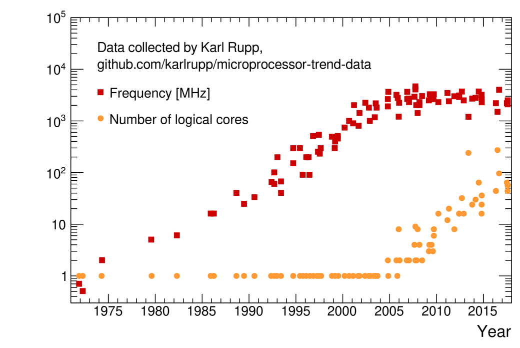
ATLAS High Level Trigger within the multi-threaded software framework AthenaMT

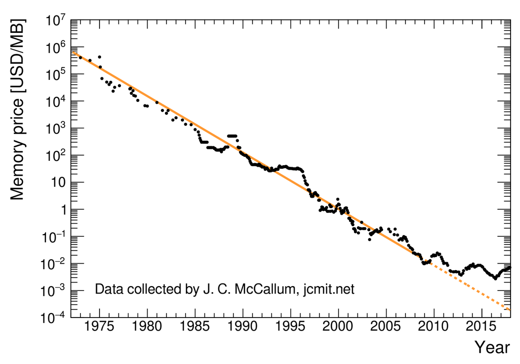
19th International Workshop on Advanced Computing and Analysis Techniques in Physics Research, 11–15 March 2019, Saas-Fee

AthenaMT

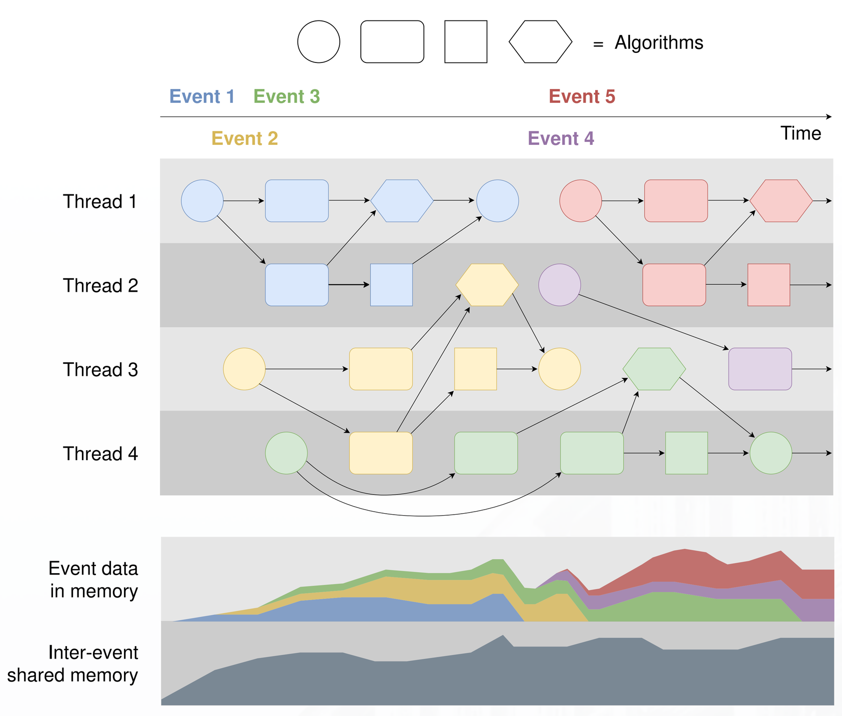
Motivation

- Athena is the ATLAS software framework used in trigger, reconstruction, simulation and analysis
- Based on Gaudi – core framework shared with LHCb
- Designed in early 2000s without multi-threading in mind
- The computing market transitioned towards many-core CPUs while memory price plateaued → **less memory/core**
- Already in Run 2 ATLAS struggled to use the processing resources (WLCG, Tier0) efficiently with Athena
- A stopgap solution was to use forking to reduce memory per process (thanks to copy-on-write)
- Ultimate solution → redesign the core framework for native, efficient and user-friendly multi-threading support → **AthenaMT**





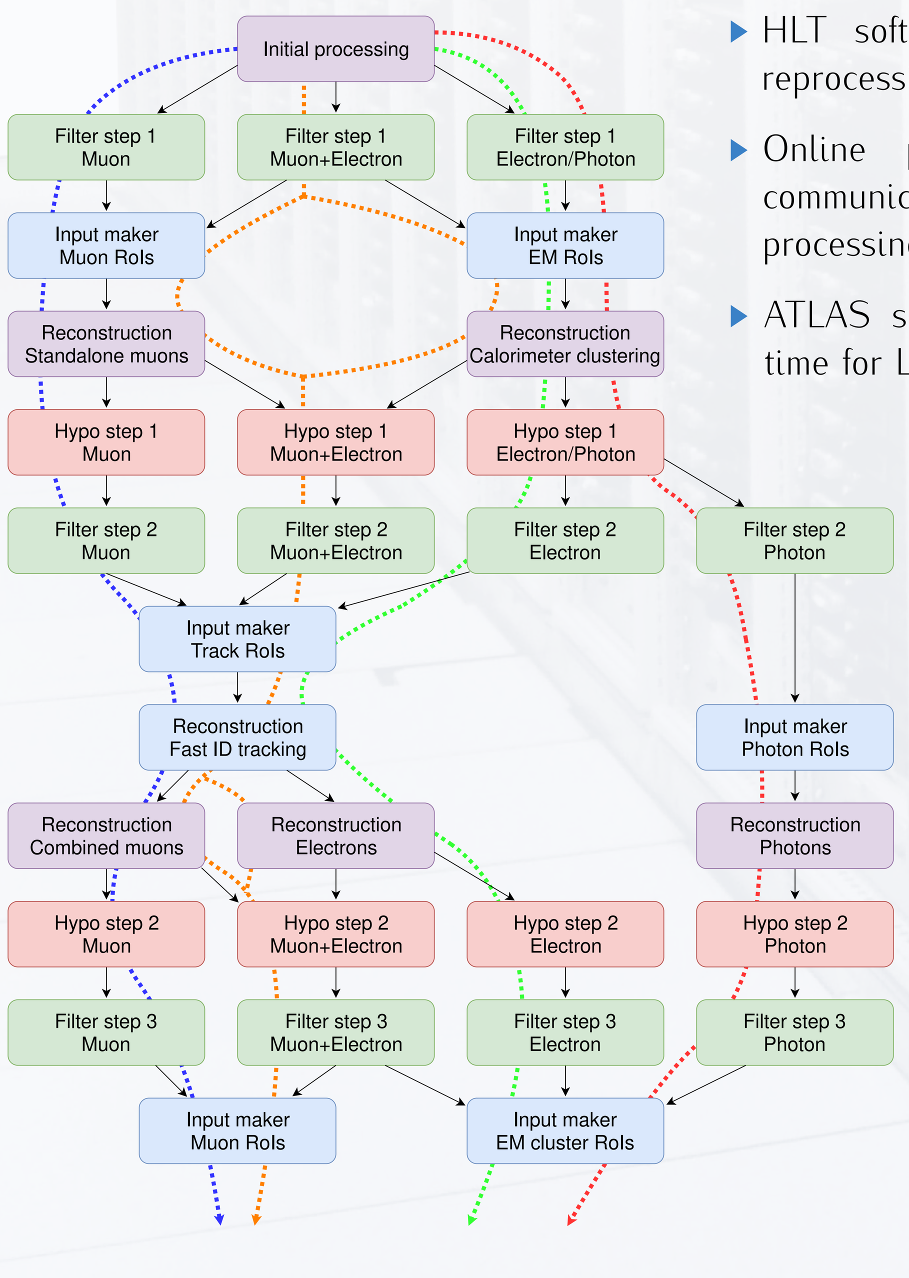
Implementation



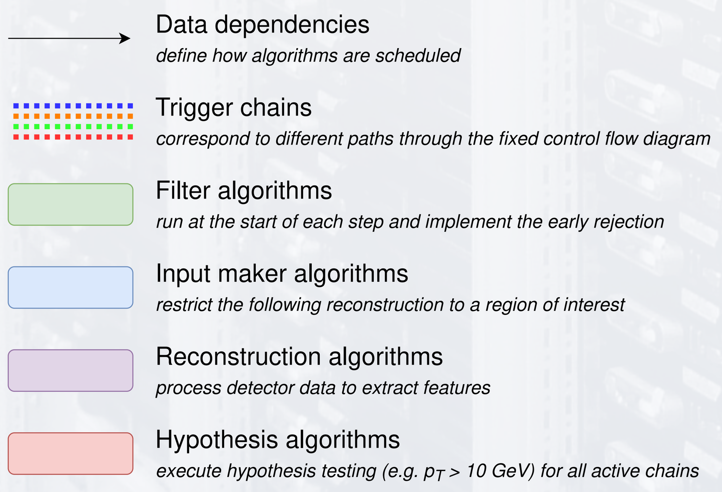
- Based on GaudiHive which uses Intel TBB
- Both **inter-event** and **intra-event** concurrency
- Defines algorithm execution order based on data dependencies declared as ReadHandles and WriteHandles
- Decides when to execute an algorithm based on input/output and the configured number of **threads** and **event slots**
- When input dependencies are met, Scheduler pushes the algorithm into an Intel TBB queue
- AthenaMT design encompassed the HLT requirements from the beginning, e.g. support for partial event data processing

High-Level Trigger in AthenaMT

- Taking the opportunity of AthenaMT migration to rewrite the HLT framework
- Run-2 HLT framework used a dedicated top-level algorithm taking care of algorithm scheduling
- HLT in AthenaMT is closer to the offline reconstruction framework – using the Gaudi Scheduler and removing the trigger-specific layer allows to use offline algorithms directly in HLT without wrappers
- Processing of partial event data (**regional reconstruction**) integrated in Gaudi as **Event Views** – algorithms can use partial or full data as input without any modification
- HLT Control Flow configures an execution graph including Event Views preparation (Input Maker) and **early termination** of an execution path if trigger not accepted (Filter Step)
- Each HLT chain corresponds to an execution path through the CF graph

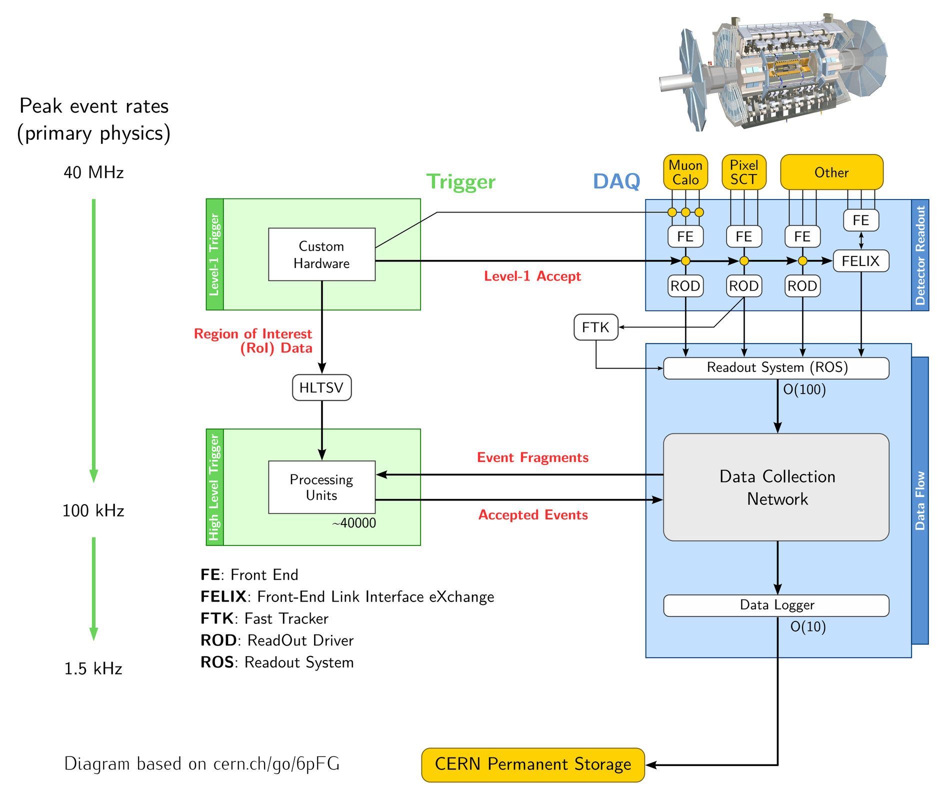


- HLT software is used both online and offline (data reprocessing, simulation)
- Online processing requires a dedicated layer of communication to the TDAQ System, but the event processing remains unchanged w.r.t. offline
- ATLAS software upgrade needs to be fully finished in time for LHC Run 3 starting in 2021



ATLAS TDAQ System

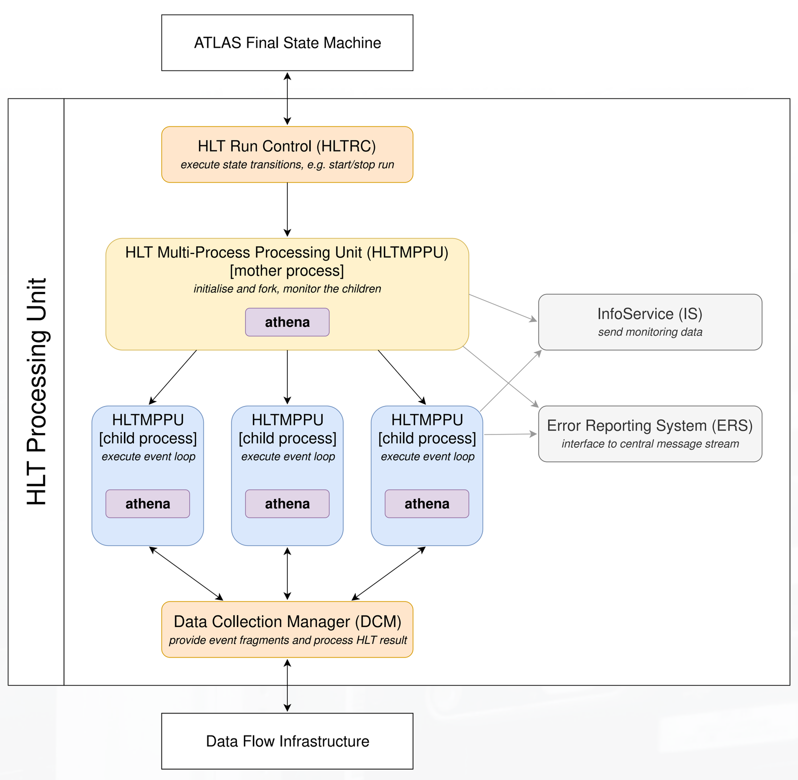
Data flow

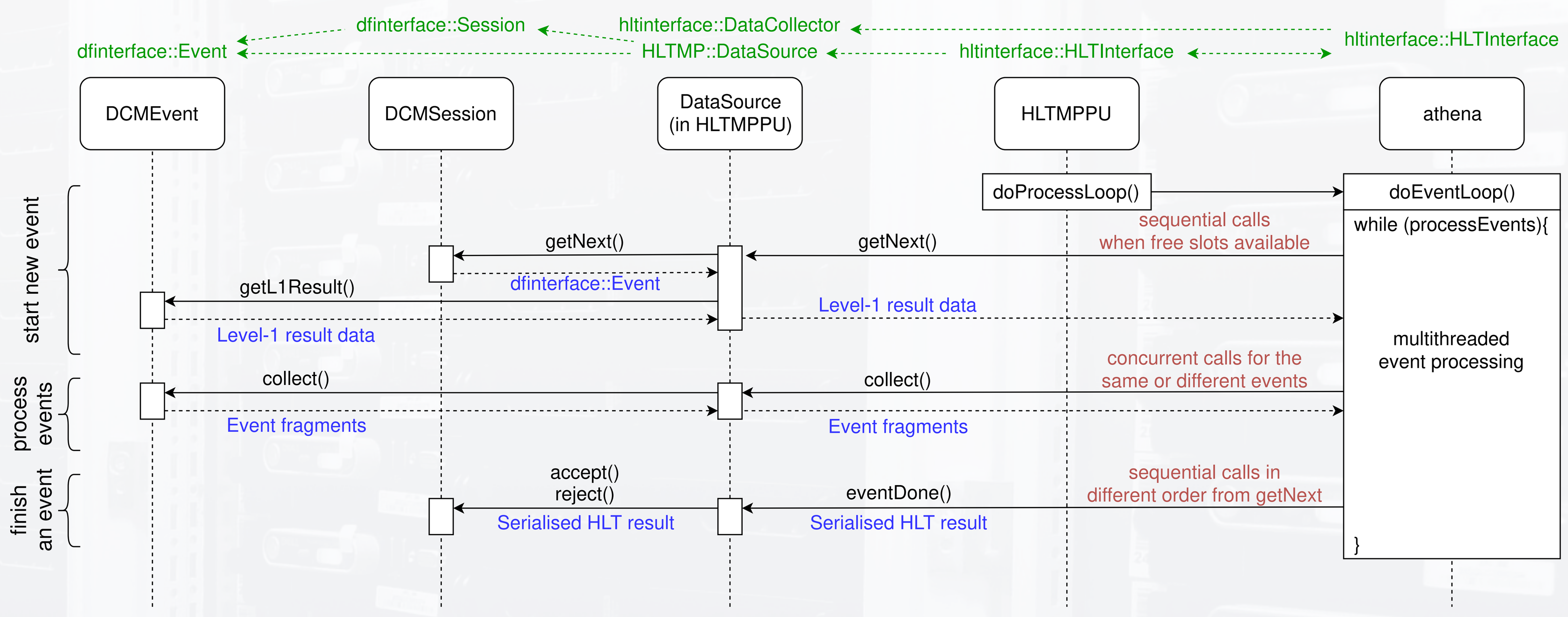


- Hardware Level-1 Trigger reduces the rate to 100 kHz
- Software HLT is required to provide further reduction to ~1.5 kHz
- Run-2 HLT consisted of ~40k physical cores and typically ran one instance of athena per core
- Run-3 HLT infrastructure will remain similar to Run-2

HLT Processing Unit applications

- HLTPU structure in Run 3 will consist of the same applications as in Run 2, but the data flow within the HLTPU will change
- Keep using **multi-process** approach, but now each fork has an athena instance which can run **multiple threads**
- Large flexibility for optimising performance of the system – adjust number of forks, threads, event slots
- In Run 2, HLTMPPU steered the event loop, requesting events from DCM and executing athena for each event sequentially
- In Run 3, Athena will **actively request events** from DCM (via HLTMPPU) when it has free processing slots





Operating AthenaMT within TDAQ

- The online-specific layer implements additional requirements for data-taking operation and integration with the TDAQ system
  - Reading/writing ROOT files replaced with an interface to TDAQ applications (**DataCollector**)
  - Extended error handling to prevent application exit where possible – send erroneous events to a special data stream ('**debug stream**') for later investigation and recovery into physics streams
  - Additional thread to monitor event processing time and interrupt **timed-out events**
- Multi-threading brings new crash debugging challenges
  - Cannot determine which concurrently processed event crashed the application – send all to the debug stream and investigate all of them offline
  - More concurrent events = more good events in the debug stream in case of a crash
  - Execution order depends on the machine performance – **possible irreproducibility** of problems
- Performance measurements will be needed to determine the optimal number of forks, threads and slots

Rafał Bielski, CERN
on behalf of the ATLAS Collaboration