DATA ANALYSIS

EGAN Workshop, Liverpool, December 5-9, 2011

O. Stézowski

Goals

- *Get a general (clear) view of what means AGATA data analysis / data processing*
- *Be able to play with data*
- *Be able to become an actor*

Overview

Basic elements

What already exists

How to extend

Overview

Complexity

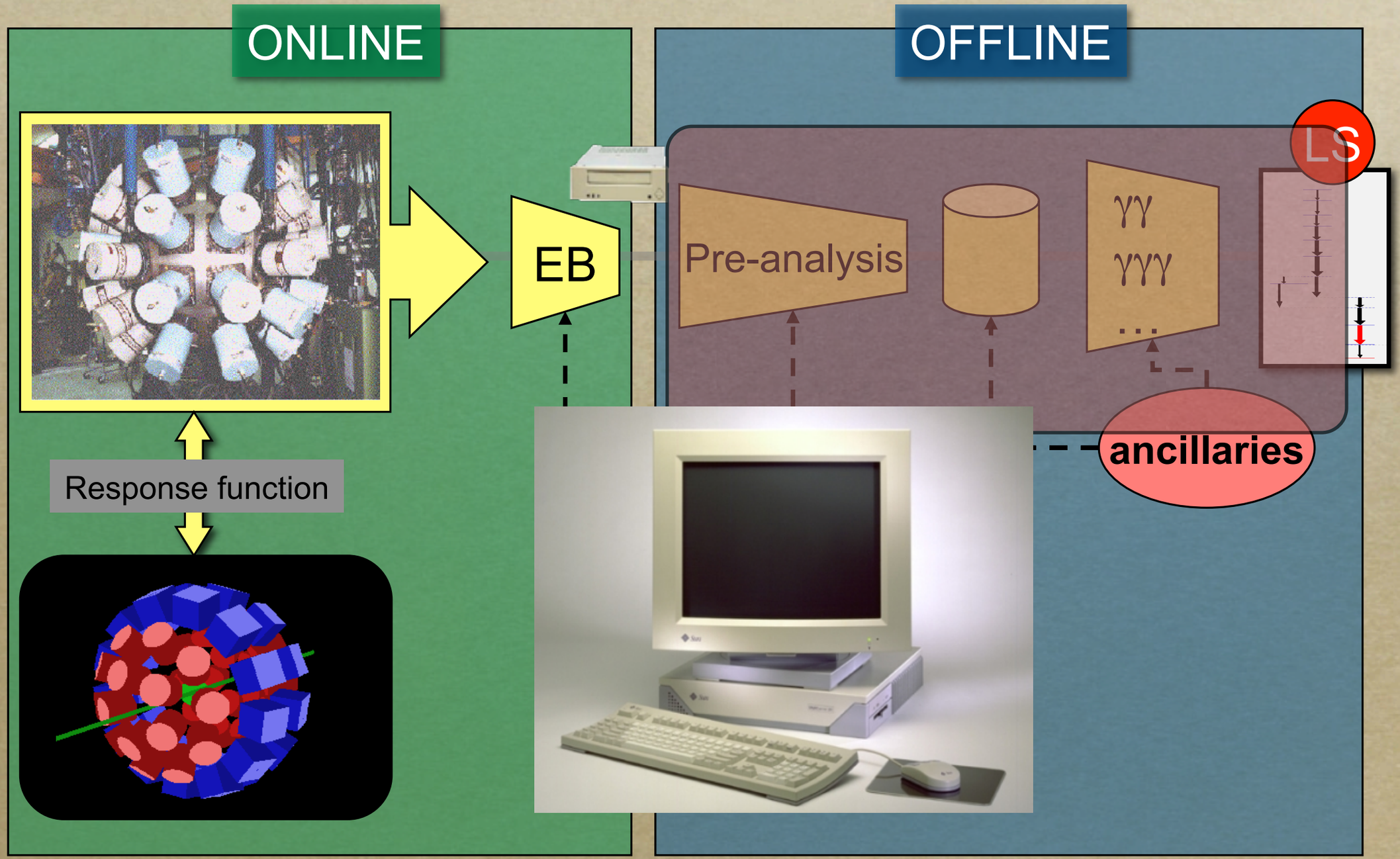
Basic elements

Narval philosophy

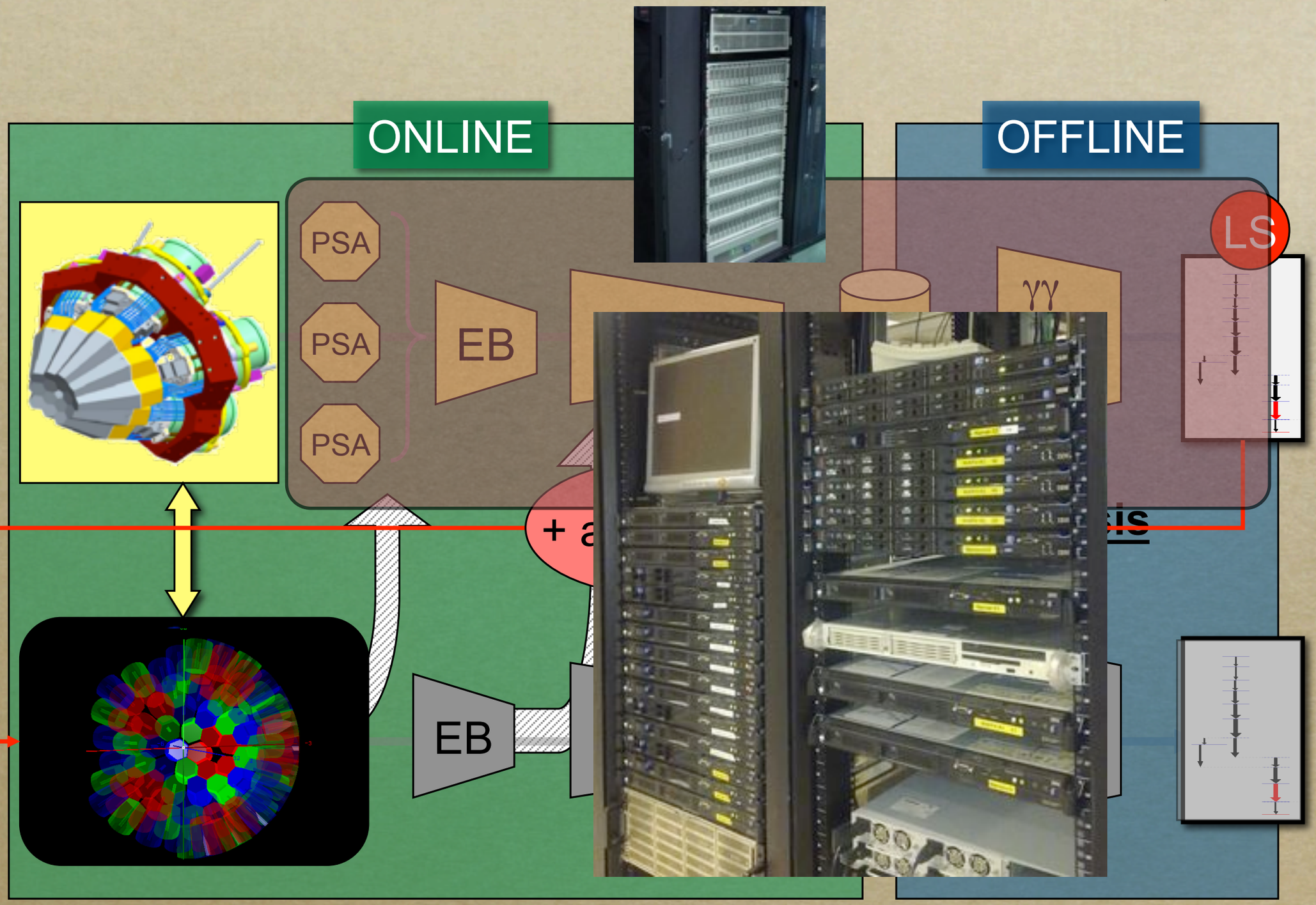
Organisation

Complex environment

Illustration : my thesis .....



Complex environment



Complex environment

➥ *many 'builders' involved in software*

(building software packages is like building a bridge)

➥ *need of coordinations*

shared files, doc (user's guide, code), tutorial, bugtracker, elog ...

[model : G4, ROOT]

➥ *different roles :*

users : use the existing bricks

developers : create new bricks

➥ *several packages to be assembled*

interfaces between different parts

dependencies (compatibilities between the different parts)

Complex environment

Narval

Femul

Root

Emulators

ISMA

ADF

Boost

How are they linked, dependent, used ???

GammaWare

ADPD

ADPL

Overview

Complexity

Basic elements

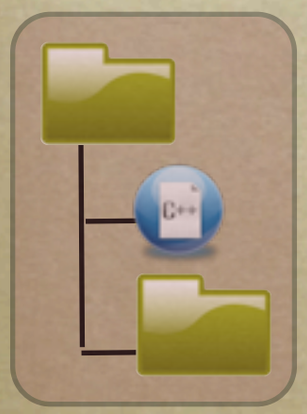
Narval philosophy

Organisation

Work with svn*

**subversion*

Repository, on a server



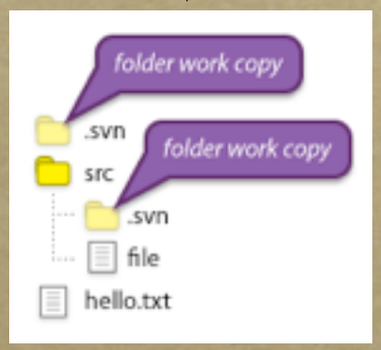
can get any version, any time !

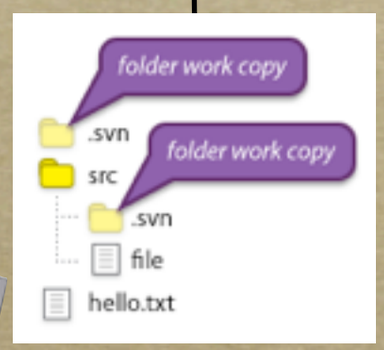
main develoment line : trunk

0

n

freezed : tag, release



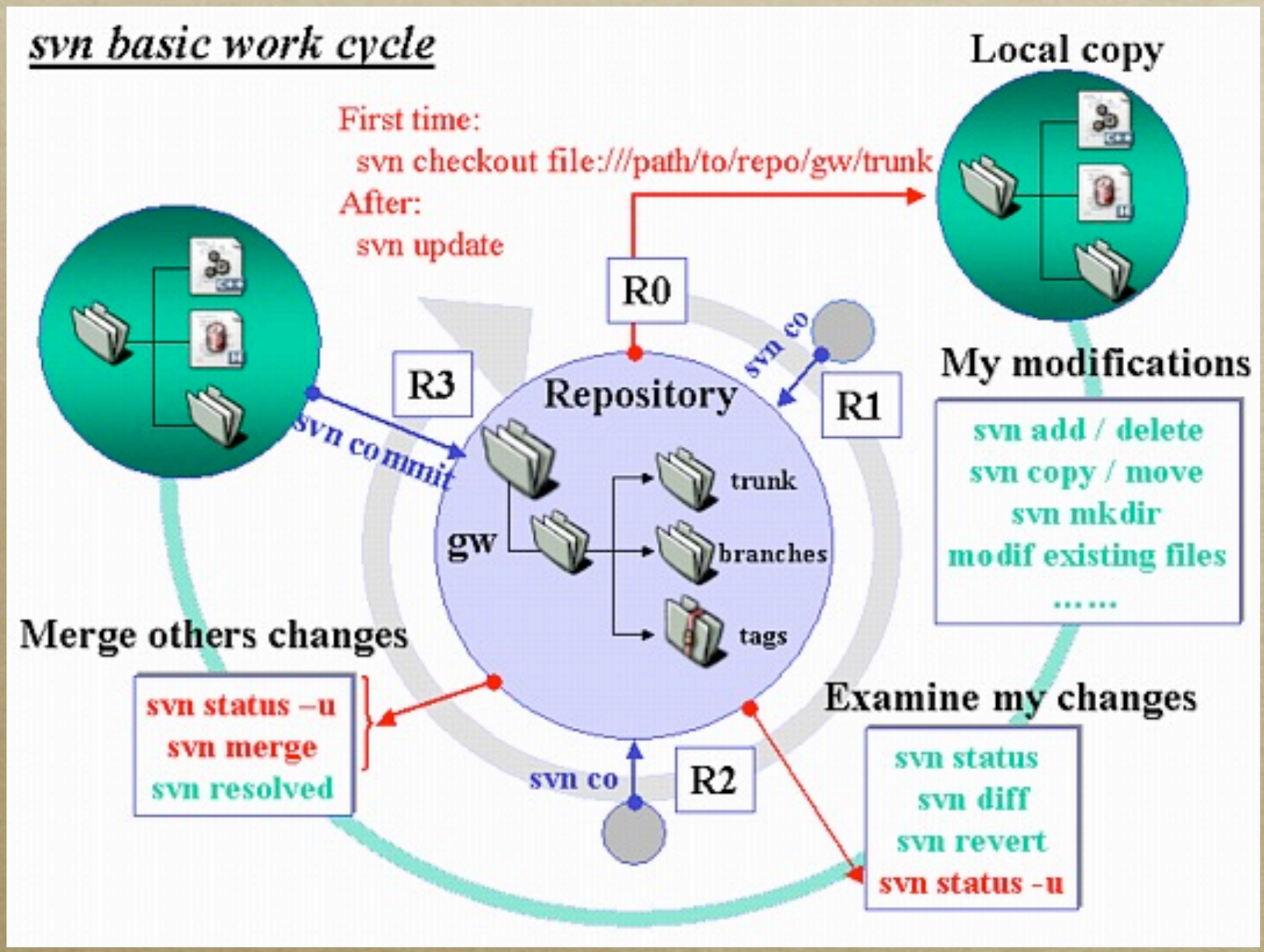


work on a local copy

synchronize the local copy with the repository

Work with svn*

**subversion*



svn cycle with the main commands

Work with svn

[svn] / gammaware / trunk / src / adf / CrystalFrame.h Repository: svn Go

Log of /gammaware/trunk/src/adf/CrystalFrame.h

Compare 1163 vs. Local — CrystalFrame.h

Debug | root5.24D_dev | x86_64

Overview Breakpoints Build and Debug Tasks Ungrouped Project

Compare 1163 vs. Local — CrystalFrame.h

<No selected symbol>

```
   //! to get individual segment
   /*!
   */
   virtual GeSegment  *GetSegment(UShort_t) = 0;

   //! to get each core
   /*!
   */
   virtual GeCore     *GetCore(UShort_t) = 0;
};

//! General interface for a CrystalFrame
/*!
   A crystal frame gives some global data, 36 segments signals and 2
*/
class ACrystalFrame : public AgataDataFrame< CrystalInterface >
{
protected:
   ACrystalFrame(const Key *akey):
      AgataDataFrame< CrystalInterface >(akey) {;}
public:
   virtual ~ACrystalFrame()
      {;}
};

// typedef ProxyDataFrame<ACrystalFrame,CrystalInterface> CrystalFra

} // namespace ADF
#endif
```

<No selected symbol>

```
   //! to get individual segment
   /*!
   */
   virtual GeSegment  *GetSegment(UShort_t) = 0;

   //! to get each core
   /*!
   */
   virtual GeCore     *GetCore(UShort_t) = 0;
};

//! General interface for a CrystalFrame
/*!
   A crystal frame gives some global data, 36 segments signals and 2
*/
class ACrystalFrame : public AgataDataFrame< CrystalInterface >
{
protected:
   ACrystalFrame(Key *akey):
      AgataDataFrame< CrystalInterface >(akey) {;}
public:
   virtual ~ACrystalFrame()
      {;}
};

// typedef ProxyDataFrame<ACrystalFrame,CrystalInterface> CrystalFra

} // namespace ADF
#endif
```

Work with svn

❶ *Release versions: not that used ... so far (GSI ?)*

lack of policy (root, any 6 months with objectives)
at least be aware of the version (release) you are using

❷ *Mercurial (Hg) also used @ GSI*

(different but same philosophy)

❸ *Good practice in case of problems/bugs, give :*

the distribution (which linux, mac), environment
version of the codes, output etc ...

☛ *not only : it does not work ...*

Main svn servers

'narval emulator'

svn://gamma01.lnl.infn.it/agata/trunk/narval_emulator

Orsay Forward Tracki

http://csngwinfo.in2p3.fr:2401

GammaWare

svn+ssh://anonsvn@anonsvn.in2

➥*ADF*

svn+ssh://anonsvn@anonsvn.in2p3.fr/agata/gammaware/src/adf

python script
AgataLegnaro.py
to help
loading, compiling, installing
(see practical sessions)

Questions : GSI, (GANIL), other Tracking, PSA ?

Others useful sites

GammaWare Head Version for release 0.9

Search

Gw::Link
Gw::LoopControl
Gw::LoopOnTasks
Gw::LSaxis
Gw::MatPlayer
Gw::Measure< Data_T >
Gw::NuclearLevel
Gw::PadManager
Gw::Parity
Gw::Peak1D
Gw::PeakCreator
Gw::QNumber
Gw::RadLevelSchemeReader
Gw::RandObj
Gw::Random
Gw::RootSpectrumPlayer
Gw::Spin
Gw::StdHit
Gw::SymCorrelatedSpace
Gw::TestFeeding
Gw::TrackHit
Gw::TTreeBuilder

AutoSave
DoCanvas
Exec
GetLastSaveArg1
GetLastSaveArg2
GetLastSaveArg3
GetLastSaveArg4
LoopControl
Save
SetAutoTime
Zero
~LoopControl

```
void LoopControl::Save ( const Char_t *  main,
                         const Char_t *  tag,
                         const Char_t *  option,
                         Int_t           autotime
                       )
```

Save the spectra in files.

- main_file is the name of the root file to save spectra
- tag_file is the name of the root file to save tagged spectra
- option has the following format :
  - ; means save spactra in the current TDirectory
  - (means open a new file
  -) means close the file
  - ch means works with a chain i.e. sequence of names with the same pattern.
  - option to Zero spectra is given using {opt1:opt2}.
- autotime : call automatically this command again after autotime seconds. Off if set to 0.

Here are some examples.

- if you have open your watcher is a root directory and would like just to save there the current histograms : "" "0" ";" 0
- to save in the current root dir, open a new one and move all hitograms there : "my_new.root" "0" ";(" 0
- to dump histograms in a root file (open, save and close the file) : "my_dump" "0" "(;)" 0
- to dump histograms in a chain of root files (open, save and close the file) : "my_dump" "0" "ch(;)" 0
  it looks for existing my_dumpXXXX.root files in the current dir XXXX = [0, ...[. It takes the first non existing one.
- to open a new files that becomes the mother file of all watchers, and save histo there : "my_dump" "0" "(;" 0
- to set spectra of the pool to zero : "my_dump" "0" "(; {pool}" 0
- to set all spectra to zero and change the binning : "my_dump" "0" "(; {pool:100 0 100}" 0

Definition at line 1046 of file Watchers.cpp.

Gw › LoopControl

Generated on Wed Nov 30 2011 02:05:25 for GammaWare by doxygen 1.7.4

http://www.ipnl.in2p3.fr/gammaware/doc/html/

[html doc of the gw code]

http://agata-0.lnl.infn.it:8989/

Fill it, IMPORTANT

installation of the code

[system/user]

required root privileges

apt-get, rpm, dmg etc ...

boost

skstream

Grid, never admin

(except if we are reached to buy a virtual box on each site)

sometime better to install on the user side

For other AGATA packages, nothing like this

configure
make
make install

[user]

- *narval emulator : make, make install*
- *PRISMA : make, make install*
- *Gammaware : configure, make, make install*
- *ake install*
- *ake install*

python script
AgataLegnaro.py
to help
loading, compiling, installing
(see practical sessions)

Should be used widely in the future

Overview

Complexity

Basic elements

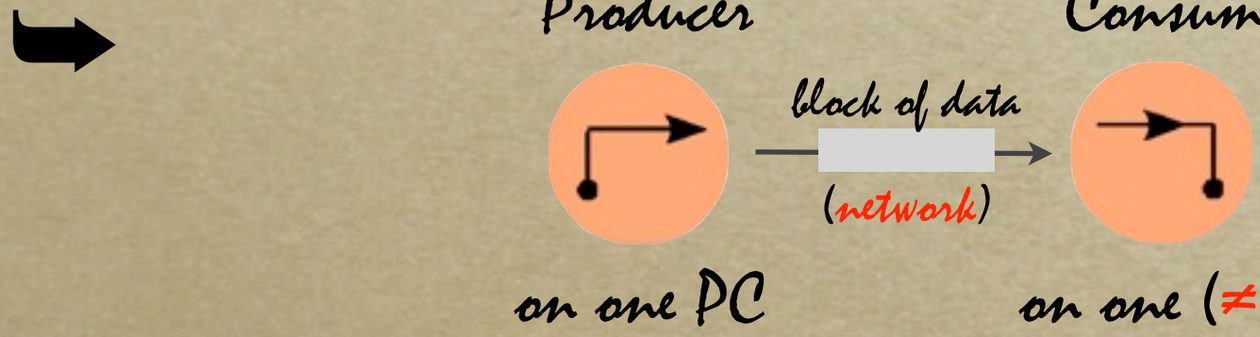
Narval philosophy

Organisation

Narval philosophy

Decomposition of processing in chain of actions

Ex - simple analysis : read data AND create spectra



Advantages { *if source of data changes, just change the producer*
the same data flow could be redirected to different consumers

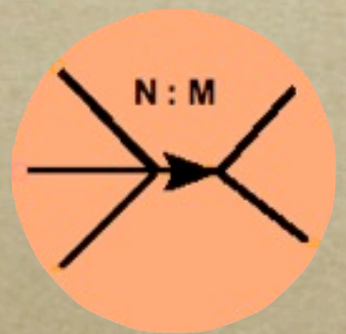
Main actors

- Producer (produced data) — *0 input, 1 output*
- Consumer (consumer data) — *1 input, 0 output*
- Filter (consume data, apply algo, produce new data) — *1 input, 1 output*

algorithms ! (tracking, PSA)

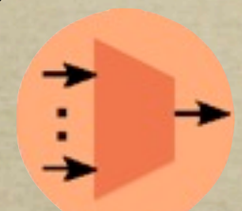
Narval philosophy

♣ *More complex actors : organisation of the data flow*



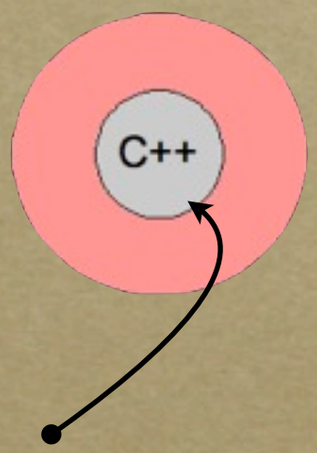
n inputs, m outputs

⇨ *Event Builder, Merger i.e. n inputs, 1 output*



♣ *Narval is written in ADA BUT can load C/C++ code*

⇨ *the library has to defined a precise interface(*)*



libActor.so

```
TrackingFilter *process_register (...)
void process_config (...)
void process_block(...)
void process_initialise(...)
void process_reset(...)
void process_start(...)
void process_stop(...)
void process_pause(...)
void process_resume(...)
```

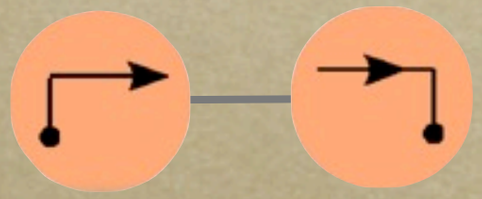
⇨ *only for producer, consumer and filter !*

(*)Agata PSA and Tracking Algorithm Integration, J. Cresswell and X. Grave, 2nd Draft

Narval philosophy

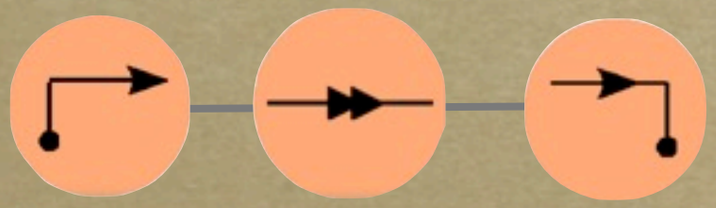
A <u>topology</u> defines how actors are connected.

some examples



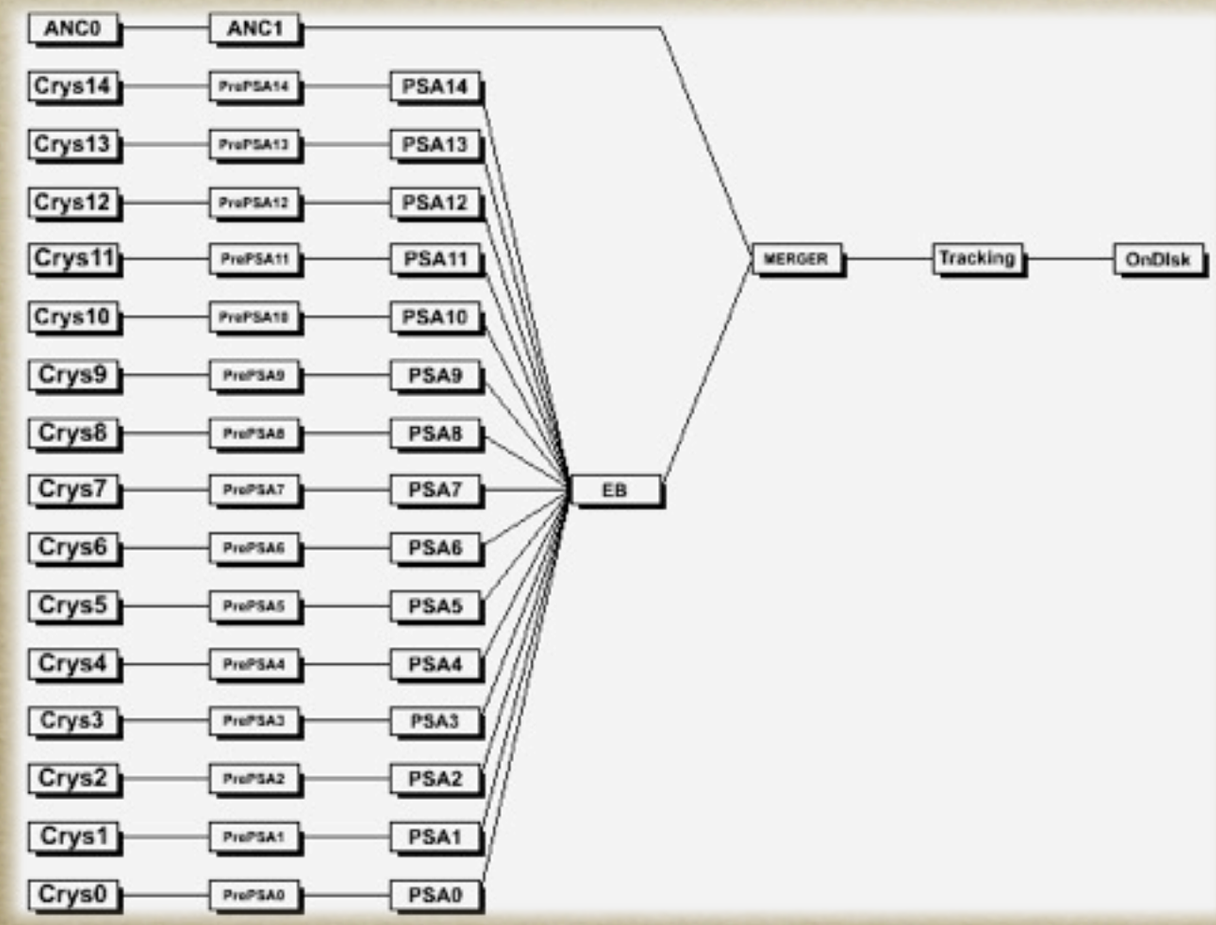
- read data, produce spectra
- read data, produce root tree

...



- read data, apply filter, save
- read data, apply algo, produce spectra

...



almost current online topology

Narval philosophy

Narval can :

- *built complex topology*
- *run actors (ADA,C++) on clusters of PC*

C++ actors can be assembled in C++ environment

- *so-called <u>emulators</u>*

Emulators are useful for :

- *debugging*
- *developing new actor*
- *'light' analysis ... i.e. does not required parallelisation*

You can also install Narval in your institute !

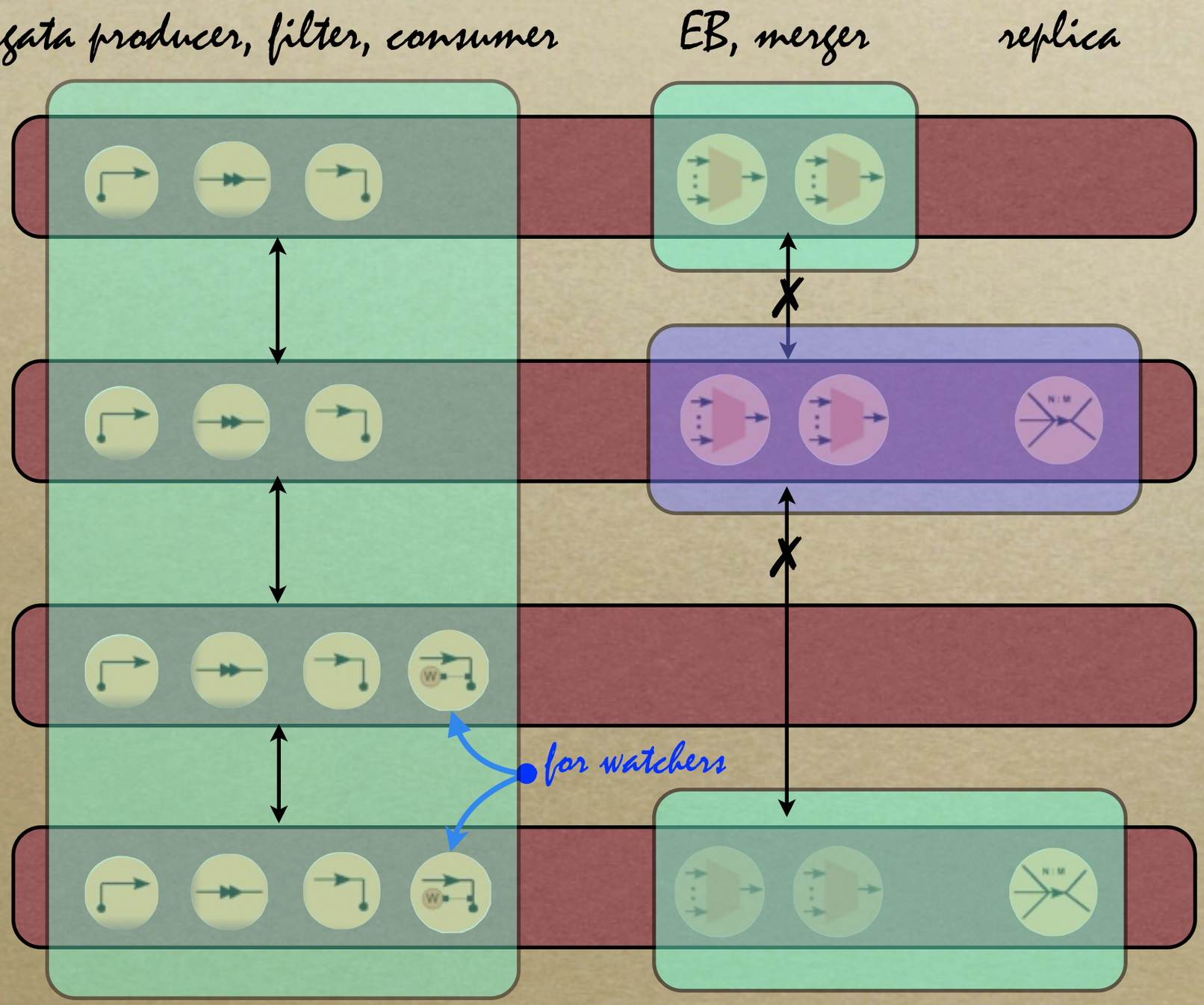
Actors in frameworks

femul
standalone exec

Narval
framework

ROOT[Gw]
framework

ROOT[Gw]
framework



c++
ada

EmulatorPC
EmulatorPFC

DEmulator
RMTEmulator ...
(BoostEmulator)
ProofEmulator ?

Agata shared actors

The svn called 'narval-emulator' contains :

ADPP

- *the actors shared between the frameworks*
  - ➥ (`BasicAFC, AncillaryProducerTCP, TrackingFilter PSAFilterGridSearch etc ...`)
- *the PRISMA library*
- *femul + femul event builder / merger*

ADPD

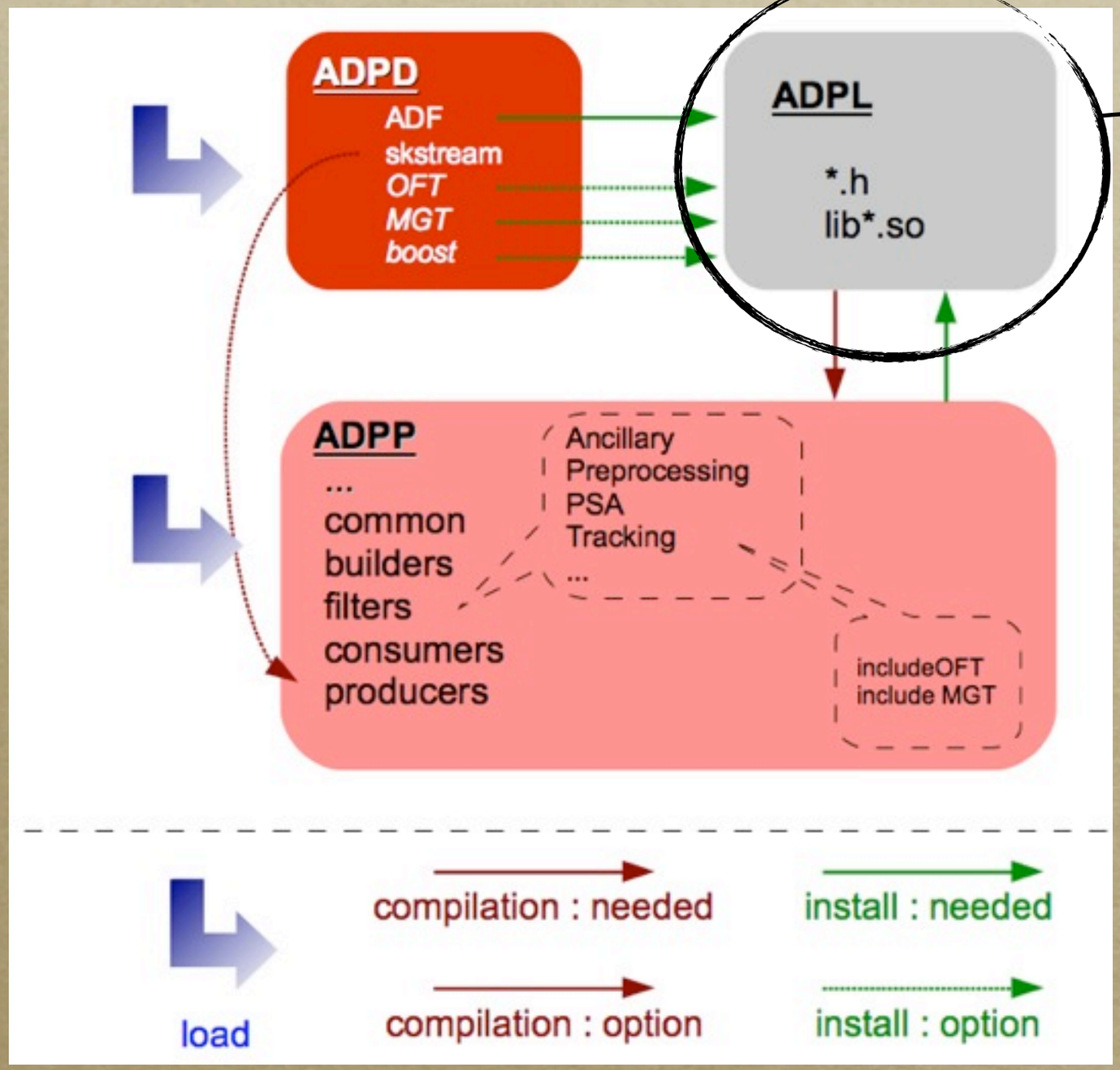
+ a copy of ADF, OFT

- *the actors shared between the frameworks*
  - ➥ `Should be external packages !` (GSI?)
  - ➥ `Otherwise pb of synchronisation`

All libraries built in ADPL

Note : ADP is for Agata Data Processing { P for Package, D for Dependencies, L for Library

Agata shared actors



needs

Narval

/

Emulators

+

PRISMA

PreSPEC

...

Complex environment

Narval

Femul

Root

Emulators

ISMA

I hope you have a better view of this !

ADF

Boost

GammaWare

ADPD

ADPL

Overview

Basic elements

What already exists

How to extend

Overview

Actor's actions

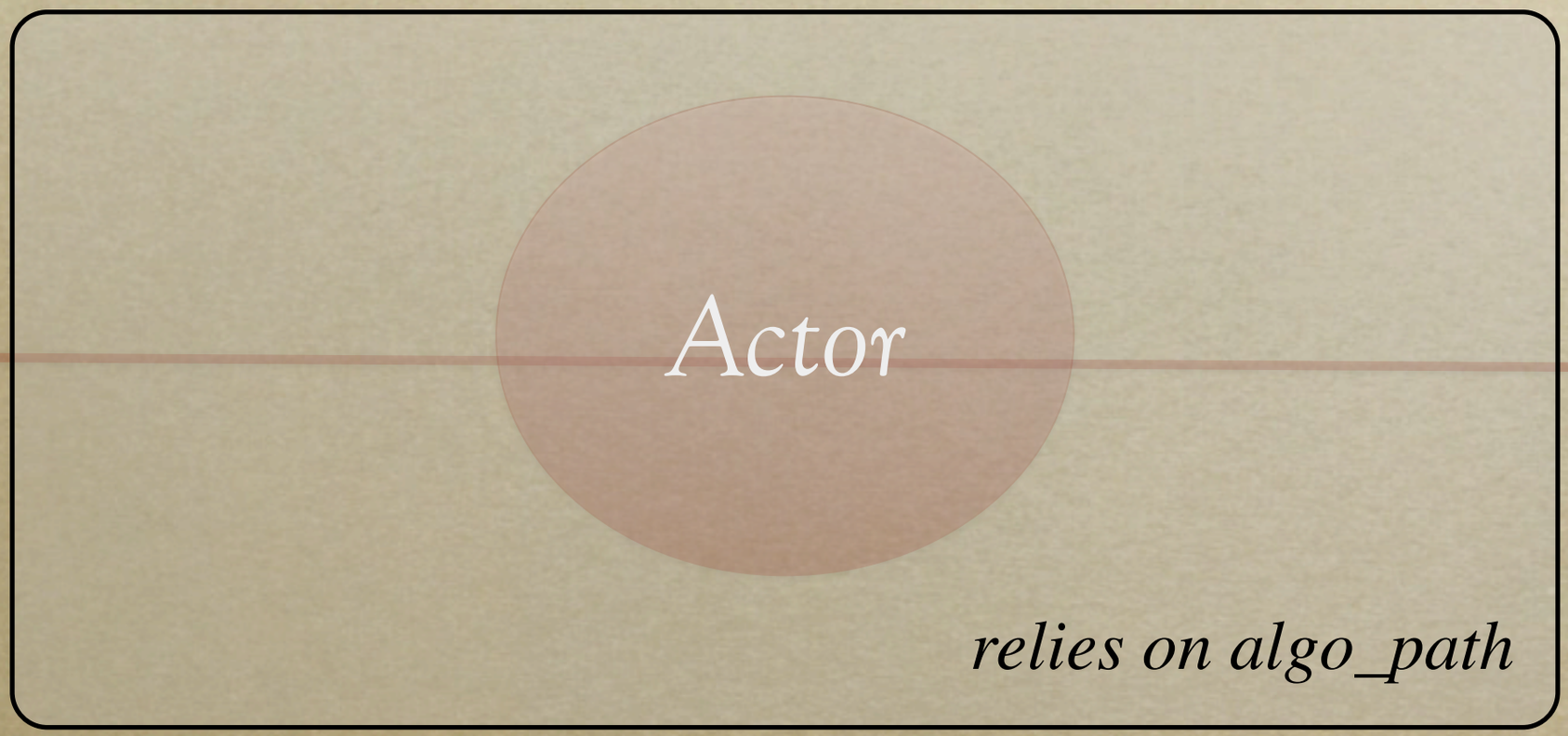
What already exists

Watchers

Data flow structure

Actor's communication

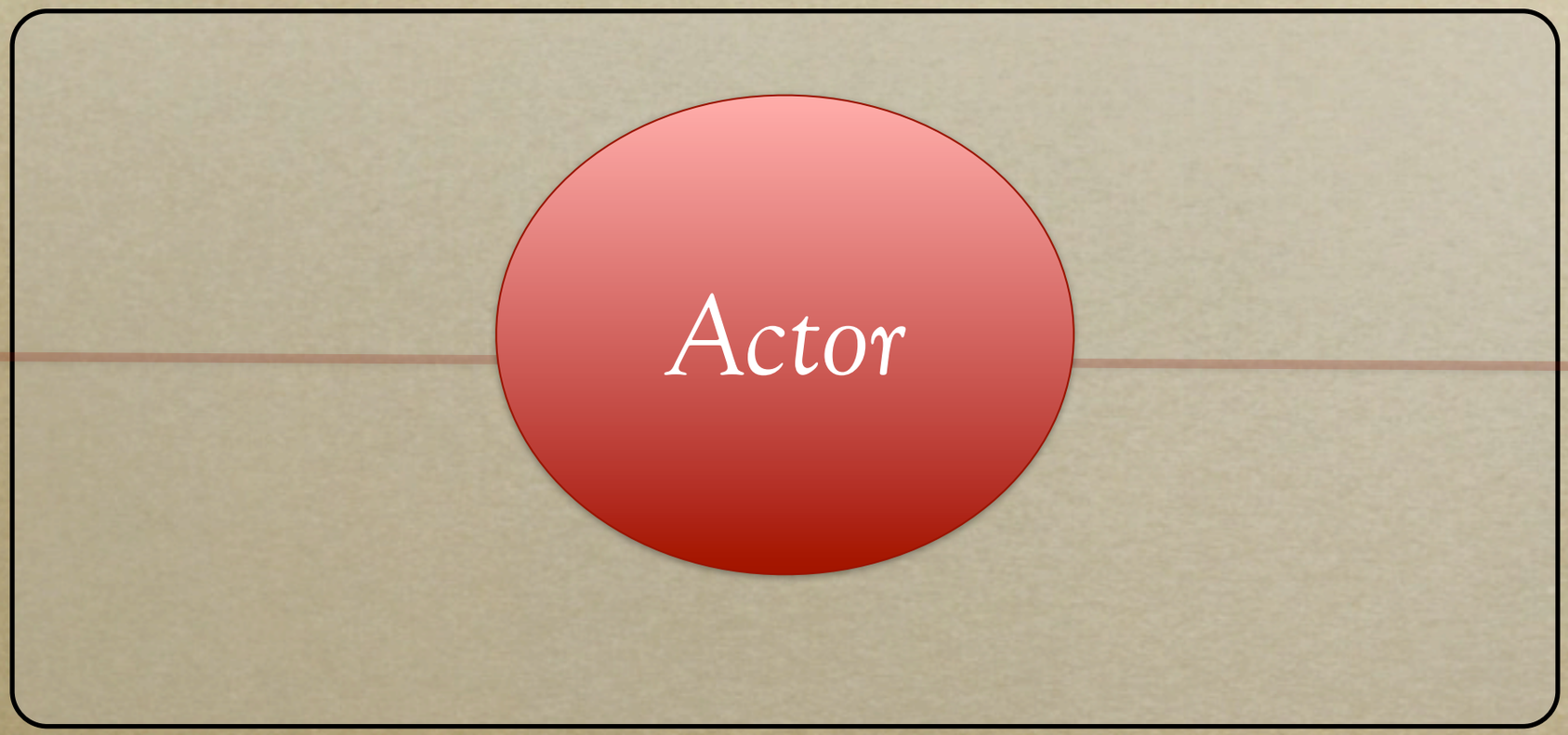
an actor is configured first ...



... means ~ the environment around is set

Actor's communication

an actor is configured first ... then created



Actor's communication

the actor is initialised (just once) using .conf files

for PSA algorithm

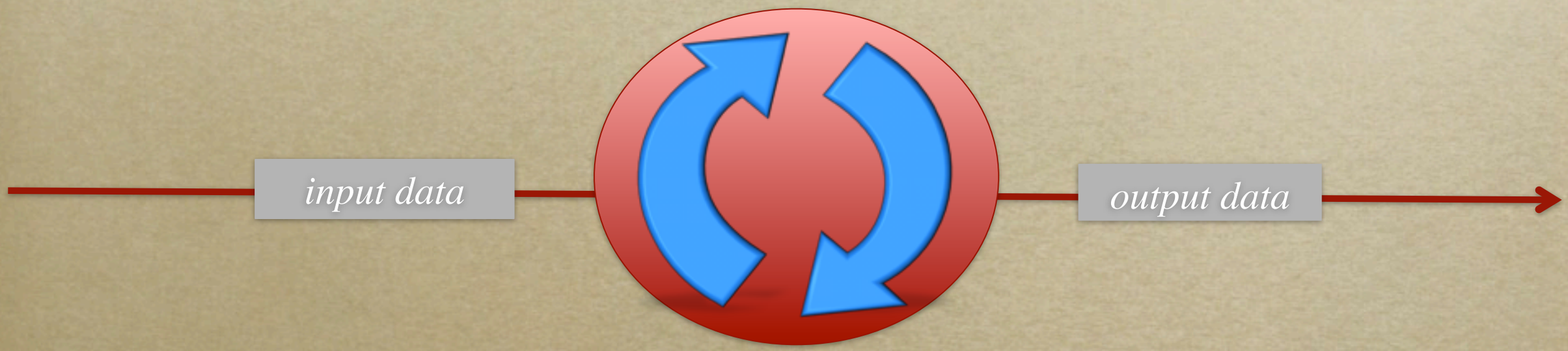
ActualClass PSAFilterGridSearch
BasisFile /agatadisks/bases/ADL/LibTrap_C002.dat
SaveDataDir /agatadisks/data/2010_week24/Replay_run100/Out/1B
EnergyGain 2
WriteTraces 1000
XtalkFile xinv_1325-1340.cal

for tracking algorithm

ActualClass TrackingFilterMGT
SaveDataDir /agatadisks/data/2010_week24/Replay_run100/Out/Global
EnergyGain 2
Ancillary
SourcePosition 0 0 -59
TimeWindowGeAnc 640 690
WriteInputHits

Actor's communication

at running time ...



the actor

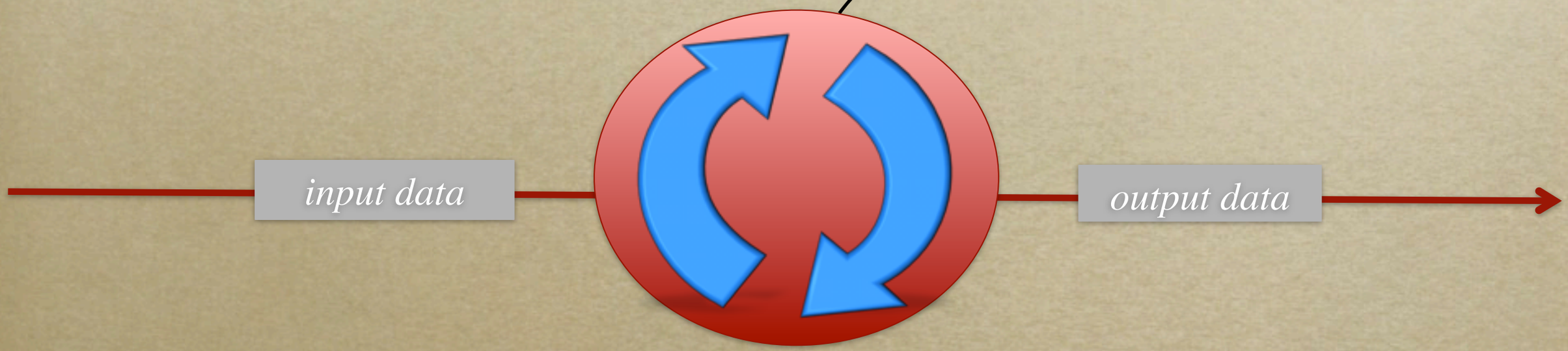
- *extracts data from the input*
- *consumes them or not*
- *produces new data*

not only data can come from input
(ex : configuration frames)

Actor's communication

at running time ...

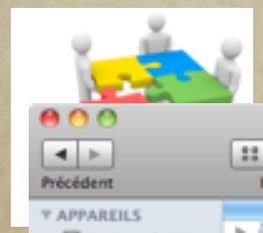
messages (printout, log system)



the actor

- *sends messages*

Actor's communication



save spectra in files

SaveDataDir /agatadisks/data/2010_week24/Replay_run100/Out/1B

Prep__1000-42-100-S__Traces.samp
Prep__2-10-16384-UI__Esum.spec
Prep__2-40-1024-UI__TT1.spec
Prep__38-16384-UI__Ener.spec
Prod__1000-42-100-S__Traces.samp
Prod__38-65536-UI__Ener.spec
Psa__1000-2-42-60-S__Traces.samp

or spectra available with GRU

actor is a server of spectra
☛ *machine, port #*

the actor

- *produces spectra (if activated)*

Actor's communication

at running time ...

Set/GetParameters()

Only in Narval

GSI !

input data

output data

the actor can

- *have some parameters modifie*
- *deliver copies of output from ti*
  - ➥ *Watchers*

```
in GW, OnlineWatchers.C

   // machine on which crystal producer is running
   std::string gMachineCrys[gNumberOfCrystals] =
   {
        "10.10.1.119", "10.10.1.113", "10.10.1.114",
        "10.10.1.115", "10.10.1.116", "10.10.1.117",
        "10.10.1.109", "10.10.1.110", "10.10.1.111",
        "10.10.1.106", "10.10.1.107", "10.10.1.108",
        "10.10.1.103", "10.10.1.104", "10.10.1.105"
   };
   UInt_t gPortCrys[gNumberOfCrystals] =
   {
        9012, 9013, 9014,
        9015, 9016, 9017,
        9009, 9010, 9011,
        9006, 9007, 9008,
        9003, 9004, 9005
   };
```

Overview

Actor's actions

What already exists

Watchers

Data flow structure

Data flow structure

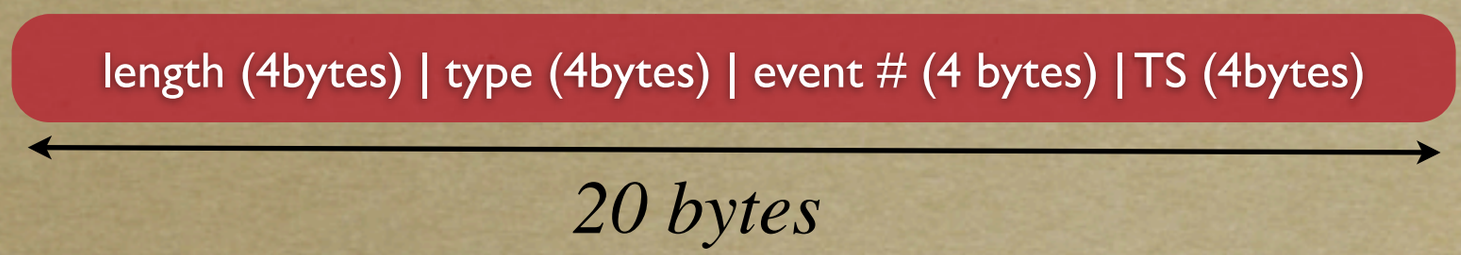
How look like the input and output buffers ?

➥ **Basic element : Frame**

*A Frame is composed of a **key** and a **content***

the key part give information on the content ☛ *always starts with frame length* ☛ *mapping possible without knowning the content*

Agata key



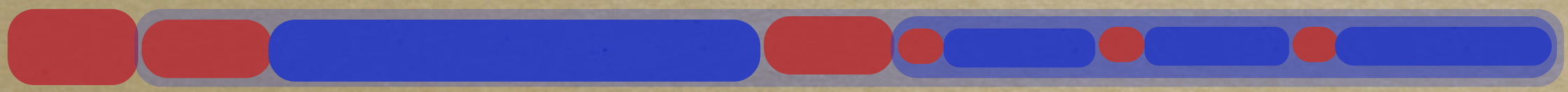
Data flow structure

There are different kind of Frames

- *data : produced (consumed by algo.) [binary]*
- *conf : to pass human readable info [ascii]*
- *meta : general data (ex: Vertex)*

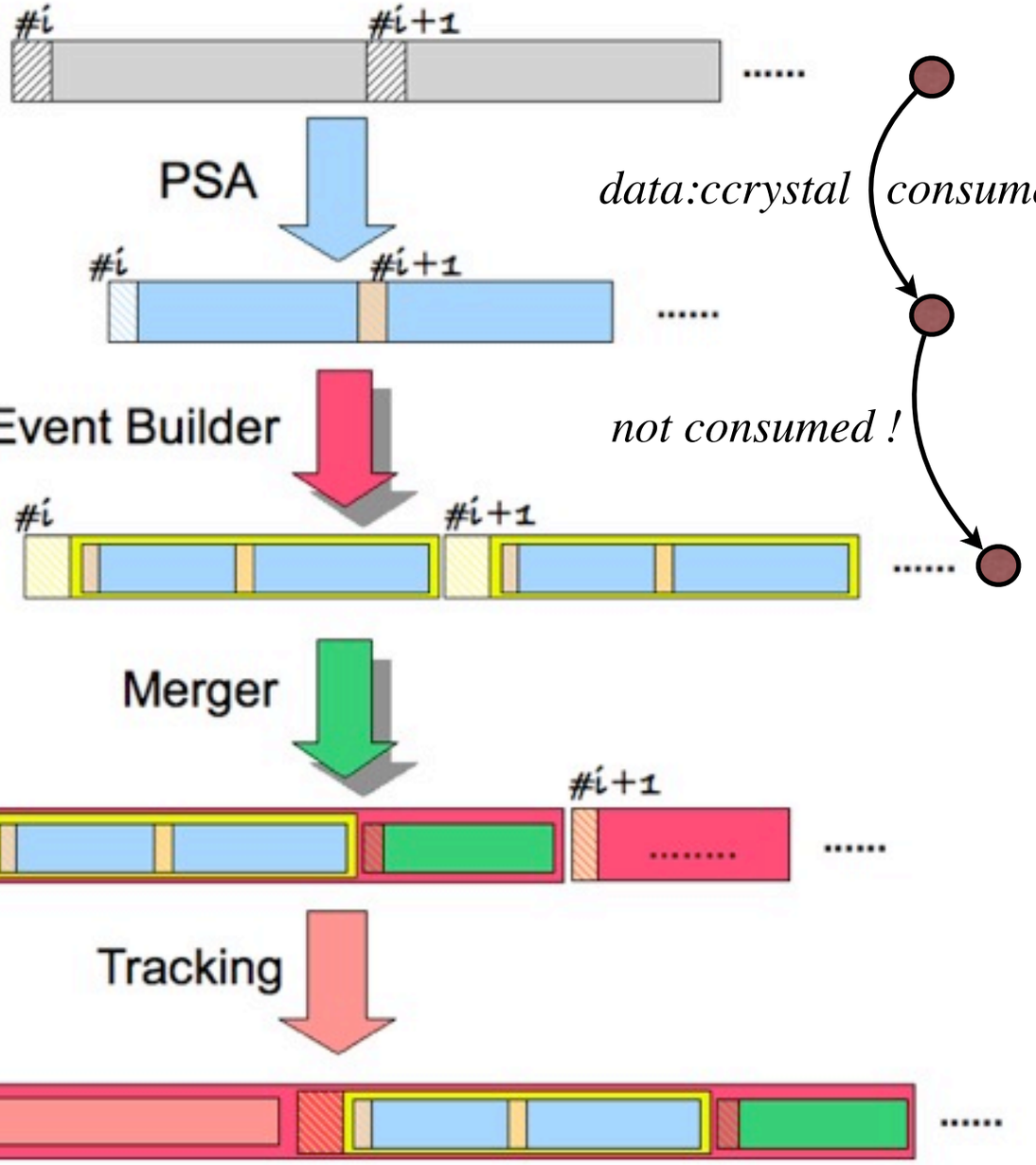
A Frame could be composed of Frames : CompositeFrame

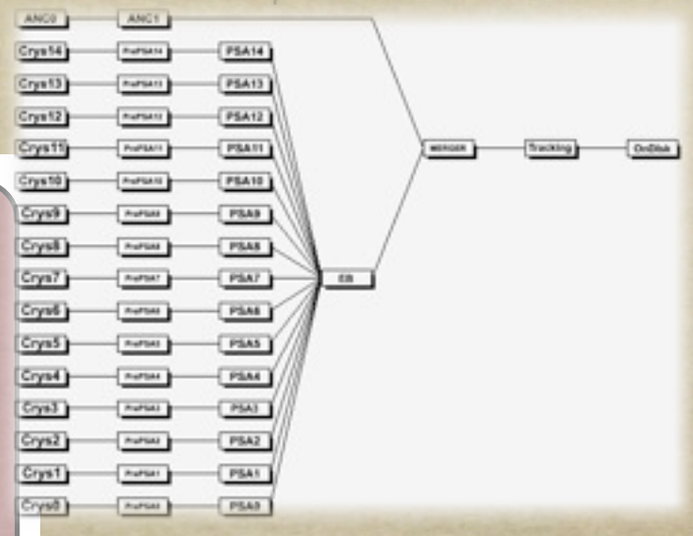
➥ *coincidences ≋ event*



Data flow structure

Frames currently in uses

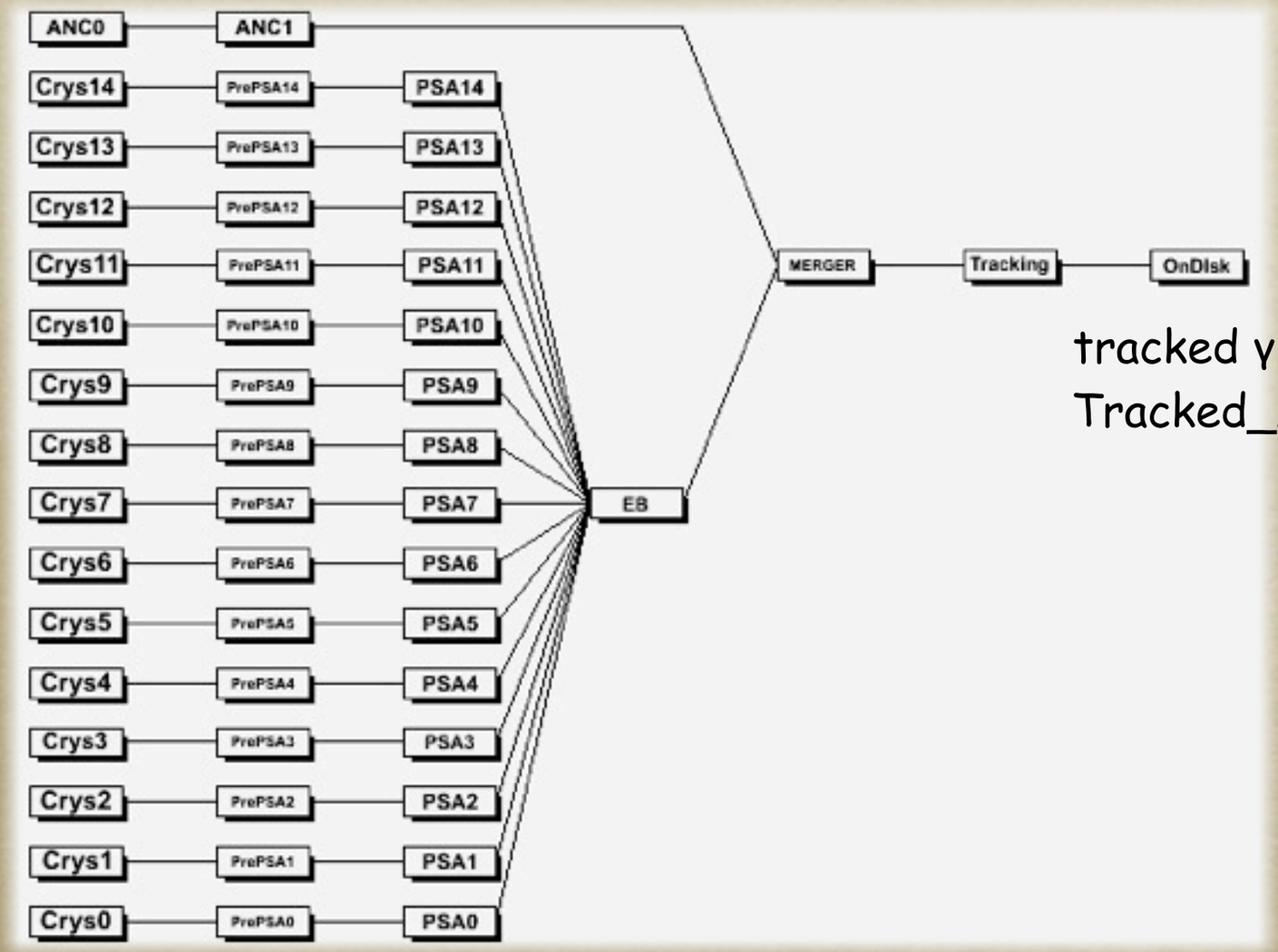




disctinction using Key[type]

to be defined @ GSI

Data flow structure



tracked γ saved in
Tracked_XXX.adf

traces saved in
event_mezzdata.bdat

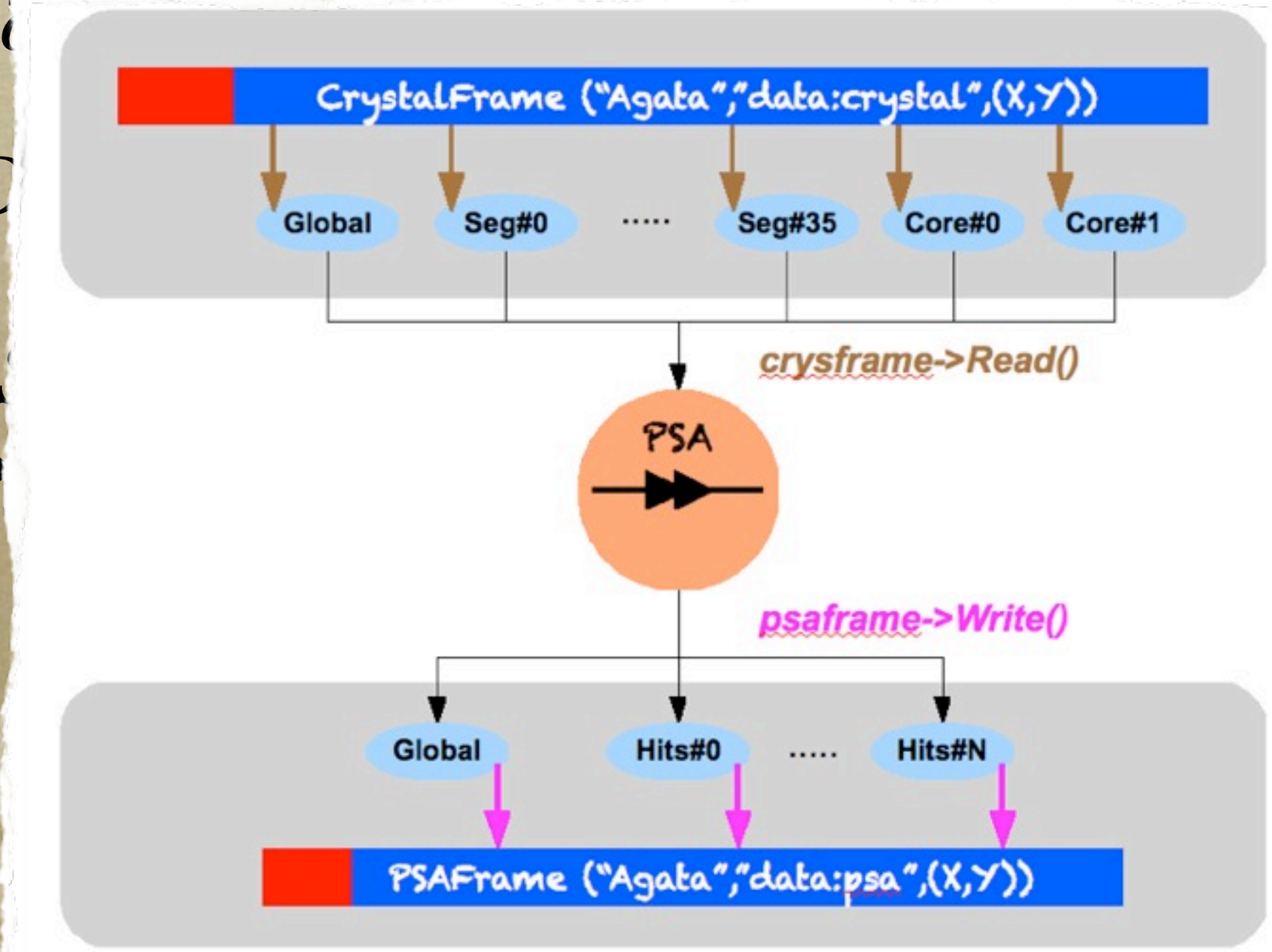
psa hits saved in
PSA_1R_XXX.adf

The Data Interface

To

D

Ex : PS



virtual void SetT (Double_t, UInt_t=0u)=0
to set the time associated to the core

Frame has versions

Why version numbers ?

AGATA : 10 years project

➥ *algorithms (Tracking, PSA) likely to change*

➥ *better to include this since the beginning*

small modifications - float ☛ double

Version (Major, Minor)

large modifications - error on hits

Frame has versions

the full definition of circulating frames are in ADF.conf

```
...
Frame:  Agata data:crystal 4 0  Agata data:crystal 65000 0
Frame:  Agata data:ccrystal 4 0  Agata data:ccrystal 65000 0
Frame:  Agata data:ranc0 4 0  Agata data:ranc0 65000 0
Frame:  Agata data:ranc1 4 0  Agata data:ranc1 65000 0
Frame:  Agata data:psa 4 0  Agata data:psa 65000 1
Frame:  Agata data:tracked 4 0  Agata data:tracked 65000 0
Frame:  Agata event:data 4 0  Agata event:data 4 0
Frame:  Agata event:data:psa 4 0  Agata event:data:psa 4 0
Frame:  Agata meta:vertex 4 0  Agata meta:vertex 0 1
...
```

Note : ADF.conf should be in the directory pointed by ADF_CONF_PATH

```
SharedFP * FrameTrigger::Add ( const FactoryItem &  key_item,
                               const FactoryItem &  frame_item,
                               bool                 iscons = true,
                               const Char_t *       option = ""
                             )                      [virtual]
```

Add a frame to the list of required frames.

the frame is added with a flag to tell if the given frame is consumable or not i.e. if it is still on the output dataflow in case the trigger defines also an output frame. isconc true if

A FrameTrigger has a tree (stack) structure as followed :

```
MainFrame (#0)
        SubFrame (#1)
        SubFrame (#2)
        ...
```

The mainframe is mandatory, other ones (subframes inside the main one) are optional.

Options are :

- none (default) means one and exactly one such a kind of frame is expected to trig
- ! : anti coincidence means the current trigger does not contain the frame
- | : means the frame is there or not. In such case, check if the frame is there with IsIndividualFired

Implements ADF::DFTrigger.

Reimplemented in ADF::AgataFrameTrigger.

Definition at line 594 of file Trigger.cpp.

```
AgataFrameTrigger *trig =
     AgataFrameTrigger::Build("TRACKING", "event:data event:data:psa");
```

adf fill the frame for you, not the data interface

Overview

Actor's actions

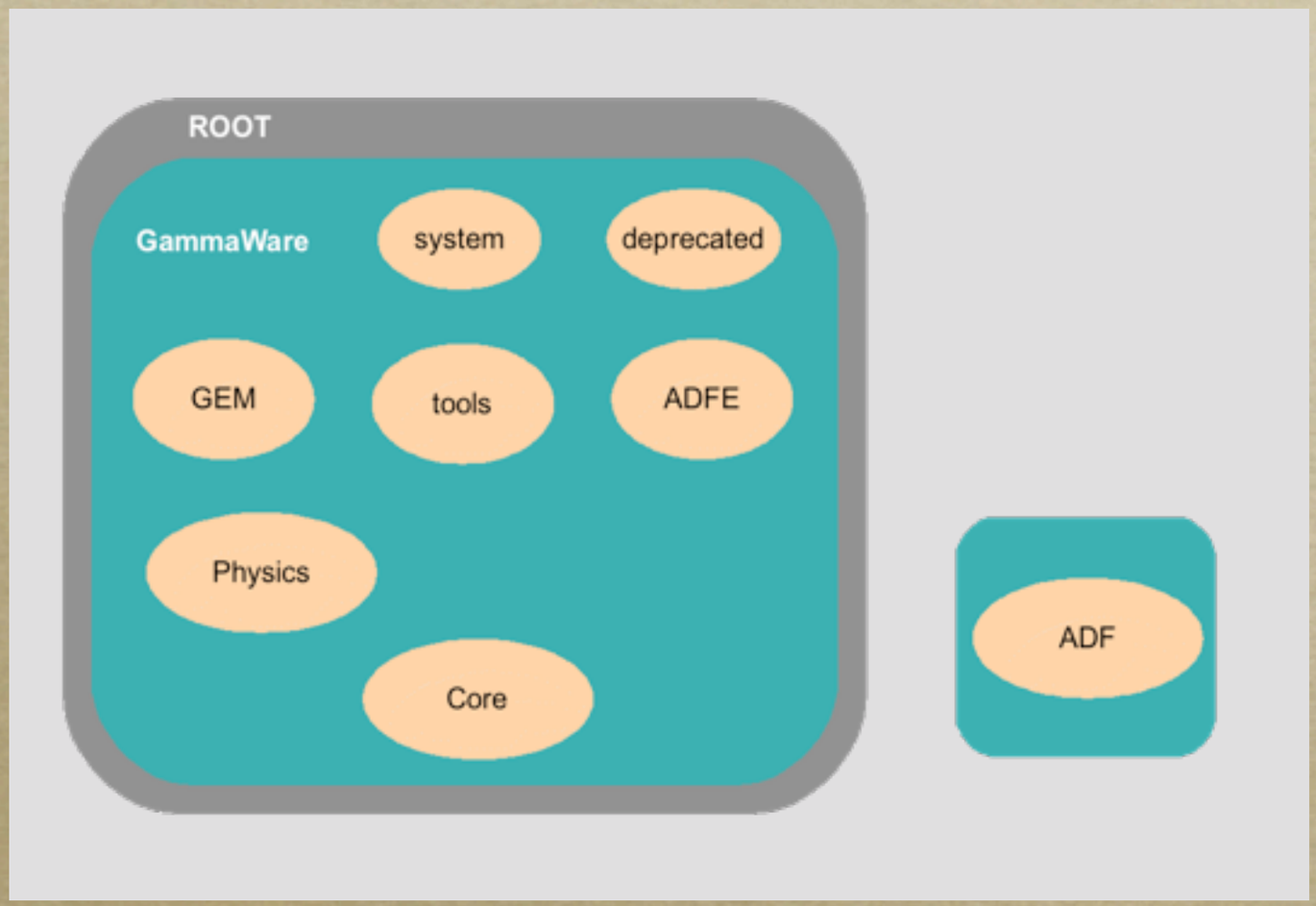
What already exists

Watchers

Data flow structure

GammaWare

ROOT : huge framework for data analysis



Gw: libraries to add to root γ-ray like analysis

Watchers[Gw]

<u>Def</u> : task dedicated to a given Frame (Trigger)

idea : re-usable bricks

<u>Ex of watchers</u> :

on data:psa

task to display hits in 3D

on data:ranc1

display raw spectra

calibrate + do recoil velocity, make it available for other watchers

display recoil velocity

on data:tracked

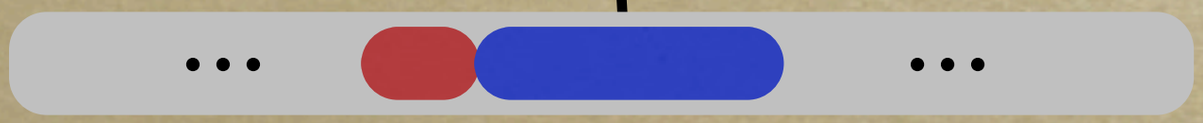
γγ matrix

write out in a root tree

Watchers[Gw]

Watchers can be linked in actors

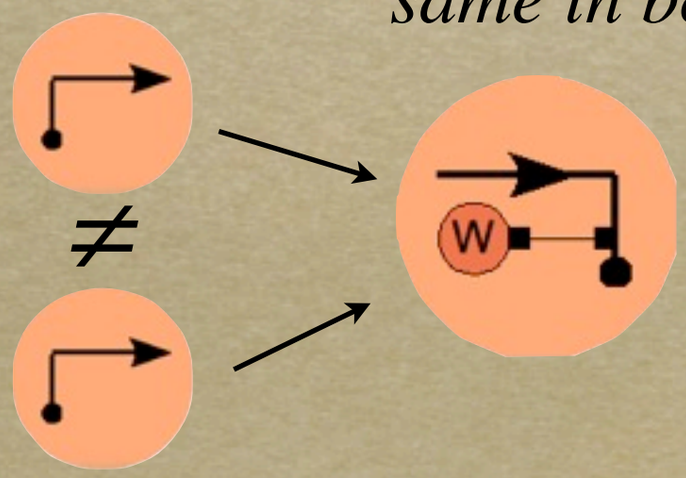
```
...
FrameDispatcher *fd;
...
// a trigger on event with hits, tracked gammas and ancillary
AgataFrameTrigger *trig =
     AgataFrameTrigger::Build("TRACKING", "event:data event:data:psa data:tracked data:ranc1");

// set the main trigger to coincidences between ancillary et gammas
fd->SetTrigger(trig);

// specific to tracked frame
// gamma-gamma matrix
fd->Add<Coinc2D>(GetWN("GxG",ext),"Gamma Gamma coincidence");
...
```



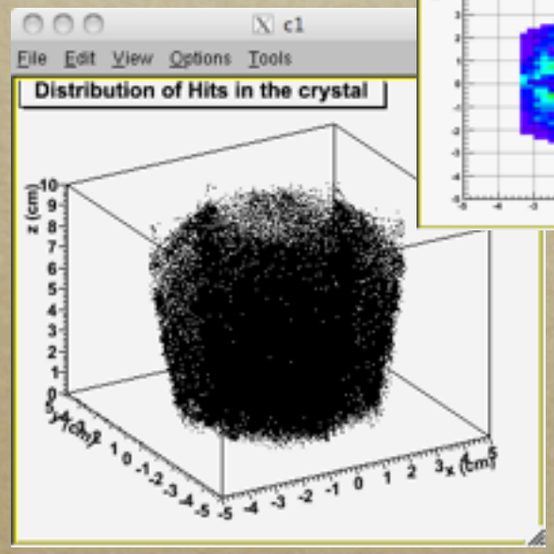
same in both cases !!

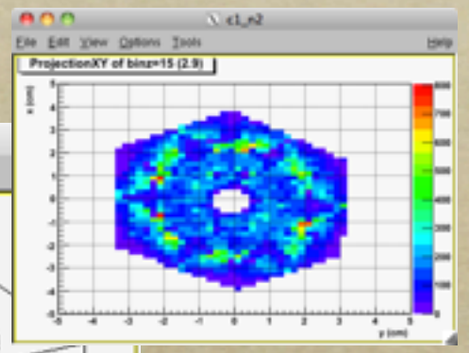
Online, get data from socket

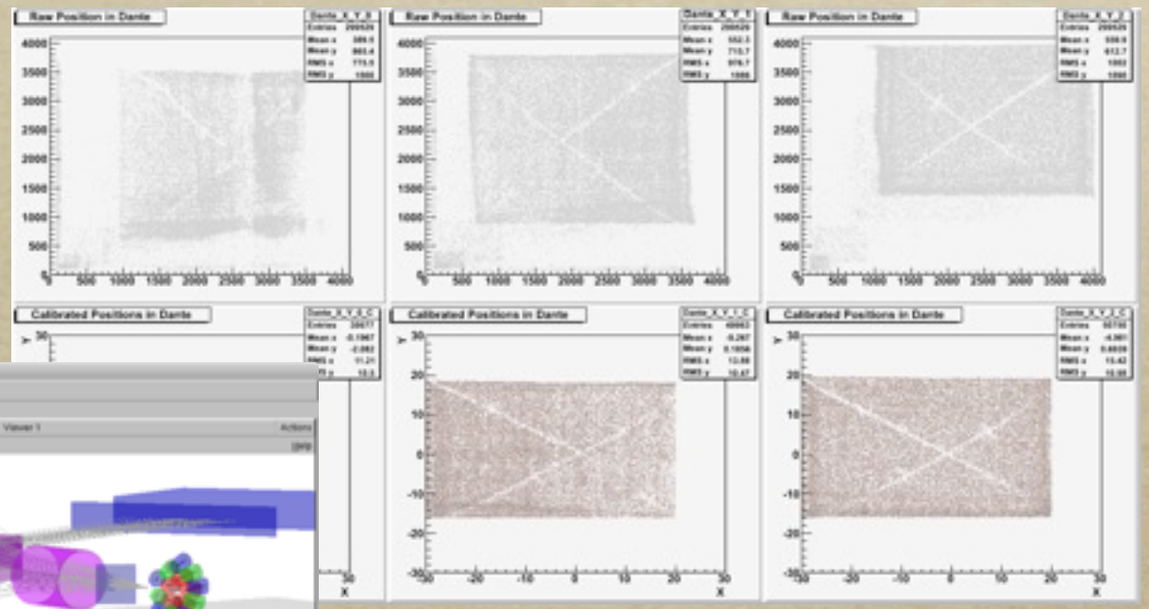
Offline, get data from .adf files

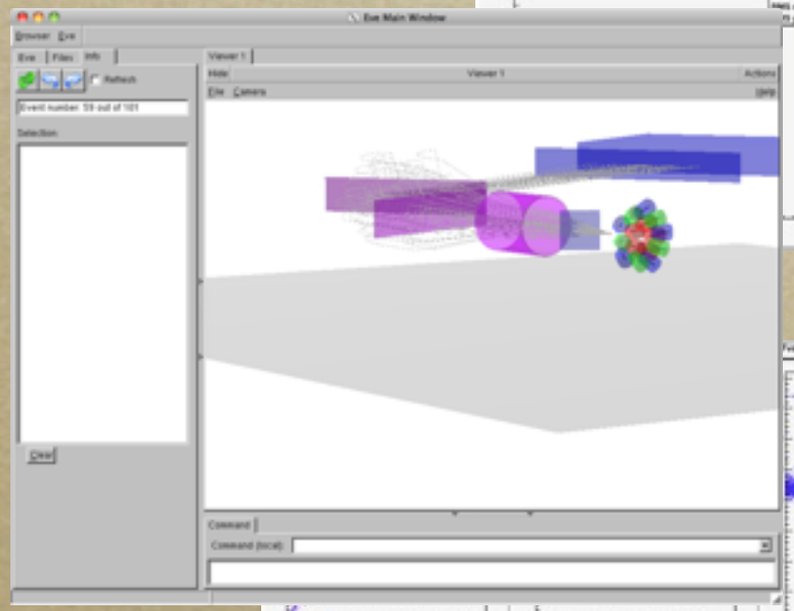


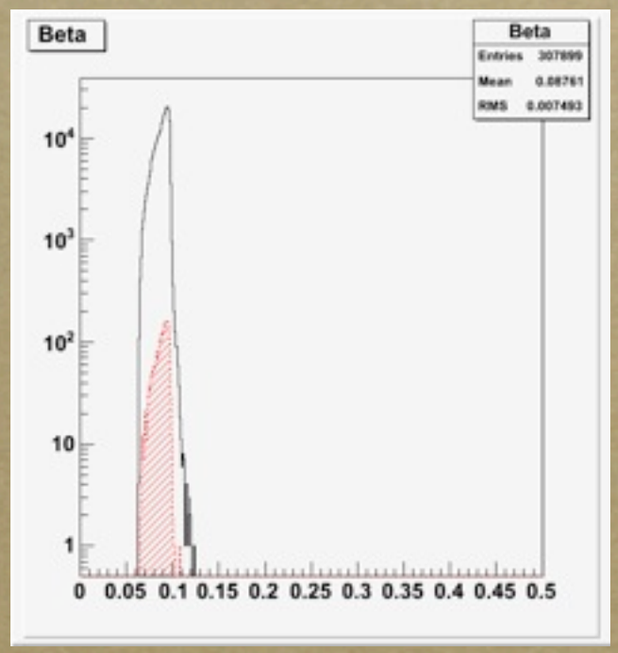
Watchers[GW]

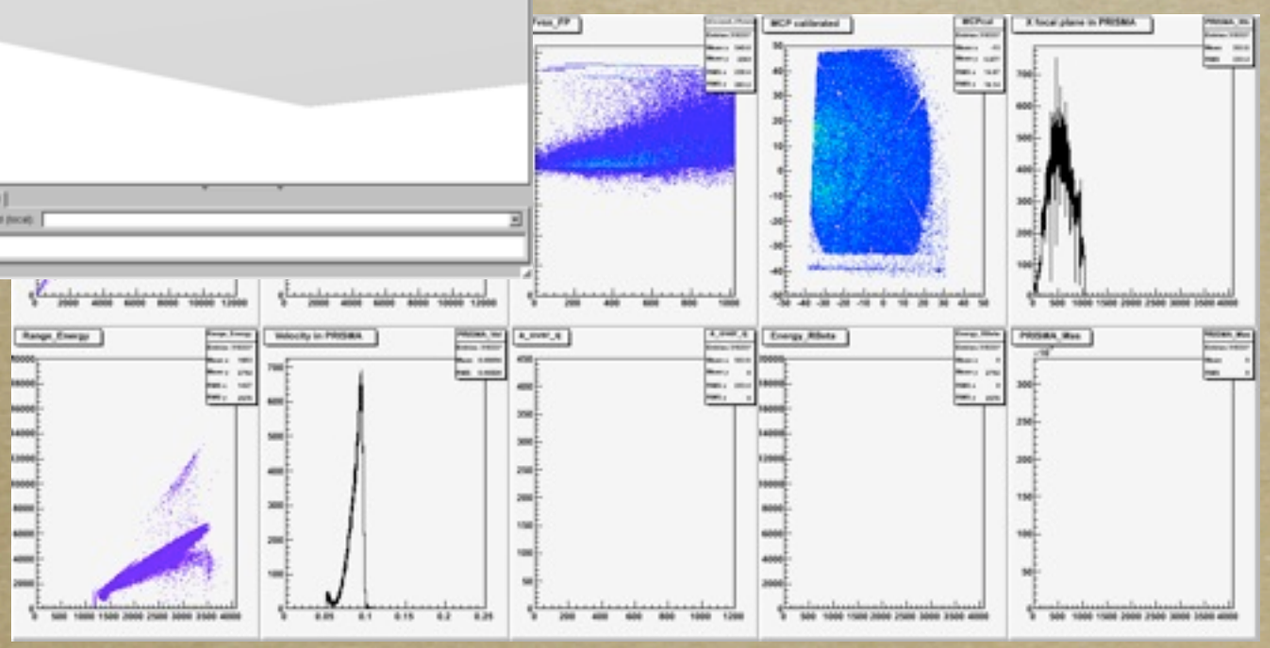

c1
File Edit View Options Tools
Distribution of Hits in the crystal
z (cm)
x (cm)


c1_n2
File Edit View Options Tools
ProjectionXY of binz=15 (2.9)

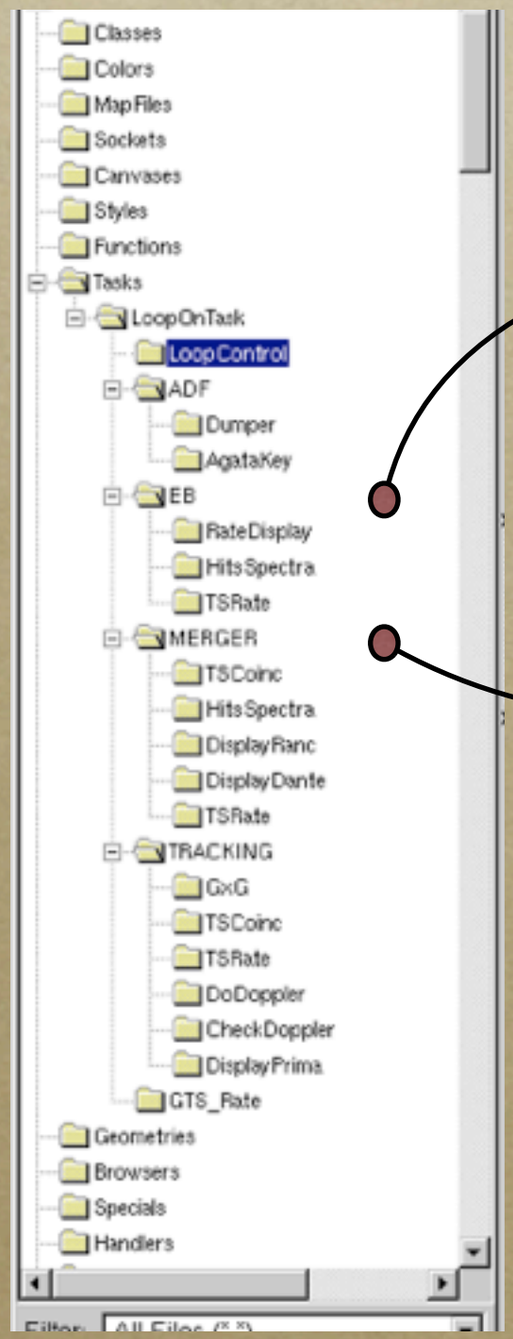

Raw Position in Dante
Calibrated Positions in Dante


Eve Main Window

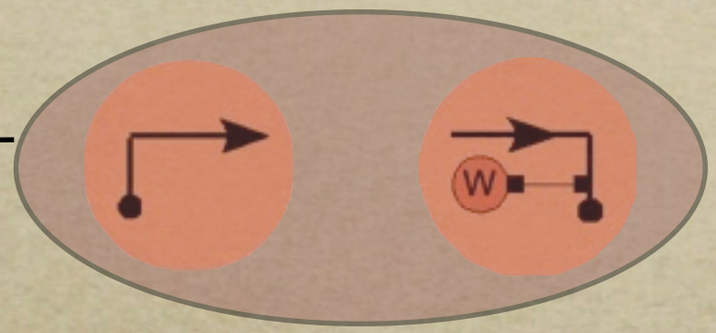

Beta
Entries 307899
Mean 0.08761
RMS 0.007493

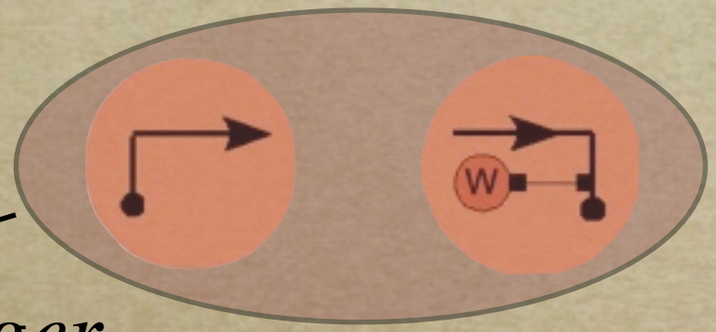

MCP calibrated
X focal plane in PRISMA
Range_Energy
Velocity in PRISMA
Energy_RBeta

Watchers[Gw]



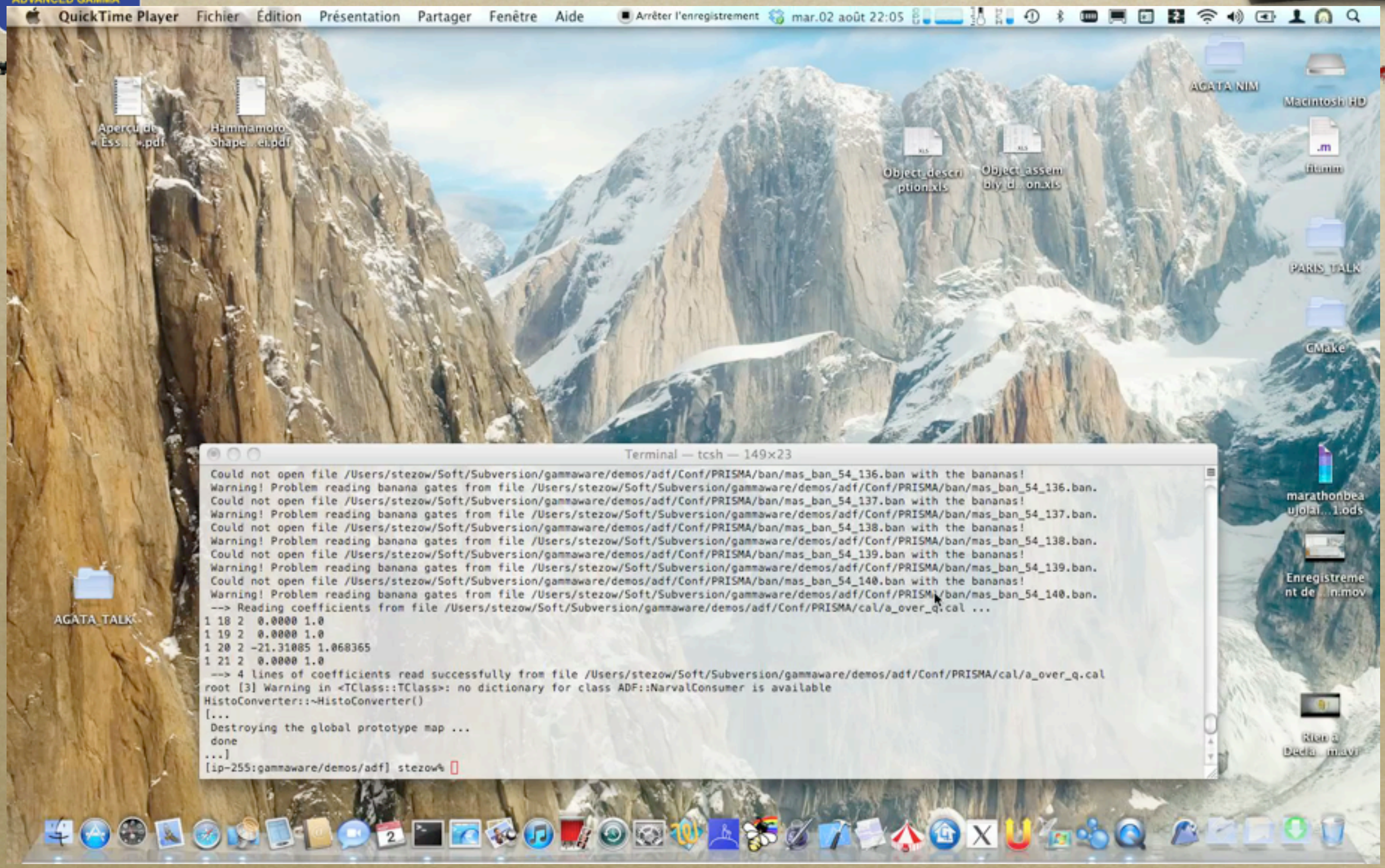
out of EB





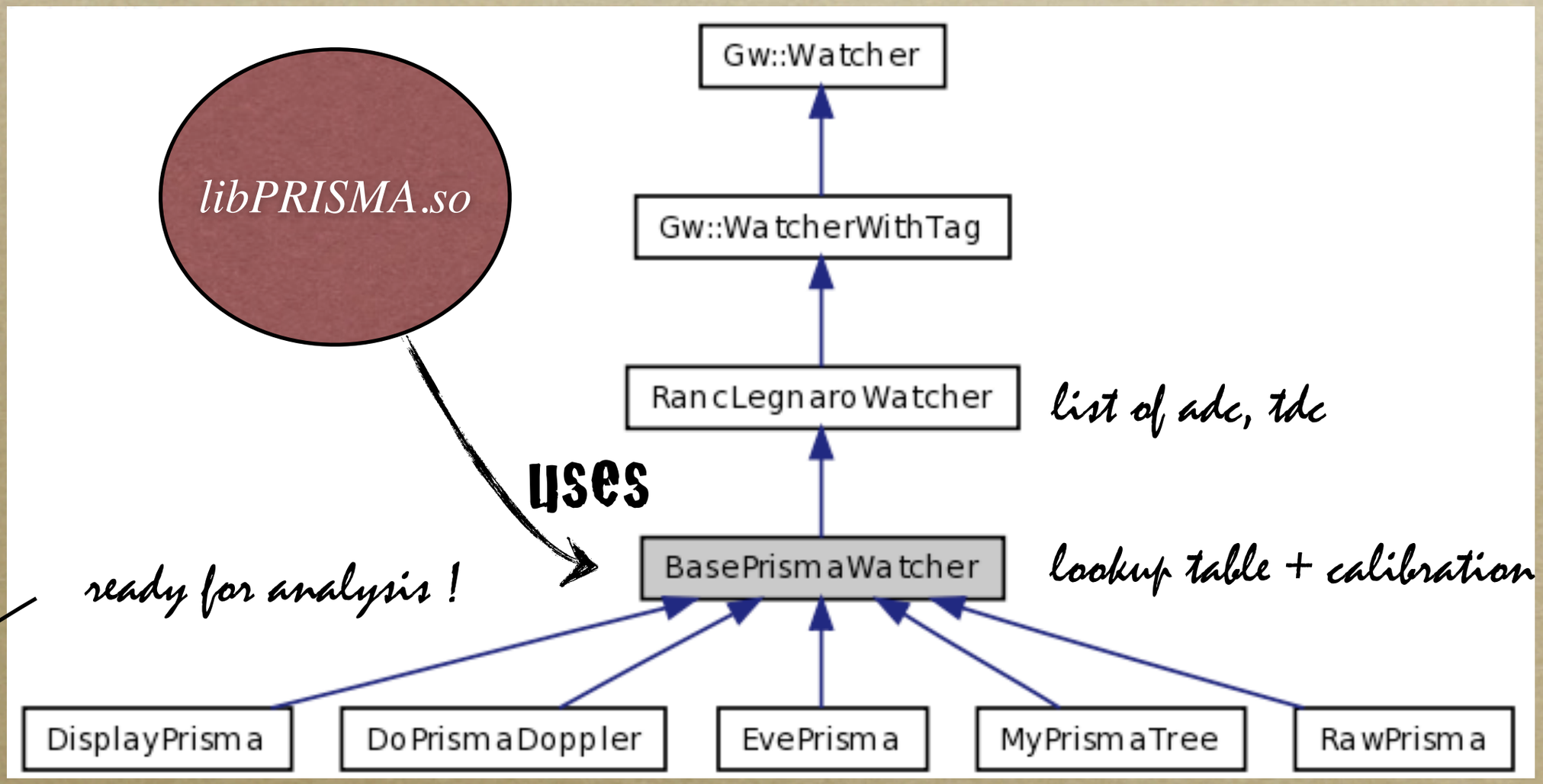
out of merger

Online Watchers[Gw]



Watchers[Gw]

Inheritance : base class, service for daughter classes



Watchers[Gw]

constructor : build spectra

exec : fill spectra

```
DisplayPrisma::DisplayPrisma(const char *name, const char *title): BasePrismaWatcher(name, title)
{
   IcABvsABCD       = new TH2F("Ic_ABvsABCD", "E_DeltaE_AB", 1024,0,12000,2048,0,8000);
   AddToPool(IcABvsABCD);
   IcAvsABCD        = new TH2F("Ic_AvsABCD", "E_DeltaE_A", 1024,0,12000,2048,0,8000);
   AddToPool(IcAvsABCD);
   TOF_D            = new TH2F("ToF(ns)vsX_FP(mm)", "ToFvsx_FP", 1024,0,1024,4096,0,4096);
   AddToPool(TOF_D);
...
}

void DisplayPrisma::Exec(Option_t *option)
{
   const double  c_light = 29.97295 * cm/ns;

   // fill the array of floats and compute the PRISMA physics
   SetLastError(0u);
   BasePrismaWatcher::Exec(option);
   if ( GetLastError() > 0 )
          return;

   if ( fPM->length() > 0. )  {

          Vector3D Vel = fPM->velocity()/c_light;

          IcABvsABCD    -> Fill(fPM->ic_energy(),fPM->ic_energy_AB_DE());
          TOF_D         -> Fill(fPM->x_fp() /mm, 10*fPM->tof()/ns);
          PRISMA_MCPcal -> Fill(fPM->mcp_x()/mm , fPM->mcp_y()/mm) ;
          PRISMA_X_FP   -> Fill( fPM->x_fp() /mm) ;
          PRISMA_Vel    -> Fill( Vel.rho() ) ;
          Range_Energy->Fill(fPM->range(), fPM->ic_energy()) ;
          Vel_Theta->Fill(10.*fPM->theta_c()/degree, Vel.rho() );
          a_over_q->Fill(fPM->x_fp()/mm, fPM->a_over_q_uncal());
          Energy_RBeta->Fill(fPM->r_beta(),fPM->ic_energy());
          PRISMA_Mass->Fill(fPM->Mass());
   }
}
```

just to allow global actions (zero, save, delete ...)

Overview

Basic elements

no standard analysis

➥ *always new spectrum, cuts*

Improvement of existing actors

New actors (GSI) !

What already exists

How to extend

Why.?

Overview

New Watcher

How to extend

About replay

New Actor

Add a new Watcher

Question : on what

➥ *give the*

Ex : LaBr3 have be

➥ *ancillary data*

➥ *base class is R*

➥ *create a class*

lookup table + calibration

➥ *create a class for your analysis*

- *built spectra*
- *built TTree*

```
void MyLLaBr3Tree::Exec(Option_t *option)
{
   if ( gDebug > 3 )
           printf("MyLLaBr3Tree::Exec is called \n");

   // fill the array of floats and compute the PRISMA physics
   BaseLLaBr3Watcher::Exec(option);

        fNbLaBr3 = 0;
   for (Int_t i = 0u; i < fNumberOfModules; i++) {
/*         if ( !fLaBr3[i].has_fired ) {
                   continue;
           } */

                fWhichLaBr3[fNbLaBr3] = i;
                fELaBr3[fNbLaBr3] = fLaBr3[i].E;
                fTLaBr3[fNbLaBr3] = fLaBr3[i].T;

                fNbLaBr3++;
   }

   // fill the tree only if this is the owner of the tree ... otherwise by
TreeMaster
   FillTree();
}
```

Overview

New Watcher

How to extend

About replay

New Actor

Add a new Filter

As a filter, it inherits from NarvalFilter

(or other inheriting from it)

You need to implement the :

process_config method

process_initialise method

read .conf file

set the trigger

SetInput/Process/SetOuput method

Many examples + simple tracking soon in Gw

Overview

New Watcher

How to extend

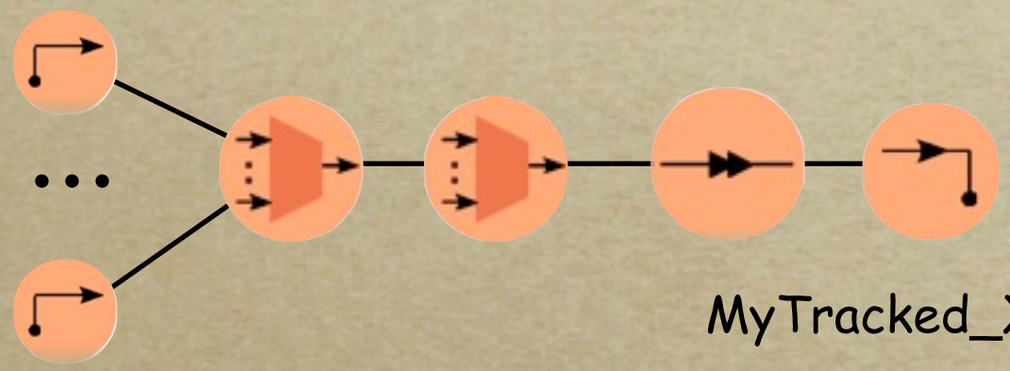
About replay

New Actor

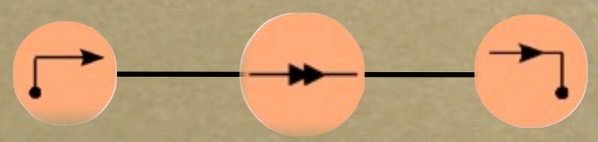
Replay Tracking

There are several possibilities
What is needed is PSA Hits

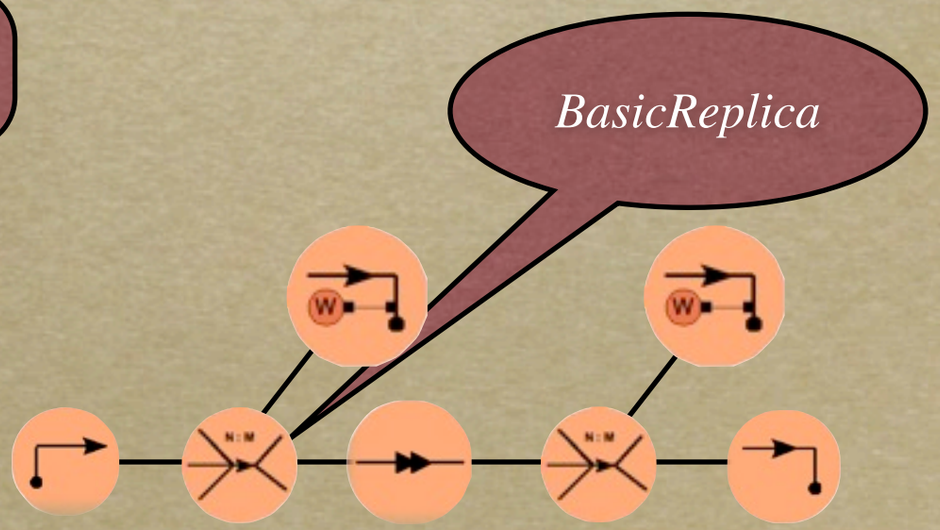
❶



❷



consume data:tracked, produce data:tracked



About Replay

It depends from what to what !

Tracking from hits (precedent slide) : femul, gw
Pb of Event builder / merger hits ➙ end : femul

From traces
one detector : femul, gw
all of them : Narval or femul (slow ...)

Another ('future') solution : the GRID

tests have been done but not yet fully available for the community

About Replay

GRID : resources (CPU/disk) available everywhere

backup

process

mainly tests so far :

have show gain in speed/time

still not in production level

code to be adapted

facilities for users to be developed

hope : for the coming year ...

Ti
(Legn

er2-3
asbourg)

er2-3
alencia)

(IPN Lyon)