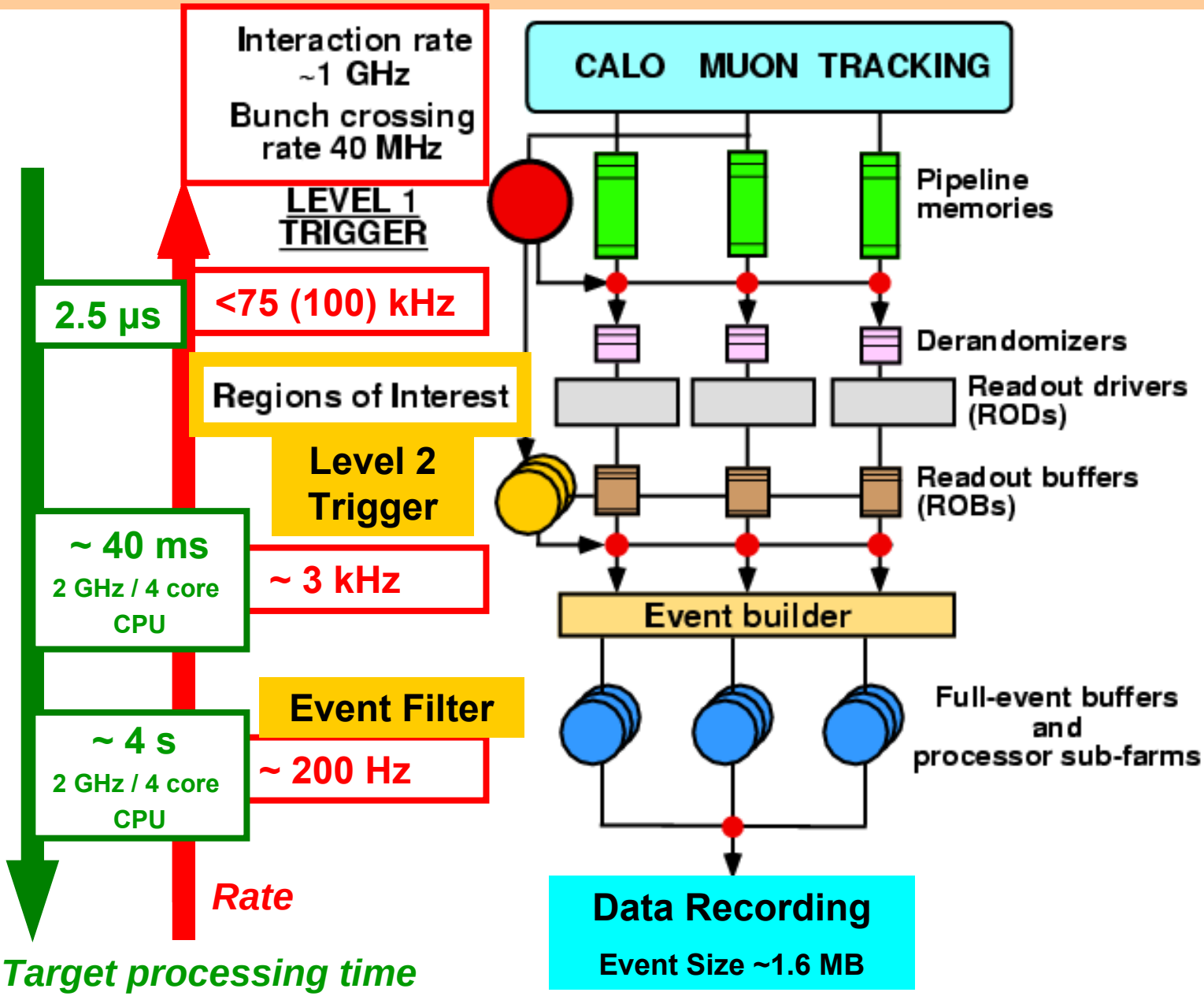
The ATLAS Online High Level Trigger Framework: Experience reusing Offline Software Components in the ATLAS Trigger

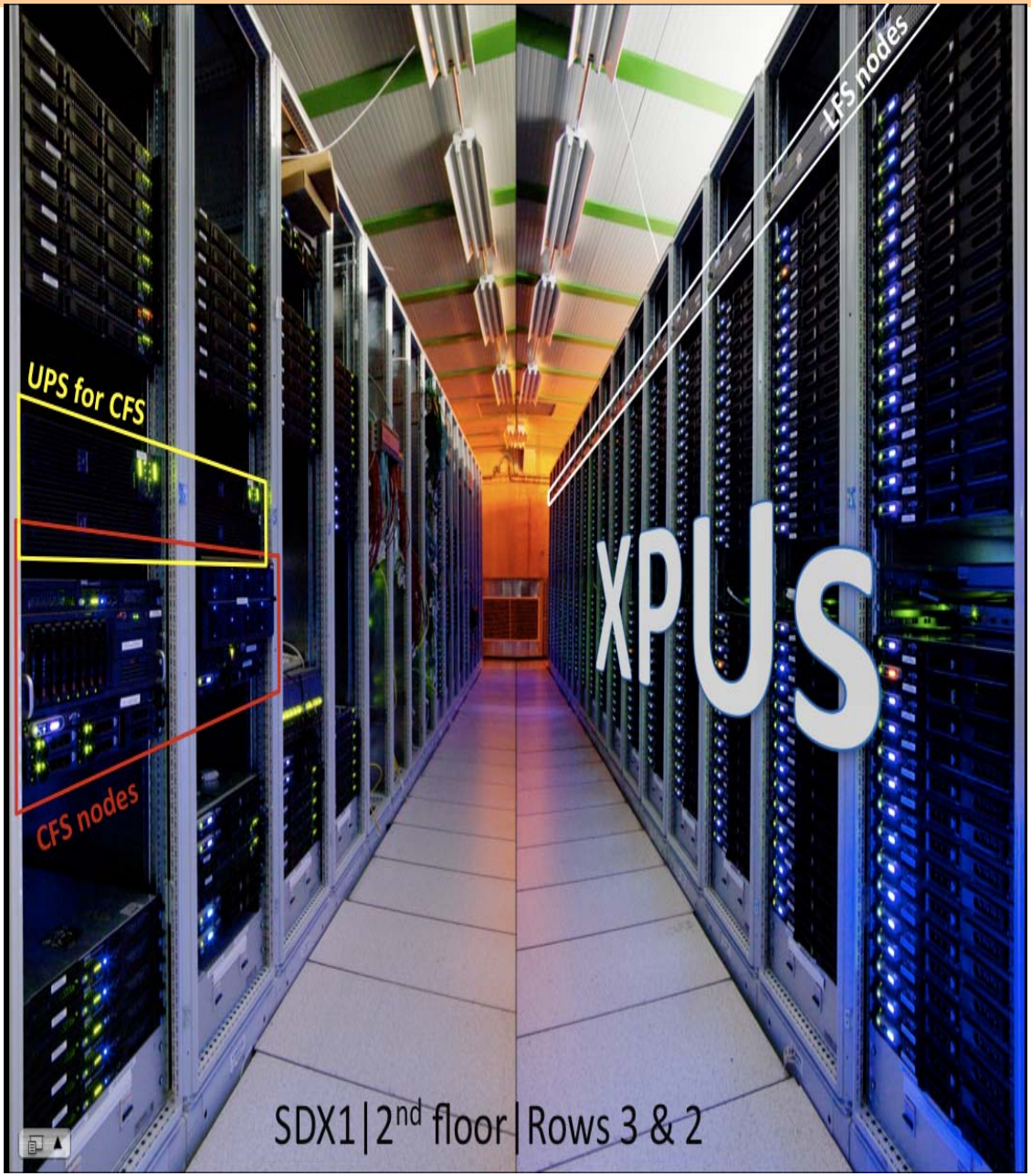
Werner Wiedenmann
University of Wisconsin, Madison, Wisconsin
on behalf of the ATLAS Collaboration

CHEP 2009
Prague, 21-27 March 2009

The ATLAS Trigger



HLT Hardware



- Final system:
  - 17 L2 racks
  - 62 EF racks
  - 1 rack = 31 PCs (1U)
  - 28 racks configurable as L2 or EF (XPU)
  - 1 Local File Server per rack → served by Central File Servers (CFS)
- 27 XPU racks installed
  - ~35% of final system
- XPU
  - CPU → 2x Harpertown quad-core @ 2.5 GHz
  - 2 GB Memory/core
  - Network booted

ATLAS High Level Triggers

- **Level-2** (75 kHz→3 kHz)
  - *Partial event reconstruction* in *Regions of Interest* (RoI) provided by Level 1→ event data in RoI are requested from selection algorithms over the network from the readout system
  - HLT selection software runs in the Level-2 Processing Unit (L2PU). Selection algorithms are run in a *worker thread*.
- **Event Filter** (3 kHz→200 Hz)
  - *Full event reconstruction* seeded by Level-2 result with *offline-type algorithms*
  - Independent *Processing Tasks (PT)* run selection software on Event Filter (EF) farm nodes
- **HLT Event Selection Software is based on the ATLAS Athena offline Framework**
  - *HLT framework* interfaces the HLT event selection algorithms to online
  - Driven by run control and data flow software → Event loop managed by data flow software
  - Allows *HLT algorithms* to run unchanged in the trigger and offline environment

HLT software in the Level 2 and Event Filter Data Flow

Level-2

Event processing in Worker threads

Online/Data Collection Framework

Data Collection

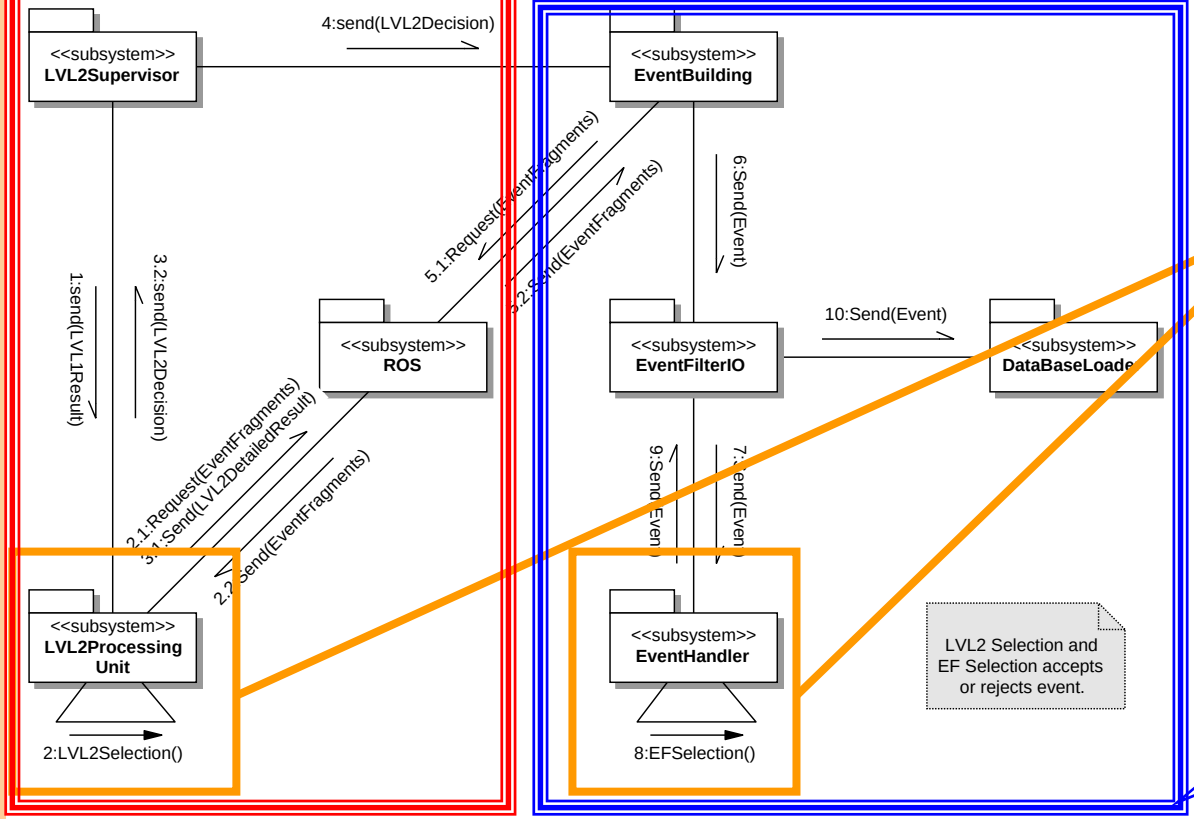
L2PU/EFPT

Steering Controller

HLT Event Selection Software

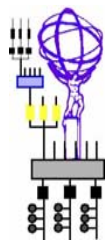
HLT Framework

based on ATHENA/GAUDI

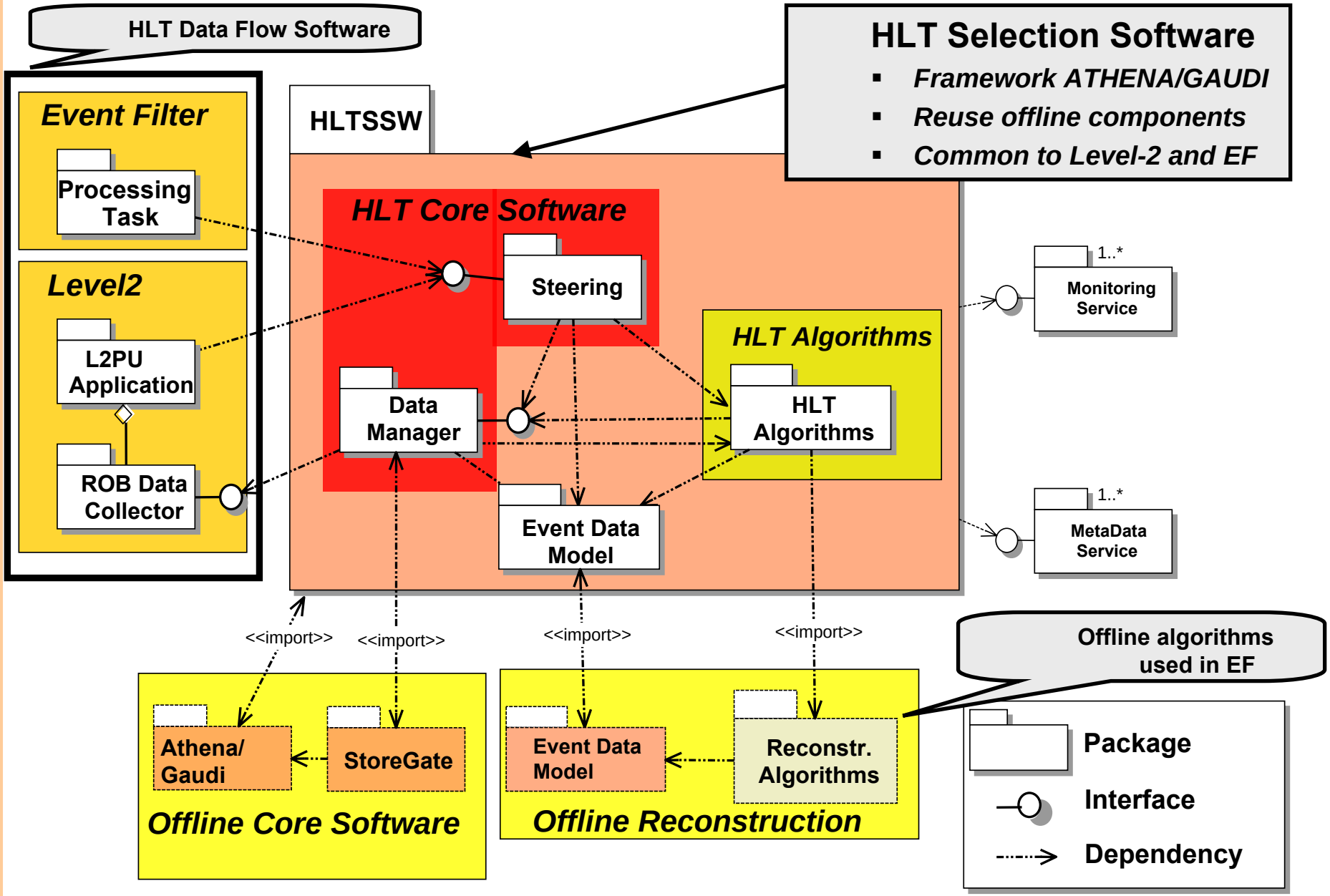


Event Filter

Event processing in Processing Tasks



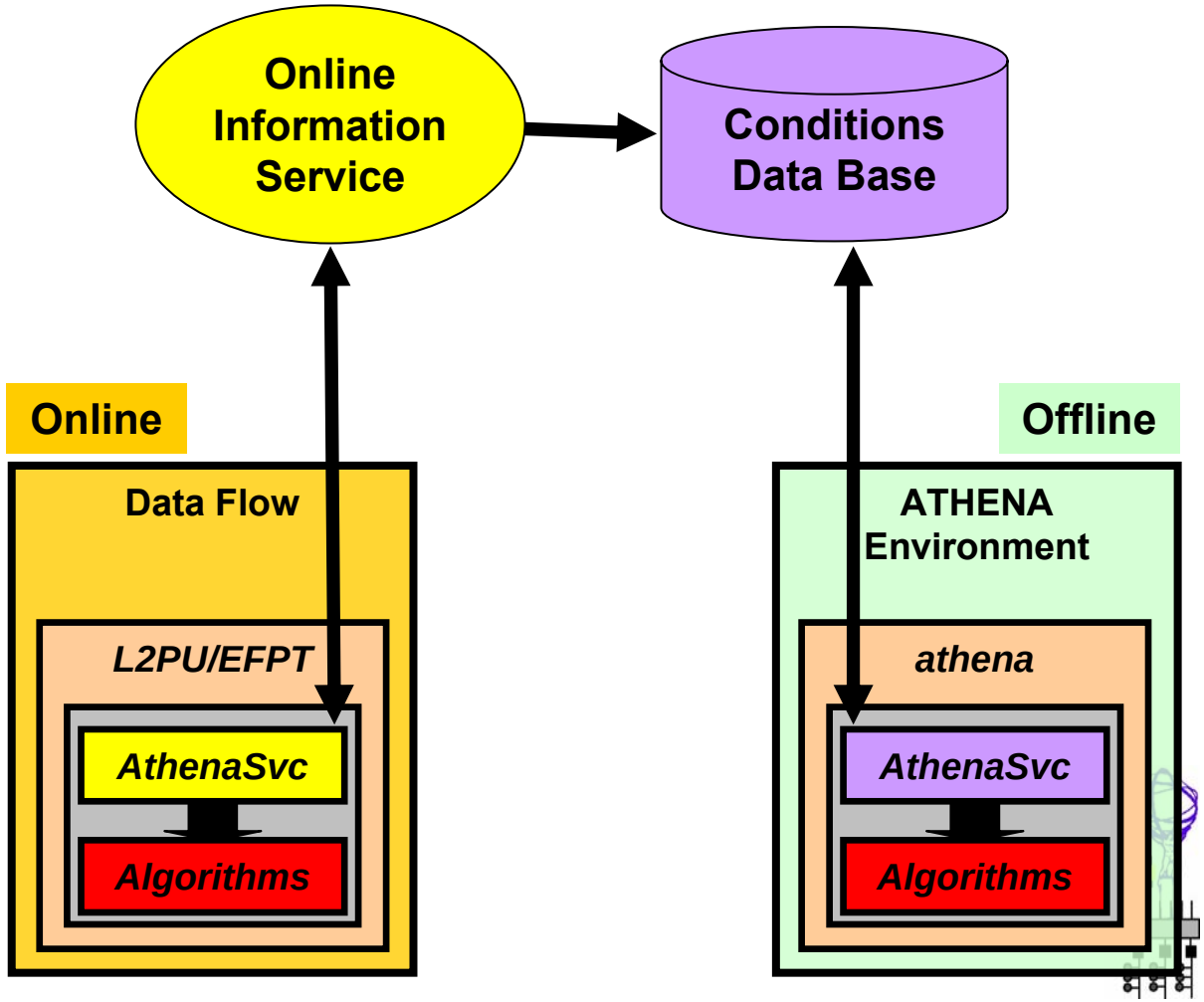
HLT Event Selection Software



HLT Framework

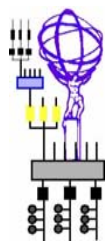
Provides special Athena services when necessary to allow algorithms a transparent running in online / offline. Whenever possible the offline infrastructure services are reused. Athena services with a special "online" backend are used e.g. for

- ***Interfacing to online run control, state machine and the online configuration database***
  - e.g. for the readout buffer configuration
- ***Interfacing the algorithm configuration to the Trigger Database***
  - Holds menu configuration
- ***Interfacing to online conditions data***
- ***Forwarding of data requests to data flow in Level 2***
- ***…***



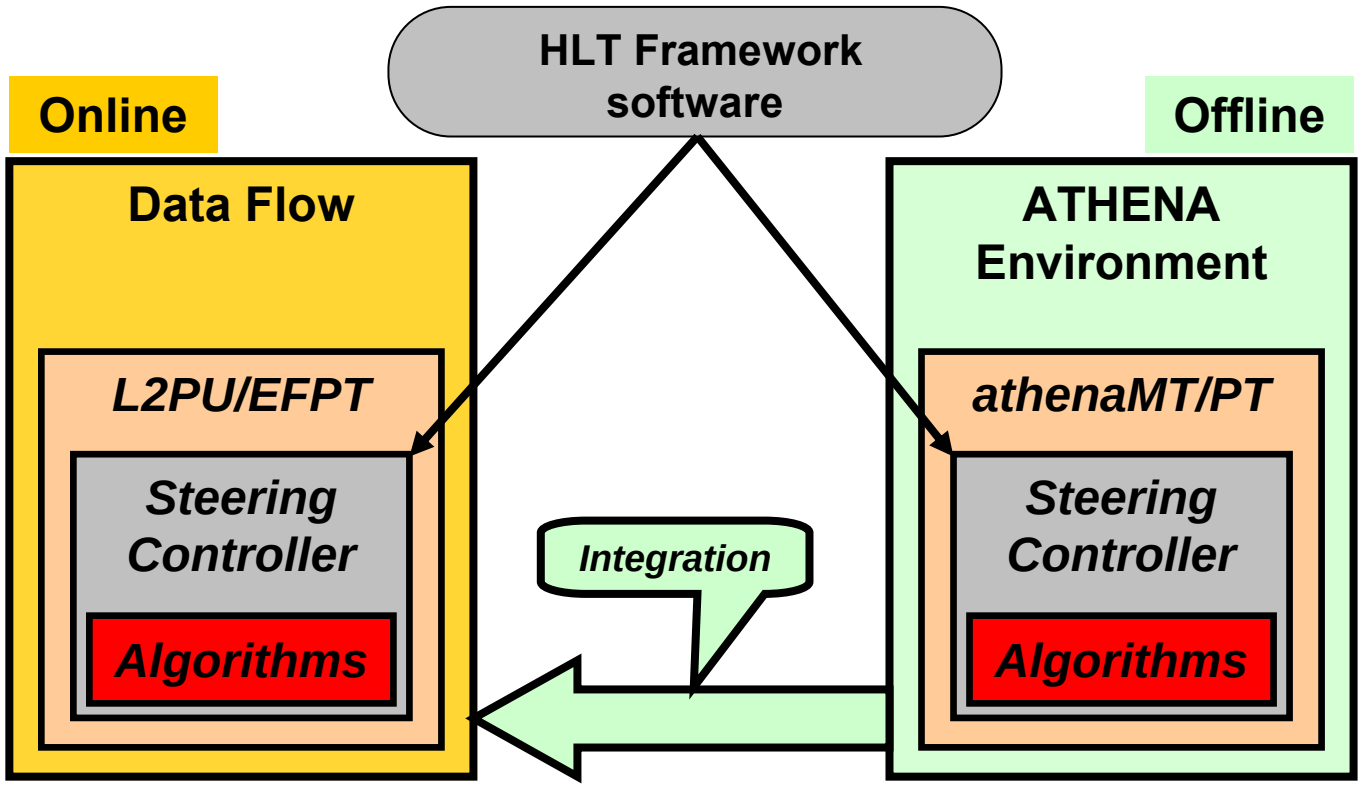
HLT Framework

- The HLT framework is responsible
  - For packaging and forwarding of the HLT decisions to online, e.g.
    - Detailed Level 2 and Event Filter decision record for subsequent reconstruction steps
    - Event output destination (Stream tag information)
    - Use of event for calibration purposes and building event only partially
    - ….
  - For error handling and forwarding of error conditions from algorithms to online applications
  - For forwarding of monitoring histograms and counters to online monitoring infrastructure
  - …
- Level-2 and Event Filter share many common components from the framework → differences mainly due to different data access in RoI or of full event



Development Model

- HLT algorithm development, testing and optimization is done mainly in offline environment.
- Developers can use online emulators athenaMT for Level-2 and athenaPT for Event filter to test their applications → Decoupling HLT algorithm development and TDAQ data flow development



Offline support for developers

Emulators athenaMT/PT

Emulate complete L2PU/PT environment with specific boundary conditions for data access

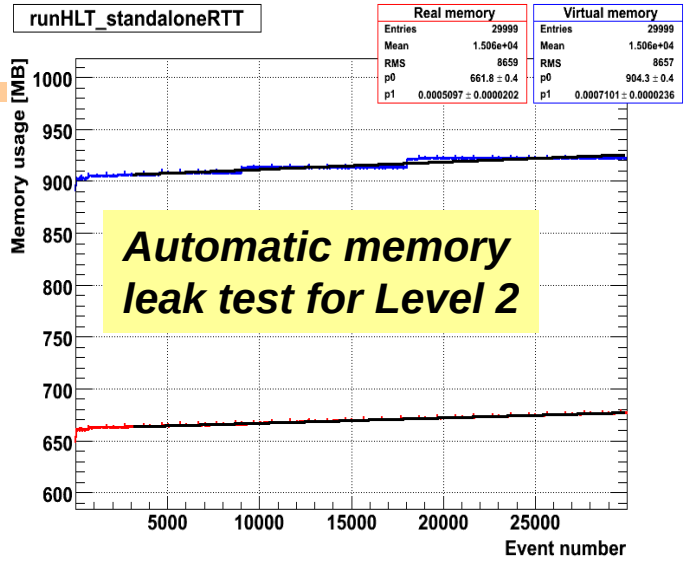
Command line tools → no need to setup complex Data Flow systems

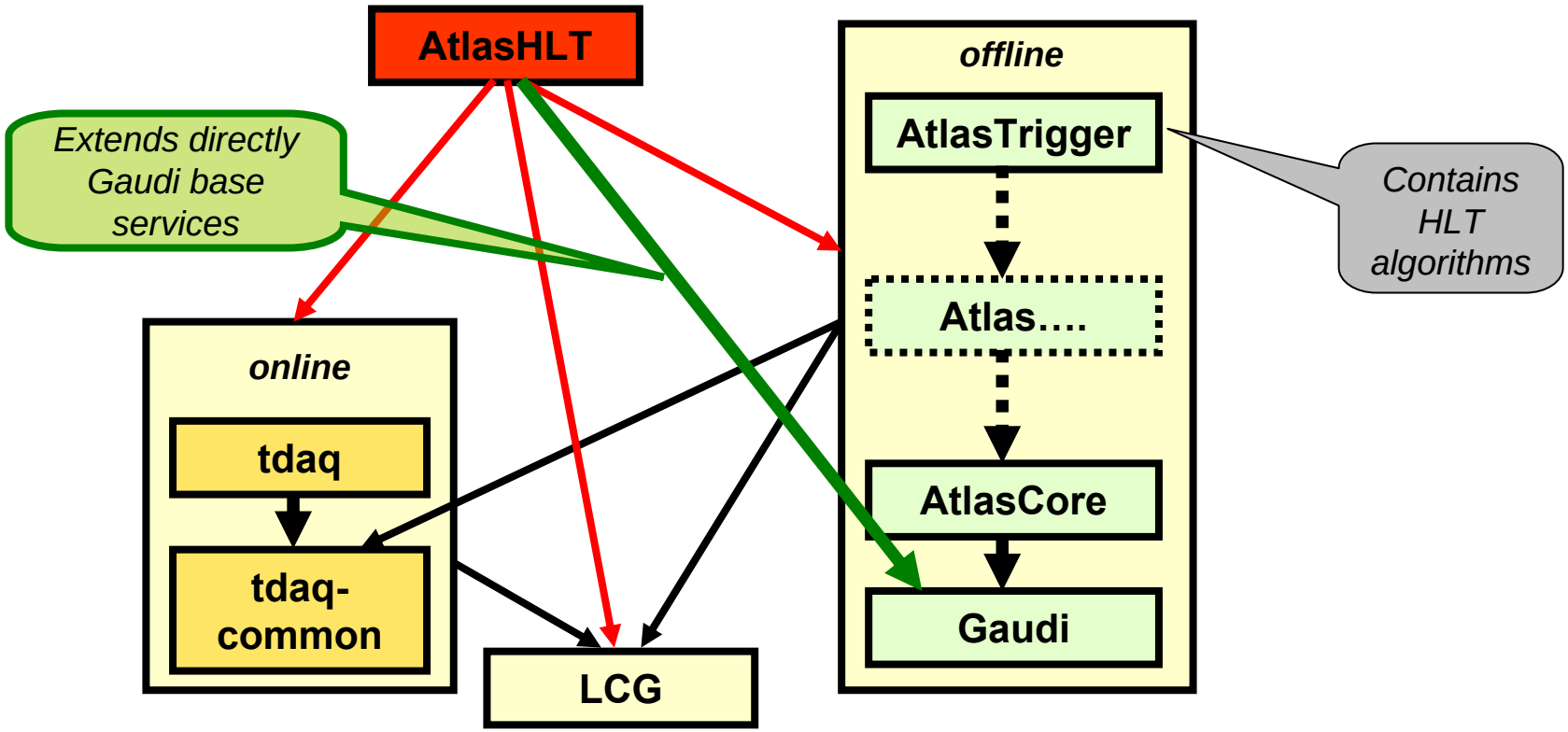
Allow testing of complete state machine

Written in Python/C++

The Atlas HLT Software Project

- The HLT framework packages form the AtlasHLT software project → setup like other offline software projects
  - uses many facilities from offline build and testing system
  - is hosted in the offline version control repository
  - Follows offline build and patching strategy
  - Automatic tests
  - …
- AtlasHLT project has dependencies on most of the Atlas software projects → AtlasHLT has to be last in build sequence

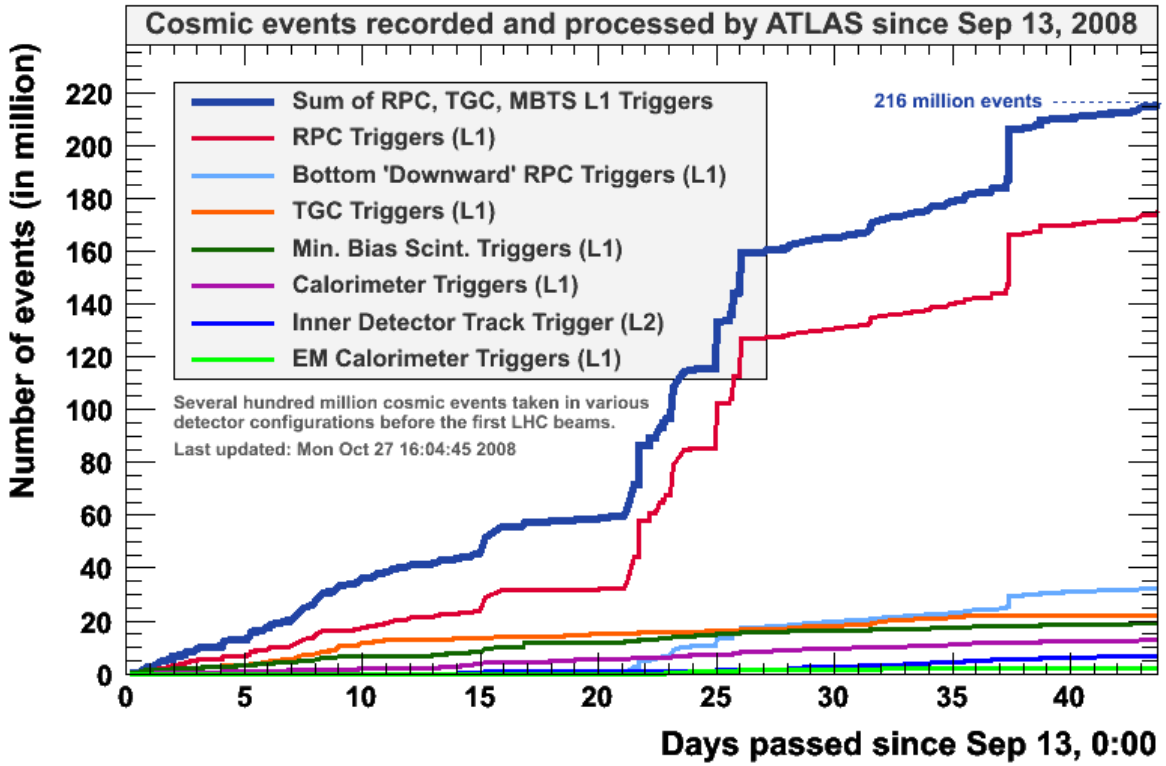




Operational Experience

- **The HLT framework has been successfully tested and used now over many years in e.g.**
  - **The ATLAS test beam in 2004 and TDAQ commissioning runs**
  - **In cosmic data taking and detector commissioning periods**
  - **1st beam on September 10th: configured in pass through mode**

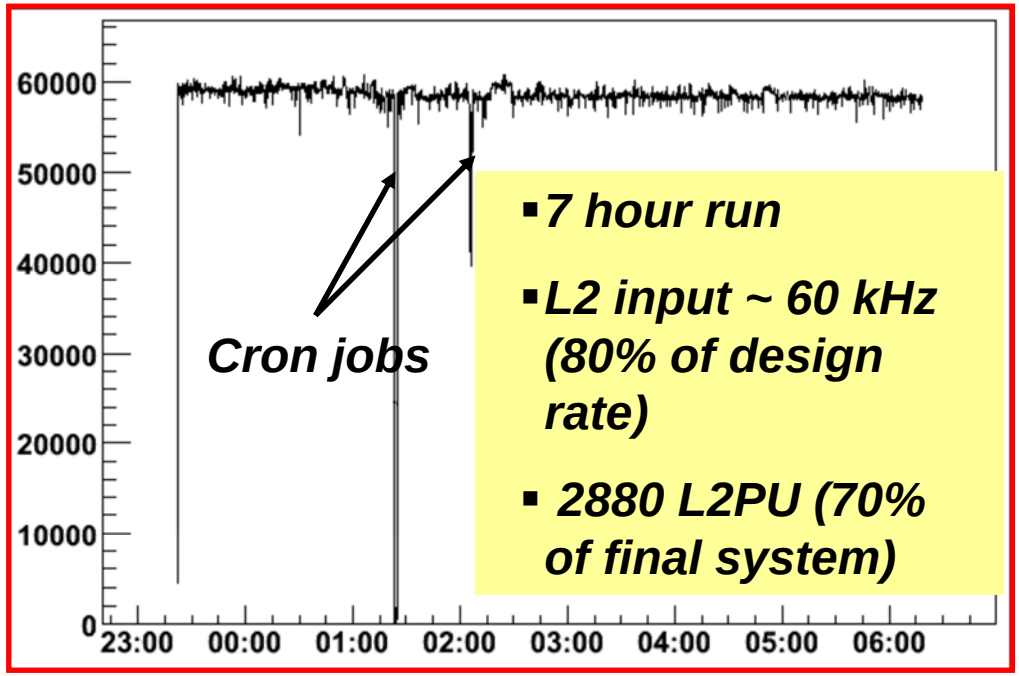
- *HLT framework used regularly for cosmic data taking and commissioning runs: e.g. continuous cosmic data taking after Sept .12:*
  - *216 millions events*
  - *453 TB data*
  - *400k files*
- *Proved to be very versatile and allowed to react fast on changing detector conditions*
- *Emulators athenaMT/PT used to analyze events offline in DEBUG streams and to eventually reassign them to the analysis streams.*
- *Valuable feedback on functionality, stability and reliability*

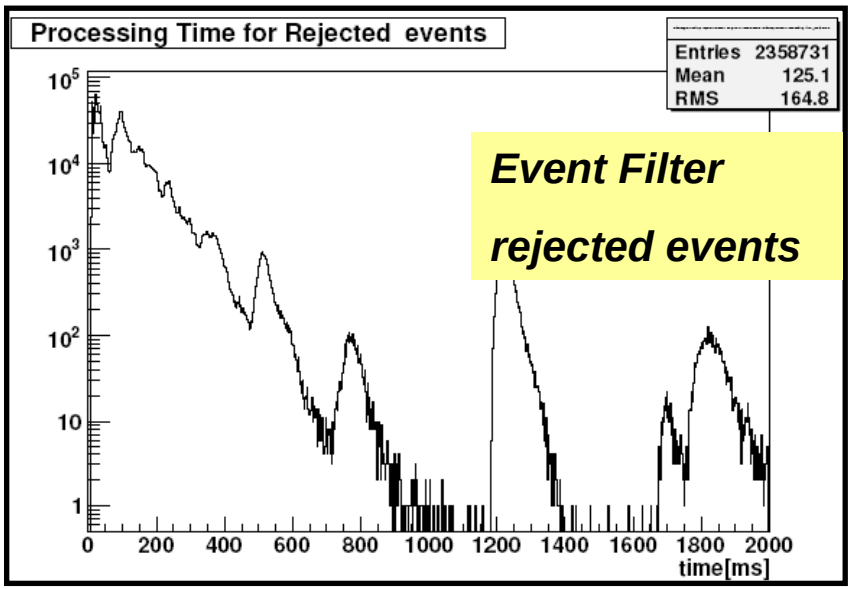


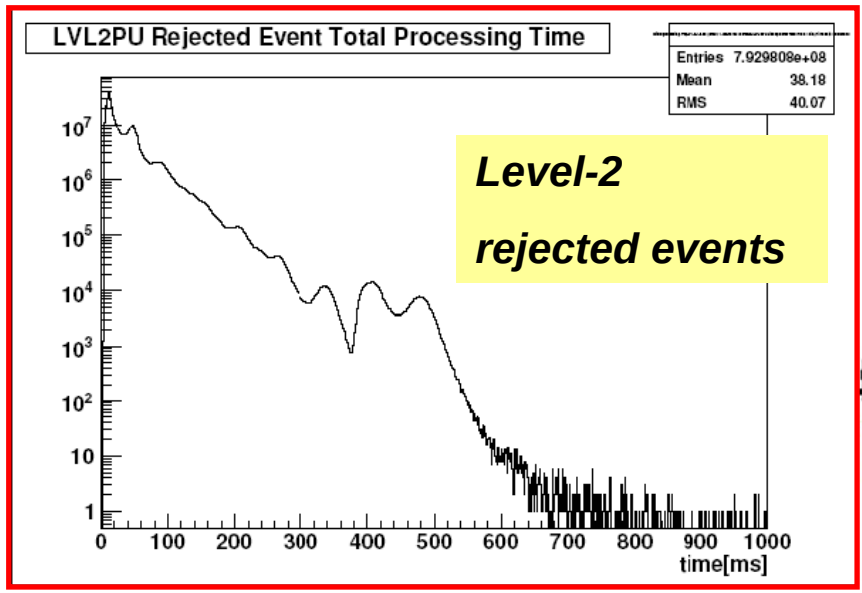
Operational Experience

System tests with full 10^{31} menu and simulated data showed that *timing for event processing is at specification* (2.5 GHz quad core CPUs)

- **Level-2: ~ 45 ms**
- **Event Filter: ~ 260 ms (Event Filter not exploited by this setup)**

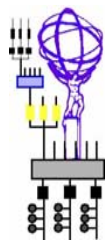




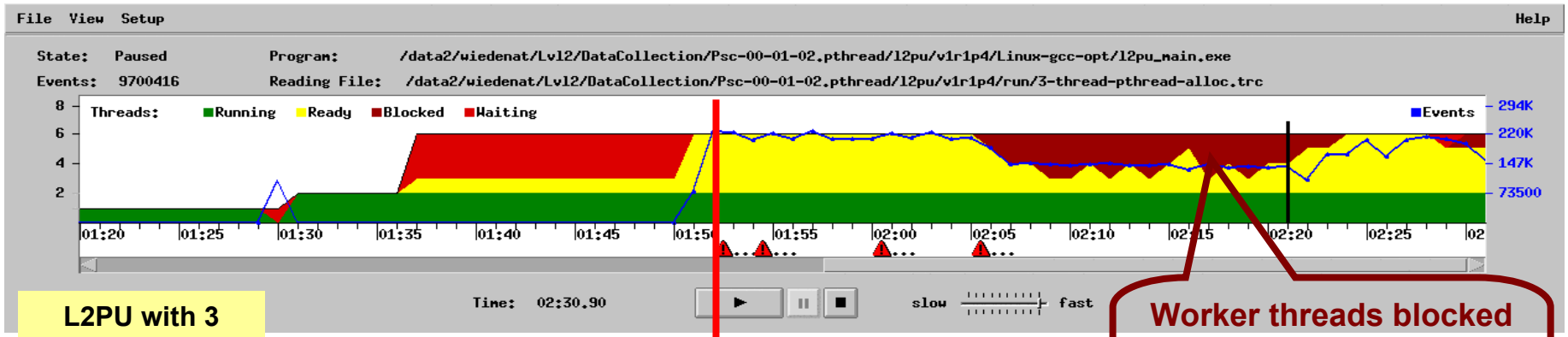


Event Parallelism and Multi-Core Machines

- Multi-core machines are used for the HLT processors
  - Processor development → rapidly increasing number of CPU cores
  - Calls for massive parallelism → typically inherent to high energy physics selection and reconstruction as "event parallelism"
- The original ATLAS HLT design exploits event parallelism in
  - Level-2: parallel event processing in multiple worker threads (with the possibility to fall back to multiple selection processes)
  - Event filter: multiple event processing tasks
- The HLT framework supports both ways of exploiting event parallelism
  - Special version of Gaudi/Athena framework to create selection algorithm instances for worker threads
  - Experience with multi threading in Level-2 has shown that it is very difficult to maintain a large code base in an open developer community thread safe and thread efficient.
- Use Level-2 processing unit with one worker thread with one Level-2 processing unit per CPU core
  - Allows also in Level-2 to profit from large ATLAS offline code basis (mostly written and designed in "pre-multi-core era")
  - Increases software communality between Level-2 and Event Filter and allows transparent move of Level-2 algorithms to Event Filter
  - Increases however the resource usage (memory, network connections, …)



Experience with Multi Threading in HLT Framework

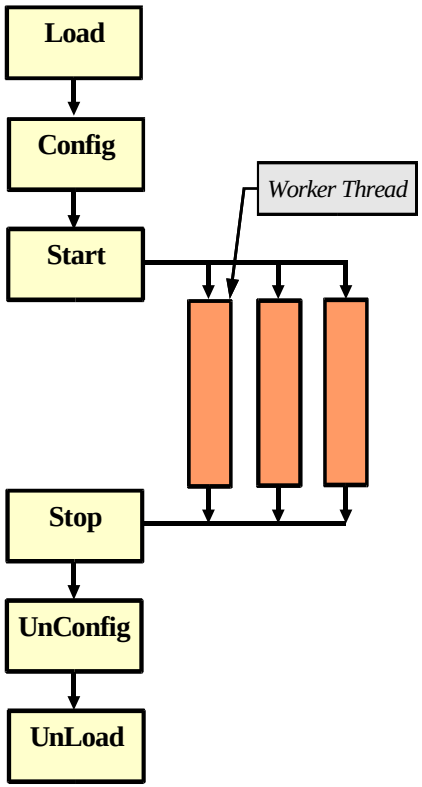


L2PU with 3 worker threads

Initialization

Event processing

Worker threads blocked due to common memory pool for STL containers



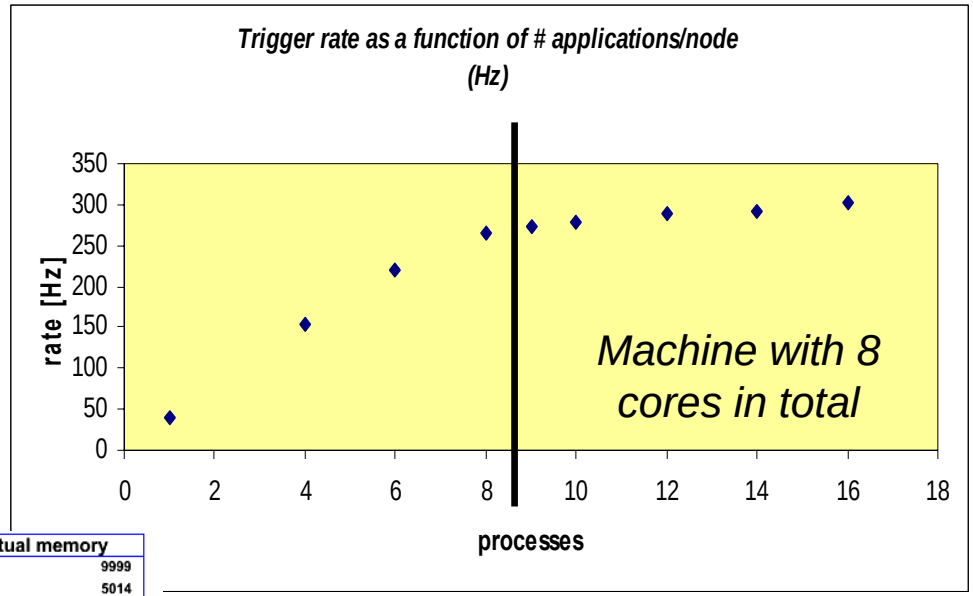
- *Event processing in multiple worker threads*
  - *Code sharing*
  - *Small context switch times → "lightweight processes"*
  - *Automatic sharing of many hardware resources*
- *Requires careful tuning and proves difficult to keep code thread safe and efficient over release cycles → impacts capability to react fast on changing conditions and to deploy new code*

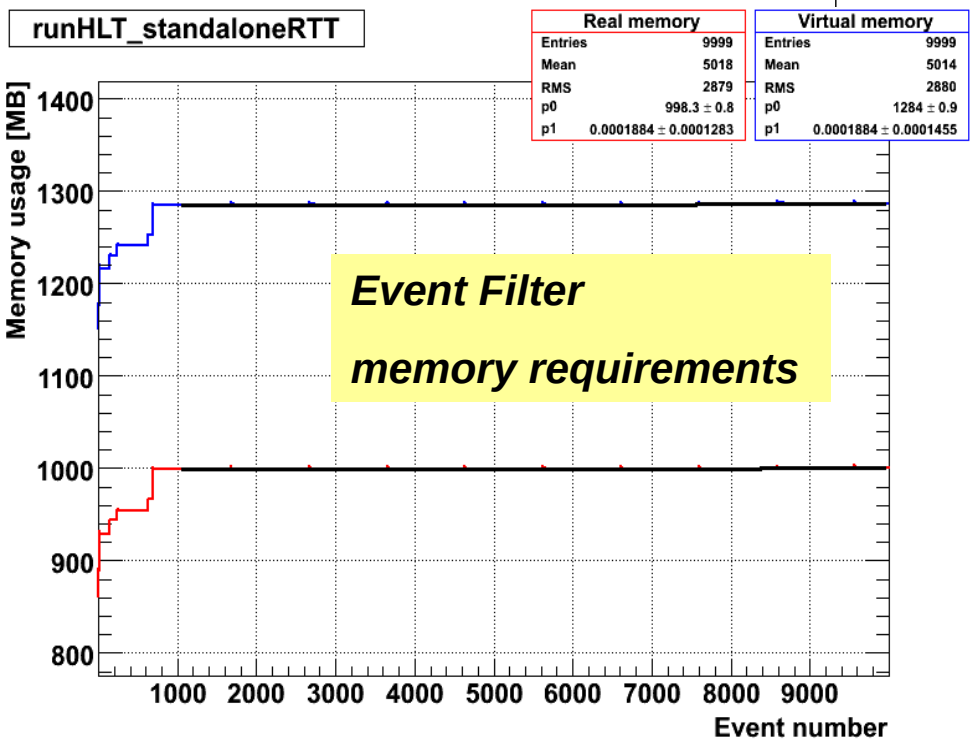
Experience with Multiple Processes in HLT Framework

Run on every CPU core one instance of a Level-2 processing unit or Event Filter processing task

- Easy to do with existing code basis → a priori no code changes required
- Facilitates reuse of offline components

Observe good scaling with number of cores → provides required throughput



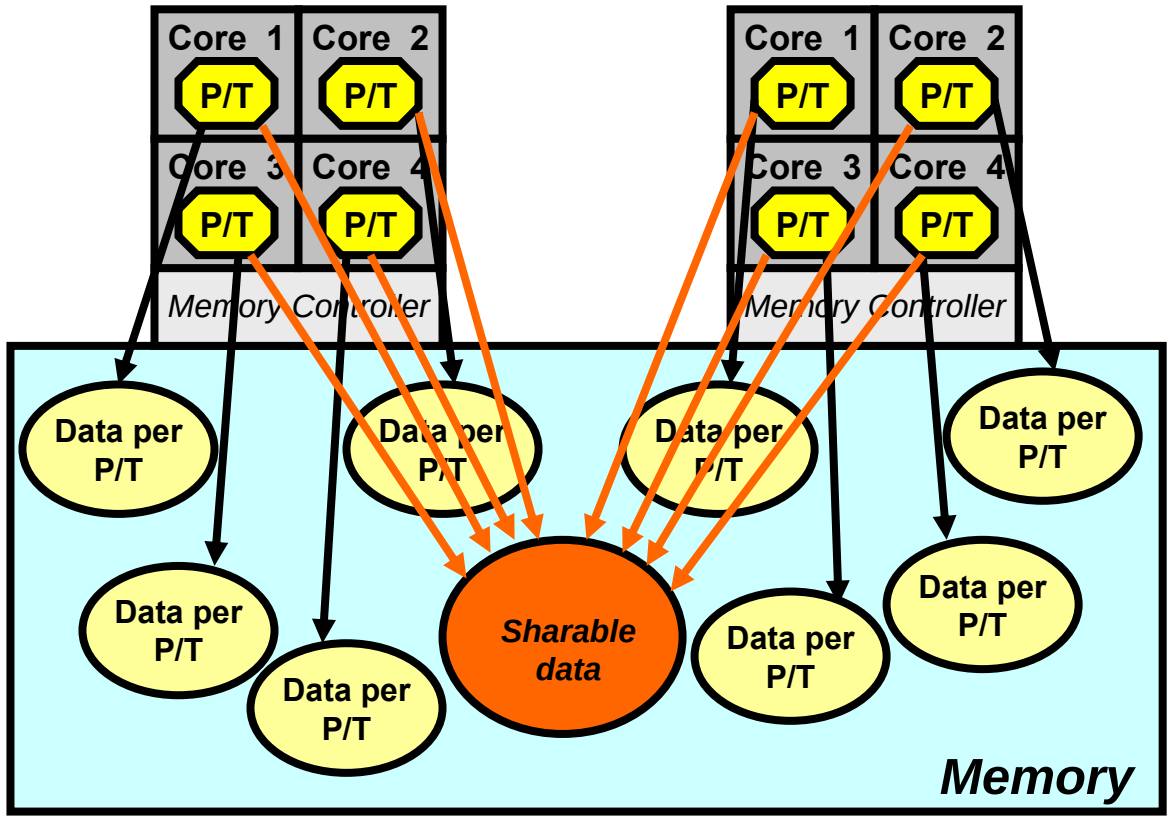


Resource requirements are multiplied with number of process instances

- **Memory**
  - *~ 1– 1.5 GByte/Application*
- **file descriptors,**
- **network sockets,**
- **number of controlled applications**
  - *~ 7k presently*
  - *~ 20k final system*

Resource Optimization

- Typically in HEP applications all processes use a large amount of constant configuration and detector description data.
- Common problem for offline reconstruction and trigger
- Different prototypes tried → see contr. # 244, S. Binet, *"Harnessing multicores: strategies and implementations in ATLAS"*
- Very interesting also "OS based" memory optimization "KSM"



Conclusions

- The *ATLAS HLT framework based on the ATHENA offline framework has successfully been used* in many different data taking periods and *provides the required performance*
- The reuse of offline software components allows to *benefit from a big developer community* and a large code basis.
- The ability to develop, optimize and test trigger algorithms in offline *is successfully used to establish the ATLAS HLT selection menu*
- The *optimal use of multi core processors requires further frame work enhancements* to reduce resource utilization. Many issues there are shared with offline and can profit from common solutions.
- The HLT Framework is ready for the new data taking periods.