is nontrivial to carry out on large problems that require hundreds of nodes running for several days. All-Pairs is similar in spirit to other abstractions such as Dryad [28], Map-Reduce [16], Pegasus [17], and Swift [42], but it addresses a different category of applications.

Our implementation of All-Pairs is currently in production use on a 500 CPU campus grid at the University of Notre Dame, using Condor [40] to manage the CPUs and Chirp [39] to manage the storage. To demonstrate the performance benefits of using an abstraction, we compare two different implementations. The *conventional* implementation executes the specification by simply submitting a series of batch jobs that use a central file server to read data on demand and write

- *The authors are with the Department of Computer Science and Engineering, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556. E-mail: {cmoretti, hbui, kholling, brich, flynn, dthain}@nd.edu.*

Manuscript received 22 July 2008; revised 11 Feb. 2009; accepted 9 Mar. 2009; published online 13 Mar. 2009.
Recommended for acceptance by D. Bader.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-07-0277.
Digital Object Identifier no. 10.1109/TPDS.2009.49.

 Published by the IEEE Computer Society

be a lot of work that revisits
:
ut data
changing characteristics
dynamic | optimisation
ptimisation (“*fine-tuning*”)
or workload X on (idealised) platform Y
pecific middleware
t of uncharted territory, but that
more reading!)

Current State of Affairs

edinburgh
data-intensive
research

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 21, NO. 1, JANUARY 2010

All-Pairs: An Abstraction for Da
Computing on Campus (

Christopher Moretti, *Student Member, IEEE*, Hoang Bui, *Student Memb*
Brandon Rich, Patrick Flynn, *Senior Member, IEEE*, and Dougla

Abstract—Today, campus grids provide users with easy access to thousands of CPUs. Howev
users to harness these systems effectively. A large workload composed in what seems to be t
accidentally abuse shared resources and achieve very poor performance. To address this proble
provide end users with high-level abstractions that allow for the easy expression and efficient e
We present one example of an abstraction—All-Pairs—that fits the needs of several application
data mining. We demonstrate that an optimized All-Pairs abstraction is both easier to use than
performance orders of magnitude better than the obvious but naive approach, and is both faste
conventional approach. This abstraction has been in production use for one year on a 500 CPU
Dame and has been used to carry out a groundbreaking analysis of biometric data.

Index Terms—All-pairs, biometrics, cloud computing, data intensive computing, grid computin

1 Introduction

Many fields of science and engineering have the potential to use large numbers of CPUs to attack problems of enormous scale. Campus-scale computing grids are now a standard tool employed by many academic institutions to provide large-scale computing power. Using middleware such as Condor [40] or Globus [21], many disparate clusters and stand-alone machines may be joined into a single computing system with many providers and consumers. Today, campus grids of about one thousand machines are commonplace [41], and are being grouped into larger structures, such as the 20,000-CPU Indiana Diagrid and the 40,000-CPU Open Science Grid [36].

Campus grids have the unique property that consumers of the system must always defer to the needs of the resource providers. For example, if a desktop computer is donated to the campus grid, then a visiting job may use it during idle times, but will be preempted when the owner is busy at the keyboard. If a research cluster is donated to the campus grid, visiting jobs may use it, but might be preempted by higher priority batch jobs submitted by the owner of the cluster. In short, the user of the system has access to an enormous number of CPUs, but must expect to be preempted from many of them as a normal condition.

Because of this property, scaling up an application to a campus grid is a nontrivial undertaking. Parallel libraries and languages such as MPI [18], OpenMP [14], and Cilk [8] are not usable in this context because they do not explicitly address preemption and failure as a normal case. Instead,

large workloads mus
processes connected
arrange the I/O beha
urations can result in
the application, and a
others. All too often, a
runs correctly on one
fails disastrously on 1
Providing an *abstr*
these problems. An al
workload composed o
the data that they proc
the workload will be
away details hides con
important to a novice, li
Because an abstraction
way, it can be realize
satisfies cost, policy, ar
tions could also be imp
such as dedicated clust
address those here.
We have impleme
—for a class of prob
is the Cartesian prod
a custom comparison
nontrivial to carry
hundreds of nodes
similar in spirit to o
Map-Reduce [16], F
addresses a different
Our implementatio
tion use on a 500 CPU
Dame, using Condor
to manage the stora
benefits of using an a
implementations. The
the specification by s
that use a central file s

• *The authors are with the Department of Computer Science and Engineering, University of Notre Dame, 384 Fitzpatrick Hall, Notre Dame, IN 46556. E-mail: {cmoretti, hbui, kholling, brich, flynn, dthain}@nd.edu.*

Manuscript received 22 July 2008; revised 11 Feb. 2009; accepted 9 Mar. 2009; published online 13 Mar. 2009.
Recommended for acceptance by D. Bader.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2008-07-0277. Digital Object Identifier no. 10.1109/TPDS.2009.49.

1045-9219/10/$26.00 © 2010 IEEE Published by the IEEE Computer

Cluster Comput (2010) 13: 243–256
DOI 10.1007/s10586-010-0134-7

Harnessing parallelism in multicore clusters with the All-Pairs, Wavefront, and Makeflow abstractions

Li Yu · Christopher Moretti · Andrew Thrasher · Scott Emrich · Kenneth Judd · Douglas Thain

Received: 9 November 2009 / Accepted: 16 March 2010 / Published online: 23 April 2010

Abstract Both distributed systems and multicore systems are difficult programming environments. Although the expert programmer may be able to carefully tune these systems to achieve high performance, the non-expert may struggle. We argue that high level abstractions are an effective way of making parallel computing accessible to the non-expert. An abstraction is a regularly structured framework into which a user may plug in simple sequential programs to create very large parallel programs. By virtue of a regular structure and declarative specification, abstractions may be materialized on distributed, multicore, and distributed multicore systems with robust performance across a wide range of problem sizes. In previous work, we presented the All-Pairs abstraction for computing on distributed systems of single CPUs. In this paper, we extend All-Pairs to multicore systems, and introduce the Wavefront and Makeflow abstractions, which represent a number of problems in economics and bioinformatics. We demonstrate good scaling of both abstractions up to 32 cores on one machine and hundreds of cores in a distributed system.

Keywords Abstractions · Multicore · Distributed systems · Bioinformatics · Economics

L. Yu (✉) · C. Moretti · A. Thrasher · S. Emrich · D. Thain
Department of Computer Science and Engineering, University of Notre Dame, South Bend, USA
e-mail: lyu2@nd.edu

K. Judd
Hoover Institution, Stanford University, Stanford, USA

1 Introduction

Distributed systems such as clusters, clouds, and grids are very challenging programming environments. (Hereafter, we refer to all of these systems as *clusters*.) A user that wishes to execute a large workload with some inherent parallelism is confronted with a dizzying array of choices. How should the workload be broken up into jobs? How should the data be distributed to each node? How many nodes should be used? Will the network be a bottleneck? Often, the answers to these questions depend heavily on the properties of the system and workload in use. Changing one parameter, such as the size of a file or the runtime of a job, may require a completely different strategy.

Multicore systems present many of the same challenges. The orders of magnitude change, but the questions are similar. How should work be divided among threads? Should we use message passing or shared memory? How many CPUs should be used? Will memory access present a bottleneck? When we consider clusters of multicore computers, then the problems become more complex.

We argue that *abstractions* are an effective way of enabling non-expert users to harness clusters, multicore computers, and clusters of multicore computers. An abstraction is a declarative structure that joins simple data structures and small sequential programs into parallel graphs that can be scaled to very large sizes. Because an abstraction is specialized to a restricted class of workloads, it is possible to create an efficient, robust, scalable, and fault tolerant implementation. In previous work, we introduced the All-Pairs [12] and Classify [13] abstractions, and described how they can be used to solve data intensive problems in the fields of biometrics, bioinformatics, and data mining. Our implementations allow non-experts to harness hundreds of processors on problems that run for hours or days using the Condor [27] distributed batch system.

Springer

hat revisits

ics

on

uning")

ealised) platform Y

itory, but that

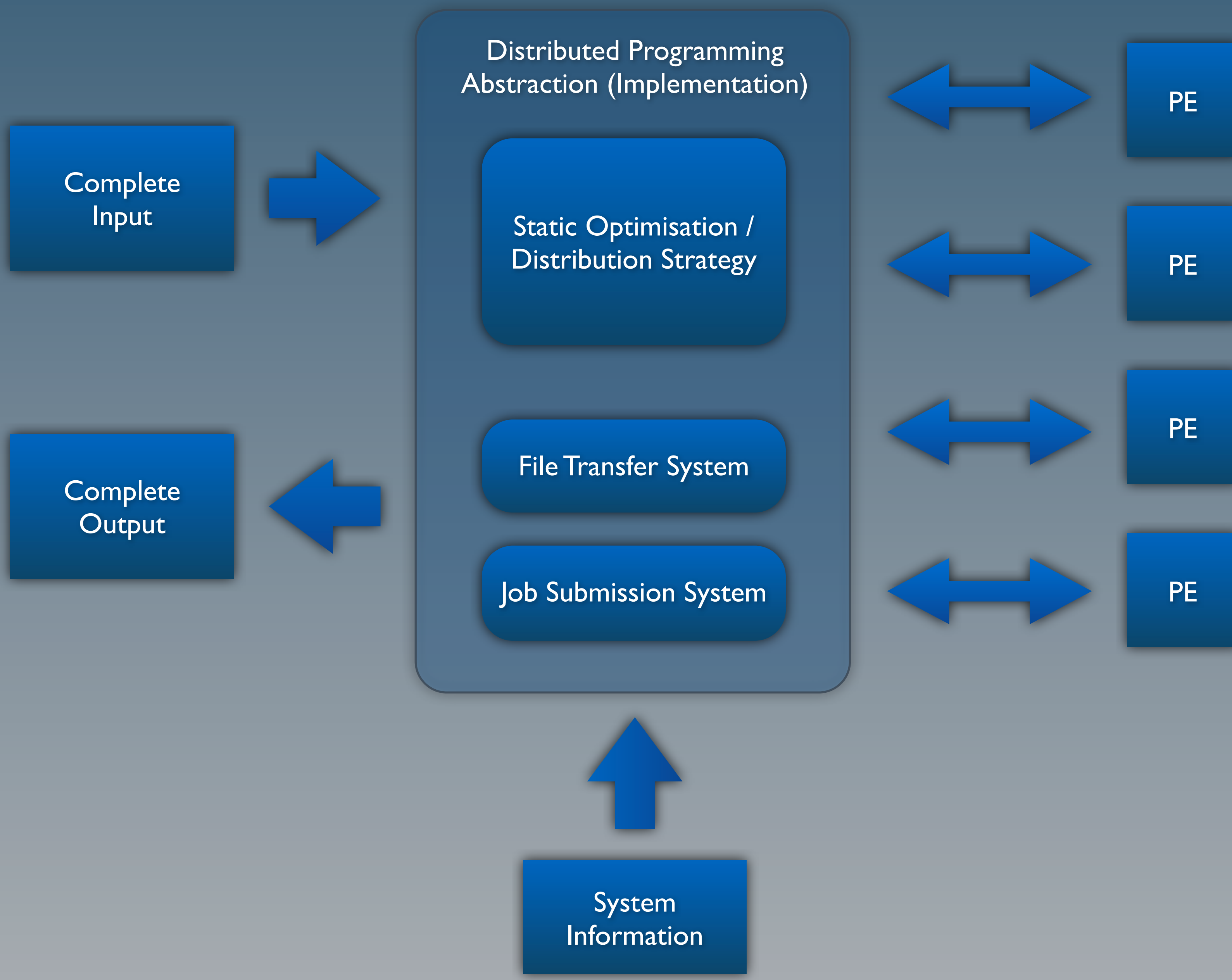
IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 21, NO. 1, JANUARY 2010
All-Pairs: An Abstraction for Da
Computing on Campus
Christopher Moretti, Student Member, IEEE, Hoang Bui, Student Memb
Brandon Rich, Patrick Flynn, Senior Member, IEEE, and Dougla
1 INTRODUCTION
Cluster Comput (2010) 13: 243–256
DOI 10.1007/s10586-010-0134-7
Harnessing parallelism in multicore clusters with t
Wavefront, and Makeflow abstractions
Li Yu · Christopher Moretti · Andrew Thrasher · Scott Emrich · Kenneth Judd · Douglas Thain
Received: 9 November 2009 / Accepted: 16 March 2010 / Published online: 23 April 2010
© Springer Science+Business Media, LLC 2010
Abstract Both distributed systems and multicore systems are difficult programming environments. Although the expert programmer may be able to carefully tune these systems to achieve high performance, the non-expert may struggle. We argue that high level abstractions are an effective way of making parallel computing accessible to the non-expert. An abstraction is a regularly structured framework into which a user may plug in simple sequential programs to create very large parallel programs. By virtue of a regular structure and declarative specification, abstractions may be materialized on distributed, multicore, and distributed multicore systems with robust performance across a wide range of problem sizes. In previous work, we presented the All-Pairs abstraction for computing on distributed systems of single CPUs. In this paper, we extend All-Pairs to multicore systems, and introduce the Wavefront and Makeflow abstractions, which represent a number of problems in economics and bioinformatics. We demonstrate good scaling of both abstractions up to 32 cores on one machine and hundreds of cores in a distributed system.
Keywords Abstractions · Multicore · Distributed systems · Bioinformatics · Economics
1 Introduction
All-Pairs: An Abstraction for Data-Intensive Cloud Computing
Christopher Moretti, Jared Bulosan, Douglas Thain, and Patrick J. Flynn
Department of Computer Science and Engineering, University of Notre Dame
Abstract
Although modern parallel and distributed computing systems provide easy access to large amounts of computing power, it is not always easy for non-expert users to harness these large systems effectively. A large workload composed in what seems to be the obvious way by a naive user may accidentally abuse shared resources and achieve very poor performance. To address this problem, we propose that production systems should provide end users with high-level abstractions that allow for the easy expression and efficient execution of data intensive workloads. We present one example of an abstraction – All-Pairs – that fits the needs of several data-intensive scientific applications. We demonstrate that an optimized All-Pairs abstraction is both easier to use than the underlying system, and achieves performance orders of magnitude better than the obvious but naive approach, and twice as fast as a hand-optimized conventional approach.
1 Introduction
2 The All-Pairs Problem
All-Pairs(set A, set B, function F) returns matrix M:
Compare all elements of set A to all elements of set B via function F, yielding matrix M, such that M[i,j] = F(A[i],B[j]).
978-1-4244-1694-3/08/$25.00 ©2008 IEEE

Current State of Affairs

edinburgh data-intensive research

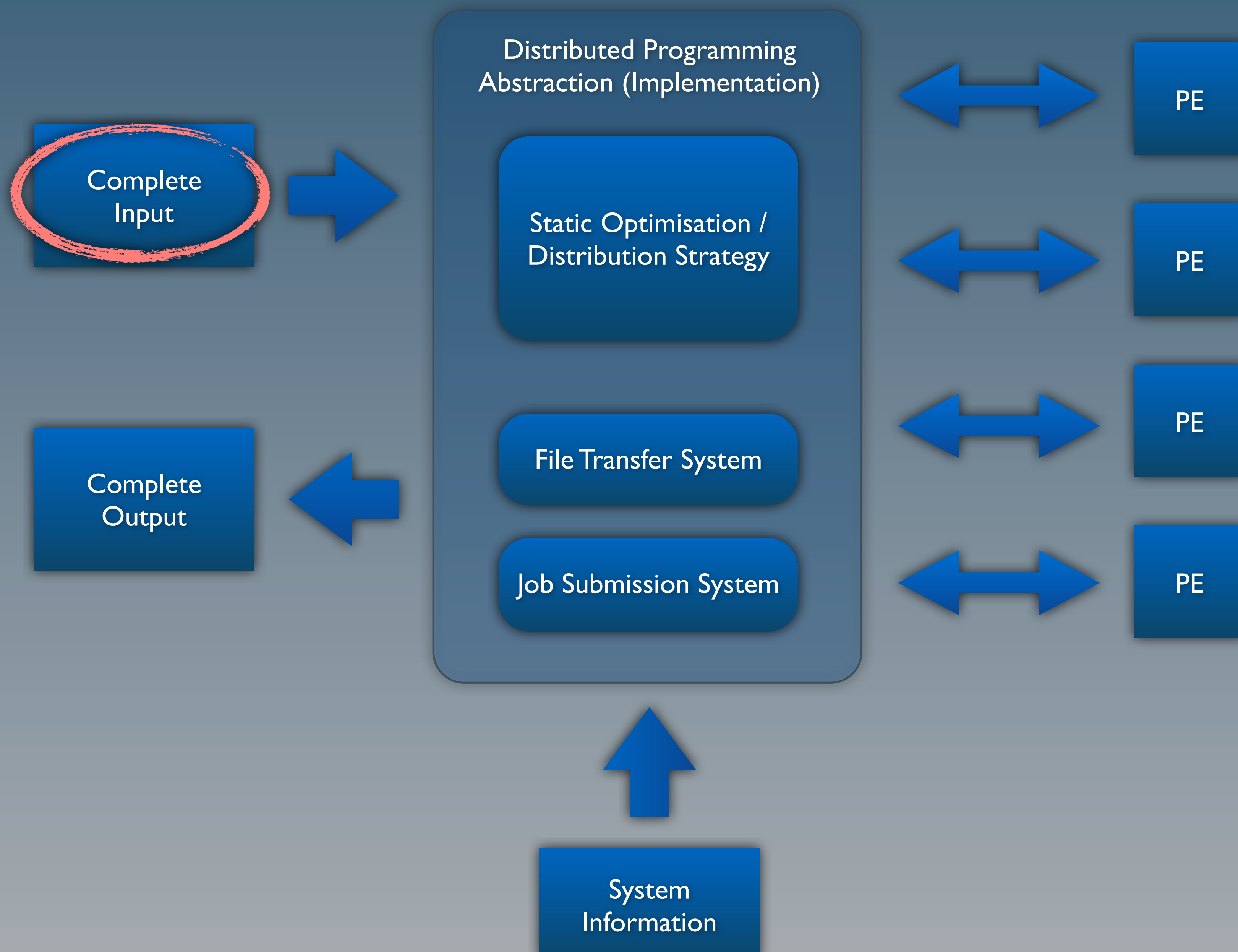
- There doesn't seem to be a lot of work that revisits DPAs in the context of:
  - Streaming input / output data
  - Data with dynamically changing characteristics
  - Autonomic | adaptive | dynamic | optimisation
- Lots of work in static optimisation ("*fine-tuning*")
  - Optimise throughput for workload X on (idealised) platform Y
  - Usually confined to a specific middleware
- There seems to be a lot of uncharted territory, but that remains to be proved (more reading!)

Everything is Static



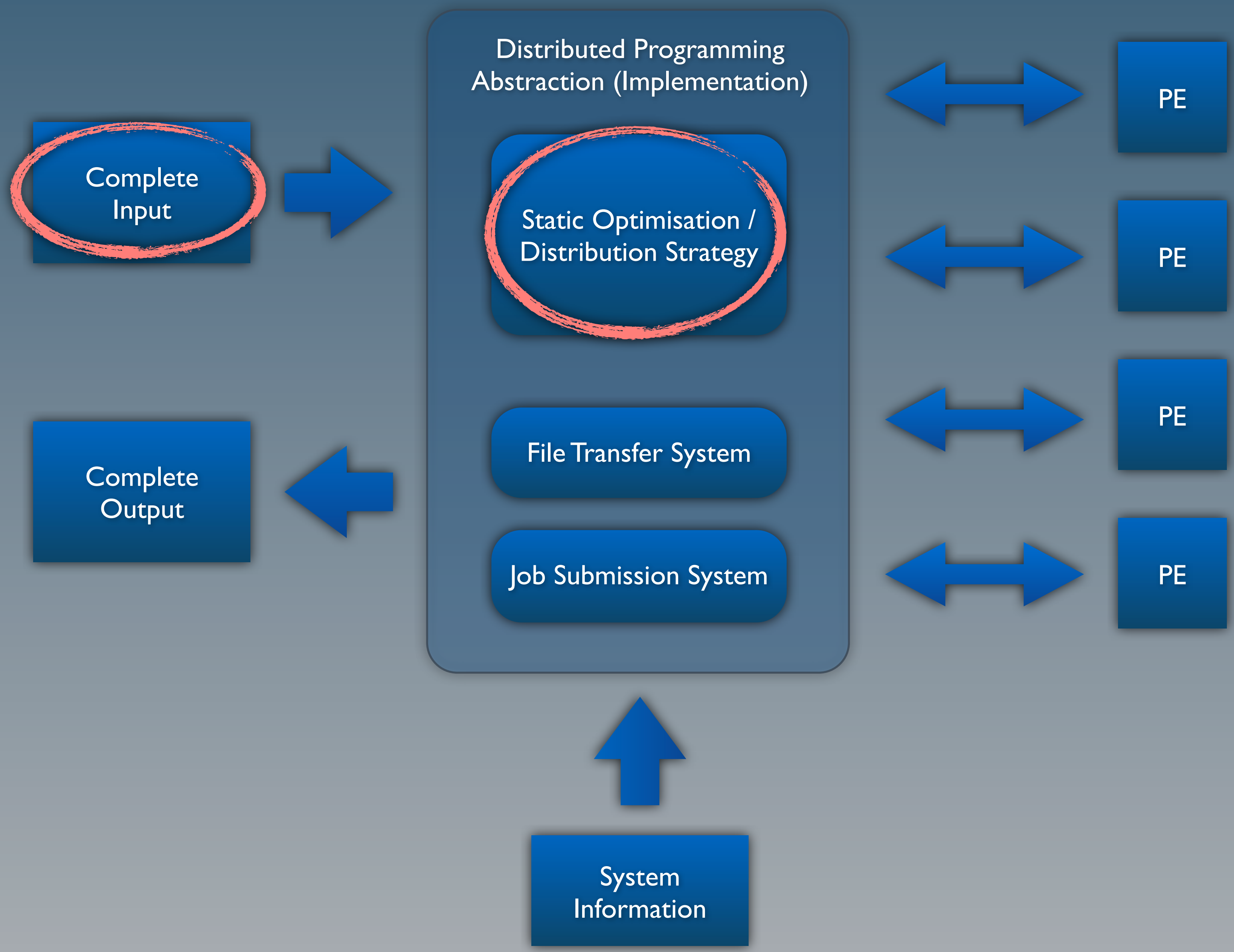
ole.weidner@ed.ac.uk

Everything is Static



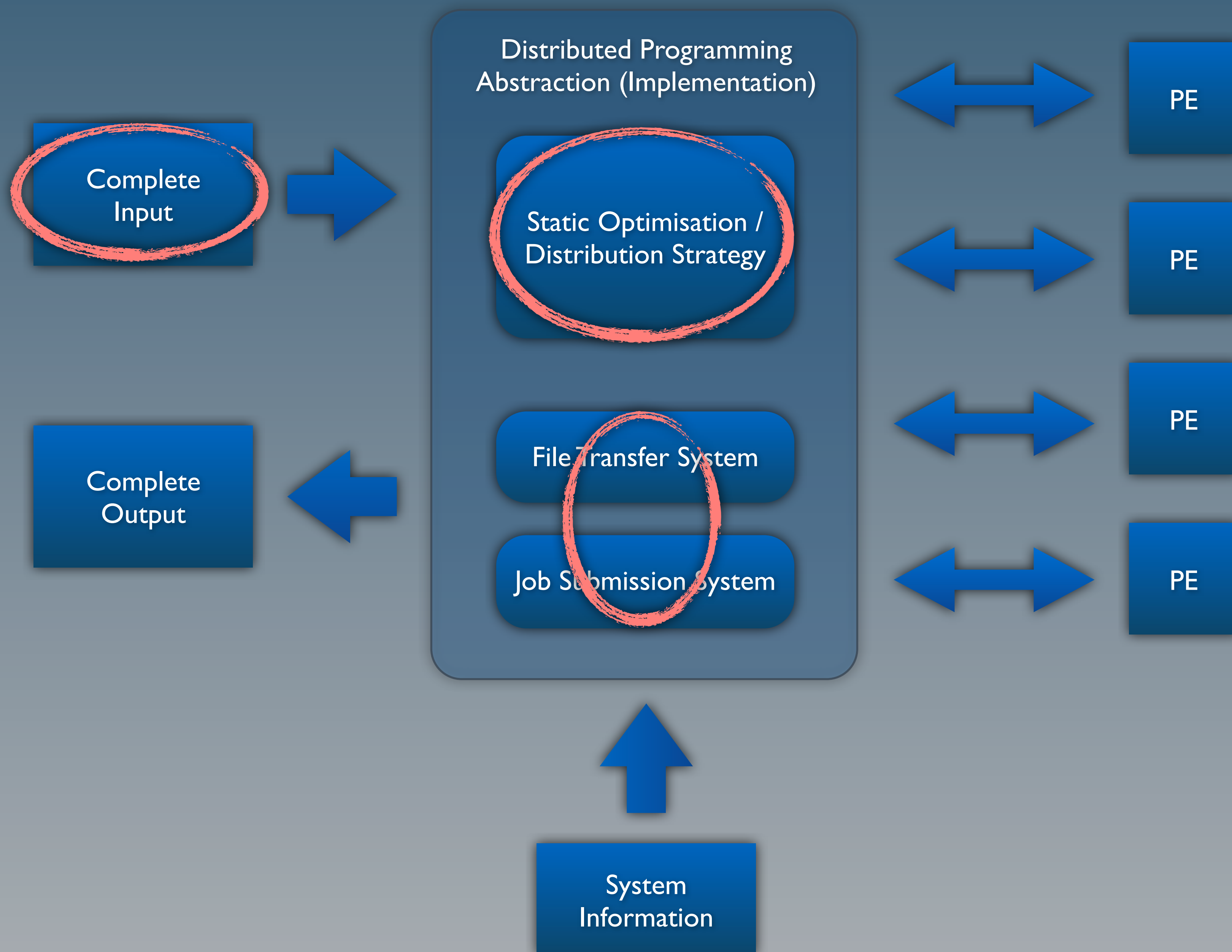
ole.weidner@ed.ac.uk

Everything is Static



ole.weidner@ed.ac.uk

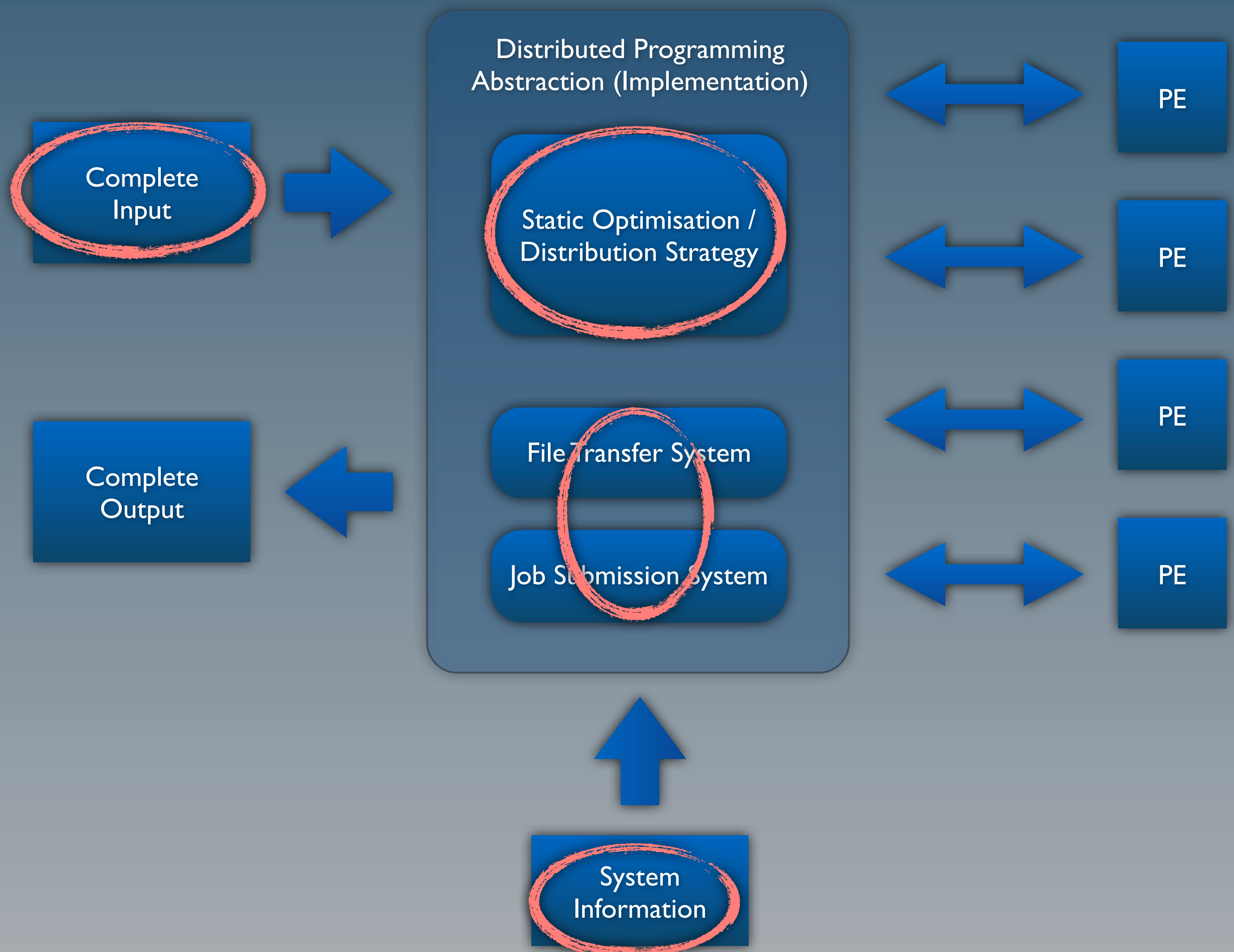
Everything is Static



ole.weidner@ed.ac.uk

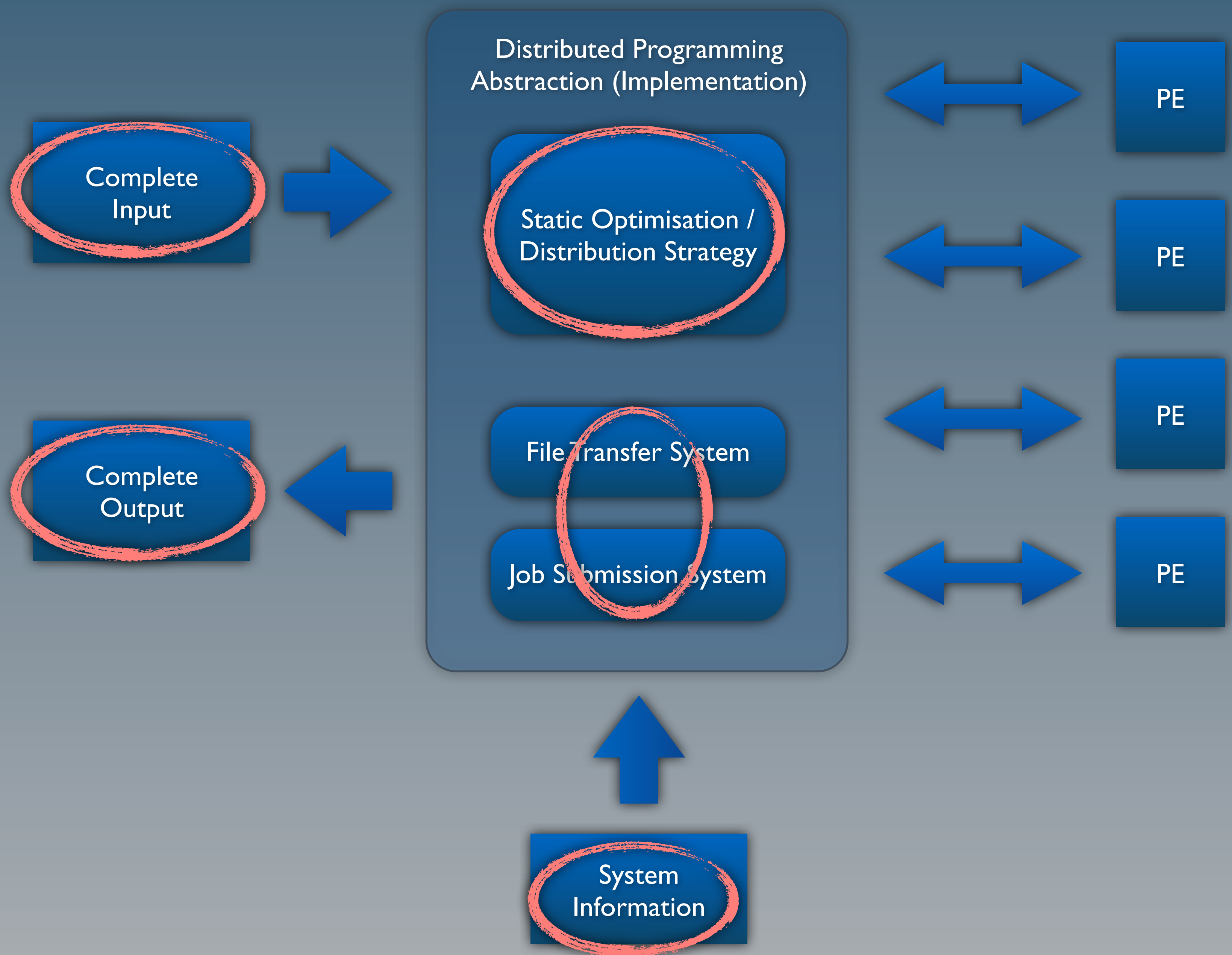
Everything is Static

edinburgh data-intensive research



Everything is Static

edinburgh data-intensive research



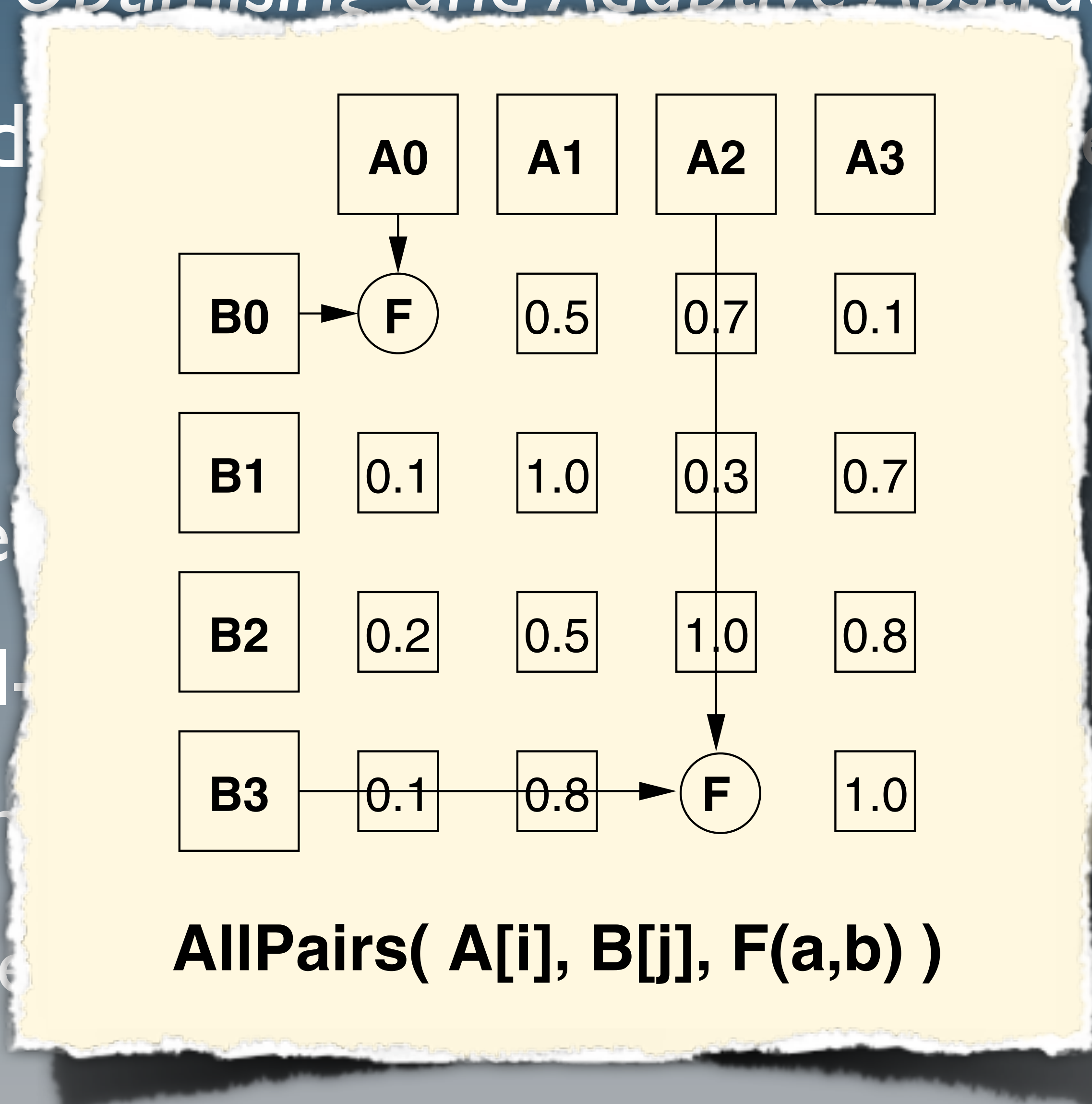
ole.weidner@ed.ac.uk

A *Somewhat* Concrete Plan

- Pick a common distributed abstraction and make it a "*Dynamically Optimising and Adaptive Abstraction*" (working title)
- A really good candidate so far seems to be the *All-Pairs* abstraction:
  - Simple and generic
  - Well defined input and output
  - Lots of real-world applications and data available
  - Many potential dynamic and big-data use-cases
  - Not confined to a specific type of infrastructure

A *Somewhat* Concrete Plan

- Pick a common distributed abstraction and make it a "*Dynamically Optimising and Adaptive Abstraction*" (working title)
- A really good … e the *All-Pairs* abstraction:
  - Simple and …
  - Well define…
  - Lots of real-… lable
  - Many poten… es
  - Not confin… ture



A *Somewhat* Concrete Plan

edinburgh
data-intensive
research

- Pick a common distributed abstraction and make it a "*Dynamically Optimising and Adaptive Abstraction*" (working title)
- A really good candidate so far seems to be the *All-Pairs* abstraction:
  - Simple and generic
  - Well defined input and output
  - Lots of real-world applications and data available
  - Many potential dynamic and big-data use-cases
  - Not confined to a specific type of infrastructure

All-Pairs Application Areas

edinburgh data-intensive research

- Bioinformatics
  - Phylogenetic tree generation
  - Sequence Alignment
- Biometrics
  - Feature (e.g., face) detection
- Machine Learning
  - Unsupervised learning (e.g., clustering, density estimation)
  - Evaluation of new learning functions
- *N*-body simulation (Cosmology)

All-Pairs Application Areas (cont.)

edinburgh data-intensive research

- Web / Data-mining
  - (Near) duplicate document detection
    - "*more-like-this queries*", collaborative filtering
    - "*data-shedding*", duplicate deletion
  - Query refinement
  - Coalition detection
  - Interesting: some of the matching algorithms in data-mining already support speed v.s. accuracy tuning

A *Somewhat* Concrete Plan (cont.)

edinburgh data-intensive research

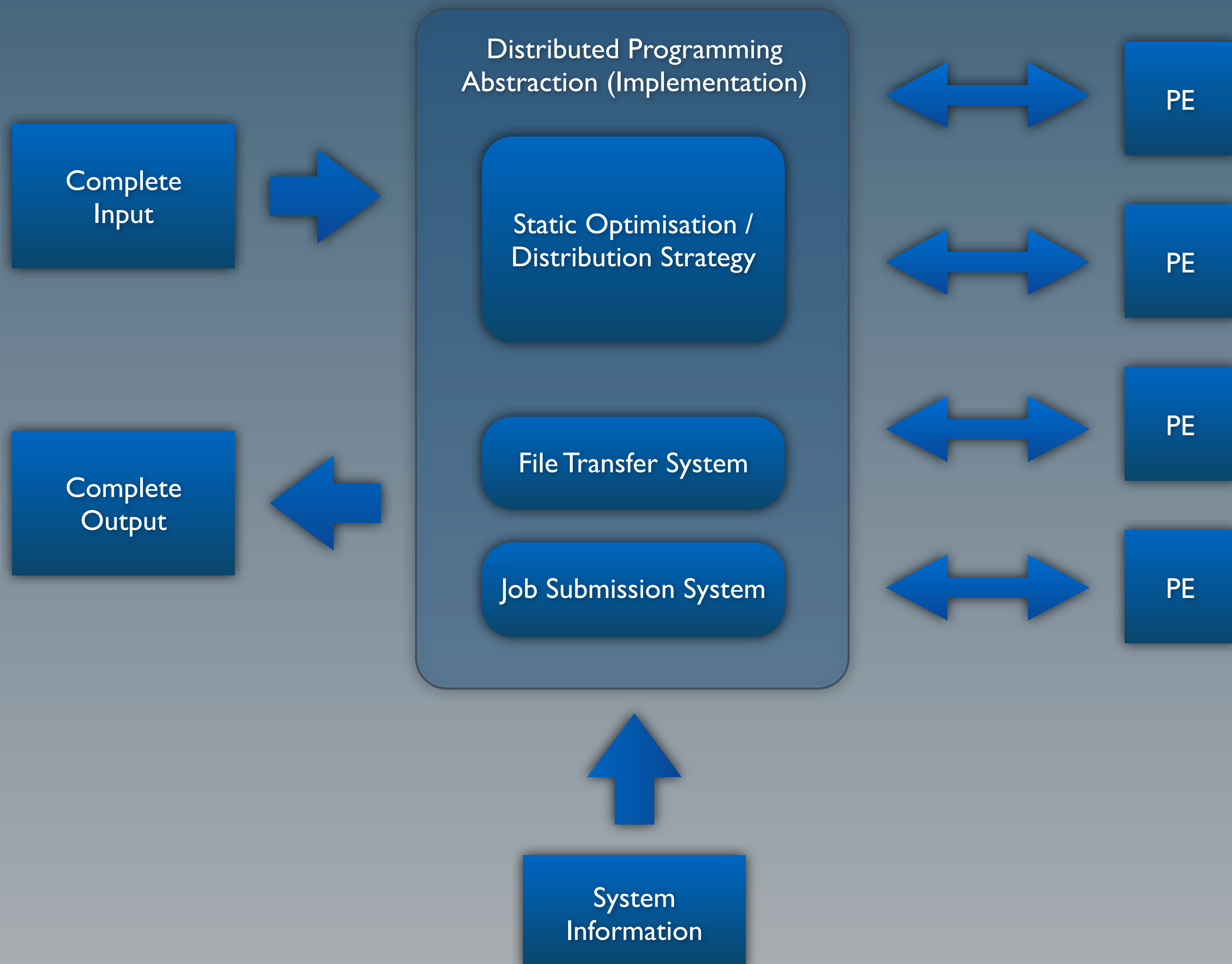
- Develop a model that helps us to understand the relevant aspects of data in an *All-Pairs* context
- Define the characteristics we need to capture / extract in order to support dynamic decision-making
  - Static / dynamic ?
  - Rate ?
  - Dependency ?
  - Affinity / Relationship ?

A *Somewhat* Concrete Plan (cont.)

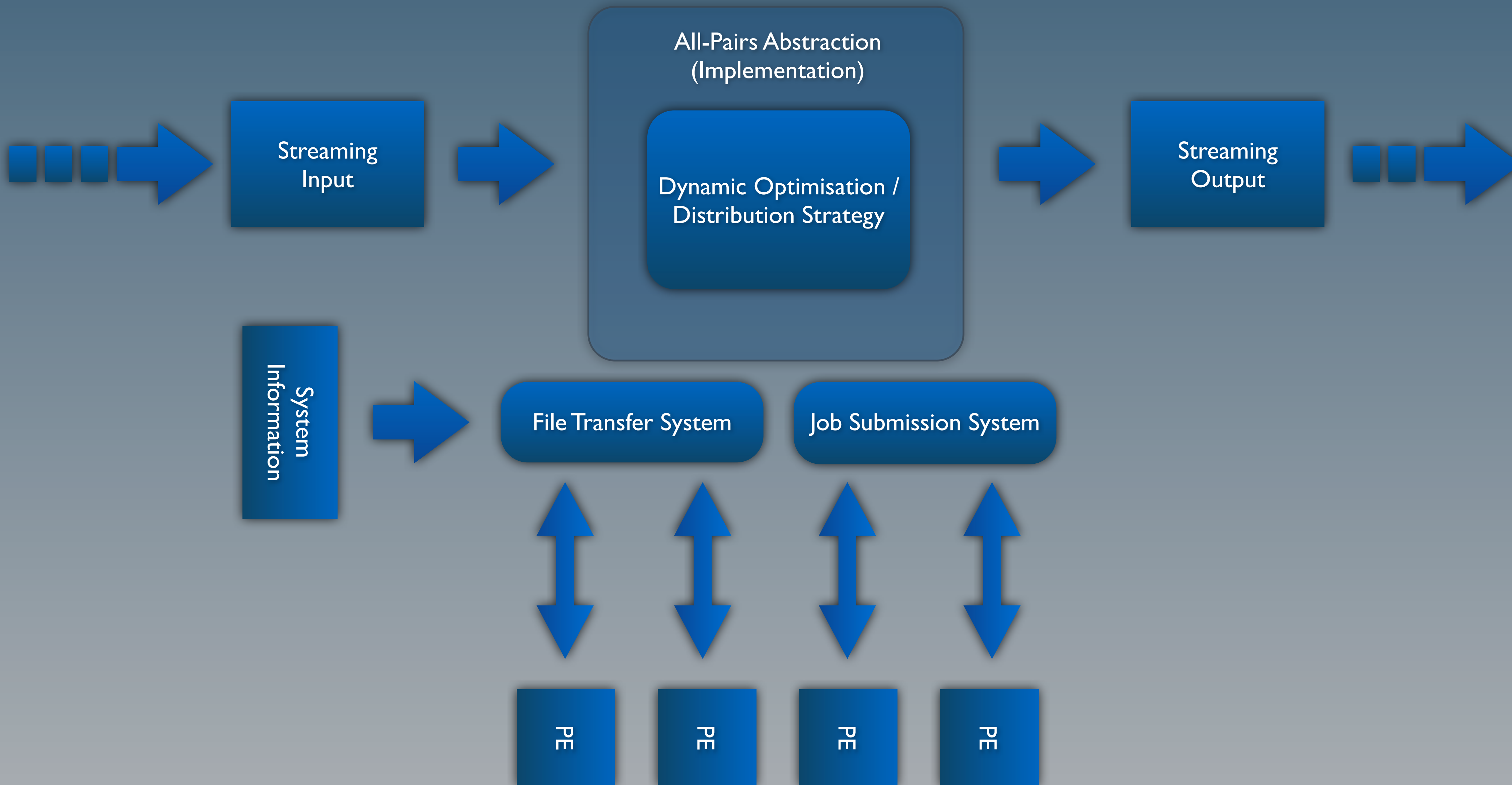
- Revisit the issue of optimisation objectives
  - Minimise energy consumption ?
  - Get me the best possible solution within X hours ?
  - Don't exceed bandwidth X or storage Y ?
- Develop an optimisation framework
  - Possibly based on control theory concepts
  - "*Control plant*" being the All-Pairs execution framework
  - *"Sensors"* measuring input and output data
  - *"Controller"* objective-specific optimisation rules and functions

A *Somewhat* Concrete Plan (cont.)

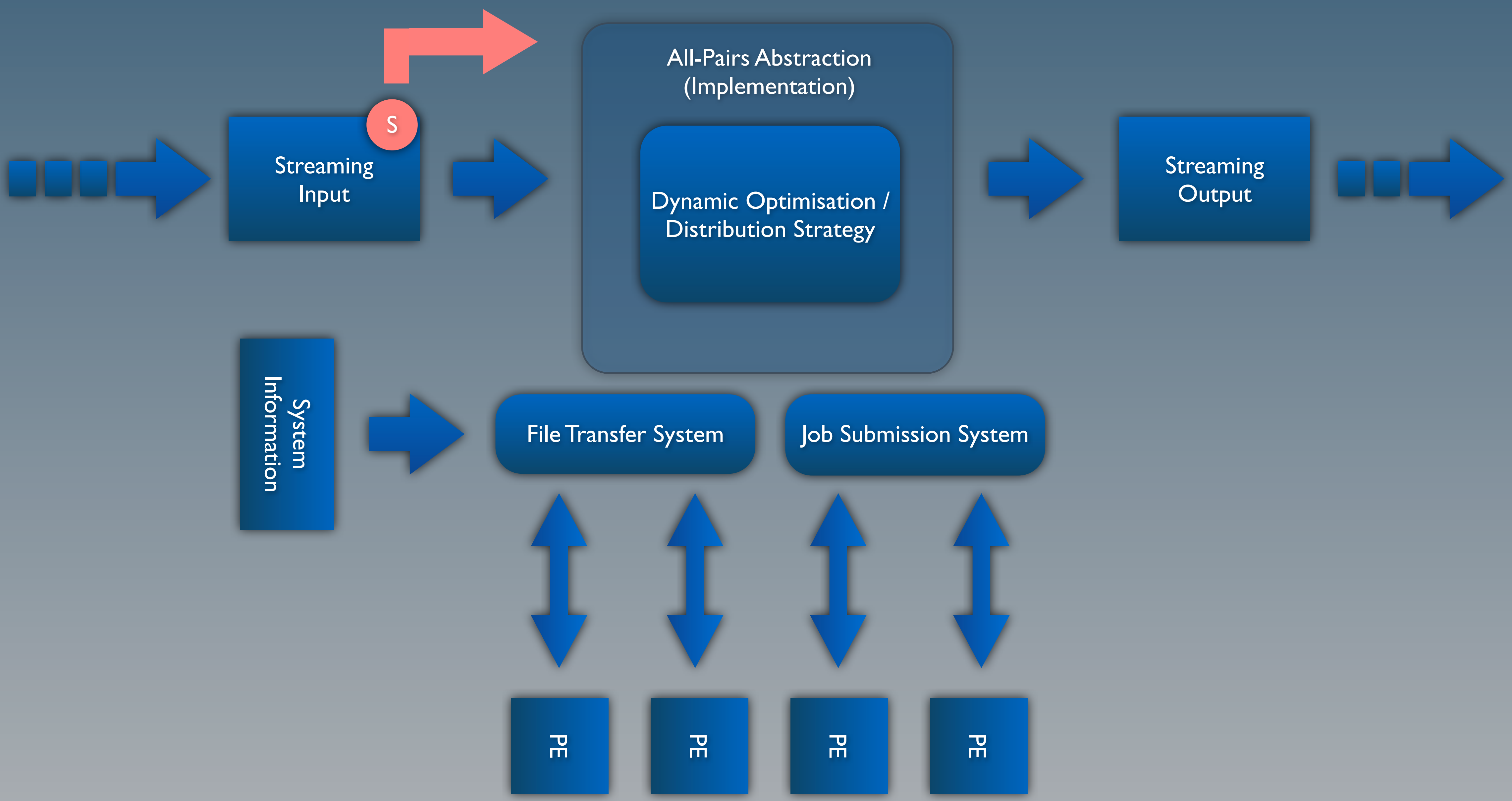
edinburgh data-intensive research



A *Somewhat* Concrete Plan (cont.)

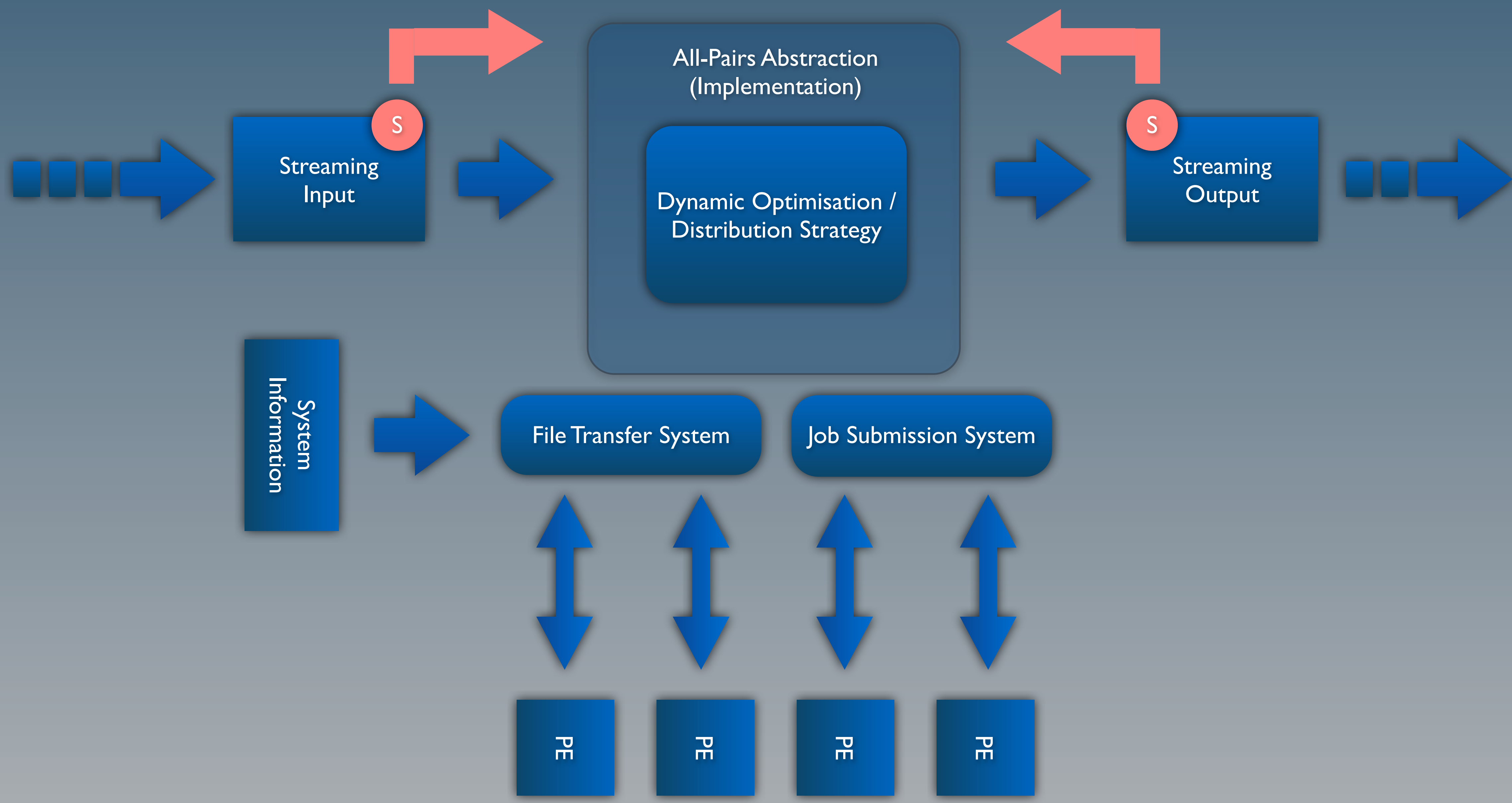


A *Somewhat* Concrete Plan (cont.)



A *Somewhat* Concrete Plan (cont.)

edinburgh data-intensive research



Lots of Open Questions

edinburgh data-intensive research

- Can we predict data ?
- How do the optimisation rules and functions look like ?
    - Finite set of rules ?
    - Machine Learning ?
    - Bayesian Networks ?
    - . . .
    - A combination of the above ?
- Is dynamic optimisation cheap enough to use it with static problems as well ?
- . . .

Engineering Challenges

- What is the right level of abstraction ?
- How do we implement sensors ?
- How do we implement actuators ?
- How would an unobtrusive '*user-interface*' look like ?
- Can we be completely infrastructure independent ?
- What about fault-tolerance ?
- Can we develop a *generic* dynamic optimisation framework that can be used not just for All-Pairs ?

Application Challenge

- How can we find real-world dynamic applications ?
    - *“Social engineering”*
- Can we generate synthetic workload ?
- What are the testbeds we should use ?
    - TeraGrid (HPC Grid)
    - EGI (HTC Grid)
    - Clouds ?

More Information ?

SAGA:

C++/Python: http://saga.cct.lsu.edu

OGF GFD.90: http://www.ogf.org/documents/GFD.90.pdf

Mailing-Lists: saga-users@cct.lsu.edu

My Ph.D. Research:

Homepage: http://www.oleweidner.com

Code Repo.: https://github.com/oweidner

Email: ole.weidner@ed.ac.uk