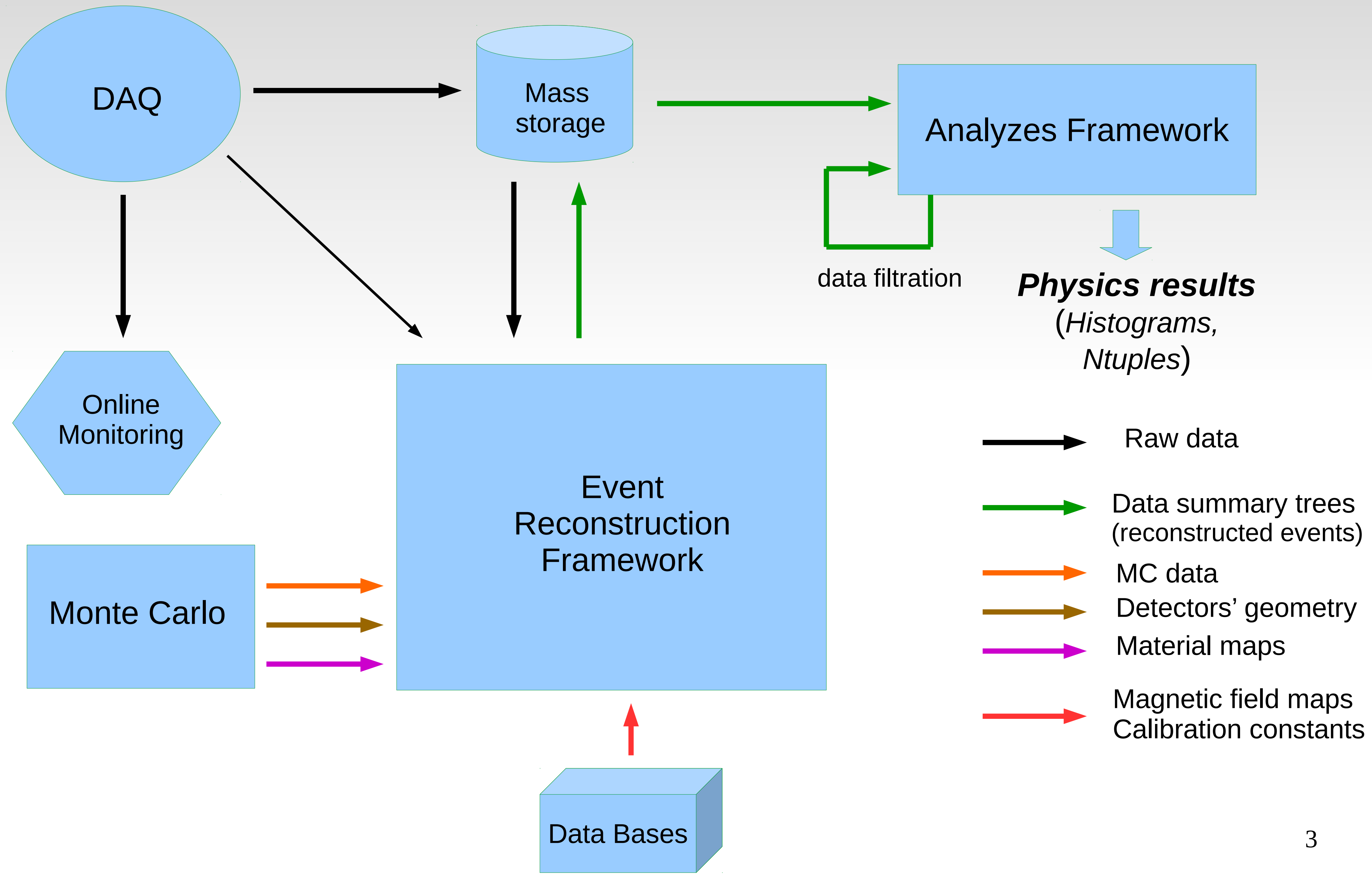
Software design of the COMPASS experiment in CERN
(solutions, which could be useful for SPD)

Overview

- General structure of the software
- Monte Carlo
- Monte Carlo → Event Reconstruction Framework interface
- Event Reconstruction Framework
    - Decoding
    - Track reconstruction
        - Kalman filter tree algorithm
    - Reconstructed Data structure
- Analysis Framework
- Conclusions

General structure



Monte Carlo

- Geant3:
  - Input:
    - Physics processes generator’s data (text file)
    - Geometry of detectors are defined by text file
  - Output:
    - Simulated tracks and vertices (Zebra binary)
    - MC hits (see below)
    - Detectors geometry description (text file)
    - Maps of materials (self-made binary format)

Monte Carlo

- Geant4:
  - Input:
    - Physics processes generator’s data (text file)
    - Geometry of detectors are defines by code.
  - Output:
    - Simulated tracks and vertices (self-made format)
    - MC hits (see below)
    - Detectors geometry description (text file)
    - Maps of materials (ROOT Geometry package)

Monte Carlo

- Better solution would be:
  - Virtual Monte-Carlo (ROOT’s TVirtualMC)
    - Common interface for different “transport code” (currently Geant3 and Geant4)
    - Interface for “user-defined” physical processes
    - Fully integrated with ROOT Geometry package (TGeom)

Monte Carlo. Interface with reconstruction software.

- Information is needed to be passed into Event Reconstruction Framework:

  1) for every detector:

     a) position, orientation, size

     b) internal structure. E.g.

        - number of wires/strips/straws/...
        - pitch/step
        - readout direction (1st wire position)
        - dead zones

  2) Materials map (to take into account multiple scattering and energy losses).

  3) Magnetic filed map

Monte Carlo. Interface with reconstruction software.

- ROOT geometry is a good solution for description of:
  - Detectors' position, orientation, size
  - Materials (detectors' materials and "passive materials")
- For magnetic field map ROOT geometry has only an interface (TVirtualMagField). i.e. implementation has to be done by user.

In COMPASS:

- special interface (table) is used to pass information on detector's geometry (including internal structure)
- ROOT geometry is used for material map

Event Reconstruction Framework (main components)

- Decoding of the Raw Data
- Detectors' response simulations in the case of MC data
- Track reconstruction
- Reconstruction of calorimeters' data
    - electromagnetic and hadronic calorimeters
- Particle identification (PID)
    - RICH, muon ID, hadrons ID etc.
- Vertex finding
- Event display (for algorithms debugging)
- DST (data summary trees) creation and storage

Decoding

- Decoding Library takes DAQ format data (Raw Data) as an input and produce so called DAQdigits objects as an output. (This is the 1-st step of decoding)
- DAQdigit contains 3 main data-members:
  - Detectors' unique ID
  - Sensitive element number (strip/pad/straw/wire/fiber/calorimeter cell ..) of this detector where information had been produced
  - Associated information (depends on front-end electronics type). E.g.
    - amplitudes for ADCs
    - times for TDCs

Decoding

- Correspondence between electronic addresses of information sources and detectors’ sensitive elements are defined in external, human readable files (so called “mapping files”):
  - Mapping files are XML format based, where connection of electronics to wires/strips/pads/ calorimeter cells .. etc. are described.
  - Mapping files may have time-dependent sections (“validity periods”) what allows to support possible different commutation of electronics and detectors at different periods of data taking time.

Decoding

- The same Decoding Library (and the same mapping files) is used in Online Monitoring software and on the first step of decoding (creation of DAQdigits) in Event Reconstruction Framework.
  - Advantage: on the stage of commissioning and running of experiment, detectors responsible persons maintain and update mapping files of their detectors and (rarely) code of Decoding Library to be able to used Online Monitoring

  → offline software get it “for free”

Decoding

- 2-d step of decoding is creation of Digits out of DAQdigits:
  - Digit contains also 3 data-members:
    - Detectors' unique ID
    - Sensitive element number (strip/pad/straw/wire/fiber/calorimeter cell/..)
    - Measurement. E.g.:
      - Amplitude with subtracted pedestals
      - Time corrected on T0
      - Energy deposited in calorimeter cells, recalculated from amplitudes
- On this step various types of available calibrations are used

Decoding

- 3-d step of decoding is creation of Clusters (sometime called also Hits) out of Digits.
  - Typically Cluster is grouped Digits e.g. group of adjacent wires / pixels /strips ... Or group of calorimeter cells with non-zero energy deposition.
  - Content of Clusters:
    - Measured coordinate with errors
    - Possible associated information:
      - Amplitude or time for tracking detectors
      - Energy for calorimeters
- Clusters created on this step are physics measurements which are inputs for event reconstruction algorithms

Decoding
in the case of Monte Carlo

- Monte Carlo do not simulate Raw Data format but produce so called MChits.
- Contents of MChits:
  - X,Y,Z of simulated particle on the entry to sensitive volume (i.e. detector)
  - Momentum Px,Py,Pz of simulated particle on the entry to sensitive volume.
  - X,Y,Z of simulated particle on the exit from sensitive volume (if particle had traversed the detector)
  - Momentum Px,Py,Pz of simulated particle on the exit from sensitive volume (if particle had traversed the detector) .

Decoding in the case of Monte Carlo

- Simulation of detector response was done in the Event Reconstruction Framework.
- It is a software (usually functions of detectors' classes), different for different type of detectors.
  - Typically it is developed and maintained by detectors' experts.
- Those functions produce Digits out of MChits. i.e. simulate detector response if particle traverse detector of this type.

After this step (after creation of Digits) event reconstruction software is identical for Real data and for MC data (almost :-).

Track reconstruction

- Important point is choice of track parametrization. Requirements are:
  - Minimal number of parameters
  - Minimal correlation between parameters
  - Preferably Gaussian distribution of parameters errors.

- We use following parametrization:

  X = (x, y, dx/dz, dy/dz, q/P) at some surface (detector's surface or virtual one). x,y, are coordinates on the surface, z is normal to the surface, dx/dz, dy/dz are slopes, q is charge and P is momentum.

Track reconstruction

- 3 main steps of tracking:
    - Track pieces finding in different detector groups (“pattern recognition”)
    - Joining track pieces to form “long” tracks (“bridging”)
    - Track fit.

Track reconstruction

- Pattern recognition:
  - Classical combinatorics of tracking detectors' clusters in "projections", i.e. in detectors measuring coordinates at the same angle (2D track pieces)
  - Making track pieces in space (3D track pieces) by combining 2D track pieces

- Disadvantages of the method: it assumes certain model of the track pieces (straight line in the case of COMPASS)
- Better solution: "Kalman filter tree" (see below)

Track reconstruction

- Bridging:
  - Combinatorial connection of track pieces in different detector groups to form track candidate
  - Attempt to fit track candidate (by Kalman fit) to find best combination of track pieces in terms of number of used clusters and chi2.

Track reconstruction

- Track fit: Kalman fit technique was used.
- Kalman fit is iterative procedure:
- On every step track parameters vector **X** with it's **covariance matrix** are ...

  (1) extrapolated to the surface of tracking detector. Extrapolation procedure uses magnetic field map and materials map (multiple scattering)

  (2) updated by measurement **M**, where **M** is cluster coordinates with **covariance matrix (**if any).
- At the end of iterations we have the best track parameters estimator with it's covariance matrix

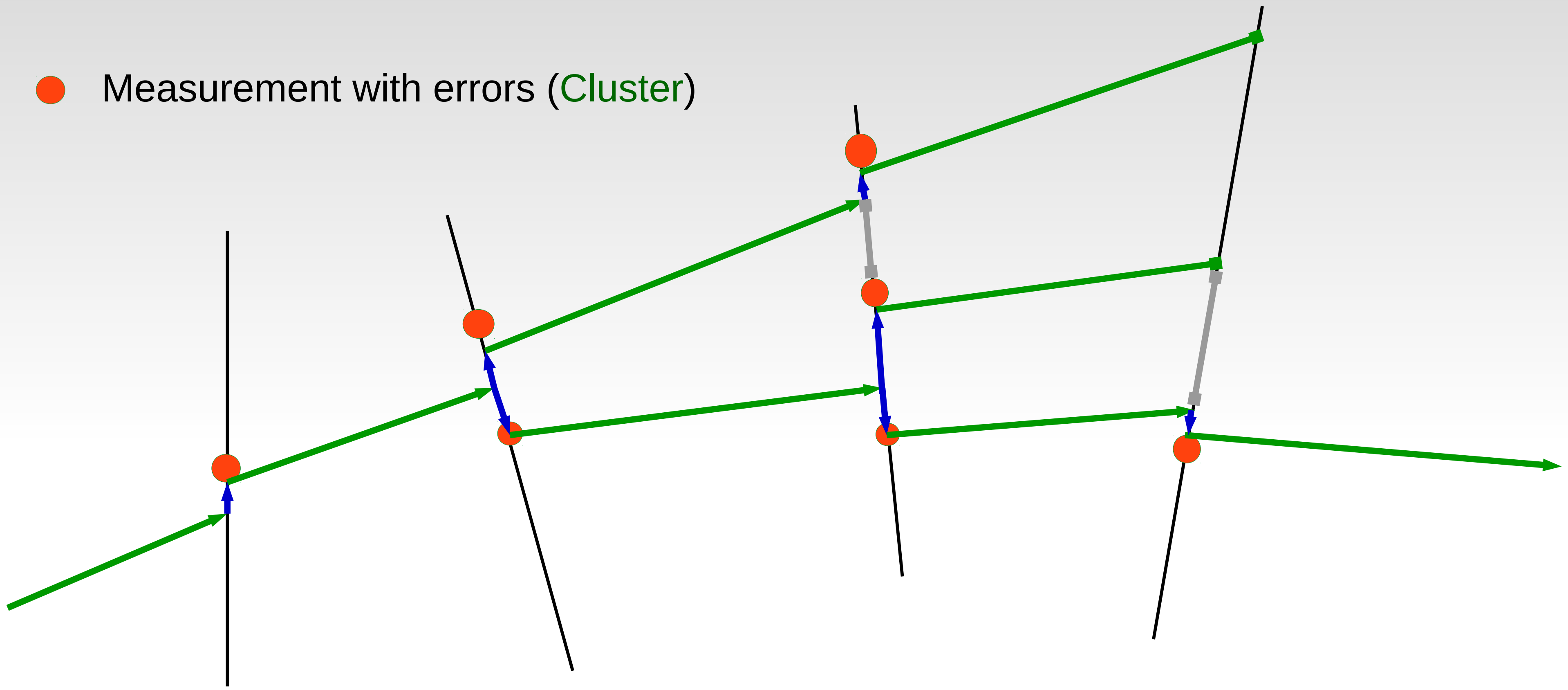
Track reconstruction

- Advantages of Kalman fit technique:
  - Manipulation with matrices of maximal size NxN where N is length of track parameters vector (in our case it is 5) independent of number of used measurements (number of clusters) on the track.
  - Before "update" step, contribution of measurement **M** to final tracks' Chi2 could be calculated $\rightarrow$ we can decide to include this measurements to the track or not (i.e. to do filtering).
    - This allows to build pattern recognition algorithm based on Kalman filter (Kalman filter tree).

Kalman filter tree

(simultaneous pattern recognition and fit)

● Measurement with errors (Cluster)



⟶	Extrapolate **X** and it's **covariance matrix** to detector's surface
⟶	Update **X** by measurement **M**
▪—▪	Check Chi2 increment. No "update" if it's too big.
—▪	Stop track following ("dead branch of the tree")

Track reconstruction

- Usually Kalman fit has 2 passes.
  - Forward direction (to get best estimator of track parameters at the end)
  - Backward direction (to get best estimator of track parameters at interaction vertex)
- It is possible to get best estimator of track parameters at any point of trajectory (e.g. on detector’s surface) using so called “Kalman smoothing” procedure

Data summary tree (DST) structure

- DST structure is based on ROOT package trees and contains:
  - Reconstructed events tree (multiple event-type objects with possibility of random access)
  - Event independent information (tree with 1 object):
    - Detectors geometry descriptions
    - Magnetic field maps
    - Materials map …
    - ...etc. I.e. everything which is not changed from event to event but could be needed for analysis

Data summary tree (DST) structure

- Reconstructed events contain:
  - Reconstructed tracks
  - Tracks refitted in the vertex (if associated to some vertex)
  - Vertices
  - Calorimeters' clusters
  - Clusters of tracking detectors (optional)
  - DAQdigits (optional)

Data summary tree (DST) structure

- Reconstructed events may also contain (in the case if input is Monte Carlo):
  - MC tracks
  - MC vertices
  - MC hits
  - Original MC generator information

Data summary tree (DST) structure

- It is data hierarchy, organized “by containment”
- I.e. “reconstructed event” have containers (STL vectors) of lower level objects (e.g. “reconstructed tracks”, “reconstructed vertices”, MCtracks etc.) each of them may have containers with other objects (e.g. MCtrack may have vector of MChits) etc.
- So, structure of the code represents naturally logical structure of HEP event and MC data.
- Flexible enough to be changed during experiment lifetime

Analysis Framework ...

- provides simple access to reconstructed events (for physicists who are not experts in C++ and ROOT)
- is an environment for physics analysis code development
- is a tool for DST ...
  - processing of events (may add info to events)
  - filtering of events' sub-samples (reduction in number of events)
  - discarding of information not needed (reduction in size of events)
- also provides DST output data stream module in the Event Reconstruction Framework.

Analysis Framework ...

- was designed to have minimal dependencies on other software (only ROOT is needed)
- provides encapsulated place for user’s analysis code and executable build tools for different platforms
- allows to change DST content without loss of backward compatibility (due to ROOT’s “schema evolution” mechanism)

Conclusions

- Some of software design solutions successfully used in COMPASS during many years could be taken into consideration in SPD at software design and development stage.

Decoding

- Fragment of typical mapping file:

```
<Map>

  <ChipAPV
   options       = ""
   runs          = "255235-265099"
   year          = "2015"
   version       = "1"
   detector      = "GEM detectors"
   maintainer    = "John Smith">

<!-- Station GM01 -->

    <!-- name srcID adcID chipID chanF chanS chanN    wireF wireL wireS -->

    <!-- TGEM23 -->
        0  GM01U1__     736    1  1  0     1  128          127   0  -1
        0  GM01U1__     736    1  2  0     1  128          255 128  -1
        0  GM01U1__     736    1  3  0     1  128          383 256  -1
        0  GM01U1__     736    1  4  0     1  128          511 384  -1
        0  GM01U1__     736    1  5  0     1  128          639 512  -1
        0  GM01U1__     736    1  6  0     1  128          767 640  -1

        0  GM01V1__     736    2  6  0     1  128          127   0  -1
        0  GM01V1__     736    2  5  0     1  128          255 128  -1
        0  GM01V1__     736    2  4  0     1  128          383 256  -1
        0  GM01V1__     736    2  3  0     1  128          511 384  -1
        0  GM01V1__     736    2  2  0     1  128          639 512  -1
        0  GM01V1__     736    2  1  0     1  128          767 640  -1

    <!-- TGEM24 -->
        0  GM01X1__     736    3  6  0     1  128          0   127   1
        0  GM01X1__     736    3  5  0     1  128          128 255   1
        0  GM01X1__     736    3  4  0     1  128          256 383   1
        0  GM01X1__     736    3  3  0     1  128          384 511   1
        0  GM01X1__     736    3  2  0     1  128          512 639   1
        0  GM01X1__     736    3  1  0     1  128          640 767   1

        0  GM01Y1__     736    4  1  0     1  128          0   127   1
        0  GM01Y1__     736    4  2  0     1  128          128 255   1
        0  GM01Y1__     736    4  3  0     1  128          256 383   1
        0  GM01Y1__     736    4  4  0     1  128          384 511   1
        0  GM01Y1__     736    4  5  0     1  128          512 639   1
        0  GM01Y1__     736    4  6  0     1  128          640 767   1

  </ChipAPV>
```

Data summary tree (DST) structure

- "Reconstructed event" class (only container data-members):

```
vector<UInt_t>     vecHeader;            // header info
vector<UInt_t>     vecScaler;            // scalers info
vector<PaTrack>    vecTrack;             // Charged tracks
vector<PaVertex>   vecVertex;             // Vertices
vector<PaParticle> vecParticle;          // Particles
vector<PaCaloClus> vecCaloClus;           // Calorimeter clusters
vector<PaMCtrack>  vecMCtrk;             // MC tracks
vector<PaMCvertex> vecMCvtx;             // MC vertices
vector<PaMCgen>    vecMCgen;             // MC generator's info
vector<PaHit>      vecHit;               // Hits
vector<PaDigit>    vecVIPdigit;          // Digits important for analysis
vector<PaMChit>    vecMChit;             // MC Hits
vector<Float_t>    vecAux;                // Auxiliary information
vector<Float_t>    vecMisc;              // Auxiliary information
vector<UInt_t>     vecOnlFlt;            // Online filter information
vector<UInt_t>     vecRawBuffer;         // Event's raw buffer
vector<UInt_t>     vecDaqDecoErr;        // DAQ decoding errors
vector<Float_t>    vecCEDAR;              // CEDAR information
```

Data summary tree (DST) structure

- “Reconstructed track” class (fragment)

```
 Float_t chi2tot;           // total Chi2
Float_t meanTime;       // mean track time relative to trigger time
Float_t sigmaTime;       // mean track time error
Float_t chi2Time;          // time chi2
Float_t xx0;                  // X/X0 - radiation length of materials track passed
UInt_t expected[HIT_MAP_SIZE];     // bitmap of passed detectors
UInt_t    found[HIT_MAP_SIZE];      // bitmap of found hits
vector<PaTPar>    vecTPar;            // track parameters
vector<Float_t>   vecRich;              // RICH identification
vector<Float_t>   vecAux;               // Auxiliary information
int nmeas;                    // Number of measurements on track (as given CsTrack::_ndf)
int nhits;                     //!(-) number of hits counter from hitpattern (Hit = Cluster)
int  ind;                      //!(-) this track's index (position in vector)
int indPart;                  //!(-) reference to corresponding particle
int indMC;                    //!(-) reference to MC track (MC track # in vMCtrack or -1)
vector<Int_t>    vecHitRef; //!(-) references to hits
```

Data summary tree (DST) structure

- “Track parameters” class (fragment)

```
// persistent data members ("float" to decrease mDST volume)

Float_t Par[6];  // Track parameters: x0(fixed), x1, x2, dx1/dx0, dx2/dx0, q/P

Float_t Cov[15]; // Cov matrix lower triangle (15 elements)


// non-persistent data members (filled if Extrapolate() method was used)

double path;       //!(-) trajectory length in extrapolation

double radLenFr;   //!(-) X/X0 along path

double Eloss;      //!(-) energy loss along the path
```