Messaging Services for ATLAS Offline Data Quality

Peter Onyisi *on behalf of the ATLAS Collaboration*

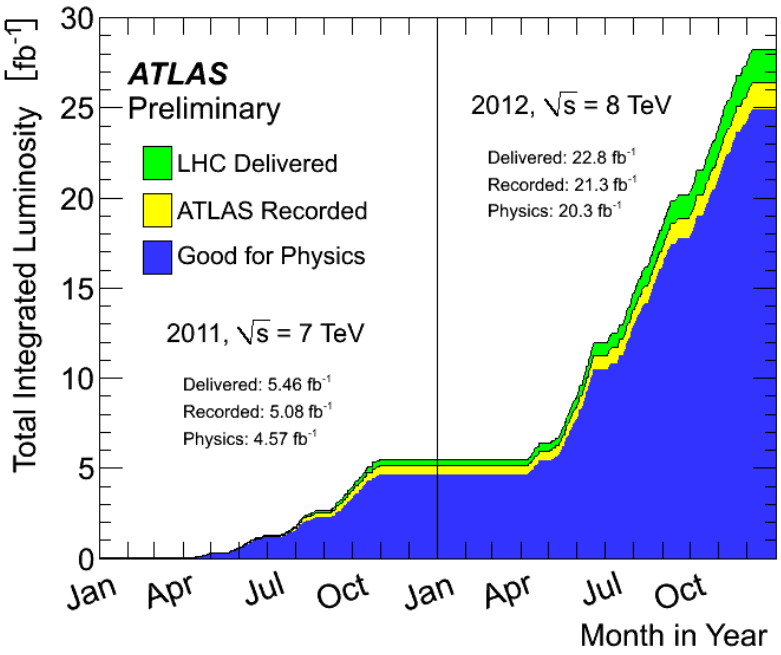
CHEP, 14 Apr 2015

TEXAS

The University of Texas at Austin

ATLAS DQ Performance

- Data Quality (DQ) system allows identification & flagging of detector problems
  - and, in the longer run, prevention/mitigation
- Fast turnaround, quick understanding of problems needed for rapid physics analysis!

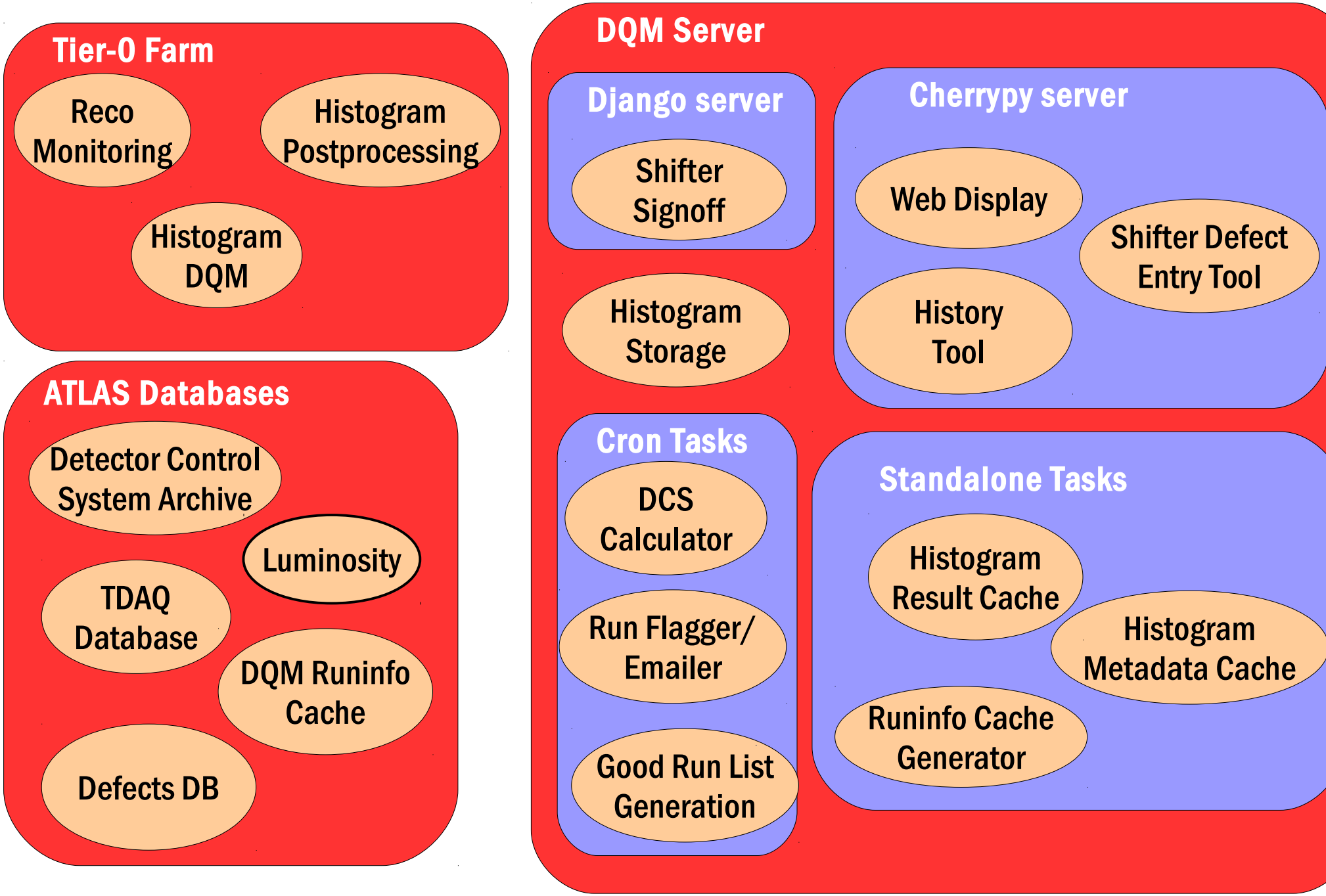


ATLAS p-p run: April-December 2012										
Inner Tracker			Calorimeters		Muon Spectrometer				Magnets	
Pixel	SCT	TRT	LAr	Tile	MDT	RPC	CSC	TGC	Solenoid	Toroid
99.9	99.1	99.8	99.1	99.6	99.6	99.8	100.	99.6	99.8	99.5
All good for physics: 95.5%										

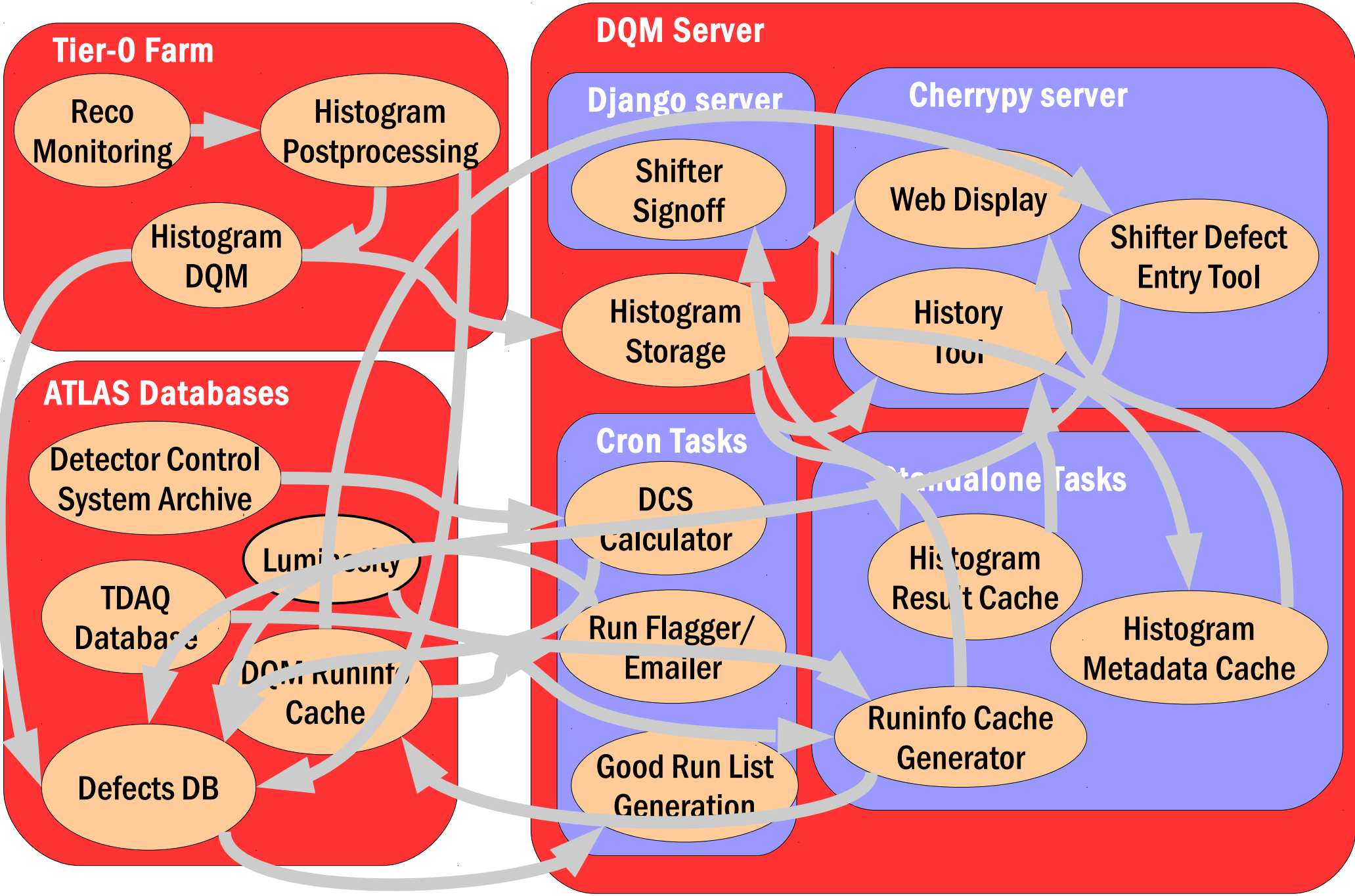
Luminosity weighted relative detector uptime and good quality data delivery during 2012 stable beams in pp collisions at √s=8 TeV between April 4th and December 6th (in %) – corresponding to 21.3 fb-1 of recorded data.

Very high efficiency for good data!

ATLAS Offline DQM



ATLAS Offline DQM



Program Coupling

Programs need to cooperate!

- New run finished?
  - Mark run as unexamined by shifters
  - Prepare signoff database to receive comments
  - Run DCS Calculator on detector status information
- Tier-0 reconstruction/histogram processing complete?
  - Remove intermediate histograms
  - Upload history results
- Run period closed?
  - Create good run list
- etc…

+ many clients outside of central DQ infrastructure

Distributed system (across nodes, programs, containers) discourages/forbids strong coupling

Run 1: rendezvous via polling disk files or database records

Serious problem with virtualized servers none of the Linux file notification systems really work for us

Example: Histogram Web Interface

Tier 0 Processing DQ Monitoring

Please consider trying out the development server!

Change source: Tier 0 Processing

Display results from last 100 monitored runs, or runs to

Include non-stable-beam runs

Submit

***Indicates reconstruction is in progress; histograms represent accumulated statistics and are temporary.
If you wonder why you don't see the runs you are looking for, try checking "Include non-stable-beam runs" checkbox above

For extra information on the run, hover the mouse pointer over the run number.

Run Number	Iteration	Streams
261182	ES1: x313_h16	[express_express***] [physics_CosmicCalo***] [physics_IDCosmic***]
261142	ES1: x313_h16	[express_express] [physics_CosmicCalo] [physics_IDCosmic]
261141	ES1: x313_h16	[express_express] [physics_CosmicCalo] [physics_IDCosmic]
261140	ES1: x313_h16	[express_express] [physics_CosmicCalo] [physics_IDCosmic]
261070	BLK: f568_h17	[physics_MinBias***]
	ES1: x313_h16	[express_express] [physics_CosmicCalo] [physics_IDCosmic]
260758	BLK: f567_h16	[physics_CosmicCalo] [physics_CosmicMuons] [physics_IDCosmic] [physics_MinBias]
	ES1: x313_h16	[physics_CosmicCalo] [physics_IDCosmic]
260679	BLK: f567_h16	[express_express] [physics_CosmicMuons] [physics_MinBias]
	ES1: x313_h16	[express_express]
260670	BLK: f567_h16	[express_express] [physics_CosmicMuons] [physics_MinBias]
	ES1: x313_h16	[express_express]
260658	BLK: f567_h16	[express_express] [physics_CosmicCalo] [physics_CosmicMuons] [physics_IDCosmic] [physics_MinBias]
	ES1: x313_h16	[express_express] [physics_CosmicCalo] [physics_IDCosmic]

In progress reco: updates frequently need to clear caches & remove temp files when reco is done

run information: keep cache up to date

reconstruction configuration: cache run/reco metadata

Error Logging

- Many disparate logging systems
- Generally *no error notification* – only able to track down errors once noticed externally
(shifter/first pass reconstruction [Tier-0] ops: "something is wrong")

Error notification:
Tier-0 postprocessing

Logging:
Standalone tasks: supervisord logs
Django services: Django logs
Cherrypy services: Cherrypy logs
Cron tasks: text files, emails
Tier-0 tasks: Tier-0 logging

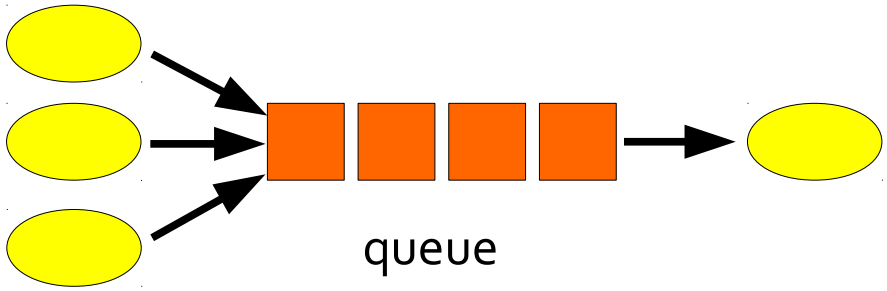
Goals:

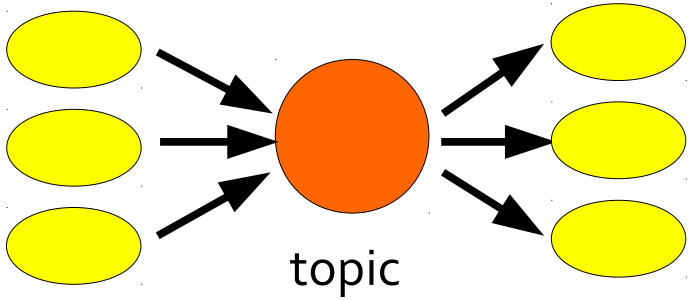
- Provide *simple to use* notification for serious errors (`panic("A serious error has happened")` → on call expert is notified by email/SMS/etc.)
- Collect logfiles centrally

Simplicity is essential requirement
we want non-experts to use the system

Solution: Messaging Queues

- Processes communicate by *messages* sent via a *broker*
  - payload format generally up to user, we use JSON
- Processes do not need to know about each other directly. Instead they publish to/listen to abstract *queues* and *topics*
  - Queue: like a letter: messages delivered to one reader
  - Topic: like a town crier: messages delivered to whatever process is listening
- Processes can be separated in *space* (different nodes) and in *time* (message sending and delivery are asynchronous for queues)





Messaging Queues at CERN

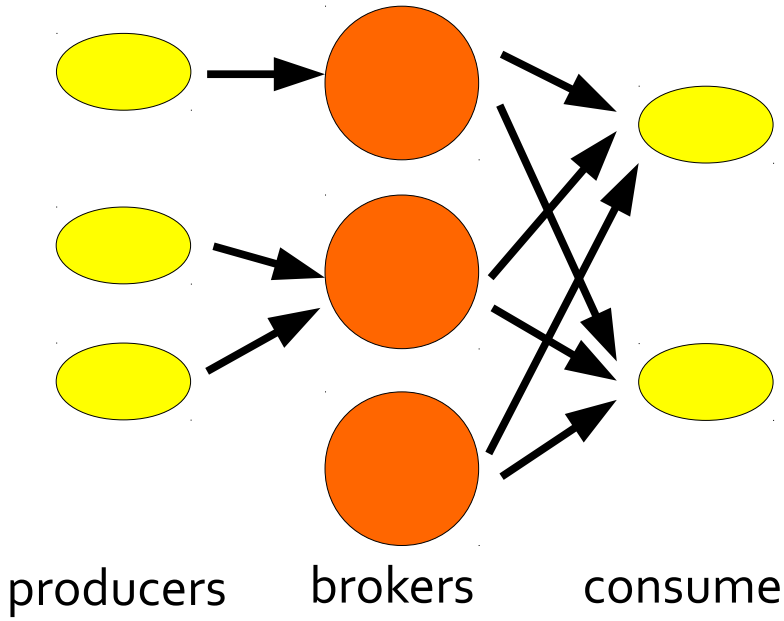
ØMQ in bad location on complexity/benefit curve

Would need to support RabbitMQ ourselves

- Many options for messaging brokers
- CERN has standardized on **ActiveMQ**
  - Wrinkle: doesn't integrate at all with standard CERN auth mechanisms (must use app-specific passwords or certificates)
  - for security reasons, creating queues requires coordination with CERN IT
- Use **STOMP** protocol
  - near-universal availability of client libraries for different languages
- Piggybacking on servers set up for ATLAS Event Server project
  - message rate of few/minute is negligible perturbation

Server Configuration

- Configuration has multiple load-shared brokers which do not communicate
- Producers can connect to any broker but consumers must connect to all
  - Slightly non-trivial task. Hide in python library used by clients.



Queues & Messages

- Queues:
  - atlas.dqm.panic → Sends panic email with relevant information
  - atlas.dqm.logging → Record in logfile server (future implementation)
- Topic:
  - atlas.dqm.progress → Messages describing task completion
- Message bodies are JSON; content is left up to specific service
  - e.g. Run Start notification: run number, start time, run type …
  - panic: host, command name/args, traceback
- Logistically hard to set up multiple queues; use *selector* mechanism to filter messages for clients
  - Producers can generate arbitrary headers; clients filter on headers using SQL syntax
  - For all messages, define `MsgClass` (set to "DQ") and MsgType (e.g. "RunStart")

Sample Messages

DQ message

```
header: key MsgClass , value DQ
header: key expires , value 1430128977908
header: key ack , value auto
header: key timestamp , value 1428919377908
header: key JMSXUserID , value atlasdqm
header: key destination , value /topic/atlas.dqm.progress
header: key persistent , value true
header: key priority , value 4
header: key MsgType , value RunEnd
header: key message-id , value ID:mb201.cern.ch-28854-1426507888700-1:304485:-1:1:1
header: key type , value textMessage
received message
```

Run end message

Run information

```
 {"runinfo": {"run_type": "LEDandLumat", "det_mask": "0", "sol_current": 7730.0073242187
5, "sol_set_current": 7730.0, "ef_events": 107608, "project_tag": "data_test", "tor_curr
ent": -0.03032509796321392, "lb": 21, "partition": "partition_lucid", "rec_enable": fals
e, "tor_set_current": 0.0, "run_start": 1428918733, "physics_events": -1, "run_end": 142
8919355}, "run": 261222}
```

```
header: key MsgClass , value DQ
header: key expires , value 1430129284434
header: key ack , value auto
header: key timestamp , value 1428919684434
header: key JMSXUserID , value atlasdqm
header: key destination , value /topic/atlas.dqm.progress
header: key persistent , value true
header: key priority , value 4
header: key MsgType , value WebDisplayIncremental
header: key message-id , value ID:mb101.cern.ch-56026-1426587371290-1:293597:-1:1:1
header: key type , value textMessage
received message
```

Incremental DQ histogram merging

Reco/Tier-0 DQM config

```
 {"ami": "x316_h17", "project_tag": "data15_cos", "run": 261070, "stream": "physics_MinB
ias", "hcfg": {"Collisions": {"minutes10": "Collisions/collisions_minutes10.656686.hcfg"
, "run": "Collisions/collisions_run.657167.hcfg", "minutes30": "Collisions/collisions_mi
nutes30.511679.hcfg"}, "Cosmics": {"minutes10": "Cosmics/cosmics_minutes10.656686.hcfg",
 "run": "Cosmics/cosmics_run.657047.hcfg", "minutes30": "Cosmics/cosmics_minutes30.51167
9.hcfg"}, "basename": "/afs/cern.ch/user/a/atlasdqm/dqmdisk/tier0/han_config/", "HeavyIo
ns": {"minutes10": "HeavyIons/heavyions_minutes10.535717.hcfg", "run": "HeavyIons/heavyi
ons_run.538804.hcfg", "minutes30": "HeavyIons/heavyions_minutes30.494913.hcfg"}}, "pass"
: 1}
```

Example Application: Panic Wrapper

- Simple Python wrapper (13 lines) launches arbitrary tasks and sends `panic` message if status code != 0
  - messaging brings central logging of messages, no requirements on node email configuration
- Simple example of implementation

```
#!/usr/bin/env python

import subprocess
import sys
from DataQualityUtils.panic import panic

try:
    subprocess.check_output(sys.argv[1:], stderr=subprocess.STDOUT)
except subprocess.CalledProcessError, e:
    rv = ['Error executing %s' % (' '.join(sys.argv[1:])),
          'Return value: %d' % e.returncode,
          'Output:\n%s' % e.output]
    panic('\n'.join(rv))
```

Services

Implemented:

- Run Start and Run End
- Tier-0 histogram processing status
- Panic queue & email (shared Python module to send messages)

Allows:

- Much better awareness of severe errors
- An end to disk polling for updates

Planned:

- Central log archive
- "DQ Status Board": use Redis key/value store to present state of system

Also expect other applications to join

Detector-specific DQ monitoring
Luminosity monitoring
…

Experience

- It helps a lot that the STOMP Python client is part of LCG distribution!
- Auth system of ActiveMQ somewhat painful
  - requires use of robot certificates (*not* proxies!) or passwords that must be kept secret somewhere …
  - at least with CERN instance, PW auth is sent in the clear (!)
  - from what we understand, RabbitMQ's auth is much more flexible
- Libraries needed to simplify client writing
  - in particular, subscription to multiple servers
- Otherwise, smooth sailing so far – very little "impedance mismatch"
  - e.g. any ATLAS python code can invoke `panic` in 2 lines

Summary

- Distributed nature of ATLAS offline DQM system motivates a unified, robust, loosely coupled IPC system
  - messaging queues fit the bill
- Starting deployment of a system built on CERN's choice of ActiveMQ
  - Python client available as part of LCG distribution, lightweight
- No showstoppers found; system will give us new capabilities

Advancing via *new capabilities*
Improve monitoring, reduce needed personnel