Introduction to Bristol Analysis Tools

Lukasz Kreczko
University of Bristol

Outline

- Short introduction
- Concepts & Structure
- Code examples of the most important files
- **Break for questions**
- Where to find X?
- How to do my own analysis
- Current Review & planned changes
- Scripts
- Editors to make you life easier
- Useful links
  - Analysis & Ntuples

What are the BAT?

- A light framework to analyse data in nTuple format
  - Read the variables stored in a safe way
  - Create objects (electrons, jets, MET etc) to work with
- Collection of scripts to help with
  - nTuple preparation (merging & compression for faster access)
  - Make plots

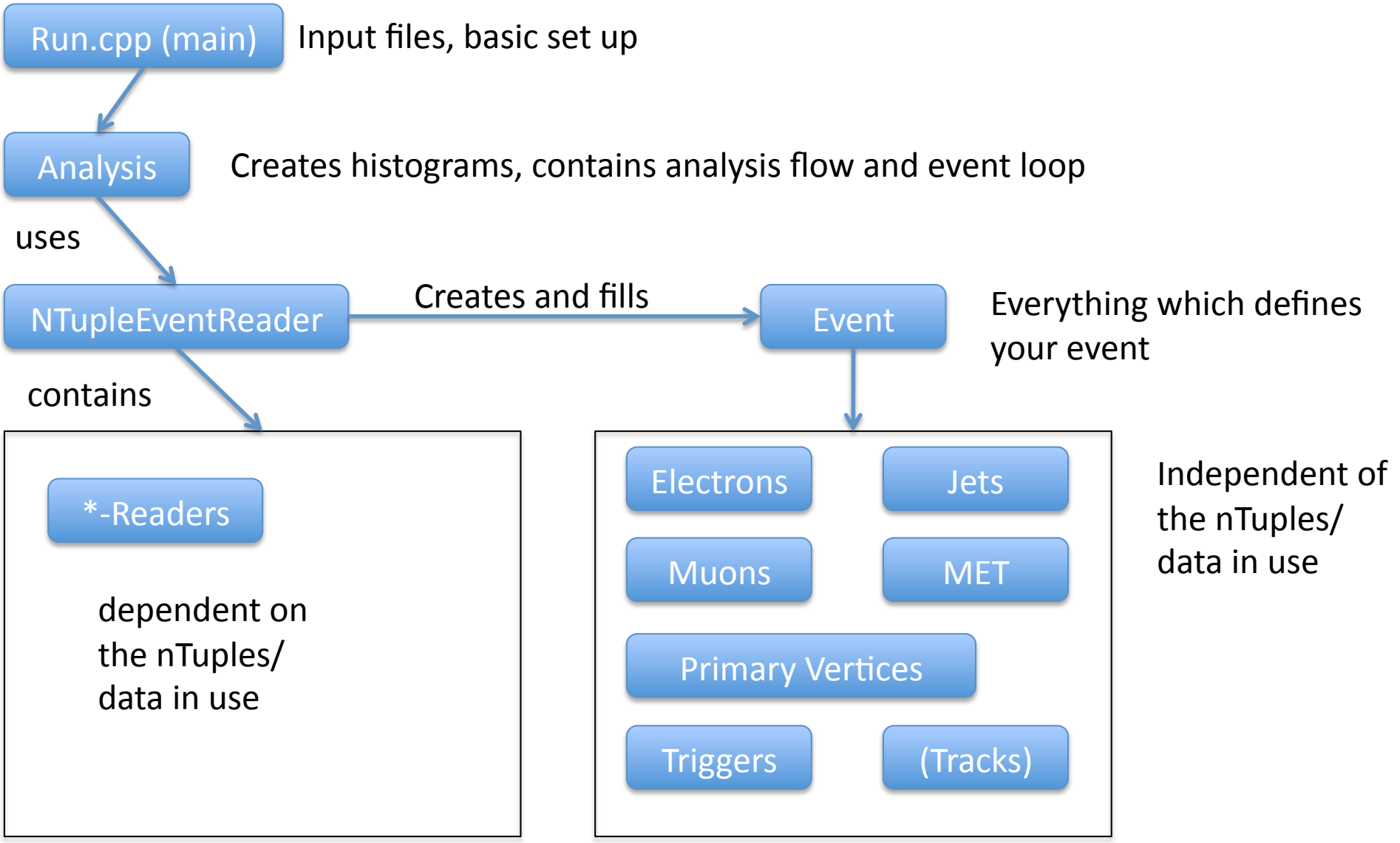
Concepts

- Object oriented approach to group functionality and data: Electron.pt(), Event.Electrons(), etc.
- Fragmentation by frequency of access:
  - Some functions (reading data, histogram creation etc) almost never change: can be >10 files(one for each purpose)
  - Particle Objects (electron, jets etc) change less frequent
  - Analysis setup (input, cuts) and the analysis itself might change more frequently: only 2-3 files
- Performance and safety
  - Don't load variables which are not used
  - Inform user if variable doesn't exist (exception)
  - Inform user if no valid input files exist

Structure

- BAT
  - bin (Analysis setup and ‘main’)
  - interface (all objects and constants)
    - HistHelpers (Containers for histograms)
    - OnlyForTests (special set used for unit tests)
    - Printers (output modules for event content and cut flow)
    - Readers (data -> object)
    - RecoObjects (Particle objects: Electron, Jet, MET, etc)
    - Taggers (ConversionTagger and Btagger)
  - scripts (pre- and post-analysis scripts)
  - src (implementation of interface)
  - test (unit tests)

Structure


Run.cpp (main)
Input files, basic set up
Analysis
Creates histograms, contains analysis flow and event loop
uses
NTupleEventReader
Creates and fills
Event
Everything which defines your event
contains
*-Readers
dependent on the nTuples/ data in use
Electrons
Jets
Muons
MET
Primary Vertices
Triggers
(Tracks)
Independent of the nTuples/ data in use

Analysis setup – bin/Run.cpp

```
int main(int argc, char **argv) {
    setUpOnce();
    TStopwatch watch;
    watch.Start();
    Analysis::useJetAlgorithm(JetAlgorithm::PF2PAT);
    Analysis::useElectronAlgorithm(ElectronAlgorithm::ParticleFlow);
    Analysis::useMuonAlgorithm(MuonAlgorithm::ParticleFlow);
    Analysis::useMETAlgorithm(METAlgorithm::ParticleFlowMET);
    Analysis::luminosity = 36.145;

    Analysis::useCustomConversionTagger(false);
    Analysis::usePFIsolation(true);

    boost::scoped_ptr<Analysis> myAnalysis(new Analysis());
    myAnalysis->setUsedNeutrinoSelectionForTopPairReconstruction
                                (NeutrinoSelectionCriterion::chi2);
    myAnalysis->addInputFile("/storage/top/data_38X/
   Nov4ReReco_JEC_Spring_V8_36.145pb_e25skim/Run2010B/*.root");
   …
    cout << "starting analysis" << endl;
    myAnalysis->analyze();
    watch.Stop();
    watch.Print();
```

bin/Analysis.cpp

```
void Analysis::analyze() {
    createHistograms();
    cout << "detected samples:" << endl;
    for (unsigned int sample = 0; sample <
   DataType::NUMBER_OF_DATA_TYPES; ++sample) {
        if (eventReader->getSeenDatatypes()[sample])
            cout << DataType::names[sample] << endl;
    }


    while (eventReader->hasNextEvent()) {
        printNumberOfProccessedEventsEvery(100000);
        initiateEvent();
        inspectEvents();
        doDiElectronAnalysis();
        doTTBarAnalysis();
        doQCDStudy();
    }
    checkForDuplicatedEvents();
    printInterestingEvents();
    printSummary();
}
```

bin/Analysis.cpp: histograms

```
void Analysis::createHistograms() {
    histMan.setCurrentLumi(Analysis::luminosity);
    histMan.prepareForSeenDataTypes(eventReader->getSeenDatatypes());
    histMan.addH1D("electron_et", "electron_et", 500, 0, 500);
    histMan.addH1D_JetBinned("diElectronMass", "diElectronMass", 1000, 0, 1000);
    histMan.addH1D_BJetBinned("mttbar_conversions", "mttbar", 5000, 0, 5000);
+ 2D
…
}

Histogram filling
void Analysis::doTTBarAnalysis() {
…
histMan.H1D_BJetBinned("mttbar")->Fill(mttbar, weight);
…
}
```

Event Class

- Offers access to all analysis object collections:

```
const PrimaryVertexPointer PrimaryVertex() const;
const TrackCollection& Tracks() const;
const ElectronCollection& Electrons() const;
const JetCollection& Jets() const;
const MuonCollection& Muons() const;
const METPointer MET() const;
```

- Sorts objects according to some properties:

```
for (unsigned int index = 0; index < allElectrons.size(); ++index) {
        ElectronPointer electron = allElectrons.at(index);

        if (electron->isGood())
            goodElectrons.push_back(electron);

        if (electron->isGood() && electron->isIsolated())
            goodIsolatedElectrons.push_back(electron);
…
}
```

- Cleans jets against isolated electrons or most isolated electron

TopPairEventCandidate

- Inherits from Event
- Adds Top selection, Top pair reconstruction
  - Reference selection: https://twiki.cern.ch/twiki/bin/viewauth/CMS/TopLeptonPlusJetsRefSel_el
- Adds definition of additional variables
  - M3, transverseMass, variables for chi2
- Gives access to reconstructed event and all possible solutions/combinations

Break for questions

After break:

- Where to find most important things
- How would I create my own analysis (i.e. Higgs search)
- Scripts
- Current code review and planned changes
- Tools which make coding easier

Where to find X?

- Input files & basic set up (which jet/electron/MET type to use): bin/Run.cpp
- Top cuts and their order
  - Order if you use the loop, N-1 function or passesFullTtbarSelection(): interface/Selection.h (will be moved to Constants.h). Otherwise you specify the order
  - Cuts: src/Filter.cpp (will be moved to TopEventCandidate and the Object files (isGood, isIsolated, etc.)

Where to find X?

- CrossSections, number of produced events and weight calculation:
  - src/CrossSections.cpp
- Constants.h
  - MC process recognition strings, jet multiplicity segmentation (used for histogram creation), list of HLTs, available algorithms (electron, jets, MET) etc.
- If you looking for something in the code and can't find it within 5 min a short email to me will get you the answer

How to create my own analysis

- If the objects/variables you need are already available (the fast way):
  - Add a new function ‘void doMyAnalysis()’ to Analysis class
  - Call the function in the event loop in Analysis::analyze()
  - Add the histograms you need in Analysis::createHistograms()
  - Short example is Analysis::doDiElectronAnalysis()

How to create my own analysis

- If the objects/variables you need are already available (the OO way):
  - If the selection is a bit longer or you have long special requirements on what you consider a good particle (i.e. electron)
  - Add Electron::isGoodForMyAnalysis(…)
  - Add new electron collection to event or create a new event class (HiggsEventCandidate) and overwrite the Event::selectElectronsByQuality() function
  - You can then add your event to the Analysis class and fill it in Analysis::initiateEvent()

How to create my own analysis

- If the objects/variables you need do not exist:
  - If the object does not exists, look first at Particle.cpp (maybe some of the functionality already exists)
  - Create your object + the corresponding Reader if needed
  - If the object exists, add the variables you need and fill them in the corresponding reader
  - Write tests to verify everything works as intended
  - Use them in Analysis.cpp or in the specific Event-class you use

The scripts

- mergeROOTFilesWithCompression.py
  - Merged root files into files just below 2GB and compresses them for faster access
  - Takes a folder as input
  - Output is created in current folder
  - Filenames: 'process'_merged_#.root
- plotMttbar.py
  - Creates plots from analysis output files
- getFileSize.py
  - Scans the file sizes of a folder. Useful if experimenting with number of lumis/events per crab job
- remove_duplicates
  - Looks for duplicate files in a folder
  - Prints and removes the duplicate files
- delete_gridFolder
  - Since 'rm –r' is not available for SE access, this script enables you do delete a whole folder

- What will change:
  - Common code between scripts will merge
  - Names will be adjusted
  - More scripts to come

Current code review

- This code was written in order to provide better readability
  - Fails in some placed, need to be fixed
  - Examples: The cut flow is done in a for-loop which is short, but non-informative as to which cuts are applied
- Should group what belongs together:
  - The selection is currently spread (Filter.cpp, TopPairCandidate.cpp, ParticleX::isGood()
  - Constants are still segmented (Constants.h, TopSelection.h, VBTFID, etc)
- Feedback is needed
  - If you indent to use the code it should be easy to use/ modify for you. What needs to be done to ensure this?

Planned changes

- Fix discoveries from the current code review
- Add more cut flow tables as it was the case in the old code + have them printed in a separate latex file which is ready to compile
- Switch to SVN
  - Will bring ticket system and hopefully change notifications for all interested.
- Change the unit tests so they work with CMSSW setup (for now only with Eclipse)

Helpful Tools

- It can be very hard to read and edit code in simple editors and even harder to navigate
- Modern (last 10 years) IDEs help a lot:
  - Eclipse (Windows, Linux, OS X)
  - Visual Studio C++ (Windows)
  - Xcode (OS X)
- The IDEs (Eclipse as an example) can:
  - Auto-complete code (if used with an instance of a class it gives a list of available functions)
  - Navigate through code by clicking on variables, functions, classes
  - Show function implementation when hovering over the call (example given next slide)
  - And of course syntax highlighting, refactoring and compile tools

Eclipse example

One reason for short functions:

```
76 void Event::selectElectronsByQuality() {
77     goodElectrons.clear();
78     goodIsolatedElectrons.clear();
79     goodPFIsolatedElectrons.clear();
80     for (unsigned int index = 0; index < allElectrons.size(); ++index) {
81         ElectronPointer electron = allElectrons.at(index);
82
83         if (electron->isGood())
84             goodElectr
85
86         if(electron->i
87             qcdElectro
88
89         if (electron->
90             goodIsolat
91
92         if(electron->a
93             if(electro
94                 goodPF
95         }
96
97         if (electron->isGood() == false && electron->isLoose())
98             looseElectrons.push_back(electron);
99     }
100 }
```

```
bool Electron::isGood(const float minEt) const {
    bool passesEt = et() > minEt;
    bool passesEta = fabs(eta()) < goodElectronMaximalAbsoluteEta && !isInCrack();

    bool passesD0 = false;
    if(usedAlgorithm == ElectronAlgorithm::Calo)
        passesD0 = fabs(d0_BS()) < goodElectronMaximalDistanceFromInteractionPoint;
    else
        passesD0 = fabs(d0()) < goodElectronMaximalDistanceFromInteractionPoint;

    bool passesDistanceToPV = fabs(zDistanceToPrimaryVertex) < 1;
    bool passesID = VBTF W70 ElectronID();
```

Press 'F2' for focus

Will be replaced by the values

How it will be for all

This method saves lookups in the specific files!

Useful links

- Current code repository:
  - http://cmssw.cvs.cern.ch/cgi-bin/cmssw.cgi/UserCode/LostMotivation/BAT/
- How to set up the AnalysisTools:
  - https://twiki.cern.ch/twiki/bin/view/CMS/BristolAnalysisTools
  - A set up guide for Eclipse will be added
- How to set up the NTupleTools:
  - https://twiki.cern.ch/twiki/bin/view/CMS/BristolAvailableNtuples
- Eclipse:
  - http://www.eclipse.org/

Summary

- The analysis framework is object oriented to group functions and data
- Most important files are bin/Run.cpp and bin/Analysis.cpp (and src/Event.cpp)
- Review in progress
- Fragmentation of important parts (mainly cuts) will be fixed
- Today's IDEs can make it very easy to manage large amount of code (even if fragmented)