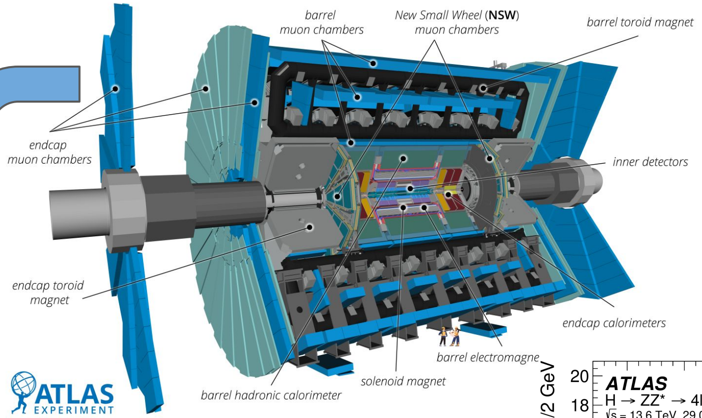
Accelerating HEP Software

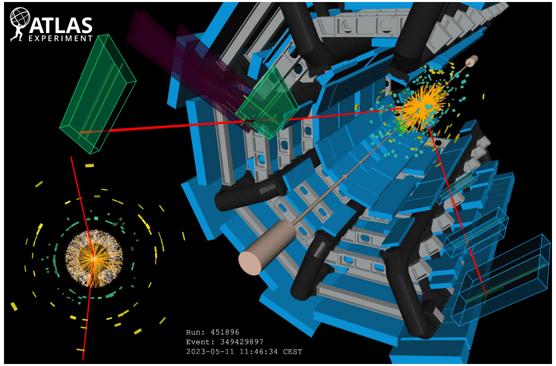
Attila Krasznahorkay

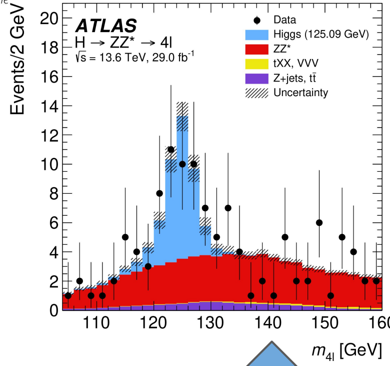
trying to sample / expose the work of a lot of people…

HEP Software Basics

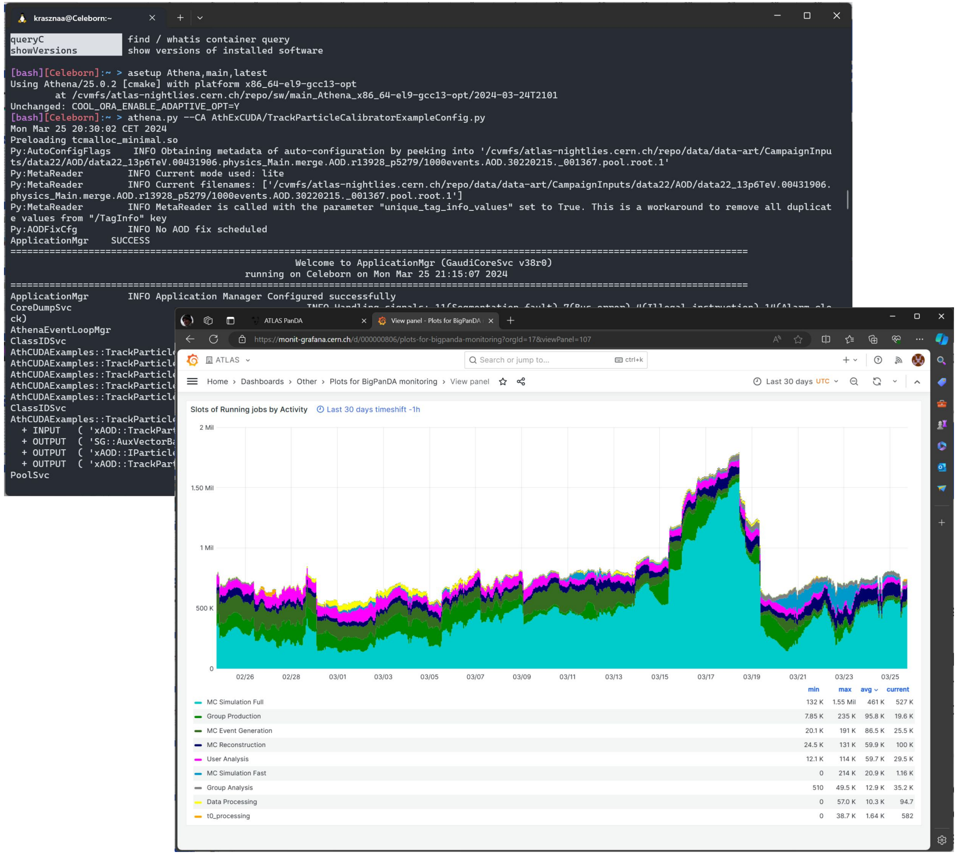
- "Software" is used practically everywhere in HEP
  - But here I'll only focus on software used in "data processing". Software used for controlling the accelerator, detector hardware, etc. is a different beast entirely.
- The goal is to "do physics"
  - Fast and nicely written software is the means, not the actual goal of the endeavour.







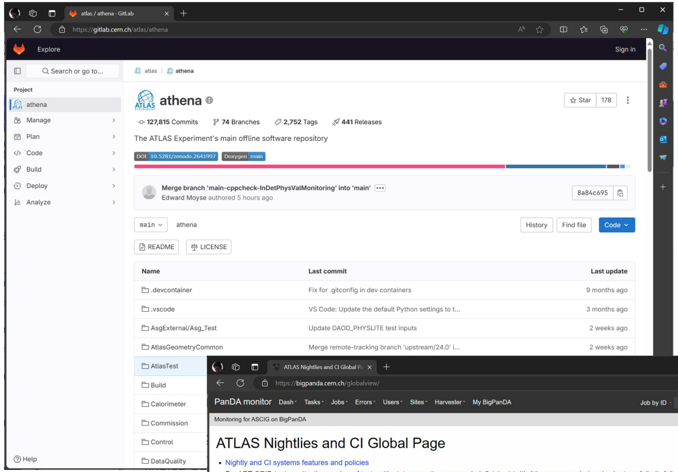
Software ↔ Computing

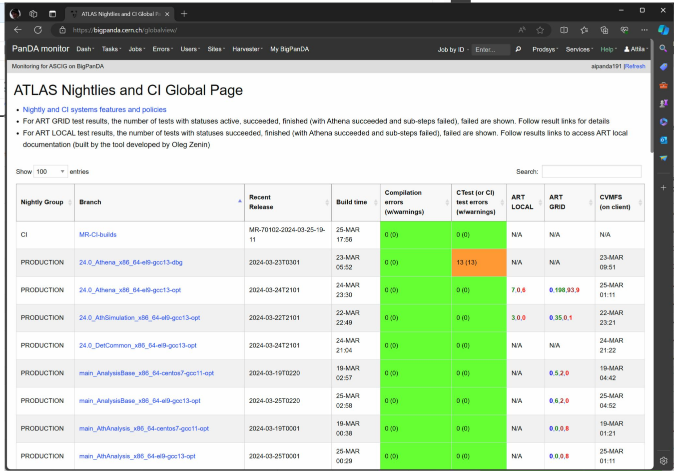


- Even in data processing, we have 2 distinct areas
  - “Software” covers all the software written to perform the data transformations needed “for physics”.
  - “Computing” covers all the software created to allow the first category to execute on up to millions of CPU cores in parallel.
- Here I will only cover “software”
  - Even though using accelerators in job/network optimization, and even just making batch systems “accelerator aware” is a very important thing on our plates.

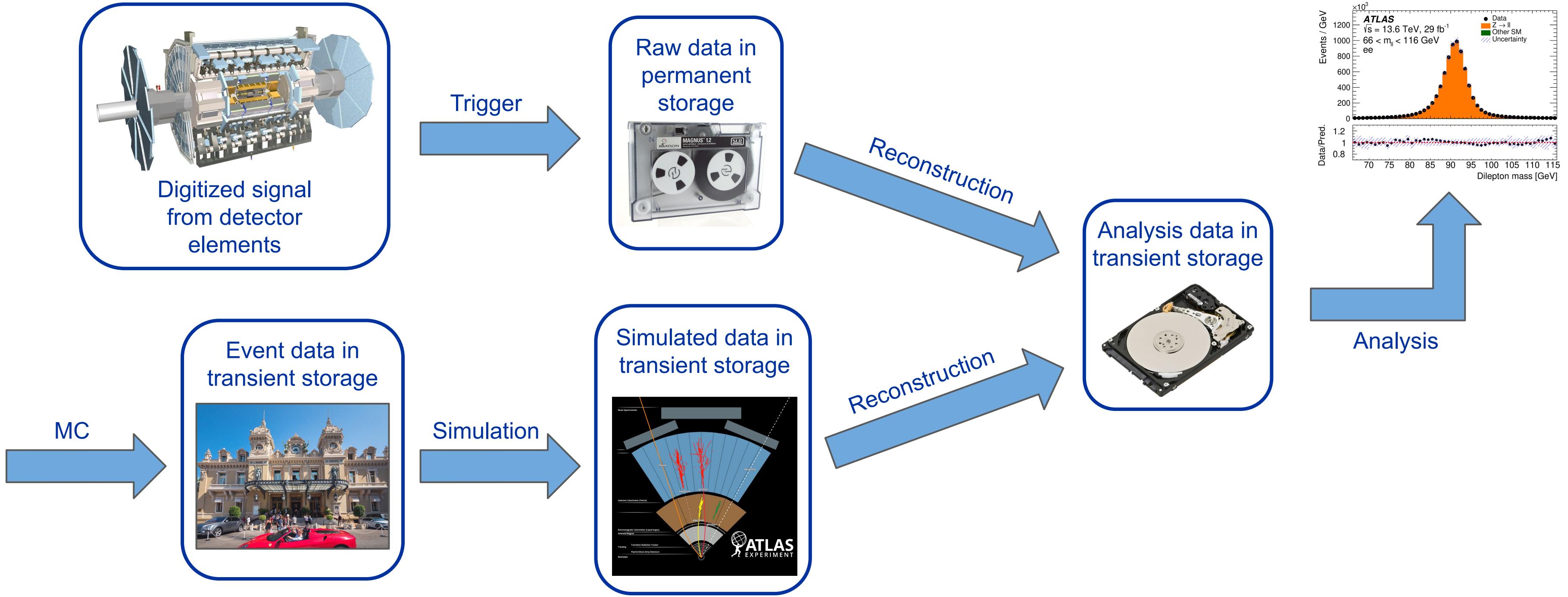
Development, Build and Deployment

- Interpreting / processing the data coming from the LHC detectors takes many millions of C++ / Python code
    - We use very customized build systems to allow developers to deal with a small portion of the software at any given time, testing its integration with the rest of the software.
    - This requires highly curated, many GB development environments to be made available around the world.
- The distribution uses [CVMFS]
    - Which needs to be able to house the full SDKs (making licensing a challenge…)

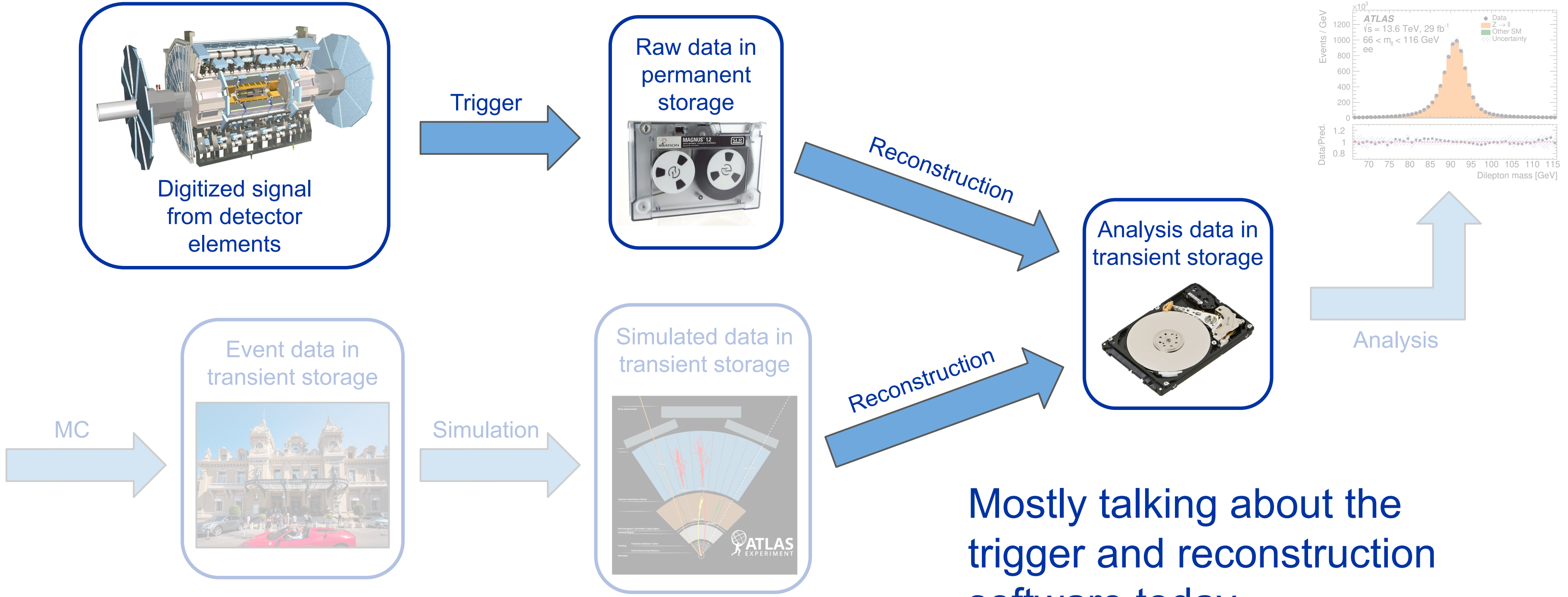




HEP Data Processing (1)

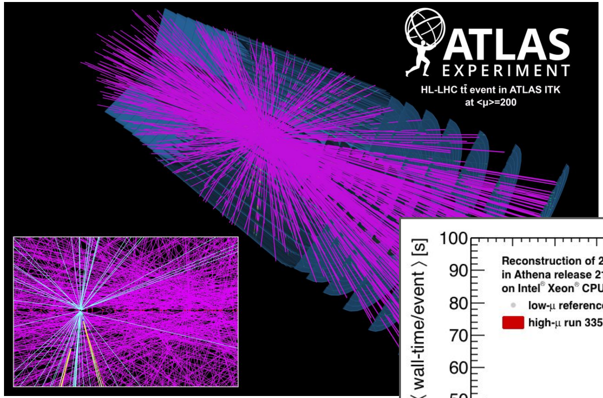


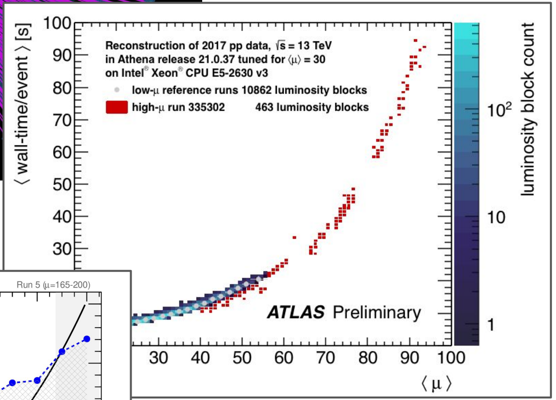
HEP Data Processing (2)

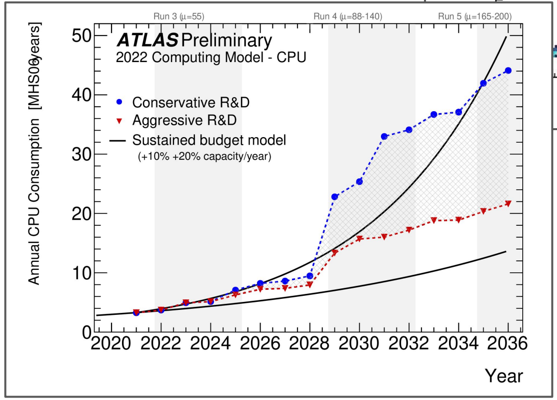


Mostly talking about the trigger and reconstruction software today.

Trigger / Reconstruction







- The goal of this software is to figure out what particles, with what exact properties, left signals in the detector
  - A little similar to “finding the cats” in a CCD camera’s recorded data 😼
  - In trillions of >100 megapixel, “3D camera” pictures
- “Classical” algorithms do this very well in the ongoing LHC data taking. (Though ATLAS is the only one **only** using classical algorithms at this point.)
  - However during the High-Luminosity LHC era these current algorithms would need too much CPU power.

Things To Do Right: Event Data Model

- Our software uses “rich” data
  - Many, well defined properties, in **very** jagged arrays
- Our 20+ year old existing software tends to manage these in ways not suited for accelerators (AoS vs. SoA)
- We need to move this data with a high frequency between H↔D
- Must be manageable to implement/maintain the code that describes all our data types
  - Note that a single “type” usually requires multiple C++ classes to describe it

```cpp
#ifndef DataFormats_PortableTestObjects_interface_TestSoA_h
#define DataFormats_PortableTestObjects_interface_TestSoA_h

#include <array>

#include <Eigen/Core>
#include <Eigen/Dense>

#include "DataFormats/Common/interface/StdArray.h"
#include "DataFormats/SoATemplate/interface/SoACommon.h"
#include "DataFormats/SoATemplate/interface/SoALayout.h"
#include "DataFormats/SoATemplate/interface/SoAView.h"

namespace portabletest {

  // the type aliases are needed because commas confuse macros
  using Array = edm::StdArray<short, 4>;
  //using Array = std::array<short, 4>;
  using Matrix = Eigen::Matrix<double, 3, 6>;

  // SoA layout with x, y, z, id fields
  GENERATE_SOA_LAYOUT(TestSoALayout,
                      // columns: one value per element
                      SOA_COLUMN(double, x),
                      SOA_COLUMN(double, y),
                      SOA_COLUMN(double, z),
                      SOA_COLUMN(int32_t, id),
                      // scalars: one value for the whole structure
                      SOA_SCALAR(double, r),
                      // column of arrays: one fixed-size array per element
                      SOA_COLUMN(Array, flags),
                      // Eigen columns: each matrix element is stored in a separate column
                      SOA_EIGEN_COLUMN(Matrix, m))

  using TestSoA = TestSoALayout<>;
```

```cpp
        /// Global "average" of something (non-const)
        VECMEM_HOST_AND_DEVICE
        auto& average() { return BASE::template get<3>(); }
        /// Global "average" of something (const)
        VECMEM_HOST_AND_DEVICE
        const auto& average() const { return BASE::template get<3>(); }

        /// "Indices" of something (non-const)
        VECMEM_HOST_AND_DEVICE
        auto& indices() { return BASE::template get<4>(); }
        /// "Indices" of something (const)
        VECMEM_HOST_AND_DEVICE
        const auto& indices() const { return BASE::template get<4>(); }

        /// "Index" of something (non-const)
        VECMEM_HOST_AND_DEVICE
        auto& index() { return BASE::template get<5>(); }
        /// "Index" of something (const)
        VECMEM_HOST_AND_DEVICE
        const auto& index() const { return BASE::template get<5>(); }

    };  // class jagged_interface

    /// "Jagged" container for the tests
    using jagged_soa_container =
        edm::container<jagged_soa_interface, edm::type::scalar<int>,
                       edm::type::vector<float>, edm::type::jagged_vector<double>,
                       edm::type::scalar<float>, edm::type::jagged_vector<int>,
                       edm::type::vector<int> >;

    }  // namespace testing
    }  // namespace vecmem
```

C++ data layout portability

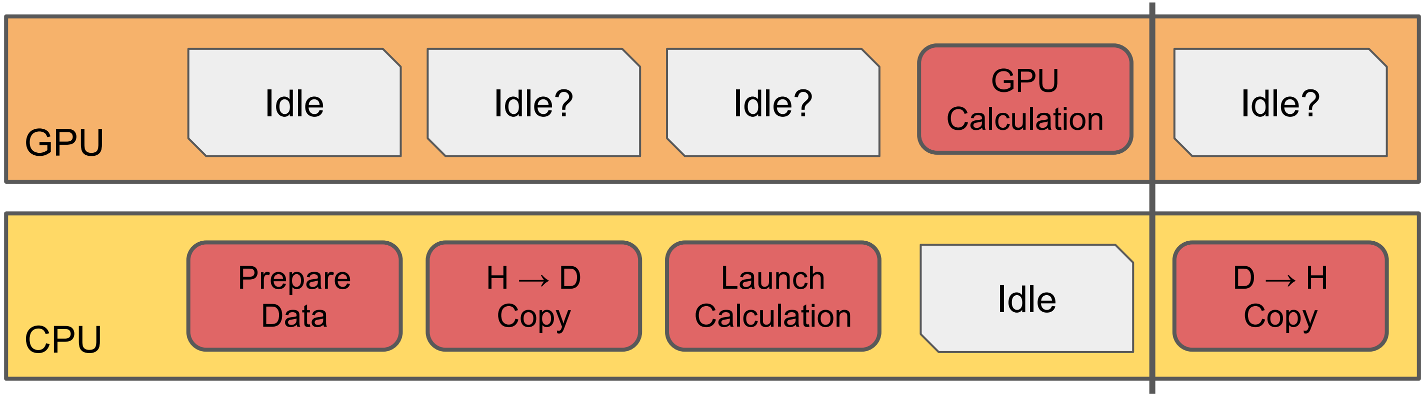
Thursday 30 Nov 2023, 16:00 → 17:30 Europe/Zurich

32/1-A24 (CERN)

Videoconference: C++ data layout portability workshop — Join

- 16:00 → 16:30 **Data layout: our reality** — 30m
  Speaker: Eric Cano (CERN)
  Data layout our reali...
- 16:30 → 16:50 **How reflection could help** — 20m
  Speaker: Bernhard Manfred Gruber (Technische Universitaet Dresden (DE))
  data layout and refle... | data layout and refle...
- 16:50 → 17:30 **Discussion** — 40m

Things To Do Right: Task Scheduling



- Not all aspects of our data processing are suited for accelerators
  - Any calculation not using an accelerator must be using the available CPU cores efficiently in parallel with whatever the GPU is doing, maximising the usage of all components
- A number of different solutions are used by the experiments currently
  - Built around TBB (for ATLAS and CMS) and ZeroMQ (for ALICE and LHCb)
- But new, hopefully better solutions seem to be on the horizon
  - Using Boost::fiber and/or C++ coroutines

Things To Do Right: Algorithms

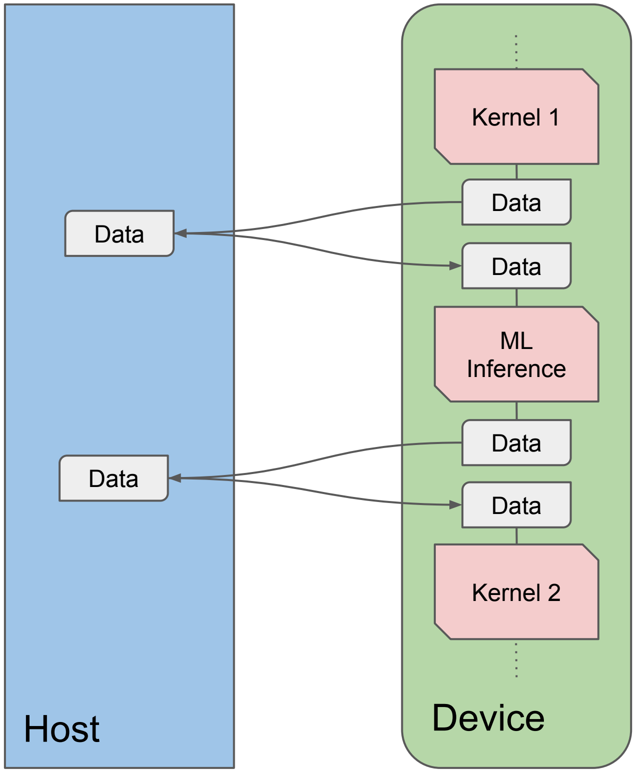
```cpp
namespace traccc::device {

/// Implementation of a FastSV algorithm with the following steps:
///   1) mix of stochastic and aggressive hooking
///   2) shortcutting
///
/// The implementation corresponds to an adapted versiion of Algorithm 3 of
/// the following paper:
/// https://www.sciencedirect.com/science/article/pii/S07437
///
///                        This array only gets updated at the
///                        to prevent race conditions.
/// @param[in] adjc        The number of adjacent cells
/// @param[in] adjv        Vector of adjacent cells
/// @param[in] tid         The thread index
/// @param[in] blckDim     The block size
/// @param[inout] f        array holding the parent cell ID for
/// iteration.
/// @param[inout] gf       array holding grandparent cell ID fr
/// iteration.
/// @param[in] barrier     A generic object for block-wide sync
///
template <typename barrier_t>
TRACCC_DEVICE void fast_sv_1(index_t* f, index_t* gf,
                             unsigned char adjc[MAX_CELLS_PE
                             index_t adjv[MAX_CELLS_PER_THRE
                             const index_t tid, const index_
                             barrier_t& barrier) {
    /*
     * The algorithm finishes if an iteration leaves the arr
     * This varible will be set if a change is made, and dic
     * loop is necessary.
     */
    bool gf_changed;

    do {
        /*
         * Reset the end-parameter to false, so we can set i
         * make a change to the gf array.
         */
        gf_changed = false;

        /*
         * The algorithm executes in a loop of three distinc
         * stages. In this first one, a mix of stochastic and aggressive
         * hooking, we examine adjacent cells and copy their grand parents
         * cluster ID if it is lower than ours, essentially merging the two
         * together.
         */
        for (index_t tst = 0; tst < MAX_CELLS_PER_THREAD; ++tst) {
            const index_t cid = tst * blckDim + tid;
```

```cpp
  __global__
    //  __launch_bounds__(256,4)
    void
    findClus(uint16_t const* __restrict__ id,           // module id of each pixel
             uint16_t const* __restrict__ x,            // local coordinates of each pixel
             uint16_t const* __restrict__ y,            //
             uint32_t const* __restrict__ moduleStart,  // index of the first pixel of each module
             uint32_t* __restrict__ nClustersInModule,  // output: number of clusters found in each module
             uint32_t* __restrict__ moduleId,           // output: module id of each module
             int32_t* __restrict__ clusterId,           // output: cluster id of each pixel
             int numElements) {
  if (blockIdx.x >= moduleStart[0])
    return;

  auto firstPixel = moduleStart[1 + blockIdx.x];
  auto thisModuleId = id[firstPixel];
  assert(thisModuleId < MaxNumModules);

#ifdef GPU_DEBUG
  if (thisModuleId % 100 == 1)
    if (threadIdx.x == 0)
      printf("start clusterizer for module %d in block %d\n", thisModuleId, blockIdx.x);
#endif

  auto first = firstPixel + threadIdx.x;

  // find the index of the first pixel not belonging to this module (or invalid)
  __shared__ int msize;
  msize = numElements;
  __syncthreads();

  // skip threads not associated to an existing pixel
  for (int i = first; i < numElements; i += blockDim.x) {
    if (id[i] == InvId)  // skip invalid pixels
      continue;
    if (id[i] != thisModuleId) {  // find the first pixel in a different module
      atomicMin(&msize, i);
      break;
    }
  }
```

- We have a wealth of experience with writing efficient data processing code in a single thread
  - Even our multi-threaded jobs are built from single threaded tasks that do independent, well defined operations
- We have been developing novel algorithms for many years now, that could take advantage of hundreds of thousands of threads
  - But I believe we are still only at the beginning of doing this correctly 🤔

Things To Do Right: ML Inference

- HEP software has used machine learning since a long-long time
  - At the same time, the innovations in industry do present very new opportunities for us
- ML/AI techniques already exist for doing very complicated operations in the trigger / reconstruction software of the experiments
- What is missing are efficient algorithm chains that would “stay on a device”, interleaving ML/AI inference and hand-written algorithmic steps



Next Generation Triggers

Next Generation HEP Triggers Proposal

CERN - European Organization for Nuclear Research

V 1.2_dist (20231212)

Executive Summary

The High Energy Physics (HEP) program at CERN has achieved major breakthroughs in particle physics, technology, and algorithms, including the discovery of the Higgs boson in 2012. This allowed the validation of compatibility of the theoretical construction behind the Standard Model (SM) of particle physics with the data, but the existing uncertainties leave room for models beyond the SM. With the experimental collider framework in place, scientific exploration continues to answer questions around the origin of dark matter, the disproportionately low abundance of antimatter and the nature of the discovered Higgs boson. Hard physics problems aside, much can be gained from improvements to the data acquisition pipeline allowing for capturing a richer set of collision events, furthering scientific understanding.

The Large Hadron Collider (LHC) consists of a 27 km tunnel where superconducting magnets guide bunches of protons, circulating in opposite directions, which are then caused to collide at experimental sites (e.g. ATLAS and CMS) at a rate of 40 million times per second. The collision events emit various particles, which are tracked through a multitude of radiation-hardened detectors and fed into the L1 trigger system, which needs to reject >99% of the events within 10 microseconds due to detector cache constraints and available network capacity.

This data is further reduced by >99% in the High-Level Trigger (HLT) to conform to the current event analysis and simulation capacity. HEP experimentation is fundamentally stochastic, so without changing other factors, an increase in data collection throughput would allow for higher confidence in current results while increasing the likelihood of detecting novel particles in the current LHC setup. Furthermore, this capacity increase is absolutely needed for future LHC upgrades where each collision will have many more interesting events.

The interpretation of the LHC data relies on theoretical simulations of particle interactions in the Standard Model (SM) and in scenarios of new physics beyond the SM (BSM). The full exploitation of the immense HL-LHC datasets, and in perspective of the data from Future Colliders, will require radical improvements in the computing strategies of theory calculations, to increase their accuracy while keeping affordable computing times. A multitude of theoretical tools must be addressed, in a coordinated effort, to preserve their interoperability and harmonize the overall precision. In addition to the several ingredients needed to describe the final states of proton collisions, the infrastructure developed for the triggers, e.g. the GPU cluster, also supports the advancement of software and algorithms for lattice Quantum Field Theory (LQFT) calculations, as a unique approach to control relevant non-perturbative ingredients. The engagement of LQFT experts would also bring to the trigger

- All of the previously highlighted areas are pursued by the "Next Generation Triggers" project
  - With external funding, CERN is starting a 5-year R&D project that will hire a large number of people, and get a large amount of accelerator hardware
- The goal is to find new ways for using the latest types of hardware in processing data from the HL-LHC
- If you're a graduate/post-doc, and interested in such developments, consider applying! 😉

Summary

- HEP experiments worldwide (and especially at CERN) are implementing support for accelerators in their software
  - With industry moving to providing computing power primarily in such a form in the future, this is a must
- CERN (experiments) have a rich development program for the following years to put accelerator usage into production
  - Both on our custom-built clusters, and the Worldwide LHC Computing Grid
- Some of this development is difficult to get help with from non-physicists
  - But with most of it we can certainly take advantage of industry support! Partnerships through CERN OpenLab can be of great help for the HL-LHC data taking / analysis.

CERN

http://home.cern