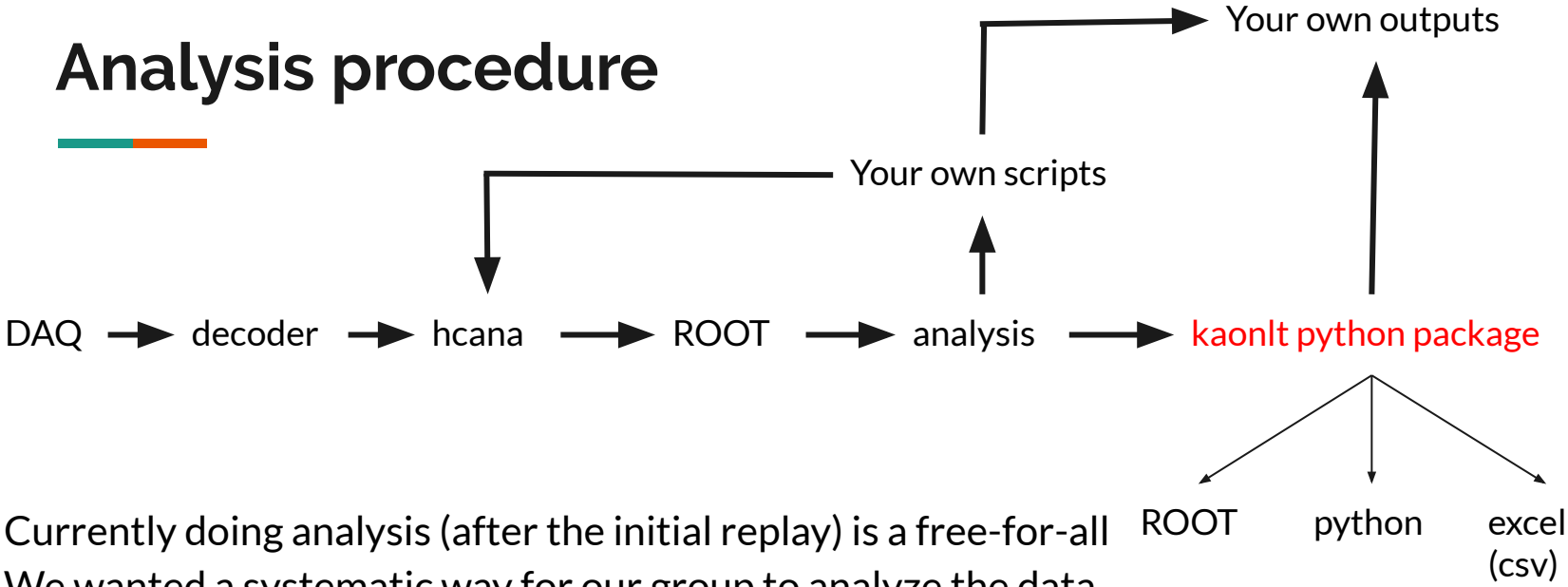
# Kaon LT Status Update

## April 15th, 2020

Richard Trotta

# Analysis procedure




- Currently doing analysis (after the initial replay) is a free-for-all
- We wanted a systematic way for our group to analyze the data
- ROOT gives me a headache so I began doing my analysis in python
- I developed a python package to initially just apply cuts, but quickly Stephen and I expanded this into a full data analysis procedure

# Python Advantages

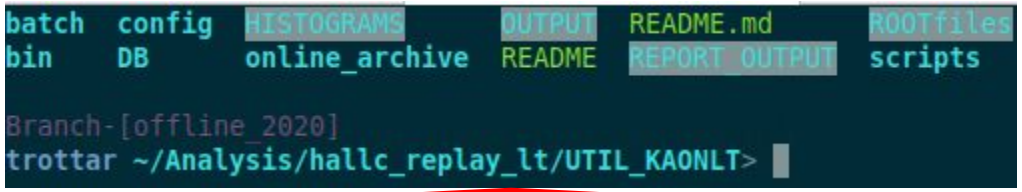

- Code is very readable and syntax is easy to learn
- Debugging is a cinch
  - Large community from a myriad of different fields
- Vast array of third party packages
  - NumPy, SciPy, Numeric, etc.
- Built in types and tools; diminishing C++ woes
- Runs on virtually every major platform used today
- Python programs run in exact same manner irrespective of platform
- See Eric Pooser's 2018 talk for more details…
  - https://redmine.jlab.org/projects/podd/wiki/Workshop2018
- **Easy to grab data from files to use as inputs**
  - This is the central idea behind the kaonlt package

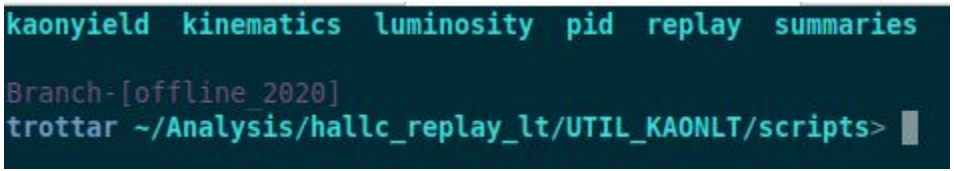
# KaonLT package capabilities

- Easily apply cuts with dynamic cut values
  - e.g. p_track_lumi_before = c.add_cut(P_dc_ntrack,"p_track_lumi_before")
  - Cut values are grabbed from a CSV database (more on this later)
- Adjust bins and create 2D plots easily in python
- If you rather ROOT…
  - One can trivially create slimmed ROOT files of post-analysis plots (See Stephen's talk)
- Define equations and use with easy
  - e.g. mm = missmass(kaon)
  - Still under development

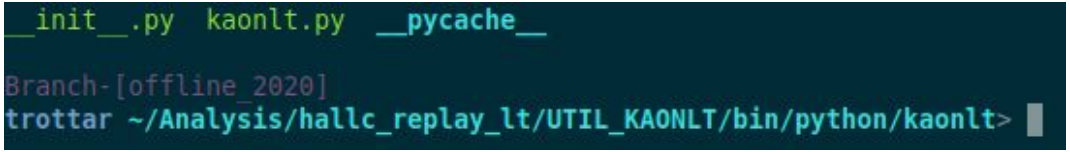
# UTIL_KAONLT Structure

- Correct directory structure is required for kaonlt package to function properly
- Directories of importance for kaonlt
  - DB, scripts, bin



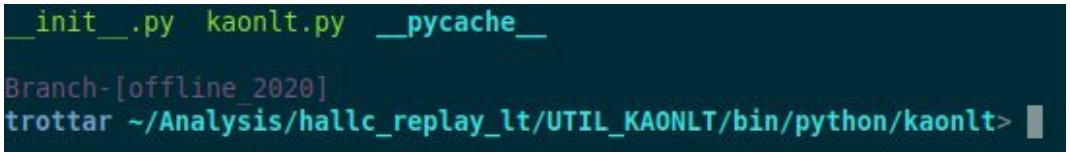





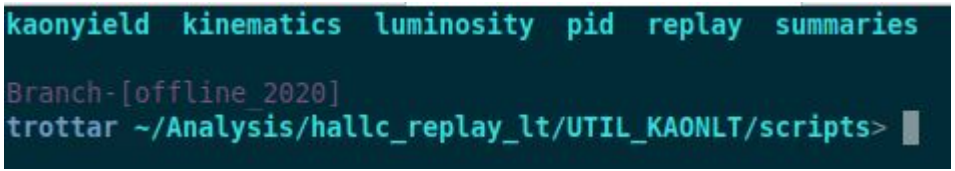


# bin/python/kaonlt

- The bin file is the temporary location of the kaonlt package
  - Once package published (i.e. pip-able) this directory will be removed
- In bin/python/kaonlt/
  - kaonlt.py is where all the methods of importance are defined
  - Pathing is also defined here (need to make more flexible in the future)



# scripts




- Scripts are where the analysis scripts are located
  - e.g. lumi, prod, elastics, etc.
- This directory is pretty much free reign depending on your needs
- Analysis scripts using kaonlt package will need to have a few lines of code to call the package correctly…..

Array name must match what is defined in DB/CUTS/general/ (e.g. H.cal.etotnorm leaf is defined as H_cal_etotnorm)

```
import uproot as up
sys.path.insert(0, 'path_to/bin/python/')
import kaonlt as klt
```

Import uproot and kaonlt

```
# Convert root leaf to array with uproot
array  = tree.array("leaf")
```

Uproot: root to numpy array

```
# Not required for applying cuts, but required for converting back to root files
r = klt.pyRoot()

fout = "<path_to_run_type_cut>"
c = klt.pyPlot(None) # See below for pyPlot class definition
readDict = c.read_dict(fout) # read in run type cuts file and makes dictionary

# This method calls several methods in kaonlt package. It is required to create properly formated
# dictionaries. The evaluation must be in the analysis script because the analysis variables, i.e. the
# leaves of interest, are not defined in the kaonlt package. This makes the system more flexible
# overall, but a bit more cumbersome in the analysis script. Perhaps one day a better solution will be
# implimented.
def make_cutDict(cut,inputDict=None):

    global c

    c = klt.pyPlot(readDict)
    x = c.w_dict(cut)

    # Only for first key of dictionary
    if inputDict == None:
        inputDict = {}

    # Update dictionary with cuts (as strings) from readDict
    for key,val in readDict.items():
        if key == cut:
            inputDict.update({key : {}})

    # Evaluate strings to cut values. Creates a dictionary in a dictionary...dict-ception!
    for i,val in enumerate(x):
        tmp = x[i]
        # Checks for removed leaves
        if tmp == "":
            continue
        else:
            inputDict[cut].update(eval(tmp))

    return inputDict
```

Required for fastest cut method

```
cutDict = make_cutDict("cut1")
cutDict = make_cutDict("cut2",cutDict)
# Continue this for all run type cuts required
# ---> If multple run type files are required then define a new run type file altogether. Do not try
to
# chain run type files. It can be done, but is computationally wasteful and pointless.
```

Define cuts

```
# To apply cuts to array...
c.add_cut(array,"cut#")
```

Apply cuts

** Template analysis script creator is in the works

# Database

```
CUTS  PARAM  Production_Config

Branch-[offline_2020]
trottar ~/Analysis/hallc_replay_lt/UTIL_KAONLT/DB>
```

This is the central hub for all cut definitions, and cut values

1. The kaonlt package grabs the cuts from DB/CUTS/run_type
2. These cuts are broken into general cuts defined in DB/CUTS/general
3. Once the required general cuts are defined, the package searches DB/PARAM/ for the correct cut values (dependent on run number)

```
general  run_type

Branch-[offline_2020]
trottar ~/Analysis/hallc_replay_lt/UTIL_KAONLT/DB/CUTS>
```

```
Acceptance_Parameters.csv  PID_Parameters.csv  README  Timing_Parameters.csv

Branch-[offline_2020]
trottar ~/Analysis/hallc_replay_lt/UTIL_KAONLT/DB/PARAM>
```

# Example

```
coin_prod.cuts      lumi.cuts          pSing_prod.cuts
hSing_optics.cuts   pid_eff.cuts       test
hSing_prod.cuts     pSing_optics.cuts  test.cuts

Branch-[offline_2020]
trottar ~/Analysis/hallc_replay_lt/UTIL_KAONLT/DB/CUTS/run_type>
```

Define the cuts of interest in DB/CUTS/run_type/coin_prod.cuts

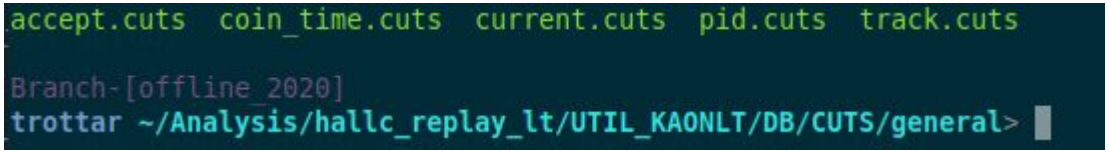
```
##############
# COIN picut #
##############
coin_epi_cut = pid.p_picut+accept.delta+accept.h_pfp+accept.p_pfp
#
coin_epi_cut_orig = pid.p_picut+accept.delta+accept.h_pfp+accept.p_pfp-pid.p_picut.P_cal_etotnorm-pid.p_picut.H_cer_npeSum
```

Breaking it down…

- pid.p_picut+accept.delta+accept.h_pfp+accept.p_pfp are added cuts
  - Looking at pid.p_picut...pid tells kaonlt the type of general cut to be added, p_picut tells kaonlt the specific cut in the general cuts to be added (similarly for accept.delta)
- -pid.p_picut.P_cal_etotnorm-pid.p_picut.H_cer_npeSum are subtracted cuts
  - Looking at -pid.p_picut.P_cal_etotnorm...pid.p_picut are the same as above but now subtracted. P_cal_etotnorm tells kaonlt the detector of interest to subtract

To summarize “+” means add cuts, “-” means subtract cut, any combination of cuts can be achieved from this

# Example




Define the general cuts in DB/CUTS/general/pid.cuts

```
##############
# SHMS pid cuts #
##############
p_picut = {"H_cal_etotnorm" : (H_cal_etotnorm > pid.P_picut_H_cal)},{"H_cer_npeSum" : (H_cer_npeSum > pid.P_picut_H_cer)},
{"P_gtr_beta" : ((abs(P_gtr_beta)-1) < pid.P_picut_P_beta)}, {"P_hgcer_npeSum" : (P_hgcer_npeSum > pid.P_picut_P_hgcer)},
{"P_aero_npeSum" : (P_aero_npeSum > pid.P_picut_P_aero)}, {"P_cal_etotnorm" : (P_cal_etotnorm > pid.P_picut_P_cal)}
```

Breaking it down…

- pid.p_picut (from the last slide)
  - Kaonlt goes into DB/CUTS/general/pid.cuts and selects the specific cut (i.e. p_picut)
  - Above you can see all the defined detector cuts
- If I wanted to subtract the HMS cal cut I would simply put -pid.p_picut.H_cal_etotnorm in DB/CUTS/run_type/coin_prod.cuts
- The variable names (e.g. pid.P_picut_P_cal) are grabbed from DB/PARAM

# Example

```
Acceptance_Parameters.csv  PID_Parameters.csv  README  Timing_Parameters.csv

Branch-[offline_2020]
trottar ~/Analysis/hallc_replay_lt/UTIL_KAONLT/DB/PARAM>
```

Define the cuts values in DB/CUTS/PARAM/PID_Parameters.csv

| Run_Start | Run_End | H_ecut_H_cal | H_ecut_P_cal | H_ecut_H_beta | H_ecut_H_cer | H_picut_H_cal | H_picut_P_cal | H_picut_H_beta | H_picut_H_cer | H_hac |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 9999 | 0.7 | 0.7 | 0.3 | 1.5 | 0.7 | 0.7 | 0.3 | 1.5 | 0.7 |

Breaking it down…

- DB/CUTS/general/pid.cuts (from the last slide)
  - The variable pid.P_picut_P_cal specifies where kaonlt should get the cut value
  - Similar to DB/CUTS/run_type...pid tells kaonlt which parameter file and P_picut_P_cal tells kaonlt the cut value
  - The value grabbed is based off the run number

# Bring it all together

- This may seem like a lot, but the beauty is that all of this is done behind the scenes.
- As a user you will only need to edit…
  - The analysis script in scripts/
  - The cut definitions in DB/CUTS/run_type/
  - The cut values in DB/PARAM
- This means if something goes wrong it is only in one of three places
- All of your final analysis can be converted into super slim ROOT files
  - Multiple runs can be chained together with ease as well
- Therefore the kaonlt package provides…
  - Easy debugging
  - More flexibility applying cuts on a run by run basis
  - Easier to see which cuts have been applied
  - Slimmer and less repetitive scripts
  - Very small ROOT files for easy tradability among groups

**See Stephen's talk for even more details**

# Why such an elaborate cut procedure?

- Python is very slow compared to C++ but there are ways to shorten this gap
- One could easily apply cuts like…

```
for val in arr:
    if cut1:
        if cut2:
            new_arr.append(val)
```

- But this is very slow…. (~$6.1 \times 10^{-2}$ seconds for 50k events)
- Once could speed this up…. (~$5.7 \times 10^{-2}$ seconds for 50k events)

```
new_cut = [val for val in arr
           if cut1
           if cut2]
```

- But even this is slow compared to C++
- There are many faster ways, one of which is array indexing
- Array indexing is how my cuts are applied (~$3.5 \times 10^{-4}$ seconds for 50k events)

```
new_arr = arr[cut1 & cut2]
```

# Current state

- Only tracking cuts still need to be defined
- Farm testing needs to be done
- Need to fix some minor naming schemes (e.g. gtr_th->gtr_xp)
- Better way of applying pid cuts when there is a max and min cut
- The kaonlt package is still in DEBUG mode.
  - Once all cuts are in this will be changed
  - I want to make it super obvious where an issue arises and how to fix it by including detailed error messages.
- Currently only 1D arrays are accepted as cut inputs, but want to expand to multidimensional arrays as well
  - Note: multidimensional arrays still work, they just need to be looped over in the analysis script

# Looking to the future

- A requirements.txt files needs to be included to assure python package requirements are met
  - Also may need to check ROOT versions (ROOT 6.14+ is required for most pyROOT type packages)
- Comprehensive speed test
- Add commonly used equations to kaonlt (e.g. missing mass)
- Include a template analysis script creator to assure proper syntax, plus convenience
- Pathing in kaonlt is rather rigid so this needs to be expanded
  - After this, publish kaonlt package so it can be pip-ed and remove bin directory
- (Specific to KaonLT group), incorporate Bill's code into the framework