Event View Design

John Baines, Tomasz Bold, Paolo Calafiura,
Charles Leggett, Bartosz Mindur, Francesca Pastore,
Scott Snyder, Ben Wynne

09/11/15

Event views

An event view should effectively fulfil the same role as trigger elements do now

- Define a subset of data from a whole event, corresponding to an RoI
- Provide access to that data
- Allow new data objects to be attached to the RoI

The plan is to use far fewer event views than we do TEs though

The views should also introduce minimal overhead, and it must be possible for the offline software to ignore them

We've been looking at how to integrate views into the Future Framework

By the end of the year, we aim to have a simple HLT-like workflow running:

1) Create views (based on L1 RoIs)
2) Execute a simple sequence of algorithms in each view
3) Merge the results

Current status

EventViews implement the IProxyDict interface, and can be found in the package Control/AthViews

The package currently contains a PIMPL class called View, and a single implementation of view behaviour called SimpleView

Allows for alternative implementations with performance features like caching, as discussed earlier in the year, but for now I'm just focussing on functionality

DataHandles have been modified (in VarHandleBase) to accept an IProxyDict pointer and use it in place of StoreGateSvc if it is non-zero

So, if an algorithm uses DataHandles, it should be able to use EventViews

Current status

The proposed workflow runs correctly, and you can try out v1 of the EventViews demonstrator for yourself

Instructions on the twiki page:
https://twiki.cern.ch/twiki/bin/viewauth/Atlas/EventViewsDemonstrator

Will run multithreaded in AthenaHive
athena --threads=4 dflow_jobo.py

The demonstrator is a self-contained package, and there is now a compatible version of StoreGate (03-06-14-02) which should be merged into the main codebase

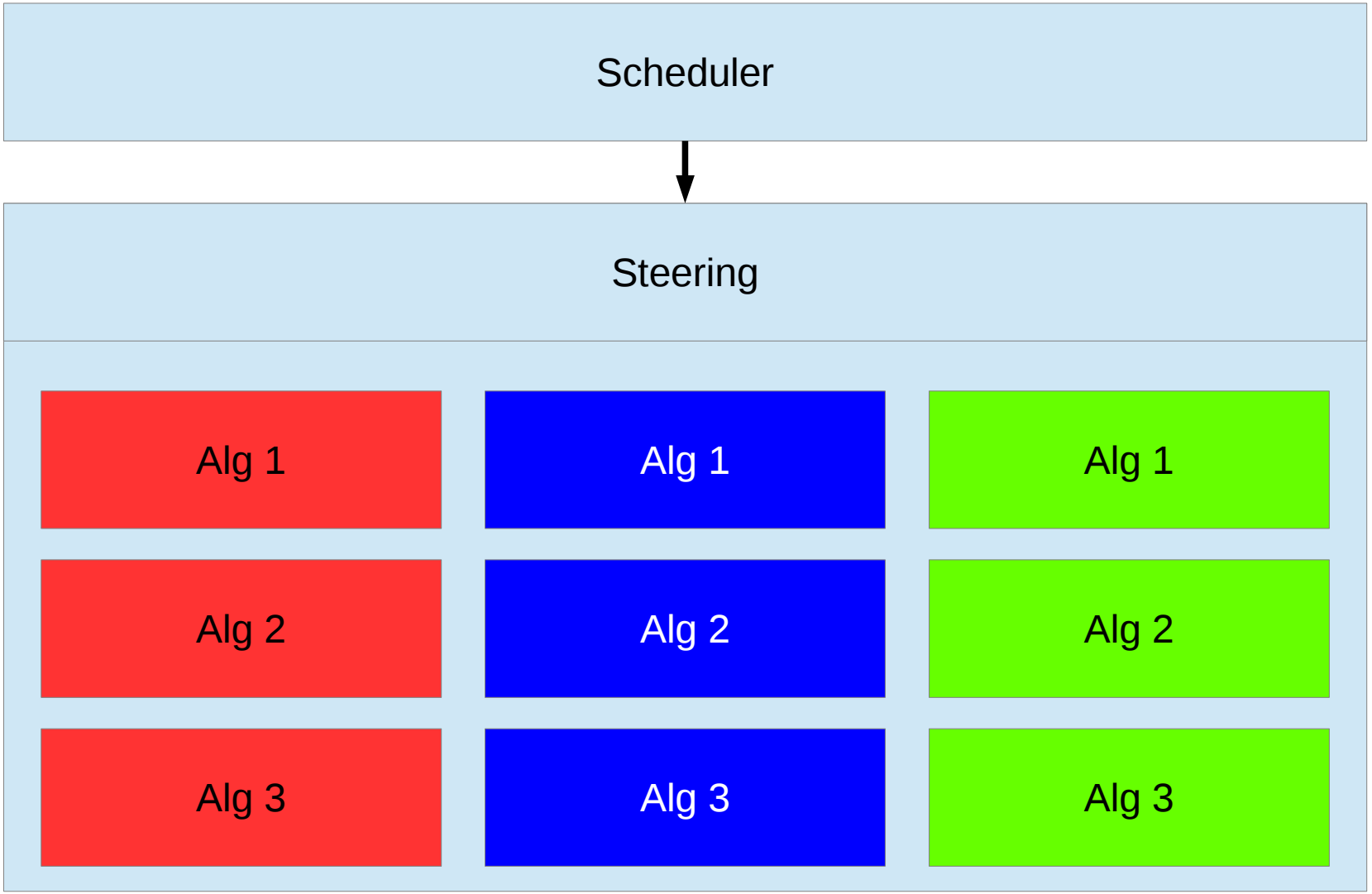
However, the creation and scheduling of the views is artificial in v1

- We want to create an arbitrary number of views, based on L1 info, and have the scheduler include them
- Currently the views are fixed, defined in the configuration file

I've considered three possible solutions...

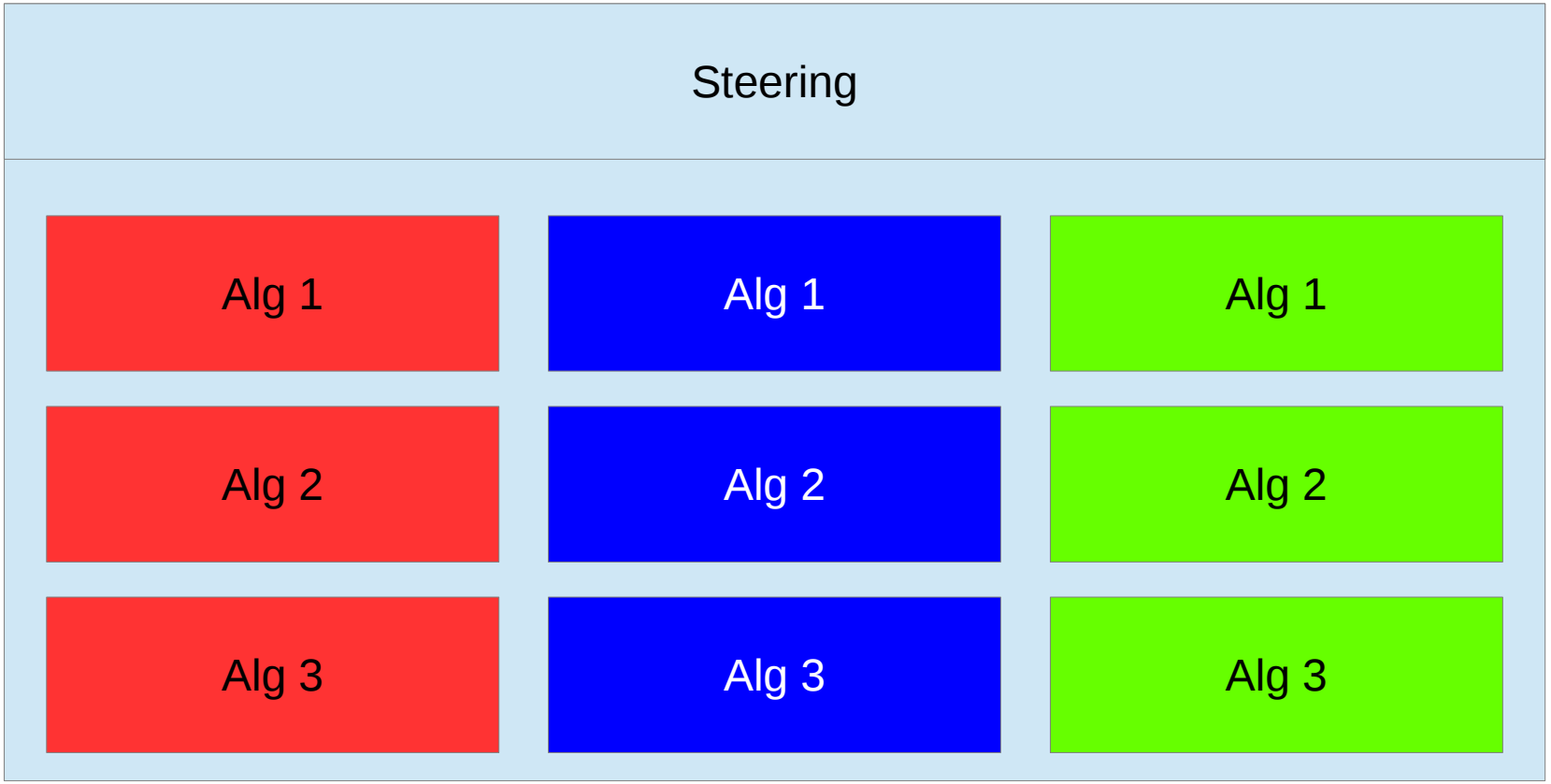
Separate HLT scheduler

Since the new framework allows for multi-threading within algorithms, we just run a single steering algorithm that handles all the processing within views



Separate HLT scheduler

In fact, the steering could replace the scheduler entirely, since this is a modular component of the framework



Separate HLT scheduler

In fact, the steering could replace the scheduler entirely, since this is a modular component of the framework

PRO:

- Requires ~no change to the Hive scheduler
- Conceptually similar to current HLT
- Merge step is simple

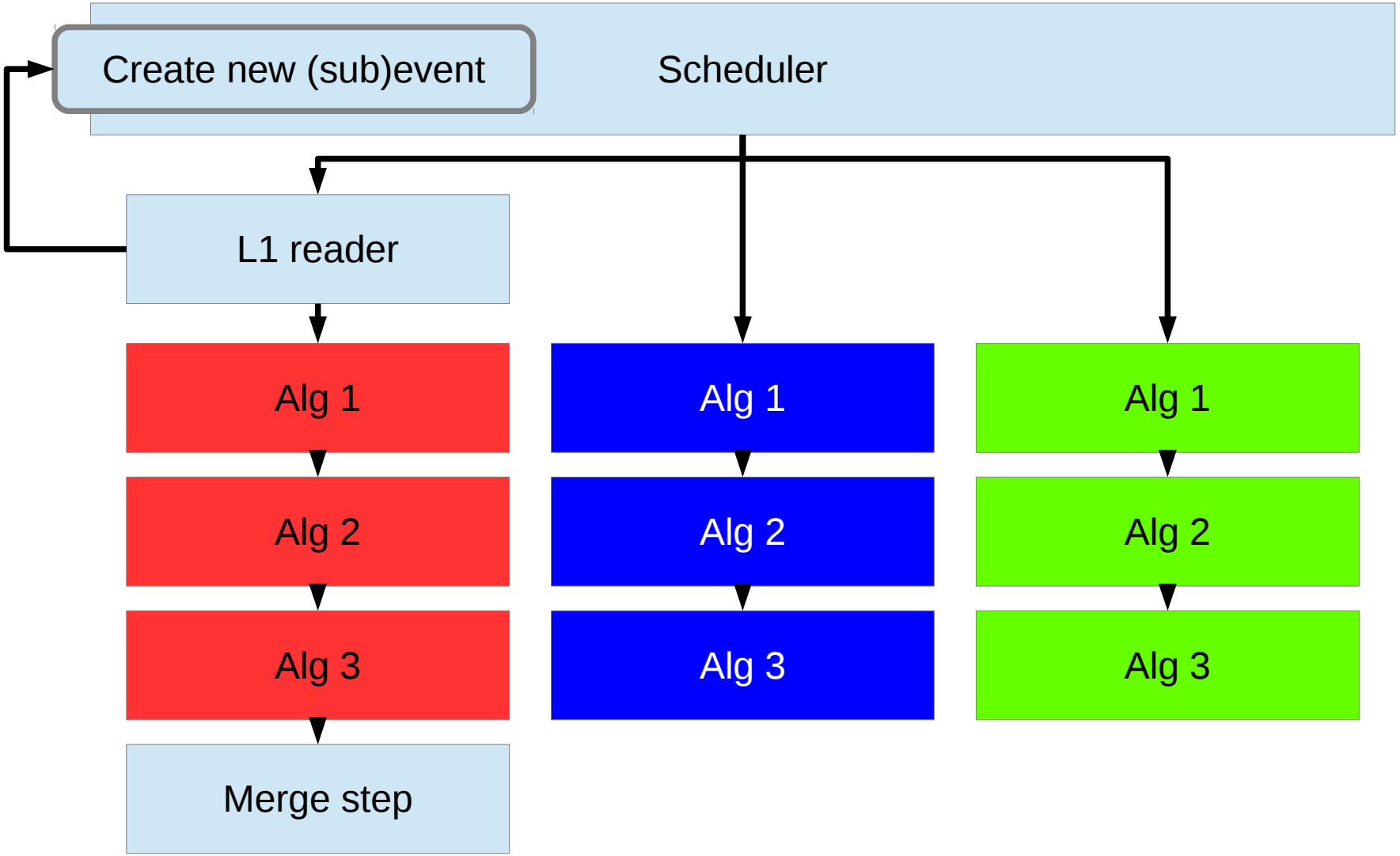
CON:

- Does not use any of the functionality of the Hive scheduler
- Requires a separate HLT implementation of multithreaded steering (although Tomasz has already experimented with this)
- Not really compatible with our goal of offline and HLT sharing a framework

LAST RESORT

Full integration

Simply create event views based on the L1 regions of interest, and have the scheduler treat them as sub-events with their own data flow



New components

AthEventContext

- Inherits from EventContext
- Stores IProxyDict pointer to EventView
- Also adds virtual destructor to allow dynamic_cast – requires modification to base class

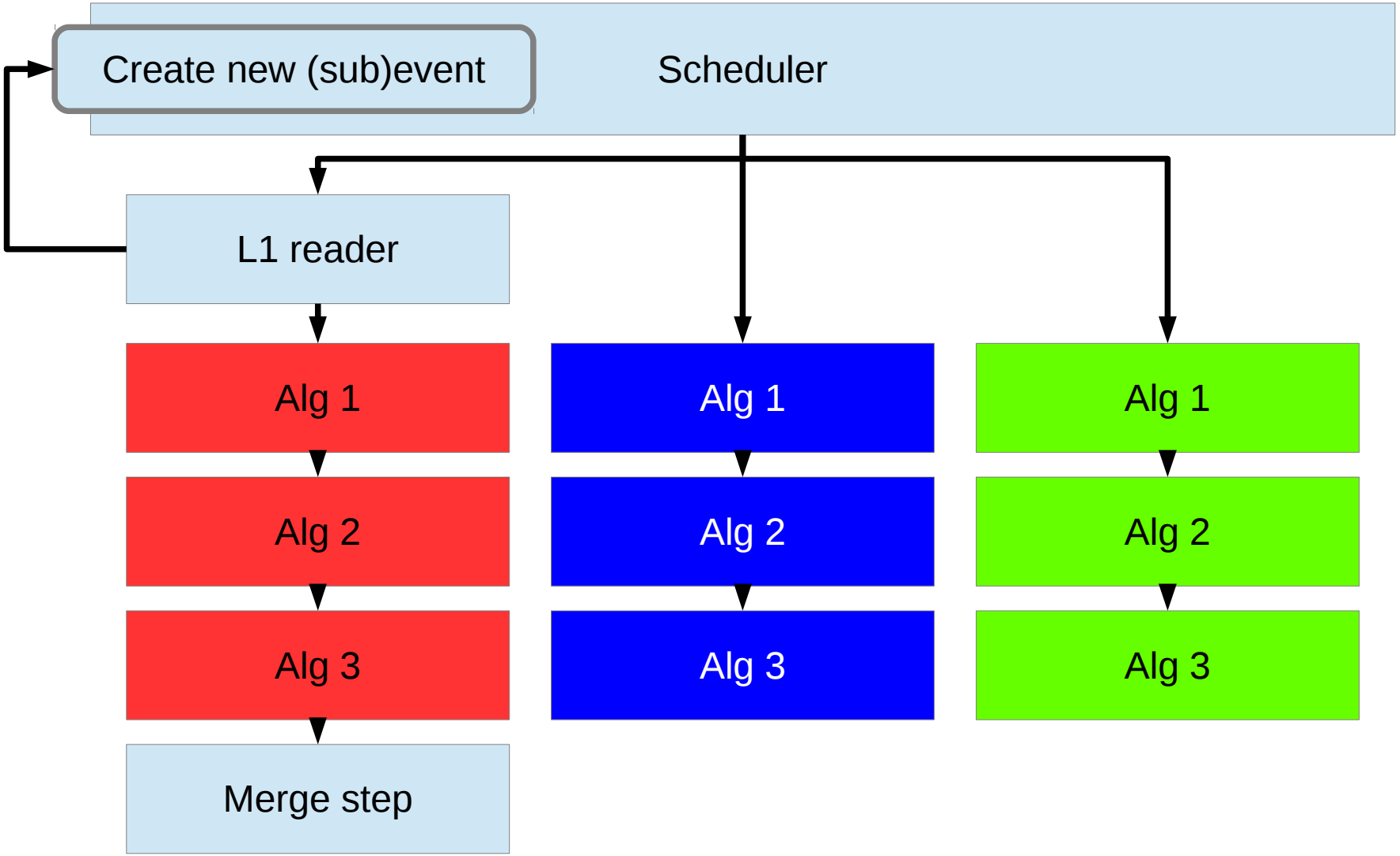
AthViewAlgorithm

- Inherits from AthAlgorithm
- Adds method to return pointer to current EventView (or zero if using SGSvc)
- sysExecute() changed to dynamic_cast the EventContext, retrieve the view if there is one and set the DataHandles to use it
- Also a little hack: see next slide

Perhaps just modify EventContext directly, to avoid the dynamic_casts?

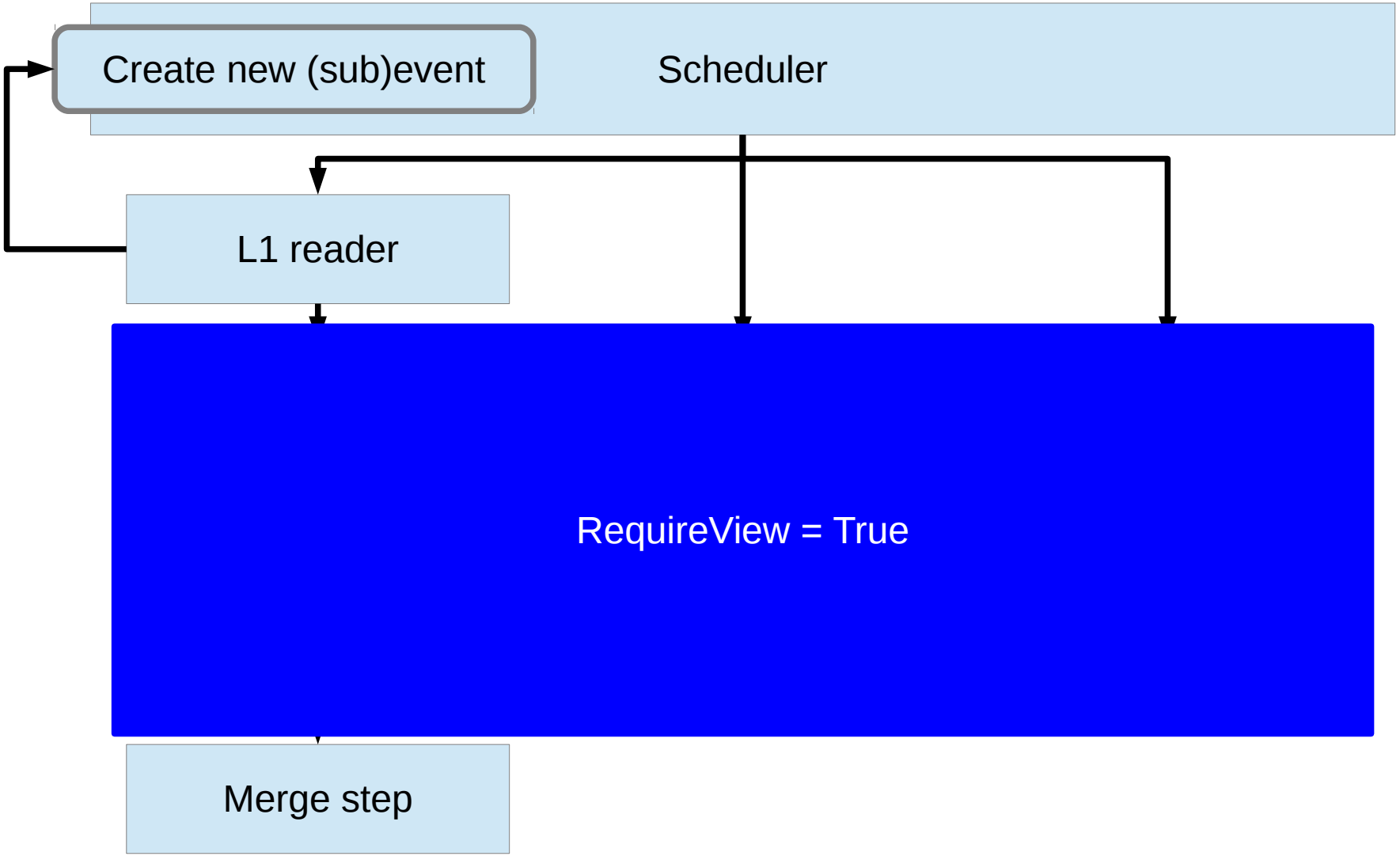
Full integration

Also included a bit of a hack, since all algorithms are scheduled to run on all (sub)events:



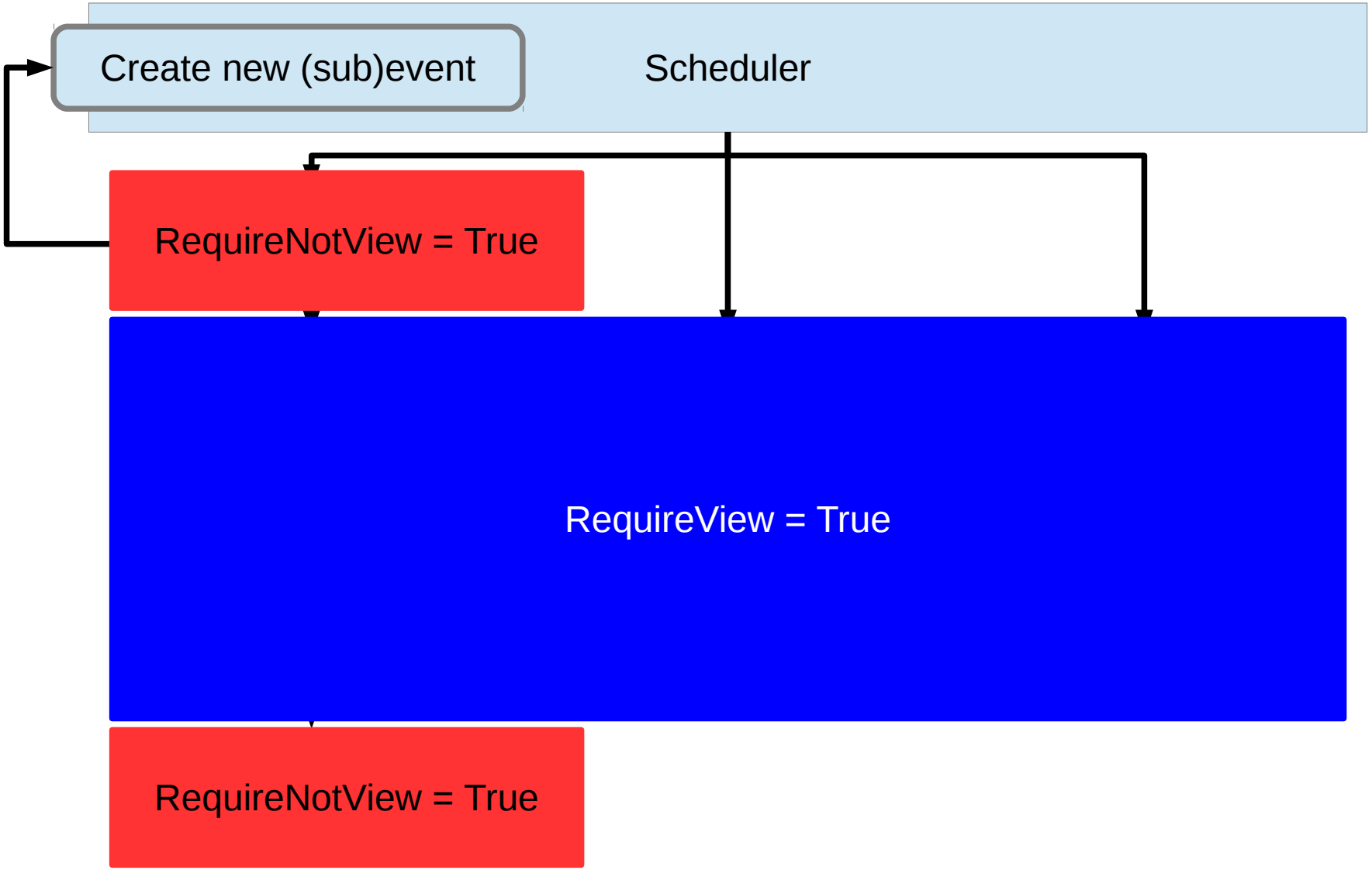
Full integration

Prevents view algorithms from also running on the whole event (i.e. negating the purpose of RoI processing)



Full integration

Prevents creation of infinitely branching tree of views



Full integration

Simply create event views based on the L1 regions of interest, and have the scheduler treat them as sub-events with their own data flow

PRO:

- We're aiming for full integration of offline and HLT
- Can use all the features of the Hive scheduler, running algorithms in parallel within views based on data flow
- Ideally requires minimal/no modification of the scheduler

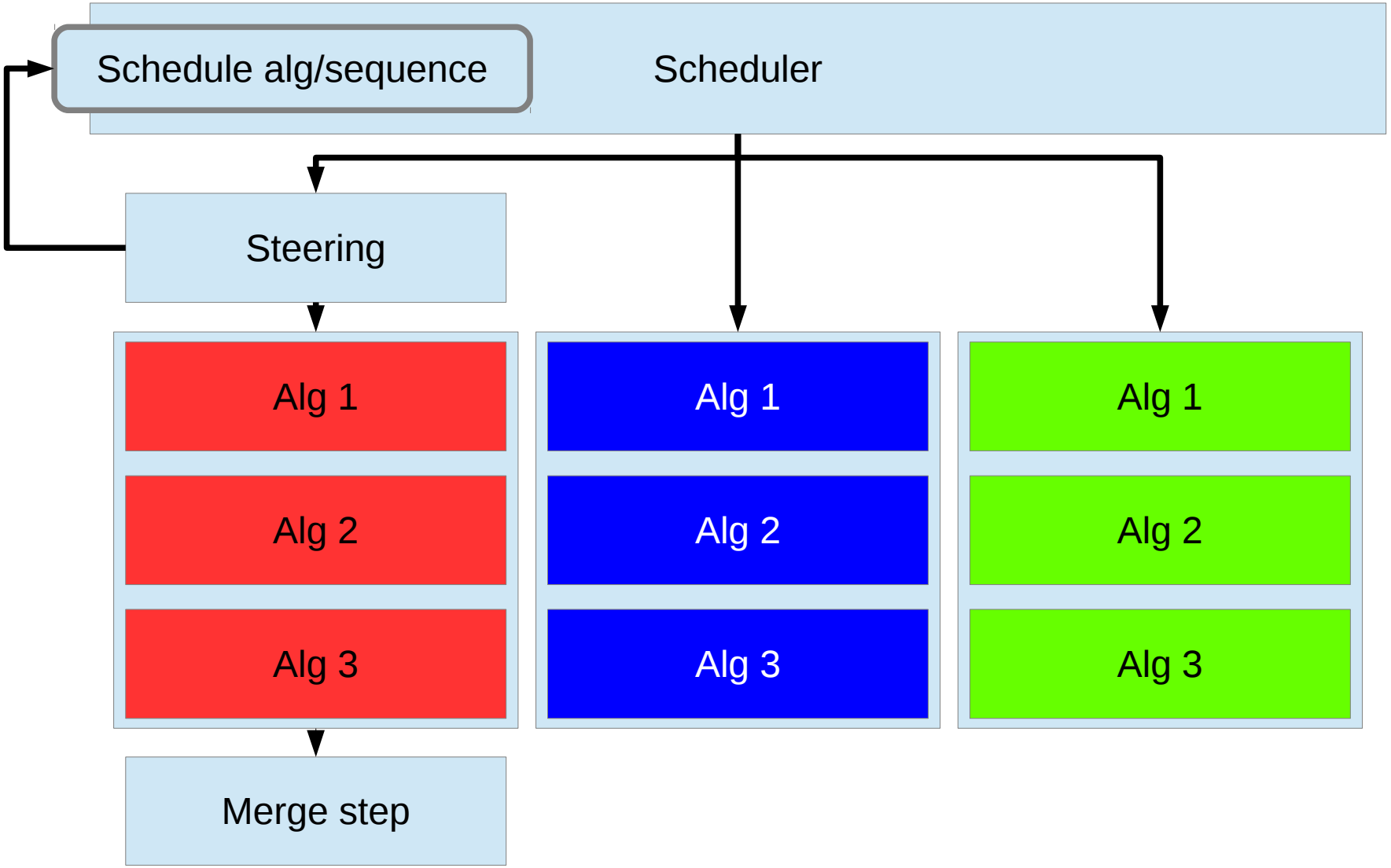
CON:

- At the moment a new (sub) event implies a new copy of StoreGate, which is potentially inefficient
- The merge step is impossible without significant design changes

INVESTIGATE

Somewhere in the middle

Have a steering algorithm that defines sequences to run in views, but let the scheduler manage the execution of these sequences



Somewhere in the middle

Have a steering algorithm that defines sequences to run in views, but let the scheduler manage the execution of these sequences

PRO:

- Requires a modification to the Hive scheduler, but it's a feature we need anyway
- Uses the multithreading capability of the scheduler
- Merge step is straightforward

CON:

- Any multithreading more sophisticated than “1 thread per view” will require extra work
- Relies on implementation of sequences

INVESTIGATE
Preferred solution?

Modifying the scheduler

The concept of “schedulable incidents” means that we need a way of informing the scheduler that there is work to be done:

- High priority
- No upstream dependencies: if the incident is scheduled, it should be run
- Could potentially occur multiple times per event (or zero)

I think it's very important to get a clearer idea of how schedulable incidents will behave

Will incidents modify the scheduler in a way that's useful for EventViews?

Testing the scheduler

Separate scheduler approach: avoiding this for now

Full integration approach: issues to solve

I have the easy bit working – the “L1 reader” algorithm can create a load of event views as sub-events and pass them to the scheduler
However, I don't know how to do the merge step properly

Middleground approach: very rough, but essentially working

The “L1 reader” algorithm is able to inform the scheduler that a particular algorithm should be run in a particular view
- Required an extra method in the IScheduler interface
- Takes an existing algorithm from the pool
- Creates an AthEventContext to pass the EventView pointer
- The algorithm will not be marked as EVTACCEPTED so it can be re-used
- No sequences yet, but these are on the framework to-do list

Plan

End 2015: functional prototype with views

- Extend current simple prototype: Tomasz & Ben
  - Create views from L1

- Extension of GaudiHive scheduler to support views: Tomasz/Ben/Francesca
  - Need to coordinate with GaudiHive devs
  - Precise design open to debate

- Add 1 or 2 real algorithms: looking for volunteers
  - Trigger algorithms to inherit from AthAlgorithm directly
  - DataHandle migration

- Develop online use-case: integration into athenaHLT: Werner?

End 2016: wider set of algorithms

- Simple menu with chains
- Real HLT-format output
- Additional functionality e.g. Monitoring (validation & cost)

Summary

Working demonstrator for the views' basic functionality
- Try it yourself, no code editing required!
- Opportunity to test different implementations of the view object
  - Current “SimpleView” just a thin layer on top of StoreGate

Busy developing interaction between views and scheduler
- Two potential approaches (and one way to dodge the issue)
- “Ideal” approach is awkward in practise
- Compromise approach has a lot to recommend it (thanks to Pere for the inspiration)

Real algorithm migration needs some work
- For now I'm just storing arbitrary numbers with dummy algorithms