Hall B style database

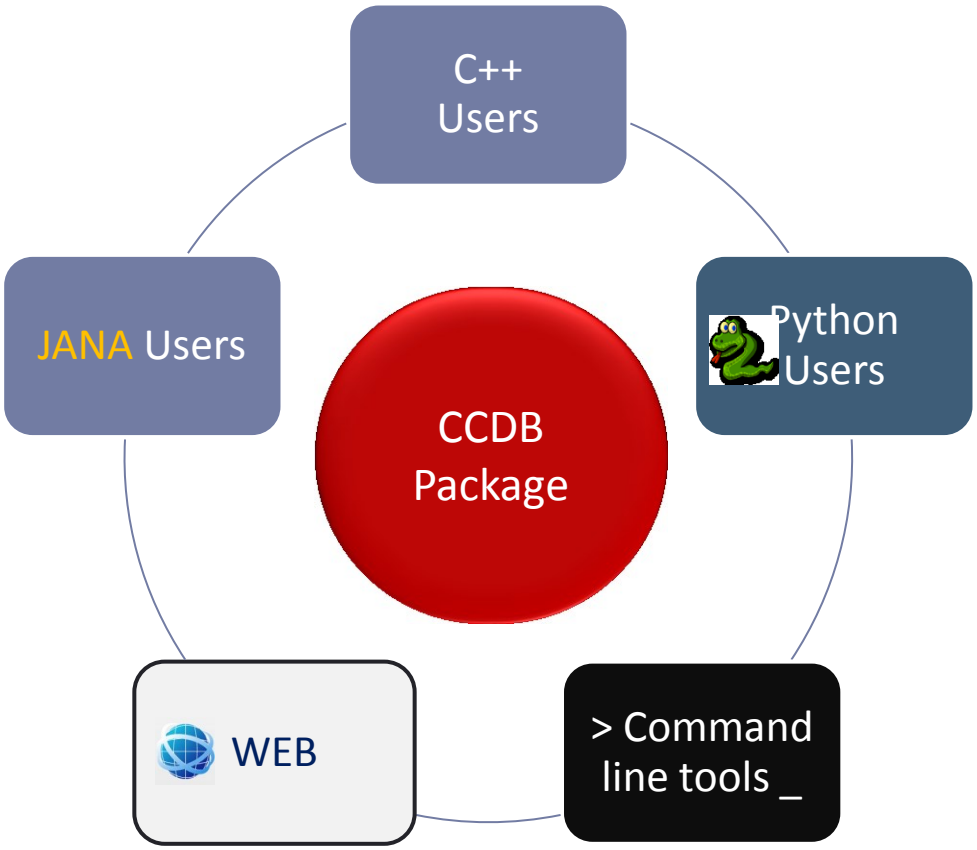
- Never delete always update
- Access database only through API
- Used during for DVCS
  - Root interface
  - Issue of speed of database at beginning of runs
  - Issue running off site
    - Seems addressed with SQL Lite
- Presentation of CCDB, in the doc directory of the CCDB distribution

Calibration database

Dmitry Romanov

February 2, 2011

CCDB package overview



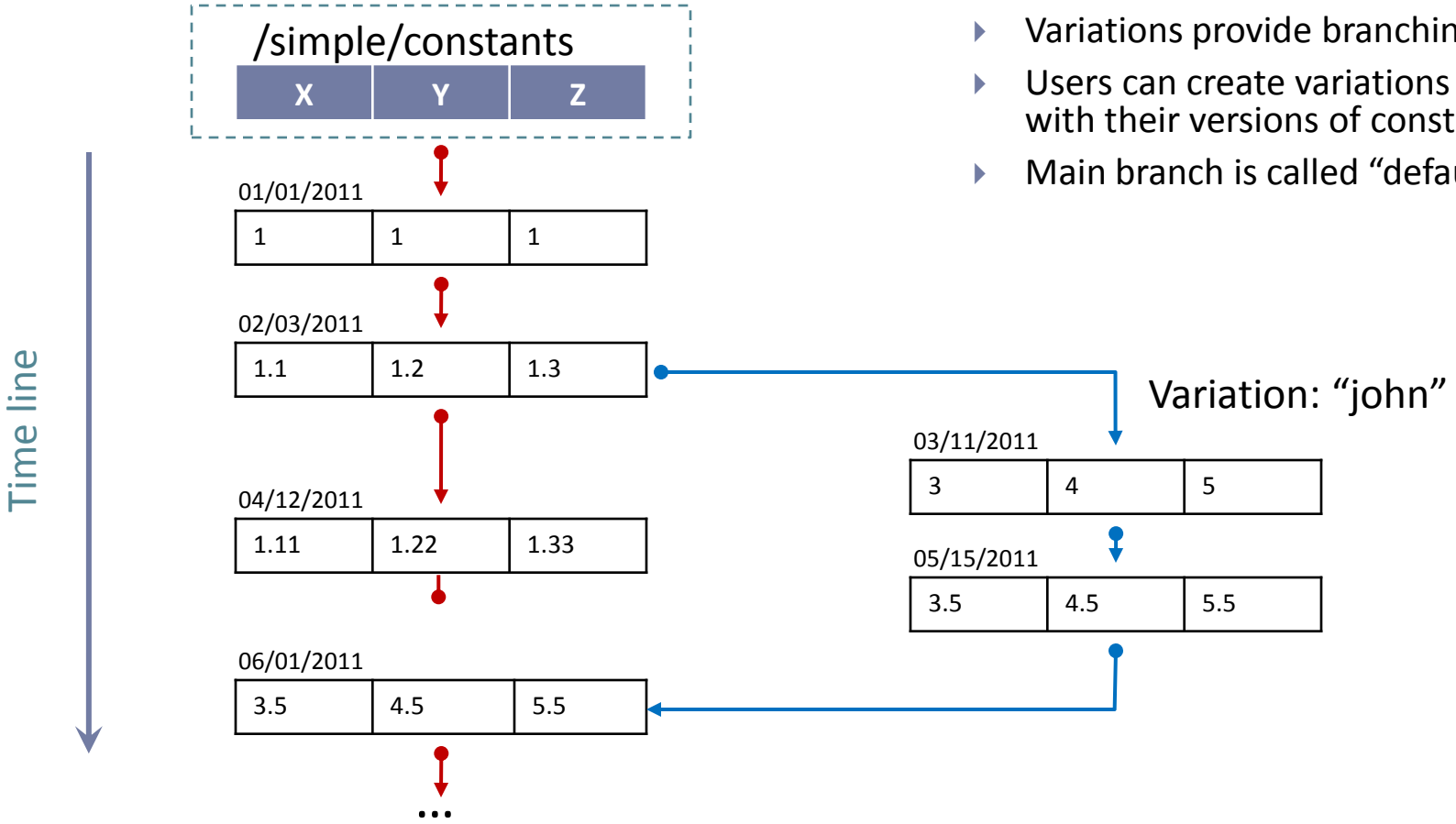
- Versioning and branching
- Full feature storage is DB (MySQL).
- Interfaces JANA, plain C++, Python (and other)
- Web interface and command line tools to manage the data.
- Tools to browse, import, export data
- **Simple for user**

Basic concepts

Column 1	Column 2	Column 3	Column 4	Column 5
double 11	double 12	double 13	int 14	double 15
double 21	double 22	double 23	int 24	double 25
double 31	double 32	double 33	int 34	double 35

- Calibration constants as tables
- Columns might have different types
- Name paths. I.e. /FDC/pedestals
- Data have versions and branches
- “Data Update” by adding new version
- No Delete

Deep into variations

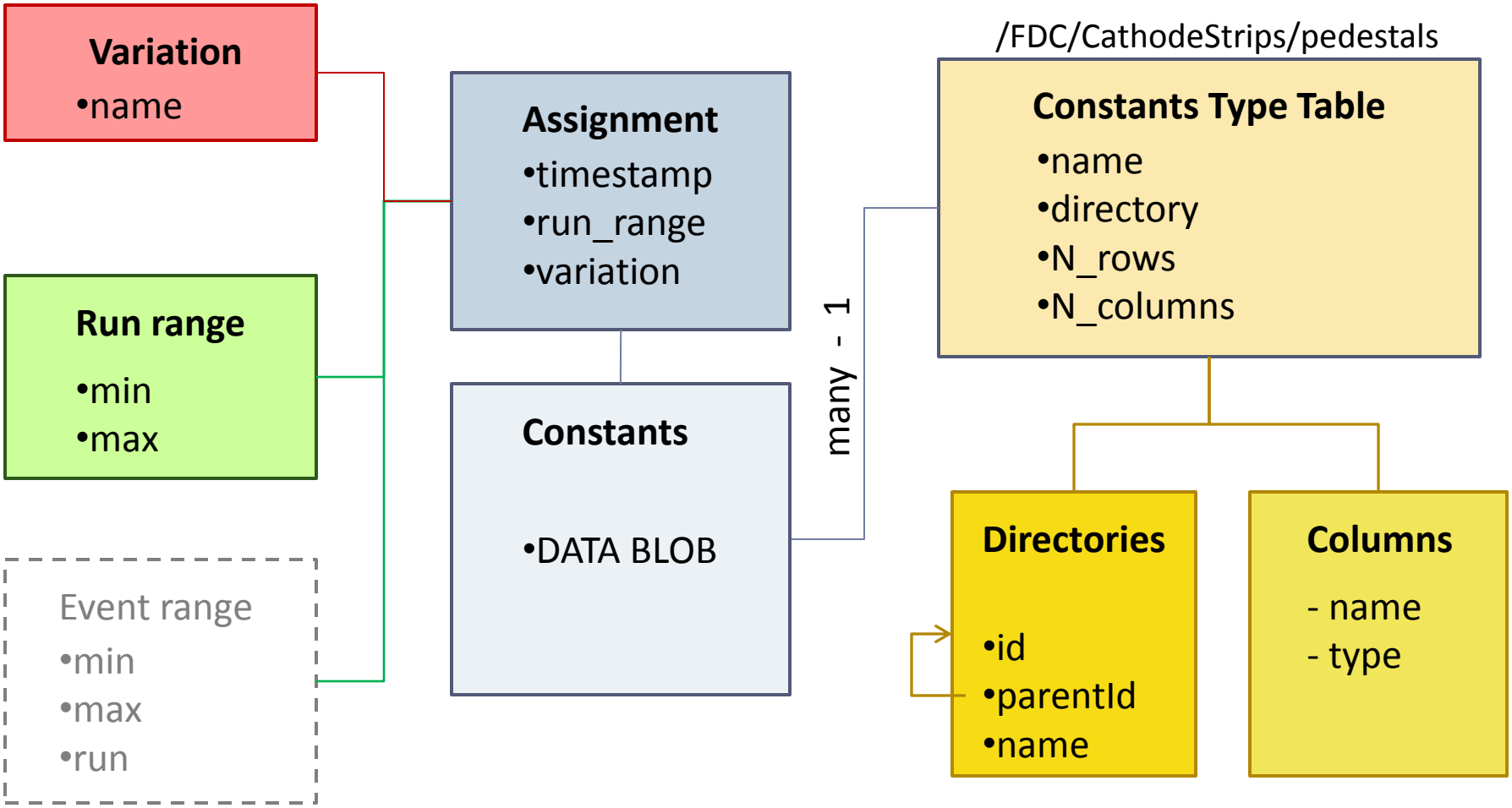


- Variations provide branching
- Users can create variations to work with their versions of constants
- Main branch is called “default” variation

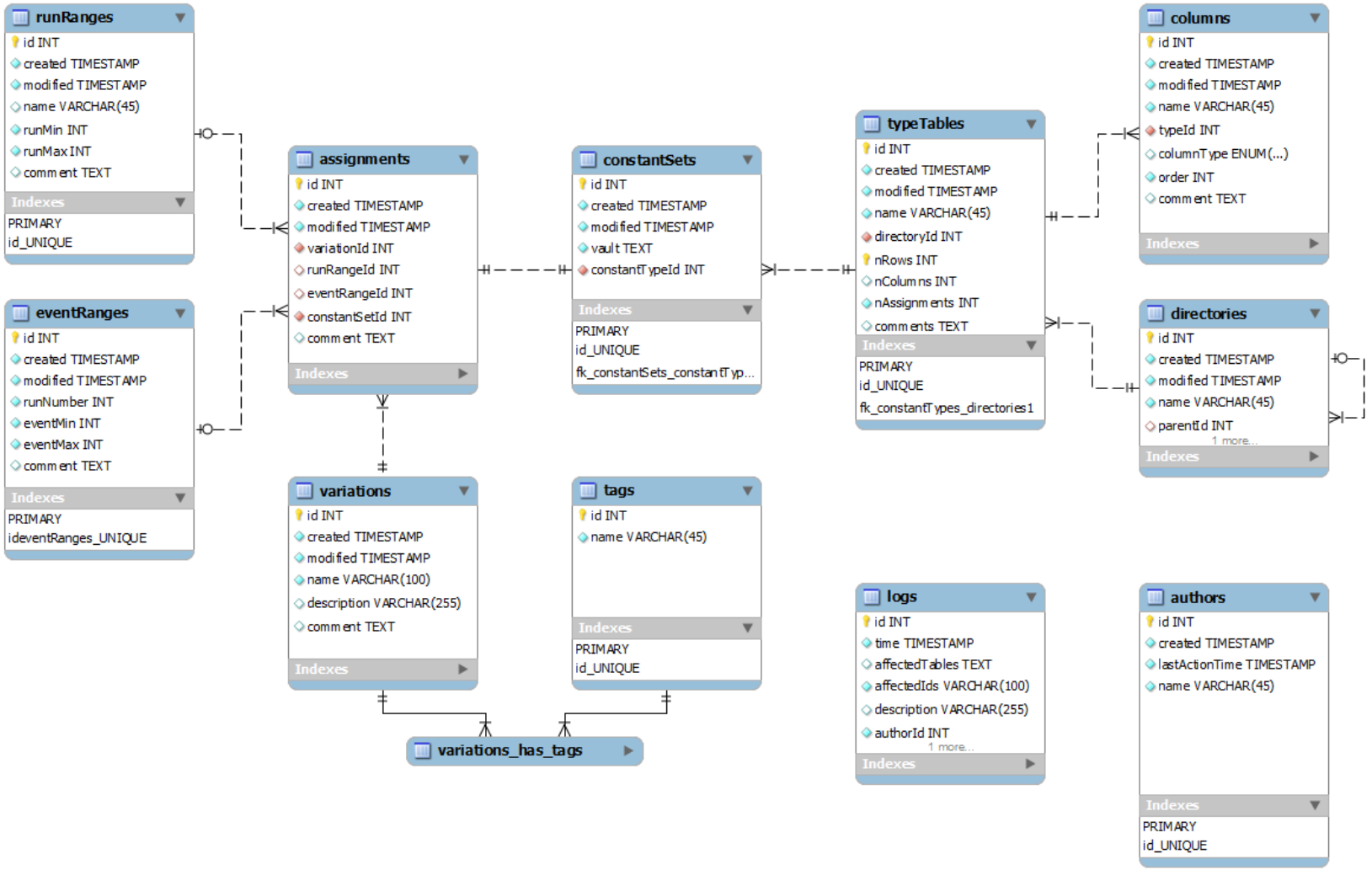
User enters: /simple/constants

User gets: 3.5 ; 4.5 ; 5.5;

Database layout structure



MySQL database layout



C++ and JANA interface

Two main parts:

1) **JCalibrationGenerator** A generator class (e.g. JCalibrationGeneratorMySQL)

```
const char* Description(void);

double CheckOpenable(std::string url, int run, std::string context);

JCalibration* MakeJCalibration(std::string url, int run, std::string context);
```

2) **Jcalibration** class implementing the backend itself.

- Retrieve calibration constants from CCDB (GetCalib())
- Discovery of available constants (GetListOfNamepaths()) etc.

```
Jcallibration * calib = jApp->GetCalib()
vector<double> constants;
calib->GetCalib(constants, "/simple/constants");
 calib->GetCalib(constants, "/simple/constants", /*variation */ "john");
 calib->GetCalib(constants, "/simple/constants", /*variation */ "john", /*time*/ time);
```

Code example

Initialization.

```
//… Somewhere in “application” class
DCalibration *mCalibration = DCalibrationGenerator->GetCalibration(
     “mysql://ccdb_user@halld1.jlab.org”, /*run number*/ 1000, /*variation*/“default”);
//…
```

Getting the data.

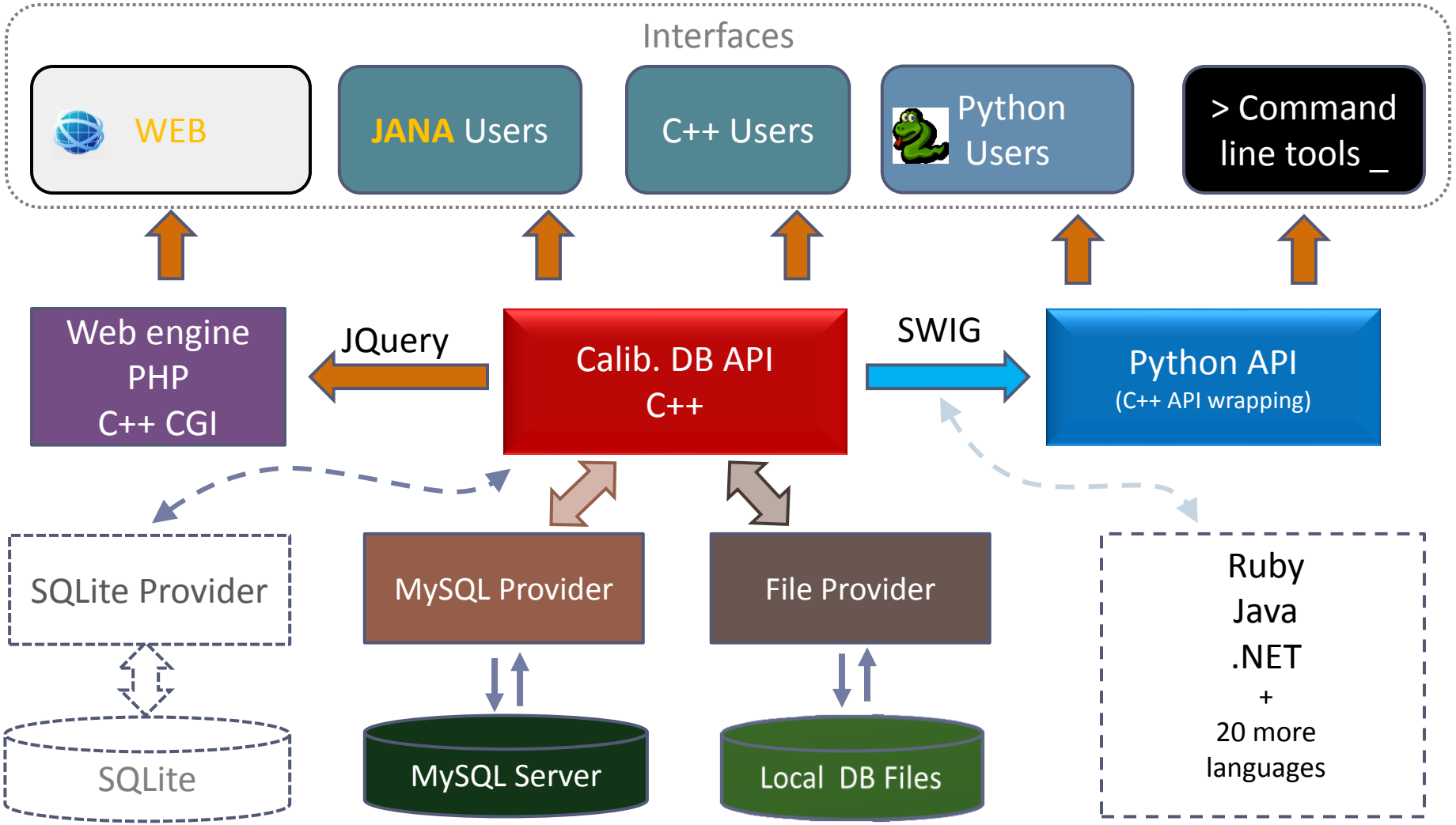
```
DCallibration * calib = application->GetCalib(); /* returns mCalibration */


//getting double constants
vector<double> constants;
calib->GetCalib(constants, “/simple/constants”);
calib->GetCalib(constants, “/simple/constants”, /*variation */ “john”);
calib->GetCalib(constants, “/simple/constants”, /*variation */ “john”, /*time*/ time);
```

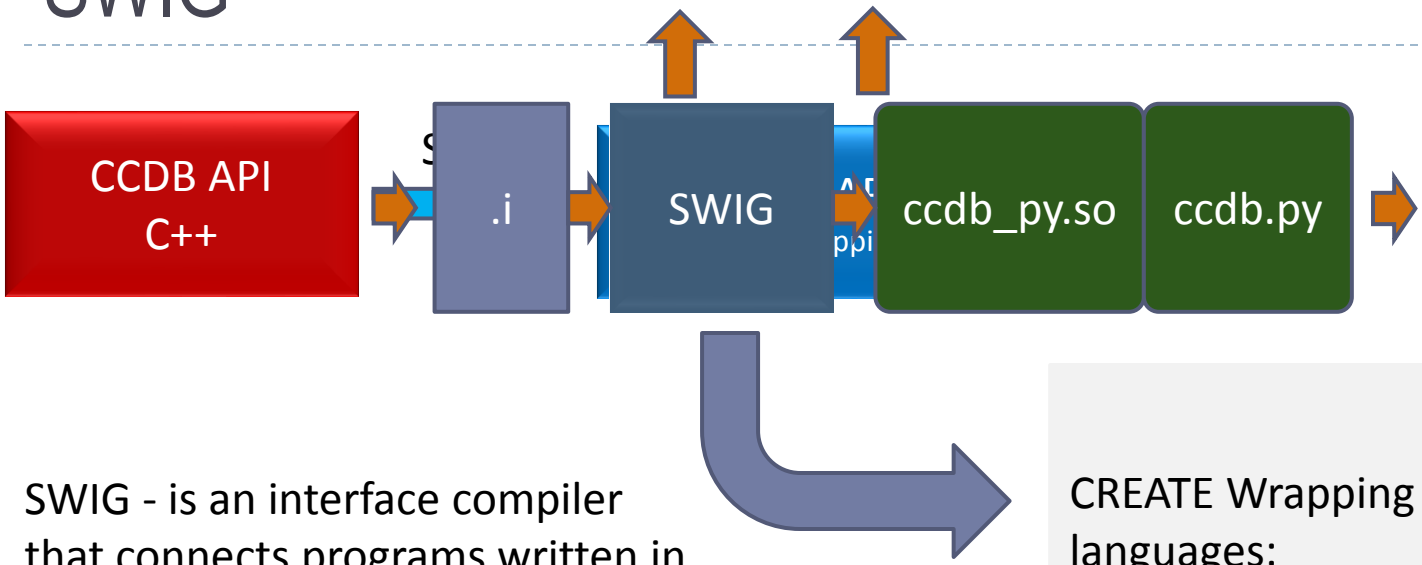
Multi-row and multi-type data.

```
//getting constants by rows and column names
map<string, double> constants;
//getting multi type constants
vector<vector<double> > constants;
calib->GetCalib(constants, “/simple/multi_rows_constants”);
```

Overall package design



SWIG



SWIG - is an interface compiler that connects programs written in C and C++ with scripting languages

How to use SWIG?

- Make SWIG interface files .h
  (it is headers without private members)
- Run SWIG
- Get C++ wrapping source files, compale it.
- Get library
- Get library in target language

CREATE Wrapping for any of languages:

Allegro, CL, C#, CFFI, CLISP, Chicken, Go, Guile, Java, Lua, Modula-3, Mzscheme, OCAML, Octave, Perl, PHP, Python, R, Ruby, Tcl, UFFI

CCDB shell and command line tools

Aim: simplicity + better learning curve

Use POSIX like commands: ls, rm, mkdir, mv, … etc. commands

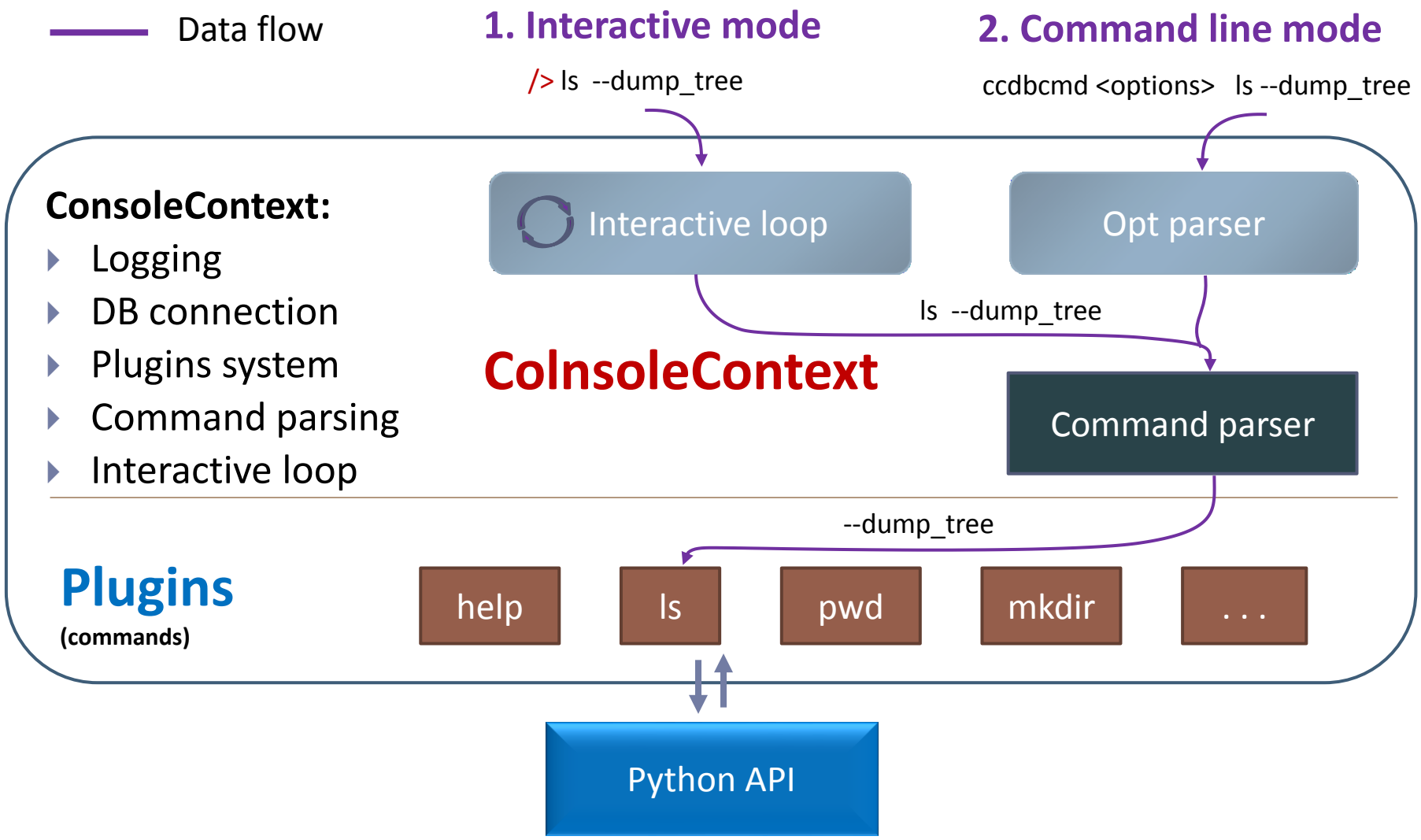
- **Interactive mode**

```
>> ls
adc/    dca/
>> cd adc
adc> > ls
../    adc _pedestals       adc_scales
adc>>  ls   -l
adc _pedestals :    3 variations,   5 versions … <full info>
adc_scales:         4 variations,   11 versions … <full info>
adc>>dump adc _pedestals  --run 1200 --variation mc  > /home/user/pedestals.txt
```

Command line mode

```
ccdbcmd dump /adc/adcpedestals   --run=1200  --variation=mc  > /home/user/pedestals.txt
```

Command line tools implementation



CCDBCMD Live example

```
+-------------------------+
  CCDB shell v.0.0.1
  HallD JLab
+-------------------------+

Interactive mode
print help to get help
print quit or q to exit
print !<command> to execute shell command

/> ls
test

/> cd test/subtest
/test/subtest> ls
some_more_directory
test_table
test_table2

/test/subtest> info test_table
+----------------------------------------+
| Type table information                 |
+----------------------------------------+
 Name       :  test_table
 Full path  :  /test/subtest/test_table
 Rows       :  2
 Columns    :  3
 Created    :  2011-01-27_19-30-50
 Modified   :  2011-01-27_19-30-50
```

Interactive mode

...

```
/test/subtest> dump test_table --run=100 --variation=mc test_table.txt
Saved /test/subtest/test_table to file test_table.txt
/test/subtest> quit
```

```
d:\AProjects\CCDB\ccdb\python>ccdbcmd  dump test_table --run=100 --variation=mc test_table.txt
Saved /test/subtest/test_table to file test_table.txt

d:\AProjects\CCDB\ccdb\python>
```

Command line mode

Web interface approach

Task

- Have some framework to generate pages.
- Connect this framework with C++ CCDB API

Solution

- PHP for page generation.
- CGI from C++ API for accessing data.
- JQuery to glue both and make UI reach.

What is JQuery?

- **jQuery** is a cross-browser JavaScript library designed to simplify the client-side scripting of HTML.
- Released January 2006. Used by over 31% of the 10,000 most visited websites at the end of 2011.
- Works on all todays browser. Moreover, makes JavaScript programming browser independent.
- Makes AJAX – really easy. Enriches web User Interface.

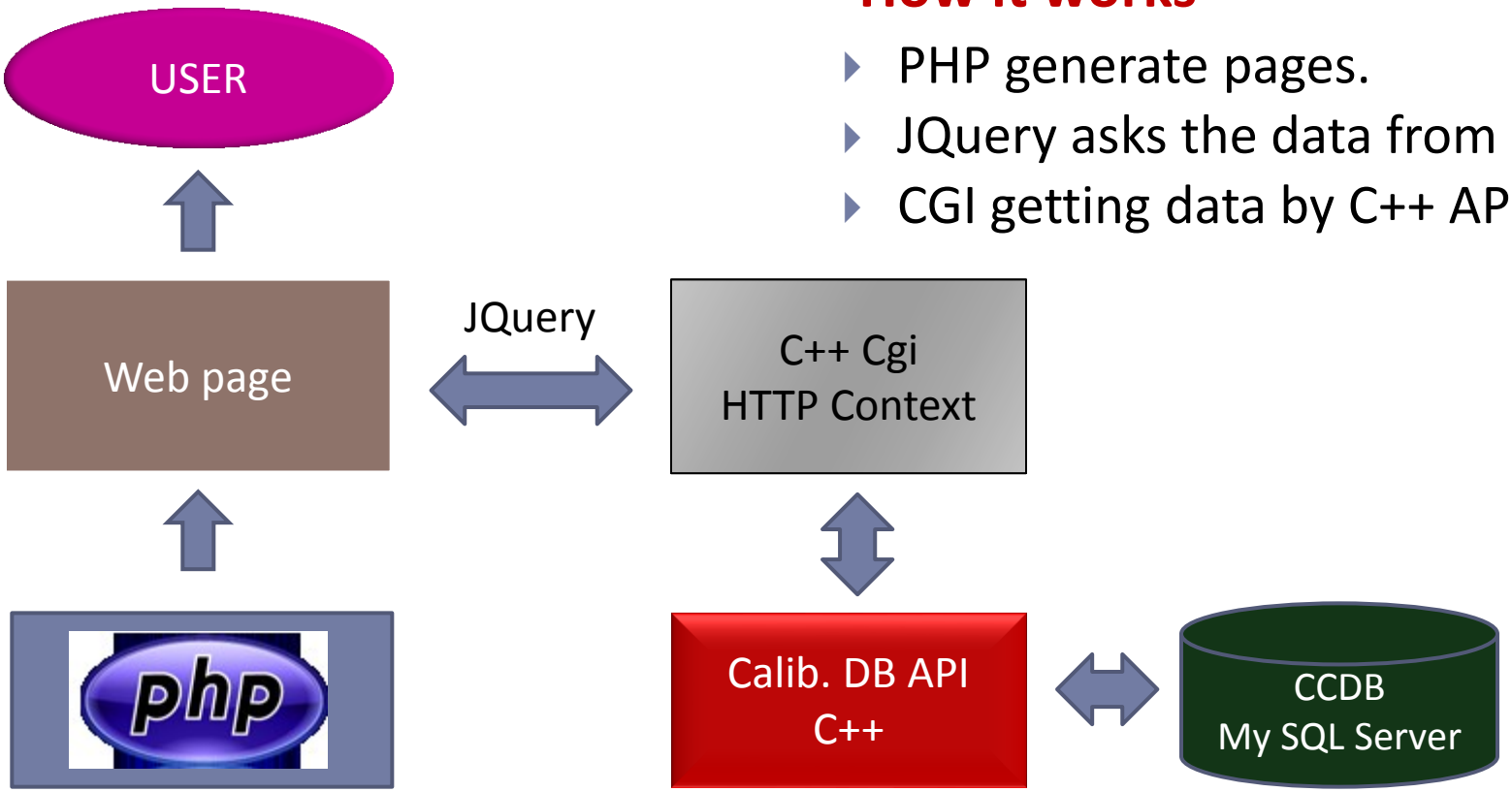
AJAX - Asynchronous JavaScript and XML

- Classic HTML – reload page on **every** user update.
- AJAX – Possibility to reload only part of the page.

Web interface

How it works

- PHP generate pages.
- JQuery asks the data from CGI.
- CGI getting data by C++ API.



- How compex is it?

```
$("#tree_place").treeview({url: “cgi-bin/ccdb.cgi?op=ajaxdirs “})
```

CCDB text files

```
#meta run rage: 100 - 200
#meta variation: default
# This is a simple table with 4 columns and
# 3 rows. Any users comments are here…
#& col1      col2      col3      col4
   value     value     value     value
   value     value     value     value
   value     value     value     value
```

- Text files with tables.
- One file – one table.
- Comments and metadata optional

Using as separate storage

- Files location related to namepaths of the tables
- This means that if a table has a namepath: /simple/constants there is file constants in ($BASE_PATH)/simple/
- All manipulations for import and export of such files

Performance budget

- The values are grouped in sets. 1 to 10k values per a single set.
- Access to a single number - less than 10 ms using a 1 Gigabit connection.
- Retrieval of 10k values should - less than 1 second under similar conditions.

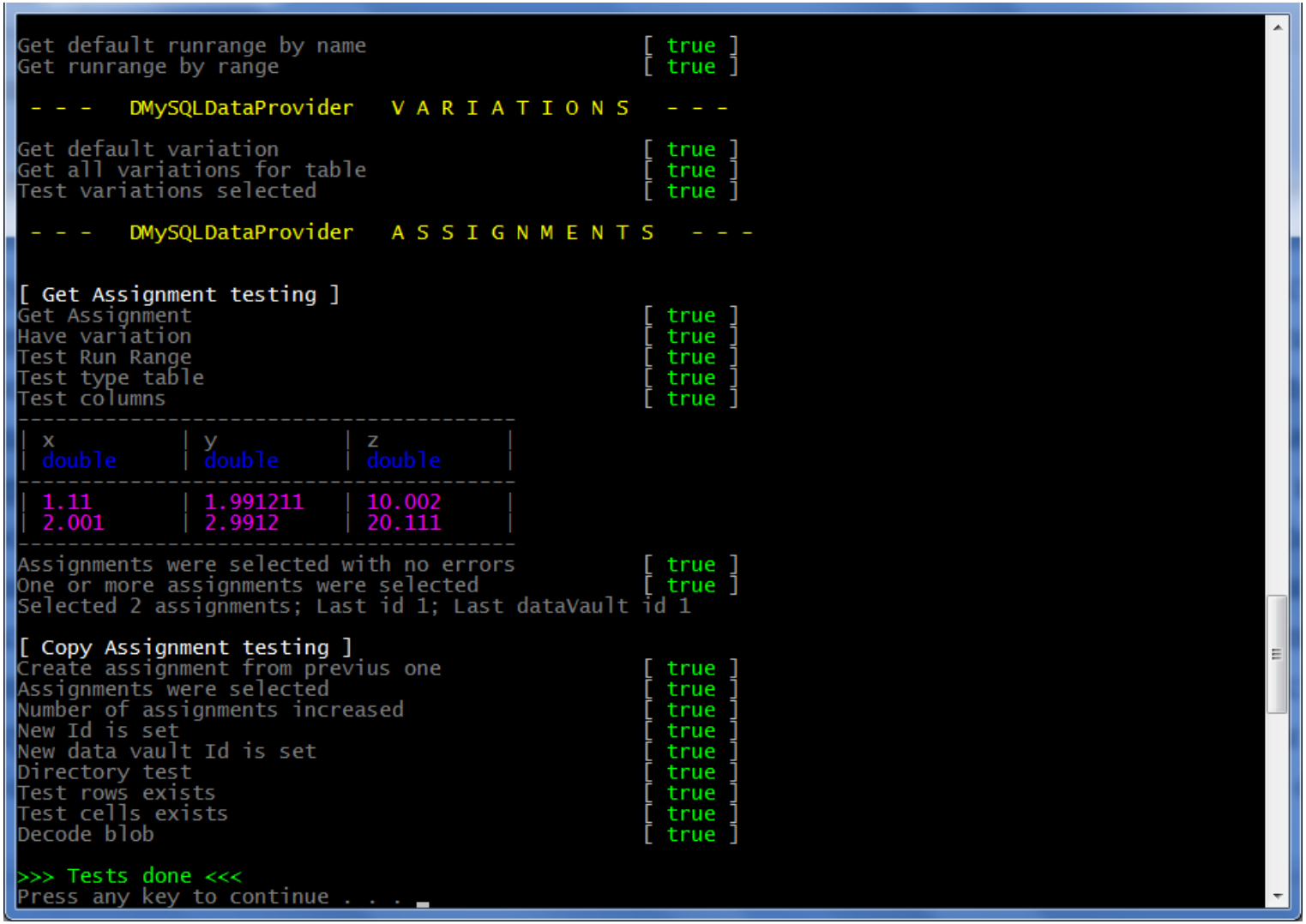
1 100 000 records with data blobs; 50 double doubles (8 digits) for each blob.
Queried randomly with 1100 requests.
Core 2 Duo 1800 GHz, 2 Gb – 0.23 sec average
Core i7 4 cores 2800 GHz, 8 Gb – 0.04 sec average

Stick to 1 second to load calibration constants

- Data queries + network – 0.23 – 0.45 sec
- Framework overhead ~ 4%

Conclusion: possible to keep in 1 second.

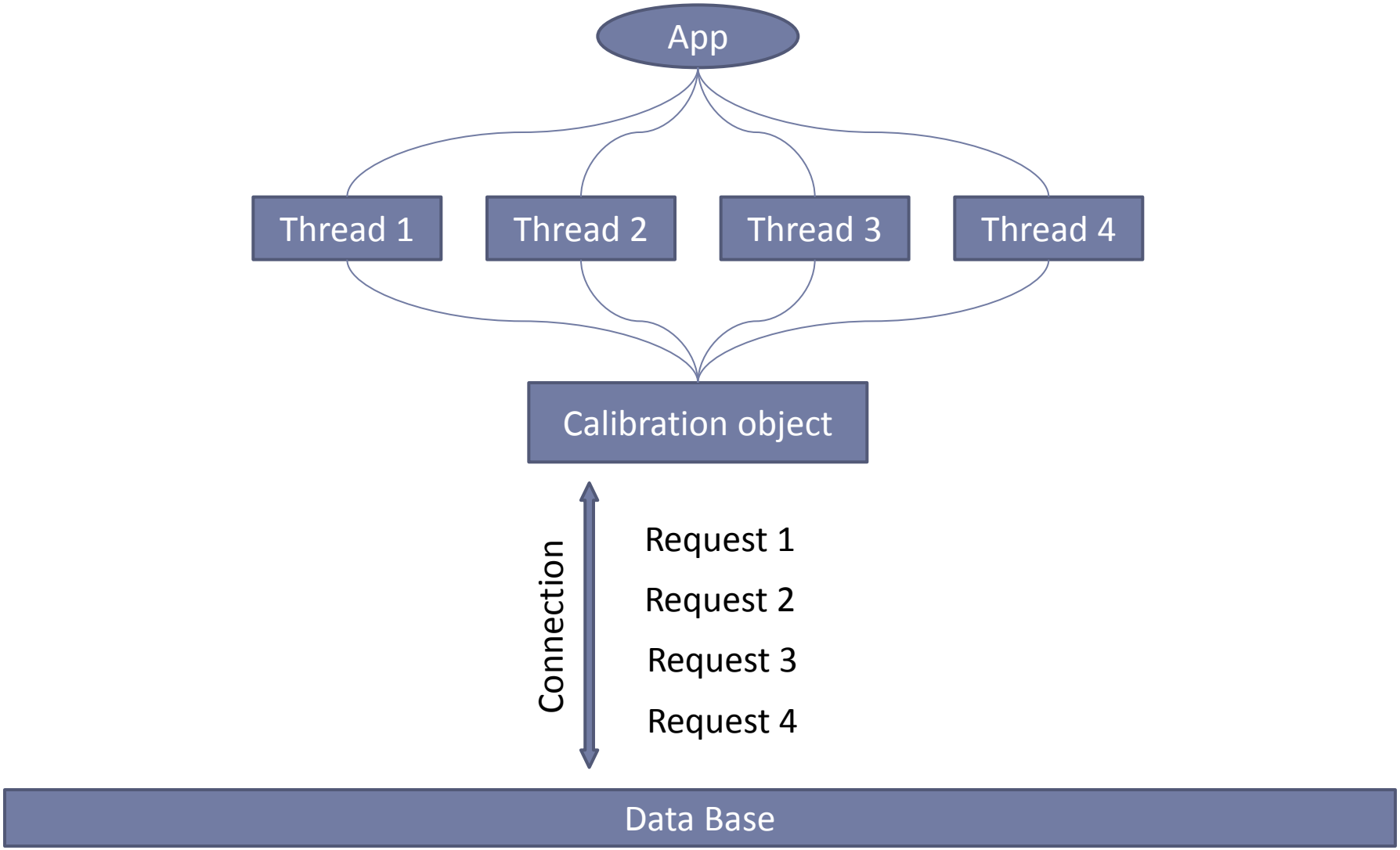
Unit testing



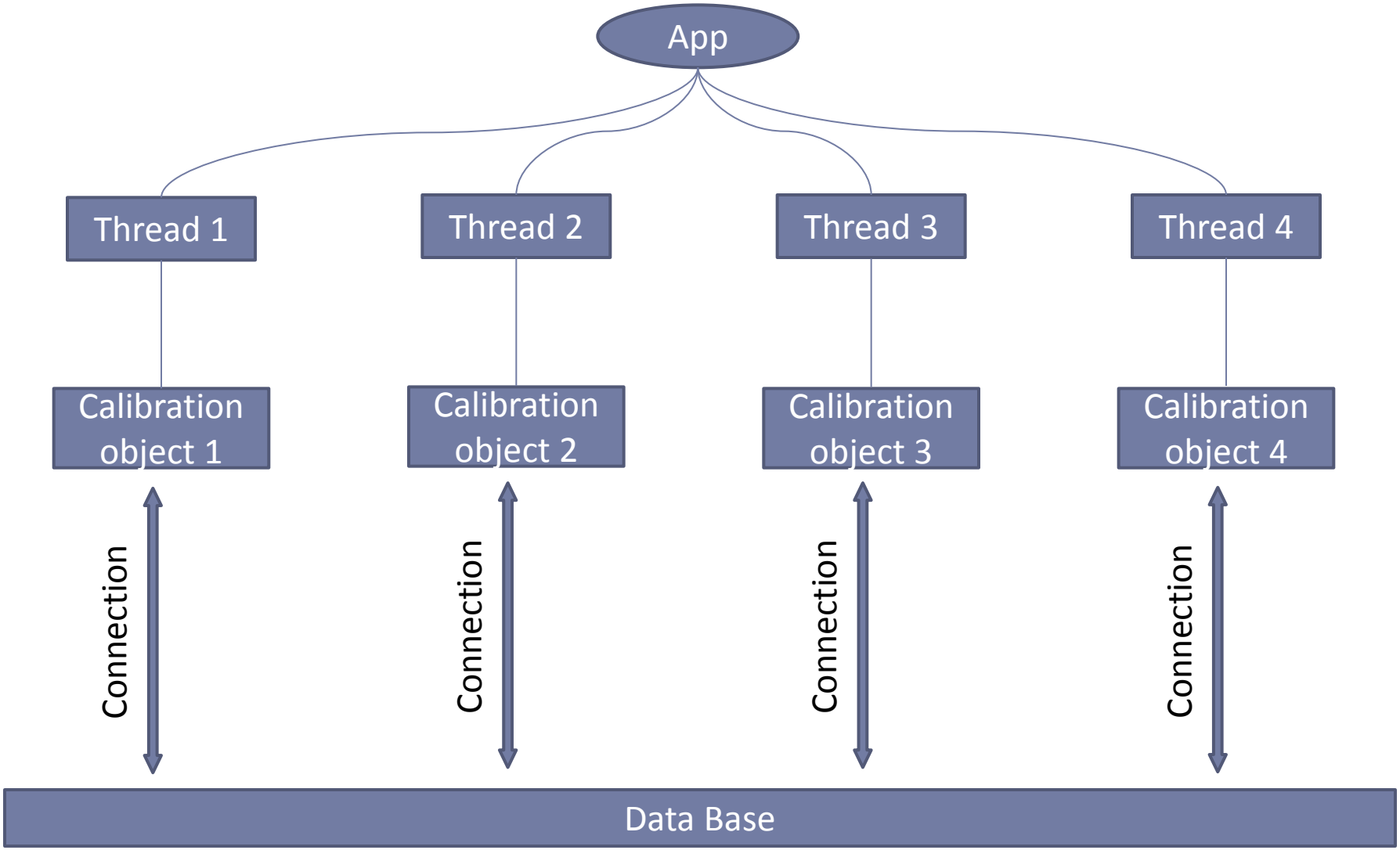
Multi Threading

- C++ User API is designed
- CCDB is thread safe.

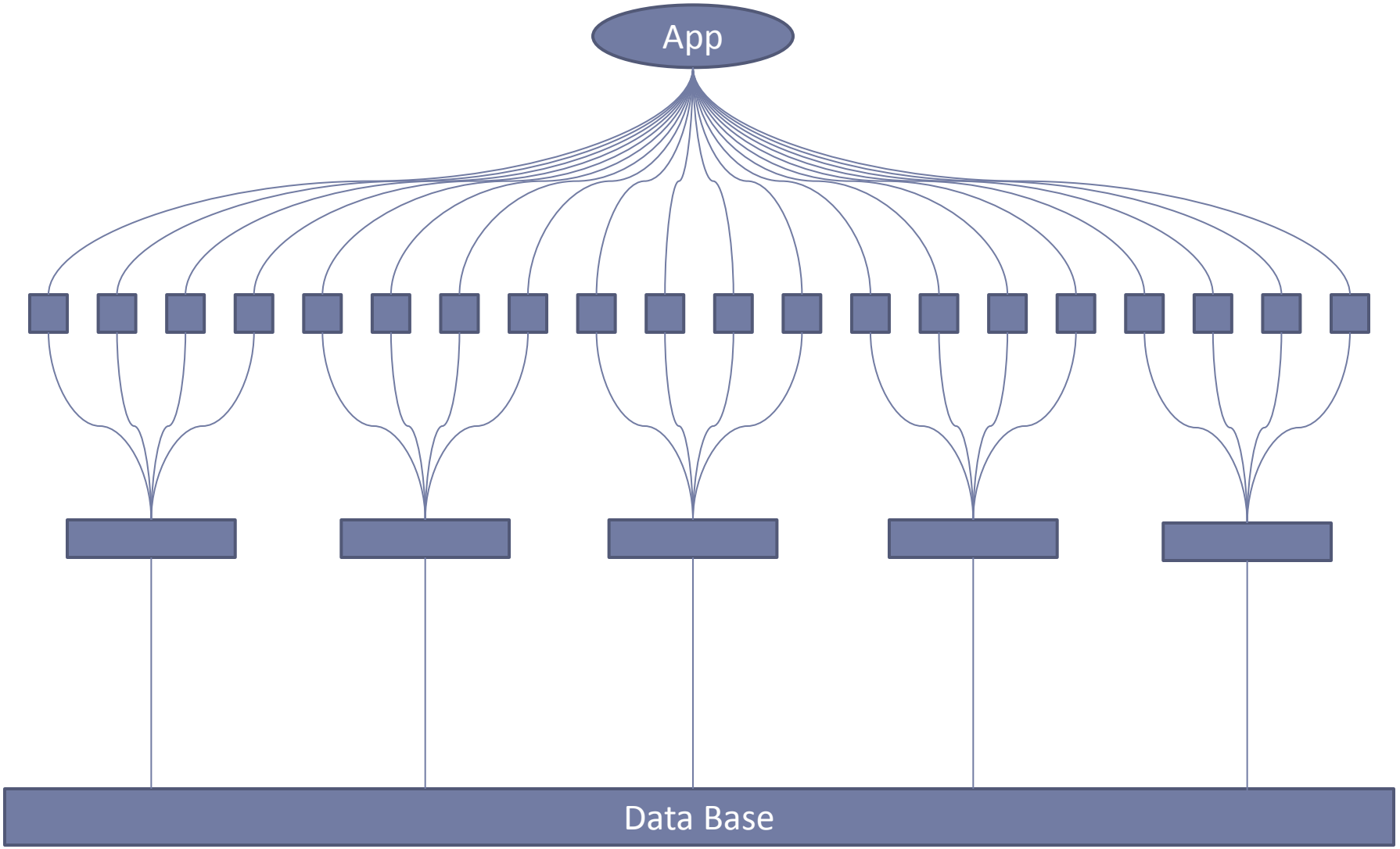
Simple synchronization



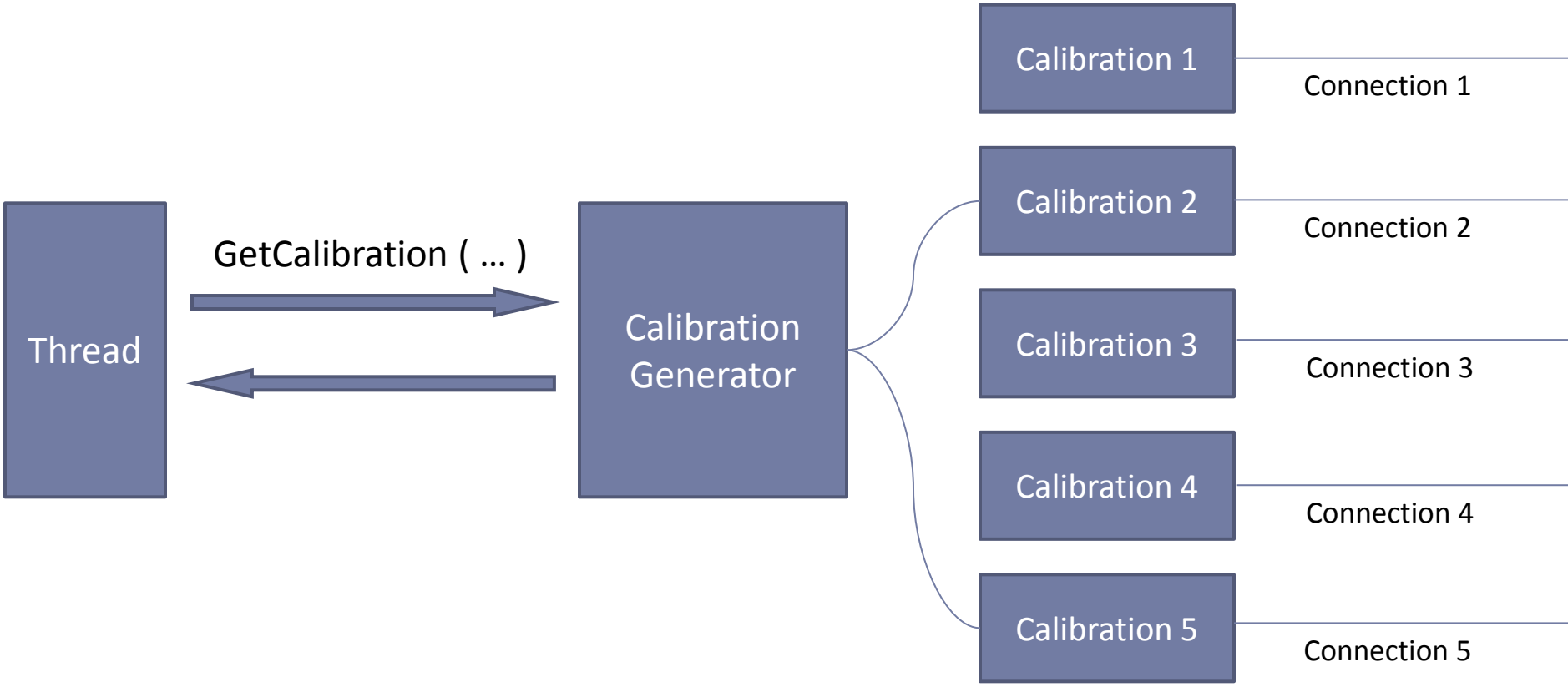
Simple synchronization 2



Synchronization



Getting threads



Performance

Dependencies summary

Concise overview

- C++ API the only dependency is libmysql
- Python wrapping is done by SWIG. No SWIG dependencies are needed for users.
- Console tools are made by python 2.6+. No additional dependencies.
- Web interface consists of two parts PHP and C++ CGI.
- Web pages are driven by JQuery

Technologies overview

- ✓ C++, STL
- ✓ C API for MySQL
- ✓ Python
- ✓ SWIG
- ✓ PHP
- ✓ JQuery

Conclusion

- CCDB Package provides interfaces to read and manage calibration constants for:
- JANA Users, plain C++, Python, command line, web.
- Full featured storage is relational database (MySQL). Other storage mechanisms are possible.
- Versioning and branching for data available.
- Core API is in C++.
- Modular structure allows to extend package to use other storing sources. The only dependency for C++ is libmysql.
- Python wrapping is done by SWIG. That allow to have API for 19 other languages.