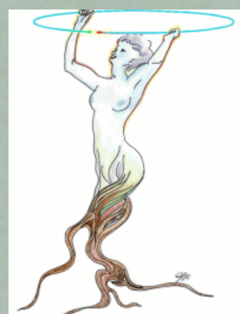
Proposal for ROOT Math Libraries

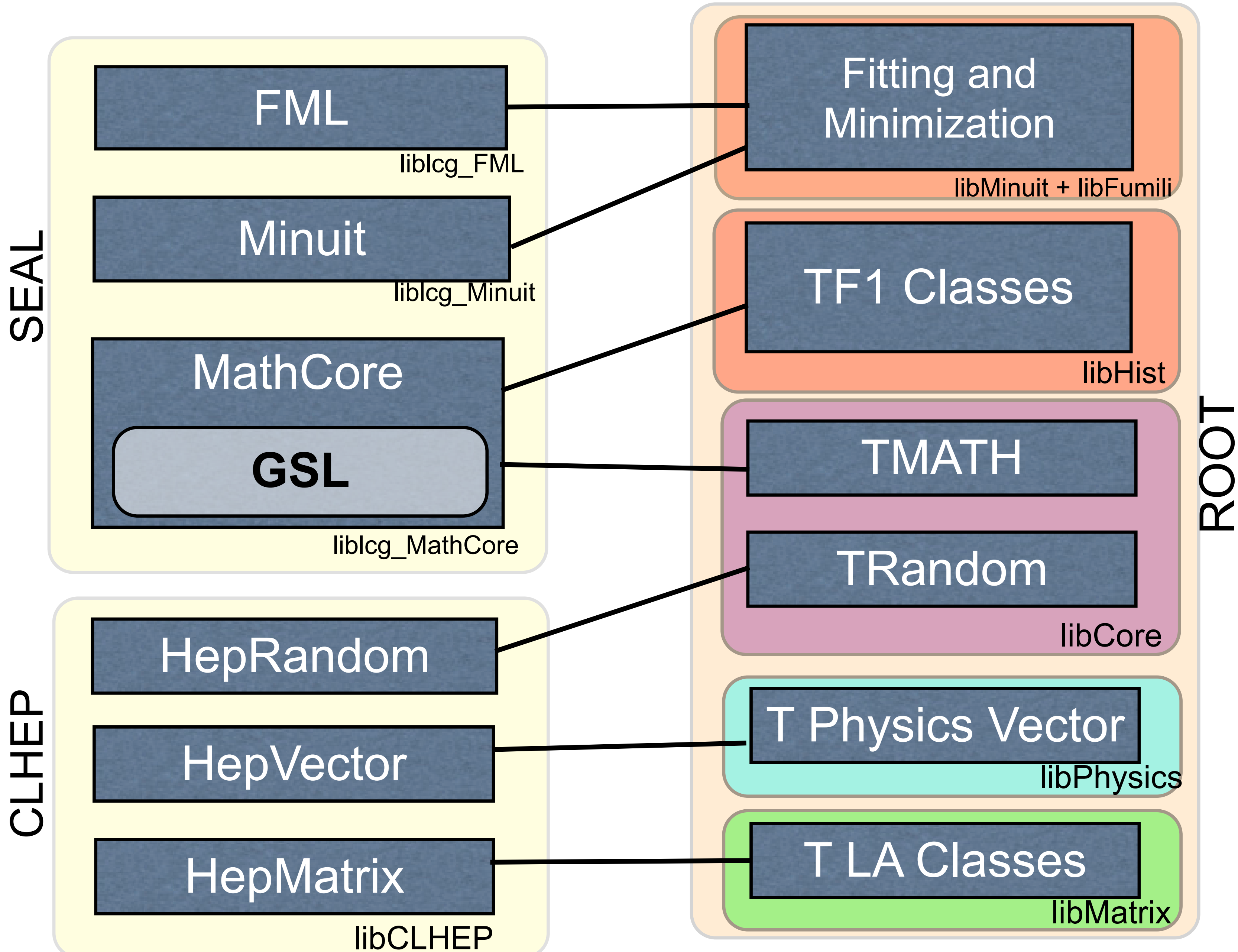


- MathLib work package from ROOT SEAL merge
- new proposed structure for Math:
  - *MathCore* and *MathMore* libraries
- new Vector package for 3D and LorentzVector
- Random, Linear Algebra and Fitting
- Conclusions

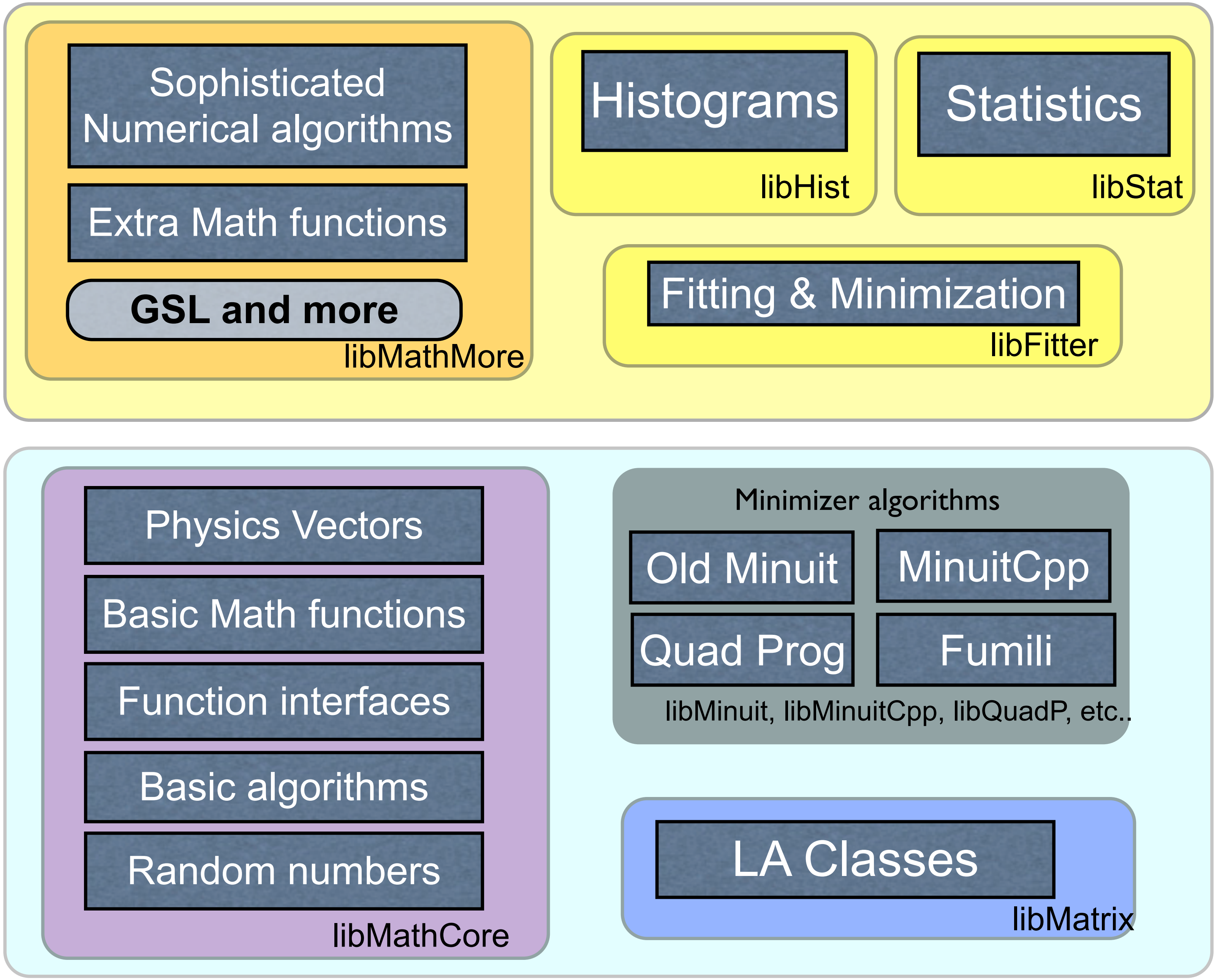
ROOT MathLib Work Package

- **Work package from ROOT-SEAL merge**
  - **people: Andras Zsenei, Anna Kreshuk, Lorenzo Moneta, Eddy Offermann**
  - **contribution also from Fermilab: Mark Fischler and Walter Brown**
- **Main responsibilities for this work package:**
  - **Basic Mathematical functions**
  - **Functions and Fitting**
  - **Random Numbers**
  - **Linear Algebra**
  - **Physics and geometry Vectors (3D and 4D)**
- **Not considered now, but still relevant :**
  - **Histograms**
  - **Statistics (confidence level)**
  - **Neural Net, multivariate analysis, etc..**

Current Math Libraries



New Math Libraries



MathCore

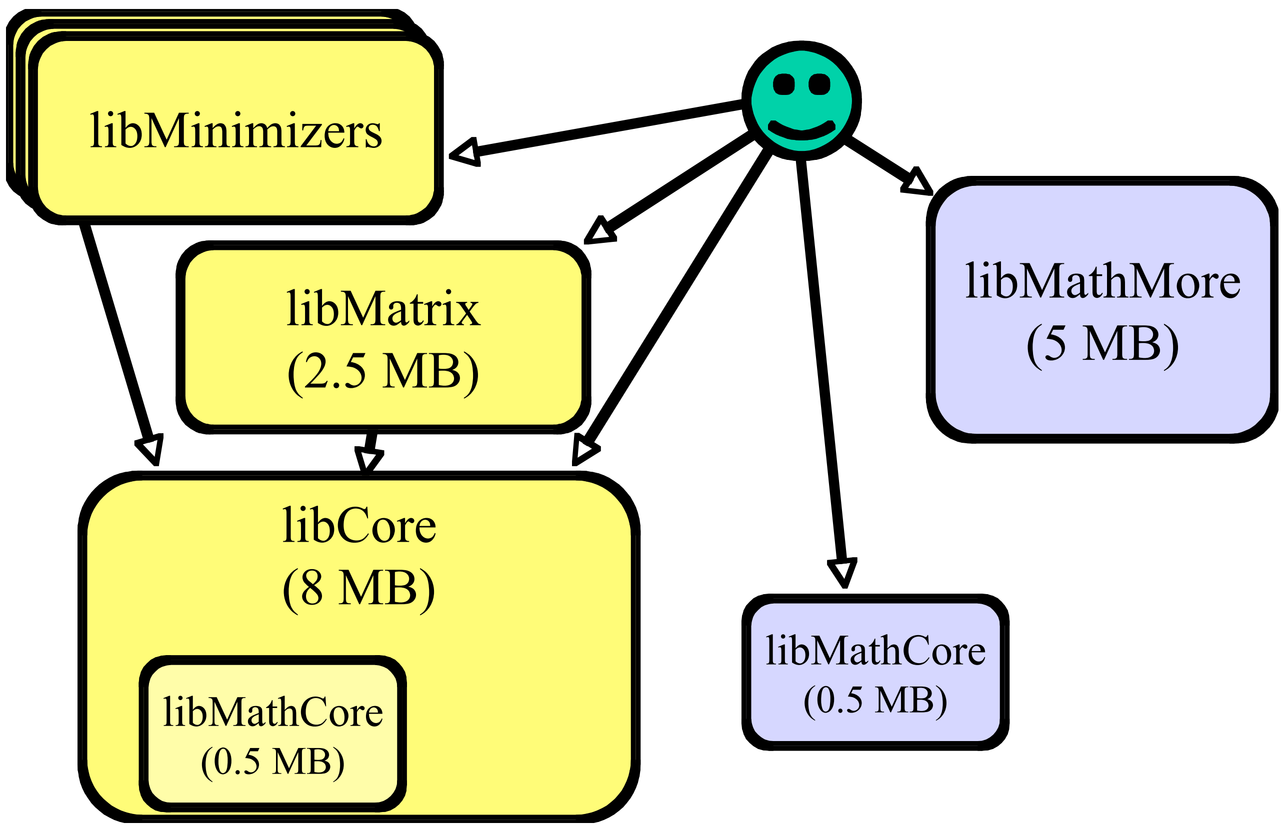
- CVS repository *mathcore* with basic functionality
- build-able as a standalone library (*libMathCore.so*)
  - no dependency on others ROOT packages or external libraries
- in ROOT is inside *libCore* for convenience
- content of *MathCore:*
  - Basic and common used mathematical functions
  - Random numbers
  - Basic numerical algorithms
  - 3D and LorentzVectors
- will not use algorithms from GSL
  - for ROOT we need to be distributed with free license
  - and the GSL is based on the GPL (restricted) license

MathMore

- will include more mathematical functions
  - less used special functions (i.e. Bessel)
- additional and more sophisticated algorithms
  - will use interface defined in MathCore
  - we will start putting there the C++ wrapper to GSL, which are now in SEAL
    - see http://seal.web.cern.ch/seal/MathLibs/MathCore/html/index.html
    - use GSL and build library together
    - will include tar file with the needed GSL functions in CVS
      - use similar procedure to existing one in ROOT (freetype, libAfterImage)
    - GSL interfaces are not exposed to the users
- repository for needed and useful extra Math functionality
  - could include other useful math libraries

Proposed new structure

- **Core Mathematical Library: *MathCore***
- **An extended library: *MathMore***



Special Functions

- most common and basic functions in *MathCore*
  - gamma functions
    - *tgamma* with log (*lgamma*) and incomplete Gamma
  - Error functions (*erf* and *erfc*)
- other less used functions will be in *MathMore*
  - use same namespace, it will be transparent for the user
- Implement the functions using proposed interface to the C++ standard
  - as it is currently done in SEAL

```
namespace ROOT {
 namespace Math {
    double cyl_bessel_i (double nu, double x);
   ......
  }
 }
```

  - use GSL for the functions present in *MathMore*

Statistical Functions

- Probability functions used in statistics
  - Many of these functions are computed using the gamma and error functions
    - those can be in *MathCore*
- We will have a consistent set of:
  - Probability distributions (pdf)
    - Gaussian, BreitWigner, Gamma, Chi2, Landau,...
  - Cumulative distributions (cdf)
    - lower and upper integrals of each pdf we provide
  - Inverse of each cdf
- Provide also functionality to generate randoms according to these pdf's
- Have also pdf C++ classes to be used for fitting

Numerical Algorithms

- some basic numerical algorithms for
  - adaptive integration, differentiation, interpolation, root finders, simple minimization (1D)
  - define interface for these algorithms and have implementations in *MathCore* or/and *MathMore*
    - start defining interfaces and API for the algorithms
    - import SEAL implementations based on GSL in *MathMore*
    - move what is in ROOT (from TF1) in *MathCore*
      - adapt TF1 to use new classes
- have more sophisticated and less used algorithms in *MathMore*
  - MonteCarlo integration, Differential Equations, FFT

Physics and Geometry Vectors

- Classes for 3D Vectors and LorentzVectors with their operations and transformations (rotations and boosts)
  - specialized vector for geometry and kinematics and not generic Linear Algebra Vectors
- Merge functionality from ROOT Physics classes and CLHEP Vector and Geometry packages
- A new prototype with API available since one month
  - work in collaboration with Fermilab computing group (Mark Fischler and Walter Brown)
    - contribute in reviewing the code, provide some of the implementations and the tests
- Developments done in contact with the LHC experiments
  - re-use some ideas from CMS Common Vector package
  - had useful discussions with some representative from the 4 LHC experiments

New Vector Classes

- Have minimal interfaces (and possibly stable)
  - minimal number of methods and try to avoid duplications
    - no x() and getX() like in CLHEP
    - no single setter methods (setX() or setPhi())
  - Separate extra functionality in global functions in a namespace
    - *deltaR(v1,v2) , invariantMass(q1,q2)*
    - template functions which can work with any Vector type with the pre-requisite:
      - implements a well defined set of coordinate accessors:
        - x(), y(), z(), r(), phi(), theta(), eta(), etc....
      - with the current interface works with CLHEP vectors

New Vector classes properties

- New classes template on the scalar type
  - Vector based on single precision (float) to decrease memory usage and for persistency
- Generic Coordinate type
  - describe the Coordinates concept as a type
  - have Vectors based on the coordinate system type:
    - can have Vectors represented by cartesian (x,y,z), polar (r, theta,phi) or cylindrical coordinates (rho, eta, phi)
    - express this as a template parameter on the Vector
      - *PositionVector3D<double, Cartesian3D>*
      - *PositionVector3D<double, Polar3D>*
    - allow conversions and operations between mixed vectors
  - can improve performances in some use cases
  - some representation can be optimal for persistency

New Vector classes properties (2)

- Points and Vector distinction
  - have in the geometry case (3D) different classes for Points and Vectors:
    - PositionVector
      - rotate and translate
      - cannot be added and their difference results in a DisplacementVector
    - DisplacementVector
      - only rotate
      - have cross and dot multiplications
- This distinction is present in the CLHEP Geometry
  - but using a common base class
- No need to have this separation for LorentzVectors
  - used in kinematics (DisplacementVectors in 4D)

Rotations and Transformations

- 3D Rotations
  - describe them according to different representations:
    - 3x3 orthogonal matrix representation (9 numbers)
    - 3 Euler angles
    - Direction Axis (Vector) + angle
    - could add also quaternion (4 numbers)
  - generic rotation is template on the representation type
- LorentzRotations (Boost + 3D Rotations)
  - described by a 4x4 matrix
  - symmetric 4x4 in the case of pure Boosts
- 3D Transformations (3D Rotations + Translation)
  - described as a 3D Rotation + 3D Vector
  - have interface to look like a 4x4 matrix (as CLHEP)

Examples of usage: LorentzVector

- we have Lorentz Vectors based on
  - Cartesian4D (x,y,z,t) or (px,py,pz,E)
  - CylindricalEta4D (pt, eta, phi, E)
  - EEtaPhiMSystem (E, eta, phi, M)
  - and we could have more type of system (flexible to extend)
    - one based on px,py,pz, M to avoid some numerical problems (electrons at LHC)

- template class on scalar type and Coordinate type

- use typedef's to hide template complexity to the users
  - `typedef BasicLorentzVector<double, Cartesian4D> LorentzVector;`
  - `typedef BasicLorentzVector<double, CylindricalEta4D> LorentzVectorPtEtaPhiE;`

LorentzVector Example

Constructors

```
LorentzVector              v0;               // create an empty vector (x=y=z=t=0)
LorentzVector              v1(1,2,3,4);      // create a vector (x=1, y=2, z=3, t=4)
LorentzVectorPtEtaPhiE     v2(1,2,M_PI,5);   // create a vector (pt=1,eta=2,phi=PI,E=5)


LorentzVectorPtEtaPhiE     v3(v1);           // create from a Cartesian4D LV
CLHEP::HepLorentzVector    q(1,2,3,4);
LorentzVector              v3(q)             // create from a CLHEP LV
```

Accessors

```
double x     = v1.x() = v1.px();             // have both x() and px()
double t     = v1.t() = v1.e();              // have both t() and e()
double eta   = v1.eta();
XYZVector w  = v1.vec();                     // return vector with spatial components
```

Operations

```
v1 += v2;  v1 -= v2;                         // additions and subtructions
v3 = v1 + v2;
v3 = v1 - v2;
double a; v1 *= a; v1 /= a;                  // multipl. and divisions with a scalar
double p = v1.dot(v2);                       // prefer dot (less ambiguous)
```

Connection to Linear Algebra

- Some experiments require easy connection/ conversion
  - between 3D/4D Vectors and Linear Algebra Vectors
  - between 3D/4D Rotations and Linear Algebra matrices
- Avoid direct dependency on any LA package
- Proposed solution:
  - construct and assignment using template member functions for LA objects implementing the operator[]
  - store vector and rotation data in a C array :
    - construct/assign from C array pointers (double *)
    - return a C array pointer
    - able to use Vector/Rotation content in a LA package
      - ROOT Linear Algebra allows to create matrices by copying the data or by using the data

Linear Algebra Example

From a Linear Algebra Vector

```
TVectorD       a(3);            // ROOT Linear Algebra Vector
XYZVector      v1(a,0);         // construct vector from x=a[0], y=a[1], z=a[2]

double *dd   = a.GetMatrixArray();
XYZPoint       p1(dd);          // construct point from x=a[0], y=a[1], z=a[2]

TVectorD       b(N);        // ROOT Linear Algebra Vector containing many vectors
XYZVector      v2(b, INDEX);  // construct vector from x=b[INDEX], y=b[INDEX+1],...


HepVector      c(4);        // CLHEP Linear algebra vector
LorentzVector  q(c,0);      // construct using px=c[0], py=c[1], pz=c[2], E=c[4]
```

To a Linear Algebra Vector

```
XYZVector      v(x,y,z);
double * pp = v.coordinates().data();

TVectorD       t(3,pp);      // create a new Linear Algebra vector copying the data


TVectorD       w;
w.Use(3,p);                 // fill an existing Vector using the data (no copying)
```

Note that ROOT Linear Algebra Object can use external data storage

Vector Performance Tests

- comparison between CLHEP, ROOT and new classes for LorentzVector's
  - better results than ROOT because a 4D Vector is not now based on a 3D object
    - factor of 2 improvements in additions of LorentzVectors
  - performance improvements if used optimal coordinate system when needed
    - Example: DeltaR for a large set of Vectors
      - some order of magnitude in speed improvements
- I/O tests
  - some performance obtained with TLorentzVector if TObject stream is ignored for TLorentzVector
    - otherwise performance improvement ~ 20%

Current Status

- **Current proposed version available for feedback:**
  - **http://seal.web.cern.ch/seal/MathLibs/GenVector/0-1-0/html/index.html**

Main Page | Namespace List | Class List | Directories | File List | Namespace Members | Class Members | File Members | Related Pages

Proposal for a new Vector Package

0_1_0

Generic Vectors for 2, 3 and 4 dimensions

This is a proposal for a new vector package, **GenVector**, describing vectors and their operations in 2, 3, and 4 dimensions. The 4 dimensional space is used for describing relativistic particles. These vectors are different from generic vectors of the Linear Algebra package which describe N-dimensional vectors. The functionality of this package is currently provided by the CLHEP Vector And Geometry packages and the ROOT Physics Vector classes (Tvector2, TVector3 and TLorentzVector). It is also re-uses concepts and ideas from the CMS Common Vector package. The main characteristics of this package are :

- **Minimal interface**

  We define a minimal interface trying to avoid duplications in contrast to what is currently provided by the Vector package of CLHEP.

Vector Feedback

- asked for feedback to LHC experiments and also ROOT users
- Received very useful comments
  - requested connection to Linear Algebra
  - representation based on px,py,pz, m instead of E to avoid negative masses when E >> m
  - have well defined set of accessors like x(),y(),z()
    - like that to have generic helper functions which can then be used by the experiment classes
  - want some compatibility (inter-operability) with CLHEP for a smooth migration
  - have also concept of coordinate errors for providing error propagation in the operations
  - use quaternion to represent rotations

Some Open questions

- Error handling (related to all Math libraries)
  - what to do with Nan and infinities
    - Proposed solution:
      - throw exception
      - In the Vectors have a a simple exception class deriving from std::runtime_error
    - Returning a Nan to the user could be OK for a simple user application but NOT for a reconstruction job
- Naming convention for member functions
  - need to decide on names like x() or X() ?
  - advantage of having same signature as CLHEP will provide some inter-operability
  - for ROOT users the TLorentzVector and TVector3 classes will not disappear
    - will be implemented as proxy to new classes

Future Work for Vectors

- Solve open issues and finalize implementations
  - take into account the feedback received
- Move in the ROOT CVS directory
  - Module.mk for building already exist
  - need to integrate also the tests
    - preferable to have them in same location as the code in a tests sub-directory
  - solve some remaining problems in generating CINT dictionary for some template member functions
- Should be ready for first ROOT 5 release at the end of the month

Random Numbers

- Merge CLHEP and ROOT Random number classes
- CLHEP and ROOT have a different design
  - ROOT has a common base class for all the engines and defining also the distributions
    - easier to use (no need to create separate classes)
  - CLHEP separates distribution classes from engine classes
    - easier to extend if user wants to add new distributions
    - distributions classes can have a state
- New design has been proposed to the C++ standard
  - Fermilab people are implementing a first version of this new library
  - need to evaluate it and try to re-implement the TRandom classes using the new library

New C++ Random Numbers

- design based on generic engine classes and distributions
- define engine using template parameters:

```
typedef mersenne_twister<double,32,624,......> mt19937
typedef subtruct_with_carry_01<double,48,10,24>
ranlux64_base_01
```

- Distribution classes template on value type:

```
uniform_real<T>, exponential_distribution<T>,
normal_distribution<T>
```

- class `variate_generator<Engine,Distribution>` to generate the random numbers:

```
mt19937 engine(seed);
uniform_real<double> dist(xMin,xMax);
variate_generator<mt19937, uniform_real<double> > r(engine, dist);
// generate random number xMin < x < xMax
double x = r();
```

- rather complex for end-users (should not be exposed)

Linear Algebra

- Proposal is to base on ROOT Linear Algebra
- Functionality in ROOT Linear Algebra
  - decompositions for solving LA systems
  - support for sparse matrices
  - support for external data storage
  - pre-allocation on the stack up to 6x6 matrix and optimized inversion
- Consider to move in the long term to template classes for double/float matrices
- Continue detailed evaluation with new LA packages
  - decide if need later a standalone library optimized for small matrices
  - follow evolution of new GLAS (Boost) project

Fitting and Minimization

- Import new C++ Minuit from SEAL in ROOT
  - contains all minimizer (Migrad, Simplex) and in addition the Fumili algorithm
  - have already a class which implements TVirtualFitter
    - complete with support for Fumili
    - it could be migrated soon
  - evaluate/merge with minimizer package from Fermilab
- Improve ROOT fitting and minimization interfaces
  - current interfaces are too much adapted to old Fortran Minuit API
  - have a more generic interface to satisfy requirements from different minimizers and fitting algorithms
    - new linear and robust fitters
- work is on-going on importing the RooFit package

Conclusions

- **Have first version of *MathCore* and *MathMore* libraries**
  - **Vectors, math functions and basic algorithms for the first ROOT 5 release (end of June)**
  - **first proposal for Vector package already exists**
    - **fruitful collaboration with CLHEP editors (M.F.)**
    - **received feedback from experiments and ROOT users**
      - **any other comment or feedback is still highly desirable**
- **Later activities:**
  - **Evaluate new C++ standard Random number**
    - **decide if to use for re-implementing ROOT Random**
  - **detailed evaluation of the Linear Algebra**
  - **improve ROOT Fitting and import RooFit and new C++ Minuit**

References

- Special functions C++ proposal
  - link to C++ extension draft (includes Random proposal)
    - http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2004/n1687.pdf
- Statistical functions proposal (for Boost)
  - http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1069.pdf
- SEAL Math inventory
  - http://seal.web.cern.ch/seal/snapshot/work-packages/mathlibs/mathTable.html
- SEAL MathCore reference doc (GSL C++ wrappers)
  - http://seal.web.cern.ch/seal/MathLibs/MathCore/html
- Proposal for new Physics Vectors
  - http://seal.web.cern.ch/seal/MathLibs/GenVector/0-0-2/html/index.html