Object Oriented Data Analysis in ALEPH

Universität Heidelberg

Richard Cavanaugh
Kirchhoff-Institut für Physik

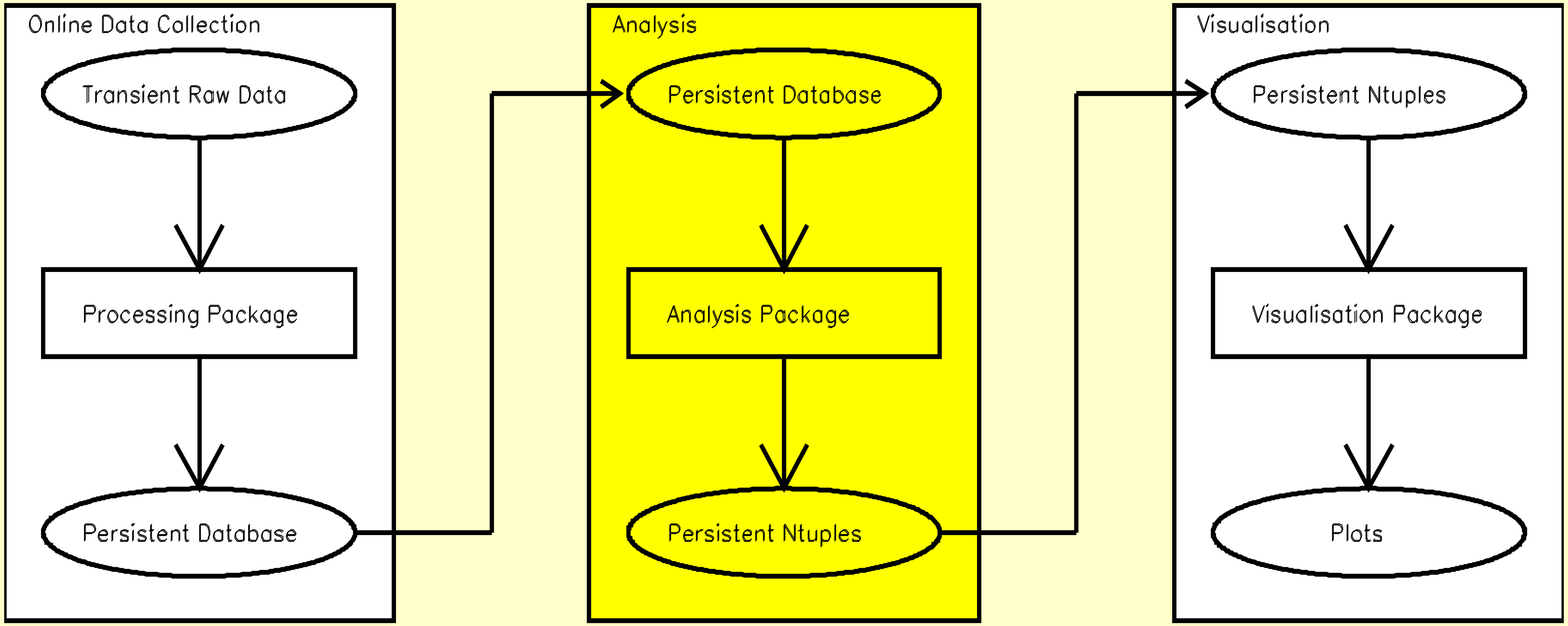
2-5 May, 2001
HepVis, Boston

- Goals
- Data Structure
- Analysis Package
- Case Studies
- Conclusion

Goals

- Convert the ALEPH data from persistent BOS Bank format into persistent a ODBMS
- Rewrite the ALEPH analysis package, ALPHA, into an OO version
- Compare standard versus OO performance
- Provide input for the ALEPH archive project
- Give an opportunity to learn OO programming and design

The Basic Data Flow in ALEPH



Transient Raw Data:	BOS Banks	Processing Package:	JULIA
Persistent Database:	BOS, Objectivity	Analysis Package:	ALPHA, ALPHA++
Persistent Ntuples:	Hbook, HTL, Root	Visualisation Package:	PAW, Lizard, Root
Plots:	Postscript		

The ALEPH Data Structure Model

- ALEPH uses BOS for the persistent memory management:
  - Event data stored in memory in `COMMON/BCS/IW(...)`
  - BOS provides the I/O stuff and the Bank data-structure concept
- The ALEPH data are organised in Banks
- Banks are described in an almost OO language ADAMO (entity-relation)
- ADAMO offers a conversion to C header file (structures)

The ALPHA Data Structure Model

- Translation from ADAMO DDL to C++ classes is trivial

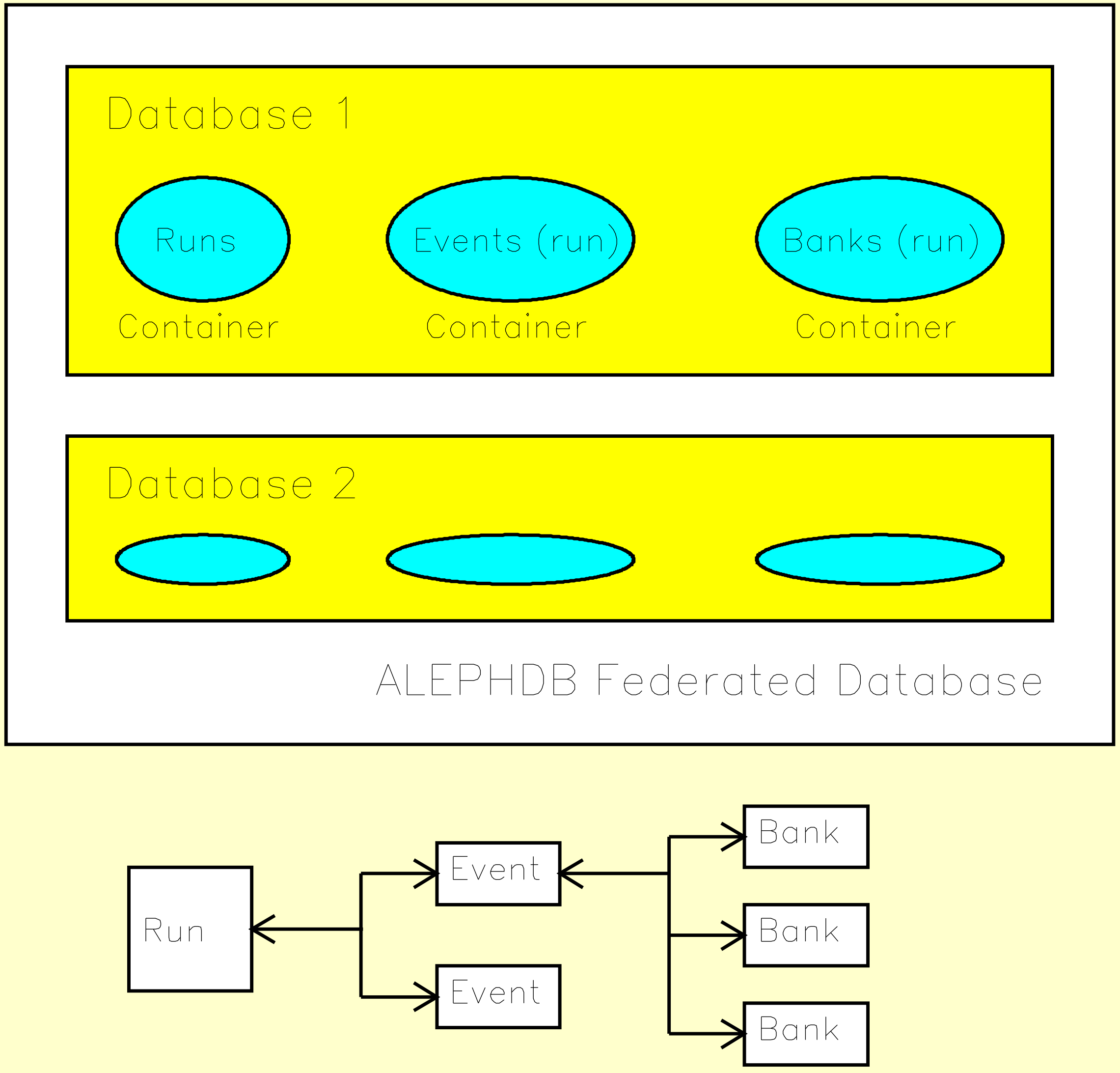
ADAMO DDL

```
FRFT
  : 'Global Geometrical track FiT
    NR=0.(JUL)
    Number of words/track
    Number of tracks'
  STATIC=
  (
    InverseRadi  = REAL[*, *],
    TanLambda    = REAL[*, *],
    Phi0         = REAL[0., 6.3],
    D0           = REAL[-180., 180.],
    Z0           = REAL[-220., 220.],
    Alpha        = REAL[-3.15, 3.15],
    EcovarM(21)  = REAL[*, *],
    Chis2        = REAL[0., *],
    numDegFree   = INTE[0, 63],
    nopt         = INTE[0, 149]
  );
```

C++ CLASS

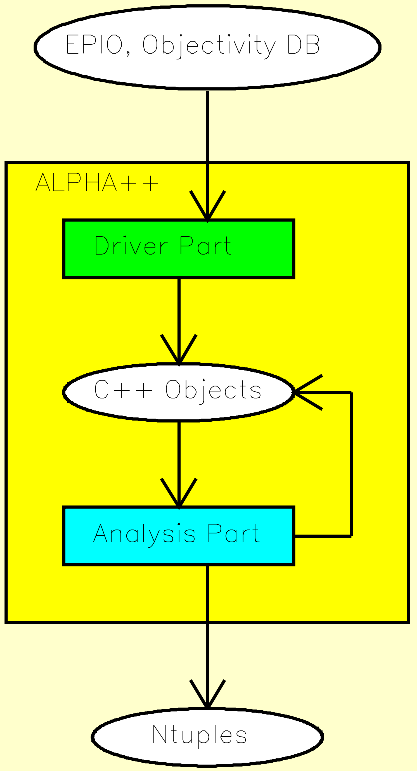
```
class FRFT
{
public:

    FRFT() {} // default constructor


    float InverseRadi;
    float TanLambda;
    float Phi0;
    float D0;
    float Z0;
    float Alpha;
    float EcovarM[21];
    float Chis2;
    int   numDegFree;
    int   nopt;
};
```

The ALEPH Objectivity Federated DB Structure



ALPHA++ Analysis Package: Basic Idea

Put a layer between the DB and the visual part of the analysis:



- Transient objects are built from persistent objects
- Analysis only runs on transient objects.
- Driver part:

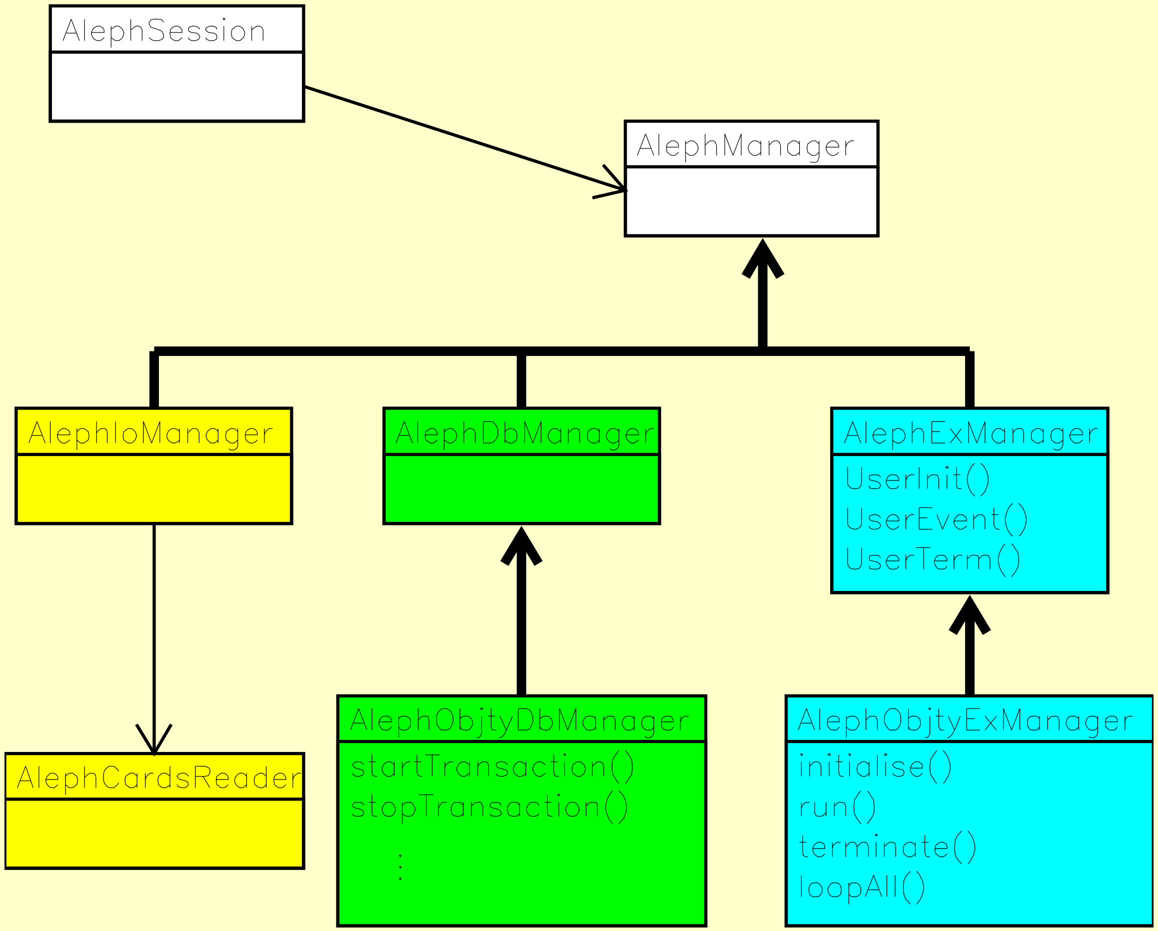
  Persistent DB $\rightarrow$ transient C++ objects (Energy Flow)

  initialise, loop-over events, terminate
- Analysis part:

  Follows the successful structure of the ALPHA Package

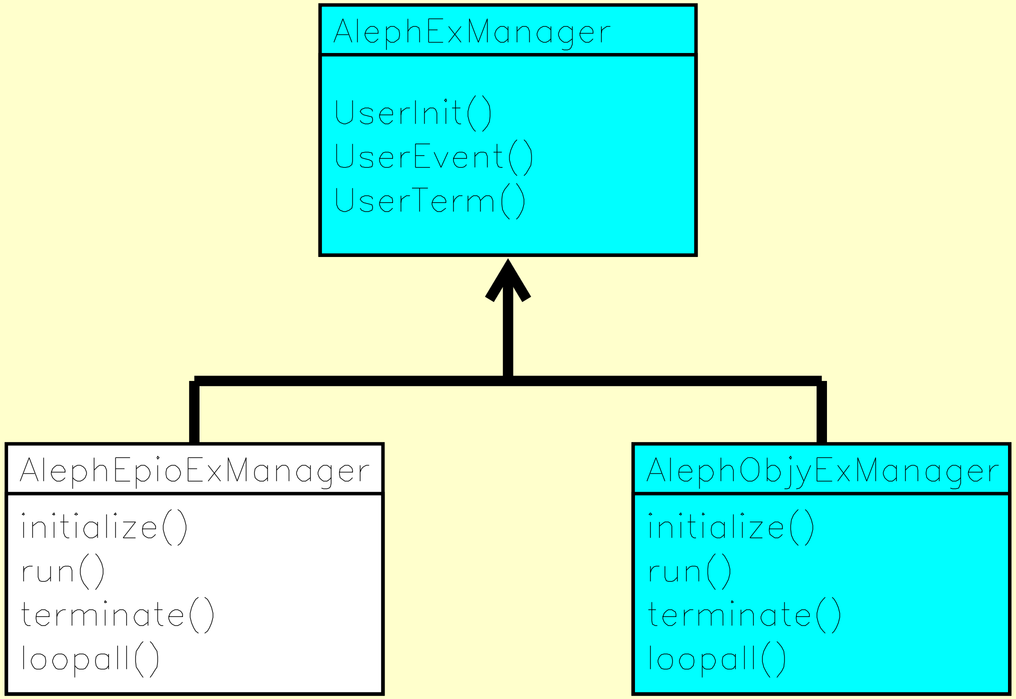
Practical choice: build the ALPHA++ Package from a "wrapped Fortran" version of the ALPHA Package

The ALPHA++ Driver



Persistent ODBMS Current Status

- Almost all of LEP1 data, some LEP2 data, almost no MC
- Not practical to convert all of the MC to ODBMS (data volume $\rightarrow$ CHF)
- Serious analyses must be able to use the BOS Bank DB format



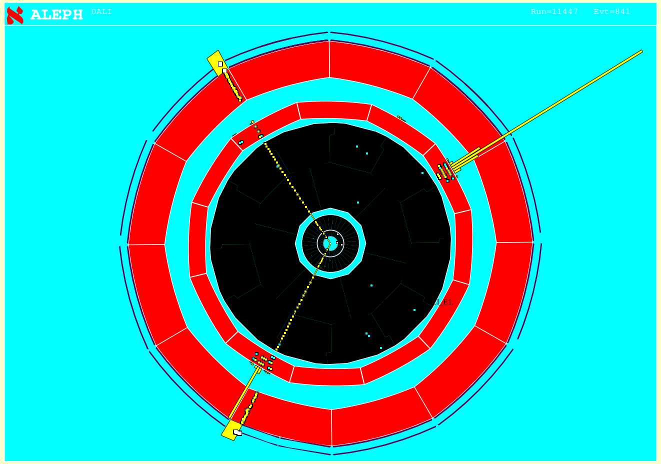
Driver Part: User Routines

In the `AlephExManager` class, only three routines need to be provided by the user:

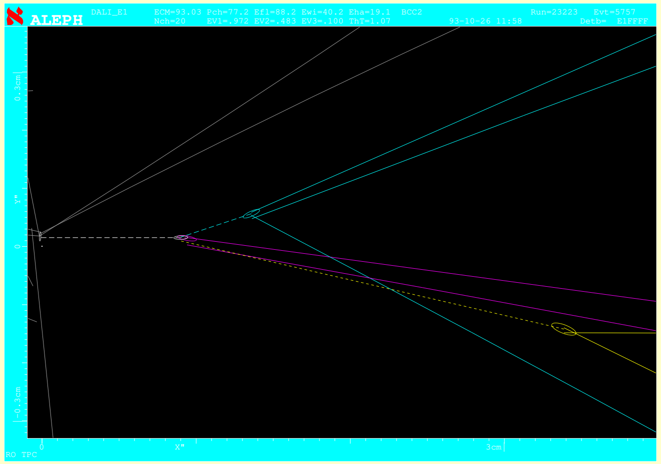
- `UserInit();` Used to open files, book histograms, ntuples, etc.
- `UserEvent( event );` Used for analysis; the current event is passed to the user, ready to be analysed.
- `UserTerm();` Used to close files and anything which needs to be done at the end of the job.

ALPHA: Two basic “collections”

- Generic “Tracks”
  - charged tracks (TPC)
  - photons (ECAL)
  - “Energy flow” (TPC + Cal.)
  - Monte Carlo truth



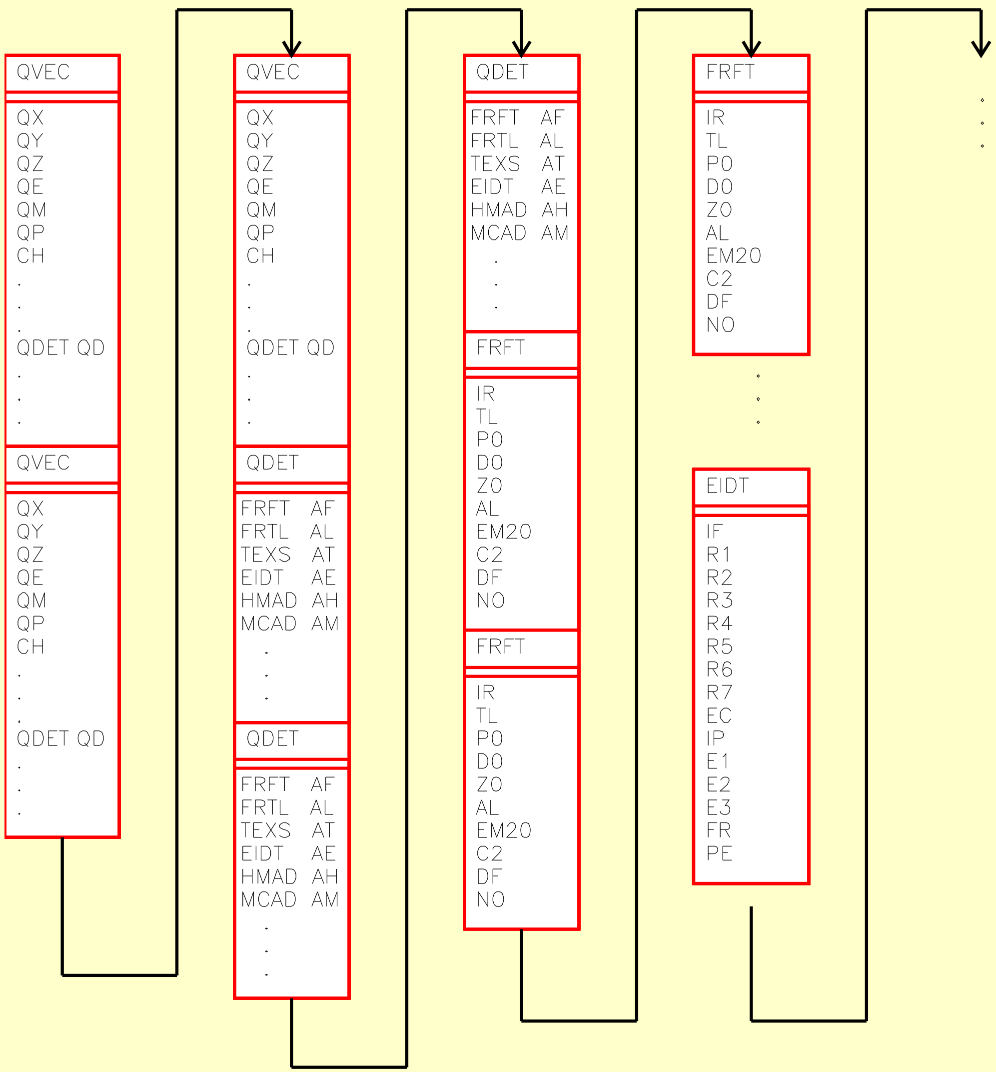
- “‘Vertices”
  - Interaction Point
  - Secondary (tertiery) Verticies



- Possible to “lock” or “unlock” collections
  many algorithms apply only to “unlocked” collections (jet finding, etc.)

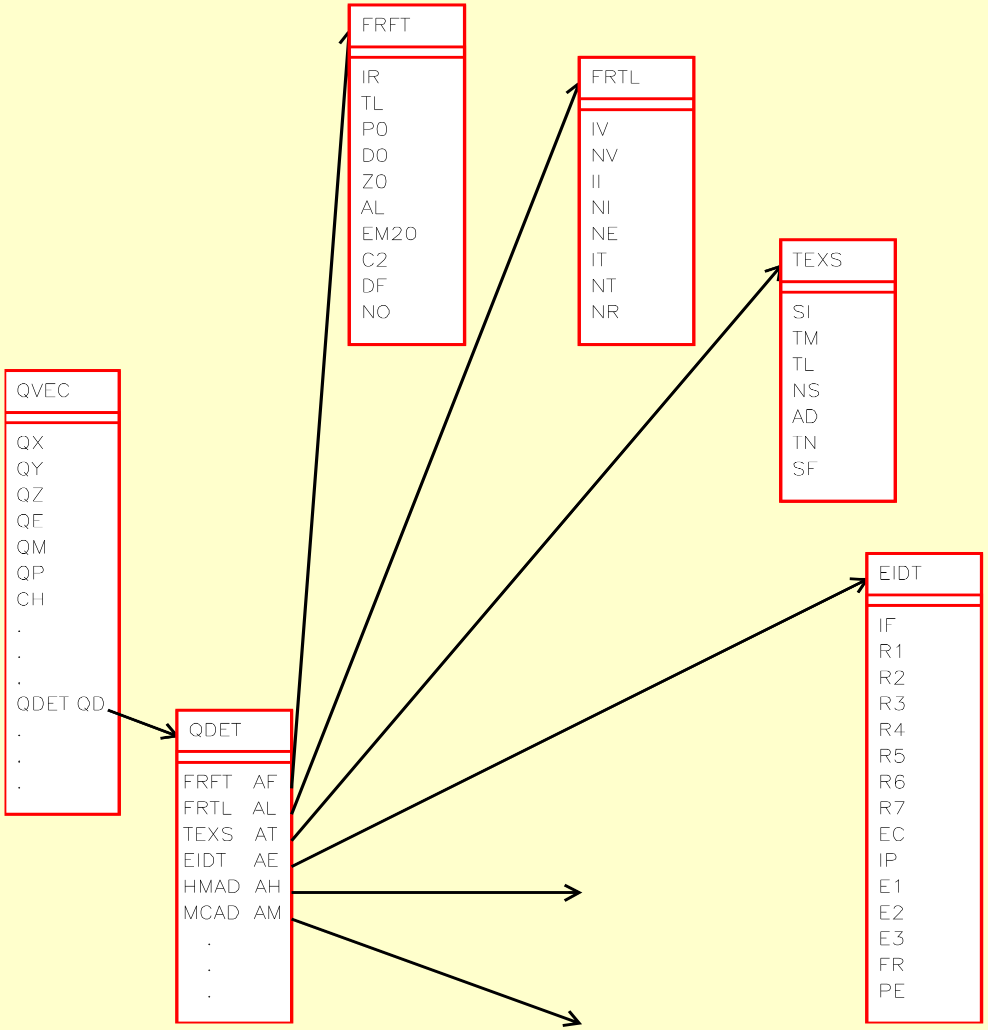
The ALPHA Data Structure Model

- BOS memory management
  - Data stored in `COMMON/BCS/IW(...)`
  - BOS provides I/O stuff and utilities to navigate the event data
- Entity-relation data organised in Banks



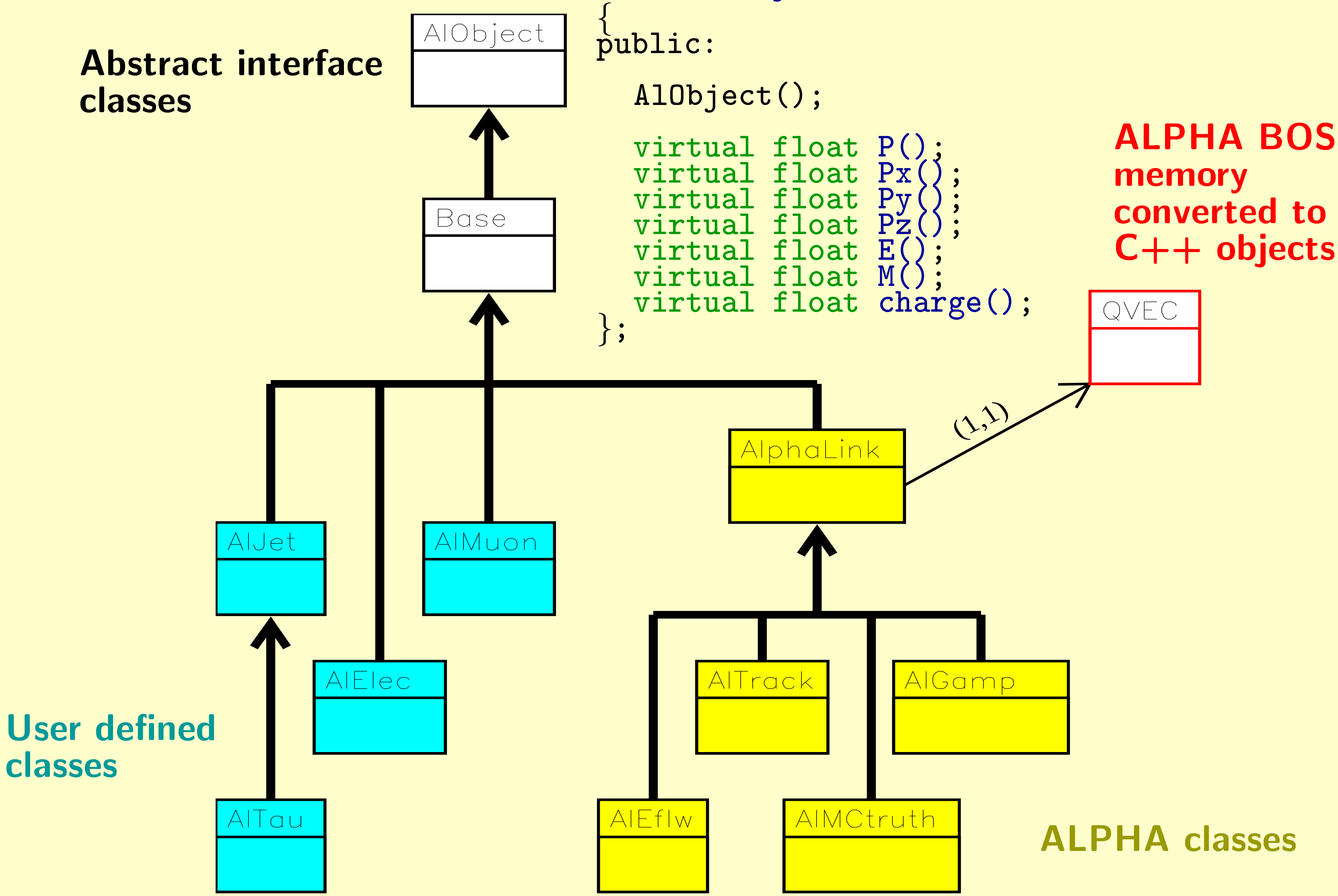
The ALPHA++ Data Structure Implementation

- One-to-one copy of a BOS Bank to C++ object (including relations)
- QVEC is the fundamental "4-vector" like object from ALPHA
- QVEC has a pointer to a detector information class QDET
- QDET then has pointers to FRFT (helix fit), FRTL (tracking), TEXS (dE/dx), etc.



The Analysis Part: Transient Objects

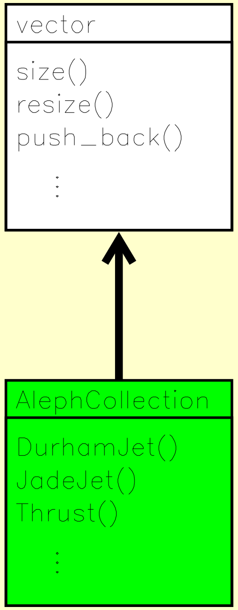
```
class AlObject
{
public:

   AlObject();

   virtual float P();
   virtual float Px();
   virtual float Py();
   virtual float Pz();
   virtual float E();
   virtual float M();
   virtual float charge();
};
```



Analysis Part: Use of the Standard Template Library

ALPHA++ uses STL to define collections of objects:

- The class `AlephCollection` inherits from the STL class `vector`.
- each element in an `AlephCollection` corresponds to a transient C++ object (e.g. AlTrack, AlJet, AlTau, etc.)
- inter-object relations are automatically preserved (e.g. Mother-daughter)

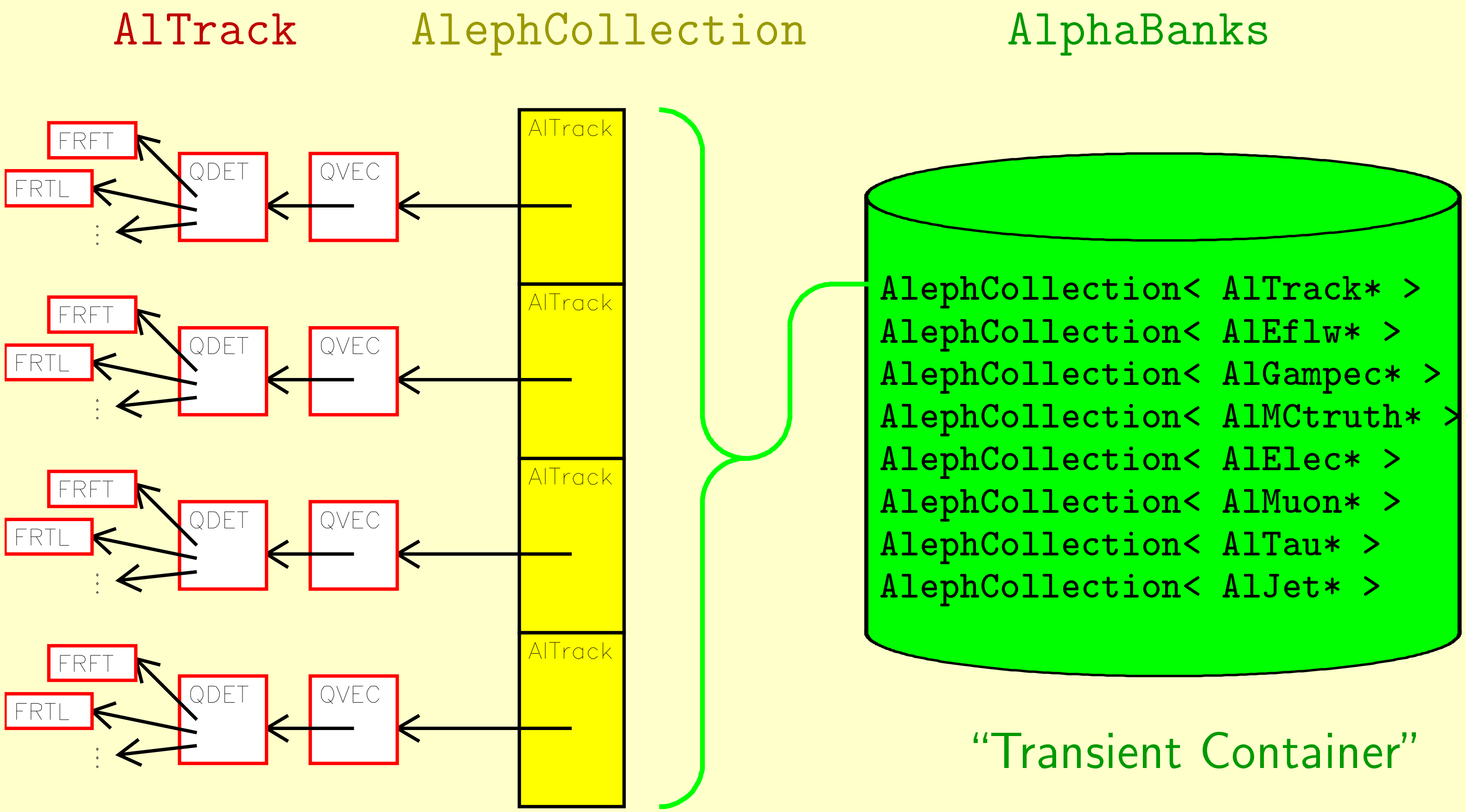


- `AlephCollection` allows one to define physics algorithms on itself (e.g. jet finding)

User code is simple, compact, readable, modular...

Analysis Part: The AlphaBanks Container

The class AlphaBanks serves as a container for all transient objects.



"Transient Container"

Analysis Part: Algorithms

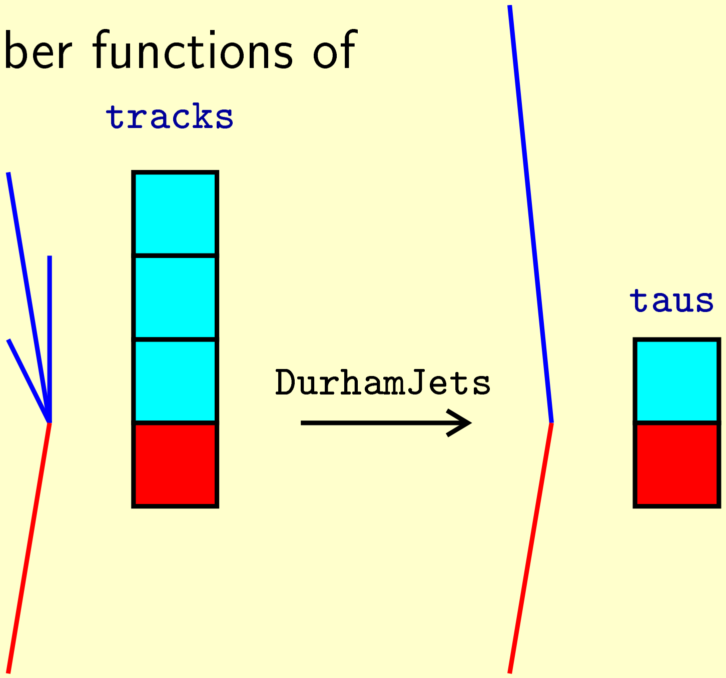
Topology algorithms are defined as member functions of `AlephCollection`:

Jet finding (Durham, Jade, etc.),

Thrust,

Sphericity,

etc.



Allows the user to produce a collection of jets from charged tracks, or energy flow objects, etc.

```
AlephExManager::UserEvent(AlphaBanks event)
{
 AlephCollection< AlTrack* > tracks = event.AlTrackPV();
 AlephCollection< AlTau* > taus = tracks -> DurhamJets( ycut,scheme,Evis );
 AlephCollection< AlTau* >::iterator tauItr = taus.begin();
}
```

Analysis Part: Filtering

Filter algorithms are defined as member functions of the abstract AlObject:

`tauItr -> lock();` (lock, i.e. flag, this `AlTau` object)

`tauItr -> lock(1);` (recursively locks all tracks used to build tau jet object)

`tauItr -> unlock();` (unlock this `AlTau` object)

...and as member functions of `AlephCollection`:

`taus.looperase( tauItr );` (remove tau object from `taus`)

`taus.FilterTau( pmin, ...);` (remove all objects not passing $\tau^{\pm}$ user cuts)

This allows the user to filter a collection of candidates according to sophisticated needs. For example, the topology algorithms only use objects which are *unlocked*.

Analysis Part: Sorting

By overloading the `<` operator (used by STL to compare two objects), an `AlephCollection` can be easily sorted according to Energy, Pt, etc. using the STL `sort` function:

tracks:

15	7	25	20

```
AlObjectBase::SortCriterium = ENERGY;
sort( tracks.begin(), tracks.end() );
```

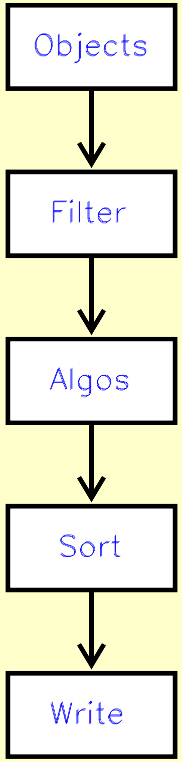
tracks:

25	20	15	7

Structure of an ALPHA++ analysis

Typical structure of the ALPHA++ `UserEvent` routine:

1. get collections of pointers to the basic objects
2. apply filtering algorithms and/or loop over the collections
3. produce jets and/or calculate other topological variables
4. apply sorting algorithms to the collections
5. write out ntuple (Hbook, HTL, Root)

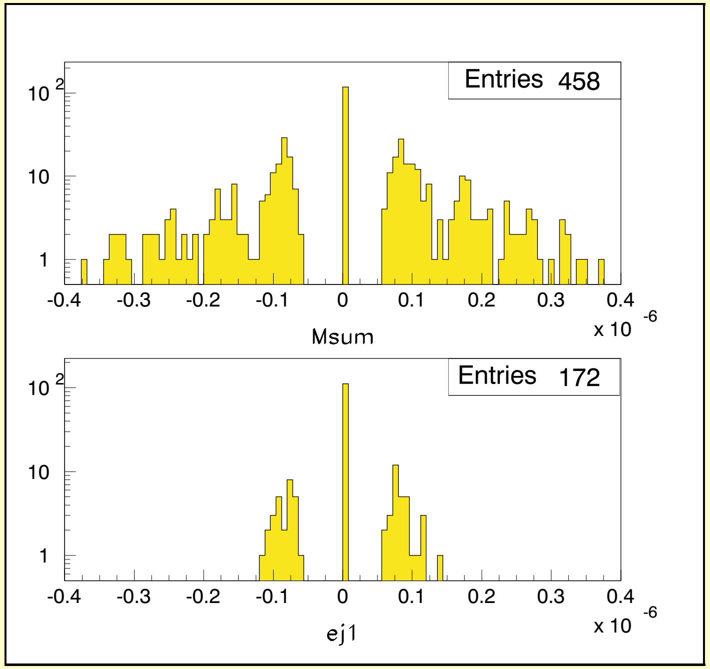


Simple linear structure → helps produce readable code!

Performance

	CPU (sec)/ev	Init. (sec)
ALPHA	1.9×10^{-3}	1.48
ALPHA++	2.6×10^{-3}	1.75

- Benchmark made before STL was introduced to ALPHA++
- Both ALPHA and ALPHA++ "unpack and fill" BOS banks into memory
- Difference (at that time) was primarily due to I/O from Objectivity/DB
- Performance with STL is still acceptable (benchmarks not yet done).



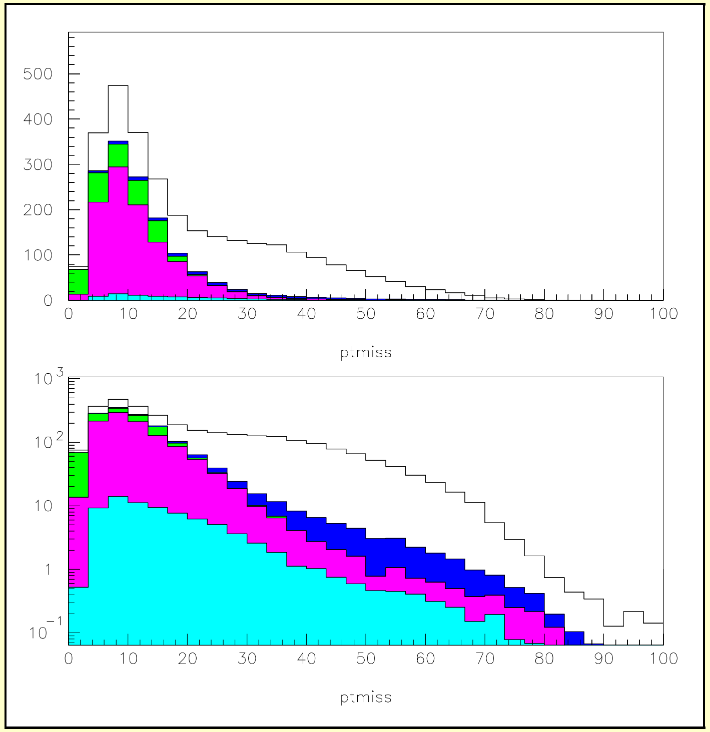
Plotted is **ALPHA - ALPHA++** for:

- Msum: visible mass in the event
- ej1: most energetic jet in the event

Difference is $\mathcal{O}(10^{-7})$!

Invisible Higgs Analysis

Full scale $e^+e^- \rightarrow HZ$ analysis:

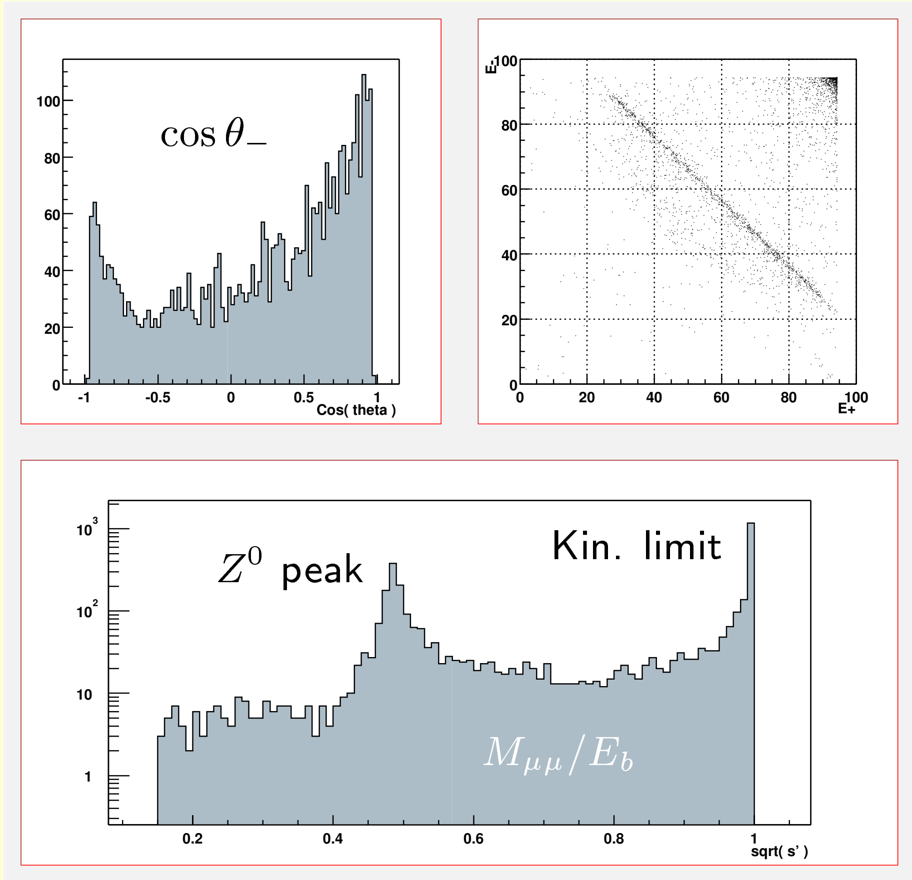


- Performed by Christophe Delaere from University of Louvain la Neuve
- Look for 2 acoplanar jets + missing energy.
- Extensive use of STL (looping and sorting), recursive (un)lock...
- Currently uses "Fortran wrapped" Hbook and PAW...will soon move to Root.

- 10,000 signal MC events
- 1,500,000 background MC events
- Only possible using the standard ALEPH BOS Bank DB

Brief Experience using HTL and Root

- All analyses are very fresh; have tended (so far) to use Hbook because of its familiarity.
- Both **Root** and **HTL** have been interfaced with ALPHA++
- $e^+e^- \rightarrow \mu^+\mu^-$ analysis using **Root**.
- $e^+e^- \rightarrow hadrons$ analysis using **HTL**.
- Good, but brief, experience with both...

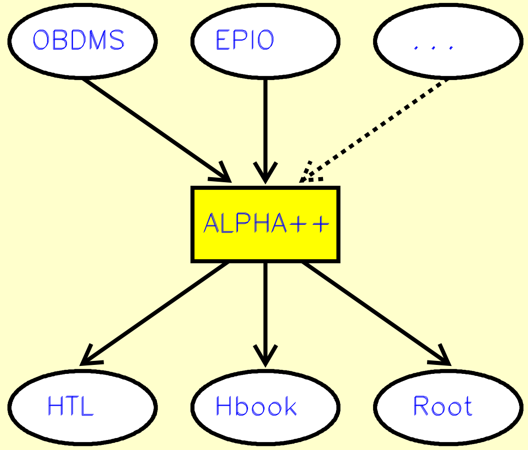


Conclusion

Lessons learned:

- Modularity allows DB to be transparent to user

  Allows one to change the DB according to advances in technology...



- Allow users to define their own classes...classes are personal!
- STL was found to be very natural and useful
- Simple, modular structure allows readable code!