Data Analysis Trains for CLAS12

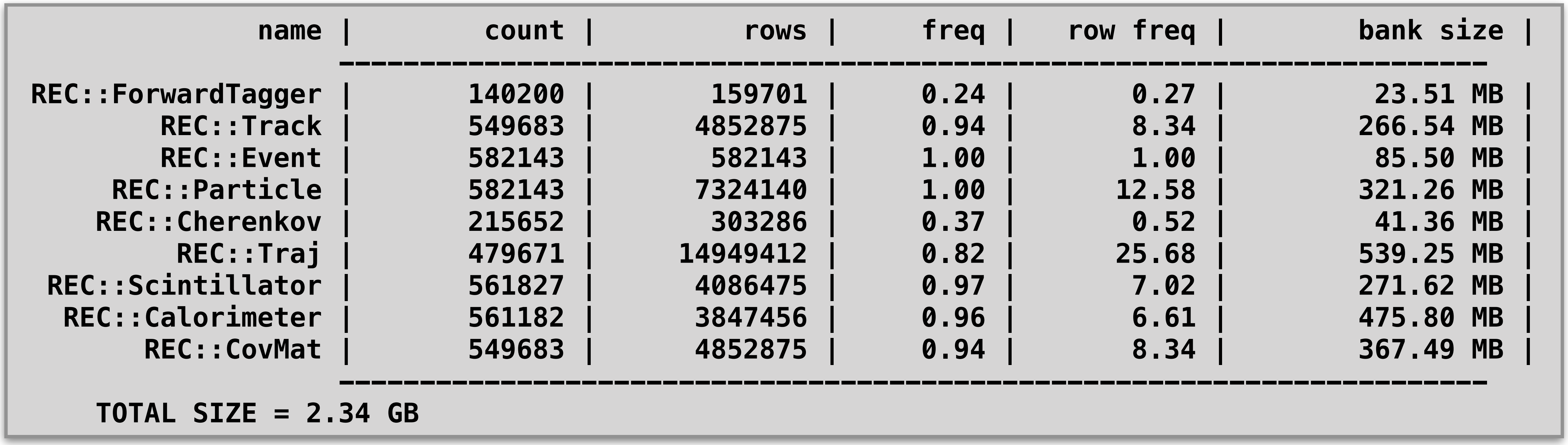
G.Gavalian (JLAB)

CLAS12 Data Analysis

- CLAS12 producing large amounts of data:
  - about 11TB of DSTs per week.
- Typical experiment produces 45 TB of DST data per month. (for comparison CLAS6 e2a and e2b experiments together produced 1 TB of data).
- Running data filtering for each analysis group on large data set will be computationally expensive.
- Unified approach is needed for filtering DST's for analysis groups.
- The solution is to develop analysis "Trains" for producing skimmed files for each analysis group to filter out events of interest.

DST

- After running reconstruction the output file is filtered to keep only relevant banks for physics analysis.



```
              name |         count |          rows |       freq |   row freq |          bank size |
                   -----------------------------------------------------------------------
REC::ForwardTagger |        140200 |        159701 |       0.24 |       0.27 |           23.51 MB |
        REC::Track |        549683 |       4852875 |       0.94 |       8.34 |          266.54 MB |
        REC::Event |        582143 |        582143 |       1.00 |       1.00 |           85.50 MB |
     REC::Particle |        582143 |       7324140 |       1.00 |      12.58 |          321.26 MB |
    REC::Cherenkov |        215652 |        303286 |       0.37 |       0.52 |           41.36 MB |
         REC::Traj |        479671 |      14949412 |       0.82 |      25.68 |          539.25 MB |
  REC::Scintillator |       561827 |       4086475 |       0.97 |       7.02 |          271.62 MB |
  REC::Calorimeter |        561182 |       3847456 |       0.96 |       6.61 |          475.80 MB |
       REC::CovMat |        549683 |       4852875 |       0.94 |       8.34 |          367.49 MB |
                   -----------------------------------------------------------------------
    TOTAL SIZE = 2.34 GB
```

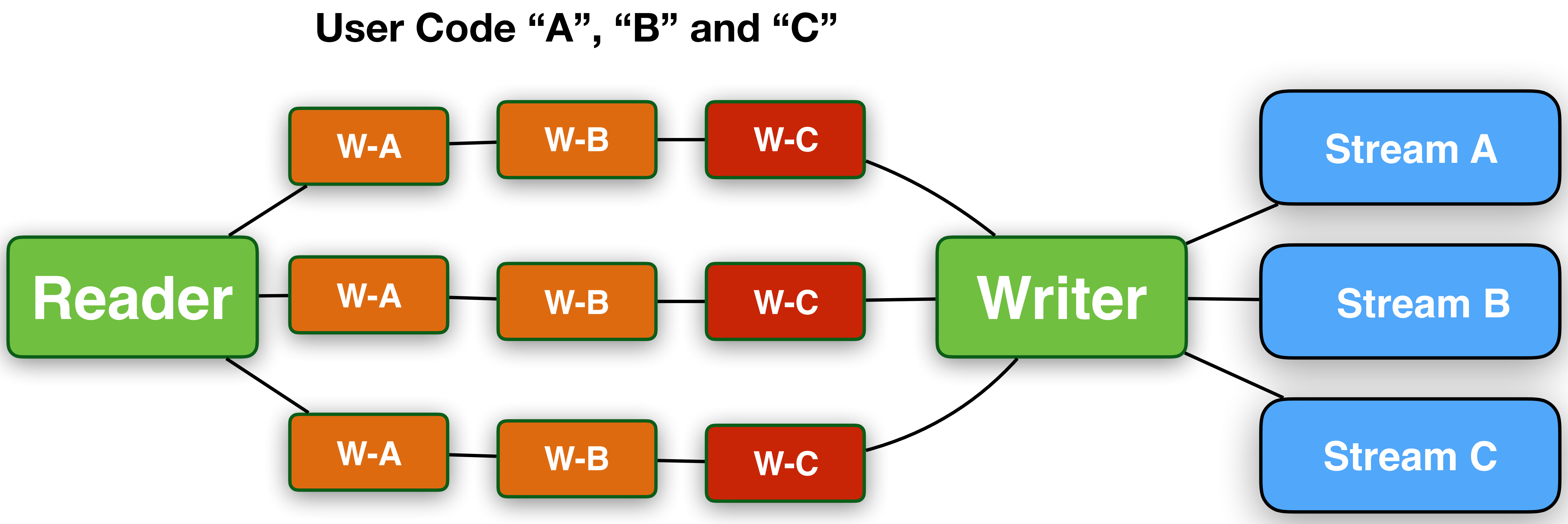
- Description of banks included in the DST can be found:
  - https://clasweb.jlab.org/wiki/index.php/CLAS12_DSTs
- The utilities for working with HIPO files, can be found:
  - https://userweb.jlab.org/~gavalian/docs/sphinx/hipo/html/index.html

Data Analysis for CLAS12 (CLARA)

- Data Analysis Trains are developed for skimming DST's
- Each analysis code (wagon) is a CLARA service that will run multithreaded as part of the analysis train.
- Generalized wagon service code is available for users to implement their selection of criteria for each event.
- Standard wagons are provided that can be configured externally (through YAML file) to perform basic selections.
- More complicated analysis code will be developed for more flexible data selection algorithms through configuration file.
- Generic analysis framework is being developed to provide users access to DST information through easy to use interfaces in JAVA. (will be presented later)

Analysis Train Design

- Reader Service from CLARA Reconstruction is used
- Writer Service was rewritten to output multiple streams
- Generic Wagon service was implemented for validating event
- Standard Wagon was implemented to make particle selections



Simple Case

```
io-services:
  reader:
    class: org.jlab.jnp.grapes.io.HipoStreamReader
    name: HipoStreamReader
  writer:
    class: org.jlab.jnp.grapes.io.HipoStreamWriter
    name: HipoStreamWriter
services:
  - class: org.jlab.jnp.grapes.services.GenericWagon
    name: WAGON
  - class: org.jlab.jnp.grapes.services.GenericWagon
    name: EXCLUSIVE
  - class: org.jlab.jnp.grapes.services.GenericWagon
    name: PIONSELECTION
configuration:
  services:
    WAGON:
      id: 1
      filter: 11:2212:211:X+:X-:Xn
    EXCLUSIVE:
      id: 2
      filter: 11:22:22:X+:X-:Xn
    PIONSELECTION:
      id: 3
      filter: 11:211:-211:2212:X+:X-:Xn
mime-types:
  - binary/data-hipo
```

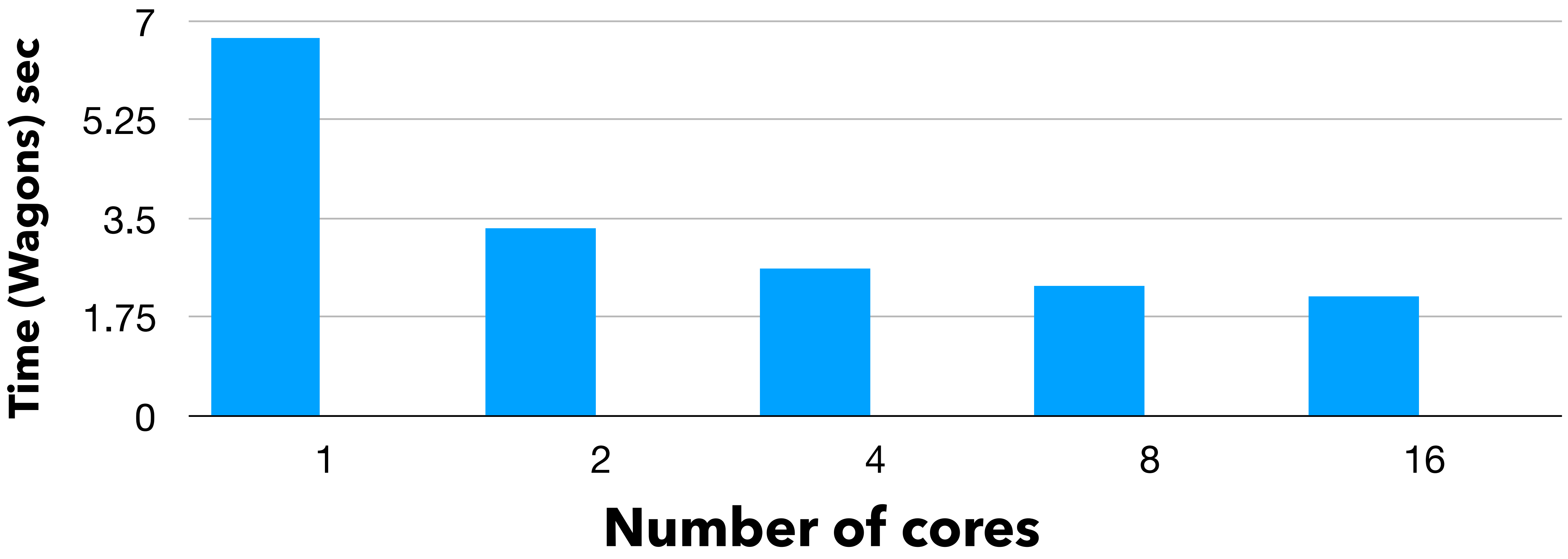
- **Simple case usage includes:**
  - Import GenericWagon service into CLARA chain.
  - Assign name to the reaction
  - Provide particle ID's to filter

More Flexible use

```
public class CustomWagon extends Wagon {
   public CustomWagon(){
       super("CustomWagon","myname","0.1");
   }
   @Override
   public boolean processDataEvent(HipoEvent event) {
       ParticleList pl = DataManager.getParticleList(event);
       if(pl.countByCharge(-1)<1) return false;
       if(pl.countByCharge(+1)<1) return false;
       if(pl.countByChargeMinMom(-1, 1.5)<1) return false;
       if(pl.countByChargeMinMom(+1, 1.5)<1) return false;
       return true;
   }
   @Override
   public boolean init(String jsonString) {
       // Initialization code goes here, this shows
         return true;
   }
}
```

Benchmarks (Very Extremely Preliminary)

- ~600K events in DST (1.1 GB file) run through the train
- three final states were selected and written in separate files
- time measured vs number of cores used.
- writer service is single threaded with many threads running it causes locks to distribute data into several streams.
- improvements are needed to writer to multithread the writing stream (coming soon)



Post-Train Analysis

- ROOT based data analysis:
  - standard code for converting DST's to ROOT (with collaboration support)
  - provide tools in ROOT for common data analysis environment (with collaboration support)
  - HASPECT framework can be used as a base, it already has many features implemented (contributors are needed)
- JAVA based data analysis:
  - develop analysis library in JAVA for accessing detector specific information from the events. Basic topology selection and final state identification
  - implement framework for doing standard data corrections and pid cuts
  - implement standardized fiducial cuts interfaces, complete with initial implementations
  - framework for incorporating Fast-MC into physics analysis

C++ Interface

- **C++/ROOT transition**
  - collaboration requested library to read HIPO in C++ to convert DSTs.
  - software group provided code for reading branches from HIPO in C++ for conversion to ROOT.
  - provided library has clean interface for reading the file and branches by types.
  - no writer is currently present in C++ library, because no one in collaboration requested ROOT to HIPO conversion tools.
  - the library is used by HASPECT6/ROOT analysis framework (successfully), which provides physics analysis tools as well as ROOT conversion tools.
  - The documentation on usage of C++ HIPO API can be found:
    - https://userweb.jlab.org/~gavalian/docs/sphinx/hipo/html/index.html

C++ interface

```
int main(int argc, char** argv) {
   std::cout << " reading file example program (HIPO) " << std::endl;
   char inputFile[256];

   if(argc>1) {
      sprintf(inputFile,"%s",argv[1]);
   } else {
      std::cout << " *** please provide a file name..." << std::endl;
     exit(0);
   }

   hipo::reader  reader;
   reader.open(inputFile);

   hipo::node<int32_t>         *LTCC__adc_ADC = reader.getBranch<int32_t>("LTCC::adc","ADC");
   hipo::node<int16_t>   *LTCC__adc_component = reader.getBranch<int16_t>("LTCC::adc","component");
   hipo::node<int8_t>        *LTCC__adc_layer = reader.getBranch<int8_t>("LTCC::adc","layer");
   hipo::node<int8_t>        *LTCC__adc_order = reader.getBranch<int8_t>("LTCC::adc","order");
   hipo::node<int16_t>         *LTCC__adc_ped = reader.getBranch<int16_t>("LTCC::adc","ped");
   hipo::node<int8_t>       *LTCC__adc_sector = reader.getBranch<int8_t>("LTCC::adc","sector");
   hipo::node<float>          *LTCC__adc_time = reader.getBranch<float>("LTCC::adc","time");

   //---------------------------------------------------
   //--  Main LOOP running through events and printing
   //--  values of the first decalred branch
   //---------------------------------------------------
   int entry = 0;
   while(reader.next()==true){
      entry++;
      std::cout << "event # " << entry << std::endl;
      int n_LTCC__adc_ADC = LTCC__adc_ADC->getLength();
      for(int b = 0; b < n_LTCC__adc_ADC; b++){
         std::cout << LTCC__adc_ADC->getValue(b) << " " ;
       }
      std::cout << std::endl;
   }
   //---------------------------------------------------
}
```

https://userweb.jlab.org/~gavalian/docs/sphinx/hipo/html/index.html

HIPO C++ interface

- **HIPO C++ library was updated:**
  - parsing dictionary is implemented.
  - rendering each branch by name or by id.
  - automated code generation for read function.
  - currently only reading is supported.
  - support for random and sequential read.
- **HIPO C++ future developments:**
  - writer code will be developed.
  - improvements on dictionaries parsing.
  - automated class generation for banks to improve performance.

Summary

- **The Analysis Activity:**
  - Analysis trains are being developed for unified data filtering. Test runs will be carried out mid July.
  - Will support configurable services as well as custom user code. Downloadable project with will be available for users to get started.
  - JAVA analysis library is in development, including pid cuts, momentum corrections and fiducial cuts.
  - Work will start on developing standard ROOT conversion tools and analysis environment.

Project	Completion
Development of Analysis Trains. Running over first production data. Debugging and optimizing the code.	August 15
Analysis library. Includes : final state identification, data correction framework and fiducial cuts.	Sep 15
ROOT utilities for data analysis. HIPO to ROOT conversion, analysis tools in ROOT (with contribution from collaboration)	Sep 15