# ExpressReco Sampling and Performance Test

**Daniel Jacobi**
dajaco@uni-bonn.de

Physikalisches Institut der Rheinischen Friedrich-Wilhelms-Universität Bonn

Belle II Trigger/DAQ Workshop 01. Dezember 2022

Belle II

Bundesministerium für Bildung und Forschung

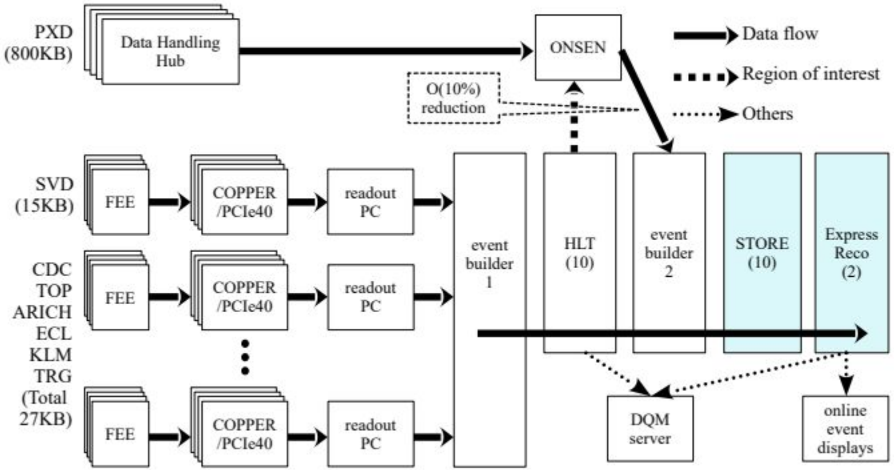
ErUM-FSP T09 Belle II

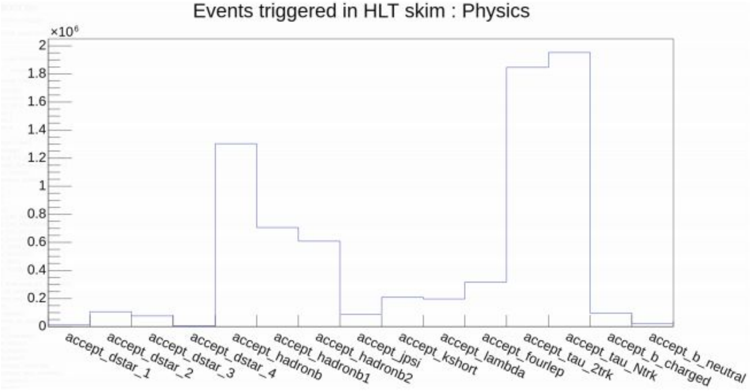
BLUB
Belle group
University of Bonn

UNIVERSITÄT BONN

# Introduction

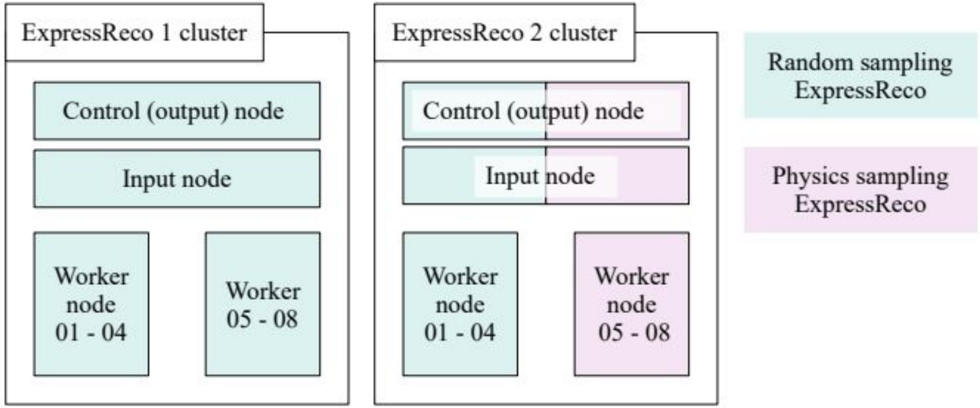
- Store is sending randomly sampled events to ExpressReco (ERECO) for monitoring
- Low statistics for rare physics tagged events leads to inefficient monitoring



# The Plan

- Prepare two ERECO clusters
  - Random sampling for PXD related monitoring
  - Random and physics sampling ➪ Monitor entire runs
- Number of worker nodes depending by ERECO performance and selected trigger lines




Main Task of this performance study:

- Depending on the selected trigger lines, how many worker nodes do we need?

# The Script, Calculations and First Results

# The Script and Calculations

- Adapted the *PrintCalibTriggerResults* module
- We can look at any given combination of used trigger lines (skims)

For this talk

- Ran over different runs from different experiments (runtime > 1h)
- Chose 8 different skims and their combinations

- We want to calculate the trigger rate for certain skims

$$\text{Rate}_{\text{Skim}} = \frac{\text{Events passing skim}}{\text{Time retrieved from RunDB}}$$

How many ERECO worker nodes do we therefore need

$$\text{WN}_{\text{Needed}} = \frac{\overline{\text{Rate}}_{\text{Skim}}}{\text{Rate}_{\text{WN is capable of}}}$$

```
class PrintCalibTriggerResults(b2.Module):

    '''
    Prints Calibration trigger results in a well formatted way.
    User is prompted to continue or quit at each event
    '''

    def event(self):
        '''
        Print log likelihoods and wait for user respond.
        '''

        evtMetaData = Belle2.PyStoreObj('EventMetaData')
        exp = evtMetaData.obj().getExperiment()
        run = evtMetaData.obj().getRun()
        evt = evtMetaData.obj().getEvent()
```

# Trigger Lines of Interest

## software_trigger_cut&skim&accept_dstar1/2/3/4

| Particles | Cuts |
|---|---|
| $\pi$ list | $\lvert d_0\rvert < 2\,\mathrm{cm}$<br>$\lvert z_0\rvert < 4\,\mathrm{cm}$ |
| $K$ list | $\lvert d_0\rvert < 2\,\mathrm{cm}$<br>$\lvert z_0\rvert < 4\,\mathrm{cm}$ |
| $\gamma$ list | $0.296706 < \theta(\mathrm{rad}) < 2.61799$<br>[[clusterReg = 1 and E > 0.03 GeV] or [clusterReg = 2 and E > 0.02 GeV] or [clusterReg = 3 and E > 0.03 GeV]] and \|clusterTiming\| < 1.0 · clusterErrorTiming or E > 0.1 GeV] and [clusterE1E9 > 0.3 or E > 0.1 GeV] |
| $\pi^0$ list | $0.075 < \mathrm{M}\ (\mathrm{GeV}/c^2) < 0.175$<br>successful mass fit (`kFit`) |
| $D^0$ list | $1.7 < \mathrm{M}\ (\mathrm{GeV}/c^2) < 2.1$ |
| $D^*$ list | $p^* > 2.2\ \mathrm{GeV}/c$<br>$\mathrm{dM} < 0.16\ \mathrm{GeV}/c^2$ |
| **Event** | **Cuts** |
| `hadron` | $= 1$ |
| n($D^*$) | $> 0$ |

Cuts corresponding to `Dstpi == 1` ($i = 1, 2, 3$ for respectively the channels $K\pi$, $K\pi\pi^0$ and $K3\pi$).

## software_trigger_cut&skim&accept_jpsi

| Track List | $J/\psi$ List |
|---|---|
| $\lvert d_0\rvert < 2\,\mathrm{cm}$<br>$\lvert z_0\rvert < 4\,\mathrm{cm}$<br>$p_t > 0.2\,\mathrm{GeV}/c$ | $\lvert dM\rvert < 0.11\ \mathrm{GeV}/c^2$ |

## software_trigger_cut&skim&accept_kshort

The $K_S^0$ list is built from the *standard K_S0:merged* list, obtained with the vertex fit performed by *kFit*. The sample is purified by adding the conditions:
*0.468 < M < 0.528 and goodBelleKshort == 1*

## software_trigger_cut&skim&accept_mumutight

| | Pion List | Photon List | Event |
|---|---|---|---|
| Cuts | $\lvert d_0\rvert < 2\,\mathrm{cm}$<br>$\lvert z_0\rvert < 4\,\mathrm{cm}$<br>$clusterE < 0.5\,\mathrm{GeV}$<br>$p_t > 0.2\,\mathrm{GeV}/c$<br>$p^* > 0.5\,\mathrm{GeV}/c$ | $E > 0.1\,\mathrm{GeV}$ | Tot. Energy (ECL)< 2<br>n. Tracks= 2<br>Pion List size= 2<br>$\lvert\lvert\theta^*_+ + \theta^*_-\rvert - 180°\rvert < 10°$<br>$\lvert 180° - \lvert\phi^*_+ - \phi^*_-\rvert\rvert < 10°$ |

## software_trigger_cut&skim&accept_lambda

The list of lambda candidates is built from the *Lambda_0:merged* list, which makes use of *TreeFitter* with the additional cuts:
$(p_p - p_\pi) / (p_p - p_\pi) > 0.41$ and
*flightDistance / flightDistanceErr > 10*

Full explanation and definition can be found here: https://docs.belle2.org/record/1965/files/BELLE2-NOTE-TE-2020-018.pdf

# The Script

```python
class PrintCalibTriggerResults(Module):

    # The idea of the module is to count the events
    # from a real data file which pass certain skims

    nevts = 0

    def event(self):

        evtMetaData = Belle2.PyStoreObj('EventMetaData')

        exp = evtMetaData.obj().getExperiment()
        run = evtMetaData.obj().getRun()
        evt = evtMetaData.obj().getEvent()

        event_info['Exp'] = exp
        event_info['Run'] = run

        PrintCalibTriggerResults.nevts += 1

        trigger_result = (Belle2.PyStoreObj('SoftwareTriggerResult').getResults())

        wanted_vars = [ 'software_trigger_cut&skim&accept_dstar_1',
                        'software_trigger_cut&skim&accept_dstar_2',
                        'software_trigger_cut&skim&accept_dstar_3',
                        'software_trigger_cut&skim&accept_dstar_4',
                        'software_trigger_cut&skim&accept_mumutight',
                        'software_trigger_cut&skim&accept_jpsi',
                        'software_trigger_cut&skim&accept_kshort',
                        'software_trigger_cut&skim&accept_lambda',]

        for name, result in trigger_result:

            if name in wanted_vars:

                if result == 1:

                    global results

                    results[name] += 1

                    global double_count

                    double_count[name] = 1

                else:
                    double_count[name] = 0
```

- Loop over every event in a given run and experiment
- Retrieve metadata for later identification
- Get the Software Trigger Results
- Trigger lines of interest
- Loop over all trigger results
- If event passed certain skim (*result == 1*) we add them up
- Need this at a later point

# The Script

- After we loop over every event we write the results to a root file
- We use this file to run over real data events registered on the grid
- We then can use the output to study the performance

```
Experiment = event_info['Exp']
Run = event_info['Run']
Events = PrintCalibTriggerResults.nevts

accept_dstar_1 = results['software_trigger_cut&skim&accept_dstar_1']
```

```
# Minimal Example
outfile = TFile( outputFile, 'RECREATE', 'ROOT file with an NTuple' )

results_0 = [Experiment, Run, Events, accept_dstar_1]

labels_0 = ['Experiment','Run','Events','accept_dstar_1']

df_0  = TNtuple( 'df_0', '', ':'.join(labels_0) )
row_0 = map( float, results_0 )
df_0.Fill(*row_0)

outfile.Write()
```

| Experiment | Run | Events | accept_dstar_1 |
|---|---|---|---|
| 14.0 | 2133.0 | 1427181.0 | 362.0 |
| 18.0 | 1984.0 | 182155.0 | 297.0 |
| 18.0 | 875.0 | 286763.0 | 410.0 |
| 18.0 | 998.0 | 271064.0 | 471.0 |
| 24.0 | 1447.0 | 258389.0 | 464.0 |
| 24.0 | 1447.0 | 261232.0 | 456.0 |
| 20.0 | 874.0 | 383523.0 | 564.0 |
| 18.0 | 1984.0 | 181280.0 | 288.0 |

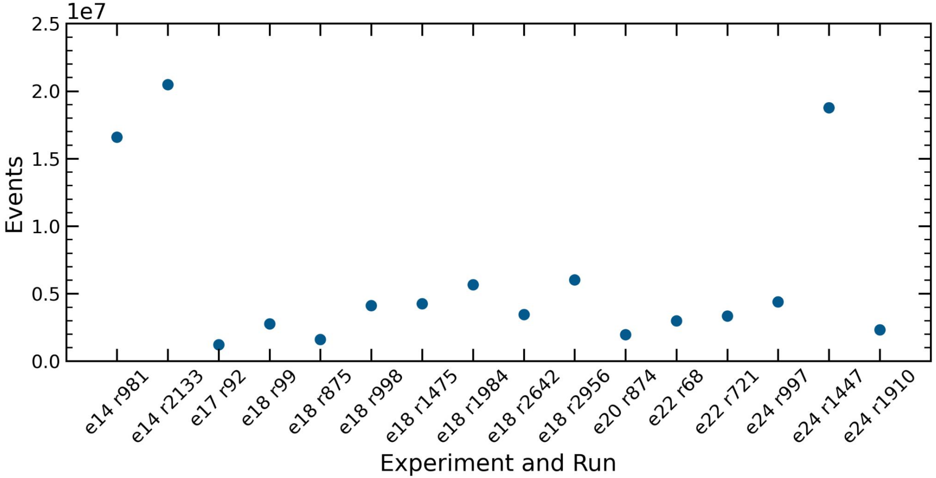
- One entry per file
- Output from two different registered files but for same experiment and run
  → We need to add them up

# The Script

- Generate dictionary where we have one entry for each experiment and run
- All related entries are added up
- Now we have all the information, do the calculations and look at the results

| | Experiment | Run | Events | accept_dstar_1 | accept_dstar_1&accept |
|---|---|---|---|---|---|
| 0 | 24.0 | 1447.0 | 18776206.0 | 34469.0 | |

| | Experiment | Run | Events | accept_dstar_1 | accept_dstar_1&accept_c |
|---|---|---|---|---|---|
| 0 | 20.0 | 874.0 | 1968165.0 | 2842.0 | |




```python
list_exp = list(dict.fromkeys((DF_all.Experiment).values.flatten().tolist()))

exp_dict = {}
for name in list_exp:
    exp_dict[name] = pd.DataFrame()
    exp_dict[name] = DF_all[DF_all['Experiment'] == name]


run_dict = {}


for i in list_exp:

    run_list = list(dict.fromkeys((exp_dict[i].Run).values.flatten().tolist()))

    for var in run_list:

        df = exp_dict[i]

        run_dict[i,var] = pd.DataFrame()
        run_dict[i,var] = df[df['Run'] == var]

        df_len = len(run_dict[i,var])

        run_dict[i,var] = run_dict[i,var].sum().to_frame().T
        run_dict[i,var]['Run'] = run_dict[i,var]['Run']/df_len
        run_dict[i,var]['Experiment'] = run_dict[i,var]['Experiment']/df_len
```

# The Calculations

- Reminder: $\text{Rate}_{\text{Skim}} = \dfrac{\text{Events passing skim}}{\text{Time retrieved from RunDB}}$

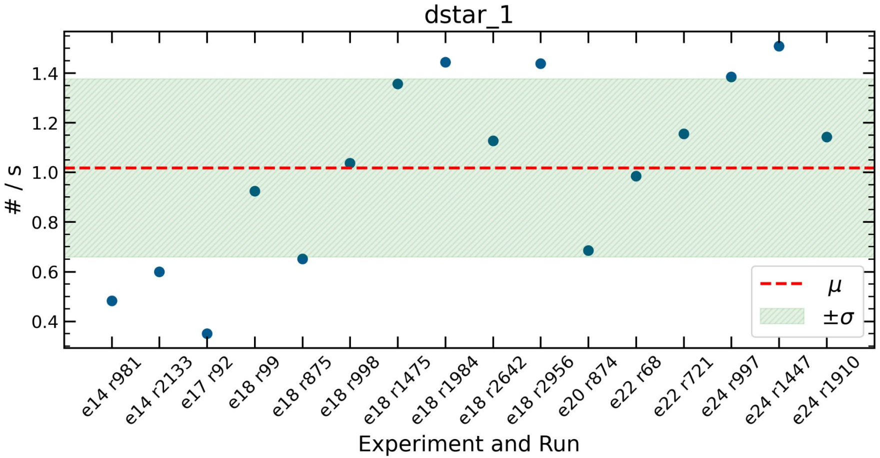
```
def calc_sec(h,m,s):
    return h*60*60+m*60+s
```

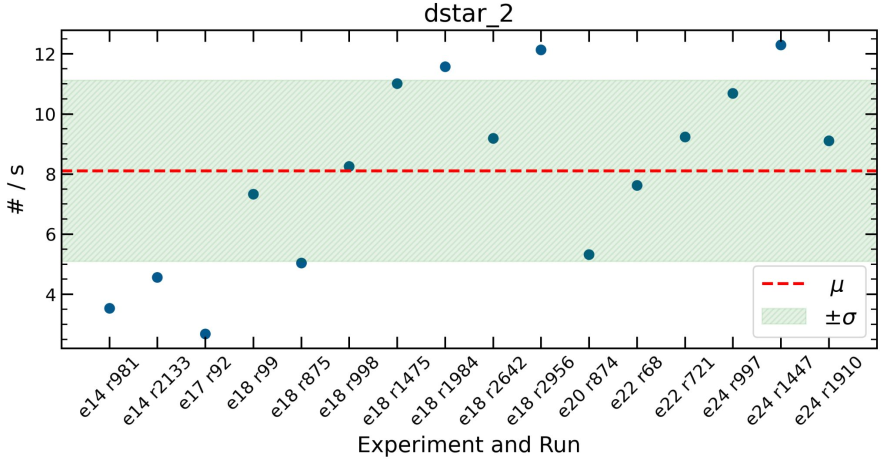
```
run_dict[(24.0, 1910.0)]["time"] = calc_sec(1,1,9)
run_dict[(24.0, 1447.0)]["time"] = calc_sec(6,20,37)
run_dict[(24.0, 997.0)]["time"] = calc_sec(1,37,52)
```

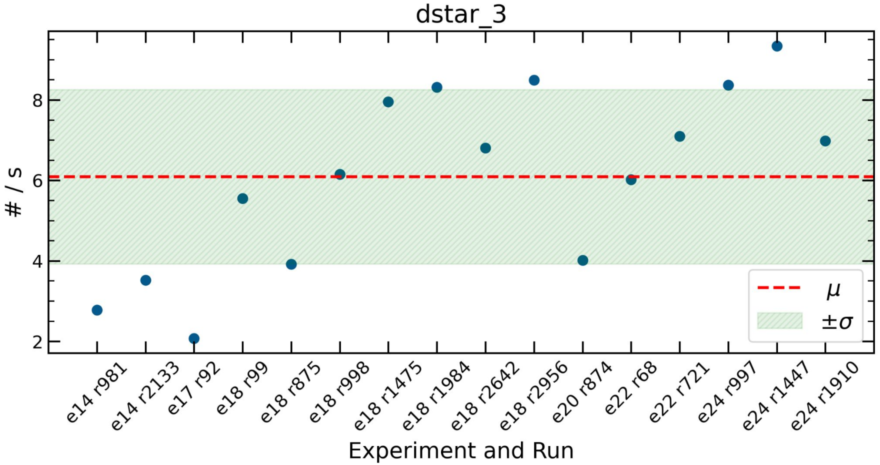
```
def calculations(df):

    df['factor'] = 1/df['time']

    df["dstar_1"] = (df["accept_dstar_1"]) * df["factor"]
    df["dstar_2"] = (df["accept_dstar_2"]) * df["factor"]
```

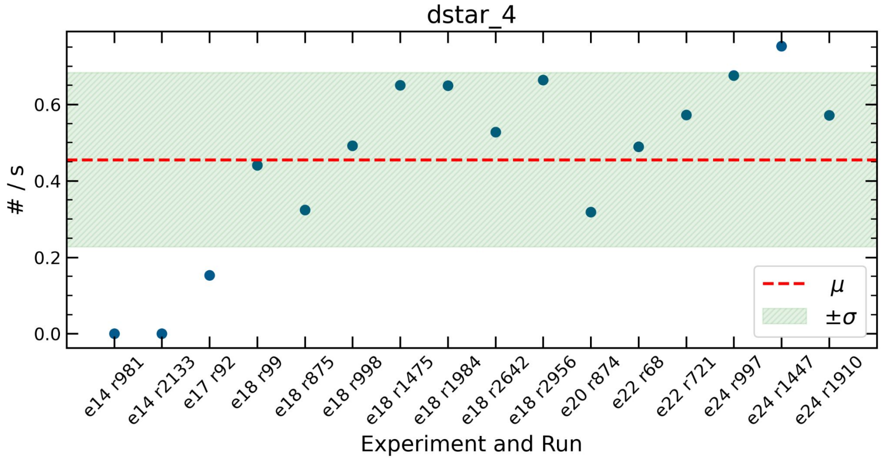
| | |
|---|---|
| Run Type<br>Run type as reported by run control | physics |
| Start time<br>Start time of the run in local time (JST) | 2022-04-12 22:45 |
| Stop time<br>Stop time of the run in local time (JST) | 2022-04-12 23:46 |
| Run Time<br>Time between run start time and stop time | 1:01:09 |

# Results for Trigger Lines of Interest

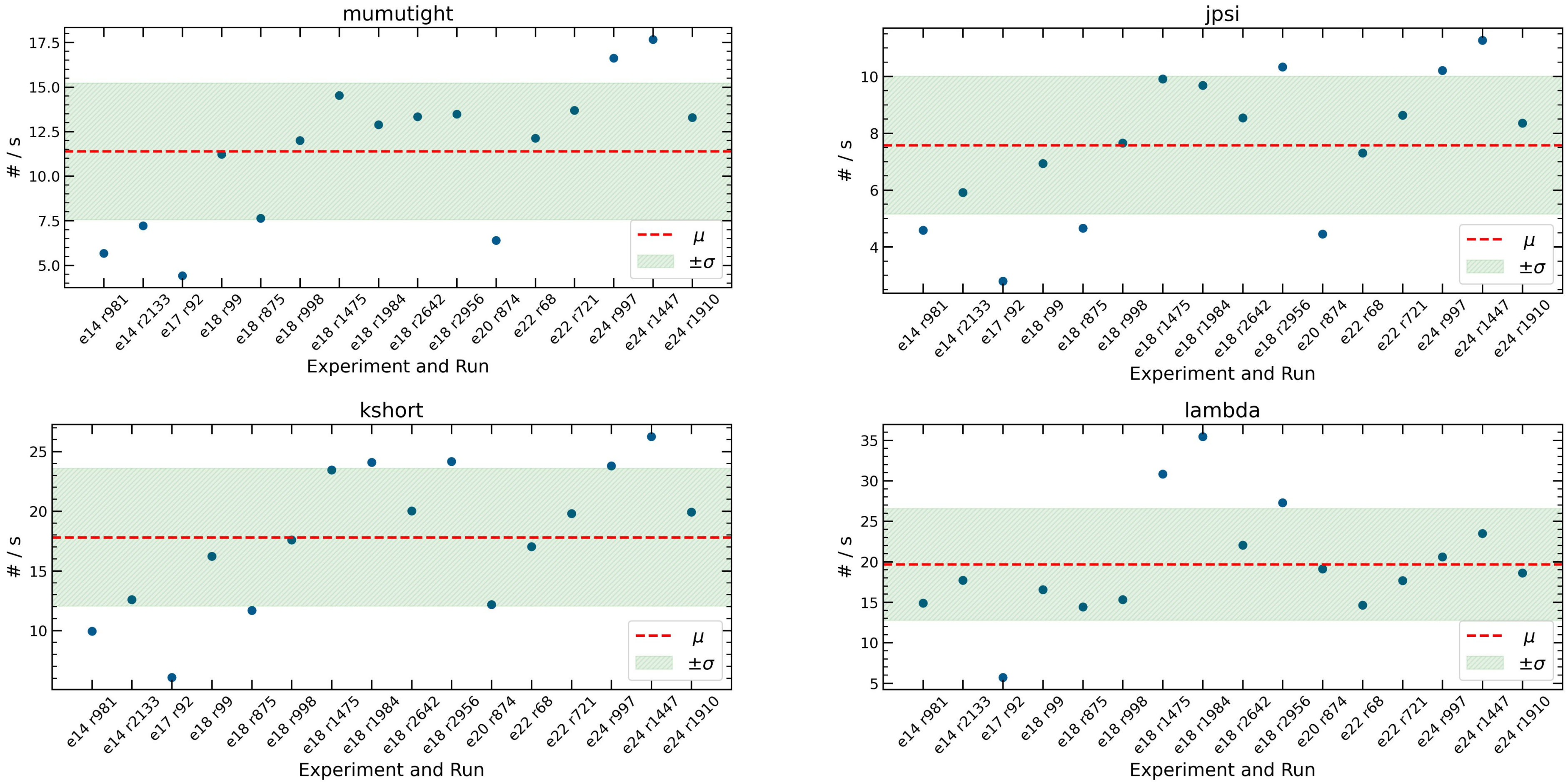


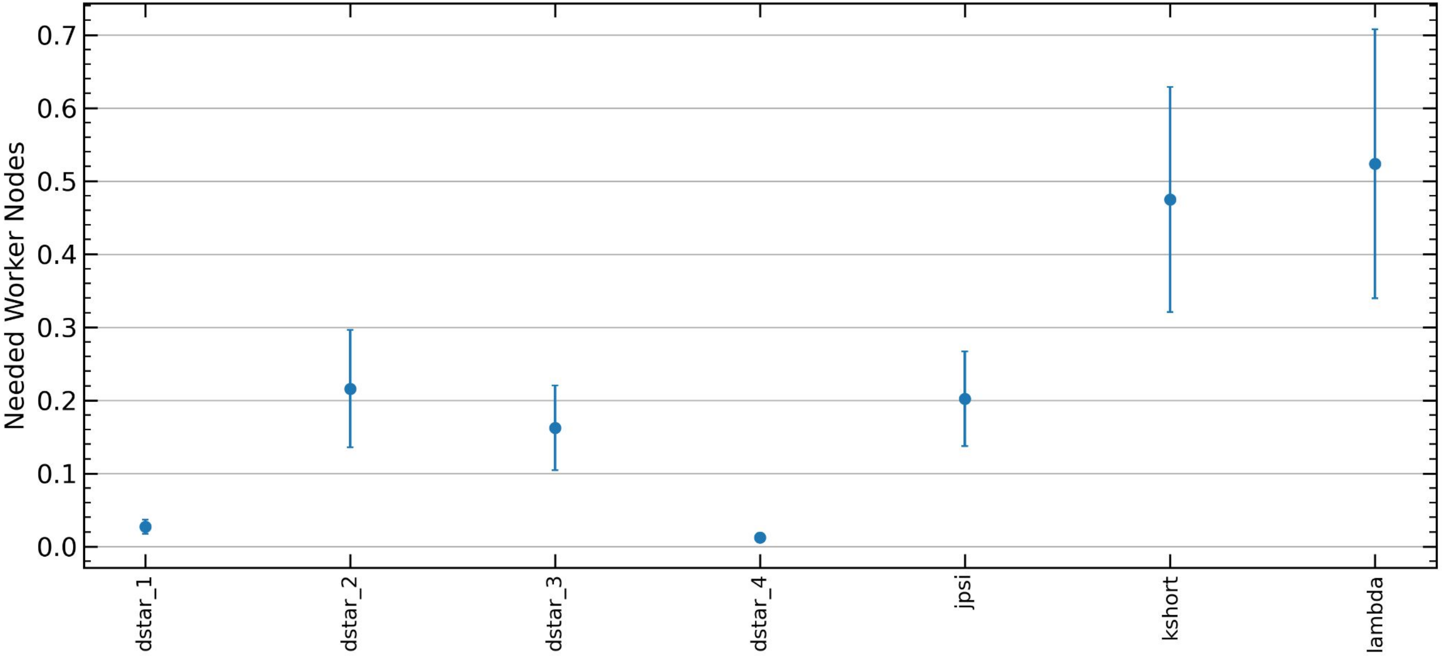
# Results for Trigger Lines of Interest



# How many Worker Nodes do we need

- We have:
  - Two ERECO with 8 worker nodes each
  - One ERECO can deal with 300 events per s
  - Therefore one worker node can deal with 37.5 events per s

$$\mathrm{WN}_{\text{Needed}} = \frac{\overline{\mathrm{Rate}}_{\text{Skim}}}{\mathrm{Rate}_{\text{WN is capable of}}}$$



# Some Sanity Checks

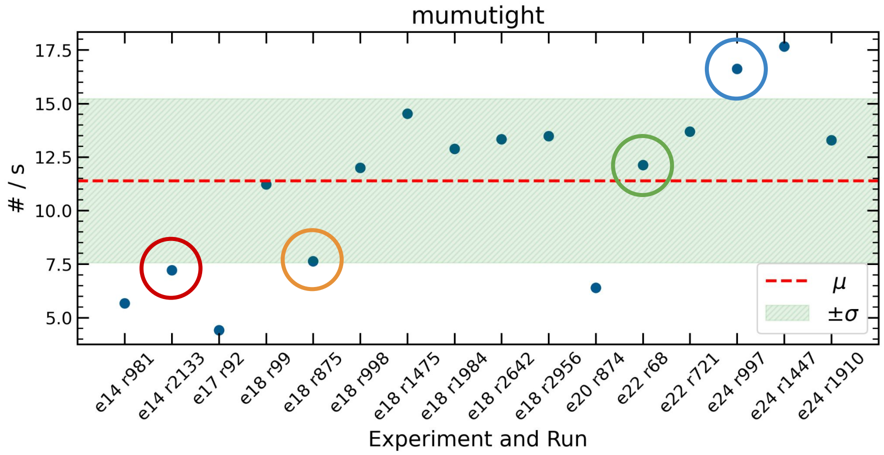
# Check: Script versus RunDB

- Compare the $\mu\mu_{Tight}$ Events registered in the RunDB to the ones calculated with the script
- Overall good agreement

### HLT Skim Information

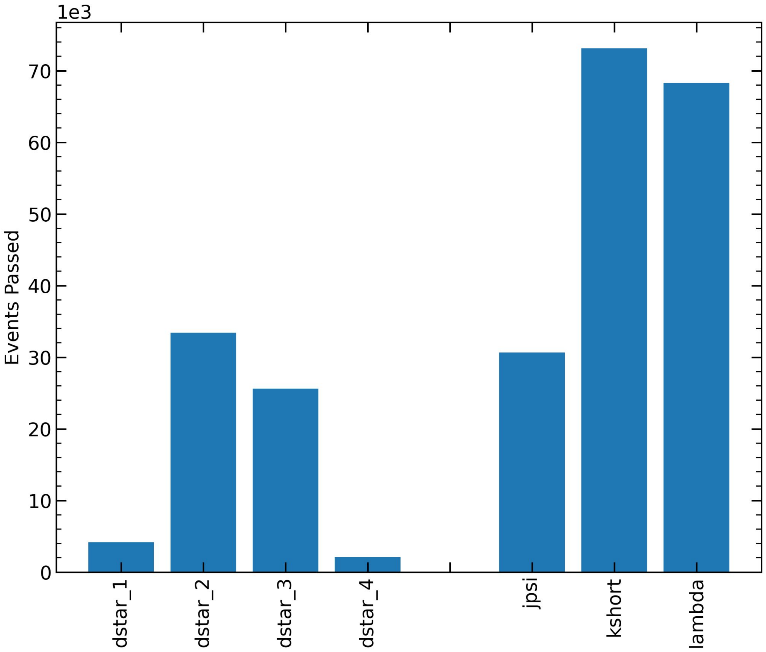
| Name | Events |
|---|---|
| mumutight | 56744 |

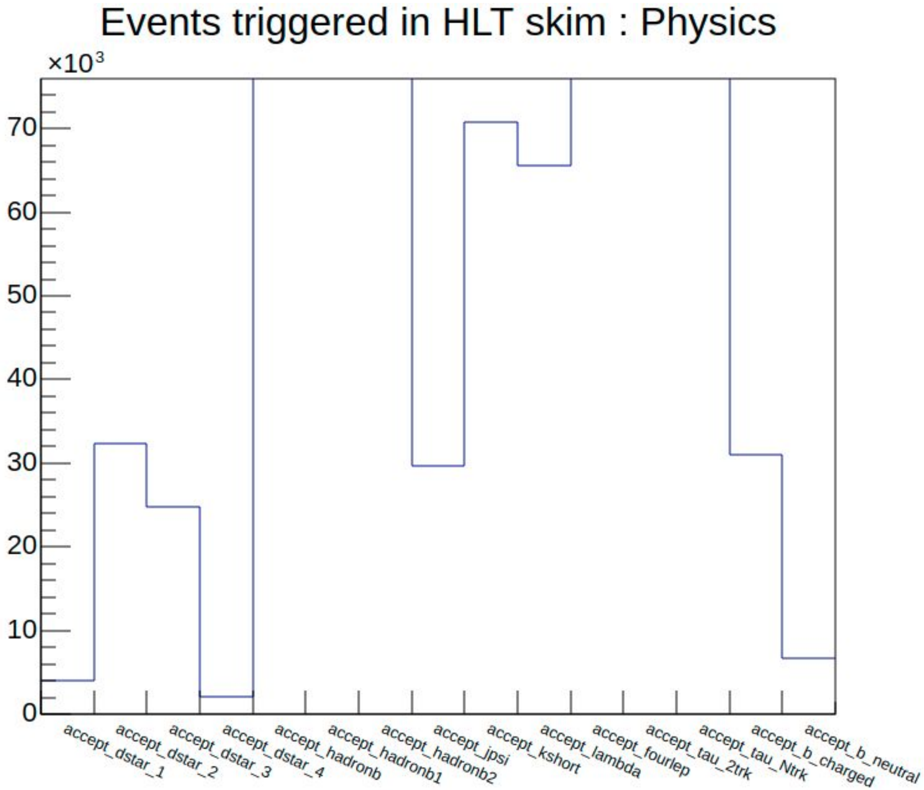
| | Exp. | Run | Run Time | $\mu\mu_{Tight}$ Events | #/s (RunDB) |
|---|---|---|---|---|---|
| (red) | 14 | 2133 | 2:10:41 | 56744 | 7.24 |
| (orange) | 18 | 875 | 1:05:18 | 29904 | 7.63 |
| (green) | 22 | 68 | 1:25:40 | 62371 | 12.13 |
| (blue) | 24 | 997 | 1:37:52 | 97601 | 16.62 |



# Check: Script versus DQM

- Compare the events registered in the DQM to the ones calculated with the script



# Additional Studies and Results

# Combination of Different Trigger Lines

```
for name, result in trigger_result:
    if name in wanted_vars:
        if result == 1:
            global results
            results[name] += 1
            global double_count
            double_count[name] = 1
        else:
            double_count[name] = 0
```

- We want to look at the combination of different trigger lines (*or* logic)
- For each event, set a flag if skim is passed
- If certain combinations are all flagged, add the events up
- 8 skims → 255 possible combinations
  → Code to generate the text

```
global results_double

if double_count["software_trigger_cut&skim&accept_dstar_1"] == 1 and double_count["software_trigger_cut&skim&accept_dstar_2"] == 1:
    results_double["accept_dstar_1&accept_dstar_2"] += 1
if double_count["software_trigger_cut&skim&accept_dstar_1"] == 1 and double_count["software_trigger_cut&skim&accept_dstar_2"] == 1 and double_count["software_trigger_cut&skim&accept_dstar_3"] == 1:
        results_double["accept_dstar_1&accept_dstar_2&accept_dstar_3"] += 1
```

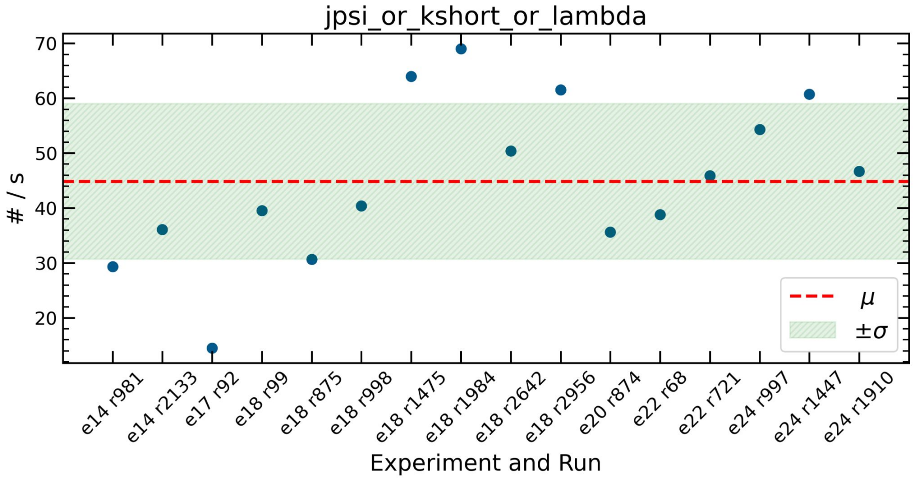
- Why do we need this? We could just add up the events passing $skim_1$ and events passing $skim_2$ ?
  → One event could pass both skims (avoid double counting)

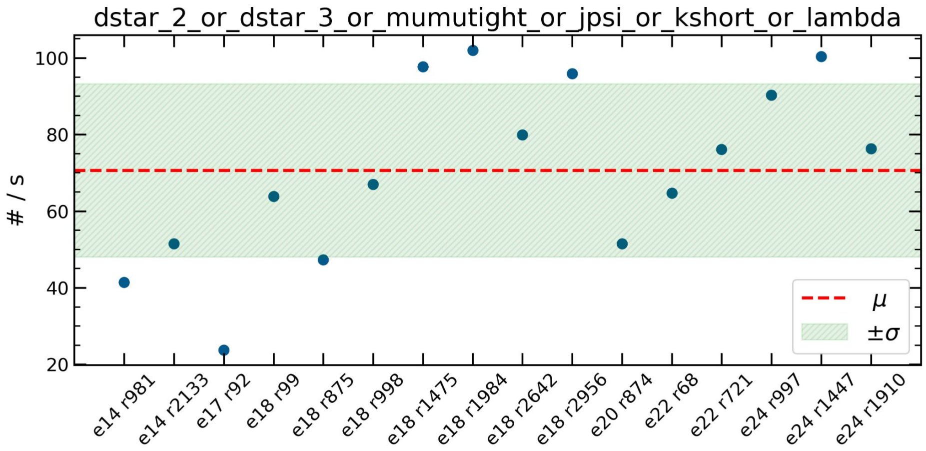
Example:

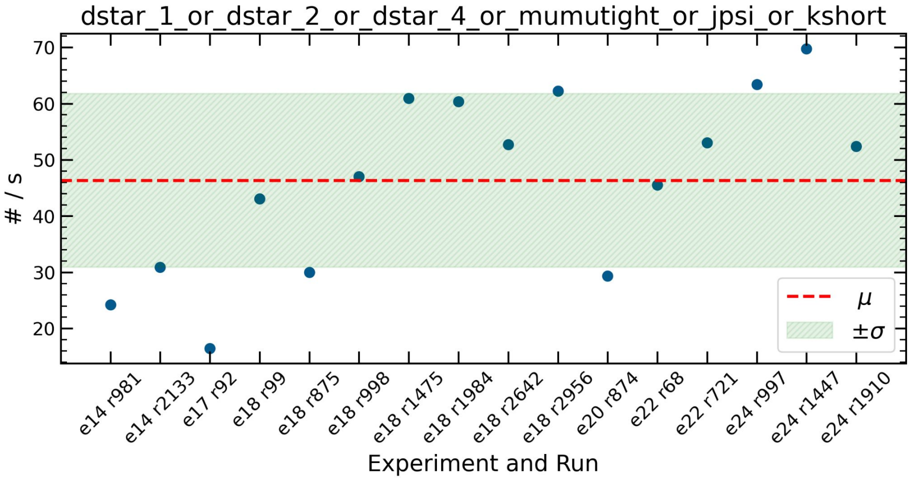
5 events pass $skim_1$ (2 of them also pass $skim_2$)
10 events pass $skim_2$
Events passing $skim_1$ or $skim_2$ → 5 + 10 - 2 = 13 events pass $skim_1$ *or* $skim_2$

# Combinations of Different Trigger Lines

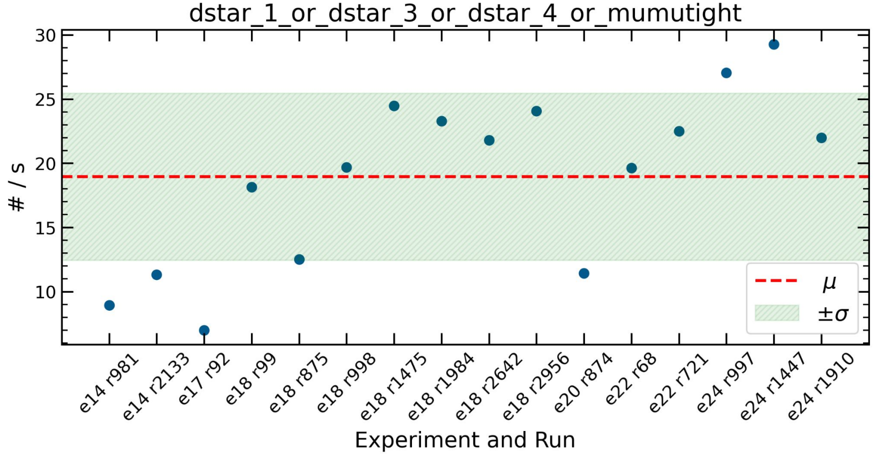
- Now we can have a look at any given combination of trigger lines
- 8 skims used → 255 possible combinations

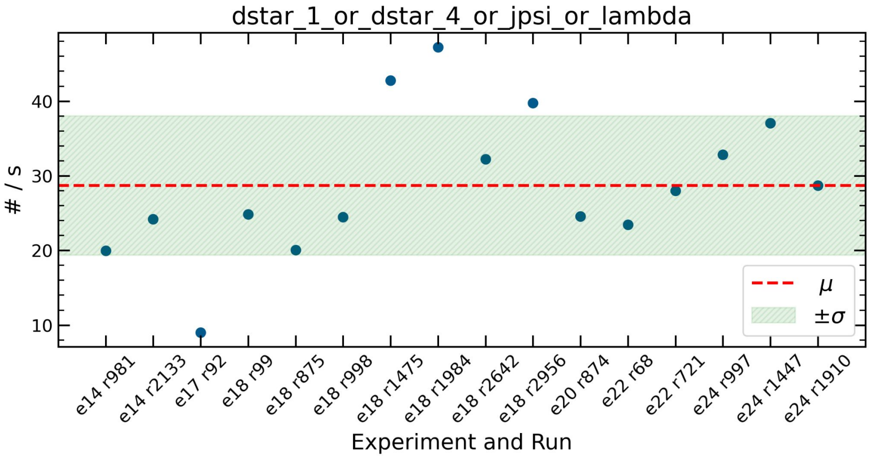


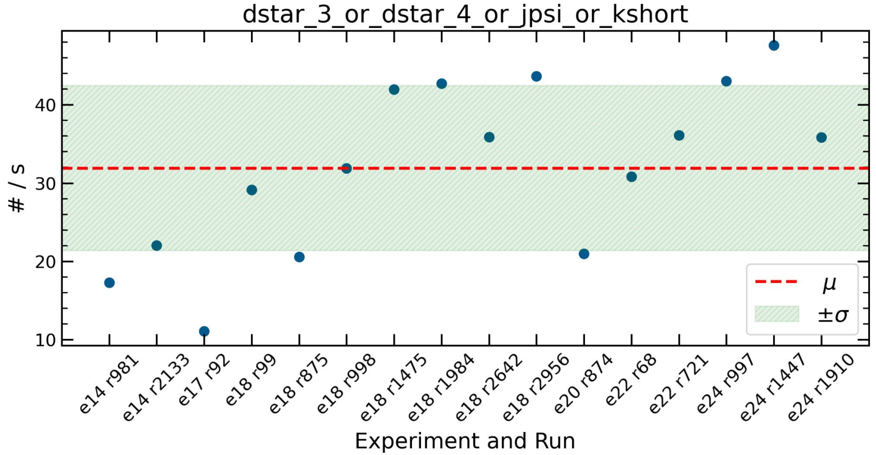



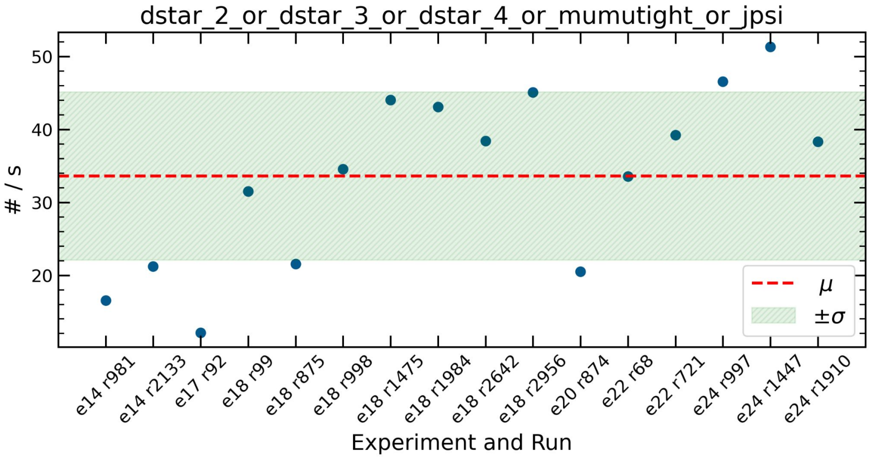



# Combinations of Different Trigger Lines

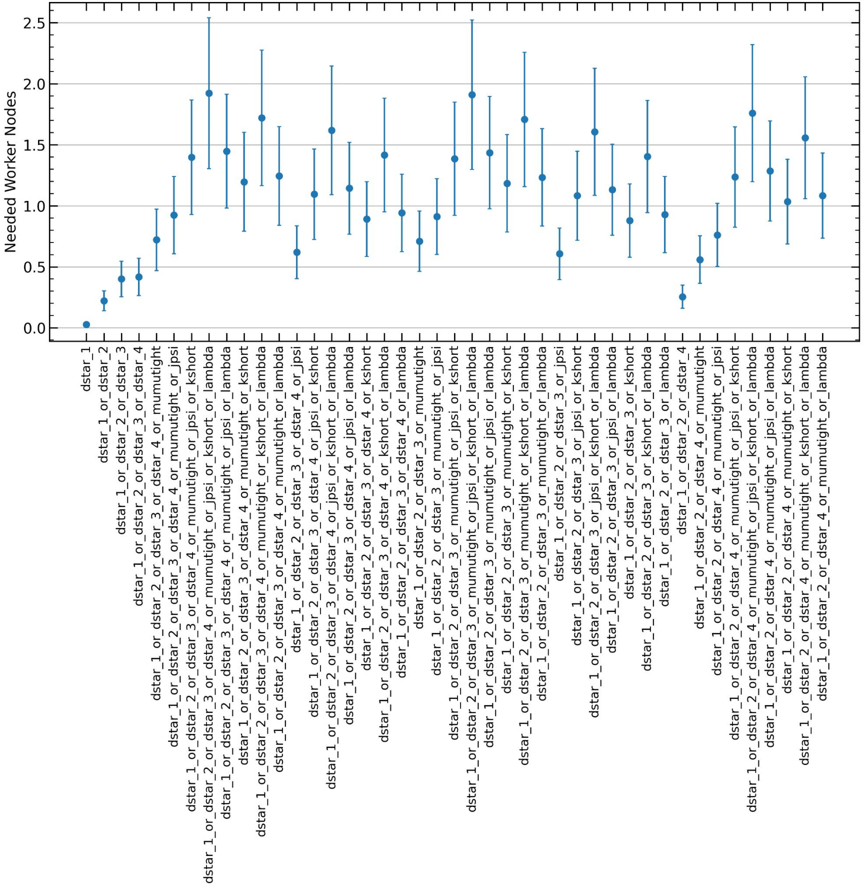


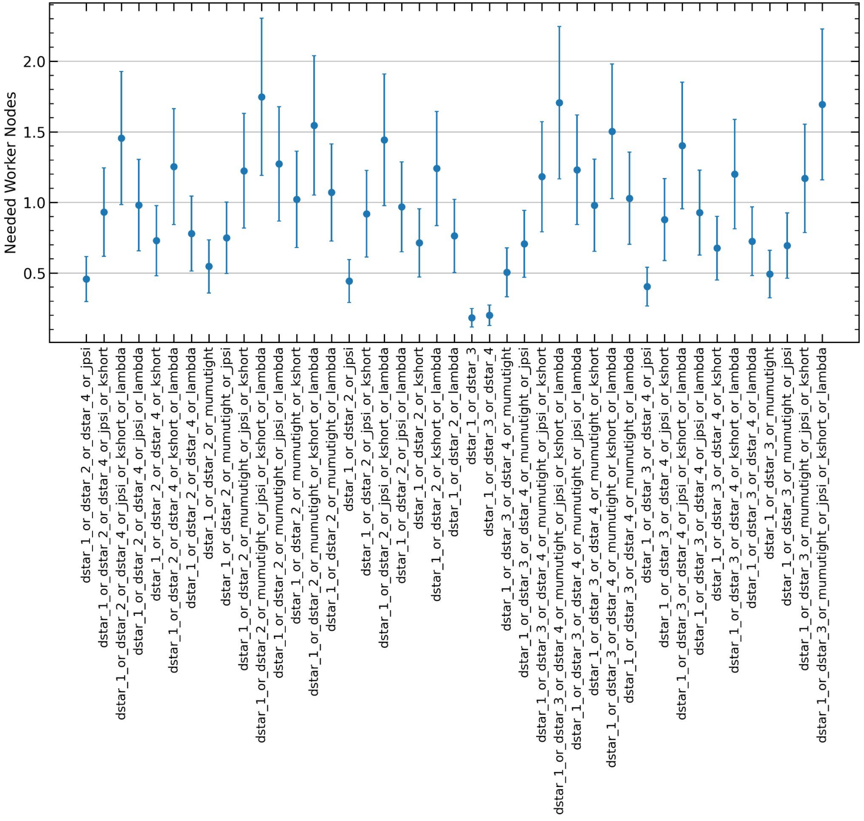



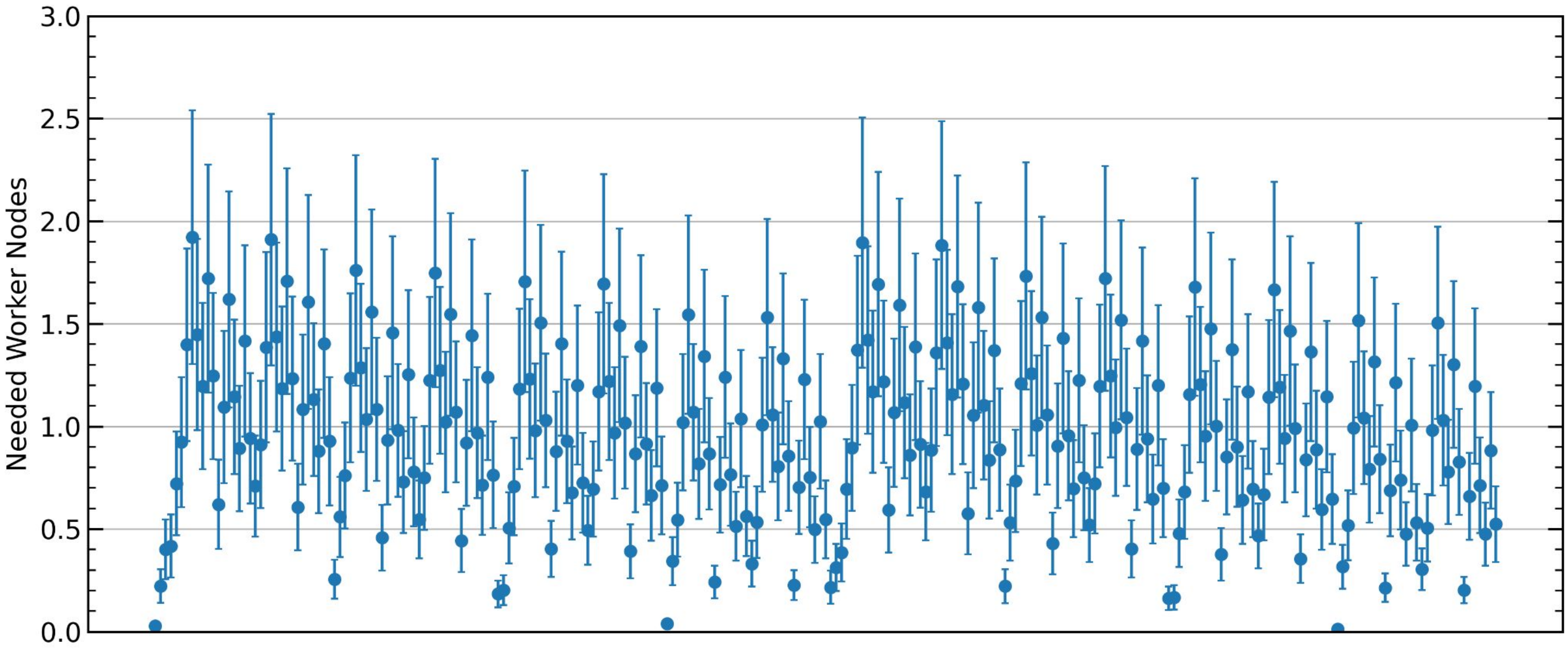



# Needed Worker Nodes for the Combinations



# Needed Worker Nodes for the Combinations




For any given combination we stay below 3 worker nodes (if standard deviation included)

# Summary and Next Steps

# Summary and Next Steps

- Investigated 8 different trigger lines and any possible combination of them
- Depending on the chosen trigger line we can estimate the worker nodes needed in order to deal with the expected rate
- We ran a few checks to verify that the script is producing the correct results

- We could test additional trigger lines
- We will start using the accept_dstar1 trigger line and allocate one worker node at a global run
- Preparation of first results to see if findings of this study are also valid in the experiment
- If successful:
  $\longrightarrow$ Add additional trigger lines
  $\longrightarrow$ Start looking at variables we could use for monitoring

Thank you for your attention! Are there any questions?