Trying out SANs
Structured Athena-aware Ntuples

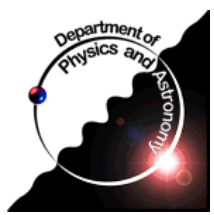
Personal notes and impressions
Comments, advice more than welcome
Updated version of talk

Disclaimer:
I have not tried everything!
I do not understand everything!

LAPP, 20 April 2007
Michel Lefebvre

Physics and Astronomy
University of Victoria
British Columbia, Canada

Laboratoire d'Annecy-le-vieux de physique des particules, France



Introduction

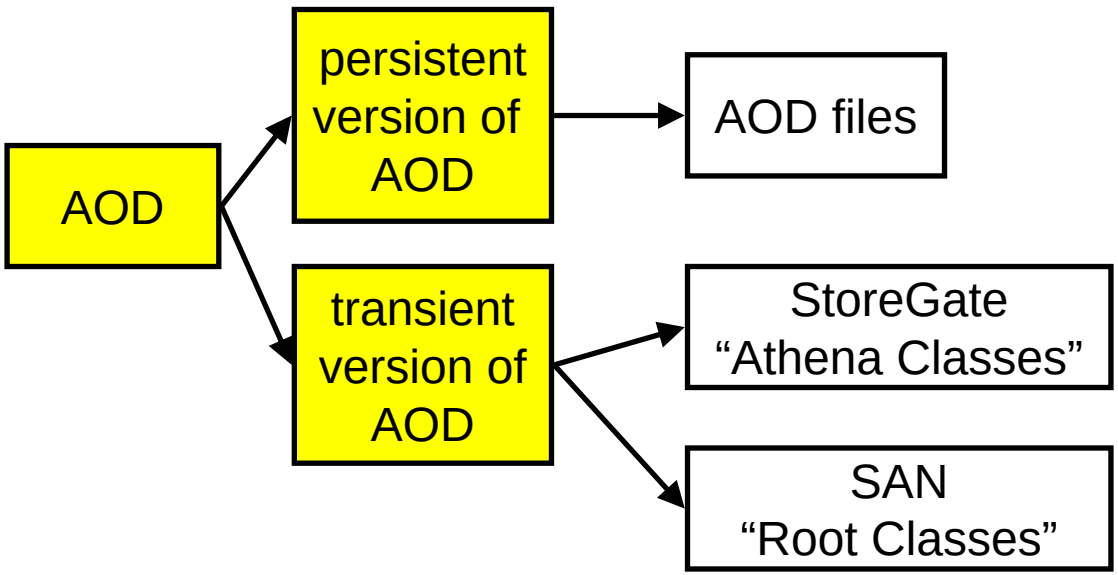
- Analyzing reconstructed events using Athena
  - perform analysis using Athena (from ESD or AOD)
    - can use tags or Athena-aware Ntuples (AAN) to select events
    - can produce analysis dependent Ntuples (EventView)
      - continue analysis using Root macros
- Analyzing reconstructed events using Root
  - using Athena, produce Ntuples containing the reconstruction information
    - combined Ntuples (CBNT)
    - Athena-aware Ntuples (AAN)
    - Structured Athena-aware Ntuples (SAN)
  - perform analysis using Root macros
  - AAN (including SAN) allow back navigation (to AOD, ESD,...) when used as tag

Introduction

- Ntuple formats
  - CBNT, old AAN, and EventView ntuples (AAN)
    - “flat” format, that is containing int, float, etc.
      - easy to deal with in Root
      - but Root macro code very different from Athena analysis code
  - SAN
    - “structured” format, that is containing classes
      - needs “dictionary” to teach Root about the classes
      - but Root macro code very similar to (ideally the same as) Athena code
- AOD
  - an AOD file contains a persistent version of the AOD events: pAOD
    - optimized for data storage and data access speed
    - has no methods needed by the user for event analysis
    - this is what you see if you try to access an AOD file directly with Root
  - in Athena, a transient version of the AOD event is available
    - this is what you access in your Athena code

SAN concept

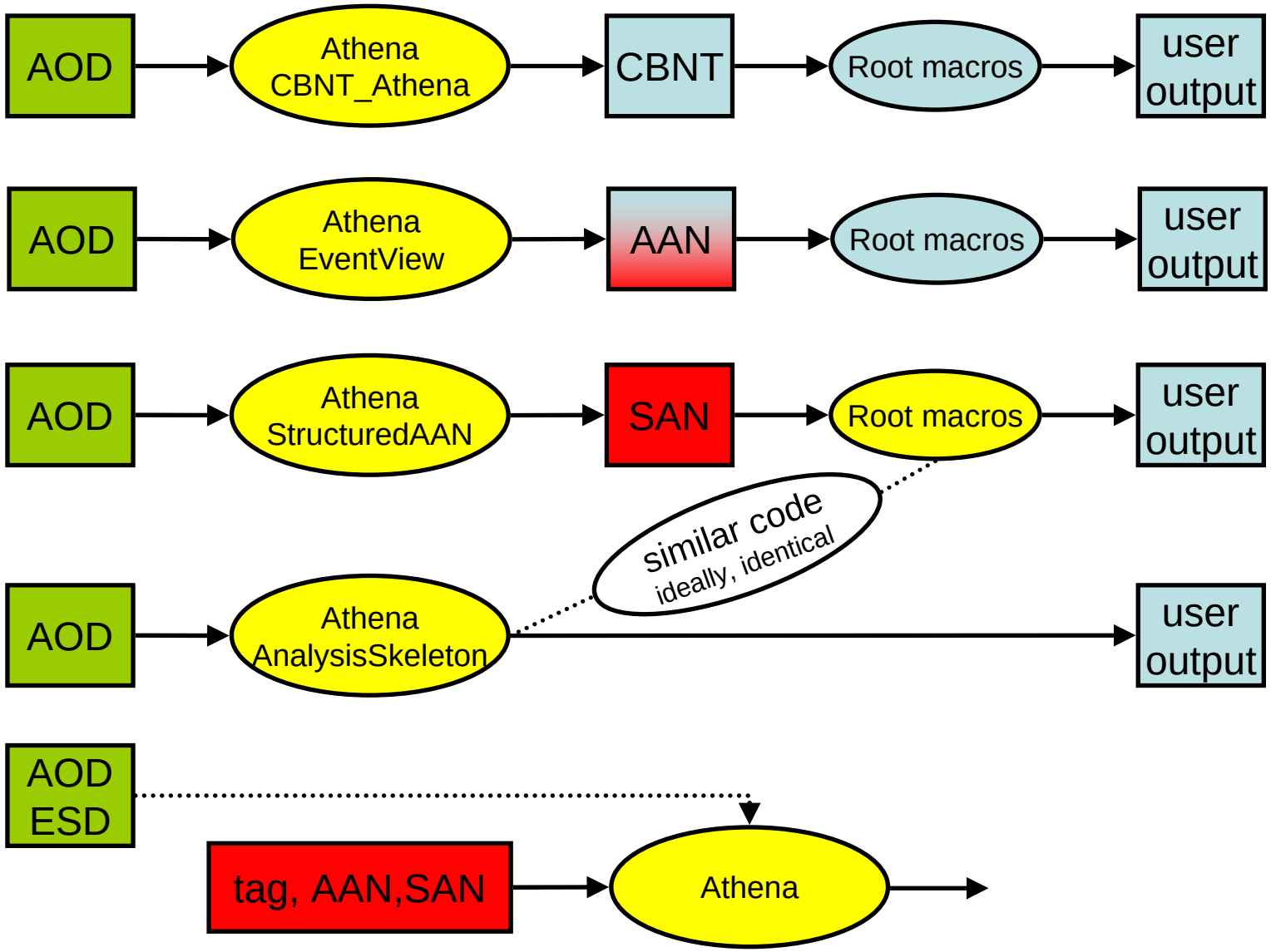
- add the transient AOD objects to a structured Root tree to provide Root access to the AOD
  - transient AOD objects is what you access in an Athena analysis
  - need to provide “Root version” of these objects
    - User:: namespace classes
    - ....extra maintenance!
  - same functionalities and same interface in Root and in Athena



atlas/PhysicsAnalysis/JetTagging /JetTagEvent/JetTagEvent/ParticleJet.h

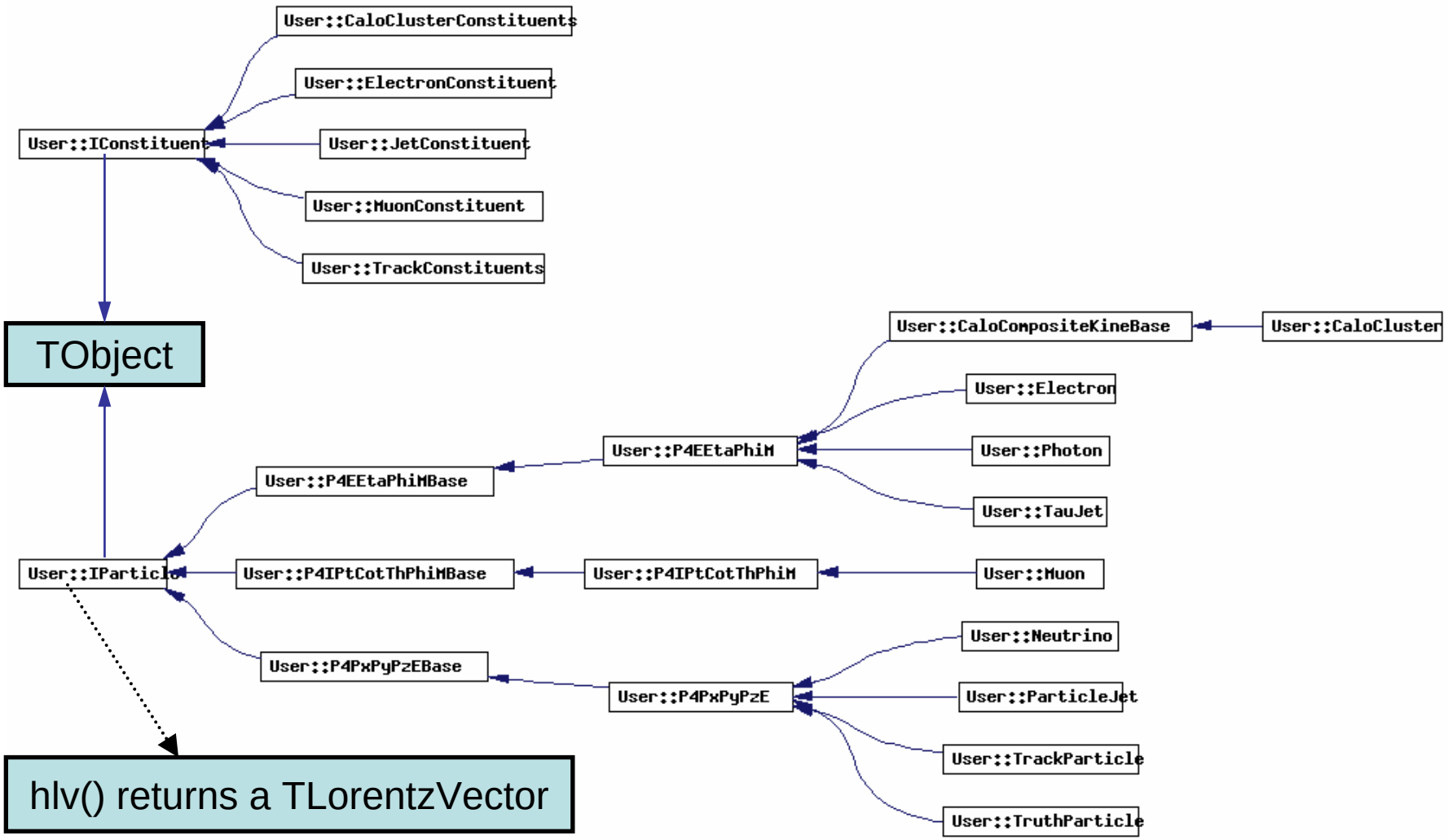
atlas/PhysicsAnalysis/AnalysisCommon/ UserAnalysisEvent/UserAnalysisEvent/ParticleJet.h

Analysis models



UserAnalysisEvent EDM

- The User namespace was implemented for SAN
  - here doxygen from 12.0.5: http://atldbdev01.cern.ch:20080/swbrowser/current.html



Getting started

- Start from the SAN TWiki page
  - https://twiki.cern.ch/twiki/bin/view/Atlas/SAN
- I used Athena 12.0.6 on ccali
  - following tags used (token from klog.krb on cern.ch)
    - PhysicsAnalysis/AnalysisCommon/UserAnalysis-00-09-10-14
    - PhysicsAnalysis/AnalysisCommon/UserAnalysisUtils-00-01-01-07
    - PhysicsAnalysis/AnalysisCommon/ParticleBuilderOptions-00-00-28-07
  - following $W \rightarrow e\nu$ AOD used (thanks to Thibault Guillemin)
    - trig1_misal1_csc11.005104.PythiaWenu.recon.AOD.v12000601_tid006048._*.root.*
    - on dcache (I needed to issue lcg_env to be able to access dcache files)
    - a total of 1911 AOD files @ often 250 events each = 476 656 events
    - each AOD is about 35 MB

AOD content

- You can look at the AOD content, for each event, with the jobOptions

```
StoreGateSvc = Service( "StoreGateSvc" )
StoreGateSvc.Dump = True  #true will dump data store contents
StoreGateSvc.OutputLevel = DEBUG
```

- Example excerpts for ParticleJets:

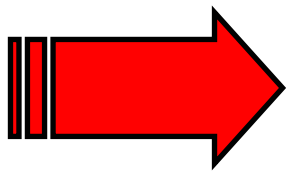
```
Found 13 proxies for ClassID 1118613496 (ParticleJetContainer):
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: AtlfastParticleJetContainer
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Cone4TopoParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data: 0xb3798f28 --- key: Cone4TowerParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Cone4TruthParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: ConeTopoParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data: 0xb394b6a8 --- key: ConeTowerParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: ConeTruthParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Kt4TopoParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Kt4TowerParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Kt4TruthParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Kt6TopoParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Kt6TowerParticleJets
 flags: (  valid, UNLOCKED,   reset) --- data:          0 --- key: Kt6TruthParticleJets
```

jobOptions to produce SAN

- There are variations depending on the package version
  - UserAnalysis/share → UserAnalysis/run
- StructuredAAN_topOptions.py
  - RecExCommon/RecExCommon_topOptions.py
    - ParticleBuilderOptions/share/SAN_Builder_jobOptions.py
- You can modify
  - StructuredAAN_topOptions.py (now in your run directory)
    - AOD files, number of events
  - SAN_Builder_jobOptions.py (do not move it)
    - set which collections will go in the SAN!!!
- Let's look at aspects of SAN_Builder_jobOptions.py

jobOptions to produce SAN

```
theApp.TopAlg += [ "StructuredAAN/SAN" ]
############## The properties of the StructuredNTuple Algorithm
SAN = Algorithm( "SAN" )
############## The SAN making AlgTools ---
SAN.AlgTools = [
   "SanRecVertexBranchTool/RecVertexBranches",
   "SanClusterBranchTool/ClusterBranches",
   "SanTrackBranchTool/TrackBranches",
   "SanElectronBranchTool/ElectronBranches",
   "SanPhotonBranchTool/PhotonBranches",
   "SanMuonBranchTool/MuonBranches",
   "SanMissingETBranchTool/MissingETBranches",
   "SanTauJetBranchTool/TauJetBranches",
   "SanParticleJetBranchTool/ParticleJetBranches",
   "SanTruthParticleBranchTool/TruthParticleBranches“
   ]
```



VERTICES
CLUSTERS
TRACKS
ELECTRONS
PHOTONS
MUONS
MET
TAUS
JETS
TRUTH

- Each type of SAN branches can be configured
- Trigger information is flat: it will be structured in Athena 13

Electron branches

```
# Electrons - List all the containers to go to the SAN
SAN.ElectronBranches.ElectronContainers = [
   "ElectronCollection",
   "AtlfastElectronCollection"
   ]
UserElectronTool = Algorithm ( "SAN.ElectronBranches.UserElectronTool" )
UserElectronTool.TrackRefKey    = UserTrackParticleTool.TrackRefKey
UserElectronTool.ClusterRefKey  = UserCaloClusterTool.ClusterRefKey
UserElectronTool.ElectronRefKey = "ElectronRef"
SAN.ElectronBranches.TTreeBranchBufferSize = 2000
SAN.ElectronBranches.TTreeBranchSplitLevel = 99
```

2 containers

- Two electron branches configured
  - each container is made of User::Electron
    - include info related to tracks and calorimeter clusters

ParticleJet branches

```
# ParticleJets - List all the ParticleJet contianers to appear in the SAN
SAN.ParticleJetBranches.ParticleJetContainers = [
   "Kt4TowerParticleJets",
   "Kt6TowerParticleJets",
   "Cone4TowerParticleJets",
   "ConeTowerParticleJets",
   "Kt4TopoParticleJets",
   "Kt6TopoParticleJets",
   "Cone4TopoParticleJets",
   "ConeTopoParticleJets",
   "Kt4TruthParticleJets",
   "Kt6TruthParticleJets",
   "Cone4TruthParticleJets",
   "ConeTruthParticleJets",
   "AtlfastParticleJetContainer"
   ]
UserParticleJetTool = Algorithm ( "SAN.ParticleJetBranches.UserParticleJetTool" )
UserParticleJetTool.TrackRefKey       = UserTrackParticleTool.TrackRefKey
UserParticleJetTool.ClusterRefKey     = UserCaloClusterTool.ClusterRefKey
UserParticleJetTool.ElectronRefKey    = UserElectronTool.ElectronRefKey
UserParticleJetTool.ParticleJetRefKey = "JetRef"
```

13 containers

each containers is made of User::ParticleJet

Involves tracks, clusters and electrons

Trigger branches

```
if doTrigger:
    include( "TriggerRelease/TriggerFlags.py" )
    #include ("TriggerRelease/jobOfragment_TriggerCBNT.py")
    include("TrigT1Calo/jobOfragment_L1CaloCBNT.py")
    include( "CBNT_AOD/CBNT_AodTrigger_jobOptions.py" )
    include("TrigNtCalo/jobOfragment_TrigNtCalo.py")
    include("TrigNtInDet/jobOfragment_TrigNtInDet.py")
    include("TrigNtInDet/CBNT_TrigEFParticle_jobOptions.py")
    include("TrigNtInDet/InDetTrigPriVxCBNT_jobOptions.py")
    include("TrigNtEgamma/jobOfragment_TrigNtEgamma.py")
    include("TrigNtBphys/jobOfragment_TrigNtBphys.py")
    include("TrigNtBjet/jobOfragment_TrigNtBjet.py")
    include("TrigNtTau/jobOfragment_TrigNtTau.py")
    include("TrigDecisionMaker/jobOfragment_CBNTAA_TriggerDecision.py")
```

- 1 word per branch
  - expected to be structured in Athena 13
  - notice the useful TriggerDecision branches
    - allowing analysis code like

```
if (m_triggerDecisions[L1_EM25]) {
    // do something
}
```

SAN production

- Batch jobs on BQS on ccali
- Each job:
  - reads 10 AOD, a total of 2500 events
  - produces one SAN file about 110 MB in size
  - takes between 8800 and 14500 normalized time units
    - elapsed time of about 45 min
- A total of 192 jobs (default jobOptions)
  - SAN files at

/sps/atlas/m/mlefebvr/SAN_Wenu/SAN/job*/SAN.root

SAN content

- I defined two different choices of set of containers

branch	default	trimmed
RecVertex	1	1
CaloCluster	8	5
TrackParticle	9	1
Electron	2	1
Photon	2	1
Muon	3	2
MissingET	11	11
TauJet	4	2
ParticleJet	13	2
TruthParticle	1	1
Trigger branches	all	only “Decisions”

SAN memory usage

- Looking at the first SAN file (first 10 AOD, 2500 events)

	B		KB/event	**trimmed** KB/event
header	34851	0.03%	0.01	0.01
Muon	220329	0.21%	0.09	0.06
RecVertex	283475	0.27%	0.11	0.11
Photon	371760	0.36%	0.15	0.13
MissingET	489931	0.47%	0.19	0.19
TauJet	617778	0.59%	0.24	0.21
Electron	2011830	1.93%	0.79	0.73
TrackParticle	6165966	5.92%	2.41	1.41
TruthParticle	17407646	16.71%	6.80	6.80
ParticleJet	23956596	23.00%	9.36	1.79
Trigger	25112937	24.11%	9.81	0.00
CaloCluster	27473540	26.38%	10.73	1.51
TOTAL	104146639		40.68	12.96

- comparing original and trimmed SAN
  - some small changes in size for untouched branches: not understood
  - give identical results for me so far

Analysis with SAN

- Skeleton root macros to access the SAN are provided
  - the TWiki was helpful
  - proposed steps:

```
root
root[0] .x startup.C
root[1] TFile *file = TFile::Open("SAN.root")
root[2] TTree * tree = (TTree*)gDirectory->Get("CollectionTree")
root[3] tree->Process("AnalysisSkeleton.C+")
```

  - I found it easy to follow and to modify
    - AnalysisSkeleton.C contains detailed examples on how to access data
    - the code will be familiar to those analyzing AODs using Athena
    - other *.C files give good examples for specific analyses
      - focussing on trigger, or electron, or...
  - The development turnaround time is small
    - modify, compile, and run on a few thousand events in less than 2 min

Root analysis code

- I assume one should try to produce code as close as Athena code
  - to be able to port it to an Athena algorithm if needed
  - this requires some discipline
  - I noticed some intrinsic differences
    - there are surely more... but by design they are kept to a minimum
    - User::IParticle::hlv() returns a TLorentzVector, not HepLorentzVector
      - issues with members, such as DeltaR() and deltaR()
  - the ultimate test would be to actually try to port some analysis code from a Root macro (or set of user Root classes) into an Athena algorithm
    - I have not tried that yet

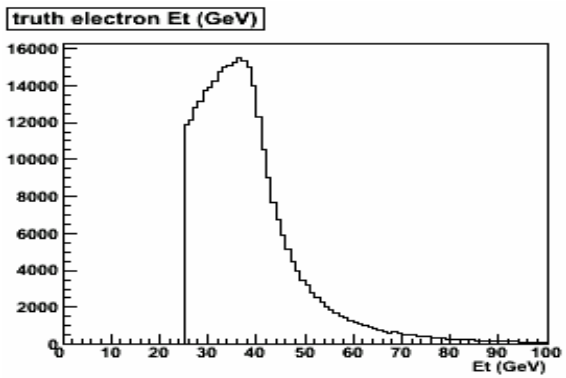
Small analysis with SAN

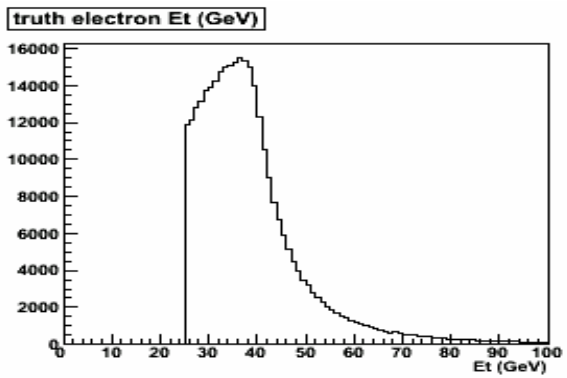
- Analysis using truth electron info
  - select truth electron
    - abs(pdgId) == Pdg::e_minus
    - isGenStable()
      - status%1000 == 1 || (status%1000 == 2 && status > 1000)
      - barcode < 100000
  - define “good” truth electron
    - Et > 25 GeV
    - |η| < 2.5
  - keep events with only one good truth electron

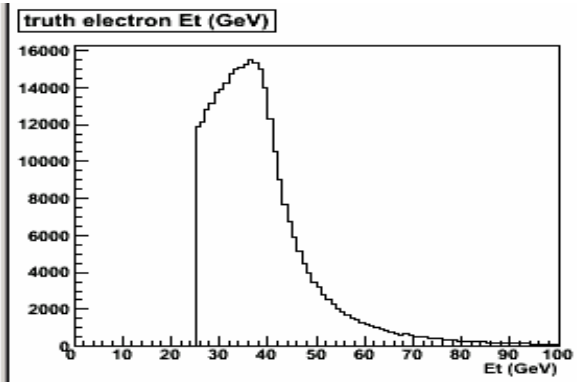
```
-
events read.....................................476656  100%
->with at least one good truth electron...321015  67.3%
->with one good truth electron..............320966  67.3%   100%
-->passing trigger L1_EM25..................314140           97.9%
-->passing trigger L2_e25i..................265790           82.8%
-->passing trigger EF_e25i..................237388           74.0%
-->matched 1 to 1 with an electron.........287154           89.5%
```

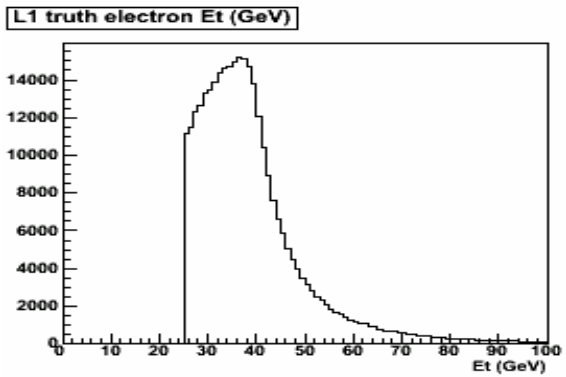
49 events have more than one good truth electrons...

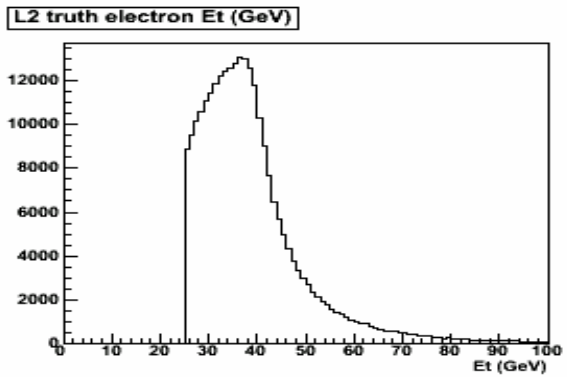
Trigger efficiency study (truth only)

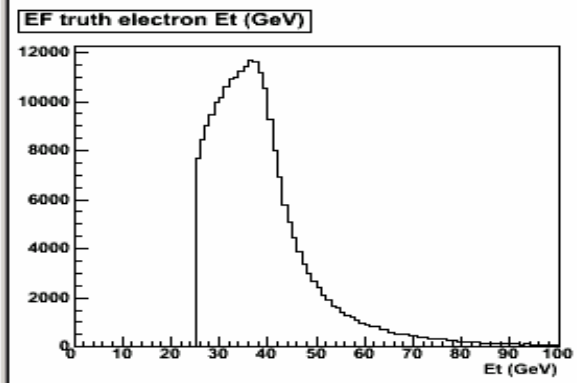


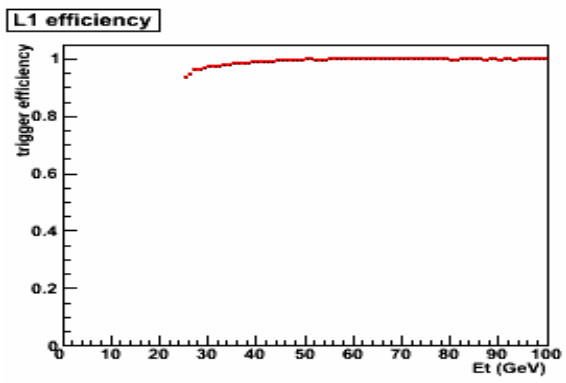


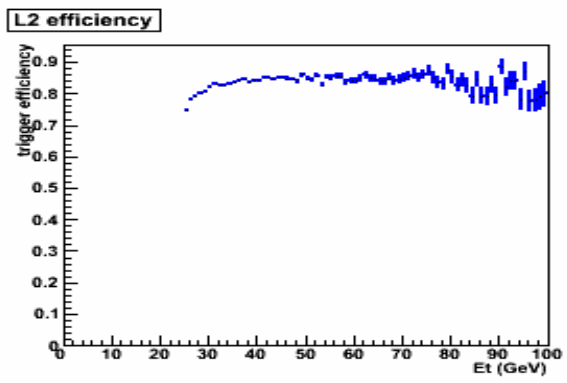


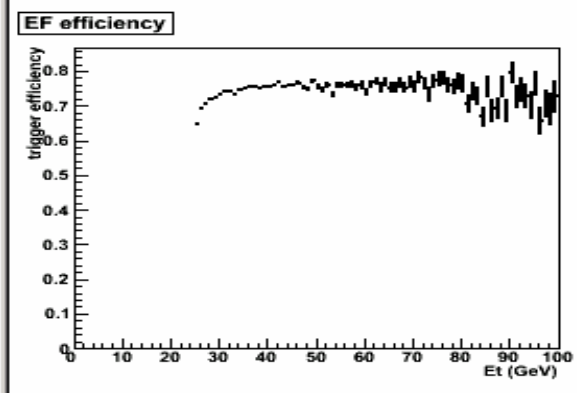




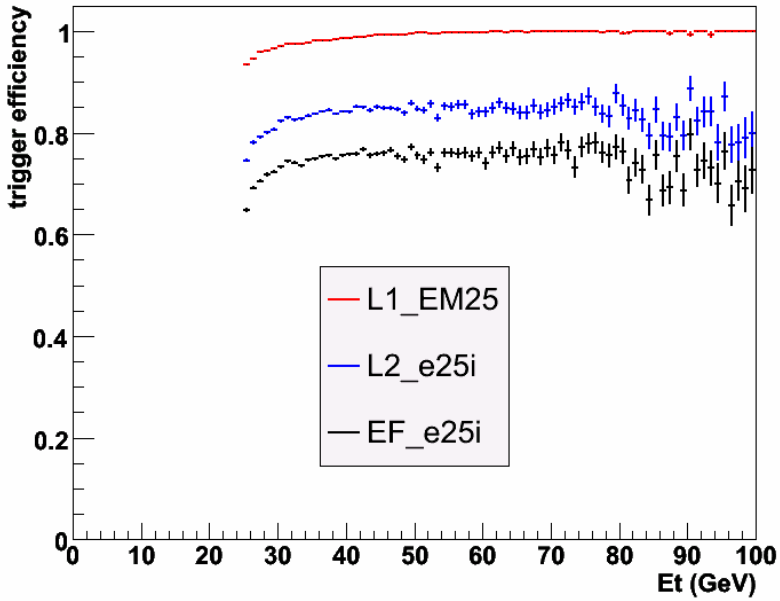


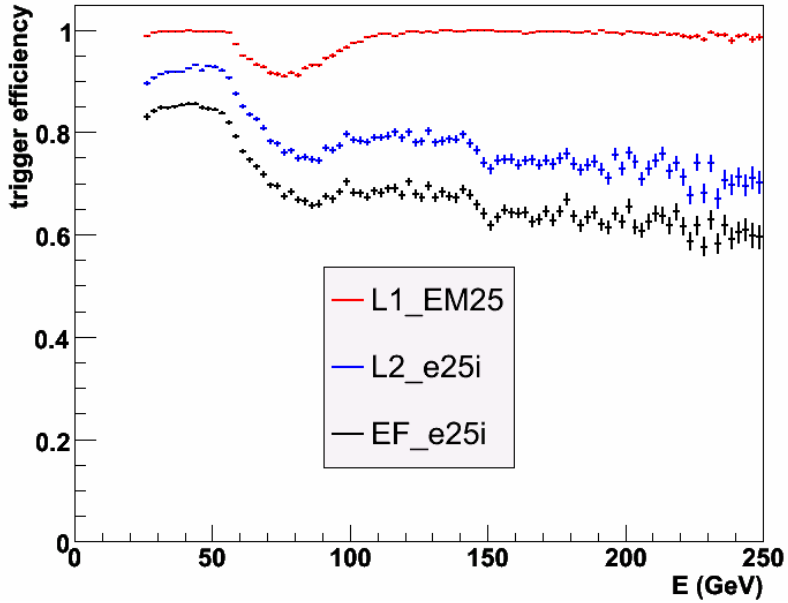


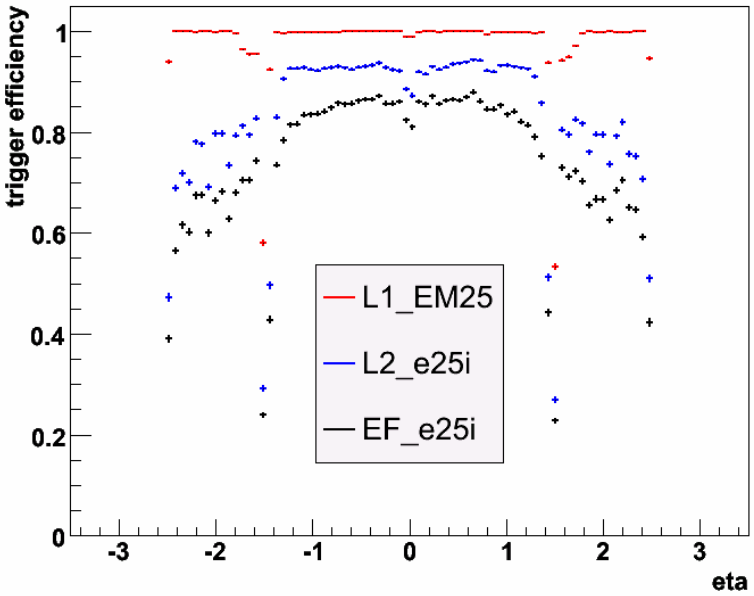


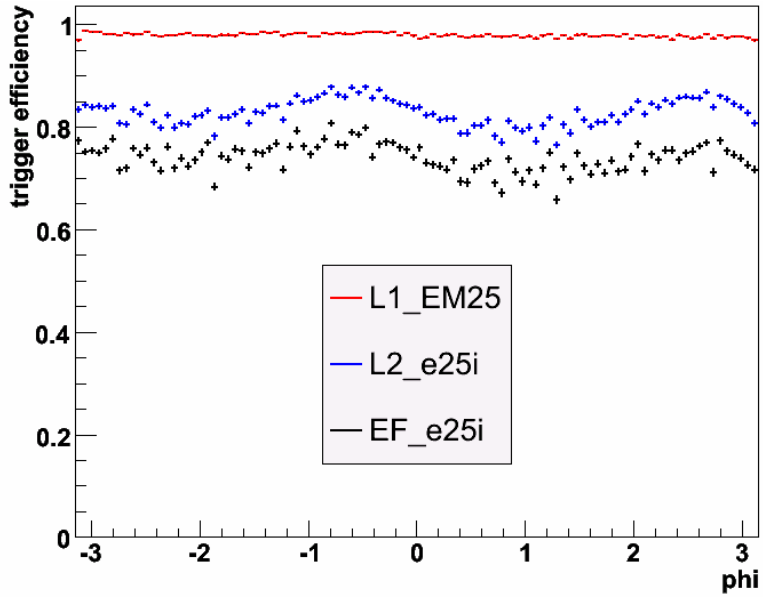


Trigger efficiency study (truth only)

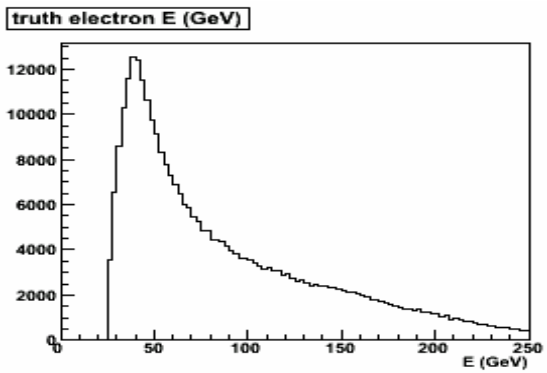


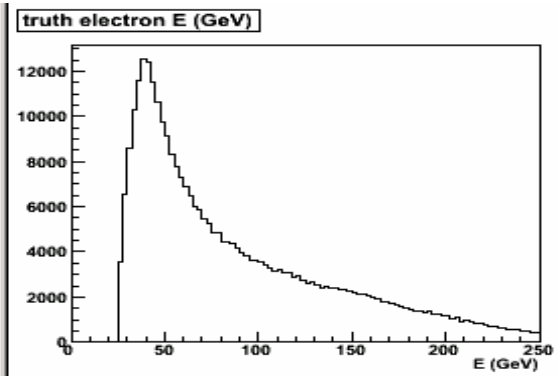


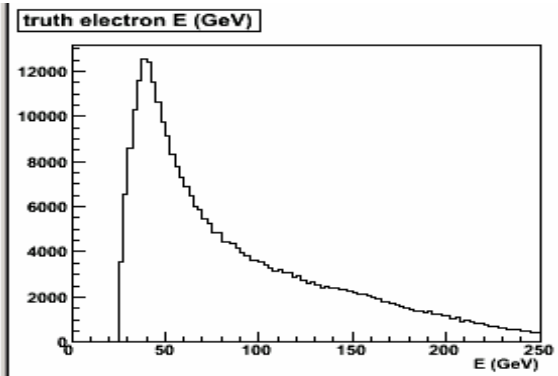


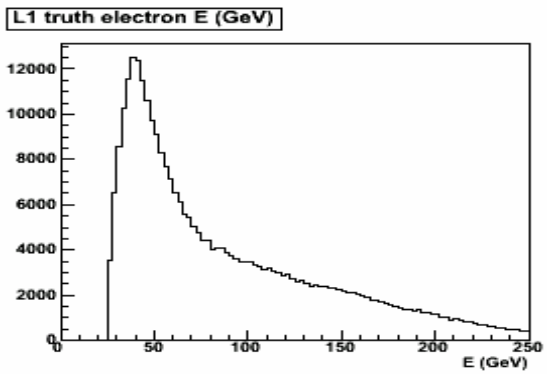


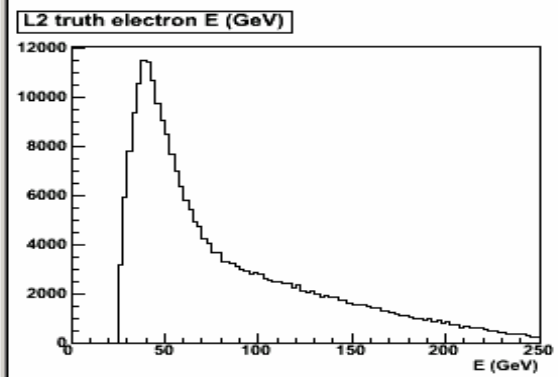
Trigger efficiency study (truth only)

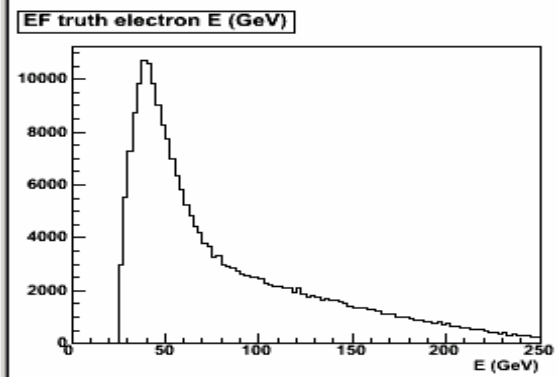


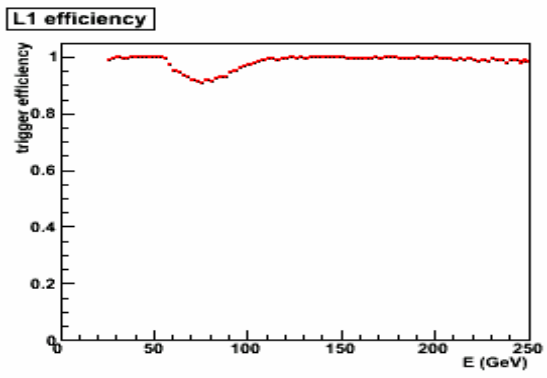


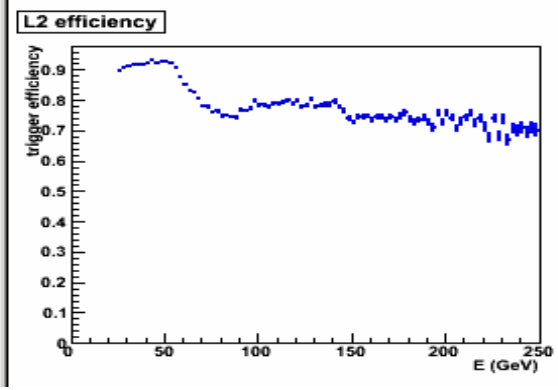


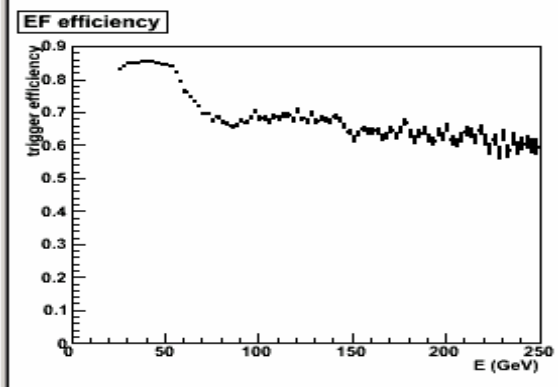






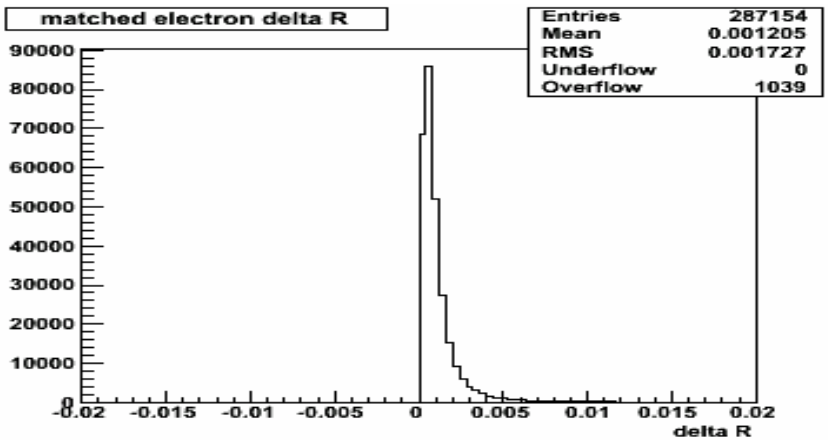


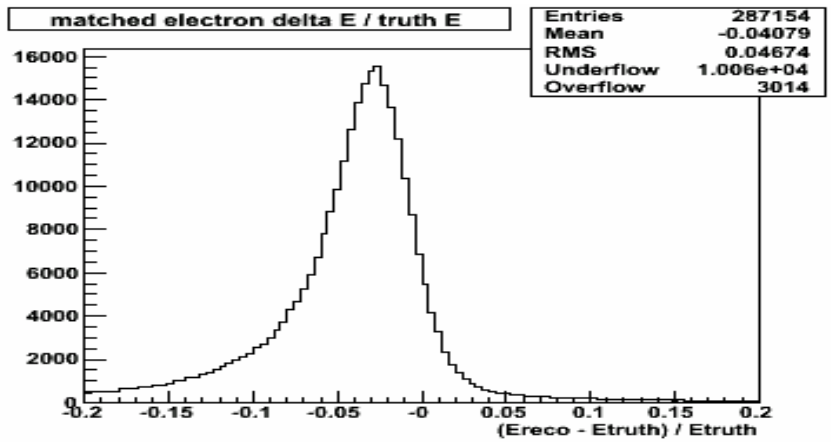


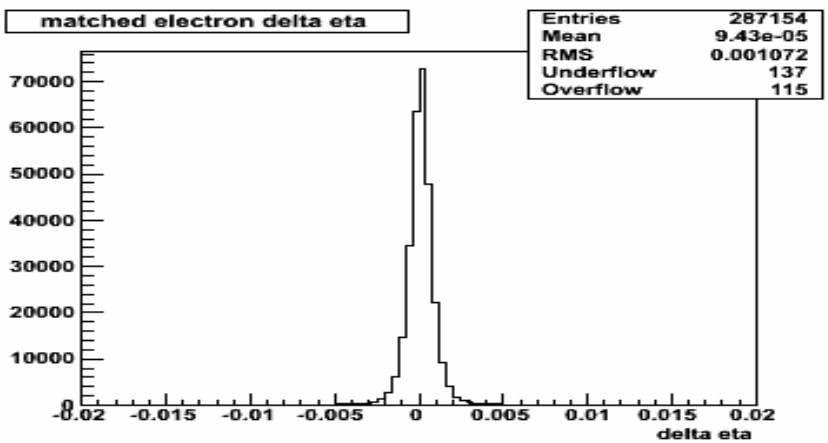


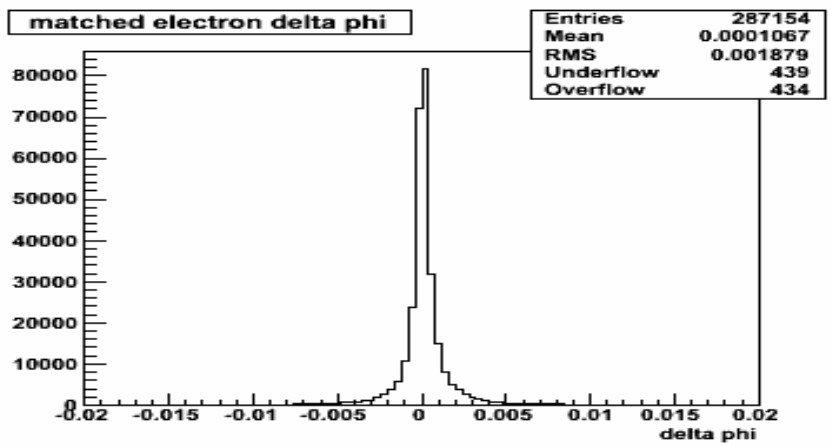
Truth and Reco electron matching

- Look for 1 to 1 ΔR match between the truth electron and one reco electron (no selection on reco electrons)
  - limit match search to ΔR < 0.2



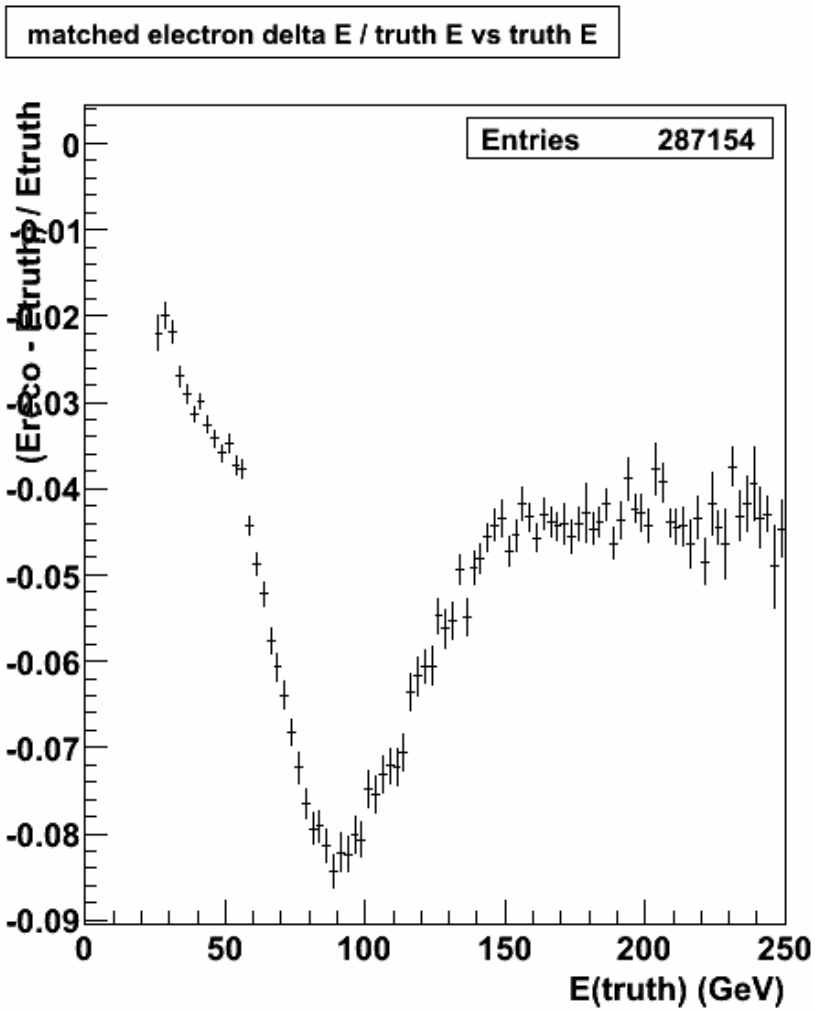


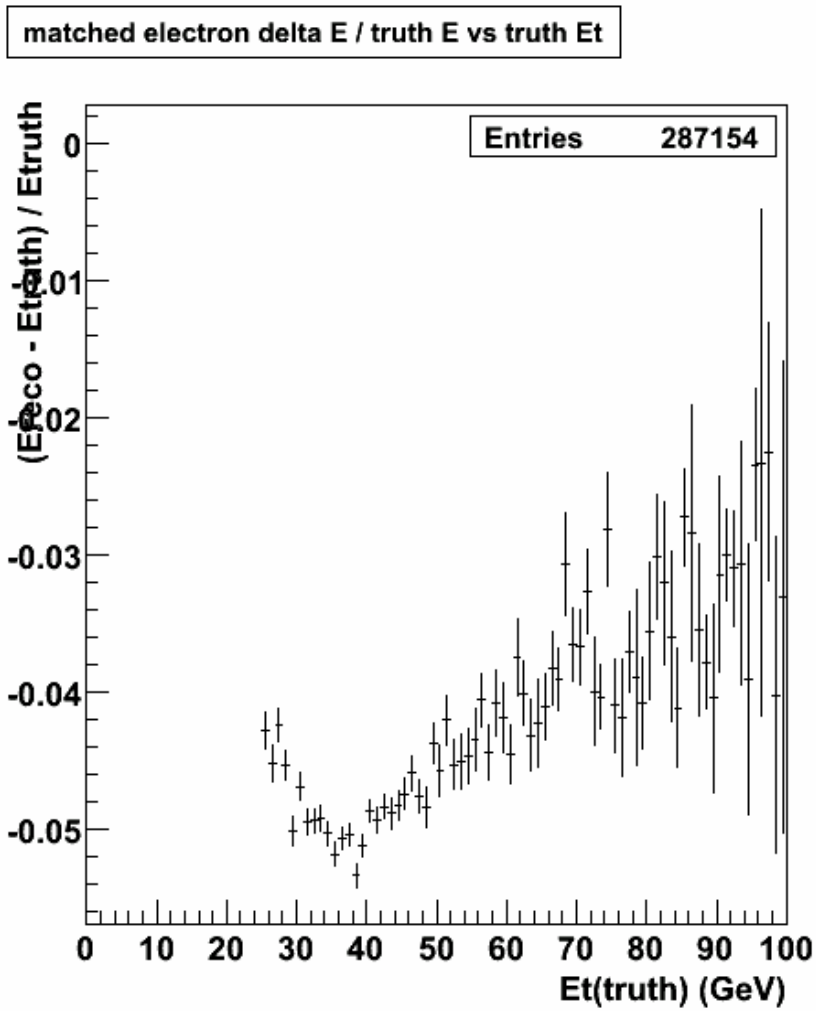




Truth and Reco electron energy matching

E(reco) – E(truth) / E(truth)





Small analysis with SAN

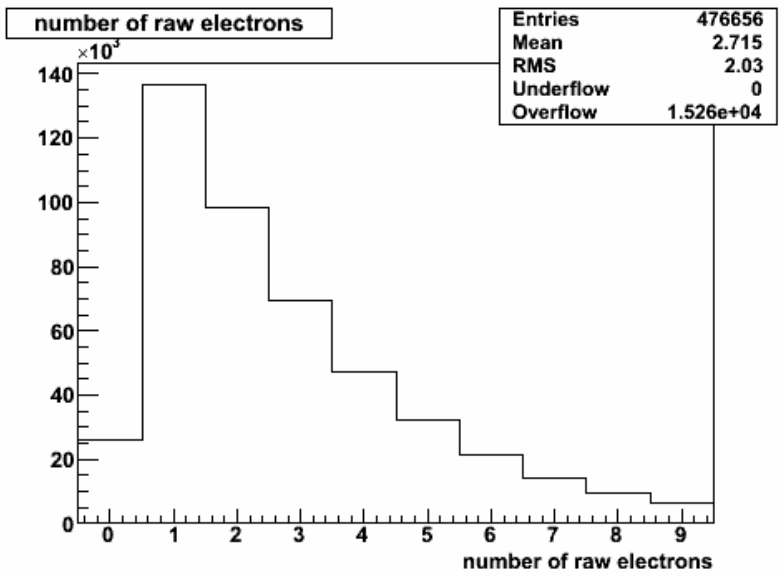
- Analysis using reco electron only
  - define “good” reco electron (after discussion with Thibault)
    - isEM == 0
    - Et > 25 GeV
    - $|\eta| < 1.3$ or $|\eta| > 1.6$ and $|\eta| < 2.4$
  - missingEt obtained from MET_Final
  - look for jets not overlapping with a good electron
    - I only consider as overlapping, jets that are 1 to 1 overlapping with an $e^{\pm}$
- Event selection (goal here is generating $M_T(W)$ distribution)
  - at least one good electron
  - missingEt > 25 GeV
  - no non overlapping jets with Et > 30 GeV

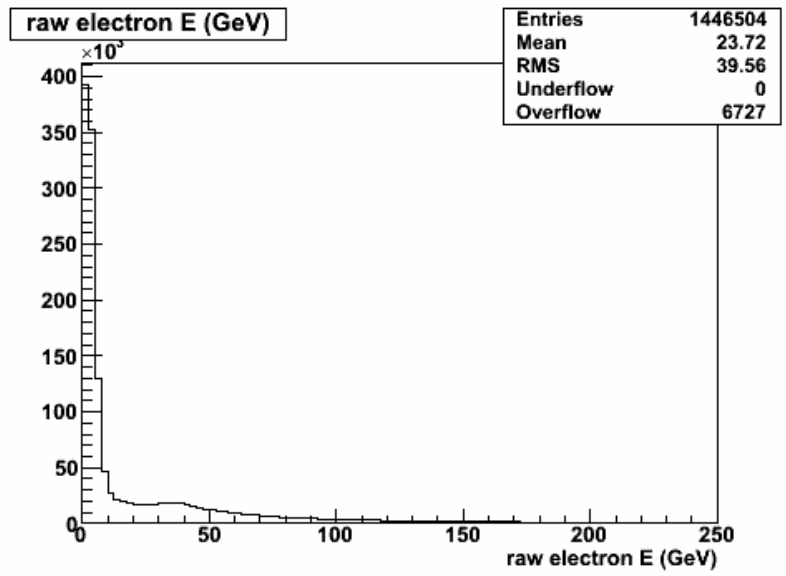
```
events read.....476656
->with at least one good electron.....162284
-->with high missing Et.....143693
--->with no high Et non overlapping jet...106043
```

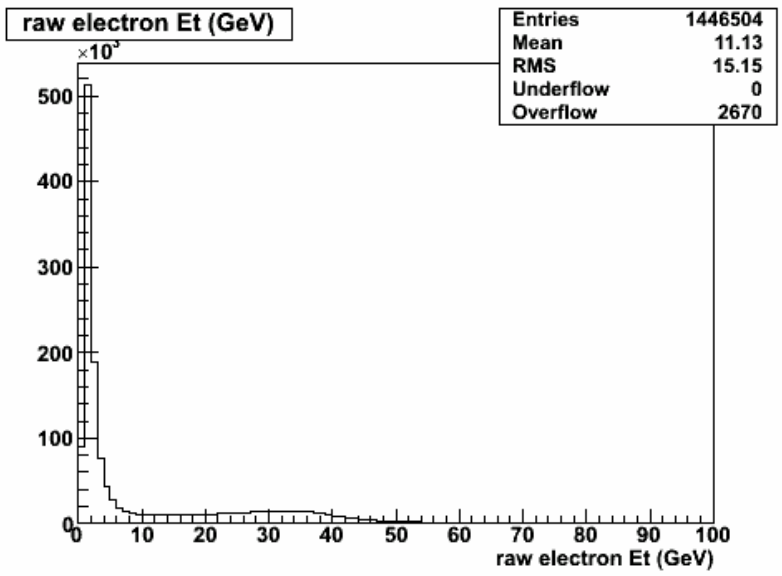
100%
34.0%
30.1%
22.2%

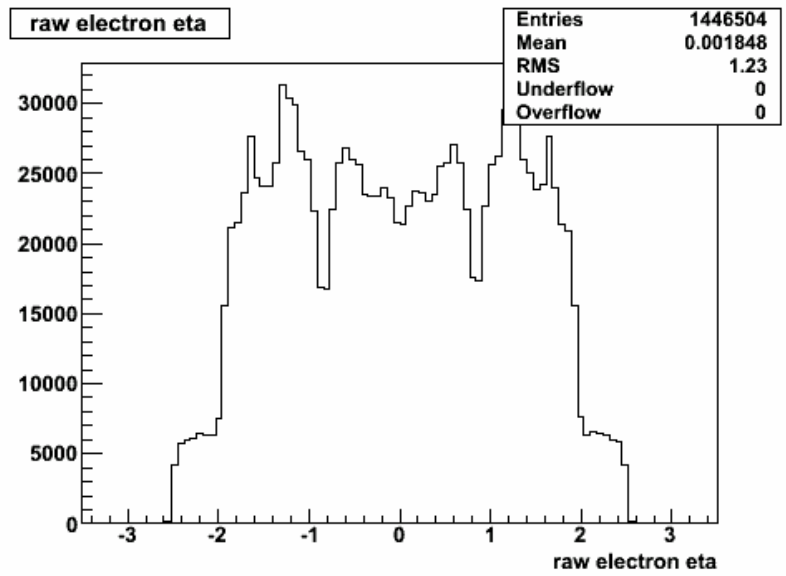
106025 with 1 $e^{\pm}$
18 with 2 $e^{\pm}$

Before selection: electrons

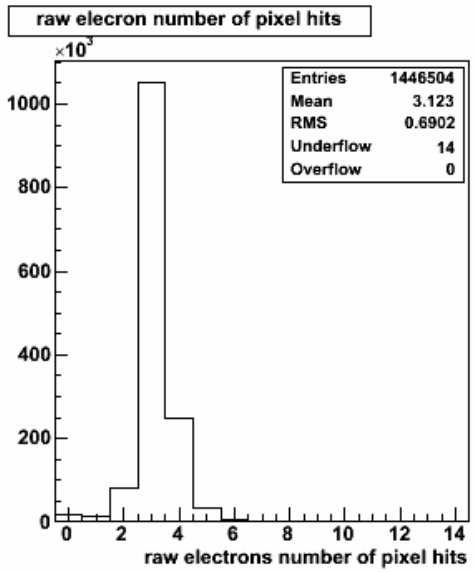


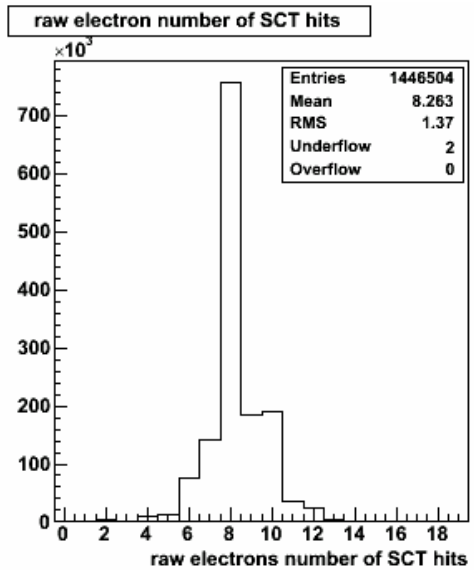


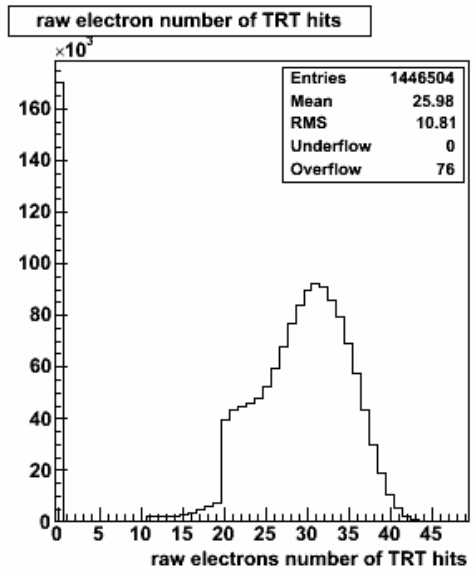


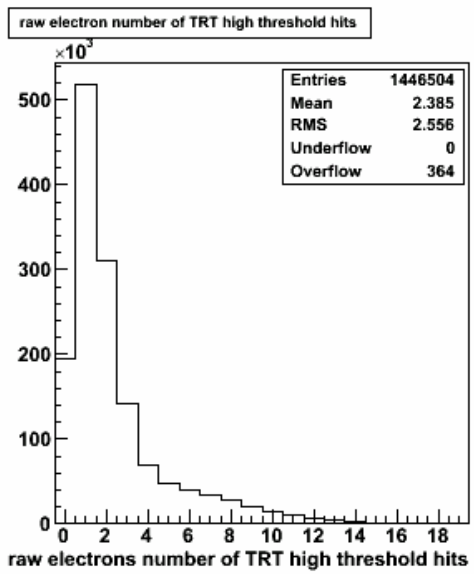


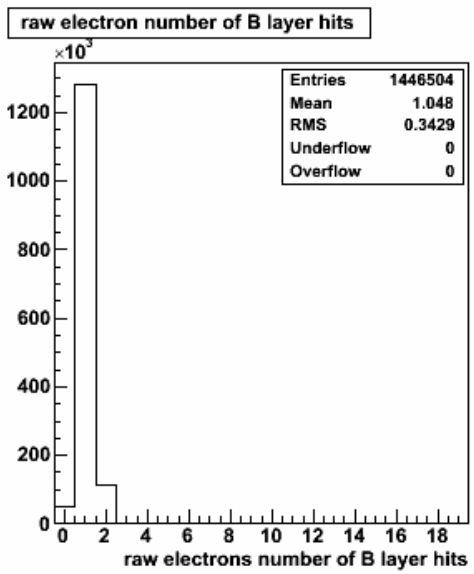
Before selection: electrons

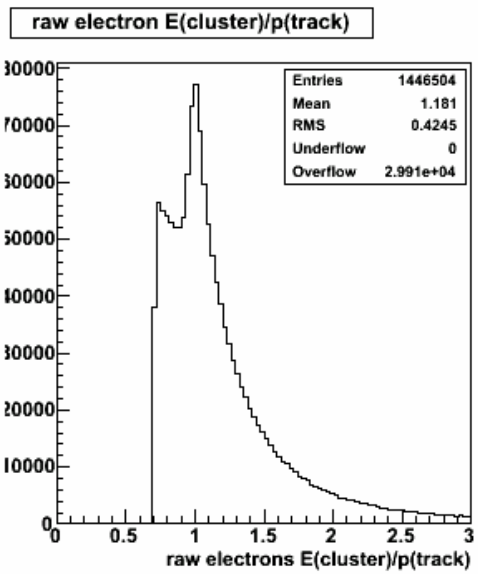




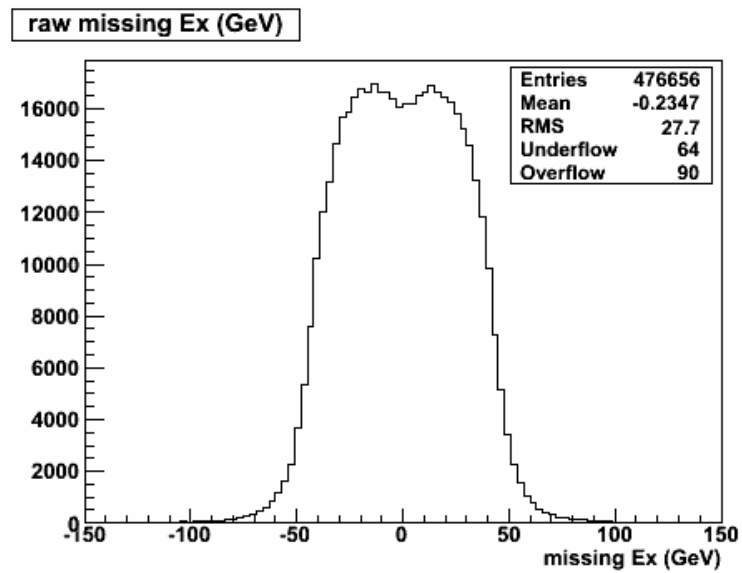


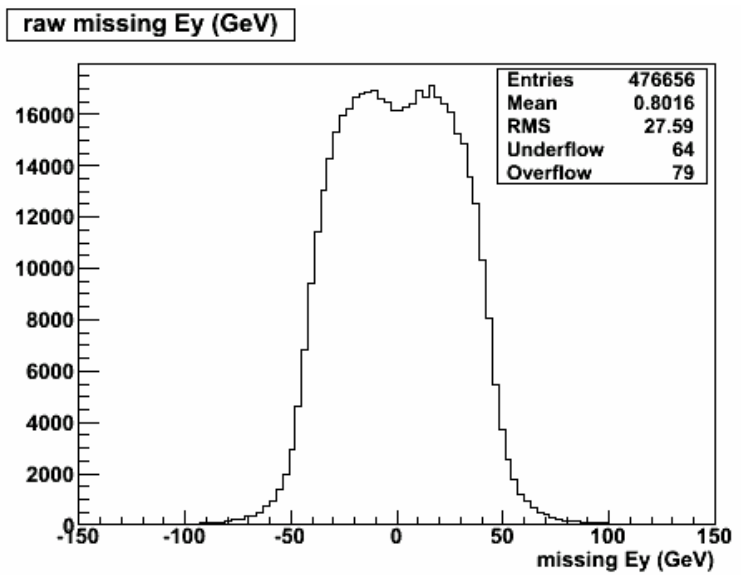


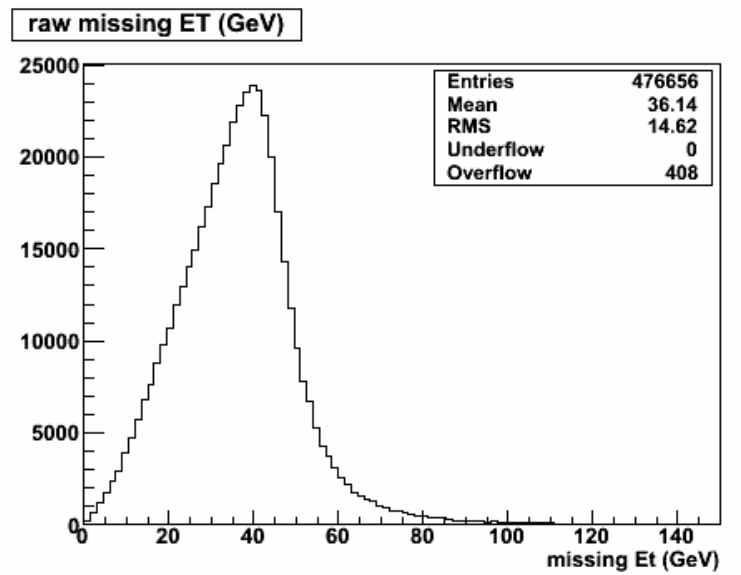


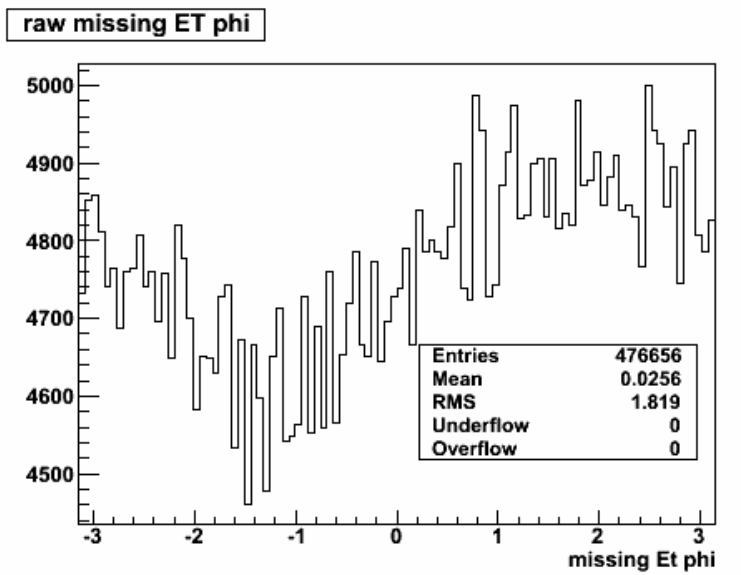


Before selection: missing Et

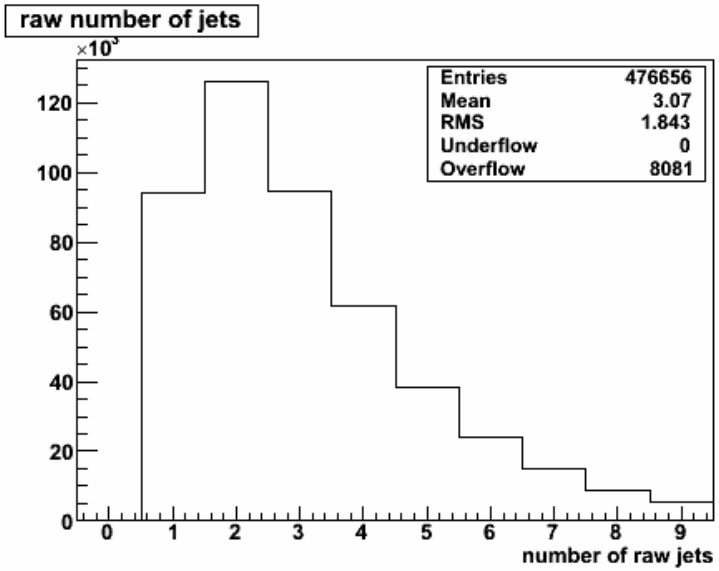


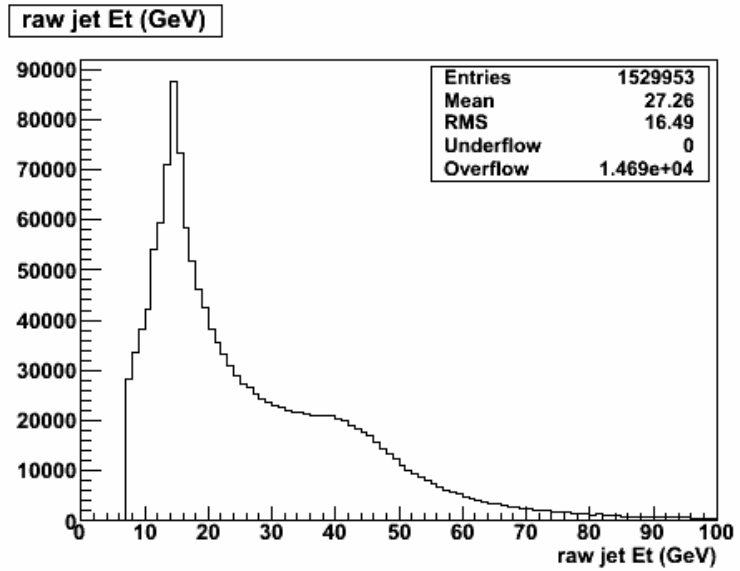


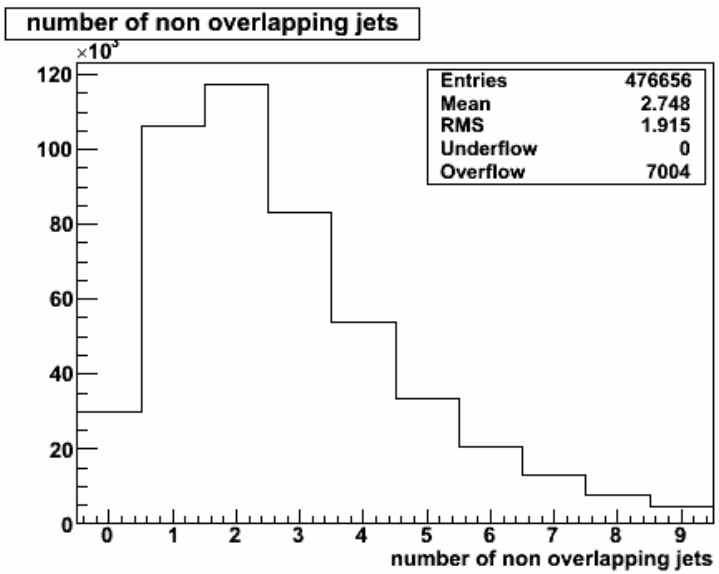


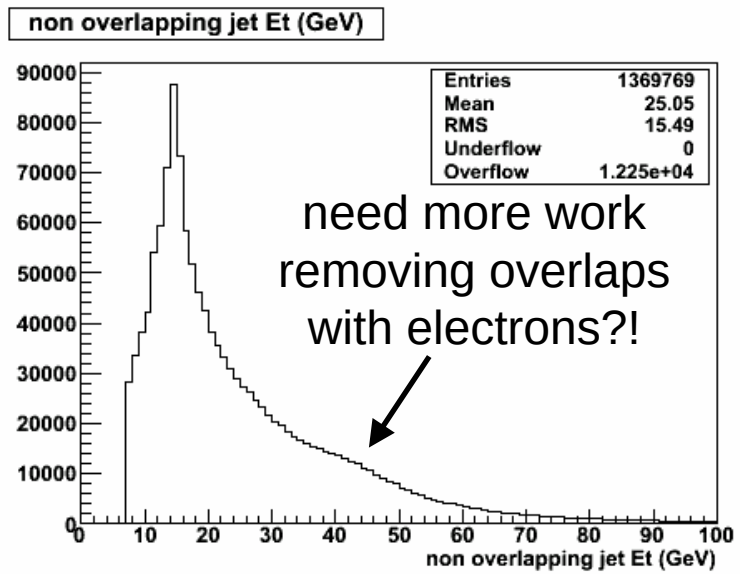


Jets

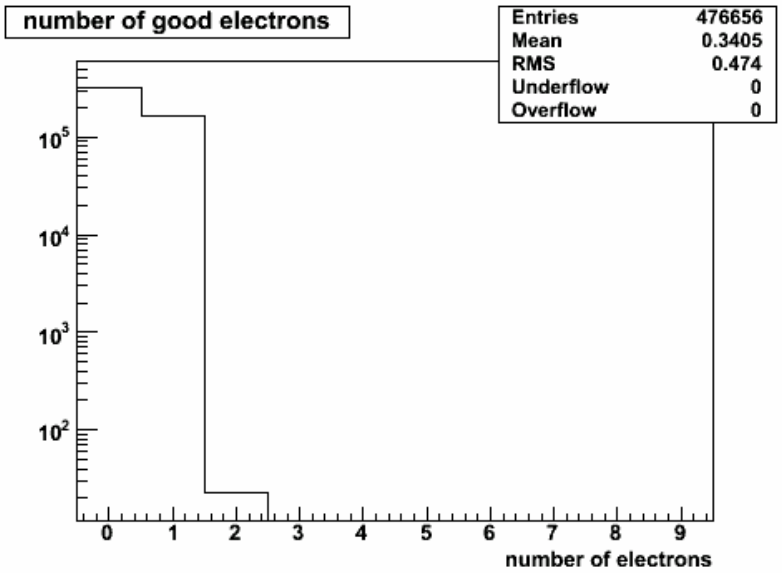

raw number of jets
×10³
Entries 476656
Mean 3.07
RMS 1.843
Underflow 0
Overflow 8081
number of raw jets

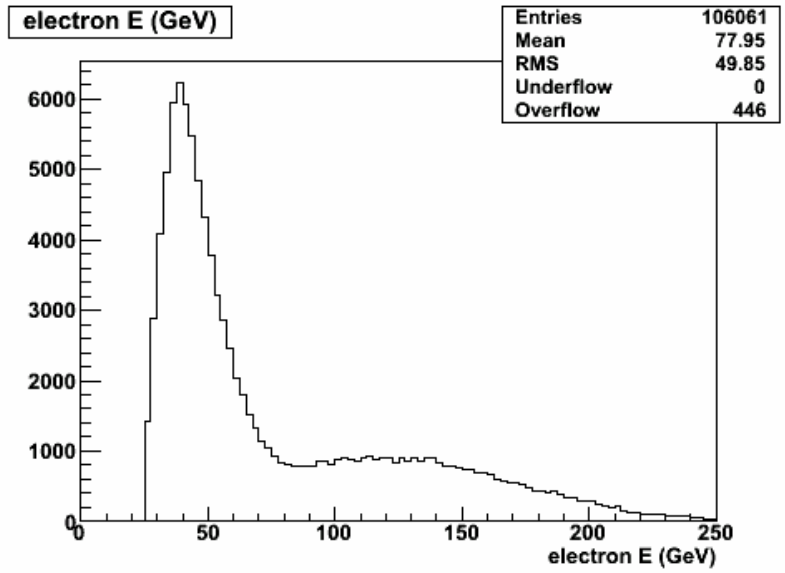

raw jet Et (GeV)
Entries 1529953
Mean 27.26
RMS 16.49
Underflow 0
Overflow 1.469e+04
raw jet Et (GeV)

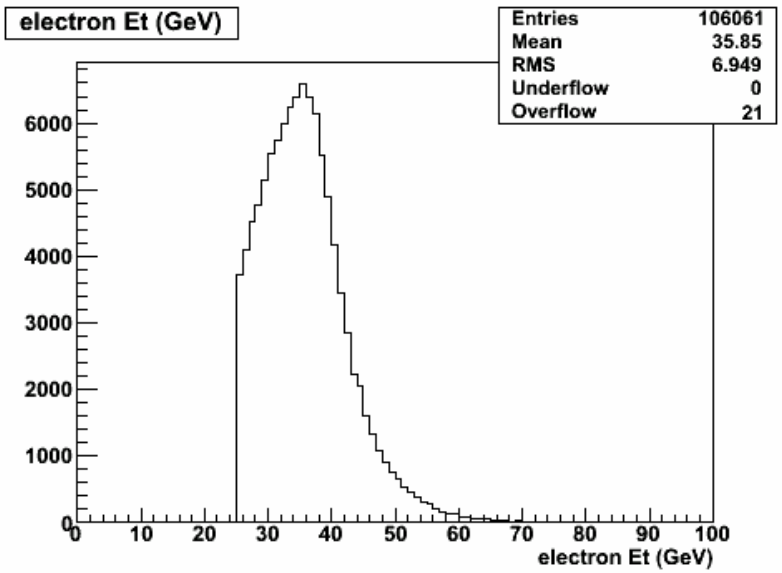

number of non overlapping jets
×10³
Entries 476656
Mean 2.748
RMS 1.915
Underflow 0
Overflow 7004
number of non overlapping jets

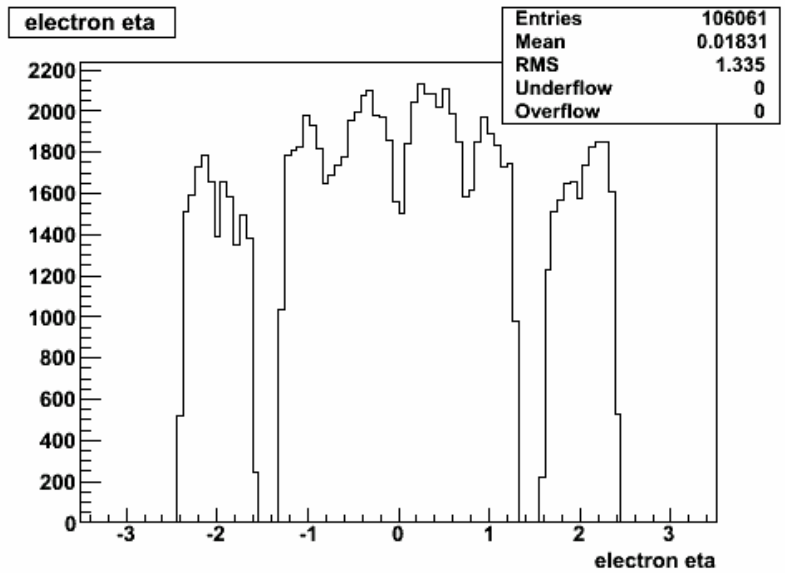

non overlapping jet Et (GeV)
Entries 1369769
Mean 25.05
RMS 15.49
Underflow 0
Overflow 1.225e+04
need more work
removing overlaps
with electrons?!
non overlapping jet Et (GeV)

After selection: electrons



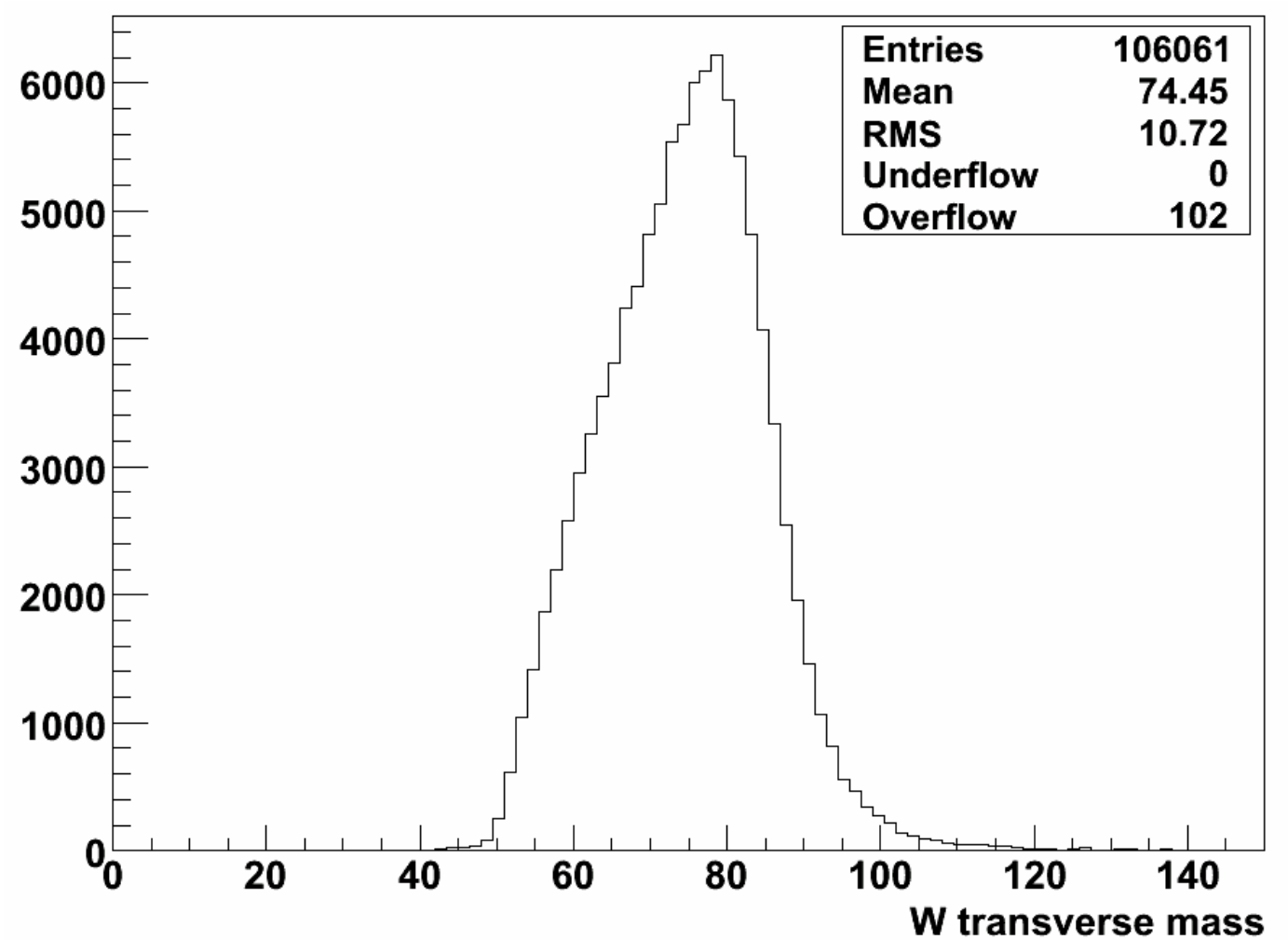






W transverse mass

- 1 entry per good electron
  - only one entry per event for all but 18 events



Root code example

- A code fragment (if you are keen!)

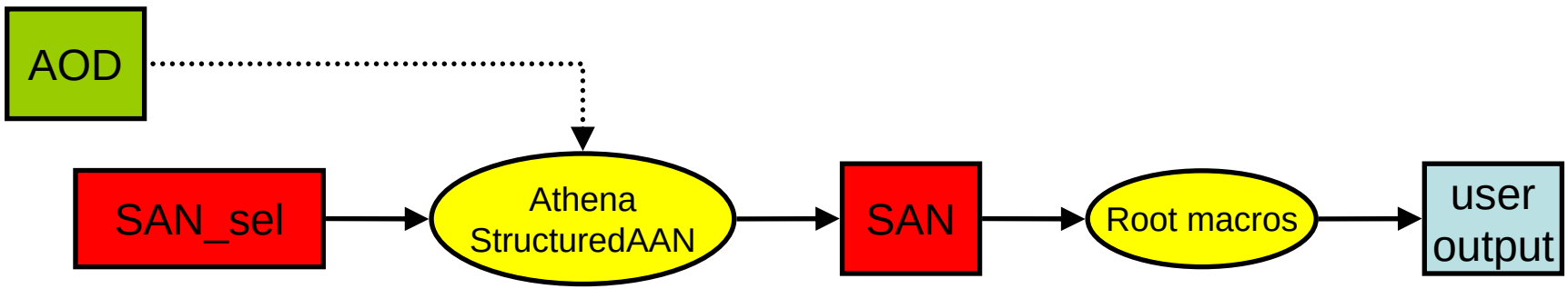
```
// loop over electrons and look for good electrons
 std::vector<Electron> goodElectronV;
 for (std::vector<Electron>::const_iterator Itr  = m_electron->begin(); Itr != m_electron->end(); ++Itr) {
   const Electron& electron = *Itr;
   m_RelectronEt->Fill(electron.et()/GeV);
   m_RelectronNumberOfPixelHits->Fill(electron.numberOfPixelHits());
   // get the electron track and cluster
   const TrackParticle* track = electron.track();
   const CaloCluster* cluster = electron.cluster();
   // compute E/p
   if (cluster && track) {
     double e_over_p = (track->p() > 0.) ? cluster->e() / track->p() : 0.;
     m_RelectronEoverP->Fill(e_over_p);
   }
   // look for good electrons
   if (electron.isEM() != 0) continue;                    // isEM cut
   if (electron.et() < 25.*GeV) continue;                 // Et cut
   double absEta = fabs(electron.eta());
   if ((absEta > 1.3 && absEta < 1.6) || absEta > 2.4) continue;   // eta range cut
   goodElectronV.push_back(electron);
 }
```

Root code execution time

- Analyzing 476 656 events
  - 97 213 normalized time units on BQS
  - elapsed time: about 3h
- Analyzing 2500 events interactively
  - less than 1 min

Using SAN as tag

- Using SAN as tags to run an Athena job
  - use pool_insertFileToCatalog to produce PoolFileCatalog.xml
  - make simple selection
    - missingET > 25 GeV
    - at least one electron with
      - Et > 25 GeV
      - $|\eta| < 1.3$ or $|\eta| > 1.6$ and $|\eta| < 2.4$
  - here, to produce another SAN!



Using SAN as tag, jobOption

- Only jobO available to use SAN as selection not set up to produce trigger branches
  - I produced my own, and it worked
    - trigger independent results were identical with the default or my modified jobOptions
  - In all cases, the relevant lines are

```
EventSelector.InputCollections = [ "SANfile" ]  # for the file SANfile.root to be used as tag
EventSelector.Query="(MET_Final.et() >= 25000.) && (ElectronCollection.et() >= 25000.) &&
    (abs(ElectronCollection.eta()) <= 2.4) && (abs(ElectronCollection.eta()) <= 1.3 ||
    abs(ElectronCollection.eta()) >= 1.6)"
EventSelector.CollectionType = "ExplicitROOT"
```

Using SAN as tag: comparing results

- Run analysis on one SAN file (2500 events)
  - original SAN
  - SAN obtained after event selection using original SAN
- Final results should be identical...
  - but they are not

original

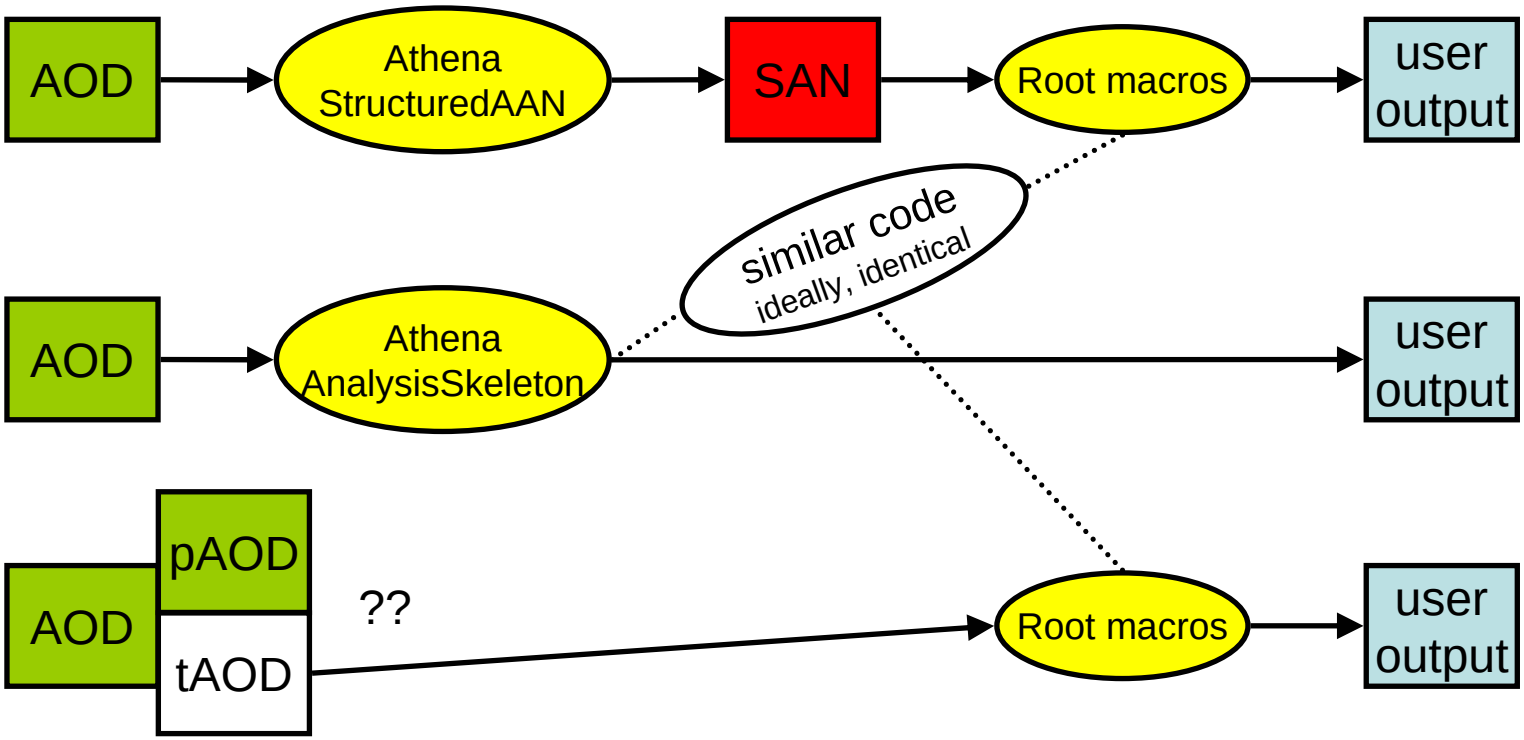
```
events read.....2500
->with at least one good truth electron...1713
->with one good truth electron.....1713
-->passing trigger L1_EM25.....1679
-->passing trigger L2_e25i.....1427
-->passing trigger EF_e25i.....1258
-->matched 1 to 1 with an electron.....1530
->with at least one good electron.....873
-->with high missing Et.....770
--->with no high Et non overlapping jet....579
```

SAN selected

```
events read.....1087
->with at least one good truth electron...1048
->with one good truth electron.....1048
-->passing trigger L1_EM25.....1047
-->passing trigger L2_e25i.....954
-->passing trigger EF_e25i.....904
-->matched 1 to 1 with an electron.....1044
->with at least one good electron.....747
-->with high missing Et.....747
--->with no high Et non overlapping jet....545
```

- $770 \neq 747$, but they should be equal numbers
  - somehow the selection rejects 23 events wrongly
  - after much effort, I still do not understand why

Analysis models: new idea



See talk by RD Schaffer 2007/03/13:
“Accessing transient data objects from ROOT”

http://indico.cern.ch/conferenceDisplay.py?confId=13815

Comments and Conclusions

- Easy to produce SAN from AOD
  - code exists, only need to tune jobOptions for SAN content
- Easy to modify Root macros provided for analysis
  - if you are familiar with Root and C++, of course!
  - much scope for modifications and improvements
    - should aim at reducing compilation time for quicker user turnaround
      - important for complex analyses
      - could split the code in smaller bits somehow
- Future developments should be followed closely
    - I think working with SANs now is useful