The Evolution of the ATLAS Data Flow system for Run 2

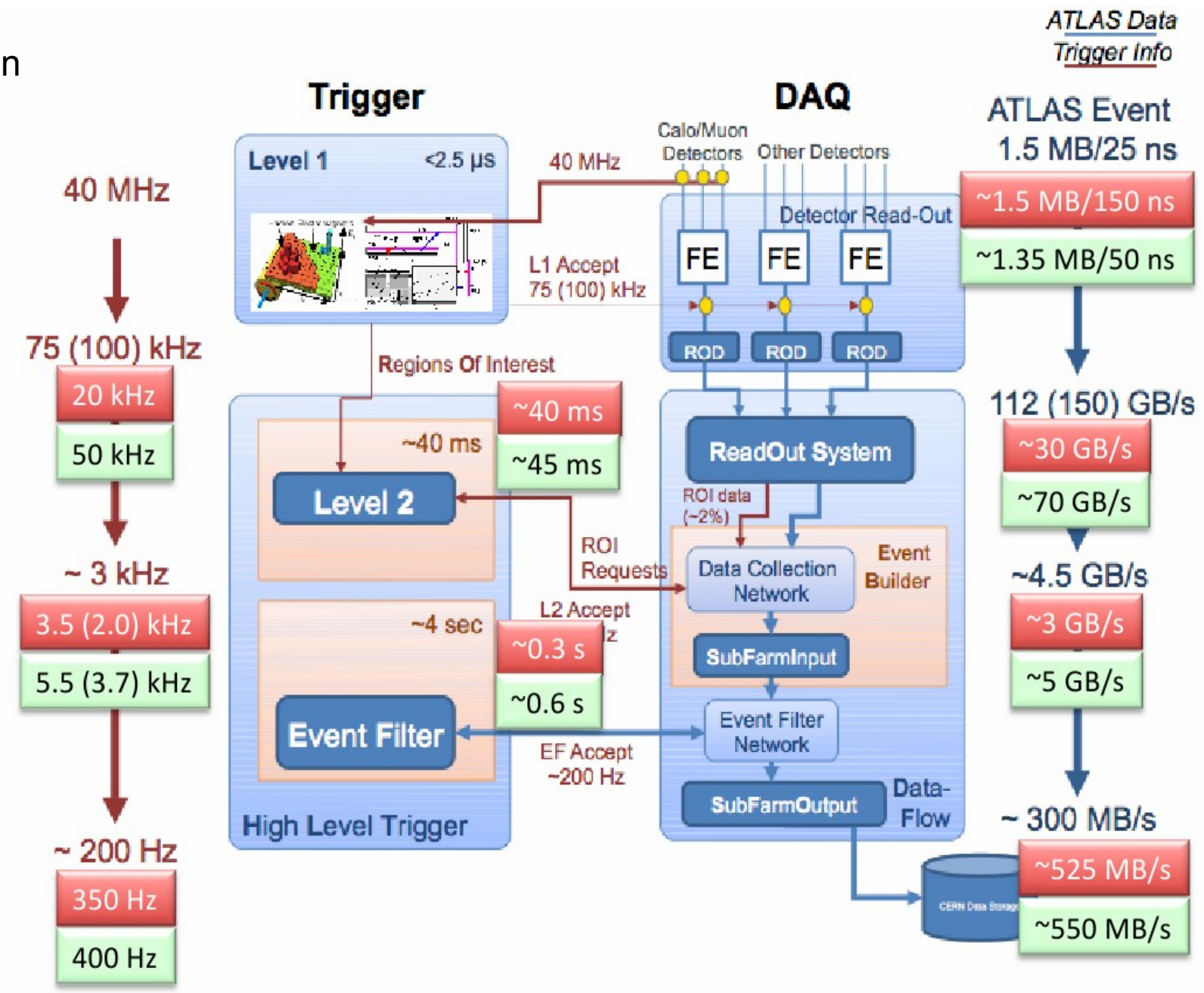
Reiner Hauser
Michigan State University

Outline

- Reminder: The ATLAS DAQ/HLT system in Run 1
- Reasons for Evolving the Data Flow Architecture
- Evolution of Components and their Interactions
  - New Data Flow Core for merged HLT
    - HLT Supervisor
    - Data Collection Manager
    - HLT Processing Unit
  - Network
  - Upgrades and evolution of existing components
    - Readout System
    - Region of Interest Builder
    - Data Logger

Reminder: The ATLAS Trigger/DAQ Architecture in Run 1

Design
2010
2012



Key Aspects of Run 1 Architecture

- Level 1 trigger forwards Regions of Interest (RoI) to Level 2 supervisor via the hardware based RoIBuilder.
  - 8 input links @ 75 kHz in Run 1 (→ 100 kHz in Run 2)
- Region of Interest based Level 2 trigger system
  - Requests data from readout system as needed
- Explicit event building step after Level 2 accepts event by dedicated machines.
- “Off-line” style processing of full event in Event Filter (EF)
- Accepted Events are send to data logger (SFO)

Why change things ?

- Two distinct networks for Level 2 and EF, but same technology in practice
  - Origin: Can one network technology handle both types of traffic ?
  - A subset of HLT nodes was connected to both switches in such a way that they could be moved from one to the other by a configuration change → required a stop of the on-going run.
- Flexible common communication library for all data exchange *except* for last step (EF to data logger) which used a different protocol for historical reasons → streamline software.
  - Originally to support a variety of possible transport mechanisms or technologies – in the end only TCP and UDP for multicast were used.

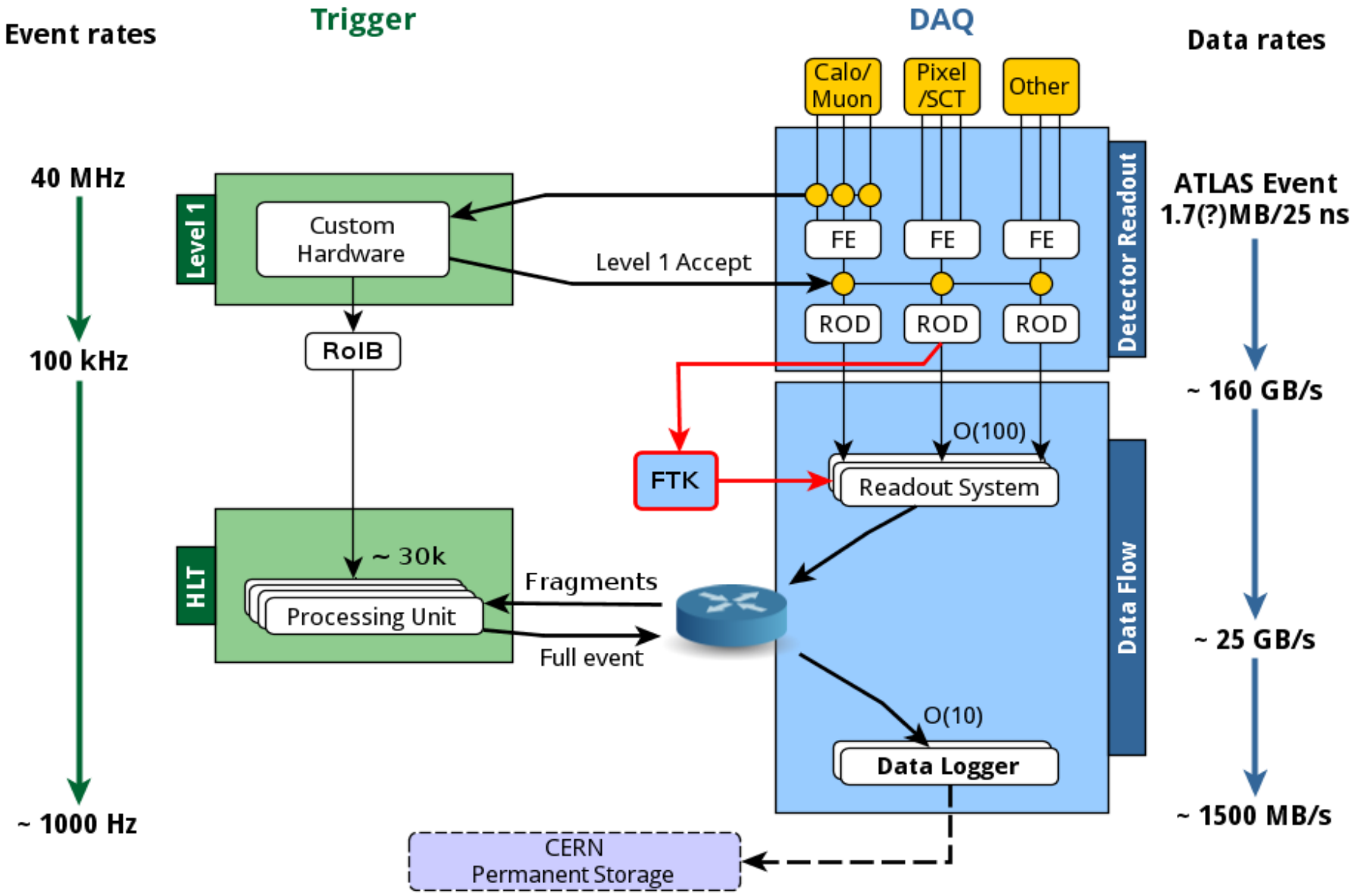
Why Change (2)

- Same software framework (Athena based on Gaudi) and trigger menu/steering was used for both Level 2 and EF, only difference was data access.
  - This decision was made ~2005, in the 199x's people still looked at hardware or hardware assisted solutions.
  - Quantities calculated in Level 2 had to be re-calculated in Event Filter.
- The appearance of multi/many-core rather than multi/many GHz CPUs made it necessary to run multiple instances of Level 2 and EF applications on a single machine.
  - Event filter had a separate process responsible for I/O, but in the Level 2 case the I/O was integrated into each application.
    - Every ROS host had ~6000 TCP connections to Level 2 applications.
  - The increasing number of cores per CPU would increase our memory requirements (2 GByte per core was our assumption for HLT applications)

The Run 2 Data Flow System Development Timeline

- Discussions started in **2009** about merging Level 2 and Event Filter for Run 2.
- Prototyping and design started in serious in **~2011**
  - We wanted to have a usable system by the ***start*** of LS1
  - We did not quite make it, but 2013/14 was spent on regular Technical Runs (~1 week long, every other month) where we used the full read out system and all of the HLT infrastructure for testing the new software.
  - Upgrades happened during that time to accommodate for the new architecture → Network
  - In addition other components that were assumed to be stable in terms of interfaces went through their own evolution.
    - → Read out system (2014)
    - → Region of Interest Builder (this last winter shutdown)

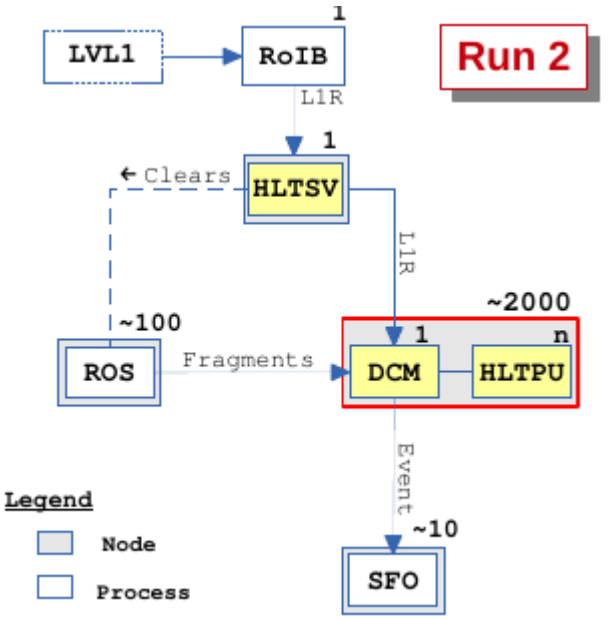
The ATLAS Trigger and DAQ System in Run 2



Components of the ATLAS Data Flow

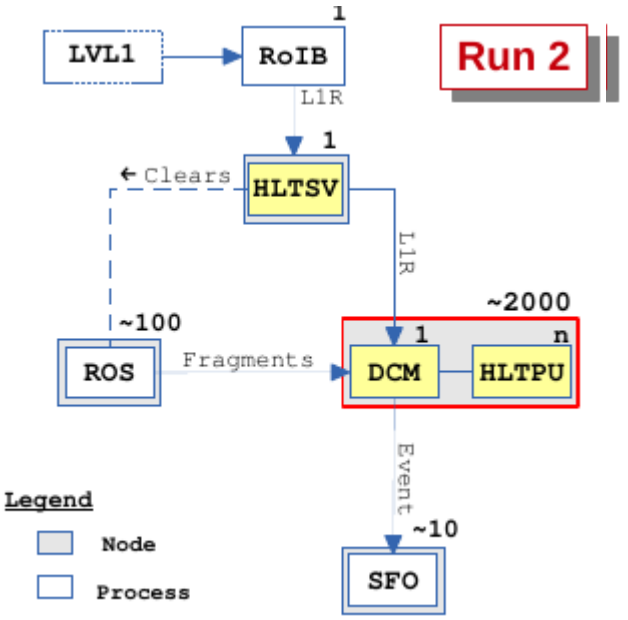
- For Run 2 the former Level 2 and Event Filter farms have been merged into a ***single HLT farm***.
  - A big simplification in terms of applications (almost a factor 2) and network architecture.
- The ***Region of Interest (RoI) concept has been kept***, and processing and data collection proceeds in stages, beginning with fast algorithms based on RoIs. The ***decision when to build the event can be taken on a per event basis.***
  - The **Readout System (ROS)** buffers front-end data from the detectors and provides a standard interface to the DAQ.
  - The **Region of Interest Builder (RoIB)** receives L1 trigger information and RoIs and combines this information for the HLT Supervisor.
  - The **HLT supervisor (HLTSV)** schedules events to the HLT farm and handles eventual time-outs.

A common message passing library based on Boost ASIO is used by the applications: TCP/UPD only, vastly simplified and dynamic configuration



Components of the ATLAS Data Flow (2)

- Components (cont'd)
  - The **Data Collection Manager (DCM)** handles all I/O on the HLT nodes, including RoI requests from the HLT and full event building.
  - The **HLT processing tasks** are forked from a single mother process to maximize memory sharing, and run the ATLAS Athena/Gaudi framework in a special online mode.
  - The **data loggers (SFO)** are responsible for saving accepted events to disk, and send the files to EOS.



The yellow boxes represent the core of the "new" architecture, the interfaces to the other parts were kept the same. We also assumed that we had to work with the same limits as in Run 1, at least in the initial stages.

HLT Supervisor (HLTSV)

- A ***single*** HLT supervisor replaces the set of (~6) L2 supervisors used in Run 1.
  - Run 1 system required manual load-balancing of the farm over the multiple L2 supervisors since the hardware RoIB was strictly round-robin.
  - A single application has the complete overview of the whole farm and simplifies scheduling, global event ID assignment, recovery etc.
- It uses a heavily multi-threaded design using the ***Boost ASIO*** library for communication and ***Intel Thread Building Blocks*** for concurrent data structures.
- ***2 x 10 Gbit/s Ethernet*** interfaces to the data flow network.
- Same hardware as ROS machines

HLT Supervisor (2)

- It handles input from the RoIB, assigns events to HLT nodes with free cores, handles timeouts and sends clear requests for events to all ROS machines using ***UDP multicast.***
- A single application can handle the input from the RoIB and manage the HLT farm of ***~1500 machines at ~115 kHz*** under realistic ATLAS conditions.
  - In pre-loaded mode (no S-Link input), and minimal data size the application can sustain more than ***250 kHz***. This gives an idea of the network and multi-threading capability of a current Linux system: about ***4 μs/event***, including sending and receiving at least one package, scheduling and handling timeouts for each connection etc.
- A lot of effort was spent to have the HLTSV run as a single application – the idea behind this was always to add the Region of Interest builder functionality to it in software later.

Data Collection Manager (DCM)

- The DCM is a ***single application per HLT node*** that deals with all I/O data requests from ***multiple HLT processing tasks*** on the same node.
  - It handles both Region of Interest request and full event building.
- It communicates to the HLT tasks via sockets and shared memory, using the Boost Interprocess library.
  - The shared memory is backed by a file, so that events can be recovered in case of crashes.
- Its design is essentially ***single-threaded*** based on non-blocking I/O using the Boost ASIO library.
  - A credit based traffic shaping mechanism is used to prevent overloading the incoming network link (into the node and the rack). Cost for a request is based on the number of read-out links that are requested.

Data Collection Manager (2)

- For accepted events the DCM also handles the preparation (duplication) for an event going to ***multiple output streams*** (e.g. for calibration purposes).
- Finally it ***compresses*** the event payload before sending it to the data logger.
  - These last two functions were originally foreseen in the data logger itself. Since there were only a few instances, it was considered a potential bottleneck.
  - The compression was in practice always done in Tier-0, never on the data logger.

HLT Processing Unit (HLTMPPU)

- The HLT processing unit ***encapsulates the Athena framework*** that is running the actual HLT algorithms.
- It communicates with the DCM for I/O requests and provides the trigger decision for each event.
- On each node a ***mother process is started first and goes through all the configuration***. A set of child processes is forked when the run starts, ***maximizing the memory sharing through the kernel's copy on write mechanism.***
  - → Crashed HLT applications can be quickly replaced by forking another child instance.
  - Tests with the full trigger menu show a memory consumption of ~ 1.8 GByte + N x 700 MByte (N = 8)

Stopping halfway...

- This was the first pass at the new architecture.
- The Readout System and the RoIBuilder were still the same.
- The connection to the data logger was still using the old protocol
- The data logger was still doing the streaming.
- We discussed a lot what was the best strategy to trigger the event building step (assuming the old memory limitations of the readout system)
  - Let DCM decide since it knows the distribution of read-out links to physical ROS hosts.
  - Let the HLT decide since it knows when an event is for sure (or even most likely) accepted.
- In the meantime other developments went on in parallel...

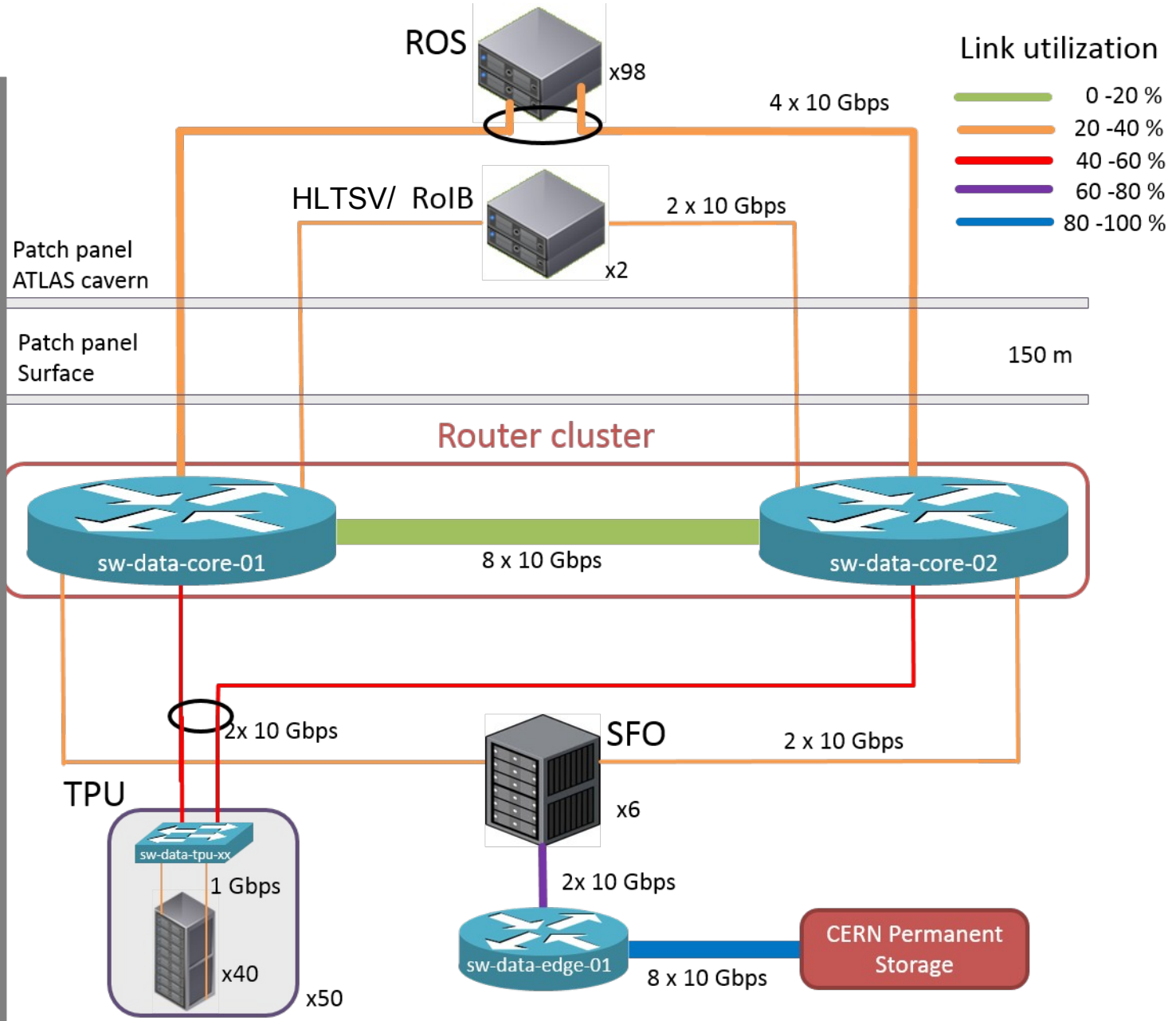
Data Flow Network

The network architecture was changed during the rolling replacement to match the new data flow.

All duplication here is for redundancy at every level in case of link/switch failures.

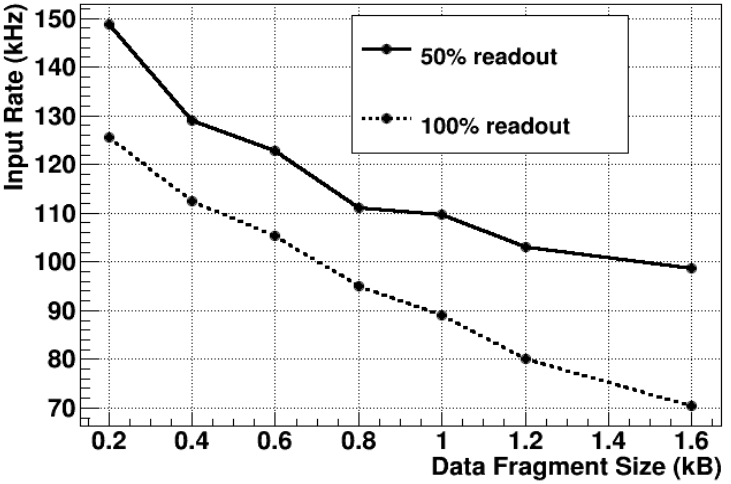
Functionality wise a single network is used for RoI based access, event building, and sending the data to the data logger (SFO).

Remaining main “issue” is ingress traffic into racks/nodes when requesting data.



Readout System (ROS)

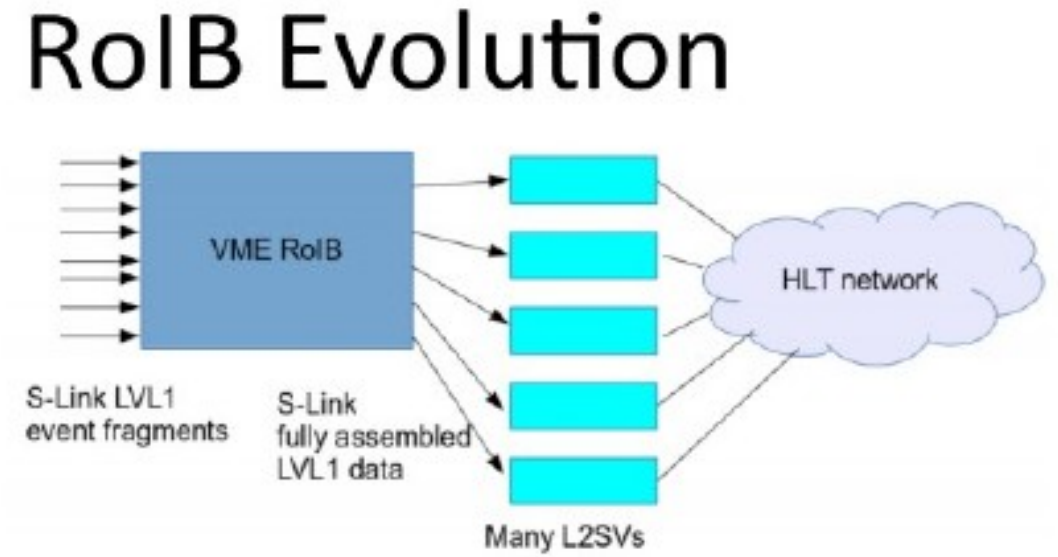
- The S-Link input and buffer hardware (Robin) has been upgraded to a new board (RobinNP) based on the Alice C-RORC card with ATLAS specific firmware.
  - Switch from PCI-X to PCI Express.
  - Higher density of optical link connectors:
    - 12 per card, 2 cards per PC.
  - **Larger memory buffer.**
  - New set of ROS PCs:
    - 2U form factor instead of 4U.
  - **4 x 10 Gbit/s Ethernet** per ROS PC.
    - (was 2x1 Gbit/s)



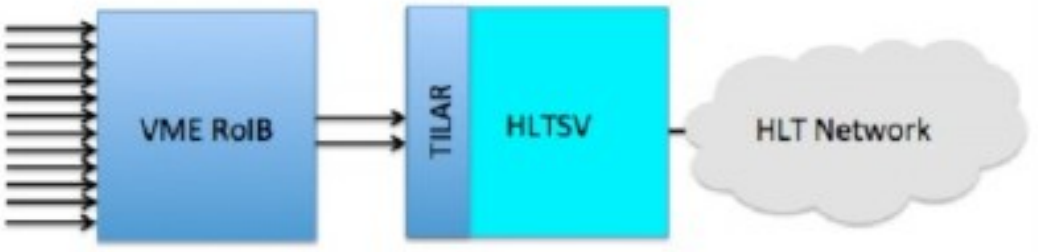
- A fully connected ROS (24 links) sustains the required RoIB request rate plus ~50 kHz of event building rate.
  - A ROS with fewer input links and/or small enough fragments can run at 100 kHz.

RoI Builder Evolution

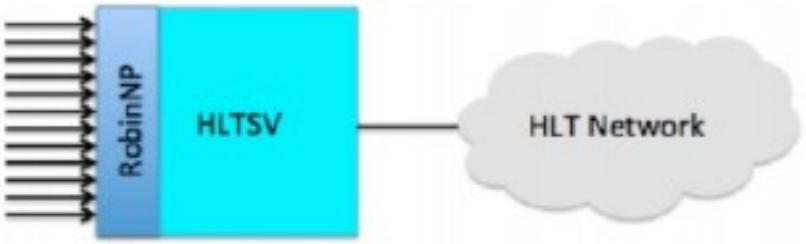
Run 1



2015



2016

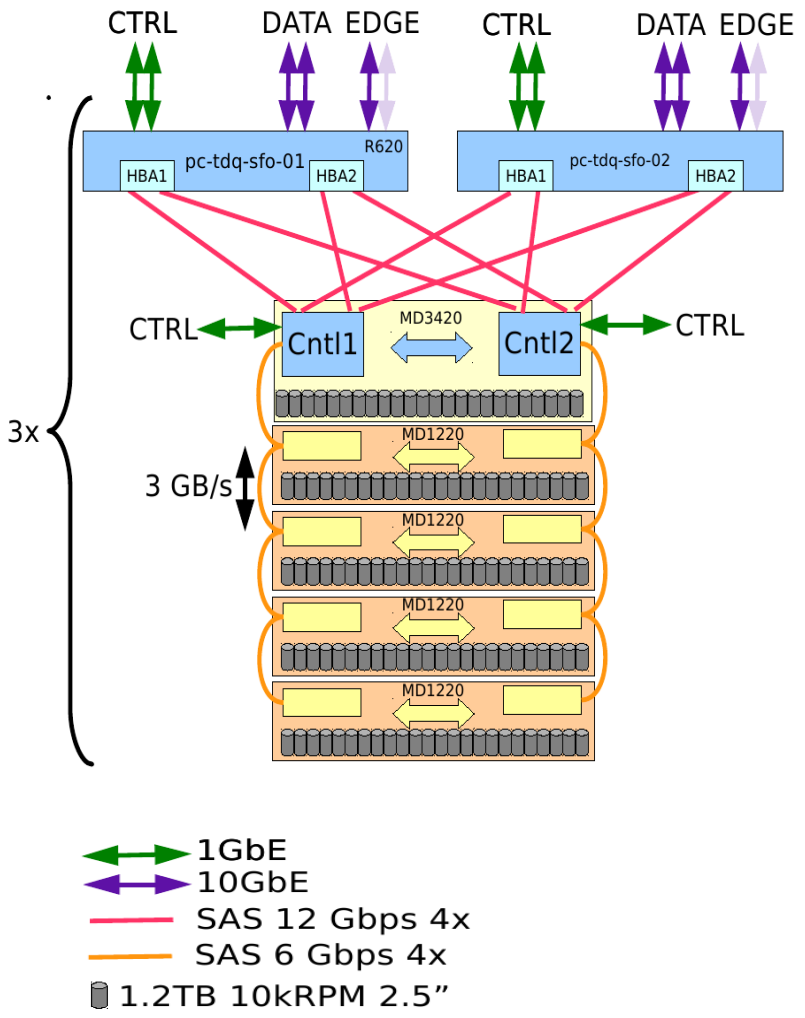


Region of Interest Builder (RoIB)

- The original RoIB is a ***9U VME based custom hardware solution***, consisting of multiple cards.
  - The original hardware from Run 1 was the baseline in 2015.
  - Larger input fragments for the Run 2 upgrade showed that the hardware is close to its limits.
- A replacement based on the C-RORC board was developed.
  - Common hardware between ROS and RoIB.
  - A single PCI Express board suffices for all 10 inputs.
  - The board directly integrates into the HLT supervisor.
  - The combining of the input fragments is done in software.
  - The new system was commissioned during this winter shutdown and has been in successful operation since then.
    - Vastly improved monitoring, flexibility regarding timeouts, error handling etc.
  - The final system can sustain a rate of 115 kHz under realistic ATLAS conditions.

Data Logger

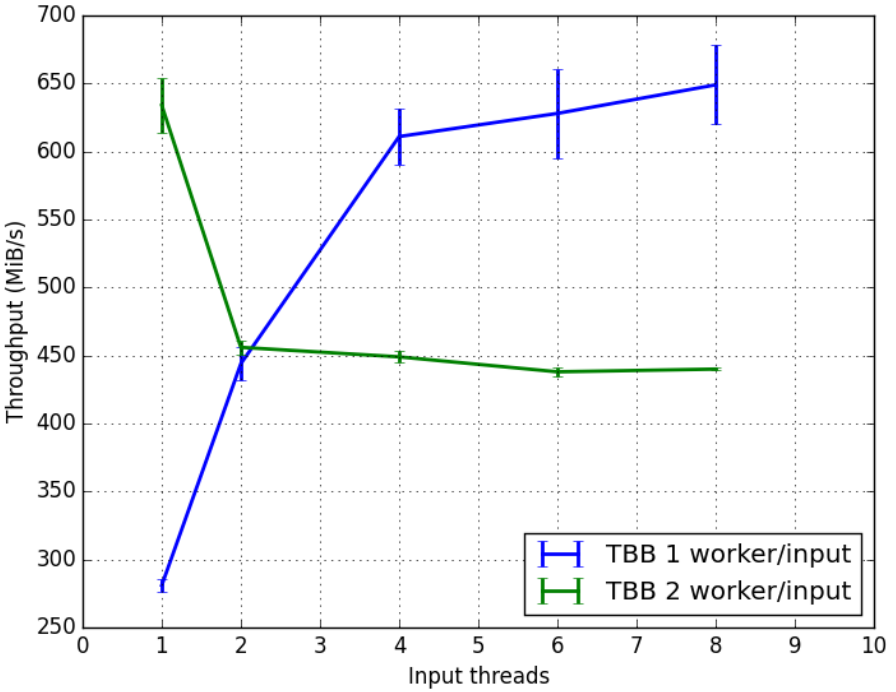
- In Run 1 a data logger was a PC with 3 internal Raid5 raid arrays of 8 disks each.
- For Run 2 a direct attached storage unit is used, with multiple front-ends and redundant data paths for fault tolerance and resilience.
- Experience so far show more than adequate performance.
  - Maximum bandwidth: >3 Gbyte/s
  - Operation point: ~2 Gbyte/s
- Background jobs copy the files to permanent storage, deleting them on the local disk only when they are safely on tape.

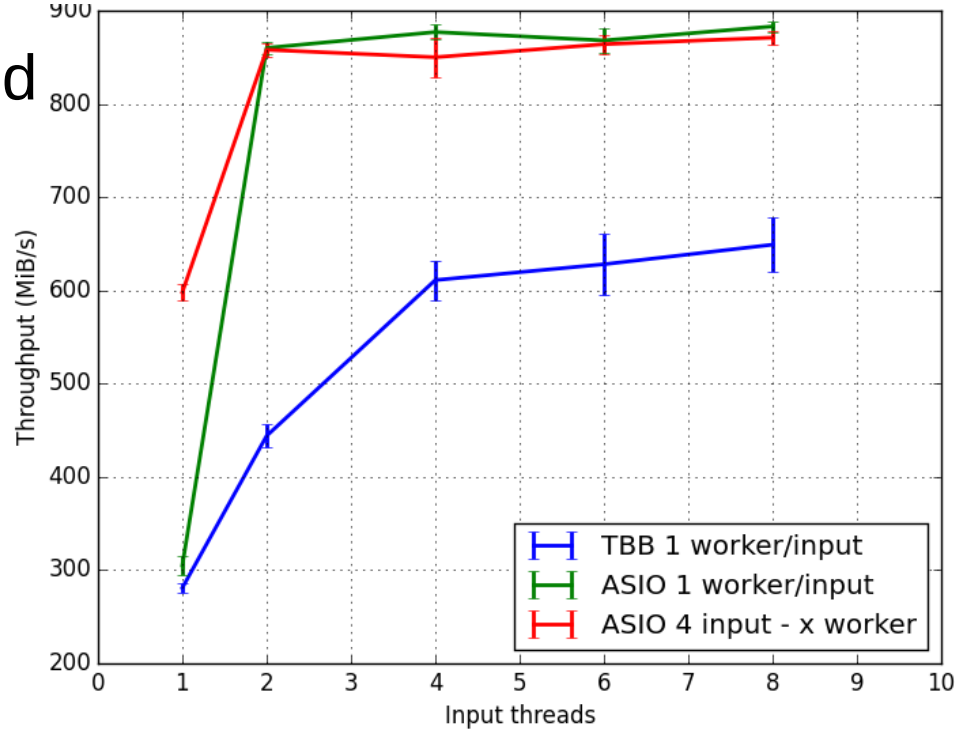


120 disks per node
340 TB total effectively for 3 systems.

Data Logger (2)

- The original data logger application of Run 1 was replaced with an Intel TBB task based version.
  - Input threads could handle 20 Gbit/s input
  - Hit limit at writing to disk at ~620 Mbyte/s due to contention in worke threads.
- Rewritten to use Boost ASIO based thread pool
  - 2 input + 2 worker threads → ~ 850 Mbyte/s





What has changed from our initial ideas ?

- The ROS buffers are much (much...) larger than before...
  - → We have given up discussing the best time to do event building: the HLT just continues to request data as needed. The full event is ***only*** built after the HLT processing is done and the event is accepted.
- Originally the DCM took responsibility of the event after building it, so it could be deleted from the readout system.
  - Recovering events from the shared memory file is a pain – it was implemented but never used in Run 1. It is also almost impossible to properly include them into the physics streams after the luminosity blocks are closed in the various databases.
  - → ***Leave event in the ROS and only clear it when the data logger has written it to disk (or thinks it has...)***
  - Eases DCM responsibilities, and we can avoid backing the shared memory by a file.

What has changed (2)

- The data logger was replaced during LS1 with a new highly multi-threaded implementation.
  - Before the start of Run 2 we replaced the old protocol with the common message passing library used for all other applications.
  - The streaming and compression functionality was moved from the data logger to the DCM applications (6 vs. 1500) → no longer a possible bottleneck.
- Out network went from one large single switched network to a routed network (e.g. 1 subnet per HLT rack). Only miminal software changes were needed (like making sure the UDP multicasts go beyond the routers).

What has changed (3)

- Remaining annoying issues are not completely under our control: e.g. ARP storms when system was inactive for certain time.
  - → This happened even during running at low event rate, so we implemented keep-alive messages on the application level to keep the ARP entries for the TCP connections “hot”.
- Our copy-on-write idea for the HLT was a big success: **except** for the unconfigure step, where all the allocated memory was properly deallocated, causing massive unsharing and hitting the local rack NFS servers hard.
  - → We omit the last finalize() step in Athena and just kill the child processes…
  - Whoever wants to write something to log files at this point, has to do it in a HLT specific finalize step.

Summary

- The ATLAS data flow has seen a considerable simplification compared to Run 1.
  - Both in the amount of software and the network resources (links and switches).
- After LS1, every component has been either upgraded hardware wise, or rewritten to take advantage of better design choices.
  - We had a working system at every intermediate step
- The new data flow system was in place beginning of 2014 and has been used for all commissioning, integration, cosmic runs, splash events and collisions taken by ATLAS since then.