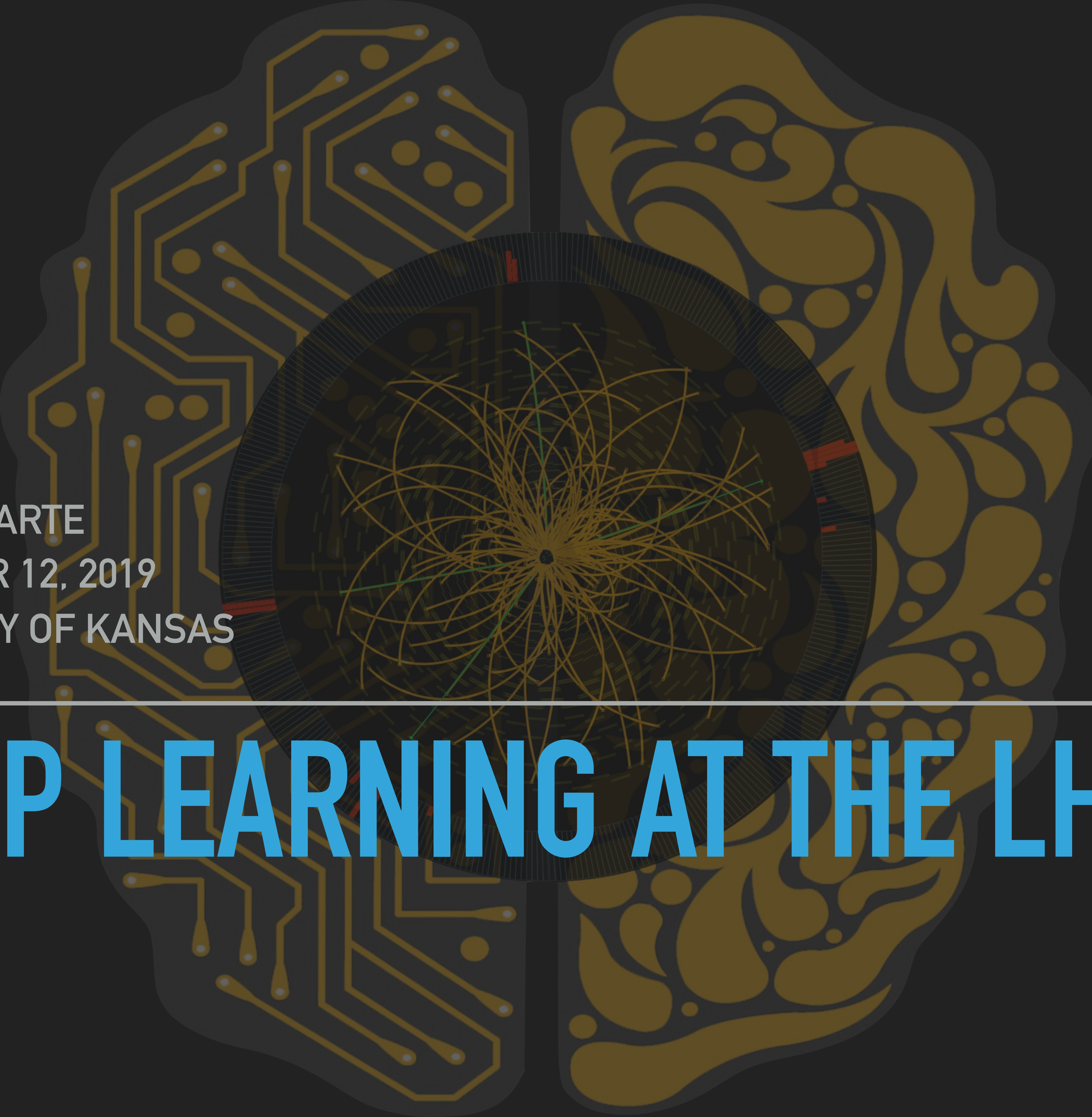


JAVIER DUARTE

NOVEMBER 12, 2019

UNIVERSITY OF KANSAS

DEEP LEARNING AT THE LHC

THE LARGE HADRON COLLIDER

THE LARGE HADRON COLLIDER

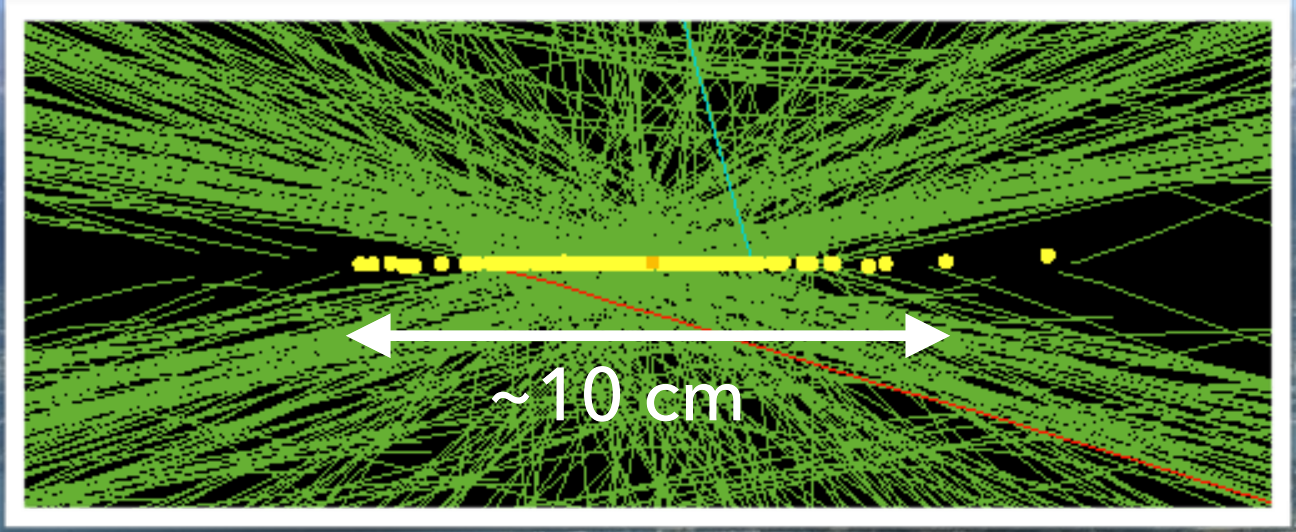
proton-proton collider @ 13 TeV center-of-mass energy

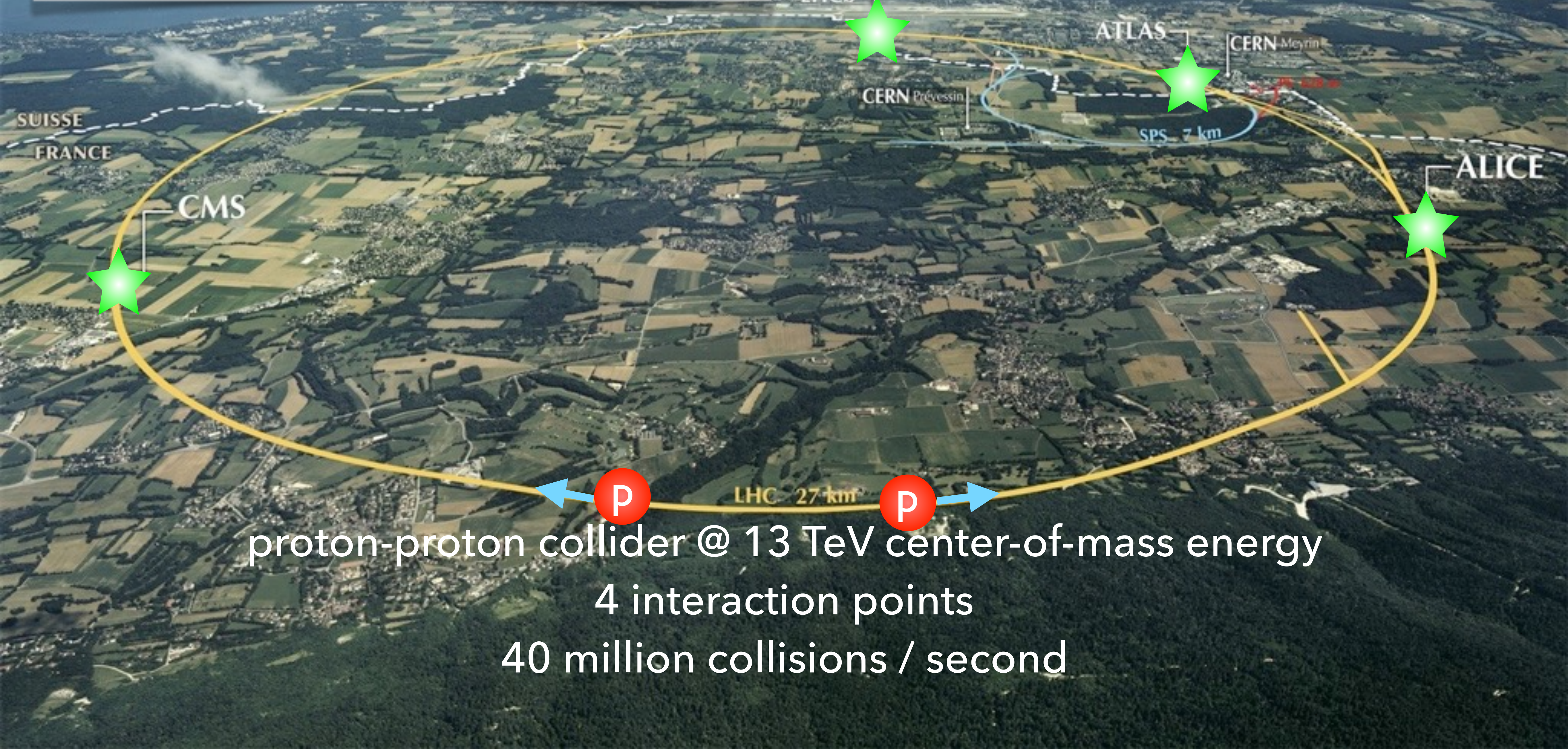
THE LARGE HADRON COLLIDER

proton-proton collider @ 13 TeV center-of-mass energy

4 interaction points

THE LARGE HADRON COLLIDER



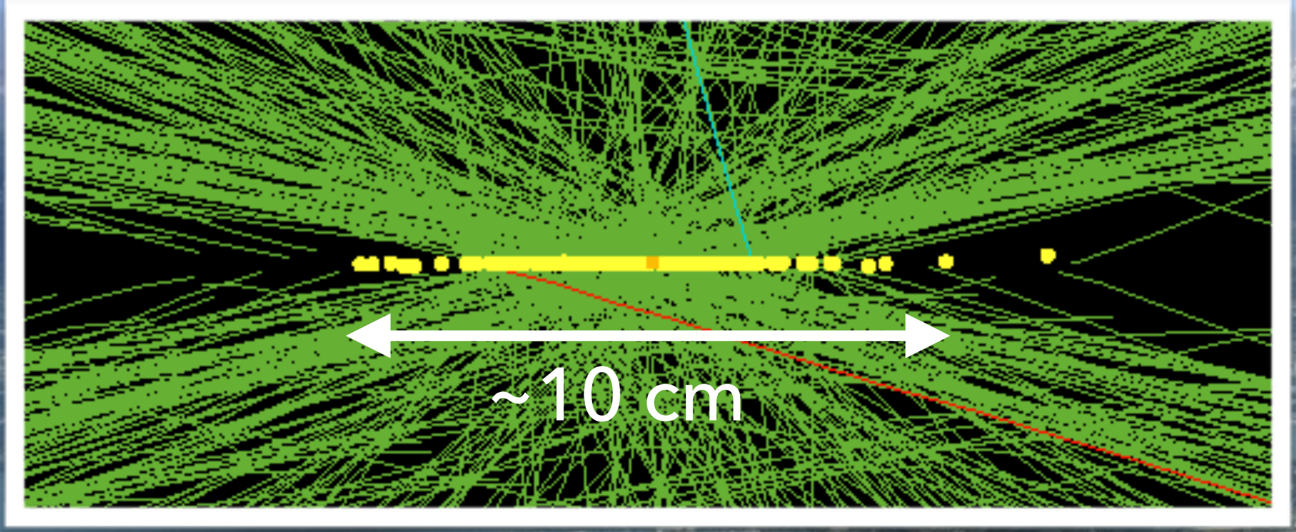


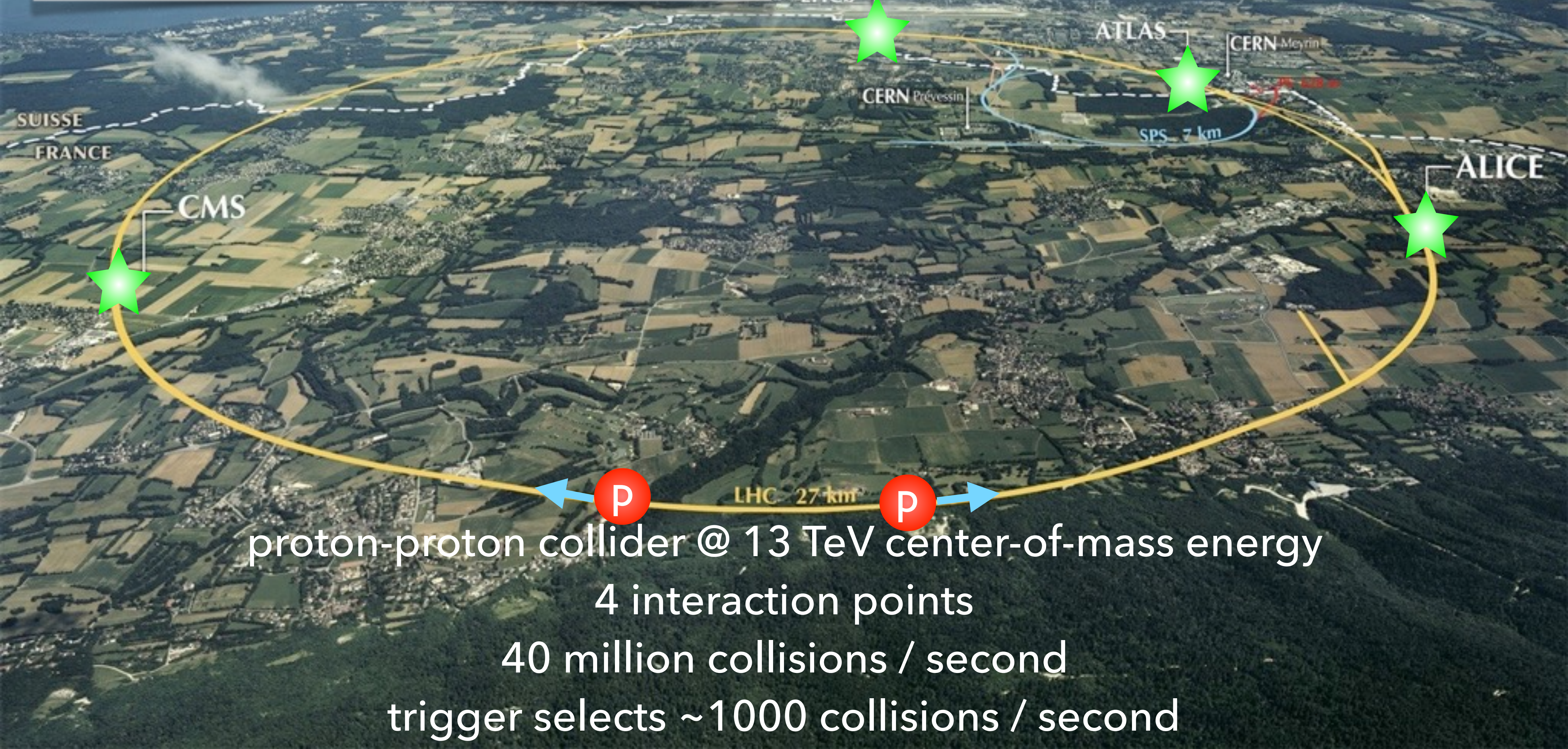
proton-proton collider @ 13 TeV center-of-mass energy

4 interaction points

40 million collisions / second

THE LARGE HADRON COLLIDER





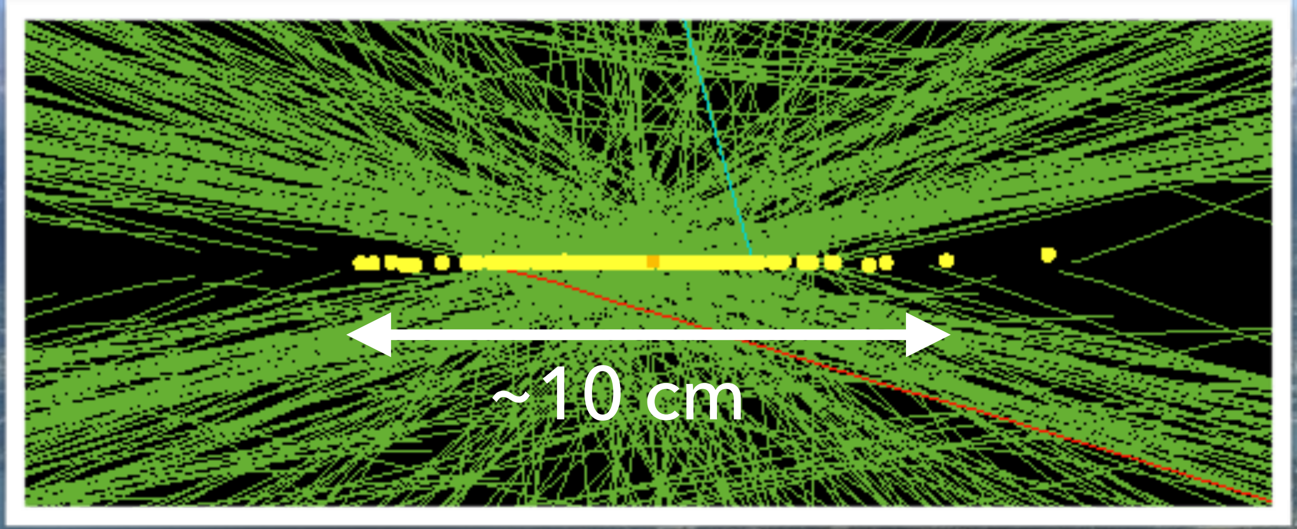
proton-proton collider @ 13 TeV center-of-mass energy

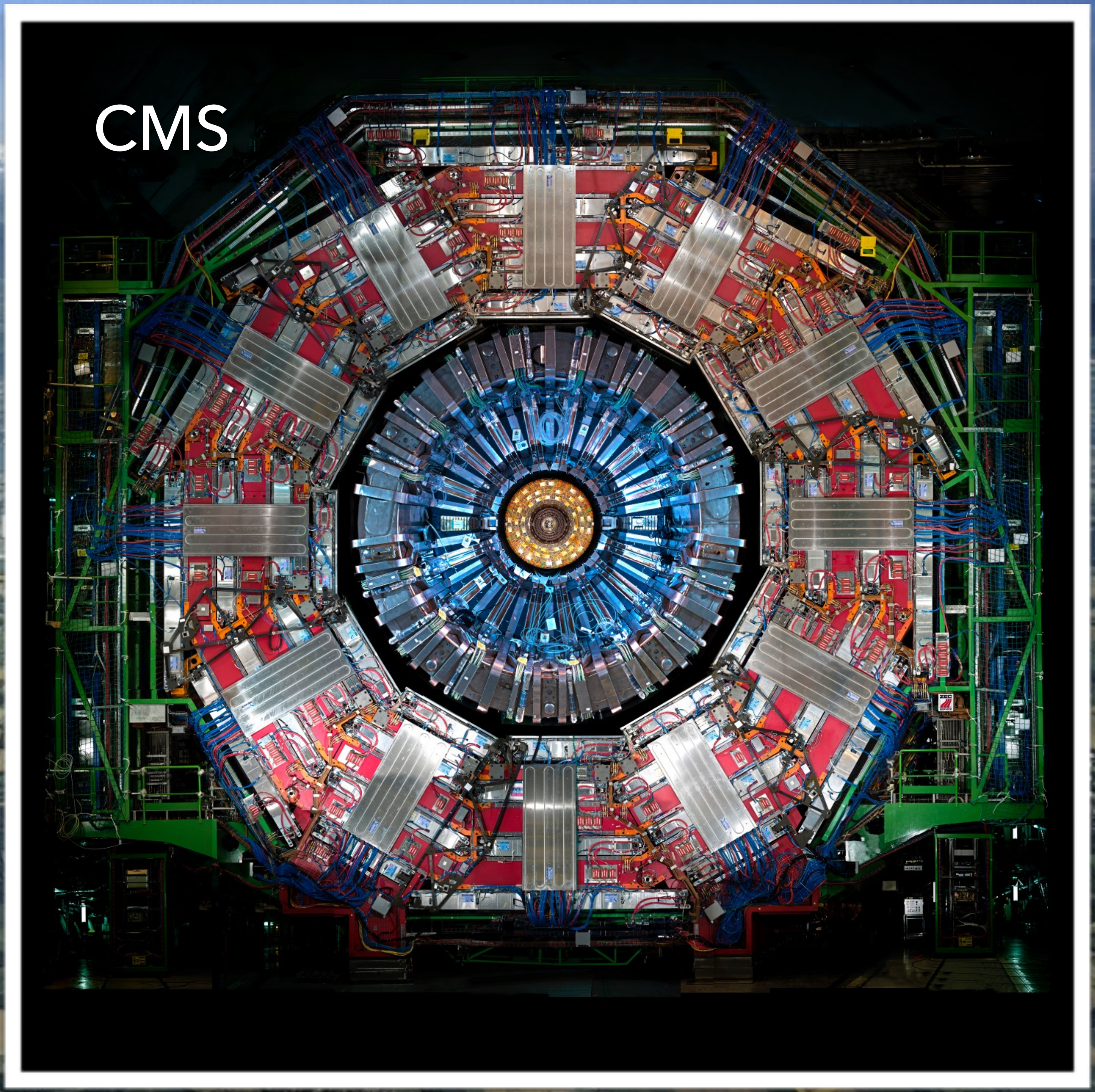
4 interaction points

40 million collisions / second

trigger selects ~1000 collisions / second

THE LARGE HADRON COLLIDER





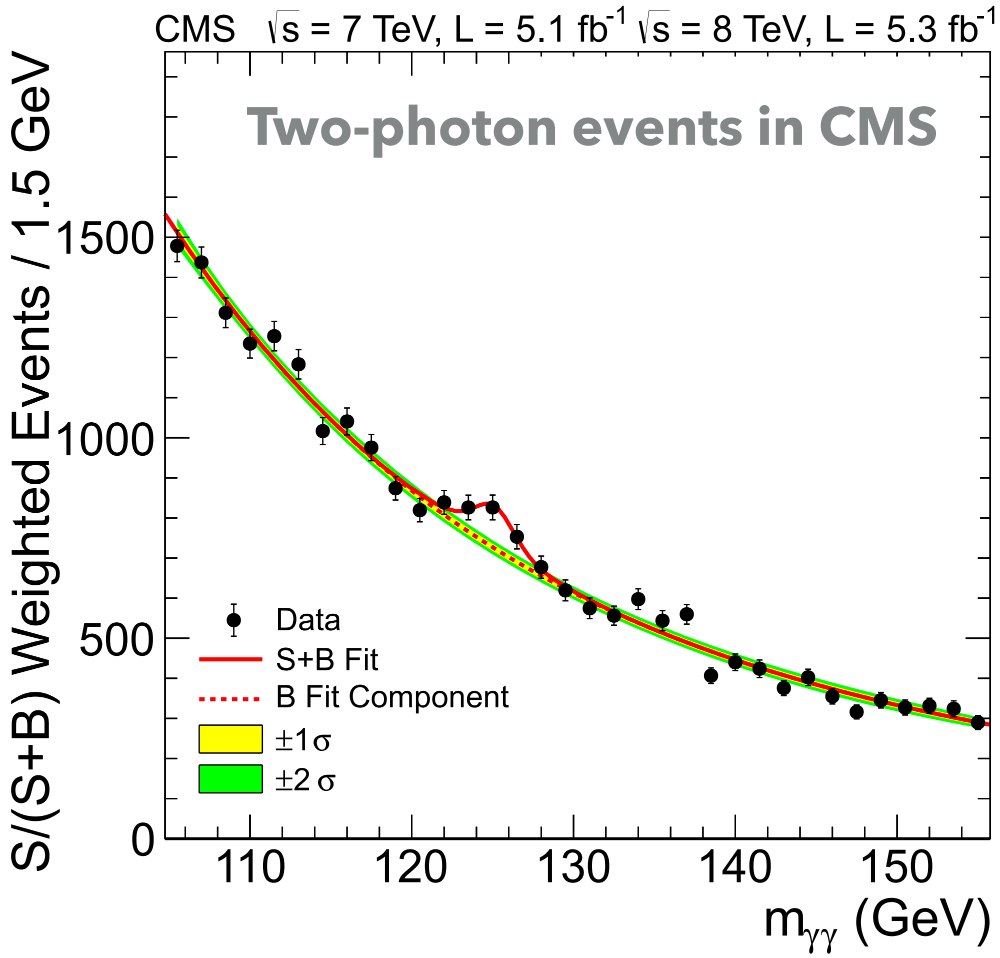
proton-proton collider @ 13 TeV center-of-mass energy

4 interaction points

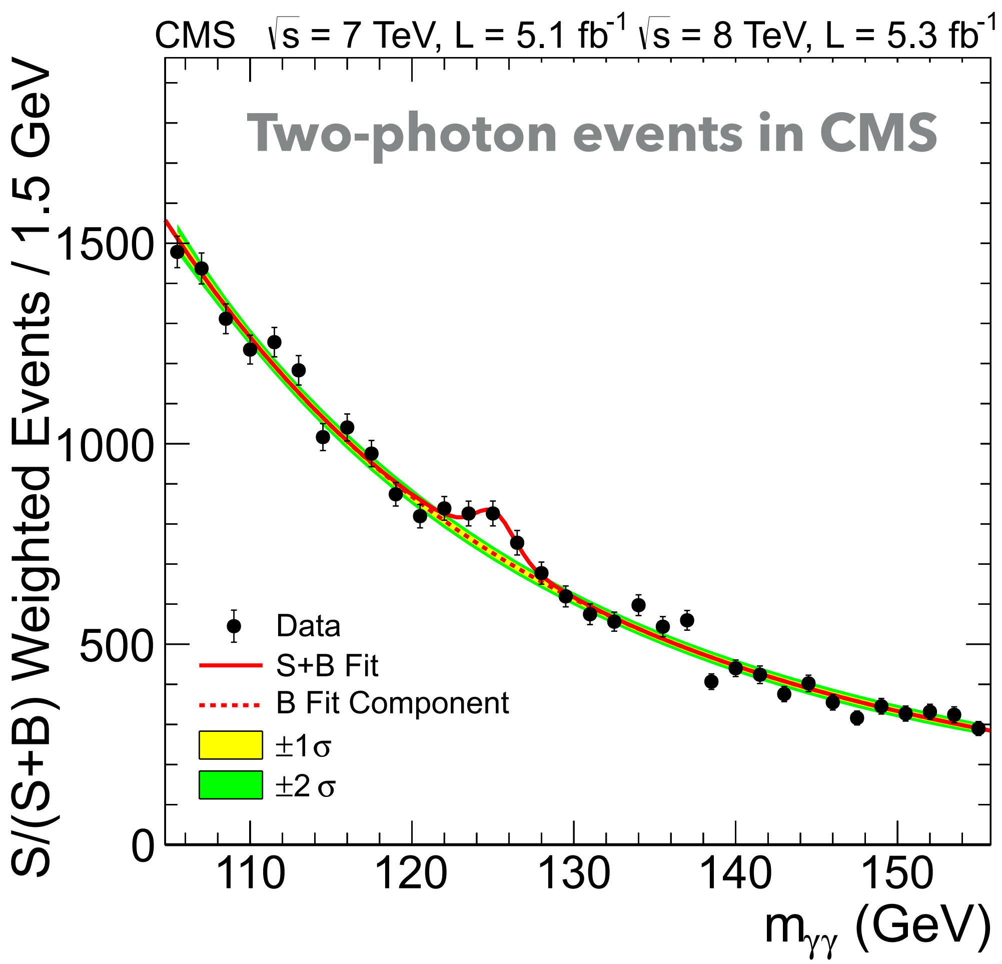
40 million collisions / second

trigger selects ~1000 collisions / second

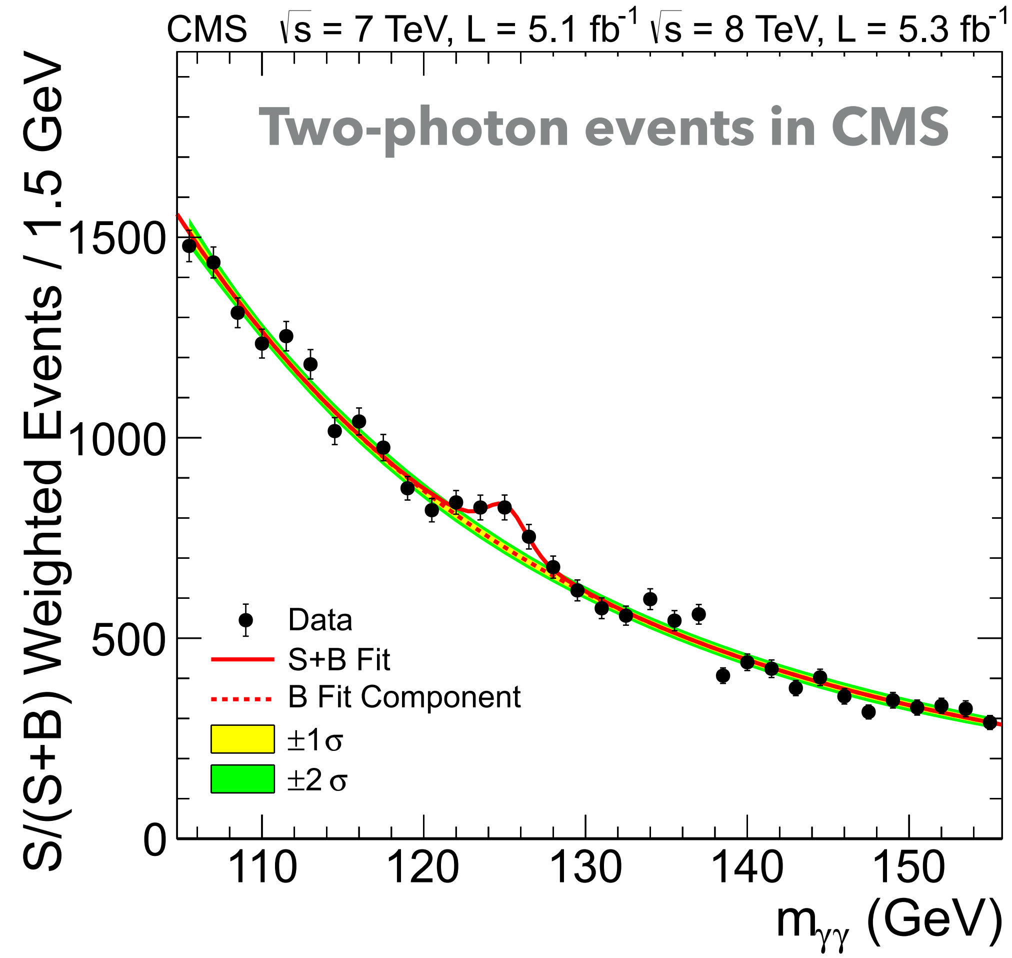
HIGH ENERGY PHYSICS + MACHINE LEARNING

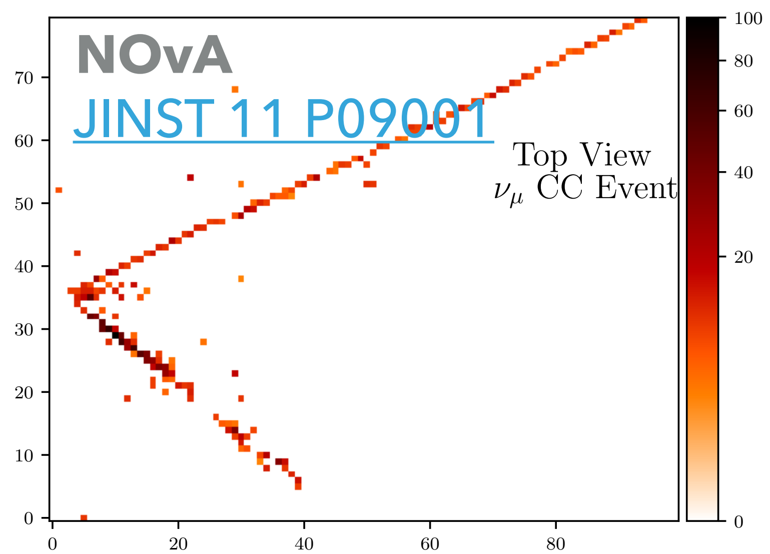


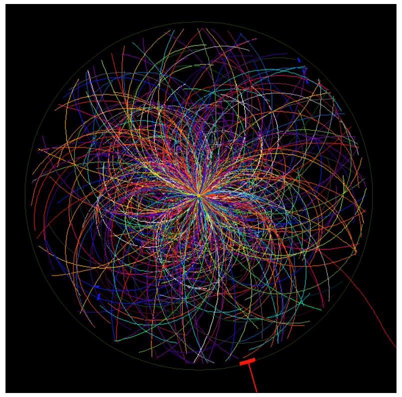
- **Machine learning** was vital to make big discoveries like the Higgs boson on July 4, 2012

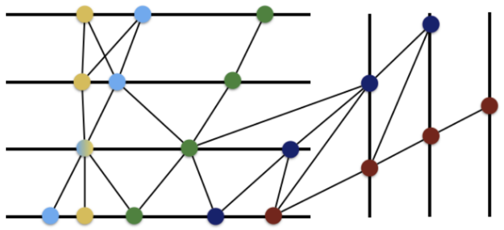


- **Machine learning** was vital to make big discoveries like the Higgs boson on July 4, 2012
- Today, ML is **enabling** new detection techniques, measurements, and searches







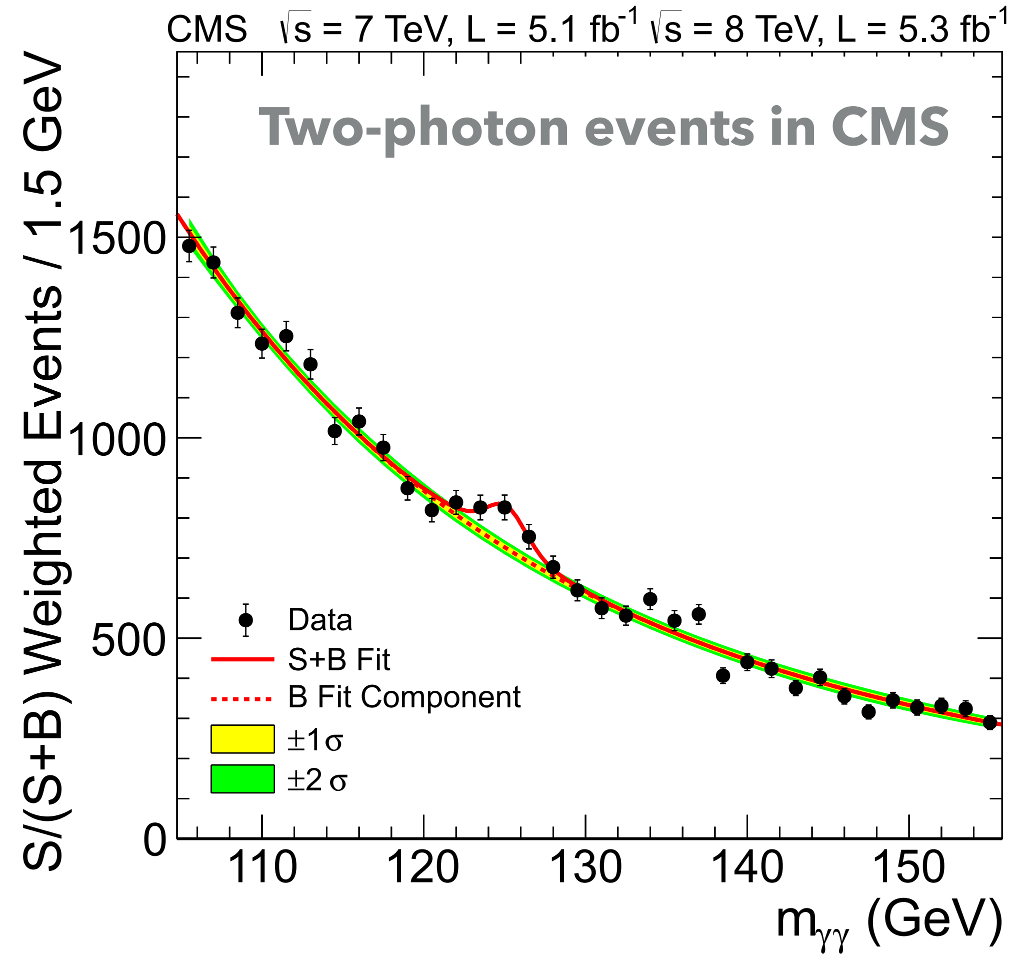


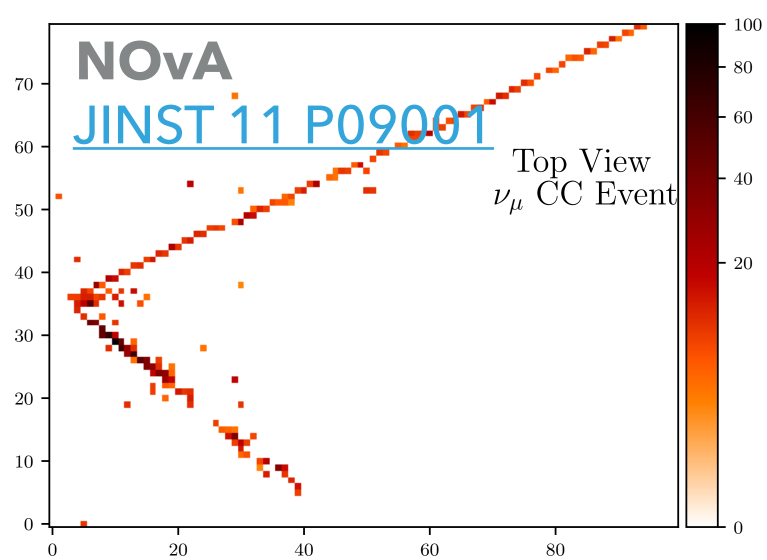
HEP.TrkX

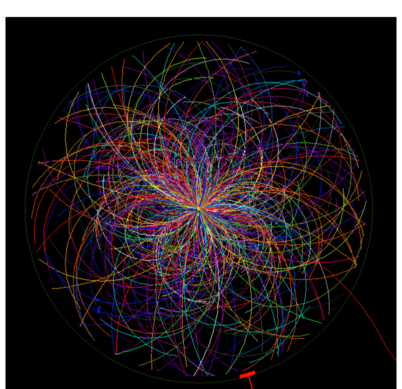
HIGH ENERGY PHYSICS + MACHINE LEARNING

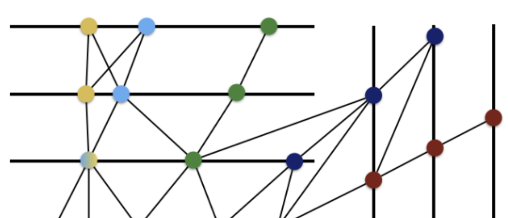
- **Machine learning** was vital to make big discoveries like the Higgs boson on July 4, 2012

- Today, ML is **enabling** new detection techniques, measurements, and searches







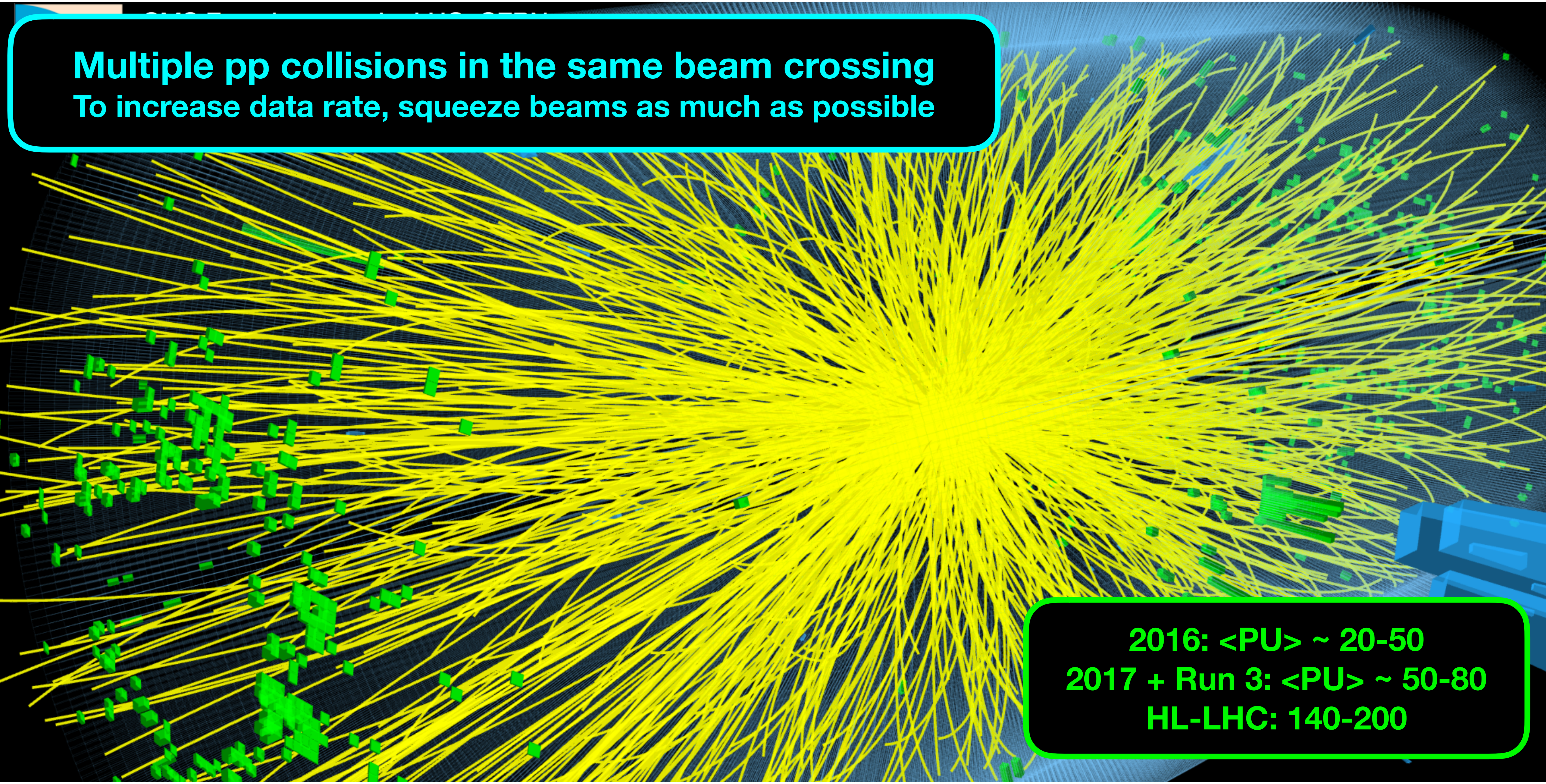


HEP.TrkX

- At the same time, we must **plan** how we will overcome challenges in the **next generation of experiments**

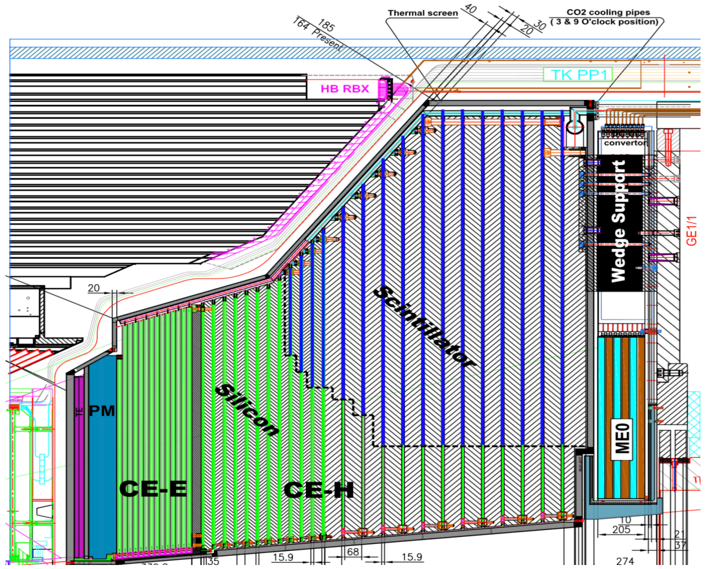
- ML may be a way out

CHALLENGE: PILEUP



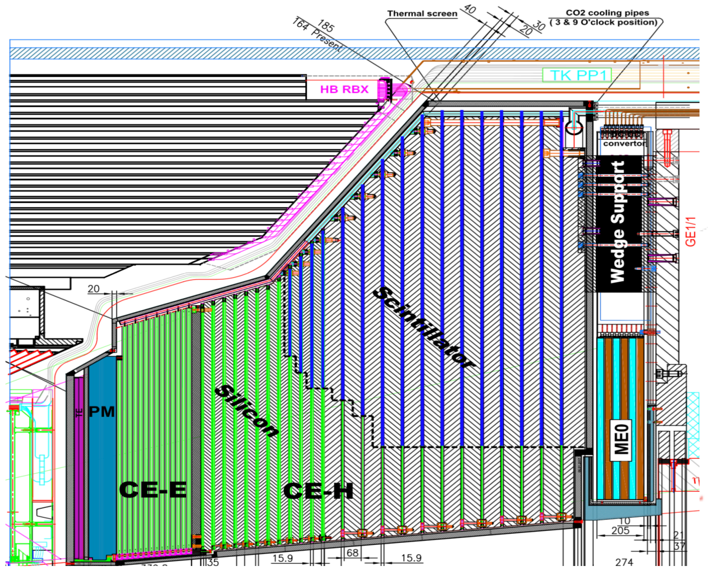
- At high luminosity, many collisions happen simultaneously (pileup)!
- Pileup makes our data more complex and noisy

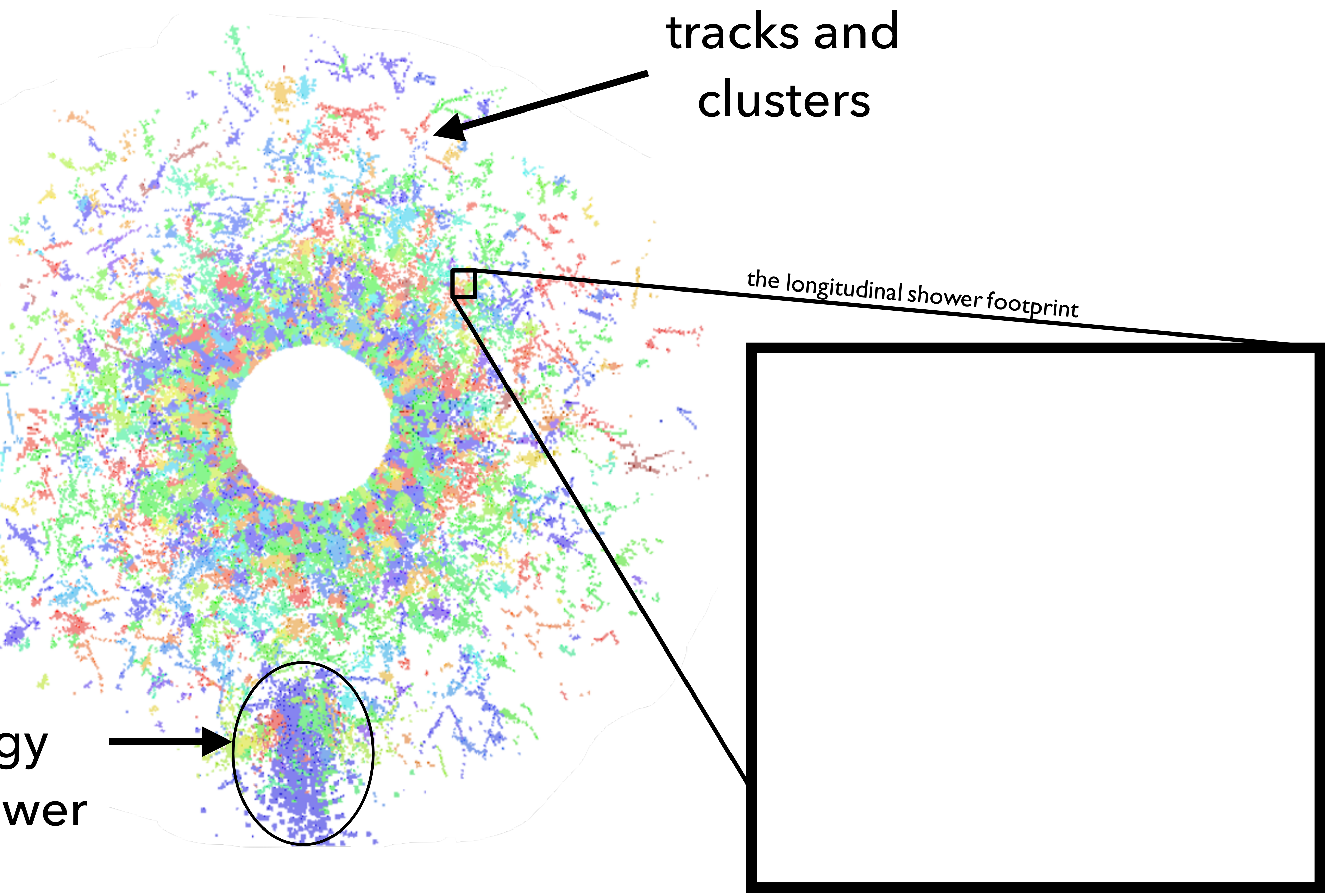
CHALLENGE: NEW DETECTORS



- High Granularity Calorimeter will provide 3D information of a particle shower as it evolves

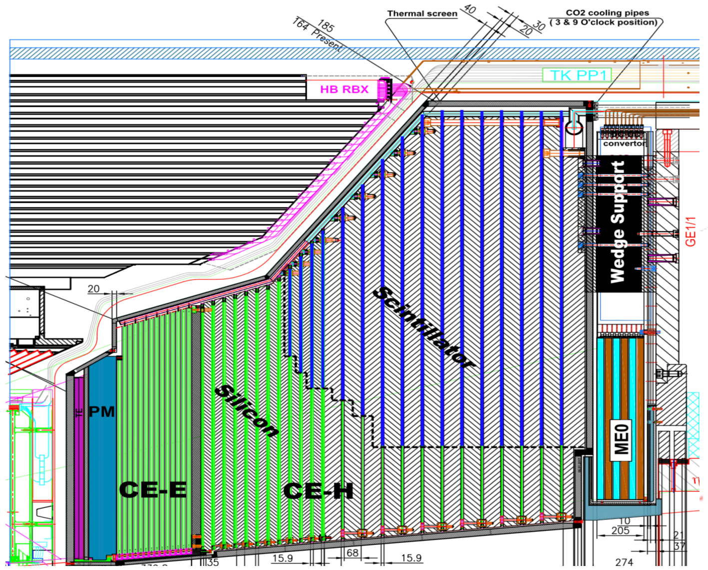
CHALLENGE: NEW DETECTORS

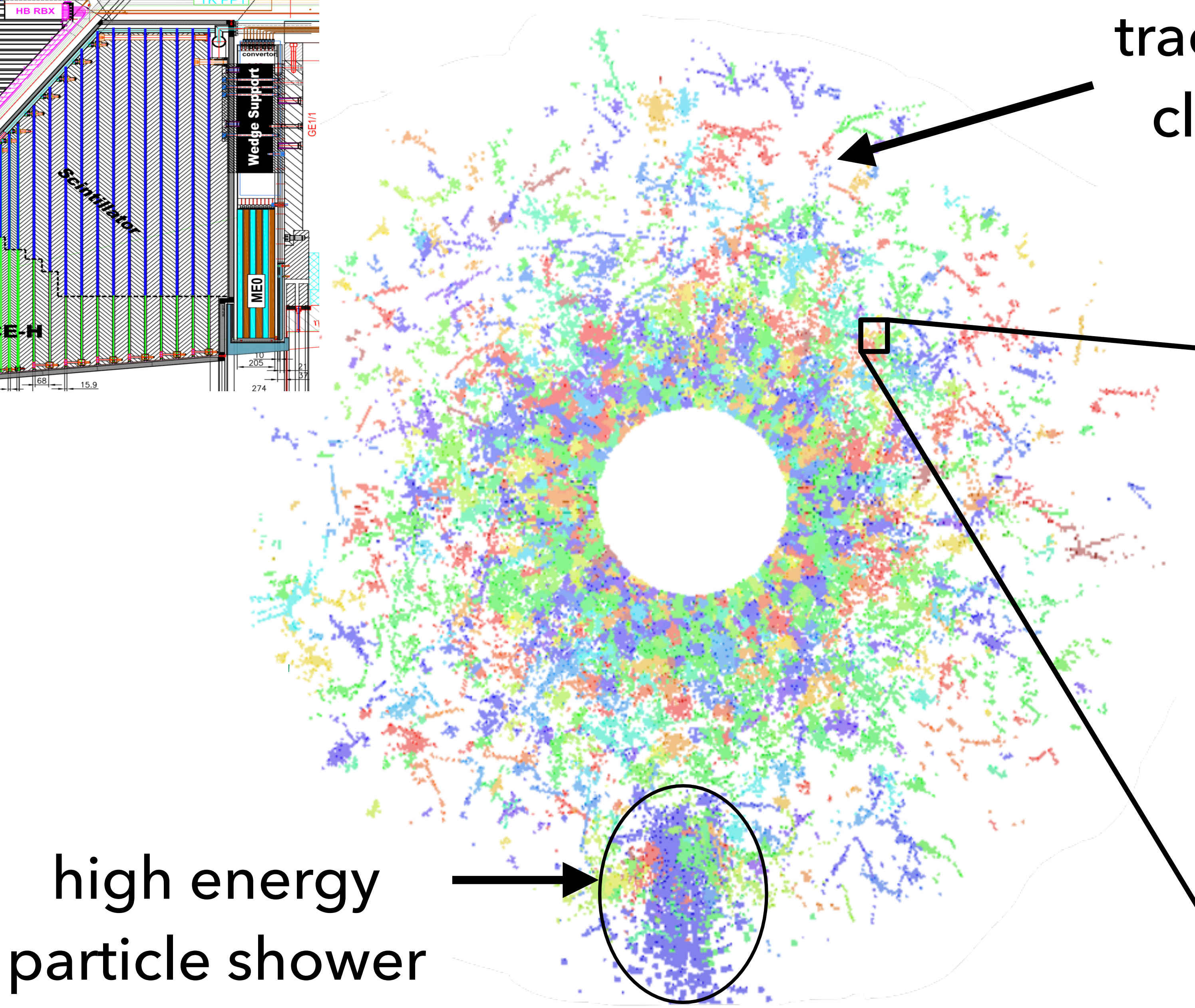


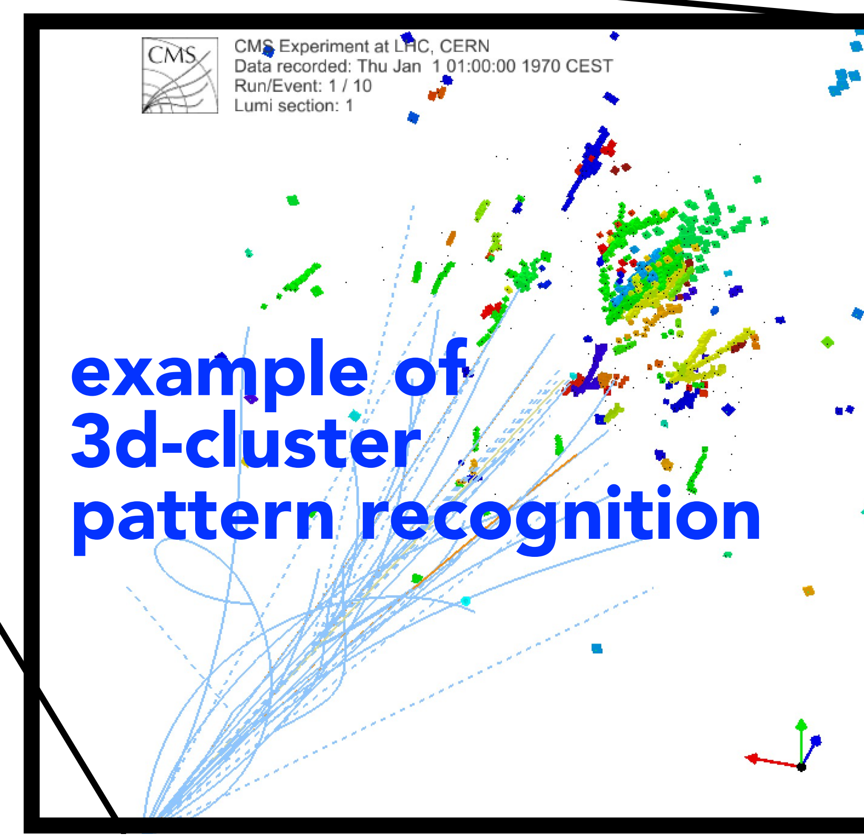


- High Granularity Calorimeter will provide 3D information of a particle shower as it evolves

CHALLENGE: NEW DETECTORS

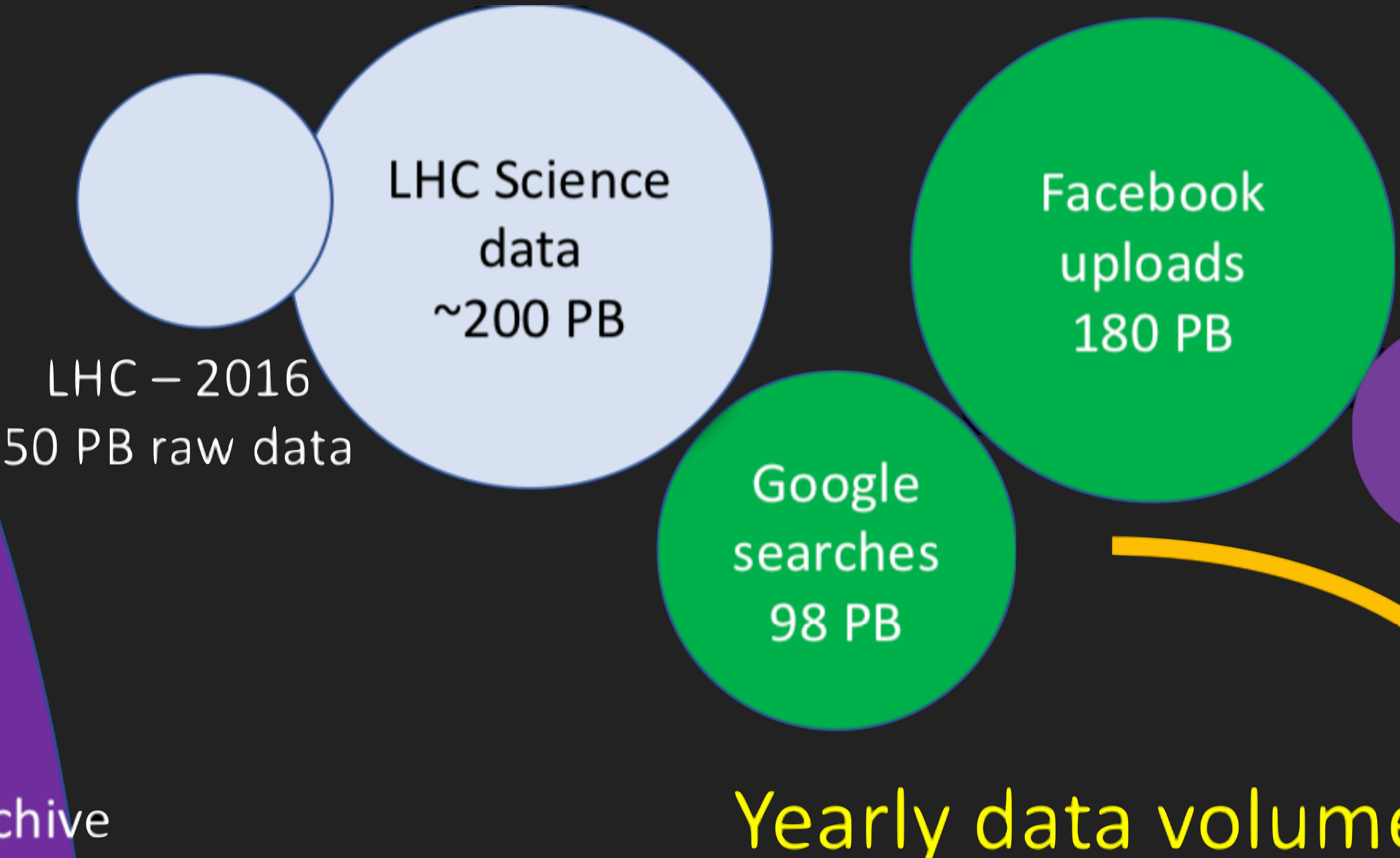






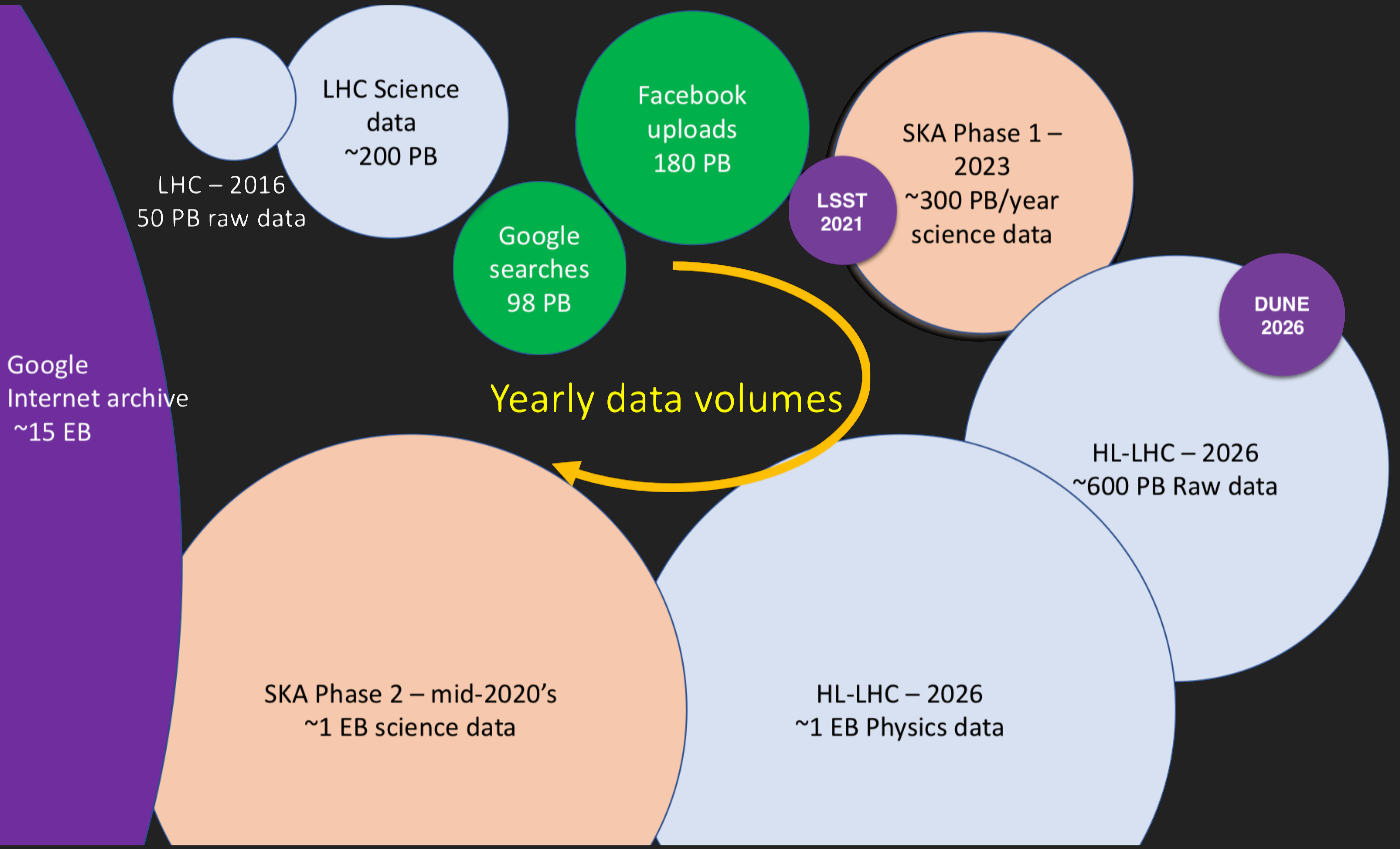
- High Granularity Calorimeter will provide 3D information of a particle shower as it evolves

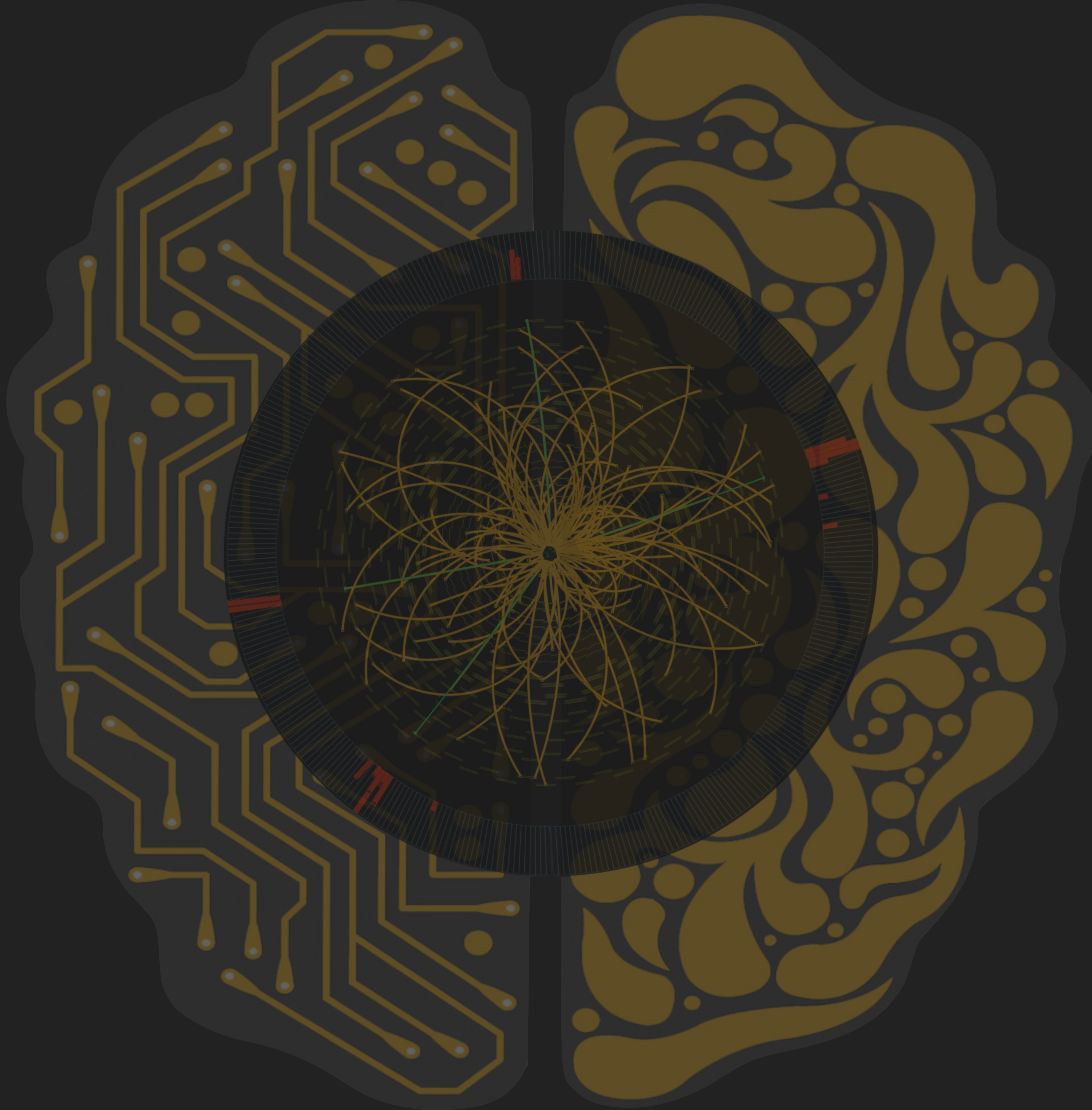
CHALLENGE: BIG DATA



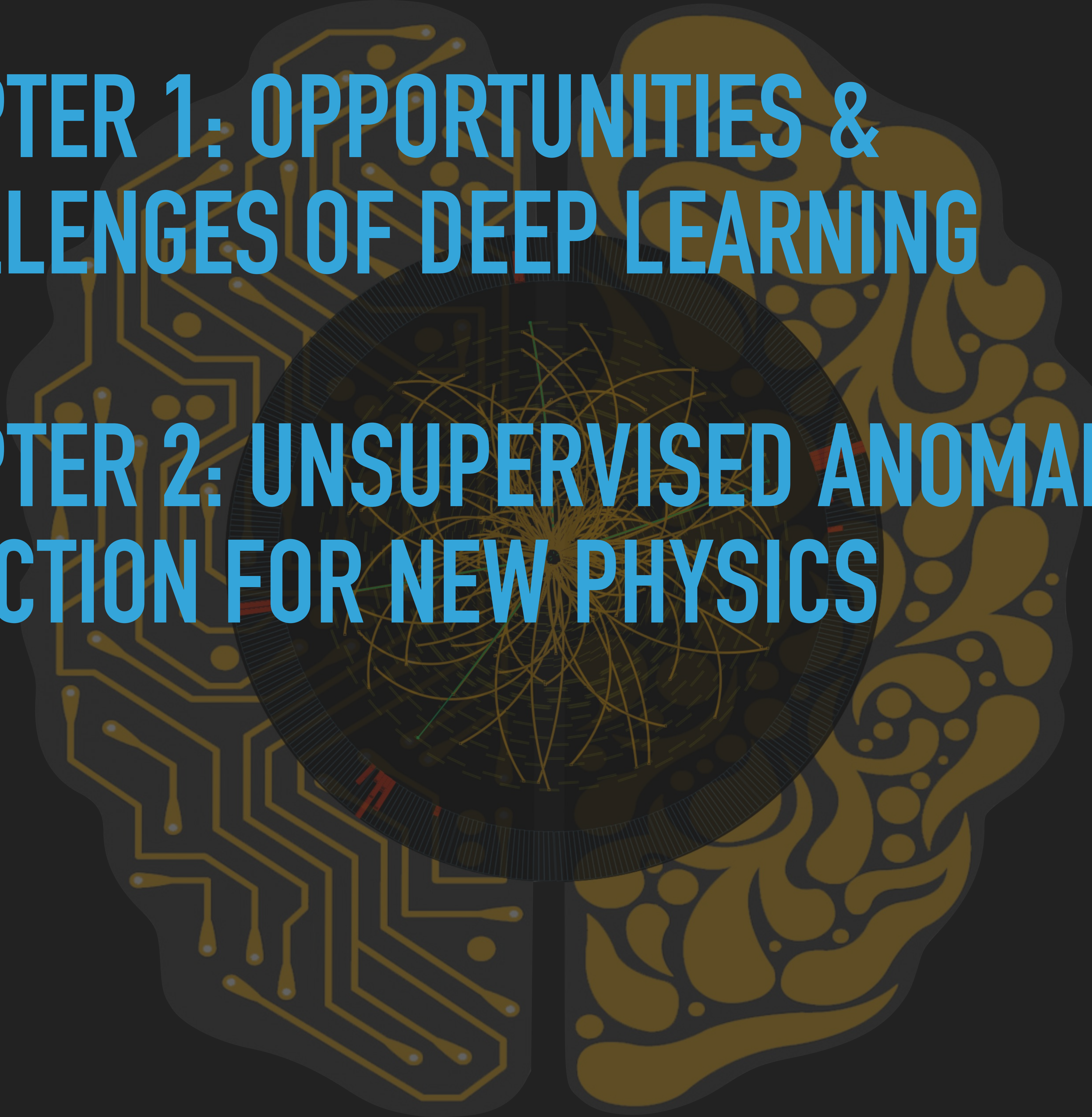
CHALLENGE: BIG DATA

- HL-LHC will reach 1 exabyte of data per year



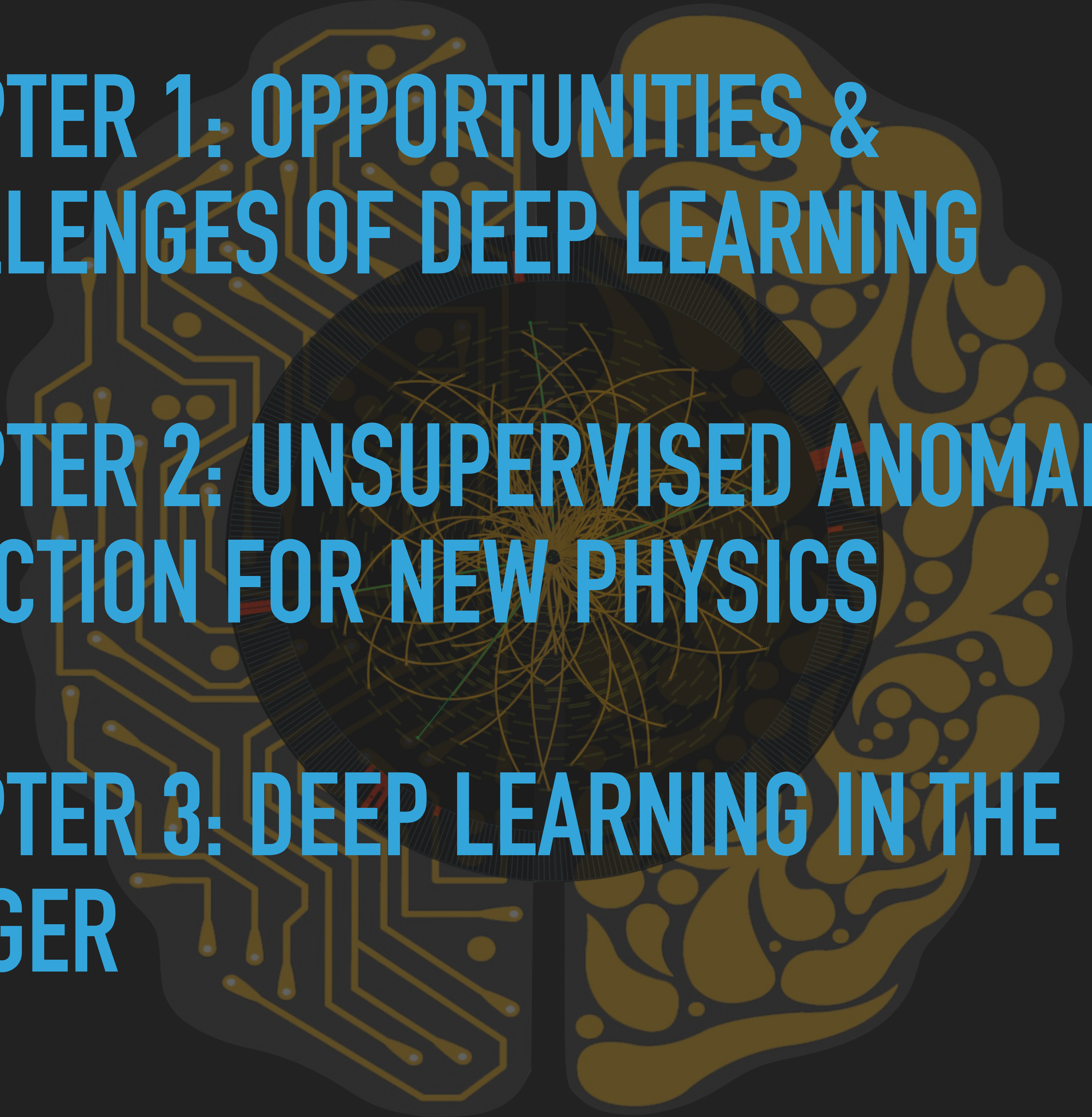


CHAPTER 1: OPPORTUNITIES & CHALLENGES OF DEEP LEARNING



CHAPTER 1: OPPORTUNITIES & CHALLENGES OF DEEP LEARNING

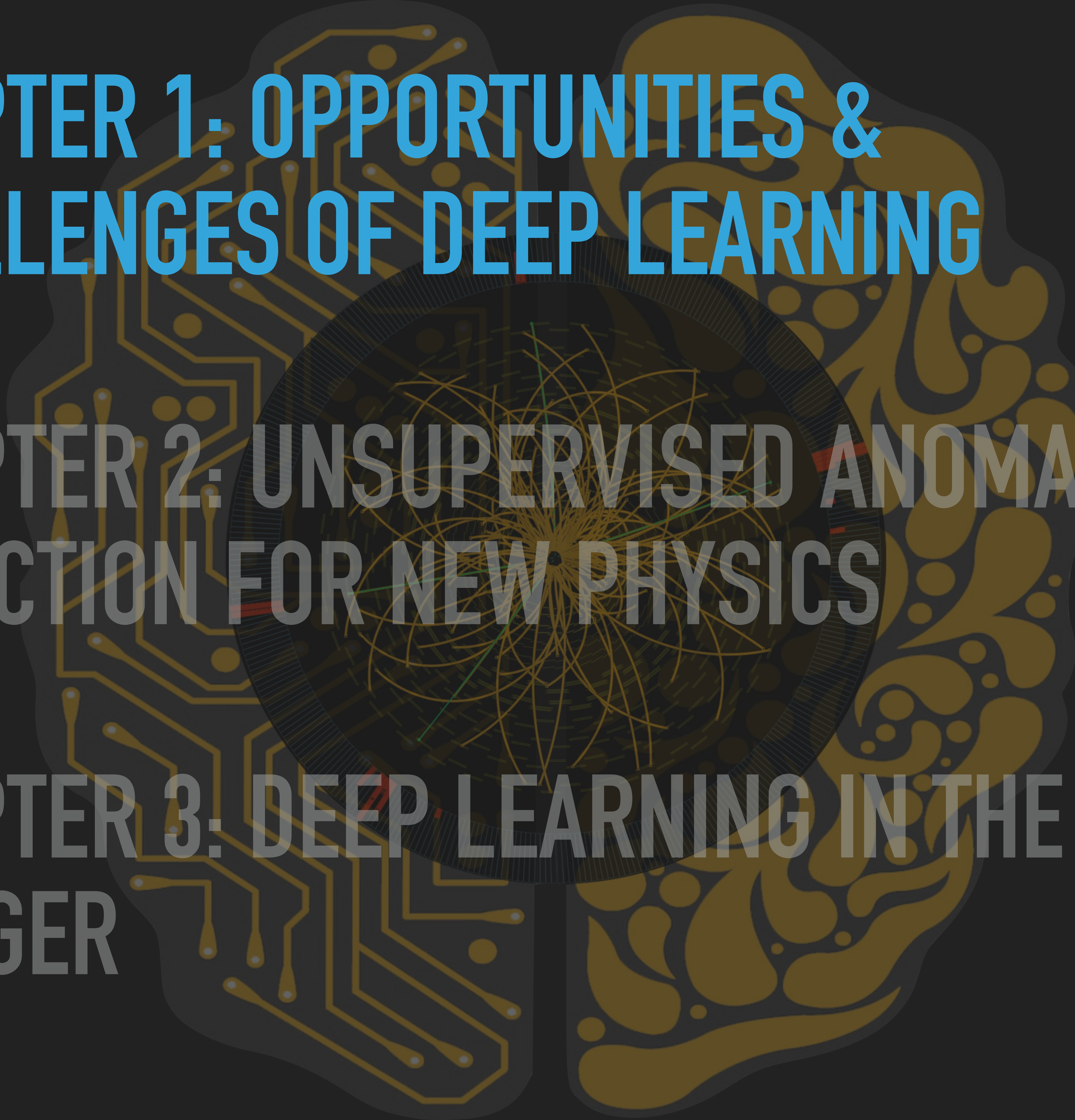
CHAPTER 2: UNSUPERVISED ANOMALY DETECTION FOR NEW PHYSICS



CHAPTER 1: OPPORTUNITIES & CHALLENGES OF DEEP LEARNING

CHAPTER 2: UNSUPERVISED ANOMALY DETECTION FOR NEW PHYSICS

CHAPTER 3: DEEP LEARNING IN THE TRIGGER



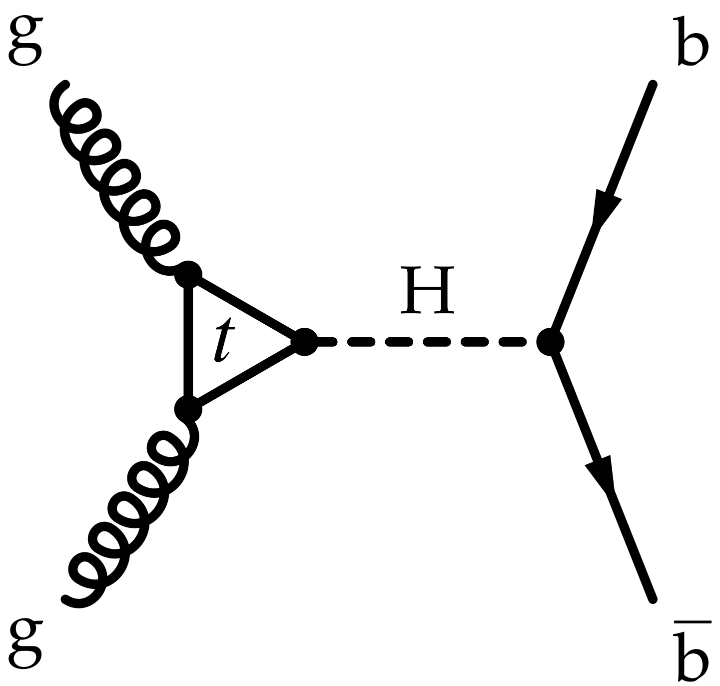
CHAPTER 1: OPPORTUNITIES & CHALLENGES OF DEEP LEARNING

CHAPTER 2: UNSUPERVISED ANOMALY DETECTION FOR NEW PHYSICS

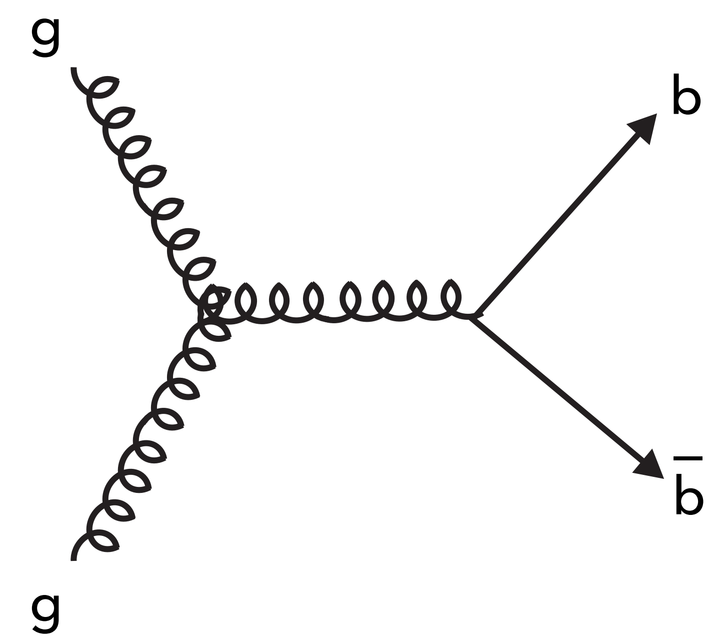
CHAPTER 3: DEEP LEARNING IN THE TRIGGER

signal

gg→H→bb



AN OVERWHELMING BACKGROUND

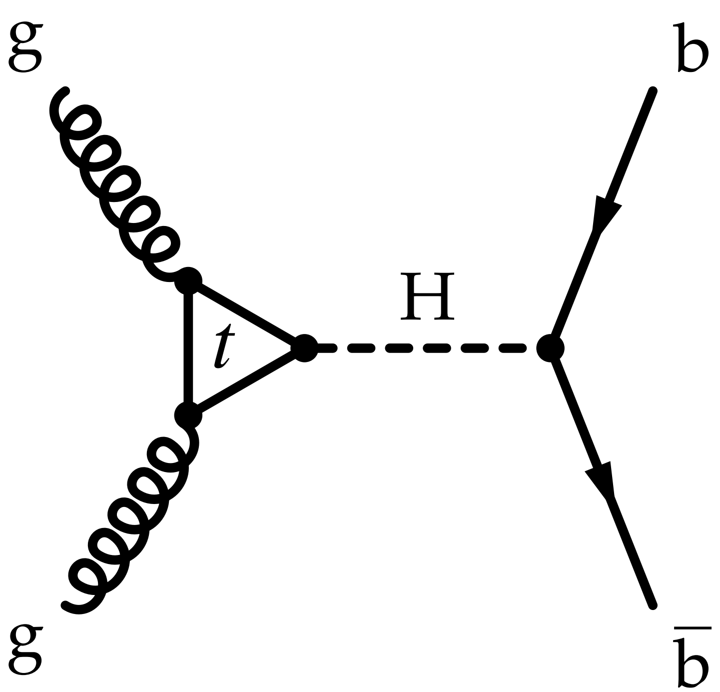


background

gg→bb

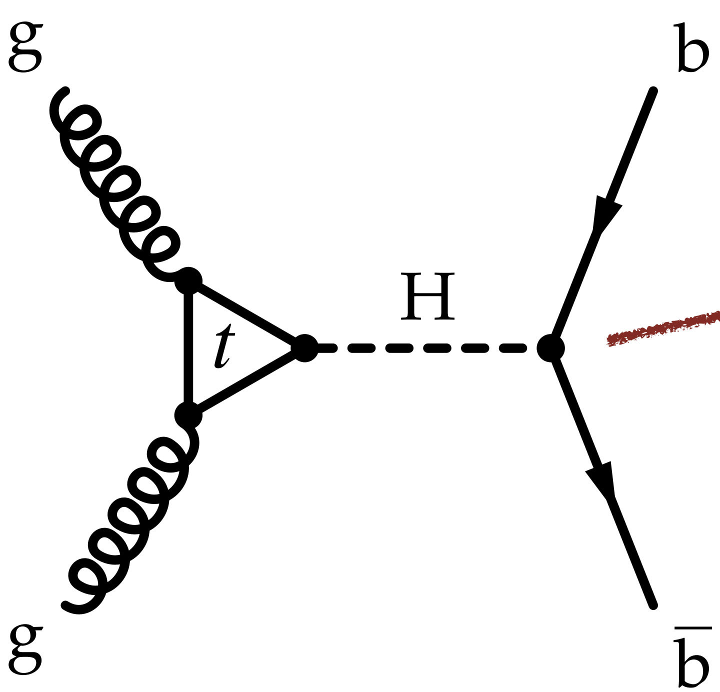
signal

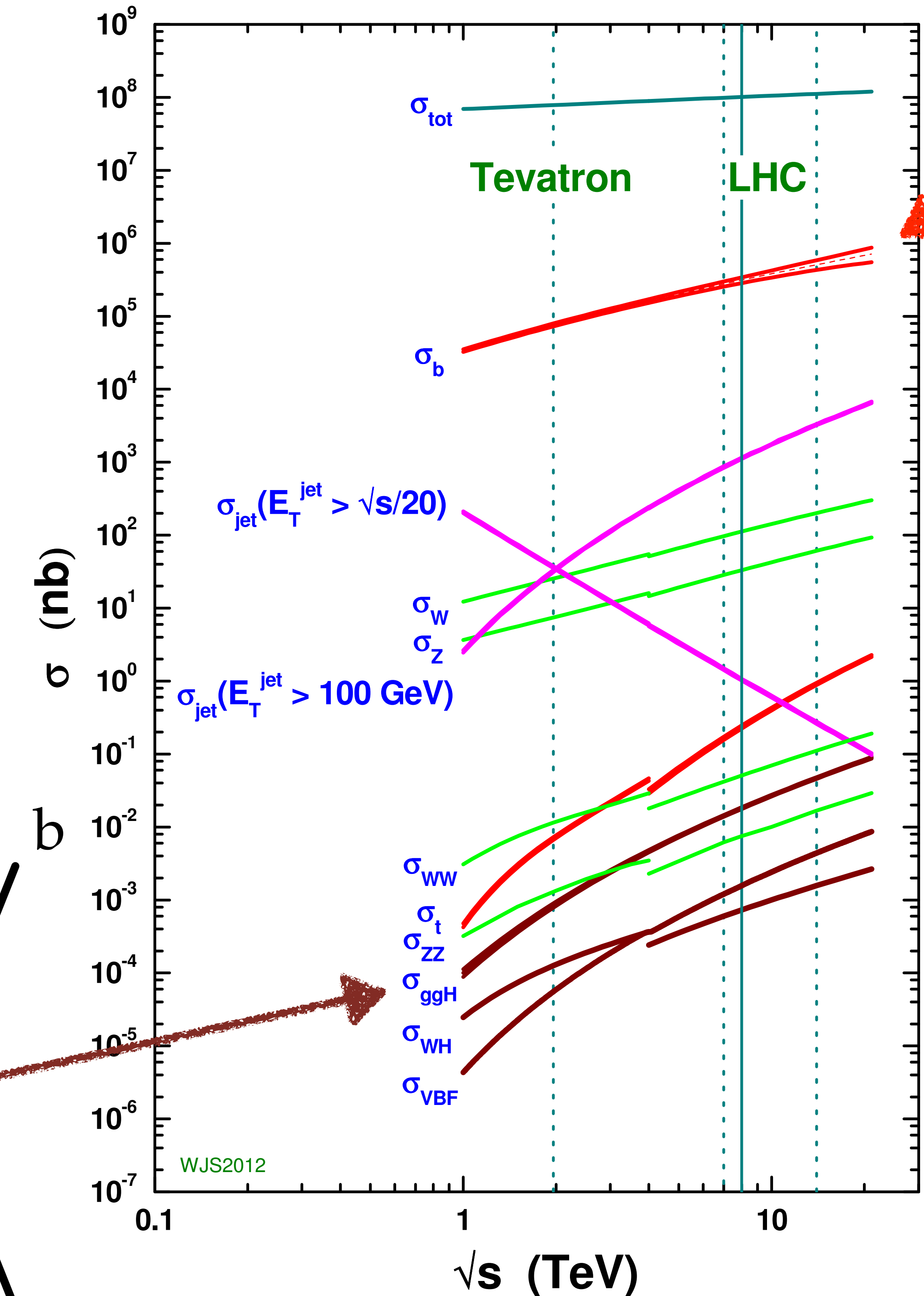
gg→H→bb

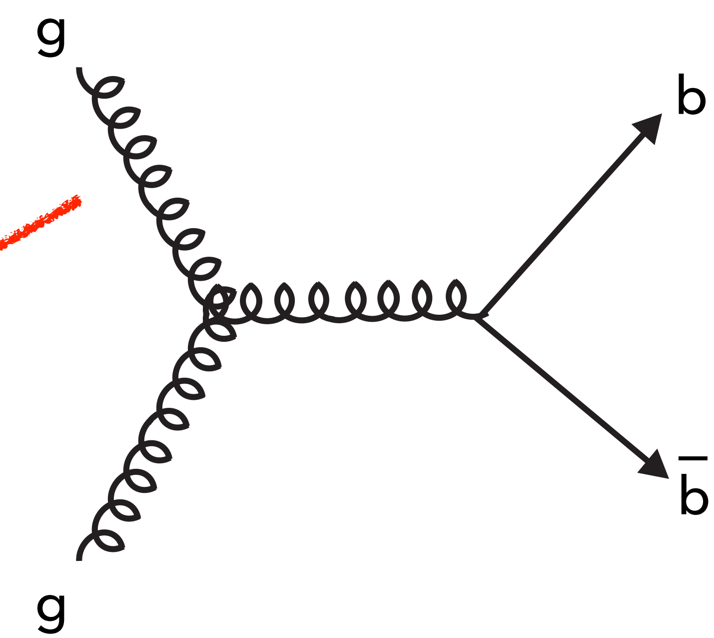


AN OVERWHELMING BACKGROUND

signal
gg→H→bb





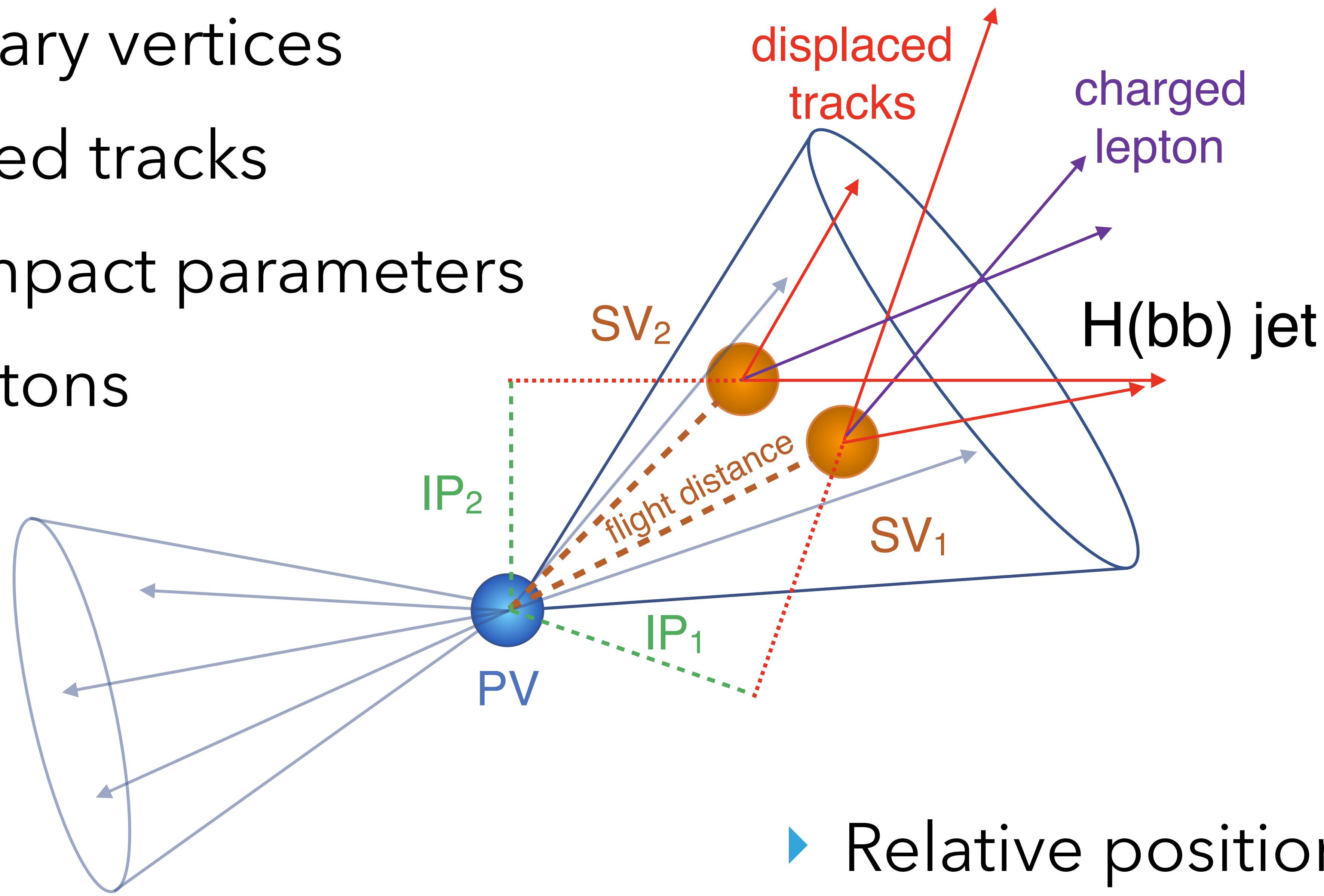


background
gg→bb

BASICS OF DOUBLE-B TAGGING (RECAP)

b hadrons have long lifetimes:
travel O(mm) before decay!

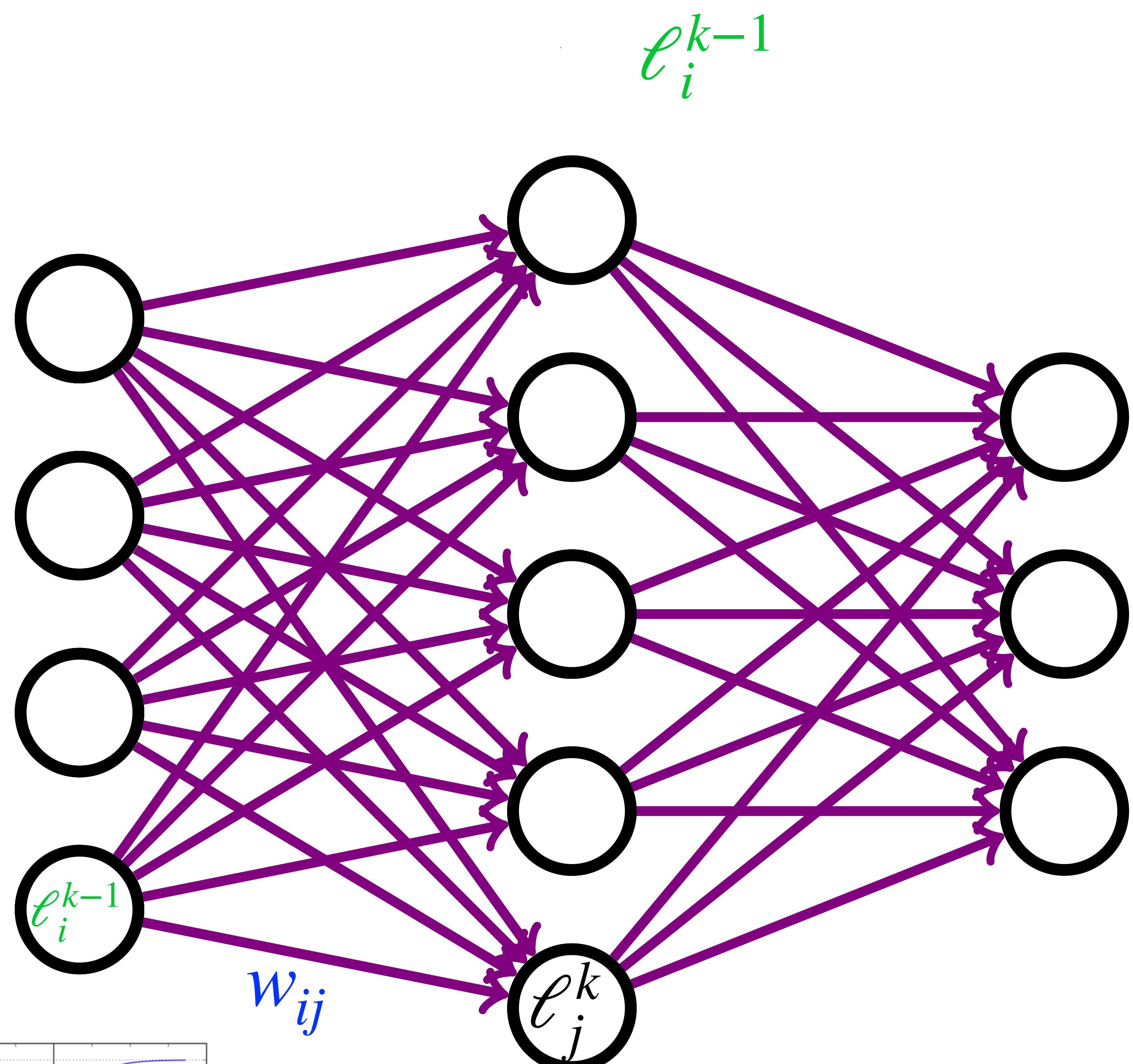
- Handles:
  - secondary vertices
  - displaced tracks
  - large impact parameters
  - soft leptons

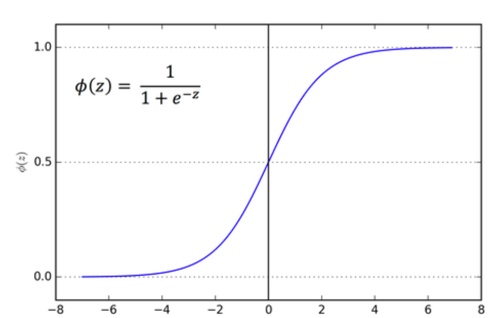


- Relative positions of SVs

NEURAL NETWORK (RECAP)

- Classic fully connected architecture
- Each **input** multiplied by a **weight**
- **Weighted** values are summed, **bias** is added
- Nonlinear **activation function** is applied
- Trained by varying the **parameters** to minimize a loss function (quantifies how many mistakes the network makes)





A sufficiently "wide" neural network can approximate any function!

- **Step 0**: Define the problem (choice of loss function)

- **Step 0**: Define the problem (choice of loss function)

$$L = -y \log(p) + (1-y)\log(1-p)$$

y = 0 (background) or 1 (signal)

p = output of our NN (probability of signal)

- **Step 0**: Define the problem (choice of loss function)

$$L = -y\log(p) + (1-y)\log(1-p)$$

y = 0 (background) or 1 (signal)

p = output of our NN (probability of signal)

if $p \sim y$, $L \sim 0$ (correct!)

if $p \sim 1-y$, $L \sim \infty$ (incorrect!)

- **Step 0**: Define the problem (choice of loss function)

$$L = -y\log(p) + (1-y)\log(1-p)$$

y = 0 (background) or 1 (signal)
p = output of our NN (probability of signal)

if $p \sim y$, $L \sim 0$ (correct!)
if $p \sim 1-y$, $L \sim \infty$ (incorrect!)

- Step 1: Acquire lots of labeled data and split into training and testing sets

- **Step 0**: Define the problem (choice of loss function)

$$L = -y\log(p) + (1-y)\log(1-p)$$

y = 0 (background) or 1 (signal)
p = output of our NN (probability of signal)

if $p \sim y$, $L \sim 0$ (correct!)
if $p \sim 1-y$, $L \sim \infty$ (incorrect!)

- Step 1: Acquire lots of labeled data and split into training and testing sets
- Step 2: Select input features

- **Step 0**: Define the problem (choice of loss function)

$$L = -y\log(p) + (1-y)\log(1-p)$$

y = 0 (background) or 1 (signal)
p = output of our NN (probability of signal)

if $p \sim y$, $L \sim 0$ (correct!)
if $p \sim 1-y$, $L \sim \infty$ (incorrect!)

- Step 1: Acquire lots of labeled data and split into training and testing sets
- Step 2: Select input features
- Step 3: Explore/train different neural network architectures

- **Step 0**: Define the problem (choice of loss function)

$$L = -y\log(p) + (1-y)\log(1-p)$$

y = 0 (background) or 1 (signal)
p = output of our NN (probability of signal)

if $p \sim y$, $L \sim 0$ (correct!)
if $p \sim 1\text{-}y$, $L \sim \infty$ (incorrect!)

- Step 1: Acquire lots of labeled data and split into training and testing sets
- Step 2: Select input features
- Step 3: Explore/train different neural network architectures
- **Step 4**: Evaluate performance

CMS OPEN H(BB) DATASET

opendata CERN

Search

About

Sample with jet, track and secondary vertex properties for Hbb tagging ML studies HiggsToBBNTuple_HiggsToBB_QCD_RunII_13TeV_MC

Duarte, Javier;

Dataset Derived Datascience CMS CERN-LHC

Parent Dataset: /BulkGravTohhTohbbhbb_narrow_M-600_13TeV-madgraph/RunIISummer16MiniAODv2-PUMoriond17_80X_mcRun2_asymptotic_2016_TrancheIV_v6_ext1-v1/MINIAODSIM

Description

The dataset consists of particle jets extracted from simulated proton-proton collision events at a center-of-mass energy of 13 TeV generated with Pythia 8. It has been produced for developing machine-learning algorithms to differentiate jets originating from a Higgs boson decaying to a bottom quark-antiquark pair (Hbb) from quark or gluon jets originating from quantum chromodynamic (QCD) multijet production.

The reconstructed jets are clustered using the anti-kT algorithm with R=0.8 from particle flow (PF) candidates (AK8 jets). The standard L1+L2+L3+residual jet energy corrections are applied to the jets and pileup contamination is mitigated using the charged hadron subtraction (CHS) algorithm. Features of the AK8 jets with transverse momentum pT > 200 GeV and pseudorapidity |η| < 2.4 are provided. Selected features of inclusive (both charged and neutral) PF candidates with pT > 0.95 GeV associated to the AK8 jet are provided. Additional features of charged PF candidates (formed primarily by a charged particle track) with pT > 0.95 GeV associated to the AK8 jet are also provided. Finally, additional features of reconstructed secondary vertices (SVs) associated to the AK8 jet (within ΔR < 0.8) are also provided.

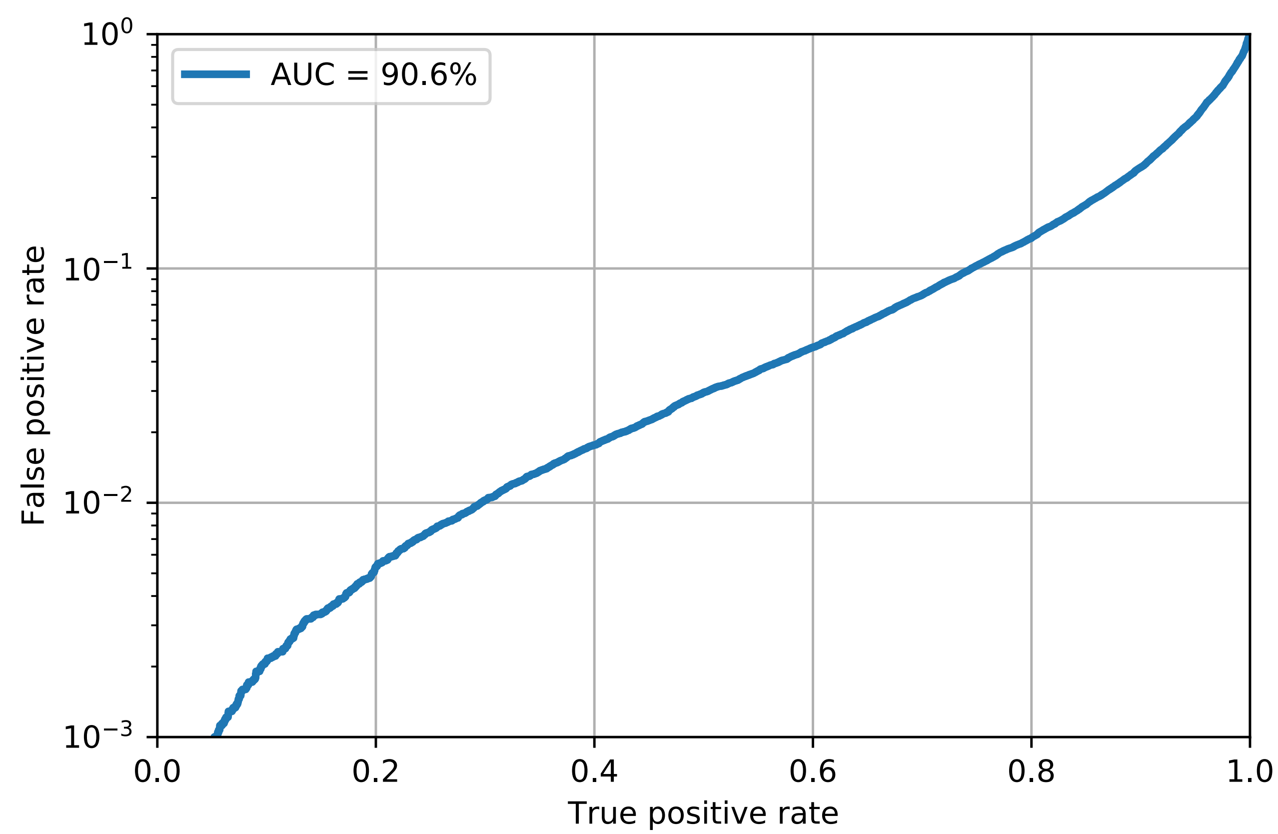
- Derived datasets (ROOT & HDF5): http://opendata-dev.web.cern.ch/record/12102
- 182 files, 245 GB, 18 million total entries (jets)
  - event features, e.g. MET, ρ (average density)
  - jet features, e.g. mass, p_T, N-subjettiness variables
  - particle candidate features, e.g. p_T, η, **ϕ** (*for up to 100 particles*)
  - charged particle / track features, e.g. impact parameter (*for up to 60 tracks*)
  - secondary vertex features, e.g. flight distance (*for up to 5 vertices*)

DEMO: SIMPLE NEURAL NETWORK TRAINING

https://github.com/cernopendata-datascience/HiggsToBBMachineLearning

- Train fully connected neural network with high level features in ~30 lines of code

```
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           (None, 27)                0
_________________________________________________________________
bn_1 (BatchNormalization)    (None, 27)                108
_________________________________________________________________
dense_1 (Dense)              (None, 64)                1792
_________________________________________________________________
dense_2 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 32)                1056
_________________________________________________________________
output (Dense)               (None, 2)                 66
=================================================================
Total params: 5,102
Trainable params: 5,048
Non-trainable params: 54
_________________________________________________________________
```



“DEEP” DOUBLE-B TAGGER

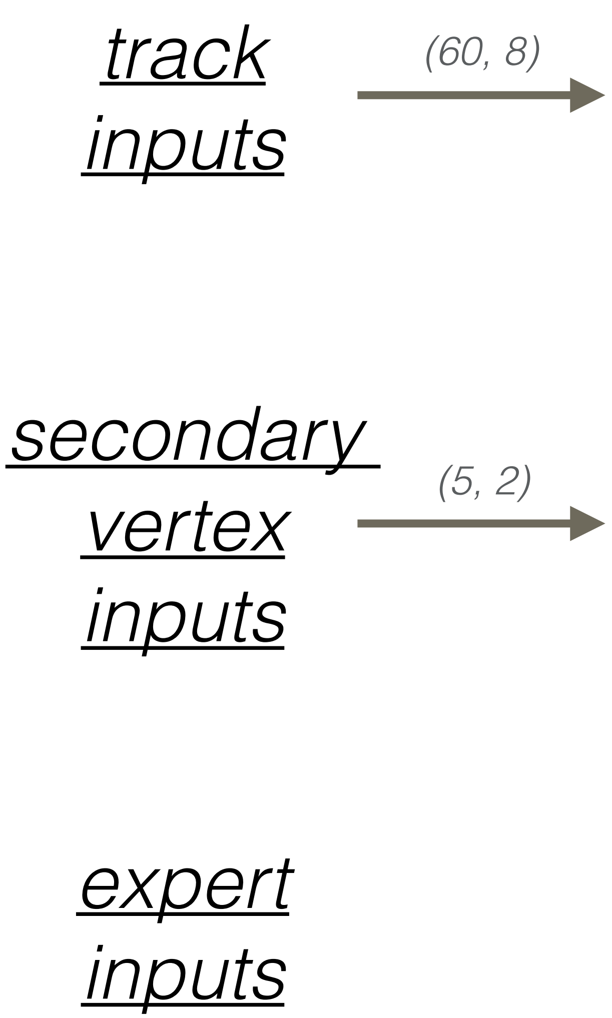
track inputs

secondary vertex inputs

expert inputs

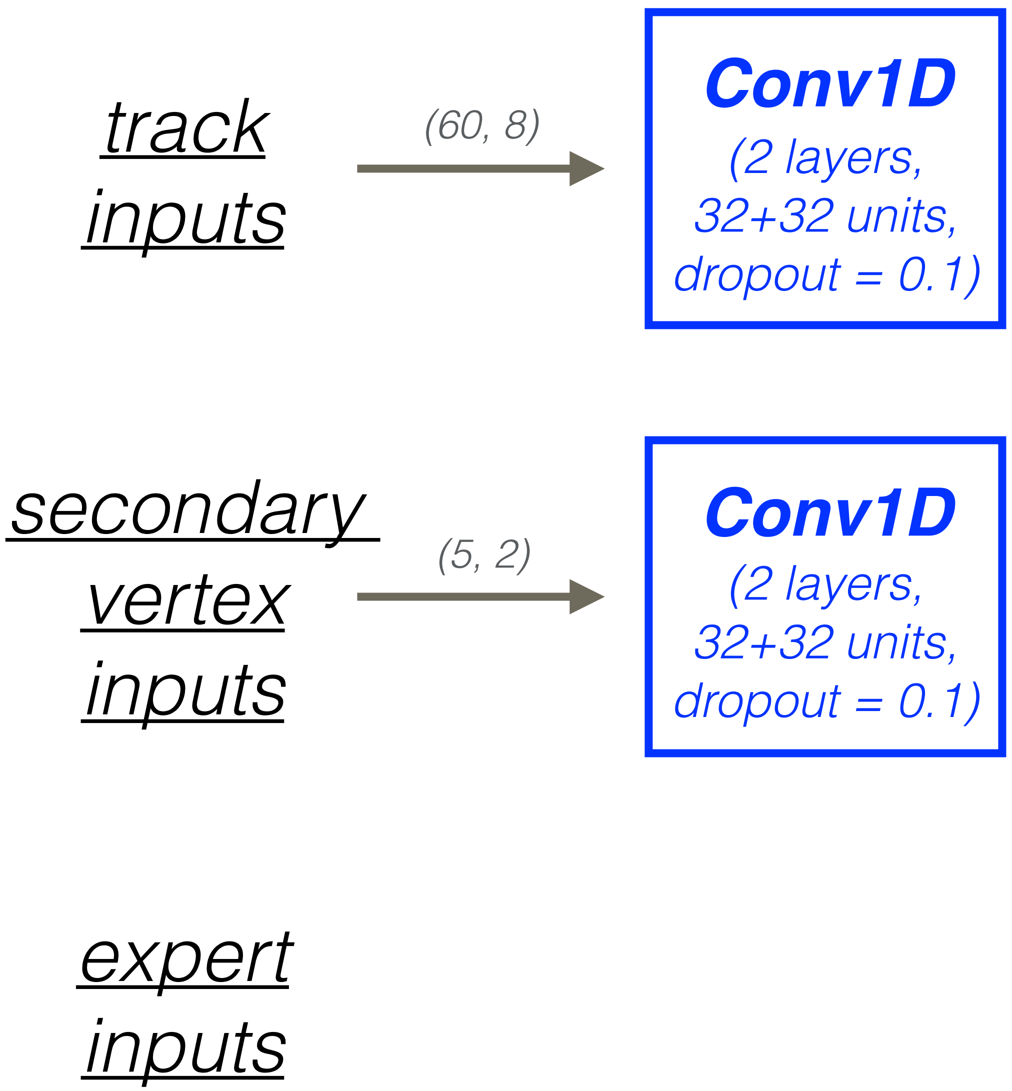
"DEEP" DOUBLE-B TAGGER

- Process low-level track and SV inputs as ***ordered lists***



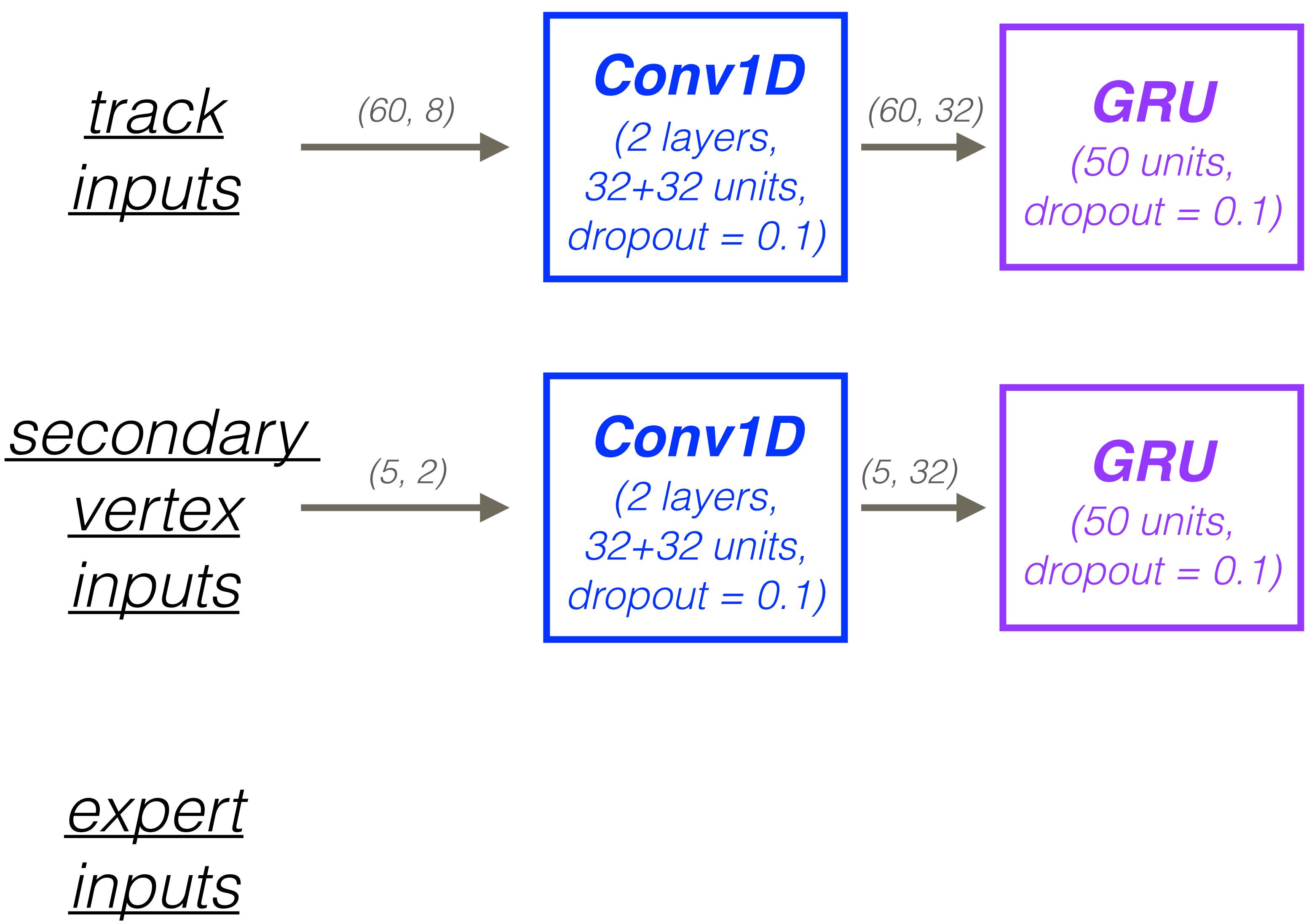
"DEEP" DOUBLE-B TAGGER

- Process low-level track and SV inputs as ***ordered lists***
  - **Convolutional** NN layers: share parameters across inputs, …



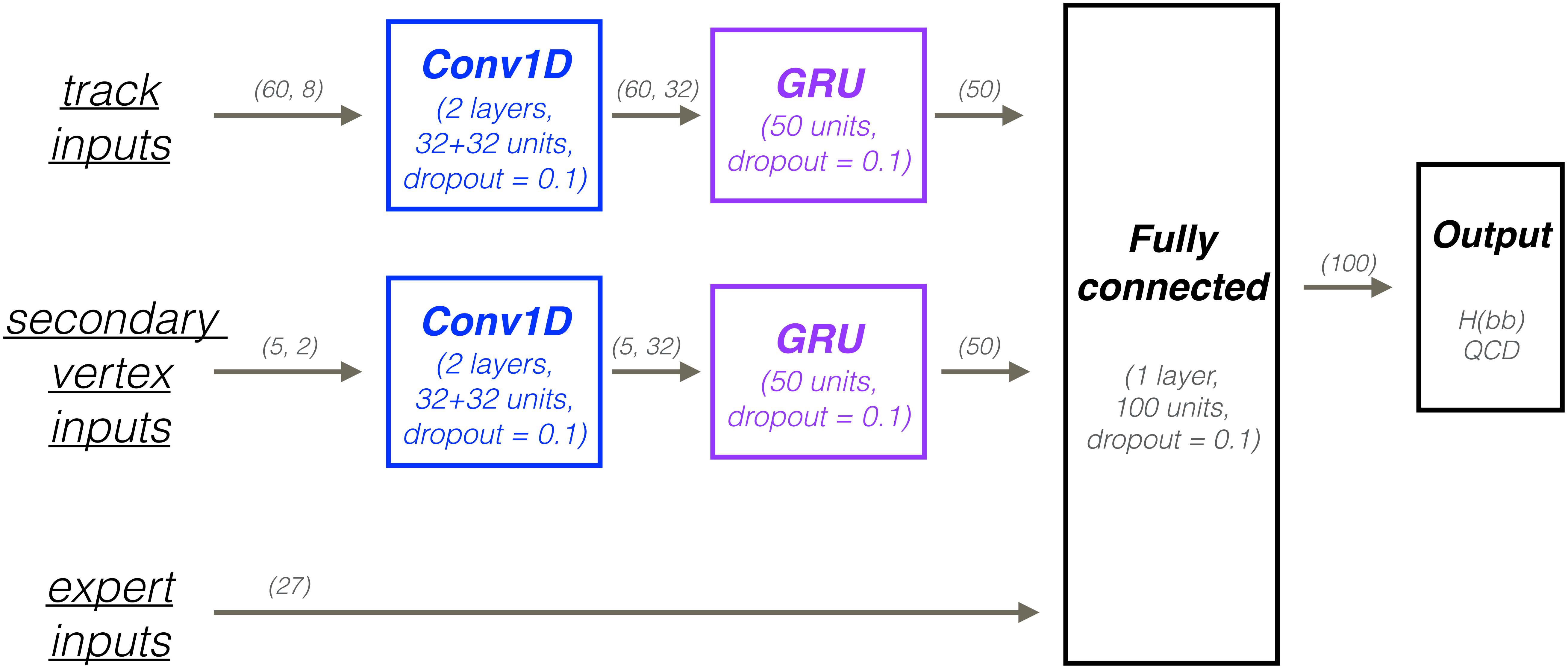
"DEEP" DOUBLE-B TAGGER

- Process low-level track and SV inputs as ***ordered lists***
  - **Convolutional** NN layers: share parameters across inputs, …
  - **Recurrent** NN layers: performs dimensional reduction, …



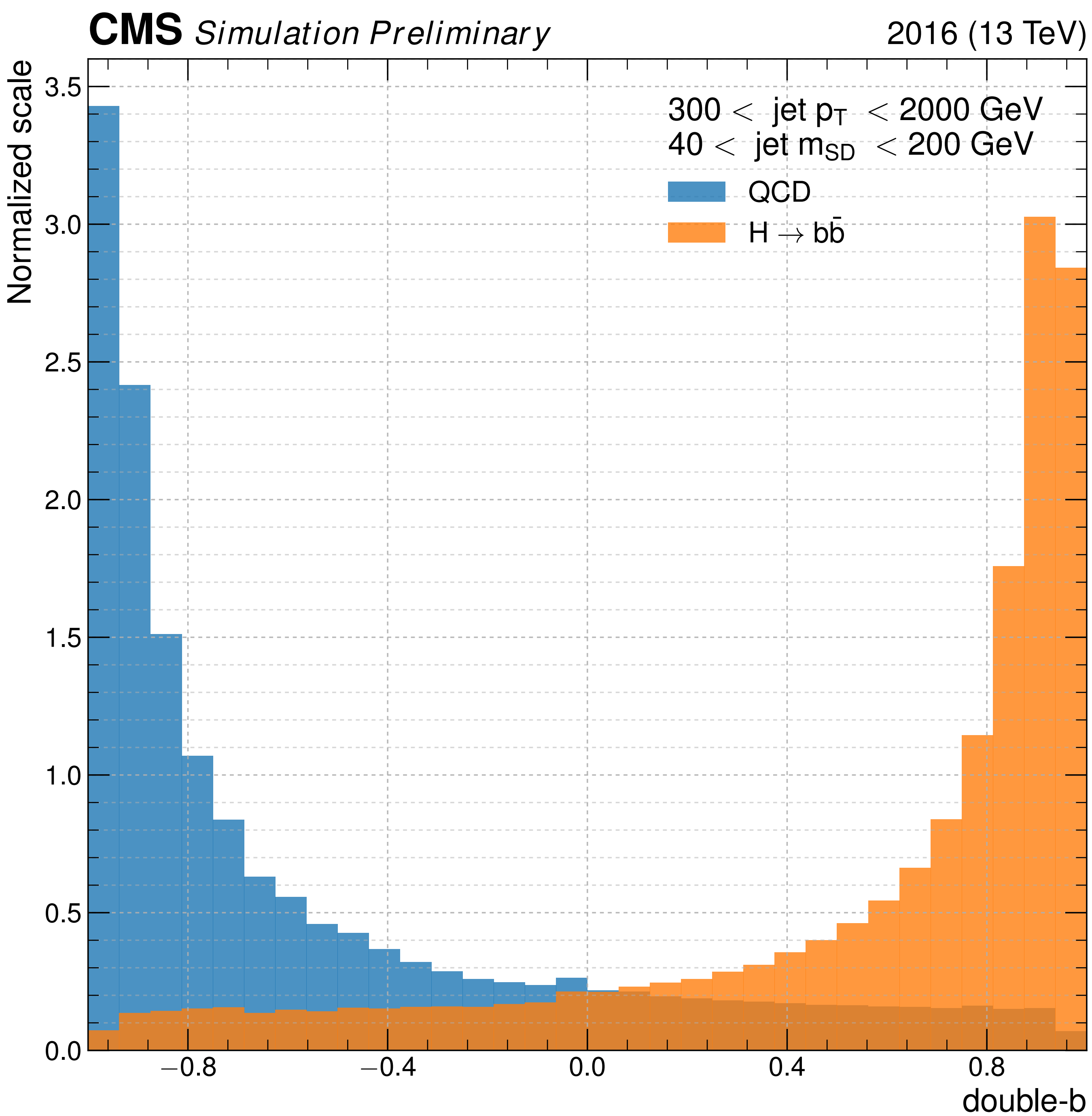
“DEEP” DOUBLE-B TAGGER

- Process low-level track and SV inputs as ***ordered lists***
  - **Convolutional** NN layers: share parameters across inputs, …
  - **Recurrent** NN layers: performs dimensional reduction, …
- Combine in final layer with expert inputs



TAGGING PERFORMANCE

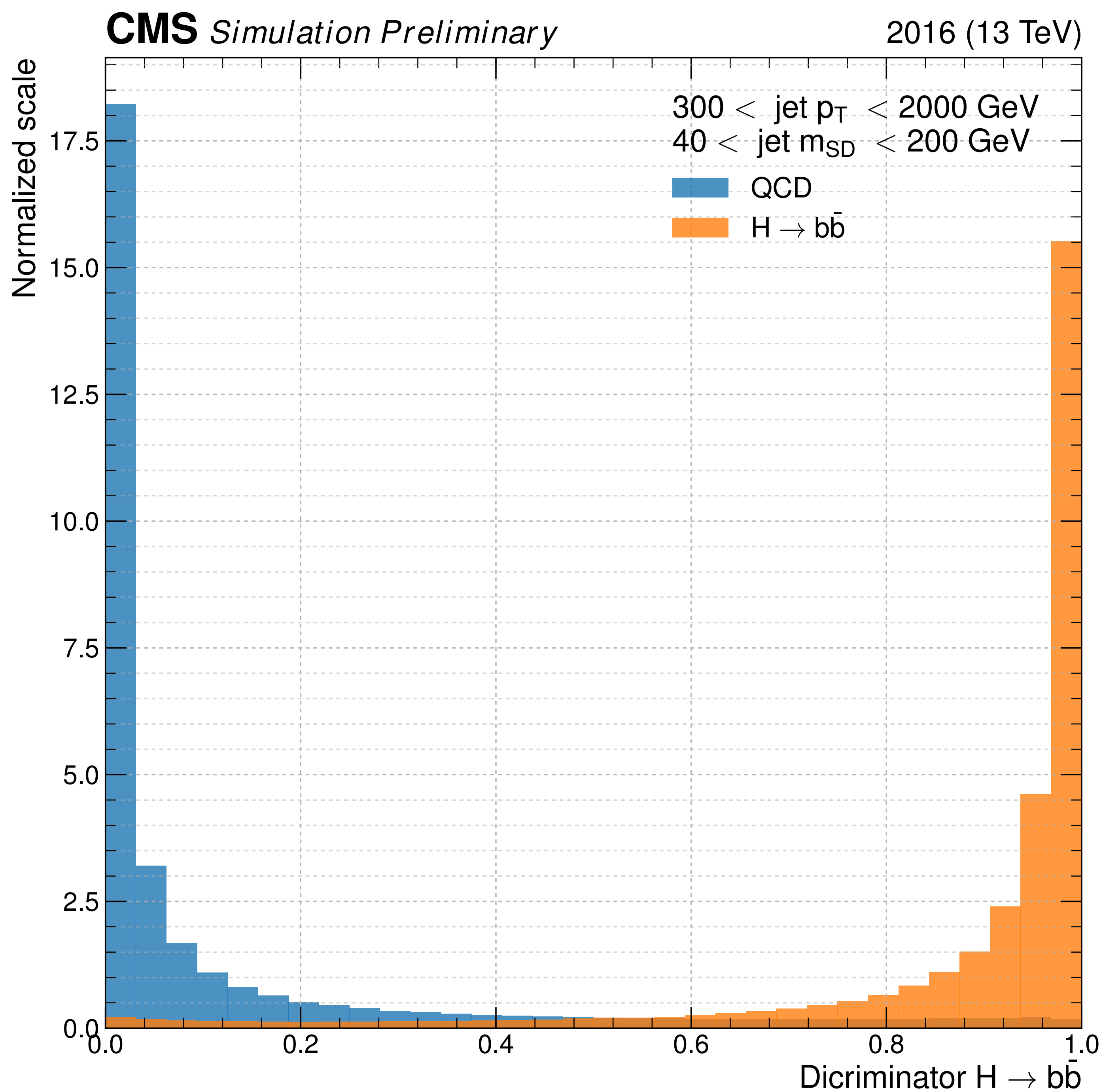
DP-2018/033



TAGGING PERFORMANCE

DP-2018/033

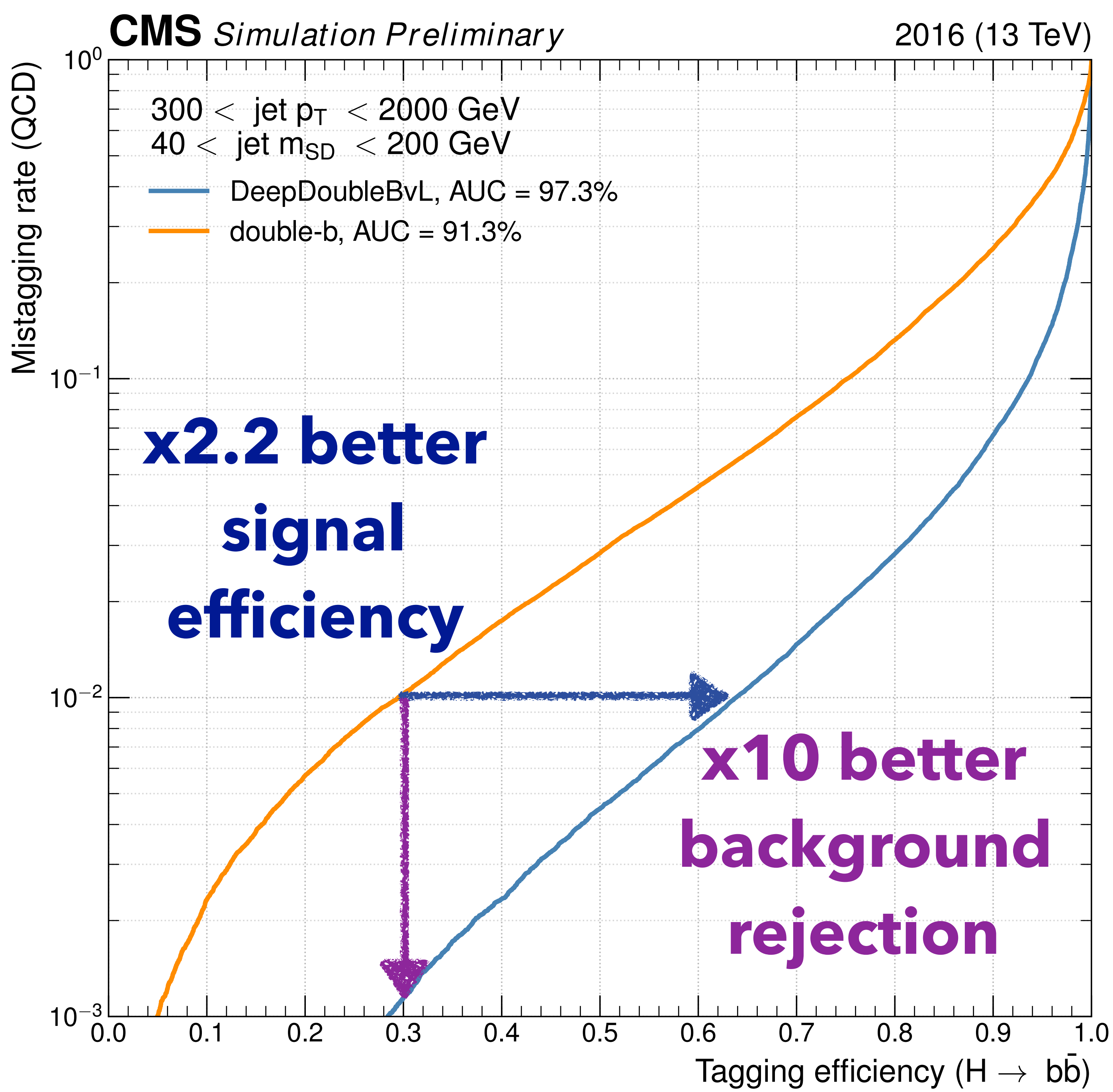
- Success for deep learning!



TAGGING PERFORMANCE

DP-2018/033

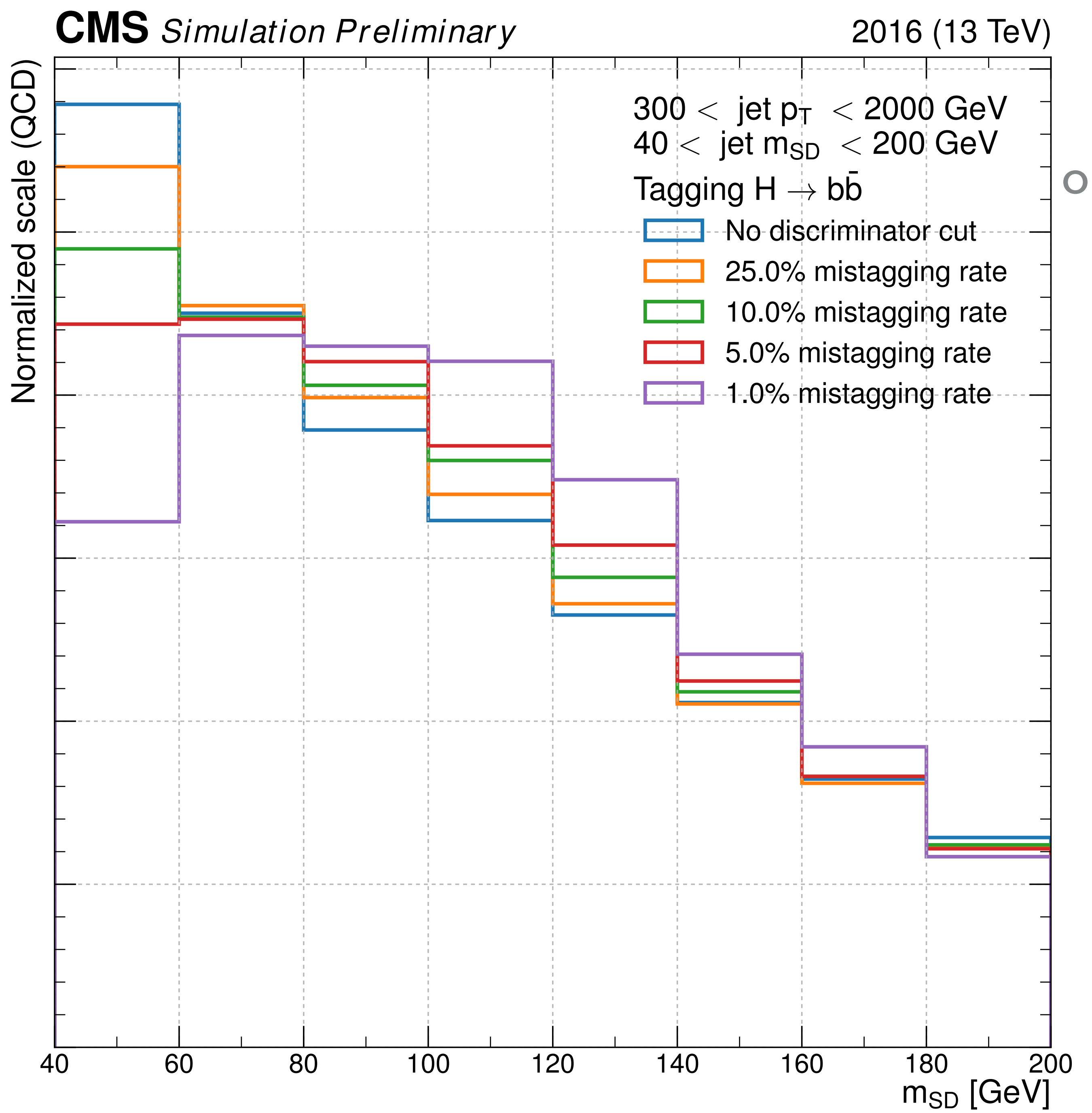
- Success for deep learning!



LEARNING THE JET MASS (SCULPTING)

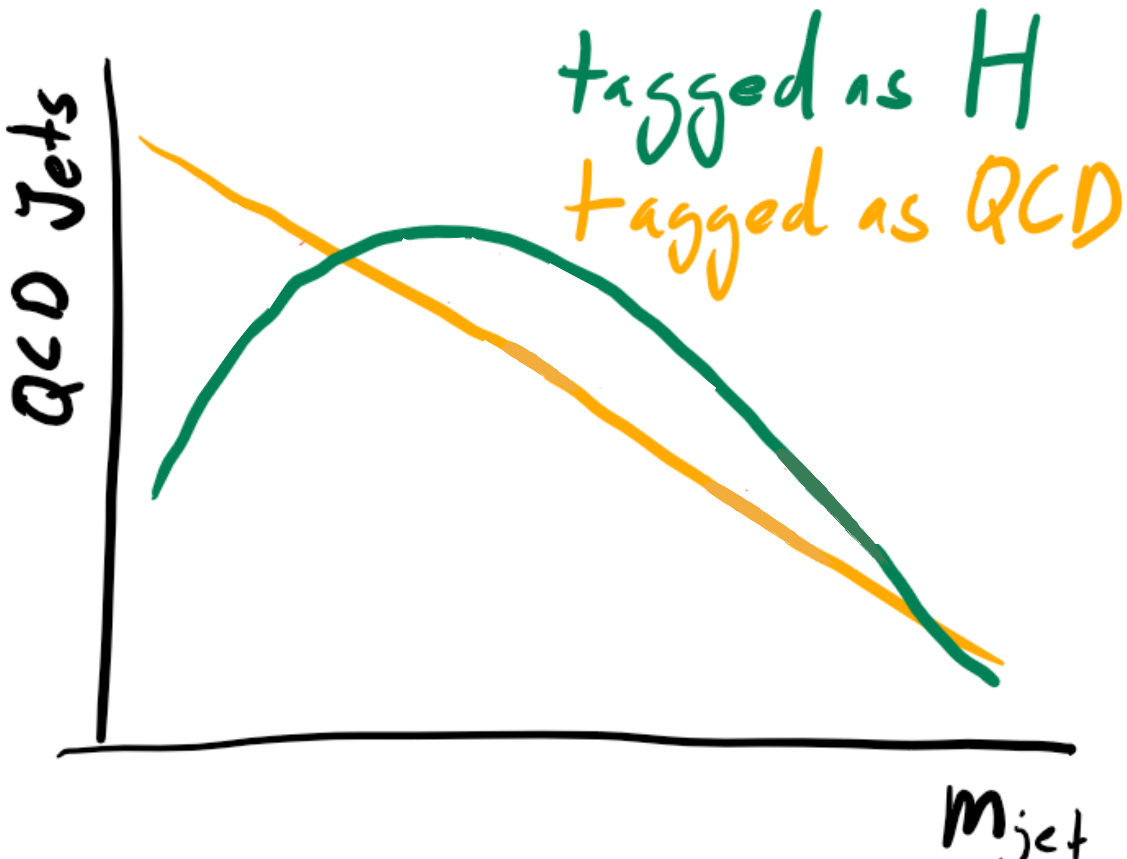
DP-2018/033

- An unintended consequence: network "learns" the jet mass



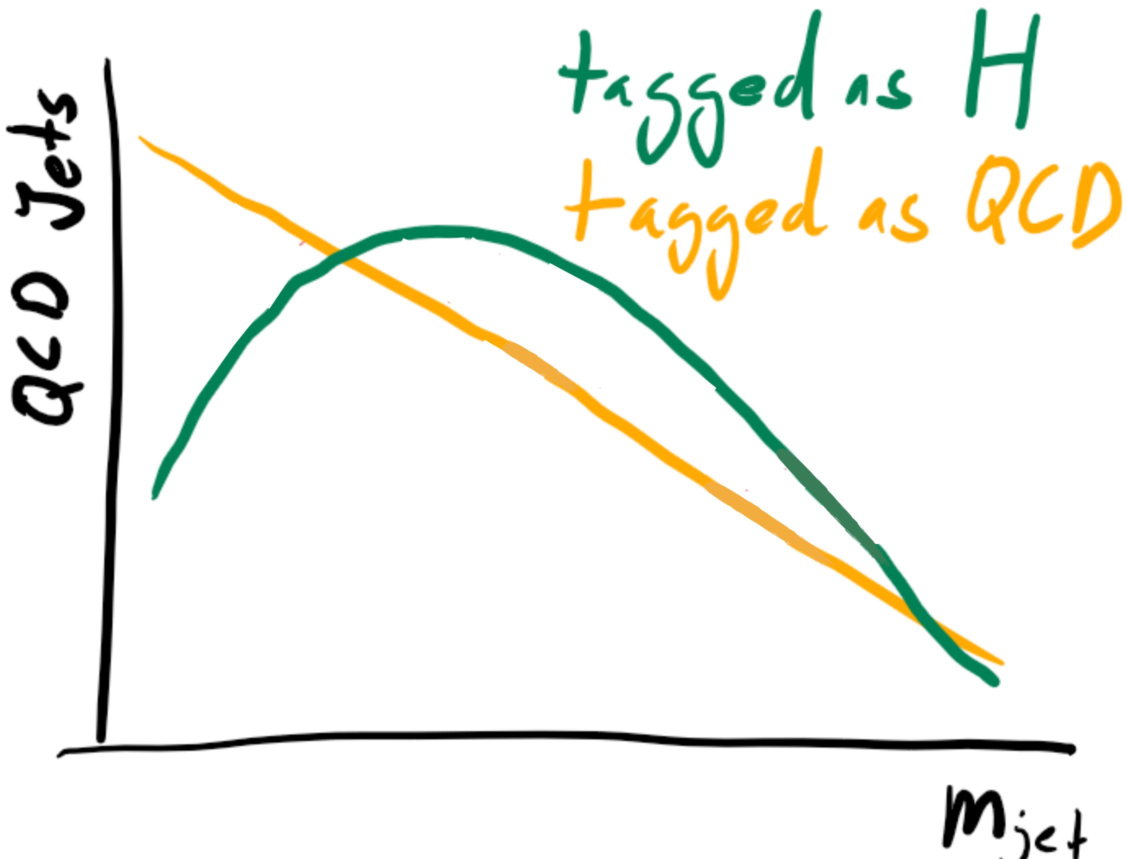
tighter cuts on discriminator

MITIGATING THE MASS SCULPTING



MITIGATING THE MASS SCULPTING

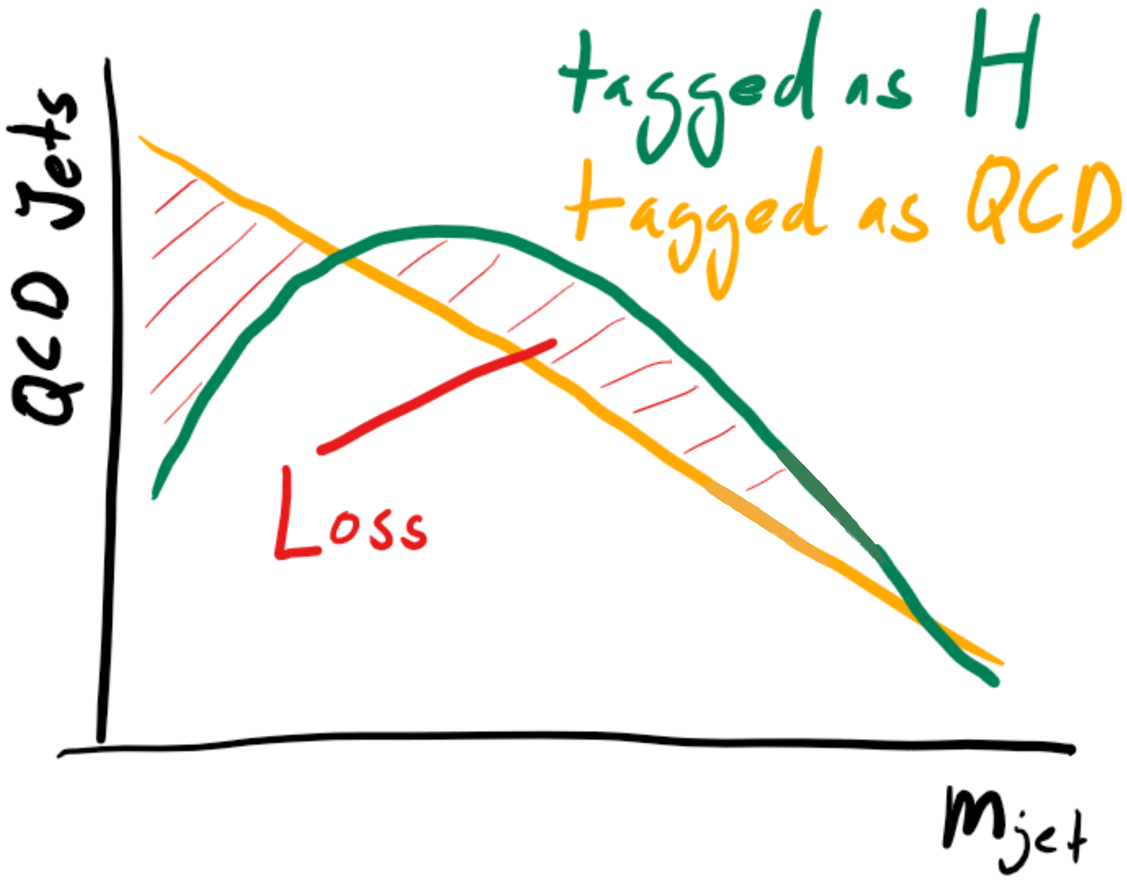
- How can we quantify the mass sculpting?



MITIGATING THE MASS SCULPTING

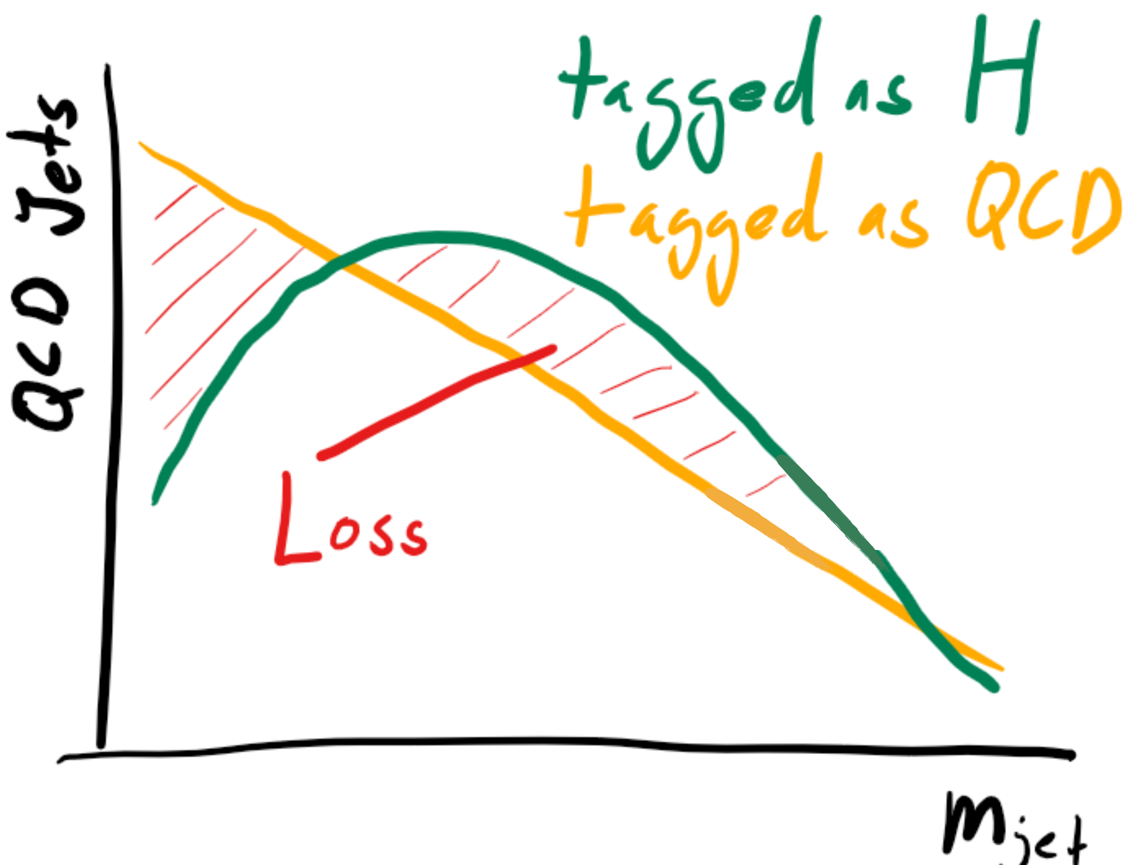
- How can we quantify the mass sculpting?
  - Kullback-Liebler divergence

$$D_{\mathrm{KL}} = h(x)\ \log\left(\frac{h(x)}{q(x)}\right)$$



MITIGATING THE MASS SCULPTING

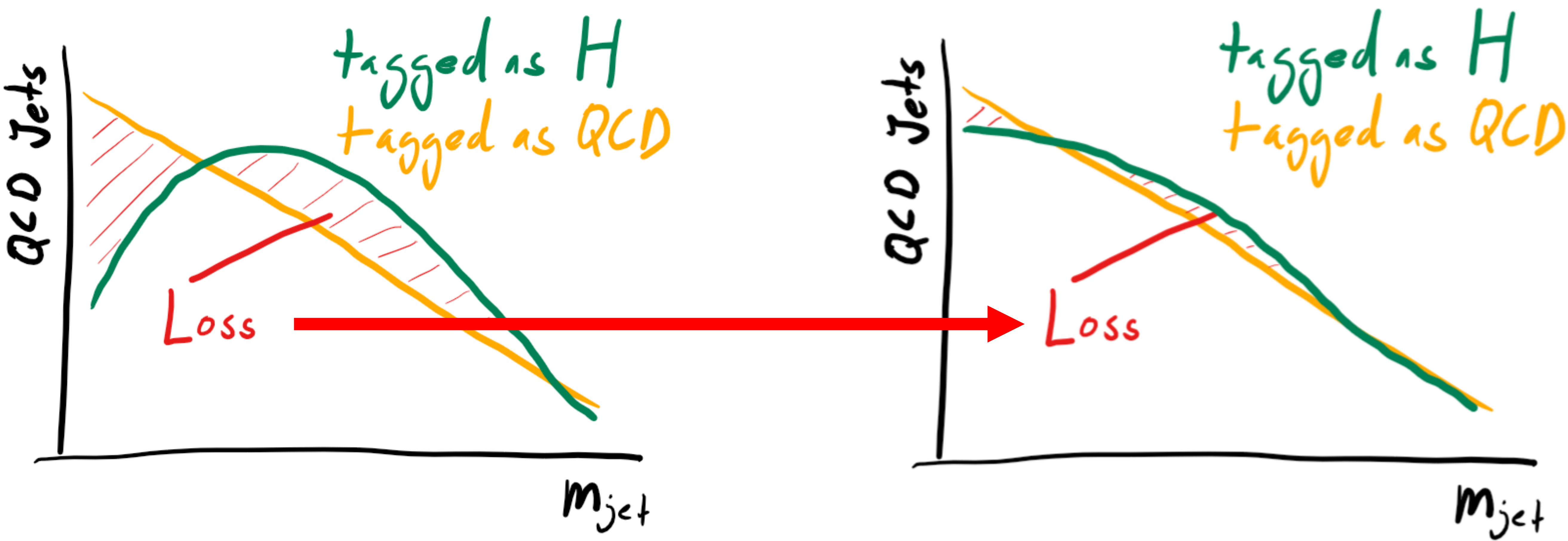
- How can we quantify the mass sculpting?
  - Kullback-Liebler divergence $$D_{\mathrm{KL}} = h(x)\ \log\left(\frac{h(x)}{q(x)}\right)$$
- How can we mitigate the mass sculpting?



MITIGATING THE MASS SCULPTING

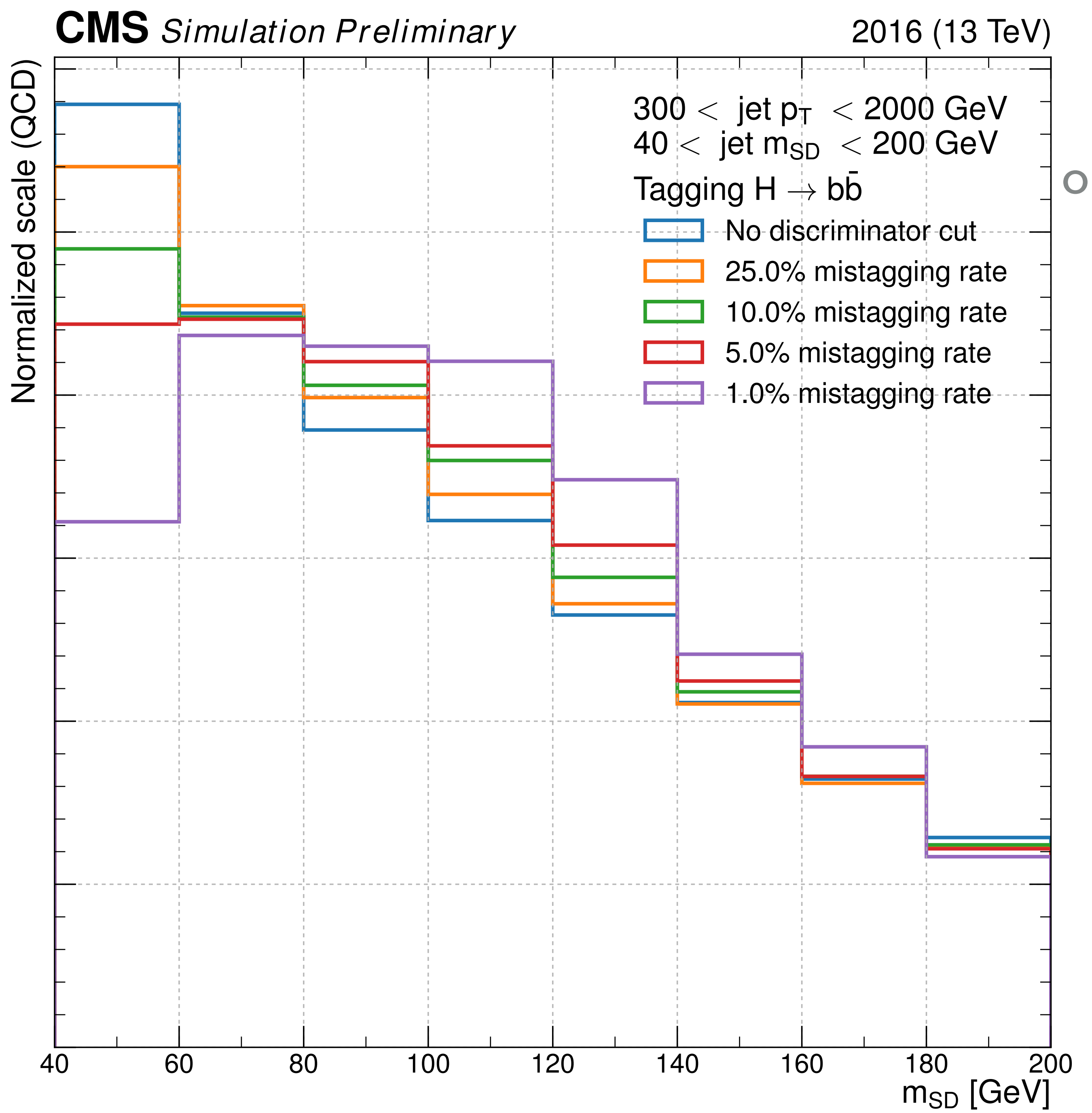
- How can we quantify the mass sculpting?
  - Kullback-Liebler divergence $D_{\mathrm{KL}} = h(x)\ \log\left(\frac{h(x)}{q(x)}\right)$
- How can we mitigate the mass sculpting?
  - Add it to the loss function as a "penalty"

$$L = L_{\mathrm{disc}} + \lambda D_{\mathrm{KL}}$$



MASS SCULPTING MITIGATION

DP-2018/033

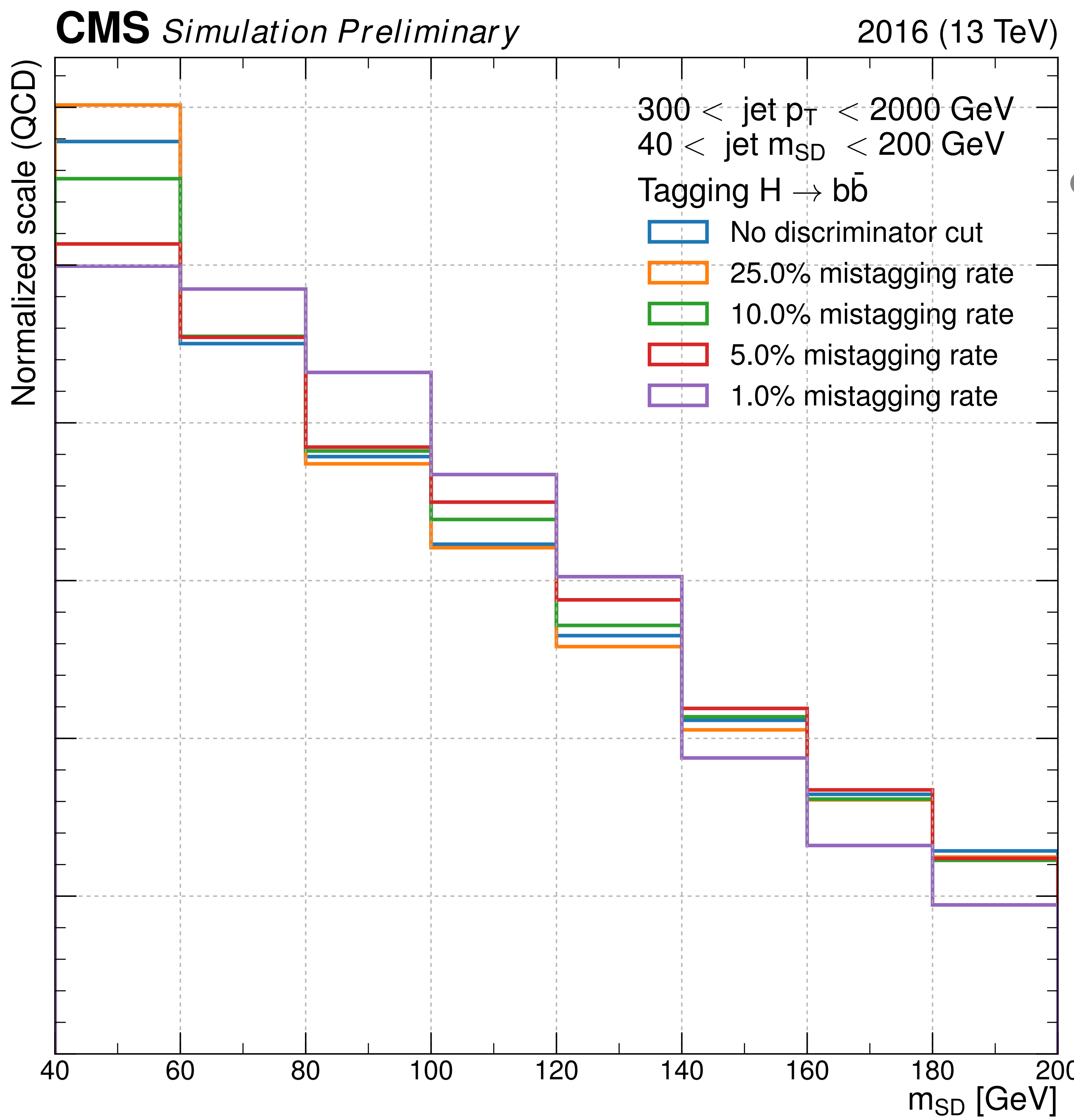


tighter cuts on discriminator

MASS SCULPTING MITIGATION

DP-2018/033

- Penalty term mitigates the mass sculpting

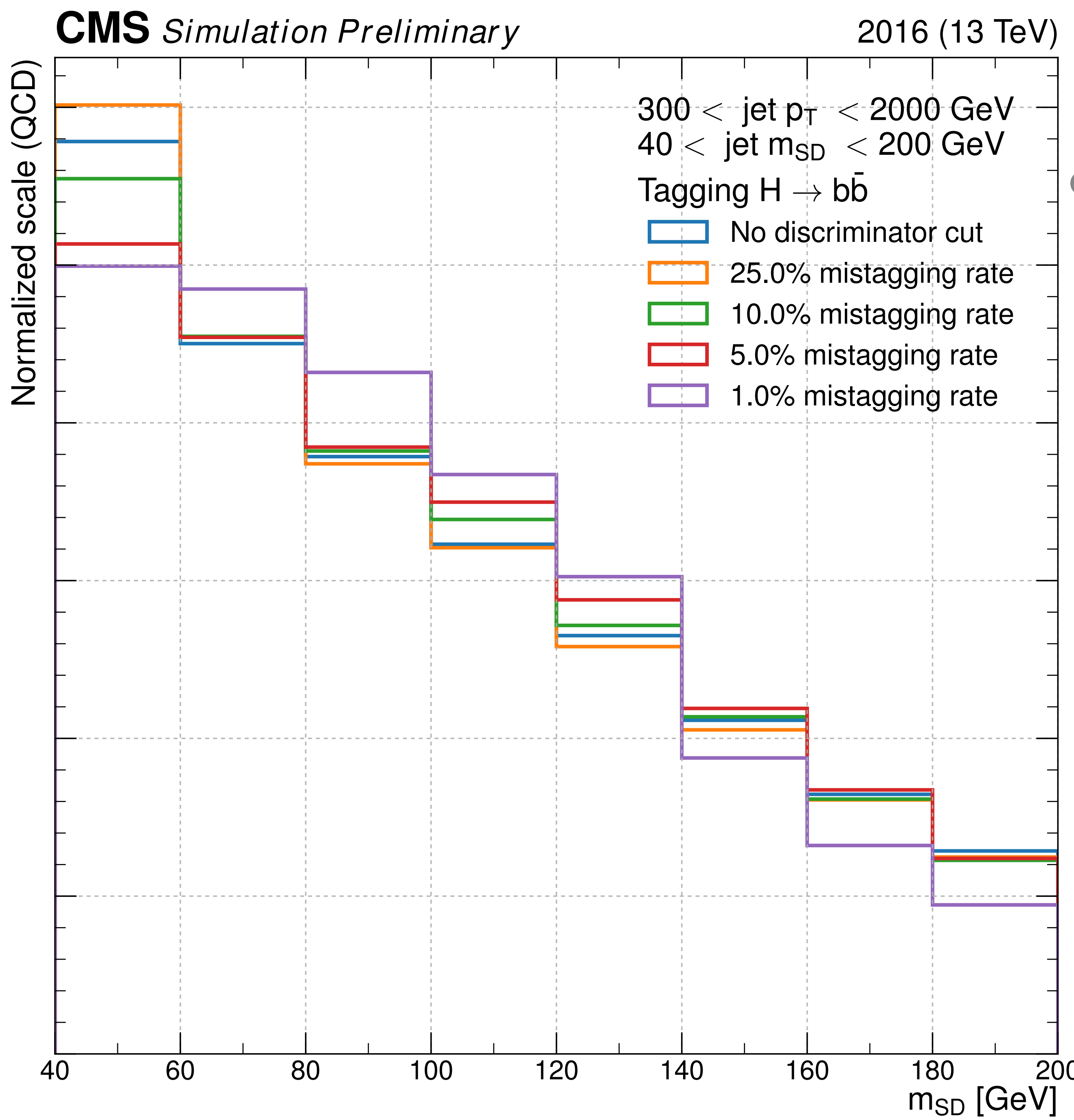


tighter cuts on discriminator

MASS SCULPTING MITIGATION

DP-2018/033

- Penalty term mitigates the mass sculpting
- Small trade-off with performance



tighter cuts on discriminator

CAN WE DO EVEN BETTER?

CAN WE DO EVEN BETTER?

- Ordered lists of particles not the most natural representation of a jet
  - What if we consider each jet as a **graph** of interconnected particles?

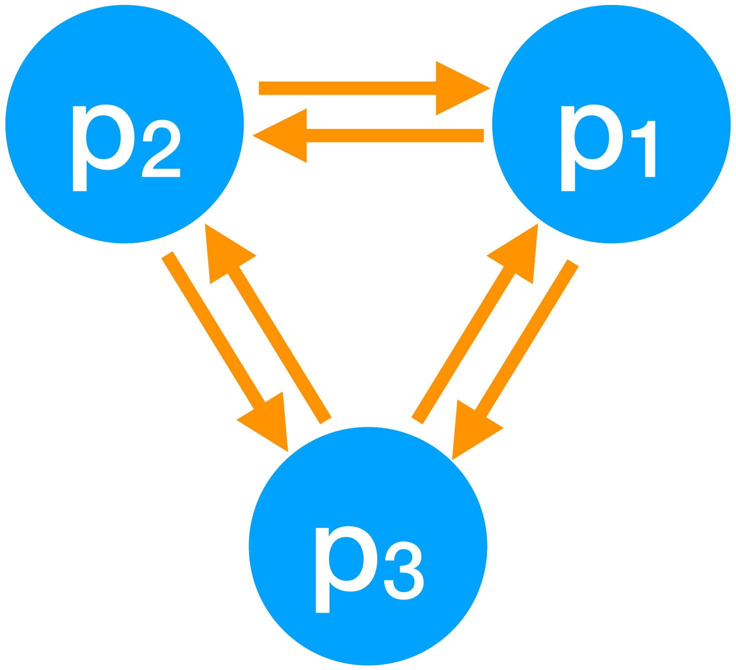
CAN WE DO EVEN BETTER?

- Ordered lists of particles not the most natural representation of a jet
  - What if we consider each jet as a **graph** of interconnected particles?
- **Geometric deep learning** (a.k.a graph neural networks, interaction networks, message-passing neural networks) is the extension of deep learning to deal with data structured as a graph or on a manifold
  - See Interaction Networks for Learning about Objects, Relations, and Physics [**arXiv:1612.0222**], Neural Message Passing for Quantum Chemistry [**arXiv:1704.01212**], Dynamic Graph CNN for Learning on Point Clouds [**arXiv:1801.07829**], Fast Graph Representation Learning with PyTorch Geometric [**arXiv:1903.0242**]

INTERACTION NETWORKS FOR JET TAGGING

arXiv:1908.05318

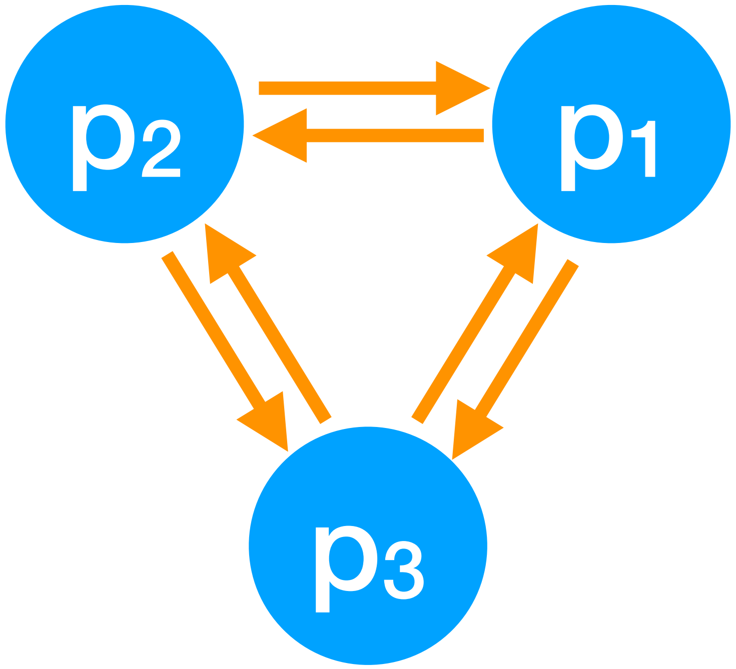
arXiv:1909.12285



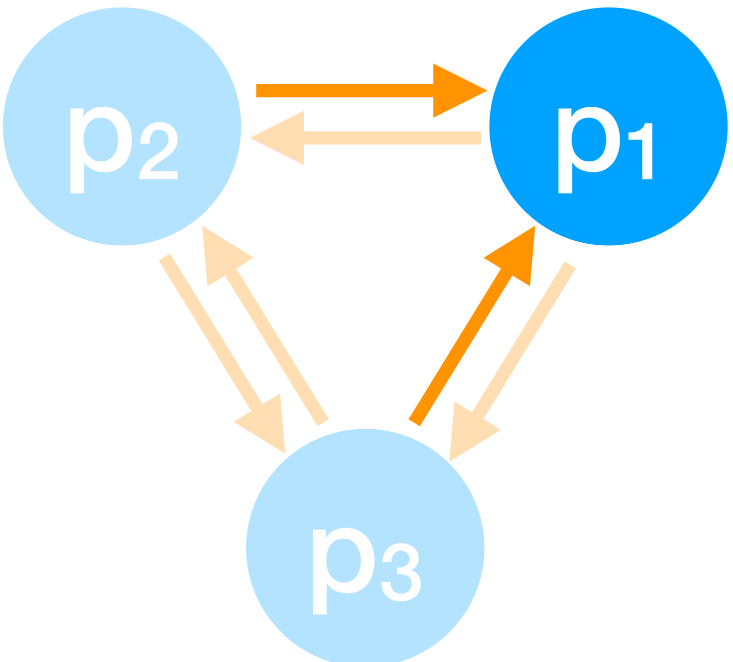
INTERACTION NETWORKS FOR JET TAGGING

arXiv:1908.05318

arXiv:1909.12285



- A **graph** of objects and their connections is defined

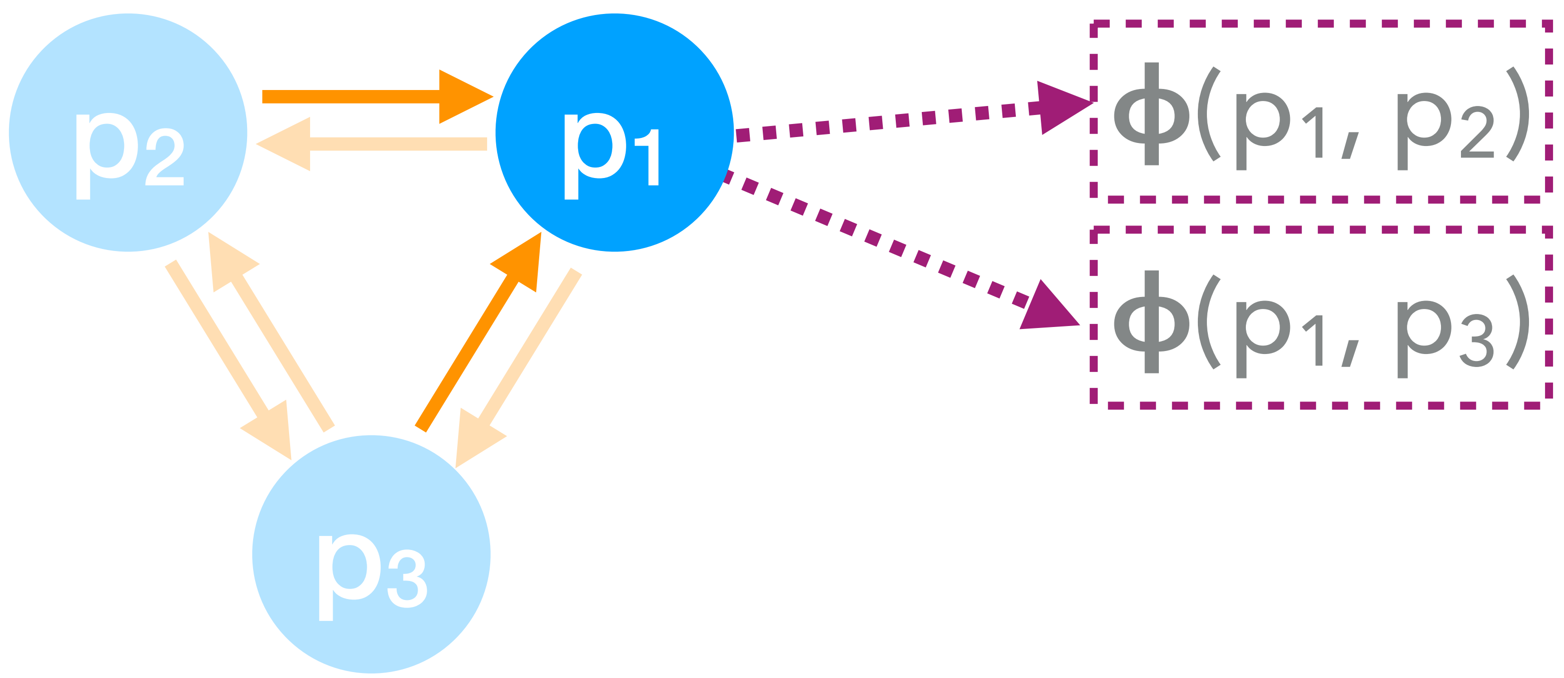


- A **graph** of objects and their connections is defined

INTERACTION NETWORKS FOR JET TAGGING

arXiv:1908.05318

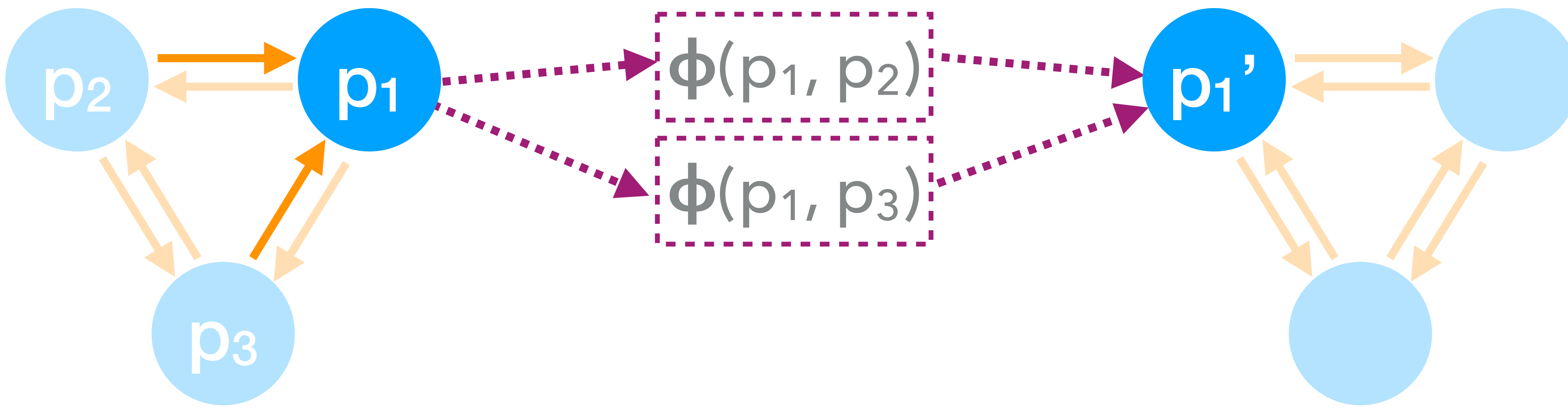
arXiv:1909.12285



- A **graph** of objects and their connections is defined
- **NN** is evaluated on **pairs of connected objects** to produce a message

INTERACTION NETWORKS FOR JET TAGGING

arXiv:1908.05318
arXiv:1909.12285

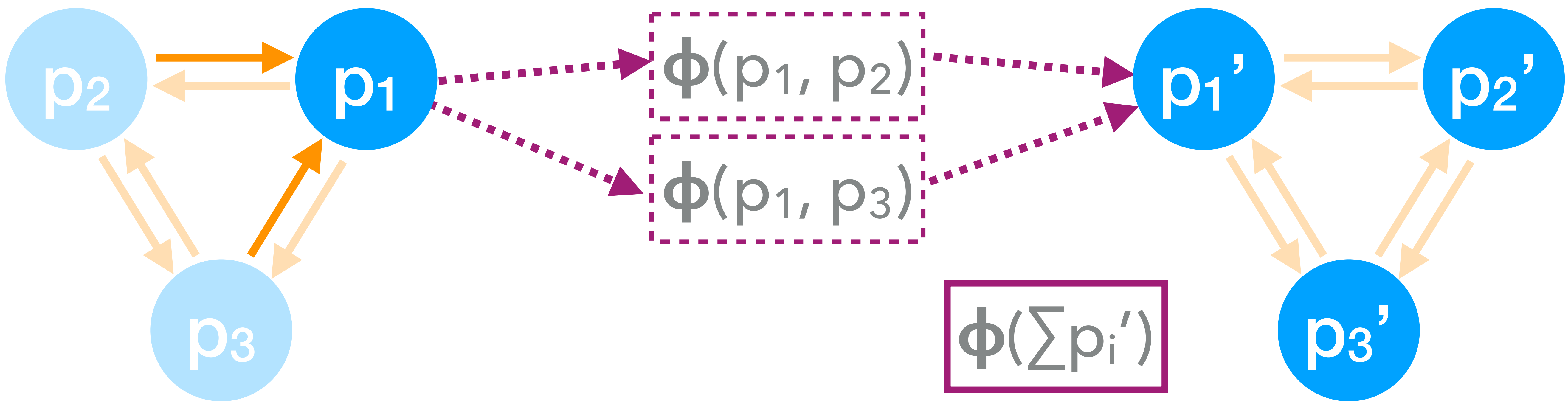


- A **graph** of objects and their connections is defined
- **NN** is evaluated on **pairs of connected objects** to produce a message
- **Messages** are communicated from nearest neighbors (and *summed**) to **update** each object's **hidden state**

*sum preserves permutation invariance

INTERACTION NETWORKS FOR JET TAGGING

arXiv:1908.05318
arXiv:1909.12285



- A **graph** of objects and their connections is defined
- **NN** is evaluated on **pairs of connected objects** to produce a message
- **Messages** are communicated from nearest neighbors (and *summed**) to **update** each object's **hidden state**
- A **single output** is computed based on the *summed** hidden states of all objects in the graph

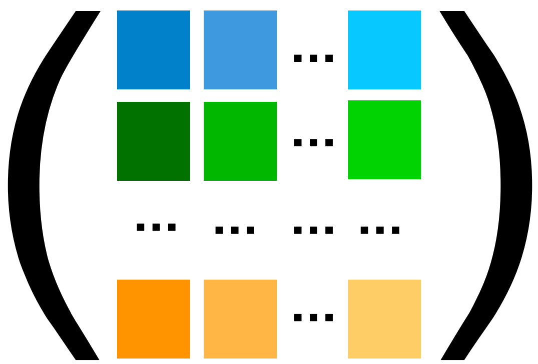
*sum preserves permutation invariance

FULL IN ARCHITECTURE

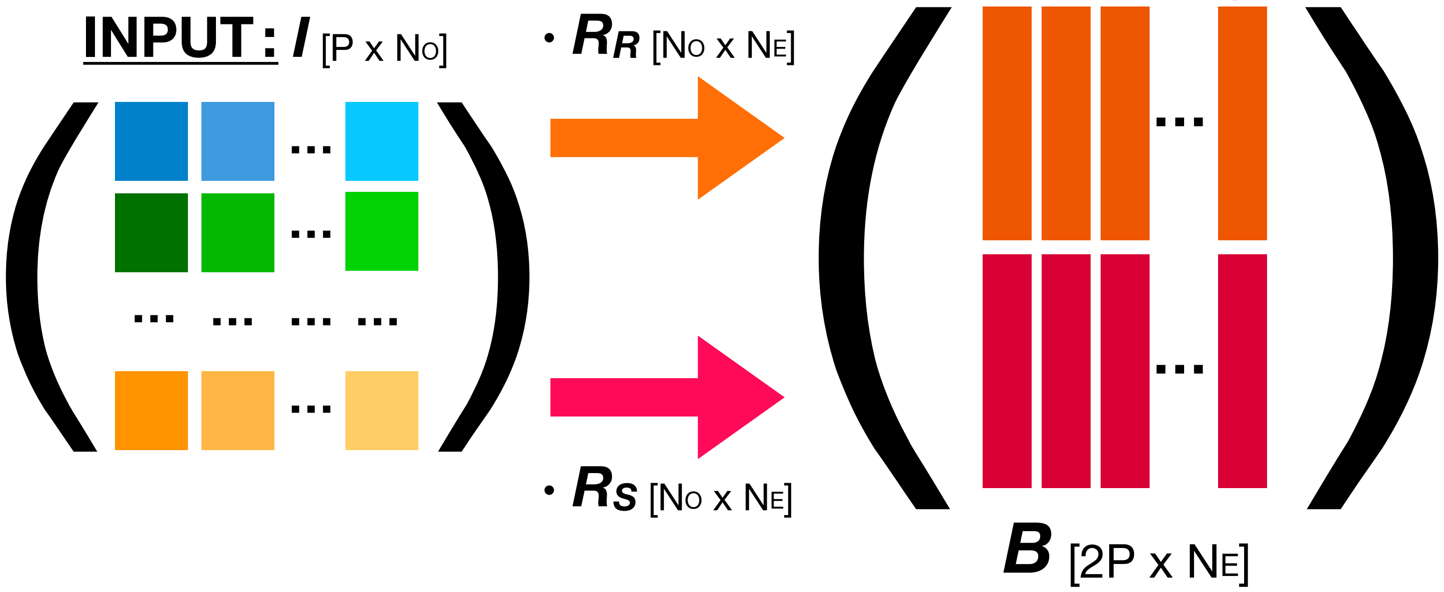
arXiv:1908.05318

arXiv:1909.12285

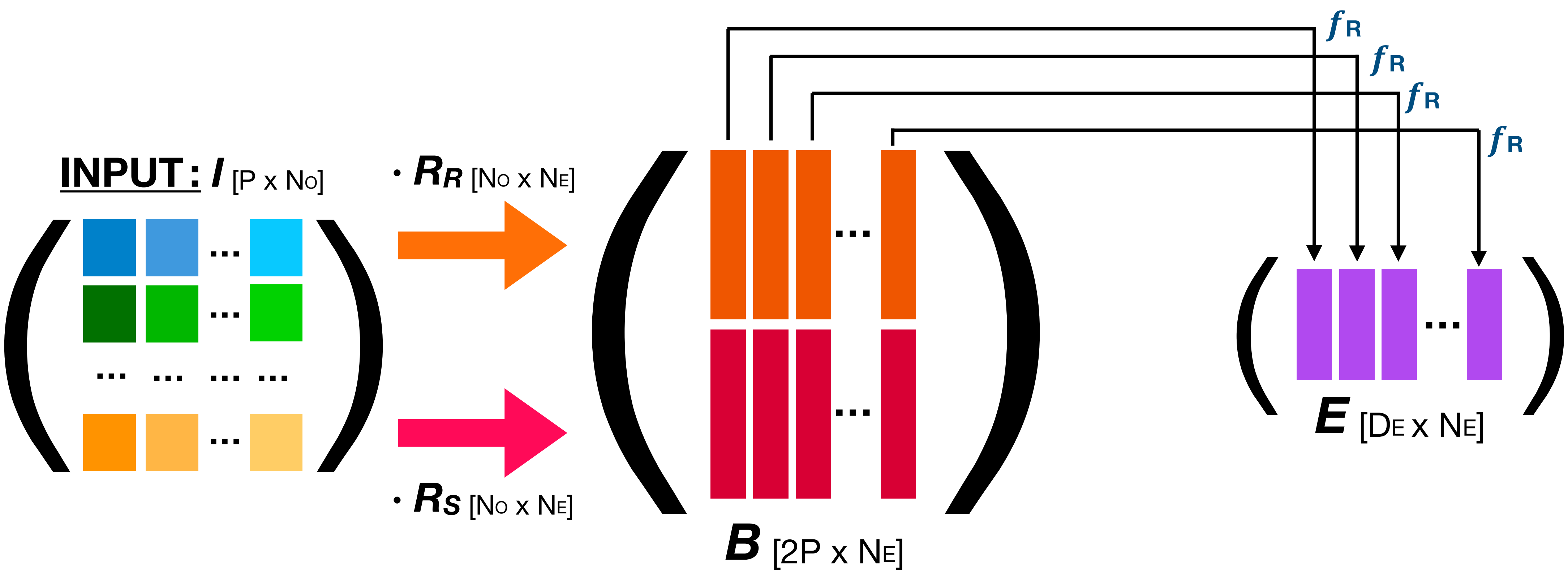
INPUT : $I_{[P \times N_O]}$



FULL IN ARCHITECTURE

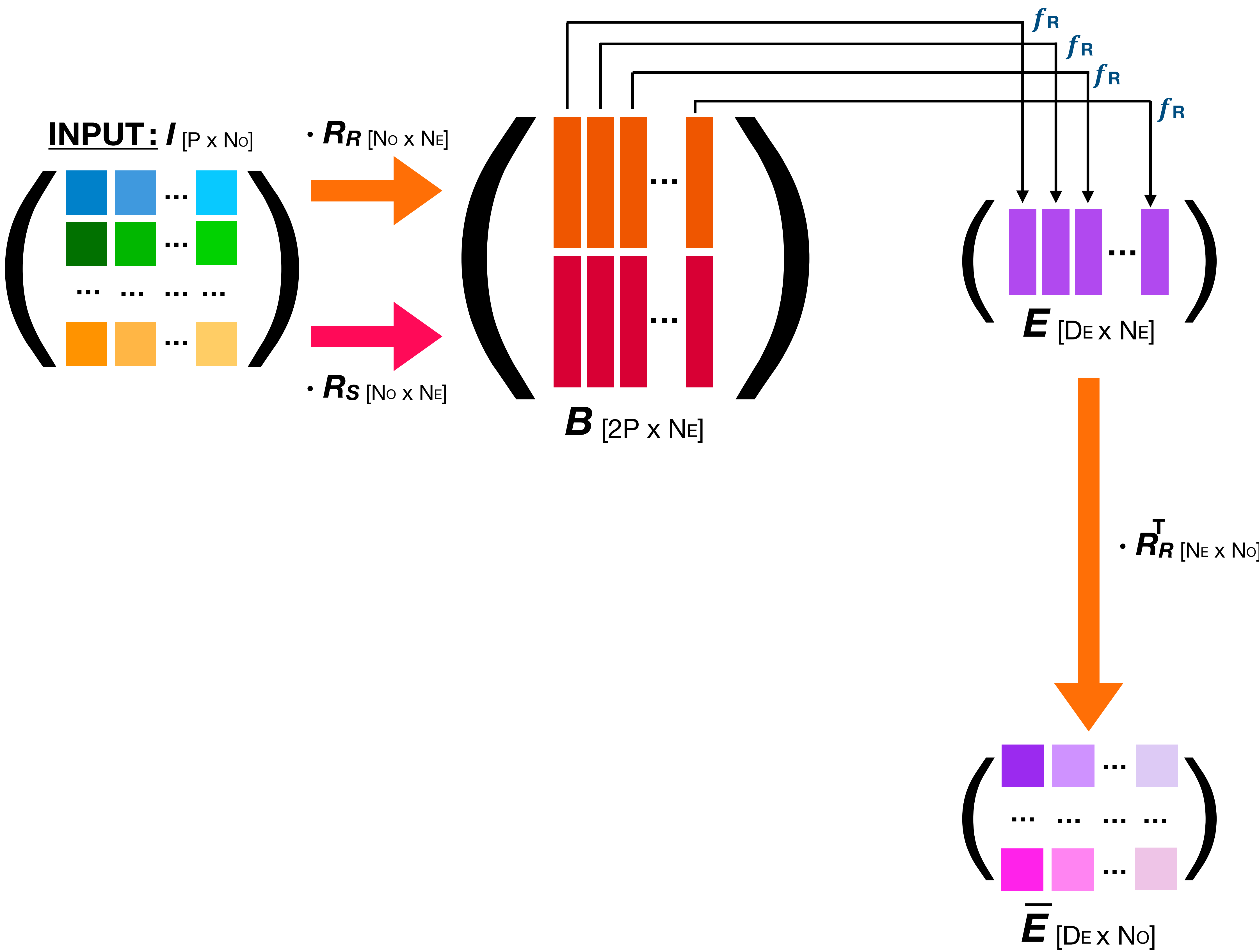


FULL IN ARCHITECTURE



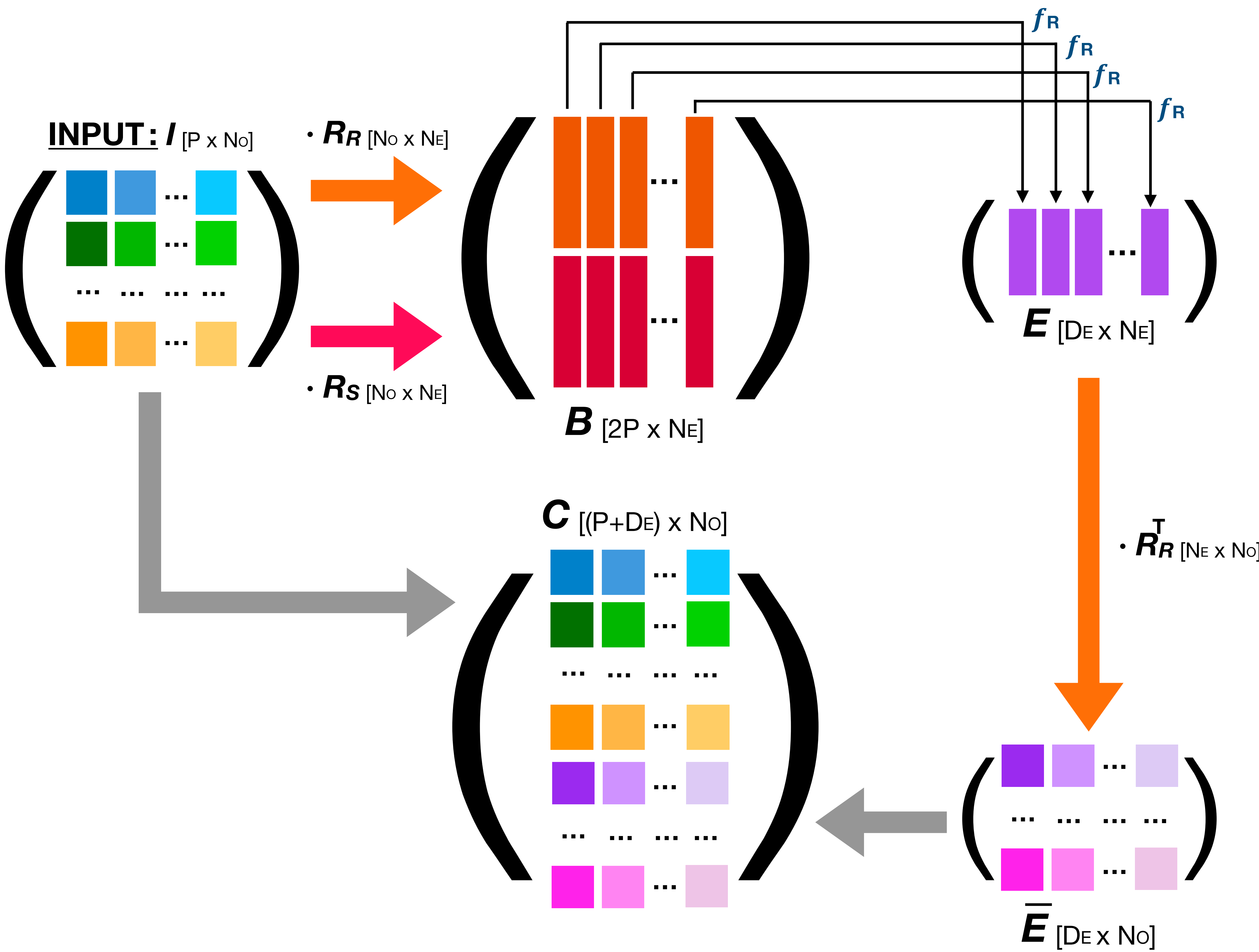
FULL IN ARCHITECTURE

arXiv:1908.05318
arXiv:1909.12285



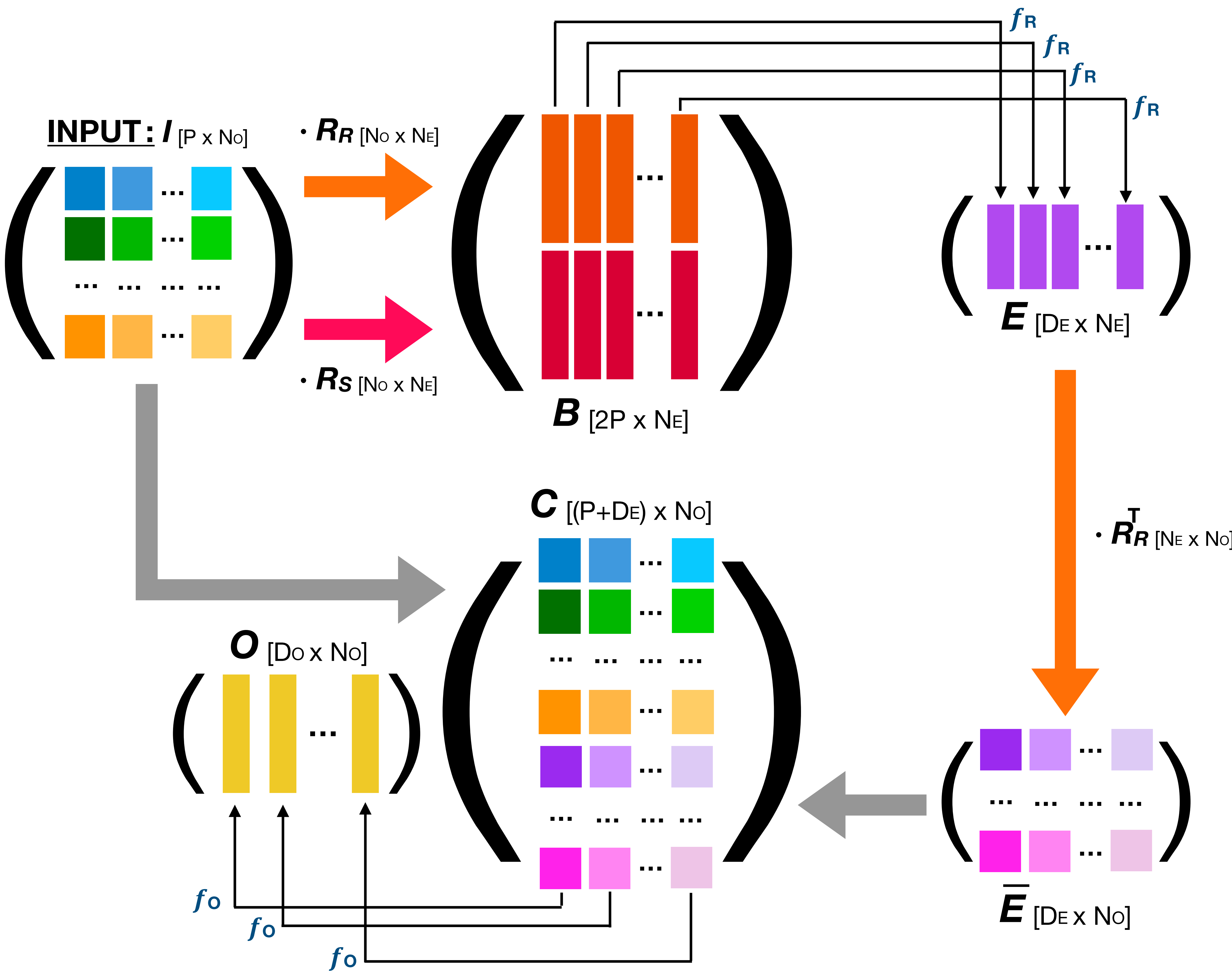
FULL IN ARCHITECTURE

arXiv:1908.05318
arXiv:1909.12285



FULL IN ARCHITECTURE

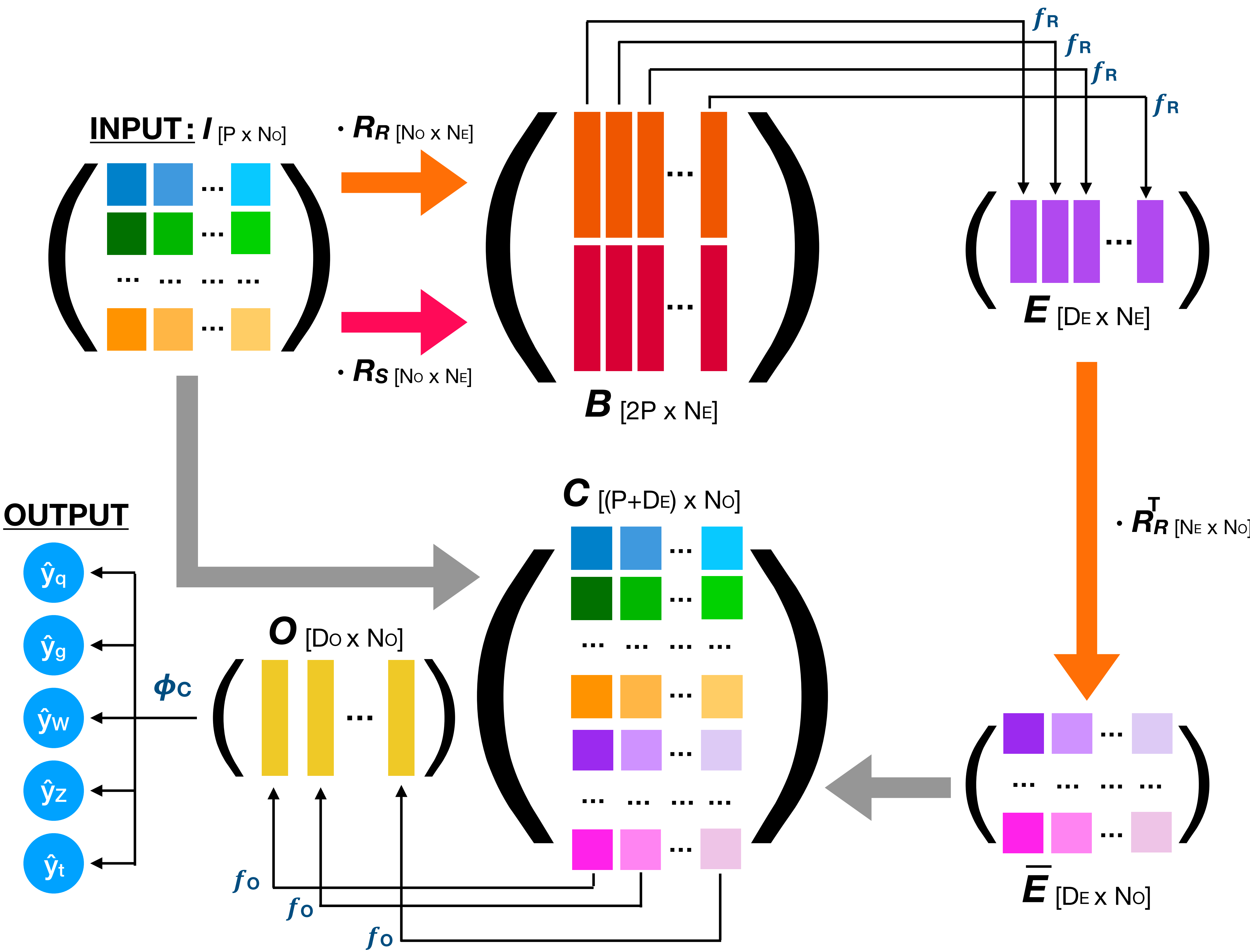
arXiv:1908.05318
arXiv:1909.12285



FULL IN ARCHITECTURE

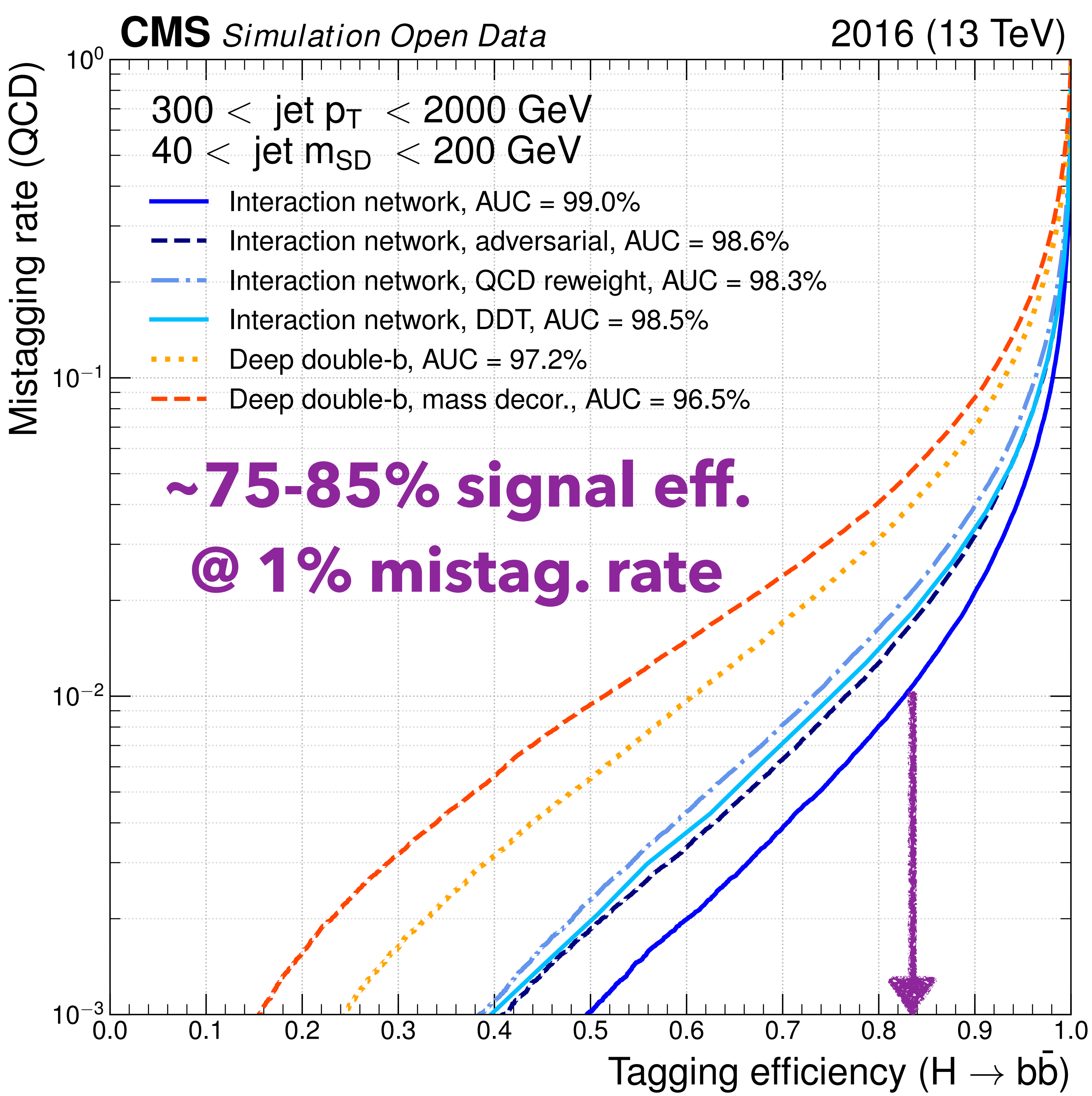
arXiv:1908.05318

arXiv:1909.12285



IN PERFORMANCE FOR HIGGS TAGGING

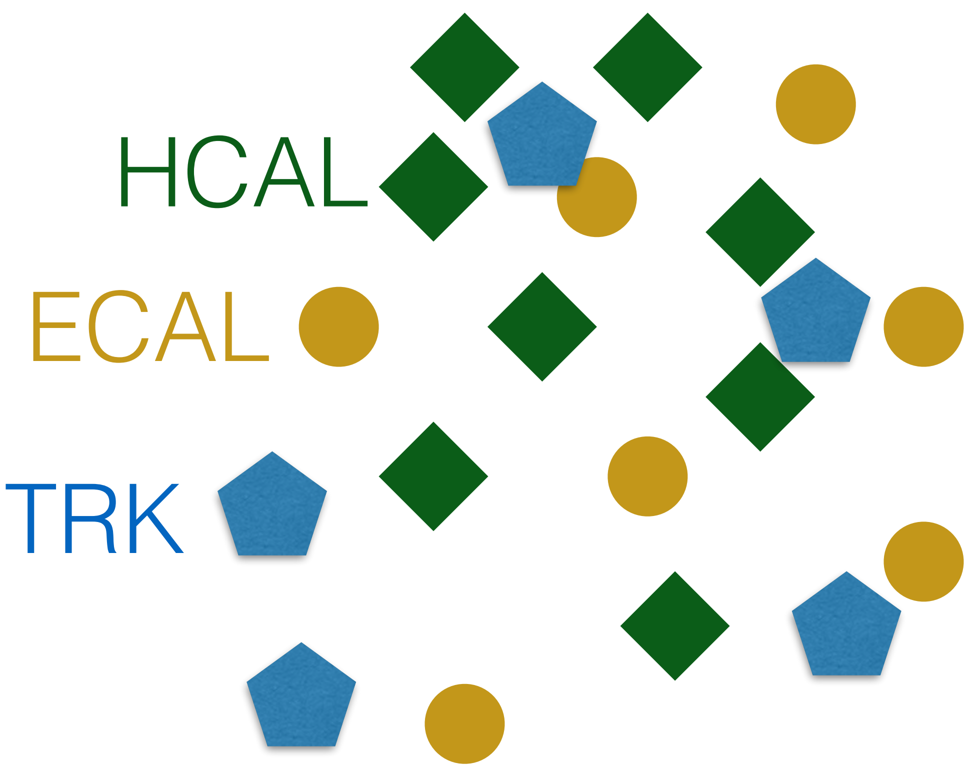
arXiv:1908.05318
arXiv:1909.12285



- **Performance gain**
- GNNs have many other applications in HEP
  - tracking [arXiv:1810.06111]
  - clustering [arXiv:1902.07987]
  - **detector linking (i.e. particle flow)**
  - exotic particle tagging
  - anomaly detection
  - detector simulation

input per event:

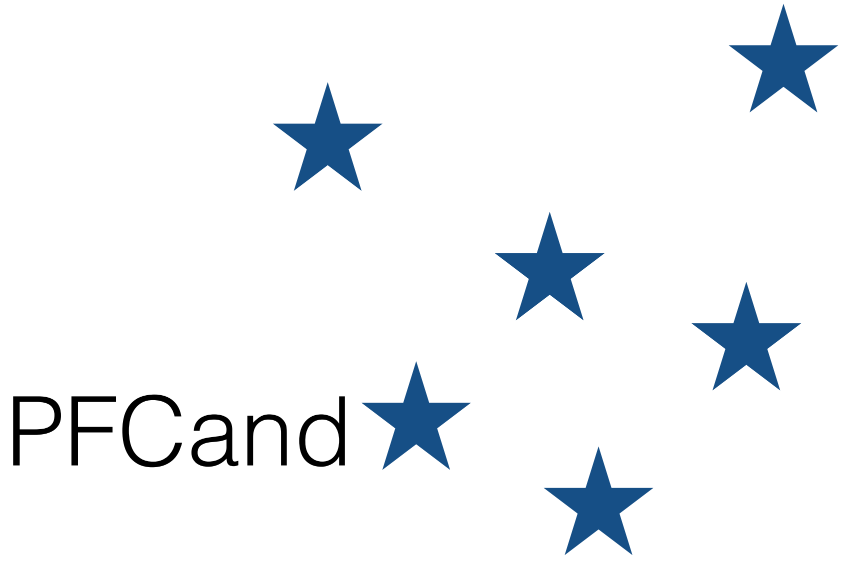
set of elements $\mathbf{e}$={ECAL, HCAL, TRK,...}



~O(5000)

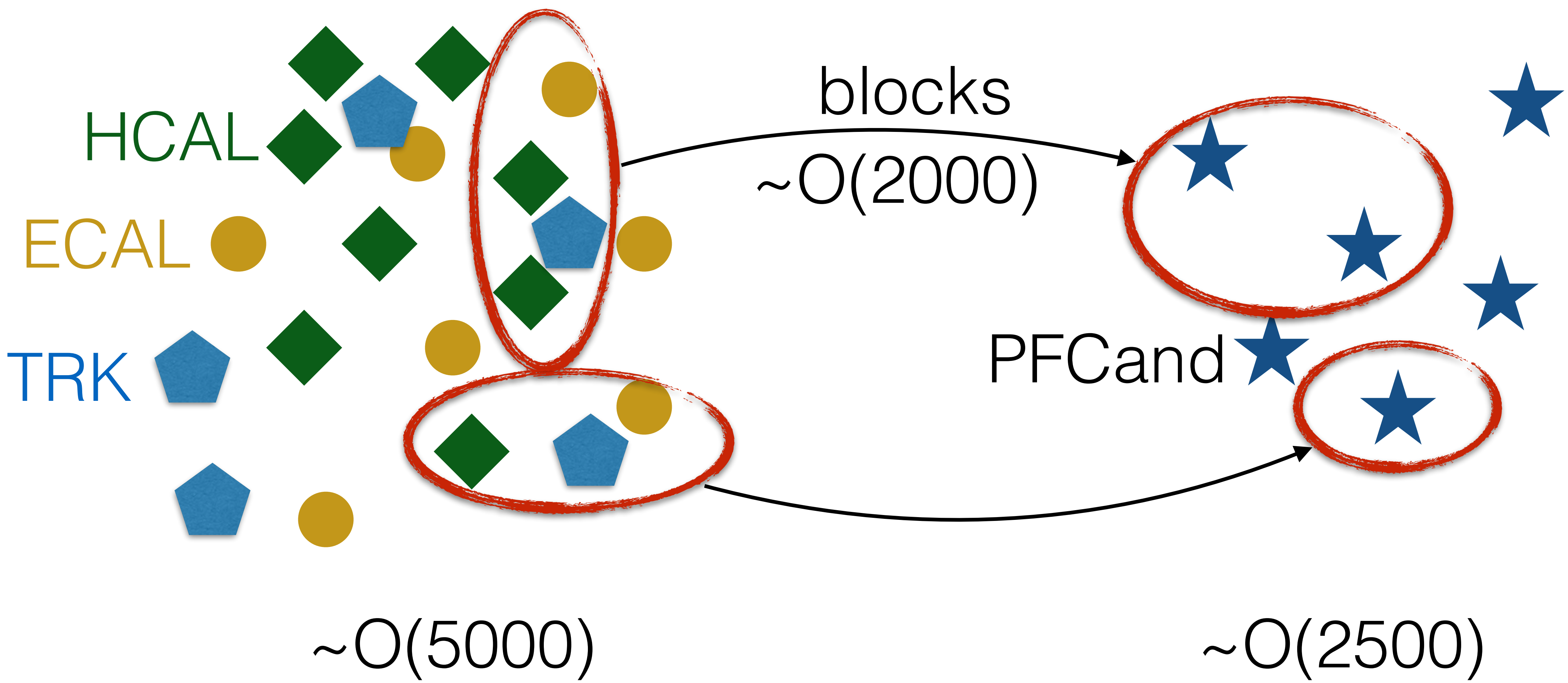
output per event:

set of candidates $\mathfrak{c}=\{\text{PFCand},...\}$



~O(2500)

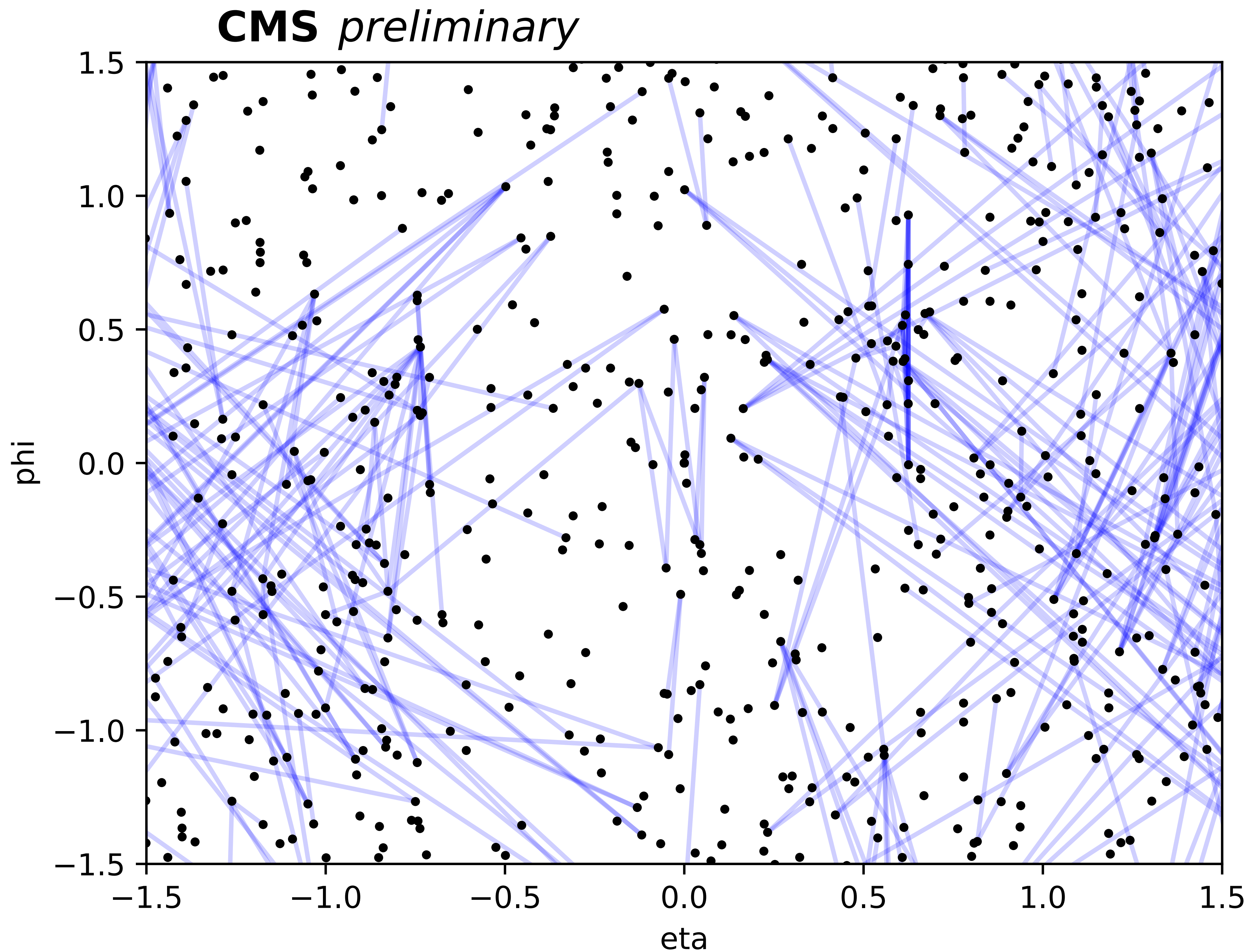
output per event:
set of candidates $\mathfrak{c}$={PFCand,...}



block: candidate associated to elements
A few elements → a few candidates

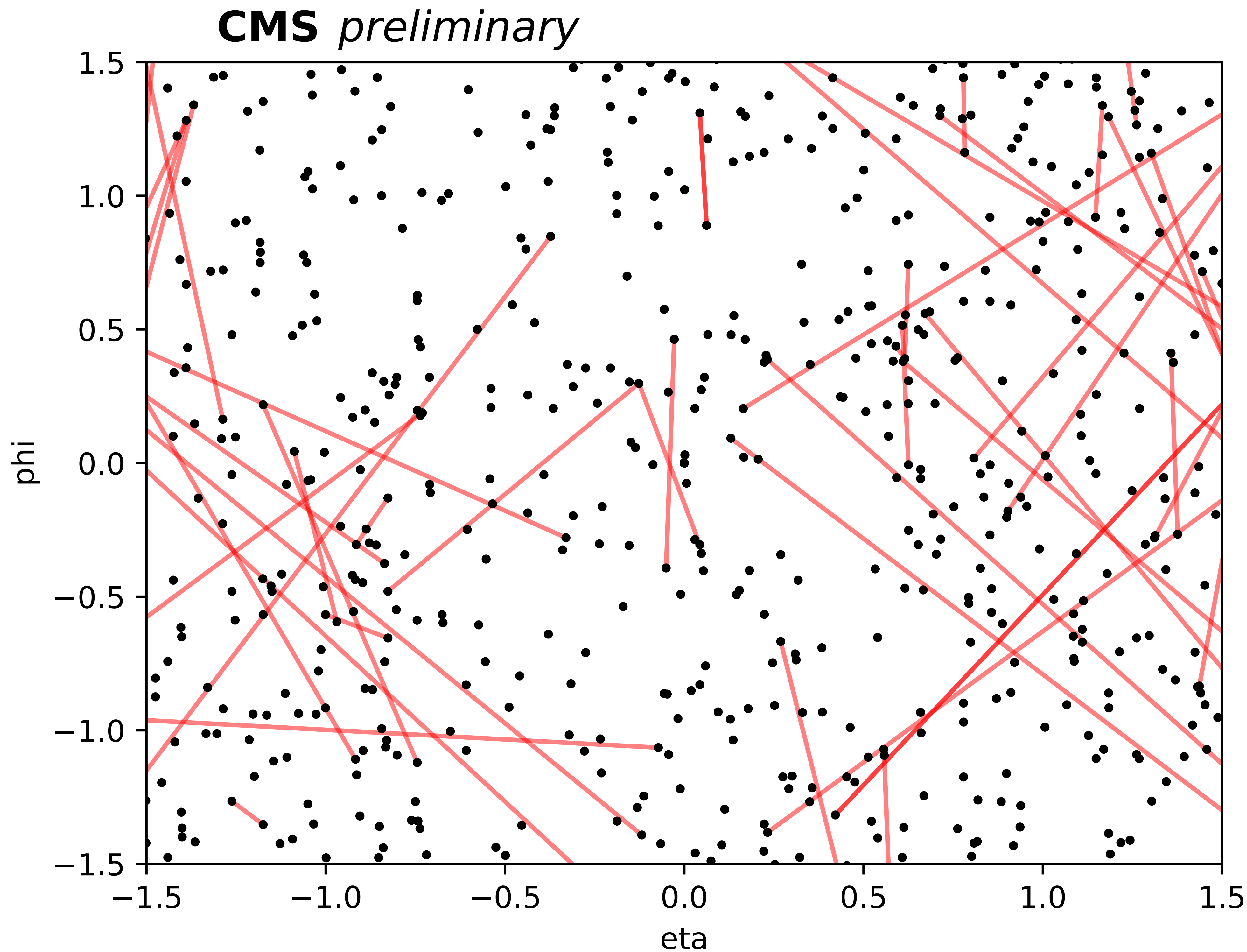
GNNS FOR PARTICLE FLOW

Input Graph



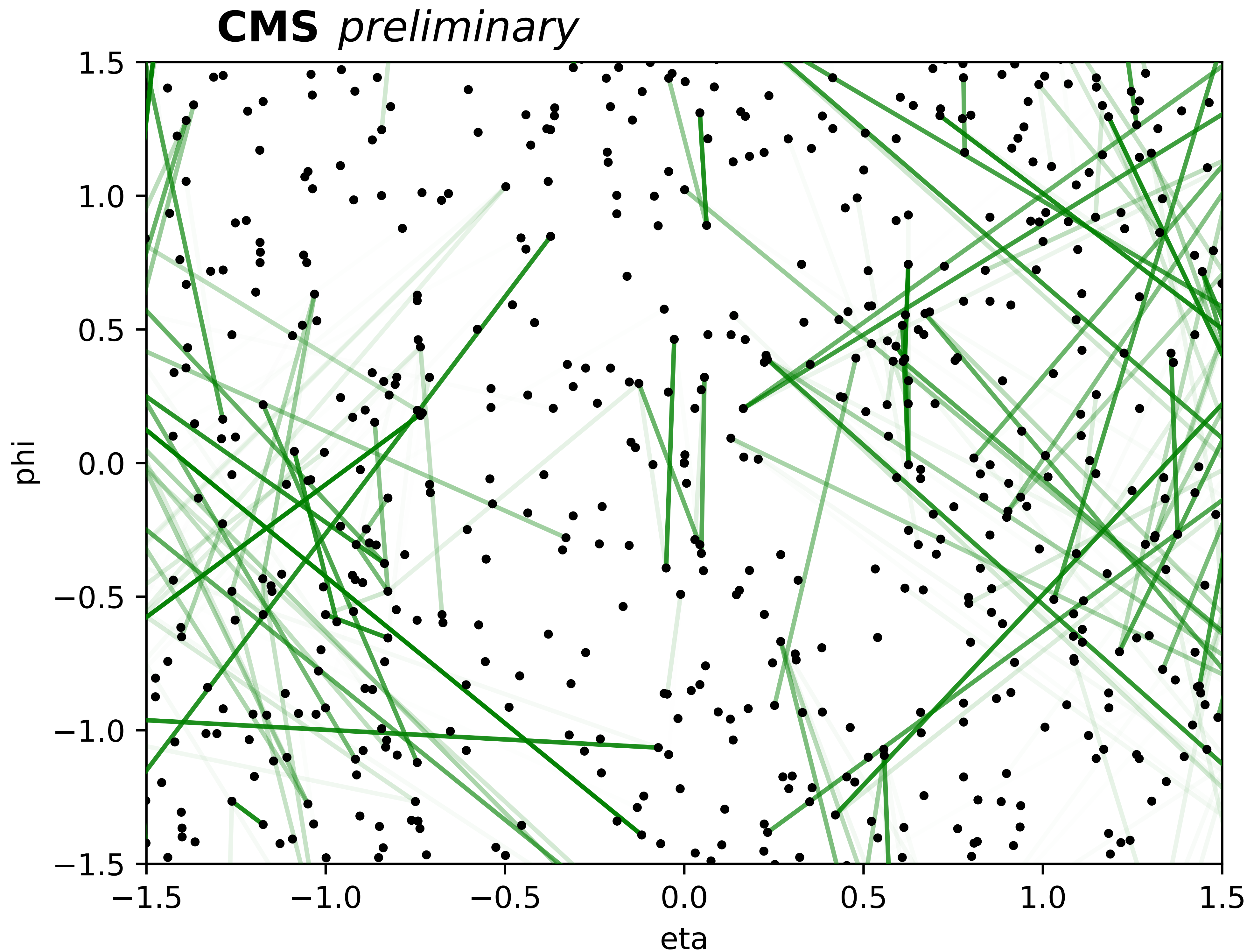
GNNS FOR PARTICLE FLOW

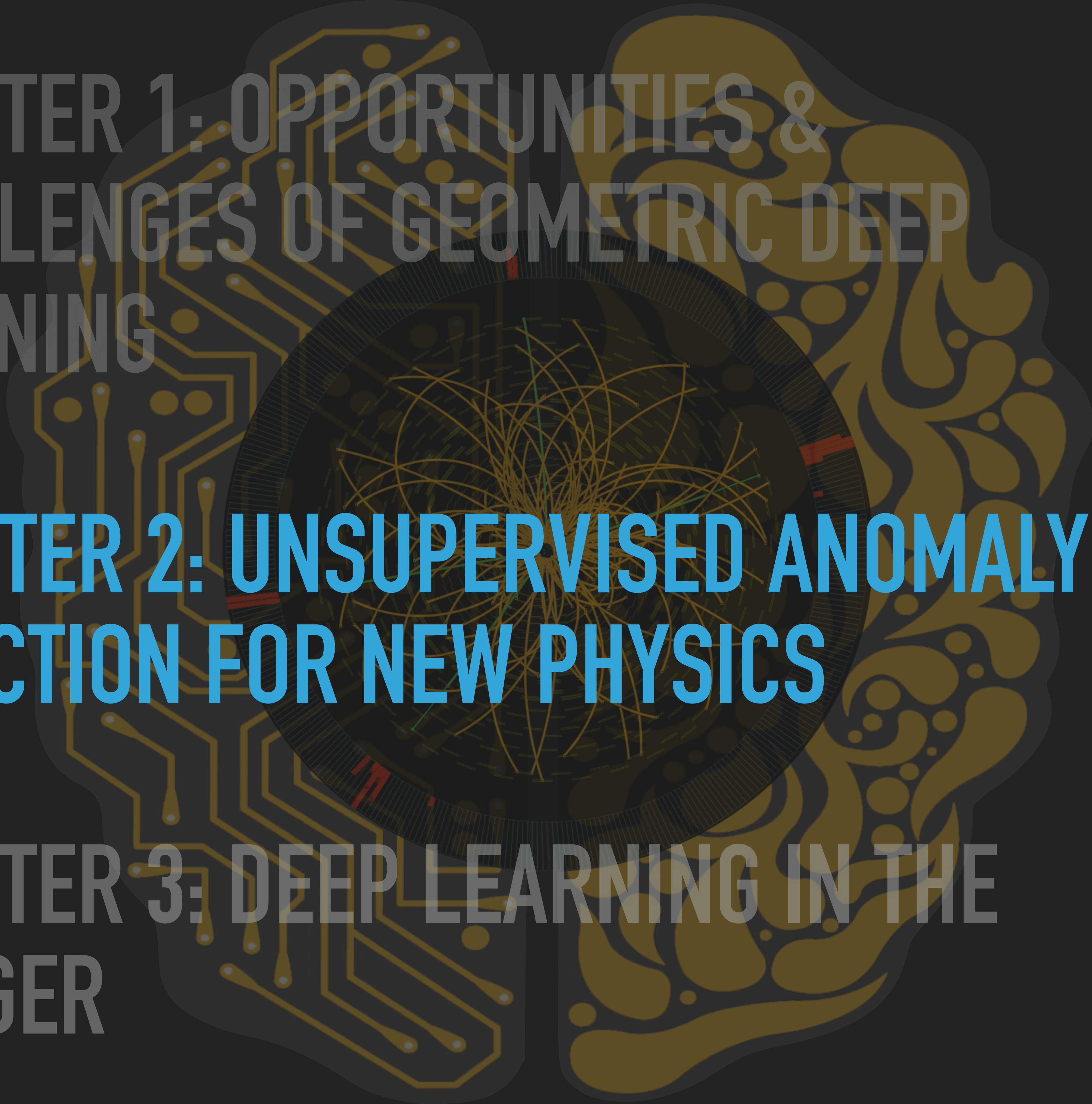
Input Graph Truth Graph



GNNS FOR PARTICLE FLOW

Input Graph Truth Graph Output Graph



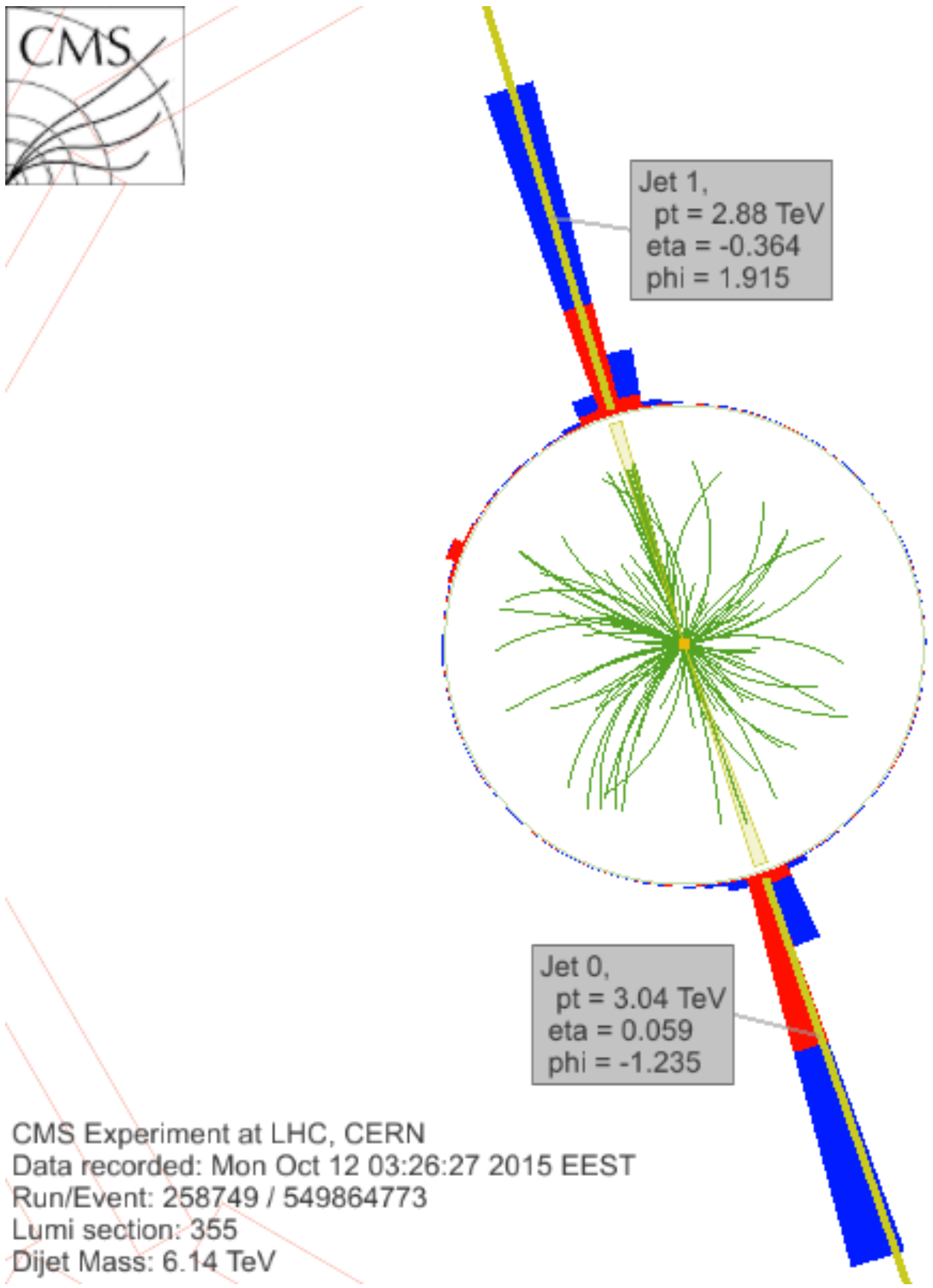


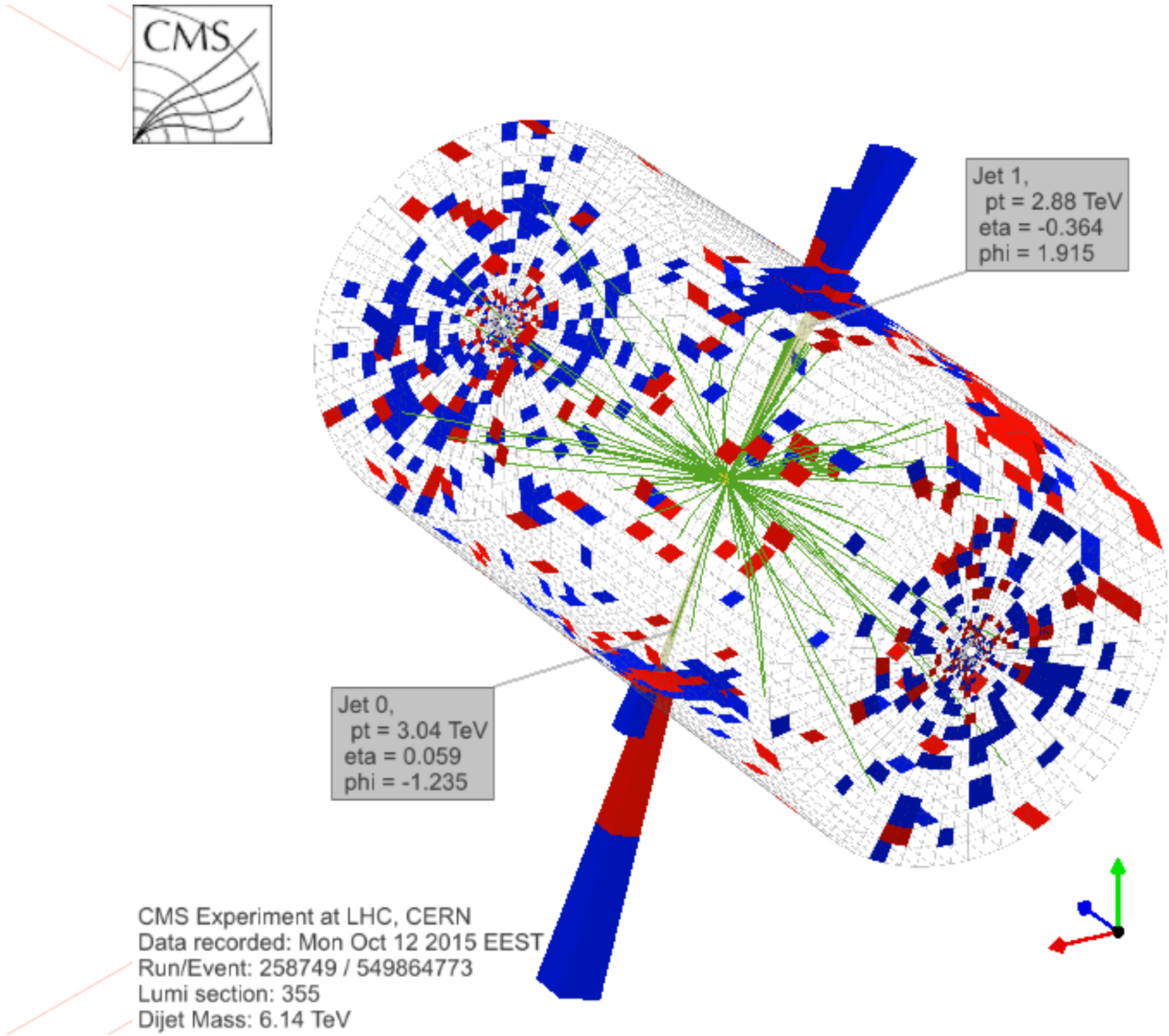
CHAPTER 1: OPPORTUNITIES & CHALLENGES OF GEOMETRIC DEEP LEARNING

CHAPTER 2: UNSUPERVISED ANOMALY DETECTION FOR NEW PHYSICS

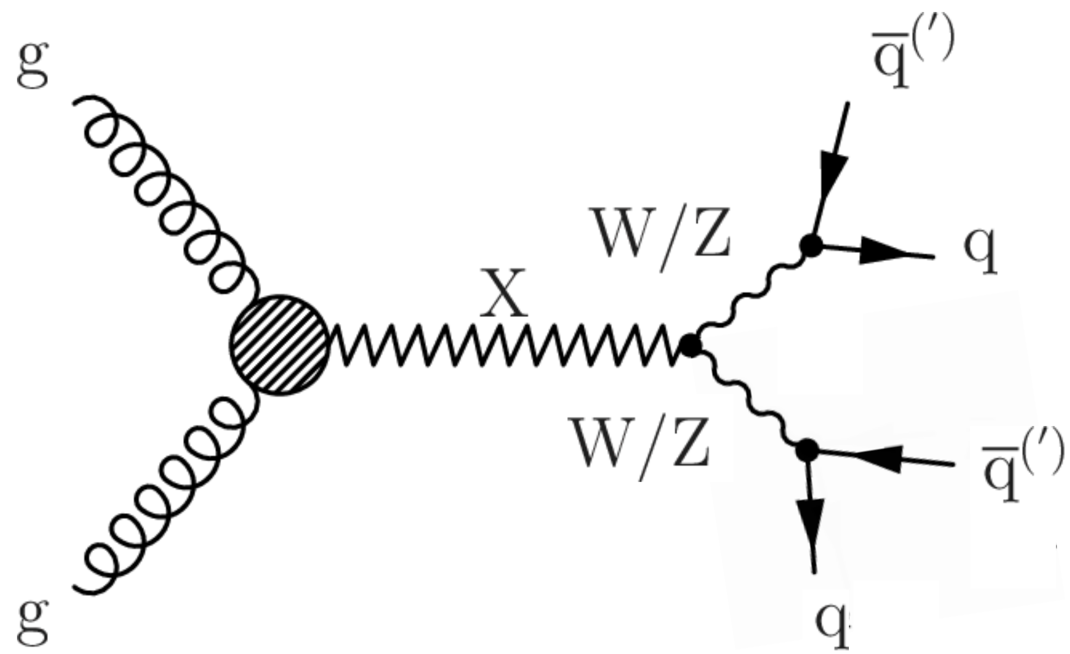
CHAPTER 3: DEEP LEARNING IN THE TRIGGER

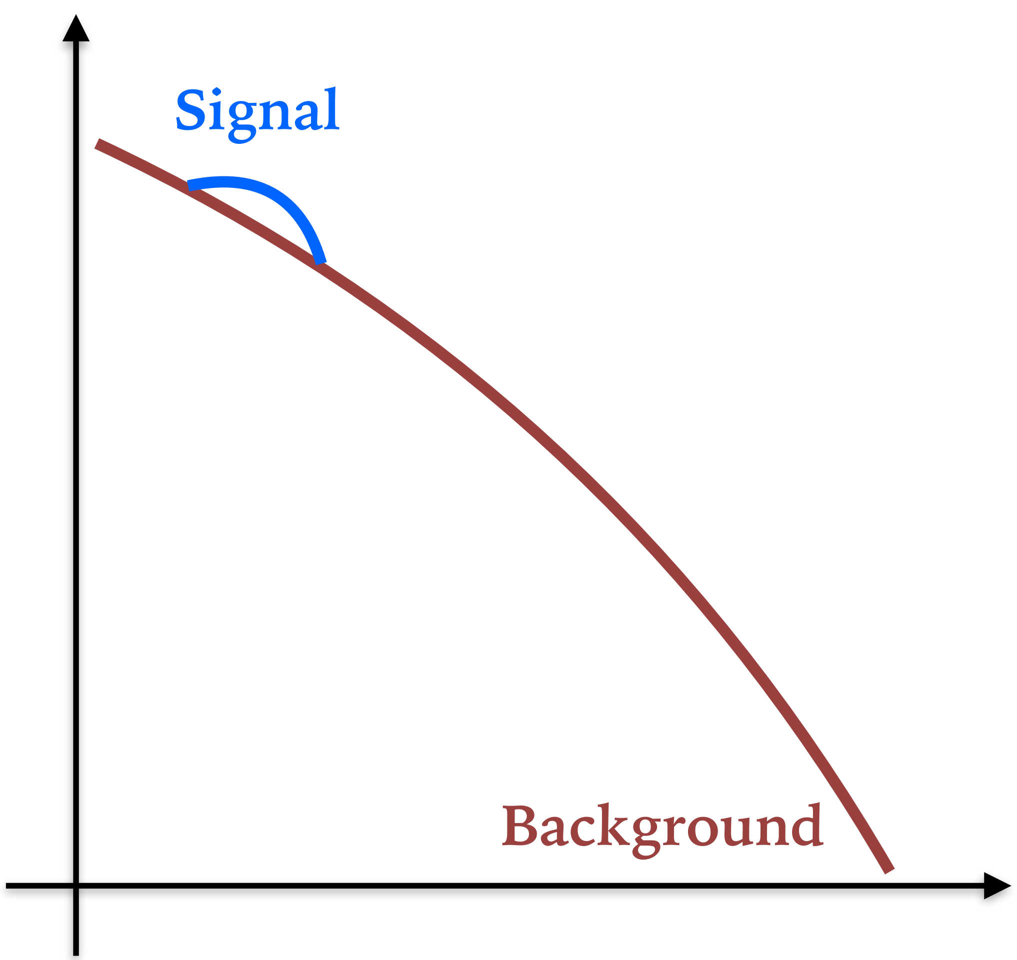
TYPICAL DIJET SEARCH





TYPICAL DIJET SEARCH

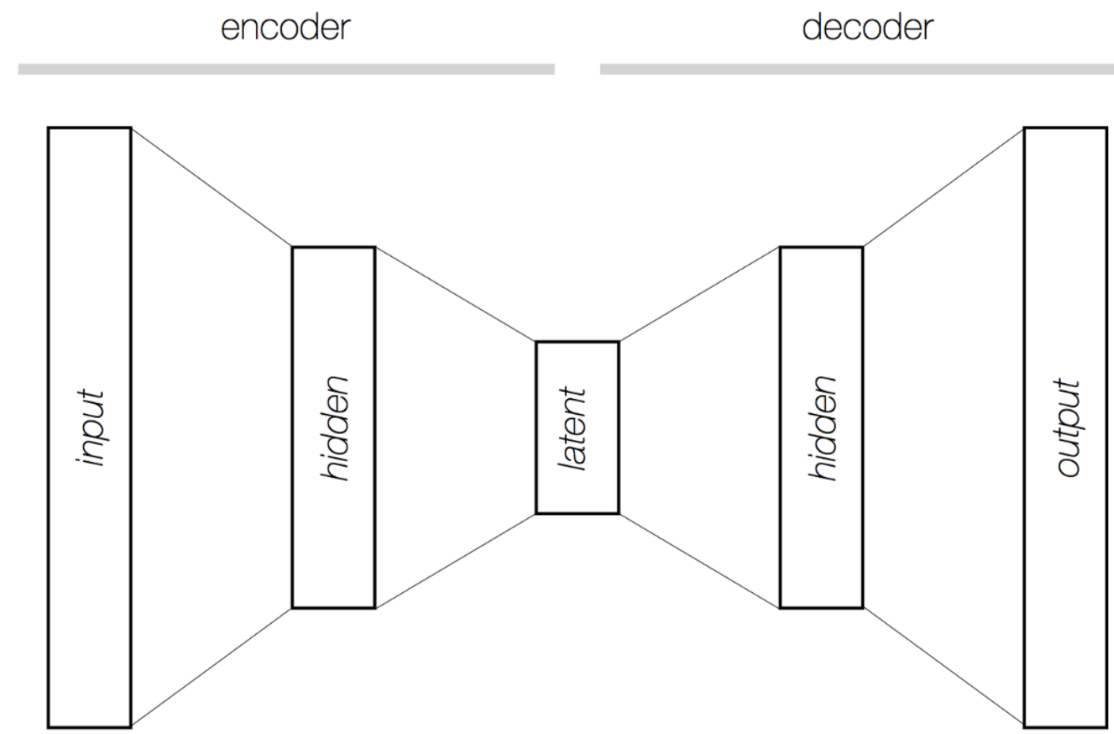


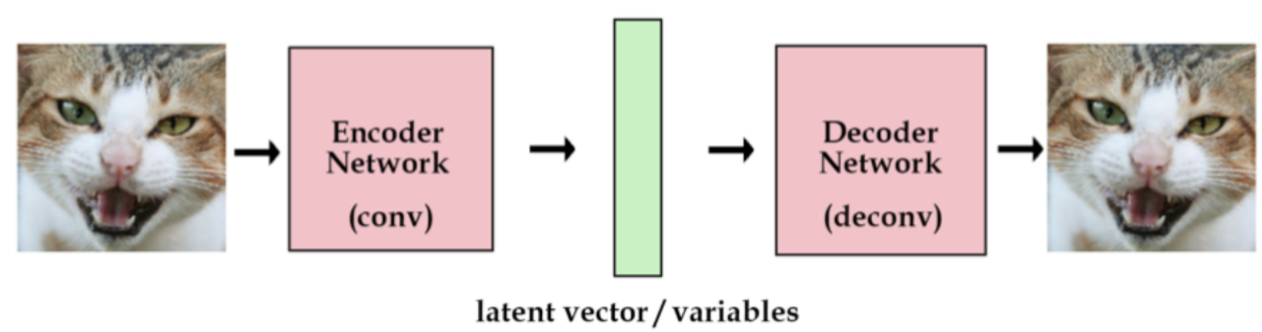


- Look for new heavy particle decaying to two (wide) jets
  - Compute invariant mass of two high-p_T (wide) jets
  - Look for a bump (indicating a new resonance) over a smoothly falling background
- Problems
  - Very large background
  - Many signal models; should we create an ML algorithm to identify each one?
- Can we use unsupervised methods for model-agnostic new physics searches?

AUTO ENCODERS IN ONE SLIDE

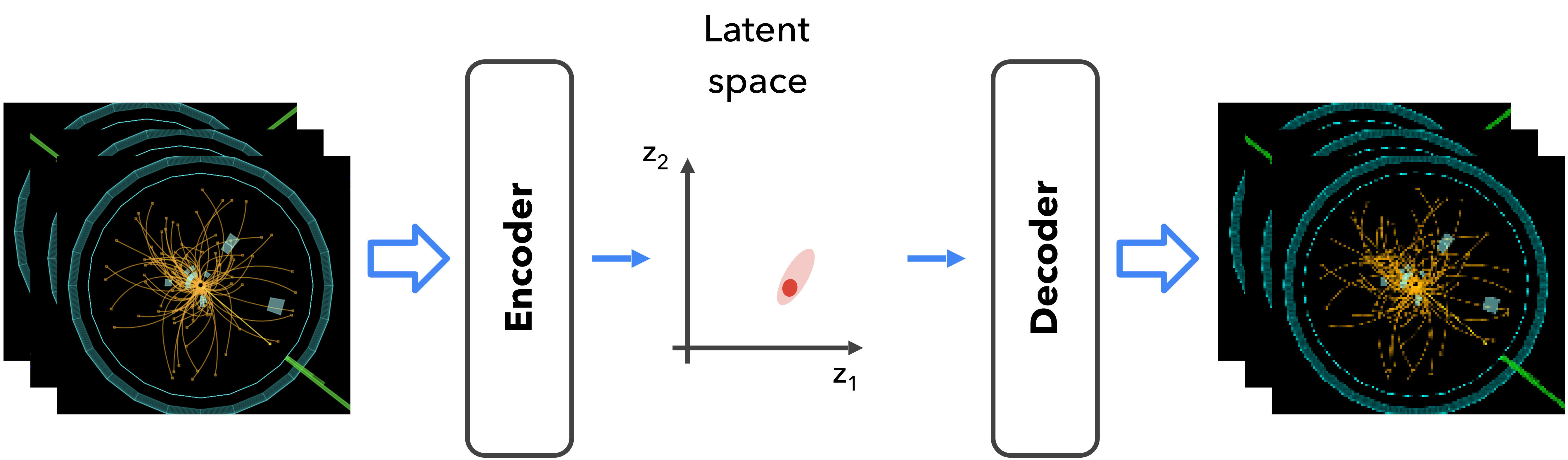
- Map an input onto itself passing through a latent representation
- Unsupervised algorithm, used for data compression, generation, clustering, etc.
- Anomaly: any event whose output is “far” from the input

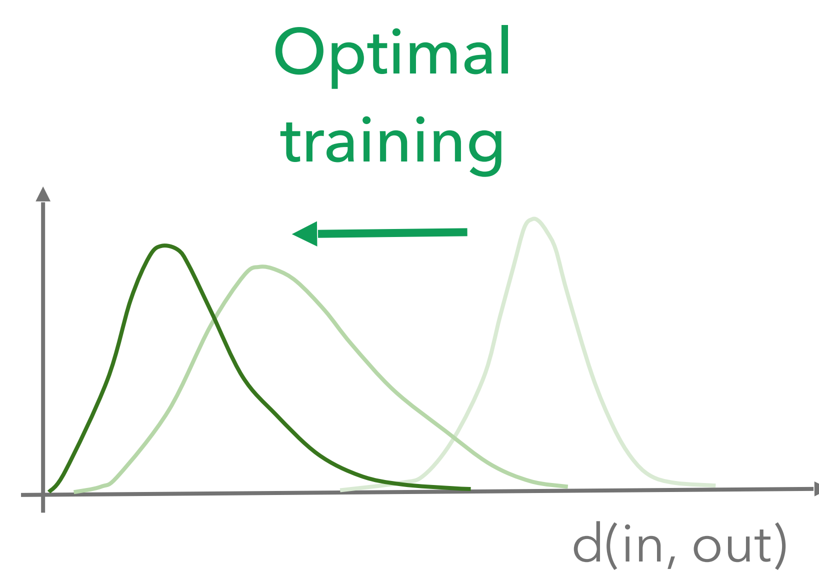




See: [O. Cerri Lepton Photon 2019](), [K. Wozniak CHEP 2019]()

AE ANOMALY DETECTION: TRAINING

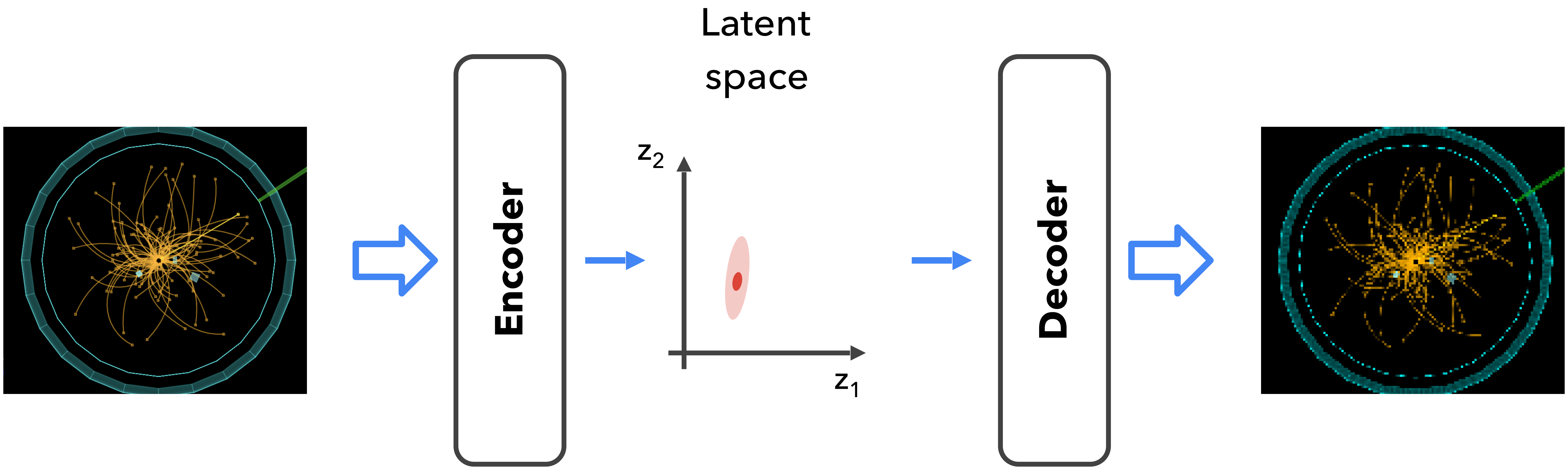




Training:

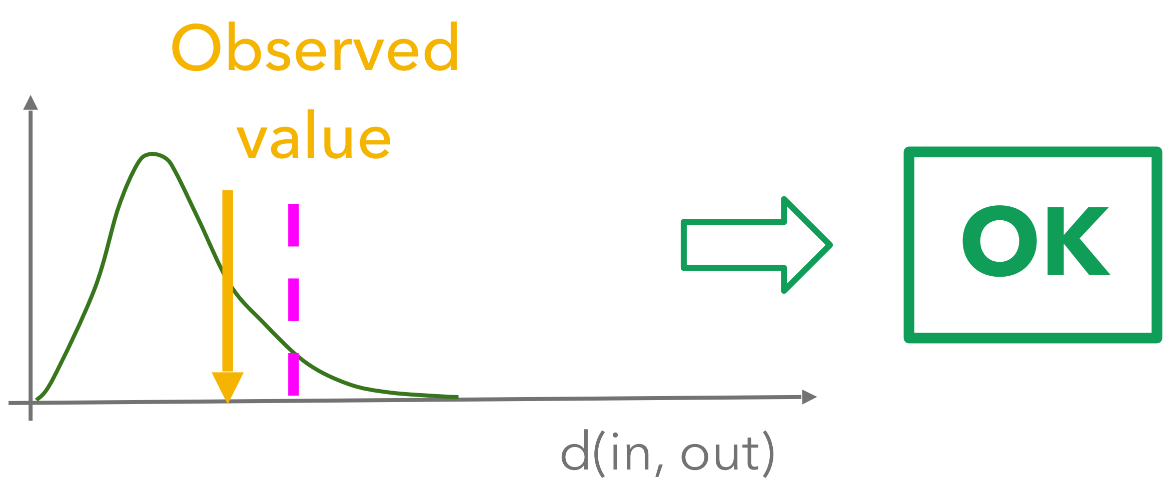
Fit the VAE params to minimize the input-output distance

AE ANOMALY DETECTION: INFERENCE

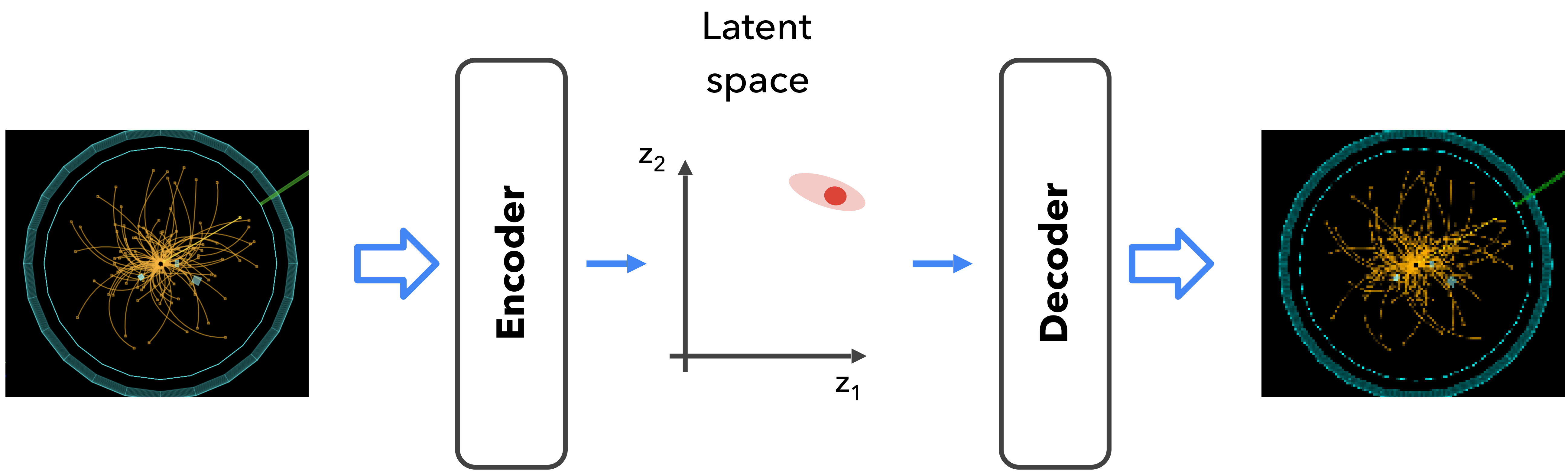


Evaluate:

One-sided hypothesis test on the input-output distance

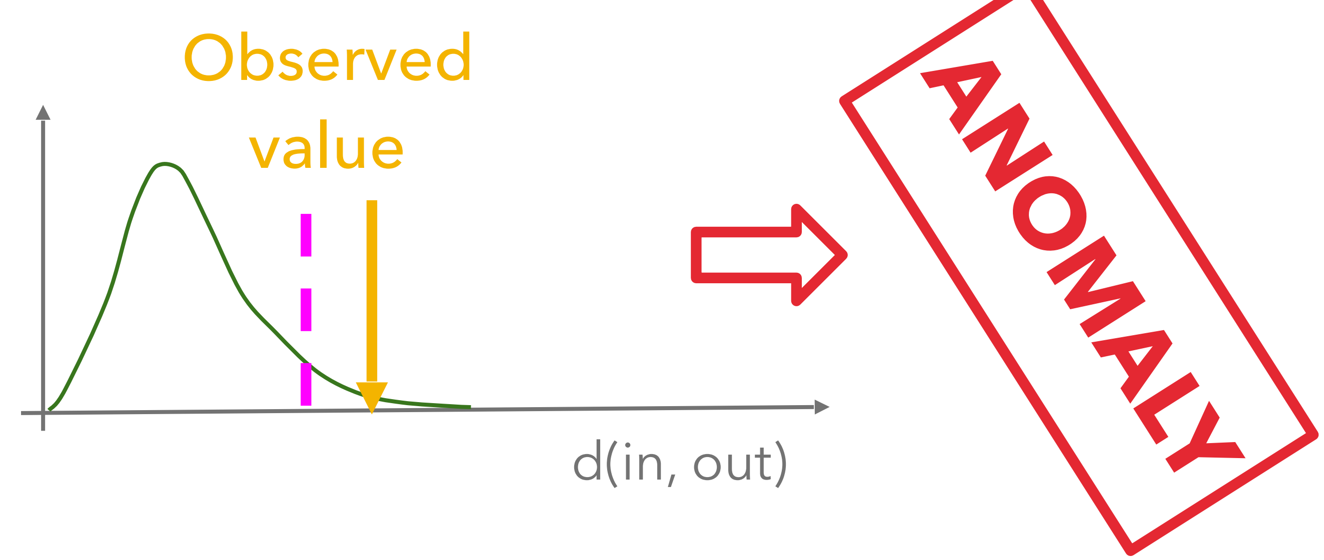


AE ANOMALY DETECTION: INFERENCE



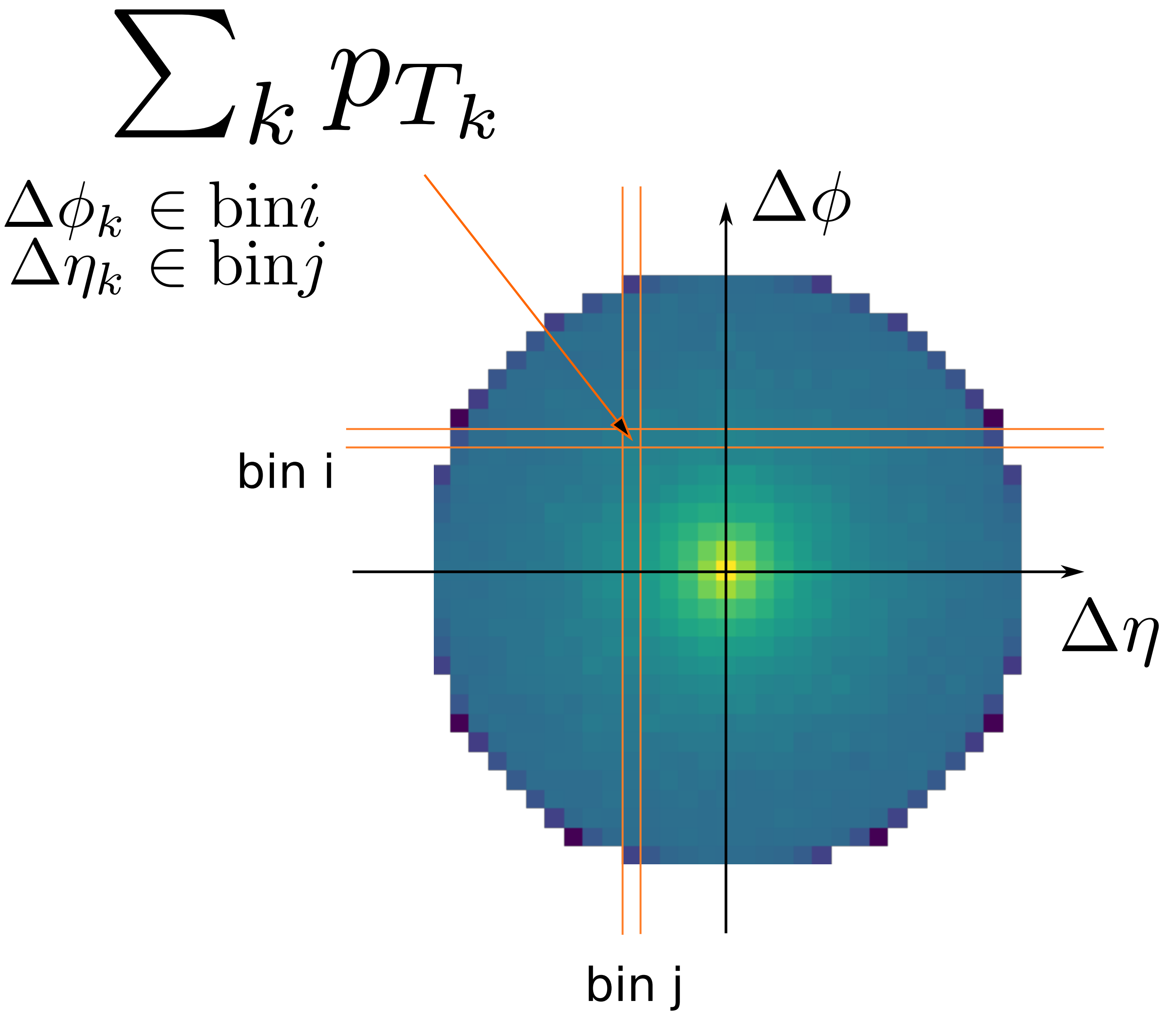
Evaluate:

One-sided hypothesis test on the input-output distance

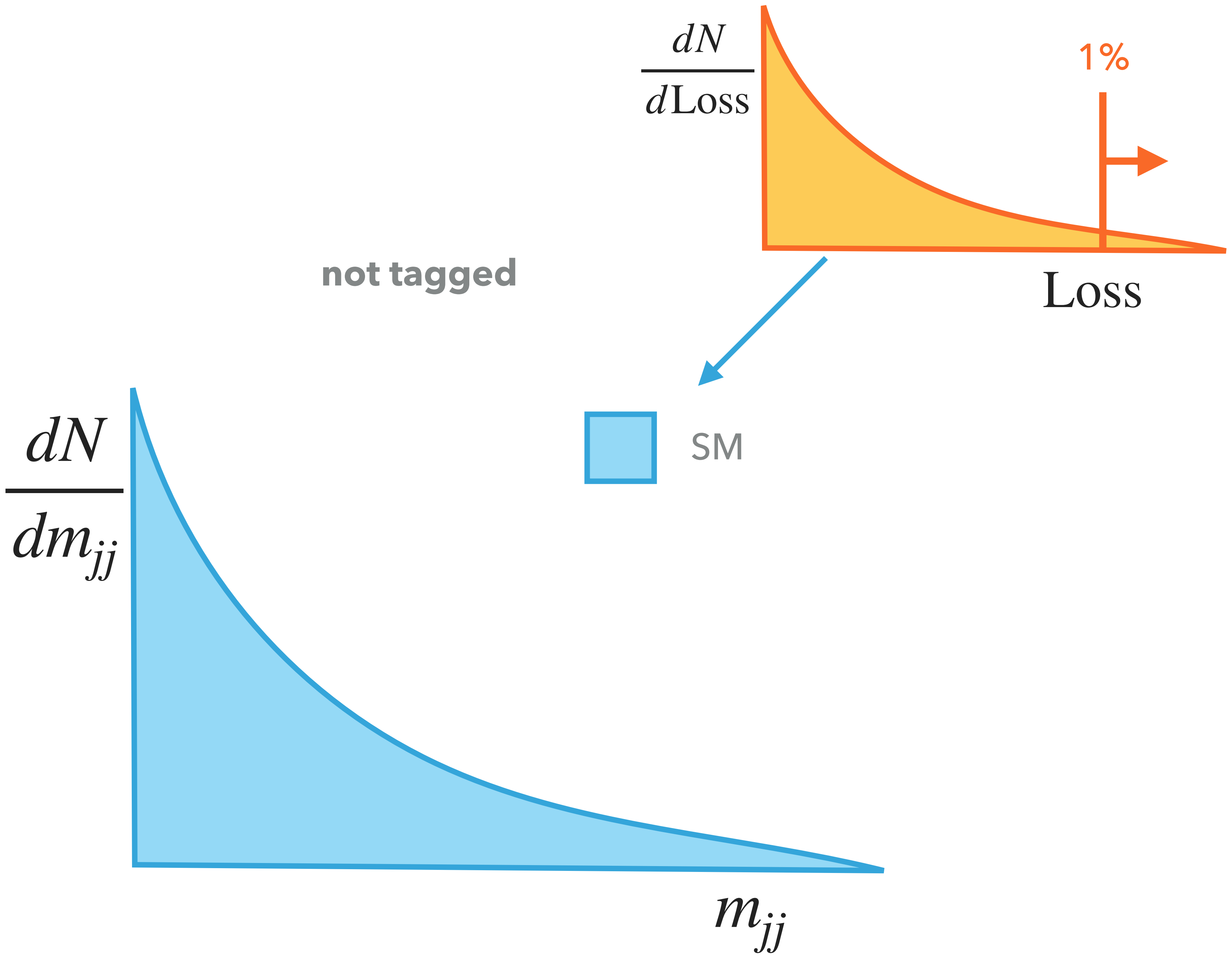


VAE FOR JET IMAGES

- Dataset:
  - QCD dijet simulation (Pythia + Delphes)
- Input:
  - anti-k_T R=0.8 jets
  - transformed to binned, p_T-weighted ***jet images***
- Training in **control region**: 1.4 < |Δη| < 2.4
- Application in **signal region**: |Δη| < 1.4

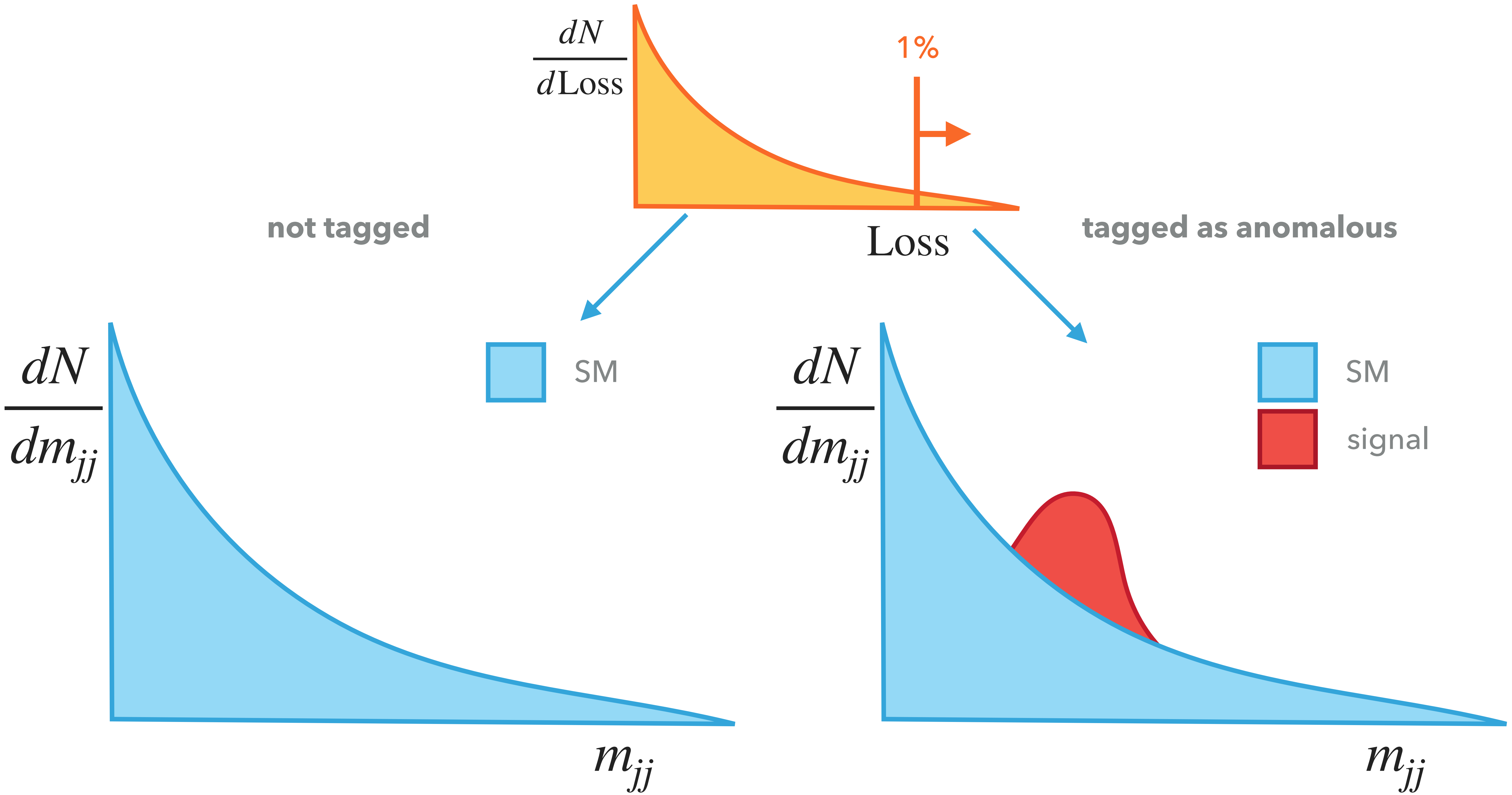


SIGNAL REGIONS



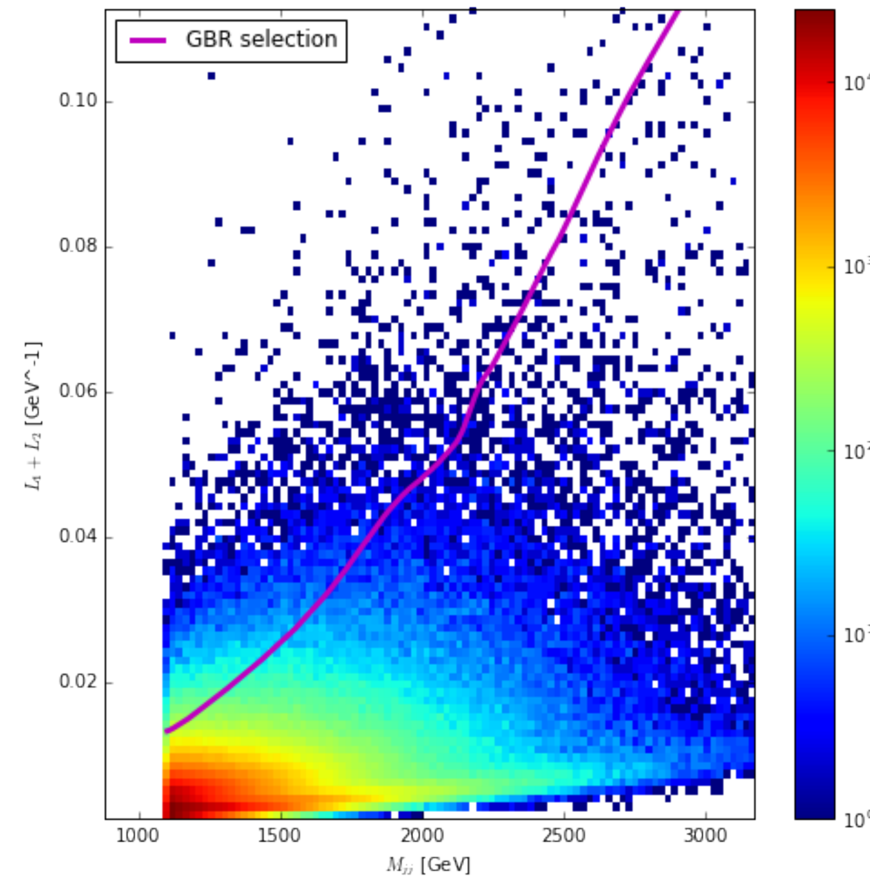
- Cut on the loss to keep 1% of background events
- Use untagged events to constrain background shape in tagged region

SIGNAL REGIONS



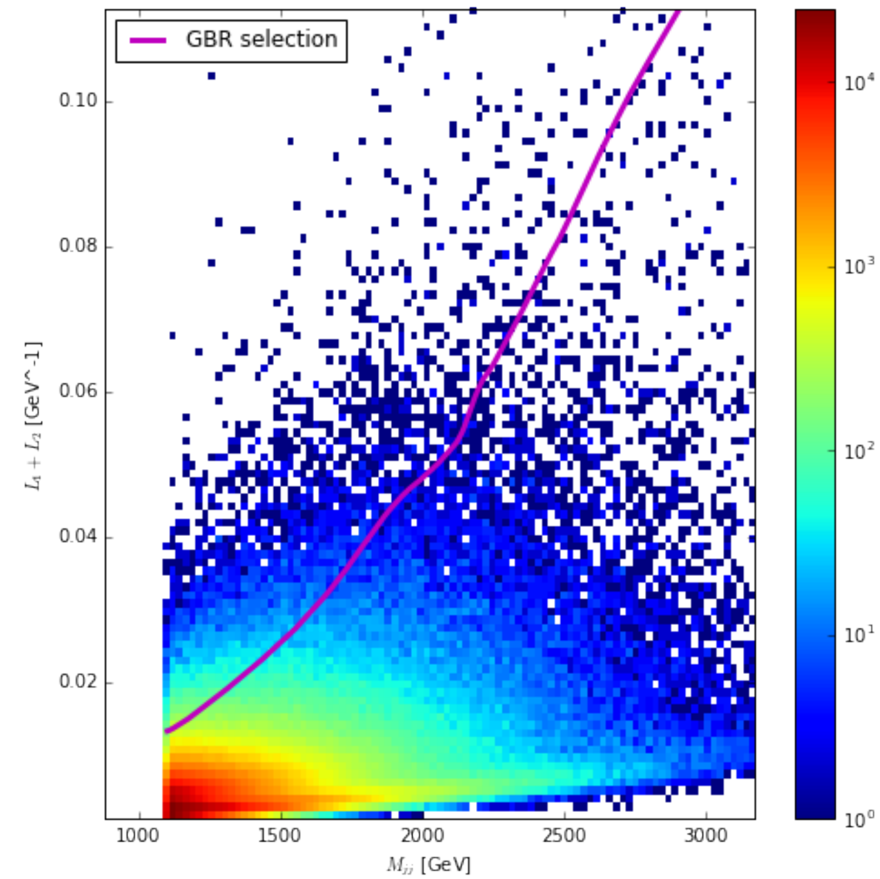
- Cut on the loss to keep 1% of background events
- Use untagged events to constrain background shape in tagged region

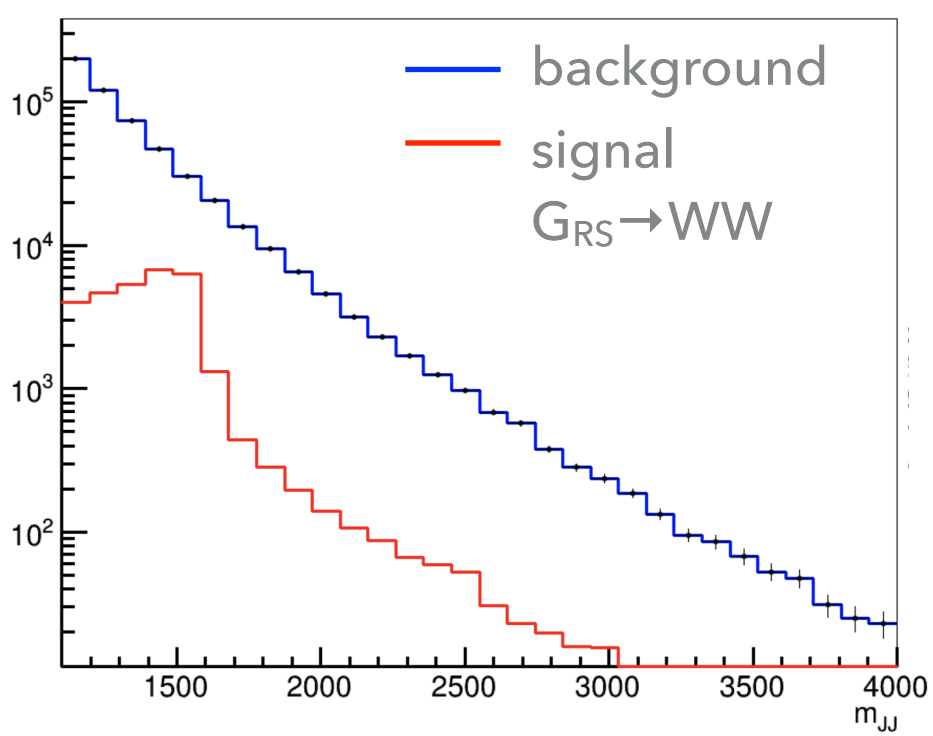
- Note background m_{jj} distribution is not preserved after applying selection on loss (***sculpting***!)
- But we can apply a m_{jj} dependent threshold on the loss to preserve shape of background



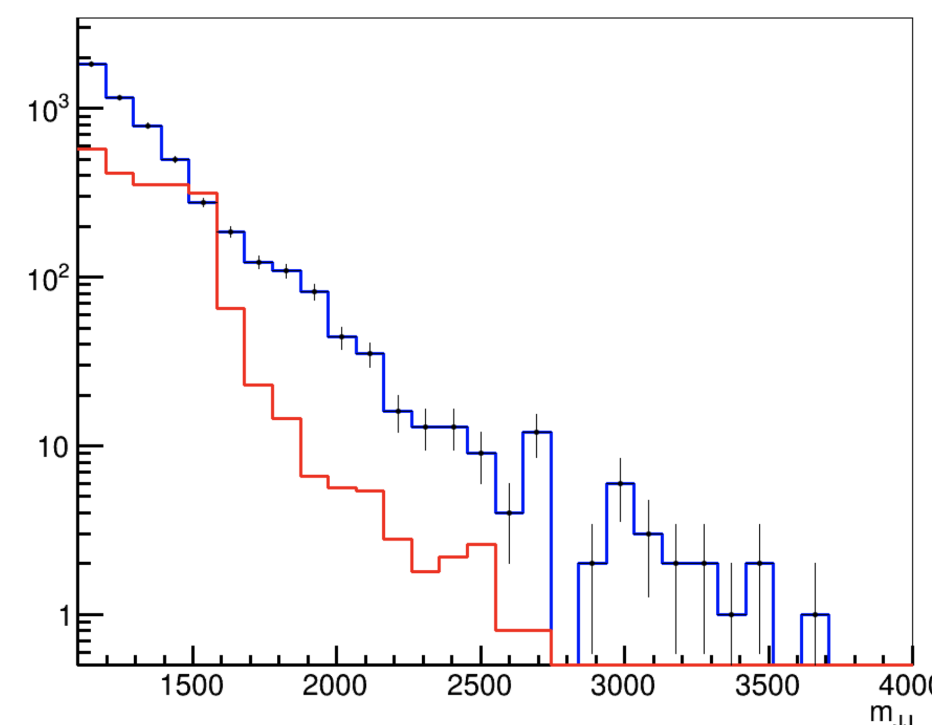
VAE FOR JET IMAGES

- Note background m_{jj} distribution is not preserved after applying selection on loss (***sculpting***!)
- But we can apply a m_{jj} dependent threshold on the loss to preserve shape of background





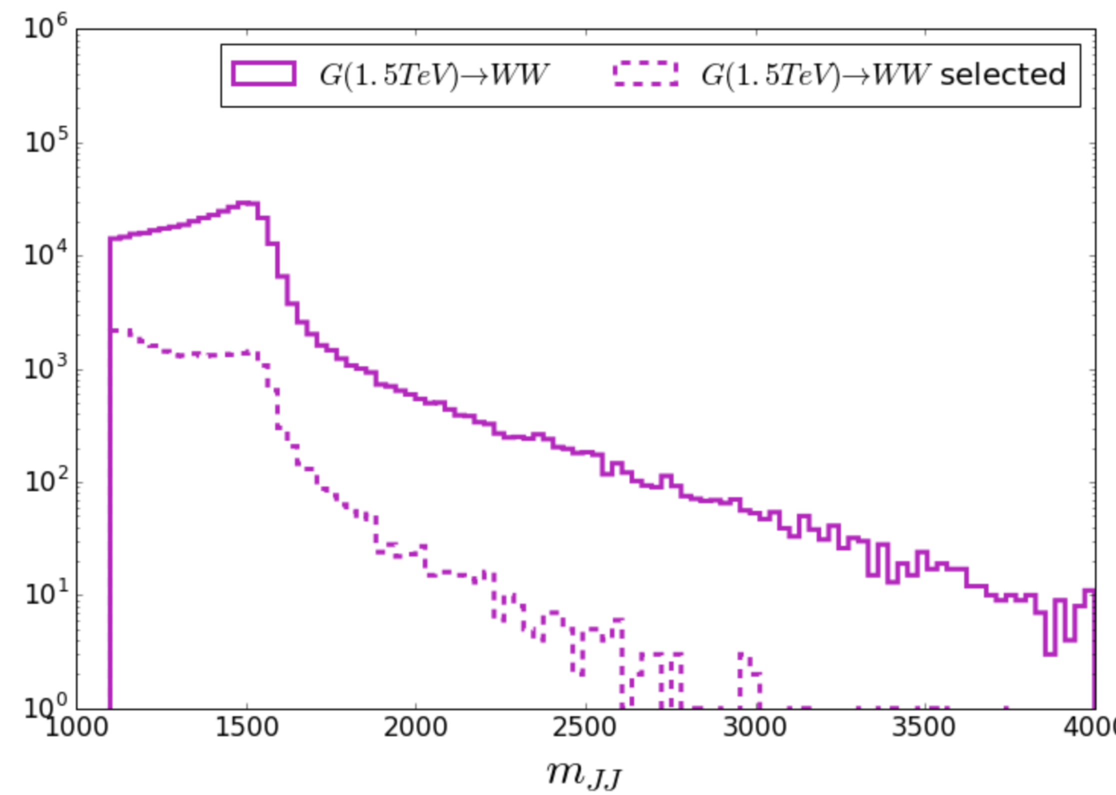
$L > L_T$

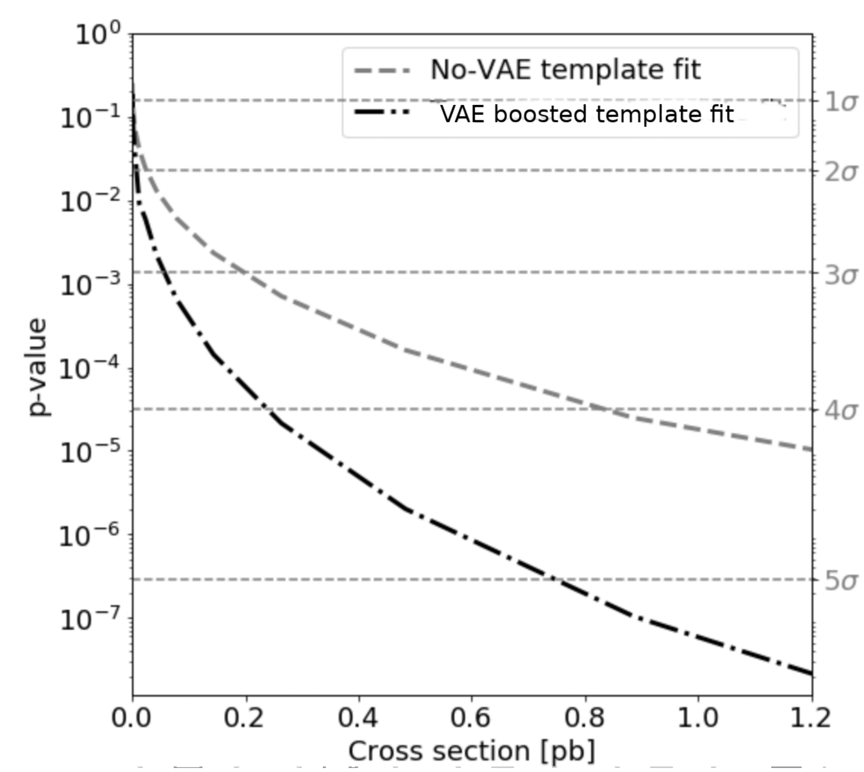


RESULTS

- Comparison between standard dijet search and VAE-assisted search
- Sensitivity boosted from 3σ to 4σ

- Can we apply this technique in the “trigger” algorithm in hardware?



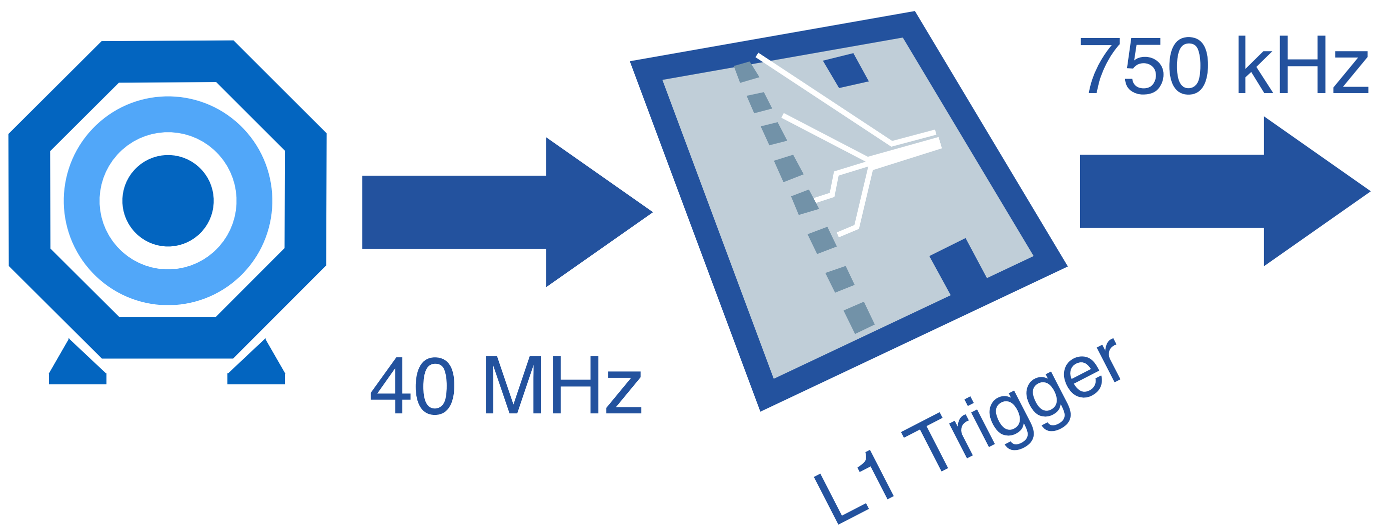


CHAPTER 1: OPPORTUNITIES & CHALLENGES OF GEOMETRIC DEEP LEARNING

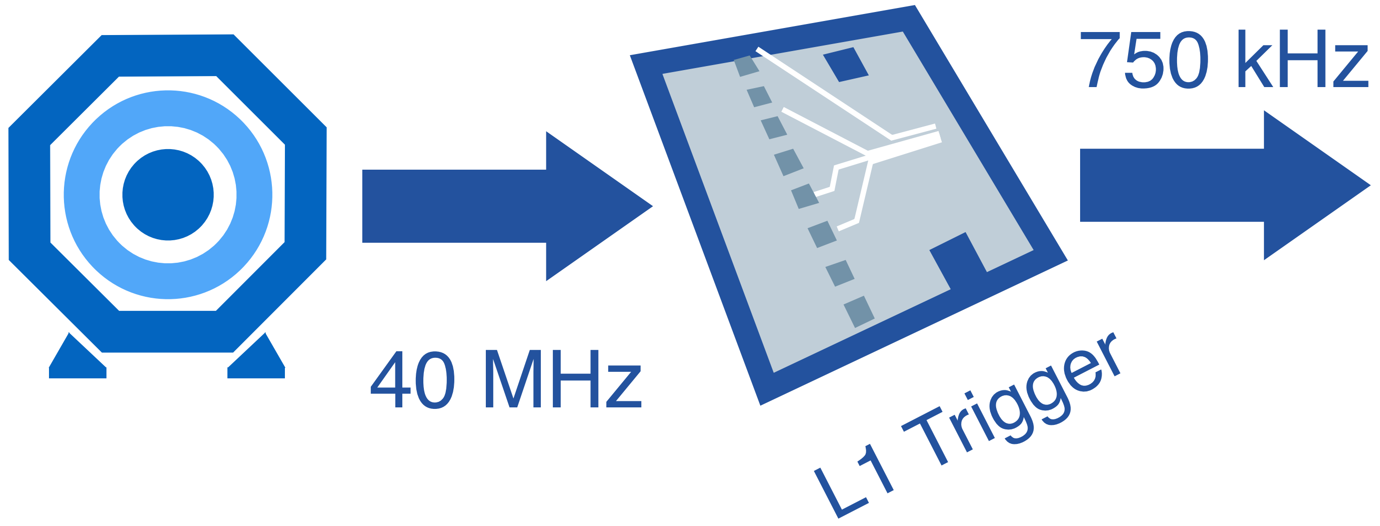
CHAPTER 2: UNSUPERVISED ANOMALY DETECTION FOR NEW PHYSICS

CHAPTER 3: DEEP LEARNING IN THE TRIGGER

UPGRADED LEVEL-1 TRIGGER

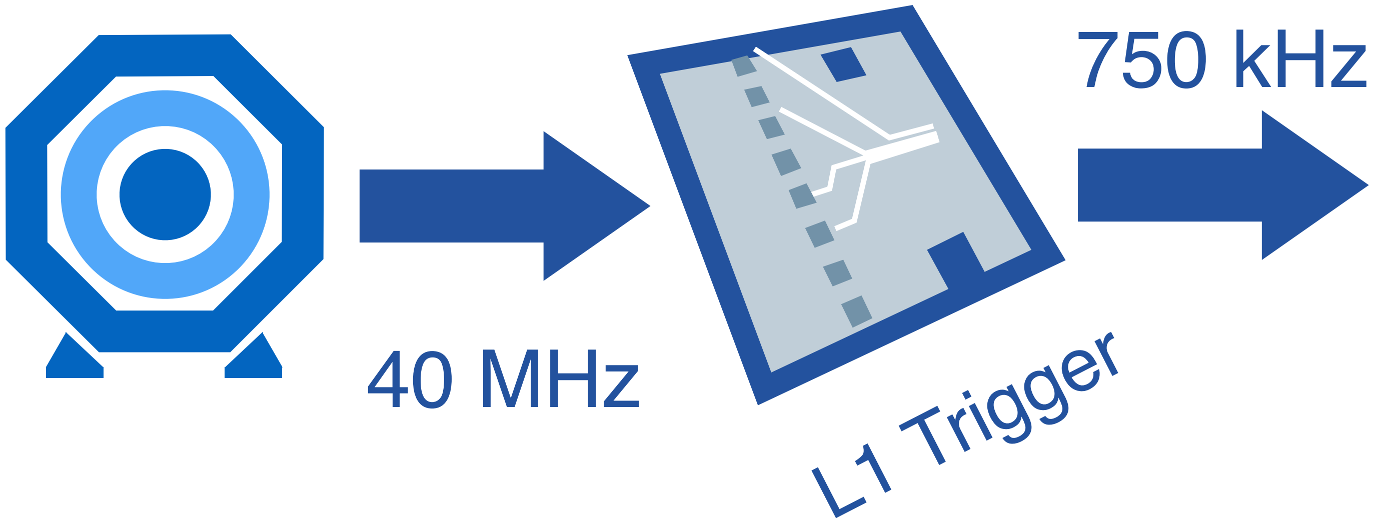


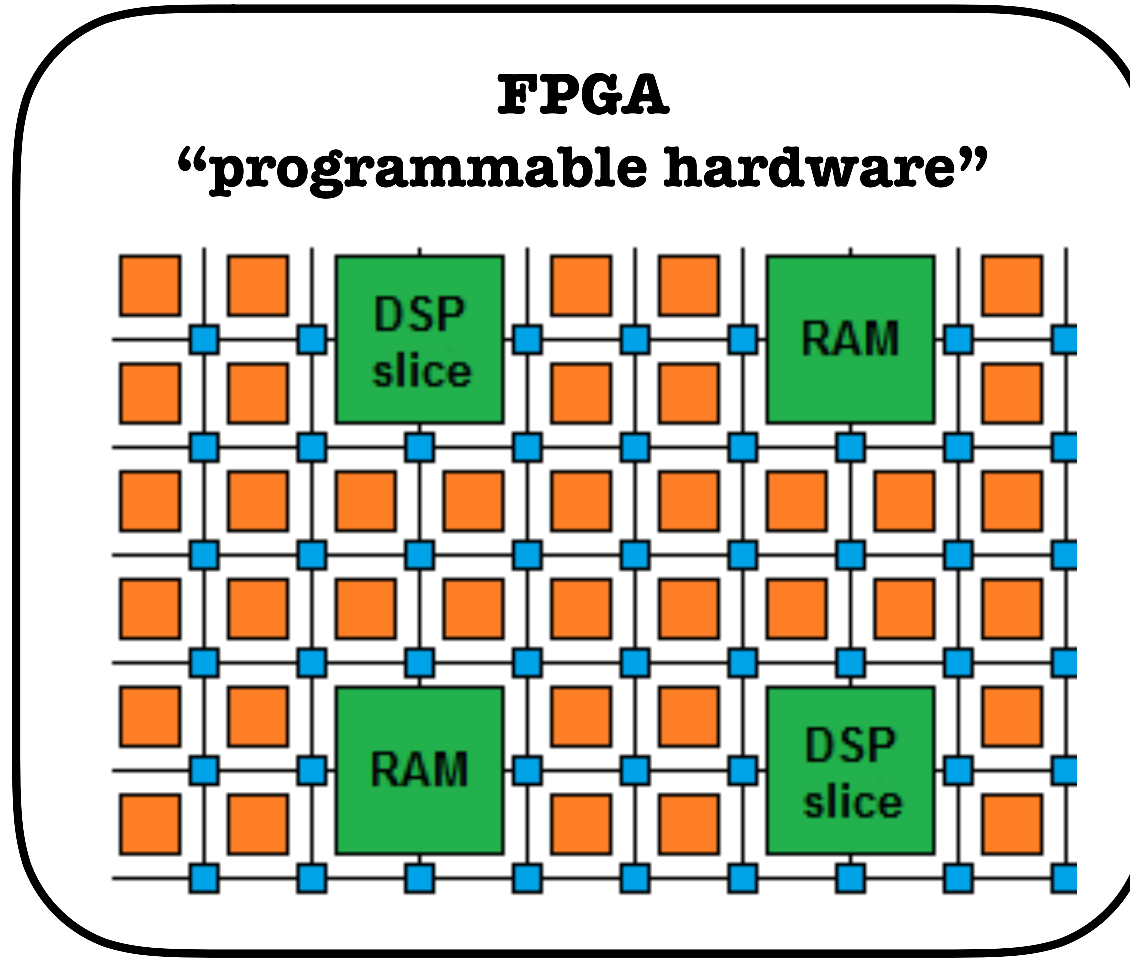
UPGRADED LEVEL-1 TRIGGER



- Level-1 Trigger:
  40 MHz → 750 kHz
- Reconstruct and filter
  2% of events in ~12 μs

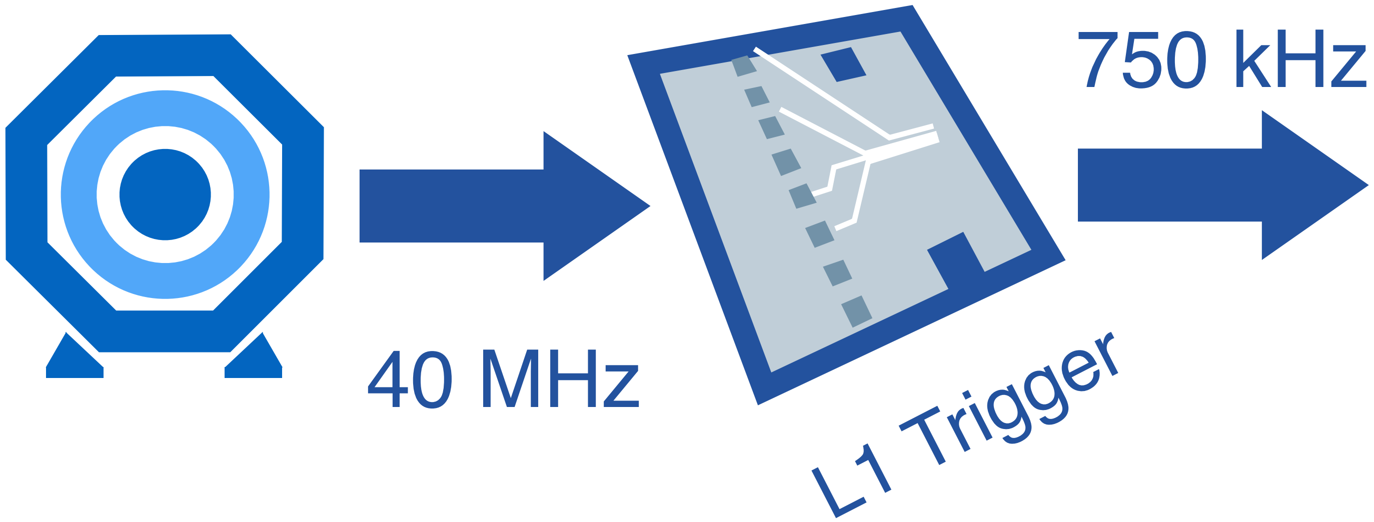
UPGRADED LEVEL-1 TRIGGER

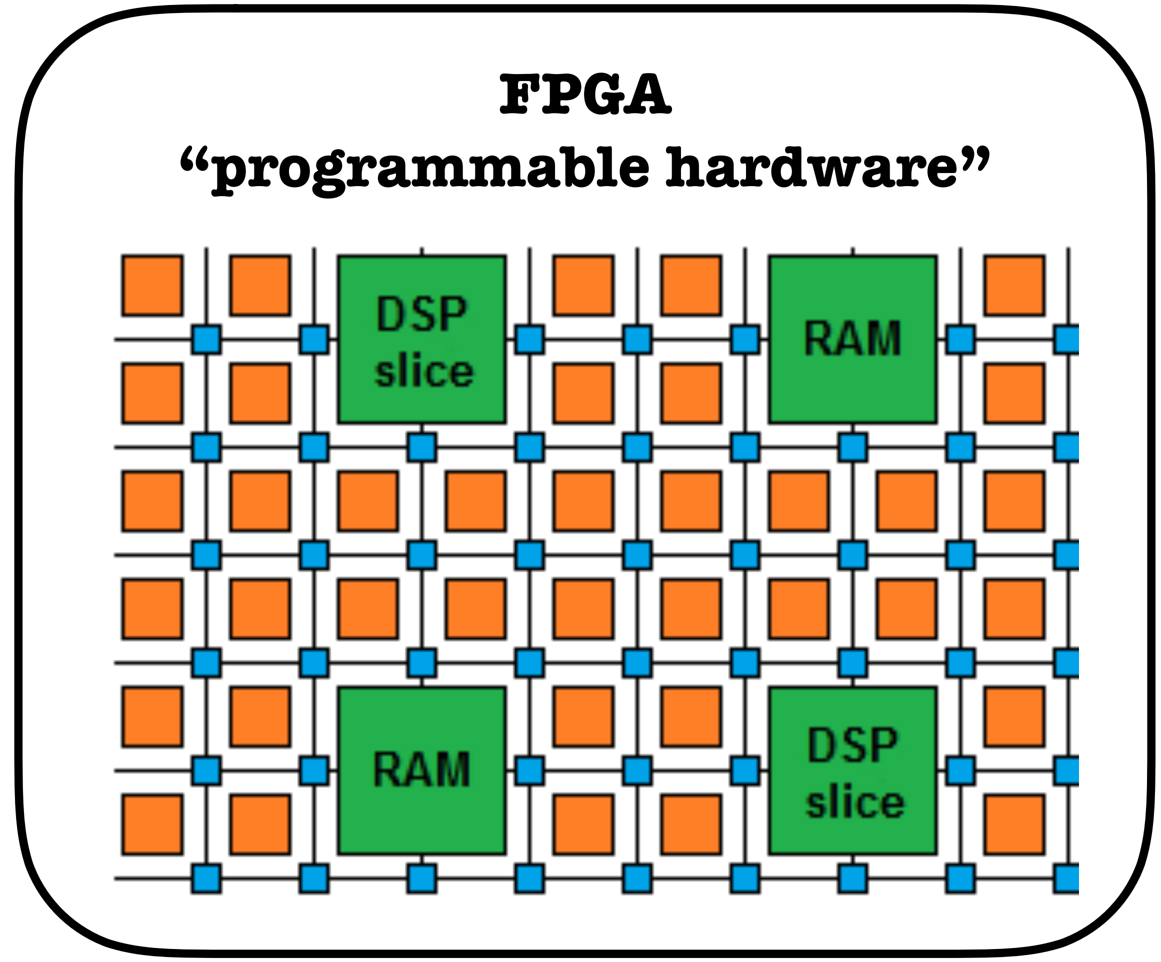




- Level-1 Trigger:
  40 MHz → 750 kHz
- Reconstruct and filter
  2% of events in ~12 μs
- Latency necessitates all
  FPGA design

UPGRADED LEVEL-1 TRIGGER

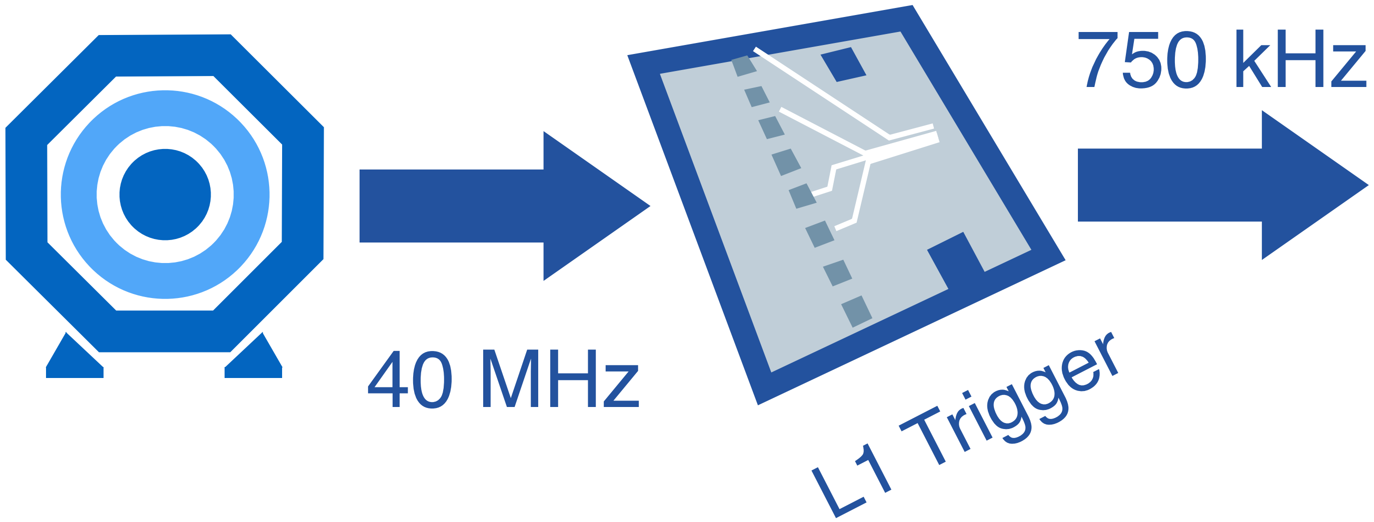


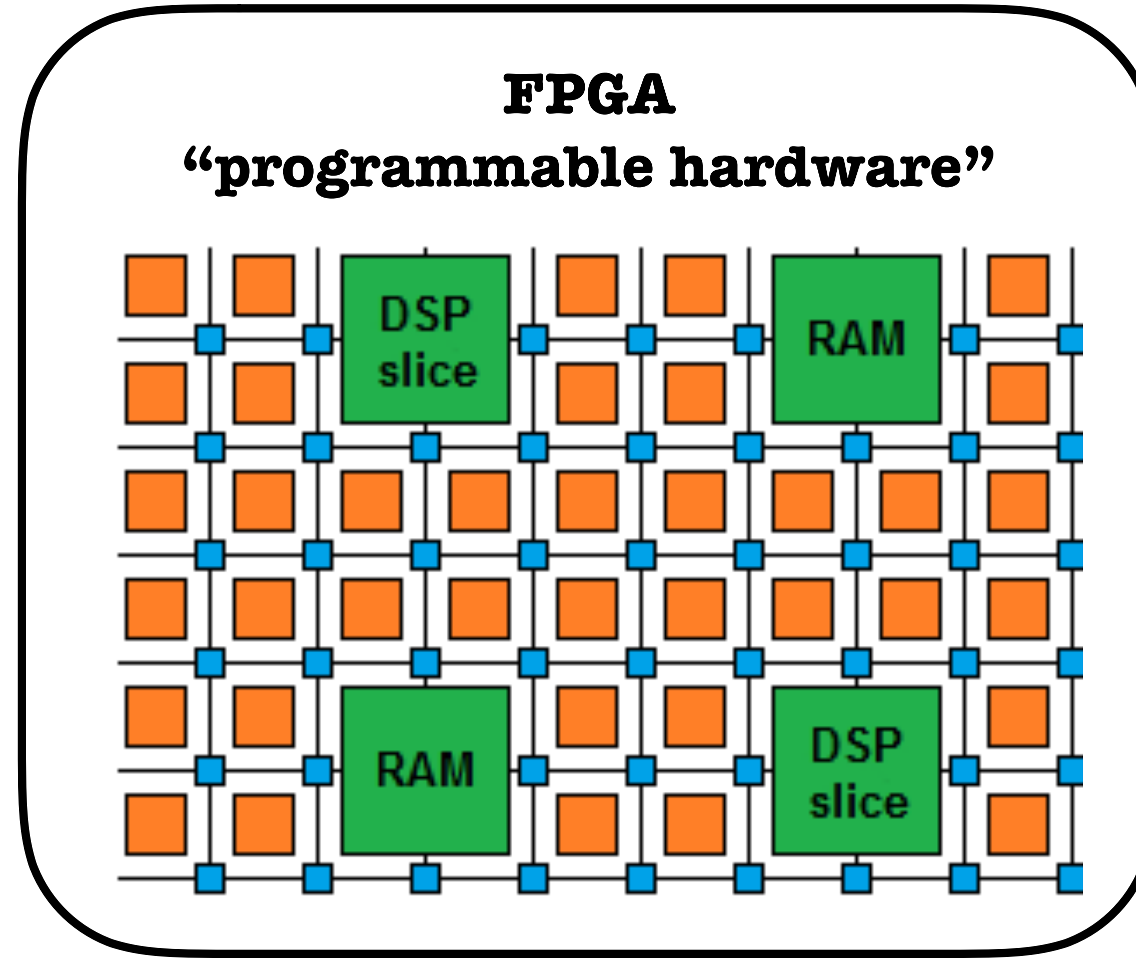


- Level-1 Trigger: 40 MHz → 750 kHz
- Reconstruct and filter 2% of events in ~12 μs
- Latency necessitates all **FPGA** design

- **Pros**: reprogrammable, high throughput, massively parallel, & low power

UPGRADED LEVEL-1 TRIGGER



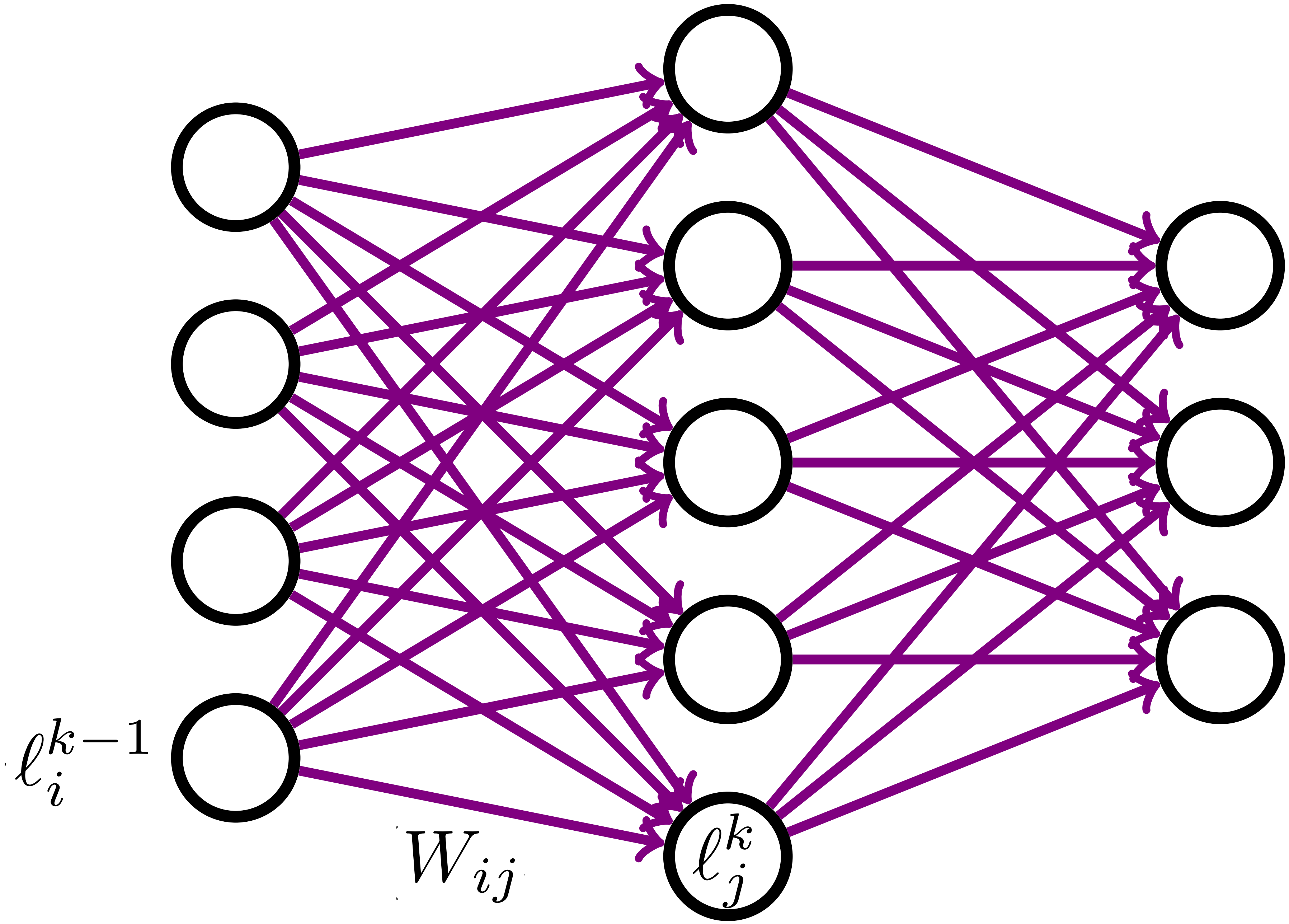


- Level-1 Trigger: 40 MHz → 750 kHz
- Reconstruct and filter 2% of events in ~12 μs
- Latency necessitates all **FPGA** design

- **Pros**: reprogrammable, high throughput, massively parallel, & low power
- **Con**: requires domain knowledge to program

NEURAL NETWORK OPERATIONS

$$\boxed{\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)}$$



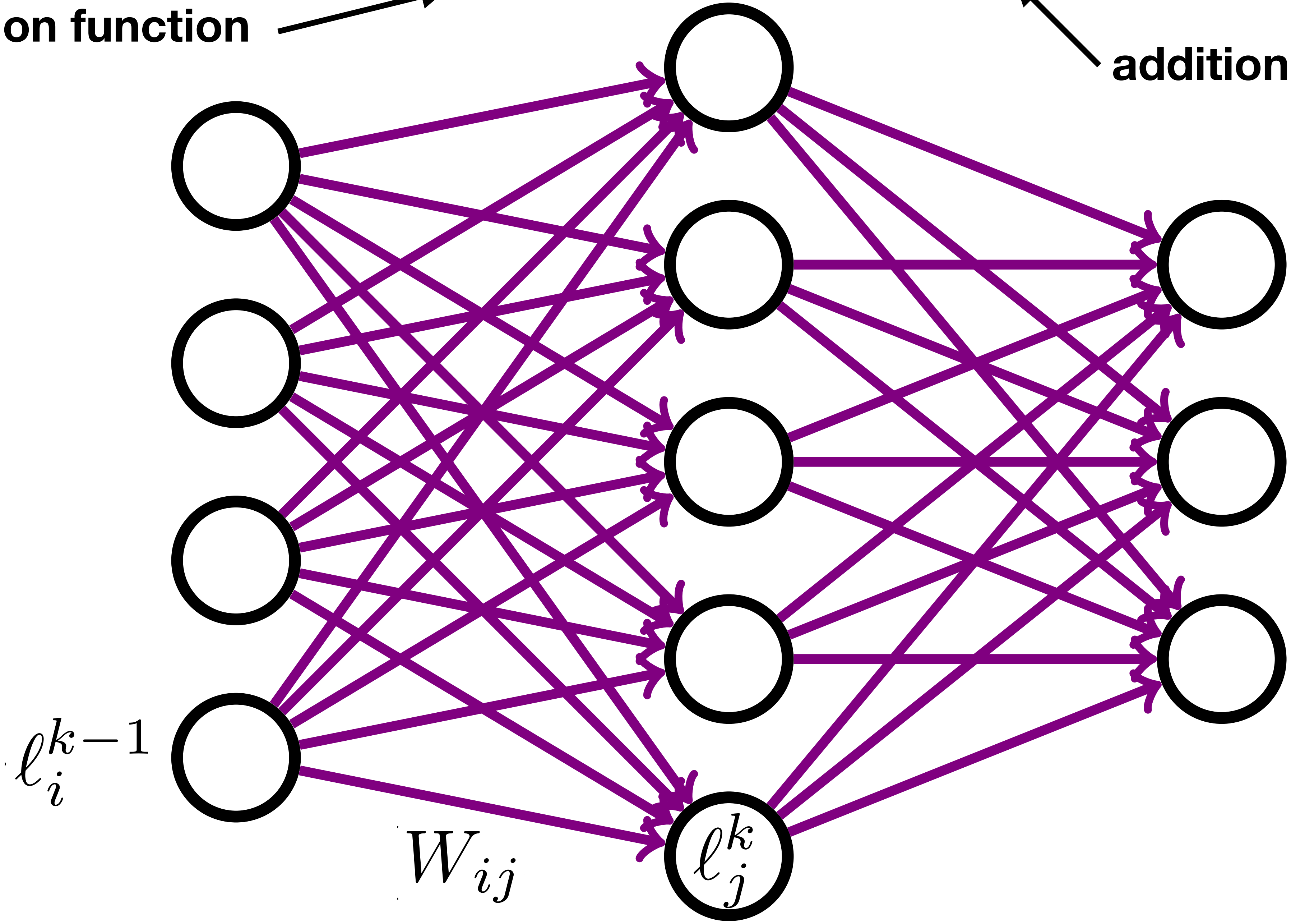
NEURAL NETWORK OPERATIONS

multiplication

$$\boxed{\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)}$$

activation function

addition



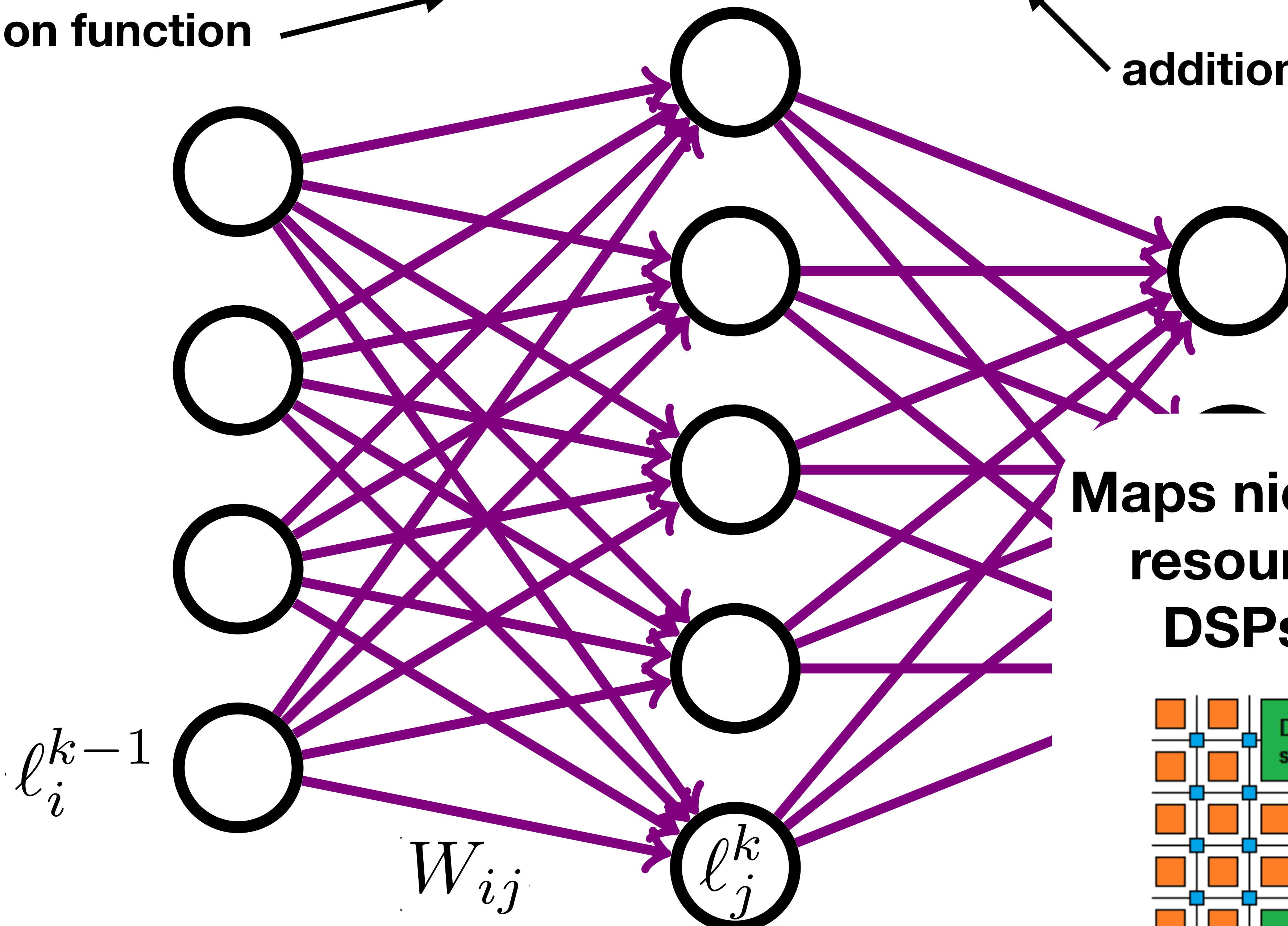
NEURAL NETWORK OPERATIONS

multiplication

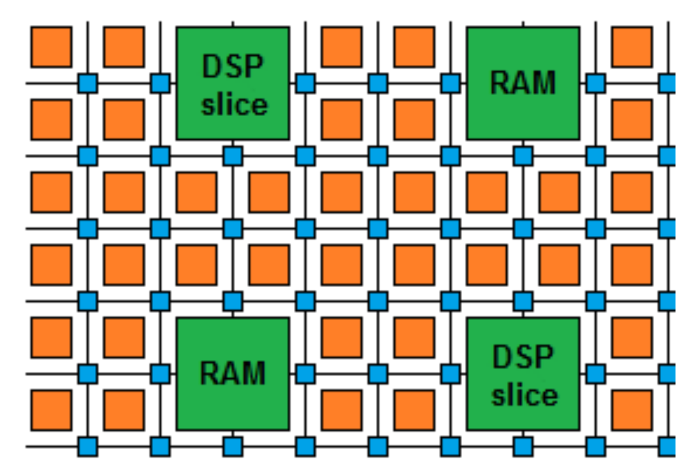
$$\boxed{\ell_j^k = \phi(W_{ij}\ell_i^{k-1} + b_j)}$$

activation function

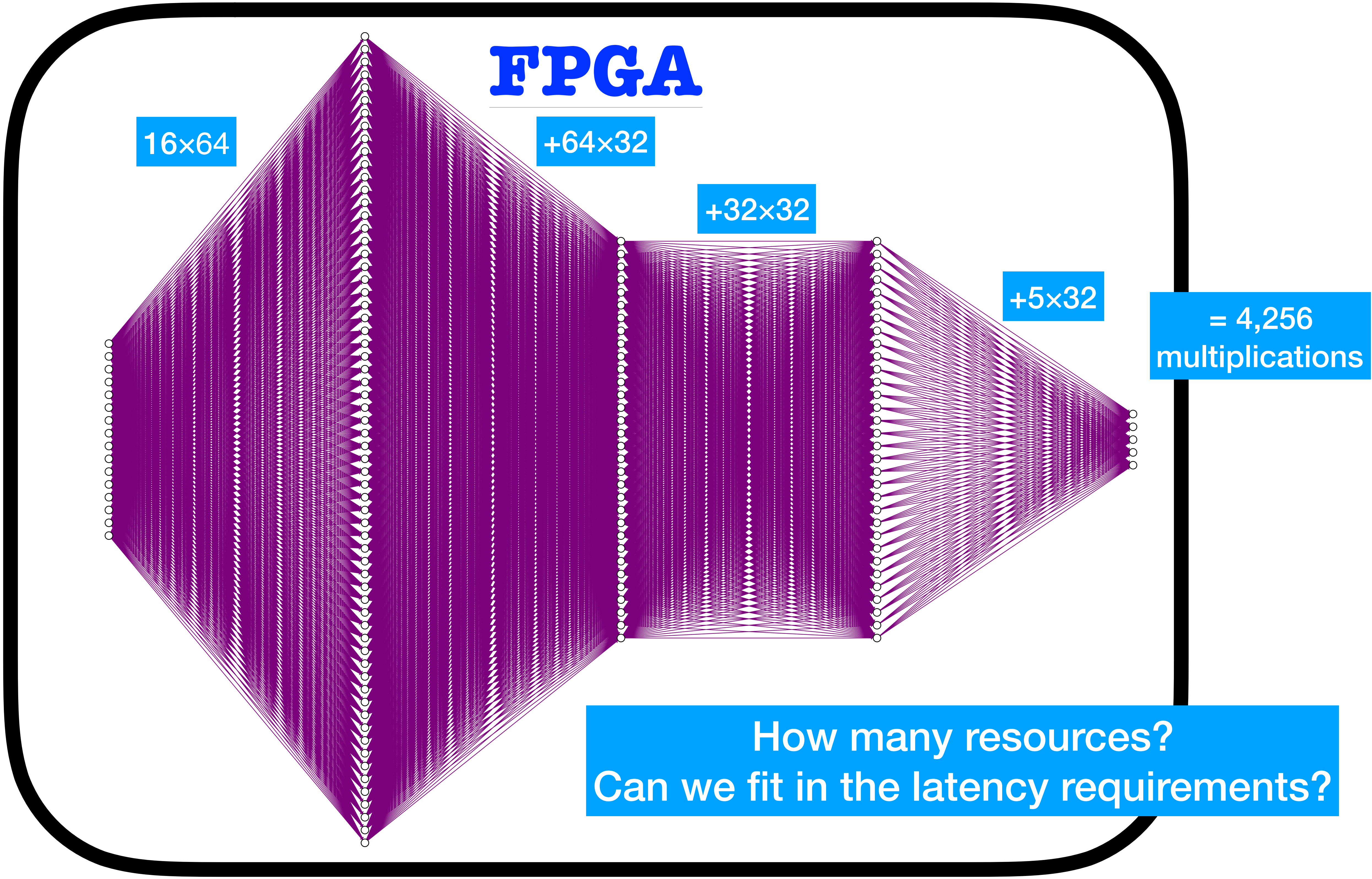
addition



Maps nicely onto FPGA resources: high IO, DSPs, LUTs, etc.



MACHINE LEARNING IN FPGAS?

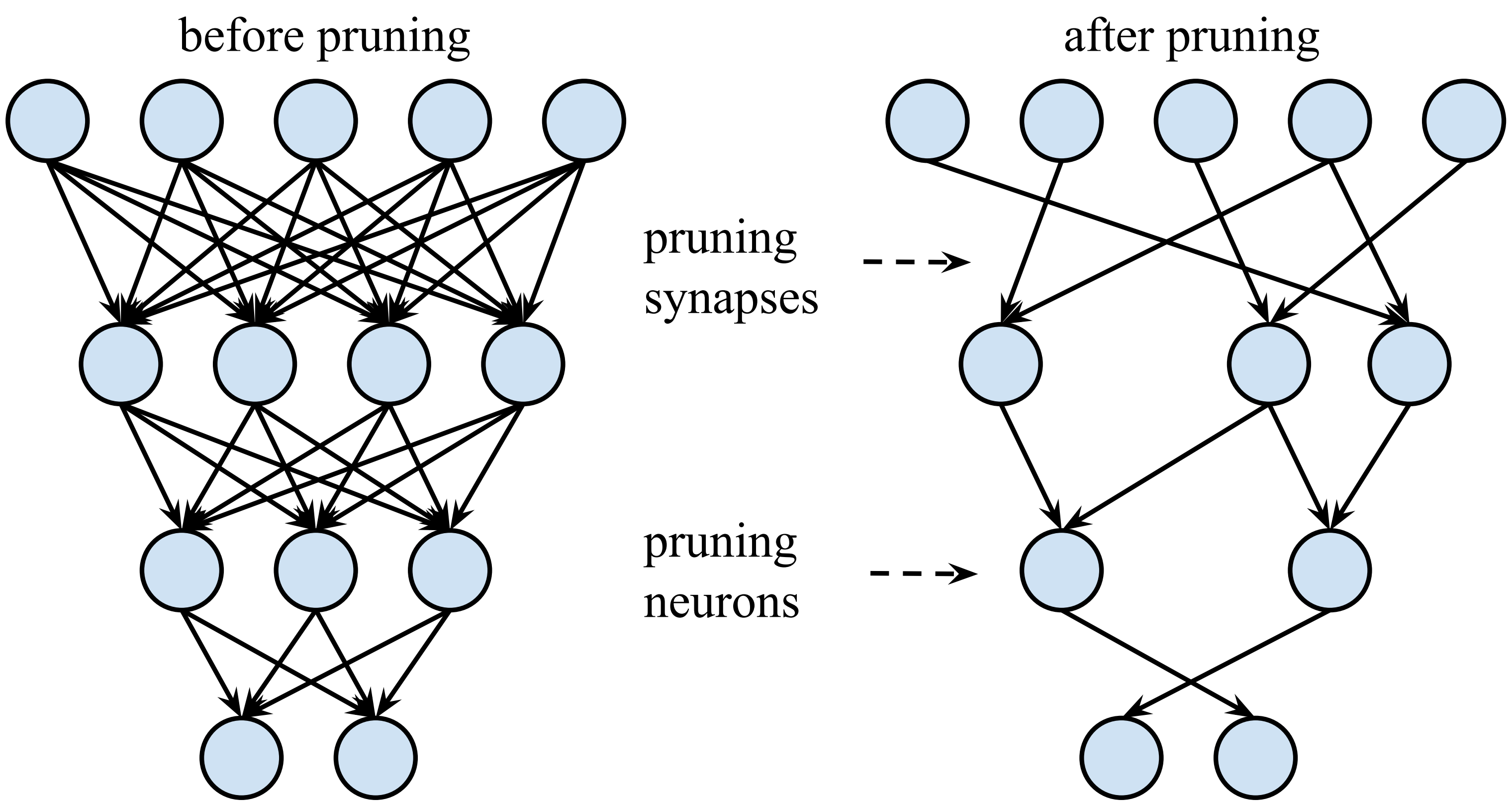


How many resources?
Can we fit in the latency requirements?

(ENERGY) EFFICIENT NEURAL NETWORKS

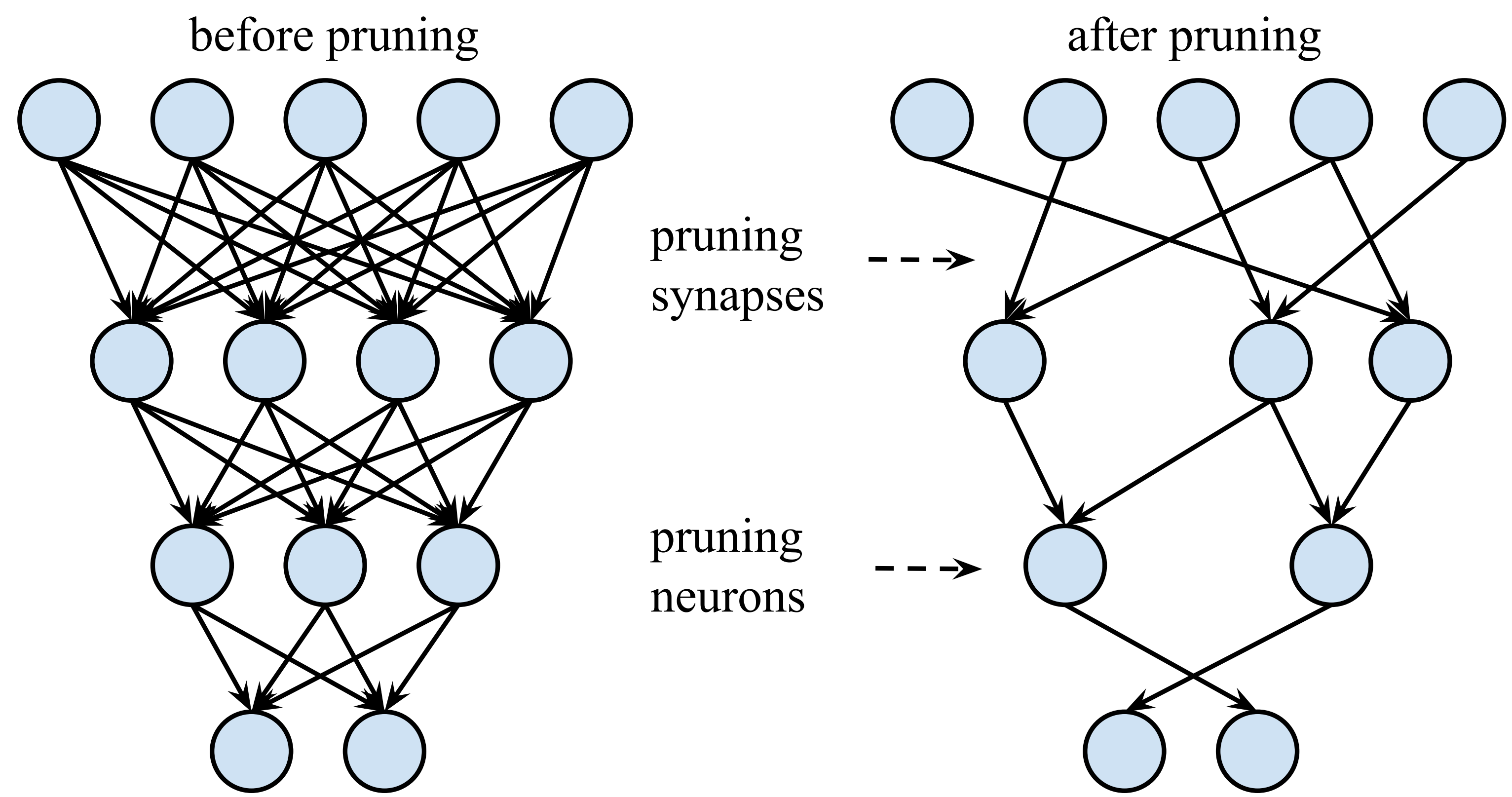
Further reading: arXiv:1510.00149

1. Compression
   - Maintain high performance while removing redundant synapses and neurons



Further reading: arXiv:1510.00149

1. Compression
   - Maintain high performance while removing redundant synapses and neurons

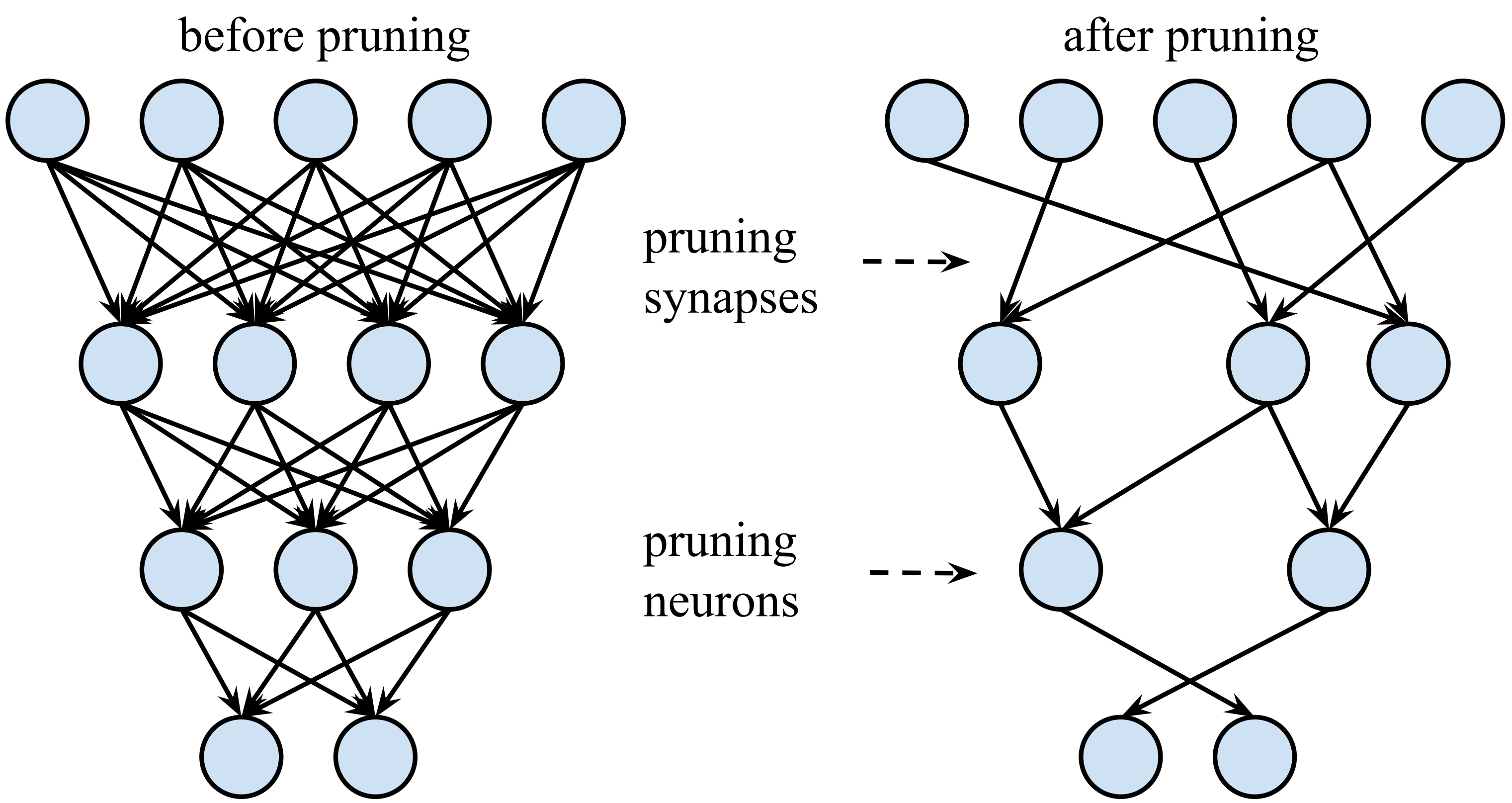


2. Quantization
   - Reduce precision from 32-bit floating point to 20-bit, 8-bit, …

Further reading: arXiv:1510.00149

(ENERGY) EFFICIENT NEURAL NETWORKS

1. Compression
   - Maintain high performance while removing redundant synapses and neurons

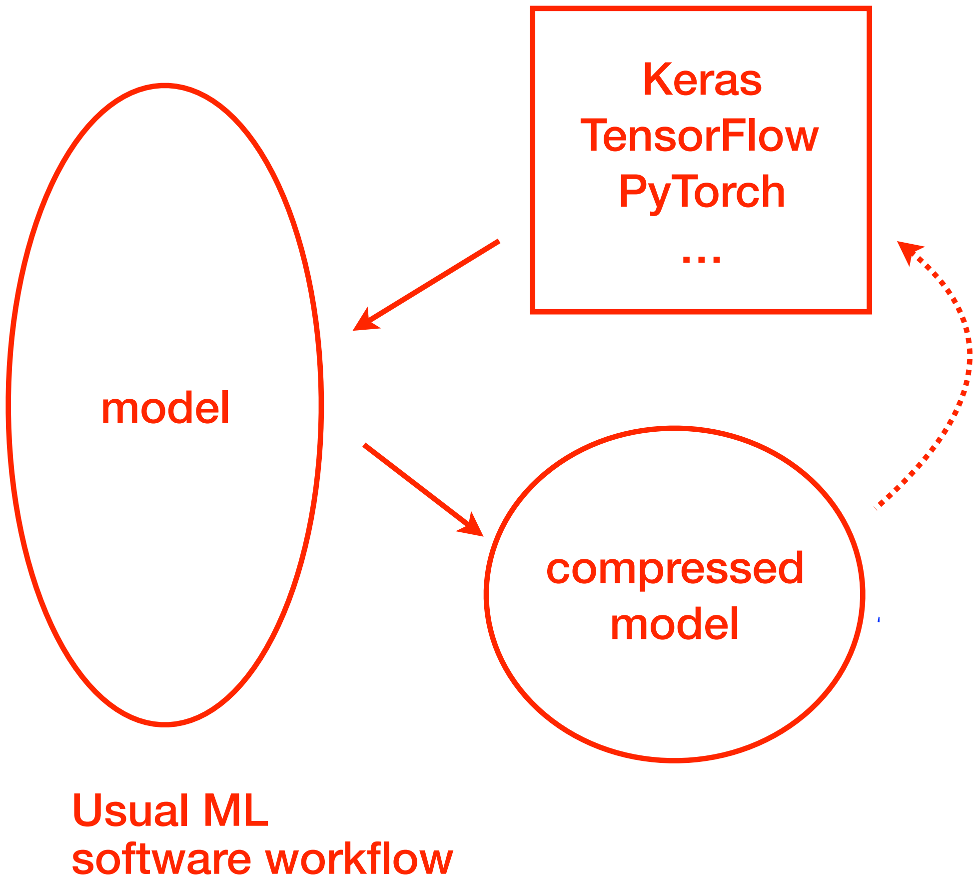


2. Quantization
   - Reduce precision from 32-bit floating point to 20-bit, 8-bit, …
3. Parallelization/Reuse
   - Balance parallelization (how fast) with resources needed (how costly)

Further reading: arXiv:1510.00149

DESIGN EXPLORATION WITH HLS4ML

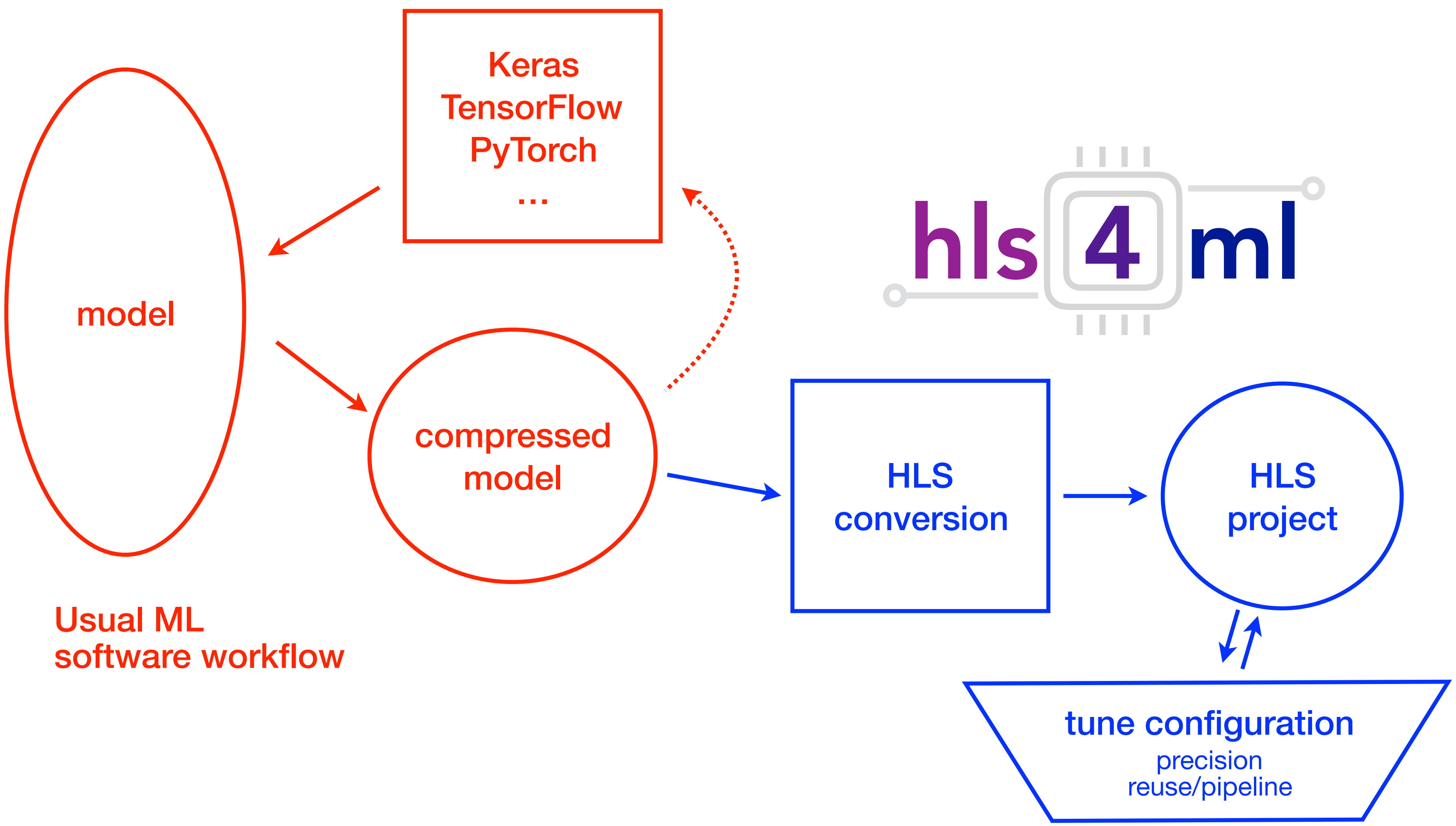
- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



DESIGN EXPLORATION WITH HLS4ML

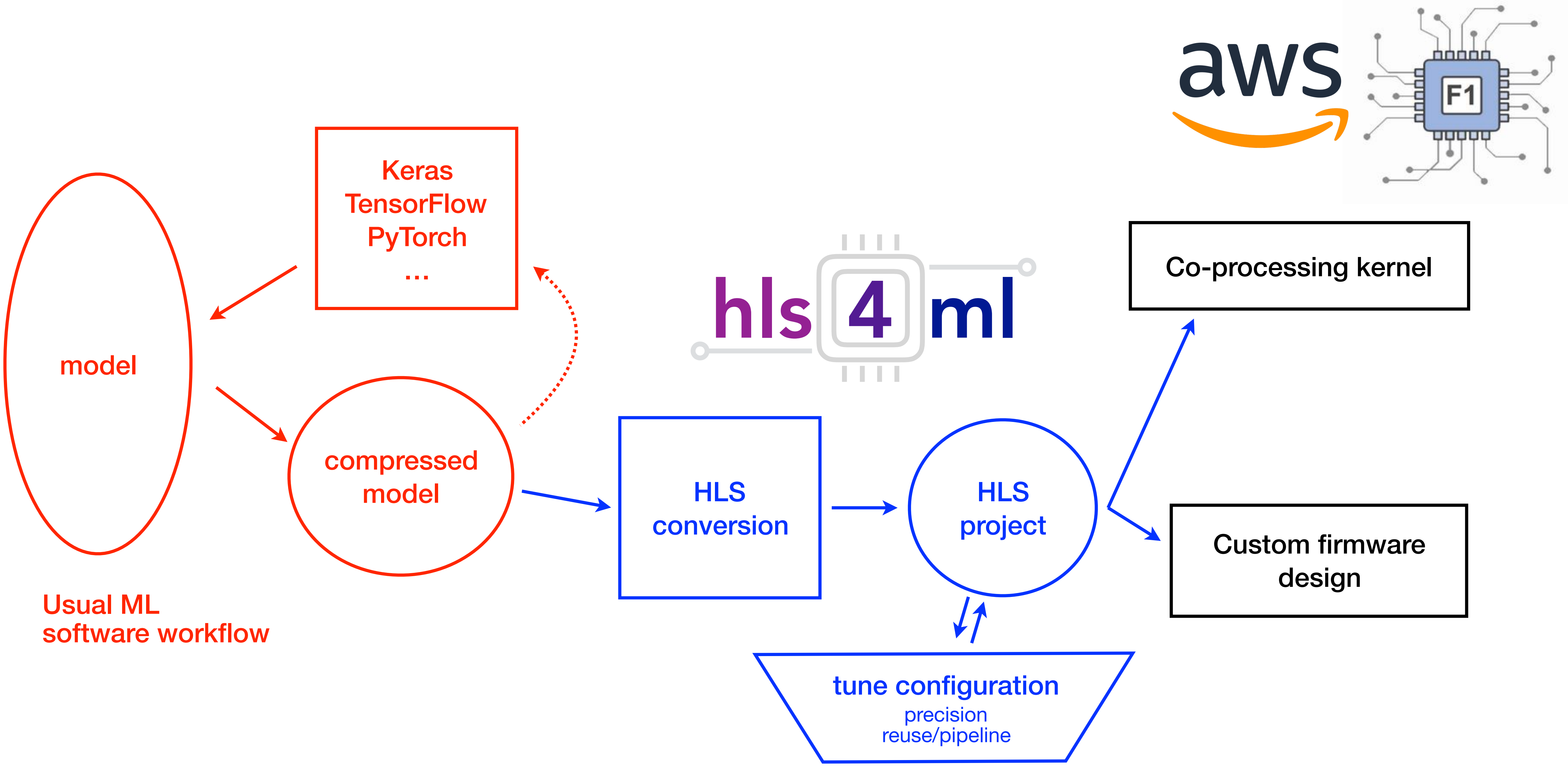
JINST 13 (2018) P07027

- hls4ml for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



DESIGN EXPLORATION WITH HLS4ML

- **hls4ml** for physicists or ML experts to translate **ML algorithms** into **FPGA firmware**



TRANSLATION OF ML MODELS

TRANSLATION OF ML MODELS

Translation →

```
hls4ml convert -c keras-config.yml
```

TRANSLATION OF ML MODELS

Translation

```
hls4ml convert -c keras-config.yml
```

```yaml
KerasJson: keras/KERAS_3layer.json
KerasH5:   keras/KERAS_3layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5
IOType: io_parallel # options: io_serial/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 1
    Strategy: Resource # options: Latency/Resource
```

TRANSLATION OF ML MODELS

Translation

```
hls4ml convert -c keras-config.yml
```

Model

Keras


```
KerasJson: keras/KERAS_3layer.json
KerasH5:   keras/KERAS_3layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5
IOType: io_parallel # options: io_serial/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 1
    Strategy: Resource # options: Latency/Resource
```

Config

TRANSLATION OF ML MODELS

Translation

```
hls4ml convert -c keras-config.yml
```

Model

Keras

HDF5

Config

```
KerasJson: keras/KERAS_3layer.json
KerasH5:   keras/KERAS_3layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5
IOType: io_parallel # options: io_serial/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 1
    Strategy: Resource # options: Latency/Resource
```

- **IOType**: parallel or serial
- **ReuseFactor**: how much to parallelize
- **Precision**: inputs, weights, biases
- **Strategy**:
  - Resource for large NN
  - Latency for small NN (fully pipelined)

TRANSLATION OF ML MODELS

Translation

```
hls4ml convert -c keras-config.yml
```

Model

Keras

HDF5

```
KerasJson: keras/KERAS_3layer.json
KerasH5:   keras/KERAS_3layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xcku115-flvb2104-2-i
ClockPeriod: 5
IOType: io_parallel # options: io_serial/io_parallel
HLSConfig:
  Model:
    Precision: ap_fixed<16,6>
    ReuseFactor: 1
    Strategy: Resource # options: Latency/Resource
```

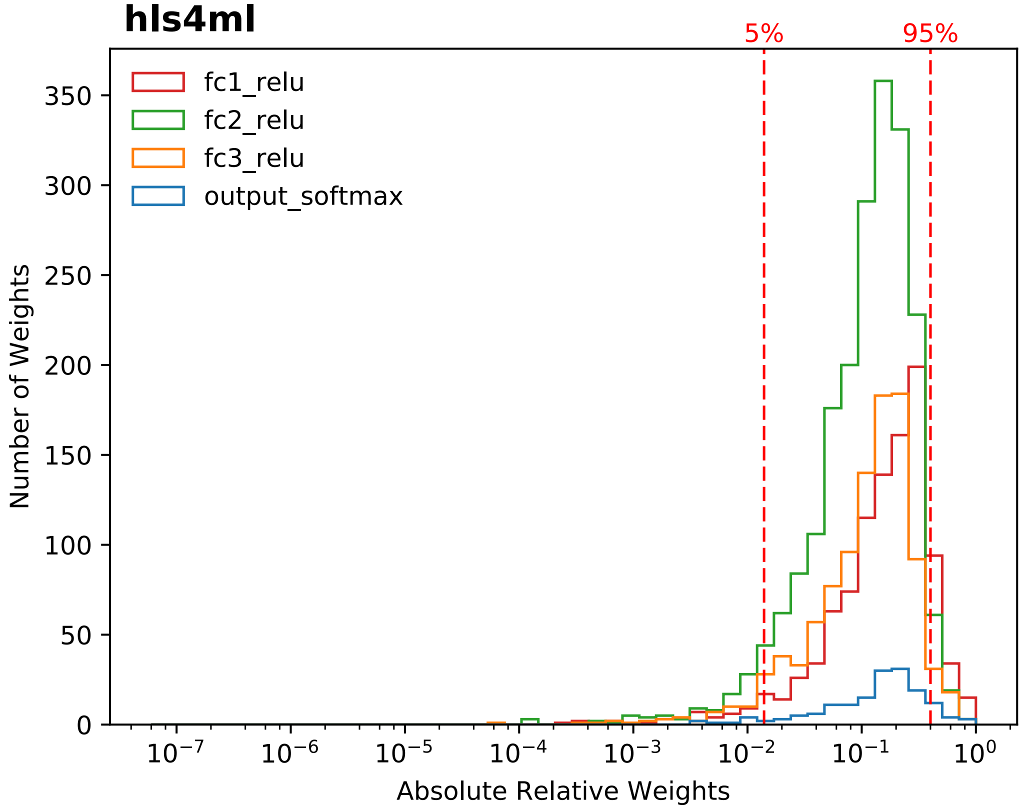
Config

- **IOType**: parallel or serial
- **ReuseFactor**: how much to parallelize
- **Precision**: inputs, weights, biases
- **Strategy**:
  - Resource for large NN
  - Latency for small NN (fully pipelined)

Build HLS project

```
hls4ml build -p my-hls-test -a
```

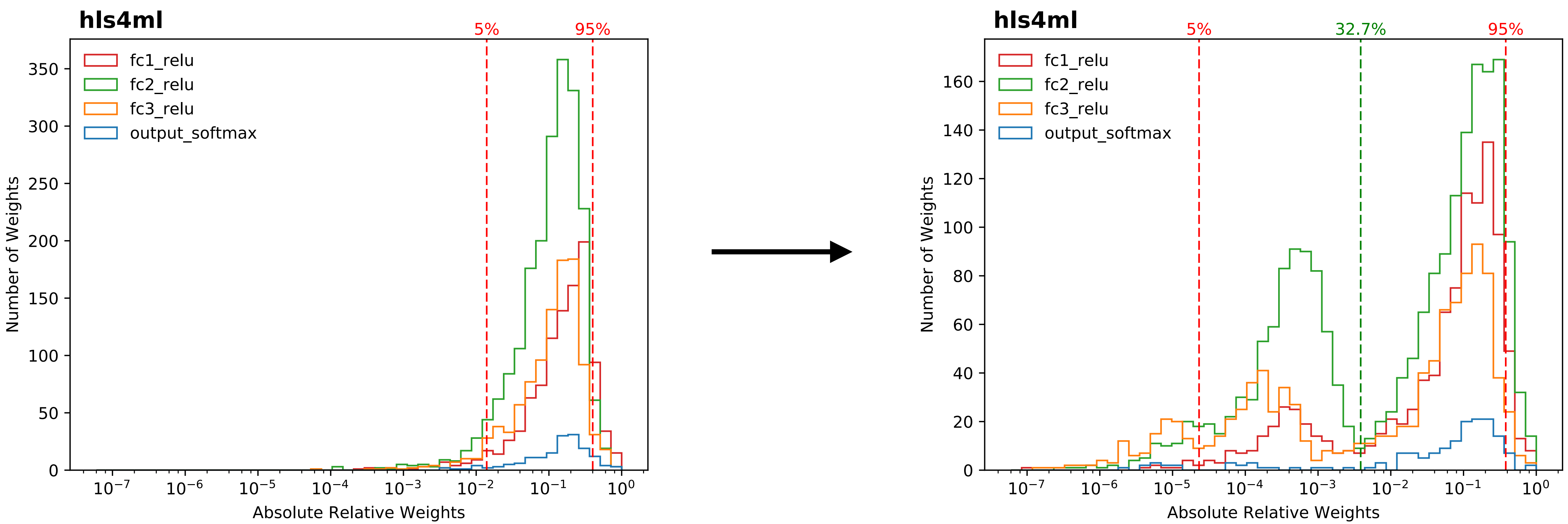
NETWORK TUNING: COMPRESSION



NETWORK TUNING: COMPRESSION

- Train with **L_1 regularization** (down-weights unimportant synapses)

$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \qquad \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

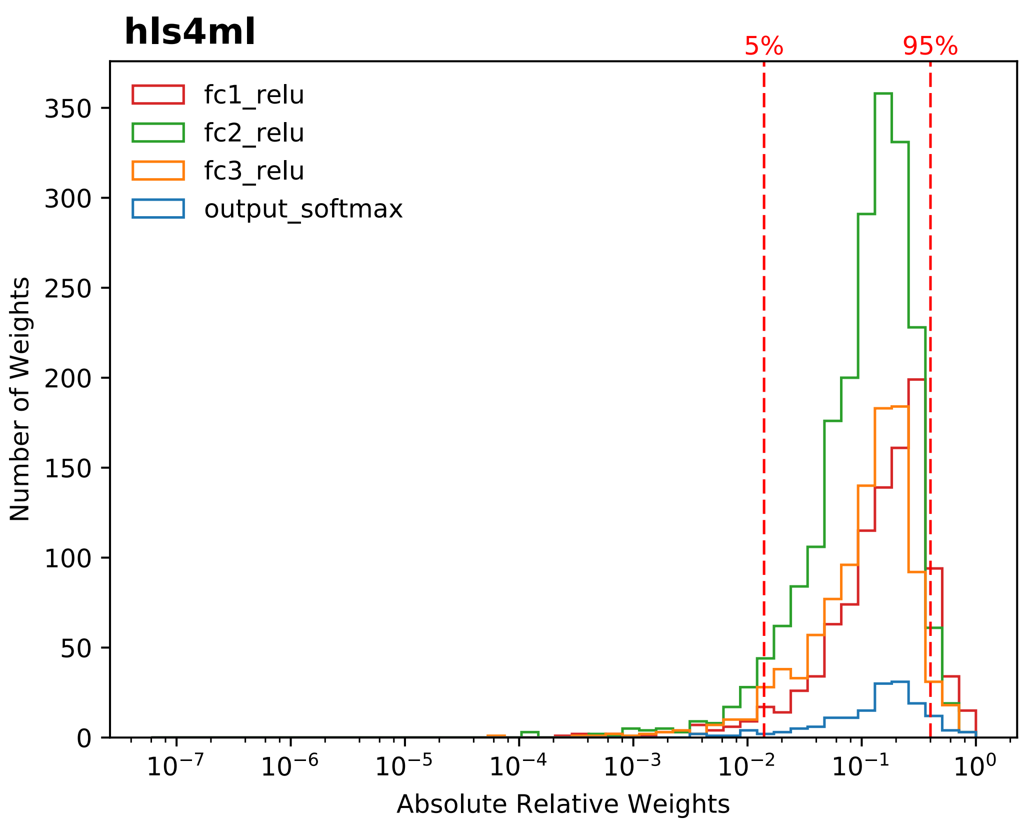


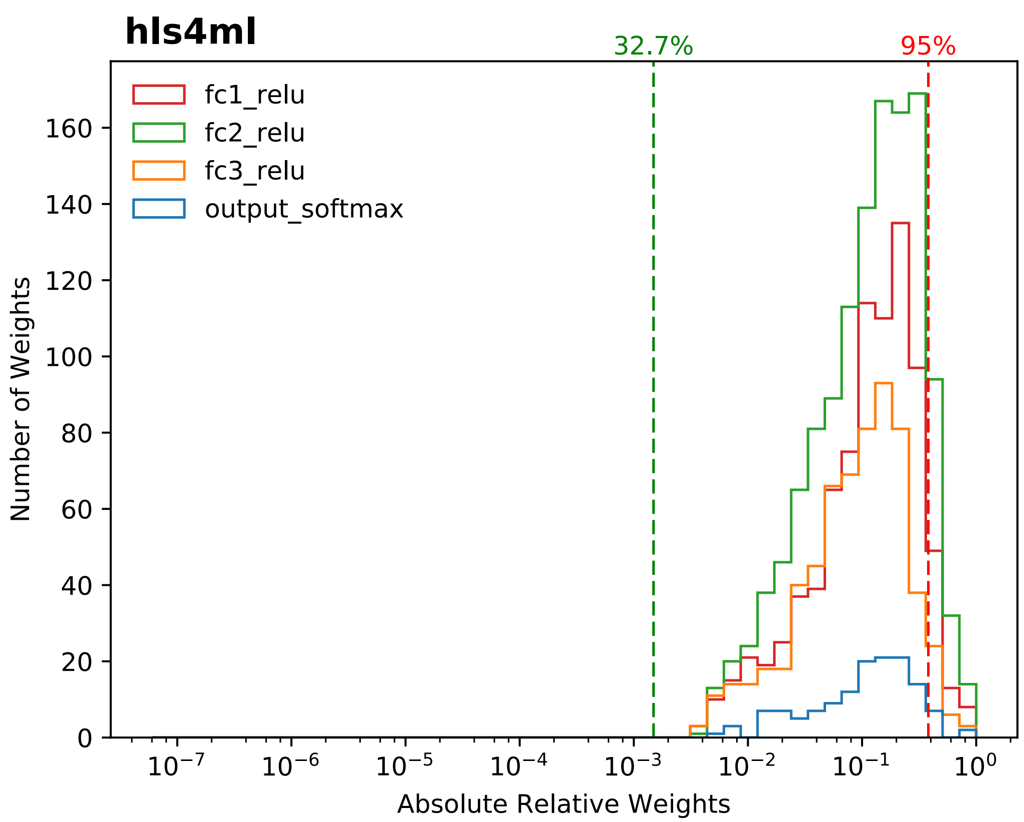
NETWORK TUNING: COMPRESSION

- Train with **L_1 regularization** (down-weights unimportant synapses)

$$L_\lambda(w) = L(w) + \lambda \|w\|_1 \qquad \|w\|_1 = \sum_i |w_i|$$

- Remove **smallest** weights





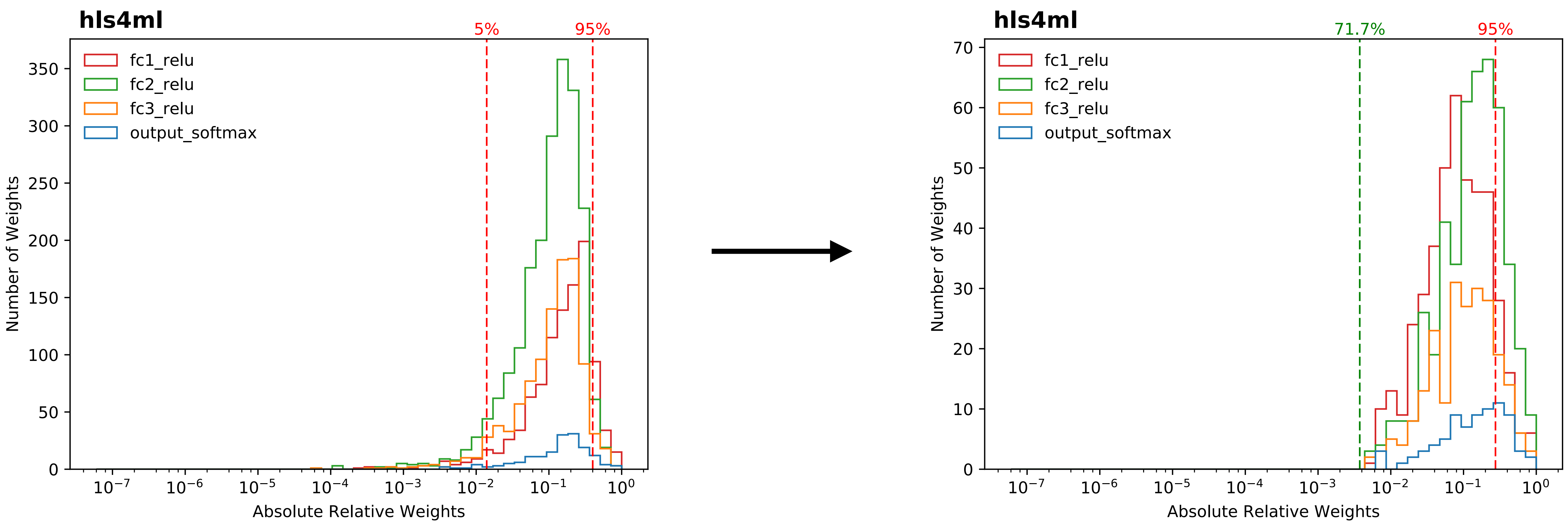
NETWORK TUNING: COMPRESSION

- Train with **L_1 regularization** (down-weights unimportant synapses)

$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \qquad \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

- Remove **smallest** weights
- Iterate

70% REDUCTION OF WEIGHTS WITH NO LOSS IN PERF.



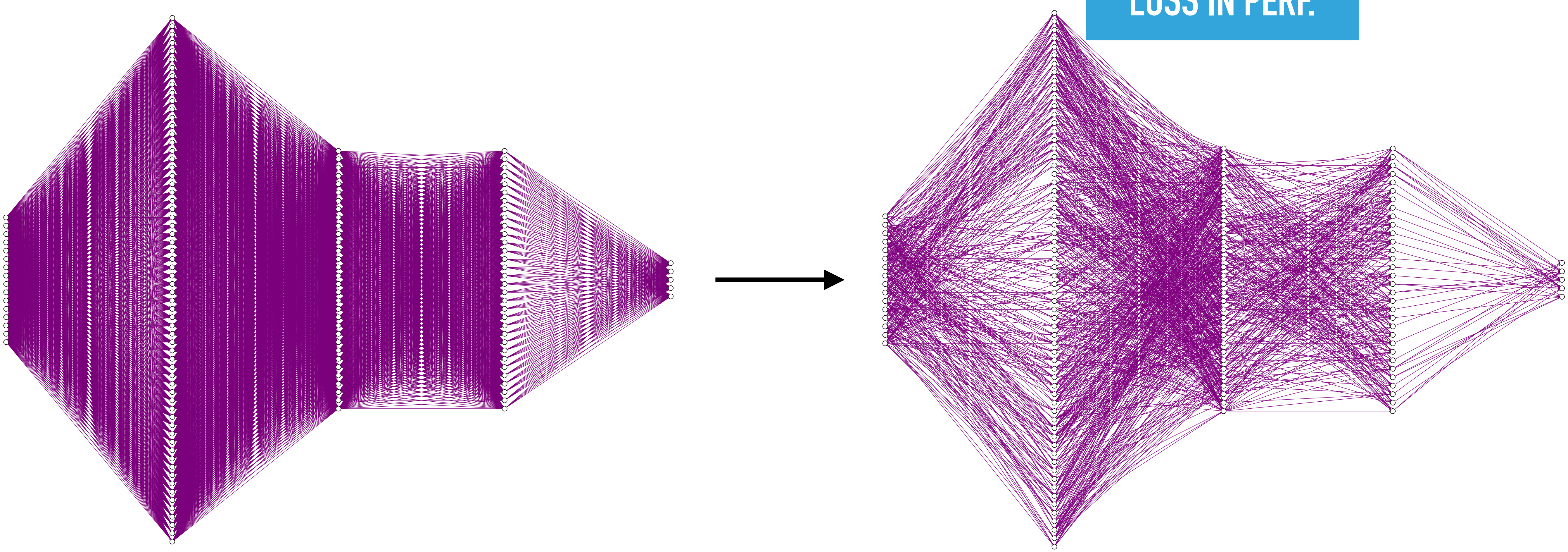
NETWORK TUNING: COMPRESSION

- Train with **L_1 regularization** (down-weights unimportant synapses)

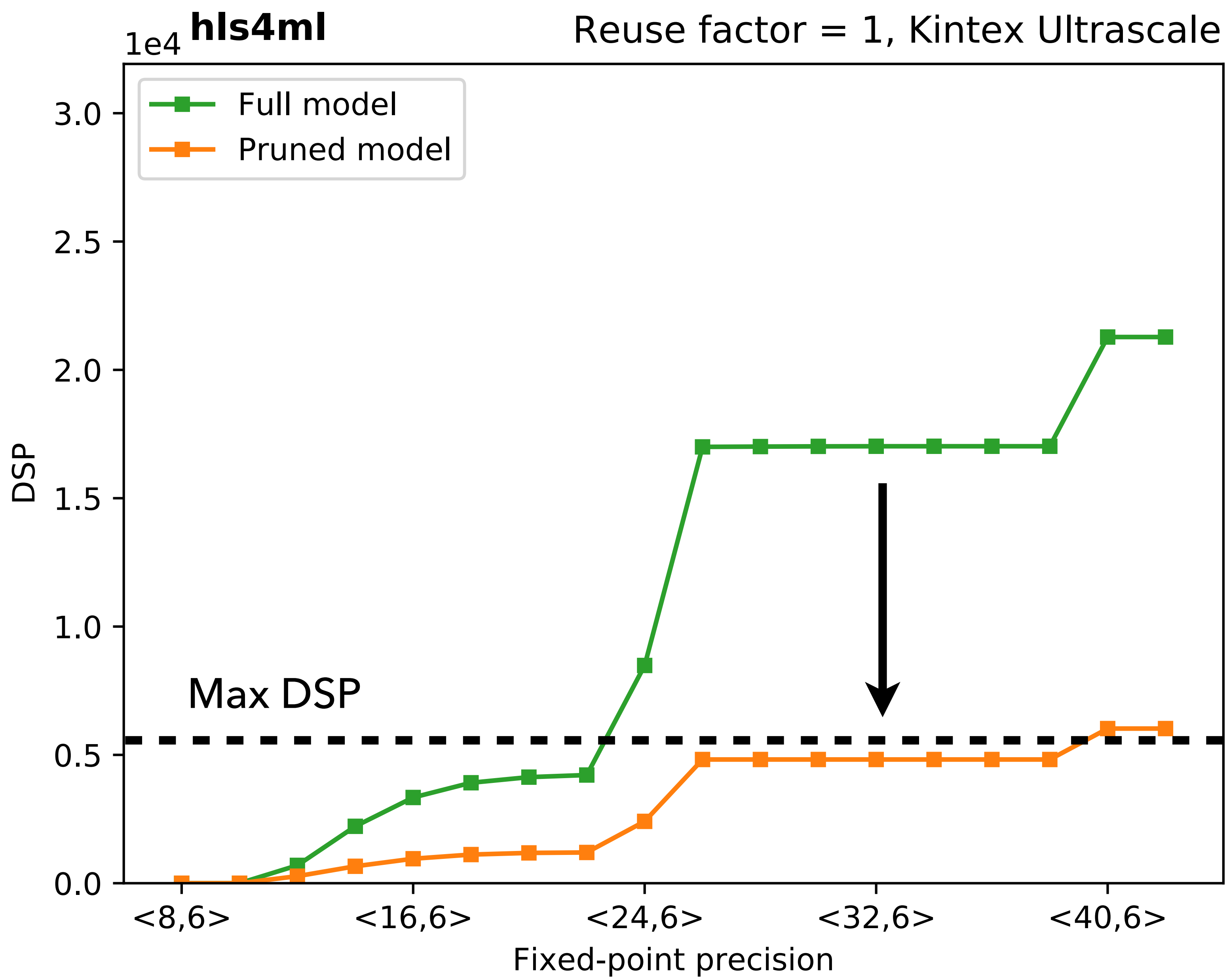
$$L_\lambda(\boldsymbol{w}) = L(\boldsymbol{w}) + \lambda \|\boldsymbol{w}\|_1 \qquad \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

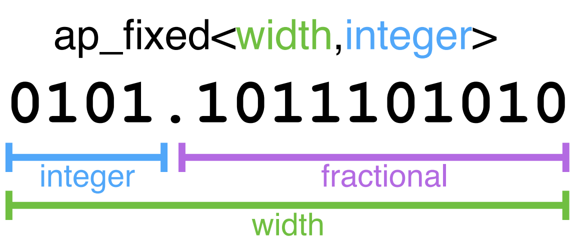
- Remove **smallest** weights
- Iterate

70% REDUCTION OF WEIGHTS WITH NO LOSS IN PERF.

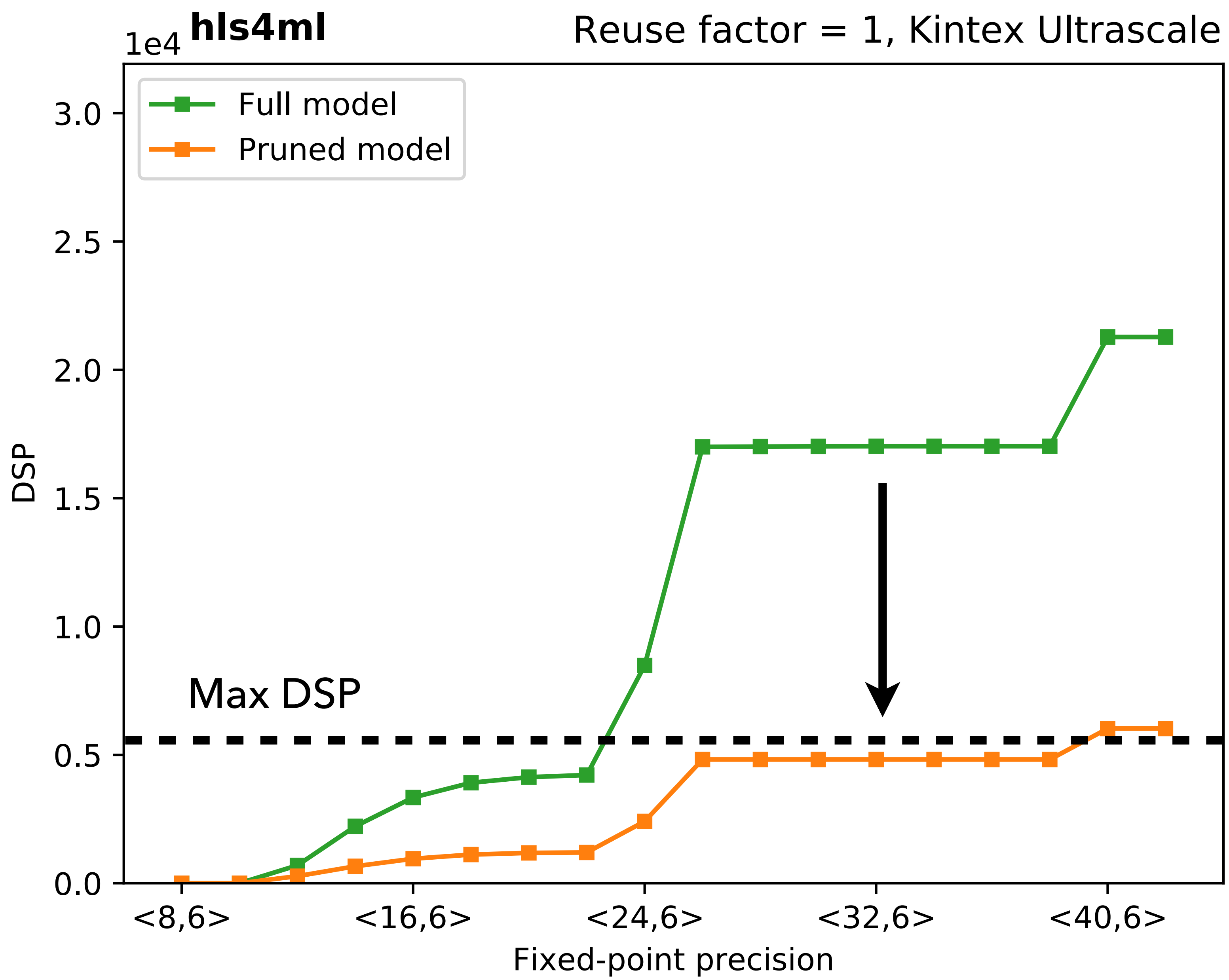


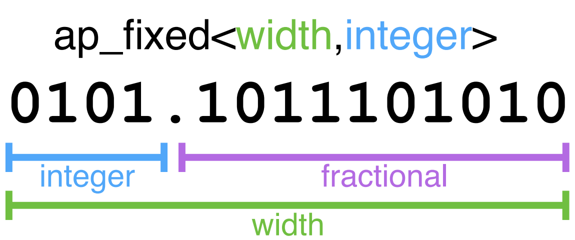
NETWORK TUNING: COMPRESSION & RESOURCES





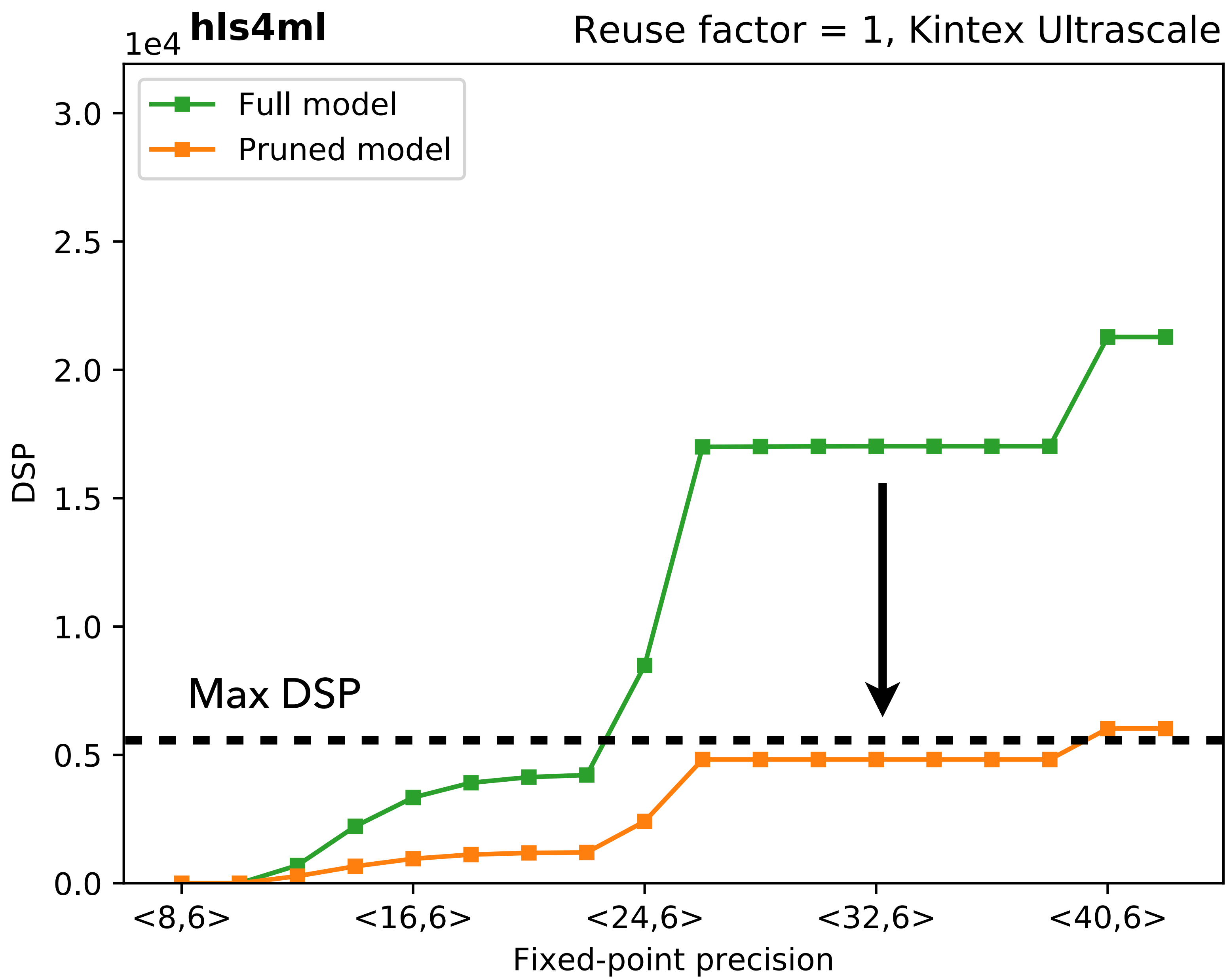
NETWORK TUNING: COMPRESSION & RESOURCES

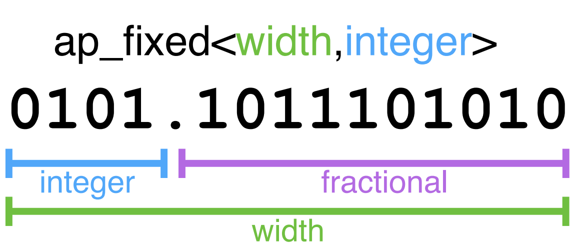




- Big reduction in DSPs (multipliers) with compression

NETWORK TUNING: COMPRESSION & RESOURCES





- Big reduction in DSPs (multipliers) with compression
- Easily fits on 1 FPGA **after compression**

MACHINE LEARNING IN FPGAS

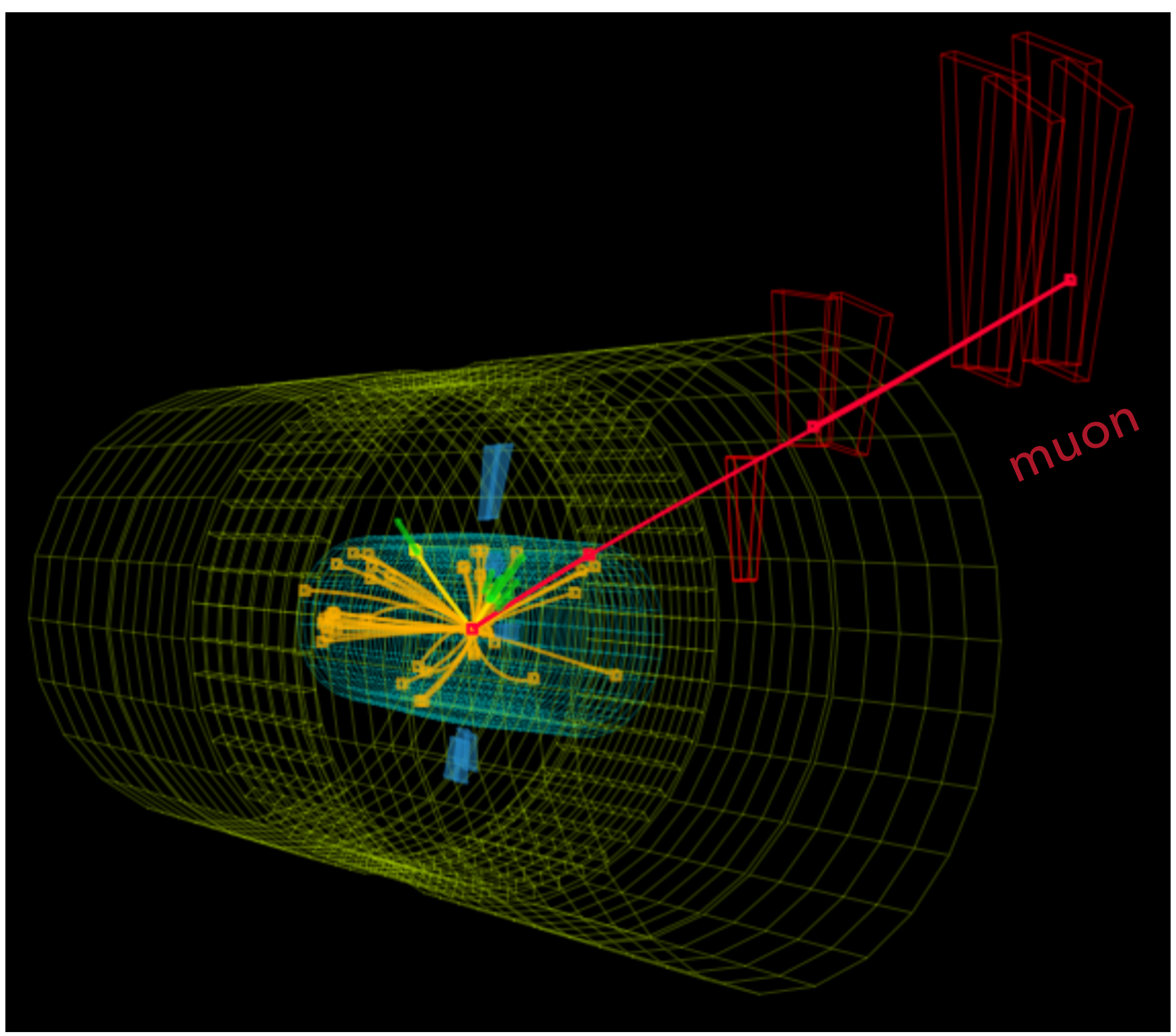
- Inference of ML algorithms possible in **O(100 ns)** on **1 FPGA** with **hls4ml**!

MACHINE LEARNING IN FPGAS

- Inference of ML algorithms possible in **O(100 ns)** on **1 FPGA** with **hls4ml**!
  - Applications across CMS, ATLAS, DUNE, and accelerator controls

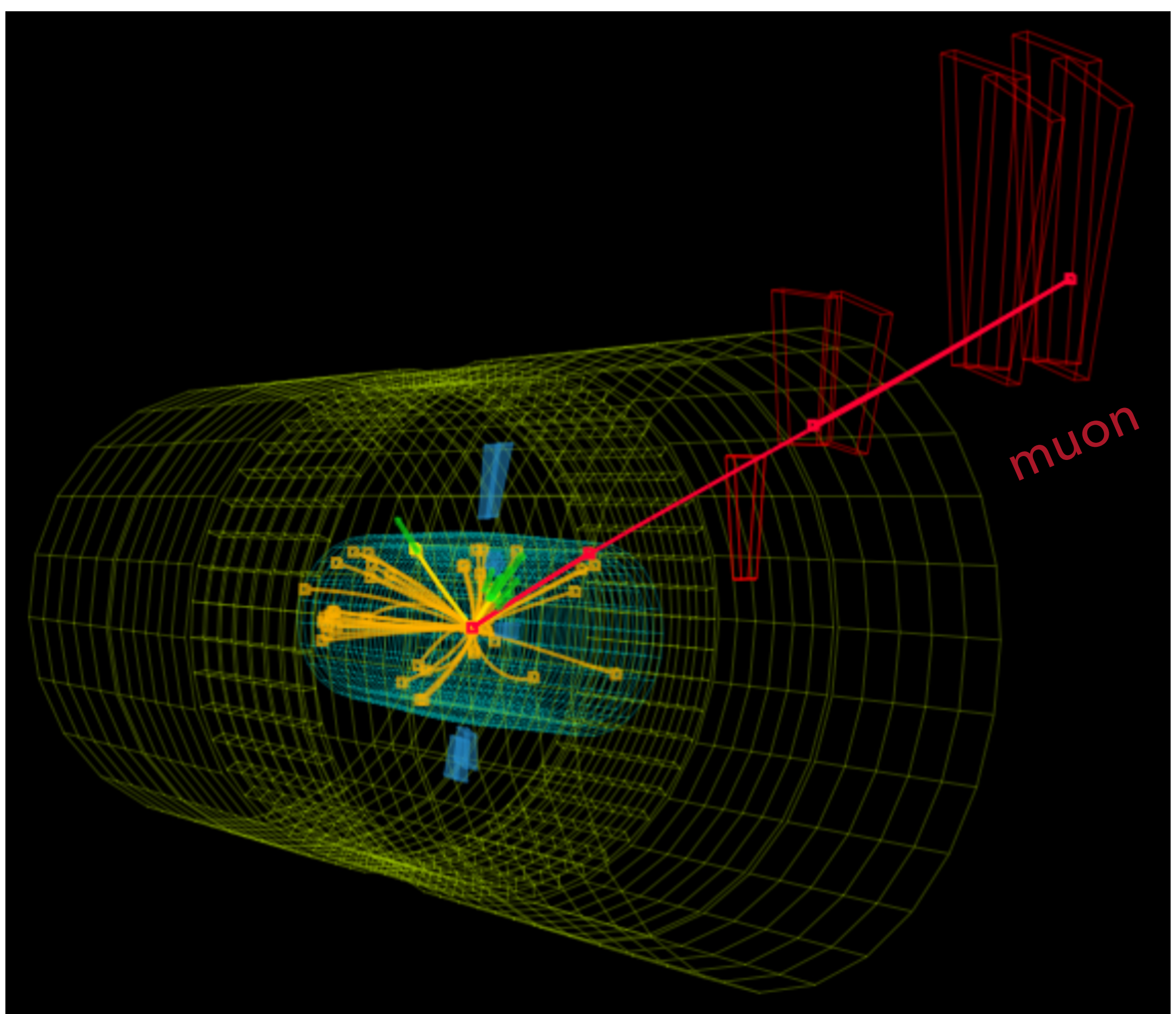
MACHINE LEARNING IN FPGAS

- Inference of ML algorithms possible in **O(100 ns)** on **1 FPGA** with **hls4ml**!
    - Applications across CMS, ATLAS, DUNE, and accelerator controls
    - E.g. muon p_T determination in the CMS endcap with a DNN: **runs in 160 ns** on an FPGA and **reduces** the **fake muon rate** by **up to 80%**

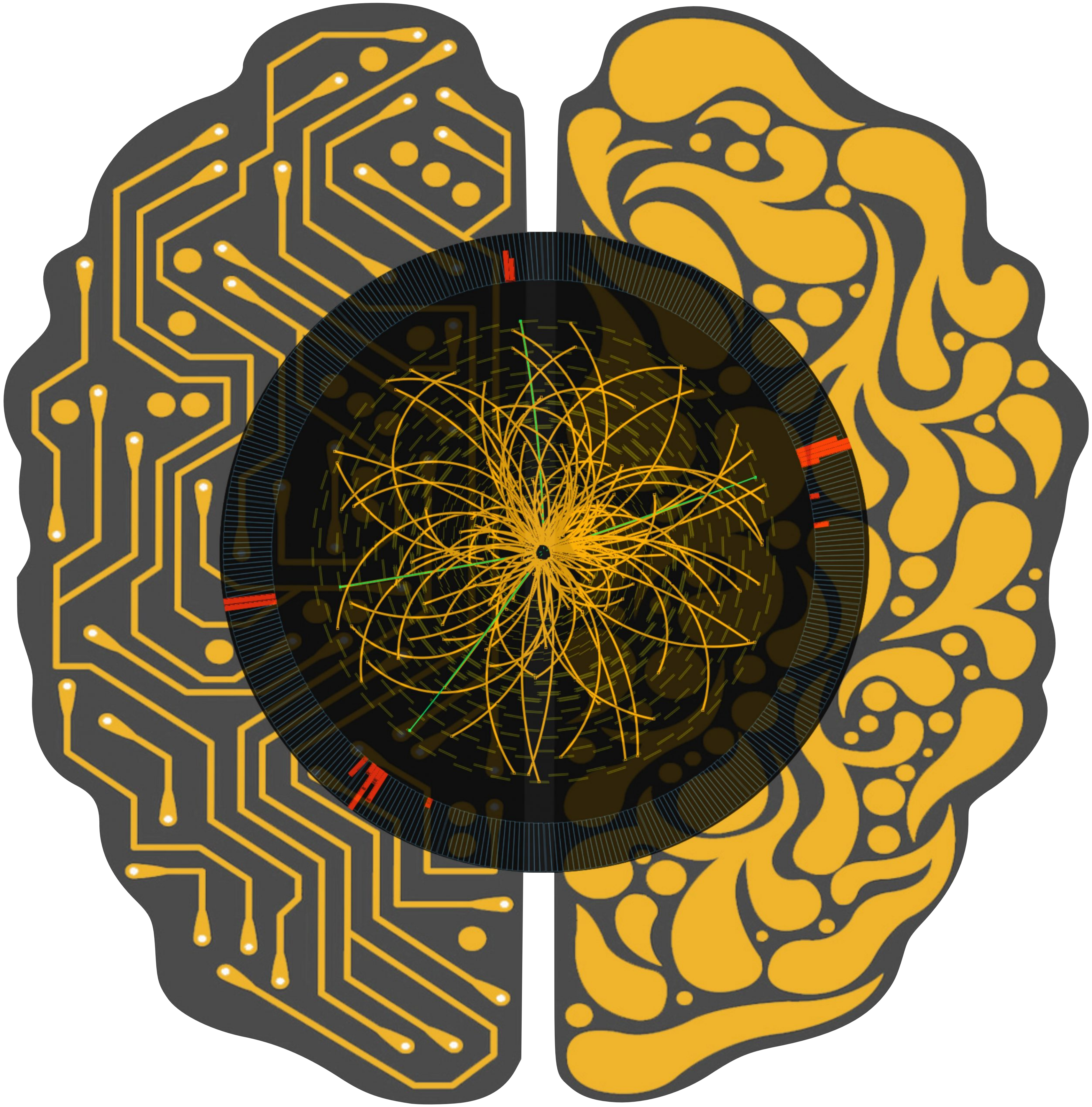


MACHINE LEARNING IN FPGAS

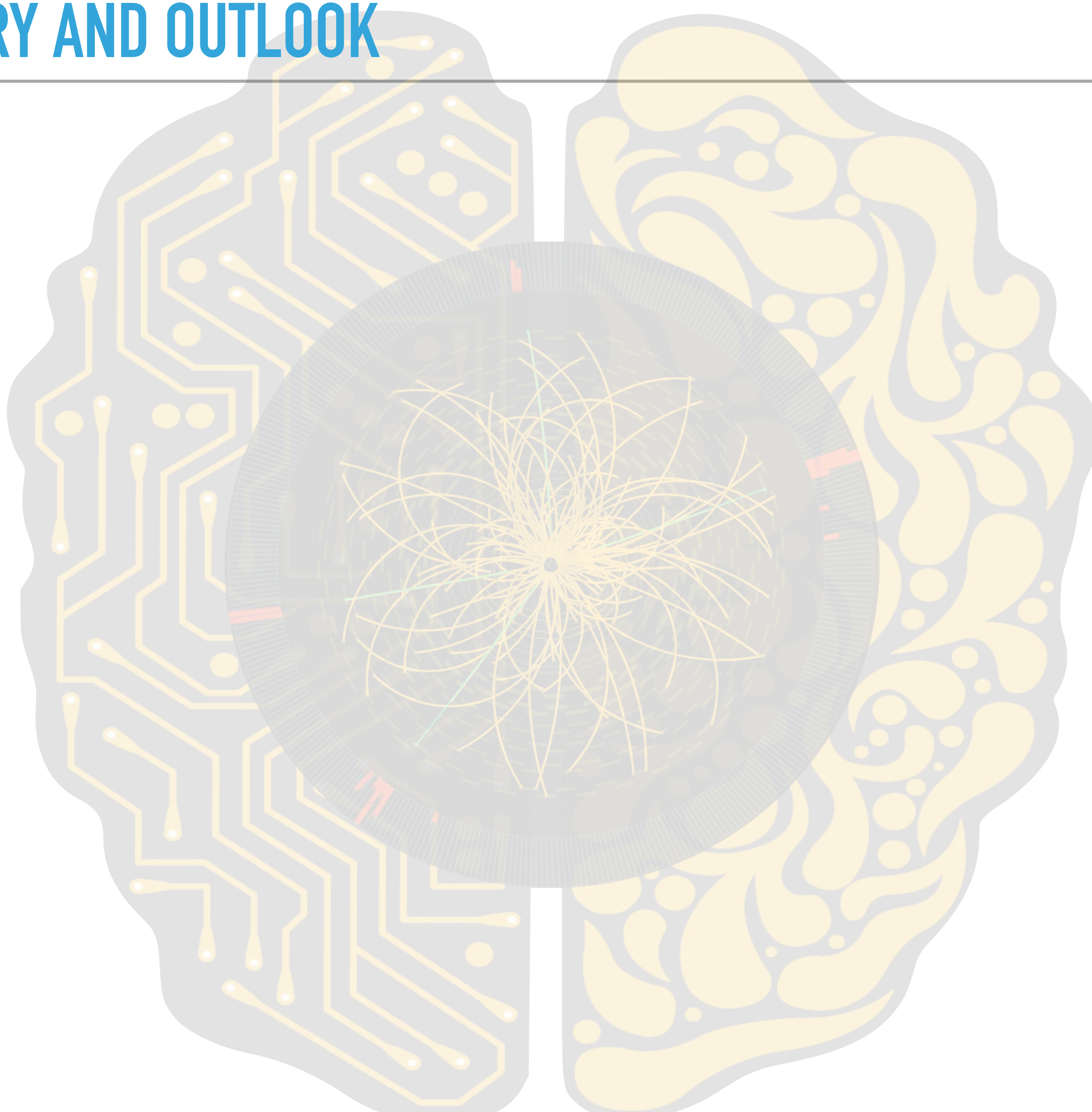
- Inference of ML algorithms possible in **O(100 ns)** on **1 FPGA** with **hls4ml**!
  - Applications across CMS, ATLAS, DUNE, and accelerator controls
  - E.g. muon p_T determination in the CMS endcap with a DNN: **runs in 160 ns** on an FPGA and **reduces** the **fake muon rate** by **up to 80%**



- Currently supported:
  - Small and large dense NNs
  - Bernary and ternary NNs
  - Small 1D/2D CNNs
- Planned support
  - Big 1D/2D CNNs
  - Graph NNs
  - Other HLS/RTL backends

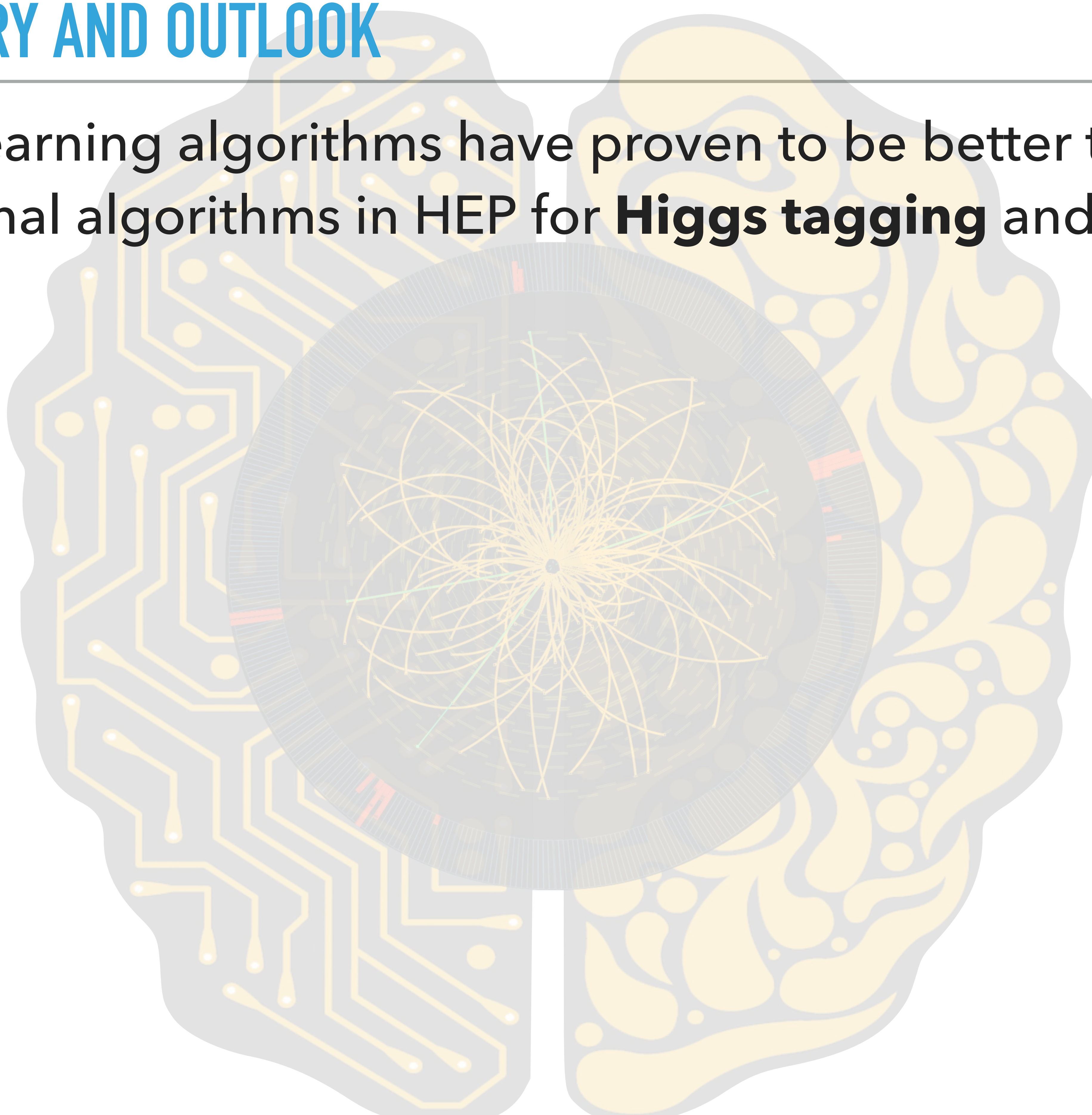


SUMMARY AND OUTLOOK



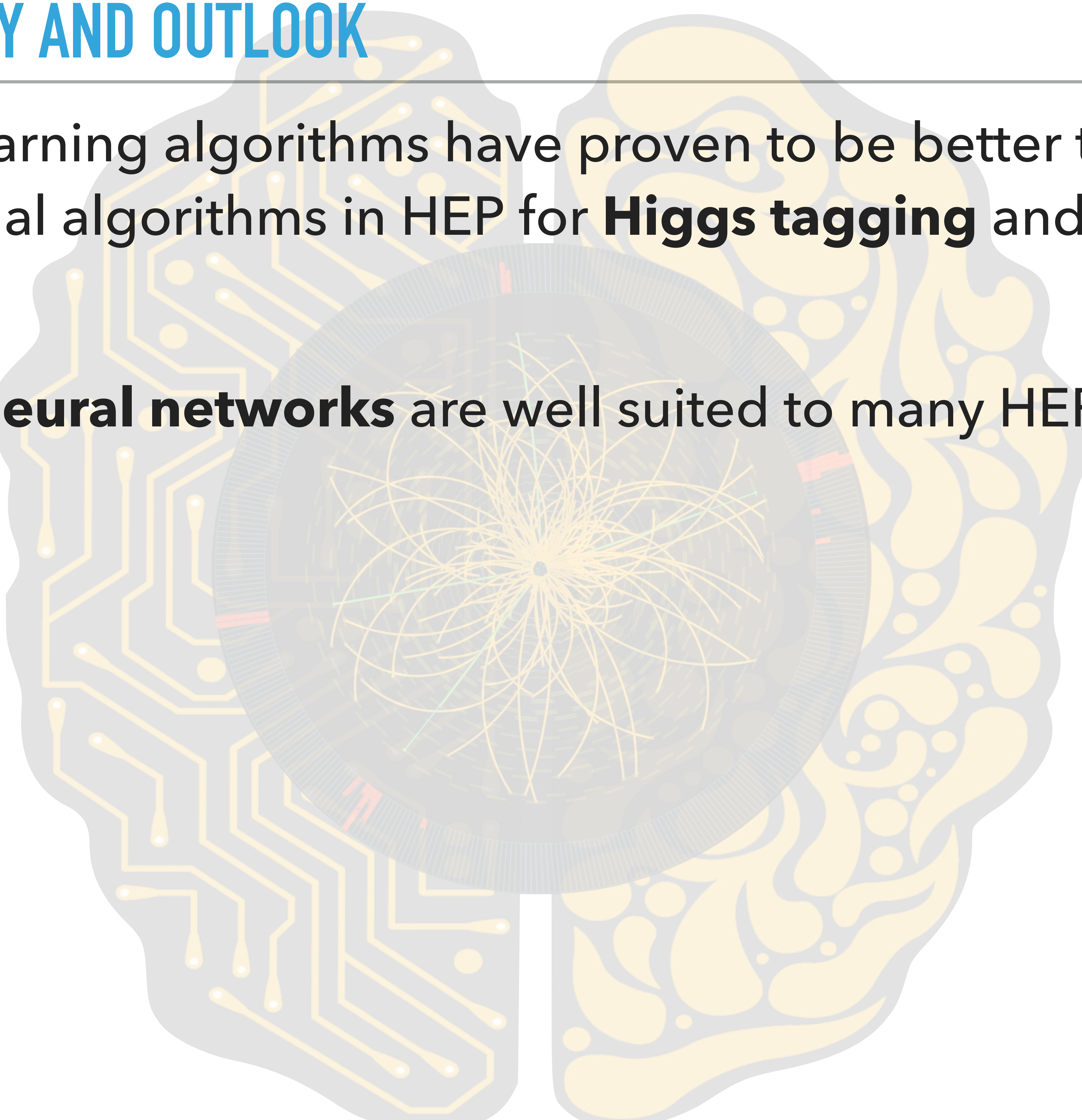
SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **Higgs tagging** and much more



SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **Higgs tagging** and much more
- **Graph neural networks** are well suited to many HEP tasks

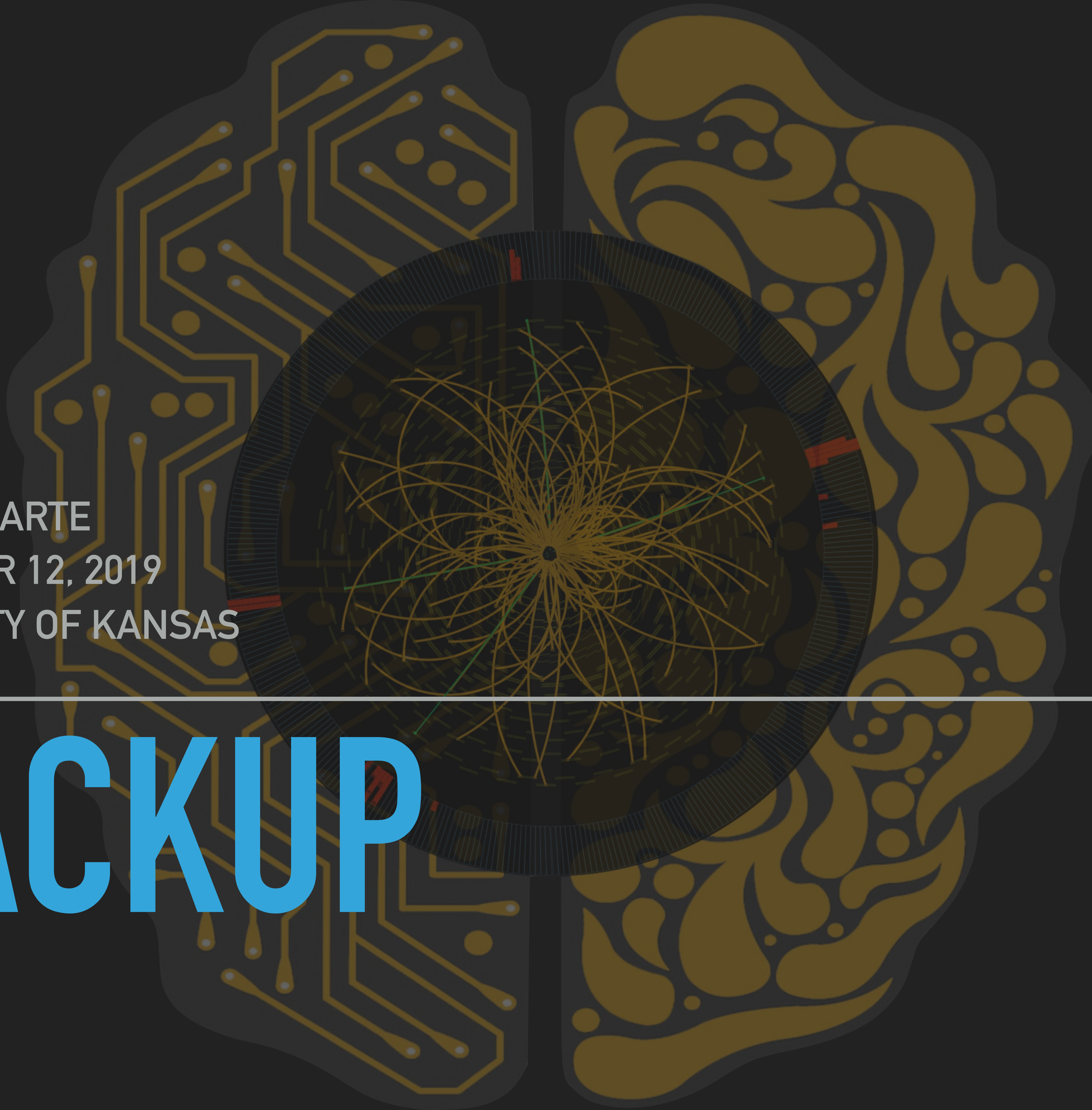


SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **Higgs tagging** and much more
- **Graph neural networks** are well suited to many HEP tasks
- **Unsupervised methods** may help us discover “unexpected” new physics

SUMMARY AND OUTLOOK

- Deep learning algorithms have proven to be better than traditional algorithms in HEP for **Higgs tagging** and much more
- **Graph neural networks** are well suited to many HEP tasks
- **Unsupervised methods** may help us discover “unexpected” new physics
- With FPGAs, ML methods can be implemented **quickly** and **efficiently**

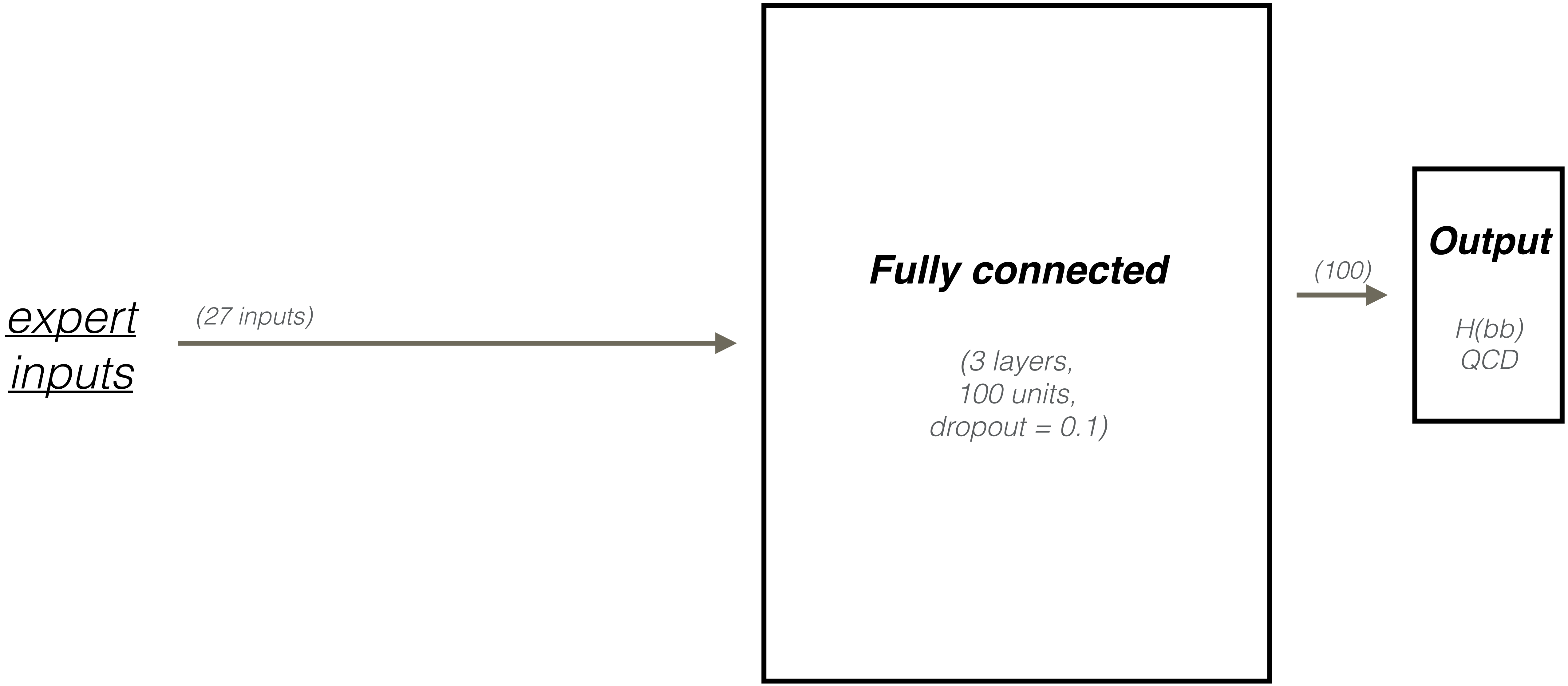


JAVIER DUARTE
NOVEMBER 12, 2019
UNIVERSITY OF KANSAS

BACKUP

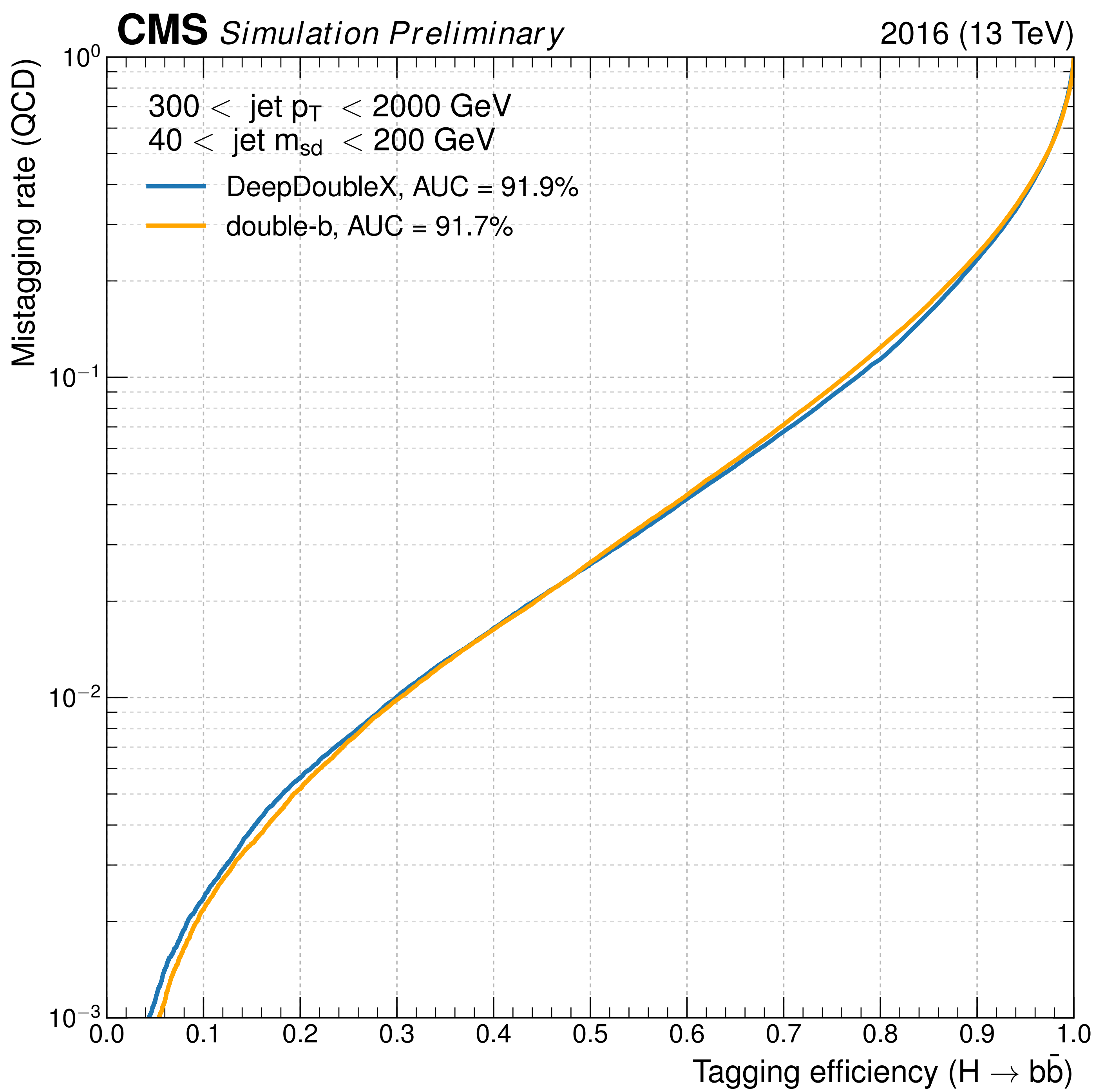
FROM DOUBLE-B TO DEEP DOUBLE-B

- First, we can change the architecture from a BDT to a neural network



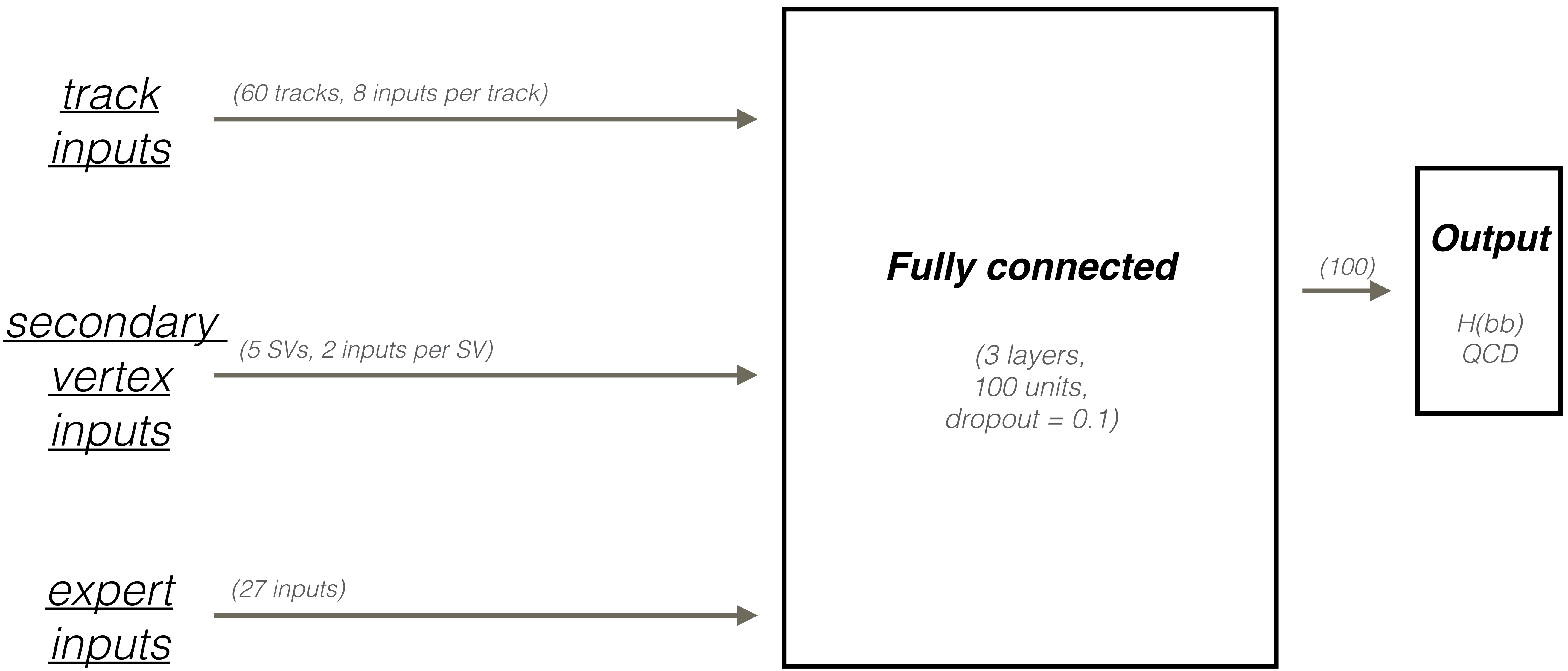
FROM DOUBLE-B TO DEEP DOUBLE-B

Same inputs, simple neural network → same performance



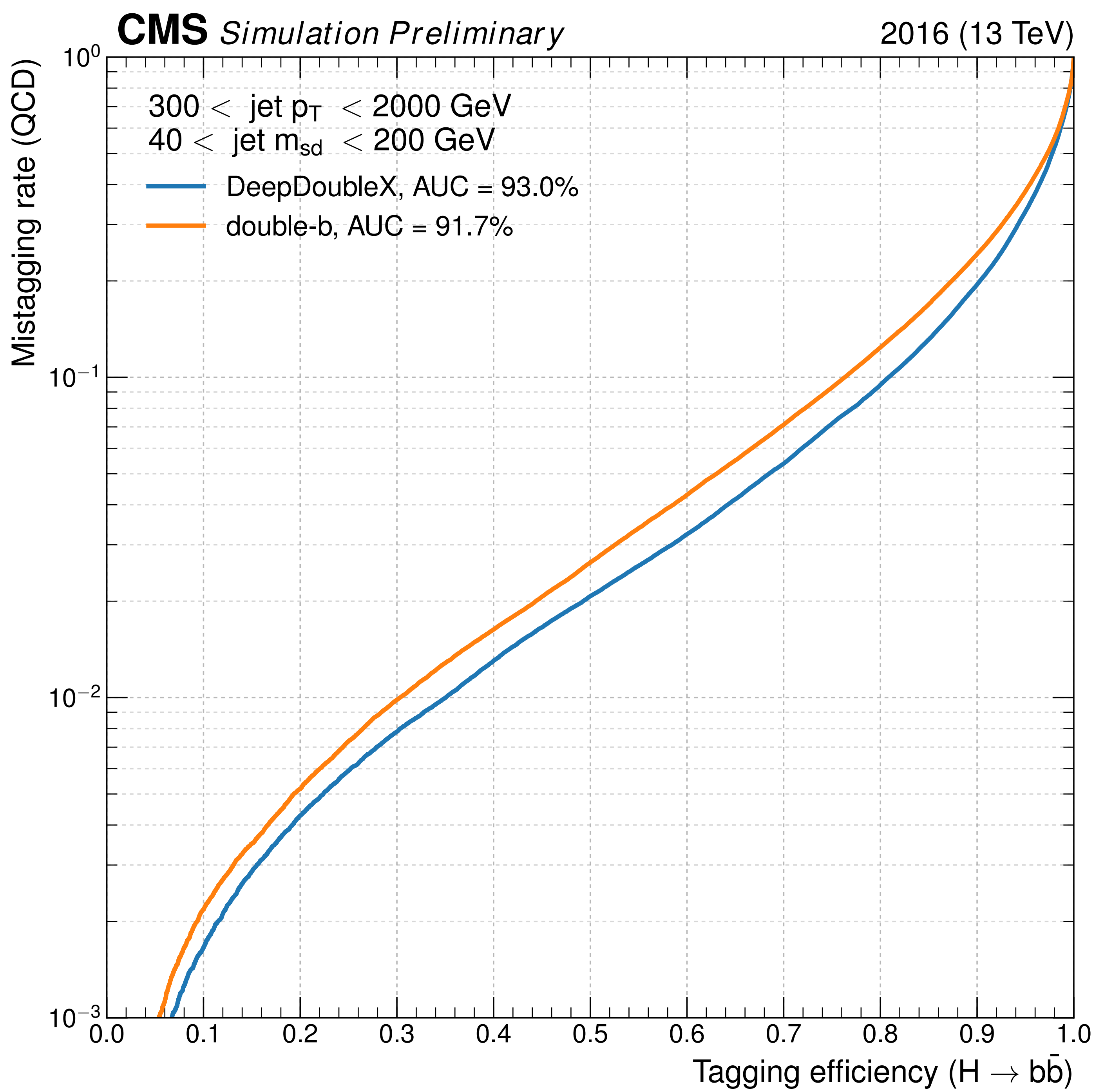
FROM DOUBLE-B TO DEEP DOUBLE-B

- We can switch to a neural network and add more low-level inputs based on track information and secondary vertex information: up to 517 input variables!



FROM DOUBLE-B TO DEEP DOUBLE-B

No big gain in performance…



FROM DOUBLE-B TO DEEP DOUBLE-B

No big gain in performance…

CMS *Simulation Preliminary* 2016 (13 TeV)

WHY?

(1) MAYBE WE NEED A BIGGER NETWORK, MORE DATA, MORE TRAINING TIME, . . .

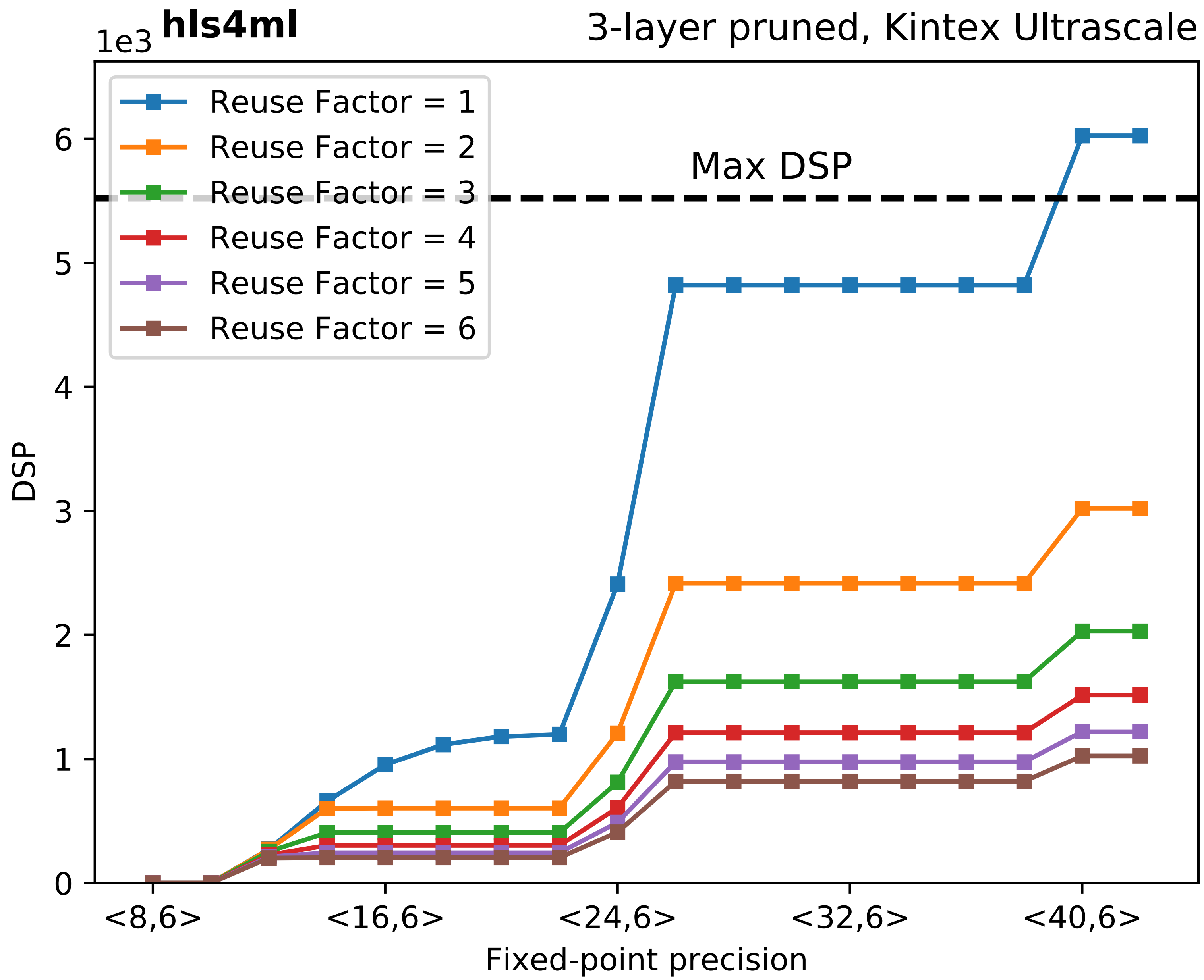
OR

(2) MAYBE WE NEED A SMARTER NETWORK

Tagging efficiency (H → bb)

NETWORK TUNING: PARALLELIZATION & RESOURCES

- **Increasing** reuse factor, **decreases** resources



5000 DSPs (90%)

800 DSPs (15%)

NETWORK TUNING: QUANTIZATION

- Scan the bit width until you reach optimal performance

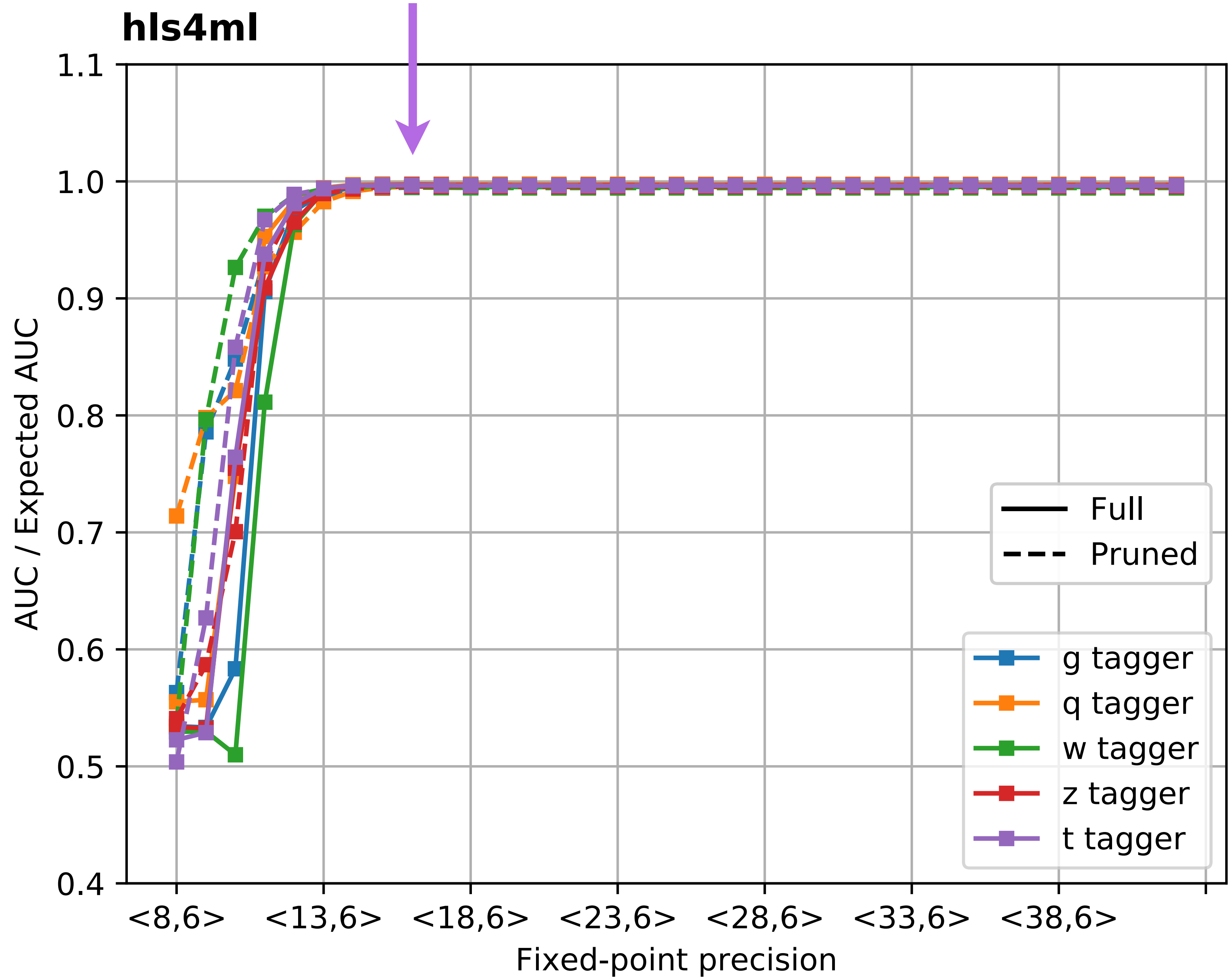
ap_fixed<width,integer>

0101.1011101010

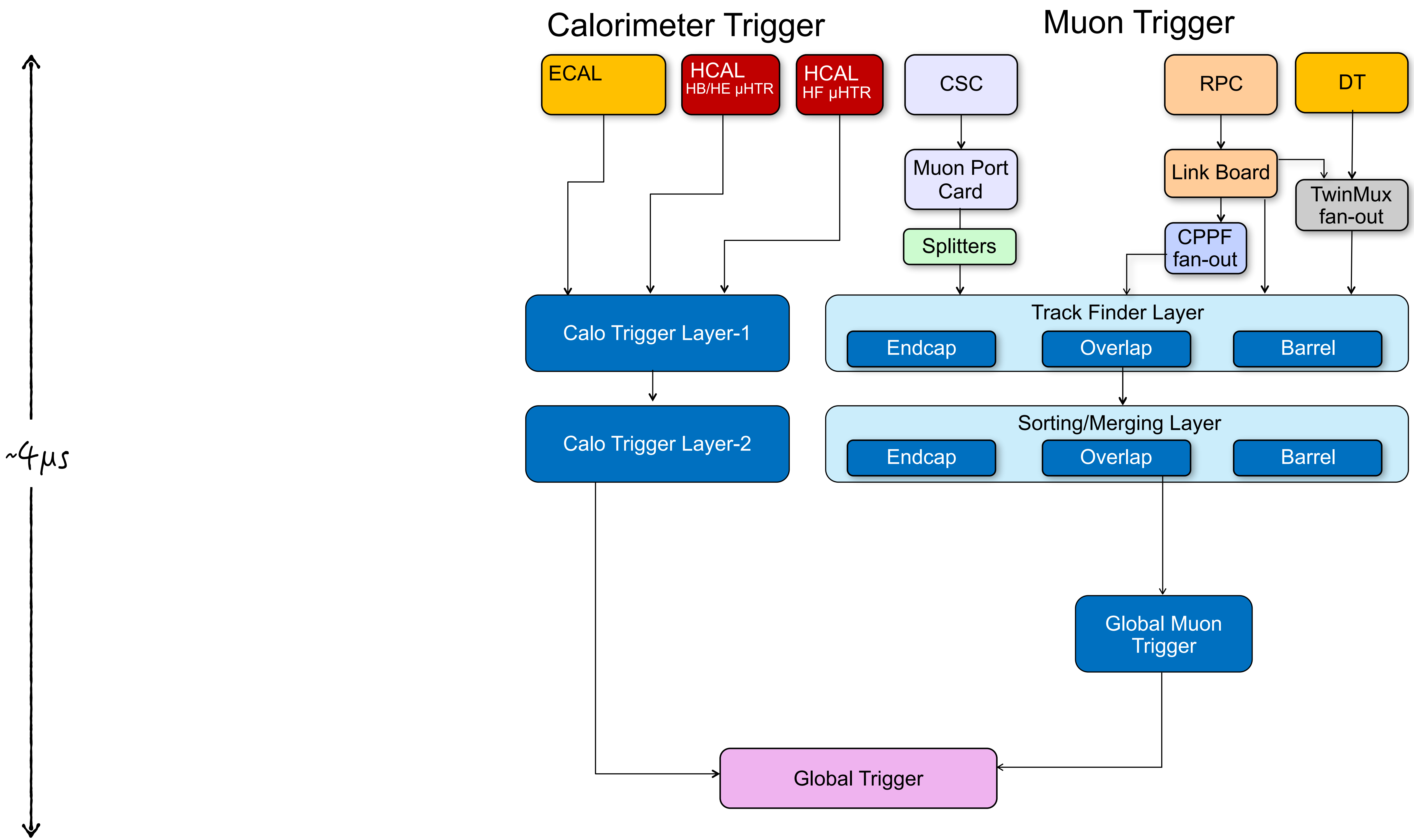
integer — fractional

width

Full performance with 16 bits



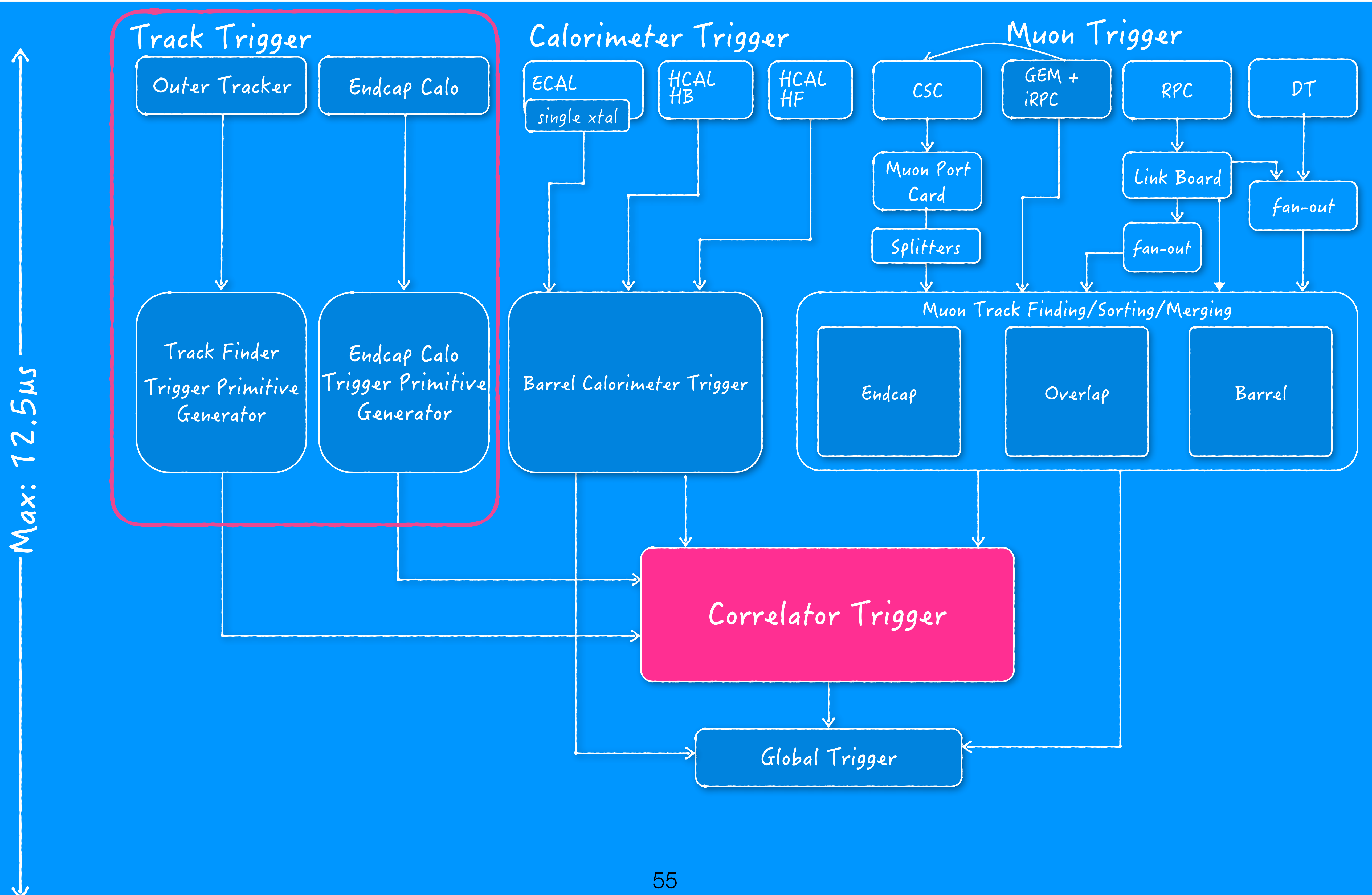
UPGRADING THE LEVEL-1 TRIGGER (BEFORE)



UPGRADING THE LEVEL-1 TRIGGER (AFTER)

More and better information available in the Level-1 trigger!

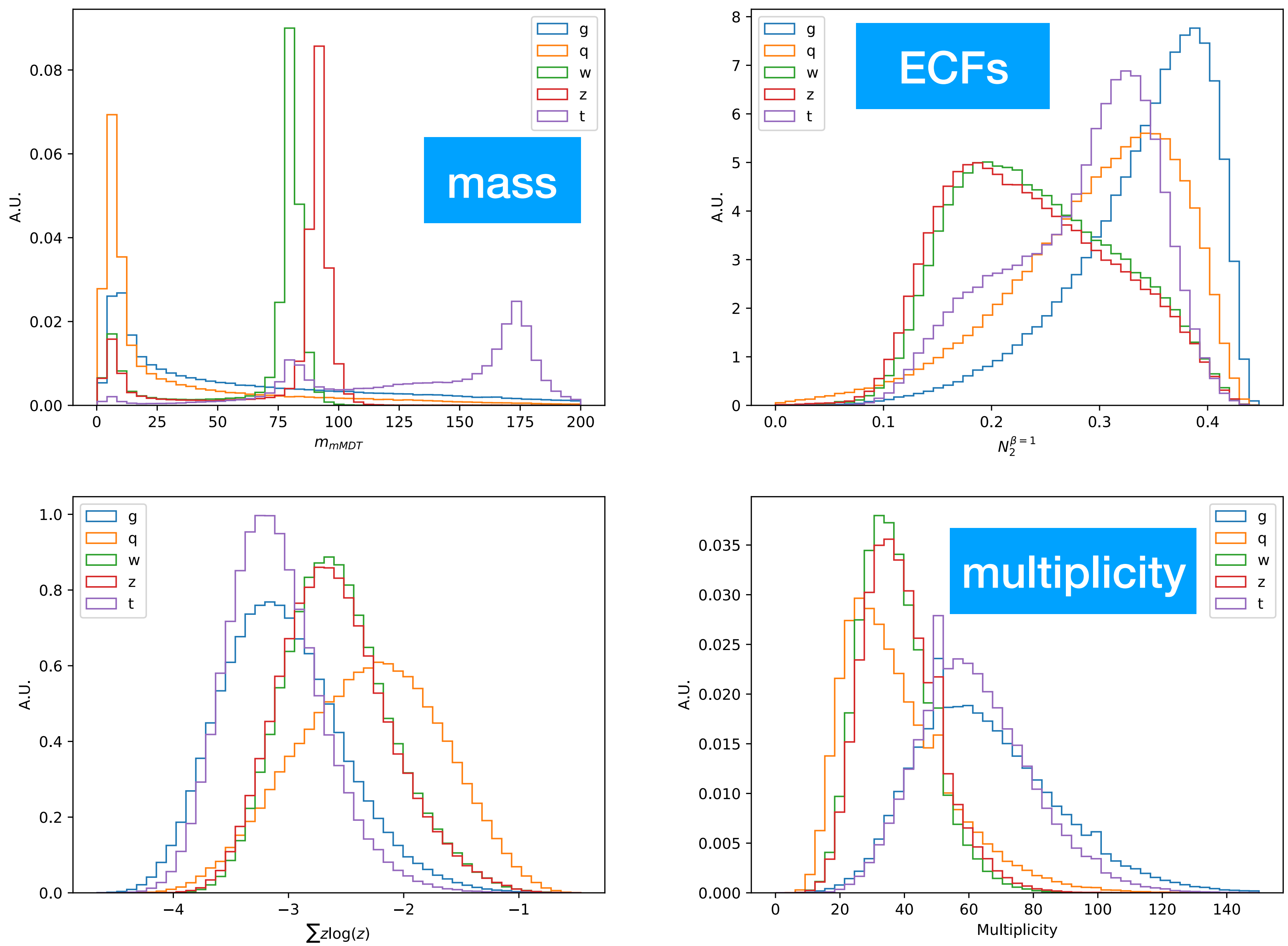
What can we do with it?



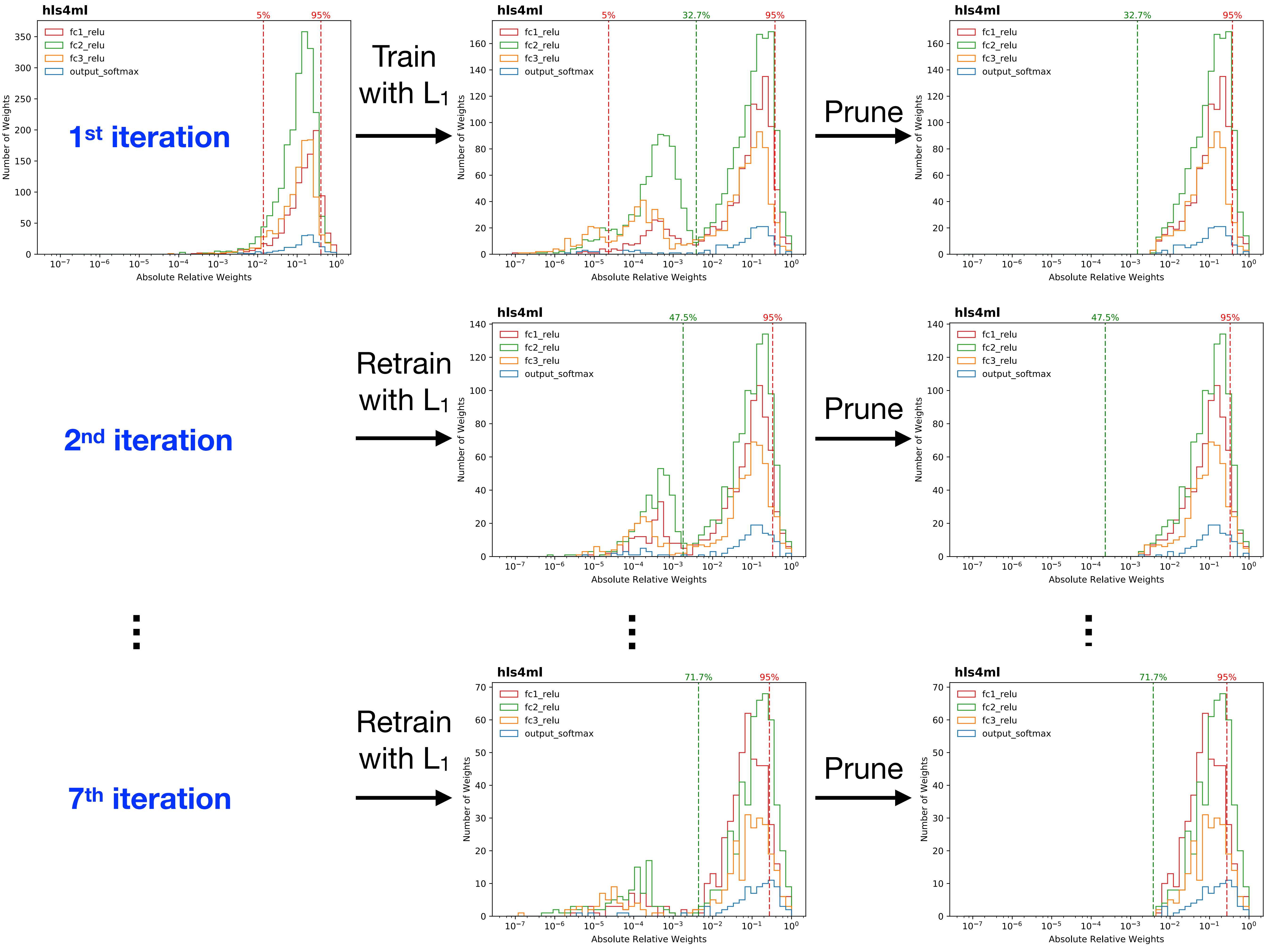
CASE STUDY: JET TAGGING INPUTS

Observables

m_{mMDT}
$N_2^{\beta=1,2}$
$M_2^{\beta=1,2}$
$C_1^{\beta=0,1,2}$
$C_2^{\beta=1,2}$
$D_2^{\beta=1,2}$
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$
$\sum z \log z$
Multiplicity

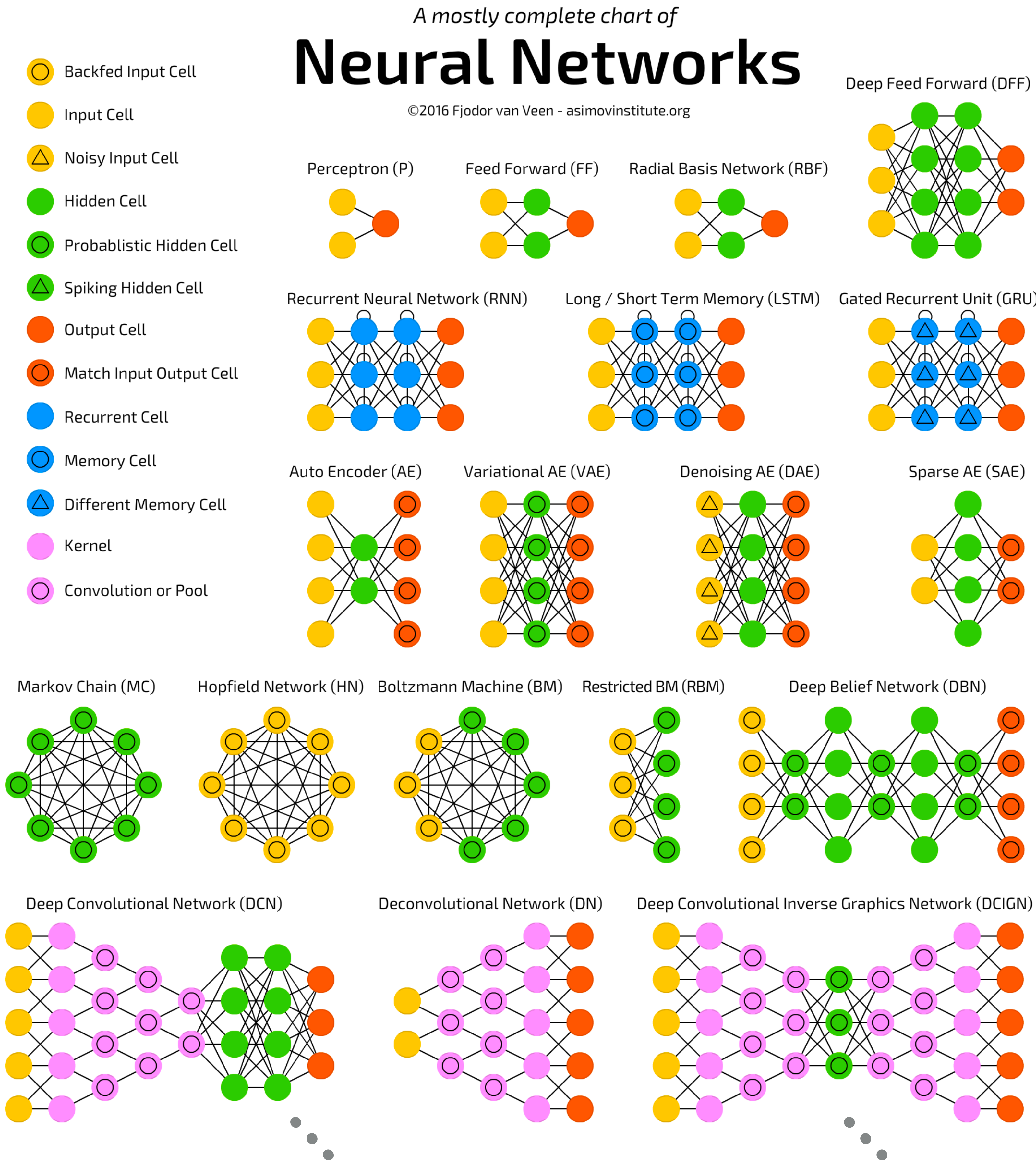


- 16 expert observables provide separation between top, W/Z, and quark/gluon



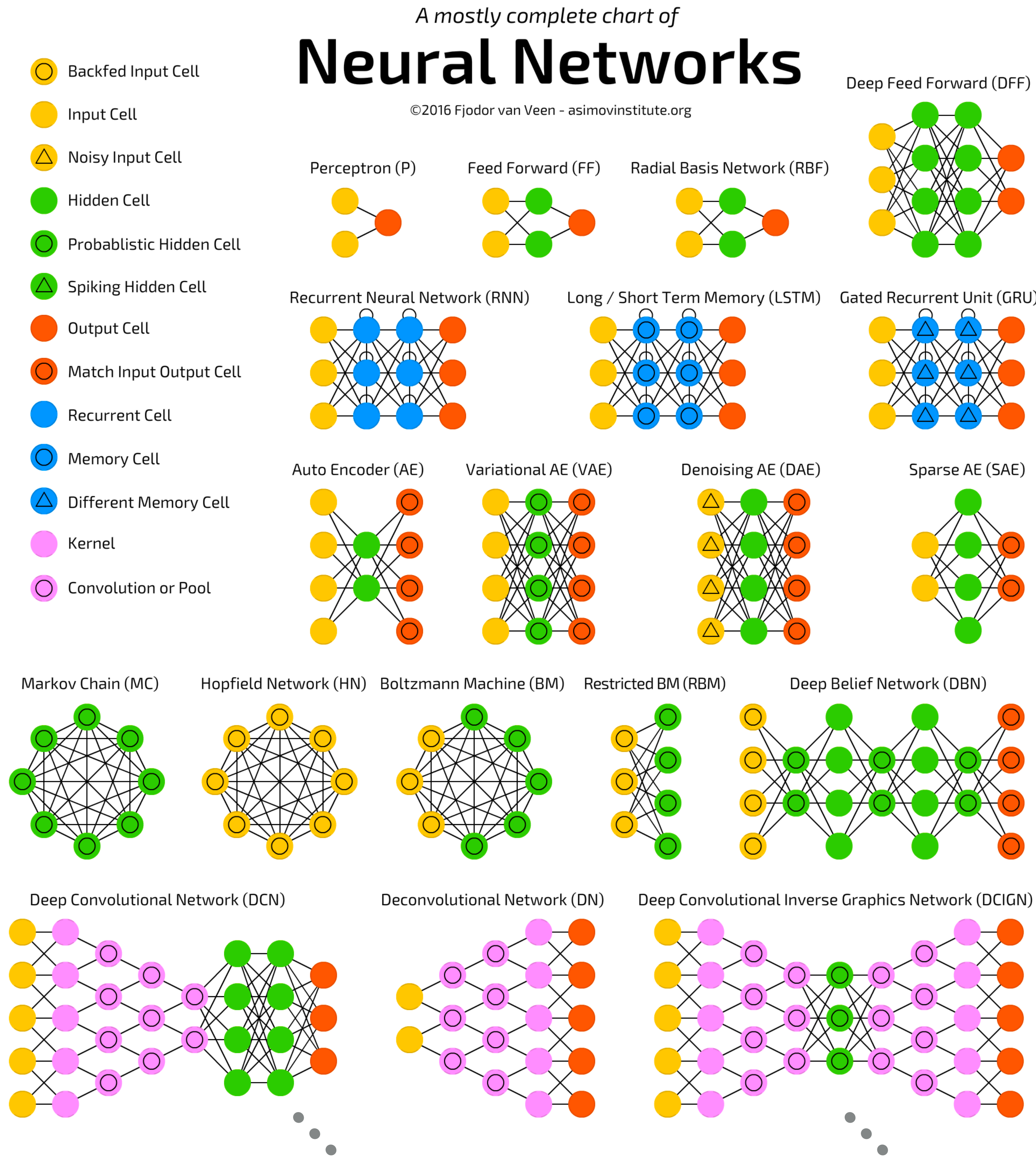
1st iteration
2nd iteration
7th iteration
Train with L1
Retrain with L1
Prune
hls4ml
fc1_relu
fc2_relu
fc3_relu
output_softmax
5%
95%
32.7%
47.5%
71.7%
Number of Weights
Absolute Relative Weights

NEURAL NETWORK ZOO



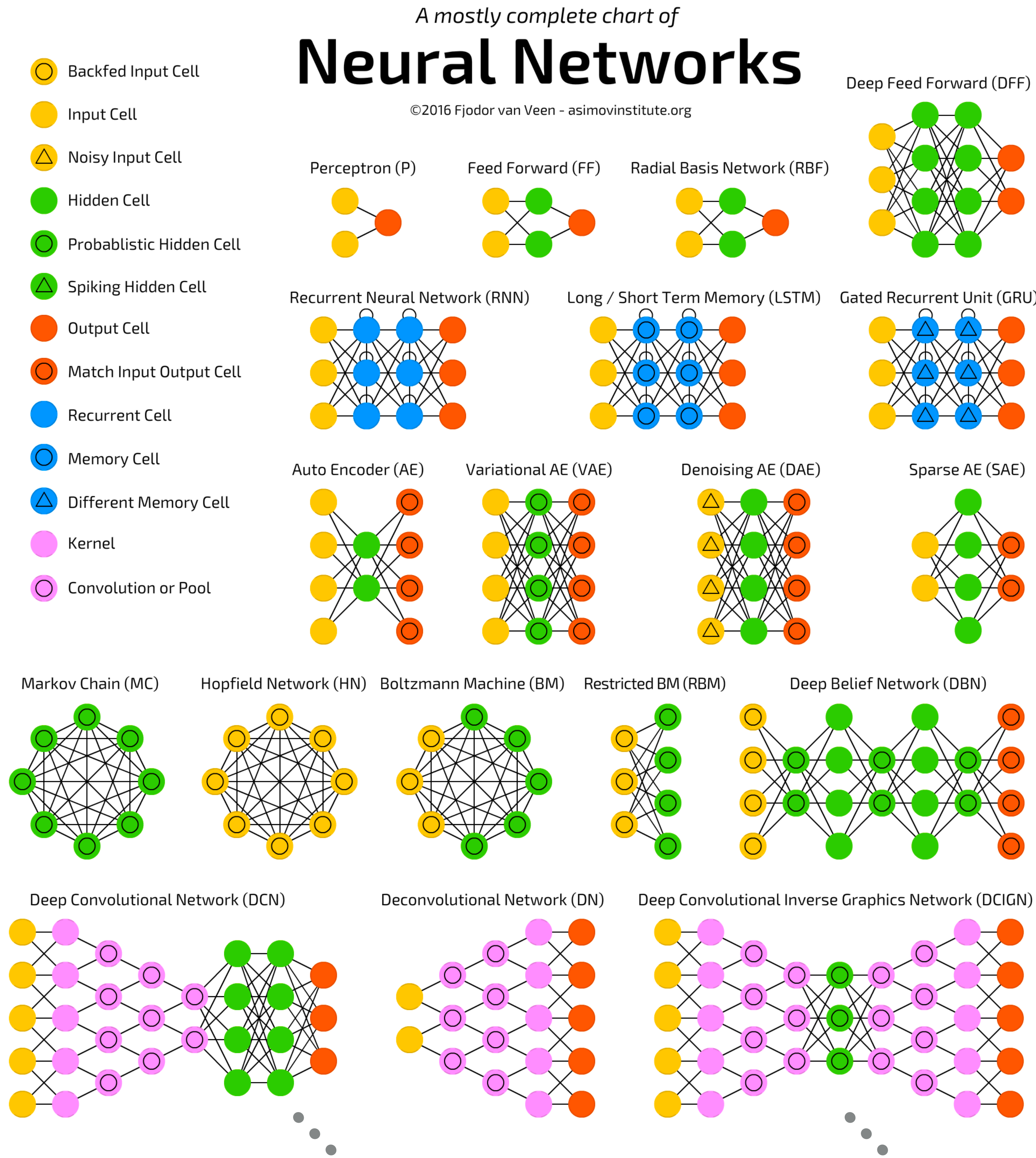
NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want



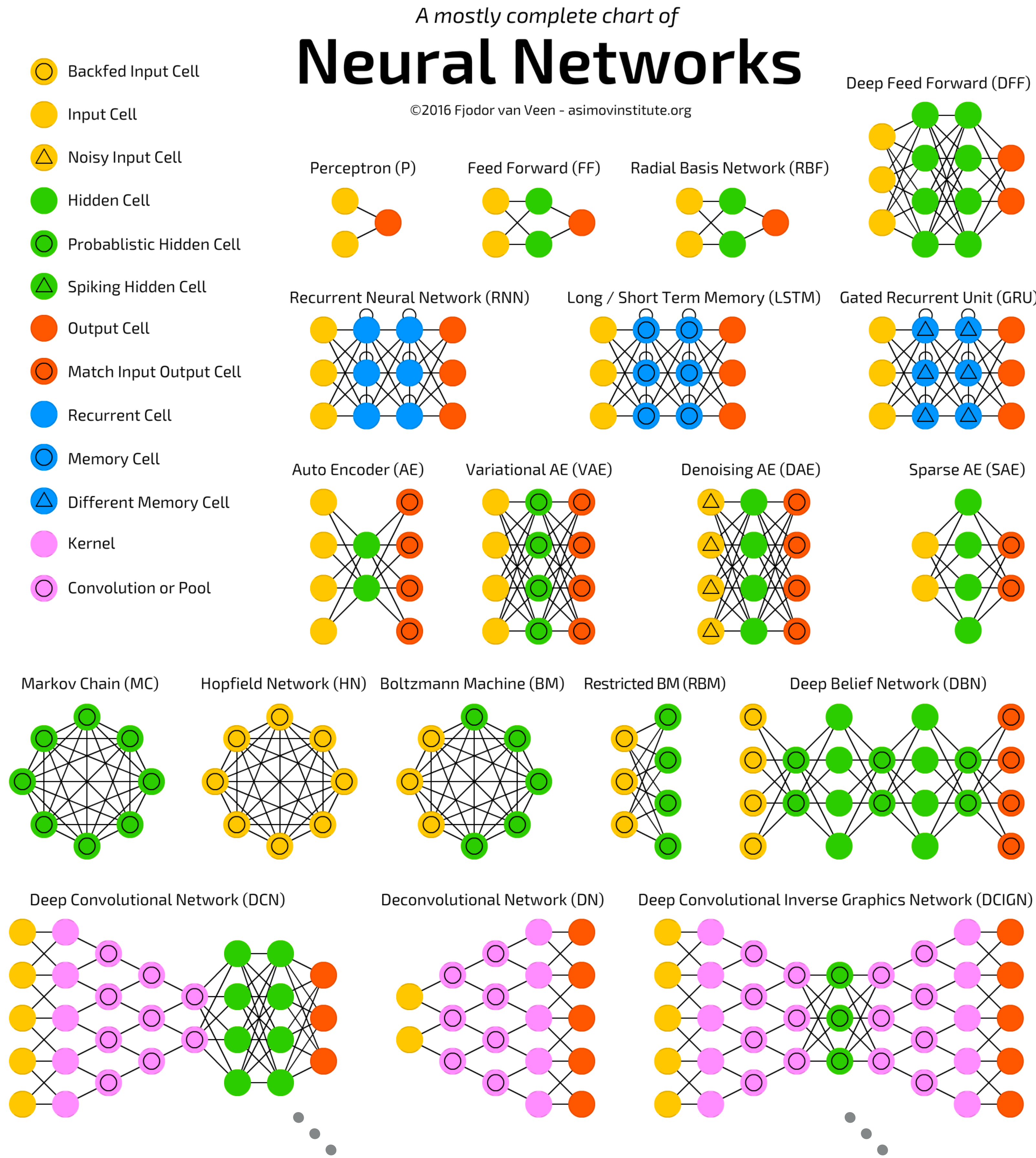
NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
    - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)



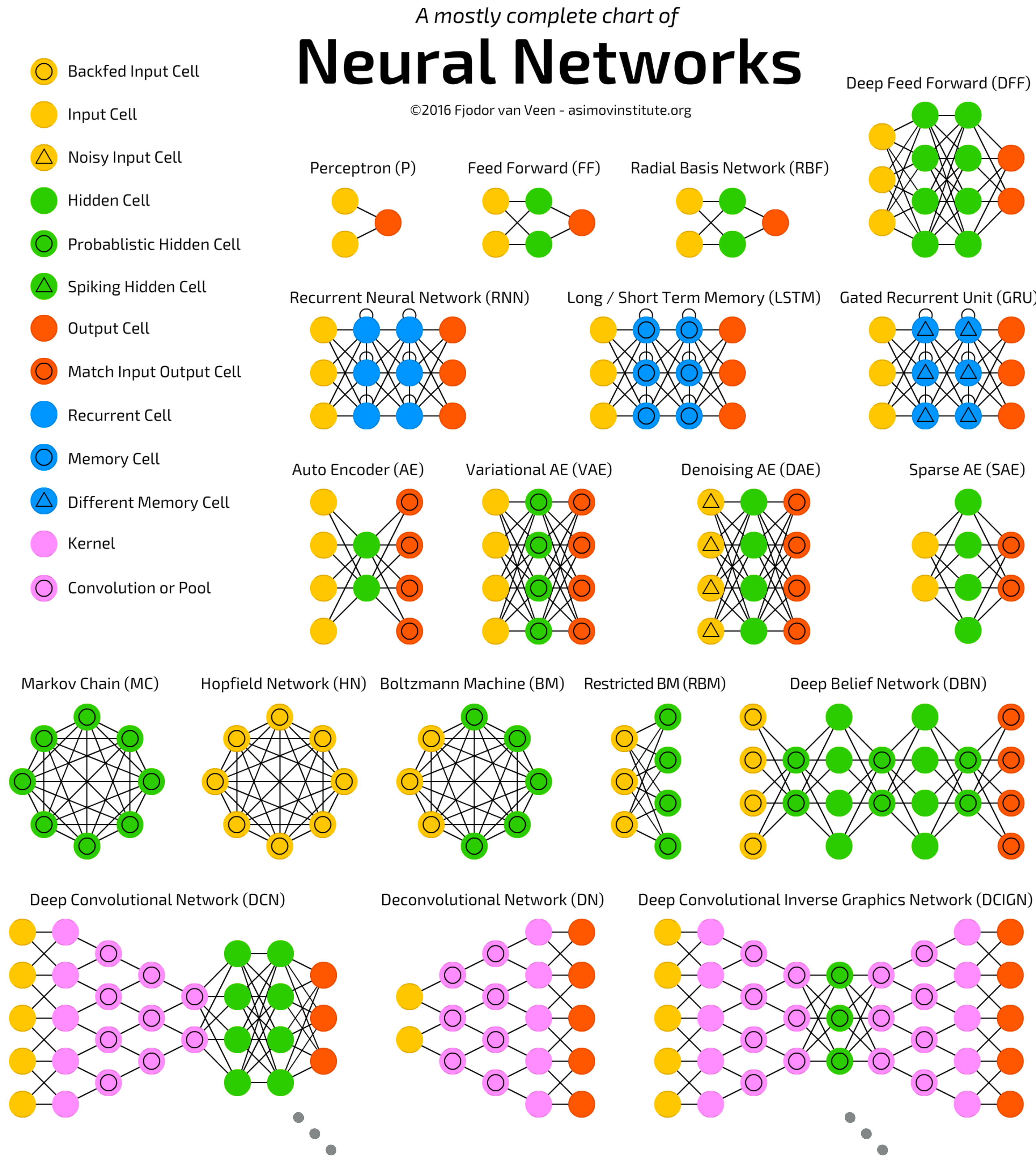
NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
    - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)
- NNs are the best ML solution on the market *today*



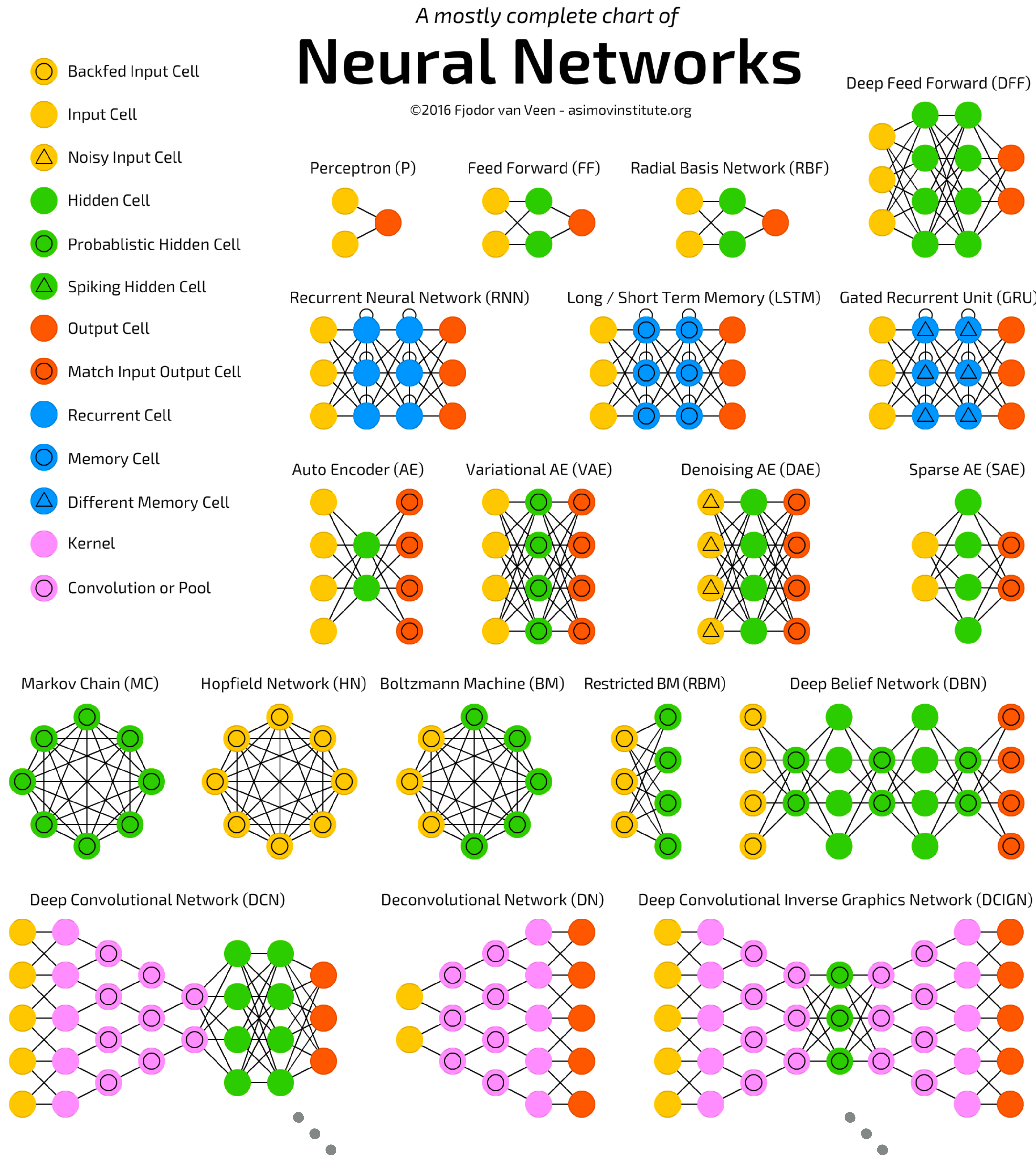
NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
    - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)
- NNs are the best ML solution on the market *today*
    - Each node performs a math operation on the input



NEURAL NETWORK ZOO

- You have a task to accomplish, which can be represented as a smooth function from your inputs to the answer you want
  - Train an algorithm to learn an approximation of the optimal solution function (Machine Learning)
- NNs are the best ML solution on the market *today*
  - Each node performs a math operation on the input
  - Edges represent the flow of nodes' inputs & outputs



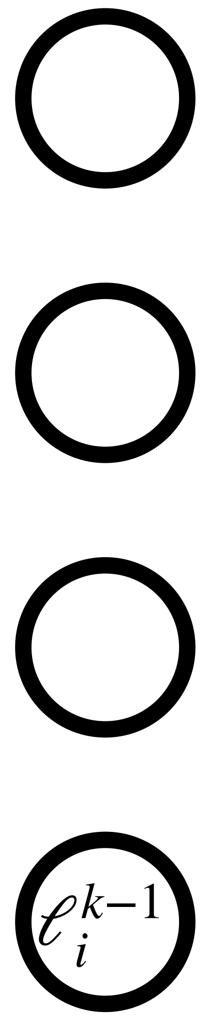
ℓ_i^{k-1}

ℓ_i^{k-1}

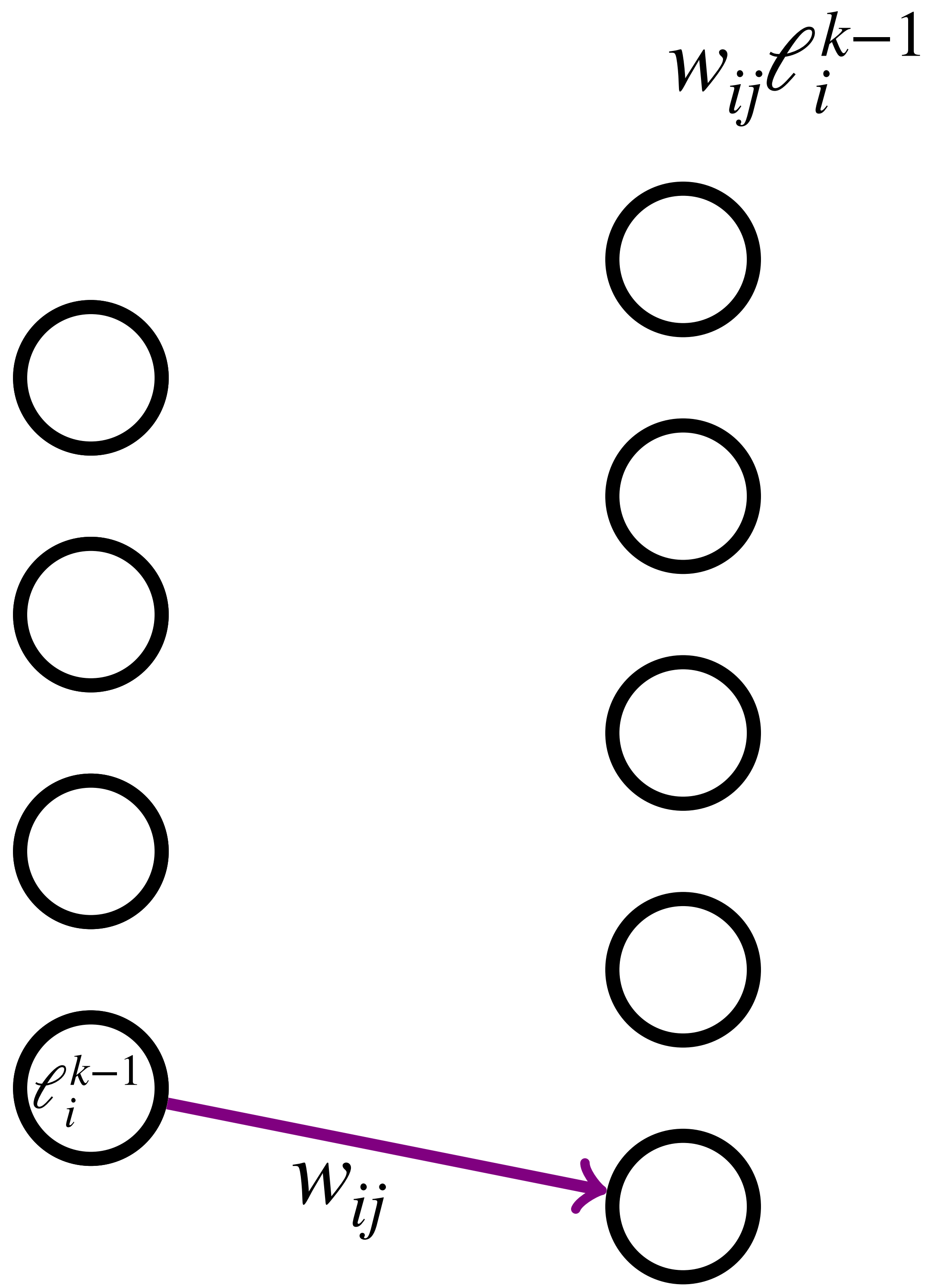
NEURAL NETWORK

- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.

ℓ_i^{k-1}

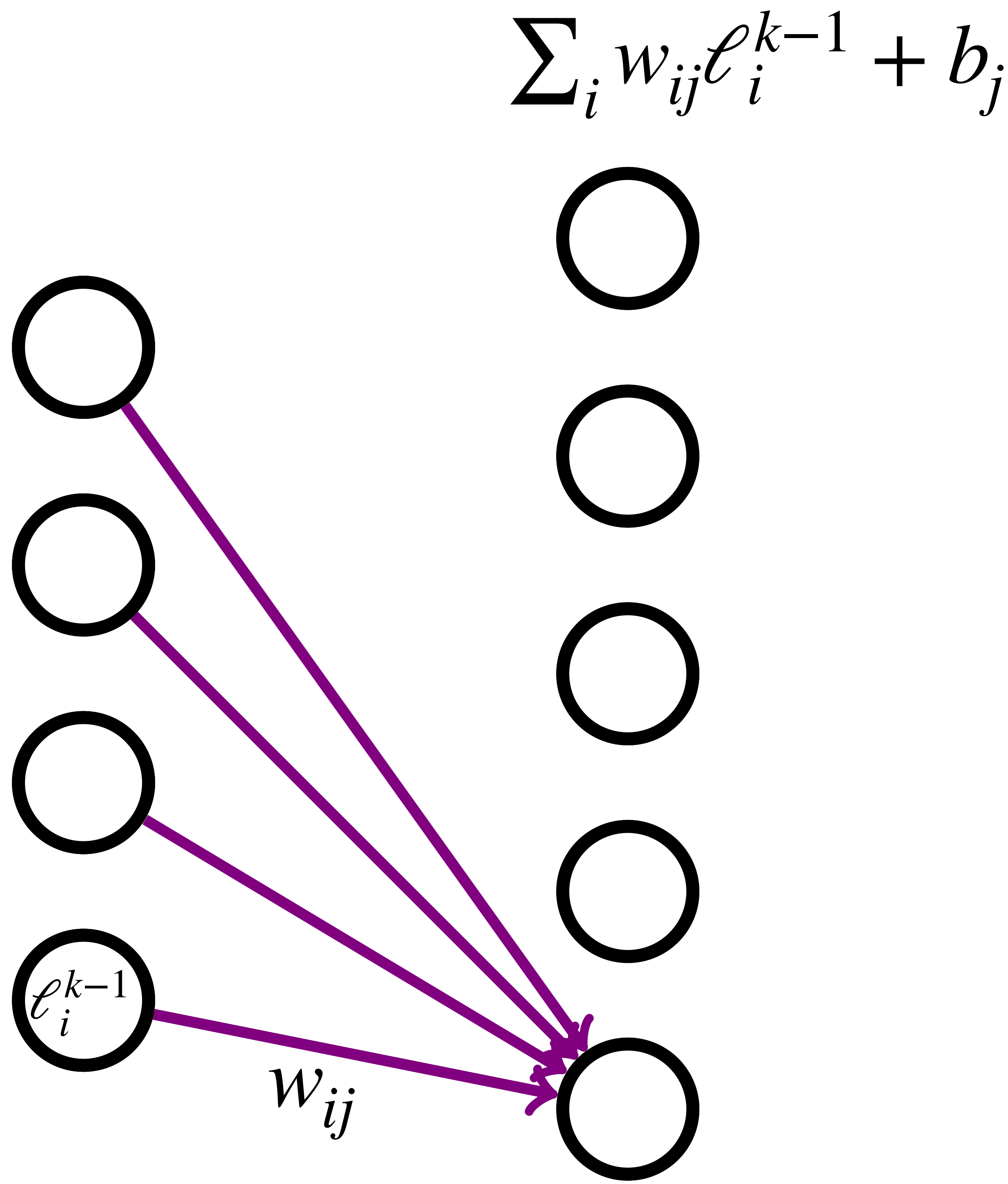


- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.
- Each input multiplied by a weight



NEURAL NETWORK

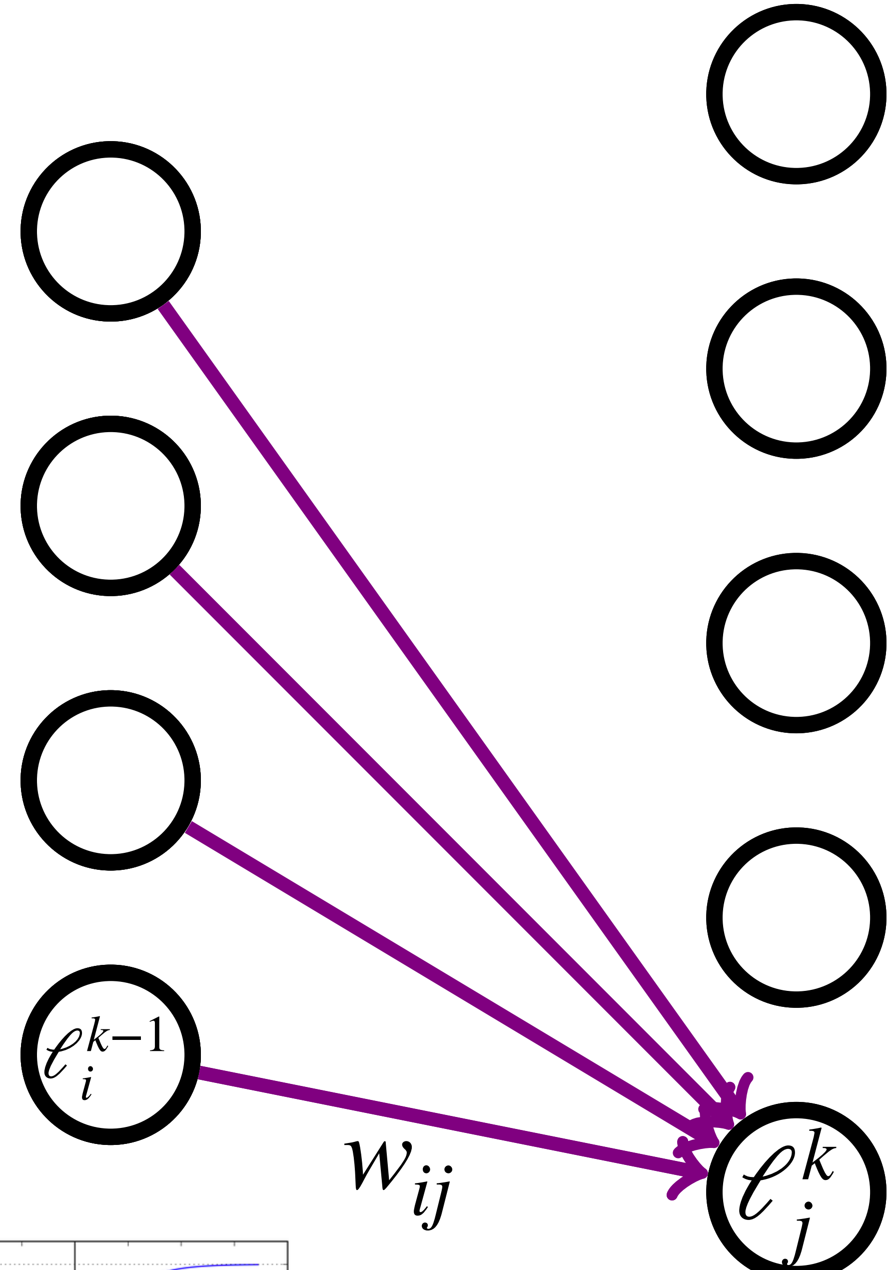
- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.
- Each input multiplied by a weight
- Weighted values are summed, bias is added

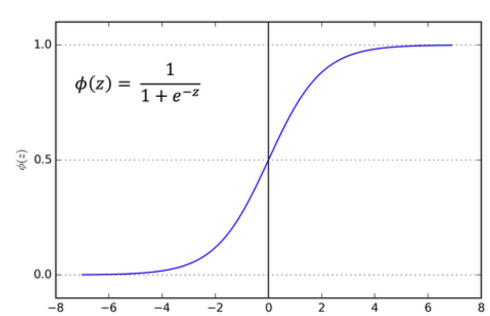


NEURAL NETWORK

- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.
- Each input multiplied by a weight
- Weighted values are summed, bias is added
- Nonlinear activation function is applied

$$\ell_j^k = \phi\left(\sum_i w_{ij}\ell_i^{k-1} + b_j\right)$$

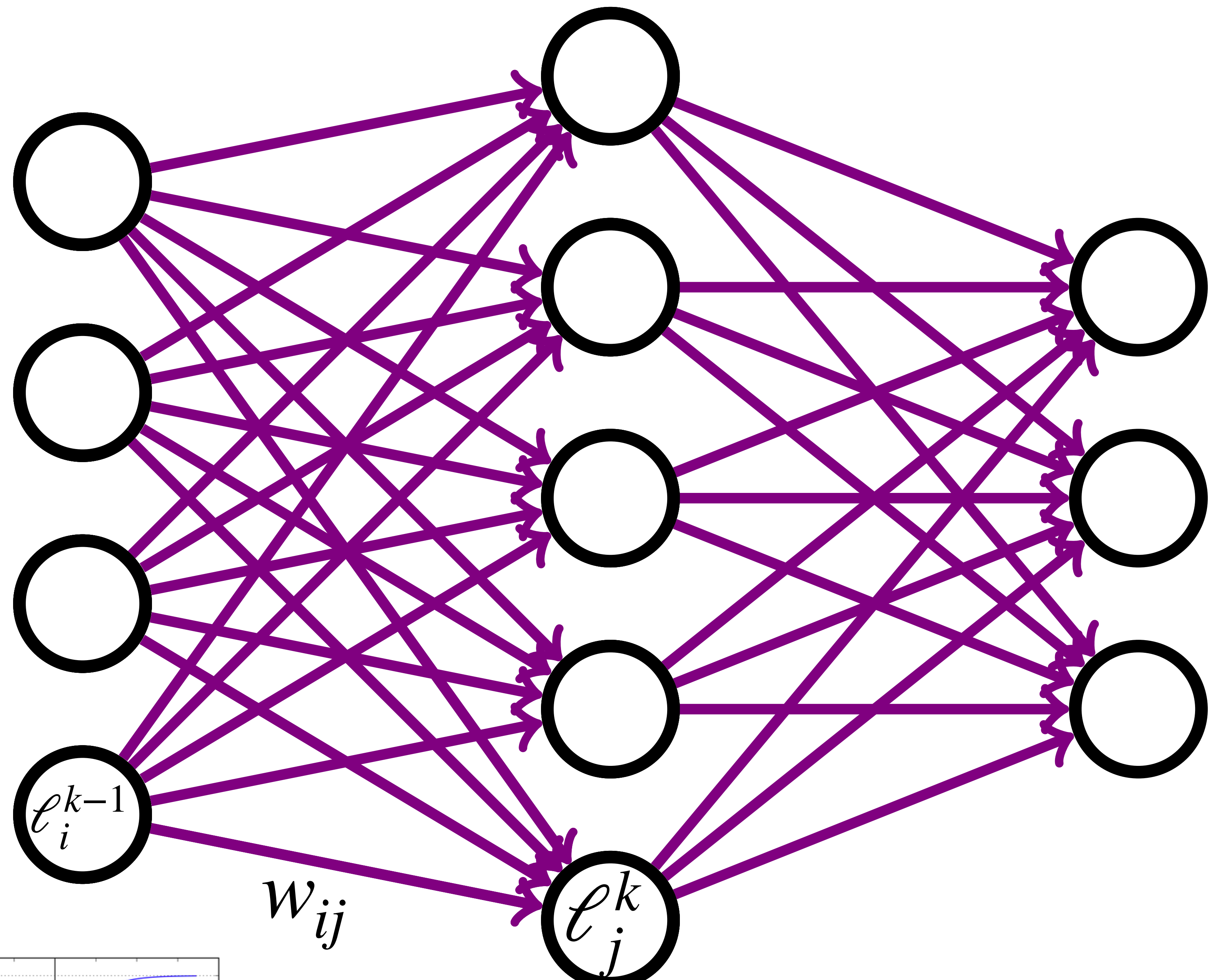


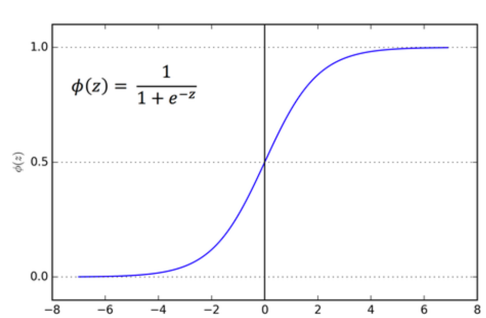


NEURAL NETWORK

$$\ell_j^k = \phi\left(\sum_i w_{ij}\ell_i^{k-1} + b_j\right)$$

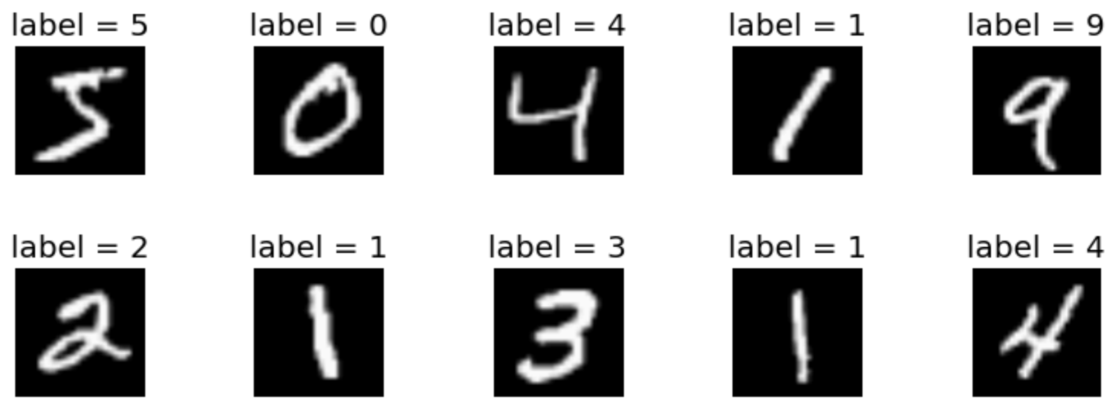
- Classic feed forward architecture with some modifications responsible for revolutions in computer vision, language processing, etc.
- Each input multiplied by a weight
- Weighted values are summed, bias is added
- Nonlinear activation function is applied

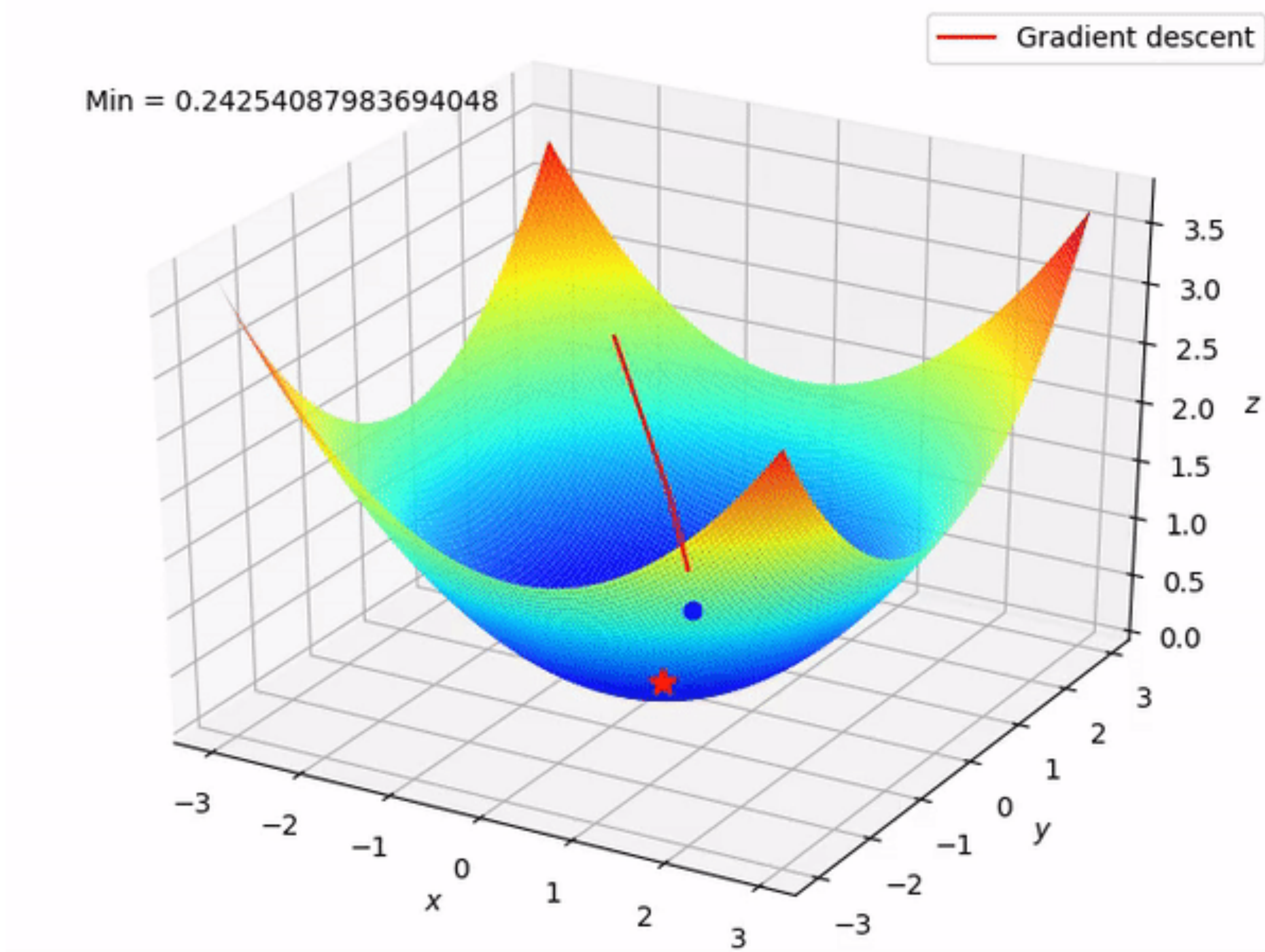




A sufficiently "wide" neural network can approximate any function!

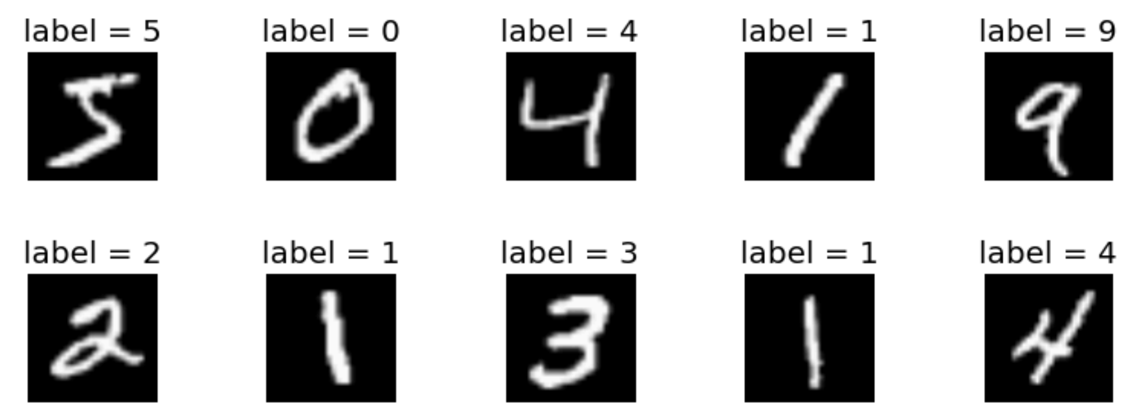
TRAINING

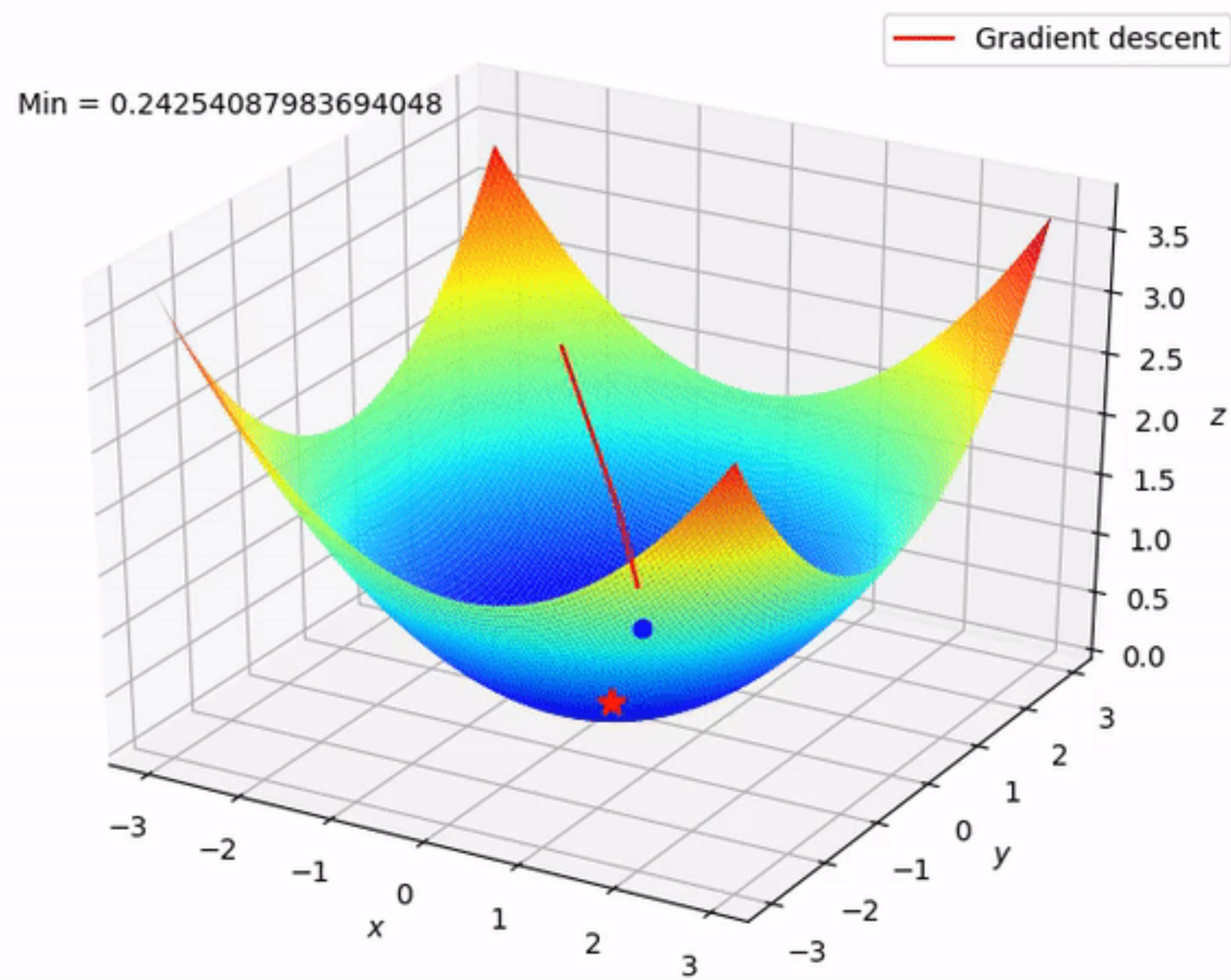

label = 5
label = 0
label = 4
label = 1
label = 9
label = 2
label = 1
label = 3
label = 1
label = 4


Gradient descent
Min = 0.24254087983694048
x
y
z

TRAINING

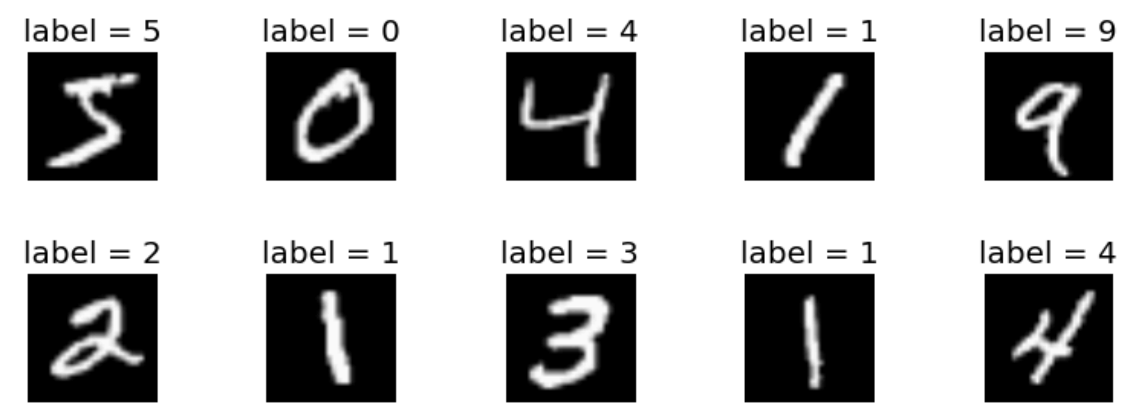
- A network is trained by specifying inputs, targets, and a loss function
  - Target is what the network should learn for that input, can be a "truth" label (supervised) or the input itself (unsupervised)
  - Loss function quantifies how many mistakes the network makes
- Training is the minimization of the loss function by varying the network parameters

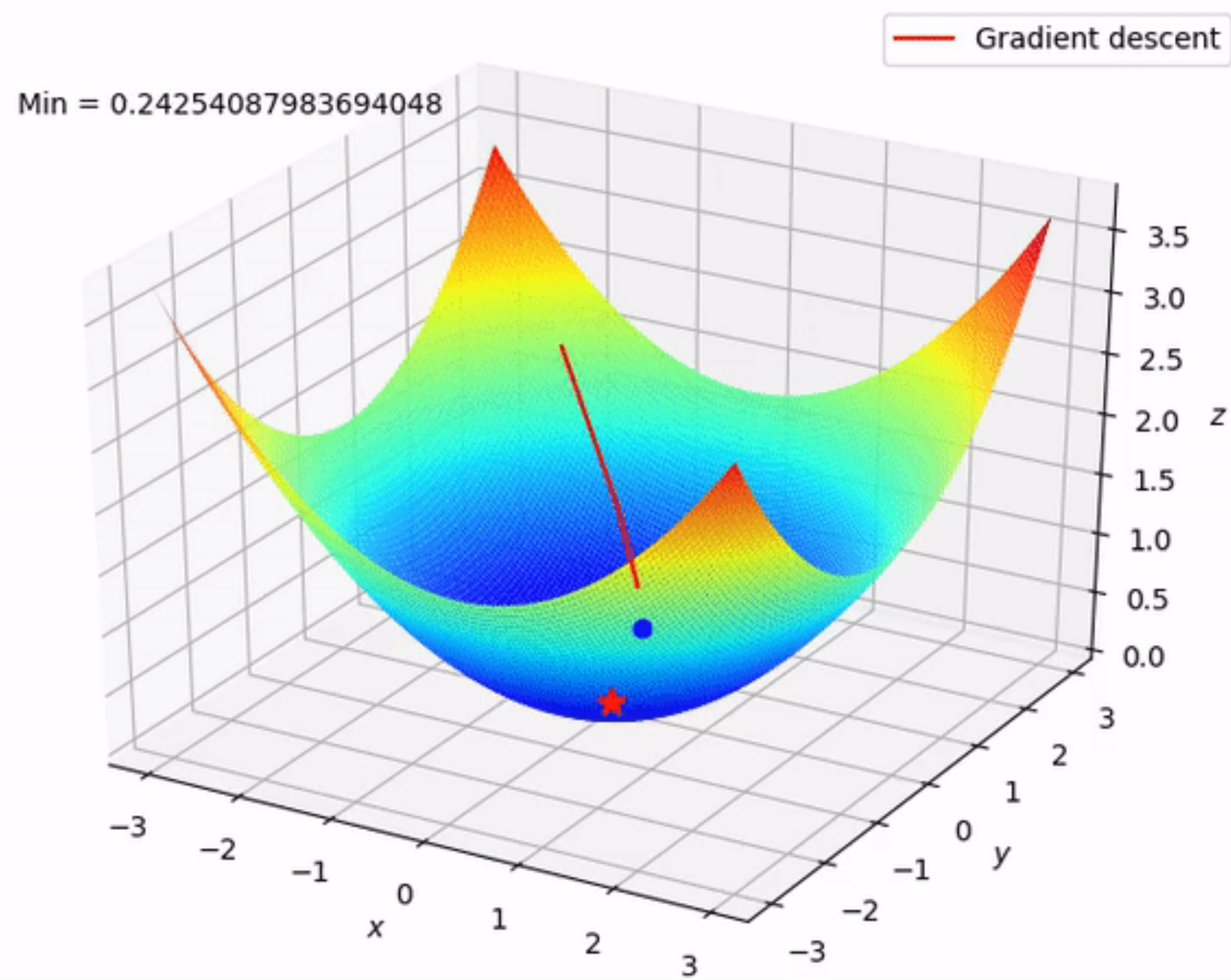




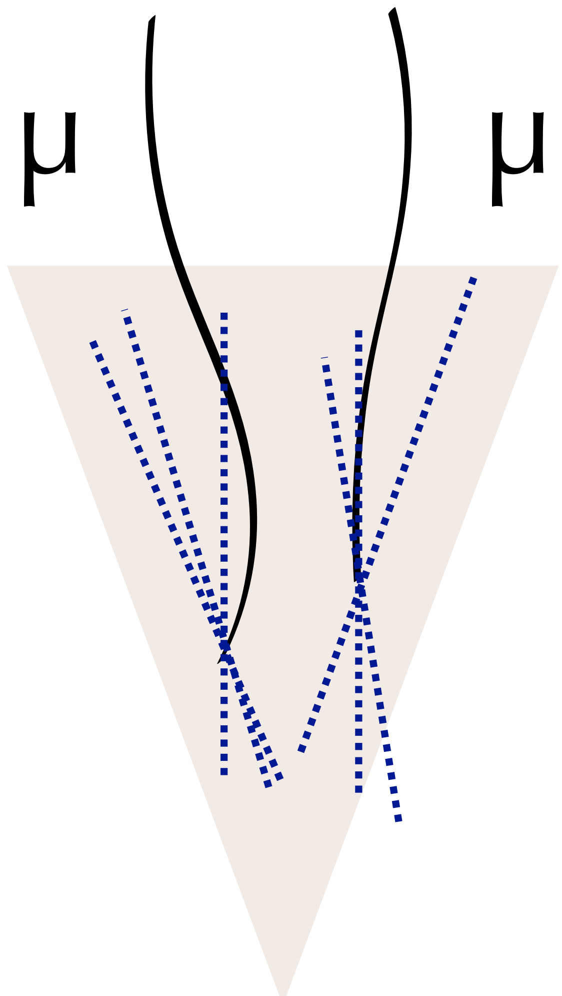
TRAINING

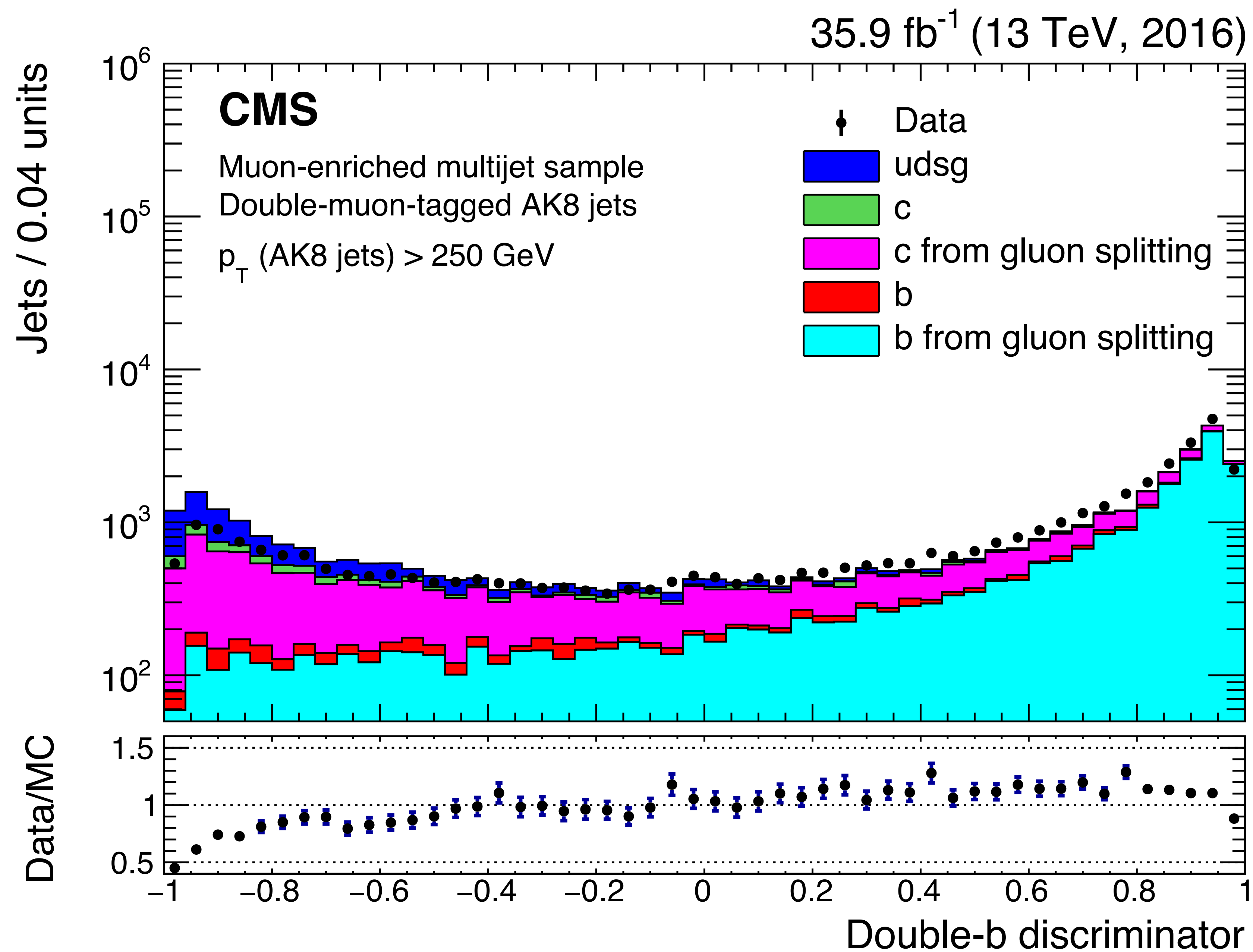
- A network is trained by specifying inputs, targets, and a loss function
  - Target is what the network should learn for that input, can be a "truth" label (supervised) or the input itself (unsupervised)
  - Loss function quantifies how many mistakes the network makes
- Training is the minimization of the loss function by varying the network parameters





EFFICIENCY IN DATA





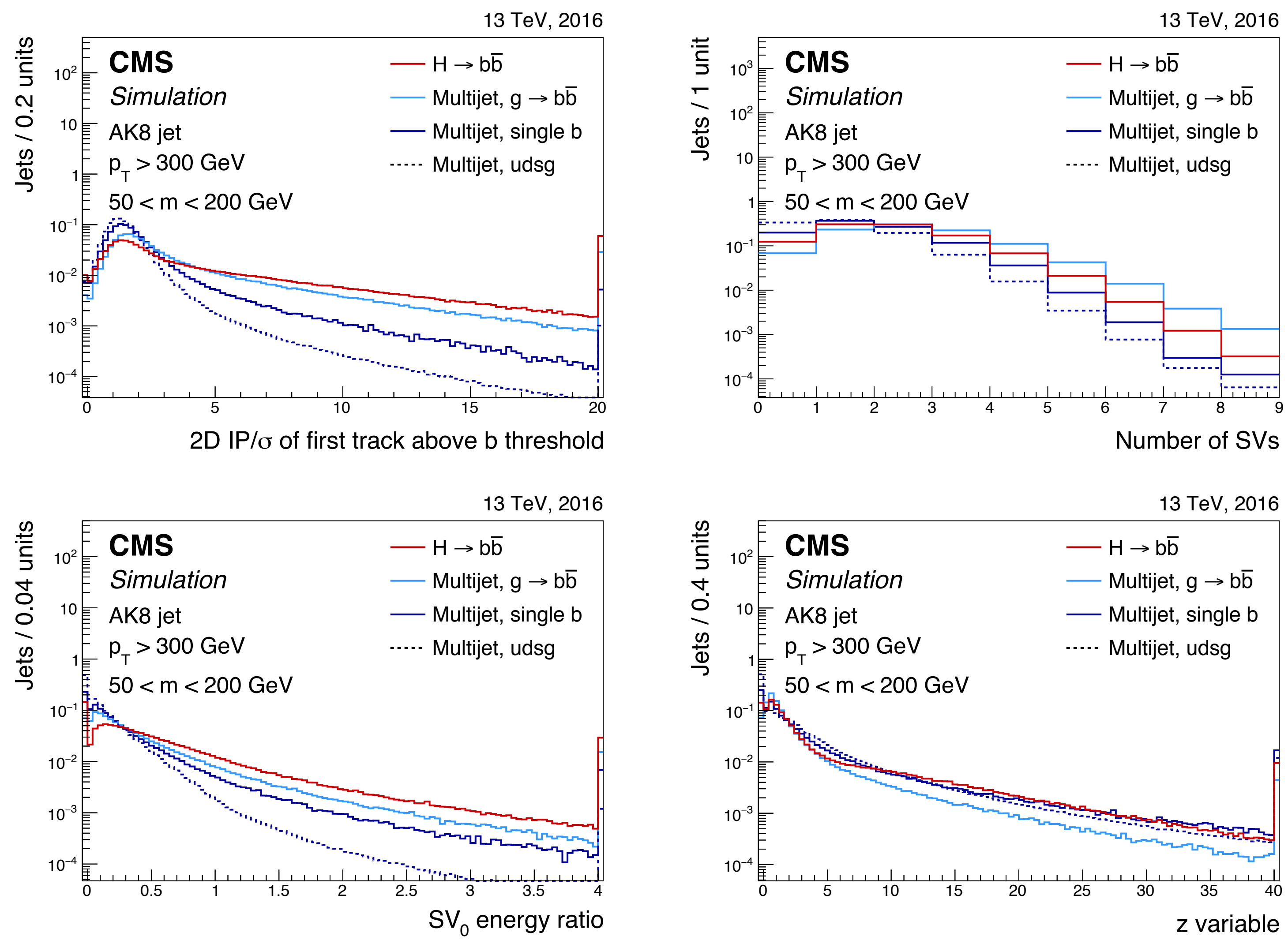
- Using g→bb jets as a proxy in **double muon** tagged jet sample
- Associated data/MC uncertainty 3-5%

DOUBLE-B TAGGER INPUTS

- The first four SIP values for selected tracks ordered in decreasing SIP;
- For each τ-axis we consider the first two SIP values for their respective associated tracks ordered in decreasing SIP, to further discriminate against single b quark and light flavor jets from QCD when one or both SV are not reconstructed due to IVF inefficiencies;
- The measured IP significance in the plane transverse to the beam axis, 2D SIP, of the first two tracks (first track) that raises the SV invariant mass above the bottom (charm) threshold of 5.2 (1.5) GeV;
- The number of SV associated to the jet;
- The significance of the 2D distance between the primary vertex and the secondary vertex, flight distance, for the SV with the smallest 3D flight distance uncertainty, for each of the two τ-axes;
- The ΔR between the SVs with the smallest 3D flight distance uncertainty and its τ-axis, for each of the two τ-axes;
- The relative pseudorapidity, η_{rel}, of the tracks from all SVs with respect to their τ-axis for the three leading tracks ordered in increasing η_{rel}, for each of the two τ-axes;
- The total SV mass, defined as the total mass of all SVs associated to a given τ-axis, for each of the two τ-axes;
- The ratio of the total SV energy, defined as the total energy of all SVs associated to a given τ-axis, and the total energy of all the tracks associated to the fat jet that are consistent with the primary vertex, for each of the two τ-axes;
- The information related to the two-SV system, the z variable, defined as:

$$z = \Delta R(\mathrm{SV}_0, \mathrm{SV}_1) \cdot \frac{p_{\mathrm{T,SV}_1}}{m(\mathrm{SV}_0, \mathrm{SV}_1)} \qquad (2)$$

where SV_0 and SV_1 are SVs with the smallest 3D flight distance uncertainty. The z variable helps rejecting the $\mathrm{b\bar{b}}$ background from gluon splitting relying on the different kinematic properties compared to the $\mathrm{b\bar{b}}$ pair from the decay of a massive resonance.



ADDITIONAL DEEP AK8 / DEEP DOUBLE-B TAGGER INPUTS

Table 10: Full list of charged PF candidate features used as input to the DeepAK8 network

feature	comment
trackEtaRel	BTV
trackPtRatio	BTV
trackPParRatio	BTV
trackSip2dVal	BTV
trackSip2dSig	BTV
trackSip3dVal	BTV
trackSip3dSig	BTV
trackJetDistVal	BTV
$p_T(cPF)/p_T(j)$	
$E_{rel}(cPF)$	
$\Delta\phi(cPF, j)$	
$\Delta\eta(cPF, j)$	
$\Delta R(cPF, j)$	
$\Delta R_m(cPF, SV)$	
$\Delta R(cPF, \text{subjet 1})$	
$\Delta R(cPF, \text{subjet 2})$	
χ^2_n	
quality	
d_z	
S_z	
d_{xy}	
S_{xy}	
track_dptdpt	track covariance
track_detadeta	track covariance
track_dphidphi	track covariance
track_dxydxy	track covariance
track_dzdz	track covariance
track_dxydz	track covariance
track_dphidxy	track covariance
track_dlambdadz	track covariance

Table 11: Full list of inclusive PF candidate features used as input to the DeepAK8 network

feature
$p_T(PF)/p_T(j)$
$E_{rel}(PF)$
$\Delta\phi(PF, j)$
$\Delta\eta(PF, j)$
$\Delta R(PF, j)$
$\Delta R_m(PF, SV)$
$\Delta R(PF, \text{subjet 1})$
$\Delta R(PF, \text{subjet 2})$
$w_p(PF)$
f_{HCAL}

Table 12: Full list of secondary vertex features used as input to the DeepAK8 network

feature
$p_T(SV)/p_T(j)$
$E_{rel}(SV)$
$\Delta\phi(SV, j)$
$\Delta\eta(SV, j)$
$\Delta R(SV, j)$
$p_T(SV)$
m_{SV}
$N_{\text{tracks}}(SV)$
$\chi^2_n(SV)$
$d_{xy}(SV)$
$S_{xy}(SV)$
$d_{3D}(SV)$
$S_{3D}(SV)$
$\cos\theta(SV)$