THE UNIVERSITY OF ARIZONA

ATLAS EXPERIMENT

How To Review 4 Million Lines of ATLAS Code

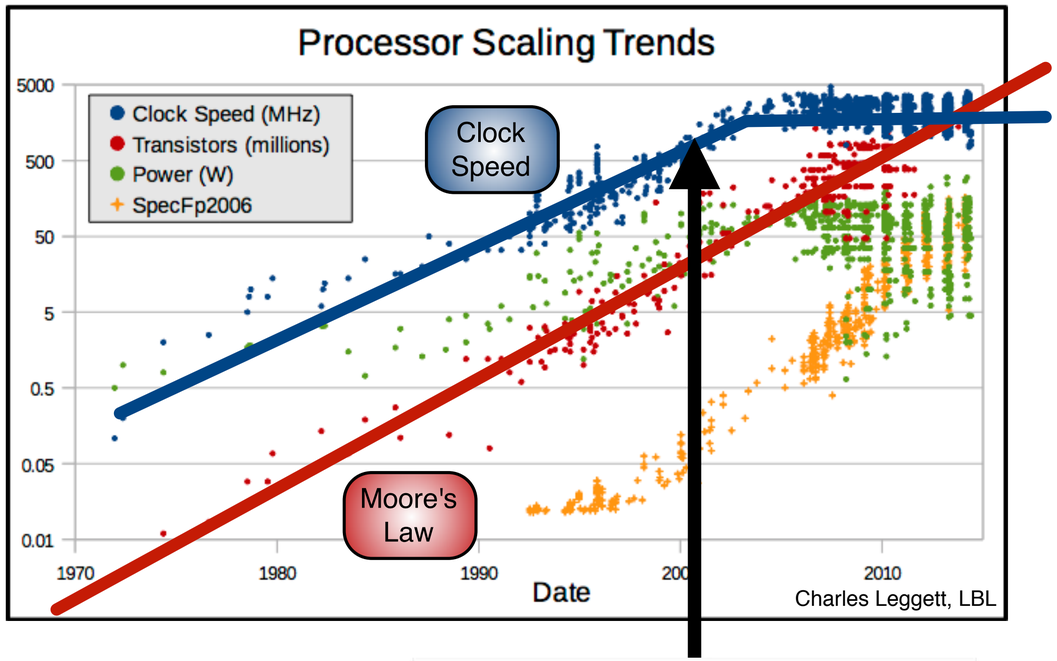
Graeme Stewart
Walter Lampl
for the ATLAS Collaboration

University of Glasgow | School of Physics & Astronomy

CHEP San Francisco
October 2016

ATLAS and the Future Computing Challenge

- ATLAS's offline software, Athena* was designed in the early 2000s
  - Commodity hardware was *single core* and clock speeds were *rising exponentially*
- Since 2005, single core clock speeds stalled
  - Single Athena job runs at much the same speed as it did a decade ago
- It was clear we would have to rethink many core assumptions from the original design
  - Especially in the light of HL-LHC in the future

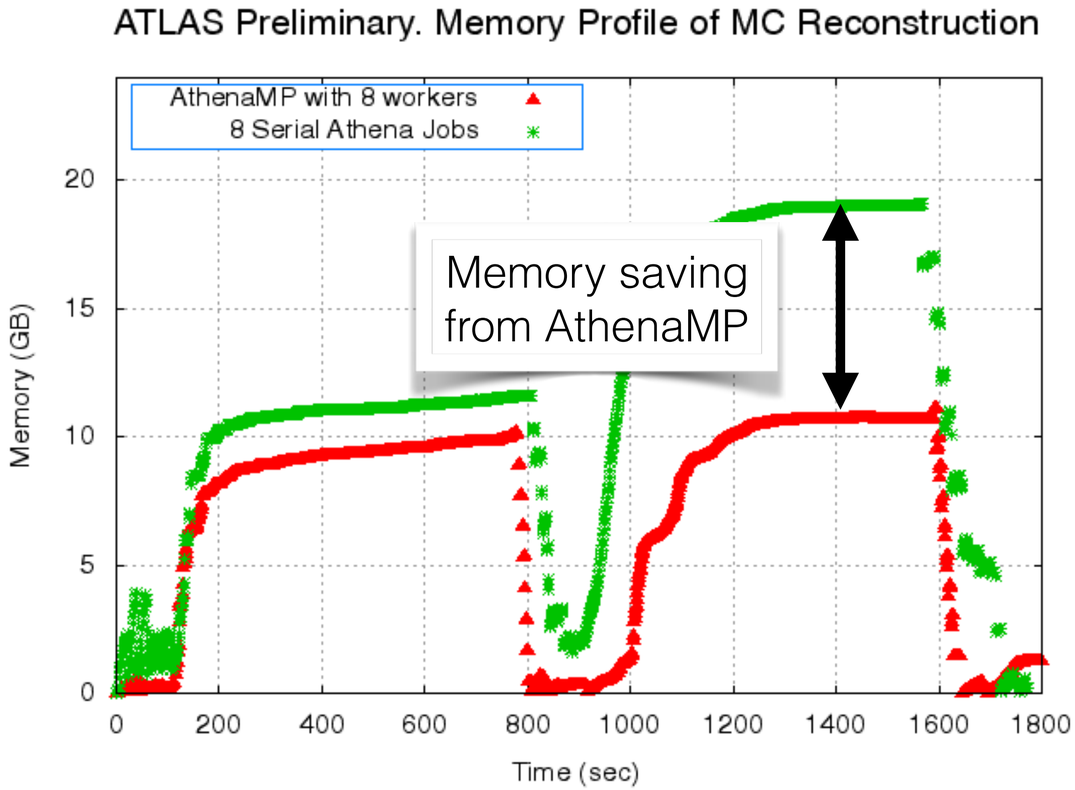


ATLAS software designed here

*based on shared Gaudi framework

Concurrency Challenge

- As Moore's Law has been mostly healthy for the last decade transistors are used for
  - Multiple Cores
    - with techniques like hyper-threading or FPU sharing
  - Vector Registers
  - Larger Caches
- ATLAS have trivially exploited multicore machines using
  - Simple naive parallelism (run N jobs)
  - Multi-processing with 'copy on write' sharing (AthenaMP, run N worker processes after fork)
- These techniques have worked well, but use other machine resources inefficiently
  - Memory, in particular, is now under great pressure
    - Especially on newer architectures*: Aarch64, Xeon Phi



*see ATLAS software stack on ARM64, poster (138) Tuesday (today!);
Multi-threaded ATLAS Simulation on Intel Knights Landing Processors, Oral Thursday 2pm Track 2

A Future Framework

- ATLAS studied the requirements for an updated software framework*
- Utilising multi-threading to orchestrate the work on a single machine
  - Share memory
  - Share i/o and other resources
- However, the challenge of ATLAS's legacy code was not well quantified at that time
  - "…a substantial effort will be required for algorithmic changes and implementing new, improved, design patterns."
- But how much was this *substantial effort* and how would we engage the development community in this process?

ATLAS NOTE

ATLAS-SOFT-COM-2014-048

2014-03-13

ATLAS Future Framework Requirements Group Report

John Baines, Tomasz Bold, Paolo Calafiura, Sami Kama, Charles Leggett, David Malon, Graeme A Stewart, Benjamin M Wynne

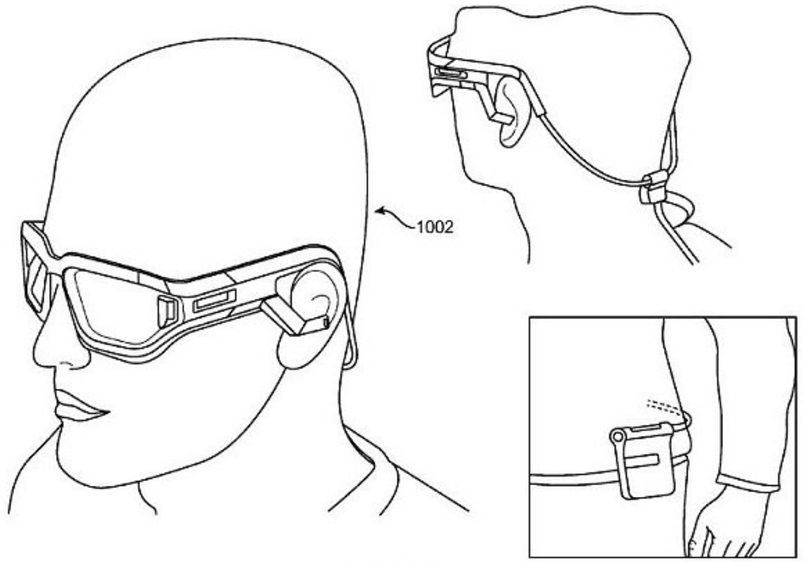
*Latest updates, AthenaMT: Upgrading the ATLAS Software Framework… Monday Track 2

Software Design Review

- Scope of the problem
  - 4.1M lines of C++ code
  - Many pieces of code unchanged for years, original authors no longer active
  - Original sense of design often lost after years of patching
- We proposed a Software *Design* Review to get all subsystems to look afresh at their code
  - We did not have the resources to look at 4M lines in detail
    - In any case, the design might be so flawed that a rewrite was the conclusion (as had already happened in some areas)
    - Data flow and interaction with other areas was critical
    - We provided advice on threading issues to watch out for
  - Organised as an Internal review
    - We have a lot of expertise in house, and from the outside understanding all the framework idioms and patterns would be an extra hurdle
    - This is also better for cross-fertilising ideas between sub-domains
    - Every domain would contribute their review material and review other areas

Put on your design goggles…

- Initial response from the community was not enthusiastic
  - Many other things on people's plate: ongoing 2015 data taking, 2015 reprocessing release, deliverables for 2016, etc.
- However, there were strong arguments in favour
  - We knew that LHC Long Shutdown 1 work had been hampered by lack of planning
    - We were sure this was a deeper and more difficult migration
  - There is *never* a time when there are not urgent things to do
    - We sacrifice long term goals too readily
  - Framework development was proceeding anyway
    - Without a feedback loop from algorithmic code there were higher risks of not capturing key requirements
- Thus, we decided to *make it so…*



ATLAS offline Software Review 2016

The goal of this review is to understand how much effort it required to port the code we use for simu
described in the Future Framework Requirements report.

We want to concentrate on algorithmic code that is running as part of RecExCommon. Subdetector-

This is intended to be a community review, i.e., no external reviewers. We are asking each sub-dom

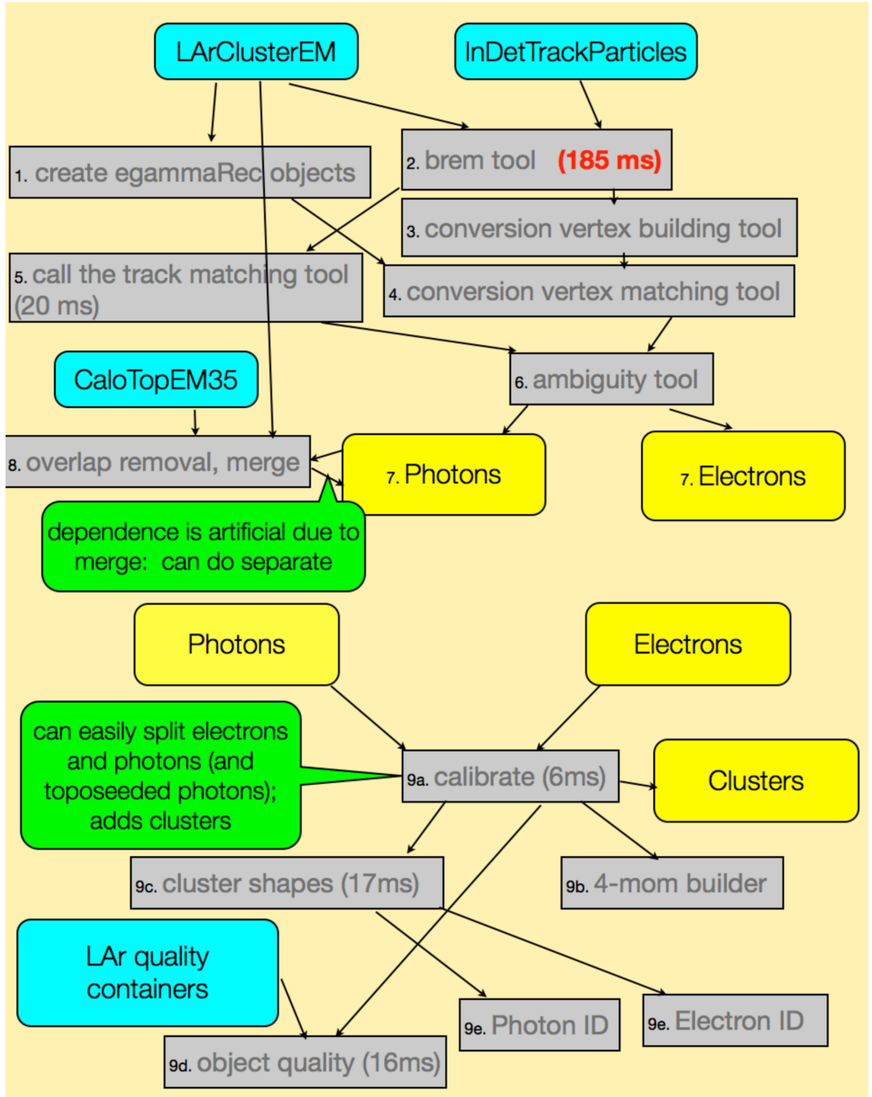
Sub-domains to be reviewed

We plan to break down the review into the following sub-domains:

- Digitzation & Simulation:
  - Geant4 (Core Simulation)
  - FastSim
  - Core Digtization
- Detector Reconstruction and Digitization
  - InDet
    - TRT
    - SCT
    - Pixel
  - Calo
  - Muon
  - Forward
- Combined Reconstruction
  - Tracking & Muon
  - egamma

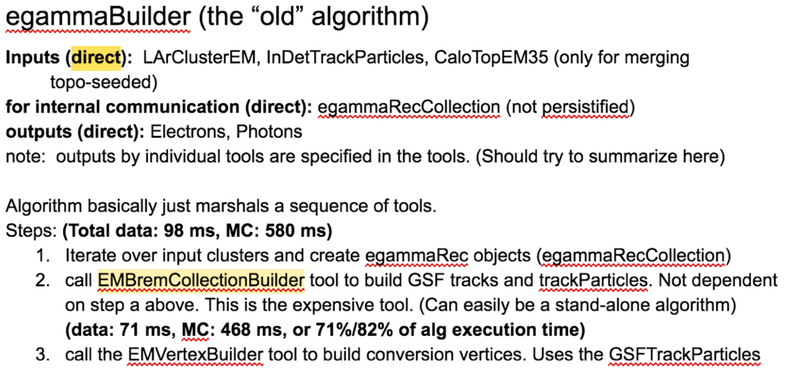
Review Inputs

- The review organisers provided guidance on a number of items to be addressed in each sub-domain
  - High level design description
    - Data flow diagram, showing interactions with the event store and conditions service
  - Documentation links and description of testing procedures
  - Threading gotchas: non-const statics, const_cast, backdoor communication channels
  - New framework issues to be addressed, e.g.,
    - All tools to become private
    - Forbid python in the event loop
    - Incident handling (e.g., code that does something special after each event is processed)
- We asked that all input should be ready at least 3 days in advance of the review



Data and code flow in Egamma domain (Jovan Metrevski)

Review Process

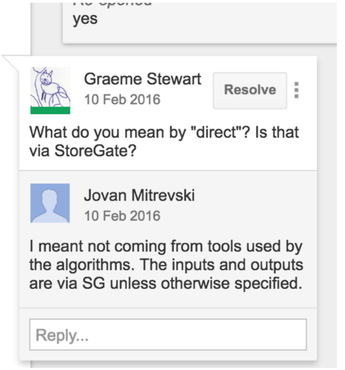


egammaBuilder (the “old” algorithm)

Inputs (direct): LArClusterEM, InDetTrackParticles, CaloTopEM35 (only for merging topo-seeded)
for internal communication (direct): egammaRecCollection (not persistified)
outputs (direct): Electrons, Photons
note: outputs by individual tools are specified in the tools. (Should try to summarize here)

Algorithm basically just marshals a sequence of tools.
Steps: **(Total data: 98 ms, MC: 580 ms)**

1. Iterate over input clusters and create egammaRec objects (egammaRecCollection)
2. call EMBremCollectionBuilder tool to build GSF tracks and trackParticles. Not dependent on step a above. This is the expensive tool. (Can easily be a stand-alone algorithm) **(data: 71 ms, MC: 468 ms, or 71%/82% of alg execution time)**
3. call the EMVertexBuilder tool to build conversion vertices. Uses the GSFTrackParticles



yes

Graeme Stewart
10 Feb 2016
Resolve

What do you mean by "direct"? Is that via StoreGate?

Jovan Mitrevski
10 Feb 2016

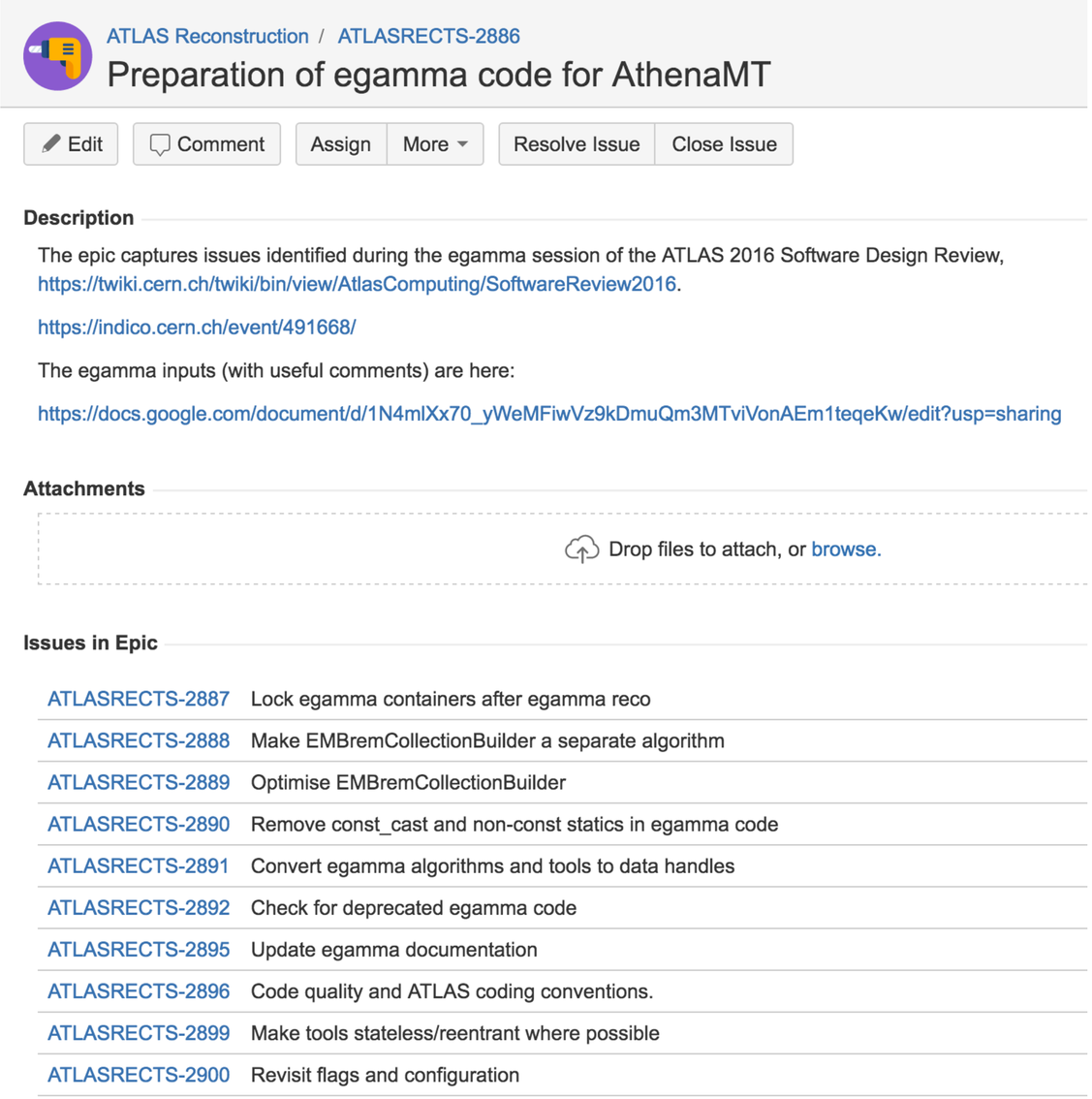
I meant not coming from tools used by the algorithms. The inputs and outputs are via SG unless otherwise specified.

Reply...

- There were always 3-4 nominated reviewers for each area
  - These reviewers really took the time to go through the material and comment on it
  - But the reviews were generally more widely attended — people found them useful
  - We had a note taker for all the reviews
- Having material in advance was extremely good
  - Many discussions and clarifications could happen in advance of the review meeting itself
  - Google Docs and Google Slides proved themselves to be superb collaborative tools
- We tried (and mostly succeeded) to keep the review time to 1 hour
  - Avoided just walking through the material verbatim
  - This helped us concentrate on the key points and keep the discussion focused
  - In many reviews we actually did discuss quite detailed code points — kudos to the reviewers!

Review Outcomes

- We had to make sure that the review outcomes would be actionable improvements that could be tracked
  - Not just an *it would be good if, blah, blah, …*
- So software coordination processed the input documentation and the review notes into a Jira Epic Ticket
  - One per domain area
  - Individual tickets in the epic record a single issue
  - Domain software experts can break this down into sub-issues if they want too
- We tried to ensure we made sensible estimates of how long each task would take



What we Learned

- As expected the level of software expertise varied widely between different groups
  - Some groups were already well advanced with Run 3 plans
  - Other groups had clearly lost a lot of their original coding expertise, both generic and specific
    - Warnings about non-const statics and const_cast led some groups to believe that static_cast was a threading danger (it's not)
    - Design could not be understood even when the code (after some effort) was
- Groups that were in a good shape contributed most to the general discussions and helped to promote good design patterns
- Groups that were in poor shape learned most from the process
  - In particular, they started the *process of re-learning* the code again
  - In many cases the review input *became the documentation* for the current code
- Many people commented that the review had been much more useful than they were expecting

Conclusions and Future Challenges

- ATLAS undertook a wide review of its Athena software in 2016
  - Our review is now mostly complete, with all key areas covered
- The review process has produced an up to date body of documentation on the current state of the code
  - Which has helped start a process of re-learning forgotten corners of ATLAS software
- There has been substantial exchange of ideas about good design patterns and discussion about how to mitigate and overcome poor ones
- The process did involve a considerable investment from a hard pressed development community
  - Which has been recognised now as being worthwhile
- ATLAS can approach the transition to a multi-threaded framework for LHC Run 3 with a much better understanding of the work required and the key difficulties to be overcome