The ATLAS High Level Trigger

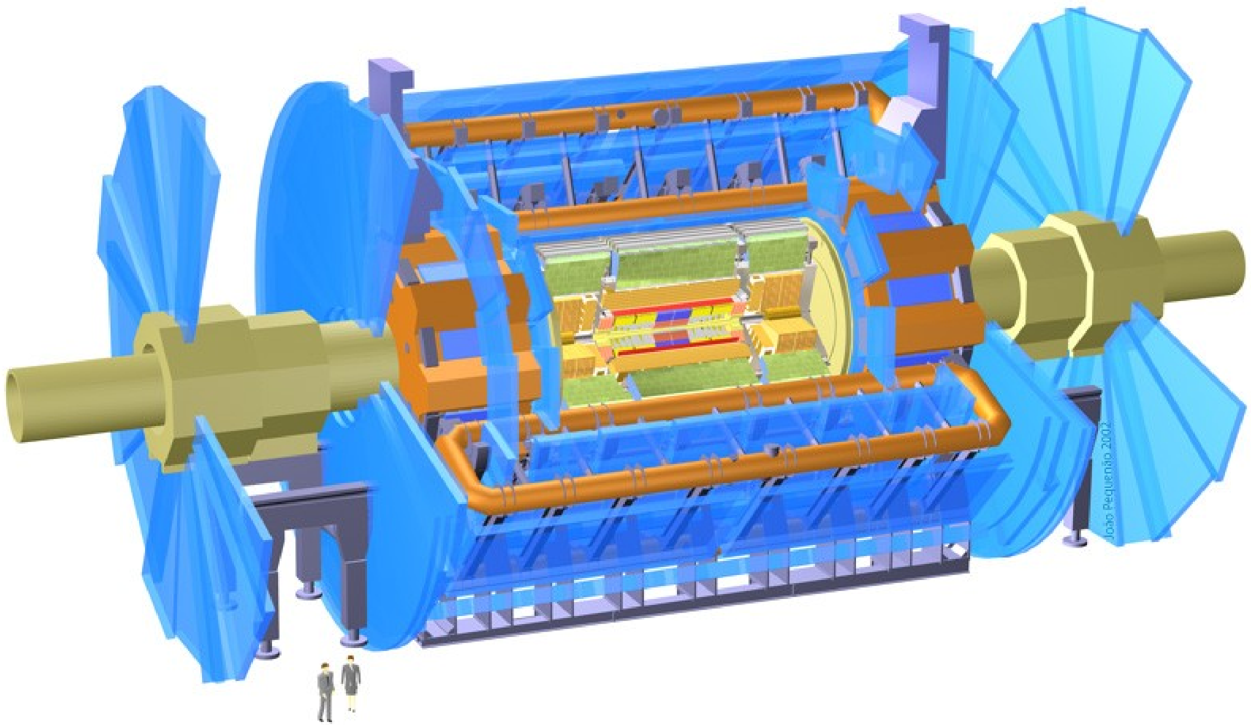
Infrastructure, Performance and Future Developments

Frank Winklmeier (CERN)
on behalf of the ALTAS Collaboration

IEEE 2009 Real Time Conference
IHEP Beijing, 10-15 May 2009

The ATLAS Detector

proton-proton collisions at 14 TeV with a bunch crossing rate of 40 MHz

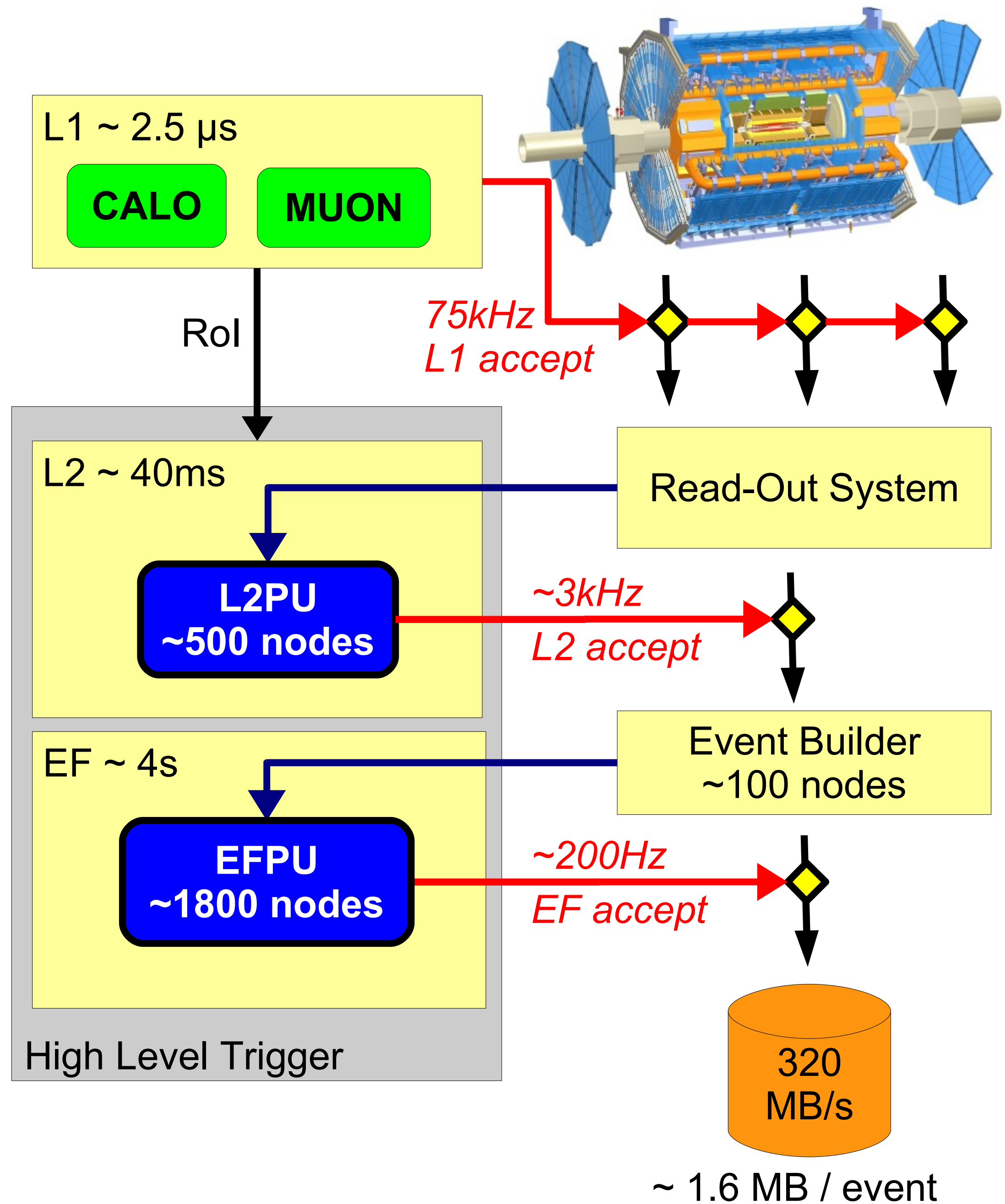


Other talks related to ATLAS Trigger/DAQ

HLT-4, C. Padilla, Commissioning of the ATLAS High Level Trigger with Single-Beam and Cosmic Rays
TDA2-4, J. Zhang, Atlas DAQ and Controls
TDA3-2, C. Padilla, The ATLAS Trigger System
TDA4-6, D. della Volpe, The ATLAS DataFlow System: Present Implementation, Performance and Future Evolution

ATLAS Trigger and DAQ System

- Level-1
  - Analyzes coarse granularity data from CALO and MUON detectors
  - Identifies Region of Interest (RoI) used to seed Level-2
- Level-2 (L2)
  - Partial event reconstruction in RoI
  - Event fragments requested from Read-Out System
  - Algorithms optimized for fast rejection running in L2 Processing Unit (L2PU)
- Event Filter (EF)
  - Full event reconstruction seeded by L2
  - Full event provided by Event Builder
  - Offline-type algorithms running in EFPU

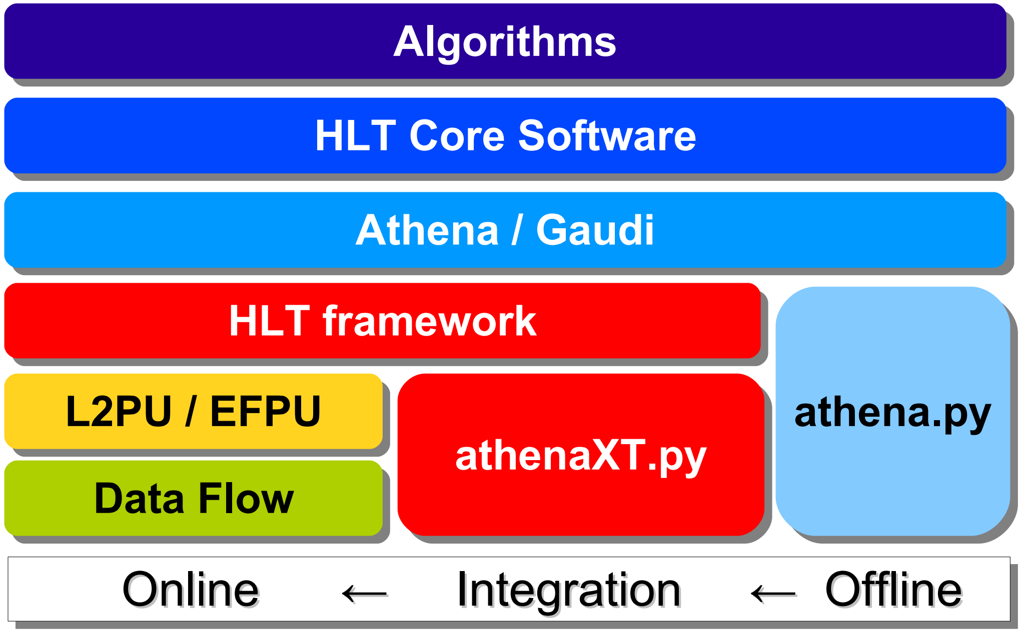


High Level Trigger Hardware

- Final system:
  - 17 Level-2 racks
  - 62 Event Filter racks
  - 1 rack = 31 PCs (1U)
  - → ~2300 HLT worker nodes
  - 28 racks configurable as L2 or EF (XPU)
  - 1 Local File Server (LFS) per rack served by Central File Servers (CFS)
  - Network booted
- Current system:
  - 27 racks installed
  - ~35% of final system
  - 2 x 4-core Harpertown @ 2.5 GHz
  - 2 GB Memory/core (16 GB / node)
- OS
  - Running Scientific Linux 4 (soon 5)

High Level Trigger Software

- HLT is based on offline event processing framework (Athena/Gaudi)
  - Component based software framework (co-developed with LHCb)
  - Abstract interfaces allow for transparent replacement of components
  - Python interface for job configuration
  - Whenever possible core-offline components are reused in HLT
  - Allows trigger algorithm development using offline environment and tools
  - Hundreds of developers with O(1M) lines of code

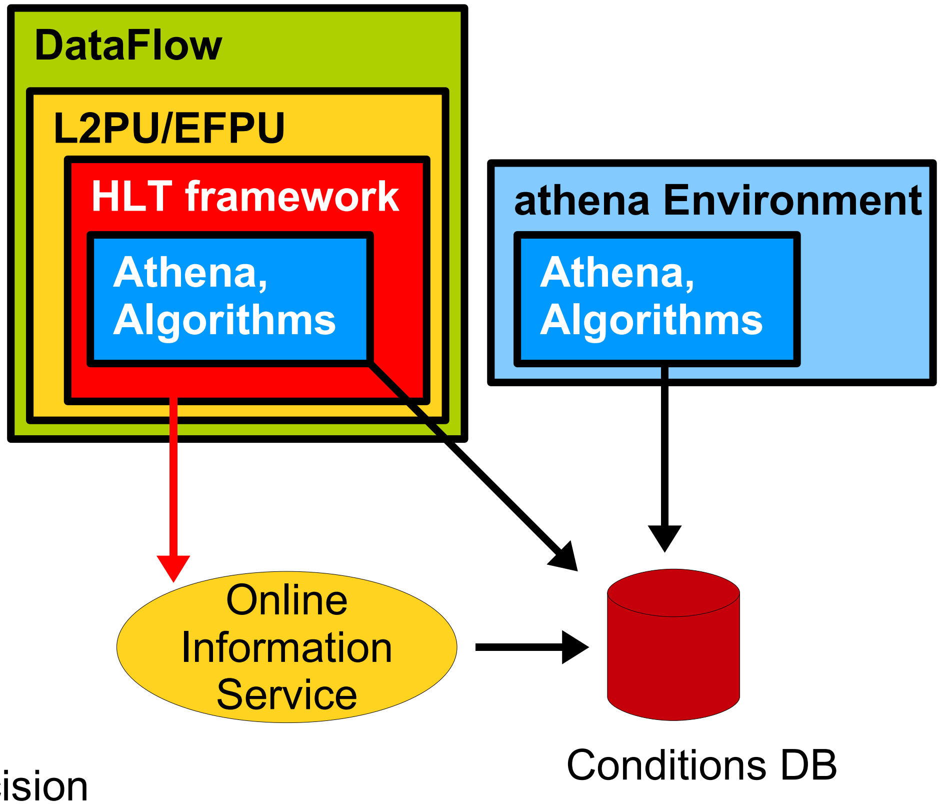


HLT framework

- Decoupling of DataFlow and HLT software development
- Online emulator (athenaXT) allows offline running of HLT software
- Facilitates testing and development

HLT Framework

- Provides special Athena services with “online backend”
  - Job configuration (from DB vs. python)
  - Messaging (interface to online messaging)
  - Histogramming (publish to online system)



- Interface to online infrastructure
  - Run and process control, state machine
  - Online configurations DB
    - Provide access to Readout configuration
  - Access to online conditions
    - Detector Mask
    - Magnet configuration

- Event handling
  - Assign stream tag information based on L2/EF decision
  - Special handling of calibration events
  - Forwarding of error conditions to other online applications

Online/Offline software requirements

- HLT software requirements
  - Often stricter and sometimes contradictory to offline software requirements

	Offline	Online (L2)
Initialization	on-demand (lazy)	before the 1st event
Job lifetime	O(1k) events	O(1M) events
Memory usage	Total < 2GB	+ Leak < 10 bytes/event

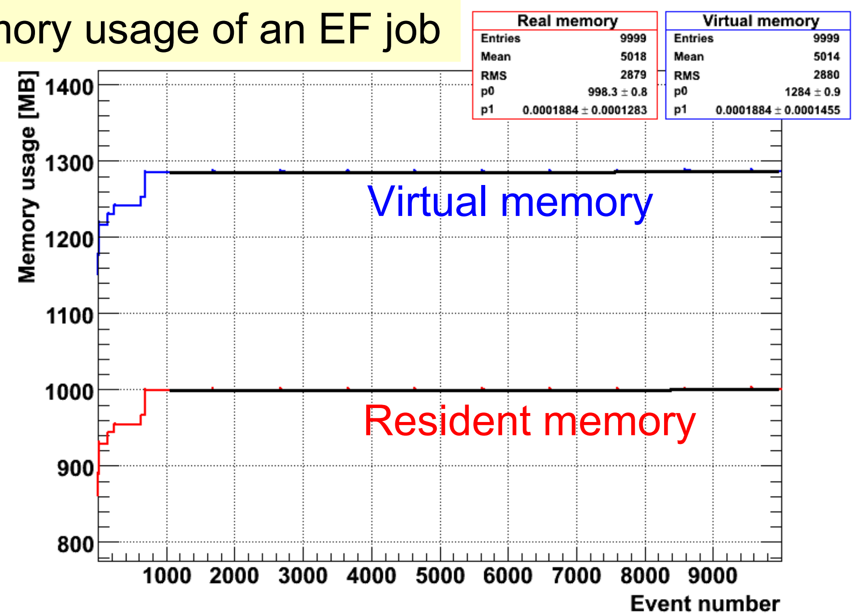
- Job initialization
  - Initializations have to be done before the first event (otherwise danger of timeout)
  - Developed tool based on valgrind to compare code execution profiles of 1st and Nth event

	event #1	event #2
`do_lookup_x`	14795	89
`CaloTowerStore::buildLookUp(CaloTowerContainer*,`	**2**	**0**
`CaloTowerContainer::getTowerIndices(double, doub`	**898243**	**0**
`TClassEdit::CleanType(char const*, int, char const**)`	6381	0
`P4EEtaPhiMBase::hlv() con`	4844	4844
`operator new(unsigned)`	**2080837**	**445405**
`LArRodBlockPhysicsV0::getNextEnergy(int&, int&, int&`	194532	0
`_int_free`	1264564	348732
`PoolSvc::testDictionary(std::string const&) const`	2	0
`ServiceManager::getService(std::string const&, IService*&`	1489	351

Performance monitoring

- Memory usage
  - A 10kB per-event leak at L2 will grow to ~900MB in one hour (@ 25 Hz)
  - Using public (valgrind, memprof) and custom leak checkers
  - Monitor memory usage for every nightly build
  - Need to make sure all code paths are executed → use ttbar or black-hole events

Memory usage of an EF job



Summary of Trigger PerfMon RTT results

The tuple in each cell is the result of a linear fit to the vmem/event graph, i.e. the first number is the initial **virtual memory** consumption and the second number the memory increase per event (in kB or MB). **Left-click on the links for more options.** See the legend for an explanation of the colors and symbols.

Links: offline(TriggerTest) offline(TrigAnalysisTest) online(HLTTesting)

Offline validation tests

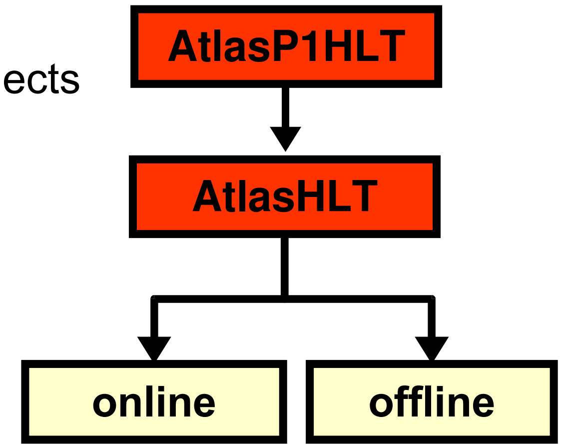
TriggerTest

Show builds: ☑ dev ☐ dewal

Test	Build	rel_0	rel_1	rel_2	rel_3	rel_4	rel_5	rel_6
Combined test - black-hole events	dev	x	x	x	x	x	x	x
Combined test - default menu	dev	x	x	x	x	x	x	x
Combined test - mem.leak check	dev	x	x	1734M/5.7k	x	x	x	x
Combined test - with default lumi1E31 menu	dev	x	x	1439M/0B	1437M/0B	1438M/0B	...	x
ElectronSliceAthenaModernRDO - single e	dev	x	x	x	x	x	x	x
LVL1 only test - default menu	dev	ok	ok	ok	ok	ok	...	ok
Physics lumi1E31 menu	dev	x	x	1374M/6.1k	1374M/7.2k	1319M/10.1k	...	x
Physics lumi1E31 no prescales	dev	x	x	1425M/33.3k	1400M/14.4k	1354M/8.8k	...	x
caching test - calo	dev	x	x	1105M/10.6k	1106M/5.7k	1106M/9.6k	...	x
lumi1E31 menu - no prescales	dev	x	x	1509M/42.3k	1510M/35.2k	1511M/33.4k	...	x
testBjetSliceAthenaModernRDO	dev	1124M/28.0k	1125M/26.7k	1058M/25.1k	1056M/29.3k	1055M/30.3k	...	1125M/25.1k
testBphysicsSliceAthenaModernRDO	dev	1251M/20.2k	1252M/16.8k	1177M/27.4k	1178M/23.2k	1179M/23.7k	...	1252M/19.9k
testBphysicsSliceAthenaModernRDO_MuGirl	dev	1238M/25.8k	1239M/22.1k	1162M/27.8k	1162M/25.7k	1161M/28.4k	...	1237M/25.1k
testBphysicsSliceAthenaModernRDO_noMuon	dev	1209M/57.4k	1210M/56.9k	1147M/54.7k	1146M/53.7k	1146M/59.3k	...	1211M/54.6k
testElectronSliceAthenaModernRDO	dev	562M/0.1k	x	x	x	x	x	562M/0.1k
testJetSliceAthenaModernRDO	dev	948M/9.5k	946M/8.7k	948M/9.5k	946M/10.4k	949M/9.5k	...	946M/10.7k
testMETSliceAthenaModernRDO	dev	1187M/16.1k	1186M/20.7k	1120M/22.7k	1121M/18.1k	1120M/20.4k	...	1188M/16.7k
testMinBiasSliceAthenaModernRDO	dev	1053M/5.6k	1054M/5.0k	985M/4.8k	983M/6.2k	986M/0B	985M/4.8k	1049M/4.4k
testMuonSliceAthenaModernRDO	dev	x	x	x	x	x	...	1191M/29.1k
testPhotonSliceAthenaModernRDO	dev	x	x	1115M/13.6k	1113M/16.1k	1116M/10.2k	...	x
testTauSliceAthenaModernRDO	dev	1219M/12.4k	1215M/21.0k	1154M/10.5k	1152M/12.4k	1153M/17.8k	...	1219M/17.6k

Release building and validation

- AtlasHLT software project
  - Software project depending on both the online and offline software projects
  - Special patching project to allow for fast patching within O(hour)



- Release validation
  - Functional and regression nightly tests
  - Testing both algorithms and infrastructure using online emulators
  - Running “localhost HLT partition”

Trigger ATN test results summary

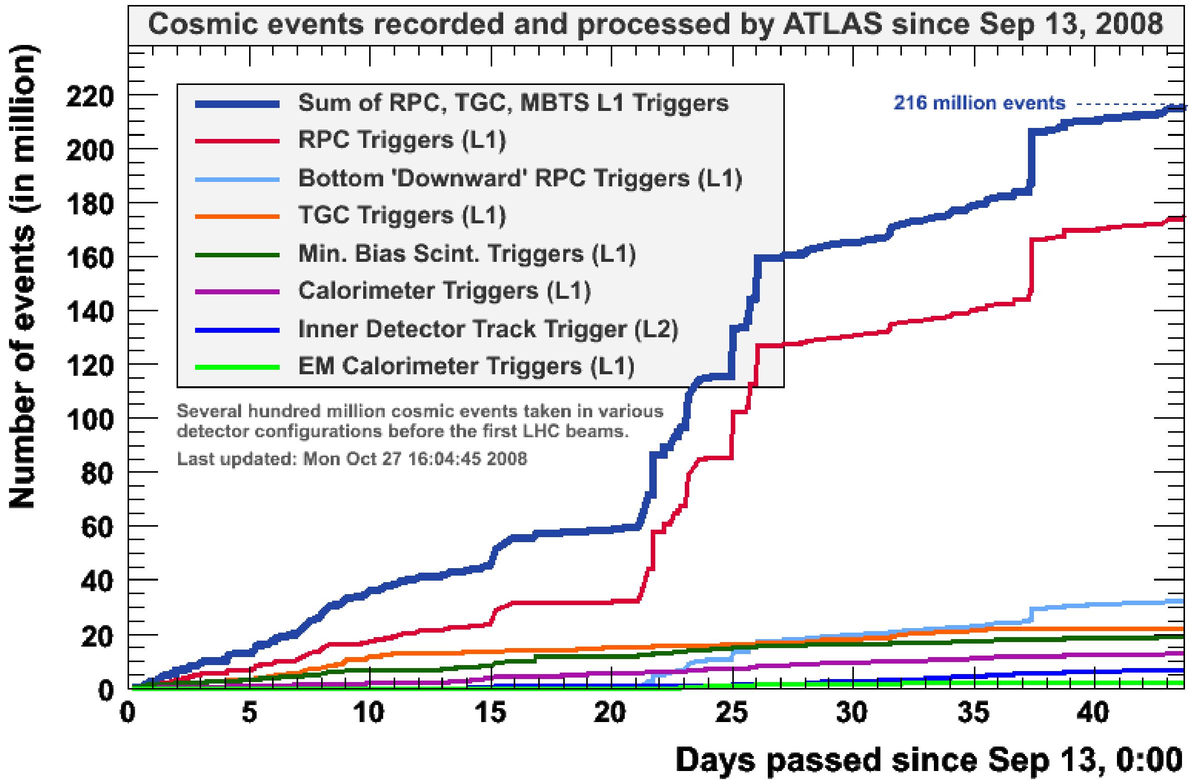
Nightly test: 1422XYP1HLT32BS4P1HLTOpt **rel_4**

Other nightlies: 0 1 2 3 4 5 6

Test name	Test script	Athena exit	Error Msgs	Reg. tests	Rootcomp	Exit code	Post cmd	Dir. link	Log link
HelloWorldMT_run-stop-run	-	OK	OK	N/A	N/A	0	N/A	dir	MTHelloWorldOptions_tail.log MTHelloWorldOptions_test.log
HelloWorldPT_run-stop-run	-	OK	OK	N/A	N/A	0	N/A	dir	MTHelloWorldOptions_tail.log MTHelloWorldOptions_test.log
MTMonHistOH	-	OK	OK	N/A	N/A	0	N/A	dir	MTMonHist_tail.log MTMonHist_test.log
testAllAthena	-	OK	OK	OK	MISMATCH [ps]	4	N/A	dir	runHLT_standalone_tail.log runHLT_standalone_test.log
testAllMT	-	OK	OK	OK	MATCH	0	N/A	dir	runHLT_standalone_tail.log runHLT_standalone_test.log
testAllMT_run-stop-run	-	OK	OK	N/A	N/A	0	N/A	dir	runHLT_standalone_tail.log runHLT_standalone_test.log
testAllPT	-	OK	OK	OK	MISMATCH [ps]	6	FAIL	dir	runHLT_standalone_tail.log runHLT_standalone_test.log
testAllPT_run-stop-run	-	OK	OK	N/A	N/A	0	N/A	dir	runHLT_standalone_tail.log runHLT_standalone_test.log
testAllPartition	-	OK	OK	N/A	N/A	2	FAIL	dir	runHLT_standalone_tail.log runHLT_standalone_test.log

Operational Experience

- HLT framework successfully used since many years
  - 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
  - More than 40 days of 24/7 cosmic data taking after "LHC incident"



216 millions events

453 TB data

400k files

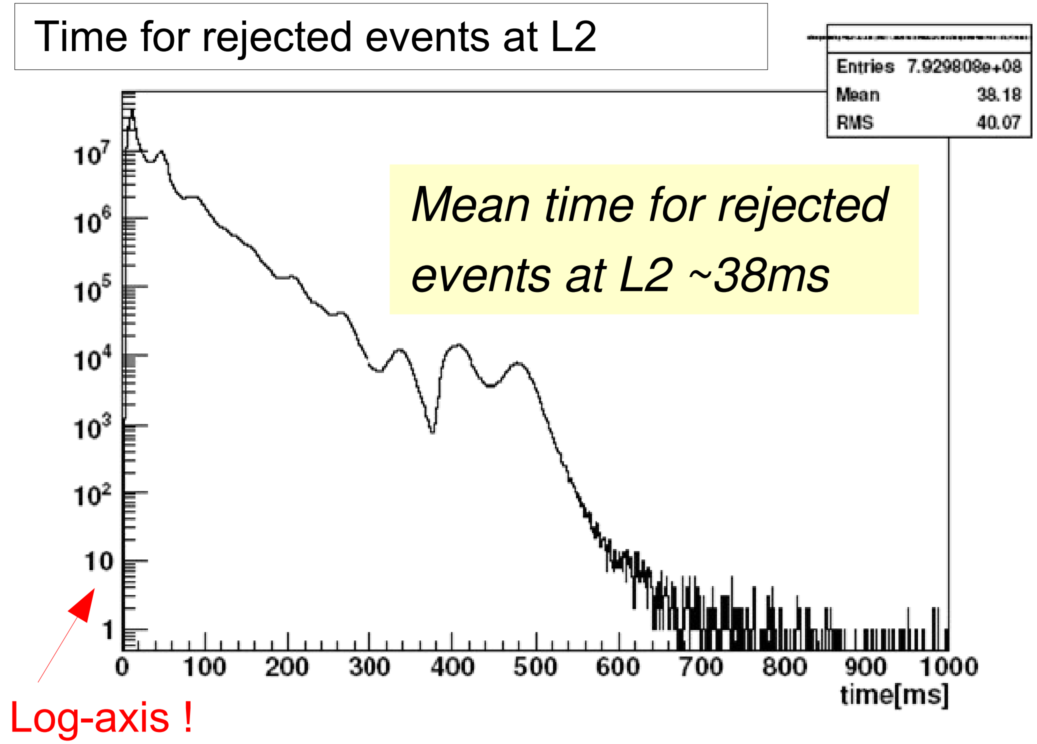
several streams

(see the following talk)

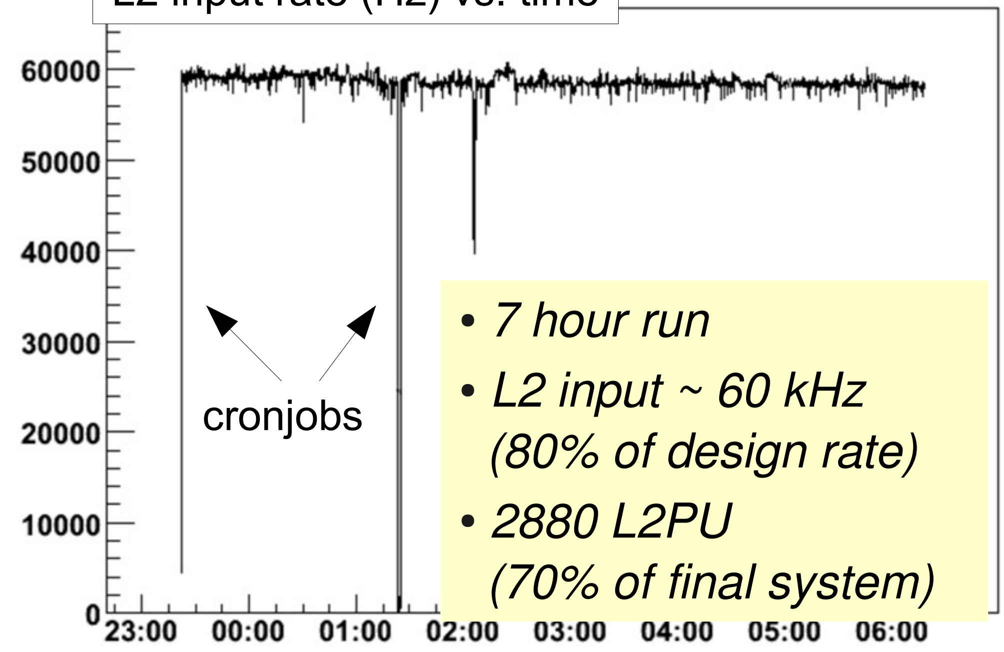
Operational Experience

- High-rate tests
  - Stress-test system with 10^{31} trigger menu and simulated data
  - Use most of available machines for L2
  - Timing for event processing is at specification

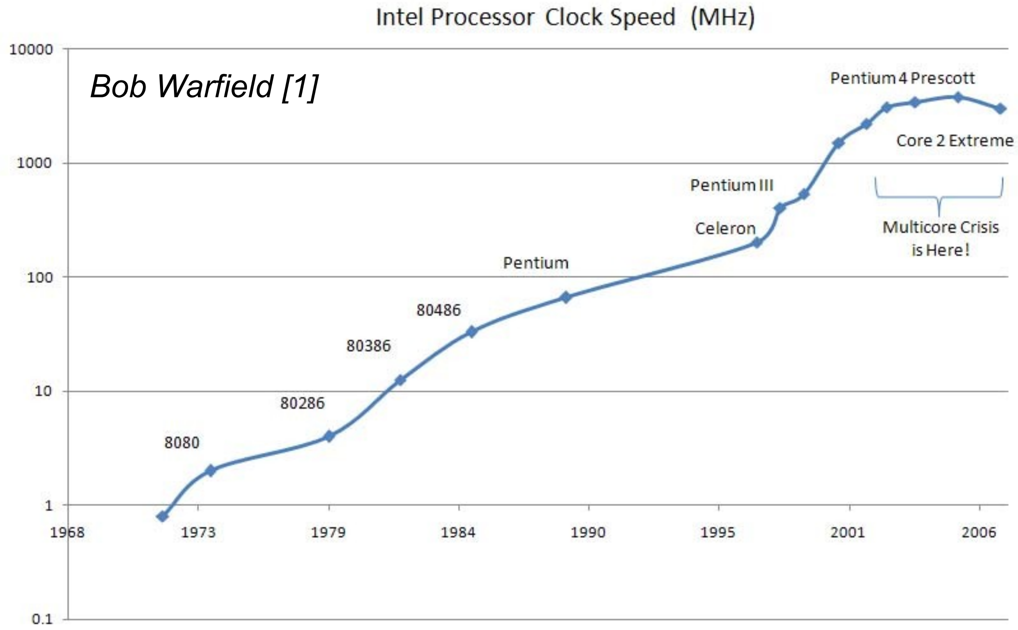
Time for rejected events at L2



L2 input rate (Hz) vs. time



Is Mr. Moore still valid ?



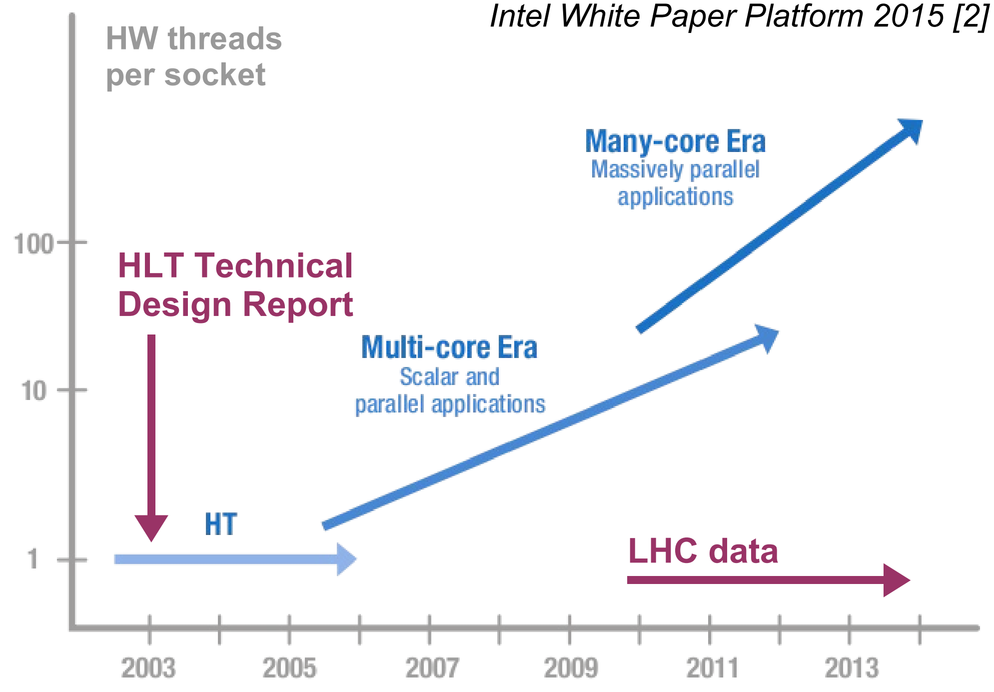
Over the past decades Moore's doubling of transistors was accompanied by a similar increase in clock speed.

"Free lunch" for HEP application developers

HLT-TDR (2003) assumed 8GHz CPU

This era has ended!

Intel White Paper Platform 2015 [2]



Moore is still valid, but most of the N times more transistors are packed into additional cores.

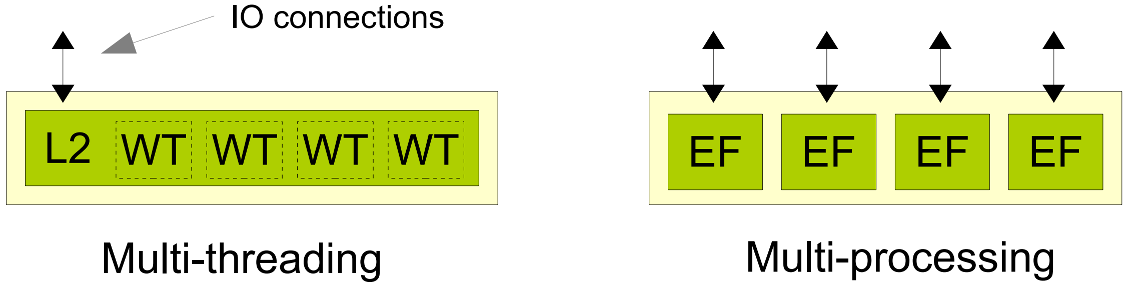
Not obvious that throughput will scale by simply multiplying the number of applications.

We have to make sure our applications make optimal use of the additional cores.

Parallelism and Multi-Core CPUs

- Parallelizing HEP applications
  - Exploit event-level parallelism
  - Computer Scientists call this “embarrassingly parallel”
  - Use multi-threading or multi-processing techniques

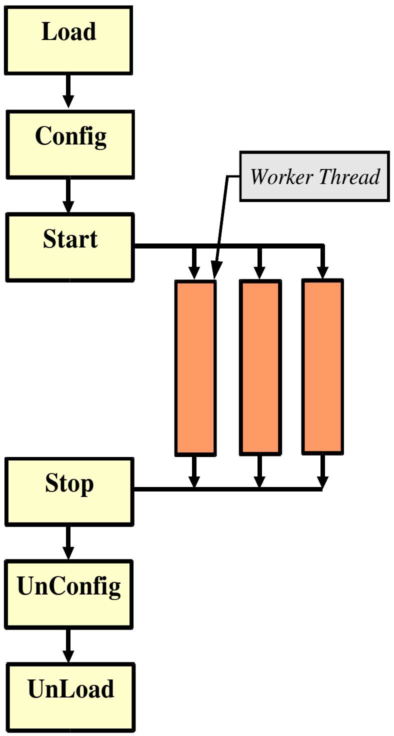
- Initial design of HLT applications
  - L2: multi-threaded process (alternatively multiple single-threaded processes)
  - EF: multiple processes
  - Over the past years we gained experience with both multi-processing and multi-threading



- Boundary conditions
  - Our applications are memory intensive (1-2 GB)
  - Most of our very large code base has been developed in pre-multi-core era
  - Changes have to be as transparent as possible
  - Currently O(1 GBit/s) bandwidth available to worker nodes

Experience with Multi-Threading

- Event processing in multiple worker threads
  - Use multiple worker threads per process each processing one event
  - Code sharing
  - Resource sharing (memory, sockets, ...)

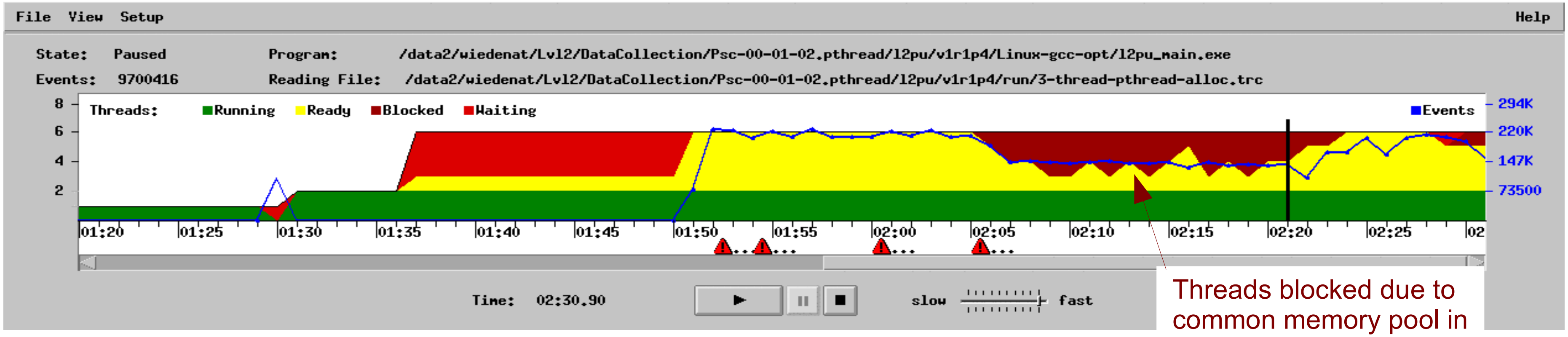


Multi-threading in L2PU

- In reality...

  It is very difficult to maintain a large code base **thread-safe** and **thread-efficient** in an open developer community

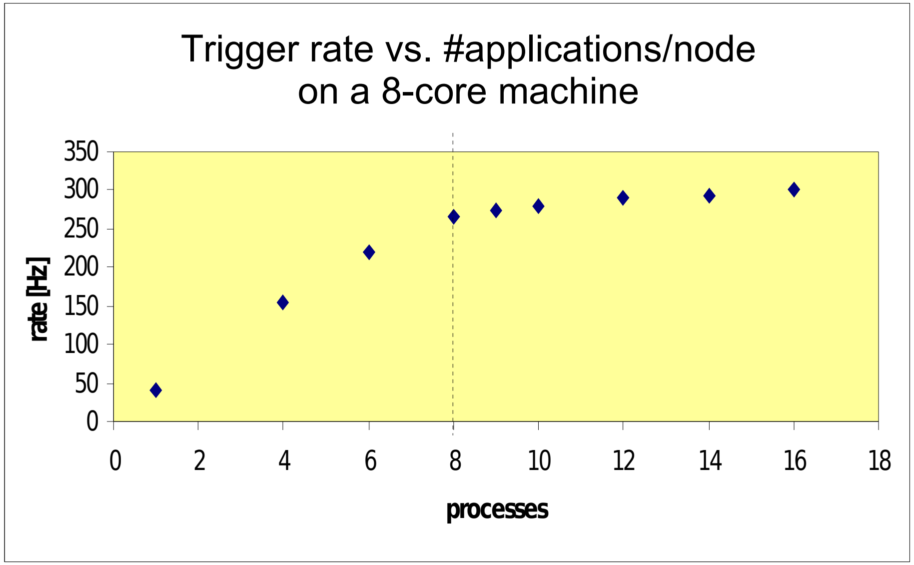
  - Requires careful tuning and expert knowledge
  - Difficult and time consuming, limits code-reuse from offline projects
  - Impact on turn-around times for patches, code improvements
  - External code is beyond our control. (see below for an STL example)
  - Multi-threading no longer used/supported for HLT Algorithms



Threads blocked due to common memory pool in STL containers

Experience with Multi-Processing

- Run one L2/EF application per core
  - Make use of the inherent event-level parallelism
  - No code changes needed
  - Independent processing units
  - Observed good scaling with number of cores →



- Problem: resource sharing
  - Resource requirements scale with number of applications
    - Memory size
    - OS resources: file descriptors, network sockets, ...
    - number of controlled applications
    - number of network connections to readout system
    - transfer of same configuration data N times to the same machine
  - Naive process multiplication will not scale into many-core era

Resource sharing

- Typical HEP application
  - Large amount of constant configuration and conditions data of O(100 MB)
  - Small event data of O(1 MB)
  - Common problem for offline reconstruction and HLT

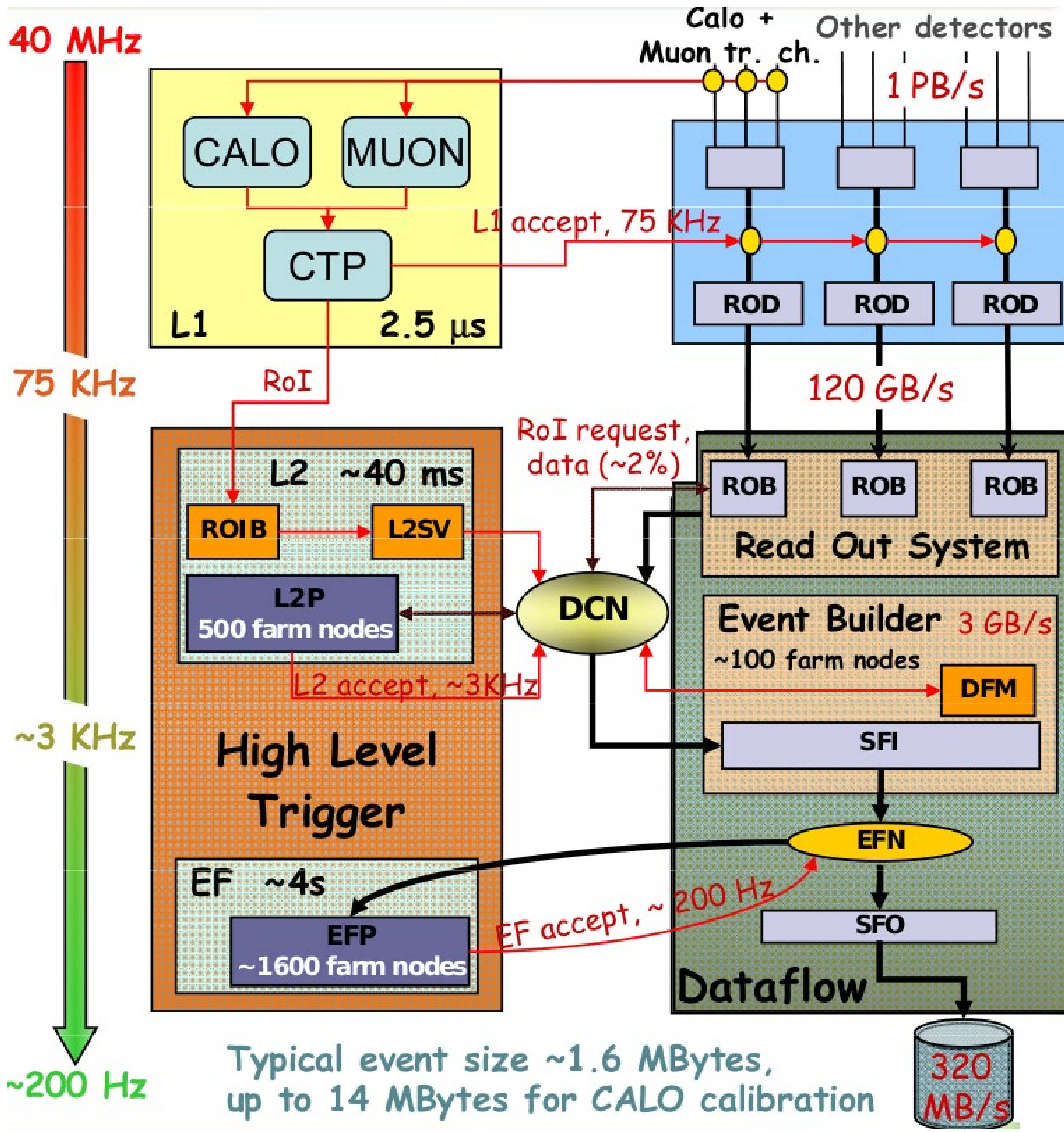
- Possible solutions
  - Multi-threading (see our experience)
  - Shared memory segments for constant data
    - Challenge is to implement this transparently for user-code
  - Exploit copy-on-write via fork (see [3] S. Binet, CHEP 2009)
    - Little code changes but difficult to implement for online (sockets, process control, etc.)
    - Unshared data never becomes shared again
  - Use special kernel module (KSM by RedHat)
    - Automatically identifies identical memory pages between processes
    - Identical pages are merged and marked as “shared”
    - Initial tests have been done within HLT and offline, but not conclusive yet

Summary

- The HLT framework based on the athena offline framework has been successfully 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.
- Constant performance monitoring is absolutely necessary
- The optimal use of multi-core processors requires further framework enhancements to reduce resource utilization. Many issues are shared with offline and can profit from common solutions.

Backup Slides

ATLAS Trigger/DAQ in Detail



A. Di Mattia, CHEP 2009

Appendix

- References

  - [1] Bob Warfield, http://smoothspan.wordpress.com/2007/09/06/a-picture-of-the-multicore-crisis/
  - [2] Intel® White Paper Platform 2015: Intel® Processor and Platform Evolution for the Next Decade
  - [3] S. Binet, Harnessing multicores: strategies and implementations in ATLAS, CHEP 2009