Round table
Today’s toolset

Several use cases

- Along the experiment life time
  - Early data taking => more details, detector level information, calibrations run by hand, less "standards" for reconstructed objects
  - Late data taking (with stable lumi conditions) => best practice are set, only need high level information per event, calibration procedure (semi-)automatic
- For different type of analyses
  - Many analyses access large fraction of the datasets looking at top level reconstructed objects
  - Exotic analyses (e.g. displace jets) may need alternative reconstruction or detector level information
  - Final plot with few events vs million events
  - Scouting => reading only few information (e.g. jets) at rates above nominal maximum trigger output rate
  - Calibration workflows may need more details, for specific detectors, on specific datasets

Analysis "needs" are not a single target

Tools that are well established

- ROOT
- CVS/SVN/GIT
- RooFit
- Theano/TensorFlow (via Keras or other python things), Caffe
- something else?

- Languages:
  - C++
  - Python
  - (Javascript?)

What is not common

“Analysis framework” - often the trivial book-keeping, interface to a batch system/grid

Sample (re)weighting/normalisation, latest K-factors/NNLO xsections

What does a calibration tool look like, systematic evaluation, EDM (how experiment-specific is an electron?)

Could we propose a common interface for calibration tools as a starting point ?

Getting your data, finding your data (show me a student who didn’t waste weeks on this!)

What can we easily improve

Python is more suitable for fast development, improve integration for analysis? Should we simply adopt python as the language for analysis?

More encapsulation and simpler interfaces, minimise the LOC written for standard tasks, improve code quality by definition, spend more time doing physics

Accept that dream.analyse("my fave semantics == True") won't encapsulate the whole analysis, but components could maybe look more like that

Professional s/w development workflows (CI/jenkins) to be an accepted norm also in analysis software?

What happens outside HEP

Are we sure we are following what is changing in the IT world outside HEP? (in ~2000 we were still using some 50 years old technology called FORTRAN… C/C++ is turning 40/30 y.o.)

- GPU and Parallel stuff: check
- DeepNN : check
- Python: check
- Java: for next gen experiments (?)
- Javascript: ROOT going there
- Less “loops” and more “query like” data analysis:
    - LHCb and Belle II already going in that direction (on the tech side is TDataFrame?)

Senior people lost when we upgrade techs?

Road ahead of us

- Let the machines do more of our job
    - But do not forget how we learned do our job
- Find the patterns, abstract the problems
    - Need to work more on (common) interfaces not just on concrete implementations
- What are the low-hanging fruit?
    - Anything we are just doing wrong for historical reasons that we can easily fix?
- Which problems will require work, but ultimately will have a large pay-off?

(Selection of) Questions

- Can we really get very fast IO on complex formats (not ntuples)?
- Is python going to be **The** Analysis Language? Drawbacks? Is ROOT ready to support this?
- Ready for a common format / toolset for non-event data (as common a TTree for event-data) ? Experiments are moving to REST services, also for analysis?
- Are we approaching GPU/parallel stuff the right way? (how much can we expose details to analysis people)
- Do we need integration of DeepNN stuff in TMVA or should we rather feed our data to existing tools?