The Likelihood Machine

A Tool for Model-Agnostic Cross-Section Analyses and Other Counting Experiments

L. Koch
III. Physikalisches Institut B, RWTH Aachen University
State of the Nu-tion, NuInt2017, University of Toronto

Challenges of cross-section measurements

- What me measure is not what happened
  - reco space $\neq$ truth space

My model of how we measure reality:

1. Something happens (truth space)
2. The happening causes signals in the detector
3. Reconstruction and analysis create an imperfect ("smeared") picture of what happened (reco space)

- In general we are interested in what happened, *not* in the picture

A "standard" way to get back to truth:

1. Measure something (take a picture in reco space)
2. Unfold to remove the detector effects
3. Publish unfolded data points with statistical and systematic uncertainties (truth space)

Challenges of cross-section measurements

- What me measure is not what happened
  - reco space $\neq$ truth space

My model of how we measure reality:

1. Something happens (truth space)
2. The happening causes signals in the detector
3. Reconstruction and analysis create an imperfect ("smeared") picture of what happened (reco space)

- In general we are interested in what happened, *not* in the picture

A "standard" way to get back to truth:

1. Measure something (take a picture in reco space)
2. Unfold to remove the detector effects $\leftarrow$ **this is really hard!**
3. Publish unfolded data points with statistical and systematic uncertainties (truth space)

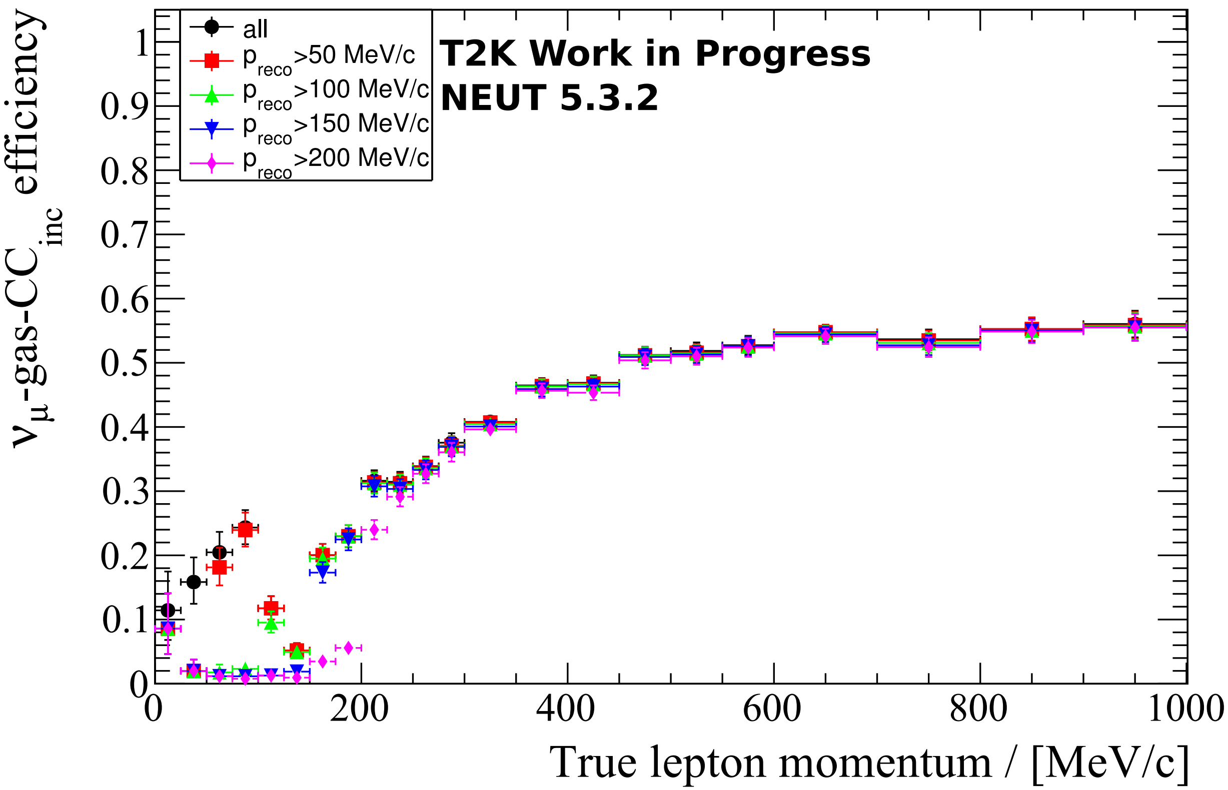
Let's try something simple

- Working on first neutrino cross-section measurement on gaseous argon at T2K
- Low density target $\Rightarrow$ low statistics
- Original plan was aiming for simple, single-bin measurement of inclusive CC reactions
  1. Find a muons starting in the TPC
  2. Count number of events
  3. Apply purity and efficiency correction ("unfolding light")
  4. Done.

Let's try something simple

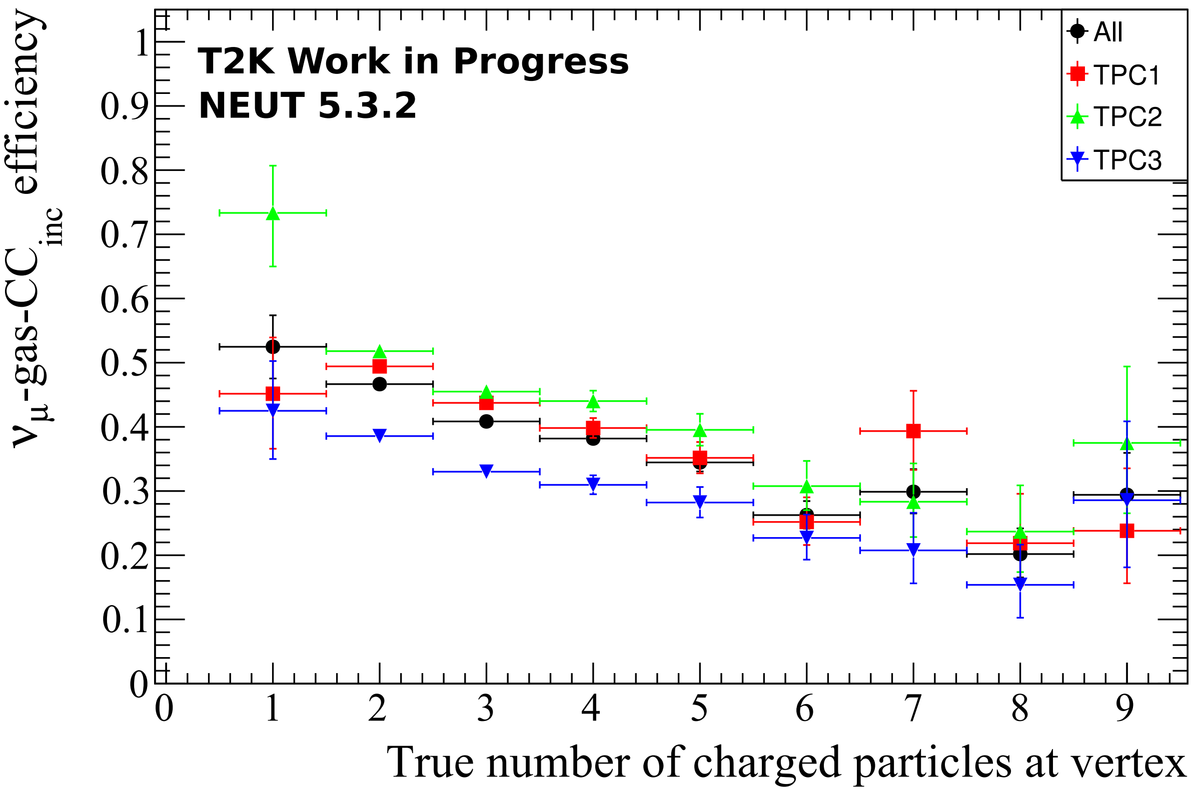
- Working on first neutrino cross-section measurement on gaseous argon at T2K
- Low density target $\Rightarrow$ low statistics
- Original plan was aiming for simple, single-bin measurement of inclusive CC reactions
  1. Find a muons starting in the TPC
  2. Count number of events
  3. Apply purity and efficiency correction ("unfolding light")
  4. Done.

No such thing as “simple”



- Detection efficiency depends on event properties
  - E.g. muon direction and momentum
- Overall efficiency and purity of sample depend on the physics model
- Fix: Constrain measurement to phase space of flat efficiency

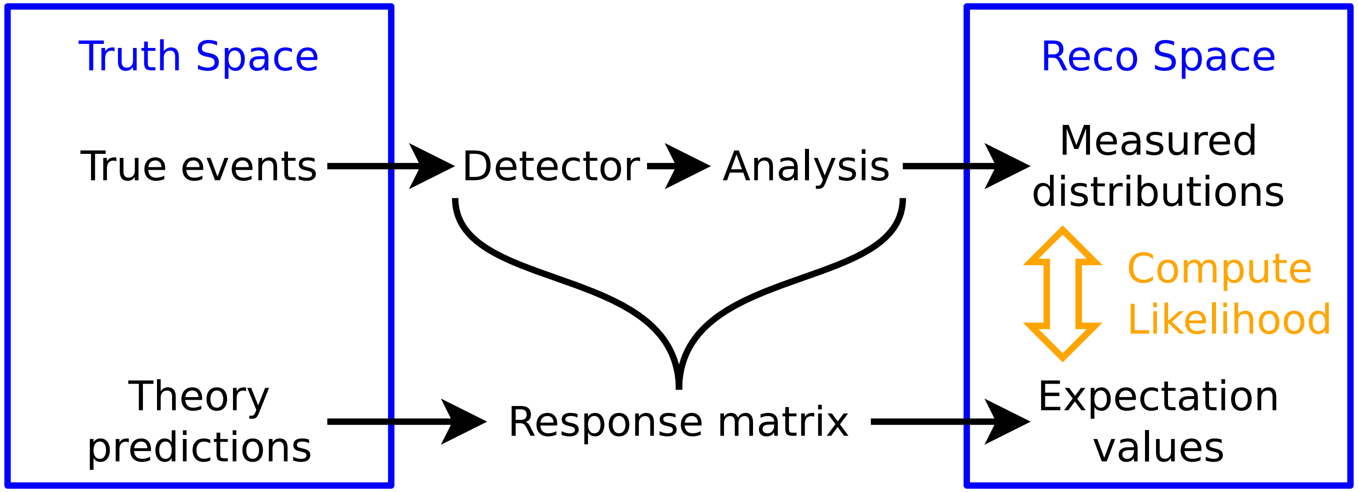
And it gets even un-simpler



- Detection efficiency also depends on particle multiplicity
  - The more stuff is going on, the easier the reconstruction gets confused
- Unlike with the muon momentum, there is no clear plateau
  - Where to put the constraints of phase-space?

Let's try something smart

Since simple won't work...



- Transition from truth to reco space and back is not symmetric
  - Differences in truth space are smeared out in reco space
- It is hard to find the original truth distribution from a given reco distribution
- It is *easy* to get the smeared reco distribution from a given truth distribution
- Instead of bringing reco data to truth space, bring model predictions to reco space
- Use response matrix to handle smearing and efficiency
  - Contains all information about the detector, reconstruction and event selection

Details

- Forward only
  - The matrix will only be used in the “forward folding” direction
  - Does not need not be invertible, can have arbitrary shape

$$\nu_i = R_{ij}\mu_j$$

- Likelihood calculation for given truth expectation values is straight-forward

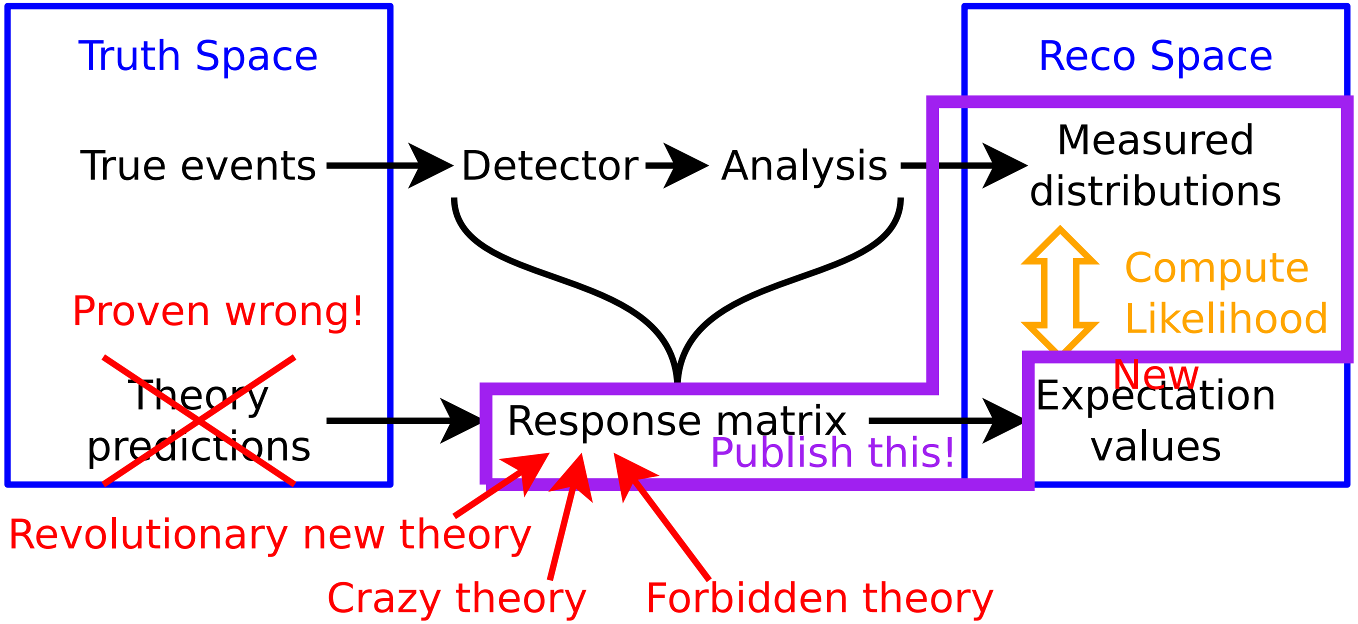
$$L = \prod_i \frac{(R_{ij}\mu_j)^{n_i}}{n_i!} \exp(-R_{ij}\mu_j)$$

- Systematic uncertainties on matrix only
  - The data is the data is the data
    - Actually, it is the only thing we are 100% sure of!
- Data statistics are automatically considered when using likelihood to build confidence/credible intervals
  - Stat errors on data points only make sense when interpreting data as imperfect measurements of expectation values
  - We generate expectation values from truth with the response matrix, though!

Systematic uncertainties

- Systematics only apply to the response matrix
- Analytical propagation difficult and usually assumes normal distribution
- Do numerical propagation instead
- Detector simulation is varied according to systematics
- Each variation yields its own response matrix
- Set of response matrices contains all information about systematics
- Each matrix yields own likelihood for given truth prediction
- Treat selection of response matrix as nuisance parameter
  - Calculate average for marginal likelihood
  - Select maximum for profile likelihood
  - Let MCMC select toy matrix randomly according to likelihood

Think of the possibilities



- A matrix is a simple mathematical object
  - Easy to share with others, easy to use
- Publishing data together with response matrix!
  - Enables anyone to test their physics model against our data
  - Longer “shelf life” of data
  - Faster test of new theories
- Even better: Publish framework for using the response matrix

Introducing the Likelihood Machine

Taking care of the hard stuff for you

Aims

- Be accessible
  - It should be easy to obtain the software and get started
- Be user friendly
  - Make it as easy as possible to use the software
- Be flexible
  - Allow more complicated analyses, if the user wishes to do so
- Be maintainable
  - Ideally this software will be used for years to come!

Info

- Written in Python
  - Very easy install with `pip` (not implemented yet)
  - Flexible to a fault
- Open source
  - Currently hosted on Github: `https://github.com/ast0815/likelihood-machine`

Defining the binning

- To create a response matrix one needs to define the bins in truth and reco space
- N-dimensional binnings are described in YAML files:

```yaml
!RecBinning
binedges:
- - reco_momentum
  - [0.0,
    100.0,
    200.0,
    .inf]
- - selection
  - [0,
    1,
    2,
    3]
include_upper: false
```

```yaml
!RecBinning
binedges:
- - true_momentum
  - [0.0,
    100.0,
    200.0,
    300.0,
    .inf]
- - reaction_type
  - [0,
    1,
    2]
include_upper: false
```

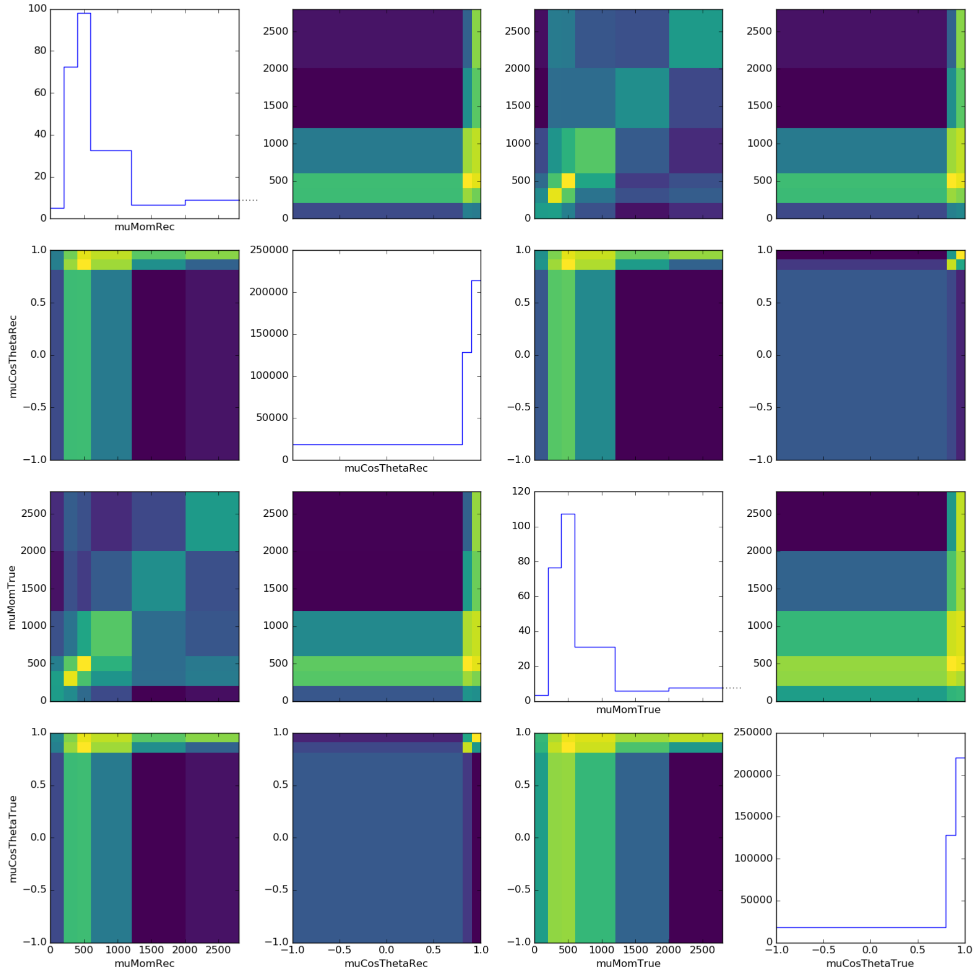
Building the response matrix

```
import binning
from migration import ResponseMatrix
with open("truth-binning.yml", 'r') as f:
    truth_binning = binning.yaml.load(f)
with open("reco-binning.yml", 'r') as f:
    reco_binning = binning.yaml.load(f)
resp = ResponseMatrix(reco_binning, truth_binning)
resp.fill_from_csv_file("simulated_data.csv")
resp.fill_up_truth_from_csv_file("true_events.csv")
```

- Data is taken from CSV files
  - One event per line, each binned variable as a column
- `fill_up_truth_from_csv_file` step optional
  - Needed only when `simulated_data.csv` does not contain all true events
- CSV is an extremely low-level data format
  - Easy to output by about every simulation software
  - Easy to read into about every kind of analysis software
- More specialised formats can be added in the future
  - E.g. direct reading of ROOT trees

Use the power of someone else's work

```
resp.plot_response_values('output.png', variables=(None,None))
```



- Many useful methods to check the matrix are already implemented
  - Plot values
  - Plot uncertainties due to MC statistics
  - Plot expected variation inside bins
  - ...
- I will add everything I need for my analysis, so others will be able to use it in theirs
- When other people start to use this, I will be very happy to work with them to implement whatever they need

The actual likelihood machines

```
from likelihood import LikelihoodMachine, CompositeHypothesis

matrix = get_response_matrix_as_ndarray()
lm = LikelihoodMachine(data_array, matrix)

print lm.log_likelihood(some_truth_prediction)

def my_theory(par):
    return truth0 * par[0] + truth1 * par[1]
H0 = CompositeHypothesis(my_theory,
                         parameter_limits=[(0,None),(0,1)])

print lm.max_log_likelihood(H0)
print lm.max_likelihood_p_value(H0)
```

- All calculations handled by Numpy (Numeric Python)
  - Flexibility of Python, performance of compiled C++
- Offers methods to do “common” analysis tasks

Also supports Bayesian analyses

Posterior sampling with Markov Chain Monte Carlo

```
from likelihood import JeffreysPrior
from pymc.Matplot import plot as mcplot

prior = JeffreysPrior(matrix, my_theory,
             [(0,None), (0,1)], [0.5, 0.5])
H0 = CompositeHypothesis(my_theory,
        parameter_priors=[prior])
mcmc = lm.mcmc(H0)

mcmc.sample(5000, burn=1000, thin=10)
mcplot(mcmc)
```

- A Markov Chain Monte Carlo algorithm is implemented using the PyMC package
- `JeffreysPrior` class takes care of calculating “objective” priors for parameters
  - Works numerically
  - Should be applicable to all hypotheses with continuous parameters

Outlook

Likelihood Machine

- Software framework for cross-section and general counting experiments
- Centered around response matrix concept
- Focus on ease-of-use and flexibility

Present

- Developed alongside T2K neutrino gas interaction analysis
- Entering “final integration” stage
  - Core functionality realised
  - Most of convenience functions done

Future

- Release v1.0 together with gas interaction analysis
  - Distribution via `pip`
- Adapt to needs of other analyses
  - Thinking about using this? Get in touch as early as possible!

Thank you!

The Likelihood Machine in T2K

This is not a tutorial!

- Development is not finished and things might change without notice. Talk to me!

1. Use Highland to build data selection and do systematic variations
2. Create a large sample of varied MC data
3. Parse Highland ROOT output files to CSV
   - Could implement direct ROOT tree loading in the future
4. Define truth and reco binning for response matrix in YAML (text) files
5. Use Likelihood Machine `ResponseMatrix` to build set of response matrices and check their properties
6. Create (fake) data sample with Highland and parse to CSV
7. Use Likelihood Machine `LikelihoodMachine` to test hypotheses against data
   - Aside from the “hyper-hypothesis” (all truth bins free-floating), hypotheses must define truth expectation values.
   - Can be achieved by generating Monte Carlo events (truth only, no detector simulation needed!), filling it into truth binning and scaling to match POT
   - Mixed mode also possible: BG bins free-floating, signal bins defined by generator
8. When publishing, publish response matrix together with selection!

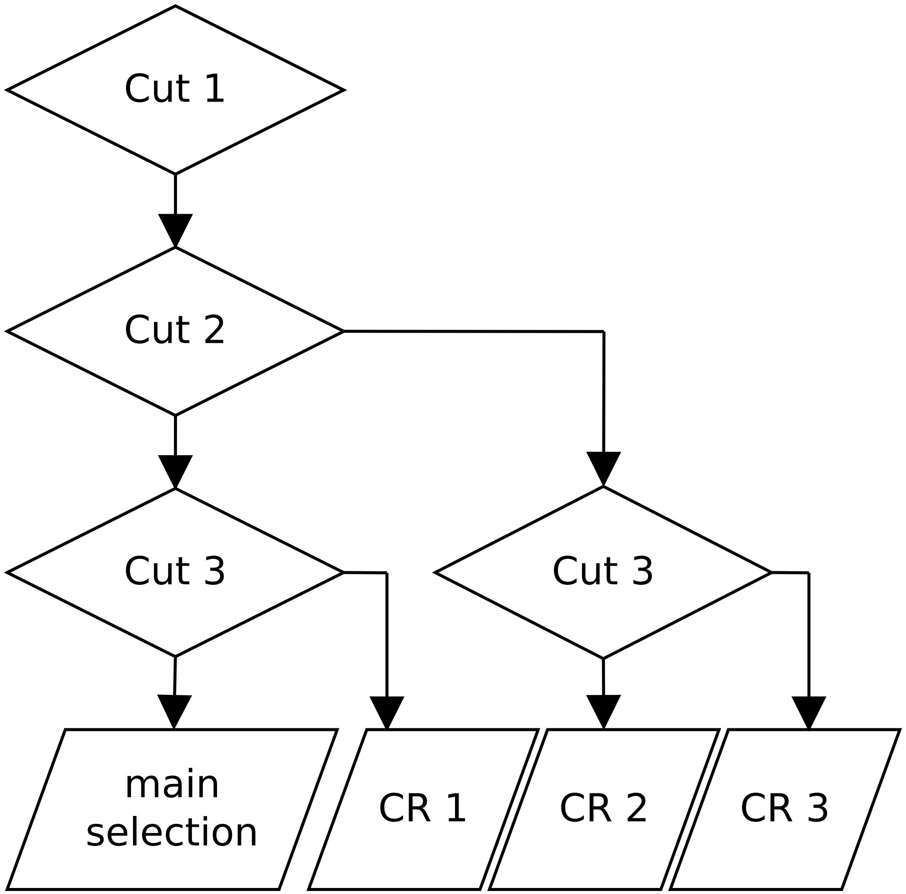
This is not a tutorial!

- Development is not finished and things might change without notice. Talk to me!

1. Use Highland to build data selection and do systematic variations
2. Create a large sample of varied MC data
3. Parse Highland ROOT output files to CSV
   – Could implement direct ROOT tree loading in the future
4. Define truth and reco binning for response matrix in YAML (text) files
5. Use Likelihood Machine `ResponseMatrix` to build set of response matrices and check their properties
6. Create (fake) data sample with Highland and parse to CSV
7. Use Likelihood Machine `LikelihoodMachine` to test hypotheses against data
   – Aside from the "hyper-hypothesis" (all truth bins free-floating), hypotheses must define truth expectation values.
   – Can be achieved by generating Monte Carlo events (truth only, no detector simulation needed!), filling it into truth binning and scaling to match POT
   – Mixed mode also possible: BG bins free-floating, signal bins defined by generator
8. When publishing, publish response matrix together with selection!

Creating a selection for the Likelihood Machine

- Create and optimise main selection as usual
- Add mutually exclusive branches as control regions for BG constraining
  - The Likelihood Machine will handle BG the same way it does signal
  - BG will be constrained by data in a "natural" way
    - No model dependence!
  - Re-claims signal events that end up in the control regions
    - Increased efficiency!



Caveat

- Systematic weight variations must conserve the number of events!
- Otherwise efficiency will be wrong

Parsing Highland output to CSV

- Currently done in analysis-specific script, will write something more universal later

```
true_mom,true_costheta,reactionBG,reco_mom,reco_costheta, \
    successful_branch,weight_syst_total__0,weight_syst_total__1
400.3,0.45,1,444.4,0.46,0,0.98,1.05
443.6,-0.05,7,666.6,0.06,3,1.04,1.12
```

- First line: variables names
  - Need not match root names
  - Should not contain special characters
- Each toy variable needs its own column
- Create two CSV files: One for `all_syst` tree and one for `truth` tree
  - `all_syst` used for mapping from truth bin to reco bin
  - `truth` used to get proper efficiencies

Caveat

- Truth variable contents in both trees must match *exactly*!
- For all truth bins, either *all* events in `all_syst` must be also present in `truth` (signal), or *none* (BG)