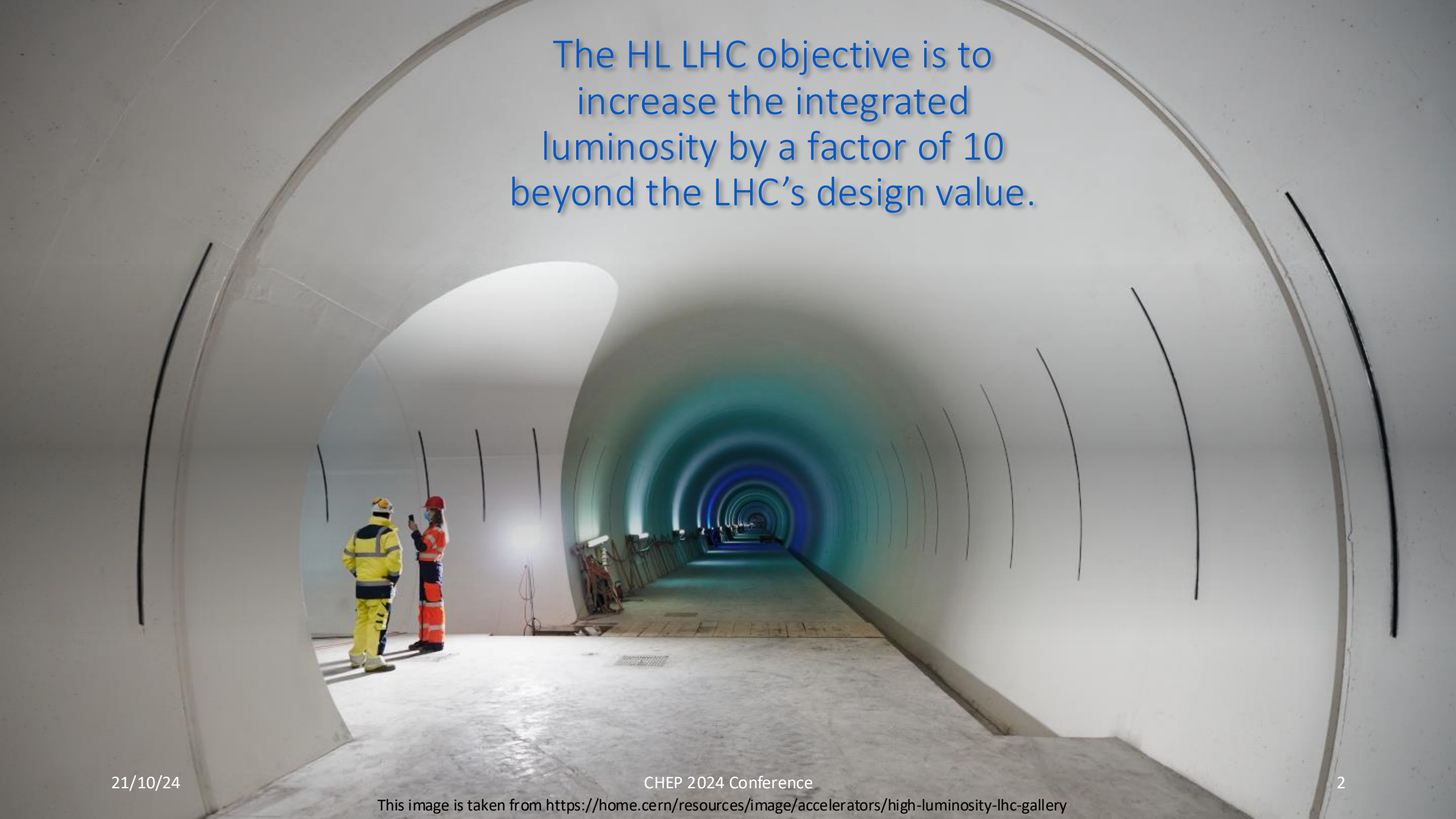
Leveraging the Run 3 experience for the evolution of the ATLAS software-based readout towards HL-LHC

On behalf of the ATLAS TDAQ Collaboration

Serguei Kolos

University of California, Irvine

The HL LHC objective is to increase the integrated luminosity by a factor of 10 beyond the LHC's design value.

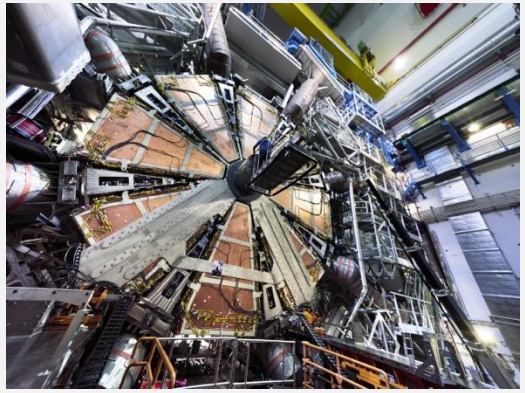


This image is taken from https://home.cern/resources/image/accelerators/high-luminosity-lhc-gallery

The Evolution of ATLAS

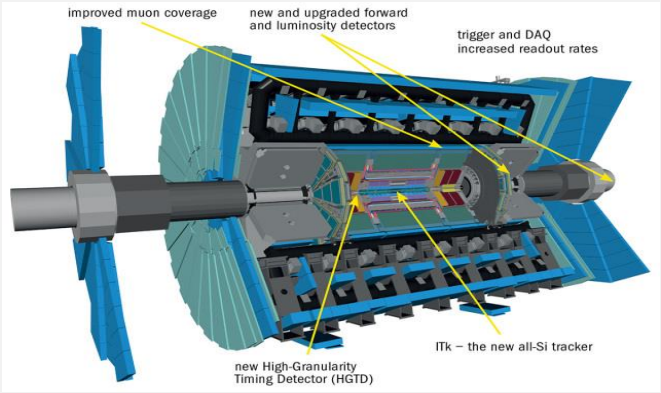
Some systems have been partially upgraded during LS2

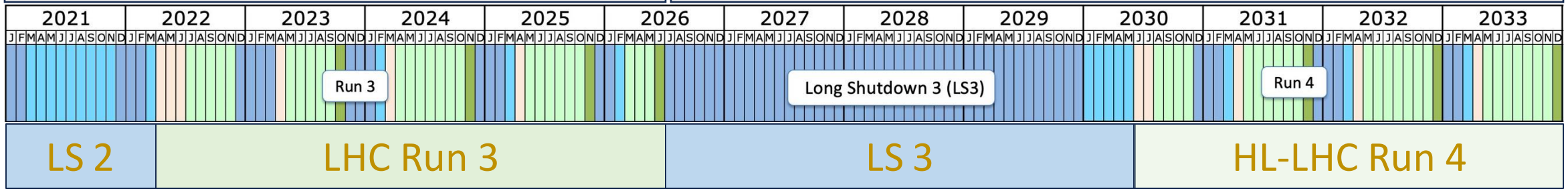
- New Small Wheel muon detector
- Liquid Argon Trigger
- Calorimeter Trigger
- Readout System



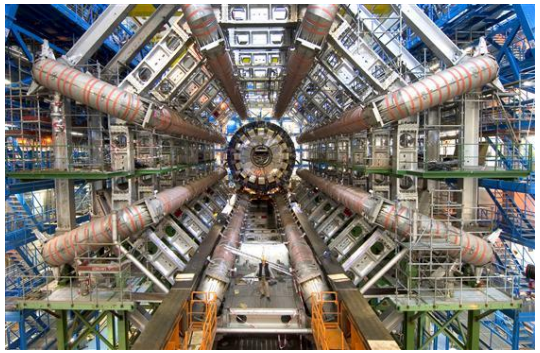
Most of the legacy systems will be completely replaced with the new ones

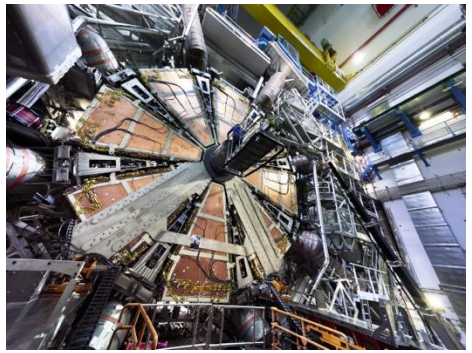
- New Tracking system
- New Muon detector
- New Timing detector
- New Luminosity detector
- New Trigger and DAQ systems

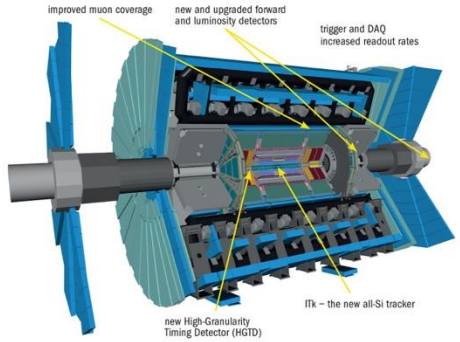




The ATLAS Readout System Evolution







The Legacy FE Electronics

Readout Driver (VME Board)

Readout System

High-Level Trigger

New FE Electronics

FELIX

SW Readout Driver

High-Level Trigger

LHC Run 3

New FE Electronics

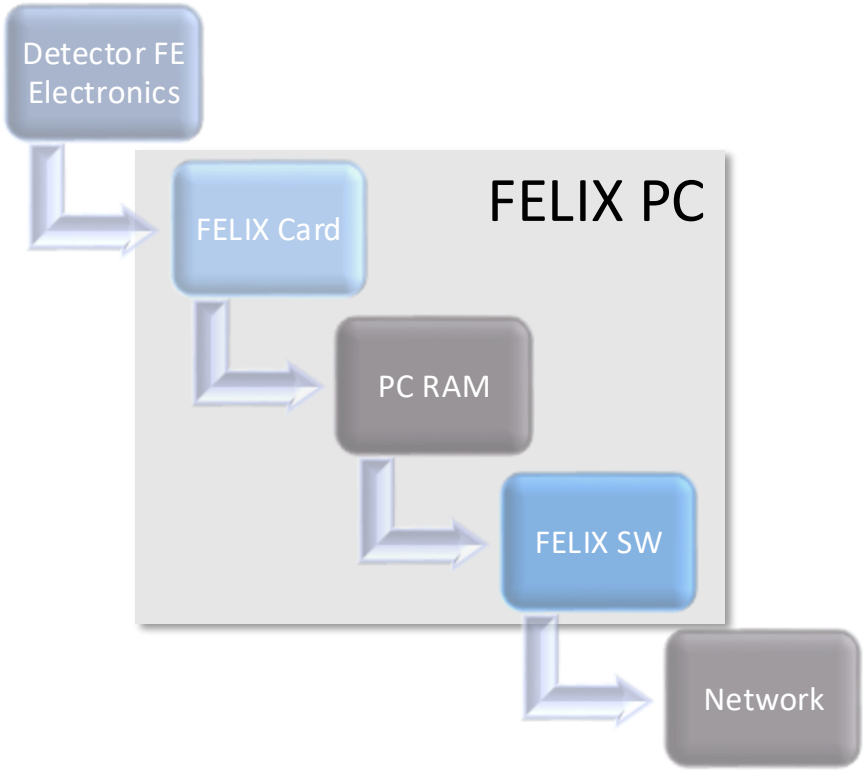
FELIX

Data Handler

Event Filter

HL-LHC Run 4

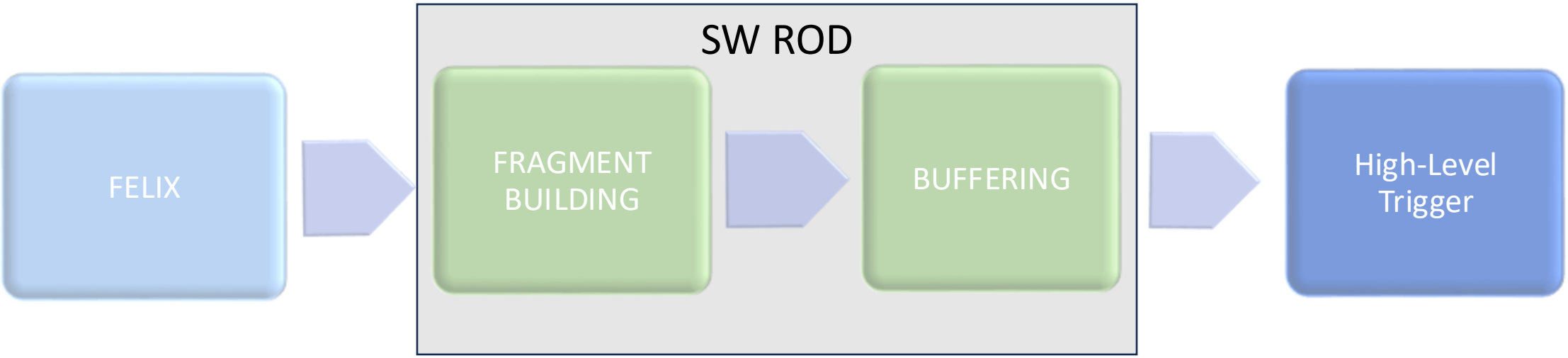
Front-End Link eXchange



- Custom PCIe card installed in commodity servers
- Receive data from the FE electronics via optical fibers:
  - Up to 48 input links per card
  - A physical link can be split into multiple virtual channels, called E-Links
- FELIX SW routes data to high-speed commercial network using RoCE* protocol

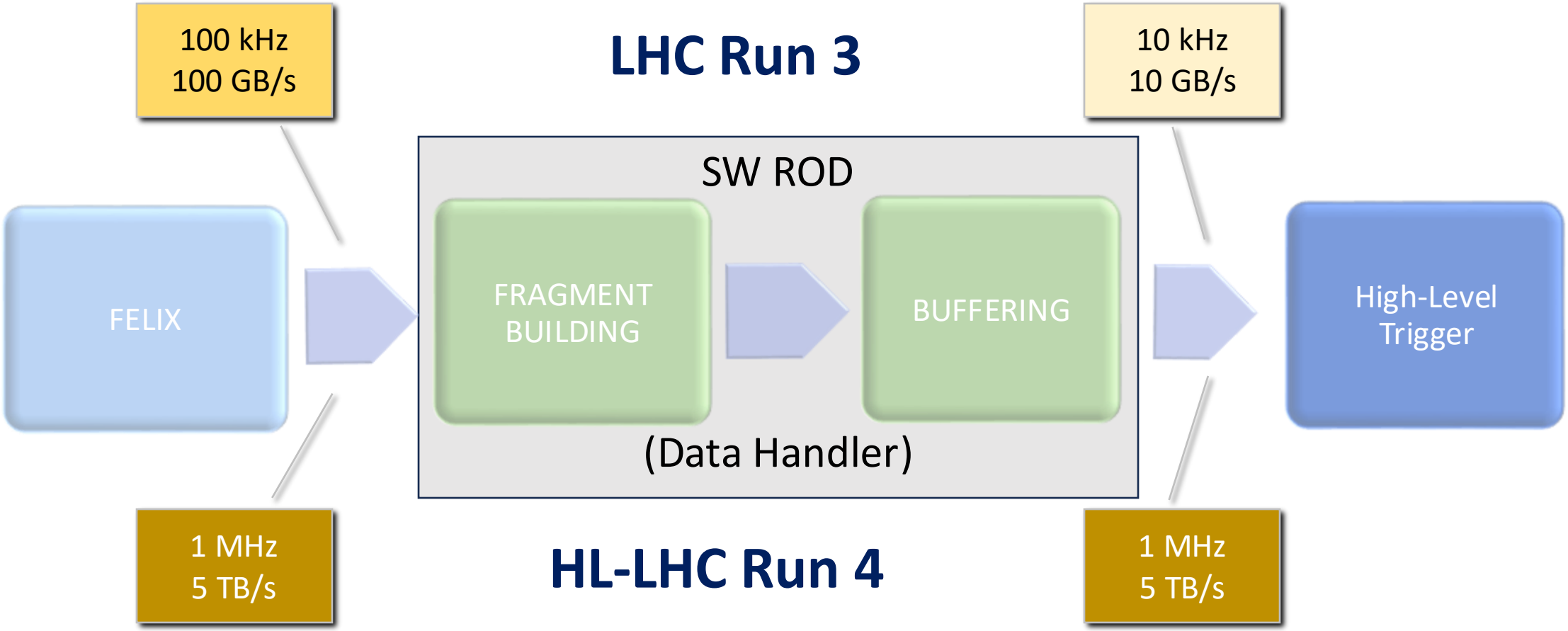
**Remote Direct Memory Access over Converged Ethernet*

SoftWare based Read-Out Driver

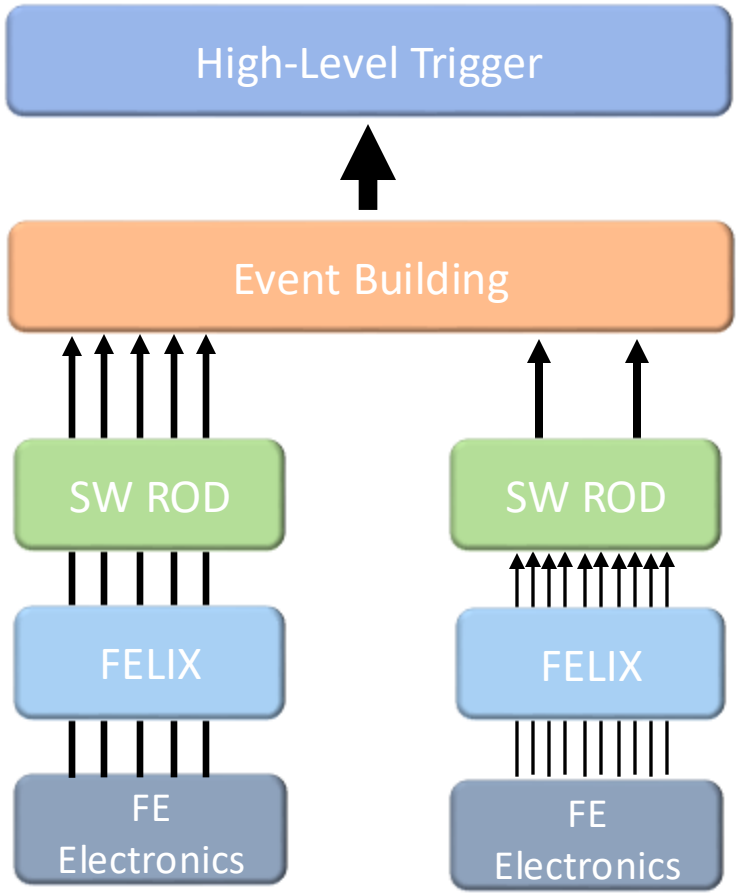


- SW processes running on commodity PCs
  - Aggregate data packets from FELIX E-Links into larger fragments
  - Buffer aggregated fragments until they are requested by the HLT
  - Release the fragments after they have been consumed by the HLT

Data Rate and Throughput Challenges for HL-LHC



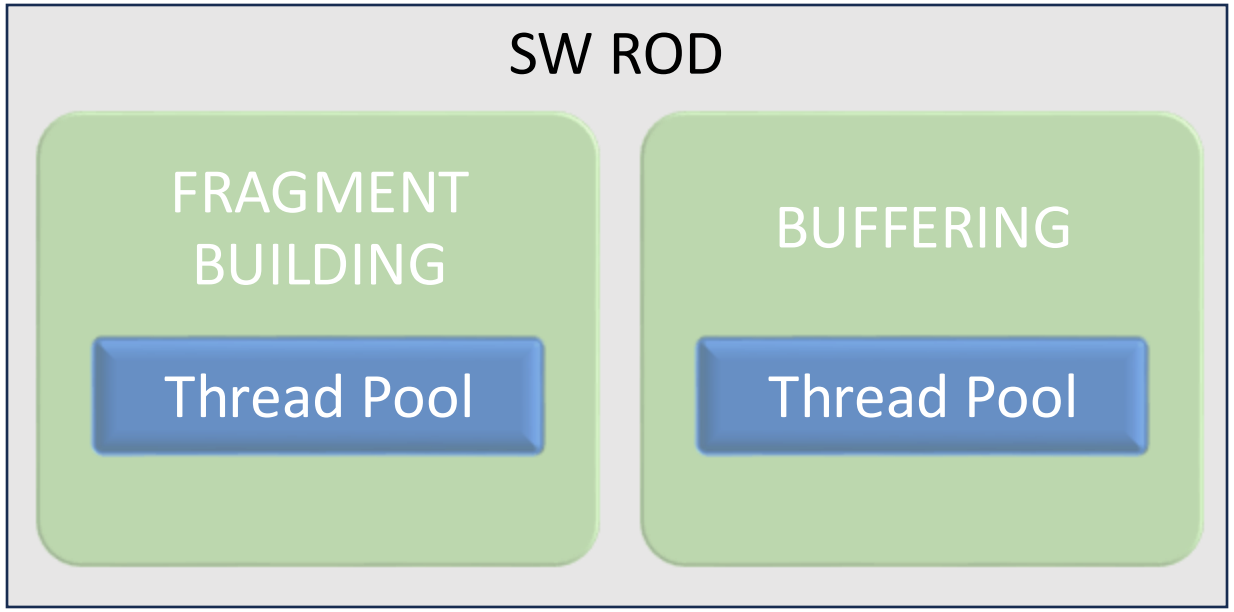
SW ROD Data Handling



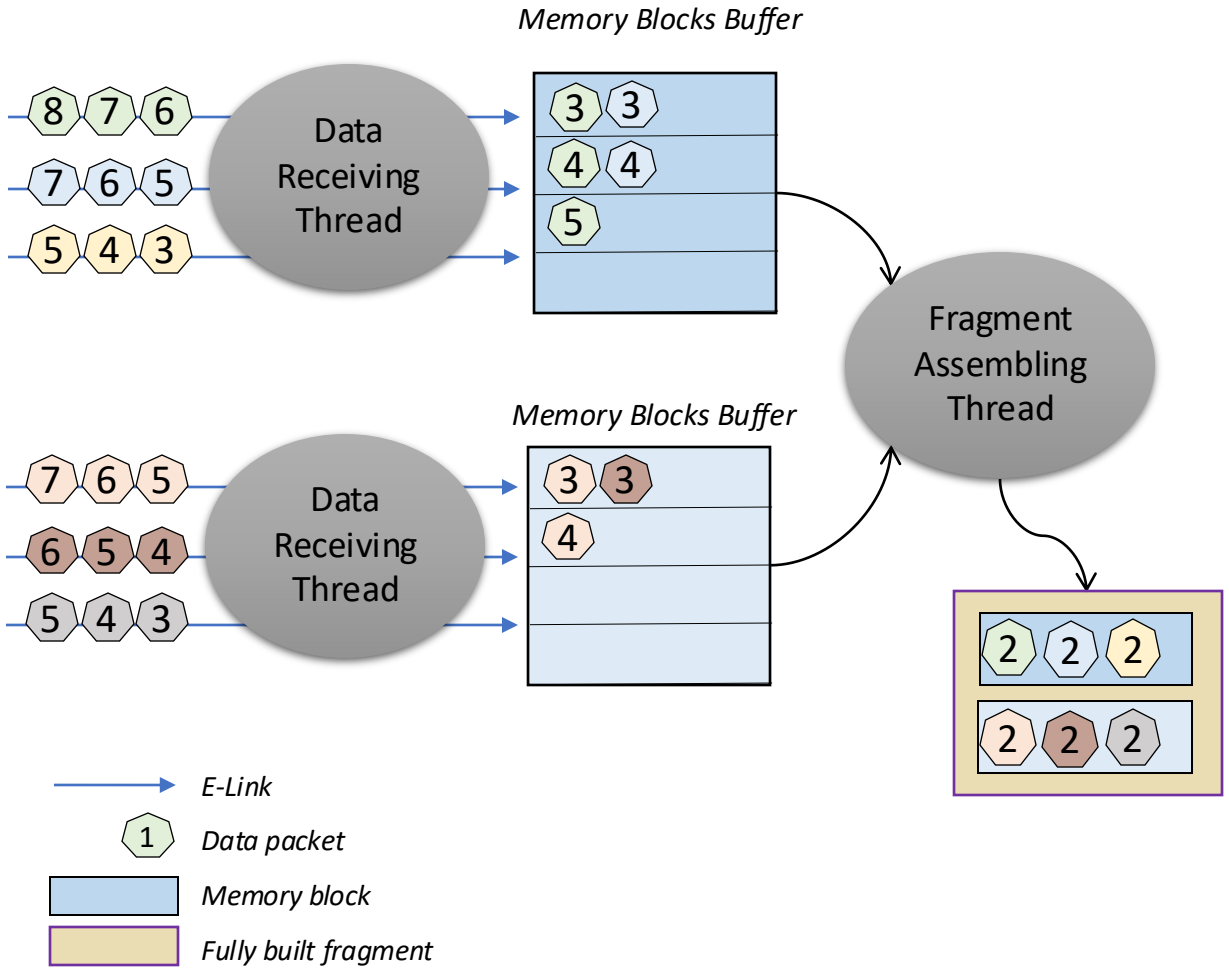
- For some detectors, FE electronics aggregate data from individual channels to larger fragments
- For others, FE electronics send data for every channel separately:
  - Aggregation must be done by SW ROD
- The final Event Building is done before passing data to the High-Level Trigger

The Fragment Building Challenge

- Data packets from O(100) E–Links must be aggregated into one fragment
  - Total packet rate **O(100)MHz**
- A single 3 GHz CPU core would offer **O(10)** cycles per data packet
- Requires extensive and efficient use of multi-threading



The SW ROD Fragment Builder Algorithm



- A fully built fragment consists of **N** memory blocks
  - **N** is the number of Data receiving threads
- The number of allocated memory blocks is proportional to **N**
- Aggregation performance scales well with the number of Data Receiving threads
- The first implementation performed much worse that expected

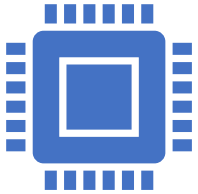
What could Affect the Fragment Builder Performance?

SW/HW environment

Network latency is O(1) us

Worst case OS scheduler latency is O(1) ms

Can only be measured and accepted



Code quality

CPU cache-friendly

Efficient multi-threading

Scalable to many CPU cores

Under control of the SW developers

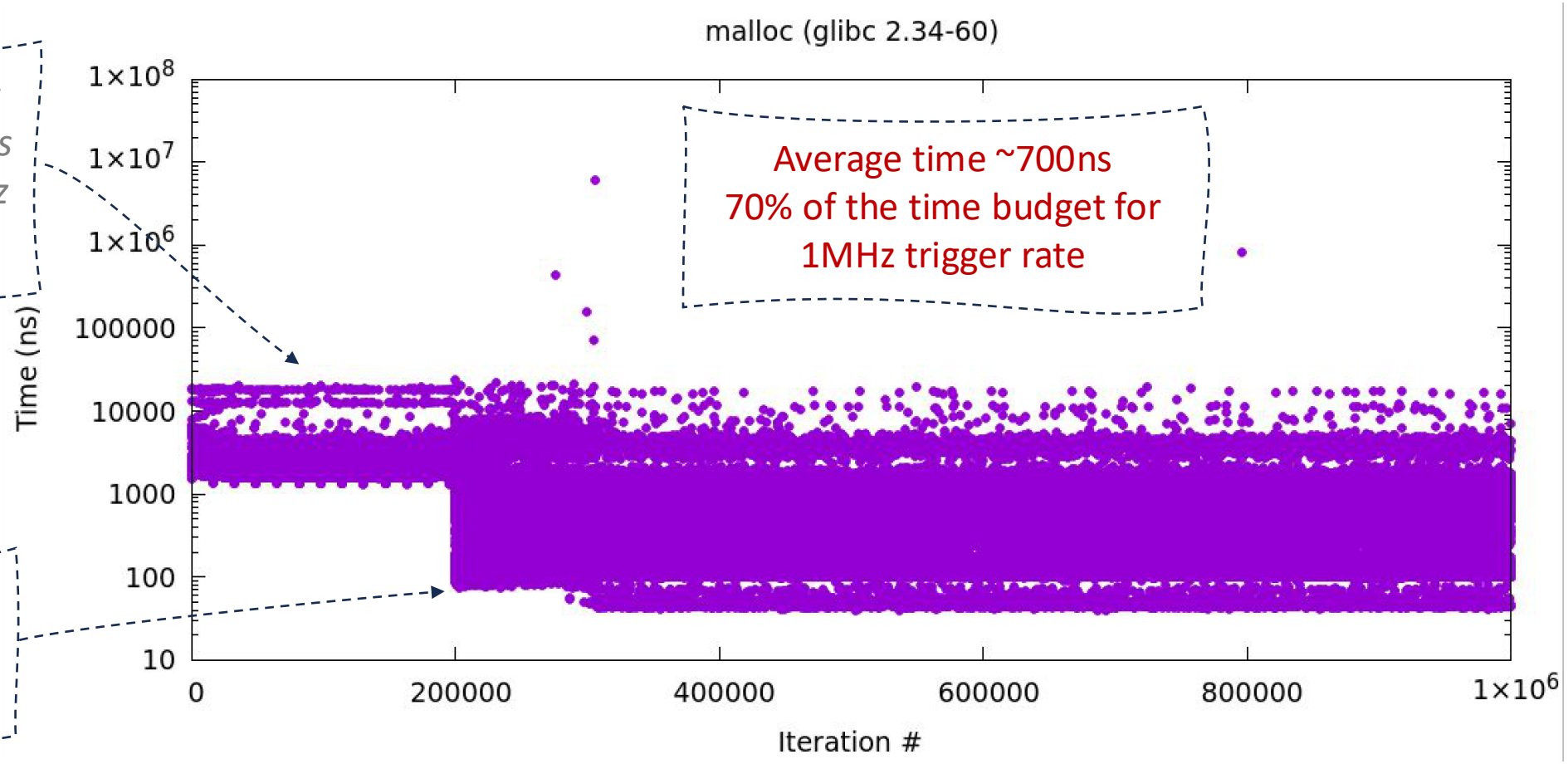
OS services

Memory management

Must be taken under control

Memory Allocation Time is the Primary Issue

200'000 allocations correspond to 200ms HLT latency at 1MHz trigger rate



Average time ~700ns
70% of the time budget for 1MHz trigger rate

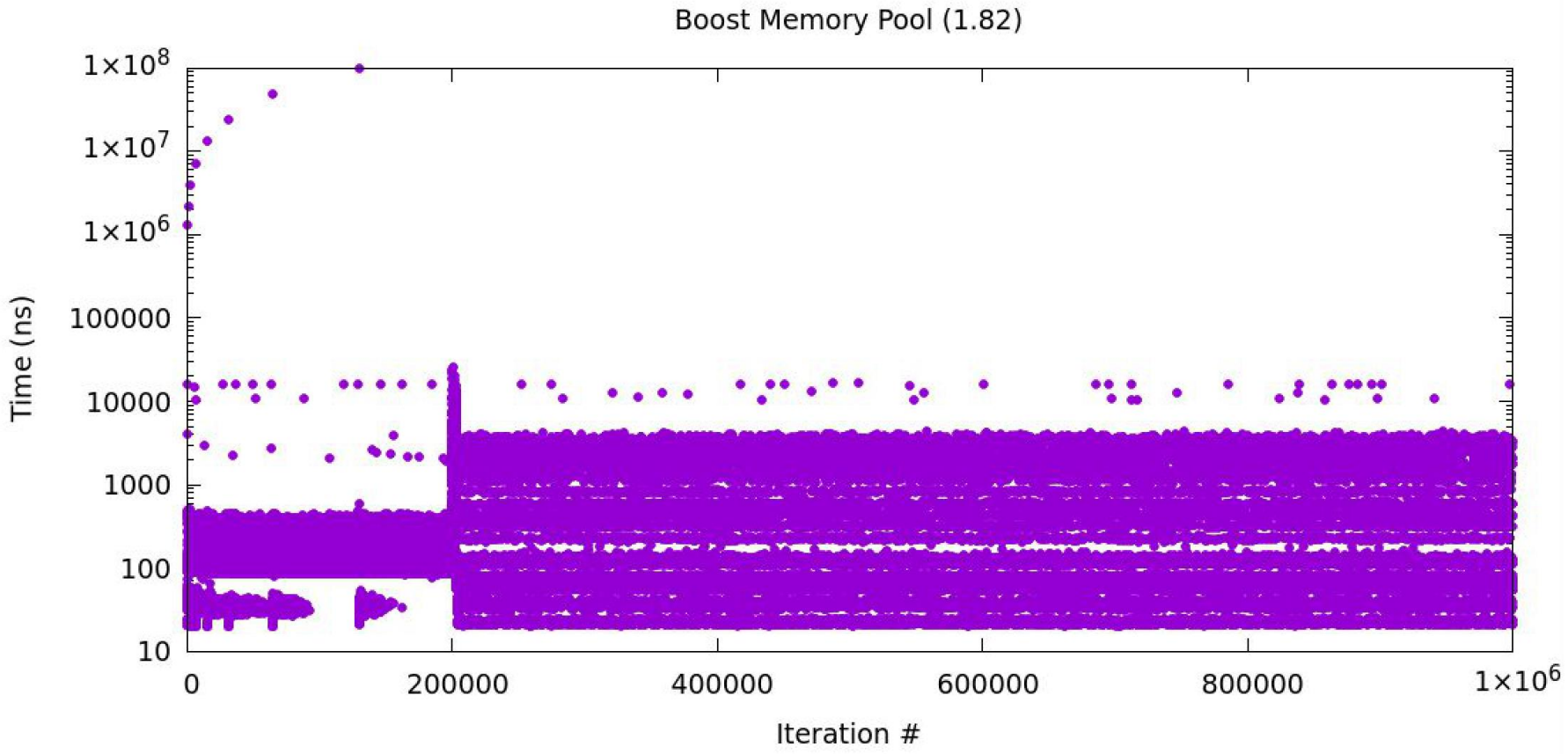
At this point memory deallocation starts

These and the other measurements were done on a standard Run 3 SW ROD server (specification is given in the Backup section)

Boost Memory Pool vs Malloc

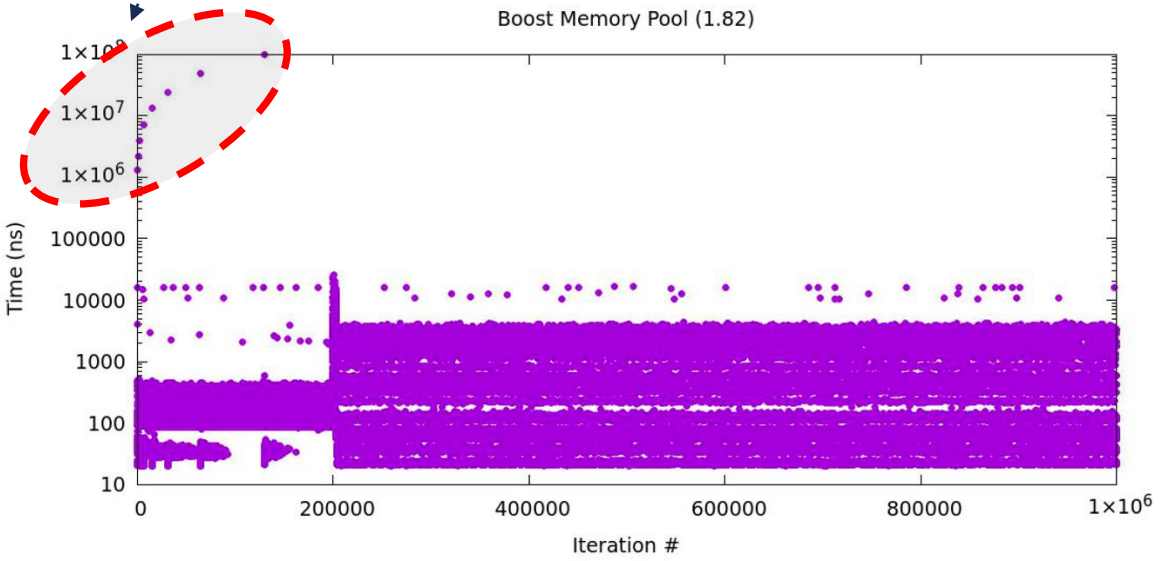
Reduced average allocation time to ~100ns

Reduces overhead from 70% to 10%



The Boost Memory Pool Issue

A new block size is two time larger than the previous one and allocation time grows up exponentially

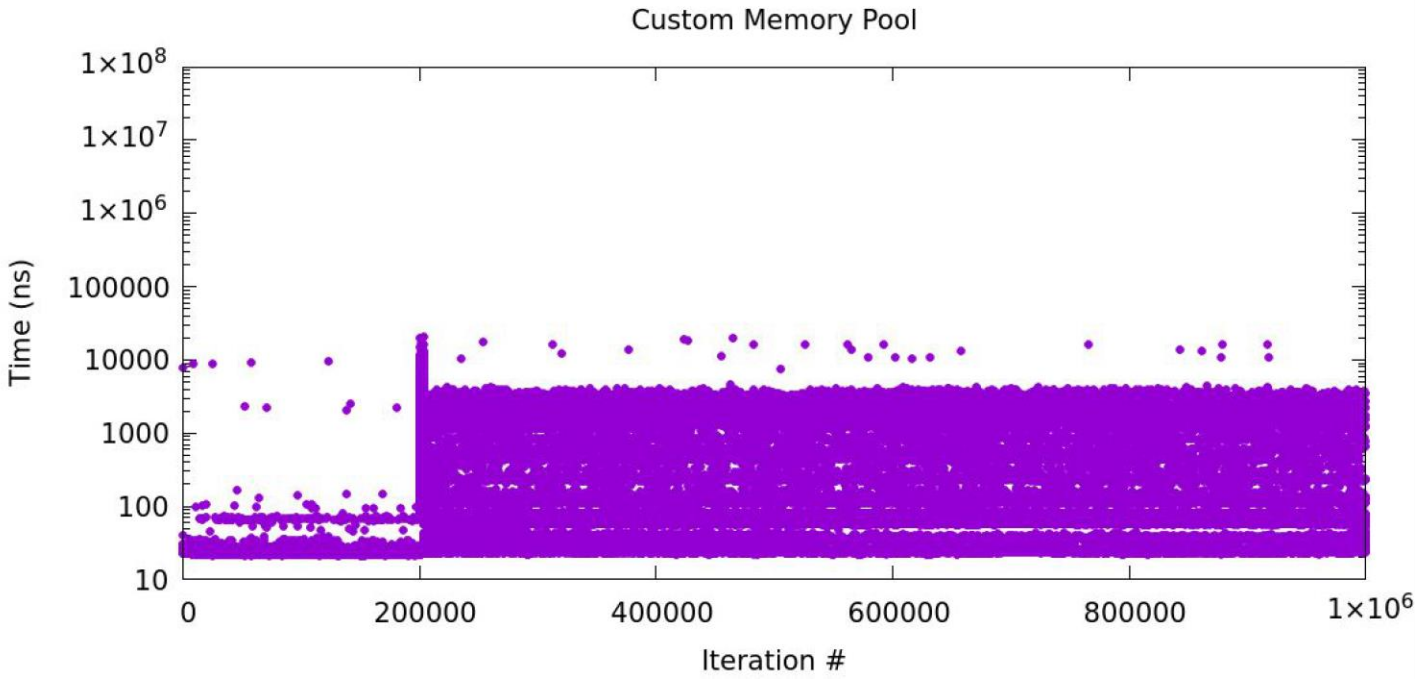


- Memory pool grows dynamically requesting large memory blocks from the OS
- Each time a new block is allocated Boost Memory Pool organizes individual data chunks into a single-linked list
  - The time is proportional to the number of chunks in the block
- Caused data loss when accumulated allocation time exceeded the maximum HLT latency

The New Memory Pool

- A custom Memory Pool implementation has been produced to address the issue:
  - Data chunks are added to the list of free chunks only when they are freed (returned to the Memory Pool)
  - Single open source header file

	Boost MemoryPool (boost 1.82)	Custom Memory Pool
Average allocation time (ns)	100	80
Maximum allocation time (ns)	10^8	10^4



Virtual Memory Management

Any memory allocation routine (e.g., malloc) returns a block of Virtual Memory

Is not immediately mapped to Physical Memory

Virtual Memory is mapped to Physical Memory when it's written for the first time

No mapping happens when memory is **read**

The mapping is done by 4KB pages

Large memory blocks are mapped in many steps

Every mapping operation incurs an overhead

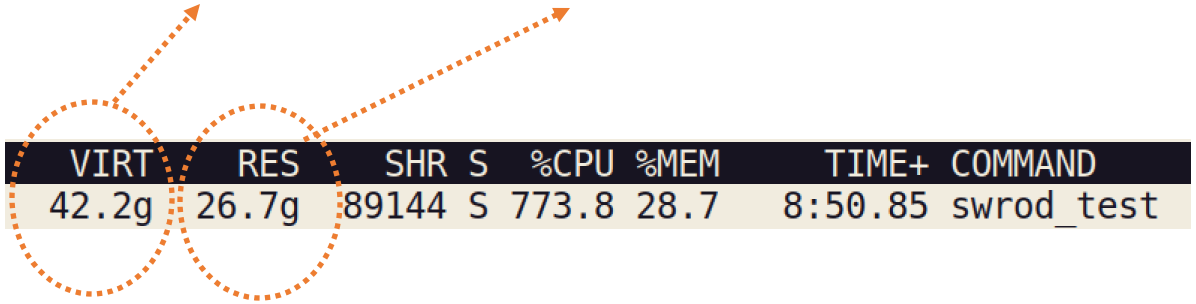
OS creates a new **Page Table Entry (PTE)** in main memory:

- One table per process
- A mutex lock is used to support multi-threading

This PTE is added to the **Translation Lookaside Buffer (TLB)** in CPU's cache memory

The cost of one page mapping on SW ROD computer is O(100)us

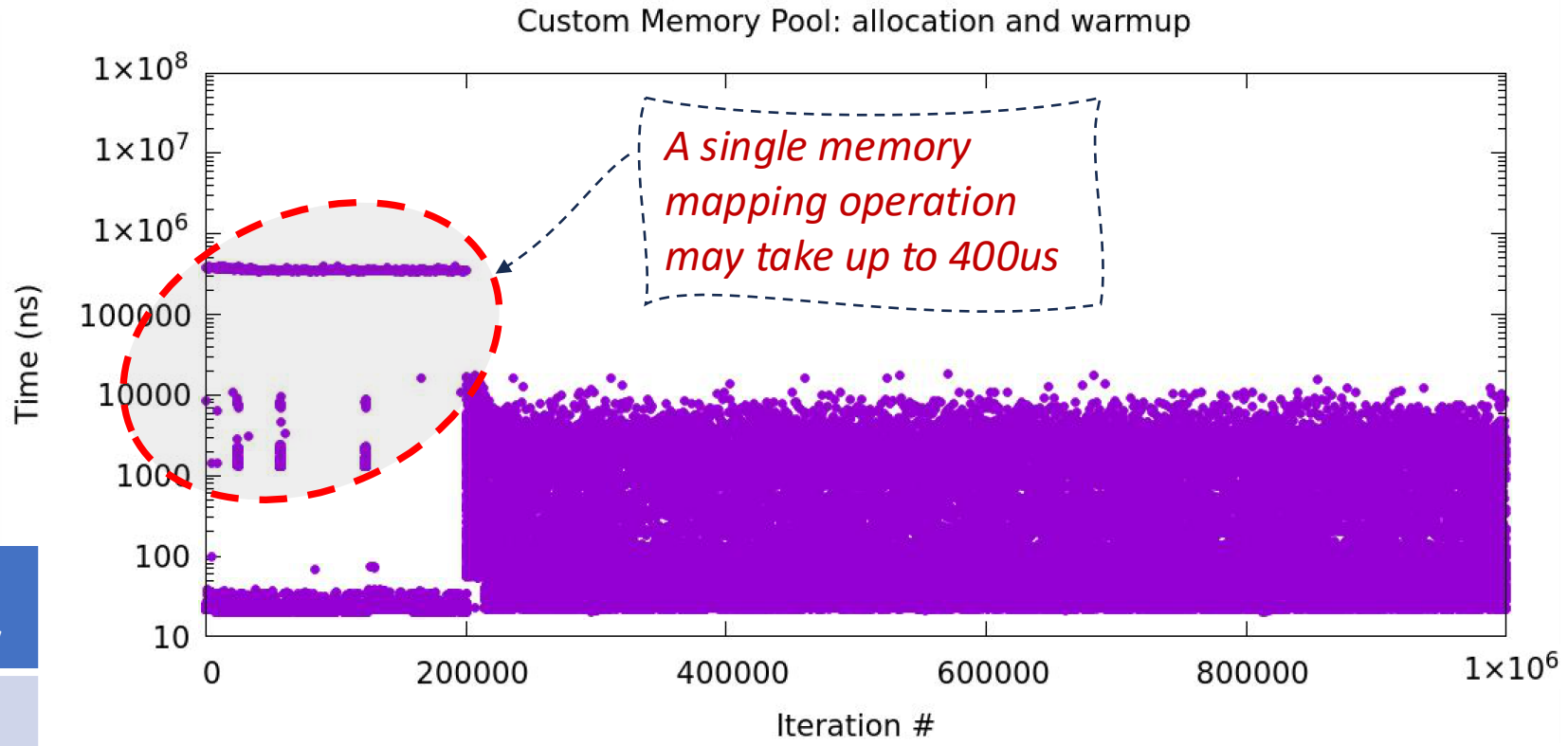
Running Alma9 Linux



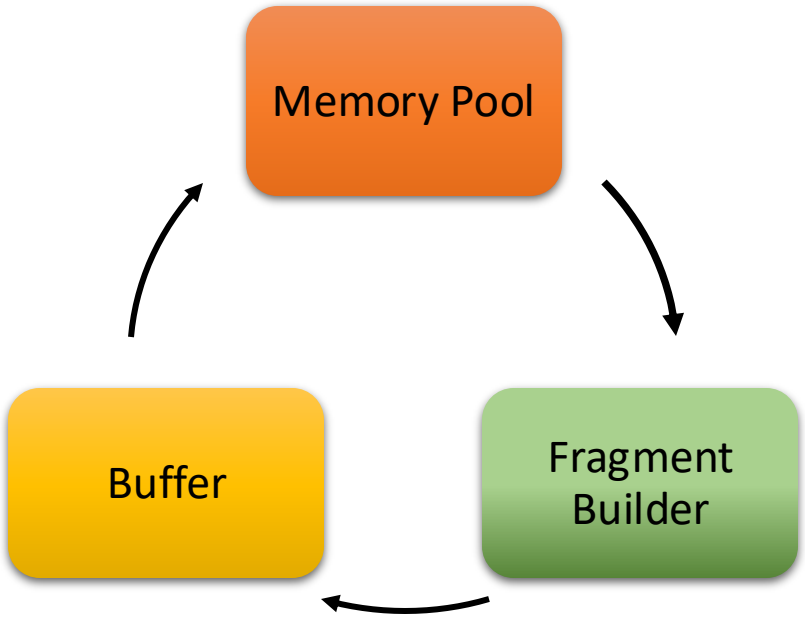
The Virtual Memory Mapping Overhead

- Custom Memory Pool with 4KB memory chunks:
  - The worst case
- Written 1 byte to the beginning of each block during allocation

	Cold Memory	Warm Memory
Average time (ns)	700	80



How to Mitigate this Issue?

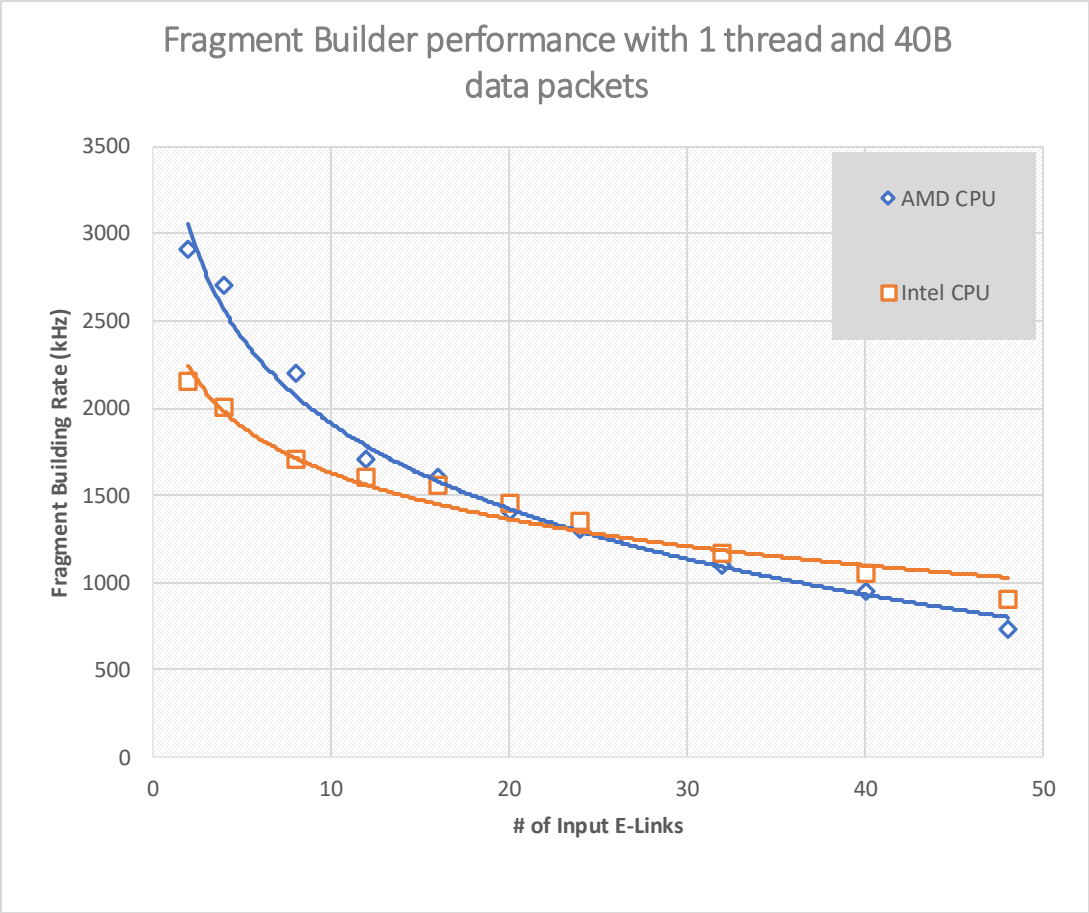


100K is a typical total number of memory blocks for a 100kHz run

Pre-allocate and warm up 100K memory blocks **before** starting a new run

One must be careful as this would require pre-allocation of a lot of memory for 1MHz data taking

Putting it all Together

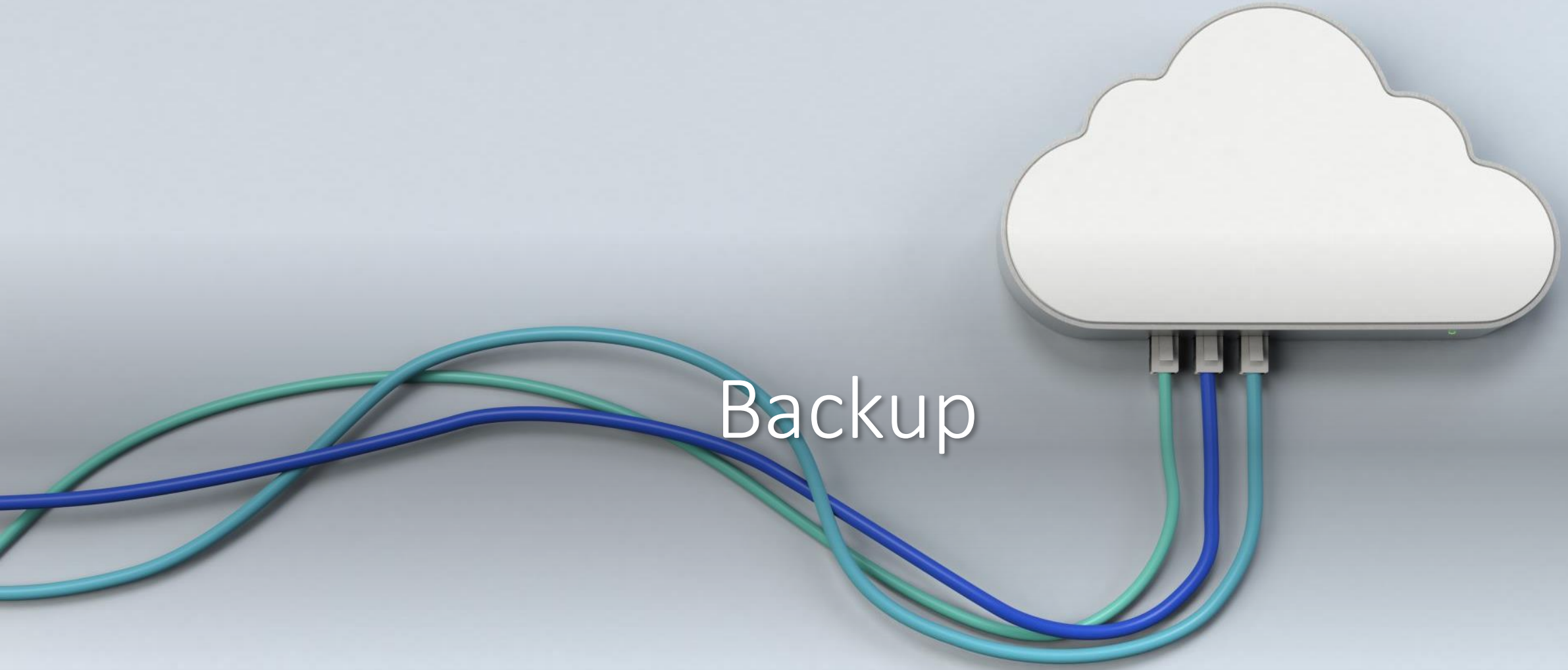


- The measurements were done on the Run 4 SW ROD candidate computers:
  - Specifications in the second slide of the Backup section
- Affinity was set to optimize CPU cache usage:
  - Cores 0-15 for **Intel**
  - Cores 0-3 for **AMD** (*use the same L3 Cache instance*)
- AMD is faster for up to 24 E-Links:
  - Better cache performance
- Intel is faster for more than 24 E-Links:
  - Higher CPU frequency

Summary

- HL-LHC Run 4 Readout system of the ATLAS experiment will operate at 1MHz input data rate
  - This poses extremely challenging performance requirements to the new SW-based Readout Application
- Very efficient multi-threading algorithm has been developed to utilize the full power of modern CPUs:
  - Memory management was the main issue that affected performance and maximum latency
  - Using custom Memory Pool implementation, the issues have been successfully addressed

Backup



Run 3 SW ROD Computer

CPU	**2 x** Intel(R) Xeon(R) Gold 5218 **2300 MHz**	
	2 x 16 physical cores	
Cache	L1d:	**1 MiB** (32 instances)
	L1i:	**1 MiB** (32 instances)
	L2:	**32 MiB** (32 instances)
	L3:	**44 MiB** (2 instances)
RAM	**96 GB**	
	12 x 8GB DDR4 Samsung **2666 MT/s**	
Network	Mellanox MT27800 Connect-X5 **100 Gb/s**	

Run 4 Data Handler Candidate Computers

CPU	**2 x** Intel(R) Xeon(R) Gold 6444Y **4000 MHz**	
	2 x 16 physical cores	
Cache	L1d:	**1.5 MiB** (32 instances)
	L1i:	**1 MiB** (32 instances)
	L2:	**64 MiB** (32 instances)
	L3:	**90 MiB** (2 instances)
RAM	**128 GB**	
	8 x 16GB DDR5 Micron Technology **4800 MT/s**	
Network	Mellanox MT2910 Connect-X7 **400 Gb/s**	

CPU	AMD EPYC 9354P **3800 MHz**	
	32 physical cores	
Cache	L1d:	**1 MiB** (32 instances)
	L1i:	**1 MiB** (32 instances)
	L2:	**32 MiB** (32 instances)
	L3:	**256 MiB** (8 instances)
RAM	**64 GB**	
	4 x 16GB DDR5 Micron Technology **4800 MT/s**	
Network	Mellanox MT2910 Connect-X7 **400 Gb/s**	