ATLAS HLT and EventViews

B. Wynne

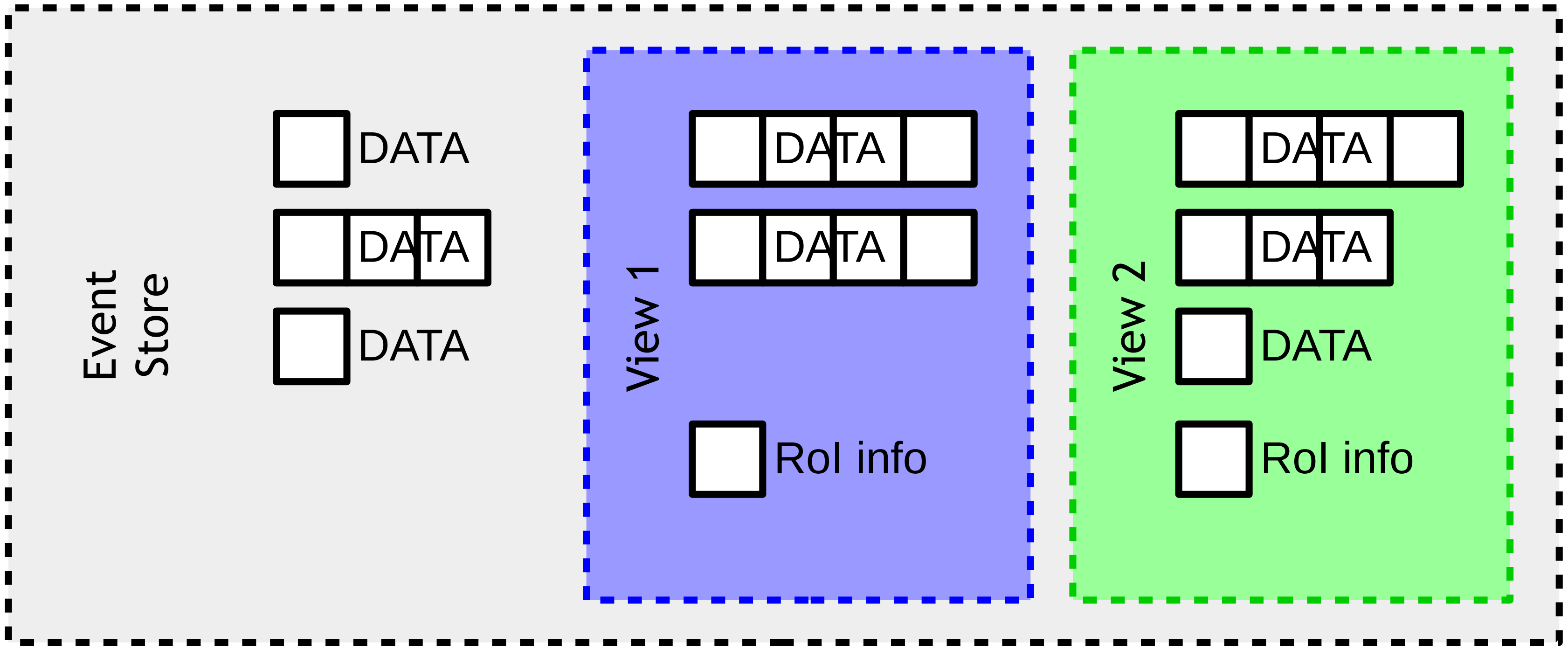
22/09/16

EventViews

Each EventView implements the same interface as the whole event store, and presents a subset of the data it contains

The views are intended to be general-purpose objects

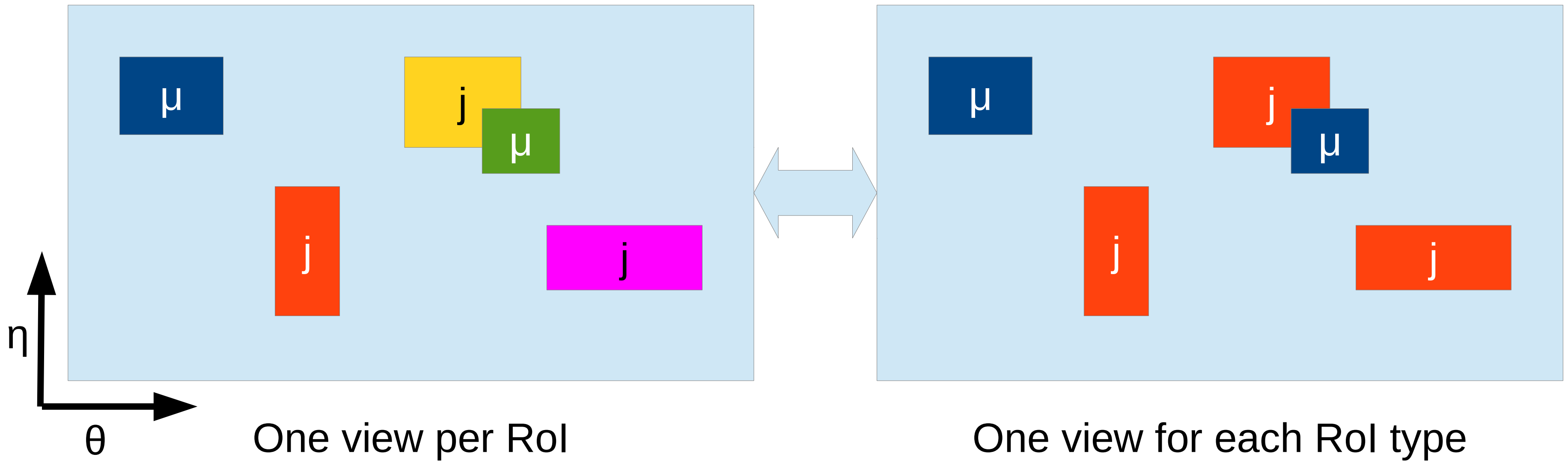
- They can contain data objects that describe a corresponding RoI
- Allows for potential alternative use-cases



Event views – open question

The HLT as it is today processes Regions of Interest (RoIs). Different numbers of RoIs are created each event, and algorithms can be executed multiple times per event, using a different RoI each time

We are investigating a more GaudiHive/AthenaMT-friendly approach: to merge multiple RoIs of the same type into single views that can be defined in advance, but this would be a significant, untested change for the HLT



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
Cannot support today's HLT

Issues with EventViews

1) How does an algorithm run in an EventView?

At the moment it looks something like this (Static views case):

GAUDI	ATHENA
	View class in Control/AthViews
	View implements IProxyDict
	(AthAlgorithm base class takes View pointer from EventStore)
	VarHandleBase has method setProxyDict

Issues with EventViews

1) How does an algorithm run in an EventView?

At the moment it looks something like this (Dynamic views case):

GAUDI	ATHENA
	View class in Control/AthViews
	View implements IProxyDict
EventContext has IProxyDict pointer (forward-declared)	
Scheduler passes EventContext to algorithm	
	(AthAlgorithm base class takes View pointer from EventContext)
	VarHandleBase has method setProxyDict

Issues with EventViews

2) Who runs the algorithms in the EventViews?

In the static views case, it's the scheduler as it currently stands
- The scheduler executes the algorithms
- The algorithm is configured to use a particular view

In the dynamic views case, it's a little more open...

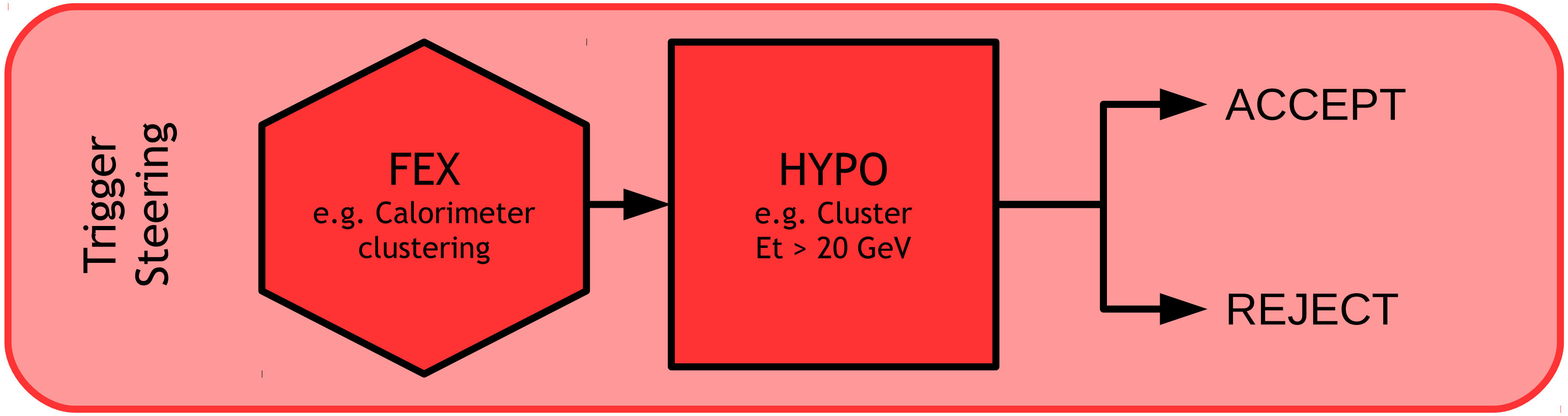
Trigger menu

In the HLT-specific layer of the old Athena framework, the scheduling of algorithms and resulting event accept/reject decisions were made by the “steering” class

The AthenaMT scheduler replaces steering, but does not take trigger decisions

A decision is made in three stages:
1) Feature EXtraction (FEX) algorithms reconstruct detector data
2) Hypothesis (HYPO) algorithms apply selection criteria
3) Passed/failed hypotheses compared to trigger “menu” to select events

The first two stages were handled by algorithms, the third by the steering itself



Trigger menu

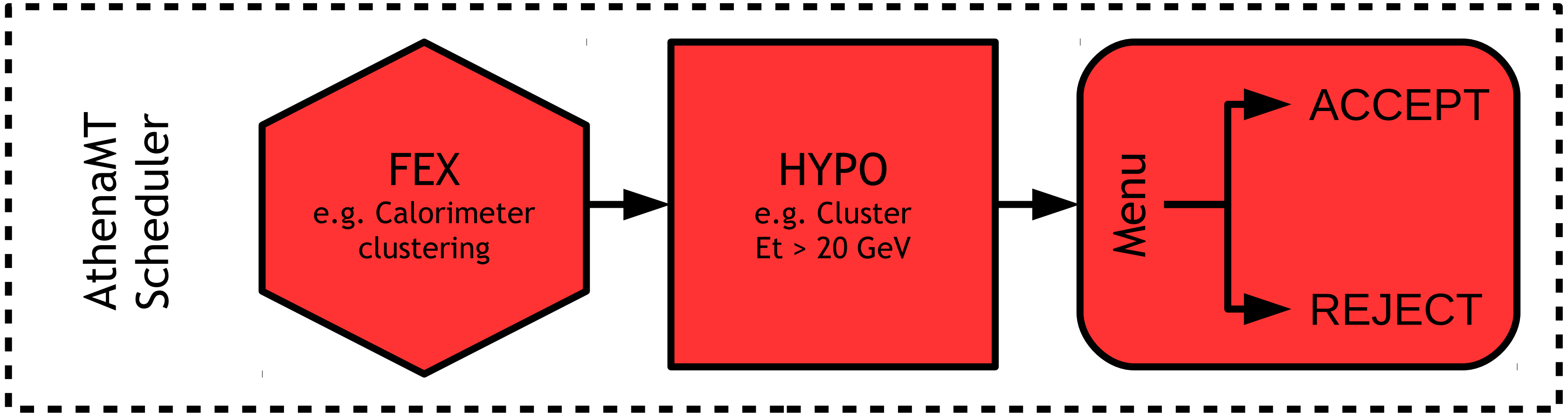
In the HLT-specific layer of the old Athena framework, the scheduling of algorithms and resulting event accept/reject decisions were made by the “steering” class

The AthenaMT scheduler replaces steering, but does not take trigger decisions

A decision is made in three stages:
1) Feature EXtraction (FEX) algorithms reconstruct detector data
2) Hypothesis (HYPO) algorithms apply selection criteria
3) Passed/failed hypotheses compared to trigger “menu” to select events

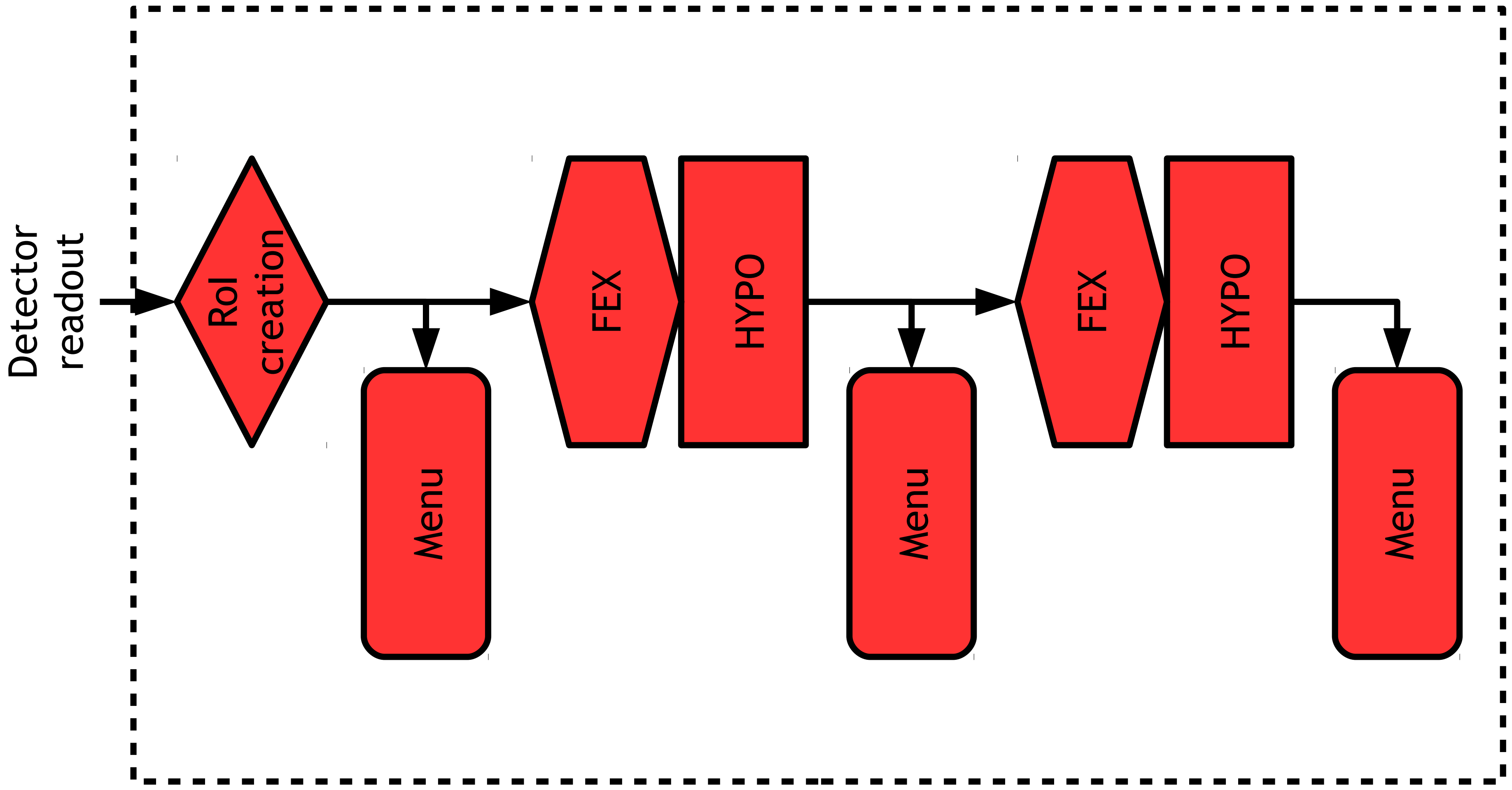
The first two stages were handled by algorithms, the third by the steering itself

We now introduce menu algorithms, fully replacing the steering



Menu algorithms

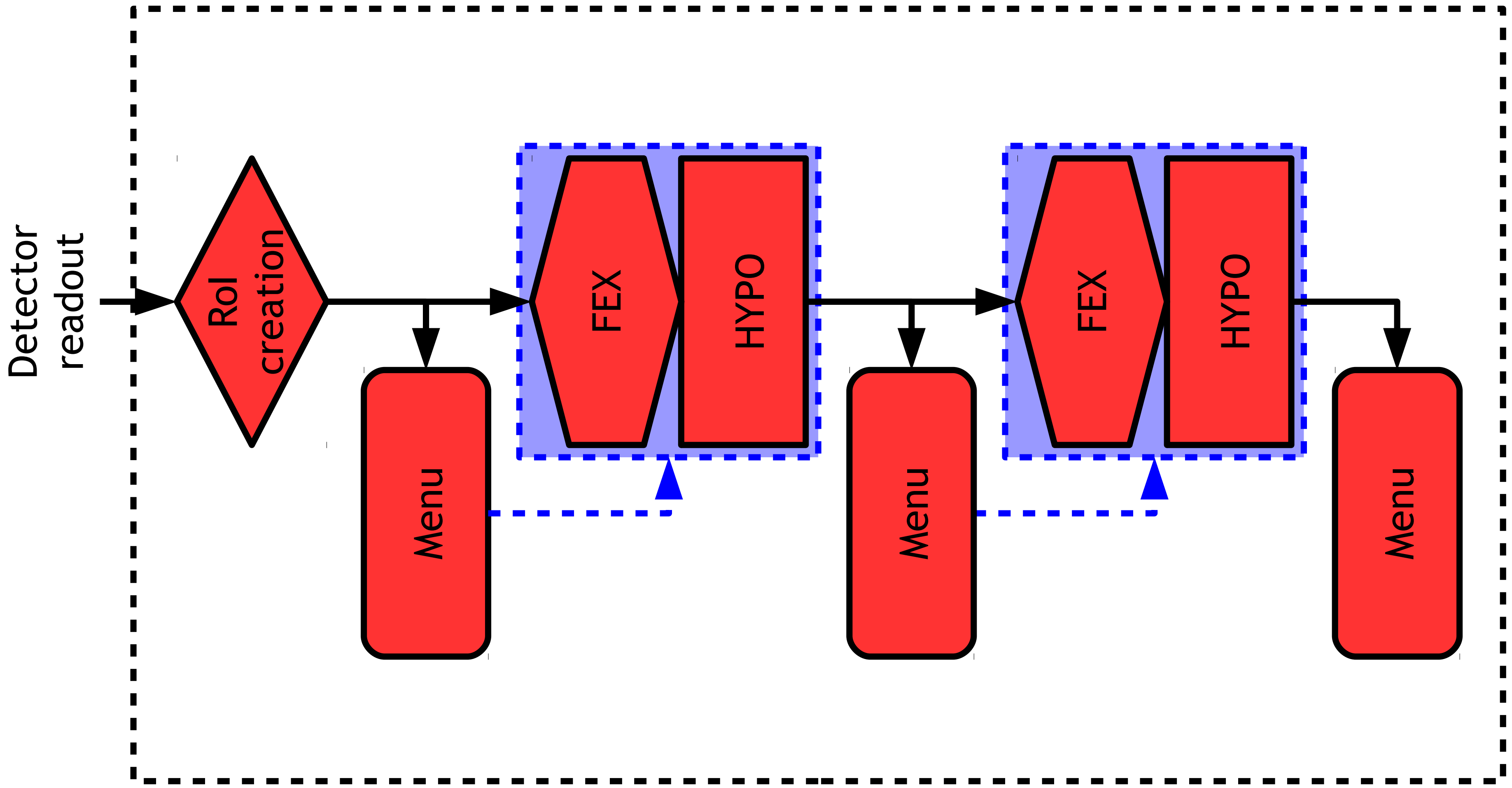
To provide early rejection, menu decisions are taken in several stages, with FEX and HYPO algorithms scheduled in between



Menu algorithms

RoI information is read in from the detector and used to create and populate EventViews

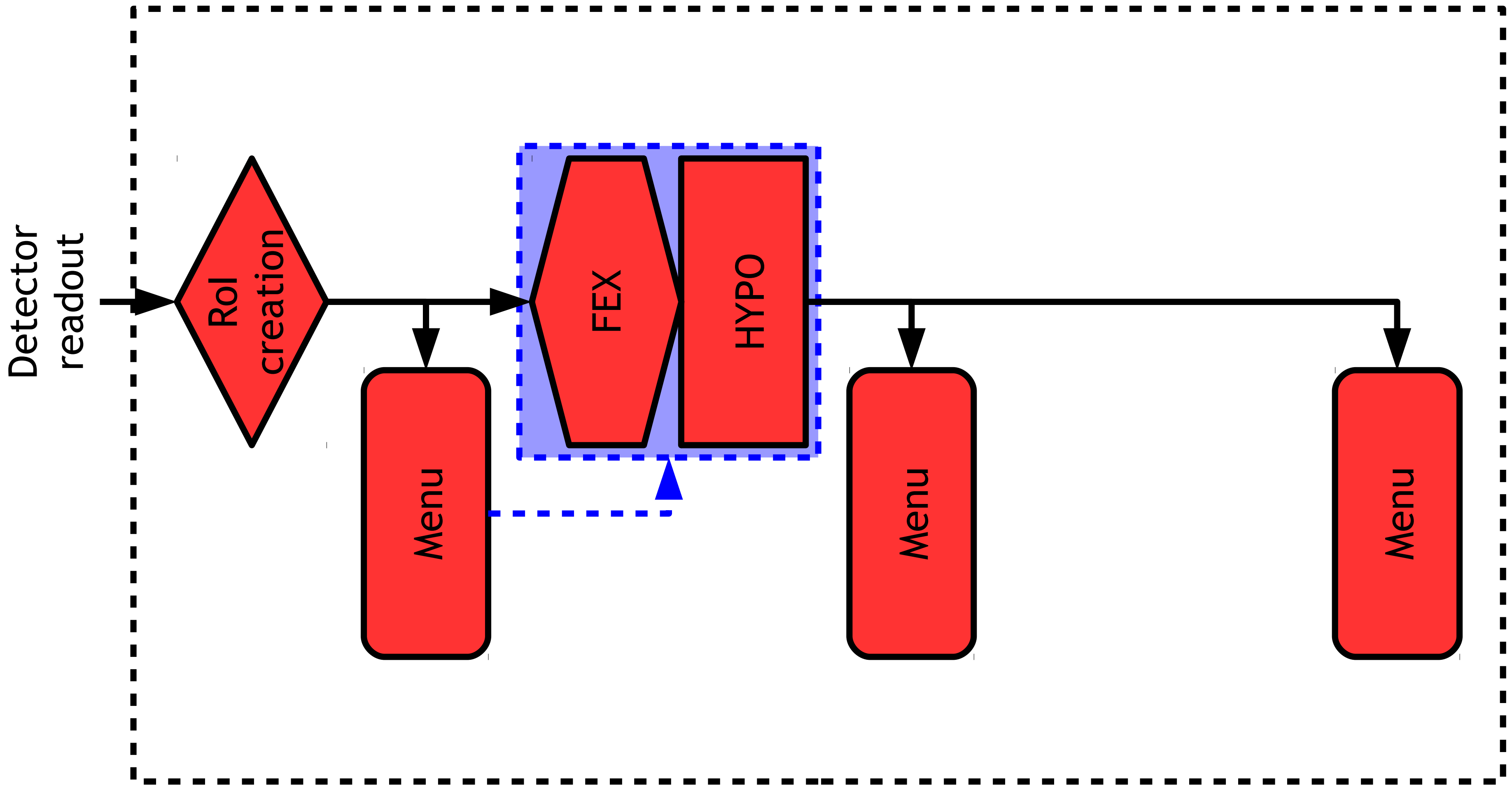
The menu algorithms are responsible for assigning FEX and HYPO algorithms to an appropriate view



Menu algorithms

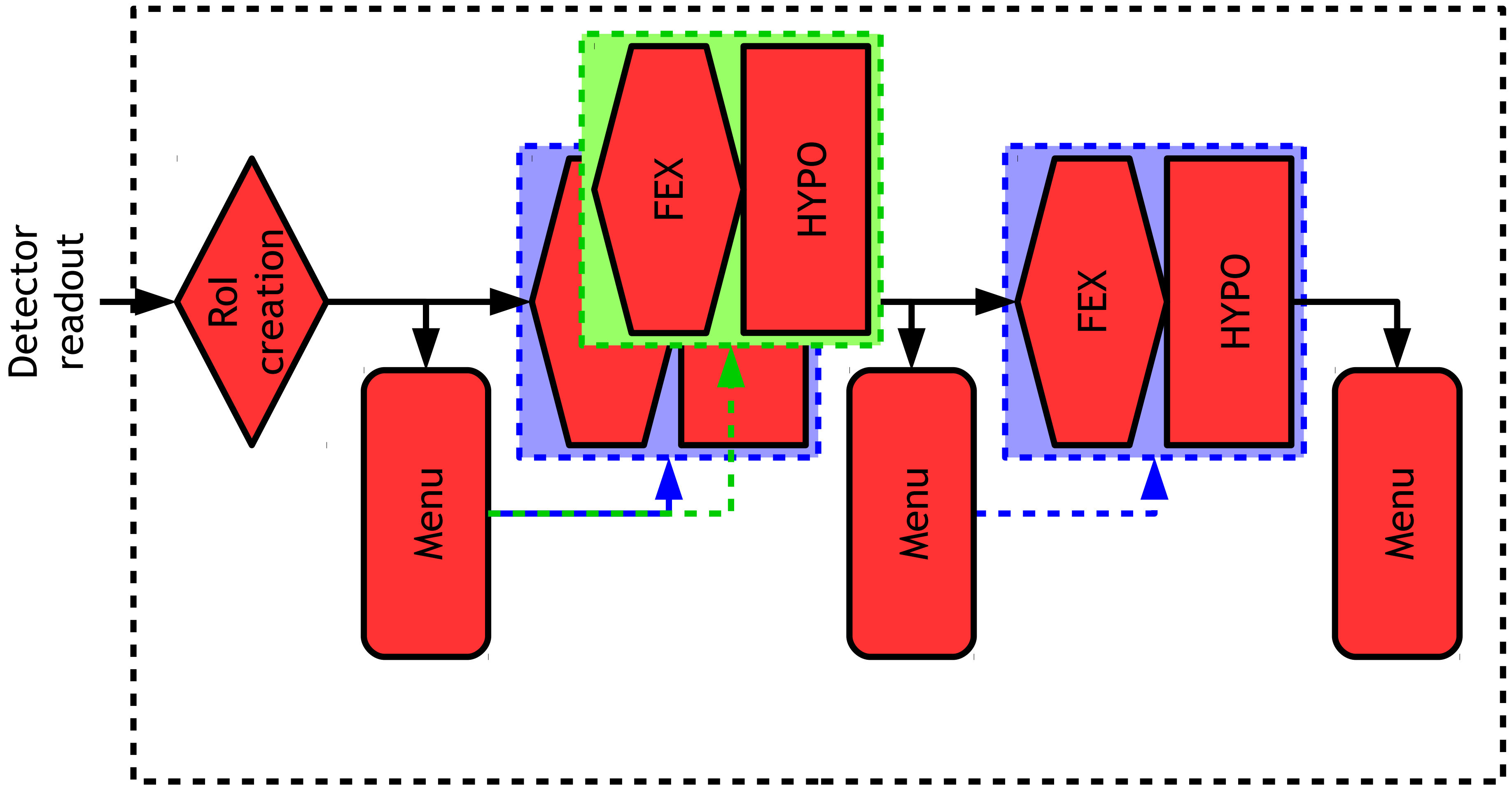
If there is no appropriate RoI, or if a decision is taken to reject the event, then FEX and HYPO algorithms are never scheduled

FEX and HYPO algorithms are configured never to run on a whole event, and are skipped by the AthenaMT scheduler



Menu algorithms

The menu algorithms prompt the scheduling of FEX and HYPO algorithms, allowing them to be run multiple times per event where there are multiple RoIs to process



Issues with EventViews

2) Who runs the algorithms in the EventViews?

In the static views case, it's the scheduler as it currently stands
- The scheduler executes the algorithms
- The algorithm is configured to use a particular view

In the dynamic views case, it's a little more open...

We need a way to
- configure algorithms without them running each event
- consume new EventContexts (with attached Views) created by algorithms
- assign one or more algorithms to the context+view
- have a control-flow kind of dependency between the running views and a downstream algorithm that inspects outcomes

Early reject

In the dynamic views case, most algorithms do not run by default
- Early reject ~= choosing not to run more algorithms in views

In the static views case, we have the same (large) data flow graph for each event

Even just passing around dummy empty data objects is a pretty significant overhead

In either case – but particularly in the static case – there would be an advantage to being able to abort processing