Experiments Toward a Modern Analysis Environment

G. Watts (UW/Seattle)

Functional Programming Style, Scriptless, Continuous Integration, and Everything in Source Control

January 21, 2016
ACAT 2016

Declarative Programming

Plot the p_T of all the jets with $|\eta| < 2.0$

Imperative: for loop over all events:

```
For loop over all jets:
    If |jet.eta| < 2.0:
        hist.Fill(jet.pt)
```

Declarative: Events

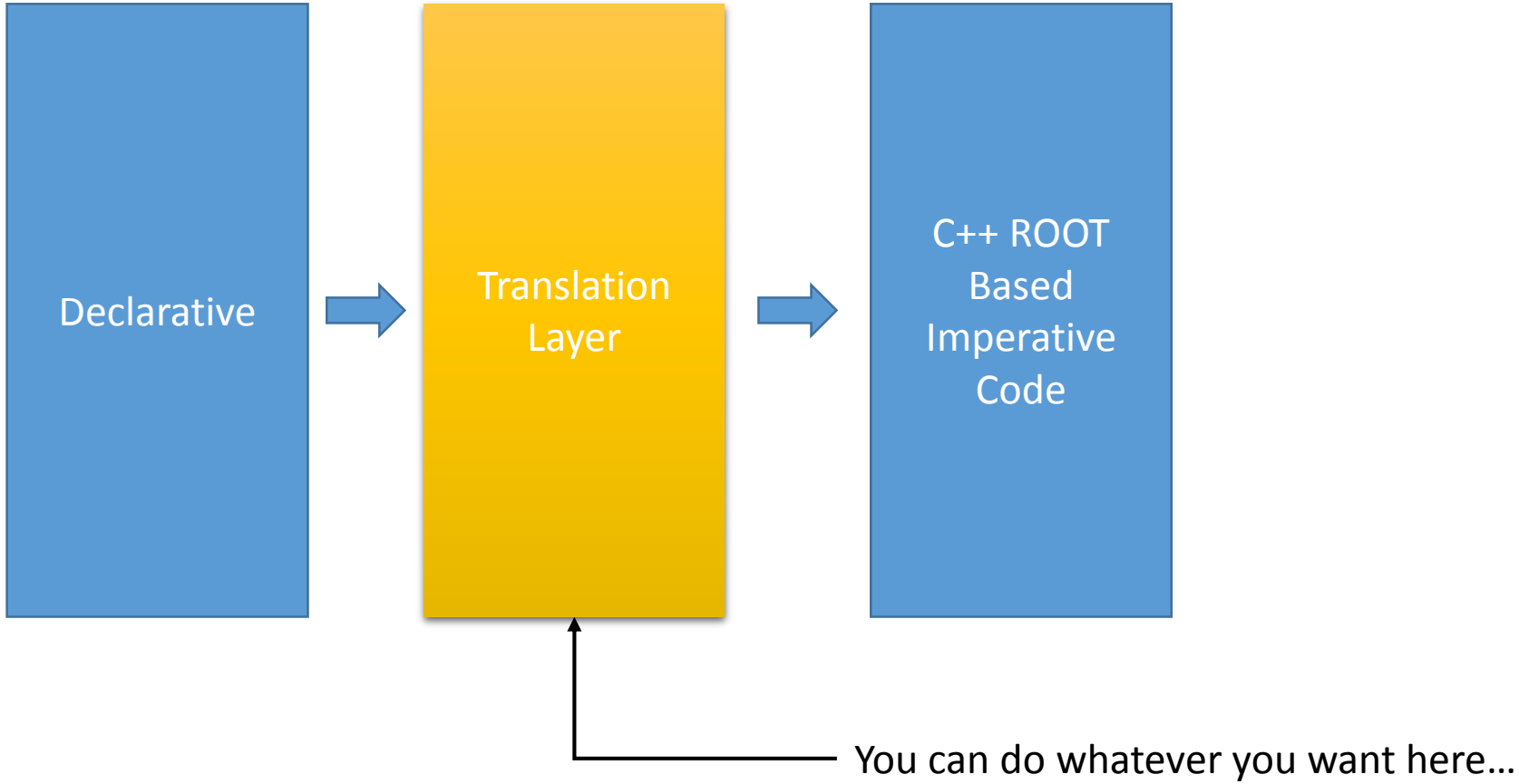
```
.SelectMany(jets)
.Where(|jet.eta| < 2.0)
.Do(hist.Fill(jet.pT));
```

Declarative Programming

Lack of ceremony!

- Trigger Lists
- Configuration for analysis chains

Translation Layer



What Can You Do?

Really… it is your imagination!

Translation Layer

Run on different backend
- Local
- Proof/Proof-lite
- Remote Linux
- Batch??

Variable Scaling
- Change units from MeV to GeV

Collect common variables
- Give flat ntuple appearance of objects
- Arrays with bounds
- Links from one list to another can be pointers to other objects

Optimize generated imperative code
- You know everything about the query
- Only activate columns that are used

Leak the abstraction
- Insert arbitrary C++ code

Manage I/O
- Fetch files from multiple locations
- Download from the GRID

Manage Results
- Cache results for faster re-running

Functional Programming

A programming paradigm

- Mathematical functions
- No mutable variables

- No global variables
- Stateless

In computer science, **functional programming** is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data

"Fluent Programming"

Functional Programming

```
var h = new NTH1F("h1", "Hist", 100, 0.0, 10.0)
                .XaxisTitle("p_T [GeV]")
                .YaxisTitle("N/1 GeV");
```

- Function Chaining
- Argument currying
- Monads
- Unmutable State

```
public static NTH1F XaxisTitleC(this NTH1F h, string title)
        {
            h = h.Clone() as NTH1F;
            h.Xaxis.SetTitle(title);
            return h;
        }
```

```
var firstValue = Range(0, r.NbinsX)
        .Where(bin => greater ? r.GetBinContent(bin) > bv : r.GetBinContent(bin) < bv)
        .FirstOrDefault();
```

A Loop!

There are pitfalls

ROOT is horribly tunable!

This framework has opinions → Choices are limited

Leaky Abstractions

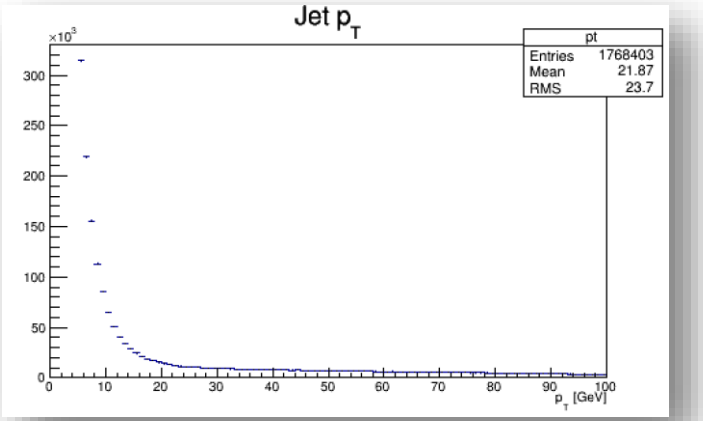
Many tools in particle physics do not have a functional API

- Anything that returns an error status code is not functional!
- Methods that self-modify are not functional

Optimization

Sample Plotting of Jet Pt

Runs over 493,000 events on Windows Laptop




```
static void Main(string[] args)
        {
            var dataset = "user.emmat.mc15_13TeV…2__EXOT15_v3_EXT0";
            var jobFiles = GRIDJobs.FindJobFiles("DiVertAnalysis", 3, dataset);

            var events = QueryablerecoTree.CreateQueriable(jobFiles);

            using (var f = new FutureTFile("test.root"))
            {
                events
                    .SelectMany(e => e.Jet)
                    .Where(j => Math.Abs(j.eta) < 1.5)
                    .FuturePlot("pt", "Jet pT; pT [GeV]", 100, 0.0, 100.0, j => j.pT)
                    .Save(f);
            }
```

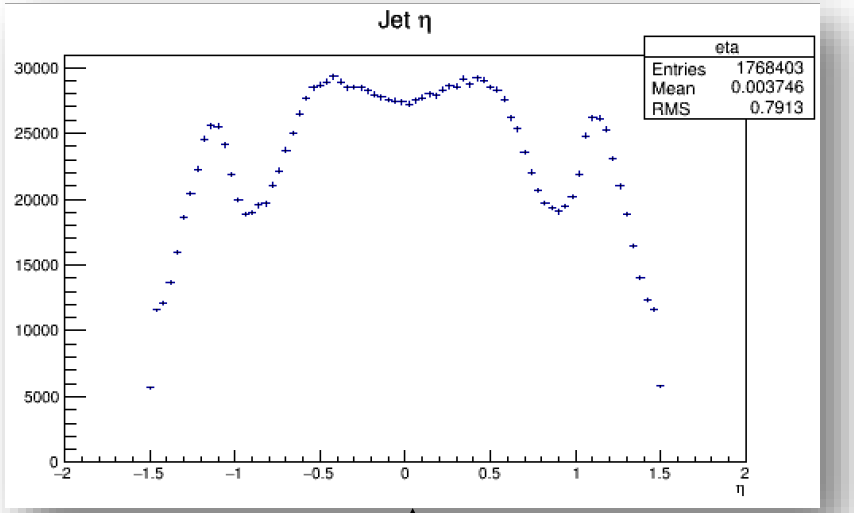
Setup

Plot jet p_T

Save the plot

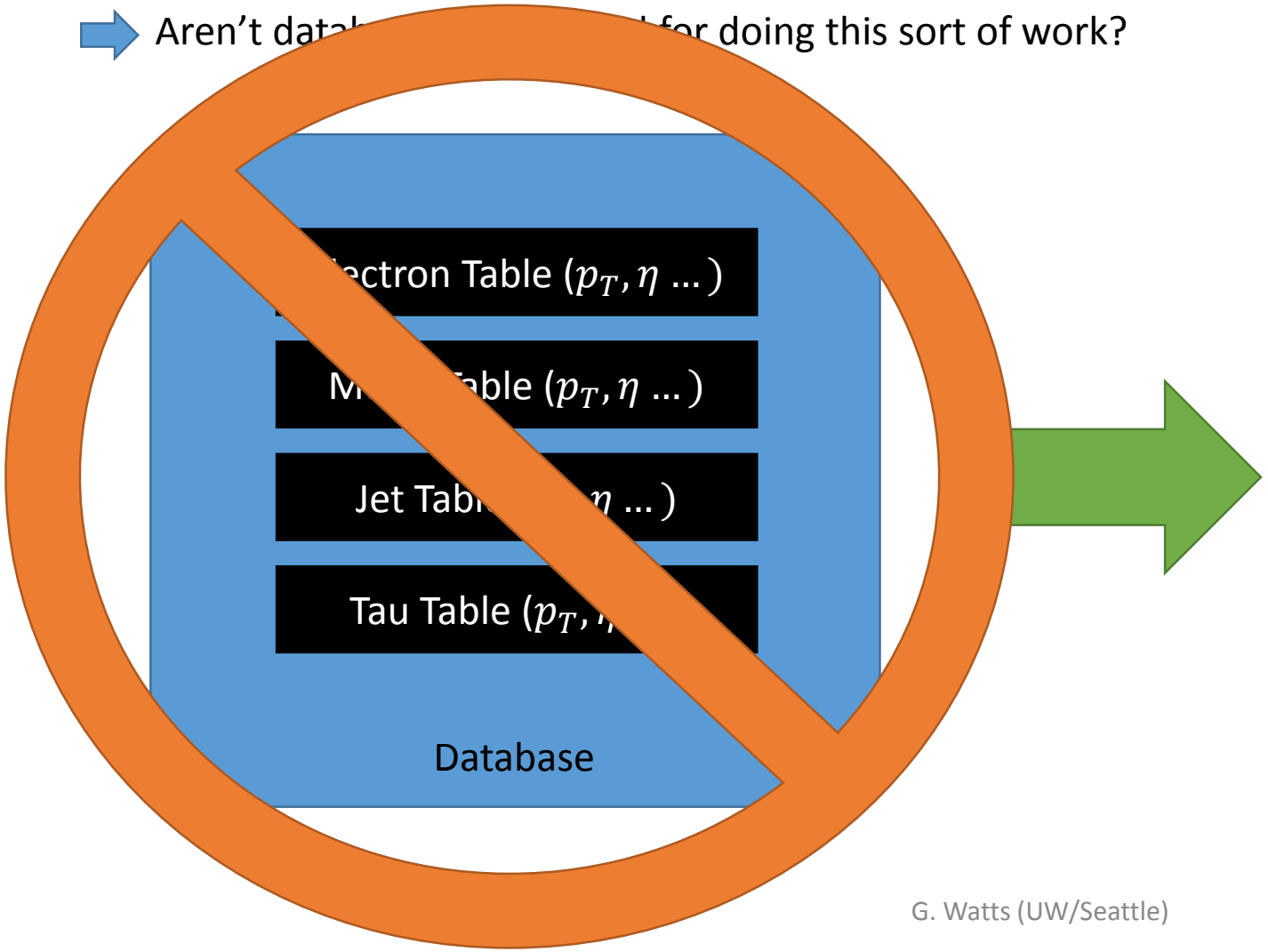
Going functional to also plot η

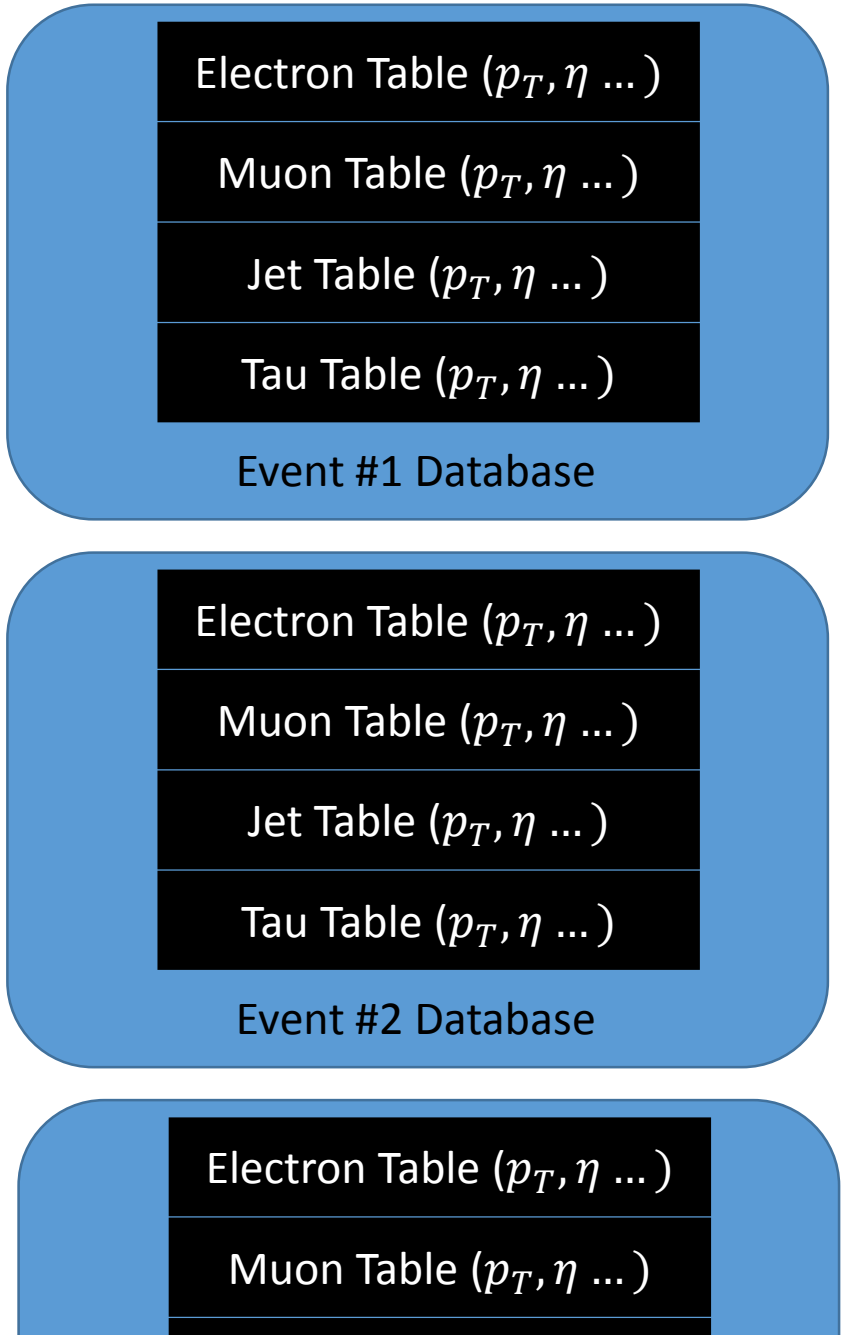
```
var events = QueryablerecoTree.CreateQueriable(jobFiles);

using (var f = new FutureTFile("S2.root"))
{
    var count = events.FutureCount();

    var jets = events
        .SelectMany(e => e.Jet)
        .Where(j => Math.Abs(j.eta) < 1.5);

    jets
        .FuturePlot("pt", "Jet pT; pT [GeV]", 100, 0.0, 100.0, j => j.pT)
        .Save(f);

    jets
        .FuturePlot("eta", "Jet eta; eta", 100, -2.0, 2.0, j => j.eta)
        .Save(f);

    Console.WriteLine($"Saw {count.Value} events.");
}
```



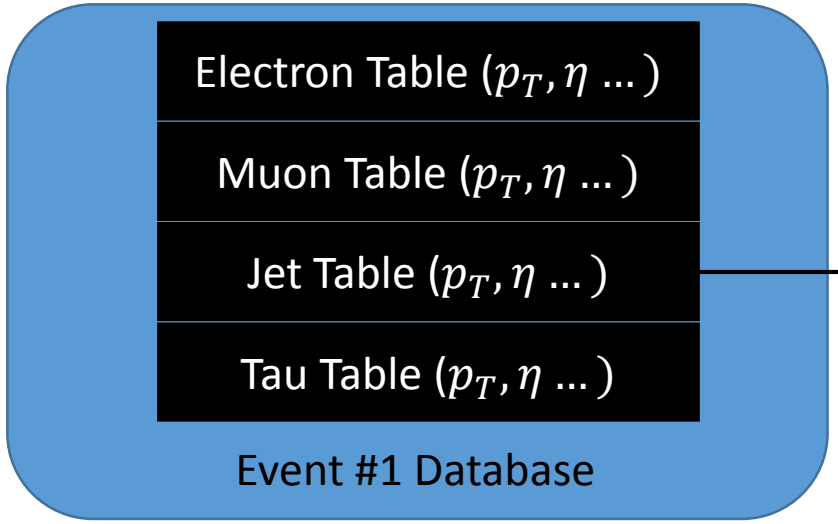
That looks like SQL!

- Aren't data[illegible] for doing this sort of work?





That looks like SQL!




```
events
  .SelectMany(e => e.Jet)
  .Where(j => Math.Abs(j.eta) < 1.5)
  .FuturePlot("pt", "Jet pT; pT [GeV]", 100, 0.0, 100.0, j => j.pT)
  .Save(f);
```

Like a little min-query over the data in each event

Goals

I am a professor

I have a low duty cycle
I would like things to just work
I don't mind reduced choice if I can get plots faster as a result

1. Encourage Exploration

   Make it easy to build a new plot or try some sort of new algorithm

2. Easily see what the analysis is

   What selection cuts went into that plot?

3. Capable of running on large datasets, efficiently

   ROOT and C++ for the event code

4. Reproducible

   Make the analysis automated

Some of these goals have evolved as the analysis has grown

Environment

- A “C++-light”, language.
  - No multiple inheritance
  - Simplified template capability
  - Managed, garbage collected language
- Functional Extensions

- Not the first choice for HEP for analysis

- Implies a set of tools and utilities I must use for analysis

Environment

ROOT
Data Analysis Framework

- Not possible to do a modern analysis without it.
- Currently ROOT 5 only (Windows!)

Get Rid Of It As Fast As We Can

- Have to reproduce the entire I/O sub-system
- All data files have to be translated

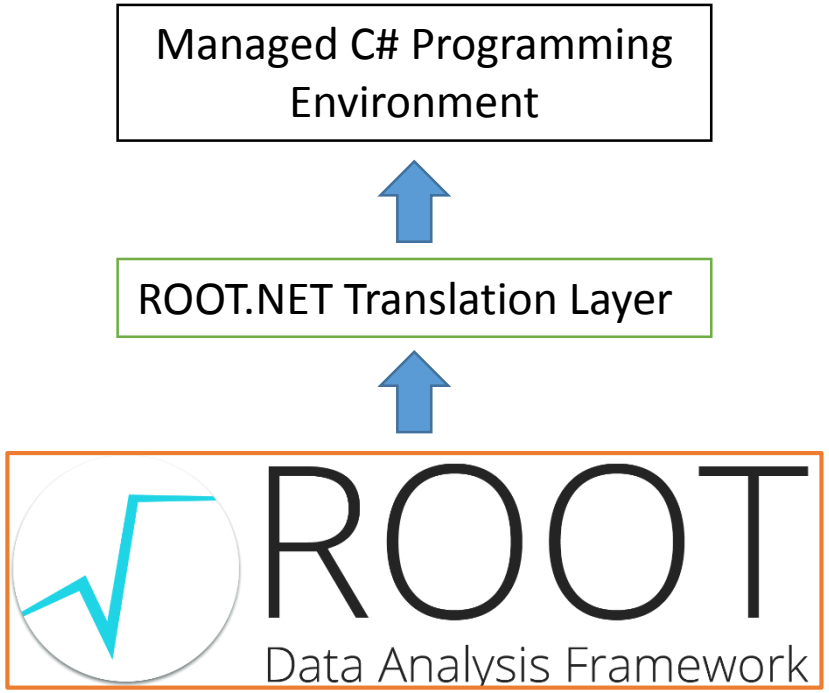
Work With (around) it

- Hide most of ROOT boiler plate.
- Expose most of ROOT on an as-need basis.

Environment

- Uses ROOT/CINT's metadata system to discover all objects and methods.
- Uses *managed C++* to provide a high speed translation layer between the managed and unmanaged world
- Does proper reference counting to make efficient use of memory.

https://github.com/gordonwatts/ROOT.NET



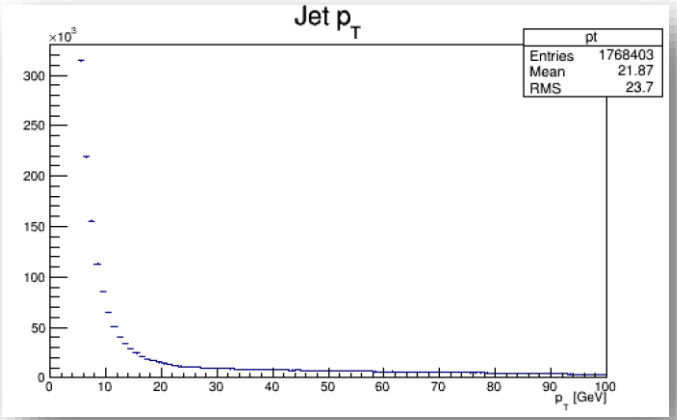
Efficiency

Runs over 493,000 events on Windows Laptop

```
static void Main(string[] args)
        {
            var dataset = "user.emmat.mc15_13TeV…2__EXOT15_v3_EXT0";
            var jobFiles = GRIDJobs.FindJobFiles("DiVertAnalysis", 3, dataset);

            var events = QueryablerecoTree.CreateQueriable(jobFiles);

            using
            {
                e
                .Where(j => Math.Abs(j.eta) < 1.5)
                .FuturePlot("pt", "Jet pT; pT [GeV]", 100, 0.0, 100.0, j => j.pT)
                .Save(f);
            }
```

Takes less than 4 seconds running on modern Surface Pro 4

- Core i5, lots of memory and SSD
- Most of time is compiling C++



Setup

Plot jet p_T

Save the plot

Efficiency

There is no database!

Aspect 1 Use ROOT & C++ to the fullest possible extent

Use transition layer to generate C++ against the raw ntuple

All inside a TSelector

```
int aInt32_3 = (*(*this).CalibJet_E).size();
for (int aInt32_2=0; aInt32_2 < aInt32_3; aInt32_2++)
{
  double aDouble_4=std::abs((double)(*(*this).Jet_eta).at(aInt32_2));
  if (aDouble_4<1.5)
  {
    int aInt32_7=(*aNTH1F_5).Fill((*(*this).Jet_pT).at(aInt32_2),1.0);
    aInt32_7;
  }
}
```

Loop over all jets

Test on η

Plot jet p_T

Big Bonus: Only load leaves that we need

Efficiency – Multiple Plots

```
int aInt32_3 = (*(*this).CalibJet_E).size();
for (int aInt32_2=0; aInt32_2 < aInt32_3; aInt32_2++)
{
  double aDouble_4=std::abs((double)(*(*this).Jet_eta).at(aInt32_2));
  if (aDouble_4<1.5)
  {
    int aInt32_13=(*aNTH1F_11).Fill((*(*this).Jet_eta).at(aInt32_2),1.0);
    aInt32_13;
    int aInt32_7=(*aNTH1F_5).Fill((*(*this).Jet_pT).at(aInt32_2),1.0);
    aInt32_7;
  }
}
aInt32_1=aInt32_1+1;
```

Plot jet p_T

Plot jet η

Count # of events

Efficiency & Exploration

Aspect 2 If you have already made the plot, don't remake it

Keep a cache of all plots

If you've already made the plot → Return the same plot as last time without re-running on the data!

If not →
1. Generate C++ to make the plot(s)
2. Compile C++
3. Run over the complete data, reading the full ntuple

Say your app makes 500 plots, and you want to add one more

- Making 500 plots requires 1 hour of running
- Making 501 plots requires 1 hour + ϵ of running
- Making 1 plot requires less than a minute ← Cache of 500 plots means you can take this option!

Automated Analysis

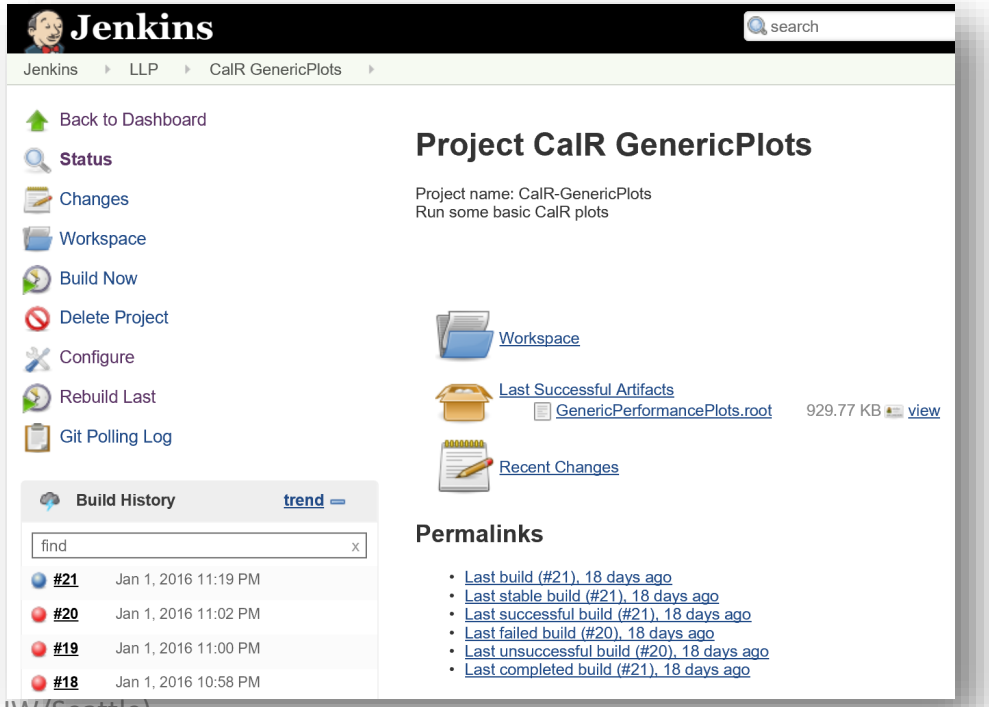
Continuous Integration

Re-run all your tests every single time you modify your software

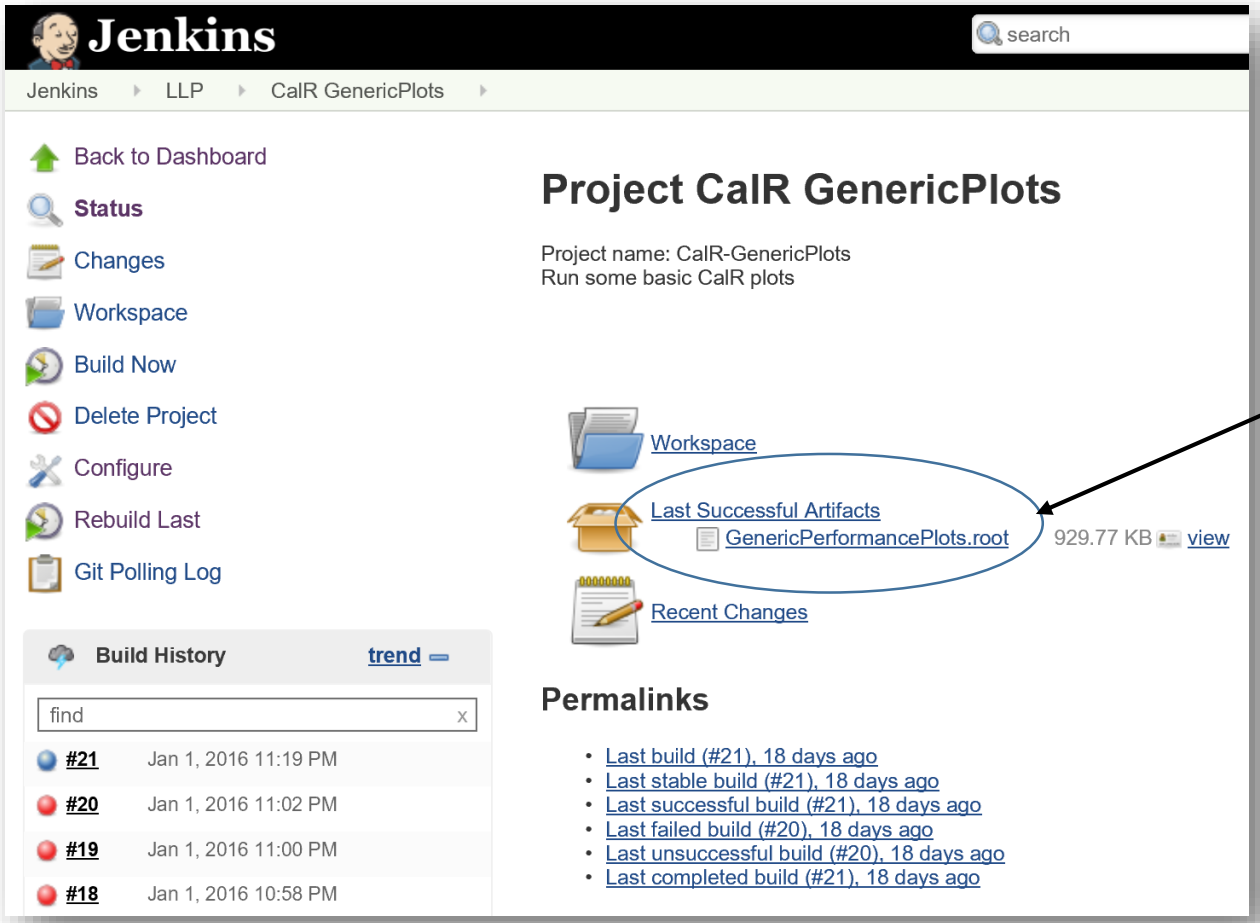
Re-run your physics analysis each time you have a change

Jenkins

- Open source software
- Monitors source control repository
- Kicks off arbitrary script each time it detects a change
- Archives a set of *results* from the run



Automated Analysis

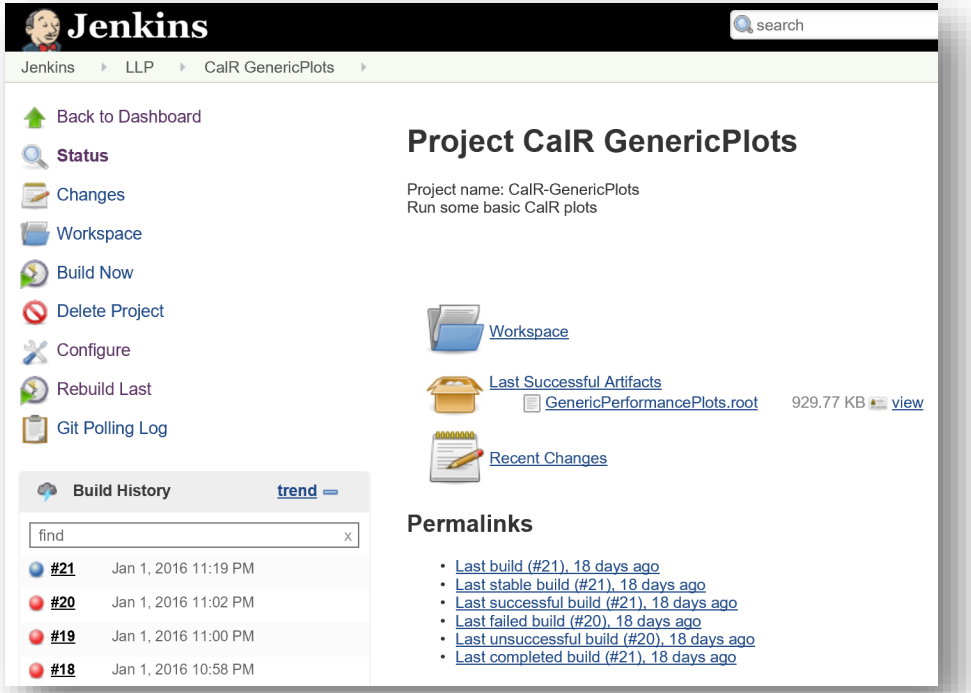


ROOT file containing the results

A text log file is also archived

This information is kept as long as you want

Automated Analysis



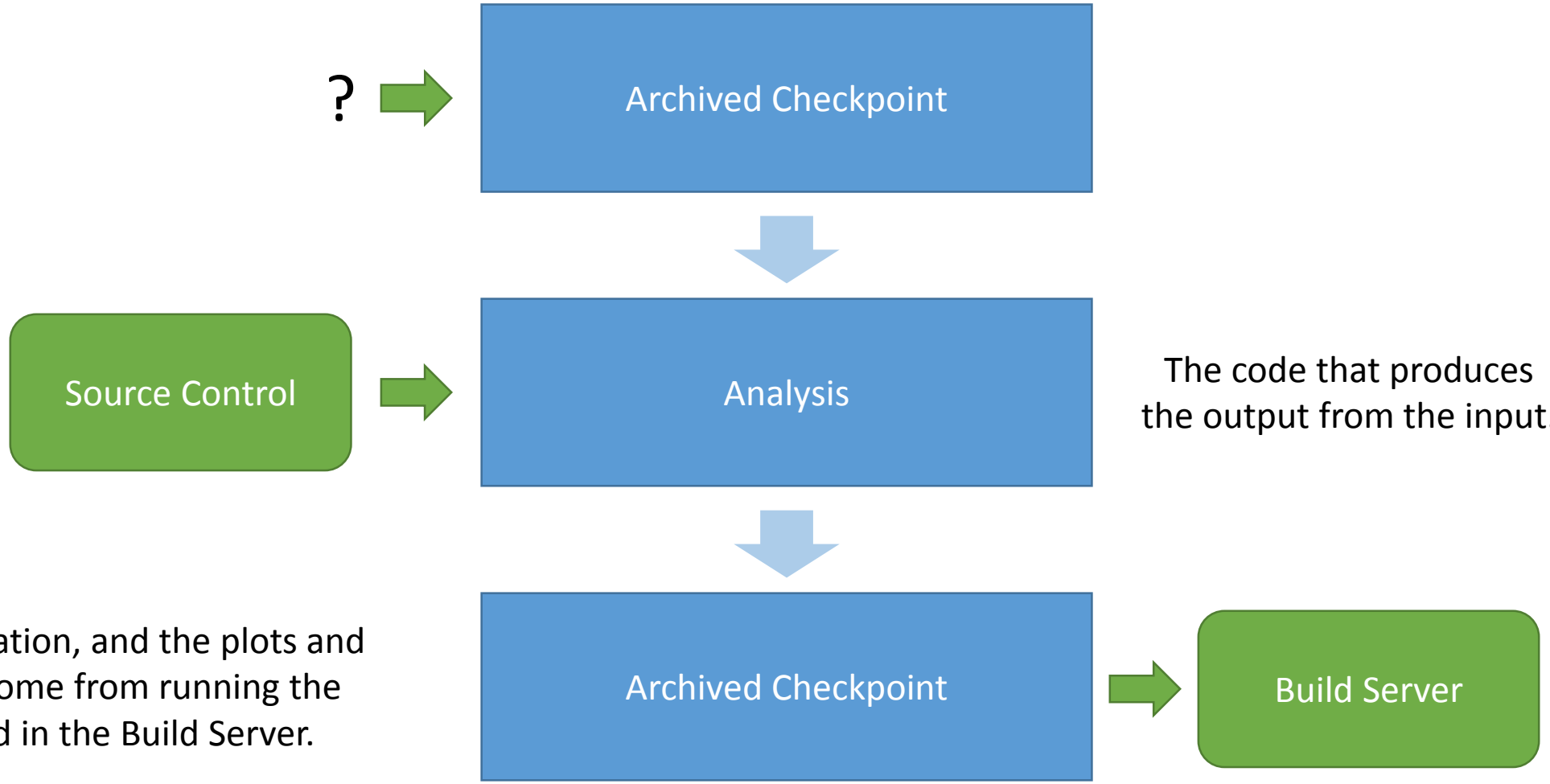
What does it take to make a physics analysis behave this way?

- Repeated runs must be fast. Minutes at most.
- Results must be a reasonable size.

The cached results enable exactly this!

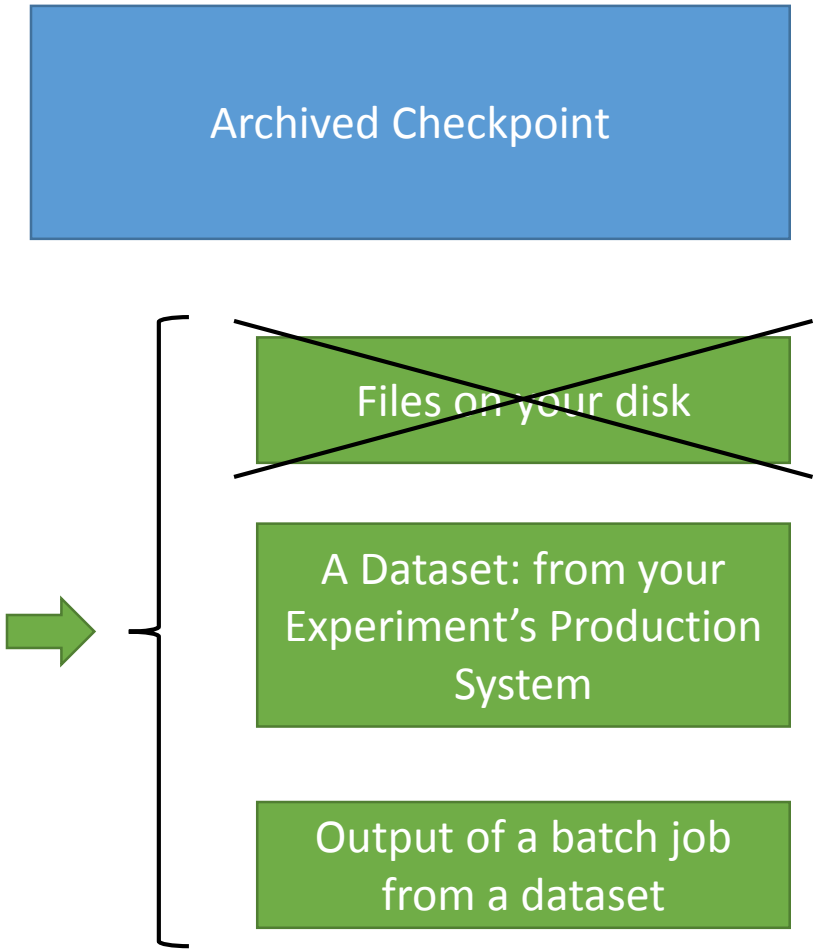
Build machines have a web API which allows automatic downloading of results

Reproducible



The log files, run configuration, and the plots and output ROOT files that come from running the Analysis code. Archived in the Build Server.

The Job Input



Remember how you made those 6 months later?

In ATLAS, this means the data is located on the GRID.

- Dataset name is unique for lifetime of ATLAS
- Can only be downloaded via Linux

In ATLAS, this means the data is located on the GRID.

- User dataset name is unique for lifetime of ATLAS
- Job can only be submitted on Linux
- Can only be downloaded via Linux

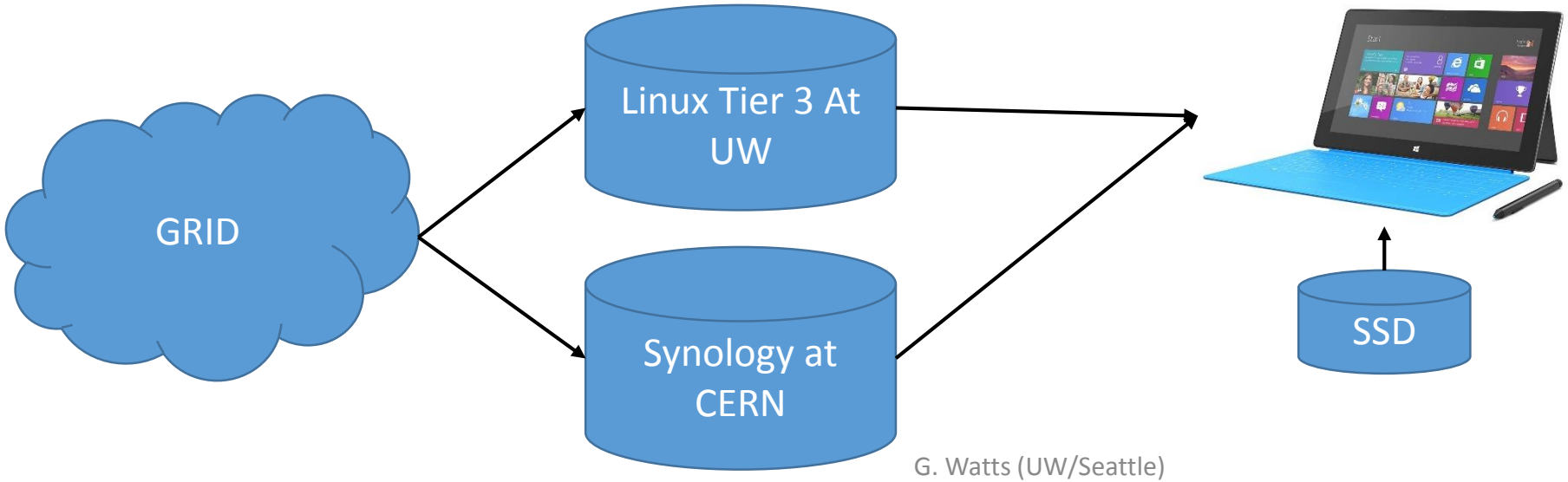
Location Independence

A Dataset: from your Experiment's Production System

Output of a batch job from a dataset

I want to run:

- At 30,000 feet
- In my office at UW
- At CERN
- On one or two files of the dataset
- On multi-TB datasets
- Automatically from build server
- PROOF server (Linux based)



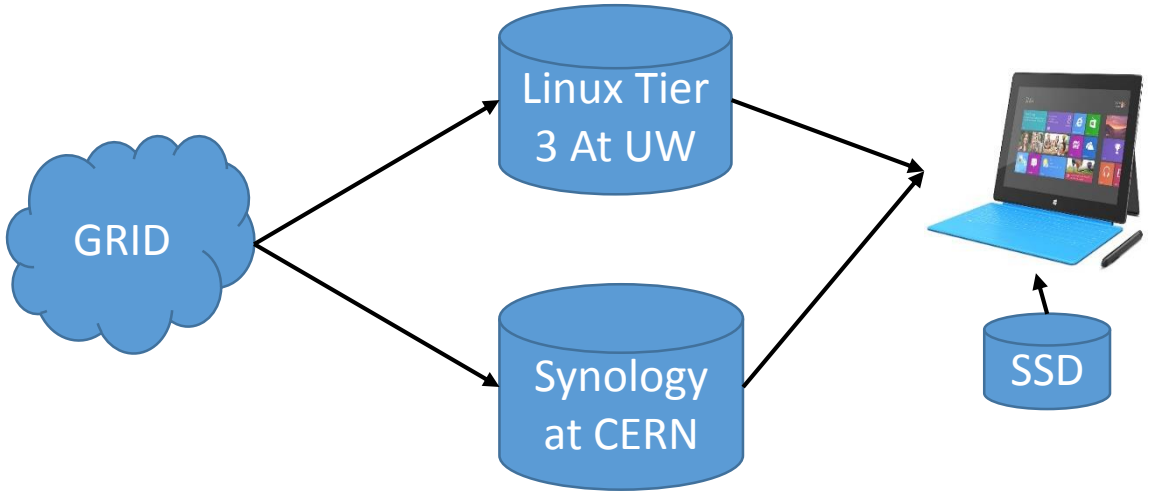
AtlasSSH

https://github.com/LHCAtlas/AtlasSSH

A small project!

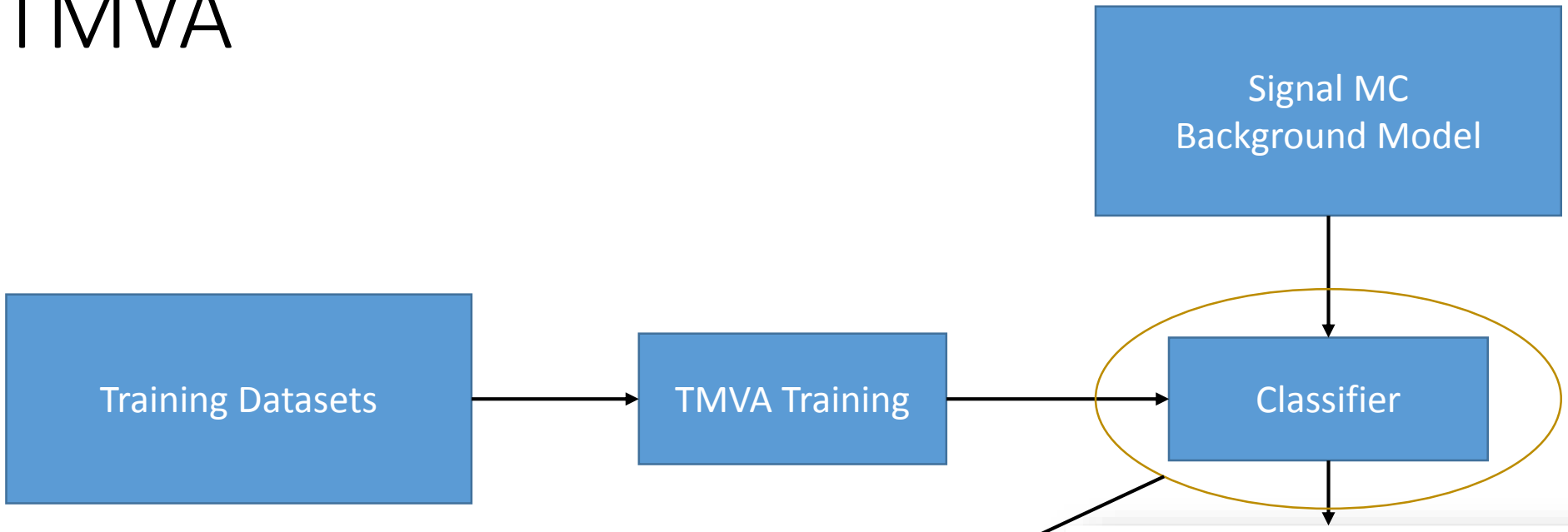
Files Can Be Located:

- On Linux via SMB
- On the local disk
- In the GRID to be copied to one of the above locations
- As the result of a GRID job it must submit
- C# & Powershell interface

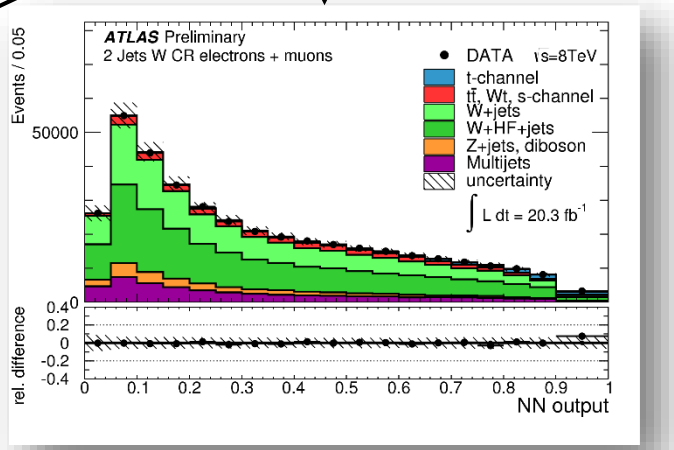



```
var job = new AtlasJob()
                .NameVersionRelease("DiVertAnalysis", 3, "2.3.32")
                .Package("JetSelectorTools")
                .Package("atlasphys-exo/Physics/…/AnalysisCode/trunk/DiVertAnalysis", "248132")
                .Command("grep -v \"emf < 0.05\" …/JetCleaningTool.cxx > …/JetCleaningTool-New.cxx")
                .Command("mv …/JetCleaningTool-New.cxx …/JetCleaningTool.cxx")
                .SubmitCommand("DiVertAnalysisRunner -EventLoopDriver GRID *INPUTDS* -ELGRIDOutputSampleName
*OUTPUTDS* -WaitTillDone FALSE -isLLPMC true");
```

TMVA



Must be cached for future runs!



Why Not C++?

I want to be able to run on multiple platforms:

- Windows for testing and development and debugging
- Cluster of computers for large dataset running (PROOF)

Cross compilation required

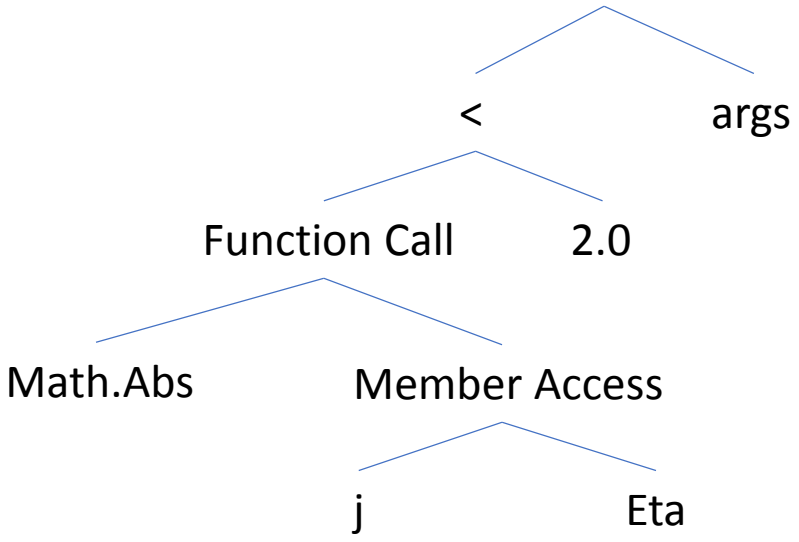
```
var p1 = data
    .SelectMany(evt => evt.Jets)
    .Where(j => Math.Abs(j.Eta) < 2.0)
    .Plot("jetpT", "Jet pT", 100, 0.0, 100.0, j => j.Pt / 1000.0);
```

This is a C# lambda and must be translated into C++

Why Not C++?

```
var p1 = data
    .SelectMany(evt => evt.Jets)
    .Where(j => Math.Abs(j.Eta) < 2.0)
    .Plot("jetpT", "Jet pT", 100, 0.0, 100.0, j => j.Pt / 1000.0);
```

j => Math.Abs(j.Eta) < 2.0 → lambda



C# makes code as data when requested

- Data structure is easily iterated over
- Sub expression extraction and optimizations easy
- Support for full expressions
- No support for multi-line statements (C# language limitation)
  - New compiler may have changed that

https://github.com/gordonwatts/LINQTOTreeACAT2016Samples

https://github.com/gordonwatts/LINQtoROOT

Conclusions

- This has been used in a published analysis
  - QCD background study
- Problems ahead
  - Leaky abstraction
  - Ability to accommodate things like TMVA
  - Cache mechanism starts to take real time with more than 500 plots.
  - Inefficient queries like "Number of tracks with $\Delta R < 0.2$ near jet axis"
- Future Development Plans
  - Finish the TMVA work
  - Still working on each plot containing its complete history
  - Plot manipulation is sub optimal
  - Build a POD backend
  - Jobs must contain multiple steps
  - What about Notebooks?
- How can this transition to a more mainstream project?
  - C# is going open source and is actively supported on Linux now… But…