Cornell University
Laboratory for Elementary-Particle Physics

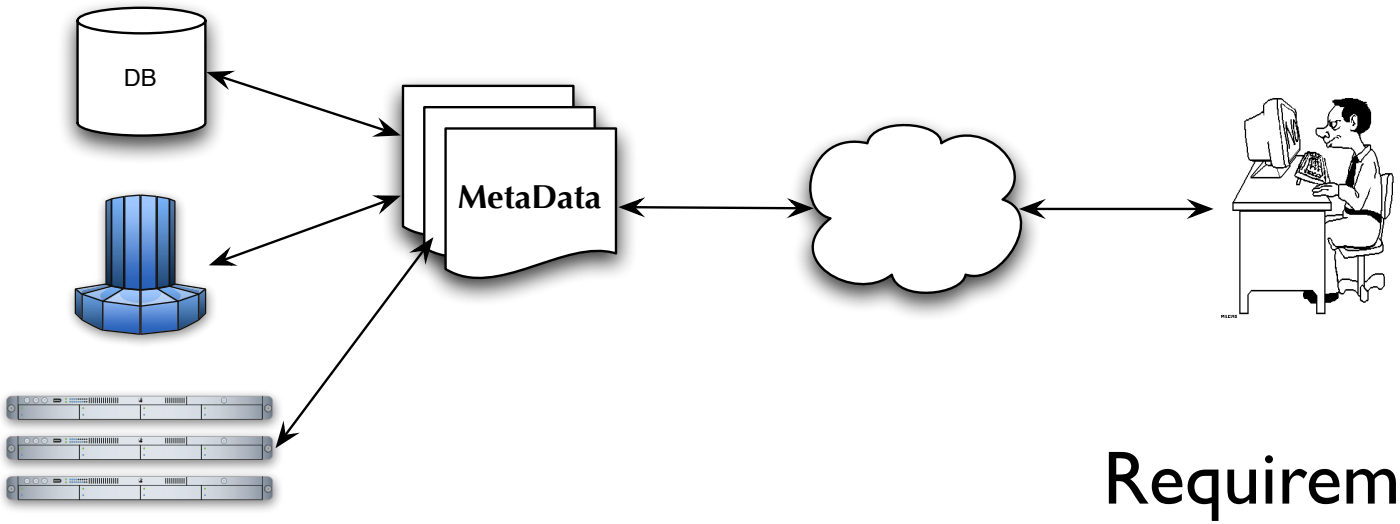
Search for CMS data: rapid web development using python and AJAX

Valentin Kuznetsov
Cornell University

CMS

Introduction

LHC experiments set a new scale for data management



Requirements:

- Easy data access
- Dynamic content and navigation
- intelligent data query
- integration with other applications

Choice

Requirements

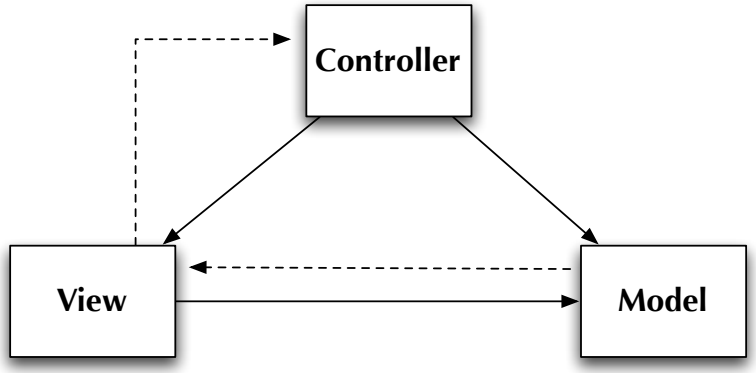
- Fast turn around with changes, schema/tools evolution
- Rapid web development
- different views for data discovery (production, physicists, etc.)

- Java too heavy, Ruby too new
- Python is well adopted, used almost everywhere in CMS
  - easy to use, very flexible, tons of tools, ideal for prototyping
- AJAX (Asynchronous JavaScript and XML) is a tool to provide dynamic web content

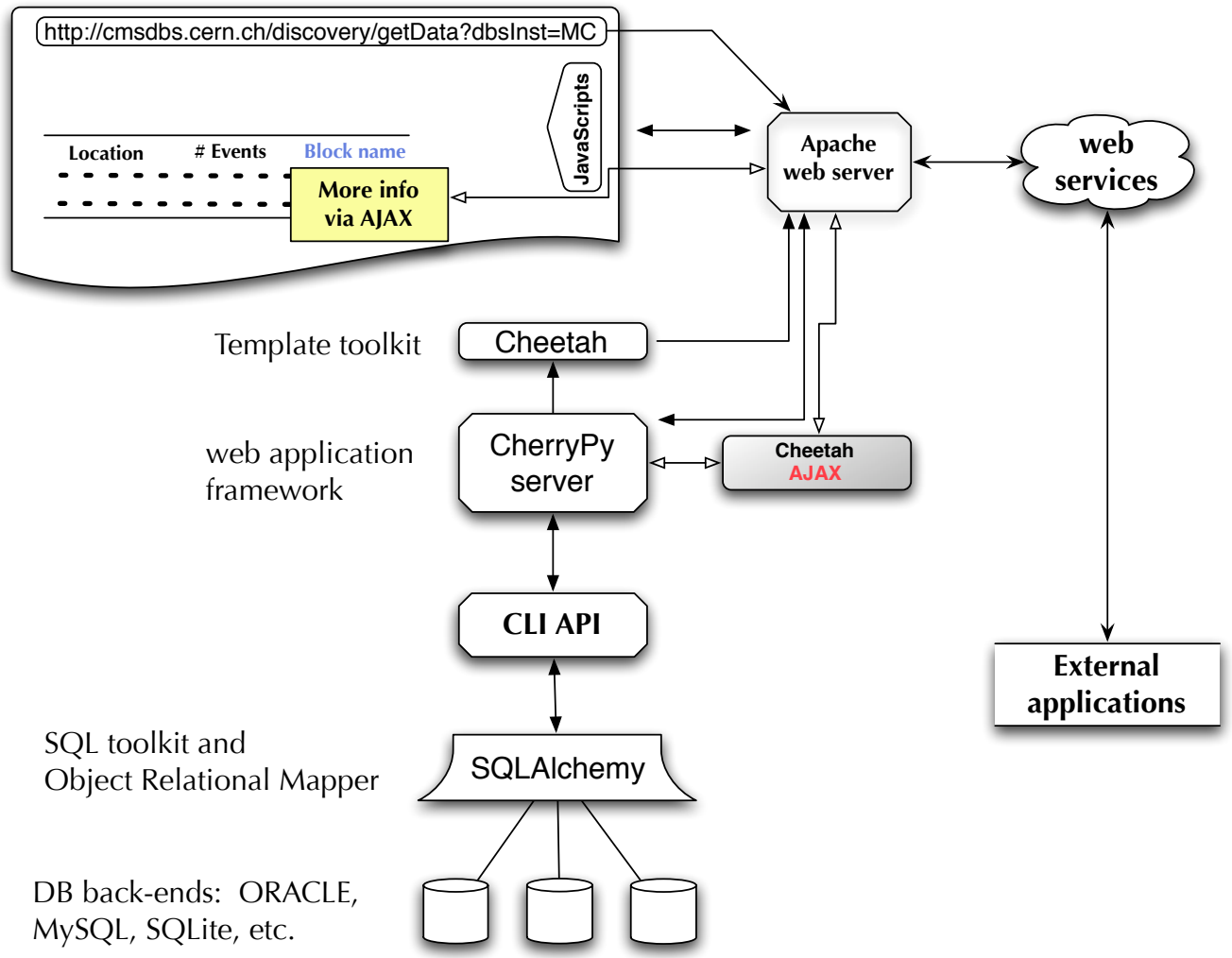
Tools

- Java Script toolkits:
  - Really Simple History (RSH) framework makes it easy for AJAX applications to incorporate bookmarking and back and button support.
  - Yahoo UI (YUI) as a cross-browser JavaScript library
  - OpenRICO as AJAX framework
- Cheetah for web page templates
- CherryPy as a reach web server
- SQLAlchemy for transparent DB access (SQL abstraction layer/Object Relational Mapper)

Stackable approach: model-view-controller architecture



Workflow


http://cmsdbs.cern.ch/discovery/getData?dbsInst=MC
JavaScripts
Location
Events
Block name
More info via AJAX
Apache web server
web services
Template toolkit
Cheetah
web application framework
CherryPy server
Cheetah AJAX
CLI API
External applications
SQL toolkit and Object Relational Mapper
SQLAlchemy
DB back-ends: ORACLE, MySQL, SQLite, etc.

Example

CherryPy server

```
import cherrypy, MyTemplate
from   Cheetah.Template import Template

class Main(object):
    def ajaxResponse(self,data,**kwargs):
        cherrypy.response.headerMap['Content-Type']="text/xml"
        return """<ajax-response>
<response type="element" id="dataHolder">
<div>We got data: '%s'</div>
</response></ajax-response>
        """%data
    ajaxDo.exposed=True

    def index(self):
        t = Template(MyTemplate,searchList=[{'iList':[1,2,3]}] )
        return str(t)
    index.exposed = True

cherrypy.root = Main()
cherrypy.server.start()
```

JavaScript (AJAX callback)

```
function getData(data) {
   ajaxEngine.sendRequest('getData',"data="+data);
}
function registerAjaxCalls() {
ajaxEngine.registerRequest('getData','ajaxResponse');
ajaxEngine.registerAjaxElement('dataHolder');
}
```

MyTemplate

```
<div id="dataHolder"></div>
#for item in $iList
<a href="#" onclick="getData('$item')">Ajax call $item</a>
#end for
```

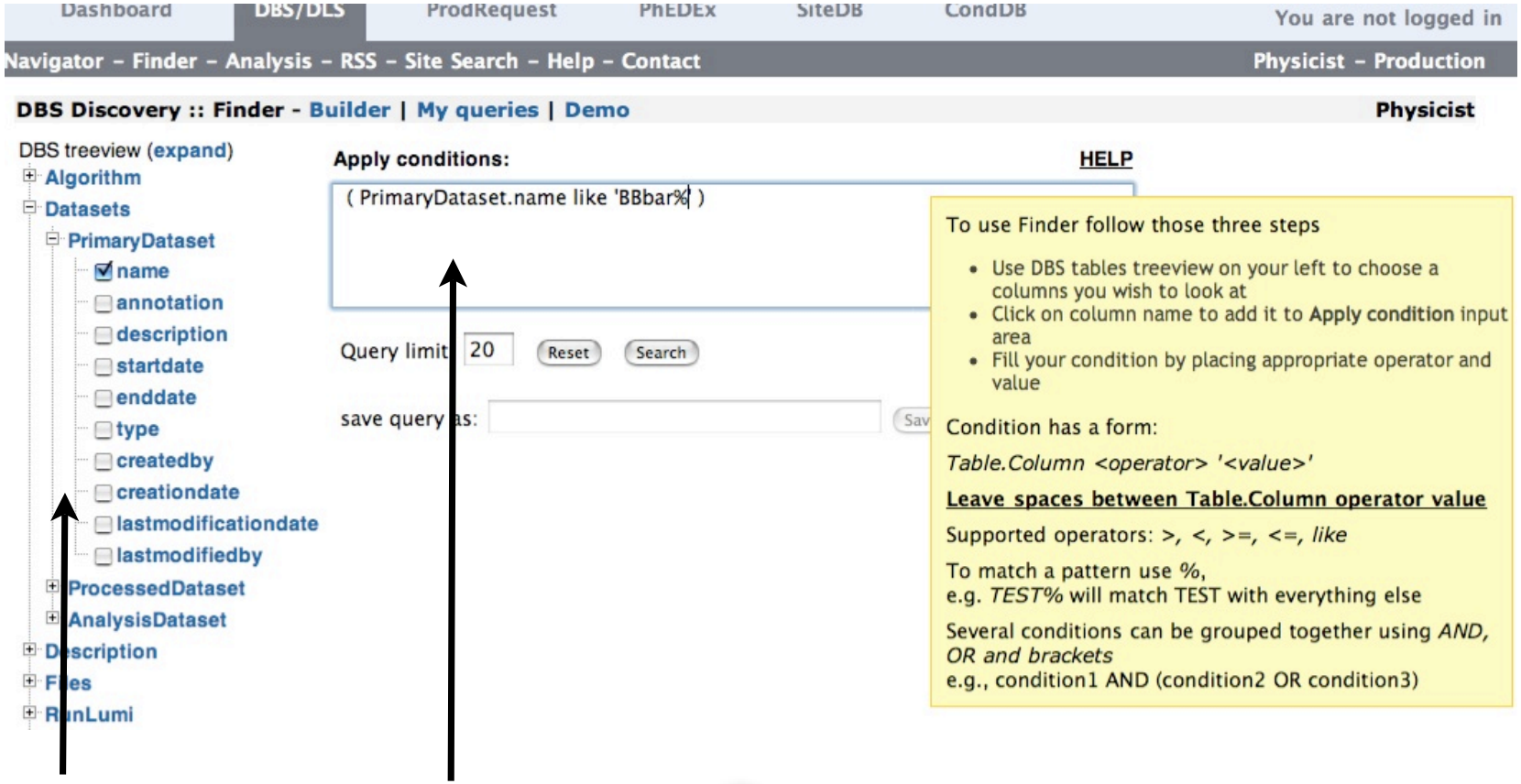
Fast, simple, elegant

Data discovery

- **Physicists & Production views** provide different level of details over your data lookup
- **Navigator** is a menu driven approach
- **Finder** constructs your own queries
- **Site search** explores data on your site
- **RSS feeds** to stay tune with your data

Regardless which way you go your data is just one click away

Finder as an example



Explore MetaData

Place your condition

Query builds dynamically

Finder under the hood

- Auto-load schema from DB using SQLAlchemy
  - store it into the cache
  - send them via AJAX to the web form
- Establish a short path between Table A to Table D using foreign key relationships
  - Build query, validate applied condition (where clause)
  - send back results with a form to lookup dataset out of your query results
- iterate if necessary

Summary

- Web development using python & AJAX is simple as 1,2,3
  - Python: a lot of tools around
  - AJAX: variety of toolkits is available, pick right one for your needs
- We used stackable design and Model-View-Controller architecture
  - concentrate on physicists needs