MAUS

Durga Rajaram
Illinois Institute of Technology, Chicago

Janusz Martyniak
Imperial College, London

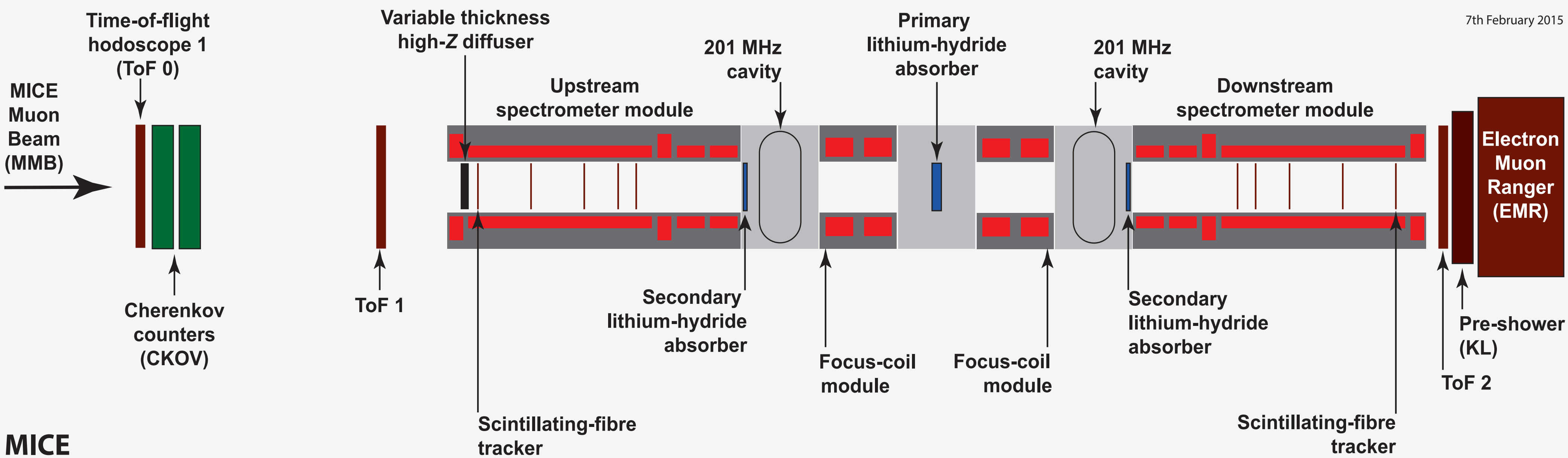
April 13, 2015
CHEP 2015
Okinawa, Japan

OUTLINE

- MICE Overview
- Software Aims
- MAUS
  – Design
  – Framework
- MAUS Online
- Performance
- Conclusions

Muon Ionization Cooling Experiment

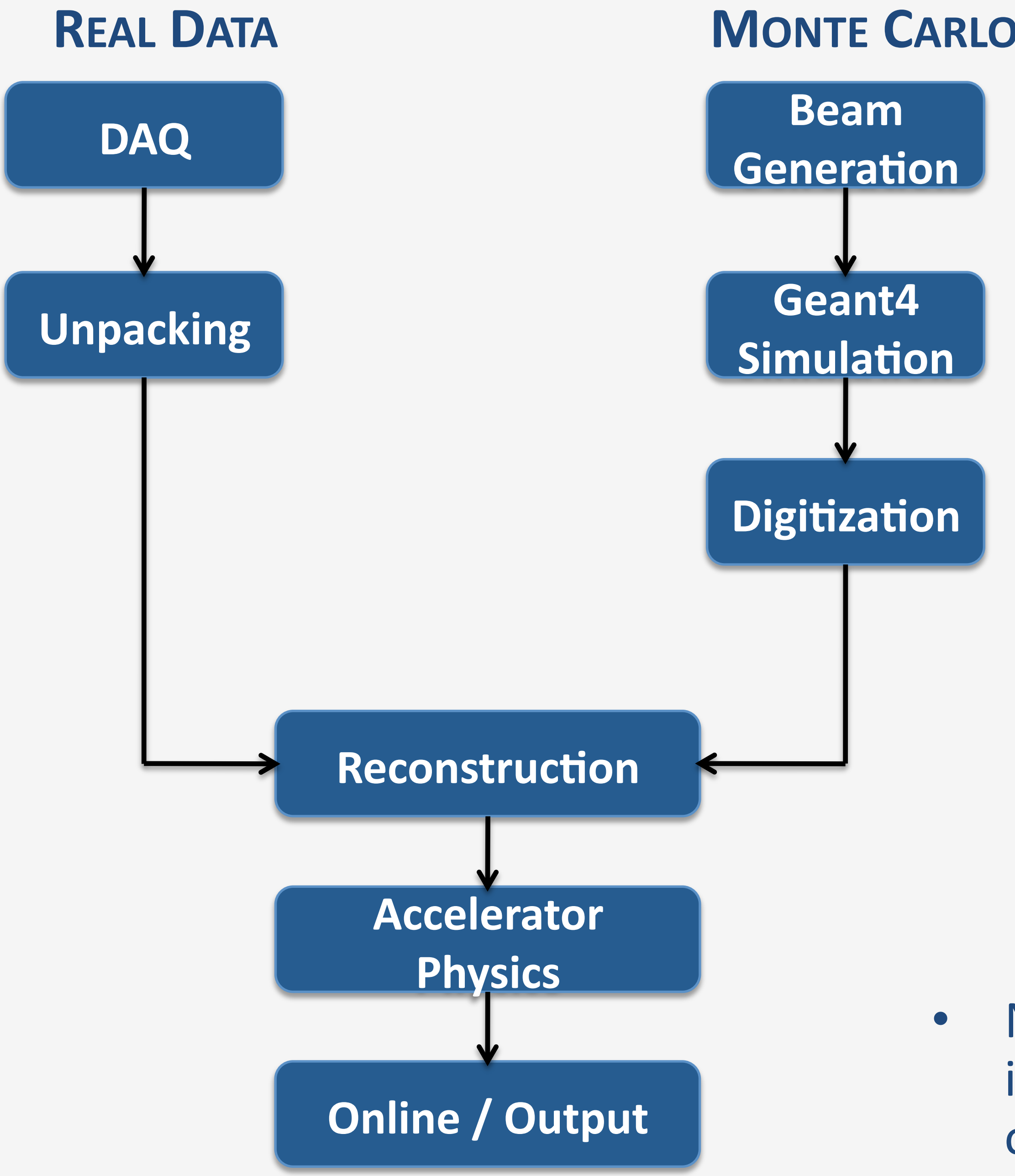
- Muon cooling: essential for future high-luminosity μ-colliders & high-intensity ν-factories
- μ from π decays have high emittance & must be cooled
- Traditional beam cooling techniques too slow due to the short μ lifetime
- Ionization Cooling is the only practical means:
  - Reduce momentum by dE/dX in absorber, followed by RF reacceleration to restore $p_{||}$
- Design, build & commission a realistic section of cooling channel to precisely measure emittance reduction
  - Lessons from MICE => the design of a full cooling channel for a ν Factory/μ Collider



MICE Software Goals

- Support particle physics and accelerator physics needs
  - Traditional detector simulation & reconstruction, Transfer matrices, emittances
- Long-term maintainability
  - MICE is built & operated in stages
- Monte Carlo simulation of the experiment
  - Geometrical description of beamline, detectors, fields
  - Digitization of detector response
- Reconstruction & Particle ID
  - Fiber trackers for momentum measurement. Time-of-Flight, Cherenkov, Calorimeters for PID
- Online reconstruction & monitoring during data-taking
- Interface to Configurations & Calibrations Database
- Framework for analysis tools
- Developer tools
  - Code testing, Issue tracking, Documentation
- Well defined APIs and framework

DATA FLOW



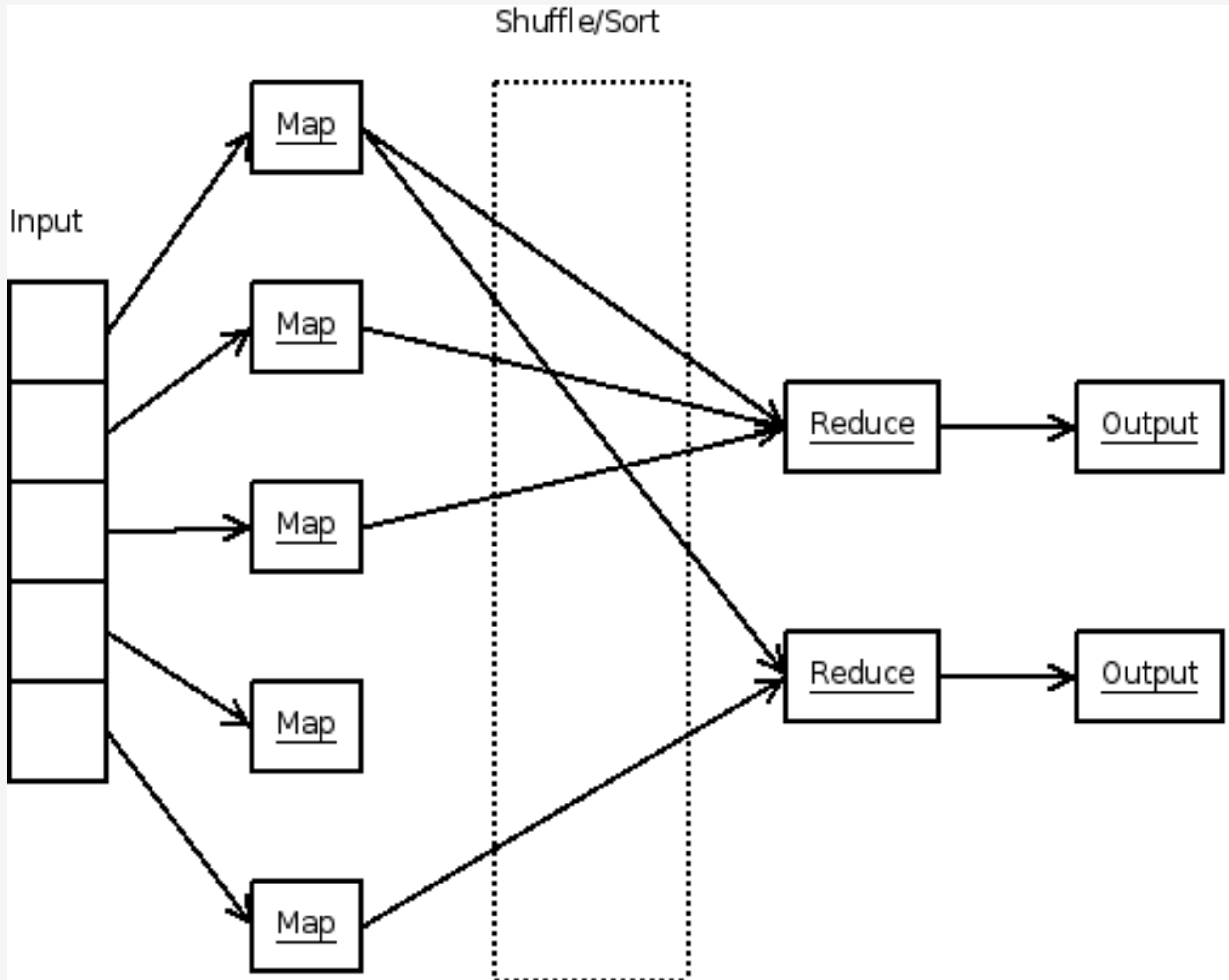
CONFIGURATIONS DB
Cabling, Calibrations, Geometry

- Note that reconstruction is same regardless of MC or real data

MAUS: MICE ANALYSIS AND USER SOFTWARE

- Design inspired by MapReduce
  - Google, Hadoop

- Map
  - Operate on a single event
  - Can run in parallel
  - e.g. Simulate, reconstruct
- Reduce
  - Operate on a collection of events
  - e.g., Summary histograms



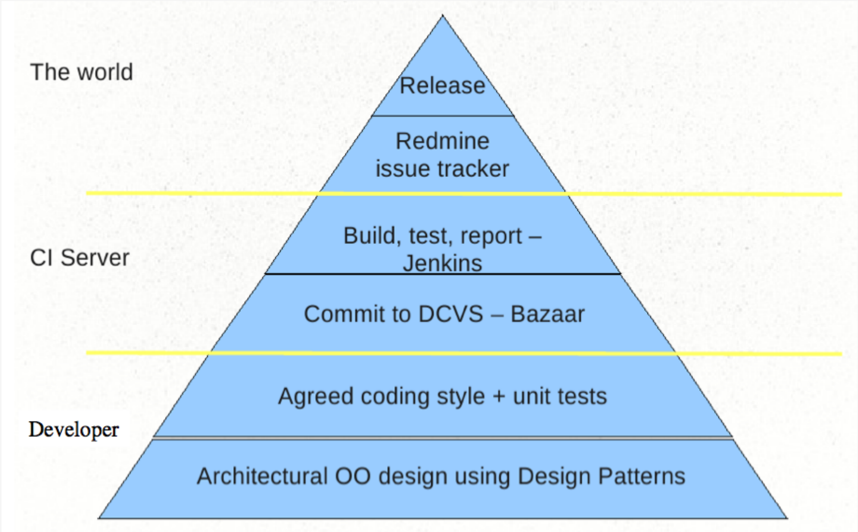
MAUS:
MICE ANALYSIS AND USER SOFTWARE

- Design inspired by MapReduce
  - In MAUS: Input-Map-Reduce-Output
- API framework built on modules
- **INPUT:** Read in data
  - DAQ data, beam library for Monte Carlo
- **MAP:** Process spills & return modified data
  - A spill is the primary data block & consists of several event triggers
  - Monte Carlo simulation, Detector reconstructions
  - Mappers have no internal state & can operate in parallel
- **REDUCE:** Process accumulated data from *all* mapped spills
  - Summary histograms, run performance plots, etc
- **OUTPUT:** Save data
  - Write out in ROOT/JSON format

MAUS Framework

- Developers write modules in C++ or Python
  - Python for higher-level algorithms, or where development time is a factor
  - C++ for lower-level computationally intensive algorithms particle tracking, reconstruction
  - Python-C++ bindings handled by wrappers or SWIG
- Data representation: ROOT, JSON
  - Default is ROOT, but developers find JSON quite useful for quick debugging
  - Mapper modules are templated to a data type
  - conversion between data types handled by API

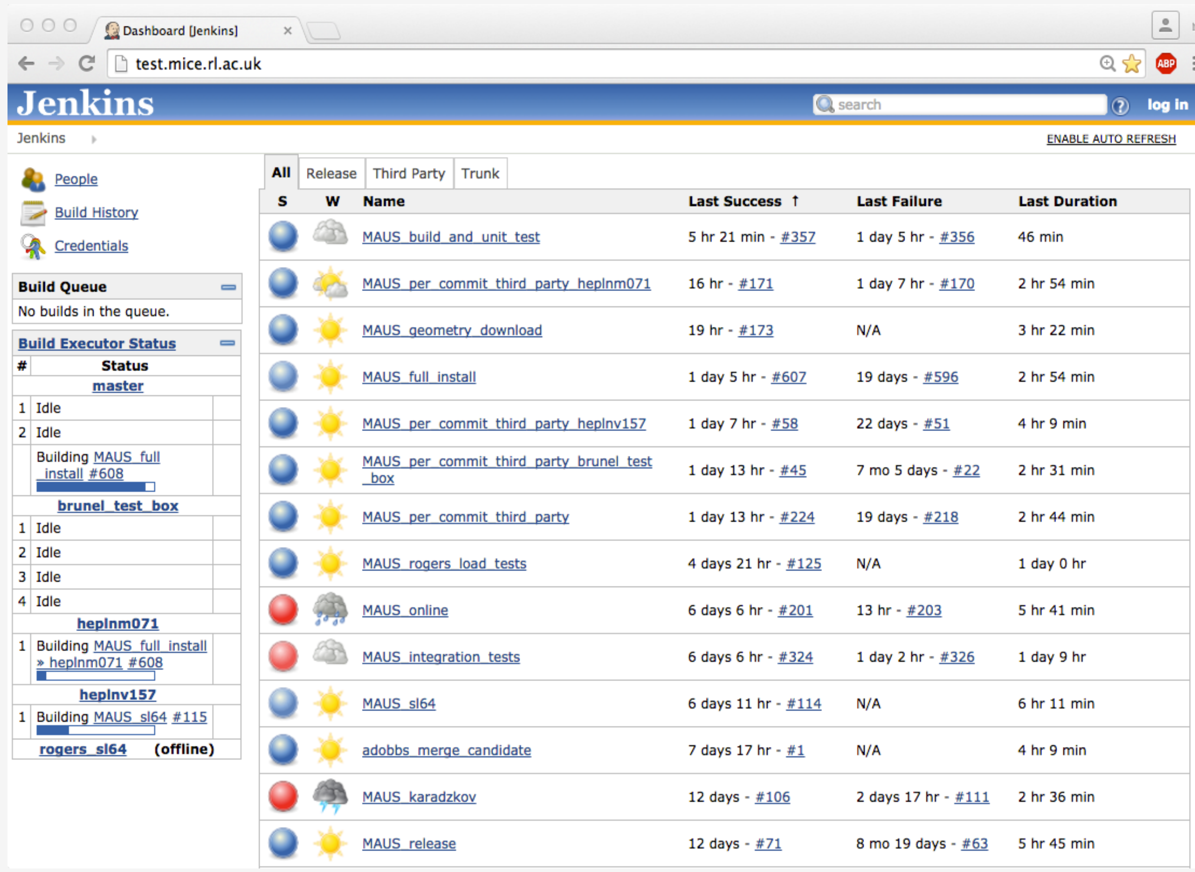
Code Management



- 10-15 developers in the UK, Europe, USA
  - Headed by Adam Dobbs @ Imperial College
- Distributed version control
  - Bazaar repository
  - hosted on Launchpad
- SCons build system
- QA:
  - Standard installation forces compliance with Py/C++ style guidelines
  - Modules required to have unit tests for inclusion in mainline branch
  - Code monitored for line & function coverage: aim ≥ 70% line coverage
- Redmine wiki & issue tracker
- Scientific Linux 6 is officially supported OS
- Several external dependencies
  - Python, ROOT, Geant4, G4Beamline, XBoa …
  - Dependencies built as "third party" libaries during installation; build scripts come with MAUS

Continuous Integration

- Unit tests
  - Test individual modules/pieces of code
- Integration tests
  - Test if units work together and with external libraries
- Jenkins CI test server with multiple slaves at RAL and Brunel University
  - One of the slaves set up as an online-mimic to test DAQ-input & Celery-queue functionalities
  - Developers run jobs on the test servers, validate & test their user branch before proposing to merge in the mainline trunk



MAUS Online

- Classic Map-Reduce:
  - Reduction happens after map operations terminate (e.g. end of run)
- However during live data-taking:
  - need to continually update plots as data arrive and are reconstructed
  - Reconstruction (map) & plotting (reduce) need to happen concurrently
- Implementation:
  - Since spills are independent, process them in parallel
  - Celery distributed task queue with RabbitMQ as queue-broker
  - Output of "tasks" (map transforms) written to a NoSQL Mongo DB
  - Reducers query the DB, read result from mappers, make plots
  - Django front-end makes plot images available for online viewing
  - The reconstruction software is identical for both online & offline

MAUS Online

DAQ

Input

spill N-1 spill N spill N+1

Celery Workers

Transform	Transform	Transform

parallel transform execution

(spill N-1)’ (spill N)’ (spill N+1)’

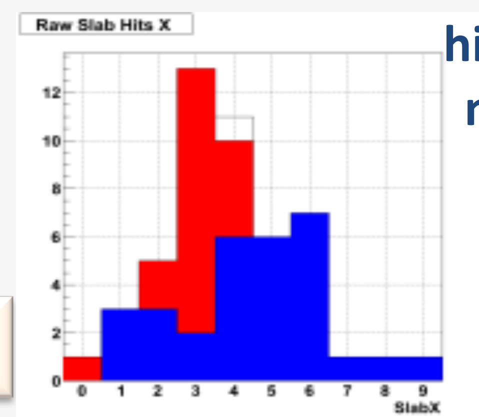
document-oriented database

web front-end

Merge

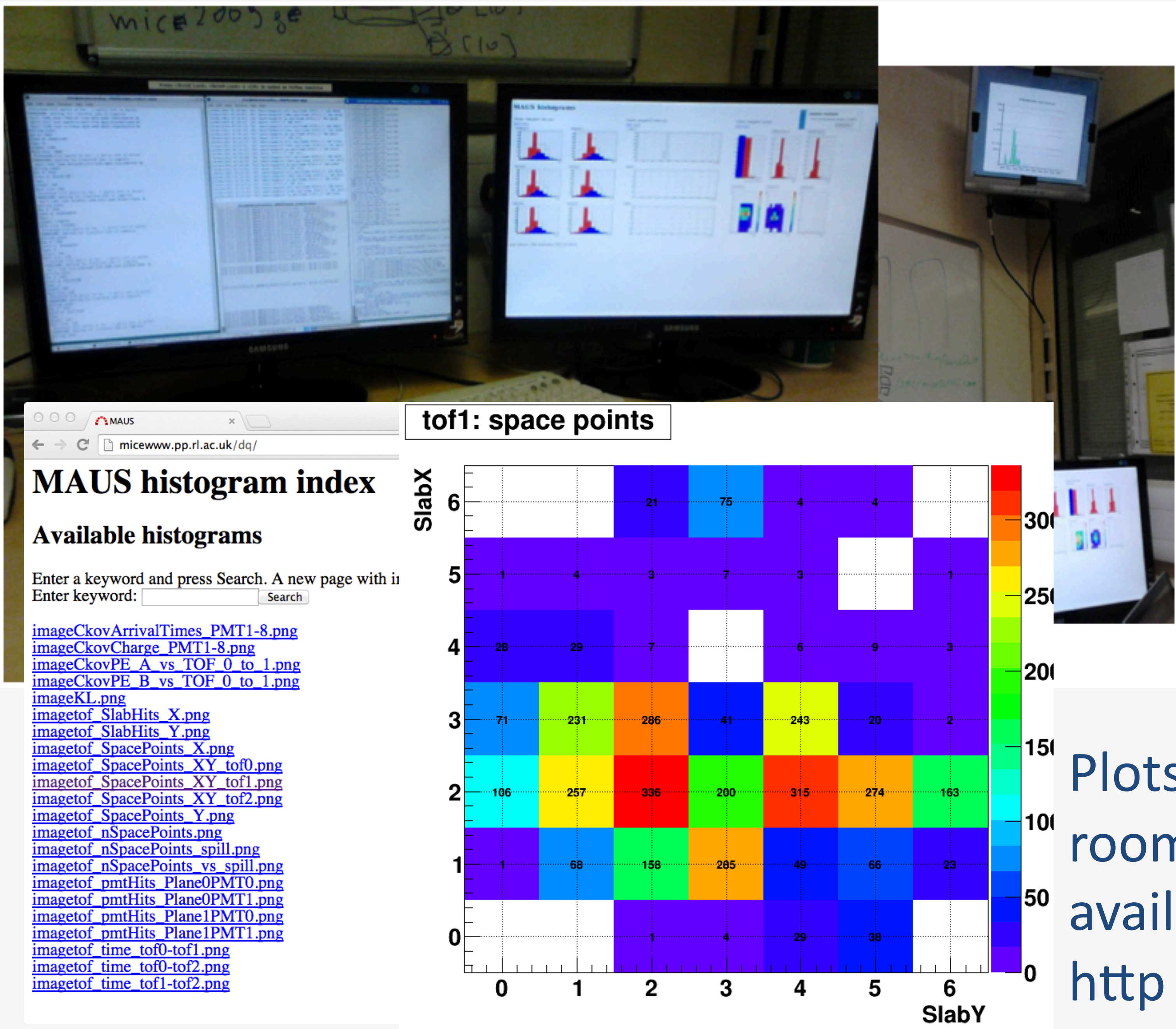
Output

Web front-end



histogram mergers

MAUS Online Visualization



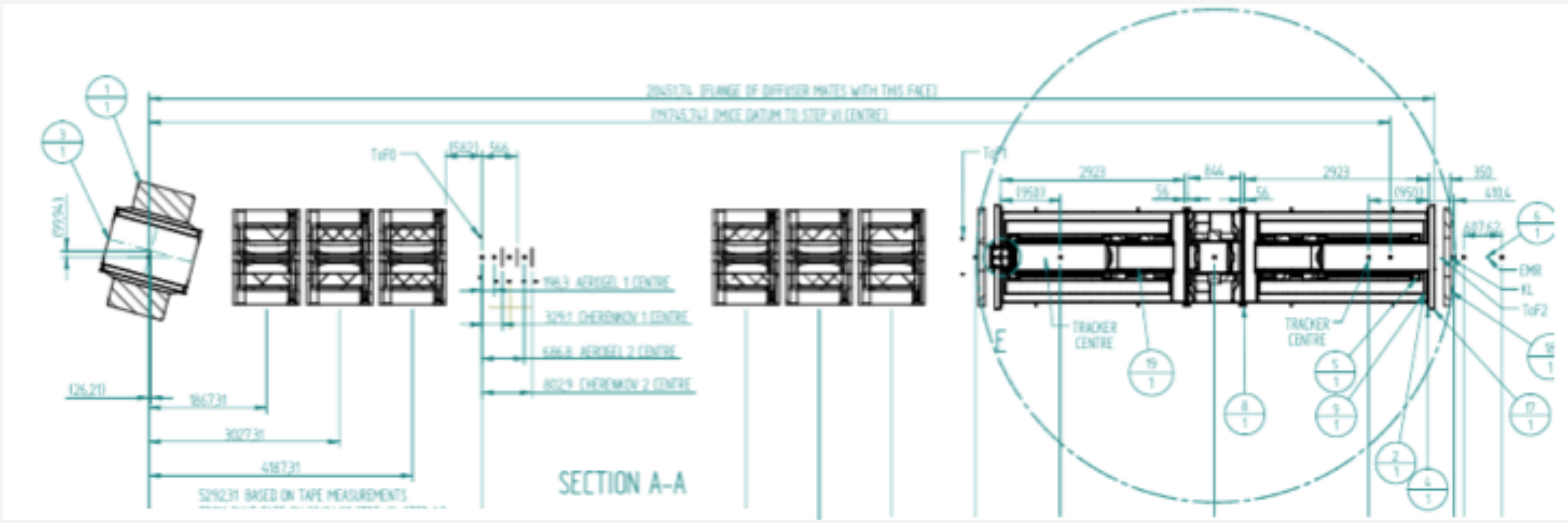
Control Room

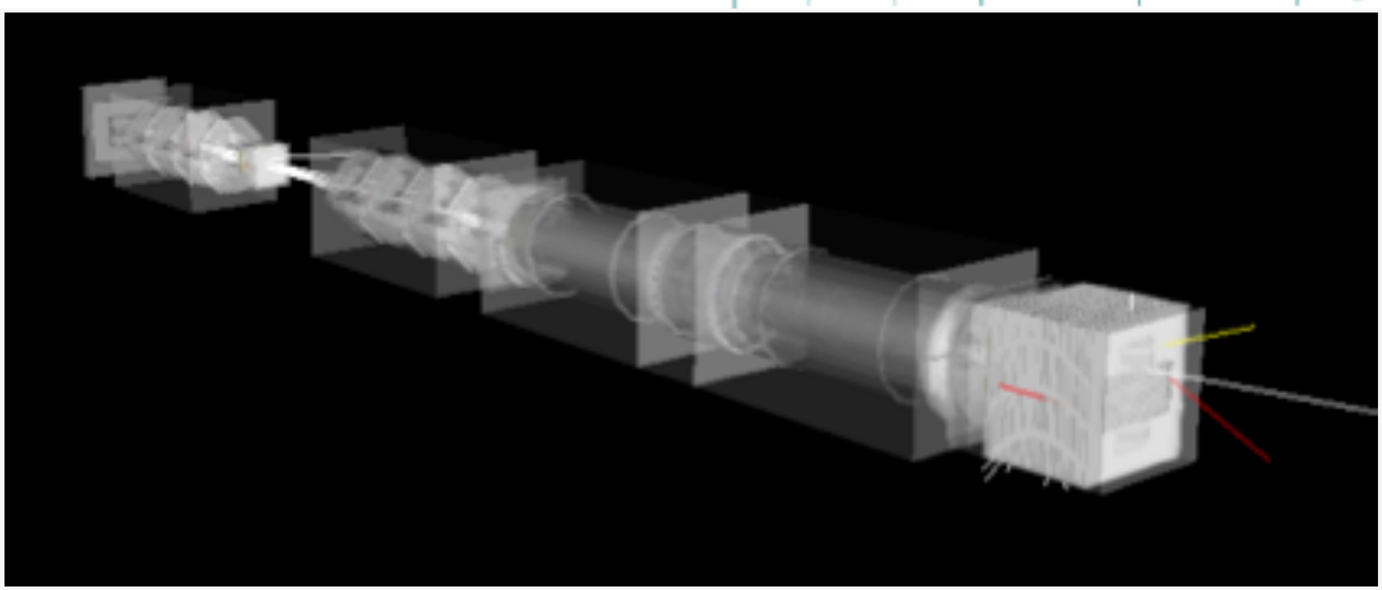
Plots from control room made available on public http

GEOMETRY

- Must provide accurate description of beamline geometry, currents, detector geometries, superconducting magnet fields
- CAD drawings of beamline -> FastRad -> GDML

  + GDML description of detectors + magnet currents → stored in DB

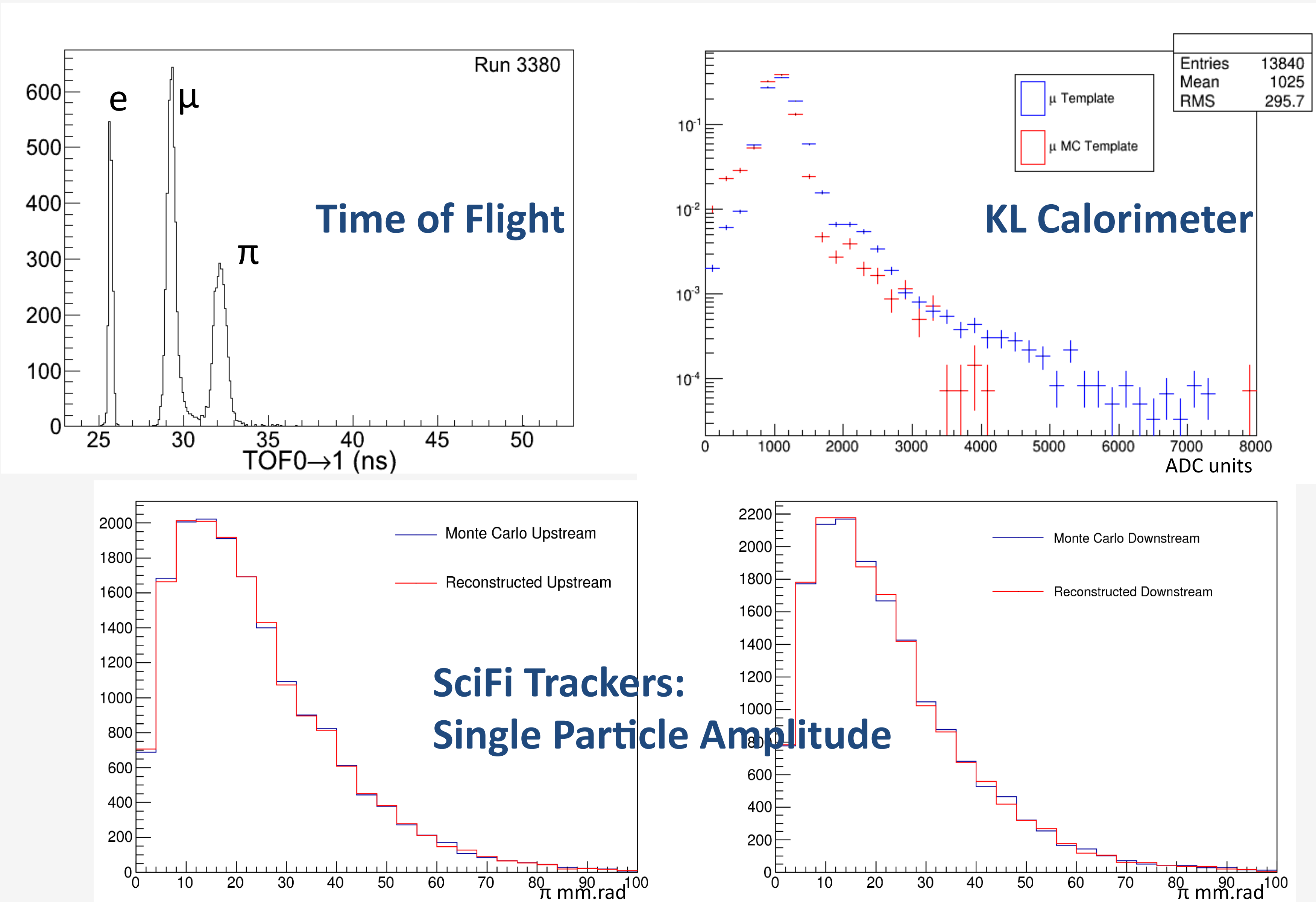




MAUS Offline

- Offline reconstruction & simulation with MAUS performed on GRID
  - RAL Tier-1 for "fast reconstruction" during data-taking
  - Batch production & re-processing on Tier-2
- MAUS installation for GRID via CVFMS at RAL
- See Poster by J. Martyniak
  - *A Grid-based Batch Reconstruction Framework for MICE*

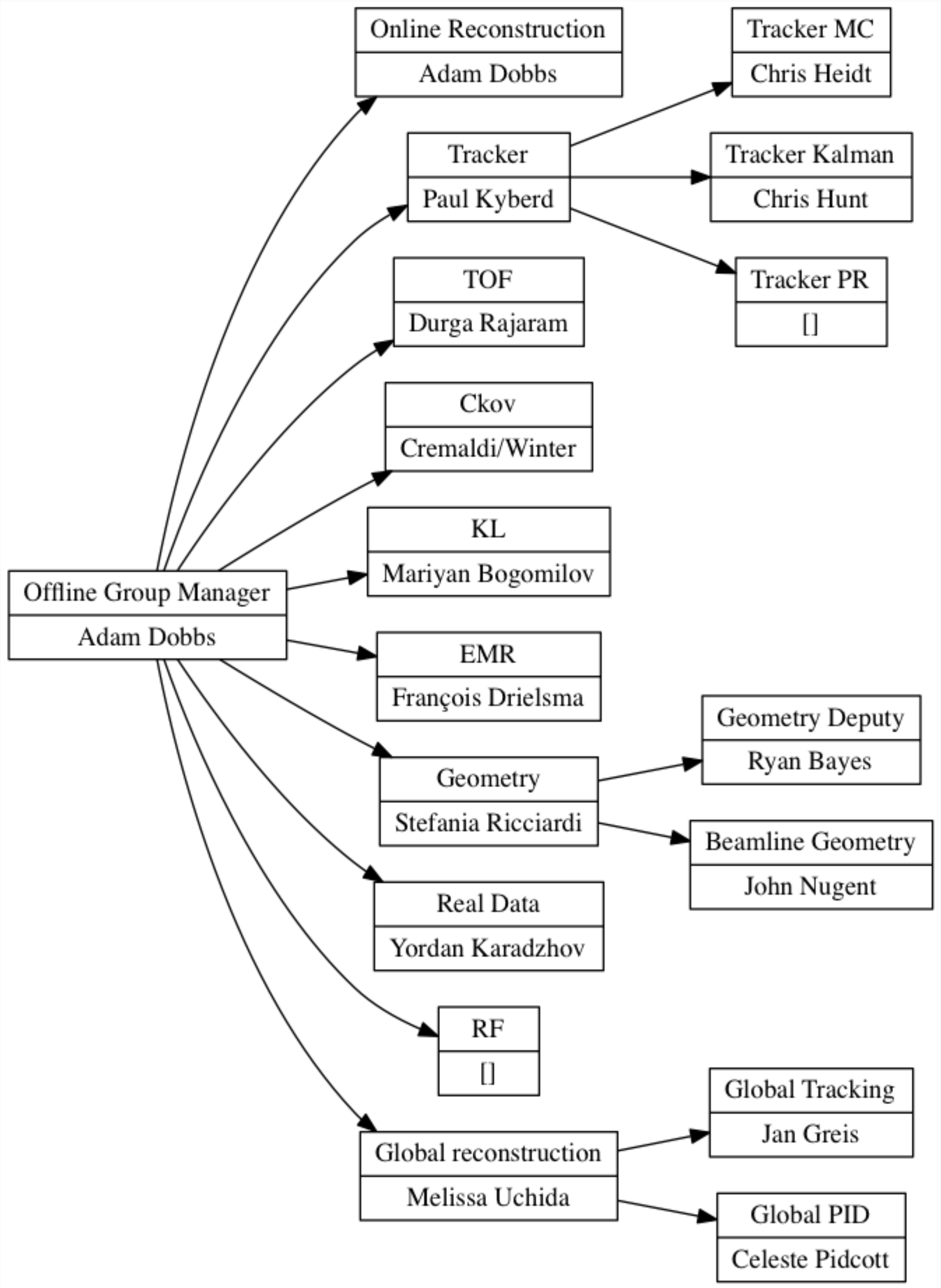
SIMULATION & RECONSTRUCTION PERFORMANCE



SUMMARY

- MAUS provides a simulation, reconstruction, and accelerator physics analysis framework for MICE
- Implemented based on MapReduce
- Well-defined & yet flexible framework
- Online distributed processing capabilities implemented using open source software
- Several industry-standard QA practices adopted
  - Code coverage, continuous integration testing
- Simulation and reconstruction software in place
- Improvements & performance enhancements continue
- Data-taking has begun & MAUS is in action

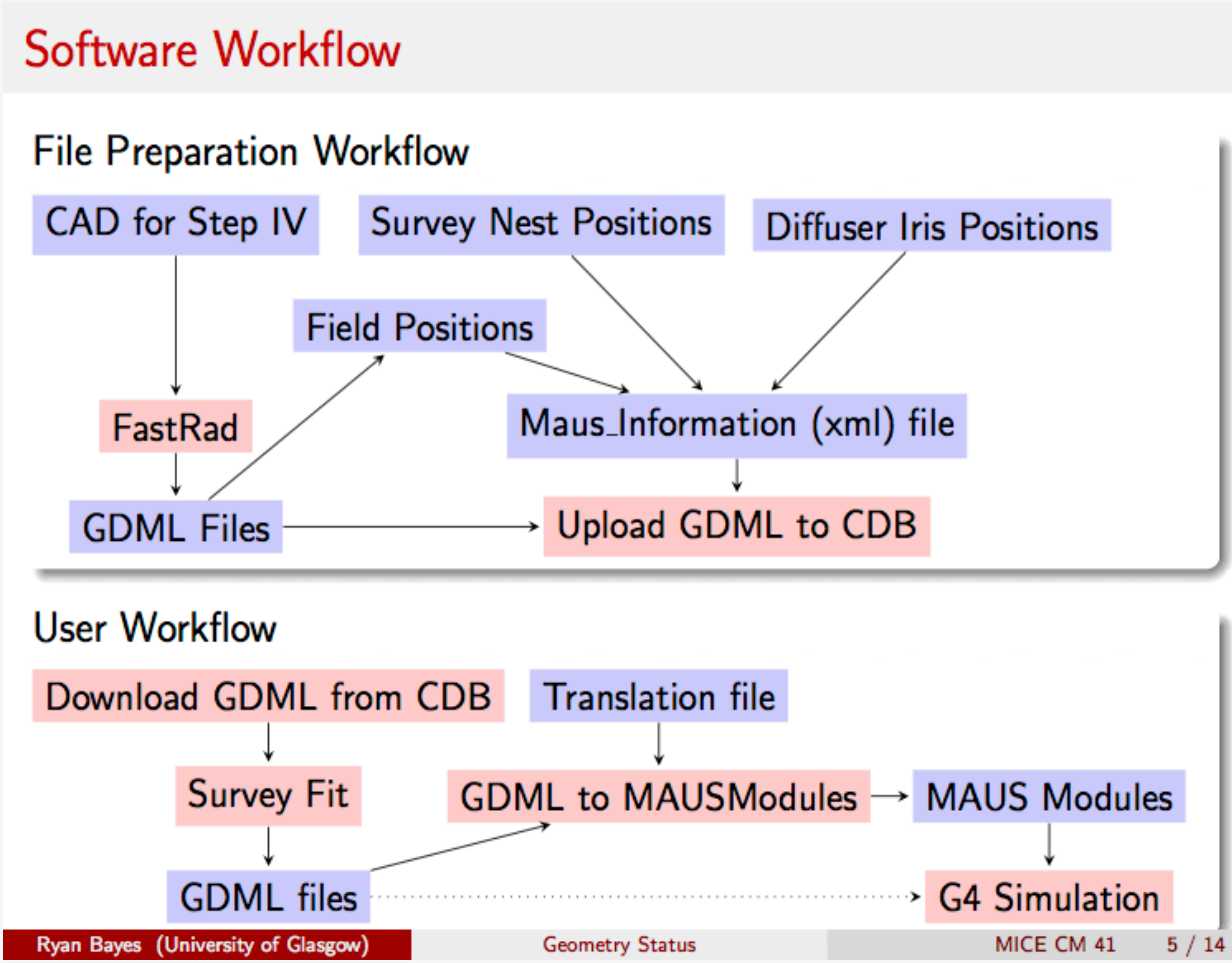
THANKS TO ALL MAUS DEVELOPERS



And many more..

Backup

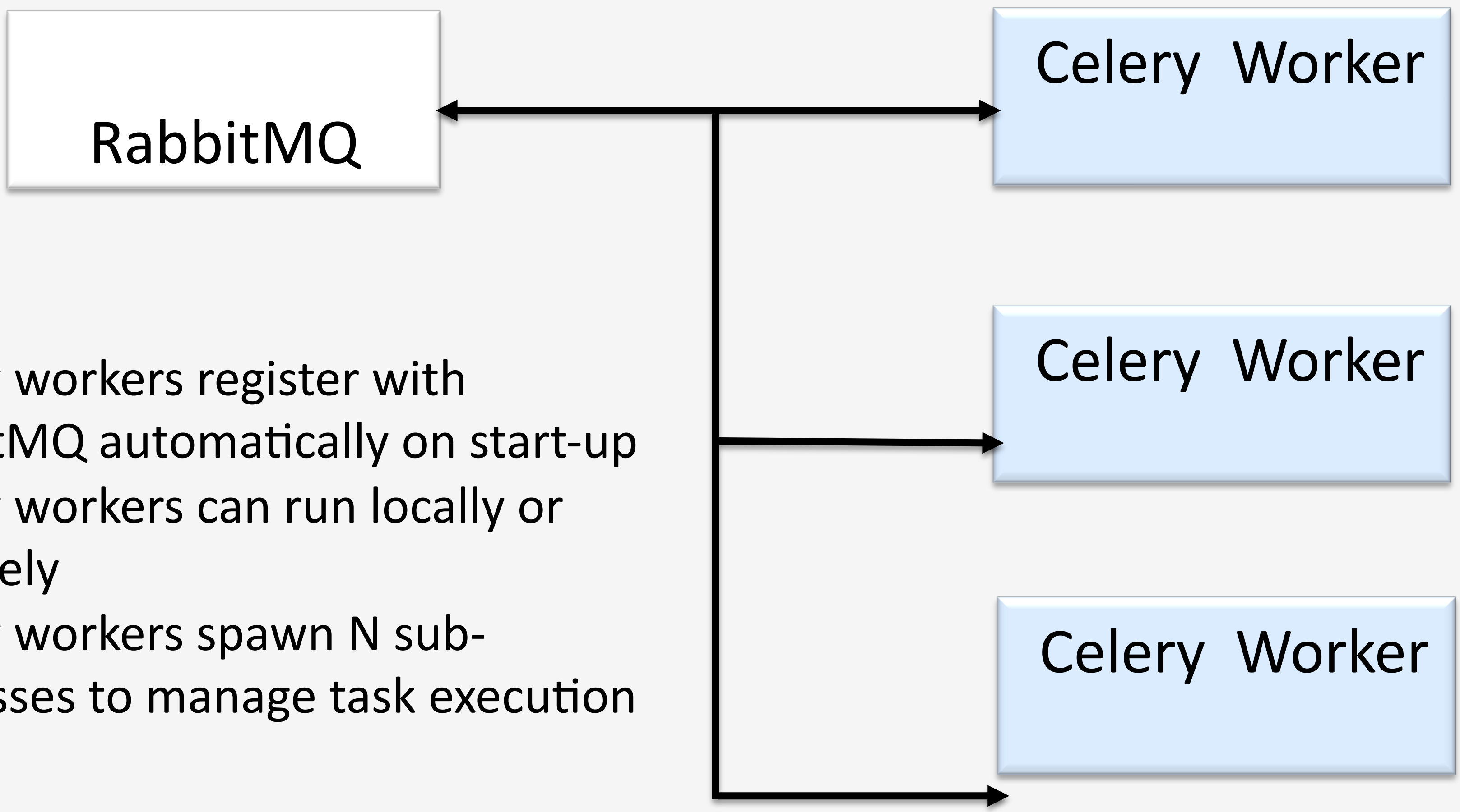
Geometry work flow



Database

- Varying configurations & running conditions
  - Beam momentum, cooling channel magnets, absorbers, calibrations ....
  - Must be monitored (Controls & Monitoring – EPICS) & stored
- Configuration DB holds
  - Run conditions from data-taking
  - Beamline settings
  - Electronics cabling maps
  - Calibration constants
  - Geometry models
- Postgres DB
- Master DB within control-room LAN, public read-only slave at RAL
- Most APIs in Python, some in C (multi-threaded EPICS does not like Python)

Celery and RabbitMQ

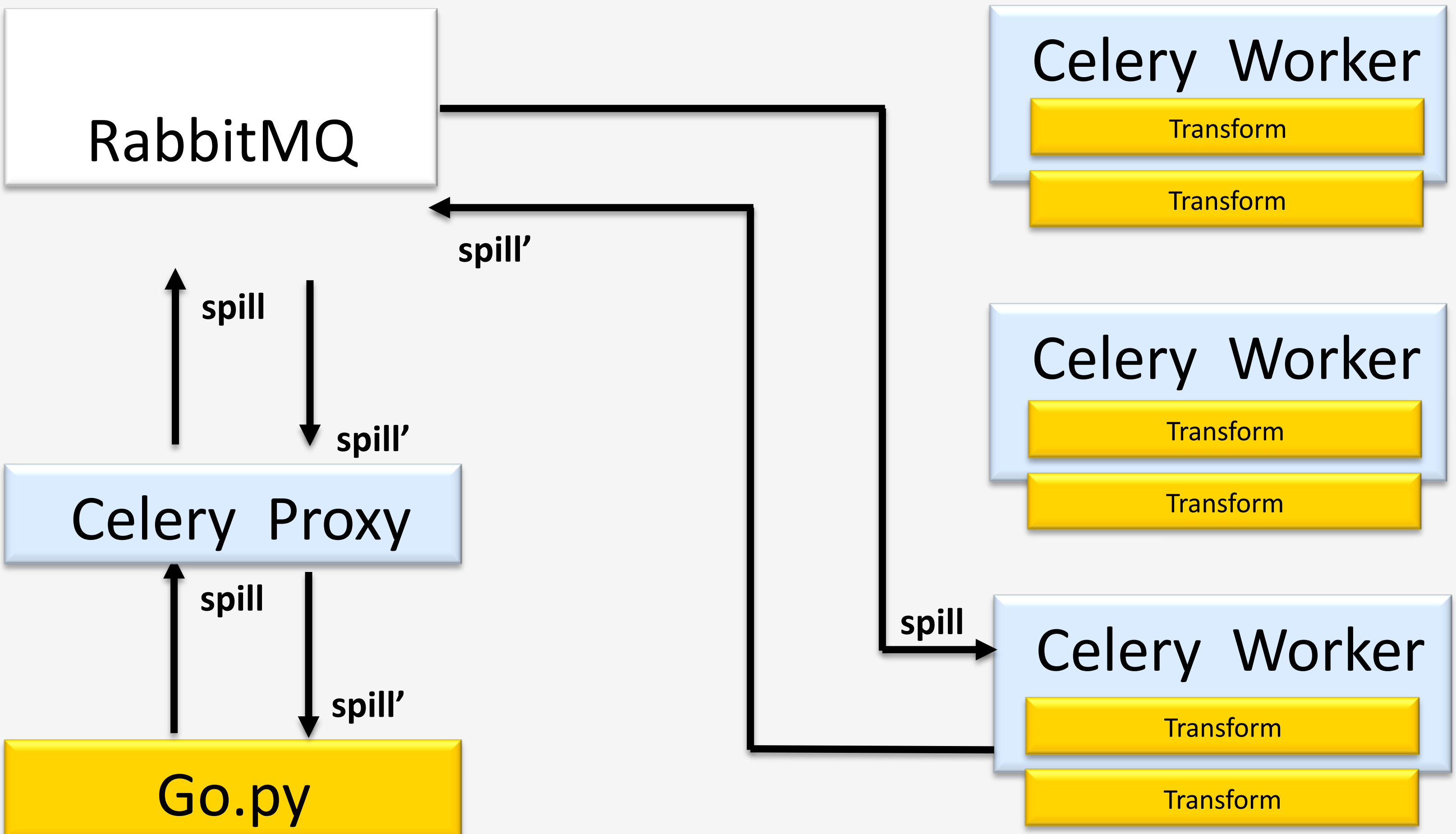


- Celery workers register with RabbitMQ automatically on start-up
- Celery workers can run locally or remotely
- Celery workers spawn N sub-processes to manage task execution

Credit: M. Jackson

Celery workers and tasks

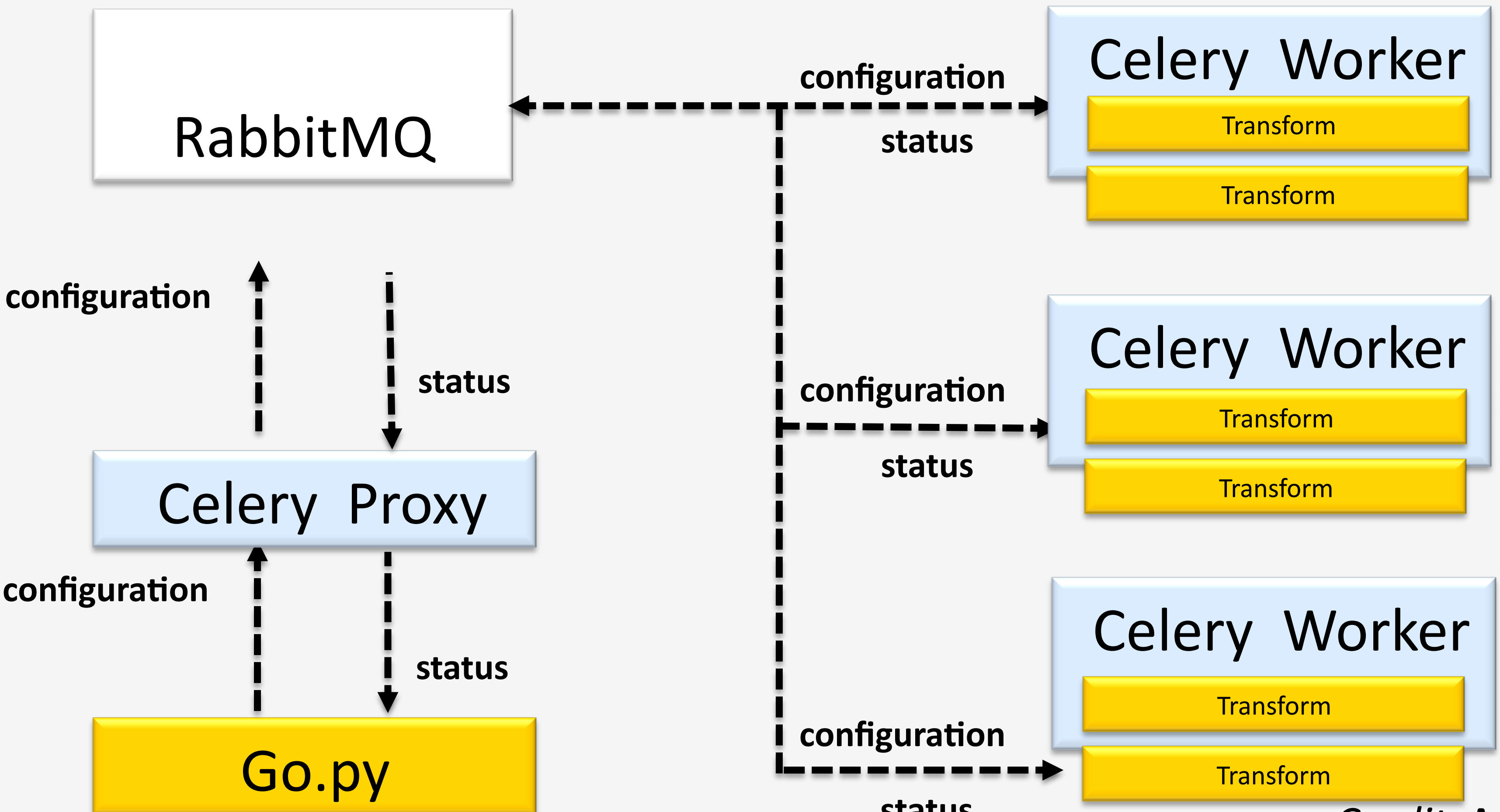
- Celery tasks (request to transform a spill) are dispatched to the next available worker
- Worker dispatches task to a free sub-process
- Each sub-process is configured to apply a transform
- Each worker has its own MAUS deployment



Credit: M. Jackson

Celery workers and broadcasts

- Celery broadcasts are dispatched all workers
- Custom code to force broadcast into all sub-processes
- Broadcast is used to ensure all workers have the same MAUS configuration and transform – dynamic worker configuration



Credit: M. Jackson