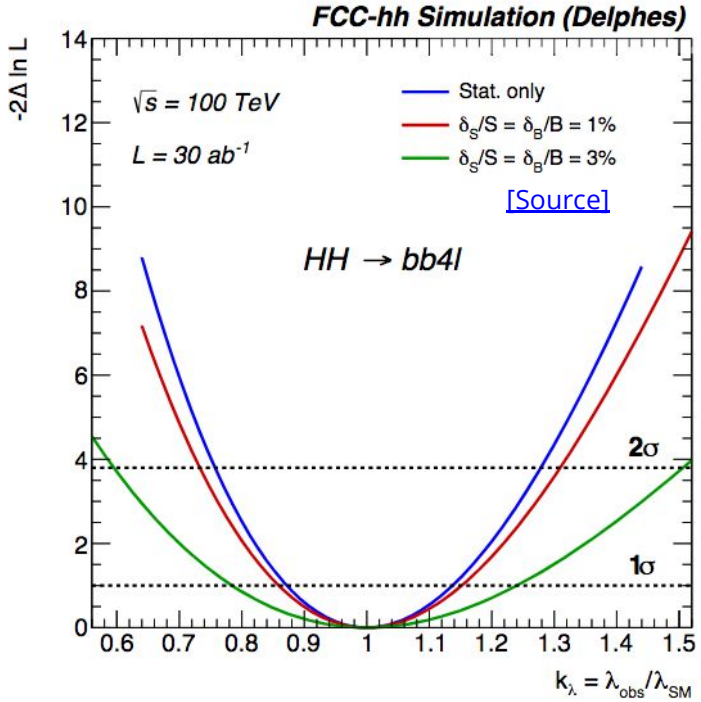
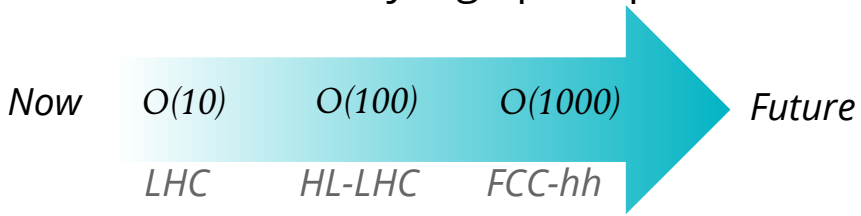
The idea

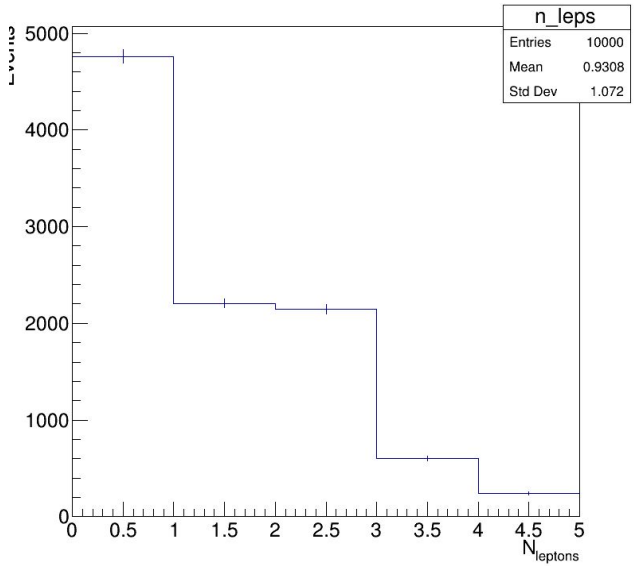


- Study Higgs potential through self coupling **λ** measurement with di-Higgs production at the FCC-hh
  - 20 x better precision than HL-LHC
- Some decay channels studied extensively already, for example $HH \rightarrow bbZZ \rightarrow bb4l$
  - Small background, simple event selection
  - Expected precision 15-24%, for 1-3% systematic uncertainty on bkg and signal
- What about higher BR channels $HH \rightarrow bbZZ \rightarrow bbllvv$, $HH \rightarrow bbZZ \rightarrow bb4v$ or $HH \rightarrow bbWW$?
  - Draw conclusions about MET reconstruction in the face of extremely high pile-up

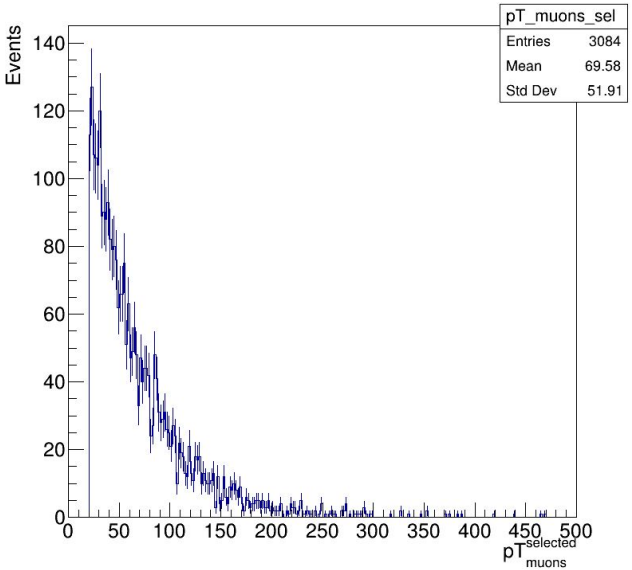


First distributions

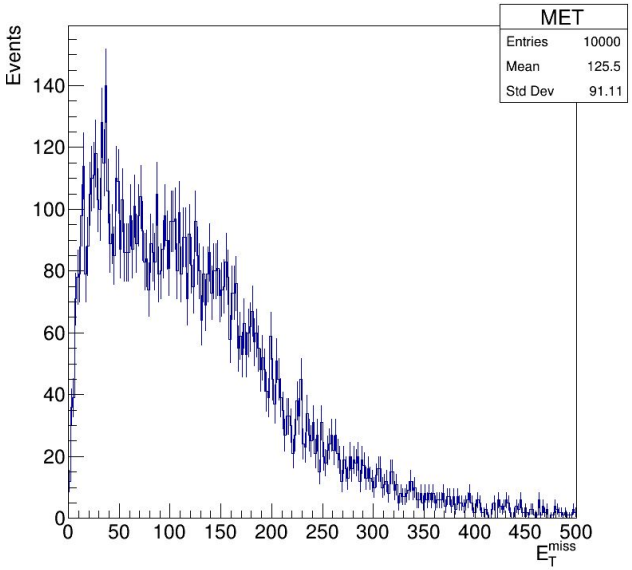
leptons (= electrons and muons)



Muon pT > 20 GeV



Missing transverse energy



- Started to study *HH→bbZZ(leptonic)* events, samples generated with Delphes (by Clement)
  - pp-collisions at 100 GeV
  - Tester with 10k events
- Working on: dilepton mass, filter events by number of neutrinos, compare pT neutrinos to MET, ...

The software

- Using the FCCAnalyses software framework
  - "Is a common tool for analyzing large amount of data using RootDataFrame and produce flat ntuple" - Clement's slides
- Framework structure:
  - C++ "analyzers" for reading the EDM4HEP format events, making them suitable for RDataFrame, implementing common functions for e.g. accessing a reconstructed or generated particle's pT etc.
  - Python interface to write the RDataFrame based specific analysis code, i.e. input/output files, defining event/object selections, filling and writing branches to ntuple
    - The event loop is "hidden" by the RDataFrame syntax, and calculation of (potentially complex) variables are "hidden" in the common C++ code
    - Each usecase (= physics analysis) defines its own analysis.py class, having several examples to run is therefore simple
    - The python files become then more like elaborate config files, in a way

The software

- Using the FCCAnalyses software framework
  - "Is a common tool for analyzing large amount of data using RootDataFrame and produce flat ntuple" - Clement's slides
- Framework structure:
  - C++ "analyzers" for reading the EDM4HEP format events, making them suitable for RDataFrame, implementing common functions for e.g. accessing a reconstructed or generated particle's pT etc.
  - Python interface to write the RDataFrame based specific analysis code, i.e. input/output files, defining event/object selections, filling and writing branches to ntuple
    - The event loop is "hidden" by the RDataFrame syntax, and calculation of (potentially complex) variables are "hidden" in the common C++ code
    - Each usecase (= physics analysis) defines its own analysis.py class, having several examples to run is therefore simple
    - The python files become then more like elaborate config files, in a way

Example analysis class from official git repo

```
class analysis():

    #____________________________________________________________
    def __init__(self, inputlist, outname, ncpu):
        self.outname = outname
        if ".root" not in outname:
            self.outname+=".root"

        ROOT.ROOT.EnableImplicitMT(ncpu)

        self.df = ROOT.RDataFrame("events", inputlist)
        print (" done")
    #____________________________________________________________
    def run(self):

        df2 = (self.df
               .Define("selected_jets", "selRP_pT(50.)(Jet)")
               .Define("jet_pT",        "getRP_pt(Jet)")
               .Define("seljet_pT",     "getRP_pt(selected_jets)")



               )

        # select branches for output file
        branchList = ROOT.vector('string')()
        for branchName in [
                "jet_pT",
                "seljet_pT",

                ]:
            branchList.push_back(branchName)
        df2.Snapshot("events", self.outname, branchList)
```

- Analysis.py can be run standalone on (one or a few) input files
- Build full analysis code around it, that implements pre-/final event selection and can run over (long) list of input files/different processes (e.g. signal and background)

Select objects: Jets with pT > 50 GeV

Define variables: pTs of (selected) jets in the event - note: returns a vector!

Writing the ntuple: Specify which branches to add

Some thoughts

- Advantage of the C++ analyzers + python structure over directly analysing k4SimDelphes output files with e.g. simple python event loop ?
  - Speed ?
  - Implementation that makes Reco-MC links work with RDataFrame ?
- Some problems setting up
  - Git repo version did not work out of the box
  - For running the examples input files come from eos, but the yaml files to load them live in a private afs directory (need to request access)
  - Example code makes assumptions on from where you execute the .py code and will produce error if you deviate from that
  - Documentation is rather terse
    - Structure not super intuitive (at least to me)
- Other concerns:
  - Everything technical is hidden away (good for analysers, but perhaps not what we intend), e.g. only have the final samples, no Delphes cards etc.
  - RDataframe structure makes it harder to debug the physics side?
    - E.g. not quickly possible to print 4vec of electrons+muons in an event to test calculation of a more complex variable like dielectron mass because event loop is hidden ?