EDM4hep

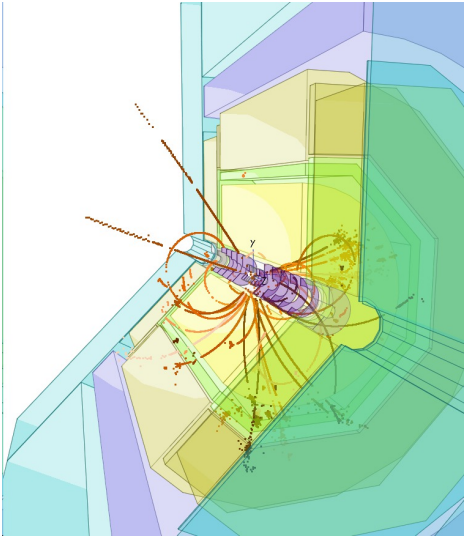
A Common Event Data Model

F.Gaede DESY

ias HEP2020, Hongkong, Jan 17,2020

Outline

- Introduction
- LCIO
- fcc-edm
- PODIO
- EDM4hep
- Status and Plans
- Conclusion



Introduction

- EDM4hep: common event data model for *future (lepton) collider studies*
- project started after the **Future Collider Software Workshop** in Bologna (June 2019)
- will provide foundation of the *Turnkey Software Stack*

Basic idea:

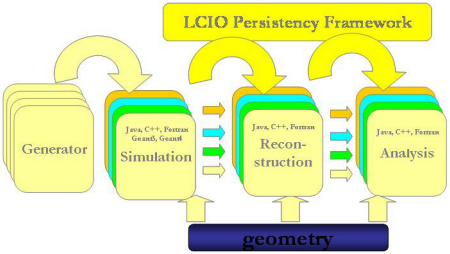
- use **PODIO** as underlying EDM tool
- define complete EDM for future collider studies based on **LCIO** and **fcc-edm**
  - use what worked best
  - extend/modify as needed
- carried out as an **HSF** project

iLCSoft: a blueprint for the Turnkey Software Stack

- in 2002 there where three LC projects in the world: Tesla, JLC and NLC
  - and about four or five different detector concepts and software frameworks
  - using C++, Java and F77 (no Python yet)

- decided to provide the basis for collaboration and common development by defining the common language, i.e. the event data model: **LCIO**

- adding **Marlin** application framework already provided the basis for iLCSoft
- last major evolution: developed and incorporated the **DD4hep** geometry toolkit



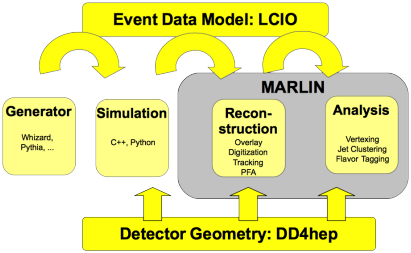
from CHEP2003 LCIO paper

iLCSoft: a blueprint for the Turnkey Software Stack

- in 2002 there where three LC projects in the world: Tesla, JLC and NLC
  - and about four or five different detector concepts and software frameworks
  - using C++, Java and F77 (no Python yet)

- decided to provide the basis for collaboration and common development by defining the common language, i.e. the event data model: **LCIO**

- adding **Marlin** application framework already provided the basis for iLCSoft
- last major evolution: developed and incorporated the **DD4hep** geometry toolkit



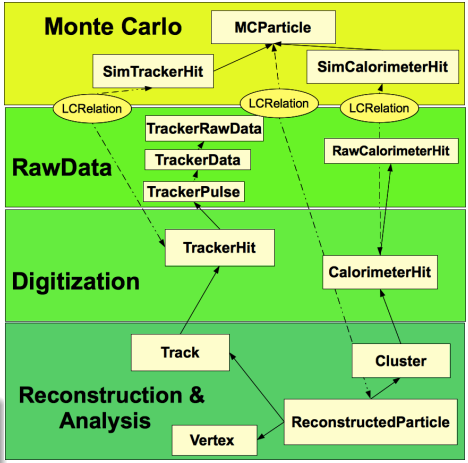
iLCsoft schema today

LCIO - Overview

- LCIO provides the common **EDM** and **persistency**
  - originally support Java, C++ and F77
  - now effectively only C++ and Python used
- Features:
  - hierarchical EDM
  - objects stored in named collections
  - implementation strictly separated from persistency
  - no direct link to MC-Truth in high level classes
- Persistency: **SIO** a simple binary I/O system with
  - compression
  - machine agnostic format
  - pointer chasing

users of LCIO

- ILD, SiD, CLICdp, CEPC, FCC-ee (partly), Calice, LC-TPC, EU-Telescope, HPS,...



http://lcio.desy.de

Development of the LCIO EDM - I

- getting an EDM is not entirely *trivial*
- for LCIO we started out with experienced people from several different experiments (SLC, LEP, HERA) in 2003
- initially only **Simulation EDM:**
  - *MCParticle, SimCalorimeterHit, SimTrackerHit*
- later extended with **Reconstruction EDM**
  - *CalorimeterHit, TrackerHit, Track, Cluster, ReconstructedParticle,...*
- all done in close communication with detector and analysis physicists
- LCIO EDM was iteratively extended and *improved* over the last **15 years**
  - $\Rightarrow$ need to foresee *schema evolution* also for **EDM4hep**

Development of the LCIO EDM - II

- the current LCIO EDM has been battle-proven for many large Monte Carlo campaigns for future e^+e^- colliders: ILC, CLIC, CEPC
  - and many test beam campaigns for Calice, LCTPC, FCAL, ...

- it has been adopted - with *independend implementations* - in **DD4hep** for the simulation and in **PandoraPFA** for the reconstruction part

- the LCIO EDM fulfills all the needs of the e^+e^- community*
- should try as much as possible to be *compatible* with LCIO in EDM4hep
  - would enable a smooth transition for the LC community to EDM4hep

*not yet used for hadron colliders - we might need some extensions here ?

additional LCIO Features

- most classes have useful convenient methods, e.g.
  - `MCParticle::isDecayedInTracker()`
- arbitrary relation between objects: 1-to-1, 1-to-N, N-to-M
  - using base type LCObject
  - in PODIO we have so far only typed relations
- relation navigator helper class (to-and from-relations) using maps
- dumping of events, collections, elements, e.g. with `operator<<()`
- encoding/decoding cellIDs (bitfields) from naming string: `"system:5,module:3,..."`

- need to see how we can implement these features in EDM4hep
- some of these are (or can be) implemented in **PODIO**

Some Lessons learned from LCIO

- the exact details of the EDM don't really matter, as long as users can store and access all information they need for their analyses
- LCIO has been **extremely stable** over the last decade or so
- the API looks a bit *old fashioned* these days
  - using bare pointers, abstract interfaces, get/set-syntax,...
  - inspired by Java/C++ coding style at the time
- *performance* was not one of the main design goals for LCIO
  - there could be room for improvement there
- transition from **LCIO** to **EDM4hep** offers great oportunity for modernizing and renewing the LC software tools chain

fcc-edm

- EDM developed *from scratch* for the FCC studies
  - from the start implemented in **PODIO**
- introduce base components: point, Lorentz-vector, hit,. . .
- create higher level objects, based on these using composition, e.g.
  - *TrackHit, CaloHit, PositionedTrackHit, PositionedCaloHit*
  - *TrackState, Track, WeightedTrack*
  - . . .
- inspired/motivated by LHC experiments

- to my knowledge this has not yet been used in large Monte Carlo campaigns with full simulation and reconstruction
- used with fast paramerized Monte Carlo and for dedicated simulation studies (occupancies, etc)

- base components:

```
fcc::LorentzVector:
fcc::Point:
fcc::BareHit:
fcc::BareCluster:
fcc::BareParticle:
fcc::BareJet:
```

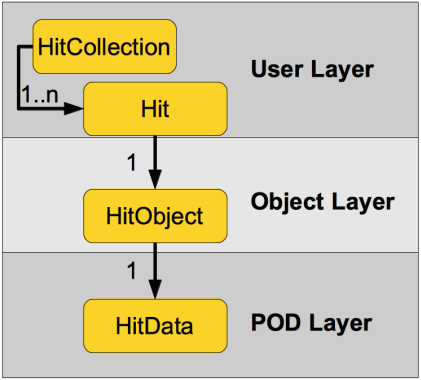
PODIO

- PODIO is an EDM toolkit developed in AIDA2020 EU-project
- based on the idea of using **PODs** for the event data objects (**P**lain-**O**ld-**D**ata object)
  - gain simplicity and performance for memory and I/O operations

- PODIO originally developed in context of the FCC study (**fcc-edm**)
- based on experience from LHC-EDMs and **LCIO**
- addressing the problem in a generic way
- allowing potential re-use by other HEP groups
- planned application to **LCIO** from the start
  - see **pLCIO** later

Implementation: the three PODIO layers

- user layer (API):
  - handles to EDM objects (e.g. **Hit**)
  - collections of EDM object handles (e.g. **HitCollection**).
- object layer
  - transient objects (e.g. **HitObject**) handling *references* to other objects and *vector members*
- POD layer
  - the actual POD data structures holding the persistent information (e.g. **HitData**)



direct access to POD also possible - if needed for performance reason

Core Features of PODIO I

- clear design of **ownership** (hard to make mistakes) in two stages:
  - objects added to event store are *owned by event store*
  - objects created stand-alone are *reference counted* and automatically garbage collected:
- allow to have *1-1*, *1-N* or *N-M* **relationships**
  - referenced objects can be accessed via iterator or directly
  - also stand-alone relations between arbitrary EDM objects

```
# LCIO MCParticle
  MCParticle:
    Description: "LCIO MC Particle"
    Author : "F.Gaede, B. Hegner"
    Members:
      - int pDG                // PDG code of the particle
      - int generatorStatus    // status as defined by the gene
      - int simulatorStatus    // status from the simulation
        #...
    OneToManyRelations:
      - MCParticle parents // The parents of this particle.
      - MCParticle daughters // The daughters this particle.
    ExtraCode:
      const_declaration:
      "bool isCreatedInSimulation() const {
          return simulatorStatus() != 0 ;
       } \n"
```

- **code generation** (C++/Python) for EDM classes from *yaml* files
  - EDM objects (data structures) are built from basic types and components
  - additional user code (member functions) can be defined in the yaml files

Core Features of PODIO II

- **Python** is treated as first class citizen in the provided library
- can use *pythonic* code for iterators etc.
- implemented with PyROOT and some special usability code in Python

```
store = EventStore(filenames)
for i, event in enumerate(store):
  hits = store.get("hits")
  for h in hits:
    print h.energy()
```

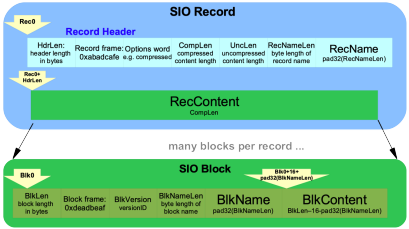
- PODIO I/O is based on **ROOT-I/O** at the moment
  - auto generated streamer code via dictionary
  - one **branch** per pod **member**
  - not really optimized for PODs
  - object references are translated into *ObjectIDs* before being stored

```
ObjectIDs = CollectionID << 32
          + CollectionIndex`
```

- had GSoC project this year to implement an **HDF5** I/O layer
  - still work in progress

Additional I/O implementations: SIO

- implemented an *experimental*, simple binary I/O for storing *array-of-structs* directly
- **SIO** - simple I/O, used in **LCIO** (>15 years)
- recently rewritten to be *thread-safe* and allow for parallel *compression* and *un-packing*
- event data stored in *records* with many *blocks* (collections)
  - compression of complete records/events
- PODIO data collections are stored as they are in memory
  - using essentially *memcpy*
  - no *byte-swapping* done currently

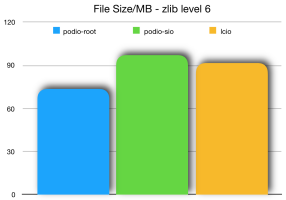


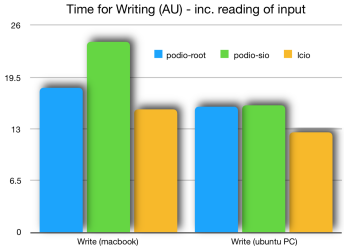
Compare ROOT to SIO I/O implementation

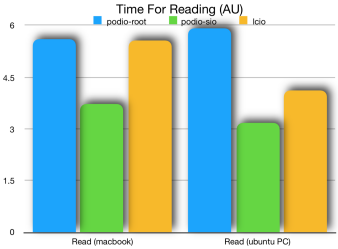
- compare the performance of these two different I/O implementations:
- ROOT (6.18)
  - *optimized columnar storage*
  - applying XDR (*byte swapping*) for machine independence
  - compression per branch
- SIO
  - store plain *array-of-structs*
  - no *byte swapping*
  - compression per record

- *"benchmark"* used here:
- convert binary *stdhep* generator ($e^+e^- \rightarrow t\bar{t}$ @ $500\mathrm{GeV}$) files to the different formats:
  - podio-ROOT, podio-SIO, LCIO
- write and read files
  - very little computation on reading
- two machines:
  - Mac-book-pro with SSD, llvm 10.01
  - x86, quad-core PC w/ Ubuntu 16.04, gcc 5.4

Benchmarking the ROOT vs SIO implementation







- ROOTI/O file size is ~**76%** of SIO file size
- ROOTI/O writing takes **75-99%** of SIO writing time [1]
- ROOTI/O reading takes **150-186%** of SIO reading time

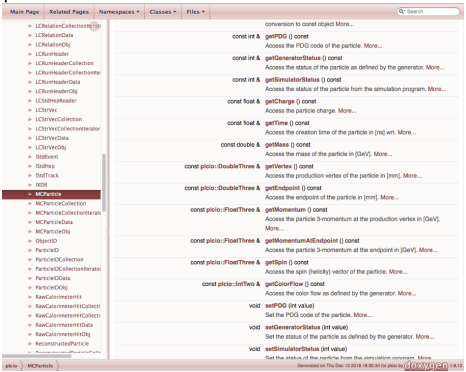
- the improvement in reading speed justifies further investigation
  - maybe even interesting to eventually develop a *POD-mode* in ROOT-I/O ?

[1] includes the time for reading stdhep input file

LCIO EDM in PODIO: pLCIO

- **pLCIO**: package that implements **complete LCIO EDM** (almost):
- original idea to be able to create classes that are almost 100% backward compatible did not fully work out
  - true for most of the actual member functions of the EDM classes
  - not true for handling of collections and collection types, creation of objects, user defined parameters, ...
- planned transition from LCIO to pLCIO would be feasible at '*reasonable cost*'

plcio



- we could use this transition to *evolve the LCIO EDM*
  - or go to **EDM4hep** directly ...

EDM4hep - First Steps

- after the Bologna workshop, we started the EDM4hep project with contributers from all four future e^+e^- colliders
- hold meetings every 3-4 weeks: https://indico.cern.ch/category/11461/
- announced at *hsf-edm4hep-wg@cern.ch*
- new contributers welcome

- Github page: https://github.com/HSF/EDM4hep

EDM4hep - Status

- project is in rather early phase
- so far have implemented the simulation model, i.e.
  - *MCParticle*, *SimTrackerHit* and *SimCalorimeterHit*
  - *Vector3f, Vector3d, Vector2i*
- for details see appendix or https://github.com/HSF/EDM4hep/blob/master/edm4hep.yaml
- include simple tests/examples for reading/writing events

- started discussion on *Track* and *TrackState*
- not so *trivial*: need to work with current LC track parameters:
  - $d_0, z_0, \omega, \phi_0, tan(\lambda)$
- at the same time support ACTS with (slightly) different parameters and potentially other tracking tools - should converge soon...

Next Steps and Plans

- continue the implementation of a *core* EDM, ie add missing classes for tracking and PFA
- implement same *real code* with this first implementation
  - e.g. write out EDM4hep in DD4hep
- in parallel improve PODIO further:
- implement/provide documentation for how to do **schema evolution**
  - absolutely need for using PODIO in a production environment with *EDM4hep*
- generalize the use of **different I/O implementations** in PODIO
  - HDF5, SIO, potentially others ? ...
  - need to iterate on and standardize the interface between *EventStore* and *Reader/Writer* implementations

Conclusion

- **EDM4hep** common event data model for *future (lepton) collider studies* will provide foundation of the *Turnkey Software Stack*
- use **PODIO** EDM toolkit and rely on experience with LCIO and fcc-edm
- implemented a simulation EDM: *MCParticle*, *SimTrackerHit* and *SimCalorimeterHit*
  - reconstruction part to follow soon

- would like to make **fast** progress
- but also need to get it **right**
- the closer we end up to the existing LCIO-EDM the easier the transition of the linear collider tools to the turnkey software stack will be

additional material

Links and Pointers

- EDM4hep:
  - https://github.com/HSF/EDM4HEP
- PODIO
  - https://github.com/aidasoft/podio
- LCIO
  - https://github.com/ilcsoft/lcio
- plcio
  - https://stash.desy.de/projects/IL/repos/plcio
- fcc-edm
  - https://github.com/hep-fcc/fcc-edm.git

basic components in EDM4hep

```
#- Vector3D with floats
edm4hep::Vector3f :
  x : float
  y : float
  z : float
  ExtraCode :
    declaration: "
    Vector3f() : x(0),y(0),z(0) {}\n
    Vector3f(float xx, float yy, float zz) : x(xx),y(yy),z(zz) {}\n
    Vector3f(const float* v) : x(v[0]),y(v[1]),z(v[2]) {}\n
    bool operator==(const Vector3f& v) const { return (x==v.x&&y==v.y&&z==v.z) ; }\n
    float operator[](unsigned i) const { return *( &x + i ) ; }\n
    "

#- Vector3D with doubles
edm4hep::Vector3d :
  x : double
  y : double
  z : double
  ExtraCode :
    declaration: "
    Vector3d() : x(0),y(0),z(0) {}\n
    Vector3d(double xx, double yy, double zz) : x(xx),y(yy),z(zz) {}\n
    Vector3d(const double* v) : x(v[0]),y(v[1]),z(v[2]) {}\n
    Vector3d(const float* v) : x(v[0]),y(v[1]),z(v[2]) {}\n
    bool operator==(const Vector3d& v) const { return (x==v.x&&y==v.y&&z==v.z) ; }\n
    double operator[](unsigned i) const { return *( &x + i ) ; }\n
    "

#- Vector2D with ints
edm4hep::Vector2i :
```

MCParticle

```
edm4hep::MCParticle:
  Description: "The Monte Carlo particle - based on the lcio::MCParticle."
  Author : "F.Gaede, DESY"
  Members:
    - int PDG                          //PDG code of the particle
    - int generatorStatus              //status of the particle as defined by the generator
    - int simulatorStatus              //status of the particle from the simulation program - use BIT constants below
    - float charge                     //particle charge
    - float time                       //creation time of the particle in [ns] wrt. the event, e.g. for preassigned decays or decays in fli
    - double mass                      //mass of the particle in [GeV]
    - edm4hep::Vector3d vertex               //production vertex of the particle in [mm].
    - edm4hep::Vector3d endpoint             //endpoint of the particle in [mm]
    - edm4hep::Vector3f momentum              //particle 3-momentum at the production vertex in [GeV]
    - edm4hep::Vector3f momentumAtEndpoint   //particle 3-momentum at the endpoint in [GeV]
    - edm4hep::Vector3f spin                  //spin (helicity) vector of the particle.
    - edm4hep::Vector2i colorFlow                 //color flow as defined by the generator
  OneToManyRelations:
    - edm4hep::MCParticle parents // The parents of this particle.
    - edm4hep::MCParticle daughters // The daughters this particle.
  ExtraCode :
    includes: "#include <math.h>\n
    "
    const_declaration: "
    // define the bit positions for the simulation flag\n
    static const int BITEndpoint = 31;\n
    static const int BITCreatedInSimulation = 30;\n
    static const int BITBackscatter = 29 ;\n
    static const int BITVertexIsNotEndpointOfParent = 28 ;  \n
    static const int BITDecayedInTracker = 27 ; \n
    static const int BITDecayedInCalorimeter = 26 ;   \n
    static const int BITLeftDetector = 25 ;     \n
```

SimTrackerHit

```
edm4hep::SimTrackerHit:
Description: "LCIO simulated tracker hit"
Author : "F.Gaede, DESY"
Members:
  - unsigned long long cellID            //ID of the sensor that created this hit
  - float EDep                  //energy deposited in the hit [GeV].
  - float time                  //proper time of the hit in the lab frame in [ns].
  - float pathLength            //path length of the particle in the sensitive material that resulted in this hit.
  - int   quality               //quality bit flag.
  - edm4hep::Vector3d position  //the hit position in [mm].
  - edm4hep::Vector3f momentum  //the 3-momentum of the particle at the hits position in [GeV]
OneToOneRelations:
  - edm4hep::MCParticle MCParticle //MCParticle that caused the hit.
ExtraCode :
  const_declaration: "
  static const int  BITOverlay = 31;\n
  static const int  BITProducedBySecondary = 30;\n
  bool isOverlay() const { return getQuality() & (1 << BITOverlay) ; }\n
  bool isProducedBySecondary() const { return getQuality() & (1 << BITProducedBySecondary) ; }\n
  "
  declaration: "
  int  set_bit(int val, int num, bool bitval){ return (val & ~(1<<num)) | (bitval << num); }\n
  void setOverlay(bool val) { setQuality( set_bit( getQuality() , BITOverlay , val ) ) ;   }\n
  void setProducedBySecondary(bool val) { setQuality( set_bit( getQuality() , BITProducedBySecondary , val ) ) ;   }\n
  "
```

SimCalorimeterHit

```
#------------- LCIO CaloHitContribution
  edm4hep::CaloHitContribution:
    Description: "Monte Carlo contribution to SimCalorimeterHit"
    Author : "F.Gaede, DESY"
    Members:
      - int   PDG                          //PDG code of the shower particle that caused this contribution.
      - float energy                       //energy in [GeV] of the this contribution
      - float time                         //time in [ns] of this contribution
      - edm4hep::Vector3f stepPosition     //position of this energy deposition (step)
    OneToOneRelations:
      - edm4hep::MCParticle particle       //primary MCParticle that caused the shower responsible for this contribution to the hit.

  #------------- LCIO SimCalorimeterHit
  edm4hep::SimCalorimeterHit:
    Description: "LCIO simulated calorimeter hit"
    Author : "F.Gaede, DESY"
    Members:
      - unsigned long long cellID       //ID of the sensor that created this hit
      - float energy                    //energy of the hit in [GeV].
      - edm4hep::Vector3f position      //position of the hit in world coordinates.
    OneToManyRelations:
      - edm4hep::CaloHitContribution contributions  //Monte Carlo step contribution - parallel to particle
```

Recent Developments in PODIO (since CHEP 2016)

- moved to use C++14 and C++17
- implemented support for I/O of *std::array*
  - needed iteration with ROOT developers
- simplified the example `EventStore`
- implemented streaming with `operator<<(...)` for all EDM classes and collections:

```
std::ostream& operator<<(std::ostream& o,
                         const ConstMCParticle& v);
std::ostream& operator<<(std::ostream& o,
                         const MCParticleCollection& v);
```

- moved GitHub repository to: https://github.com/aidasoft/podio
  - added some standard templates from iLCSoft for release notes, issues, etc.
- implemented **vector member** streaming

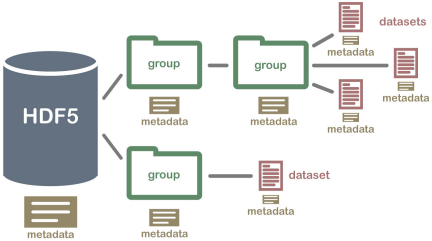
Vector Members in PODIO

- strictly speaking vector members break the *PODness* of the data classes
- treat vector members analogous to the reference vectors:
  - hold vector in *Object*-layer, start/end in *POD*-layer and iterator in *User*-layer
  - stream vector members as one large vector per collection
  - after reading keep in one large vector to minimize copying

```
plcio::LCGenericObject:
  Description: "LCIO LCGenericObject"
  Author : "F.Gaede, DESY"
  Members:
    - int isFixedSize              // all objects have a fixed size
    - std::string typeName         // type name of the user class
    - std::string dataDescription // data description string
  VectorMembers:
    - int    intVals    // integer value at given index.
    - float  floatVals  // float value at given index.
    - double doubleVals // double value at given index.
```

PODIO: Additional I/O implementations: HDF5

- had *Google Summer of Code* project this year
- **Implementation of an HDF5 I/O layer for PODIO**
  - task turned out to be a bit more involved than anticipated originally
- developed prototype code for writing some example data structures to HDF5 files
  - mapping events to groups and collections to datasets
  - unclear if this is optimal way of doing it in HDF5 !?
  - HEP data is inherently *heterogeneous*. . .



- further work needed on HDF5 implementation . . .
- potentially useful for *Machine Learning* !?