Programming SZTAKI Desktop Grid and XtremWeb by DC-API

József Kovács,
smith@sztaki.hu
MTA SZTAKI

The EDGeS project receives Community research funding

Introduction

- Grid systems are useful only if there are applications and users
- Application developers do not like learning grid internals
- Application developers do not like complex APIs
- Porting applications to a new grid system should be easy

Motivation

- SZTAKI has created the public SZTAKI Desktop Grid project: http://szdg.lpds.sztaki.hu
  - Based on BOINC
- The SZTAKI Local Desktop Grid is a customized version of BOINC tailored for educational and industry use
  - Easy installation, custom helpers
  - Reduced web interface for easier control
- We needed an API for developing applications that hides the details of BOINC, so we are not bound to it. By hiding the native API we allow the middleware to be replaced in the future if a better alternative surfaces

Tasks to Solve

- BOINC has a rather strict programming model
  - Master-worker concept. The master runs near the BOINC server while workers are stand-alone applications running on the client nodes
  - Communication is only between the master and the workers, never between two workers
  - Communication is via input/output files. There is limited messaging support, but it is unreliable by design and can have an arbitrarily high latency.
  - Support for redundant computing

Related Work

- There are existing API definition efforts for traditional grid systems
  - GAT – developed by the GridLab project
    - Supports run-time switching of the middleware
  - SAGA – the successor of GAT under the OGF umbrella
    - Key areas are security, data management, job management and inter-process communication
  - DRMAA – job submission, monitoring and control API

Related Work

- Problems with existing APIs
  - Different programming model
    - Generic vs. strict master-worker with no internal communication between workers
    - In BOINC, the features available for the master and the clients are disjunct
    - Many features could not be implemented on top of BOINC so the usefulness of such an implementation would be questionable
  - The BOINC environment is much more limited, most of the complexity present in traditional grids is not needed
  - GAT, SAGA and DRMAA are much more complex than the BOINC native API

Related Work

- The existing APIs are also missing some DG-specific functionalities
  - Support for redundant computing. In BOINC, contrary to traditional grids, computing resources are inherently not trusted
  - Result validation
  - Logical name resolution on the client side
  - Checkpointing support on the client side

What is DC-API…

- The DC-API is designed to be simple and easy to use
- It supports a master-worker programming model only
  - Workers are stand-alone sequential programs
- Primarily targeted for developing applications for desktop grids, but it is not bound to BOINC
- Can be implemented on top of other grid systems or job managers
  - The different backend implementations are not run-time switchable, relinking of the application is needed

What is not DC-API…

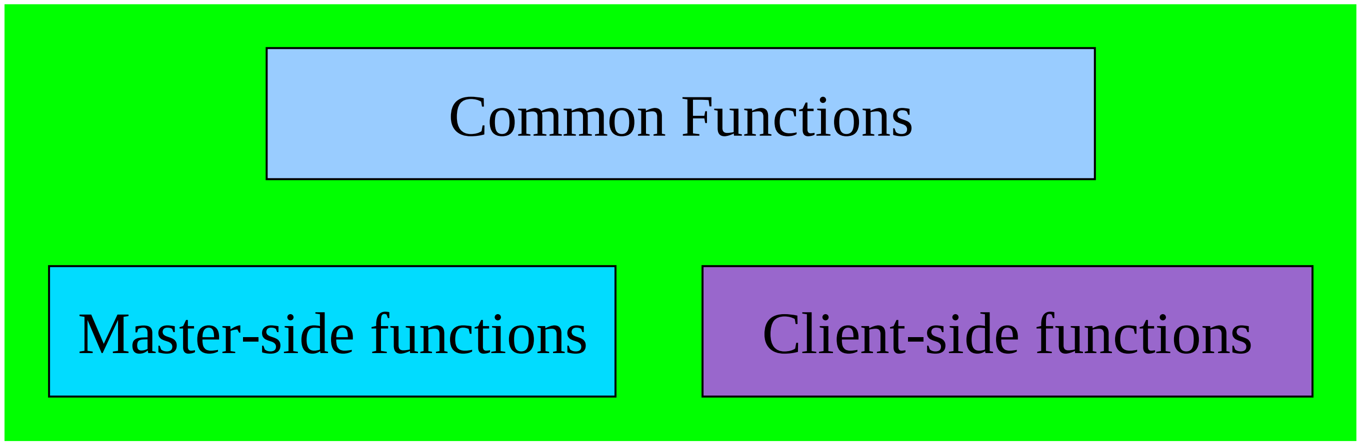
- The DC-API does not cover application deployment
  - That is very infrastructure-dependent
  - The SZTAKI LDG packages contain helper scripts for that
    - These scripts however have some knowledge about DC-API

DC-API implementations

- Existing DC-API implementations (backends)
  - BOINC – the original implementation
    - Supports all important components of the BOINC API
    - Intentionally lacks support for too specific features like the graphical screen saver on the client machine
  - Condor
    - NFS is mandatory
    - The master must be started on a submit node
  - XtremWeb
  - Local processes – to make application debugging easier
- On the following slides mainly the BOINC implementation will be discussed

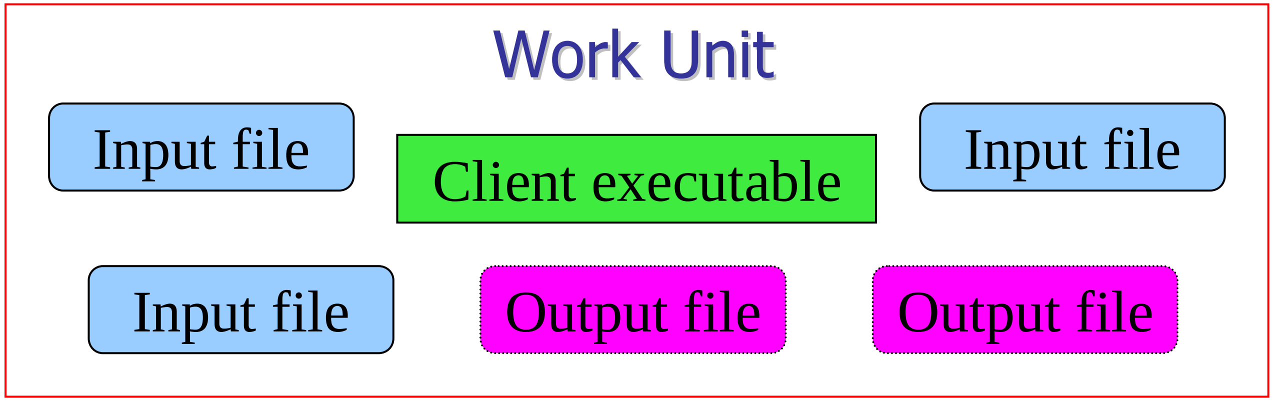
DC-API function classes

- The DC-API has two major components: the client-side API and the master-side API
- There are also a set of common utility functions (logging, configuration management etc.)



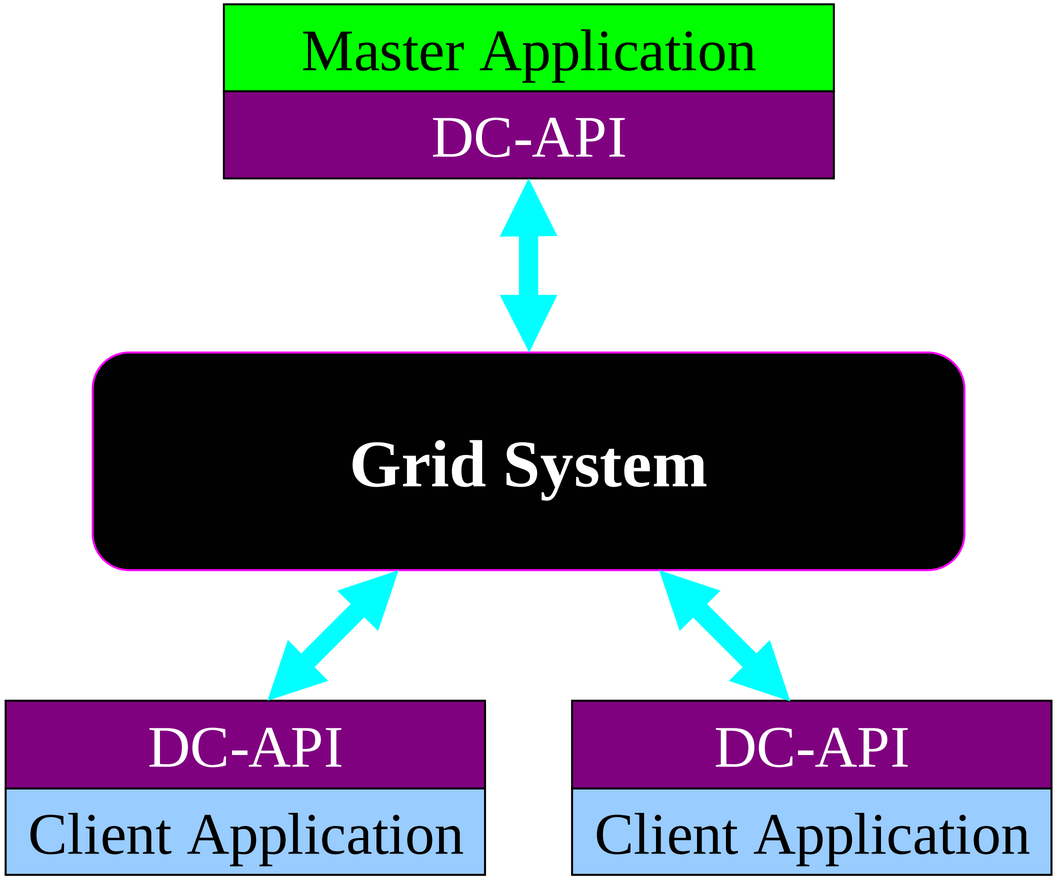
BOINC workunit structure

- Following BOINC terminology, the master divides the task into workunits
- A workunit encapsulates the client executable, its input files, and the description of the output files to return



Application Structure

- The master application creates workunits and submits them to the grid system
- The grid system acts like a real black box: you either get back results or a notice of failure
- The grid system is responsible for sending the workunits to the appropriate client nodes
- The DC-API provides the link between the master application, the grid system and the client application(s)



Common Functionality

- There are some functions available both on the master and on the client side. These functions include:
  - Logging functions
  - Configuration file processing
  - Querying the capabilities of the grid middleware

The Master Application

- The DC-API master application runs on a central server. It is responsible for splitting the global input into reasonable pieces (workunits), submitting the workunits to the grid infrastructure, and combining the received results to form a global result
- Work units consist of
  - The logical name of the client application. The physical name of the application is determined by the middleware or DC-API based on e.g. the architecture of the target machine
  - The input files
  - The list of expected output files. All output files must be declared explicitly
  - A configuration file for the client-side DC-API implementation

The Master Application

- The main loop of the master application is event based
  - When an event occurs (such as a workunit has completed or failed or a message has arrived), the DC-API notifies the master using callbacks
  - The master can query a limited set of grid status variables to decide when to submit new work units
- The DC-API uses a configuration file that contains the backend-specific parameters
  - Location of the BOINC project files
  - Required redundancy level
  - Client application name mapping when the middleware does not have such a feature

Typical steps of a master application

- Initializes the DC-API library by calling DC_initMaster() function.
- Calls the DC_setResultCB() function and optionally some of the DC_setSubresultCb(), DC_setMessageCb(), DC_setSuspendCb() and DC_setValidateCb() functions, depending on the features (messaging, subresults etc.) it wants to use.
- In its main loop, the master calls the DC_createWU() function to create new work units when needed. The master may use the DC_getWUNumber() function to determine the number of running work units, and create new work units only if this number falls below a certain threshold.
- Also in its main loop the master calls the DC_processMasterEvents() function that checks for outstanding events and invokes the appropriate callbacks.
- Alternatively, the master may use the DC_waitMasterEvents() and DC_waitWUEvent() functions instead of DC_processMasterEvents() if the developer prefers to handle event structures instead of using callbacks.

The Master Application

- DC-API does not do everything. A real-world master application still has to implement things like:
  - Checkpointing – DC-API helps with saving the WU states, the rest is up to the application
  - Preventing the middleware from being flooded. Instead, the number of active workunits should be kept at a constant level

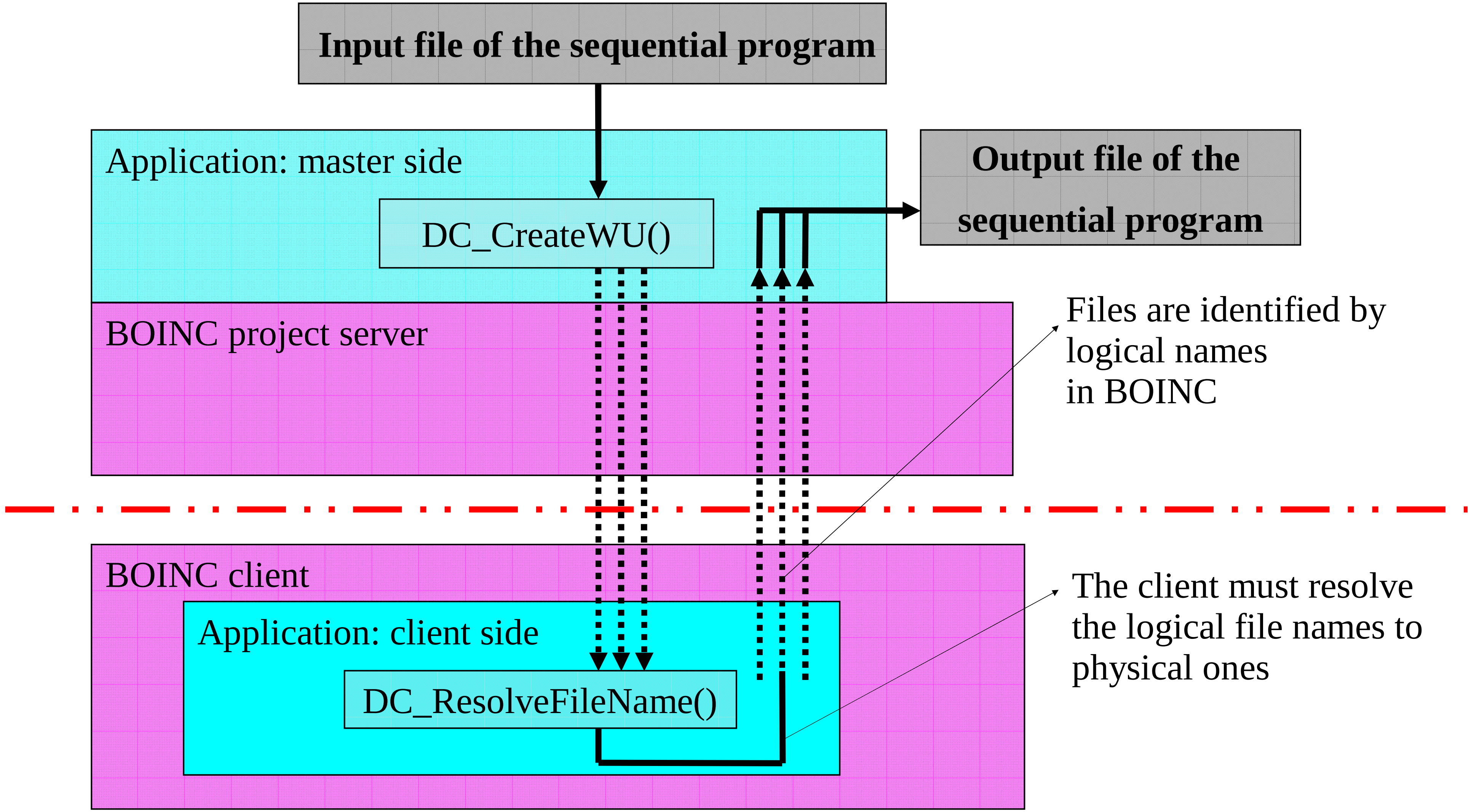
The Client Application

- Client applications are stand-alone sequential programs that run on the worker nodes and perform the real computation
  - Must take their input from files or command line, no interactivity is possible
  - Since desktop grids often have clients of various architectures/operating systems, the client applications should be portable
- Application-level checkpointing is essential in a non-dedicated desktop grid environment
  - DC-API delivers the checkpoint request signal and can notify the middleware when the checkpoint has been completed but it is up to the application to implement the checkpoint itself

The Client Application

- The following places need modifications for adapting an existing applications to DC-API:
  - Program startup and exit
  - Opening a file – the file name must be mapped from the logical name used by the client to a physical file name
  - Periodically checking for events such as checkpoint request

How Files Move Around



Special File Handling

- Besides the input and output files specified by the master application when a workunit is created, the DC-API also handles some special files automatically
  - A configuration file is sent for every workunit, so settings in the master's configuration can be propagated to the clients
  - The standard output and error is always collected and transferred back to the master
    - There is no support for the standard input

Other Features

- Limited support for messaging
  - Non-reliable, only between master-client. This is an inherent limitation of BOINC
  - Large delays are possible. In BOINC, the master can not initiate communication, the client must poll the master for new messages
  - Redundancy is problematic
- Checkpointing support
  - Both on the client and on the master side
- Integration with the SZTAKI Local Desktop Grid packages
  - Auto configuration of grid-specific parameters
- Subresults for sending back intermediate results
  - The max. number of subresults must be specified when the workunit is created
  - Handling of subresults can be difficult when redundancy is enabled

Example: Client Side

- Including the DC-API header

```
#include <dc_client.h>
```

- Initializing the DC-API on startup

```
int main()
{
  DC_initClient();
```

Example: Client Side

- Resolving logical file names

```
char *real_name =
  DC_resolveFileName(DC_FILE_IN,
                     INPUT_FILENAME);
read_input_file(real_name);
...
real_name =
  DC_resolveFileName(DC_FILE_OUT,
                     OUTPUT_FILENAME);
write_output_file(real_name);
```

Example: Client Side

- Exiting the application must go through DC-API, too

```
DC_finishClient(exit_code);
```

- ... and that's enough for a basic functional client application

Features Specific to BOINC

- BOINC uses several variables to control how many times a work unit should be computed, how many successful results are needed, how many errors are tolerated etc. The DC-API uses a single "Redundancy" parameter that can be specified in the configuration file, all the BOINC-specific parameters are derived from this number
- There is a slight terminology skew: BOINC results are workunit instances waiting to be downloaded to a specific client; DC-API results are more like the "canonical result" concept in BOINC
- BOINC requires specifying resource limits for client applications in advance. These limits can also be specified in the DC-API configuration file

Existing Applications

- BinSYS: searching for generalized binary number systems – runs on the public SZTAKI desktop grid
- Cancergrid: drug discovery and testing
- UC-Explorer: understanding basic universality classes of nonequilibrium systems
- Alias-free Digital Signal Processing (Univ. of Westminster)
- 3D Video Rendering (Univ. of Westminster)
- … and many more in EDGeS

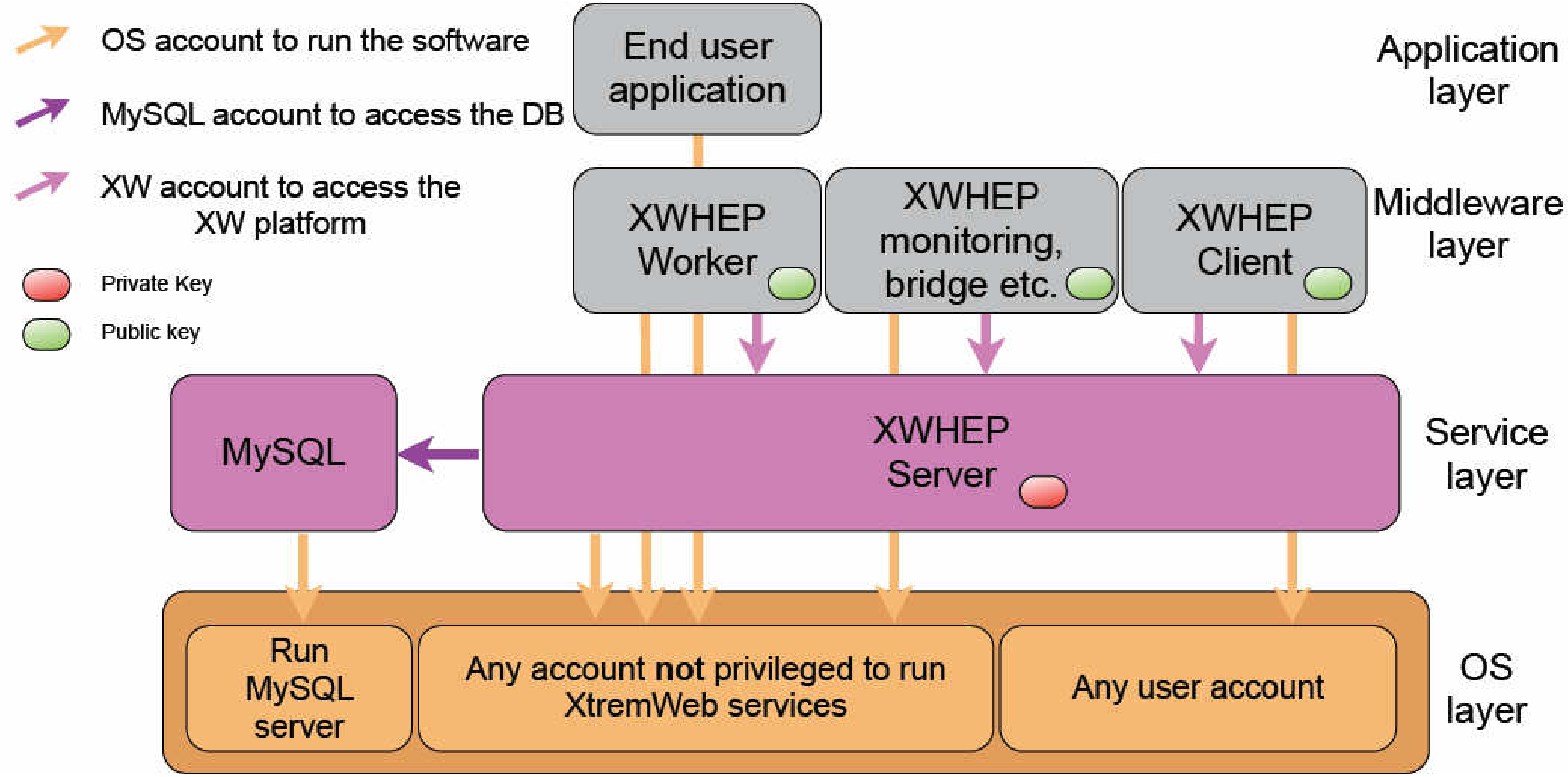
For more applications listen to the following presentation:

13:30 Applications ported to EDGeS (30') Tamás Kiss (Thursday)

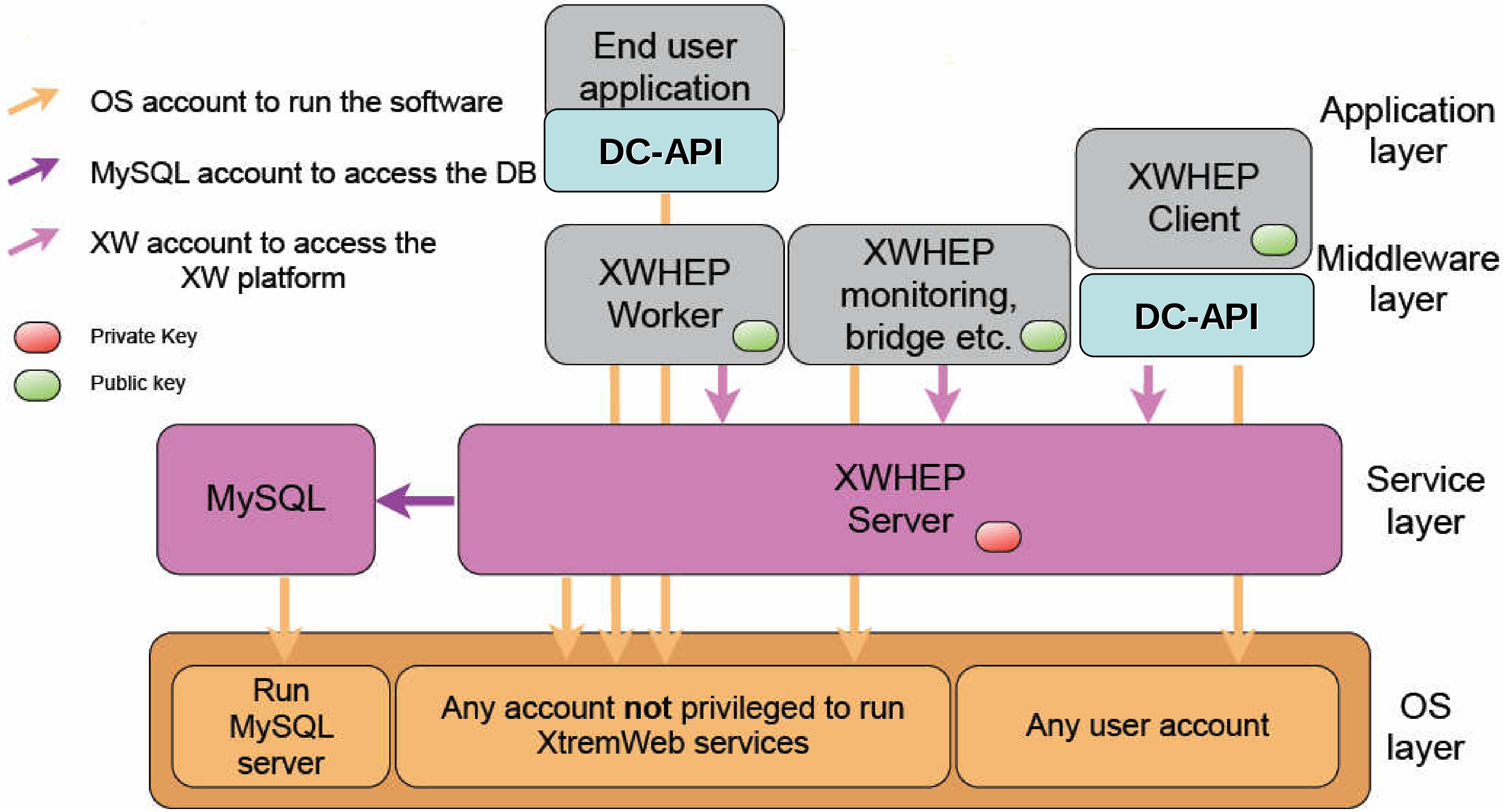
Language Support

- The primary implementation of DC-API has a C/C++ interface
- There is experimental support for Java on the client side
  - Uses JNI glue code between the Java code and the DC-API/middleware
  - There are still some issues that need to be solved before it can be used in production
- Once there was a Fortran interface but it got removed because we did not have any users/testers
  - It can be added back if really needed

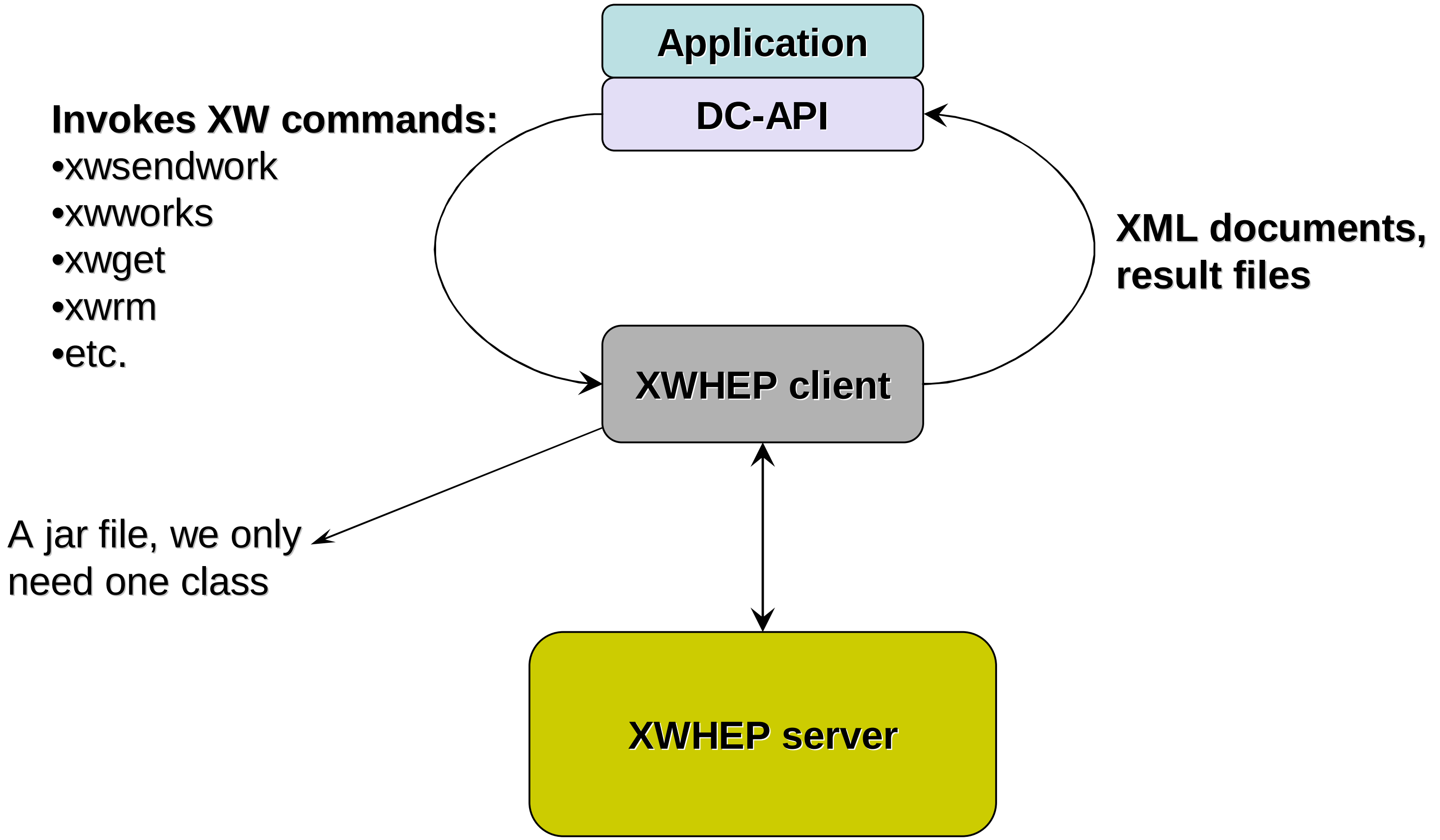
XtremWeb architecture



The DC-API backend



Implementation (master)



Implementation (client)

- Simple wrapper
- XtremWeb lacks many of the features of DC-API:
  - No need for resolving files
  - No checkpointing
  - No events
- XtremWeb does not need any client-side coding
- Some functions had to be implemented to ensure compatibility with other backends

Future Directions

- More implementations
  - Other desktop grids
  - Traditional grids
- Support more languages
  - Exists: C/C++ (working) and Java (experimental)
- More high-level features
  - Variable number of input/output files
  - Extended support for legacy code

Conclusion

- The DC-API is a simple, easy to use grid API
- It was designed to aid the development of desktop grid applications but it can also be adapted to other grid systems
- Together with the SZTAKI Local Desktop Grid packages, the DC-API makes it very easy to set up a desktop grid system and develop/port applications
- This is one of the most important tool in EDGeS in porting applications to DesktopGrids

To use DC-API, take part on the following hands-on:
14:30 Programming SZTAKI Desktop Grid and XtremWeb by DC-API
(40') Attila Marosi (Thursday)

If you need more detailed (technical) information, email to desktopgrid@lpds.sztaki.hu or visit www.desktopgrid.hu

Thank you for your attention!

Questions?

Special thanks to

Gábor Gombás (SZTAKI, developer of DC-API for BOINC) and

Gábor Szmetankó (UoW, developer of DC-API for XtremWeb)

for the slides!