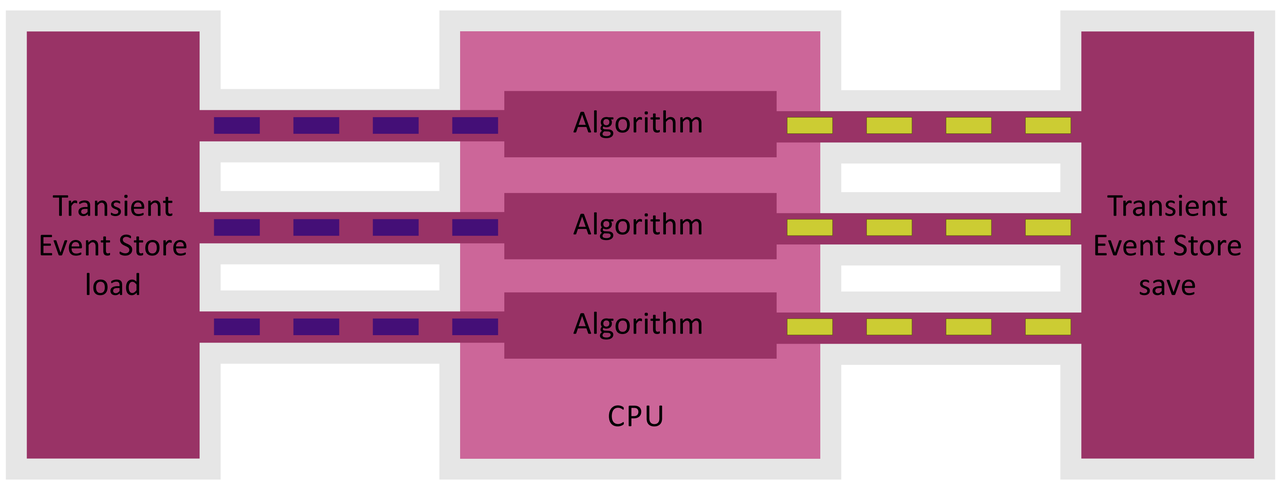
# A GPU OFFLOADING MECHANISM FOR LHCb

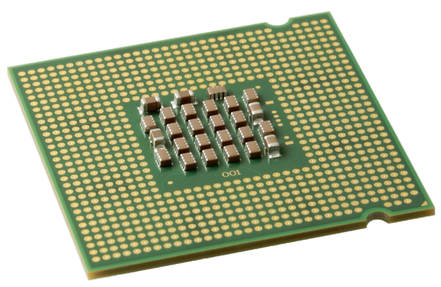
*Alexey Badalov, Daniel Hugo Campora Perez, Alexander Zvyagin, Niko Neufeld, Xavier Vilasis Cardona*
*alexey.badalov@cern.ch, dcampora@cern.ch*
*La Salle — Ramon Llull University, CERN*

## Background

The LHCb software infrastructure is built around a flexible, extensible, modular framework named Gaudi. This framework is used in the context of the Online and Offline software, for all the event processing applications of LHCb, including the data taking software, the full event reconstruction chain and analysis software.

The hardware market has changed since the time of Gaudi's conception at the turn of the century. There is now readily available hardware for massively parallel computation, such as consumer graphics processing units, NVIDIA Tesla cards, and Intel Xeon/Phi coprocessors. Enabling the use of these in the Gaudi framework has the potential to speed up and enhance Physics selection in HEP experiments.



## Gaudi framework

The Gaudi framework was conceived for a sequential processing paradigm, where all individual events of event reconstruction follow a process chain with dependencies along the computation stages.

The execution chain of an event in LHCb incurs data and control flow dependencies. Some algorithms require execution of others, creating RAW (Read After Write) dependencies in the TES (Transient Event Store). Since the nature of the problem is to discard uninteresting events, the chain may terminate, thus also creating control flow dependencies.

An algorithm running on Gaudi can choose to take advantage of multiple processor cores independently of the framework, but it can process only a single piece of data at a time. It is also possible to run several instances of the framework in parallel. For LHCb, this solution has been shown to work well up to 32 cores with diminishing returns. It is expected to scale up only slightly farther. Given the small size of individual data sets at LHCb (about 60 KB raw event size), an individual algorithm running on Gaudi cannot properly take advantage of massively parallel software.

## Offloading mechanism

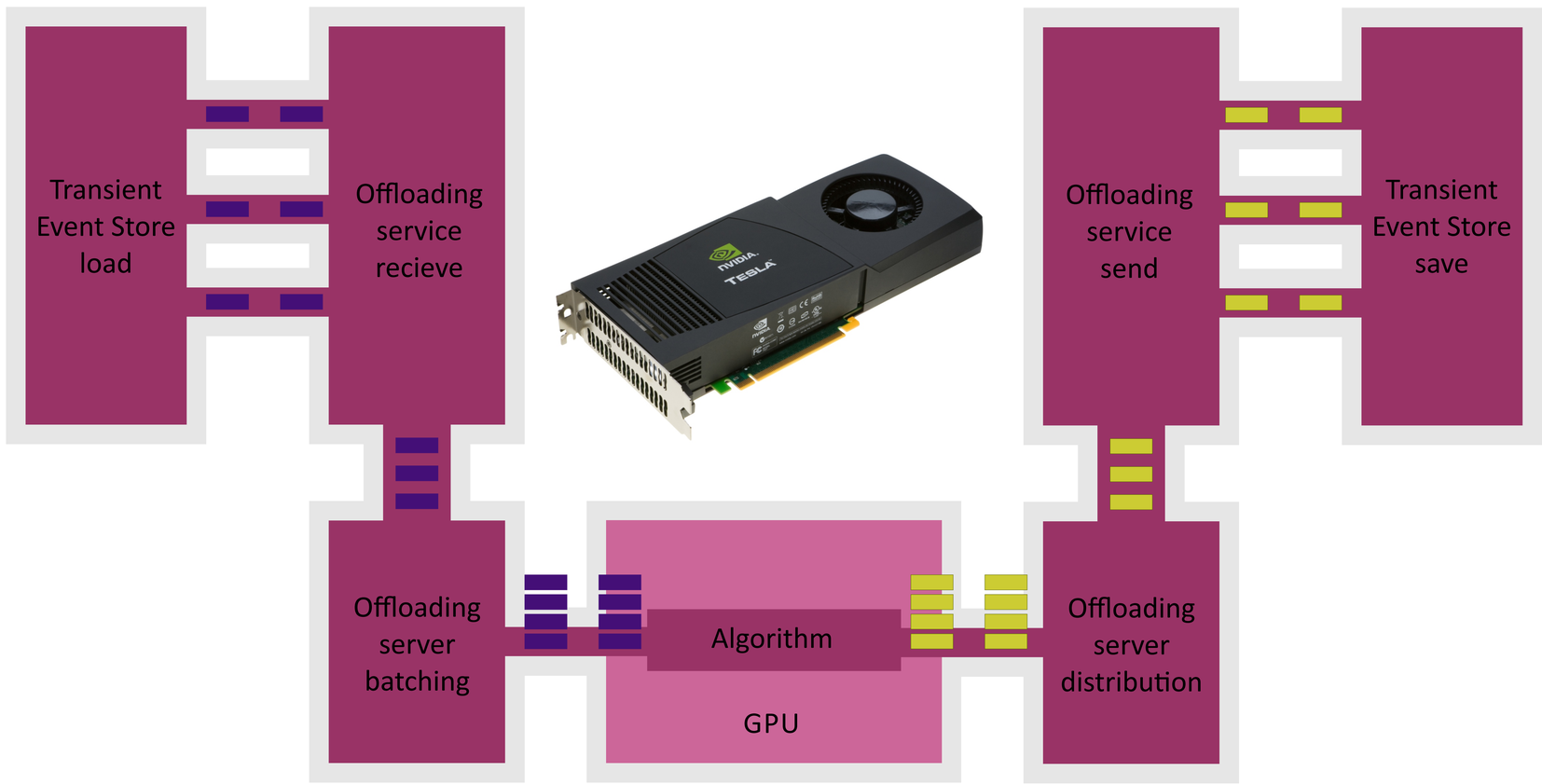
We address Gaudi's limitation by creating a server process tied to the massively parallel computation unit. This process receives data from multiple Gaudi instances, processes them in large batches, and then distributes the results back to senders. The instances can be located on the same node as the server process or on different ones.

Gaudi instances communicate with the server through a Gaudi service. Each instance submits data in a format required by the GPU algorithm, and then waits for the result.

Our preliminary tests of this system on VELO reconstruction on the GPU show promising results, albeit with additional overhead that has to be amortised over multiple events.

Our test requires *receive* and *send* stages, in which the AoS (Array of Structures) data from TES is converted to a coprocessor-friendly SoA format and back. The converted data is sent to the GPU kernel.

$$TS(AoS) \xrightarrow{prepare} SoA \xrightarrow{kernel\_execute} SoA(result) \xrightarrow{store} TS(AoS)$$




## Preliminary results

A GPU Offloading engine has been developed that allows execution of code on a coprocessor. A working example outperforms the sequential algorithm in execution time, and a GPU Scheduler combined with Gaudi-MultiThreaded is a good prospect to reduce the transport time overhead.