Improving Code For Future Users For The Wire Chambers (MWPCs)

Neomy O. Gutierrez, Loyola University Chicago, SIST

Dr. Mandy Rominsky & Dr. Evan Niner

Final Presentation

30 July 2019

Table Of Contents

- Introduction
  - Multi-Wire Proportional Chambers
  - TDCs
- Beam Overview
  - MTest
  - MCenter
  - Instrumentation
- Processing and Collecting Data
  - OTSDAQ
  - ArtDAQ
  - ROOT
- Code
  - Book Canvas
  - Book Histos
  - Fill
  - Write
- Histograms
  - Profile Hits
  - Online Monitoring
  - TDC
  - Hit Time
  - Time Difference
  - XMWPC
- Future Work

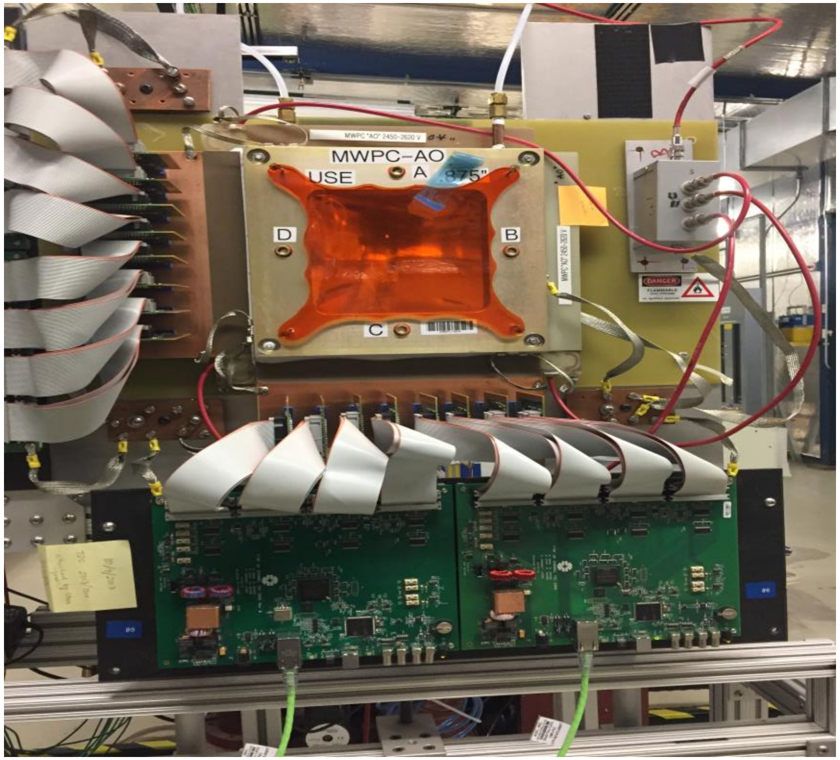
Introduction

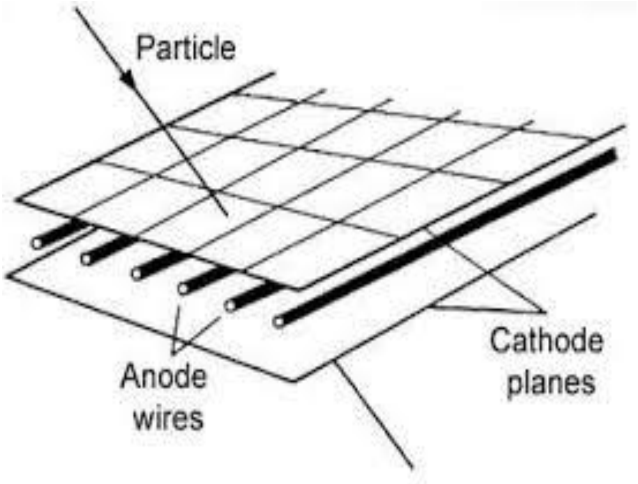
- Project
  - Improve and organize the multiple codes/programs, so that they are accessible and user-friendly for future users to run their detectors to collect the data from the Wire Chambers

- Fermilab Test Beam Facility (FTBF)
  - The Fermilab Test Beam Facility (FTBF) is a location that has a beam of high energy particles for researchers' (users') detectors.
  - The FTBF has two beam lines, MTest and MCenter, which provide a variety of particle types such as proton beam and secondary beams with muons, pions, electrons, and kaons.

Multi-Wired Proportional Chambers (MWPCs)

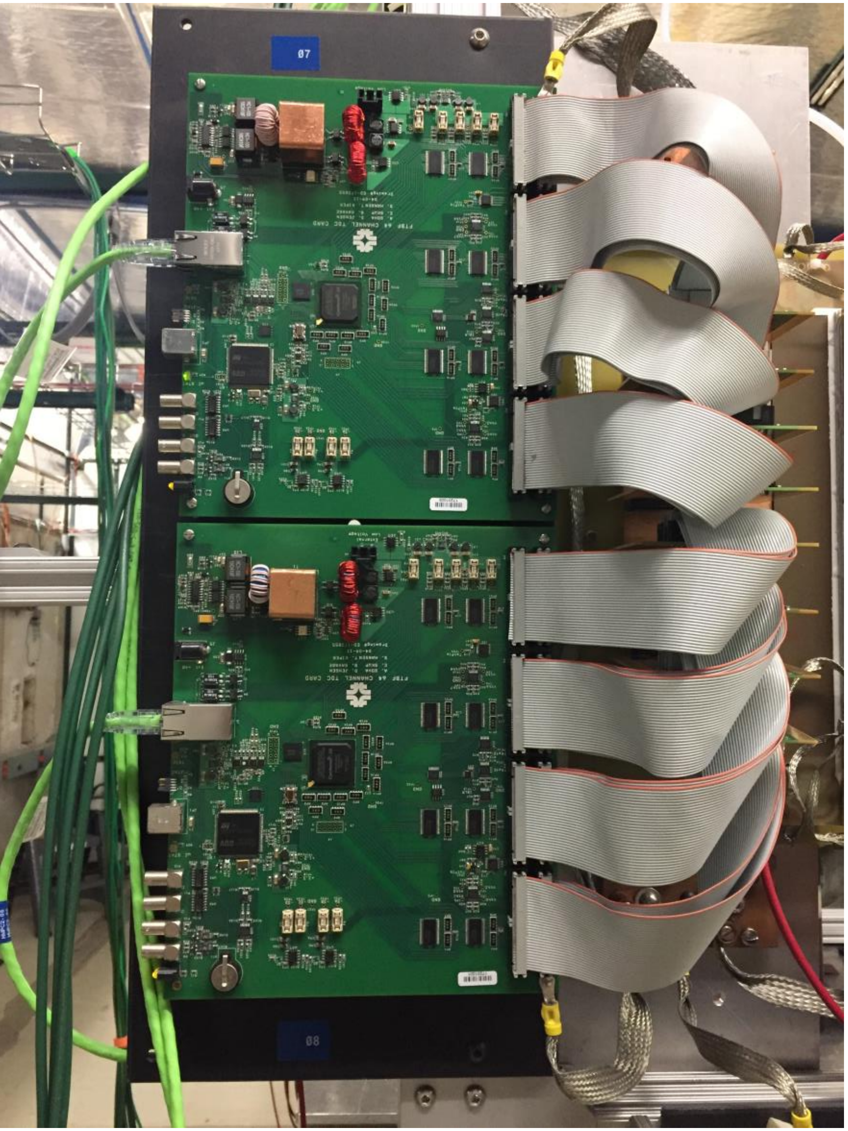
- 128 wires are placed in a perpendicular position
- It was designed to reduce the amount of matter in the path of the beam.
- When the beam is passing through these chambers, it will hit these wires causing them to collect data of where and when the beam hits



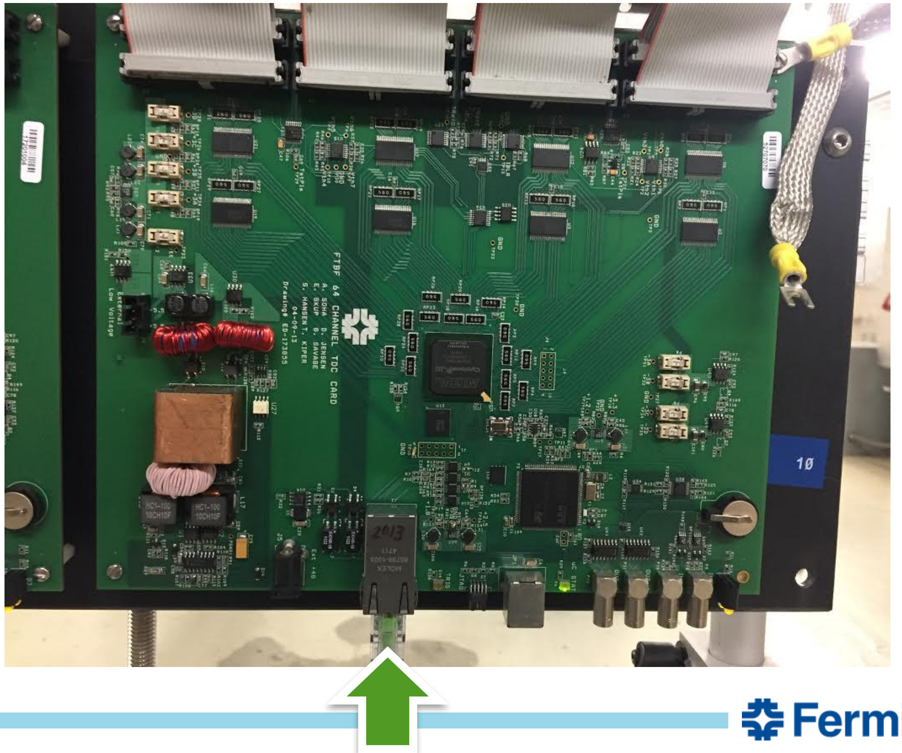


- Due to the intense level of the beam, these chambers sometimes have inefficient collecting data
- Each chamber can only handle a certain level of voltage

TDCs



- Time To Digital Converter
- Each chamber carries four non-metric amplifier discriminator cards, 16 in total, called TDC.
- The read-out software is written in Python and executes on a computer running Linux



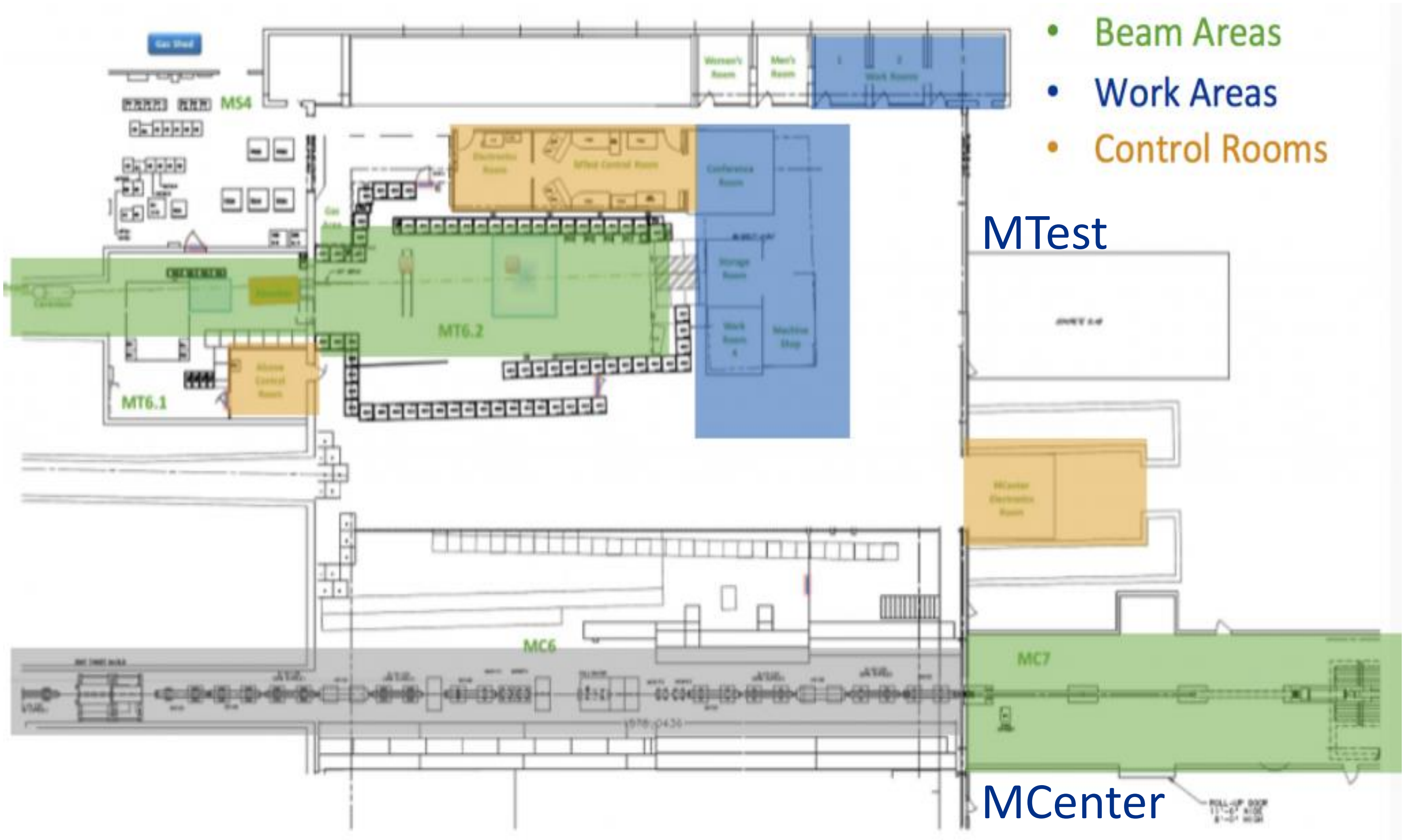
Beam Overview

- **MTest**
  - is the primary beam, which carries high-energy protons that are 120 GeV at moderate intensities.
  - can create secondary particles of energies about 1 GeV, which are pions, muons, and/or electrons.
  - The MTest is used for a short period of time due to overheating.

- **MCenter**
  - This beamline is used for long-term experiments. Rather than the MTest due to the summer shut down.
  - This beamline carries the same particles as the MTest, yet there is the addition of a tertiary beamline.
  - It can produce pions and/or protons down to energies of 0.20 GeV.

Both of these beamlines have about an equal amount of facility infrastructure and instrumentation.

Figure 2



- Beam Areas
- Work Areas
- Control Rooms
MTest
MCenter
Gas Shed
MS4
Women's Room
Men's Room
Work Rooms
Electronics Room
MTest Control Room
Conference Room
Storage Room
Work Room
Machine Shop
Gas Area
MT6.2
MT6.1
Above Control Room
MCenter Electronics Room
MC6
MC7
ROLL-UP DOOR

Instrumentation

- Both MTest and MCenter contain detector instruments for tracking, particle identification, and triggering.

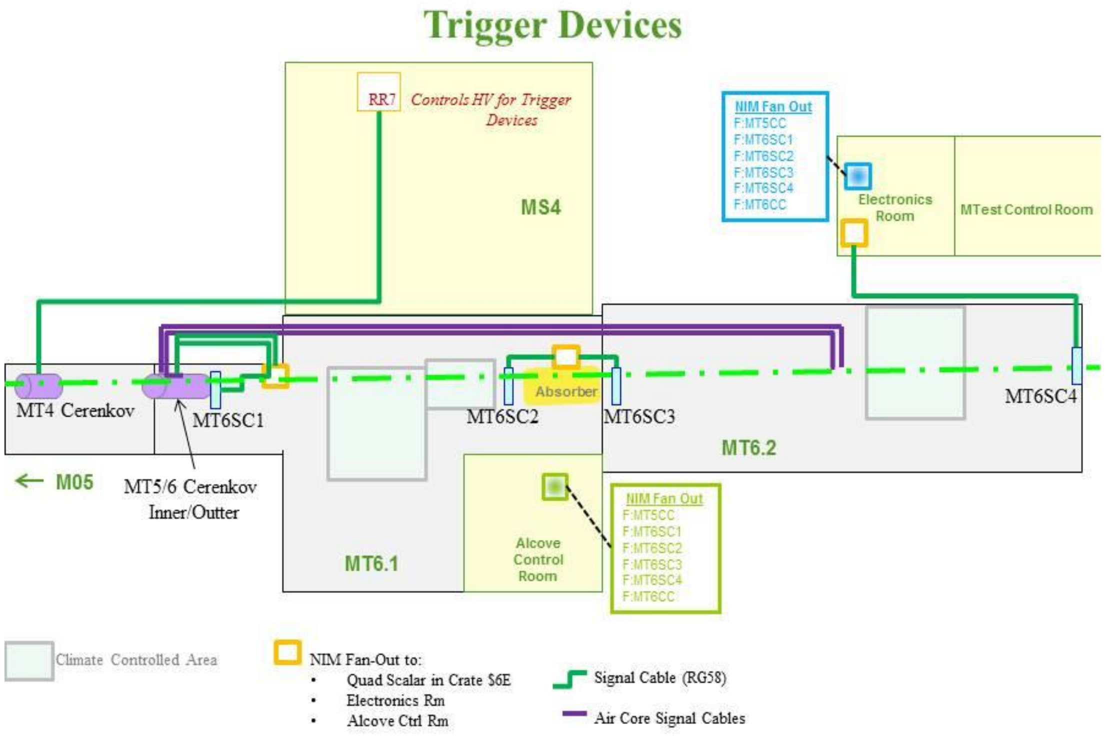
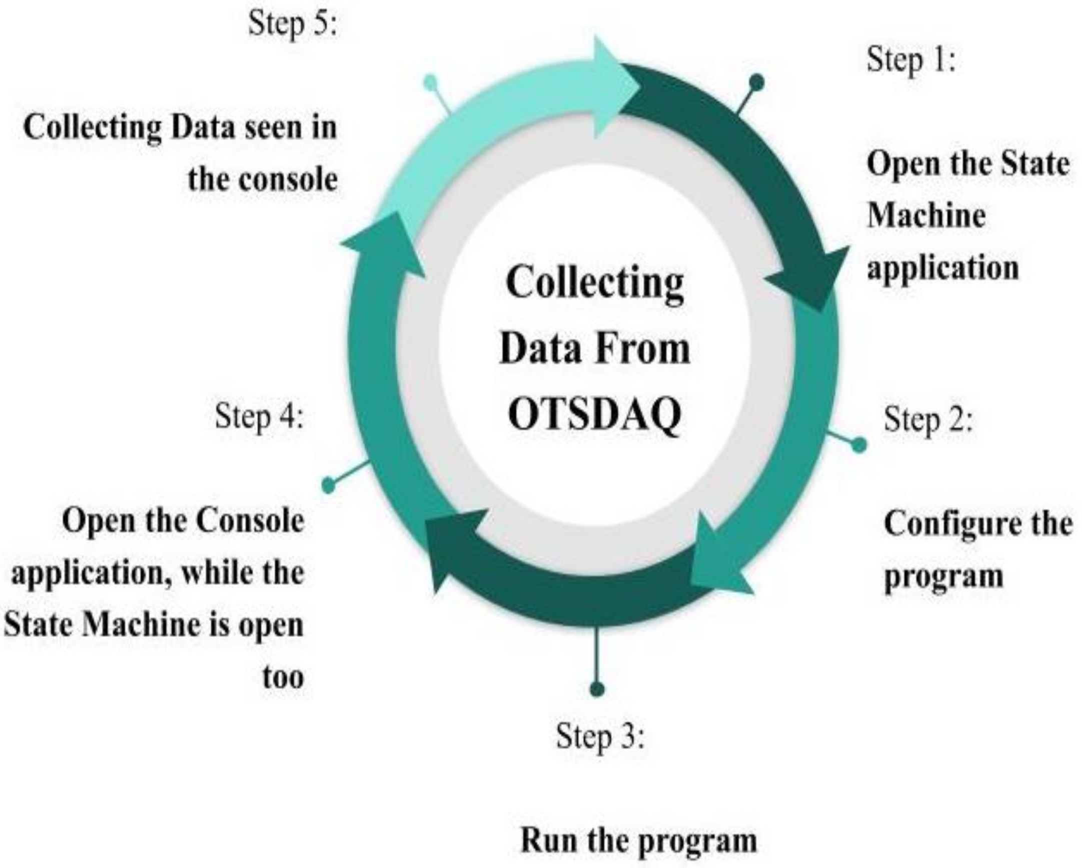


Figure 3

- These include scintillators, Čerenkov detectors, lead glass calorimeters, silicon detectors, time-of-flight systems, and wire chambers.
- These systems can work alone and come with their DAQ system or they can be integrated into the user’s setup.

Processing a Collecting Data

- **OTSDAQ**
  - a Ready-to-Use data-acquisition (DAQ) solution aimed at test-beam, detector development, and other rapid-deployment scenarios.
  - it provides a library of supported front-end boards and firmware modules.
  - Collects data, as seen in the figure



Collecting Data From OTSDAQ

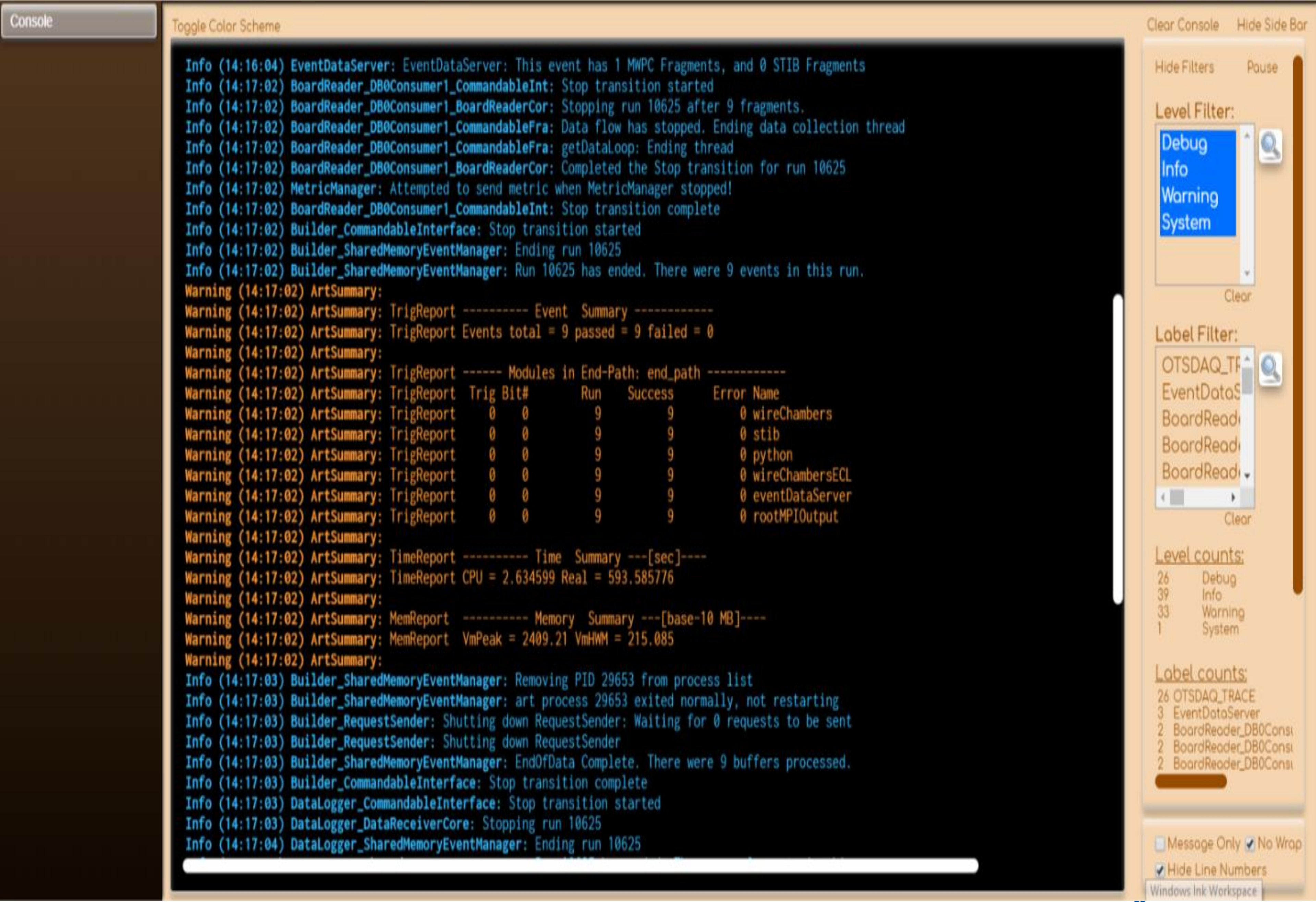
Step 1: Open the State Machine application

Step 2: Configure the program

Step 3: Run the program

Step 4: Open the Console application, while the State Machine is open too

Step 5: Collecting Data seen in the console



```
Info (14:16:04) EventDataServer: EventDataServer: This event has 1 MWPC Fragments, and 0 STIB Fragments
Info (14:17:02) BoardReader_DBOConsumer1_CommandableInt: Stop transition started
Info (14:17:02) BoardReader_DBOConsumer1_BoardReaderCor: Stopping run 10625 after 9 fragments.
Info (14:17:02) BoardReader_DBOConsumer1_CommandableFra: Data flow has stopped. Ending data collection thread
Info (14:17:02) BoardReader_DBOConsumer1_CommandableFra: getDataLoop: Ending thread
Info (14:17:02) BoardReader_DBOConsumer1_BoardReaderCor: Completed the Stop transition for run 10625
Info (14:17:02) MetricManager: Attempted to send metric when MetricManager stopped!
Info (14:17:02) BoardReader_DBOConsumer1_CommandableInt: Stop transition complete
Info (14:17:02) Builder_CommandableInterface: Stop transition started
Info (14:17:02) Builder_SharedMemoryEventManager: Ending run 10625
Info (14:17:02) Builder_SharedMemoryEventManager: Run 10625 has ended. There were 9 events in this run.
Warning (14:17:02) ArtSummary:
Warning (14:17:02) ArtSummary: TrigReport ---------- Event  Summary ------------
Warning (14:17:02) ArtSummary: TrigReport Events total = 9 passed = 9 failed = 0
Warning (14:17:02) ArtSummary:
Warning (14:17:02) ArtSummary: TrigReport ------ Modules in End-Path: end_path ------------
Warning (14:17:02) ArtSummary: TrigReport  Trig Bit#       Run    Success      Error Name
Warning (14:17:02) ArtSummary: TrigReport     0    0         9          9          0 wireChambers
Warning (14:17:02) ArtSummary: TrigReport     0    0         9          9          0 stib
Warning (14:17:02) ArtSummary: TrigReport     0    0         9          9          0 python
Warning (14:17:02) ArtSummary: TrigReport     0    0         9          9          0 wireChambersECL
Warning (14:17:02) ArtSummary: TrigReport     0    0         9          9          0 eventDataServer
Warning (14:17:02) ArtSummary: TrigReport     0    0         9          9          0 rootMPIOutput
Warning (14:17:02) ArtSummary:
Warning (14:17:02) ArtSummary: TimeReport ---------- Time  Summary ---[sec]----
Warning (14:17:02) ArtSummary: TimeReport CPU = 2.634599 Real = 593.585776
Warning (14:17:02) ArtSummary:
Warning (14:17:02) ArtSummary: MemReport  ---------- Memory  Summary ---[base-10 MB]----
Warning (14:17:02) ArtSummary: MemReport  VmPeak = 2409.21 VmHWM = 215.085
Warning (14:17:02) ArtSummary:
Info (14:17:03) Builder_SharedMemoryEventManager: Removing PID 29653 from process list
Info (14:17:03) Builder_SharedMemoryEventManager: art process 29653 exited normally, not restarting
Info (14:17:03) Builder_RequestSender: Shutting down RequestSender: Waiting for 0 requests to be sent
Info (14:17:03) Builder_RequestSender: Shutting down RequestSender
Info (14:17:03) Builder_SharedMemoryEventManager: EndOfData Complete. There were 9 buffers processed.
Info (14:17:03) Builder_CommandableInterface: Stop transition complete
Info (14:17:03) DataLogger_CommandableInterface: Stop transition started
Info (14:17:03) DataLogger_DataReceiverCore: Stopping run 10625
Info (14:17:04) DataLogger_SharedMemoryEventManager: Ending run 10625
```

Level counts:
26 Debug
39 Info
33 Warning
1 System

Label counts:
26 OTSDAQ_TRACE
3 EventDataServer
2 BoardReader_DBOConsu
2 BoardReader_DBOConsu
2 BoardReader_DBOConsu

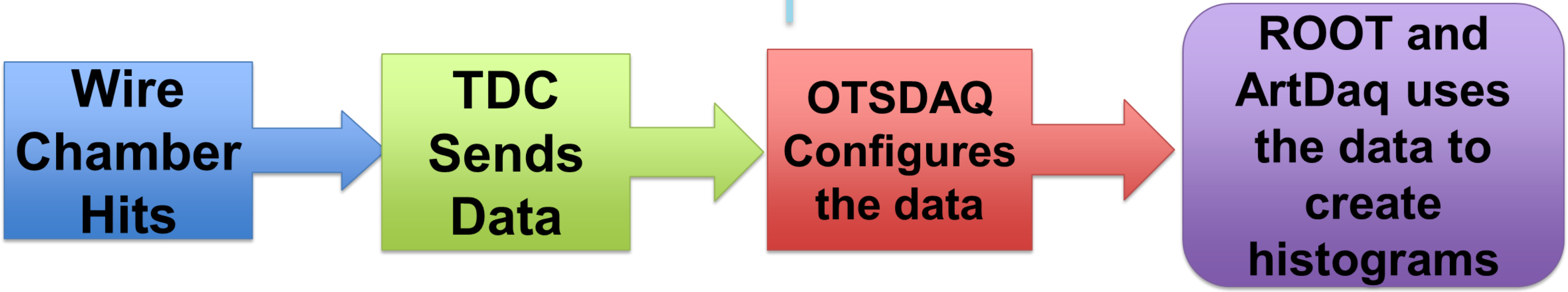
Fermilab

ArtDAQ

- The toolkit currently provides functionality for data transfer, event building, event reconstruction and analysis process management, etc.
- DAQ process and *art* module configuration, and the writing of event data to disk in the ROOT format.

ROOT

- A modular scientific software toolkit. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualization and storage.
- It is mainly written in C++ but integrated with other languages such as Python.
- Included are histogramming methods in 1-3 dimensions, curve fitting,etc. to allow an easy setup for the process data to be seen as a visual.



Wire Chamber Hits

TDC Sends Data

OTSDAQ Configures the data

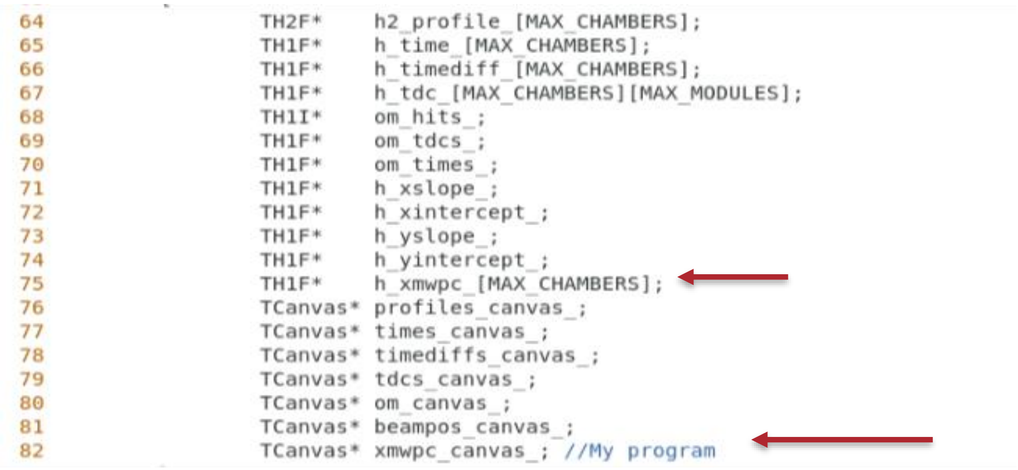
ROOT and ArtDaq uses the data to create histograms

Code

Pwd: /data-08/otsdaq_dev_neomy/srcs/otsdaq_fermilabtestbeam/otsdaq-fermilabtestbeam/ArtModules

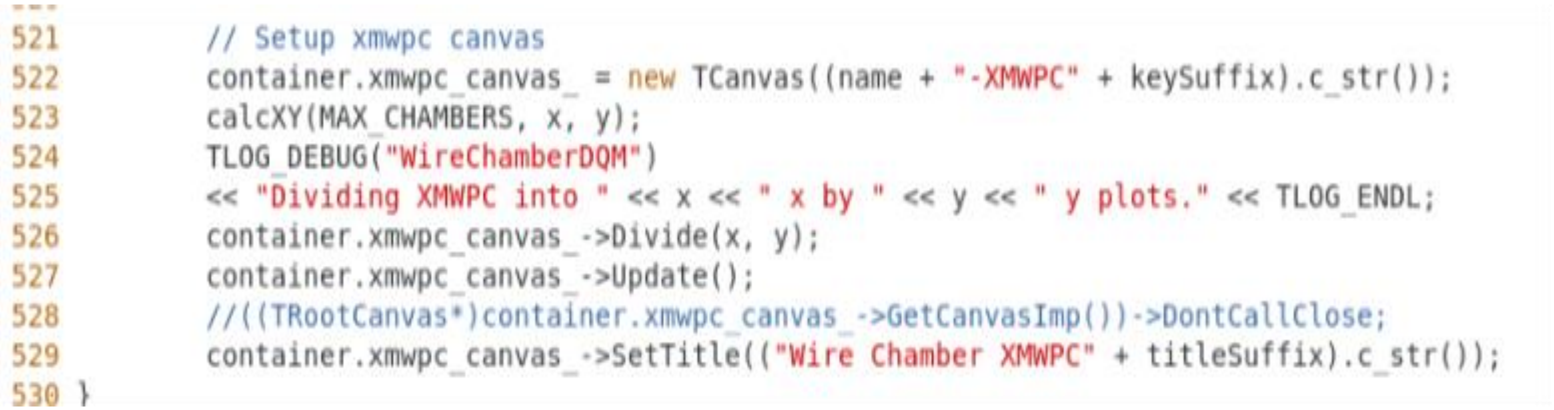
vi WireChamberDQM_module.cc

Declaring the variable



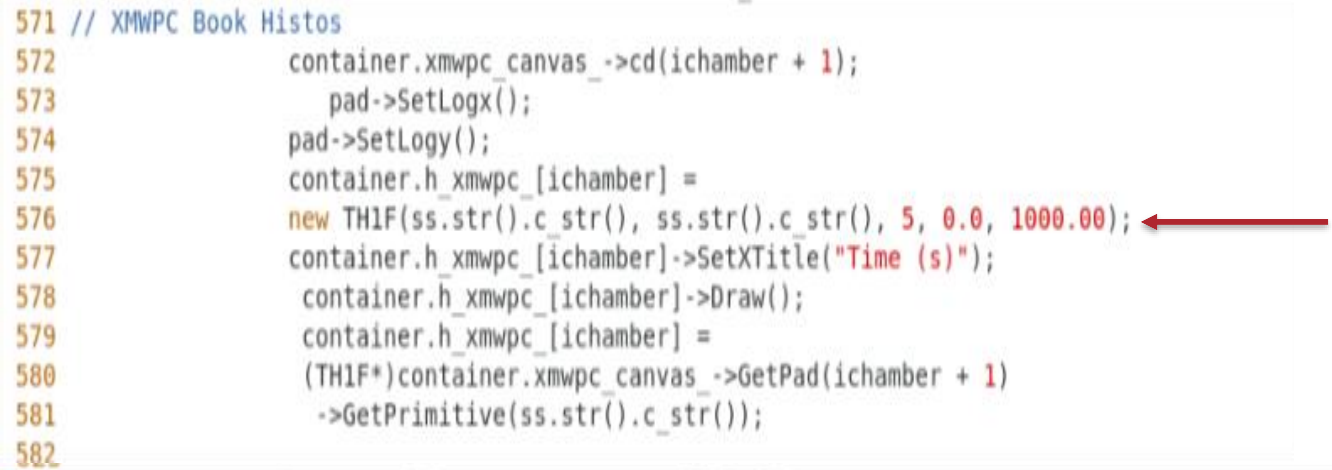
```
        TH2F*     h2_profile_[MAX_CHAMBERS];
        TH1F*     h_time_[MAX_CHAMBERS];
        TH1F*     h_timediff_[MAX_CHAMBERS];
        TH1F*     h_tdc_[MAX_CHAMBERS][MAX_MODULES];
        TH1I*     om_hits_;
        TH1F*     om_tdcs_;
        TH1F*     om_times_;
        TH1F*     h_xslope_;
        TH1F*     h_xintercept_;
        TH1F*     h_yslope_;
        TH1F*     h_yintercept_;
        TH1F*     h_xmwpc_[MAX_CHAMBERS];
        TCanvas* profiles_canvas_;
        TCanvas* times_canvas_;
        TCanvas* timediffs_canvas_;
        TCanvas* tdcs_canvas_;
        TCanvas* om_canvas_;
        TCanvas* beampos_canvas_;
        TCanvas* xmwpc_canvas_; //My program
```

Book Canvas → uses the ROOT classes to split the canvas into four sections



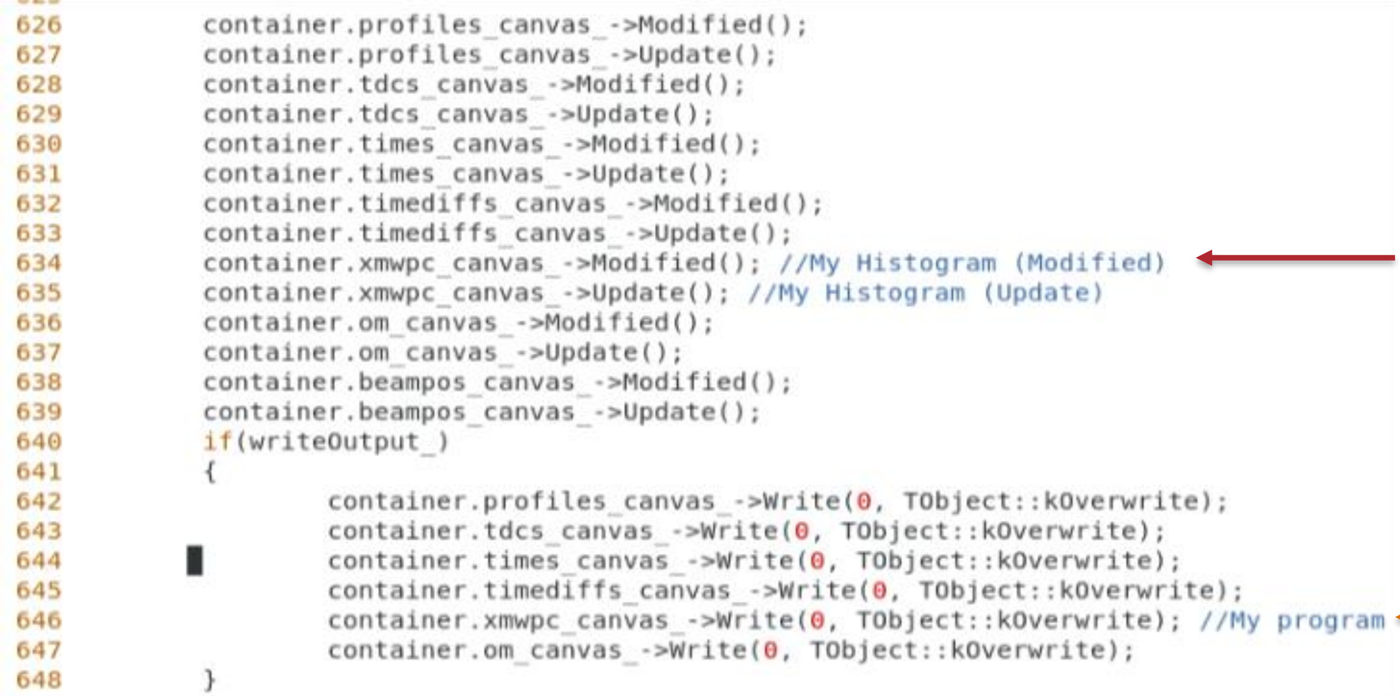
```
        // Setup xmwpc canvas
        container.xmwpc_canvas_ = new TCanvas((name + "-XMWPC" + keySuffix).c_str());
        calcXY(MAX_CHAMBERS, x, y);
        TLOG_DEBUG("WireChamberDQM")
        << "Dividing XMWPC into " << x << " x by " << y << " y plots." << TLOG_ENDL;
        container.xmwpc_canvas_->Divide(x, y);
        container.xmwpc_canvas_->Update();
        //((TRootCanvas*)container.xmwpc_canvas_->GetCanvasImp())->DontCallClose;
        container.xmwpc_canvas_->SetTitle(("Wire Chamber XMWPC" + titleSuffix).c_str());
}
```

Book Histos→ uses the ROOT classes above to create the desired histograms. This can be numbers of bins and the ranges of both x and y.



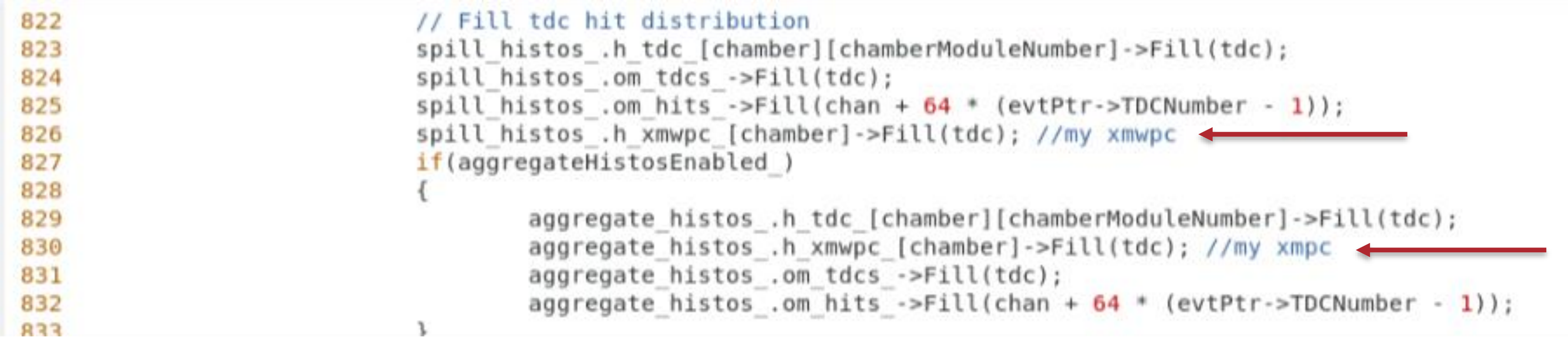
```
// XMWPC Book Histos
            container.xmwpc_canvas_->cd(ichamber + 1);
              pad->SetLogx();
            pad->SetLogy();
            container.h_xmwpc_[ichamber] =
            new TH1F(ss.str().c_str(), ss.str().c_str(), 5, 0.0, 1000.00);
            container.h_xmwpc_[ichamber]->SetXTitle("Time (s)");
             container.h_xmwpc_[ichamber]->Draw();
             container.h_xmwpc_[ichamber] =
             (TH1F*)container.xmwpc_canvas_->GetPad(ichamber + 1)
              ->GetPrimitive(ss.str().c_str());

```



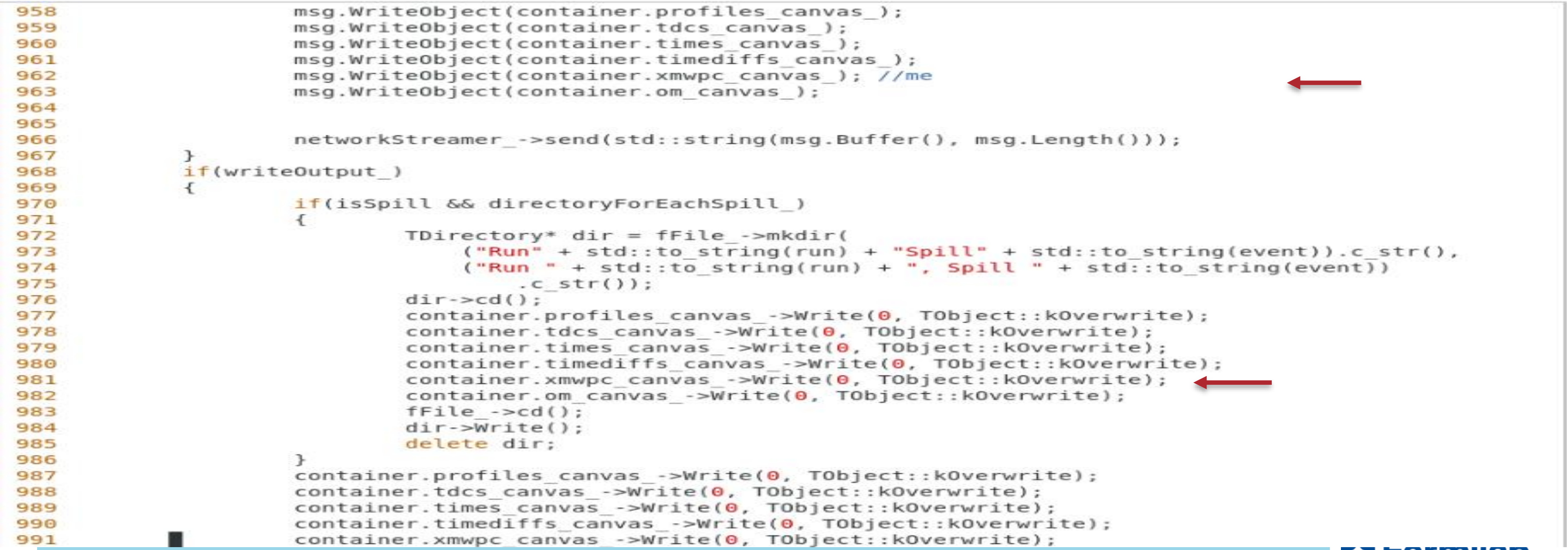
```
        container.profiles_canvas_->Modified();
        container.profiles_canvas_->Update();
        container.tdcs_canvas_->Modified();
        container.tdcs_canvas_->Update();
        container.times_canvas_->Modified();
        container.times_canvas_->Update();
        container.timediffs_canvas_->Modified();
        container.timediffs_canvas_->Update();
        container.xmwpc_canvas_->Modified(); //My Histogram (Modified)
        container.xmwpc_canvas_->Update(); //My Histogram (Update)
        container.om_canvas_->Modified();
        container.om_canvas_->Update();
        container.beampos_canvas_->Modified();
        container.beampos_canvas_->Update();
        if(writeOutput_)
        {
                container.profiles_canvas_->Write(0, TObject::kOverwrite);
                container.tdcs_canvas_->Write(0, TObject::kOverwrite);
                container.times_canvas_->Write(0, TObject::kOverwrite);
                container.timediffs_canvas_->Write(0, TObject::kOverwrite);
                container.xmwpc_canvas_->Write(0, TObject::kOverwrite); //My program
                container.om_canvas_->Write(0, TObject::kOverwrite);
        }
```

Fill → pulls the data from the electronic modules in the control room and plots it into the desired histogram. It then transfers the final histogram on the canvas created.



```
// Fill tdc hit distribution
spill_histos_.h_tdc_[chamber][chamberModuleNumber]->Fill(tdc);
spill_histos_.om_tdcs_->Fill(tdc);
spill_histos_.om_hits_->Fill(chan + 64 * (evtPtr->TDCNumber - 1));
spill_histos_.h_xmwpc_[chamber]->Fill(tdc); //my xmwpc
if(aggregateHistosEnabled_)
{
        aggregate_histos_.h_tdc_[chamber][chamberModuleNumber]->Fill(tdc);
        aggregate_histos_.h_xmwpc_[chamber]->Fill(tdc); //my xmpc
        aggregate_histos_.om_tdcs_->Fill(tdc);
        aggregate_histos_.om_hits_->Fill(chan + 64 * (evtPtr->TDCNumber - 1));
}
```

Write → creates a directory in root where all the canvases are stored



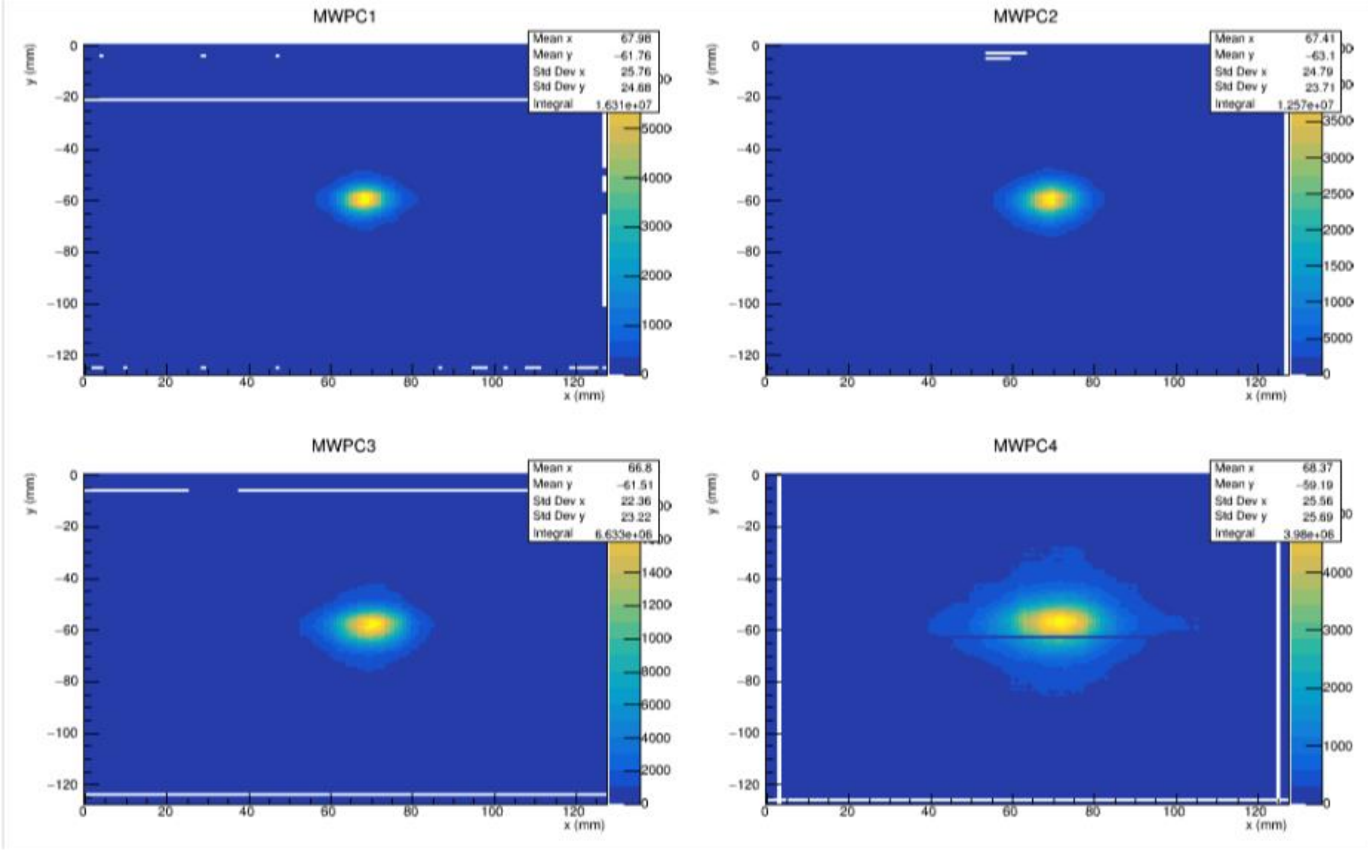
```
                msg.WriteObject(container.profiles_canvas_);
                msg.WriteObject(container.tdcs_canvas_);
                msg.WriteObject(container.times_canvas_);
                msg.WriteObject(container.timediffs_canvas_);
                msg.WriteObject(container.xmwpc_canvas_); //me
                msg.WriteObject(container.om_canvas_);


                networkStreamer_->send(std::string(msg.Buffer(), msg.Length()));
        }
        if(writeOutput_)
        {
                if(isSpill && directoryForEachSpill_)
                {
                        TDirectory* dir = fFile_->mkdir(
                            ("Run" + std::to_string(run) + "Spill" + std::to_string(event)).c_str(),
                            ("Run " + std::to_string(run) + ", Spill " + std::to_string(event))
                                .c_str());
                        dir->cd();
                        container.profiles_canvas_->Write(0, TObject::kOverwrite);
                        container.tdcs_canvas_->Write(0, TObject::kOverwrite);
                        container.times_canvas_->Write(0, TObject::kOverwrite);
                        container.timediffs_canvas_->Write(0, TObject::kOverwrite);
                        container.xmwpc_canvas_->Write(0, TObject::kOverwrite);
                        container.om_canvas_->Write(0, TObject::kOverwrite);
                        fFile_->cd();
                        dir->Write();
                        delete dir;
                }
                container.profiles_canvas_->Write(0, TObject::kOverwrite);
                container.tdcs_canvas_->Write(0, TObject::kOverwrite);
                container.times_canvas_->Write(0, TObject::kOverwrite);
                container.timediffs_canvas_->Write(0, TObject::kOverwrite);
                container.xmwpc_canvas_->Write(0, TObject::kOverwrite);
```

Histograms

- From the data gathering, there was a total of ten spills. Each spill creates six histograms from ArtDAQ and ROOT.
- Other then the ten spills, there is the last section called “All Spills,” which calculates and diagrams the average of all ten spills.

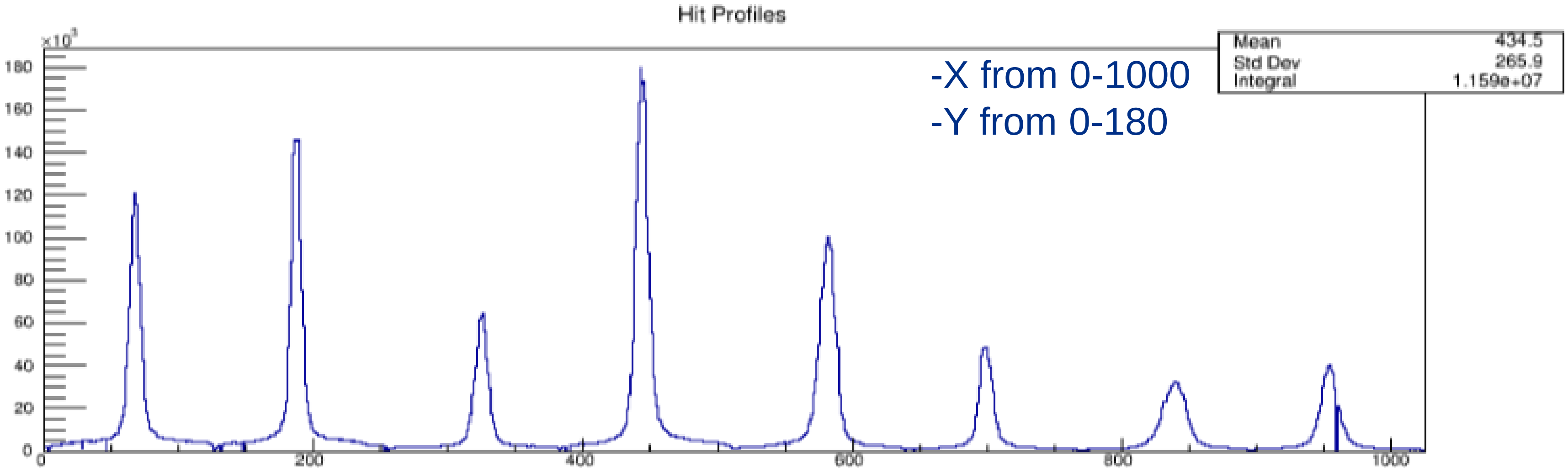
All Spills:



Profile Hits:

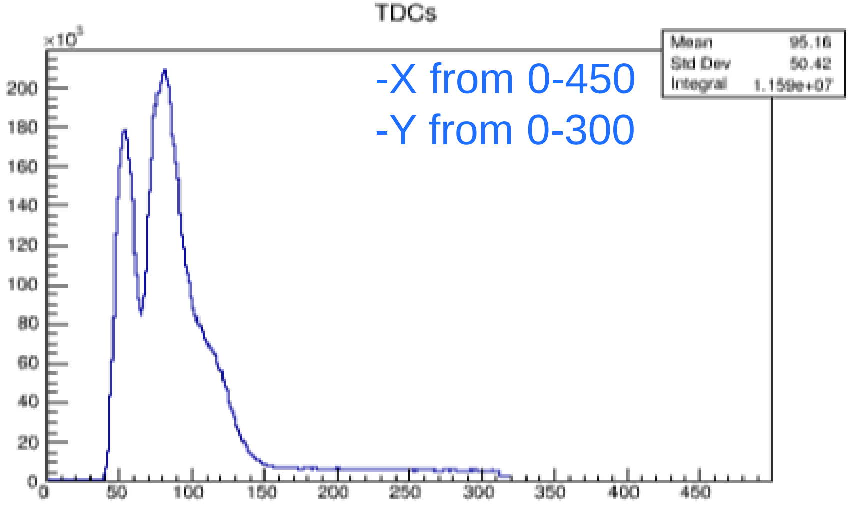
- Both x and y are measured in mm
- X Ranges from 0-120
- Y Ranges from -120-0
- Intensity level from 0-6000

Online Monitoring



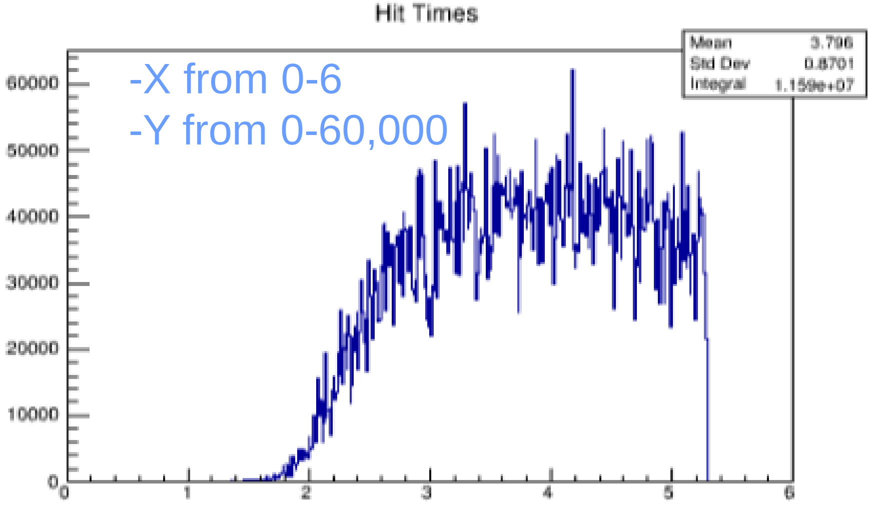
-X from 0-1000

-Y from 0-180



-X from 0-450

-Y from 0-300



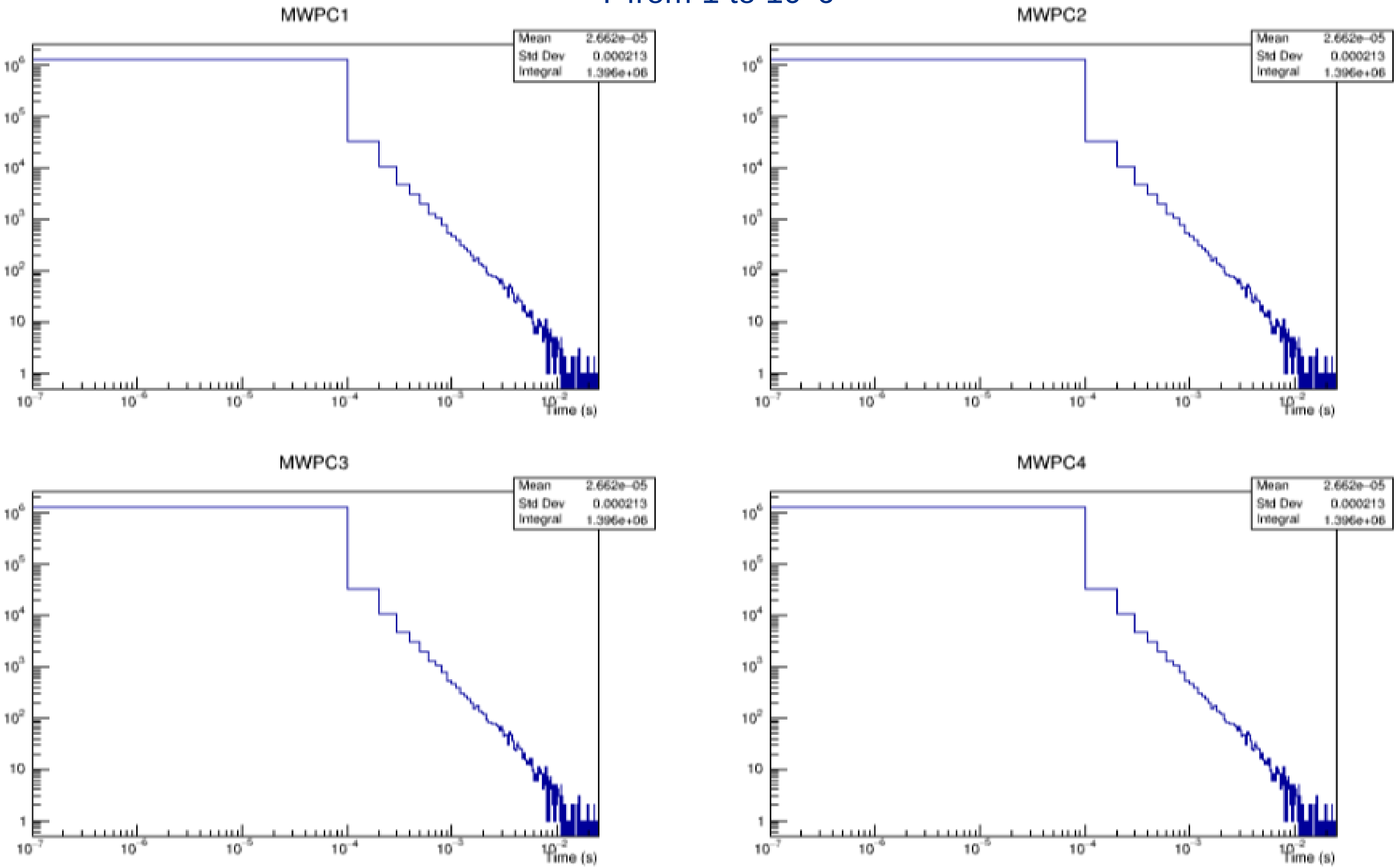
-X from 0-6

-Y from 0-60,000

Time Differences

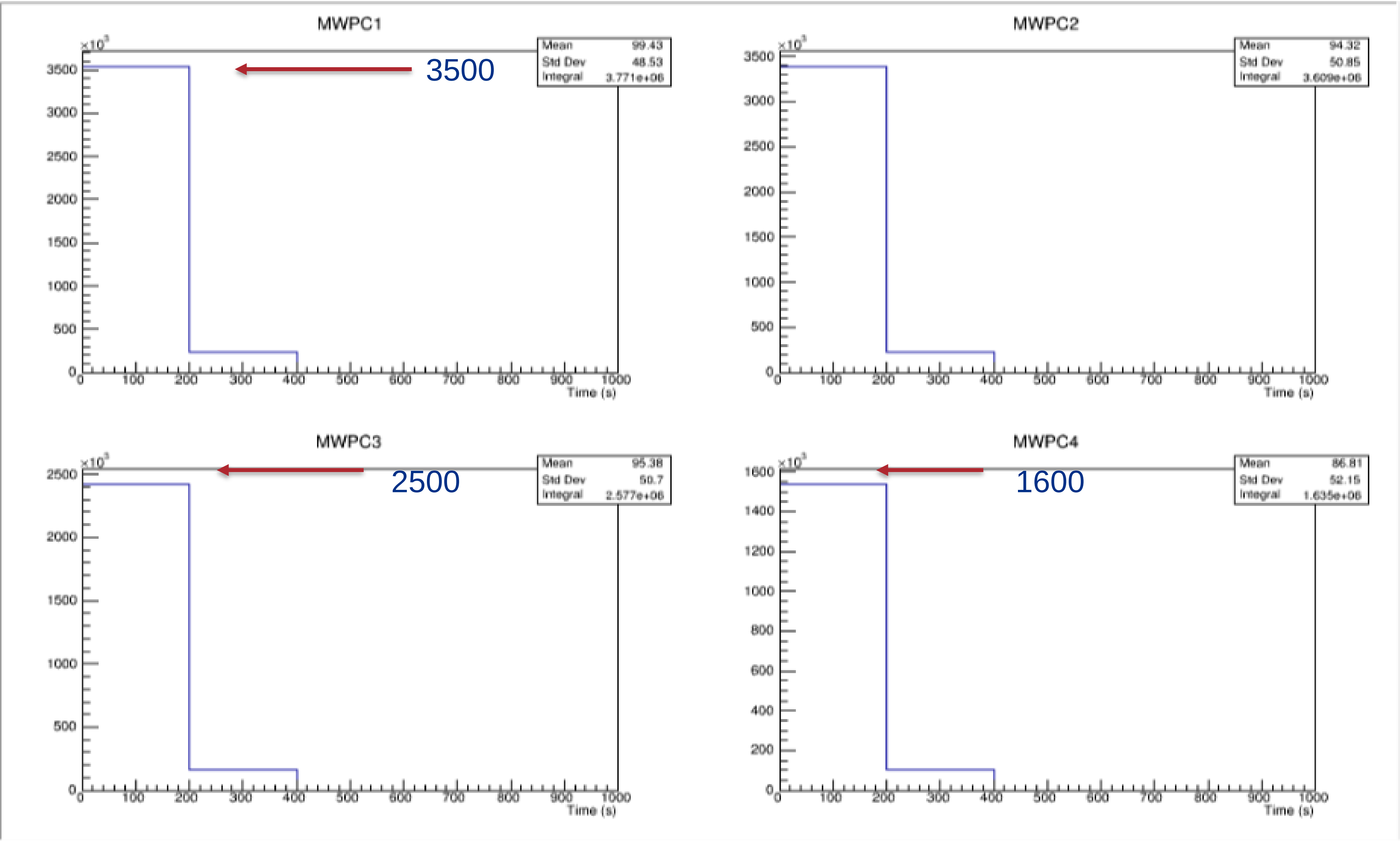
-X from 10^-7 to 10^-2

-Y from 1 to 10^6



XMWPC: shows the intensity of the beam hitting the chambers (1-4)

X: from 0-1000



3500

2500

1600

Future Work

- Multiple codes can be developed and translated to C++ for both the machines and ROOT to understand the programs.
- Using the code I generated, users will be able to edit it to create whichever histogram they wish.

.

Thank you!

Acknowledgement:

- Advisors: Dr. Evan Niner and Dr. Mandy Rominsky
- FTBF Instructors: Ewa Skup and Todd Nebel
- Mentors: Camille, Donovan, and Arden
- Guidance: Sandra Charles

References

1. https://ftbf.fnal.gov/beam-overview/
2. https://ftbf.fnal.gov/instrumentation-overview/
3. https://cdcvs.fnal.gov/redmine/projects/artdaq
4. https://www-d0.fnal.gov/computing/tools/root/about_root.html
5. https://root.cern.ch/