ATLAS Future Frameworks

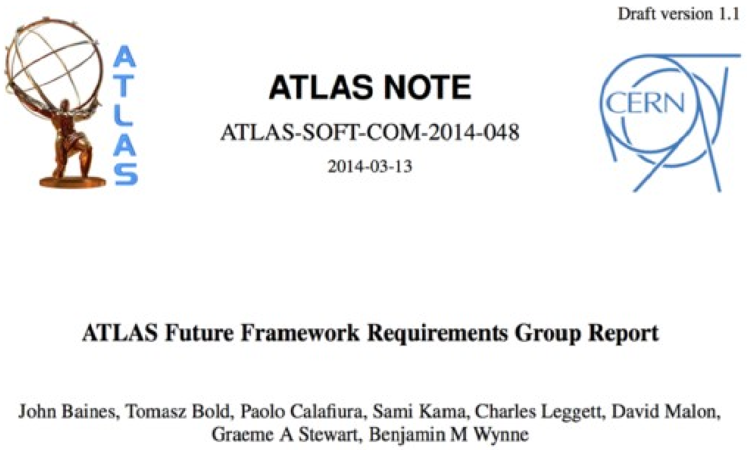
Ben Wynne
for the Future Frameworks Group and Software Project

(slides mostly stolen from Graeme)

23/09/15

Future Framework Requirements Group

- Established between TDAQ and Computing
  - Working in earnest from Spring 2014
- Examine needs of a future framework to satisfy both offline and HLT use cases
  - Can HLT requirements be absorbed more directly?
  - At the moment HLT layers a lot of functionality on top of Gaudi/Athena — could we do better?
- Reported in December 2014 (https://cds.cern.ch/record/1974156/)



Draft version 1.1

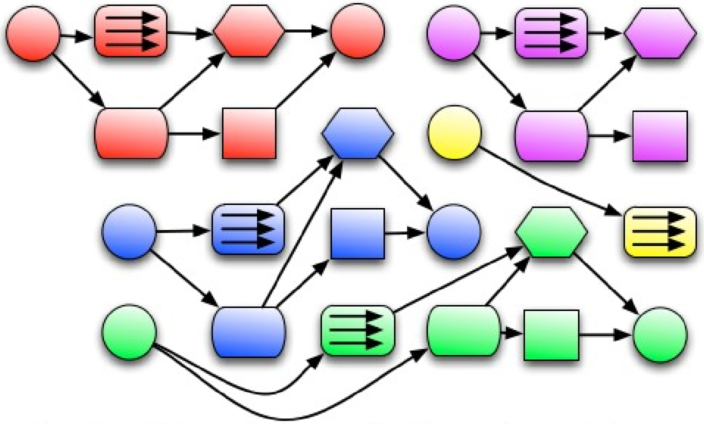
ATLAS NOTE

ATLAS-SOFT-COM-2014-048

2014-03-13

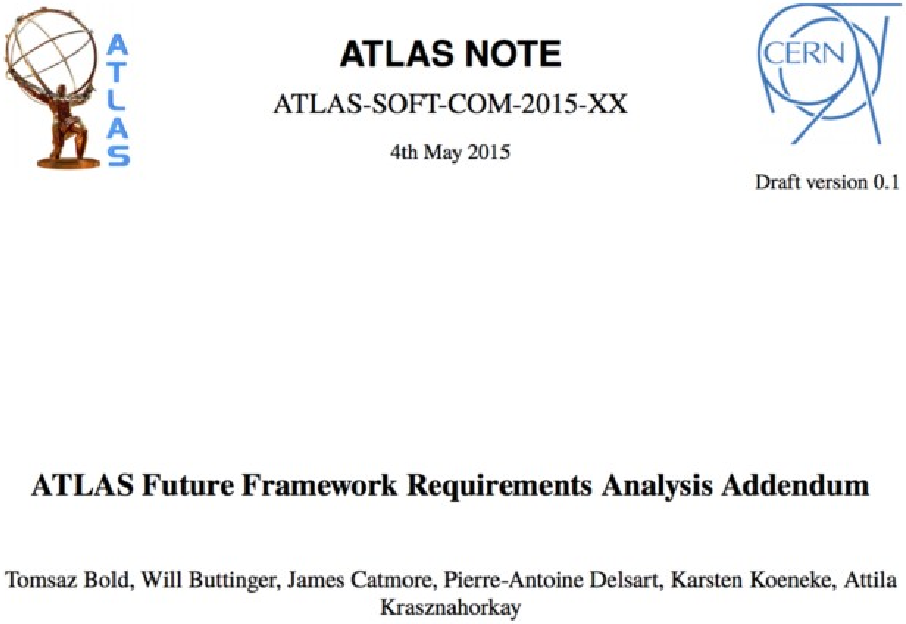
ATLAS Future Framework Requirements Group Report

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



Run3 multi-threaded reconstruction cartoon: Colours represent different events, shapes different algorithms; all one process running multiple threads

Analysis Addendum



ATLAS NOTE

ATLAS-SOFT-COM-2015-XX

4th May 2015

Draft version 0.1

ATLAS Future Framework Requirements Analysis Addendum

Tomsaz Bold, Will Buttinger, James Catmore, Pierre-Antoine Delsart, Karsten Koeneke, Attila Krasznahorkay

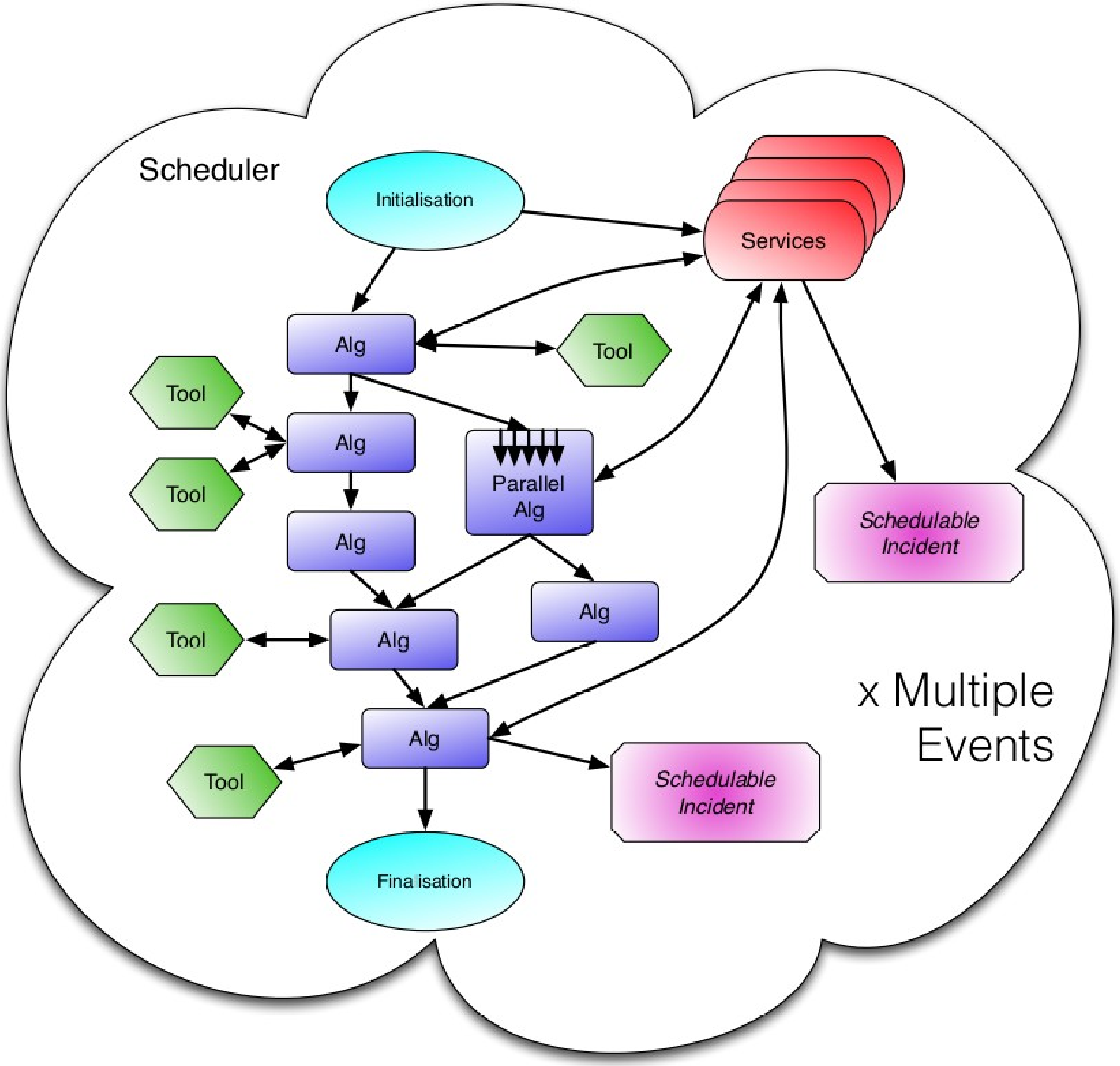
- Consideration of analysis use cases and convergence between physics groups' work and the mainline framework is important
- Small addendum task force recently reported on these aspects
  - Will help us to guide development in the right direction for the long term
  - Maximise skills cross-over between physics analysis, offline software and HLT

Overall Server costs and a Memory Wall

- Continued march of number of transistors leading to increasingly multi/many cores
- Gap between CPU cores an *affordable** memory is increasing
  - Current Xeon grid servers have 2-4GB of memory per core
    - WLCG Requirement is only 2GB, of course
    - Per hyper-threaded core this is halved: 1-2GB available, with 20-25% performance boost
  - Current Xeon Phi has 60 cores and 16GB of memory
    - ~256MB per core (64MB if running 4 threads/core)
      - Knights Landing has more (a lot more), but still needs to be paid for and cores are weaker than Xeon servers
  - Tesla K40 has 2880 cores and 12GB of memory
    - ~4MB per core
- Undoubtedly HEP needs to make efforts to lower its memory footprint, even to make best use of current hardware
  - Only way to go now is multi-threaded — new frameworks

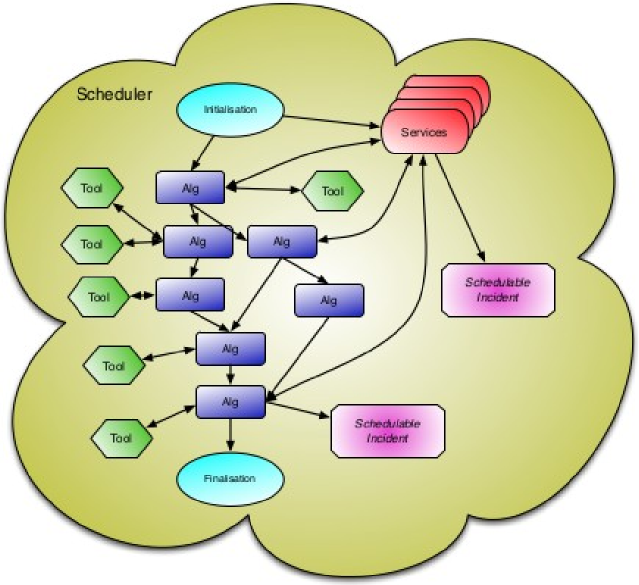
*affordable in capital costs and in thermal costs

Framework Elements



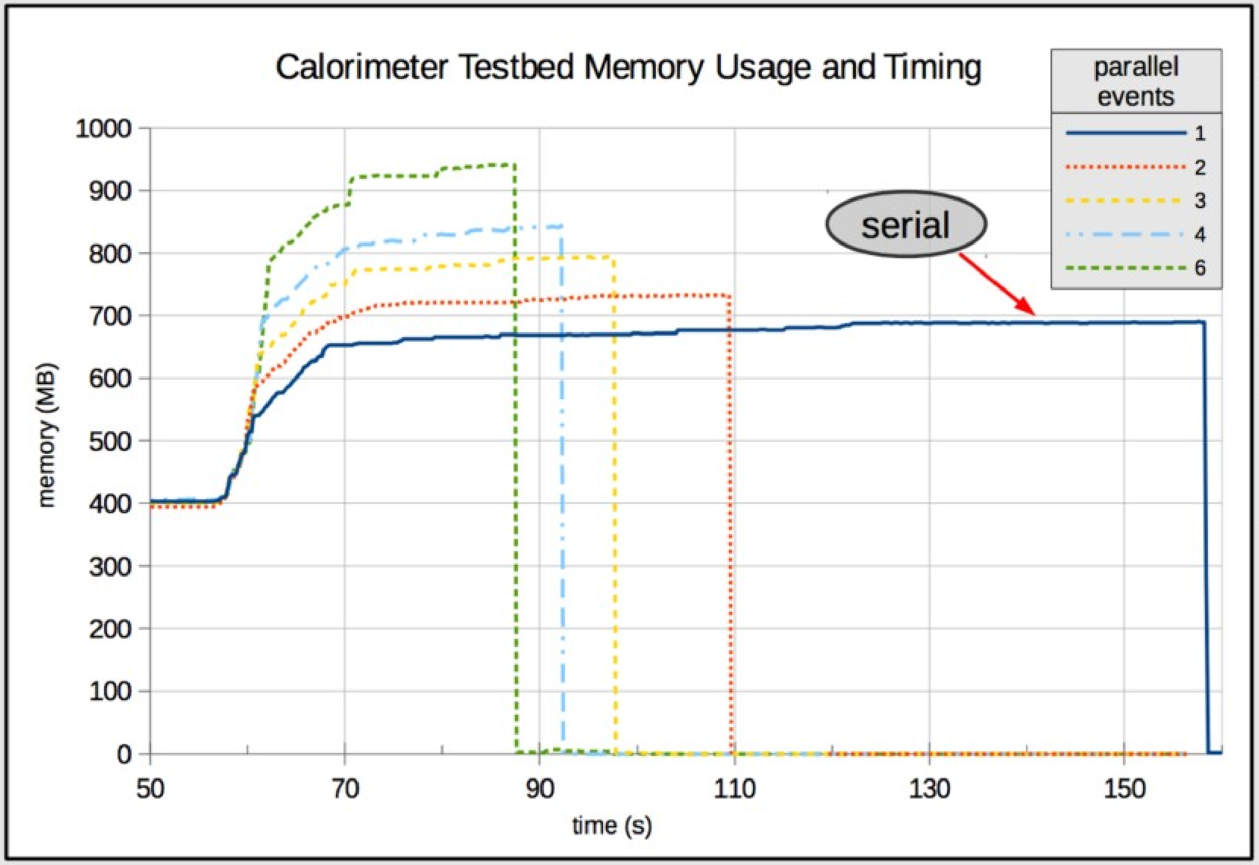
- Many ideas and concepts stay the same
- Mature model of event processing already
- Evolve towards concurrency
  - Because it's possible
  - And we need to keep a running system working, with continuity

Key Concept Changes



- Data dependencies are explicit and visible
  - Happen via the whiteboard
- Scheduler will parallelise algorithms and events when possible (subject to constraints)
- Scheduler handles non-event work
  - e.g., Incidents become ‘tasks’
    - (However, these are discouraged compared to data and control flows)
- Algorithms and tools are event specific
- Tools are always private
  - Use **only the whiteboard** for inter-algorithm communication
  - Use *sequences* for algorithms that create, modify, modify (+done) a data object
- Services are global - must be aware of context when called from algs and tools

Calorimeter Testbed Timing Results

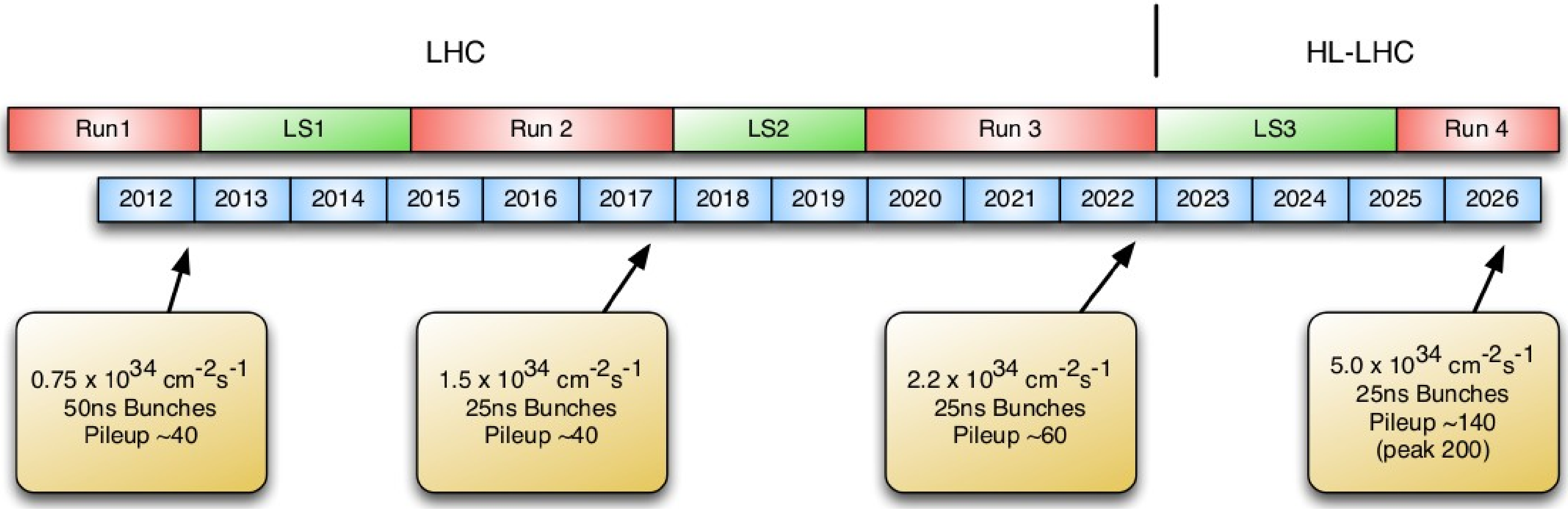


Calorimeter Testbed Memory Usage and Timing

parallel events	speedup wrt serial	speedup wrt n*Serial	memory ratio to serial	memory ratio to n*Serial
1	1.00	1.00	1.00	1.00
2	2.07	1.04	1.06	0.53
3	2.80	0.93	1.15	0.38
4	3.33	0.83	1.22	0.31
6	4.01	0.67	1.36	0.23

- Concurrency limited by small number of Algorithms in configuration, some of which could not be run concurrently for thread safety issues
- Best performance is with **6** concurrent events
  - **401%** event throughput, (ignoring startup to 1st event), and **36%** increase in memory consumption *vs* one serial job
  - **67**% event throughput, and **23%** of memory utilization of 6 serial jobs running concurrently

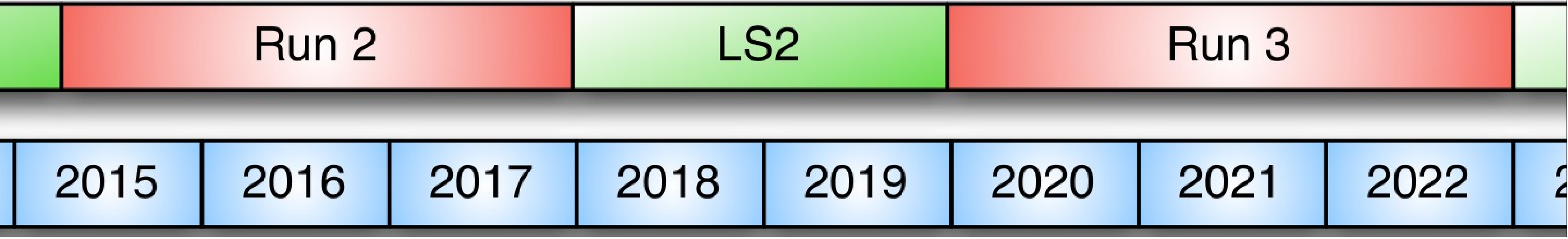
Timeline



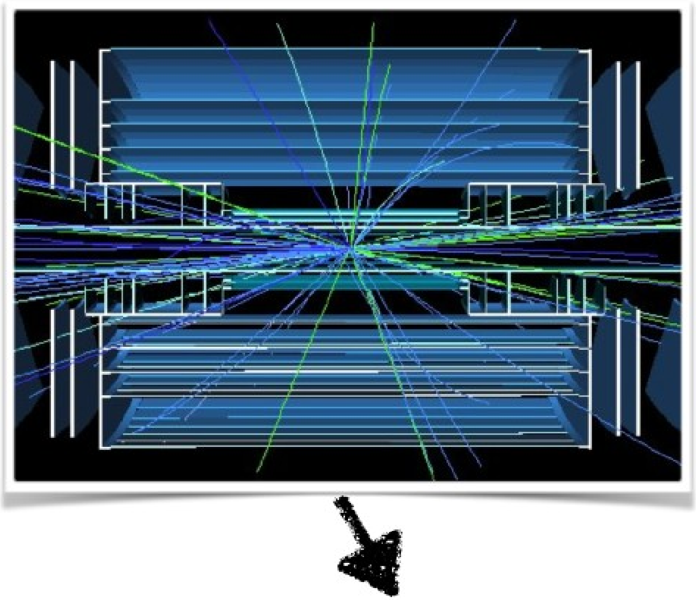
- Want to have a multi-threading framework in place for Run3
  - Allows experience with running multi-threaded before HL-LHC
- Thus most development should be done by the start of LS2
  - This is now only 2 1/2 years away
- At the end of Run2 we should have a functional multi-threaded prototype ready for testing

Some More Details

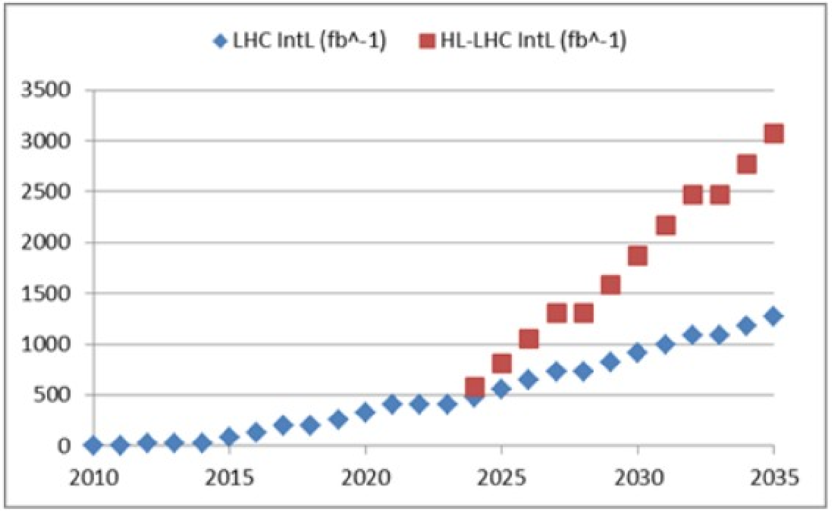
Dates	Framework	Algorithmic Code
2015	Baseline Functionality	Design Review; Real work with a aew algorithms, concentrate on high inherent parallelism; general clean-up
2016	Most functionality available (including views)	Wider set, including CPU expensive algorithms with internal parallelism; continue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



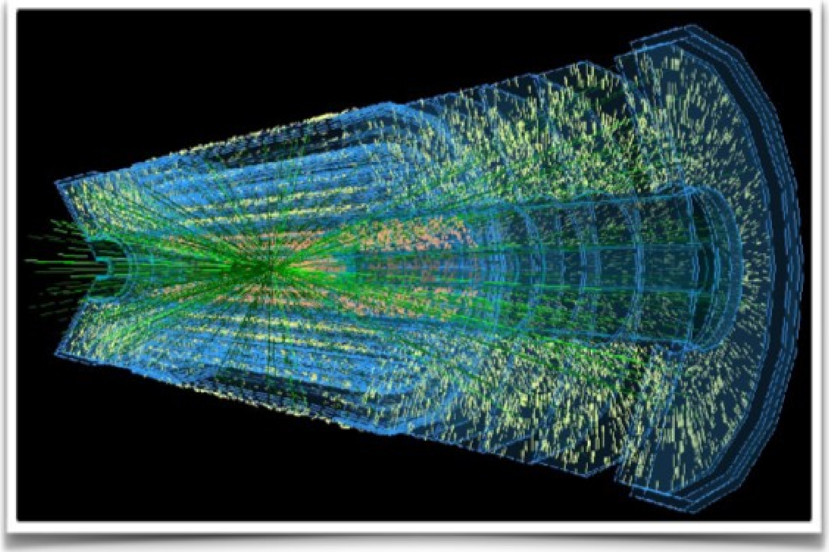
The Luminosity Challenge



X



Rende Steerenberg, CERN

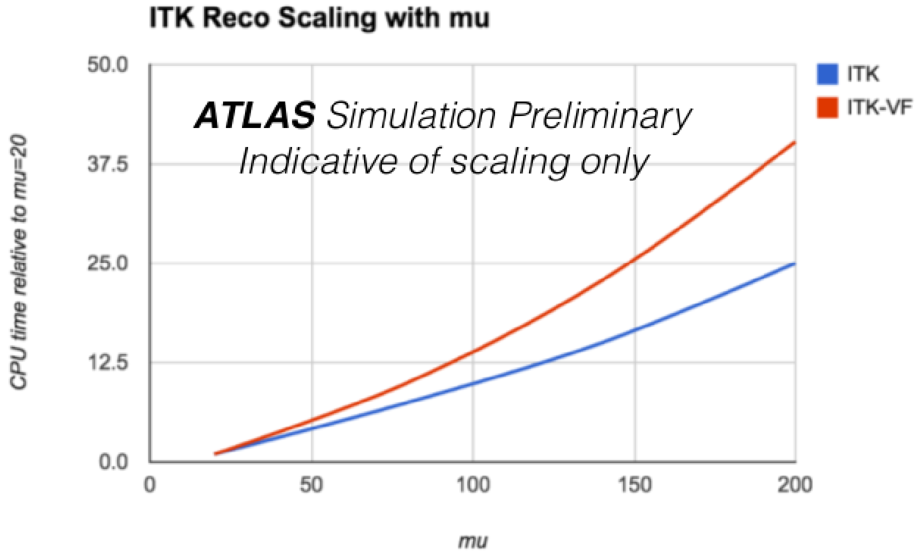


Event Complexity x Rate = Computing Challenge

- Reconstruction event complexity is naively μ! (factorial)
  - 140-200 expected by Run4
- Rate increases
  - 1kHz→5-10kHz for Run4

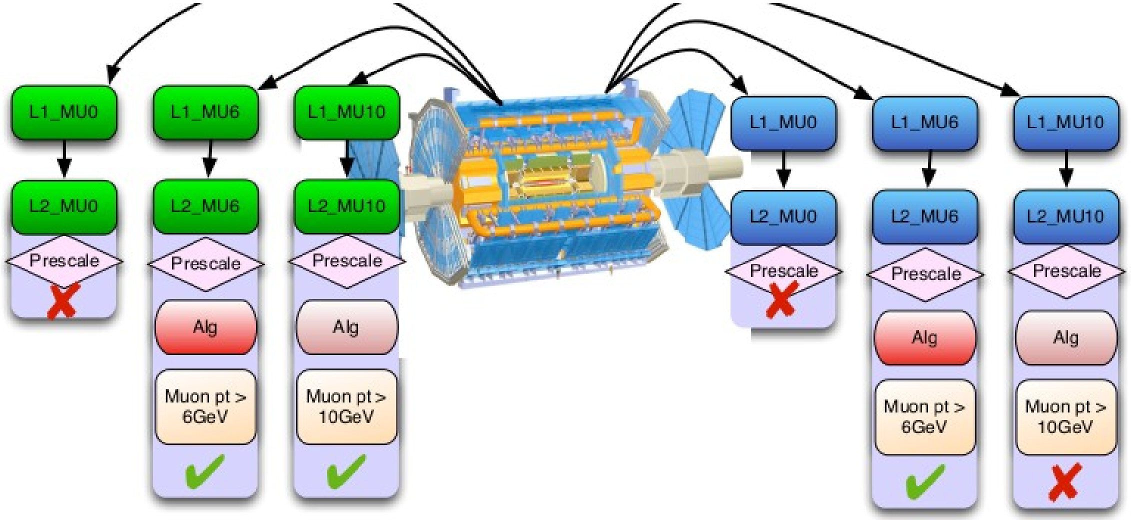
Parallelism within Algorithms

- At HL-LHC luminosities inner detector tracking becomes dominant in CPU time
- To avoid excessive numbers of events in flight algorithms need to be able to use parallelism internally
- Framework has to offer support for this mode
  - Need to find the solution to some tricky problems about event context when running on multiple threads
- General point: need a close relationship between the algorithm developers and the frameworks team



- High performance tracking with parallel algorithms will require particular attention to memory layout and use of cache hierarchy

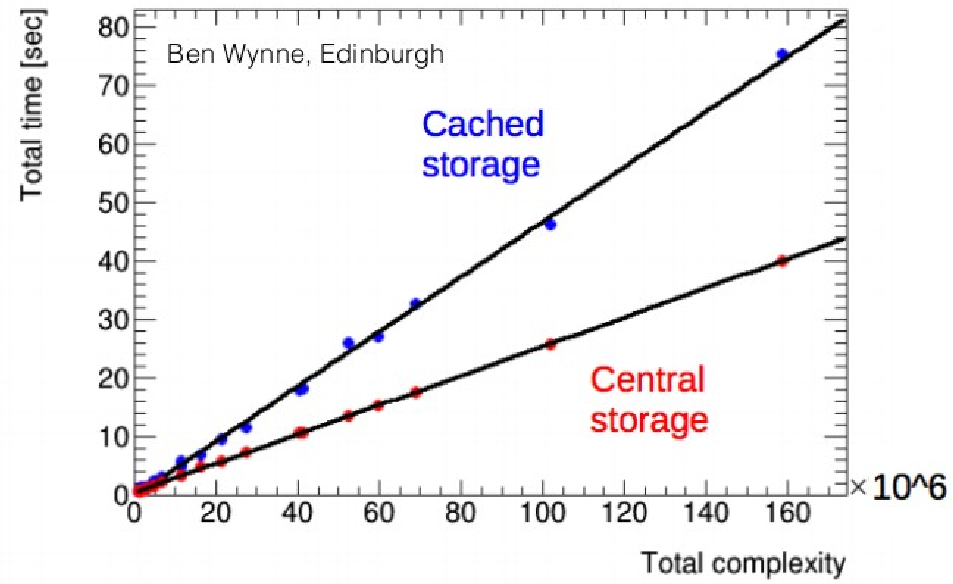
RoIs and Event Views



- Supporting running algorithms and tools on parts of an event is vital to support HLT workflow
- This is not something that offline have ever used
  - Dangling 'event views' in front of the software community didn't get much response, but might still be useful
  - Must, at minimum, be ignorable for offline ('do no evil')
- In any case it will definitely make the scheduler more complex
  - All (Alg + Data + Control + View) combinations need to run
  - Need to carefully examine logical implications (especially for overlaps)

Event Views Prototypes

- Early implementations of stand alone event views concentrated on evaluation of views as objects in a single StoreGate instance (“Central”) many separate mini StoreGates (“Cached”)
  - Seems that using a single central store for all views is optimal
- Now progressing to an in-Athena implementation
  - Views implemented to run over all ROIs and merge results
  - Interaction with scheduler is next step



DataHandles

A major component of the new framework is the concept of DataHandles

Eventually the only way to store/retrieve data objects, replacing all exisiting StoreGate access

This will require a change to ~all offline code

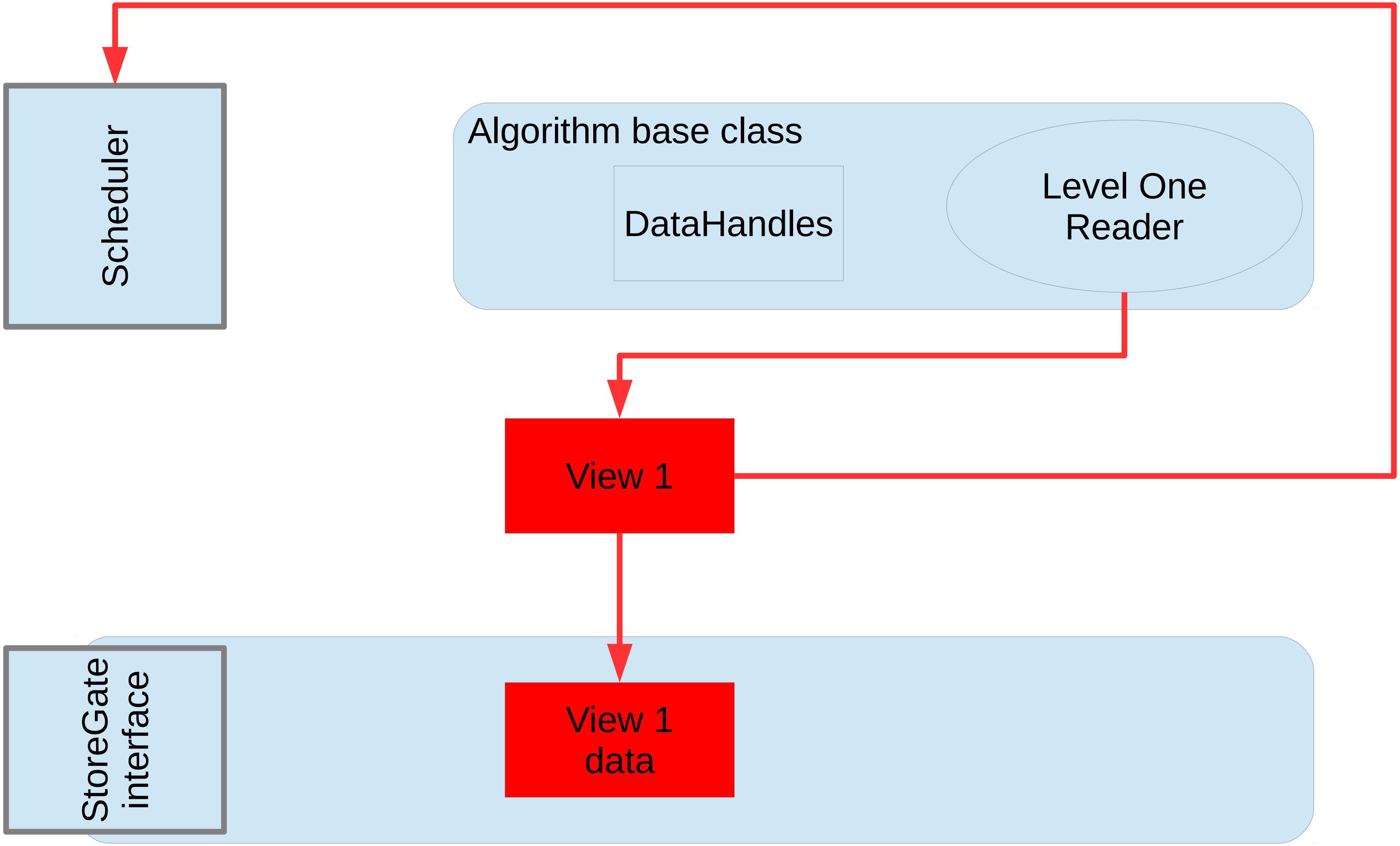
HLT migration work depends on how many offline algorithms can be used

Given that this migration is required anyway, we use it to introduce EventView behaviour

DataHandles should be able to use an EventView or StoreGate transparently, at least from the point of view of the algorithm code

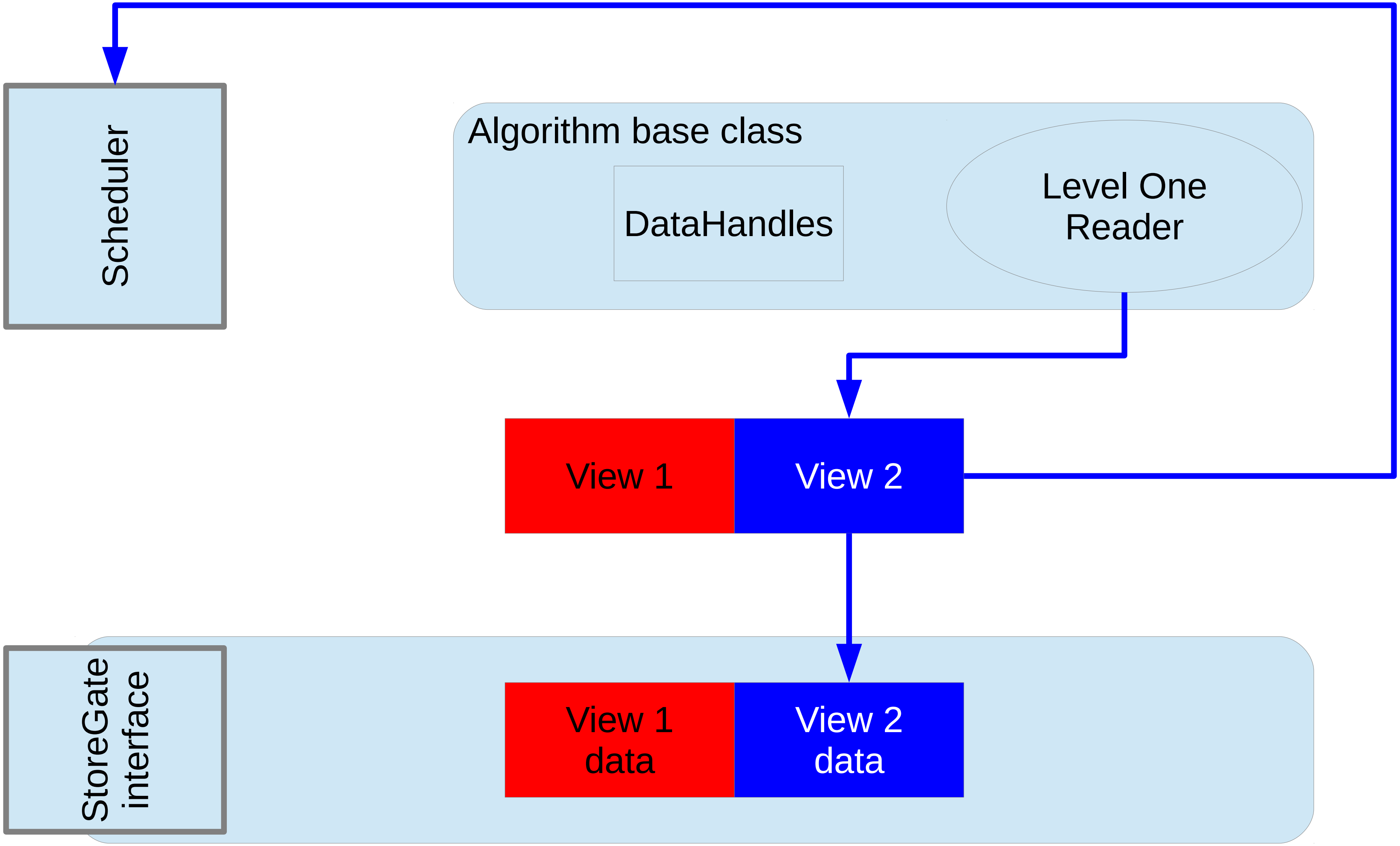
HLT workflow

The first algorithm launched by the HLT should be a “Level One Reader” that creates a new view for each RoI



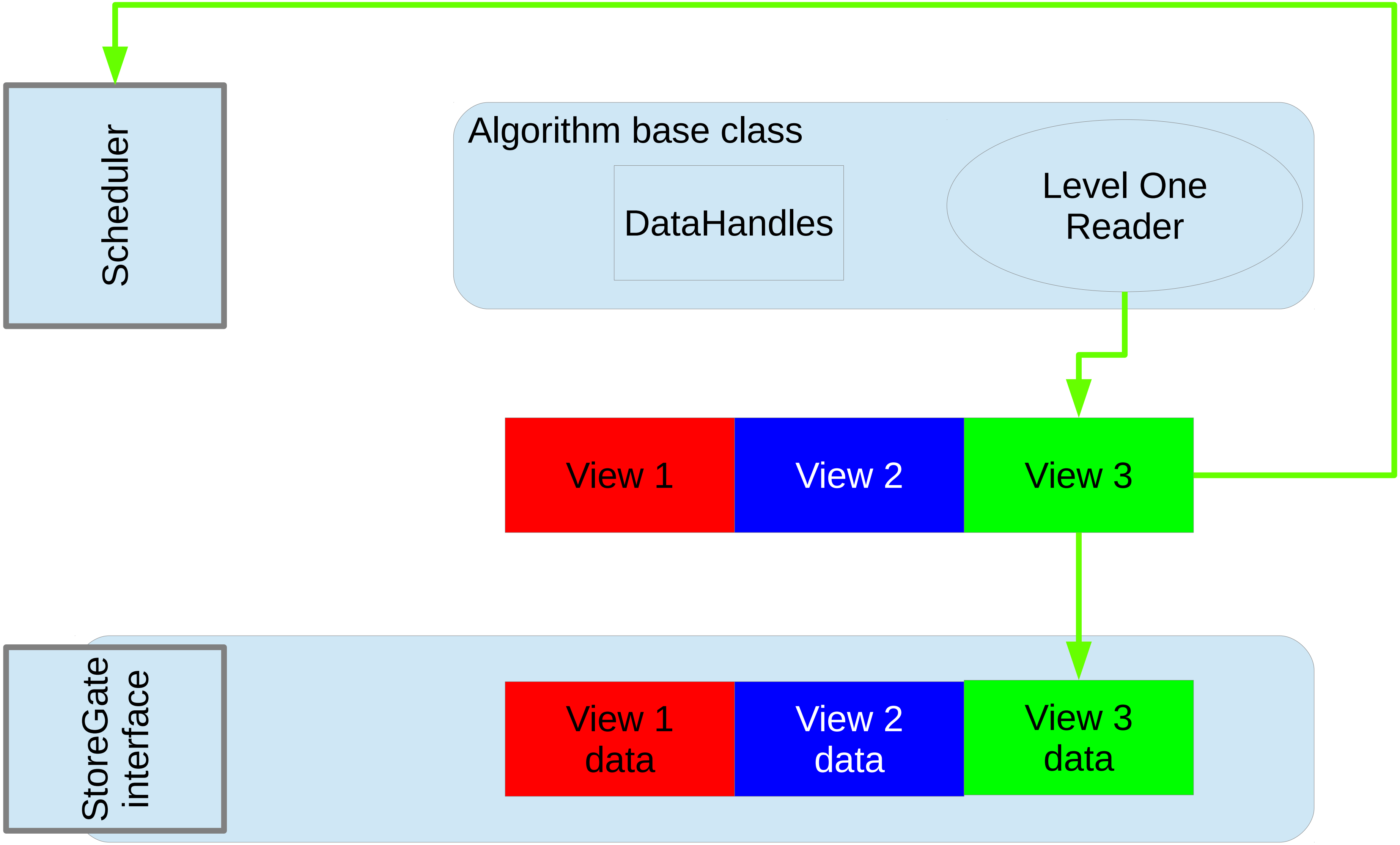
HLT workflow

The first algorithm launched by the HLT should be a “Level One Reader” that creates a new view for each RoI



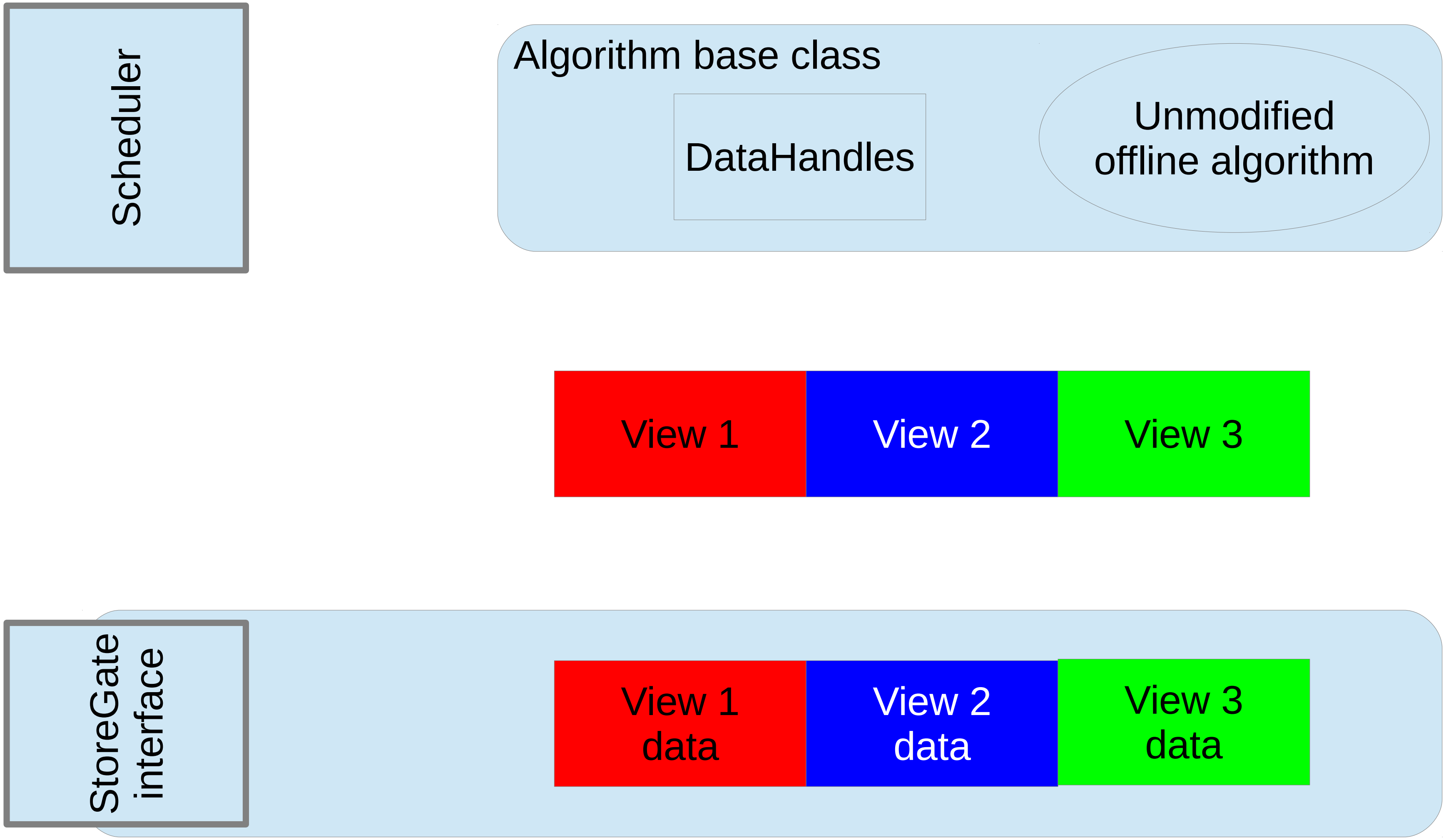
HLT workflow

The first algorithm launched by the HLT should be a “Level One Reader” that creates a new view for each RoI



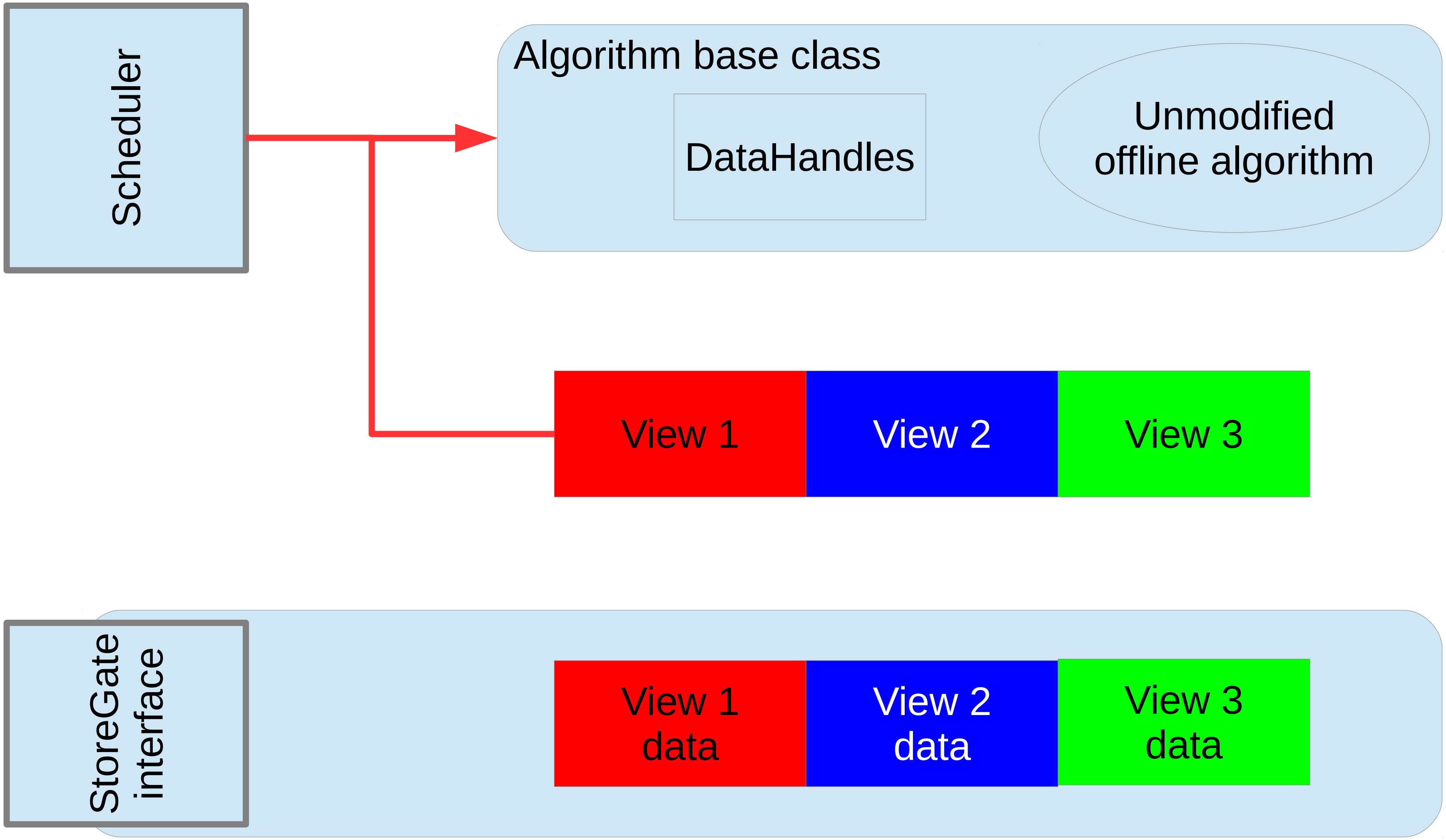
On algorithm execute

When the scheduler runs an algorithm, it passes the appropriate view
The data handles belonging to the algorithm are updated



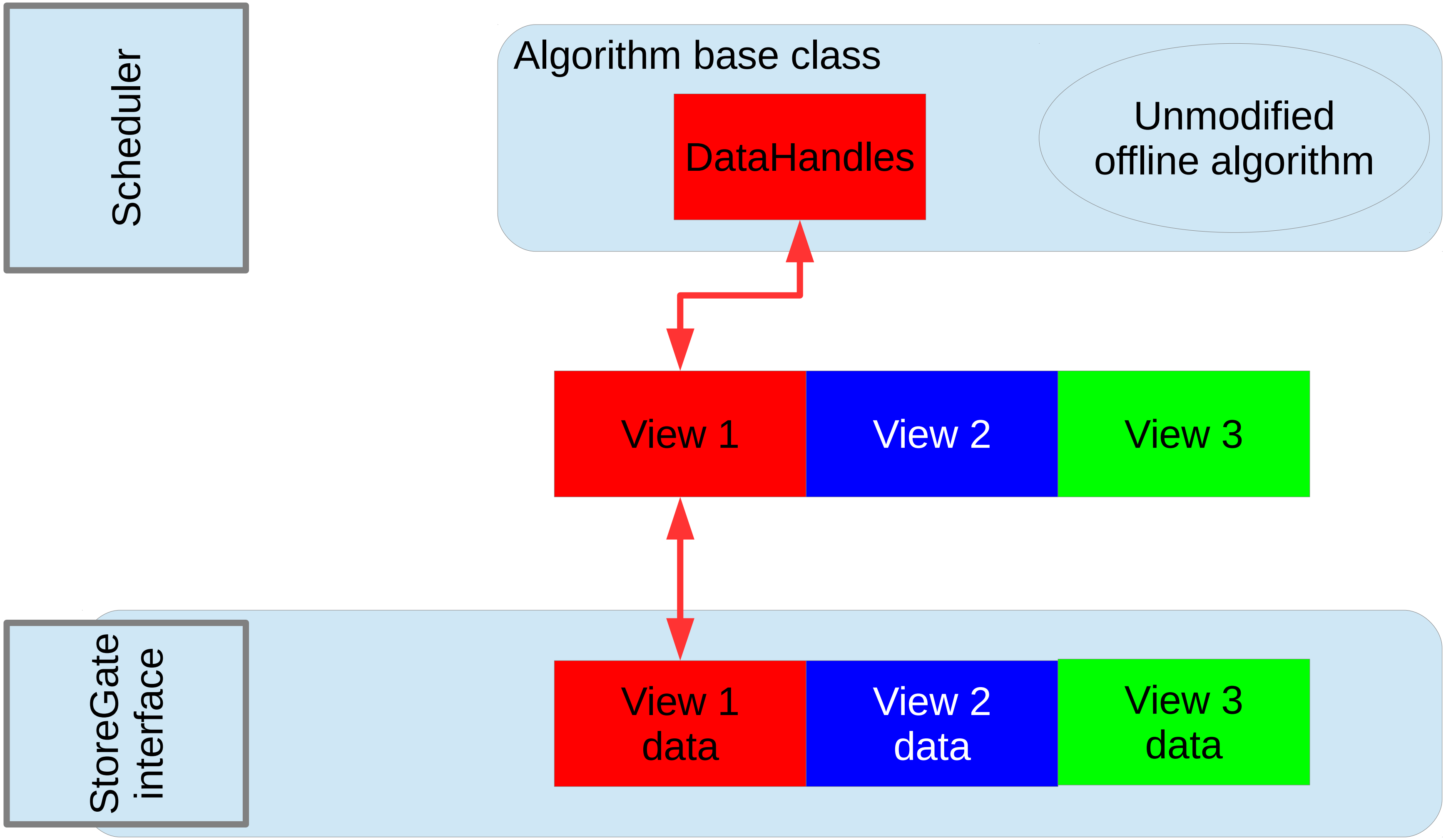
On algorithm execute

When the scheduler runs an algorithm, it passes the appropriate view
The data handles belonging to the algorithm are updated



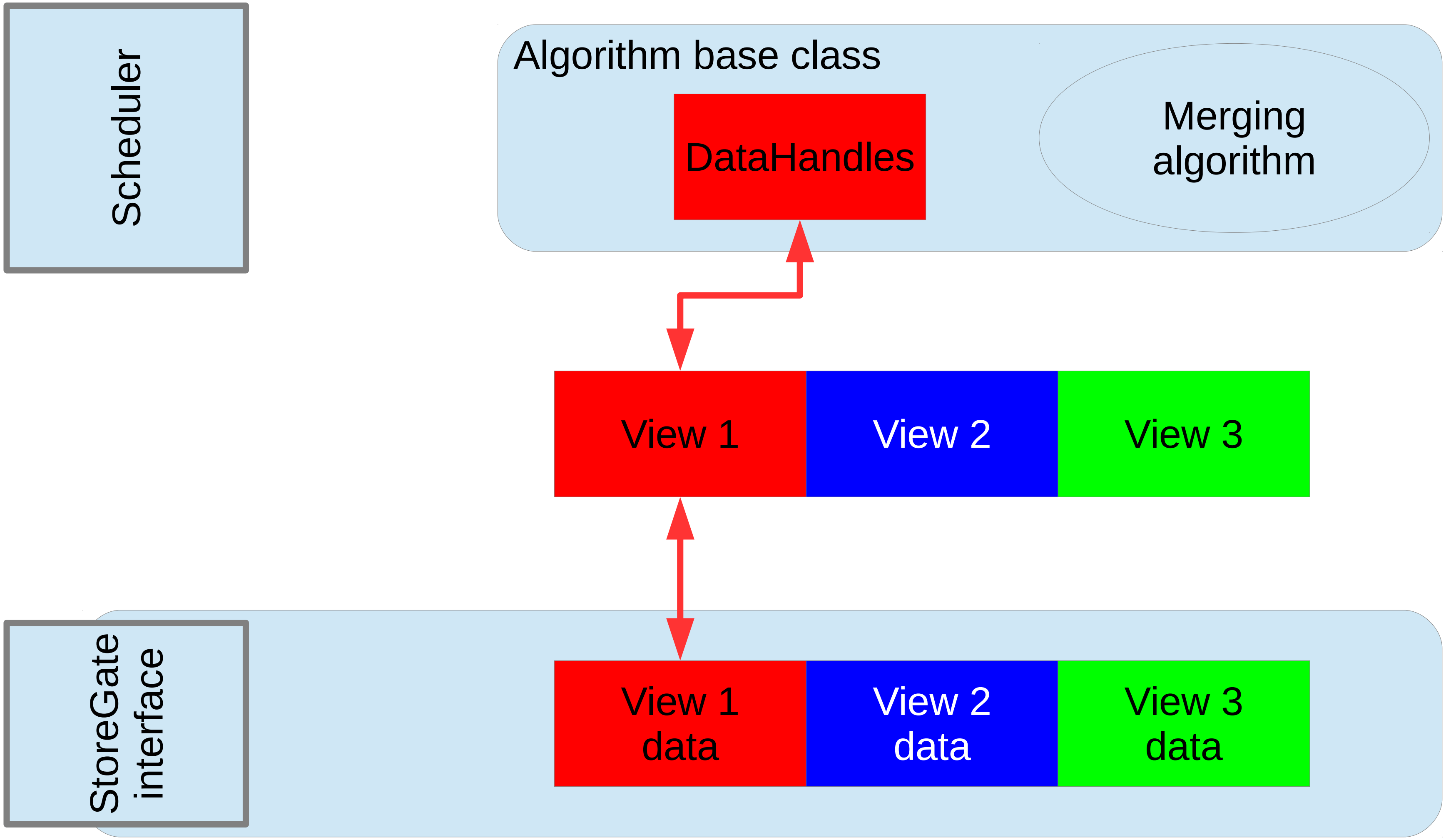
On algorithm execute

When the scheduler runs an algorithm, it passes the appropriate view
The data handles belonging to the algorithm are updated



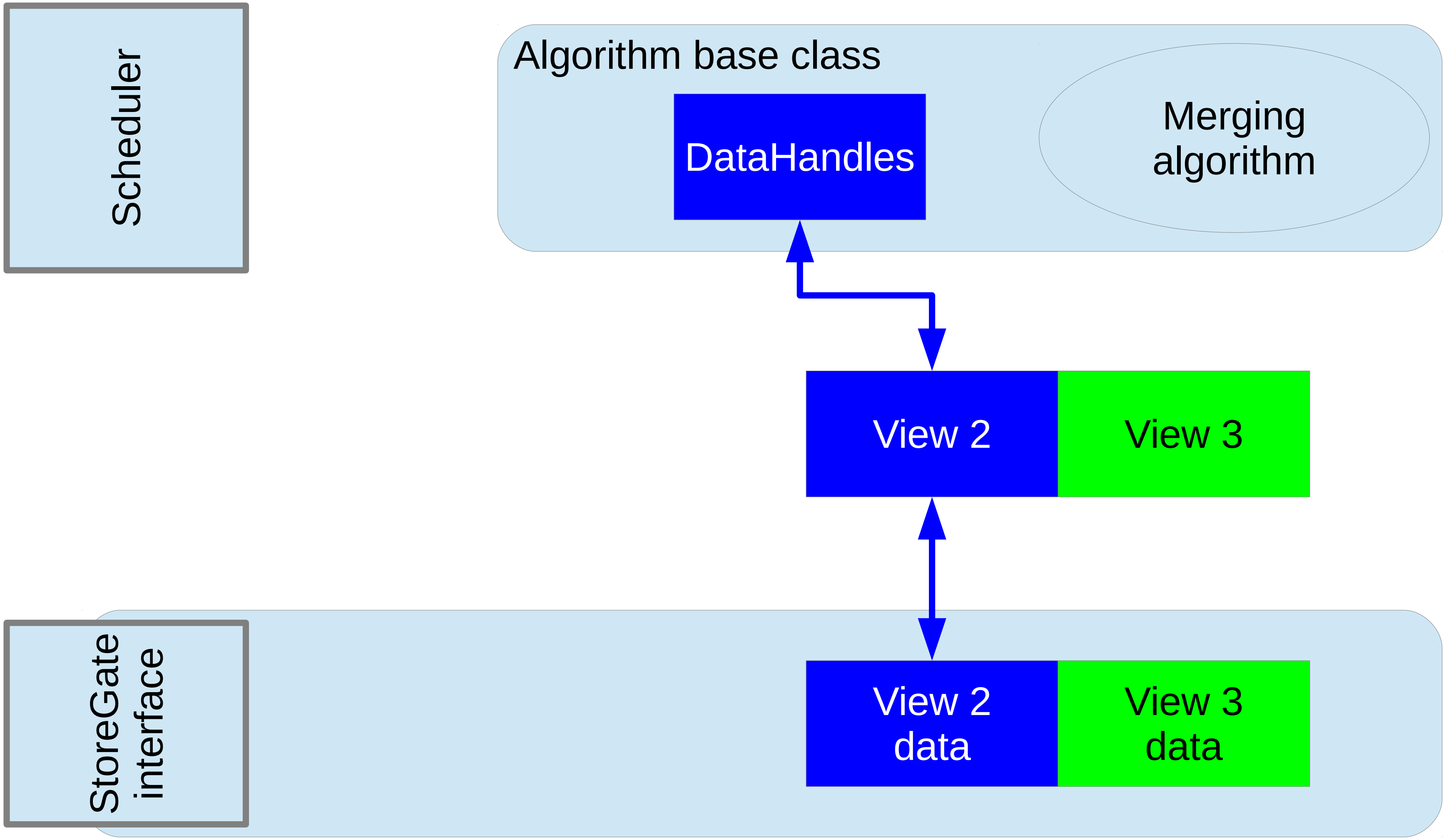
View persistification

When writing the event data to disk, view data should be merged
The views themselves should be represented by index objects describing which part of the merged data came from each view



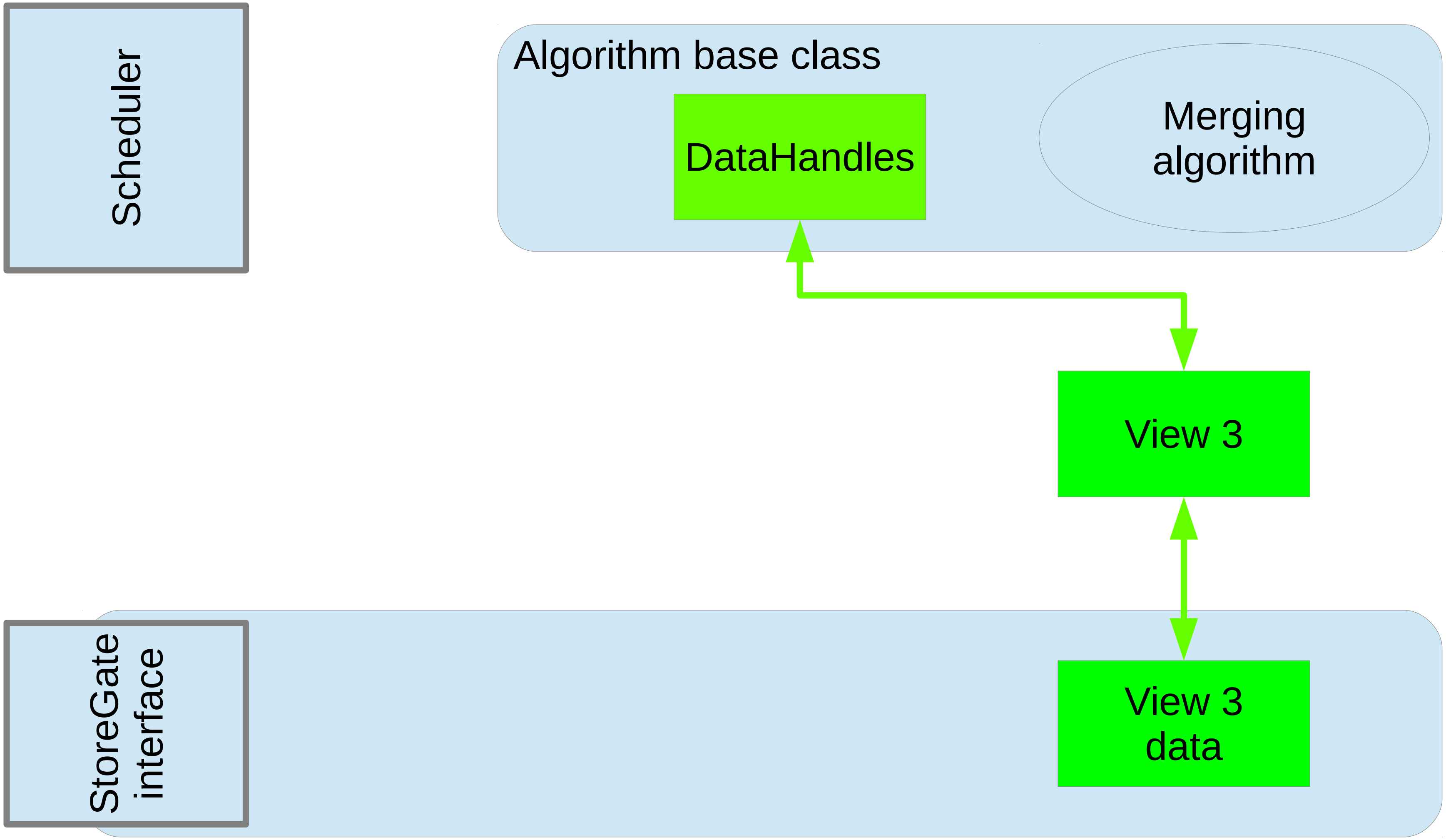
View persistification

When writing the event data to disk, view data should be merged
The views themselves should be represented by index objects describing which part of the merged data came from each view



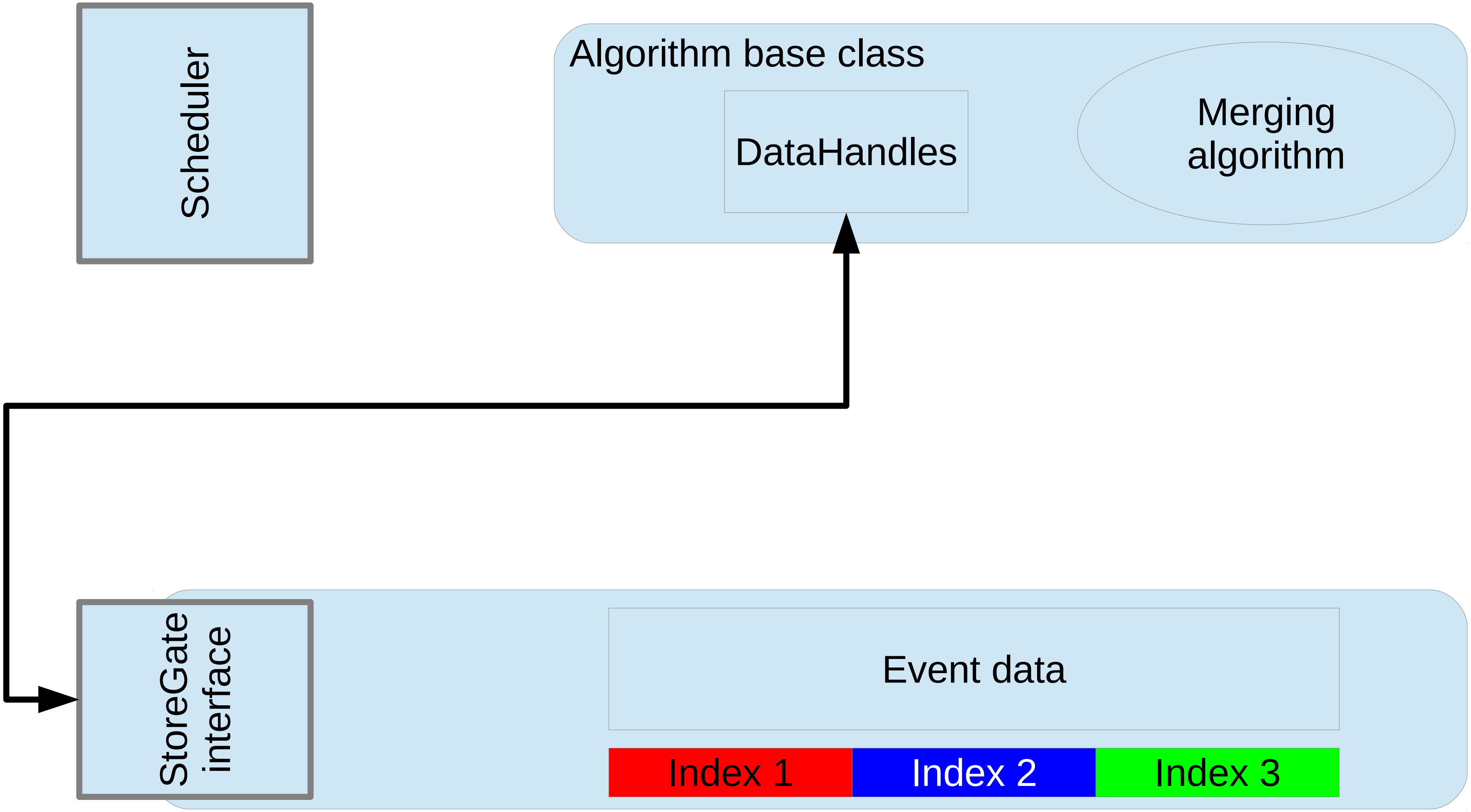
View persistification

When writing the event data to disk, view data should be merged
The views themselves should be represented by index objects describing which part of the merged data came from each view



View persistification

When writing the event data to disk, view data should be merged
The views themselves should be represented by index objects describing which part of the merged data came from each view



Design questions

How should the scheduler treat the creation of a view?

- (Main/whole event) thread waits for the (worker/event view) thread?
  - Deadlocks?
- Must be able to run single-threaded
- Can we avoid having separate HLT/offline schedulers?

How does the scheduler track data dependencies within a view?

- Need separate data flow for each view
- Some algorithms access data across views

How is the merge step scheduled?

- All views must be allowed to complete processing before merge is attempted
- Early reject means that not all view algorithms will definitely run

Summary

Prototyping of the new framework is well underway

- Mature AthenaHive build
- Gaudi collaboration re-established
- New features (e.g. EventViews) in development

Migration of algorithms is a huge task

- We don't actually know how huge
- Some simple 1-1 changes, but potentially some complete re-writes
- Importing current HLT code versus using offline algorithms for the HLT
- Comprehensive code review beginning

Next workshop as part of Software Technical Meeting in Berkeley (November)