

Introduction to Analysis with DSTs

CMS Software Tutorial

Norbert Neumeister

Workshop on Muons at Hadron Colliders

Fermilab, April 15, 2004

Outline

- Introduction
- What are analysis objects
- Why persistent analysis objects
- DSTs and Streams
- COBRA framework interfaces
- RecCollections

Disclaimer

- This is not a ROOT tutorial
- I assume that you know a bit of C++
- I will not talk about Grid tools to perform distributed analysis
- I will not talk about how to handle METADATA
- ORCA is in a rapid development phase (try to use the latest greatest)
 - Today: ORCA_8_0_1
- We need feedback from the users

Introduction

- **Analysis Objects (reconstructed objects):**
 - clusters, RecHits, track segments, etc.
 - tracks, vertices, muons, electrons, jets, etc.
- **Persistent analysis objects**
 - store the result of event reconstruction → **DST/ESD**
 - first prototype for DC04
- **Persistency**
 - Use COBRA/POOL
 - Use RootTrees (rootcint)
 - User defined persistency (e.g. ntuple like RootTree)

Reconstruction

Reconstruction is structured in several hierarchical steps:

Detector-specific processing: detector data unpacking and decoding, apply detector calibration constants, reconstruct *clusters* or *hit* objects.

Tracking: hits in the silicon and muon detectors are used to reconstruct global tracks. This is one of the most CPU-intensive activity.

Vertexing: primary, secondary vertex candidates.

Particle identification: produces the objects most associated with physics analyses. Using a wide variety of sophisticated algorithms, standard ***physics object*** candidates are created (*electrons, photons, muons, missing E_T and jets; heavy-quarks, tau decay*)

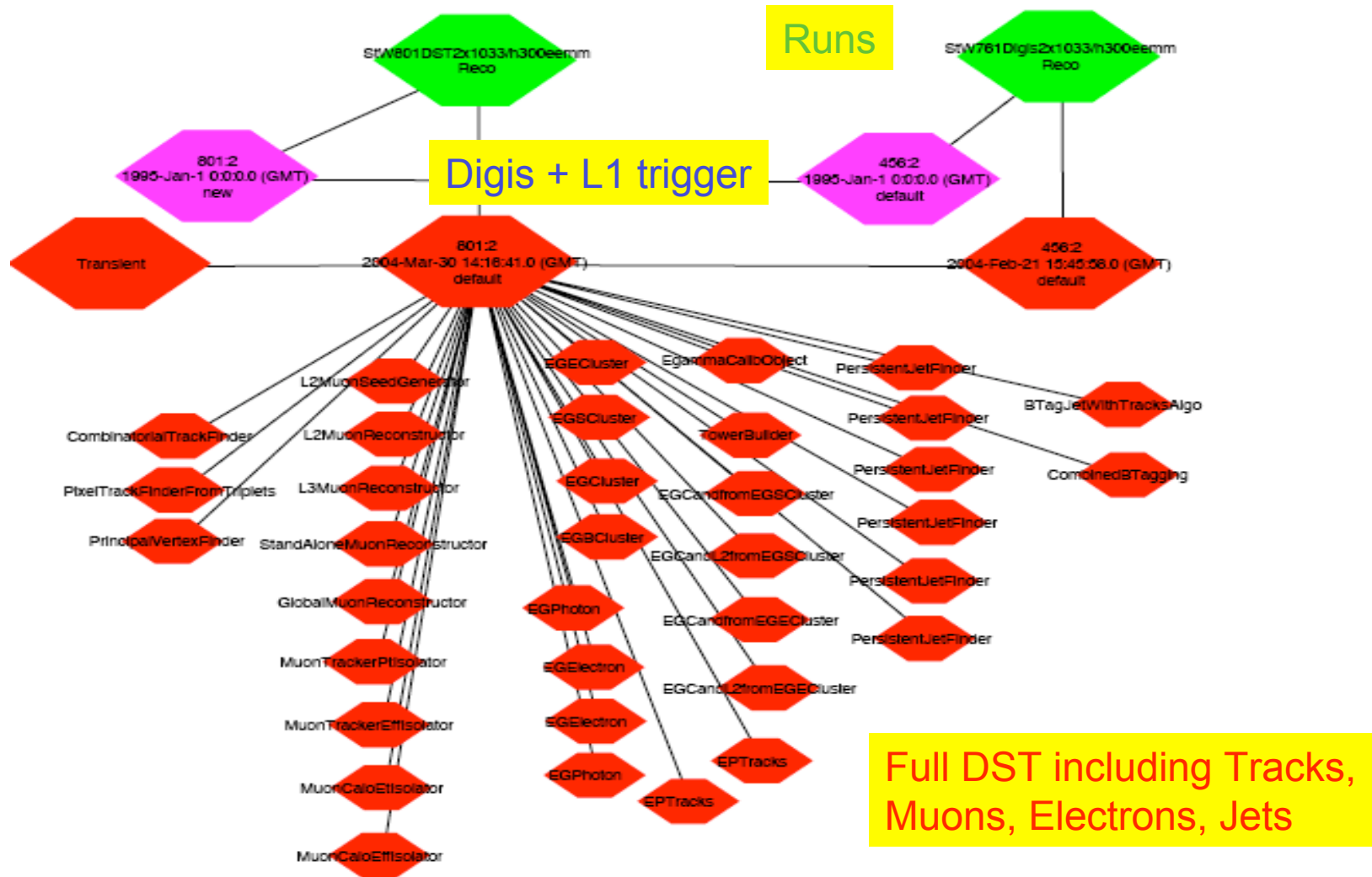
Analysis

- Analysis is **not** to use *the tool* to plot an histogram, but the full chain from accessing event data up to producing the final plot for publication
- Analysis is an iterative process:
 - Reduce data samples to more interesting subsets (selection)
 - Compute higher level information
 - Calculate statistical entities
- Several steps:
 - Run analysis job on full dataset (few times)
 - Use interactive analysis tool to run several times on reduced dataset and make plots
- Only one part of the work can be done (*today*) with an interactive analysis tool

What is a DST

- We stored already
 - MCInfo: MC generator, SimTrack, SimVertex
 - SimHits: subdetector specific
 - Digis & Associations
- Add Level-1 Trigger info and RecHits (Calorimeter)
- DST: Store a complete record of all objects created during reconstruction (organized in collections)
- DST contains collections of reconstructed objects
 - Tracker tracks, Muons, Electrons, Jets, etc.
 - In total about 50 different RecCollections
 - Selection of what to store via orcarc

DST



DST Contents

- DSTs contain reconstructed objects for “users”

HLT info

Level-1 Trigger

CombinatorialTrackFinder

CombinedBTagging

PixelTrackFinderFromTriplets

PrincipalVertexFinder

EGCandL2

EGBCluster/SECluster/Cluster

EGHLTelectron/photon

EGTracks

EGcalibration

EGofflineCandidates

EGofflineElectron/Photon

MET-L1 Trigger

METfromCaloRecHits/EPHTowers

METfromIterConeJets/KtJets

RecJet-Itercone0.5/0.7

RecJet-Ktrecom1/4

TowerBuilder

StandAloneMuonReconstructor

GlobalMuonReconstructor

L2MuonReconstructor

L3MuonReconstructor

MuonCaloEffIsolator

MuonCaloEtIsolator

MuonTkEffIsolator

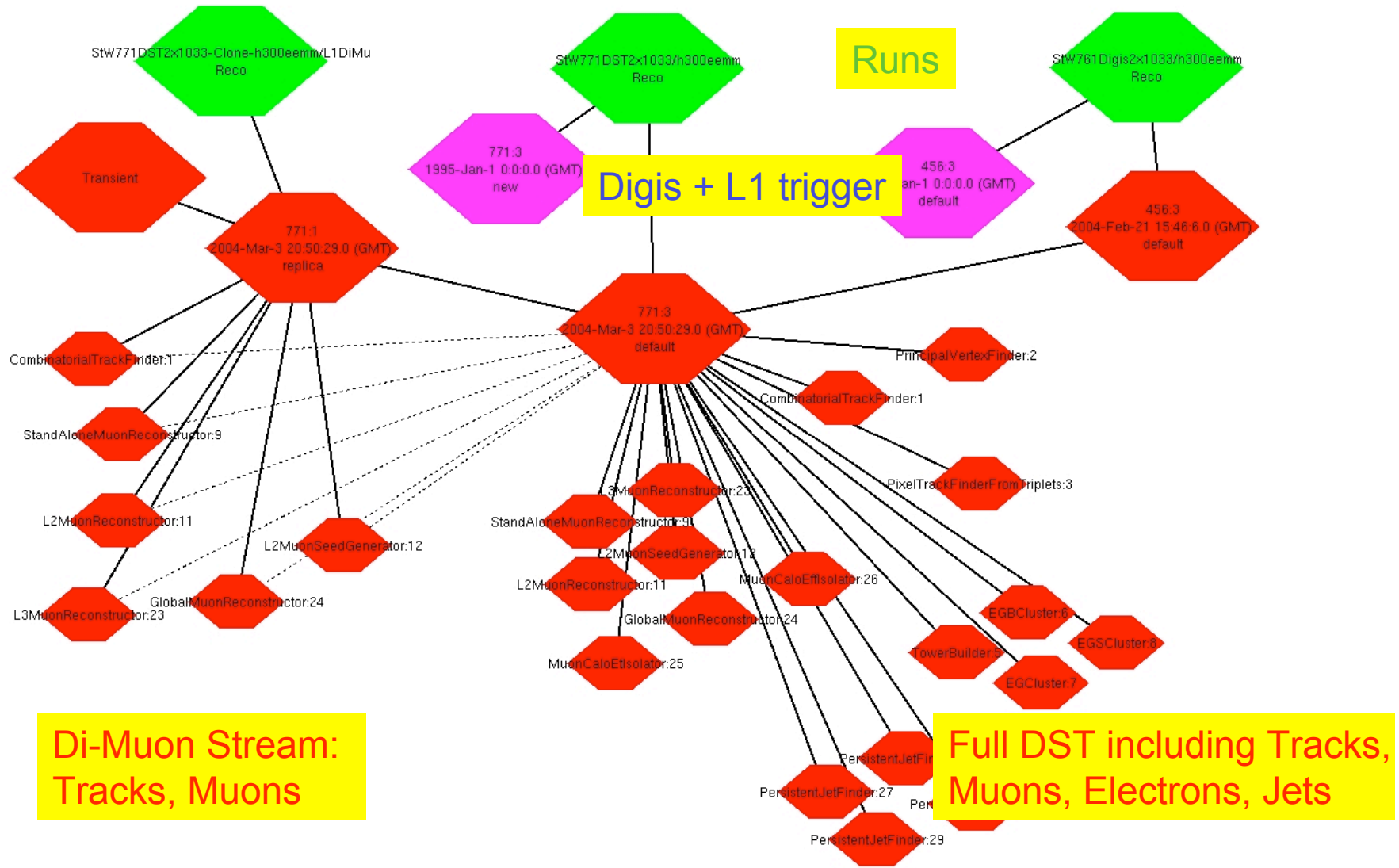
MuonTkPtIsolator

Streams

- Clone the collections of objects you want to have in a separate dataset
- Write PARTS of SELECTED events into different datasets
- Event selection by filter classes
 - inherit from RecCloner and LazyObserver
- Match RecCollections (and Digis) to Streams via orcarc
- All streams share the same owner
 - InputCollection = /System/DigiOwner/Dataset
 - OutputDataSet = /System/DSTOwner/Dataset
- Streams created as
/System/DSTOwner-C-Dataset/Stream

- Links back to Digis, etc.
- Streams get complex
- Writing streams creates many files

DST and Streams



RecObject Persistency

Requirements:

- Ability to make the data part of the object persistent
- Ability to store relations between RecObjects
- Ability to query for the stored objects, and trigger reconstruction if the stored objects do not correspond to the query

Analysis Objects

- **Reconstructed objects are hierarchical:**
 - High-level objects are constructed from lower level objects
 - Examples: Jets are constructed from Tracks, Vertices are constructed from Tracks, etc.
 - At the lowest level all objects are constructed from raw data (Digis)
- **It is necessary to get the *configuration* used to create the objects together with the *object collection*!**

Requirements (I)

- A collection of reconstructed objects is defined by:
 - **Type of reconstructed object**
 - **Name of algorithm**
 - identifies the algorithm builder
 - **Version of algorithm**
 - the developer must set the version of an algorithm
 - avoid new version when only documentation is changed
 - **ParameterSet** (parameters of all algorithms but not of all components)
 - **ComponentSet** (dependency on other RecUnits)
 - **CalibrationSet**
- Save full dependency and check state of underlying objects
 - i.e. collection of jets depends on tracks and CaloClusters, tracks depend on tracker digis, etc.
- Guarantee same results for same configuration

Requirements (II)

- The user can access a collection of reconstructed objects by specifying the RecObject type and a RecQuery (*see later*)
- Provide one documented user interface to retrieve reconstructed objects: RecCollection
 - In addition there is an interface for filtered RecCollections

RecCollections

- RecCollection addresses the off-line analysis use case: iterating over reconstructed objects of some type and using them for analysis or for further reconstruction of higher-level objects.
- **The collection is homogeneous: All objects in the collection are reconstructed in the same way**
 - Within the event
 - In all events
 - Reproducibly
- **Reconstruction is “expensive” and should not be wasted:**
 - Reconstruction is done “on demand”
 - Reconstruction results are cached for further access
 - Reconstruction results are persistent (if desired)

Algorithm Configuration

- The configuration is decomposed into
 - Algorithm name
 - Version identifier
 - Fixed **parameters** directly used by the algorithm
 - Do not depend on the input data
 - If they change the reconstruction result changes
 - Modeled by **ParameterSet**
 - **Calibration** parameters directly used by the algorithm
 - Depend on the input data
 - If they change the reconstruction result changes
 - Modeled by **CalibrationSet**
 - Configuration of the input RecCollections (**components**)
 - Modeled by **ComponentSet**
- All these compose the **RecConfig**

RecQuery

- **User-friendly way to define RecConfigs**
 - Only the algorithm name is mandatory
 - All other fields are taken from the default Config
 - Only the parameters, components, or calibrations which need to be changed (or fixed) need to be set
 - You don't have to know the names of all parameters, only the ones you want to modify
 - You can put anything in a RecQuery, e.g. Parameter names not known to the RecAlgorithm
 - The check will be done when you *use* the RecQuery, and an exception will be thrown if it's not OK
 - Some parameter type conversions are accepted and done automatically
 - int to double

RecConfig Settings

Order of initialization:

1) defaultConfig of the algorithm

- That's the only place where parameter names and types are defined

2) .orcarc: Every parameter is configurable with the following syntax

- AlgorithmName:ParameterName = value
- overwrites 1)

3) The current RecQuery overwrites 2)

For components same sequence as for parameters in a recursive way:

AlgorithmName:ComponentName:ParameterName = value

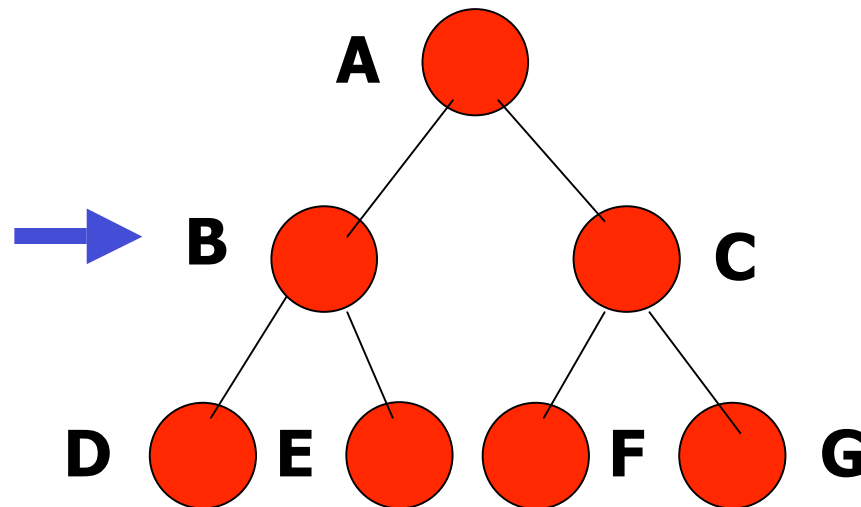
ParameterSet

- Parameters are organized as pair of string and value (name, value)
 - The name is in the scope of the ParameterSet and can be reused in other ParameterSets
 - The only way to instantiate a non-empty ParameterSet is by using a ParameterSetBuilder (prevent from changing an existing ParameterSet)
 - The following types are supported: int, bool, double, string
 - For double the equal operator allows tolerances
 - If nothing set explicitly all parameters are default values
- Usage:

```
ParameterSetBuilder builder;  
// add integer  
builder.addParameter( "numberOfTracks" ,10);  
// add double with tolerance  
builder.addParameter( "deltaRCut" ,0.6,0.002)  
ParameterSet ps = builder.result();  
int nTracks = ps.value<int>( "numberOfTracks" );
```

ComponentSet

- If a RecUnit uses the result of other RecUnits, like a Vertex algorithm uses reconstructed tracks, it should fully specify the configuration of the lower level RecUnits via RecConfigs.
- A ComponentSet is a set of pairs (name, RecQuery)
 - The name is in the scope of the concrete component
- The ComponentSet is a set of all RecQuery objects
 - use a recursive query!



RecUnits: A, B, C, D, E, F, G

B, C, D, E, F, G are components of A
In addition RecUnit A can use a list of algorithms (TrajectoryBuilder, TrajectoryCleaner, TrajectorySmoother).

RecCollection

Retrieving (persistent) objects:

MyRecObj inherits from RecObj

```
RecCollection<MyRecObj> myobjects(RecQuery("Name_of_algorithm"));  
cout << myobjects.query() << endl;  
RecCollection<MyRecObj>::const_iterator it;
```

Usage

```
RecQuery q("JetFinder");
q.setParameter("ConeSize", 0.77);
RecQuery myTF("TrackFinder");
q.setComponent("TrackReconstructor", myTF);
...
RecCollection<Jet> myjets(q);
cout << myjets.size() << endl;
cout << myjets.query() << endl;
```

Persistent Muon

- **Persistent Track data:**
 - First and last Trajectory State
 - Track quality criteria
 - χ^2
 - Number of degrees of freedom
 - Number of hits used in the fit
 - Number of “lost” hits
 - Individual RecHits used in muon fit (possible to refit track!)
- **Persistent RecMuon data:**
 - innermost and outermost Trajectory State
 - Vertex extrapolated State (stateAtIP, stateAtVertex)
 - Track quality
 - TrajectoryState persistency as for Tracker
 - TRecRef<TTrack> muonTrack
 - TRecRef<TTrack> trackerTrack

Muon Reconstruction

- For the time being we have four different muon reconstruction algorithms implemented:
 - StandAloneMuonReconstructor
 - GlobalMuonReconstructor
 - L2MuonReconstructor
 - L3MuonReconstructor
- All four algorithms are `RecAlgorithm<RecMuon>`
 - inheriting from `RecAlgorithm<T>`
- `RecMuon` inherits from `Ttrack (RecObj)`
- Easy access to reconstructed muons

```
RecQuery q("<Name of Reconstructor>");
RecCollection<RecMuon> theCollection(q);
for (RecCollection<RecMuon>::const_iterator
     it = theCollection.begin(); it != theCollection.end(); ++it)
cout << "RecMuon: " << (**it) << endl;
```

Muon Isolation

- Output again a RecMuon, but with isolation info filled
- `float RecMuon::isolation()`
- **Several isolation algorithms are available**
 - **L2MuonCaloIsolator** for HLT: uses calo
 - **L3MuonTrackerIsolator** for HLT: uses Tracker
 - **MuonCaloEtlisolator** for offline: return ΣE_T
 - **MuonCaloEffisolator** return discriminating parameter $[0,1]$
 - **MuonTrackerPtisolator** return the Σp_T
 - **MuonTrackerEffisolator** as above

Summary

- DSTs contain reconstructed objects for “users”
- **Status**
 - Prototype has been implemented for DC04 (PRS + CCS)
 - Used in DC04: get experience in producing DSTs
- **Interface:**
 - One clean COBRA interface (RecCollection)
- **Muon software:**
 - All algorithm converted to RecAlgorithms
 - New RecMuon class
 - Waiting for feedback

Exercise

Tasks

- 500 H(300 GeV) \rightarrow $e\bar{e}\mu\mu$ events
- Read a DST
- Extract MC, Level-1 Trigger and muon information
- Fill histograms
- Fill ROOT tree
- Plot fill p_T spectrum of simulated and reconstructed muons
- Plot di-muon invariant mass
- Have a look at the interface of RecMuon:
 - ORCA reference manual
 - <http://cmsdoc.cern.ch/swdev/snapshot/ORCA/ReferenceManual/html/classes.html>
- Get global and standalone RecMuon information
- **Include electrons**

First Step

```
wget http://home.cern.ch/neumeist/ORCATutorial/go
```

```
chmod 555 go
```

```
./go
```

```
cd ORCA_8_0_1/src/Workspace
```

Try to understand the code!

Modify MuonAnalysis.cc

```
scram b
```

```
eval `scram runtime -csh`
```

```
Tutorial -c orcarc
```

recompile

set runtime environment

run executable

Solution

```
wget http://home.cern.ch/neumeist/ORCATutorial/MuonAnalysis.cc
```