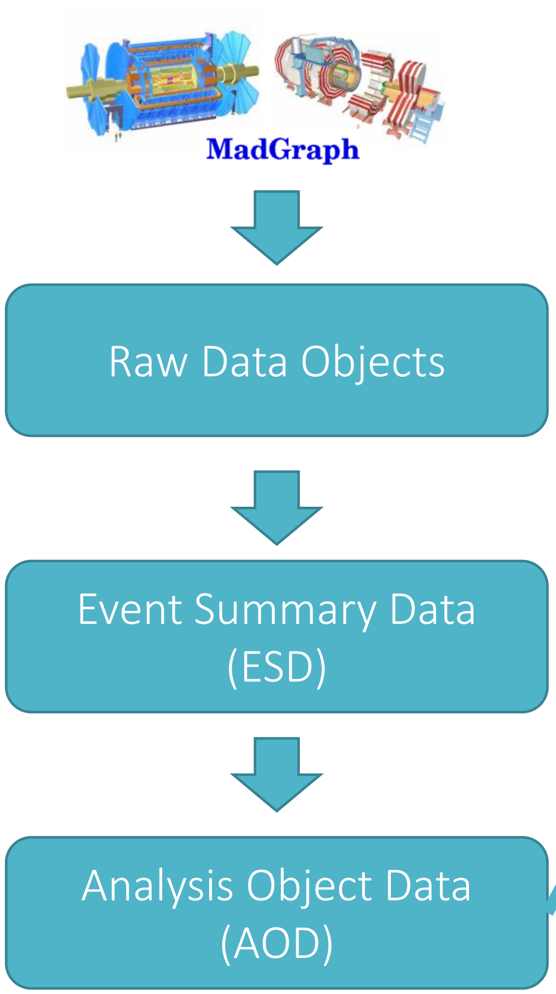
Declarative Analysis Languages

G. Watts (UW/Seattle)

CHEP 2018
Sofia Bulgaria
July 9-13, 2018

What is an Analysis Language?

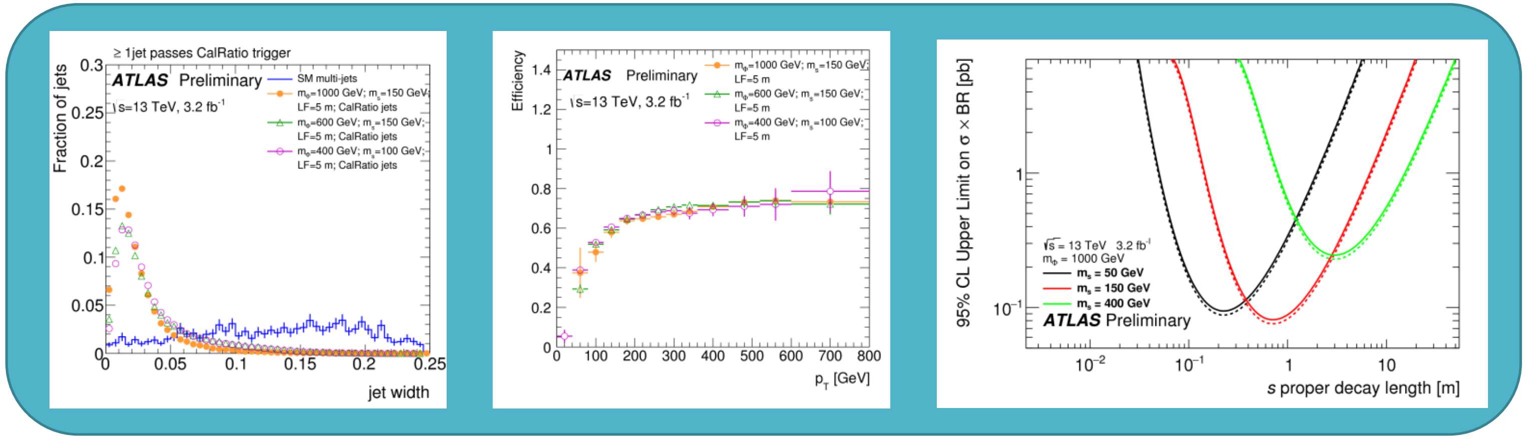


MadGraph

Raw Data Objects

Event Summary Data (ESD)

Analysis Object Data (AOD)



Analysis Language Tasks:

- Generate plots from a dataset
- Ratios from different datasets (efficiencies, etc.)
- Statistical Analysis (limit plots)
- Machine Learning
- Tables of numbers for publication

In use today:

- C++, ROOT
- Python, data science tools (including ROOT)
- Long tail: C#, Go, DSL's etc.

Can We Do Better?

What would a language look like explicitly designed for particle physics data?

A language that explicitly supported **both fast exploration and slower production?**

A language that could easily **scale from your laptop to a cluster** with minimal change (or knowledge) by the analyzer?

A language that had a **minimal amount of boiler plate**?

I tried…

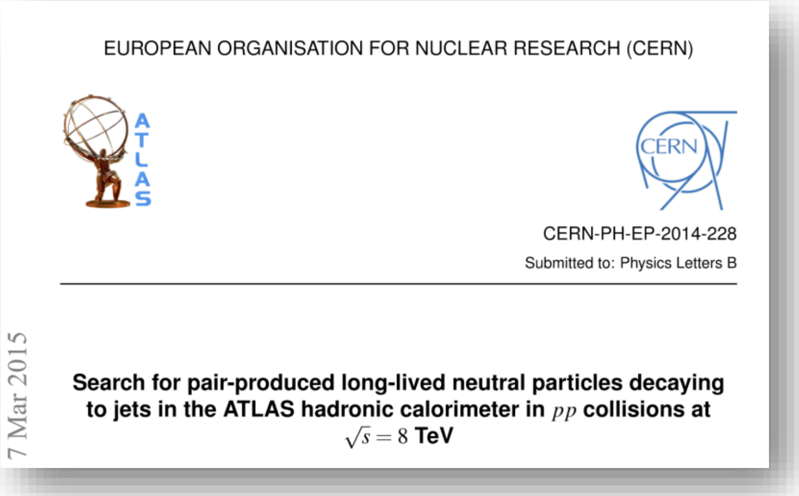
I'd like to think I've learned some lessons…

The Physics: Search for a long lived particle decaying in the calorimeter

- Unique signature
- Weird calibration
- Messy QCD background

What was done in this framework?

- QCD Background studies
- Studies to select LLPs
- Eventual BDT training using TMVA



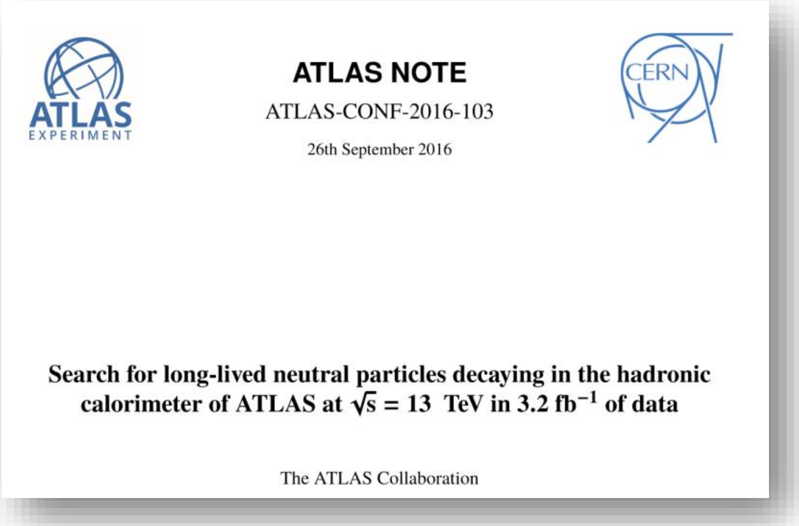
EUROPEAN ORGANISATION FOR NUCLEAR RESEARCH (CERN)

CERN-PH-EP-2014-228

Submitted to: Physics Letters B

7 Mar 2015

Search for pair-produced long-lived neutral particles decaying to jets in the ATLAS hadronic calorimeter in pp collisions at $\sqrt{s} = 8$ TeV



ATLAS NOTE

ATLAS-CONF-2016-103

26th September 2016

Search for long-lived neutral particles decaying in the hadronic calorimeter of ATLAS at $\sqrt{s}$ = 13 TeV in 3.2 fb^{-1} of data

The ATLAS Collaboration

Sense of Scale

Number of events (signal + background + control)	→	~ 2 billion
GRID Data Samples	→	~ 200
Size of input files	→	1-2 TB
Number of leaves in processed ntuples	→	~340
Plots made per job	→	~400-800
Number of users	→	1
Number of Developers	→	1

A small analysis by HL-LHC standards…

But a decent sized one for Run 1 and Run 2

Physics Data

Event 1

Event 2

Event 3

Event 4

Event 5

Each event is a small hierarchical structured collection of data:

- Some # of electrons
- Some # of muons
- Missing ET value
- Run #
- Etc.

What do we want to ask of the data?

- Invariant mass of two highest p_T good electrons
- Etc.

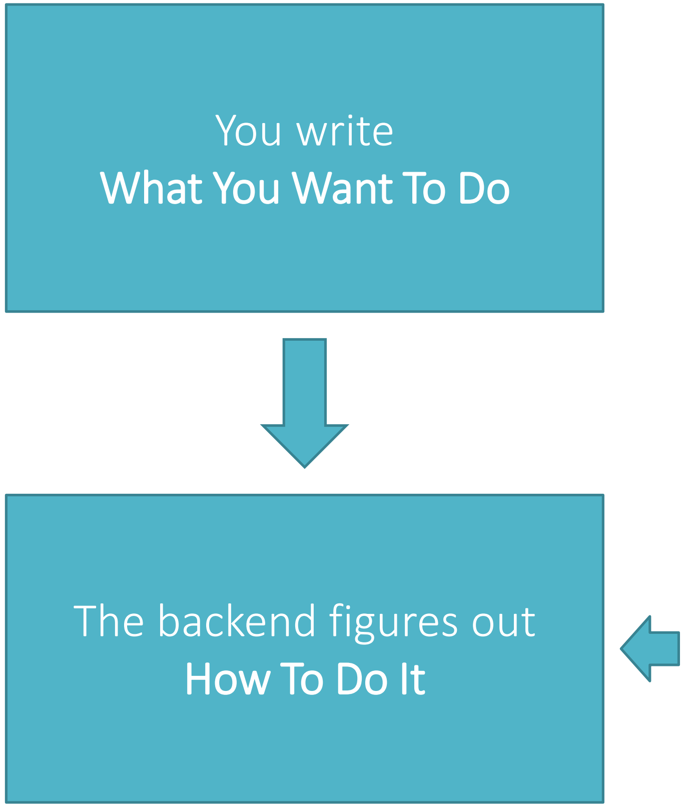
These are very SQL like questions!!

Take a ‘query’ and repeat it for each event!

Declarative Analysis

“The problems with 1000 events are different from the problems with 1M events are different from the problems with 1B events”

But you still want to make the same plots…



You write
What You Want To Do

The backend figures out
How To Do It

SQL Is a Declarative Language

```
SELECT * FROM
Production.Product
ORDER BY Name ASC;
```

Switch the backend out for 1B events vs 1M events!

C# and LINQ

I chose Microsoft’s C# language due to built in SQL-like language, LINQ:

- Strongly typed
- LINQ is extensible to new backends by design
- Automatic tooling support
- Fully capable language with lots of Open Source libraries
- Statically typed

```
events
   .Select(e => e.Data.eventWeight)
   .FuturePlot("event_weights", "Sample EventWeights",
      100, 0.0, 1000.0)
   .Save(hdir);
```

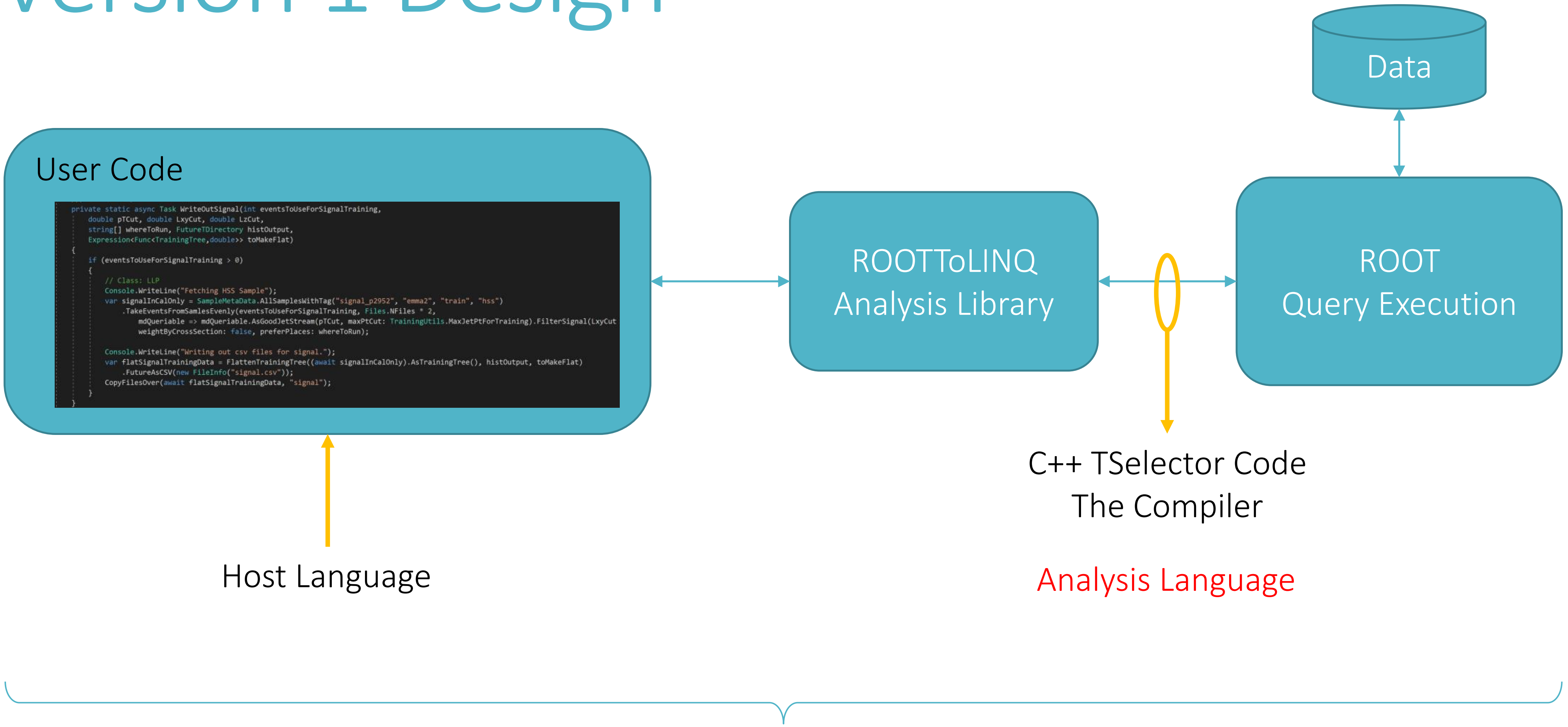
What we want to plot

1D Histogram Declaration

Save the plot in a file

Note: There is no explicit loop!

Version 1 Design



User Code

```
private static async Task WriteOutSignal(int eventsToUseForSignalTraining,
    double pTCut, double LxyCut, double LzCut,
    string[] whereToRun, FutureTDirectory histOutput,
    Expression<Func<TrainingTree,double>> toMakeFlat)
{
    if (eventsToUseForSignalTraining > 0)
    {
        // Class: LLP
        Console.WriteLine("Fetching HSS Sample");
        var signalInCalOnly = SampleMetaData.AllSamplesWithTag("signal_p2952", "emma2", "train", "hss")
            .TakeEventsFromSamlesEvenly(eventsToUseForSignalTraining, Files.NFiles * 2,
                mdQueriable => mdQueriable.AsGoodJetStream(pTCut, maxPtCut: TrainingUtils.MaxJetPtForTraining).FilterSignal(LxyCut
                weightByCrossSection: false, preferPlaces: whereToRun);

        Console.WriteLine("Writing out csv files for signal.");
        var flatSignalTrainingData = FlattenTrainingTree((await signalInCalOnly).AsTrainingTree(), histOutput, toMakeFlat)
            .FutureAsCSV(new FileInfo("signal.csv"));
        CopyFilesOver(await flatSignalTrainingData, "signal");
    }
}
```

Host Language

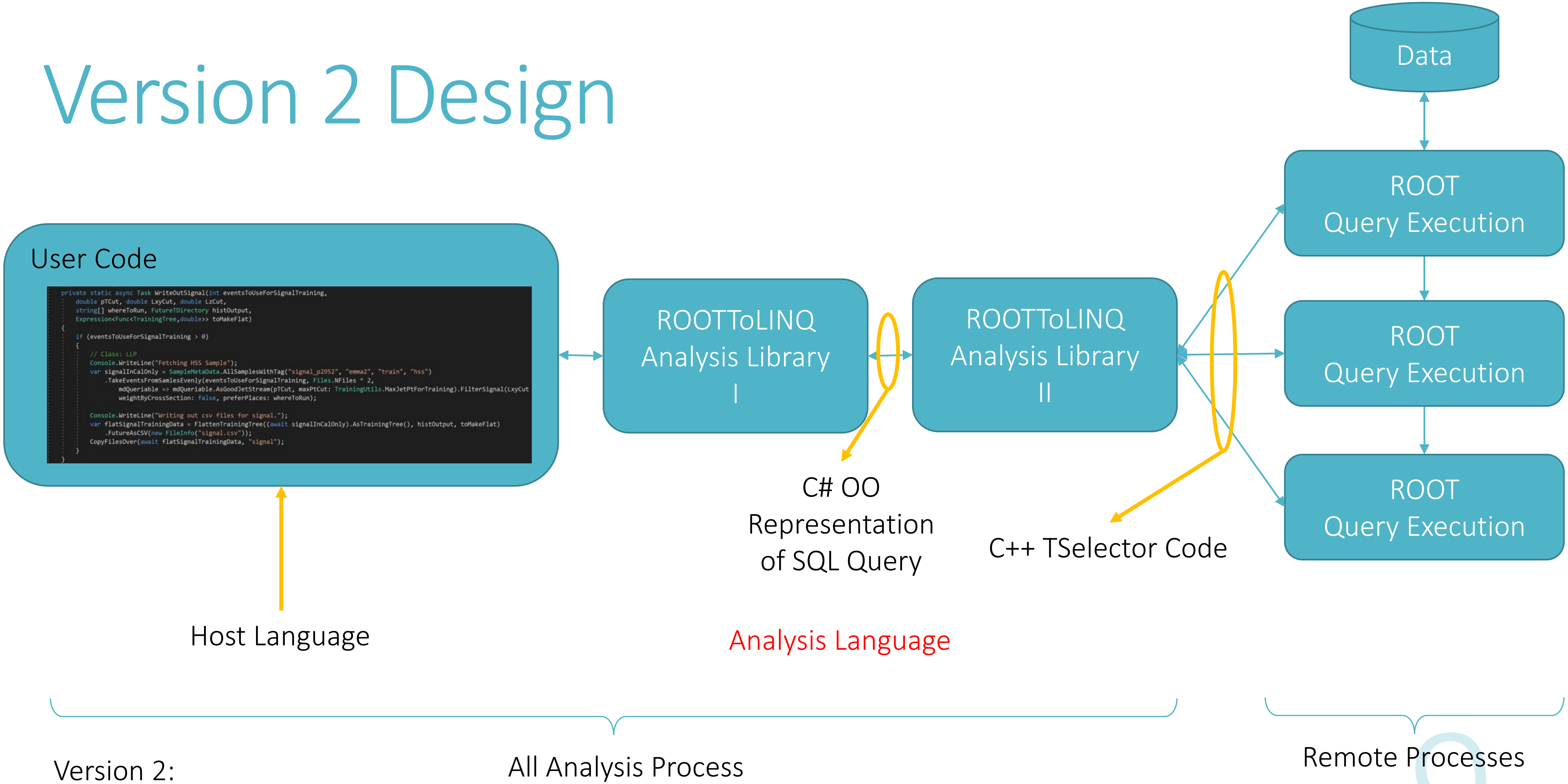
C++ TSelector Code
The Compiler

Analysis Language

All In Process

G. WATTS (UW/SEATTLE)

Version 2 Design



```
private static async Task WriteOutSignal(int eventsToUseForSignalTraining,
    double pTCut, double LxyCut, double LzCut,
    string[] whereToRun, FutureTDirectory histOutput,
    Expression<Func<TrainingTree,double>> toMakeFlat)
{
    if (eventsToUseForSignalTraining > 0)
    {
        // Class: LLP
        Console.WriteLine("Fetching HSS Sample");
        var signalInCalOnly = SampleMetaData.AllSamplesWithTag("signal_p2952", "emma2", "train", "hss")
            .TakeEventsFromSamlesEvenly(eventsToUseForSignalTraining, Files.NFiles * 2,
                mdQueriable => mdQueriable.AsGoodJetStream(pTCut, maxPtCut: TrainingUtils.MaxJetPtForTraining).FilterSignal(LxyCut
                weightByCrossSection: false, preferPlaces: whereToRun);

        Console.WriteLine("Writing out csv files for signal.");
        var flatSignalTrainingData = FlattenTrainingTree((await signalInCalOnly).AsTrainingTree(), histOutput, toMakeFlat)
            .FutureAsCSV(new FileInfo("signal.csv"));
        CopyFilesOver(await flatSignalTrainingData, "signal");
    }
}
```

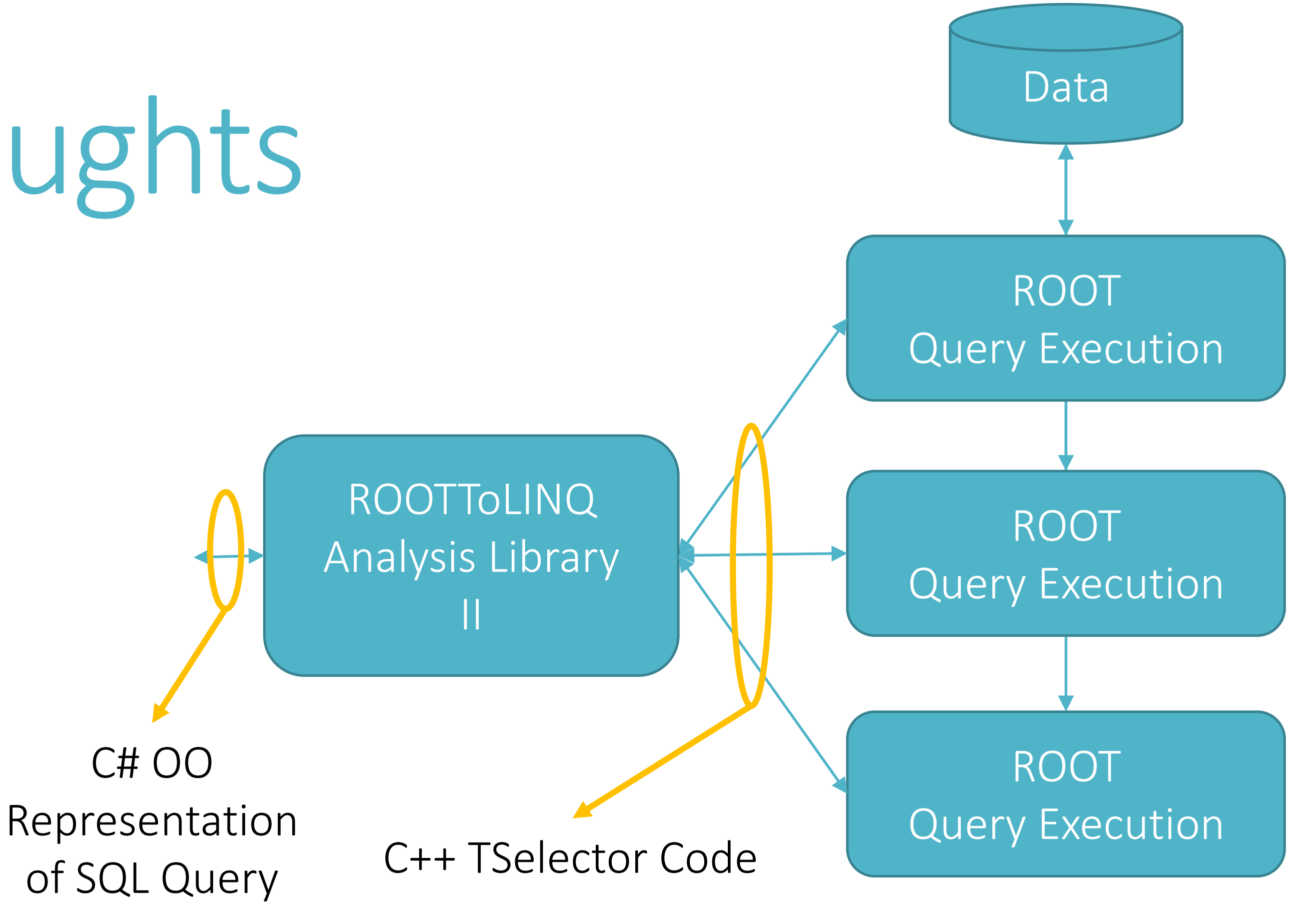
Version 2:

Backend Design Thoughts

The Backend:

- Code for a specific backend (HPC, GPU, etc.)
- I have 4 implementations of the back end co-existing
- Run locally on your laptop (in process?)
- Optimizations for specific types of problems
- Update independently of the analysis code
- Allows same analysis code to be run: no changes by analyzer!

The Role of the Host language and the Analysis Language



Analysis Language

Analysis Language Wish List

Expressiveness:

- Implied loops over objects (jets, electrons)
- Remapping of data to build e.g. a muon object from a flat ntuple
- Build new structures on the fly
- Complex functions and math calculations
- Type system expressive enough to handle experiment's AOD format
-

Analysis UI Capabilities

Operational:

- Capable of being transmitted over the wire
- Can be easily combined with other queries for efficiency
    - Analysis server needs to support this
- Expressive and easy to parse by code

Analysis Language Wish List

Expressiveness:

- Implied loops over objects (jets, electrons)
- Remapping of data to build e.g. a muon object from a flat ntuple
- Build new structures on the fly
- Complex functions and math calculations
- Type system expressive enough to handle experiment's AOD format
- Deal with leaky abstractions

Analysis UI Capabilities

Operational:

- Capable of being transmitted over the wire
- Can be easily combined with other queries for efficiency
    - Analysis server needs to support this
- Expressive and easy to parse by code

I bet we know enough about where and why this happens

Host Language Wish List

<u>Expressiveness:</u>

- **Analysis language embeddable**
  - As a string (can do)
  - As actual code (better)
- Queue up and track multiple queries
- Manipulate returned objects
- Easy to read list of steps that are being taken to do the analysis

➡ Analysis UI ⬅

<u>Operational:</u>

- Backend library to translate queries into the Analysis language
- Library to handle return objects (histograms, numbers, etc.)

Possible examples:

- C#
- C++
- Jupyter Notebook/python

Analysis Server

Goal:

⇨ One analysis language
⇨ Many host languages
⇨ Many analysis servers

Operational:

- Translate queries into efficient execution code
- Access to datasets
  - Along with common way to refer to them
- Scheduler to queue up queries
- Query can be treated as a cache key to speed up common requests
- Result can be tagged with query to preserve operations that generated result
- Types of result
  - May have to be transmitted across the wire
  - Numbers, arrays, histograms, TTree's and csv files (for ML).

Scalability & Status

Code can be found on github: https://github.com/gordonwatts/LINQtoROOT

Time to move on from this experiment

Successes

- LINQ syntax is very well matched for HEP data analysis
- Decent way to express common histogram binning, etc.
- Code is composable: large complex queries from small ones
- Easy mapping features to map flat ntuple into structured ntuple
- Can deal with any type that ROOT has a dictionary for…
- PROOF like backend works

C# worked well too:

- Strongly typed after you declare ntuple
- Query syntax is part of the native language
- Caching works, but it is slow to generate a key

More Work Required

- C# has trouble managing multiple queries is hard (monad hell) – makes code ugly
- Running on multiple machines is fragile – why am I reinventing SPARK/PROOF/DASK!?
- Referring to datasets with a single name space across multiple machines hard to solve!
- Caching is only per-local machine
- Code optimization is… meh. At best.
- Backend has TSelector baked in for no good reason.

Conclusions

C# and LINQ based system works well

- Produced one paper and one conference note, and soon a second paper

SQL-like analysis language is well suited to HEP data analysis

Caching idea works well

- Makes it very quick to add a new plot without re-running old plots

Analysis Backend is a problem in need of further R&D

- How are datasets specified to mean the same thing no matter where you are?
- Multiple-machine running needs a real backend

What is next?

- Take a step back and design a real system with this knowledge
- Additional features – ML Training
- Design it for more than just me to use