The core trigger software framework of the ATLAS experiment

T. Bold – AGH UST Krakow

on behalf of ATLAS collaboration

2013-10-15

Plan

- The Trigger Core Software Framework
  - Configuration system
  - Decision evaluation system
- Unique solutions
  - Fine algorithms granularity
  - Optimizations
- Future
  - Adapting to new hardware
  - Heavy computational tasks

The architecture of the ATLAS Trigger/ DAQ system

Requirements:

L1: work wiht LHC frequency and hard time budget

HLT:work with limited CPU and network resources, with soft limited latency

L1 Design:

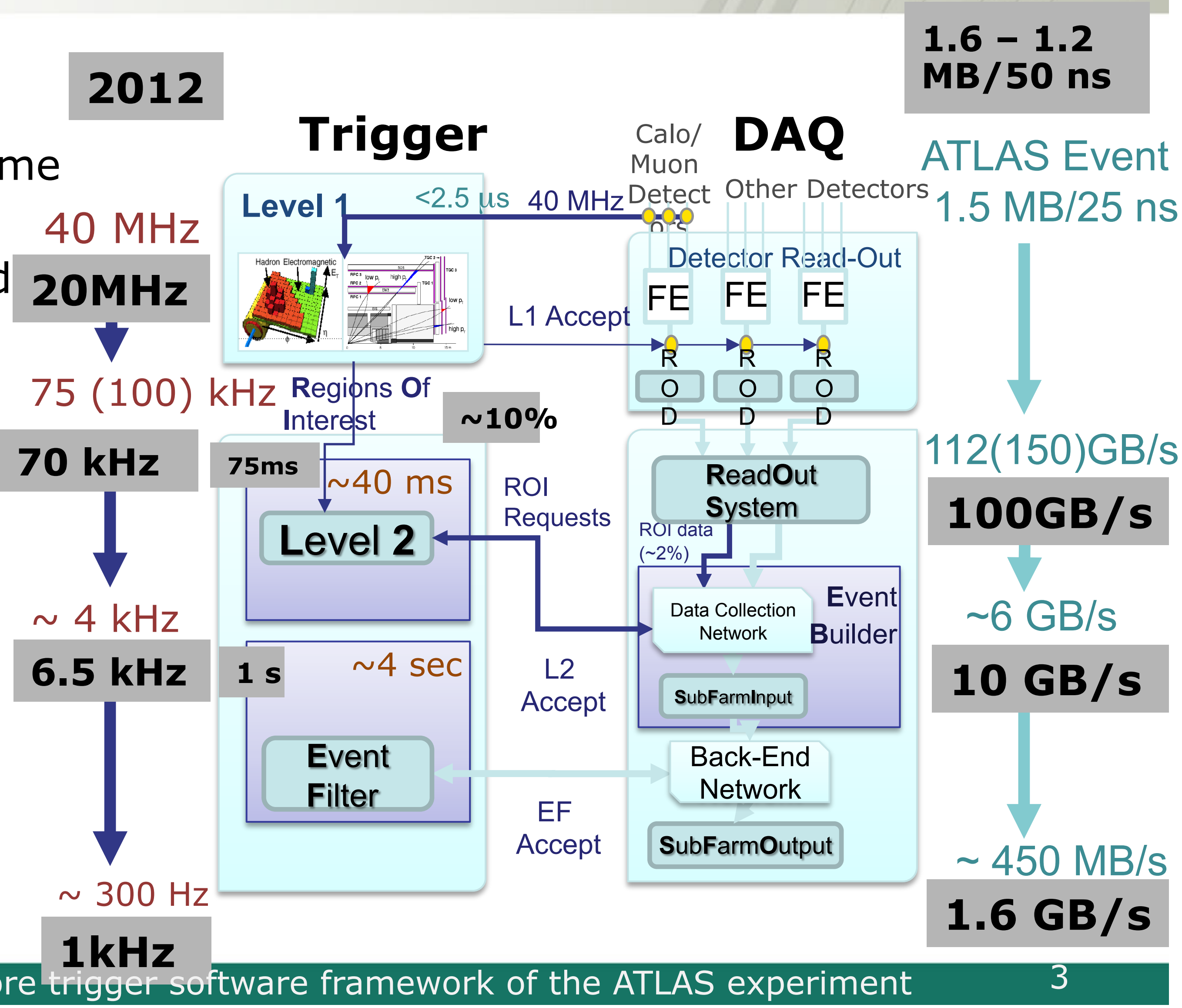
- Custom hardware

HLT Design:

-Reconstruction in **R**egion **of I**nterest

-Data on demand

-Early rejection



HLT Key concepts

- The L1 trigger locates RoIs (μ,jet,e/γ,τ,E_T^{miss}) with approximate p_T measurement
  → by counting them, up to 256 L1 triggers are generated
- Events accepted by L1 are passed over to the L2
  - Each **L1 trigger is a seed** for the HLT chain(s)
    → Each **chain is organized in steps,** at any step it can be rejected
    → Each step: execution of **Feature Extraction** (**FEX**) and **hypothesis testing** algorithm (**Hypo**)
    → At least one chain surviving till the last step → event accepted
  - The algorithms at **L2** are highly **specialized** and **fast**. **Detector data** is **delivered on demand** (**~2%** of it is used).
- Event accepted by L2 is delivered to EF
  - The organization is as in L2. **Offline** algorithms are **adapted** to perform reconstruction in RoIs or on full event data.
- Navigation between algorithms is organized as a directed-graph.

The ATLAS Trigger configuration system

- The Trigger DB – central source of the trigger configuration
  - DB & tooling designed to never overwrite any setting – only reuse

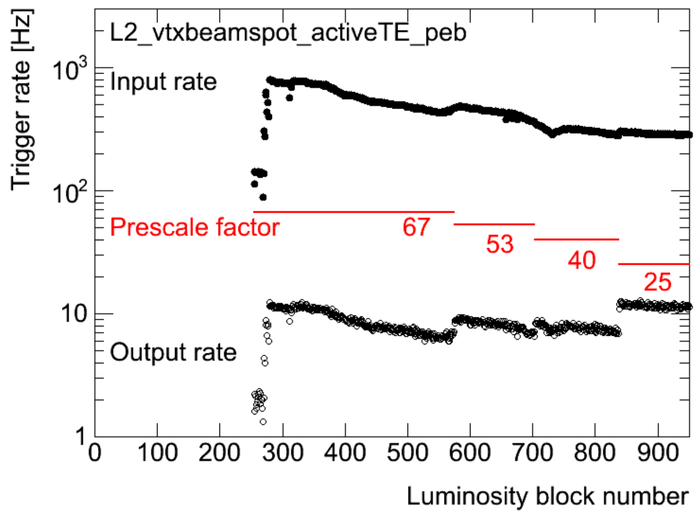
So far ~1500 trigger configurations stored

- Content
  - L1 system configurations - essentially the firmware settings
  - Bunch groups
  - Trigger chains definitions ~1k
  - Algorithm properties ~4k
  - Prescale values ~1k for HLT, 256 for L1
    The prescale values are organized in sets and are managed independently

About 10k prescale value sets for L1 and 8k for HLT were used so far

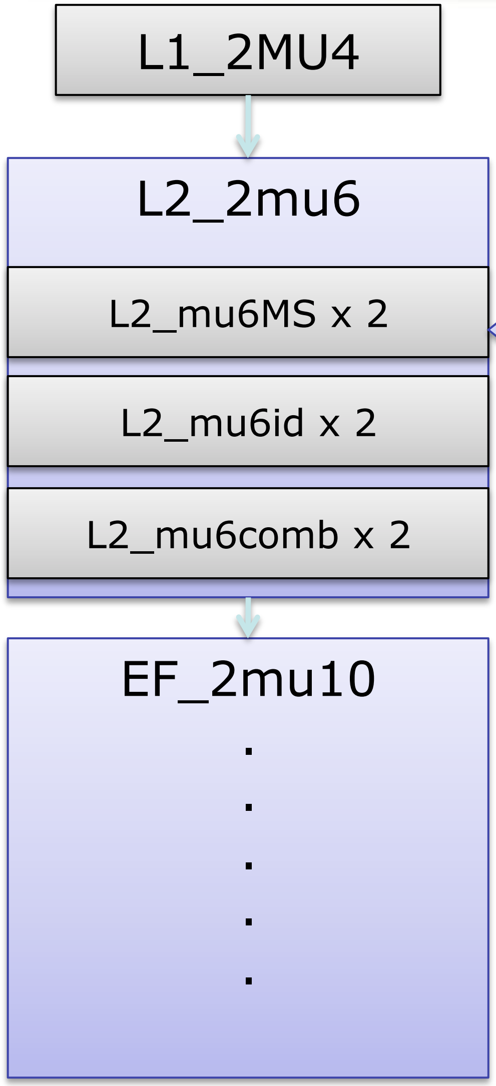
Use of the trigger configuration

- Start of run: bulk of the data is read i.e. chain definitions
  - With database proxy to scale for the HLT farm
- During the run: vital updates
  - The routine operation involves settings known in advance
    - In order to adapt to the changing luminosity the trigger rates are tuned
  - On-the-spot changes are also possible
    - Test runs, masking problems



- After the run: prescale values and L1/L2/EF seeding relations saved as conditions meta-data (conditions DB and derived data formats)

The HLT Steering -from the trigger chain to algorithms

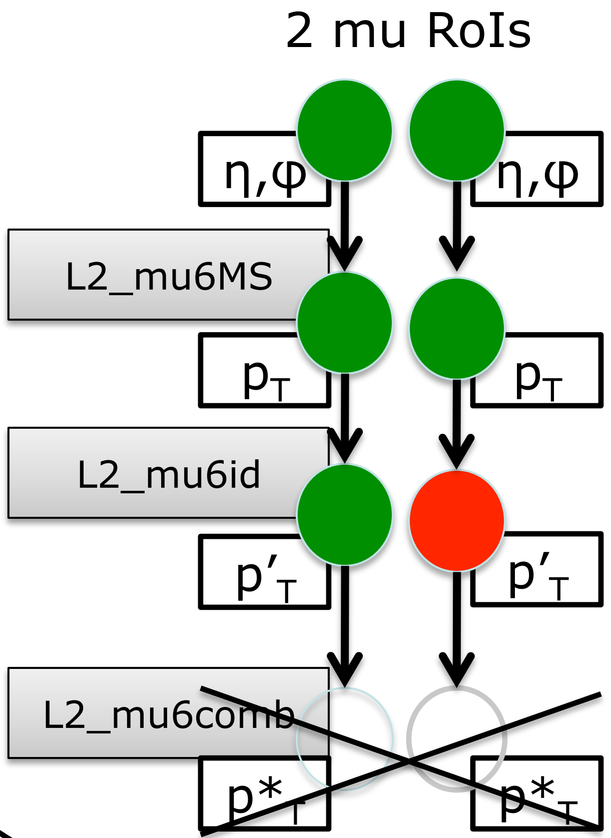


HLT trigger chain is a requirement of **presence** of abstract objects and their multiplicity

L2_mu6MS means that the muon track of p_T>6GeV has been found in Muon Spectrometer

Failing the requirement→ End of evaluation → Resources saving!

Algorithms grouped in sequences. FEX algorithms are reused in many of them.



Resources saving via caching

- Normal configuration consists of **~1k** chains, **~1k** sequences and similar number of algorithms
  - Each sequence of algorithms is processed once per event
  - The detector data are transferred once (τ/jet/*e*/γ algorithms share the same data)

Typical memoization techniques

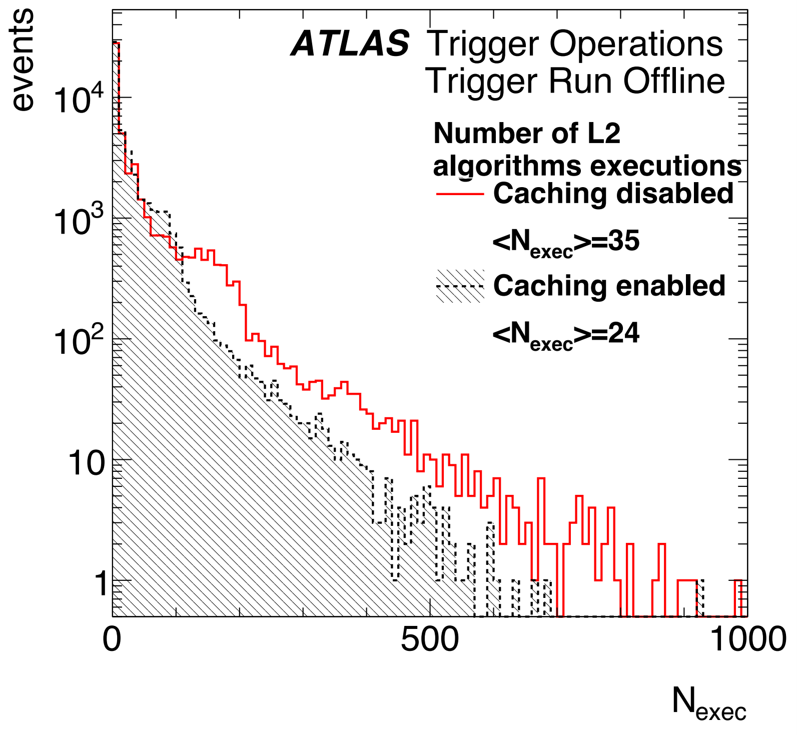
- Challenge: sparing the execution of a **FEX** on the **RoIs** when it was already done maintaining the flexibility of the algorithm
  - Simple memoization technique is not aplicable because the input is unknown (depens on the logic of an algorithm)

Deep memoization

- Solution:
  - During the first execution **capture** input data requests, obtained objects and results
  - At the next execution **reply requests** before algorithm runs
  - if objects are the same **reuse** the results w/o actual execution

The effect of caching at the level of RoI

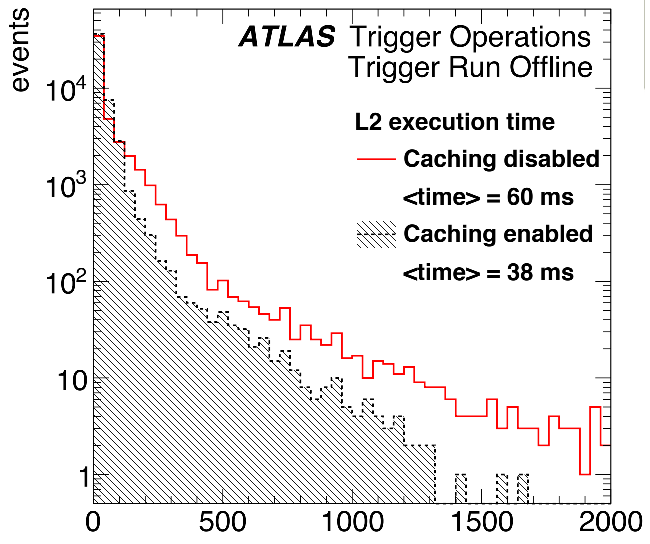
When running the trigger offline, the caching mechanism can be disabled to measure its impact.

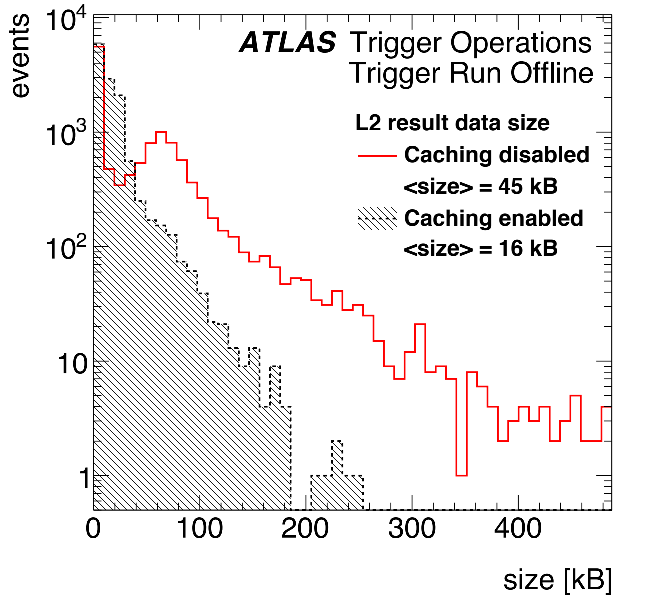


← Number of executions is reduced by ~30% and so the time to accept/reject event →

and the size of the L2 result data (~70%) →saving disk and network traffic

Enhanced minimum-bias data





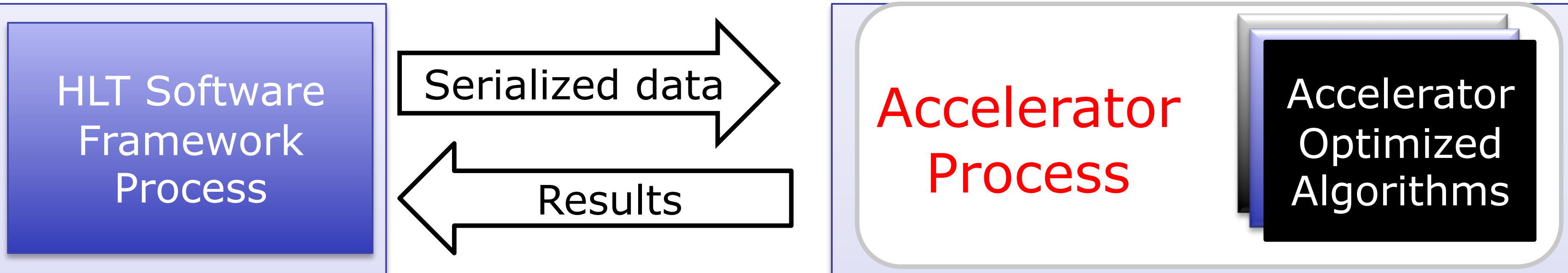
Look ahead

- The HLT algorithms execution is sequential
  - As of now the concurrency in single PC box was achieved by replicating entire HLT processes (Run 1)→ has limits, forking single process planned (Run 2)
- Many-core CPUs can only be fully used by parallel code
  - Memory size does not scale as fast as the # cores
  - Network and process management becoming hard
- The fine granularity of the HLT algorithms (due to the RoIs & early rejection) suggests intra-event concurrency (Run 3)
  - Also the event level parallelism is considered → The R&D are ongoing
- Studies in this area are important in the context of increased data volume with LHC luminosity

Going beyond the HLT algorithm granularity

- Most of the CPU resources are spent on few specialized tasks https://cds.cern.ch/record/1602918/ (data preparation, tracking, vertexing, clustering)
- An approach is to accelerate this **few tasks** and gain the necessary speedup
- A demonstration project has been launched to accelerate **HLT ID data preparation**
  - Study feasibility of framework-friendly integration of the acceleration
  - Decouple from the hardware architecture (CPU cores, GPGPUs)
- Addressed through
  - Single interface (based on IPC) which integrates the Accelerator as an external process, unique for each architecture.

Accelerator dataflow

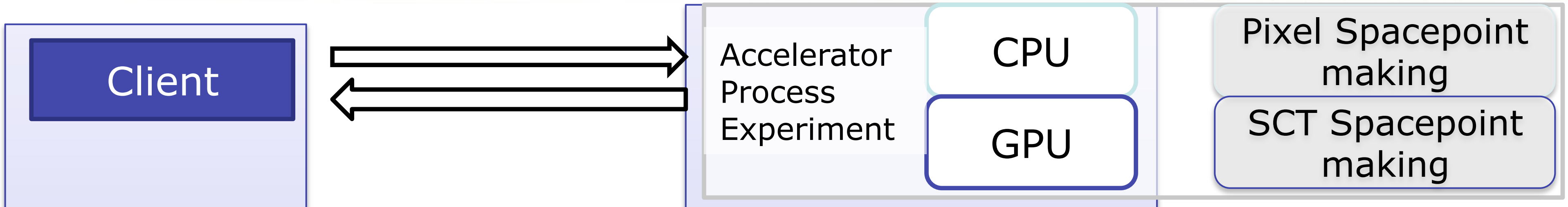

HLT Software Framework Process
Serialized data
Results
Accelerator Process
Accelerator Optimized Algorithms

- Instead of processing the data sends it to the Accelerator Process
- Non-blocking operation possible. Framework may perform other task meanwhile.

Data is serialized and sent to the Accelerator Process through IPC

- Handles resources (CPU,GPU or Xeon-Phi)
- Contains optimized algorithms for available hardware
- Uses optimal data structures
- Manages communication with the framework process(es)
- May be on the same host with Framework process

Test implementation



Reads and sends 200 Monte-Carlo events in cyclic manner to accelerator process to simulate framework behavior.

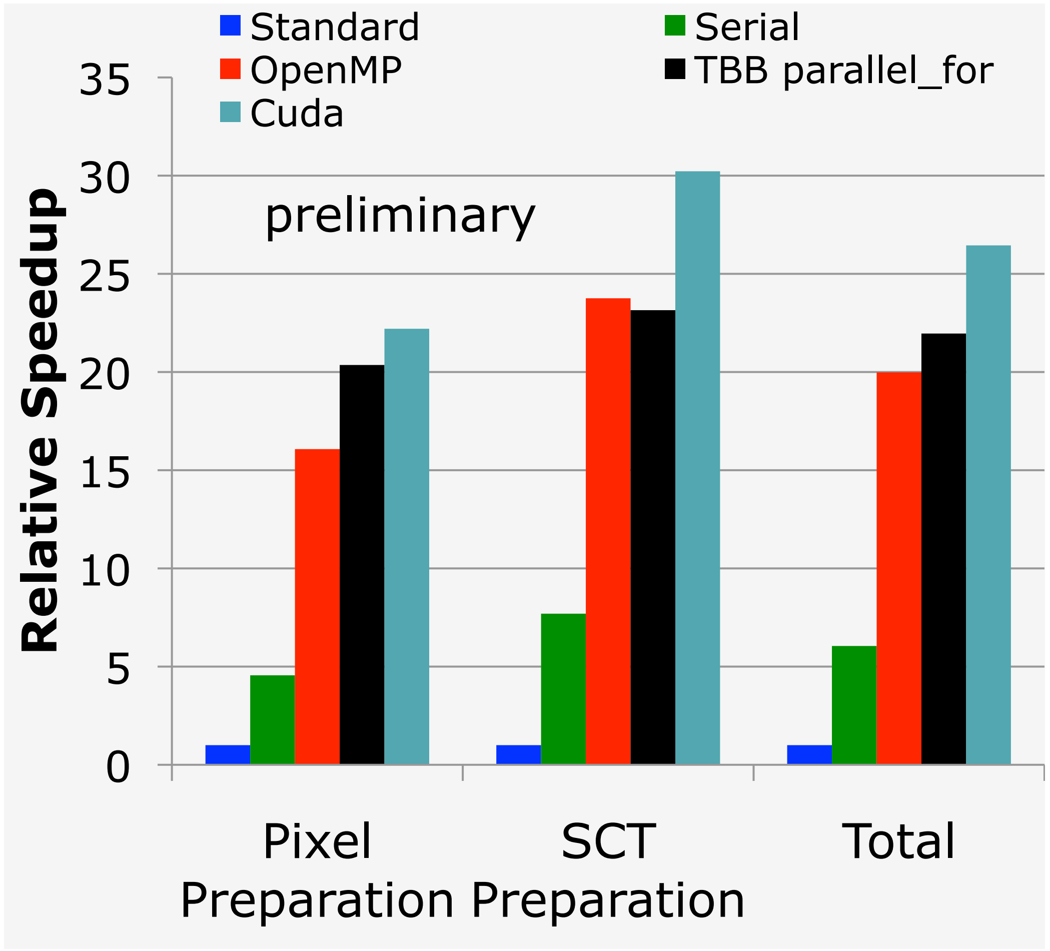
Inter-Process-Communication is accomplished using **yampl** library.
It abstracts underlying IPC mechanism, providing fast communication and supporting network transfers.

4 different modules are implemented , one using CUDA for NVidia GPUs and 3 for CPUs.
CPU implementations are serial, openmp, TBB parallel_for.

Two types of work are implemented, space point making algorithms for ATLAS Pixel and SCT detectors. Algorithms decode detector data and apply clusterization and centroid finding.

Results

- This approach facilitates
  - Utilization of resources until framework adapts to new hardware architectures
  - Testing of new algorithms
- Significant speedup can easily be achieved even adding the IPC overhead



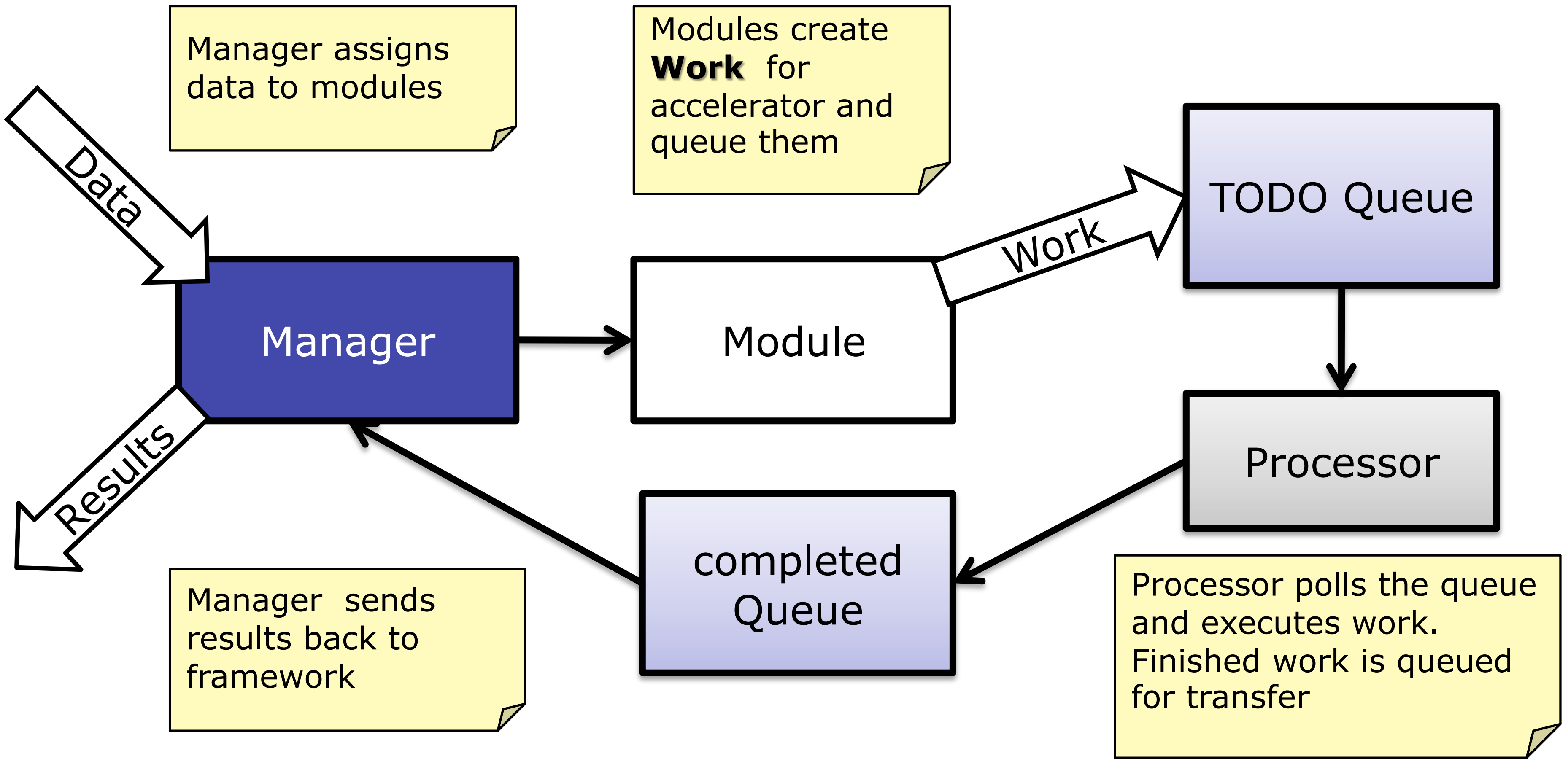
Intel(R) Core(TM) i7-2760QM CPU @ 2.40GHz, 4 cores, 8 threads
NVidia C2050 (Tesla card, Fermi GPU, 14 SM)

Summary

- The HLT core framework has been instrumental for the implementation of the ATLAS physics programme
- Its flexibility allowed for optimal use of limited CPU and network resources
- R&D starts in order to prepare for new hardware architectures and demanding data taking conditions

Backup

Accelerator Process architecture


Manager assigns data to modules
Modules create **Work** for accelerator and queue them
Data
Manager
Module
Work
TODO Queue
Processor
Results
completed Queue
Manager sends results back to framework
Processor polls the queue and executes work. Finished work is queued for transfer