Faculty of Physics

WARSAW UNIVERSITY OF TECHNOLOGY

PID in ALICE with Machine Learning

Łukasz Graczykowski[1]
Maciej Buczyński[1], Michał Glinka[2]
[1] Faculty of Physics
[2] Faculty of Electronics and IT

Machine Learning and Quality Control in ALICE
CERN
4 December 2018

Goals of the data science group at WUT

- Use ALICE and its data as a unique environment to advance the Machine Learning field of science
- Identify areas where both ALICE (or HEP in general) and ML communities can mutually benefit
- More focus on Machine Learning research rather than simple implementations of standard ML tools for ALICE use cases
- Disclaimer:
  - I'm a physicist without a ML expertise
  - My task is to guide and coordinate the work of WUT ML computer scientists within ALICE

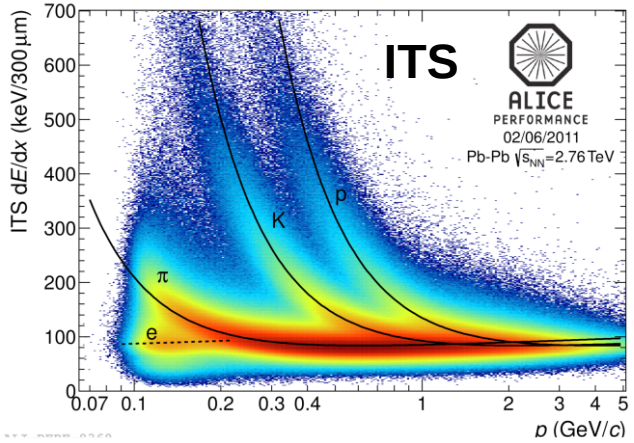
Three areas of research

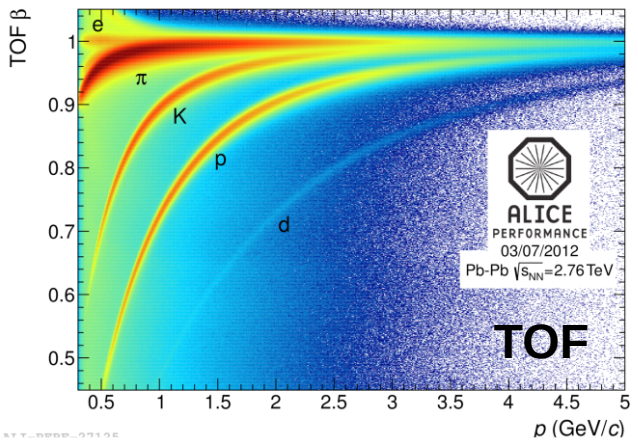
- **Data Quality Assurance – prediction of detector quality label assignment**
  - covered by Kamil Deja
- **Simulation of TPC clusters in Monte Carlo data using generative networks**
  - not covered this week
- **Development of more precise particle identification (PID)**
  - scope of this talk

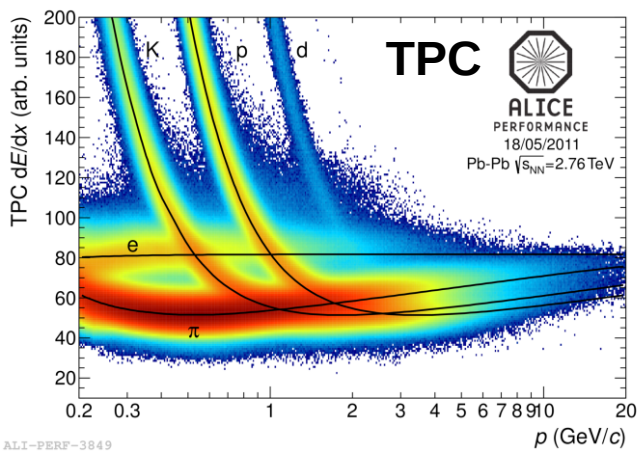
Particle identification

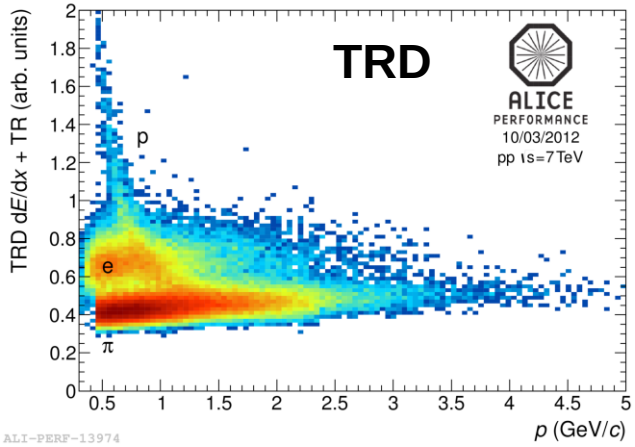
- **Particle identification (PID) is one of the most important steps in many physics analyses**
- **Crucial for Quark-Gluon Plasma measurements**
- **PID is one of the strongest advantages of ALICE:**
  - practically all known techniques used (dE/dx energy loss, time-of-flight, Cherenkov radiation for hadrons and transition radiation for electrons)
  - possibility to identify (anti-)nuclei
  - very good separation of pions, kaons, protons, electrons over a wide momentum range
  - separation of signals of charged hadrons and electrons for very low momenta (down to 0.1 GeV/c)

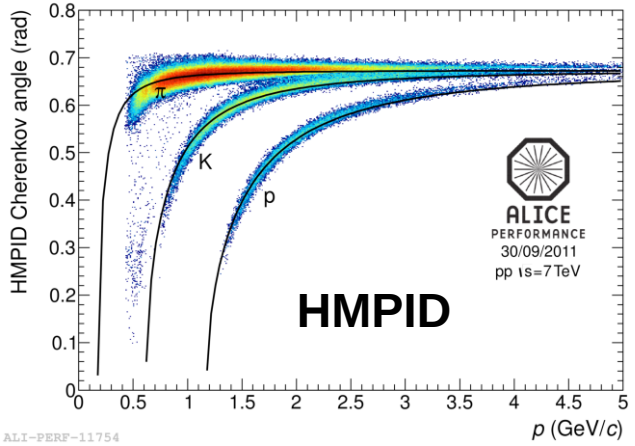
Particle identification

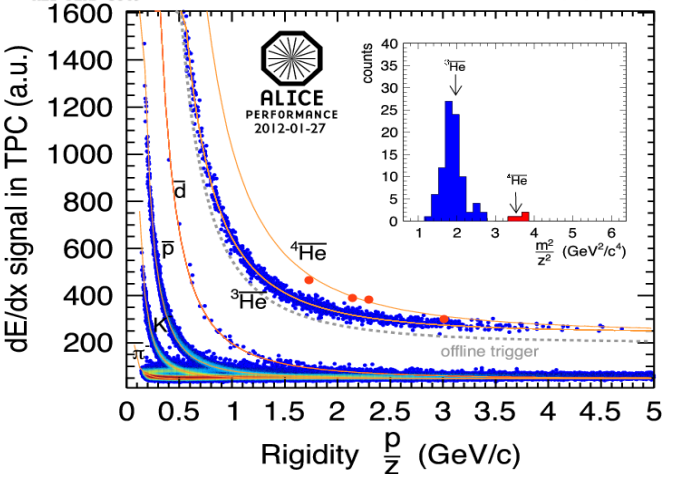


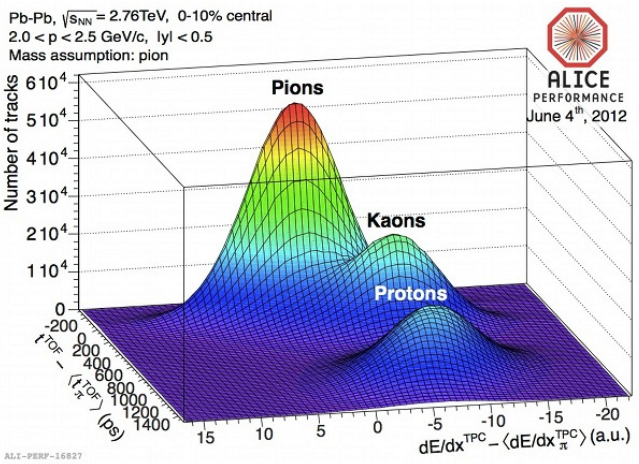






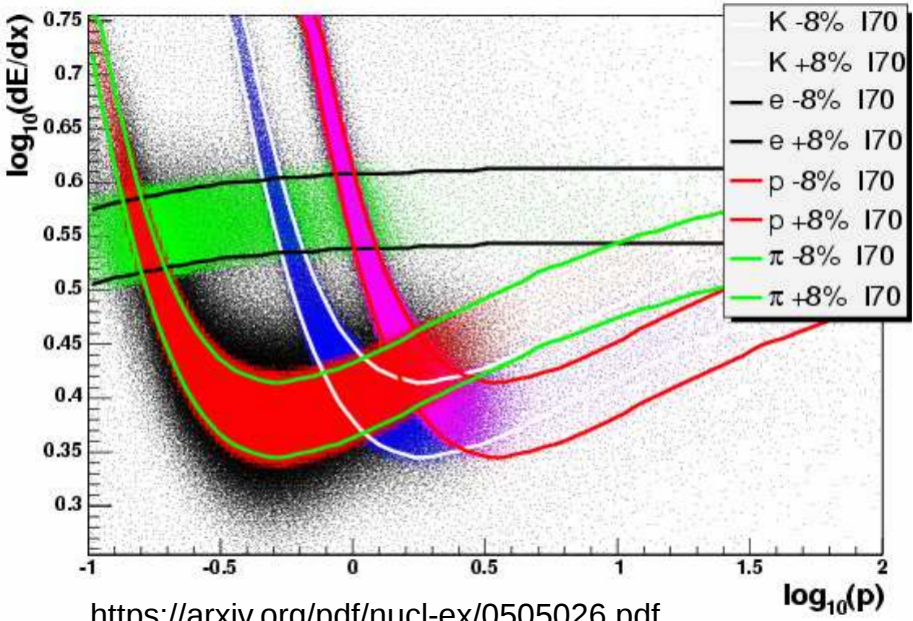






Traditional vs ML PID

- **Traditional PID:**
  - a typical analyzer selects particles "manually" by cutting on certain quantities, like the number of standard deviations of a signal from the expected value
  - most limitations come in the regions where signals from different particle species cross
  - "cut" optimization is a time-consuming task



https://arxiv.org/pdf/nucl-ex/0505026.pdf

- **Machine learning PID:**
  - perfect task for machine learning
  - can learn non-trivial relations between different track parameters and PID
  - no "trial and error" approach

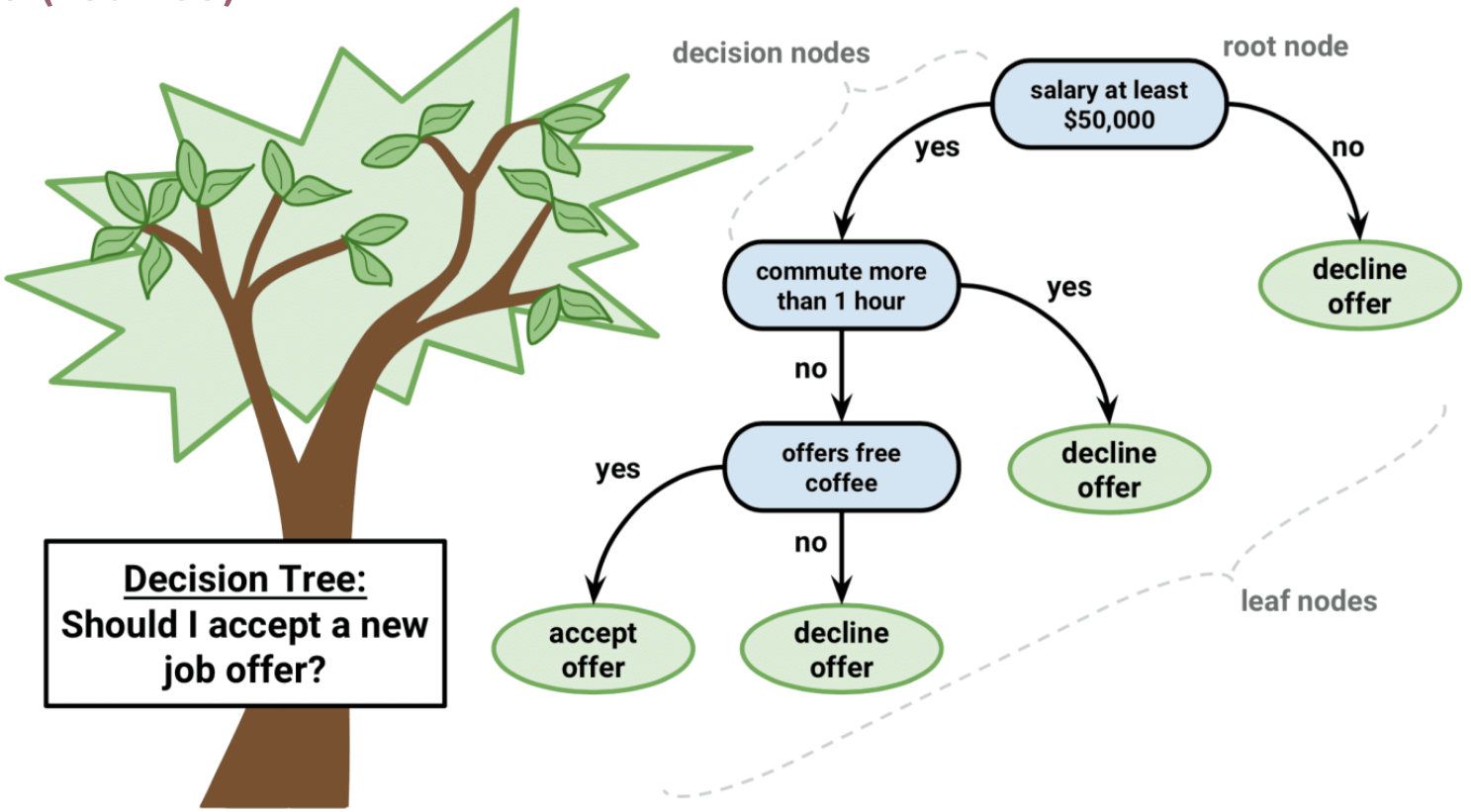
Proposed solution for PID

- Build a ML classifier that can outperform traditional PID
- Train and validate the classifier on Monte Carlo and real data
- Create a simple interface for users in AliRoot
- In the first step – use AOD files and AOD tracks for classification as the users do while manually setting their cuts

- Limitations:
  - Quality of the classifier will depend on the MC sample (discrepancies between data and MC)
  - No easy way to calculate systematic uncertainties from the procedure
  - The classifier is a “black box” - no easy way to tell what’s going on inside

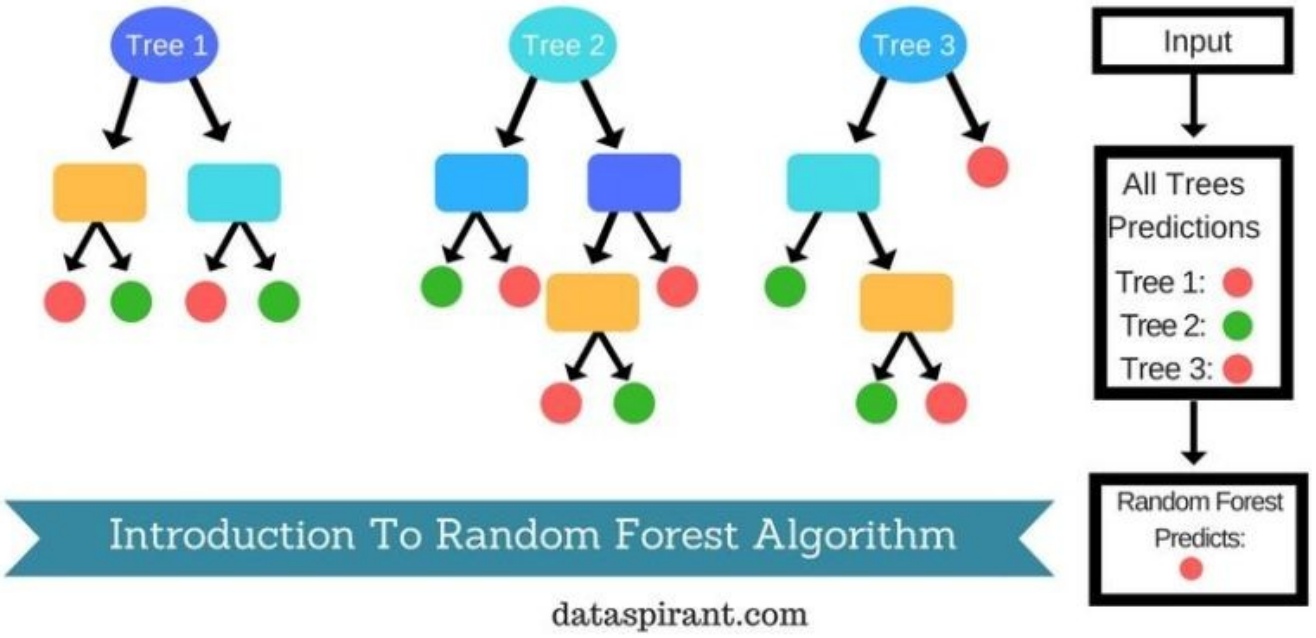
Decision tree

- A decision tree is a tree where each node represents a feature (attribute), each link (branch) represents a decision (rule) and each leaf represents an outcome (categorical or continues value)
- Decision tree learning uses a decision tree to go from observations about an item (attributes) to conclusions about the item's target value (leaves)



Random Forest

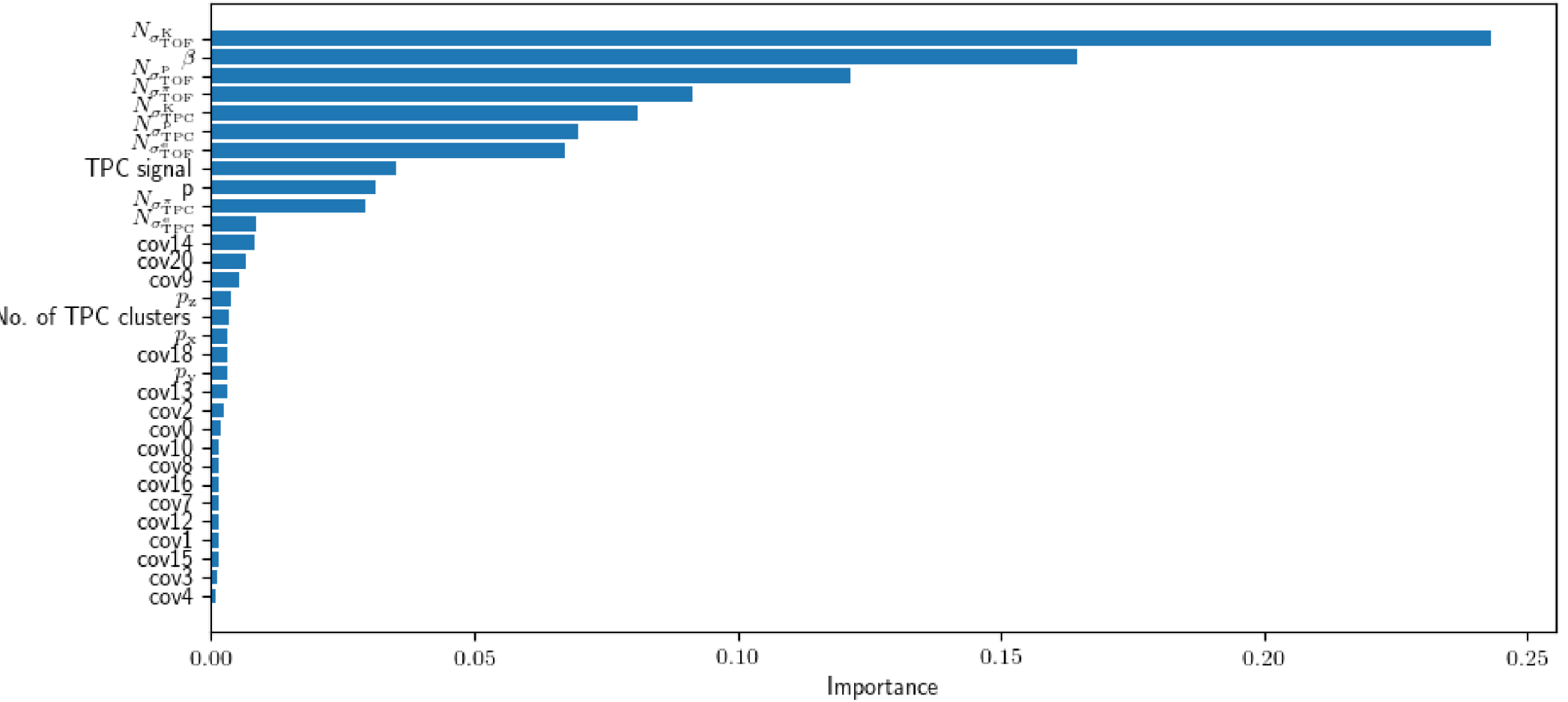
- A collection of decision trees ("forest") where each tree votes for a final decision
- Each tree is trained on a subset of randomly selected training data
- The final result is (in most cases) the one with majority of votes
- ... in addition, adaptive boosting was used



Let's see some results

PID parameter importance

- Focus on kaons
- Input parameters were reduced to the most significant ones
- Importance of AOD track parameters – their contribution to the final result (kaon selection)



Results – kaon selection

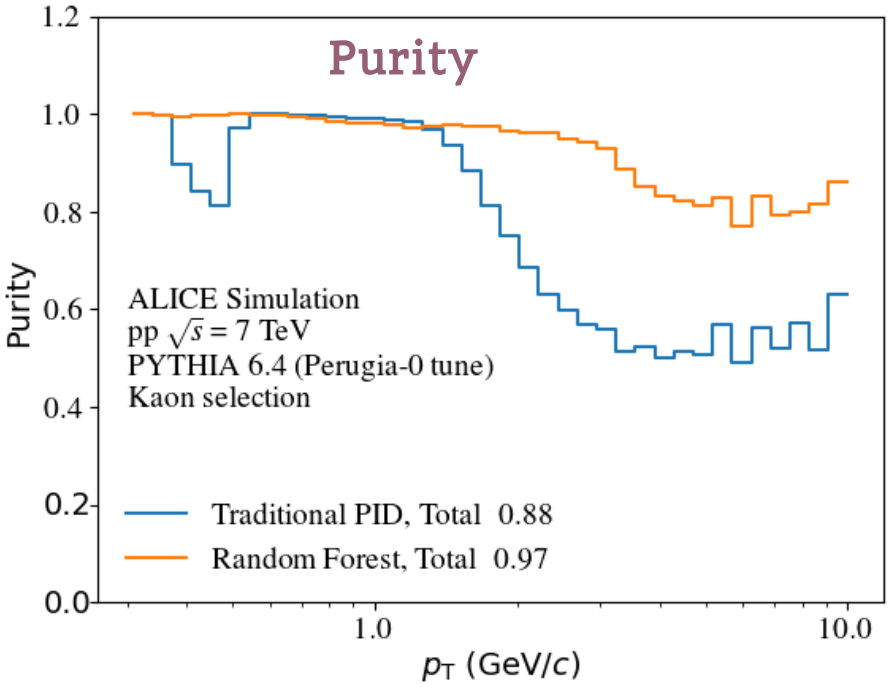
- **Test data sample:**
  - pp @ 7 TeV, Pythia 6 Perugia-0
- **Traditional PID:**
  - $n^2_{\sigma,TPC} < 2$, for $p_T \leq 0.5$ GeV/c
  - $\sqrt{n^2_{\sigma,TPC} + n^2_{\sigma,TOF}} < 2$, for $p_T > 0.5$ GeV/c
  - veto on other particles

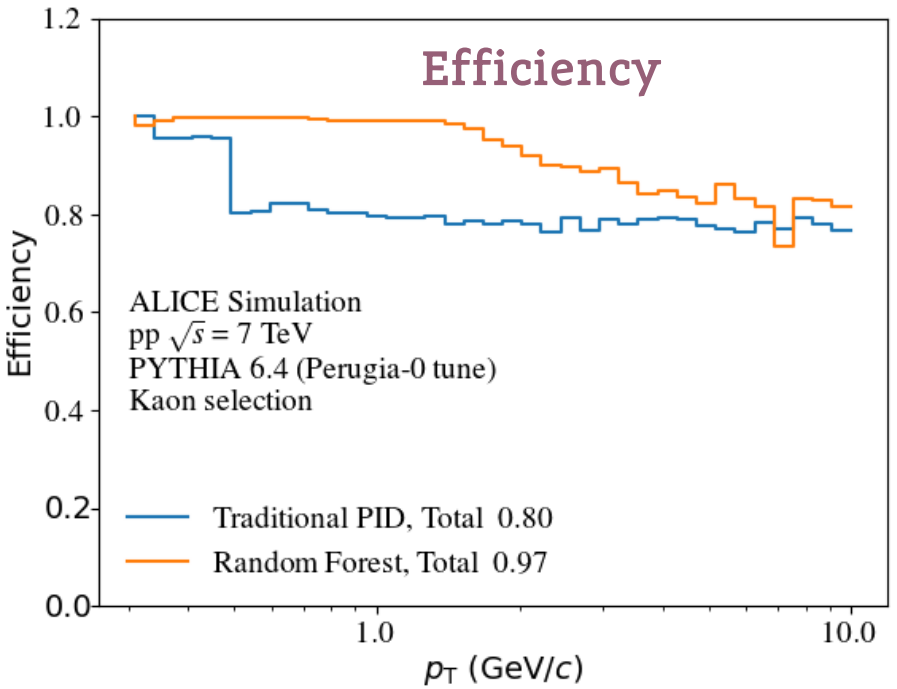
- **Efficiency**

$$\varepsilon_{Total} = \frac{N^{survived-all-cuts-including-PID}_{primaries}}{N^{generated}_{primaries}}$$

- **Purity = 1 – C**

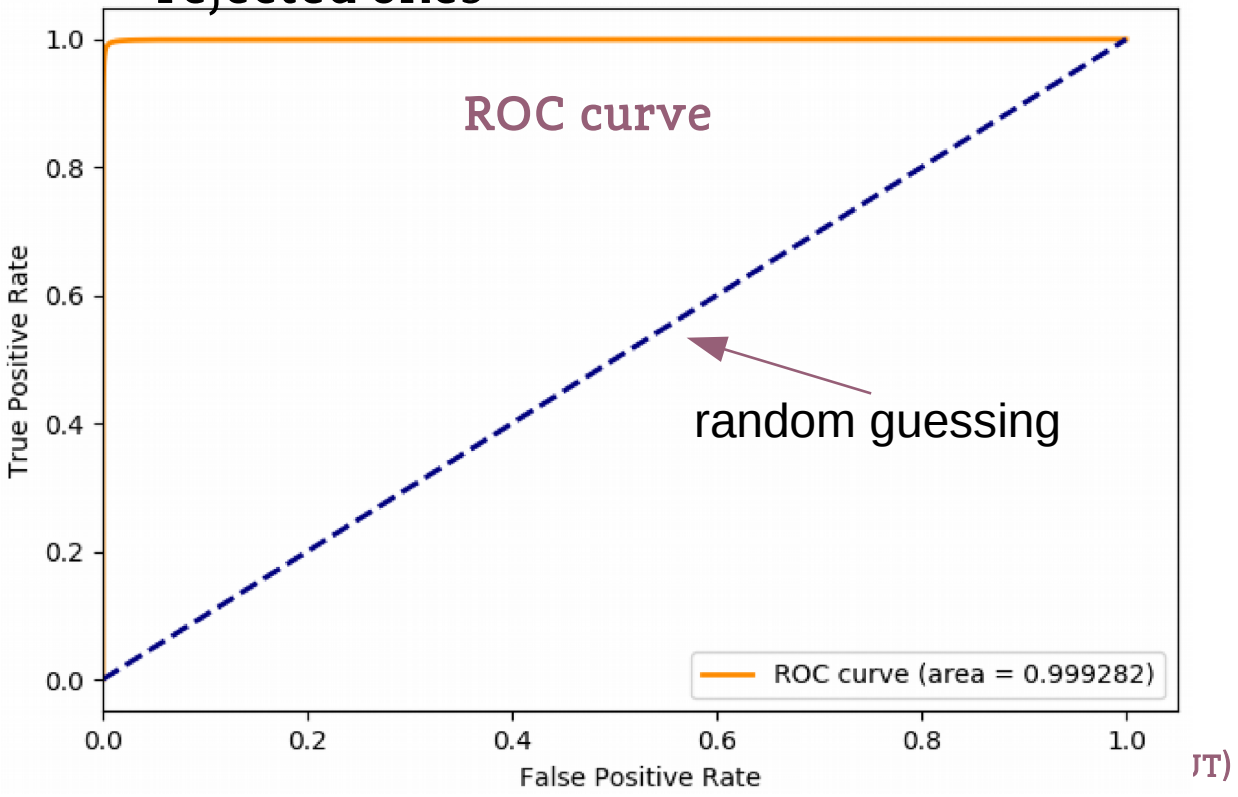
  C – contamination
  (fraction of correctly identified)

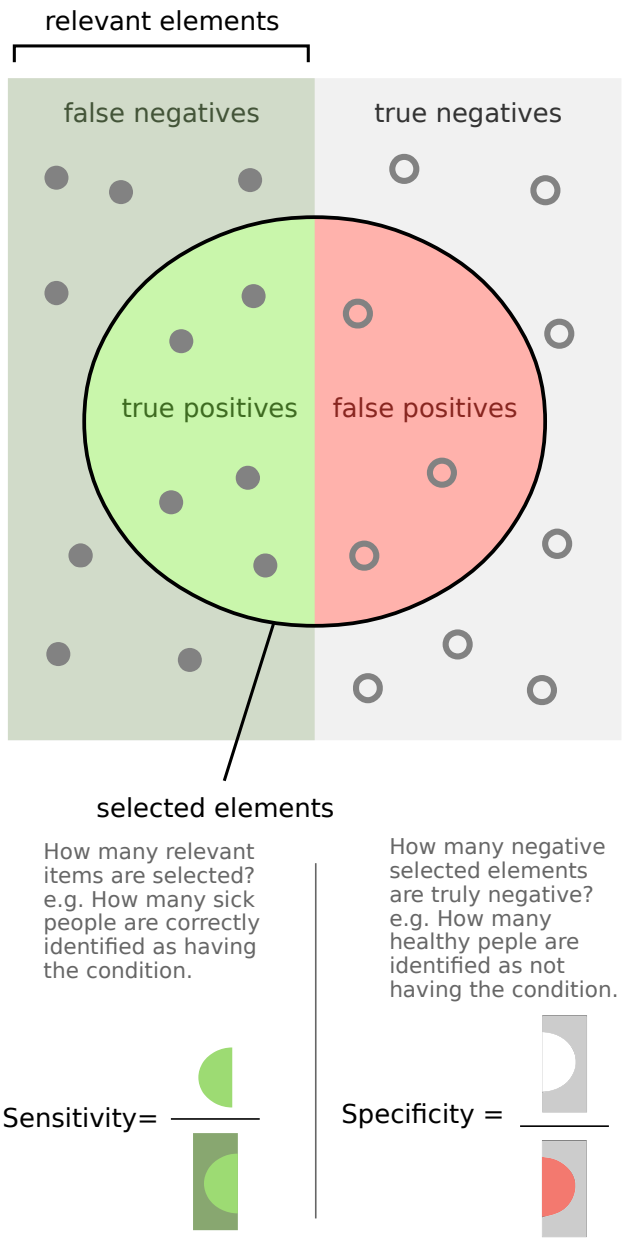




ROC curve (kaons)

- **Receiver Operating Curve (ROC) – it's a plot of true positive rate (TPR) vs false positive rate (FPR)**
- **Statistical measures of a classifier:**
  - "Sensitivity" (=TPR), proportion of correctly identified → in our case it's simply **purity**
  - "Specificity" (=1-FPR), proportion of correctly rejected ones

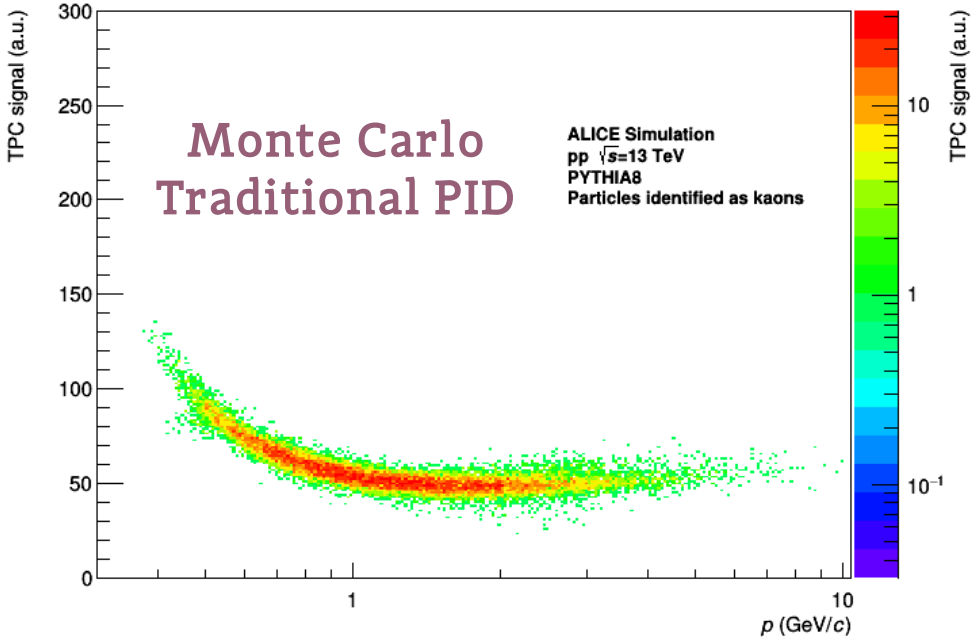


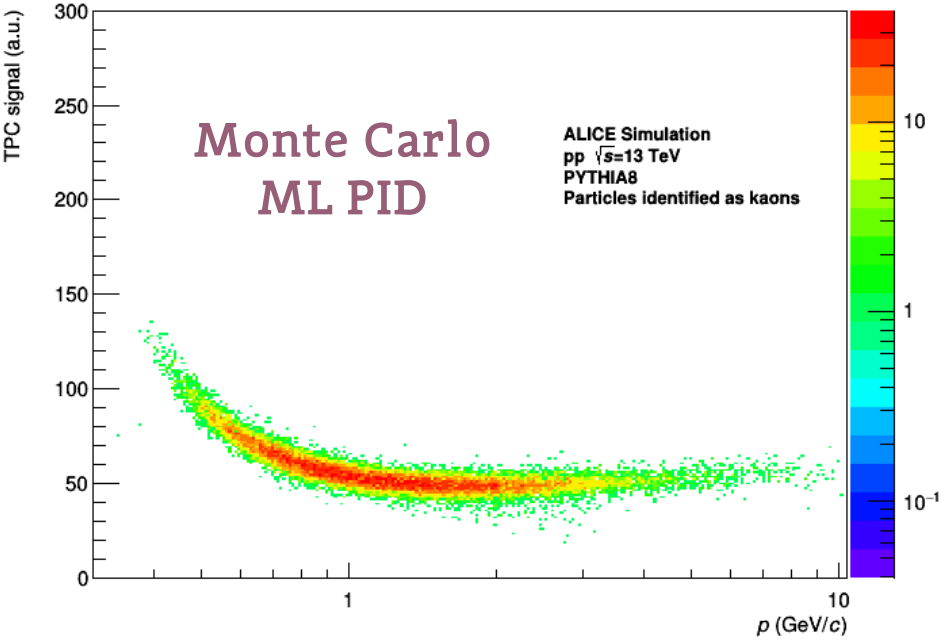


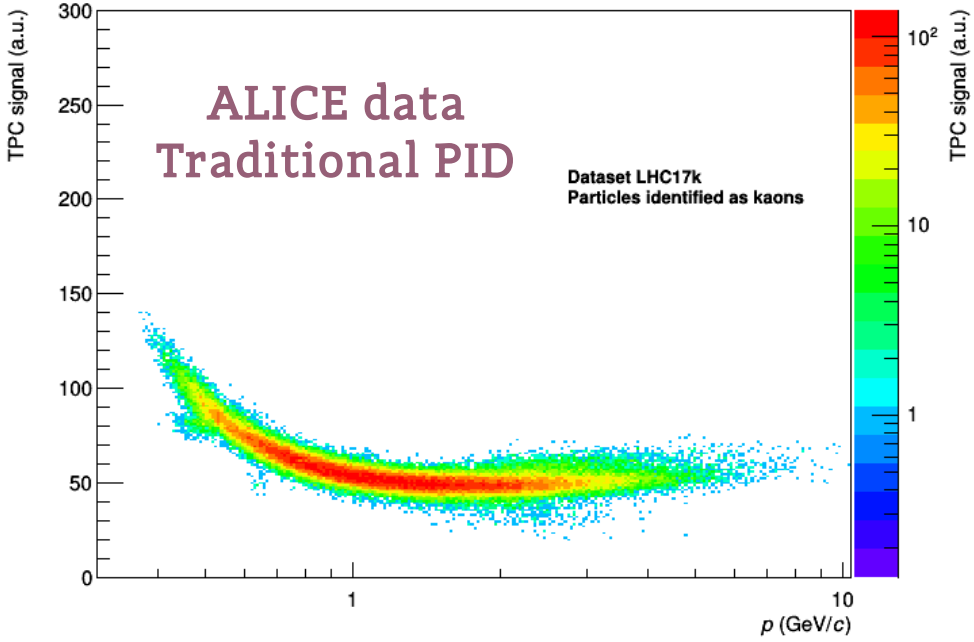
Monte Carlo and data

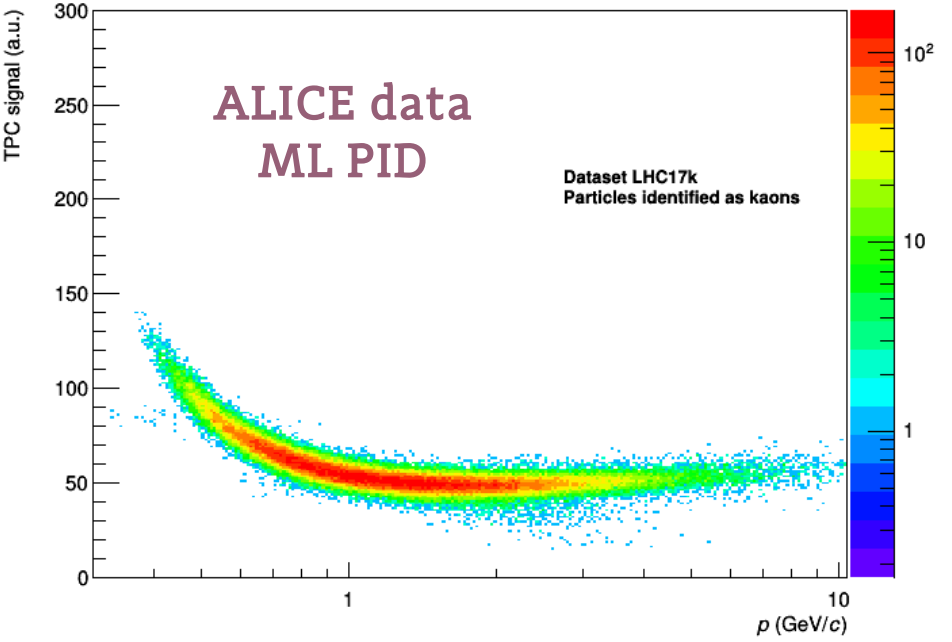
- So far we've seen the results of the classification on MC data only
- How does it actually correspond to experimental data?
- Can we use the classifier in a real analysis?
- Let's see the TPC dE/dx and TOF beta plots for experimental data and Monte Carlo

TPC accepted kaons

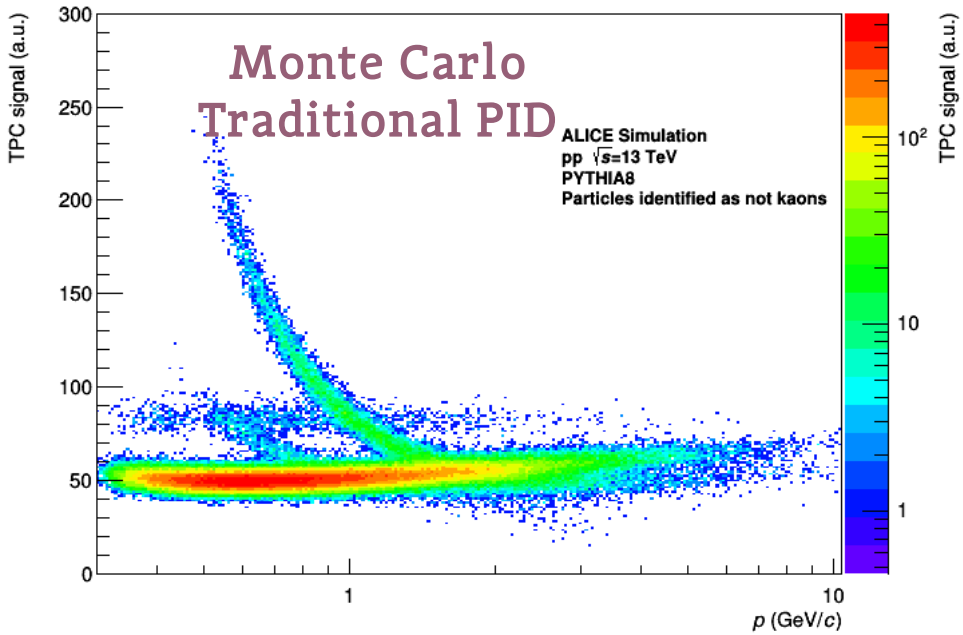


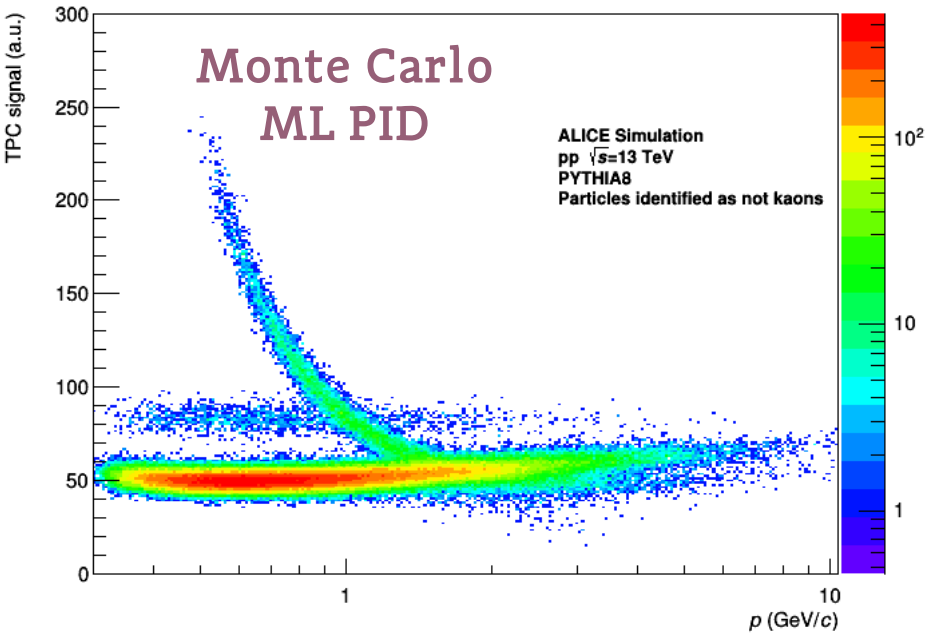


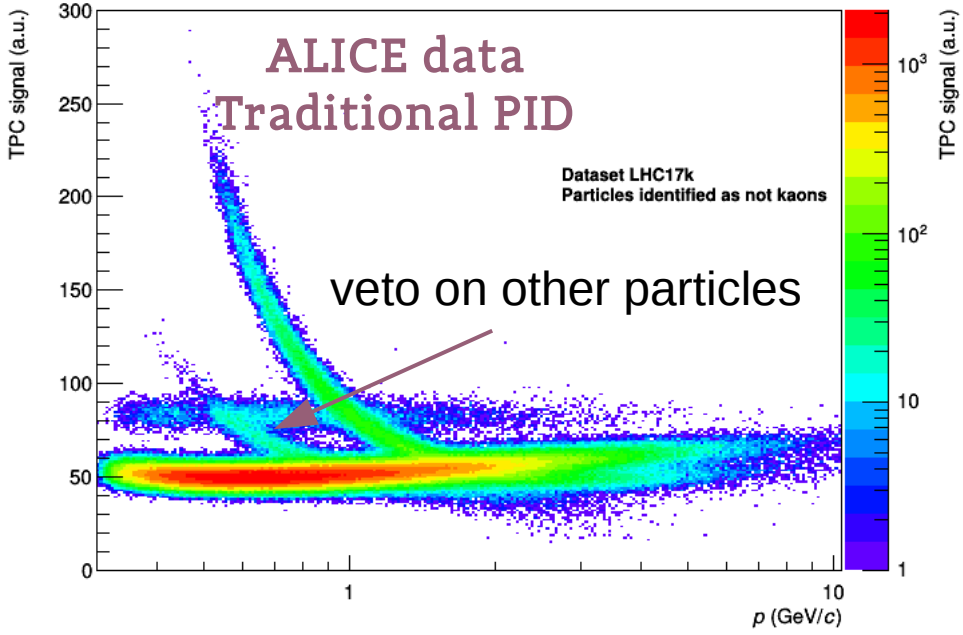


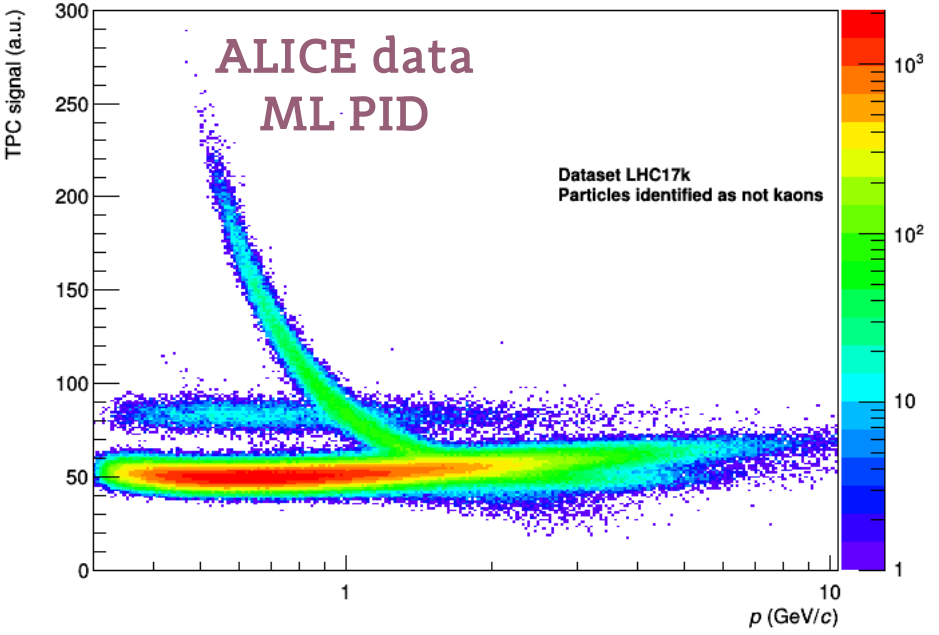


TPC rejected (not kaons)

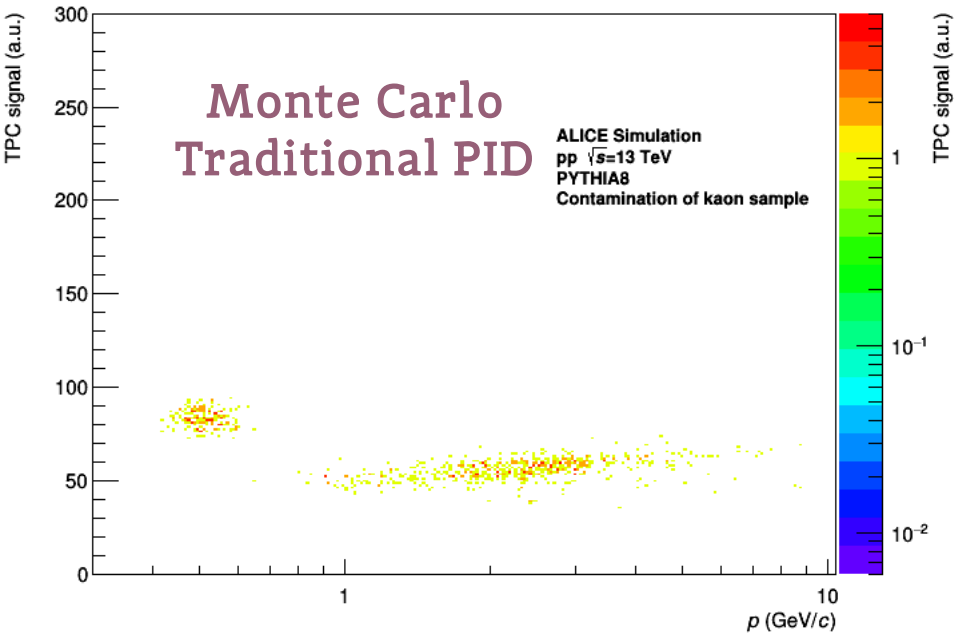


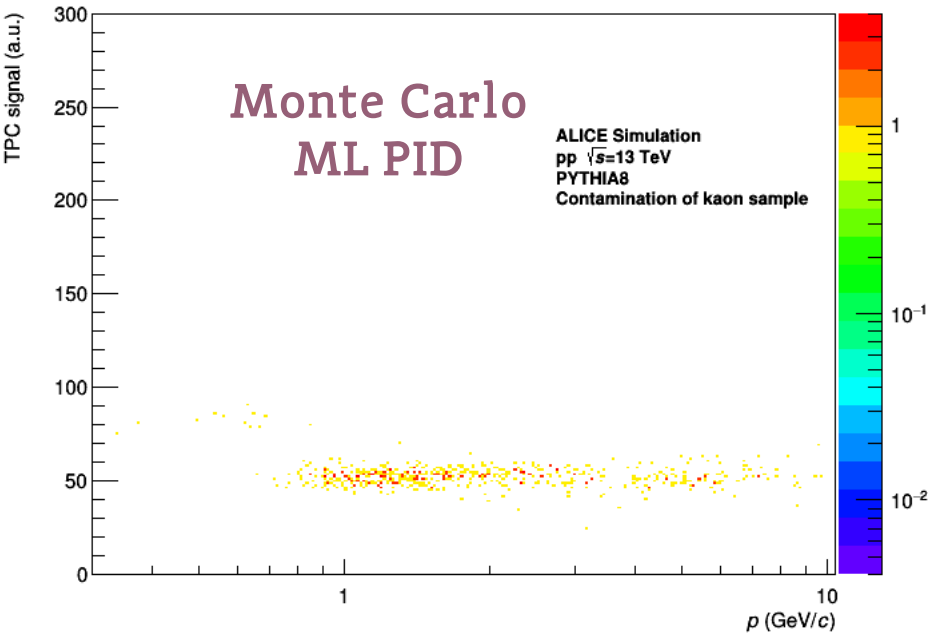




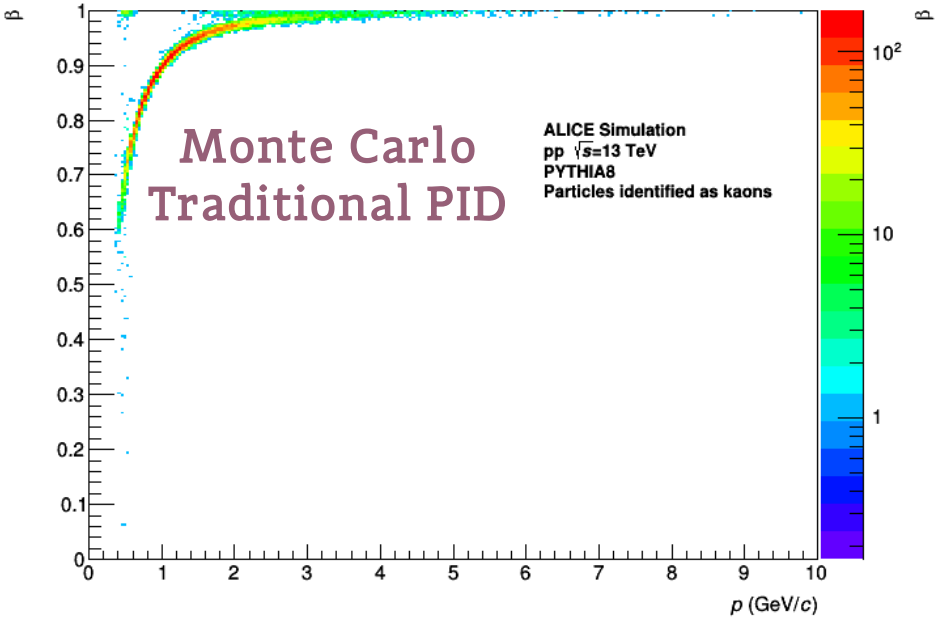


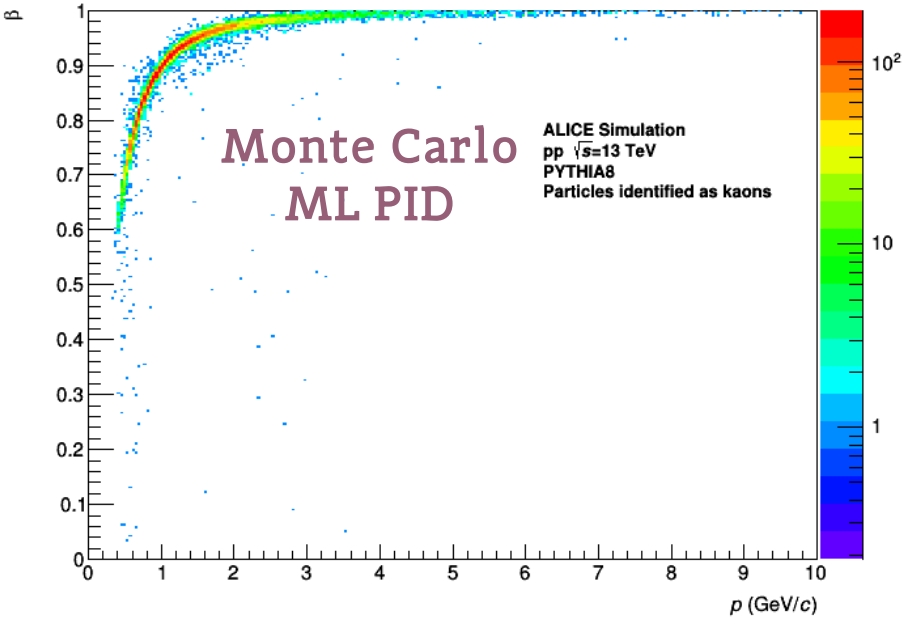
TPC contamination (kaons)

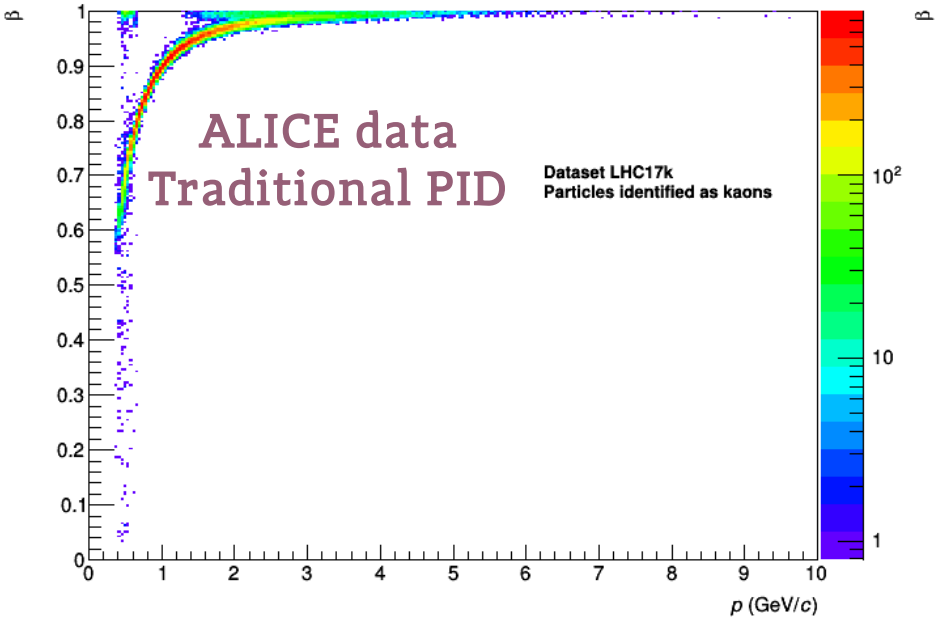


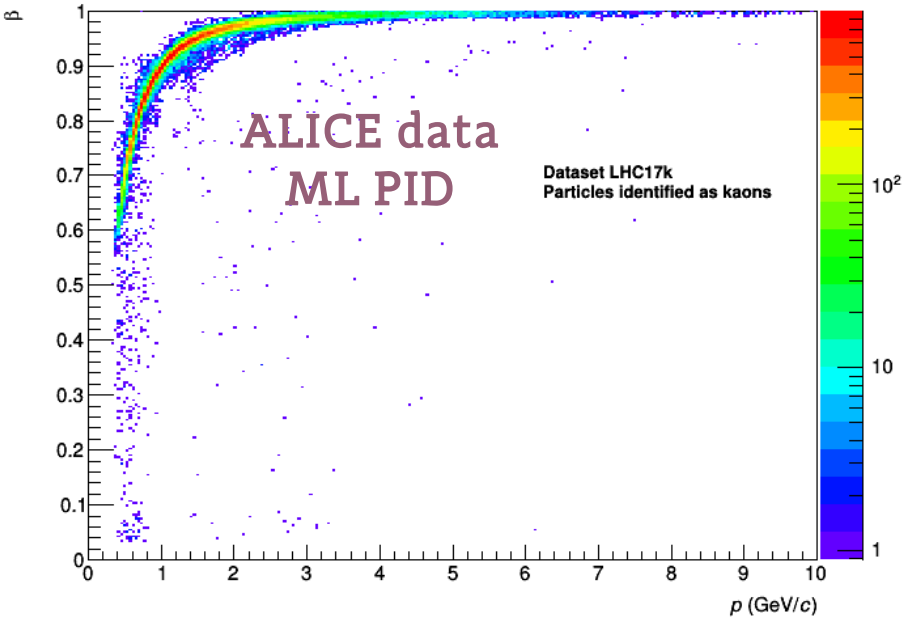


TOF accepted kaons

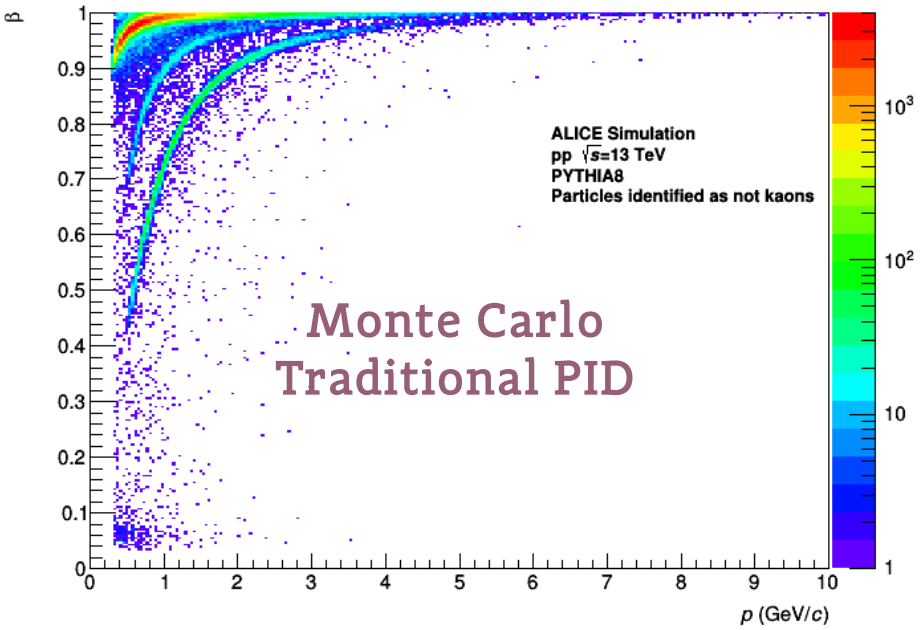




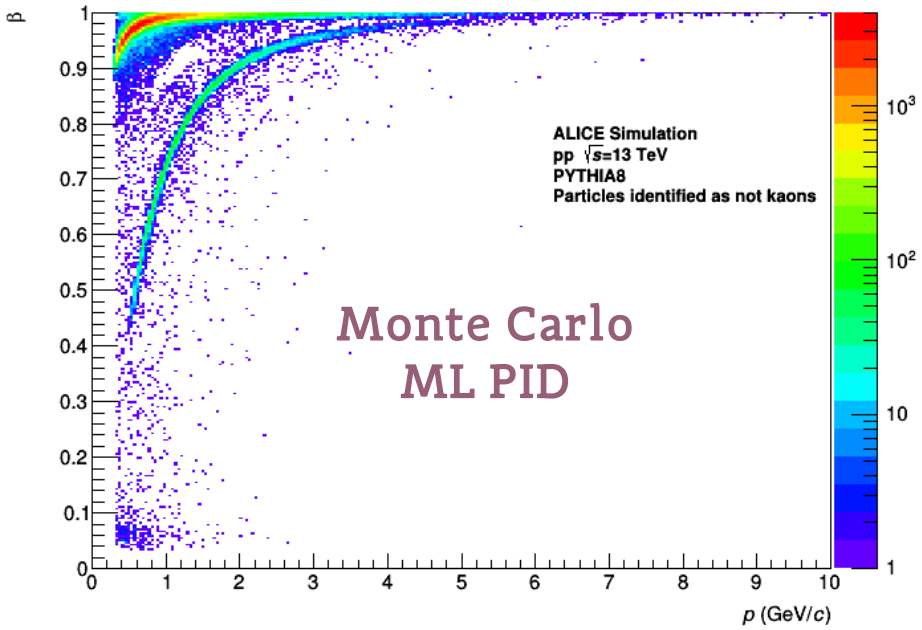




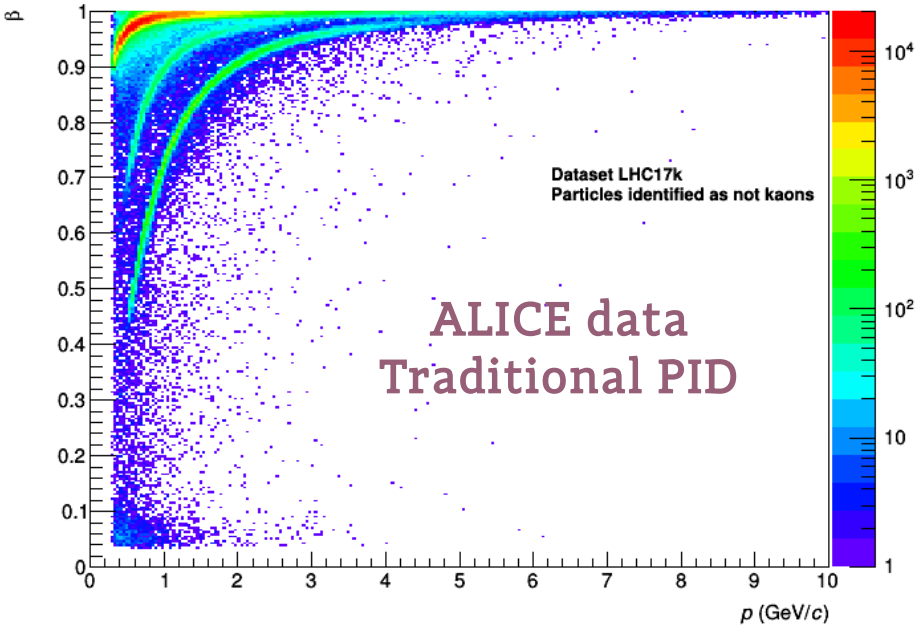
TOF rejected (not kaons)



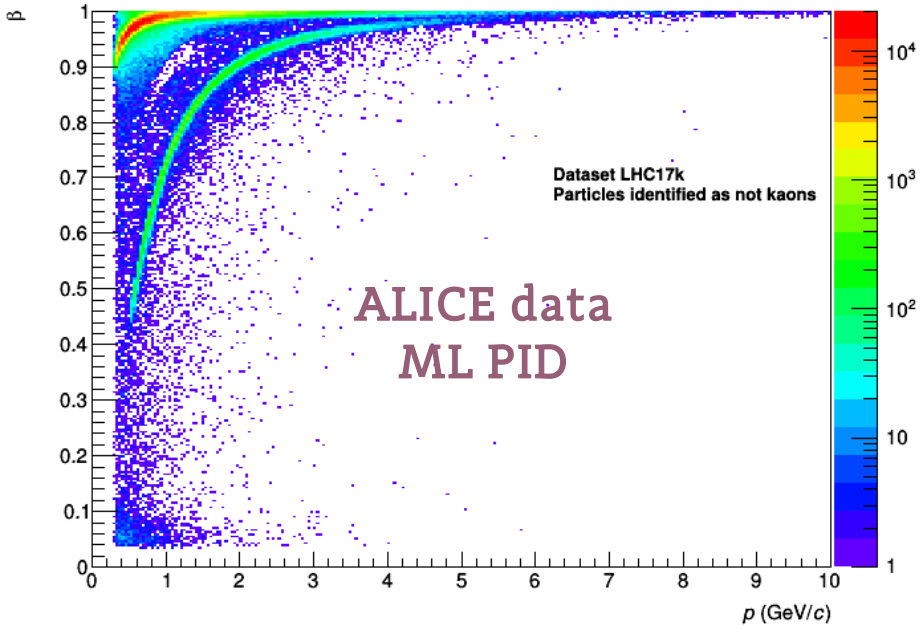
Monte Carlo
Traditional PID



Monte Carlo
ML PID

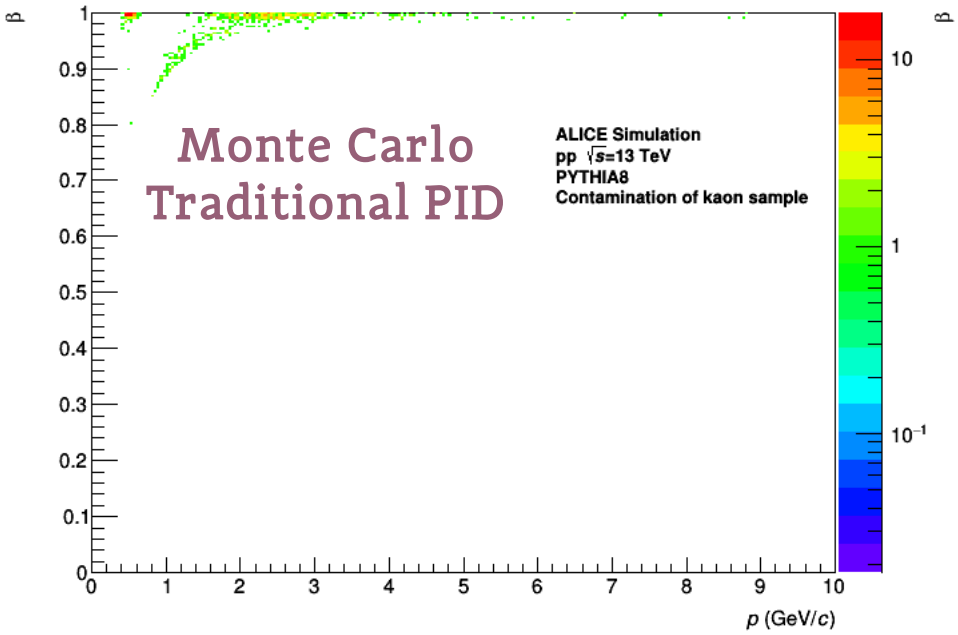


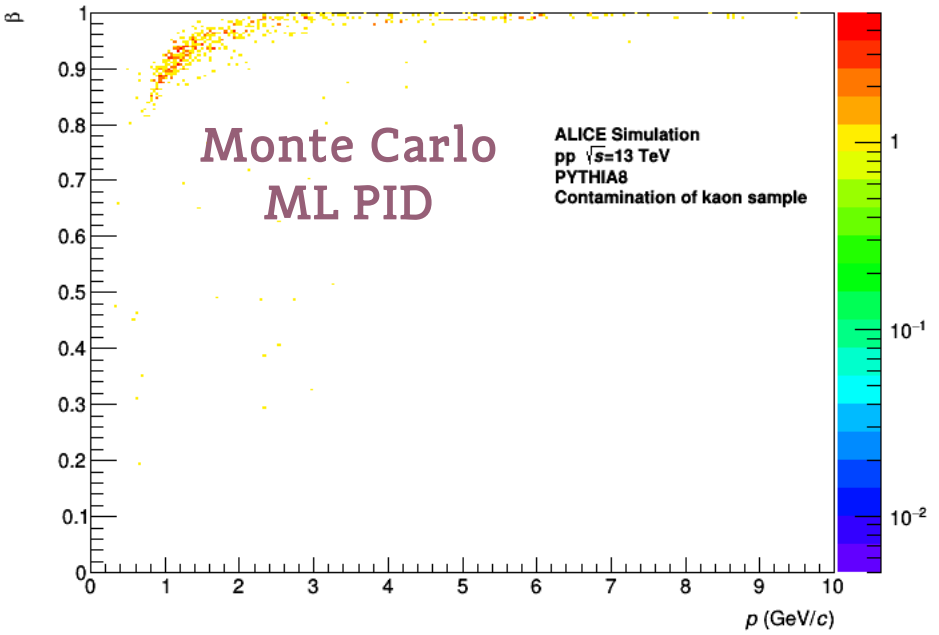
ALICE data
Traditional PID



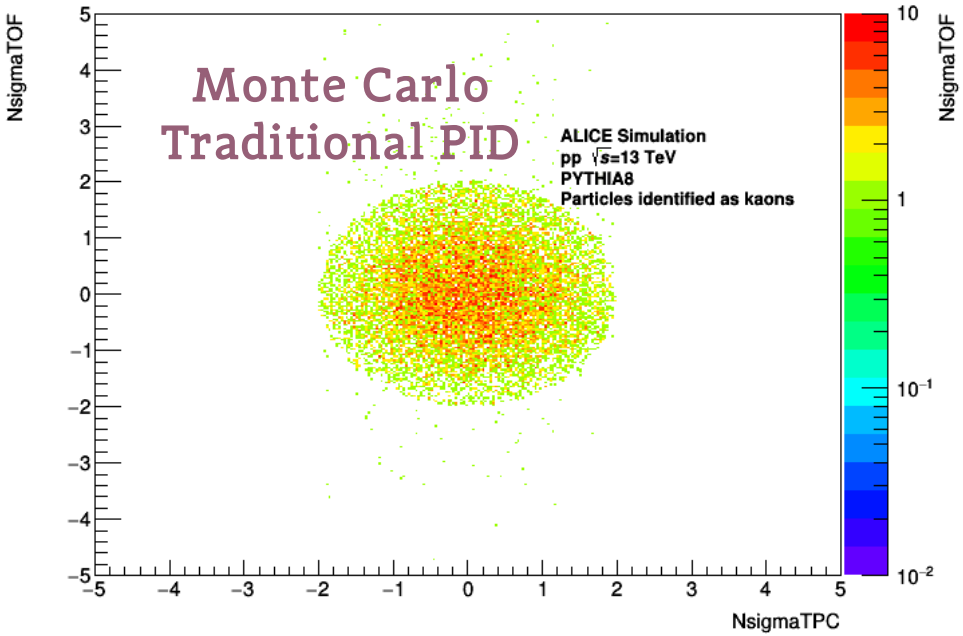
ALICE data
ML PID

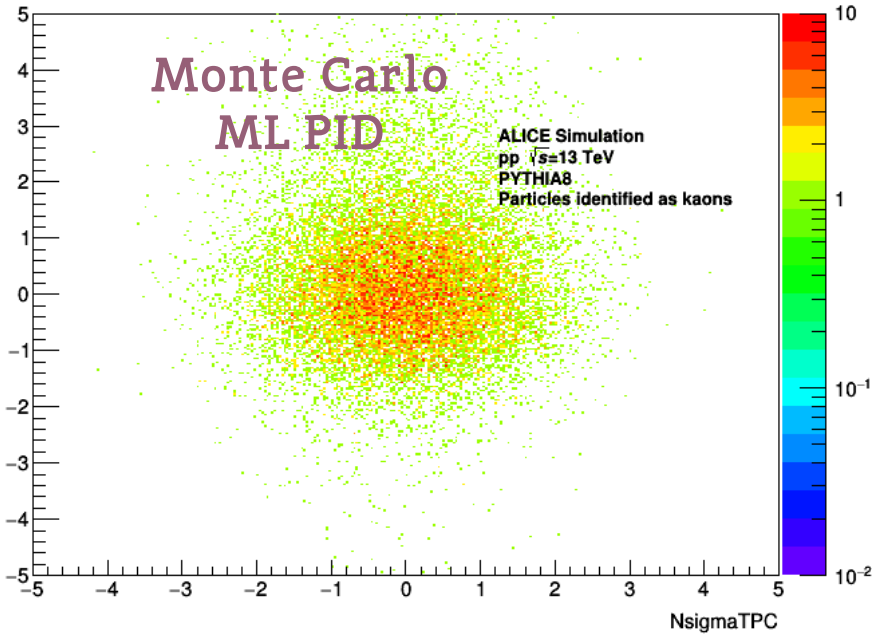
TOF contamination (kaons)

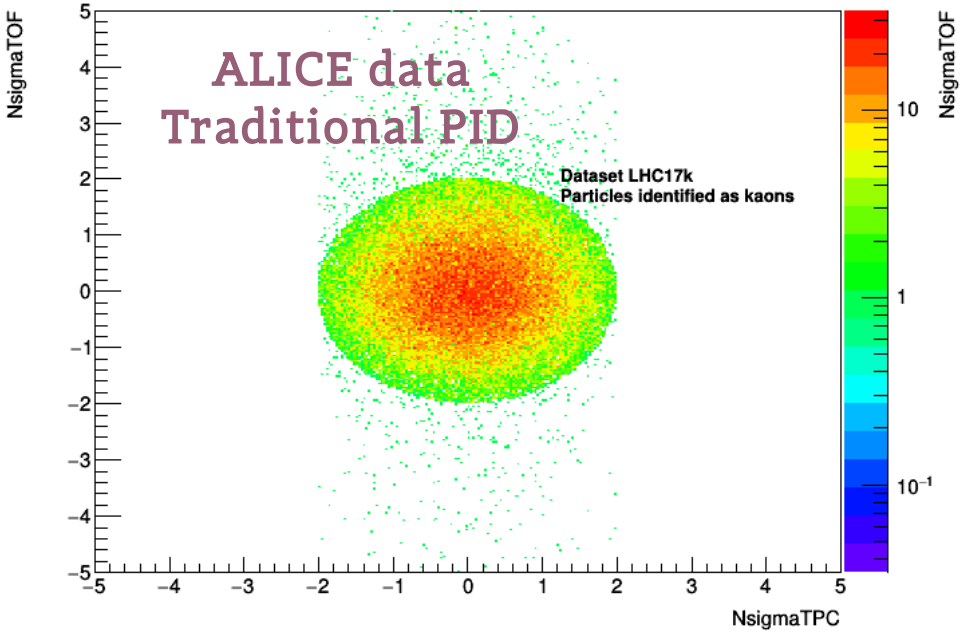


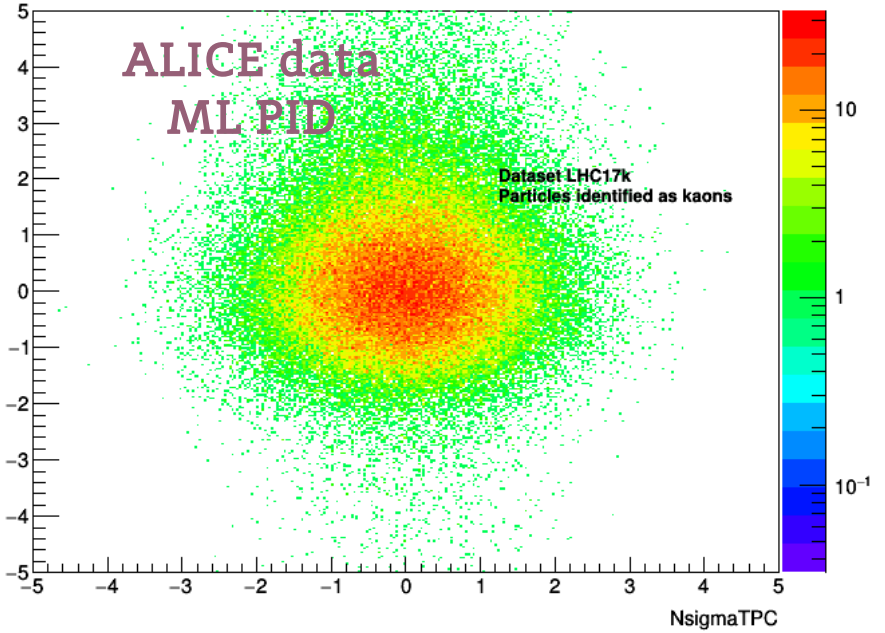


TPC vs TOF accepted kaons

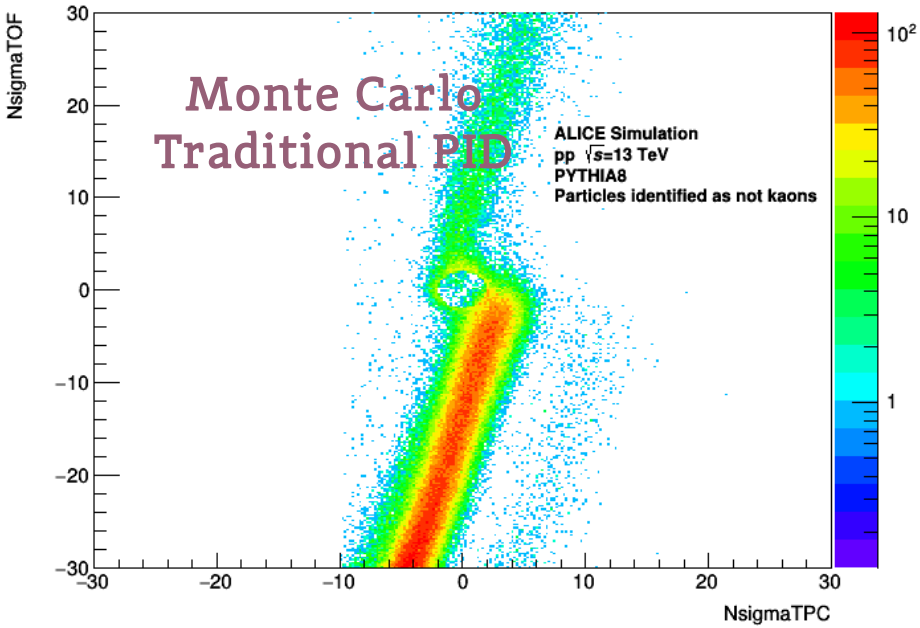


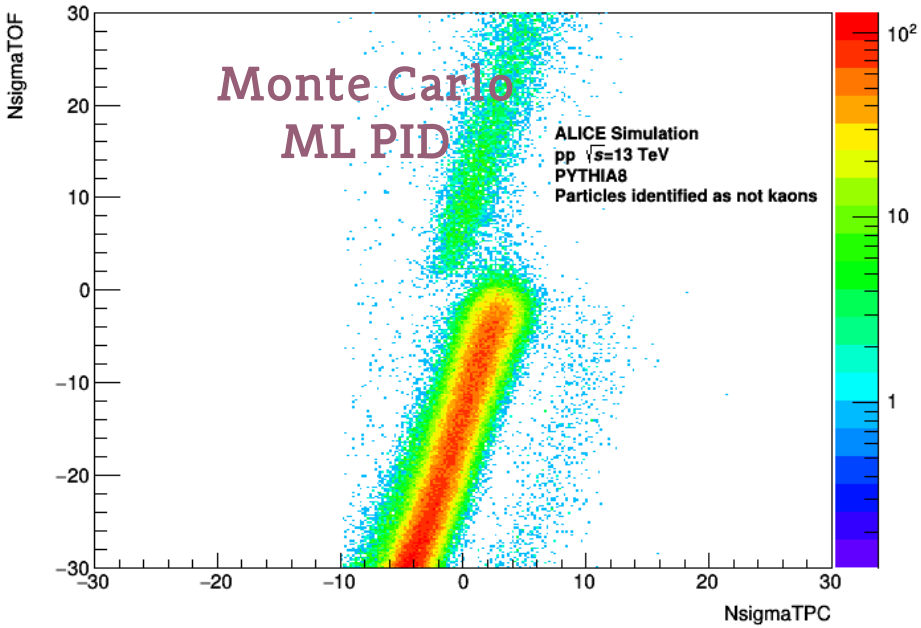


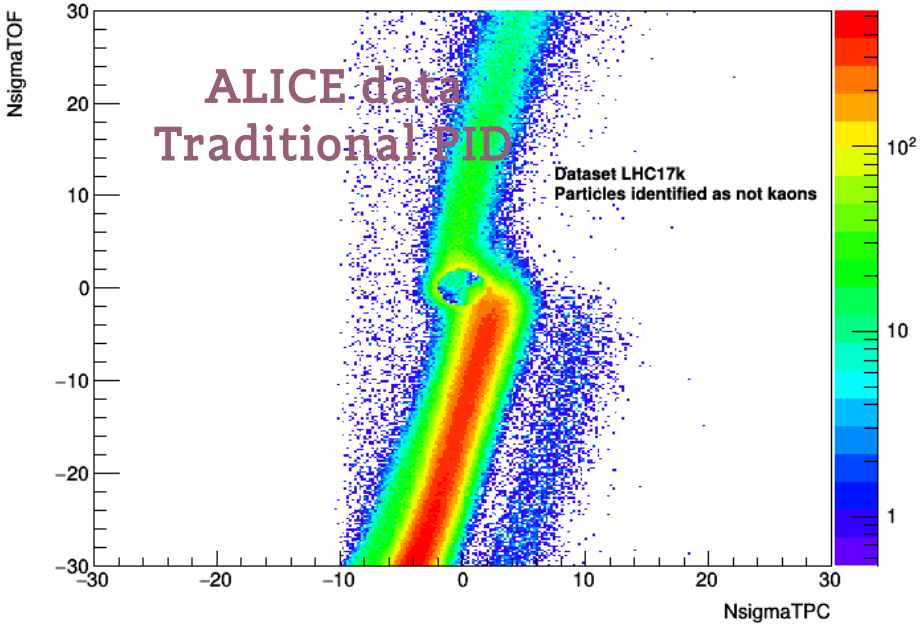


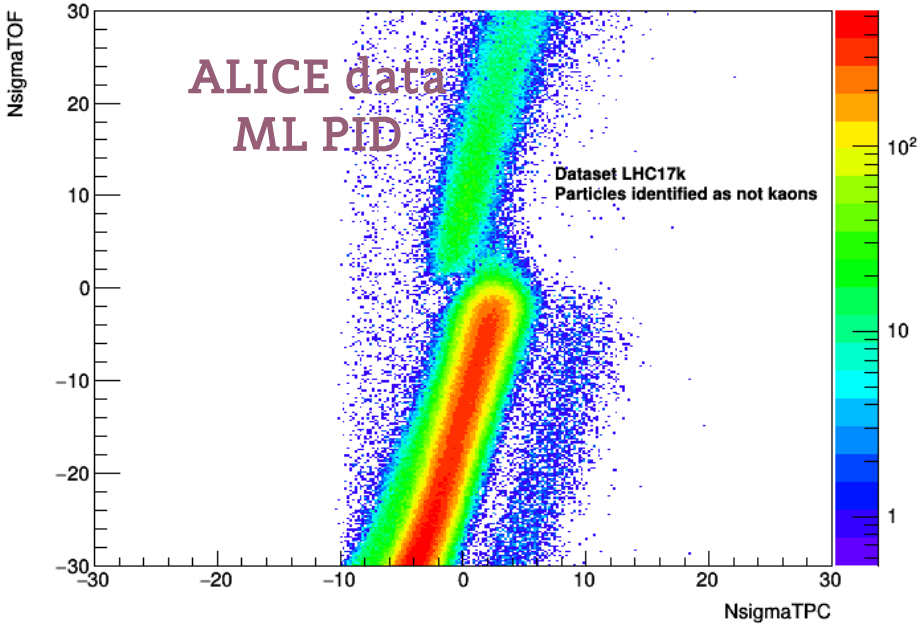


TPC vs TOF accepted kaons









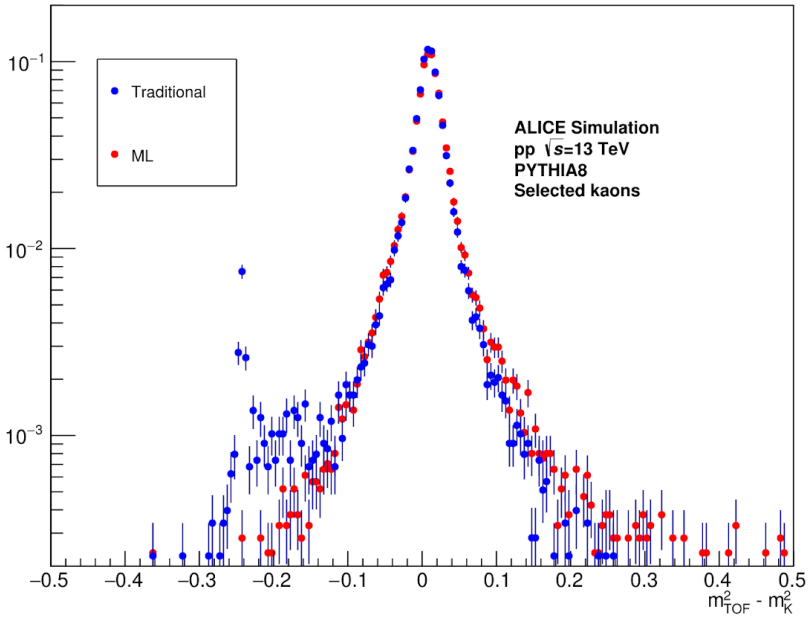
TOF time

- From our point of view TOF has a fantastic feature of a possibility to calculate mass of the recorded particle and compare it to the one from PDG

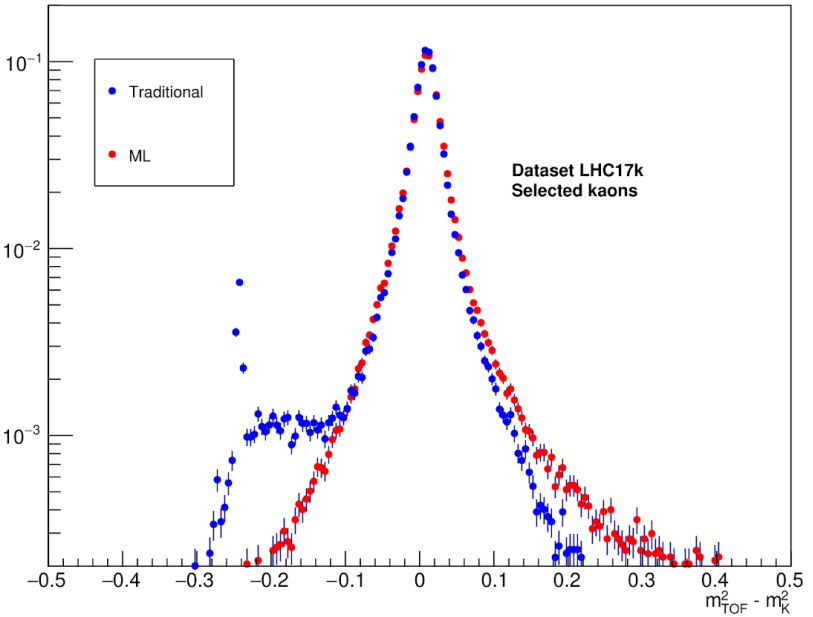
$$m^2_{TOF} = p^2\left(\frac{1}{\beta} - 1\right)$$

- Thanks to that we can test contamination independently of MC simulations

Monte Carlo



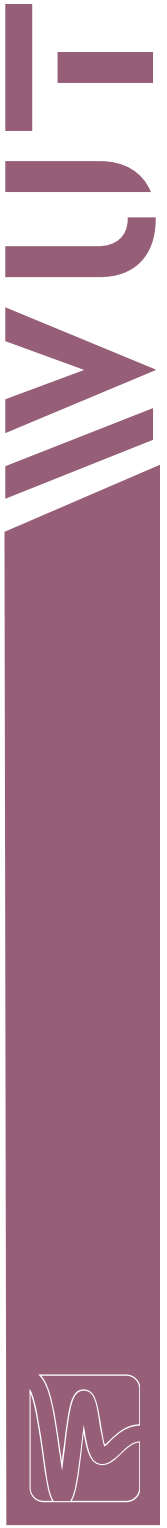
ALICE data



Implementation in AliRoot

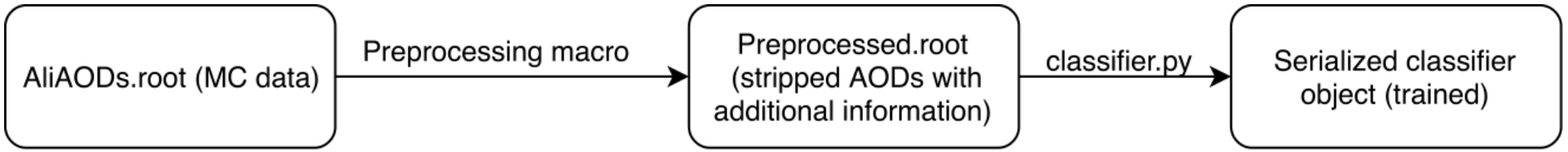
Implementation

- **Training part**
  - Not covered in this talk
  - Service work by Aaron Capon (SMI Vienna)
  - Proposed solution: to be done in a centralized way
- **Classification part**
  - Classifier prepared by Michał Glinka, an IT student
  - Work by Maciej Buczyński, a physics student
  - Different attempts tested during Maciej's three summer months at CERN this year
  - Demo/beta version already works



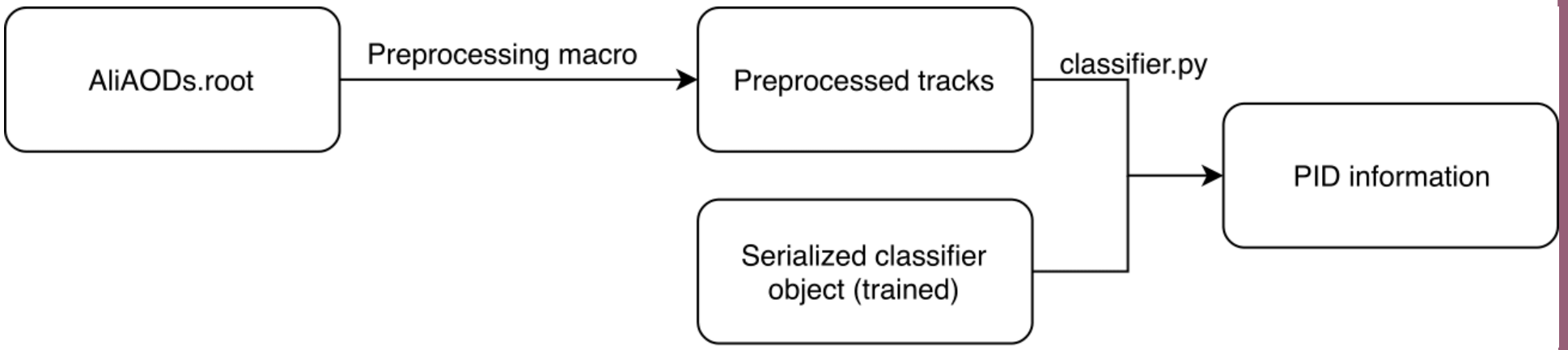
Classifier

- We get the trained classifier in a Python format (serialized classifier object) via the Python procedure (classifier.py)



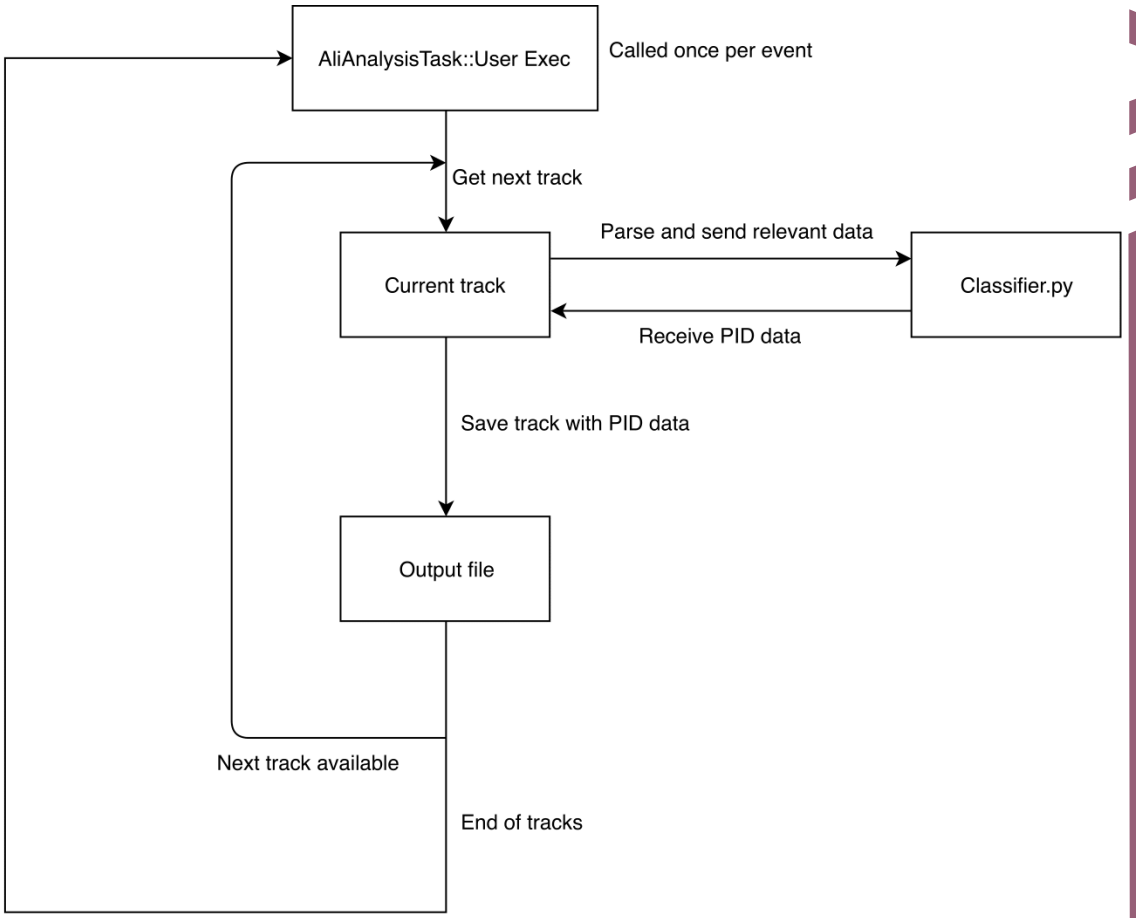
Classification – general idea

- Get the tracks (from AOD files) and the trained model in Python
- Propagate AOD tracks through the model
- The ML PID information consists of a PDG code of the predicted particle and probabilities for other PDG codes
- Present the information to the user
  - via new `AliMLPIDresponse` task and `AliMLPIDUtil` object



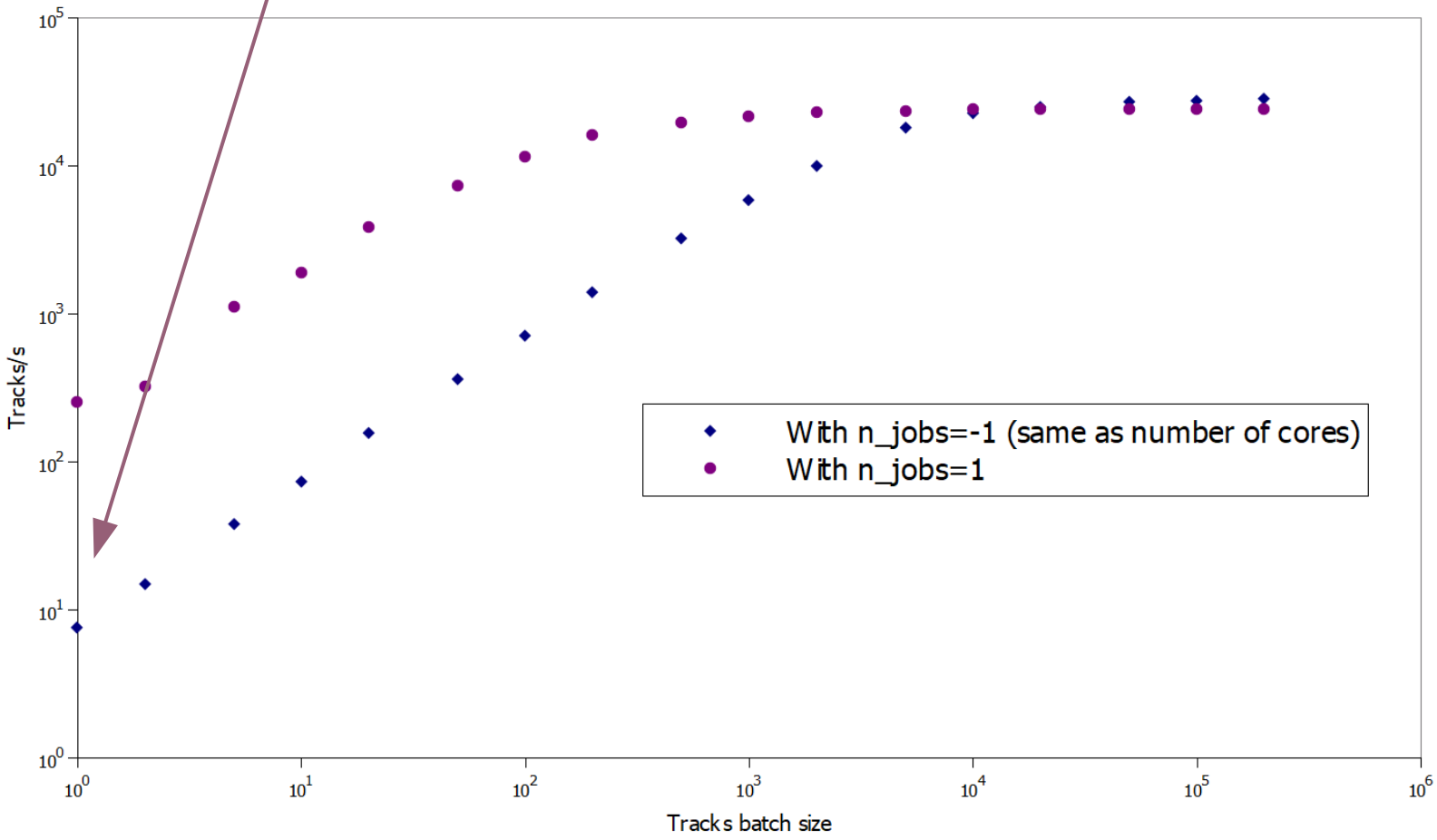
First attempt

- Track-by-track implementation
- Framework to iterate over events, loop over tracks in UserExec
- Classifier listens in the background
- Stripped files sent via pipe
- PID results received via another pipe
- The method is VERY SLOW



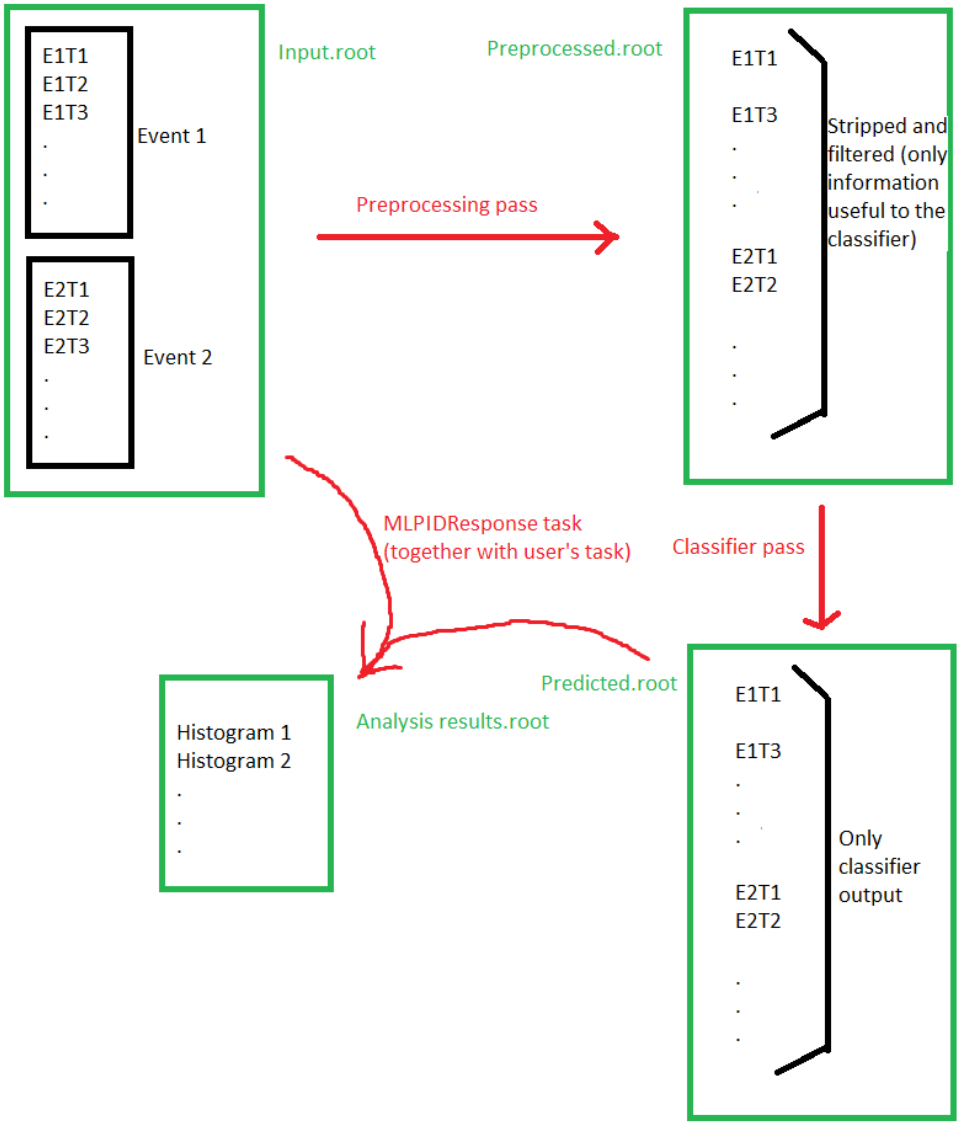
Scikit-learn benchmark

- In default track-by-track implementation, with threads, we can process only ~9 tracks/s (overhead from the thread creation) → no multiple threads allowed on the GRID
- Increase to more than 100 tracks/s if we do not allow threads
- Not very much difference with multiple tracks wrt single track



Second attempt

- Propagate multiple tracks through classifier
- Two loops over events needed
- No easy way in AliRoot:
  - create a temporary (stripped) file
  - propagate the temporary file through the classifier
  - produce predicted.root file
- In the second loop over events use a lookup table to match the two files



Third ~~attempt~~ idea

- Since there is no need to change the classifier by users, one can centralize the classification part as well
- Run the classification once (for example together with reconstruction pass or AOD creation) and store the classifier for every run
- Users would access the already existing ML PID attributes for a given run
- But... this also has some drawbacks:
  - no possibility to modify the classifier by the user
  - reliable Monte Carlo has to be ready before
  - no easy way to pair up events globally (see next slide)

Event pairing problem

How can we pair events from predicted.root with the ones a user gets in his/her analysis from the framework?

- **The predicted.root file consists of only those tracks for which PID information was available**
- **Tracks are not necessarily in the same order**
- **A track identification within the event is easy (`track->GetID()`)...**
- **... but no variable to identify tracks within the AOD file (across multiple events)**
  - candidates like `fTimeStamp` or `GetEventIDAsLong` (combines period, orbit, and bunch id) – may work with real data, but not present in MC

Our current proposal

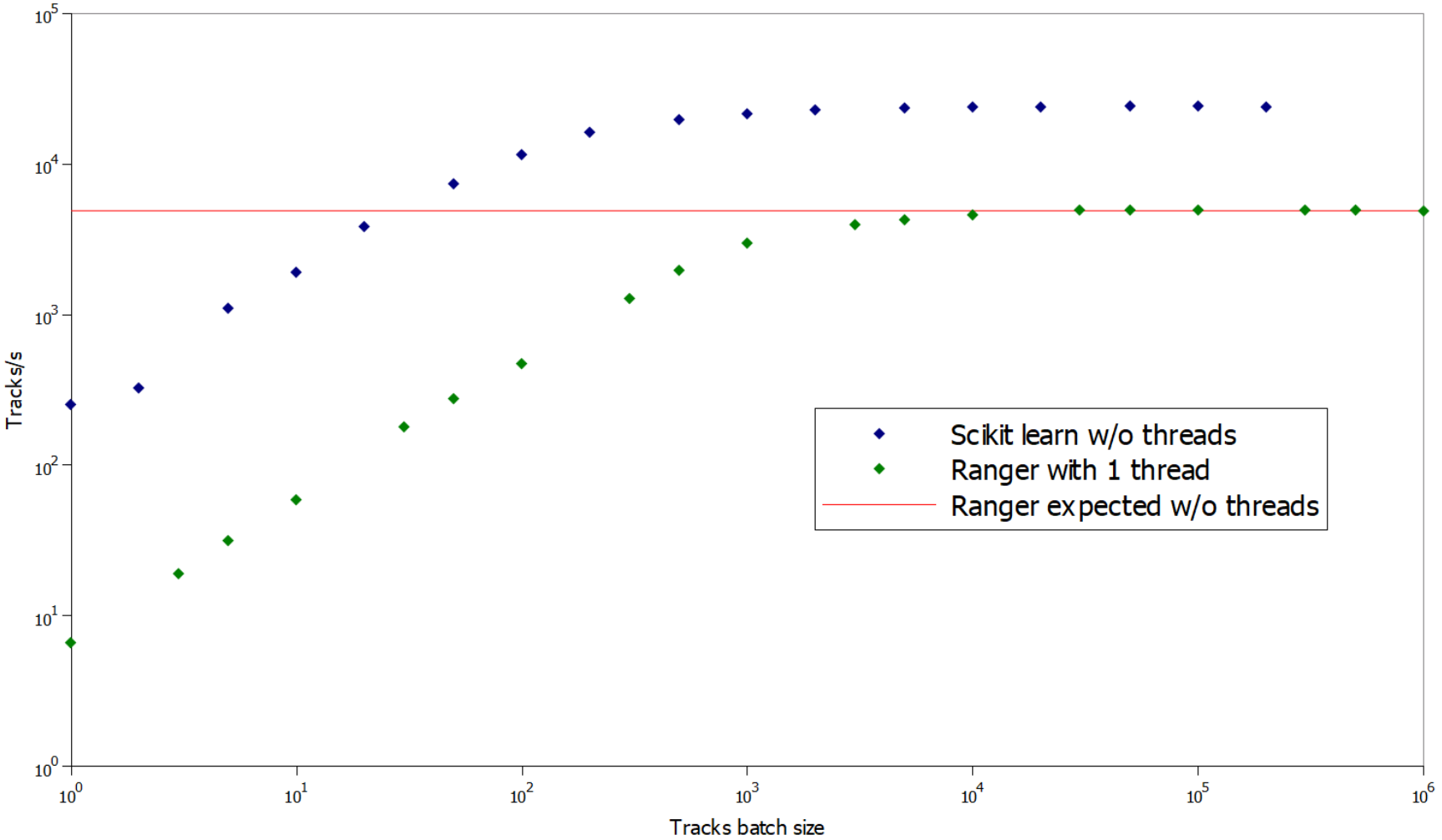
- **Propagate multiple tracks through the classifier combined from single events (do not combine multiple events)**
  - computational time of a simple p_T analysis task with ML PID (scikit-learn) and without ML PID (one 200 MB AOD file):

Real time 0:00:34 --- Without ML PID

Real time 0:01:33 --- With ML PID

  - the analysis with ML PID is 3x slower than without ML PID
- **If providing a framework in Python is not possible now, use the C++ Random Forest library (for example Ranger) instead of Python**
- **First tests:**
  - created a "random" C++ Random Forest of the same size and depth
  - compare Ranger and scikit-learn speed tests (next slide)

Scikit-learn vs Ranger



- Ranger (C++) is slower than scikit-learn (Python) → Python is faster
- Ranger creates threads even when set to 1 – we expect a speed up when removing that

Working demo/beta example

```
AliAnalysisTaskMLPt *myTask = new AliAnalysisTaskMLPt("MyTask");
AliAnalysisTaskMLPIDResponse *mlpidTask = new AliAnalysisTaskMLPIDResponse("MLPIDTask");

myTask->SelectCollisionCandidates(AliVEvent::kINT7);
if( !myTask )
  exit(-1);
mgr->AddTask(mlpidTask);
mgr->AddTask(myTask);

// Create containers for input/output
AliAnalysisDataContainer *cinput = mgr->GetCommonInputContainer();
AliAnalysisDataContainer *coutput2 = mgr->CreateContainer("MyTree",
            TList::Class(),AliAnalysisManager::kOutputContainer,outfilename);

//connect them to future analysis
mgr->ConnectInput(mlpidTask,0,cinput);
//mgr->ConnectOutput(mlpidTask,1,coutput2);
mgr->ConnectInput(myTask,0,cinput);
mgr->ConnectOutput(myTask,1,coutput2);

if ( !mgr->InitAnalysis() )
      return;
mgr->PrintStatus();

//start analysis
mgr->StartAnalysis("local",chain,Nevents);
```

run macro

user's analysis task

```
//loop over AOD reconstructed tracks
for (Int_t iTracks = 0; iTracks < aodEvent->GetNumberOfTracks(); iTracks++) {
      //get track
      AliAODTrack *track = (AliAODTrack*)aodEvent->GetTrack(iTracks);
      if (!track)
            continue;

      UInt_t filterBit = 96;
      if(!track->TestFilterBit(filterBit))
            continue;

if (!fMLpidUtil)
   continue;

AliMLPIDResponse* resp = fMLpidUtil->getTrackPIDResponse(track->GetID());
if (!resp)
   continue;
else
   cout<<"Good PID: "<<resp->predictedPDG<<endl;


int pdg = resp->predictedPDG;
if(pdg == 211)
   ptHistPions->Fill(track->Pt());
if(pdg == 321)
   ptHistKaons->Fill(track->Pt());
if(pdg == 2212)
   ptHistProtons->Fill(track->Pt());


      //save all attributes into TTree
      //treeOutput->Fill();
}
```

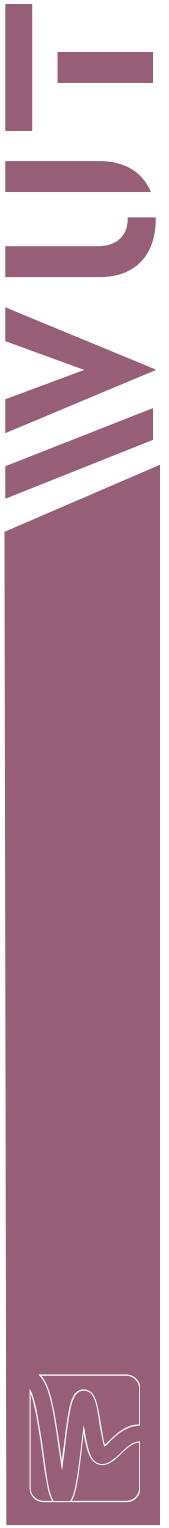
- User just needs to add a couple of lines – like for traditional PID response task

Summary

- **Advantages:**
  - ML-based PID outperforms traditional PID, clearly seen in practically all tests
  - training needed only once for each data set – no need for manual cut optimizations
- **Problems:**
  - Track-by-track implementation (optimal from our side) is very slow
  - No global track id information (across multiple data files) stored both in real data and MC data needed to match files
  - C++ ↔ Python connection is also a weak point

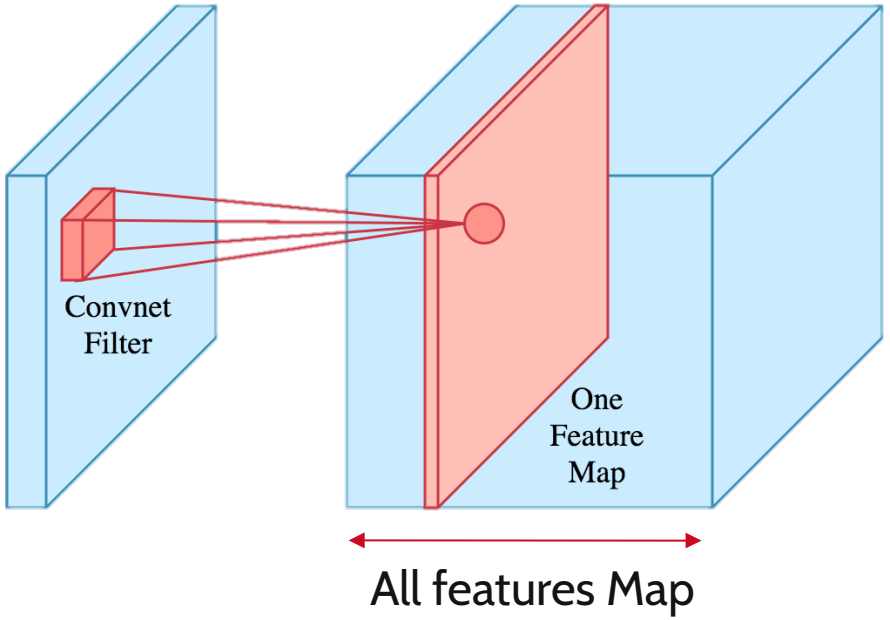
Thank you

Backup



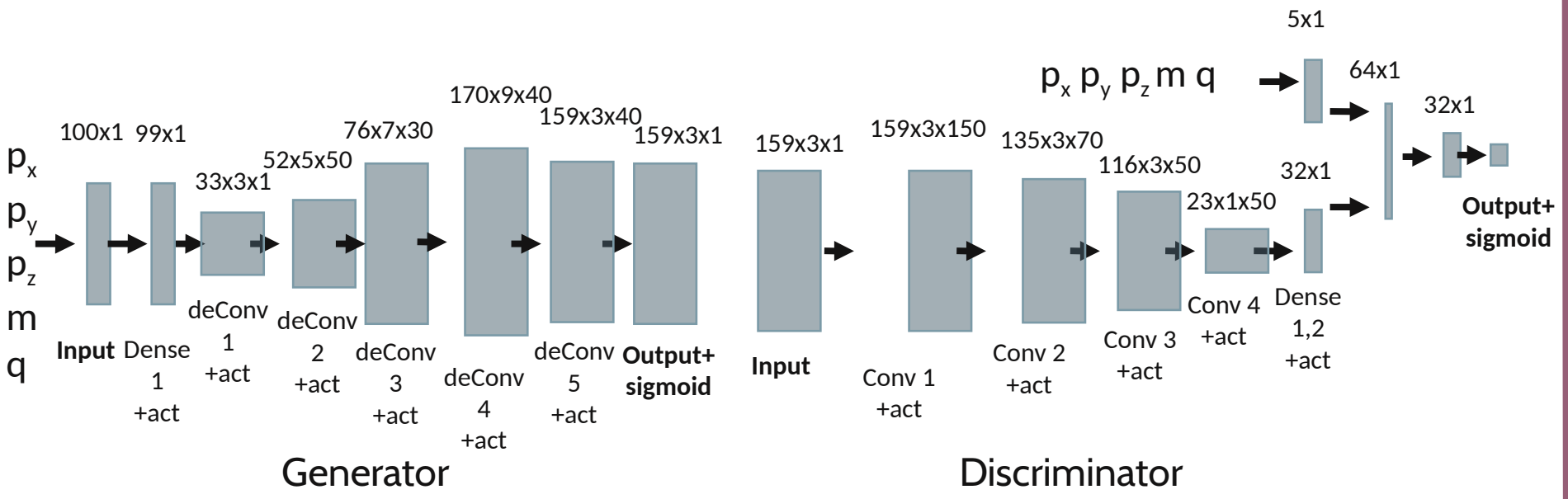
Deep Convolutional GAN

- Class of architectures which use the convolutional tools and deconvolutional layers – mostly used with images



condDCGAN: Conditional DCGAN

- Generator – deconvolutional layers
- Discriminator – convolutional layers
- Network conditioned on particle momenta, mass, and charge
- Output classification – sigmoid function



condDCGAN+: combined loss

- Training on on the full MC simulations
- Preparing the noise from initial parameters of MC simulations
- Comparing the generated samples with original ones
- Combining origininal conditional GAN loss with the results of comparison

$$\mathcal{L}_G(m, X) = \mathbb{E}_{z \sim p_z(z|m)}[\alpha \log(1 - D(G(z))) + \beta \frac{1}{n} \sum_{i=1}^{n} (X_i - G(\hat{z})_i)^2]$$

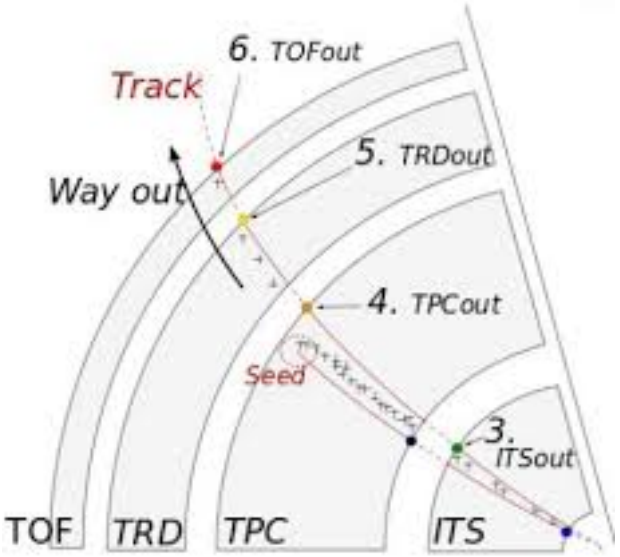
m - initial parameters (particle momenta),
X - original value corresponding to m ,
p(z|m) - distribution of a noise vector under initial parameters m
z - input into a generator
G and D - generator and discriminator
n - the number of produced clusters

Additional parameters α and β are used to weight the share of individual losses.
Best performing values are α = 0.6 and β = 0.8

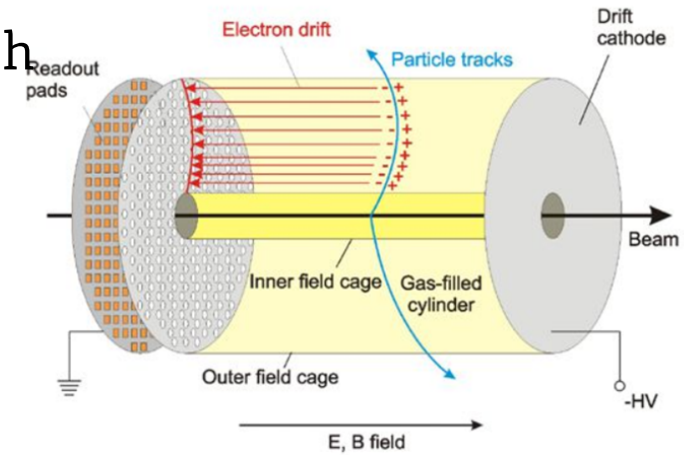
Simulation of TPC clusters in Monte Carlo data using generative networks

Time Projection Chamber

- **Tracking in ALICE is performed by ITS, TPC, TRD and TOF**
- **First attempts – focus on the TPC only:**
  - main tracking device
  - located from 0.8 m (inner radius) to 2.5 m (outer radius) from the beam and extending ~2.5 m in each direction along the beam axis
  - volume of 95 m^3
  - filled with Ne-CO_2 gas mixture (90%-10%)
  - **clusters** - points in 3D space, together with the energy loss, which were presumably generated by a particle traveling through
  - provides up to 159 clusters per track



ALICE Data Preparation Group

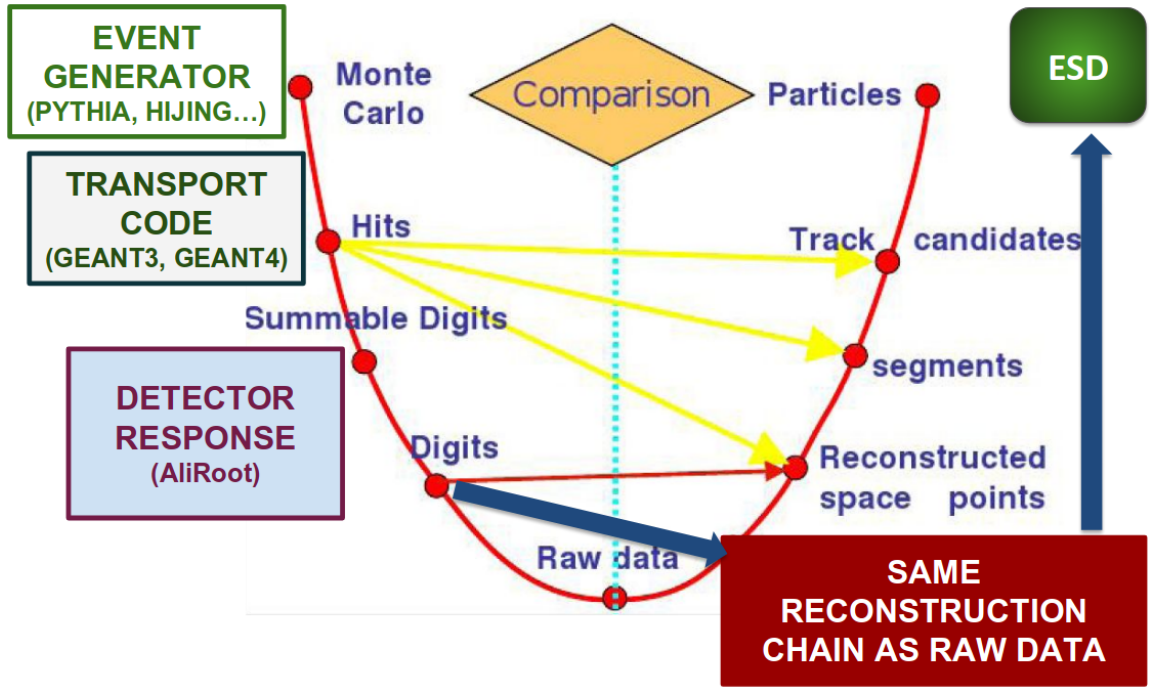


I.Konorov, Front-end electronics for Time Projection chamber

Simulation and reconstruction

- Current process relies on 5 independent modules
- The computationally most expensive module is particle propagation through the detector's matter

Collision generator → Particle propagation → Electronic signals (digits) → Digits to Clusters → Tracking

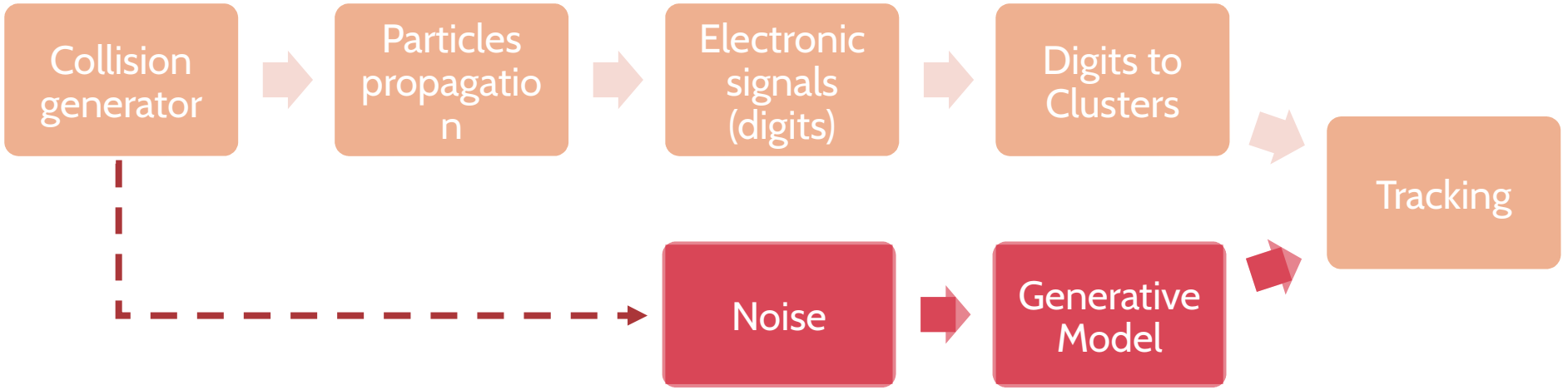


Monte Carlo simulation

Real data

ALICE Data Preparation Group

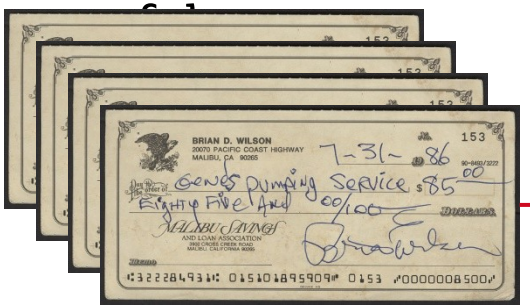
Simulation and reconstruction



- **Generative solution for cluster simulation:**
  - substitute the detector simulation and check for the speed-up
  - full simulation **still needed** to generate training samples
  - **immediate drawback:** quality of such MC data can be either comparable or lower than the full detector simulation – limits potential applications

Generative Adversarial Networks

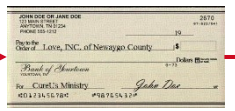
- Generative Adversarial Network (GAN) is a neural network architecture of two networks competing with each other (playing a min-max game)
  - **"Generator"** is trained to produce fake data resembling the real data
  - **"Discriminator"** aims to predict whether an example data is real or



https://33milesinnewaygocounty.files.wordpress.com

Generator

https://giphy.com/gifs/leonardo-dicaprio-catch-me-if-you-can-5leocharacters-t1h4nnWEWKfn2



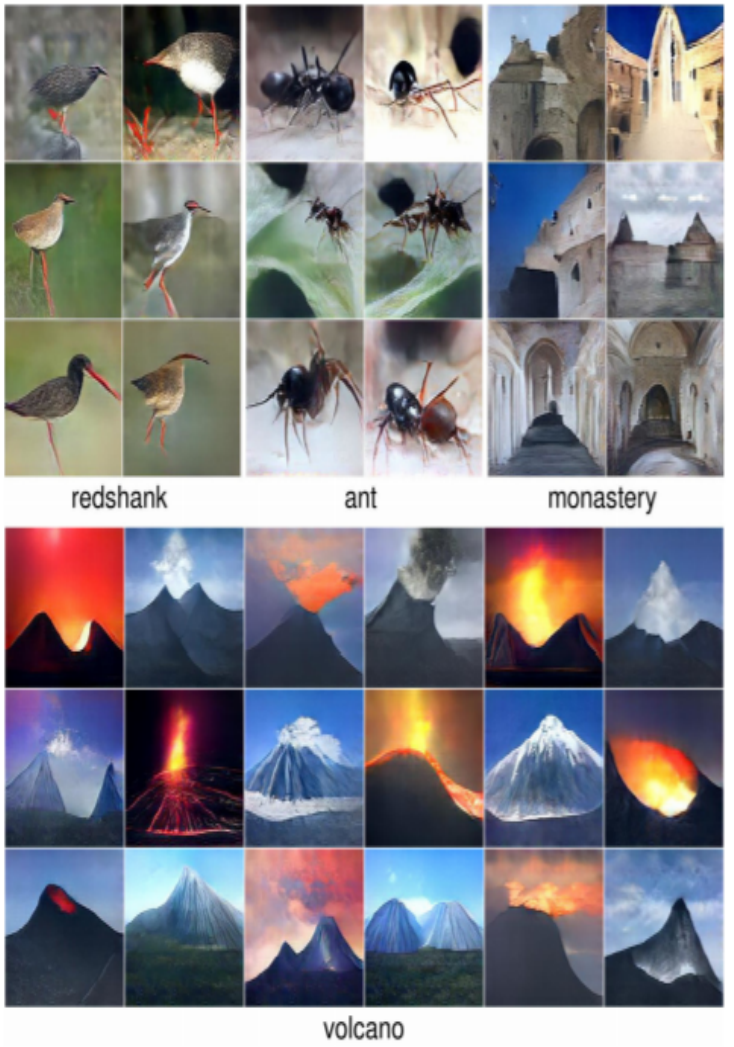
Discriminator

https://thechive.files.wordpress.com

Generative Adversarial Networks

- Typical use cases:
  - mainly generation of photo quality fake images (i.e. of celebrities)

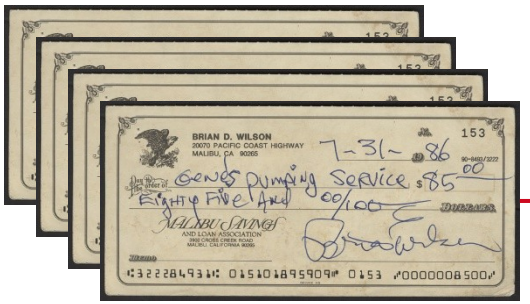
https://arxiv.org/abs/1710.10196



https://arxiv.org/pdf/1612.00005v1.pdf

Generative Adversarial Networks

- **Extending the GAN architecture – provide a set of initial parameters for the generator and discriminator:**
  - generator would not generate a random output, but a customized one
  - in our case: initial momenta of Monte Carlo particles



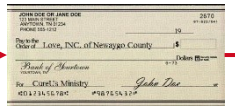
https://33milesinnewaygocounty.files.wordpress.com

Discriminator

https://thechive.files.wordpress.com

Generator

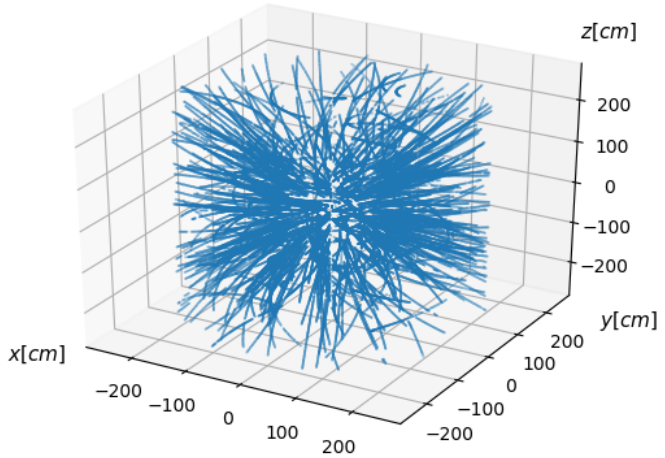
https://giphy.com/gifs/leonardo-dicaprio-catch-me-if-you-can-5leocharacters-t1h4nnWEWKfn2

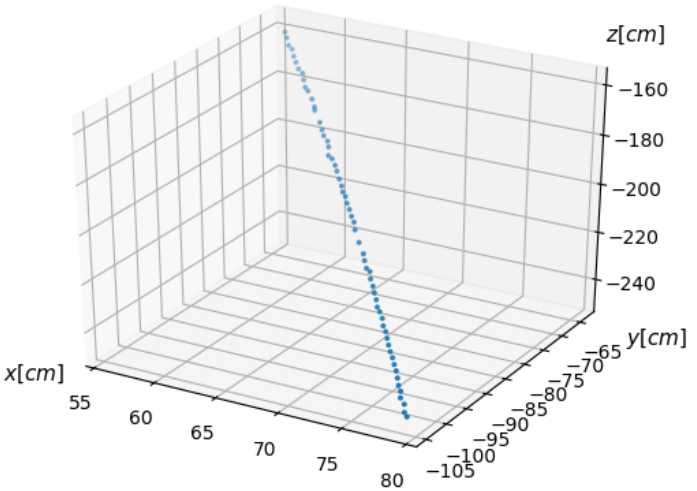


Initial Parameters

TPC clusters with GANs

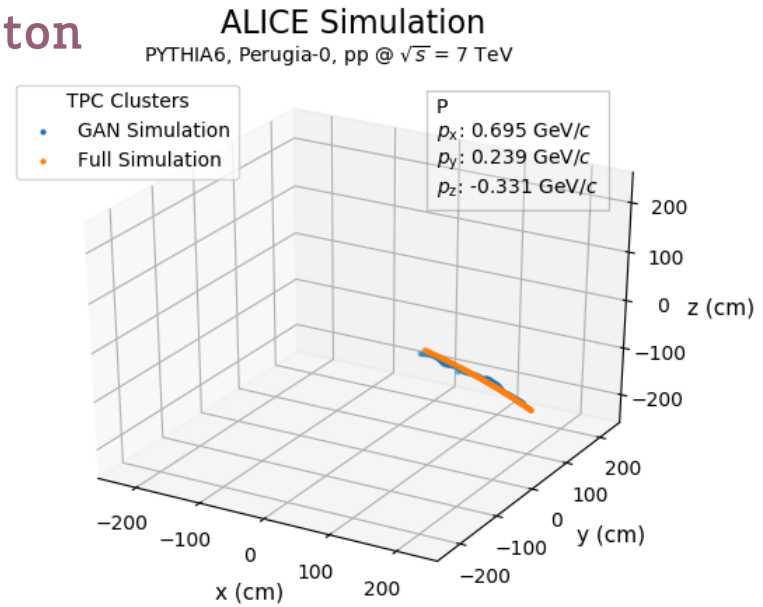
- It is not (yet!) possible to generate the full 3D image of the event at once (especially in the Pb-Pb event)
- Our solution is to:
  - generate clusters for single particles
  - two separate flows for spatial coordinates (x,y,z) and the energy
  - in the beginning focus only on 3D coordinates
  - merge generated samples (individual particles) into full images
  - training of the GAN on original full simulations



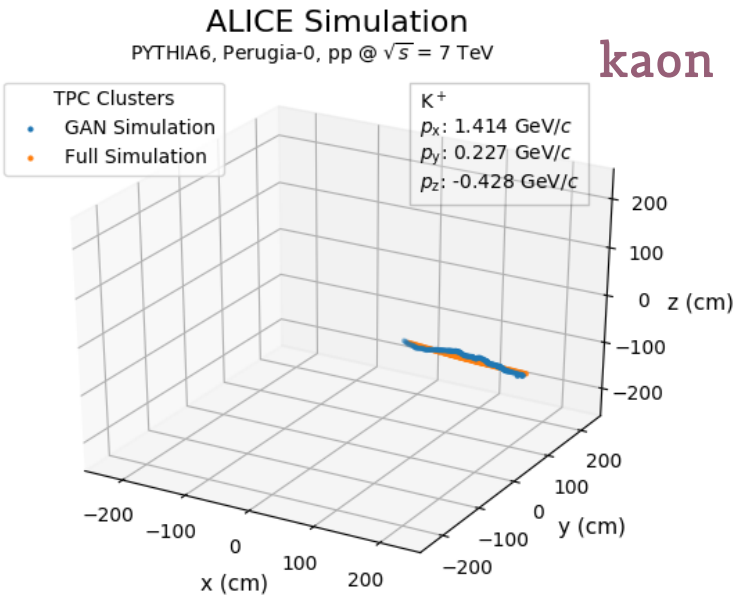


Example results

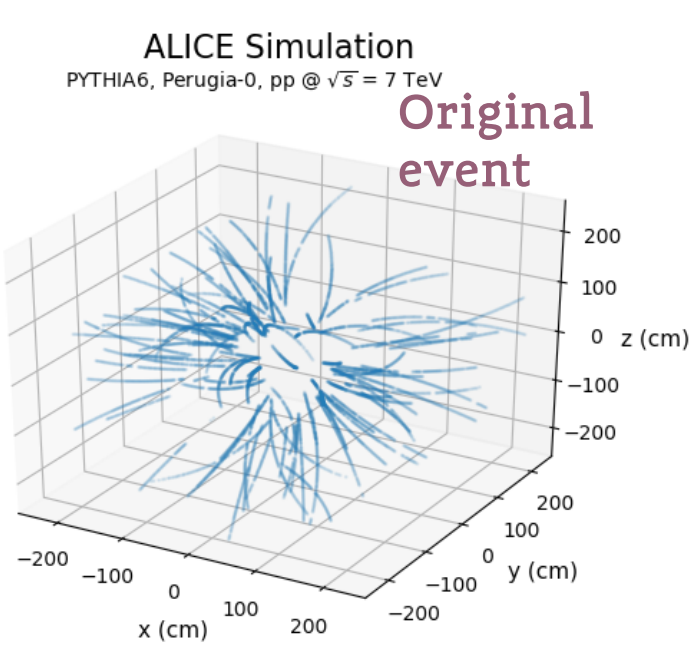
proton



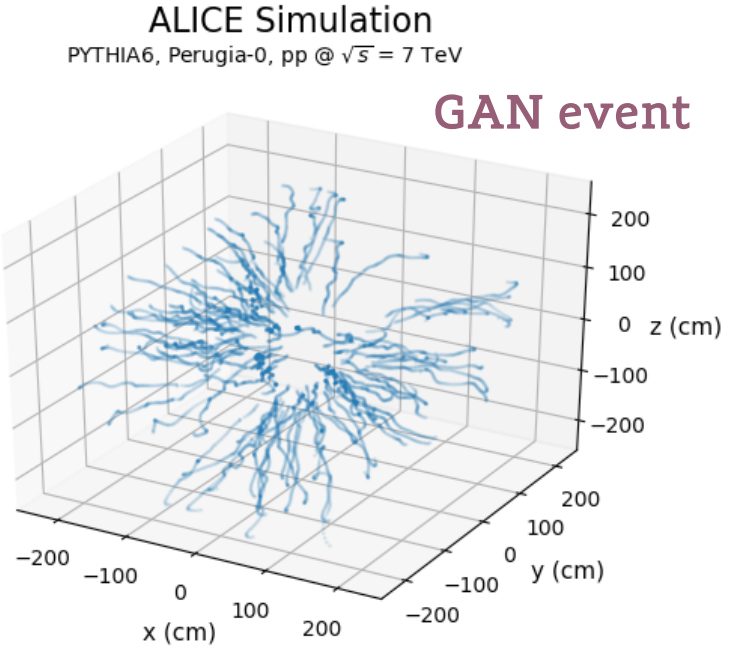
kaon



Original event



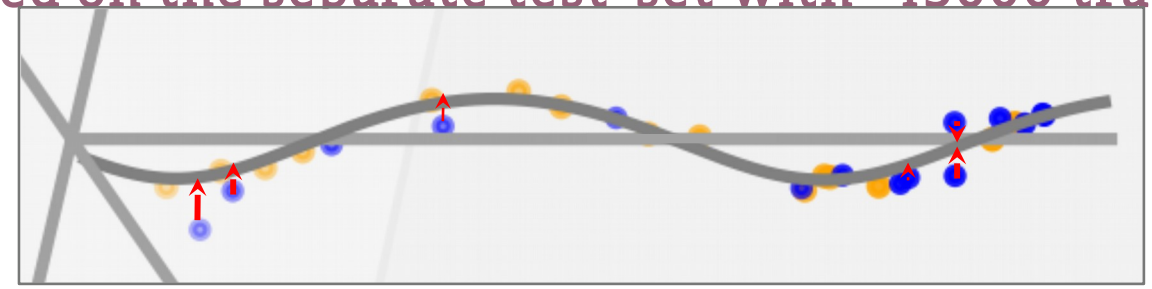
GAN event



Results

- Mean Squared Error (MSE) from the original helix as a quality measure
- Evaluation conducted on the separate test-set with ~15000 tracks

MSE visualisation:
Red - error
Grey- ideal helix
Orange – original clusters
Blue – generated clusters



Method	Mean MSE (mm)	Median MSE (mm)	Speed-up
GEANT3	1.20	1.12	1
Random (estimated)	2500	2500	N/A
condLSTM GAN	2093.69	2070.32	100
condLSTM GAN+	221.78	190.17	
condDCGAN	795.08	738.71	25
condDCGAN+	**136.84**	**82.72**	

Computational cost

- Performance test conducted on the standalone machine with Intel Core i7-6850K (3.60 GHz) CPU using single core and no GPU
- Additional order of magnitude speed-up for GAN models with nVidia Titan Xp GPU