Evolution of the ATLAS Software Framework towards Concurrency

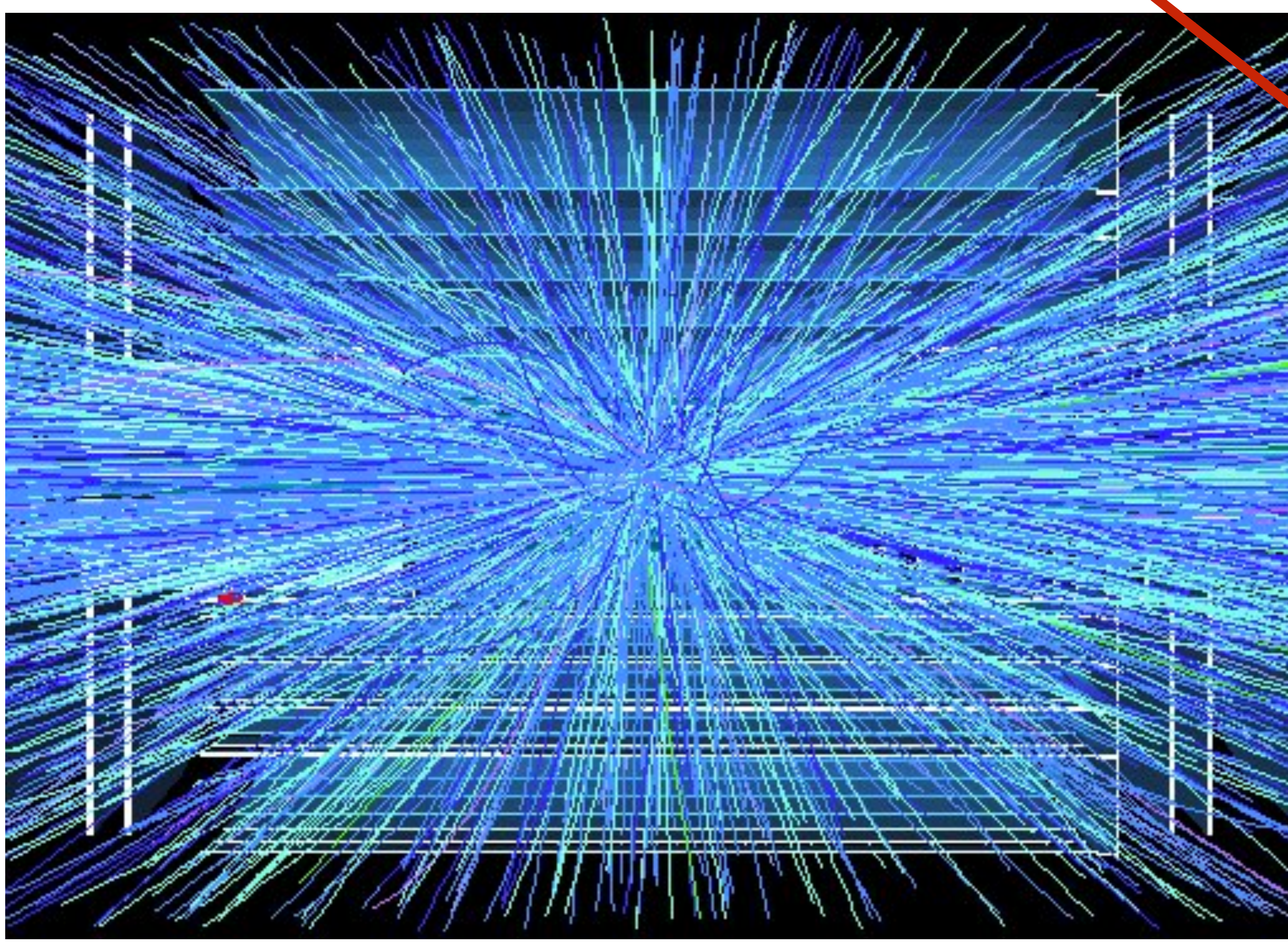
Roger Jones, Graeme Stewart, Charles Leggett, Ben Wynne
for the ATLAS collaboration

See also:
Gaudi Components for Concurrency: Concurrency for Existing and Future Experiments: Daniel Funke

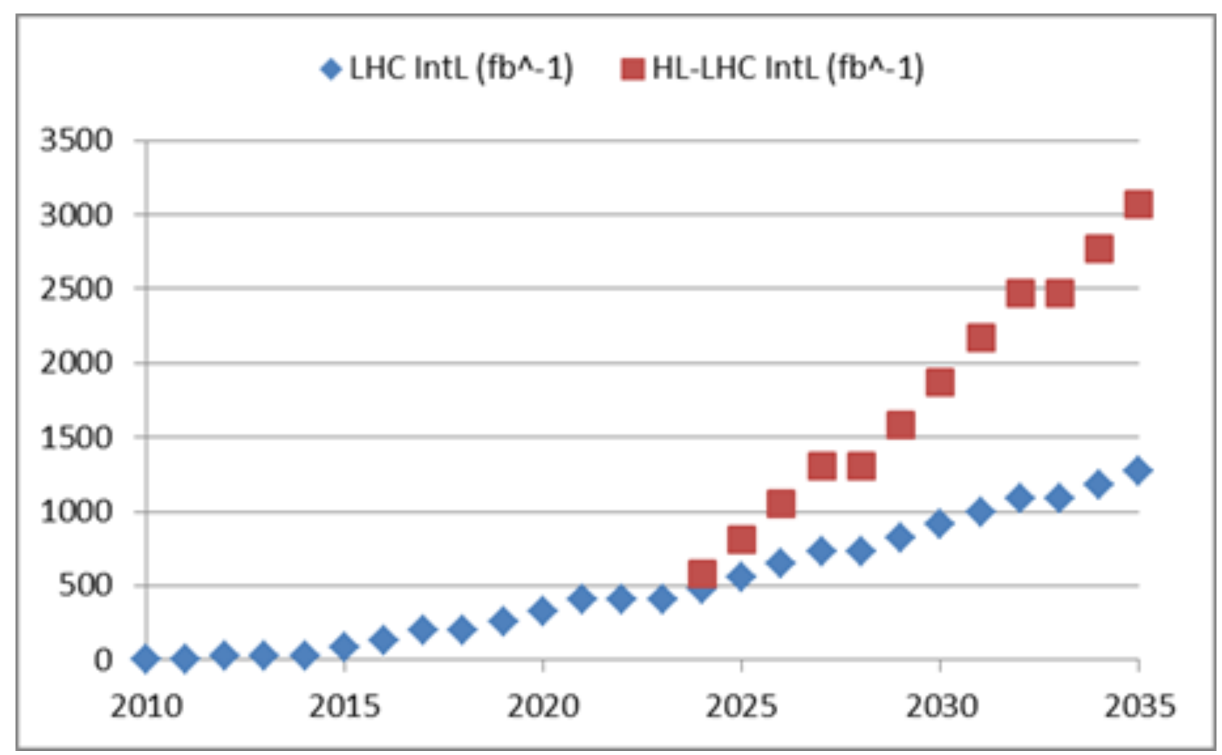
2014-09-02

High Luminosity LHC

Event Complexity
x Rate

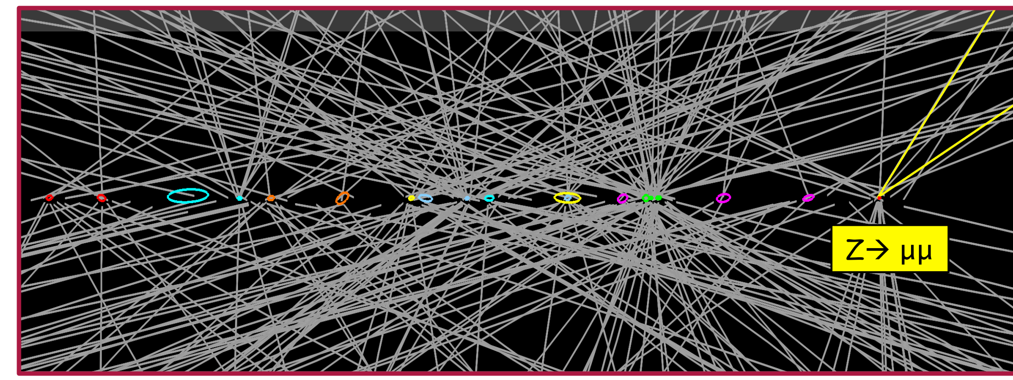


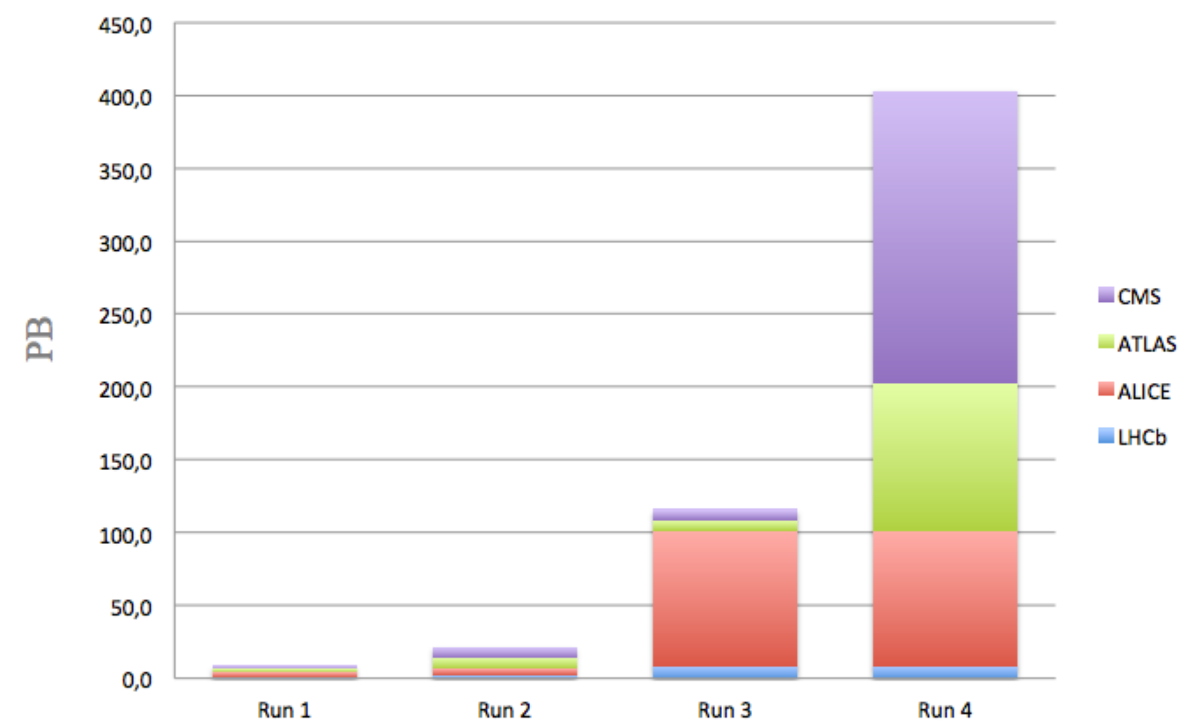
- Very high pile up
- Very high trigger acceptance rates
- Very challenging computing



HL-LHC In Numbers

- Pileup likely to be about 150 instead of <m> = 25 in Run 1
  - Exceedingly difficult conditions for tracking
- Readout rate will be x10 higher than Run 1
  - So data rate will be much higher
- Storage, archive, processing loads go up

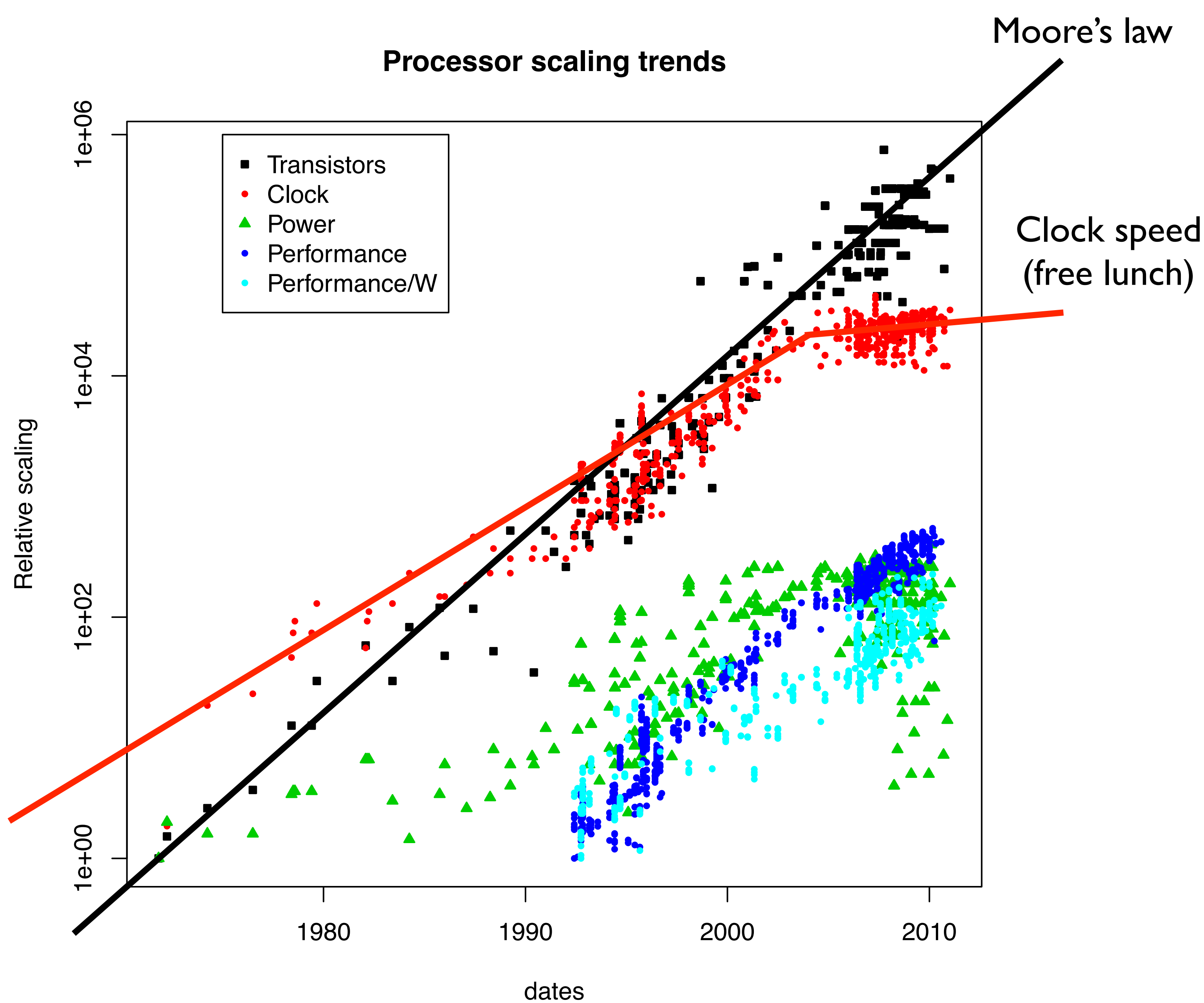




Total RAW data recorded

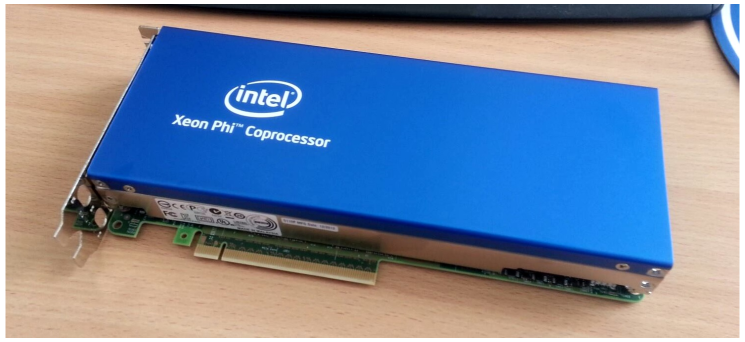
Processor Landscape

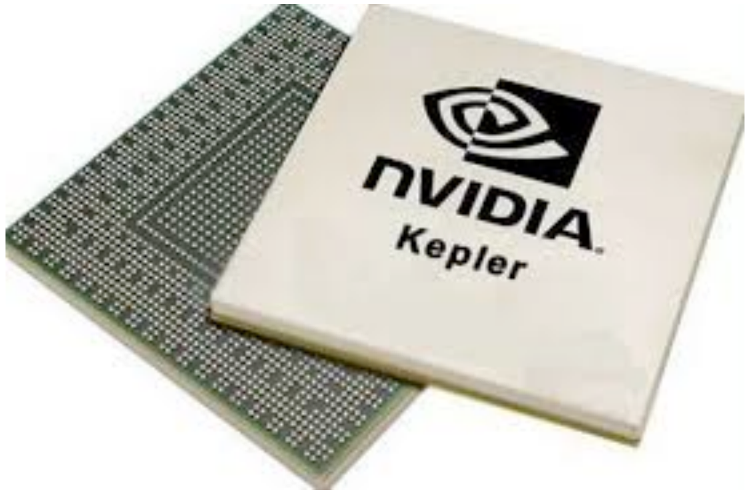
- Moore's law, still alive and well: 2 years → 2 x transistors
- There is now a lot of transistors looking for something do do:
  - Vector registers
  - Out of order execution
  - Multiple Cores
  - Hyperthreading
- All of these techniques increase the theoretical performance of a processor
- But hard to achieve this performance (or close to it) with HEP applications



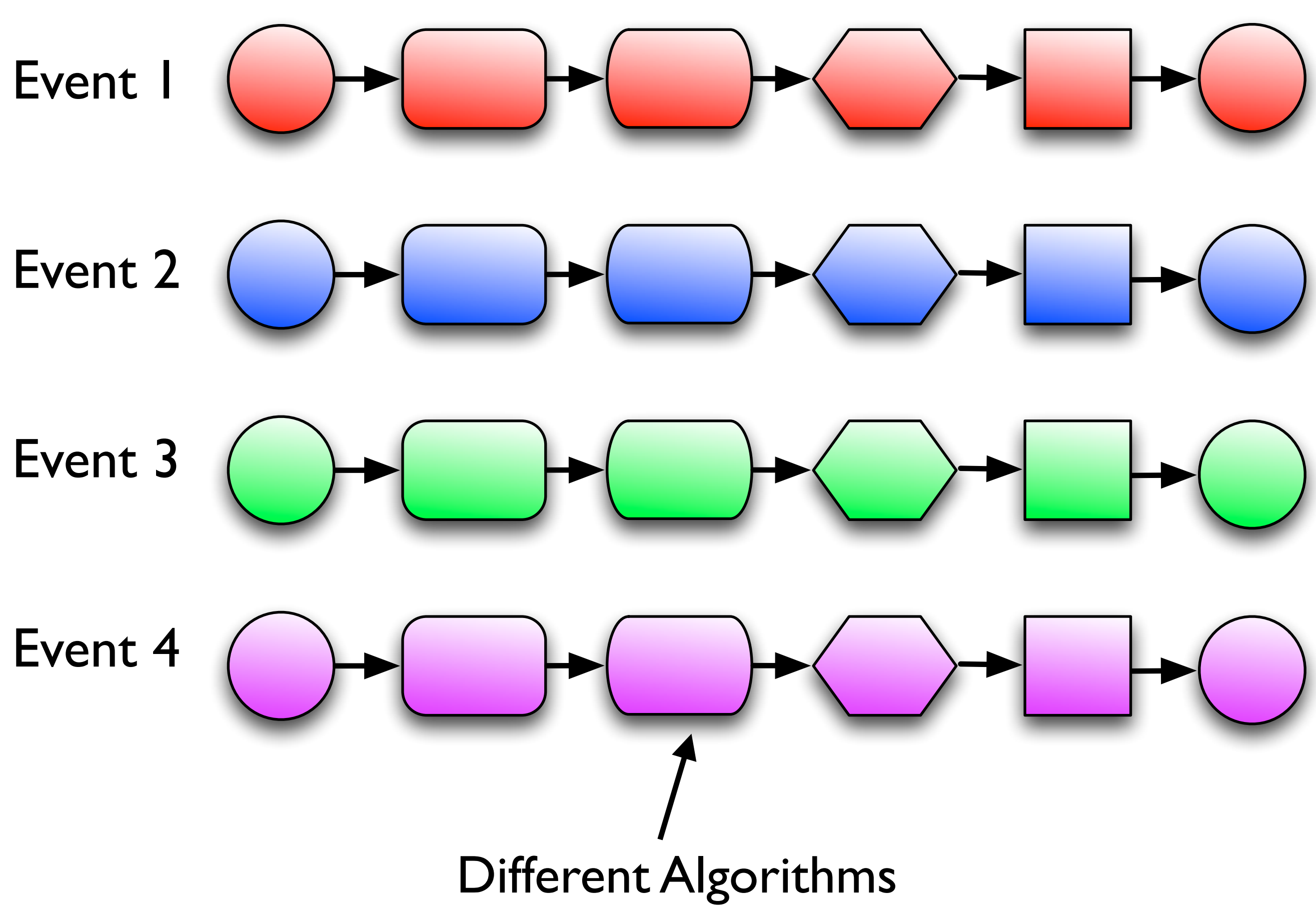
Processor Heterogeneity

- Seeing an increase in the range of platforms that will be available to do HEP computing on
  - Xeon Phi with very high x86 core counts (>60)
  - ‘Weak’ 64bit ARM multi-core processors and Intel Atoms and Avatons
  - GPGPU architectures with huge FLOPS
- Flat or falling computing budgets make it imperative that we are prepared for anything



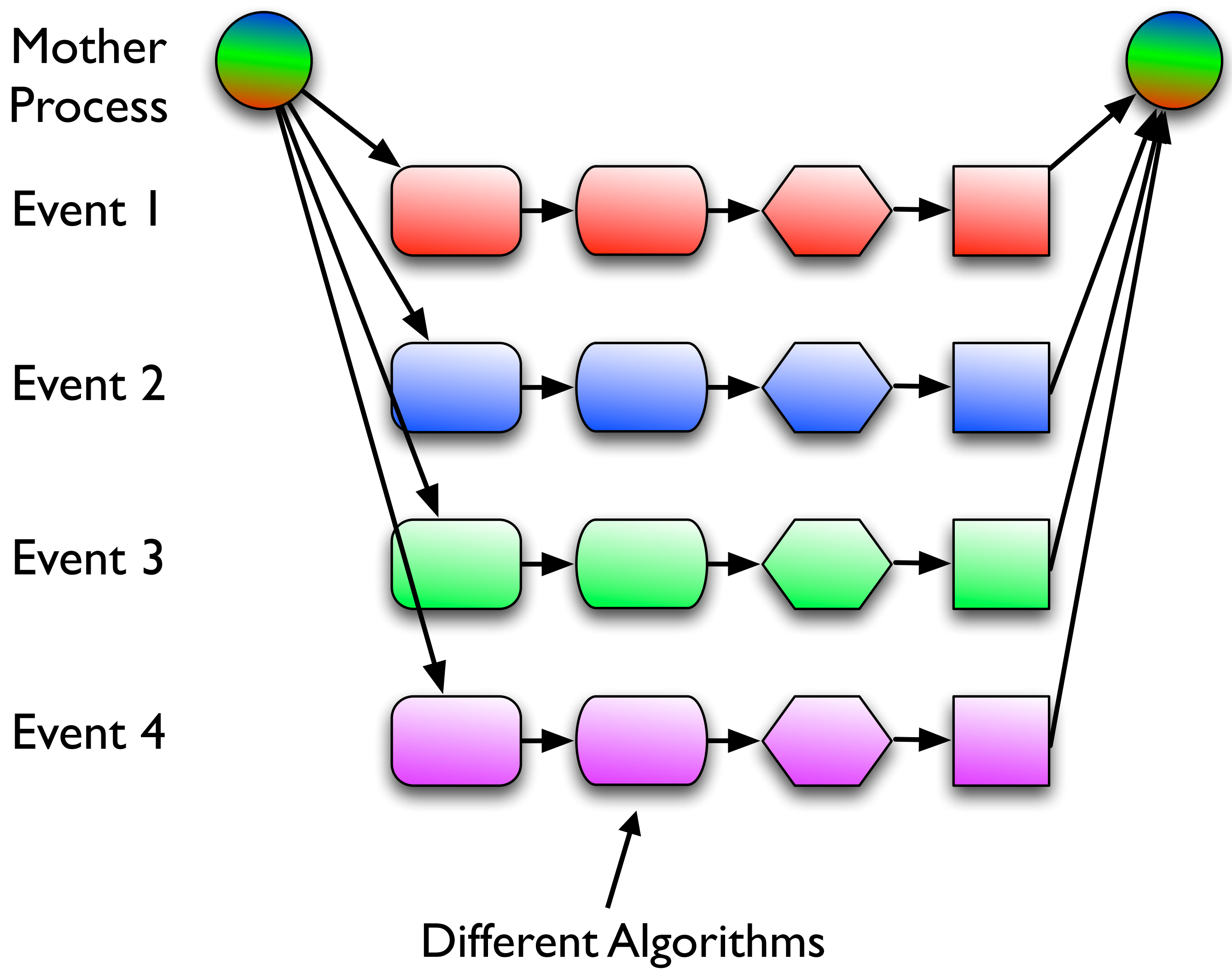


ATLAS Offline Processing in Run I



- Trivially parallel event processing
- Independent processes are efficient, but memory hungry
  - 2GB physical memory per core

Multi-Processing Athena

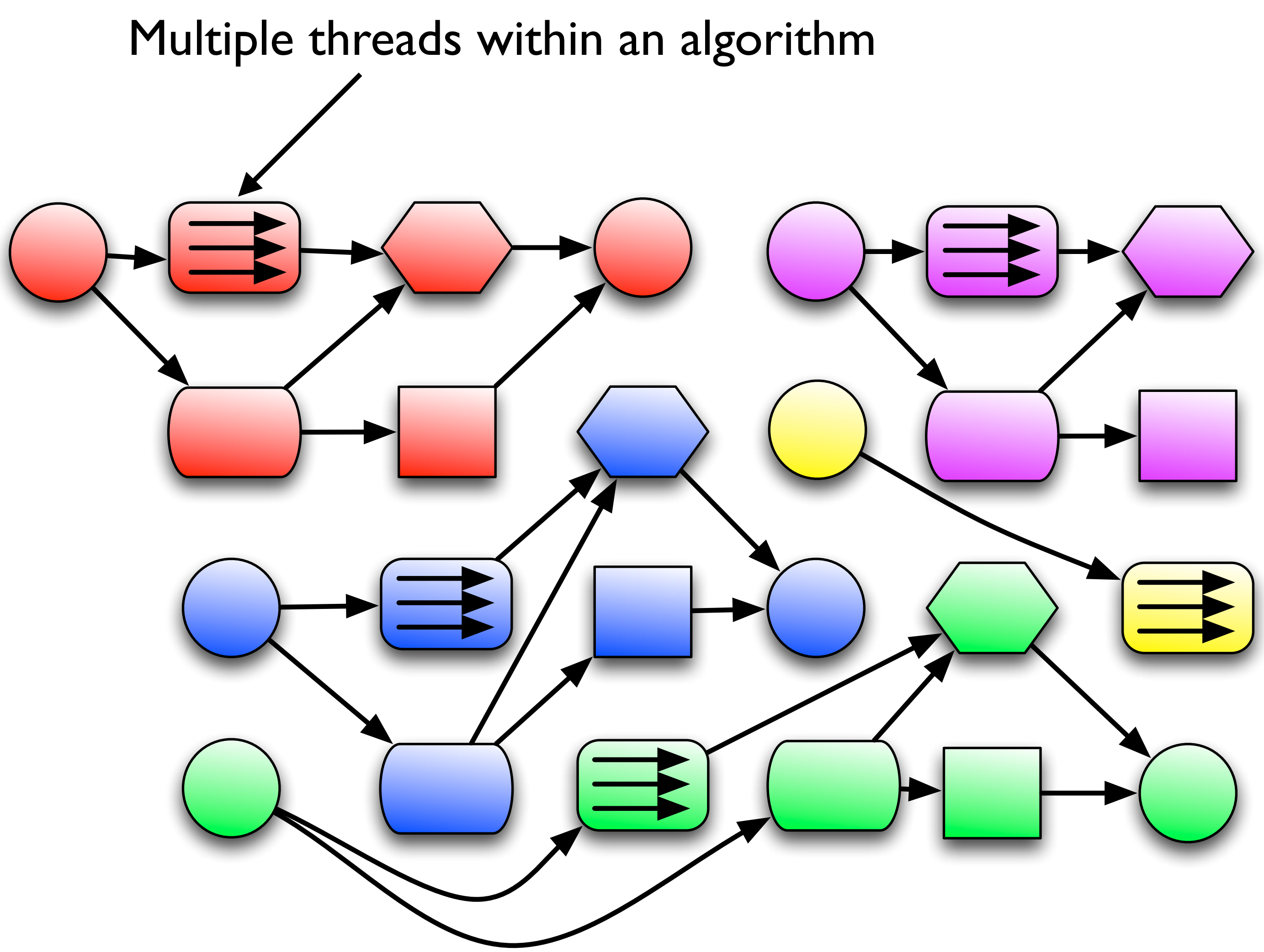


- Simple parallelisation of the ATLAS framework, Athena, by forking after initialisation
- Saves considerable memory using Linux kernel's 'copy on write' feature
- Will be a major part of ATLAS's Run 2 processing (online and offline)
- However, memory savings (~0.8GB/additional event) are unlikely to be enough for post-Xeon architectures

Memory Savings Beyond Multi-Processing

- To save memory beyond multi-processing it's necessary to use a multi-threading framework
    - Memory savings can be huge as all heap memory is shared
    - However, a more difficult programming model as threads can interfere with each other: data races and deadlocks
- Especially difficult to back-port threading into a framework and physics code base which has been run in a serial mode for (more than) a decade
    - We need to consider options for evolution carefully

A possible future…



- Take advantage of concurrency by exploiting event parallelism, algorithmic parallelism and in-algorithm parallelism
- Scenario should be suitable for many-core machines or light cores, where memory per core is limited

First Experiments

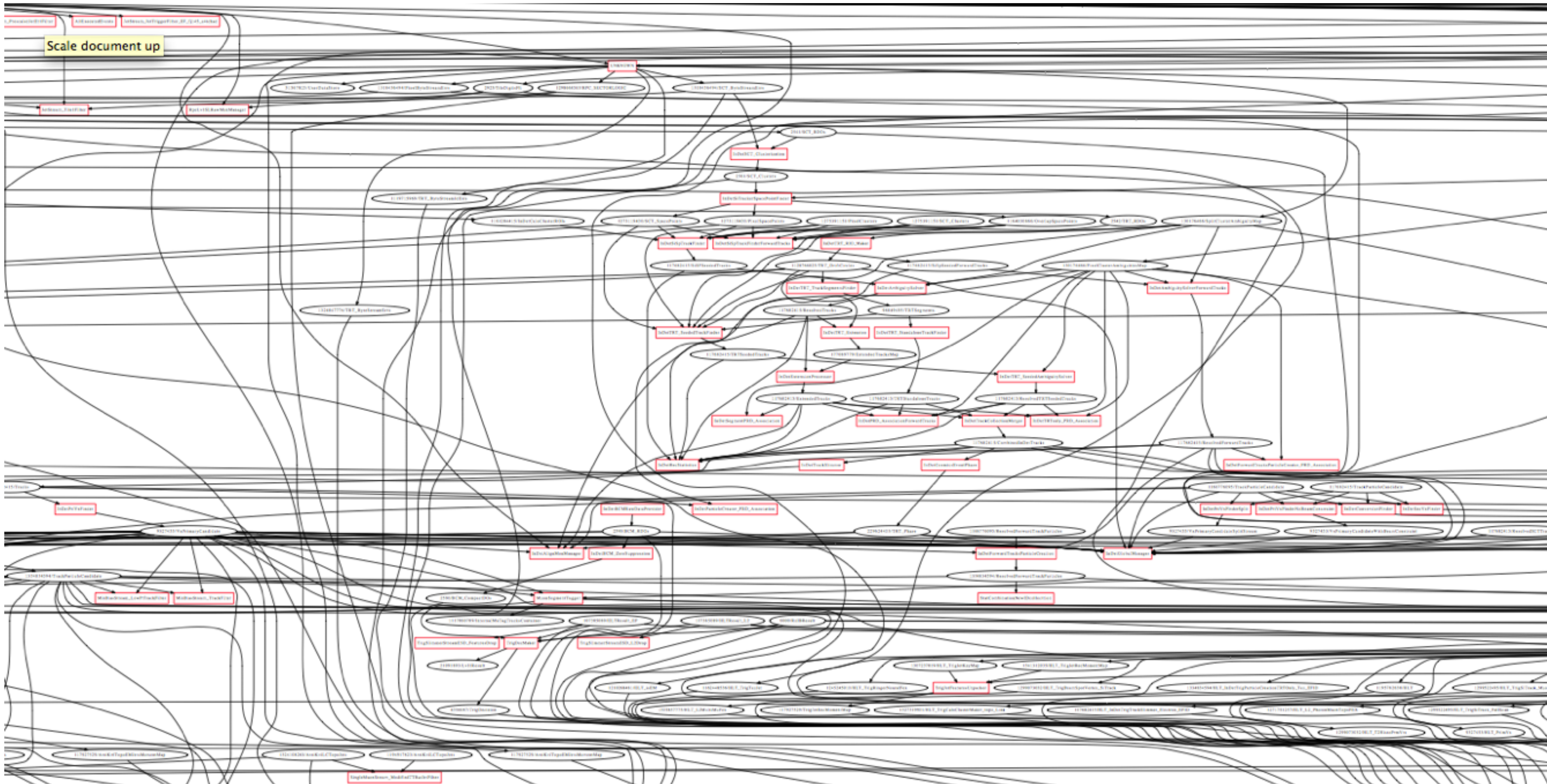
- Clearly the investment needed to convert ATLAS reconstruction is considerable
- Start with some tests with simulated algorithms (CPU crunchers) to see if multi-threading can work in principle
- For this we used the GaudiHive prototype framework developed by CERN SFT and LHCb
  - Recall that ATLAS already shares the Gaudi framework (Athena is a derivative) with LHCb

Data Flow

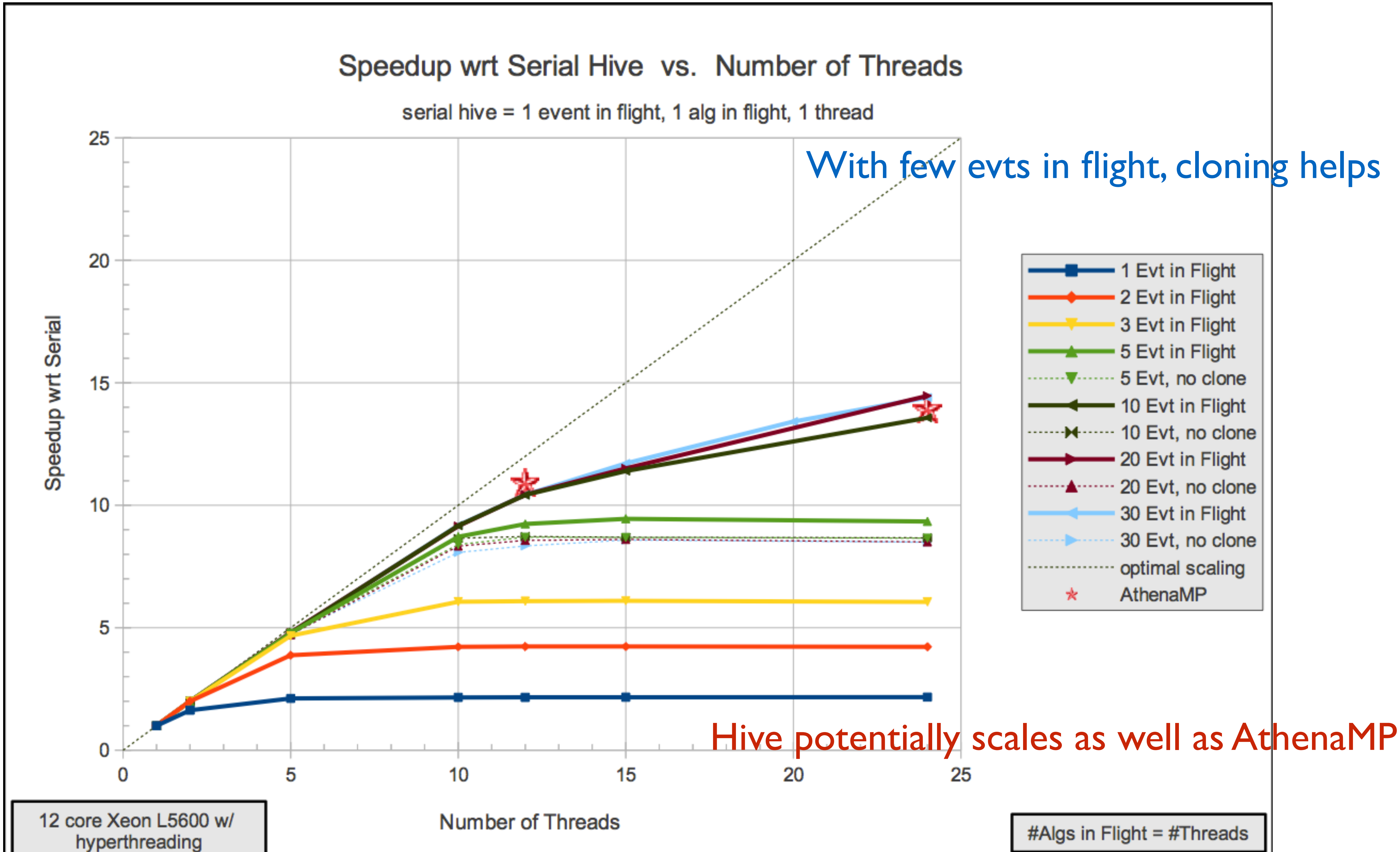
- ATLAS reconstruction runs several 100 algorithms with 100s of data objects produced in the event store

This is just a snippet!

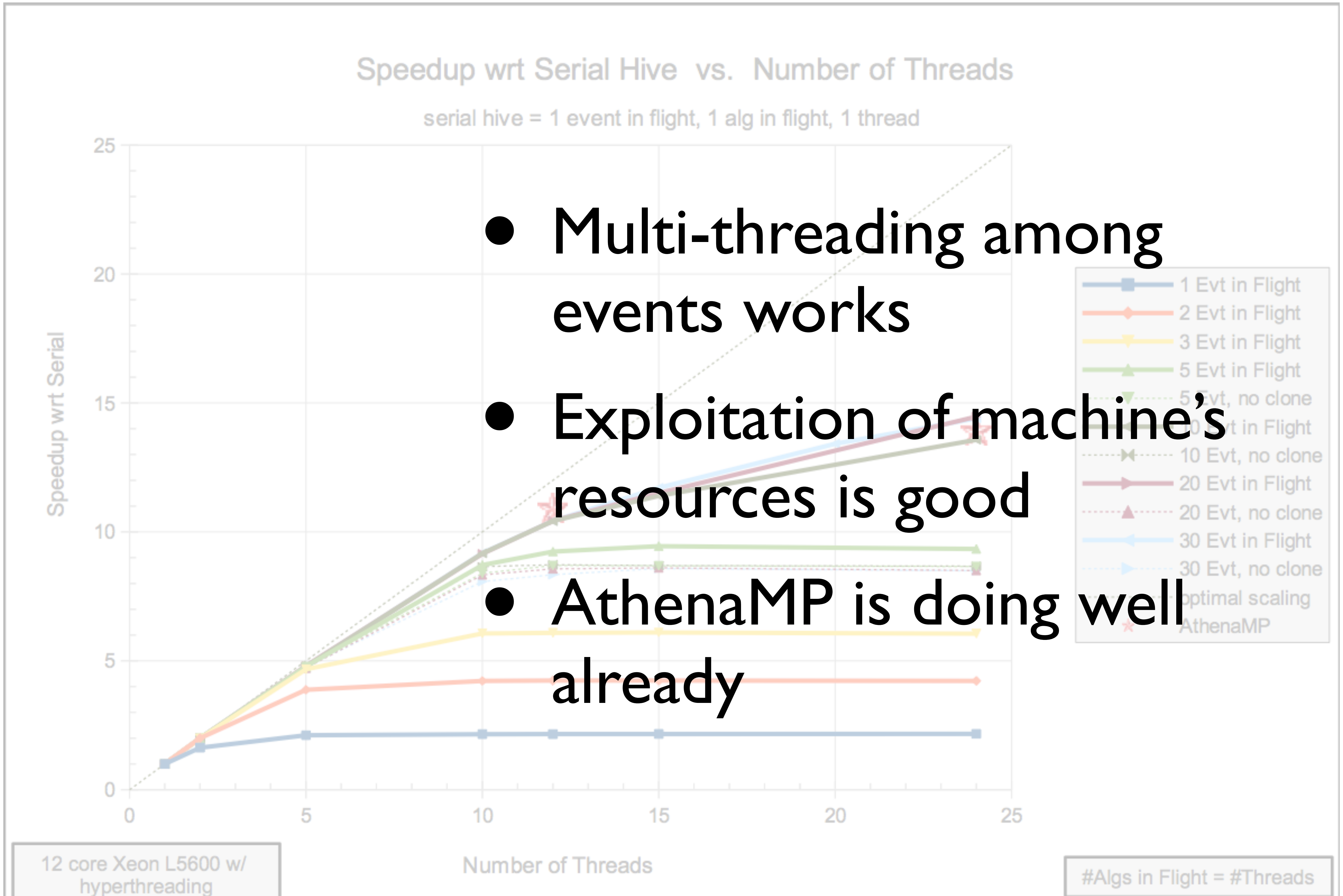
And there are hidden dependencies through public tools



CPU Cruncher Results



CPU Cruncher Results

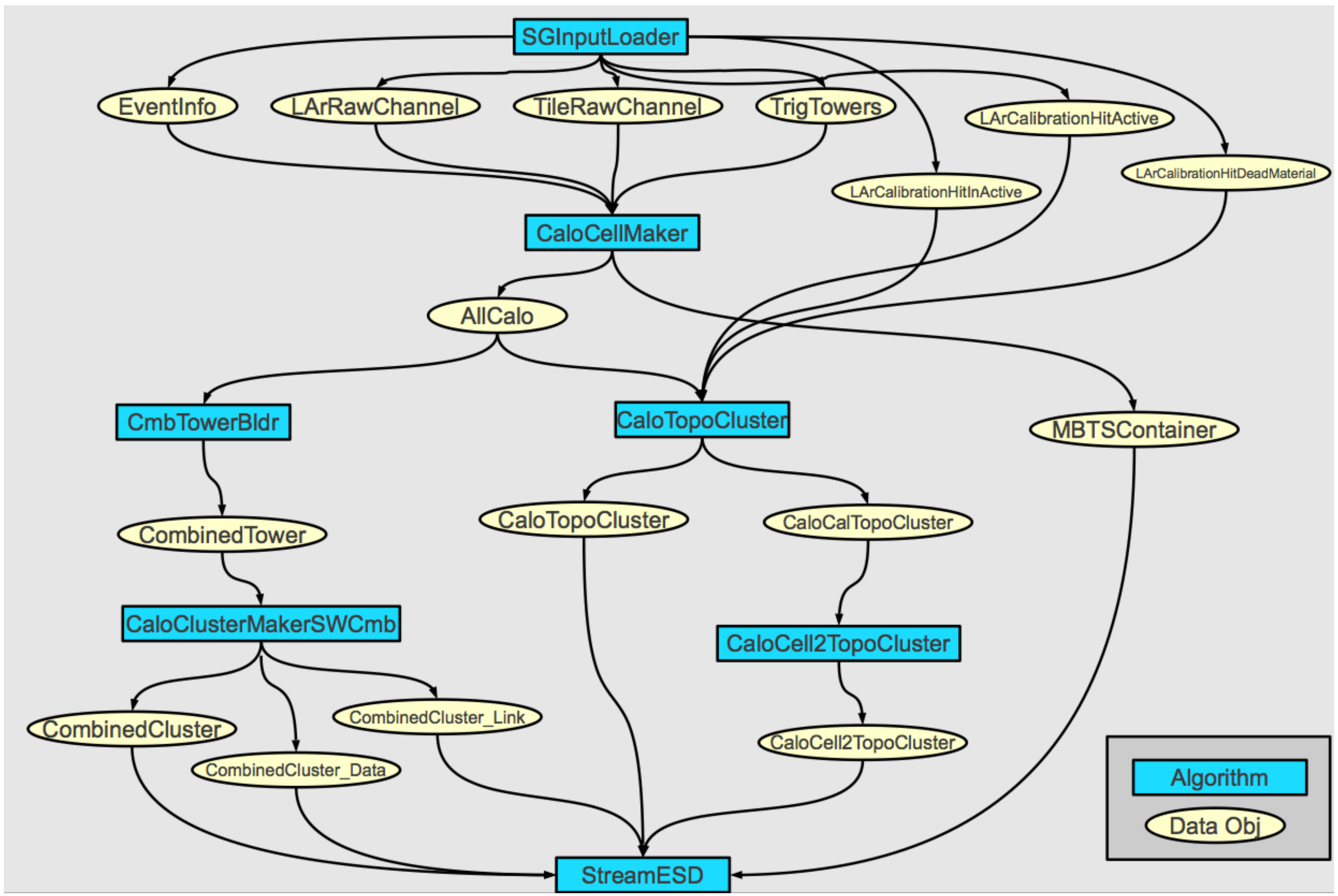


- Multi-threading among events works
- Exploitation of machine's resources is good
- AthenaMP is doing well already

Next Steps - Real Reconstruction

- After demonstrating potential improvements with a simulation we embarked on running real ATLAS reconstruction in GaudiHive
- Converting large parts of the reconstruction would be time consuming, so we picked calorimeter reconstruction as a suitable piece of work

Calorimeter Data Flow



- 7 algorithms
- 16 data objects
- Hopefully Tractable!

Code Modifications

- Even so, many pieces of code around the framework need to change, which are quite independent of the number of algorithms involved
  - Athena EventSelector
    - redesign for Hive
  - event data I/O (Converters) event data store (StoreGate)
    - make thread safe by locking (no attempt at parallelisation for now)
    - Multiple events blocked to a single convertor (could be improved)
  - AthAlgorithm and AthAlgTool base classes
    - enhance thread/event slot information for debugging
  - bypass Incidents (BeginEvent, EndEvent)
- Plus some necessary changes to the user code
  - Do not use incidents to trigger actions
  - Do not add data to the event store until it's ready to be used by other algorithms
    - Unfortunately a frequent pattern in ATLAS is to create a container then update its contents - this confuses the GaudiHive scheduler of today

Results without Cloning

Events in Flight	Runtime (s) 100 events	Memory (MB)
1 Athena	305	445
1, 5 alg 30 th	283	550
2	175	590
3	161	630
4	161	680
10	161	950

Algs 5, up to 30 threads

- Only one instance of each algorithm
  - Throughput is bottlenecked by slowest single algorithm
  - Improvements plateau after 3 events in flight, after which you are partially processing more events at once

Algorithm Cloning

- Important to be able to clone the most expensive algorithms (and their tools)
  - This is an intermediate solution on the way to full thread safe algorithms*
  - Helps alleviate the choke point at the single most expensive algorithm
- However, in this case some algorithms were inherently thread unsafe and would have required substantial recoding to work
  - At least clone the easier ones then

* Probably only necessary for the most expensive algorithms

Results with Cloning

Events in Flight	Runtime (s)	Memory (MB)
1	305	450
2	175	570
3	135	618
4	129	667

- With this limited cloning bottlenecks are alleviated
- Best results are 2.3x increase in speed for a 28% increase in memory
  - Discounting initialisation, speed increase is 3.3x

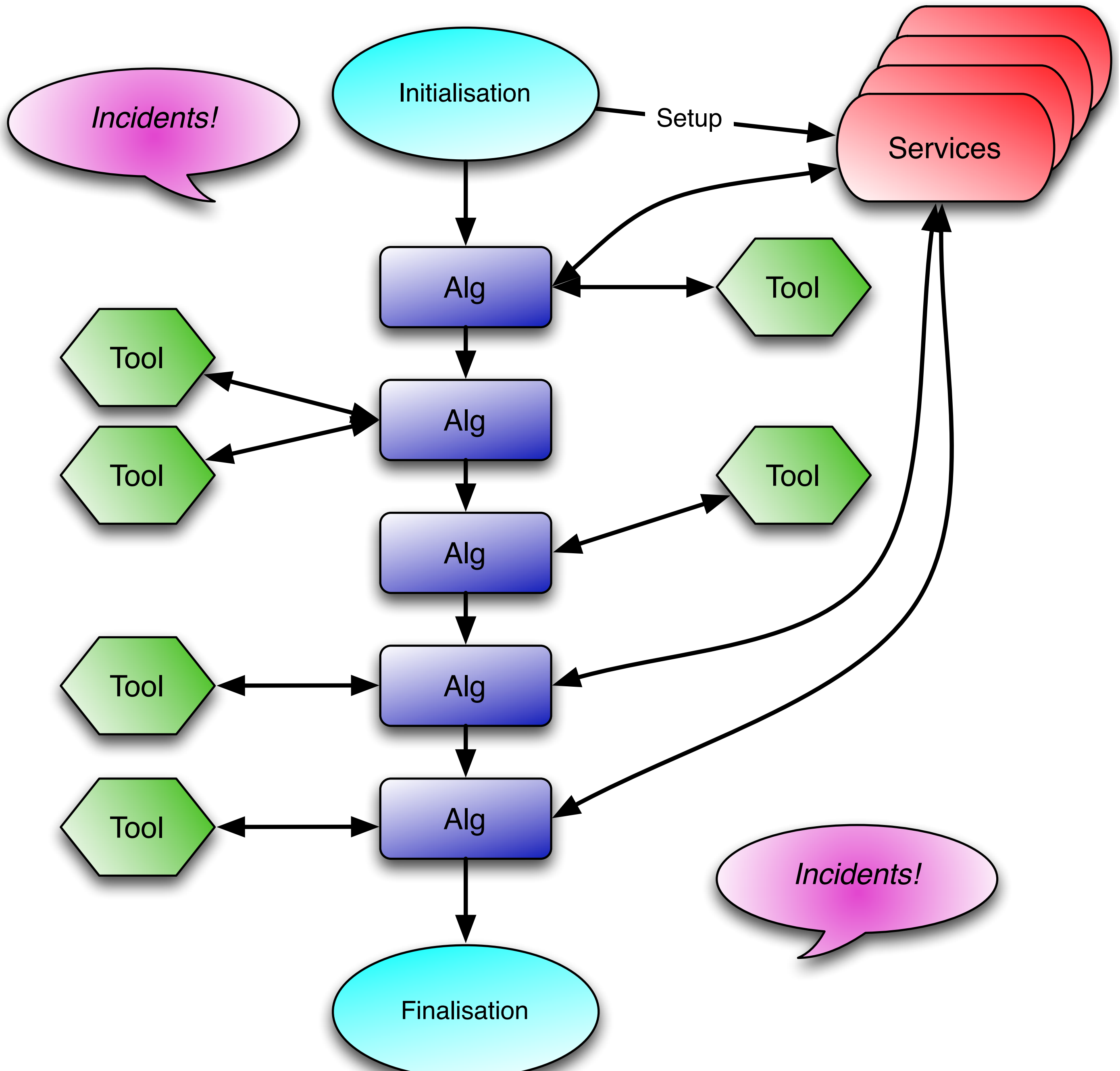
Testbed Conclusions

- Multi-threading can speed up processing at a lower memory cost than serial or multi-process techniques
- Not being able to clone certain Algorithms (and their tools) is a bottleneck
- How to fix?
  - make tools thread safe
  - split up monster algorithms into smaller ones
  - all requires significant intervention into user code.
- Incidents are definitely problematic
  - Experience shows that in many places ATLAS code uses this pattern, but in a way that only works in serial processing
- Access to the event store should have clearer patterns, more suited to a concurrent framework
- Data dependencies need to be automatically propagated to the lowest level component where they are used; algorithms may not know about the data they are using via their tools

Framework Requirements

- Encouraged by these results, ATLAS constituted a *Future Frameworks Requirements Group, FFREQ*
  - Plan how our framework should evolve for Run 3 and beyond
  - Incorporate the requirements of the ATLAS High Level Trigger (HLT)
    - At the moment, while code is shared between offline and HLT there are also substantial differences
- Emphasis on gathering requirements and use cases

Elements of the offline Framework



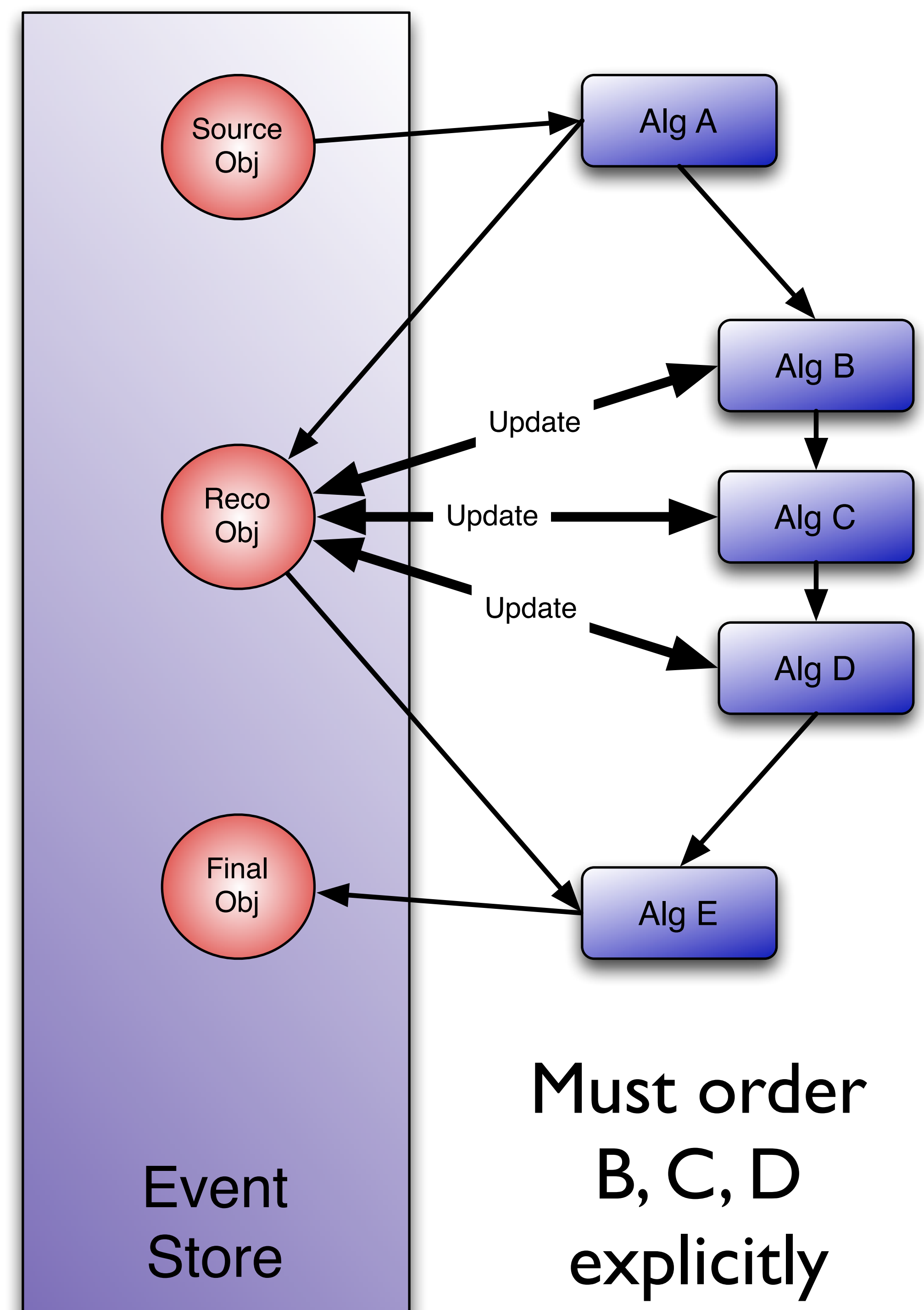
- At the start of processing services are setup (Event Store, Magnetic Field)
- Events pass through a sequence of algorithms
  - Read event and other data, calculate, write new objects
- Much work actually done through tools
  - Tools can be private (used within only one algorithm)
- Incidents are callbacks triggered by any event during the workflow

Changes we should make to framework patterns

- Building on the knowledge gained from running in the GaudiHive prototype:
  - Get rid of public tools (i.e., tools which are used by more than one algorithm)
    - Replace these with services, which need to be thread safe
  - Reduce reliance on incidents and replace them with data flow or control flow dependencies
  - Interactions with the event store should proceed via *data handles*
    - Data handles provide a well defined interface that declares an algorithm's intent with an event store object
      - Read, Write, Update

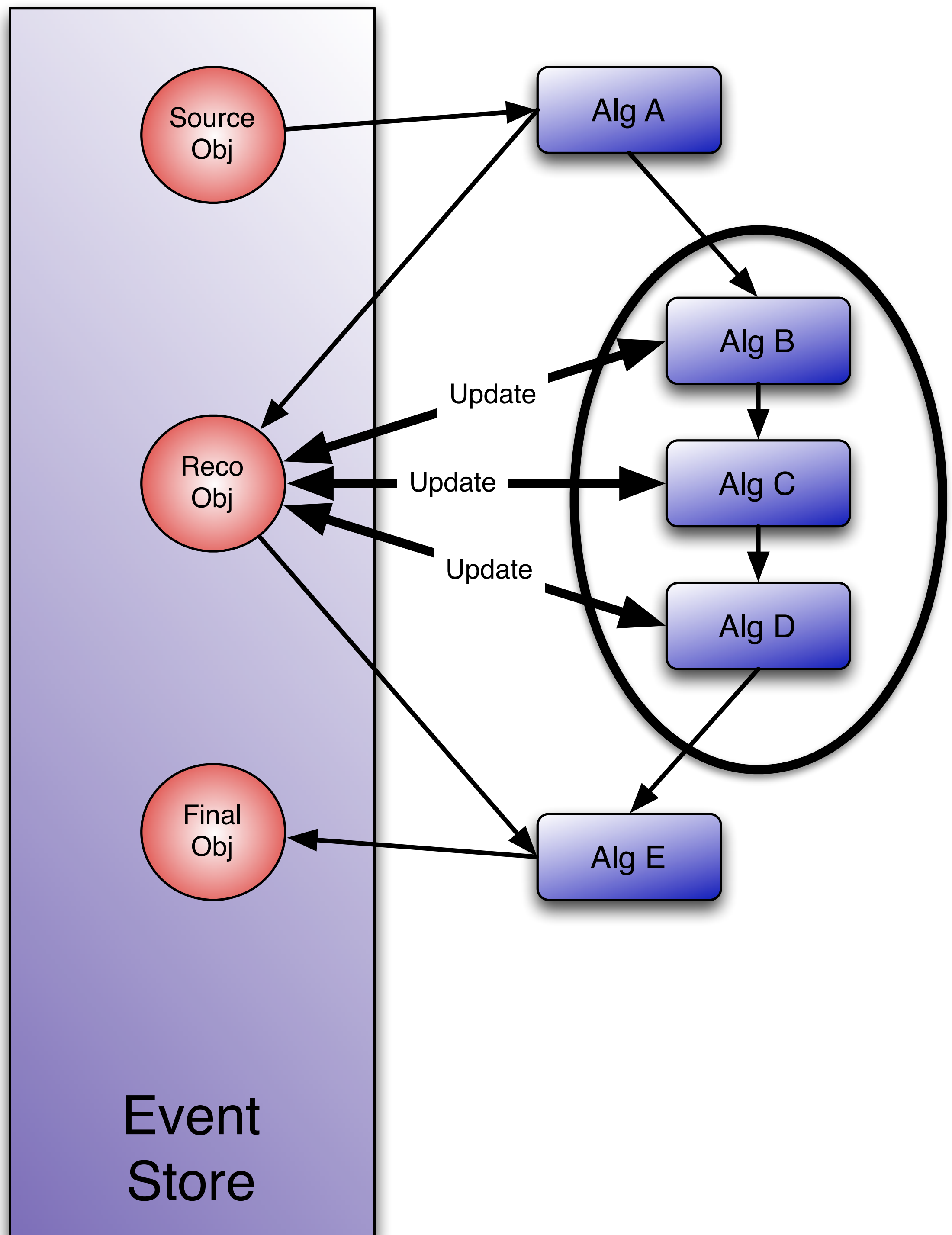
Mutable Event Store Data

- Event store data may be mutable
  - An algorithm can ask to read and write the same object in the store
  - e.g., performing calibrations or decorating the object
- If multiple algorithms wish to mutate the same data item then their order must be given explicitly at configuration time
  - Otherwise there is a configuration error
- Algorithms that only read the data object will be scheduled after all mutations
  - i.e., they get the final version of the object
- Obviously scheduling of B, C, D (or anything else that might update 'Reco Obj') is decided at runtime



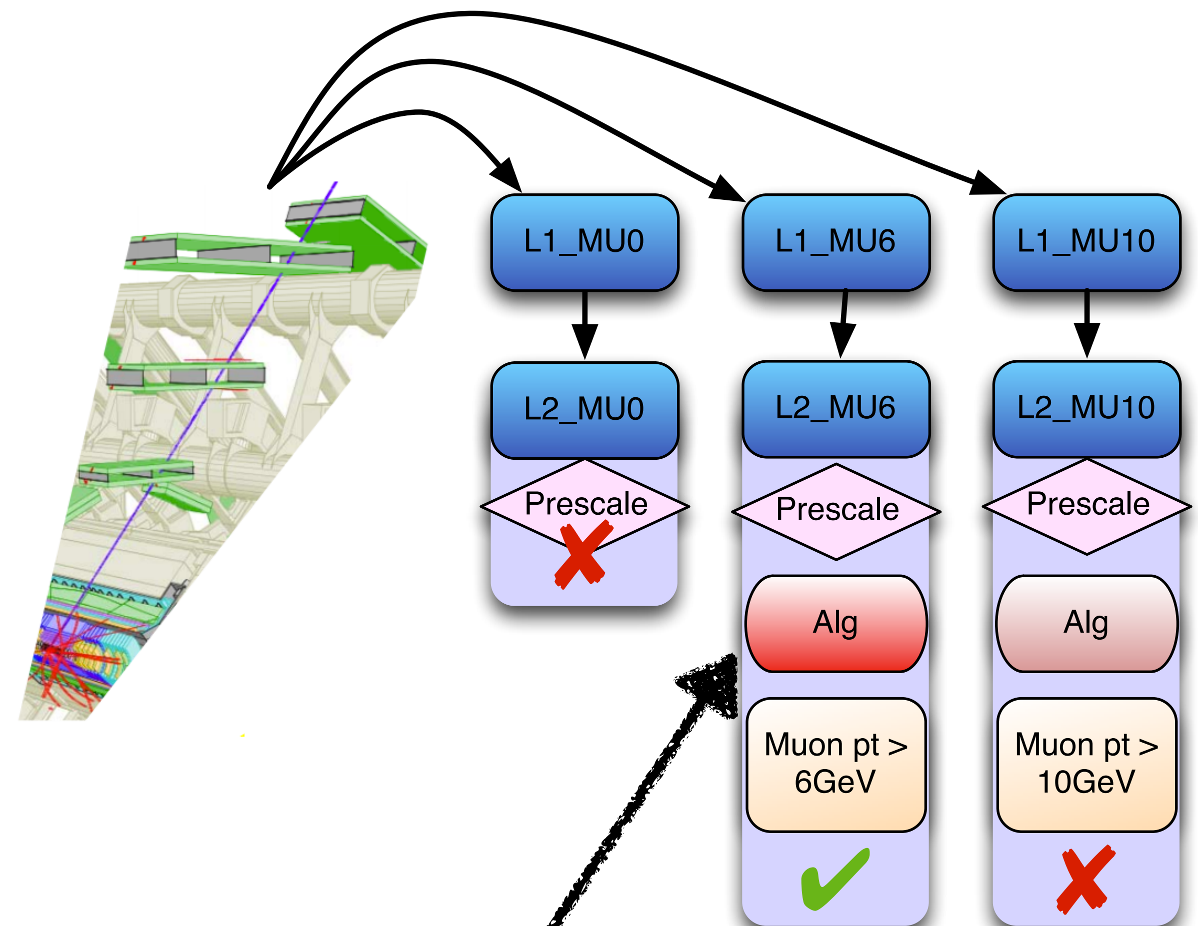
Super Algorithms

- These chained algorithms are schedulable units
- Scheduler only needs to be concerned with schedulable units
  - Will reduce the load on the scheduler
- One possibility is to schedule super algorithms as Threaded Building Blocks graphs
  - TBB graph scheduler is efficient even with small computational units (10^5 instructions)



HLT Picture

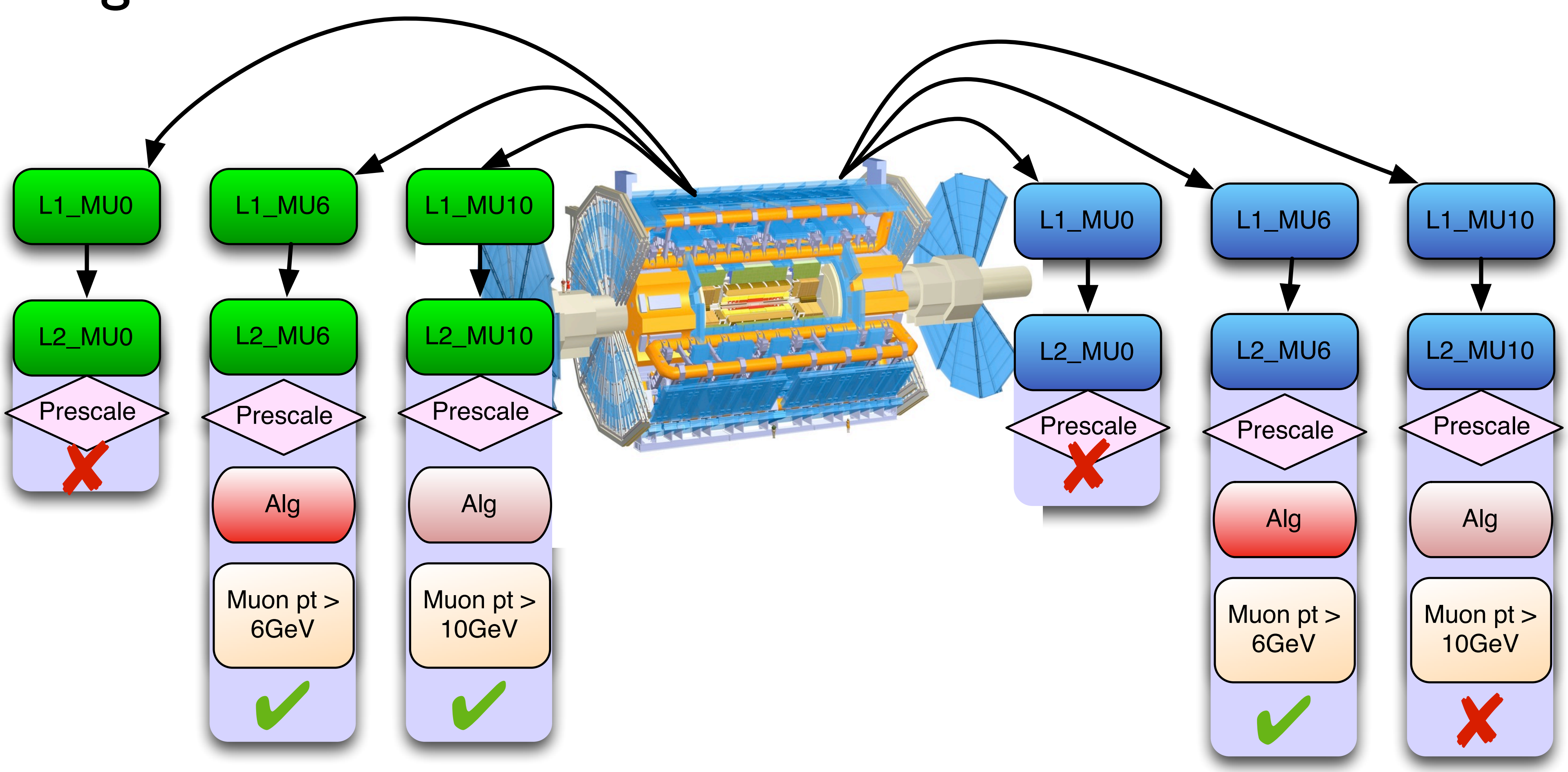
- HLT is more complicated than offline
  - Multiple regions of interest (RoI) seeded from level 1
  - Reconstruction starts within a RoI
  - Then every trigger chain must be run
    - Until a chain fails a hypothesis (e.g., $p_t >$ 10GeV) or event is accepted
  - 99% of events are rejected
- Need to minimise resource usage:
  - Data requests from read out system
  - CPU time/event (HLT farm size)
  - Time until event built & read out buffers cleared
- Events are built on demand



Different chains could configure the same algorithm, but only run once on the same data

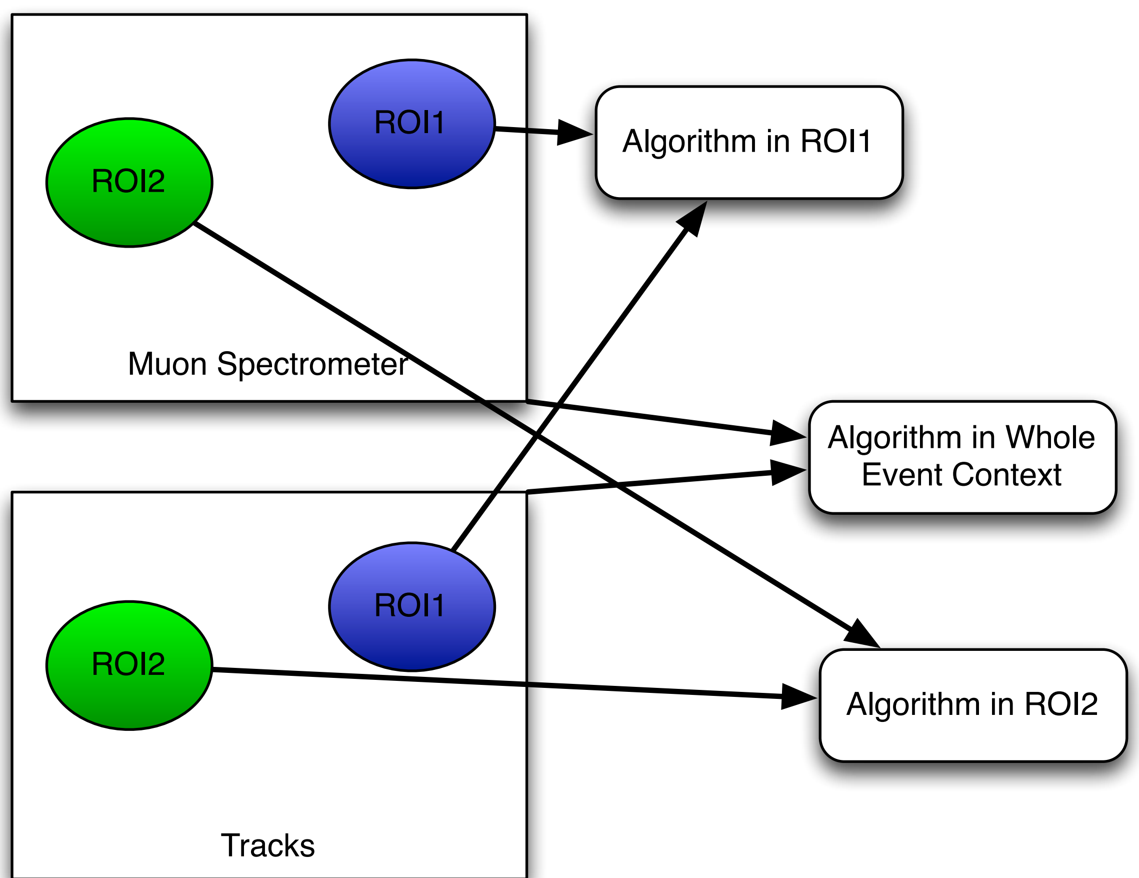
Regions of Interest

- Seeding multiple regions of interest runs the same algorithms in the trigger chain over different detector regions



Event Store Views

- To accommodate the HLT regions of interest, different views of the event store are needed



- This means that the combinations of Algs/RoIs become event dependent
- Complicates event scheduling
- Accepted events move to the 'whole event' context
- Views could also be useful offline - e.g., running more expensive tracking for taus

Exact implementation under discussion

Code Changes

- For simple algorithms that do not consume a lot of resources, single instances with no special constraints are fine
  - Physics code can remain more or less unchanged
  - However, patterns that were hostile to threading need to be removed (public tools, incidents)
- For time consuming code, care must be taken to ensure thread safety
  - Aim for cloning in existing code, true thread safety in new code
  - Thread-safety work can commence before the new framework is ready
  - Can already use multithreading within a single algorithm; works well with Hive
- Services which can be called multiple times in different contexts must be thread safe
- Essential to provide good programming models and training to the developer community

Summary

- Evolution of computing hardware makes multi-threading inevitable for HEP
- ATLAS has demonstrated that multi-threading can work successfully with Gaudi and Athena
- Now undertaking a careful look at requirements to design a framework for Runs 3 and 4
- Integrating trigger requirements needs modifications to the scheduler and the event store beyond the hive prototype
  - Offline may benefit from these extensions to speed up simulation and reconstruction
- Timescales are to design and implement the new framework during Run 2, ready for developers to adapt during LS2 and be ready for Run 3