Fast TPC online tracking on GPUs and asynchronous data-processing in the ALICE HLT to facilitate online calibration

Dr. David Rohr for the ALICE Collaboration

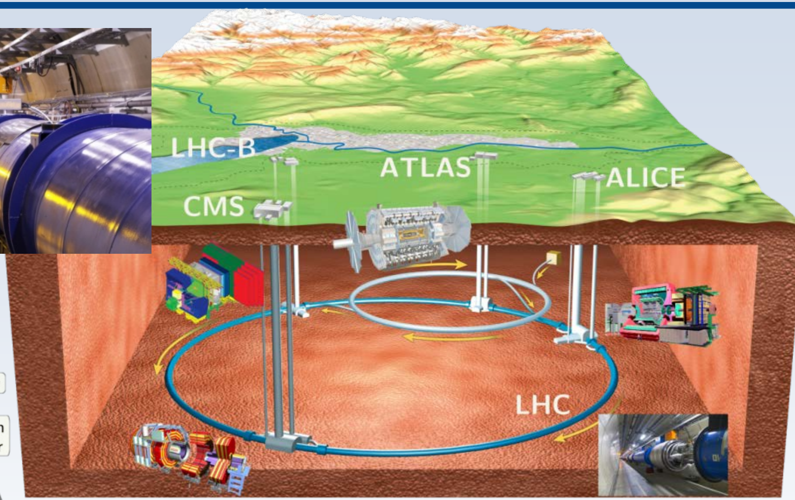
drohr@cern.ch

Frankfurt Institute for Advanced Studies

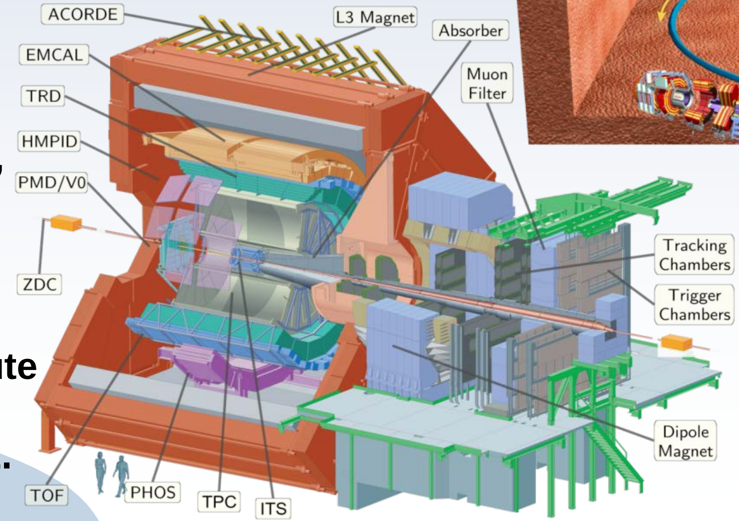
CHEP 2015, Okinawa, 13.4.2015

Funded by:
HIC for FAIR
Helmholtz International Center

ALICE at the LHC



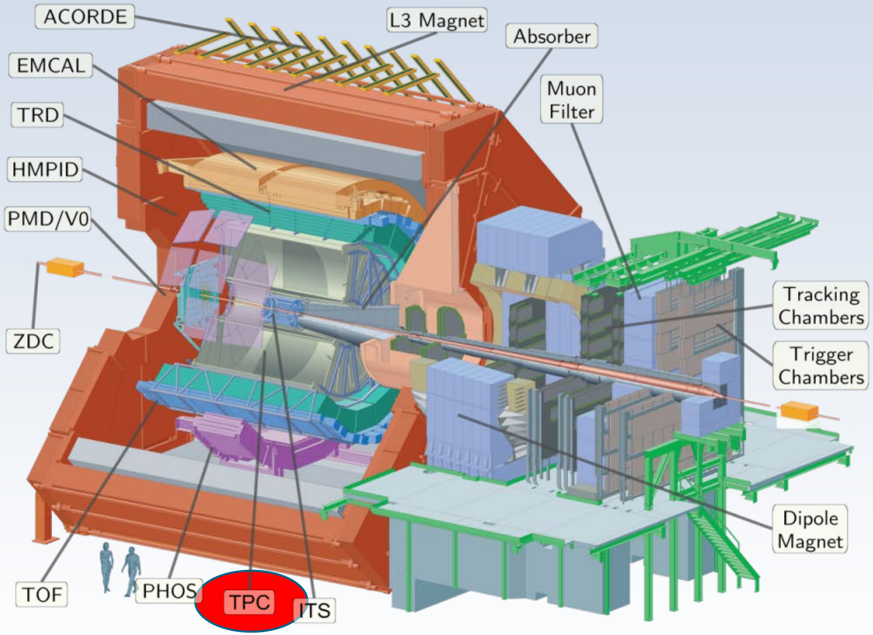
- The Large Hadron Collider (LHC) at CERN is today‘s most powerful particle accelerator colliding protons and lead ions.
- ALICE is one of the four major experiments, designed primarily for heavy ion studies.
- The Time Projection Chamber (TPC) is ALICE’ primary detector for track reconstruction.
- The High Level trigger (HLT) is an online compute farm for real-time data reconstruction for ALICE.



Challenges in this talk

- **Reconstruction of particle trajectories in the TPC is computationally very expensive:**
  - Several thousand tracks per event.
  - High combinatorial complexity.

- **As a gas-based detector, the TPC is sensitive to calibration.**
  - Environment variables such as temperature and pressure affect the calibration.
  - The conditions change during a run.

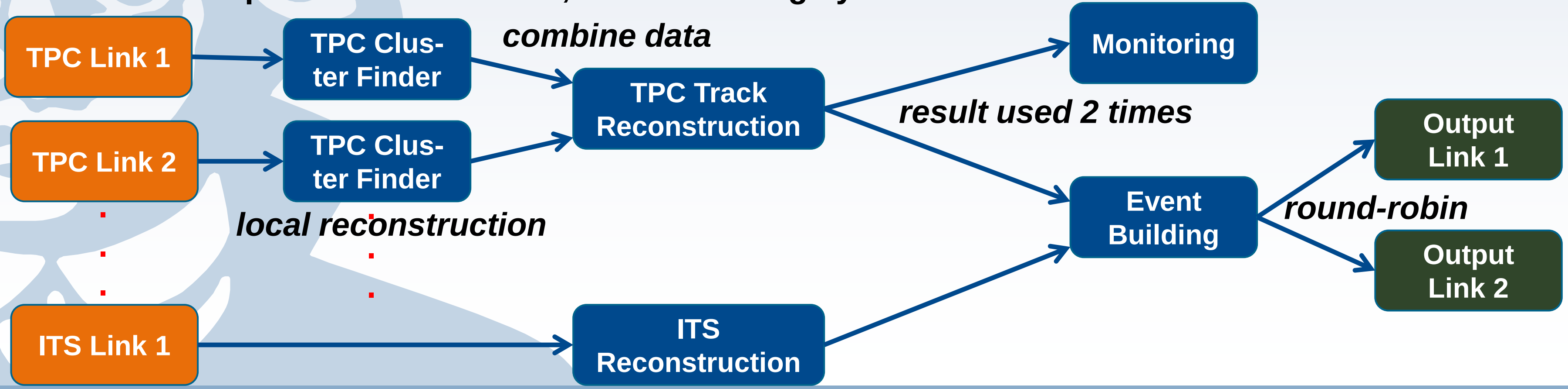


→ **Challenging tasks for the HLT:**
  - Needs fast reconstruction algorithm for online operation.
  - Detectors must be continuously calibrated online.

Online calibration can save compute resources in the future by removing some calibration passes.

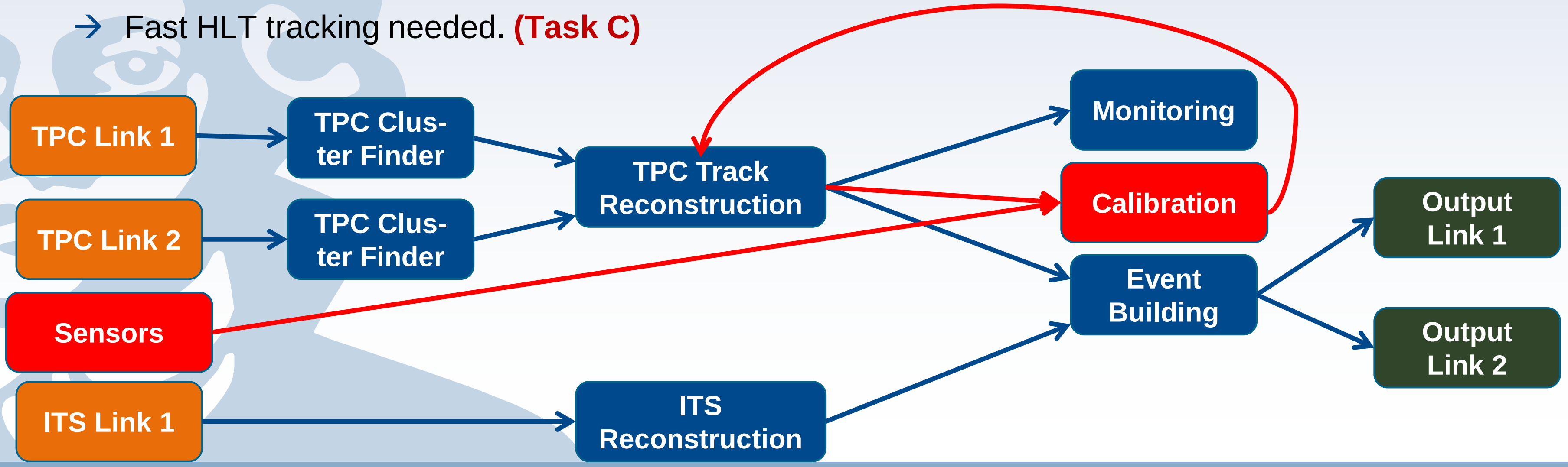
The HLT Framework

- **HLT reconstruction is performed in processing chains.**
  - **Sources** (detector links) feed data in the chain.
  - **Sinks** (output links to DAQ) collect the results.
  - In between, **processing components** can process data / (parts of) events.
  - → The HLT is a directed graph without loops. (Original design decision for technical reasons.)
- **Models: local reconstruction first, combine data for global reconstruction, use results of one component for two tasks, load-balancing by round-robin distribution**



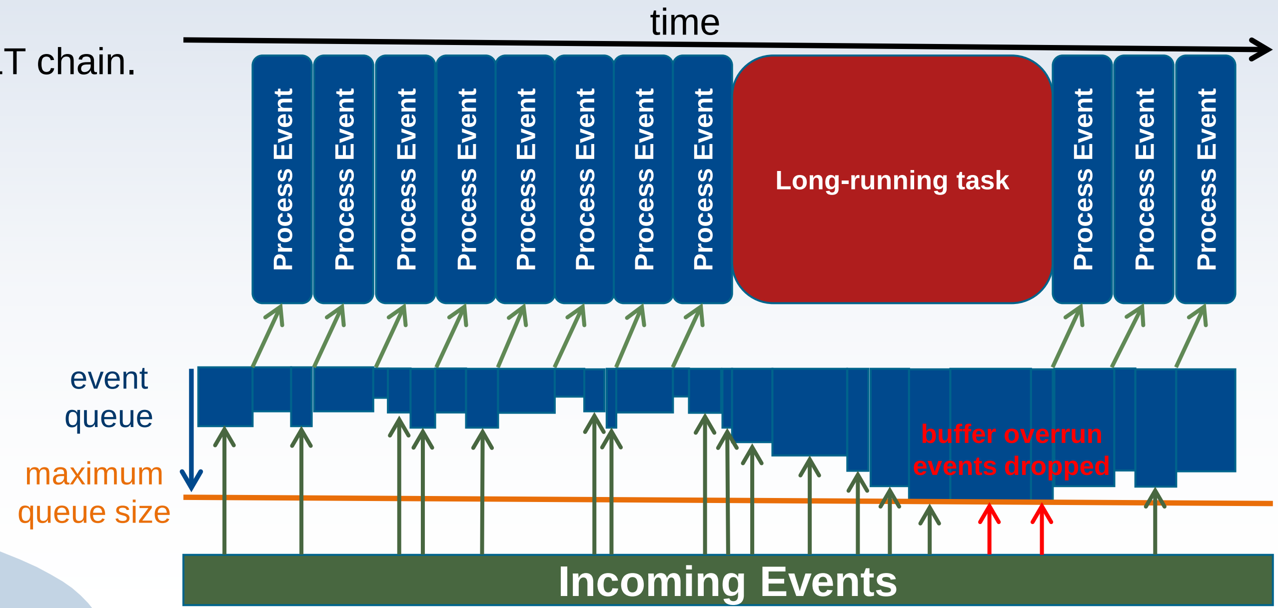
Challenges for Online Calibration

- **Online calibration would need to feed back data into the reconstruction.**
  → Problem: HLT Framework does not support loops. **(Task A)**
- **Additional Input for sensors needed (temperature / pressure).**
  → Problem: data transport and synchronization event-based, sensor data not event-based. **(Task B)**
- **Calibration needs real-time tracking.**
  → Fast HLT tracking needed. **(Task C)**



Challenges for Online Calibration

- **HLT processing based on events.**
- **Components process one event after another.**
  - → Long running infrequent tasks could make the event buffer overrun, even if the average processing time is short.
  - → This will stall the entire HLT chain.
  - → Events will be lost.
    - → **Task D**
- **Example in calibration:**
  - Accumulating events first.
  - Long-running fit later.



Overview

We have identified four necessary prerequisites for online calibration:

- **Task A: Framework capability to feed back data (loops).**
- **Task B: Custom data sources in the framework.**
- **Task C: Fast track reconstruction.**
- **Task D: Infrequent long-running tasks in a component.**

The prerequisites as above (except C) are formulated abstractly – not related to calibration.

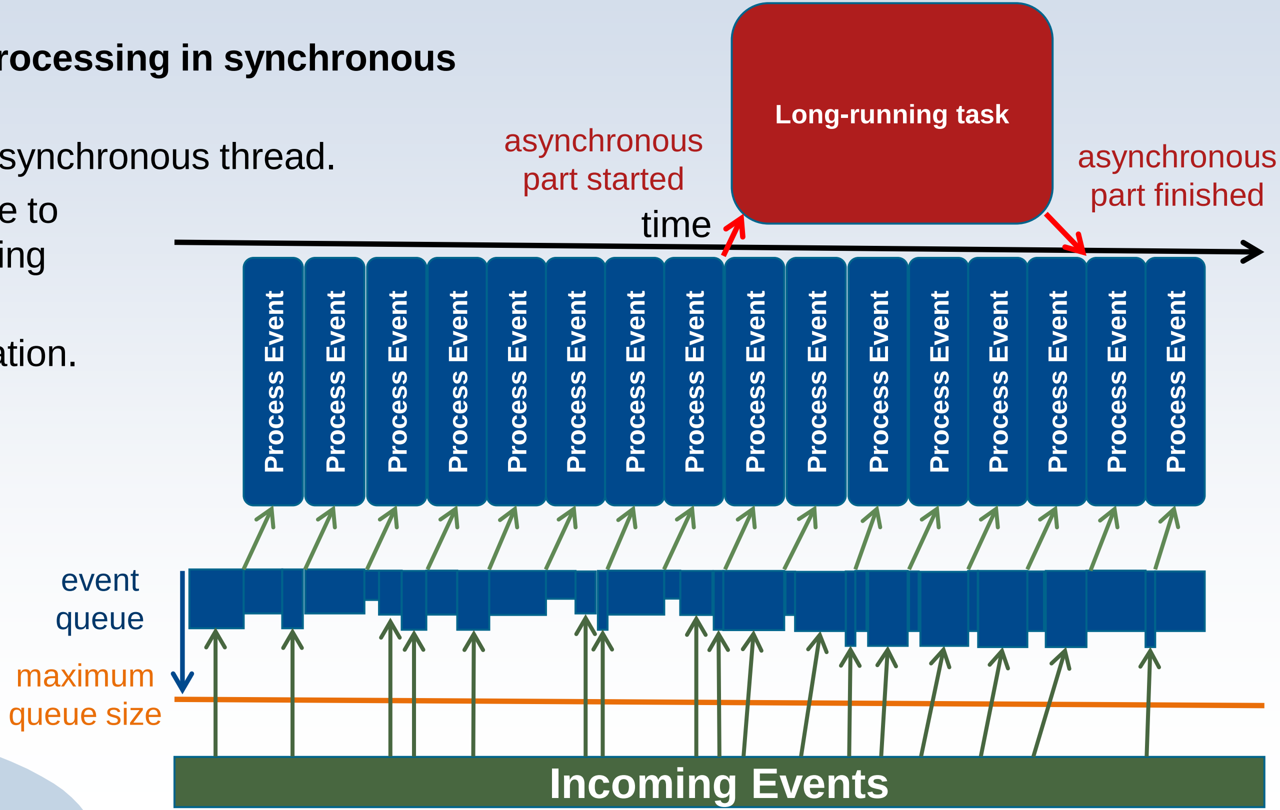
- We want to implement them as standalone features because they can be used in other scenarios as well.

We want to introduce these capabilities in the least invasive way.

- The HLT framework was proven stable in run 1.
- We want to avoid serious changes.
- We try to implement the new features on a component level instead of the framework level.

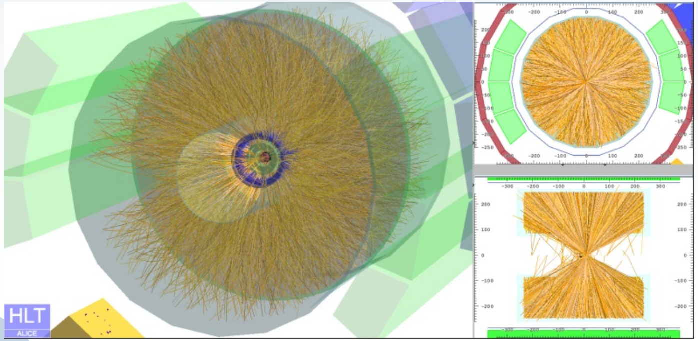
Asynchronous Side Tasks

- **Solution (Task D): Split processing in synchronous and asynchronous part.**
  - Frameworks spawns an asynchronous thread.
  - It provides simple interface to the component for offloading asynchronous tasks.
  - It handles the synchronization.



Tracking on GPUs

- **Solution** **(Task C)**: **Use GPUs for fast track reconstruction.**
  - GPU tracker was successfully used in ALICE run 1 on 64 GPU-enabled nodes.
    - D. Rohr: **“ALICE TPC Online Tracker on GPUs for Heavy-Ion Events”**, in *13th International Workshop on Cellular Nanoscale Networks and their Applications*, pp. 298–303 [2012].
    - D. Rohr, S. Gorbunov, A. Szostak, M. Kretz, T. Kollegger, T. Breitner, T. Alt: **“ALICE HLT TPC Tracking of Pb-Pb Events on GPUs”**, *Journal of Physics: Conference Series*, vol. 396 , no. 1 : p. 12044 [2012].
    - S. Gorbunov, D. Rohr, K. Aamodt, T. Alt, H. Appelsh, A. Arend, M. Bach, B. Becker, T. Breitner, et al.: **“ALICE HLT High Speed Tracking on GPU”**, *IEEE Transactions on Nuclear Science*, vol. 58 , no. 4 [2011].
  - GPU Tracker based on NVIDIA CUDA → Vendor lock
  - New GPUs with new features available in the meantime → Possible improvements to GPU tracking



Event reconstructed by GPU tracker during ALICE run 1

Common Tracker Source Code

- CPU and GPU tracker (in CUDA) share common source files.
- Specialist wrappers for CPU and GPU exist, that include these common files.

common.cpp:

```
__DECL FitTrack(int n) {
….
}
```

cpu_wrapper.cpp:

```
#define __DECL void
#include ``common.cpp``

void FitTracks() {
  for (int i = 0;i < nTr;i++) {
    FitTrack(n);
  }
}
```

cuda_wrapper.cpp:

```
#define __DECL __device void
#include ``common.cpp``

__global void FitTracksGPU() {
  FitTrack(threadIdx.x);
}

void FitTracks() {
  FitTracksGPU<<<nTr>>>();
}
```

→ **Same source code for CPU and GPU version**

- The macros are used for API-specific keywords only.
- The fraction of common source code is above 90%.

CUDA, OpenCL and C++

- **For the first GPU tracker implementation, CUDA was the only option.**
  - CUDA was the only GPU framework supporting C++, and AliRoot needs C++.
  - OpenCL (currently the main alternative) was new, only early beta SDKs available.
- **Now, AMD provides a stable OpenCL SDK with C++ kernel language extensions.**
  - Adaption possible.
  - OpenCL and CUDA very similar.

- **We can easily add other versions, by constructing appropriate wrappers.**
  - Wrapper can be adapted from CUDA wrapper by replacing language specific keywords, e.g. __global → __kernel.

- **The problem is: Our OpenCL code uses AMD’s C++ extensions and can thus run on AMD GPUs only. However:**
  - We can still use CUDA on NVIDIA GPUs.
  - New OpenCL standard may specify C++ kernel language.s

Adaptation for OpenCL

FIAS Frankfurt Institute for Advanced Studies

- **Main Problem for OpenCL adaptation:**
  - Pointers in OpenCL have address type qualifiers: global memory, private memory, etc.

- **Tracker objects can reside in all address spaces. (Important for optimizations!)**
  - Address type qualifier in function parameters can be treated with templates:
    - void foo(int* bar); → template <class T> void foo(T bar);
  - But: this cannot work for the return type, because overloaded functions cannot be distinguished by return type only.
    - template <class T> T foo(); DOES NOT WORK

  - Two solution:
    - Assign an address type qualifier to objects themselves.
    - Use generic address space specified by OpenCL 2.0

Performance on New GPUs

- **Important new GPU feature relevant for GPU tracker:**
  - GPUs can run multiple different kernels in parallel.
  - This can improve GPU utilization. Preliminary tests show 15% improvement already.

- **GPU tracking time on central PbPb event.**

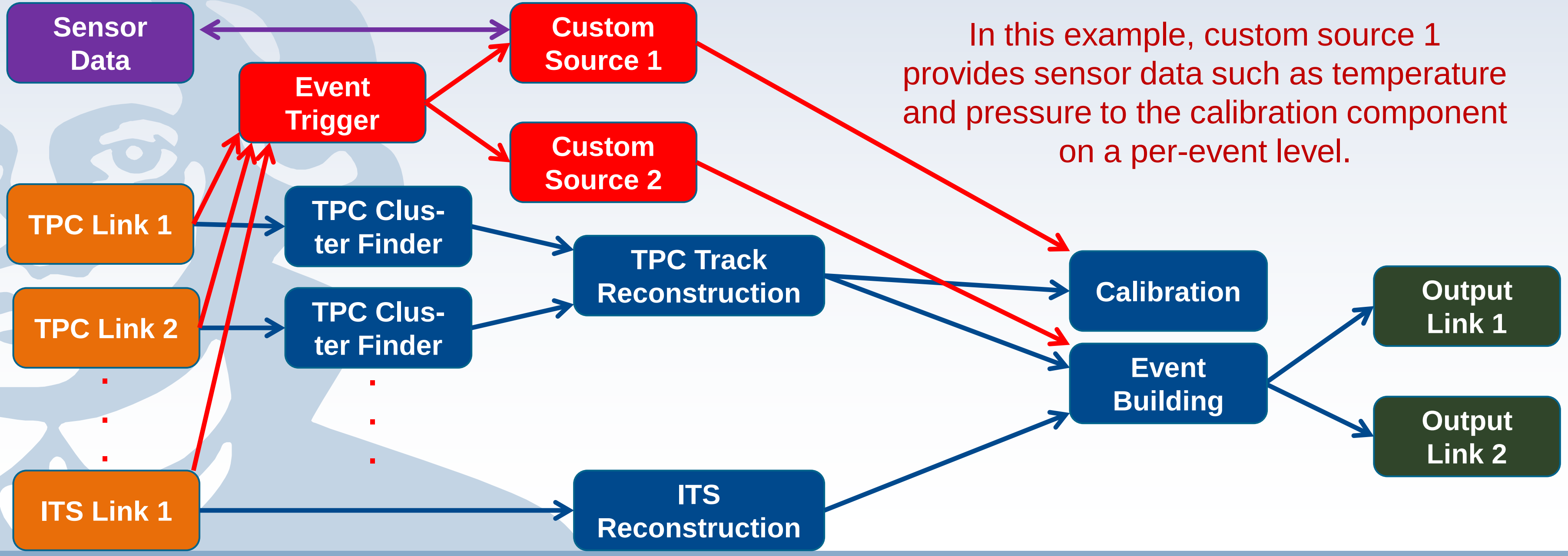
NVIDIA GTX480 (Fermi)	448 shader, 1215 MHz	174 ms	(used in the old HLT)
NVIDIA GTX780 (Kepler)	2304 shader, 863 MHz	155 ms	
NVIDIA Titan (Kepler)	2688 shader, 837 MHz	146 ms	
AMD S9000 (Tahiti)	**1792 shader, 900 MHz**	**145 ms**	(used in the new HLT)
NVIDIA GTX980 (Maxwell)	2048 shader, 1126 MHz	120 ms	

- **With both NVIDIA and AMD as possible vendors, we are no longer vendor-locked!**

- **New GPUs with more shaders not optimally used yet. We assume a speed benefit of up to 30% by further tuning the tracker for the new GPU chips.**

Custom Data Sources

- An intermediate component (Event Trigger) scans for events, by receiving 0-payload packages.
- It can trigger the input of custom data sources. (Solution to Task B)
- Allows synchronous input of custom sources, by using current event number.

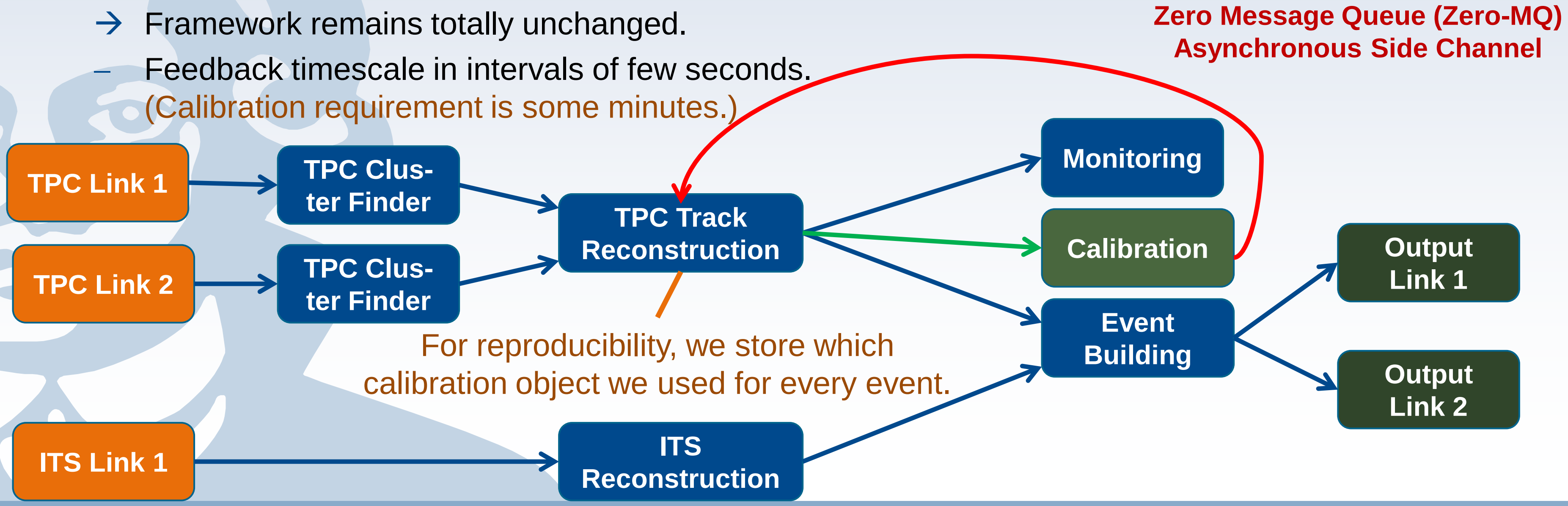


In this example, custom source 1 provides sensor data such as temperature and pressure to the calibration component on a per-event level.

Asynchronous Side Channel

FIAS Frankfurt Institute for Advanced Studies

- **We use the asynchronous tasks introduced as solution to “Task D”, to create an asynchronous side queue for feeding back data. (Solution to Task A.)**
  - Data transport via Zero-MQ.
  - Loop channel is asynchronous on component level.
  - → Framework remains totally unchanged.
  - Feedback timescale in intervals of few seconds. (Calibration requirement is some minutes.)



Summary

- **We have identified four requirements in the HLT framework needed for online calibration.**
- **We have presented solutions to these requirements.**
- **All solutions are available on component-level.**

  → No changes to the HLT framework needed.

  – Asynchronous data channel via ZeroMQ (implementation ongoing)
  – Custom source input in framework (implemented)
  – Fast tracking on GPU with OpenCL (implemented, further tuning possible)
  – Asynchronous tasks inside component (implemented)

- **In parallel, development of the calibration component itself is work in progress by HLT and Offline groups.**