Event views

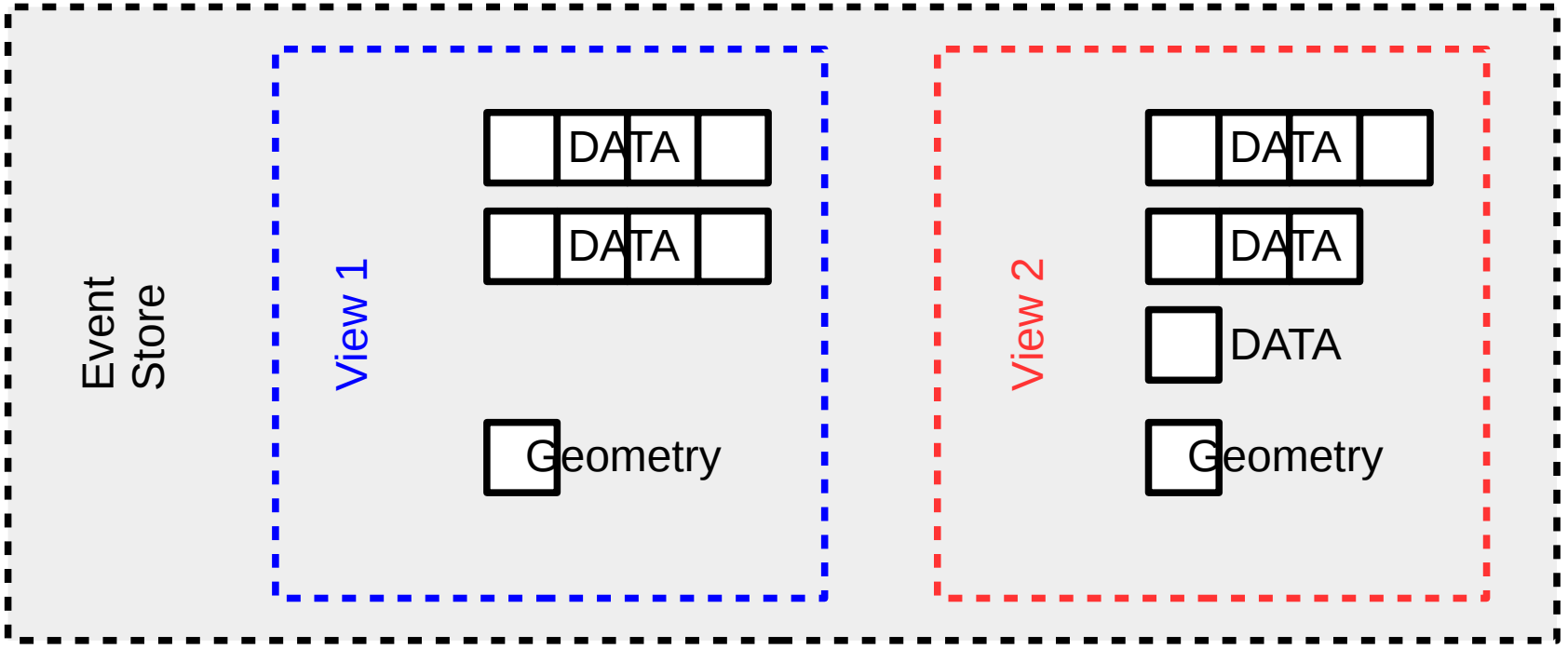
B. Wynne

06/06/16

Event views

The event views are the mechanism used to provide a subset of the whole event data to an algorithm, without the algorithm needing to be modified
- identical code can be used in HLT and offline, with no wrapper

A view presents the same interface as the EventStore, and is no more or less than that: any geometry or provenance information must be added as a data object within the view
- a view is not necessarily an RoI or a TriggerElement

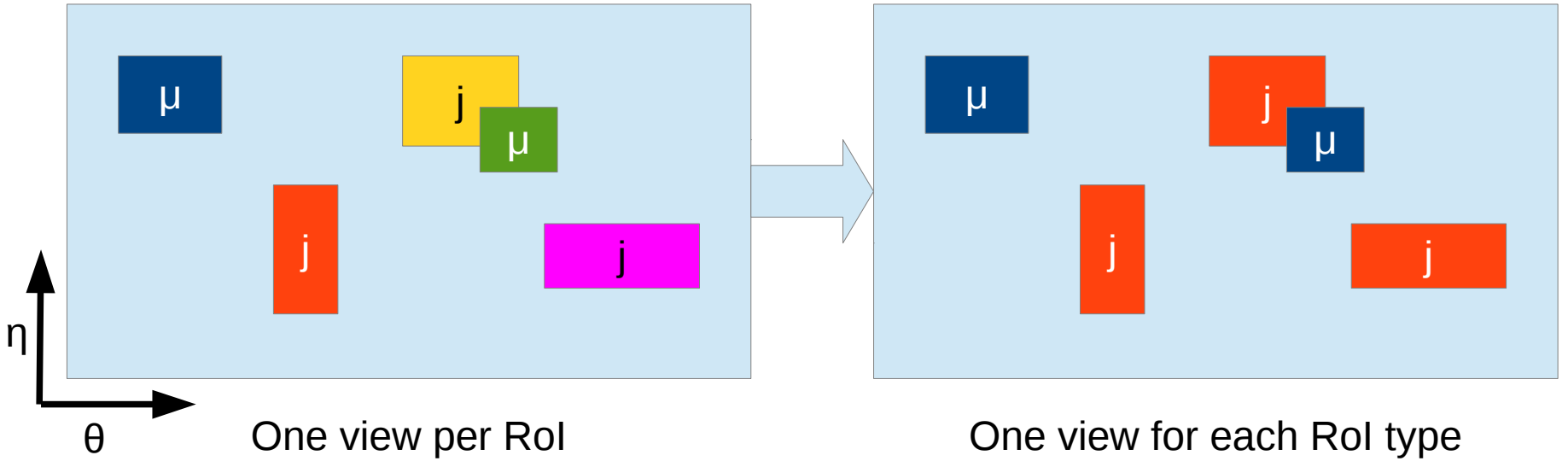


Event views - plans

Long term: we need to decide precisely how we are going to use views

Short term: probably the best way to figure that out is to try using them more

We're experimenting with having one view encompass all of a particular type of reconstruction (mu, egamma, etc.)



One view per RoI

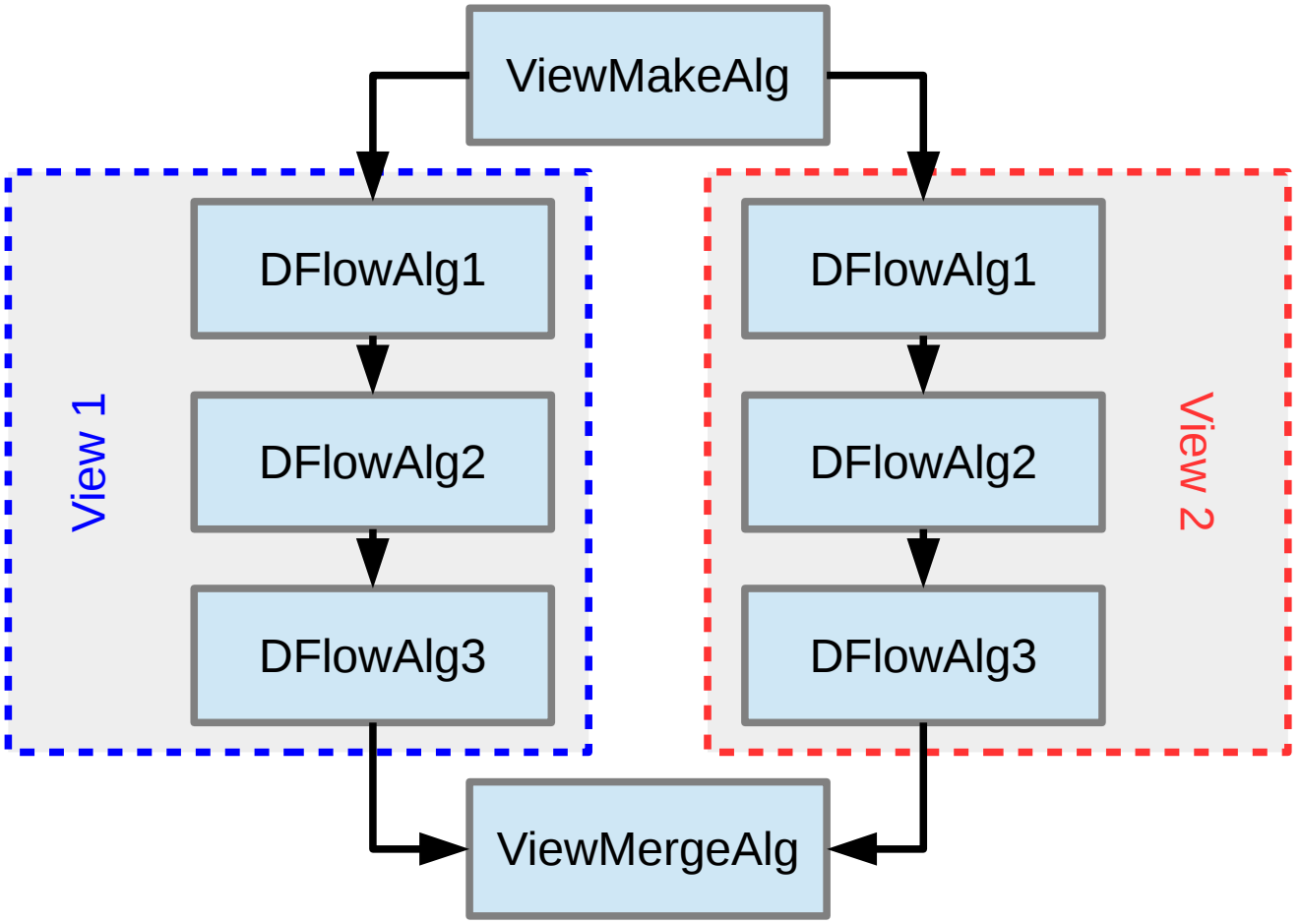
One view for each RoI type

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

STATIC VIEWS:
Can be defined in configuration

Event views demonstrator

The event views package (Control/AthViews) is part of the 20.8.X build and contains three demostrators that all run the following workflow:



Event views demonstrator

You can run each demonstrator like this:

```
asetup 20.8.X-VAL,rel_6,here
athena --threads=N AthViews/StaticViews.py
```

This one tests EventViews directly, with the algorithm code updating its own DataHandles to use a particular view:

```
StatusCode sc = myHandle.setStore( myView );
```

It doesn't look very much like what we'd actually want to do with views, but it gives you an idea of what's needed to make them run

Event views demonstrator

You can run the next demonstrator the same way:

```
asetup 20.8.X-VAL,rel_6,here
athena --threads=N AthViews/StaticMigrationViews.py
```

Note the “Migration” part: this shows a simple (but not final!) approach for using views with existing code

In the JO file, you tell an algorithm to run in a view by setting its view name property:

```
job += CfgMgr.MyAlgorithm("my_algorithm_instance")
job.my_algorithm_instance.ViewName = “view1”
```

Algorithm differences

To try out views this way, you need to use a new algorithm base class:

```
#include "AthViews/AthViewAlgorithm.h"
class MyAlgorithm : public ::AthViewAlgorithm
```

You also need to make a small addition to the execute method:

```
StatusCode MyAlgorithm::execute() {
      UseView();
```

Both this and the previous demonstrator show STATIC views, i.e. views that can be defined at configuration time and do not change

Dynamic views

We might still want to create DYNAMIC EventViews, based on event information, rather than statically at configuration time

This is shown in the third demonstrator, which prototypes a way to set algorithms to run in one or more views without specific configuration:

```
asetup 20.8.X-VAL,rel_6,here
athena --threads=1 AthViews/GraphViews.py
```

This only requires the change to the algorithm base class

Note that here, it will only work with one thread

Algorithm base class

View behaviour is introduced via DataHandles

Therefore the algorithm base class needs to call the setStore() method for the DataHandles before execute() begins

At the moment I do this in sysExecute()

Areas that need work:

- Moving this behaviour from the test AthViewAlgorithm base class, into the standard algorithm base class (get rid of UseView() method)

- Making this work with re-entrant algorithms (doubt it'll be a show-stopper)

This is all that is needed for STATIC views. Discussion from here on concerns DYNAMIC views only

DYNAMIC VIEW DISCUSSION

Graph execution task

As I've mentioned before, dynamic views break a fundamental assumption of GaudiHive/AthenaMT, namely that an algorithm is executed exactly once per event

To get around this I have implemented a GraphExecutionTask, in contrast with the AlgoExecutionTask that is used otherwise

- Inherits from tbb::task

- Sets algorithm context, and calls sysExecute()

- Ignores algorithm states and data-dependencies

- Acquires ownership of algorithm pointer at the moment of execution, not beforehand (reschedules self if algorithm unavailable)

- Supports subgraph of algorithms (currently just a sequence)

This component raises a lot of questions...

Scheduler stall behaviour

The first thing to note with the dynamic views demonstrator, which uses GraphExecutionTasks, is that it doesn't work with more than one thread

This is because the GraphExecutionTasks exist outside the scheduler state machine, and it has no idea that they are running

So, it can reach a point where there are no algorithms left to schedule except for those that depend on the view results. Since it does not know that the GraphExecutionTasks are running, it assumes that the dependencies cannot be satisfied and declares a stall

Can/should we introduce a mechanism for an algorithm to tell the scheduler that some task it doesn't know about is running?

There's possibly a connection here to GPU offloading: we're waiting for work happening “elsewhere”

NECESSARY for dynamic views

Views and data dependencies

I've been banging on about this for years now, but it's still an open question:

How do we set up data dependencies for an algorithm that merges input from an unknown number of views?

It's an algorithm that

- Depends on an unknown (at config time) number of inputs

- Must not be scheduled until every such input that will be made, has been made (i.e. it can't run when the first view is finished, it must wait for all views)

A simple but unpleasant solution is for the algorithm to block internally until it can read all the data it wants – can we do better?

Related, perhaps to the discussion Adam has been having about optional data dependencies

IMPORTANT for dynamic views

The “I Will Create” notification?

I think I have a solution (untested: recent idea) to both of these problems

We need to tell the scheduler that there is work ongoing but not complete, and we need to have an algorithm wait for an arbitrary amount of input

Perhaps we can solve both by having algorithms able to tell the scheduler that a particular data object will be created at some unknown point in the future

- Scheduler doesn't declare a stall if there are pending “I Will Create” flags

- View merging algorithm can be postponed until all pending flags of the correct name and type are cleared (potentially still misses a very late declaration)

Algorithms that exist but aren't scheduled

Another part of ignoring the state machine is that any algorithm that has been instantiated will be run on the whole event, as well as in the dynamic view

Marking the algorithm as “executed” in the view is not sufficient, because it might be scheduled before the view is processed (currently no way to do it anyway)

At the moment I use the base class to introduce a “RequireView” property – if an algorithm is run outside of a view it just returns SUCCESS immediately

Obviously this is a little hacky, and might get inefficient if there are very many view algorithms: I've shown in the past that scheduling a lot of do-nothing algorithms can cause problems

Can/should we create a way to add algorithms to the AlgPool without giving them to the scheduler?

WOULD BE NICE for dynamic views

Algorithm dependencies within views

At the moment the GraphExecutionTask takes a list of algorithms in its constructor, and executes them in order

It might be better/more useful to use the data dependency information already collected to set this up

Depends a bit on how we want to use the views, but potentially we might want

- Some set of algorithms (with deps?) from config file

- Arbitrary bag of algorithms (throw them in, let them sort dependencies)

- One algorithm and all possible downstream (collect everything)

- Everything needed to produce some output?

WOULD BE NICE for dynamic views

Summary and plans

Testing EventViews

- We have two possible models for using event views, dynamic and static
- Both have working demonstrators, ready to be used with migration code
- Priority to use them, and decide what we want to do

Static EventViews

- Defined at configuration time
- Sufficient for the approach with one view per type of reconstruction
- Only require modifications to the algorithm base class, minimal disruption

Dynamic EventViews

- Create 0-N of them per event
- One view per Region of Interest, much like the current HLT
- Require changes to the scheduler

Dynamic views are definitely more work, but might also be more useful, and some of the changes could be applicable beyond the HLT