THE UNIVERSITY OF ARIZONA

ATLAS EXPERIMENT

Usage and Requirements for Gaudi in ATLAS

Graeme Stewart

with much input from the ATLAS software community

University of Glasgow | School of Physics & Astronomy

Gaudi Workshop
September 21 2016

In the beginning…

- We started work serious for a multi-threading future by
  - Beginning prototyping work in Athena, using the Gaudi Hive demonstrator
    - see, e.g., Charles' talk at the concurrency forum 2 years ago
  - Establishing a study group to see what our future framework requirements really were
    - Report dates from December 2014 and is now approved as a public note
- I assume that by now most people here are quite familiar with that work
- Here I will briefly run through the main use cases we have and functional and technical requirements that arise from that

Draft version 1.1

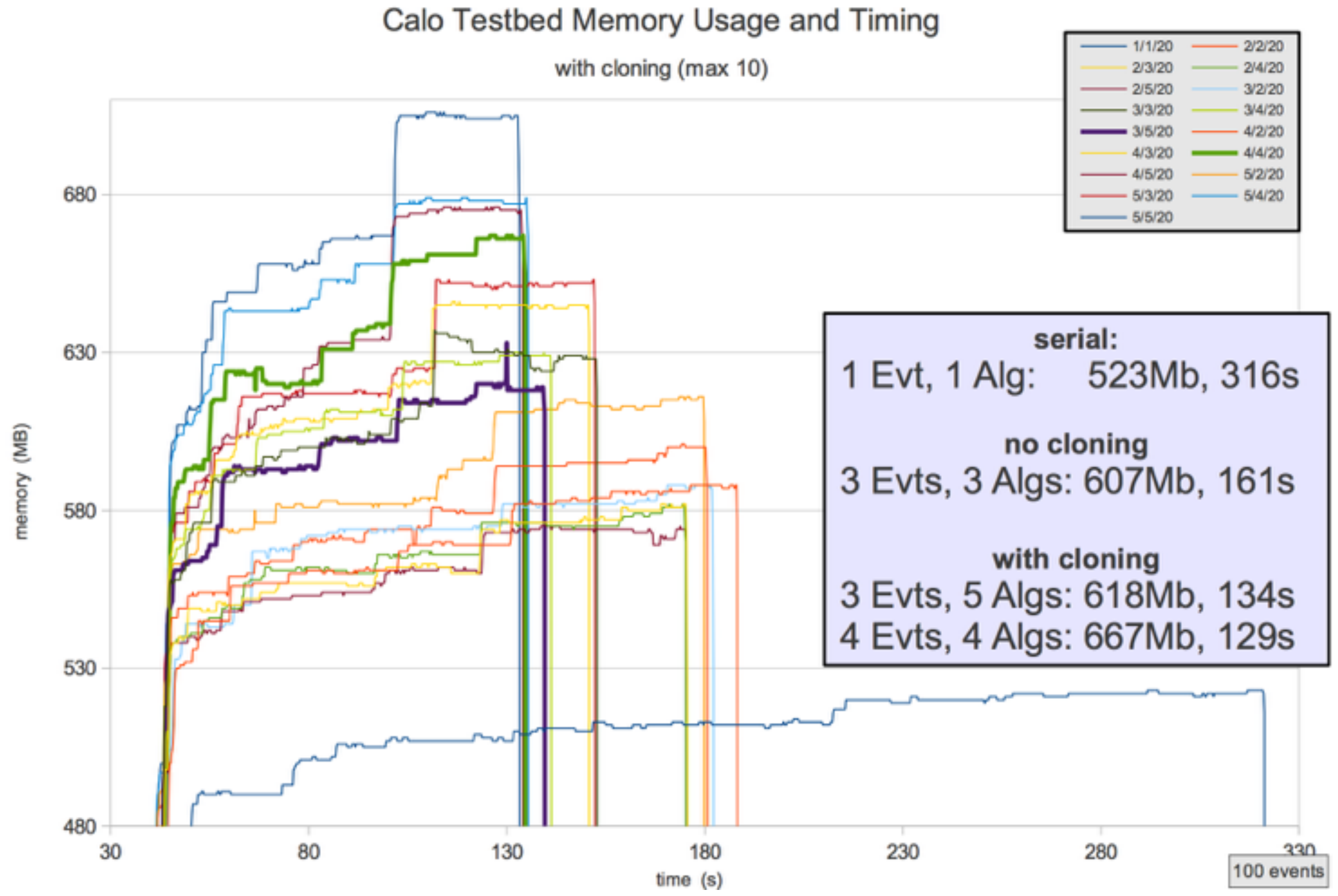
ATLAS NOTE

ATLAS-SOFT-COM-2014-048

2014-03-13

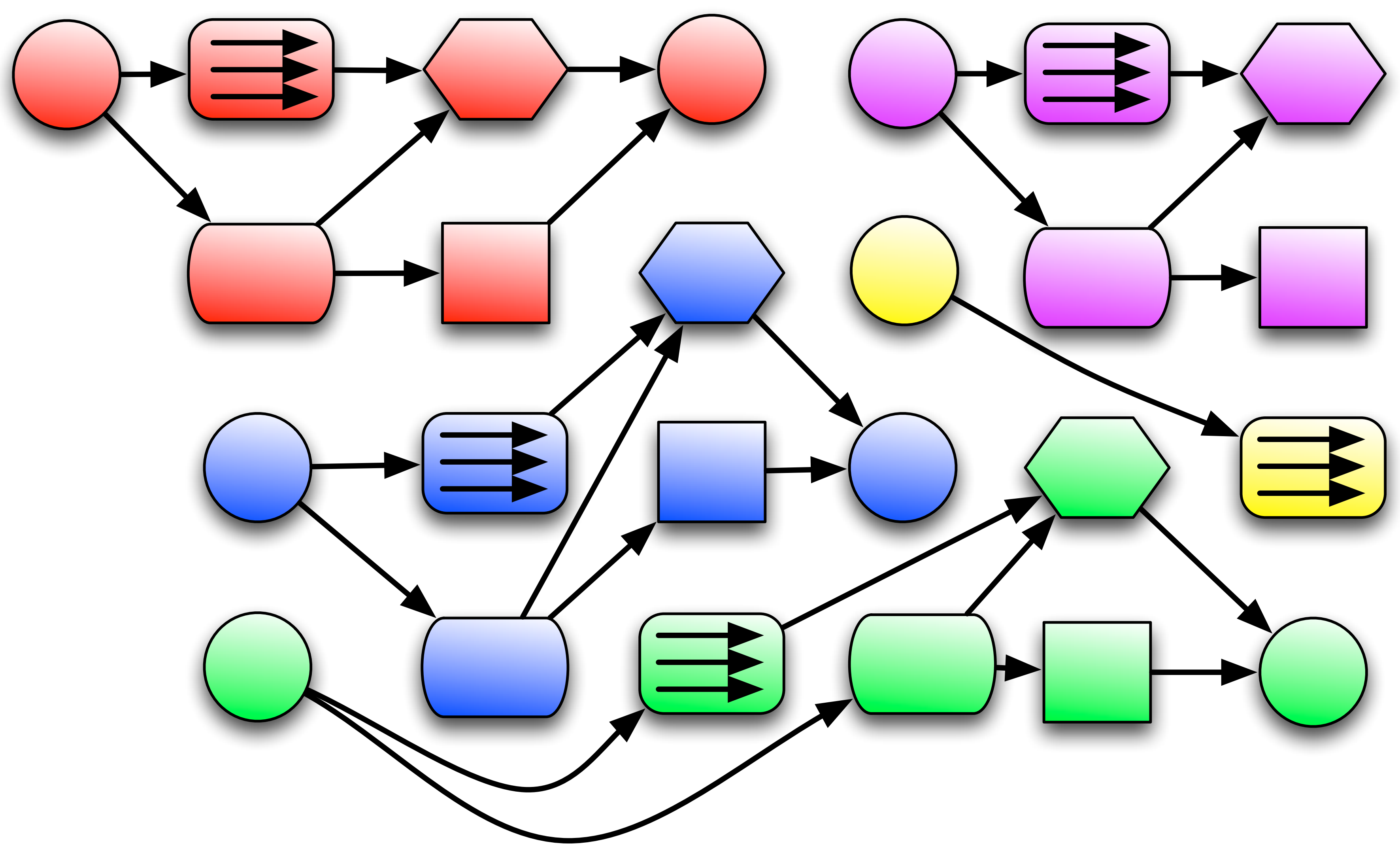
ATLAS Future Framework Requirements Group Report

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



Reconstruction

- Mainline reconstruction is the classical use case that Gaudi addresses
- In the multi-threaded version we certainly anticipate taking advantage of the key points of the new framework
  - Multiple events in flight
  - Parallelisation of independent algorithms
- This will probably open up enough parallelism to get us to ATLAS Run 3 conditions
  - Without any more that basic thread safety for many algorithms (so that should always be supported)
- Beyond that we would want to take more advantage of parallelism *inside* an algorithm
  - Mainly tracking, where CPU time/event is likely to be high
  - Continuing to throw events into flight will exhaust our memory at some point
  - We should start prototyping this, bearing in mind the ATLAS pattern would usually be to parallelise at the tool level



Event Data and Condition Data

- Evidently a hot topic
- We view conditions data as *just another piece of data*
    - It has a scope, which is usually greater than a single event
        - So it's inefficient to store it directly as event data
    - While most of our conditions change in step (e.g. at LB boundaries), there are others which are time-indexed and change more frequently (can have hundreds of changes in a 1K event reco job)
        - We are not at all convinced that trying to "bunch" conditions into groups makes that much sense — it looks like quite a restriction
            - Doesn't fit all use cases (or experiments?)
- Consequently our prototyping has been concerned with trying to handle this data *re-using as much of the current infrastructure as possible*
    - Data dependency for event processing algorithms and data handles for access
    - Retrieve data from underlying source with a service
    - Process (calibrate) the data with an algorithm
    - Orchestrate all of this using the scheduler
- At the moment we don't see any compelling reason to have fundamentally different components handling conditions data

ATLAS Conditions in Practice

- Some parts of our workflows turn out to have none or relatively few changes in conditions data
  - Most simulation and digitisation
- However, data reconstruction jobs need to access DCS data that may change frequently (e.g., 700 callbacks in a 1400 event job)
  - This is true even of our stable beams physics_main stream (99% of Tier-0 jobs have a conditions boundary with at least one piece of DCS data updated; even alignment can change in the muon system)
- In other areas we change conditions even more dramatically
  - Processing skimmed streams (e.g., DRAW_ZMUMU)
  - Detector calibration streams
  - Data overlay (more on this later)
- Additionally, conditions come in all shapes and sizes — from simple floats to detector geometry
  - We may discuss detector geometry more, e.g., Vakho's prototype <u>dealing with geometry and alignment efficiently</u>
    - Alignment objects overlayed onto base geometry
- No convincing case for adding barriers to handle conditions and it looks quite dangerous to some workflows that are important and significant
- Should settle interfaces that can handle various approaches
  - Allow for different prototypes testing real workflows

Re-Reconstruction

- This is an important use case for reconstruction and analysis
  - We want to read event data, generate a modified version of it
    - Can be recalculation of some values or decoration/augmentation
  - Persistify new versions
- This is also a critical use case for framework development
  - Not all components will be ready for multi-threading at the same time, so it should be possible to test one domain independently of others
  - e.g., testing an egamma algorithm running in parallel from already reconstructed tracks and calorimeter data

Simulation and Digitisation

- Simulation use case does not look hard for Gaudi
  - We have a well advanced multi-threaded simulation version of Athena
    - Uses one event per thread (*a la* Geant4.10)
    - This seems to suffice for utilising all cores, even on many core devices (low base memory footprint)
  - Conditions are fixed for a whole job
- Digitisation is less trivial
  - At least as ATLAS does it today, where background events are mixed at digitisation time
  - Requires many events to be loaded at once — done for now with many StoreGate instances
  - Large i/o demands on machine
  - Future tradeoff would be to use freed up memory from multi-threading to hold background events longer
    - However, we want a large sample of background events, picked in an unbiased way
      - Background event might wait a long time to be reused

Data Overlay

- ATLAS uses a data overlay technique to build simulated events with a higher fidelity than is possible with Geant4 alone
    - Combine a Geant4 signal event with randomly selected background of detector events
- This is very good for physics as the backgrounds do not suffer from any uncertainties from simulation
- However, like standard digitisation, there is a need for a large number of events to be loaded only once
- Worse, as these are real data events, they require *a priori* different conditions each time
    - At the moment this places a huge load on our Frontier servers
    - Possibly a solution here (IMO) is to build an event that, in itself, contains the associated conditions data
        - Conditions handling infrastructure has to be able to cope with that

High Level Trigger

- High Level Trigger brings a number of use cases to Gaudi
  - First, there should be a mechanism to run in *daemon mode*
    - Event processing controlled from outside the event loop
  - Including the ability to handle a change of some “conditions" on the fly, during the run
    - In particular we change pre-scales during the run, as the luminosity drops
    - Note that this is rather a limited set of in-run changes, not a full blown reconfiguration
- This mode of running is also what we would like for the ATLAS Event Service, e.g., on an opportunistic resource processing events one at a time fired in from a controller

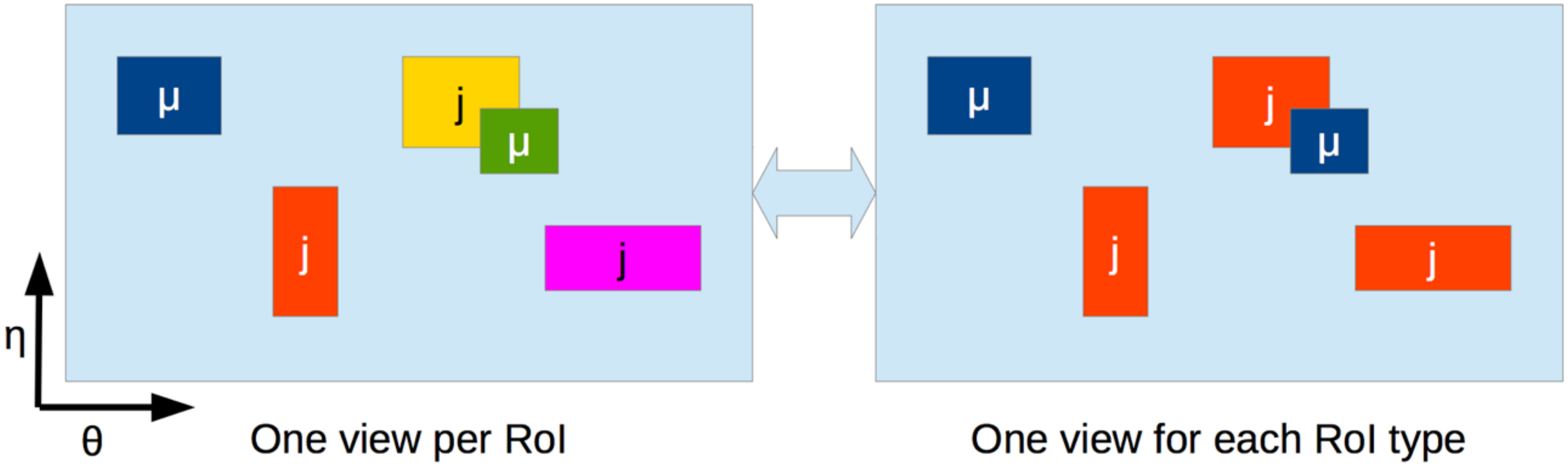
HLT Processing

- HLT will process events with a very high rejection rate
  - Typically 99%
- In addition the HLT farm has an overall CPU and network budget (latency less important — assume sufficient buffers)
  - So it's important not to over-invest in events that do not get accepted
    - Early rejection at lowest CPU cost
- Each trigger chain runs until it has flagged an event as accepted/rejected
  - The trigger decision is the OR of these chains
    - So all chains much be run until they make a decision
    - Final event status not known until the end
  - Each chain has multiple points at which it may decide not to proceed

Event Views

- A key strategy for HLT is to only reconstruct a limited part of the event data early in the event's processing chain
  - These *Regions of Interest* are signalled by L1 triggers
- We have two ideas how to do this, which Ben has presented (and we discuss this again tomorrow)
  - Essential idea is a *view* is subset of event data, but it's interface is exactly the same as the main event store
    - Thus algorithms and tools never need to know if they are running over a view
- A *static view* would consist of a scheduleable element that would always be the same for each event (e.g., the jet view), but would have an event dependent set of geometric ROIs internally
- A *dynamic view* (actually a set of!) would generate a view for each L1 ROI, and would thus vary event by event

A view of views…



DYNAMIC VIEWS:
Arbitrary number each event
More like today's HLT

STATIC VIEWS:
Can be defined in configuration
Cannot support today's HLT

- It is a critical ATLAS use case that views are handled natively by Gaudi
  - If not, the project has failed to deliver a key feature for our HLT
- Static views look much easier for the scheduler, but require deeper changes in the current HLT code

Configuration

- Current Gaudi configuration system via python is working well as a base on which to build
- It is very desirable to be able to serialise a job/task configuration in a serialisable way
  - Should be language neutral, i.e., JSON better than pickle
    - Thus reload without needing the python layer
  - This is done by ATLAS HLT, but it’s quite hacky at the moment (load from DB)
  - Light resetting of a few options is needed, i.e., specific input file for a pre-configured task
- The lack of general structure to the configuration has led to rather a mess in ATLAS
  - Procedural, complex, fragile (global namespace)
  - Mostly this is our mess, but having a model of how to do it in Gaudi would help
  - Especially for expressing a control flow syntax
- Configuration should be accessible to the quotidian day to day analyst — shouldn’t need a super expert

Analysis

- The failure of the Gaudi/Athena framework for Run 1 analysis was a big problem for ATLAS
- There is a general view that ATLAS analysis could migrate to Gaudi for Run3
  - Bring benefits of much greater skills sharing
  - And technical infrastructure (multi-threading)
- However, to do so we must understand the analysts' environment
  - Laptop based (OS X)
  - Frequent build cycles
- So we need Gaudi to be very portable and to have minimal dependencies
  - Fine to have some optional dependencies (per experiment?), but going beyond ROOT, Boost, TBB, UUID we should really try to avoid
  - Build system should work 'out of the box' on supported platforms
- N.B. We don't see any need for Windows support at the moment

Toolkits and Testing

- Analysis also likes to construct workflows as tinker toys
  - Connect an algorithm and a few tools together, run from a simple executable
    - That should be made easy to configure as well
- This is an important part of making Gaudi lightweight and accessible
  - Lower the barrier of entry
- So, even if we have a default “framework” construction, it would be really useful to have a low level toolkit interface
  - Toolkit to framework works, but not the other way around
  - As an example, running over two input files, side by side, then producing comparison histograms would be easier with a toolkit approach
- This is also vital for unit testing framework components
  - We know that we are weak on this point, and much of this weakness stems from how hard it is to spin up a few components, inject data and compare results

Evolving ATLAS Infrastructure

- ATLAS will migrate from SVN to git and simplify its build setup
- We want to make sure that we can build and run against any arbitrary commit/tag from the Gaudi repository
  - We want to check that Gaudi HEAD didn't break an ATLAS use case
- That said, we are quiet happy to deprecate old cruft out of Gaudi
  - In fact, we see it as quite critical to the health and success of the project
  - Technically these migrations should follow the usual pattern
    - Deprecation Warnings→ Removal

And don’t forget about…

- Documentation
  - Think how a naive user would get started
- I/O
  - Critical to do this well in a concurrent environment
  - ROOT and nothing but…?

Timelines

Dates	Framework	Algorithmic Code
2015	Baseline Functionality	Very few algorithms, concentrate on high inherent parallelism; general clean-up
2016	Most functionality available (including views)	Wider set, including CPU expensive algorithms with internal parallelism; continue clean-up/prep; first trigger chains
2017	Performance improvements and final features	Migration starts with select groups
2018	Performance improvements	Start bulk migration
2019	Bug fixes	Finish bulk migration
2020	Bug fixes	Integration

- Finally, a reminder of the main timelines for ATLAS… and notice that *we already started to fall behind our schedule*
- So we need to be quick and agile in our development now