The Defects Database

Peter Onyisi (Chicago), Peter Waller (Liverpool)

7 Apr 2011

THE UNIVERSITY OF CHICAGO

UNIVERSITY OF LIVERPOOL

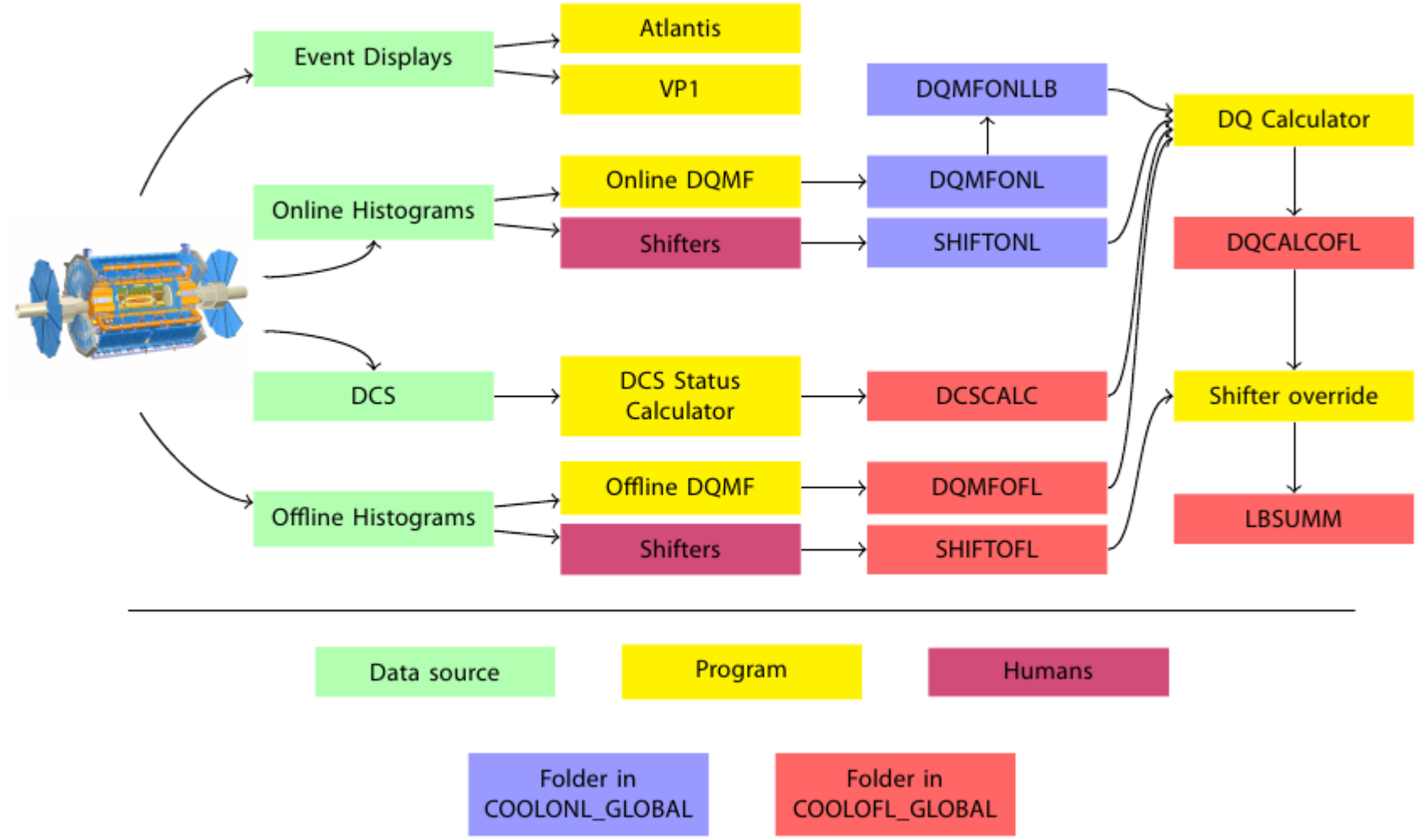
Introduction

- The central repository of Data Quality information is kept in COOL
  - essential requirement: we must be able to produce TAG-style XML files that encode run/lumiblocks to be used for various analyses
- The “flag” system used in 2009-10 has recently been replaced with a new “defect” system
  - Improved transparency for DQ experts, end users

Old System

- Database granularity was “subsystem” – e.g. LAr barrel A side, HLT muon slice, barrel photons
- Several COOL folders with identical channels used to store various inputs (from shifters, automatic DQMF, automatic DCS) and their combinations (all automatic, final)
- Information stored per IOV:
  - Flag color (Green, Yellow, Red, Grey, Black)
  - Comment string
  - Dead fraction and Thrust (former used only for DCS, latter never used)

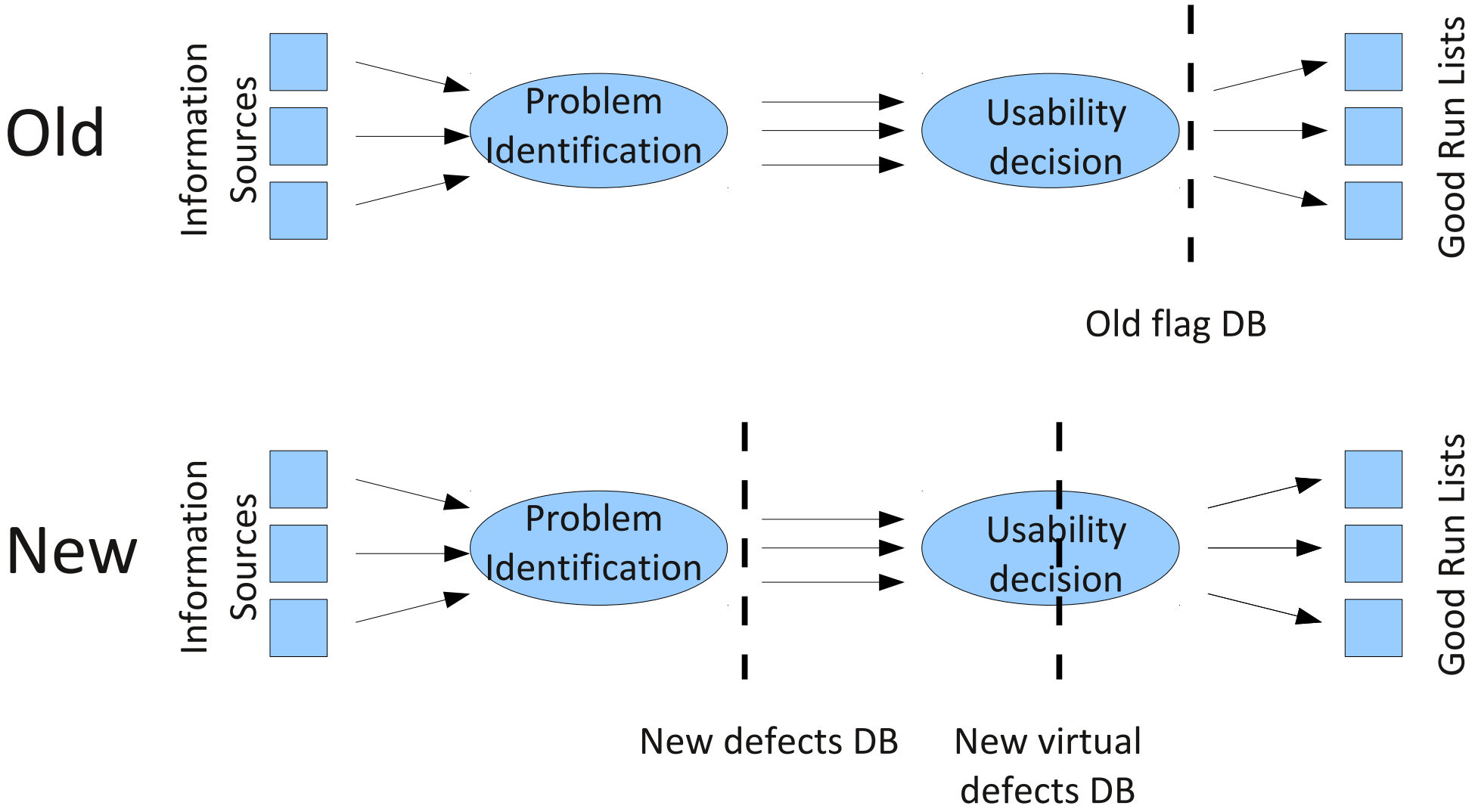
Old System (2)


Event Displays
Atlantis
VP1
Online Histograms
Online DQMF
Shifters
DCS
DCS Status Calculator
Offline Histograms
Offline DQMF
Shifters
DQMFONLLB
DQMFONL
SHIFTONL
DCSCALC
DQMFOFL
SHIFTOFL
DQ Calculator
DQCALCOFL
Shifter override
LBSUMM
Data source
Program
Humans
Folder in COOLONL_GLOBAL
Folder in COOLOFL_GLOBAL

Problems with Old System

- Could only explicitly query flag color: no way of getting exact problems
  - Had to search through comments
- Colors had no obvious meaning other than “Green” = Good and “Red” = “Do not use”
  - “Yellow” had semantic drift from “use with caution” to “expected recoverable”
- Interdependence of flags not explicitly stated
  - much shifter confusion: e.g. if LAr has hot cells that will be masked, are electrons green or not?

New “defects” vs old “flags”



New defect concept

- What we really want to store is what went wrong with the detector, not only a “yes-no” decision
  - We can put that off to a later stage!
- Create a new database with one channel per problem, or “defect”
  - with reasonable granularity, e.g. “severe noise” or “2 or more RODs disabled”
- Then combine these defects together (“virtual defects”) to make default selections for what is bad enough to make you throw away data
  - These defaults are the “intolerable” defects

Implementation

- Two COOL folders: DEFECTS and DEFECTLOGIC
- DEFECTS has one channel per defect. IOVs need not be written for any given run/LB (they are considered absent in this case). The payload is:
  - Present: boolean, true if defect is present, false if absent (e.g. shifter unset an LB set in error)
  - User: string, username that set the defect
  - Recoverable: boolean, true if problem is expected to be recoverable
  - Comment: string, user comment

Implementation (2)

- DEFECTLOGIC implements the virtual defects
- A virtual defect is defined by the set of defects (or virtual defects) it depends on. Any of them being present will cause the virtual defect to be present.
- Old virtual flag system allowed run-dependent definitions. We decided this introduced too many problems for the benefit and have removed this feature from the virtual defects.

Statistics

- Currently:
  - 563 defined defects (old system ~ 100 flags)
  - 165 virtual defects
  - 26596 defect IOVs defined in HEAD
    - < 2 sec to retrieve all of them into a python job from a cold start
    - average ~ 50 defect IOVs per good run: many related to warm start/stop

Defect Access

The primary defect DB access API is in the package "DataQuality/DQDefects"

- written in Python
- implements all standard manipulations: defect creation, entry, retrieval, folder tagging, some metadata operations, test SQLite DB creation
- Relies on COOL access utilities in "DataQuality/DQUtils" – very useful tools for anyone working with COOL!
- additional scripts provided for common actions e.g. copying defects from one DB/tag to another
    - AtlCoolCopy & friends **do not work for defects** because channel numbering is not guaranteed to be consistent!

Defect Access (2)

- We have only identified one use case for direct C++ access to the defects DB so far: Athena filters
  - Here we want to stop monitoring algorithms from looking at known bad LBs
  - Information on channels to use is obtained from the DB at Athena configuration time using python, then IOVDbSvc retrieves the actual defects
- In all other cases we want people to interact with the DB only through the python API

Python API example

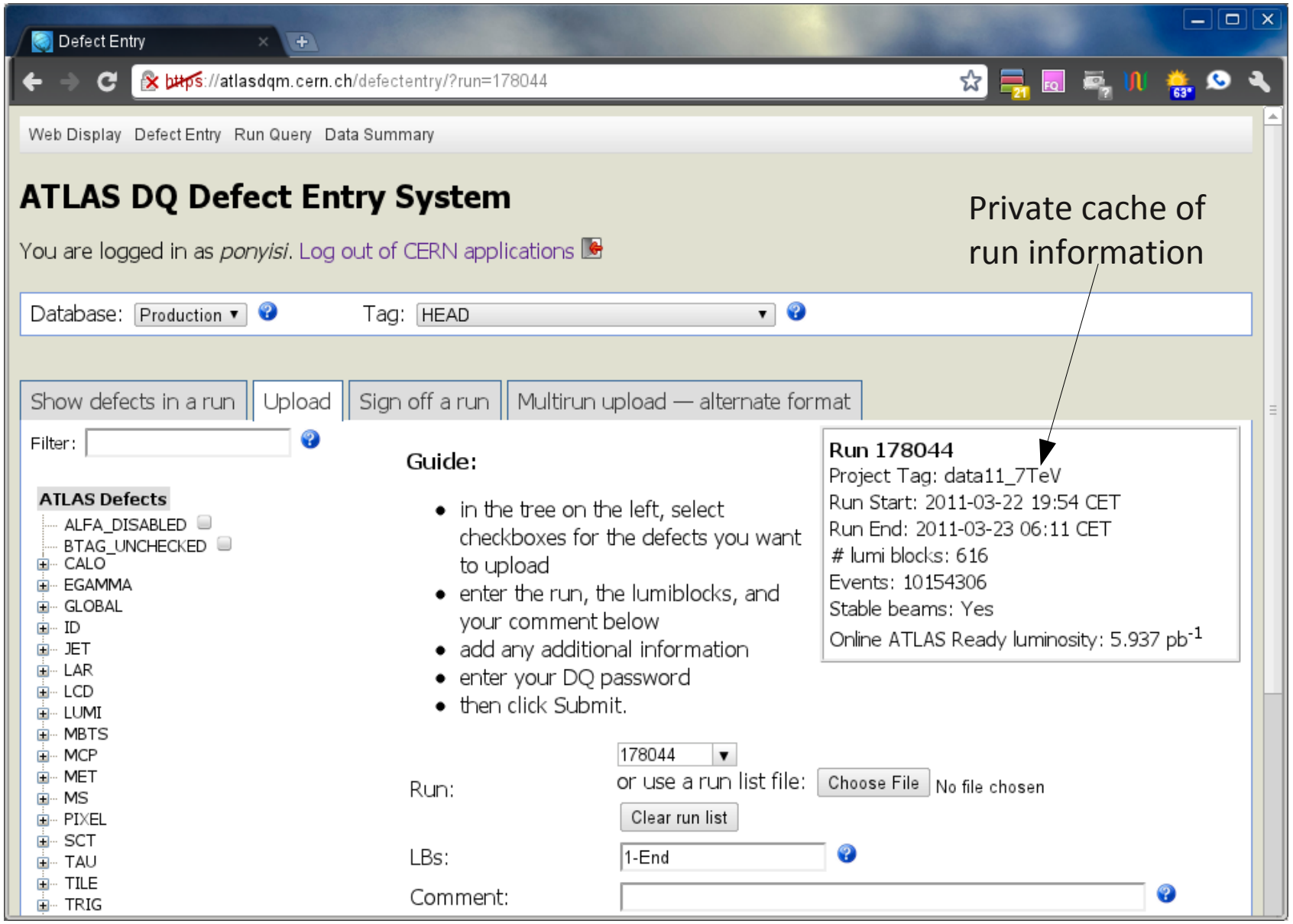
```
>>> from DQDefects import DefectsDB
>>> ddb=DefectsDB('oracle://ATLAS_COOLPROD;schema=ATLAS_COOLOFL_GLOBAL;dbname=COMP200')
>>> iovs=ddb.retrieve(since=(178044,1), until=(178044,0xffffffff), primary_only=True)
>>> len(iovs)
122
```

- Works in any recent 16.6 release

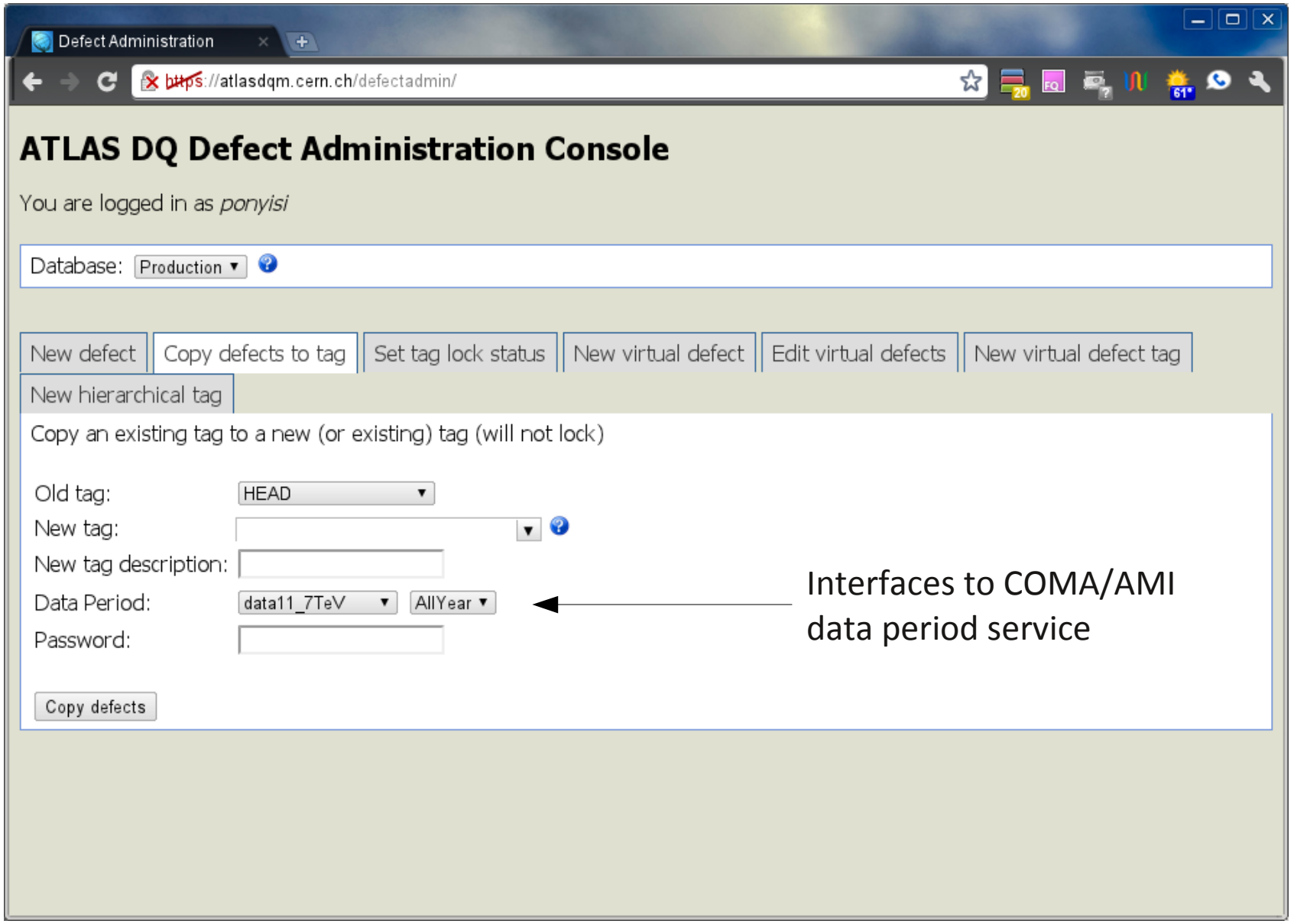
Utilities

- There are five main user-facing applications using the defects DB
  - Shifter entry interface
  - Administration interface
  - Run Query
  - GRL generator
  - XML-RPC APIs
- The first two are newly-developed web applications
  - backend is our CherryPy server using the DQDefects code
  - applications rely heavily on AJAX (actually JSON) – much more simple, reliable, responsive than previous application
- The latter three were adapted to the new system

Shifter Entry Interface



Administration Interface



Run Query

Thanks to Joerg Stelzer

Virtual defects shown by default as well

Run	Links	#LB	Start and endtime (CEST)	#Events	Data Quality (* # HEAD)	
178109 Period: AllYear,B,B2	DS, RS, BS, AMI, DQ, ELOG, DCS:SoR/EoR, OKS	865 (60 s)	Wed Mar 23 2011 19:53:40 – Thu Mar 24, 10:21:57	10,806,772 (207.4 Hz)	CP_BTAG_LIFE	153, 165, 183, 185-187, 233, 238-246, 247, 321-323, 351, 358, 359, 398, 419, 425, 449, 468, 470, 479, 494, 570, 605, 652, 745, 804, 849-851
					CP_BTAG_SOFTE	153, 165, 183, 185-187, 233, 238-246, 247, 321-323, 351, 358, 359, 398, 419, 425, 449, 468, 470, 479, 494, 570, 605, 652, 745, 804, 849-865
					CP_BTAG_SOFTM	153, 165, 183, 185-187, 233, 238-246, 247, 321-323, 351, 358, 359, 398, 419, 425, 449, 468, 470, 479, 494, 570, 605, 652, 745, 804, 849-865
					CP_EG_ELECTRON_BARREL	183, 238, 239-246, 247, 398, 425, 849-865
					CP_EG_ELECTRON_CRACK	153, 165, 183, 233, 238, 239-246, 247, 351, 358, 359, 398, 419, 425, 449, 468, 470, 479, 494, 570, 605, 652, 745, 804, 849-865
					CP_EG_ELECTRON_ENDCAP	153, 165, 183, 185-187, 233, 247, 321-323, 351, 358, 359, 398, 419, 425, 449, 468, 470, 479, 494, 570, 605, 652, 745, 804, 849-865
					CP_EG_ELECTRON_FORWARD	145-153, 165, 183, 185-187, 233, 238, 239-246, 247, 321-323, 351, 358, 359, 398, 419, 425, 449, 468, 470, 479, 494, 570, 605, 652, 745, 804
					CP_EG_EMCLUSTER_BARREL	183, 238, 239-246, 247, 398, 425, 849-865
						153, 165, 183, 233, 238,

Summary

- We have developed and deployed a new database for Data Quality information
  - Auxiliary tools developed/converted to handle new system
- In almost all cases using well-defined, fast Python API for access
  - Underlying DB manipulation tools available for general use