Geant4 collaboration meeting - 23/09

A generic system for generating, storing and analyzing validation results

George Lestaris (PH-SFT, CERN)

Supervised by Witold Pokorski and Alberto Ribon

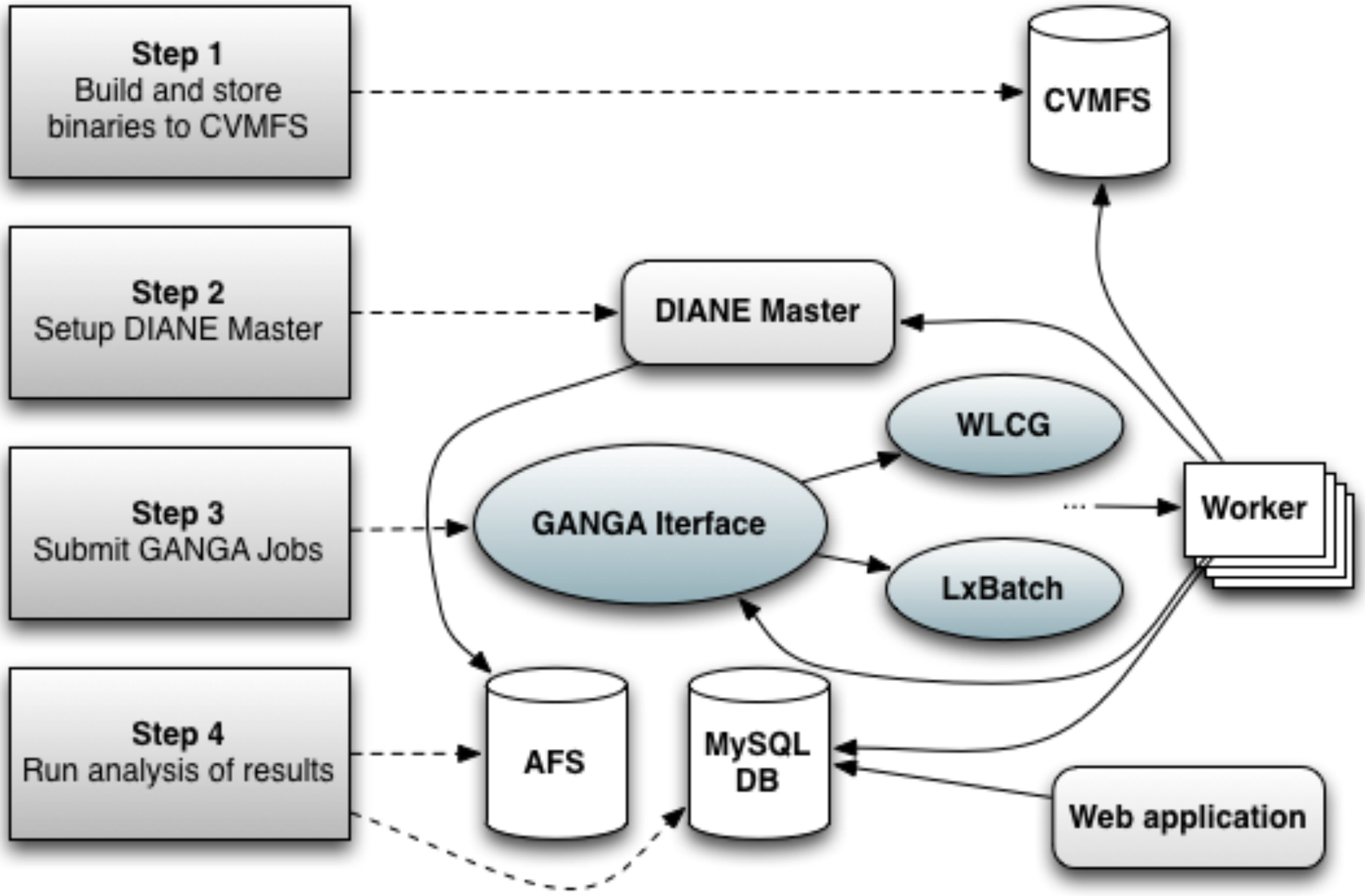
Outline

- **Motivation**
- **Architecture:** production, storage, analysis and presentation
- **Use cases**
  - Simplified calorimeter
  - Geant4 test `test30`
- **Conclusion**

Motivation

- **A system for regression testing**
  - Physics validation: statistical tests / human looking at the results
- that can be used with **Geant4 applications, tests and examples**
- that is **flexible and well tested** to reduce *maintenance cost* and *development needs*
- and that can utilize painless **heterogenous computing resources**

Existing infrastructure



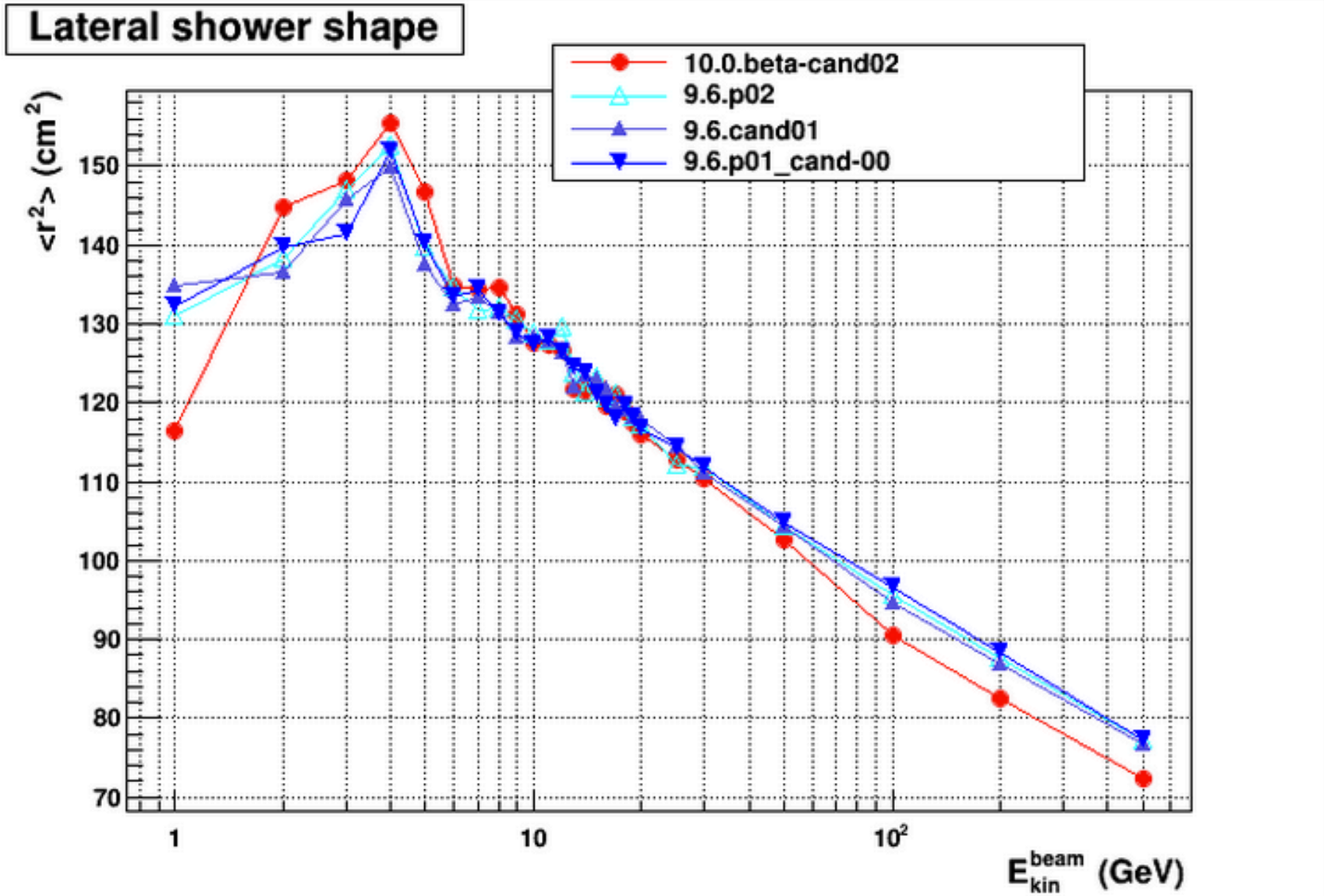
Existing infrastructure

- Based on the SimplifiedCalo application
- Uses DIANE which is not formally supported
  - and GANGA
- Not easy to deploy a validation run
  - four manual steps
- Monitoring interfaces are not providing useful information
  - when something breaks, hacky tricks are used to understand why...

Requirements for S.C.

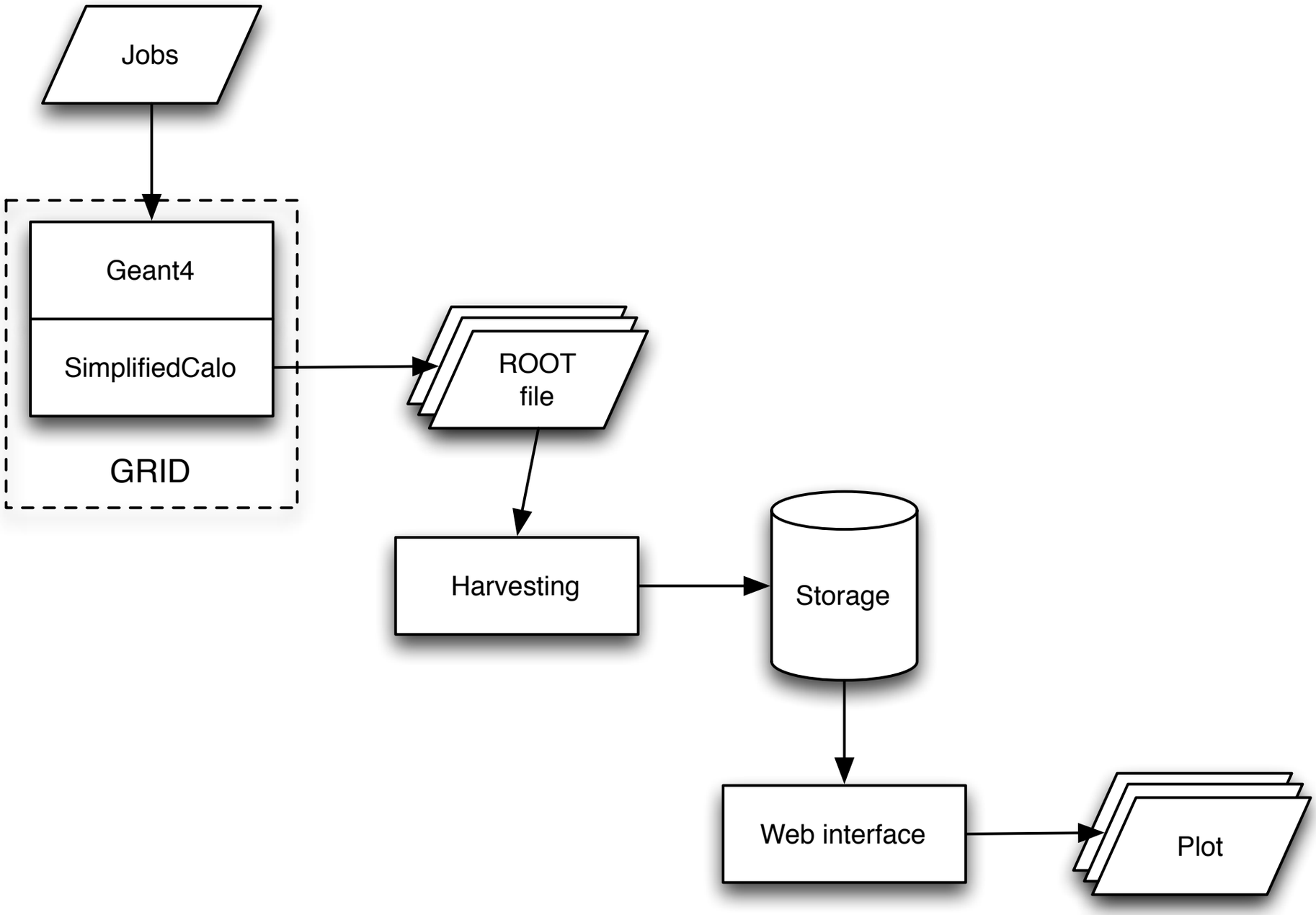
- The SimplifiedCalorimeter though remains our first and most important use case
- Running for each internal/beta/public release ~ 4K jobs on the GRID
- Jobs are Calorimeter-Beam type-Energy-Physics list combinations
- Merging of results - storage - presentation with an interface that allows for dynamic creation of comparison graphs...

...like this

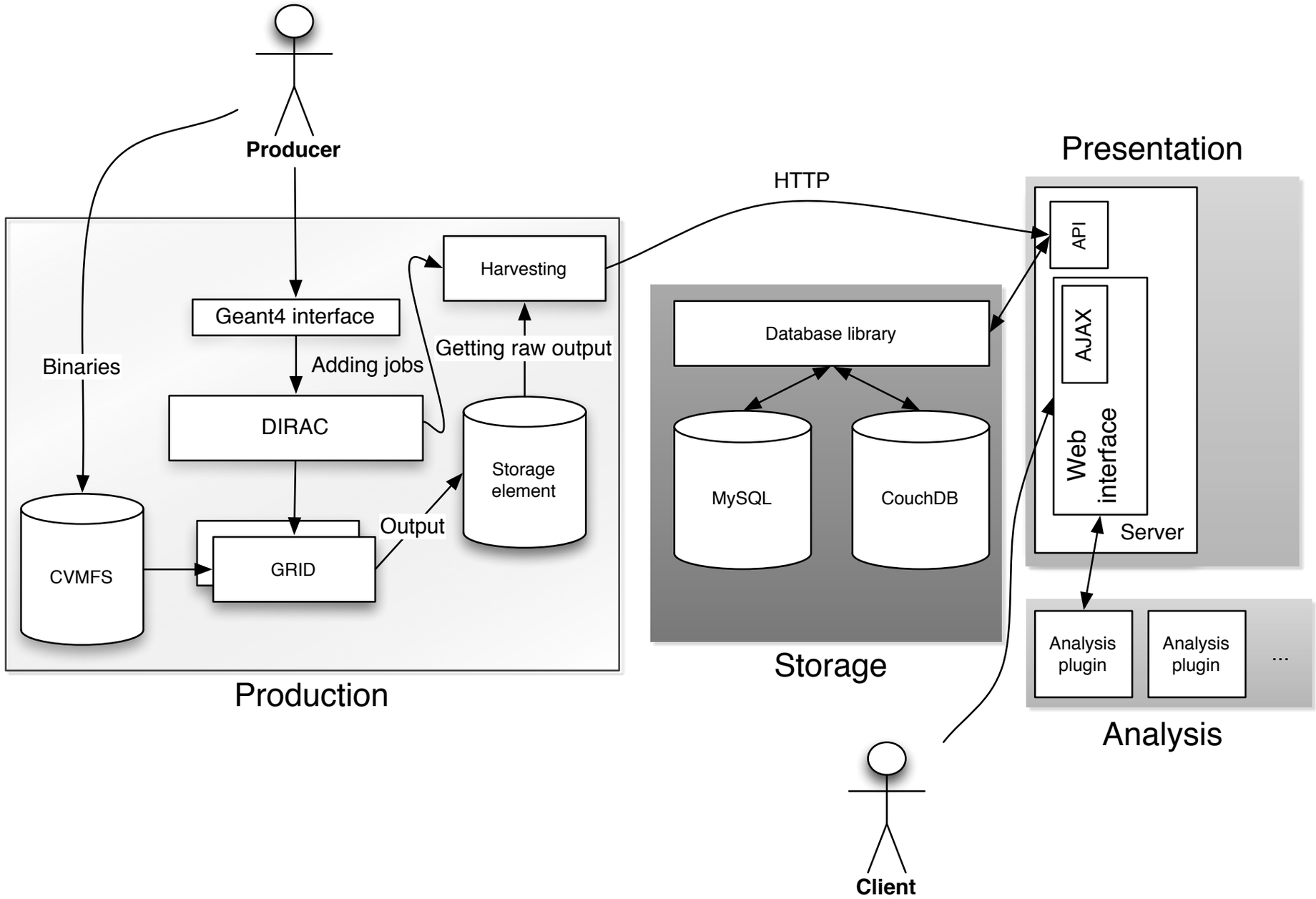


Calorimeter: AtlasHEC | **Beam particle**: pi- | **Physics list**: FTFP_BERT | **Observable**: Lateral shower shape

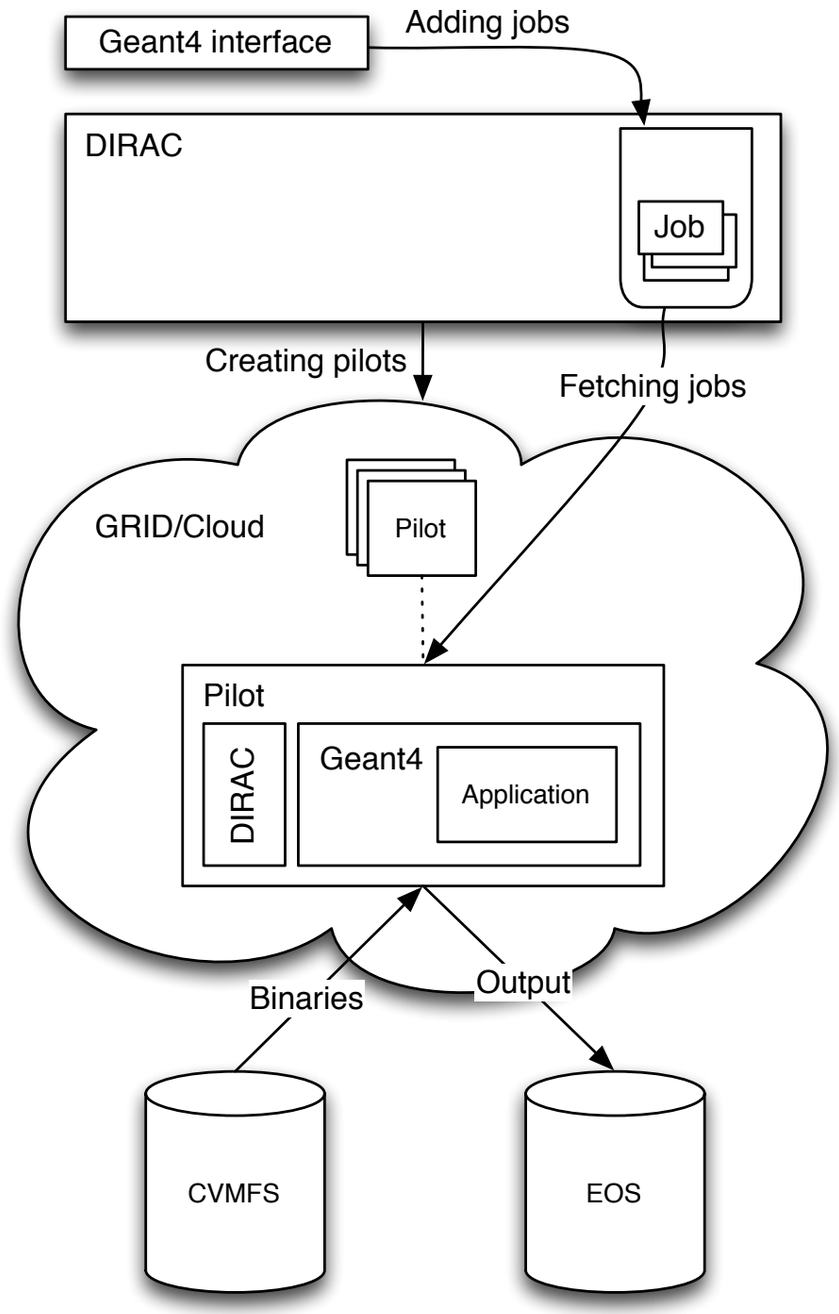
Visualizing the problem



Architecture



Production of results

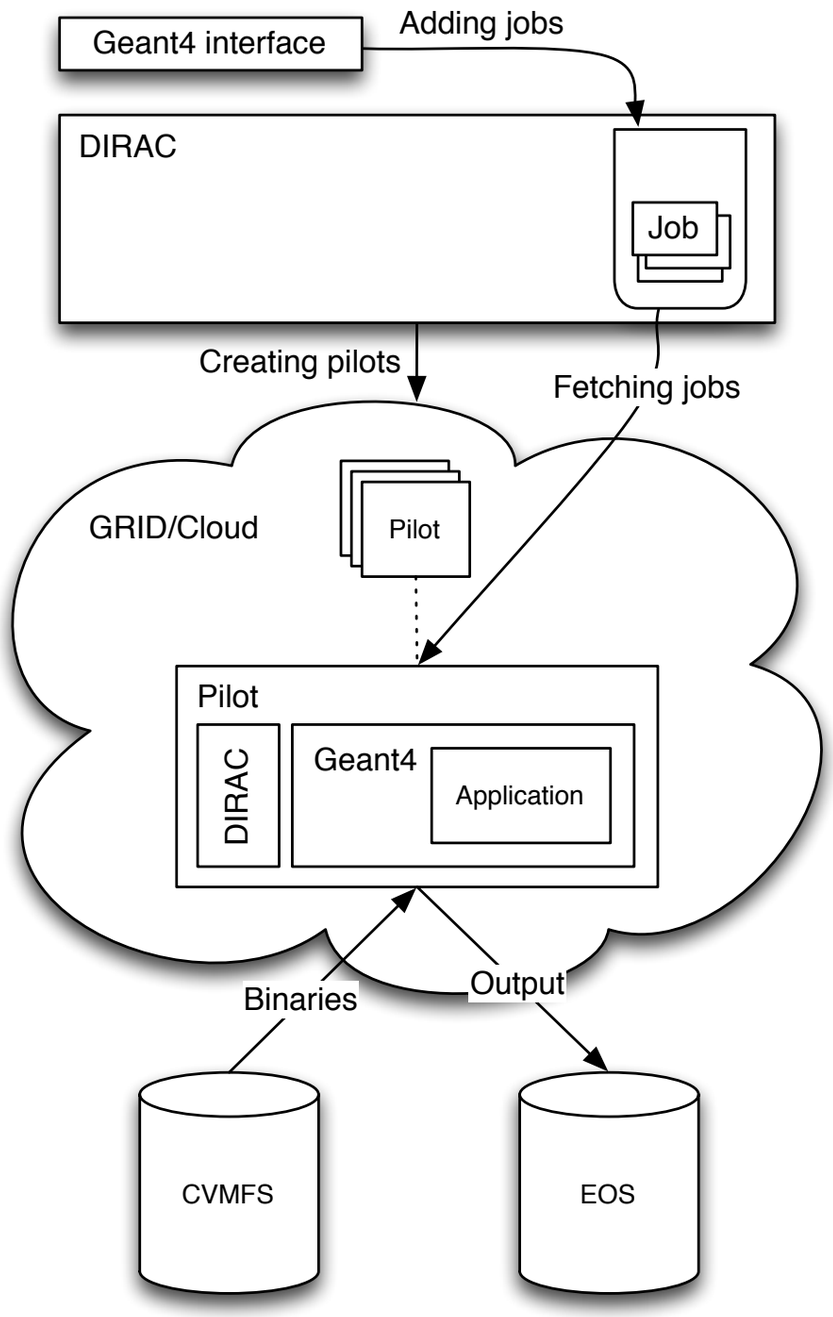


CernVM-FS for distributing the binaries and libraries

DIRAC for steering the process

- Centralized queue, pilot agents model
- Can use GRID, cloud, local queues

Production of results



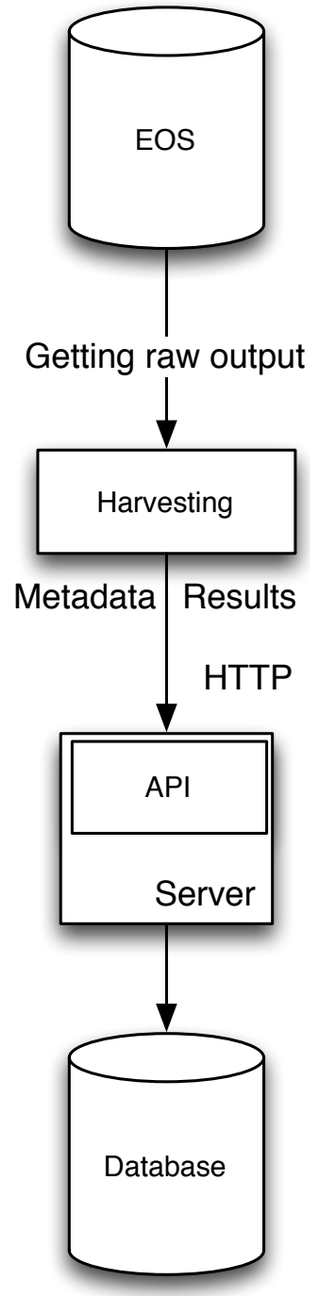
Thin layer to act as a **Geant4 specific interface**

- **define executables**
  - input parameters, macro generation, ...
  - environment (libraries, platforms, ... etc)
- **define runs**

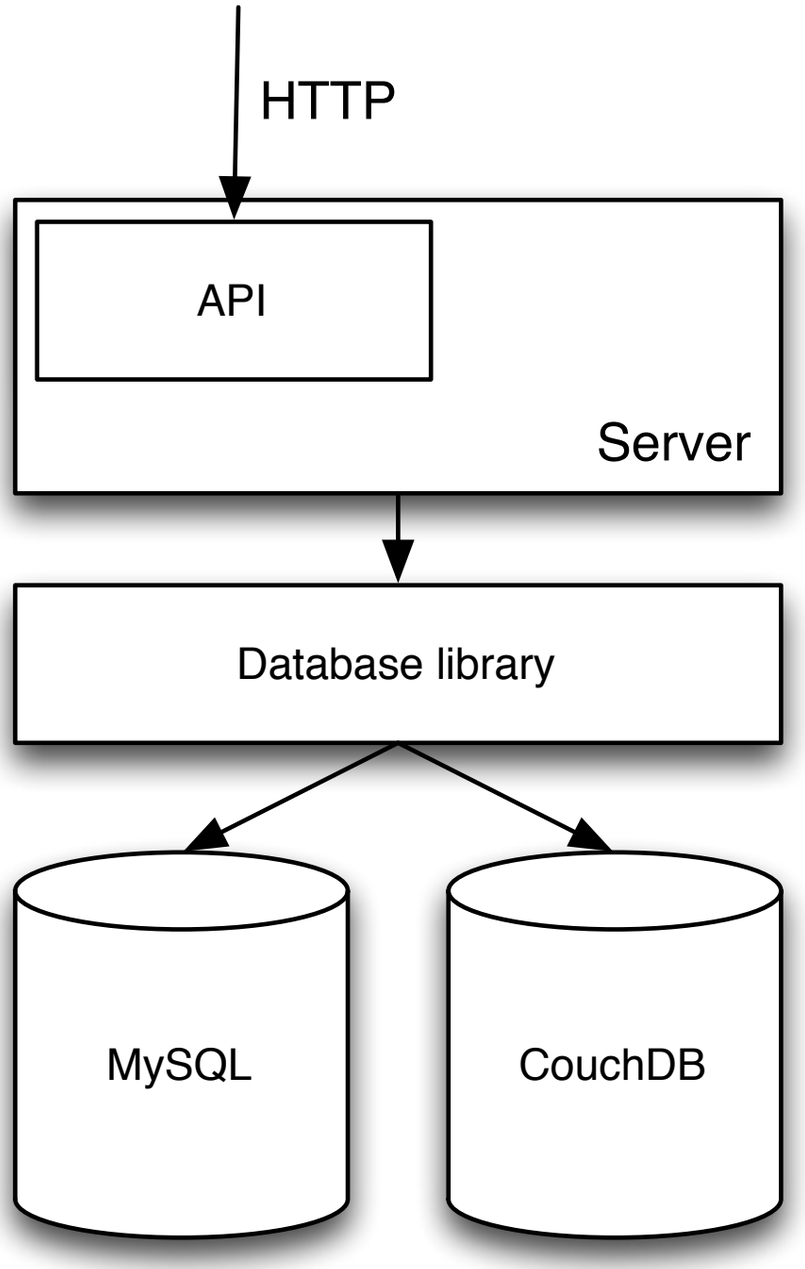
EOS Storage element to store the raw output (ROOT files)

Harvesting results

- will be called automatically by the **DIRAC transformation system**, when all the jobs are done.
- making fits and statistical analysis where required
- storing only what is important to the **database**
  - using the Validation server API

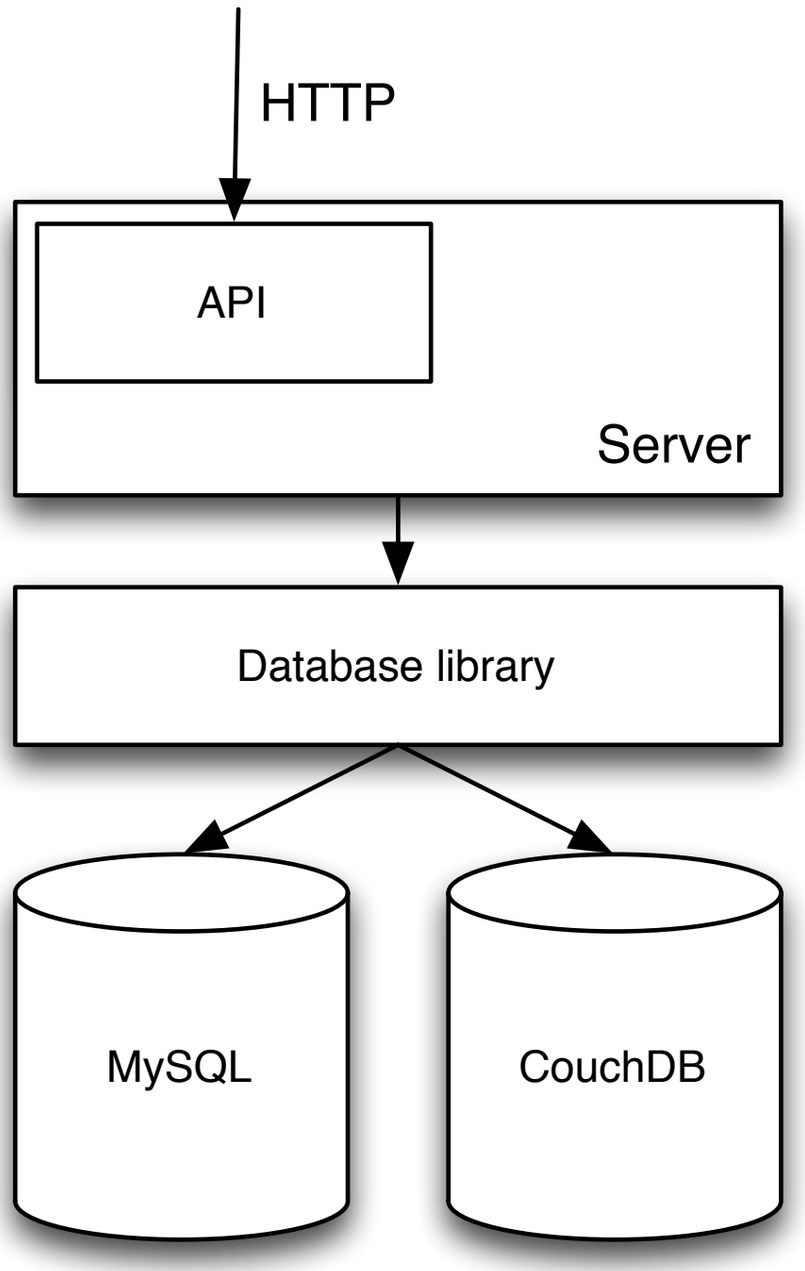


Database



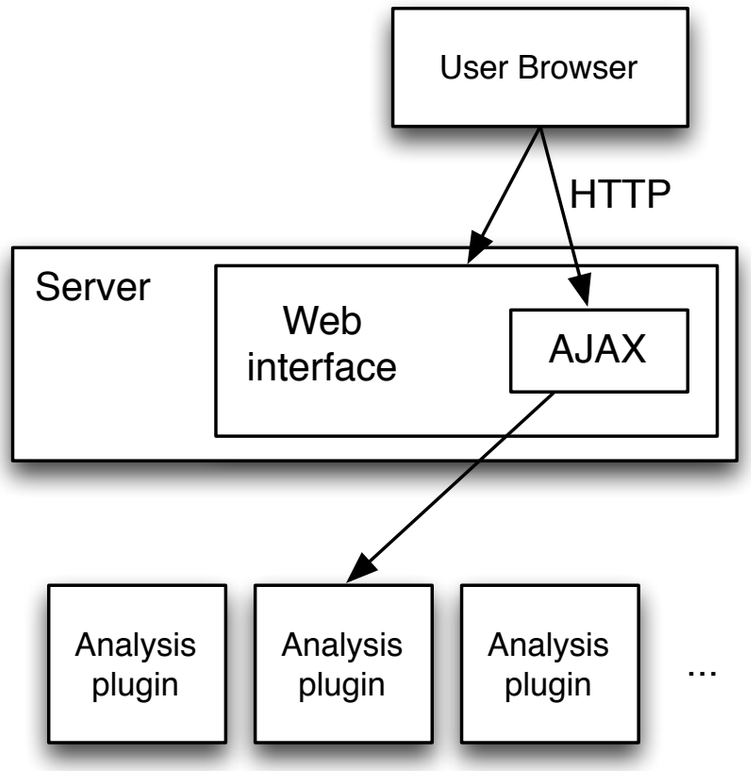
- **Two backends**
  - **Relational:** as a catalog
    - optimized schema
    - references to objects stored in CouchDB
  - **Non relational:** for binary data

Database



- **Validation server** (middleware) provides **API** to add/remove/get runs and results
- Python library **encapsulates** the low level functionality and synchronization

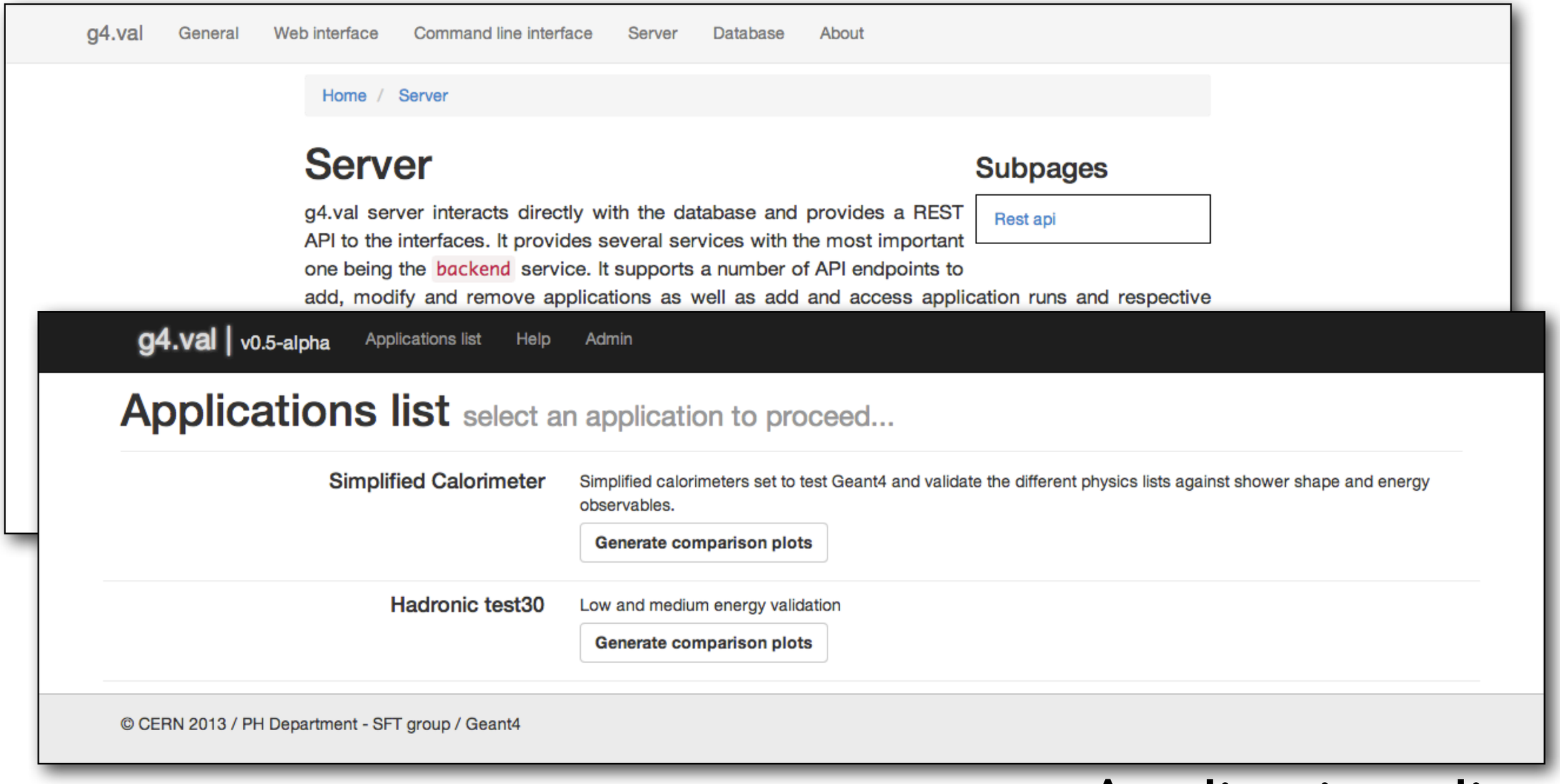
Results analysis



- **Initiated by the user**
  - through the web interface (AJAX API)
- Analysis functionality is served by the **analysis plugins**
  - which can run outside of the server

Web interface

Documentation



Applications list

Web interface

g4.val | v0.5-alpha Applications list Help Admin

Comparing runs Simplified Calorimeter (change application)

Parameters

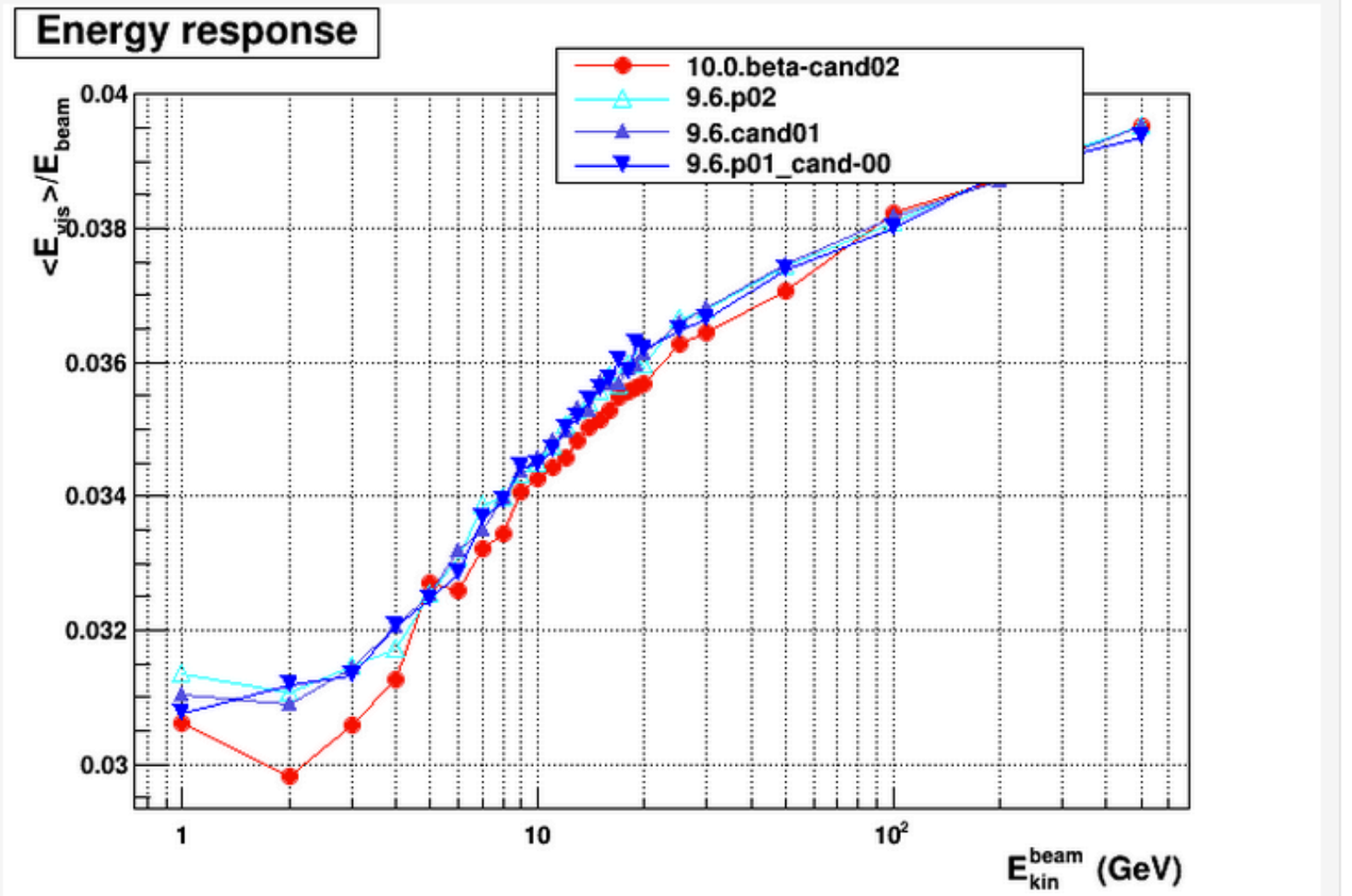
Observables

- [x] Energy response
- [x] Energy resolution
- [ ] Normalized width
- [ ] Shower center
- [x] Lateral shower shape
- [x] Longitudinal shower shape

Comparing runs options

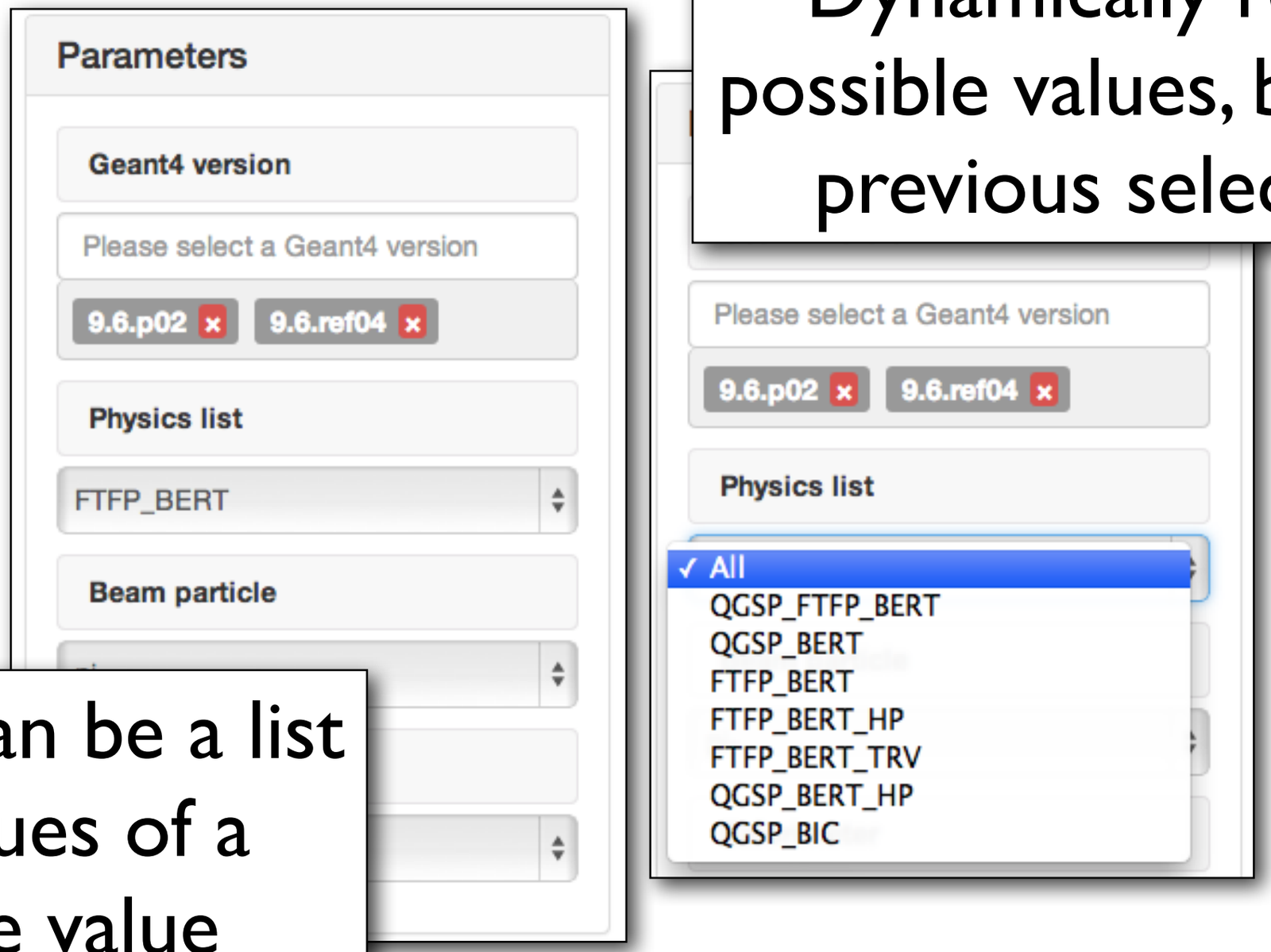
Run Bookmark form

Tab #2



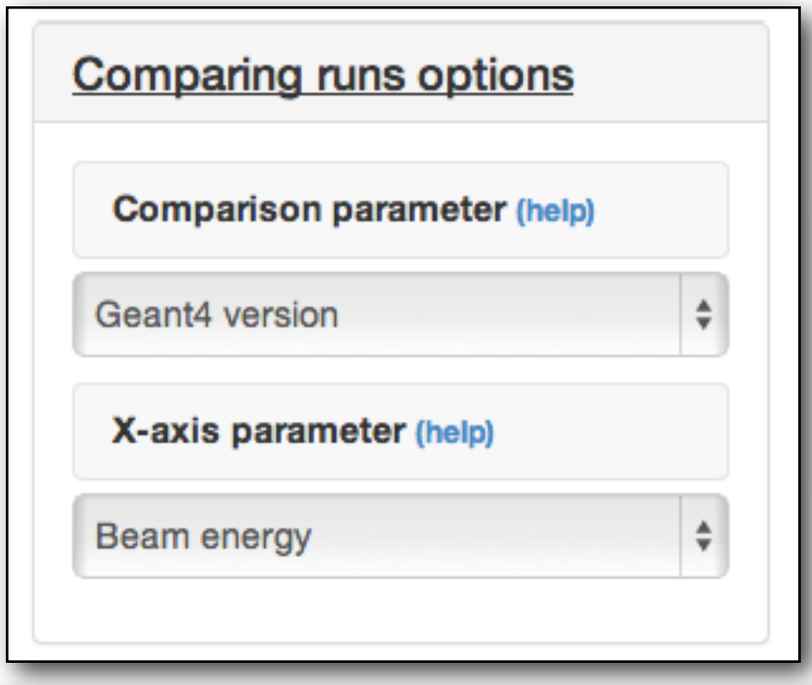
Web interface

Dynamically reduce possible values, based on previous selections

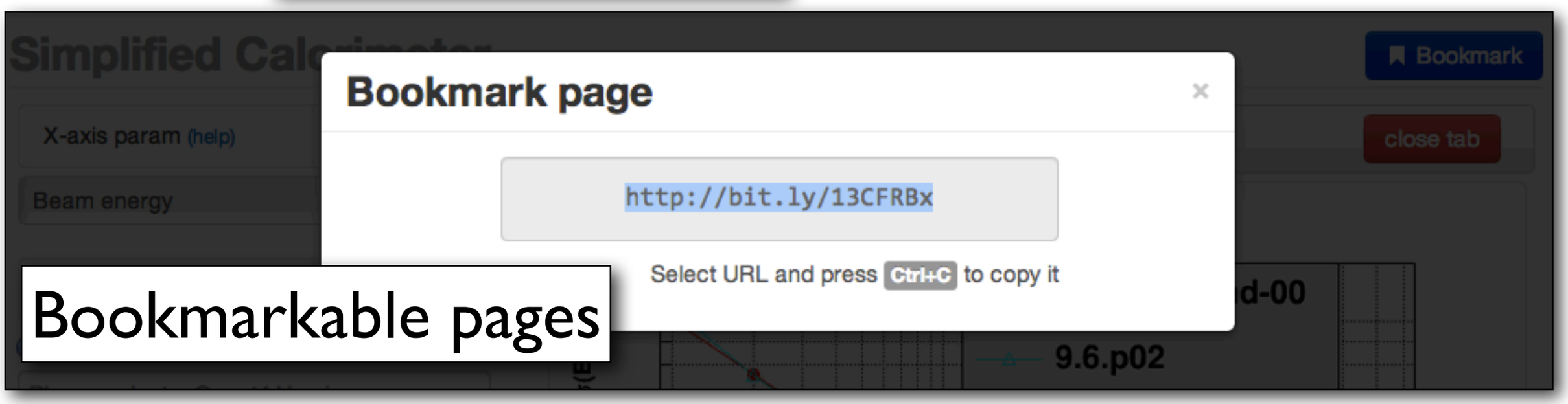


Filters can be a list of values of a single value

Web interface



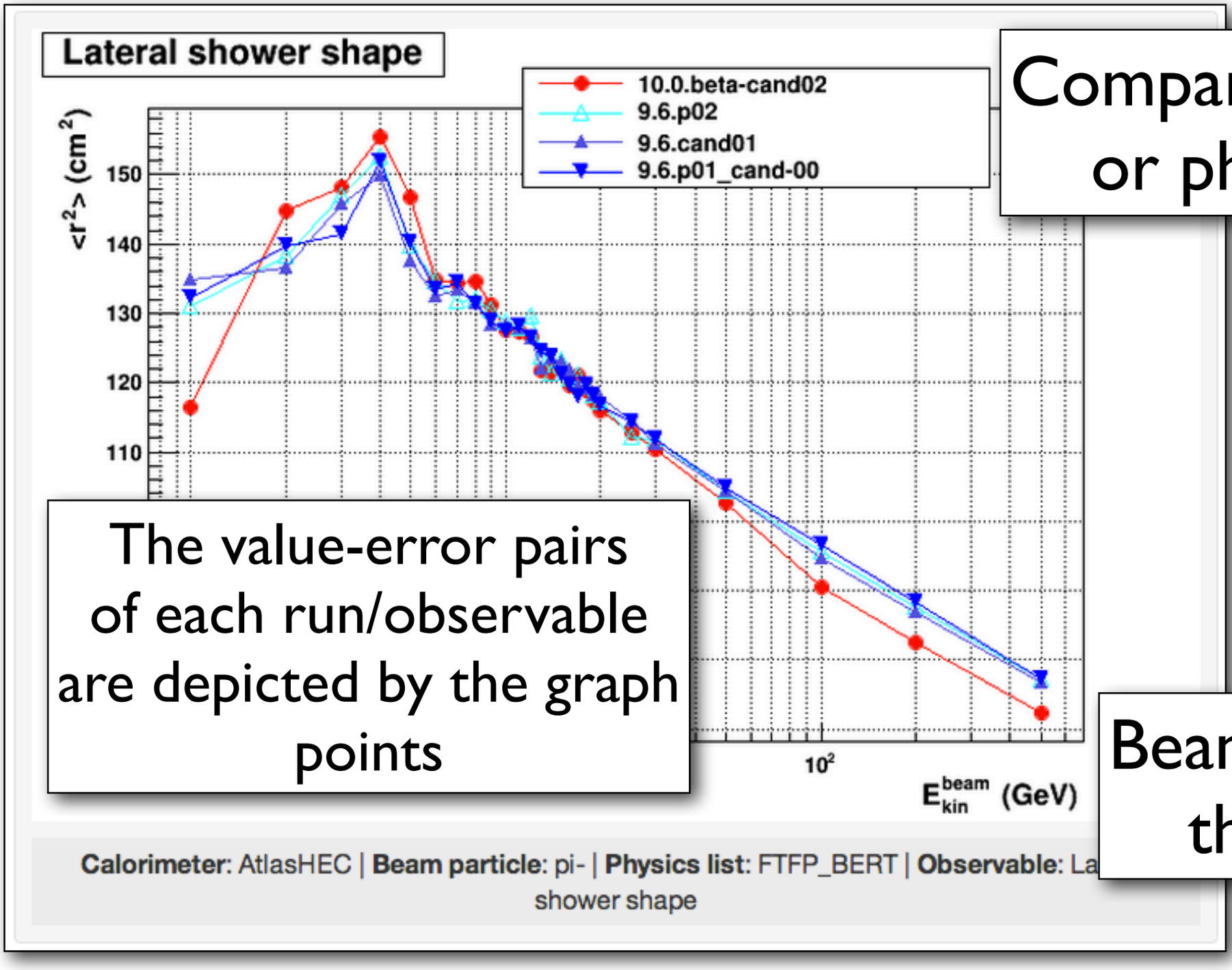
Plotting plugin options



Bookmarkable pages

Use cases

Simplified calorimeter

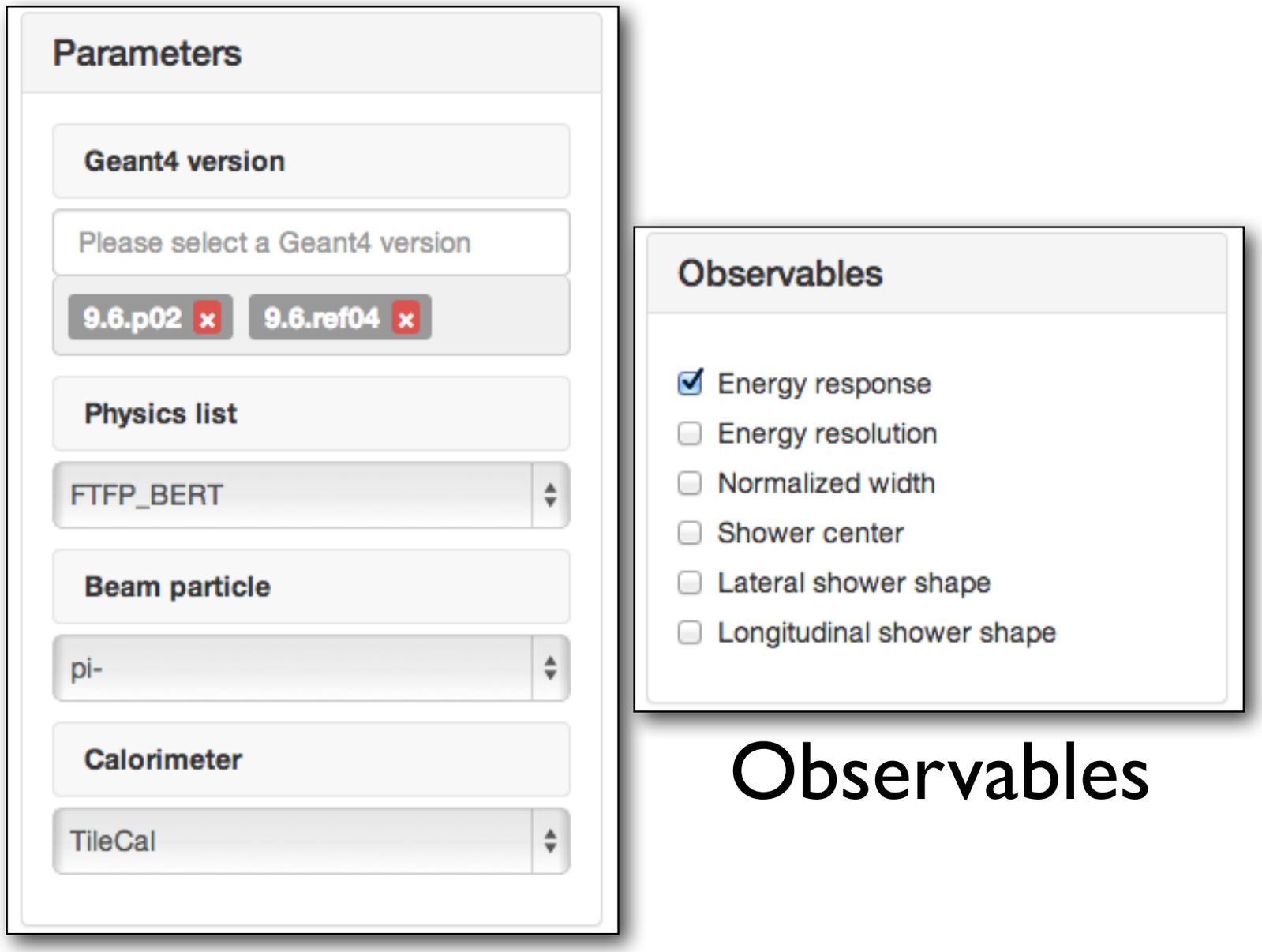


Comparing versions or physics lists

The value-error pairs of each run/observable are depicted by the graph points

Beam energy as the X-axis

Simplified calorimeter



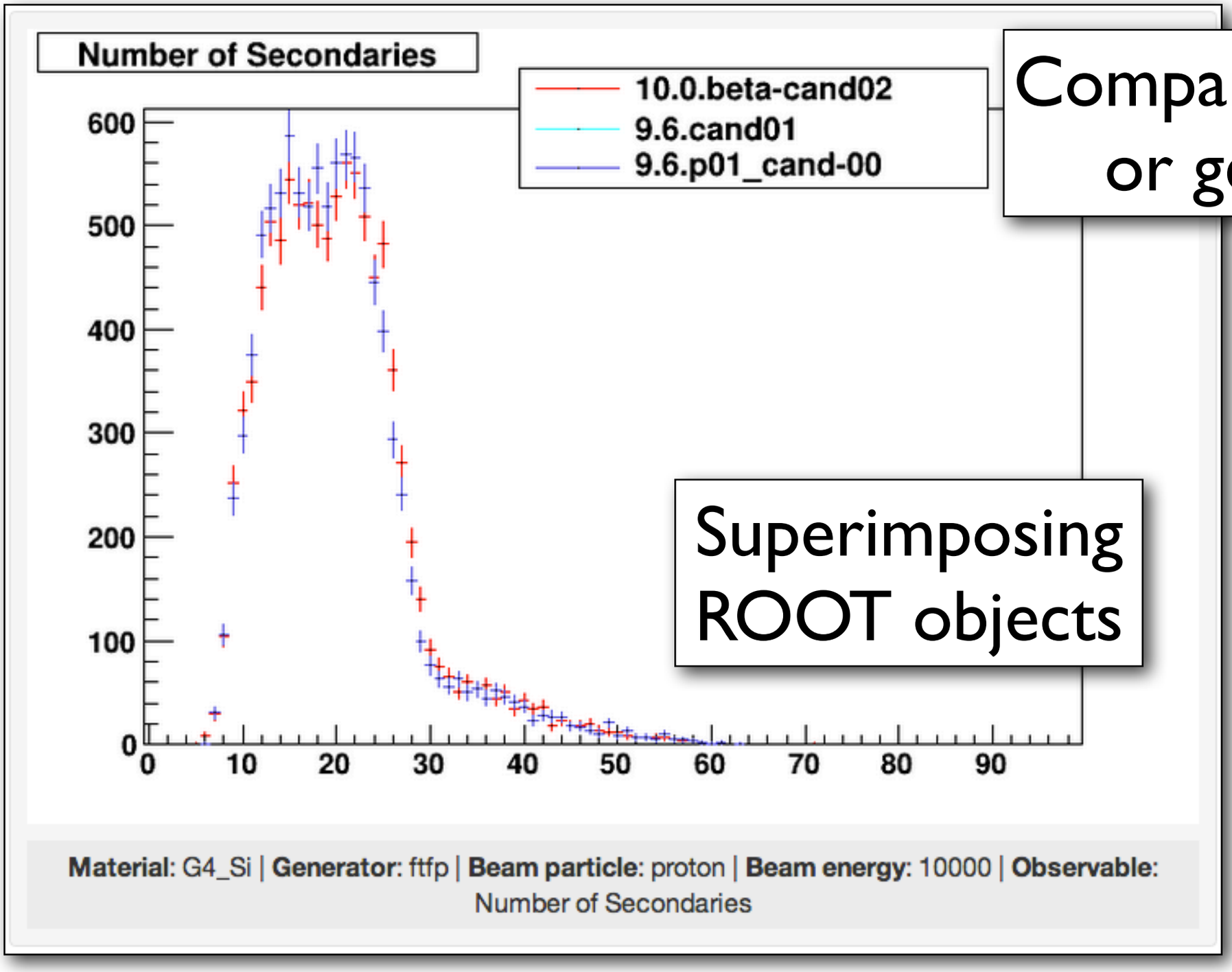
Observables

Parameters

Simplified calorimeter

- **Harvesting** is done within 350 lines of Python code
  - filters the output ROOT files
  - makes some fits and get the mean and variance from them
- Results are stored in Runs metadata file which is sent to the database using the **CLI**
  - a direct REST API call is also possible

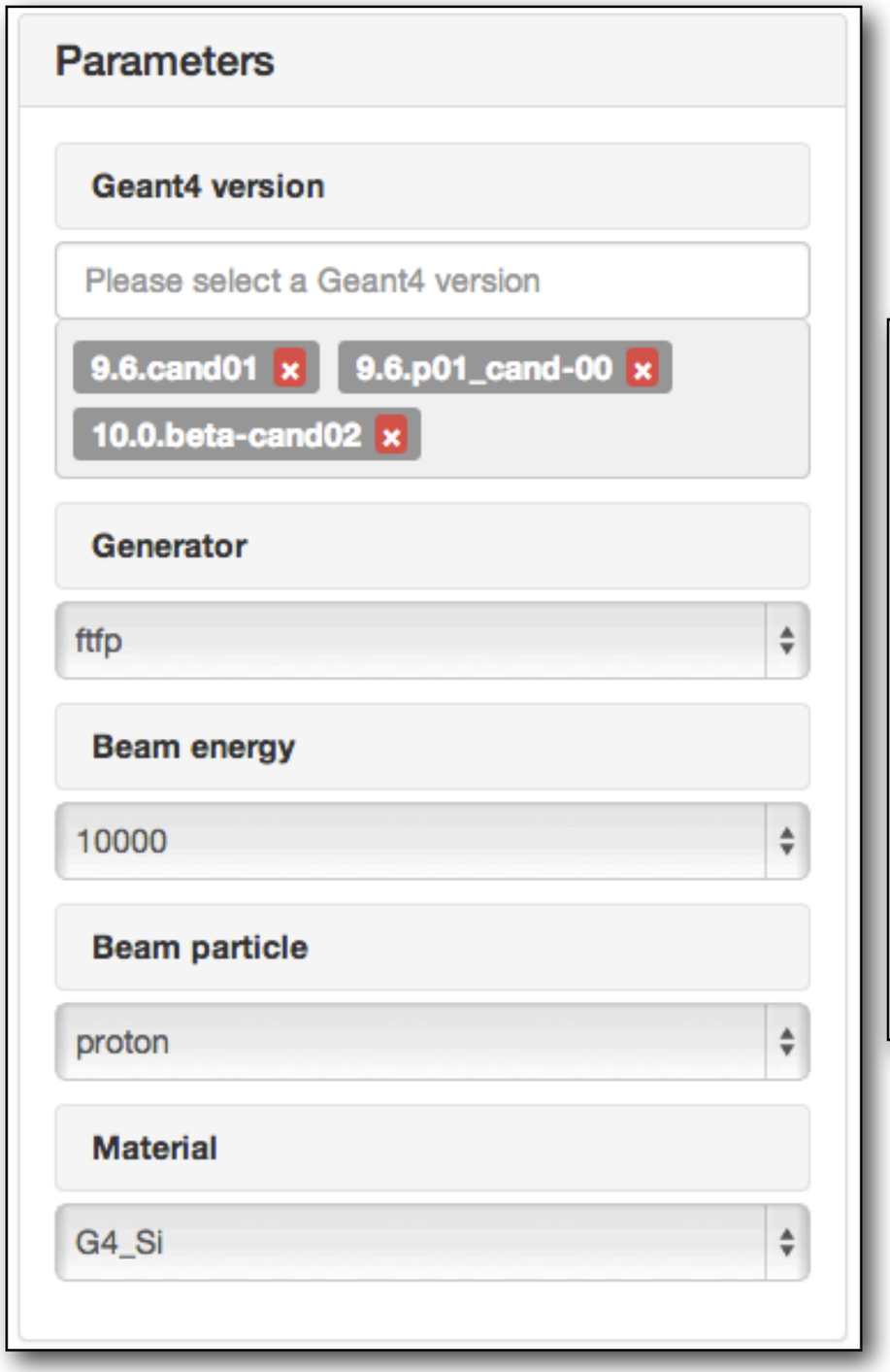
Test30



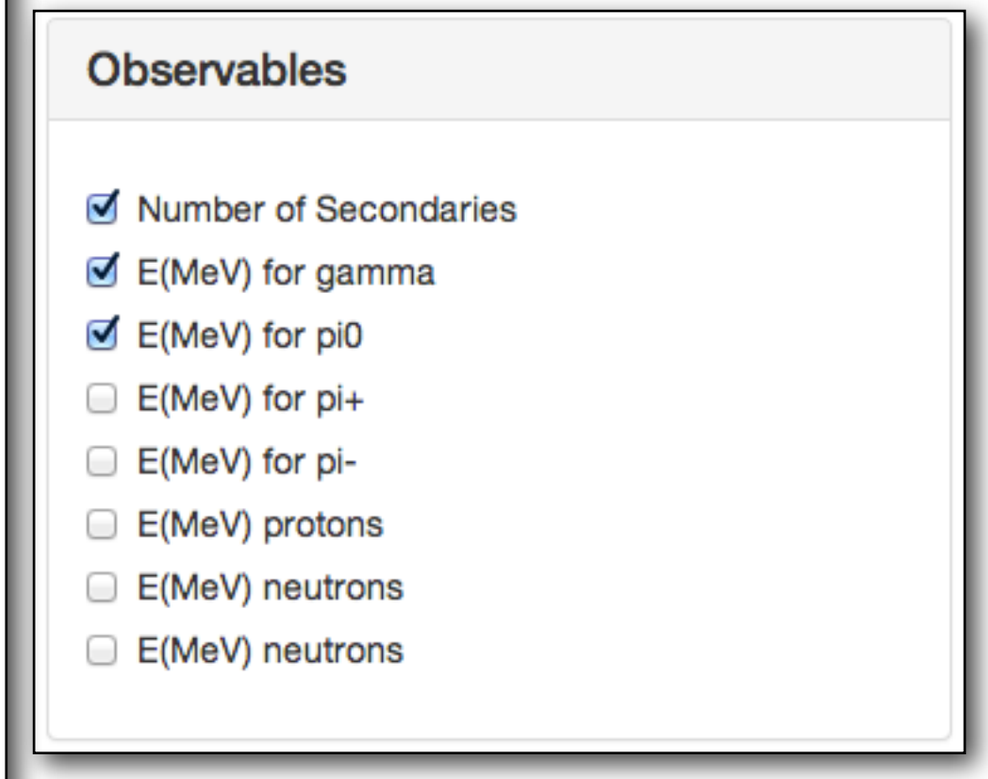
Comparing versions or generators

Superimposing ROOT objects

Test30



Parameters



Observables

Test30

- Uploading the **produced** (by the test) **ROOT files**.
- Observables are **mapped** to some of the **histograms** stored in this files.
- Plotting analysis plugin is fetching and **superimposing these histograms**.

Conclusion

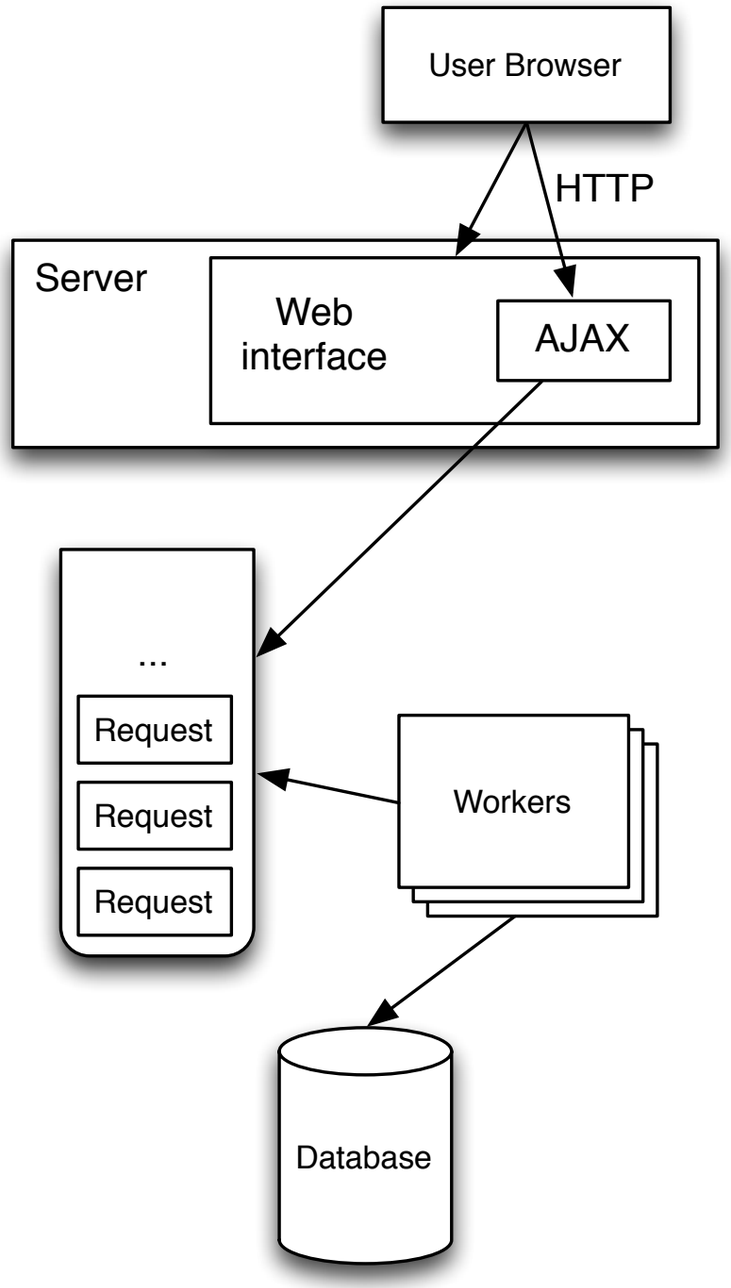
- A **generic system** for producing, storing, analyzing and presenting validation results
- DIRAC based production with Geant4 interface
- Optimized schema for storage of runs and results
  - performance critical part
- Flexible analysis (not only plotting)
- Interactive web interface

Backup slides

Modeling the problem

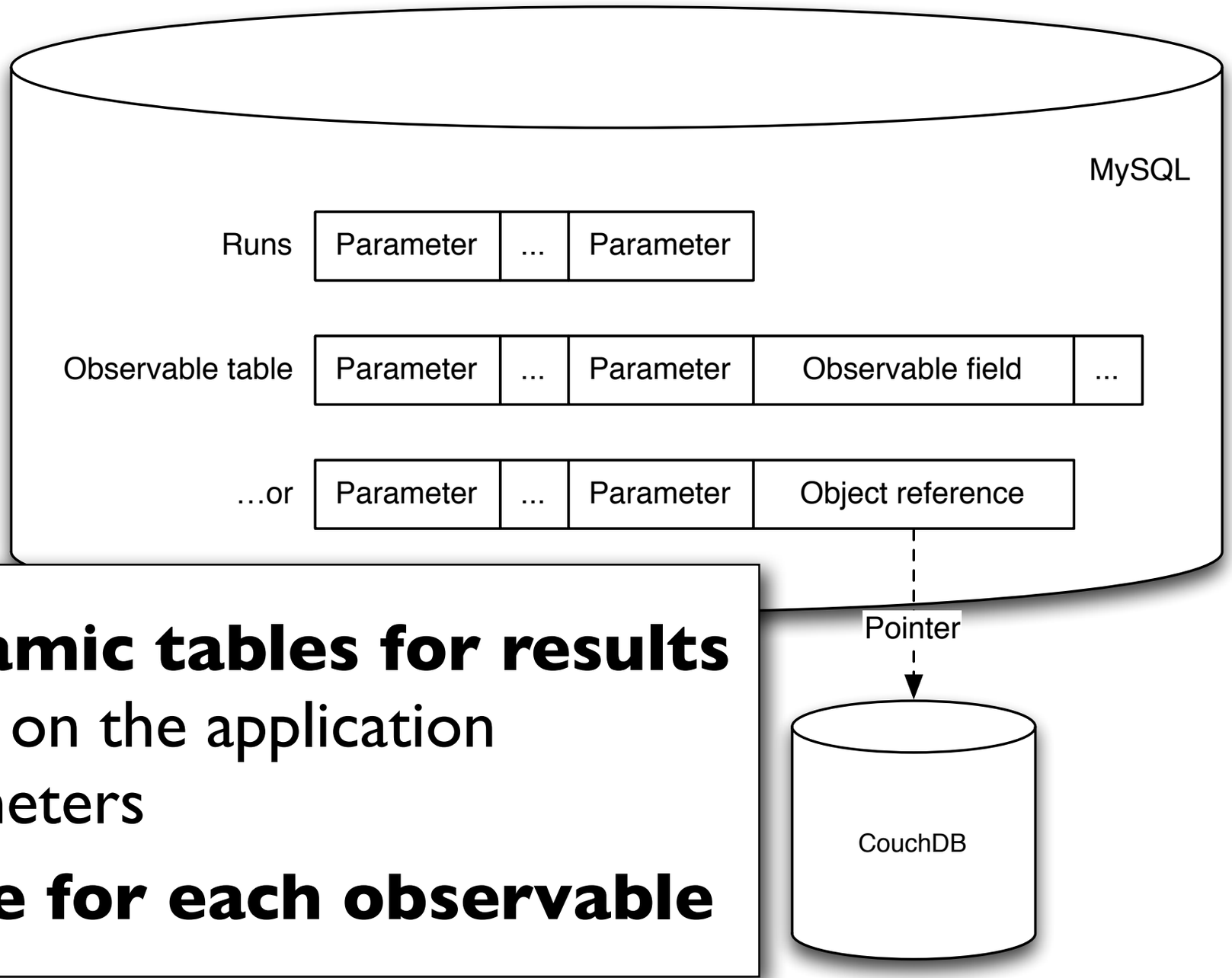
- **Application** independent test that focus on a few aspects of produced results
- **Run** that identifies uniquely the results of an application under certain conditions
- **Parameters** which define a run
  - the Geant4 version, physics list,...
- **Observables** that describe the results
  - CPU utilization, energy response,...

Analysis workers



- Load of analysis is distributed by using a set of **workers**
  - Requests are forwarded to an internal **queue**
  - A simple load balancer is monitoring the queue and deploys or kills workers
- Workers can use **external software**

Database schema



- **Dynamic tables for results** based on the application parameters
- **Table for each observable**

Database schema

- **Dynamic tables for results** based on the application parameters
  - runs and results indexing in MySQL
- **Table for each observable**
  - no repeated data (except than keys) -- Boyce–Codd normal form
- **Performance**
  - using each model (relational / non-relational) for what it is designed for
  - database normalization

Web interface technologies

Web interface technologies

- **Django:** web server framework
- **Twitter Bootstrap (jQuery):** CSS / JavaScript framework to implement the interface
- **Backbone.js:** JavaScript library for *MVC driven* interfaces
- **Redis and Python rq:** for the analysis plugins' workers