Object Oriented data Analysis in ALEPH

Giuseppe Bagliesi
INFN - Pisa

EPS HEP99 - Tampere
15-21 July 1999

Outline

- The ALPHA++ project
- The current set-up
  - The ALEPH data structure and its conversion to persistent objects
  - Status of the ALEPH database (ALEPHDB)
  - The analysis program
- Preliminary performances test
- Summary

The ALPHA++ Project: goals...

- convert the ALEPH data from the *BOS* bank (Zebra-like style) into persistent objects and write them to a object database (*Objectivity/DB*)
- rewrite a mini-version of the ALEPH analysis package ALPHA in an object oriented language (C++), based on the Objectivity database
- compare standard and OO performance with regard to efficient access of the data

The ALPHA++ Project: goals (II)

- test the software engineered by the RD45 and LHC++ projects.
- Provide some input/experience for a possible archiving of ALEPH's data
- Give an opportunity to learn OO programming and design

The ALPHA++ project: status

- Release 4.0:
  - upgrade to Objy Version 5.1
  - move from HP to DEC (the preferred ALEPH platform is on Digital Unix)
  - release 2.0 (beta) of the Analysis package
  - ~ 8GB of data written on the Objy/DB

The ALEPH data structure

- Aleph uses BOS for the memory management:
  - Event data are in memory in COMMON/BCS/IW(...)
  - BOS provides the I/O stuff and the utilities to “navigate” in BCS through the BANK concept
- The ALEPH data are organized in BANKS
- The BANKS are described in an “almost” OO language: ADAMO
- ADAMO offers a conversion to C headers files (structures)
- The translation ADAMO DDL $\leftrightarrow$ C++ class is trivial

Example: The FRFT bank

ADAMO DDL

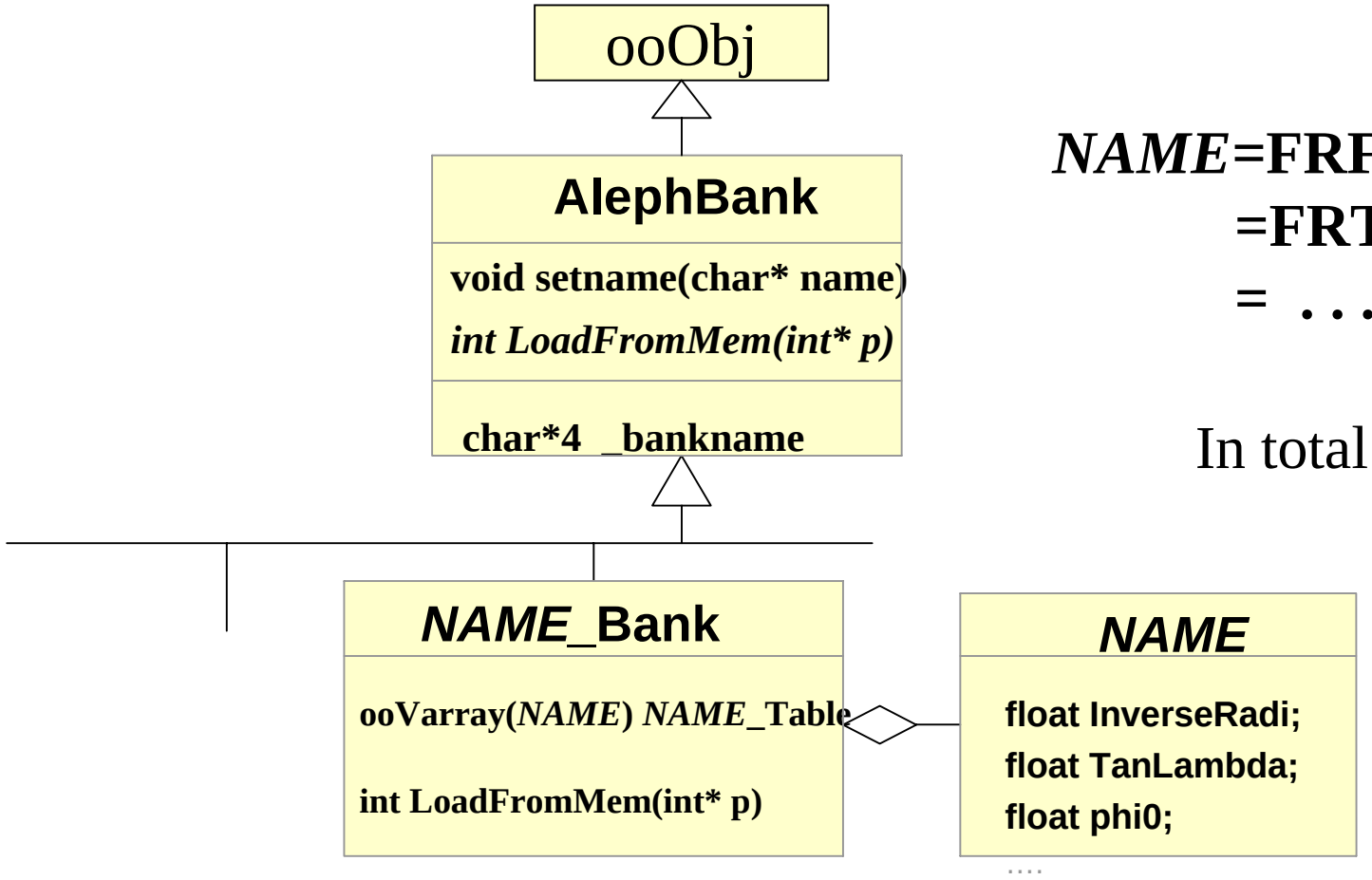
```
FRFT
    :   'Global Geometrical track FiT
        NR=0.(JUL)\
        Number of words/track\
        Number of tracks'
  STATIC
   = (InverseRadi   = REAL [*,*],
      TanLambda     = REAL  [*,*],
      Phi0          = REAL  [0.,6.3],
      D0            = REAL  [-180.,180.],
      Z0            = 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:
// default constructor
  FRFT() {}

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

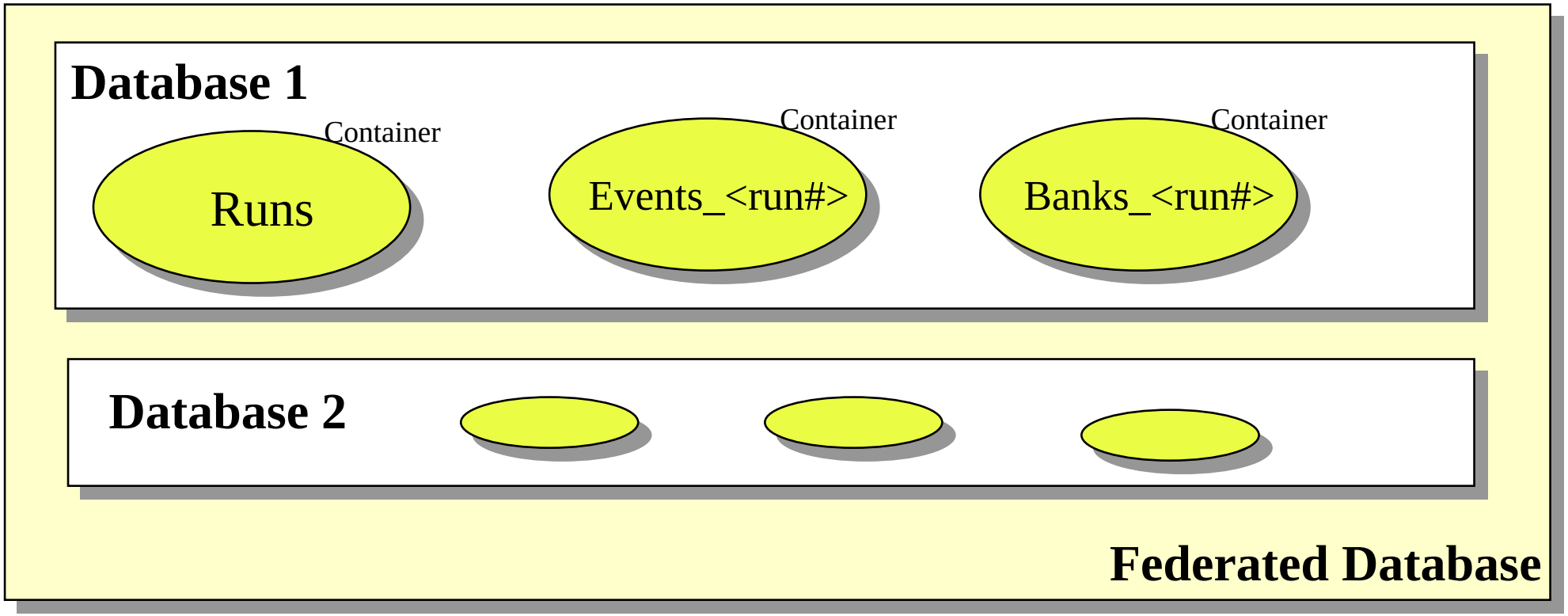
The Objectivity DDL structure

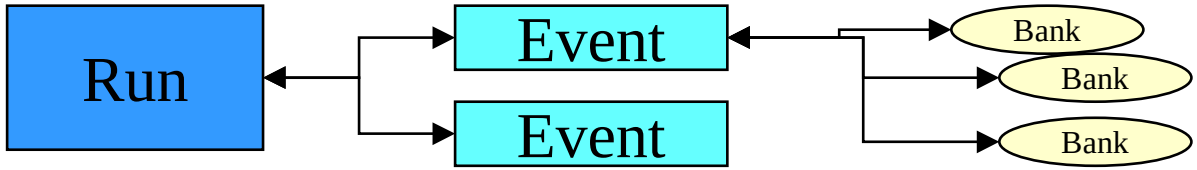


***NAME*=FRFT**
=FRTL
= . . . .

In total : 173 banks!!

The Database structure





Status of the Aleph Database (ALEPHDB)

- Database populated with ~100K 1994 data and ~20K MC events
- In total ~ 8GB written on the Objy database including some LEP2 data
- The ootidy function saves ~5% of the space in the Objy database

Event type	Number of events	Size/event Objy	Time/event Write	Size/event EPIO	Banks/event
POT 1994	102784	12 KB	17 msec	9.5 KB	~19
Class 16	6197	145 KB	111 msec	114 KB	~240
MC 1994 QQ events	17678	150 KB	99 msec	124 KB	~177

Class 16 events: hadronic Z decays

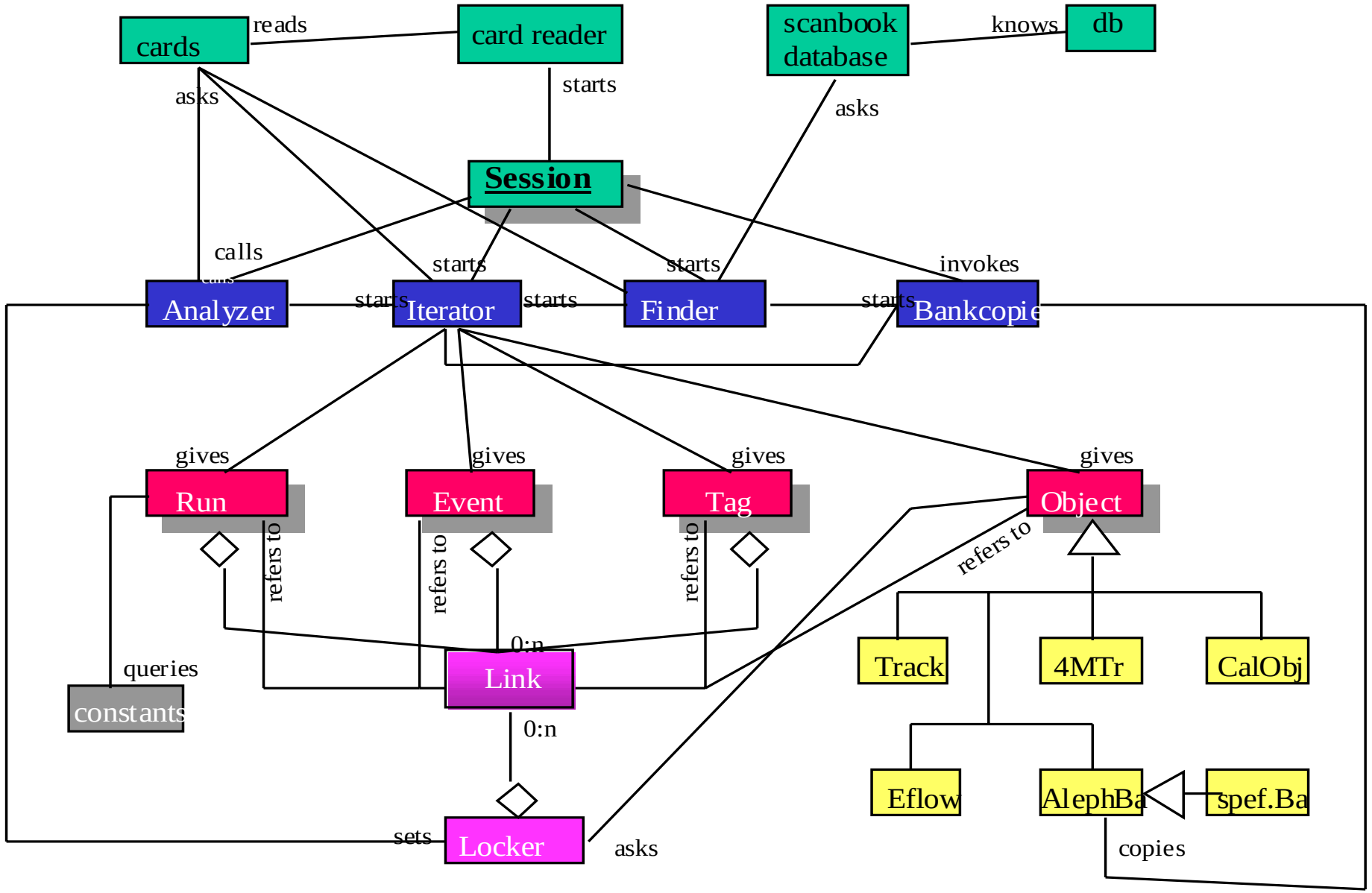
CPU Alpha 8400: 185 CERN units

Status of ALPHA++

- Two simple C++ programs exist:
  - populateDb:
    - read the aleph EPIO data files and populate the Objectivity/DB
  - readDb:
    - Loop over the events and over the banks
    - Copy the BANKS from Objectivity in memory to the BOS common (FORTRAN)
- With the banks stored in the BOS common it is possible:
  - To run the "standard" ALPHA reading the events from Objectivity and calling the FORTRAN from C++
  - To run DALI (the ALEPH event display), reading the events from Objectivity
  - To simplify the development of the OO analysis program by using many algorithms already developed in FORTRAN

Entities and Relationships

Preliminary analysis model



The analysis program (ALPHA)

- How does the ALEPH analysis program (ALPHA) work ?
- Two basic “Objects”:
  - “Tracks” (data structure QVEC)
    - charged tracks (TPC)
    - photons (ECAL)
    - Energy flow Objects (TPC + Calorimeters)
  - “Vertices” (data structure QVRT)
    - Main Vertex (holds the position of the interaction point)
    - General Vertex (reconstructed secondary vertices)
  - In addition it is possible to “lock” or “unlock” single objects in order to select them in the current analysis
    - many algorithms are applied only to “unlocked” objects (jet finder, thrust, eflow ...)

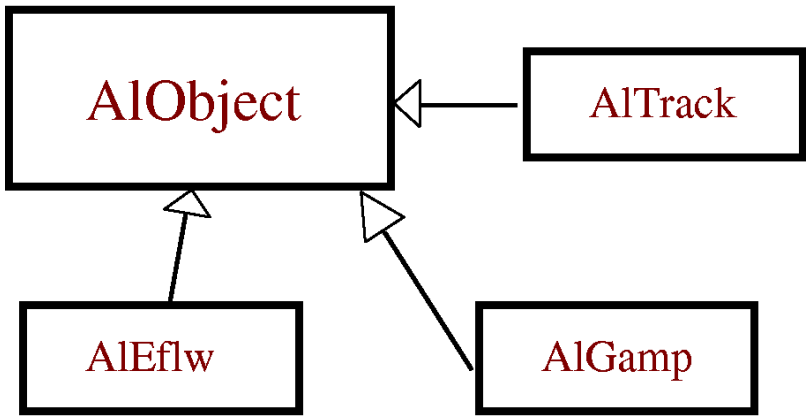
The analysis program (ALPHA++)

- Basic ideas:
  - put a *layer* between database and analysis
  - *transient objects* are built from the persistent ones, and the analysis runs only on these transient objects
- Practical choice: develop a preliminary “FORTRAN wrapped” analysis program
    - in a short time scale, an analysis program already working has been developed
    - use this preliminary version as a basis to develop new C++ code and algorithms
- For each event:
  - the relevant persistent classes are read from the ALEPHDB and the corresponding BOS banks are filled;
  - the internal QVEC and QVRT data structures are filled;
  - the transient C++ classes are instantiated using the data contained in QVEC and QVRT

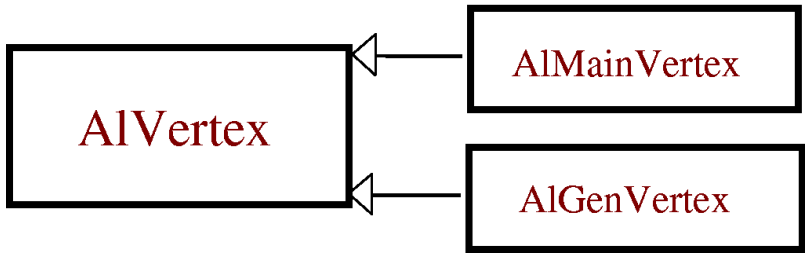
The analysis program (contd...)

The current version of the ALPHA++ *analysis program* is based on the *same ideas* and *data structures* of ALPHA

“TRACKS”



“VERTICES”

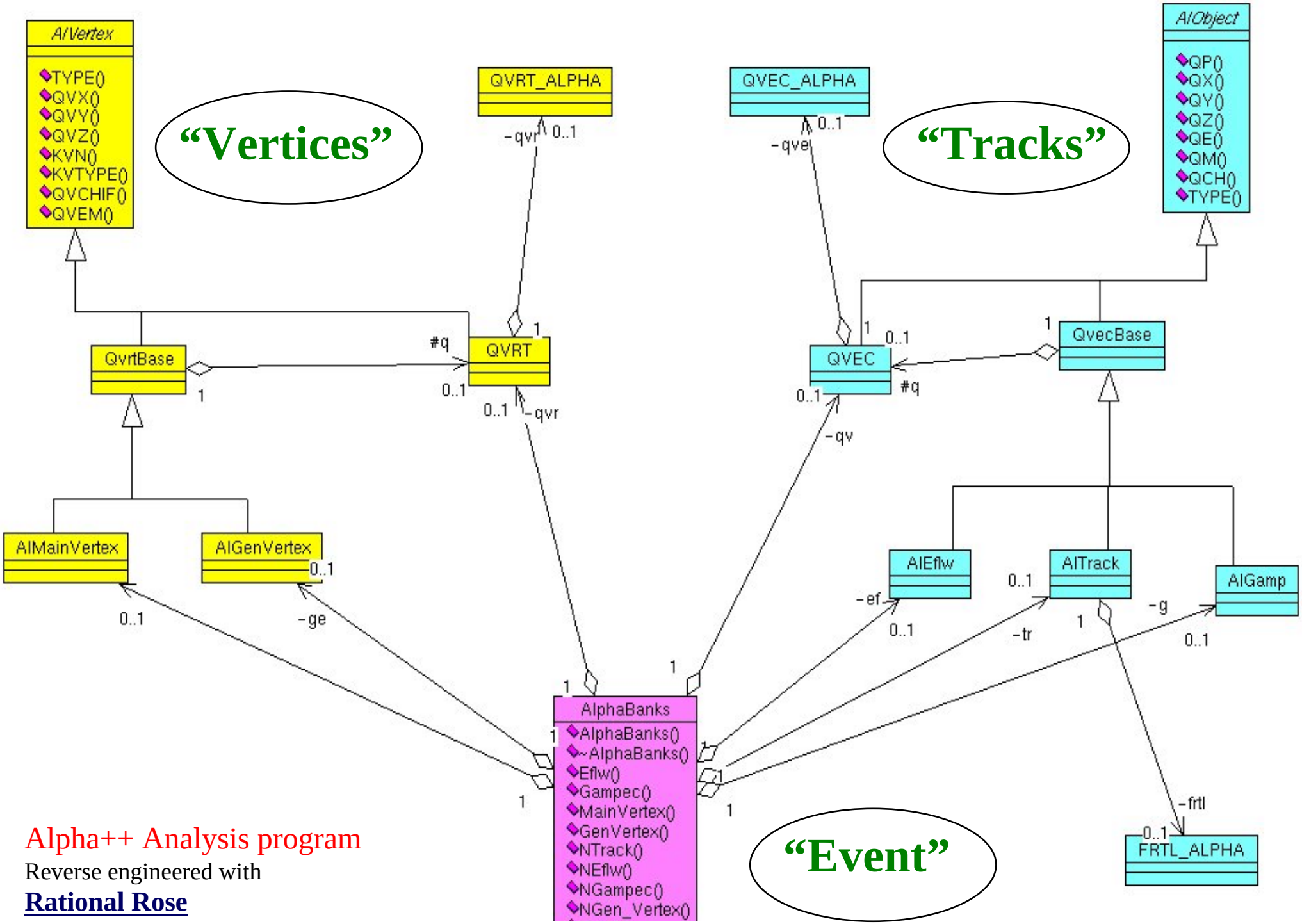


The analysis objects...

- In ALPHA the “tracks” have common attributes:
  - QP, QX, QY, QZ …
- Reproduce the ALPHA structure
  - tracks, Eflow, Calobjects, photons…
    - Inheritance from the abstract class AlObject
  - Vertices
    - Inheritance from AlVertex

- Abstract Interface

```
class AlObject {
 public:
  ~AlObject();
  virtual float QP() = 0;
  virtual float QX() = 0;
  virtual float QY() = 0;
  virtual float QZ() = 0;
  virtual float QE() = 0;
  virtual float QM() = 0;
  virtual float QCH() = 0;
};
```



Alpha++ Analysis program

Reverse engineered with

Rational Rose

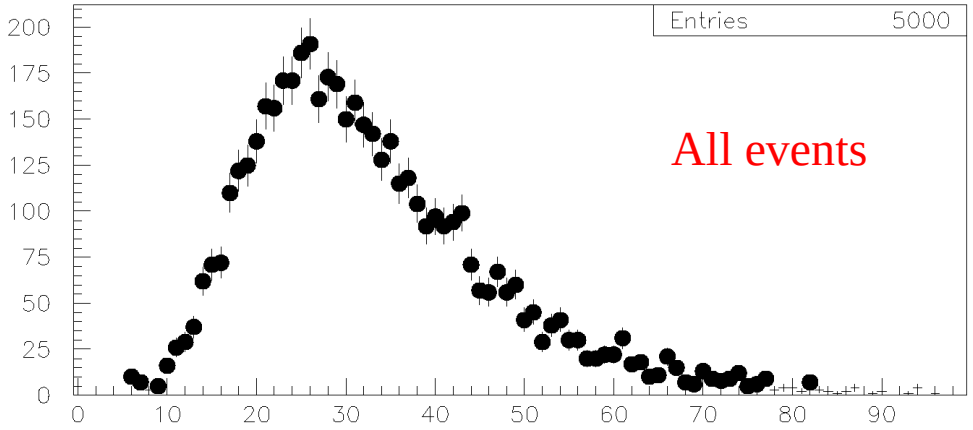
Preliminary performance test: setup

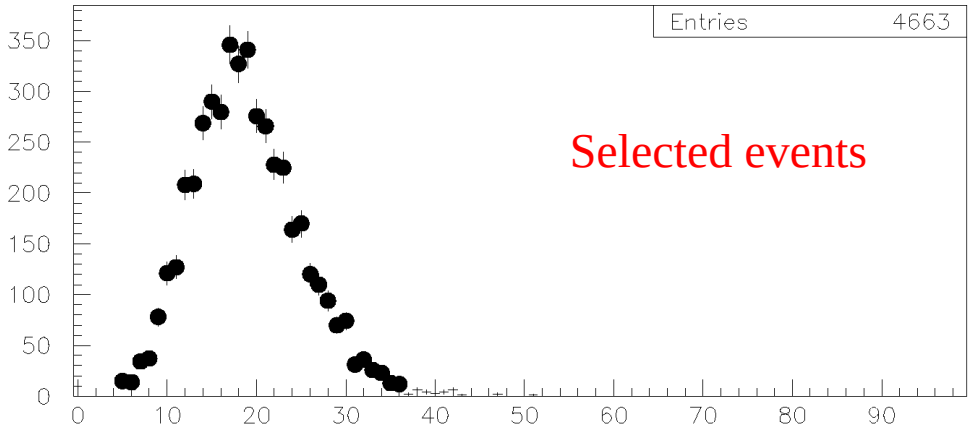
- Fortran
  - Read pre-selected hadronic events from EDIRs (class 16 bit)
  - Unpack the relevant BOS banks in memory
  - Fill the QVEC and QVRT data structure
  - Run a simple FORTRAN event selection program
    - QCD events pre-selection
  - Fill some HBOOK histograms
- C++
  - Loop over the events in the OBJY/DB asking for the class 16 bit
  - Read in memory the relevant classes from Objy
  - Unpack the corresponding BOS banks (FORTRAN calls)
  - Fill the QVEC and QVRT data structure (FORTRAN calls)
  - Run a simple C^{++} event selection program
    - QCD events pre-selection
  - Fill some HBOOK histograms (FORTRAN calls)

Event selection

QCD event selection

- Class 16 events
- Good Tracks:
  - N_{tpc} >=4
  - P_t>0.2 Gev
  - abs(cos(θ)) < 0.9
  - d_0<2. cm
  - z_0<10. cm
- N_{sel} Track >=5
- E_{sel} Track >= 15. Gev





Ntrack/event

Preliminary performance test: results

- ALPHA++ does also the unpacking/filling of the BOS banks in memory
- The event analysis time is negligible
- The histogram filling time is negligible

	CPU time/ev (class 16) (sec)	CPU time/ev (all) (sec)	Init. Time (sec)
ALPHA	**15.1×10^{-3}**	**1.9×10^{-3}**	**1.48**
ALPHA++	**29×10^{-3}**	**2.6×10^{-3}**	**1.75**

The factor ~2 difference in CPU time between ALPHA++ and ALPHA is due to the **I/O** from **Objy/DB**

CPU Alpha 8400: 185 CERN units

Summary

- The setup of an OO database was rather simple and successful
  - Work done part-time by few people
- A working OO analysis program has been developed and some preliminary performance tests have been done
- We have not yet tested (in detail) LHC++ analysis tools such as *IRIS EXPLORER and HTL (or HistOOgrams)*
  - for simplicity we are still using wrapped fortran calls to HBOOK
- next release of the analysis program:
  - use STL (it seems to work on DEC now...)
  - try HTL and pawHTL