# VISPA: a Novel Concept for Visual Physics Analysis

**Tatsiana Klimkovich** for the **VISPA** group

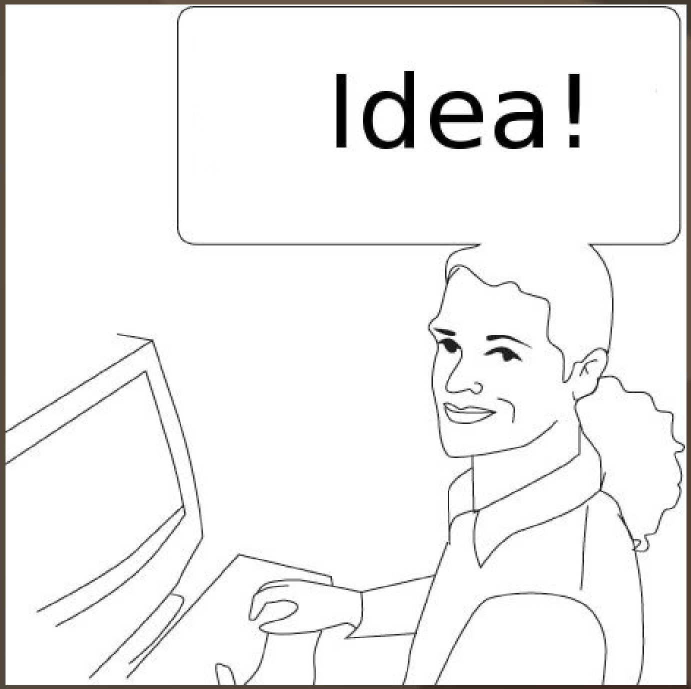
(O.Actis, M.Erdmann, R.Fischer, A.Hinzmann, M.Kirsch, G.Müller, M.Plum, J.Steggemann)

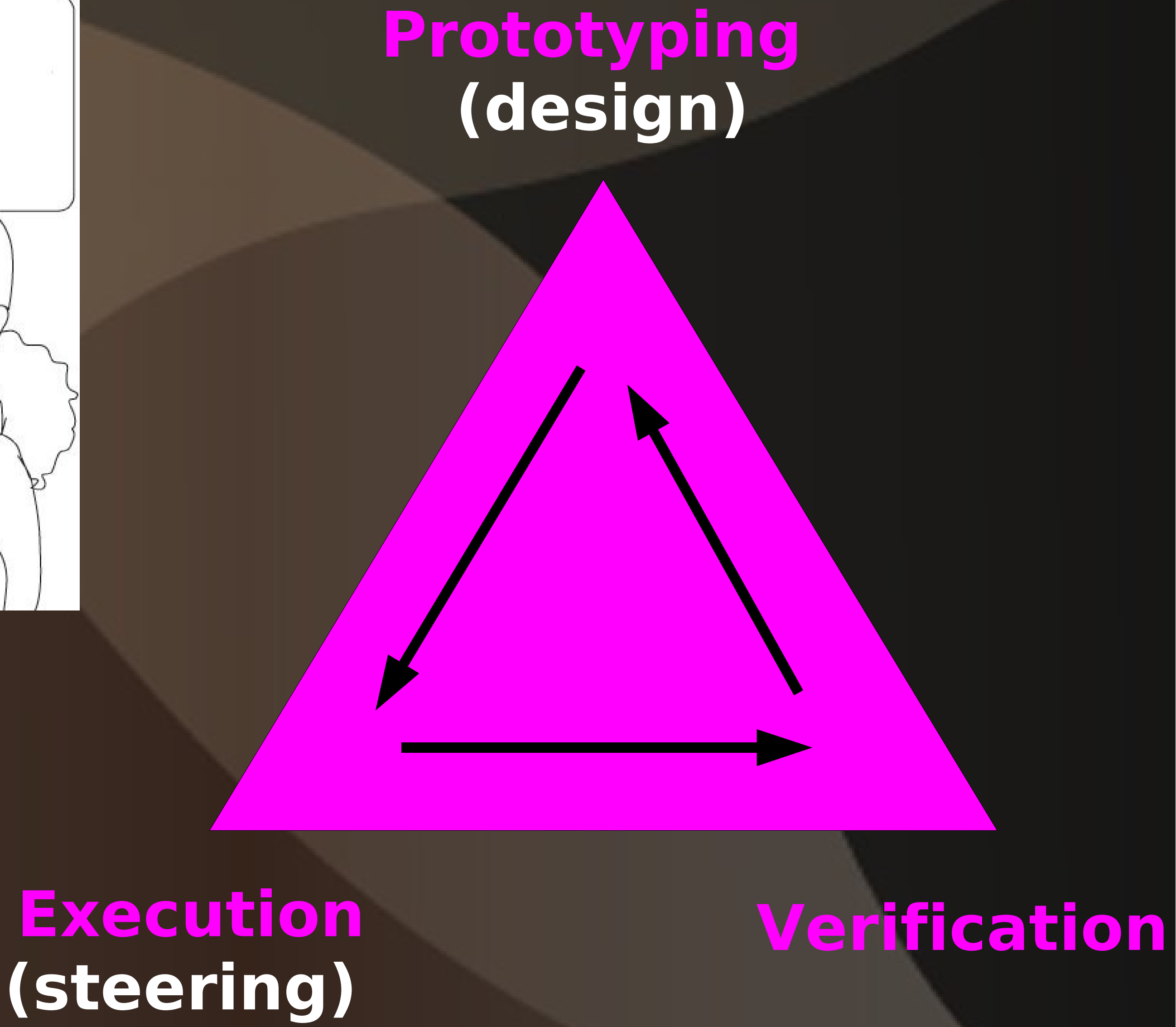
ACAT2008, Erice, November 2008

# Contents

- Physics Analysis in High Energy Physics experiment
- Novel Concept for HEP analyses: VISPA
- Look & feel with analysis example
- VISPA key ingredients:
  - PXL: underlying functionality
  - Python interface to C++ functionality
  - Module steering system
  - Autoprocess: automatic decay chain reconstruction

# High Energy Physics Analysis



# High Energy Physics Analysis

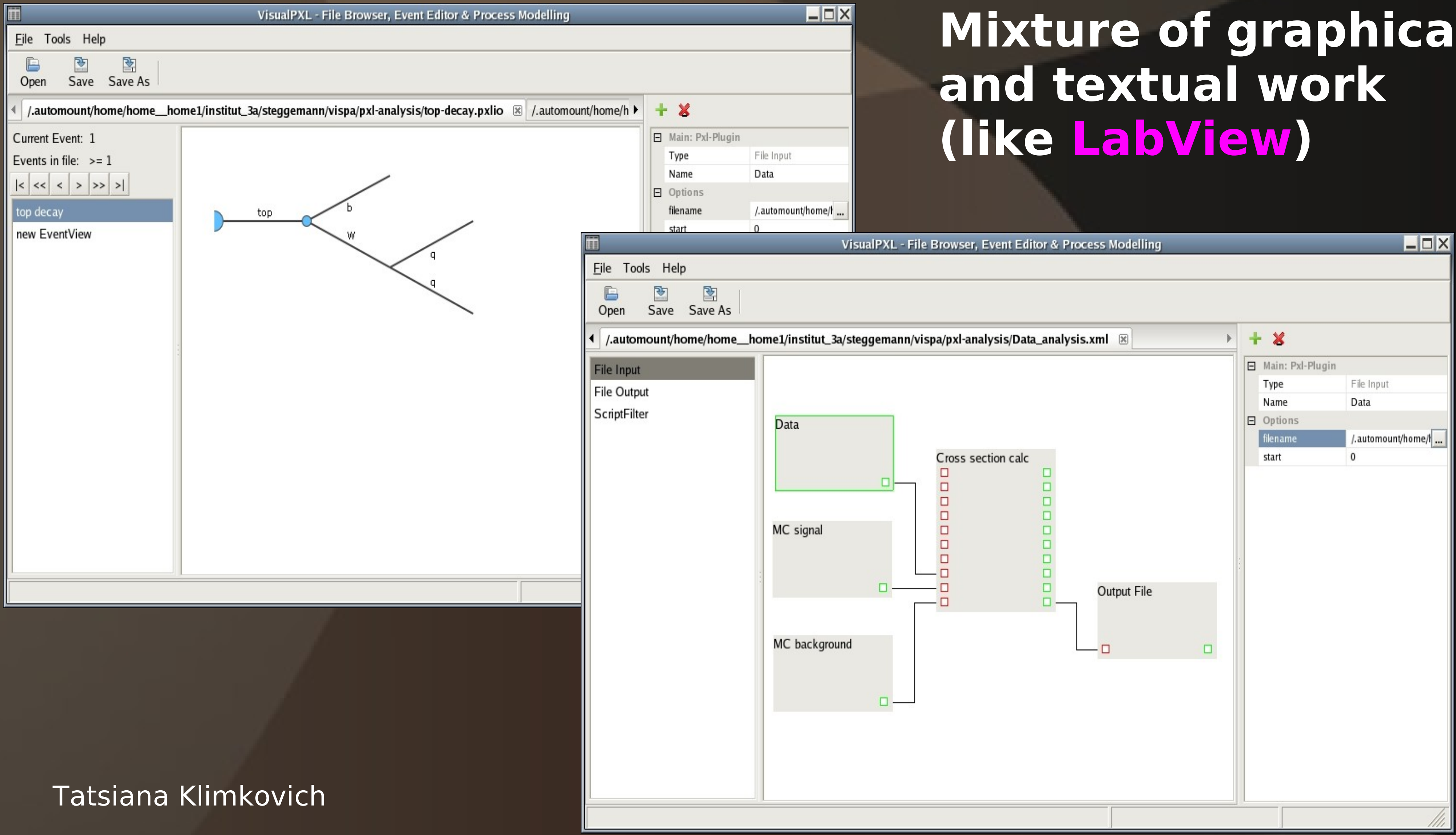
- During last years big achievements have been done in developing analysis software for the experiments
- Experiments have different software frameworks e.g. **H1OO** in H1, **CMSSW** in CMS, **ATHENA** in ATLAS etc.
- On top of them **more analysis specific software** has been developed and used:
  - Requires less time to start
  - The output files are smaller
  - Reduce time for making analysis
  - Allows to perform analysis on the laptop

# The wish list of the analyser

- To have an easy way to develop analysis
- To start fast
- To dedicate minimal time for learning
- To have modular structure of the software
- To have many reusable components
- To have small summary data sets (ntuples)
- To have clear picture of what you are doing
- To transmit your knowledge to other people

# VISPA: Visual Physics Analysis

## Novel Concept of making physics analysis




**Mixture of graphical and textual work (like LabView)**

Tatsiana Klimkovich

# VISPA: Main Features

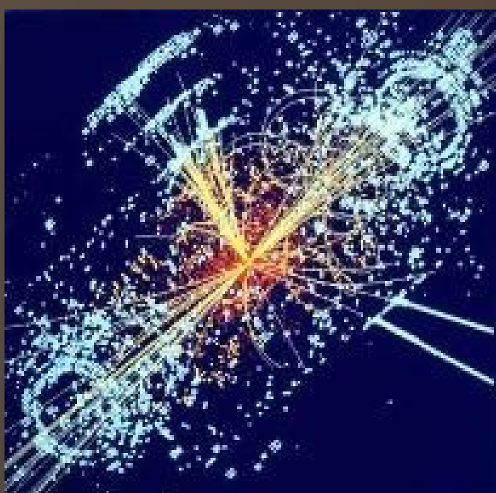
**Development environment for HEP analyses**

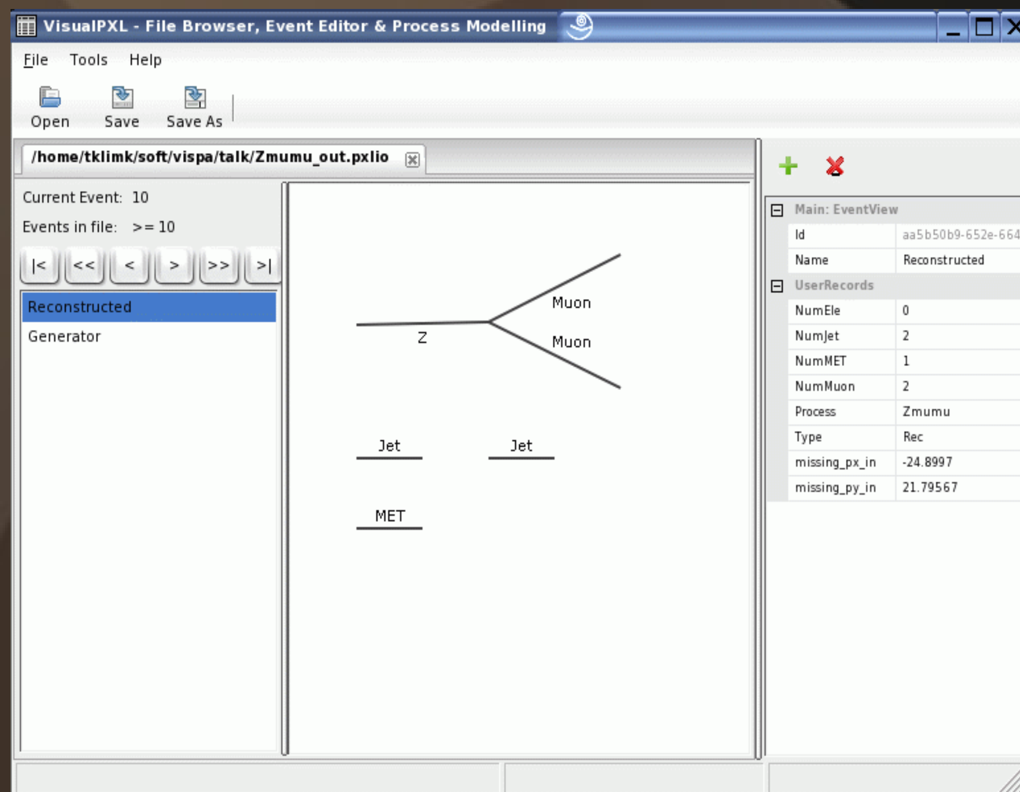
**(first prototype)**

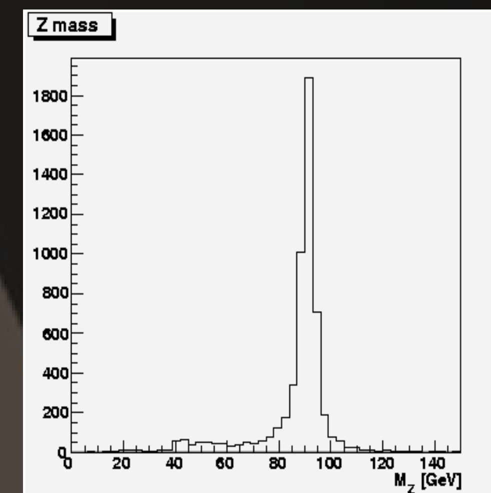
- **Combines graphical and textual steering**
- **Module steering**
- **Works for any experiment**

# Structure of Physics Analysis

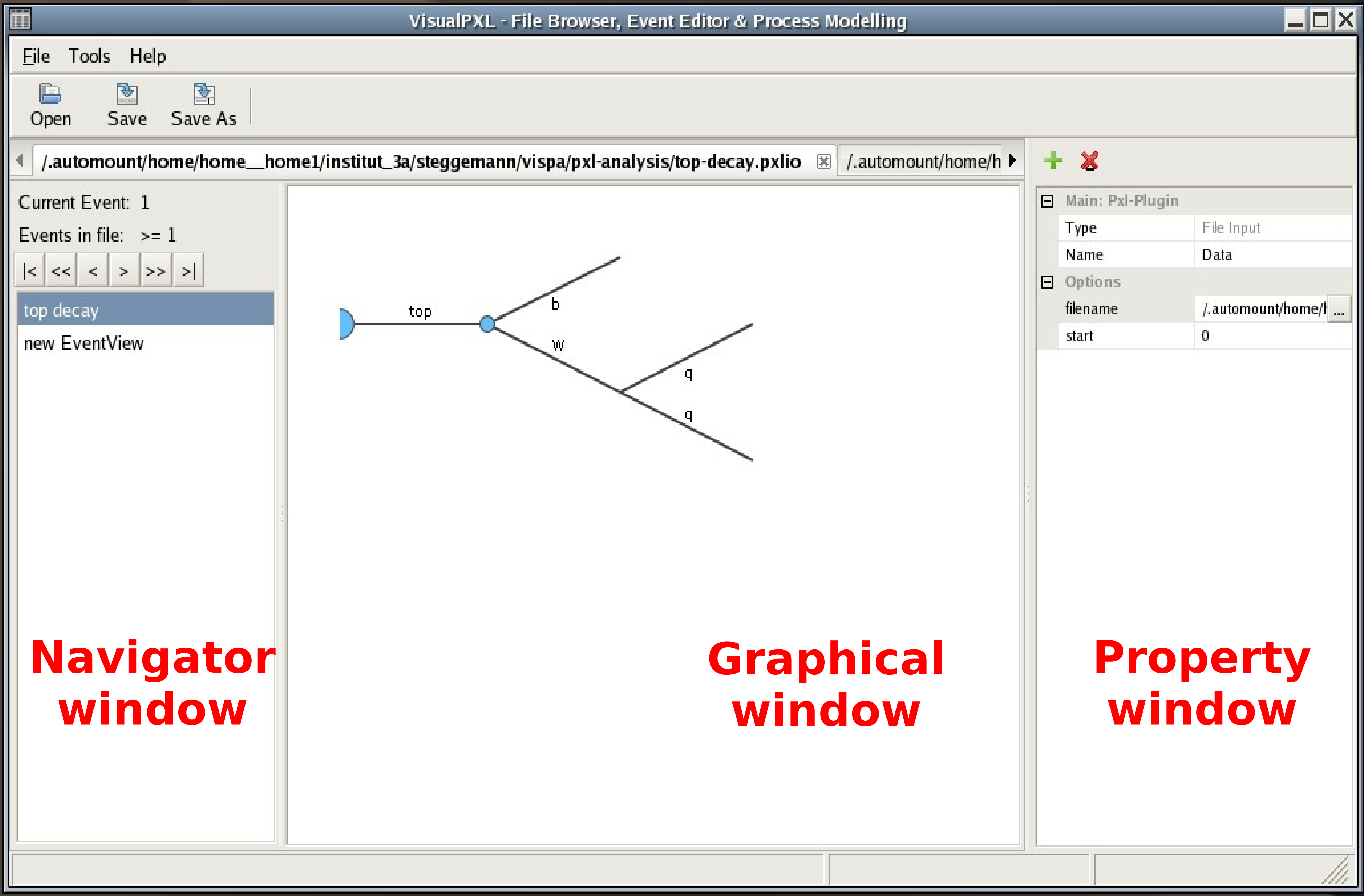
Data input → Data selection → Advanced analysis → Histograms





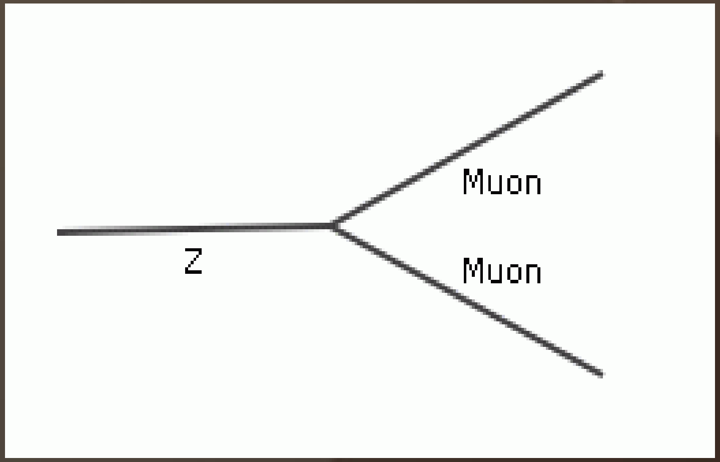
# Multipurpose Window



# Closer consideration:

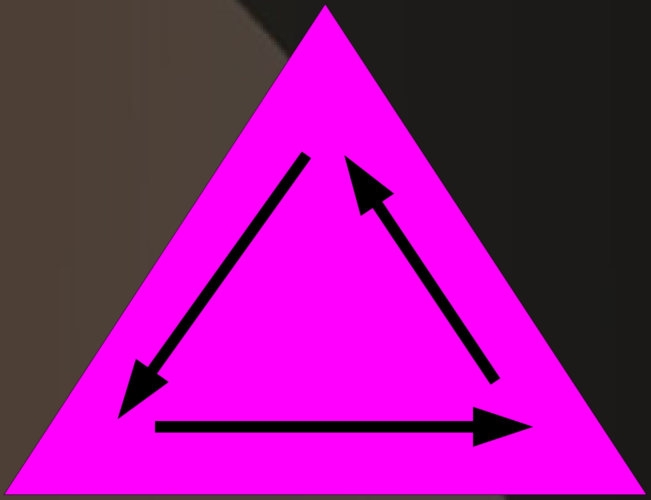
**Very simple example analysis**

**Z --> mu mu**



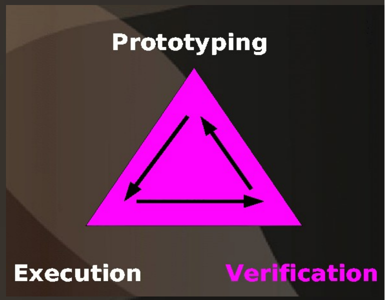

**Sample at LHC energies**

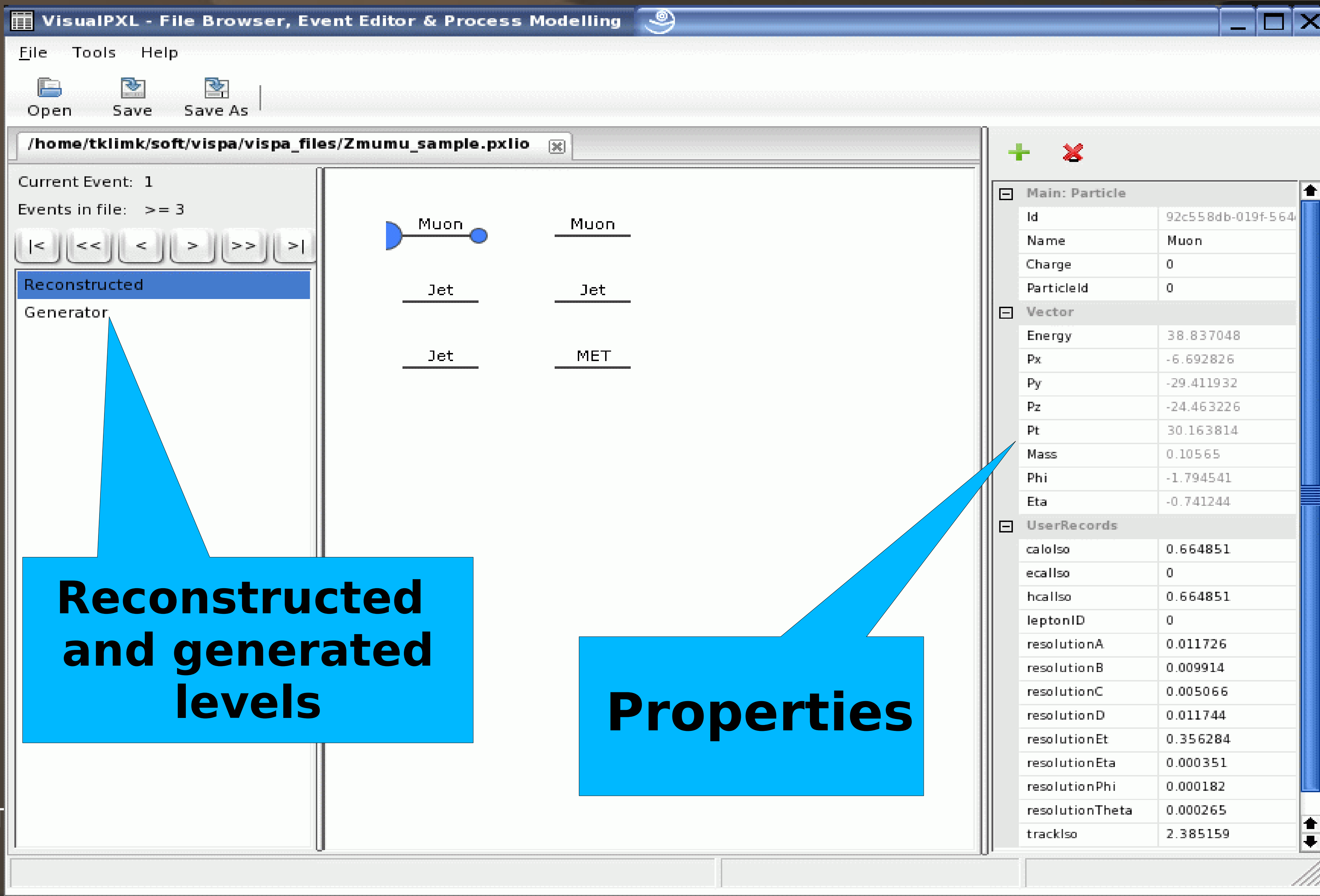
**Prototyping**



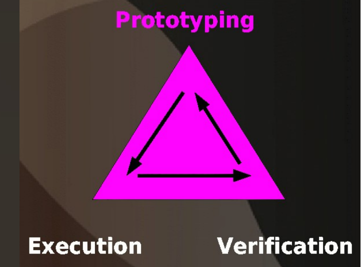
**Execution** **Verification**

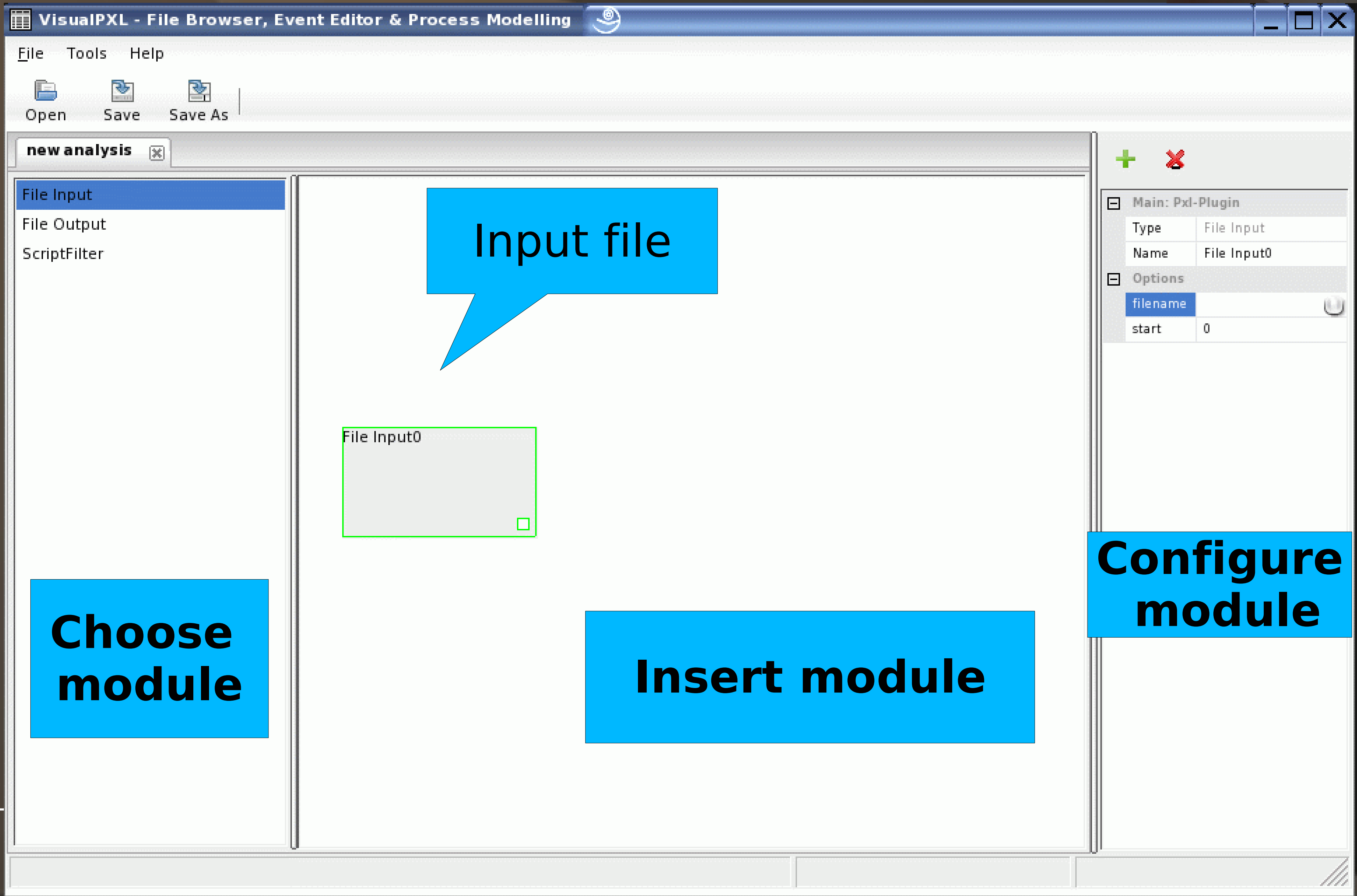
# Z boson reconstruction from muons
## First step: inspect an input file



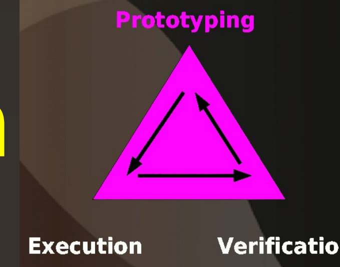



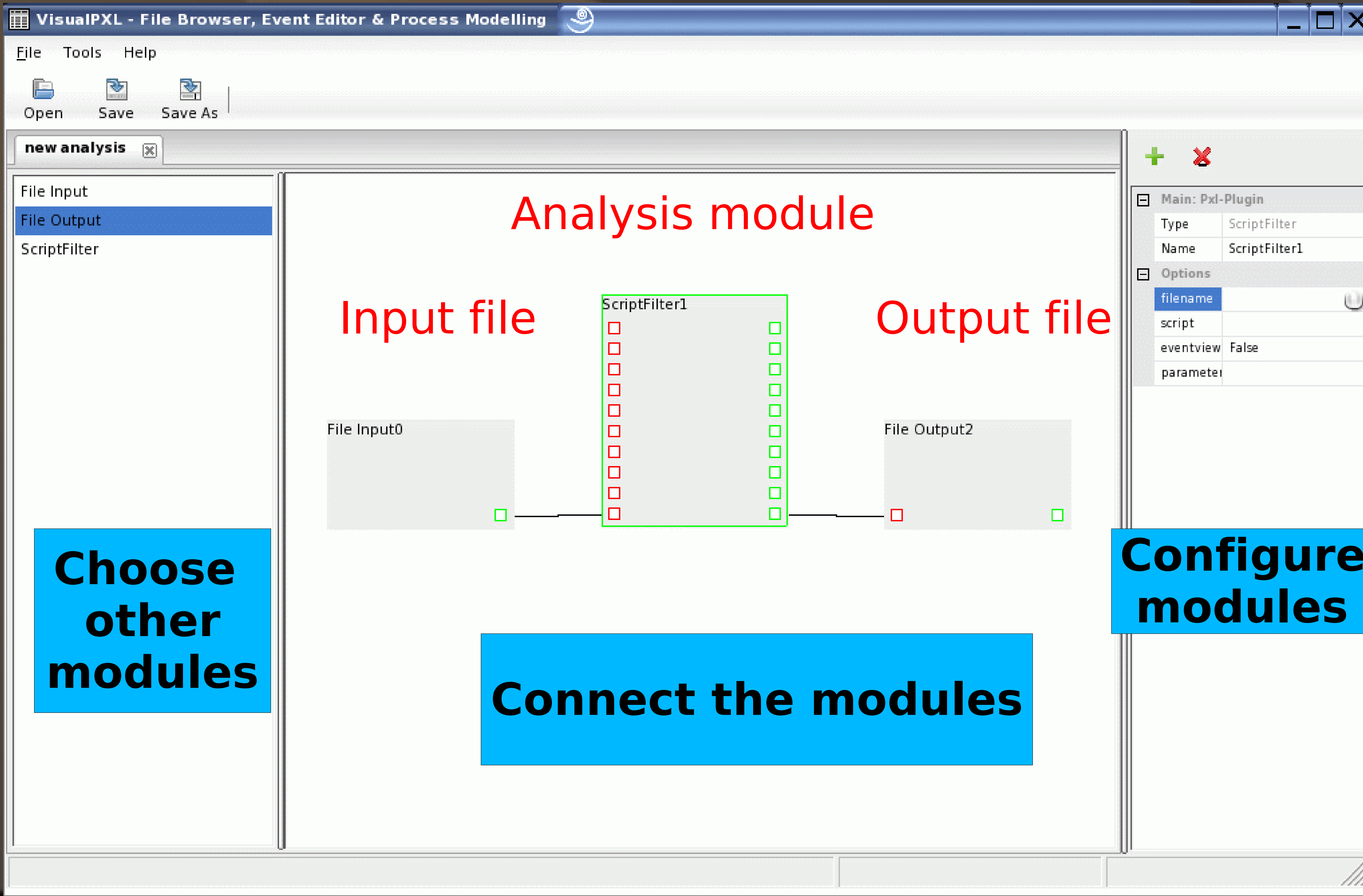
# Use GUI to design analysis



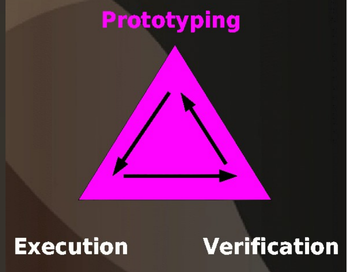



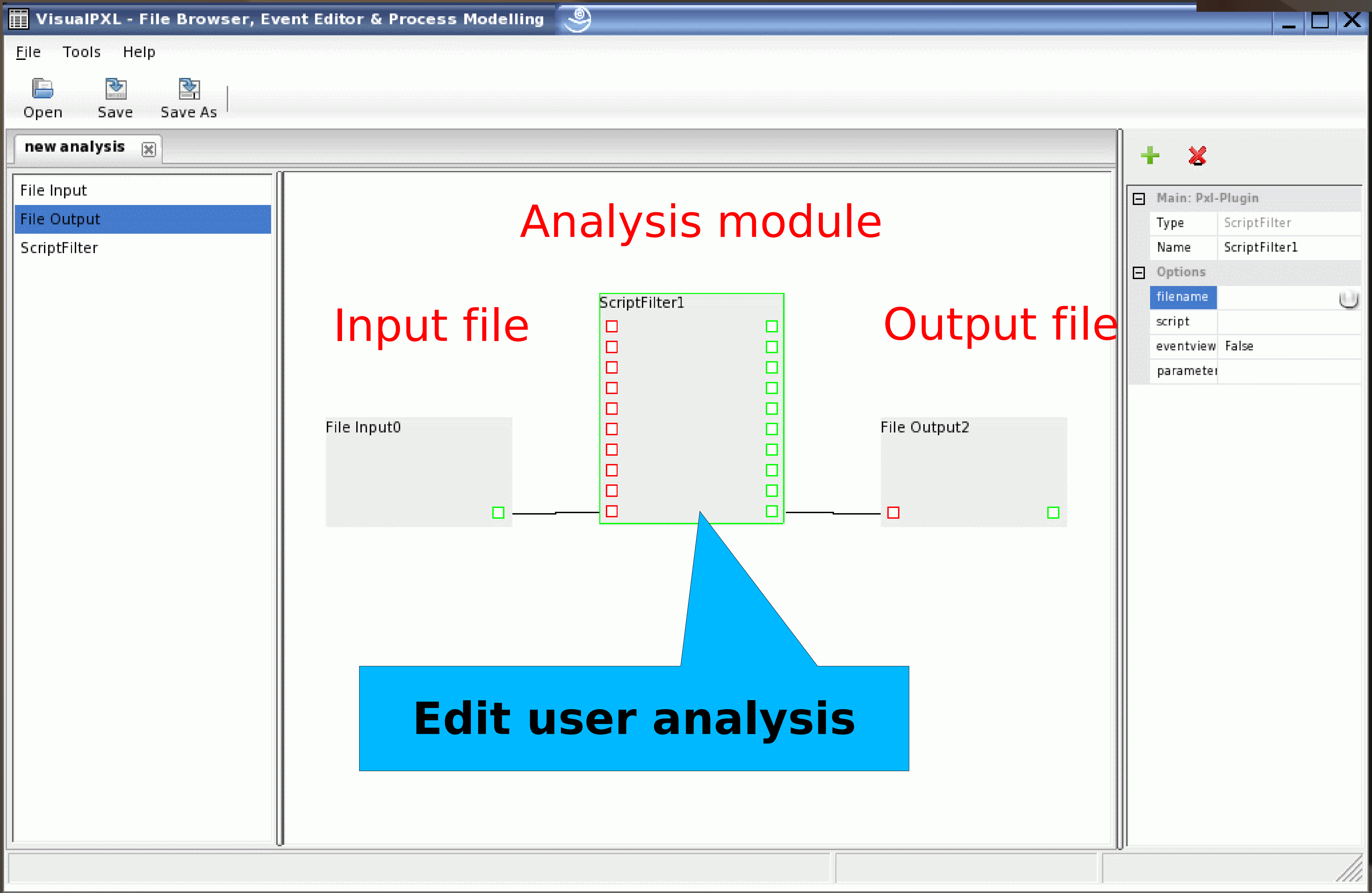
# Complete Analysis Design



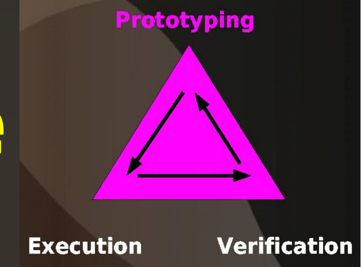



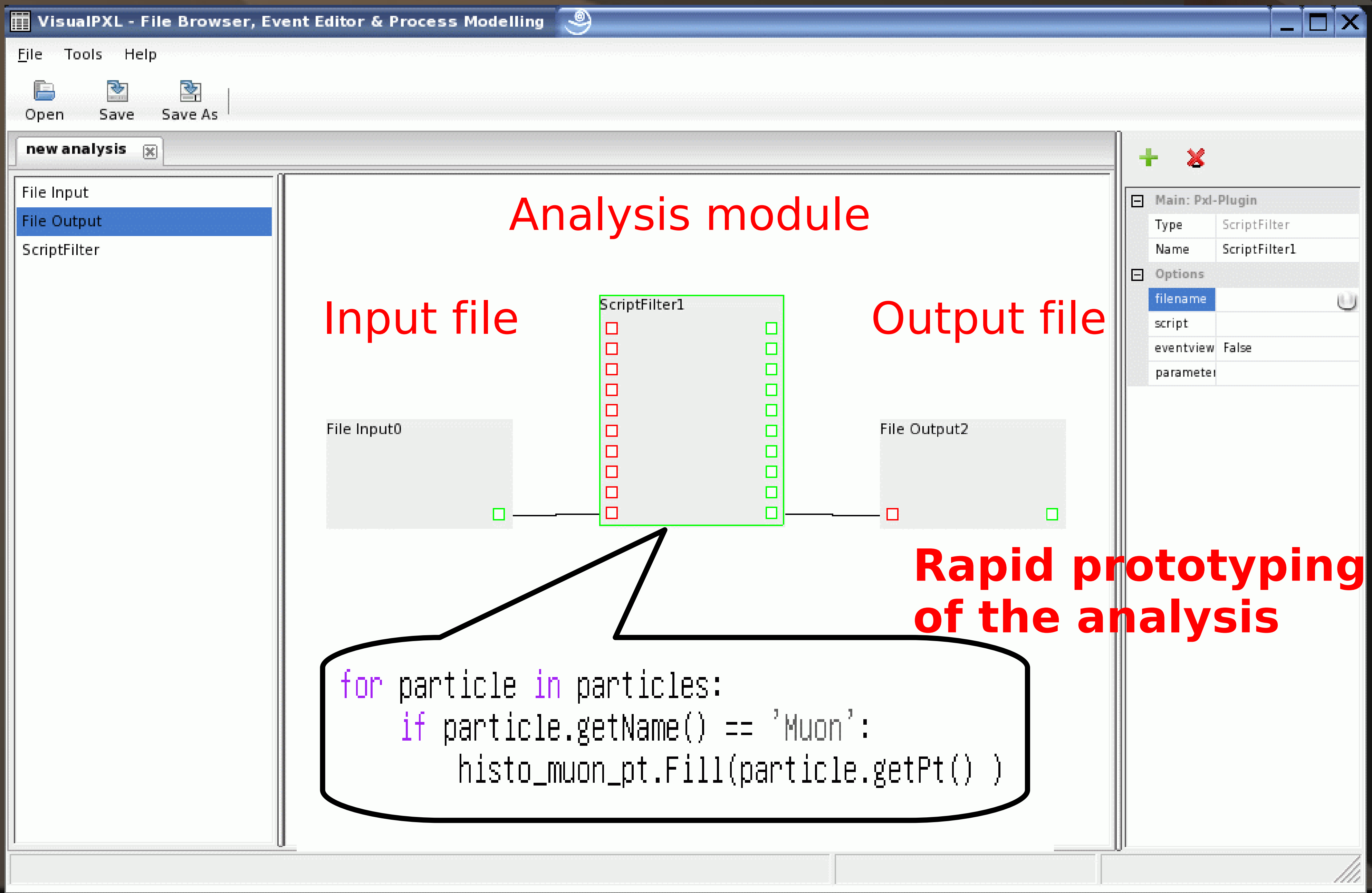
# Create Python Analysis Module



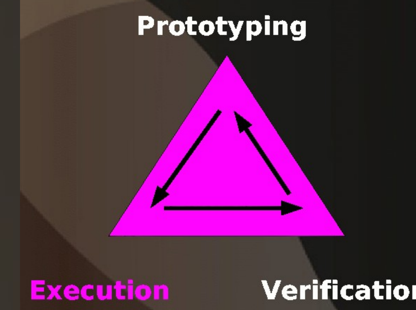



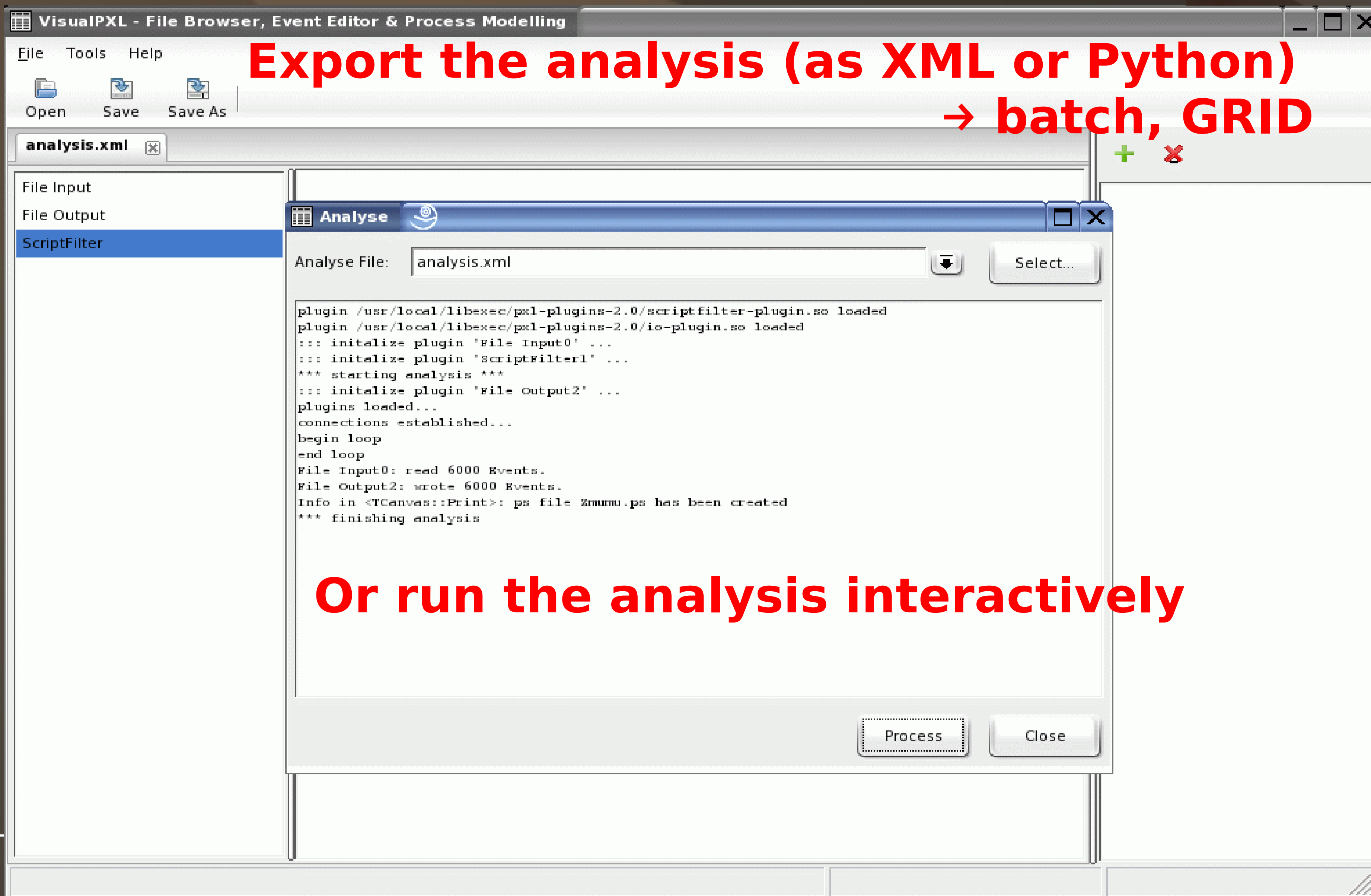
# Create Python Analysis Module



# Run Analysis

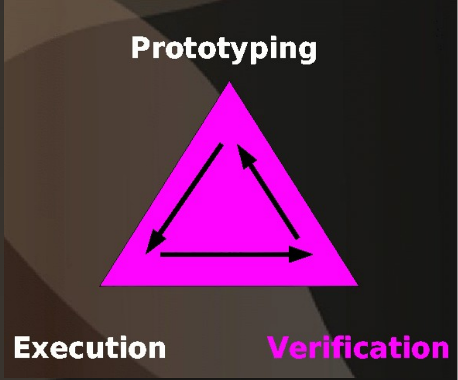


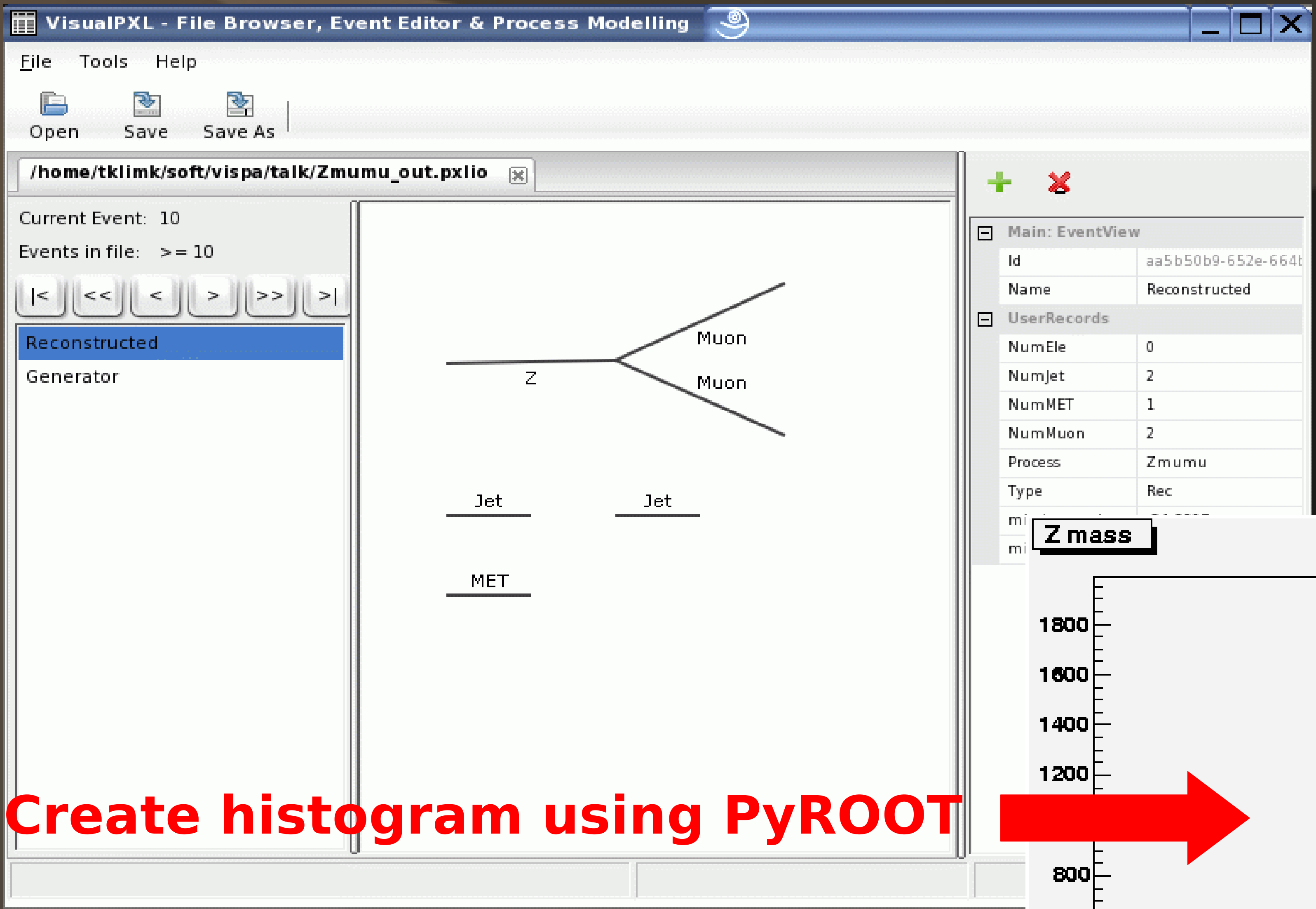




**Export the analysis (as XML or Python) → batch, GRID**

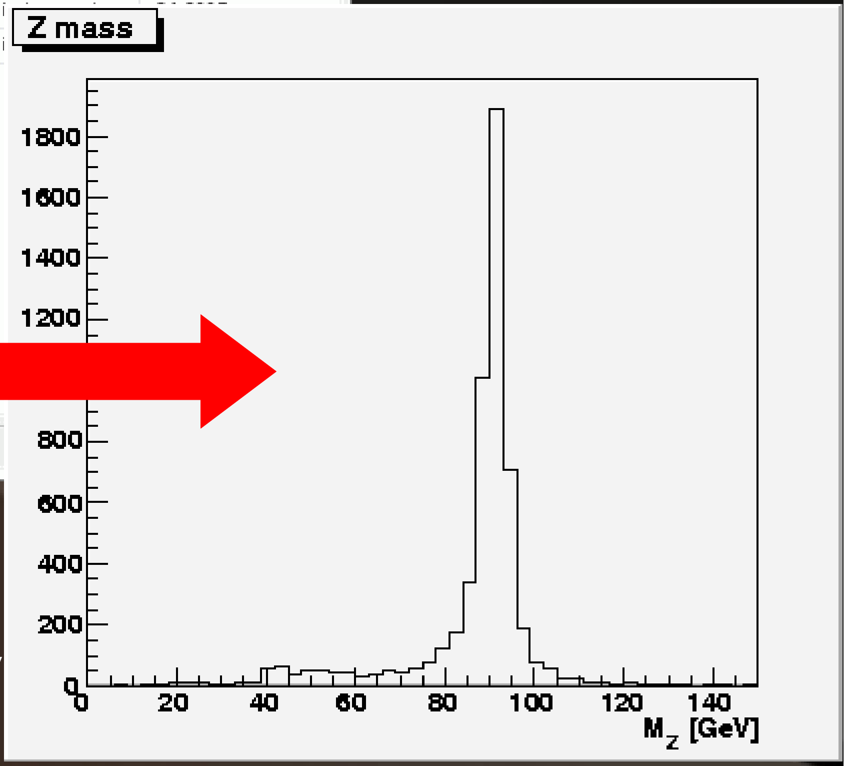
**Or run the analysis interactively**

# Verify Analysis Output



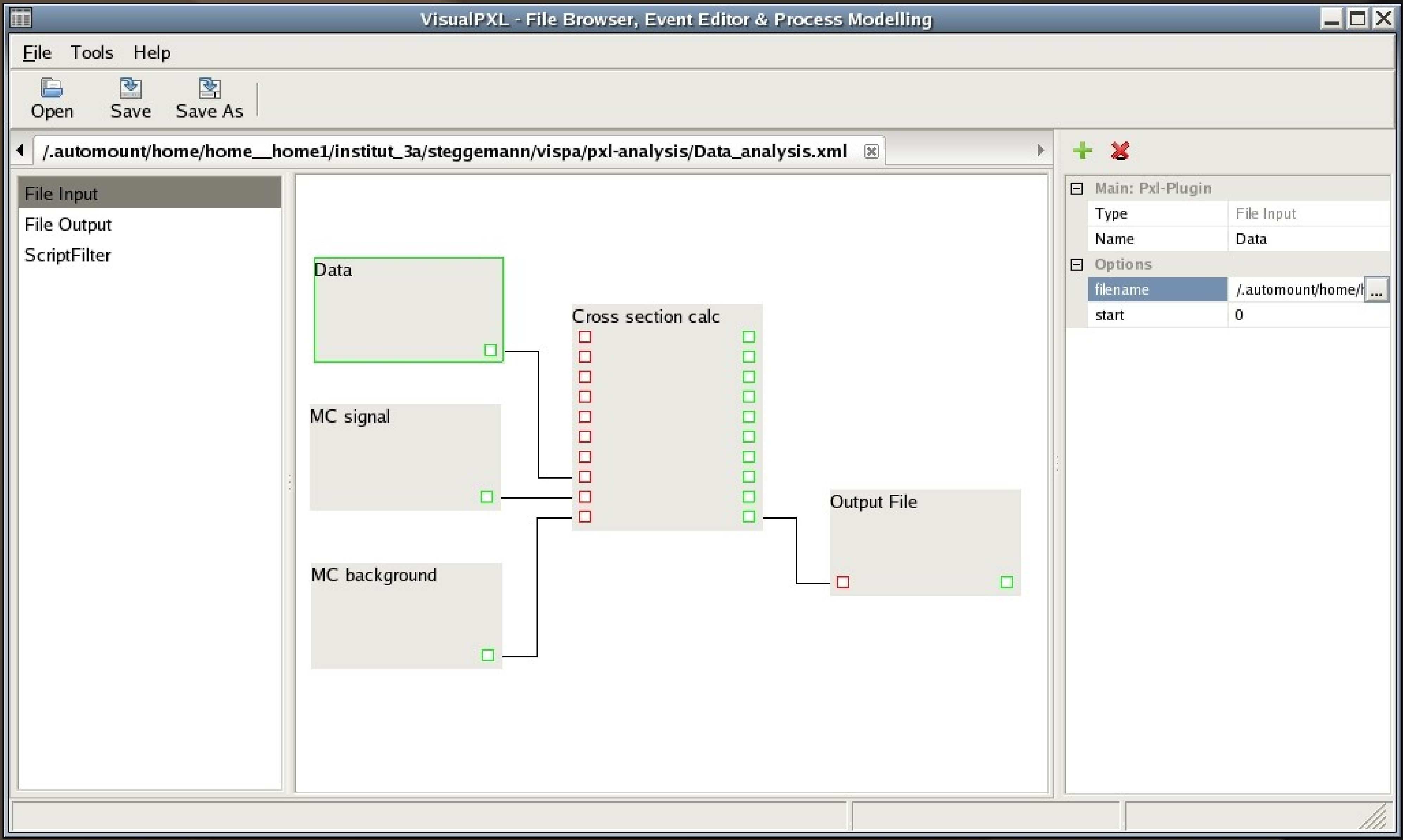



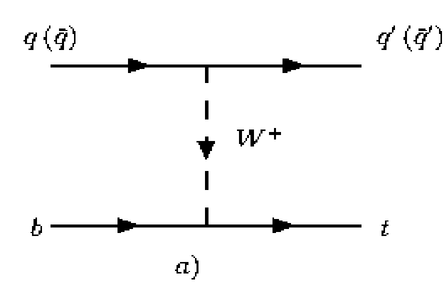

**Create histogram using PyROOT**



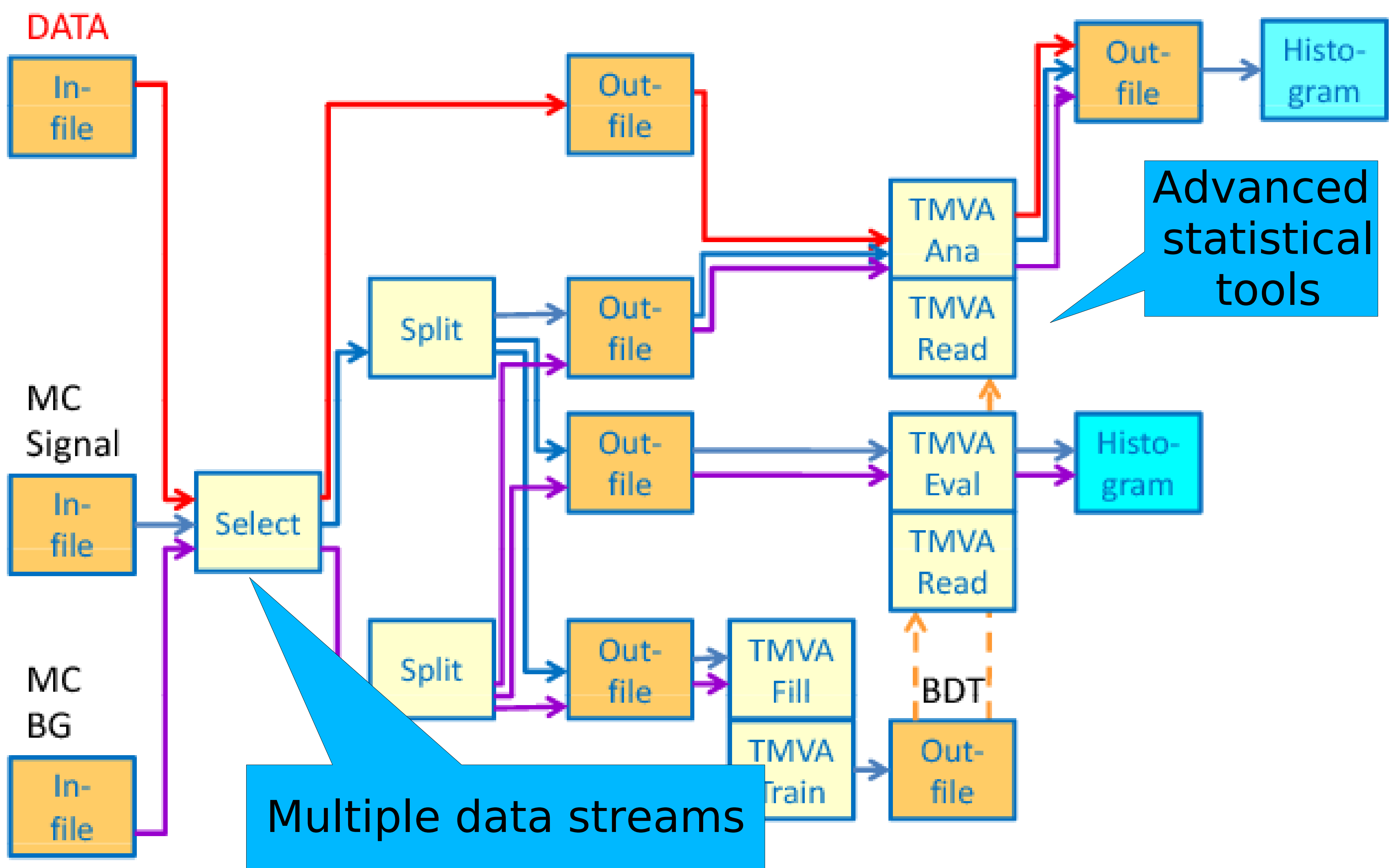

**If needed repeat prototyping, execution, verification**

# More complex analysis



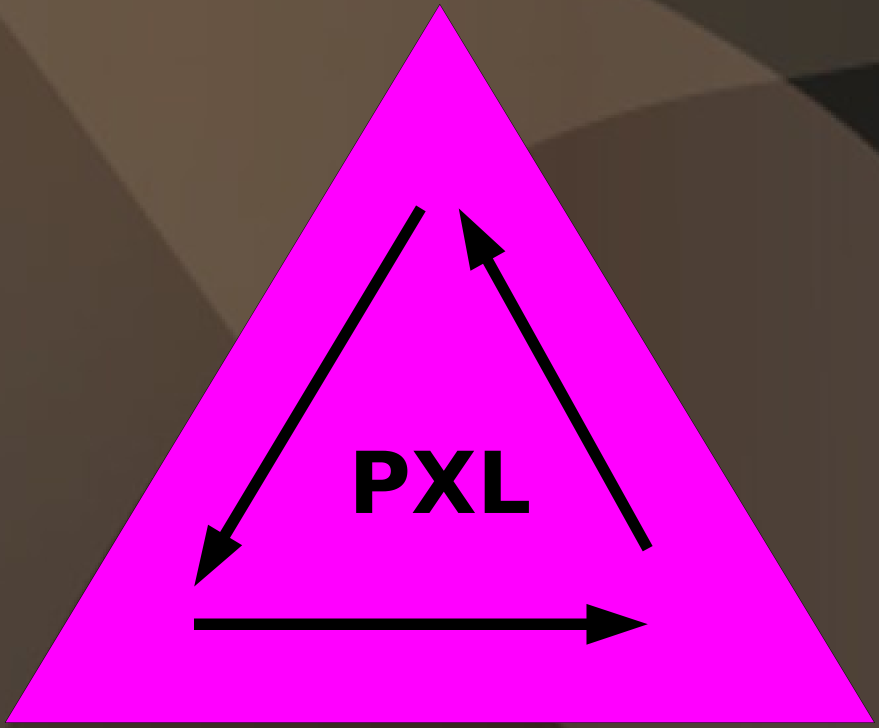



# Single top quark analysis



# Summary: Analysis flow with VISPA

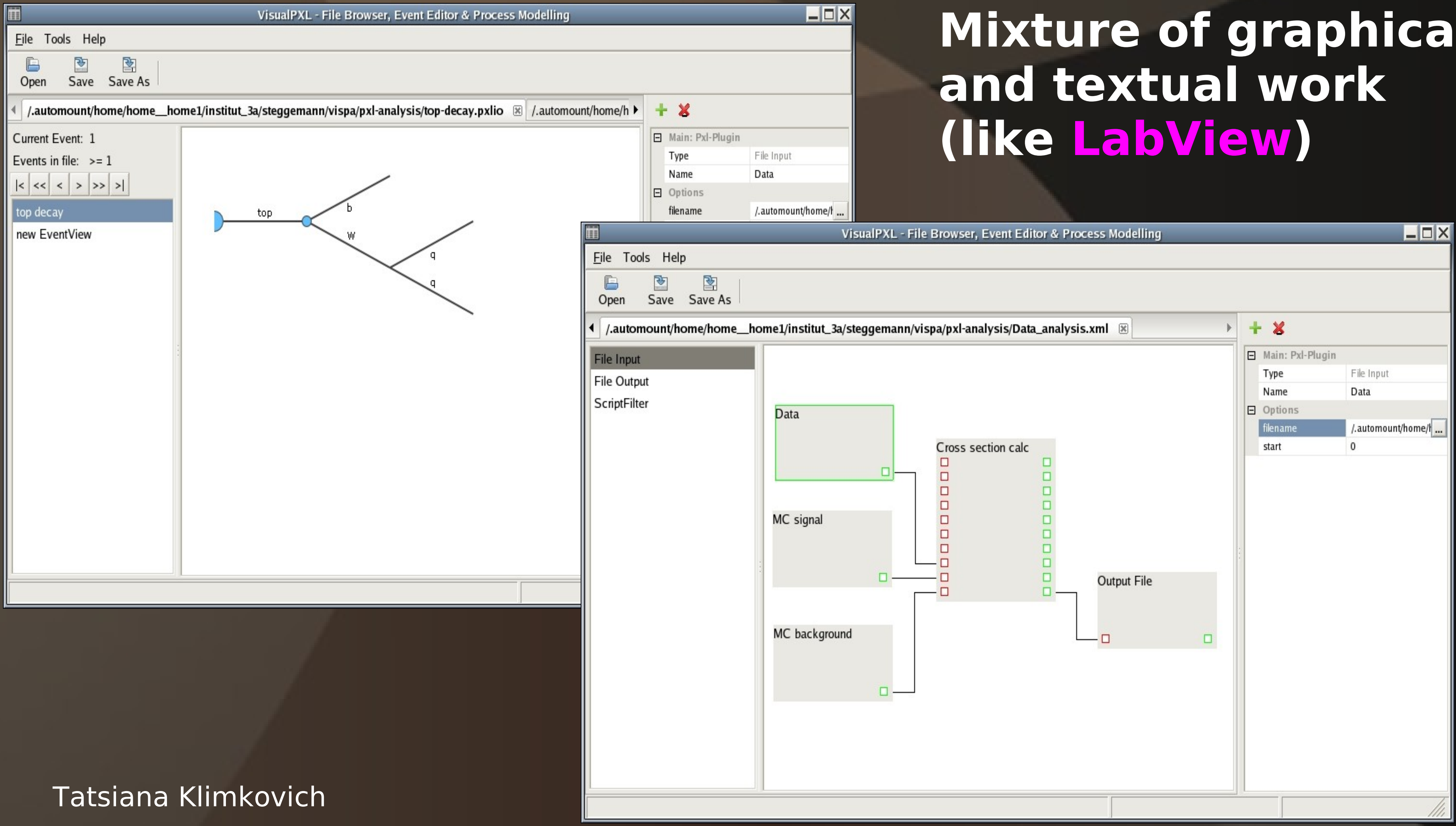
**Prototyping:** Graphical platform
Analysis modules: Python or C++




**Execution**
(XML or Python steering)
(interactive or batch)

**Verification**
(Event Browser)
(ROOT histograms)

# VISPA: till now was look & feel
# Now: to the ingredients




**Mixture of graphical and textual work (like LabView)**

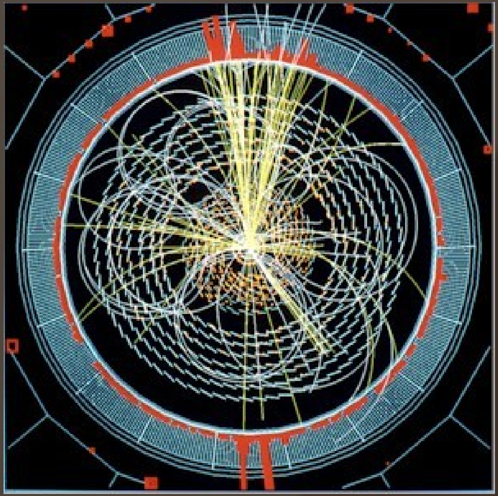
Tatsiana Klimkovich

# VISPA Key Components

- **PXL: C++ package providing underlying functionality**
- **PyPXL: Python interface to PXL**
- **Module steering system**
- **Autoprocess: automatic decay chain reconstruction**

# PXL (Physics eXtension Library)

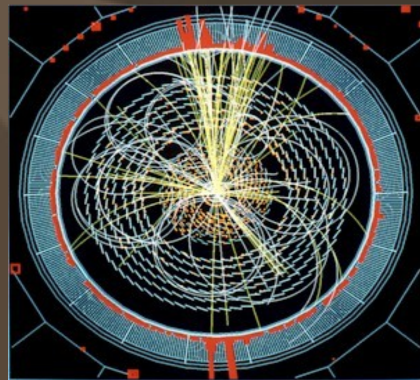
- C++ toolkit for high-level physics analysis
- Provides underlying functionality for Visual Physics Analysis (VISPA)
- Version 2.0 (2008)
- Successor of PAX (Physics Analysis Expert) (2002-2007)

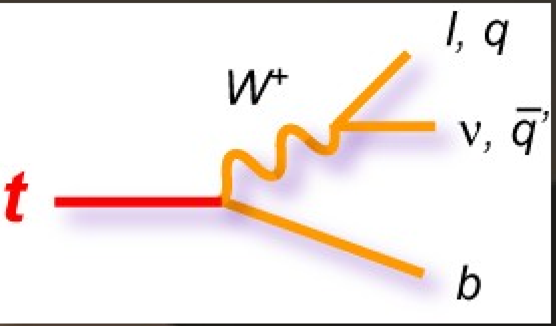


Experiment Software Framework → PXL → ROOT

ROOT
An Object-Oriented Data Analysis Framework

# PXL key components: Event Container






- Particles (pxl::Particle)
- Vertices (pxl::Vertex)
- Collisions (pxl::Collision)
- User data (pxl::UserRecord)
- Their **relations** and **roles**

**Physics objects**

**Event View**

**pxl::EventViews**

**Event container pxl::Event c**an hold several **pxl::EventView**s

Allows **deep copies** (physics objects with redirected relations, data members, user records)

# PXL key components: UserRecord

- PXL physics object is not only a fourvector
- PXL physics object has also a UserRecord where user data are stored:

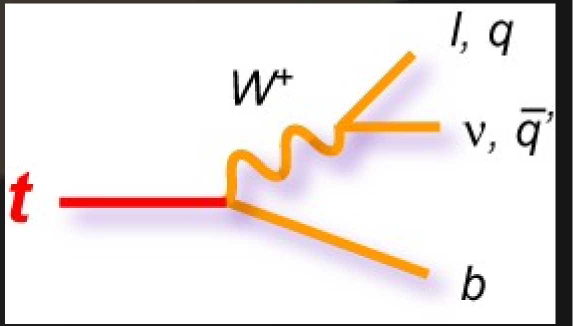
  - these are pairs of names and all basic C++ types (int, double, string, ...)

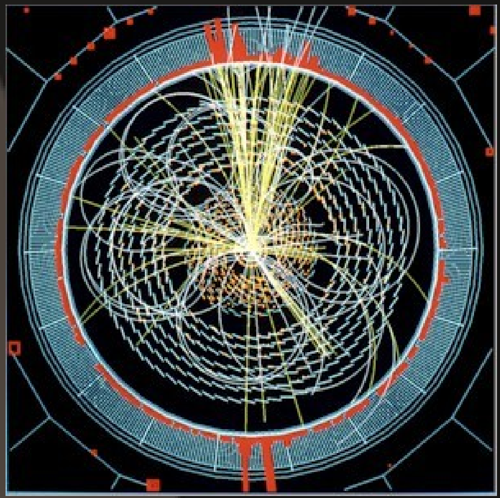
  - it can be e.g. data from the condition databases etc.

- Deploys Copy-On-Write mechanism
- Flexible and simple extension of objects

# PXL key components: Relation Management

- **“Hard” relations:**
  - **Mother, daughter, and flat relations**
  - **Between objects within the same event container**
  - **Safe removal in case of object deletion**



- **“Soft” relations:**
  - **Between any objects**
  - **Have an arbitrary name *(string)***

# PXL key components: Input / Output

- Main class pxl::Serializable
- Fast, Flexible
- Small file size: use ZLIB library for data compression
- Information chunks to structure files
- Each object knows how to stream itself
  - methods “serialize” and “deserialize”
- Simple inclusion of user classes into I/O scheme

# Python interface to PXL

- **Why Python?**
  - **Python code is easy to read**
  - **Less code compared to C++**
  - **Dynamic typing**
  - **Automatic memory management**
  - **Python does not need compilation**
  - **Has an interactive mode for testing**
  - **Object oriented, works on multiple platforms, open source**
- **Use of SWIG for automatic transfer**

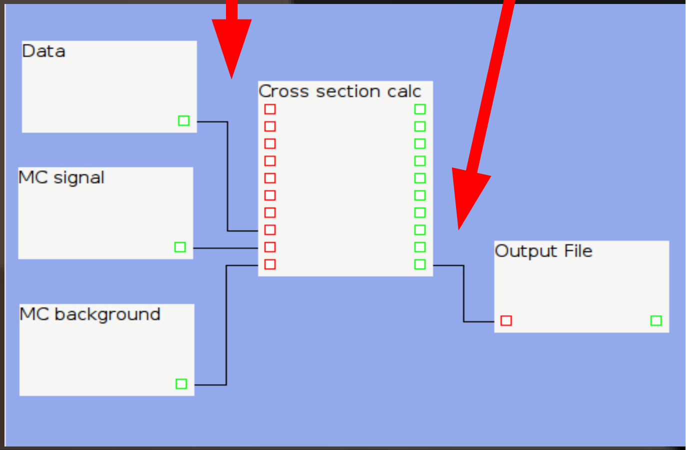
# Python popularity in HEP

- Starts to be more popular for doing physics analyses
- Bender – LHCb Python-based physics analysis application
- Possible to perform analysis with Python in CMS
- Some use in D0

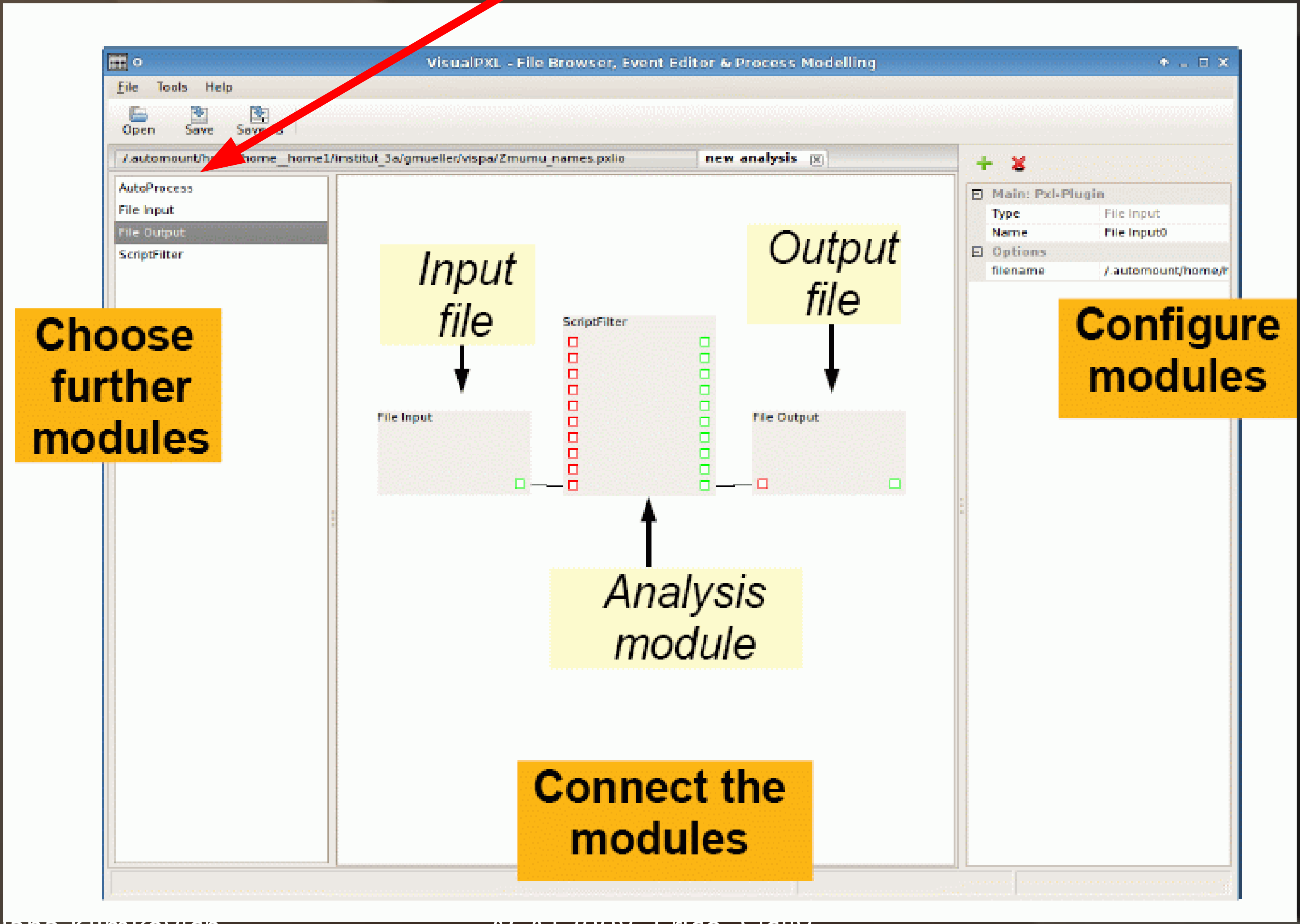
# Module Steering System

- **Data flow**
  - ➔ each module has a number of sources and sinks
  - ➔ interface between modules: PXL event container
- **Modules**
  - ➔ plug-in mechanism
  - ➔ interactive creation of PYTHON modules
- **XML configuration**
  - ➔ exchange format
  - ➔ save and restore any state of the analysis
- **PYTHON configuration**
  - ➔ high flexibility
  - ➔ easy-to-read

**Interface: Event Container**

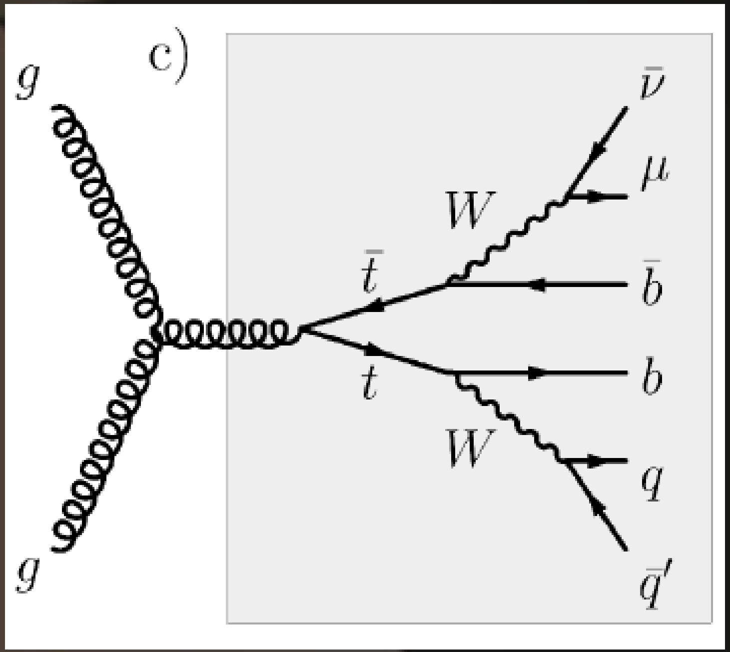


# Autoprocess



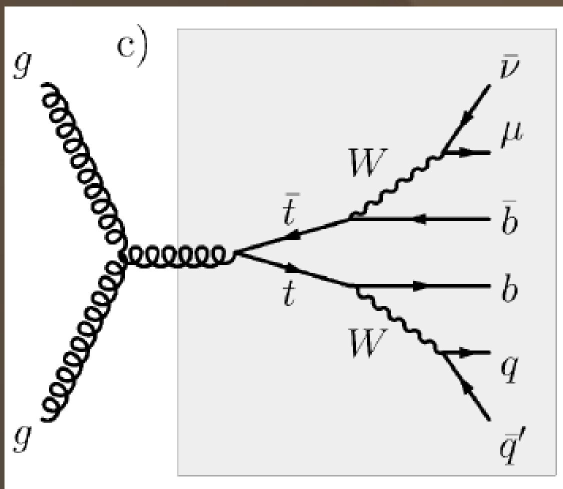
# Autoprocess

- In some physics analysis (Top, Higgs, SUSY) a reconstruction of the whole decay chain often needed
- Several possible configurations need to be built
- **Autoprocess** is an algorithm for automated reconstruction of particle cascades
- Avoid programming reconstruction code for every physics process individually



# Autoprocess

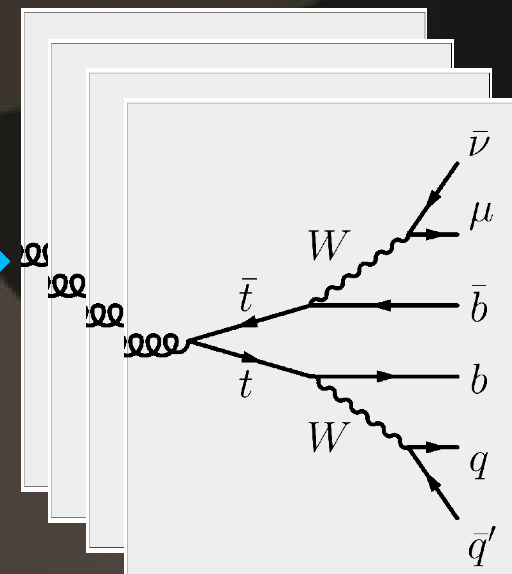
**Steering event container**




**Data event container**

**Jets**
**Muons**
**MET**

**Output event containers**



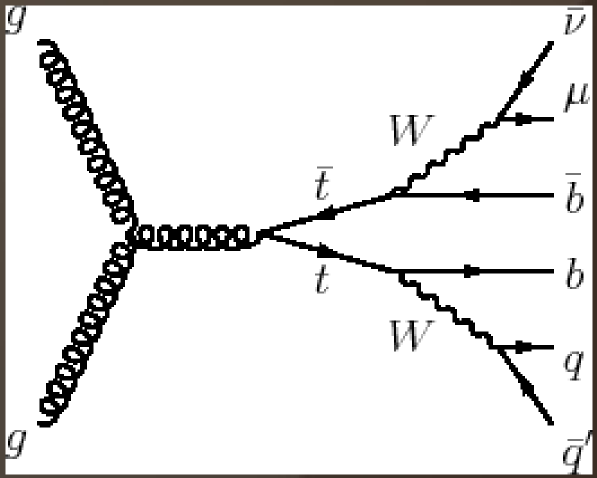

**Autoprocess algorithm**

- **Steering by an event container* holding (part of a) Feynman diagram**
- **Intuitive for physicist**

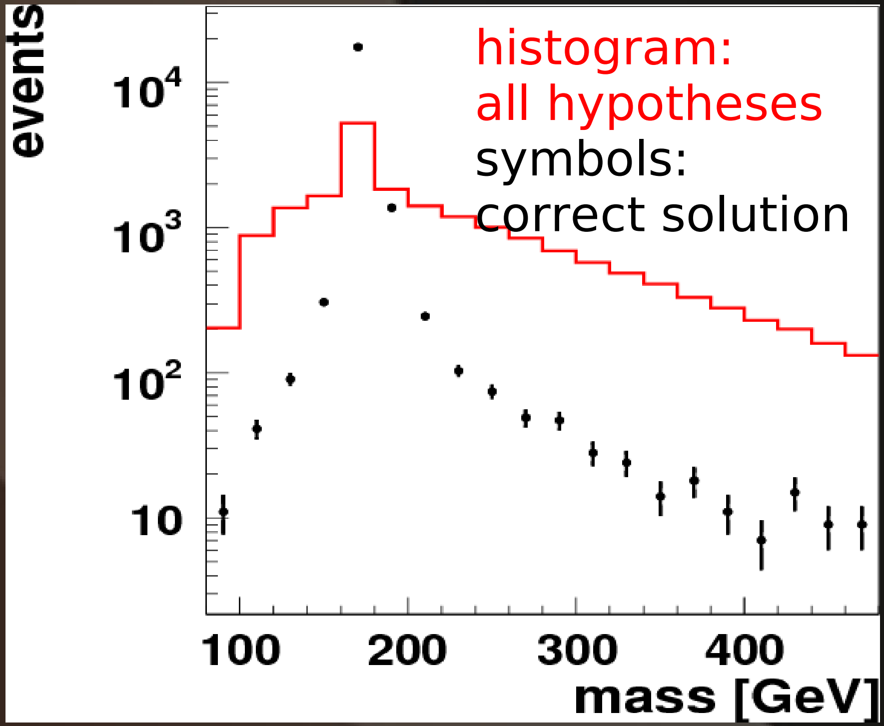
*PXL event container and particles (including relations)

# Autoprocess

- Tool for automatic calculation of all event configurations (hypotheses)
- Well suited for e.g. top physics
- Good performance in **CPU time** and **memory consumption**




24 hypotheses



# Summary

- VISPA is a novel graphical analysis design package for high energy physics analyses
  - Combines visual and textual programming
  - Allows fast prototyping, execution, and verification of an analysis
  - Can work for any experiment
  - First applications for CMS analysis
- Key components:
  - PXL: underlying functionalities for VISPA
  - Module steering system
  - Autoprocess: automatic calculation of particle cascades

# Summary II

- All software is continuously maintained
- Fully documented:

  - Manual for PXL:

  http://pxl.sourceforge.net/manual.pdf

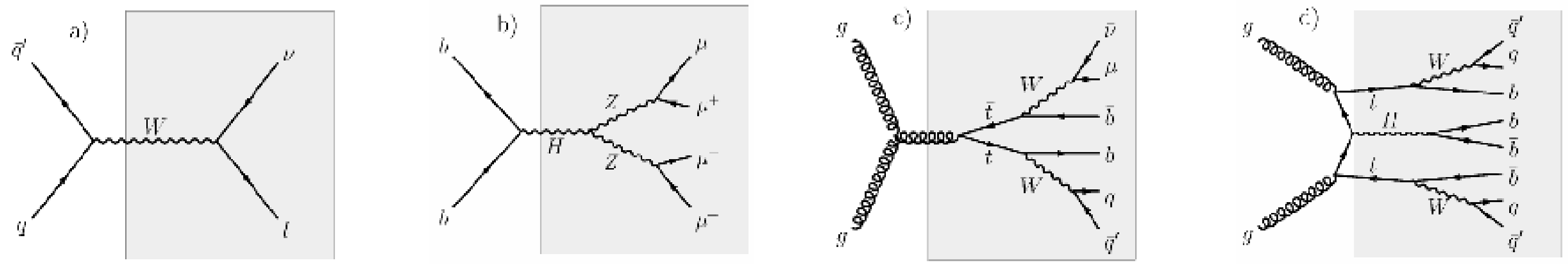
  - Doxygen
- Available online at http://pxl.sourceforge.net
- Look & feel version for MAC is provided
- Publications:

  http://arxiv.org/abs/0810.3609

  http://arxiv.org/abs/0801.1302v2

# Backup

# Autoprocess: performance

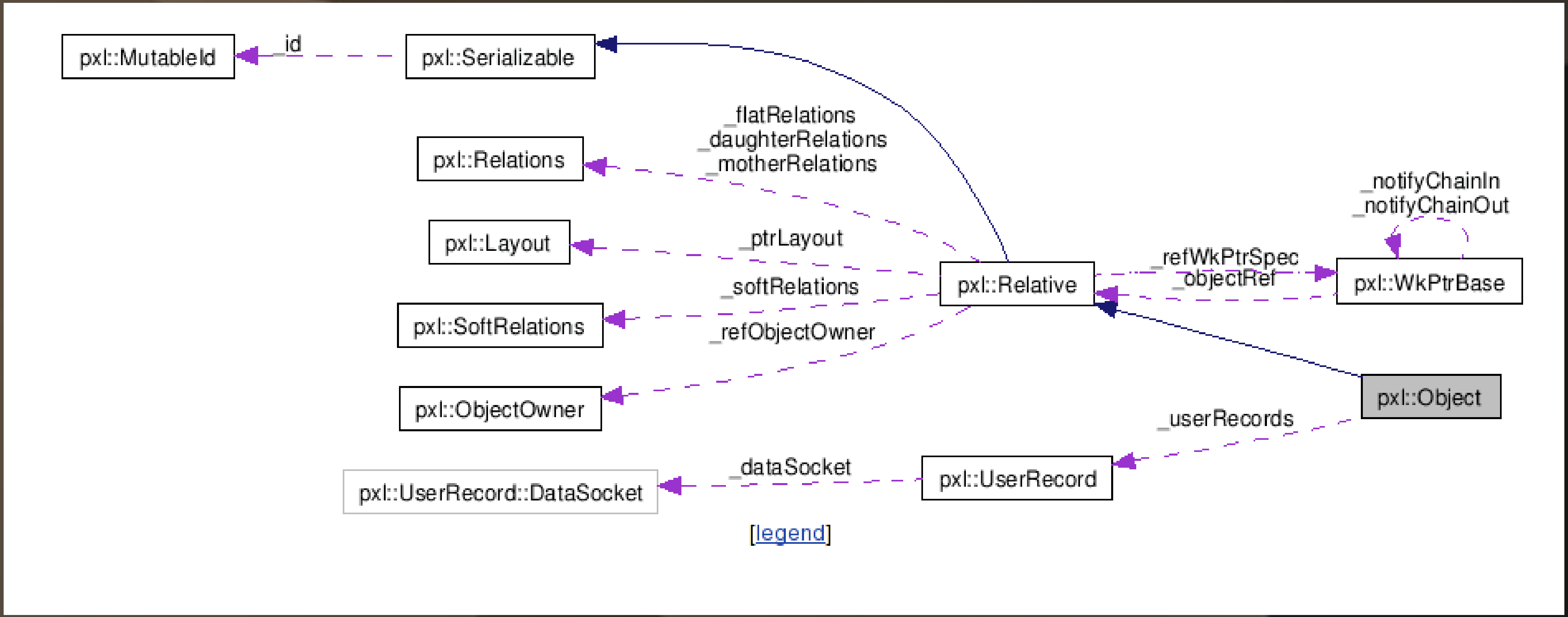


| Physics process | number of particles | number of vertices | number of configurations | Time/event [ms] | Mem. alloc. [MByte] |
|---|---|---|---|---|---|
| $W \to l\nu$ | 3 | 1 | 2 | 0.06 | $< 1$ |
| $H \to 4\mu$ | 7 | 3 | 3 | 0.4 | $< 1$ |
| $t\bar{t} \to \mu\nu 4j$ | 11 | 5 | 24 | 2.3 | $< 1$ |
| $Ht\bar{t} \to 8j$ | 13 | 5 | 5040 | 411 | 36 |

on a standard personal computer:

- ~20 µs per reconstructed decay vertex
- ~0.6 kByte per reconstructed particle in the decay trees

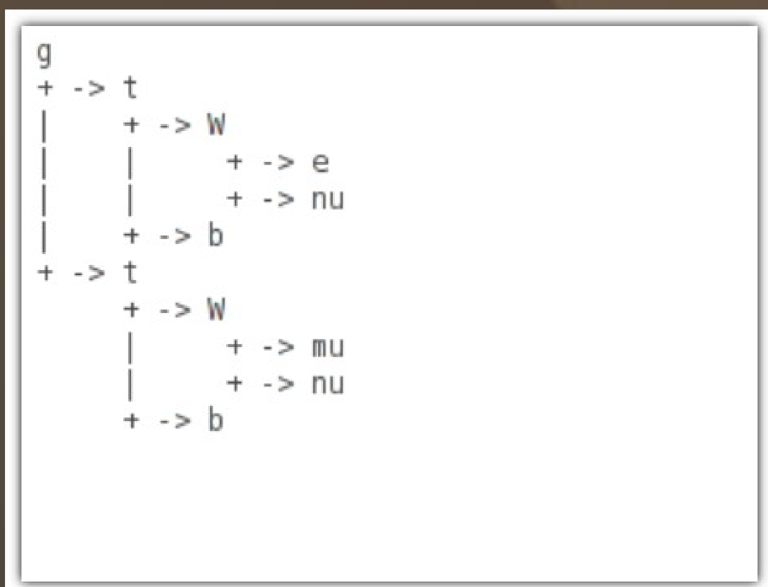
# PXL Objects Structure




- **Inheritance and composition**
  - **I/O** ***(pxl::Serializable)***
  - **relations** ***(pxl::Relative)***
  - **User data** ***(pxl::Object)***
  - **Object container** ***(pxl::ObjectManager)***

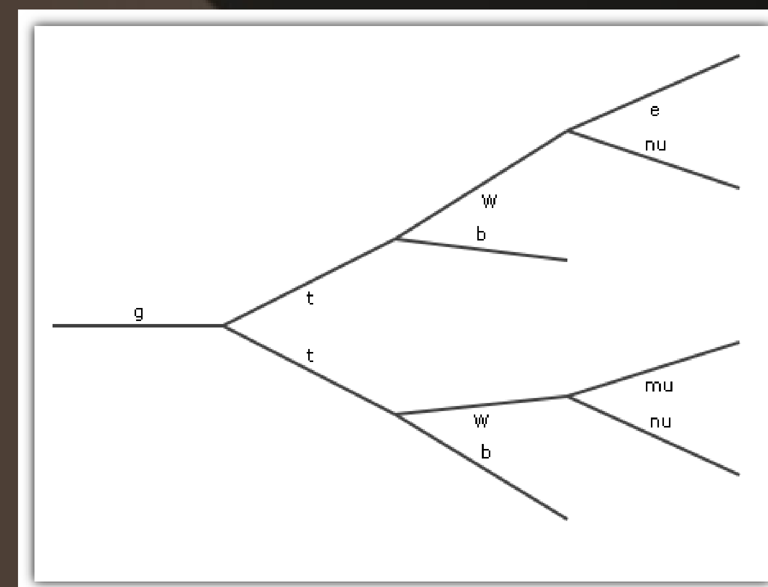
# Graphical platform: autolayout algorithm

Hard to read text output

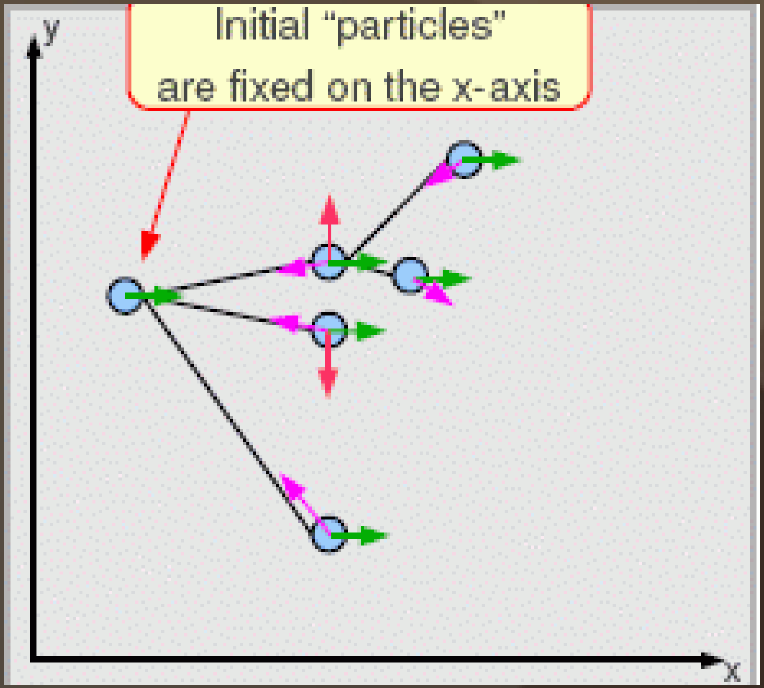


```
g
+ -> t
|     + -> W
|     |      + -> e
|     |      + -> nu
|     + -> b
+ -> t
      + -> W
      |      + -> mu
      |      + -> nu
      + -> b
```

Desired, easy to read decay tree



# Autolayout algorithm based on model forces




**Constant forces** to all nodes

**Repelling forces** between close node

**Spring forces** to all daughter nodes

**Friction forces** to all moving nodes

- **After some iterations all nodes moved into the positions where all forces on the node cancel each other**
- **After this the system is in stable state**

# Autolayout algorithm based on model forces: final result