Multi-threading in the ATLAS High-Level Trigger

Presented by Adam Barton
On behalf of the ATLAS Collaboration

Introduction

- By 2021 the LHC will be producing collisions at a higher rate than ever before.
- By 2026 ATLAS' *trigger* will need to be able to handle an input rate between **4x and 10x the current maximum** of 100kHz.
- Using predicted hardware specifications and budget constraints our multi-process approach will be insufficient.
- A given "event" (record of a collision) will also be more busy, requiring more complex algorithms such as those currently used in "*Offline software*".

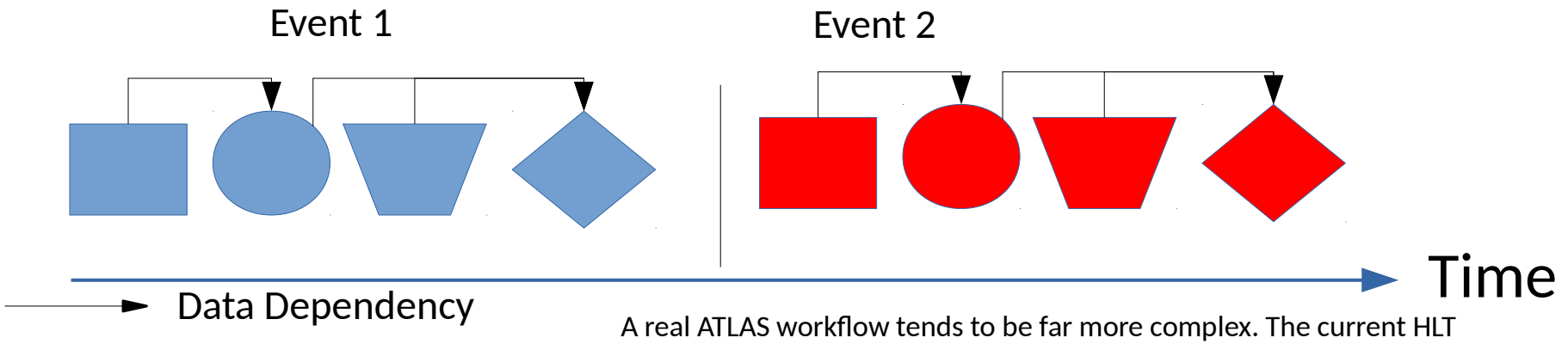
Glossary
Luminosity – "rate of proton collisions"
Trigger – "hardware and software system that rapidly selects data to record"
Offline software - "software used to analyse recorded data"

Outline

- The current system Athena
- The Limitations
- AthenaMT
  - Algorithm Design
  - Multiple events in flight
- AthenaMT and the trigger
  - EventViews
  - Trigger Menus
  - Control Flow
- Conclusions

Athena

- Athena is ATLAS' primary trigger, offline reconstruction and analysis framework, it is a system of C++ libraries configured from python or database.
- Athena implements many of the components of **Gaudi** – a inter-experiment framework.
- Algorithms are run sequentially over each event and data is passed via a singleton service "EventStore".
- Each algorithm is implicitly data-dependent on a previous one.



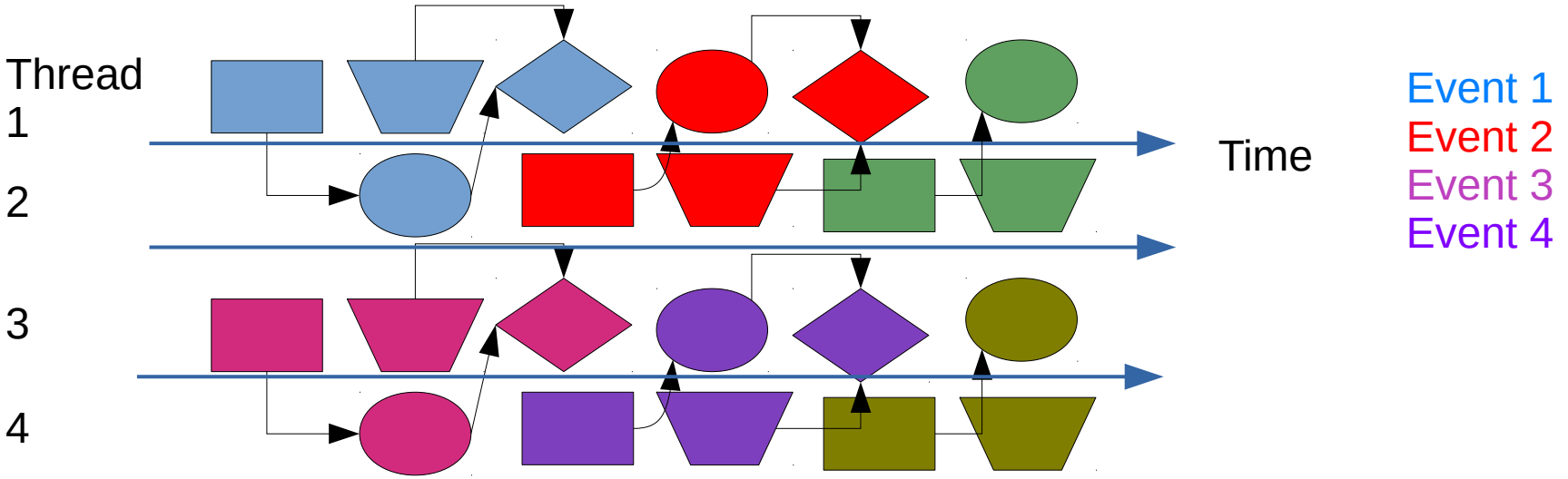
A real ATLAS workflow tends to be far more complex. The current HLT includes ~150 algorithms in ~2000 dependency chains

Core revolution

- To exploit multiple cores we current use a multi-process framework AthenaMP.
  - Each process executes a single event sequentially
  - Memory is shared with a copy-on-write mechanism (causes overhead)
- Algorithms for any given event are still processed on a single thread.
- This multi-process approach will not be sufficient in the future if core-counts continue to increase and memory costs remain static – memory constrained.
- Memory sharing in a new concurrent framework will have less overhead and allow further optimizations.

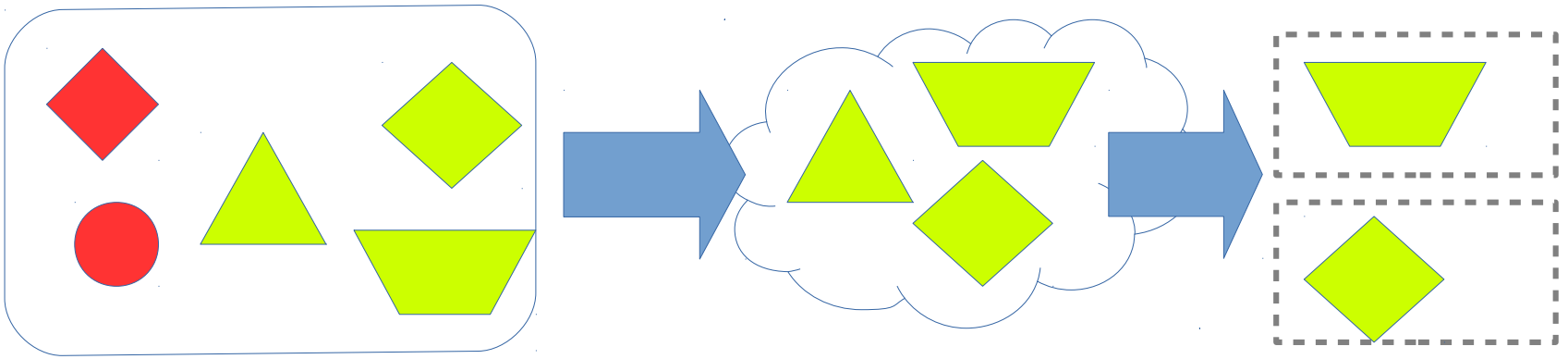
AthenaMT

- AthenaMT will use components of the multi-threaded framework GaudiHive.
- AthenaMT will use the *Intel Thread Building Blocks* library to execute algorithms on available CPU threads.
- Algorithms using data from one event/collision can be parallelized and multi events can also run in parallel . (**Intra**-event and **Inter**-event parallelism).



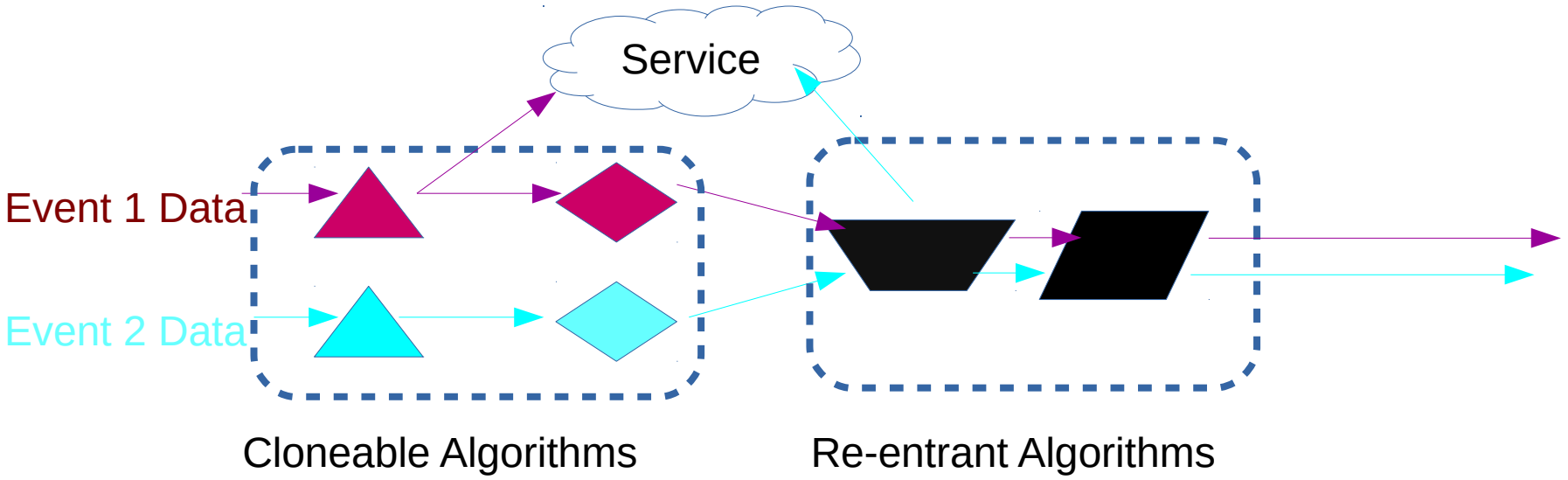
Algorithm Design

- In current Athena algorithms are scheduled in a manually defined sequence.
- This would be very sub-optimal in a MT environment yet data-dependencies must be respected.
- AthenaMT will implement a data-dependent scheduler.
  - Each algorithm will explicitly declare its dependencies (“DataHandles”).
  - The dependency graph is constructed and verified during initialization.
  - The scheduler will create TBB tasks to run algorithms as their dependencies are met.
  - TBB will run tasks minimizing thread idle time.



“Multiple Events in Flight”

- Algorithms executed concurrently in different threads may be from the same event or different events (multiple events in flight).
- Algorithms will be can to be either re-entrant (purely procedural) or cloneable – singleton algorithms may be permitted as a migration step.
- Singleton services will be required to be thread-safe.



AthenaMT and the Trigger

- The ATLAS trigger system is based on the concept of Regions of Interest (RoIs).
  - Algorithms run over separate geometric regions of the detector only reconstructing parts of the detector identified by the Level 1 trigger (on average ~10%).
- The goal for AthenaMT is to support both offline and HLT processing, to allow sharing of algorithmic code:

Offline Requirement

All algorithms are run once per event

All algorithms share data via data-stores isolated from other **events**

Online Requirement

Algorithms can be run multiple times per event or not at all

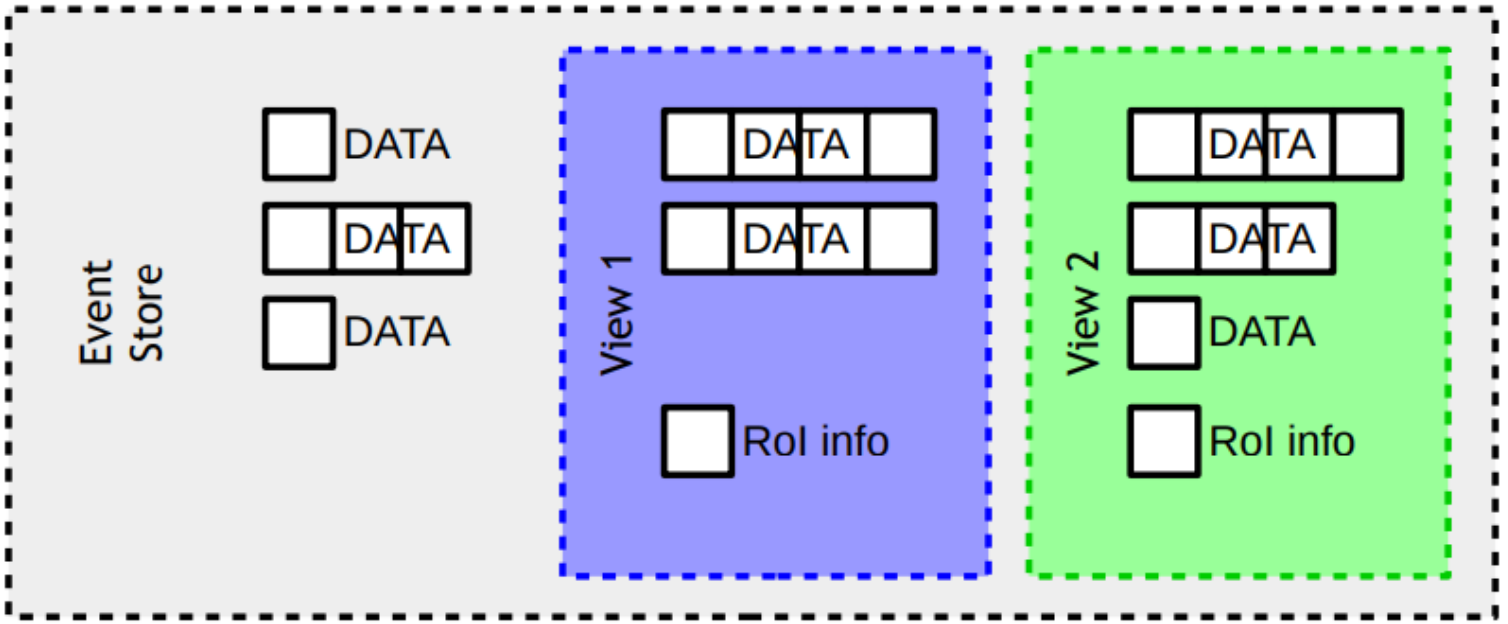
All algorithms share data via datastores isolated from other **RoIs**

AthenaMT and the Trigger

- Currently the HLT uses:
  - 150 algorithms (connected by ~2000 “chains” defining data flow and selection)
  - 400 tools (configurable sub-components of algorithms)
  - 100 services (globally accessible interfaces to I/O, configuration, etc.) - must be thread-safe
- By default offline algorithms process all available data and are unaware of RoI requirements.
- To avoid having to adapt all the necessary offline algorithms a transparent method of limiting them to the required RoI is required.

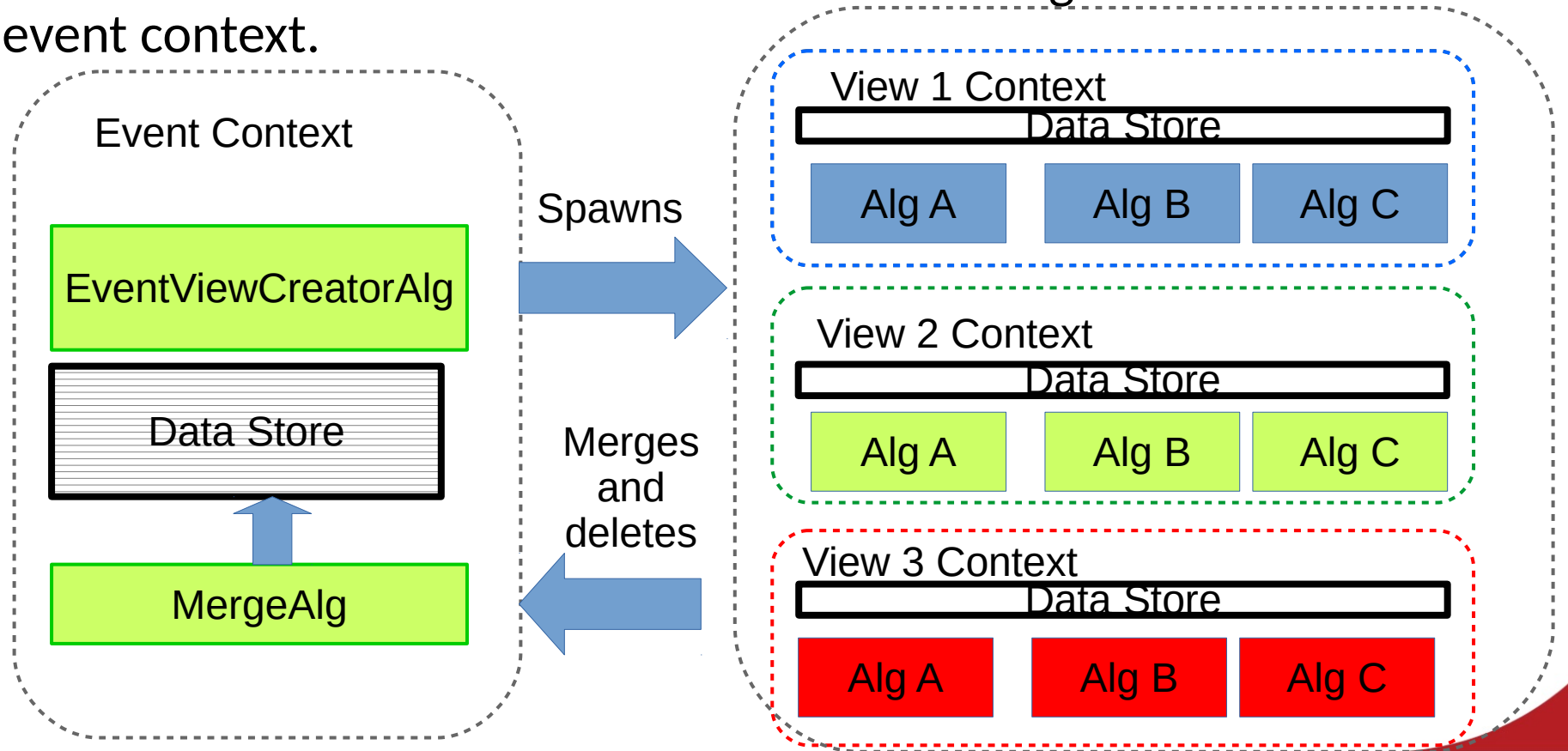
EventViews

- To allow for RoI processing without redesigning all the offline algorithms we have developed “EventViews”.
- EventViews implement the EventStore interface but will provide objects from specific regions of interest.
  - algorithms will not need to be altered and will be transparently directed to the correct ‘EventView’.



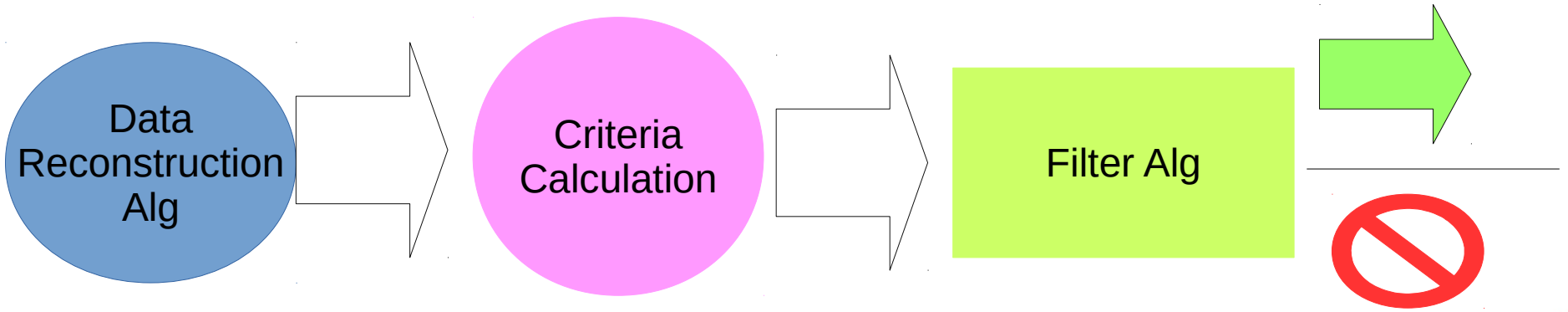
EventViews

- A creator algorithm will run in the event context to create EventView stores within the primary store. Algorithms can then run in their respective view context.
- Data from views can then be accessed and merged back into the main event context.

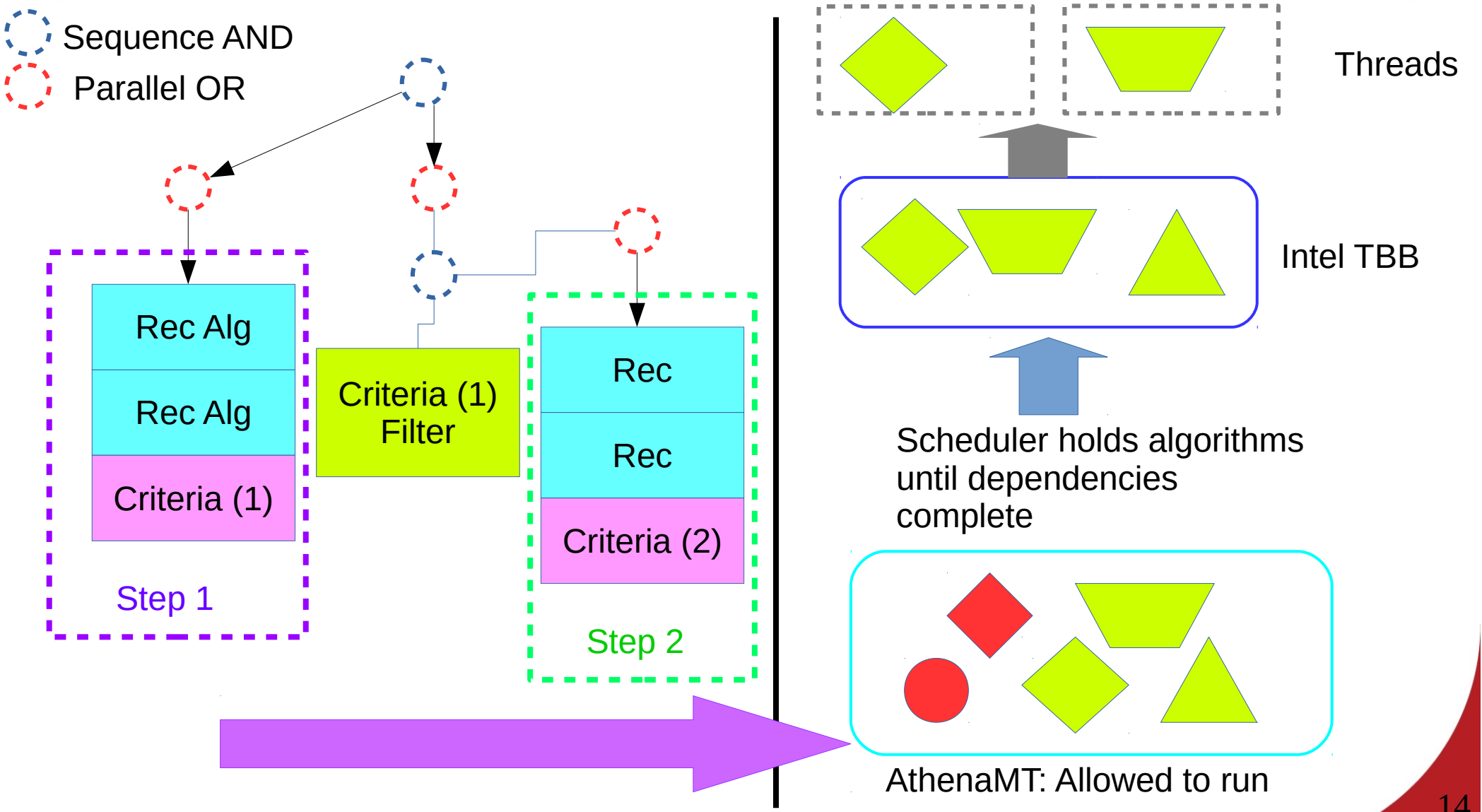


Trigger Menu

- The trigger menu is a list of conditions and the event is accepted if any condition is met.
- HLT performs reconstruction incrementally in chains of steps of reconstruction+selection.
- Filter algorithms placed in the chain prevent execution of subsequent algorithms following a failed selection in order to save CPU time (uses control flow).
- If no chains are left active the event is rejected - “early rejection” - 99% of events.



Control Flow



Conclusion

- ATLAS is redesigning its analysis framework to cope with increasing data demands exploiting expected trends in hardware.
- The trigger will upgrade its software to exploit this and unify the online and offline code-bases.
- A basic set of algorithms are now functional.
- Algorithms will be migrated over the following years.
- We expect to have a first implementation by the end of 2018 and a full implementation by 2020.

Backup

Control Flow

- This method ensures only required algorithms are run.
- Fast-running high-rejection algorithms are placed first for most efficient rejection.
- Accepted events may “Rerun” the menu to fill in previously vetoed data the mechanism will not rerun previous accepted data.

Types of Sequence

- The offline AthenaMT scheduler replaces steering, but needed to be expanded to include control flow mechanism:
- Step-wise control flow (AND sequence)
  - Sequence = (chain1_step1 || chain2_step1 || …)
    && (chain1_step2 || chain2_step2 || …) …
- Chain-wise control flow (OR sequence)
  - Sequence = (chain1_step1 && chain1_step2 && …)
    || (chain2_step1 && chain2_step2 && …)
  - Can be configured to use concurrent-ORs