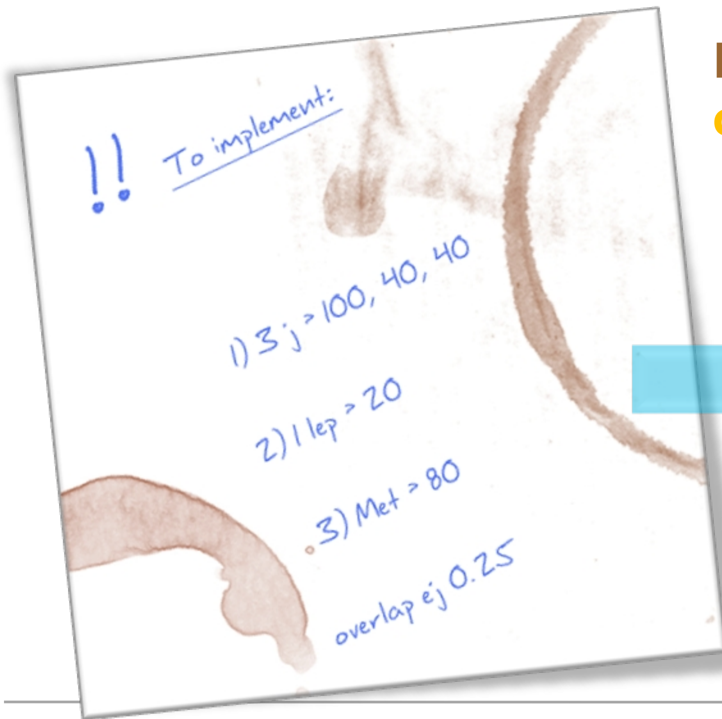
WatchMan Project

Computer Aided Software Engineering applied to HEP Analysis Code Building for LHC

Riccardo-Maria BIANCHI, *Renaud BRUNELIERE* (*Freiburg University*)



From the coffee table... to the analysis code!

```
f cutPassed:

    if not len(leptons) == len( value ):
        cutPassed = False
        if self.printLevel < 3: print self.name,'.
    if cutPassed:
        for i,lepPtCut in enumerate( value ):
            if leptons[i]['Pt'] < lepPtCut:
                if self.printLevel < 3:
                    print self.name, '.selectEvent
                cutPassed = False
                pass
```

ACAT 2010 - 13th Workshop on Advanced Computing and Analysis Techniques in Physics Research
Jaipur, India, 22-27 Feb. 2010

Physikalisches Institut
Albert-Ludwigs-Universität Freiburg

GEFÖRDERT VOM
Bundesministerium für Bildung und Forschung

WatchMan: Analysis Code Generator

- WatchMan is an Analysis Code Generator:

  ...it automatically builds Analysis Code, from user settings.

- Here we will present the main features of the package.

- **Different from other frameworks and toolboxes:** ***code is generated, not written!***

- **Less error-prone, more efficient, easier to debug and validate**

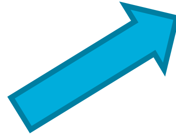
First Question: Why automating Code writing?

Growing of common code…

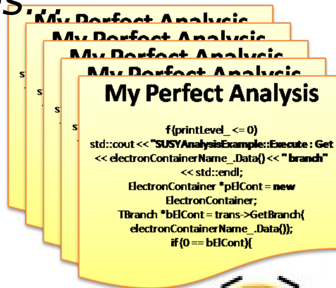
Usually we start with a class for one analysis…

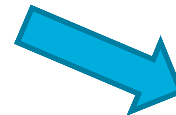
My Perfect Analysis

```
f(printLevel_ <= 0)
std::cout << "SUSYAnalysisExample::Execute : Get
<< electronContainerName_.Data() << " branch"
<< std::endl;
ElectronContainer *pElCont = new
ElectronContainer;
TBranch *bElCont = trans->GetBranch(
electronContainerName_.Data());
if (0 == bElCont){
```

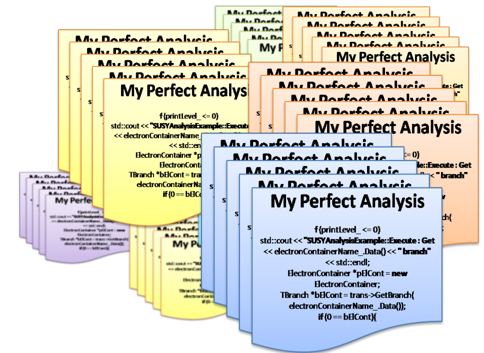


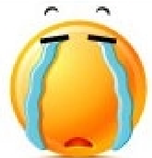
*…then we **cut-and-paste** code when we start other classes…*





…and too often we end up with a plethora of classes with most of the code in common…





This approach is
Error Prone:
many classes
to debug and maintain

Let's automate the analysis code generation!!

The main idea that led to WatchMan development

You specify in a simple way:

- Algorithms;
- Cuts;
- Samples to run upon;
- Plots you want

...and the actual analysis code is **automatically generated**, besides a series of tools to take care of running it.

CASE – Computer Aided Software Engineering

Computer-aided Software Engineering, or just **CASE**, means

"let the computers do the tedious work"

And generally ***results in high-quality, defect-free (ideally!!), and maintainable software products***. *(1)*

CASE packages are a set of ***automated tools that can be used in the software development process***. *(2)*

...and let us have fun thinking about new analysis strategies! :-)

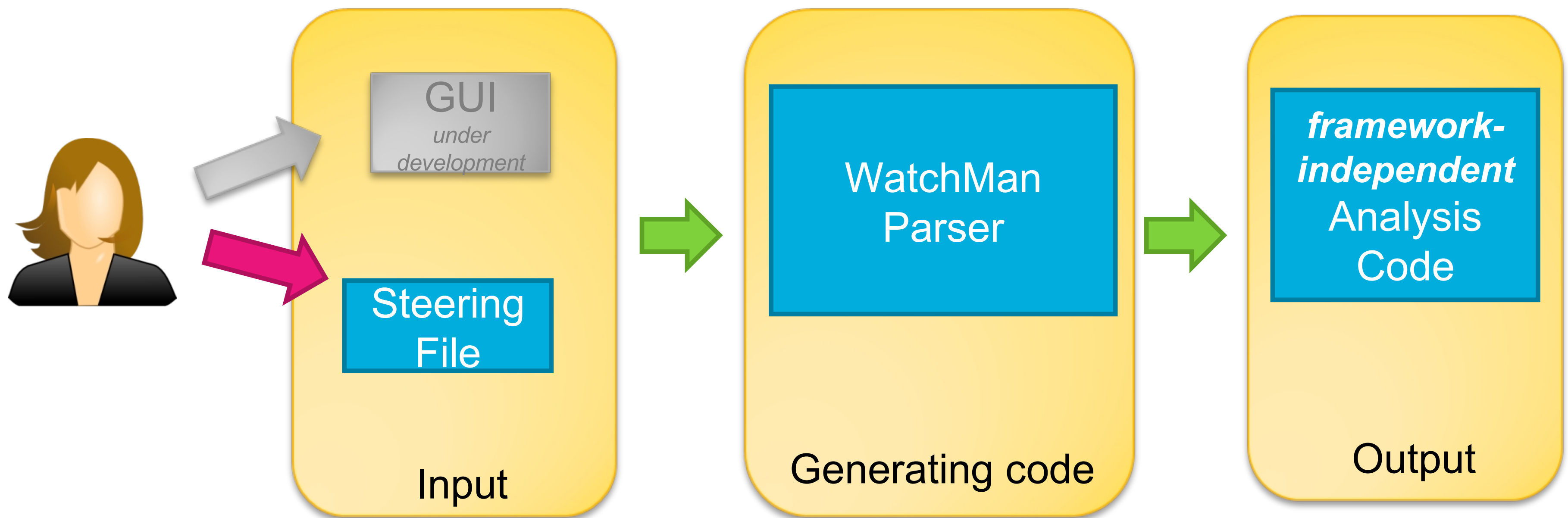
Thus it gives an ***Analysis Code***...

...More flexible;
...Less error prone;
...Easier to validate;
...Easier to maintain.

(1) Kuhn, D.L (1989). "Selecting and effectively using a computer aided software engineering tool". Annual Westinghouse computer symposium; 6-7 Nov 1989; Pittsburgh, PA (USA); DOE Project.
(2) P.Loucopoulus and V. Karakostas. System Requirement Engineering

Automated Building of the Analysis Code

WatchMan is a Code Generator, a **Software Factory**, aimed to build AnalysisCode easily



The user defines her/his analyses (as many as wanted) in an easy way, and the actual analysis code is **dynamically generated**

Less error-prone, more efficient, easier to debug and validate

Features

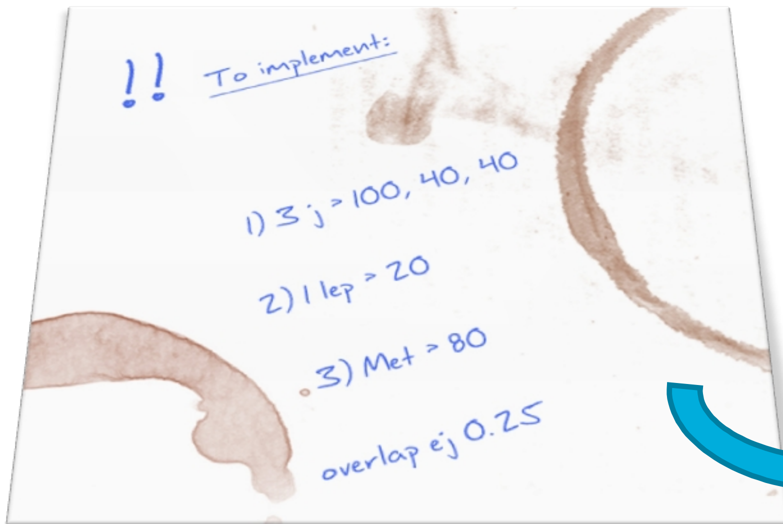
- WatchMan only depends on ROOT and Python. And it's a stand-alone self-contained package.
  Download and try it!

- The user can:
  - define as many physics analyses / channels as wanted
  - Define many object selection definitions
  - add UserData to output Ntuple
  - plug in user-defined custom code

- Combining all this in one steering file in order to get a Ntuple with all that info and objects.

1st example of user input via steering file: Defining an Analysis

- Basically in the ***steering file*** you transfer exactly what you scribbled down on paper when you thought about your new analysis, atthe coffee table…




```
channels = {
   '3j0lepMediumCuts':{'channel’: ‘ljjjv’,

         'objSelection':  {'electron': {'deltaR_ej': 0.25}},

         ‘cuts': {  1:   { ‘label': ‘leptonPtCutsExclusive‘
                           ‘value': [20*Units.GeV]},
                    2:   { 'label': 'jetPtCuts',
                           'value': [100*Units.GeV,
                                     40*Units.GeV,
                                     40*Units.GeV],  },
                    3:   { 'label': 'missingEtCut',
                           'value': 80*Units.GeV, },
                    },
               },
```

Adding new Analyses

Here an example of Steering File with 3 channels and 2 object selection definitions

- Here, we wanted to add:
  - another channel like the 1st one to test a different order in the event selection cuts;
  - a channel with same object selection definition, but different event selection cuts
  - another new channel with new object selection definition and new event selection
- We just changed the cut keys to change the order of cuts
- And we made use of the **objectSelection** variable to use a common definition for more channels, adding as many definitions as wanted

```
objectSelectionAndOverlap = {
  'myObjSelCuts': {{ 'muon':{'ptMin': 20.*Units.GeV},
                      'electron':{'ptMin': 20.*Units.GeV},

  'myNewObjSel': {'muon':{'ptMin': 5.*Units.GeV},
                  'electron':{'ptMin': 10.*Units.GeV}, }, }

channels = {

 'myChan':{'channel': 'ljjjv'
     'objSelection': 'myObjSelCuts',
     'cuts': {  1:   { 'label': 'leptonPtCutsExclusive' 'value': [20*Units.GeV]},
                2:   { 'label': 'jetPtCuts', 'value': [100, 40, 40]*Units.GeV],  },
                3:   { 'label': 'jetPtVeto', 'value': [40*Units.GeV] },
                4:   { 'label': 'missingEtCut', 'value': 80*Units.GeV, },     },

'myChan_New':{'channel': 'ljjjv'
     'objSelection': 'myObjSelCuts',
     'cuts': {  3:   { 'label': 'leptonPtCutsExclusive' 'value': [20*Units.GeV]},
                1:   { 'label': 'jetPtCuts', 'value': [100, 40, 40]*Units.GeV],  },
                2:   { 'label': 'jetPtVeto', 'value': [40*Units.GeV] },
                4:   { 'label': 'missingEtCut', 'value': 80*Units.GeV, },     },

'anotherChan' : {channel': 'emjjjj'
     'objSelection': 'myNewObjSel'
     'cuts': {  3:   { 'label': 'electronPtCutsExclusive' 'value':
                                                          [20*Units.GeV]},
                3:   { 'label': 'muonPtCutsExclusive' 'value': [10*Units.GeV]},
                1:   { 'label': 'jetPtCutsInclusive', 'value':
                                                 [100,40,40]*Units.GeV] },}
```

2nd example of user input via steering file: User-defined Formulas

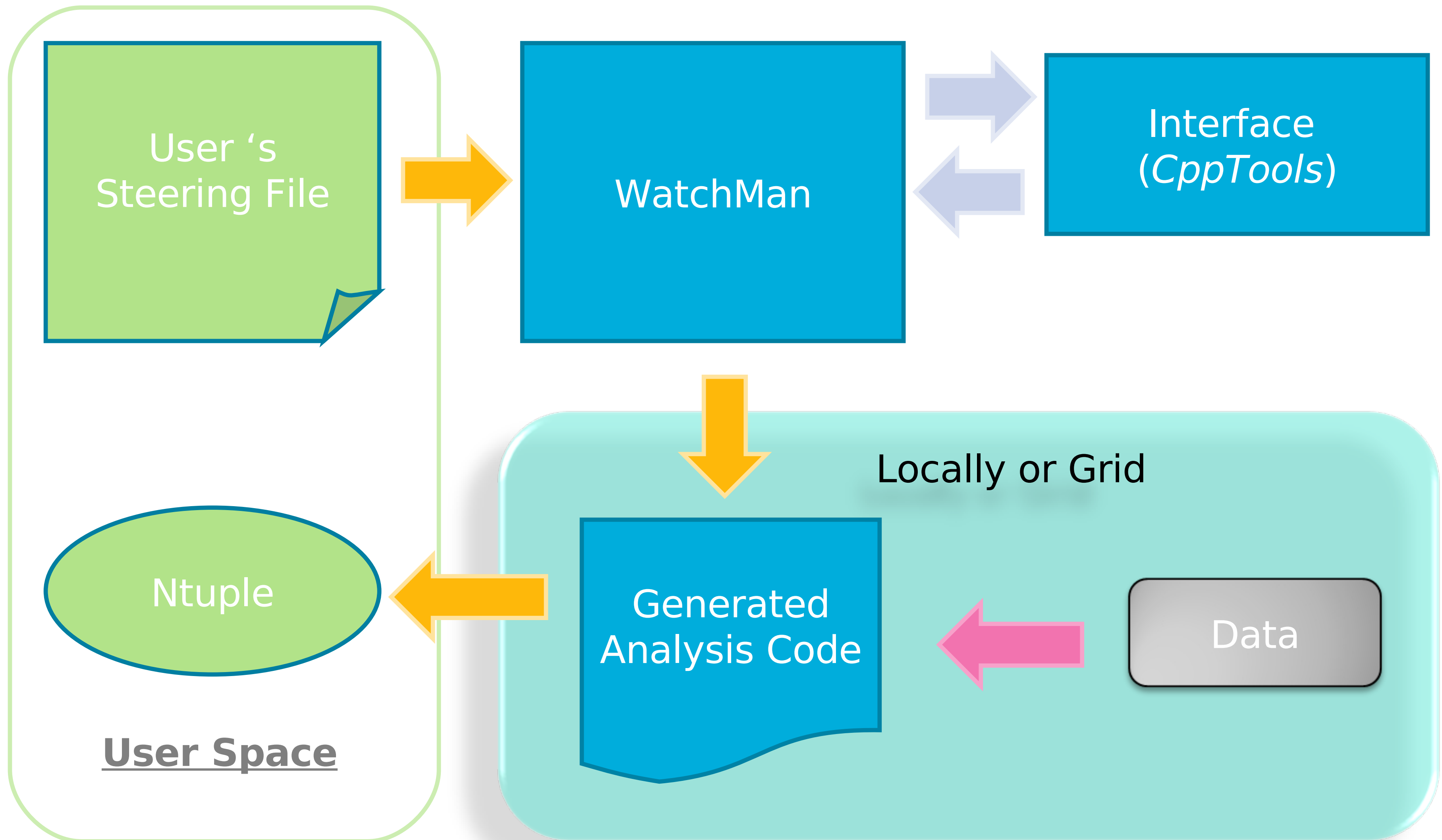
And a <u>user-defined custom formula</u>, as e.g. that for the M_{eff} calculation, can be written in the userFormula variable as Python/PyROOT code

The same formula **can be used for cuts or to fill branches with UserData**

The user can also asks for dumping objects or adding collections...see the backup slides and the Wiki for more info...

```
userFormula = {
'meff4j':
"""
meff = 0.
if len(candidates['jet']) < 4: return meff
for i,jet in enumerate(candidates['jet']):
    if i >= 4: break
    meff += candidates['jet'][i].pt()
    pass
for i,el in
enumerate(candidates['electron']):
    meff += candidates['electron'][i].pt()
    pass
for i,mu in enumerate(candidates['muon']):
    meff += candidates['muon'][i].pt()
    pass
meff += candidates['met'].et()
return meff
""", }
```

WatchMan Current Layout



WatchMan Ntuple Model

- In a single Ntuple all the results from all the analyses/channels are stored, together with branch storing UserData and dumped objects.
- Data are stored in a smart way to get a good compromise between storage space and speed when reading/analyzing the Ntuple.
- **All objects** (particles and branch UserData) are stored only once, and **flagged according the Object Selection Definition** they passed.
- **All events** are **flagged according the Event Selection** they passed
- *More info and examples of reading/analyzing this Ntuple on the Wiki* *(see backup slides for links)*

WatchMan Ntuple: a unique ROOT file to store info related of all your analyses

Events Flagging

Events are flagged according to the the set of Event Selection cuts they pass.

channels
`vector<string>`

When **plotting distributions** you can ask:

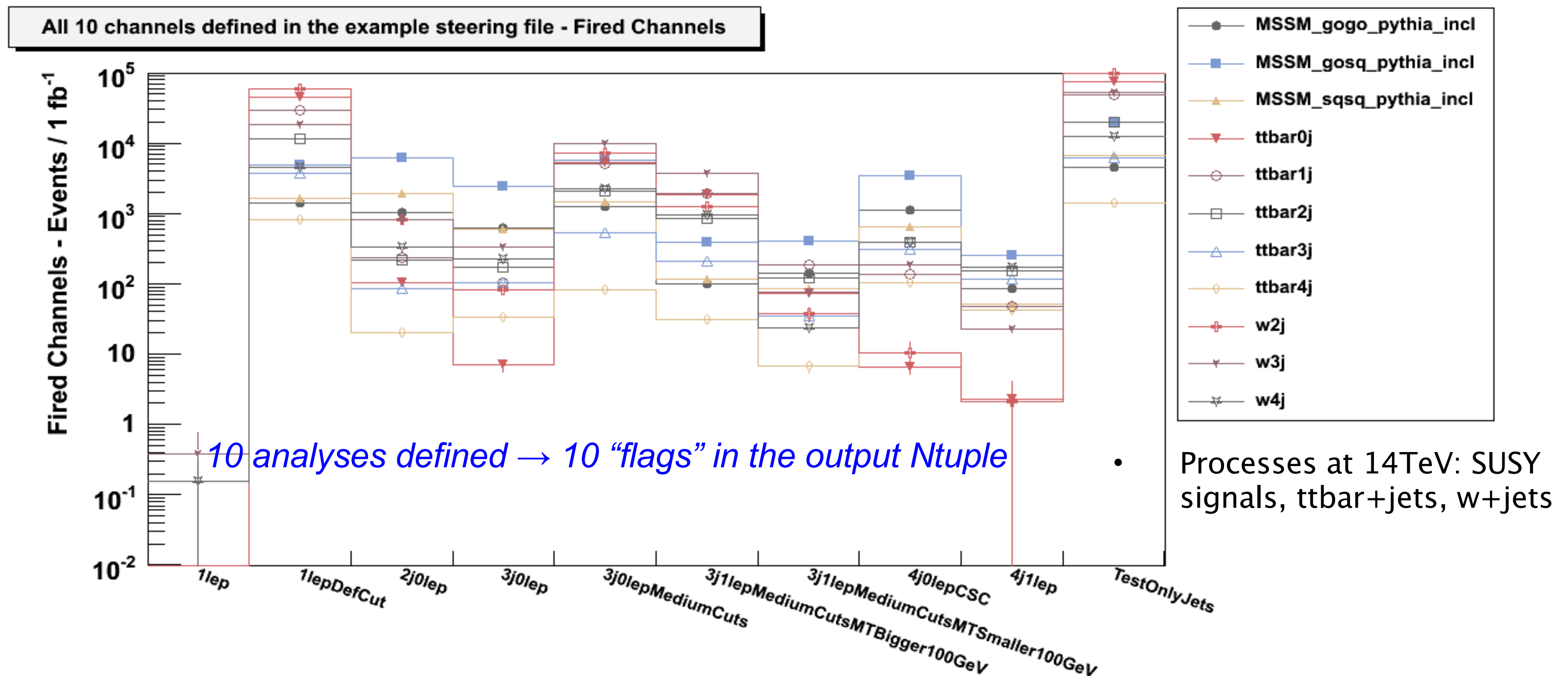
```
if channels[i] == “4j”
```

And you will get only those events having passed event selection cuts defined in “4j” channel/analysis

Particle-like objects, like jets or muons, **are flagged** in a similar way, according to the the set of object selection cuts that they passed.
See backup slides for more details...

Plotting the `channels` branch: example of an Analysis Output

- Here we used the default example steering file shipped with the package, and we run over some *Delphes* data files. *(about Delphes, sand used samples see backup slides)*
- 10 Analyses defined in the steering file, most of them SUSY-oriented.
- We just plot the **`channels`** branch from the output TTree.



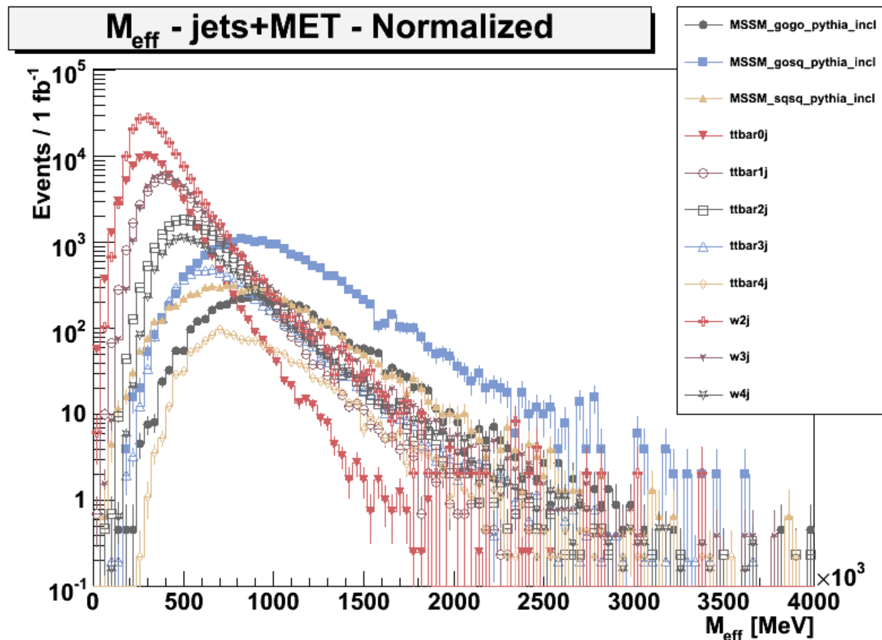
Easy to select only events having passed an analysis:

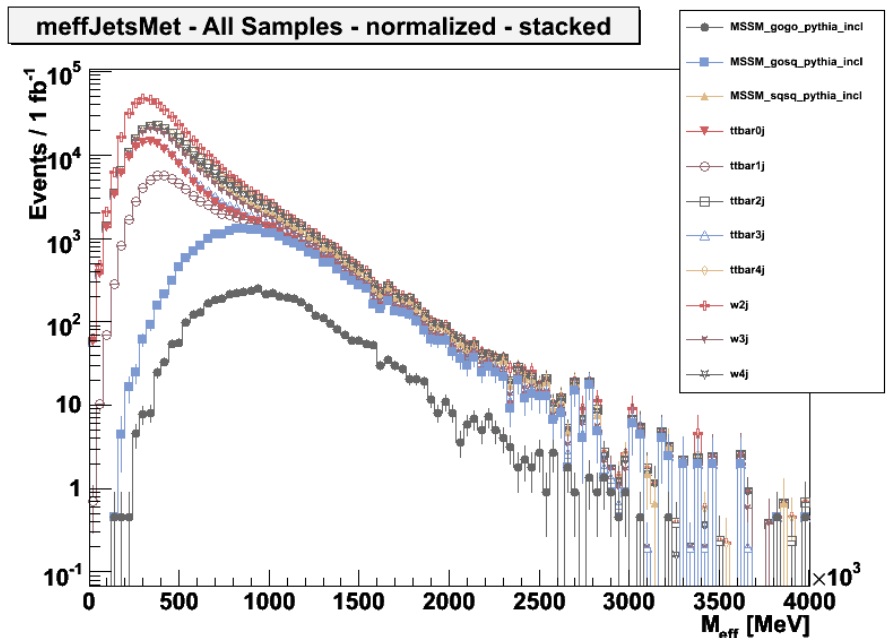
```
Tree.Draw( "jet.Pt", "channels.data == '4jets1lep' ")
```

Example of Analysis: Meff distribution

- As already set, all the branches defined by the user are computed for every object selection set of cuts, taking the right set of selected particles
- Here we plot the distribution of the SUSY variable Meff (here: *sum of pT of jets+MET*) from the output Ntuple, having fixed the object selection flag, and the event selection flag (the flag indicating a particular analysis). Metacode:

```
tree.Draw( ''meff'' )  if  'channels = ''4j1lep'' && objSelection = ''myObjSel''
```

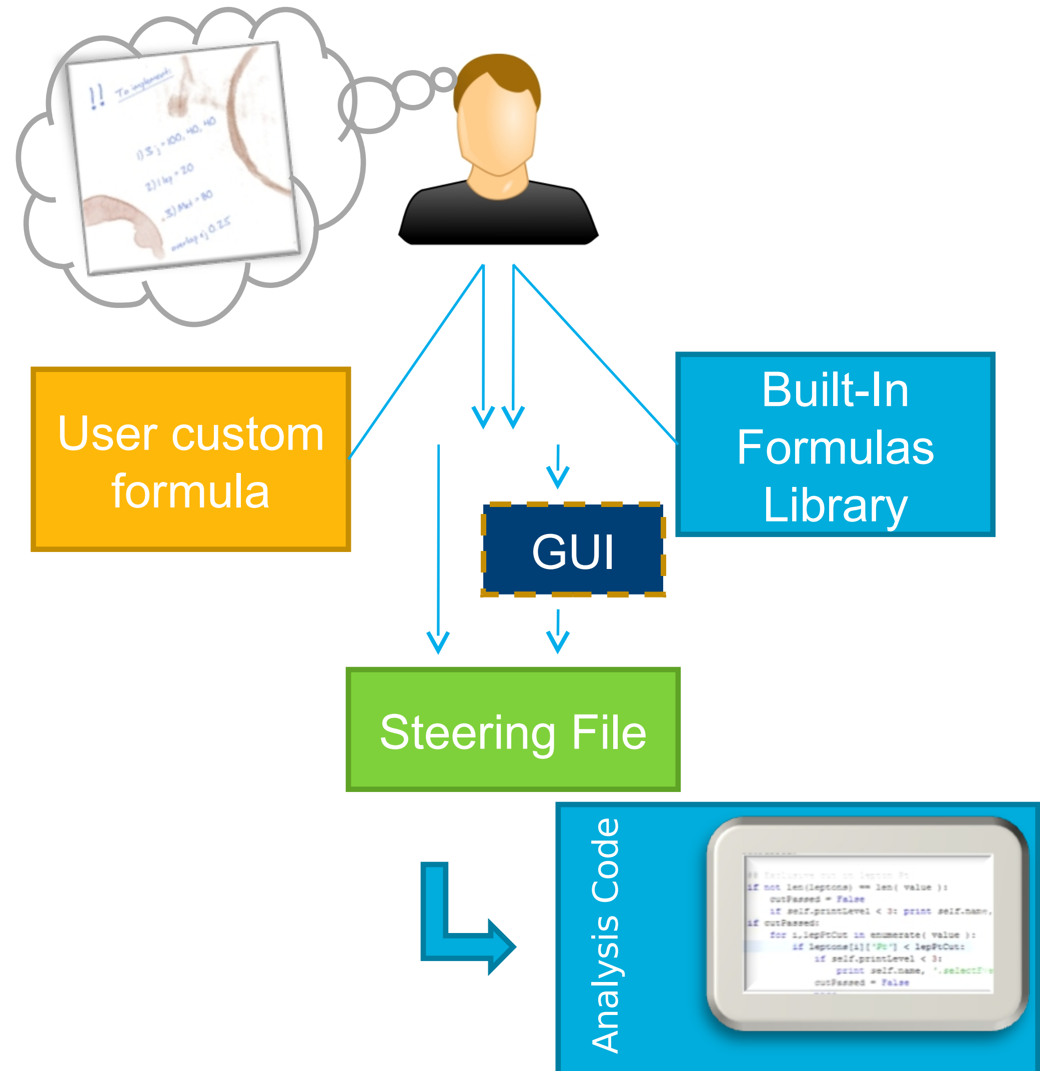




Default built-in cuts and formulas

- In order to ease analysis building, **a series of pre-defined built-in formulas**, collection names and cut values are provided with the package

- User can use those, **or add own custom formulas**

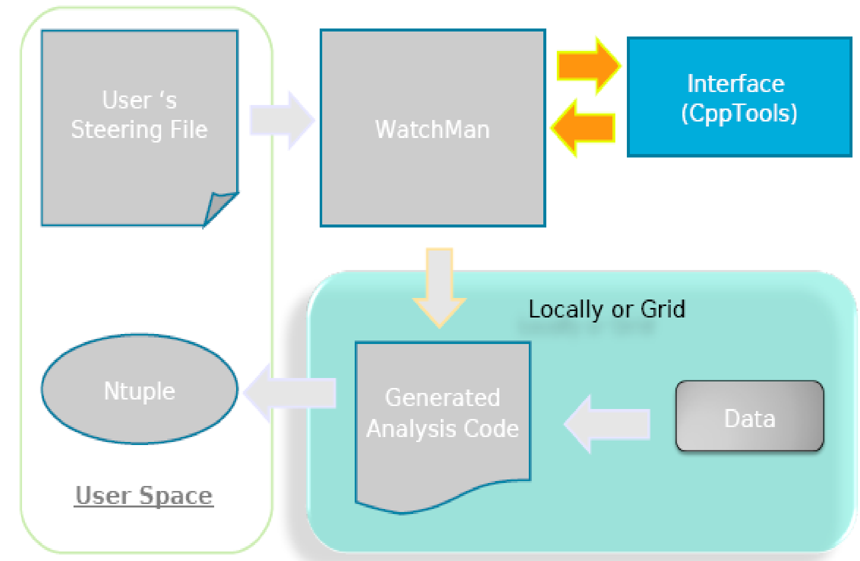


Other features

- **Trigger**:
  the user can ask for a list of triggers to be checked, to flag or skim events according to them
- **Skimming**:
  events can be skimmed or not according to the event selection (skimmed if it does not pass any channel)
- **Overlap Removal flags:**
  particles can be removed or only flagged if overlapping
- **Modularity:**
  Object Selection, Overlap Removal and Event Selection can be switched on/off independently, for each channel or for group of channels.
- **Cut-Flow Table:**
  For every analysis/channel the info on event selection cuts passed by each event is stored. Easy to extract then a cut-flow table

Plug-in a custom interface to your data format

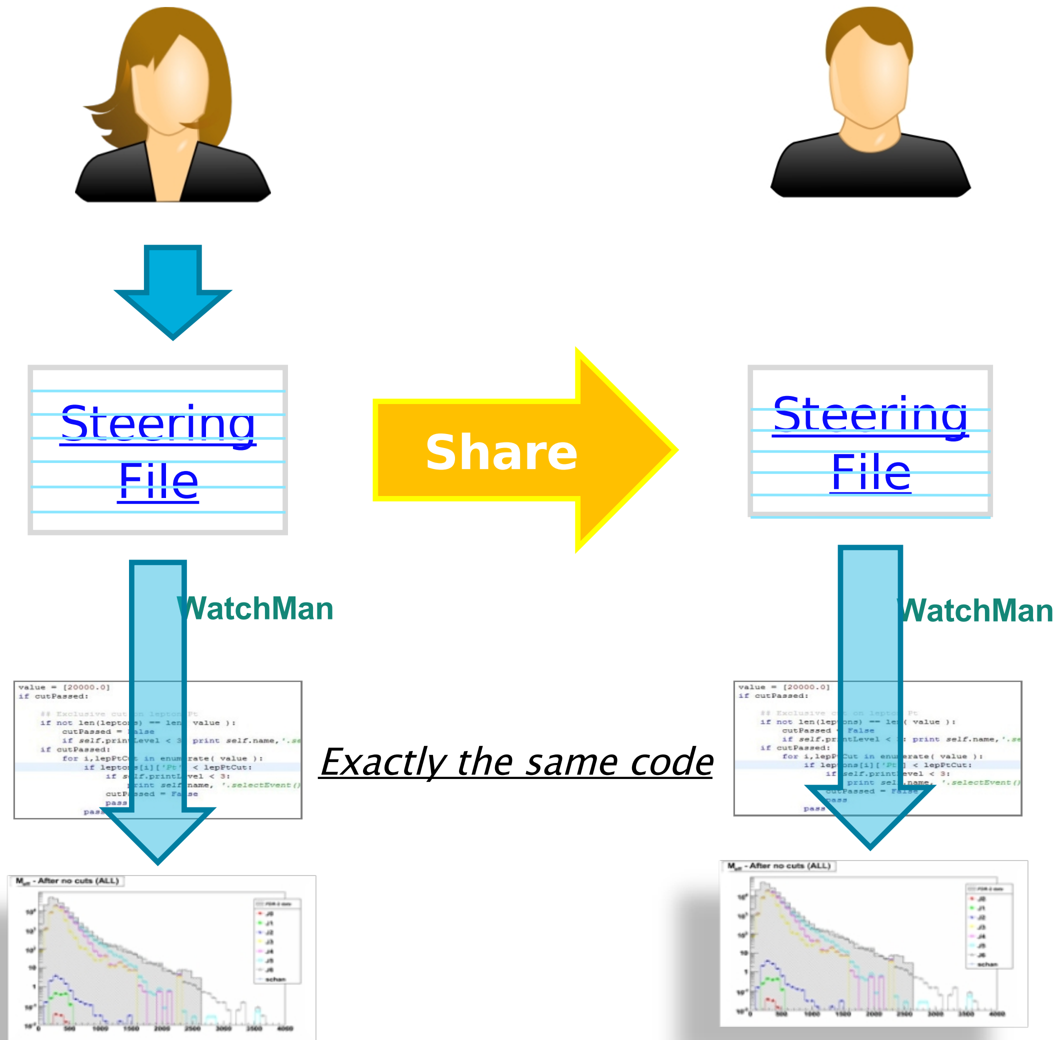
- WatchMan is a modular package
- You can build and plug-in a custom interface to your experiment framework, or your particular data file format
- Basically you just have to match some few API requirements
- The plug-in interface can be written in C++/Python
- ***More on that on the Wiki, soon***



2 interfaces provided with the package: Delphes and ATLAS.
More can be added.

Sharing code and comparing results

- It's very easy to share code and to compare results with other groups
- You **only need to send a text file (the *steering file*)**, and the others will obtain exactly the actual Analysis Code that you use.



VERY EASY to share, compare and validate results.

Validation

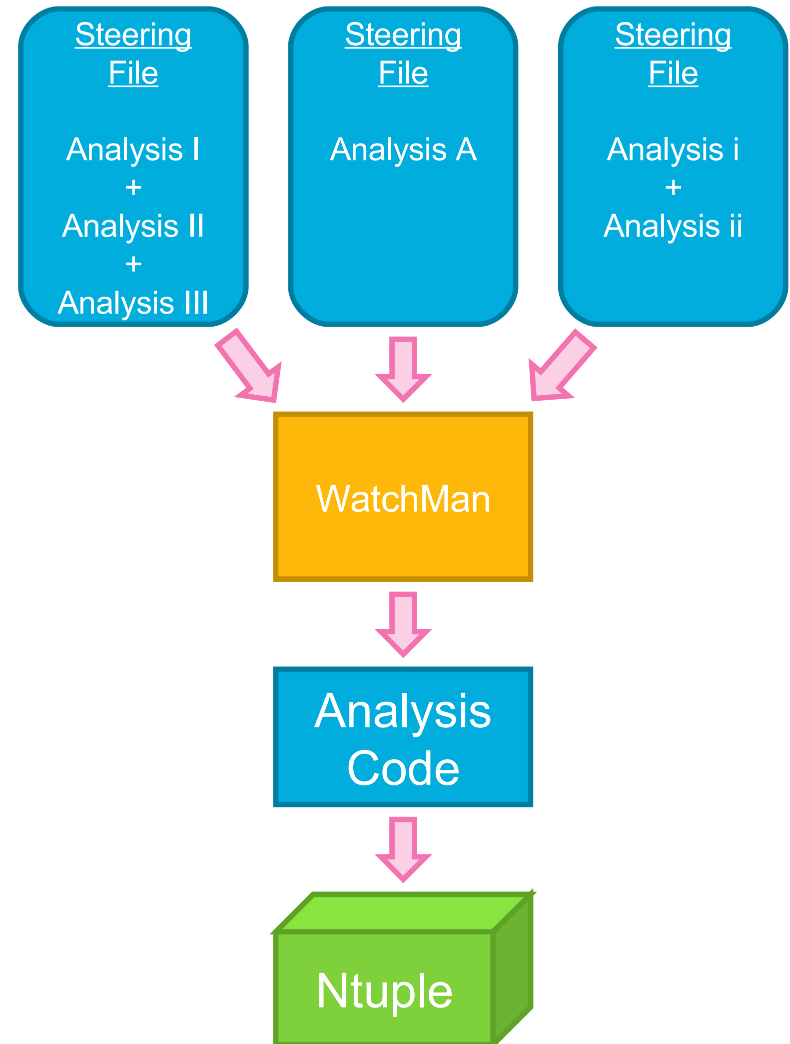
- WatchMan **generates** the analysis code.
- Once the package itself and its built-in formulas are validated, all the analyses using those formulas are automatically validated for free!
- In case the user uses custom formulas, only those should be validated.

Combine more steering files

With WatchMan you can combine different analyses in a snap!

It's a collaborative tool!!

•All the analyses defined in different steering files will be merged and run together at the same time… and all particles and events flagged according to the analysis they passed.



Who is using WatchMan?

- WatchMan is under **active development**

- Currently WatchMan is used by ~10-15 users to perform:

  - With the **ATLAS interface**:
    - Tau analysis and performance studies
    - Top Analysis
    - Photon Analysis
    - SUSY Analyses on many channels
    - Background estimations

  - With the **Delphes interface**
    - Studies of SUSY signatures

- It has also been used to run over data to create Ntuples for the ATLAS SUSY discovery reach plots for Chamonix'09 and '10.

Conclusions

- WatchMan is different from other frameworks and toolboxes: **code is generated, not written!**

- **Less error-prone, more efficient, easier to debug and validate**
- The user can define as many analyses, branches, objects to dump out, custom formulas he/she wants
- The output generated python code is framework-independent and ready to be run
- The output of the generated Analysis Code is an Ntuple tailored in order to optimize storage space and reading speed

- We are preparing a detailed Tutorial...
- ...but a simpler Quick Start on the Wiki is already present: **"Get your Analysis Code ready in 5 minutes!"** ;-)

WatchMan Docs and How To's

Quick Start:
Get your Analysis Code ready in 5 minutes! :-)

https://twiki.cern.ch/twiki/bin/view/Main/WatchManQuickStart

Main Wiki:
https://twiki.cern.ch/twiki/bin/view/Main/WatchMan

Trac Repository:
https://svnweb.cern.ch/trac/WatchMan/

Source Code:
https://svnweb.cern.ch/trac/WatchMan/browser

How to read the generated output Ntuple, example script:
https://twiki.cern.ch/twiki/bin/view/Main/WatchManPythonScriptExample

WatchMan
An automated Analysis Code Generator

Physikalisches Institut
Albert-Ludwigs-Universität Freiburg

ALBERT-LUDWIGS-UNIVERSITÄT FREIBURG

NEW

WatchMan is a *new, very light, highly automated* framework to easily and dynamically build Analysis Code for various data file formats.

Please notice: this Wiki is under construction. So many new info will come soon...

Home About Trac repository Source Code News Documentation Examples & Tutorials Community Download & Install Bugs Tracking Enter new ticket

Introduction
NEW Highlights

Quick Start!

The Wiki is under working and much more documentation will appear soon…

…and don't hesitate to contact us, for questions or suggestions! ☺
rbianchi@cern.ch, bruneli@cern.ch

Backup Slides

3rd example of user input: UserData

- It's very easy to add a branch to store UserData from a user-defined custom formula
- Below we define three branches storing three quantities: two from built-in formulas, one from user-defined custom formula

```
userD3PDBranchesToFill = {
 'meff4j' : {'label': 'meff4j', 'type': 'float', 'formula': 'meff4j'},
 'sphericity' : {'label': 'spher', 'type': 'float', 'formula': 'sphericity'},
 'TM' : {'label': 'TM', 'type': 'float', 'formula': 'transverseMass'},     }
```

- Those quantities will be **calculated for every object selection definition**, thus we will get the **right quantity from the right set of selected particles**
- Those quantities will be than stored in a TTree branch.

Other possible user inputs

adding new interface-related Collections

- It's easy to dynamically add new collections to the built-in default ones (electron, muon, jet, MET, …), when we need those extra collections for our analysis. Those collections can then be used then in cuts, branch as UserData and user-defined custom formulas.
- Particularly useful using the ATLAS interface, or a custom-built interface to other experiments, where data files contain a large number of various collections, and we do not want to access always all of them.

```
collections = {
   'track': {'type':'Rec::TrackParticleContainer','name':'TrackParticleCandidate'},
   'METSig':{'type':'MissingETSig','name':'METSig','select':False},    }
```

Dump objects or collections from data files

- Also this feature is particularly useful when analyzing data files from experiments (for example using the ATLAS interface). Those data files contain a lot of physics and metadata information, not always related to our analysis, but that we might want to dump and save in our Ntuple output.

```
dumpContainers =  { 'METSig':{'L':{'type':'float','method':'.sigL()'}},    }
```

Generating the Analysis Code!

- Once you filled the steering file with as many channels, branches, dumped objects you want…
- …just run WatchMan Parser and **you get the Analysis Code dynamically generated! Ready to be run!!**

```
$run/> python WatchManBuilder.py mySteeringFile.py
```

Object Selection Flagging

Let's say in the *Steering File* I define 3 object selections

Map in *InfoTree*

Default	TauSelec	MyChan
0	1	2

objSelection definitions

154	128	85	80	71	54	11

jet4mom.pt()

vector<TLorentzVector>

Default	1	1	1	0	0	0	0
TauSelec	1	1	0	0	1	1	0
MyChan	1	1	1	0	1	0	0

jetObjSelection

vector<vector<string>>

This from a particular Channel with a custom objSelection defined by the user

In this case the 3rd jet did not pass the cuts in the "TauSelec" object selection, and a "0" was put in the corresponding place inside the ***jetObjSel*** *vector*

About Delphes format

Delphes is a fast simulation framework to simulate the response of a general High Energy Physics detector, starting from MC saples

Web page: http://www.fynu.ucl.ac.be/users/s.ovyn/Delphes/UserMan.html

All the Delphes data samples used in this presentation are freely available at the CERN MonteCarlo DataBase site:

http://mcdb.cern.ch