BOINC II

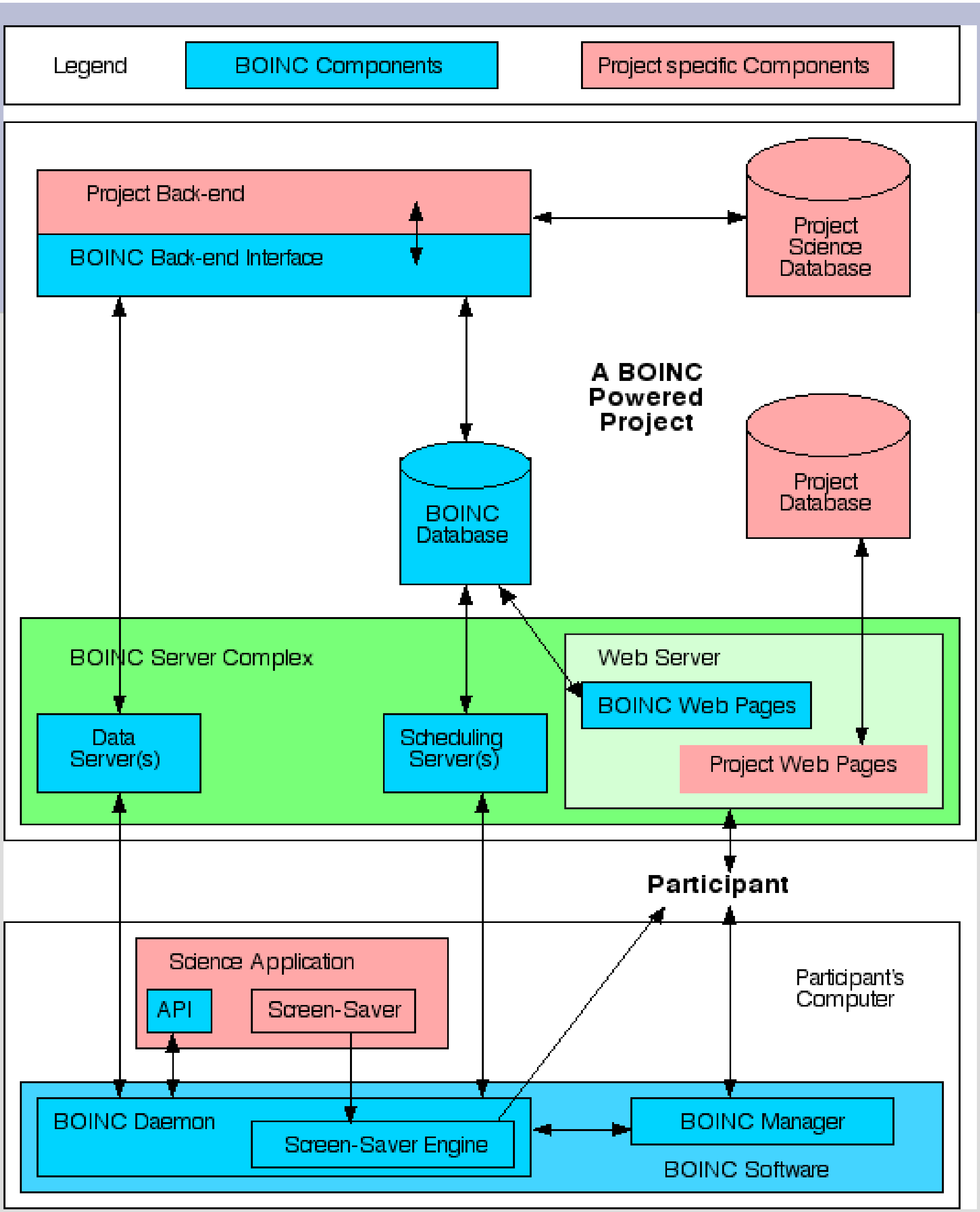
Nicolas Maire, Swiss Tropical Institute

with Christian Ulrik Søttrup, Niels Bohr Institute

Overview BOINC II

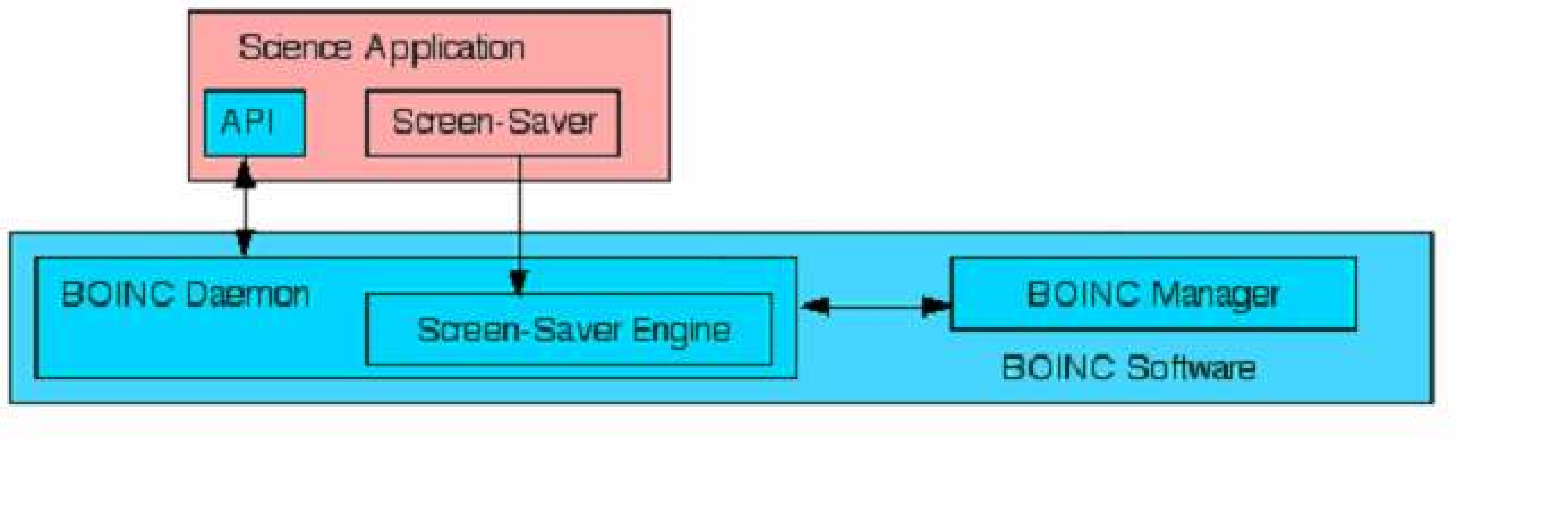
- BOINC architecture
- BOINC Client
  - Core client and manager
  - BOINC API
- Server architecture
  - DB
  - Daemons and tasks
  - Project directory structure
  - Templates
  - Configuration
  - Some examples from malariacontrol.net
- Client-Server interaction
  - Scheduling server protocol
- Server deployment

Architecture



Client

- Project-independent
- Communicates with the project server(s)
- Download and upload of data
- Update of science application
- Launches and monitors the science app



- Note: New BOINC API separates screensaver into separate program

BOINC-API

- For science applications to communicate with the core client (project independent client)
- The BOINC API is a set of C++ functions.
- Not covered here: Graphics-API

Initialization and termination

- `int boinc_init();`
  - Call before any other BOINC functions
  - Several initialization tasks, e.g. parse init_data.xml
- `int boinc_finish(int status);`
  - Call after science application terminates
  - Let the BOINC client know we’re done, and if we’ve succeeded

Resolving file names

-
  ```
  int boinc_resolve_filename(
     char *logical_name, char *physical_name,
     int len);
  ```

- convert logical file names to physical names
  - **Logical name**: the name by which the science application will refer to the file
  - **Physical name:** unique identifier for the file

Checkpointing

- Write the state of the job to disk, in order not start from scratch if the computation is interrupted
- **`int boinc_time_to_checkpoint();`**
  - Checkpointing frequency is a user preference
  - Science application ask BOINC if it’s time for a checkpoint at a suitable place
  - Checkpoint immediately if returns non-zero (true)
- **`void boinc_checkpoint_completed();`**
  - Tell BOINC we have checkpointed, to reset the timer to the next checkpoint

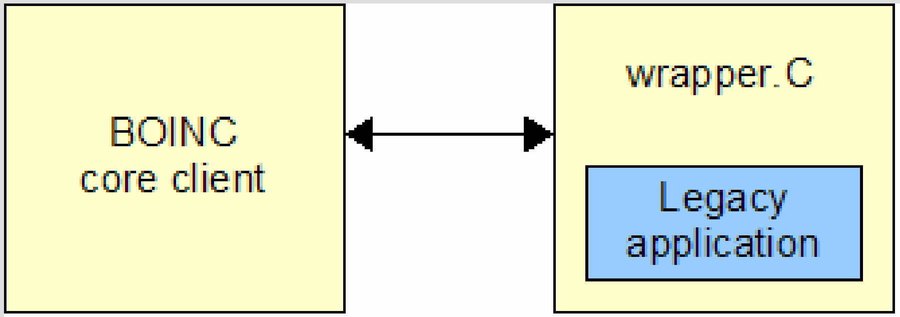
Reporting progress

- **`boinc_fraction_done(double fraction_done);`**

- The client GUI displays percent done of a running workunit
- The user can see that the workunit is running ok
- The malariacontrol science application updates this after each completed 5-day time step with the proportion of competed simulation steps

Legacy applications

- Not possible to use BOINC API
  - No source code
  - But also if language does not allow C-calls
  - Or simply no resources for SW-development
- Possible to run under BOINC using the “wrapper approach”
- The wrapper handles communication with the core client, and runs science application as a subprocess



Database I

- BOINC stores state information in a mysql database
  - platform
    - Compilation targets of the core client and/or applications.
  - app
    - Applications. A project can run several science applications
  - app_version
    - Versions of applications. Includes URL, and MD5 checksum.
  - user
    - including email, name, web password, authenticator.

Database II

- host
  - OS, CPU, RAM, userid,reliability
- workunit
  - Contains input file descriptions. Includes counts of the number of results linked to this workunit, and the numbers that have been sent, that have succeeded, and that have failed.
- result
  - Includes state and a number of items relevant only after the result has been returned: CPU time, exit status, and validation status.

- Web-interface related tables

Scheduler

- The scheduler is a cgi script that is contacted by the client.
- By default, a new instance is spawned for each connection (but can use fast CGI).
- The instance will then find an available job and give it to the client.
- The scheduler can run on its own machine

Feeder

- The Feeder takes jobs (results) ready for execution and places them in a queue in memory.
- This queue is used by the scheduler.
- More efficient than letting each scheduler instance create a database connection.
- Feeder return jobs arbitrarily but generally with increasing id.
- Prioritization of workunits and weighting of applications is possible

Transitioner

- Takes care of state transition for WUs.
  - Create results from WUs.
    - Newly created WUs
    - Timed-out results
  - Flags results for:
    - Validation
    - Assimilation
    - Deletion
- Can be split into many instances, each taking care of a subset of Wus. This also goes for most other daemons.

Validator

- Validates results
  - Once enough (configurable) have been marked with NEED_VALIDATE by the transitioner.
  - Validator compares the results using a project supplied algorithm.
    - complete binary equality
    - One that compares only parts of the results and 5% discrepancy in those parts.
  - This means that you may have to write your own validator, i.e. you must decide what is a valid result.
- Chooses canonical result and grants credits
  - Credit granting algorithm can also be supplied.

Assimilator

- The assimilator must also be supplied by the project.
- It must process the canonical result.
  - Could copy result to a result database
  - Could extract data from result and do calculation based on that
  - Could even generate new jobs based on data from result
- Mark results as assimilated

File deleter

- Once a job is done and the WU has been marked as ready for deletion, the file deleter will delete all input and result files from that WU on the server.
- Option to: preserve_wu_files, preserve_result_files

DB purger

- This daemon will move database entries that are old and no longer needed to an XML storage file.
- This clears up the result and workunit tables that could otherwise easily become so big they could not fit in RAM.
- Projects typically keep results at least a few days in the DB, so that users have a record of their recent contribution

Project directory structure I

- Apps
  - Contains applications(boinc clients, your science application)
- Bin
  - Boinc executables
- Cgi-bin
  - Scheduler and file upload handler
- Download
  - Input data and programs
- Upload
  - Result data

Project directory structure II

- Html
  - Project website and administration website
- log_servername
  - Logs for BOINC and project-specific daemons and tasks
- pid_servername
  - Lock files for daemons
- Templates
  - Templates for workunits and result xml templates

config.xml

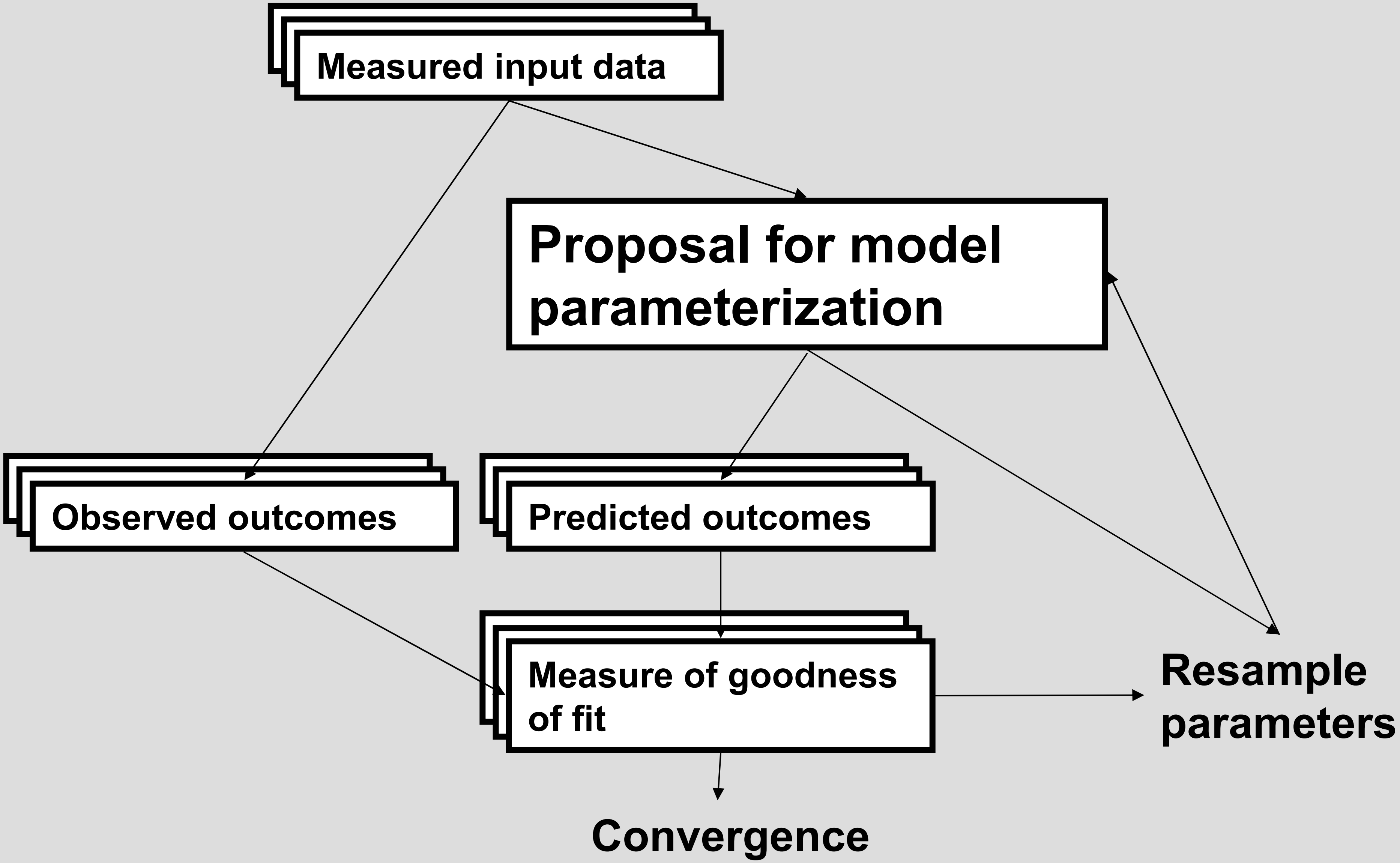
- Main project configuration file
- Options for disabling account creation, max wu per host per day, one result per user per WU, and many more
- Project specific tasks can be setup to be run by the main daemon

```
<boinc>

<config> [ configuration options ] </config>
<daemons> [ list of daemons ] </daemons>
<tasks> [ list of periodic tasks ] </tasks>

</boinc>
```

Generating work

- Write XML 'template files' that describe the job's input and outputs. Example templates follow later on.
- Create the job's input file(s)
- Invoke a BOINC function or script that submits the job

A BOINC project example: malariacontrol.net



Validator daemon

- Validate incoming results against others of the same workunit
- Grant credit
- Projects use the BOINC supplied C code and implement compare_results and compute_granted_credit functions
- Here we use the BOINC-provided sample_bitwise_validator

Assimilator daemon

- Processes validated results
- Reads the simulation output file
- Compares the predictions with the corresponding field data
- Computes a measure of fit
- Creates new work if necessary
  - Sample a new model parameterization and store it to the backend database
  - Based on the completed parameterizations in the backend database
  - Create workunit files, copy them to the download directory and all create_work to add to the BOINC database

mcdn config.xml

```
<boinc>
  <config>
<one_result_per_user_per_wu>1</one_result_per_user_per_wu>
…
<tasks>
  <task>
    <cmd>
      generator -db_name malariaModel -template_name
../templates/generator_template.xml -d 3
    </cmd>
    <period>1 min</period>
   </task>
</tasks>
<daemons>
  <daemon>
       <cmd>validator -d 3  -app malariacontrol</cmd>
  </daemon>
  …..
</daemons>
</boinc>
```

Templates I: Input template

- Input file references
- Workunit attributes

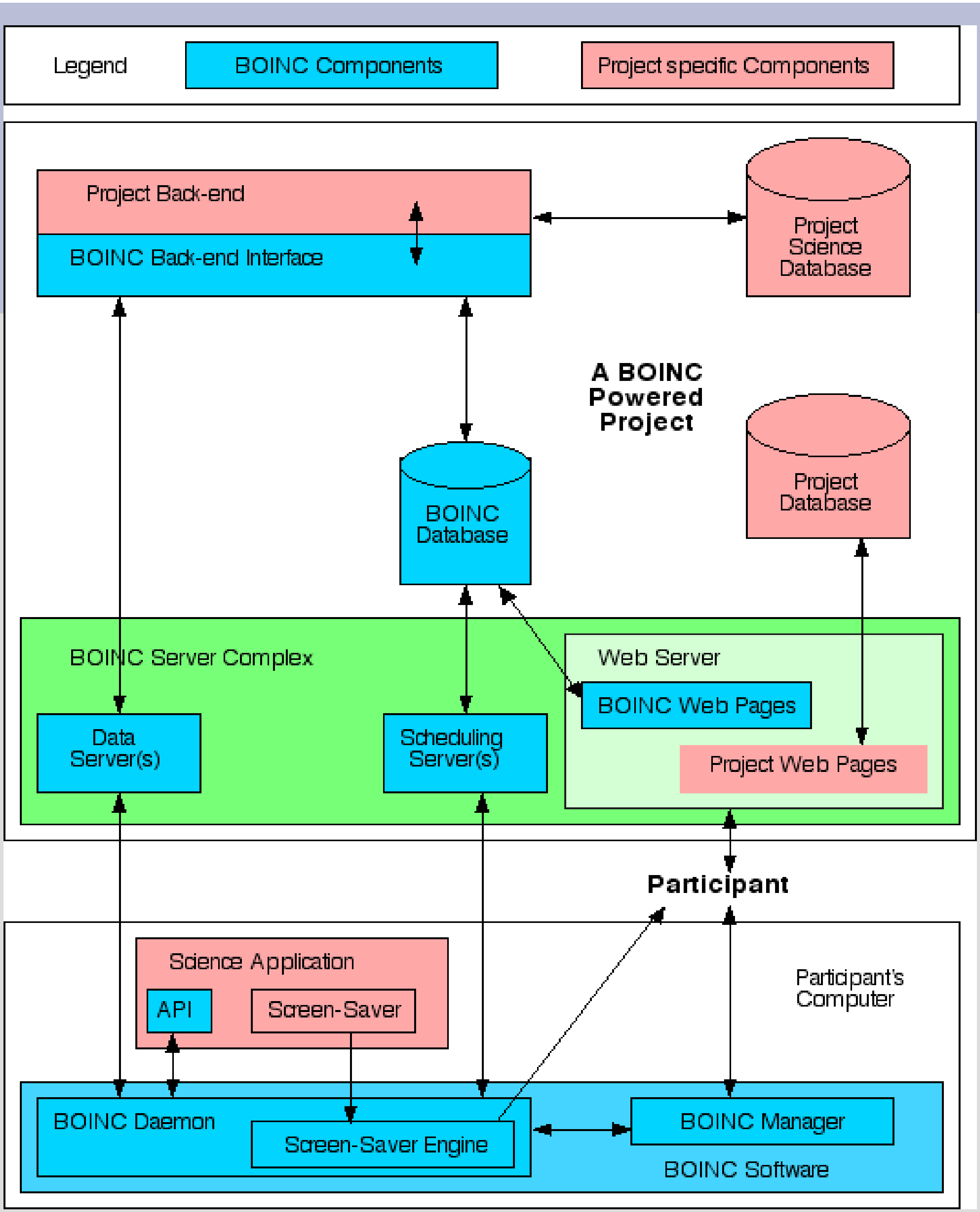
```
<file_info>
        <number>0</number>
</file_info>
<workunit>
<file_ref>
        <file_number>0</file_number>
        <open_name>scenario.xml</open_name>
</file_ref>
<min_quorum>2</min_quorum>
<rsc_fpops_bound>120000000000000.0</rsc_fpops_bound>
<rsc_fpops_est>10000000000000</rsc_fpops_est>
<delay_bound>300000</delay_bound>
<max_error_results>5</max_error_results>
</workunit>
```

Templates II: Output template

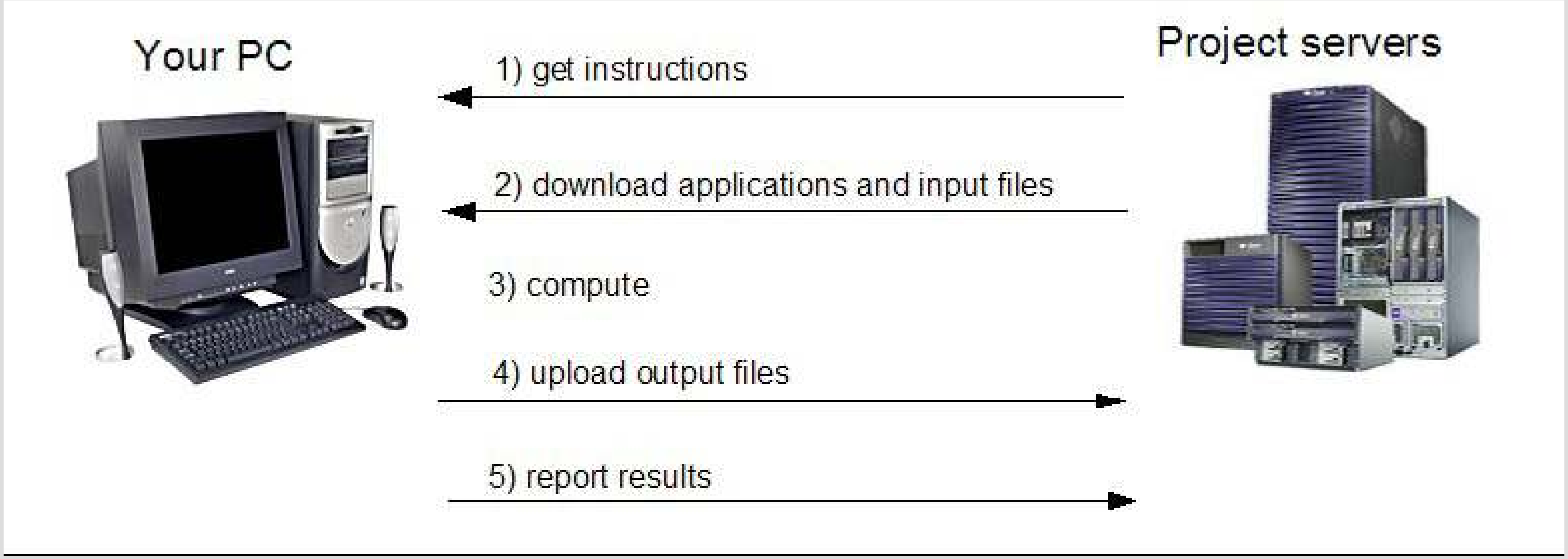
- Definition of output files and the way they are referenced

```
<file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <max_nbytes>10000000</max_nbytes>
    <url><UPLOAD_URL/></url>
</file_info>
<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>output.txt</open_name>
    </file_ref>
</result>
```

Architecture



Client-Server interaction



Your PC

Project servers

1) get instructions

2) download applications and input files

3) compute

4) upload output files

5) report results

Client-Server interaction

- Communication via http
- Scheduling server protocol:
- Client sends scheduler_request_X.xml
- Server replies with scheduler_reply xml
- Example request (fragment):

```
<scheduler_request>
<hostid>146</hostid>
  <core_client_major_version>5</core_client_major_version>
  <core_client_minor_version>10</core_client_minor_version>
  <work_req_seconds>234</work_req_seconds>
<global_preferences> …………. </global_preferences>
<result>
  <name>uc_1192745072_44_0</name>
  <final_cpu_time>9.890625</final_cpu_time>
  <exit_status>0</exit_status>
  <platform>windows_intelx86</platform>
  ……..
</result>
  ……...
</scheduler_request>
```

Server setup

- There is no official RPM or other package for the server
  - There are privately developed ones. CERN has one for scientific linux.
- By now installing a server works well on a range of Linux distributions and is well documented
- Many people have published guides and how-tos
- Use the resources shown at the end of the presentation

Prerequisites

- See Boinc website for latest list.
- Gnutools
  - Gcc, make, autoconf, automake ...
- MySQL
  - Server and client
- Python
  - MySQL and XML extensions
- Apache
  - Mod_ssl and PHP
- Openssl
  - v0.9.8+

Building

- Get the source from SVN repository
  - svn co http://boinc.berkeley.edu/svn/trunk/boinc
- Build it
  - ./_autosetup
  - ./configure
    - --disable-client
  - ./make
  - ./Make install

Installation

- Make sure that all prerequisites (apache, mysql, php, etc) are configured correctly.
- Create keys for uploads and downloads (code), preferably not on the server
  - Store the private key somewhere safe

BOINC Server VM

- VMWare image available: boinc.berkeley.edu
- Comes with all prerequisites
- See Hands-on

- Outlook: Amazon Computing Cloud VM

Installation

- Run the make_project project script.
  - --project_root <path>
  - --db_user <database_user>
  - --db_passwd <database_password>
  - --key_dir <key_directory>
  - --url_base <url_base>
  - (--drop_db_first, --delete_prev_inst)
    - Optional, used to clean up previous installs.
  - <short_name> <long_name>
- This creates the DB, directory structure, BOINC-generic daemons and tasks, web-app
- Also some configuration files with sensible defaults

Website and forum

- Start by password protecting the html/ops/ administration interface pages.
  - Either with .htaccess or apache.conf file approach
- Edit html/project/project.inc
  - Change data to fit with your project
- Css file in html/user/ can be customized.
- Edit html/ops/create_forums.php
  - Decide what forums are needed/wanted.
  - Run it.

Final steps

- Add application and workunits
- Write and install your own assimilator and validator
- Start the server
  - bin/start from the project directory.

Administration

- The admin web pages
  - Grants access to a load of statistics and status information
    - Users
    - Hosts
    - Applications
    - WU and results
    - more
- BOINC logs
- Custom logs
- BOINC DB for comprehensive state information and manipulation

BOINC resources

- BOINC website: http://boinc.berkeley.edu/
  - Source code
  - Documentation
  - Forums
- BOINC email lists:
  - boinc_projects for project admins
  - boinc_dev for boinc developers
- http://wiki.aims.ac.za/mediawiki/index.php/AIMS_workshop_on_Volunteer_Computing
- Project forums
- nicolas.maire@unibas.ch