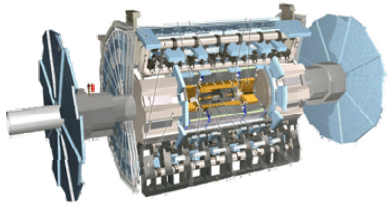
Distributed Computing in ATLAS

ATLAS data types



RAW data

- "bytestream" format, ~1.6MB/event, 200Hz (plus 20Hz of calibration trigger data)

reconstruction

ESD (Event Summary Data)

- full output of reconstruction in object (POOL/ROOT) format
- tracks and their hits, calo clusters, calo cells, combined reconstruction objects etc. (almost complete info)
- ~1 MB/ev initially, to decrease as the understanding of the detector improves
- compromise between "being able to do everything on the ESD" and "not enough disk space to store too large events"

AOD (Analysis Object Data)

- summary of event reconstruction with "physics" (POOL/ROOT) objects (electrons, muons, jets, etc.)
- nominal size 100 kB/event (now 200 kB/event including MC truth)
- reduced event representation, suitable for analysis

TAG: event level metadata, for identification and selection of events in AOD/ESD, stored also in relational databases, ~10KB/ev

DPD: Derived Physics Data, ntuple-style representation of event data for end-user analysis and histogramming, POOL/ROOT obj, ~20KB/ev

ATLAS Grid model

The ATLAS computing model embraces the Grid paradigm

Tier-0

- Copy RAW data to CERN Castor Mass Storage System tape for archival
- Copy RAW data to Tier-1s for storage and subsequent reprocessing
- Run first-pass calibration/alignment (within 24 hrs)
- Run first-pass reconstruction (within 48 hrs)
- Distribute reconstruction output (ESDs, AODs & TAGS) to Tier-1s

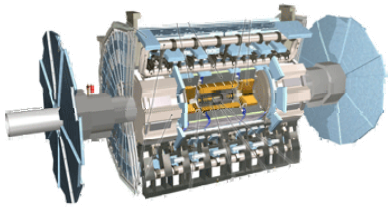
Tier-1s

- Store and take care of a fraction of RAW data (forever)
- Run “slow” calibration/alignment procedures
- Rerun reconstruction with better calib/align and/or algorithms
- Distribute reconstruction output to Tier-2s
- Keep current versions of ESDs and AODs on disk for analysis

Tier-2s

- Run simulation (and calibration/alignment when appropriate)
- Keep current versions of AODs on disk for analysis

ATLAS Grid model



RAW

T0

RAW, ESD,
AOD, TAG

CERN
CAF

RAW, ESD,
AOD, TAG

T1

T1

T1

AOD, TAG

AOD, TAG

AOD, TAG

T2

T2

T2

T2

T2

T2

T2

T2

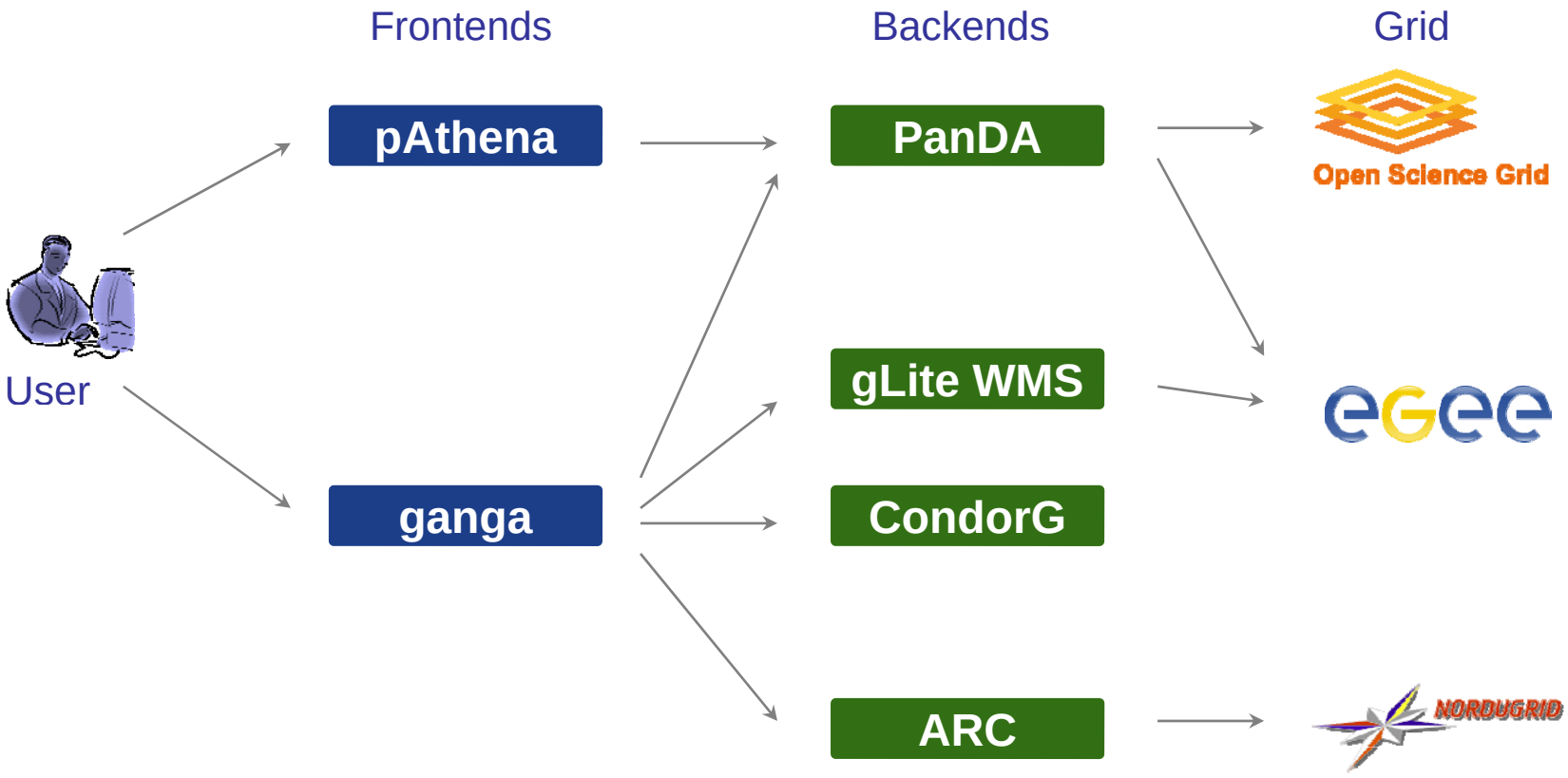
T2

ATLAS (simplified) analysis action sequence

- Access the metadata catalogue (AMI) and find the datasets of interest
  - Based on physics trigger signatures, time range, detector status etc.
- (Optional) Use the TAG data (in Oracle DB or ROOT format) to build a list of interesting events to analyse further
- Use Distributed Analysis tools (Ganga or pAthena) to submit jobs running on AOD data at Tier-2s (or on ESD at Tier-1s for larger-scale group-level analysis tasks)
  - Accessing only the selected datasets
  - (Optionally) taking the event list from the TAG selection as input
  - Producing DPD (Derived Physics Data) samples as output
    - Selected events in AOD format (skimming)
    - "Thinned/Slimmed" events in AOD format (selected event contents)
    - Any other simpler format (e.g. ntuples) for subsequent interactive analysis
  - Storing DPD on the Grid for group access or on local resources for interactive access
- DPD production can be also a group activity in case they can be used by several analyses
  - In this case DPDs must be stored on Tier-2s for global access
- Finish with interactive analysis (typically using ROOT) on the DPD files
  - Producing histograms and physics results

ATLAS Distributed Analysis

Using the grid “transparently”
Jobs are sent to the sites where needed input files are stored
Frontends take care of communicating with underlying layers



pAthena and PanDA

pAthena

pAthena is a glue script to submit user-defined jobs to distributed analysis systems (such as PanDA)

It provides a consistent user-interface to Athena users

- archive user's work directory
- send the archive to Atlas.Panda
- extract job configration from jobOs
- define jobs automatically
- submit jobs

pAthena

- available on AFS
- command line interface (`pathena` exec plus `pathena_util`)
- configured with cmt:

```
cmt co PhysicsAnalysis/DistributedAnalysis/PandaTools
```

- job submission syntax quite straightforward:

  when you run Athena with

```
athena jobO_1.py jobO_2.py
```

  all you need is

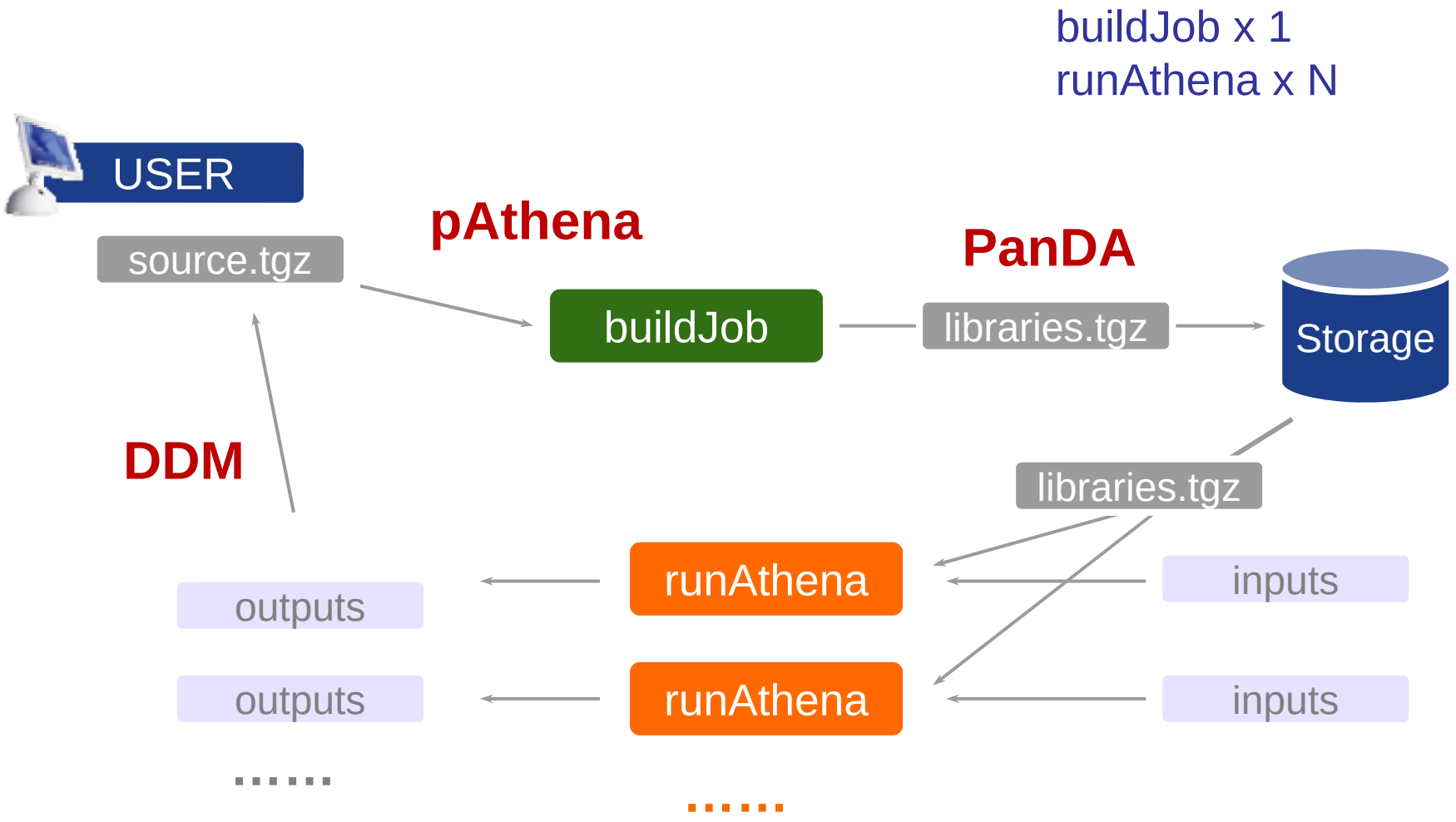
```
pathena jobO_1.py jobO_2.py [--inDS inputDataset] --outDS outputDataset
```

- extensive options set

```
--inDS, --outDS, --libDS, --split, -nFilesPerJob, --site, ...
```

- output files registered in DDM

pAthena architecture



pAthena supported job types

Can run all sort of Athena job types:

- all production steps (evgen, simul, pileup, digi, reco, merge, analysis)
- arbitrary package configuration
  - add new packages
  - modify cmt/requirements in any package
- customize source code and/or jobOptions
- multiple-input streams
  - e.g. signal + minimum bias
- TAG based analysis
- protection against missing or corrupted files
- production releases and nightlies

pAthena monitoring

from the command line:

```
pathena_util

>>> status(1)
=====================================
JobID    : 1
time     : 2008-02-16 15:12:05
inDS     : trig1_misal1_mc12.006384.PythiaH120gamgam.recon.AOD.v13003002_tid016421
outDS    : user.LeonardoCarminati.Hgg120-tutorialRoma3-rel13040
libDS    : user.LeonardoCarminati.lxplus225_76.lib._000001
build    : 7659827
run      : 7659828-7659837
jobO     : HggAnalysis_jobOption.py
site     : ANALY_BNL_ATLAS_1
----------
buildJob   : succeeded
----------
runAthena  :
          total : 10
      succeeded : 8
         failed : 1
        running : 1
        unknown : 0
----------------------
```

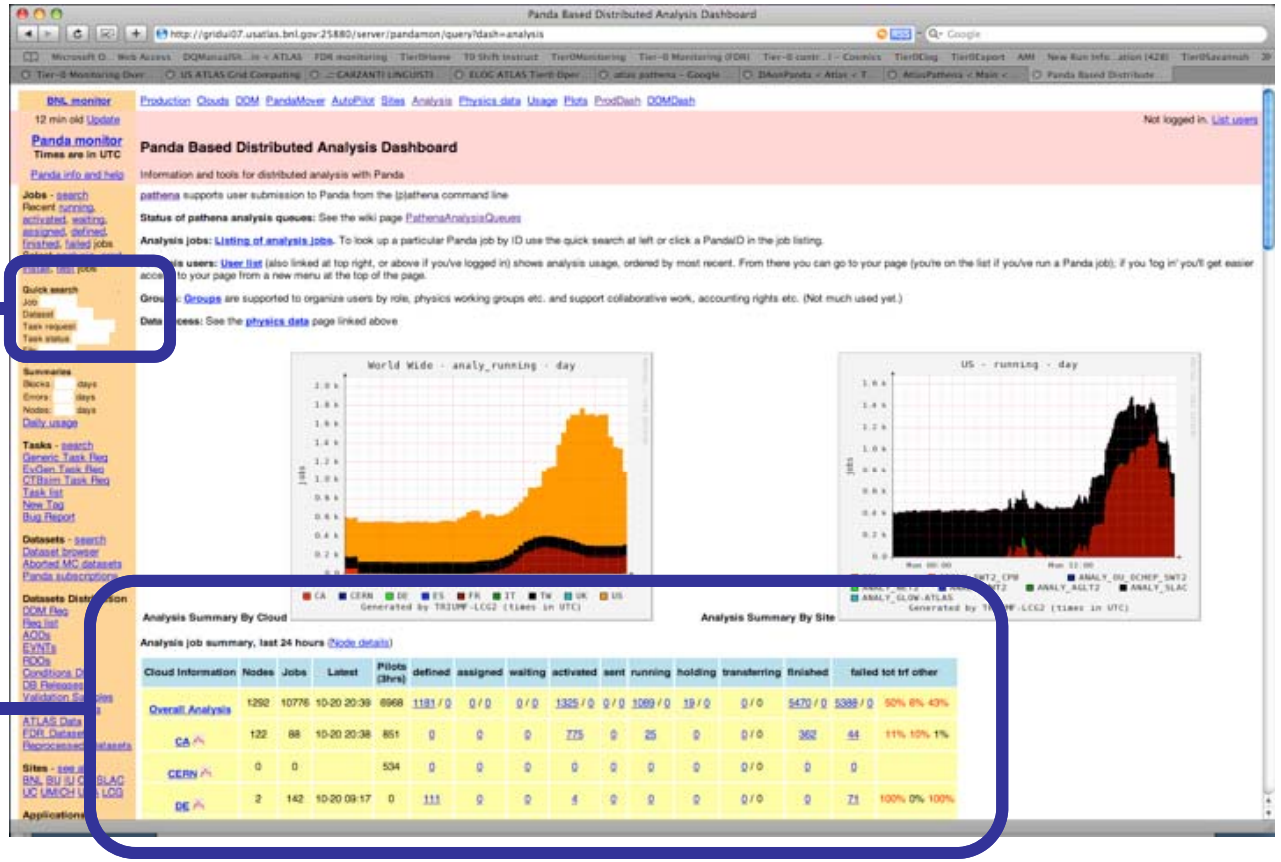
actual status of the job
going through
activated - running - holding

pAthena monitoring

Jobs submitted using pAthena pass through PanDA, thus they can be monitored through the PanDA monitoring page

enter PanDA ID
for specific job mon

overall statistics
per cloud



PanDA: Production and Distributed Analysis

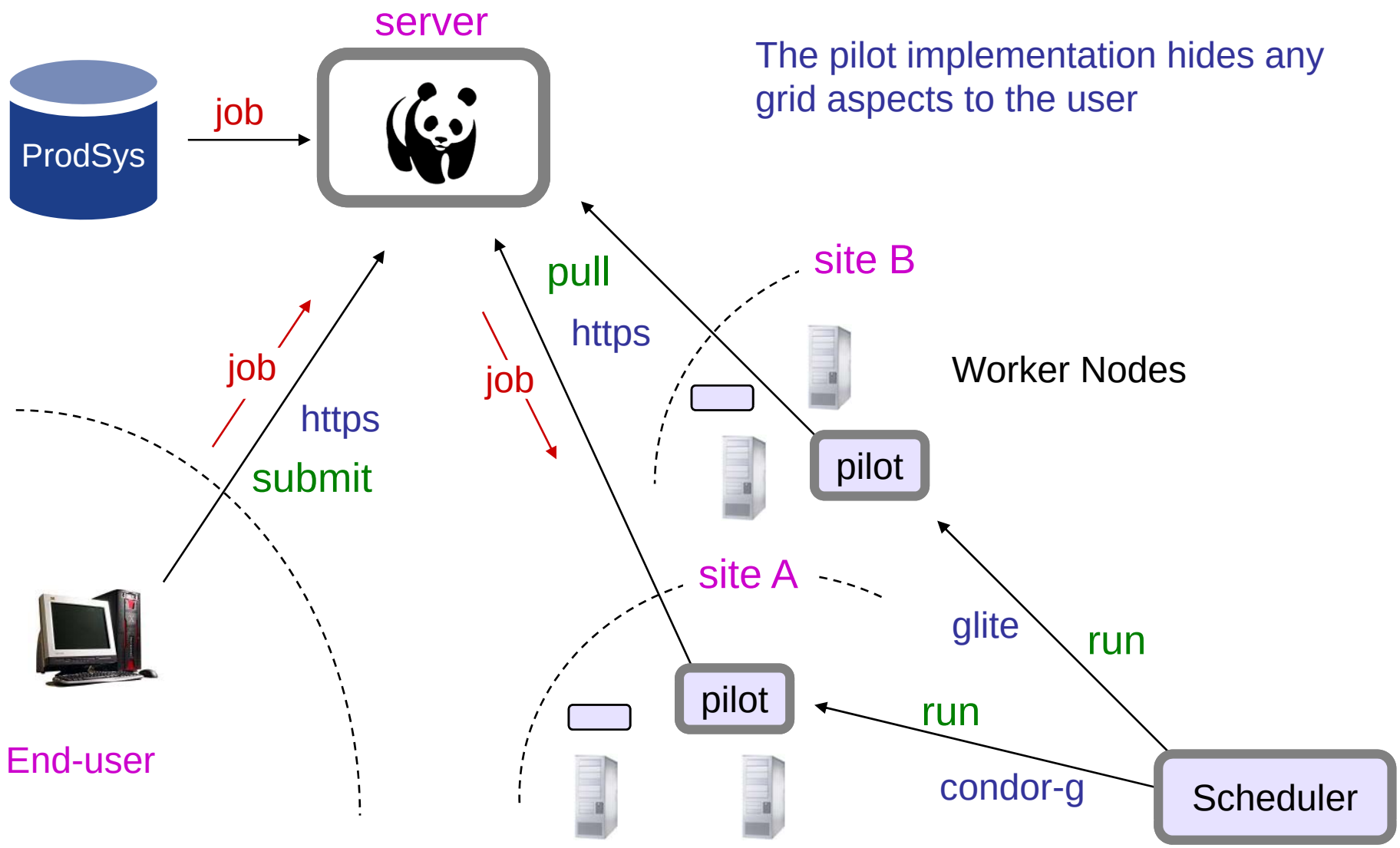
PanDA system developed by ATLAS aug 2005, in production dec 2005

- to meet requirements for data-driven workload mgt sys for prod and analysis
- Works both with OSG and EGEE middleware
- A single task queue and pilots
  - Apache-based central server
  - Pilots retrieve jobs from the server as soon as CPU available, hence low latency

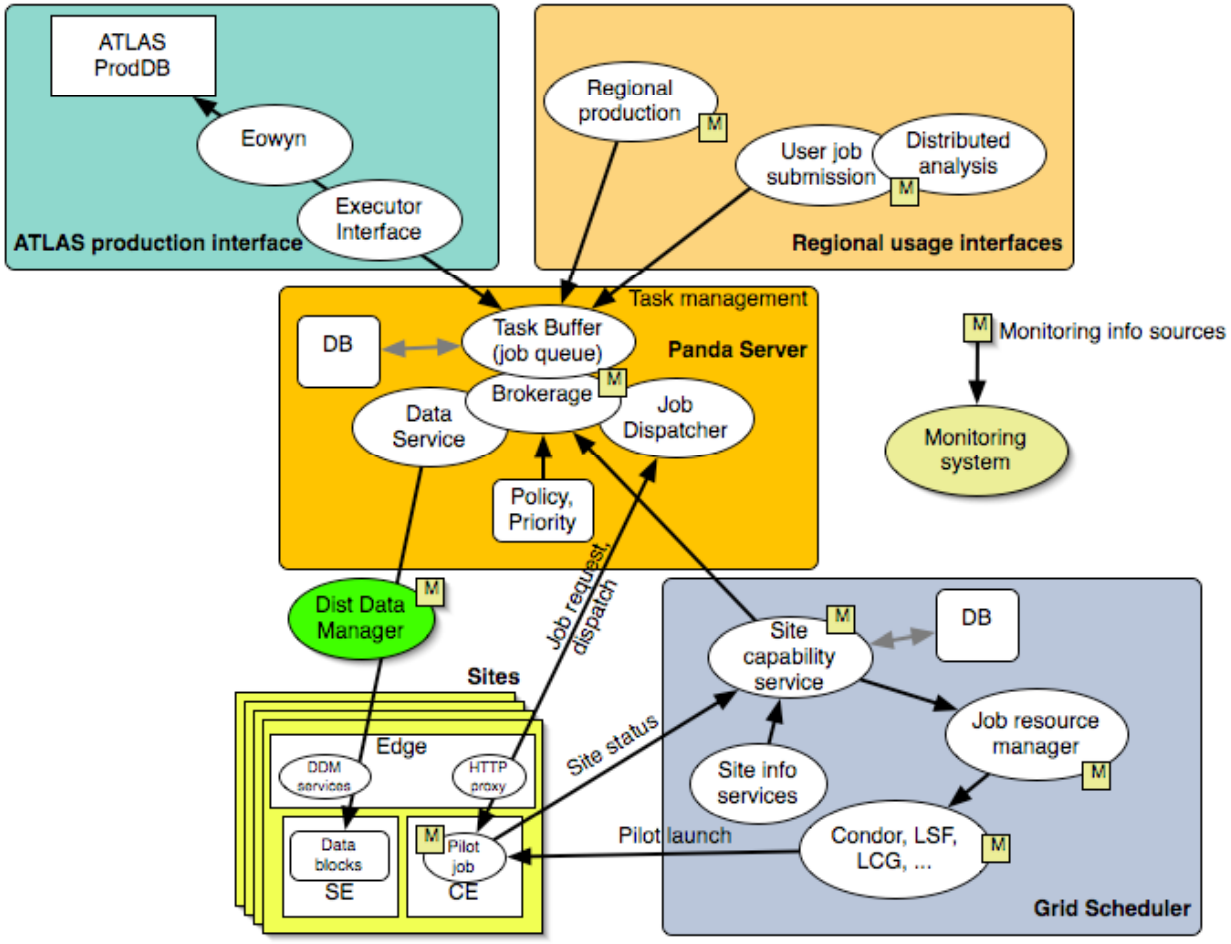
Requirements:

- throughput
- scalability
- robustness
- efficient resource utilization
- minimal operations manpower
- tight integration of data management with processing workflow

PanDA Architecture

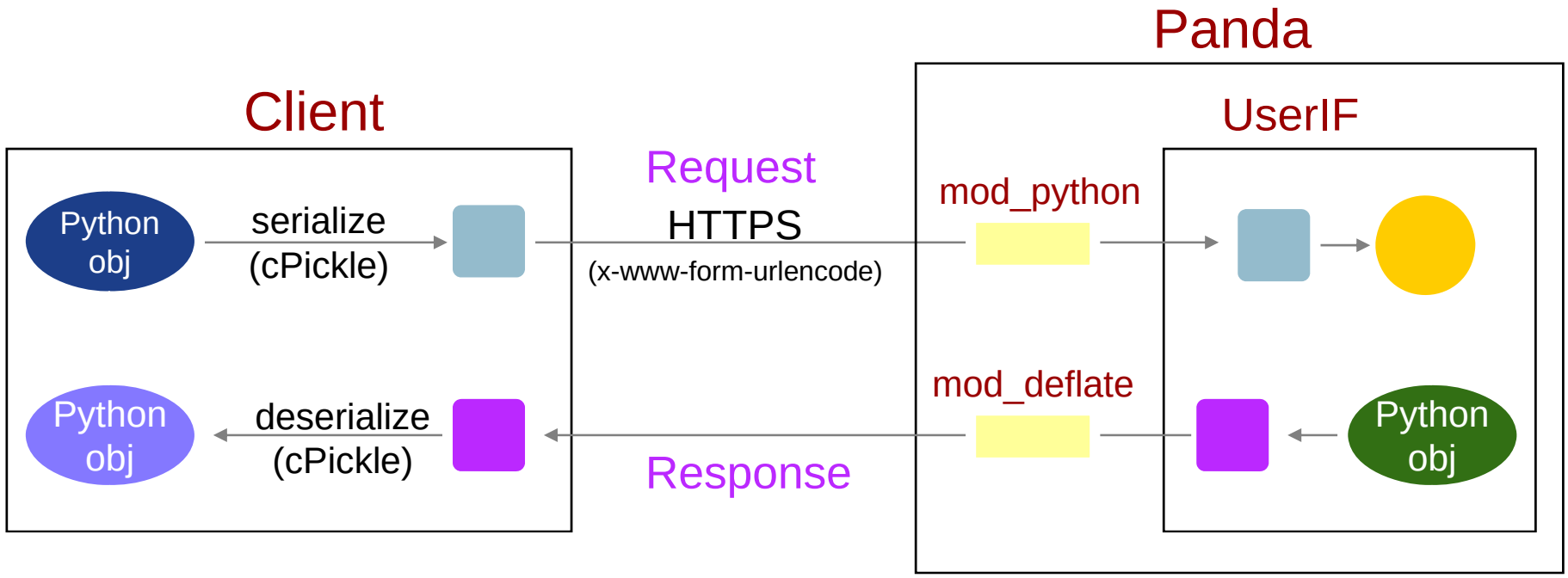


PanDA Architecture



PanDA: client-server communication

- HTTP/S-based communication (curl+grid proxy+python)
- GSI authentication via mod_gridsite
- Most of communications are asynchronous
  - Panda server runs python threads as soon as it receives HTTP requests, and then sends responses back immediately. Threads do heavy procedures (e.g., DB access) in background (hence better throughput)
  - Several are synchronous



PanDA components

Panda Server

- Dispatches jobs to pilots as they request them. HTTPS-based. Connects to central Panda DB

Panda Monitoring Service, Panda Logging Server

- Provides graphic read-only information about Panda function via HTTP. Connects to central PandaDB
- Log Panda Server Events into the Panda DB

Autopilot submission systems (local pilots and global pilots)

- Using Condor-g/site gatekeepers to fill sites with pilots
- Using local condor batch system to fill sites with pilots
- Using gLite WMS to fill sites with pilots

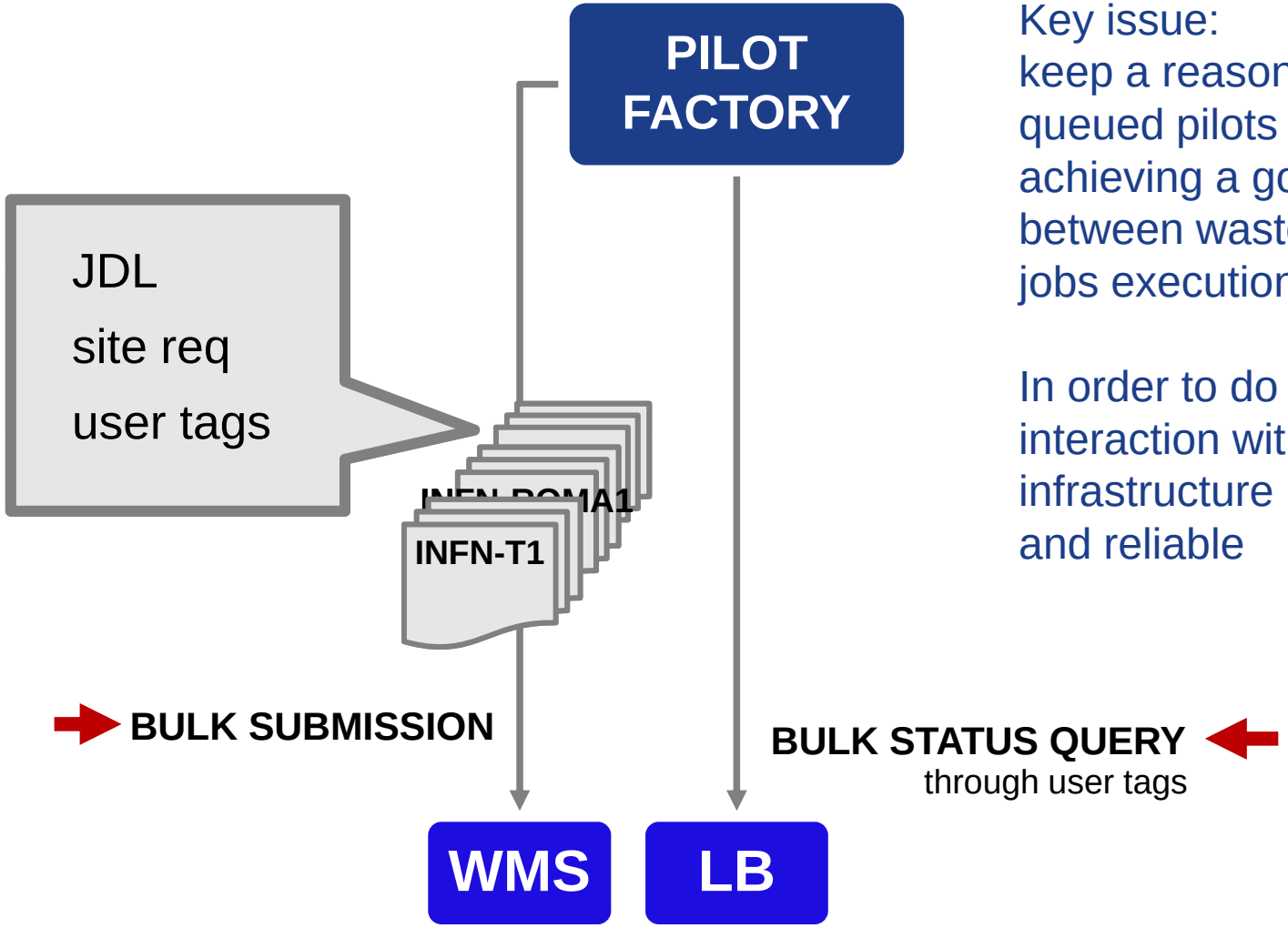
Panda Pilot Wrapper Code Distributor

- Subversion with Web front-end
- Dynamically download pilot wrapper script from the Subversion web cache

PanDA design and implementation

- Support for managed production and user analysis
- Coherent, homogeneous processing system layered over diverse resources
- Pilot submission through CondorG, local batch or gLite WMS
- Use of pilot jobs for acquisition of resources. Workload jobs assigned to successfully activated pilots based on Panda-managed brokerage criteria
- System-wide site/queue information database
- Integrated data management relying on dataset-based organization of data (integrated with DDM)
- Support for running arbitrary user code
- Comprehensive monitoring system supporting production and analysis operations
- the idea is to move towards site-local pilot schedulers using Condor glideins

PanDA pilot submission using gLite WMS

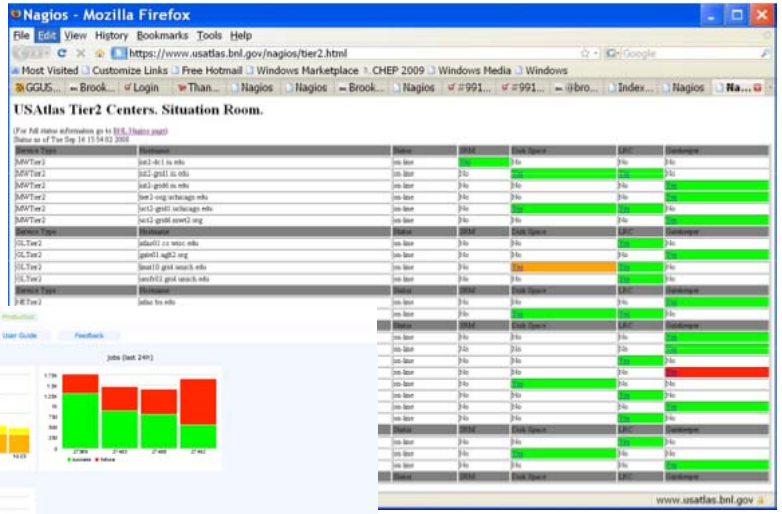


Key issue:
keep a reasonable number of queued pilots in every site, achieving a good trade-off between wasted CPU time and jobs execution latency

In order to do that, the interaction with the submission infrastructure has to be efficient and reliable

PanDA monitoring

PanDA mainly relies on a Nagios-based monitoring system and it's now also well integrated in the ARDA dashboard system



The pilot factory submitting thru the gLite WMS, as well, is integrated in the ARDA dashboard (through API)

GANGA

Ganga

Started as a LHCb/ATLAS project

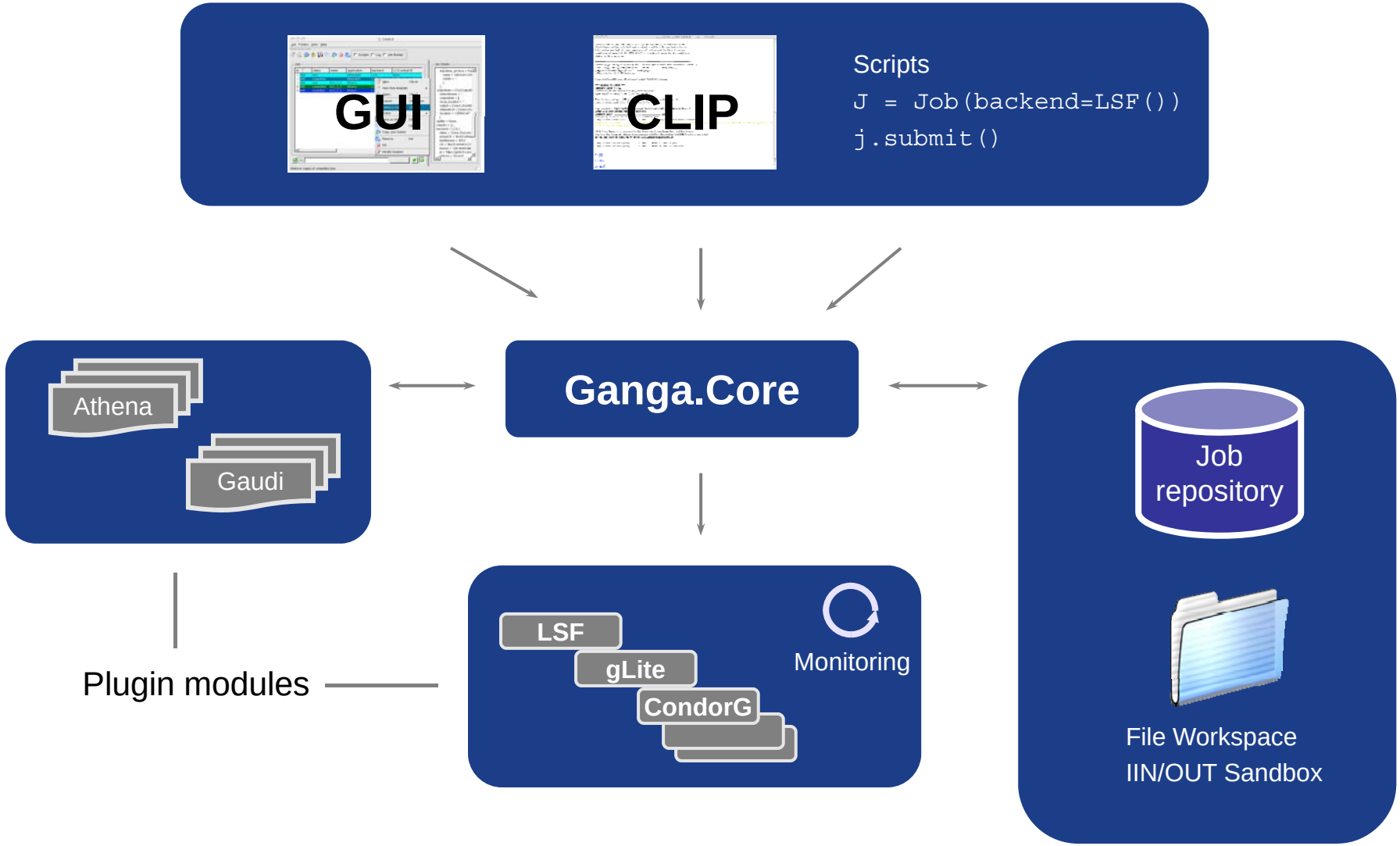
Ganga is an application to enable a user to perform the complete lifecycle of a job:

Build – Configure – Prepare – Monitor – Submit – Merge – Plot

Jobs can be submitted to

- The local machine (interactive or in background)
- Batch systems (LSF, PBS, SGE, Condor)
- Grid systems (LCG, gLite, NorduGrid)
- Production systems (Dirac, Panda)
- Jobs look the same whether the run locally or on the Grid

Ganga architecture



Ganga structure

ATLAS Applications:

- Athena: Analysis: athena.py jobOptions input.py
- AthenaMC: Wrapper for Production system transformations

Data input:

- DQ2Dataset: all DQ2 dataset handling in client, LFC/SE interaction on worker node, used by all backends
- ATLASDataset: LFC file access
- ATLASLocalDataset: local file system, Local/Batch backend

Data output:

- DQ2OutputDataset: stores files on Grid SE, registration in DQ2
- AtlasOutputDataset: multi-purpose for Grid and Local output

Splitter:

- DQ2JobSplitter: intelligent splitter, uses site-index/tracker, knows file locations and dataset replicas, fine for incomplete and complete datasets
- AthenaSplitterJob: legacy splitter, knows only dataset replicas, good for complete datasets

Ganga structure

Merger:

- DQ2OutputDataset, AtlasOutputDataset: users downloads outputfiles and merges on local disk

GangaTNT:

- Tag Navigator Tool: access to TAG database, e.g. AOD skimming

Tasks:

- AnaTask, MCTask: automate job (re-)submission and job chaining on LCG

Backends:

- LCG (glite WMS/LCG RB), all above supported
- NG (ARC): Analysis and DQ2 supported
- Panda: Analysis and DQ2 supported

Ganga generic:

- GUI, Executable, Root

Ganga structure

Ganga is based on python and has an enhanced python prompt (Ipython):

Python programming/scripting

```
myvariable = 5
print myvariable*10
```

Easy access to the shell commands

```
!less ~/.gangarc      # personal config file
!pwd
```

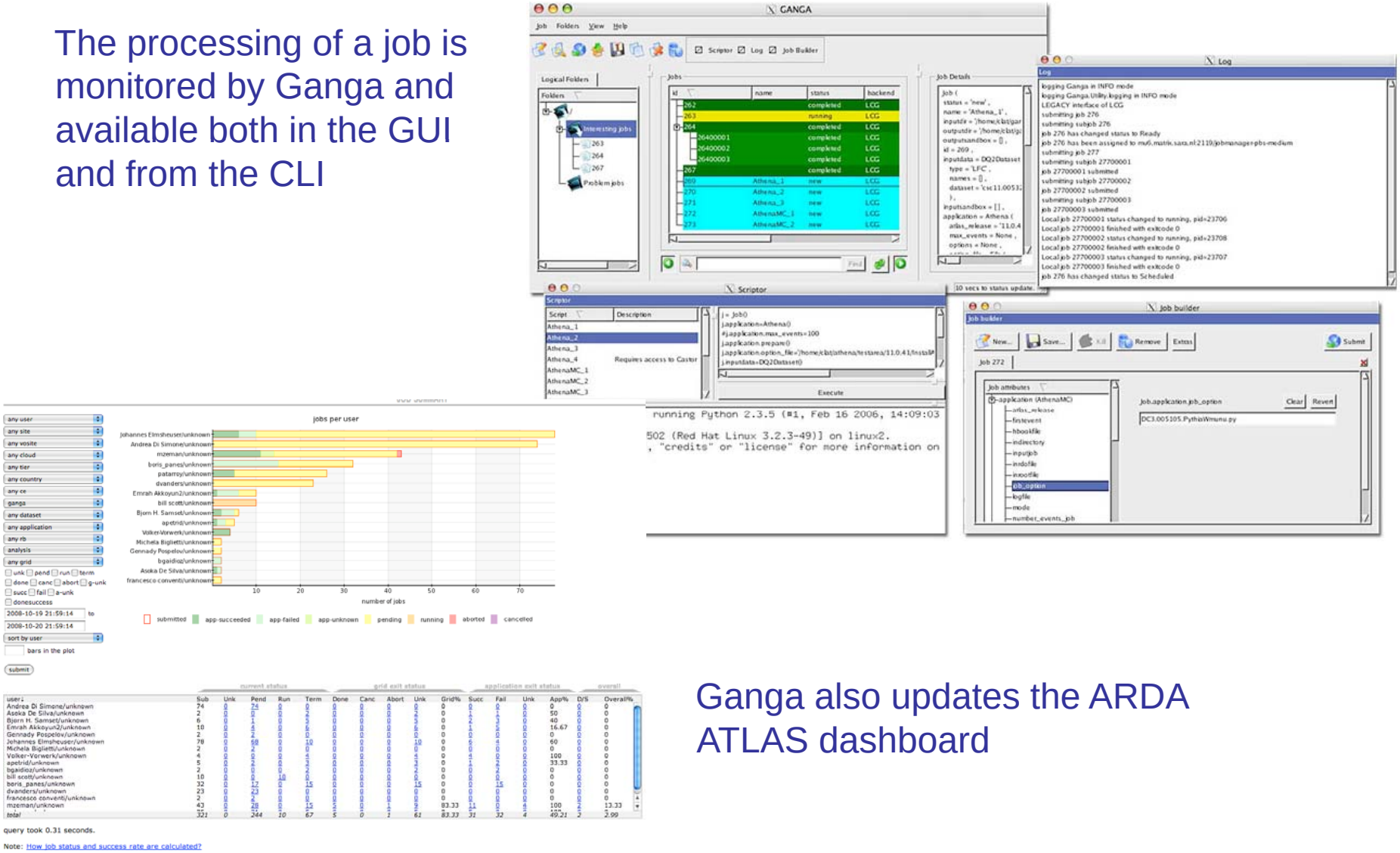
History <arrow up>, Search (Ctrl-r)

TAB completion works on keywords, variables, objects

Availability of all Python modules plus builtin methods and objects (as the job object)

Ganga monitoring

The processing of a job is monitored by Ganga and available both in the GUI and from the CLI



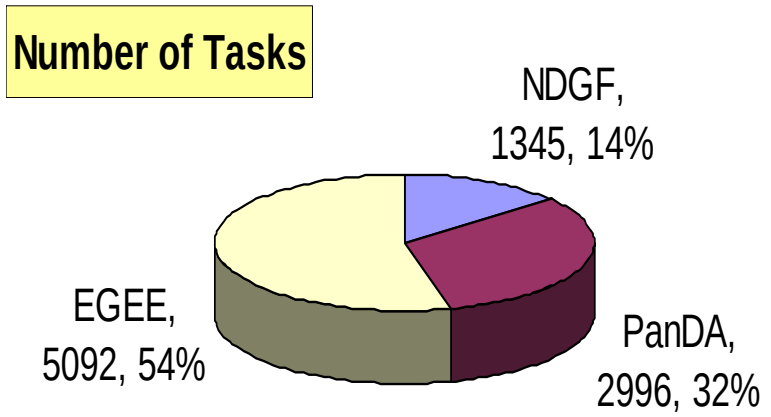
Ganga also updates the ARDA ATLAS dashboard

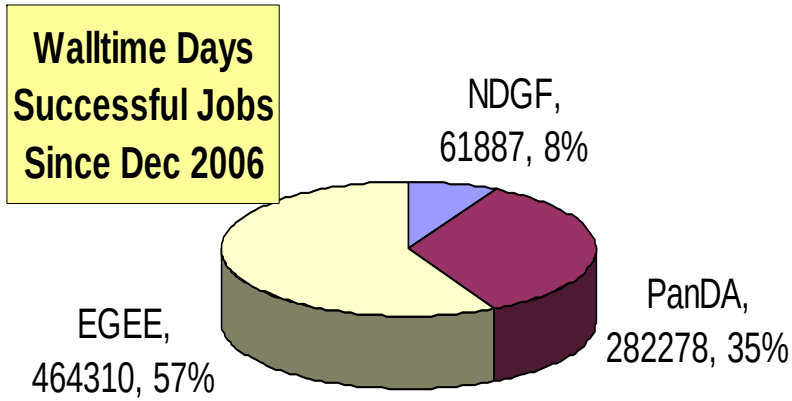
Back up

PanDA/Ganga statistics

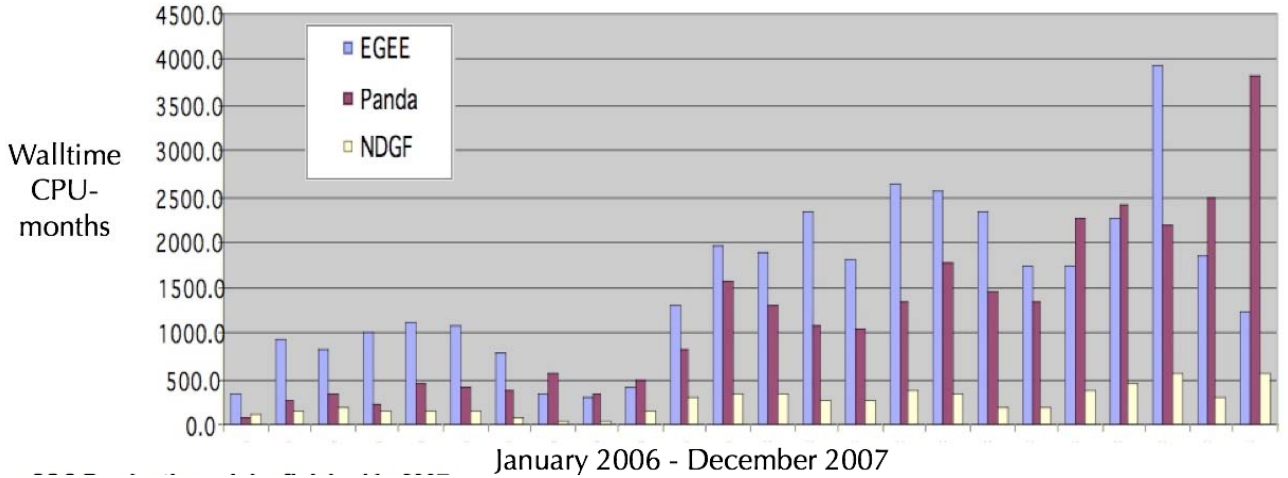
PanDA (US, Canada) production statistics

- Many hundreds of physics processes have been simulated
- Tens of thousands of tasks spanning two major releases
- Dozens of sub-releases (about every three weeks) have been tested and validated
- Thousands of 'bug reports' fed back to software and physics
- 50M+ events done from CSC12
- >300 TB of MC data on disk





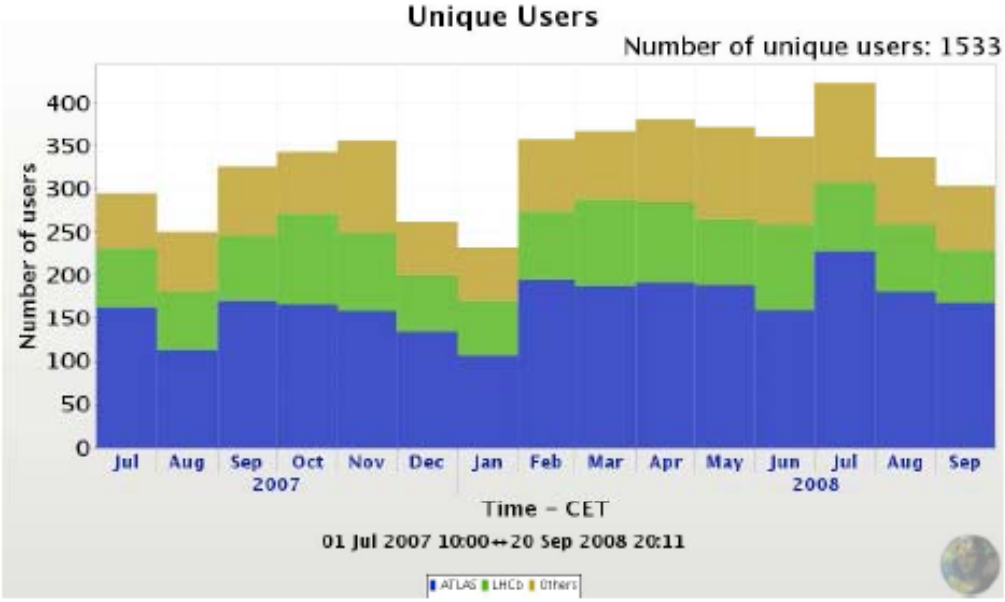
PanDA statistics



- Roll-out across ATLAS began fall 07
- Today PanDA production spans all 10 ATLAS regions
- almost 200 sites/queues
- 10k-18k concurrent jobs
- Nordic grid (NDGF) just coming online
- Italy ramping up; served by distinct (CERN) PanDA system instance (using gLite WMS for submission)

Ganga stats

- Ganga has been used by over 1500 users in total
- approx. 70 ATLAS Ganga users per week and similar for pathena



Distributed Data Management

Distributed Data Management

DDM project estd Spring 2005

Aim: develop **DonQuijote 2** (**DQ2**)

Goals:

- scalability
- robustness
- flexibility
- Grid interoperability (wLCG: OSG, EGEE and NG)

DQ2 is the primary responsible for bookkeeping of file-based data.

Responsibilities of DDM:

- provide functionality for bookkeeping all file-based data
- manage movement of data across sites
- enforce access controls and manage user quotas

Distributed Data Management

Expected volume of data, just from the detector
assuming 50k seconds@200Hz per day:
16TB of RAW
10TB of ESD
2 TB of AOD

Size per event	
RAW	1.6MB
ESD	1MB
AOD	0.2MB
TAG	0.01MB
DPD	0.02MB

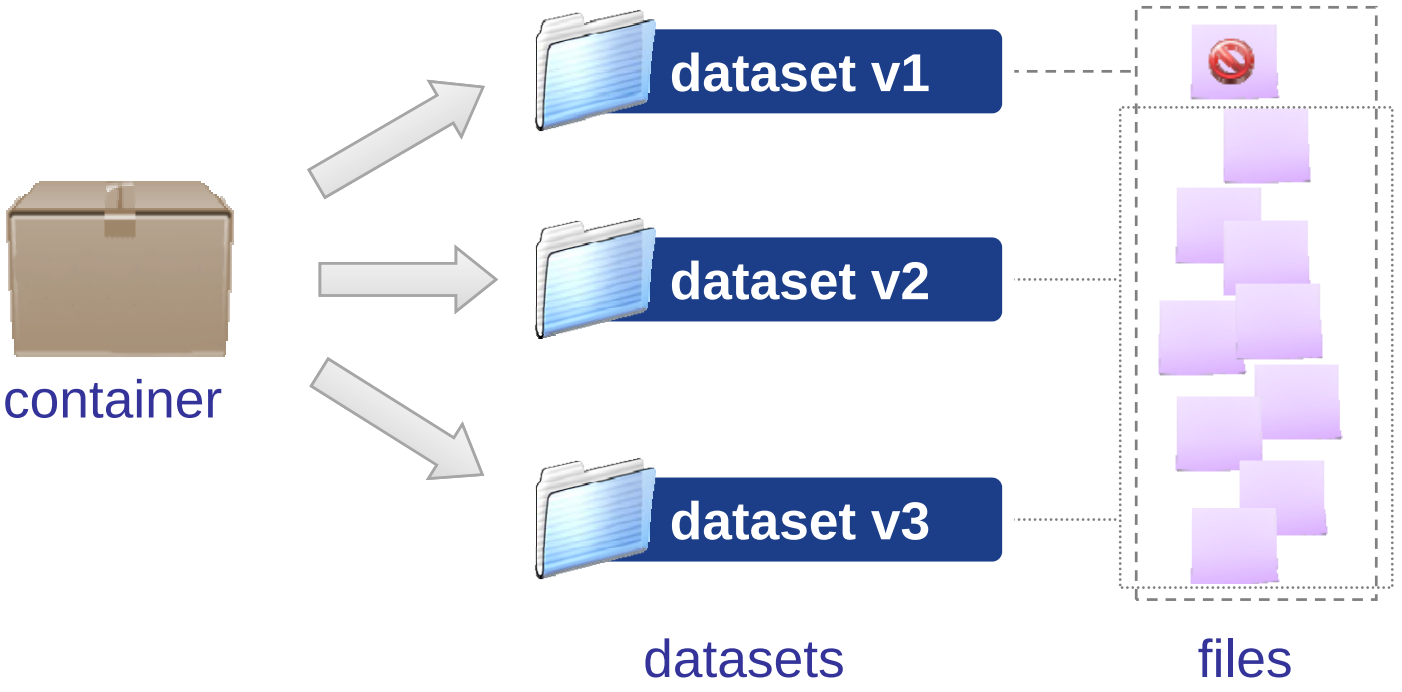
To be moved:
~20 PB/year (including reproc and user-generated data)

O(4000)/day new datasets from Trigger and Data Acquisition (TDAQ)
O(1500)/day new ds from MonteCarlo
O(100) files per dataset
O(100) sites providing storage for ATLAS as dataset locations
O(100) data-transfers at any moment per site
O(100) ATLAS users and O(10) groups

DDM: data organization

The primary concept of the DDM system is the dataset: aggregation of data (in one or more files) processed together and used collectively as input/output of a computation process

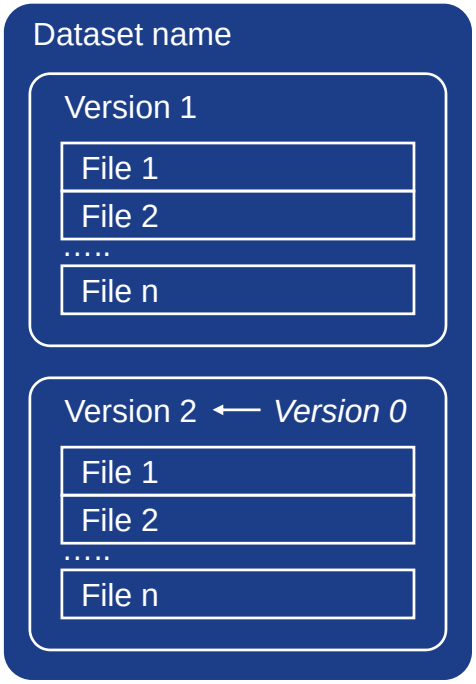
Datasets can be organized in containers



DDM: datasets

Advantages

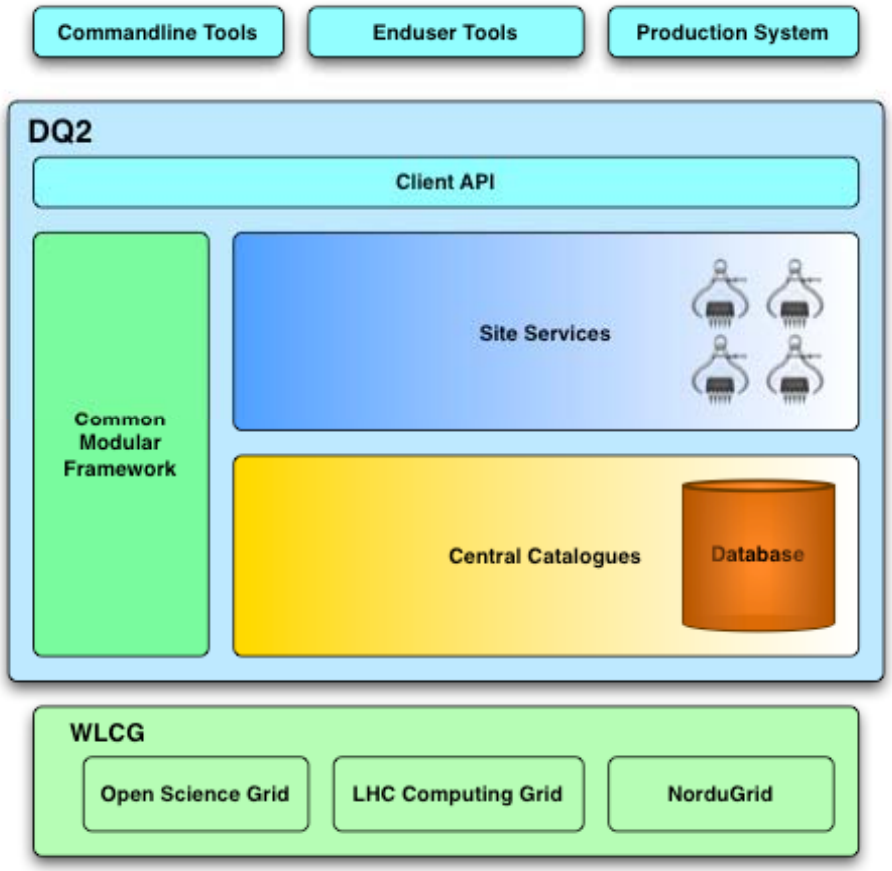
- granularity - primary grouping for data
- scalability - unit of data to be transferred



Properties

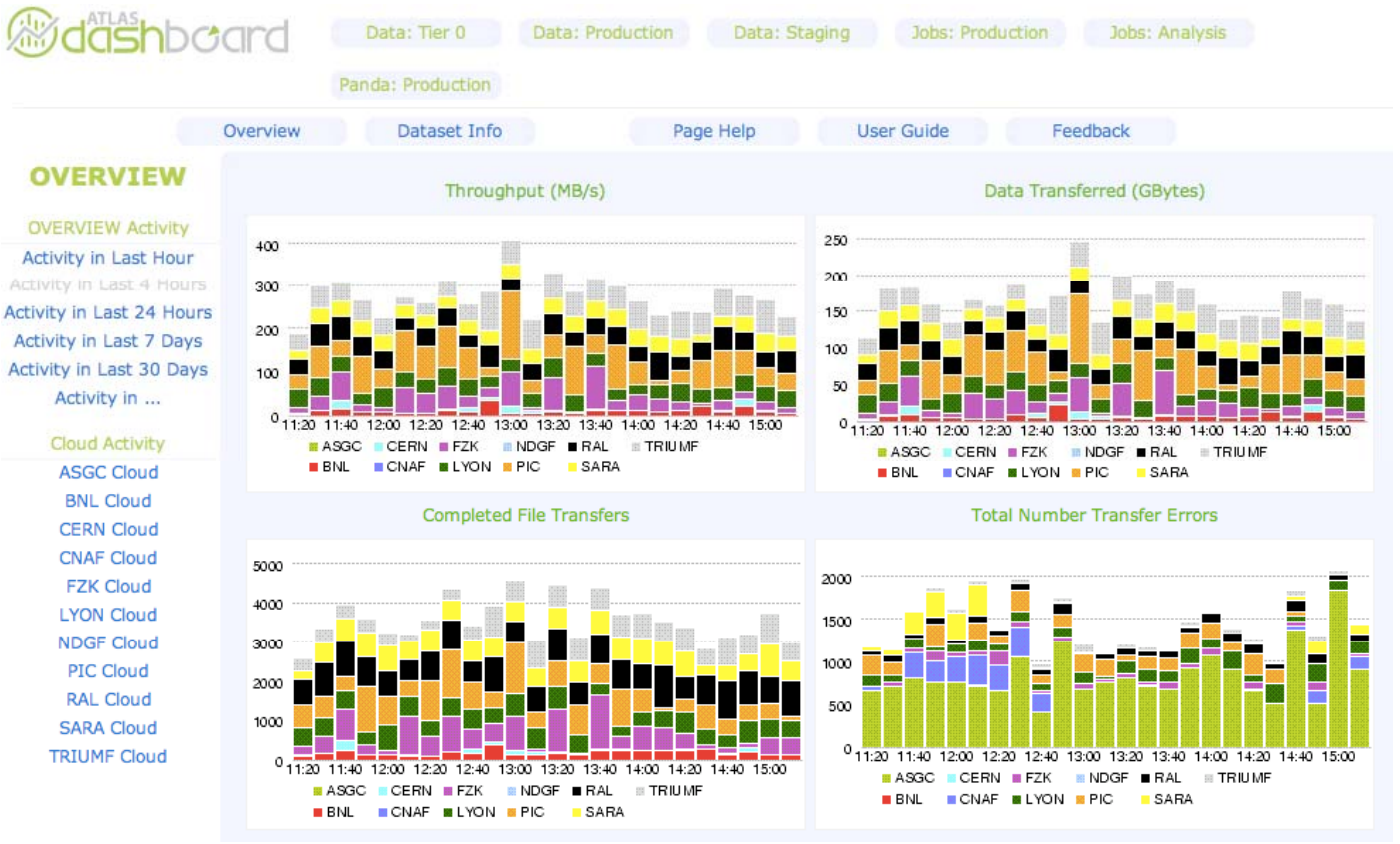
- versioning: modifications (add/removal of files) leads to new version
- immutability: status open (add/remove files), closed (create new versions), frozen (latest version closed and no more versions allowed)

DDM system architecture



DDM monitoring

DDM is perfectly integrated with the ARDA dashboard project
Subscriptions and file transfers can be monitored through the dashboard, for all different spacetokens and clouds

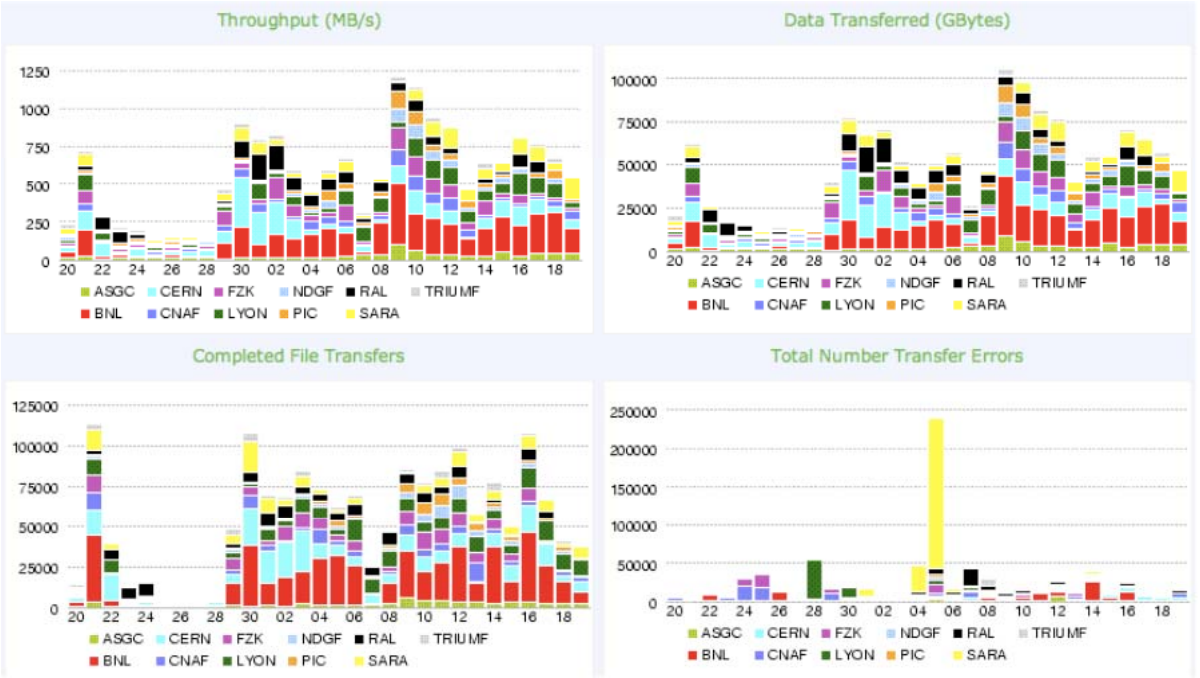


DDM: the cosmics experience

Cosmics data taking:

- 24h x 7d
- Bigger RAW than expected (Lar samplings, from 3 to 9 MB, 1MB ESDs)

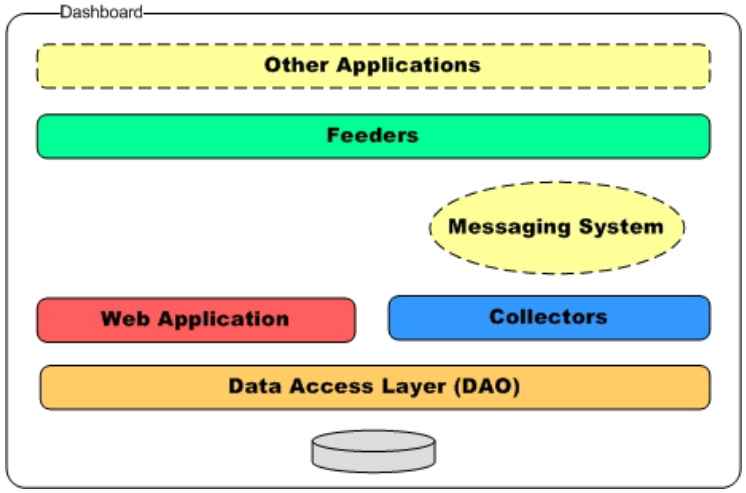
Data transfer from 20th Sept to 19th Oct



Dashboard

Dashboard concept

Collect, store and expose to users information coming from different sources.



Other Applications and *Messaging System* are not part of the system implementation, but external components that make use of the dashboard and that the system uses to achieve its own goals

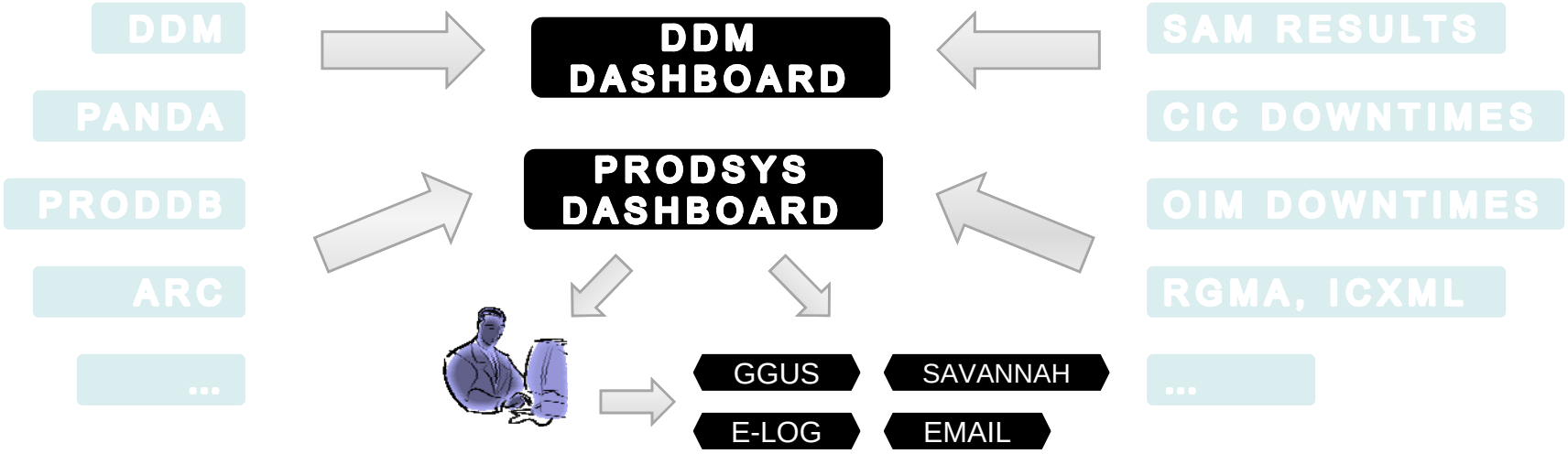
- *Data Access Layer (DAO)*: components responsible for the management of the persistent data (generally stored in database
- *Web Application*: responsible for the HTTP entry point to the data available in the dashboard, exposes data to users. Uses *DAO* as an abstraction (to avoid eed for knowledge of the underlying storage)
- *Collectors:* all components that listen to messages/events coming from the *Messaging Infrastructure* (typically put there by the *Feeders*), analyze data to be be passed to the *DAO*
- *Feeders:* components that listen to interesting events and put them in the *Messaging Infrastructure* to be taken by *Collectors*

Dashboard applications

- **All applications built on top of the dashboard framework**
  - Build and testing environment, persistent data access, messaging APIs, command line tools, agent management, plotting libraries, multiple output formats (CSV / XML / RSS / …)
  - Some of these packages have been taken in ATLAS for other uses (build, messaging APIs, cli tools, agent management)
- **Some are generic Experiment Dashboards**
  - As seen also in other experiments, with minor additions
- **But others are very much ATLAS specific**
  - Developed in close collaboration with ATLAS application providers

Dashboard status

- Features mainly driven by shifter's needs
  - With many additional features filling other use cases (e.g. overview plots for managers, many historic summaries)
- Integration with ATLAS and GRID operations tools
  - Both as input (CIC portal, SAM, BDII, …) and output (GGUS, Savannah, e-Logs, …)
- Critical tools with extensive use in the ATLAS shifters effort (24/7)



Dashboard for DDM

Monitoring of data movement within clouds and individual sites

Clouds

DDM

STATS GENERATION

NOTIFICATIONS

DOWNTIME RET.

...

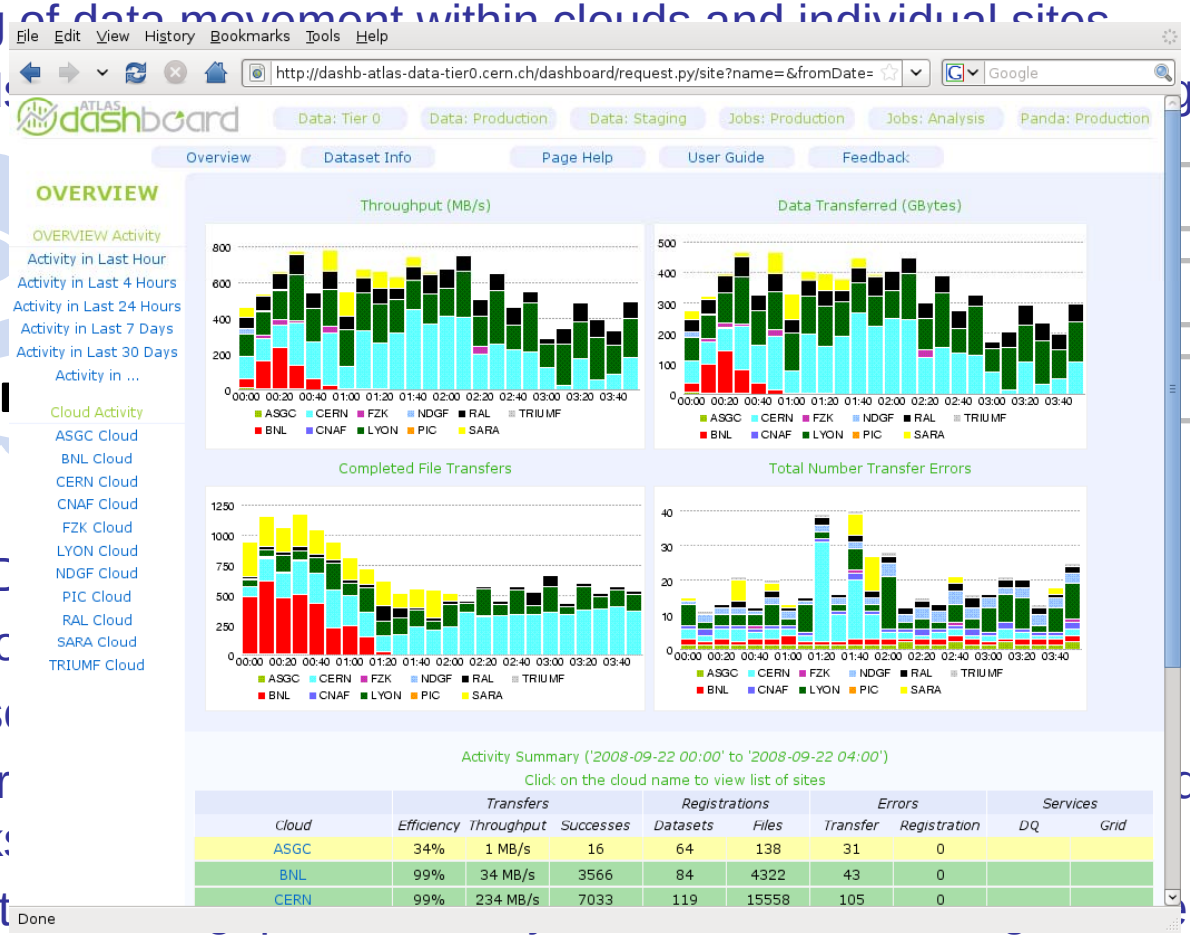
Available D

Topolc

Datas

File: tr dest surl, checks

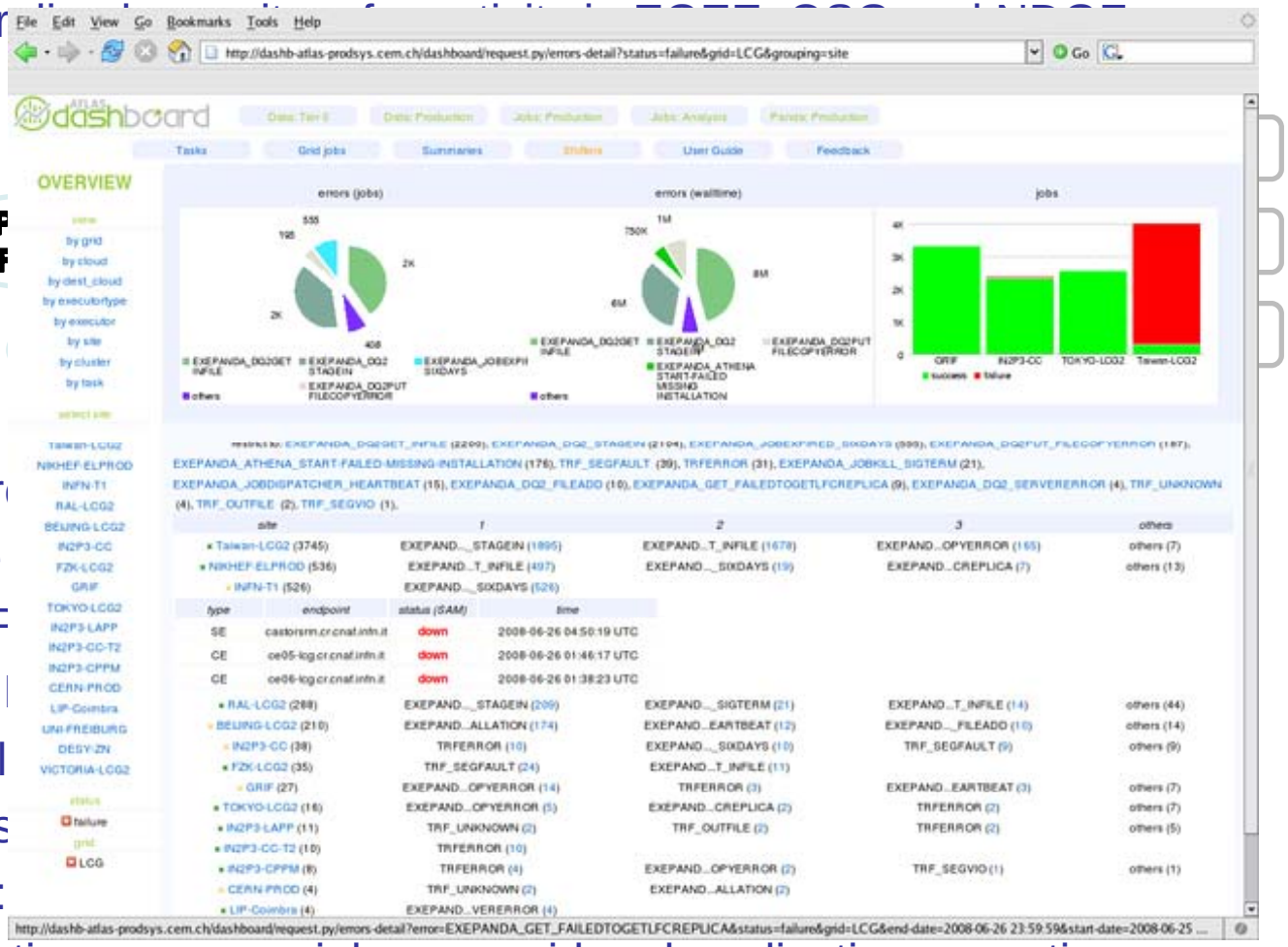
Statist r attempt number, dataset queued/completion time, …)



Dashboard for ProdSys

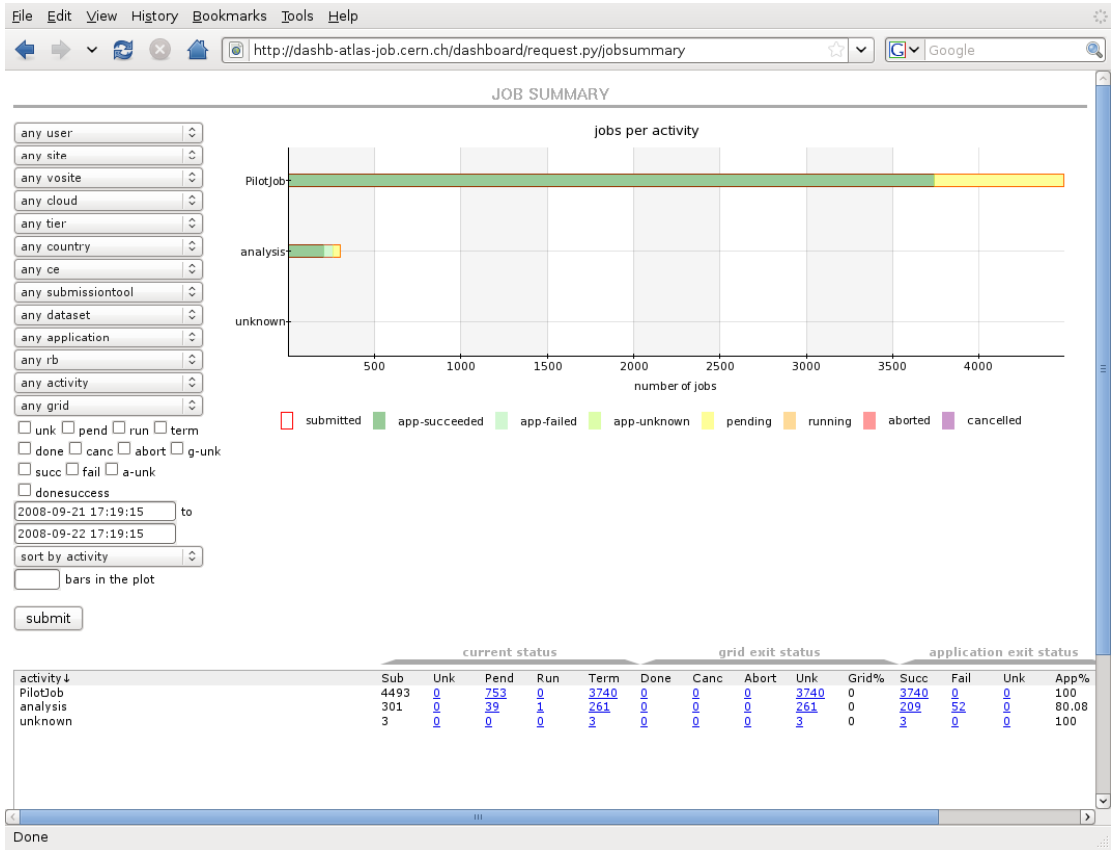
Monitoring of production jobs in all ATLAS grids

- Centr
- Data sour
- More
- F
- Available
- Topol
- Tasks
- Jobs:
- Statistics: progress, jobs run, grid and application execution errors summaries



Generic dashboard

JOB MONITORING



Mostly analysis users

PANDA jobs collected directly from their db

GANGA jobs collected via the messaging API

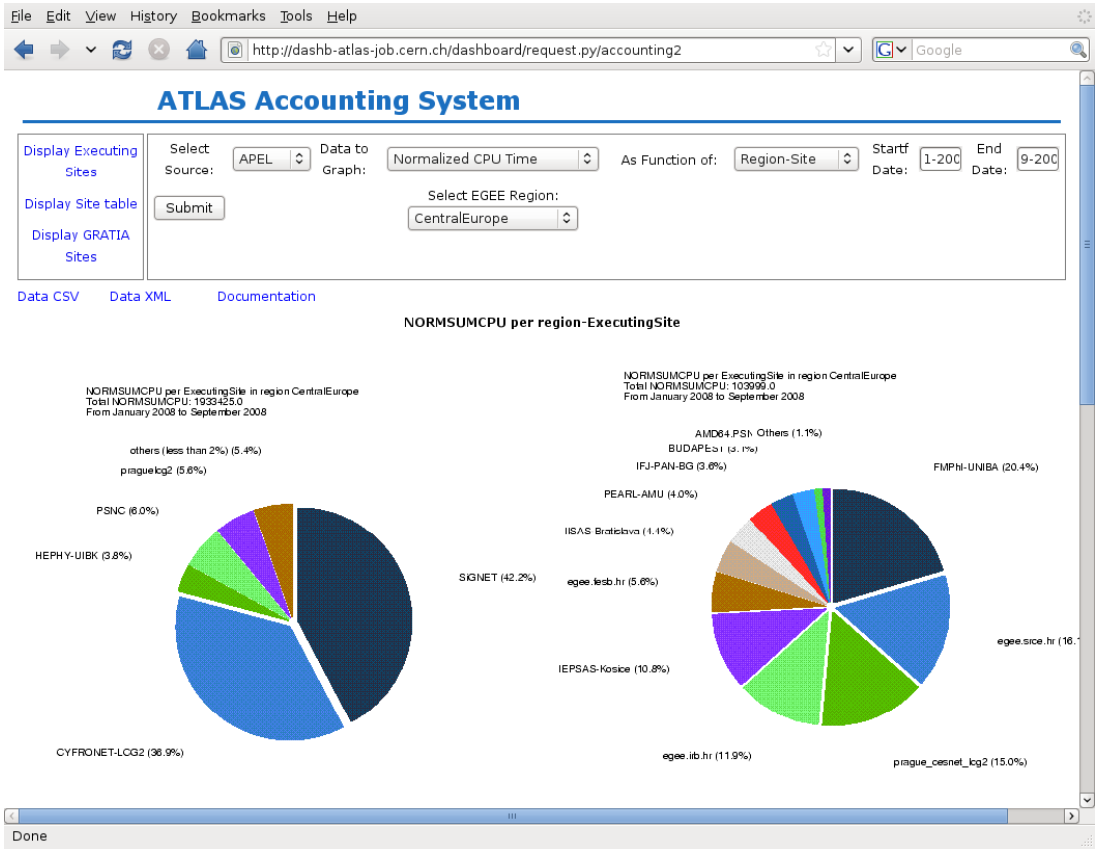
Generic dashboard

ACCOUNTING

Developed by an ATLAS collaborator who joined the dashboard team

Contribution now available to all experiments

Data gathered via APEL and GRATIA



Dashboard for the Tier0

Still in a prototype version (ready very soon).
New view based on javascript/canvas

Monitors jobs, Castor and AFS pools usage, DQ2 and AMI registration backlogs

DAO acting on the very production database