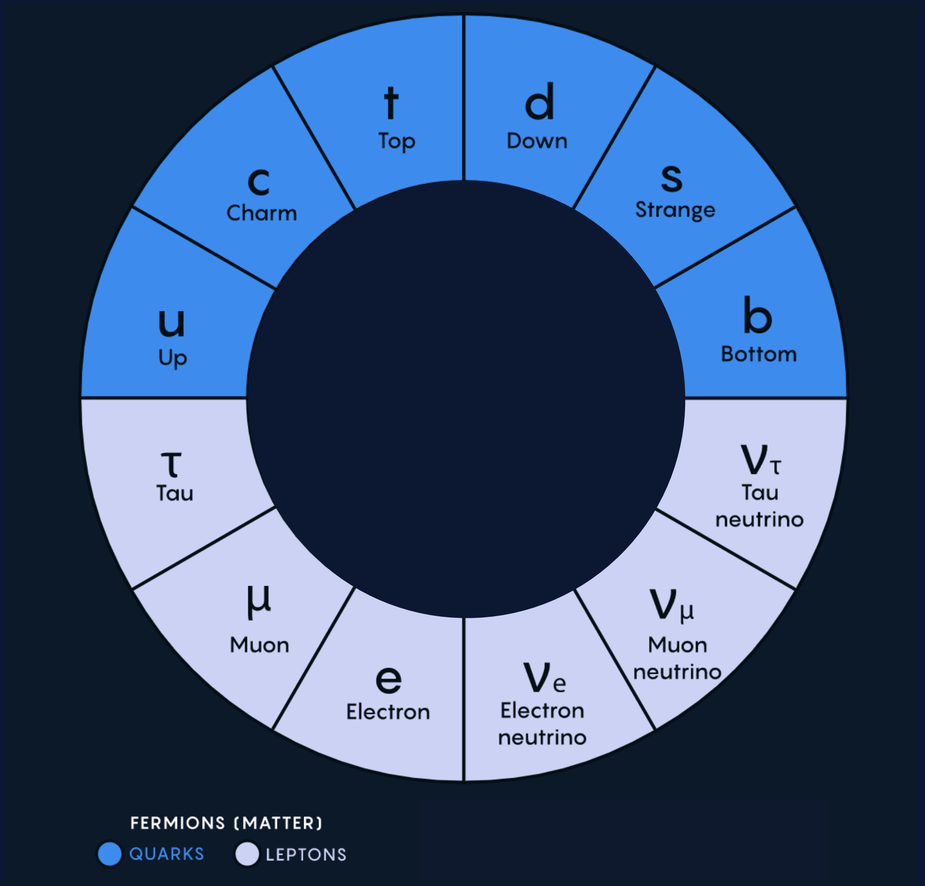
JAVIER DUARTE
UNIVERSITY OF MINNESOTA HEP SEMINAR
APRIL 28, 2021

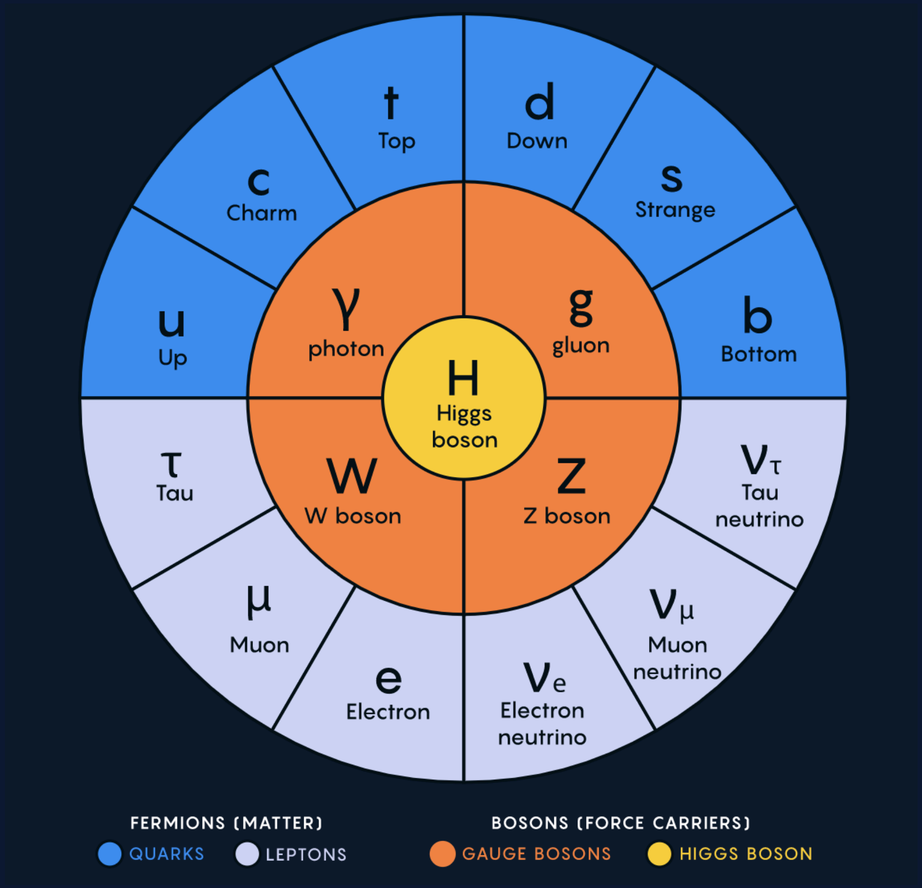
GRAPH NEURAL NETWORKS FOR PARTICLE PHYSICS

UCSD

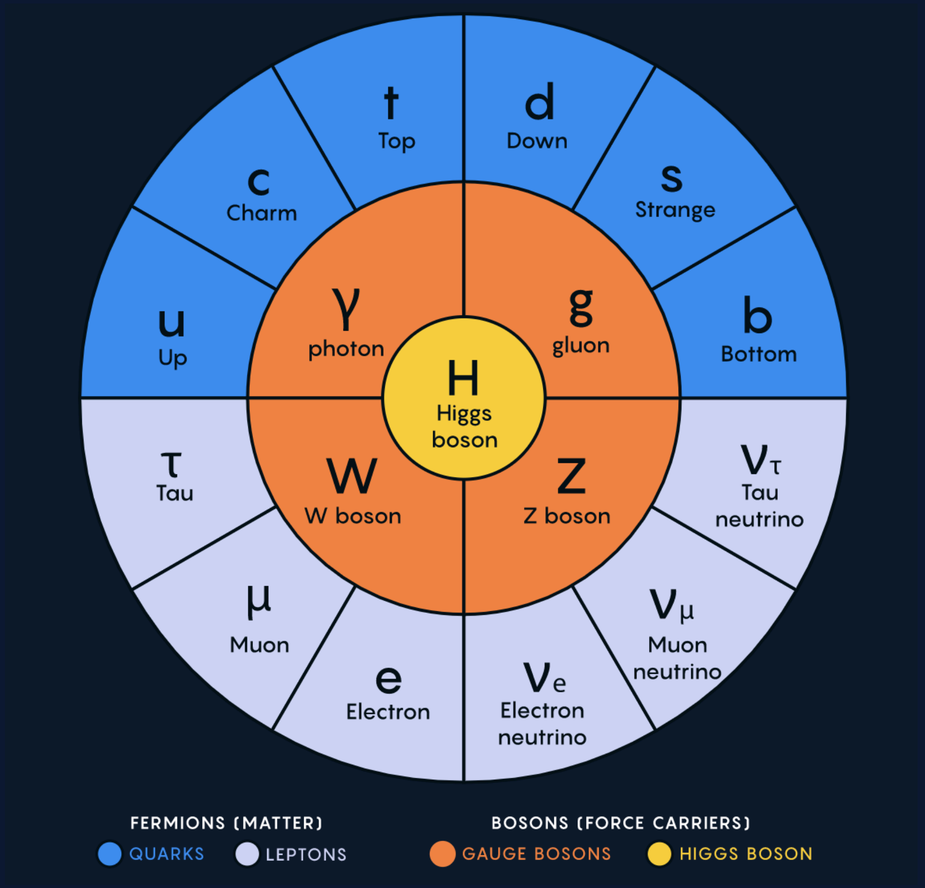
THE STANDARD MODEL (OR WHY STUDY PARTICLE PHYSICS?)

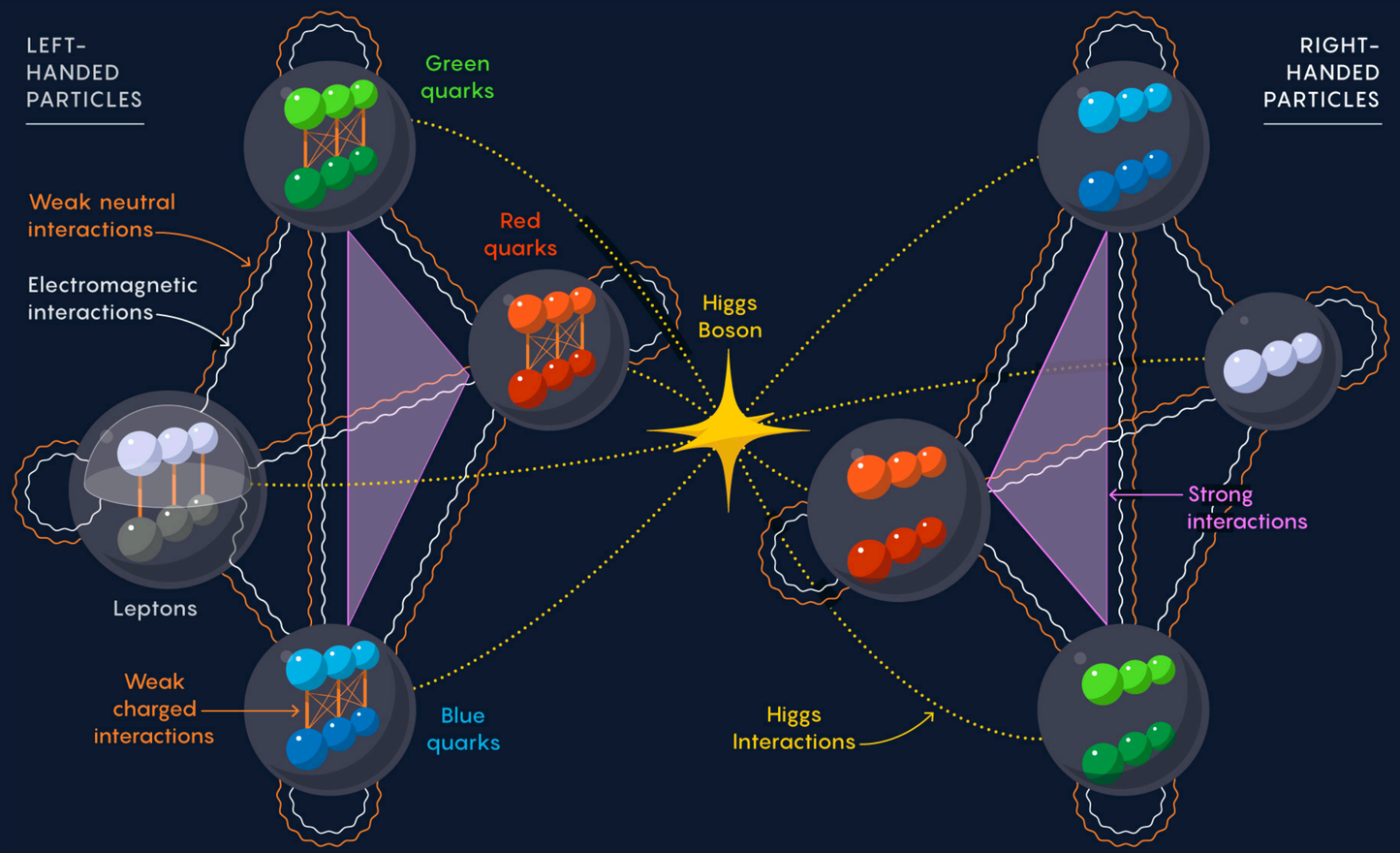


THE STANDARD MODEL (OR WHY STUDY PARTICLE PHYSICS?)

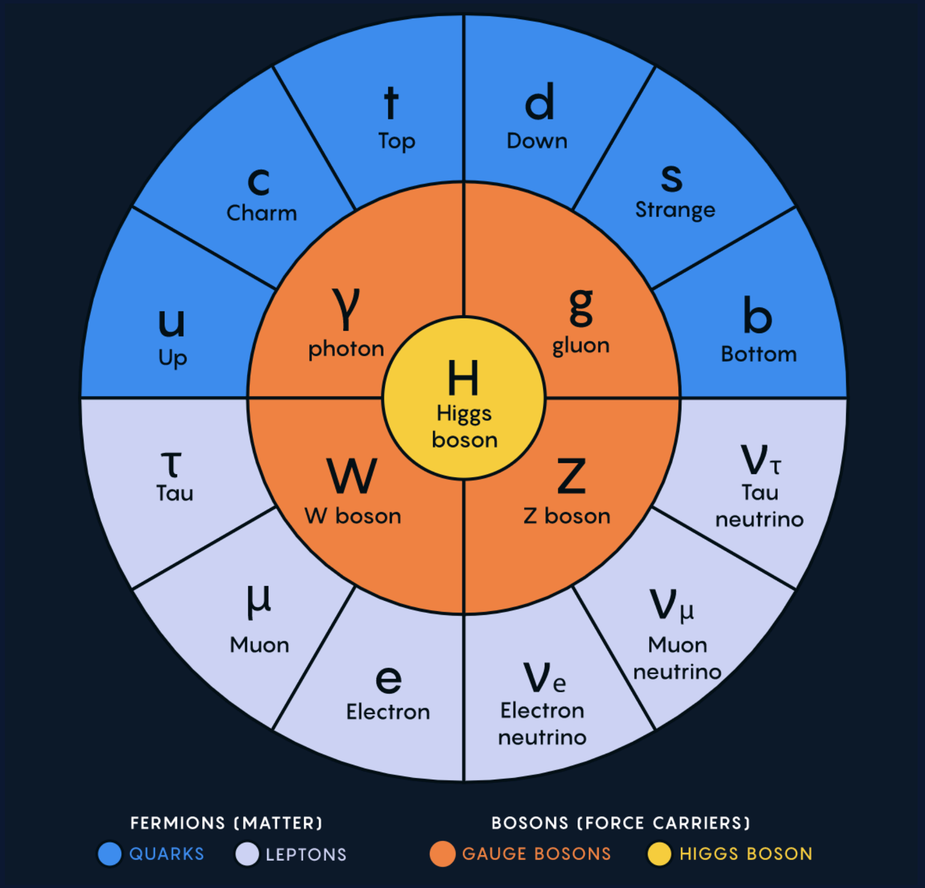


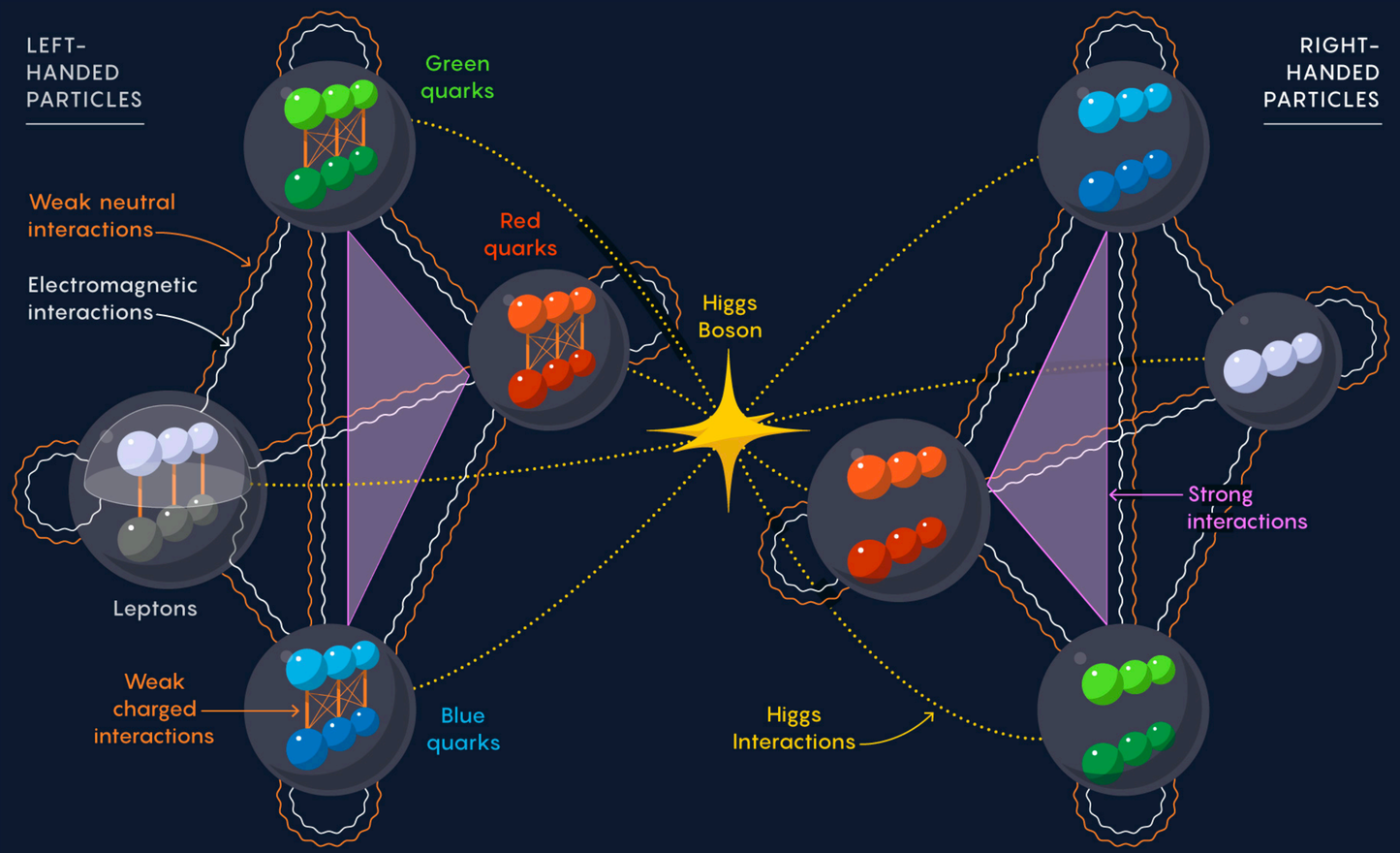
THE STANDARD MODEL (OR WHY STUDY PARTICLE PHYSICS?)





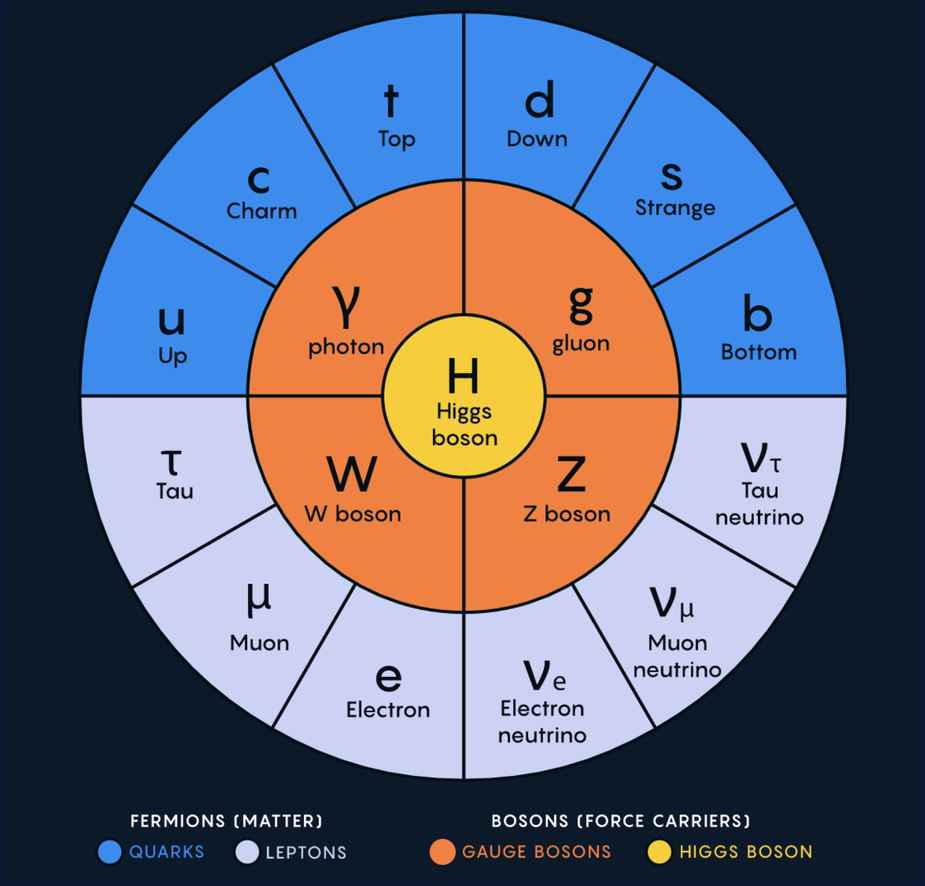
THE STANDARD MODEL (OR WHY STUDY PARTICLE PHYSICS?)

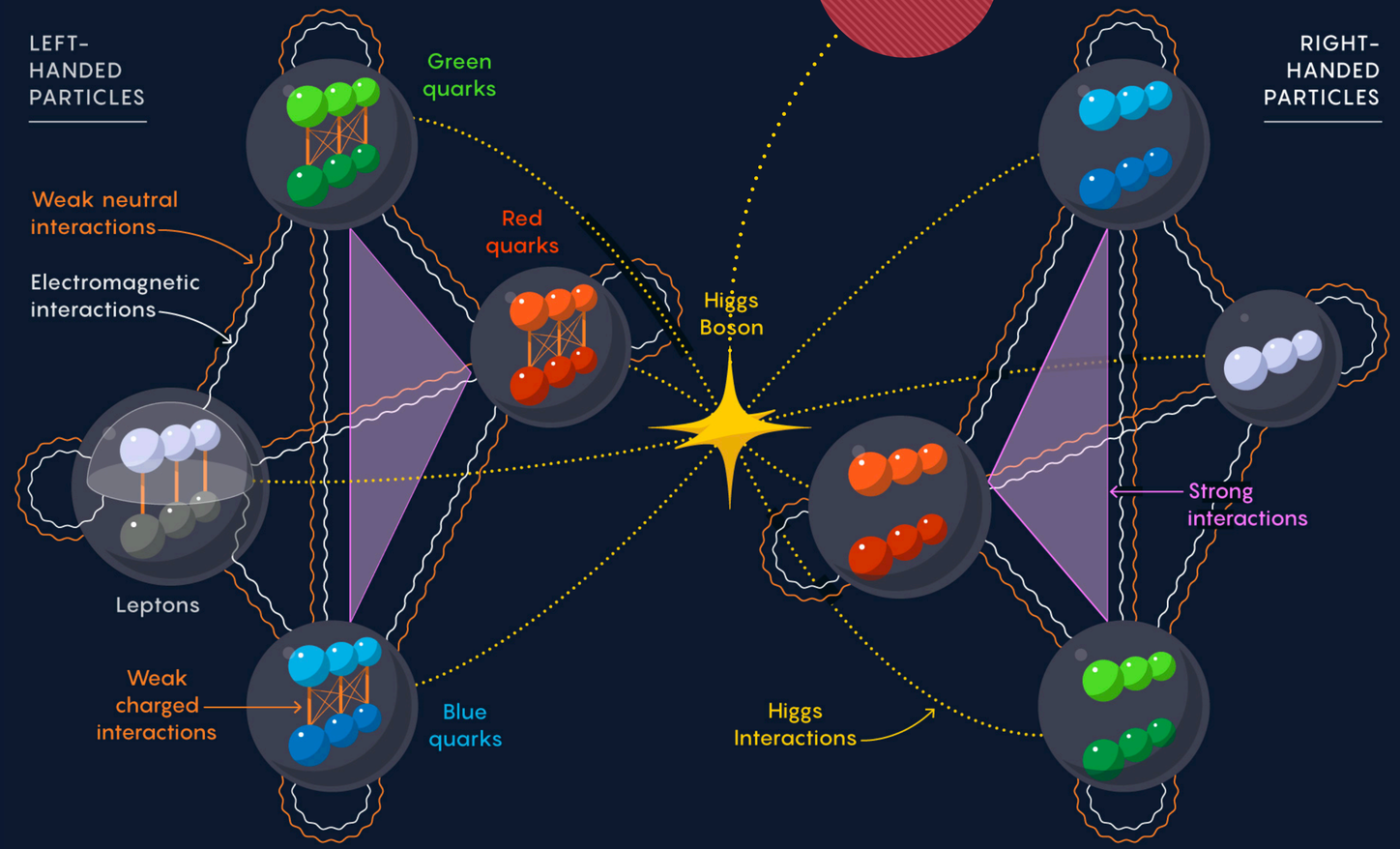




- But there has to be more to it! SM does not explain **dark matter**, **neutrino masses**, and the **matter-antimatter asymmetry**…
- Higgs boson is the ***centerpiece:*** all particles interact with it

THE STANDARD MODEL (OR WHY STUDY PARTICLE PHYSICS?)





- But there has to be more to it! SM does not explain **dark matter**, **neutrino masses**, and the **matter-antimatter asymmetry**…
- Higgs boson is the ***centerpiece:*** all particles interact with it
- May be a link to new particles or interactions

THE LARGE HADRON COLLIDER

3

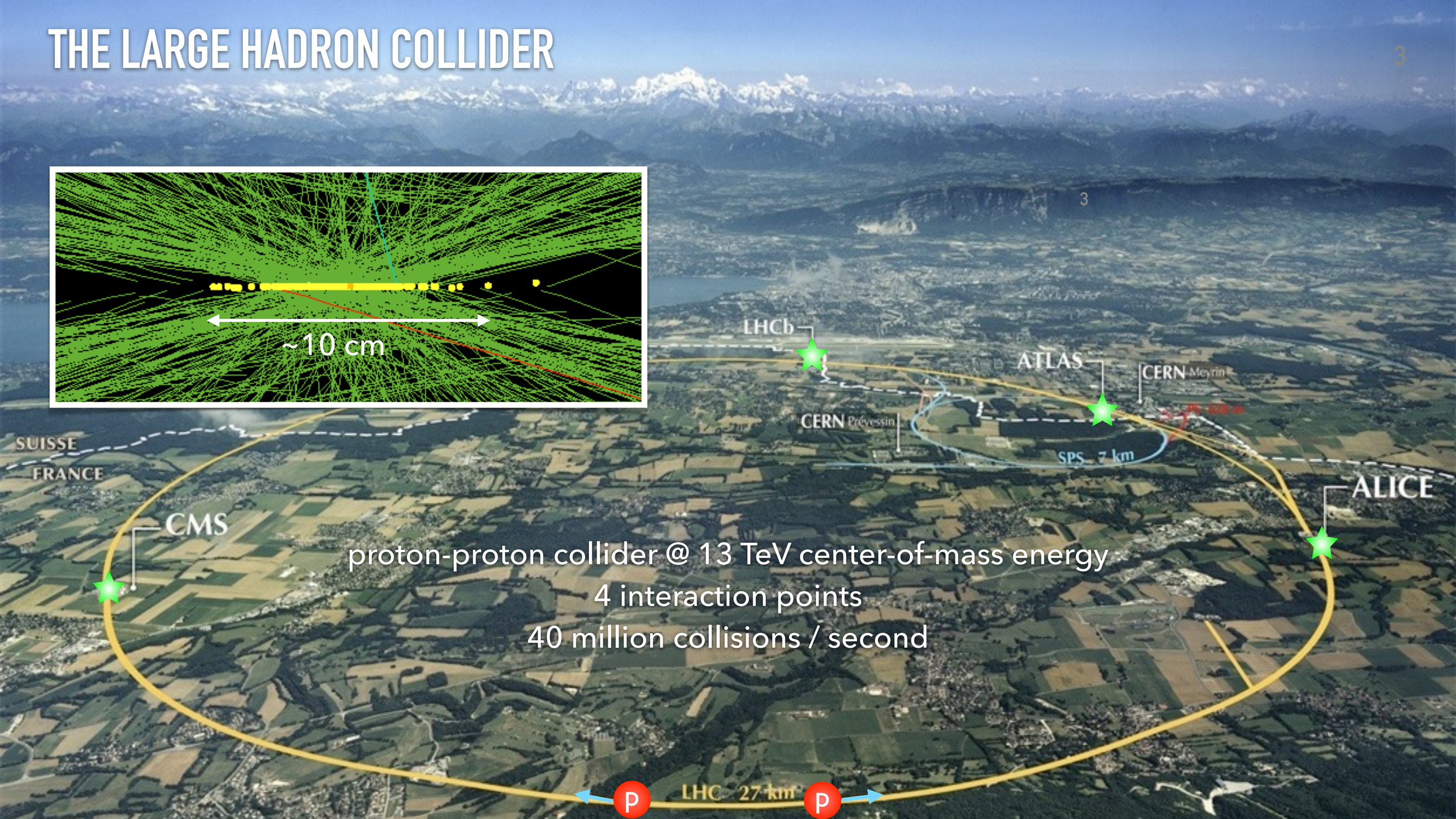
THE LARGE HADRON COLLIDER

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

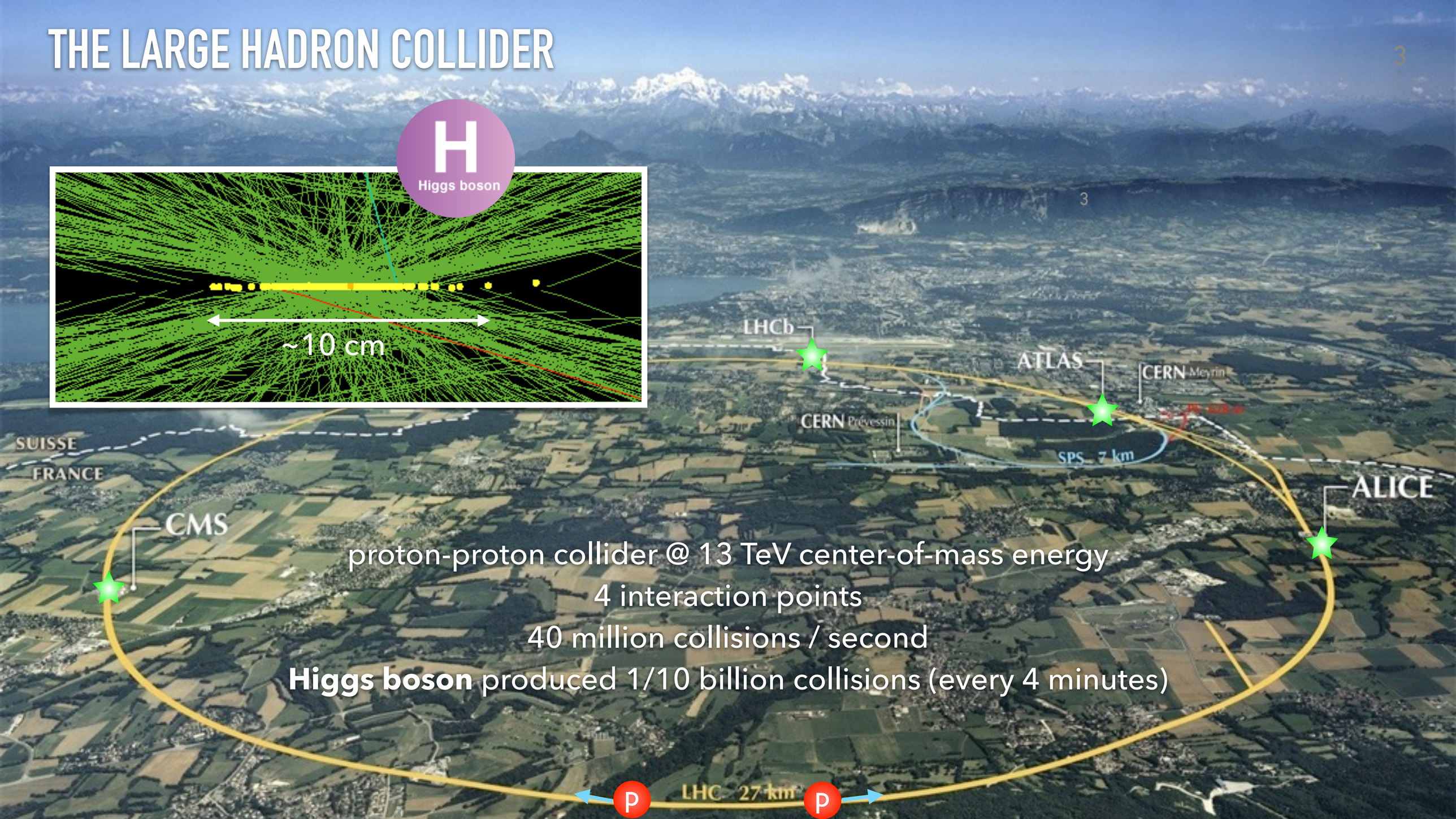


proton-proton collider @ 13 TeV center-of-mass energy

4 interaction points

40 million collisions / second

THE LARGE HADRON COLLIDER



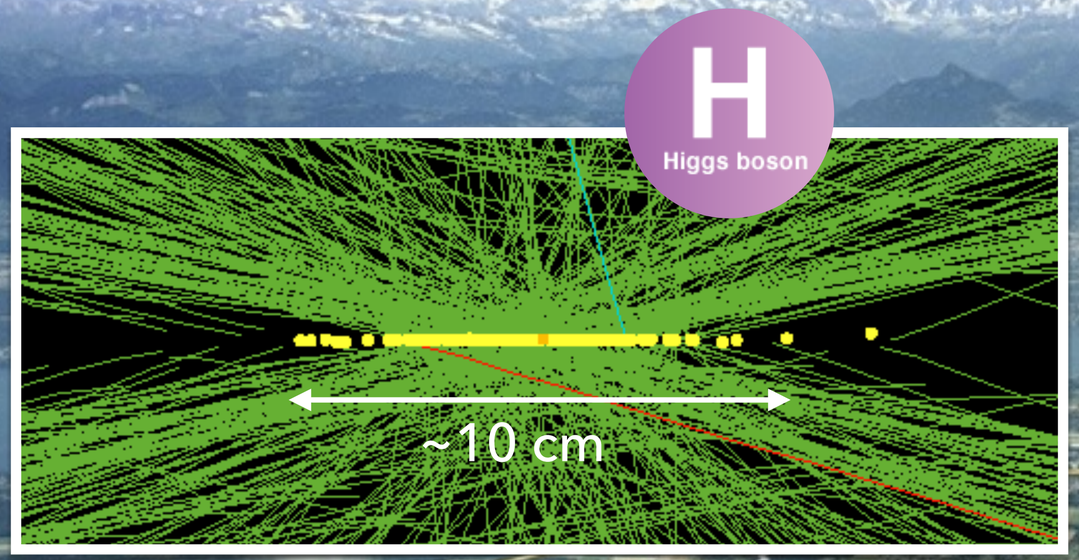
proton-proton collider @ 13 TeV center-of-mass energy

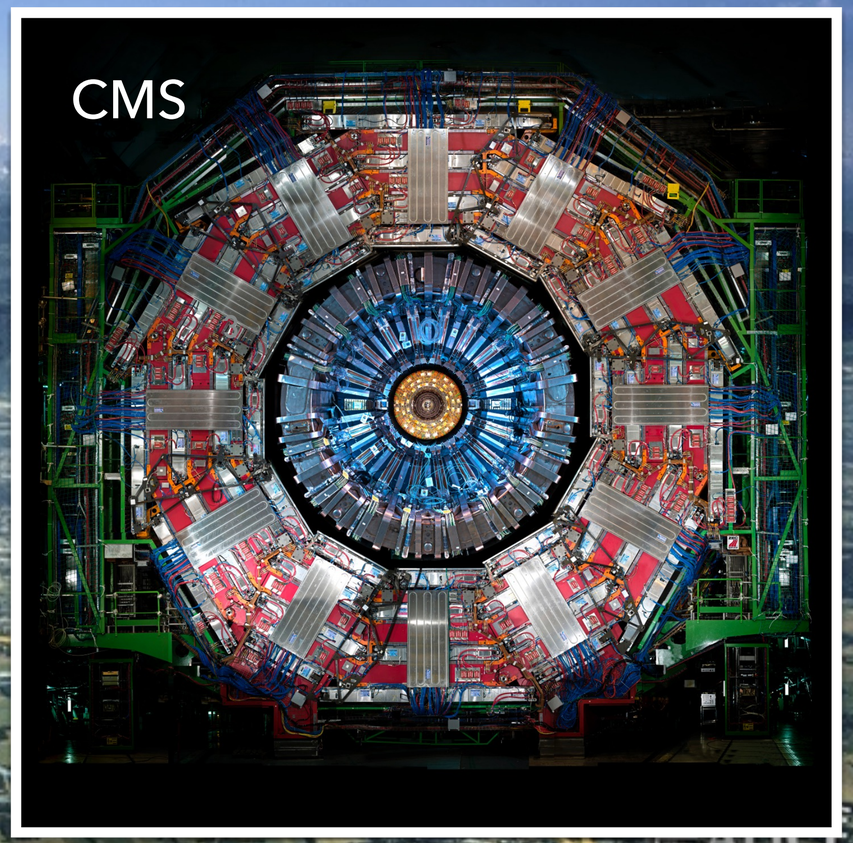
4 interaction points

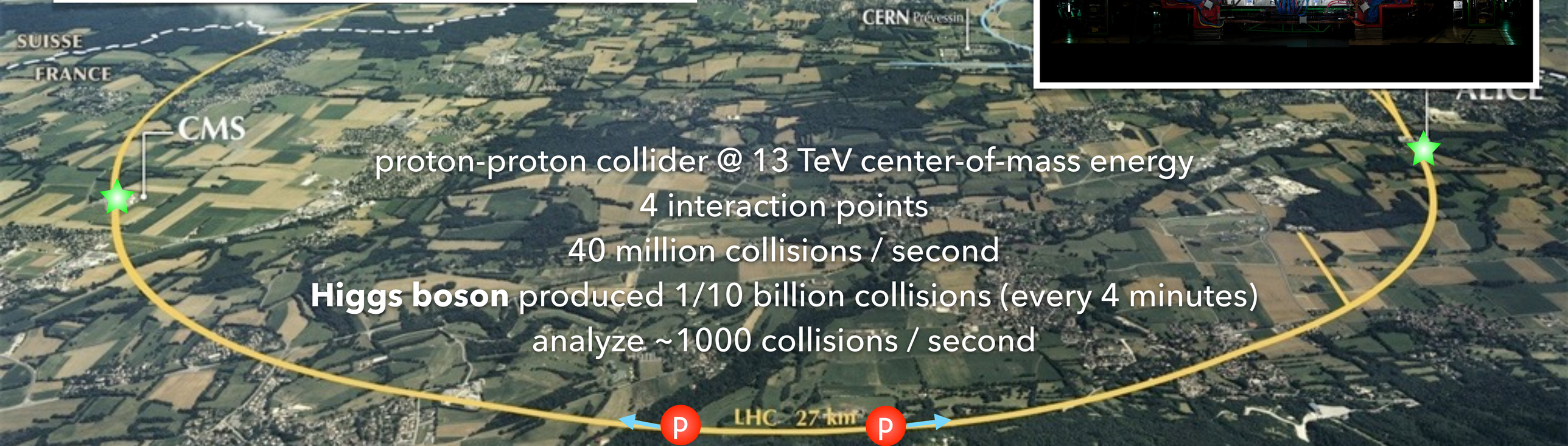
40 million collisions / second

Higgs boson produced 1/10 billion collisions (every 4 minutes)

THE LARGE HADRON COLLIDER







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

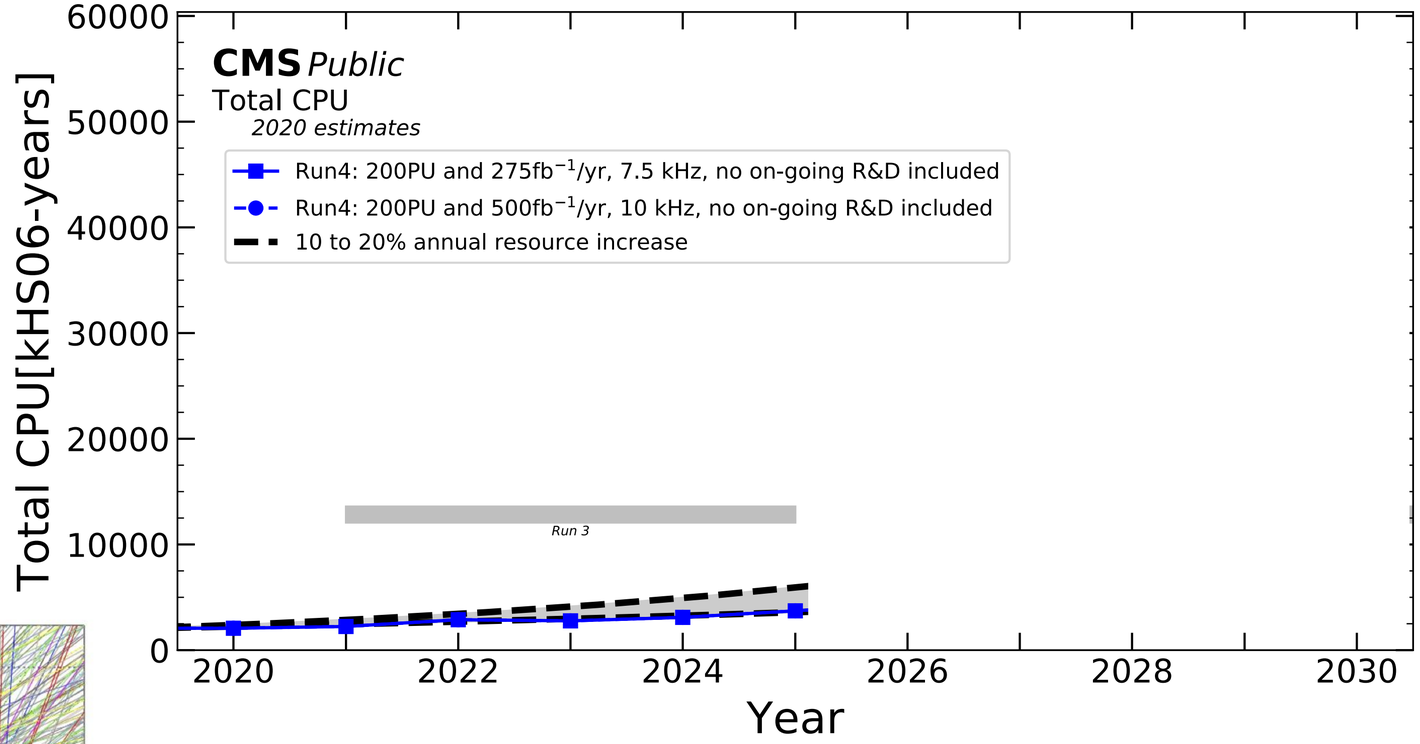
4 interaction points

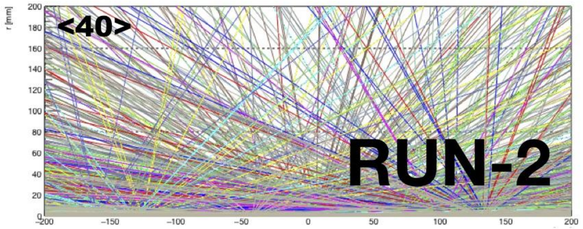
40 million collisions / second

Higgs boson produced 1/10 billion collisions (every 4 minutes)

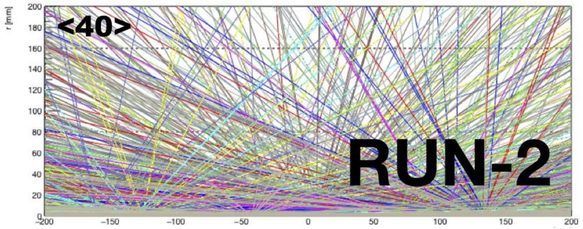
analyze ~1000 collisions / second

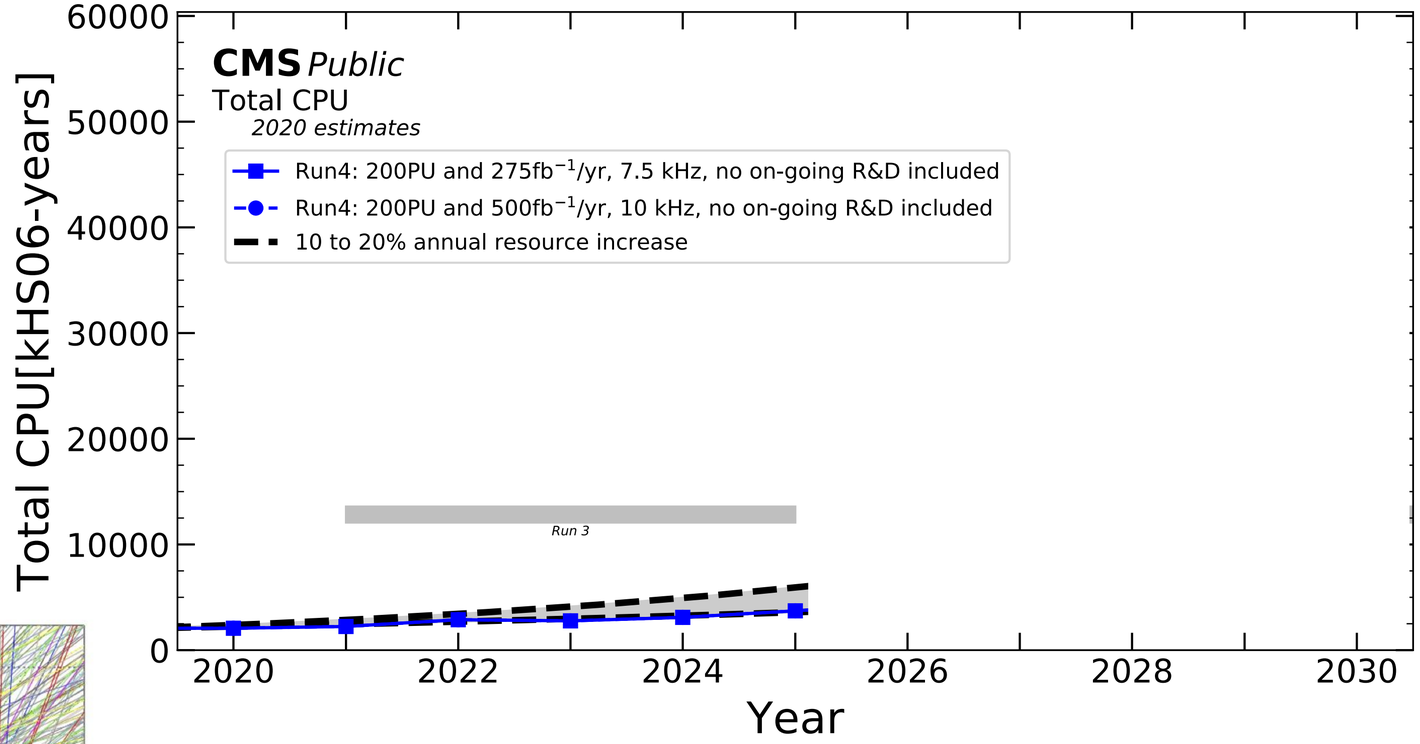
MOTIVATION 1: CPU DEMANDS AT THE UPGRADED LHC



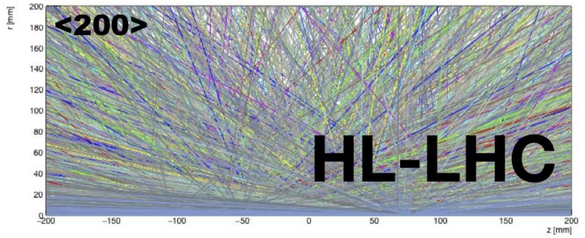


MOTIVATION 1: CPU DEMANDS AT THE UPGRADED LHC



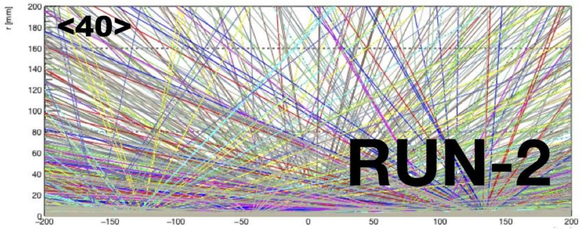


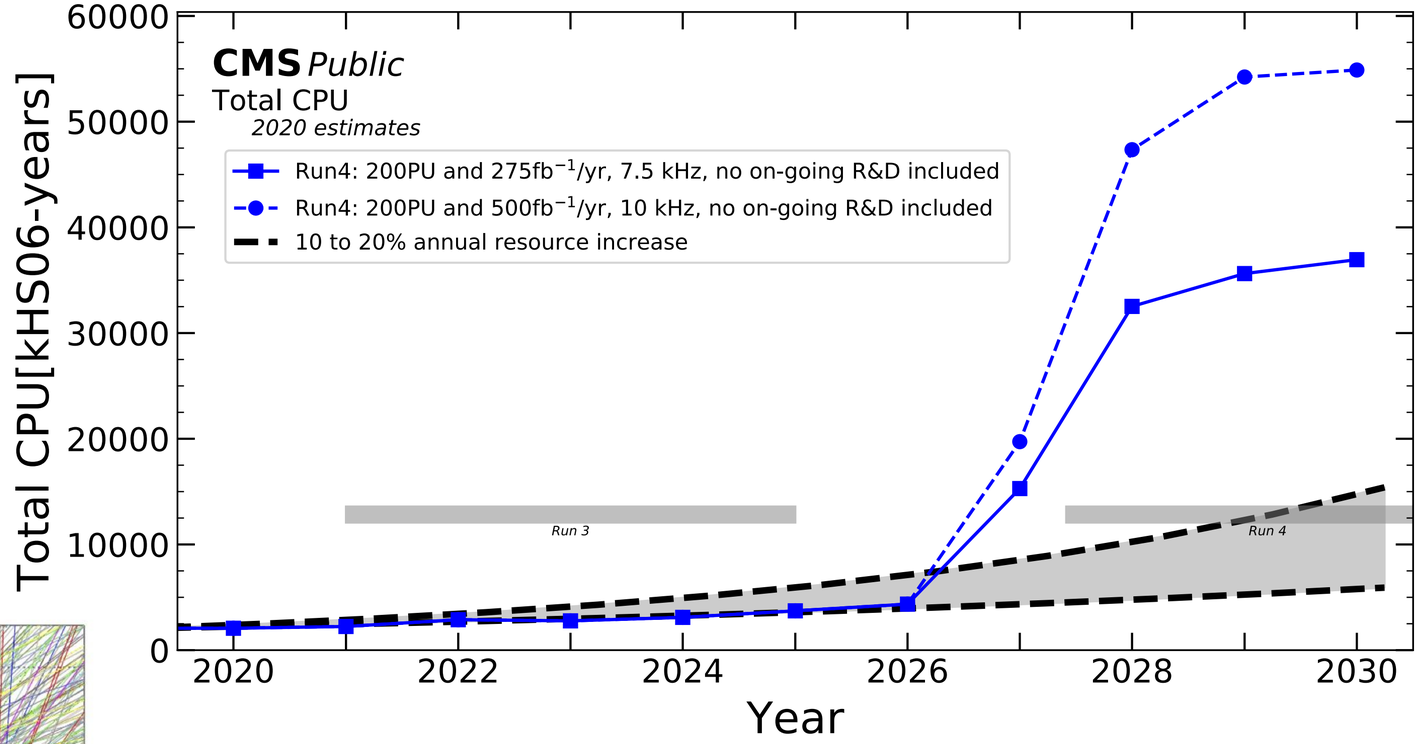
LHC upgrade



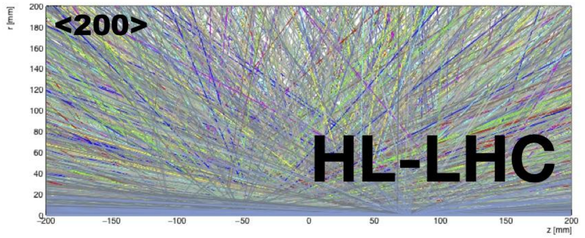
- To analyze more Higgs bosons, need more data, i.e. more **intense beams**

MOTIVATION 1: CPU DEMANDS AT THE UPGRADED LHC



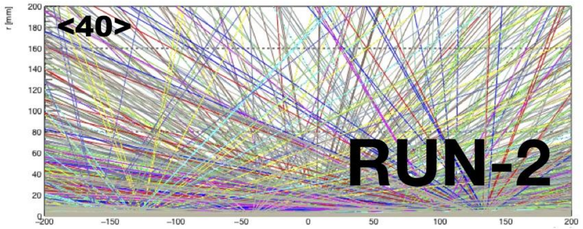


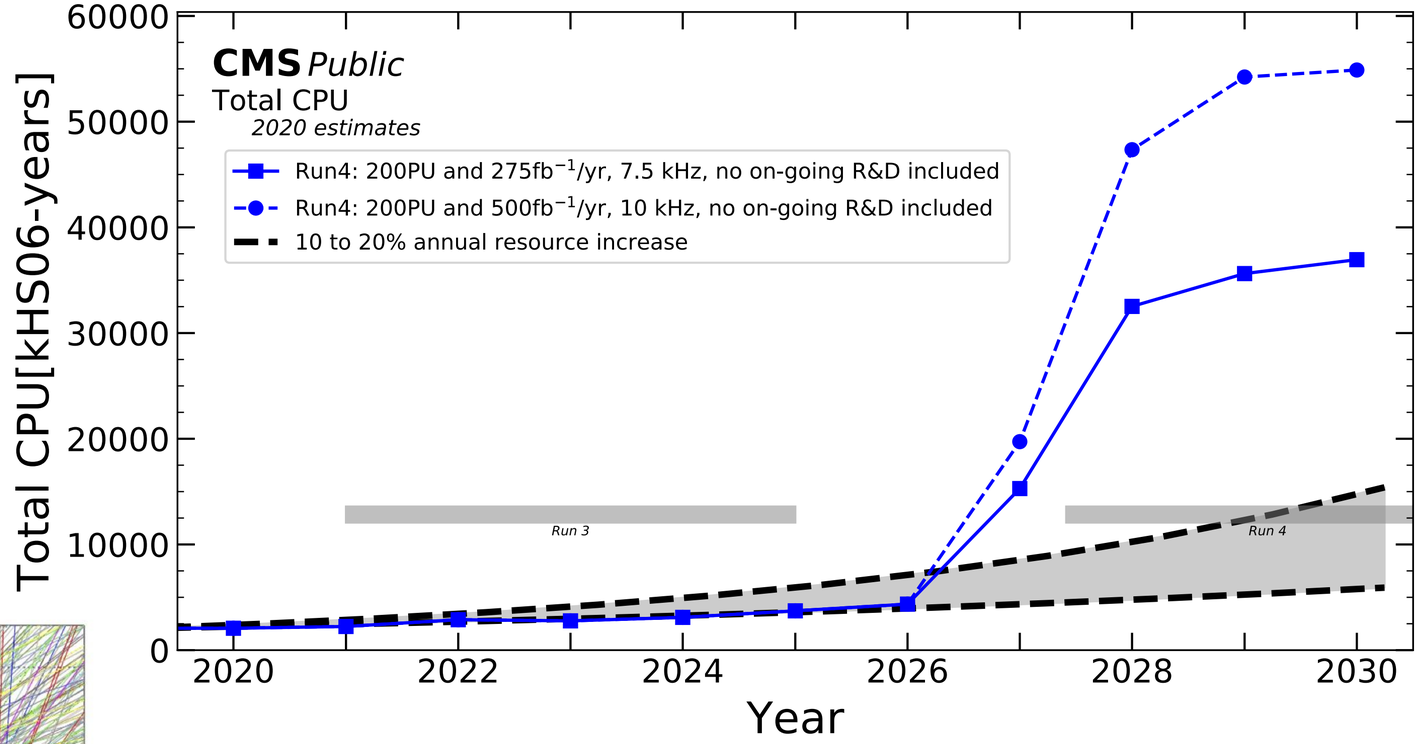
LHC upgrade



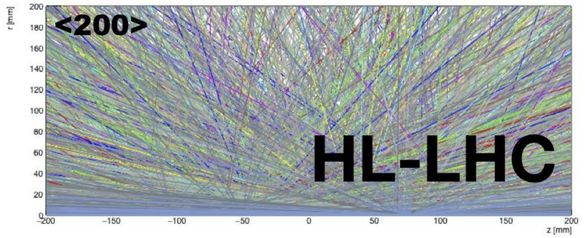
- To analyze more Higgs bosons, need more data, i.e. more **intense beams**
- Computing demands (for event reco.) increase nonlinearly with increasing “pileup”

MOTIVATION 1: CPU DEMANDS AT THE UPGRADED LHC





LHC upgrade



- To analyze more Higgs bosons, need more data, i.e. more **intense beams**
- Computing demands (for event reco.) increase nonlinearly with increasing “pileup”
- Need more processing power (or smarter algorithms like **deep learning**) to keep up with data demands

WHY DEEP LEARNING?

WHY DEEP LEARNING?

- Deep learning is the use of **structured neural networks** with **many hidden layers** as generic functions to approximate the optimal solution for a given task

WHY DEEP LEARNING?

- Deep learning is the use of **structured neural networks** with **many hidden layers** as generic functions to approximate the optimal solution for a given task
- Why use deep neural networks in particle physics?

WHY DEEP LEARNING?

- Deep learning is the use of **structured neural networks** with **many hidden layers** as generic functions to approximate the optimal solution for a given task
- Why use deep neural networks in particle physics?
  - They work!

WHY DEEP LEARNING?

- Deep learning is the use of **structured neural networks** with **many hidden layers** as generic functions to approximate the optimal solution for a given task
- Why use deep neural networks in particle physics?
  - They work!
  - Lots of data & simulation to train them

WHY DEEP LEARNING?

- Deep learning is the use of **structured neural networks** with **many hidden layers** as generic functions to approximate the optimal solution for a given task
- Why use deep neural networks in particle physics?
  - They work!
  - Lots of data & simulation to train them
  - Fast ways to train and run them

TensorFlow

Keras PyTorch

WHY DEEP LEARNING?

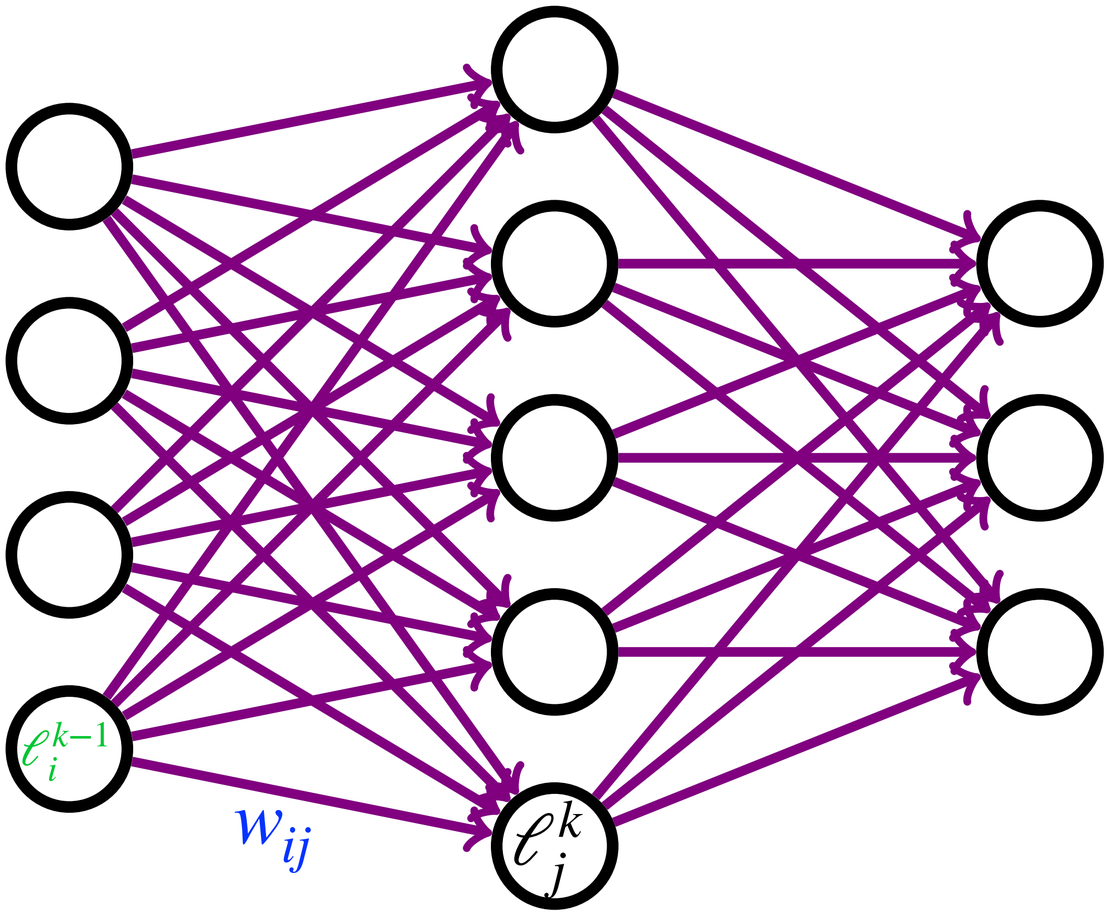
- Deep learning is the use of **structured neural networks** with **many hidden layers** as generic functions to approximate the optimal solution for a given task
- Why use deep neural networks in particle physics?
  - They work!
  - Lots of data & simulation to train them
  - Fast ways to train and run them
  - Possibly gain new insights…

TensorFlow

Keras PyTorch

NEURAL NETWORK

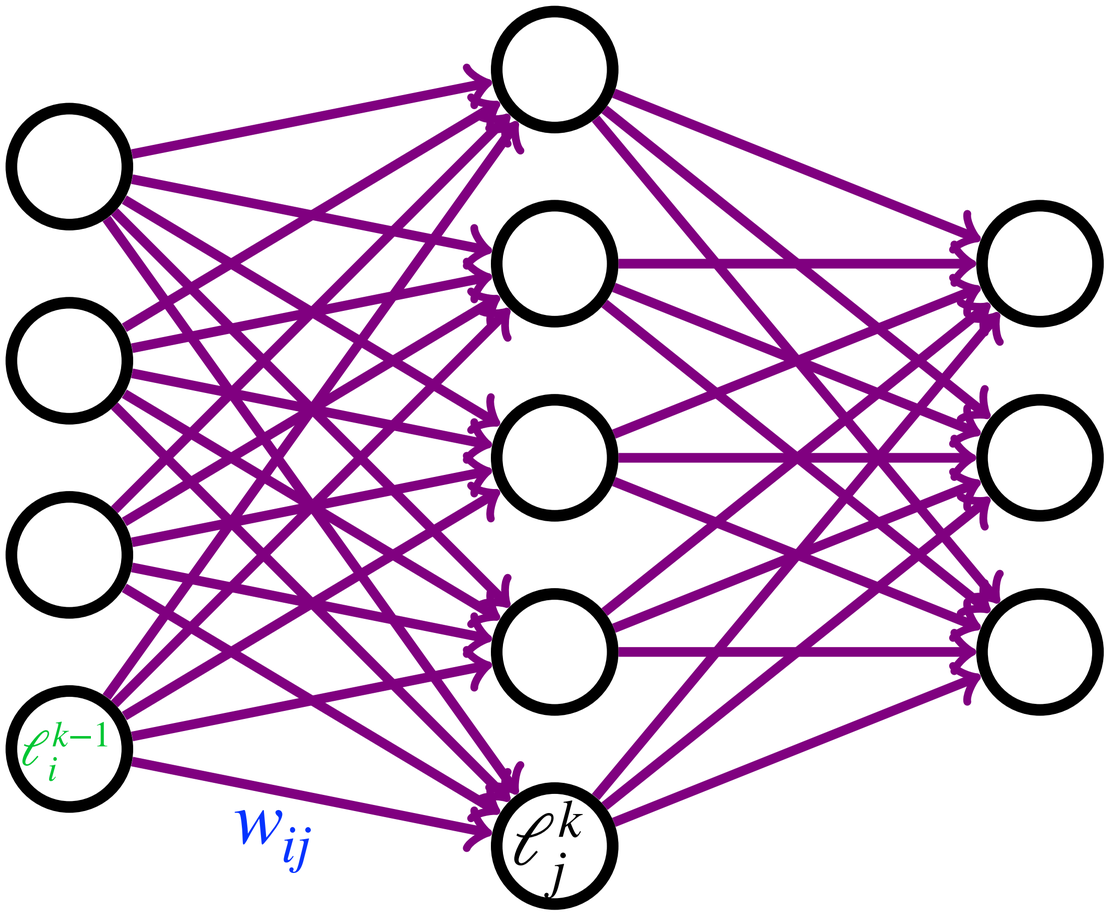
$$\ell_j^k = \phi\left(\sum_i w_{ij}\ell_i^{k-1} + b_j\right)$$



NEURAL NETWORK

- Classic fully connected architecture

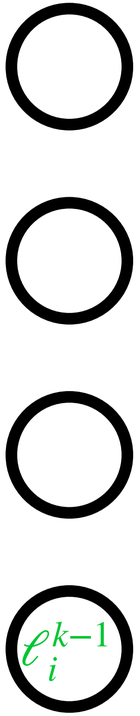
$$\ell_j^k = \phi\left(\sum_i w_{ij}\ell_i^{k-1} + b_j\right)$$



NEURAL NETWORK

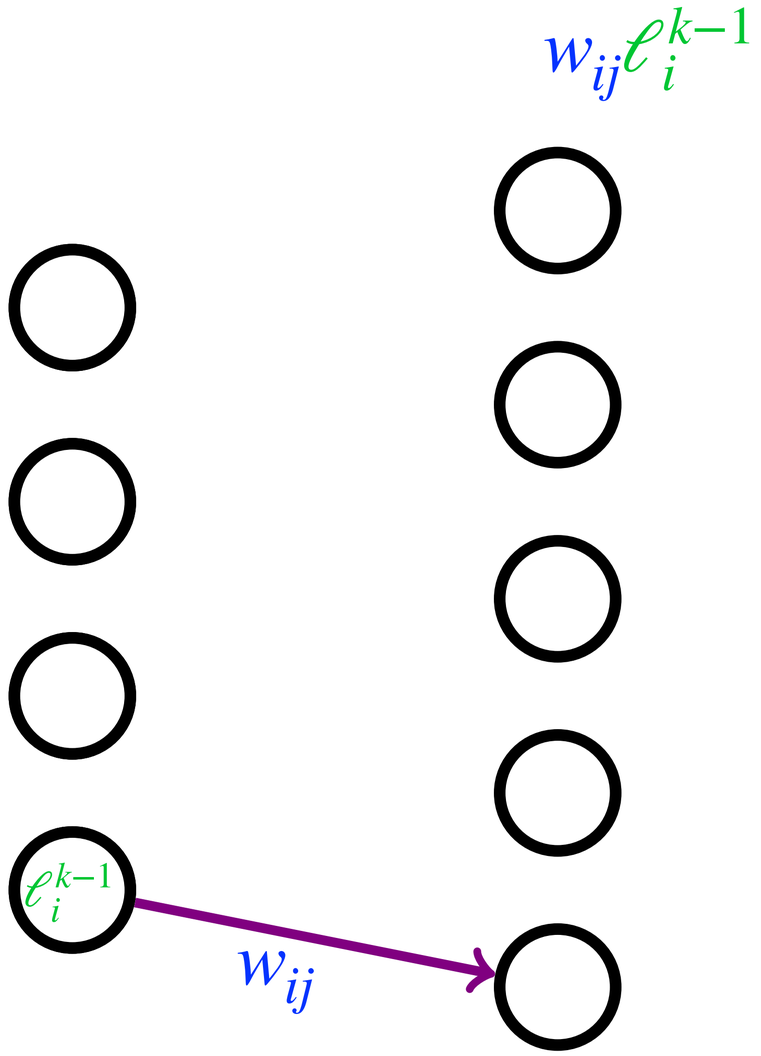
- Classic fully connected architecture

ℓ_i^{k-1}



NEURAL NETWORK

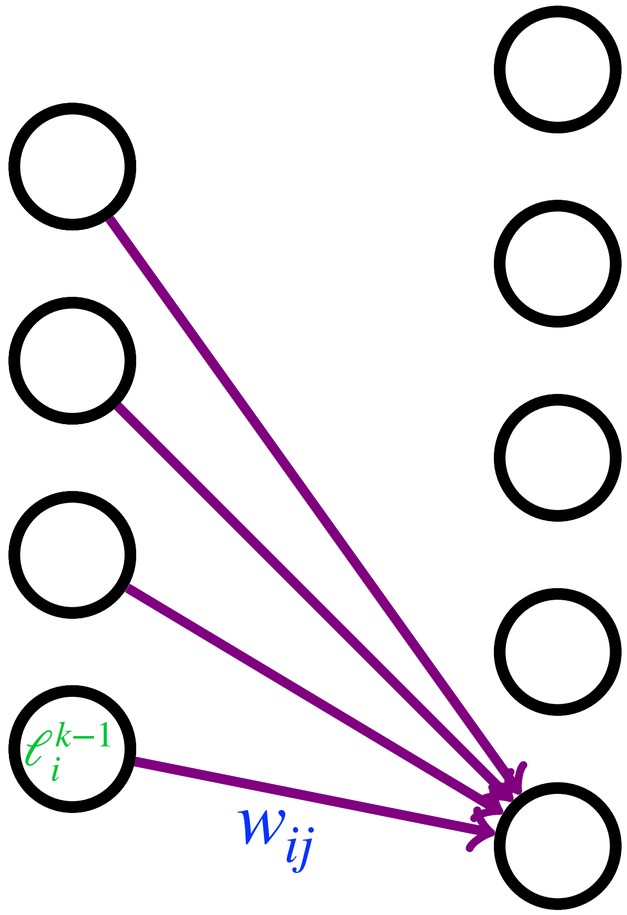
- Classic fully connected architecture
- Each **input** multiplied by a **weight**



NEURAL NETWORK

- Classic fully connected architecture
- Each **input** multiplied by a **weight**
- **Weighted** values are summed, **bias** is added

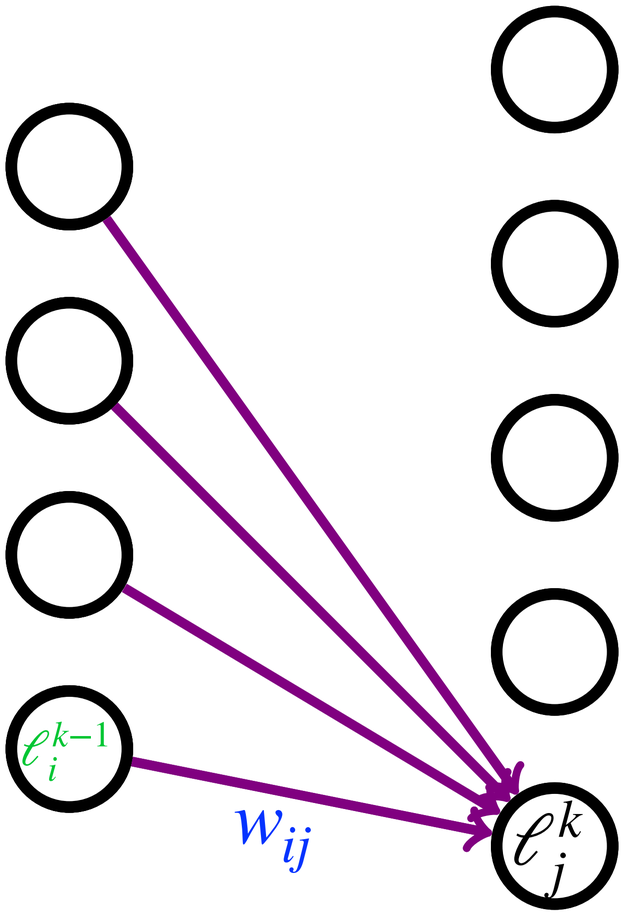
$$\sum_i w_{ij} \ell_i^{k-1} + b_j$$

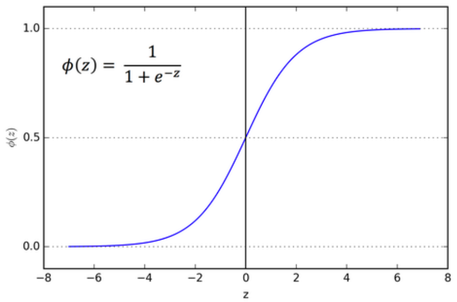


NEURAL NETWORK

- Classic fully connected architecture
- Each **input** multiplied by a **weight**
- **Weighted** values are summed, **bias** is added
- Nonlinear **activation function** is applied

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

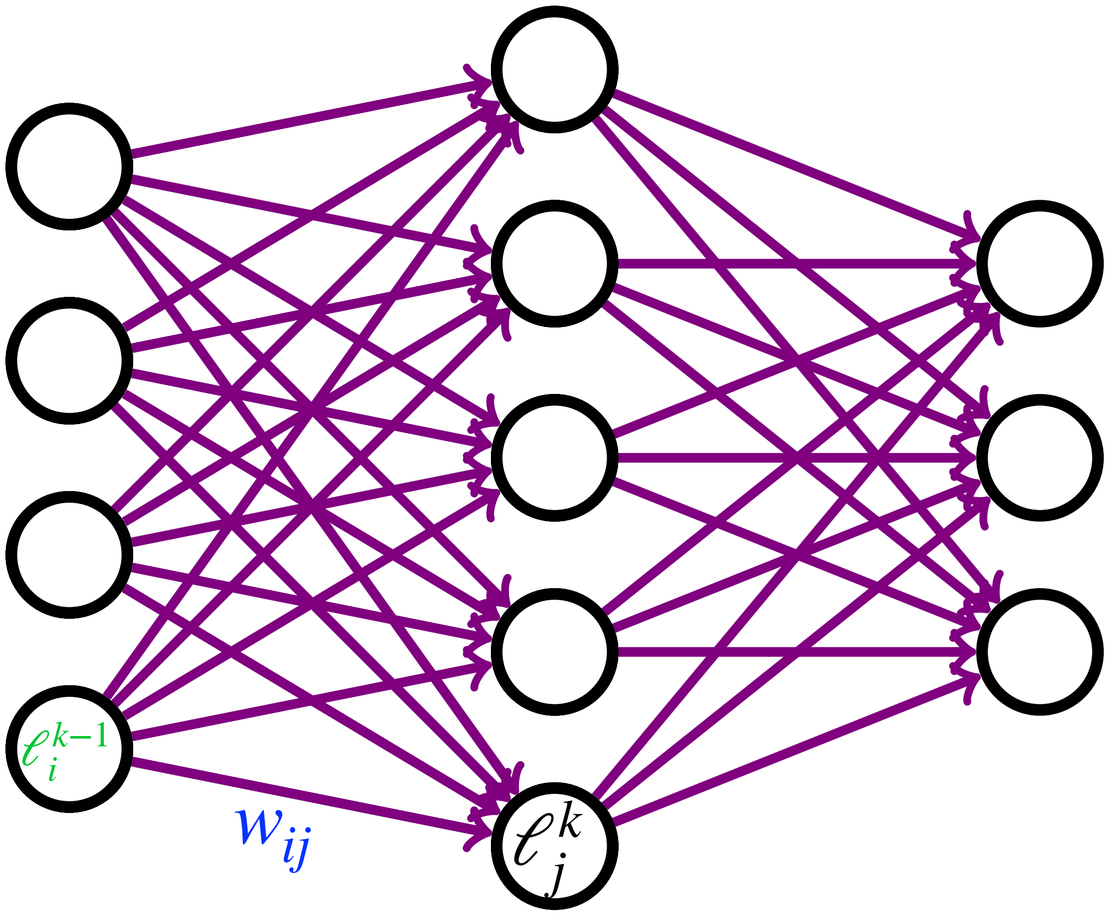


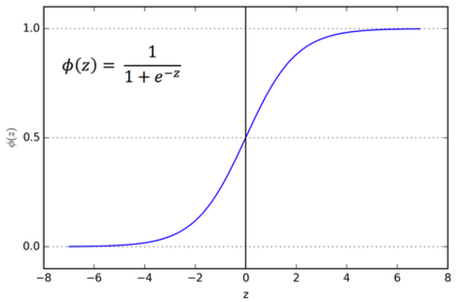


NEURAL NETWORK

- Classic fully connected architecture
- Each **input** multiplied by a **weight**
- **Weighted** values are summed, **bias** is added
- Nonlinear **activation function** is applied

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

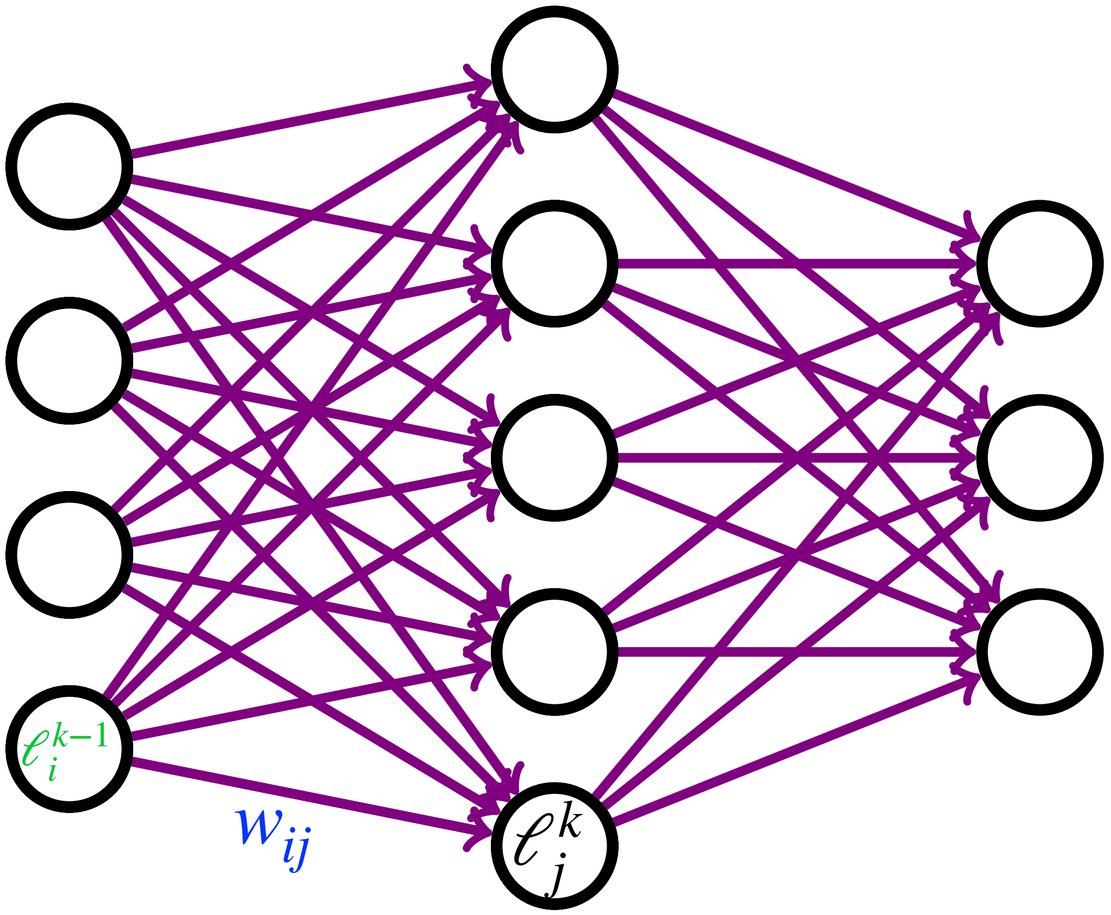


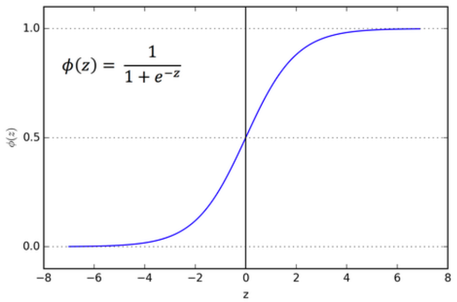


NEURAL NETWORK

- 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)

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

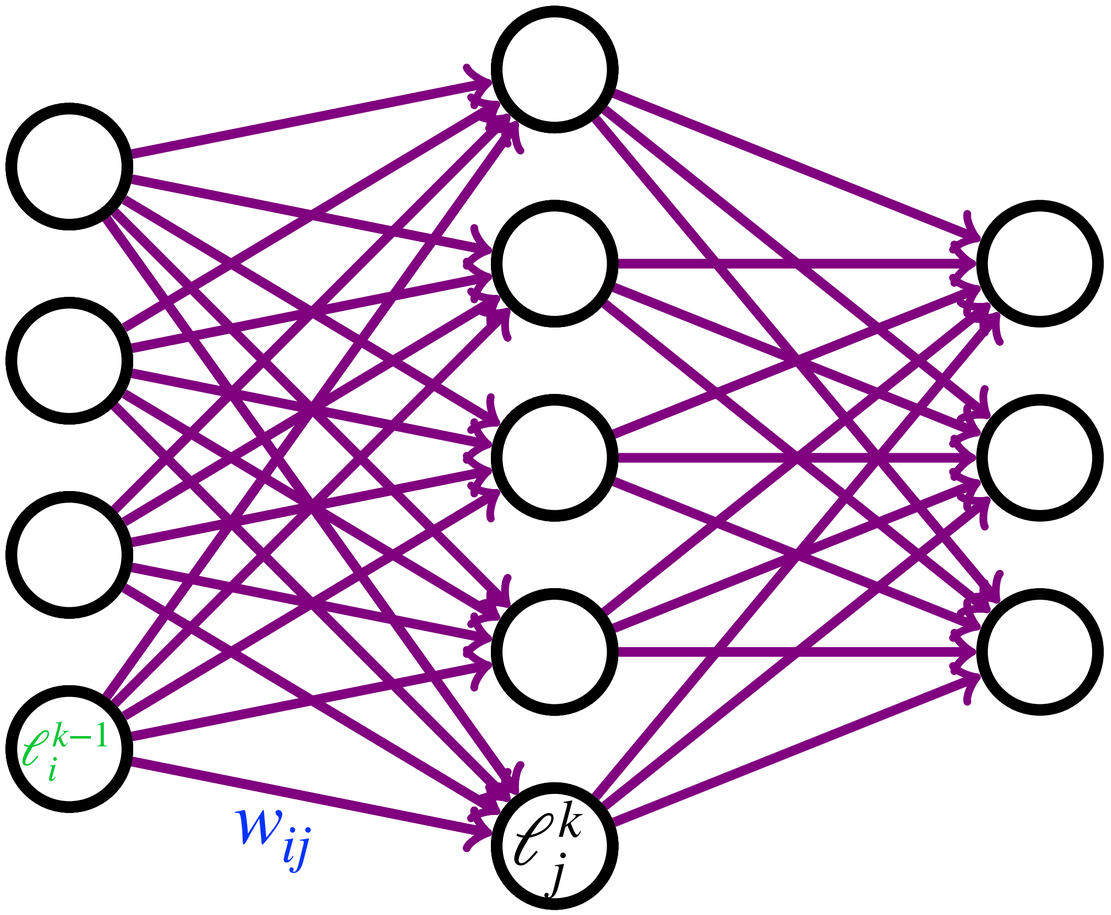


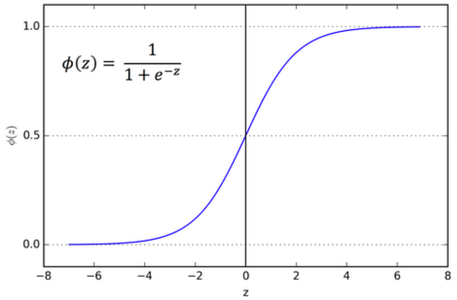


NEURAL NETWORK

- 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)

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

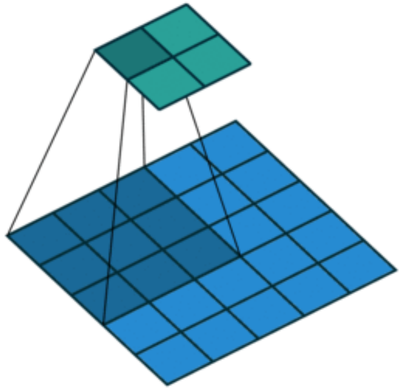




A sufficiently "wide" neural network can approximate any function!

DEEP NEURAL NETWORK: RESNET-50

- ResNet-50 is a deep convolutional neural network
  - inputs (pixels) are locally grouped and the same set of weights is applied to each patch
  - 50 hidden layers, 8 G-Ops, ~2 s on CPU
  - Good performance for natural images



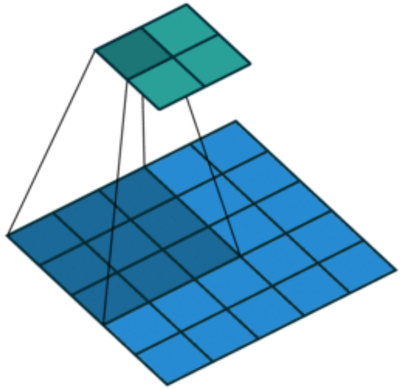
African elephant from Wikipedia

```
ResNet-50 predictions:

African elephant: 90%
Tusker: 6%
Indian elephant: 3%
```

DEEP NEURAL NETWORK: RESNET-50

- ResNet-50 is a deep convolutional neural network
  - inputs (pixels) are locally grouped and the same set of weights is applied to each patch
  - 50 hidden layers, 8 G-Ops, ~2 s on CPU
  - Good performance for natural images



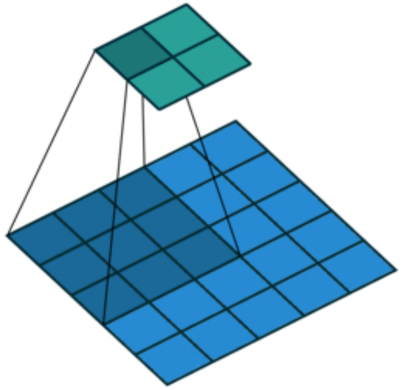
African elephant from Wikipedia

```
ResNet-50 predictions:

African elephant: 90%
Tusker: 6%
Indian elephant: 3%
```

DEEP NEURAL NETWORK: RESNET-50

- ResNet-50 is a deep convolutional neural network
  - inputs (pixels) are locally grouped and the same set of weights is applied to each patch
  - 50 hidden layers, 8 G-Ops, ~2 s on CPU
  - Good performance for natural images



African elephant from Wikipedia

```
ResNet-50 predictions:

African elephant: 90%
Tusker: 6%
Indian elephant: 3%
```

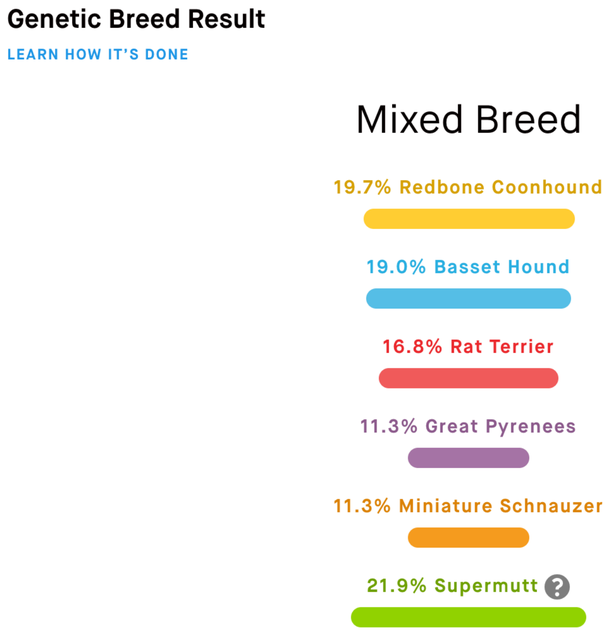
My dog: ???

```
ResNet-50 predictions:

Rhodesian ridgeback: 50%
redbone: 28%
bloodhound: 6%
```

DEEP NEURAL NETWORK: RESNET-50

- ResNet-50 is a deep convolutional neural network
  - inputs (pixels) are locally grouped and the same set of weights is applied to each patch
  - 50 hidden layers, 8 G-Ops, ~2 s on CPU
  - Good performance for natural images



African elephant from Wikipedia

```
ResNet-50 predictions:

African elephant: 90%
Tusker: 6%
Indian elephant: 3%
```

My dog: ???

```
ResNet-50 predictions:

Rhodesian ridgeback: 50%
redbone: 28%
bloodhound: 6%
```

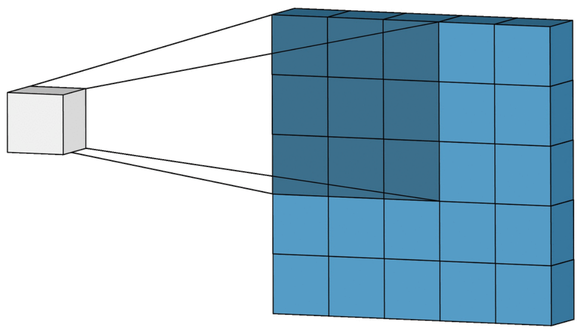
MOTIVATION 2: A NATURAL LANGUAGE

MOTIVATION 2: A NATURAL LANGUAGE

- In deep learning, tailoring algorithms to the structure (and symmetries) of the data has led to groundbreaking performance

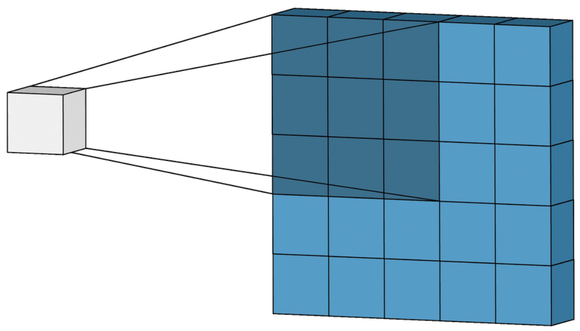
MOTIVATION 2: A NATURAL LANGUAGE

- In deep learning, tailoring algorithms to the structure (and symmetries) of the data has led to groundbreaking performance
  - CNNs for images

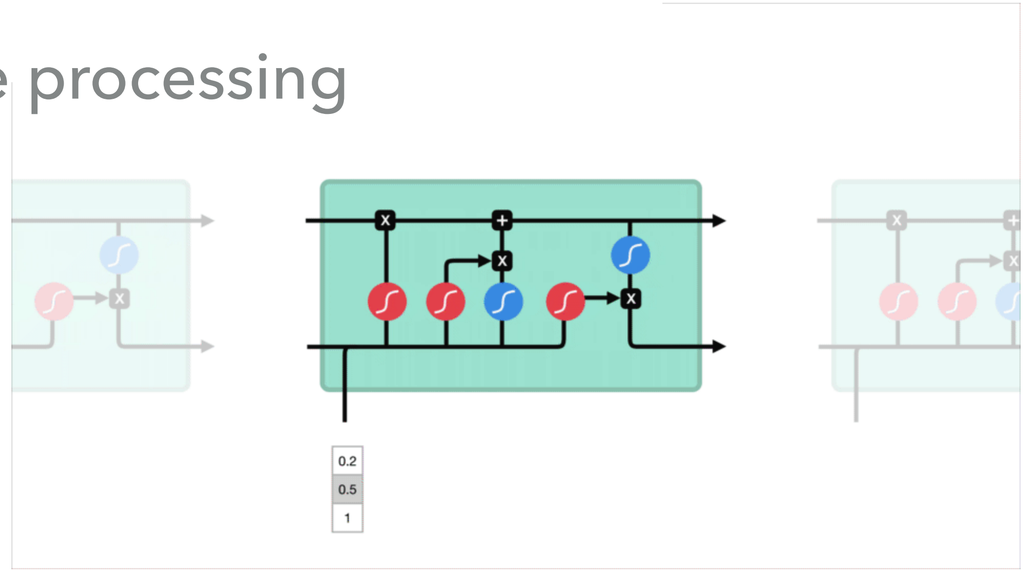


MOTIVATION 2: A NATURAL LANGUAGE

- In deep learning, tailoring algorithms to the structure (and symmetries) of the data has led to groundbreaking performance
  - CNNs for images

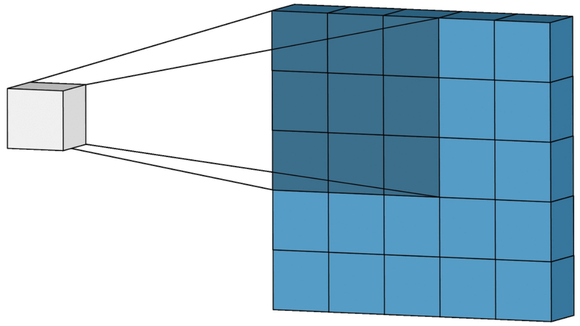


  - RNNs for language processing

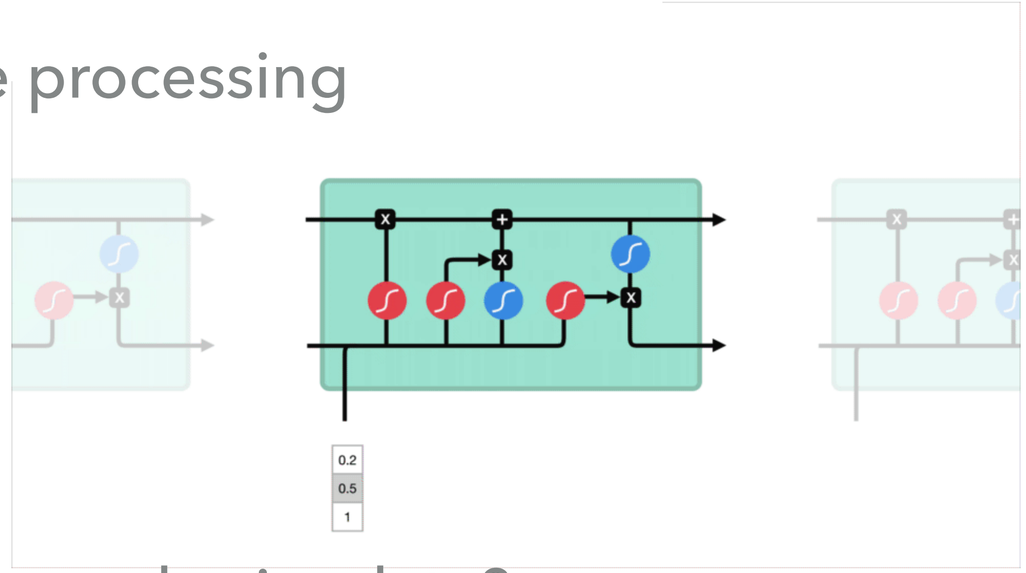


MOTIVATION 2: A NATURAL LANGUAGE

- In deep learning, tailoring algorithms to the structure (and symmetries) of the data has led to groundbreaking performance
  - CNNs for images

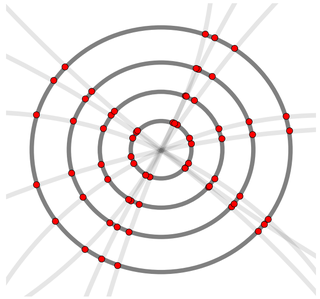


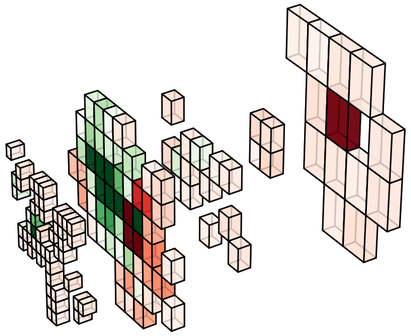
  - RNNs for language processing

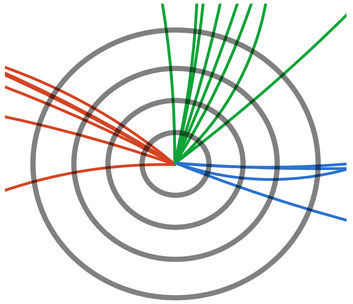


- What about high energy physics data?

HIGH ENERGY PHYSICS DATA

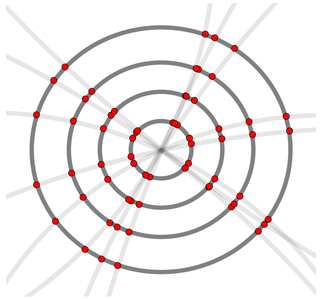




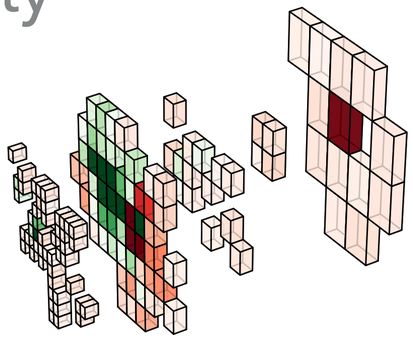


HIGH ENERGY PHYSICS DATA

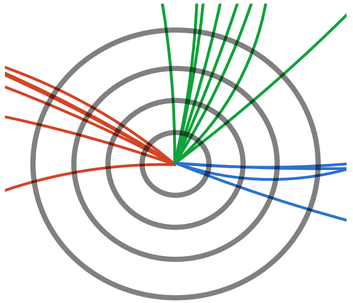
- Properties
  - Measurements distributed in space (and time) irregularly



  - Sparse (most detector channels are empty), but pockets of density



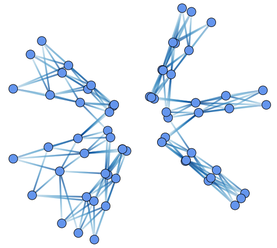
  - Complex interdependencies between measurements
  - Physics “objects” composed of multiple measurements



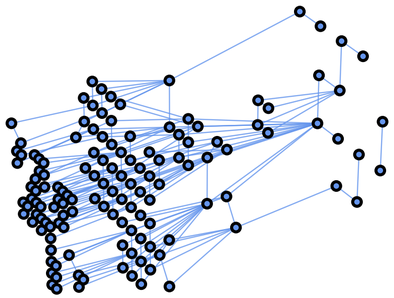
  - Inherent symmetries (Lorentz boosts, rotational)*

HIGH ENERGY PHYSICS DATA

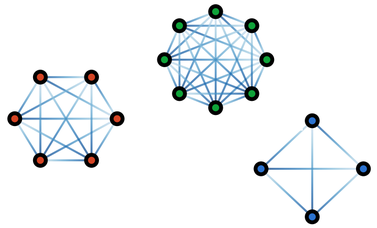
- Properties
  - Measurements distributed in space (and time) irregularly



  - Sparse (most detector channels are empty), but pockets of density

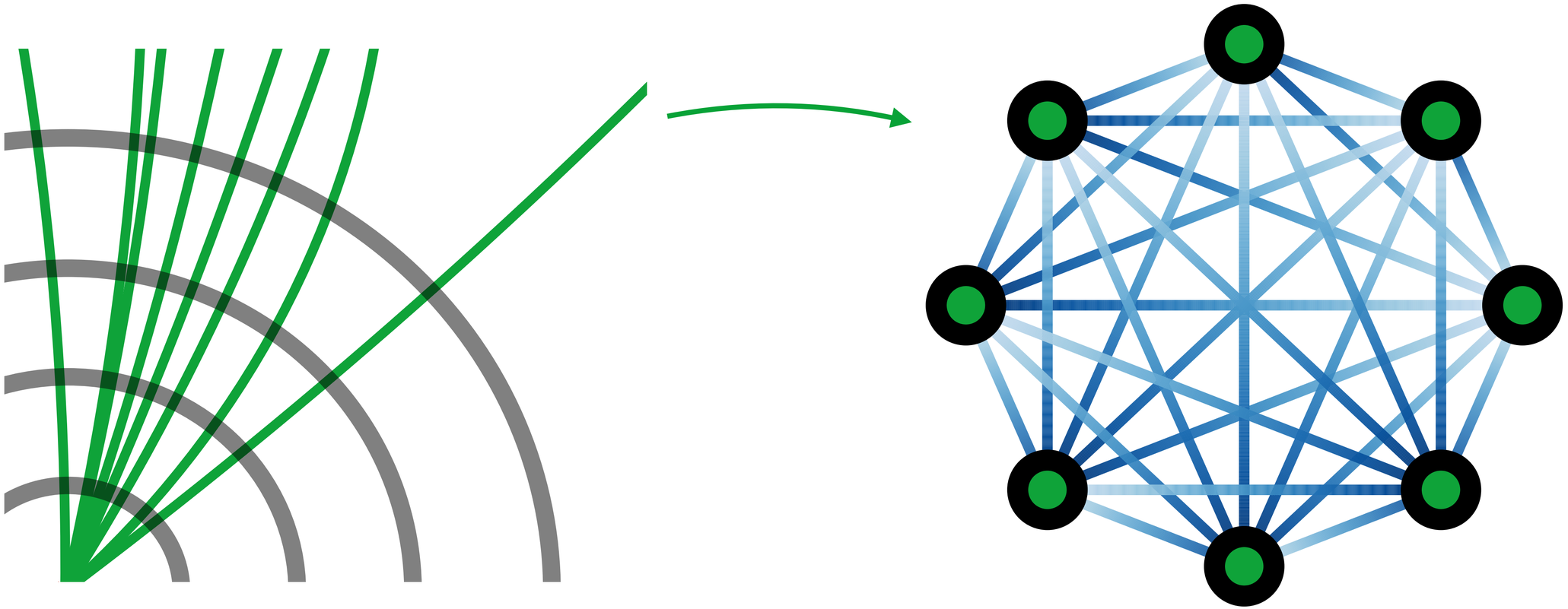


  - Complex interdependencies between measurements
  - Physics “objects” composed of multiple measurements



  - Inherent symmetries (Lorentz boosts, rotational)*
- Graph (or point cloud) embedding of the data can handle these properties

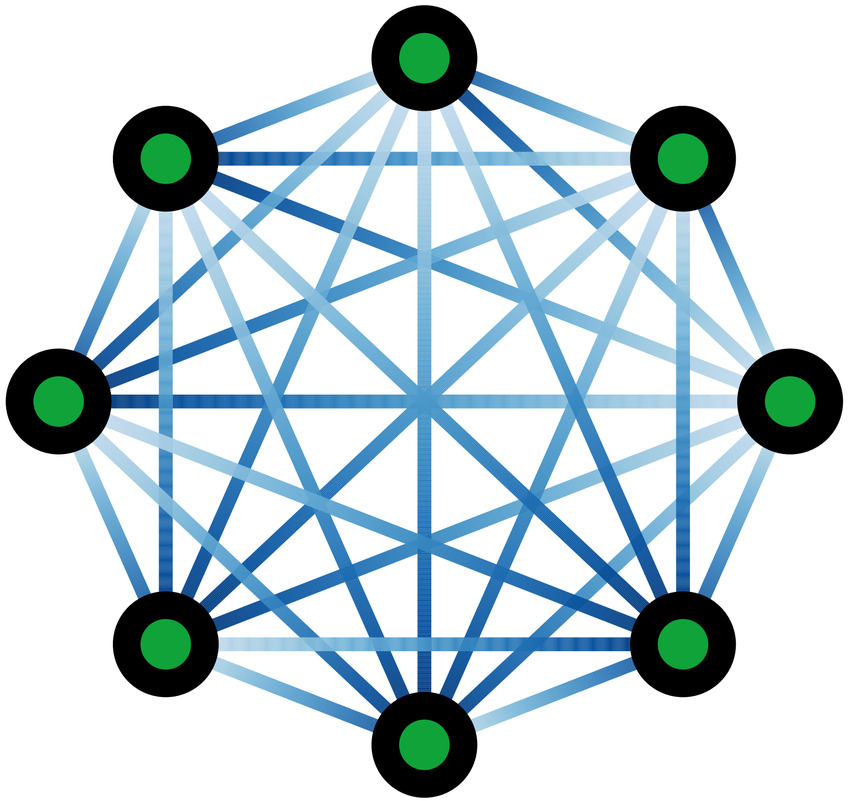
NODE, EDGE, GRAPH FEATURES IN HEP (E.G. JET)



NODE, EDGE, GRAPH FEATURES IN HEP (E.G. JET)

- Node features: particle 4-momentum

$$p = [E, p_x, p_y, p_z] \equiv [p_\mathrm{T}, \eta, \phi, m]$$



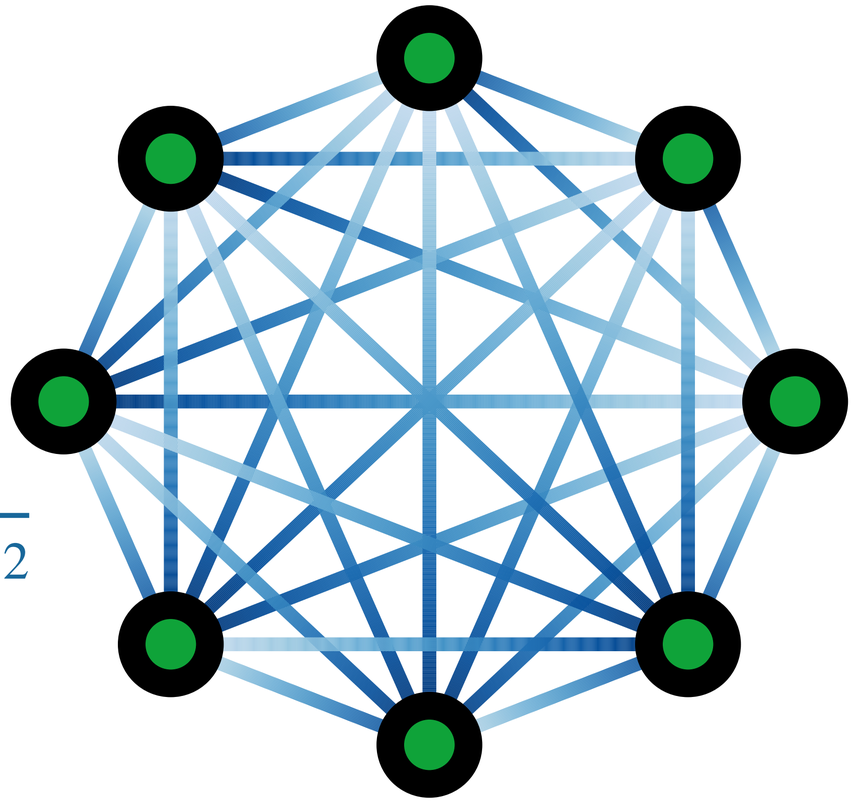
NODE, EDGE, GRAPH FEATURES IN HEP (E.G. JET)

- Node features: particle 4-momentum

$$p = [E, p_x, p_y, p_z] \equiv [p_\mathrm{T}, \eta, \phi, m]$$

- Edge features: pseudoangular distance between particles

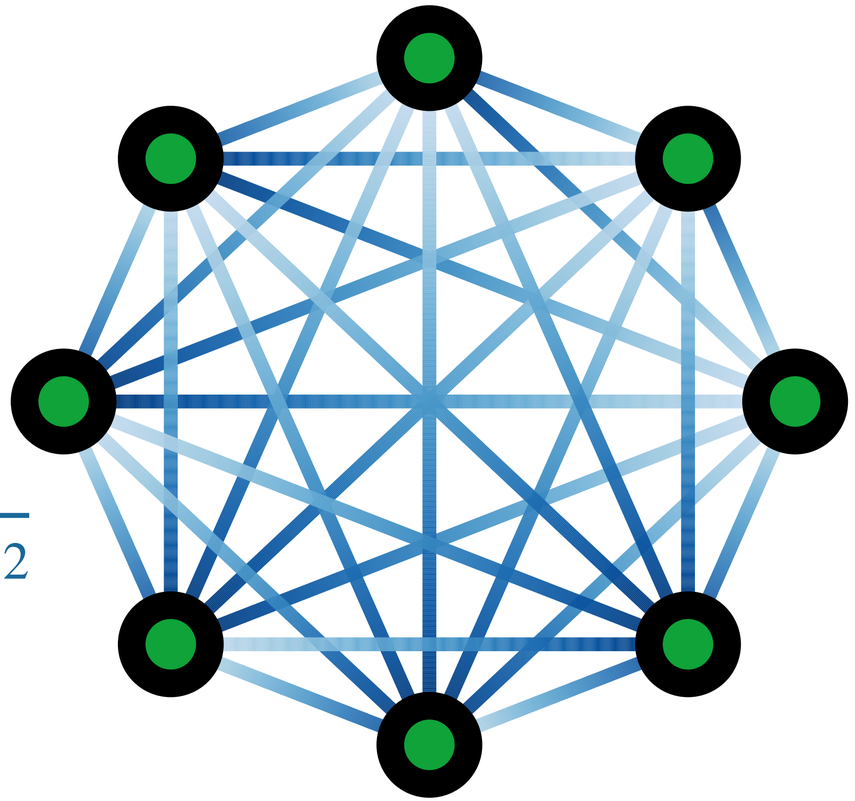
$$\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$$



NODE, EDGE, GRAPH FEATURES IN HEP (E.G. JET)

- Node features: particle 4-momentum

$$p = [E, p_x, p_y, p_z] \equiv [p_\mathrm{T}, \eta, \phi, m]$$



- Edge features: pseudoangular distance between particles

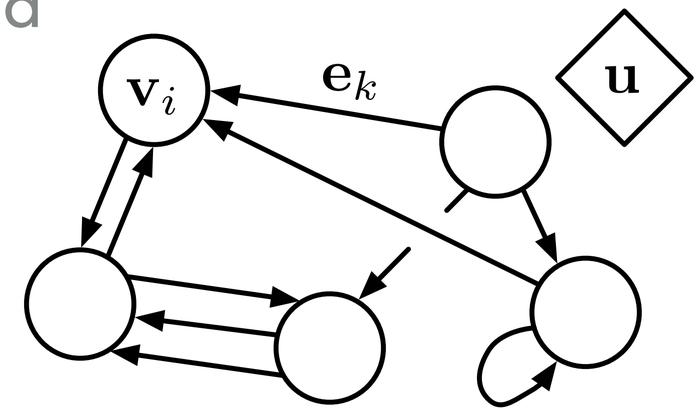
$$\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$$

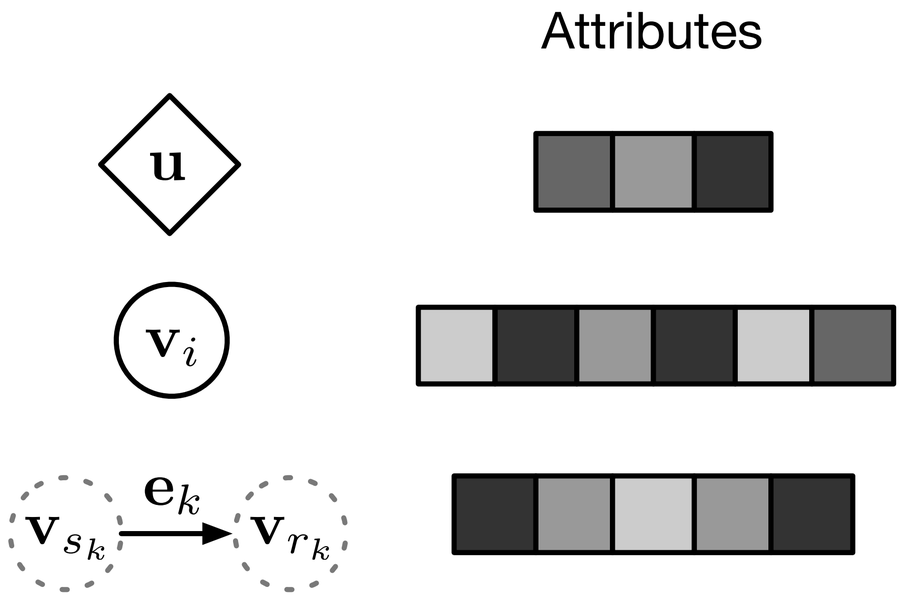
- Graph (global) features: jet mass

$$m = \sqrt{\sum_{i \in \mathrm{jet}} E_i^2 - p_{x,i}^2 - p_{y,i}^2 - p_{z,i}^2}$$

FORMALIZING A GRAPH

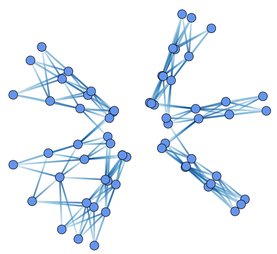
- Features: triplet of global features, node features, and edge features: $(\mathbf{u}, V, E)$
- Graph connectivity: adjacency matrix
  $A = \{a_{ij} = 1 \text{ if } i \text{ is connected to } j\}$
  Sparse representation:
  "receiver" indices $\boldsymbol{r}$ and "sender" indices $\boldsymbol{s}$
  e.g. k^{th} edge connects node s_k to node r_k
- Graph $G = (\mathbf{u}, V, E)$
  - Global attribute $\mathbf{u}$
  - Node attributes $V = \{\mathbf{v}_i\}$
  - Edge attributes, receivers, senders
    $E = \{\mathbf{e}_k, r_k, s_k\}$

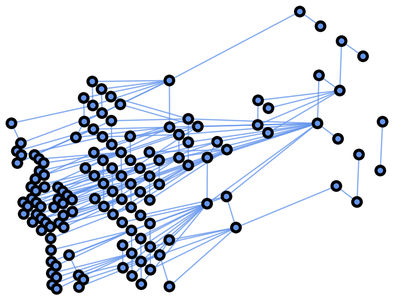




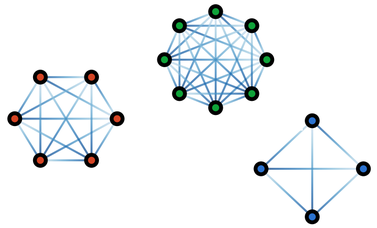
GRAPH TASKS

- Edge-level tasks
  - **Identify track segments**
  - Estimate track parameters
  - Secondary vertex reconstruction
- Node-level tasks
  - Correct cluster energies
  - Identify "pileup" particles
  - **Particle-flow reconstruction**



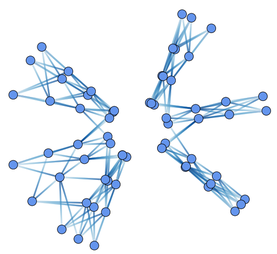


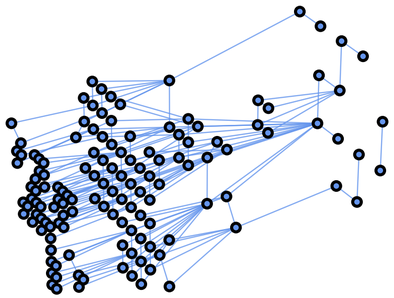
- Graph-level tasks
  - **Jet tagging**
  - Estimate shower energy
  - Signal-to-background event discrimination



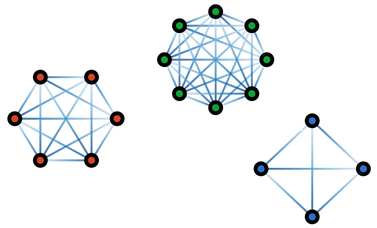
GRAPH TASKS

- Edge-level tasks
  - **Identify track segments**
  - Estimate track parameters
  - Secondary vertex reconstruction
- Node-level tasks
  - Correct cluster energies
  - Identify "pileup" particles
  - **Particle-flow reconstruction**





- Graph-level tasks
  - **Jet tagging**
  - Estimate shower energy
  - Signal-to-background event discrimination



⇒ GRAPH-TO-GRAPH MAPPINGS

DESIRED PROPERTIES OF GRAPH-BASED NEURAL NETWORKS

$$(\mathbf{u}, V, E) \rightarrow (\mathbf{u}', V', E')$$

DESIRED PROPERTIES OF GRAPH-BASED NEURAL NETWORKS

- Must have:
  - General graph-to-graph mappings valid for graphs of varying sizes

$$(\mathbf{u}, V, E) \rightarrow (\mathbf{u}', V', E')$$

DESIRED PROPERTIES OF GRAPH-BASED NEURAL NETWORKS

- Must have:
  - General graph-to-graph mappings valid for graphs of varying sizes
  - Graph-level outputs should be *invariant* under permutations of the nodes (node ordering should not matter)

$$(\mathbf{u}, V, E) \rightarrow (\mathbf{u}', V', E')$$

$$f(V) = f(PV)$$

DESIRED PROPERTIES OF GRAPH-BASED NEURAL NETWORKS

- Must have:
  - General graph-to-graph mappings valid for graphs of varying sizes

$$(\mathbf{u}, V, E) \rightarrow (\mathbf{u}', V', E')$$

  - Graph-level outputs should be *invariant* under permutations of the nodes (node ordering should not matter)

$$f(V) = f(PV)$$

  - Node- or edge-level outputs should be *equivariant* under permutations of the nodes (outputs should be permuted if inputs are permuted)

$$f(PV, PAP^T) = Pf(V, A)$$

DESIRED PROPERTIES OF GRAPH-BASED NEURAL NETWORKS

- Must have:
  - General graph-to-graph mappings valid for graphs of varying sizes

$$(\mathbf{u}, V, E) \rightarrow (\mathbf{u}', V', E')$$

  - Graph-level outputs should be *invariant* under permutations of the nodes (node ordering should not matter)

$$f(V) = f(PV)$$

  - Node- or edge-level outputs should be *equivariant* under permutations of the nodes (outputs should be permuted if inputs are permuted)

$$f(PV, PAP^T) = Pf(V, A)$$

  - Learning uses "locality" (outputs of nodes in the same neighborhood should be more similar)

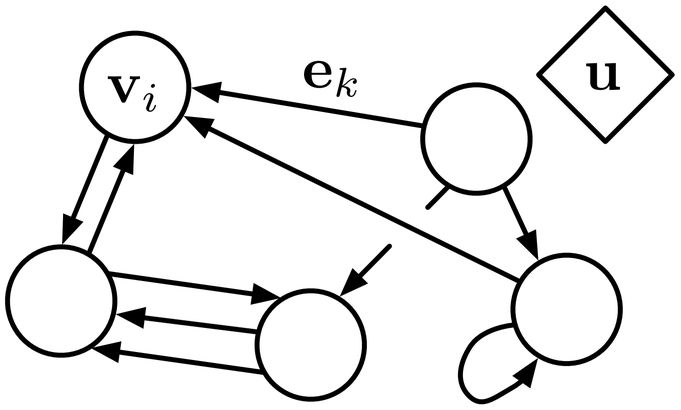
$$g(\mathbf{v}_i) \sim g(\mathbf{v}_j) \text{ if } \mathcal{N}_i \sim \mathcal{N}_j$$

DESIRED PROPERTIES OF GRAPH-BASED NEURAL NETWORKS

- Must have:
  - General graph-to-graph mappings valid for graphs of varying sizes $$(\mathbf{u}, V, E) \rightarrow (\mathbf{u}', V', E')$$
  - Graph-level outputs should be *invariant* under permutations of the nodes (node ordering should not matter) $$f(V) = f(PV)$$
  - Node- or edge-level outputs should be *equivariant* under permutations of the nodes (outputs should be permuted if inputs are permuted) $$f(PV, PAP^T) = Pf(V, A)$$
  - Learning uses "locality" (outputs of nodes in the same neighborhood should be more similar) $$g(\mathbf{v}_i) \sim g(\mathbf{v}_j) \text{ if } \mathcal{N}_i \sim \mathcal{N}_j$$
- Nice to have:
  - Account for nodes of different types ("heterogeneous graphs")
  - Generalizes to smaller/larger graphs than those seen in training
  - Computationally efficient

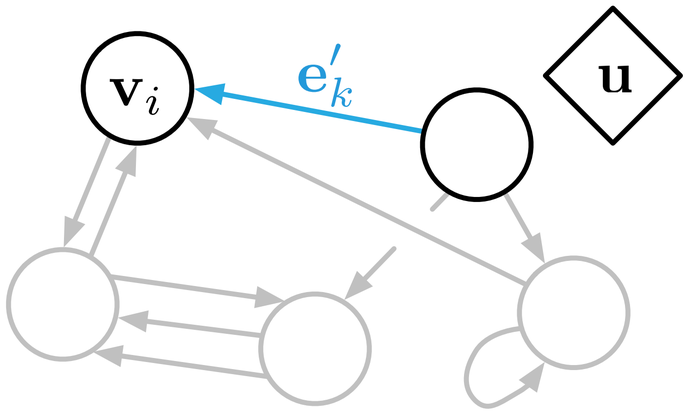
GRAPH NEURAL NETWORKS*

- One framework for describing GNNs is "Graph Networks" [arXiv:1806.01261]
- GNN is a graph-to-graph function approximator
- Inference divided into three parts: edge block, node block, global block



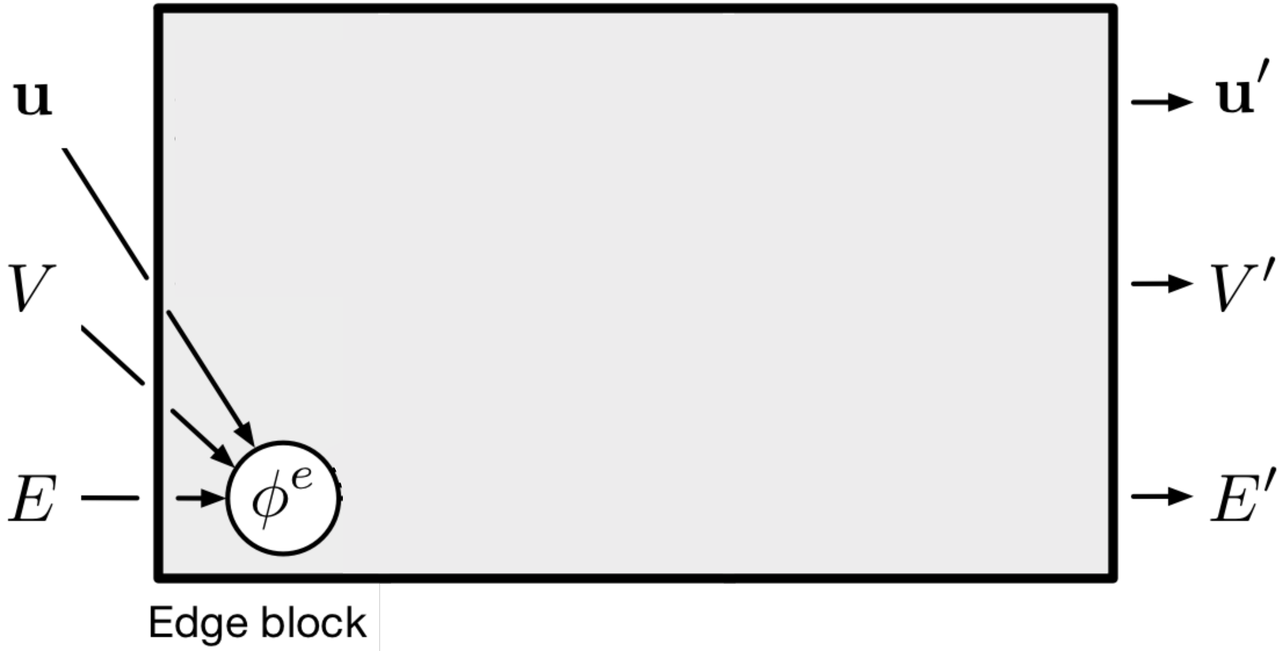
GRAPH NEURAL NETWORKS*

- One framework for describing GNNs is "Graph Networks" [arXiv:1806.01261]
- GNN is a graph-to-graph function approximator
- Inference divided into three parts: edge block, node block, global block



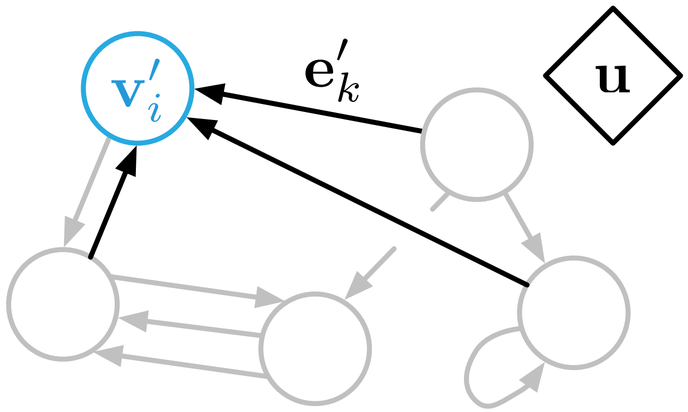
$\mathbf{e}'_k$: message computed for edge k connecting nodes r_k, s_k

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u})$$



GRAPH NEURAL NETWORKS*

- One framework for describing GNNs is "Graph Networks" [arXiv:1806.01261]
- GNN is a graph-to-graph function approximator
- Inference divided into three parts: edge block, node block, global block

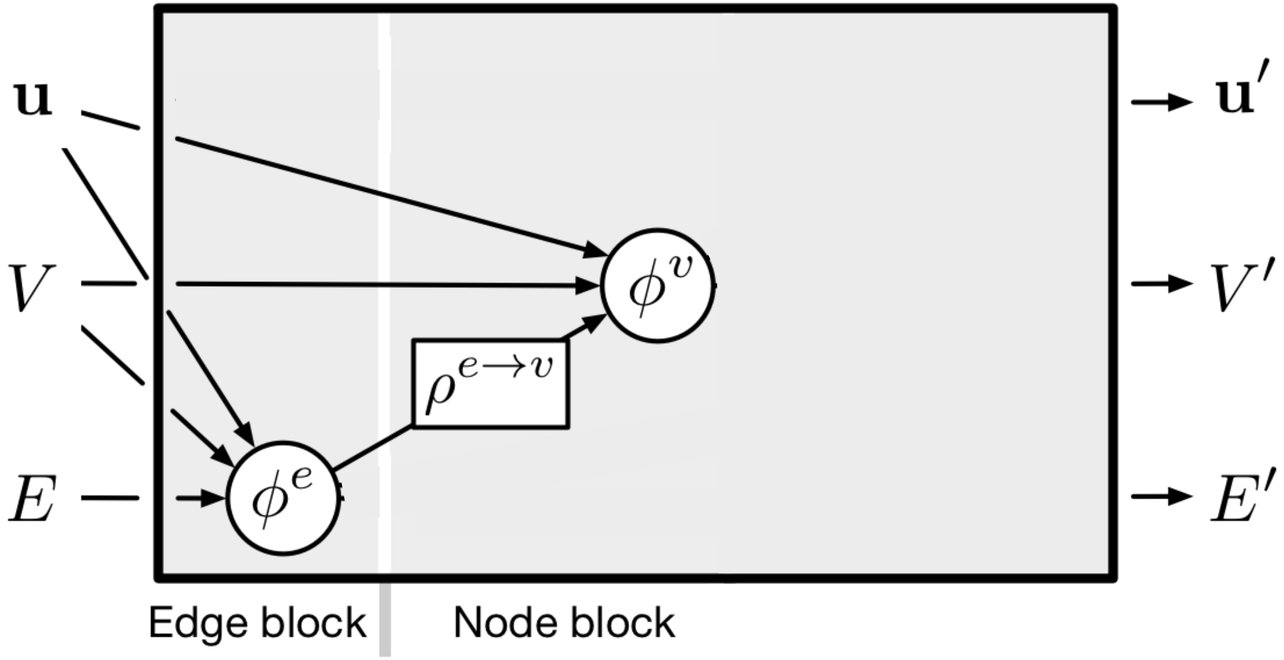


$\mathbf{e}'_k$: message computed for edge k connecting nodes r_k, s_k

$\mathbf{v}'_i$: node feature update based on aggregated messages and previous features

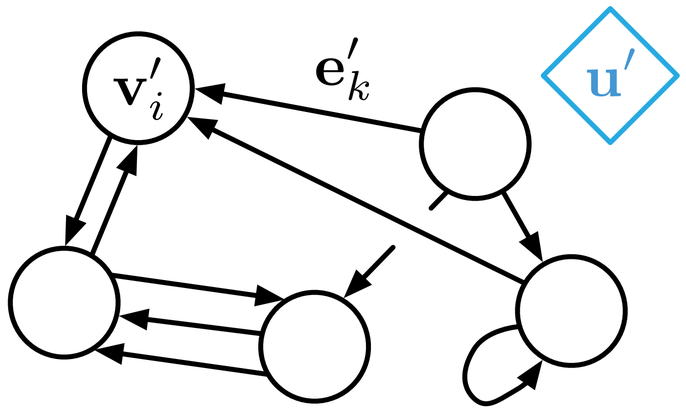
$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) \qquad \bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i)$$

$$\mathbf{v}'_i = \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right)$$



GRAPH NEURAL NETWORKS*

- One framework for describing GNNs is "Graph Networks" [arXiv:1806.01261]
- GNN is a graph-to-graph function approximator
- Inference divided into three parts: edge block, node block, global block



$\mathbf{e}'_k$: message computed for edge k connecting nodes r_k, s_k

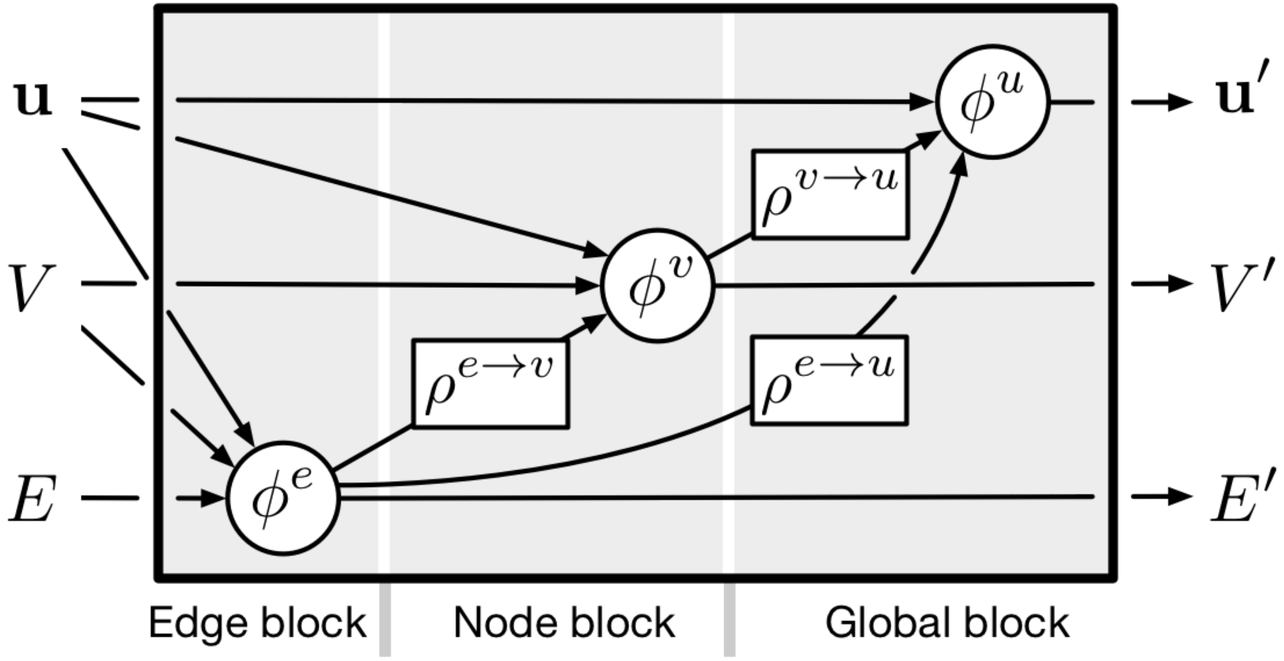
$\mathbf{v}'_i$: node feature update based on aggregated messages and previous features

$\mathbf{u}'$: global feature update based on aggregated, updated node and edge features

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) \qquad \bar{\mathbf{e}}'_i = \rho^{e\to v}(E'_i)$$

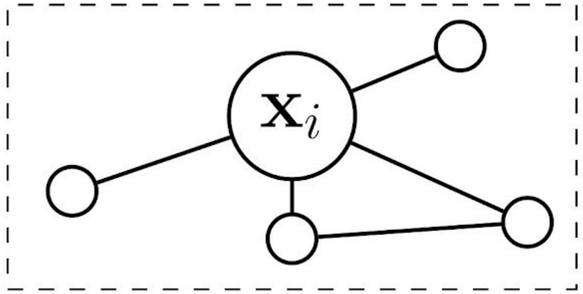
$$\mathbf{v}'_i = \phi^v\left(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}\right) \qquad \bar{\mathbf{e}}' = \rho^{e\to u}(E')$$

$$\mathbf{u}' = \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \boldsymbol{u}) \qquad \bar{\mathbf{v}}' = \rho^{v\to u}(V')$$



HOW TO USE GNNS

Source: https://youtu.be/uF53xsT7mjc

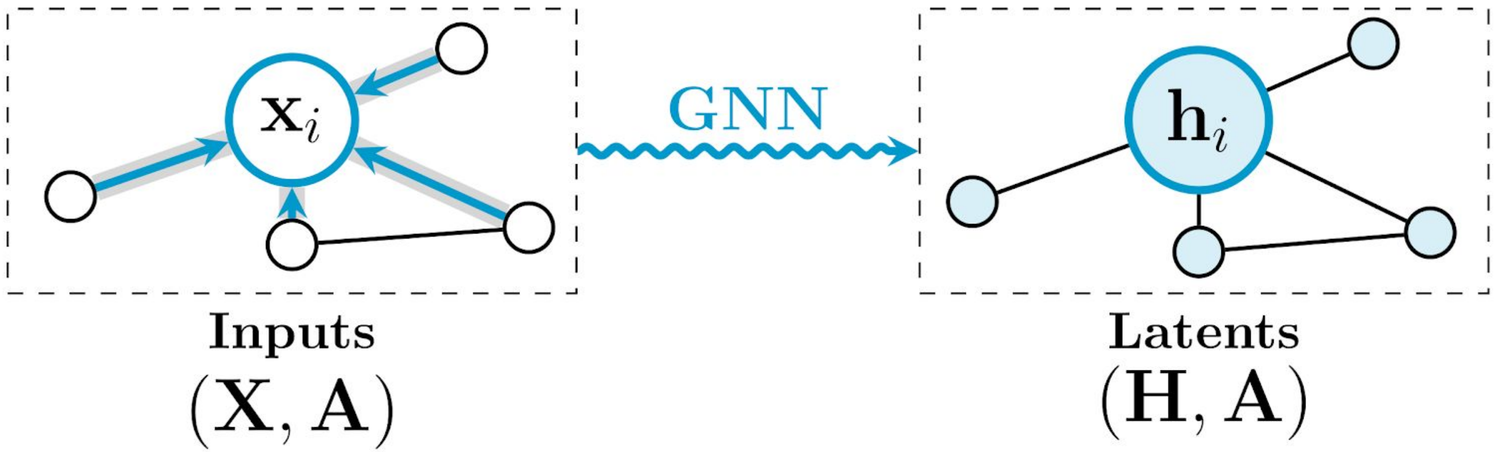


Inputs

$(\mathbf{X}, \mathbf{A})$

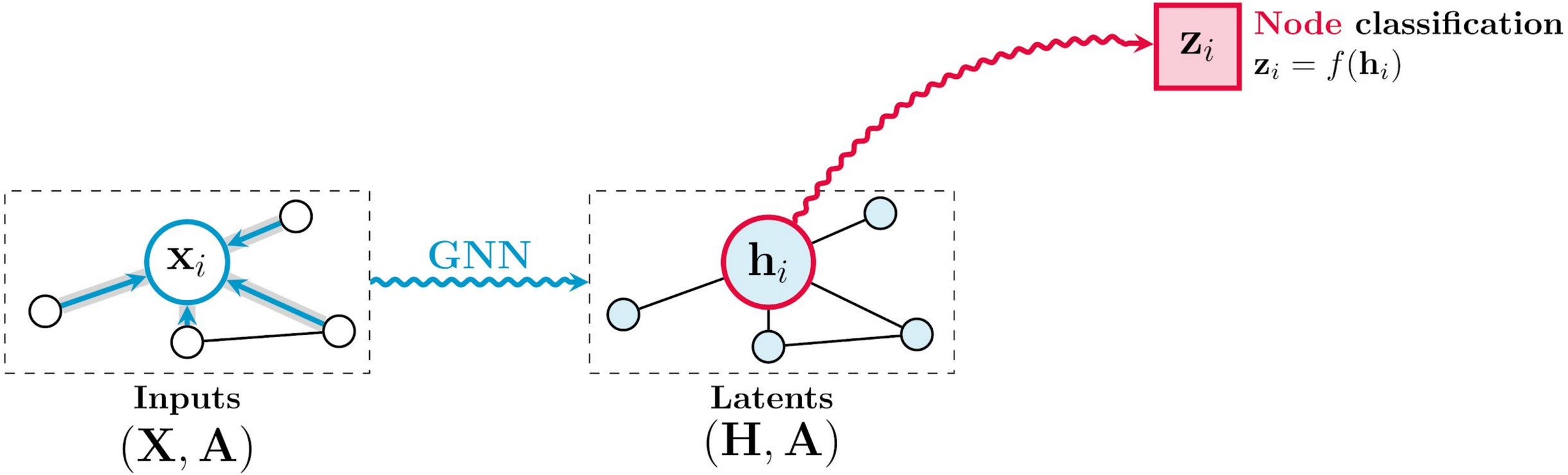
HOW TO USE GNNS

Source: https://youtu.be/uF53xsT7mjc



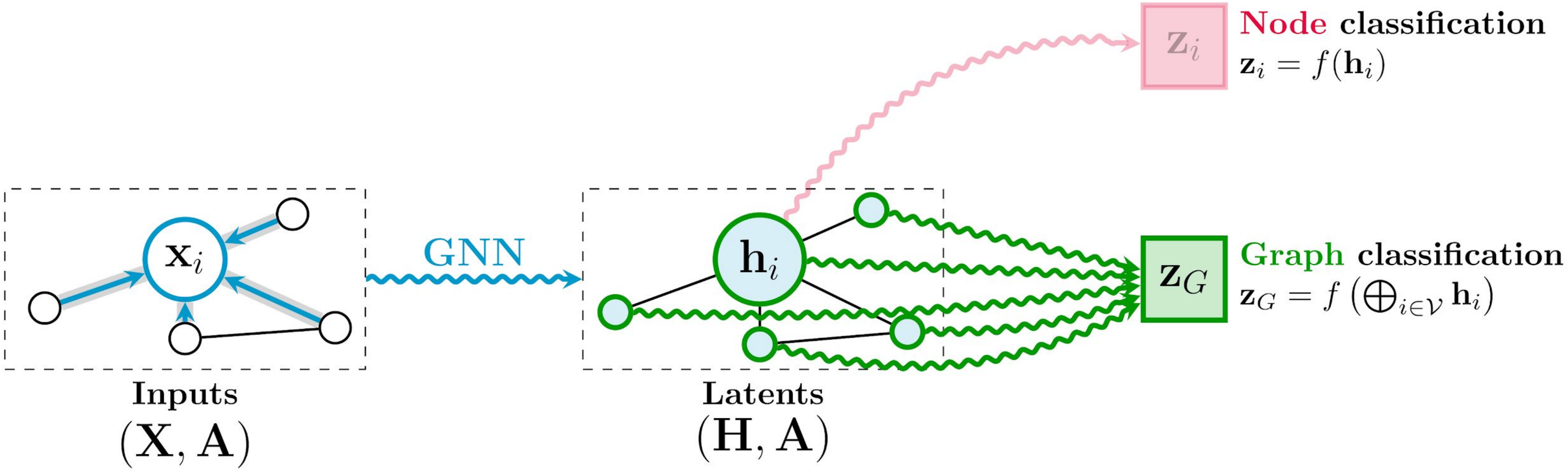
HOW TO USE GNNS

Source: https://youtu.be/uF53xsT7mjc



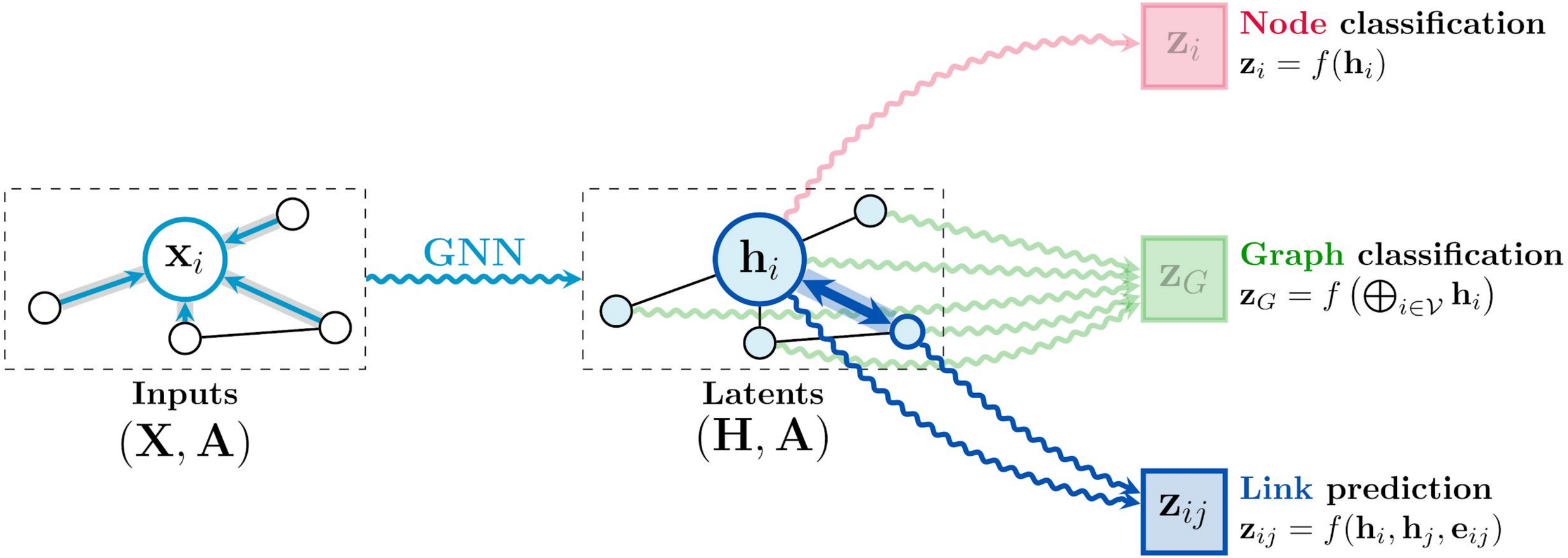
HOW TO USE GNNS

Source: https://youtu.be/uF53xsT7mjc



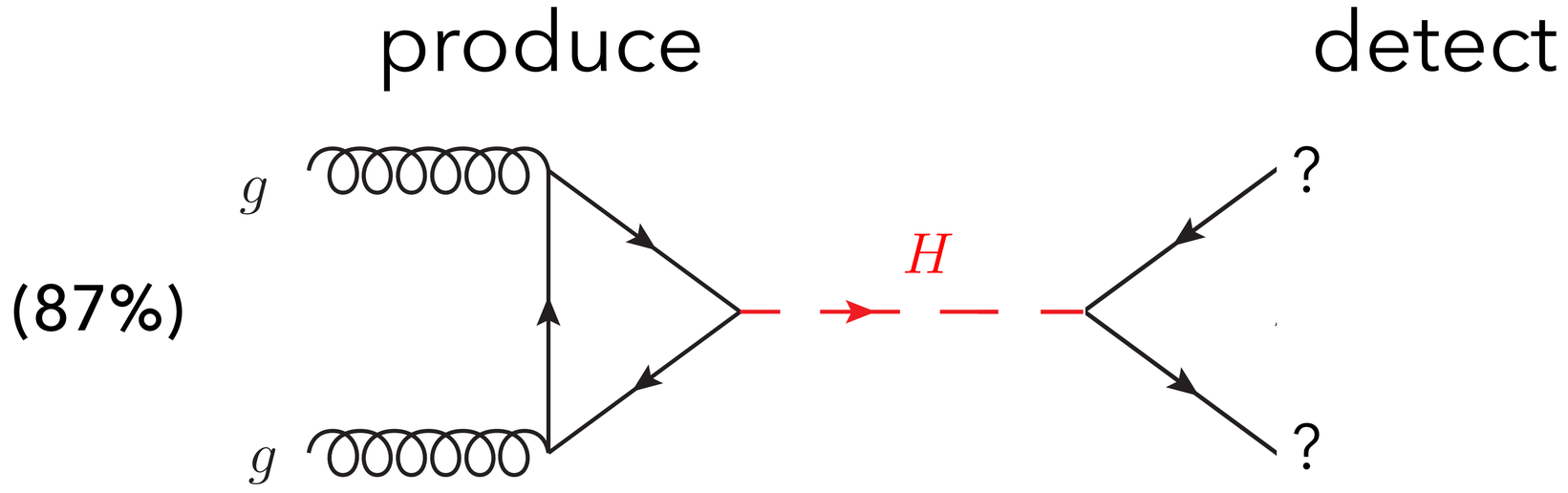
HOW TO USE GNNS

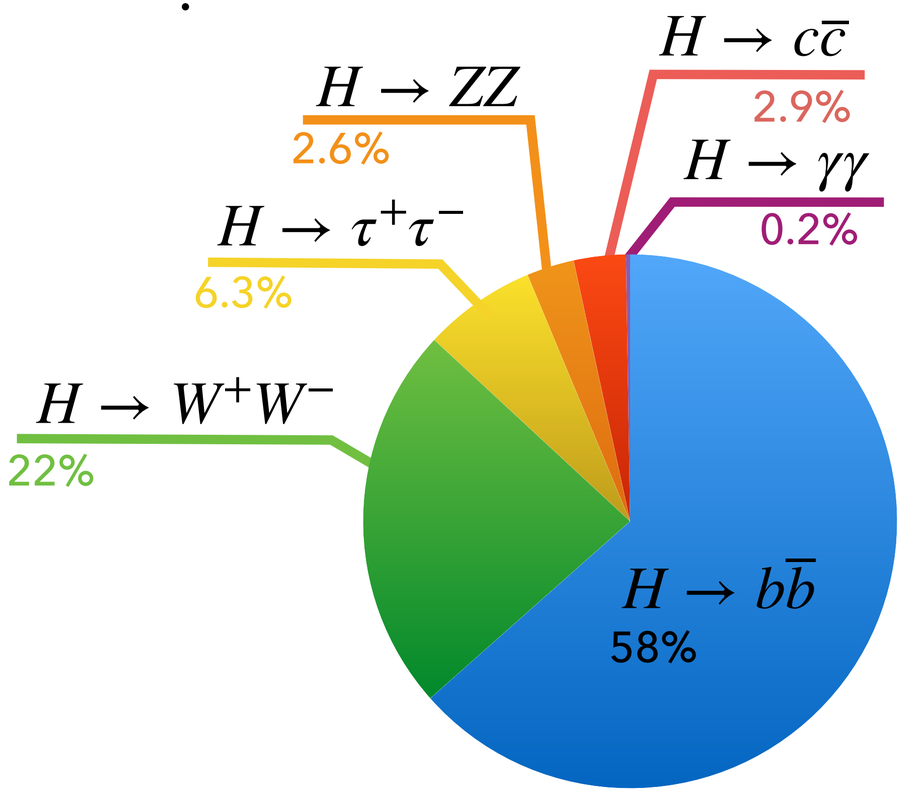
Source: https://youtu.be/uF53xsT7mjc



HIGGS JET TAGGING

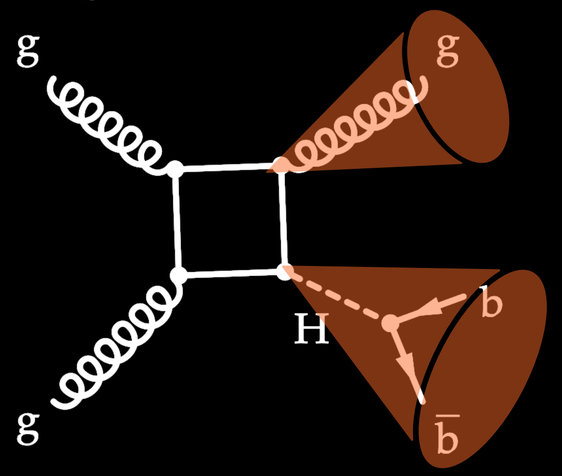
HOW DO WE DETECT HIGGS BOSONS?



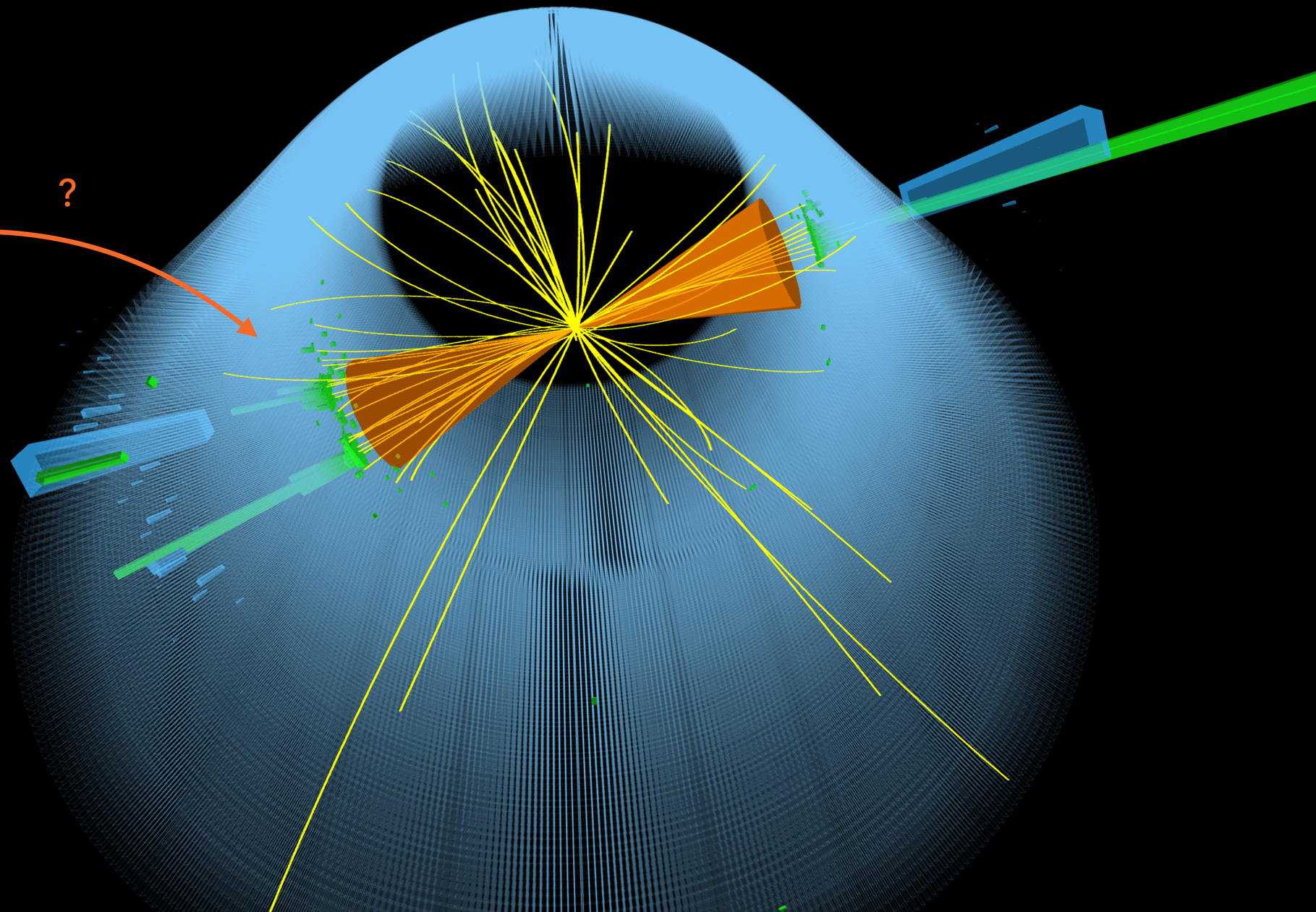


- H→γγ and H→ZZ have small rate but are very "clean"
- H→bb is large, but more difficult due to large backgrounds

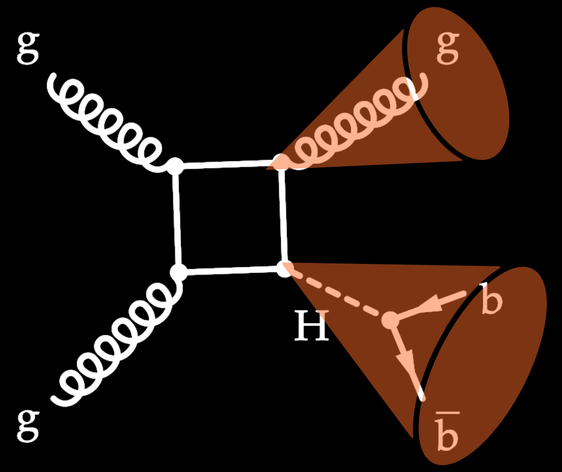
Signal:



?

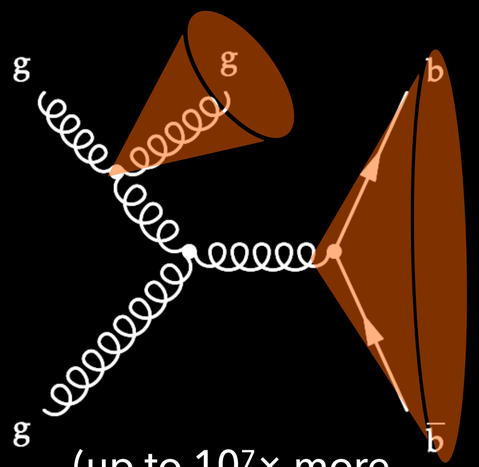


Signal:



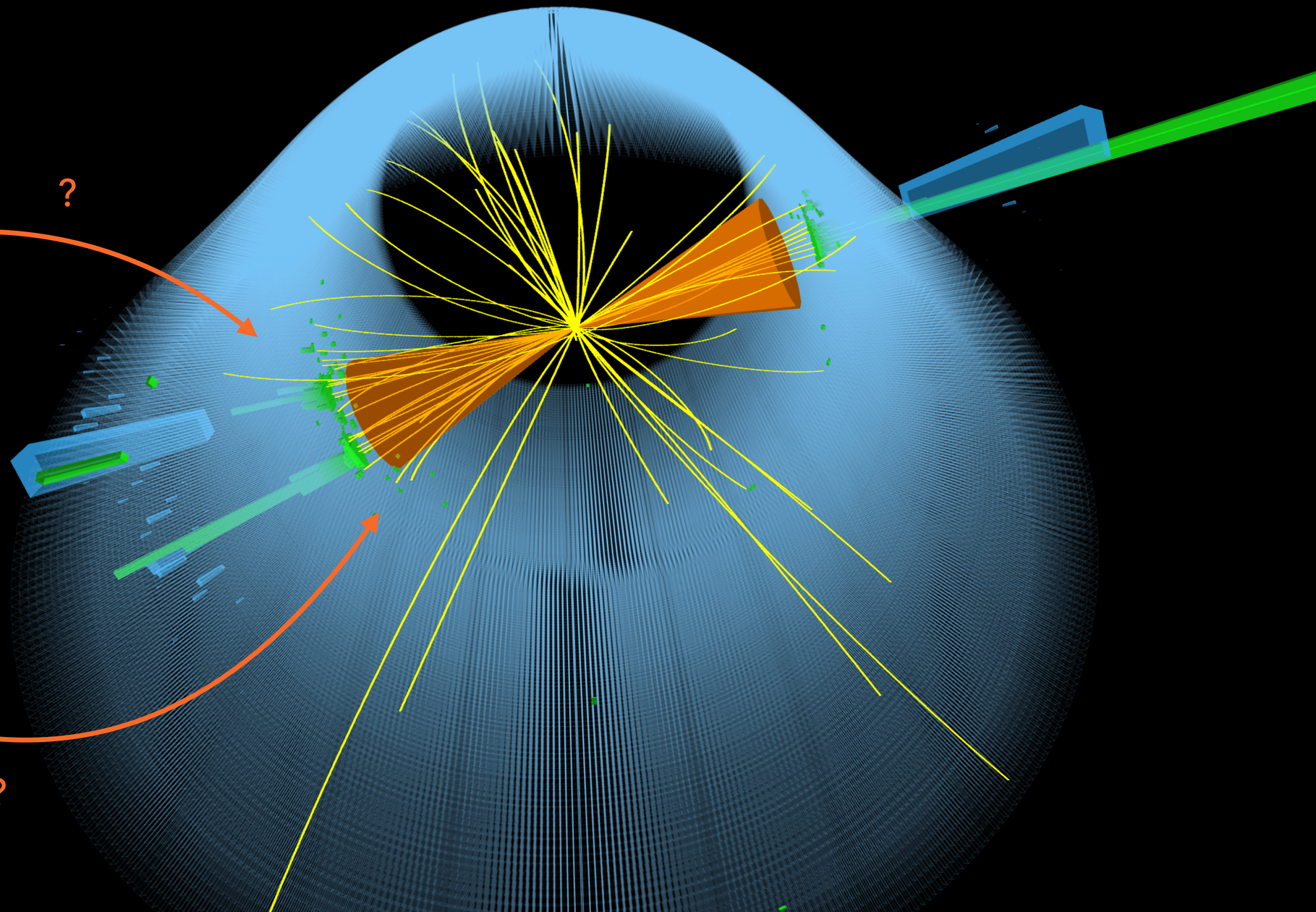
?

Background:



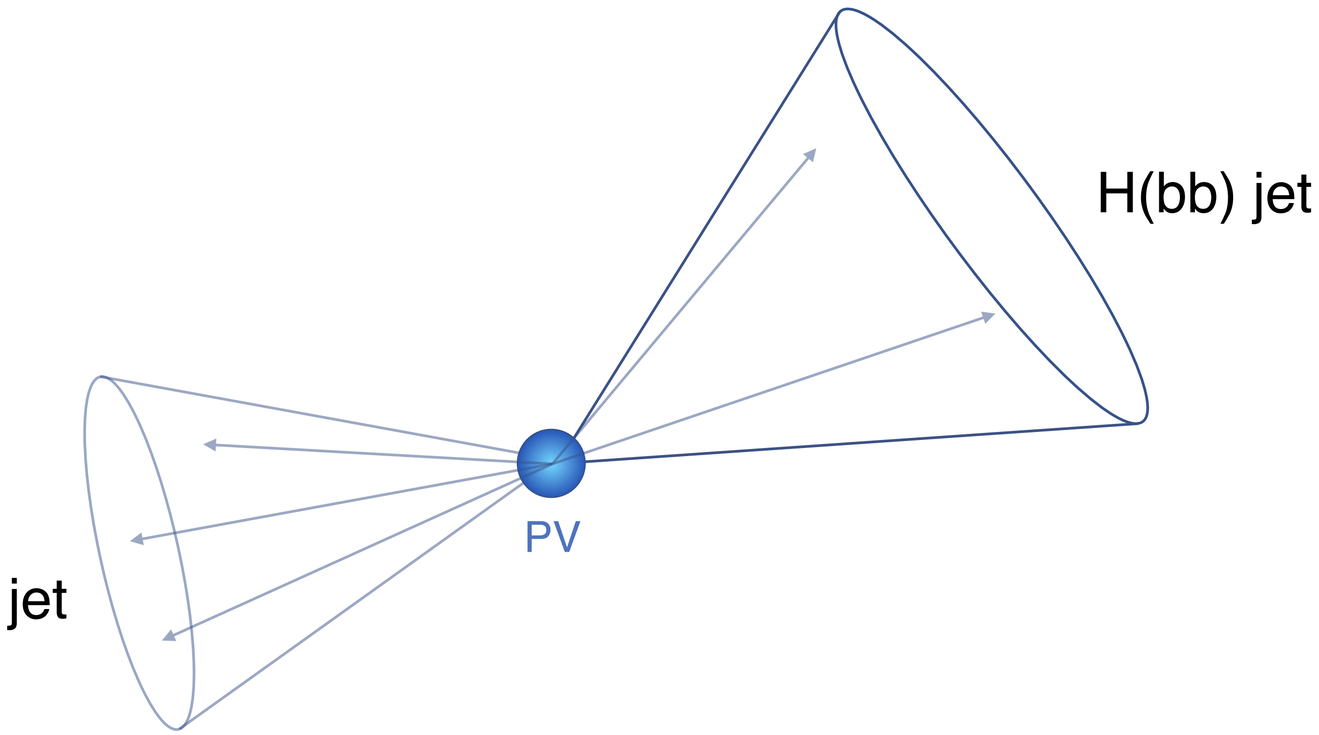
(up to 10^7× more frequent)

?



BASICS OF HIGGS (DOUBLE-B) TAGGING

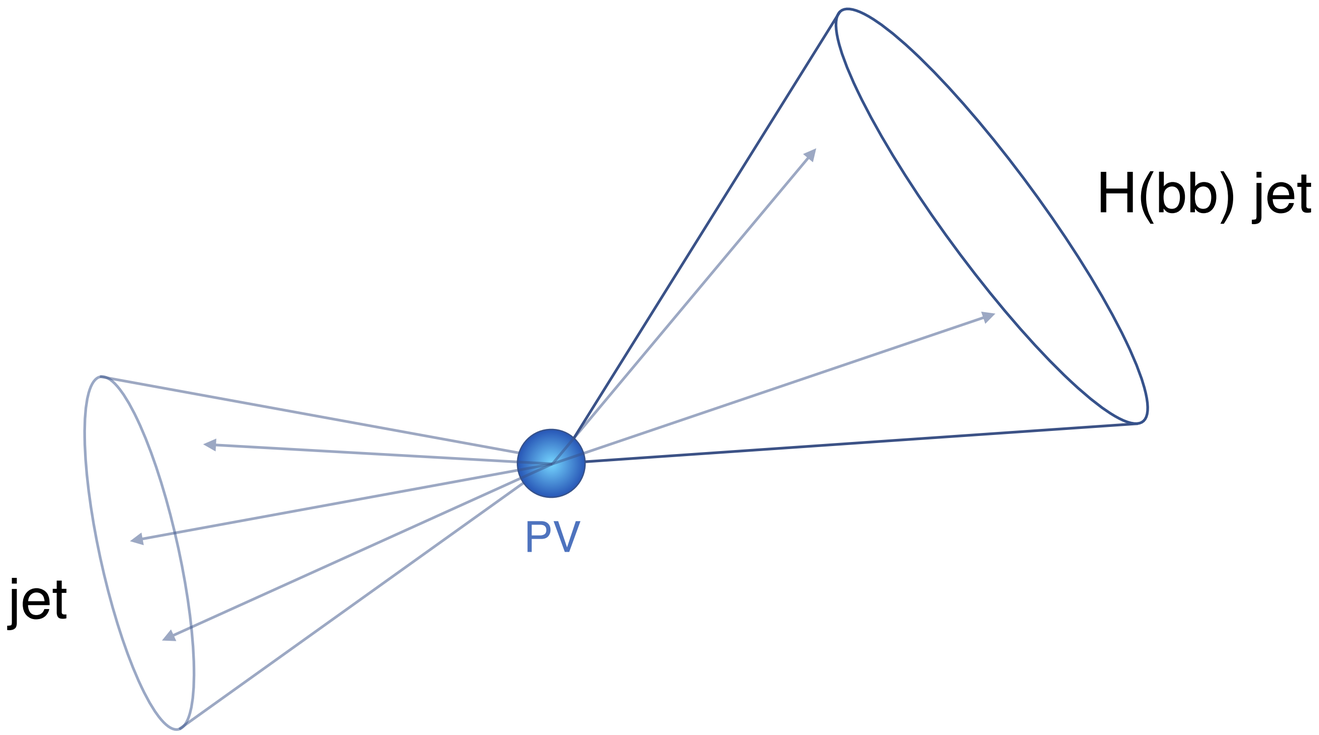
b hadrons have long lifetimes:
travel O(mm) before decay!



BASICS OF HIGGS (DOUBLE-B) TAGGING

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

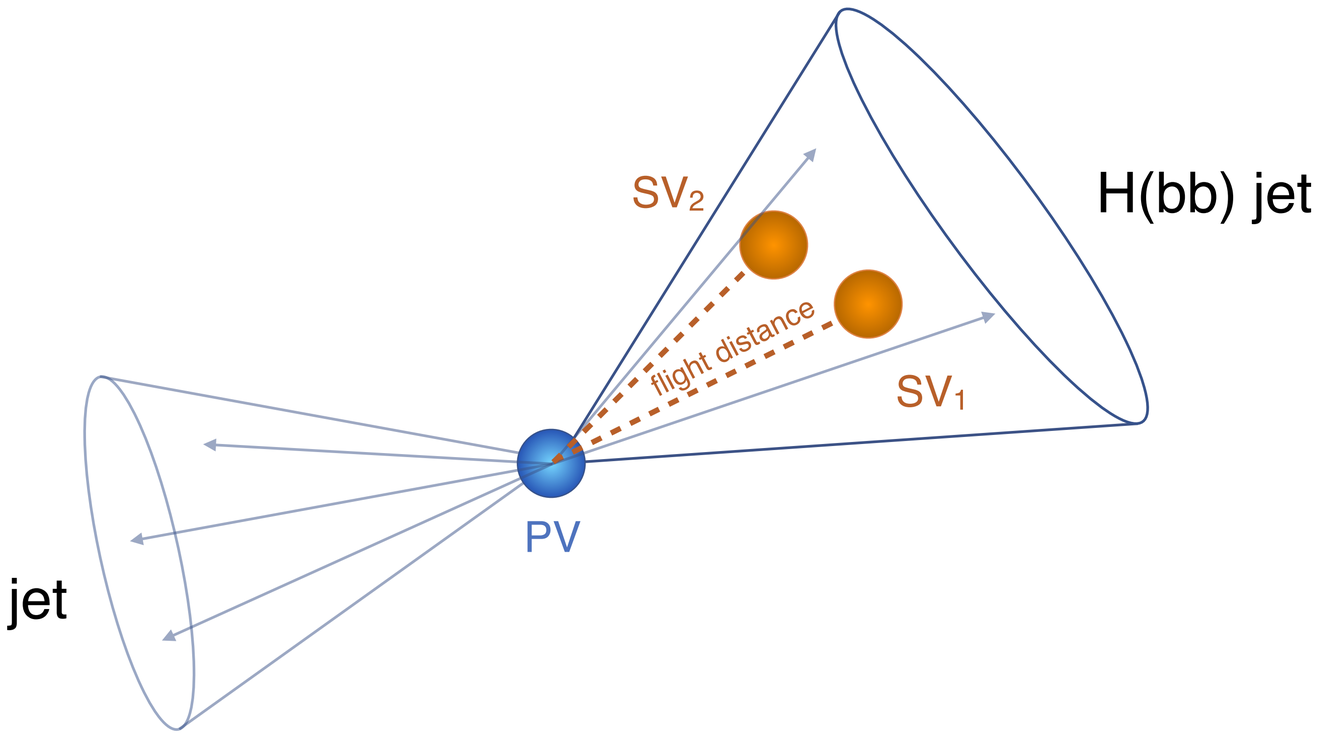
- Handles:



BASICS OF HIGGS (DOUBLE-B) TAGGING

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

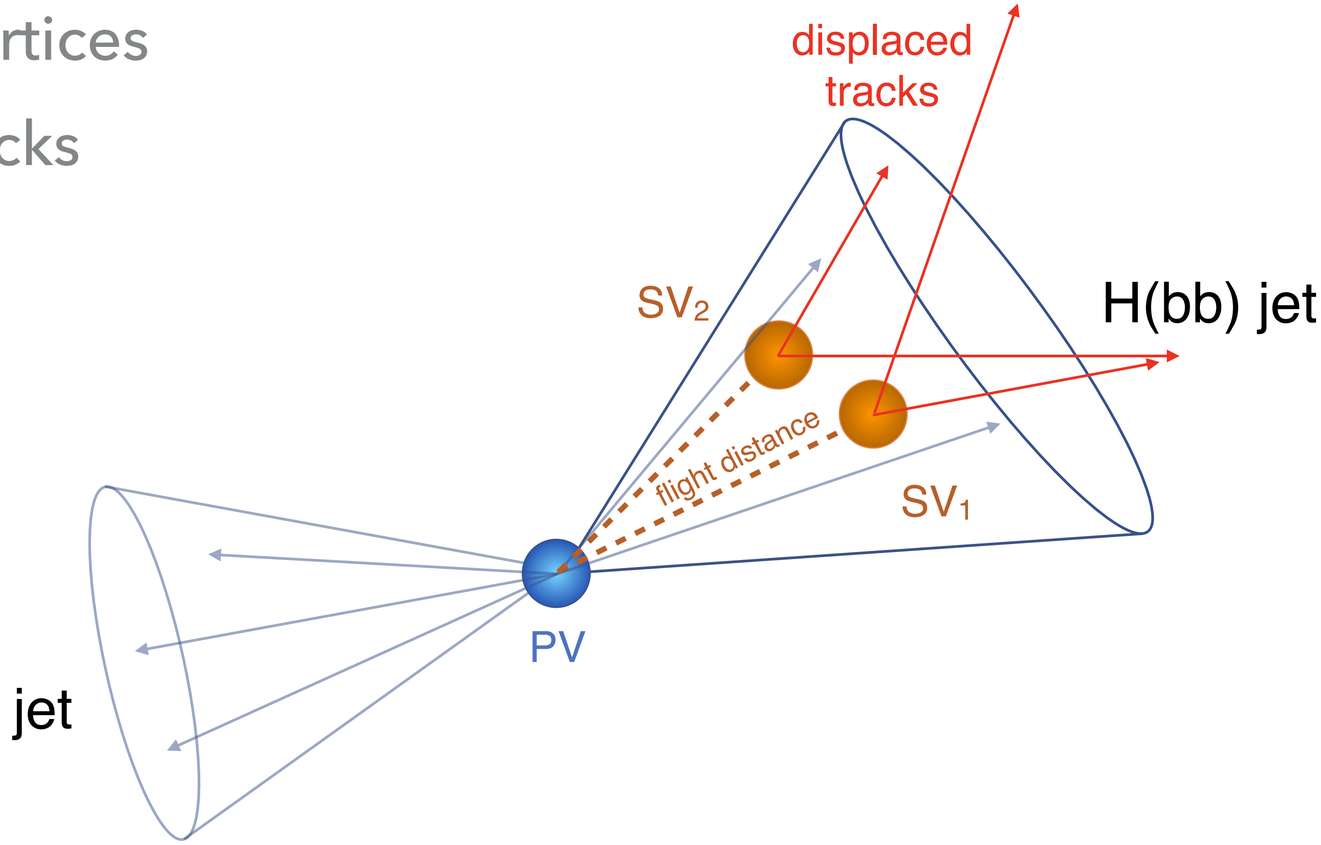
- Handles:
  - secondary vertices



BASICS OF HIGGS (DOUBLE-B) TAGGING

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

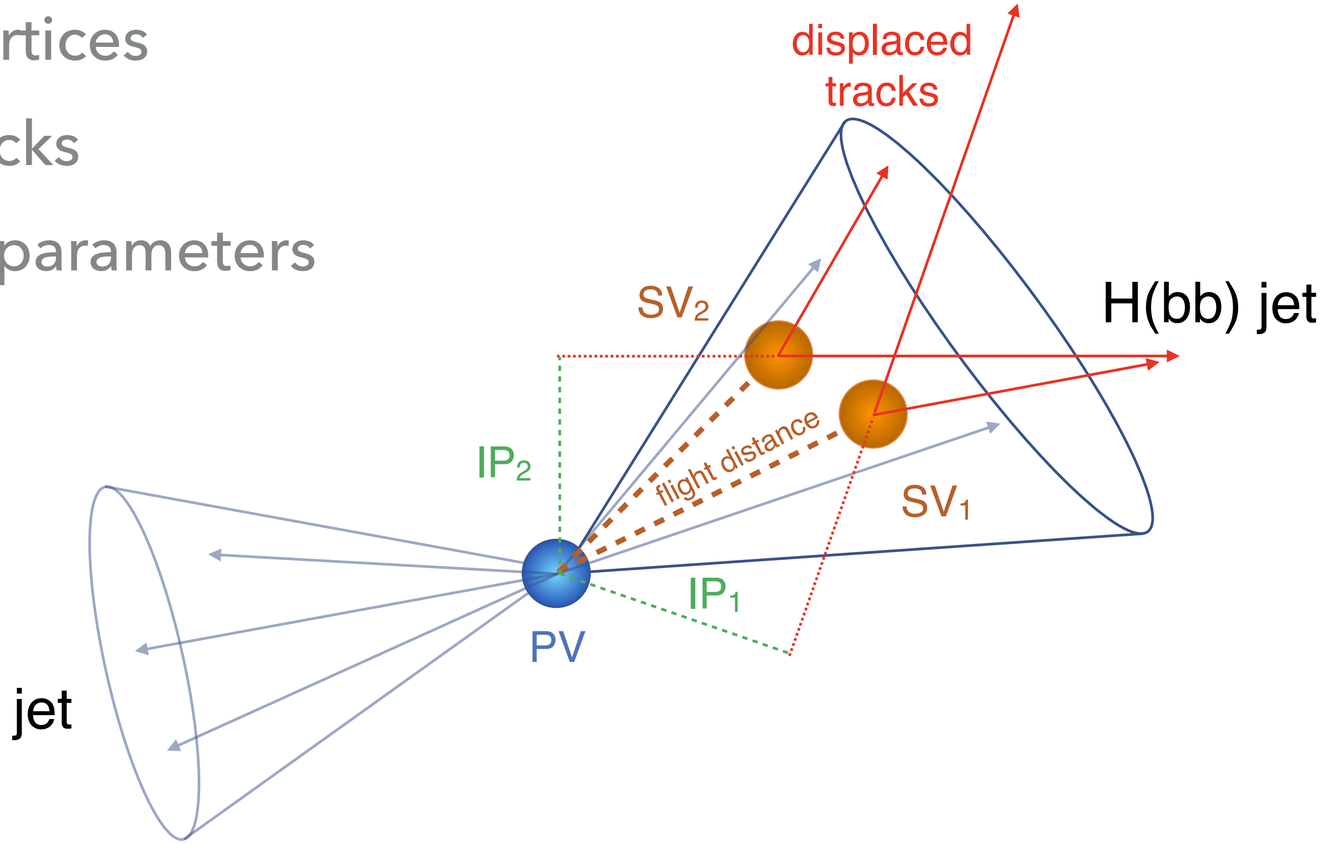
- Handles:
  - secondary vertices
  - displaced tracks



BASICS OF HIGGS (DOUBLE-B) TAGGING

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

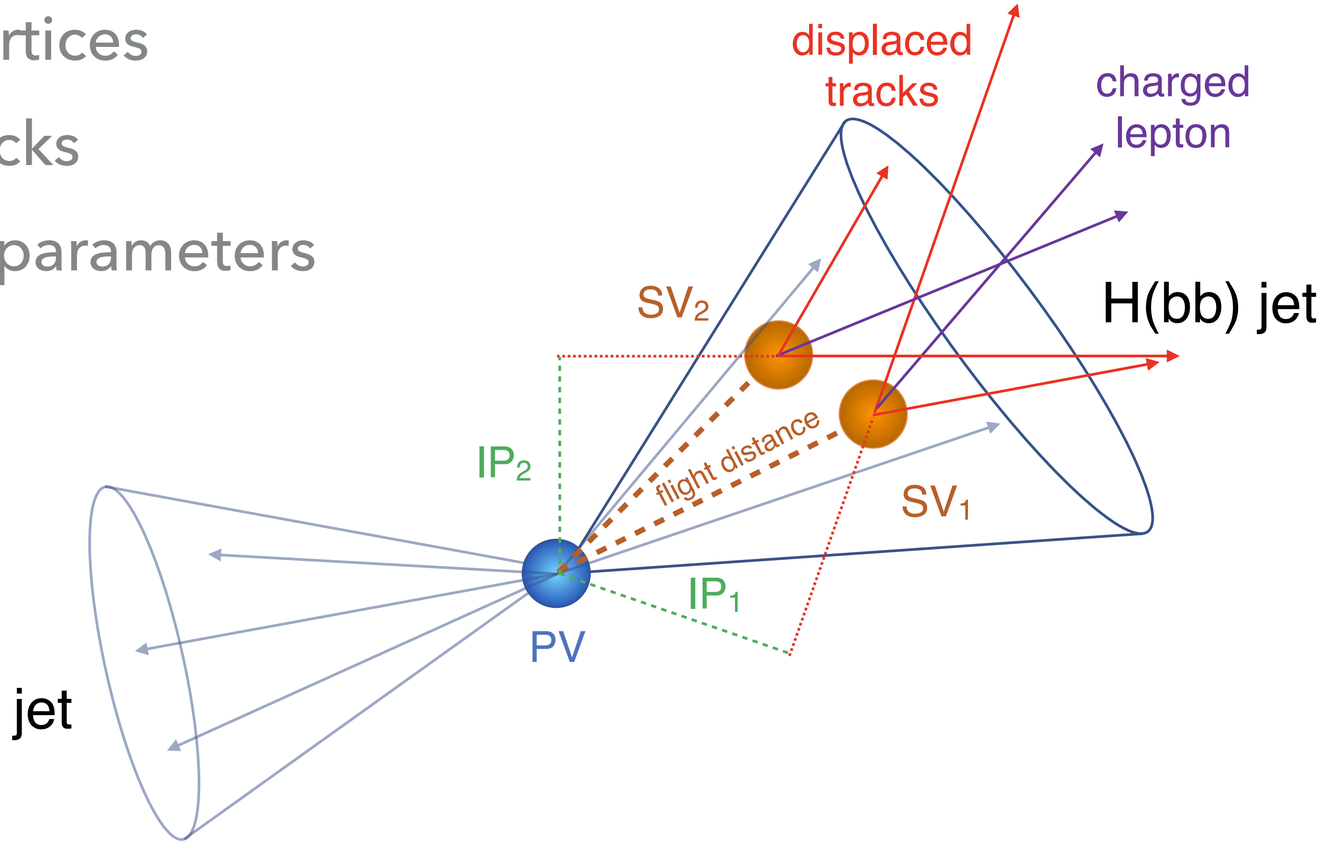
- Handles:
  - secondary vertices
  - displaced tracks
  - large impact parameters



BASICS OF HIGGS (DOUBLE-B) TAGGING

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

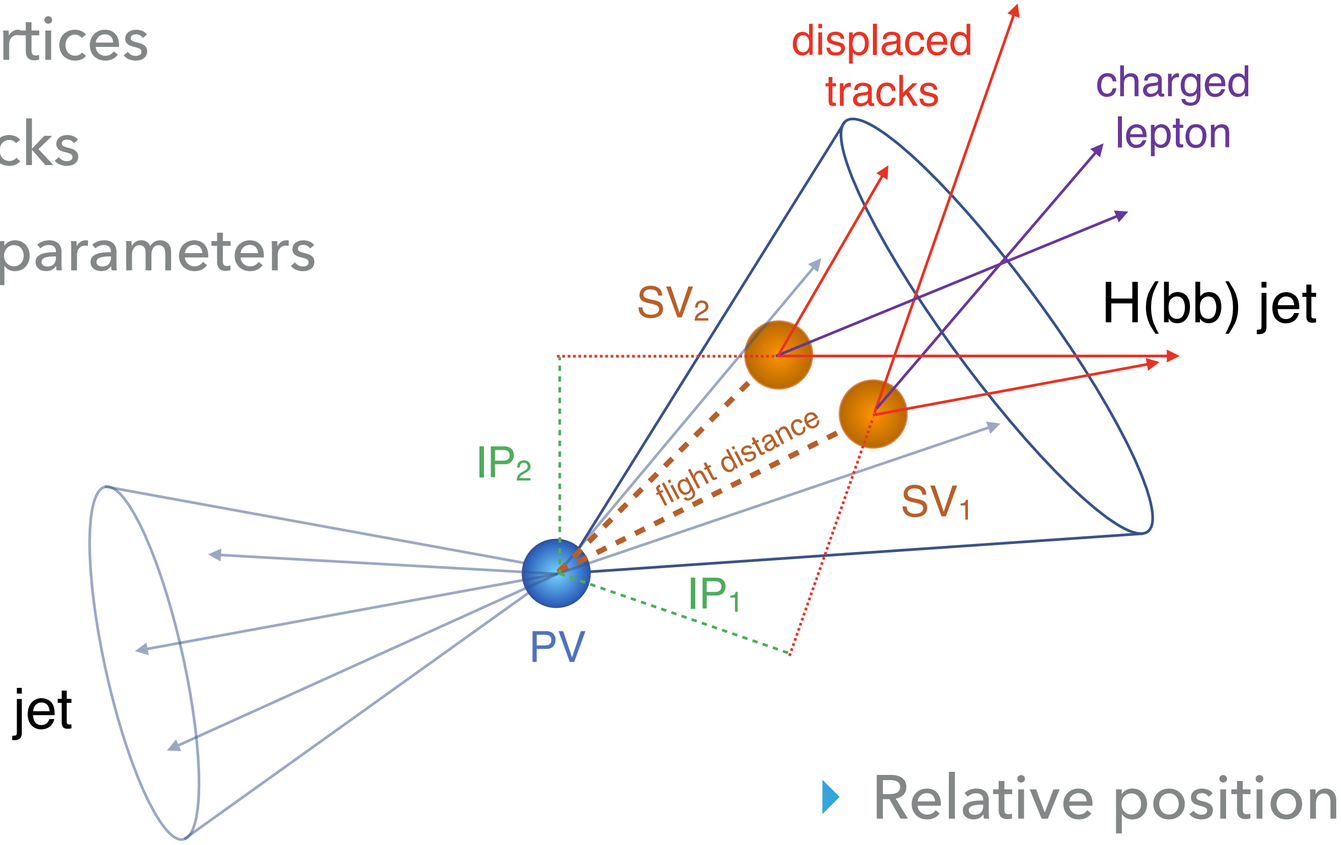
- Handles:
  - secondary vertices
  - displaced tracks
  - large impact parameters
  - soft leptons



BASICS OF HIGGS (DOUBLE-B) TAGGING

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

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

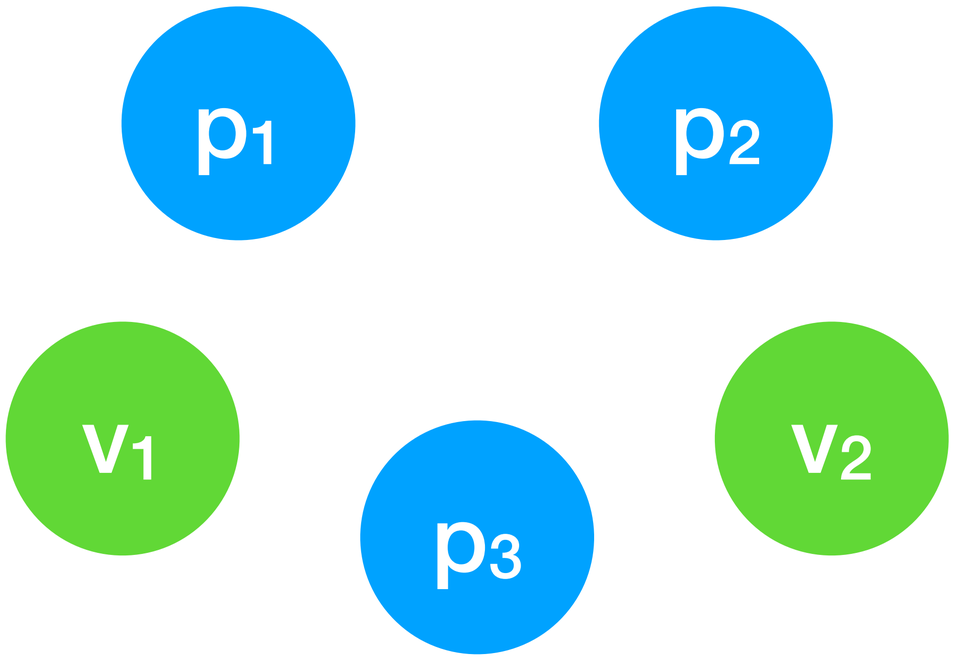


- Relative positions of SVs

PARTICLES AND SECONDARY VERTICES: TWO INPUT GRAPHS

arXiv:1909.12285

$$p_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, d_{\mathrm{3D}}, \mathrm{cov}(p_{\mathrm{T}}, p_{\mathrm{T}}), \ldots]$$



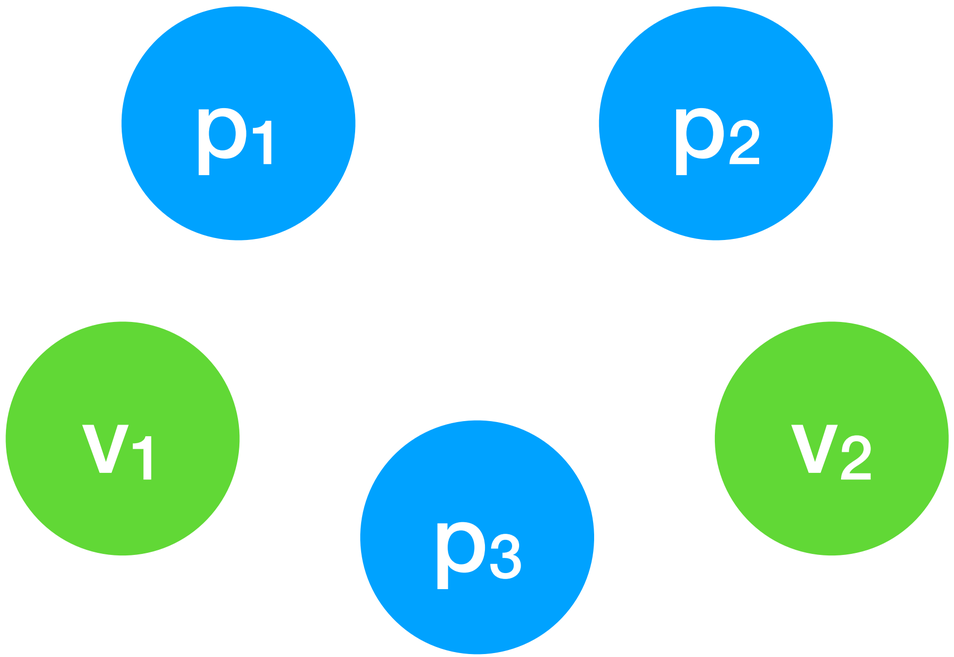
$$v_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, n_{\mathrm{tracks}}, \cos\theta_{\mathrm{PV}}, \ldots]$$

PARTICLES AND SECONDARY VERTICES: TWO INPUT GRAPHS

arXiv:1909.12285

- Particles (i.e. tracks) and vertices are two separate inputs with different feature vectors (*heterogenous graph*)

$$p_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, d_{\mathrm{3D}}, \mathrm{cov}(p_{\mathrm{T}}, p_{\mathrm{T}}), \ldots]$$



$$v_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, n_{\mathrm{tracks}}, \cos\theta_{\mathrm{PV}}, \ldots]$$

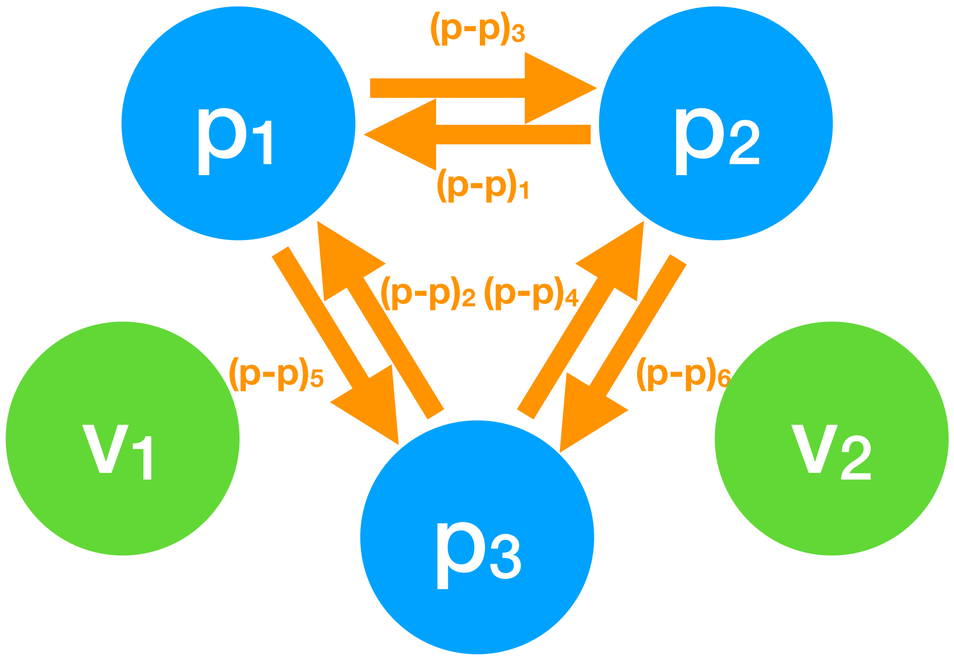
PARTICLES AND SECONDARY VERTICES: TWO INPUT GRAPHS

arXiv:1909.12285

- Particles (i.e. tracks) and vertices are two separate inputs with different feature vectors (*heterogenous graph*)

- GNNs typically consider a *homogenous graph* (e.g. particle-particle graph)

$$p_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, d_{\mathrm{3D}}, \mathrm{cov}(p_{\mathrm{T}}, p_{\mathrm{T}}), \ldots]$$

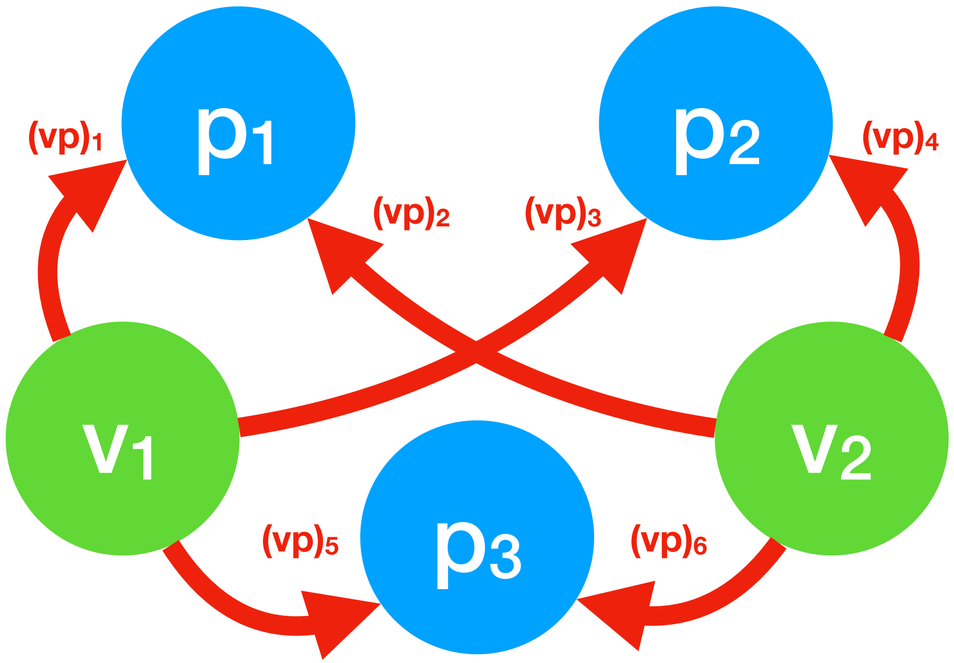


$$v_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, n_{\mathrm{tracks}}, \cos\theta_{\mathrm{PV}}, \ldots]$$

PARTICLES AND SECONDARY VERTICES: TWO INPUT GRAPHS

- Particles (i.e. tracks) and vertices are two separate inputs with different feature vectors (*heterogenous graph*)
- GNNs typically consider a *homogenous graph* (e.g. particle-particle graph)
- Vertex-particle graph can also be considered

$$p_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, d_{\mathrm{3D}}, \mathrm{cov}(p_{\mathrm{T}}, p_{\mathrm{T}}), \ldots]$$



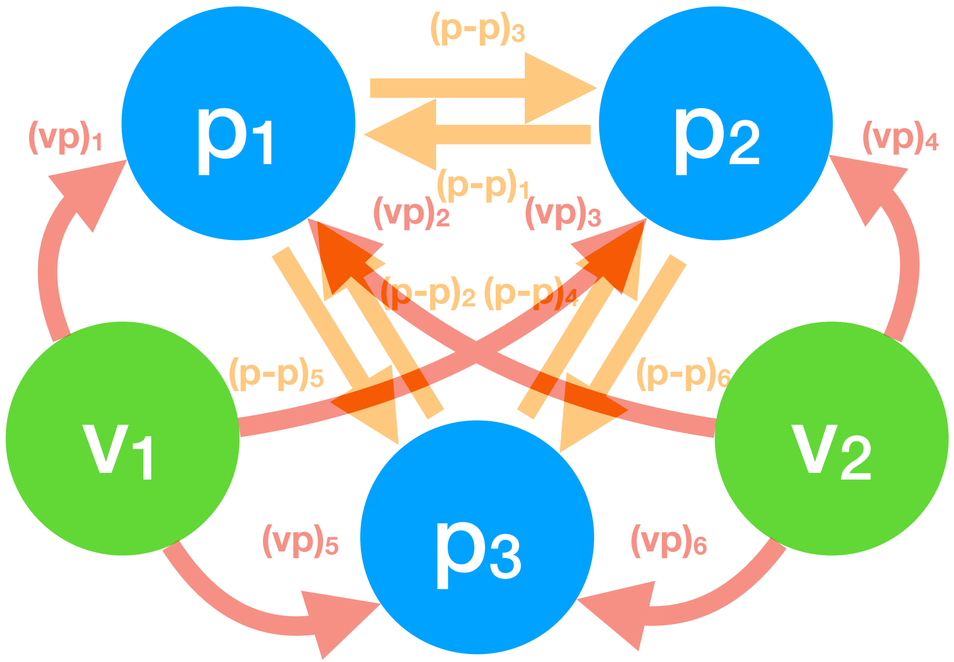
$$v_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, n_{\mathrm{tracks}}, \cos\theta_{\mathrm{PV}}, \ldots]$$

PARTICLES AND SECONDARY VERTICES: TWO INPUT GRAPHS

arXiv:1909.12285

- Particles (i.e. tracks) and vertices are two separate inputs with different feature vectors (*heterogenous graph*)
- GNNs typically consider a *homogenous graph* (e.g. particle-particle graph)
- Vertex-particle graph can also be considered
- Combined GNN can consider both by constructing two separate graphs

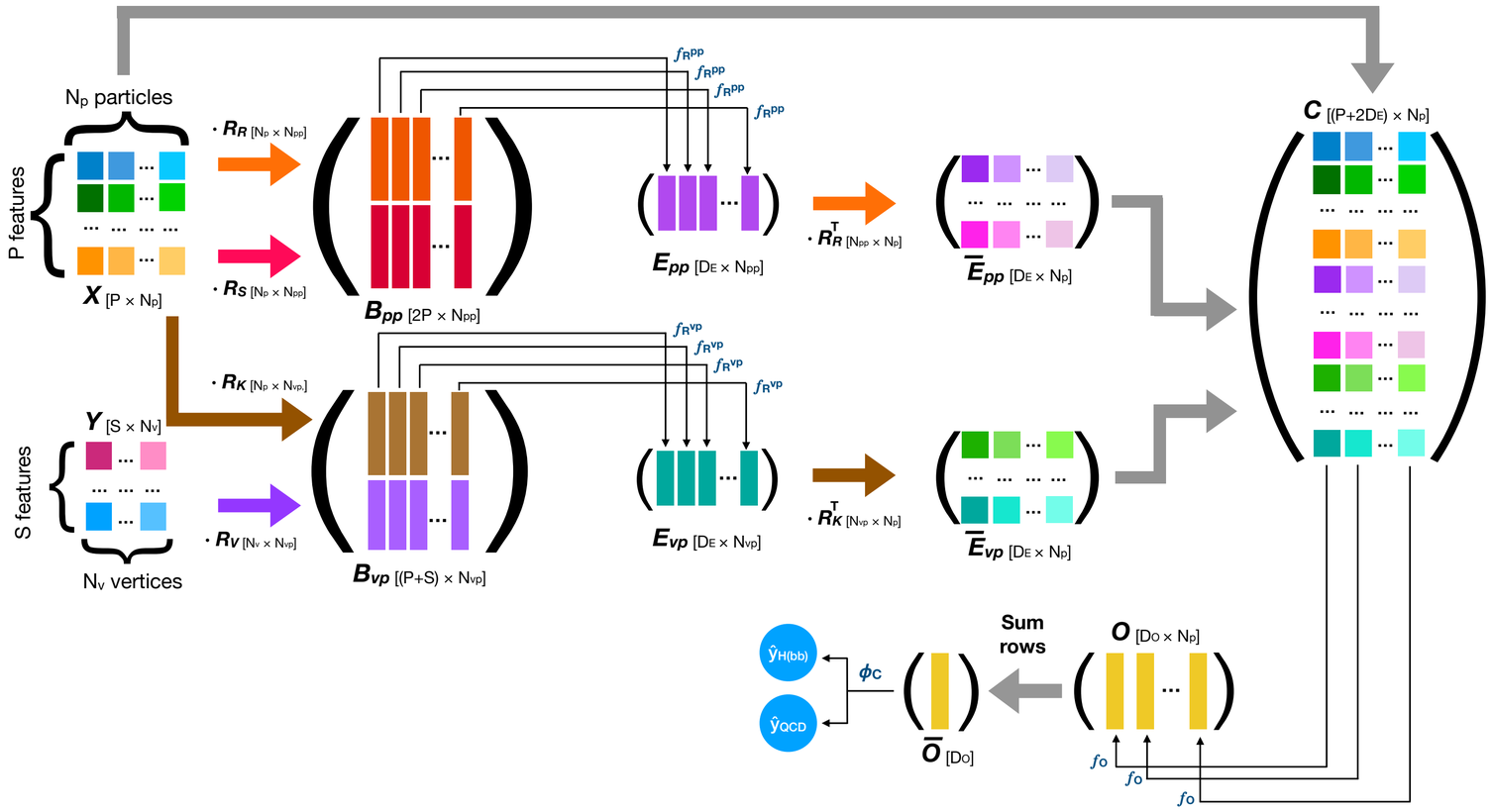
$$p_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, d_{\mathrm{3D}}, \mathrm{cov}(p_{\mathrm{T}}, p_{\mathrm{T}}), \ldots]$$

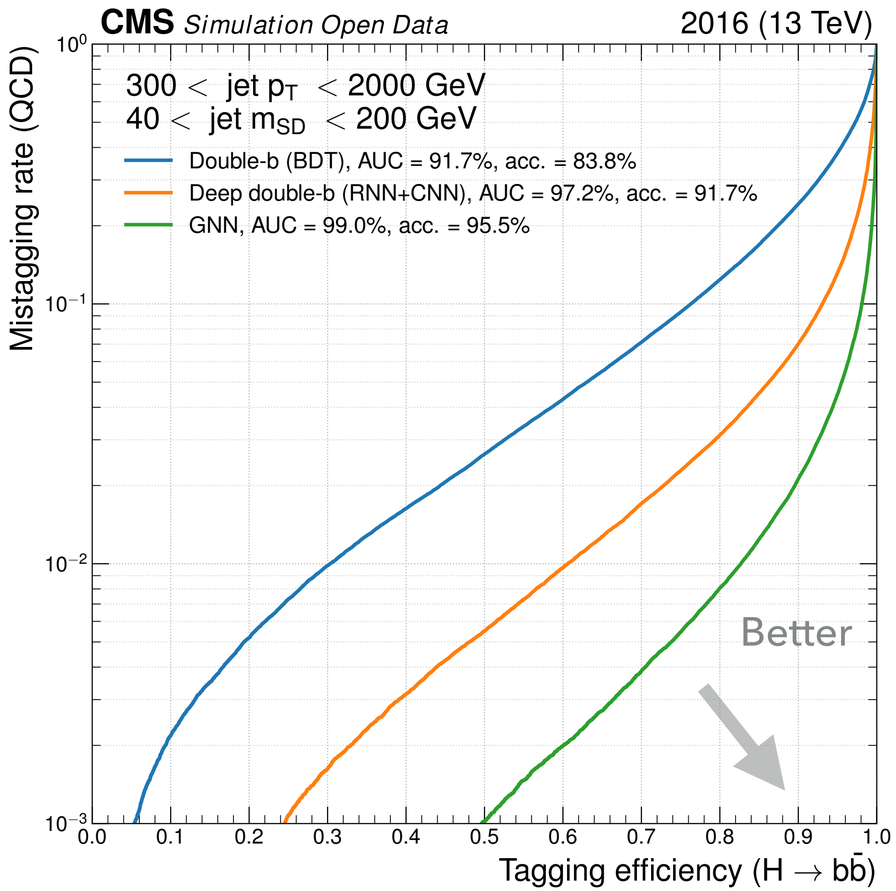


$$v_i = [p_{\mathrm{T}}^{\mathrm{rel}}, \phi^{\mathrm{rel}}, \eta^{\mathrm{rel}}, \ldots, n_{\mathrm{tracks}}, \cos\theta_{\mathrm{PV}}, \ldots]$$

MODEL AND PERFORMANCE

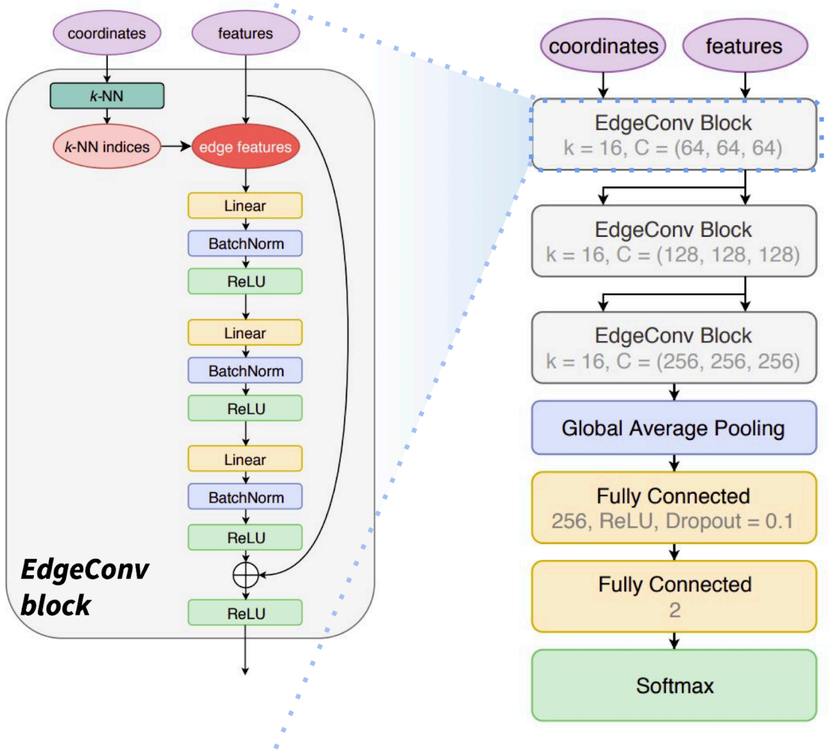
- Edge convolutions for particle-particle and particle-vertex connections update particle features; summed particle features used to predict H(bb) or QCD prob.
- GNN improves on previous methods



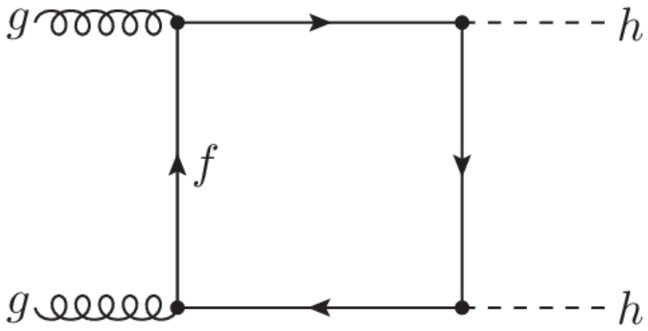


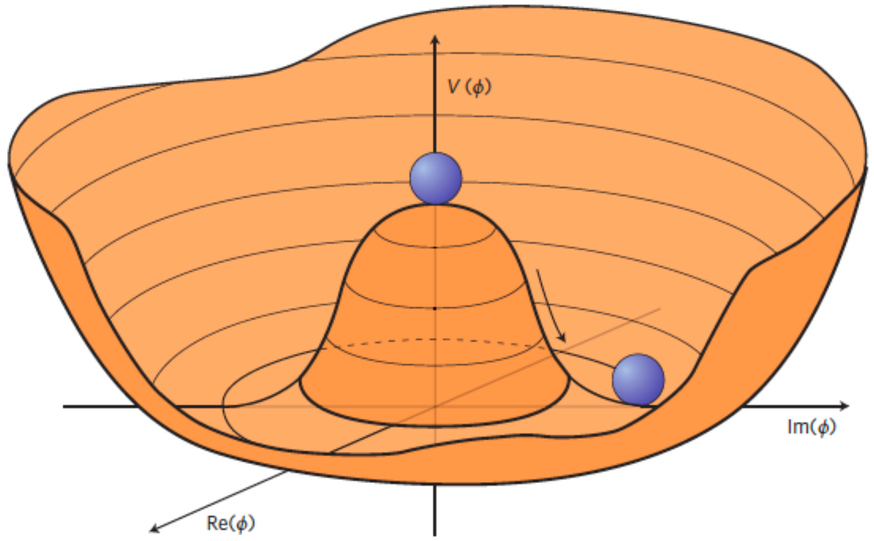
IMPACT

- CMS search for nonresonant SM HH→4b using ParticleNet (similar GNN) as mass-independent H(bb) jet tagger
  - Improves S/B by factor of ~2

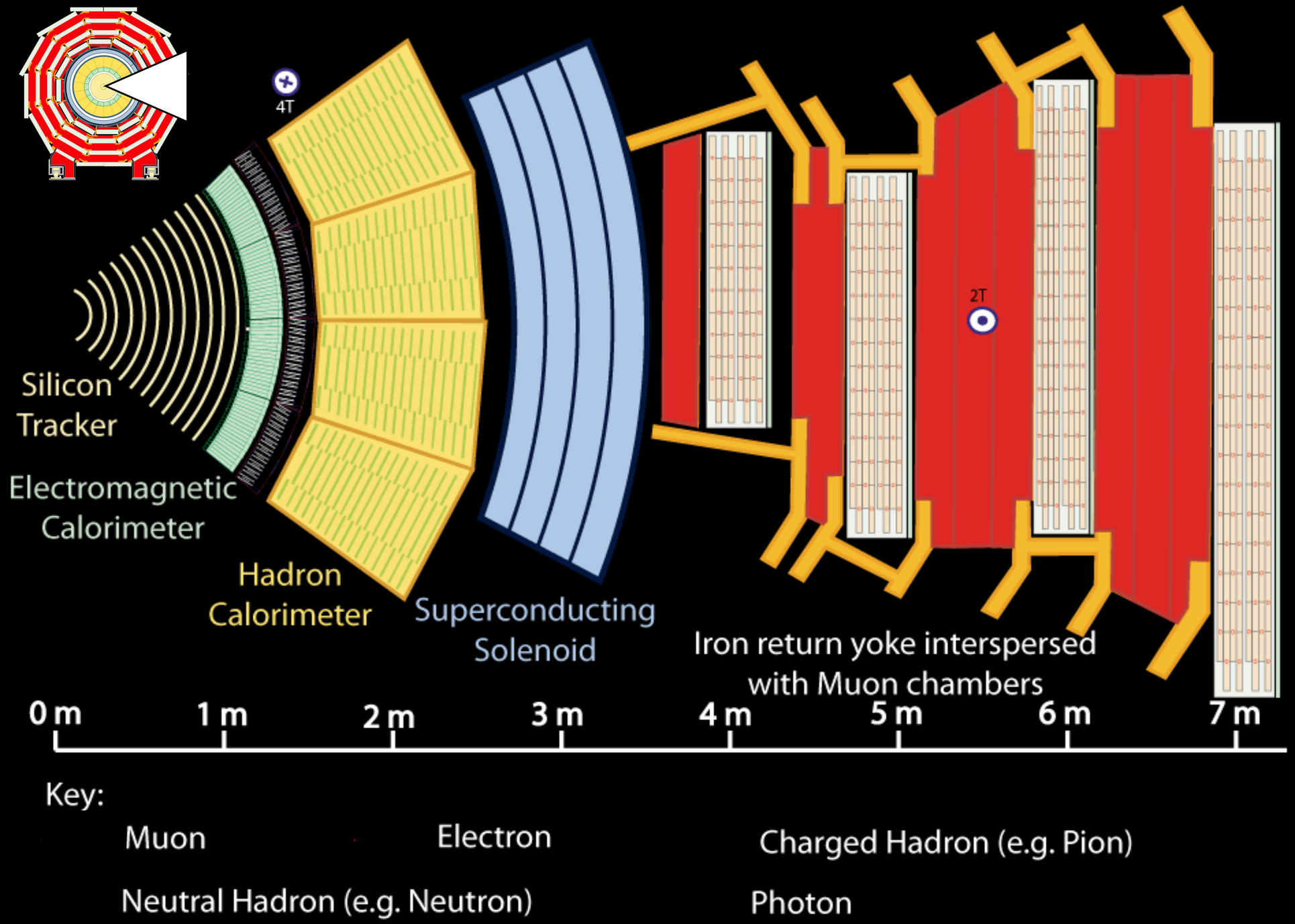


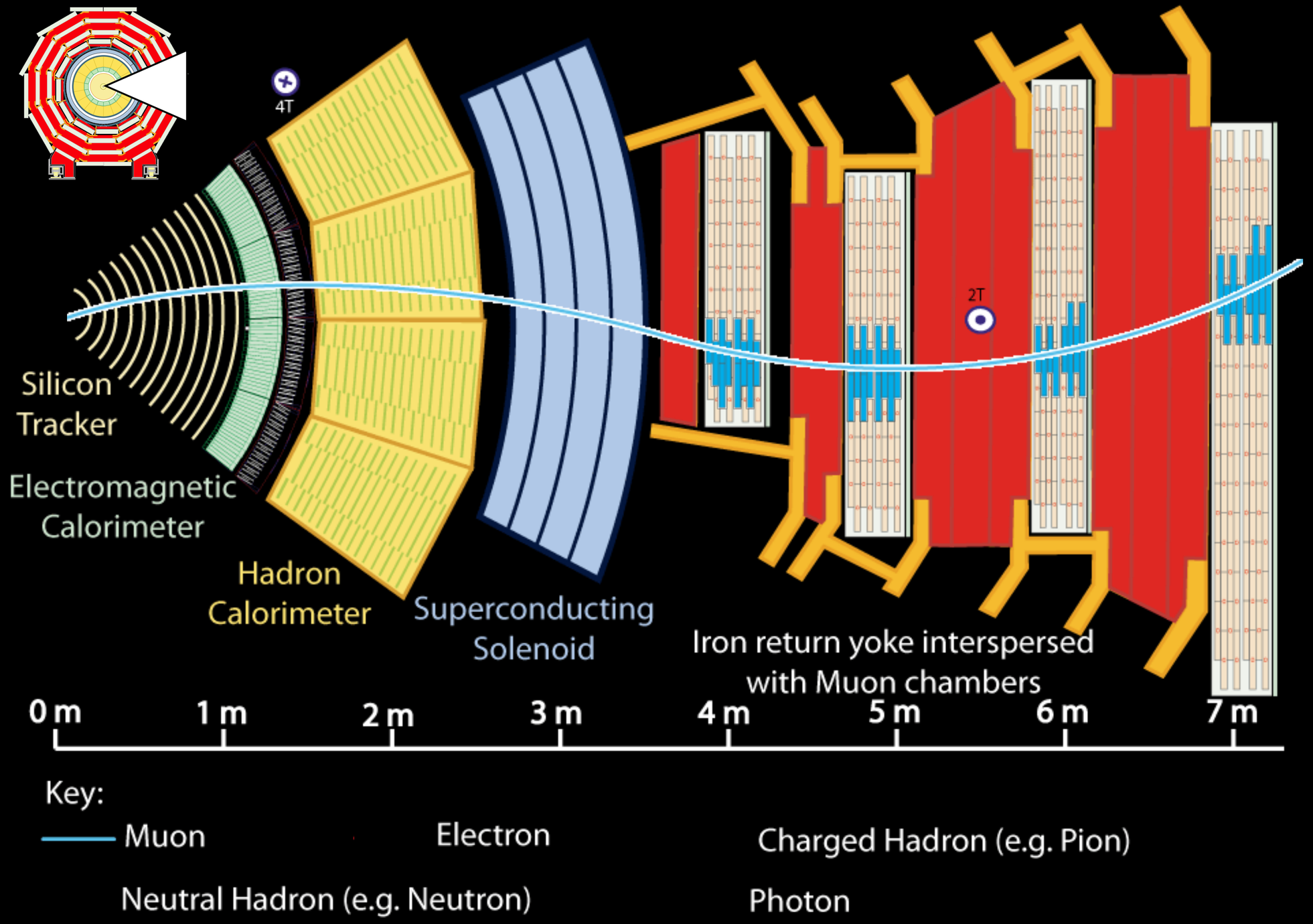
ParticleNet architecture
(applied to CMS)

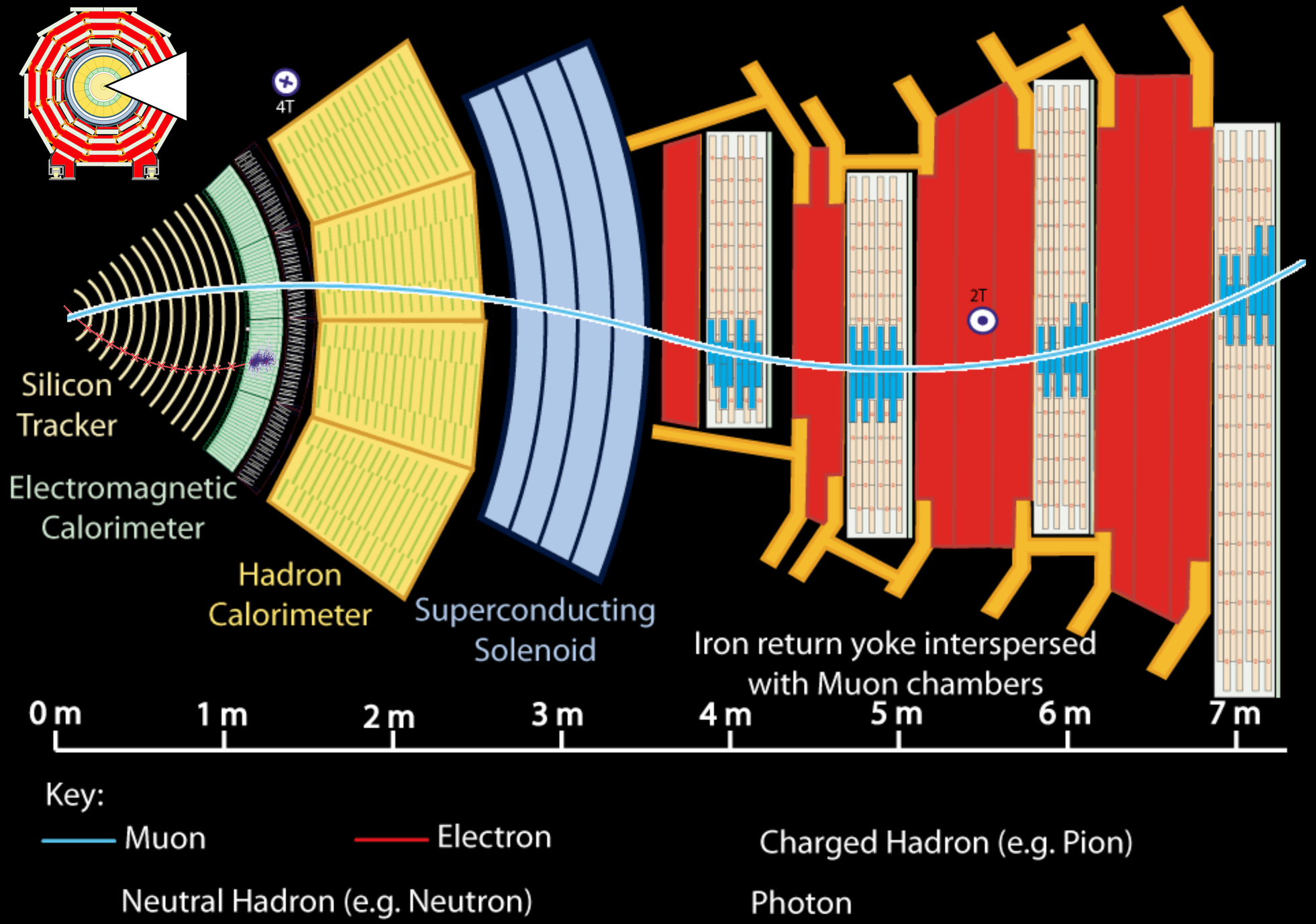


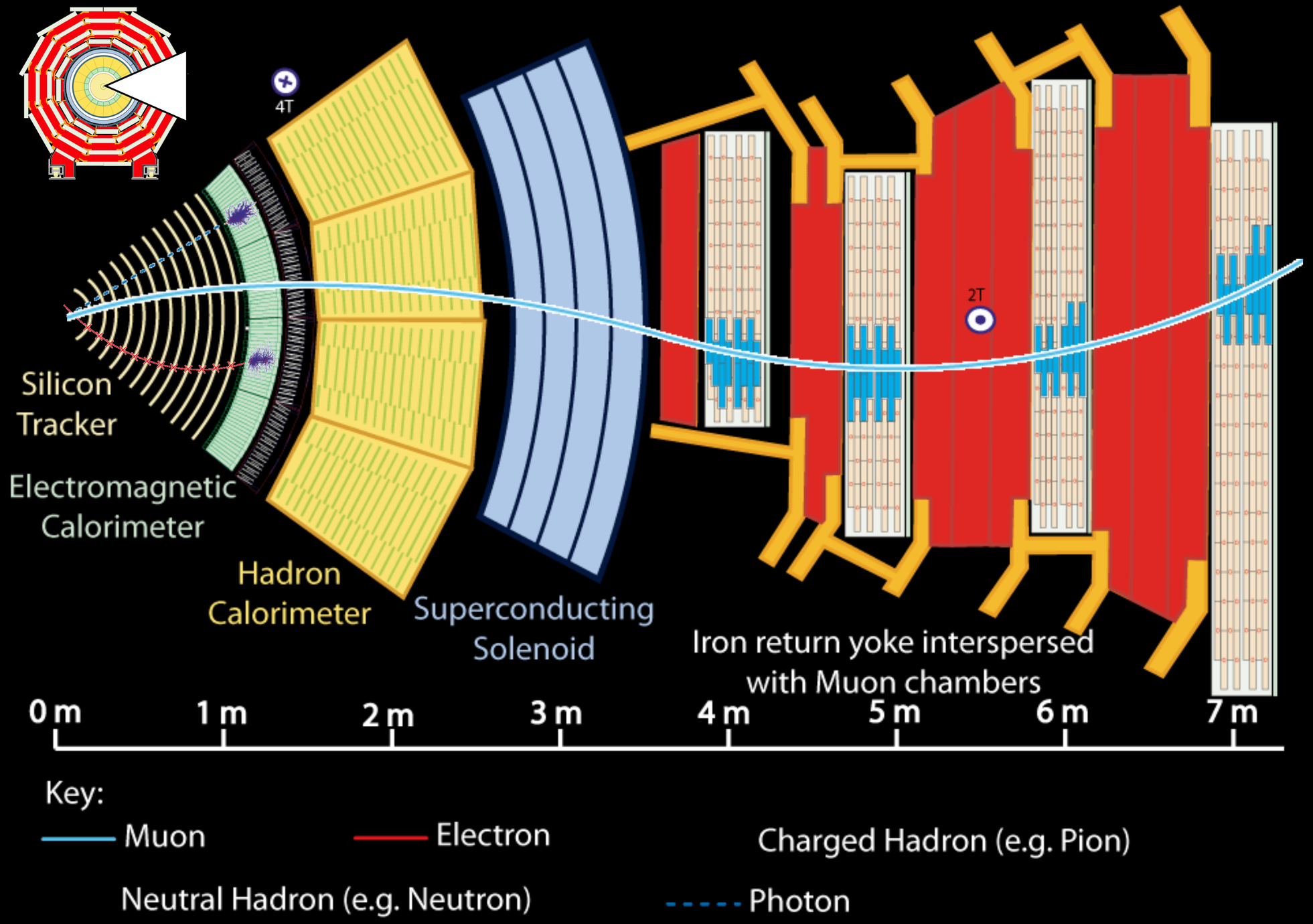


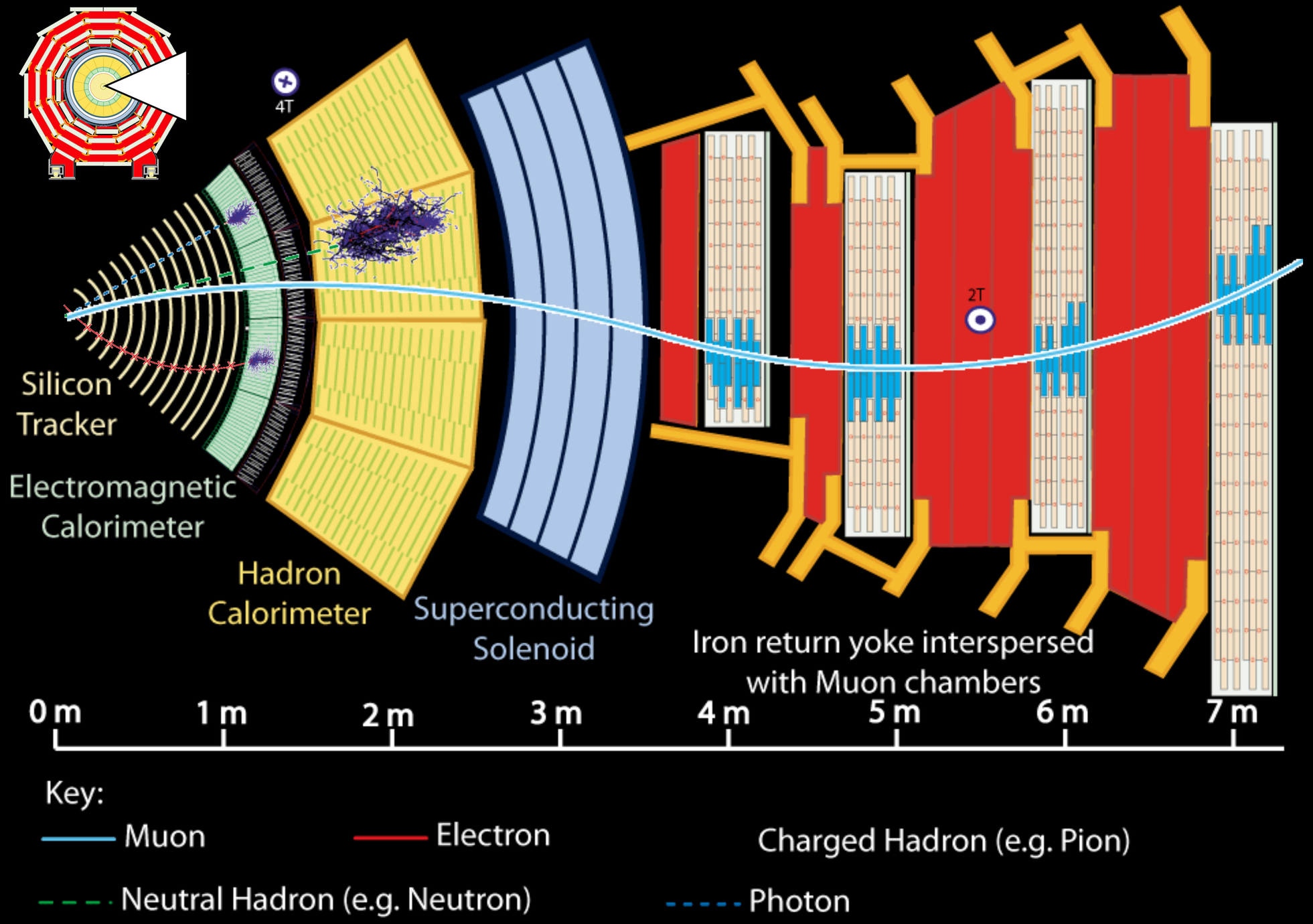
PARTICLE-FLOW

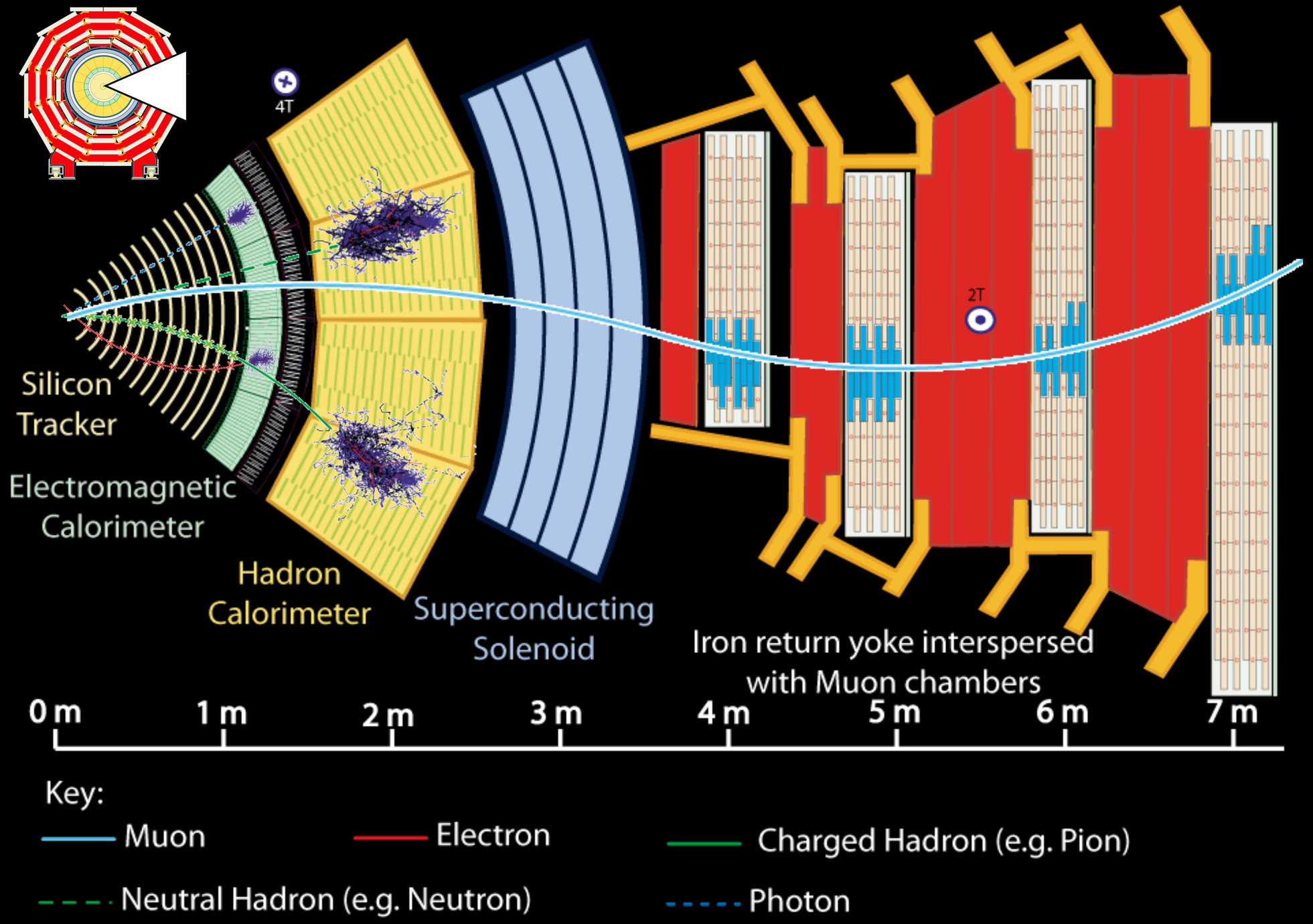

4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
0 m
1 m
2 m
3 m
4 m
5 m
6 m
7 m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon


4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
0 m
1 m
2 m
3 m
4 m
5 m
6 m
7 m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon


4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
0 m
1 m
2 m
3 m
4 m
5 m
6 m
7 m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon

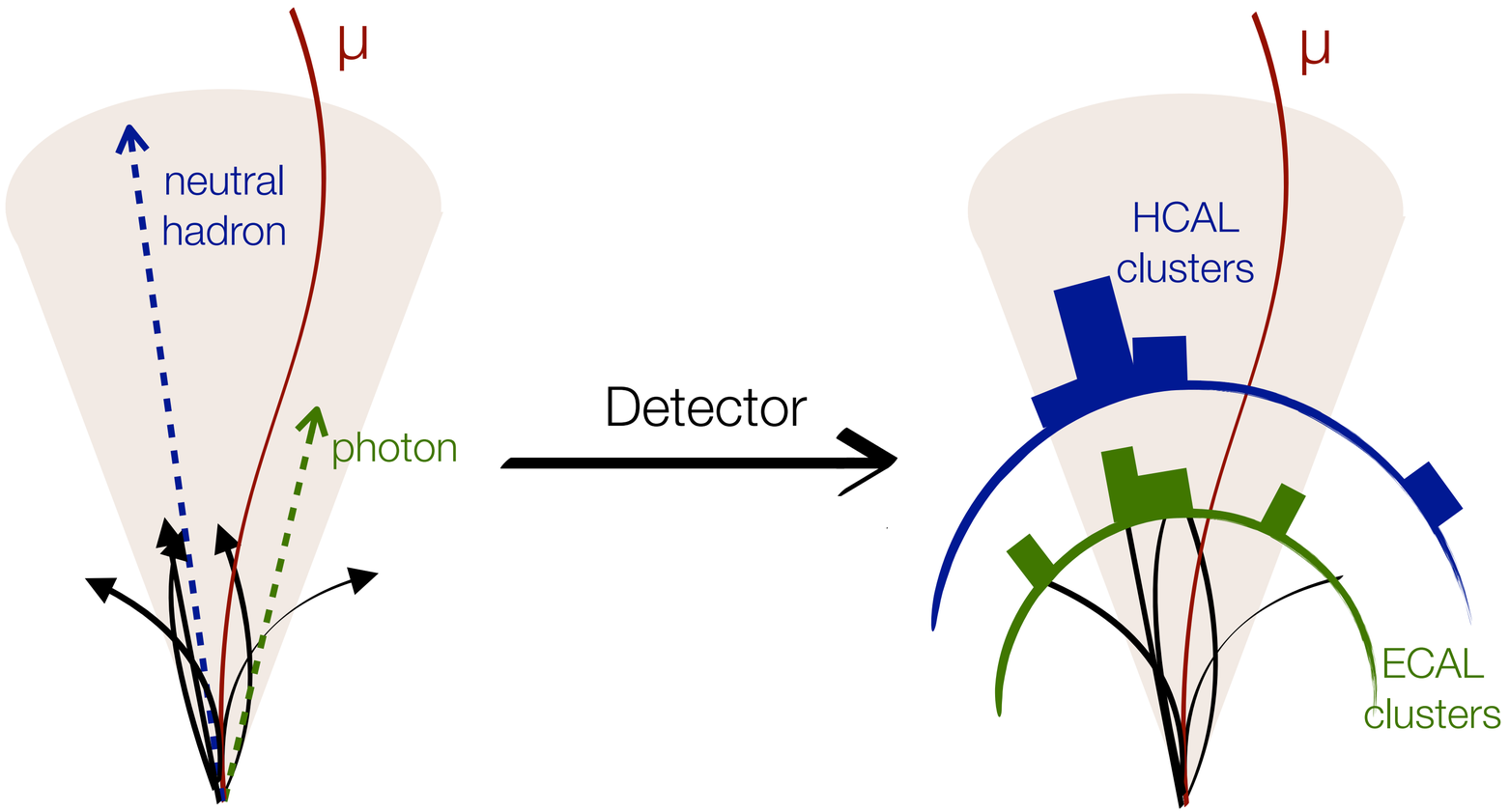

4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
0 m
1 m
2 m
3 m
4 m
5 m
6 m
7 m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon


4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
0 m
1 m
2 m
3 m
4 m
5 m
6 m
7 m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon


4T
2T
Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with Muon chambers
0 m
1 m
2 m
3 m
4 m
5 m
6 m
7 m
Key:
Muon
Electron
Charged Hadron (e.g. Pion)
Neutral Hadron (e.g. Neutron)
Photon

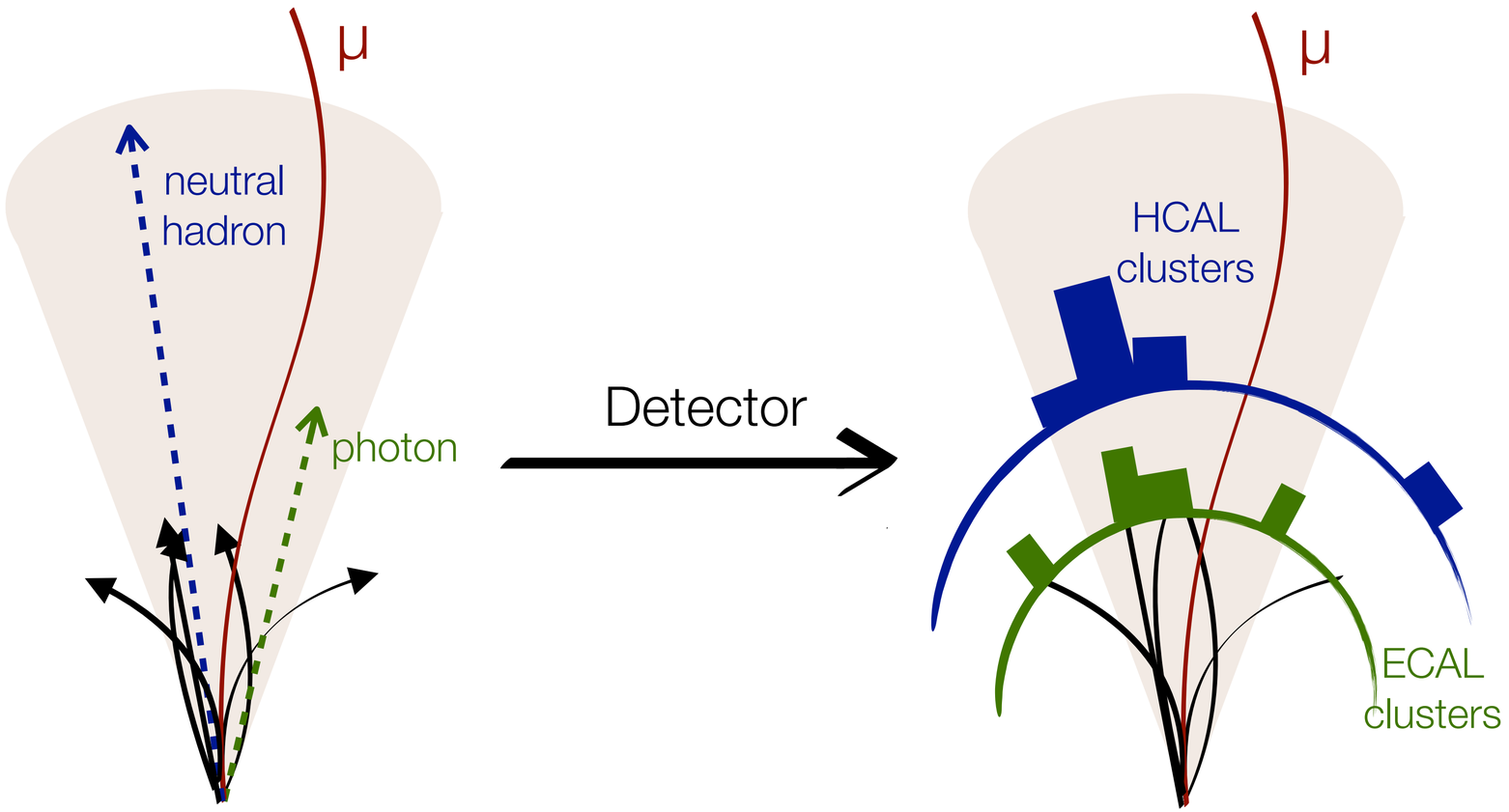
PARTICLE-FLOW RECONSTRUCTION

arXiv:1706.04965



PARTICLE-FLOW RECONSTRUCTION

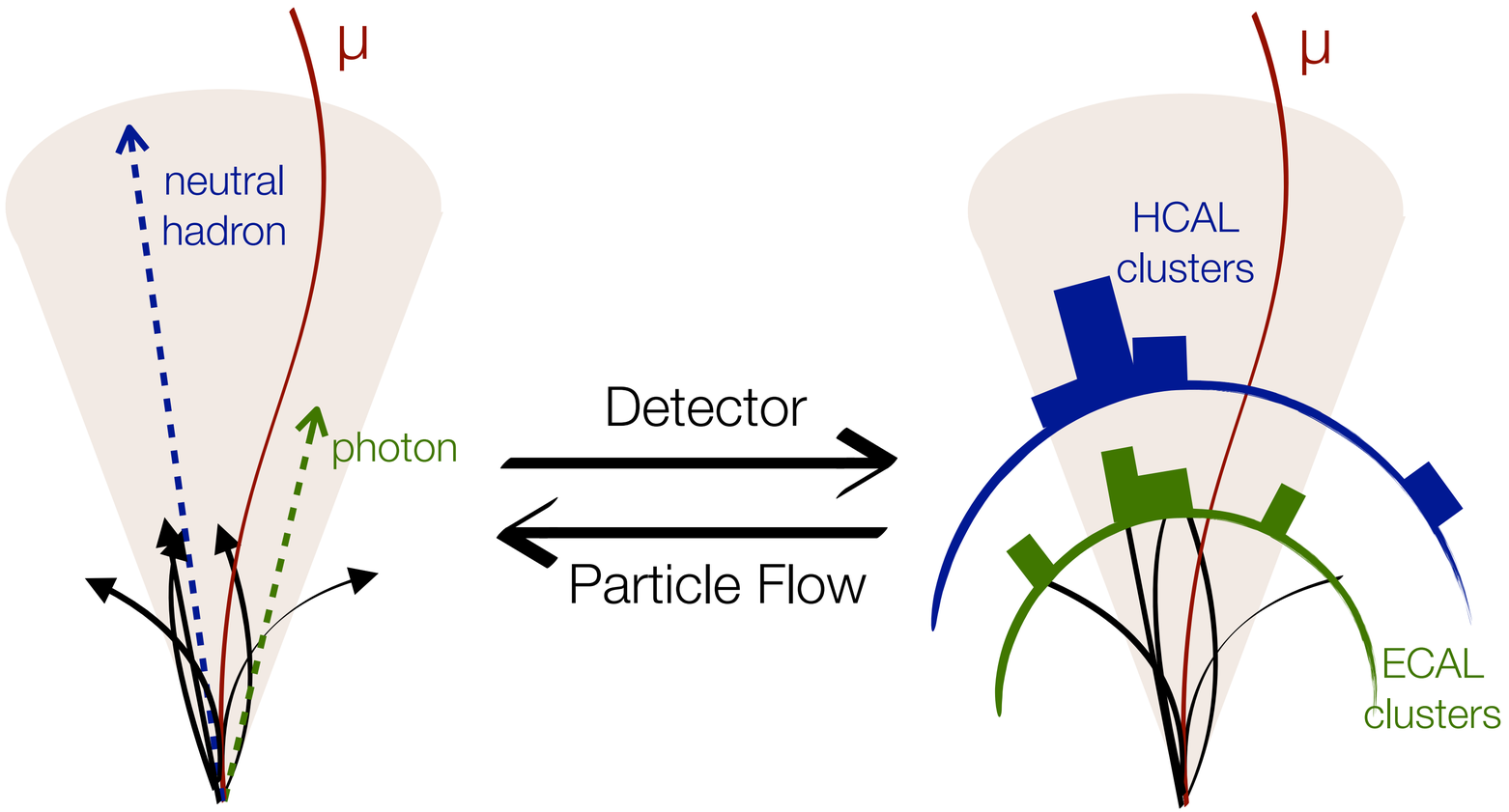
- Particles interact with detector, leaving energy deposits and tracks



PARTICLE-FLOW RECONSTRUCTION

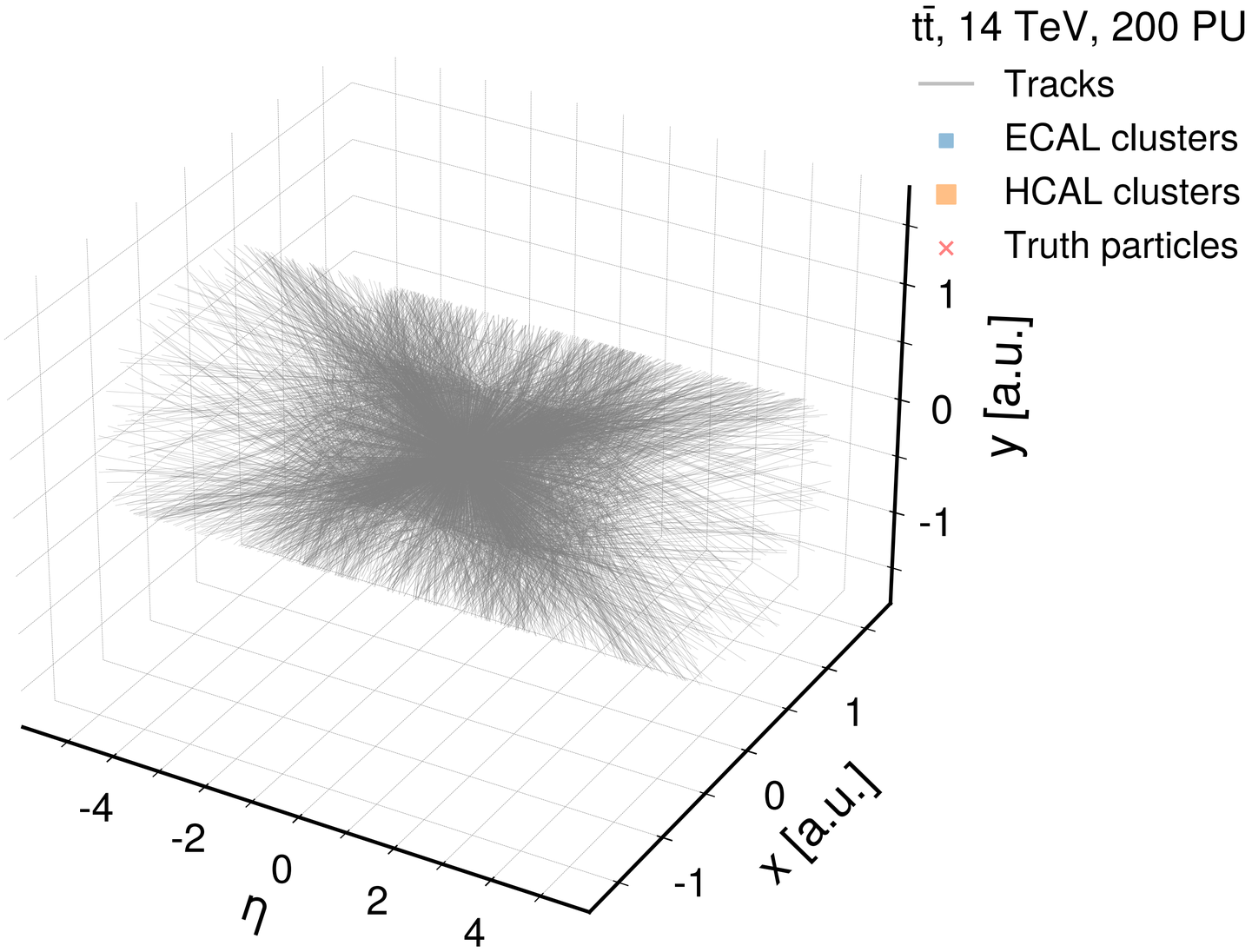
arXiv:1706.04965

- Particles interact with detector, leaving energy deposits and tracks
- Efficient combination of info. from complementary detector subsystems to produce a holistic, particle interpretation of the event (that improves on any individual subsystem)



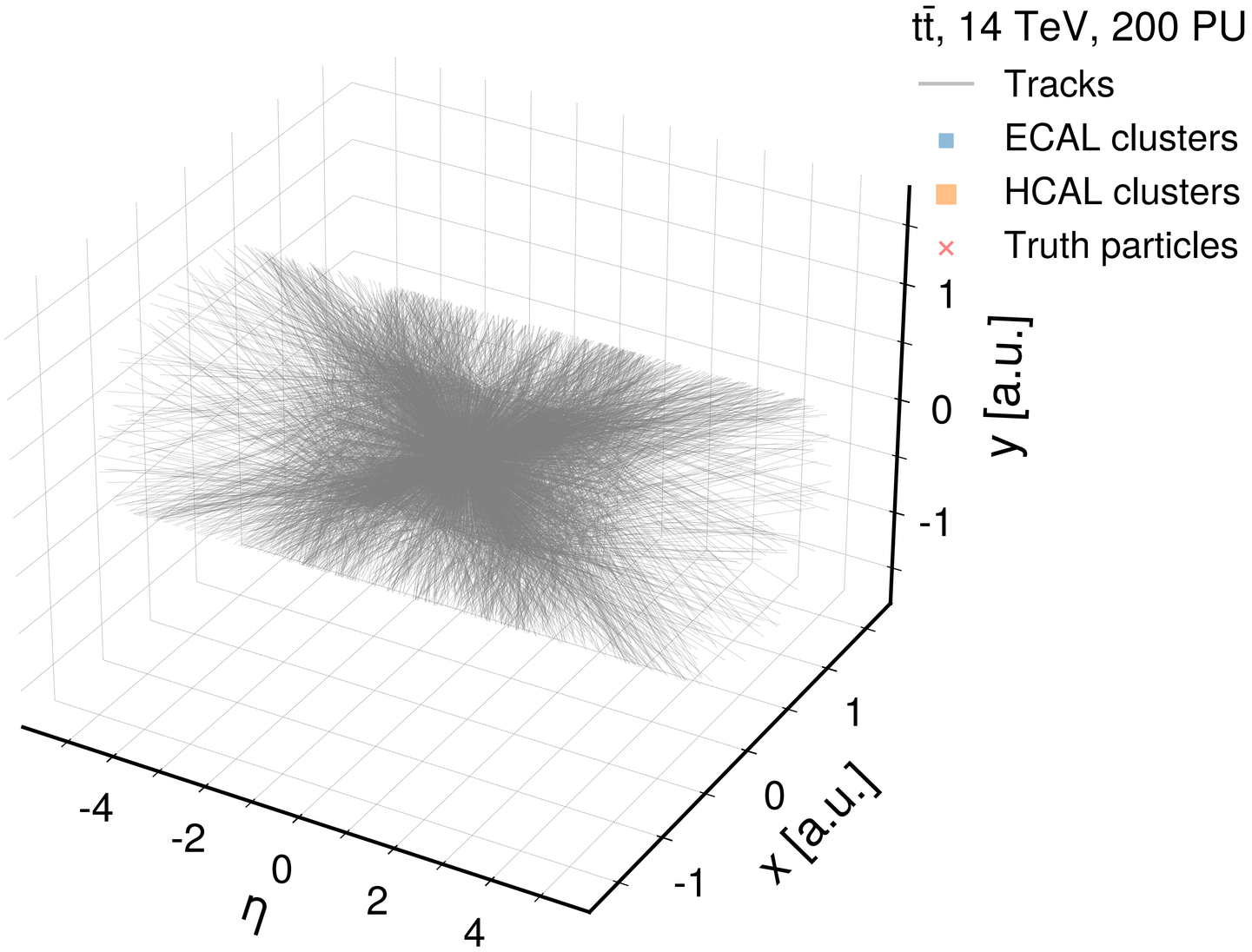
PARTICLE-FLOW AS ML TASK

arXiv:2101.08578



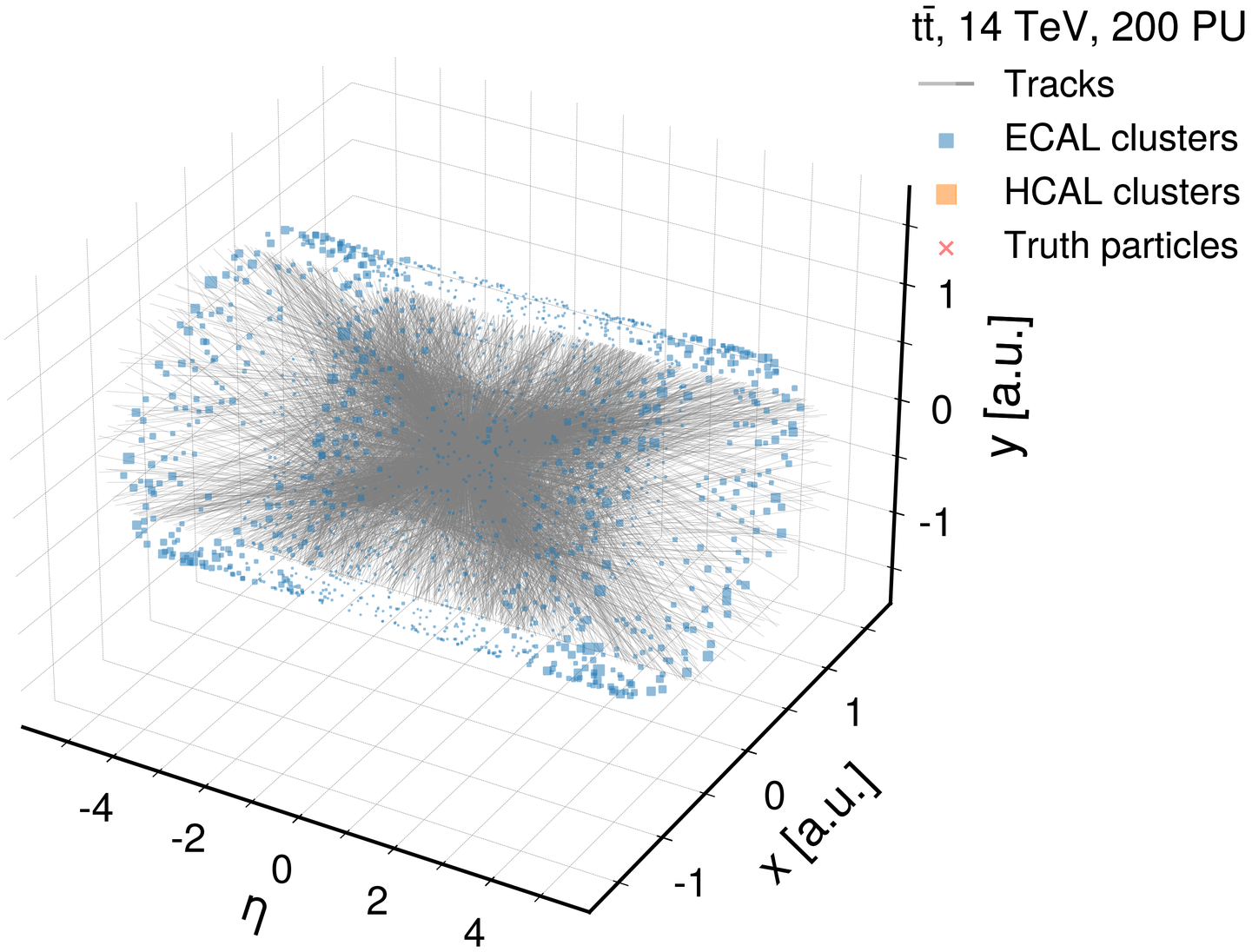
PARTICLE-FLOW AS ML TASK

- Input set (or point could or heterogeneous graph) of tracks,



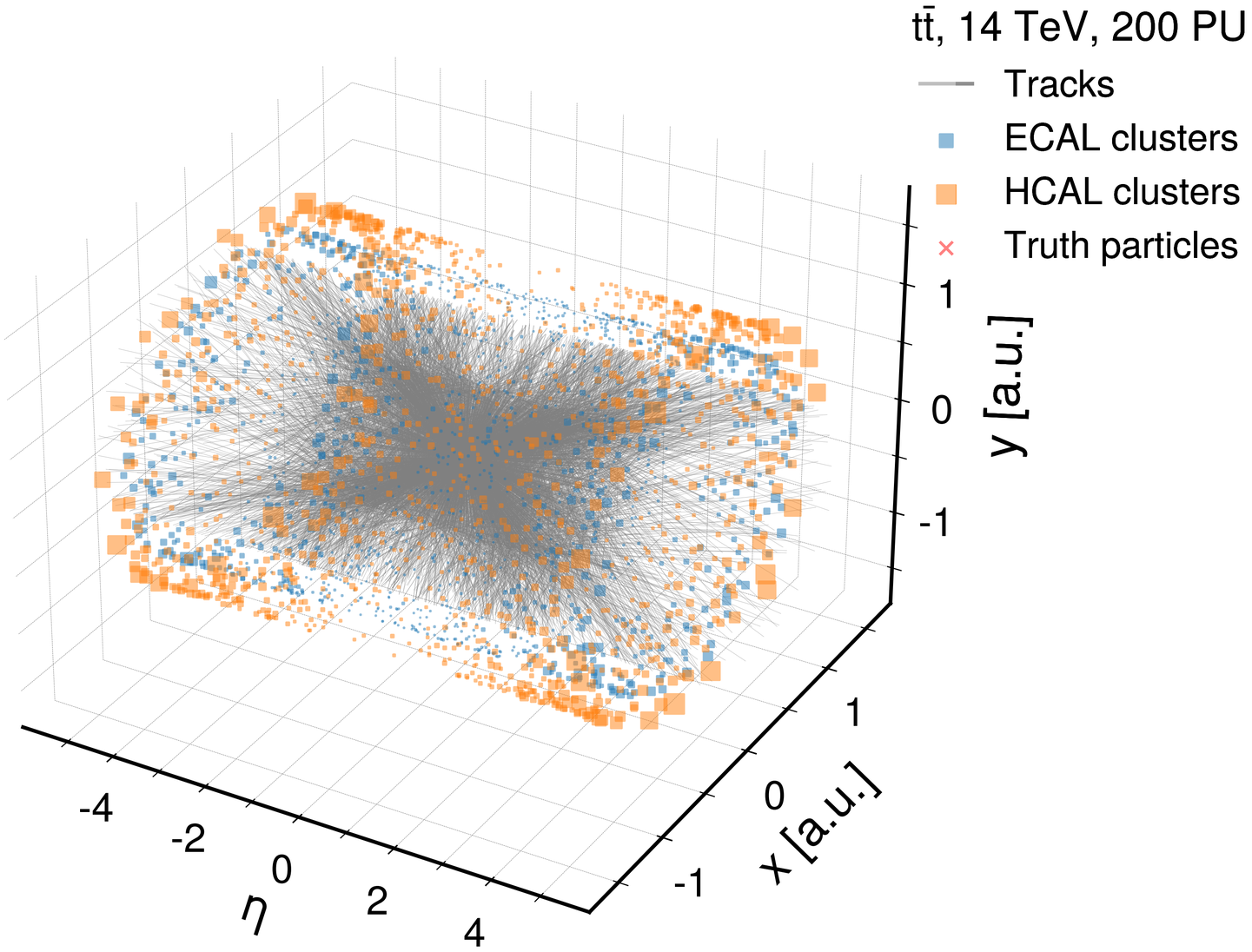
PARTICLE-FLOW AS ML TASK

- Input set (or point could or heterogeneous graph) of tracks, ECAL clusters,



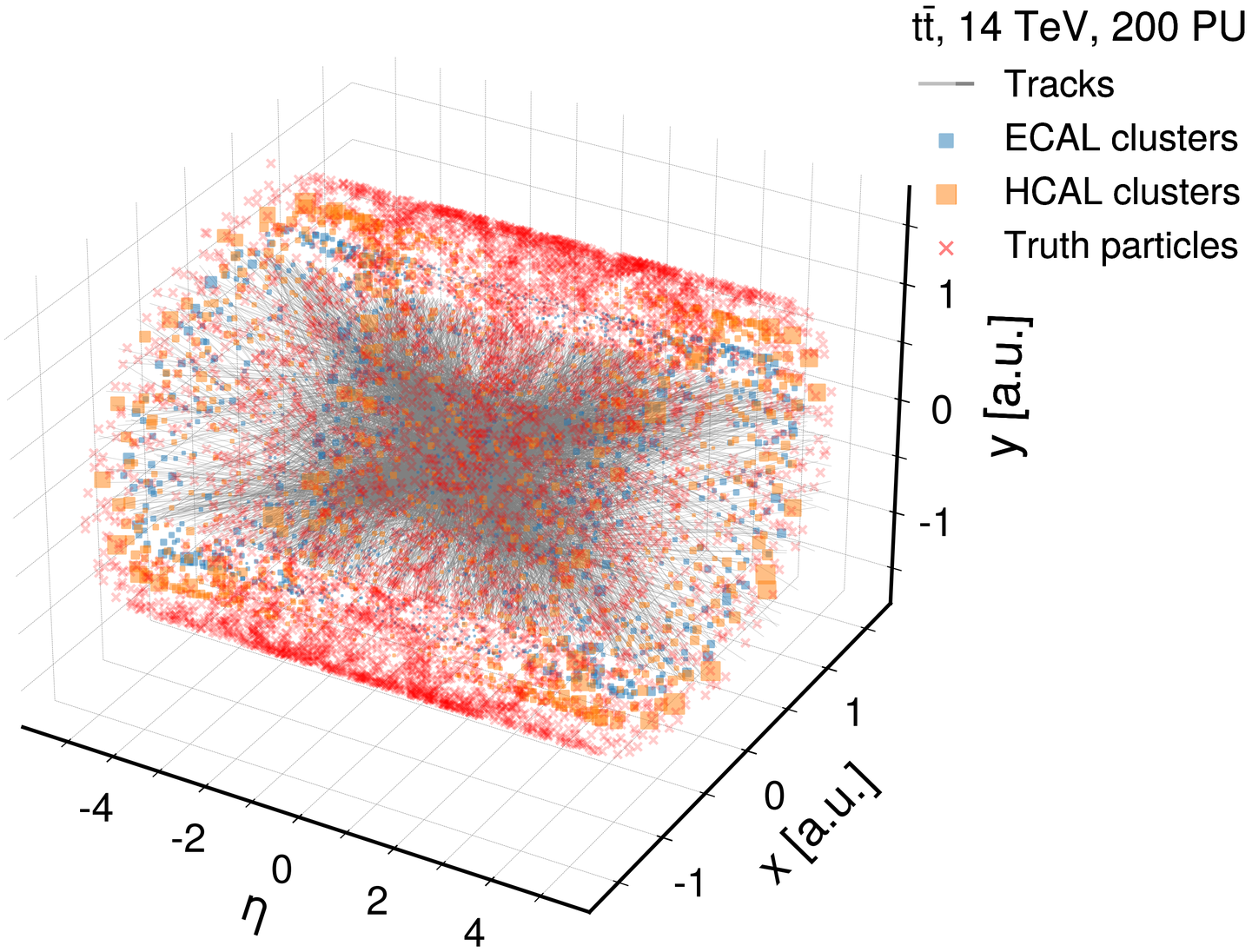
PARTICLE-FLOW AS ML TASK

- Input set (or point could or heterogeneous graph) of tracks, ECAL clusters, HCAL clusters



PARTICLE-FLOW AS ML TASK

- Input set (or point could or heterogeneous graph) of tracks, ECAL clusters, HCAL clusters
- Target set of truth particles $Y = \{y_i\}$

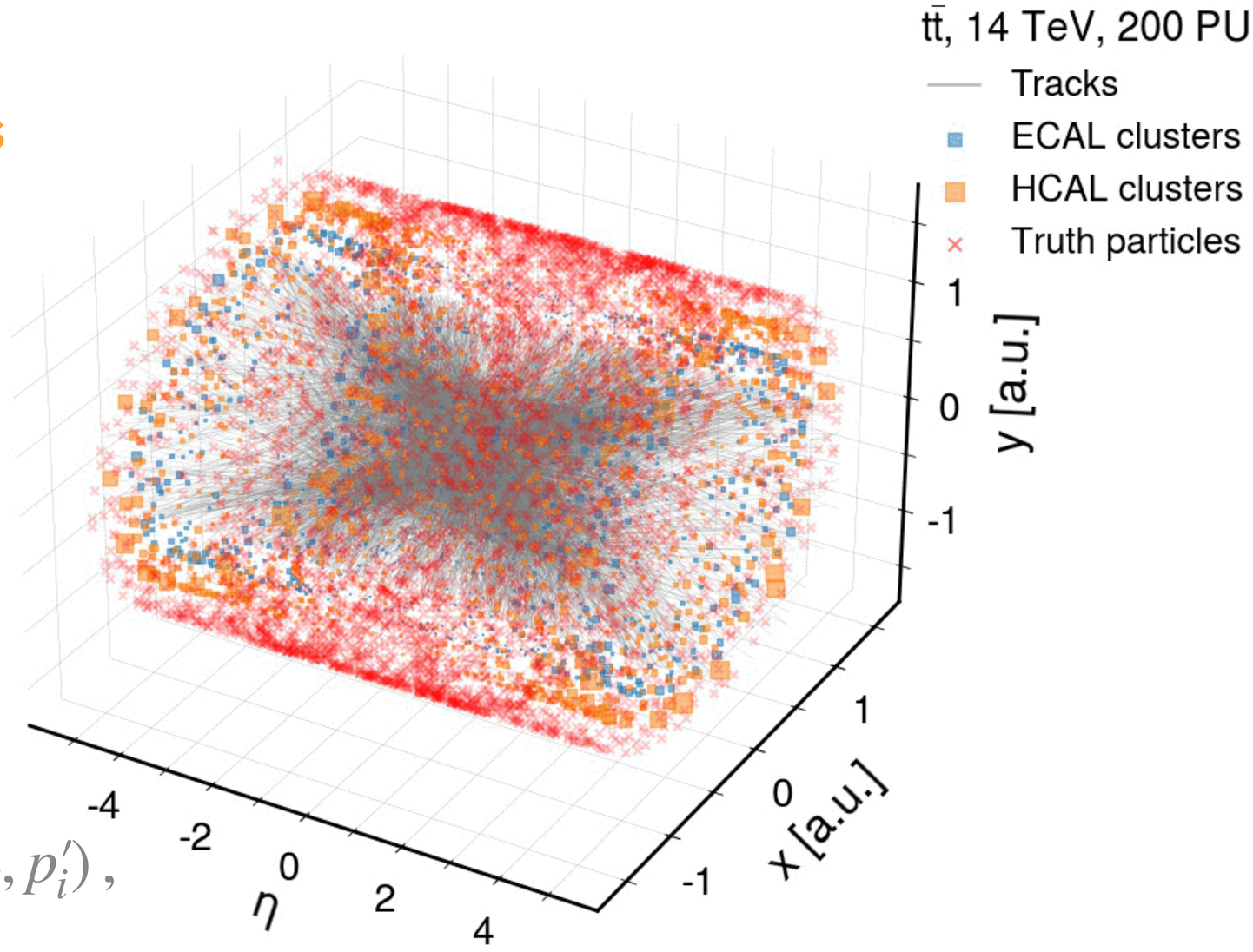


PARTICLE-FLOW AS ML TASK

- Input set (or point could or heterogeneous graph) of tracks, ECAL clusters, HCAL clusters
- Target set of truth particles $Y = \{y_i\}$
- Goal: construct a mapping $\mathcal{U}(X) = Y' \sim Y$ that minimizes some distance

$$||Y - Y'|| \equiv \sum_{i \in \text{event}} L(y_i, y_i') \,,$$

$$L(y_i, y_i') \equiv \text{CLS}(c_i, c_i') + \alpha \text{REG}(p_i, p_i') \,,$$

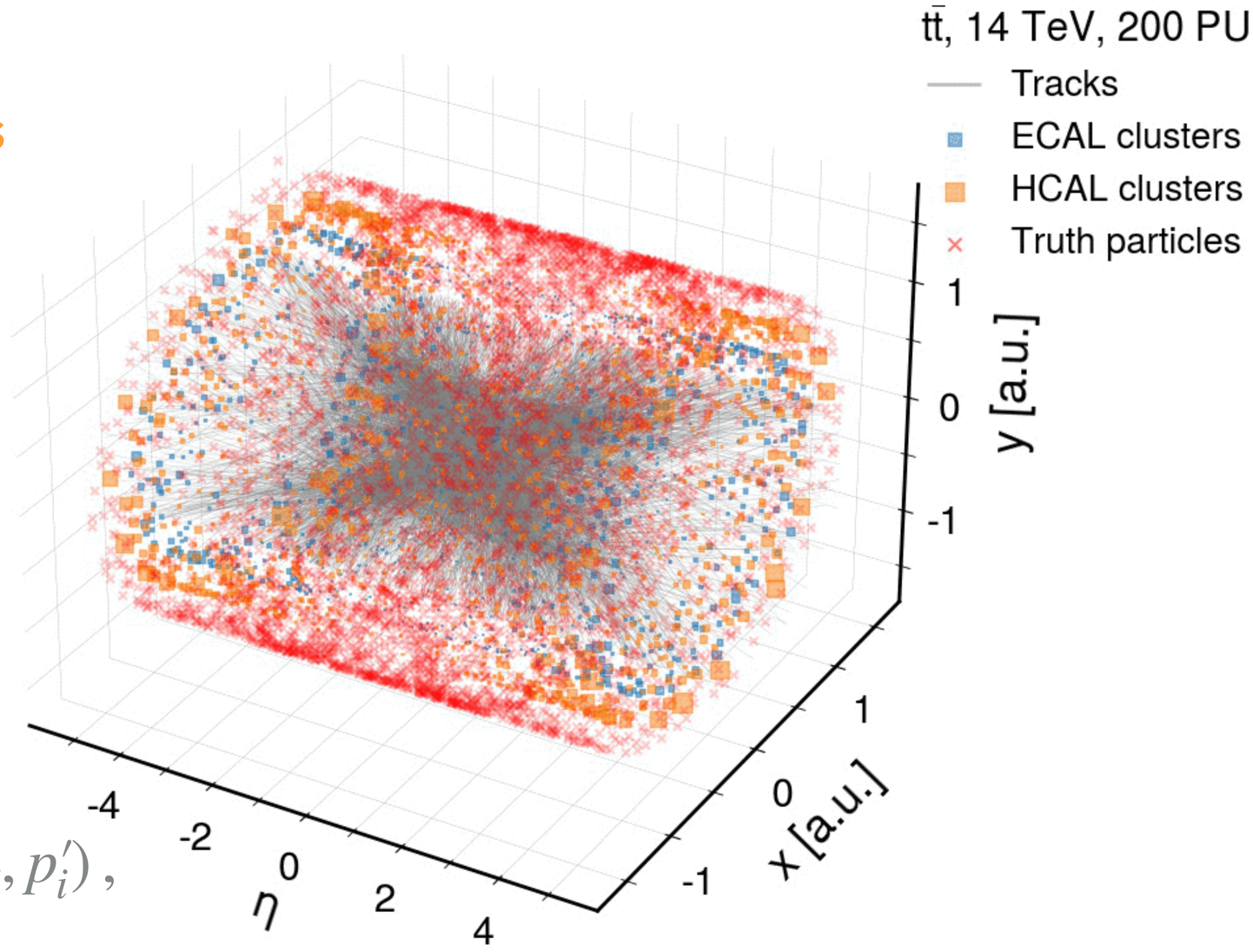


PARTICLE-FLOW AS ML TASK

- Input set (or point could or heterogeneous graph) of tracks, ECAL clusters, HCAL clusters
- Target set of truth particles $Y = \{y_i\}$
- Goal: construct a mapping $\mathcal{U}(X) = Y' \sim Y$ that minimizes some distance

$$||Y - Y'|| \equiv \sum_{i \in \text{event}} L(y_i, y_i') \,,$$

$$L(y_i, y_i') \equiv \text{CLS}(c_i, c_i') + \alpha \text{REG}(p_i, p_i') \,,$$



DATA PREPROCESSING FOR NODE PREDICTION

DATA PREPROCESSING FOR NODE PREDICTION

- Formulate task as node prediction with data preprocessing

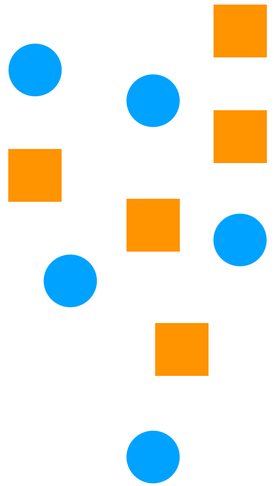
DATA PREPROCESSING FOR NODE PREDICTION

- Formulate task as node prediction with data preprocessing
- Input set arranged in (arbitrarily ordered) matrix

$$X = \begin{bmatrix} & \cdots & \\ \text{type}, p_{\text{T}}, E_{\text{ECAL}}, E_{\text{HCAL}}, \eta, \phi, \eta_{\text{outer}}, \phi_{\text{outer}}, q \\ & \cdots & \end{bmatrix}$$

$$\text{type} \in \{\text{track}, \text{cluster}\}$$

Input set $X = \{x_i\}$



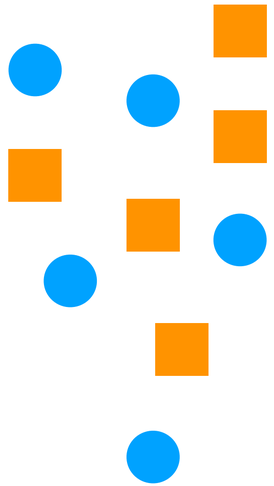
DATA PREPROCESSING FOR NODE PREDICTION

- Formulate task as node prediction with data preprocessing
- Input set arranged in (arbitrarily ordered) matrix

$$X = \begin{bmatrix} \cdots \\ \text{type}, p_{\mathrm{T}}, E_{\mathrm{ECAL}}, E_{\mathrm{HCAL}}, \eta, \phi, \eta_{\mathrm{outer}}, \phi_{\mathrm{outer}}, q \\ \cdots \end{bmatrix}$$

$$\text{type} \in \{\text{track}, \text{cluster}\}$$

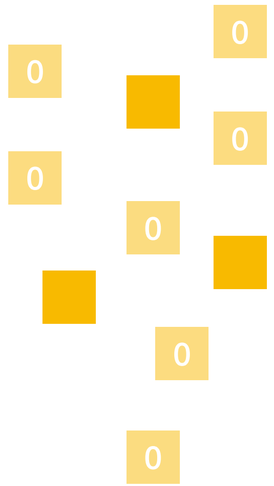
Input set $X = \{x_i\}$



- Target set zero-padded to same size $|X| = |Y|$, with each output particle arranged in same array position as best-matched input element

$$Y = \begin{bmatrix} \cdots \\ \text{PID}, p_{\mathrm{T}}, E, \eta, \phi, q \\ \cdots \end{bmatrix} \quad \text{PID} \in \{\text{none}, \text{charged hadron}, \text{neutral hadron}, \gamma, e^{\pm}, \mu^{\pm}\}$$

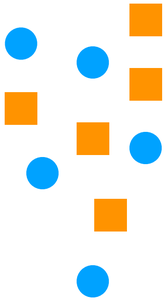
Target set $Y = \{y_i\}$



GRAPH NEURAL NETWORK APPROACH

Event as input set

$X = \{x_i\}$

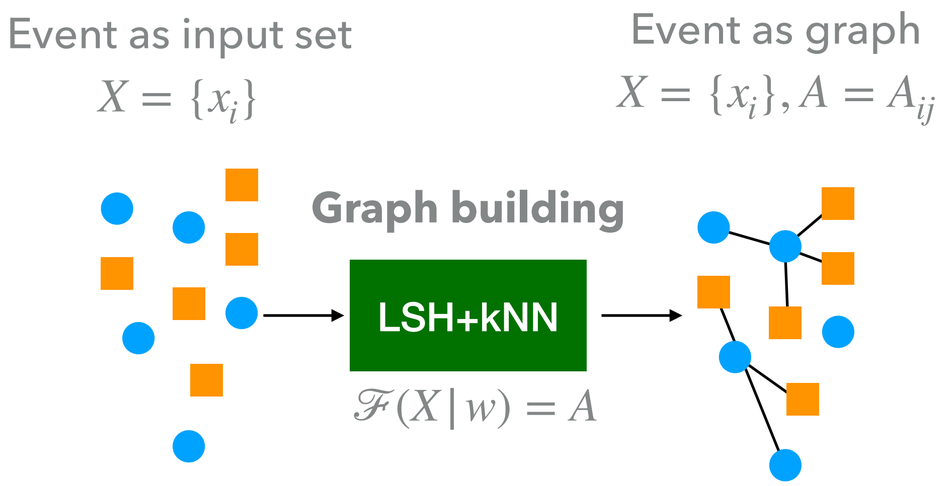


$h_i \in \mathbb{R}^N, N = 256$

Trainable neural networks: $\mathscr{F}, \mathscr{G}, \mathscr{D}$

GRAPH NEURAL NETWORK APPROACH

- Convert input set to a locally, **sparsely connected** graph

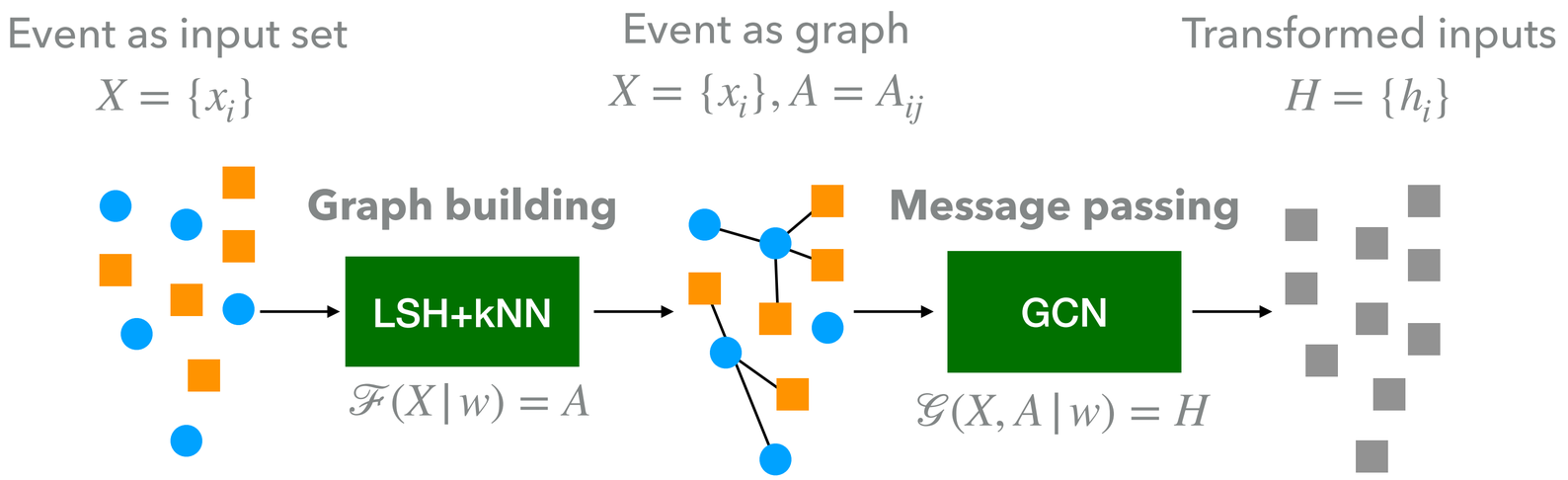


$h_i \in \mathbb{R}^N, N = 256$

Trainable neural networks: $\mathscr{F}, \mathscr{G}, \mathscr{D}$

GRAPH NEURAL NETWORK APPROACH

- Convert input set to a locally, **sparsely connected** graph
- Message-passing NN to transform features

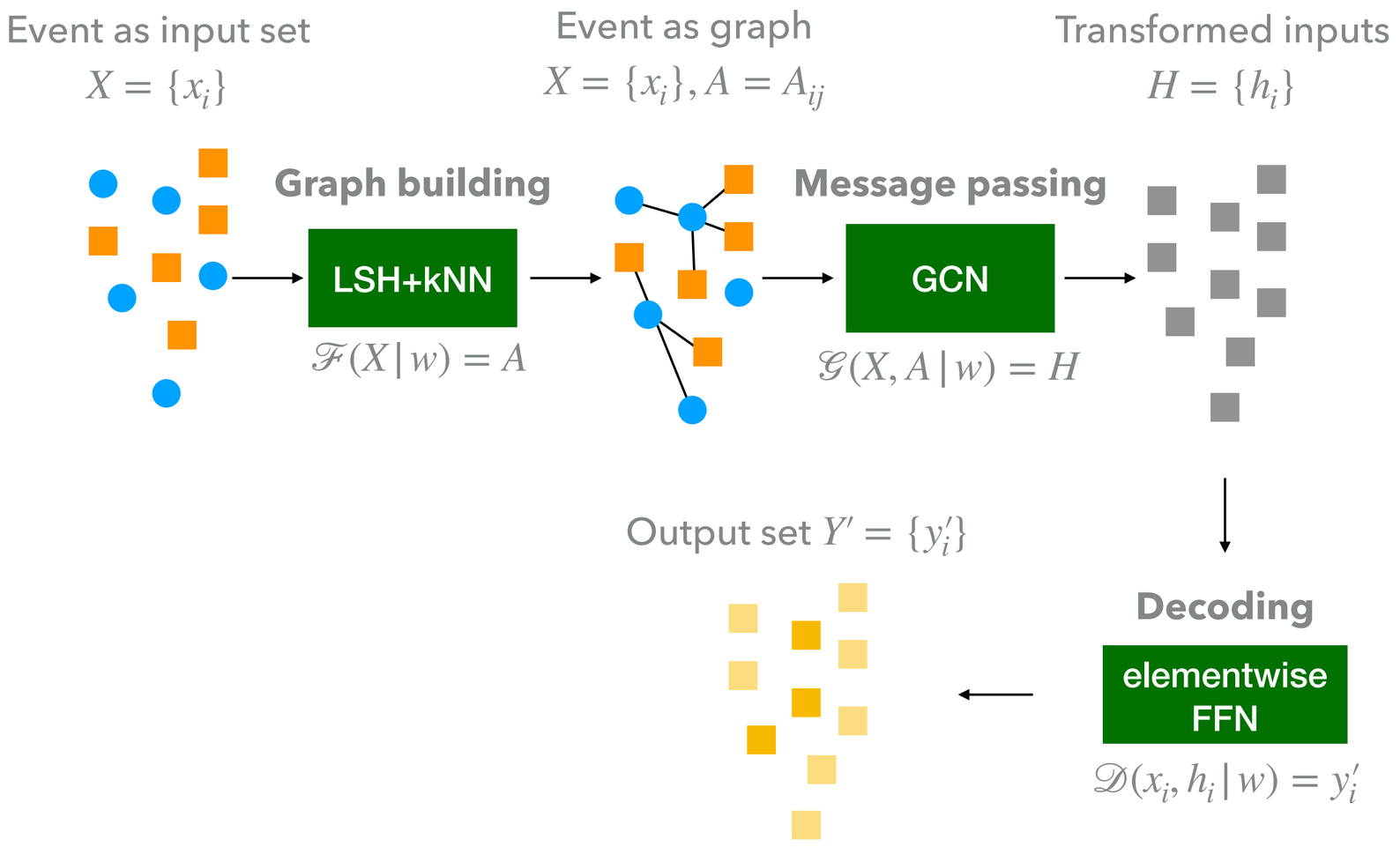


$h_i \in \mathbb{R}^N, N = 256$

Trainable neural networks: $\mathscr{F}, \mathscr{G}, \mathscr{D}$

GRAPH NEURAL NETWORK APPROACH

- Convert input set to a locally, **sparsely connected** graph
- Message-passing NN to transform features
- Decode transformed inputs elementwise

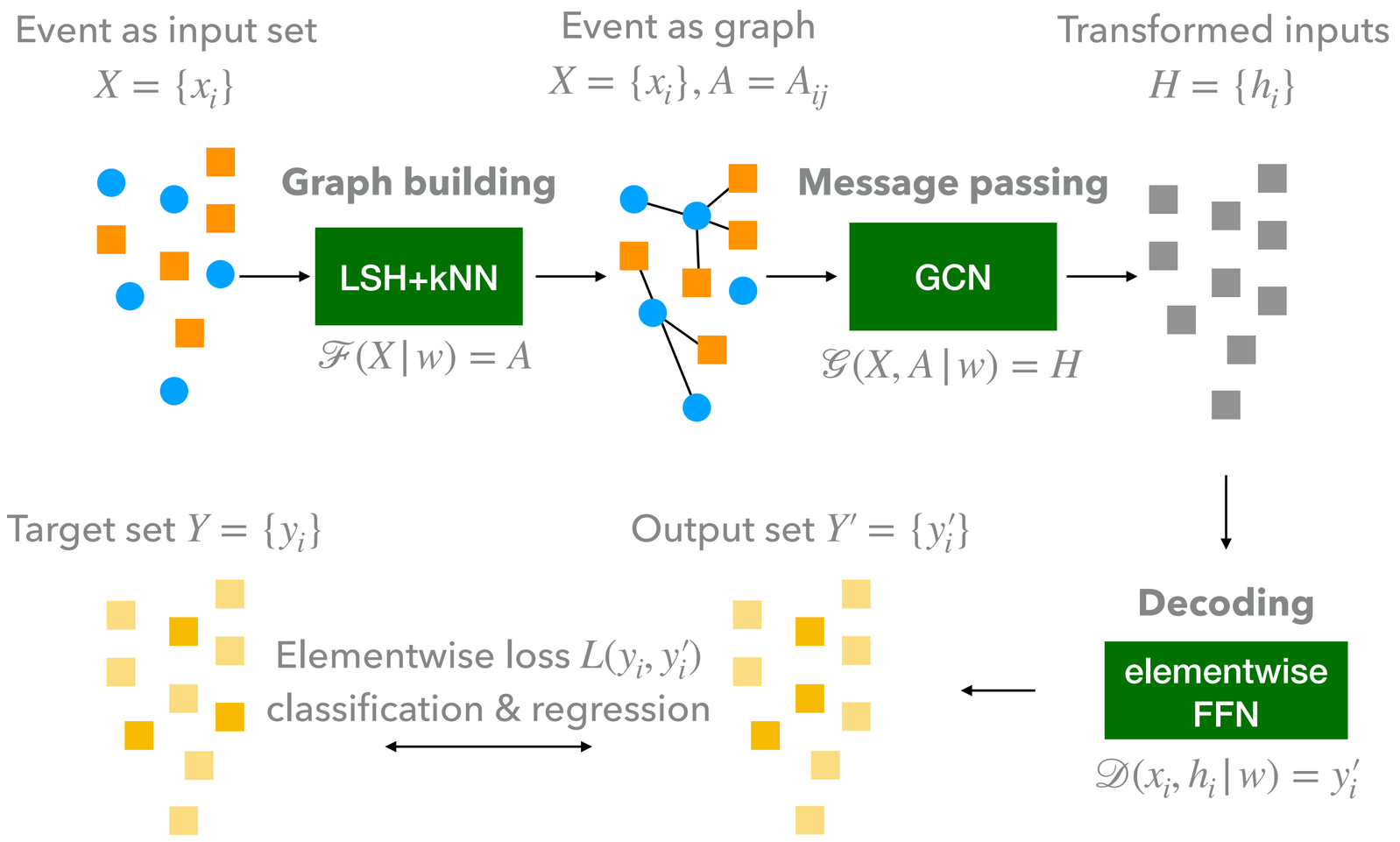


$h_i \in \mathbb{R}^N, N = 256$

Trainable neural networks: $\mathscr{F}, \mathscr{G}, \mathscr{D}$

GRAPH NEURAL NETWORK APPROACH

- Convert input set to a locally, **sparsely connected** graph
- Message-passing NN to transform features
- Decode transformed inputs elementwise
- (During training) Compare to target set, optimize weights



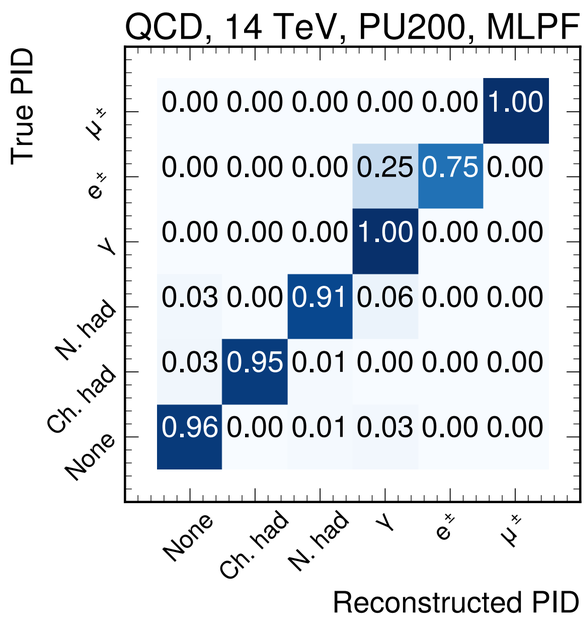
$h_i \in \mathbb{R}^N, N = 256$

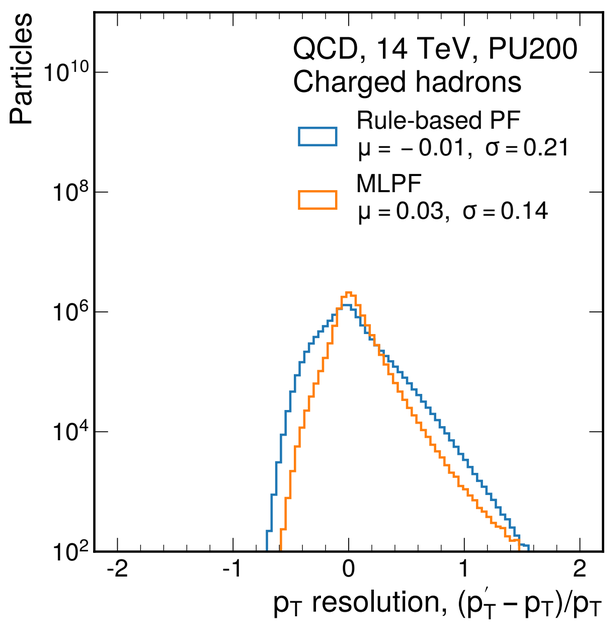
Trainable neural networks: $\mathcal{F}, \mathcal{G}, \mathcal{D}$

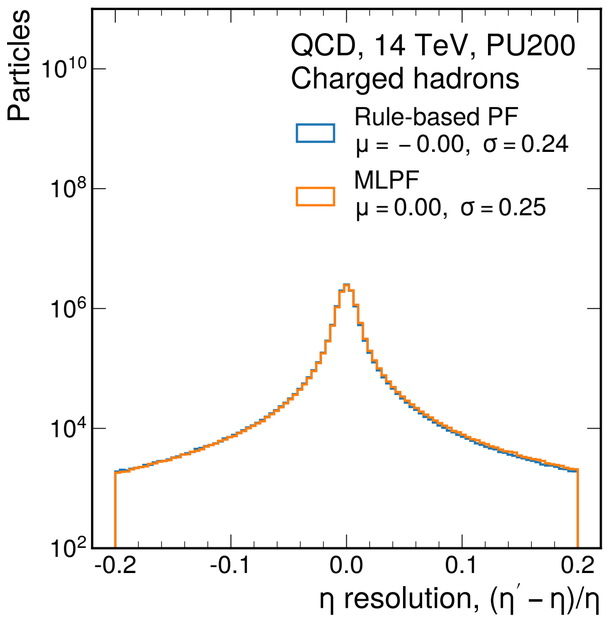
CLASSIFICATION/REGRESSION PERFORMANCE

- Classification performance: MLPF same or better than rule-based PF algorithm

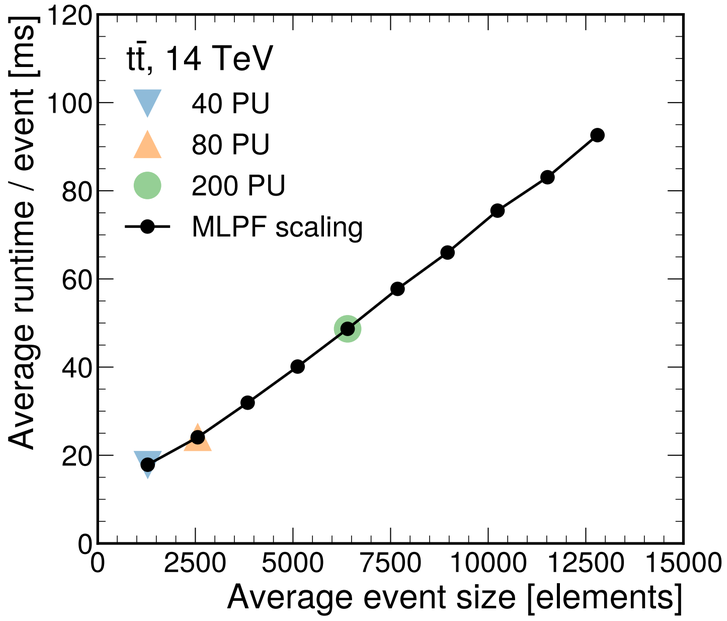
- Regression performance: MLPF same or better than rule-based PF algorithm





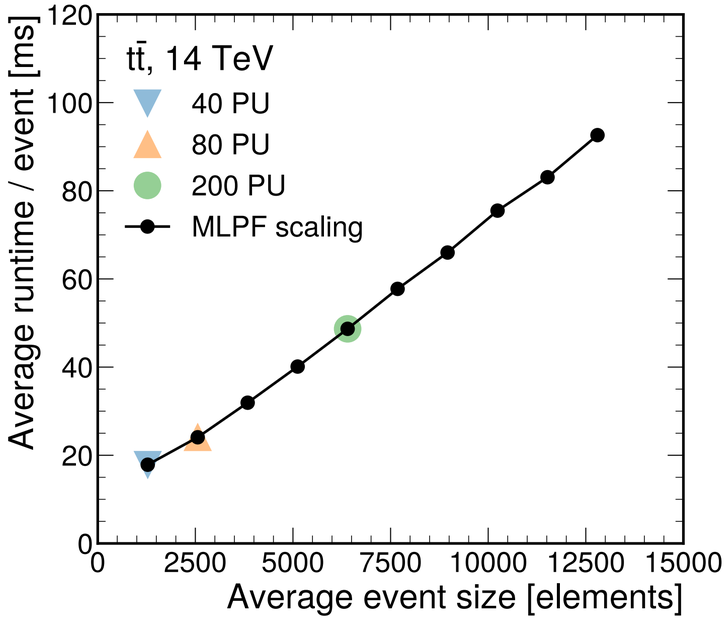


INFERENCE TIME



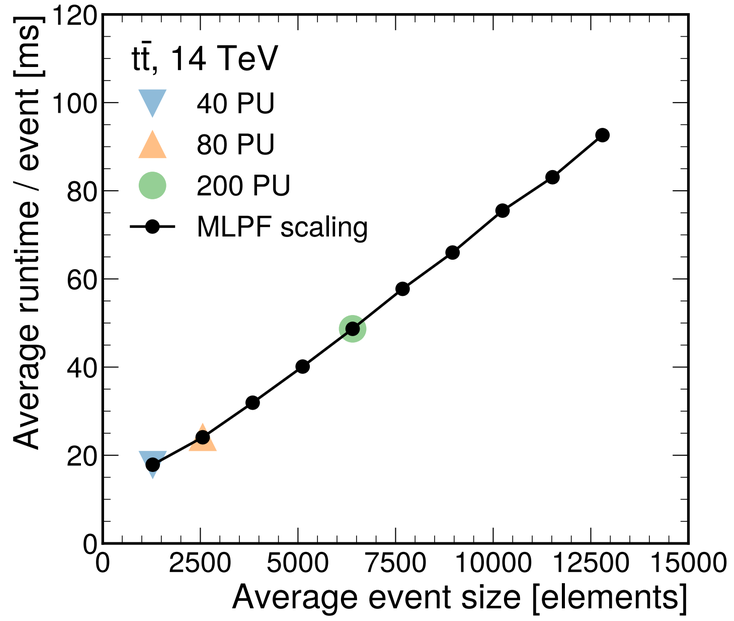
INFERENCE TIME

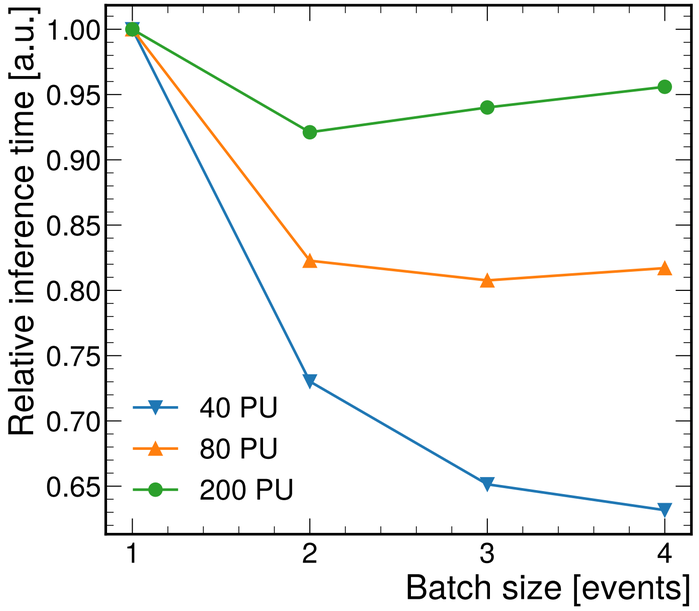
- Model inference time scales linearly with increasing pileup/event size!



INFERENCE TIME

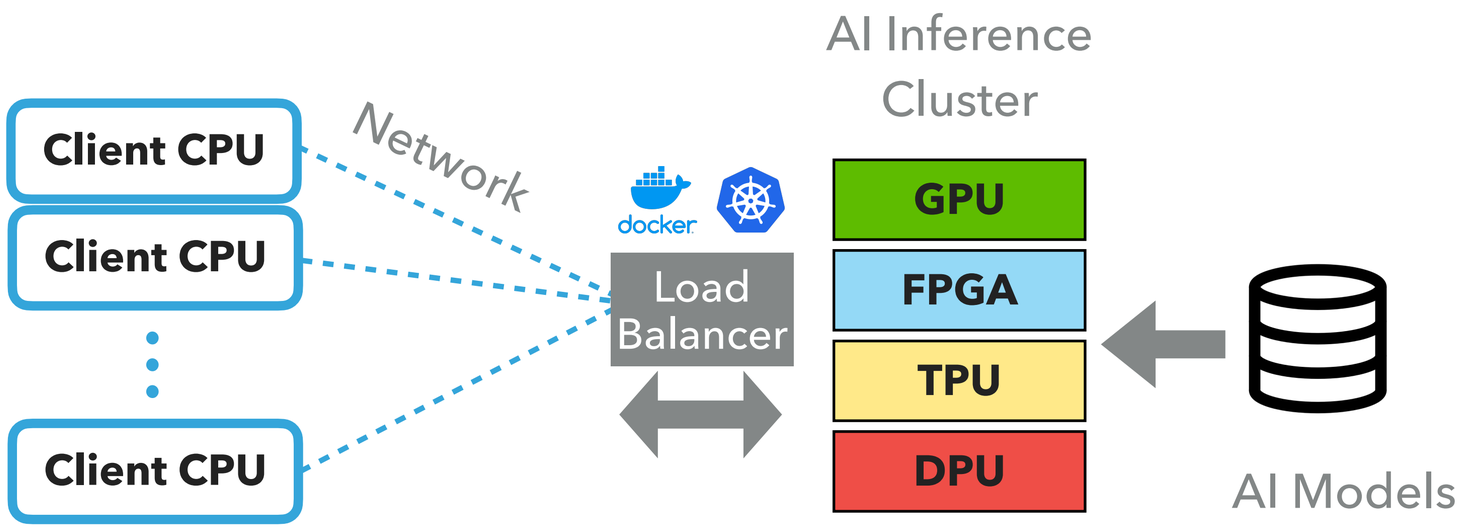
- Model inference time scales linearly with increasing pileup/event size!
- Increasing batch size also helps

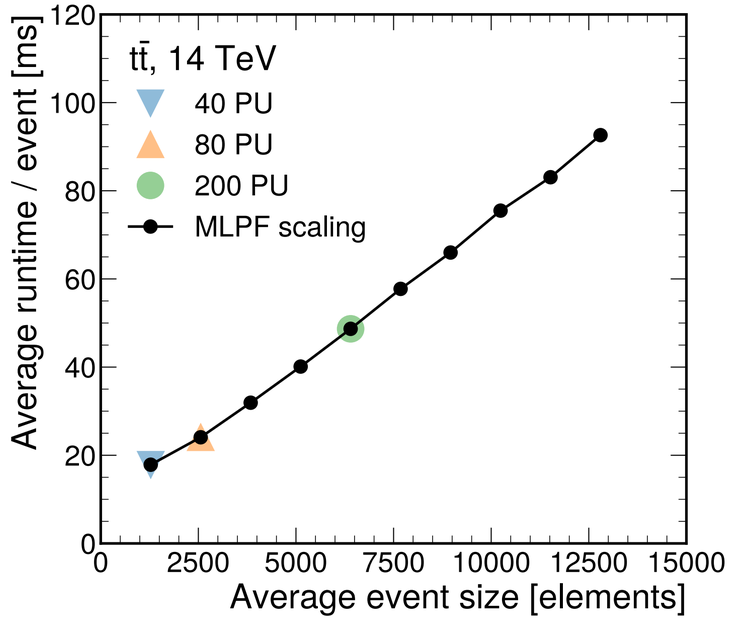


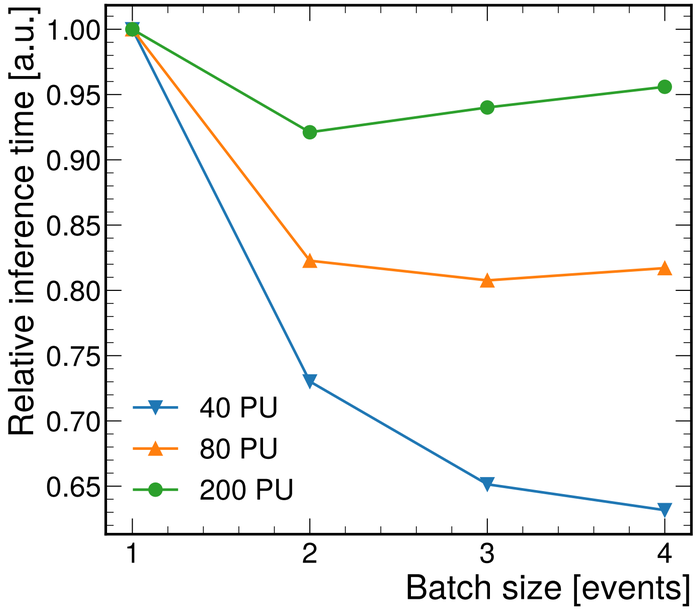


INFERENCE TIME

- Model inference time scales linearly with increasing pileup/event size!
- Increasing batch size also helps
- AI-dedicated hardware can accelerate this

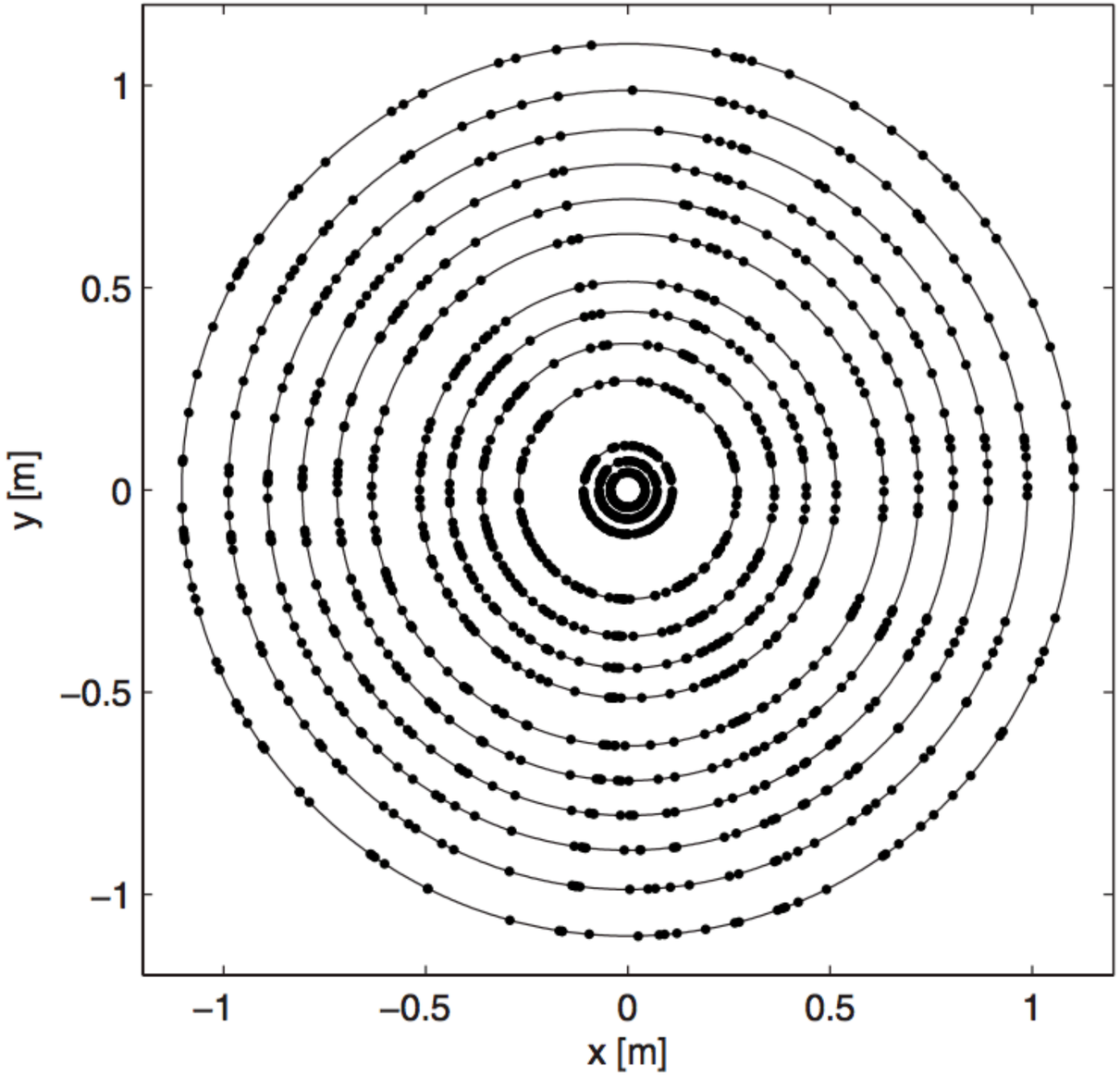






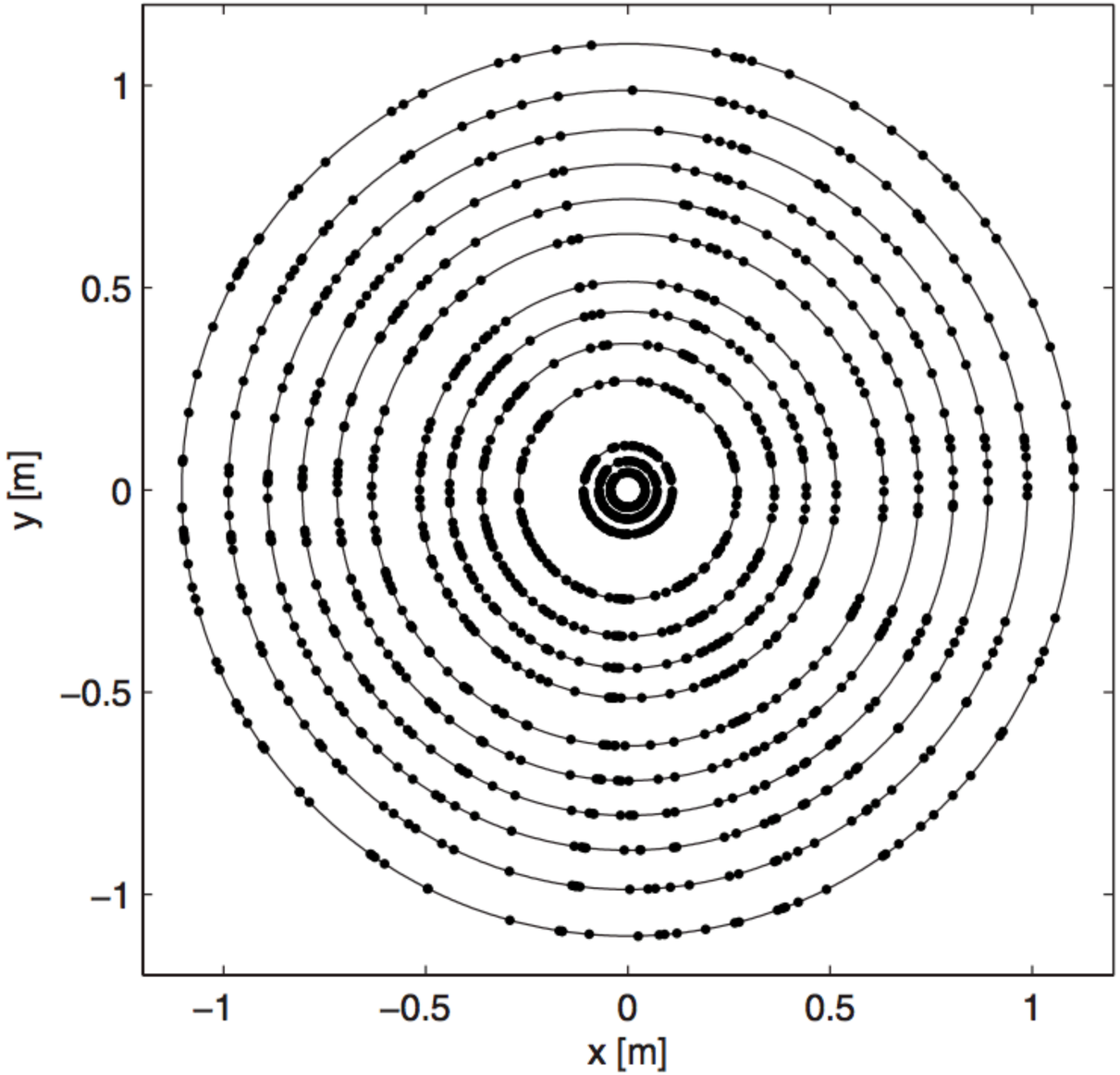
TRACKING

TRACKING (CONNECTING THE DOTS)



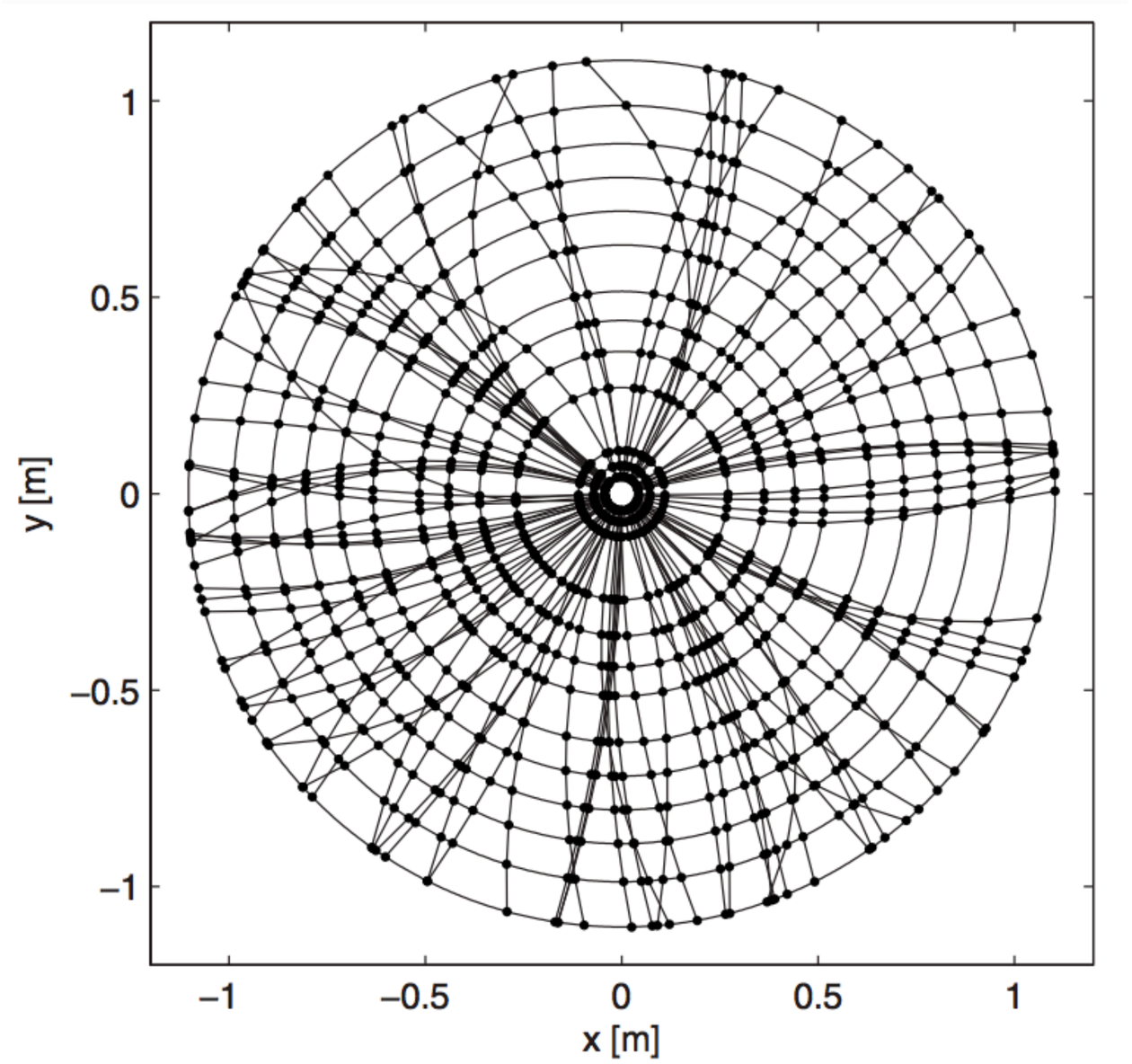
TRACKING (CONNECTING THE DOTS)

- Particle tracking is a classic reconstruction task



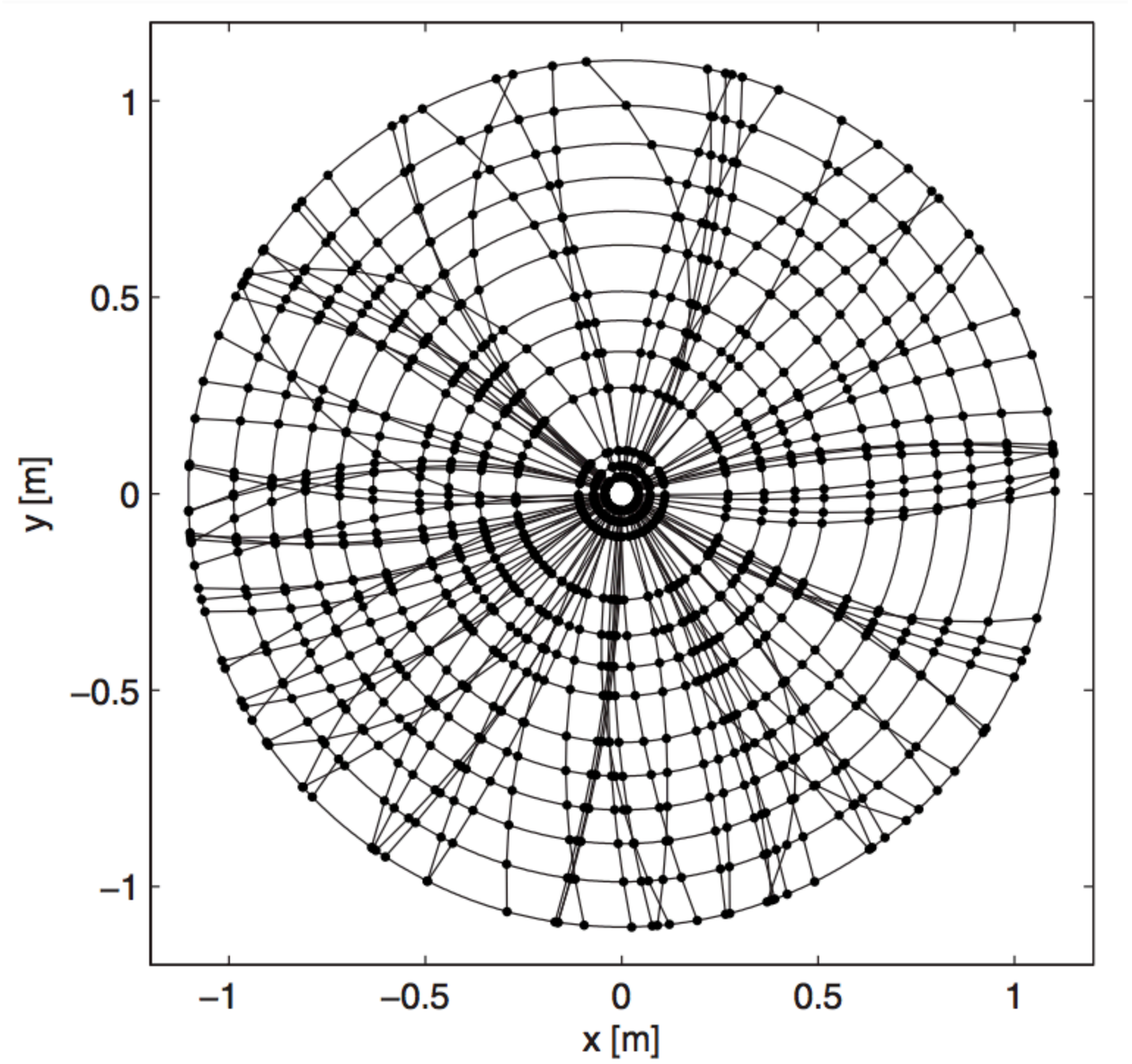
TRACKING (CONNECTING THE DOTS)

- Particle tracking is a classic reconstruction task
- From a set of hits sampled sparsely in 3D, reconstruct the helical trajectories of particles



TRACKING (CONNECTING THE DOTS)

- Particle tracking is a classic reconstruction task
- From a set of hits sampled sparsely in 3D, reconstruct the helical trajectories of particles
- Traditional algorithms scale badly with the number of hits
- GNNs may be able to do better [arXiv:1810.06111, arXiv:2003.11603, arXiv:2007.00149]

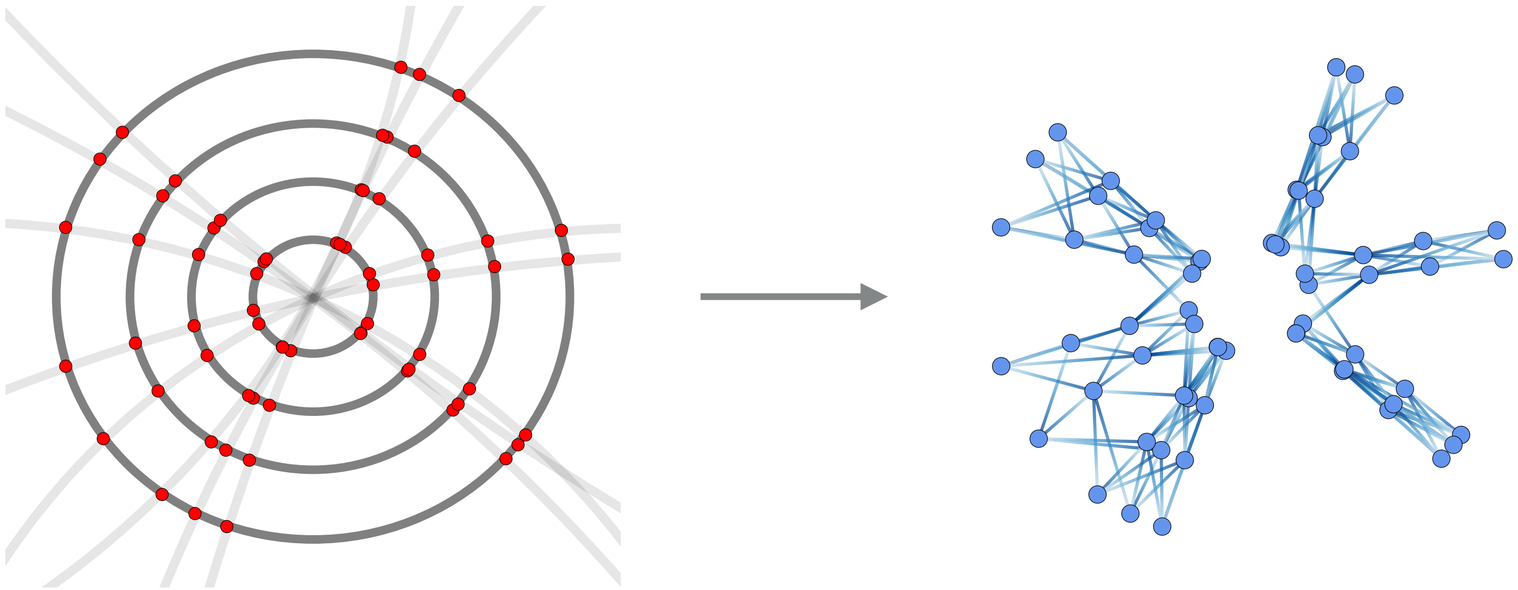


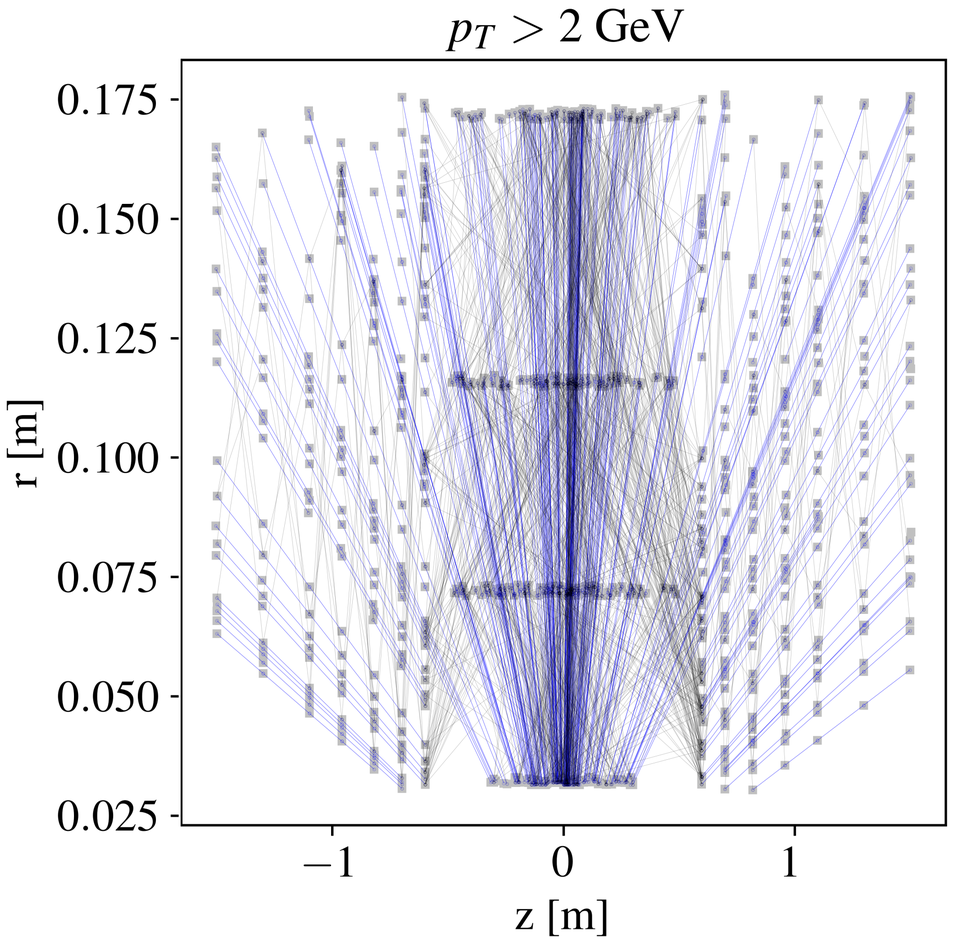
GRAPH ENCODING

- Hits in tracking detector → nodes
- Plausible track segments → edges

Kaggle TrackML dataset

Good tracks (black) defined with min. p_T in inner "pixel" detector



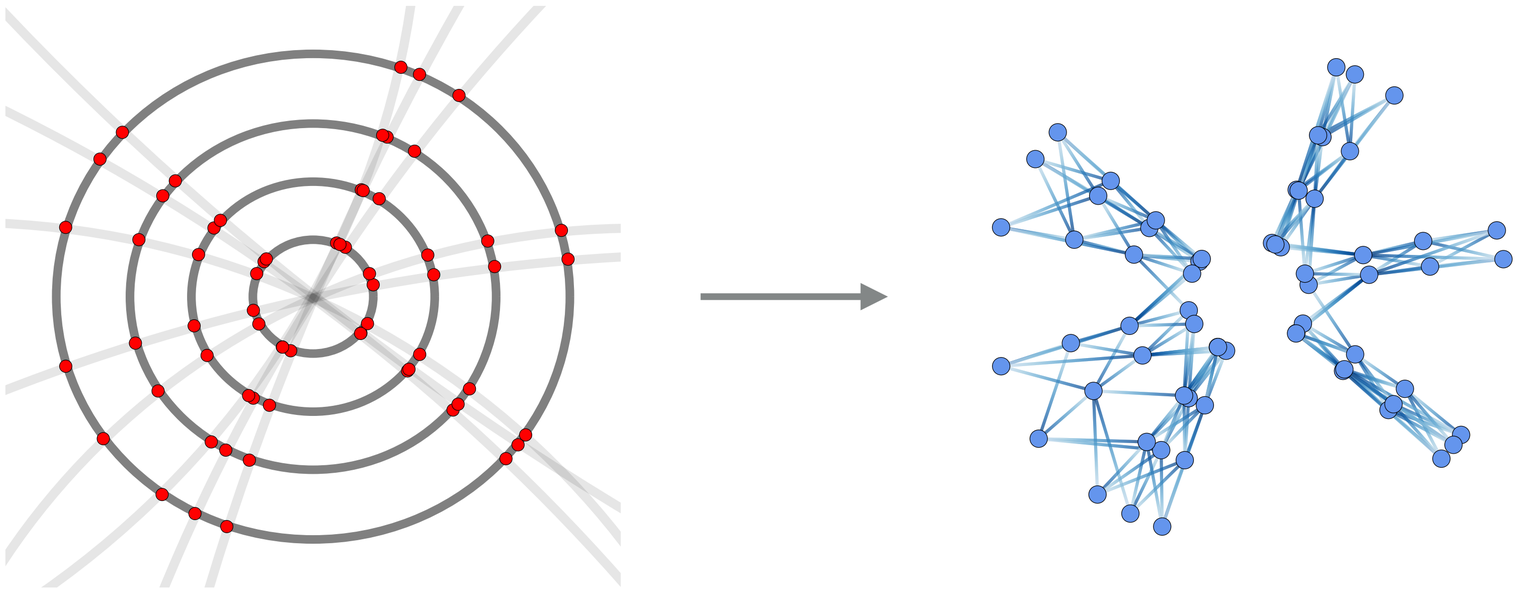


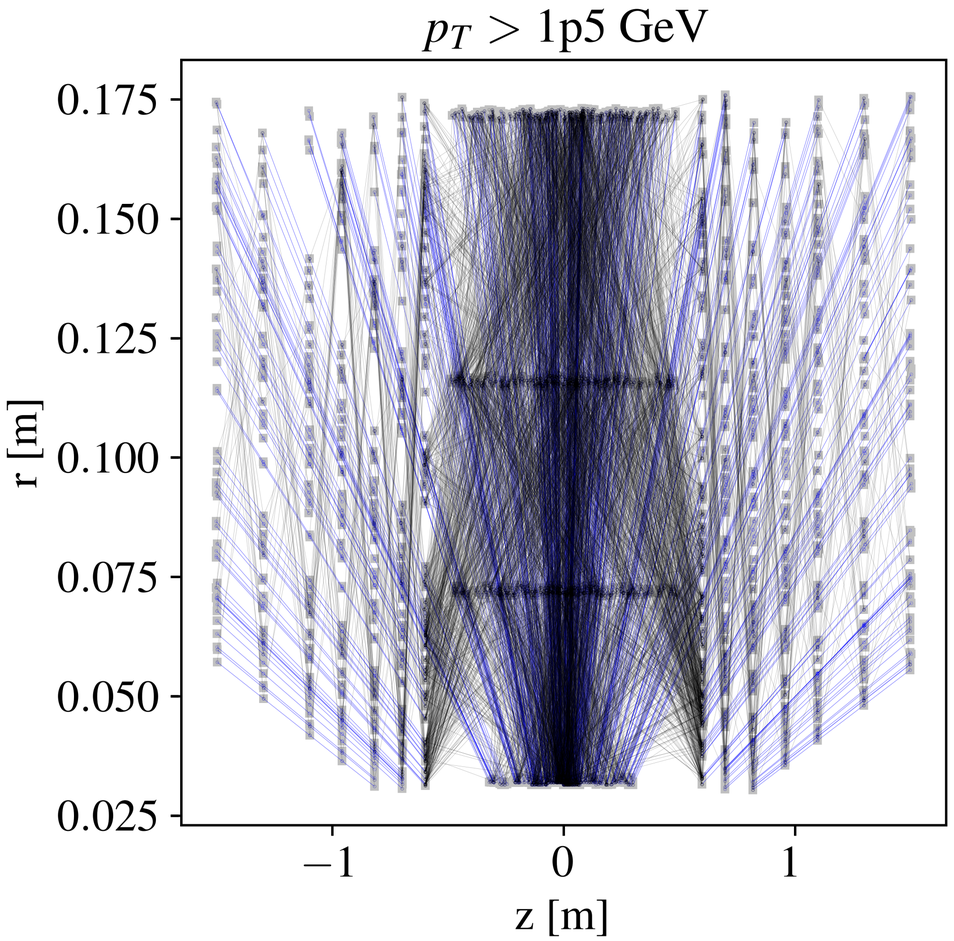
GRAPH ENCODING

- Hits in tracking detector → nodes
- Plausible track segments → edges

Kaggle TrackML dataset

Good tracks (black) defined with min. p_T in inner "pixel" detector



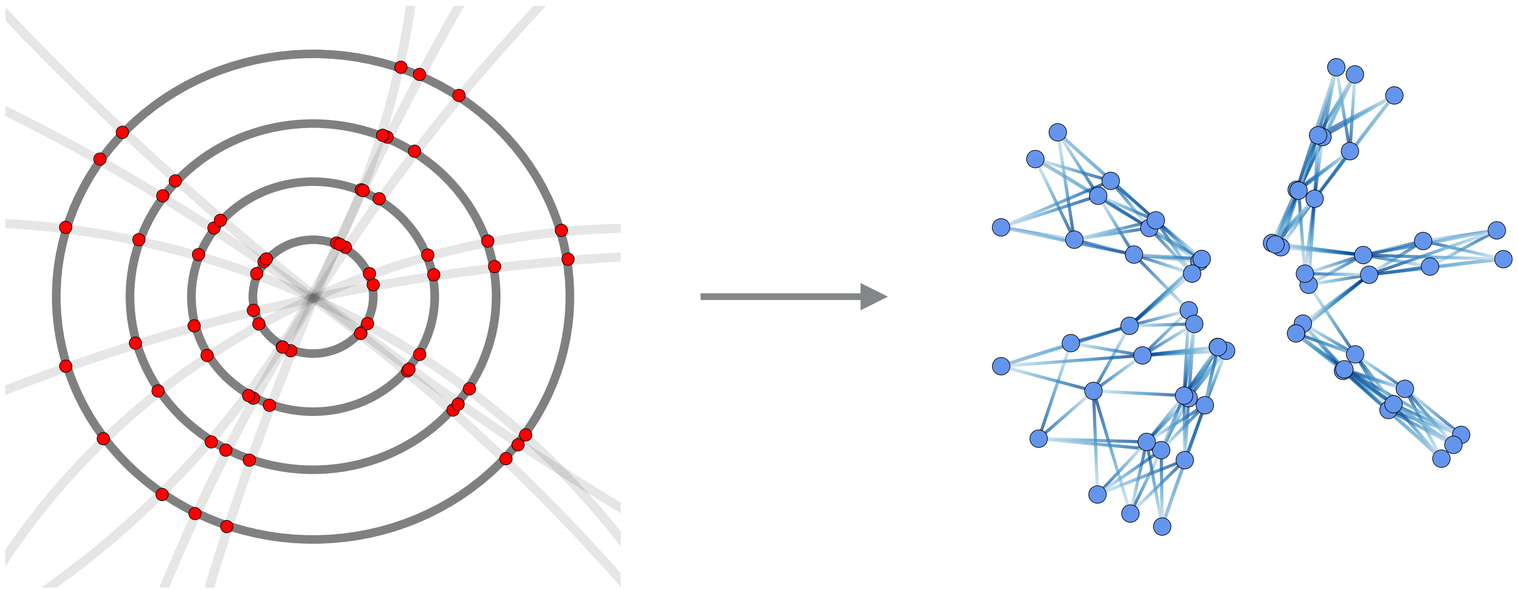


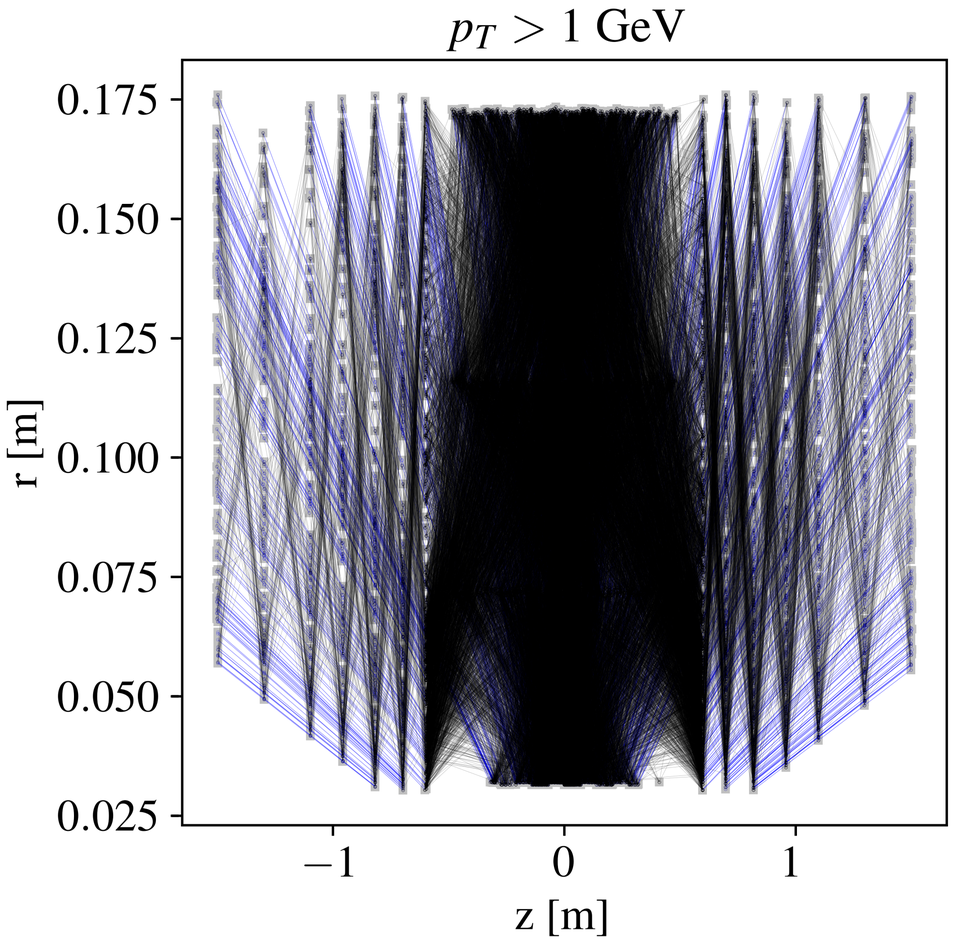
GRAPH ENCODING

- Hits in tracking detector → nodes
- Plausible track segments → edges

Kaggle TrackML dataset

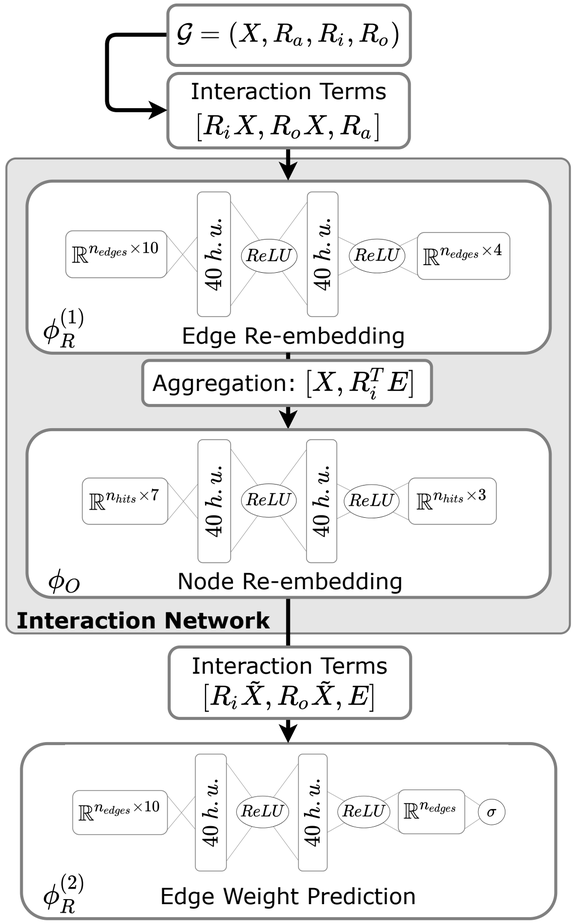
Good tracks (black) defined with min. p_T in inner "pixel" detector





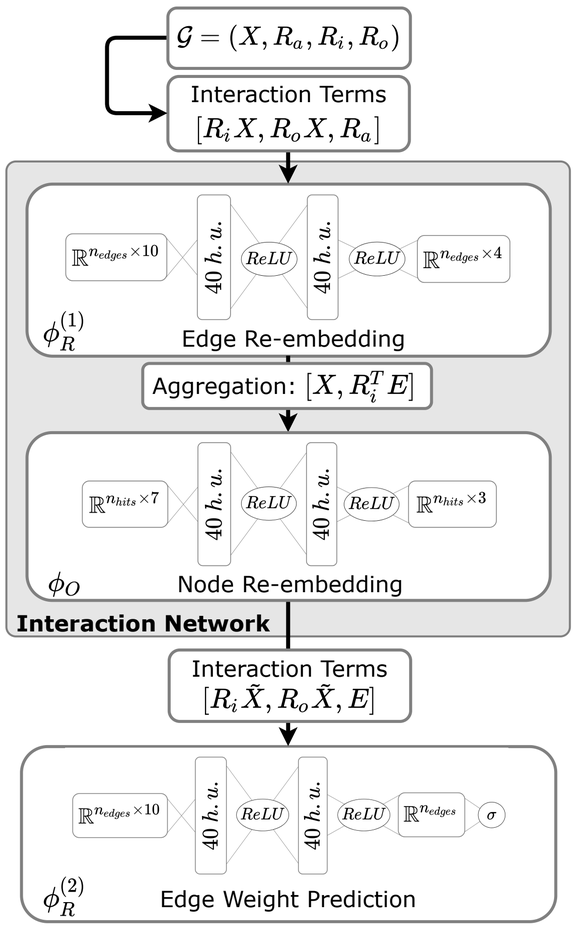
GENERALIZATION POWER OF GNNS

arXiv:2103.16701



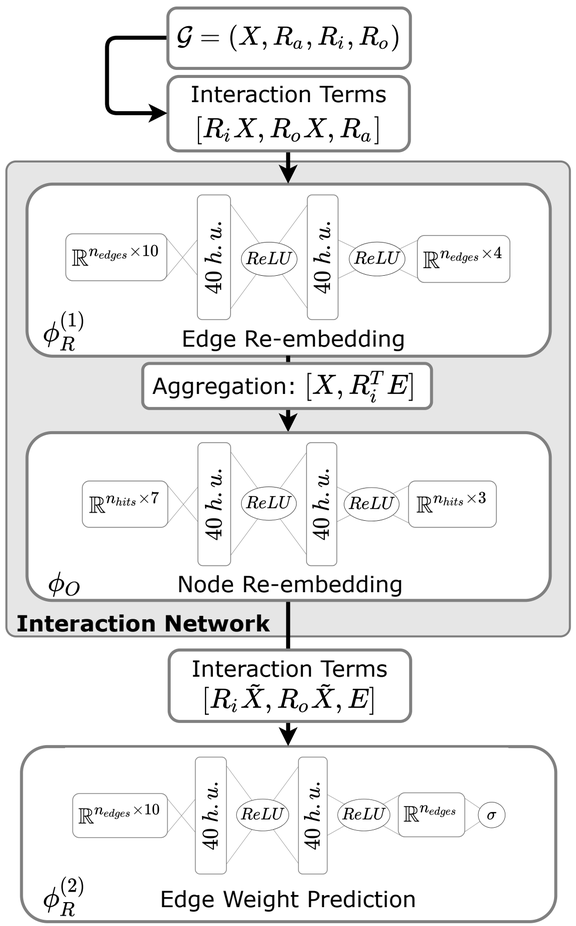
GENERALIZATION POWER OF GNNS

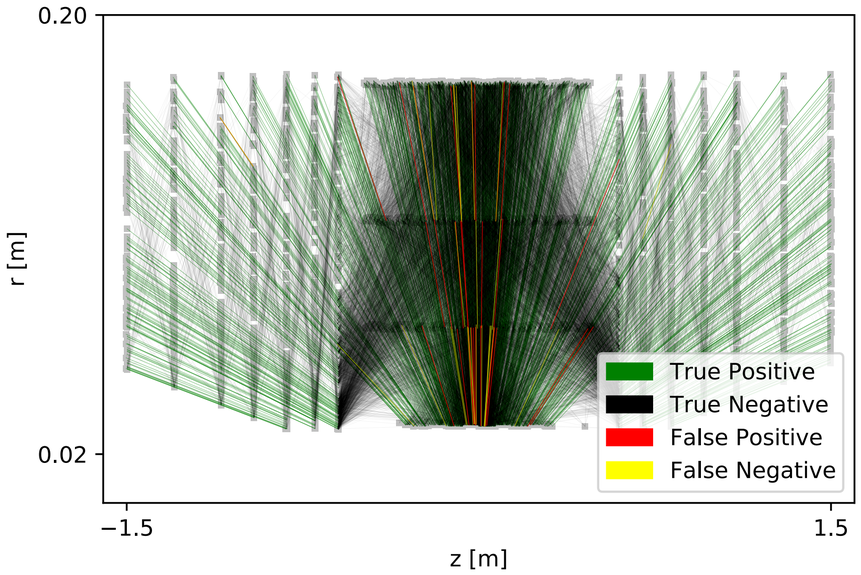
- Model is fairly simple "interaction network": 4 edge features and 3 node features are updated by NNs before final prediction of the link probability



GENERALIZATION POWER OF GNNS

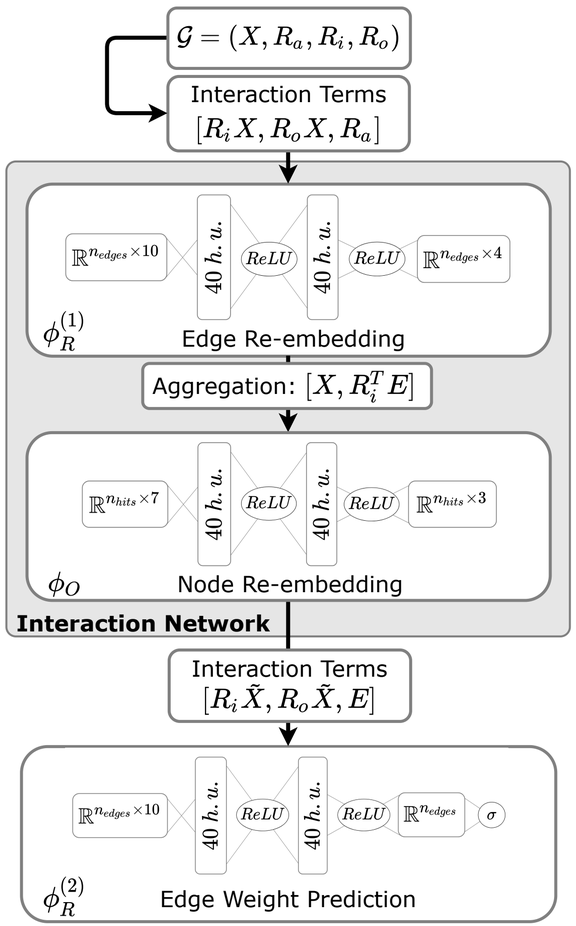
- Model is fairly simple "interaction network": 4 edge features and 3 node features are updated by NNs before final prediction of the link probability
- Excellent edge classification performance

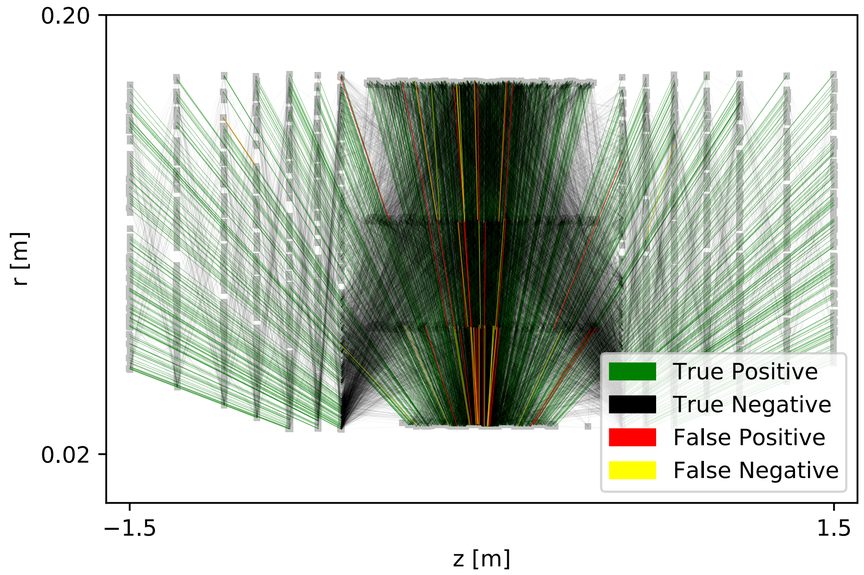




GENERALIZATION POWER OF GNNS

- Model is fairly simple "interaction network": 4 edge features and 3 node features are updated by NNs before final prediction of the link probability
- Excellent edge classification performance
- Model can generalize to more/less complex graphs (varying p_T cut used when constructing graphs)





Edge-Classification Accuracy

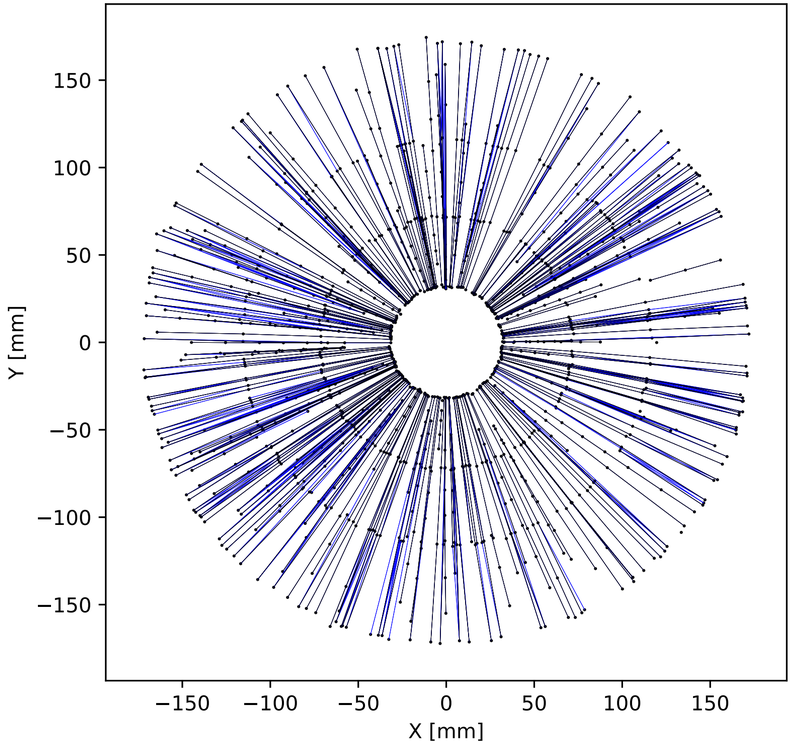
Training p_T^{min} [GeV] \ Testing p_T^{min} [GeV]	2	1.5	1	0.75	0.6
2	0.9965(29)	0.9963(17)	0.9906(12)	0.9802(32)	0.9768(46)
1.5	0.9962(32)	0.9975(13)	0.9934(10)	0.9868(10)	0.9799(18)
1	0.9945(40)	0.9968(16)	0.9973(06)	0.9931(07)	0.9842(08)
0.75	0.9918(61)	0.9952(23)	0.9965(09)	0.9960(05)	0.9932(05)
0.6	0.9881(79)	0.9926(30)	0.9947(10)	0.9953(05)	0.9943(05)

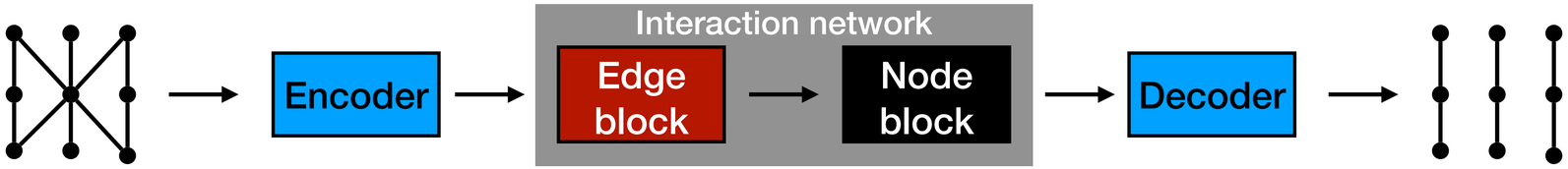
BEYOND

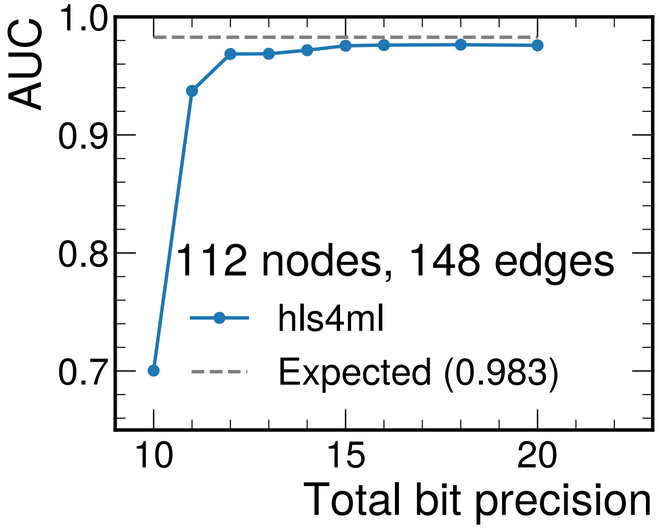
UCSD

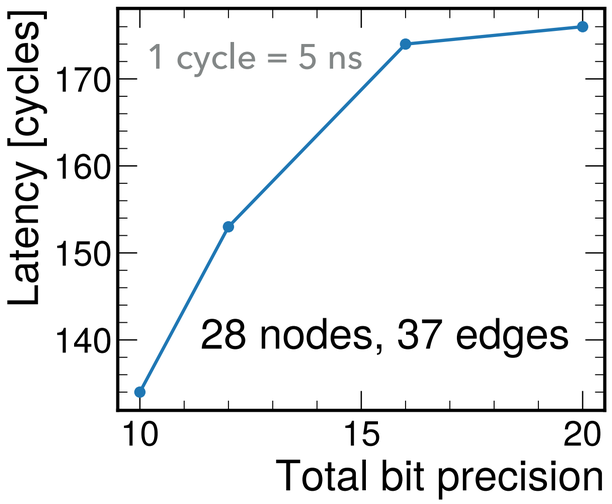
GRAPH NEURAL NETWORKS ON FPGAS FOR PARTICLE TRACKING

arXiv:2012.01563





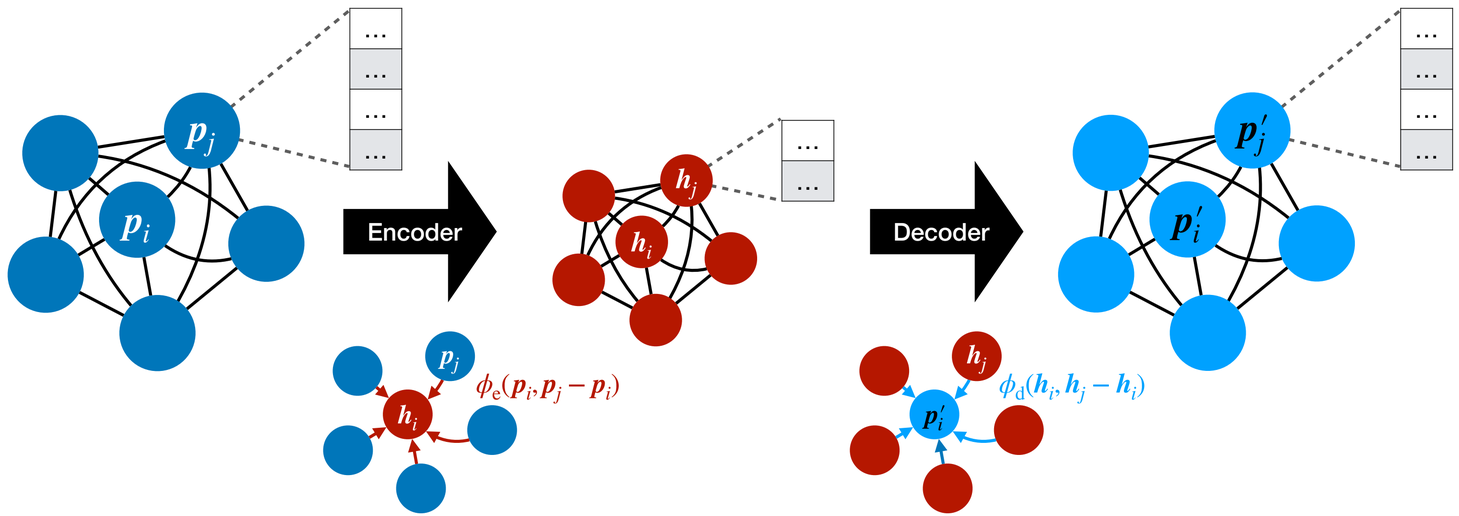


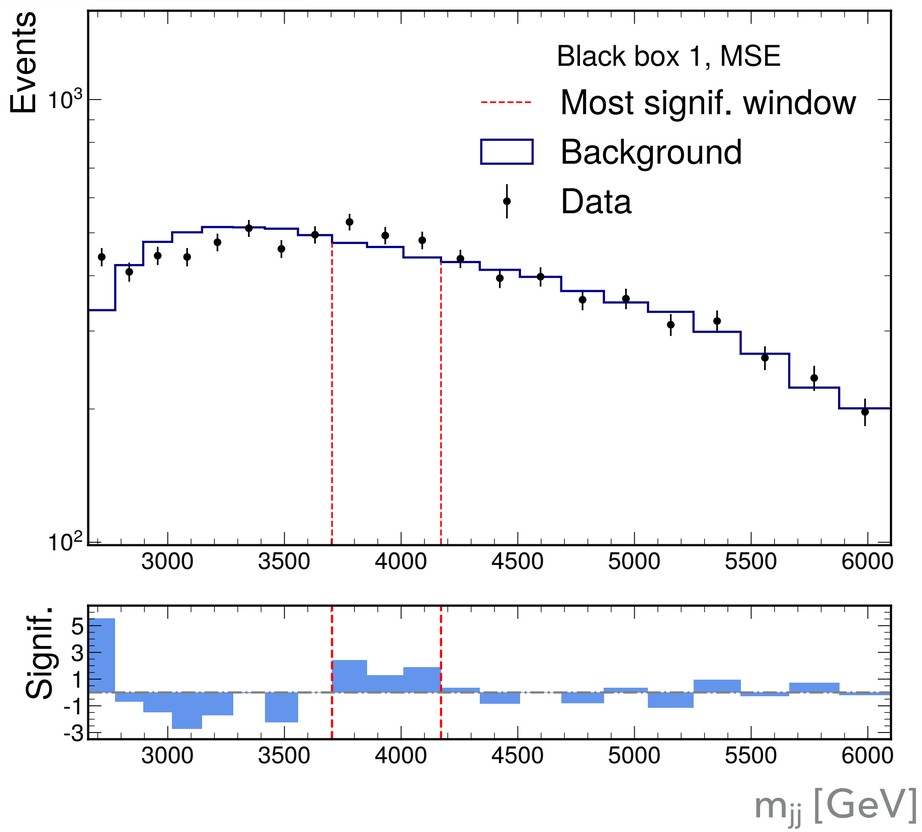


- Implemented a GNN on an FPGA for particle tracking on small sectors
- Found it can complete a tracking task in < 1 microsecond

PARTICLE GRAPH AUTOENCODER FOR ANOMALY DETECTION

arXiv:2101.08320

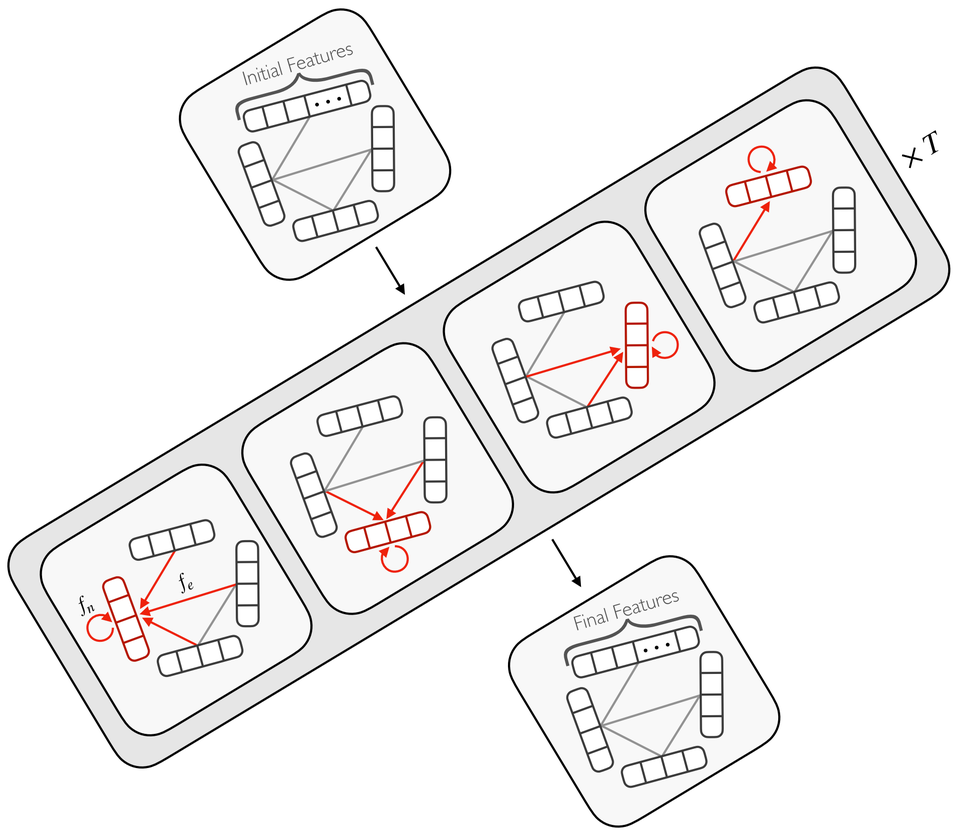


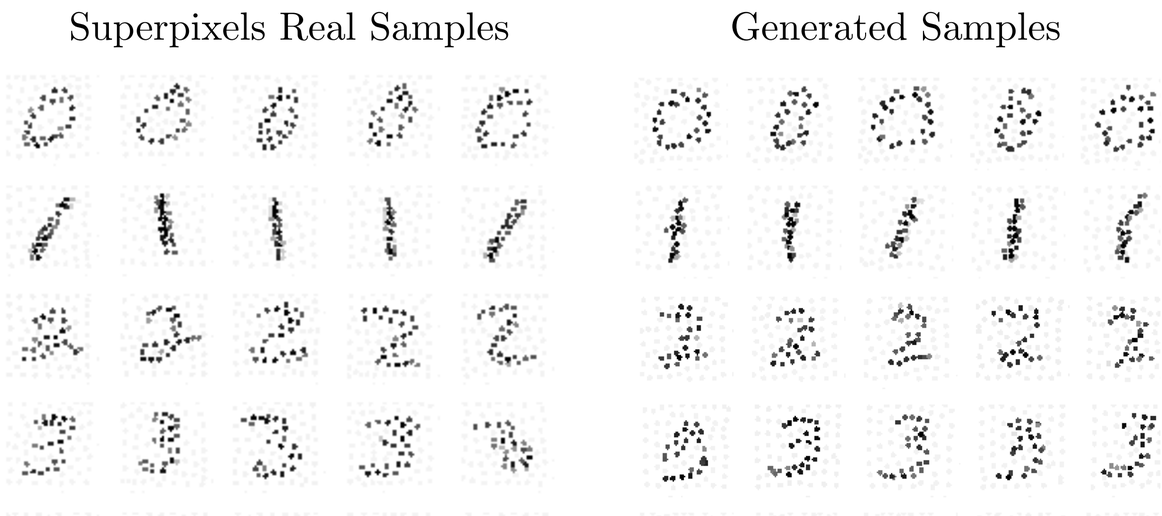


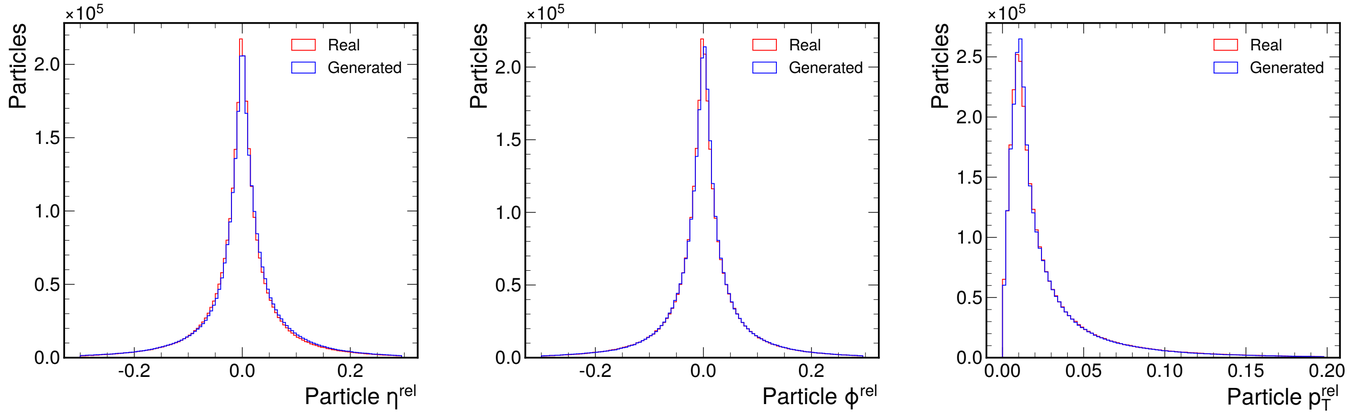
- Autoencoders compress data to a smaller representation and reconstruct it
  - Apply it to particles (E, ***p***): train autoencoder on known SM data
- Found to be effective on a mock dataset with a signal injected at 3.8 TeV

GRAPH GENERATIVE ADVERSARIAL NETWORK

arXiv:2012.00173







- Graph-based generative adversarial network can be used to generate MNIST handwritten digit graphs and particle jet graphs

SUMMARY AND OUTLOOK

SUMMARY AND OUTLOOK

- GNNs are NNs that operate on graph-structured data
  - Well suited to HEP data due to its sparsity, irregular geometry, and intercorrelations

SUMMARY AND OUTLOOK

- GNNs are NNs that operate on graph-structured data
  - Well suited to HEP data due to its sparsity, irregular geometry, and intercorrelations
- GNNs are being developed and used for many different tasks in HEP from particle tracking to event classification

SUMMARY AND OUTLOOK

- GNNs are NNs that operate on graph-structured data
  - Well suited to HEP data due to its sparsity, irregular geometry, and intercorrelations
- GNNs are being developed and used for many different tasks in HEP from particle tracking to event classification
- Many new avenues for GNNs are also being explored:
  - End-to-end or “single-stage” inference
  - Unsupervised learning (like compression and anomaly detection)
  - Generative methods for sparse data
  - Accelerated inference in GPUs and FPGAs

REFERENCES

- GNNs:
  - Graph Neural Network Model Survey [doi:10.1109/TNN.2008.2005605]
  - Interaction Network [arXiv:1612.0222]
  - Geometric deep learning review [arXiv:1611.08097]
  - Message Passing NN [arXiv:1704.01212]
  - Dynamic Graph CNN [arXiv:1801.07829]
  - Graph Networks [arXiv:1806.01261]
  - Deep Sets [arXiv:1703.06114]
- Tools:
  - PyTorch Geometric [arXiv:1903.0242]
  - graph_nets [deepmind/graph_nets]
  - DGL [dgl.ai]
  - Spektral [danielegrattarola/spektral]
  - jraph [jraph]
- Physics:
  - Neural message passing for jets [nips_dlps_2017_29]
  - Energy Flow Networks [arXiv:1810.05165]
  - HEP.TrkX GNN [arXiv:1810.06111]
  - GarNet and GravNet [arXiv:1902.07987]
  - ParticleNet [arXiv:1902.08570]
  - JEDI-Net [arXiv:1908.05318]
  - Interaction networks for H(bb) decays [arXiv:1909.12285]
  - ABCNet [arXiv:2001.05311]
  - Object condensation [arXiv:2002.03605]
  - Exa.TrkX GNN [arXiv:2003.11603]
  - Two-point correlation GNN [arXiv:2003.11787]
  - MPNN for HH [arXiv:2005.11086]
  - Graph net for dark showers [arXiv:2006.08639]
  - GNNs in Particle Physics Review [arXiv:2007.13681]
  - Exa.TrkX track seeding and labelling [arXiv:2007.00149]
  - GarNet on FPGAs [arXiv:2008.03601]
  - Jet clustering [arXiv:2008.06064]

JAVIER DUARTE
UNIVERSITY OF MINNESOTA HEP SEMINAR
APRIL 28, 2021

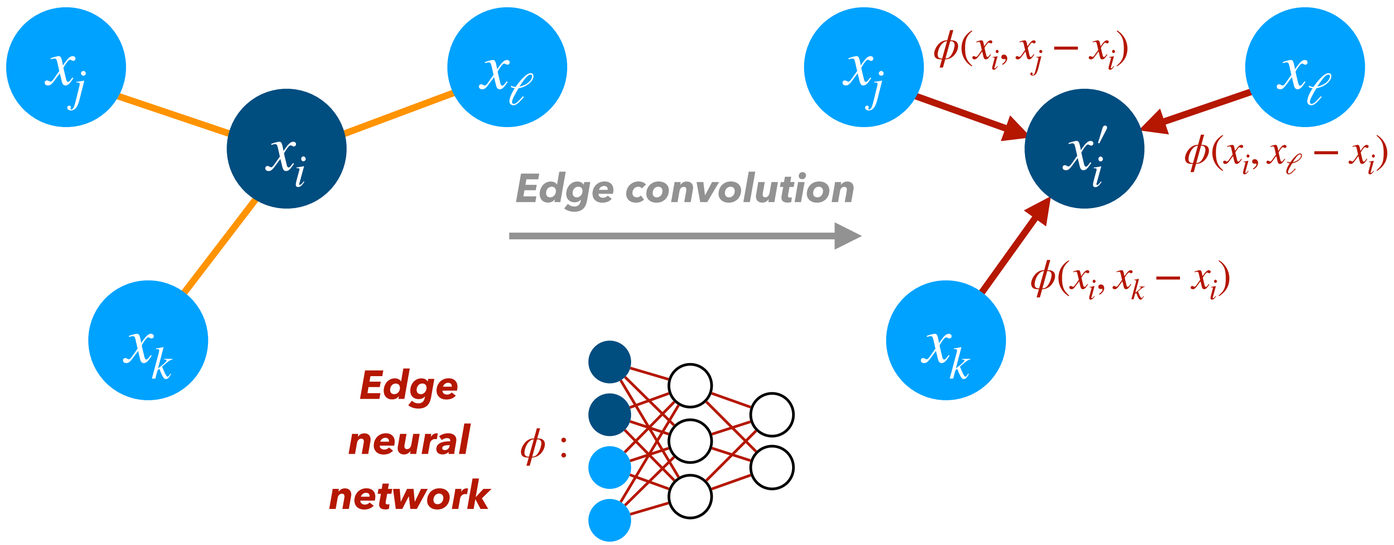
BACKUP

UCSD

CONNECTIONS TO OTHER GNNS

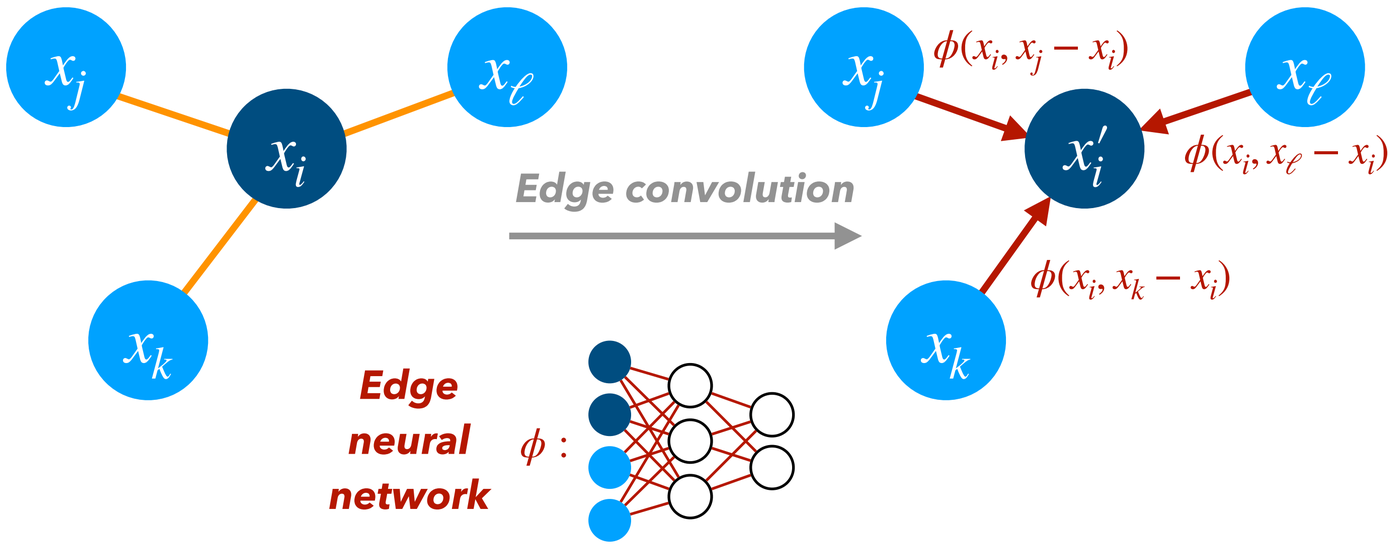
CONNECTIONS TO OTHER GNNS

- Edge convolution from DGCNN [arXiv:1801.07829] is a variant of the edge block step (basis of ParticleNet [arXiv:1902.08570])

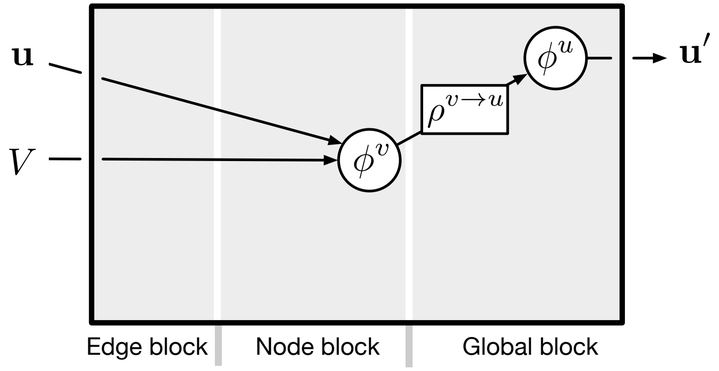


CONNECTIONS TO OTHER GNNS

- Edge convolution from DGCNN [arXiv:1801.07829] is a variant of the edge block step (basis of ParticleNet [arXiv:1902.08570])

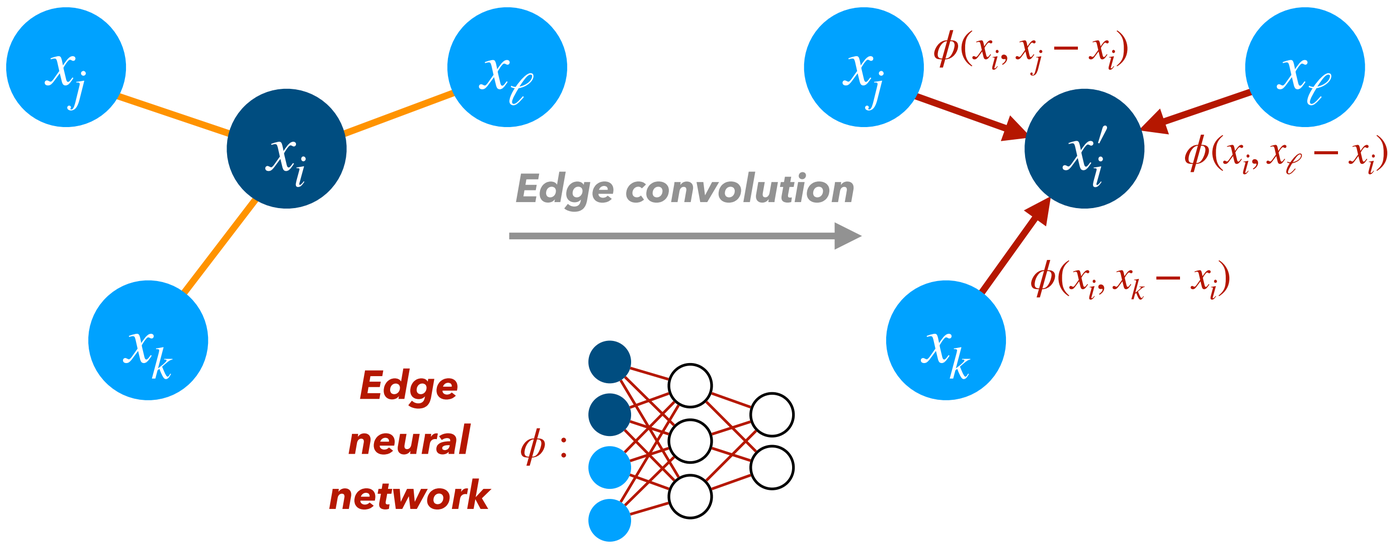


- Deep Sets [arXiv:1703.06114] does not consider edge features (basis of Energy Flow Network [arXiv:1810.05165])

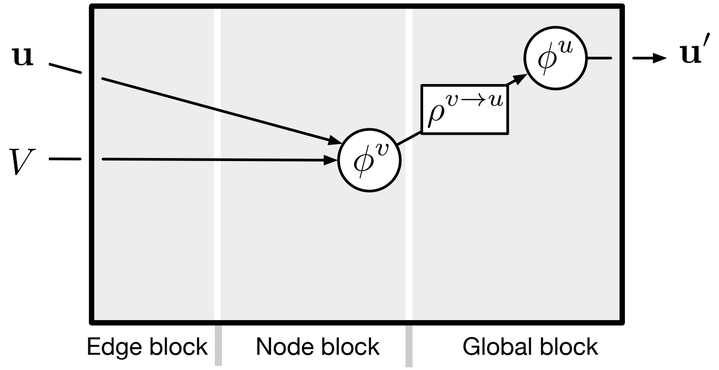


CONNECTIONS TO OTHER GNNS

- Edge convolution from DGCNN [arXiv:1801.07829] is a variant of the edge block step (basis of ParticleNet [arXiv:1902.08570])



- Deep Sets [arXiv:1703.06114] does not consider edge features (basis of Energy Flow Network [arXiv:1810.05165])



- Interaction network [arXiv:1612.00222] ignores global features (basis of JEDI-Net, H(bb) tagger [arXiv:1908.05318, arXiv:1909.12285], Exa.TrkX GNN [arXiv:2003.11603])